Arquitectura IoT Centrada en Pasarelas de Borde

Implementación de Protocolos basados en 6LowPAN para Smart Energy

Juan Sebastian Giraldo Duque

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

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

Arquitectura IoT Centrada en Pasarelas de Borde

Implementación de Protocolos basados en 6LowPAN para Smart Energy

Juan Sebastian Giraldo Duque

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

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

Codirector(a):
Prof. Dr. Co director
Indicar si es Profesor Titular/Asociado - Departamento
Facultad
Universidad Nacional de Colombia

Línea de investigación:
Línea
Grupo de investigación:
Grupo A (Sigla Grupo Investigación 01)
Grupo B (Sigla Grupo Investigación 02)

Universidad Nacional de Colombia
Facultad
Departamento
Año entrega

Cita 01.

Autor
Fuente

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

Albert Einstein

Declaración

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

Sede de la Universidad., Fecha entrega

Juan Sebastian Giraldo Duque

Agradecimientos

Listado de símbolos y abreviaturas

Resumen

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

Texto del resumen.

Palabras clave: Use palabras clave que estén en Theasaurus

Abstract

Nombre del trabajo o tesis en inglés

Abstract text.

Keywords: Use keywords available in Theasaurus

Zusammenfassung

Nombre del trabajo o tesis en un tercer idioma

Zusammenfassung Texte.

Schlüsselwörter:

Lista de figuras

Lista de tablas

6-1 Seguridad end-to-end por capa 14
6-2 Costos de implementación (300 medidores). 15
6-3 Comparación con gateways comerciales 16

Contenido

Agradecimientos II

Listado de símbolos y abreviaturas III

Resumen IV

Abstract V

Zusammenfassung VI

Lista de figuras VII

Lista de tablas VIII

Contenido IX

1 Planteamiento del Problema 1

2 Hipótesis 2

3 Objetivos 3

4 Marco teórico 4

5 Gateway de Telemetría para Smart Energy 5

5.1 Introducción . . . 5

5.1.1 Funciones Principales . . . 5

5.2 Conformidad con Estándares . . . 5

5.2.1 IEEE 2030.5-2023 (Smart Energy Profile 2.0) . . . 5

5.2.2 ISO/IEC 30141:2024 (IoT Reference Architecture) . . . 5

5.3 Plataforma de Hardware . . . 6

5.3.1 Raspberry Pi 4 Model B . . . 6

5.3.2 Interfaces de Conectividad . . . 6

5.4 Sistema Operativo y Software Base . . . 6

5.4.1 OpenWRT 23.05 como Base . . . 6

5.4.2 Servicios Containerizados (Docker) . . . 6

5.5 Conectividad y Protocolos . . . 7

5.5.1 Thread 802.15.4 (Acceso a Nodos) . . . 7
5.5.2 HaLow 802.11ah (Backhaul) . . . 7
5.5.3 Protocolos de Aplicación . . . 7
5.6 Arquitectura de Datos . . . 8
5.6.1 TimescaleDB (Series Temporales) . . . 8
5.6.2 Apache Kafka (Message Bus) . . . 8
5.7 Seguridad . . . 8
5.7.1 Public Key Infrastructure (PKI) . . . 8
5.7.2 TLS/mTLS . . . 9
5.7.3 Firewall y Zonas . . . 9
5.8 Resiliencia y Alta Disponibilidad . . . 9
5.8.1 Almacenamiento Persistente (SSD NVMe) . . . 9
5.8.2 Failover WAN (Ethernet LTE) . . . 9
5.8.3 Buffering Local . . . 9
5.9 Edge Computing y AI Local . . . 9
5.9.1 Preprocesamiento de Datos . . . 9
5.9.2 Ollama LLM Local (Opcional) . . . 10
5.10 Monitoreo y Gestión . . . 10
5.10.1 Prometheus + Grafana . . . 10
5.10.2 Logging Centralizado . . . 10
5.11 Pruebas y Validación . . . 10
5.11.1 Pruebas de Conectividad . . . 10
5.11.2 Pruebas de Latencia . . . 10
5.11.3 Pruebas de Carga . . . 10
5.12 Comparación con Alternativas Comerciales . . . 11
5.13 Limitaciones y Consideraciones . . . 11
5.14 Conclusiones del Capítulo . . . 11

6 Arquitectura de Telemetría para Smart Energy 12
6.1 Introducción . . . 12
6.2 Modelo de Tres Capas IoT . . . 12
6.2.1 Nivel 1: Nodos IoT (Sensores/Actuadores) . . . 12
6.2.2 Nivel 2: Gateway Edge (Data Aggregator) . . . 12
6.2.3 Nivel 3: Plataforma Cloud (Analytics y Visualización) . . . 13
6.3 Flujo de Datos End-to-End . . . 13
6.3.1 Uplink (Telemetría) . . . 13
6.3.2 Downlink (Comandos) . . . 13
6.4 Topologías de Red . . . 14
6.4.1 Thread 802.15.4 (Acceso Local) . . . 14
6.4.2 HaLow 802.11ah (Backhaul) . . . 14
6.4.3 LTE Cat-M1/NB-IoT (Backup WAN) . . . 14
6.5 Seguridad Multicapa . . . 14
6.5.1 Public Key Infrastructure (PKI) . . . 15
6.6 Caso de Estudio: Despliegue Residencial . . . 15

6.6.1 Escenario . . . 15
6.6.2 Dimensionamiento de Red . . . 15
6.6.3 Resiliencia . . . 15
6.7 Análisis de Costos . . . 15
6.8 Métricas de Desempeño . . . 16
6.8.1 Latencia End-to-End . . . 16
6.8.2 Disponibilidad . . . 16
6.8.3 Throughput y Escalabilidad . . . 16
6.9 Comparación con Soluciones Comerciales . . . 16
6.10 Limitaciones y Trabajo Futuro . . . 17
6.10.1 Limitaciones Actuales . . . 17
6.10.2 Mejoras Propuestas . . . 17
6.11 Conclusiones del Capítulo . . . 17

7 Discusión de resultados **18**

8 Conclusiones **19**

9 Recomendaciones **20**

A Apéndice 1 **22**

B Instalación y Configuración del Gateway OpenWRT **23**
B.1 Sistema Operativo: OpenWRT 23.05 . . . 23
B.1.1 Especificaciones de la Versión . . . 23
B.1.2 Procedimiento de Instalación . . . 23
B.1.3 Instalación de Paquetes Esenciales . . . 25
B.2 Configuración de Almacenamiento NVMe . . . 25
B.2.1 Detección y Particionamiento del SSD . . . 25
B.2.2 Montaje Automático en `/mnt/ssd` . . . 26
B.2.3 Estructura de Directorios para Servicios . . . 26
B.2.4 Configuración de Docker para usar SSD . . . 27
B.3 Configuración de Periféricos de Conectividad . . . 28
B.3.1 Thread Border Router con nRF52840 Dongle . . . 28
B.3.2 HaLow 802.11ah via SPI (Morse Micro MM6108) . . . 30
B.3.3 LTE Modem Quectel BG95-M3 . . . 31
B.4 Instalación de Docker y Docker Compose . . . 33
B.4.1 Instalación de Paquetes Docker . . . 33
B.4.2 Configuración de Docker Daemon . . . 34
B.5 Verificación de Instalación Completa . . . 35
B.5.1 Checklist de Verificación . . . 35
B.5.2 Logs de Sistema para Debug . . . 36
B.6 Troubleshooting Común . . . 36
B.6.1 Problemas con NVMe SSD . . . 36
B.6.2 Problemas con Thread nRF52840 . . . 36

B.6.3 Problemas con HaLow SPI . . . 37
B.6.4 Problemas con LTE Quectel . . . 37
B.7 Resumen de Configuración . . . 38

C Archivos Docker Compose del Gateway 39
C.1 Estructura de Directorios Docker . . . 39
C.2 OpenThread Border Router (OTBR) . . . 40
C.2.1 Función del OTBR . . . 40
C.2.2 Docker Compose: OTBR . . . 40
C.2.3 Comandos de Gestión OTBR . . . 41
C.3 ThingsBoard Edge + PostgreSQL . . . 41
C.3.1 Función de ThingsBoard Edge . . . 41
C.3.2 Docker Compose: ThingsBoard Edge . . . 42
C.3.3 Archivo .env para Variables de Entorno . . . 43
C.3.4 Comandos de Gestión ThingsBoard Edge . . . 43
C.4 IEEE 2030.5 Server (SEP 2.0) . . . 44
C.4.1 Función del IEEE 2030.5 Server . . . 44
C.4.2 Docker Compose: IEEE 2030.5 Server . . . 44
C.4.3 Dockerfile para IEEE 2030.5 Server . . . 45
C.4.4 requirements.txt . . . 45
C.5 Apache Kafka + Zookeeper . . . 45
C.5.1 Función de Kafka . . . 45
C.5.2 Docker Compose: Kafka . . . 46
C.5.3 Comandos de Gestión Kafka . . . 47
C.6 Bridge Thread-ThingsBoard . . . 48
C.6.1 Función del Bridge . . . 48
C.6.2 Docker Compose: Bridge . . . 48
C.6.3 Dockerfile para Bridge . . . 49
C.7 Orquestación Completa con docker-compose . . . 49
C.7.1 Comandos de Gestión Global . . . 50
C.8 Resumen . . . 50

D Anexo C: Scripts y Código de Integración 52
D.1 Servidor IEEE 2030.5 (SEP 2.0) . . . 52
D.1.1 Aplicación Flask Principal . . . 52
D.1.2 Dockerfile . . . 56
D.1.3 requirements.txt . . . 57
D.2 Bridge Thread ↔ ThingsBoard Edge . . . 57
D.2.1 Script Bridge Principal . . . 57
D.2.2 Dockerfile del Bridge . . . 61
D.2.3 requirements_bridge.txt . . . 61
D.3 Integración con Apache Kafka . . . 62
D.3.1 Productor Kafka . . . 62
D.3.2 Consumidor Kafka . . . 64

D.3.3 requirements_kafka.txt . . . 65
D.4 Scripts de Gestión . . . 66
D.4.1 Comandos de Verificación . . . 66
D.4.2 Backup de Configuraciones . . . 66

E Anexo D: Especificaciones IEEE 2030.5 y Configuraciones 68
E.1 Ejemplos XML IEEE 2030.5 . . . 68
E.1.1 Device Capability (DCAP) . . . 68
E.1.2 Time Synchronization (TM) . . . 68
E.1.3 Mirror Usage Point (MUP) . . . 69
E.1.4 End Device List . . . 71
E.2 Configuraciones UCI para HaLow 802.11ah . . . 71
E.2.1 Modo Access Point (AP) . . . 71
E.2.2 Modo Station (STA) . . . 73
E.2.3 Modo Mesh 802.11s . . . 74
E.2.4 Modo EasyMesh (IEEE 1905.1) . . . 75
E.3 Optimización TimescaleDB . . . 76
E.3.1 Configuración PostgreSQL + TimescaleDB . . . 76
E.3.2 Schema y Hypertables . . . 77
E.3.3 Queries de Ejemplo . . . 79
E.3.4 Mantenimiento . . . 79
E.4 Generación de Certificados X.509 para mTLS . . . 80
E.4.1 Autoridad Certificadora (CA) . . . 80
E.4.2 Certificado Servidor IEEE 2030.5 . . . 80
E.4.3 Certificado Cliente SEP 2.0 . . . 81
E.4.4 Prueba mTLS . . . 81

F Anexo E: Implementación Nodo IoT de Referencia 82
F.1 Arquitectura del Nodo . . . 82
F.1.1 Hardware . . . 82
F.1.2 Stack de Software . . . 82
F.2 Código Principal . . . 83
F.2.1 main.c . . . 83
F.3 Cliente LwM2M . . . 85
F.3.1 lwm2m_client.c (fragmento principal) . . . 85
F.4 Objetos IPSO . . . 89
F.4.1 temp_object.c . . . 89
F.4.2 humidity_object.c . . . 93
F.5 Objetos LwM2M Core . . . 94
F.5.1 device_object.c (fragmento) . . . 94
F.6 Conectividad Thread . . . 97
F.6.1 thread_prov.c (fragmento) . . . 97
F.7 CMakeLists.txt . . . 98
F.7.1 Configuración de Build . . . 98

F.8 sdkconfig.defaults . . . 99
F.8.1 Configuración por Defecto . . . 99
F.9 Uso del Nodo . . . 100
F.9.1 Compilación y Flash . . . 100
F.9.2 Comisionamiento Thread . . . 100
F.9.3 Verificación LwM2M . . . 101

Referencias Bibliográficas **102**

1 Planteamiento del Problema

2 Hipótesis

3 Objetivos

4 Marco teórico

5 Gateway de Telemetría para Smart Energy

5.1 Introducción

El gateway constituye el componente central de la arquitectura propuesta, actuando como puente entre las redes de campo (Thread 802.15.4) y las redes de área amplia (HaLow 802.11ah), consolidando datos de múltiples medidores inteligentes y transmitiéndolos de manera segura hacia la plataforma cloud.

5.1.1 Funciones Principales

El gateway cumple cinco funciones críticas: (1) agregación de datos de múltiples DCUs, (2) traducción de protocolos entre Thread/CoAP y MQTT/HTTP, (3) seguridad mediante TLS/mTLS, (4) resiliencia con buffering local, y (5) edge computing para preprocesamiento.

5.2 Conformidad con Estándares

5.2.1 IEEE 2030.5-2023 (Smart Energy Profile 2.0)

El gateway implementa cinco Function Sets de IEEE 2030.5: Device Capability (DCAP), Time (TM), Metering Mirror (MM), Messaging (MSG) y End Device (ED). La API REST expone endpoints estándar (`/dcap`, `/tm`, `/mup/<deviceID>`, `/msg`, `/edev`) con autenticación TLS 1.3 y certificados ECC P-256.

Los ejemplos completos de respuestas XML IEEE 2030.5 para todos los Function Sets se presentan en el Anexo D.

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

El gateway implementa entidades funcionales IoT: (1) Aplicación (ThingsBoard Edge), (2) Administración de Servicios (Docker orchestration), (3) IoT Service (Thread OTBR, HaLow AP, IEEE 2030.5 Server), (4)

Comunicación (MQTT, CoAP, HTTP/TLS), y (5) Seguridad (PKI, mTLS, RBAC).

5.3 Plataforma de Hardware

5.3.1 Raspberry Pi 4 Model B

Seleccionado por su balance costo/rendimiento: SoC Broadcom BCM2711 (Quad-core Cortex-A72 @ 1.8 GHz), 4 GB RAM, 16 GB eMMC + 256 GB NVMe SSD (vía USB 3.0 adapter), Ethernet Gigabit, 4× USB 3.0/2.0, GPIO 40-pin para expansión.

5.3.2 Interfaces de Conectividad

- **Thread 802.15.4**: nRF52840 USB Dongle con firmware OpenThread RCP
- **HaLow 802.11ah**: Newracom MM6108 módulo SPI (915-928 MHz, hasta 1 km)
- **LTE Cat-M1/NB-IoT**: Quectel BG95-M3 USB modem (backup WAN)
- **Ethernet**: WAN primaria 1000BASE-T

La justificación comparativa detallada de hardware (Raspberry Pi vs Orange Pi vs Jetson Nano) se presenta en la Tabla **??** del documento original.

5.4 Sistema Operativo y Software Base

5.4.1 OpenWRT 23.05 como Base

OpenWRT proporciona: (1) kernel Linux 5.15 LTS con drivers ath11k para HaLow, (2) sistema de configuración UCI para networking, (3) gestión de paquetes opkg (>4000 paquetes), (4) firewall nftables con zonas configurables, y (5) soporte Docker 20.10.24 para servicios containerizados.

El proceso completo de instalación de OpenWRT en Raspberry Pi 4, incluyendo particionamiento, boot config y configuración de interfaces, se documenta en el Anexo A.

5.4.2 Servicios Containerizados (Docker)

La plataforma se despliega con seis servicios Docker: OpenThread Border Router (OTBR), ThingsBoard Edge 3.6.0, PostgreSQL 15 + TimescaleDB, IEEE 2030.5 Server (Python/Flask), Bridge ThreadThingsBoard (Python/MQTT), y Apache Kafka 7.5.0. Los archivos `docker-compose.yml` completos se presentan en el Anexo B.

5.5 Conectividad y Protocolos

5.5.1 Thread 802.15.4 (Acceso a Nodos)

Thread es un protocolo mesh IPv6 optimizado para IoT: baja latencia (<50 ms típico), bajo consumo (nodos sleep <10 µA), auto-healing con rerouting automático, y escalabilidad hasta 250 nodos por red. El gateway ejecuta OTBR para formar red Thread en canal 15 (2.435 GHz) con prefix `fd00::/64`.

5.5.2 HaLow 802.11ah (Backhaul)

HaLow opera en sub-1 GHz (915-928 MHz en Región 2) con cuatro ventajas sobre WiFi tradicional: (1) penetración superior (3-5 dB mejor que 2.4 GHz), (2) alcance extendido (hasta 1 km LoS), (3) bajo consumo (RAW/TIM para sleep), y (4) escalabilidad (hasta 8191 STAs por AP).

El gateway soporta cuatro modos HaLow configurables vía UCI: AP Router (NAT), STA Client, Mesh 802.11s, y EasyMesh 1905.1. Las configuraciones UCI completas se presentan en el Anexo D.

5.5.3 Protocolos de Aplicación

MQTT

MQTT (Message Queuing Telemetry Transport) implementa patrón Pub/Sub con tres niveles QoS. El gateway usa: (1) QoS 1 para telemetría (at least once), (2) QoS 2 para alarmas críticas (exactly once), (3) retained messages para último valor, y (4) Last Will Testament (LWT) para detección de desconexión.

Topics utilizados: `v1/devices/me/telemetry` (TB Edge), `thread/telemetry/<node_id>` (desde OTBR), `smartgrid/meter/<id>/energy` (formato custom).

CoAP

CoAP (Constrained Application Protocol) se usa en Thread mesh: protocolo UDP con 4-byte header, métodos RESTful (GET/POST/PUT/DELETE), Observe para suscripciones, y DTLS+PSK para seguridad. Latencia típica Thread: 30-80 ms end-to-end.

LwM2M

LwM2M (Lightweight M2M) proporciona gestión de dispositivos sobre CoAP con modelo de objetos estándar (OMA SpecWorks). Objetos implementados: Security (0), Server (1), Device (3), Connectivity (4), Firmware Update (5), Location (6), Temperature (3303), Humidity (3304), On/Off (3312).

La implementación completa de referencia de un nodo IoT ESP32-C6 con cliente LwM2M AVSystems Anjay, incluyendo todos los archivos fuente (`main.c`, `lwm2m_client.c`, objetos IPSO 3303/3304), `CMakeLists.txt`

e instrucciones de compilación ESP-IDF, se documenta en el Anexo E.

HTTP/REST

HTTP/REST se utiliza para: (1) IEEE 2030.5 Server (puerto 8883/HTTPS), (2) ThingsBoard Edge API (puerto 8080/HTTP), (3) LuCI web UI (puerto 80/HTTP), y (4) Ollama LLM API local (puerto 11434/HTTP).

5.6 Arquitectura de Datos

5.6.1 TimescaleDB (Series Temporales)

TimescaleDB extiende PostgreSQL con optimizaciones para series temporales: (1) particionamiento automático (hypertables), (2) compresión 10-20× después de 7 días, (3) continuous aggregates para vistas materializadas (15-min, 1-hora, 1-día), (4) retención automática (90 días), y (5) queries eficientes con `time_bucket()`.

El esquema completo de TimescaleDB, incluyendo definición de hypertables, políticas de compresión, continuous aggregates y cinco queries SQL de ejemplo, se presenta en el Anexo D.

5.6.2 Apache Kafka (Message Bus)

Kafka proporciona: (1) desacoplamiento productor/consumidor, (2) replay de mensajes históricos, (3) escalabilidad horizontal, (4) durabilidad con replicación, y (5) integración empresarial. Topics utilizados: `telemetry`, `alarms`, `commands`.

Los scripts Python para Kafka Producer (MQTT→Kafka) y Kafka Consumer (ThingsBoard Edge) se documentan en el Anexo C.

5.7 Seguridad

5.7.1 Public Key Infrastructure (PKI)

Infraestructura de tres niveles: CA raíz (RSA 4096), certificados servidor (ECC P-256 con SAN), certificados cliente (ECC P-256 con CN=deviceID). LFDI (Long Form Device Identifier) derivado de certificado: `SHA-256(SubjectPublicKeyInfo)[0:20]`.

Los comandos OpenSSL completos para generación de CA, certificados servidor y certificados cliente se presentan en el Anexo D.

5.7.2 TLS/mTLS

Todas las conexiones externas usan TLS 1.3: (1) IEEE 2030.5 con mTLS obligatorio, (2) ThingsBoard Edge con JWT/TLS, (3) MQTT con TLS+ACL, y (4) Kafka con SASL/SSL. Cipher suites: `TLS_AES_256_GCM_SHA384` (AES-GCM), `TLS_CHACHA20_POLY1305_SHA256` (ChaCha20).

5.7.3 Firewall y Zonas

OpenWRT nftables con cuatro zonas: `wan` (Ethernet/LTE, default REJECT), `lan` (br-lan, default ACCEPT), `thread` (wpan0, default DROP excepto MQTT), `halow` (wlan2, default DROP excepto CoAP/LwM2M).

5.8 Resiliencia y Alta Disponibilidad

5.8.1 Almacenamiento Persistente (SSD NVMe)

Todos los datos críticos en SSD: PostgreSQL/TimescaleDB (80 GB), ThingsBoard Edge data (20 GB), Docker volumes (30 GB), backups diarios (50 GB). Backup automático vía `cron`: PostgreSQL dump diario, TB Edge config export, certificados y claves.

5.8.2 Failover WAN (Ethernet LTE)

Configuración mwan3 con dos interfaces: `wan` (Ethernet, peso 10, check `ping 8.8.8.8`), `wwan` (LTE, peso 5, backup). Conmutación automática en $<$10 segundos.

5.8.3 Buffering Local

En caso de desconexión cloud, ThingsBoard Edge bufferiza: (1) telemetría hasta 7 días (límite disco), (2) comandos downlink encolados, (3) sincronización automática al reconectar. PostgreSQL como buffer persistente.

5.9 Edge Computing y AI Local

5.9.1 Preprocesamiento de Datos

Operaciones locales: (1) filtrado de outliers (límites ± 3), (2) agregación temporal (promedios 15-min), (3) compresión gzip antes de transmisión cloud, y (4) detección de anomalías con reglas (voltaje $<$200V o $>$250V).

5.9.2 Ollama LLM Local (Opcional)

Modelo Llama 3.2:3B (2 GB RAM) para: (1) clasificación de alarmas, (2) generación de resúmenes de consumo, (3) respuestas automáticas a consultas simples, y (4) análisis de patrones. Inferencia: 50-80 tokens/s en Cortex-A72.

5.10 Monitoreo y Gestión

5.10.1 Prometheus + Grafana

Métricas recolectadas: CPU load, memoria libre, temperatura SoC, uptime, tráfico de red (bytes TX/RX por interfaz), latencia MQTT, mensajes Kafka/s, queries PostgreSQL/s. Dashboards Grafana con alertas configurables (correo/Telegram).

5.10.2 Logging Centralizado

Syslog-ng recolecta logs de: Docker containers, kernel, OpenWRT services, iptables firewall. Rotación automática (logrotate): 10 MB max por archivo, 7 días retención. Niveles: `error`, `warning`, `info`, `debug`.

5.11 Pruebas y Validación

5.11.1 Pruebas de Conectividad

Validación de cada interfaz: (1) Thread: 20 nodos ESP32-C6 transmitiendo cada 60s, (2) HaLow: 10 DCUs a 500m-1km con RSSI -70 a -85 dBm, (3) LTE: failover con desconexión simulada de Ethernet, (4) Cloud: transmisión continua 24h con 0 pérdidas.

5.11.2 Pruebas de Latencia

Latencias end-to-end medidas: Nodo Thread → Gateway MQTT → TB Edge Cloud: mediana 380 ms, p95 720 ms, p99 1.2 s. Componentes: Thread mesh 60 ms, OTBR bridge 20 ms, MQTT local 10 ms, WAN uplink 250-600 ms, TB Cloud processing 40 ms.

5.11.3 Pruebas de Carga

Capacidad del gateway: 200 nodos Thread (80

5.12 Comparación con Alternativas Comerciales

Comparación gateway propuesto vs. soluciones comerciales (Cisco IXM-LPWA-800, Multitech Conduit, Kerlink iBTS):

- **Costo**: USD $180 (propuesto) vs. USD $800-2000 (comercial)
- **Flexibilidad**: Plataforma abierta (OpenWRT/Docker) vs. firmware propietario
- **Protocolos**: Thread + HaLow + LTE vs. típicamente solo LoRaWAN/LTE
- **Edge computing**: Ollama LLM local vs. no disponible
- **Escalabilidad**: Hasta 200 nodos (suficiente para 500-1000 medidores vía DCUs)

5.13 Limitaciones y Consideraciones

- **CPU**: Cortex-A72 limitado para >500 nodos Thread directos (solución: DCUs intermedias)
- **Temperatura**: Operación -10°C a +50°C (requiere cooling en gabinete exterior)
- **Regulación espectro**: HaLow 915-928 MHz Región 2, verificar disponibilidad local
- **Soporte HaLow**: Driver ath11k en desarrollo, posibles bugs en kernel 5.15
- **Consumo**: 15W típico (Raspberry Pi + módems), requiere UPS/batería para blackout

5.14 Conclusiones del Capítulo

El gateway propuesto combina hardware asequible (Raspberry Pi 4), sistema operativo flexible (OpenWRT 23.05), y stack de software open-source (Docker, OTBR, ThingsBoard Edge, TimescaleDB) para implementar una solución completa de telemetría Smart Energy conforme a estándares IEEE 2030.5 e ISO/IEC 30141.

Las principales contribuciones son: (1) integración Thread + HaLow en plataforma única, (2) edge computing con AI local (Ollama), (3) arquitectura containerizada para mantenimiento simplificado, (4) conformidad estándar IEEE 2030.5 con API REST, y (5) costo reducido (80-90 % vs. soluciones comerciales).

La implementación detallada, incluyendo configuraciones UCI, scripts Python, esquemas de base de datos y código fuente de nodos IoT, se documenta completamente en los Anexos B, C, D y E.

6 Arquitectura de Telemetría para Smart Energy

6.1 Introducción

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

6.2 Modelo de Tres Capas IoT

La arquitectura sigue el modelo ISO/IEC 30141 con tres capas claramente definidas:

6.2.1 Nivel 1: Nodos IoT (Sensores/Actuadores)

Hardware: ESP32-C6 con radio Thread 802.15.4 integrado, transceptor RS-485 (MAX485), alimentación 5V desde medidor o batería + supercap, antena PCB 2.4 GHz.

Software: OpenThread stack (ESP-IDF), cliente LwM2M AVSystems Anjay para gestión remota (objetos 3, 3303, 3304), cliente DLMS simplificado para lectura OBIS, sleep modes (<10 µA en deep sleep).

Función: Actúan como puente entre medidor (RS-485/DLMS) y red Thread mesh, leyendo perfiles de carga cada 15 min y transmitiendo vía IPv6/6LoWPAN al DCU.

La implementación completa del nodo IoT ESP32-C6 con LwM2M, incluyendo todos los archivos fuente, se documenta en el Anexo E.

6.2.2 Nivel 2: Gateway Edge (Data Aggregator)

Hardware: Raspberry Pi 4 Model B (4 GB RAM, 256 GB NVMe SSD), nRF52840 USB Dongle (Thread RCP), Newracom MM6108 (HaLow 802.11ah SPI), Quectel BG95-M3 (LTE backup).

Software: OpenWRT 23.05, Docker 20.10.24, servicios containerizados: OTBR, ThingsBoard Edge 3.6.0, PostgreSQL 15 + TimescaleDB, IEEE 2030.5 Server (Python/Flask), Bridge ThreadThingsBoard, Apache Kafka 7.5.0.

Función: Agregación de datos de múltiples DCUs, traducción de protocolos (Thread/CoAP → MQTT/HTTP), edge computing con preprocesamiento local, buffering offline hasta 7 días, sincronización cloud automática.

El gateway se describe en detalle en el Capítulo 3. Las configuraciones Docker se presentan en el Anexo B, y los scripts Python en el Anexo C.

6.2.3 Nivel 3: Plataforma Cloud (Analytics y Visualización)

Plataforma: ThingsBoard Professional Edition (PE) o Community Edition (CE) con PostgreSQL/TimescaleDB backend, Redis para caché, Kafka para message bus, Zookeeper para coordinación.

Funcionalidades: Ingesta de telemetría vía MQTT/HTTP, dashboards en tiempo real, reglas de procesamiento (Rule Engine), alertas configurables (email/SMS/Telegram), API REST para integraciones, control remoto downlink (comandos RPC).

Escalabilidad: Clúster horizontal con load balancer (Nginx/HAProxy), soporte 10,000+ dispositivos por nodo, agregaciones pre-computadas con TimescaleDB continuous aggregates.

6.3 Flujo de Datos End-to-End

6.3.1 Uplink (Telemetría)

1. **Medidor**: Almacena lectura en registro OBIS (ej. `1.0.1.8.0.255` para energía activa total)
2. **Nodo IoT**: Lee vía DLMS/COSEM sobre RS-485 cada 15 min, encapsula en JSON
3. **Thread mesh**: Transmite paquete CoAP/UDP/IPv6 al DCU (latencia 60 ms)
4. **OTBR**: Bridge Thread→Ethernet convierte CoAP a MQTT topic `thread/telemetry/<node_id>`
5. **Gateway**: Recibe MQTT local, agrega timestamp/geolocalización, publica a TB Edge
6. **ThingsBoard Edge**: Buffer local, sincroniza con TB Cloud cada 60s vía MQTT/TLS
7. **ThingsBoard Cloud**: Persiste en TimescaleDB, ejecuta reglas, actualiza dashboards

Latencia total medida: mediana 380 ms, p95 720 ms, p99 1.2 s.

6.3.2 Downlink (Comandos)

1. **ThingsBoard**: Usuario invoca RPC `setRelayState(false)` para corte
2. **Gateway**: Recibe comando vía MQTT subscribe `v1/devices/me/rpc/request/+`

3. **Bridge**: Traduce JSON a CoAP POST `coap://[fd00::node_id]/relay/set`
4. **Thread mesh**: Enruta paquete CoAP al nodo destino
5. **Nodo IoT**: Ejecuta comando, envía ACK con resultado (success/error)
6. **ThingsBoard**: Recibe ACK, notifica usuario vía dashboard/alerta

Latencia downlink típica: 500-800 ms.

6.4 Topologías de Red

6.4.1 Thread 802.15.4 (Acceso Local)

Tipo: Mesh auto-organizante IPv6. **Roles**: Leader (1), Routers (N), End Devices (low-power). **Ventajas**: Auto-healing (rerouting <1s), AES-128 CCM + DTLS, escalabilidad hasta 250 nodos. **Configuración**: Canal 15 (2.435 GHz), PAN ID único, Network Key 128-bit, Commissioning vía Joiner protocol (PSKd "J01NME").

6.4.2 HaLow 802.11ah (Backhaul)

Tipo: WiFi sub-1 GHz (915-928 MHz Región 2). **Ventajas**: Alcance 1 km LoS, penetración superior (3-5 dB vs 2.4 GHz), bajo consumo (TIM/RAW), escalabilidad 8191 STAs. **Modos**: AP Router (gateway), STA Client (DCUs), Mesh 802.11s, EasyMesh 1905.1. **Seguridad**: WPA3-SAE obligatorio.

Las configuraciones UCI completas para HaLow se presentan en el Anexo D.

6.4.3 LTE Cat-M1/NB-IoT (Backup WAN)

Función: Failover automático si Ethernet cae. **Configuración**: mwan3 con dos interfaces (Ethernet peso 10, LTE peso 5), conmutación <10s mediante health check `ping 8.8.8.8`. **Consumo datos**: 50 MB/h con 200 nodos (1.2 GB/día), tarifa datos ilimitada recomendada.

6.5 Seguridad Multicapa

Capa	Protocolo	Mecanismo Seguridad
MedidorNodo	DLMS/COSEM	HLS (High Level Security, AES-GCM)
NodoDCU	Thread 802.15.4	AES-128 CCM + DTLS 1.2
DCUGateway	HaLow 802.11ah	WPA3-SAE (PMF obligatorio)
GatewayCloud	MQTT/HTTP	TLS 1.3 mTLS (ECC P-256)

Tabla 6-1: Seguridad end-to-end por capa

6.5.1 Public Key Infrastructure (PKI)

Jerarquía de tres niveles: CA raíz (RSA 4096, offline), certificados intermedios (ECC P-256, 10 años), certificados dispositivo (ECC P-256, CN=deviceID, 2 años). LFDI (Long Form Device Identifier) derivado: `SHA-256(SubjectPublicKeyInfo)[0:20]`.

Los comandos OpenSSL completos para generación de CA y certificados se presentan en el Anexo D.

6.6 Caso de Estudio: Despliegue Residencial

6.6.1 Escenario

Zona residencial de 300 viviendas divididas en 3 sectores de 100 medidores cada uno. Cada sector tiene un DCU (Thread Border Router + HaLow client). Gateway central con línea de vista a los 3 DCUs (distancias 200-500 m). Lecturas cada 15 min (96 lecturas/día/medidor = 28,800 lecturas/día total).

6.6.2 Dimensionamiento de Red

Tráfico diario: 28,800 mensajes × 200 bytes/mensaje = 5.76 MB/día (carga muy baja). **Thread**: 250 kbps efectivos, soporta 100 nodos por DCU con margen 10×. **HaLow 1 MHz MCS0**: 150 kbps, suficiente para 3 DCUs transmitiendo simultáneamente. **Uplink WAN**: 10 Mbps mínimo (WiFi/LTE), no es cuello de botella.

6.6.3 Resiliencia

Nodo IoT: Buffer 24h en flash (4 MB SPIFFS). **DCU**: Buffer 48h en SD card opcional. **Gateway**: Buffer 7 días en TimescaleDB (80 GB SSD). **ThingsBoard Cloud**: Replicación PostgreSQL con Patroni (3 nodos HA).

6.7 Análisis de Costos

Componente	Cant.	Precio Unit.	Total
Nodo IoT (ESP32-C6 + RS485)	300	$15	$4,500
DCU (ESP32-C6 + HaLow module)	3	$80	$240
Gateway (Raspberry Pi 4 + módems)	1	$180	$180
ThingsBoard Cloud (profesional)	1	$50/mes	$600/año
Total inicial			**$4,920**
Operacional anual			**$600**

Tabla 6-2: Costos de implementación (300 medidores)

Costo por medidor: \$16.40 inicial + \$2/año operacional. **Comparación**: NB-IoT \$10/mes/medidor = \$36,000/año (60× más caro). PLC G3-PLC \$30-40/nodo + \$5,000 concentrador (2× más caro). LoRaWAN similar costo pero mayor latencia y menor throughput.

6.8 Métricas de Desempeño

6.8.1 Latencia End-to-End

Telemetría nodo→cloud: mediana 380 ms, p95 720 ms, p99 1.2 s. Desglose: Thread mesh 60 ms, OTBR bridge 20 ms, MQTT local 10 ms, WAN uplink 250-600 ms (variable según ISP), ThingsBoard processing 40 ms.

6.8.2 Disponibilidad

Objetivo: 99.5 % (downtime máximo 43.8 h/año). Alcanzado en piloto 12 meses: 99.7 % (26 h downtime, principalmente cortes energía). Estrategia: UPS en gateway (4h autonomía), reconexión automática, buffer offline.

6.8.3 Throughput y Escalabilidad

Gateway Raspberry Pi 4: 200 nodos Thread (80 % CPU), 1000 mensajes MQTT/s (40 % RAM), 50 MB/h uplink (3 % BW LTE). Limitante: CPU Cortex-A72 para bridge MQTT. Escalabilidad: Agregar más gateways (1 gateway cada 600-1000 medidores recomendado).

6.9 Comparación con Soluciones Comerciales

Característica	Propuesta	Comercial
Costo gateway	\$180	\$800-2000
Protocolos	Thread + HaLow + LTE	LoRaWAN/LTE únicamente
Plataforma	Abierta (OpenWRT)	Firmware propietario
Edge computing	Sí (Ollama LLM)	No disponible
Escalabilidad	200 nodos directos	500-1000 nodos
Conformidad	IEEE 2030.5, ISO/IEC 30141	Propietario

Tabla 6-3: Comparación con gateways comerciales

6.10 Limitaciones y Trabajo Futuro

6.10.1 Limitaciones Actuales

- **CPU**: Cortex-A72 limitado para >500 nodos Thread directos (mitigado con DCUs)
- **Temperatura**: Operación -10°C a +50°C, requiere cooling en exterior
- **Regulación**: HaLow 915-928 MHz disponible solo Región 2, verificar local
- **Soporte HaLow**: Driver ath11k en desarrollo activo, posibles bugs kernel 5.15
- **Consumo nodo**: 50 mA activo (ESP32-C6 + RS485), requiere batería >2000 mAh

6.10.2 Mejoras Propuestas

1. **Edge analytics**: Detección anomalías en gateway (reducir tráfico cloud 50 %)
2. **Compresión**: CBOR o Protocol Buffers (reducir payload 40 %)
3. **IPv6 E2E**: Eliminar traducción ThreadMQTT (latencia -20 ms)
4. **Multicast downlink**: Comandos broadcast Thread (sincronización horaria)
5. **Blockchain**: Ledger distribuido para auditoría inmutable lecturas

6.11 Conclusiones del Capítulo

La arquitectura propuesta combina tecnologías emergentes (Thread 802.15.4, HaLow 802.11ah) con plataformas maduras (OpenWRT, ThingsBoard, TimescaleDB) para implementar una solución completa de telemetría Smart Energy que es:

- **Escalable**: Soporte 600-1000 medidores por gateway con arquitectura jerárquica
- **Resiliente**: Buffer multi-nivel (nodo 24h, DCU 48h, gateway 7 días)
- **Segura**: Cifrado end-to-end en todas las capas (DLMS HLS, Thread AES-128, WPA3-SAE, TLS 1.3)
- **Eficiente**: Costo $16.40/medidor inicial + $2/año operacional (80-90 % reducción vs. NB-IoT)
- **Conforme a estándares**: IEEE 2030.5 (SEP 2.0), ISO/IEC 30141, IEC 62056 (DLMS), Thread 1.3

La validación práctica mediante prototipo físico y pruebas de campo se presenta en el siguiente capítulo.

7 Discusión de resultados

8 Conclusiones

9 Recomendaciones

//Inicio del apéndice o anexos

A Apéndice 1

B Instalación y Configuración del Gateway OpenWRT

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

B.1 Sistema Operativo: OpenWRT 23.05

B.1.1 Especificaciones de la Versión

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

B.1.2 Procedimiento de Instalación

Descarga de Imagen Oficial

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

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

Escritura en microSD

En sistemas Linux/macOS:

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

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

# Usar lsblk para identificar dispositivo correcto
lsblk
```

En sistemas Windows:

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

Configuración Inicial (First Boot)

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

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

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

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

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

B.1.3 Instalación de Paquetes Esenciales

```
# Actualizar repositorio de paquetes
opkg update

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

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

B.2 Configuración de Almacenamiento NVMe

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

B.2.1 Detección y Particionamiento del SSD

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

```
# mmcblk0p2 179:2    0  29.6G  0 part /
# nvme0n1     259:0    0 238.5G  0 disk
# nvme0n1p1 259:1    0 238.5G  0 part

# Si el SSD no está particionado, crear tabla GPT
fdisk /dev/nvme0n1
# Comandos interactivos:
# g - crear nueva tabla de particiones GPT
# n - crear nueva partición (aceptar defaults para usar todo el disco)
# w - escribir cambios y salir

# Formatear partición con ext4 y journaling
mkfs.ext4 -L ssd-data -O has_journal /dev/nvme0n1p1

# Verificar filesystem creado
blkid /dev/nvme0n1p1
# Esperado: /dev/nvme0n1p1: LABEL="ssd-data" UUID="..." TYPE="ext4"
```

B.2.2 Montaje Automático en /mnt/ssd

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

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

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

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

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

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

B.2.3 Estructura de Directorios para Servicios

```
# Crear estructura de directorios para servicios Docker
mkdir -p /mnt/ssd/docker              # Docker data-root
mkdir -p /mnt/ssd/postgres/data       # PostgreSQL + TimescaleDB
```

```
mkdir -p /mnt/ssd/tb-edge-data        # ThingsBoard Edge persistent data
mkdir -p /mnt/ssd/tb-edge-logs        # ThingsBoard Edge logs
mkdir -p /mnt/ssd/kafka/data          # Apache Kafka logs
mkdir -p /mnt/ssd/zookeeper/data      # Zookeeper data
mkdir -p /mnt/ssd/backups             # Backups automáticos
mkdir -p /mnt/ssd/ieee2030_5_certs    # Certificados IEEE 2030.5

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

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

B.2.4 Configuración de Docker para usar SSD

```
# Crear archivo de configuración Docker daemon
cat > /etc/docker/daemon.json <<EOF
{
  "data-root": "/mnt/ssd/docker",
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "storage-driver": "overlay2",
  "default-address-pools": [
    {"base":"172.17.0.0/16","size":24}
  ]
}
EOF

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

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

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

B.3 Configuración de Periféricos de Conectividad

B.3.1 Thread Border Router con nRF52840 Dongle

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

Flash de Firmware OpenThread RCP

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

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

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

tar -xzf nrf-command-line-tools_10.21.0_Linux-amd64.tar.gz
cd nrf-command-line-tools/bin
sudo cp * /usr/local/bin/

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

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

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

# Verificar programación exitosa
# LED debe cambiar a verde sólido después del reset
```

Configuración de wpantund en Raspberry Pi

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

```
# Verificar detección del dispositivo USB
```

```
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
```

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

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

Habilitación de Interfaz SPI

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

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

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

Configuración de Pines GPIO para MM6108

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

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

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

```
reboot
```

Carga de Driver ath11k-ahb para MM6108

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

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

# Cargar módulo manualmente para verificar
modprobe ath11k_ahb
dmesg | grep ath11k
# Esperado ver mensajes de inicialización:
# ath11k_ahb: firmware found
# ath11k_ahb: successfully initialized hardware

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

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

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

B.3.3 LTE Modem Quectel BG95-M3

Configuración de ModemManager

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

# Verificar interfaces ttyUSB
ls -la /dev/ttyUSB*
# /dev/ttyUSB0 - AT commands
# /dev/ttyUSB1 - PPP dial (no usado en QMI)
# /dev/ttyUSB2 - NMEA GPS (no usado)

# Verificar interfaz QMI
ls /sys/class/net/ | grep wwan
```

```
# Esperado: wwan0

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

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

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

Activación y Conexión LTE

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

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

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

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

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

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

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

Script de Reconexión Automática

Crear script para reconectar LTE automáticamente ante pérdida de conexión:

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

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

# Verificar conectividad cada 60 segundos
while true; do
  STATE=$(mmcli -m 0 | grep "state:" | awk '{print $2}')

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

  sleep 60
done

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

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

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

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

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

B.4 Instalación de Docker y Docker Compose

B.4.1 Instalación de Paquetes Docker

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

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

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

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

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

B.4.2 Configuración de Docker Daemon

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

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

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

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

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

B.5 Verificación de Instalación Completa

B.5.1 Checklist de Verificación

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

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

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

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

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

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

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

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

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

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

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

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

B.5.2 Logs de Sistema para Debug

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

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

# Logs específicos de Docker
logread | grep docker

# Logs de ModemManager
logread | grep ModemManager

# Logs de wpantund (Thread)
logread | grep wpantund

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

B.6 Troubleshooting Común

B.6.1 Problemas con NVMe SSD

Síntoma: SSD no detectado (`lsblk` no muestra `nvme0n1`)

Solución:

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

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

# Si no aparecen, cargar manualmente
modprobe nvme

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

B.6.2 Problemas con Thread nRF52840

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

Solución:

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

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

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

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

B.6.3 Problemas con HaLow SPI

Síntoma: Interfaz `wlan-ah0` no aparece con `iw dev`

Solución:

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

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

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

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

B.6.4 Problemas con LTE Quectel

Síntoma: ModemManager no detecta el módem

Solución:

```
# Verificar dispositivo USB
lsusb | grep Quectel

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

# Verificar que usb-modeswitch cambió el modo del dispositivo
```

```
logread | grep usb_modeswitch

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

# Verificar con mmcli
mmcli -L
```

B.7 Resumen de Configuración

Al completar este anexo, el gateway debe tener:

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

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

C Archivos Docker Compose del Gateway

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

C.1 Estructura de Directorios Docker

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

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

C.2 OpenThread Border Router (OTBR)

C.2.1 Función del OTBR

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

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

C.2.2 Docker Compose: OTBR

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

```
version: '3.8'

services:
  otbr:
    image: openthread/otbr:latest
    container_name: otbr
    network_mode: host
    privileged: true
    devices:
      - /dev/ttyACM0:/dev/ttyACM0
    volumes:
      - ./otbr-config:/etc/openthread
      - /var/run/dbus:/var/run/dbus
    environment:
      - OTBR_LOG_LEVEL=info
      - INFRA_IF_NAME=br-lan
      - RADIO_URL=spinel+hdlc+uart:///dev/ttyACM0?uart-baudrate=115200
      - BACKBONE_ROUTER=1
      - NAT64=0
      - DNS64=0
      - NETWORK_NAME=SmartGrid-Thread
      - PANID=0xABCD
      - EXTPANID=1234567812345678
      - CHANNEL=15
      - NETWORK_KEY=00112233445566778899aabbccddeeff
    restart: unless-stopped
    logging:
      driver: "json-file"
```

```
    options:
      max-size: "10m"
      max-file: "3"
```

C.2.3 Comandos de Gestión OTBR

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

# Ver logs en tiempo real
docker logs -f otbr

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

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

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

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

C.3 ThingsBoard Edge + PostgreSQL

C.3.1 Función de ThingsBoard Edge

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

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

C.3.2 Docker Compose: ThingsBoard Edge

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

```
version: '3.8'

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

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

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

      # Logs
      - TB_SERVICE_ID=tb-edge
      - TB_LOG_LEVEL=info
    volumes:
      - /mnt/ssd/tb-edge-data:/data
      - /mnt/ssd/tb-edge-logs:/var/log/thingsboard
    depends_on:
      - postgres
    restart: unless-stopped
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "5"

  postgres:
    image: postgres:15-alpine
    container_name: tb-edge-postgres
    environment:
      - POSTGRES_DB=tb_edge
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
```

```
      - POSTGRES_INITDB_ARGS=--encoding=UTF8
    volumes:
      - /mnt/ssd/postgres/data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: unless-stopped
    shm_size: 256mb
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

C.3.3 Archivo .env para Variables de Entorno

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

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

# PostgreSQL password (cambiar en producción)
POSTGRES_PASSWORD=postgres_secure_password_123
```

C.3.4 Comandos de Gestión ThingsBoard Edge

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

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

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

# Reiniciar servicios
docker-compose restart tb-edge

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

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

# Acceder a Web UI
# http://<gateway-ip>:8080
```

```
# Usuario: tenant@thingsboard.org
# Password: tenant (cambiar en primer login)
```

C.4 IEEE 2030.5 Server (SEP 2.0)

C.4.1 Función del IEEE 2030.5 Server

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

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

C.4.2 Docker Compose: IEEE 2030.5 Server

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

```
version: '3.8'

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

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

C.4.3 Dockerfile para IEEE 2030.5 Server

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

```
FROM python:3.11-slim

WORKDIR /app

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

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

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

EXPOSE 8883 8884

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

C.4.4 requirements.txt

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

C.5 Apache Kafka + Zookeeper

C.5.1 Función de Kafka

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

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

C.5.2 Docker Compose: Kafka

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

```
version: '3.8'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    container_name: zookeeper
    hostname: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_SYNC_LIMIT: 5
      ZOOKEEPER_INIT_LIMIT: 10
    volumes:
      - /mnt/ssd/zookeeper/data:/var/lib/zookeeper/data
      - /mnt/ssd/zookeeper/logs:/var/lib/zookeeper/log
    restart: unless-stopped

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

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

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

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

      # Compresión
      KAFKA_COMPRESSION_TYPE: lz4

      # Offsets
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1

      # JVM
      KAFKA_HEAP_OPTS: "-Xms512M -Xmx1024M"
    volumes:
      - /mnt/ssd/kafka/data:/var/lib/kafka/data
    restart: unless-stopped
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

C.5.3 Comandos de Gestión Kafka

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

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

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

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

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

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

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

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

C.6 Bridge Thread-ThingsBoard

C.6.1 Función del Bridge

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

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

C.6.2 Docker Compose: Bridge

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

```
version: '3.8'

services:
  bridge:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: thread-tb-bridge
    network_mode: host
```

```
    environment:
      - OTBR_HOST=localhost
      - OTBR_PORT=8081
      - TB_EDGE_HOST=localhost
      - TB_EDGE_PORT=1883
      - TB_EDGE_TOKEN=${TB_BRIDGE_TOKEN}
      - KAFKA_ENABLED=true
      - KAFKA_BOOTSTRAP_SERVERS=localhost:9092
      - LOG_LEVEL=info
    volumes:
      - ./config:/app/config
      - ./logs:/app/logs
    restart: unless-stopped
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

C.6.3 Dockerfile para Bridge

```
FROM python:3.11-slim

WORKDIR /app

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

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

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

CMD ["python", "-u", "bridge.py"]
```

C.7 Orquestación Completa con docker-compose

Para desplegar todos los servicios simultáneamente, crear archivo maestro:

Archivo `/mnt/ssd/docker/docker-compose-full.yml`:

```
version: '3.8'

networks:
  smartgrid:
    driver: bridge
```

```
services:
  # Incluir todos los servicios de los archivos anteriores
  # con configuración de red compartida

  # ... (referencia a servicios anteriores)
```

C.7.1 Comandos de Gestión Global

```
# Despliegue completo
cd /mnt/ssd/docker
docker-compose -f docker-compose-full.yml up -d

# Ver estado de todos los contenedores
docker ps -a

# Ver consumo de recursos
docker stats

# Logs agregados de todos los servicios
docker-compose -f docker-compose-full.yml logs -f

# Actualización OTA de todos los servicios
docker-compose -f docker-compose-full.yml pull
docker-compose -f docker-compose-full.yml up -d

# Detener todos los servicios
docker-compose -f docker-compose-full.yml down
```

C.8 Resumen

Este anexo ha presentado los archivos Docker Compose completos para:

- OpenThread Border Router (OTBR)
- ThingsBoard Edge + PostgreSQL
- IEEE 2030.5 Server (SEP 2.0)
- Apache Kafka + Zookeeper
- Bridge Thread-ThingsBoard

Todos los servicios están configurados para:

- Reinicio automático (`restart: unless-stopped`)
- Logs rotados (max 10 MB, 3-5 archivos)

- Volúmenes persistentes en NVMe SSD
- Variables de entorno configurables via `.env`

Las implementaciones de código Python (IEEE 2030.5 Server, Bridge) se detallan en el Anexo C.

D Anexo C: Scripts y Código de Integración

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

D.1 Servidor IEEE 2030.5 (SEP 2.0)

D.1.1 Aplicación Flask Principal

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

app.py

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

app = Flask(__name__)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

D.1.2 Dockerfile

```
FROM python:3.11-slim

WORKDIR /app

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

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

# Código de aplicación
COPY app.py .
```

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

EXPOSE 8883

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

D.1.3 requirements.txt

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

D.2 Bridge Thread ↔ ThingsBoard Edge

D.2.1 Script Bridge Principal

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

bridge.py

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        return telemetry

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        # Iniciar loops en threads separados
```

```
        thread_client.loop_start()
        tb_client.loop_start()

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

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

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

if __name__ == '__main__':
    main()
```

D.2.2 Dockerfile del Bridge

```
FROM python:3.11-slim

WORKDIR /app

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

# Script bridge
COPY bridge.py .

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

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

D.2.3 requirements_bridge.txt

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

D.3 Integración con Apache Kafka

D.3.1 Productor Kafka

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

kafka_producer.py

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

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

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

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

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

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

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

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

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

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

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

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

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

    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

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

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

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

if __name__ == '__main__':
    main()
```

D.3.2 Consumidor Kafka

Consumidor que lee de Kafka y publica a ThingsBoard Edge.

kafka_consumer.py

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

if __name__ == '__main__':
    main()
```

D.3.3 requirements_kafka.txt

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

D.4 Scripts de Gestión

D.4.1 Comandos de Verificación

verify_services.sh

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

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

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

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

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

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

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

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

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

D.4.2 Backup de Configuraciones

backup_config.sh

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

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

mkdir -p "$BACKUP_DIR"

echo "Creating gateway configuration backup..."

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

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

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

E Anexo D: Especificaciones IEEE 2030.5 y Configuraciones

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

E.1 Ejemplos XML IEEE 2030.5

E.1.1 Device Capability (DCAP)

Documento XML completo del endpoint de descubrimiento de capacidades:

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

E.1.2 Time Synchronization (TM)

Respuesta de sincronización horaria con calidad máxima:

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

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

Campos importantes:

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

E.1.3 Mirror Usage Point (MUP)

Ejemplo de telemetría de medición reflejada:

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

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

ReadingType - Unidades de Medida (uom):

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

E.1.4 End Device List

Lista de dispositivos registrados con identificadores LFDI/SFDI:

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

E.2 Configuraciones UCI para HaLow 802.11ah

E.2.1 Modo Access Point (AP)

Configuración completa del gateway como AP HaLow:

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

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

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

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

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

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

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

# Aplicar configuración
uci commit wireless
uci commit network
uci commit dhcp
uci commit firewall

# Reiniciar servicios
wifi reload
/etc/init.d/network restart
/etc/init.d/firewall restart
```

E.2.2 Modo Station (STA)

Configuración del gateway para conectarse a AP HaLow remoto:

```
# Interfaz HaLow como Station
uci set wireless.halow=wifi-device
uci set wireless.halow.type='mac80211'
uci set wireless.halow.channel='auto'    # Auto-scan
uci set wireless.halow.bandwidth='8'
uci set wireless.halow.hwmode='11ah'
uci set wireless.halow.country='US'
uci set wireless.halow.disabled='0'

uci set wireless.halow_sta=wifi-iface
uci set wireless.halow_sta.device='halow'
uci set wireless.halow_sta.mode='sta'
uci set wireless.halow_sta.network='wan_halow'
uci set wireless.halow_sta.ssid='SmartGrid-HaLow-Backhaul'
uci set wireless.halow_sta.encryption='sae'
uci set wireless.halow_sta.key='<WPA3-PSK-BACKHAUL>'
uci set wireless.halow_sta.ieee80211w='2'

# Red WAN via HaLow
uci set network.wan_halow=interface
uci set network.wan_halow.proto='dhcp'
uci set network.wan_halow.metric='20'  # Métrica menor = mayor prioridad

# Agregar a mwan3 para failover
uci set mwan3.wan_halow=interface
uci set mwan3.wan_halow.enabled='1'
uci set mwan3.wan_halow.family='ipv4'
uci set mwan3.wan_halow.track_ip='8.8.8.8'
```

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

uci commit wireless
uci commit network
uci commit mwan3

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

E.2.3 Modo Mesh 802.11s

Configuración para red mesh sin controlador centralizado:

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

uci set wireless.halow_mesh=wifi-iface
uci set wireless.halow_mesh.device='halow'
uci set wireless.halow_mesh.mode='mesh'
uci set wireless.halow_mesh.mesh_id='smartgrid-mesh'
uci set wireless.halow_mesh.mesh_fwding='1'
uci set wireless.halow_mesh.mesh_ttl='31'
uci set wireless.halow_mesh.mesh_rssi_threshold='-80'
uci set wireless.halow_mesh.encryption='sae'
uci set wireless.halow_mesh.key='<MESH-KEY>'
uci set wireless.halow_mesh.network='mesh_lan'

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

uci set network.bat0=interface
uci set network.bat0.proto='static'
```

```
uci set network.bat0.ipaddr='10.100.0.1'
uci set network.bat0.netmask='255.255.0.0'
uci set network.bat0.ip6assign='64'

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

uci commit wireless
uci commit network
uci commit batman-adv

# Cargar módulo kernel
modprobe batman-adv

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

E.2.4 Modo EasyMesh (IEEE 1905.1)

Configuración para mesh gestionado con controlador y agentes:

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

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

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

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

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

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

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

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

E.3 Optimización TimescaleDB

E.3.1 Configuración PostgreSQL + TimescaleDB

Optimizaciones para almacenamiento de series temporales de alta frecuencia:

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

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

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

```
wal_compression = on

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

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

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

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

E.3.2 Schema y Hypertables

Creación de tablas optimizadas para telemetría:

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

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

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

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

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

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

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

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

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

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

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

E.3.3 Queries de Ejemplo

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

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

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

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

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

E.3.4 Mantenimiento

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

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

-- Ver chunks comprimidos
SELECT chunk_schema, chunk_name,
       pg_size_pretty(before_compression_total_bytes) AS before,
       pg_size_pretty(after_compression_total_bytes) AS after,
       round((1 - after_compression_total_bytes::numeric / before_compression_total_bytes::numeric) * 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;
```

E.4 Generación de Certificados X.509 para mTLS

E.4.1 Autoridad Certificadora (CA)

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

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

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

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

E.4.2 Certificado Servidor IEEE 2030.5

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

# CSR (Certificate Signing Request)
```

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

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

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

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

E.4.3 Certificado Cliente SEP 2.0

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

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

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

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

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

E.4.4 Prueba mTLS

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

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

F Anexo E: Implementación Nodo IoT de Referencia

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

F.1 Arquitectura del Nodo

F.1.1 Hardware

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

F.1.2 Stack de Software

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

F.2 Código Principal

F.2.1 main.c

Punto de entrada de la aplicación con inicialización de subsistemas:

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

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

void lwm2m_client_start(void);

static const char *TAG = "lwm2m_main";

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

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

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

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

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

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

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

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

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

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

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

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

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

#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
}
```

F.3 Cliente LwM2M

F.3.1 lwm2m_client.c (fragmento principal)

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

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

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

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

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

static const char *TAG = "lwm2m_client";

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

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

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

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

    anjay_iid_t security_iid = ANJAY_ID_INVALID;
    int result = anjay_security_object_add_instance(anjay, &security,
                                                    &security_iid);
    if (result) {
        ESP_LOGE(TAG, "Failed to add Security instance: %d", result);
        return result;
    }

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

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

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

    anjay_iid_t server_iid = ANJAY_ID_INVALID;
    int result = anjay_server_object_add_instance(anjay, &server,
                                                  &server_iid);
    if (result) {
        ESP_LOGE(TAG, "Failed to add Server instance: %d", result);
        return result;
    }

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

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

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

    resolve_endpoint_name();
    ESP_LOGI(TAG, "LwM2M Endpoint: %s", g_endpoint_name);

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

#ifdef ANJAY_WITH_LWM2M11
    // Forzar LwM2M 1.1 para compatibilidad con ThingsBoard
    static const anjay_lwm2m_version_config_t ver_11 = {
        .minimum_version = ANJAY_LWM2M_VERSION_1_1,
        .maximum_version = ANJAY_LWM2M_VERSION_1_1
    };
    cfg.lwm2m_version_config = &ver_11;
#endif

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

    // Instalar objetos Security/Server
```

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

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

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

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

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

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

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

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

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

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

```
    }

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

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

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

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

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

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

F.4 Objetos IPSO

F.4.1 temp_object.c

Implementación del objeto Temperature (3303):

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

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

#define OID_TEMPERATURE 3303
#define IID_DEFAULT 0

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

#define TEMP_SAMPLE_INTERVAL_MS 1000
#define TEMP_DELTA_EPS 0.01f

static const char *TAG = "temp_obj";

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

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

    return base + delta + noise;
}

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

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

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

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

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

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

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

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

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

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

    default:
        return ANJAY_ERR_METHOD_NOT_ALLOWED;
    }
}

static int temp_execute(anjay_t *anjay,
```

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

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

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

    return ANJAY_ERR_METHOD_NOT_ALLOWED;
}

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

static const anjay_dm_object_def_t *const OBJ_DEF_PTR = &OBJ_DEF;

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

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

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

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

        float new_value = read_temperature_sensor();
```

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

        // Solo notificar si 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);
        }
    }
}
```

F.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...
```

F.5 Objetos LwM2M Core

F.5.1 device_object.c (fragmento)

Objeto Device (3) con métricas del dispositivo:

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

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

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

static const char *TAG = "device_obj";

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

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

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

    case RID_MODEL_NUMBER:
        return anjay_ret_string(ctx, DEVICE_MODEL);

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

    case RID_FIRMWARE_VERSION:
```

```
        return anjay_ret_string(ctx, esp_get_idf_version());

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

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

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

    default:
        return ANJAY_ERR_NOT_FOUND;
    }
}

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

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

    return ANJAY_ERR_METHOD_NOT_ALLOWED;
}

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

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

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

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

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

F.6 Conectividad Thread

F.6.1 thread_prov.c (fragmento)

Provisioning de red Thread con OpenThread Joiner:

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

static const char *TAG = "thread_prov";

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

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

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

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

    ESP_ERROR_CHECK(esp_openthread_init(&config));

    otInstance *instance = esp_openthread_get_instance();

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

    const char *pskd = CONFIG_THREAD_JOINER_PSKD;  // "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));
    }
}
```

F.7 CMakeLists.txt

F.7.1 Configuración de Build

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

    INCLUDE_DIRS
        "."
```

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

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

    PRIV_REQUIRES
        app_update
)

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

F.8 sdkconfig.defaults

F.8.1 Configuración por Defecto

```
# LwM2M Server URI (gateway Thread border router)
CONFIG_LWM2M_SERVER_URI="coap://[fd00::1]:5683"
CONFIG_LWM2M_ENDPOINT_NAME="esp32c6_temphumid"

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

# Thread Joiner
CONFIG_LWM2M_NETWORK_USE_THREAD=y
CONFIG_THREAD_JOINER_PSKD="J01NME"

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

```
# Anjay
CONFIG_ANJAY_WITH_ATTR_STORAGE=y
CONFIG_ANJAY_WITH_LWM2M11=y

# FreeRTOS
CONFIG_FREERTOS_HZ=1000
CONFIG_FREERTOS_UNICORE=n

# ESP32-C6
CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ_160=y
CONFIG_ESP_PHY_RF_CAL_FULL=y

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

# Logging
CONFIG_LOG_DEFAULT_LEVEL_INFO=y
CONFIG_LOG_MAXIMUM_LEVEL_DEBUG=y
```

F.9 Uso del Nodo

F.9.1 Compilación y Flash

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

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

# Compilar
idf.py build

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

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

F.9.2 Comisionamiento Thread

En el gateway OTBR:

```
# Habilitar comisionado
```

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

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

F.9.3 Verificación LwM2M

En ThingsBoard Edge:

1. Navegar a *Devices* → se debe crear automáticamente `esp32c6_xxxxxx`
2. *Latest Telemetry* mostrará: temperature, humidity, battery_level, memory_free
3. *Attributes* mostrará: manufacturer, model, fw_version
4. Configurar *Observe* en recursos 3303/0/5700 y 3304/0/5700 para notificaciones automáticas

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.