



UNIVERSITARY MASTER IN INDUSTRIAL ENGINEERING

FINAL MASTER THESIS

Analysis of an edge-computing-based solution for
local data processing at secondary substations

Author: Néstor Rodríguez Pérez

Director: Miguel Ángel Sanz Bobi

Co-Director: Aurelio Sánchez Paniagua

Madrid

August, 2020

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
**“Analysis of an edge-computing-based solution for local data processing
at secondary substations”**

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico **2019/2020** es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos. El Proyecto no es
plagio de otro, ni total ni parcialmente y la información que ha sido tomada
de otros documentos está debidamente referenciada.



Fdo.: Néstor Rodríguez Pérez

Fecha: 11/08/2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Miguel Ángel Sanz Bobi

Fecha:/08/2020

EL CO-DIRECTOR DEL PROYECTO

Aurelio
Sanchez
Paniagua

Firmado
digitalmente por
Aurelio Sanchez
Paniagua
Fecha: 2020.08.11
13:12:41 +02'00'



Fdo.: Aurelio Sánchez Paniagua

Fecha: ..11../08/2020



UNIVERSITARY MASTER IN INDUSTRIAL ENGINEERING

FINAL MASTER THESIS

Analysis of an edge-computing-based solution for
local data processing at secondary substations

Author: Néstor Rodríguez Pérez

Director: Miguel Ángel Sanz Bobi

Co-Director: Aurelio Sánchez Paniagua

Madrid

August, 2020

ANÁLISIS DE UNA SOLUCIÓN BASADA EN “EDGE COMPUTING” PARA EL PROCESAMIENTO LOCAL DE DATOS EN CENTROS DE TRANSFORMACIÓN

Autor: Rodríguez Pérez, Néstor.

Director: Sanz Bobi, Miguel Ángel

Co-Director: Sánchez Paniagua, Aurelio

Entidad Colaboradora: i-DE (Grupo Iberdrola)

RESUMEN DEL PROYECTO

1. Introducción

El uso de “Edge computing”, siguiendo un enfoque de Internet de las cosas y microservicios, en la red de distribución, está considerado como el siguiente paso hacia una red más inteligente, ya que aliviaría la carga del sistema central y de la infraestructura de comunicaciones y, al mismo tiempo, proporcionaría capacidades de control y vigilancia en tiempo semi-real.

En Europa, el proyecto OpenNode [ASGM12] intentó implementar una solución modular de “Edge computing” para construir el Centro de Transformación "Inteligente". Más tarde, Siemens introdujo Gridlink [CeHP16], una infraestructura de comunicaciones para desplegar aplicaciones Java, probándola con una aplicación de control de tensión mediante transformadores OLTC. ABB [Abb16] sólo considera dispositivos inteligentes especializados, pero no diferentes "microservicios" en el mismo dispositivo. Sin entrar en detalles arquitectónicos, [JWHY18] analiza las aplicaciones de “Edge computing” en las redes de distribución. [CWWL19] define una arquitectura “Edge” que demuestra reducir el ancho de banda de transmisión y el retardo en las comunicaciones para el mismo número de dispositivos y [WLYC18] propone una arquitectura “fog computing” (que se basa en “Edge computing”) para la red de distribución que utiliza el protocolo MQTT y Node-RED como herramienta de programación.

En febrero de 2020, Intel y Minsait (Indra) publicaron la arquitectura de consolidación de la carga de trabajo en el borde (“Edge”) (eWLCA) [OSMC20] que incluye el uso de diferentes tecnologías de código abierto para el despliegue de microservicios en el borde. Minsait y la empresa española de distribución eléctrica i-DE (Grupo Iberdrola) están desarrollando una prueba de concepto (PoC) de “Edge computing” en la red de distribución siguiendo esta arquitectura.

Este proyecto analiza las tecnologías propuestas en el eWLCA y sus posibles alternativas, considerando los requisitos de la empresa distribuidora (operación, seguridad, etc.); sus ventajas, desafíos y posibles funcionalidades en la red de distribución de BT, y el impacto económico que tendría la implementación de esta solución en todo el sistema y en la distribuidora, a fin de determinar la conveniencia (o no) de aplicar “Edge computing” en centros de transformación (CTs) con esta arquitectura. Para ello, se utilizan dos entornos de prueba: uno local y otro remoto. El entorno local se utiliza para comparar tecnologías y para desarrollar y probar una aplicación de balance en tiempo real programada en Python, para lo que también se desarrolla un sencillo generador de datos en Node-RED. El entorno remoto se utilizará para probar el proceso de despliegue a distancia y para probar una versión modificada de la aplicación de balance para procesar informes similares a los S02 que suelen ser proporcionados por los concentradores de datos de los CTs.

1.1 Visión general de la solución

Para el análisis, se distinguen tres elementos en la solución basada en el eWLCA:

- El **Nodo “Edge”**. El hardware que se instale en el CT y su software.
- El **Sistema de Gestión** (en la nube u “on premises”). Para supervisar y gestionar el despliegue remoto de microservicios en los nodos Edge.
- Las **comunicaciones**. Tres entornos: las comunicaciones internas entre las aplicaciones del nodo; comunicaciones entre el nodo y el sistema de gestión; y las comunicaciones con otros dispositivos (fuera o dentro del CT)

2. Análisis

2.1 Ventajas, desafíos y funcionalidades

Las **ventajas** que se esperan de esta solución son numerosas. El procesamiento de datos en los CTs reduce el retraso en las comunicaciones y las decisiones, y proporciona autonomía a los CTs en caso de fallos de comunicación. El uso de tecnologías de virtualización (por ejemplo, Docker) permite aislar las aplicaciones. El despliegue de estas funcionalidades (o microservicios) se haría desde el sistema central, reduciendo, por tanto, el tiempo entre desarrollo y la producción (*metodología Agile*).

En esta arquitectura, software y hardware están dissociados, lo que aumentará la competitividad para el desarrollo de microservicios, y proporcionará flexibilidad, ya que se podrían desplegar distintos equipos con distintas funcionalidades en función del CT ("*Personalización*" de las funcionalidades). El uso de un único dispositivo para múltiples funcionalidades conducirá, inevitablemente, a la sustitución futura de dispositivos actualmente presentes en los CTs (concentradores de datos, SABTs...). En cuanto al funcionamiento, se podrían desplegar muchas funcionalidades para un mejor control y monitorización de la red de BT, lo que redundaría en una mejora final de la calidad del servicio. Además, esta solución puede ser el impulso para el desarrollo de funcionalidades innovadoras y disruptivas, y las comunicaciones actuales entre los CTs (a través de fibra óptica/PLC) podrían utilizarse para que los nodos trabajen juntos en procesos de alta carga computacional (procesamiento de imágenes, etc).

No sólo la solución analizada presentará ventajas, sino también algunos **desafíos**. Inicialmente, el principal desafío será el relacionado con el hardware. Será complejo determinar la relación adecuada entre la capacidad y el coste, así como encontrar un dispositivo en el mercado que también cumpla con los requisitos y dimensiones para ser instalado en un CT (por ejemplo, pruebas de aislamiento eléctrico, temperatura, etc.). En cuanto al funcionamiento, algunas funcionalidades disruptivas que podría soportar la solución no están aun totalmente reguladas (flexibilidad de la demanda, el almacenamiento de energía, la integración de los VEs, etc.). Además, algunas funcionalidades requieren datos de los contadores inteligentes que, actualmente, pueden estar incompletos o no estar disponibles debido a desconexiones temporales, por lo que debe mejorarse estas comunicaciones en cuanto a fiabilidad y velocidad para aprovechar al máximo la solución.

En cuanto a las posibles **funcionalidades** de la solución, éstas cubren diferentes áreas. **Conocimiento y análisis de la red**: algoritmos de conectividad de fase, cálculo de la impedancia de las líneas de BT, algoritmos de detección de fraude y contabilidad de las pérdidas de energía. **Monitorización de la red**: detección, localización y clasificación de faltas en BT y estimación del estado de la red de BT. **Gestión y mantenimiento de**

activos: aplicaciones de mantenimiento predictivo, funcionalidades de seguridad (cámaras térmicas, etc.) y predicción de la demanda del CT. **Flexibilidad y funcionalidades innovadoras:** supervisión y control de la generación distribuida y del almacenamiento de energía, integración del VE y flexibilidad de la demanda. Este tipo de funcionalidades todavía necesita una fuerte regulación. Con toda probabilidad, esta lista de funcionalidades aumentará cuando la solución se pruebe en campo con éxito.

2.2 Análisis del Nodo Edge

a) Tecnología de Virtualización

Se analizan tres tecnologías de virtualización: Máquinas Virtuales (MVs), Contenedores (i.e. Docker) y Unikernels (Figura 1)

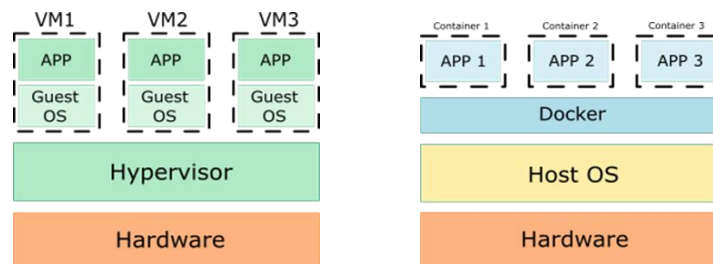


Figura 1. General Virtual Machine (left) and containerization (right) architectures

El despliegue de funcionalidades utilizando exclusivamente **máquinas virtuales (MVs)** estándar (un sistema operativo completo por cada máquina virtual, Figura 1) sería extremadamente difícil a pesar de su madurez y del alto nivel de seguridad y aislamiento proporcionado. Las imágenes resultantes son muy grandes (GB) y requieren mucho tiempo de desarrollo. Además, como se espera que se desplieguen múltiples aplicaciones, el consumo de recursos en el dispositivo sería altamente ineficiente [CDLR19] y el mantenimiento del SO requeriría más de una actualización por nodo.

El uso de **contenedores** (Docker) presenta mejores características en los casos en los que las máquinas virtuales carecen de ellas. Las aplicaciones comparten el mismo sistema operativo del host (Figura 1) y son más ligeras (MBs). También muestra un buen nivel de seguridad desde el punto de vista local (cuando no hay intermediarios externos) [MRCD18]. En cuanto a la seguridad de la propiedad intelectual, las imágenes de los dockers tendrían que construirse con un usuario no root, y con su código fuente ofuscado. También existen herramientas comerciales para gestionar los permisos de los usuarios (por ejemplo, Docker Enterprise). Docker es muy popular entre los desarrolladores porque es relativamente sencillo, de código abierto (versión comunitaria gratuita) y reduce el tiempo entre desarrollo y producción.

Finalmente, en la última década, los **Unikernels** han surgido como alternativa. Los Unikernels son similares a las MVs, pero con un *kernel* optimizado en lugar de un SO completo. Las imágenes son incluso más ligeras que los contenedores [Luci17] pero más difíciles de depurar. Las pruebas realizadas por [GSAV18] muestran que los Unikernels podrían lograr un mejor rendimiento que los contenedores. Su mayor inconveniente es que esta tecnología se encuentra en una fase temprana para su uso en producción [MRCD18] y todavía es demasiado compleja para el desarrollador medio [Eybe19].

Por lo tanto, en base a este análisis, la tecnología recomendada para utilizar en el Nodo Edge es el **uso de contenedores mediante Docker**. Docker es también la tecnología definida en el eWLCA y la utilizada por Minsait para su PoC con i-DE.

b) Sistema Operativo (SO)

Docker necesita un *kernel* de linux para ser ejecutado. Por lo tanto, el uso de Windows Server se descarta. Entre los sistemas operativos Linux disponibles, se recomienda el uso de uno ya homologado por la distribuidora (RedHat o RedHat Oracle Linux en el caso de i-DE) para que el mantenimiento pueda ser llevado a cabo por la distribuidora de un modo sencillo.

c) Base de Datos

Las distribuidoras españolas están trabajando conjuntamente (FutuRed) para definir un esquema de datos común en JSON basado en la Arquitectura de la Red de Cosas (WOT-A), patrocinado por el W3C [KMLK20]. Por lo tanto, la base de datos que se utilice debe ser apropiada para trabajar con documentos JSON.

El uso de una base de datos SQL se descarta, ya que existen bases No-SQL optimizadas para almacenar documentos como JSON y una base de datos relacional suele requerir más tiempo en las consultas de datos para aplicaciones en tiempo semi-real [CeME15].

Entre los tipos de bases de datos no relacionales (No-SQL), las orientadas a documentos (MongoDB) y los de series temporales (InfluxDB) son las más apropiadas. Tanto MongoDB como InfluxDB se probaron en el entorno de pruebas local para comparar su funcionamiento. En este entorno, MongoDB consume 6 veces más CPU y tiene 4 veces más procesos abiertos que InfluxDB, aunque ambos muestran un consumo de memoria similar (Figura 2). Además, MongoDB necesita al menos 3 réplicas para garantizar disponibilidad (modelo CAP).

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	BLOCK I/O	PIDS
ed85c328f79d	influxdb	0.09%	72.48MiB / 2.87GiB	2.47%	63.8MB / 1.25MB	10
8108f762bae6	mongodb	0.61%	73.2MiB / 2.87GiB	2.49%	57.5MB / 1.2MB	37

Figura 2. Captura de pantalla de las estadísticas de Docker: consumo de InfluxDB y MongoDB como contenedores

Como la capacidad de computación del Nodo Edge es muy limitada, se recomienda utilizar InfluxDB como base de datos: muestra un buen rendimiento al trabajar con sensores y para el análisis de datos (soporte de lenguajes R y Python) [NaAb19], tiene un lenguaje de consulta tipo SQL, es compatible con JSON y está diseñado específicamente para métricas y eventos, que son el tipo de información que el Nodo Edge procesará. InfluxDB es también la base de datos definida en el eWLCA y la utilizada por Minsait para su PoC con i-DE.

d) Hardware

Algunas plataformas Edge requieren su propio hardware, mientras que otras son más flexibles y pueden utilizar uno de terceros. Independientemente de esto, el dispositivo debe satisfacer los requisitos y pruebas para ser instalado en un CT. Específicamente, se deben pasar hasta 29 pruebas, clasificadas en 6 grupos: *aislamiento*, *perturbaciones radioeléctricas*, *inmunidad eléctrica*, *mecánica* y *climática*. Estas pruebas deben ser certificadas por un laboratorio. Además, las dimensiones del aparato no deben exceder de 220x140x130 mm (ancho x alto x fondo) para poder ser incluido en los armarios eléctricos actualmente desplegados por i-DE.

2.3 Comunicaciones

Los dos protocolos IoT considerados para este análisis son MQTT(v3.1, v5.0 es demasiado reciente) [Oasi19] y AMQP (v1.0) [Oasi12] ya que son los más populares en

el ámbito IoT. Se basan en un sistema publicación/suscripción y ambos son protocolos abiertos, aunque difieren en su funcionamiento y otras características. Como se mencionó en 1.1, en la solución se distinguen tres entornos de comunicaciones con diferentes requisitos principales.

a) Comunicaciones internas

Para proporcionar datos a los microservicios y permitir las posibles comunicaciones entre ellos, se necesita un bus de comunicaciones interno. Los principales requisitos son que sea **ligero** y **fiable**. Para comparar MQTT y AMQP en estos aspectos, se realizó una prueba con brokers "dockerizados" de MQTT y AMQP publicando cuatro mensajes JSON ficticios simultáneamente en cuatro temas/colas diferentes cada cinco segundos. Todos los mensajes fueron recibidos correctamente por los suscriptores. Las estadísticas de rendimiento se muestran en la Figura 3.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
6290cdd07982	edge_amqp	0.57%	87.36MiB / 2.87GiB	2.97%	91.9kB / 40.3kB	41.5MB / 602kB	86
642db35d3a29	edge.mqtt	0.06%	916KiB / 2.87GiB	0.03%	22.7kB / 6.04kB	3.37MB / 0B	1

Figura 3. Captura de pantalla de los resultados de la prueba de comparación MQTT-AMQP.

Observando la Figura 3, MQTT es notablemente más ligero que AMQP. Además, MQTT proporciona tres niveles de calidad de servicio (es decir, fiabilidad) frente a los dos niveles proporcionados por AMQP. Por lo tanto, se recomienda el uso de **MQTT** para las comunicaciones internas. MQTT es también el que utiliza Minsait para su PoC con i-DE.

b) Comunicaciones con el Sistema de gestión

En este caso, el principal aspecto a considerar es la **seguridad** del protocolo. Ambos protocolos proporcionan seguridad TLS/SSL pero AMQP tiene, además, compatibilidad con SASL (autenticación). El aumento de la carga en los mensajes no debería ser un problema, ya que la solución reduciría la cantidad de datos que deben enviarse al sistema central para su procesamiento, por lo que la actual infraestructura de comunicaciones debería ser suficiente. Además, AMQP proporciona más control sobre las colas y tiene un tipo de función de solicitud/respuesta (Remote Procedure Call) para ejecutar funciones bajo demanda. Por lo tanto, se recomienda el uso de **AMQP** para estas comunicaciones. El PoC de Minsait con i-DE utiliza MQTT+TLS.

c) Comunicaciones con otros dispositivos

Inicialmente, será necesario el desarrollo de **adaptadores de protocolo** para comunicarse con los diferentes dispositivos presentes actualmente en un CT (Figura 4). En el caso de los nuevos dispositivos (por ejemplo, HEMS), MQTT es el más utilizado por los investigadores y es tan ligero que podría desplegarse en el Nodo Edge un agente MQTT adicional para estas comunicaciones. Además, como se trata de un protocolo abierto, los fabricantes pueden producir fácilmente dispositivos compatibles.

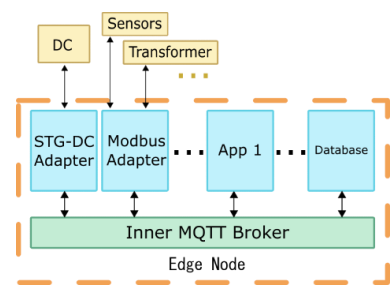


Figura 4. Integración del Nodo Edge con los protocolos de comunicación ya en uso.

2.4 Sistema de Gestión

Hay decenas de plataformas Edge en el mercado que tienen diferentes propósitos y características. De acuerdo con MachNation [Toka17], los principales aspectos que hay que tener en cuenta son: la **compatibilidad de los protocolos** de comunicación, el nivel

de **autonomía** alcanzable, el alojamiento (nube u “on premises”), la **dependencia del hardware** y las **capacidades de visualización**.

En cuanto a la orquestación de la carga de trabajo, Docker Compose es la herramienta utilizada. La Figura 5 muestra el proceso de despliegue que se utiliza y se prueba en la prueba #2.

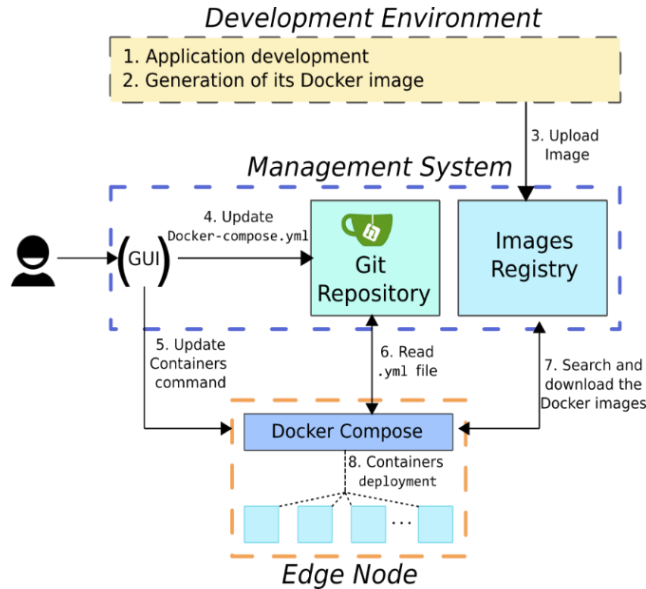


Figura 5. Diagrama simplificado del proceso de despliegue de una aplicación. Fuente del icono: <https://icons8.com>

2.5 Impacto Económico

La solución analizada sería clasificada como una inversión de tipo 2 por el organismo regulador (relacionada con la red inteligente). Se asume un coste del dispositivo de ~450€ por unidad y una vida útil reglamentaria de 12 años (equipo de Smart Grids). Los ahorros inmediatos estarían relacionados con la reducción del **Time To Market** (TTM) de las funcionalidades (Tabla 1).

Tabla 1. Resumen de las diferencias entre los dos enfoques para el desarrollo de una nueva funcionalidad

	Enfoque Tradicional (Hardware+software)	Nuevo Enfoque (Nodo Edge) (Solo software)
Tiempo medio de desarrollo	3 años	4 meses
Num. Proveedores necesarios	3	1
Competencia para los desarrollos	Baja	Muy Alta
Coste de material	✓	✗
Coste de servicio	✓	✗
Coste de la aplicación	✓	✓

El coste equivalente de desplegar sólo una nueva funcionalidad se reduciría en un 87,5% con respecto al enfoque actual. La futura sustitución de dispositivos actuales por su versión "en contenedor" reducirá los correspondientes costes de mantenimiento. Además, las funcionalidades de **mantenimiento predictivo** en el nodo podrían reducir el coste de mantenimiento en un 12% y aumentar la vida útil de los activos en un 20% [HKDM18], lo que se retribuiría en un $\geq 30\%$ adicional a la retribución, por activo, por operación y mantenimiento.

Considerando que, hoy en día, el algoritmo de conectividad desarrollado por Ariadna Grid se ejecuta en el sistema central, si se desplegara en el Nodo Edge el ahorro estimado en **computación y almacenamiento central, y comunicaciones**, supondría ~15.000€ y ~280.000€ (i-DE) anuales, respectivamente.

Hoy en día, el valor anual estimado de las **pérdidas técnicas y no técnicas** para i-DE es de 364,62 millones de euros. Algunas de las funcionalidades que podrían desplegarse en el Nodo Edge contribuirían a la reducción de las pérdidas (por ejemplo, detección de fraudes, algoritmo de conectividad, etc.). La Tabla 2 muestra un resumen de los ahorros calculados en este aspecto.

Tabla 2.. Resumen de los ahorros en pérdidas de energía

Ahorros por dispositivo sustituido/evitado y año (despliegue en el 25% de los CTs de i-DE)	100,000 €
Detección de fraude (cualitativamente)	Mayor rapidez en la detección
	Menos "falsos positivos" => Menos inspecciones
	Nuevos tipos de fraude detectados
Ahorros por año debido al equilibrado de fases (escenario del 25%)	≥ 5,500,000 €

3. Pruebas

3.1 Prueba #1: Balance en tiempo real en entorno local de pruebas

Se desarrolla un generador de datos aleatorios para proporcionar la funcionalidad de “en tiempo real”. La aplicación de balance, programada en Python, se conecta al bróker MQTT interno para recibir los datos, en JSON, cada 10s (consumo de potencia activa registrado por un supervisor y tres contadores), luego calcula el balance de potencia activa (independientemente del número de dispositivos) y publica el resultado en un tema MQTT. Node-RED se utiliza entonces para comprobar el correcto funcionamiento de la aplicación y para almacenar los resultados en la base de datos InfluxDB.

Esta prueba muestra las posibilidades que la solución puede proporcionar. La aplicación desarrollada puede adaptarse fácilmente a un entorno real, funciona en tiempo real con MQTT sin necesidad de consultar la base de datos o cualquier otro intermediario y funciona de forma aislada (es decir, un error en la aplicación no afectaría a los demás contenedores). Además, los datos de entrada y el resultado del balance pueden visualizarse en tiempo real utilizando Grafana (Figura 6) o Chronograph, sin necesidad de almacenar los datos en el sistema central.

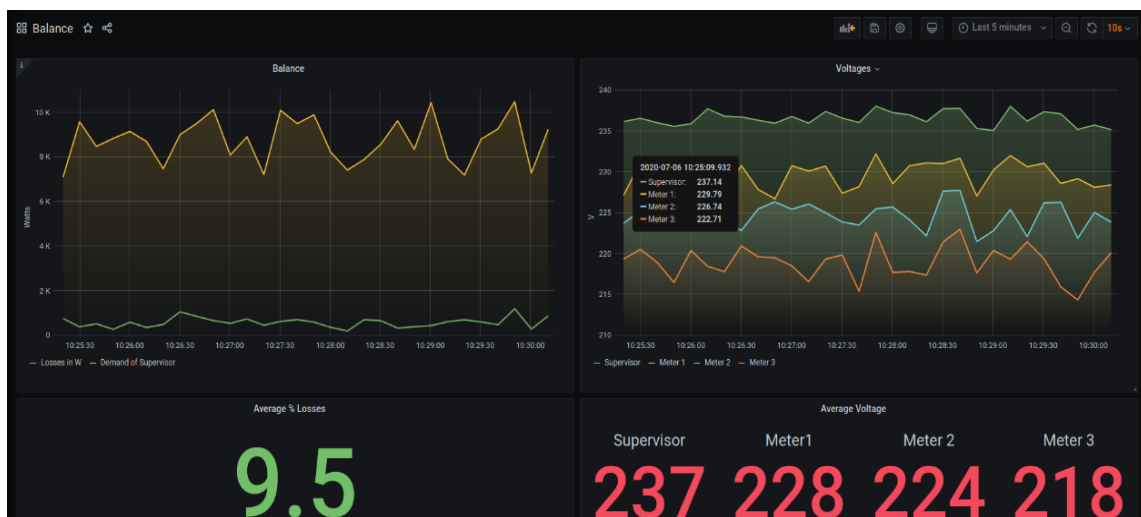


Figura 6. Panel Grafana para visualizar los datos y los resultados del balance (media de pérdidas = 9,5% y las pérdidas absolutas)

3.2 Prueba #2: Balance en tiempo real en el entorno remoto de pruebas usando informes tipo S02.

En esta prueba se comprobó el proceso de despliegue que se muestra en la Figura 5 con una versión modificada de la aplicación de balance de la prueba #1 para leer los mensajes JSON que siguen una estructura similar a la de S02 (S02 son los informes de "incremento diario" definidos por la especificación STG-DC). El entorno de prueba remoto es proporcionado por Minsait para i-DE, por lo que todo el proceso de despliegue se hace de manera similar a como se haría en un despliegue de funcionalidad real en un CT.

El correcto despliegue y funcionamiento de la aplicación se muestra en la Figura 7. En esta fase inicial de la PoC, los pasos 2, 3 y 4 del proceso (Figura 5) pueden convertirse en una fuente de errores durante el despliegue, principalmente debido a la complejidad de los archivos de configuración involucrados y al uso de la línea de comandos, que no es muy amigable de usar.



Figura 7. Captura de pantalla de Node-RED para comprobar el correcto funcionamiento del balance en este entorno

4. Conclusiones

Este proyecto ha analizado las tecnologías (y sus alternativas) que intervienen en una solución basada en "Edge computing" que sigue el eWLCA [OSMC20]; las ventajas, desventajas y funcionalidades que esta solución podría tener en la distribución en BT y su impacto económico en el sistema y en la distribuidora (i-DE), cumpliendo los objetivos inicialmente establecidos. Además, la solución se ha probado parcialmente utilizando dos entornos de prueba diferentes (local y remoto) mediante el desarrollo de una aplicación de balance energético que funciona en tiempo real.

La conclusión que puede extraerse de este proyecto es que la aplicación de "Edge computing" a nivel de centro de transformación tiene el potencial para ser el nuevo paradigma de cómo se monitorea y controla la red de BT, proporcionando grandes beneficios a la empresa distribuidora, al sistema y a la industria eléctrica en general. Las tecnologías utilizadas en la arquitectura analizada se consideran apropiadas. Su modularidad y el hecho de ser de código abierto proporcionará gran flexibilidad a la distribuidora eléctrica y promoverá los desarrollos en la universidad y en la industria, aunque el proceso de despliegue probado debería mejorar la "amigabilidad" de uso y la capacidad de realizar cambios de configuración en línea y despliegues de nuevas aplicaciones. No obstante, estos aspectos, junto con el desafío del hardware, están cerca de solucionarse en las futuras fases de la prueba de concepto llevada a cabo en i-DE.

ANALYSIS OF AN EDGE-COMPUTING-BASED SOLUTION FOR LOCAL DATA PROCESSING AT SECONDARY SUBSTATIONS

Author: Rodríguez Pérez, Néstor.

Supervisor: Sanz Bobi, Miguel Ángel.

Co-Supervisor: Sánchez Paniagua, Aurelio

Collaborating Entity: i-DE (Iberdrola Group)

SUMMARY OF THE PROJECT

1. Introduction

The implementation of edge computing, following an IoT and microservices approach, in the distribution grid, is considered to be the next step towards a smarter grid, since it could alleviate the load of the central system and the communications infrastructure and, at the same time, provide semi-real time control and monitoring capabilities.

In Europe, the OpenNode project [ASGM12] tried to implement a modular edge computing solution to achieve the “Smart” Secondary Substation. Later, Siemens introduced Gridlink [CeHP16], a communications infrastructure to deploy Java applications, testing it with a voltage control application using OLTC transformers. ABB [Abb16] only considers specialized smart devices, but not different modular applications in the same device. Without entering into architectural details, [JWHY18] discusses the applications of edge computing in distribution networks. [CWVL19] defines an edge computing architecture that proves to reduce the bandwidth and delay in communications for the same number of devices and [WLYC18] proposes a fog-computing-based architecture (which is based on edge computing) for the distribution grid that uses the MQTT protocol and Node-RED as programming tool.

On February 2020, Intel and Minsait (Indra) published the open Edge Workload Consolidation Architecture (eWLCA) [OSMC20] that specifies the use of different open source technologies for the deployment of microservices in the edge. Minsait and the Spanish electric distribution utility i-DE (Iberdrola Group) are developing a proof of concept (PoC) of edge computing at the distribution grid following this architecture.

This project analyses the technologies proposed in the eWLCA and their possible alternatives, considering the requirements of the distribution utility (e.g. operation, security, etc.); its advantages, challenges, and possible functionalities in the LV distribution grid, and the economic impact that the implementation of this solution would have on the system and on the utility, in order to determine the convenience (or not) of applying edge computing at secondary substations (SSs) with this architecture. For this, two tests environments are used: one local and one remote. The local environment is used to compare technologies and to develop and test a real-time balance application programmed in Python, for what a simple data generator in Node-RED is also developed. The remote environment is used to test the remote deployment process and to test a modified version of the balance application to process S02-like reports that are commonly provided by data concentrators at SSs.

1.1 Overview of the solution

For the analysis, three elements are distinguished in the solution based on the eWLCA:

- The **Edge Node**. The hardware installed in the SS and its software architecture.

- The **Management System** (on cloud or on premises). To monitor and manage the deployment of microservices (i.e. applications, functionalities) on the Edge nodes.
- **Communications**. Three environments: inner communications between applications in the node; communications between the node and the management system; and communications with other devices (in or out of the SS).

2. Analysis

2.1 Advantages, challenges and functionalities

The **advantages** that are expected from this solution are numerous. Data processing at SSs reduces the delay in communications and decisions and provides autonomy to the SS in case of communication failures. The use of virtualization technologies (e.g. Docker) allows the isolation between applications. The functionalities (i.e. microservices) deployment would be done from the central system and hence reducing the time between development and production (*Agile methodology*).

In this architecture, software is decoupled from hardware, which will significantly increase the competitiveness for the development of microservices, and will provide flexibility, since different hardware with different functionalities could be deployed based on the type of SS (*“Customization” of functionalities*). The use of a unique device for multiple functionalities (i.e. microservices) will inevitably lead to the future substitution of devices currently present in SSs (e.g. data concentrators, low voltage supervisors...). In terms of operation, many functionalities could be deployed for a better control and monitor of the LV grid, resulting in an ultimate improvement of the quality of service. Besides, this solution can be the driver for innovative and disruptive functionalities, and current communications between SSs (through optical fiber/PLC) could be used to make the Edge Nodes work together for high computing processes (e.g. image processing, etc.).

Not only the analyzed solution will present advantages but also some **challenges** to be faced. Initially, the main challenge will be hardware related. It will be complex to determine the adequate relation between computing capacity and cost, as well as to find a device in the market that also complies with the requirements and dimensions to be installed in SSs (e.g. electric isolation tests, temperature, etc.). In terms of operation, some disruptive functionalities that this solution could support are not fully regulated yet (e.g. demand response, energy storage, EV integration, etc.). Furthermore, some functionalities require data from smart meters which, currently, might be incomplete, or unavailable due to temporary disconnections, so reliability and speed of these communications should be improved to take full advantage of the solution.

Regarding the **functionalities** of the edge-computing-based solution, they cover different areas. **Grid knowledge and analysis:** phase connectivity algorithms, calculus of the impedance of LV lines, fraud detection algorithms and power losses accountability. **Grid monitoring:** detection, location and classification of LV faults, and state estimation of the LV grid. **Asset management and maintenance:** predictive maintenance applications, security functionalities (e.g. thermal cameras, movement detectors, etc.) and demand prediction for the SS. **Flexibility and innovative functionalities:** monitoring and control of distributed generation and energy storage, EV integration and demand response schemes. These types of functionalities still need strong regulation. In all likelihood, this overall list of functionalities will grow significantly when the solution gets tested on field successfully.

2.2 Analysis of the Edge Node

a) Virtualization Technology

Three virtualization technologies are analyzed: Virtual Machines (VMs), Containerization (i.e. Docker) and Unikernels (Figure 1)

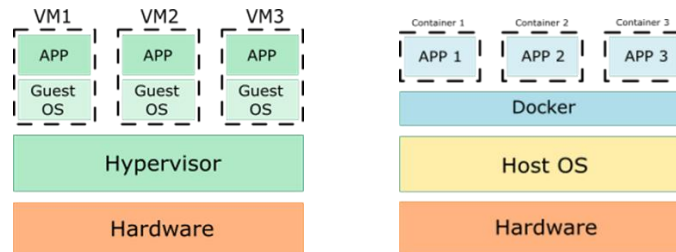


Figure 1. General Virtual Machine (left) and containerization (right) architectures

The deployment of functionalities using exclusively **standard VMs** (a complete OS per VM, Figure 1) would be extremely challenging despite its maturity and the high level of security and isolation provided. The resulting images are very large (GBs) and have long development times. Besides, as multiple applications are expected to be deployed, the resources consumption in the device would be highly inefficient [CDLR19] and the maintenance of the OS would require more than one update per node.

Containerization (Docker) presents better characteristics where VMs lack. Applications share the same host OS (Figure 1) and are lighter (MBs). It also shows good security level from a local point of view (when there are no external intermediaries) [MRC18]. In terms of intellectual property security, docker images should be built with a non-root user, and with their source code obfuscated. There are also commercial tools to manage user's permissions (e.g. Docker Enterprise). Docker is very popular among developers because it is relatively simple, open source (free community version) and reduces time between development and production.

Finally, in the last decade **Unikernels** have emerged as an alternative. They are similar to VMs but with an optimized kernel instead of an entire OS. Images are even lighter than containers [Luci17] but more difficult to debug. Tests carried out by [GSAV18] showed that Unikernels could achieve better performance than containers. The biggest drawback is that this technology is at an early stage of development for production environments [MRC18] and is still too complex for the average developer [Eybe19].

Therefore, based on this discussion, the recommended technology to be used in the Edge Node is **containerization using Docker**. Docker is also the technology defined in the eWLCA and the one used by Minsait for its PoC with i-DE.

b) Operating System (OS)

Docker needs a linux kernel in order to be executed. Therefore, Windows Server is discarded. Among the Linux OS available, it is recommended the use of one already homologated by the utility (RedHat or RedHat Oracle Linux in the case of i-DE) so that the maintenance can be carried out by the utility easily.

c) Database

The Spanish utilities are working together (FutuRed) to define a common JSON schema based on the Web of Things Architecture (WOT-A), sponsored by W3C [KMLK20]. Therefore, the database used must be appropriate to work with JSON documents.

The use of a relational (SQL) database is discarded since there are No-SQL databases optimized to store documents like JSON and a relational database usually requires more time to query data for semi-real time applications [CeME15].

Among the non-relational (No-SQL) database types, the document-oriented (MongoDB) and the time series (InfluxDB) types are the most appropriate. Both MongoDB and InfluxDB were tested on the local test environment to compare their functioning. In this environment, MongoDB consumes 6 times more CPU and have 4 times more open processes than InfluxDB, although both show similar memory consumption (Figure 2). In addition to this, MongoDB needs at least 3 replicas to have availability (CAP model).

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	BLOCK I/O	PIDS
ed85c328f79d	influxdb	0.09%	72.48MiB / 2.87GiB	2.47%	63.8MB / 1.25MB	10
8108f762bae6	mongodb	0.61%	73.2MiB / 2.87GiB	2.49%	57.5MB / 1.2MB	37

Figure 2. Screenshot of Docker stats showing the consumption of InfluxDB and MongoDB in Docker containers.

As the computing capacity of the Edge Node is very limited, **InfluxDB** is recommended to be used as database. It shows good performance when working with sensors and for data analytics (it supports R and Python) [NaAb19], it has a SQL-like query language, it is compatible with JSON and it is specifically designed for metrics and events, which are the type of data that the Edge Node will process. InfluxDB is also the database defined in the eWLCA and the one used by Minsait for its PoC with i-DE.

d) Hardware

Some Edge platforms require their own specific device whereas other are more flexible and can use a third-party device. Regardless of this, the device must pass the requirements and tests to be installed in a SS. Specifically, up to 29 tests have to be passed, classified in 6 groups: *insulation*, *radioelectric disturbances*, *immunity*, *electrical*, *mechanical*, and *climatic*. These tests must be certificated by a laboratory. Besides, the dimensions of the device should not exceed 220x140x130 mm (width x height x depth) in order to be included in the electric cabinets currently deployed by i-DE.

2.3 Communications

The two IoT protocols considered for this analysis are MQTT(v3.1, v5.0 is too recent) [Oasi19] and AMQP (v1.0) [Oasi12] since they are the most popular in IoT. They are based on publish/subscribe and both of them are open protocols, although they differ in functioning and other characteristics. As mentioned in 1.1, three communications environments with different main requirements are distinguished in the solution.

a) Inner communications

In order to provide input data to the microservices and to allow the possible communications between them, an inner communications bus is needed. The main requirements are **lightness** and **reliability**. In order to compare MQTT and AMQP in these aspects, a test was carried out using “dockerized” MQTT and AMQP brokers by publishing four dummy JSON messages simultaneously on four different topics/queues every five seconds. All the messages were correctly received by the subscribers. Performance results are shown in Figure 3.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
6290cdd07982	edge_amqp	0.57%	87.36MiB / 2.87GiB	2.97%	91.9kB / 40.3kB	41.5MB / 602kB	86
642db35d3a29	edge_mqtt	0.06%	916KiB / 2.87GiB	0.03%	22.7kB / 6.04kB	3.37MB / 0B	1

Figure 3. Screenshot of the MQTT-AMQP comparison test results.

Observing Figure 3, MQTT is remarkably lighter than AMQP. Besides, MQTT provides three levels of quality of service (i.e. reliability) versus the two levels provided by AMQP. Therefore, the use of **MQTT** is recommended for the inner communications. MQTT is also the one used by Minsait for its PoC with i-DE.

b) Communications with the Management System

In this case, the main aspect to consider is the **security** of the protocol. Both protocols provide TLS/SSL security but AMQP additionally has compatibility with SASL (authentication). The increase in the messages load should not be a problem since the solution would reduce the amount of data that has to be sent to the central system for processing, so the current communications infrastructure should be enough. Furthermore, AMQP provides more control over queues and has a sort of request/response feature (Remote Procedure Call) to execute functions on demand. Therefore, the use of **AMQP** is recommended for these communications. The PoC of Minsait with i-DE uses MQTT+TLS.

c) Communications with other devices

Initially, it will be necessary the development of **protocol adapters** to communicate with the different devices currently present in a SS (Figure 4). For new devices (e.g. HEMS), **MQTT** is the most used by researchers and is so light that an additional MQTT broker for these communications could be deployed on the Edge Node. Furthermore, as it is an open protocol, manufacturers can easily produce compatible devices.

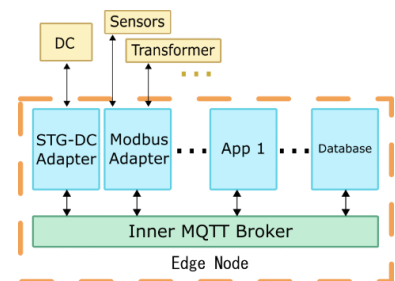


Figure 4. Integration of the Edge Node with already in-use communication protocols.

2.4 Management System

There are tens of Edge platforms in the market that have different purposes and characteristics. According to MachNation [Toka17], the main aspects to consider are: communications **protocols compatibility**, level of **autonomy** achievable, the **hosting** (on cloud or on premises), **hardware dependence** and **visualization capabilities**.

Docker compose is the tool used for workload orchestration. Figure 5 shows the development and deployment process that is used and tested in test #2.

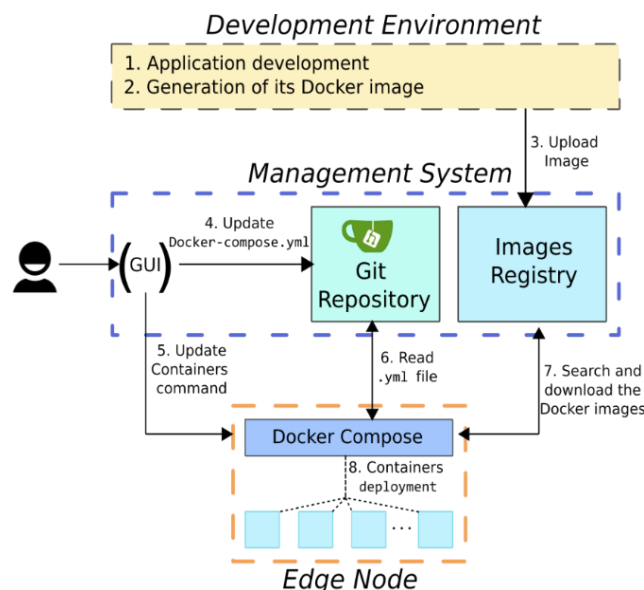


Figure 5. Simplified diagram of the application deployment process. Icon source: <https://icons8.com>

2.5 Economic Impact

The analyzed solution would be categorized as a Type 2 investment by the regulatory body (smart grid related). The device is assumed to cost ~450€ per unit and to have a regulatory life of 12 years (Smart Grids equipment). The immediate savings would be related with the reduction in **Time To Market** (TTM) of the functionalities (Table 1).

Table 1. Summary of the differences between the two approaches for the development of a new functionality

	Traditional Approach (Hardware+software)	New Approach (Edge Node) (Only software)
Average dev. time	3 years	4 months
No. Providers needed	3	1
Competitiveness for developments	Low	Very high
Cost of material	✓	✗
Cost of service	✓	✗
Cost of application	✓	✓

The equivalent cost of just deploying one new functionality would be reduced in an 87.5% with respect to the current approach. The future substitution of current devices by their “containerized” version will reduce the corresponding maintenance costs. Furthermore, **predictive maintenance** functionalities in the node could reduce maintenance cost in 12% and increase the lifetime of assets by 20% [HKDM18], which would suppose an additional $\geq 30\%$ of the retribution, per asset, for O&M.

Considering that, nowadays, the connectivity algorithm developed by Ariadna Grid is executed at the central system, if it was deployed in the Edge Node the estimated savings in **central computing and storage, and communications** would account for ~15,000€ and ~280,000€ (i-DE) per year, respectively.

Nowadays, the estimated annual value of **technical and non-technical losses** for i-DE are 364.62 M€. Some of the functionalities that could be deployed in the Edge Node would contribute to the reduction of losses (e.g. fraud detection, connectivity algorithm, etc.). Table 2 shows a summary of the calculated savings in this aspect.

Table 2. Summary of energy losses savings

Energy Losses	
Savings per device substituted/avoided and year (considering 25% of i-DE's SSs)	100,000 €
Fraud detection (qualitative)	Faster detection
	Less inspections
	New frauds detected
Savings per year for phase balancing (25% scenario)	$\geq 5,500,000$ €

3. Tests

3.1 Test #1: Real-time balance in local test environment

A random data generator is developed to provide the real-time characteristic. The balance application, written in python, connects to a topic of the inner MQTT broker to receive the data, in JSON, every 10s (active power demand registered by a supervisor and three meters), then it calculates the active power balance (independently of the number of

devices) and publishes the result on a MQTT topic. Node-RED is then used to check the correct functioning of the application and to store the results in the InfluxDB database.

This test shows the possibilities that the solution could provide. The application can be easily adapted to a real environment, it works on real-time with MQTT without the need of querying the database or any other intermediary and it works in an isolated way (i.e. an error in the application would not affect the other containers). Furthermore, the input data and the result of the balance can be visualized in real time by using a Grafana (Figure 6) or Chronograph, without the need of storing the data in the central system.

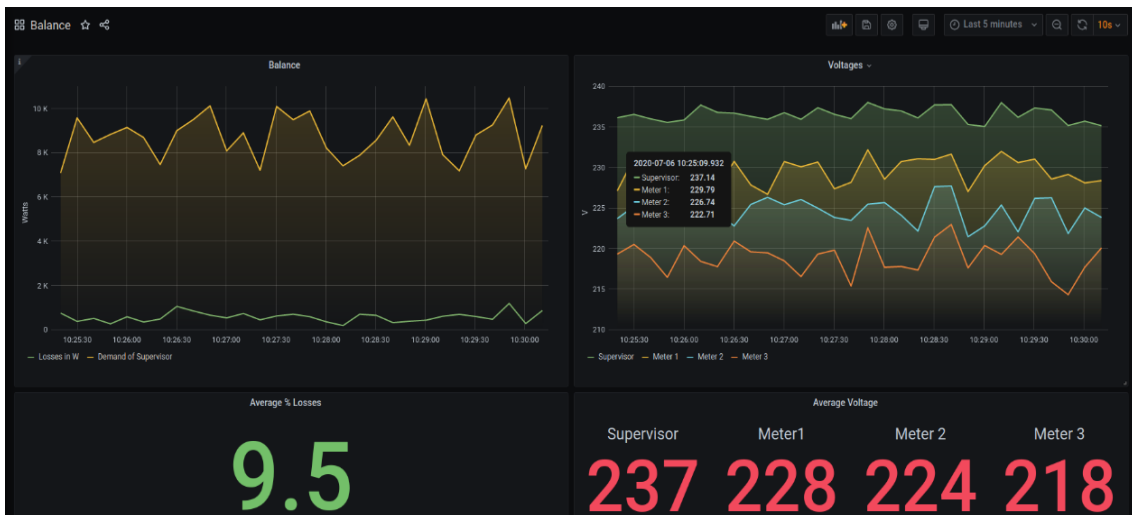


Figure 6. Grafana dashboard to visualize the data and the results of the balance (9.5% average losses and absolute losses)

3.2 Test #2: Real-time balance in remote test environment using S02-like reports.

This test checked the deployment process shown in Figure 5 with a modified version of the balance application from test #1 to read JSON messages that follow an S02-like structure (S02 are “Daily incremental” reports defined by the STG-DC specification). The remote test environment is provided by Minsait for i-DE, so all the deployment process is done similarly to how it would be done in a real functionality deployment in a SS.

The correct deployment and functioning of the application is shown in Figure 7. At this initial phase of the PoC, steps 2, 3 and 4 of the process (Figure 5) can become a source of errors during deployment, mainly because the complexity of the configuration files involved and because of the use of the command line tool, which is not very user-friendly.

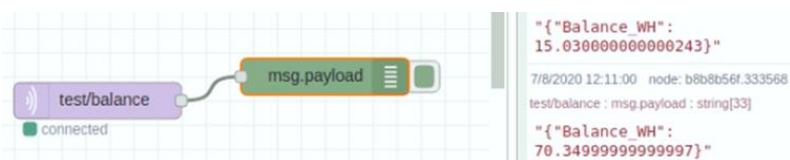


Figure 7. Node-RED screenshot to check the functioning of the balance application in the remote environment

4. Conclusions

This project has analyzed the technologies (and their alternatives) involved in an edge-computing-based solution that follows the eWLCA [OSMC20]; the advantages, disadvantages, and functionalities that this solution could have in the LV distribution and its economic impact on the system and on the utility (i-DE), fulfilling the objectives initially set. Furthermore, the solution has been partially tested using two different test

environments (local and remote) by developing an active energy balance application that works in real time.

The conclusion that can be extracted from this project is that the application of edge computing at the secondary substation level has the potential to be the new paradigm of how the LV grid is monitored and controlled, providing great benefits to the utility, to the system, and to the electric industry in general. The technologies used in the analyzed architecture are found to be the appropriate. Its modularity and the fact of being open-source-based will provide great flexibility to the utility and promote developments by academics and industry, although the deployment process tested should improve on user friendliness and on the capacity of making online configuration changes and new application deployments. However, these aspects, together with the hardware challenge, are close to be solved in future phases of the PoC developed at i-DE.

References

- [Abb16] ABB: Technical Guide: "Smart Grids 2. The "smart" secondary substation" (2016)
- [ASGM12] ALBERTO, MARTA ; SORIANO, RAÚL ; GÖTZ, JÜRGEN ; MOSSHAMMER, RALF ; ESPEJO, NICOLÁS ; LEMÉNAGER, FLORENT ; BACHILLER, RAÚL: "OpenNode: A smart secondary substation node and its integration in a distribution grid of the future". In: *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2012, S. 1277–1284
- [CDLR19] CAPROLU, MAURANTONIO ; DI PIETRO, ROBERTO ; LOMBARDI, FLAVIO ; RAPONI, SIMONE: "Edge Computing Perspectives: Architectures, Technologies, and Open Security Issues". In: *2019 IEEE International Conference on Edge Computing (EDGE)*. Milan, Italy : IEEE, 2019 — ISBN 978-1-72812-708-8, S. 116–123
- [CeHP16] CEJKA, STEPHAN ; HANZLIK, ALEXANDER ; PLANK, ANDREAS: "A framework for communication and provisioning in an intelligent secondary substation". In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, S. 1–5
- [CeME15] CEJKA, STEPHAN ; MOSSHAMMER, RALF ; EINFALT, ALFRED: "Java embedded storage for time series and meta data in Smart Grids". In: *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2015, S. 434–439
- [CWWL19] CHEN, SONGLIN ; WEN, HONG ; WU, JINSONG ; LEI, WENXIN ; HOU, WENJING ; LIU, WENJIE ; XU, AIDONG ; JIANG, YIXIN: "Internet of Things Based Smart Grids Supported by Intelligent Edge Computing". In: *IEEE Access* Bd. 7 (2019), S. 74089–74102
- [Eybe19] EYBERG, IAN: "Introduction To Unikernels". URL <https://nordicapis.com/introduction-to-unikernels/>. - accessed on 2020-06-03. — Nordic APIs
- [GSAV18] GOETHALS, TOM ; SEBRECHTS, MERLIJN ; ATREY, ANKITA ; VOLCKAERT, BRUNO ; DE TURCK, FILIP: "Unikernels vs Containers: An In-Depth Benchmarking Study in the Context of Microservice Applications". In: *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*, 2018, S. 1–8
- [HKDM18] HAARMAN, MARK ; DE KLERK, PIETER ; DECAIGNY, PETER ; MULDER, MICHEL ; VASSILIADIS, COSTAS ; SIJTSEMA, HEDWICH ; GALLO, IVAN: "Predictive Maintenance - Beyond the hype: PdM 4.0 delivers results" : PWC and Mainnovation, 2018
- [JWHY18] JINMING, CHEN ; WEI, JIANG ; HAO, JIAO ; YAJUAN, GUO ; GUOJI, NIE ; WU, CHEN: "Application Prospect of Edge Computing in Smart Distribution". In: *2018 China International Conference on Electricity Distribution (CICED)*. Tianjin, China : IEEE, 2018 — ISBN 978-1-5386-6775-0, S. 1370–1375
- [KMLK20] KOVATSCH, MATTHIAS ; MATSUKURA, RYUICHI ; LAGALLY, MICHAEL ; KAWAGUCHI, TORU ; TOUMURA, KUNIHIKO ; KAJIMOTO, KAZUO: "Web of Things"

- (WoT) Architecture*". URL <https://www.w3.org/TR/wot-architecture/>. - accessed on 2020-08-03
- [Luci17] LUCIA, MICHAEL J DE: "A Survey on Security Isolation of Virtualization, Containers, and Unikernels". In: *US Army Research Laboratory* (2017), S. 18
- [MRCD18] MARTIN, A. ; RAPONI, S. ; COMBE, T. ; DI PIETRO, R.: "Docker ecosystem – Vulnerability Analysis". In: *Computer Communications* Bd. 122 (2018), S. 30–43
- [NaAb19] NASAR, MOHAMMAD ; ABU KAUSAR, MOHAMMAD: "Suitability Of Influxdb Database For Iot Applications". In: *International Journal of Innovative Technology and Exploring Engineering* Bd. 8 (2019), Nr. 10, S. 1850–1857
- [Oasi12] OASIS STANDARD: "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0", Oasis Open (2012)
- [Oasi19] OASIS STANDARD: "OASIS MQTT Version 5.0.", Oasis Open (2019)
- [OSMC20] ORTEGA DE MUES, MARIANO ; SESEÑA, DANIEL ; MARTÍNEZ SPESSOT, CÉSAR ; CARRANZA, MARCOS ; LANG, JORGE: "Creating an effective and scalable IoT infrastructure by introducing Edge Workload Consolidation (eWLC)." URL <https://www.intel.com/content/www/us/en/develop/articles/edge-workload-consolidation-ewlc.html>. - accessed on 2020-06-24. — Intel
- [Toka17] TOKAR, DIMA: "Whitepaper: Five requirements of a leading IoT edge platform." URL <https://www.machnation.com/2017/09/18/whitepaper-five-requirements-leading-iot-edge-platform/>. - accessed on 2020-07-10. — MachNation
- [WLYC18] WANG, PAN ; LIU, SHIDONG ; YE, FENG ; CHEN, XUEJIAO: "A Fog-based Architecture and Programming Model for IoT Applications in the Smart Grid". In: *arXiv:1804.01239 [cs]* (2018). — arXiv: 1804.01239

1. Table of content

1.	TABLE OF CONTENT	1
2.	INDEX OF FIGURES	2
3.	INDEX OF TABLES.....	4
4.	GLOSSARY OF TERMS.....	6
5.	INTRODUCTION	8
5.1	CONTEXT	8
5.2	STATE OF THE ART	11
5.3	MOTIVATION	16
5.4	OBJECTIVES.....	17
6.	RELEVANT THEORY AND TOOLS	18
6.1	OVERVIEW OF CURRENT INFRASTRUCTURE.....	18
6.2	MQTT.....	21
6.3	AMQP	24
6.4	VIRTUAL MACHINES AND CONTAINERIZATION	26
6.5	KUBERNETES	28
6.6	NODE-RED.....	28
6.7	DATABASES.....	28
6.7.1	<i>Relational databases</i>	29
6.7.2	<i>Non-relational databases</i>	29
6.8	LOCAL TEST ENVIRONMENT	30
6.9	REMOTE TEST ENVIRONMENT	32
7.	ANALYSIS	33
7.2	OVERVIEW OF THE SOLUTION.....	33
7.3	ADVANTAGES, CHALLENGES AND FUNCTIONALITIES.....	35
7.4	ANALYSIS OF THE EDGE NODE.....	39
7.4.1	<i>Virtualization Technology</i>	39
7.4.2	<i>Operating System (OS)</i>	44
7.4.3	<i>Database</i>	45
7.4.4	<i>Hardware</i>	50
7.5	COMMUNICATIONS	51
7.5.1	<i>Inner Communications</i>	51
7.5.2	<i>Communications with Management System</i>	54

7.5.3	<i>Communications with other devices</i>	56
7.6	MANAGEMENT SYSTEM.....	58
7.6.1	<i>Criteria for vendor selection</i>	58
7.6.2	<i>Remote access to the Edge Node</i>	59
7.6.3	<i>Workload orchestration and containers deployment</i>	60
8.	ECONOMIC IMPACT	61
8.1	SAVINGS RELATED TO THE NEW APPROACH FOR NEW FUNCTIONALITIES IMPLEMENTATION	62
8.2	SAVINGS RELATED TO DEVICE SUBSTITUTION IN A SS.....	64
8.4	SAVINGS IN CENTRAL SYSTEM COMPUTING CAPABILITIES.....	66
8.5	SAVINGS RELATED TO ENERGY LOSSES	67
8.5.1	<i>Devices consumption</i>	68
8.5.2	<i>Fraud detection</i>	68
8.5.3	<i>Phase balancing</i>	69
8.6	SUMMARY OF ECONOMIC IMPACT	71
9.	TESTS	72
9.1	TEST #1: REAL-TIME BALANCE IN LOCAL TEST ENVIRONMENT	72
9.2	TEST #2: REAL-TIME BALANCE IN REMOTE TEST ENVIRONMENT USING S02-LIKE REPORTS	77
10.	CONCLUSIONS	80
11.	RECOMMENDATIONS FOR FUTURE WORKS	82
	ANNEX I. HARDWARE REQUIREMENTS	83
	ANNEX II. UNITED NATIONS SDG	84
	ANNEX III. CODE DEVELOPED	87
12.	BIBLIOGRAPHY	90

2.Index of figures

Figure 1.	Simplified diagram of the main communications needed for a secondary substation.....	9
Figure 2.	Communications Smart Meters-Data Concentrator and Data Concentrator-Central System.....	20
Figure 3.	Communications diagram for a group of secondary substations that form a level 2 aggregation	20
Figure 4.	Example of a MQTT network with 4 clients and 3 topics	21

Figure 5. AMQP functioning with a broker. Publish/Subscribe and Request/Response.	24
Figure 6. General Virtual Machine architecture.....	26
Figure 7. Container-based architecture using docker.....	27
Figure 8. CAP model.....	30
Figure 9. Overall representation of the architecture.....	34
Figure 10. Screenshot of MongoDB GUI (mongo express) and how it stores the JSON documents.....	47
Figure 11. Screenshot of a query in InfluxDB that shows how data is stored.....	47
Figure 12. Screenshot of Docker stats showing the consumption of InfluxDB and MongoDB in Docker containers.....	48
Figure 13. Simple general representation of the inner communications in the Edge Node	51
Figure 14. Screenshot of the MQTT-AMQP comparison test results.....	52
Figure 15. Screenshot of the error when trying to publish/subscribe to a non-defined queue ('measurements') in the AMQP broker.....	55
Figure 16. Diagram of the integration of the Edge Node with already in-use communication protocols.....	56
Figure 17. Simplified diagram of the application deployment process. Icon source: https://icons8.com/	60
Figure 18. Classification of the investments made by electric distribution utilities. [Cnmc19].....	61
Figure 19. Difference between the equivalent cost of deploying a single functionality in 25% of i-DE's SSs for the two approaches: 5.76 M€ for one device / two functionalities approach and 0.72 M€ for the Edge Node approach (edge computing).....	64
Figure 20. Approximate percentage of consumers per electric distribution company in Spain. Based on the number of meters provided by the CNMC in [PVBL19] and by the CIDE [Cide00].....	67
Figure 21. Estimated economic savings per year for phase balancing in i-DE considering different percentages of achievement. Calculated using the estimation of 20000 GWh of losses in 2016.....	70
Figure 22. Screenshot of the random data generator developed in Node-RED. It generates data pretending to be a LV supervisor and three smart meters.....	72

Figure 23. Screenshot of the Node-RED flow to store the JSON published on "feeder/balance" MQTT topic, in the local InfluxDB database	74
Figure 24. Grafana dashboard to visualize, in real time, the data from the devices and the results of the balance application (9.5% of average losses and the absolute losses).....	75
Figure 25. Screenshot of Docker stats for test #1	76
Figure 26. Node-RED Flow screenshot to check that the balance application deployed in the remote test environment works correctly	79

3.Index of tables

Table 1. Summary of the characteristics of this work in comparison to the literature presented in the State of the Art	15
Table 2. Examples of using wildcards for subscriptions to MQTT topics	22
Table 3. Summary of MQTT. Advantages and Disadvantages	23
Table 4. Summary of AMQP. Advantages and Disadvantages.....	25
Table 5. Docker client and docker engine versions installed in the VM.....	30
Table 6. Docker-compose version	31
Table 7. Python packages installed	31
Table 8. Summary of the comparison between virtualization alternatives.....	43
Table 9. Summary of the comparison between possible Operating Systems.....	44
Table 10. Examples of JSON and XML formats. Dummy data of a LV Supervisor	45
Table 11. Summary of the comparison between Database alternatives	49
Table 12. Summary for the selection of the inner communications protocol.....	53
Table 13. Summary for the selection of the protocol for communications with the Management System	55
Table 14. Summary for the selection of the protocol for communications with new devices	57
Table 15. Summary of the differences between the two approaches for the development of a new functionality.....	63
Table 16. RLE factor for each extended lifetime year. Calculated according to the formulas provided by the CNMC in [Cnmc19].....	65
Table 17. Cost in € for a Google Cloud server in Belgium (europe-west1). [Data00]...	66
Table 18. Estimated annual value of technical and non-technical losses for i-DE	68

Table 19. Estimated economic savings for the system under different achievement scenarios for phase balancing in i-DE. Present Value of each scenario assuming $r=2.58\%$ (WACC of Iberdrola on 18/06/2020)	70
Table 20. Summary of the economic impact of applying edge computing at SSs	71
Table 21. Examples of the JSON documents generated by the random data generator.	73
Table 22. Range of values for the active power factor and voltage factor for each meter.	73
Table 23. Example of part of the JSON document that contains the AI signal data for the supervision meter. This structure is repeated for every magnitude contained in a S02 report and for every meter.	77
Table 24. List of requirements for the device to be installed in an i-DE's SS. Source: i-DE	83
Table 25. Summary of the SDGs that the solution analysed in this project could have an impact on. Images source: [Unit00]	86

4. Glossary of terms

ACID	Atomicity, Consistency, Isolation, Durability
ADSL	Asymmetric Digital Subscriber Line
AMI	Advanced Metering Infrastructure
AMQP	Advanced Message Queuing Protocol
BN	Base Node in a PRIME subnetwork
DC	Data concentrator
CLI	Command Line Interface
COSEM	Companion Specification for Energy Metering
DLMS	Device Language Message Specification
DG	Distributed Generation
DR	Demand Response
EDF	Électricité de France
EDP	Energias De Portugal
eWLCA	Edge WorkLoad Consolidation Architecture
EV	Electric Vehicle
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
HEMS	Home Energy Management System
IoT	Internet of Things
JSON	JavaScript Object Notation
LAN	Local Area Network
LV	Low Voltage
LVAS	Low Voltage Advanced Supervisor
M2M	Machine-to-machine
MQTT	Message Queuing Telemetry Transport
MV	Medium Voltage
OASIS	Organization for the Advancement of Structured Information Standards
OLTC	On Load Tap Changer (transformer)

O&M	Operation and Maintenance
OS	Operative System
OT	Operational Technology
PLC	Power Line Communications
PoC	Proof of Concept
PRIME	PowerLine Intelligent Metering Evolution
PSU	Power Supply Unit
RLE	Retribution due to Lifetime Extension
RTU	Remote Terminal Unit
RUL	Remaining Useful Life
RMS	Remote Management System
SASL	Simple Authentication and Security Layer
SCADA	Supervisory Control And Data Acquisition
SCTP	Stream Control Transmission Protocol
SDG	Sustainable Development Goal
SN	Service Node in a PRIME subnetwork
SS	Secondary Substation
SSH	Secure SHell
SSL	Secure Sockets Layer
SSS	Smart Secondary Substation
STG	“Sistema de TeleGestión” in Spanish. Equivalent to the RMS
TTM	Time To Market
TLS	Transport Layer Security
WAN	Wide Area Network
WOT-A	Web Of Things Architecture
W3C	World Wide Web Consortium
WS	WebService
XML	eXtensible Markup Language

5. Introduction

5.1 Context

In the recent years, the concept of Smart Grid has emerged in the electric power industry as the natural evolution of traditional power grids to be more flexible and reliable. There are several definitions of what a Smart Grid is. One of them is the definition given by the Spanish electric distribution utility i-DE Redes Eléctricas Inteligentes (Iberdrola): “*Smart Grid is a technological evolution of the energy distribution system that combines traditional facilities with modern monitoring technologies, information and telecommunications systems. It will offer a wider range of customers services, improve supply quality, respond to the demand for electric power required of society in the future and achieve an optimal power distribution management*”[Iber00].

A great step in this technological evolution was the mandate of the European Union to deploy smart electricity metering covering, at least, 80% of electricity consumers by 2020 if the cost-benefit analysis of this deployment resulted positive in each member state [Coun14]. In the case of Spain, this target will be successfully accomplished. This massive deployment has meant the adaptation of thousands of secondary substations with additional equipment (e.g. Data Concentrator or DC) and telecommunications systems. Smart Meters do not only send information related to energy billing and consumption, they also send information (events) related to quality, security, fraud, high occurrence, etc. Therefore, the DC must cope with hundreds of millions of smart meter events every month and send this data to the Remote Management System (RMS), where it can be processed and analyzed in order to make decisions to improve grid performance and quality of service (e.g. commercial losses detection).

In addition to smart metering, grid automation and monitoring in real time are also important features to be considered during the path to a smarter grid. These features involve additional components, with different functionalities and communications protocols, that advanced secondary substations must have installed. Some of these components are:

- Remote Terminal Units (RTUs): It is a device that connects physical devices (e.g. breakers) with an automation system (Supervisory Control And Data Acquisition, SCADA).
- Power Supply Unit (PSU): It is a device in charge of managing the internal batteries present in the secondary substation.
- Low Voltage Advanced Supervisor (LVAS): It monitors electric (e.g. current profile) and quality parameters of the secondary substation low voltage feeders.
- Switch/Router to connect those components that allow TCP/IP communications with the control system.

Figure 1 shows a simplified diagram of the main communications that take place between a secondary substation and the corresponding centralized system (although it depends on the equipment available at the secondary substation).

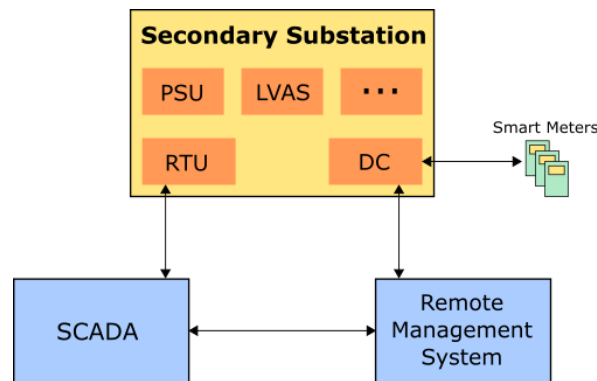


Figure 1. Simplified diagram of the main communications needed for a secondary substation

Once the data is sent to the RMS or to the SCADA, it is processed/analyzed to decide the actuation or to generate a report. Several research has been done in this field (Data Science, Artificial Intelligence), proposing multiple machine learning models and algorithms to get very interesting outcomes from the collected data, from demand forecasting to fault detection. Some of these algorithms are even designed to work with real-time measured data. However, these real-time applications must face some important constraints when applied in industry:

- The delay in the communications between the secondary substation and the meters, caused mainly by the characteristics of this type of communication, which uses the PRIME PLC standard. Obviously, physical distance is also an important factor that affects this delay. If smart meter data has to be sent to the central system to be processed, and the result of the data analysis is an action over any component of the secondary substation, this delay would limit real time applications despite the delay between the secondary substation and the central system is minimum in most cases.
- The increasing computing capacity (and storage) needed to process the collected data and its results/reports.

These constraints have also their effects on cost when additional equipment (e.g. sensors) is installed in a secondary substation: communications links might need a reinforcement and the necessary computing capacity as well as storage increases.

To address these problems, numerous researchers propose the implementation of edge computing ([CWWL19, HLWF18, JWHY18]), following an Internet of Things (IoT) approach, as a significant step towards a smarter grid.

Edge computing is the utilization of hardware and software resources installed at the edge of the network [JWHY18] (in this case, the secondary substation) to process and analyze

the data generated near/at this edge. It must not be seen as a substitute of cloud/centralized computing, but as a supplement or complement to it, since it would alleviate the workload of both the communications systems and the centralized data system (whether it is on cloud or not). Non-real-time, big data processes could be left to the cloud while real-time and short data processing could be done at the edge.

This approach is considered to have a great potential for the development of some Smart Grid-related features such as lines monitoring, demand response (DR), home energy management systems (HEMS) and smart substations [JWHY18].

In this context, the Spanish electric distribution utility i-DE Redes Eléctricas Inteligentes wants to evaluate the deployment of edge computing in terms of potential applications, benefits, and integration with the existing elements (i.e. DCs, sensors, communications systems...) in the grid.

5.2 State of the art

The idea of giving secondary substations the capacity of local data processing and autonomous actuation has been around for years in order to achieve a more reliable and smarter grid, giving birth to the “Smart” Secondary Substation (SSS) concept.

In 2010, the European Commission started the OpenNode project, carried out by a group of companies such as i-DE, EDP and EDF, among others. The objective of the project was to develop an “*Open Architecture for Secondary Nodes of the Electricity SmartGrid*” [Ref00a]. This Open Architecture, presented in [ASGM12] in 2012 (end of the project), has some similarities with the edge-computing architecture that is analysed in this project. First of all, it outlines the convenience of implementing changes over the traditional grid step by step, considering not only the integration with existing apparatus but also the consequent new assets management and return of the investment. The OpenNode architecture [ASGM12] is made of three main elements that would turn a traditional SS into a SSS:

- The Secondary Substation Node (SSN). Defined as a device with the ability of making decisions autonomously and remotely controlled. It is able to interact with any electrical equipment and local devices that are available in the SS.
- The Middleware. The element in charge of managing the information of the devices of the system, software provisioning, etc.
- Communications architecture. Fundamental element to provide the interactions between the SSN and the rest of devices, between the SSN and the Middleware, and even inside the SSN.

The essence of these three elements is also present in the solution discussed in this project: the edge computing device or “intelligent” node, the management system (on cloud or on premise) and the communications architecture. However, as expected when comparing two architectures spaced eight years, great differences arise in terms of technologies used. Starting with the software of the SSN, [ASGM12] distinguishes two (literal) partitions of software:

- Base Partition (BP): include the basic functionality of the SSN and provides the interface for communicating with the Extensibility Partition.
- Extensibility Partition (EP): A Java Virtual Machine that contains the software (extension modules) that provides additional secure functionalities.

The idea behind these partitions was to provide some sort of flexibility, since it was intended that the extension modules could be developed by third parties (respect to the SSN manufacturer) and that these modules could be run/stop/start without rebooting the SSN. These features also constitute the basis for the solution discussed in this project, but as mentioned before, by other means. The use of a Java Virtual Machine in the EP defined in OpenNode [ASGM12] requires the extension modules to be programmed in

Java, which limits software suppliers with know-how in other languages and big data/machine learning applications that might be written in Python or R, common languages for these purposes. Nowadays, containerization technology or even additional virtual machines (not only two partitions), provide isolated environments that give this programming language flexibility and, at the same time, isolated functioning.

As for the Middleware, analogous to the foreseen management system on cloud/premises, its purpose is basically the same: communicate with the SSN and store data provided by this. The OpenNode project [ASGM12] chooses to use a NoSQL database (Apache Cassandra) to store this data, based on the prevision of daily big data collection (50 TB). In this project it is not expected such big data collection by the management system, so the choice between relational and NoSQL databases may be discussed. This reduction in the amount of data collected affects directly to the communications architecture. The architecture detailed in [ASGM12] considers the use of WiFi, cellular networks (e.g. GPRS), PLC, Industrial Ethernet, etc. together with protocols such as DLMS/COSEM, IEC 61870-5-104, IEC 61850, etc. Although the OpenNode solutions were planned to be built on TCP/IP, it does not consider open protocols such as MQTT or AMQP, which are currently considered to be used in this kind of architectures.

All in all, the OpenNode project presented in [ASGM12] can be considered one of the first approaches to edge computing in electric distribution grids and it is a proof of the interest that this field was already raising a decade ago.

Based on the OpenNode idea, Siemens proposed in 2016 Gridlink [CeHP16], an architecture for an “intelligent” SSN. Basically, a communications infrastructure to deploy distributed applications written in Java. The reference [CeHP16], however, does not give details about the protocols used: it only mentions that is based on messages handled in a bus and that two types of commands are supported, `send` and `publish`, which might indicate some sort of publish/subscribe protocol (e.g. MQTT or AMQP). However, it mentions that Gridlink has compatibility for REST and XMPP protocols for communications with external components. The reference [CeHP16] basically focuses on introducing Gridlink in general terms and on the possible applications, such as the voltage control through OLTC transformers, which constitutes the experiment carried out by Siemens to test Gridlink and whose results are discussed in the mentioned paper.

In a later publication [FCSF17], the functioning of Gridlink for a non-real-time operation is depicted, describing the interaction between the Java modules, mainly based on requests and responses between them and with the storage module (database). It also specifies the database used in the local machine (node installed in the SS), Storacle, a time-series database embedded in Java, which significantly improves its functioning in comparison to other databases, as discussed by some of the same authors in [CeME15].

ABB, one of the leading manufacturers in electrical apparatus, defines the SSS simply as secondary substations where the apparatus are also “smart”: circuit-breakers with

automation logic, OLTC transformers with automatic switch, communications with the control centre, etc. [Abb16]. In the Technical Guide about SSS [Abb16], published in 2016, ABB highlights the importance of interoperability between the devices produced by different manufacturers, and the importance of choosing a communications system that guarantees this interoperability as well as security and a good performance.

However, this ABB's document only considers specialized smart devices, that is to say, hardware with embedded software that runs a specific functionality (e.g. protections automation) and that communicates with the central system of the utility. It does not contemplate a unique device in the SS where multiple functionalities are deployed to send orders to different electrical apparatus (e.g. OLTC transformers, circuit breakers, etc.) based on local data processing.

From an economic perspective, [SoLi12] conducts a pure economic study of the life cycle cost of a smart substation in China. Despite it considers CAPEX and OPEX, it does not consider possible retributions due to added functionalities nor savings in time of deployment (as it is commonly said, time is money). Besides, it only makes reference to primary substations and not SSSs, and its description of a smart substation is very general, not matching the approach that is discussed in this project.

From a general perspective, [JWHY18] discusses the possibilities of applying edge computing to distribution networks, presenting edge computing not as a substitute of cloud computing, but as a supplement to it to carry out real and semi-real time operations. It also discusses cases of use such as LV grid management, LV topology identification, fault identification and losses analysis, as well as its possible usefulness for applications related to EVs, DG and energy storage. Additionally, it identifies some challenges to be addressed. The first one is the definition of a framework that satisfies the requirements of distribution grids; it mentions ParaDrop (which is based on containerization) and Cloudlet, but considers that they need modifications. The other challenges are the problem of scalability due to the computing capacity-price relationship of chips, the need of collaboration between cloud and the node to apply AI algorithms, and the challenges related to security and privacy in the Edge node [JWHY18]. This reference [JWHY18] is an "Application Prospect", so it does not discuss the technologies that may conform that necessary framework, the factors to take into account, and the possible economic benefits or savings (if any) of implementing edge computing in distribution grids.

An edge computing system for smart grids is proposed in [CWWL19]. It defines an architecture made of five layers: cloud computing, application layer, data layer, network layer and device layer. However, it does not discuss which technologies are used (e.g. if it uses virtual machines or containers, IoT communication protocols, etc.), but it does review the advantages of edge computing in electric distribution systems, the challenges, and some possible functionalities (e.g. video surveillance, micro-grid systems...), proposing some algorithms for data protection and prediction, and task grading. To prove

some of the advantages, [CWWL19] carries out some simulations that show that the transmission bandwidth in an edge computing architecture would be significantly lower for the same number of devices than in a cloud architecture (centralized). Same with the delay in communications.

Similar to edge computing, a fog-based architecture is proposed by [WLYC18] to deploy applications in the distribution grid. It uses an additional layer, made of fog computing coordinators, that is in charge of managing the different groups of fog nodes (Edge nodes in this work), so that they can work together to carry out complex tasks. However, the result of these tasks must be sent back to the corresponding fog computing coordinator to send an order to the action device. Although this approach is significantly different from the one discussed in this work, it uses some technologies that will also be present in the discussion of this work: it uses MQTT as the communication protocol between fog nodes and fog coordinators and it uses Node-RED as programming tool. [WLYC18] uses what it calls “OSS servers” in order to execute the applications, assuring better characteristics than VMs and Docker containers, but it does not provide much information about them. The paper does not give details about what database would be used for storage capabilities in the fog nodes, it does not consider inner communications between applications in the fog nodes, how communications with other devices would be done, and, although intuitive, it does not specify where the fog computing coordinators and the fog nodes would be located, which is something to consider in order to determine the communications possibilities.

In an attempt of defining a standardized architecture for edge computing, Intel and Minsait (Indra) published on February 2020 the open Edge Workload Consolidation Architecture (eWLCA) [OSMC20]. This architecture defines four tiers (that use different Intel processors) that may be or not in a final deployment, depending on the industry and the client:

- Edge devices: what in industry are known as IoT Gateways or Edge Compute Devices. They are placed close to where the data is generated. Enough computing capacity to host processing applications.
- Edge server: collects data from the Edge devices. They are usually servers on-premise, to reduce dependence from cloud providers. Its role could be similar to the fog computing coordinator presented in [WLYC18].
- Edge core: a management system for the edge devices and servers.
- Edge cloud: to carry out cloud operations.

This open architecture specifies the use of different open source technologies, such as the MQTT protocol for communication, InfluxDB as the database to store the collected data, Docker as the containerization environment, Kubernetes to manage the workload in the Edge server... Minsait has adopted this open architecture (with few modifications) for its edge computing solution and has developed its own (proprietary) management system (edge core tier), called Onesait Things Platform. This solution will be tested by i-DE.

Table 1. Summary of the characteristics of this work in comparison to the literature presented in the State of the Art

Ref.	Compares virtualization alternatives (VMs, Unikernels, Containers...)	Analyses the use of open IoT communication protocols (MQTT, AMQP) in three levels (Inner, with Central System, with other devices)	Economic study of possible benefits and savings derived from Edge Computing functionalities	Advantages/Challenges/Functionalities of Edge Computing at SS	Smart Secondary Substation concept	Considers Software decoupled from hardware	Analyzes the database to be used in the Edge compute device	Test the solution
[ASGM12]	✗	✗	✗	✗ / ✓ / ~	✓	✓	✓ (Different)	✓
[CeHP16, CeME15, FCSF17]	✗	✗	✗	✗ / ✗ / ✓	✓	✓	✓ (Different)	✓
[Abb16]	✗	✗	✗	✗ / ✓ / ✓	✓	✗	✗	...
[JWHY18]	✗	✗	✗	✓ / ✓ / ✓	~(Mentioned)	✓	✗	✗
[WLYC18]	~(Mentioned)	✗	✗	✓ / ✓ / ✓	✗	✓	✗	✓
[CWWL19]	✗	✗	✗	✓ / ✓ / ✓	✓	✓ (Deduced)	✗	✓
[SoLi12]	Focuses on CAPEX and OPEX of the substation	...	~(Not secondary)	✗
This Project	✓	✓	✓	✓ / ✓ / ✓	✓	✓	✓	✓

5.3 Motivation

The State-of-the-art shows that there is an interest by part of the electric industry of moving data processing and decision-making to the edge of the distribution network, more specifically, to the SS. It also shows that, along the years, there have been different proposals about how to do this approach by academics and industry.

Table 1 shows that these proposals mainly focus on the technical and operational advantages, challenges and functionalities that edge computing in the distribution grid could provide but do not usually discuss the alternatives for each technology used and, therefore, do not fully justify why the selected one is better than the others. In addition to this, none of the literature reviewed in Table 1 distinguishes clearly the three communications environments (and which IoT protocol would suit better for each) that must be considered by the electric distribution company to evaluate its future integration in a SS.

This work will analyse the main technologies (and their main alternatives) defined in the eWLCA defined by Minsait and Intel [OSMC20] since i-DE and Minsait are working on a Proof of Concept (PoC) following this architecture. This architecture is general (i.e. it is not specifically designed for electric distribution), so this analysis is necessary to make sure these technologies can address the requirements of an electric utility in terms of operation and security.

The possible economic benefits and savings that Edge Computing (and its possible functionalities) could provide to the electric utility and to the entire system are also discussed in this project (no confidential data from i-DE is used). This type of discussion has not been found in the State-of-the-art (see Table 1), but it should be considered when analysing a solution that could constitute an important investment and an important change in the future operation of the LV grid, overall considering that the electric distribution in Spain is a regulated sector.

Therefore, this work is a great opportunity to evaluate the convenience (or not) of applying edge computing at secondary substations, the technologies that might be used nowadays and the possible economic benefits for the system and for the utility.

5.4 Objectives

The main objectives of this project are:

1. To determine advantages, disadvantages, and challenges of edge computing respect to current secondary substations, including functionalities that could be implemented with this solution.
2. To evaluate an edge-computing-based solution (its supporting architecture, protocols...) for the deployment at secondary substations and compare it with alternative solutions in different aspects: technical, maturity of the technology used, flexibility...
3. To economically study this new approach and identify benefits in comparison to the traditional procedures for the deployment of new functionalities/devices in a secondary substation.

6. Relevant theory and tools

6.1 Overview of current infrastructure

Although the devices and communications that most I-DE's secondary substations have been briefly mentioned in the Introduction section, this section will explain in more detail the current infrastructure (systems, protocols, etc.) in order to understand fully the environment where an edge computing device would be installed.

The deployment of smart metering made by I-DE in Spain constitutes what is called an Advanced Metering Infrastructure (AMI), which includes bidirectional communication between the meters and the central system, allowing functionalities such as service connection/disconnection and to obtain measures (not always related with consumption, such as voltages) at specific times to improve grid performance.

However, the connection between I-DE's central system (STG, Spanish acronym of "Sistema de TeleGestión", which in English would be Remote Management System) and the millions of smart meters that are deployed is not usually direct. There is a device, the Data Concentrator (DC), installed in the SS, which acts as an intermediary between them.

The communication between the DC and the smart meter is done using DLMS/COSEM¹ specification over PRIME² PLC³, which is a standard designed for "*Advanced Metering, Grid Control and Asset Monitoring applications*"[Prim00a], that allows the use of power lines to exchange information, avoiding the deployment of a costly additional communications infrastructure. Two nodes can be distinguished in a PRIME subnetwork: the Base Node (BN), which is the master node that manages the network resources, and the Service Nodes (SN), nodes managed by the BN, that can act as terminals or as switches (repeating other nodes' information packets for connectivity expansion) [SSBS16]. PRIME v1.3.6, which is the version currently deployed, reaches up to 128.6 kbps of data rate, while PRIME v1.4 (compatible devices still under development) would allow between 5.4 kbps and 1028.8 kbps of data rate, depending on the technical characteristics (number of channels, modulation scheme...) [SSBS16]. PRIME v1.4 will provide new modes, the "robust modes", that will expand the supported frequency up to 500kHz (allowing higher data rates in specific environments) and will enable communication in those environments with low signal-to-noise ratio [Prim00b] . Generally speaking, PRIME v1.4 improves robustness, data transfer speed, and safety (encryption mechanisms), being appropriate for networks with high noise, while keeping compatibility with v1.3.6 existing devices. Considering this, v1.4 is expected to allow

¹ DLMS stands for Device Language Message Specification. It is the application layer protocol that creates the messages using the information stored by the meter [Over00]. COSEM stands for Companion Specification for Energy Metering. It is the model that allows the description of almost any application. [Over00]

² PRIME stands for PoweRline Intelligent Metering Evolution, defined and maintained by the PRIME Alliance [Prim00a]

³ PLC stands for Power Line Communications

new functionalities that are currently limited by v1.3.6 in the smart meter [Garc16, Prim00b].

On the other hand, the communications between the STG and the DC are based on WebServices (WS) (HTTP/1.1 transport protocol) and FTP (File Transfer Protocol), using a proprietary STG-DC interface protocol. This protocol defines how the STG and the DC both exchange information based on XML messages. This bidirectional communication can be due to different reasons [CMFL15]:

- The STG sends a request to a DC to take any action or to provide the requested information. This information can be sent as soon as possible (synchronous request, usually using WS) or at a specific time (asynchronous request, using WS or FTP). Smart meter/DC firmware updates are done using FTP, due to the complexity and weight of the files.
- The DC sends to the STG the data corresponding to the programmed internal tasks or related to events (e.g. alarms) in a meter or in the DC. This is done using WS or FTP. The programmed tasks typically included in a DC are [CMFL15]:
 - Daily billing values collection
 - Daily load profile collection
 - End of billing profile collection
 - Event reports collection
 - Daily absolute report collection

When the STG requests information, there are different alternatives, depending on the source required: it can be a request to the DC database, a request directly to the meter(s) (e.g. for instantaneous values), or a combination of the two previous, where the request is done to the DC and, if it does not have the information or it is incomplete, it asks the meter(s).

Figure 2 shows a diagram of the communications protocols and links that currently take place at an i-DE secondary substation (for AMI). If a SS wants to communicate with the central system (STG), it must have a router (level 3) or a switch (level 2) that aggregates several secondary substations and connects them with the central system through a router. The connection between the DC and the router/switch is usually made via ethernet since both devices are usually placed in the same cabinet as the DC. Some DC models do not even need connection to a router since they already have direct connection with the STG through a built-in 2G/3G modem. The connection between the router and the central system, on the other hand, is usually done using communication networks owned by the electric utility (in the case of i-DE, optical fiber or MV PLC). Due to the natural geographical dispersion of secondary substations, this utility-owned network does not reach to every SS, so, in those cases, this communication is provided by a third party, mostly 3G (GPRS, General Packer Radio Service is currently in disuse, with few exceptions) or ADSL (Asymmetric Digital Subscriber Line) if there is no coverage.

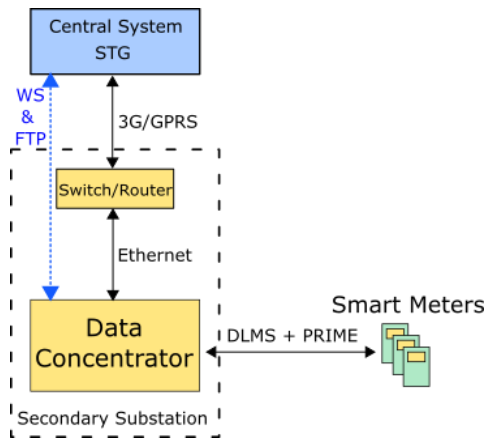


Figure 2. Communications Smart Meters-Data Concentrator and Data Concentrator-Central System

The aggregation of multiple secondary substations in the level 2 is usually done using PLC MV or optical fiber for the connections between switches (Figure 3). The final communication with the central system is done via a router (level 3), usually called “Concentration header”.

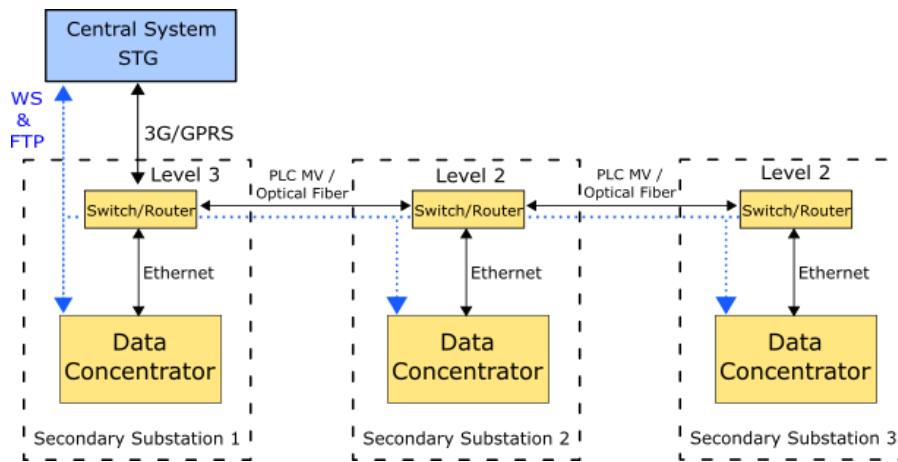


Figure 3. Communications diagram for a group of secondary substations that form a level 2 aggregation

6.2 MQTT

Message Queuing Telemetry Transport (MQTT) is an open OASIS and ISO standard based on a lightweight publish/subscribe architecture that allows the communication between machines (M2M) relying on TCP/IP. It has gain popularity for IoT devices as it works well with low network bandwidth due to its lightness and it also has low power consumption [Oasi19].

The publish/subscribe protocol is based on a simple idea: those devices that want to transmit any information must publish it as a message on a topic; every other device that is subscribed to that topic will receive the published message with the data inside. To manage the topics (subscribers and publishers, which are the MQTT clients), it is needed the figure of a MQTT broker. A MQTT broker is just a server that routes all the published messages on a topic to the corresponding subscribed MQTT clients. This way, publishers do not need to know how many subscribers their topics have or where these subscribers are; the broker manages it.

If a topic does not have any subscriber, the MQTT broker does not store any message unless it is marked as a *retained message*. In this case, when the first client subscribes to the topic, the last retained message will be sent to the subscriber. Although the figure of the MQTT broker does not allow the MQTT clients to “see” each other, publishers can set a message to be sent to the subscribers of the topic if the publisher suddenly loses connection with the broker, as a way of keeping track of remote devices and possible failures. This feature is commonly known as “Last Will and Testament” [Mqtt00a]. Figure 4 shows a simplified example of the functioning of the MQTT protocol in a network made up of four clients and three topics.

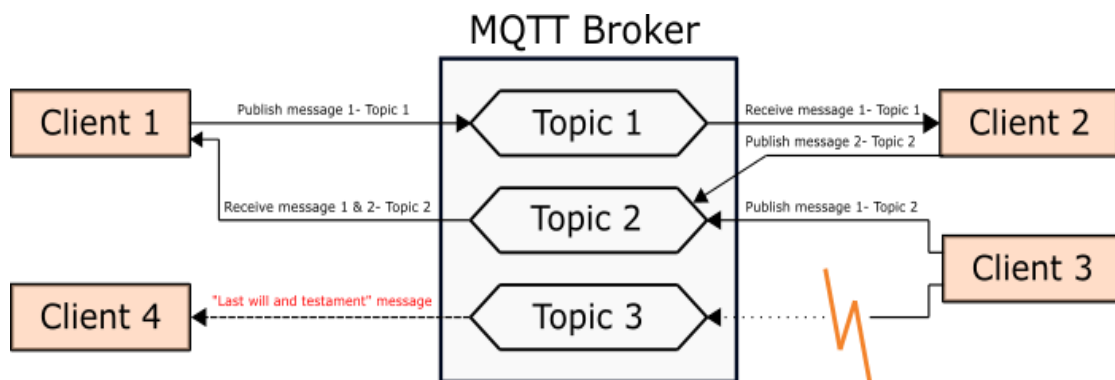


Figure 4. Example of a MQTT network with 4 clients and 3 topics

Topics in MQTT follow a tree architecture. For example, a topic could be *substation1/temp_sensor/status*, each term separated by a forward slash constitutes a topic level (branches of the tree). To allow the subscription to various topics by the same client, MQTT allows single-level and multi-level wildcards [Oasi19]. Table 2 shows some examples of the use of wildcards in topics. Wildcards can only be used by subscribers and not by publishers.

Table 2. Examples of using wildcards for subscriptions to MQTT topics

Single-level (+)	Multi-level (#)
Example: <i>substation1/+/status</i>	Example: <i>substation1/#</i>
✓ <i>substation1/temp_sensor/status</i>	✓ <i>substation1/temp_sensor/status</i>
✓ <i>substation1/O2_sensor/status</i>	✓ <i>substation1/O2_sensor/status</i>
✗ <i>substation1/temp_sensor/data</i>	✓ <i>substation1/temp_sensor/data</i>
✗ <i>substation2/temp_sensor/status</i>	✗ <i>substation2/temp_sensor/status</i>

Three quality of service levels are defined in MQTT [Oasi19]:

- 0: The message will be delivered by the broker once, without needing confirmation of delivery.
- 1: The message will be delivered by the broker at least once, needing confirmation of delivery.
- 2: The message will be delivered by the broker once, but using a four-step handshake.

An important feature of MQTT is that it is data agnostic. This means that messages do not follow any format, it is flexible, the payload of the message can be anything up to around 250-260 MB (256 MB) [Mqtt00b]. However, this payload cannot be a file, it must be sent as a text. For example, if a CSV file has to be sent, it would be necessary to extract its content to be included as the payload of a MQTT message (or by traducing it to binary). The subscribers will have to process and interpret this message conveniently.

As for security features, since version 3.1, the MQTT protocol allows to set a username and a password. Nevertheless, encryption of messages is not built-in to the protocol (it would lose its lightness). If, due to the importance of the data, encryption is needed, it should be done by an external encryption application. For example, TLS/SSL protocol can be used to manage encryption in the network, but using a different TCP port (8883) [Oasi19].

Table 3. Summary of MQTT. Advantages and Disadvantages

<u>MQTT</u>	
Advantages	Disadvantages
<ul style="list-style-type: none"> • Very simple to implement 	<ul style="list-style-type: none"> • It is not convenient for sending files
<ul style="list-style-type: none"> • Publish/Subscribe allows broadcast, multicast and M2M. MQTT v5.0 introduces request-response feature. 	<ul style="list-style-type: none"> • Does not work well with very high volumes of data. Limit of payload.
<ul style="list-style-type: none"> • It needs virtually no administration (e.g. the topics are created automatically) 	<ul style="list-style-type: none"> • Encryption of messages and advanced security is not built-in to the protocol. It must be set apart.
<ul style="list-style-type: none"> • Light and low power consumption 	<ul style="list-style-type: none"> • A broker is needed. Weak point of the network (bottleneck)
<ul style="list-style-type: none"> • 3 levels of Quality of service 	<ul style="list-style-type: none"> • Cannot control data flow, automatically done by the broker
<ul style="list-style-type: none"> • Compatible with TLS/TLS 	
<ul style="list-style-type: none"> • Availability to be used in different programming languages (e.g. Java, Python, etc) 	
<ul style="list-style-type: none"> • Supported by big companies (e.g. IBM, Facebook, etc.) 	
<ul style="list-style-type: none"> • Data agnostic (binary encoding) 	
<ul style="list-style-type: none"> • It is open source 	
<ul style="list-style-type: none"> • “Last Will and Testament” feature 	

6.3 AMQP

The Advanced Message Queuing Protocol (AMQP) is an open OASIS and ISO standard that supports two architectures: publish/subscribe (like MQTT) and a sort of request/response. Similar to MQTT, AMQP is characterized for also being a light M2M protocol which relies on TCP/IP or SCTP, but focused on interoperability [Naik17].

Although AMQP is based on publish/subscribe, its functioning is not as simple as MQTT. In AMQP, two elements can be distinguished inside the broker: exchanges and queues. An exchange is where messages are delivered by the publishers. Queues are created by the consumers and linked to an exchange. When the exchange receives a message, it looks at its routing key (message attribute) in order to route the message to the corresponding queue; this process is called “binding” [Naik17]. Once in the queue, the message can be directly sent to the consumers (publish/subscribe) or it can be stored to be delivered when requested by the consumer (request/response). Figure 5 shows the basic functioning of AMQP with the supported architectures.

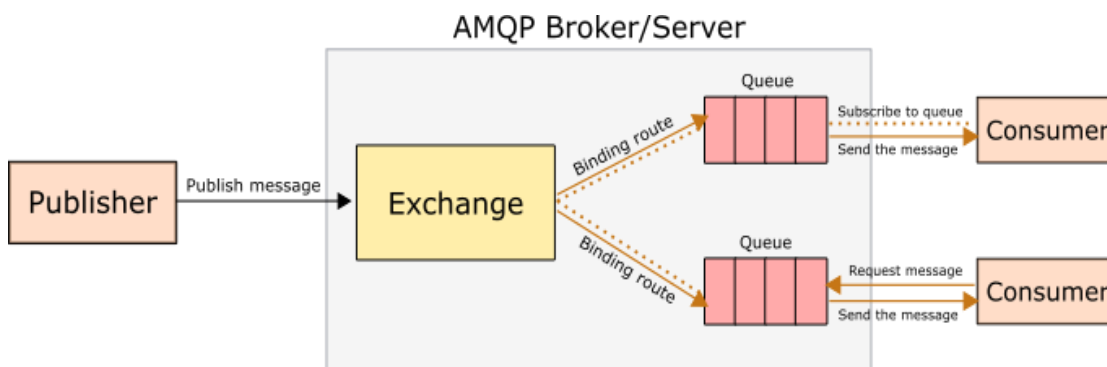


Figure 5. AMQP functioning with a broker. Publish/Subscribe and Request/Response.

Different types of exchange are defined in AMQP [Naik17]:

- Direct: the message is assigned to the queue(s) whose binding key is exactly the routing key defined in the message.
- Fan-out: the exchange routes the message to all the queues linked to it.
- Topic: the message is routed based on the matching between the routing key and the routing pattern (wildcard). This way, messages can be routed to one or more queues at the same time.
- Header: similar to the topic exchange, it differs in that the routing criteria is contained in the message Header and not in the routing key.

AMQP messages are encoded in binary format (data agnostic as MQTT) and the message size is not defined, it depends on the capacity of the broker or on the programming technology [Naik17]. One of the main features of AMQP is its reliability and, to achieve it, two main quality of service levels are defined: settle format (message is sent at most once) and unsettle format (message is sent at least once) [Naik17].

As for security, AMQP v1.0 specifies how to establish TLS sessions (to encrypt communications) and how to establish a SASL layer (for authentication purposes) [Oasi12].

Table 4. Summary of AMQP. Advantages and Disadvantages

<u>AMQP</u>	
Advantages	Disadvantages
<ul style="list-style-type: none"> • Publish/Subscribe allows broadcast, multicast and M2M 	<ul style="list-style-type: none"> • Maximum message size depends on the broker and programming tool
<ul style="list-style-type: none"> • Allows request/response (RPC) 	<ul style="list-style-type: none"> • Not as simple and light as MQTT
<ul style="list-style-type: none"> • Interoperability, reliability, and security are at its core. 	<ul style="list-style-type: none"> • Although a broker is not necessary, it has a significant impact on reliability.
<ul style="list-style-type: none"> • A broker is not strictly necessary. 	
<ul style="list-style-type: none"> • Every new version is compatible with previous ones 	
<ul style="list-style-type: none"> • It does not specify limit in maximum message size 	
<ul style="list-style-type: none"> • Big companies support the standard (e.g. Microsoft, Thales, etc.) 	
<ul style="list-style-type: none"> • Specifies advanced security measures (TLS and SASL) 	
<ul style="list-style-type: none"> • Availability to be used in different programming languages (e.g. Java, Python, etc) 	
<ul style="list-style-type: none"> • Data agnostic (binary encoding) 	
<ul style="list-style-type: none"> • It is open source 	

6.4 Virtual machines and containerization

The edge computing device is expected to run based on virtualization and/or containers (docker), which constitutes one of its main features and advantages. For this reason, first it is necessary to define what virtualization is, what docker is and what is the difference between a virtual machine and a container.

First thing to note is that virtual machines and containers are compatible; you can deploy both at the same time, and both of them result in self-contained virtual packages. However, they have different characteristics and operation, resulting in distinct use cases.

The result of virtualization is called a “virtual machine” (VM): an emulation of a physical computer with its own Operating System (OS). Therefore, several different operating systems can be run on the same hardware (host machine) through VMs. These VMs run thanks to a hypervisor, which is the software/firmware/hardware in charge of assigning the resources to each VM (storage, RAM, etc.) [Suma13]. Two types of hypervisors can be distinguished [Suma13]:

- Type I or “Bare metal” hypervisors: When it is directly implemented between the hardware and the working framework (OS).
- Type II or “Embedded” hypervisors: When the hypervisor is installed as software on the OS. The most known hypervisors of this type are VirtualBox and VMWare.

Figure 6 shows the general virtualization architecture. The difference between type I and II in the diagram would be the placement of the OS stack: over the hypervisor for type I and below the hypervisor for type II.

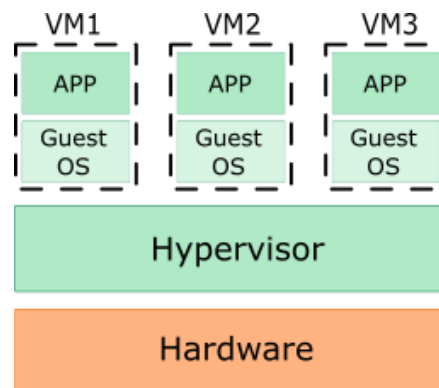


Figure 6. General Virtual Machine architecture

Containerization, on the other hand, consists on the encapsulation of an application (typically one per container) together with all its requirements (libraries, settings, etc.) so that the application can run anywhere, independently of the host OS, as it already has everything it needs. As opposite to virtual machines, containers run its operating environment on the same host OS. The most famous tool for containerization is Docker, an open source project that has an extraordinary adoption by developers. Figure 7 shows the container-based architecture using docker (simplified).

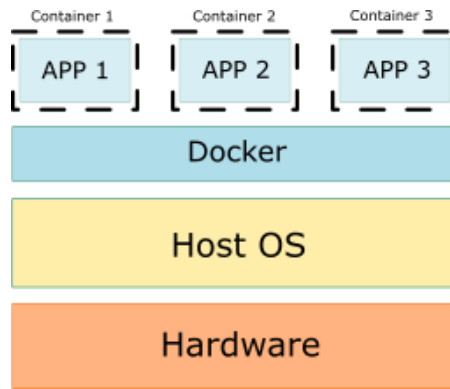


Figure 7. Container-based architecture using docker

Briefly explained, docker architecture is basically made up of three components [Dock20a]:

- Client (docker CLI): the way users interact with Docker by using commands.
- REST API: it sets the interface used by programs/users to talk to the docker daemon.
- Docker daemon: it receives the Docker API requests and operates the instructions.

In addition to these architectural components, there are the so-called Docker objects, the following are the basic ones [Dock20a]:

- Images: it is a read-only file that indicates how to create a Docker container. Images can be created by you or by other people who publish them in a registry. The creation of an image is based on a Dockerfile, which is a file that specifies the steps that must be followed to create and run it.
- Containers: it is a “*runnable instance of an image*”[Dock20a]. Containers can be manipulated (stop, delete, move...) using the docker CLI. The level of isolation from other containers and the host machine is also controllable, as well as its storage configuration.

Sometimes, containerized applications may not be made of just one container but of many that need to be able to interact between them. For multi-container applications, Docker offers a tool called Docker Compose that, based on a defined file (docker-compose.yml), establishes the services that build the application [Dock20b].

6.5 Kubernetes

Kubernetes, initially developed by Google as open source since 2014, is a platform for the management of containerized workloads and services [Kube20a]. The result of the deployment of Kubernetes is a cluster, which is one or more worker devices (nodes) where containers are run. The basic unit created and managed by Kubernetes is a pod. A pod is just a group of containers that share the same storage/network and that contains a file that specifies how the containers must be run. Therefore, pods make it simple to deploy and manage applications whose components need to share data/communicate between them [Kube20b].

Kubernetes is specifically designed to run distributed systems in a resilient way. If more than one container is needed to run an application, and one of these containers fails for whatever reason, Kubernetes restarts the failing container, so that the application can still run (it is self-healing). Kubernetes is, therefore, very useful for high computing applications (e.g. image processing) that require multiple nodes working together, since it automatically manages the available capacity of these nodes to run the applications, saving deployment time.

6.6 Node-RED

Node-RED, initially developed by IBM and currently part of the JS Foundation, is a programming tool based on flows [Node00]. It is a free and open source tool. In Node-RED, applications are described with a set of black-boxes (nodes) connected together (building a flow). Each node receives data, works with it and then it passes the result to the following node. One of the main advantages of Node-RED is that it can run on low-cost hardware (e.g. Raspberry Pi) and on cloud, and it is designed to be used by a wider range of users (not only professionals). However, functions must be written in JavaScript, which might be a barrier for less advanced users [Node00].

6.7 Databases

Whether for the central management system or for the edge computing device, a database will be needed to store events and relevant information. Two main types of databases are distinguished: relational databases and non-relational databases. To assess the reliability of a database type, there are the so-called ACID properties [JPAK12]:

- Atomicity: a transaction fails if any part of it is incomplete.
- Consistency: if the database remains stable before and after a transaction.
- Isolation: transactions do not interfere with each other when executing at the same time.
- Durability: when a transaction is completed, it will stay in the same state.

6.7.1 Relational databases

In relational databases data is organized in well-defined tables (columns and rows). Each row in these tables constitutes a record and, each column, a field that each record should have filled. The information contained in different tables can be linked by using indexes or foreign keys (“relation”) [JPAK12].

Most of the relational databases allow to easily access and modify the data stored using SQL. Some of the main advantages of relational databases are that they are self-documenting and, if the schema of the database is required to be changed, it is a simple process. Besides, most relational databases provide ACID properties. On the other hand, some disadvantages of relational databases are its limited scalability, its complexity in case that the data cannot follow a table structure and, if the database is very large, the complexity of managing a distributed database among different servers [JPAK12].

6.7.2 Non-relational databases

The main difference between relational databases and non-relational databases is that the last does not use tables as structure. They do not use SQL as the language to modify, insert or look data, either (non-relational database = No-SQL database). According to [JPAK12], up to ten different types of non-relational databases may be distinguished:

- Key Value Stores: data is stored as key-value pairs.
- Document Oriented Database: documents (PDF, XML, Microsoft Office, etc.) are stored by assigning them a unique key .
- Graph Database: data is represented by using edges, nodes and graph data structures [JPAK12].
- Column Oriented Databases: they use columns to store the data (not in rows as relational databases).
- Object Oriented Databases: data stored as objects (with similarities to object-oriented programming).
- Grid and Cloud Databases.
- XML Databases.
- Multidimensional Databases: data is stored as n-dimensional matrix.
- Multivalued Databases: three dimension data (fields, values and subvalues) [JPAK12].
- Multimodel Databases: a mix of other non-relational databases.

Additionally, Time Series Databases should be added to the previous list. In this type of databases, data is stored as a relation between time and value(s) (time series)

Generally speaking, non-relational databases are more effective at managing large amounts of data than relational databases, so they are a good solution for cloud

architectures. However, non-relational databases have a lower reliability level since many of them renounce to some ACID properties to increase performance (usually consistency). In fact, non-relational databases must choose two out of three possible properties: Consistency, Availability and Partition tolerance (CAP model, Figure 8) [GiLy12]. Relational databases would be positioned as CA in this model whereas most of non-relational databases would be divided between CP and AP.

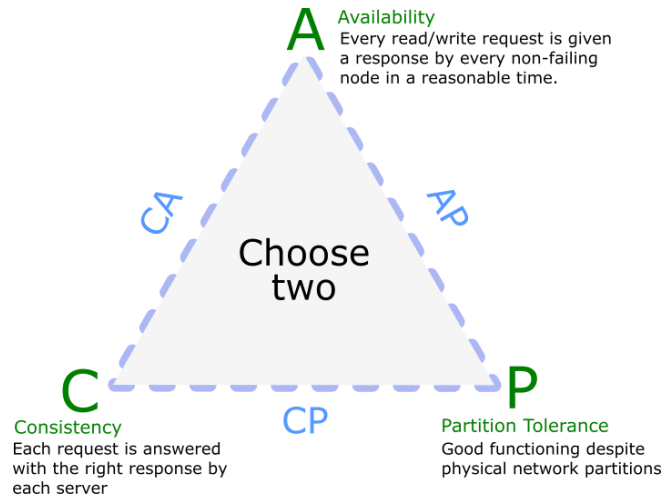


Figure 8. CAP model

6.8 Local Test environment

To carry out tests and experiments for the analysis, an edge computing device is simulated. The community version containers of the Onesait Platform (free and open source) will be used in this simulation/test environment since i-DE and Indra are developing a proof on concept of this platform (enterprise version, which mainly differs in the cloud management system) to be used at smart secondary substations.

The test environment will be a local virtual machine, with VirtualBox 5.2 as hypervisor, with Ubuntu 18.04.4 installed as OS, and giving this virtual machine 3,018 GB of RAM and 30,75 GB of storage. Docker engine (Table 5) and docker compose (Table 6) are installed in this VM. The “containerd” version used by the docker engine installed is 1.2.13.

Table 5. Docker client and docker engine versions installed in the VM

	Docker Client – Community version	Docker Engine – Community version
Version:	19.03.8	19.03.8
API version:	1.40	1.40
Go version:	go1.12.17	go1.12.17
OS/Arch:	Linux/amd64	Linux/amd64

Table 6. Docker-compose version

Docker-compose	
Docker-compose version:	1.25.5
Docker-compose build:	8a1c60f6
Docker-py version	4.1.0
CPython version	3.7.5
OpenSSL version	1.1.01 (10 Sep 2019)

The Onesait Platform webpage specifies which dockers are standardly used in the edge node and explains how to install them using docker-compose (free containers) [Edge00a]:

- Node-RED (v0.20.5) container.
- InfluxDB (v1.8.0) Database container.
- Inner MQTT broker container. Based on Mosquitto MQTT broker.

Additional containers that are installed to do the analysis and compare technologies:

- MongoDB (v4.2.7) Database container (to compare with InfluxDB).
- Inner AMQP broker container. RabbitMQ broker (to compare with MQTT).
- Grafana. To visualized stored data in a dashboard.

To develop simple test programs, Python 3.8.1 will be used as programming language. The use of docker makes it unnecessary to install Python on the VM, in fact, these programs will be developed in a separated environment that uses Windows 10 as OS, just as it would be the case for a real application provider. Some free Python packages that may be useful to install are shown in Table 7.

Table 7. Python packages installed

<u>Package</u>	<u>Description</u>	<u>Installation via pip install</u>
Paho-mqtt 1.5.0	Python MQTT Client library	paho-mqtt
Pika 1.1.0	Python AMQP Client library	pika

In addition to this, it is recommended the installation of `mqttfx`⁴, a free software developed by Jens Deters, that provides a graphical user interface to publish and subscribe to MQTT topics of an external broker server. This program might be useful to debug errors when interacting with the inner MQTT broker container in the edge node during development.

⁴ <https://mqttfx.jensd.de/>

6.9 Remote Test environment

In order to test how an application would be deployed in a remote node on field, Minsait has provided i-DE access to an instance of its Onesait Edge Management System (in Azure) and to a remote node (Ubuntu 16.04 LTS, 230.7 GB of storage and 15.6 GB of memory, not representative of the real edge node that would be deployed but useful for testing). Minsait has also provided access to a container registry in Azure in order to make tests, as well as a STG-DC simulator in a container. Since this environment is own by Minsait, no details will be given in this work about the management system interface or the STG-DC simulator.

7. Analysis

7.2 Overview of the solution

The architecture that is taken as reference, and that will be analysed, is the eWLCA defined by Intel and Minsait in [OSMC20] and that was mentioned in the State-of-the-Art.

This architecture can be understood as a compendium of open and proprietary technologies that play a specific role each. The relationship between technology and role is not unique (in most cases), so multiple technologies might be appropriate to play the same role. The choice should be made based on, not only present cases of use, but also on potential uses and ways of functioning, remembering that flexibility is a key factor to consider.

Although the eWLCA defines four tiers, not all of them are necessary. In the proof of concept (PoC) that is being developed by Minsait for i-DE to achieve a SSS, only the Edge Device tier and the Edge Core (Management System) are considered.

To carry out the analysis, more focused on i-DE's requirements and the PoC, three main elements will be distinguished in this architecture (as in the OpenNode project mentioned in the State-of-the-Art):

- The Edge Node (edge compute device, intelligent node): it includes the hardware that will be installed in the secondary substation and the corresponding software architecture (e.g. OS, VMs, Docker, databases, etc.)
- The Management System whether on cloud or on premises: it is in charge of monitoring, managing the nodes and the maintenance and deployment of microservices (i.e. applications, functionalities...) on these nodes.
- Communications: as a key component in the solution, it makes reference not only to the communications between the node and the management system, but also to inner communications in the node (between applications), communications with local devices in the SS, such as the data concentrator, OLTC transformer, etc. and communications with external devices such as smart meters, protections, energy storage etc.

Figure 9 shows, in general terms, the architecture for applying edge computing at a SS and the relationships between each of its components.

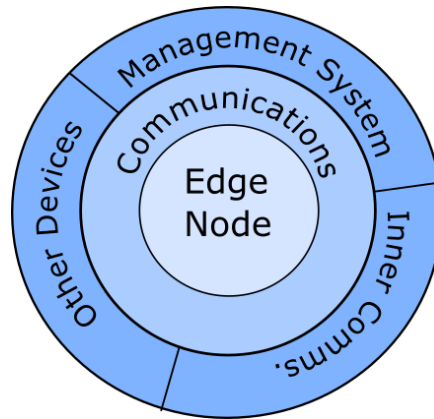


Figure 9. Overall representation of the architecture.

Each of these three main elements has, at the same time, multiple parts that can vary from one solution chosen to other. For the Edge Node these parts would be:

- Hardware: not only the hardware must have enough computing capacity but comply with the electrical requirements to be installed in a SS (e.g. electrical isolation).
- Security elements: Mechanisms that guarantee that information received at the management system is the same as the sent by the Edge Node.
- Operating System.
- Virtualization technology: it can be based on VMs, containers (dockers), both at the same time, Unikernels... Whatever the virtualization technology chosen, to be integrated with the existing devices in an i-DE's SS, it must contain specific applications to “understand” the data that the Edge Node receives through:
 - DLMS
 - STG-DC interface (i-DE)
 - Modbus. For MV/LV automation, transformers, etc.
 - Zigbee. To receive information from sensors or send orders to circuit breakers.
- Database: Between relational and non-relational, it must be determined which type adapts better to the data formats and requirements in the node.

For the communications layer, as previously mentioned, three environments can be distinguished:

- Communications with the Management System.
- Inner communications between applications in the Edge Node.
- Communications with other devices. Inside this type, it could be distinguished between devices in the SS (e.g. data concentrator, low voltage supervisor, etc.) and devices out of the SS (e.g. smart meters, breakers out of the SS, etc.).

Finally, the Management System is usually a proprietary software of the supplier, that is to say, it does not depend very much on open source software. For example, Minsait has its own platform, “Onesait Things Platform” [Edge00b], so does Nebbiolo Technologies,

with its “fogSM” [Nebb19]. Nevertheless, independently of the platform used, the Management System should have the following basic features:

- Authentication of devices. To determine that the data is being received from (or sent to) an authorized node.
- Cybersecurity. It must guarantee that the data has not been altered by third parties.
- Registry/Catalogue of applications that can be deployed remotely, together with their configuration files.
- A Graphical User Interface (GUI) so that the Management System is more user-friendly and accessible for a wider range of users.
- Database to store the data sent by the nodes and accessible through the GUI.

7.3 Advantages, Challenges and Functionalities

The implementation of edge computing at secondary substations have several advantages and cases of use. However, the challenges it poses are not few; if the final solution proves to be less scalable or flexible than expected, its viability will be questioned. The functionalities that can be carried out by the solution must also prove to have benefits in the operation of the LV grid that can be translated into economic savings/incomes, not only for the utility but for the entire system. The solution itself should prove to save money and time respect to traditional procedures (discussed in section 8. Economic Impact)

The main **advantages** that are expected from this solution are:

- As data is processed close to where it is generated, time between data collection and the decision is reduced.
- Gives SS autonomy as it relies on local communications. If, for whatever reason, communications between the SS and the central system fail, some functionalities and processes would still be guaranteed.
- The centralized management system makes it simple to deploy applications in the different nodes, avoiding the need of a technician going to the place to upload applications.
- Virtualization and/or dockerization makes it simpler to correct bugs in applications. Applications are independent between them; if one of them has errors, it can be stopped and restarted, once its code has been debugged, without the fear of having altered other functionalities.
- Time for the deployment of applications (functionalities), and their updates, is significantly reduced. It makes it easier to apply the Agile methodology during development, as applications can be tested in a real environment faster.

- Depending on the SS, some applications might be useful or not. “Customization” of functionalities based on the SS.
- Applications providers are decoupled from hardware providers. It is expected more competitiveness for the development of functionalities. Related to the previous point, this would also provide flexibility, since not only the functionalities could be “customized” depending on the SS, but also the hardware where they will be installed.
- Ultimate improvement of quality of service. Functionalities such as voltage control or control of circuit breakers have a direct impact on quality of service
- In the future, the edge computing device is expected to substitute some of the devices that are present at SSs (DCs, low voltage supervisors...). The expected flexibility of the architecture would ease this process.
- If various SSs are connected between them (same LAN through optical fiber or PLC), it would be possible to make them work together for a process that has higher computing requirements (e.g. complex machine learning models, image processing, etc.)
- It can be the driver for the development of innovative and disruptive functionalities.

On the other hand, as every solution, it will have some **disadvantages and challenges** to be faced:

- Initially, this implies a new device to be installed at the SS. Problems of space might appear, although it is not believed to be a major challenge.
- The computing capacity of the edge computing device: if excessively high, it would be quite expensive for its deployment at several nodes; if excessively low, some applications might not run properly, cause significant delays or the number of functionalities might be significantly limited.
- Some disruptive functionalities are not regulated yet. Functionalities related to demand response, energy storage at distribution level or electric vehicles need to be clearly ruled by the regulatory body yet.
- Some functionalities require data provided by smart meters. Currently, this data might be incomplete, inaccurate, or even unavailable due to temporary disconnections of smart meters. If the Edge Node is expected to carry out local data processing, these communications with other devices must be improved in reliability and speed. Meanwhile, pre-processing algorithms may be needed in order to select the appropriate sources of data (the most reliable) to be used as input in other processes.

- As edge computing in SSs is not very extended, initially, it might result challenging to find the hardware for the Edge Node, since it must pass numerous tests to be installed in a SS.

Regarding the possible functionalities of the edge-computing-based solution, they cover different areas of an electric distribution utility.

Grid control:

1. Low voltage control through actions to the OLTC transformer, using voltage measurements from smart meters and low voltage supervisors.
2. Control breakers, switches and other protections based on alarms/events.

Grid knowledge and analysis:

3. Run algorithms to detect which phase each smart meter is connected to. Nowadays, one of the biggest objectives of electric distribution companies is to have an inventory of the connections phase-meter, looking for a balanced system (approximately the same consumption on each phase).
4. Algorithms to determine the impedance of LV lines.
5. Fraud detection algorithms. If the phase is known, these algorithms will be more effective.
6. SS balance, power losses accountability.

Grid monitoring:

7. Fault detection, location and even classification in the LV grid.
8. State estimation of the LV grid.

Asset management and maintenance:

9. Predictive maintenance. Monitoring of indicators (e.g. temperature of transformer, vibrations...). If the risk surpasses a level, send an alarm to the central system in order to take an action.
10. Security functionalities. Movement detectors, thermal cameras, fire detectors, etc.
11. Demand prediction for the transformer. This could be used to plan maintenance actions in advance, minimizing the impact on consumers.

Flexibility and innovative functionalities:

12. Monitoring and control of Distributed Generation (DG). Calculus of the balance between generation-demand.
13. Control of energy storage (e.g. batteries) at the distribution level. To alleviate transformer load during peak hours and, if the subnetwork is net exporter during

just some hours, to avoid reverse power flow. However, the use of energy storage at the distribution level by the utility is not regulated (not allowed) in Spain at this moment.

14. Electric Vehicle (EV) integration.

- Smart charging EVs at night. Optimizing not only the vehicles that are at the same garage, but all the vehicles charged by the same transformer. If the phase is known (no. 3), charging could be optimized to keep a balanced system.
- The solution could enable the so-called vehicle-to-grid (V2G) functionality. Not regulated in Spain at this moment
- New possibilities for EV public charging infrastructure, such as the discussed by [OkOz16], but considering fog computing (edge nodes can interact between them). Great regulation is needed in Spain for this functionality.

15. Demand response schemes. Interaction between the edge node at the SS and a HEMS to take any action. It could be combined with load identification algorithms.

These functionalities constitute only an example of what edge computing at SSs could provide. In all likelihood, this list will grow significantly when the solution gets tested on field successfully.

7.4 Analysis of the Edge Node

7.4.1 Virtualization Technology

As mentioned in previous sections, VMs and containerization are the two main virtualization technologies that are considered as options to support the functionalities of the Edge Node. Explanation of basic concepts has been previously provided in 6.4.

Virtual Machines. The deployment of functionalities using exclusively standard VMs (a complete OS per VM, Figure 6 in subsection 0) would be extremely challenging despite it would provide high level of security and isolation between applications. As first hypothesis, if it is supposed one application per VM, a large amount of computing and storage resources would be wasted within the Edge Node, since each VM would have RAM and storage allocated by the hypervisor not only to run the specific application but to run an entire OS that contains many libraries that will never be used by the application [CDLR19]. As the Edge Node is expected to run multiple applications, this would require multiple VMs that would lead, inevitably, to a large amount of redundant processes. This hypothesis would be highly inefficient [Luci17].

The second hypothesis would be the use of VMs that contain more than one application. Despite this would improve the resources usage, it does not guarantee isolation between applications (unless containerization is added), so this hypothesis can be directly discarded.

Both hypothesis mentioned so far would have the added disadvantages of extended development time of applications [Stro19] and large image size that would make the deployment (Management System to Edge Node) a big challenge [Luci17, Stro19]. Besides, it would be necessary to keep updated the OS of each VM (which implies more than one update per node) to keep it secure.

Containerization (Docker). The deployment of applications as Docker containers would be a lighter alternative to standard VMs. Redundancy of processes is avoided since the containers are run directly on the host OS and some of its libraries may be shared [CDLR19]. This feature, however, hides a drawback: as the OS is shared by multiple containers, a successful cyberattack or vulnerability in the OS will compromise all the containers [Luci17]. On the other hand, two security analysis carried out in [Bui15] and [MRCD18] concluded that the default configuration of Docker is relatively secure. To improve this security, [Bui15] proposes to run the containers without root privileges and to enable security improvements of the Linux kernel. [MRCD18] goes deeper and concludes that a great number of vulnerabilities in Docker result from using containers as VMs (use of an OS as a container, this is not contemplated as a solution for the Edge Node). This same reference, [MRCD18], states that, although Docker is relatively secure from a local point of view, it can be vulnerable from an ecosystem point of view if there

are external intermediaries involved in the deployment of containers (e.g. Docker Hub, public repositories, etc.).

Docker has become very popular among software developers because it is a simple, open source (community version), way of deploying an application anywhere and it significantly reduces the time between development and its use in production. Tools as Docker Compose and Kubernetes, that eases the process of deploying multiple containers simultaneously, have helped in this extraordinary adoption of Docker.

Unikernels. An alternative that has also emerged in the last decade is the use of Unikernels (University of Cambridge 2013-2015 with MirageOS, [Luci17]). Unikernels are similar to VMs but with the difference that Unikernels do not have the entire kernel of an OS and that they are typically run directly on an hypervisor (no need of an intermediary OS) [CDLR19]. That is to say, the application has its own optimized kernel that only contains the processes and operations that it needs to run [MRCD18]. Application code and kernel code are not separated, which makes images even lighter than containers [Luci17] but more difficult to debug. The use of an optimized kernel has two different but coherent security consequences, according to the consulted literature; on the one hand, by removing unnecessary processes and operations of an entire kernel, the “attack surface” is strongly reduced [Luci17]; on the other hand, this removal implies, at the same time, the removal of some security mechanisms that could be used by an OS [CDLR19].

In addition to the lightness, another advantage of Unikernels is the provision of high isolation levels, similar to standard VMs, by the hypervisor [Luci17].

A deep comparison between Unikernels and containers was carried out in [GSAV18] using different programming languages (Java, Go and Python). It concludes that Unikernels are faster than containers at responding (response time) but the performance is similar with heavy workloads (except for Python, where the container outperforms the Unikernel) [GSAV18]. It also shows that Unikernels consume far more memory than containers. The explanation given by [GSAV18] is that the Unikernels applications have the extra code for its own optimized kernel whereas containers use the kernel of the host OS. Therefore, the total memory consumption of a group of Unikernels in the same hypervisor may be less than that of multiple containers together with their host OS.

Final choice. Although Unikernel technology is a good competitor of containerization (Docker) due to its lightness, good performance and isolation capabilities, the technology is still at an early stage of development and the number of implementations in production is very low [MRCD18]. To develop a Unikernel application, it is needed to have great knowledge of Operating Systems (which the average developer does not have) [Eybe19] and its deployment process is also complex, so further research is needed [Luci17].

Therefore, based on this discussion, the recommended technology to be used in the Edge Node is containerization using Docker due to its ease of use and deployment, its high adoption by developers (easy search of support), and its relative maturity. In essence, it satisfies the requirements of i-DE in terms flexibility (i.e. applications can be developed by third parties, containers are isolated so that they can be stop/run/debugged without altering other applications, etc.) and security, which can be improved taking specific measures as already discussed. Docker is also the technology chosen by Minsait for its PoC with i-DE.

However, the use of Docker requires the adoption of additional measures to protect the intellectual property of the applications. For example, if a third party is hired to develop an application, in all likelihood the utility will have to provide some containers so that the third party can test its application integration during the development. To create/run/stop containers, the user must be root-privileged (docker community version), which means that it will also have access to the containers provided by the utility (to their source code...). To tackle this, different approaches can be taken:

1. Adoption of security measures when building the docker images. To avoid the intrusion into the filesystem of a running container, the access to its `sh` and `bash` console should be denied. This is done by adding Code 1 to the Dockerfile that builds the image (source: Minsait).

Code 1. Lines to add to the Dockerfile to remove access to the `sh` and `bash` console in a container. The container gets associated to user 9000. Source: Minsait

```
RUN /bin/rm -R /bin/sh
USER 9000
```

However, if the user is root, by using Docker `export` (Code 2) the complete filesystem of a container would be included in a `.tar` file, including its source code.

Code 2. Use of Docker `Export` to obtain the filesystem of a container in a `.tar` file

```
docker export [container_id] > [file].tar
```

Therefore, some additional measures are necessary. The developer, by default, should obfuscate the code of the application before creating the container (e.g. using `pyarmor`⁵ for Python or `ProGuard`⁶ for Java). Then, in addition to this obfuscation, the container image could be encrypted using the free tool “*containerd imgcrypt*” [Lum19] so that only authorized users can run the container.

⁵ <https://pypi.org/project/pyarmor/> (3/7/2020)

⁶ <https://www.guardsquare.com/en/products/proguard> (3/7/2020)

2. To use the paid version of Docker, Docker Enterprise, whose Universal Control Plane allows the definition of different user roles, with different restrictions and privileges [Mira20]. The integration of Docker Enterprise with the Management System (proprietary by a third party) should be then studied.
3. To use a commercial security tool (e.g. Twistlock⁷) that may have mechanisms to limit user privileges and to add extra security. As in the previous point, its integration with the Management System should be then studied.
4. Non-technical mechanisms. For example, to leave the test process entirely to the utility (not advisable, would delay the deployment of the application), to substitute the provision of the containers by a simulation environment (depending on the data, the creation of this environment could be complex) or to use strong confidentiality agreements and control procedures so that the source code of these containers can be better protected (it would not completely guarantee that the source code has not been inspected).

Obviously, if all these measures were taken, the proprietary source code (algorithms, logic, etc.) of the applications would be extremely secure. Since options 2 and 3 imply an additional cost, a deeper study should be done in the future analysing the additional security and functioning benefits that these could give. Without any doubt, the first option (free) must be done regardless of the adoption of non-technical mechanisms (fourth option) and commercial tools.

⁷ <https://www.twistlock.com/solutions/docker-security/> (3/7/2020)

Table 8. Summary of the comparison between virtualization alternatives

	VMs	Unikernels	Containerization (Docker)
Security	✓ High security level	<ul style="list-style-type: none"> ✓ Reduced "attack surface" ✗ May lack of some security mechanisms that a complete OS has 	<ul style="list-style-type: none"> ✗ All the containers depend on the security of the OS ✓ Relatively secure in its default configuration and with "good practices" ✗ Requires additional mechanisms/tools to guarantee intellectual property of third-party applications
Application Isolation	✓ High isolation level	✓ High isolation level	✓ Good isolation level
Efficiency	✗ Inefficient for multiple applications	<ul style="list-style-type: none"> ✓ Better memory consumption for multiple applications than containerization ✓ Faster response than containers 	✓ Unikernel-like performance with heavy workloads
Size of images	✗ Large image size	✓ Very light images	✓ Light images
Maturity of technology	✓ Very mature	✗ Early stage of development for production	✓ Experience in production environments
Additional information	✗ Multiple OS updates per Node	<ul style="list-style-type: none"> ✓ No need of an entire OS ✗ Still too complex for the average developer 	<ul style="list-style-type: none"> ✓ Very popular and relatively easy to use ✓ Time between application development and production is reduced
Recommendation: Containerization (Docker)			

7.4.2 Operating System (OS)

Since the recommended software architecture is the use of containerization (Docker), the Edge Node must have an OS where the Docker engine can run. The Docker daemon needs some specific Linux kernel features to run, so Docker does not run natively on Windows. To do so, there are two alternatives [Micr19]:

- To run Docker in a full Linux VM, with Windows as the host OS and a type II hypervisor.
- To run Docker using Hyper-V isolation. Hyper-V provides an optimized virtual machine with its own kernel to Docker.

The objective is to run containerized applications in the Edge Node, hence there is no need of using Windows as OS, since it would only add complexity to the deployment. Therefore the OS of the Edge Node should be based on Linux. There are tens of this type of OS (e.g. Ubuntu, Debian, OpenSUSE, etc.) and, as they are open source, their vulnerabilities are rapidly corrected by the community as they arise. It is also possible for the OT staff of the electric utility to patch the OS when required. For this reason, to make the maintenance of the OS easier, it is recommended the use of a Linux OS already in use/homologated by the electric utility. In the case of i-DE, this would be RedHat 7.5 (or higher) or RedHat Oracle Linux 7 (or higher).

Table 9. Summary of the comparison between possible Operating Systems

Microsoft Windows (Windows Server \geq 2016)	Linux OS	
	Ubuntu, Fedora, Debian, etc.	RedHat \geq 7.5 RedHat Oracle Linux \geq 7
✗ Proprietary OS	✓ Open Source	
✗ Docker needs Linux Kernel (A Linux VM or Hyper-V isolation would be necessary)	✓ Vulnerabilities are rapidly corrected by the community (usually)	
	✓ Supports Docker	
✓ Homologated by i-DE	✗ Not homologated by i-DE	✓ Homologated by i-DE
Recommendation: RedHat or RedHat Oracle Linux		

7.4.3 Database

In order to discuss which database would be more appropriate to install in the Edge Node to store the data, it is necessary to define how the data would be structured. It was mentioned in 6.1 (Overview of current infrastructure) that the STG-DC interface uses XML format. Although this format is useful to represent resources, it is a bit complex to understand and requires more time to be processed than other formats such as JSON (JavaScript Object Notation). JSON format is one of the most used formats in IoT architectures to represent resources and for information exchange. It is based on key-value pairs and is also shorter and easier to be read by humans and machines.

As the Edge Node will receive data in different formats (or no format at all) from different devices, it is necessary to establish a common format to be used by the applications. The JSON format is proposed for this purpose given its extended use in modern IoT architectures and its simplicity. However, this means that every data received must be “translated” to JSON (e.g. data received by STG-DC interface should change from XML to JSON) before its storage or use by the applications. In fact, one of the objectives of i-DE and the rest of electric distribution utilities (FutuRed working groups) is to define a common JSON schema for these messages based on the Web of Things Architecture (WOT-A), sponsored by W3C [KMLK20]

Table 10. Examples of JSON and XML formats. Dummy data of a LV Supervisor

JSON	<pre>{ "supervisor": { "id": 1, "date": "2020-06-19T17:09:36.261Z", "activepower": 2524.53, "type": "triphasic", "voltage": 235.39 } }</pre>
XML	<pre><measurements> <supervisor> <activepower>2524.53</activepower> <date>2020-06-19T17:09:36.261Z</date> <id>1</id> <type>triphasic</type> <voltage>235.39</voltage> </supervisor> </measurements></pre>

SQL vs No-SQL. The first decision related to the database is which type suits better for the Edge Node, SQL or No-SQL (i.e. relational or non-relational).

SQL databases (e.g. PostgreSQL) requires data to be structured in form of a table. It is relatively easy to traduce a JSON file to a table: each key would be a column and each value would be inserted into its corresponding column. The problem is that the format of the table must be predefined and additional columns suppose a significant change in the table. For example, let's suppose that it is required that all the data sent by a LV supervisor has to be stored in a table called "*supervisor_data*", but not all the measures are sent at the same time (e.g. current and voltage every five minutes and power measures every fifteen minutes) or that other type of related data has to be stored (e.g. an event, an alarm...). This would mean that not all the columns are filled for every row, but every row would still have a value for those "empty" columns (null value). This would be extremely confusing when querying data from the table. Therefore, our intention of having all the data related to the LV supervisor in the same table would have to be discarded and create different tables, which can also be confusing. Nevertheless, an alternative would be the insertion of the JSON document in a table where a column has 'JSON' as its defined data type. Although functional, this approach is not rational since there already exist No-SQL databases optimized to store documents like JSON (e.g. MongoDB). Furthermore, SQL databases are difficult to scale [JPAK12] (the Edge Node will have to store millions of measures per month) and require more time to query data for semi-real time applications [CeME15].

On the other hand, non-relational databases can manage huge amounts of data better and the existence of different subtypes (as discussed in 6.7.2 Non-relational databases) increases the range of possibilities. Given the nature of the data, which is time-related (i.e. every measure or event is associated to the time when it was acquired) and its format (JSON), two subtypes of No-SQL databases are considered to better manage this type of data:

- Document Oriented Database: According to DB-ENGINES ranking [Db-e00] (June 2020), the most popular document database is, by far, MongoDB, which is open source and has a community version (free). MongoDB is specifically designed to store JSON documents and it provides complete ACID transactions like relational databases [Ref00b]. It is available as a Docker image (i.e. it can be used as a container).
- Time Series Database: The DB-ENGINES ranking [Db-e00] places InfluxDB (open source, free community version) as the most popular time series database, followed by Kdb+ (proprietary) and Prometheus (open source, free). For this project, only the free alternatives will be considered (InfluxDB and Prometheus) as the electric utility must try not to depend on a proprietary solution, since once the investment was made on licenses, it would be difficult to change the used database in case a better alternative appears in the future. Both InfluxDB and Prometheus are also available as Docker images.

MongoDB vs Time Series Databases. The first dilemma to solve is between MongoDB and time series databases (InfluxDB or Prometheus). In order to make the comparison, MongoDB v4.2.7 and InfluxDB v1.8.0 (because of its popularity) were installed as Docker containers in the test environment (6.8 Local Test environment). The Docker image of InfluxDB and MongoDB weights 304 MB and 388 MB, respectively.

As it is obvious, MongoDB and InfluxDB store data in a different way. In MongoDB, although the GUI shows the data (contained in a collection) structured as a table (Figure 10), each row is only the representation of a JSON document, so different types of JSON documents can be stored in the same collection (i.e. not having the same fields/keys does not suppose a problem like in relational databases, as it was mentioned previously). Each document gets an automatic id (“_id” column in Figure 10) that is related to the time when it was stored.

_id	meter_id	date	power_r	power_s	power_t
5ed4b4c7684a8d0015a5d724	1	2020-06-01T09:56:55.027Z	4783.914794857038	7640.087069407082	2128.665337284945
5ed4b4cc684a8d0015a5d725	1	2020-06-01T09:57:00.036Z	3095.818960859915	7007.474571725925	8637.359649130278
5ed4b4d1684a8d0015a5d726	1	2020-06-01T09:57:05.041Z	771.0462595524525	2258.7055037832647	1044.2935482460468
5ed4b4d6684a8d0015a5d727	1	2020-06-01T09:57:10.045Z	794.7582234435835	1413.315786495153	4469.267613276582

```

1 {
2   _id: ObjectId('5ed4b4c7684a8d0015a5d724'),
3   meter_id: 1,
4   date: '2020-06-01T09:56:55.027Z',
5   power_r: 4783.914794857038,
6   power_s: 7640.087069407082,
7   power_t: 2128.665337284945
8 }

```

Figure 10. Screenshot of MongoDB GUI (mongo express) and how it stores the JSON documents.

In InfluxDB, data is also shown in form of a table, but instead of collections, the entries are associated to a tag or “measurement” (InfluxDB notation) so, as in MongoDB, it is not necessary that every entry has the same fields/keys, as long as they get grouped under a different tag/measurement. Figure 11 shows data stored under the measurement “meter1” in a sample database. In InfluxDB, instead of _id, each entry is associated to a number that indicates the time when it was inserted (“time” column in Figure 11), which is more intuitive than the alphanumeric _id of MongoDB.

```

> select * from meter1 order by desc limit 5
name: meter1
time                date                meter_id power_r            power_s            power_t
-----
1592579304000000000 2020-06-19T17:08:24.084Z 1          3649.2087543811213 8654.226011150584 8427.755044891572
1592579297000000000 2020-06-19T17:08:17.057Z 1          552.7521105900202 7133.0396744699565 267.87174594933515
1592579290000000000 2020-06-19T17:08:10.025Z 1          2580.3182588511618 7011.433360701755 4123.001875887986
1592232774000000000 2020-06-15T16:52:54.390Z 1          3237.33831616597 650.1641890553889 5932.295647254855
1592232769000000000 2020-06-15T16:52:49.384Z 1          378.59710089802377 1860.4108718353384 3887.8831098507403

```

Figure 11. Screenshot of a query in InfluxDB that shows how data is stored

In terms of how the data is stored, neither MongoDB nor InfluxDB represent a challenge for its integration in the Edge Node: in MongoDB the JSON would be directly stored and, in InfluxDB, the information contained in a JSON can be easily extracted to be inserted in the database. However, one characteristic of InfluxDB that may be an advantage is that queries can be done using a SQL-like language (see Figure 11), whereas MongoDB has its own query language [Ref00b]. Although this query language might not be difficult to learn, the utility's staff may be more familiar with SQL and more support is available on the internet for this language. An additional interesting feature of InfluxDB is that a tool called Telegraf (open source) can be used to directly insert the data received through AMQP or MQTT (inner communications of the Node) into a database, depending on the topic/queue.

Regarding the performance and requirements, these constitute the most decisive factors to choose the database. A comparison made in [Chur18] (v1.7.2 of InfluxDB and v4.0.4 of MongoDB) shows that InfluxDB is 2.4 times more efficient at writing data, 5.7 times faster querying and uses 20 times less disk space (better compression of the stored data). In a container environment, Figure 12 shows general performance stats (Docker stats). Memory usage is similar for both databases, but the percentage of CPU used is around 6 times more in MongoDB and the number of processes (PIDS) initiated are also higher.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	BLOCK I/O	PIDS
ed85c328f79d	influxdb	0.09%	72.48MiB / 2.87GiB	2.47%	63.8MB / 1.25MB	10
8108f762bae6	mongodb	0.61%	73.2MiB / 2.87GiB	2.49%	57.5MB / 1.2MB	37

Figure 12. Screenshot of Docker stats showing the consumption of InfluxDB and MongoDB in Docker containers.

In the CAP model (explained in 6.7.2 Non-relational databases), MongoDB is originally considered CP: it is consistent and partition tolerant. To increase the availability of MongoDB, replica sets should be used (at least three MongoDBs installed in the node). Following a linear relationship, this would mean 3 times more percentage of CPU and memory usage in Figure 12 and, consequently, 60 times more disk space usage than InfluxDB (based on [Chur18]). InfluxDB, on the other hand, is a mix of CP-AP that does not need replicas [Pato16].

Final choice. From the previous discussion it can be clearly extracted the conclusion that a time series database like InfluxDB should be used to store the data in the Edge Node. Prometheus, the alternative to InfluxDB also mentioned, is more focused on monitoring applications, whereas InfluxDB is better for sensors (metrics, events, time series data) and data analytics (supporting programming languages such as R or Python) [NaAb19]. Given the popularity of InfluxDB and these facts, a deeper comparison between these two open source time series databases is not necessary in order to assure that InfluxDB would suit to the Edge Node (requirements and functioning) better. The PoC of Onesait for i-DE also chose InfluxDB as the database to be used (eWLCA, [OSMC20]).

Table 11. Summary of the comparison between Database alternatives

	SQL (e.g. PostgreSQL)	No-SQL	
		Document-Oriented (MongoDB)	Time Series (InfluxDB)
Storage Format and data insertion	~Based on tables. Needs JSON-to-table conversion or definition of JSON as a data type for a field	✓Specifically designed for JSON documents	✓Designed for metrics, events...
	✗Table fields must be predefined		✓Compatible with JSON
			✓Compatible with MQTT and AMQP through Telegraf
Performance	✗Difficult to scale	✗Test shows 6 times more CPU consumption and 4 times more open processes than InfluxDB (in Docker)	✓Better efficiency, data compression and higher speed than MongoDB (external comparison)
	✗Slow queries for semi-real time applications		
ACID properties / CAP model	✓Generally provides ACID transactions	✓Provides ACID transactions	✓Mix of CP-AP (CAP model)
		✓Is considered to be CP (CAP model)	
		✗Needs at least 3 replicas to have Availability (CAP model)	
Type of license	✓Open source and proprietary alternatives	✓Open source (free community version)	✓Open source (free community version)
Query Language	✓SQL	✗Own query language	✓SQL-like query language
Maturity of technology	✓Very mature. Some were created more than 20 years ago (e.g. PostgreSQL)	✓Very mature (2009)	✓Mature (2013)
Docker availability	✓Most available as Docker image	✓Available as Docker image	✓Available as Docker image
Additional Information	✓Historically very used. Easy to find support and staff is typically familiarized with this type of databases	✓Most popular Document-oriented database (DB-ENGINES)	✓Most popular Time series database (DB-ENGINES)
Recommendation: InfluxDB			

7.4.4 Hardware

Choosing the right hardware is one of the most difficult parts of applying edge computing in secondary substations. Some Edge platform vendors have developed their own specific device whereas other vendors are more flexible and use a third-party device. Secondary substations can be considered as high-risk installations due to the voltage, the devices it contains and the temperatures that can be achieved (e.g. summer in the south of Spain). The requirements and tests that must be satisfied by any device to be installed in an i-DE's SS are shown in ANNEX I. Hardware Requirements (insulation, radioelectric disturbances, immunity, electrical tests, mechanical tests, and climatic tests). These tests should be carried out by a laboratory that provides certification of compliance. In addition to this, the device should not exceed the dimensions 220x140x130 mm (width x height x depth) if it is going to be included in the electric cabinets currently deployed by i-DE.

Some of the main industrial computer manufacturers are Advantech, Lanner, Kontron and Axiomtek.

7.5 Communications

As previously said, three main types of communications can be distinguished in the Edge Node, inner communications between applications, communications with the Management System and communications with other devices (in and out of the SS).

7.5.1 Inner Communications

The decision of using containerized applications also involves the need of an inner communications protocol. In the traditional approach, where software is embedded in hardware, the device would receive the necessary data and, either would be immediately used, or would be stored to be used later to provide the functionality. In this new solution, the Edge Node receives a large amount of data from the data concentrator, smart meters, sensors, etc. but not all the data is relevant or will be used by every application. Therefore, it is needed an inner communications protocol, to which every container (application) has access, in order to provide to each application the data it needs to work. This provision of data could be in the form of the necessary raw data, a message indicating where the data is stored, etc.

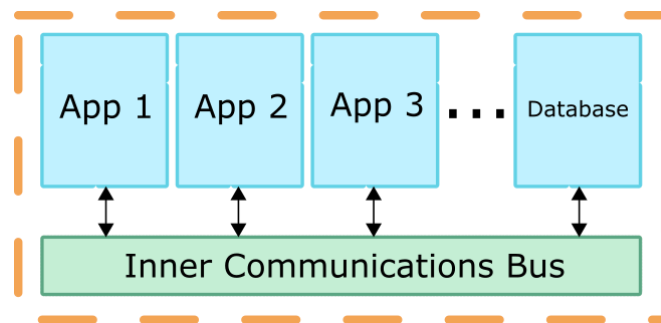


Figure 13. Simple general representation of the inner communications in the Edge Node

There are tens of communication protocols designed for IoT architectures that could be considered to be used for this purpose (except wireless protocols). In this project, only MQTT (v3.1) and AMQP (v1.0) protocols will be considered, as they are among the most used in IoT (they are light), have the support of large companies, use TCP as the transport protocol and are free to use.

As presented in subsections 6.2 and 6.3, both protocols do not put restrictions on the format of the message, so it could follow a JSON structure (very used to represent objects and related information in IoT) or any other message structure. As they both are based on publish/subscribe using an intermediary broker, the configuration of the communications is rather simple for both protocols.

As only internal applications (dockers) are expected to be communicating through this inner broker, at first, this does not have to be accessible from the outside world, that is to say, the IP address of this broker is the localhost (with ports 1883/8883 for MQTT or 5672/5671 for AMQP).

It is already known that the MQTT protocol is lighter than AMQP. However, to put numbers to that difference and to effectively compare these protocols, both MQTT and AMQP have been tested in the configured Test Environment (6.8 Local Test environment). In both cases, the brokers have been installed as Dockers: Mosquitto broker in the case of MQTT and RabbitMQ in the case of AMQP. The test consisted on publishing four dummy messages (JSON structured, same length each, without using TLS) simultaneously on four different topics/queues every five seconds. Obviously, in the real Edge Node, messages will be more complex, and the frequency of messages may be higher, but the objective of this test was to make a comparison between MQTT and AMQP working at the same time. The results of this test are shown in Figure 14 (in both cases the messages were always correctly received by the subscriber). These are obtained using Docker stats [Ref20].

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
6290cdd07982	edge_amqp	0.57%	87.36MiB / 2.87GiB	2.97%	91.9kB / 40.3kB	41.5MB / 602kB	86
642db35d3a29	edge.mqtt	0.06%	916KiB / 2.87GiB	0.03%	22.7kB / 6.04kB	3.37MB / 0B	1

Figure 14. Screenshot of the MQTT-AMQP comparison test results.

Figure 14 shows that MQTT uses 10 times less CPU and 100 times less memory than AMQP. This extreme lightness is the reason why MQTT is so used in IoT, it can run in almost any device and it works well. The column NET I/O indicates the amount of data sent (input) and received (output) over its network (it is not static, it increases every five seconds in this test) [Ref20]; although the messages are the same in both protocols, the extra load added by AMQP produces this difference between NET I/O values. The simplicity of MQTT is also shown in the number of processes (PIDs) of the container: AMQP creates 86 whereas MQTT only creates one.

A deeper benchmarking of these protocols was carried out in [Naik17]. It determines that AMQP is more interoperable than MQTT although it has a lower level of reliability since it only has two quality of service levels versus the three levels that MQTT offers. AMQP is also more standardized and secure than MQTT [Naik17]. Despite MQTT supports TLS/SSL security, its authentication features are very poor (based on username & password). On the other hand, AMQP supports different ways of including TLS (*Single-port TLS Model*, *Pure TLS* and *WebSockets Tunnel TLS Model* [Naik17]) and SASL (for authentication purposes).

If the containers deployed in the Edge Node are all authenticated and validated, and the cybersecurity of the node is appropriate, there is no need of using advanced security features for inner communications if the broker remains in the localhost, since they would increase the weight of the inner communications and the complexity.

It must be remembered that the main objective of the Edge Node is to provide functionalities, hence the more CPU and memory available for these, the better.

Considering this, the main requirements that the inner communications protocol must have are reliability and lightness. These two requirements are satisfied by MQTT better than AMQP. Besides, the MQTT functioning is a bit simpler than AMQP. For example, it does not need to create the topic -queue in AMQP- previous to publication in that topic. It also allows the use of wildcards to subscribe to more than one topic at the same time, which in AMQP would be done using different types of predefined exchanges or using multiple lines of code.

Final choice. Although AMQP would be a valid protocol to implement the inner communications in the Edge Node, it is recommended the use of MQTT since it complies better with the requirements of the Edge Node and it is also simpler to configure, while being able to provide basic TLS in case it is needed in the future. MQTT is also the protocol chosen by Onesait to be used in the inner communications of the Edge Node in the PoC with i-DE.

Table 12. Summary for the selection of the inner communications protocol.

Main Requirements	MQTT	AMQP
Message format compatibility	✓ No restrictions in format of the message	✓ No restrictions in format of the message
Simplicity	✓ Very simple	✓ Simple
Availability of a broker as a Docker container	✓ Available	✓ Available
Lightness (comparison test)	✓ 916 KiB memory usage	✗ 87.36 MiB memory usage
	✓ 1 process	✗ 86 processes
	✓ 10 times less CPU %	
QoS Levels (Reliability)	✓ 3 levels	✗ 2 levels
Recommendation: MQTT for inner communications in the Edge Node		

7.5.2 Communications with Management System

The communication protocol to be used in the bidirectional communications between the Edge Node and the Management System will also be discussed between MQTT and AMQP. For this purpose, it is considered that the broker is hosted by the Management System and not by the node, so more available CPU and memory can be expected. The reasons why it is better to host the broker in the central system (Management System), apart from the aforementioned CPU and memory savings in the node, are related to simplicity and security:

- **Simplicity.** Both the Management System and the Edge Node will be clients of the communications broker. If the broker is hosted by the Edge Node, it would be the client of thousands of brokers, which is not an efficient approach in terms of management (thousands of IP addresses to connect to).
- **Security.** When the broker is hosted by the central system, it is easier to provide security to the communications (e.g. TLS) and to the broker, since its only one IP address which is accessible and not thousands (reduction of “attack surface”).

The requirements for the communications with the Management System are significantly different from the inner communications ones. The lightness of the broker running in the central system is no longer very relevant; any utility’s OT system would not have any problem to allocate resources to the broker. The relevance taken by the size of the messages depends on the communications infrastructure between the SS and the central system. One of the main aims of applying edge computing in SSs was to increase processing capacity at the SS so that the amount of data sent to the central system could be less or less frequent, hence the extra load added by the studied communications protocols should not suppose a major problem for the infrastructure. What is especially relevant is the security. The clients of the broker must be authenticated (have permission) so no unauthorized client can publish/subscribe to topics/queues in the broker. The confidentiality and integrity of the exchanged messages must be guaranteed as well.

Final choice. Considering these requirements and the characteristics of MQTT and AMQP discussed in the previous subsection (7.5.1), AMQP can constitute a better option than MQTT for this purpose:

- Security is at its core by allowing the configuration of TLS and SASL.
- The functioning of AMQP may be more appropriate. In AMQP both the exchanges and the queues must be created before publishing any message (otherwise, it raises an error, Figure 15). This implies that the utility must create a list of queues and exchanges, and the relationships between them. It obviously requires a great methodology and organization, but in exchange it gives control over the communications broker to the utility. No unnecessary queues and exchanges could be created without the express intention of doing so.

```
pika.exceptions.ChannelClosedByBroker: (404, "NOT_FOUND - no queue 'measurements' in vhost '/'")
```

Figure 15. Screenshot of the error when trying to publish/subscribe to a non-defined queue ('measurements') in the AMQP broker.

- Another unique feature of AMQP is that it supports a request/reply (Remote Procedure Call, RPC) pattern, which can be used by the central system to obtain results from an Edge Node on demand.
- The messages in AMQP are compatible with those messages that would be used in the inner MQTT broker of the Edge Node. The content does not need to change its format.

In the PoC developed by Minsait for i-DE, the protocol used is TLS-MQTT which, although acceptable, differs from the recommended in this work for the communications between the node and the Management System.

Table 13. Summary for the selection of the protocol for communications with the Management System

Main Requirements	MQTT	AMQP
Message format compatibility	✓ No restrictions in format of the message	✓ No restrictions in format of the message
Security	✓ TLS/SSL	✓ TLS/SSL
		✓ SASL (Authentication)
Control over topics/queues	~ Topics are created as they are used.	✓ No unnecessary queues and exchanges can be created unintentionally
Other characteristics	✓ Publish / Subscribe. New MQTT v5.0 (2019) includes request/response feature	✓ Remote Procedure Call (sort of request/response)
		✓ Publish / Subscribe
Recommendation: AMQP for communications with the Management System		

7.5.3 Communications with other devices

The Edge Node, at least, initially, is thought to be a data processor, not a measuring device, although it should be able to provide this functionality in case the utility requires it. It needs to communicate not only with the measuring equipment but also with the actuation equipment (e.g. tap changer of an OLTC transformer, a breaker, etc.), whether they are out or in the SS.

Although the electric utility might modify, in the distant future, the current communication protocols used by the smart meters or by the DC (e.g. the STG-DC interface) to better communicate with the Edge Node, the definition of these is out of the scope of this work. Nevertheless, it must be remarked that, in the case of STG-DC, the new protocol should keep the transactional control currently provided by the STG-DC interface.

Since edge computing is still not applied massively in i-DE's SSs, the solution must be tested for a period of time in parallel with the current infrastructure, which means that the Edge Node must be configured to "understand" the protocols currently in use. In other words, it is considered that the Edge Node has to adapt to the existing devices and not the opposite, at least while the solution is under test. This can be done using containers (protocol adapters) dedicated to Modbus, STG-DC, DLMS, Zigbee, etc. that convert the received information into a more human-readable format, such as JSON, and inject it (publish it) into the inner MQTT communication broker (and into the database of the node) to be used by the applications containers (Figure 16).

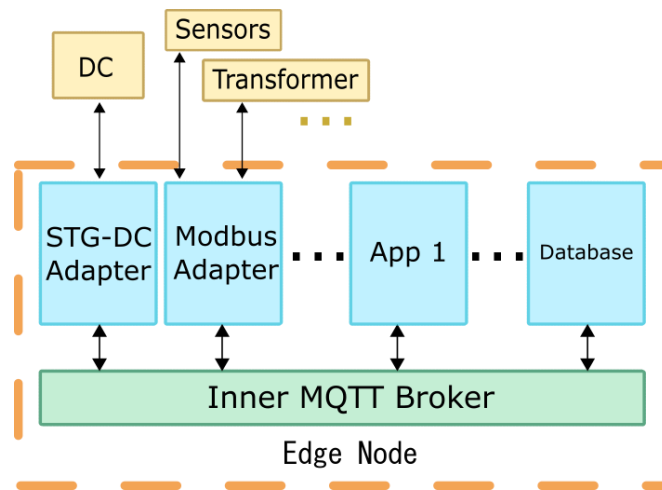


Figure 16. Diagram of the integration of the Edge Node with already in-use communication protocols.

Regarding the devices (mainly out of the SS) that have not been massively installed yet (e.g. HEMS devices, DG inverters, EV charging points owned by individuals, etc.), their future connection to the Edge Node in its corresponding SS should be considered, to take full advantage of the functionalities that this solution could provide (7.3 Advantages, Challenges and Functionalities). The communication standard to be used will definitely

depend on the environment of these new devices (e.g. coverage, internet availability...). MQTT is the protocol mainly chosen by researchers to carry out tests and experiments for this type of devices, and it is definitely one of the best options for communicating with the Edge Node; the same inner MQTT broker could be used by allowing external connections (taking security measures) or an additional broker for these communications (maybe, the best option to avoid inferences with inner communications) could be deployed in the node without any problem thanks to its lightness.

Nevertheless, the choice of the communication protocol for this purpose will require the consensus of the main manufacturers and distribution utilities. In other words, a common and normalized protocol should be agreed to make the deployment of these devices, and their integration with this Edge architecture, easier.

Table 14. Summary for the selection of the protocol for communications with new devices

Main Requirements	MQTT	AMQP
Lightness (Broker is expected to be in the Edge Node)	✓	✗
Companies Support	✓	✓
License	✓ Open Source	✓ Open Source
Security	✓ TLS/SSL	✓ TLS/SSL
		✓ SASL (Authentication)
✓ Consensus between manufacturers and distribution utilities is necessary		
✓ Most researchers choose MQTT for communications with new devices		
Recommendation: MQTT for communications with new devices		

7.6 Management System

The Management System is the part of the architecture that interacts with the Edge Nodes to deploy (or retrieve) functionalities and to receive information from these in a secure and simple way. There are tens of Edge platforms in the market that have different purposes, application areas, limitations, characteristics, etc. Most of these platforms are thought for architectures that may differ from the eWLCA, taken as reference in this work.

7.6.1 Criteria for vendor selection

MachNation, a company specialised in testing IoT platforms, publishes every year a report called “*IoT Edge Scorecard*” whose executive summary is publicly available for free [Hilt20]. The scorecard of 2020 studies 11 different IoT Edge platforms (Onesait platform, the one developed by Minsait, is not included) and categorizes them according to four main aspects [Hilt20]:

1. The capacity of processing edge data.
2. The capacity of managing edge devices.
3. The proposed architecture and how it integrates.
4. The strategy and business of the vendor.

This scorecard shows that most of the vendors who already have their Edge platform ready for the market (good performance in aspects 1, 2 and 4 of the previous list) are also the ones who are more focused on the software capabilities than on the hardware that is deployed, which proves that the complexity of implementing edge computing is generally more related with software than with hardware.

The analysis carried out in previous sections constitutes a good evaluation of aspect no.3 of the eWLCA for its integration in a SS of i-DE.

The main characteristics, according to MachNation [Toka17], that an IoT edge platform should prove to have in order to lead the market are also the characteristics that better suits i-DE’s requirements to implement edge computing at SSs:

- Support of different communications protocols so that different sources of data can be used. As discussed in subsection 7.5.3, the eWLCA would be able to cover this by using protocol adapters containers.
- The Edge compute device (Edge Node) should be able to work autonomously offline. The communications between the management system and the device (subsection 0) should only be focused on specific orders and on final reports sent by the Node.
- The management is based on cloud. For some industries, a third-party cloud provider might be a good option, but, in the case of electric utilities, the

management system should be on premises so that the cybersecurity does not depend on a third-party.

- The architecture is independent of hardware and scalable. The eWLCA complies with this fact (so does, therefore, Minsait’s solution for i-DE). However, some requirements must be considered for this hardware in order to be installed in a SS (subsection 7.4.4).
- The platform provides tools to analyse and visualise the data.

In addition to these main characteristics, the electric utility should consider the following features when choosing the platform:

- Experience of the vendor. The electric utility will trust more those vendors who have already worked on other projects with the utility or that have proven to deploy a similar system in a comparable industry (e.g. gas distribution).
- A user-friendly GUI. The deployment of application and other actions in the Edge Nodes must be easy to do through a GUI.
- Programming Languages for applications. As the analysed eWLCA uses Docker as virtualization technology, any programming language (compatible with the MQTT broker or with the database). can be used to develop a containerized application It is important not to reduce the options to just one language like in past projects (see State-of-the-Art).
- Integration with central system. Edge computing is not a substitute of the computing carried out at the central system (i.e. “cloud” computing). Therefore, the management system should have mechanisms to “share” the data with other processes carried out in the central system and vice versa.
- Security measures. How the platform guarantees that an authorized device is receiving or sending the correct information.

7.6.2 Remote access to the Edge Node

The Edge Node, from a computing point of view, is a remote server. When deployed, the electric utility should protect it using a restrictive firewall, since an undesired access to the node can compromise, not only data, but the security of the LV grid as functionalities are implemented.

Typically, system admins use SSH to access remotely to a server in a secure way, using TCP port 22, what is known as an SSH tunnel, in which the information that circulates is encrypted. However, the use of a firewall makes it difficult to connect this way. In this case, reverse SSH must be used. Briefly explained, reverse SSH consists on the remote server establishing a connection with the local computer (computer in the central system) and waiting for the local computer to request an SSH connection through the same port. Once this request is done, the local computer creates a new connection with the remote server, a “reverse” connection [Mcka19].

7.6.3 Workload orchestration and containers deployment

The most widely used tool to orchestrate containers is Kubernetes (see 6.5 Kubernetes), which is also one of the technologies, together with Docker Compose, defined by the eWLCA for this purpose [OSMC20].

Kubernetes, as previously explained, is useful when the nodes are expected to work together for a high-computing application and when the nodes are able to communicate between them. The exchange of information between SSs is possible thanks to the level 2 aggregation (6.1 Overview of current infrastructure), but the architecture would need an extra node (“master” node) to orchestrate the workloads of the “worker” nodes (Edge Nodes) with Kubernetes, which would result in a fog computing architecture (such as the one mentioned in the State-of-the-art) which differs from the one followed by Minsait in its PoC and studied in this work.

Therefore, the tool used by Minsait in its PoC with i-DE is Docker Compose. Its functioning is rather simple: it deploys all the containers listed in a `docker-compose.yml` file accordingly to the instructions defined in the same file. These instructions would include specific configuration parameters for the services, such as users, passwords, default MQTT topics, CPU limit, etc. The definition of these parameters can become quite complex for unexperienced users so, ideally, the Management System should have a friendly GUI to automatically generate the `.yml` file.

A simplified diagram of the deployment of applications in an Edge Node is shown in Figure 17. This process is the one followed for Test #2.

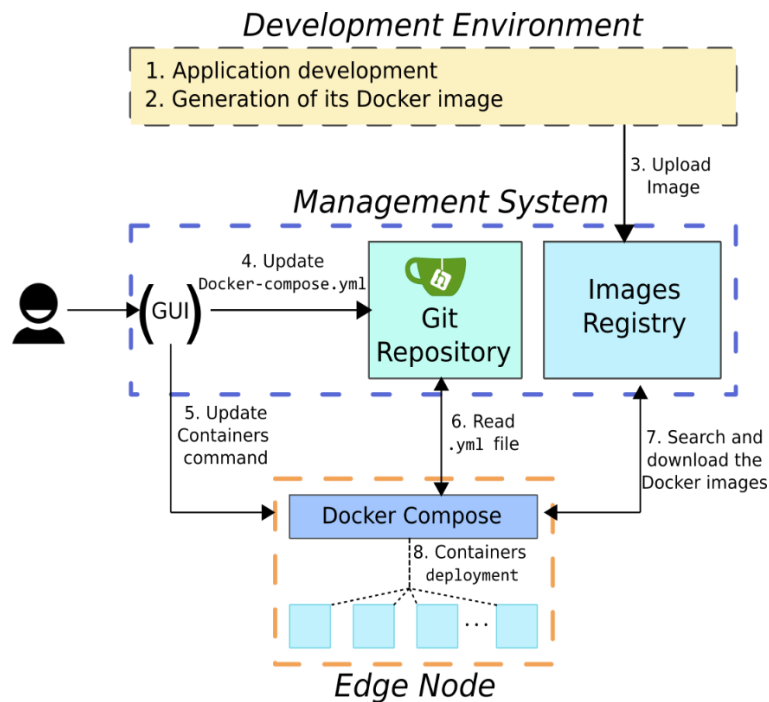


Figure 17. Simplified diagram of the application deployment process. Icon source: <https://icons8.com/>

8. Economic Impact

In this section the economic impact of applying the analysed edge computing solution is briefly discussed, not only in terms of the possible savings and benefits that the new approach would suppose compared to traditional procedures, but also in terms of the benefits that could be obtained from some possible functionalities. The values obtained in this section are only an estimation and may differ significantly under different assumptions or input data, so this discussion must be interpreted as a collection of some possible benefits and savings that the electric utility and the regulatory body should take into account to implement this new paradigm that edge computing poses.

In Spain, the electric distribution is a regulated sector that is decoupled from generation and retail. The pertinent regulatory body in Spain is the CNMC (National Commission of Markets and Competence), which is in charge of establishing the methodology to calculate the retribution (and penalties) of distribution companies based on investments, operational costs, quality of service and improvements.

In its last circular ([Cnmc19], dated December 2019), the CNMC defines three types of investments to be retributed. Figure 18 summarizes them.

Type 0	<ul style="list-style-type: none">• New installations (complete cost).• Renewal of an installation ($\geq 85\%$ of the reference cost).• Justification of actions whose cost is 150% the reference cost.• Retributed according to the audited value with mid-term review.
Type 1	<ul style="list-style-type: none">• Installations whose cost is $< 85\%$ of the reference cost.• Retributed according to the audited value.• Must be classified by the CNMC.
Type 2	<ul style="list-style-type: none">• Investments in digitalization and automation of grids, necessary for the energy transition.• Installations without an associated unitary cost.• Retributed according to the audited value.

Figure 18. Classification of the investments made by electric distribution utilities. [Cnmc19]

According to Figure 18, and based on the expected functionalities and benefits of the analysed solution, this should be categorized as a “Type 2” investment, so it must be retributed accordingly to its audited value.

To deploy the analysed solution, the electric distribution utility would initially incur in different costs:

- Initial implementation cost. Mainly the cost of implementing the management system, expected to be provided by a third party (proprietary software), and necessary software licenses.
- Cost of material: the hardware for the Edge Node, cables, adaptations of SSs etc.
- Cost of service: staff configuring the communications, displacements to the SSs for the installation, configuration of the Management System, etc.
- Cost of applications: the associated cost of developing applications for the functionalities of the Edge Node and their certification. This would be the only cost per functionality once the Edge Node is deployed.

Most of these costs are difficult to estimate reliably since they depend on the providers and on confidential information of the electric utility. Others, such as the cost of the hardware for the Edge Node, can be estimated based on general market information.

For this section, the average hardware of the Edge Node is assumed to be similar to the Axiomtek ICO300 Industrial IoT Gateway model⁸. According to Westward Sales, the basic cost of this model is around 370€, so the final cost (with additional features and considering that it must be previously certified, which is a one-time cost of around 5000€) is assumed to be ~450€ per unit. The CNMC establishes that the equipment related with “Smart Grids” has a regulatory life of 12 years.

8.1 Savings related to the new approach for new functionalities implementation

The decoupling of software from hardware analyzed in this project would change the way the electric utility deploys new functionalities. Traditionally, when i-DE wants to implement a new functionality that requires a new equipment (because it cannot be included in any of the devices already installed and it cannot be implemented in the central system), three providers are asked to develop, independently, the hardware+software ad hoc solution, that would be like a “black box” for the utility. This way, the electric utility can choose the best option while keeping the others as alternatives for the future, in case that the chosen one has any problem, or the provider cannot longer maintain it. Therefore, the investment needed for every development is multiplied by three and the process is long, requiring around 2-3 years to see the new functionality deployed on field (1.5 years if it is very urgent). This deployment is also costly since for every secondary substation the equipment must be acquired and installed physically.

In the new approach using edge computing, with the Edge Node deployed, the development process of new functionalities is strongly simplified. The hardware would no longer be part of the development and all the efforts could be focused on the software.

⁸<https://www.axiomtek.com/Default.aspx?MenuId=Products&FunctionId=ProductView&ItemId=1151&upcat=134#>

Although it depends on the complexity of the functionality, software typically takes less time to be developed (Agile methodology), overall if it does not include a GUI, so in around 3-4 months a new functionality could be deployed on field (after being certified). With these short development times, and its reduced cost in comparison with the traditional approach (since more competitiveness for software developments is expected), it is not worthy asking multiple providers to develop the same functionality. Once developed, there is no additional cost during deployment since the software container can be deployed remotely on the Edge Nodes.

Table 15 shows the different **Time To Market** (TTM) of these two approaches for a new functionality. The traditional approach, apart from being more expensive, supposes 2.5 years less of benefits (operational and economic) that a new functionality could provide. It also shows that the “typical” costs of a deployment would not appear in the new approach (with the Edge Node already on field).

Table 15. Summary of the differences between the two approaches for the development of a new functionality.

	Traditional Approach (Hardware+software)	New Approach (Edge Node) (Only software)
Average development time	3 years	4 months
No. Providers needed	3	1
Competitiveness for developments	Low	Very high
Cost of material	✓	✗
Cost of service	✓	✗
Cost of application	✓	✓

To compare and estimate the equivalent cost for the deployment of a new functionality in the two approaches, the following assumptions are made:

- A dedicated device is needed, in the traditional approach, for every two functionalities listed in this work (7.3 Advantages, Challenges and Functionalities, 15 functionalities \approx 8 devices).
- Average price per device similar to the Edge Node device (\sim 450€).
- Deployment in 25% of i-DE’s SSs (\sim 24000 secondary substations).
- A single Edge Node device would be able to support these 15 functionalities at the same time in a SS.
- Development cost is not considered.

Under these assumptions, by dividing the cost of hardware deployment between the number of functionalities that this deployment would support, the equivalent cost of deployment per functionality is estimated in order to give an idea of the economic

magnitude (Figure 19). The result shows that with the edge computing solution analyzed in this work, the equivalent cost of just deploying one new functionality is reduced in an 87.5%. This difference becomes greater as the number of functionalities that can be run in the Edge Node increases.

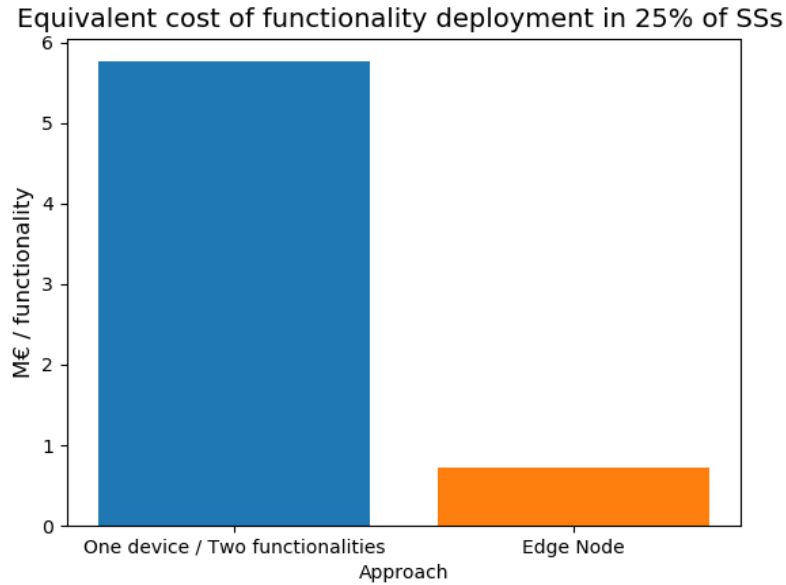


Figure 19. Difference between the equivalent cost of deploying a single functionality in 25% of i-DE’s SSs for the two approaches: 5.76 M€ for one device / two functionalities approach and 0.72 M€ for the Edge Node approach (edge computing)

The deployment of additional devices for new functionalities (traditional approach) proves to be inefficient sometimes. For many new functionalities, the cost-benefit analysis may result negative, which makes them an unjustifiable investment before the regulatory body. Furthermore, the inclusion of a new device in a SS is many times unviable due to space restrictions. For this reason, the electric utility usually tries to contact the manufacturers of the already-deployed devices to evaluate if it is possible the inclusion of the new functionality in the existing software as an update. Usually manufacturers adjust the hardware to the initial functionality, so the hardware might need to be modified or errors in the initial functionality might appear as a consequence of the update.

8.2 Savings related to device substitution in a SS

If edge computing constitutes a new approach for the deployment of new functionalities, it happens the same with the functionalities provided by currently deployed devices (e.g. DCs, RTUs, etc.). If this new way becomes successful, the path to follow would be the substitution of other devices in a SS by their equivalent software container running in the Edge Node. Therefore, the savings would be of the same type as those discussed in 8.1.

8.3 Savings in OPEX and benefits related to predictive maintenance applications

The implementation of the Edge Node in a SS would also suppose a reduction in OPEX: since it would substitute some electronic devices, the maintenance costs of these would be saved by the utility. Besides, the customization in the deployment of functionalities depending on the SS would allow the optimization of the control and monitoring of each SS, what would also lead to reductions in OPEX.

One of the functionalities mentioned in this work, that could be implemented in the Edge Node in the future, was predictive maintenance applications. The Edge Node could process semi real-time data from different sensors (e.g. sensors monitoring parameters of the transformer, the feeder, other equipment, etc.) to calculate the Remaining Useful Life (RUL) so that the state of the assets is perfectly known, and their maintenance can be planned in advance by sending a report to the central system. According to a report by PWC and Mainnovation [HKDM18], by applying predictive maintenance the asset could increase its lifetime by 20% and achieve an overall maintenance cost reduction of 12% (OPEX).

The electric distribution industry is considered to be a capital-intensive industry. Therefore, the electric utility must guarantee the lifetime of its assets until matching their regulatory life, at least (to recover the initial investment). In Spain, if the utility manages to increase the lifetime of the asset above its regulatory life, there is an increase in the regulated retribution for operation and maintenance (O&M), which receives the name of “Retribution due to Lifetime Extension” (RLE)⁹ [Cnmc19].

For example, a transformer has a regulatory life of 40 years according to the CNMC. If its lifetime is extended by 20%, that would suppose 8 years of additional retribution for the utility in concept of RLE, as Table 16 shows.

Table 16. RLE factor for each extended lifetime year. Calculated according to the formulas provided by the CNMC in [Cnmc19]

Additional Year	1	2	3	4	5	6	7	8
Additional retribution in O&M (%) (RLE)	30%	30%	30%	30%	30%	31%	32%	33%

⁹ In Spanish, this concept is REVU: “Retribución por Extensión de Vida Útil”.

8.4 Savings in Central System computing capabilities

Processing data at the edge of the LV network would alleviate the computing requirements in the Central System. As an example, the savings of running a connectivity algorithm (developed by Ariadna Grid) are estimated in this section. To run the algorithm, the S91 report by the LV advanced supervisor must be provided per line and day, adding 270 kB with respect to when the algorithm is not going to be executed, according to i-DE and Ariadna Grid. i-DE expects to implement LV advanced supervision for around 550,000 lines. Considering the simultaneous execution of the connectivity algorithm in Edge Nodes for this number of lines, the size of the data avoided to be processed in the central system would be 148.5 GB/day.

To estimate the economic value of these savings the prices of using the Dataflow tool of Google Cloud are used (Table 17). Since i-DE uses its own servers, the OPEX are difficult to estimate for these, so they are compared to the costs of renting a third-party infrastructure.

Table 17. Cost in € for a Google Cloud server in Belgium (europe-west1). [Data00]

Type of worker	vCPU / h	Memory (per GB and hour)	SSD Storage (per GB and hour)	Processed data (per GB)
Batch	0.05015	0.0035462	0.0002533	0.00935

The cost of processing 148.5 GB of data per day during a year using Google Cloud would be ~507€. If this daily amount data is stored during, at least, one month (30 days, a total of 4,455 GB), the cost of storing it in SSD would be ~9885€ per year.

$$Processing\ cost = 148.5 \frac{GB}{day} \cdot 365\ days \cdot \frac{0.00935\text{€}}{GB} = 506.79\text{€}$$

$$SSD\ cost = 148.5 \frac{GB}{day} \cdot 365\ days \cdot 24 \frac{h}{day} \cdot 30\ days \cdot 0.0002533 \frac{\text{€}}{GB \cdot h} = 9,885.24\text{€}$$

According to Google Cloud [Data00], the Batch worker consists on one vCPU, 3.75 GB of memory and 250 GB of storage. To store 30 days of data (4,455 GB), 18 Batch workers would be needed. Assuming that these workers would be working at half of their capacity, the cost of these vCPUs would be ~4,000€ and the memory usage would be ~1,000€.

$$vCPU\ cost = 18 \cdot 50\% \cdot 365\ days \cdot 24\ hours \cdot \frac{0.05015\text{€}}{vCPU} = 3,953.83\text{€}$$

$$Memory\ cost = 18 \cdot 50\% \cdot 3.75GB \cdot 365\ days \cdot 24\ hours \cdot \frac{0.0035462\text{€}}{GB \cdot hour} = 1,048.43\text{€}$$

Therefore, the economic savings of executing the connectivity algorithm in the Edge Nodes instead of the cloud (Central System) would account for ~15,000€ per year only

in computing. In addition to this, the corresponding reduction in data transmission (i.e. communications) has been valued by i-DE to be up to ~280,000€ per year (source: i-DE). The previous savings are only in terms of operation. In terms CAPEX, i-DE uses its own centralized server infrastructure, so the savings in this aspect should also be considered internally by the utility.

8.5 Savings related to energy losses

According to the CNMC [FGHR16] , in 2016 the sum of both technical and non-technical losses in the distribution system (i.e. transmission not considered) was estimated to be 20000 GWh which, at an average price of ~48 €/MWh (average price in 2019 in Spain, according to the market operator OMIE), supposes an overrun for the system of around 960 M€, which is distributed to be paid by every consumer. From these losses, it is believed that between 150-300 M€ are non-technical losses (e.g. illegal connections, smart meter manipulation, etc.). For this work, 200 M€ will be considered.

i-DE supplies energy to around 11 M customers (metering points). If it is assumed that the distribution of the losses could be proportional to the number of clients¹⁰, i-DE's grid would mean around 38% of the losses in the distribution level of Spain (Figure 20), valued in ~364.62 M€. From this losses, 64% take place in the LV grid (≤ 1 kV) [FGHR16], which is something to be considered since the Edge Node would be installed in SSs, receiving data mainly from the LV grid.

Percentage of clients supplied by each distribution company in Spain

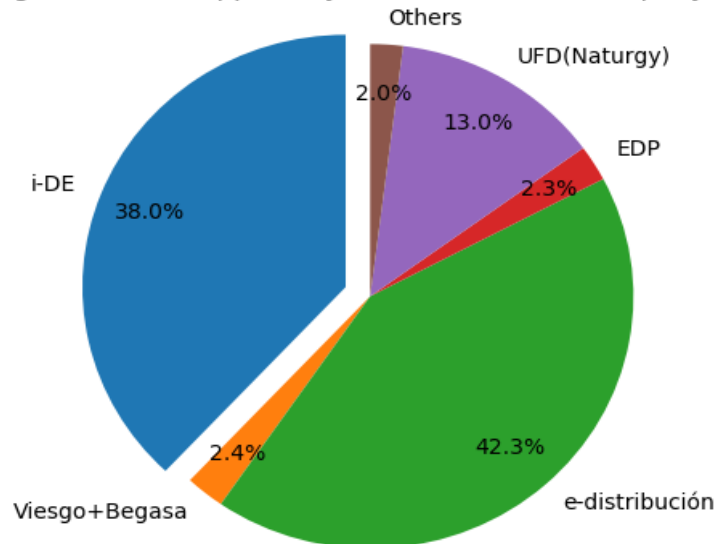


Figure 20. Approximate percentage of consumers per electric distribution company in Spain. Based on the number of meters provided by the CNMC in [PVBL19] and by the CIDE [Cide00]

¹⁰ This also implies the assumption that the clients are similarly distributed, the quality of the grids is similar for every utility, similar % of rural/urban clients, etc. No official data about the electric losses per company is public, so the results obtained in this section are approximated.

Table 18 shows an estimation of the annual value of the technical and non-technical losses for i-DE based on the assumptions and data already mentioned. These values will be used to estimate the potential savings for the system, derived from the deployment of the solution based on edge computing discussed in this project and some of its possible functionalities. It is important to remark that these savings are for the system, not for the utility. The economic impact of these savings for the utility depends on how the regulatory body retributes the reduction of losses (incentives), which is based on the self-improvement of the utility respect to previous years and limited to up to 2% of the annual retribution of the utility [Cnmc19]

Table 18. Estimated annual value of technical and non-technical losses for i-DE

Estimated i-DE Non-technical losses (M€)	75.96
Estimated i-DE Technical losses (M€)	288.66
Total (M€)	364.62

8.5.1 Devices consumption

The energy consumption of SS's devices is considered as non-technical losses. The grouping of functionalities in a single device (Edge Node) has its impact on energy consumption (less devices in the SS). Assuming an average power consumption of 10W per device (an Intel Atom processor consumes 8W), and considering that it is expected to be working 8760 hours per year, the amount of energy consumed in a year accounts for 87.6 kWh, which has an estimated yearly cost of 4.2€ (average price of 48€/MWh in 2019 in Spain). This would mean a saving for the system of ~0.1 M€ in energy per device and year if it is installed in, at least, 25% of i-DE's SSs.

8.5.2 Fraud detection

The circular published by the CNMC in December 2019 [Cnmc19] considers that the fraud detection incentive that was specified in the regulation has not been traduced into a real improvement in this area by the distribution utilities. For that reason, the CNMC proposes to delete this type of incentive and include it in the proposed losses reduction incentive.

Specialized fraud detection algorithms deployed in the Edge Node can help the utility to better (faster) detect anomalies and frauds in the electric consumption. Although the possible change in the regulation makes it even more difficult to estimate the value of the economic savings/retributions that this improvement could provide, they can be discussed in a qualitative way.

According to Smartgrids Info [Smar19], an electric distribution company similar to i-DE in the number of clients (e-Distribución, see Figure 20) detected in 2018 around 65000 fraud cases with an efficiency of 40% (4 out of 10 inspections concluded in fraud

detection). This efficiency could be improved by knowing better the LV grid (e.g. phase connectivity, line impedance, line balance, etc. possible functionalities deployed in the Edge Node) and by implementing fraud detection algorithms (whether in the Edge Node or in the central system using the results from the node). There are three possible scenarios:

- The number of fraud cases detected remains approximately the same but requiring less inspections (less “false positives”), which would let the utility to reduce the expenditure on inspections.
- Fraud cases that could remain undetected previously are now detected. After the deletion of the incentive, the economic retribution of this improvement will be extremely difficult to estimate for the utility, but it will keep its value for the system.
- A combination of the two previous. This would be, obviously, the most beneficial scenario.

In the odd case that no improvement is achieved in comparison to current methods, the wide range of possible functionalities for the Edge Node are believed to be enough to justify its future deployment.

8.5.3 Phase balancing

One of the benefits that can result from functionalities of the Edge Node, such as the phase connectivity algorithm (i.e. which phase each meter is connected to) or even demand response schemes (e.g. reduction of loads connected to a specific phase due to a strong phase imbalance), is a more balanced grid. According to data sent by i-DE to the CNMC, an optimal phase balancing would suppose a 10% reduction of (technical) losses [FGHR16].

Considering this, to calculate the expected savings of phase balancing (by i-DE) for the system, four scenarios are studied, assuming a linear relationship and a 5% reduction in the average price of energy (€/MWh) per year due to the penetration of renewables (decreasing economic value of losses):

1. 25% of achievement (2.5% reduction of technical losses).
2. 65% of achievement (6.5% reduction of technical losses).
3. 100% of achievement (10% reduction of technical losses).
4. Progressive achievement (+2.5% reduction each year)

Figure 21 plots the savings for the first three scenarios for each year and the price of energy used for each (second axis). Table 19 gathers the data plotted and the calculated present value for each scenario.

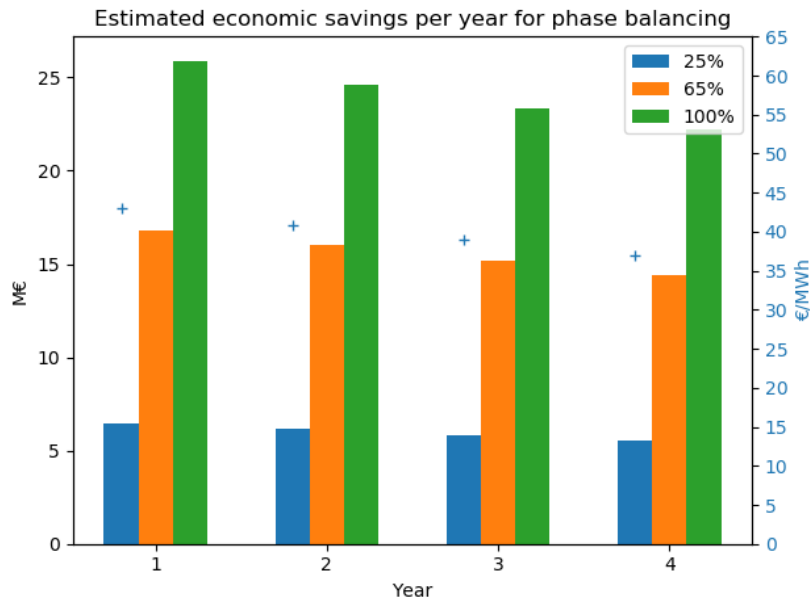


Figure 21. Estimated economic savings per year for phase balancing in i-DE considering different percentages of achievement. Calculated using the estimation of 20000 GWh of losses in 2016.

Table 19. Estimated economic savings for the system under different achievement scenarios for phase balancing in i-DE. Present Value of each scenario assuming $r=2.58\%$ (WACC of Iberdrola on 18/06/2020)

	Achievement Scenarios for Phase Balancing			
	100%	65%	25%	Progressive
Year 1	25,888,339 €	16,827,420 €	6,472,085 €	25%
Year 2	24,593,922 €	15,986,049 €	6,148,481 €	50%
Year 3	23,364,226 €	15,186,747 €	5,841,056 €	75%
Year 4	22,196,015 €	14,427,410 €	5,549,004 €	100%
PV ($r=2.58\%$)	90,300,694 €	58,695,451 €	22,575,173 €	54,275,283 €

The worst scenario studied (only 25% of achievement, 5%-annual reduction in the price of energy) would mean ~22.5 M€ of total savings (present value) in four years for the entire system. The progressive scenario, which is the most natural, would mean ~54 M€ of savings and it would also be the scenario where the utility can benefit more from incentives (it self-improves). The possible incentives that the utility could receive are not calculated in this section due to the lack of available information and complexity of the method, so it is limited to the “social” benefit.

8.6 Summary of economic impact

Table 20. Summary of the economic impact of applying edge computing at SSs

Investment classification by CNMC	Type 2: Retribution based on audited value
Regulatory life	12 years
Edge Node cost (per device)	450 €
Cost of the analysed technologies for the architecture: <ul style="list-style-type: none"> • Containerization • InfluxDB database • MQTT protocol • AMQP protocol • Linux OS 	0€
Cost of functionality development in comparison to traditional approach (hardware+software)(Table 15)	3x less (minimum)
Equivalent cost reduction in new functionality deployment	87.50%
Retribution increment per asset and year due to RLE (Predictive Maintenance applications)	≥ 30%
Cost reduction in maintenance (Predictive Maintenance applications)	Up to 12%
Decentralisation of Ariadna's connectivity algorithm	
Computing and storage savings per year	15,000 €
Communications savings per year	280,000 €
Energy losses	
Savings per device substituted/avoided and year (considering 25% of i-DE's SSs)	100,000 €
Fraud detection	Faster detection
	Less "false positives" => less inspections
	New frauds detected
Savings per year for phase balancing (25% scenario)	≥ 5,500,000 €

9. Tests

9.1 Test #1: Real-time balance in local test environment

This test consists on running a Docker container (developed in python) that calculates, in real time, the active power balance between a LV supervisor (whose ID is known) in a SS and the smart meters connected to the same feeder, so that the result is the sum of the power losses in the LV feeder (no distributed generation is considered).

This test is carried out in the local test environment (6.8 Local Test environment), (without any access to devices that could provide real-time data). To simulate the entrance of data in real time, a random data generator is developed using the Node-RED container (6.6 Node-RED) of this environment. This generator, shown in Figure 22, generates data in four different JSON documents (one per device: one LV supervisor and three smart meters) that follow the same structure (Table 21). Then, it waits until every JSON document is generated, creates a single JSON that includes all the data and finally publishes this JSON on an MQTT topic (which, in this case, is “measurements”) to be used by the balance application. Additionally, this generator also stores each individual JSON in the local InfluxDB database and publishes them on topics associated to each device.

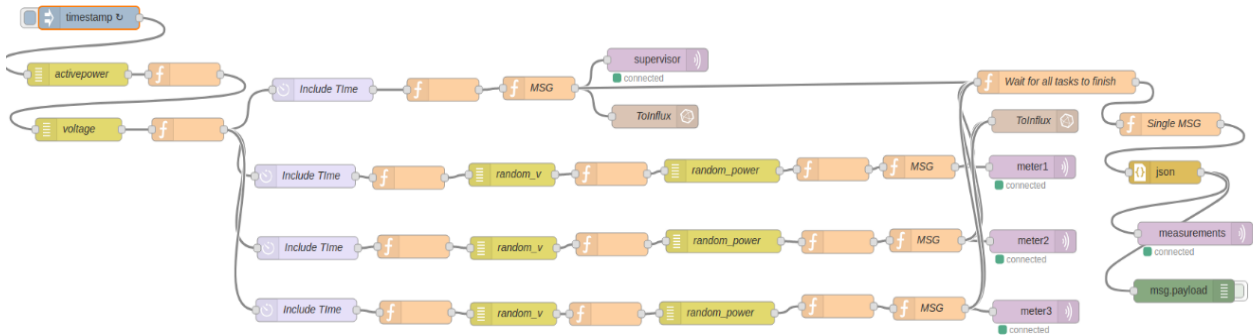


Figure 22. Screenshot of the random data generator developed in Node-RED. It generates data pretending to be a LV supervisor and three smart meters

Examples of the JSON documents generated by this random data generator are shown in Table 21. The structure of these JSONs is just an example so that the balance can be calculated, since no “official” JSON structure is defined yet for this type of data (i-DE and Futured are currently working on this definition, as mentioned in 7.4.3). For this test, data is generated every 10 seconds. This rate was chosen just to check performance in a fast way, but it can be easily changed by modifying the “timestamp” block in Figure 22.

Table 21. Examples of the JSON documents generated by the random data generator

Example of individual JSON document generated per device	Example of JSON document used by the balance application
<pre>{ "meter_id": 1, "date": "2020-07-06T09:15:46.614Z", "activepower": 8377.501330215542, "type": "triphasic", "voltage": 235.5713914634916 }</pre>	<pre>{ "supervisor": { "meter_id": 1, "date": "2020-07-06T09:09:25.069Z", "activepower": 9003.348300075664, "type": "triphasic", "voltage": 236.4842570622542 }, "meter1": { "meter_id": 2, "date": "2020-07-06T09:09:25.070Z", "activepower": 2640.9651014563265, "type": "monophasic", "voltage": 229.67703850550868 }, "meter2": { "meter_id": 3, "date": "2020-07-06T09:09:25.071Z", "activepower": 2745.90033160985, "type": "monophasic", "voltage": 222.28580672618412 }, "meter3": { "meter_id": 4, "date": "2020-07-06T09:09:25.072Z", "activepower": 2998.384320984686, "type": "monophasic", "voltage": 220.839067261682 } }</pre>

Although the data is generated randomly, the generator is designed to be coherent with reality in different aspects:

- The active power measured by the supervisor varies between 7 and 10.5 kW (approximately 3.5 kW per meter). The power measured by each meter is the result of applying a random factor to the supervisor’s power. The range of values of this factor for each meter is shown in Table 22.
- The voltage measured by the supervisor varies between 235 and 238 V. As with the active power, the voltage measured by each meter depends on a random factor applied to the supervisor’s power that varies in a defined range (Table 22).

Table 22. Range of values for the active power factor and voltage factor for each meter.

	Active Power Factor range	Voltage Factor range
Meter 1	[0.27, 0.30]	[0.96, 0.98]
Meter 2	[0.30, 0.35]	[0.94, 0.96]
Meter 3	[0.30, 0.35]	[0.91, 0.94]

The script to calculate the balance is developed in python and containerized afterwards (Code 4). The pseudocode of the application is shown in Code 3. At the end of the code, it waits 9 seconds to start again to receive messages through MQTT (the rate of generation was set to one message every 10 seconds).

Code 3. Pseudocode of the real-time balance script developed.

```

CONNECT to localhost MQTT broker container, port 1883
IF CONNECT is successful:
    PRINT "Successfully connected!"
    GO TO #1
ELSE:
    PRINT "Bad connection Returned code=" and show error code
#1
WHILE TRUE:
    SUBSCRIBE to "measurements" MQTT topic
    RECEIVE JSON Document
    EXTRACT "activepower" values
    EXTRACT "meter_id" values
    MATCH "meter_id" WITH "activepower" in a "power" DICTIONARY
    SET Balance TO 0
    FOR "meter_id" IN "power":
        IF "meter_id" == supervisor's ID:
            Balance = (Balance + "activepower")
        ELSE:
            Balance = (Balance - "activepower")
    CREATE JSON message WHERE:
        "balance_w" = Balance
        "percentage" = Balance*100/(supervisor's "activepower")
    PUBLISH JSON message ON "feeder/balance" MQTT topic
    WAIT 9 seconds

```

Once the result of the balance application is published on the "feeder/balance" MQTT topic, it is stored in the same InfluxDB database as the JSONs for each device, but with a different tag ("balance" in this case).

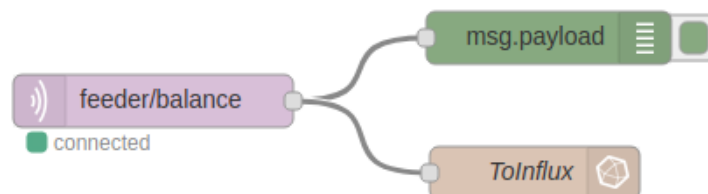


Figure 23. Screenshot of the Node-RED flow to store the JSON published on "feeder/balance" MQTT topic, in the local InfluxDB database

Finally, the real-time data and the resulting balance can be visualized in a Grafana dashboard (

Figure 24) since everything is being stored in the local InfluxDB database. Specifically,

Figure 24 shows:

1. The active power measured by the supervisor and the result of the balance (active power losses) during the last five minutes in the upper left graph.
2. The voltage of every device during the last five minutes in the upper right graph.
3. The average percentage of losses in the last five minutes (9.5%).
4. The average voltage (monophasic) of each device in the last five minutes.

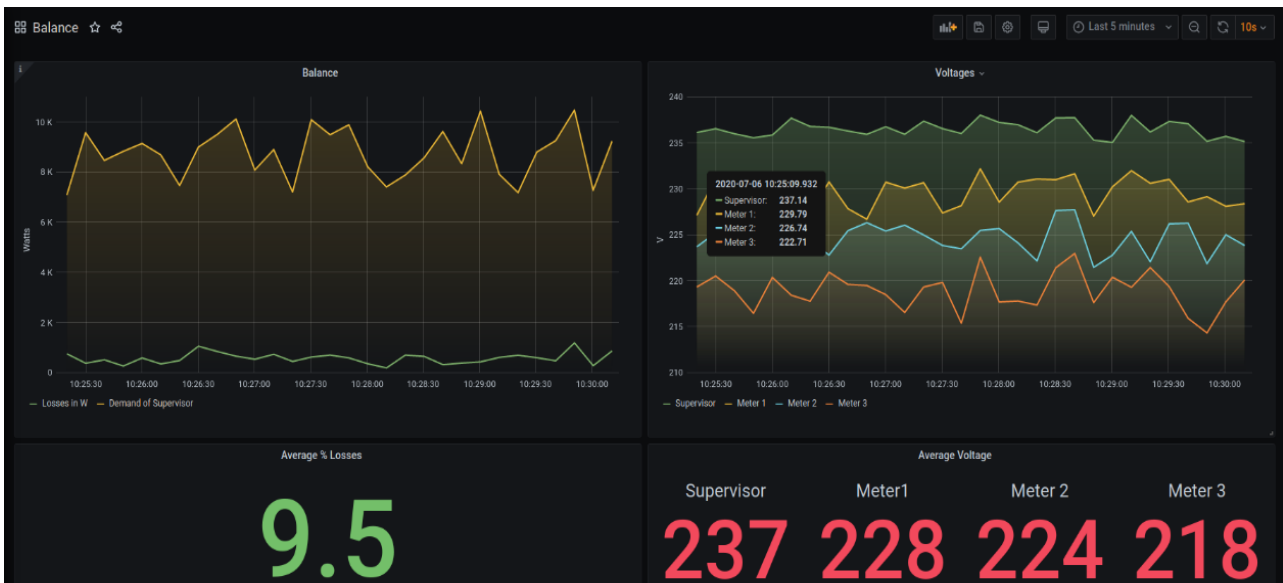


Figure 24. Grafana dashboard to visualize, in real time, the data from the devices and the results of the balance application (9.5% of average losses and the absolute losses)

The outcome of this test cannot be discussed in terms of how this feeder performs (as it is random data), but it shows the possibilities that the solution could provide. The real time balance application can be easily adapted to a real environment, since it is only necessary the supervisor's ID, the specific structure of the JSON document that contains the input data and the time between measures, as Test #2 will show. It is independent of the number of smart meters connected, as it can be deduced from Code 3.

This test also shows how different containers (e.g. Node-RED, InfluxDB, the balance application...) can work together without any difficulty by using the inner MQTT communications broker. If, for any reason, the balance application stops working or raises an error, the rest of containers would not be affected as it only interacts with them through the inner MQTT broker: the flow of Figure 23 would wait until receiving a message and the data generated would still be published on the MQTT topics and stored in the local InfluxDB database.

Finally, the possibility of deploying data visualization programs such as Grafana (the one used in Figure 24) or Chronograf as Docker containers, can be very useful for the utility's staff.

In terms of computing performance, Figure 25 shows the Docker stats for the containers actively used in this test. The InfluxDB container (database) takes up the most resources (CPU% and MEM%), which is something expected given that 5 insertions of data are done every 10 seconds and that the Grafana dashboard does 11 SQL-like queries every 10 seconds in order to represent the data (Figure 24). The MQTT broker container keeps a consumption similar to the one obtained during the AMQP-MQTT comparison test (Figure 14, subsection 7.5.1 Inner Communications), keeping an outstanding performance. It is remarkable the low requirements of the application developed for this test, the balance.app container. Although the code is not very complex, since it only consists on data processing, balance.app contains its own python environment (light version) and an additional library to interact with the MQTT broker, so the stats shown in Figure 25 for this container are acceptable. The stats for the Node-RED container in this test are not worth discussing since it is used for data generation, which would not be its use in a real deployment (data processor).

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
77b462a53e68	balance.app	0.02%	12.42MiB / 2.87GiB	0.42%	515kB / 293kB	16MB / 0B	2
ac19cb4cfdea	grafana	0.02%	14.37MiB / 2.87GiB	0.49%	5.41MB / 15.6MB	121MB / 938kB	14
ed85c328f79d	influxdb	0.08%	98.23MiB / 2.87GiB	3.34%	8.12MB / 5.78MB	168MB / 23.4MB	20
642db35d3a29	edge.mqtt	0.07%	736KiB / 2.87GiB	0.02%	1.25MB / 1.11MB	4.71MB / 12.3kB	1
a3bbb4857f5f	edge.nodered	0.00%	86.12MiB / 2.87GiB	2.93%	3.89MB / 8.34MB	163MB / 381kB	20

Figure 25. Screenshot of Docker stats for test #1

In conclusion, this test in the local test environment shows how a simple real time application would work in the edge computing solution analysed in this project, and how data can be easily stored and visualized in the node without the need of sending it immediately to the central system for that purpose.

9.2 Test #2: Real-time balance in remote test environment using S02-like reports

The objective of test #2 is to check the overall process from the development of the application to its deployment (using Onesait management system) in an Edge Node “on field” that already has other containers deployed and that already receives data that follows a determined structure. Therefore, the Remote Test environment is used for this test (described in 6.9 Remote Test environment).

The STG-DC simulator of this environment generates random S02 hourly reports, converts them into JSON (STG-DC original reports are in XML format, as explained in subsection 6.1) and publishes them on the inner MQTT broker. The number of simulated devices is 24 (1 supervision meter and 23 residential meters). Table 23 shows the main structure followed by the resulting JSON document. The STG-DC specification [CMFL15] titles the S02 report as “Daily incremental”, and it contains the values of six magnitudes (2 active and 4 reactive) per meter and hour. The relevant magnitudes (signalId) of this report to calculate the balance are:

- **AI.** Active Import. Measured in Wh for standard meters and kWh for supervision meters.
- **AE.** Active Export. Measured in the same way as AI.

Additionally, the ID of the device (deviceId) must be extracted to perform the operation.

Table 23. Example of part of the JSON document that contains the AI signal data for the supervision meter. This structure is repeated for every magnitude contained in a S02 report and for every meter.

```
{
  "profile": "STGDC",
  "deviceId": "DC1-METER0000",
  "signalId": "AI",
  "signal": "ACTIVE_ENERGY",
  "description": "Active Import",
  "value": "2.84",
  "number": 2.84,
  "timeStamp": 1596551391727,
  "timeStampInNanos": 1596551391727000000,
  "deviceType": "SUPERVISION",
  "temporality": 1
}
```

As, in this test, the S02-like report in JSON (Table 23) is used as data input, the application developed for test #1 (9.1 Test #1: Real-time balance in local test environment) is modified in order to extract the relevant information from these reports. The python code of this application can be seen in Code 6, ANNEX III. Code developed.

Once the code has been modified, the process to deploy it in the remote test environment is the one shown by Figure 17 and detailed below:

1. Application development.
2. Generation of the docker image. Using Dockerfile shown by Code 4 and `docker build` command. As it can be observed, Code 1 (subsection 7.4.1) is inserted to retrieve access to the `sh` and `bash` console (security reasons).

Code 4. Dockerfile used for Docker image generation. Used for test #1 and #2.

```
FROM python:3.8-slim-buster
# Keeps Python from generating .pyc files in the container
ENV PYTHONDONTWRITEBYTECODE 1
# Turns off buffering for easier container logging
ENV PYTHONUNBUFFERED 1
# Install pip requirements (paho-MQTT library)
ADD requirements.txt .
RUN python -m pip install -r requirements.txt
RUN /bin/rm -R /bin/sh
USER 9000
ADD balance.py /
CMD ["python", "./balance.py"]
```

3. Upload image to registry. Using `docker login` command and the user and password provided by Minsait. Due to security and confidentiality reasons, neither these credentials nor the address of this registry are shown in this work. The upload is done using `docker push` command.
4. Update `docker-compose.yml` in the git repository of the management system. Since this `.yml` file contains the configuration and details of other containers that are already deployed in the remote node, Code 5 only shows the piece of code to be included for the balance application developed. Except for the container name, the omission of any other field in Code 5 would, in the best case, disable the balance application and, in the worst case, raise an error during deployment. The application is configured to restart every time the Edge Node is rebooted.

Code 5. Configuration of the balance application developed in this work to be included in the `Docker-compose.yml` file in the private git repository.

```
version: '3'
services:
  testbalance:
    restart: always
    image: [url of the image in the registry]
    container_name: edge.testbalance
    depends_on:
      - mqtt
    links:
      - mqtt
```

```
networks:
  - edgenet
networks:
  edgenet:
    driver: bridge
```

5. Update containers deployed using the command available in the management system provided by Minsait. Although only one application is added, the system updates all the containers listed in the docker-compose.yml. Steps 6, 7 and 8 (Figure 17) are done automatically afterwards.

Once deployed, its correct functioning is checked using a node-RED flow which subscribes to the MQTT topic (“test/balance”) where the balance result is being published (Figure 26).

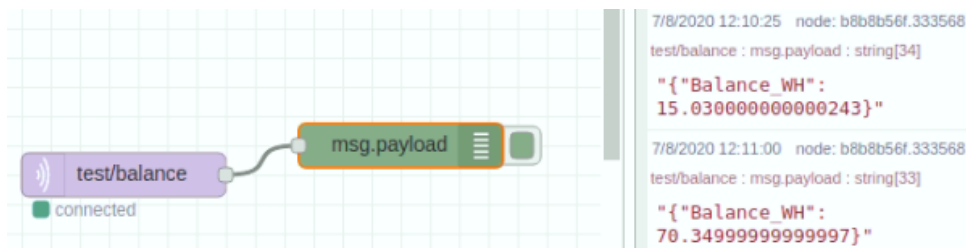


Figure 26. Node-RED Flow screenshot to check that the balance application deployed in the remote test environment works correctly

In terms of computing requirements, the modified balance application container presents the same values as in test #1 (Figure 25).

As opposite to test #1, where the results of the balance application were also inserted in the local InfluxDB database (see Figure 23), in this environment the ports of the database were not accessible, so the insertion of data from the “test/balance” topic into the database could not be done in a simple way through node-RED. It would be done by changing the configuration of the database in the Docker-compose.yml or by using Telegraf, whose configuration file is also complex. Due to the nature of this environment (own by Minsait), this configuration was not modified and, therefore, the resulting balance was not stored in the database during the test.

10. Conclusions

This project has analyzed the technologies (and their alternatives) involved in an edge-computing-based solution that follows the eWLCA [OSMC20]; the advantages, disadvantages, and functionalities that this solution could have in the LV distribution and its economic impact on the system and on the utility (i-DE), fulfilling the objectives initially set. Furthermore, the solution has been partially tested using two different test environments (local and remote) by developing an active energy balance application that works in real time. Considering this, the following conclusions can be extracted:

- ▶ The technologies proposed by the **eWLCA** are, in general, **appropriate** for its use in the LV distribution industry. A good level of intellectual property protection in Docker containers can be easily achieved by taking the measures listed in subsection 7.4.1. Additionally, the OS of the Edge Node should be one already homologated by the utility to ease future maintenance.
- ▶ The **modularity** of the solution (container-based) allows the deployment of functionalities as independent microservices based on the specific requirements and characteristics of the SS. That is to say, the deployment of microservices in SSs can be optimized to only deploy those which would really be useful in a determined SS or even during a determined period of time.
- ▶ Each of the **three communications environments** (inner, with management system and with other devices) of the solution have different requirements. This work has evaluated them and recommends the use of AMQP instead of MQTT (+TLS) for the communications between the Edge Node and the central system, increasing security in exchange of a higher load in communications. The use of MQTT for the other two environments is justified.
- ▶ The **decoupling of software and hardware** is a tendency among the edge platform vendors and one of the main advantages of the solution. However, finding the hardware that complies with all the requirements to be installed in a SS while providing acceptable computing capabilities, and at a good price, represents a major challenge that will require a deeper collaboration with manufacturers at these initial phases of implementation.
- ▶ The **economic impact** that the solution would have on the system and on the utility has a strong dependency on the final number of functionalities supported (hardware limitations) and on how the regulatory body retributes the improvements and investments of the utility. Nevertheless, the savings, benefits and advantages identified in this work for this solution are significant enough to justify the use of this edge computing solution in SSs.

- ▶ Three main **critical elements** are identified in the container **deployment process** (Test #2): the private container registry, the private git repository, and the management system. A successful attack over any of these elements would suppose a high operational and property risk, therefore they should be under the control of the utility's cybersecurity department.
- ▶ A **specification** that defines the structure of the messages to be used in the Edge Node (e.g. JSON version of the STG-DC reports) and the inner MQTT topics used is essential, since, with it, many processing applications could be developed and tested in a local environment in its final version, as test #1 showed (adaptation for test #2 would not have been necessary). This specification should also include the structure of the reports elaborated by the microservices deployed.
- ▶ The generation of a Docker image is found not to be an easy process. The employees of the utility would need some training and a guidebook to develop applications internally. The upload of the image to the registry is through the command line, which is not very user-friendly.
- ▶ The configuration of a container in the docker-compose.yml file can become extremely complex. Wrong docker-compose.yml configuration can result in errors during deployment, data losses, errors in communications, etc. This sensitivity to errors in configuration could be minimized in future phases of the PoC carried out at i-DE by, for example, the development of a user-friendly GUI to automatically generate this file or to check its syntax.
- ▶ Once deployed, containers cannot be easily reconfigured online. They have to be stopped and deployed again with the new configuration. The described (and tested) process in this work, at this initial phase of the PoC of the solution, deploys all the containers every time a new application is added or reconfigured, which is not advisable during operation (i.e. no application is running during some seconds in the node). These two aspects are especially relevant in the case of the database container, which should be operational full-time and that, currently, has to be reconfigured every time a new application (that uses a new MQTT topic) is deployed (if the database port was available, node-RED could be easily used for this purpose). This is expected to be solutioned at future phases of the PoC.
- ▶ **Visualization** (e.g. through Grafana or Chronograph) of the data stored in the node is relatively simple to achieve (it is based on SQL-like queries) and can be very useful for the utility. As test #1 showed, it would even be possible to visualize "real time" data as it is received and processed by the node, without the need of storing it in the central system, and increasing the visual monitoring of the LV grid at the SS level.
- ▶ The fact that the base technologies of this architecture are **freely available and open source**, allows the utility to partially avoid vendor-locking (not in the case of the

management system) and promotes the research and development of functionalities by academics and industry, since, as test #1 has shown, the Edge Node can be reproduced at low cost.

- ▶ This use of open source technologies, combined with the modularity of the solution, provides great **flexibility** for the utility to substitute in the future those technologies that become obsolete or when a better alternative technology arises.

All in all, the application of edge computing at the secondary substation level has the potential to be the new paradigm of how the LV grid is monitored and controlled, providing great benefits to the utility, to the system, and to the electric industry in general. The technologies used in the analyzed architecture are found to be the appropriate, although the deployment process tested should improve on user friendliness and on the capacity of making online configuration changes and new application deployments. However, these aspects, together with the hardware challenge, are close to be solved in future phases of the PoC developed at i-DE.

11. Recommendations for future works

Further to the analysis carried out in this work, some recommendations for future works are:

- Development of protocol adapter containers (e.g. STG-DC, Modbus, etc.).
- Classification of secondary substations according to their characteristics, requirements, and necessary functionalities to standardize microservices deployments according to this classification (i.e. customization of functionalities per type of SS).
- Study the future use of an IoT protocol (MQTT, AMQP, CoAP...) to communicate with smart meters while keeping PLC PRIME communication.
- Definition of the internal data models to be used in the solution taking WoT as reference.
- Taking advantage of this solution and the valuable information it will provide, study/develop the use of digital twins of SSs and the assets contained in them.
- Development of software microservices to provide functionalities in the Edge Node (similar to the balance application developed for this work).
- Study in depth (and develop) the use of edge computing for new energy services (i.e. DR schemes, energy storage, EV integration, etc.) at LV distribution level. Study technical and regulatory viability of these functionalities at the edge.

ANNEX I. Hardware Requirements

Table 24. List of requirements for the device to be installed in an i-DE's SS. Source: i-DE

N	TEST	NORM
Insulation		
1	Insulation resistance	UNE-EN 60255-27
2	Electric strength	UNE-EN 60255-27
3	Insulation with voltage impulses	UNE-EN 60255-27
Radioelectric disturbances		
4	Conducted emissions	UNE-EN 55022
5	Radiated emissions	UNE-EN 55022
Immunity		
6	Electrostatic discharge	UNE-EN 61000-4-2
7	Radiated high-frequency	UNE-EN 61000-4-3
8	Fast transients	UNE-EN 61000-4-4
9	Surges	UNE-EN 61000-4-5
10	Conducted RF	UNE-EN 61000-4-6
11	Magnetic field	UNE-EN 61000-4-8
12	Damped Magnetic field	UNE-EN 61000-4-10
13	Low Frequency Harmonics	UNE-EN 61000-4-13
14	Ring wave	UNE-EN 61000-4-18
Electrical		
15	AC Voltage DIPS/SAGS	UNE-EN 61000-4-11
16	Ground faults	EN 62052-11
17	Short time overcurrent influence	EN 62053-21
Mechanical		
20	Vibration	ETSI EN 300 019-2-2
21	Fall test (equipment)	ETSI EN 300 019-2-2
22	Fall test (equipment installed in the cabinet)	ETSI EN 300 019-2-2
23	IP protection	EN 20324
Climatic		
25	Damp heat	UNE-EN 60068-2-78
26	Dry heat	UNE-EN 60068-2-2
27	Cold	UNE-EN 60068-2-1
28	Temperature variation	UNE-EN 60068-2-14
29	Accelerated reliability test	UNE EN 62059-31-1 or similar

ANNEX II. United Nations SDG

The application of Edge computing and IoT to the LV electric distribution, as the analysed solution in this project, would be, not only a step forward to a smarter grid, but a step forward to some of the Sustainable Development Goals (SDG) for 2030 established by the United Nations in 2015 [Unit00].

Among these SDG, this project is directly related to two goals, and other impacts can be deduced from expected functionalities of the solution, although some of them depend on the degree of collaboration of third parties (e.g. regulatory bodies, electric consumers, etc.)

The first directly-related goal is SDG no.9 which is titled as "*Build resilient infrastructure, promote sustainable industrialization and foster innovation*" [Unit00]. This goal describes innovation and new infrastructures as ways of generating employment, competitiveness and income, and it considers as a key factor the efficient use of resources and energy. Among the eight targets of this goal, target 9.4¹¹ is considered to be addressed by the solution analysed in this project. The edge computing approach will mean a complete change of paradigm of how LV grids are managed and monitored. Functionalities such as those related to predictive maintenance would improve the use of the available resources and could increase the useful life of many grid components in 20%, as discussed in the Economic Impact. In the future, the Edge Node device itself, as it will include multiple functionalities that traditionally would be deployed in different hardware devices, will constitute an efficient use of the resources (less electronic hardware manufactured, less cable length used, etc), promoting indirectly target 12.2¹² under the goal no.12 of "*Responsible production and consumption*" [Unit00].

As mentioned in Advantages, Challenges and Functionalities, the solution can also be the driver for a higher penetration of EVs, DG and future DR schemes. Together with functionalities related to voltage control, power losses accountability and fraud detection, they would be in consonance with SDG no.7, "*Affordable and clean energy*", specifically target 7.3 ("*In 2030, double the global rate of improvement in energy efficiency*") [Unit00]. For example, if, as a result of different functionalities deployed in the Edge Node, i-DE achieves just a 2.5% reduction of power losses by means of a more phase-balanced LV grid (one of the scenarios studied in the Economic Impact), the amount of energy saved per year is estimated to be ~150 GWh (reducing, consequently, the bill of consumers) which is equivalent to 43200 tons of CO₂ per year (assuming an

¹¹(9.4) "*By 2030, upgrade infrastructure and retrofit industries to make them sustainable, with increased resource-use efficiency and greater adoption of clean and environmentally sound technologies and industrial processes, with all countries taking action in accordance with their respective capabilities*" [Unit00]

¹²(12.2) "*By 2030, achieve the sustainable management and efficient use of natural resources*" [Unit00]

average of 0.288 kgCO_{2e} / kWh in the electricity mix of Spain, according to the Carbon Footprint webpage [Carb00]).

In addition to this, the expected substitution of devices (both the existing ones in the SSs and the new ones for new functionalities) would decrease the energy consumption in a SS in 87.6 kWh per device and year (8. Economic Impact), which is equivalent to 25.23 kg of CO_{2e}.

Furthermore, by achieving a reduction in the losses (technical and non-technical) of the LV grid (depending on the magnitude of the reduction), the generation dispatch could change, lowering, in all likelihood, the average price of energy and increasing the interest for generation technologies with low operational costs (mainly renewables). This interest would be supported at the distribution level by the edge computing solution, since one expected functionality is the control of DG and energy storage. Flexibility functionalities (e.g. demand response, EV integration, etc.) in the node would suppose new energy services for consumers. Therefore, an indirect impact on targets 7.1¹³ and 7.2¹⁴ could be considered.




From a business strategy perspective, the SDGs aforementioned (7, 9 and 12) are also collected by Iberdrola Group's strategy¹⁵, which shows the alignment of the analysed solution with the objectives of Iberdrola Group.

¹³ (7.1) “By 2030, ensure universal access to affordable, reliable and modern energy services” [Unit00]

¹⁴ (7.2) “By 2030, increase substantially the share of renewable energy in the global energy mix” [Unit00]

¹⁵ <https://www.iberdrola.com/sustainability/committed-sustainable-development-goals>

Table 25. Summary of the SDGs that the solution analysed in this project could have an impact on. Images source: [Unit00]

SDG	Target	Relationship	Impact
	9.4	Direct	Higher resource efficiency, increase useful life of assets ($\leq 20\%$), change of paradigm in LV control and monitoring
	7.3	Direct	Just 2.5% reduction in energy losses would save 150 GWh (43200 tons of CO _{2e}) 25.23 kg CO _{2e} per device substituted and year
	7.1	Indirect	DR schemes, EV integration, DG control mechanisms
	7.2	Indirect	DG and energy storage control mechanisms in the Edge Node would ease renewables integration
	12.2	Indirect	Less electronic hardware manufactured, less cable length used, etc

ANNEX III. Code developed

Code 6. Python script to calculate the energy balance using the S02-like report received through MQTT in the remote test environment

```
"""
Version: v1.5.0
Last update: 06/08/2020
Author: Nestor Rodriguez Perez

Description: Script to calculate the balance of a feeder. Data is obtained
            through MQTT message in a JSON format (S02 Hourly Report)
"""
# Import MQTT client library, time module and JSON library
import paho.mqtt.client as mqtt
import paho.mqtt.subscribe as subscribe #Import subscription function
import time #To wait between measures
import json

#-----
#Callbacks functions for MQTT
def on_log(client, userdata, level, buf):
    print("log: "+buf)

def on_connect(client, userdata, flags, rc):
    if rc==0:
        print("Connection achieved")
    else:
        print("Wrong connection. Returned code=", rc)

def on_disconnect(client, userdata, flags, rc=0):
    print("DisConnected. Result code "+str(rc))

#-----
...

Function for JSON data extraction
Source: https://hackersandslackers.com/extract-data-from-complex-json-python/
...
def extract_values(obj, key):
    #Pull all values of specified key from nested JSON
    arr = []

    def extract(obj, arr, key):
        #Recursively search for values of key in JSON
        if isinstance(obj, dict):
            for k, v in obj.items():
                if isinstance(v, (dict, list)):
                    extract(v, arr, key)
                elif k == key:
                    arr.append(v)
        elif isinstance(obj, list):
            for item in obj:
                extract(item, arr, key)
    return arr

    results = extract(obj, arr, key)
    return results

#-----
...

Function to find the indexes in a list
```

Source: <https://stackoverflow.com/questions/5419204/index-of-duplicates-items-in-a-python-list>

```
'''
def search_index(seq,item):
    begin_at = -1
    ind = []
    while True:
        try:
            loc = seq.index(item,begin_at+1)
        except ValueError:
            break
        else:
            ind.append(loc)
            begin_at = loc
    return ind
#-----
def balance(imported, exported, supervisor_id=1):
    """Function to calculate the balance between the supervisor in a SS
    and the smart meters at that are fed by this SS (same supervisor).

    Args:
        imported (dictionary): Keys=meters_id ,values=active energy imported.
        exported (dictionary): Keys=meters_id ,values=active energy exported.
        supervisor_id (int, string): Supervisor's ID. Defaults to 1.

    Returns:
        [float]: The resulting active energy balance
    """
    balance=0
    for key in imported:
        if key == supervisor_id:
            balance = (balance + imported[key]*1000)
        else:
            balance = (balance - imported[key])

    for key in exported:
        if key == supervisor_id:
            balance = (balance - exported[key]*1000)
        else:
            balance = (balance + exported[key])

    return balance
#-----
#-----
#Connection to the inner MQTT broker
broker = "mqtt" #Inner MQTT broker IP address
client = mqtt.Client("processer_1") #Create new client of MQTT broker

client.on_connect = on_connect #Bind call back function on connection
client.on_disconnect = on_disconnect #Bind call back function on disconnection
client.on_log = on_log

print("Connecting to MQTT broker ", broker)
client.connect(host=broker, port=1883)
client.loop_start() #Start loop
#-----
#Starting the loop once the connection to MQTT is successful
condition = True
while condition:
```

```

#Subscription to the specific topic
msg = subscribe.simple(["topic/stgdc",0], hostname=broker, port=1883)
#Decodification of received message (JSON structure)
message_con = str(msg.payload.decode("utf-8"))
message_json = json.loads(message_con)

#-----
#Extract the signalId, deviceId from the JSON
signal_id = extract_values(message_json, "signalId")
meter_id = extract_values(message_json, "deviceId")
#Supervisor ID: DC1-METER0000
#Extract the values for ALL magnitudes
values = extract_values(message_json, "value")

#-----
#Initialize dictionary for energy imported
energy_import = {}
#Find the indices of the list with active energy imported
index_active_energy_import = search_index(signal_id, "AI")

#Initialize dictionary for energy exported
energy_export = {}
#Find the indices of the list with active energy exported
index_active_energy_export = search_index(signal_id, "AE")

for i in index_active_energy_import:
    #Dictionary with meter_id : active energy imported
    energy_import.update({meter_id[i]: float(values[i])})
for i in index_active_energy_export:
    #Dictionary with meter_id : active energy exported
    energy_export.update({meter_id[i]: float(values[i])})
#-----
#Calculus of the balance between supervisor and smart meters
balance_result = balance(energy_import, energy_export, supervisor_id="DC1-
METER0000")

#Result Message to be published in a MQTT topic
balance_message = {}
balance_message = {"Balance_WH":balance_result}

#Message published on the MQTT topic to be stored or used by other app
client.publish(topic="test/balance", payload=json.dumps(balance_message))
#Time between loops
time.sleep(5)

```

12. Bibliography

- [Abb16] ABB: Technical Guide: "Smart Grids 2. The "smart" secondary substation" (2016)
- [ASGM12] ALBERTO, MARTA ; SORIANO, RAÚL ; GÖTZ, JÜRGEN ; MOSSHAMMER, RALF ; ESPEJO, NICOLÁS ; LEMÉNAGER, FLORENT ; BACHILLER, RAÚL: "OpenNode: A smart secondary substation node and its integration in a distribution grid of the future". In: 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), 2012, S. 1277–1284
- [Bui15] BUI, THANH: "Analysis of Docker Security". In: arXiv:1501.02967 [cs] (2015). — arXiv: 1501.02967
- [Carb00] CARBON FOOTPRINT: "Carbon Footprint. Your solution for cutting carbon and caring for the climate". URL <https://www.carbonfootprint.com/>. - accessed on 2020-07-07
- [CDLR19] CAPROLU, MAURANTONIO ; DI PIETRO, ROBERTO ; LOMBARDI, FLAVIO ; RAPONI, SIMONE: "Edge Computing Perspectives: Architectures, Technologies, and Open Security Issues". In: 2019 IEEE International Conference on Edge Computing (EDGE). Milan, Italy : IEEE, 2019 — ISBN 978-1-72812-708-8, S. 116–123
- [CeHP16] CEJKA, STEPHAN ; HANZLIK, ALEXANDER ; PLANK, ANDREAS: "A framework for communication and provisioning in an intelligent secondary substation." In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), 2016, S. 1–5
- [CeME15] CEJKA, STEPHAN ; MOSSHAMMER, RALF ; EINFALT, ALFRED: "Java embedded storage for time series and meta data in Smart Grids". In: 2015 IEEE International Conference on Smart Grid Communications (SmartGridComm), 2015, S. 434–439
- [Chur18] CHURILO, CHRIS: "MongoDB vs InfluxDB | InfluxData Time Series Workloads." URL <https://www.influxdata.com/blog/influxdb-is-27x-faster-vs-mongodb-for-time-series-workloads/>. - accessed on 2020-06-10. — InfluxData
- [Cide00] CIDE: "CIDE". URL <http://www.cide.net/donde-distribuímos.php>. - accessed on 2020-06-17
- [CMFL15] CERERO, R. ; MARTÍNEZ, P. ; FERRER, V. ; LARUMB, I.: "STG-DC INTERFACE SPECIFICATION v3.4", Iberdrola Distribución Eléctrica, SA. (2015)
- [Cnmc19] CNMC: "Memoria Justificativa de la Circular de la CNMC por la que se establece la metodología para el cálculo de la retribución de la actividad de la distribución de energía eléctrica.", CNMC (2019)
- [Coun14] COUNET, JEROME: "Over 70% European consumers to have a smart meter for electricity by 2020" URL <https://ec.europa.eu/jrc/en/news/over-70-percent-european-consumers-have-smart-meter-electricity-2020>. - accessed on 2020-05-03. — EU Science Hub - European Commission
- [CWWL19] CHEN, SONGLIN ; WEN, HONG ; WU, JINSONG ; LEI, WENXIN ; HOU, WENJING ; LIU, WENJIE ; XU, AIDONG ; JIANG, YIXIN: "Internet of Things Based Smart Grids Supported by Intelligent Edge Computing". In: IEEE Access Bd. 7 (2019), S. 74089–74102

- [Data00] "Dataflow pricing" URL <https://cloud.google.com/dataflow/pricing>. - accessed on 2020-07-28. — Google Cloud
- [Db-e00] "DB-Engines Ranking" URL <https://db-engines.com/en/ranking/time+series+dbms>. - accessed on 2020-06-10. — DB-Engines
- [Dock20a] "Docker overview" URL <https://docs.docker.com/get-started/overview/>. - accessed on 2020-05-20. — Docker Documentation
- [Dock20b] DOCKER: "Overview of Docker Compose" URL <https://docs.docker.com/compose/>. - accessed on 2020-05-21. — Docker Documentation
- [Edge00a] EDGE ONESAIT PLATFORM: "Deployment and Installation - Onesait Platform Developer Portal" URL <https://onesaitplatform.atlassian.net/wiki/spaces/OP/pages/39813152/Deployment+and+Installation>. - accessed on 2020-05-25
- [Edge00b] EDGE ONESAIT PLATFORM: "Management System - Onesait Platform Developer Portal." URL <https://onesaitplatform.atlassian.net/wiki/spaces/OP/pages/164986881/Management+System>. - accessed on 2020-05-03
- [Eybe19] EYBERG, IAN: "Introduction To Unikernels." URL <https://nordicapis.com/introduction-to-unikernels/>. - accessed on 2020-06-03. — Nordic APIs
- [FCSF17] FASCHANG, MARIO ; CEJKA, STEPHAN ; STEFAN, MARK ; FRISCHENSCHLAGER, ALBIN ; EINFALT, ALFRED ; DIWOLD, KONRAD ; PRÖSTL ANDRÉN, FILIP ; STRASSER, THOMAS ; U. A.: "Provisioning, deployment, and operation of smart grid applications on substation level: Bringing future smart grid functionality to power distribution grids". In: *Computer Science - Research and Development* Bd. 32 (2017), Nr. 1–2, S. 117–130
- [FGHR16] FERNÁNDEZ PÉREZ, MARÍA ; GARCÍA MATILLA, EDUARDO ; DE LA HIGUERA GONZÁLEZ, CLOTILDE ; RODRÍGUEZ RODRÍGUEZ, DIEGO ; ZENARRUTZABEITIA BELDARRAÍN, IDOIA: "Informe sobre la Evaluación del Potencial de Eficiencia Energética de las Infraestructuras Eléctricas", CNMC (2016)
- [Garc16] GARCIA, BENJAMIN: "PRIME v1.4 is ready to improve Smart Grids." URL <https://www.teldat.com/blog/en/prime-plc-technology-prime-v14-smart-grid-cenelec-a-dbpsk/>. - accessed on 2020-06-08. — Teldat Blog - Connecting the World
- [GiLy12] GILBERT, SETH ; LYNCH, NANCY: "Perspectives on the CAP Theorem" In: *Computer* Bd. 45 (2012), Nr. 2, S. 30–36
- [GSAV18] GOETHALS, TOM ; SEBRECHTS, MERLIJN ; ATREY, ANKITA ; VOLCKAERT, BRUNO ; DE TURCK, FILIP: "Unikernels vs Containers: An In-Depth Benchmarking Study in the Context of Microservice Applications". In: *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*, 2018, S. 1–8
- [Hilt20] HILTON, STEVE: "MachNation rates 11 IIoT edge vendors in 2020 IoT Edge ScoreCard" URL <https://www.machnation.com/2020/02/07/machnation-rates-iiot-edge-vendors-2020-iiot-edge-scorecard/>. - accessed on 2020-07-09. — MachNation

- [HKDM18] HAARMAN, MARK ; DE KLERK, PIETER ; DECAIGNY, PETER ; MULDER, MICHEL ; VASSILIADIS, COSTAS ; SIJTSEMA, HEDWICH ; GALLO, IVAN: "Predictive Maintenance - Beyond the hype: PdM 4.0 delivers results": PWC and Mainnovation, 2018
- [HLWF18] HUANG, YUTAO ; LU, YUHE ; WANG, FENG ; FAN, XIAOYI ; LIU, JIANGCHUAN ; LEUNG, VICTOR C.M.: "An Edge Computing Framework for Real-Time Monitoring in Smart Grid". In: 2018 IEEE International Conference on Industrial Internet (ICII). Seattle, WA : IEEE, 2018 — ISBN 978-1-5386-7771-1, S. 99–108
- [Iber00] IBERDROLA: "Star Project". URL <https://www.i-de.es/smart-grids/deployment-projects-areas/star-project>. - accessed on 2020-05-03
- [JPAK12] JATANA, NISHTHA ; PURI, SAHIL ; AHUJA, MEHAK ; KATHURIA, ISHITA ; GOSAIN, DISHANT: "A Survey and Comparison of Relational and Non-Relational Database". In: International Journal of Engineering Research Bd. 1 (2012), Nr. 6, S. 5
- [JWHY18] JINMING, CHEN ; WEI, JIANG ; HAO, JIAO ; YAJUAN, GUO ; GUOJI, NIE ; WU, CHEN: "Application Prospect of Edge Computing in Smart Distribution". In: 2018 China International Conference on Electricity Distribution (CICED). Tianjin, China : IEEE, 2018 — ISBN 978-1-5386-6775-0, S. 1370–1375
- [KMLK20] KOVATSCH, MATTHIAS ; MATSUKURA, RYUICHI ; LAGALLY, MICHAEL ; KAWAGUCHI, TORU ; TOUMURA, KUNHIKO ; KAJIMOTO, KAZUO: "Web of Things (WoT) Architecture". URL <https://www.w3.org/TR/wot-architecture/>. - accessed on 2020-08-03
- [Kube20a] KUBERNETES: "What is Kubernetes?" URL <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. - accessed on 2020-05-25
- [Kube20b] KUBERNETES: "Pods." URL <https://kubernetes.io/docs/concepts/workloads/pods/pod/>. - accessed on 2020-05-26
- [Luci17] LUCIA, MICHAEL J DE: "A Survey on Security Isolation of Virtualization, Containers, and Unikernels" In: US Army Research Laboratory (2017), S. 18
- [Lum19] LUM, BRANDON: "Encrypting container images with containerd imgcrypt!" URL <https://medium.com/@lumjib/encrypting-container-images-with-containerd-imgcrypt-3c07f8e8e8d4>. - accessed on 2020-07-03. — Medium
- [Mcka19] MCKAY, DAVE: "What Is Reverse SSH Tunneling? (and How to Use It)". URL <https://www.howtogeek.com/428413/what-is-reverse-ssh-tunneling-and-how-to-use-it/>. - accessed on 2020-07-14. — How-To Geek
- [Micr19] MICROSOFT: "Linux Containers on Windows 10" URL <https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers>. - accessed on 2020-06-04
- [Mira20] MIRANTIS: "Mirantis Documentation: Define roles with authorized API operations" URL <https://docs.mirantis.com/docker-enterprise/v3.0/dockereee-products/ucp/authorization/define-roles.html>. - accessed on 2020-06-16
- [Mqtt00a] MQTT. URL <https://mqtt.org/>. - accessed on 2020-05-17
- [Mqtt00b] "mqtt/mqtt.github.io". URL <https://github.com/mqtt/mqtt.github.io>. - accessed on 2020-05-17. — GitHub

- [MRCD18] MARTIN, A. ; RAPONI, S. ; COMBE, T. ; DI PIETRO, R.: "Docker ecosystem – Vulnerability Analysis". In: *Computer Communications* Bd. 122 (2018), S. 30–43
- [NaAb19] NASAR, MOHAMMAD ; ABU KAUSAR, MOHAMMAD: "Suitability Of Influxdb Database For Iot Applications." In: *International Journal of Innovative Technology and Exploring Engineering* Bd. 8 (2019), Nr. 10, S. 1850–1857
- [Naik17] NAIK, NITIN: "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP". In: *2017 IEEE International Systems Engineering Symposium (ISSE)*. Vienna, Austria : IEEE, 2017 — ISBN 978-1-5386-3403-5, S. 1–7
- [Nebb19] NEBBIOLO TECHNOLOGIES INC.: "Overview of Nebbiolo fogSM Software", Nebbiolo Technologies Inc. (2019)
- [Node00] NODE-RED: "About Node-RED". URL <https://nodered.org/about/>. - accessed on 2020-05-23
- [Oasi12] OASIS STANDARD: "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0", Oasis Open (2012)
- [Oasi19] OASIS STANDARD: "OASIS MQTT Version 5.0", OASIS Open (2019)
- [OkOz16] OKAY, FEYZA YILDIRIM ; OZDEMIR, SUAT: "A fog computing based smart grid model". In: *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, 2016, S. 1–6
- [OSMC20] ORTEGA DE MUES, MARIANO ; SESEÑA, DANIEL ; MARTÍNEZ SPESSOT, CÉSAR ; CARRANZA, MARCOS ; LANG, JORGE: "Creating an effective and scalable IoT infrastructure by introducing Edge Workload Consolidation (eWLC)." URL <https://www.intel.com/content/www/us/en/develop/articles/edge-workload-consolidation-ewlc.html>. - accessed on 2020-06-24. — Intel
- [Over00] "Overview / dlms". URL <https://www.dlms.com/dlms-cosem/overview>. - accessed on 2020-05-12
- [Pato16] PATO, BALINT: "Jepsen and InfluxDB, Chapter II. Where is InfluxDB on the CAP scale?" URL http://www.refactorium.com/distributed_systems/InfluxDB-and-Jepsen-Chapter-II-Where-is-influxdb-on-the-cap-scale/. - accessed on 2020-06-22. — The Refactorium
- [Prim00a] PRIME Alliance | "Advanced Meter Reading & Smart Grid Standard." URL <https://www.prime-alliance.org/>. - accessed on 2020-05-12
- [Prim00b] PRIME ALLIANCE: "PRIME v1.4 White Paper", PRIME Alliance
- [PVBL19] PÉREZ, MARÍA FERNÁNDEZ ; VALDÉS DÍAZ, BENIGNO ; BACIGALUPO SAGGESE, MARIANO ; LORENZO ALMENDROS, BERNARDO ; ORMAETXEA GARAI, XABIER: "Acuerdo por el que se emite el informe sobre el cumplimiento del último hito del plan de sustitución de contadores", CNMC (2019)
- [Ref00a] CORDIS | EUROPEAN COMMISSION: "Open Architecture for Secondary Nodes of the Electricity SmartGrid: OpenNode Project". URL <https://cordis.europa.eu/project/id/248119>. - accessed on 2020-05-27
- [Ref00b] MONGODB: "MongoDB: The most popular database for modern apps". URL <https://www.mongodb.com>. - accessed on 2020-06-21. — MongoDB
- [Ref20] DOCKER: "Docker stats" URL <https://docs.docker.com/engine/reference/commandline/stats/>. - accessed on 2020-06-08. — Docker Documentation

- [Smar19] SMARTGRIDS INFO: "*Los algoritmos avanzados y el big data ayudaron a Endesa a detectar casi 65.000 fraudes eléctricos en 2018*". URL <https://www.smartgridsinfo.es/2019/02/01/algoritmos-avanzados-big-data-ayudaron-endesa-detectar-casi-65000-fraudes-electricos-2018>. - accessed on 2020-07-09. — Smartgrids Info
- [SoLi12] SONG, YI ; LI, JINGRU: "Analysis of the life cycle cost and intelligent investment benefit of smart substation". In: *IEEE PES Innovative Smart Grid Technologies*, 2012, S. 1–5
- [SSBS16] SENDIN, ALBERTO ; SANCHEZ-FORNIE, MIGUEL A. ; BERGANZA, IÑIGO ; SIMON, JAVIER ; URRUTIA, IKER: "*Telecommunication networks for the smart grid*", *Artech house power engineering series*. Norwood, Massachusetts : Artech House, 2016 — ISBN 978-1-63081-046-7
- [Stro19] STRONG, AARON: "*Containerization vs. Virtualization: What's the Difference?*" URL <https://www.burwood.com/blog-archive/containerization-vs-virtualization> - accessed on 2020-06-02. — Burwood Group
- [Suma13] SUMASTRE, MICHAEL GABRIEL: "*Virtualization 101: What is a Hypervisor?*" URL <https://www.pluralsight.com/blog/it-ops/what-is-hypervisor>. - accessed on 2020-05-20
- [Toka17] TOKAR, DIMA: "*Whitepaper: Five requirements of a leading IoT edge platform*". URL <https://www.machnation.com/2017/09/18/whitepaper-five-requirements-leading-iot-edge-platform/>. - accessed on 2020-07-10. — MachNation
- [Unit00] UNITED NATIONS: "*United Nations Sustainable Development – 17 Goals to Transform Our World*" URL <https://www.un.org/sustainabledevelopment/>. - accessed on 2020-07-07
- [WLYC18] WANG, PAN ; LIU, SHIDONG ; YE, FENG ; CHEN, XUEJIAO: "*A Fog-based Architecture and Programming Model for IoT Applications in the Smart Grid*". In: *arXiv:1804.01239 [cs]* (2018). — arXiv: 1804.01239