Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Uso de la tecnología *blockchain* en intercambios de energía en microrredes eléctricas

Autor: Miguel Gayo Abeleira

Tutor: Francisco Javier Rodríguez Sánchez

2019

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Uso de la tecnología *blockchain* en intercambios de energía en microrredes eléctricas

Autor: Miguel Gayo Abeleira

Tutor: Francisco Javier Rodríguez Sánchez

Tribunal:

Presidente: Santiago Cóbreces Álvarez

Vocal 1º: Jesús Florencio Sánchez Golmayo

Vocal 2º: Francisco Javier Rodríguez Sánchez

Calificación: .....

Fecha: .....

A mi familia

Resumen

Este documento presenta una plataforma basada en la tecnología *blockchain* para realizar intercambios de energía entre prosumidores dentro de una microrred eléctrica. La tecnología *blockchain* se emplea para garantizar la trazabilidad de la energía en todas sus fases y para todas las partes implicadas en los intercambios energéticos, desde el momento de la generación, hasta el momento de su consumo. Además, se incluye el proceso de subasta de esa energía, donde varios consumidores han realizado ofertas por ella y una de ellas ha resultado ganadora. Todos estos pasos quedan grabados en una base de datos siempre creciente, de la que cada usuario de la microrred tiene una copia sincronizada y protegida criptográficamente frente a manipulaciones. Todo el proceso descrito se basa en la tecnología de cadena de bloques o en inglés, como es más conocida, la *blockchain*. Este trabajo se enmarca dentro del proyecto PROMINT-CM, financiado por la Comunidad de Madrid y el Fondo Social Europeo (Ref: P2018/EMT4366).

Palabras clave: Blockchain, energía, intercambio, prosumidor, microrred.

Abstract

This document presents a platform based on blockchain technology to perform energy exchanges between prosumers within an electrical microgrid. The blockchain is used to guarantee the traceability of energy at all stages and for all parties involved in exchanges, from the generation to the consumption. In addition, the process of auctioning that energy has been included, where several consumers have made bids for it and one of them has won. All these steps are recorded in an ever-growing database, of which each user of the microgrid has a synchronized copy and protected cryptographically against manipulations which is the blockchain. This work is part of the PROMINT-CM project, funded by the Comunidad de Madrid and the European Social Fund (Ref: P2018/EMT4366).

Keywords: Blockchain, energy, exchange, prosumer, microgrid.

Índice general

Resumen **vii**

Abstract **ix**

Índice general **xi**

Índice de figuras **xv**

Índice de tablas **xvii**

Índice de listados de código fuente **xxi**

1 Introducción **1**

1.1 Tecnología *blockchain* 1

1.1.1 Entrelazamiento de los bloques 2

1.1.2 Árbol de Merkle 3

1.1.3 Marca de tiempo 4

1.1.4 Ventajas del *blockchain* 4

1.2 Estado de la red eléctrica en la actualidad 5

1.2.1 Problemas de la red eléctrica actual 6

1.2.2 Evolución de la red eléctrica 6

1.2.3 Usuarios prosumidores 8

1.2.4 Gestión de los usuarios prosumidores 9

1.3 Microrredes eléctricas 11

1.3.1 Intercambios de energía en una microrred 13

1.3.2 Microrredes distribuidas 13

1.4 Aplicación de la tecnología *blockchain* a los intercambios de energía 14

2 Arquitectura de la red *blockchain* **17**

2.1 *Bitcoin* y *Ethereum* 17

2.2 Arquitecturas *blockchain* 18

2.2.1 Nodos usuarios y validadores 18

2.2.2 Redes públicas y privadas . . . 18
2.2.3 Propósito general o específico . . . 19
2.3 Diferentes consensos de validación de transacciones . . . 19
2.3.1 Prueba de trabajo (PoW) . . . 21
2.3.2 Prueba de participación (PoS) . . . 22
2.3.3 Prueba de tiempo transcurrido (PoET) . . . 22
2.3.4 Prueba con tolerancia a faltas bizantinas (PBFT) . . . 22
2.3.5 Otros algoritmos de consenso . . . 23
2.4 Arquitectura *blockchain* para una plataforma de intercambios de energía en una microrred 23
2.4.1 *Ethereum* vs *Hyperledger* . . . 24
2.4.2 Decisión final . . . 25
2.5 *Hyperledger Fabric* . . . 26
2.5.1 Verificación de la identidad de los usuarios en *Hyperledger Fabric* . . . 26
2.5.1.1 Certificados Digitales . . . 27
2.5.1.2 Claves pública y privada . . . 27
2.5.1.3 Autoridad certificadora . . . 28
2.5.1.4 Lista de certificados rechazados . . . 28
2.5.2 Comprobación de la membresía en una red de *Hyperledger Fabric* . . . 29
2.5.3 Algoritmos de consenso en *Hyperledger Fabric* . . . 29
2.5.4 Smart Contracts . . . 30
2.5.5 Flujo de las transacciones . . . 31
2.5.6 *Hyperledger Composer* . . . 31

3 Estructura del sistema 35
3.1 Usuarios del sistema . . . 36
3.2 Funcionamiento de la plataforma de intercambios . . . 37
3.3 Cadena de bloques de datos personales . . . 38
3.3.1 Contratos inteligentes . . . 39
3.3.2 Permisos . . . 39
3.4 Cadena de bloques de mercado . . . 39
3.4.1 Contratos inteligentes . . . 40
3.4.2 Permisos . . . 41
3.5 Cadena de bloques de registro de parámetros eléctricos . . . 41
3.5.1 Contratos inteligentes . . . 42
3.5.2 Permisos . . . 43
3.6 Cadena de bloques de liquidación de los pagos . . . 43
3.6.1 Contratos inteligentes . . . 44
3.6.2 Permisos . . . 44

4 Interacción de los usuarios con la aplicación 47

4.1 Archivos de preparación e instalación 48

4.1.1 Instalación 48

4.1.2 Creación de usuarios prosumidores 48

4.1.3 Alta en el sistema por parte de los usuarios prosumidores 49

4.1.4 Desinstalar usuarios prosumidores 49

4.2 Clientes de interacción con la plataforma 49

4.2.1 Cliente de creación de usuarios prosumidores 50

4.2.2 Cliente del gestor del sistema 50

4.2.3 Cliente del usuario prosumidor 51

5 Pruebas de funcionamiento realizadas 53

5.1 Consideraciones previas 53

5.2 Prueba de funcionamiento de una ronda de mercado 54

5.2.1 Fase previa 56

5.2.2 Fase posterior 56

5.3 Prueba de funcionamiento de una ronda de mercado con prosumidores no aptos 57

5.4 Prueba de funcionamiento de un gran número de usuarios prosumidores en un único equipo durante un periodo de tiempo amplio 58

5.5 Prueba de funcionamiento añadiendo otro nodo 59

5.5.1 Análisis de un usuario 60

5.5.2 Análisis de diez usuarios 61

5.5.3 Análisis de cien usuarios 61

5.6 Prueba de funcionamiento con el gestor del sistema y prosumidores en equipos diferentes . 64

5.6.1 Análisis de un usuario 64

5.6.2 Análisis de diez usuarios 64

5.6.3 Análisis de cien usuarios 66

5.7 Prueba de funcionamiento de un gran número de usuarios prosumidores, durante varias rondas de mercado en dos equipos 68

5.7.1 Ancho de banda 68

5.7.2 Capacidad de computación 70

5.7.3 Capacidad de almacenamiento 70

6 Conclusiones y líneas futuras 73

6.1 Conclusiones 73

6.2 Líneas futuras 74

Bibliografía 77

A Listados de código **81**

A.1 Código de las cadenas de bloques 81

A.1.1 Cadena de bloques de datos personales 81

A.1.2 Cadena de bloques de mercado 86

A.1.3 Cadena de bloques para el registro de parámetros eléctricos 93

A.1.4 Cadena de bloques de liquidación de pagos 97

A.2 Archivos de instalación 103

A.3 Cliente en Node.js de creación de usuarios prosumidores 107

A.4 Cliente en Node.js del gestor del sistema 111

A.5 Cliente en Node.js de los prosumidores 119

B Configuración y monitorización de *Hyperledger Fabric* **121**

B.1 Descarga e instalación de *Hyperledger Composer* 121

B.2 Control del entorno de desarrollo 122

B.3 Modificación del número de nodos por defecto 122

B.4 Monitorización de recursos empleados por contenedores 128

C Pliego de condiciones **129**

C.1 Presupuesto de realización del TFM 129

C.2 Pliego de condiciones plataforma intercambios de energía 130

C.2.1 Pliego de condiciones generales 130

C.2.2 Costes prosumidor 130

C.2.3 Costes gestor del sistema 131

Índice de figuras

1.1 Plataforma de transacciones centralizada a la izquierda y descentralizada a la derecha. [1] 2

1.2 Concepto simplificado del entrelazamiento de bloques para garantizar la inmutabilidad de la *blockchain*. . . . 3

1.3 Concepto del algoritmo del árbol de Merkle. . . . 4

1.4 Datos incluidos en la cadena de bloques. . . . 4

1.5 Ventajas de la tecnología *blockchain*. [2] . . . 5

1.6 Esquema del sistema eléctrico tradicional.[3] . . . 6

1.7 Evolución del coste desglosado de las instalaciones de generación fotovoltaica en los últimos años en función de su tamaño. . . . 7

1.8 Esquema instalación de autoconsumo con almacenamiento y conexión a red. . . . 7

1.9 Evolución del coste final del kWh de almacenamiento en baterías de Ion-Litio durante los últimos años y previsión de la evolución que seguirán estos precios si se supone un ratio de aprendizaje del 18 %. . . . 8

1.10 Concepto de prosumidor. . . . 9

1.11 Previsión de la evolución del coste de las instalaciones de generación fotovoltaica según diferentes agencias. . . . 10

1.12 Microrred eléctrica. . . . 11

1.13 Microrredes eléctricas interconectadas dentro de la red de baja tensión. . . . 12

1.14 Esquema de una microrred distribuida. . . . 13

2.1 Clasificación de las arquitecturas *blockchain*. . . . 19

2.2 Resumen de las principales características de las estrategias de consenso. . . . 21

2.3 Encuesta sobre la plataforma empleada en 140 proyectos del sector de la energía en compañías y proyectos de investigación. . . . 24

2.4 Elementos que intervienen en el proceso de identificación de usuarios mediante una infraestructura de clave pública. . . . 27

2.5 Representación gráfica del proceso de firma de un documento mediante la clave privada, y verificación de la identidad del emisor e integridad del mensaje mediante la clave pública. 28

2.6 Proceso de creación de certificados digitales por parte de una autoridad certificadora. . . . 28

2.7 Ejemplo de un contrato inteligente empleado en la transferencia de la propiedad de un coche. 30

2.8 Pantalla de inicio de la herramienta *Playground* . . . 32

2.9 Entorno de programación de la herramienta *Playground* 33
2.10 Pantalla de pruebas de la herramienta *Playground* 33

3.1 Representación gráfica de las cuatro cadenas de bloques que componen el sistema. 35

5.1 Salida en el terminal de Linux de la primera prueba. 55
5.2 Resultados de mercado de la primera prueba. 56
5.3 Resultados de casamiento de desajuste de la primera prueba. 57
5.4 Resultados de la segunda prueba. 57
5.5 Análisis de la capacidad de almacenamiento necesaria para almacenar la base de datos correspondiente a la cadena de bloques de registro de los parámetros eléctricos. 59
5.6 Estructura con dos equipos y tres nodos. En el primer equipo se encuentra la autoridad certificadora y el ordenante, que inserta en el resto de nodos los bloques de transacciones. 60
5.7 Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando el usuario instalado en el equipo 1. 61
5.8 Análisis del ancho de banda necesario para la inserción de los datos de los contadores de diez usuarios en la cadena de bloques, estando estos instalados en el primer equipo. 62
5.9 Análisis del ancho de banda necesario para la inserción de los datos de los contadores de cien usuarios en la cadena de bloques, estando estos instalados en el primer equipo. 62
5.10 Análisis de los datos transmitidos entre los puertos del equipo que aloja la autoridad certificadora, el ordenante y dos nodos, con cien usuarios instalados insertando los datos de sus contadores. 63
5.11 Uso de los procesadores del primer equipo con cien usuarios instalados en él. 63
5.12 Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando el usuario instalado en el equipo 2. 65
5.13 Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando diez usuarios instalados en el equipo 2. 65
5.14 Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando cien usuarios instalados en el equipo 2. 66
5.15 Análisis del uso de capacidad de computación de ambos equipos, estando cien usuarios instalados en el equipo 2. 67
5.16 Salida de la terminal Linux del equipo gestor del sistema ejecutando rondas de mercado cada 5 minutos. 69
5.17 Análisis del ancho de banda necesario durante la ejecución de la fase previa y la fase posterior de un periodo de intercambios de energía. 70
5.18 Análisis de la capacidad de computación empleada durante la ejecución de las fases previa y posterior al periodo de intercambios de energía. 71
5.19 Análisis de la capacidad de almacenamiento necesaria para alojar la base de datos. 71

B.1 Modificación del archivo `crypto-config.yaml`. 123

Índice de tablas

2.1 Resumen de las principales características de *Ethereum* e *Hyperledger* 25

5.1 Datos de los usuarios prosumidores de la prueba 1 54

5.2 Datos de los usuarios prosumidores de la prueba 1 56

5.3 Variaciones en la cantidad de datos almacenados en la cadena de bloques de registro de los parámetros eléctricos 59

5.4 Variaciones en la cantidad de datos almacenados en las bases de datos de las cadenas de bloques 72

C.1 Presupuesto de las herramientas y recursos empleados para el desarrollo de la plataforma de intercambios de energía entre usuarios prosumidores en una microrred. 129

C.2 Presupuesto de la adquisición e instalación del equipo necesario para que el prosumidor pueda conectarse a la plataforma de intercambios de energía. 131

C.3 Presupuesto de la adquisición e instalación de los equipos necesarios para la gestión de la plataforma de intercambios de energía 131

Índice de listados de código fuente

A.1 org.creacion.cto – Código de definición del modelo de la cadena de bloques que contiene los datos personales de los usuarios prosumidores (lenguaje CTO) 81
A.2 script.js – Código de las transacciones definidas en la cadena de bloques que contiene los datos personales de los usuarios prosumidores (lenguaje node.js) 82
A.3 queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que contiene los datos personales de los usuarios prosumidores (lenguaje propio inspirado en SQL) 83
A.4 permissions.acl – Código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene los datos personales de los usuarios prosumidores 84
A.5 org.mercado.cto – Código de definición del modelo de la cadena de bloques del mercado, donde se almacenan las ofertas de los usuarios, los resultados del mercado y los precios de la red (lenguaje CTO) 86
A.6 script.js – Primera parte del código de las transacciones definidas en la cadena de bloques del mercado (lenguaje node js) 87
A.7 script.js – Segunda parte del código de las transacciones definidas en la cadena de bloques del mercado (lenguaje node js) 88
A.8 script.js – Tercera parte del código de las transacciones definidas en la cadena de bloques del mercado (lenguaje node js) 89
A.9 queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que contiene los datos del mercado y los precios de los intercambios con la red (lenguaje propio inspirado en SQL) 90
A.10 permissions.acl – Primera parte del código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene los datos del mercado y los precios de los intercambios con la red 91
A.11 permissions.acl – Segunda parte del código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene los datos del mercado y los precios de los intercambios con la red 92
A.12 org.creacion.cto – Código de definición del modelo de la cadena de bloques donde se registran los datos de los contadores inteligentes de los usuarios prosumidores (lenguaje CTO) 93
A.13 script.js – Código de las transacciones definidas en la cadena de bloques que almacena los parámetros eléctricos de los usuarios prosumidores (lenguaje node.js) 94
A.14 queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que almacena los parámetros eléctricos de los usuarios prosumidores (lenguaje propio inspirado en SQL) 94
A.15 permissions.acl – Código con los permisos de acceso, lectura, creación y edición para la cadena de bloquesque almacena los parámetros eléctricos de los usuarios prosumidores . . 95

A.16 org.liquidacion.cto – Código de definición del modelo de la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje CTO) 97
A.17 script.js – Código de las transacciones definidas en la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje node.js) 98
A.18 script.js – Segunda parte del código de las transacciones definidas en la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje node.js) 99
A.19 script.js – Tercera parte del código de las transacciones definidas en la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje node.js) 100
A.20 queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje propio inspirado en SQL) 101
A.21 permissions.acl – Código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene todas las transacciones que se realizan 102
A.22 instalacion.sh – Conjunto de comandos que se deben ejecutar para crear una red de *Hyperledger Fabric* con el nodo principal en el ordenador del gestor del sistema e instalar las cadenas de bloques 103
A.23 instalacionordenador2.sh – Conjunto de comandos que se deben ejecutar en los equipos de los usuarios prosumidores para unirse a la red de *Hyperledger Fabric* creada por el gestor del sistema 103
A.24 crearNUsuarios.sh – Conjunto de comandos que llama a la función crearUsuario.sh N veces y deja al cliente del prosumidor corriendo en segundo plano. Se emplea únicamente en pruebas para simular un gran número de prosumidores. 103
A.25 crearUsuario.sh – Conjunto de comandos que se deben ejecutar en el equipo del prosumidor para dar de alta a un usuario y crear la carpeta que posteriormente se copia en su equipo 104
A.26 instalarUsuario.sh – Conjunto de comandos para instalar el usuario del prosumidor en su dispositivo. 105
A.27 desinstalarUsuario.sh – Conjunto de comandos que elimina la instalación de las tarjetas de identificación del usuario prosumidor. 105
A.28 desinstalarUsuarios.sh – Desinstala todos los usuario prosumidores. Es un conjunto de comandos empleado únicamente en las pruebas 105
A.29 crearProsumer.js – Programa en node.js que crea un usuario en cada una de las cadenas blockchain con sus activos correspondientes 107
A.30 crearProsumer.js – Programa en node.js que crea un usuario en cada una de las cadenas blockchain con sus activos correspondientes 108
A.31 crearProsumer.js – Programa en node.js que crea un usuario en cada una de las cadenas blockchain con sus activos correspondientes 109
A.32 ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía 111
A.33 ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía 112
A.34 ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía 113
A.35 ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía 114
A.36 ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía 115

A.37 ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía 116
A.38 ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía 117
A.39 ClienteProsumer.js – Programa en node.js que conecta al usuario prosumidor con las cadenas de bloques, inserta los datos de los parámetros eléctricos y las ofertas para la subasta de energía 119
A.40 ClienteProsumer.js – Programa en node.js que conecta al usuario prosumidor con las cadenas de bloques, inserta los datos de los parámetros eléctricos y las ofertas para la subasta de energía 120

Capítulo 1

Introducción

Este trabajo fin de máster (TFM) tiene como objetivo la creación de una aplicación que haga uso de la tecnología de cadena de bloques, o como se la conoce habitualmente, por su nombre en inglés, tecnología *blockchain*, para realizar intercambios de energía entre prosumidores dentro de una microrred, gracias a la cual se podrá garantizar, en todas las fases del proceso, la inmutabilidad y veracidad de:

- La identidad de todos los agentes implicados en el intercambio.
- El mercado minorista, incluyendo las ofertas realizadas por los usuarios de la aplicación, así como los resultados que esta devuelve al casar las ofertas entre sí.
- Todos los parámetros eléctricos de manera continua (voltajes, intensidades, potencias activas y reactivas tanto consumidas como generadas, etc.)
- Los pagos y penalizaciones derivados de los intercambios energéticos efectuados.

Con la intención de explicar cómo se ha realizado el proceso de creación de la aplicación, el presente capítulo, se inicia con una introducción a la tecnología de cadena de bloques (1.1) y a las microrredes eléctricas (1.3), así como una breve exposición de las aplicaciones potenciales que diversos autores han visto de esta tecnología en este ámbito (1.4). En el capítulo 2 se hace un repaso por la corta historia de la tecnología de cadena de bloques, se muestran varias de las arquitecturas que existen para la aplicación de esta tecnología en diferentes ámbitos, se selecciona una de ellas como arquitectura más apropiada para el desarrollo de la plataforma de intercambios de energía dentro de una microrred eléctrica, y se describe en detalle dicha arquitectura. En el capítulo 3, se explica de manera detallada la estructura que se ha desarrollado para crear la plataforma de intercambios de energía. En el siguiente capítulo (4), se describen los programas creados para interaccionar con la plataforma, tanto por parte del gestor del sistema, como de los usuarios prosumidores. Las pruebas de funcionamiento que se han realizado para comprobar la viabilidad del sistema se describen en el capítulo 5. Por último, se exponen las posibles líneas de trabajo futuras y las conclusiones que se han podido sacar de la realización del presente trabajo de fin de máster (6).

1.1 Tecnología *blockchain*

Una de las tecnologías más revolucionarias del siglo 21 es la tecnología de cadena de bloques, o en inglés, nombre con el que se ha popularizado en todo el mundo, la tecnología *blockchain*. Una cadena de bloques

es una base de datos en la que se registran todos los bloques de información derivados de los intercambios entre los distintos nodos de la red, y los entrelaza con el anterior bloque, que a su vez está enlazado con el previo a este, y así sucesivamente. Esta base de datos se almacena en todos los nodos de la red, y al ser pública, puede ser vista en cada parte. Si uno de los nodos presentase alguna modificación con respecto al resto sería eliminado de manera automática, lo cual garantiza la transparencia e inmutabilidad del sistema.

Tiene la capacidad de reemplazar a los sistemas de intercambios de activos tradicionales al ser una base de datos distribuida y segura (gracias al cifrado) que se puede aplicar a todo tipo de transacciones que no tienen por qué ser necesariamente económicas.

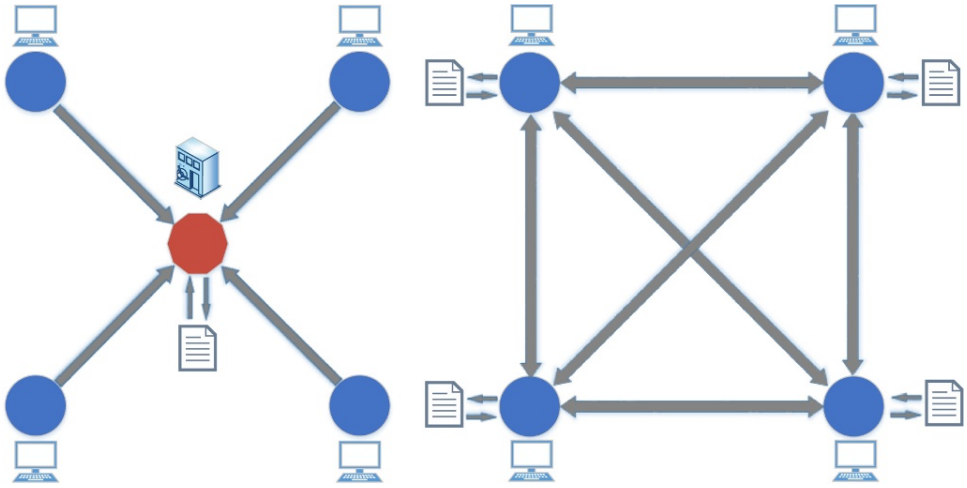


Figura 1.1: Plataforma de transacciones centralizada a la izquierda y descentralizada a la derecha. [1]

Es una solución distribuida, es decir, para que la tecnología *blockchain* tenga sentido, debe haber más de un nodo en el que se encuentre almacenada la cadena de bloques completa, y todos los nodos deben actualizarse de manera inmediata según se vayan incorporando nuevos bloques a la cadena. Al realizarse una transacción, esta ha de ser verificada mediante mecanismos de consenso de manera previa a ser replicada en todos los nodos de la red. Los nodos serán, por norma general, equipos con capacidad de computación suficiente como para realizar la función de verificación del nuevo bloque a añadir y capacidad de almacenamiento suficiente como para almacenar una base de datos siempre creciente[1].

Mediante esta tecnología se deja constancia de todas las transacciones realizadas, lo cual permite disponer de una trazabilidad completa del intercambio o venta de activos sin dejar oportunidad a las partes (involucrada o no en la transacción) de realizar cualquier tipo de modificación ya sea en busca de su propio beneficio o buscando un perjuicio para alguna de las partes involucradas. Cada una de las transacciones es verificada mediante un consenso y mecanismos de validación previamente definidos por todas las partes, sin la necesidad de la verificación de una autoridad central. [4]

1.1.1 Entrelazamiento de los bloques

Cada uno de los bloques de la cadena posee un código alfanumérico de longitud fija, calculado mediante técnicas de criptografía a partir de la información de todas las transacciones incluidas el bloque a través del algoritmo del *árbol de Merkle*, que se detalla en la sección 1.1.2, y el código alfanumérico correspondiente al bloque previo en la cadena. Esta cadena de caracteres se denomina *hash* y se emplea como si fuese

[1] Puede haber nodos, según las características de la cadena de bloques, que no tengan que realizar la función de verificación de las transacciones o almacenar la cadena de bloques completa, simplemente se conectan a la red para efectuar transacciones, que son verificadas y almacenadas por otros nodos.

la huella digital de este bloque. Al estar calculado a partir del *hash* del bloque previo en la cadena, si alguien realizara una modificación en alguno de los bloques anteriores, se sabría de manera inmediata, y la información contenida en el nodo con un *hash* erróneo sería descartada y sustituida nuevamente por la información válida, que continúa almacenada en el resto de nodos de la red. Se puede visualizar el funcionamiento del entrelazamiento de los bloques en la figura 1.2, dónde se observa que cada uno de los bloques posee dos datos, la información de todas las transacciones contenidas en el bloque calculada mediante el algoritmo del *árbol de Merkle* y el *hash* del bloque previo. Mediante estos dos datos se calcula el *hash* del bloque, que aparecerá en el siguiente bloque de la cadena, garantizando la inmutabilidad.[5]

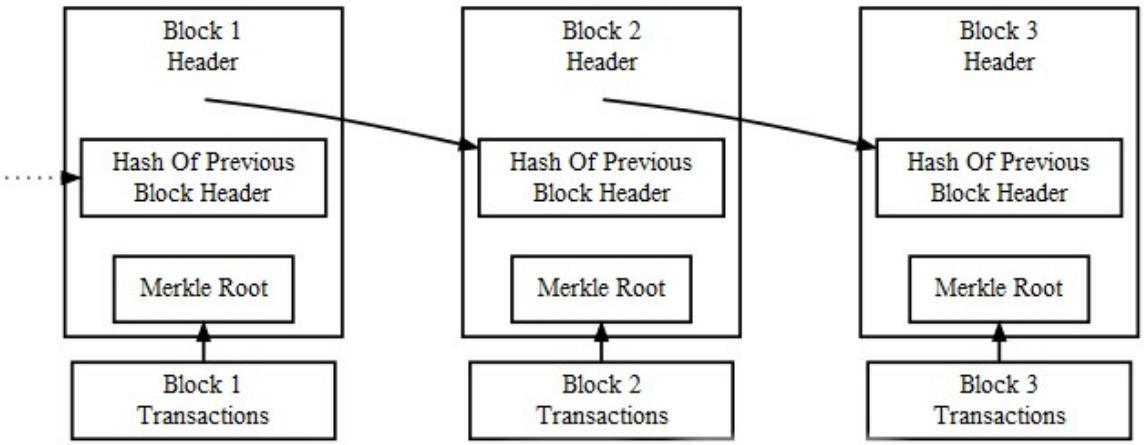


Figura 1.2: Concepto simplificado del entrelazamiento de bloques para garantizar la inmutabilidad de la *blockchain*.

1.1.2 Árbol de Merkle

Una función criptográfica *hash*, es un algoritmo que convierte cualquier bloque arbitrario de datos en una nueva cadena de caracteres con una longitud fija, de forma independiente del tamaño de los datos de entrada. [6]

El *árbol de Merkle* es el algoritmo empleado para simplificar los *hashes* correspondientes a cada una de las transacciones incluidas en un bloque de la cadena, hasta conseguir un único código alfanumérico de tamaño fijo denominado *Merkle Root Hash*. A partir de este código alfanumérico y el *hash* de cabecera del bloque anterior, se obtiene el *hash* de cabecera que caracteriza al bloque. [7]

El *árbol de Merkle* agrupa los *hashes* en pares de manera sucesiva, tal y como se puede observar en la figura 1.3, hasta que se obtiene el *Merkle Root Hash*, lo que permite conseguir un coste, en cuanto a recursos de computación y de tiempo, equivalente a validar una única transacción o miles de transacciones.

Gracias a la aplicación de este algoritmo, si se modifica alguna de las transacciones a partir de las que se calcula el *Merkle Root Hash*, este será modificado, lo que repercutirá en una modificación del *hash* de cabecera del bloque que contiene dicha transacción, y por consiguiente modificará también el *hash* de cabecera de todos los bloques situados posteriormente en la cadena.

Esta técnica permite localizar de forma rápida el origen de la modificación al comparar los *hashes* de una copia de la cadena de bloques válida y una copia modificada, sin la necesidad de realizar un esfuerzo computacional o de tiempo elevado, al no tener que ir comprobando todas y cada una de las transacciones de la cadena de bloques.

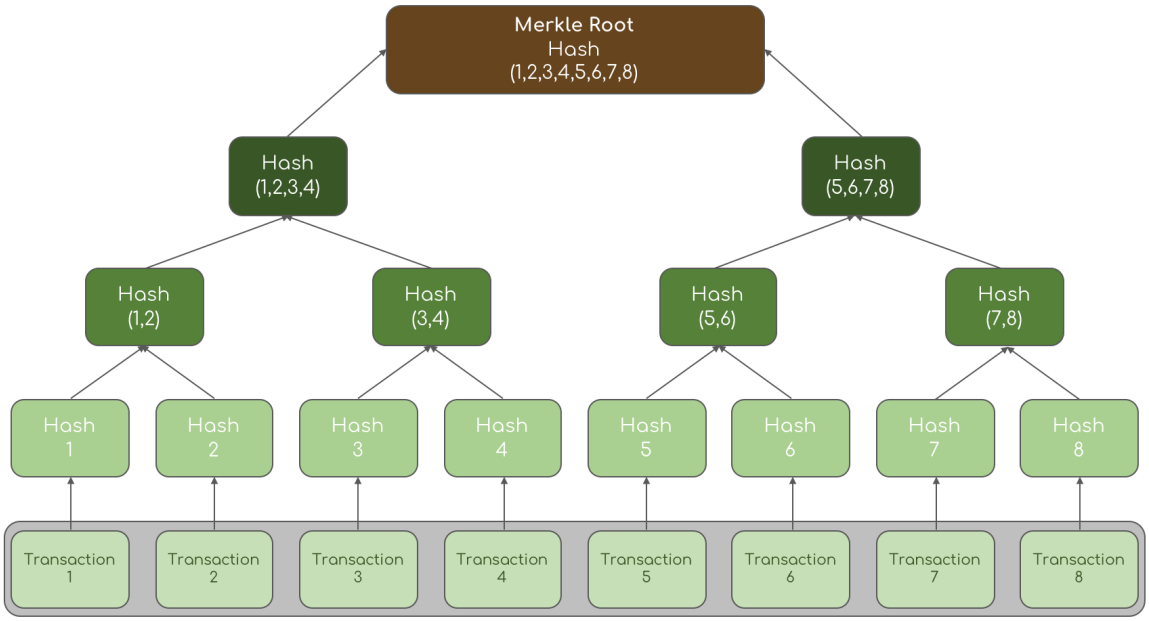


Figura 1.3: Concepto del algoritmo del árbol de Merkle.

1.1.3 Marca de tiempo

Cada bloque de la cadena posee, además, otro dato clave, la marca de tiempo. Este dato permite verificar que, cronológicamente, la última modificación de cada bloque se ha realizado en el momento correcto. Si se modificara de manera posterior alguno de los datos contenidos en un bloque, esto modificaría, en primer lugar, el *hash* de cabecera del bloque, lo cual quedaría reflejado en todos los bloques posteriores al modificar todos sus *hases* de cabecera, pero además, tendría una marca de tiempo posterior a la de los bloques posteriores, y por lo tanto sería automáticamente desechado.

En la figura 1.4 se pueden ver todos los parámetros que de manera obligatoria debe mostrar cada bloque, siendo especialmente relevantes el *hash* de cabecera del propio bloque y del previo y la marca de tiempo, a partir de los que se puede determinar la veracidad de los datos almacenados en cada uno de los bloques de la cadena.

Figura 1.4: Datos incluidos en la cadena de bloques.

1.1.4 Ventajas del *blockchain*

El *blockchain* es, según lo visto en los apartados previos de esta sección, una base de datos continuamente creciente de registros, distribuida, con una seguridad sin precedentes, y todo apunta a que es la próxima tecnología que va a reinventar la forma en la que trabajamos y vivimos. Promete facilitar compartir información a través de la red con un marco de distribución más seguro que cualquier tecnología previa.

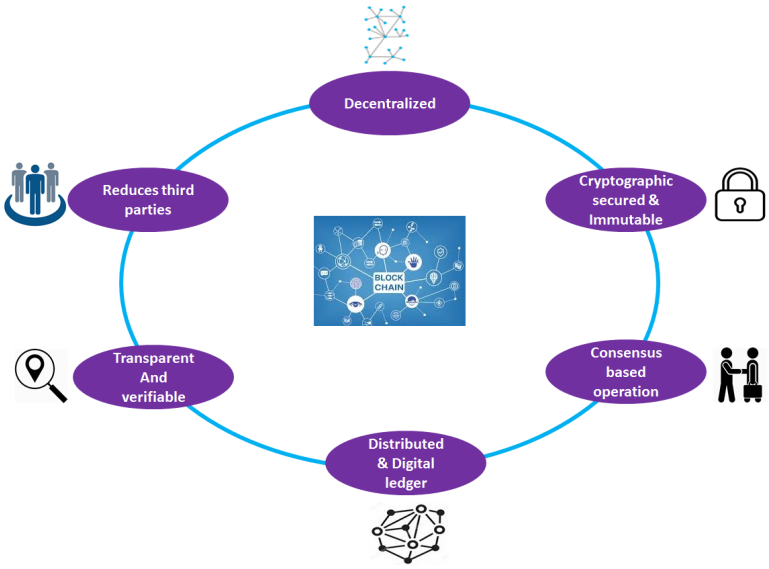


Figura 1.5: Ventajas de la tecnología *blockchain*. [2]

Entre sus numerosas ventajas se pueden destacar las siguientes:

- **Elimina las terceras partes:** Desaparece la dependencia de una tercera parte que participe en las transacciones como intermediario para garantizar que esta se realiza de acuerdo a los términos preestablecidos.
- **Control descentralizado:** No se depende de un único ente, es dependiente de varios nodos que han de comprobar que se cumplen los requisitos previamente establecidos.
- **Inmutable y protegido mediante criptografía:** Garantiza que los datos no van a ser modificados una vez introducidos en la cadena de bloques.
- **Control basado en el consenso:** Se preestablece un conjunto de características para la aprobación de la inclusión de transacciones: si se cumple el consenso, la transacción es incluida en la cadena de bloques y replicada en todos los nodos de forma inmediata, si no se cumple el consenso la transacción se rechaza.
- **Está distribuida en varios nodos:** Al ser una base de datos distribuida en diversos dispositivos presenta la ventaja de que aunque una copia sea eliminada o modificada de manera intencionada o por error, estos datos no se pierden. Además, para la consulta de los datos resulta una ventaja tener los datos disponibles en el propio nodo.
- **Transparente y verificable:** Todas las transacciones quedan almacenadas en la cadena de bloques y pueden ser visualizadas y comprobadas en el momento que se desee por todos los usuarios autorizados para ello.

1.2 Estado de la red eléctrica en la actualidad

La red eléctrica actual parte de la base de grandes centros de producción (potencias de cientos o incluso miles de megavatios) desde los que se distribuye la energía generada hacia los millones de consumidores, situados en la mayoría de los casos a cientos de kilómetros, y tras tener que haber pasado la electricidad hasta llegar a su destino por varios transformadores para primero elevar el voltaje (con el objetivo de

minimizar las pérdidas en el transporte al reducir la intensidad y además abaratar los costes de infraestructura al poder emplear conductores de menor grosor) y después reducirlo hasta las necesidades del consumidor tal y como se puede ver en la figura 1.6 [3].

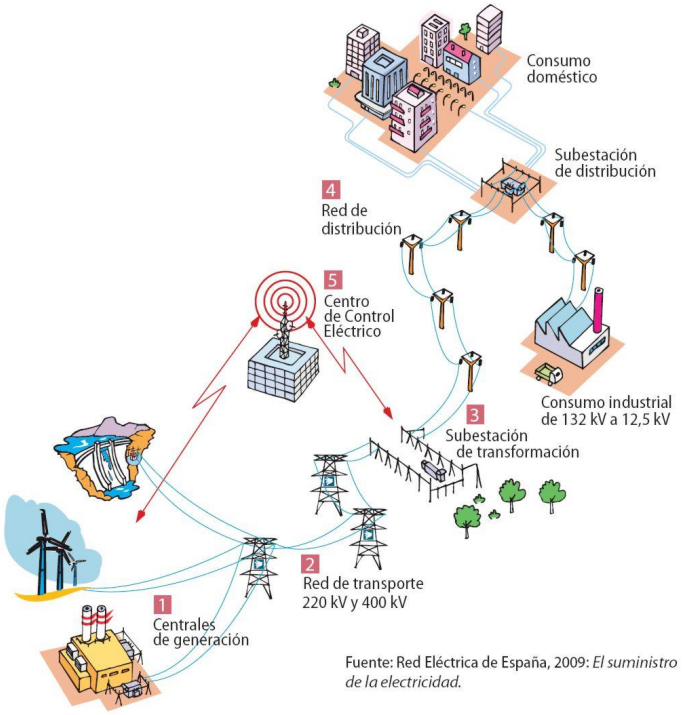


Figura 1.6: Esquema del sistema eléctrico tradicional.[3]

1.2.1 Problemas de la red eléctrica actual

Este concepto de red eléctrica implica, por su propia definición, una gran inversión en infraestructura (cables de gran capacidad, subestaciones, grandes centrales de generación, ...) y un desperdicio de recursos en forma de pérdidas en el transporte de la energía, lo que repercute en pérdidas no solamente económicas, también medioambientales, por haber tenido que producir una electricidad superior a la demandada para compensar las pérdidas que se producen durante el transporte y poder entregar al consumidor la energía que realmente necesita.

1.2.2 Evolución de la red eléctrica

La red eléctrica está evolucionando por sí misma gracias al auge y abaratamiento de las energías renovables (ver figura 1.7). Ahora pequeños usuarios pueden afrontar la compra de un pequeño generador de electricidad (principalmente fotovoltaico), que pueden emplear para abastecer su consumo propio o verter a la red para que lo consuman el resto de usuarios.

Gracias a los usuarios con capacidad de generación eléctrica, se reduce sustancialmente el problema de las pérdidas en el transporte energía desde el punto de generación hasta el punto de consumo al reducirse drásticamente la distancia (en el caso tradicional había que transportar la energía varios centenares de kilómetros, y en este nuevo escenario apenas decenas de metros) y en los centros de transformación, pues en este nuevo escenario hay un único convertidor de potencia entre el equipo generador y la red de consumo del usuario, en lugar de varias subestaciones como sucede en el escenario tradicional.

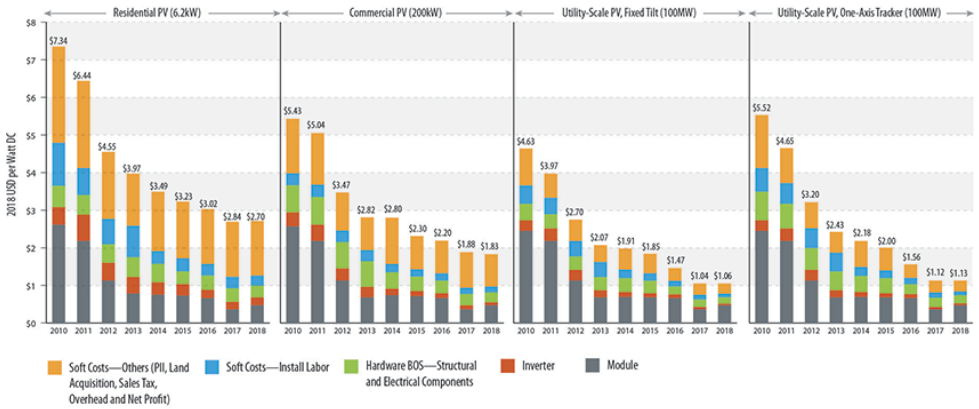


Figura 1.7: Evolución del coste desglosado de las instalaciones de generación fotovoltaica en los últimos años en función de su tamaño.

Esta opción comienza a ser muy interesante desde el punto de vista tanto económico como ecológico. Al ser la electricidad generada por el propio consumidor se evita, en primer lugar, numerosos intermediarios cuya finalidad, en muchos casos, es lucrarse con el negocio de la energía (empresas generadoras, gestores de la red de transporte de alta y media tensión, gestores de la red de distribución y empresas comercializadoras) y, por lo tanto, repercute en un coste muy inferior al de la electricidad extraída de la red. En segundo lugar, las pérdidas son significativamente más pequeñas, pues al ser de menor tamaño y más distribuidos los centros de generación, se tendrá que transportar menos energía y a una distancia menor, lo que es de interés por el ahorro de costes (tanto por la inversión en nueva infraestructura, como por la disminución del porcentaje de pérdidas), así como por el beneficio medioambiental que esto supondría al no tener que producir una energía superior a la necesaria para compensar las pérdidas.

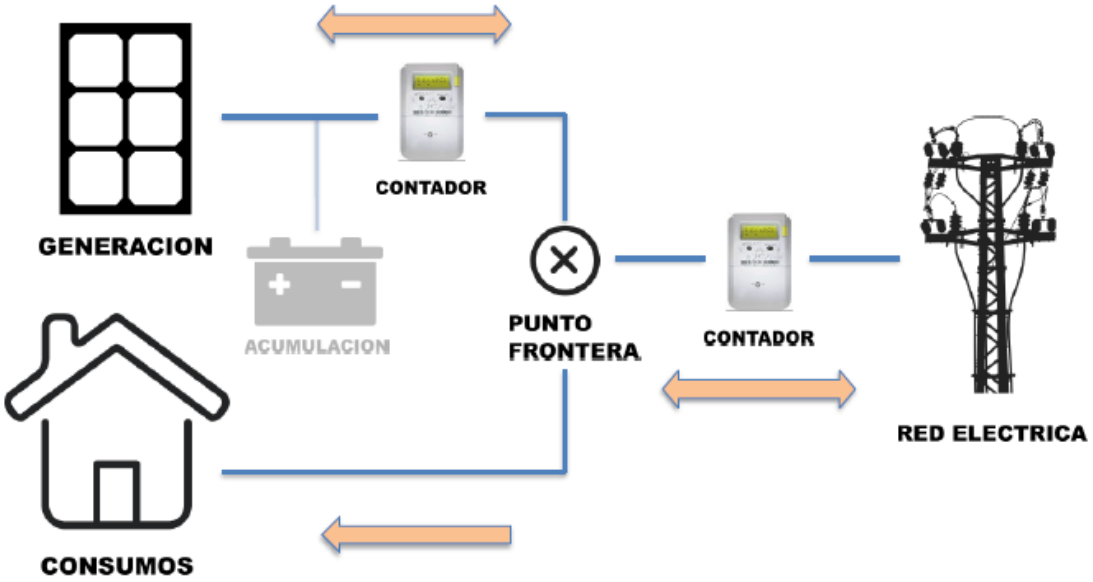


Figura 1.8: Esquema instalación de autoconsumo con almacenamiento y conexión a red.

El único inconveniente de una instalación de autoconsumo aislada es la impredecibilidad de generación, que ha de ser solucionada a través de una conexión a la red de baja tensión y/o mediante la instalación de un sistema de almacenamiento de energía que pueda proporcionar un margen para absorber los desajustes entre la generación y el consumo, ambos impredecibles al detalle, y por lo tanto imposible de mantenerse

de manera aislada con una calidad de la electricidad que cumpla con la normativa.

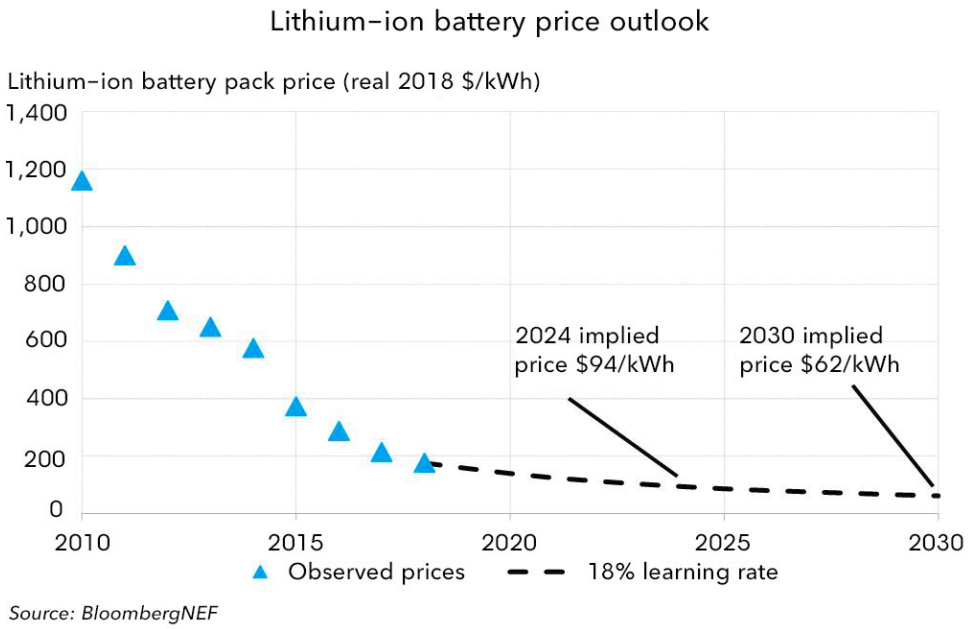


Figura 1.9: Evolución del coste final del kWh de almacenamiento en baterías de Ion-Litio durante los últimos años y previsión de la evolución que seguirán estos precios si se supone un ratio de aprendizaje del 18 %.

La otra evolución que ha tenido lugar en los últimos años en el sistema eléctrico es como consecuencia del progresivo abaratamiento de las baterías, que se han comenzado a instalar en numerosos puntos de generación o consumo como sistemas de almacenamiento de energía eléctrica. Ahora pequeños usuarios pueden abordar la compra de una batería en la que almacenar energía adquirida, bien de la red o bien de su propio sistema de generación, y tenerla disponible para emplearla cuando se desee con el consecuente ahorro económico, al poder llenar la batería durante los periodos en los que el precio sea más bajo (o incluso gratuito si se está produciendo "in-situ" mediante un sistema con bajo coste de operación como la energía fotovoltaica) y consumirla (y, por tanto, no tener que adquirirla de la red) o venderla cuando el coste sea más elevado.

En la figura 1.8 se puede observar el esquema simplificado típico de una instalación de autoconsumo con una placa fotovoltaica, almacenamiento y conexión a la red. En este caso se dispone de un sistema fotovoltaico conectado a una batería que absorbe las diferencias entre el consumo del hogar y la generación de las placas fotovoltaicas. Además, se dispone de una conexión a la red eléctrica de baja tensión para que, en caso de no poder ajustar consumos y generación únicamente mediante la batería, no se produzca una caída del sistema. En este tipo de esquemas los contadores han de ser bidireccionales, es decir, deben tener la capacidad de medir los flujos de energía tanto entrantes como salientes.

1.2.3 Usuarios prosumidores

Gracias al abaratamiento de las fuentes renovables de energía (1.7), un gran número de usuarios de la red puede abordar la inversión necesaria para instalar un pequeño punto de generación eléctrica renovable (principalmente fotovoltaico o eólico), para dar servicio, en primer lugar, a su propio consumo. Debido a las fluctuaciones tanto de producción como de demanda de energía, se producen instantes con un exceso de generación, que puede ser inyectado a la red. Asimismo, habrá momentos en los que la producción de energía sea insuficiente para el consumo que se esté realizando, y se tendrá que extraer más energía

de la que produce el generador renovable. Se crea por tanto un entorno con una red bidireccional, con intercambios de energía constantes y con un alto grado de impredecibilidad.

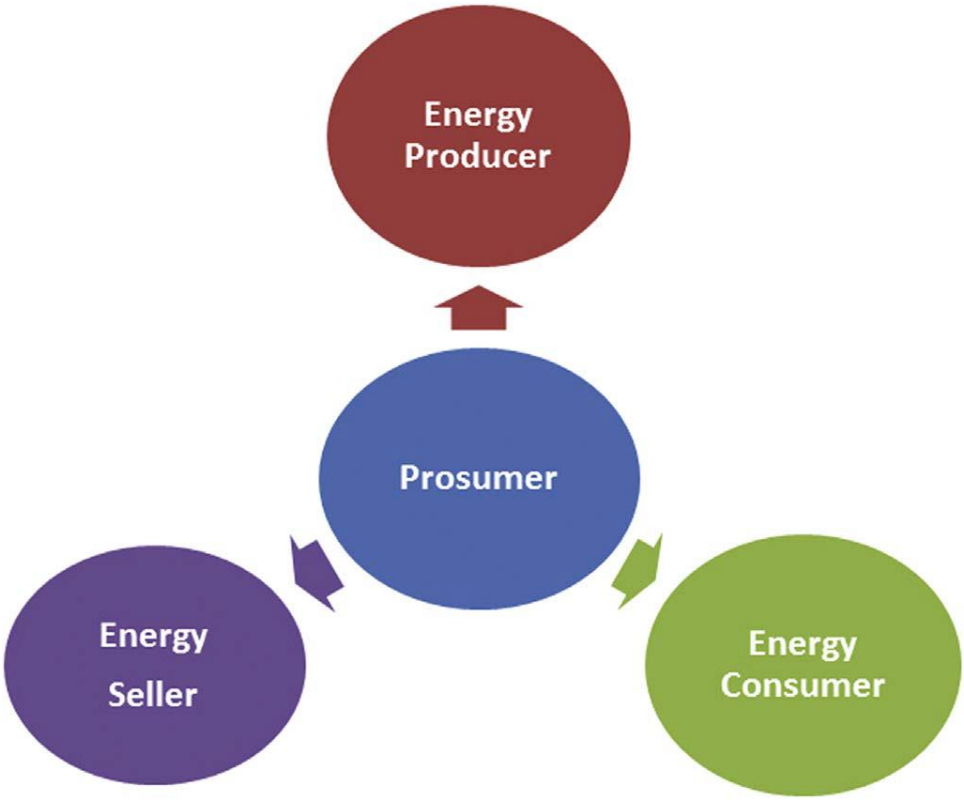


Figura 1.10: Concepto de prosumidor.

Con esta nueva situación de la red eléctrica, pequeños generadores distribuidos por todo el territorio y pequeñas baterías disponibles para absorber excesos de energía o para entregar a la red en caso de déficit, nace un tipo de usuario de la red eléctrica que no se conocía hasta ahora, el prosumidor. Este usuario tiene varios papeles en función de su interacción con la red (figura 1.10), en lugar de un único papel como tenían hasta ahora todos los usuarios de la red eléctrica: puede ser consumidor de energía (generada por él o absorbida de la red), productor de energía (para su propio consumo o para inyectarla a la red) y vendedor de energía (producida en ese mismo instante con un generador propio o energía almacenada en una batería que resulta interesante económicamente vender).

1.2.4 Gestión de los usuarios prosumidores

Debido a la impredecibilidad de las energías renovables, su generación presenta numerosas dificultades para ajustarse a un consumo, también variable, en el marco de las redes eléctricas actuales diseñadas para tener grandes productores que inyectan energía sin distinciones a la red, y una gran cantidad de consumidores que extraen la energía que necesitan de la red.

Actualmente, cuando la energía producida por los generadores de pequeño tamaño se vierte a la red lo hace sin ningún tipo de control, ni el gestor de la red eléctrica sabe cuanta energía está vertiendo a la red cada pequeño productor, ni los productores tienen la certeza de a quien están vendiendo su energía, ni los pequeños consumidores tienen la certeza de quién ha producido la energía que están consumiendo, por lo que no existe un control sobre la generación o consumo de estos usuarios que vaya más allá que el propio interés económico que tengan ellos mismos. Esta situación es sostenible por sí misma mientras los usuarios que inyectan energía a la red supongan un pequeño, casi despreciable, porcentaje de la generación, pero

las previsiones actuales de evolución de los costes de las instalaciones fotovoltaicas en los próximos años (ver figura 1.11) hacen intuir que el número de pequeños generadores va a incrementarse de manera exponencial, por lo tanto, es indudable que se debe realizar un control sobre esta generación, que debe dejar de ser una incógnita y ayudar a facilitar la operación de la red.

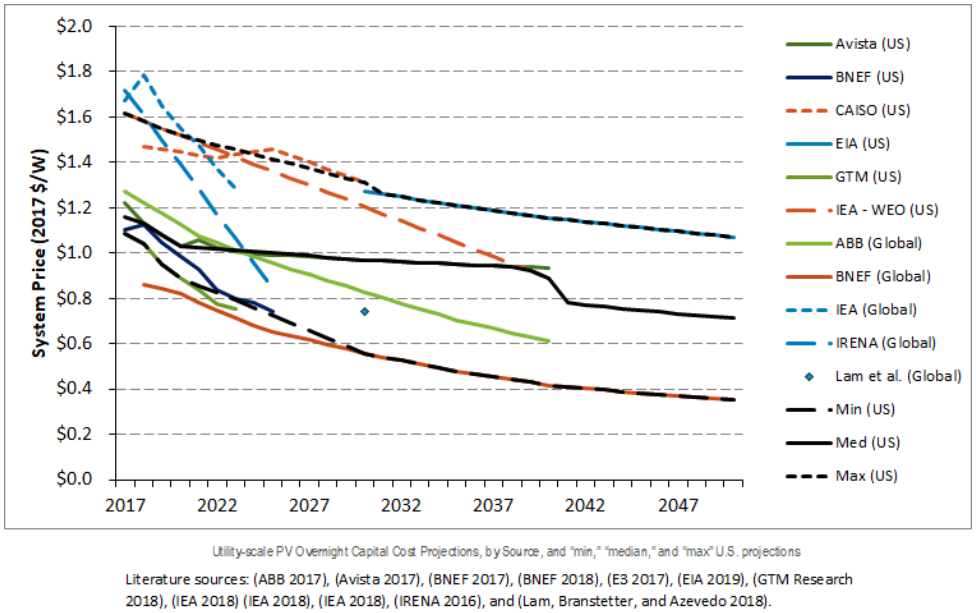


Figura 1.11: Previsión de la evolución del coste de las instalaciones de generación fotovoltaica según diferentes agencias.

Con este nuevo escenario que implica tener un enorme número de generadores de pequeño tamaño que gestionar para dar energía a un número aún mayor de consumidores, se hace indispensable para lograr un correcto funcionamiento, la posibilidad de un flujo bidireccional tanto de la energía como de la información, con una red de telecomunicaciones que permita conocer en todo momento el estado actual de la red a todos los usuarios de esta.

El concepto que actualmente se está aplicando para la inclusión de las redes de información en el sistema eléctrico se denomina *Enernet*. *Enernet* es una palabra compuesta por dos términos, energía y *net*, que es red en inglés, y es un concepto que considera la aplicación del modelo de Internet a la red eléctrica, dotándola de una cierta inteligencia que va a producir que tanto el consumo como la generación de energía ganen en eficacia, sostenibilidad, seguridad y rentabilidad.

Gracias a la incorporación de las telecomunicaciones en las redes de energía, se podrá identificar en todo momento a cada uno de los productores y consumidores, conociéndose las necesidades de cada uno de ellos, lo que permitirá un mayor control sobre toda la red, mejorándola en numerosos aspectos y optimizando la realización de intercambios de energía entre varios prosumidores. Además, en una fase más avanzada, se podrá identificar cada elemento consumidor, generador o almacenador de energía eléctrica, mejorando aún más el control sobre la red, pudiendo elegir en qué momento inyecta o absorbe energía de la red cada uno de ellos.

Una optimización de las telecomunicaciones para la transmisión de la información relativa a las redes eléctricas, permitiría una administración eficiente, lo que repercutiría en una reducción de la inversión en infraestructura, mientras se sigue garantizando el objetivo principal de la red, que es el suministro de energía de calidad a todos los usuarios de la misma.

1.3 Microrredes eléctricas

Una microrred eléctrica es un sistema de generación de electricidad bidireccional que permite la distribución de energía hasta los consumidores haciendo uso de las telecomunicaciones y la electrónica (figura 1.12) facilitando la integración de los generadores distribuidos de pequeño tamaño (principalmente de origen renovable), con el objetivo de ahorrar energía, reducir costes e incrementar la fiabilidad de suministro [8].

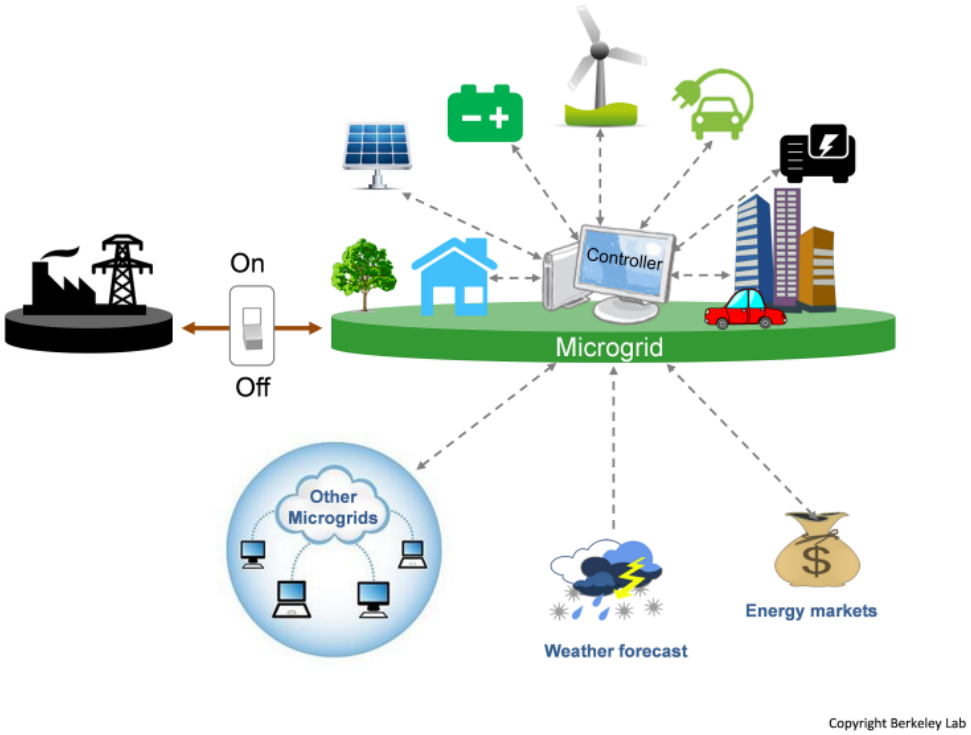


Figura 1.12: Microrred eléctrica.

Está compuesta principalmente por los siguientes elementos [9]:

- Red de distribución en baja tensión.
- Fuentes de energía distribuidas.
- Un conjunto de cargas consumidoras de energía.
- Una infraestructura de comunicación local.
- Un sistema de control y gestión.
- Elementos almacenadores de energía.

Una de sus principales ventajas es la posibilidad de funcionamiento de manera aislada de la red principal, lo que supone un aumento de la resiliencia al poder mantener el suministro en caso de caída de la red principal.

Cuando la microrred funciona en modo conectado con la red principal, esta proporciona las referencias de tensión y frecuencia para que el resto de elementos funcionen sin ningún problema. Al trabajar en modo aislado, los elementos generadores pueden no ser capaces de responder con la rapidez necesaria a

las variaciones de consumo para conseguir mantener la tensión y la frecuencia dentro de los márgenes requeridos, por tanto, se hace indispensable, para el correcto funcionamiento en este modo, el empleo de elementos almacenadores de energía, que absorban los desajustes entre el consumo y la generación. En el modo aislado los sistemas de almacenamiento deben estar conectados a la red de baja tensión a través de inversores con controles que sean capaces de mantener la tensión y la frecuencia en los márgenes adecuados para que el resto de elementos de la red puedan seguir sus referencias.

Para que las microrredes desarrollen todo su potencial, deben tener un contador que contabilice los intercambios con la red principal, y de esta forma únicamente realizar pagos a la empresa comercializadora por los intercambios de energía que se produzcan entre la microrred y la red principal y no por las cantidades de energía que los contadores de cada uno de los prosumidores de la microrred han contabilizado que se han intercambiado con la red de distribución. Por ello, en este trabajo fin de máster se parte de la hipótesis teórica de que la red de baja tensión está gestionada y mantenida por un ente público sin ánimo de lucro, como podrían ser los ayuntamientos.

El sistema de control de la microrred debe contar en todo momento con la información de precios del mercado y las ofertas de generación y demanda de energía dentro de la microrred para determinar qué modo de funcionamiento (conectado a la red o aislado) va a emplear, y qué generadores y cargas van a funcionar o se van a desconectar.

Los sistemas de control más avanzados hacen uso de un mayor número de datos (históricos de generación y consumo, previsión climatológica, etc.), que combinados con los datos de generación y consumo actuales, a partir de técnicas de predicción e inteligencia artificial, permiten una optimización de los recursos y una mayor fiabilidad del sistema.

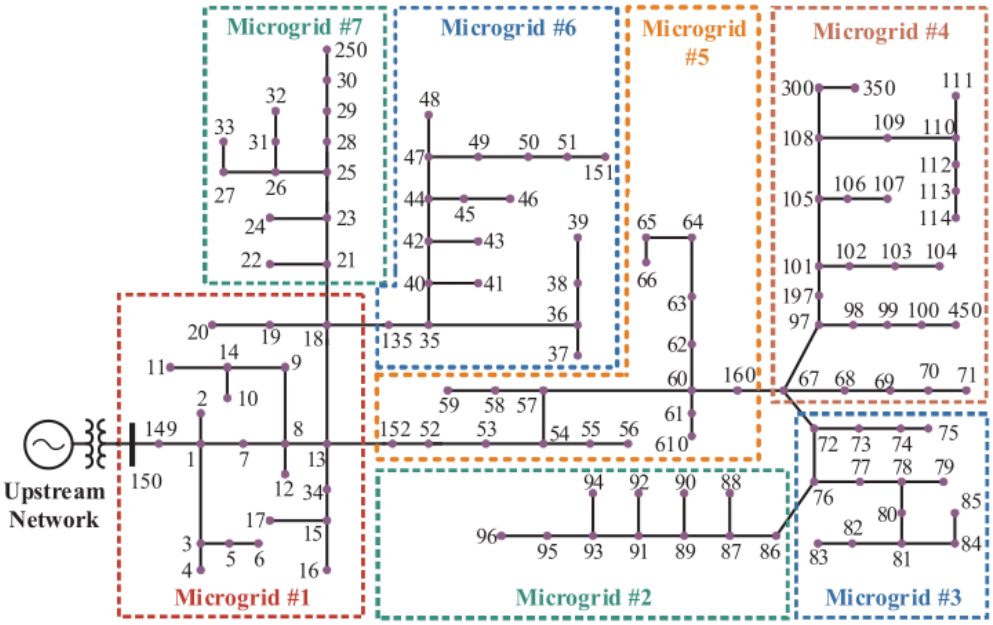


Figura 1.13: Microrredes eléctricas interconectadas dentro de la red de baja tensión.

Además, varias microrredes, geográficamente cercanas, pueden conectarse entre ellas para conseguir una mayor independencia de la red principal, mejorando aún más la fiabilidad del sistema y consiguiendo mejores precios tanto para los consumidores como para los generadores de electricidad. En la figura 1.13 se puede observar un sistema de siete microrredes, cada una de ellas encargada de la operación de una sección de la red de baja tensión.

1.3.1 Intercambios de energía en una microrred

Dentro de una microrred, es bastante improbable que cada uno de los prosumidores (1.2.3) tenga un equilibrio exacto entre lo que está generando y lo que está consumiendo, por ello, en lugar de aumentar la capacidad de almacenamiento (actualmente su coste es elevado), los excesos o defectos de generación con respecto al consumo pueden ser absorbidos por otro usuario de la microrred a cambio de un pago.

Este intercambio de energía, al realizarse dentro de la microrred, no es detectado por la red principal, y por lo tanto, no ha de pagarse ningún coste a la empresa comercializadora.

1.3.2 Microrredes distribuidas

Para conseguir realizar intercambios de energía de manera más eficiente, cada usuario de la microrred puede tener un sistema de gestión de la energía (EMS por sus siglas en inglés) que indique la forma de interactuar con la microrred que más interesa a dicho usuario, en lugar de existir un único sistema de gestión para toda la microrred que busque únicamente una estrategia global (normalmente la prioridad es garantizar el suministro y en segundo lugar minimizar los intercambios con la red principal). De esta forma cada usuario puede indicar a su EMS la estrategia que quiere seguir (reducir el coste, consumir únicamente energía renovable, etc.).

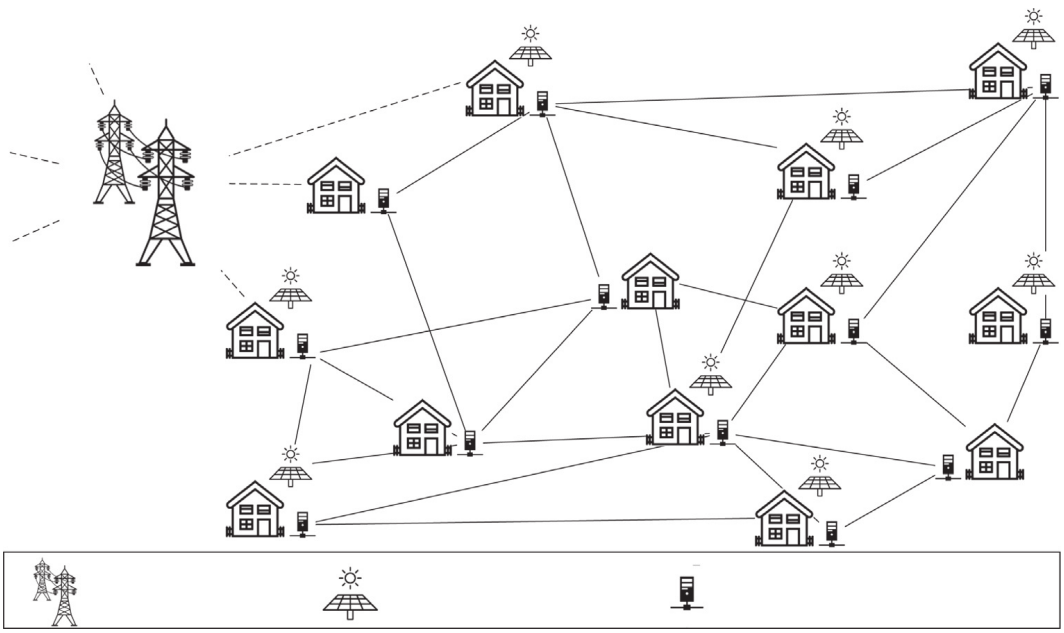


Figura 1.14: Esquema de una microrred distribuida.

En la figura 1.14 se puede observar el esquema de una microrred en la que cada uno de los usuarios prosumidores tiene un EMS propio, en constante comunicación con los sistemas de gestión de la energía del resto de prosumidores de la microrred para, de esta forma, conseguir una visión global del sistema, determinar la estrategia más apropiada (teniendo en cuenta las directrices previas dadas por el usuario) y dar las órdenes necesarias para generar o consumir una cantidad de energía adecuada a esta estrategia.

1.4 Aplicación de la tecnología *blockchain* a los intercambios de energía

Con el objetivo de flexibilizar y hacer más eficientes los procesos de intercambio de energía dentro de un marco de seguridad y rapidez de gestión, se puede hacer uso de la tecnología *blockchain* (sección 1.1), popularizada en el uso de "criptomonedas", pero que puede tener un gran número de aplicaciones fuera de los intercambios monetarios.

La posibilidad de aplicar la tecnología de cadena de bloques a las redes de energía ofrece numerosas ventajas tanto técnicas como económicas. Desde el punto de vista de un prosumidor, se puede trazar el origen de la generación, y asegurar si esta ha sido a partir de fuentes renovables o no y, de esta forma, asignar un valor concreto a cada "paquete de energía". La trazabilidad permitirá a los compradores de energía conocer toda la cadena de valor por medio de unos certificados que se generarán de manera automática y sin posibilidad de manipulación en ningún punto de la cadena.

Los sistemas de energía del futuro van a evolucionar respecto a los actuales según tres principios fundamentales: la descarbonización, la descentralización y la digitalización [1]. La estructura de mercados actual es incapaz de alcanzar estos principios desde el momento en que los pequeños usuarios que quieren participar en él están prácticamente excluidos. Las primeras investigaciones de aplicación del *blockchain* a las transacciones de energía han avanzando creando plataformas de comercio de energía P2P completamente descentralizadas, se están realizando mercados locales de energía y aplicaciones del internet de las cosas que pueden significar un gran avance en la evolución hacia la red inteligente (Smart Grid por su nombre en inglés) [10].

Las grandes eléctricas cada vez están reportando unos mayores costes de producción de energía y menores beneficios al mismo tiempo que las autoridades reguladoras exigen una mayor transparencia; como resultado se tiene que cualquier investigación que pueda suponer un ahorro de costes o una mejora en la operación de los sistemas y mercados eléctricos está siendo fuertemente apoyada por estas empresas [11]. Según Deloitte [12], la creación de plataformas de intercambio de energía basadas en el *blockchain* va a suponer un ahorro en los costes de operación, un aumento de la eficiencia, una mayor automatización y, por tanto, mayor velocidad de los procesos, un aumento de la transparencia y una menor inversión en infraestructura.

El *blockchain*, además, puede permitir la eliminación de la unidireccionalidad de la red eléctrica actual, permitiendo la integración de una gran cantidad de generadores de pequeño tamaño, que obtendrán un beneficio, económico o no, cuando inyecten energía a la red, y pagará un coste a otro productor, que puede ser de pequeño o gran tamaño, cuando extraiga energía de la red, conociéndose en todo momento quién genera, quién consume y el coste que ha tenido cada "paquete de energía", quedando registrado de manera inalterable por ninguno de los usuarios de la red, implicado o no en el intercambio.

Una ventaja añadida sería que, el empleo de la tecnología de cadena de bloques permitiría a los pequeños productores no tener que darse de alta como comercializadores de energía para vender su energía sobrante a pequeños consumidores, pues al realizar los intercambios, podría recibir una recompensa en forma de *token*, que posteriormente podría ser intercambiado por energía de nuevo. Al no llegar a obtener un beneficio económico en ningún momento, no estaría realizando nada ilegal al obtener el beneficio por la venta de la energía sobrante. De esta forma el pequeño generador podrá consumir una cantidad de energía por un valor igual a la que genera, sin tener que realizar una acumulación de energía en baterías, a un coste económico nulo a pesar de no realizar el consumo en el mismo instante que se produce la generación.

El potencial de aplicar la tecnología *blockchain* en el sector de la energía apenas acaba de comenzar

a verse, y cada vez comienzan a salir a la luz un mayor número de proyectos piloto, de investigación y *startups* que ven una aplicación real de esta tecnología en el sector eléctrico. [1]

Los aspectos donde se cree que puede tener un mayor impacto la aplicación de la tecnología de cadena de bloques en las operaciones relacionadas con el sector de la energía son las siguientes: [1]

- **Facturación:** Mediante el empleo de contadores inteligentes, contratos inteligentes y la tecnología de cadena de bloques, las compañías eléctricas podrán beneficiarse de soluciones como los pagos instantáneos según se usa la energía o la facturación automática, transparente e inmutable.
- **Comercio y mercados:** Mediante el empleo de plataformas de comercio distribuidas se podrán realizar pujas/ofertas de energía desde cualquier lugar.
- **Mercados locales:** La creación de mercados locales de energía aumentará notablemente el autoconsumo lo que repercutirá en la creación de microrredes autogestionadas y con pequeños intercambios de energía con el exterior.
- **Transferencia de datos:** Al aplicar el *blockchain* a la transferencia de datos para convertir la red eléctrica en una red inteligente, estos datos serán más seguros y transparentes lo que repercute en mejores sistemas de control y motorización.
- **Gestión de la red:** Al tener disponible la información de la capacidad de almacenamiento y de la energía disponible, se permite una gestión de la red más flexible y optimizada.
- **Gestión de identidades:** Se producirán beneficios en la protección de la privacidad de las transacciones al verificar quién puede ver cada uno de los datos que se almacenan en la cadena de bloques.
- **Recursos compartidos:** Puede mejorar los sistemas usados por numerosos usuarios, como una infraestructura de recarga de vehículos eléctricos.
- **Competencia:** Al permitir entrar a numerosos nuevos agentes en los mercados eléctricos, se producirá una bajada de los precios de la electricidad.

Pero el *blockchain* aún tiene que superar varias barreras antes de que se extienda su uso. Una de estas barreras está en la capacidad de escalabilidad manteniendo la seguridad, descentralización y un coste moderado (a día de hoy se han realizado muchas investigaciones de pequeño tamaño, pero que no han fructificado al intentar llevarlas a gran escala) [1].

Capítulo 2

Arquitectura de la red *blockchain*

En el presente capítulo se habla del origen de la tecnología de cadena de bloques de la mano del *Bitcoin* y, posteriormente, del *Ethereum* (2.1), que son la semilla de todas las arquitecturas *blockchain* actuales. En el siguiente apartado (2.2), se explican las diferentes arquitecturas, y en el apartado 2.3 se hace hincapié en una de las características más determinantes de una arquitectura, el algoritmo de consenso aplicado para determinar que el bloque de transacciones que se quiere incluir en la cadena es válido y se replica en todos los nodos de la red, para en el apartado 2.4 determinar cuál es la arquitectura de cadena de bloques más indicada para desarrollar la plataforma de intercambios de energía en una microrred eléctrica, objeto de este proyecto. En el último apartado de este capítulo (2.5), se detallan todas las características de la plataforma finalmente seleccionada, *Hyperledger Fabric*, y las herramientas que facilitan el desarrollo de aplicaciones en dicha arquitectura.

2.1 *Bitcoin* y *Ethereum*

La primera "criptomoneda", el *Bitcoin*, fue presentada en el año 2009, siguiendo el libro blanco de Satoshi Nakamoto [13], cuya identidad a día de hoy continúa siendo desconocida. Es una moneda electrónica que emplea la comunicación P2P de usuarios anónimos a través de Internet. Esta moneda digital transferida directamente entre usuarios no está controlada por ningún banco central, pero una red de ordenadores trabajando de manera colaborativa y el uso de criptografía garantiza su seguridad [14].

Cada usuario tiene una cartera digital a la que únicamente se puede acceder mediante su clave privada. La cartera digital, además, tiene una clave pública con la que se garantiza la identidad del usuario. Antes de iniciarse una transacción, ambas partes tienen que conocer la clave pública del otro usuario. La secuencia mediante la cual se transfieren fondos de una cartera digital a otra es la siguiente:

1. El usuario que envía dinero crea una transacción (cuántos fondos va a transferir y a quién).
2. La transacción es firmada por el usuario que envía dinero para verificar su identidad mediante su clave privada.
3. Se envía a la cartera digital identificada con la clave pública del usuario que va a recibir los fondos a través de la red *Bitcoin*.

Unos nodos denominados "mineros", que además, son los encargados del proceso de validación, agregan todas las transacciones en un único bloque. Cuando se tiene el bloque formado, todos los "mineros" comienzan a competir entre ellos tratando de resolver un puzzle criptográfico lo más rápido posible

(mediante poder de computación), para ganar el derecho a incluir el bloque en la cadena, lo que repercute en recibir una recompensa económica (12,5 *Bitcoins* en octubre de 2019). La solución es transmitida a la red y el resto de "mineros" comienzan a trabajar en el bloque siguiente. Los usuarios pueden estar seguros de que el bloque añadido a la cadena es válido por el enorme gasto computacional que ha supuesto producirlo, y que está unido con los bloques anteriores. De media se produce un bloque cada diez minutos [13].

Mientras el *Bitcoin* representa la aplicación de cadena de bloques más popular hasta la fecha, *Ethereum* ha dominado la creación de aplicaciones en conjunto con la tecnología de cadena de bloques. *Ethereum* [15] es una máquina virtual basada en *blockchain* que viene con su propio lenguaje embebido que permite a los usuarios crear sus propias aplicaciones para ejecutarse dentro de la cadena de bloques [16]. Introduce el concepto de los contratos inteligentes (Smart Contracts en inglés), que son programas ejecutables de manera automática (conforme a condiciones previamente dadas), dentro de la red *Ethereum*.

Aplicaciones como el *Bitcoin* representan un ecosistema completamente distribuido, en el que la integridad y la seguridad están completamente garantizados, pero es muy costoso energéticamente y requiere de un periodo de tiempo elevado para que las transacciones sean efectivas. Otras aplicaciones no tienen por qué estar completamente distribuidas, y por lo tanto, se han desarrollado otras arquitecturas que aumentan la velocidad de verificación y reducen el coste energético con el objetivo de hacer viable el empleo de la tecnología de cadena de bloques en el mayor número posible de campos [1], incluyendo aplicaciones que requieran el uso de la inserción de datos en la cadena de bloques en tiempo real.

2.2 Arquitecturas *blockchain*

Una red de cadena de bloques puede seguir diferentes arquitecturas dependiendo del caso de uso para la que haya sido diseñada la aplicación.

2.2.1 Nodos usuarios y validadores

Las redes *blockchain* están compuestas por nodos, que son cada uno de los equipos que se conectan a la red. Estos nodos pueden ser de dos tipos, usuarios y validadores, y pueden tener un único rol o ambos al mismo tiempo. En la figura 2.1, se puede observar que en la red de la izquierda todos los nodos tienen ambos roles, mientras que en la de la derecha unos nodos son validadores mientras que otros son usuarios.

- Los **nodos usuarios** pueden iniciar y recibir transacciones y, además, pueden tener una copia de la cadena de bloques en su equipo.
- Los **nodos validadores** tienen, además de permisos de lectura de la cadena de bloques, la responsabilidad de aprobar las modificaciones en la cadena mediante el seguimiento de un consenso que comprueba la validez de los cambios introducidos.

2.2.2 Redes públicas y privadas

Las redes *blockchain* también se clasifican según qué usuarios pueden unirse a ellas.

- Las **redes públicas** permiten a todos los usuarios de Internet unirse. En estas redes los usuarios son anónimos, no se conoce su identidad. Estas redes están completamente distribuidas, y cualquier usuario anónimo puede convertirse en validador.

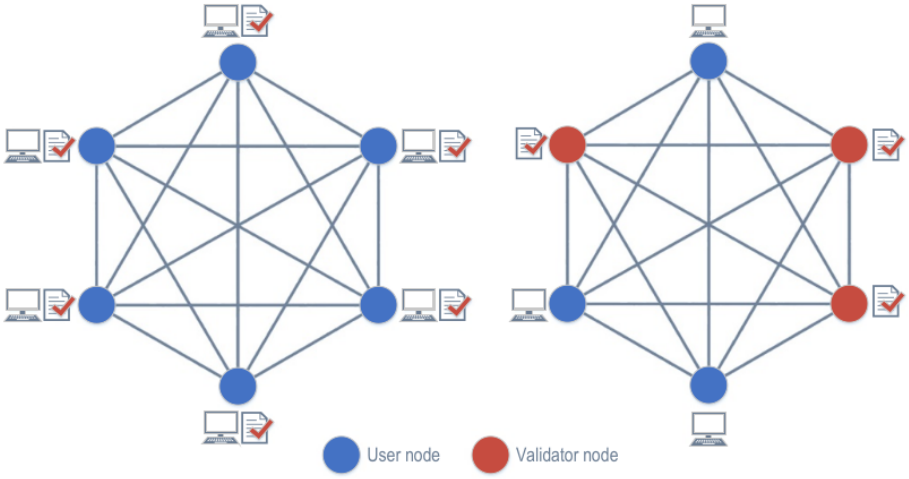


Figura 2.1: Clasificación de las arquitecturas *blockchain*.

- Por el contrario, en las **redes privadas o permisionadas**, únicamente pueden participar los usuarios que previamente han sido autorizados para ello. En este tipo de redes el rol de validador de transacciones es dado a determinados usuarios cuya identidad en conocida.

En la figura 2.1, la red de la izquierda representa una red pública, en la que todos los usuarios pueden unirse a la red con el papel que deseen, mientras que la red de la derecha representa una red privada o permisionada, en la que cada nodo ha sido autorizado a unirse a la red con un único rol ya determinado.

En las redes públicas se suele emplear un sistema de incentivos a la validación, para que haya usuarios que quieran realizar esta función, lo que supone un gasto de recursos (computación, electricidad, ...), que tiene como objetivo disuadir los comportamientos egoístas [17]. En cambio, en las redes privadas los validadores son conocidos y se confía en su veracidad, por lo tanto, no se requiere de incentivos para garantizar el funcionamiento del sistema. Como consecuencia, las redes permisionadas o privadas son muy superiores en cuanto a velocidad, flexibilidad y eficiencia, pero tiene el contrapunto de basarse en la confianza a esos validadores.

Existen también soluciones híbridas, que buscan un equilibrio entre las ventajas de las redes de uno y otro tipo.

2.2.3 Propósito general o específico

Las redes *blockchain* pueden también clasificarse según la aplicación para la que hayan sido diseñadas, teniendo dos tipos, que pueden identificarse cada uno con una de las dos redes más conocidas actualmente, *Bitcoin* y *Ethereum*. Mientras la red *Bitcoin* ha sido diseñada específicamente para el uso de la "criptomoneda" de igual nombre, *Ethereum*, por el contrario, ha sido diseñada para poder incluir un amplio rango de aplicaciones.

2.3 Diferentes consensos de validación de transacciones

El algoritmo de consenso es el proceso mediante el cual se mantienen sincronizados todos los nodos de la red para asegurar que la cadena solo acepta las transacciones aprobadas por los participantes adecuados y que una vez se ha actualizado un nodo, esta actualización se replica en el resto de nodos [18].

El proceso de inclusión de un bloque en la red de cadena de bloques se puede resumir en los siguientes pasos:

- Primero, un bloque es propuesto o generado por uno de los nodos de la red.
- Posteriormente, este bloque debe ser aceptado por los miembros de la red.
- Una vez que el bloque ha sido aceptado, este se incorpora a la cadena y se une criptográficamente a ella.
- Por último, pasado un tiempo (dependiendo del algoritmo de consenso empleado), el bloque se vuelve parte permanente e inmutable de la cadena.

En el año 2004 se introdujo el consenso de validación de prueba de trabajo reutilizable (RPoW por sus siglas en inglés). El sistema al recibir un *token* no intercambiable, mediante una prueba de trabajo, creaba un *token* firmado por RSA[1] que podría transferirse de una persona a otra. La identidad de los *tokens* se registraba en un servidor seguro que podía ser consultado por usuarios de todo el mundo en tiempo real. Este sistema fue el primer consenso de validación de transacciones, pero no sería hasta la creación del *Bitcoin* cuando una variante de este consenso se aplicó por primera vez en una red de cadena de bloques [20].

El *Bitcoin* estaba basado en el algoritmo de prueba de trabajo reutilizable. En primer lugar emplea un puzzle criptográfico como prueba de trabajo, pero en lugar de emplear un servidor seguro, la doble protección fue proporcionada por un protocolo descentralizado de igual a igual para el seguimiento y verificación de las transacciones; unos usuarios minan bloques para obtener una recompensa y posteriormente este bloque es verificado por los nodos verificadores descentralizados de la red. Este consenso se denomina algoritmo de prueba de trabajo (PoW por sus siglas en inglés).

Actualmente, existe un gran número de algoritmos de consenso tanto desarrollados como en desarrollo, cada uno con sus ventajas e inconvenientes. Según el método de validación empleado se determinan una gran cantidad de características clave de la red *blockchain*, como su escalabilidad, la velocidad de las transacciones, la seguridad o la cantidad de recursos que se consumen al añadir nuevos bloques a la cadena.

Estos algoritmos deben tener poder de superar anomalías en el funcionamiento teórico del sistema como pueden ser el fallo de alguno de los nodos, retrasos en los mensajes, mensajes erróneos, así como nodos maliciosos, irresponsables o no fidedignos.

Varios autores [21] clasifican los métodos de consenso en dos grandes grupos, los basados en la aleatoriedad y los basados en los votos, pero los más complejos tienen características de ambos grupos. El algoritmo PoW se incluye en el grupo de los basados en la aleatoriedad, se debe resolver un puzzle criptográfico del que no se sabe la solución, y cada nodo empieza por donde considere oportuno. En la figura 2.2 se puede ver una comparativa de las principales características de estos dos grupos de consensos en redes privadas comparados con el algoritmo PoW en una red pública.

Los métodos que se basan en la aleatoriedad tienen facilidad para ser escalables, pueden aumentarse el número de transacciones y de nodos, y esto no va a producir una repercusión relevante en los tiempos de latencia o en la seguridad. Pero estos métodos dan lugar a varias ramas de la cadena, y para determinar que rama es la que se acepta finalmente en la cadena y se consolida en todos los nodos, se repercute en la velocidad de las transacciones [1].

[1] RSA es un sistema criptográfico de clave pública que es válido tanto para cifrar y como para firmar digitalmente. Es un método seguro mientras no se conozcan formas rápidas de descomponer un número grande en un producto de números primos. [19]

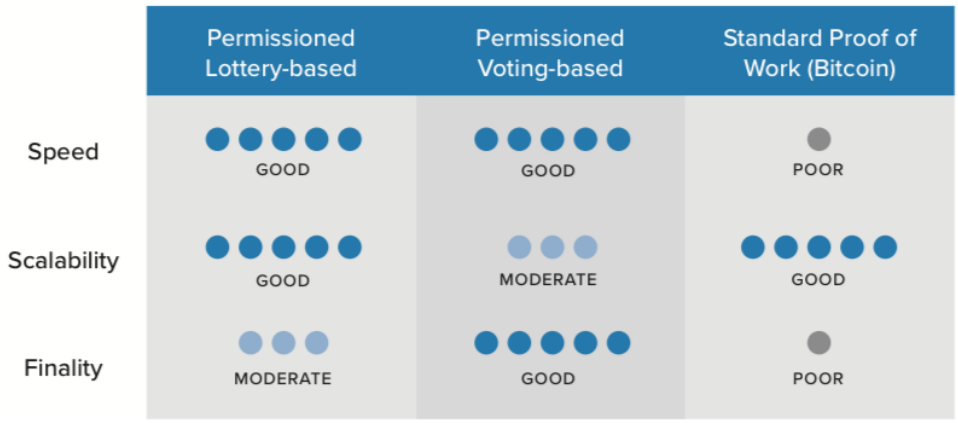


Figura 2.2: Resumen de las principales características de las estrategias de consenso.

Por el contrario, los métodos basados en los votos tienen una gran velocidad para alcanzar la consolidación del bloque en la cadena, pero tienen problemas para escalar su tamaño, pues se deben compartir mensajes entre todos los nodos y puede tener que repetirse varias veces la ronda de votaciones hasta que se alcanza un consenso [1].

Por lo tanto, la decisión entre los métodos de uno y otro grupo se basa en un compromiso entre la escalabilidad y la velocidad de las transacciones [21].

Existen numerosos algoritmos que se basan en la aleatoriedad y en el consenso de la mayoría de votos, en los apartados 2.3.1 a 2.3.4 [22] se describirán los más empleados en la actualidad, y en el apartado 2.3.5 se muestran varias alternativas que están en proceso de investigación.

2.3.1 Prueba de trabajo (PoW)

En este algoritmo la responsabilidad de incluir los bloques en la cadena recae sobre unos nodos denominados "mineros", que deben resolver un problema matemático que requiere una gran cantidad de poder computacional para ganar el derecho a incluir el bloque en la cadena. De una manera resumida, este puzzle criptográfico es el proceso de hacer *hash* de la cabecera del bloque de manera repetitiva, cambiando un parámetro conocido como *nonce*, hasta que el hash resultante coincida con un objetivo específico [23].

La única manera de producir un *hash* resultante que coincida con un objetivo específico es intentarlo una y otra vez, modificando aleatoriamente la entrada hasta que el que se desea aparezca por casualidad. Es decir, es un algoritmo de fuerza bruta.

El tiempo promedio que se tarda en resolver el problema matemático es siempre fijo e independiente del poder computacional de la red (10 minutos en el caso del *Bitcoin*), y se ajusta aumentando o reduciendo el número de ceros del *hash* de la cabecera del bloque.

Cuando un "minero" encuentra el *nonce* que da solución al problema, lo envía al resto de "mineros" para que puedan comprobar que la solución es válida. Una vez comprobado, el "minero" que encuentra el bloque añade una transacción sin entrada y con salida su propia dirección, que será la recompensa por haber encontrado la solución y haber minado el bloque, y lo añade a la cadena.

El principal problema de este algoritmo reside en su escalabilidad, pues mientras más crece la red, mayor es el número de "mineros" que tratan de resolver el problema matemático, lo que aumenta el

poder computacional de la red y por lo tanto, aumenta la dificultad de resolución del acertijo y como consecuencia el consumo de energía. Otro problema derivado de este algoritmo viene en la probable centralización de los "mineros" en las zonas geográficas con un menor coste de la electricidad, lo que elimina en parte la descentralización que se pretende obtener con la tecnología de cadena de bloques.

2.3.2 Prueba de participación (PoS)

Es un algoritmo que se ha creado para mejorar los principales problemas del algoritmo de PoW (2.3.1). Es un concepto en el que el creador del nuevo bloque se selecciona de manera determinista y dependiendo de la cantidad de monedas que posea.

Es un proceso aleatorio en el cual se tienen las mismas probabilidades de minar un nuevo bloque que porcentaje de monedas con respecto al total tenga el nodo en su billetera.

Al contrario que en el algoritmo de prueba de trabajo, donde el minero quiere participar en beneficio de la red, para que la energía empleada en minar el bloque no sea desperdiciada, ya que el resto de "mineros" descartarían el bloque minado si no fuese válido, en este algoritmo los validadores tienen que bloquear sus fondos para participar en el proceso de minado, y si el resto de "mineros" determinan que el bloque minado no es válido, se pierden [23].

Tiene el problema de que si unos pocos nodos tienen la mayoría de las monedas, únicamente estos nodos minarán nuevos bloques y podrían llegar a controlar la mayor parte del sistema, por tanto, la descentralización total sería imposible.

2.3.3 Prueba de tiempo transcurrido (PoET)

El algoritmo de prueba de tiempo transcurrido es una forma eficiente de prueba de trabajo en la que se elimina la necesidad de un uso intensivo de capacidad de computación [24]. Se usa principalmente en las redes de cadenas de bloques privadas. Estas redes deben decidir los derechos "mineros", es decir, quién puede minar nuevos bloques.

Utiliza un algoritmo particular para garantizar la transparencia de la red diseñado por Intel, en el cual a cada participante se le asigna un tiempo aleatorio firmado por una fuente de confianza. Tras la espera del tiempo asignado el usuario tendrá la certificación para completar la labor de creación de un bloque.

La velocidad de este sistema es la mayor de todos, pero puede sufrir ataques a nivel del procesador (se han descubierto ataques de este tipo en los procesadores Intel, por ejemplo, la vulnerabilidad *Spectre*), lo cual puede ser muy difícil de detectar al ser una manipulación de la información al nivel más bajo posible [24].

2.3.4 Prueba con tolerancia a faltas bizantinas (PBFT)

En un sistema distribuido puede haber una gran cantidad de cosas que pueden fallar, los ordenadores pueden fallar, los procesos se pueden bloquear, las conexiones pueden ser interrumpidas, etc., además, como cada usuario tiene sus propios intereses, estos pueden actuar de forma malintencionada [25].

Este algoritmo asume desde el primer momento que puede haber fallos en la red y que los nodos pueden tener errores en algún momento. Está diseñado para sistemas de consenso asíncronos y optimizado de forma eficiente para enfrentarse a las latencias de las telecomunicaciones [22].

Se selecciona un nodo que tendrá el rol de líder, y otros como plan de respaldo. Todos los nodos se comunican entre sí para verificar la información. Los nodos llegan a un acuerdo mediante un proceso de votación y se realiza lo que indica la mayoría de votos.

Este método implica varias ventajas, en primer lugar elimina la necesidad de confirmación, si los nodos concuerdan con un bloque específico, este se finaliza, lo que aumenta la velocidad de las transacciones. La segunda ventaja es la reducción de la energía consumida al no necesitarse un gran consumo de recursos.

2.3.5 Otros algoritmos de consenso

Se han propuesto varias alternativas a estos métodos entre las que se incluyen:

- **Fragmentación:** Esta técnica emplea un pequeño grupo diferente de entre todos los nodos validadores para verificar cada transacción. Este método permitiría la verificación en paralelo de varias transacciones y mejoraría la velocidad [26].
- **Cadenas laterales:** Es una técnica pensada para cadenas de bloques en las que los datos a transmitir tengan un gran tamaño. Para no saturar la cadena principal se crean cadenas secundarias con estos datos de gran tamaño, y en la cadena principal únicamente se almacena la prueba que verifica que la cadena secundaria es correcta. La cadena principal actúa como una capa de control [27].
- **Canales de pago:** Otra solución propuesta es la realización de una capa donde se crean los canales de pago fuera de la cadena de bloques. Los canales de pago requieren múltiples firmas para permitir a las partes transferir entre ellas por un periodo preestablecido de tiempo. Como consecuencia, la mayoría de transacciones se realizan fuera de la cadena de bloques, y únicamente las transacciones con desconocidos o nodos que no cooperan se realizarían a través de la cadena de bloques [28].

La minimización de los recursos computacionales y de la energía empleada es uno de los criterios más significativos a la hora de evaluar el rendimiento de una cadena de bloques, además del tiempo empleado en hacer definitiva, y por lo tanto, inmutable, una transacción.

Estrategias con un alto coste en recursos o energía son inevitables para cadenas de bloques públicas, pero en las privadas los métodos empleados en las cadenas públicas pueden resultar redundantes y un desperdicio tanto de energía como de recursos a la vez que se repercute en la velocidad de operación de la cadena, teniendo en cuenta que todos los usuarios están identificados y en caso de haber algún tipo de error en la cadena de bloques se va a saber en todo momento quién ha introducido ese error.

2.4 Arquitectura *blockchain* para una plataforma de intercambios de energía en una microrred

Para conseguir el mejor rendimiento posible en la plataforma de intercambios de energía que se pretende diseñar en el presente trabajo fin de máster, se comparan las arquitecturas de cadena de bloques más populares en la actualidad, comprobando cuál de ellas se ajusta más a los requerimientos.

Debe conseguirse un equilibrio entre la garantía de privacidad en los datos de los usuarios, que no quieren que sus datos estén expuestos de manera pública, y transparencia, garantizando que todos los intercambios quedan registrados de manera veraz en la cadena y van a poder ser comprobados de manera posterior por cualquiera de las partes interesadas.

Además, se deben conseguir unos tiempos de latencia suficientemente bajos como para hacer posible el correcto funcionamiento del sistema.

Como última condición, la arquitectura seleccionada tiene que incluir la posibilidad de ejecutar dentro de la cadena de bloques contratos inteligentes, para poder automatizar procesos, disminuir las posibilidades de manipulación durante el procesamiento del código y aumentar la velocidad de funcionamiento del sistema en general.

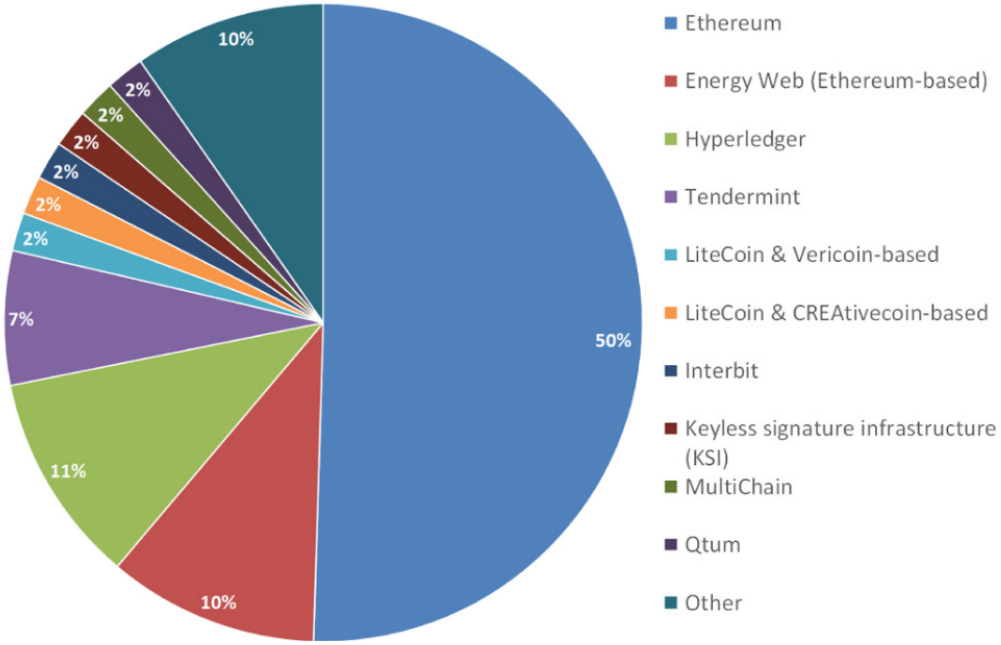


Figura 2.3: Encuesta sobre la plataforma empleada en 140 proyectos del sector de la energía en compañías y proyectos de investigación.

Teniendo en cuenta el amplio abanico de arquitecturas disponibles en la actualidad, se parte de los resultados del estudio realizado por [1], que ha tenido en cuenta 140 proyectos de aplicación de la tecnología de cadena de bloques en el sector de la energía (figura 2.3), al presuponer que mientras más elevado sea el porcentaje de proyectos relacionados haya en dichas plataformas, más sencillo será encontrar soporte en problemas similares. En este estudio se puede observar, que la plataforma en la que más proyectos se han apoyado es *Ethereum* de forma muy destacada, con un 60 % del total, seguida muy de lejos por *Hyperledger* con un 11 %.

Se decide hacer una comparativa entre estas dos plataformas para determinar cuál es la óptima para el desarrollo de este proyecto.

2.4.1 *Ethereum* vs *Hyperledger*

Para realizar la comparativa se obvian las similitudes y se enfrentan principales diferencias entre ambas plataformas con el objetivo de entender cuál resulta más adecuada para una plataforma de intercambios de energía en una microrred [29]:

- **Objetivo:**
  - *Ethereum* ha sido creado con el único propósito de ejecutar contratos inteligentes dentro de dicha cadena para crear aplicaciones descentralizadas y el consumo de masas. Las transacciones son transparentes para cualquier usuario en la red.

  - *Hyperledger* se ha diseñado con el objetivo de llevar la tecnología de cadena de bloques a las empresas al mejorar la confidencialidad y la escalabilidad. Se puede seleccionar qué partes de las transacciones son visibles y para quién. Es un sistema permisionado.

- **Participación:**
  - *Ethereum* está basado en una red completamente abierta donde cualquiera puede unirse.
  - En *Hyperledger* se requiere ser un usuario autorizado poder participar.

- **Lenguaje de programación:**
  - En *Ethereum* los contratos inteligentes se escriben en un lenguaje específico orientado a contratos que se llama *Solidity*.
  - En el caso de *Hyperledger* los contratos inteligentes se escriben principalmente en *Javascript*, *Java* o *Golang*.

- **Algoritmo de consenso:**
  - En *Ethereum* se emplea el algoritmo de prueba de trabajo (2.3.1), y tiene la particularidad de que hay que pagar una cuota a los "mineros" por la capacidad de computación que se emplee para ejecutar los contratos inteligentes.
  - En *Hyperledger* el algoritmo de consenso es configurable dependiendo del objetivo para el que esté diseñada la cadena de bloques en concreto, estando disponibles actualmente dos posibilidades, no emplear ningún mecanismo de consenso o PBFT (2.3.4), explicado en detalle en la sección 2.3.

Características	Ethereum	Hyperledger
Esencia	Plataforma de propósito general sin permisos	Plataforma modular permisionada
Lenguaje de contratos inteligentes	Solidity	JavaScript, Java, Golang, etc.
Algoritmo de consenso	PoW	PBFT, ninguno u otros
Tipo de red	Pública	Privada
Escalabilidad	Difícil de escalar	Fácil de escalar
Token	Ether	El que se diseñe para la aplicación concreta

Tabla 2.1: Resumen de las principales características de *Ethereum* e *Hyperledger* .

Ambas son plataformas de código abierto y tienen en común un fuerte apoyo por parte de desarrolladores. *Ethereum* está siendo apoyada fuertemente por una gran comunidad de desarrolladores de código libre, debido a la transparencia que consigue brindar, mientras que *Hyperledger* tiene un fuerte apoyo por parte de más de 170 organizaciones entre las que destacan la fundación Linux, IBM e Intel, por lo que las dos arquitecturas para el desarrollo de cadenas de bloques van a tener continuidad en el tiempo, y una aplicación desarrollada en cualquiera de ellas no va a quedarse obsoleta en el corto o medio plazo.

2.4.2 Decisión final

Tras valorar todas las diferencias entre ambas plataformas se opta por desarrollar la aplicación para los intercambios de energía dentro de una microrred eléctrica en la plataforma de *Hyperledger Fabric* (2.5),

mediante el conjunto de herramientas *Hyperledger Composer* [2] (2.5.6), habiendo tenido especial peso en la decisión:

- **Red permisionada:** Obliga a que los usuarios sean dados de alta en el sistema para poder conectarse a la red, gracias a esto se tiene a todos los usuarios identificados.
- **Gestión de los permisos:** Se pueden gestionar los permisos de lectura y escritura de todos los elementos de la cadena de bloques (creación de usuarios, creación y transmisión de activos, etc.)
- **Algoritmo de consenso:** No es obligatorio utilizar un consenso de prueba de trabajo (2.3.1) como en *Ethereum*[3], que sería innecesario en un entorno en el que se controla quién participa, lo que implica menores tiempos de latencia en las transacciones y un menor consumo de recursos.
- **Facilidad de uso:** *Hyperledger Composer* (2.5.6) es un conjunto de herramientas especialmente diseñadas para hacer más sencilla el desarrollo de aplicaciones de cadenas de bloques y los programas para interactuar con ellas.

2.5 *Hyperledger Fabric*

La fundación Linux creó en el año 2015 el proyecto *Hyperledger* para tratar de extender el uso de la tecnología de cadena de bloques en los entornos empresariales. En lugar de crear un único estándar lanzó el proyecto en forma de código abierto, para que la comunidad aportara y desarrollara tal y como fuera necesitándolo [18].

Hyperledger Fabric es uno de los proyectos de *blockchain* nacidos del proyecto madre *Hyperledger*. Las principales características esta tecnología de red de cadena de bloques son, ser un libro de cuentas distribuido, emplear contratos inteligentes y ser un sistema en el que los participantes dan la orden de realizar las transacciones [18]. Además, implementa una arquitectura modular, que permite a los diseñadores elegir, entre otras características, el algoritmo de identificación de usuarios, la técnica empleada para la sincronización de los nodos, el método de encriptación y el tipo de base de datos.

Se diferencia de todas las arquitecturas de cadena de bloques previas, en el control de la privacidad que ofrece sobre los datos que contiene. Las redes de *Hyperledger Fabric* se pueden configurar como redes privadas, en las que se necesita autorización para poder conectarse. Esto repercute, además, en la velocidad de las transacciones, pues en lugar de tener que emplear un protocolo como la prueba de trabajo (2.3.1) para validar las transacciones y garantizar la seguridad, los miembros se unen a la red mediante un proveedor de servicios de membresía (apartado 2.5.2) (MSP por sus siglas en inglés), que comprueba si el usuario está autorizado a unirse, lo que agiliza el proceso de aceptación de cambios en la red.

2.5.1 Verificación de la identidad de los usuarios en *Hyperledger Fabric*

Una infraestructura de clave pública (PKI por sus siglas en inglés) es un conjunto de tecnologías que proveen comunicaciones seguras a través de una red [30]. Conceptualmente, una PKI está compuesta por cuatro elementos fundamentales (representados gráficamente en la figura 2.4):

[2] En septiembre de 2019 se ha presentado la versión 1.4 de *Hyperledger Fabric*, en la que es mucho más sencillo desarrollar aplicaciones, pues han incluido las herramientas de *Hyperledger Composer*, por lo tanto, ya no se recomienda su uso por no aportar ninguna ventaja adicional

[3] Está previsto que a lo largo del año 2020 *Ethereum* cambie su algoritmo de consenso a prueba de participación (2.3.2).

- **Certificados Digitales.** (2.5.1.1)
- **Claves pública y privada.** (2.5.1.2)
- **Autoridad certificadora.** (2.5.1.3)
- **Lista de certificados rechazados.** (2.5.1.4)

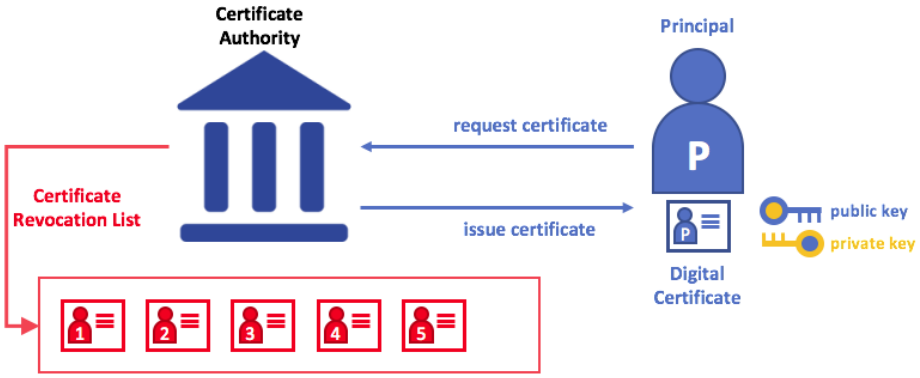


Figura 2.4: Elementos que intervienen en el proceso de identificación de usuarios mediante una infraestructura de clave pública.

2.5.1.1 Certificados Digitales

Un certificado digital es un documento digital que contiene un conjunto de atributos relativos a su dueño. Es similar al documento nacional de identificación emitido por los gobiernos de los países.

En *Hyperledger Fabric* se emplea un certificado digital siguiendo el estándar X.509 que contiene, entre otros datos, las claves pública y privada del usuario al que identifica, una identidad y un algoritmo de validación de la ruta de certificación. Se define empleando el lenguaje ASN.1. [31]

2.5.1.2 Claves pública y privada

La garantía de la identidad de los usuarios y la integridad del mensaje son las partes fundamentales en la seguridad de las comunicaciones. Para que se pueda asegurar que el emisor o el receptor del mensaje es el usuario destinado para ello, se emplean las firmas digitales.

Mediante la clave privada almacenada en el certificado digital del usuario emisor, se firma el mensaje a transmitir que, mediante criptografía, crea una cadena de texto de longitud determinada que identifica al mensaje. Cualquier variación en el contenido del mensaje variará esta cadena de texto. El receptor empleará la clave pública del emisor del mensaje para desencriptarlo. Si el mensaje ha sido modificado o la firma no es la correcta el mensaje no se desencriptaría, por lo tanto, queda asegurada la identidad del emisor y la integridad del mensaje.

Las claves públicas y privadas también se pueden emplear de manera inversa, para garantizar que el receptor es únicamente el usuario para el que está destinado el mensaje. Para ello, el emisor firma el mensaje con la clave pública del receptor, y solo podrá ser desencriptado mediante la clave privada del destinatario.

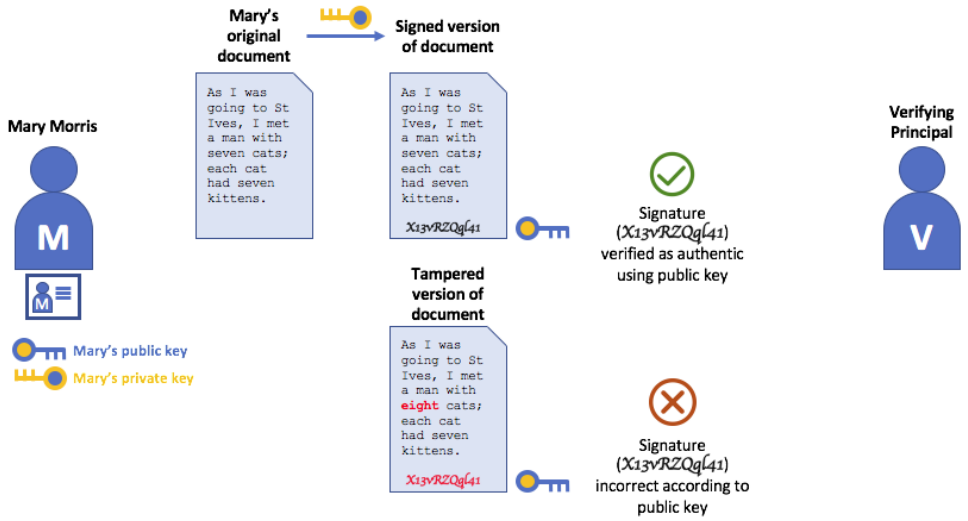


Figura 2.5: Representación gráfica del proceso de firma de un documento mediante la clave privada, y verificación de la identidad del emisor e integridad del mensaje mediante la clave pública.

2.5.1.3 Autoridad certificadora

Cada nodo y cada usuario en *Hyperledger Fabric* tiene un certificado digital que contiene sus claves públicas y privadas. Para garantizar que estos certificados digitales son válidos, se emiten mediante una autoridad certificadora firmándolos con su clave privada (figura 2.6), por lo tanto, si se confía en la autoridad certificadora y se conoce su clave pública, se puede confiar en la autenticidad del certificado digital del usuario.

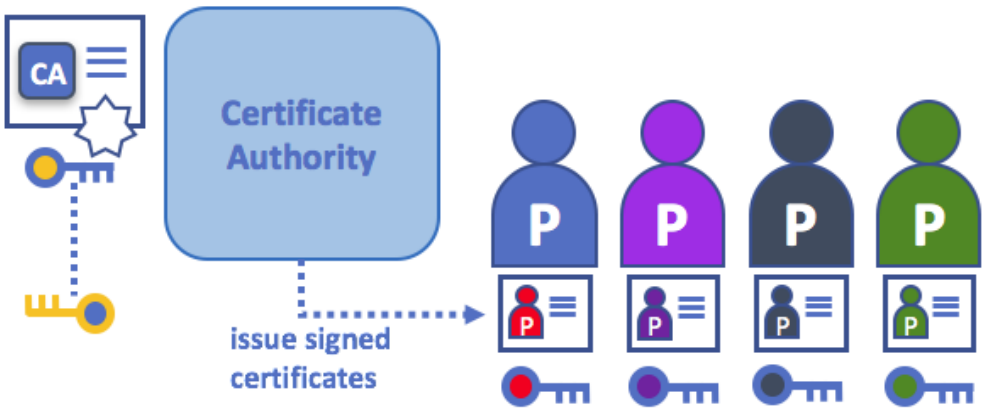


Figura 2.6: Proceso de creación de certificados digitales por parte de una autoridad certificadora.

En *Hyperledger Fabric* se pueden crear tantos nodos de autoridad certificadora como se desee, y cada usuario de la red puede confiar en el que crea conveniente.

2.5.1.4 Lista de certificados rechazados

La lista de certificados rechazados es el conjunto de certificados que la autoridad certificadora sabe de antemano que no son confiables, por lo tanto, son rechazados automáticamente a pesar de tener una firma válida.

Los certificados rechazados son totalmente válidos, pero por algún motivo se les ha revocado la autorización (un robo, mal uso de la identidad, etc.).

2.5.2 Comprobación de la membresía en una red de *Hyperledger Fabric*

Una vez analizado el proceso que garantiza la identidad de los usuarios (2.5.1), cada red *blockchain* tiene una lista con las identidades autorizadas a registrarse en ella.

En esta lista, además, se indica con qué permisos se puede unir a la red (lectura, escritura, modificación o eliminación) para cada uno de los elementos que componen la red (usuarios, activos, transacciones o eventos).

Hyperledger Fabric permite una característica que no tienen el resto de redes de cadenas de bloques, la creación de canales dentro de la red. Estos canales tienen una lista de identidades autorizadas propia. De esta forma, se puede crear una red *blockchain* en la que los usuarios que estén autorizados a unirse no tengan permiso para unirse a algún canal, y para cada uno de los canales en los que es miembro, puede tener diferentes permisos de interacción con los datos, de esta forma se consiguen aislar los datos de los usuarios que no deben conocerlos, aumentando la privacidad de estos.

2.5.3 Algoritmos de consenso en *Hyperledger Fabric*

Hyperledger Fabric ha sido diseñado para permitir a los desarrolladores de redes de cadena de bloques seleccionar el algoritmo de consenso que más les convenga en función de sus necesidades, si quieren una red muy estructurada jerárquicamente o una más descentralizada.

En septiembre de 2019, soporta tres mecanismos de consenso diferentes, que pueden cambiarse de manera sencilla a la hora de desarrollar la red de cadena de bloques. Al nodo que ejecuta el algoritmo de consenso se le denomina ordenante, y se puede configurar para que esta ejecución sea mediante *SOLO, Kafka o Raft.*

SOLO es el mecanismo más simple, únicamente transmite la transacción sin aplicar ningún consenso real [32]. Se elige un nodo ordenante como líder y el resto aceptan el bloque propuesto por este. Si hubiera algún fallo en el nodo líder, se transmitiría un bloque erróneo a todos los nodos. Este algoritmo no está recomendado para aplicaciones finales, pero está disponible para realizar pruebas de concepto.

Tanto *Kafka* como *Raft* son algoritmos de prueba con tolerancia a fallos (CFT por sus siglas en inglés), es decir, tiene un nivel de resiliencia frente a fallos de los dispositivos (fallo físico del nodo, caída del proceso, ...) o de las comunicaciones (pérdida de la conexión, latencias excesivas, ...). Están basados en el algoritmo de prueba con tolerancia a faltas bizantinas (2.3.4) pero son más sencillos, ya que presuponen que no hay nodos malintencionados al estar todos los participantes de la red identificados [25]. Estos algoritmos se basan en elegir a un líder entre los nodos ordenantes, y esperar a que el resto de nodos ordenantes confirmen el bloque propuesto por el líder mediante una votación, si se obtiene una mayoría en el bloque propuesto por el líder, el bloque se replica en todos los nodos de la red. Si falla el nodo ordenante líder, como todos los nodos tienen una copia igual de la cadena de bloques, se elige otro como líder.

Raft, el último algoritmo compatibilizado con *Hyperledger Fabric*, añadido en septiembre de 2019, debido a que los desarrolladores no implementaban *Kafka* porque resultaba intimidante su complicada configuración [33]. Además, supone una mejora con respecto a *Kafka* en cuanto a velocidad, tarda únicamente 50 ms en minar un bloque [34], y en escalabilidad, es más fácil de implementar en redes con

un gran número de nodos [35], por lo tanto, actualmente es el único algoritmo de consenso recomendado para utilizar en proyectos definitivos.

Puede haber más de un nodo ordenante en una red, y para que los algoritmos con tolerancia a fallos tengan sentido, debe haber al menos tres de estos nodos, de forma que en la votación haya una mayoría que no tenga en cuenta un nodo con fallo (si solo hubiera dos nodos ordenantes y uno de ellos presenta un fallo, por lo tanto, cada uno vota una cosa diferente y no se llegaría a un acuerdo válido).

A lo largo del año 2019 o principios del 2020 se espera que se añada un nuevo algoritmo de consenso basado en la prueba de tolerancia a faltas bizantinas, que ya está en un estado de desarrollo avanzado, para añadir un nivel más de seguridad en caso de querer crear una red con usuarios anónimos [35].

2.5.4 Smart Contracts

Los contratos inteligentes (Smart Contracts en inglés) son programas escritos en la cadena de bloques, que pueden ser llamados por una aplicación externa a la cadena cuando se necesite interactuar con la base de datos almacenada en la cadena. Actualmente los contratos inteligentes de *Hyperledger Fabric* pueden ser implementados en *Node Js* o *Go*. En *Hyperledger Fabric* a los contratos inteligentes se les denomina *chaincode*.

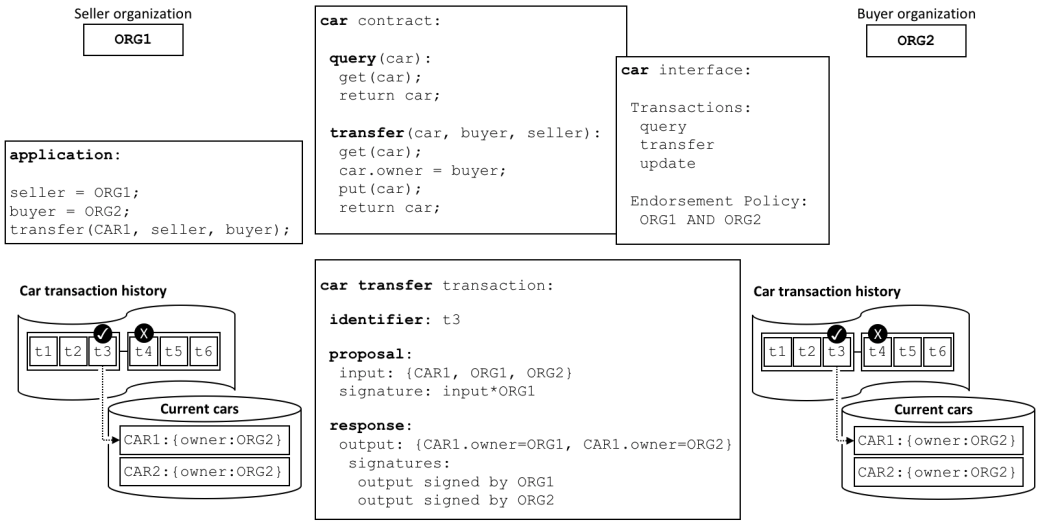


Figura 2.7: Ejemplo de un contrato inteligente empleado en la transferencia de la propiedad de un coche.

El contrato inteligente define las reglas entre los diferentes usuarios de la red mediante código ejecutable. Las aplicaciones externas ejecutan los contratos inteligentes para realizar transacciones, que son añadidas a la base de datos distribuida [36].

En la figura 2.7, se puede ver un ejemplo de contrato inteligente para realizar la transferencia de la propiedad de un coche. En dicha transacción participan dos usuarios, cada uno con su historial de transacciones. Mediante una aplicación externa se indica a la cadena de bloques que el vendedor va a transferir la propiedad del coche al comprador. El contrato inteligente entonces manda una petición a la base de datos para que extraiga de esta los datos del coche y, posteriormente, los envía a una función que cambia el parámetro que indica quién es el propietario del vehículo. Una vez modificado este parámetro, el coche se guarda en la base de datos y se añade la transacción, con la firma de ambos participantes, también a la base de datos, donde ya no puede ser modificada o borrada.

2.5.5 Flujo de las transacciones

El flujo de acciones que se suceden desde que un cliente quiere realizar una interacción con la cadena de bloques y finalmente se incluye como un bloque en la base de datos se puede resumir, en *Hyperledger Faric*, en los siguientes pasos:

1. El usuario envía, a través de su aplicación de interacción con la cadena, una propuesta firmada a todos los nodos.
2. Los nodos firman esa propuesta y la envían a la autoridad certificadora.
3. La autoridad certificadora comprueba que las firmas son válidas, tanto la del usuario que solicita la transacción como las de los nodos, y comprueba si el usuario tiene autorización para realizar la transacción que desea en esa cadena de bloques.
4. Si todas las firmas han sido válidas, se ejecuta el contrato inteligente.
5. La aplicación verifica la firma de la autoridad certificadora y comprueba que todas las propuestas de todos los nodos son iguales. Se envía la propuesta al ordenante. Si la transacción es únicamente una petición de lectura a la base de datos, no se pasa al ordenante.
6. El ordenante coloca todas las transacciones que ha recibido en un único bloque, ordenadas de manera cronológica y se la envía a los nodos.
7. Los nodos comprueban los bloques recibidos, comprueban que no ha habido modificaciones en la cadena con respecto al estado previo a que se iniciara la transacción, y los añaden a la cadena si son válidos.

2.5.6 *Hyperledger Composer*

Para facilitar el desarrollo de redes de cadenas de bloques en *Hyperledger Fabric* y los contratos inteligentes que interactúan con ellas (2.5), la fundación Linux ha ideado el conjunto de herramientas de *Hyperledger Composer*, que permite acelerar los procesos de creación y prueba tanto de las redes de cadenas de bloques como de los contratos inteligentes [37].

Gracias a las herramientas proporcionadas por *Hyperledger Composer*, se puede diseñar de manera rápida una aplicación de cadena de bloques que contenga los usuarios, activos y transacciones necesarias para la aplicación concreta, programando directamente en el navegador web y pudiendo instalar los cambios con solo dar a un botón para poder probarlos en ese mismo navegador. Los activos pueden ser bienes tangibles o intangibles, servicios o propiedades.

Está compuesto por las siguientes herramientas [38]:

- Un lenguaje de modelado propio, al que denominan *CTO*, que hace muy sencilla la definición del modelo de cadena de bloques que necesita el desarrollador.
- Una interfaz de usuario llamada *Playground* (figura 2.8), que permite el desarrollo de código, visualización de las pruebas y el despliegue rápido de redes empresariales. En esta interfaz se pueden consultar de manera inmediata qué activos se intercambian, quién participa en esos intercambios, se pueden controlar los accesos de visualización y modificación de los datos incluidos en la cadena, y se puede desarrollar la lógica de transacciones.

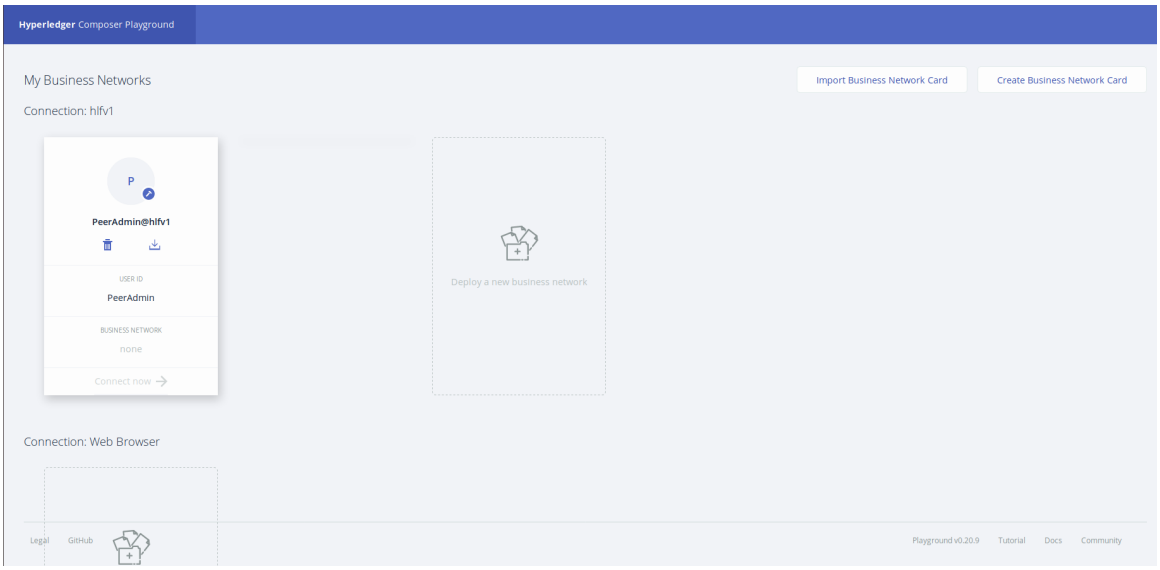


Figura 2.8: Pantalla de inicio de la herramienta *Playground*

- Herramientas de la interfaz de la línea de comandos para integrar aplicaciones modeladas con *Hyperledger Composer* con los nodos de la red *Hyperledger Fabric* (2.5) que están ejecutándose.

Hyperledger Composer permite, en definitiva, desarrollar una cadena de bloques al más alto nivel, haciendo transparente para el desarrollador la programación de bajo nivel. El desarrollador de la aplicación de cadena de bloques debe tener clara la estructura, y a partir de ahí es sencillo de modelar mediante el lenguaje *CTO*.

El programador debe conocer antes de comenzar a desarrollar código los siguientes aspectos clave de la aplicación de cadena de bloques:

- Qué usuarios van a participar.
- Qué datos se deben almacenar.
- Qué activos se van a intercambiar.
- Cómo se va a realizar el intercambio.
- Qué usuarios tienen permiso para visualizar cada dato.
- Qué usuarios tienen permiso para ejecutar cada contrato inteligente.

Playground permite su ejecución de dos modos diferentes, el primero de ellos es el modo "sólo navegador", que permite crear modelos y probar la red en una base de datos almacenada de forma local, que únicamente es útil para hacer las primeras pruebas y comprobar si el modelo es válido. El segundo modo es instalando *Playground* junto con *Hyperledger Fabric* (2.5) en el ordenador, para que los nodos validen todos los cambios en la base de datos de la cadena de bloques.

Ejecutándose en cualquiera de los dos modos, se deben crear los siguientes archivos para desarrollar la aplicación completa:

1. **README.md:** Este archivo contiene la descripción de la aplicación y la información que quiera añadir el desarrollador.

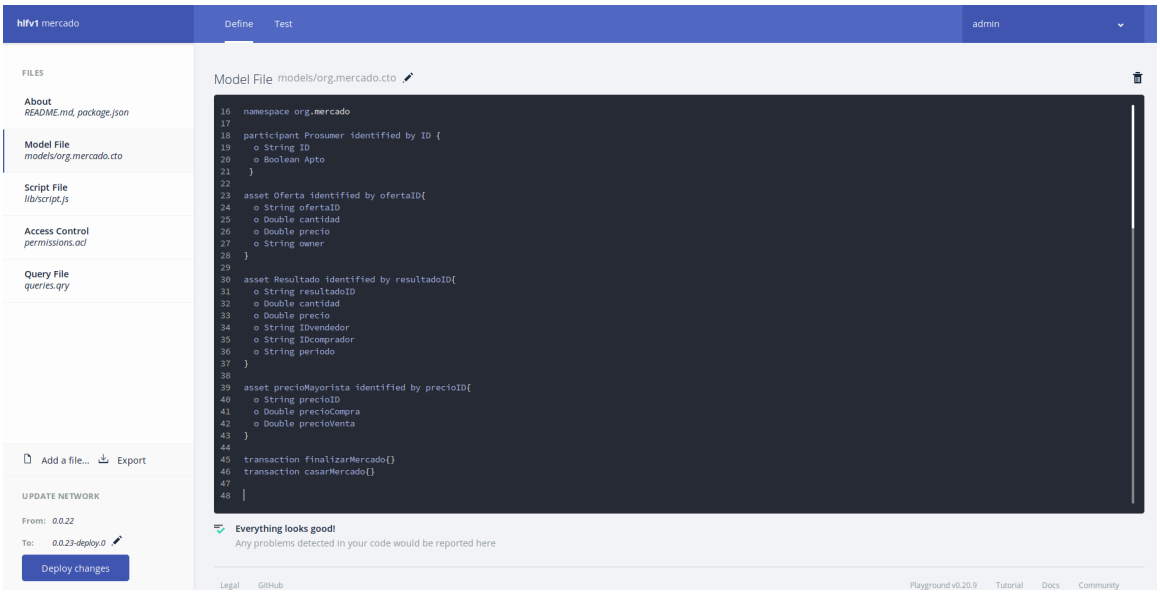


Figura 2.9: Entorno de programación de la herramienta *Playground*

2. **package.json:** Se crea automáticamente y contiene la información de las versiones de la aplicación y de los módulos a los que llama para ejecutar la aplicación.

3. **archivo.cto:** Contiene el modelo de la aplicación, en él se definen los usuarios, los activos, las transacciones y los eventos. Se escribe en lenguaje *CTO*, que es un lenguaje propio de *Hyperledger Composer* orientado a objetos.

4. **script.js:** En este archivo se desarrolla la lógica de las transacciones. Está escrito en *node js*. Es un contrato inteligente o *SmartContract*. En él se implementan las transacciones que se van a realizar en la aplicación.

5. **permissions.acl:** Contiene las reglas que permiten controlar el acceso a todos los recursos de la red.

6. **queries.qry:** Es el archivo en el que se definen las llamadas a la base de datos de *Hyperledger Fabric* que se quieran realizar. Está escrito en un lenguaje propio. Se le pueden añadir diversos parámetros para obtener una respuesta que se ajuste lo máximo posible a lo que se necesite.

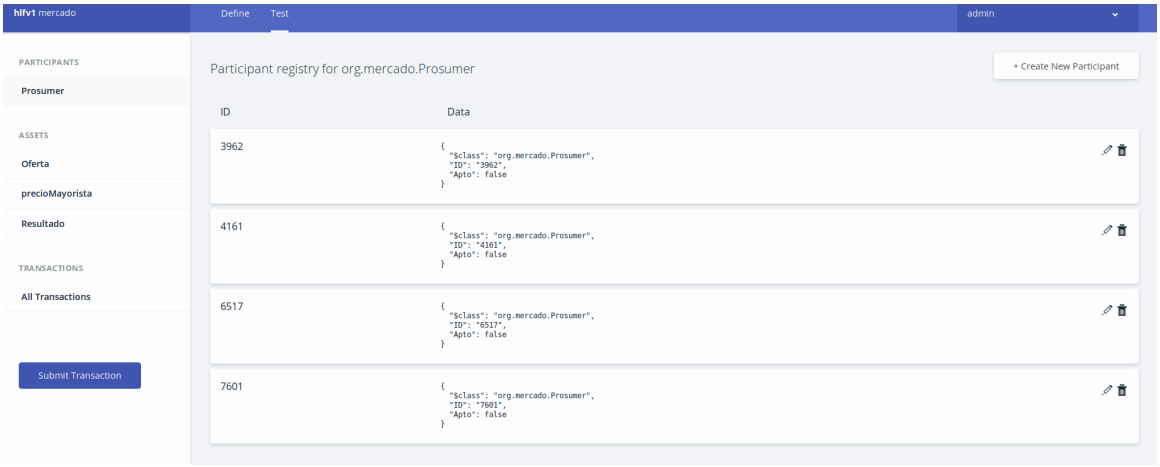


Figura 2.10: Pantalla de pruebas de la herramienta *Playground*

Se puede ver la interfaz de la herramienta *Playground* en la figura 2.9, donde se puede observar, en la parte izquierda, una lista con los archivos que se han desarrollado hasta el momento, y en la parte de la derecha de la imagen se encuentra el código del archivo seleccionado, que puede ser editado directamente en el navegador web.

Una vez terminados de desarrollar todos los archivos descritos, se puede probar la aplicación directamente en la herramienta *Playground*. En la figura 2.10, se tiene captura de pantalla de este apartado de la herramienta, en la cual, en la parte izquierda se encuentra un desglose de todos los activos y todas las transacciones que se puedan realizar en la aplicación de cadena de bloques, así como un botón para realizar nuevas transacciones de prueba. En la parte derecha de la imagen se puede interactuar con todos los activos de la cadena de bloques (creando nuevos, editándolos, eliminándolos, etc.)

La herramienta permite, además, la creación de perfiles diferentes al del administrador del sistema para realizar pruebas con varios usuarios y comprobar el correcto funcionamiento de los permisos de visualización, creación o edición de los activos.

Capítulo 3

Estructura del sistema

Tomando como referencia el artículo de investigación de Zhiyi Li [39], en el que se propone de manera conceptual el uso de la tecnología de cadena de bloques para creación de un mercado local en el que se realice la gestión de los intercambios de energía entre diferentes microrredes, se aplica el mismo concepto a los intercambios de energía entre los prosumidores dentro de una única microrred. Se parte de una estructura que se compone por cuatro cadenas de bloques *blockchain*, en las que se almacena información de diferente índole y con permisos de acceso y modificación de los datos diferentes, para garantizar la privacidad de todos los usuarios de la red a la vez que se conocen los datos necesarios para un funcionamiento correcto y se hace posible la revisión de todos los parámetros del sistema e intercambios realizados en el momento que resulte necesario para garantizar su veracidad.

En el presente capítulo se van a analizar las cuatro cadenas de bloques desarrolladas, qué datos almacena cada una de ellas, qué contratos inteligentes contienen y quién puede ejecutarlos, cómo los usuarios interaccionan con ellas y qué datos puede consultar cada miembro de la microrred.

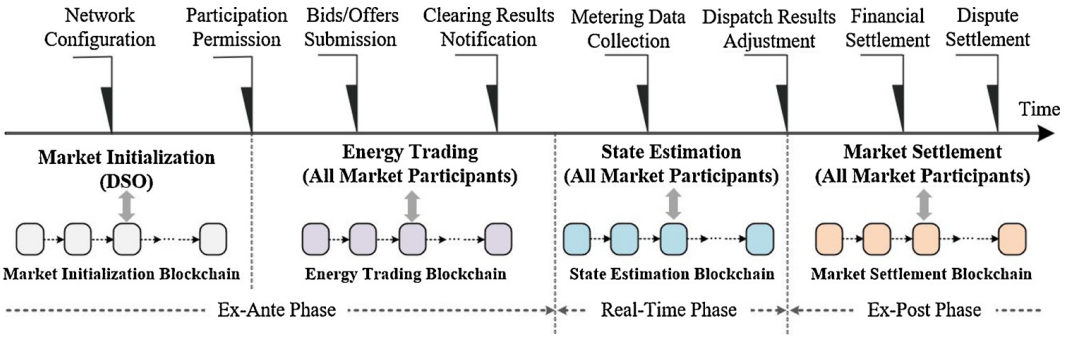


Figura 3.1: Representación gráfica de las cuatro cadenas de bloques que componen el sistema.

Estas cuatro cadenas de bloques serán empleadas en las diferentes fases temporales del mercado local interno de la microrred. En la fase previa a la realización del mercado se emplean dos de ellas, la primera con los datos de inicialización del mercado, en ella se podrá comprobar la reputación del usuario y si tiene dinero suficiente como para participar en la subasta de energía con garantías, y la segunda en la cual los prosumidores introducirán sus ofertas de compra o venta de energía. Durante el periodo para el que se ha realizado la subasta de energía, los prosumidores introducen sus datos de los parámetros eléctricos registrados por el contador inteligente en tiempo real en una tercera cadena de bloques, que sólo contendrá los parámetros eléctricos de todos los usuarios. En la cuarta y última cadena de bloques, una vez finalizado el periodo para el que se ha realizado la subasta, se registran todas las transacciones

de energía que se han realizado, registrando en cada una qué usuarios han intervenido, y qué repercusión ha tenido esa transacción económicamente y en su reputación.

En el apartado 3.1 se analizan los tipos de usuarios que tiene el sistema. En el siguiente apartado (3.2), se detalla el funcionamiento de la plataforma al completo, desde la selección de los usuarios que van a participar hasta que se realizan los pagos por los intercambios realizados. En los apartados siguientes, del 3.3 al 3.6 se analizan las cuatro cadenas de bloques en profundidad, cada uno de los datos que almacenan, qué función realizan los contratos inteligentes que se han implementado en su interior, y qué usuarios tienen permisos para leer o escribir datos en ellas y ejecutar sus contratos inteligentes.

3.1 Usuarios del sistema

La plataforma de intercambios de energía va a ser accedida por dos tipos de usuarios diferentes, que van a interaccionar de manera diferente con las cadenas de bloques para conseguir un funcionamiento correcto.

- **Gestor del sistema:** El primer tipo de usuario es el gestor del sistema, que tiene capacidad para dar de alta nuevos usuarios en el sistema, tiene la función de supervisar que todo se realiza de manera correcta, introduce los datos del precio de los intercambios de energía con la red principal y tiene capacidad para modificar los datos personales del resto de usuarios (introduce los resultados de la liquidación de pagos por los intercambios, es decir, variación del número de *tokens* que posee cada usuario y cambio en la reputación).

  Este gestor del sistema, teóricamente, es un ente público sin ánimo de lucro, pero con una sencilla modificación en los contratos inteligentes, cada transacción podría incluir un pequeño porcentaje para pagar sus servicios.

  Todos los usuarios de la microrred conocen la identidad del gestor del sistema, y depositan su confianza en él, pues tiene capacidad para visualizar todos los datos. Si por algún motivo este usuario fuera malintencionado, todas sus acciones quedarían reflejadas en las cadenas de bloques por lo que se podría asignar un culpable.

- **Prosumidor:** El segundo tipo de usuario es el prosumidor (1.2.3), el cliente final que quiere participar en la subasta de energía, ya sea como comprador o vendedor de electricidad. Interacciona con las cadenas de bloques empleando los datos que le proporciona su sistema de gestión de la energía para realizar ofertas de compra o venta de "paquetes de electricidad" en la plataforma de intercambios e introduce los datos de parámetros eléctricos proporcionados por su contador inteligente para dejar constancia de ellos, y que el sistema calcule de forma posterior a la compra o venta de energía si ha introducido o extraído de la red de baja tensión la cantidad de electricidad que se había establecido en la subasta y la calidad de esta energía (frecuencia, voltaje, etc.), para que el gestor del sistema determine los pagos que debe realizar o recibir y si debe haber una repercusión en su reputación.

  Este usuario únicamente tendrá permiso para modificar (introducir ofertas y datos del contador inteligente) y visualizar los datos almacenados en las cadenas que estén relacionados con él, concretamente los resultados de la subasta y los intercambios de energía en los que aparece como parte implicada y sus datos personales, incluyendo el número de *tokens* que posee y su reputación en la plataforma.

3.2 Funcionamiento de la plataforma de intercambios

Se plantea un algoritmo para intentar minimizar los consumos de energía de la red principal y aumentar la independencia con respecto a esta en caso de caída, además de incentivar el consumo de energías renovables mediante el abaratamiento de los precios del consumo de energías procedentes de generadores "verdes" y el retorno de la inversión más rápido para los usuarios que decidan instalar capacidad de generación en sus viviendas. El algoritmo se puede resumir en los siguientes pasos:

1. Todos los usuarios realizan una previsión los consumos y generación de electricidad durante el próximo periodo de mercado, que da lugar a un desajuste, que es la cantidad de energía que debe inyectar o absorber de la red de baja tensión.

2. En función del histórico de participaciones en el mercado a los usuarios se les asigna una reputación, que sirve para filtrar qué usuarios pueden ser admitidos en la plataforma. Además de tener buena reputación, deben tener *tokens* suficientes como para que se garantice que van a pagar las transacciones que correspondan.

3. Tomando como referencia las previsiones realizadas, todos los usuarios de la microrred admitidos en la subasta realizan una oferta de compra o venta de energía. Esta oferta no tiene por qué ser exclusivamente una cantidad de energía, puede contener más parámetros para que tanto el comprador como el vendedor puedan determinar de manera más precisa con quien realizan el intercambio. Entre otros, se plantean los siguientes parámetros adicionales:

   - Fuente de generación de la electricidad.
   - Distancia máxima entre comprador y vendedor.
   - Precio máximo (comprador) o mínimo (vendedor).
   - Una cantidad máxima o mínima de energía.

   Además, los usuarios que tienen capacidad para absorber desajustes (capacidad de almacenamiento o de variación de energía generada o consumida), ofrecen los precios de compra o venta de energía para compensar esos desajustes.

4. Se da a conocer el resultado de la subasta a los usuarios, y estos, pueden variar su estrategia de generación o consumo en consecuencia. Por ejemplo, si se hace una oferta para inyectar a la red una cantidad de energía a un precio, pero nadie la ha comprado, se puede inyectar esa energía en una batería para usarla o intentar venderla en otro momento.

5. Durante el periodo para el que se había realizado la subasta de energía, la infraestructura de medición de parámetros eléctricos de cada usuario registra todos los datos posibles en el sistema, y una vez finalizado el periodo se comprueban. Lo normal es que haya habido desajustes con respecto a los resultados de la subasta al no ser las previsiones exactas al detalle. Estos desajustes no implican una caída del sistema, pues hay usuarios con capacidad de almacenamiento para absorberlos, además, la microrred permanece conectada a la red principal para absorber o inyectar energía en caso de necesitarlo.

6. Los usuarios que tienen un desajuste una vez finalizado el periodo entran en una nueva subasta de energía a posteriori, en la que se casa a los usuarios que han tenido desajustes con los que los han absorbido, estableciéndose un precio mejor que el de la subasta a los usuarios que han absorbido desajustes, que es compensado con un precio peor para los usuarios que han tenido desajustes con respecto a su previsión.

7. Para los usuarios que no han sido casados en la segunda subasta, se crea una transacción con la empresa comercializadora de energía, pues se asigna la cantidad de energía intercambiada entre la red principal y la microrred a estos usuarios.

8. A los usuarios que no habían sido considerados aptos para participar en la subasta de energía también se les registra una transacción por una cantidad de energía igual a la diferencia entre su generación y su consumo con la empresa comercializadora, a los precios públicos establecidos.

3.3 Cadena de bloques de datos personales

En la primera de las cuatro cadenas *blockchain* que componen la plataforma de intercambios de energía, se almacenan los datos personales correspondientes a cada uno de los usuarios prosumidores de la microrred. Estos datos únicamente pueden ser modificados por el gestor del sistema (a petición del usuario prosumidor, por alguna variación en sus datos), y cada usuario tiene permiso para visualizar exclusivamente sus datos personales. En esta cadena se vincula cada prosumidor con un número de identificación de usuario secreto, que será el que se use en el resto de cadenas del sistema, de esta forma la identidad de los prosumidores se mantiene oculta al resto de usuarios de la red, y a no ser que haya algún problema no se debería dar a conocer.

Para el correcto funcionamiento del sistema, se crean unos *tokens* que son intercambiables por dinero real, es decir, cada prosumidor compra una cierta cantidad de *tokens* pagando dinero real al gestor del sistema, y se emplean como moneda de cambio para comprar electricidad dentro del sistema de intercambios de energía, haciendo posibles los pagos instantáneos entre prosumidores.

Los usuarios que tienen una producción de energía superior al consumo durante determinados periodos del día, pueden vender su energía y recibir *tokens* a cambio, que podrán usar para comprar energía cuando su producción sea inferior al consumo. Si un prosumidor habitualmente genera más energía de la que consume, tendrá un exceso de *tokens* que podrán ser vendidos de nuevo al gestor del sistema.

Además, para ser admitidos en el mercado local deben tener un número de *tokens* suficiente como para pagar el consumo medio durante un cierto intervalo de tiempo que se establezca, para garantizar que se pueden pagar los consumos de electricidad o las sanciones que se produzcan durante ese periodo.

En esta cadena se almacena, además de los datos personales del prosumidor, la información de la reputación del usuario en el sistema y la cantidad de *tokens* que tiene disponibles para pujar por energía en el mercado local.

Cuando finaliza cada ronda de mercado, el gestor del sistema modifica los *tokens* y la reputación de cada usuario prosumidor en función de los intercambios que haya realizado durante dicho periodo y si ha cumplido con los resultados de la subasta a los que se había comprometido o no.

Los datos que almacena la cadena de bloques con los datos personales de los usuarios prosumidores son:

- Nombre y apellidos del prosumidor.
- Dirección de la vivienda.
- Tipo de generación (si la hubiera)
- *Tokens* disponibles.
- Reputación.

- Número secreto de identificación en el resto de cadenas.
- Si el usuario es apto o no para participar en el mercado.

3.3.1 Contratos inteligentes

El código que se ha implementado en la cadena de bloques que contiene la información relativa a los datos personales de los usuarios prosumidores, consta de dos transacciones que pueden ser ejecutadas por el gestor del sistema:

- **Selección usuarios aptos:** La primera transacción de las implementadas, se encarga de comprobar los datos de reputación y cantidad de *tokens* que posee cada usuario, y en función de los parámetros requeridos por el mercado, se le proporciona permiso para entrar en la subasta de energía o no.
- **Finalización del mercado:** La segunda transacción tiene como objetivo eliminar los permisos de introducción de datos a todos los participantes del mercado una vez que todos los admitidos en la ronda ya hayan introducido su oferta de compra o venta de energía.

3.3.2 Permisos

Para garantizar la privacidad y la veracidad de los datos almacenados en esta cadena de bloques se han establecido una serie de permisos de edición y creación de datos y de ejecución de contratos inteligentes.

El gestor del sistema tiene permisos para realizar las siguientes funciones en la cadena de bloques que contiene la información personal de los prosumidores:

- Creación de nuevos usuarios.
- Modificación de los datos personales.
- Ejecución de los contratos inteligentes.

Por el contrario, los usuarios prosumidores no tienen ningún permiso de creación o modificación de datos, ni pueden ejecutar ninguna de las transacciones, únicamente tienen permiso para visualizar sus datos personales, lo que resulta especialmente útil para conocer el número de *tokens* que tienen disponibles o la reputación con la que cuentan en un determinado momento.

3.4 Cadena de bloques de mercado

La segunda cadena de bloques es la que almacena los datos correspondientes a la subasta de energía. Además, en esta cadena están almacenados los datos de precios de los intercambios con la red principal, para que los puedan consultar los sistemas de gestión de la energía de cada usuario prosumidor.

En cada ronda de mercado esta cadena recibe, en primer lugar, los números de identificación secreta de los usuarios que pueden participar en la subasta de energía. Posteriormente, se comunica a todos los usuarios que ya se ha seleccionado quién es apto para participar. Cada usuario comprueba si es apto, y en caso afirmativo, envía su oferta de compra o venta de energía.

Dichas ofertas pueden tener diferente grado de complejidad, desde ofertas tan simples como únicamente una cantidad de energía a comprar o vender, hasta ofertas complejas que incluyen, además de la cantidad de energía, un precio límite (máximo o mínimo), el tipo de generación que se desea, la distancia entre

el generador y el consumidor, etc. En esta primera versión del sistema únicamente se ha dotado al mercado de capacidad para casar ofertas por una cantidad de energía, pero en posteriores mejoras podría implementarse una mayor complejidad a las subastas, para garantizar que la energía que se compra o vende en dicho mercado se ajusta a la que los usuarios desean.

Posteriormente, las ofertas se casan entre sí, y el resultado se almacena en esta misma cadena. Se envía a todos los usuarios un mensaje indicando que los resultados están disponibles. Cada usuario prosumidor puede únicamente consultar los resultados de la subasta en los que es parte implicada, ya sea como vendedor o comprador de energía.

El precio al que se casan las ofertas se ha establecido, en esta primera versión (ecuación 3.1), como el precio de compra de energía a la red principal multiplicado por la cantidad de energía por la que se han enviado ofertas de compra, más el precio de venta de la energía a la red principal multiplicado por la cantidad de energía por la que se han enviado ofertas de compra entre la suma de ambas cantidades de energía, por lo tanto, si todos los usuarios quisieran comprar o vender energía, el precio sería igual al de los intercambios con la red principal, y si hay ofertas tanto de compra como de venta se ven beneficiados tanto los compradores como los vendedores de energía, pues obtienen mejores precios. Este mecanismo produce precios más baratos cuando hay excesos de energía y más caros cuando es escasa. De esta forma se incentiva a los usuarios a tener sistemas de almacenamiento que logren aplanar la curva de consumo, lo que repercutiría en una gestión más sencilla de la microrred y una menor dependencia de la red principal.

$$\frac{Precio_C \cdot Cantidad_C + Precio_V \cdot Cantidad_V}{Cantidad_C + Cantidad_V} \tag{3.1}$$

En posteriores versiones ha de implementarse un mecanismo de casado de ofertas más complejo, que introduzca más variables, no únicamente la cantidad de energía y el precio de los intercambios con la red principal, que pueda determinar un precio diferente para cada uno de los resultados del mercado y no un mismo precio para todos.

Los datos almacenados en esta cadena son:

- Precios de los intercambios con la red principal a cada hora del día.
- Ofertas de compra o venta de energía.
- Resultados del casamiento de las ofertas del mercado.

3.4.1 Contratos inteligentes

El código que se ha implementado dentro de la cadena de bloques para que pueda ser ejecutado por los usuarios consta de cuatro transacciones, tres de ellas solo pueden ser ejecutadas por el gestor del sistema, y la cuarta únicamente por los usuarios prosumidores:

- **Inicialización del mercado:** Esta transacción solo puede ser ejecutada por el gestor del sistema, mediante el envío de un conjunto de números de usuarios que van a ser admitidos en el mercado. Lo que hace al ser ejecutada es indicar dentro de la cadena de bloques del mercado que los usuarios recibidos son aptos para participar en la subasta de energía. Cuando finaliza su ejecución emite una notificación a los clientes prosumidores, que comprueban si han sido admitidos o no en la subasta.
- **Insertar oferta:** Si el usuario prosumidor ha sido admitido en la subasta, inmediatamente envía una oferta de compra o venta de energía. Estas ofertas son almacenadas en la cadena de bloques para evitar su manipulación.

- **Casar ofertas del mercado:** Mediante esta función, que únicamente puede ser llamada por el gestor del sistema, se unen las ofertas de compra y venta de energía entre sí, uniendo a cada usuario con exceso de energía con usuarios con déficit y viceversa. El resultado de esta transacción se almacena en la cadena de bloques para evitar que sea modificado.
- **Finalización del mercado:** Cuando los resultados de la ejecución del casamiento de las ofertas están disponibles, el gestor del sistema ejecuta esta transacción para poner de nuevo a todos los usuarios como no aptos para participar en el mercado, y de esta forma se deja preparada la cadena de bloques para volver a iniciar otra ronda de mercado.

3.4.2 Permisos

Para garantizar la privacidad y la veracidad de los datos almacenados en esta cadena de bloques se han establecido una serie de permisos visualización de datos y ejecución de transacciones.

El gestor del sistema tiene permisos para realizar las siguientes funciones en su interacción con la cadena de bloques:

- Ejecución de las transacciones de inicialización del mercado, casar las ofertas del mercado y la finalización del mercado.
- Visualizar las ofertas introducidas por los prosumidores.
- Visualizar los resultados del casamiento de ofertas del mercado.

El usuario prosumidor, por su parte, tiene permisos para realizar las siguientes funciones, estando el resto denegadas:

- Visualizar si ha sido admitido o no en el mercado.
- Ejecutar la transacción de inserción de la oferta de compra o venta de energía en la cadena de bloques.
- Visualizar los resultados del casamiento de ofertas del mercado en los que es parte implicada (ya sea como comprador o vendedor).

3.5 Cadena de bloques de registro de parámetros eléctricos

En la tercera cadena de bloques del sistema se introducirán los parámetros eléctricos proporcionados por el contador inteligente de cada usuario prosumidor.

Gracias al registro de estos datos se pueden realizar comprobaciones, de manera posterior al periodo de intercambios de energía, sobre la cantidad insertada o absorbida de la red de distribución, así como tener un mayor control de la calidad de la electricidad de la microrred. De esta forma, se puede asignar un responsable a las diferencias entre la energía vendida en la subasta y la que finalmente llega al comprador. Se puede determinar a qué se debe esa diferencia (menor producción, mayor consumo, pérdidas inesperadas en el transporte, etc.) y de esta forma poder asignar los costes del ajuste que ha debido realizarse, además de, si es necesario, introducir una penalización en la reputación del usuario.

La introducción de datos en esta cadena representa el punto más vulnerable del sistema, pues los datos pasan del contador inteligente del usuario prosumidor al nodo que introduce dichos datos en la

cadena de bloques que, al no estar protegido el contador inteligente con la tecnología *blockchain*, podría modificarse si el algoritmo de cifrado que emplea en su comunicación con el nodo no es suficientemente robusto, y el usuario prosumidor podría aprovecharlo para su propio beneficio. Una posible solución a este punto débil sería la implementación del nodo dentro del propio contador inteligente, de forma que se vayan introduciendo los parámetros en la cadena de bloques según se vayan midiendo.

En un primer momento se ha planteado que todos los datos de esta cadena de bloques sean introducidos, por los usuarios prosumidores, cada vez que finalice una ronda de mercado, por simplicidad en la programación, pero en posteriores versiones deberían introducirse los datos de parámetros instantáneos de manera continua, con el fin de dar un menor tiempo para la modificación malintencionada de estos datos. Si dichos datos no se introdujeran en el sistema en el momento que debieran y tuvieran un retardo, serían sospechosos de haber sufrido una modificación.

En esta cadena únicamente el gestor del sistema y el usuario prosumidor puede consultar sus propios datos, de manera que se garantiza la privacidad de los usuarios. Además, para aumentar la protección, se tiene un dato adicional, calculado por la propia cadena, que es la diferencia entre lo generado y lo consumido, que es el único dato que se transfiere a la cadena de liquidación para realizar los pagos en esta primera versión de la plataforma, así hay un menor riesgo de que los parámetros eléctricos sean visualizados, ya que podrían servir para generar un perfil de consumo del que se podría extraer mucha información.

Esta cadena de bloques se ha diseñado para almacenar los siguientes datos:

- Parámetros eléctricos acumulados
  - Potencia activa consumida.
  - Potencia reactiva consumida.
  - Potencia activa generada.
  - Potencia reactiva generada.
- Parámetros instantáneos.
  - Potencia activa consumida.
  - Potencia reactiva consumida.
  - Potencia activa generada.
  - Potencia reactiva generada.
  - Voltaje.
  - Intensidad.
  - Frecuencia.
- Diferencia entre generación y consumo.

3.5.1 Contratos inteligentes

El código que se ha implementado dentro de la cadena de bloques para que pueda ser ejecutado por los usuarios consta de una única transacción, que puede ser llamada por los usuarios prosumidores.

- **Calcular diferencia:** Coge los datos de generación y consumo acumulados durante el periodo establecido del usuario prosumidor y calcula la diferencia entre ambos parámetros, para pasar un único dato a la cadena de liquidación de pagos, y así minimizar el riesgo de una visualización de los datos de generación y consumo de electricidad del usuario prosumidor, lo que podría repercutir en una violación de su privacidad.

3.5.2 Permisos

En esta cadena de bloques el gestor del sistema solo tiene permisos de visualización de los datos introducidos por los prosumidores, para su supervisión, pero no puede realizar ninguna edición o inserción de datos.

Por su parte, el usuario prosumidor, tiene permisos para visualizar, únicamente, sus datos, garantizando de esta forma la privacidad de los datos de todos los usuarios. Además, tiene permisos para insertar los datos de su contador inteligente y para ejecutar la transacción que calcula la diferencia entre lo que ha generado y lo que ha consumido durante el periodo.

3.6 Cadena de bloques de liquidación de los pagos

En la última de las cuatro cadenas de bloques que componen la plataforma de intercambios de energía, se almacenan los datos correspondientes a la liquidación de los pagos derivados de los resultados de la subasta, almacenados en la segunda cadena de bloques, que almacena los datos del mercado (3.4), y a las penalizaciones derivadas de la comparación de los parámetros eléctricos registrados durante el periodo, almacenados en la tercera cadena (3.5) y los resultados de la subasta. En esta cadena de bloques quedan registradas todas y cada una de las transacciones de energía que se producen en el sistema, así como su precio y la estampa de tiempo.

En primer lugar, una vez finalizada la ronda de mercado, esta cadena ejecuta las transacciones económicas correspondientes a los resultados de la subasta, restando la cantidad de *tokens* establecidos en dicho resultado al comprador y sumándoselos al vendedor. Además, se actualiza el desajuste de energía de cada usuario.

En segundo lugar, se comprueba si el dato de desajuste de energía se ha quedado a cero, y si no es así, se tratan de casar las diferencias de cada uno de los usuarios prosumidores entre sí. Para realizar este casamiento se ha desarrollado un algoritmo en el que los primeros que consiguen asignar sus diferencias son los que menor diferencia tienen, para fomentar que los usuarios prosumidores tengan buenos sistemas de gestión de la energía que realicen unas predicciones adecuadas. Mientras mayor es la diferencia con cero mayor repercusión tendrá en la reputación del usuario, y mayor probabilidad de no encontrar otro prosumidor con quien casar esa diferencia realizando una transacción a un precio más interesante que el del mercado mayorista.

Por último, a los usuarios que no se les ha podido crear una transacción con otro usuario, se les anota un intercambio de energía con la red principal por una cantidad igual a la energía que no ha sido posible casar. Este intercambio supone una transacción económica con una empresa comercializadora, y puede realizarse de dos formas:

1. Si la empresa comercializadora está dada de alta en la plataforma de intercambios de la microrred como un usuario más, se puede realizar la transacción mediante *tokens* de manera instantánea, al igual que se realizan los intercambios entre el resto de usuarios.

2. Si la comercializadora no está dada de alta en el sistema, se realizará una facturación a final de mes en la que el pagador es el gestor de la microrred. En este caso, las transacciones con la comercializadora crearán o eliminarán *tokens*. El gestor del sistema tendrá dinero disponible gracias a que los usuarios ya habían pagado para recibir los *tokens* que se eliminan en las transacciones con la comercializadora.

Los datos almacenados en esta cuarta cadena de bloques son los siguientes, en los que las transacciones constan de una estampa de tiempo, una cantidad de energía, un precio, un comprador y un vendedor:

- Transacciones realizadas conforme a lo estipulado en los resultados de la subasta de energía.
- Transacciones de ajuste realizadas.
- Transacciones realizadas con la empresa comercializadora de energía.
- Repercusión de las transacciones ejecutadas en el periodo en la reputación de cada usuario prosumidor.
- Repercusión de las transacciones ejecutadas en el periodo en el número de *tokens* de cada usuario prosumidor.

3.6.1 Contratos inteligentes

Las funciones con código ejecutable que se han integrado en esta cadena de bloques únicamente pueden ser llamadas por el gestor del sistema, y se encargan de dejar constancia de todos los intercambios de energía que se han hecho en la microrred. En las transacciones que no estaban definidas previamente como resultados de la subasta de energía, se establece el comprador y vendedor, así como el precio de este intercambio. Las funciones implementadas son las siguientes:

- **Transacciones:** Se encarga de leer las transacciones que se le indique y ejecutarlas, dejando reflejado en los datos de cada usuario prosumidor el número de *tokens* y la energía que ha supuesto dicha transacción. Cuando finaliza envía una señal para que se sepa que ha terminado.
- **Casar el desajuste:** Durante el periodo de mercado, los usuarios no han tenido por qué producir o consumir exactamente lo que habían previsto en la oferta que lanzaron. Los usuarios que no han participado en el mercado, también tienen un desajuste entre lo consumido y lo generado. Esta función casa estos desajustes entre sí y crea nuevas transacciones que son liquidadas llamando de nuevo a la función de transacciones.
- **Transacciones con la comercializadora:** Para los usuarios que continúan con un desajuste después de haber realizado los dos tipos de transacciones (las establecidas por la subasta de energía y las de eliminación del desajuste), esta función crea una transacción con la empresa comercializadora por una cantidad de energía igual al desajuste que tiene dicho usuario prosumidor. Además de crear estas transacciones, la función también las liquida, dejando reflejado en los datos del prosumidor el número de *tokens* que ha supuesto la transacción.

3.6.2 Permisos

En esta cadena de bloques los usuarios prosumidores únicamente tienen permiso para visualizar las transacciones relacionadas con ellos mismos (ya sea como comprador o vendedor de energía). Pueden ver

cuántos *tokens* ha supuesto cada una de las transacciones en las que ha participado en cada ronda de mercado, y si ha habido una repercusión en su reputación derivada de dichas transacciones.

El gestor del sistema, además de poder visualizar todos los datos para supervisarlos, puede ejecutar todos los contratos inteligentes de la cadena de bloques.

Capítulo 4

Interacción de los usuarios con la aplicación

En el presente capítulo se describen los diversos programas desarrollados para interaccionar con la plataforma de intercambios de energía.

En primer lugar, se describen los *scripts* escritos con órdenes en lenguaje de consola de GNU/Linux [40] que sirven para crear una red de *Hyperledger Fabric*, unir nodos a ella, crear a los usuarios prosumidores e instalarlos posteriormente en su propio nodo en la sección 4.1.

Posteriormente, en la sección 4.2, se detallan los programas desarrollados para la interacción entre los usuarios y las cadenas de bloques. Estos programas han sido escritos en *Node.js*[1] debido a que es el lenguaje para el que hay desarrollada una API[2] por parte de *Hyperledger*, que facilita notablemente la programación en el más alto nivel y que actualmente tiene un gran soporte por parte de la comunidad.

Estas interacciones se han planteado para que los usuarios no tengan que realizar ninguna acción para que el flujo del mercado avance, es decir, el programa se activa la primera vez que se instala, y va ejecutando las acciones correspondientes sin necesidad de intervención humana.

El código de todos los programas explicados a lo largo del capítulo se encuentra en el Anexo A y parten de la base de que *Hyperledger Fabric* está descargado y modificado convenientemente tal y como se indica en el anexo B en todos los equipos que van a formar parte de la red blockchain. Además, todos estos equipos deben estar en comunicación, bien a través de Internet o de una red de telecomunicaciones propia, y ser visibles entre ellos.

Se ha creado una carpeta con una estructura de archivos fija, para que cuando se lleve a otro ordenador, los órdenes de cambio de carpeta o para buscar archivos sigan funcionando, sin tener que realizar ninguna modificación.

En principio, el equipo del gestor del sistema va a ser un ordenador, en el que se van a poder visualizar todos los datos para supervisarlos, mientras que el de los usuarios prosumidores va a ser un equipo de bajas prestaciones, probablemente una placa de desarrollo tipo Raspberry Pi [42], ya que se pretende que tenga un coste reducido, y en un futuro poderlo integrar con la infraestructura de medición avanzada y el sistema de gestión de la energía del usuario, para que con un solo producto tenga cubiertas sus necesidades con respecto a la gestión de la energía de su hogar.

[1] *Node.js* es un entorno multiplataforma, asíncrono y de código abierto, escrito en *JavaScript*, pero que no se ejecuta en el navegador, sino en el servidor [49].

[2] Una API es una interfaz de programación de aplicaciones, es decir, un conjunto de funciones que ofrece una biblioteca para ser usada por otro software [41], que proporciona un conjunto de funciones a los programadores para evitar programar toda la aplicación en el más bajo nivel.

4.1 Archivos de preparación e instalación

4.1.1 Instalación

El primer programa desarrollado consiste en una serie de órdenes (A.22) que se encargan de realizar las funciones necesarias para dejar el ordenador del gestor del sistema preparado para administrar la plataforma de intercambios de energía:

1. En primer lugar, elimina todas las instalaciones anteriores, dejando al equipo preparado para una nueva instalación limpia.
2. Desinstala a todos los usuarios de instalaciones anteriores, pues no van a funcionar en la nueva instalación.
3. Inicia *Hyperledger Fabric* y crea una tarjeta de administrador de la red.
4. Instala e inicia las cuatro cadenas de bloques en *Hyperledger Fabric*.
5. Por último, cada una de las cadenas de bloques instaladas crea una tarjeta de administrador de la cadena y la instala en el ordenador.

Para ejecutar este programa se deben introducir los siguientes comandos en el terminal de Ubuntu:

```
$ cd archivos_instalacion/
$ ./instalacion.sh
```

En el resto de equipos de la red basta con ejecutar el siguiente comando en el terminal de Linux para que se unan a la red y se conviertan en nodos una vez que tienen en su interior los archivos necesarios:

```
$ cd archivos_instalacion/
$ ./instalacion_ordenador2.sh
```

El código del script ejecutado se encuentra en el anexo y A.23, y realiza la función de iniciar un nodo de *Hyperledger Fabric* y unirlo a la red creada por el gestor del sistema.

4.1.2 Creación de usuarios prosumidores

La creación de usuarios prosumidores se debe realizar en el equipo del gestor del sistema, puesto que es el que tiene instaladas las tarjetas de identificación del administrador del sistema, único usuario al que se ha concedido permiso de creación de usuarios en las cadenas de bloques. Se inicia mediante la ejecución de las siguientes órdenes en el terminal de Linux:

```
$ cd archivos_instalacion/
$ ./crearUsuario.sh
```

Esto ejecuta una batería de órdenes que se pueden ver en el anexo A.25 que realiza todas las tareas necesarias para dar de alta a un usuario en la plataforma de intercambios de energía.

Para la creación de usuarios prosumidores se le deben indicar al programa los datos personales del usuario. En la primera versión, para realizar las pruebas, en lugar de adquirir los datos personales por entrada desde el teclado del equipo, se han dado unos valores de ejemplo para agilizar el proceso.

Una vez creados definidos los datos de usuario, bien introducidos por teclado (versión definitiva) o los que están por defecto (pruebas), se crea un usuario en la cadena de bloques de datos personales, al que se le asigna un número de identificación secreto que se genera de manera aleatoria.

Asociado a este usuario de la primera cadena de bloques, se crean otros tres usuarios en cada una de las otras tres cadenas de bloques restantes de la red, cuyo nombre de usuario será el número de identificación secreto que se generó en la primera cadena de bloques. Este número de identificación secreto es la única vinculación que existe entre el usuario de la cadena de bloques con los datos personales y el resto de cadenas de bloques, por lo tanto, si se consiguiera acceder a los datos de cualquiera de las tres cadenas de manera malintencionada, no se podría saber a quién corresponden esos datos a no ser que se tuviera acceso también a la cadena de bloques que almacena los datos personales, y de esta forma encontrar la correspondencia entre los datos personales que identifican al usuario y el resto de datos.

Tras crear los usuarios en todas las cadenas de bloques, el programa crea una carpeta en la que incluye las tarjetas de identificación correspondientes. Esta carpeta contiene todo lo necesario para iniciar, mediante un único comando, la aplicación en el equipo del prosumidor.

Para realizar pruebas se ha creado además otra versión del programa que crea el número de usuarios que se desee, los instala y pone en funcionamiento en el propio ordenador del gestor del sistema, esta versión está disponible en el anexo A.24.

4.1.3 Alta en el sistema por parte de los usuarios prosumidores

Para dar de alta a los usuarios prosumidores en el sistema, basta con copiar la carpeta generada en el equipo del gestor del sistema con la ejecución del programa descrito en el apartado anterior (4.1.2), para crear al usuario, en el equipo del prosumidor y ejecutar el *script* contenido en dicha carpeta.

```
$ cd archivos_instalacion/
$ ./instalarUsuario.sh
```

Mediante la ejecución de dicho programa se inicia *Hyperledger Fabric* en el equipo, que se une automáticamente a la red creada por el ordenador del administrador del sistema, se instalan en el equipo las tarjetas de usuario (se encuentran en la carpeta) y se inicia el programa que interacciona con las cadenas de bloques en las diferentes fases temporales del proceso de intercambios de energía, principalmente introduciendo las ofertas de compra o venta de energía y los datos de sus parámetros eléctricos.

El código de esta batería de órdenes se encuentra en el anexo A.26.

4.1.4 Desinstalar usuarios prosumidores

Además, si por alguna razón se desease desinstalar a un usuario prosumidor de un equipo, también se ha creado una batería de comandos para ello A.27, que da a elegir entre todos los usuarios que están instalados en el equipo, y del seleccionado elimina todas sus tarjetas de identificación. Se puede ejecutar mediante:

```
$ cd archivos_instalacion/
$ ./desinstalarUsuario.sh
```

Para pruebas se ha creado otra versión de este programa que elimina todas las tarjetas de identificación de usuarios prosumidores instalados en el equipo A.28.

4.2 Clientes de interacción con la plataforma

Para crear a los usuarios prosumidores, y para iniciar los programas que se estarán ejecutando de manera constante en los equipos de todos los usuarios, los archivos de instalación vistos en el apartado 4.1 hacen

uso de varios clientes escritos en *Node.js*, que comunican con las cadenas de bloques mediante un lenguaje de alto nivel, apto para programadores sin excesiva experiencia en el desarrollo de redes *blockchain*.

4.2.1 Cliente de creación de usuarios prosumidores

En el apartado 4.1.2 se ha visto el archivo que se ejecuta para la creación de los usuarios y su vinculación con una tarjeta de identificación. La primera parte de esa creación de usuarios, se realiza mediante una interacción con las cadenas de bloques, a través de un programa de *Node.js* (A.29). Este cliente se encarga de las siguientes funciones:

1. Crea un usuario en la cadena de bloques de los datos personales.
2. A este usuario se le asignan valores por cada uno de los datos que se piden.
3. Se le asigna un número de identificación secreto.
4. Se crea un usuario en cada una de las tres cadenas de bloques restantes, asignándole como nombre de usuario el número de identificación secreto almacenado en la cadena de bloques de datos personales.
5. Se crean los activos correspondientes en cada una de las tres cadenas, inicializados con valores nulos.

Para facilitar la ejecución de pruebas, a la hora de crear los usuarios prosumidores, hay varios valores que se asignan de manera fija en lugar de pedirlos por teclado. Entre otros, se establecen unos valores de datos personales iguales para todos los usuarios que se creen. Además, el nombre de identificación de usuario tampoco se pide, se asigna un número aleatorio para que no coincidan a la hora de crear una gran cantidad de prosumidores.

4.2.2 Cliente del gestor del sistema

Cuando la red de cadenas de bloques está montada y las cadenas de bloques instaladas en dicha cadena, ya puede iniciarse el mercado de subasta de energía. Para automatizar el proceso, y que el gestor del sistema no tenga que estar llamando de manera manual a cada una de las funciones, se crea un cliente en *Node.js* (A.4) al que se le da un periodo de tiempo, y ejecuta las siguientes funciones:

- En la fase previa al periodo de mercado se pide una cantidad de tokens y una reputación mínima y a partir de ahí:
  1. Se ejecuta la función implementada en la primera cadena que selecciona los usuarios que cumplen con los requisitos mínimos de participación en la subasta de energía.
  2. Los usuarios que devuelve la primera cadena se le pasan a la segunda cadena.
  3. Se detecta cuando los usuarios aceptados en el mercado han insertado sus ofertas.
  4. Se llama a la función que une unas ofertas con otras.
  5. Se detecta cuando se han casado todas las ofertas y se ejecuta la función que reinicia la cadena de mercado y la deja lista para una nueva subasta.
- Finalizado el periodo de mercado, en el que los usuarios prosumidores han estado inyectando y extrayendo energía de la red de baja tensión e insertando en la aplicación los datos de los parámetros eléctricos correspondientes se ejecuta la fase posterior al periodo de mercado del cliente del gestor del sistema:

1. Se introducen en la cadena de liquidación de pagos los datos correspondientes a los parámetros eléctricos de los usuarios prosumidores durante el periodo.
2. Se llama a la función que ejecuta todas las transacciones establecidas previamente en la subasta de energía.
3. La ejecución de transacciones ha dejado un desajuste con los datos reales, por tanto, se llama a la función que casa los desajustes.
4. Se llama a la función que une los desajustes entre sí.
5. Se llama a la función que crea y ejecuta transacciones entre los usuarios que aún tienen desajuste y las empresas comercializadoras.

- Una vez al día se ejecuta la función que descarga e introduce en el sistema los precios de venta al pequeño consumidor de la página web de e·sios[3] para que estén disponibles para todos los usuarios dentro de la cadena de bloques, y en futuras versiones, poder implementar sistemas de gestión de la energía dentro de las propias cadenas de bloques a modo de contratos inteligentes 2.5.4.

Una de las ventajas de *Node.js* es ser asíncrono. Esto permite ejecutar funciones sin importar que ya se esté ejecutando otra en ese mismo instante. De esta forma, podemos implementar la llamada a las funciones de la fase previa del mercado para seleccionar a los usuarios que participan en él y casarlos entre ellos, y la fase posterior, en la que se realizan los pagos a los usuarios, en un único programa.

4.2.3 Cliente del usuario prosumidor

Cada usuario prosumidor tiene en su equipo ejecutándose un archivo en *Node.js* (A.5) que se encarga de automatizar las tareas relacionadas con su intervención en el mercado de intercambios de energía de la microrred eléctrica en la que participa:

- En la fase previa al periodo de intercambios de energía se realizan las siguientes funciones:
  1. Se detecta cuando ha finalizado la selección de los prosumidores que participan en el mercado.
  2. Se comprueba si se puede participar.
  3. Se inserta la oferta desde un archivo que ha dejado el sistema de gestión de la energía con la estrategia a seguir.
  4. Se leen los resultados de la subasta en los que es comprador o vendedor y lo deja en un archivo para la lectura por parte del sistema de gestión de la energía.
- Desde que se instala al usuario en el equipo, se comienzan a introducir los datos de los parámetros eléctricos que se le dejan en un archivo, y los introduce en la aplicación.

[3] e·sios es el sistema de información del Operador del Sistema, desarrollado por Red Eléctrica de España. Realiza las tareas de información y gestión de los procesos relacionados con el mercado eléctrico [43]

Capítulo 5

Pruebas de funcionamiento realizadas

El presente capítulo tiene como objetivo comprobar el correcto funcionamiento del sistema desarrollado, conocer sus limitaciones y detectar los aspectos que pueden ser mejorados en futuras versiones. De manera desglosada, todas las pruebas de funcionamiento van a ir enfocadas a determinar:

- La validez de los modelos de cadenas de bloques diseñados.
- El correcto funcionamiento de los programas de interacción con las cadenas de bloques para los diferentes tipos de usuario desarrollados.
- La duración mínima de los periodos de intercambios de energía que permite al sistema funcionar de manera correcta.
- La sincronización de todos los nodos conectados a la red *blockchain* y que se pueden introducir datos en cualquiera de estos nodos.
- Los requisitos mínimos de funcionamiento, teniendo especial relevancia, el ancho de banda, la capacidad de computación y la necesidad de almacenamiento de datos.

En el apartado 5.1, se exponen las consideraciones previas que se han tenido en cuenta para modelar un escenario sobre el que realizar las pruebas de funcionamiento y, posteriormente, en los apartados 5.2 a 5.7 se describen todas las pruebas de funcionamiento que se han realizado sobre la plataforma, así como las conclusiones que se pueden extraer de los resultados obtenidos.

5.1 Consideraciones previas

En todas las pruebas realizadas a lo largo del presente capítulo, se parte de unas consideraciones previas para tratar de simular un escenario lo más realista posible y conseguir comprobar las características de funcionamiento de la plataforma. Estas consideraciones previas son:

- Los datos del contador de cada usuario prosumidor se han simulado mediante la creación de un archivo que contiene dos valores numéricos generados de manera aleatoria. Estos números aleatorios se corresponden con la cantidad de energía generada y consumida, respectivamente, a lo largo del periodo considerado.

- Los datos que ofrece el sistema de gestión de la energía para realizar una oferta en el mercado, se han simulado, al igual que los datos del contador, mediante la creación de un archivo con un valor numérico generado de manera aleatoria, que representa la cantidad de energía a comprar o vender en la subasta.

- La reputación de todos los usuarios que se dan de alta en el sistema es la máxima, y no se reduce en ningún momento, pues, aunque sería lógico que se redujera al estar introduciendo datos aleatorios, se quiere probar el funcionamiento más allá de un algoritmo de cálculo de la reputación del usuario, aún no desarrollado en esta primera versión.

- Todos los usuarios se dan de alta con un número de *tokens* suficientemente alto como para que el mercado los seleccione como aptos para participar en la subasta, aunque pasen varias rondas de mercado en las que el usuario tenga que comprar energía.

- Las primeras pruebas se realizan con un intervalo temporal entre rondas de mercado de un minuto, periodo suficientemente pequeño para simular un gran número de rondas. Cuando el se aumenta la necesidad de recursos conforme se incrementa la complejidad de las pruebas, el periodo de un minuto resulta insuficiente para realizar todas las funciones, por lo que se realizan rondas de cinco minutos.

- En todas las pruebas realizadas, se considera que el equipo del gestor del sistema funciona correctamente en todo momento, pues es el único equipo en el que está instalado el ordenante de transacciones, y ante cualquier fallo, otro equipo de la red *blockchain* no podría asumir su rol.

5.2 Prueba de funcionamiento de una ronda de mercado

La primera prueba tiene como objetivo comprobar que el flujo de mercado funciona correctamente, que cada oferta se casa produciendo un resultado de mercado mientras sea posible, que los desajustes después de la ronda de mercado se casan entre sí para eliminarlos, que se crea una transacción con la empresa comercializadora en caso de no encontrar otro usuario prosumidor, y que se suman o restan los *tokens* correspondientes a las transacciones realizadas.

Para realizar la primera prueba del sistema se emplea un único equipo, en el que se dan de alta 3 usuarios prosumidores, mediante las siguientes órdenes en el terminal de *Linux*:

```
$ cd archivos_instalacion/
$ ./instalarNUsuarios.sh
```

Tras introducir estas órdenes el terminal nos pregunta cuantos usuarios prosumidores se quieren, a lo que se responde 3. Se espera a que se creen e instalen y se observan los datos que van a introducir para posteriormente comprobar si se han realizado bien todas las fases:

Usuario	IDsecreto	Oferta (W·h)	Diferencia Real (W·h)
3322	9062	-186.74	-936.28
6963	8910	-109.25	70.87
8053	8933	305.03	-356.52

Tabla 5.1: Datos de los usuarios prosumidores de la prueba 1

Anotados en la tabla 5.1 todos los datos que se van a introducir se procede a activar el cliente del gestor para simular el flujo normal del mercado de energía mediante los siguientes comandos:

```
$ cd clientes/
$ nodejs clienteGestor.js
```

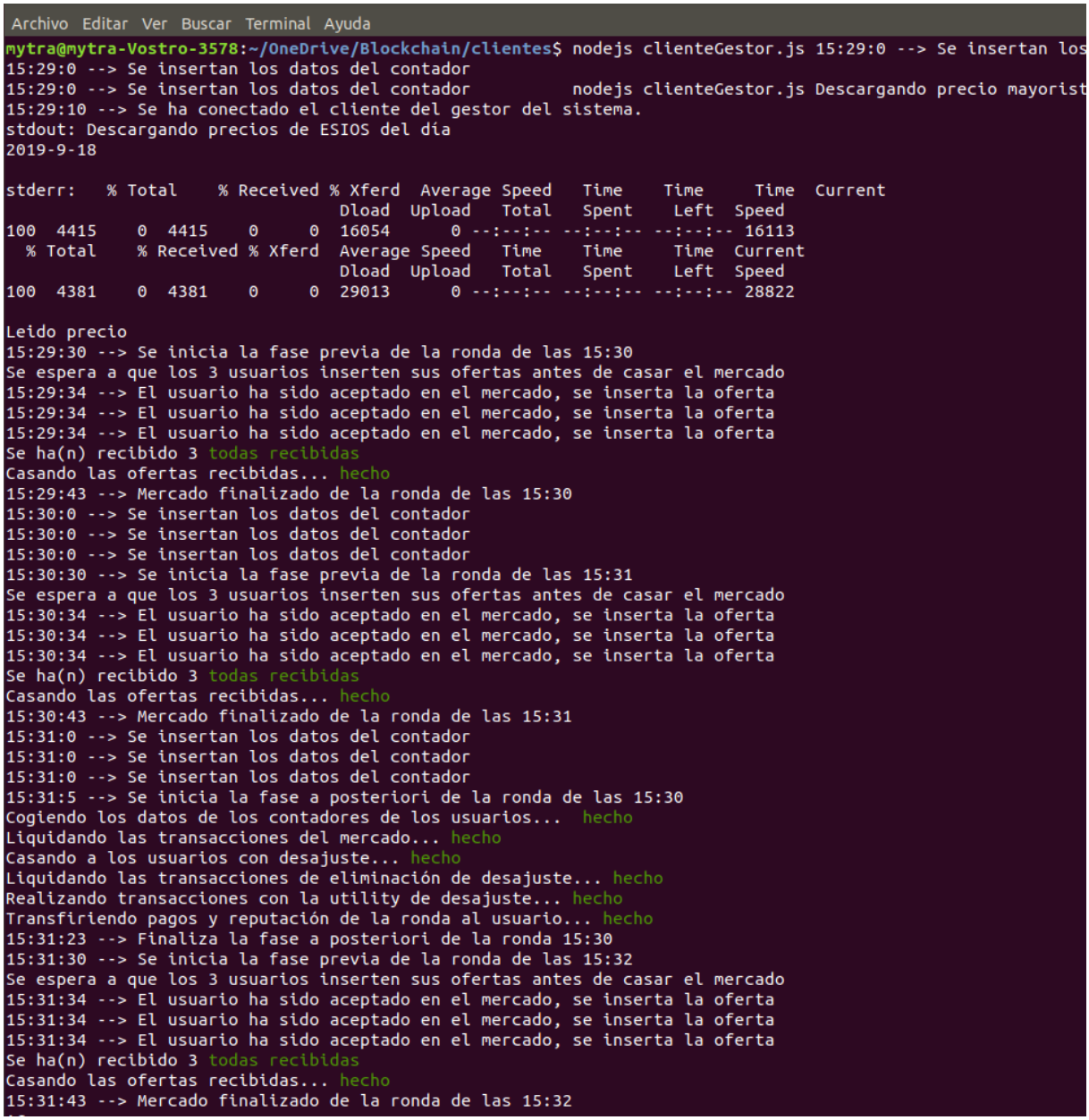


Figura 5.1: Salida en el terminal de Linux de la primera prueba.

Se obtiene el resultado que se puede observar en la figura 5.1 en el terminal de Linux, en la que se aprecia cómo en primer lugar, el cliente del gestor del sistema descarga los precios de los intercambios con la empresa comercializadora de energía. Posteriormente se puede ver que, el cliente de cada prosumidor está funcionando en segundo plano y cada vez que el reloj del equipo marca 0 segundos, introduce los datos de los parámetros eléctricos de su contador inteligente. Se observan además todos los pasos de la fase previa al periodo durante tres periodos, y una vez los pasos de la fase posterior al mercado.

5.2.1 Fase previa

Se inicia la fase previa de manera correcta y se selecciona a los tres usuarios, pues se les había dado la reputación y el número de tokens necesario para ello.

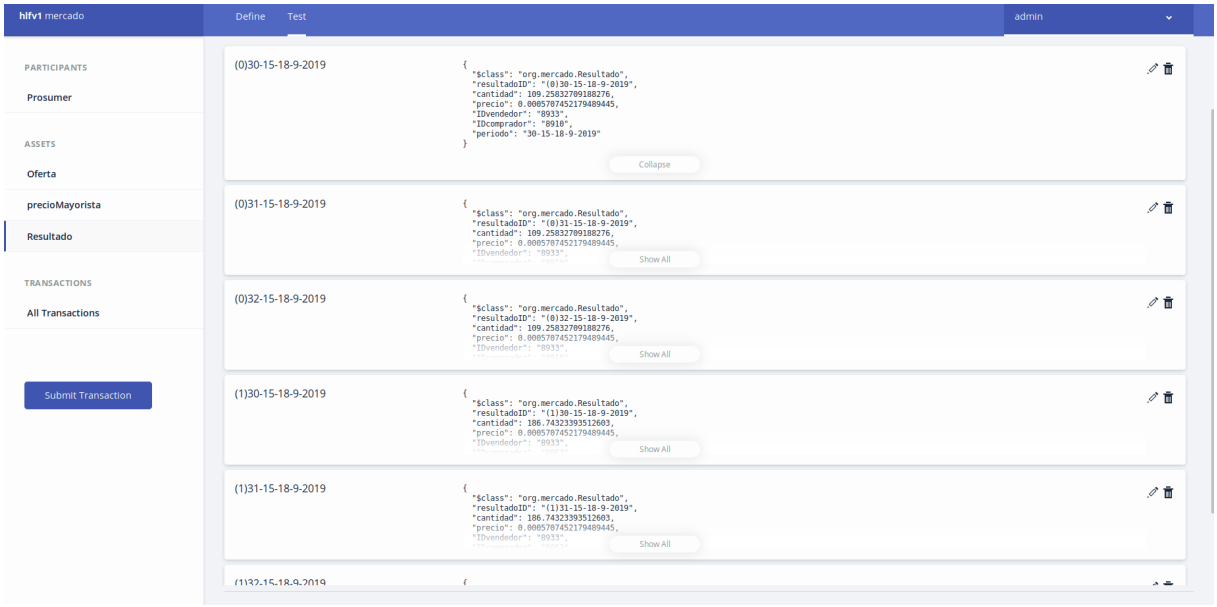


Figura 5.2: Resultados de mercado de la primera prueba.

Tras ser seleccionados para participar en el mercado, los tres clientes de los prosumidores lo detectan e insertan su oferta de manera correcta.

El mercado se casa de manera correcta, se crean dos transacciones entre el usuario 8053 (vendedor) y los otros dos prosumidores, y aun así le quedaría a este usuario 9.04 Wh que no han encontrado comprador. En la figura 5.2 se puede comprobar cómo se han creado estas dos transacciones por cada una de las veces que se ha casado el mercado.

El mercado se finaliza dejando a todos los usuarios como no aptos para participar, listo para comenzar una nueva ronda.

5.2.2 Fase posterior

Una vez finalizado el periodo de intercambios de energía, los usuarios insertan sus datos de parámetros eléctricos acumulados en la cadena de bloques para ese fin. Se recogen los datos de esta cadena y se insertan en la cadena de bloques de liquidación correctamente. Se ejecutan las transacciones con los

Usuario	IDsecreto	Oferta (W·h)	Diferencia Real (W·h)	Resultado tras transacción mercado (W·h)
3322	9062	-186.74	-936.28	-749.54
6963	8910	-109.25	70.87	180.13
8053	8933	305.03	-356.52	-652.01

Tabla 5.2: Datos de los usuarios prosumidores de la prueba 1

resultados del mercado de manera correcta, quedando a los usuarios un desajuste entre su diferencia real y la cantidad que habían conseguido vender en el mercado tal y como se puede ver en la tabla 5.2.

Como se puede observar, el resultado del desajuste tras aplicar la transacción establecida por el mercado, en lugar de mejorar ha empeorado, pero esto es debido a que la oferta se había tratado de un número aleatorio para realizar la prueba, pero el resultado de la transacción es el correcto.

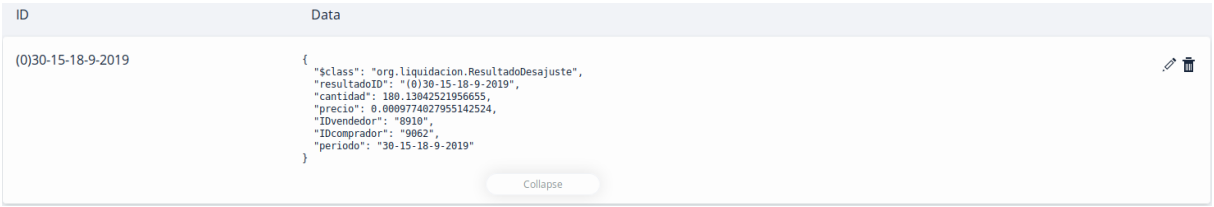


Figura 5.3: Resultados de casamiento de desajuste de la primera prueba.

Para eliminar estos desajustes de los usuarios prosumidores, se tratan de unir entre ellos. Como únicamente el usuario 6963 tiene energía excedentaria, se crea solamente una transacción entre este y el usuario 3322 (figura 5.3).

Tras ejecutarse esta transacción de manera correcta, los usuarios que tienen un desajuste diferente de cero deben crear una transacción con la empresa comercializadora. Estas transacciones se realizan de manera correcta, quedando todos los usuarios con un desajuste final igual a cero.

Cuando todos los usuarios prosumidores tienen sus desajustes igualados a cero, se transfieren los datos de *tokens* y reputación ganada o perdida durante el periodo a la cadena de datos personales. Lo realiza de manera correcta.

Los tokens de las transacciones con la comercializadora no los recibe nadie, se los queda el gestor del sistema, para que en el momento en el que la comercializadora emita la factura por la energía consumida por la microrred, este tenga el dinero para pagar.

Queda, de esta forma, comprobado el correcto funcionamiento de los contratos inteligentes implementados en las cadenas de bloques, si todos los usuarios son aptos para participar en el mercado.

5.3 Prueba de funcionamiento de una ronda de mercado con prosumidores no aptos

Con las mismas condiciones de la prueba anterior (tabla 5.1), se modifica uno de los usuarios para que no sea admitido por el mercado, y de esta forma, comprobar si se realiza bien la selección de usuarios para la participación en el mismo, al tener en cuenta la reputación y el dinero que tiene disponible cada usuario prosumidor.

Para ello, al usuario 3322, se le modifica la cantidad de *tokens* disponibles, que se establece en cero. Tras esta modificación, se vuelve a activar el cliente del gestor del sistema, que en el momento correspondiente realiza la fase previa de la ronda de intercambios de energía.

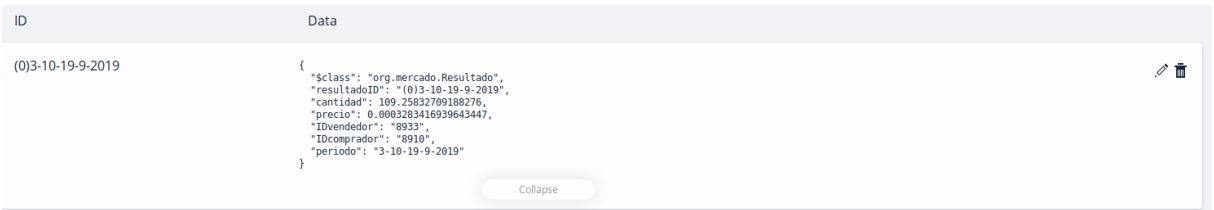


Figura 5.4: Resultados de la segunda prueba.

Esta vez, como era de esperar, el mercado devuelve un único resultado. Como se puede observar en la figura 5.4, este resultado del mercado es por una cantidad igual a la menor de entre las ofertadas por los dos prosumidores admitidos en la subasta de energía, siendo uno de ellos el comprador y otro el vendedor.

Por lo tanto, queda comprobado que el mercado excluye a los usuarios que no cumplen con las condiciones requeridas, establecidas de manera previa, para su participación.

Con estas dos pruebas realizadas (5.2 y 5.3), sin haber obtenido ningún error, ni de funcionamiento del sistema ni de cálculo, queda demostrado el correcto funcionamiento de los contratos inteligentes desarrollados, así como del modelo de cadenas de bloques.

5.4 Prueba de funcionamiento de un gran número de usuarios prosumidores en un único equipo durante un periodo de tiempo amplio

Esta prueba tiene el objetivo principal de comprobar la escalabilidad del sistema, verificando que cuando se aumenta el número de usuarios de manera significativa, estos pueden insertar datos de simultáneamente en la cadena de bloques. Además, se le deja funcionando durante cuatro horas y, de esta forma, demostrar que puede estar funcionando de manera continuada sin producir ningún error. La duración de cuatro horas de la prueba también permite comprobar la capacidad de almacenamiento necesaria.

Se dan de alta 50 usuarios prosumidores, que comienzan a introducir los datos de sus contadores inteligentes cada minuto. En esta prueba no se esperan errores por retrasos en la introducción de datos, pues los prosumidores y el ordenante de transacciones se encuentran en el mismo equipo, por lo tanto, las comunicaciones no introducen ninguna latencia.

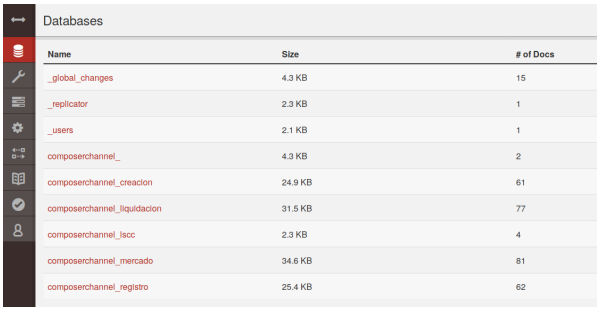
Durante toda la duración de la prueba, el sistema no ha devuelto ningún error, por lo tanto, la escalabilidad del sistema queda demostrada, al menos para 50 usuarios introduciendo datos de manera simultánea. También ha quedado demostrado el funcionamiento durante un periodo prolongado de tiempo. En cuanto a la capacidad de almacenamiento, se han realizado capturas de pantalla a lo largo de la prueba en diferentes instantes (figura 5.5), para comprobar cómo ha evolucionado la cantidad de datos almacenados en esta.

La base de datos tiene, antes de comenzar a insertar datos ningún usuario, recién instaladas las cadenas de bloques en la red, un tamaño en el orden de magnitud de los kB, como se puede apreciar en la figura 5.5a.

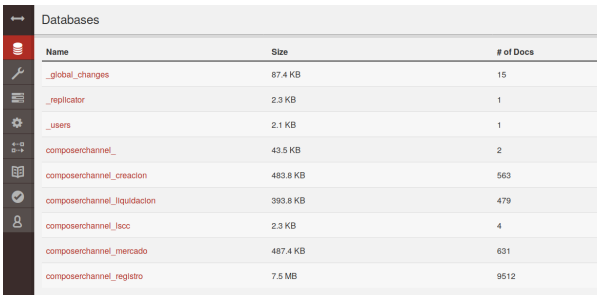
Tras una hora insertando los 50 prosumidores los datos correspondientes a sus contadores inteligentes (figura 5.5b), el orden de magnitud de la cantidad de datos almacenados en la cadena de bloques que contiene los parámetros eléctricos ha pasado de los KB a MB. En una hora ha aumentado 7,5 MB.

Tras cuatro horas en funcionamiento (figura 5.5c), el almacenamiento necesario para la cadena de bloques de registro de parámetros eléctricos ha aumentado hasta los 34,4 MB. Por lo tanto, suponiendo que el crecimiento se ha mantenido de forma lineal desde la primera hora hasta la cuarta, por cada hora en funcionamiento se necesitarían aproximadamente 9 MB de almacenamiento, y a partir de este dato, se pueden extraer los datos de la tabla 5.3, donde se observa que por cada vez que un usuario prosumidor inserta los datos de su contador en la cadena de bloques, esta aumenta en aproximadamente 30 kB.

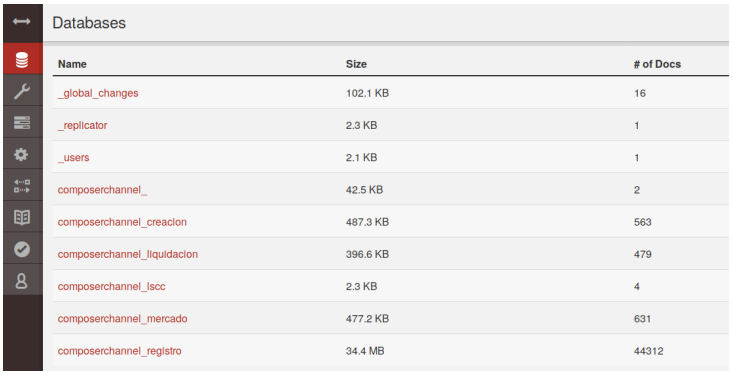
Al haber supuesto que el crecimiento ha sido lineal, y va a continuar creciendo al mismo ritmo, se puede intuir el tamaño que va a ocupar esta base de datos en un periodo de tiempo más amplio, bajo las mismas condiciones, y de esta forma, poder hacer una previsión de la capacidad de almacenamiento



(a) Antes de dar de alta a ningún usuario.



(b) Tras una hora de uso.



(c) Tras cuatro horas de uso.

Figura 5.5: Análisis de la capacidad de almacenamiento necesaria para almacenar la base de datos correspondiente a la cadena de bloques de registro de los parámetros eléctricos.

Cadena de bloques	Variación cada hora (MB)	Variación cada ronda (kB)	Variación cada usuario (kB)
Registro de parámetros eléctricos	9	150	30

Tabla 5.3: Variaciones en la cantidad de datos almacenados en la cadena de bloques de registro de los parámetros eléctricos

que sería necesaria. En un año se ocuparía 77 GB en cada uno de los nodos de la red *blockchain*, únicamente con la cadena de bloques de registro de parámetros eléctricos si se mantuviera una frecuencia de inserción de datos igual a la de la prueba, es decir, de un minuto. Esta capacidad de almacenamiento no resultaría desorbitada, pues en la actualidad hay a la venta, para el pequeño consumidor, unidades de almacenamiento de datos de varios TB a un precio relativamente asequible.

5.5 Prueba de funcionamiento añadiendo otro nodo

Se une un equipo más a la red *blockchain* tras realizar las modificaciones para unir más de un nodo indicadas en el anexo B.3, dónde se explica paso a paso qué archivos se deben modificar para conseguir adaptar la red de *Hyperledger Fabric* que se descarga por defecto al instalar por primera vez, a una red con dos equipos físicos en los que se alojan tres nodos. A partir de este ejemplo se puede modificar la red para adquirir la arquitectura de nodos y equipos que se desee.

La prueba se realiza en las condiciones de la figura 5.6, en la que se puede observar que, en el primer equipo, se crea la autoridad certificadora, el ordenante y dos nodos, mientras que el segundo equipo tiene únicamente un nodo.

En esta primera prueba con dos equipos físicos unidos a la misma red *blockchain*, se busca verificar el

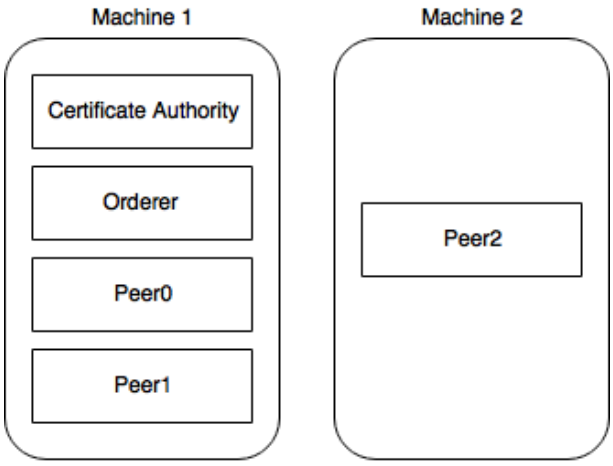


Figura 5.6: Estructura con dos equipos y tres nodos. En el primer equipo se encuentra la autoridad certificadora y el ordenante, que inserta en el resto de nodos los bloques de transacciones.

ancho de banda necesario para un correcto funcionamiento del sistema y comprobar si el nodo situado en el segundo equipo se sincroniza de manera correcta con los nodos situados en el primer equipo. Para ello, se instalan un número creciente de usuarios en el primer equipo, que están insertando los datos de sus contadores inteligentes cada minuto (simulados) en la cadena de bloques, y se analizan los intercambios de información que se producen entre ambos equipos mediante la herramienta cAdvisor [44].

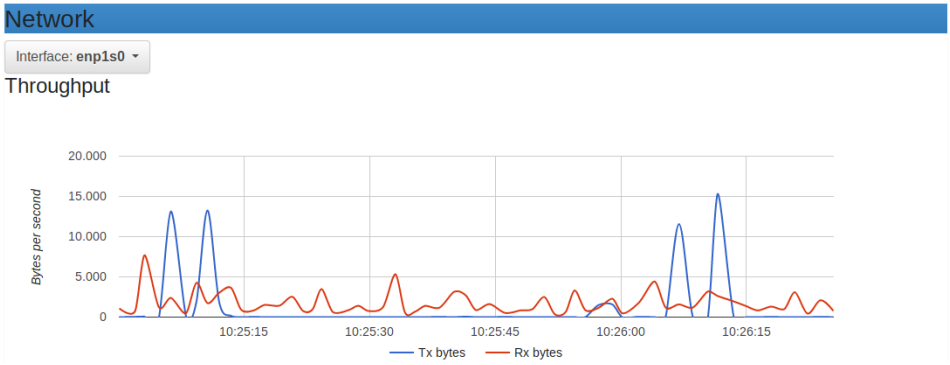
5.5.1 Análisis de un usuario

En el primer equipo, se instala un único usuario y esto produce que, cada vez que este inserta sus datos en la cadena de bloques, en el primer equipo se produce una necesidad de ancho de banda para la salida de datos del equipo de aproximadamente 15 kB/s, mientras que los picos de descarga de datos son de en torno a 7,5 kB/s como se puede ver en la imagen 5.7a. En el segundo equipo ocurre algo parecido (5.7b), pero de manera inversa, en este equipo se reciben picos de aproximadamente 13 kB/s y los picos de envío de datos están en el entorno de 4 kB/s.

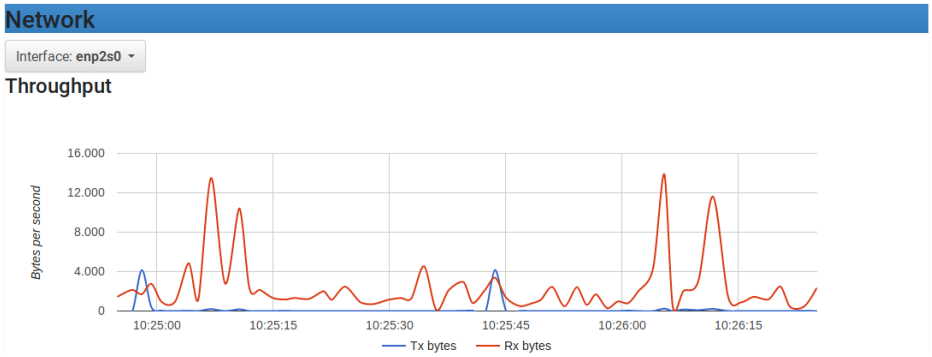
En esta primera prueba no se aprecia una forma clara en la transmisión, pues al ser tan pequeña la cantidad de datos transmitida, queda entremezclada con las señales de los nodos para asegurarse de que permanecen conectados y con otros datos que están recibiendo y enviando otros procesos que estén ejecutando los equipos en segundo plano. Para lograr una mejor comprensión de los datos que se han transmitido, el proceso de insertar los datos del contador del usuario en la cadena de bloques consiste en los siguientes pasos:

1. Se hace una petición a la cadena de bloques para que entregue al cliente la variable a actualizar.
2. El cliente envía los datos actualizados.
3. Se envía una petición de transacción

Por lo tanto, en el análisis del ancho de banda se deben apreciar tres picos, el primero, correspondiente a la recepción de la variable a actualizar, posteriormente un envío de los datos actualizados, y por último el envío de la petición de transacción. En esta primera prueba, aunque camuflado entre ruido, se pueden intuir estos tres picos en el equipo en el que se encuentra el cliente. Además, en el otro equipo, se puede ver como se reciben los datos que se envían por el primer equipo.



(a) Equipo 1.



(b) Equipo 2.

Figura 5.7: Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando el usuario instalado en el equipo 1.

5.5.2 Análisis de diez usuarios

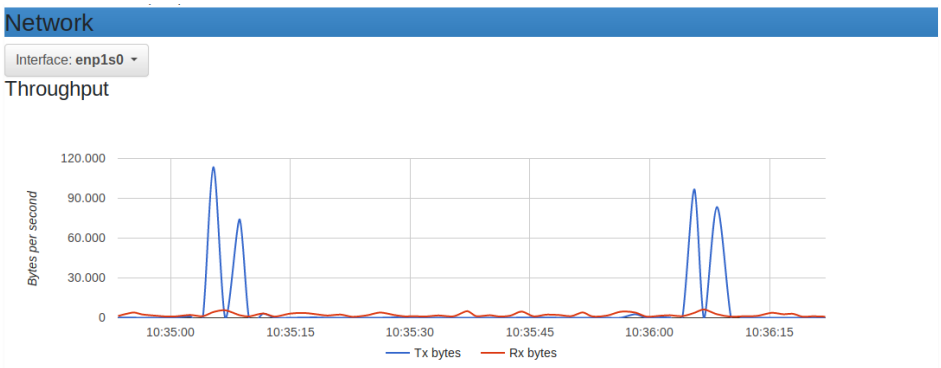
Si se eleva hasta diez el número de usuarios que están instalados, el ancho de banda necesario para el envío de datos se eleva, en este primer equipo (5.8a), hasta aproximadamente 120 kBps, mientras que la recepción de datos no se puede apreciar la variación al ser mucho más pequeño el ancho de banda que se necesita, y estar compartida la escala. En el segundo equipo (5.8b) también aumenta el ancho de banda necesario para la recepción de datos hasta aproximadamente 120 kBps, y la subida de datos ocurre el mismo problema que en el primer equipo, que no se puede apreciar debido a la escala.

En estas transmisiones de datos ya no se aprecian los tres picos del caso anterior, habiendo quedado el primero de ellos (petición de la variable a actualizar) difuminado por la escala, al ser mucho mayores los datos que se transmiten en los dos siguientes, que son los únicos que se aprecian.

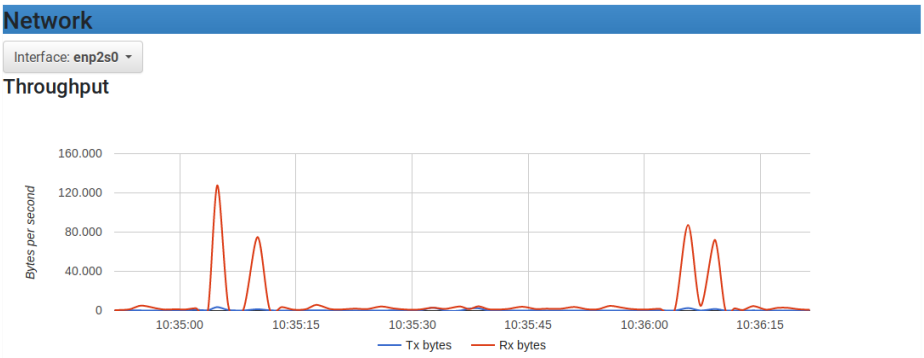
5.5.3 Análisis de cien usuarios

En un intento por comprobar el número máximo de usuarios que se pueden conectar de manera simultánea, se instalan cien clientes en el primer equipo, que introducen los datos de sus contadores cada minuto. Esto produce en el primer equipo una necesidad de ancho de banda de salida ligeramente superior a los 230 kBps, lo mismo que necesita el segundo equipo en bajada para recibir esos datos.

En esta prueba, con cien usuarios insertando datos de manera simultánea, se deforma notablemente la forma de onda de dos picos esperada y que se apreciaba en las pruebas anteriores (con uno y con diez usuarios insertando datos simultáneamente), y se produce un retardo considerable, pues la inserción de los bloques correspondientes termina a los 35 segundos, mientras que, con diez usuarios, finalizaba tras 10 segundos desde la propuesta enviada por los clientes.

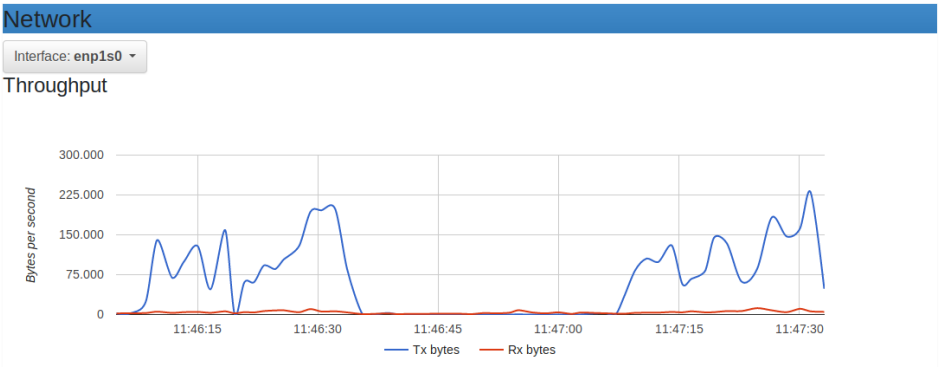


(a) Equipo 1.

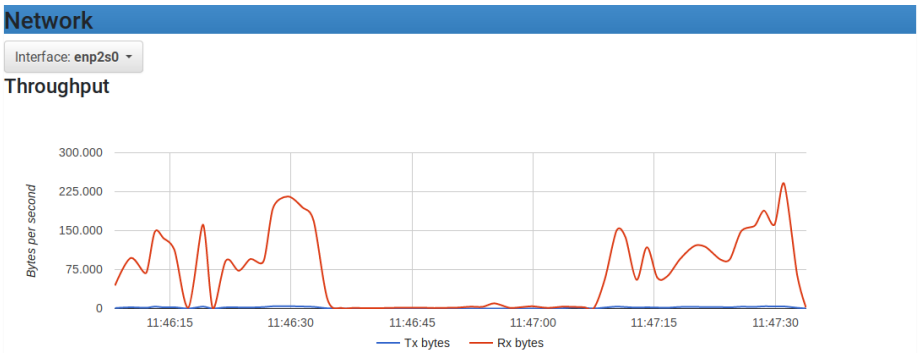


(b) Equipo 2.

Figura 5.8: Análisis del ancho de banda necesario para la inserción de los datos de los contadores de diez usuarios en la cadena de bloques, estando estos instalados en el primer equipo.



(a) Equipo 1.



(b) Equipo 2.

Figura 5.9: Análisis del ancho de banda necesario para la inserción de los datos de los contadores de cien usuarios en la cadena de bloques, estando estos instalados en el primer equipo.

La deformación de la onda del ancho de banda necesario no es debida a retrasos por limitación del ancho de banda, pues se tiene disponible una red con 500 MBps, cantidad muy superior a la necesaria.

En esta primera prueba con dos equipos, se ha podido apreciar, como se esperaba, que los datos enviados por el primer equipo son los recibidos por el segundo, debido a que las transmisiones de datos con la autoridad certificadora y el ordenante se realizan de manera interna en el primer equipo.

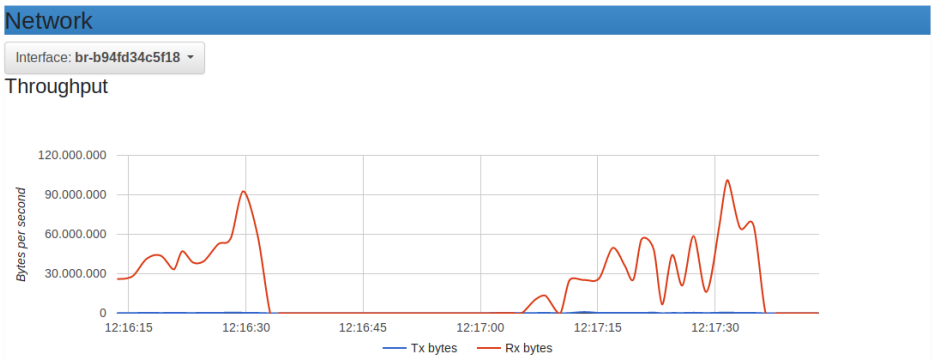


Figura 5.10: Análisis de los datos transmitidos entre los puertos del equipo que aloja la autoridad certificadora, el ordenante y dos nodos, con cien usuarios instalados insertando los datos de sus contadores.

Se consulta, con el objetivo de encontrar la causa de los retardos, el ancho de banda empleado en la transmisión de información entre puertos dentro del equipo (5.10), y en este caso se pueden apreciar picos de aproximadamente 100 MBps, que es una cantidad significativa, pero dentro del propio ordenador no debería repercutir en ningún tipo de retraso.

Además, se comprueba el uso de la CPU del ordenador (5.11), y se puede observar que esta se acerca a su máxima capacidad cada vez que los cien usuarios insertan los datos en la cadena de bloques, por lo que se puede determinar como la causa principal de los retardos.

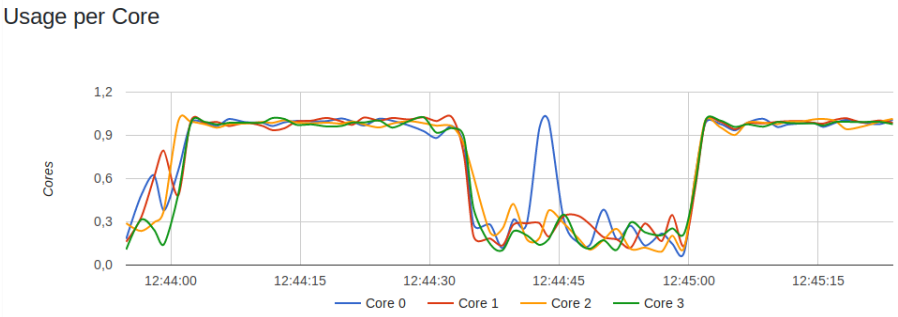


Figura 5.11: Uso de los procesadores del primer equipo con cien usuarios instalados en él.

Aunque ha habido retardos en la inclusión de los resultados en la cadena de bloques, al consultar los datos que se encuentran disponibles en las bases de datos de ambos ordenadores, se puede comprobar que están totalmente sincronizadas, por lo tanto, se puede concluir que, se ha obtenido un resultado satisfactorio en la comprobación de la sincronización de las bases de datos en todos los nodos de la red.

5.6 Prueba de funcionamiento con el gestor del sistema y prosumidores en equipos diferentes

Al alcanzarse, en el ordenador con los clientes de los prosumidores activos, el límite de capacidad de procesamiento en los cuatro núcleos del procesador, se asignan los retrasos en la inclusión de los bloques en la cadena a esta causa, y se procede a realizar una prueba más cercana a la realidad, en la que los usuarios están en otro equipo diferente al del gestor del sistema.

Este esquema sería escalable, replicando el segundo equipo tantas veces como prosumidores quieran unirse a la red *blockchain*, pero para realizar esta prueba, al no disponer de un gran número de equipos, se opta por instalar una cantidad de usuarios creciente en el segundo equipo, que aloja, además, un nodo de la red, es decir, se tiene la misma estructura de equipos conectados a la red *blockchain* que en la prueba anterior (5.6), con la diferencia de que en este caso, los clientes de los usuarios prosumidores se encuentran activos en el segundo equipo.

En esta prueba se realizan las mismas comprobaciones que en la anterior, con el objetivo de observar las diferencias entre tener los clientes de los usuarios insertando los datos en la cadena de bloques en el mismo equipo en el que se encuentran alojados la autoridad certificadora y el ordenante, y tenerlos en otro equipo diferente.

A priori, las diferencias se van a encontrar en la cantidad de datos que se deben transmitir entre ambos equipos, necesitándose un ancho de banda muy superior al que resultó necesario en la prueba anterior (se necesitó un máximo de 230 kBps cuando cien usuarios estaban insertando datos simultáneamente), ya que los intercambios de información con el ordenante y la autoridad certificadora ya no se realizan dentro del mismo equipo, y deben transmitirse por la red.

5.6.1 Análisis de un usuario

Al tener un único usuario insertando datos en la cadena de bloques con el cliente en el segundo equipo, se pueden apreciar diferencias.

La principal, es la altura de los picos, es decir, el ancho de banda necesario para introducir los datos en la cadena de bloques ha aumentado significativamente, mientras antes se transmitían picos de 15 kBps desde el primer equipo al segundo, ahora esta transmisión llega a ser de algo más de 75 kBps. La transmisión de datos del segundo equipo al primero también se ha visto aumentada hasta una velocidad de aproximadamente 12 kBps desde los 4 kBps de la prueba anterior.

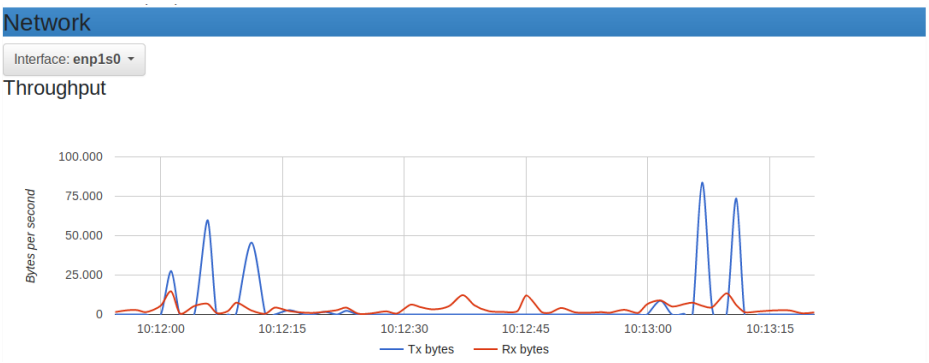
Además, en el segundo equipo, aparece el mismo primer pico que aparecía antes en el primer equipo, que se corresponde con una petición a la cadena de bloques de la variable a actualizar.

5.6.2 Análisis de diez usuarios

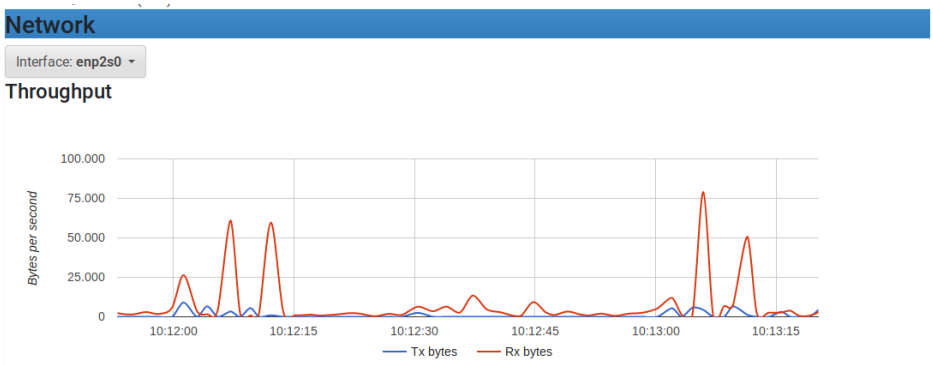
Si se aumenta a diez el número de usuarios insertando datos de manera simultánea, el ancho de banda necesario aumenta significativamente.

En el primer equipo se necesita un ancho de banda de subida de datos de aproximadamente 4,5 MBps, mientras que el de bajada resulta inapreciable al ser muy inferior.

En el segundo equipo se necesita un ancho de banda de 3 MBps de bajada, y en este caso es el de subida de datos el que resulta muy inferior y por tanto inapreciable.

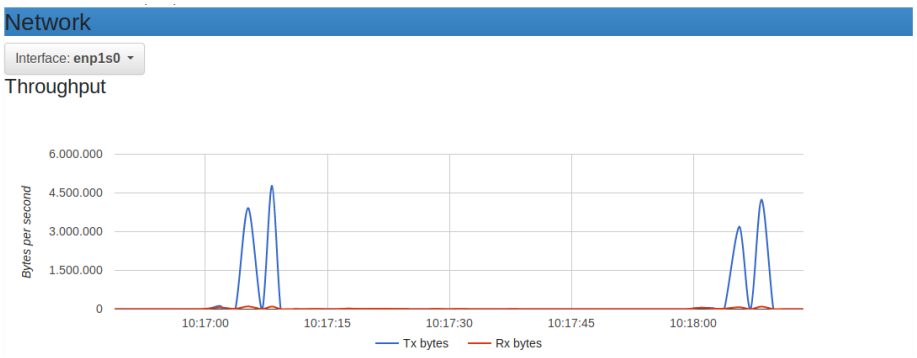


(a) Equipo 1.

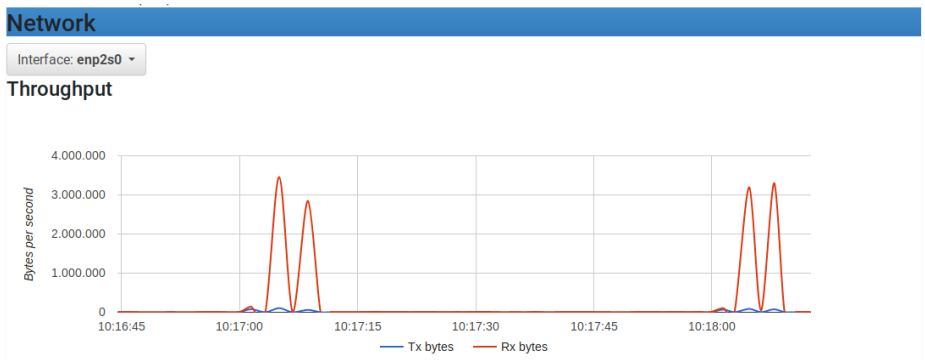


(b) Equipo 2.

Figura 5.12: Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando el usuario instalado en el equipo 2.

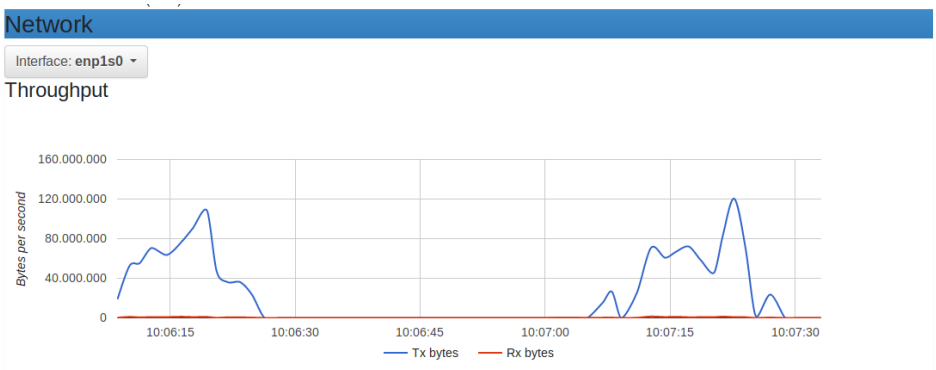


(a) Equipo 1.

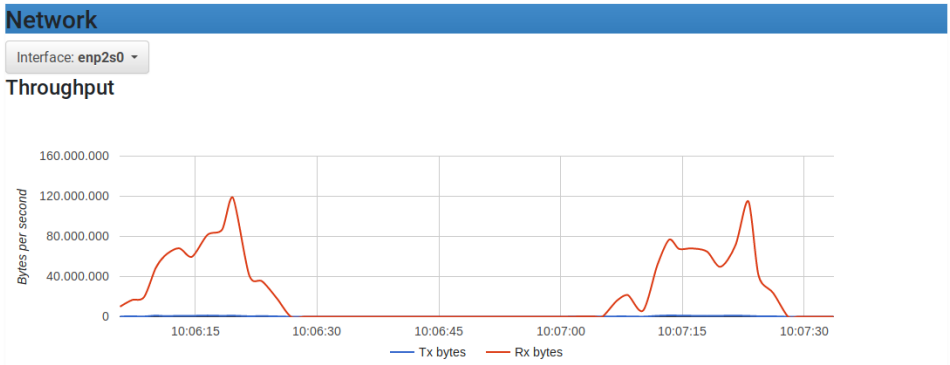


(b) Equipo 2.

Figura 5.13: Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando diez usuarios instalados en el equipo 2.



(a) Equipo 1.



(b) Equipo 2.

Figura 5.14: Análisis del ancho de banda necesario para la inserción de los datos del contador en la cadena de bloques, estando cien usuarios instalados en el equipo 2.

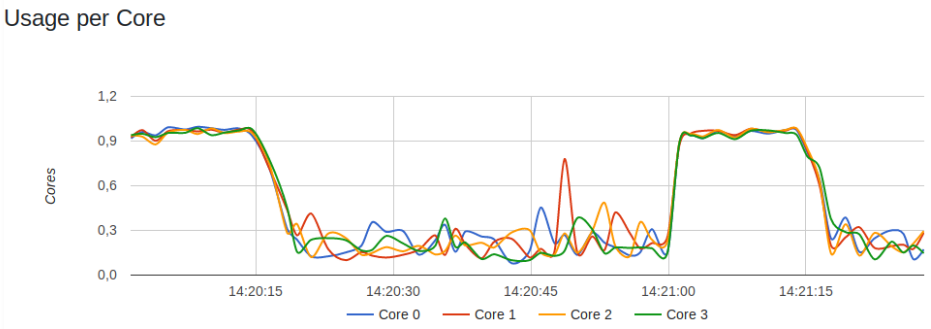
Ahora el primer pico resulta inapreciable en ambos equipos, quedando una forma de onda perfectamente definida de dos picos en ambos equipos, el primero transmitiendo y el segundo recibiendo los datos, y que finaliza a los diez segundos de iniciarse la transacción.

5.6.3 Análisis de cien usuarios

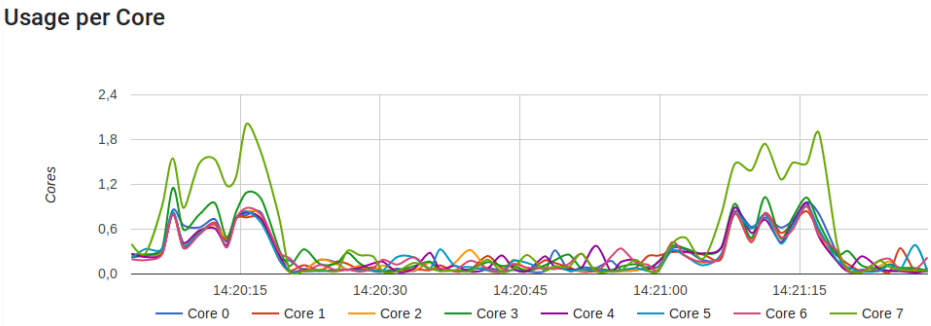
Al tener cien usuarios insertando los datos de sus contadores en el segundo equipo, la forma de onda queda totalmente deformada, al igual que en la prueba realizada con los cien usuarios insertando los datos en el primer equipo, pero con la diferencia de que, en este caso, el ancho de banda necesario es muy superior, llegando a ser necesarios 120 MBps de subida en el primer equipo y de bajada en el segundo.

Esta deformación de la curva de ancho de banda necesario repercute en retrasos en la inserción de los datos de los contadores en la cadena de bloques, pero no supondría un problema para el correcto funcionamiento del sistema con un número de usuarios similar. Ahora bien, si se aumentara el número de usuarios insertando datos de manera simultánea, el retardo sería mayor, y podría llegar a darse el caso de que los datos no están actualizados para la próxima vez que se solicitan para actualizarlos, lo que repercutiría en rechazos a los usuarios por parte de la cadena de bloques a la inserción de datos por conflictos de versiones, por lo tanto, es importante localizar la causa de estos retrasos, para comprobar si en una situación real, con todos los usuarios prosumidores insertando los datos de sus contadores, cada uno desde su equipo, podría funcionar realmente el sistema o sería inviable.

Como en la prueba anterior se determinó que la causa de la deformación de la curva de ancho de banda fue debido a que se alcanzó el máximo de capacidad de computación en el primer equipo, se realiza esta misma comprobación en este caso en ambos equipos, para comprobar si el primer equipo ha conseguido



(a) Equipo 1.



(b) Equipo 2.

Figura 5.15: Análisis del uso de capacidad de computación de ambos equipos, estando cien usuarios instalados en el equipo 2.

desahogar su capacidad de procesamiento al ser el segundo equipo el que inserta los datos, aunque tenga que ser este el que compruebe la autenticidad de las firmas y tenga en su interior al ordenante.

Se puede apreciar en la figura 5.15a que el primer equipo, cada vez que llega el momento de que los usuarios inserten sus datos, lleva su capacidad de procesamiento hasta el máximo, pero en este caso, en lugar de permanecer en esta situación durante 30 segundos como en la prueba anterior, permanece la mitad de tiempo, aproximadamente 15 segundos. El segundo equipo (5.15b), por el contrario, no se alcanza el máximo de su capacidad de procesamiento, quedando varios núcleos de su procesador en todo momento por debajo del 100 % de capacidad, aún tiene margen para ejecutar más clientes que inserten datos en la cadena de bloques.

Por lo tanto, se puede concluir esta prueba determinando que el cuello de botella y la causa de los retrasos en la inclusión de los datos en la cadena de bloques, se encuentra en la capacidad de procesamiento del equipo del gestor del sistema, pues no tiene potencia suficiente como para ejecutar de manera instantánea los procesos correspondientes a la autoridad certificadora y al ordenante de transacciones. Para un uso final correcto del sistema, el ordenador que aloje el nodo principal con la autoridad certificadora y el ordenante, debe tener una capacidad de procesamiento acorde al número de usuarios que vayan a participar en el sistema de intercambios de energía de la microrred.

Además, el ancho de banda de subida de datos del equipo con la autoridad certificadora y el ordenante, debe ser considerablemente grande, pues en las pruebas realizadas se alcanzan los 120 MBps para cien usuarios, si se tuviera un equipo con mayor capacidad de procesamiento que el actual, podría admitirse un mayor número de usuarios, lo que haría necesario un ancho de banda superior a los 120 MBps para que el cuello de botella del sistema no se encontrara en la velocidad de transmisión de los datos a través de la red.

A partir de este punto, conocidas las limitaciones del equipo en el que están instalados el ordenante

y la autoridad certificadora, el resto de pruebas se realizan con un máximo de 50 usuarios.

5.7 Prueba de funcionamiento de un gran número de usuarios prosumidores, durante varias rondas de mercado en dos equipos

Esta prueba tiene como objetivo la comprobación del correcto funcionamiento del mercado minorista de energía durante un gran número de rondas consecutivas, con una cantidad de usuarios suficientemente grande instalado en un equipo diferente al del gestor del sistema, y comprobar los recursos consumidos tanto en el equipo del gestor del sistema como una estimación de los recursos de los equipos que necesitarán los usuarios prosumidores.

Al ejecutar una ronda de mercado, los usuarios prosumidores tardan aproximadamente 7 segundos en detectar que han sido admitidos en la subasta de energía y enviar su oferta, y posteriormente, hasta que la autoridad certificadora y el ordenante finalizan sus funciones, se tarda aproximadamente 10 segundos más en insertar esos datos en la cadena de bloques (debido principalmente a la limitada capacidad de computación del ordenador empleado en las pruebas como gestor del sistema), por lo tanto, resulta inviable emplear un tiempo de periodo de intercambios tan pequeño como el que se había establecido para las pruebas anteriores, y se decide ampliarlo a 5 minutos, tiempo suficientemente pequeño como para realizar un gran número de rondas de mercado en un tiempo no demasiado extenso.

Una vez instalados los usuarios prosumidores en el segundo equipo, se inicia el cliente del gestor del sistema, que crea las rondas de subastas del mercado interno de la microrred. El sistema funciona correctamente sin ningún tipo de error durante varias horas, sacando por el terminal de Linux las fases que va realizando (5.16). Se puede comprobar que, la fase previa, finaliza en 38 segundos, es decir, los usuarios prosumidores tienen acceso a los resultados del mercado en los que son parte interesada con algo más de un minuto de antelación al inicio de la ronda de intercambios de energía, y así adaptar su producción o consumo para ajustarse a dicho resultado. La fase posterior, se inicia un minuto después de finalizar el periodo de intercambios, para dar tiempo a los usuarios a que inserten los datos de sus contadores, y tarda aproximadamente 25 segundos en ejecutar los pagos correspondientes a los consumos y generaciones reales que se han producido durante dicho periodo.

Se realiza un análisis de los recursos consumidos tanto en la fase previa como en la fase posterior, resultando de especial relevancia el consumo de ancho de banda, capacidad de computación y cantidad de almacenamiento requerido por la base de datos.

5.7.1 Ancho de banda

En la fase previa a los intercambios de energía se han obtenido los resultados que se pueden observar en las figuras 5.17a y 5.17b. El ancho de banda de subida en el equipo del gestor del sistema alcanza un pico de cerca de 100 MBps en el momento de insertar los usuarios sus ofertas para el mercado, mientras que la bajada en el equipo de los usuarios prosumidores, que también se produce en el momento de introducir los prosumidores sus ofertas, supera ligeramente los 75 MBps. De manera posterior, el ancho de banda empleado se encuentra en todo momento por debajo de los 10 MBps el requerimiento de banda ancha en ambos equipos.

La inserción de los datos de las ofertas realizadas por los prosumidores provoca una mayor necesidad de ancho de banda por estar estos funcionando en un equipo diferente al de la autoridad certificadora y

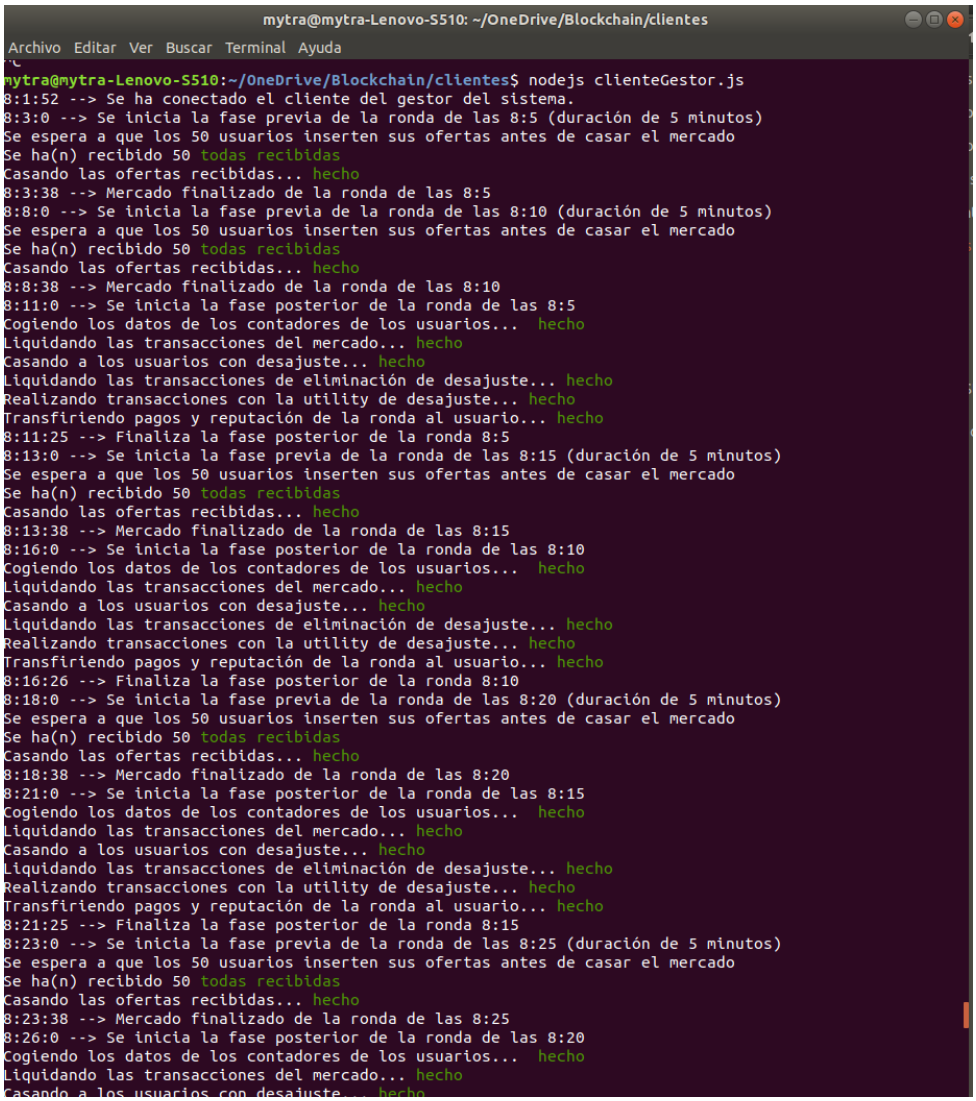


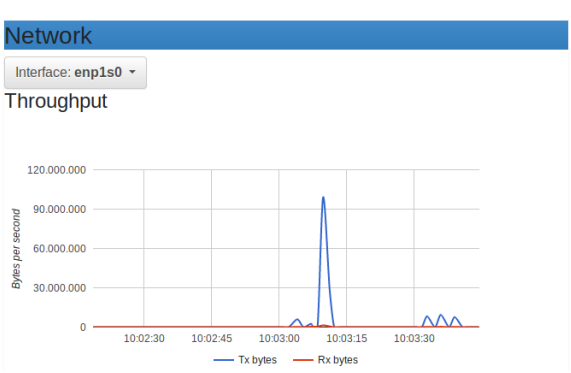
Figura 5.16: Salida de la terminal Linux del equipo gestor del sistema ejecutando rondas de mercado cada 5 minutos.

el ordenante, y no ocurre lo mismo con las transacciones que ejecuta el gestor del sistema.

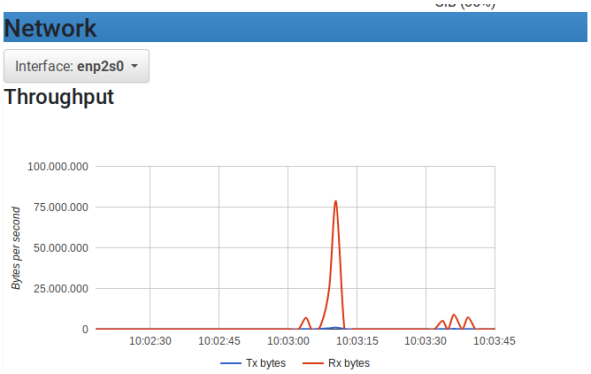
En la fase posterior, al ser todas las transacciones ejecutadas por el gestor del sistema, se requiere un menor ancho de banda, teniendo picos de subida en el equipo del gestor del sistema de aproximadamente 16 MBps (5.17c) y picos de bajada en el equipo de los prosumidores (5.17d) de 12 MBps.

Por lo tanto, observando los resultados del ancho de banda que se ha necesitado en todas las fases de mercado, el proceso que más ancho de banda exige es la inserción de las ofertas por parte de los prosumidores, con 100 MBps de subida para el equipo del gestor del sistema. Esta cifra no supone un problema teniendo en cuenta la conexión con la que se están haciendo las pruebas, pues se dispone de 500 MBps tanto de subida como de bajada, pero debe tenerse en cuenta a la hora de ampliar el número de usuarios o de emplear una conexión de menor velocidad.

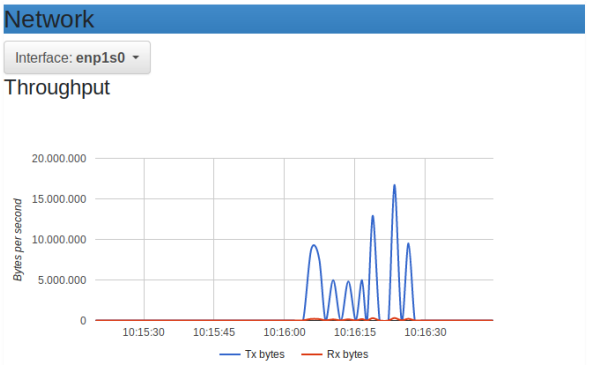
Si no se dispusiera de una conexión con el ancho de banda necesario para introducir las ofertas de los prosumidores en la cadena de bloques suficientemente rápido, esto podría repercutir en errores en el funcionamiento del sistema, pues el gestor podría dar la orden de casar las ofertas antes de que estas hayan sido incluidas de manera definitiva en la cadena de bloques.



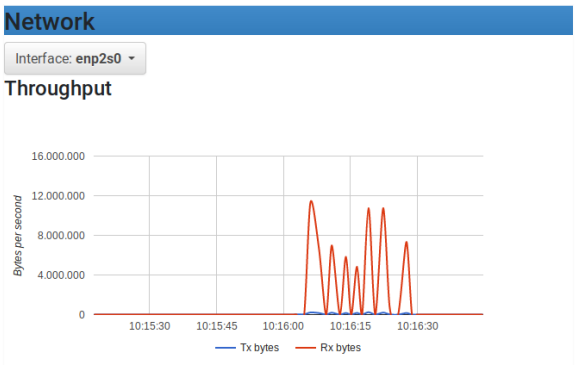
(a) Fase previa equipo del gestor del sistema.



(b) Fase previa equipo de los prosumidores.



(c) Fase posterior equipo del gestor del sistema.



(d) Fase posterior equipo de los prosumidores.

Figura 5.17: Análisis del ancho de banda necesario durante la ejecución de la fase previa y la fase posterior de un periodo de intercambios de energía.

5.7.2 Capacidad de computación

La capacidad de computación (5.18), al igual que el ancho de banda necesario, alcanza su máximo pico durante la fase previa, en el momento de inserción de las ofertas por parte de los usuarios prosumidores. Este pico, tanto en el equipo del gestor del sistema (5.18a), como en el de los usuarios prosumidores (5.18b), alcanza el límite de la capacidad, por lo tanto, un número mayor de usuarios repercutiría en un retraso mayor en la inclusión de las ofertas en la cadena de bloques.

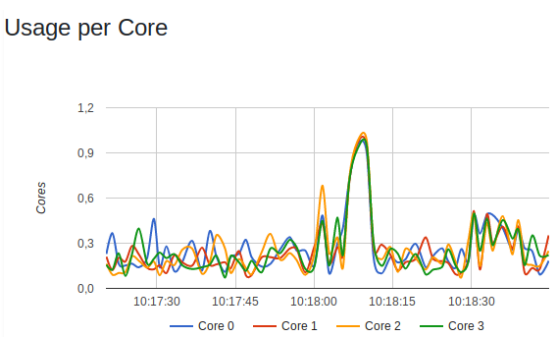
Tanto en el resto de la fase previa como en la fase posterior a los intercambios de energía (5.18c y 5.18d), la capacidad de computación no es limitante en ningún otro proceso, quedándose todos los núcleos de ambos equipos en todo momento por debajo del 75 % de su capacidad.

Cabe destacar que, aunque la capacidad de computación empleada alcanza el máximo de la disponible en el equipo de los usuarios prosumidores, este ordenador tiene ejecutándose un proceso en paralelo por cada uno de los usuarios prosumidores simulados, por tanto, un equipo con una capacidad de computación 50 veces inferior sería suficiente para ejecutar el proceso de inserción de una oferta en la cadena de bloques.

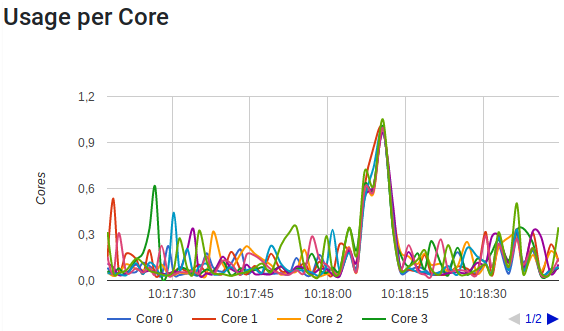
5.7.3 Capacidad de almacenamiento

Si se analizan las variaciones en la cantidad de datos almacenados en la base de datos mediante las cuatro capturas realizadas en diferentes instantes de tiempo de la figura 5.19, se puede extraer la tabla 5.4, a partir de la cual, se puede hacer una previsión de cual será el almacenamiento necesario para que el sistema pueda permanecer en funcionamiento durante un largo periodo de tiempo.

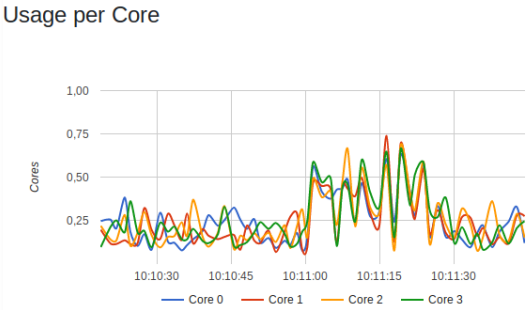
Sacando como conclusión de los datos de la tabla 5.4 que cada hora el tamaño total de las bases de



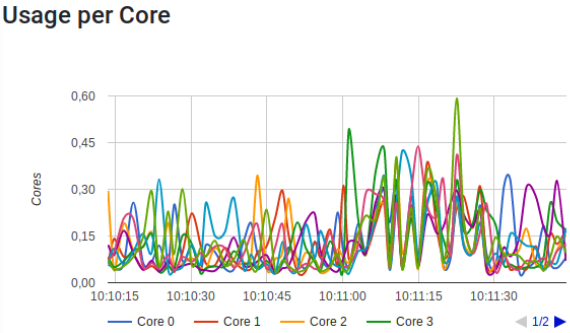
(a) Fase previa equipo del gestor del sistema.



(b) Fase previa equipo de los prosumidores.



(c) Fase posterior equipo del gestor del sistema.



(d) Fase posterior equipo de los prosumidores

Figura 5.18: Análisis de la capacidad de computación empleada durante la ejecución de las fases previa y posterior al periodo de intercambios de energía.

Name	Size	# of Docs
_global_changes	61.9 KB	15
_replicator	2.3 KB	1
_users	2.1 KB	1
composerchannel_	30.0 KB	2
composerchannel_creacion	0.6 MB	588
composerchannel_liquidacion	400.2 KB	493
composerchannel_lscc	2.3 KB	4
composerchannel_mercado	0.8 MB	1043
composerchannel_registro	1.7 MB	2050

(a) Al inicio.

Name	Size	# of Docs
_global_changes	88.1 KB	15
_replicator	2.3 KB	1
_users	2.1 KB	1
composerchannel_	43.7 KB	2
composerchannel_creacion	0.8 MB	658
composerchannel_liquidacion	1.1 MB	1543
composerchannel_lscc	2.3 KB	4
composerchannel_mercado	2.1 MB	2651
composerchannel_registro	3.7 MB	4450

(b) Tras una hora en funcionamiento.

Name	Size	# of Docs
_global_changes	106.9 KB	15
_replicator	2.3 KB	1
_users	2.1 KB	1
composerchannel_	42.3 KB	2
composerchannel_creacion	0.9 MB	746
composerchannel_liquidacion	1.8 MB	2915
composerchannel_lscc	2.3 KB	4
composerchannel_mercado	3.5 MB	4661
composerchannel_registro	5.8 MB	7250

(c) Tras dos horas en funcionamiento.

Name	Size	# of Docs
_global_changes	132.9 KB	15
_replicator	2.3 KB	1
_users	2.1 KB	1
composerchannel_	42.3 KB	2
composerchannel_creacion	1.1 MB	884
composerchannel_liquidacion	3.1 MB	5169
composerchannel_lscc	2.3 KB	4
composerchannel_mercado	5.7 MB	7743
composerchannel_registro	9.4 MB	11850

(d) Tras cuatro horas en funcionamiento.

Figura 5.19: Análisis de la capacidad de almacenamiento necesaria para alojar la base de datos.

Cadena de bloques	Variación cada hora (MB)	Variación cada ronda (KB)
Datos personales	0,1 - 0,2	16 - 33
Mercado	1,1 - 1,4	90 - 100
Registro parámetros	1,9 - 2.1	160 - 175
Liquidación de pagos	0,6 - 0,7	50 - 60
Total	**3,7 - 4,4**	**316 - 368**

Tabla 5.4: Variaciones en la cantidad de datos almacenados en las bases de datos de las cadenas de bloques

datos del sistema aumentan hasta 4,4 MB, para un funcionamiento en las mismas condiciones (50 usuarios con rondas de mercado cada 5 minutos) prolongado a lo largo de un año y suponiendo un crecimiento lineal, se necesitaría una capacidad de almacenamiento de 37,64 GB.

Esta necesidad de capacidad de almacenamiento se daría en cada uno de los nodos de la red *blockchain*, y si bien en el equipo del gestor del sistema no va a ser ningún problema, pues va a ser un equipo con unas buenas especificaciones técnicas, entre las que la capacidad de almacenamiento no va a suponer un incremento relevante en el precio del ordenador, en los equipos de los usuarios prosumidores si puede suponer un problema pues una coste elevado en la adquisición del dispositivo puede suponer la intención de no querer participar por parte del usuario.

Como solución a los problemas que supondría una necesidad elevada de capacidad de almacenamiento de datos, se podría no replicar la base de datos en cada uno de los nodos de los usuarios prosumidores, únicamente replicarla en la de un número mínimo para que se garantice el correcto funcionamiento de la plataforma aunque se perdieran los datos del equipo del gestor del sistema, ya fuese por un fallo técnico del equipo o por un borrado malintencionado.

Capítulo 6

Conclusiones y líneas futuras

En este capítulo se resumen las conclusiones obtenidas durante la realización de este trabajo fin de máster en la sección 6.1 y, a continuación, se enumeran las posibles líneas sobre las que se puede continuar avanzando para conseguir aportar el máximo potencial de la tecnología *blockchain* a la red eléctrica, en la sección 6.2.

6.1 Conclusiones

La primera y más importante conclusión que se puede extraer de la realización del presente trabajo es que, en el proyecto realizado, no se ha encontrado ningún elemento del que se derive la imposibilidad de aplicación de la tecnología *blockchain* en la creación de una plataforma de intercambios de energía, que aportaría numerosas ventajas en la gestión de las microrredes eléctricas, al delegar la responsabilidad de la gestión de la energía a los usuarios finales, que buscan un interés económico o medioambiental propio y autorregulan su consumo y generación de energía en función de los resultados del mercado.

Además, la plataforma de intercambios de energía, tendría la capacidad de aplicarse no solo dentro de una microrred, podría emplearse también, para gestionar los intercambios energéticos entre varias microrredes, sin necesidad de realizar cambios significativos en ella, únicamente dando de alta a cada uno de los gestores de las microrredes como usuarios prosumidores que van a participar en la subasta de energía, para realizar intercambios entre ellos.

Cualquiera de los algoritmos de consenso existentes, no sólo en la plataforma de desarrollo empleada (*Hyperledger Fabric*), únicamente tienen sentido cuando existe más de un nodo con la cadena de bloques completa almacenada. Si se creara una plataforma basada en la tecnología *blockchain*, que solo tuviera un nodo con la base de datos completa almacenada, este nodo podría ser modificado, y la tecnología de cadena de bloques no protegería de manera superior a un cifrado tradicional, pues no hay otros nodos con los que comparar los datos almacenados. El número mínimo de nodos con la base de datos almacenada, para un funcionamiento seguro con uno de los algoritmos de consenso a prueba de fallos, es tres, de forma que, aunque uno de los nodos haya resultado modificado, la mayoría de nodos tengan aún los datos correctos. En la plataforma de intercambios de energía desarrollada, estos nodos con la base de datos completa almacenada en su interior, han de instalarse en más de un equipo. Si bien, la situación más segura sería que cada uno de los usuarios prosumidores de la microrred tuviese en su equipo la base de datos completa, esto requeriría de una capacidad de almacenamiento significativa (5.7.3), lo cual puede repercutir en un coste más elevado de los equipos necesarios, lo que haría más difícil la extensión del sistema. Por lo tanto, se propone que los usuarios prosumidores únicamente se conecten como clientes a

la cadena de bloques, sin tenerla almacenada en sus equipos, y que sea el gestor del sistema el encargado de distribuir varios equipos por la microrred que sí tengan la cadena de bloques almacenada. Los usuarios prosumidores que deseasen tener una copia de la base de datos para estar más seguros de que los datos no son modificados, podrían hacerlo sin ningún problema, corriendo ellos con el coste del dispositivo de almacenamiento.

La inversión por parte de los prosumidores que quisieran únicamente unirse a la plataforma de intercambios, sin capacidad de almacenamiento para guardar la base de datos, sería muy pequeña, al presuponerse que se dispone de conexión a Internet, únicamente debería instalarse un equipo de pequeñas prestaciones, tipo Raspeberry Pi o similar, que contuviera los programas desarrollados y la tarjeta de identificación del usuario, con sus claves públicas y privadas. La inversión en el equipo necesario para conectarse a la plataforma de intercambios de energía, al igual que la realizada en el sistema de generación renovable o, en su caso, la batería, se ve compensada por unos mejores precios de la electricidad (tanto de compra como de venta) que se obtienen a partir del momento de su instalación.

La inversión que debe realizar el gestor del sistema es la de adquirir, al menos, tres equipos de altas prestaciones, que validen todas las transacciones que posteriormente van a ser almacenadas en la cadena de bloques y además, tengan capacidad de almacenamiento suficiente como para guardar dicha cadena de bloques (siendo esta siempre creciente). El gestor del sistema podría ser una cooperativa de clientes prosumidores, por lo que la inversión les va a retornar al obtener mejores precios en la electricidad y una mayor independencia de la red eléctrica, un ente público (ayuntamiento o similar), que trataría de mejorar la calidad de vida de los ciudadanos, que no tendría intención de recuperar la inversión, o una empresa, que trataría de monetizar la inversión en los equipos bien mediante una pequeña comisión de los intercambios que gestiona o bien mediante una cuota mensual fija a los usuarios prosumidores por poder unirse a la plataforma de intercambios. En cualquier caso, la inversión resulta asequible y con unos beneficios notablemente superiores al coste monetario inicial.

6.2 Líneas futuras

A lo largo del desarrollo de este proyecto, se han detectado carencias y posibilidades de mejora en la plataforma de intercambios de energía. A continuación, se enumeran las posibles líneas de trabajo que se pueden seguir para continuar con este proyecto y mejorarlo hasta hacerlo viable en entornos reales y que aporte el mayor potencial posible:

- Pasar la plataforma de intercambios a la nueva versión 1.4 de *Hyperledger Fabric*, sacada a la luz en septiembre de 2019 y que tiene la ventaja más relevante de la inclusión del algoritmo de consenso *Raft* (2.5.3), que haría la aplicación viable para su uso en escenarios reales, puesto que la versión en la que está implementada actualmente no soporta ningún algoritmo de consenso con una implementación sencilla (el que está implementado en la plataforma de intercambios desarrollada es un algoritmo de consenso para desarrollo, no viable para entornos reales).

- Integrar un sistema de gestión de la energía dentro de la red *blockchain*, para mejorar la fluidez del sistema completo y evitar manipulaciones de datos.

- Investigar la posibilidad de emplear datos de cadenas de datos diferentes en los contratos inteligentes, para evitar tener que leer los datos de una cadena con el cliente del gestor del sistema para introducirlos en la otra cadena y poder ejecutar el contrato inteligente. Si se consiguiera emplear datos entre cadenas de bloques se mejoraría la fluidez del sistema y se aumentaría su resistencia a posibles manipulaciones al evitar al máximo posible que los datos estén fuera de la red *blockchain*.

- Implementar un algoritmo de actualización de la reputación de los prosumidores en función de sus interacciones con el sistema, que sea lo suficientemente justo y evite participar a los usuarios que actúan de forma malintencionada, en beneficio propio o buscando perjudicar al resto de los participantes.
- Implementar un algoritmo para el casado de ofertas más complejo, que tenga en cuenta, además de la cantidad de energía como se hace en ela algoritmo actualmente implementado, una curva de precios en función de la cantidad, la fuente de generación, el precio al que se ofrece capacidad de absorción de desajustes, etc.
- Integrar de manera física con un contador inteligente. Se ha dejado el sistema preparado para leer un archivo que el contador inteligente deje en una ubicación determinada del equipo, por lo que no debería suponer un gran esfuerzo.
- Realización de pruebas del sistema completo en un entorno real con varios prosumidores intercambiando energía dentro de una microrred.
- Los usuarios prosumidores necesitan una aplicación, que pueda ser visualizada a través de un dispositivo por pantalla, en la que se puedan consultar el número de *tokens* que tiene disponibles, su reputación, cómo está mejorando su factura de electricidad el estar participando en la plataforma de intercambios de energía, de qué fuentes está adquiriendo la electricidad que consume, etc.
- El usuario prosumidor debe disponer de una aplicación con la que vender o comprar *tokens* al gestor del sistema a cambio de dinero real.

Los programas para uso por parte del cliente final, deberían tener un aspecto visual atractivo y ser muy sencillas de usar, para que se consiga una buena experiencia de uso de la plataforma, pues aunque los objetivos principales del sistema son las mejoras medioambientales, una mayor resiliencia de la microrred y el ahorro económico que obtendrían los participantes, la única comunicación entre el usuario prosumidor y la plataforma de intercambios va a ser a través de esta aplicación y va a condicionar notablemente su percepción.

Si la experiencia de uso no fuese óptima, esto supondría un obstáculo a la expansión del sistema.

Además, con el objetivo de mejorar la seguridad, se han tratado de buscar los puntos más débiles del sistema. Estos puntos se encuentran en las comunicaciones entre diferentes dispositivos, para ello, se proponen varias mejorar con el objetivo de minimizar los riesgos:

1. La comunicación entre el sistema de gestión de la energía y la red *blockchain* podría eliminarse si se consiguiera integrar un sistema de gestión de la energía en forma de contrato inteligente, eliminando cualquier posibilidad de manipulación de los datos durante la transmisión.
2. La parte más sensible de todo el sistema reside en la transmisión de los parámetros eléctricos a la plataforma, pues en caso de que la comunicación entre el sistema de medición de parámetros eléctricos (contador inteligente u otros métodos) y el equipo que se conecta a la red *blockchain* no sea suficientemente segura, esta información puede ser manipulada por las partes interesadas, por ejemplo para fingir consumir menos de lo real, o por terceros que busquen una caída del sistema. Para subsanar este punto débil en la seguridad e integridad de los datos de la plataforma de intercambios, se proponen dos alternativas:
    - Introducir los parámetros eléctricos con la mayor frecuencia posible, pues al reducir el tiempo entre introducciones de datos, se reduce la posibilidad de manipulación. Esto tendría una repercusión en las necesidades de almacenamiento de la base de datos.

- Integrar un contador inteligente dentro del mismo dispositivo que emplea cada prosumidor para conectarse a la red *blockchain*. Esto repercutiría en un mayor coste de inversión, pues habría que sustituir el contador del usuario.

Bibliografía

[1] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and Sustainable Energy Reviews*, vol. 100, pp. 143 – 174, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032118307184

[2] R. K. Kodali, S. Yerroju, and B. Y. K. Yogi, "Blockchain based energy trading," in *TENCON 2018 - 2018 IEEE Region 10 Conference*, Oct 2018, pp. 1778–1783.

[3] R. Eléctrica, "El suministro de la electricidad. un equilibrio entre generación y consumo." [Online]. Available: https://www.ree.es/es/publicaciones/educacion/el-suministro-de-la-electricidad-un-equilibrio-entre-generación-y-consumo

[4] J. Pastor, "Qué es blockchain: la explicación definitiva para la tecnología más de moda." [Online]. Available: https://www.xataka.com/especiales/que-es-blockchain-la-explicacion-definitiva-para-la-tecnologia-mas-de-moda

[5] Bit2Me, "Qué es el hash." [Online]. Available: https://academy.bit2me.com/que-es-hash/

[6] B. Donohue. ¿qué es un hash y cómo funciona? [Online]. Available: https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/

[7] Bit2Me, "Qué es el Árbol de merkle." [Online]. Available: https://academy.bit2me.com/que-es-arbol-de-merkle/

[8] eSMARTCITY. Micro redes eléctricas inteligentes. [Online]. Available: https://www.esmartcity.es/2011/03/08/micro-redes-electricas-inteligentes

[9] CENER, "Introducción a las microrredes." [Online]. Available: http://www.cener.com/introduccion-a-las-microrredes/

[10] E. SmartGrids, "Vision and strategy for europe's electricity networks of the future," *European Commission*, 2006.

[11] Competition and M. Authority., "Energy market investigation summary of final report." [Online]. Available: https://assets.publishing.service.gov.uk/media/5773de34e5274a0da3000113/final-report-energy-market-investigation.pdf

[12] Deloitte, "Blockchainenigma. paradox. opportunity." [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/uk/Documents/Innovation/deloitte-uk-blockchain-full-report.pdf

[13] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[14] P. S. Muftic, "Overview and analysis of the concept and applications of virtual currencies." [Online]. Available: http://publications.jrc.ec.europa.eu/repository/bitstream/JRC105207/lbna28386enn.pdf

[15] E. Foundation., "Ethereum blockchain app platform." [Online]. Available: https://www.ethereum.org/

[16] S. Peyrott, "An introduction to ethereum and smart contracts: a programmable blockchain." [Online]. Available: https://auth0.com/blog/an-introduction-to-ethereum-and-smart-contracts-part-2/

[17] J. Mattila, "The blockchain phenomenon–the disruptive potential of distributed consensus architectures," ETLA working papers, Tech. Rep., 2016.

[18] H. Fabric, "Introduction to hyperledger fabric." [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/blockchain.html

[19] wikipedia. Rsa. [Online]. Available: https://es.wikipedia.org/wiki/RSA

[20] B. Academy, "La historia de blockchain." [Online]. Available: https://www.binance.vision/es/blockchain/history-of-blockchain

[21] Hyperledger, "Hyperledger architecture, volume 1: Introduction to hyperledgerbusiness blockchain design philosophy and consensus." [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.

[22] N. Rodríguez, "Algoritmos de consenso: la raíz de la tecnología blockchain." [Online]. Available: https://101blockchains.com/es/algoritmos-de-consenso-blockchain/

[23] Álvaro Suárez, "Minería y algoritmos de consenso." [Online]. Available: https://aprendeblockchain.wordpress.com/fundamentos-tecnicos-de-blockchain/mineria-y-algoritmos-de-consenso/

[24] J. Maldonado, "Protocolos de consenso: Pilar en la seguridad blockchain." [Online]. Available: https://www.bitcobie.com/protocolos-de-consenso-pilar-en-la-seguridad-blockchain/

[25] K. Rilee, "Understanding hyperledger fabric: Byzantine fault tolerance." [Online]. Available: https://medium.com/kokster/understanding-hyperledger-fabric-byzantine-fault-tolerance-cf106146ef43

[26] E. Wiki, "On sharding blockchains." [Online]. Available: https://github.com/ethereum/wiki/wiki/Sharding-FAQ

[27] J. Bentke, "On-chain vs off-chain." [Online]. Available: https://energyweb.atlassian.net/wiki/spaces/EWF/pages/17760291/On-ChainvsOff-Chain

[28] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[29] A. Batabyal, "Hyperledger vs ethereum: Top 5 common differences you need to know." [Online]. Available: https://coinswitch.co/news/hyperledger-vs-ethereum-top-5-common-differences-you-need-to-know

[30] H. Fabric. Identity. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/identity/identity.html

[31] wikipedia. X.509. [Online]. Available: https://en.wikipedia.org/wiki/X.509

[32] R. Bairathi. Consensus & endorsement in hyperledger fabric. [Online]. Available: https://medium.com/coinmonks/consensus-endorsement-in-hyperledger-fabric-5dbf233b452c

[33] H. Fabric. The ordering service. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/orderer/ordering_service.html

[34] J. Zhang, "Consensus algorithms: Poa, ibft or raft?" [Online]. Available: https://kaleido.io/consensus-algorithms-poa-ibft-or-raft/

[35] S. P. Singh. Detail analysis of raft & its implementation in hyperledger fabric. [Online]. Available: https://medium.com/@spsingh559/detail-analysis-of-raft-its-implementation-in-hyperledger-fabric-d269367a79c0

[36] H. Fabric. Smart contracts. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/smartcontract/smartcontract.html

[37] Hyperledger, "Welcome to hyperledger composer." [Online]. Available: https://hyperledger.github.io/composer/v0.19/introduction/introduction.html

[38] IBM, "Aspectos básicos de hyperledger composer, parte 1: Modele y pruebe su red blockchain." [Online]. Available: https://www.ibm.com/developerworks/ssa/cloud/library/cl-model-test-your-blockchain-network-with-hyperledger-composer-playground/cl-model-test-your-blockchain-network-with-hyperledger-composer-playground-pdf.pdf

[39] Z. Li, S. Bahramirad, A. Paaso, M. Yan, and M. Shahidehpour, "Blockchain for decentralized transactive energy management system in networked microgrids," *The Electricity Journal*, vol. 32, no. 4, pp. 58–72, 2019.

[40] "Información sobre gnu/linux en wikipedia." [Online]. Available: http://es.wikipedia.org/wiki/GNU/Linux

[41] Interfaz de programación de aplicaciones. [Online]. Available: https://es.wikipedia.org/wiki/Interfaz_de_programación_de_aplicaciones

[42] rapberrypi. [Online]. Available: https://es.wikipedia.org/wiki/Raspberry_Pi

[43] R. E. de España. Sistema de información del operador del sistema. [Online]. Available: https://www.esios.ree.es/es

[44] Google. Github de cadvisor. [Online]. Available: https://github.com/google/cadvisor

[45] V. Raj. Setting up a blockchain business network with hyperledger fabric running in multiple physical machine. [Online]. Available: https://www.skcript.com/svr/setting-up-a-blockchain-business-network-with-hyperledger-fabric-and-composer-running-in-/multiple-physical-machine/

[46] A. Wahab. Extending hyperledger fabric network: Adding a new peer. [Online]. Available: https://medium.com/@wahabjawed/extending-hyperledger-fabric-network-adding-a-new-peer-4f52f70a7217

[47] Microsoft. Visual studio code (open source). [Online]. Available: https://code.visualstudio.com/

[48] L. Lamport, *LaTeX: A Document Preparation System, 2nd edition.* Addison Wesley Professional, 1994.

[49] wikipedia, "Node js." [Online]. Available: https://es.wikipedia.org/wiki/Node.js

Apéndice A

Listados de código

A.1 Código de las cadenas de bloques

A.1.1 Cadena de bloques de datos personales

Listado A.1: org.creacion.cto – Código de definición del modelo de la cadena de bloques que contiene los datos personales de los usuarios prosumidores (lenguaje CTO)

```
namespace org.creacion

participant Prosumer identified by ID{
  o String ID
  o String IDsecreto
  o String Nombre
  o String Apellidos
  o String Direccion
  o String TipoGeneracion
  o Double Coins
  o Double Reputacion
  o Boolean Apto
}

/*Comprueba si los usuarios tienen suficientes coins y reputacion como para ser admitidos en
    el mercado*/
transaction iniciarMercado{
  o Double coins
  o Double reputacion
}

/*Elimina la condicion de aptos a todos los usuarios*/
transaction finalizarMercado{}

/*Se crea un evento cuando se hayan seleccionado los usuarios aptos*/
event Evento{
  o Boolean iniciado1
}
```

Listado A.2: script.js – Código de las transacciones definidas en la cadena de bloques que contiene los datos personales de los usuarios prosumidores (lenguaje node.js)

```
'use strict';
/**
 * Comprueba que usuarios tienen suficientes Coins para depositar la fianza y se les retira
 * @param {org.creacion.iniciarMercado} update - los datos a ser actualizados
 * @transaction
 */
async function iniciarMercado(update) {
    let participantRegistry = await getParticipantRegistry('org.creacion.Prosumer');
    let results = await query('comprobarDinero',{Coins : update.coins });
    for (let n = 0; n < results.length; n++) {
        let prosumer = results[n];
        if (prosumer.Reputacion>update.reputacion){
            prosumer.Apto = true;
          await participantRegistry.update(prosumer);
      }
    }
    factory = await getFactory();
    var Evento = await factory.newEvent('org.creacion', 'Evento');
  Evento.iniciado1 = true;
    emit(Evento);
}

/**
 * Comprueba que usuarios tienen suficientes Coins para depositar la fianza y se les retira
 * @param {org.creacion.finalizarMercado} update - los datos a ser actualizados
 * @transaction
 */
async function finalizarMercado(update) {
    let participantRegistry = await getParticipantRegistry('org.creacion.Prosumer');
    let results = await query('comprobarApto');
    for (let n = 0; n < results.length; n++) {
        let prosumer = results[n];
        prosumer.Apto = false;
        await participantRegistry.update(prosumer);
    }
}
```

Listado A.3: queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que contiene los datos personales de los usuarios prosumidores (lenguaje propio inspirado en SQL)

```
query comprobarDinero {
  description: "Selecciona los prosumidores con dinero suficiente para ser considerados aptos"
  statement:
      SELECT org.creacion.Prosumer
          WHERE (Coins > _$Coins)
}

query comprobarApto {
  description: "Selecciona los prosumidores aptos para participar en el mercado"
  statement:
      SELECT org.creacion.Prosumer
          WHERE (Apto == true)
}

query prosumers {
  description: "Selecciona todos los prosumidores"
  statement:
      SELECT org.creacion.Prosumer
}
```

Listado A.4: permissions.acl – Código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene los datos personales de los usuarios prosumidores

```
rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

rule UserSystem {
    description: "Grant users full access to system resources"
    participant: "org.creacion.Prosumer"
    operation: READ
    resource: "org.hyperledger.composer.system.Network"
    action: ALLOW
}

rule UserSystemAddParticipant {
    description: "Grant users full access to system resources"
    participant: "org.creacion.Prosumer"
    operation: ALL
    resource: "org.hyperledger.composer.system.AddParticipant"
    action: DENY
}

rule SampleConditionalRule {
    description: "Cada Prosumidor puede ver sus propios datos"
    participant(m): "org.creacion.Prosumer"
    operation: READ
    resource(v): "org.creacion.Prosumer"
    condition: (v.getIdentifier() == m.getIdentifier())
    action: ALLOW
}
```

A.1.2 Cadena de bloques de mercado

Listado A.5: org.mercado.cto – Código de definición del modelo de la cadena de bloques del mercado, donde se almacenan las ofertas de los usuarios, los resultados del mercado y los precios de la red (lenguaje CTO)

```
namespace org.mercado

participant Prosumer identified by ID {
  o String ID
  o Boolean Apto
 }

asset Oferta identified by ofertaID{
  o String ofertaID
  o Double cantidad
  o Double precio
  o String owner
}

asset Resultado identified by resultadoID{
  o String resultadoID
  o Double cantidad
  o Double precio
  o String IDvendedor
  o String IDcomprador
  o String periodo
}

asset precioMayorista identified by periodo{
  o String periodo
  o DateTime timestamp
  o Double precioCompra
  o Double precioVenta
}

transaction iniciarMercado{
  o String[] prosumers
}

transaction finalizarMercado{}

transaction casarMercado{}

transaction insertarOferta{
    o String owner
    o Double cantidad
    o Double precio
}

event Evento{
  o Boolean casado
  o Boolean iniciado2
}

event Evento2{
  o Boolean oferta
}
```

Listado A.6: script.js – Primera parte del código de las transacciones definidas en la cadena de bloques del mercado (lenguaje node js)

```
'use strict';

/**
 * Los usuarios que se reciben son dados de alta como aptos para participar en el mercado
 * @param {org.mercado.iniciarMercado} update - los datos a ser actualizados
 * @transaction
 */
async function iniciarMercado(update) {
    let participantRegistry = await getParticipantRegistry('org.mercado.Prosumer');
    var array = [];
    for (let n = 0; n < update.length; n++) {
        let prosum = await participantRegistry.get(update.prosumers[n])
        prosum.Apto = true;
        array.push(prosum);
    }
    await participantRegistry.updateAll(array)
    var factory = getFactory();
    var Evento = await factory.newEvent('org.mercado', 'Evento');
  Evento.casado = false;
    Evento.iniciado2 = true;
    emit(Evento);
}
/**
 * Se pone a todos los usuarios como no aptos para participar en el mercado y se
 * borran las ofertas introducidas durante la fase de introduccion de ofertas
 * @param {org.mercado.finalizarMercado} update - los datos a ser actualizados
 * @transaction
 */
async function finalizarMercado(update) {
    let participantRegistry = await getParticipantRegistry('org.mercado.Prosumer');
    let assetRegistry1 = await getAssetRegistry('org.mercado.Oferta');
    let results = await query('Aptos');
    var array = [];
    for (let n = 0; n < results.length; n++) {
        let prosumer = results[n];
        prosumer.Apto = false;
        array.push(prosumer);
        }
    await participantRegistry.updateAll(array);
    var array2=[];
    let ofertas = await query ('ofertas');
    for (let n = 0; n < ofertas.length; n++) {
        let oferta = ofertas[n];
        oferta.cantidad = 0;
        oferta.precio = 0;
        array2.push(oferta);
        }
    await assetRegistry1.updateAll(ofertas);
    var array3=[]
}
```

Listado A.7: script.js – Segunda parte del código de las transacciones definidas en la cadena de bloques del mercado (lenguaje node js)

```
/**
 * Une a los usuarios que quieren comprar energÃa con los que quieren venderla
 * creando un activo que refleja la transaccion que van a realizar
 * @param {org.mercado.casarMercado} update - los datos a ser actualizados
 * @transaction
 */
async function casarMercado(){
    var d = new Date();
    var month = d.getMonth() + 1; //months from 1-12
    var day = d.getDate();
    var year = d.getFullYear();
    var hora_actual = d.getUTCHours();
    var minuto_actual = d.getMinutes()+1;
    var factory = getFactory();
    let precios = await query('precios',{timestamp : d })
    let precioMayorista = precios[0];
    let resultadoRegistry = await getAssetRegistry('org.mercado.Resultado')
    let energiaRequerida = 0;
    let ofertas_venta = await query('vendedores');
    let ofertas_compra = await query('compradores');
    for (let n=0; n<ofertas_compra.length; n++){energiaRequerida = energiaRequerida +
        ofertas_compra[n].cantidad*-1}
    let energiaOfertada = 0;
    for (let n=0; n<ofertas_venta.length; n++){energiaOfertada = energiaOfertada +
        ofertas_venta[n].cantidad}
    let precio = (precioMayorista.precioCompra * energiaRequerida + precioMayorista.
        precioVenta * energiaOfertada)/(energiaOfertada+energiaRequerida)
    let res=0;
    var cantidad = 0;
    let n = ofertas_compra.length-1;
    let m = ofertas_venta.length-1;
    var array=[];
    while(true){
      if (m < 0 || n < 0)break
      let resource = await factory.newResource('org.mercado','Resultado','('+res.toString()+')
          '+minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year);
      resource.IDvendedor = ofertas_venta[m].owner;
      resource.IDcomprador = ofertas_compra[n].owner;
      if (ofertas_compra[n].cantidad*-1 > ofertas_venta[m].cantidad){cantidad = ofertas_venta[
          m].cantidad}
      else {cantidad = ofertas_compra[n].cantidad*-1;}
      ofertas_compra[n].cantidad = ofertas_compra[n].cantidad + cantidad;
      ofertas_venta[m].cantidad = ofertas_venta[m]-cantidad;
      resource.precio= precio;
      resource.cantidad = cantidad;
      resource.periodo = (minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year);
      array.push(resource);
      res++;
      if (ofertas_compra[n].cantidad==0) n--;
      if (ofertas_venta[m].cantidad==0) m--;
    }
    await resultadoRegistry.addAll(array);
    var Evento = await factory.newEvent('org.mercado', 'Evento');
    Evento.iniciado2 = false;
  Evento.casado = true;
    emit(Evento);
}
```

Listado A.8: script.js – Tercera parte del código de las transacciones definidas en la cadena de bloques del mercado (lenguaje node js)

```
/**
 * Crea el activo oferta, solo lo pueden insertar los prosumidores aceptados
 * @param {org.mercado.insertarOferta} update - los datos a ser actualizados
 * @transaction
 */
async function insertarOferta(update) {
    let assetRegistry = await getAssetRegistry('org.mercado.Oferta');
    let factory = getFactory();
    let oferta = await factory.newResource('org.mercado','Oferta',update.owner.concat('_ofer')
        )
    oferta.cantidad = update.cantidad;
    oferta.precio=update.precio;
    oferta.owner=update.owner;
    await assetRegistry.update(oferta)
    var Evento = await factory.newEvent('org.mercado', 'Evento2');
  Evento.oferta = true;
    emit(Evento);
}
```

Listado A.9: queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que contiene los datos del mercado y los precios de los intercambios con la red (lenguaje propio inspirado en SQL)

```
query Aptos {
  description: "Selecciona los prosumidores aptos para participar en el mercado"
  statement:
      SELECT org.mercado.Prosumer
          WHERE (Apto == true)
}

query resultados {
  description: "Selecciona los resultados del mercado"
  statement:
      SELECT org.mercado.Resultado
          WHERE (_$periodo == periodo)
}

query vendedores {
  description: "Selecciona los prosumidores con excedente de energÃa"
  statement:
      SELECT org.mercado.Oferta
          WHERE (cantidad > 0)
          ORDER BY [cantidad DESC]
}

query compradores {
  description: "Selecciona los prosumidores con deficit de energÃa"
  statement:
      SELECT org.mercado.Oferta
          WHERE (cantidad < 0)
            ORDER BY [cantidad ASC]
}

query ofertas{
  description: "Selecciona todas las ofertas"
  statement:
      SELECT org.mercado.Oferta
}

query precios{
  description: "Selecciona todos los precios con una hora posterior a la indicada"
    statement:
        SELECT org.mercado.precioMayorista
            WHERE (_$timestamp < timestamp)
          ORDER BY [timestamp]
}
```

Listado A.10: permissions.acl – Primera parte del código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene los datos del mercado y los precios de los intercambios con la red

```
rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

rule UserSystem {
    description: "Grant users full access to system resources"
    participant: "org.mercado.Prosumer"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

rule SampleConditionalRule {
    description: "Cada Prosumidor puede ver sus propios datos"
    participant(m): "org.mercado.Prosumer"
    operation: READ
    resource(v): "org.mercado.Prosumer"
    condition: (v.getIdentifier() == m.getIdentifier())
    action: ALLOW
}

rule precioMayoristaLeerTodos {
    description: "Cada Prosumidor puede ver sus propios datos"
    participant: "org.mercado.Prosumer"
    operation: READ
    resource: "org.mercado.precioMayorista"
    action: ALLOW
}

rule resultadoMercadoSoloImplicados {
    description: "Cada Prosumidor puede ver sus propios datos"
    participant(m): "org.mercado.Prosumer"
    operation: READ
    resource(v): "org.mercado.Resultado"
    condition: (v.IDcomprador == m.getIdentifier() || v.IDvendedor == m.getIdentifier())
    action: ALLOW
}

rule ofertaSoloOwner {
    description: "Cada Prosumidor puede ver sus propios datos"
    participant(m): "org.mercado.Prosumer"
    operation: READ
    resource(v): "org.mercado.Oferta"
    condition: (v.owner == m.getIdentifier())
    action: ALLOW
}
```

Listado A.11: permissions.acl – Segunda parte del código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene los datos del mercado y los precios de los intercambios con la red

```
rule OfertaSoloTx {
    description: "Solo se pueden insertar ofertas mediante la transaccion para ello"
    participant(m): "org.mercado.Prosumer"
    operation: UPDATE
    resource(v): "org.mercado.Oferta"
    transaction: "org.mercado.insertarOferta"
    condition: (v.owner == m.getIdentifier())
    action: ALLOW
}

rule insertarOferta {
    description: "Cada prosumidor solo puede insertar sus ofertas"
    participant(m): "org.mercado.Prosumer"
    operation: CREATE
    resource(v): "org.mercado.insertarOferta"
    condition: (v.owner == m.getIdentifier())
    action: ALLOW
}

rule insertarOfertaSoloAptos {
    description: "Solo los usuarios aptos pueden insertar ofertas "
    participant(m): "org.mercado.Prosumer"
    operation: CREATE
    resource: "org.mercado.insertarOferta"
    condition: (m.Apto == true)
    action: ALLOW
}
```

A.1.3 Cadena de bloques para el registro de parámetros eléctricos

Listado A.12: org.creacion.cto – Código de definición del modelo de la cadena de bloques donde se registran los datos de los contadores inteligentes de los usuarios prosumidores (lenguaje CTO)

```
/*
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

namespace org.registro

participant Prosumer identified by ID {
  o String ID
  --> Acumulados acumulados
  --> Instantaneos instantaneos
  o Double Diferencia
}

asset Acumulados identified by acumuladosID{
  o String acumuladosID
  o String ownerID
  o Double generacion
  o Double consumo
}

asset Instantaneos identified by instantaneosID{
  o String instantaneosID
  o String ownerID
  o Double generacionActiva
  o Double generacionReactiva
  o Double consumoActiva
  o Double consumoReactiva
  o Double voltaje
}

transaction actualizarDiferencia {
  -->Prosumer prosum
}
```

Listado A.13: script.js – Código de las transacciones definidas en la cadena de bloques que almacena los parámetros eléctricos de los usuarios prosumidores (lenguaje node.js)

```
'use strict';


/**
 * Transaccion para actualizar el valor de la energia que sobra/falta una vez que se
 *han leido el consumo y la generacion
 * @param {org.registro.actualizarDiferencia} UpdateValues
 * @transaction
 */


function actualizarDiferencia(UpdateValues){
  UpdateValues.prosum.Diferencia=UpdateValues.prosum.acumulados.generacion - UpdateValues.
      prosum.acumulados.consumo;
    return getParticipantRegistry('org.registro.Prosumer')
      .then(function(participantRegistry) {
        return participantRegistry.update(UpdateValues.prosum);
        });
}
```

Listado A.14: queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que almacena los parámetros eléctricos de los usuarios prosumidores (lenguaje propio inspirado en SQL)

```
query prosumers {
  description: "Selecciona todos los prosumidores"
  statement:
      SELECT org.registro.Prosumer
}

/*En principio no se usa esta peticion*/
query consumos {
  description: "Selecciona todos los consumos y generaciones con fines estadisticos"
  statement:
      SELECT org.registro.Acumulados
}
```

Listado A.15: permissions.acl – Código con los permisos de acceso, lectura, creación y edición para la cadena de bloquesque almacena los parámetros eléctricos de los usuarios prosumidores

```
1
2   rule NetworkAdminUser {
3       description: "Grant business network administrators full access to user resources"
4       participant: "org.hyperledger.composer.system.NetworkAdmin"
5       operation: ALL
6       resource: "**"
7       action: ALLOW
8   }
9
10  rule NetworkAdminSystem {
11      description: "Grant business network administrators full access to system resources"
12      participant: "org.hyperledger.composer.system.NetworkAdmin"
13      operation: ALL
14      resource: "org.hyperledger.composer.system.**"
15      action: ALLOW
16  }
17
18  rule UserSystem {
19      description: "Grant users full access to system resources"
20      participant: "org.registro.Prosumer"
21      operation: ALL
22      resource: "org.hyperledger.composer.system.**"
23      action: ALLOW
24  }
25
26  rule SampleConditionalRule {
27      description: "Cada Prosumidor puede ver sus propios datos"
28      participant(m): "org.registro.Prosumer"
29      operation: ALL
30      resource(v): "org.registro.Prosumer"
31      condition: (v.getIdentifier() == m.getIdentifier())
32      action: ALLOW
33  }
34
35  rule acumuladosSoloUser {
36      description: "Cada Prosumidor puede ver sus propios datos"
37      participant(m): "org.registro.Prosumer"
38      operation: ALL
39      resource(v): "org.registro.Acumulados"
40      condition: (v.ownerID == m.getIdentifier())
41      action: ALLOW
42  }
43  rule instantaneosSoloUser {
44      description: "Cada Prosumidor puede ver sus propios datos"
45      participant(m): "org.registro.Prosumer"
46      operation: ALL
47      resource(v): "org.registro.Instantaneos"
48      condition: (v.ownerID == m.getIdentifier())
49      action: ALLOW
50  }
51
52  rule actualizarSoloUser {
53      description: "Cada Prosumidor puede ver sus propios datos"
54      participant(m): "org.registro.Prosumer"
55      operation: CREATE
56      resource(v): "org.registro.actualizarDiferencia"
57      condition: (v.prosum.getIdentifier() == m.getIdentifier())
58      action: ALLOW
59  }
```

A.1.4 Cadena de bloques de liquidación de pagos

Listado A.16: org.liquidacion.cto – Código de definición del modelo de la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje CTO)

```
namespace org.liquidacion

participant Prosumer identified by ID{
  o String ID
  o Double CoinsPeriodo
  o Double ReputacionPeriodo
  o Double Desajuste
}

asset Resultado identified by resultadoID{
  o String resultadoID
  o Double cantidad
  o Double precio
  o String IDvendedor
  o String IDcomprador
  o String periodo
}

asset ResultadoDesajuste identified by resultadoID{
  o String resultadoID
  o Double cantidad
  o Double precio
  o String IDvendedor
  o String IDcomprador
  o String periodo
}

asset ResultadoUtility identified by resultadoID{
  o String resultadoID
  o Double cantidad
  o Double precio
  o String IDprosumer
  o String periodo
}

transaction transacciones{
  o Boolean mercado
}

transaction casarDesajuste{
  o Double precio
}

transaction transaccionesUtility{
  o Double precioCompra
    o Double precioVenta
}

event Evento{
  o Boolean mercado
  o Boolean desajuste
  o Boolean casado
}
```

Listado A.17: script.js – Código de las transacciones definidas en la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje node.js)

```
'use strict';

/**
 * Realiza una llamada a la base de datos para comprobar que transacciones realizar
 * y las ejecuta, restando o sumando coins a los usuarios correspondientes
 * @param {org.liquidacion.transacciones} update - los datos a ser actualizados
 * @transaction
 */
async function transacciones(update) {
    let participantRegistry = await getParticipantRegistry('org.liquidacion.Prosumer');
    var d = new Date();
    var month = d.getMonth() + 1; //months from 1-12
    var day = d.getDate();
    var year = d.getFullYear();
    var hora_actual = d.getUTCHours();
    var minuto_actual = d.getMinutes()-1;
    var factory = getFactory();
    var Evento = await factory.newEvent('org.liquidacion', 'Evento');
    if(update.mercado){
        Evento.mercado=true;
        Evento.desajuste=false;
        Evento.casado=false;
      var transacciones = await query('resultados',{ periodo: (minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year)})
    }else {
        Evento.mercado=false;
        Evento.desajuste=true;
        Evento.casado=false;
      var transacciones = await query('resultadosDesajuste',{ periodo: (minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year)})
        }

    for (let n=0; n<transacciones.length; n++){
        let trans = transacciones[n];
        let coins = trans.cantidad * trans.precio;
        let comprador = await participantRegistry.get(trans.IDcomprador)
        let vendedor = await participantRegistry.get(trans.IDvendedor)
    comprador.Desajuste = comprador.Desajuste + trans.cantidad
        vendedor.Desajuste = vendedor.Desajuste - trans.cantidad
        comprador.CoinsPeriodo = comprador.CoinsPeriodo - coins;
        vendedor.CoinsPeriodo = vendedor.CoinsPeriodo + coins;
      await participantRegistry.update(comprador);
    await participantRegistry.update(vendedor);
    }

    emit(Evento);
}
```

Listado A.18: script.js – Segunda parte del código de las transacciones definidas en la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje node.js)

```
/**
 * Une los usuarios con un desajuste positivo con los que tienen un desajuste negativo
 * @param {org.liquidacion.casarDesajuste} update - los datos a ser actualizados
 * @transaction
 */
async function casarDesajuste(update){
    var d = new Date();
    var month = d.getMonth() + 1; //months from 1-12
    var day = d.getDate();
    var year = d.getFullYear();
    var hora_actual = d.getUTCHours();
    var minuto_actual = d.getMinutes()-1;
    let factory = getFactory();
    let excedentes = await query('desajustePositivo');
    let deudores = await query('desajusteNegativo');
    let m = excedentes.length-1;
    let n = deudores.length-1;
    var array = []
    var cantidad = 0;
    let res = 0;
    let resultadoRegistry = await getAssetRegistry('org.liquidacion.ResultadoDesajuste')
    while (true) {
        if (m < 0 || n < 0)break;
        let resultado = await factory.newResource('org.liquidacion','ResultadoDesajuste','('+
            res.toString()+')'+minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year);
        if (Math.abs(deudores[n].Desajuste)>excedentes[m].Desajuste){
          cantidad = excedentes[m].Desajuste;
        }
        else{
          cantidad=Math.abs(deudores[n].Desajuste);
        }
        excedentes[m].Desajuste=excedentes[m].Desajuste-cantidad;
        deudores[n].Desajuste=deudores[n].Desajuste+cantidad;
        resultado.periodo = (minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year);
        resultado.cantidad = cantidad;
        resultado.precio = update.precio;
        resultado.IDcomprador = deudores[n].ID;
        resultado.IDvendedor = excedentes[m].ID;
        if (deudores[n].Desajuste==0) n--;
        if (excedentes[m].Desajuste==0) m--;
        await resultadoRegistry.add(resultado);
        res++;
    }
  var Evento = await factory.newEvent('org.liquidacion', 'Evento');
    Evento.casado=true;
    Evento.mercado=false;
    Evento.desajuste=false;
    emit(Evento);
}
```

Listado A.19: script.js – Tercera parte del código de las transacciones definidas en la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje node.js)

```
/**
 * Los usuarios que no han sido casados entre si se les crea una transaccion con la Utility
 * @param {org.liquidacion.transaccionesUtility} update - los datos a ser actualizados
 * @transaction
 */
async function transaccionesUtility(UpdateValues) {
    var d = new Date();
    var month = d.getMonth() + 1; //months from 1-12
    var day = d.getDate();
    var year = d.getFullYear();
    var hora_actual = d.getUTCHours();
    var minuto_actual = d.getMinutes()-1;
    let factory = getFactory();
    let desajustados = await query('desajustados');
    let assetRegistry = await getAssetRegistry('org.liquidacion.ResultadoUtility');
    let participantRegistry = await getParticipantRegistry('org.liquidacion.Prosumer');
    var array = []
    var res = 0;
    for (let n=0; n<desajustados.length; n++){
      let resultado = await factory.newResource('org.liquidacion','ResultadoUtility','('+res.
          toString()+')'+minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year);
      resultado.cantidad = desajustados[n].Desajuste;
      desajustados[n].Desajuste = 0;
      resultado.IDprosumer = desajustados[n].ID;
      if(resultado.cantidad>0){resultado.precio = UpdateValues.precioVenta}
      else resultado.precio = UpdateValues.precioCompra;
      desajustados[n].CoinsPeriodo = resultado.cantidad * resultado.precio
      resultado.periodo = (minuto_actual+'-'+hora_actual+'-'+day+'-'+month+'-'+year);
      array.push(resultado);
      res++;
    }
    await assetRegistry.addAll(array);
    await participantRegistry.updateAll(desajustados);
}
```

Listado A.20: queries.qry – Código con la definición de las peticiones a la base de datos de la cadena de bloques que contiene todas las transacciones que se realizan (lenguaje propio inspirado en SQL)

```
query prosumers {
  description: "Selecciona los prosumidores que han tenido algun pago/cobro en este periodo"
  statement:
      SELECT org.liquidacion.Prosumer
          WHERE (CoinsPeriodo != 0)
}

query desajustePositivo {
  description: "Selecciona los prosumidores con mÃ¡s energÃa generada de la prevista"
  statement:
      SELECT org.liquidacion.Prosumer
          WHERE (Desajuste > 0)
          ORDER BY [Desajuste ASC]
}

query desajusteNegativo {
  description: "Selecciona los prosumidores con menos energÃa generada de la prevista"
  statement:
      SELECT org.liquidacion.Prosumer
          WHERE (Desajuste < 0)
            ORDER BY [Desajuste DESC]
}

query desajustados {
  description: "Selecciona los prosumidores con menos energÃa generada de la prevista"
  statement:
      SELECT org.liquidacion.Prosumer
          WHERE (Desajuste != 0)
            ORDER BY [Desajuste]
}

query resultados {
  description: "Selecciona los resultados del mercado"
  statement:
      SELECT org.liquidacion.Resultado
          WHERE (_$periodo == periodo)
}

query resultadosDesajuste {
  description: "Selecciona los resultados de la eliminacion del desajuste"
  statement:
      SELECT org.liquidacion.ResultadoDesajuste
          WHERE (_$periodo == periodo)
}

query resultadosUtitliy {
  description: "Selecciona los resultados de las tx con la Utility"
  statement:
      SELECT org.liquidacion.ResultadoUtility
          WHERE (_$periodo == periodo)
}
```

Listado A.21: permissions.acl – Código con los permisos de acceso, lectura, creación y edición para la cadena de bloques que contiene todas las transacciones que se realizan

```
/*
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

rule UserSystem {
    description: "Grant users full access to system resources"
    participant: "org.liquidacion.Prosumer"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

rule SampleConditionalRule {
    description: "Cada Prosumidor puede ver sus propios datos"
    participant(m): "org.liquidacion.Prosumer"
    operation: READ
    resource(v): "org.liquidacion.Prosumer"
    condition: (v.getIdentifier() == m.getIdentifier())
    action: ALLOW
}
```

A.2 Archivos de instalación

Listado A.22: instalacion.sh – Conjunto de comandos que se deben ejecutar para crear una red de *Hyperledger Fabric* con el nodo principal en el ordenador del gestor del sistema e instalar las cadenas de bloques

```
docker kill $(docker ps -q)
docker rm $(docker ps -aq)
docker rmi $(docker images dev-* -q)
./desinstalarUsuarios.sh
composer card delete -c admin@creacion
composer card delete -c admin@mercado
composer card delete -c admin@registro
composer card delete -c admin@liquidacion
cd ~/Blockchain/fabric-dev-servers
./startFabric.sh
./createPeerAdminCard.sh
cd ~/Blockchain/archivos_instalacion
./instalar_cadenas.sh
```

Listado A.23: instalacionordenador2.sh – Conjunto de comandos que se deben ejecutar en los equipos de los usuarios prosumidores para unirse a la red de *Hyperledger Fabric* creada por el gestor del sistema

```
docker kill $(docker ps -q)
docker rm $(docker ps -aq)
docker rmi $(docker images dev-* -q)
./desinstalarUsuarios.sh
cd ~/Blockchain/fabric-dev-servers
export FABRIC_VERSION=hlfv12
./teardownFabric.sh
./startFabric.sh
```

Listado A.24: crearNUsuarios.sh – Conjunto de comandos que llama a la función crearUsuario.sh N veces y deja al cliente del prosumidor corriendo en segundo plano. Se emplea únicamente en pruebas para simular un gran número de prosumidores.

```
echo "Â¿Cuantos quiere ejecutar?"
read n
for ((i = 1 ; i <= $n ; i++))
do
./crearUsuario.sh 1 &
sleep 45
done
```

Listado A.25: crearUsuario.sh – Conjunto de comandos que se deben ejecutar en el equipo del prosumidor para dar de alta a un usuario y crear la carpeta que posteriormente se copia en su equipo

```
1  echo "Creando usuario en las cadenas de bloques"
2  cd ../clientes/
3  nodejs crearProsumer.js
4  cd auxiliares
5  ID=`cat auxiliar.json | node -pe 'JSON.parse(fs.readFileSync(0)).ID'`
6  IDsecreto=`cat auxiliar.json | node -pe 'JSON.parse(fs.readFileSync(0)).IDsecreto'`
7  cd ../../cards/
8  mkdir $ID
9  cd $ID
10 cp ../../clientes/auxiliares/auxiliar.json ID.json
11 cp ../../clientes/auxiliares/oferta_auxiliar.json oferta.json
12 cp ../../clientes/auxiliares/acumulado_auxiliar.json acumulado.json
13 comando='composer identity issue -c'
14 cardRegistro=admin@registro
15 cardCreacion=admin@creacion
16 cardMercado=admin@mercado
17 cardLiquidacion=admin@liquidacion
18 $comando $cardCreacion -f $ID.creacion -u $ID.creacion -a "resource:org.creacion.Prosumer#$ID"
19 $comando $cardMercado -f $ID.mercado -u $ID.mercado -a "resource:org.mercado.Prosumer#
       $IDsecreto"
20 $comando $cardRegistro -f $ID.registro -u $ID.registro -a "resource:org.registro.Prosumer#
       $IDsecreto"
21 $comando $cardLiquidacion -f $ID.liquidacion -u $ID.liquidacion -a "resource:org.liquidacion.
       Prosumer#$IDsecreto"
22 cp ../../clientes/clienteProsumer.js clienteProsumer.js
23 cp ../../archivos_instalacion/instalarUsuario.sh instalarUsuario.sh
24 cp -r ../../clientes/node_modules/ node_modules;
25
26 #si se pone algo junto con el comando se instala directamente,
27 #eso se hace para en las pruebas instalar varios usuarios sin confirmacion
28 #si no se pone nada se pregunta si desea instalarse o no
29 if [ -z $1 ]
30 then
31 echo "Â¿Desea instalar el usuario en este ordenador e iniciar el cliente? 1/2"
32
33 select yn in "Si" "No"; do
34 case $yn in
35   Si ) ./instalarUsuario.sh $ID;cd ../../clientes/;nodejs clienteProsumer.js $ID; exit;;
36   No ) exit;;
37 esac
38 done
39 fi
40
41 ./instalarUsuario.sh $ID
42 cd ../../clientes/
43 nodejs clienteProsumer.js $ID
```

Listado A.26: instalarUsuario.sh – Conjunto de comandos para instalar el usuario del prosumidor en su dispositivo.

```
ID=$1;
archivo="ID.json"
if [ -z $ID ];then
if [ -e $archivo ]
  then
    ID=`cat ID.json | node -pe 'JSON.parse(fs.readFileSync(0)).ID'`
  else
    echo "Escriba el ID que desea instalar"
    read ID
    cd ../cards/$ID
fi;fi
composer card import --file $ID.creacion.card
composer card import --file $ID.mercado.card
composer card import --file $ID.registro.card
composer card import --file $ID.liquidacion.card
cd ../../clientes/
nodejs clienteProsumer.js $ID
```

Listado A.27: desinstalarUsuario.sh – Conjunto de comandos que elimina la instalación de las tarjetas de identificación del usuario prosumidor.

```
echo "Â¿Desea desinstalar todos los usuarios instalados en este ordenador? 1/2"
select yn in "Si" "No"; do
case $yn in
  Si ) ./desinstalarUsuarios.sh; exit;;
  No ) cd ../cards/; echo "Escriba el ID del usuario que desea instalar"; echo "Tiene los
       siguientes disponibles:"; ls;read ID; break;;
esac;done
composer card delete -c $ID.creacion@creacion
composer card delete -c $ID.mercado@mercado
composer card delete -c $ID.registro@registro
composer card delete -c $ID.liquidacion@liquidacion
rm -r $ID
```

Listado A.28: desinstalarUsuarios.sh – Desinstala todos los usuario prosumidores. Es un conjunto de comandos empleado únicamente en las pruebas

```
cd ../cards/
ID=(*/)
for i in "${ID[@]}"
do
inicio=0
longitud=$((${#i} - 1))
ID=${i:${inicio}:${longitud}}
rm -r $ID
composer card delete -c $ID.creacion@creacion
composer card delete -c $ID.mercado@mercado
composer card delete -c $ID.registro@registro
composer card delete -c $ID.liquidacion@liquidacion
done
```

A.3 Cliente en Node.js de creación de usuarios prosumidores

Listado A.29: crearProsumer.js – Programa en node.js que crea un usuario en cada una de las cadenas blockchain con sus activos correspondientes

```
const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;
var jf = require('jsonfile')
const fs = require('fs')
let cardCreacion='admin@creacion';
let cardMercado= 'admin@mercado';
let cardRegistro='admin@registro';
let cardLiquidacion = 'admin@liquidacion';
let Creacion = 'org.creacion';
let Mercado = 'org.mercado';
let Registro = 'org.registro';
let Liquidacion = 'org.liquidacion';
let ID=String(Math.floor(Math.random()*10000));
let IDsecreto=String(Math.floor(Math.random()*10000))
var coins = 9999999999999;
let user = {
    ID: ID,
    IDsecreto: IDsecreto,
};
let data = JSON.stringify(user);
fs.writeFileSync('auxiliares/auxiliar.json', data);

let oferta = {
    cantidad: (Math.random()-0.5)*1000,
    precio: Math.random()*1000,
};
let data2 = JSON.stringify(oferta);
fs.writeFileSync('auxiliares/oferta_auxiliar.json', data2);

let contador = {
    generacion: Math.random()*1000,
    consumo: Math.random()*1000,
};
let data3 = JSON.stringify(contador);
fs.writeFileSync('auxiliares/acumulado_auxiliar.json', data3);

// se crean las conexiones a cada una de las redes de blockchain
const bnc1 = new BusinessNetworkConnection();
const bnc2 = new BusinessNetworkConnection();
const bnc3 = new BusinessNetworkConnection();
const bnc4 = new BusinessNetworkConnection();

//Se crean todos los activos
async function crearInstantaneos(IDsecreto){
    await bnc3.getAssetRegistry('org.registro.Instantaneos')
    .then(function(Registry){
        let factory = bnd3.getFactory();
        let resource = factory.newResource(Registro,'Instantaneos',IDsecreto.concat('_inst'))
        resource.ownerID=IDsecreto;
        resource.generacionActiva=0;
        resource.generacionReactiva=0;
        resource.consumoActiva=0;
        resource.consumoReactiva=0;
        resource.voltaje=0;
        return Registry.add(resource)
    });
}
```

Listado A.30: crearProsumer.js – Programa en node.js que crea un usuario en cada una de las cadenas blockchain con sus activos correspondientes

```
async function crearAcumulados(IDsecreto){
    await bnc3.getAssetRegistry('org.registro.Acumulados')
    .then(function(Registry){
        let factory = bnd3.getFactory();
        let resource = factory.newResource(Registro,'Acumulados',IDsecreto.concat('_acum'))
        resource.ownerID=IDsecreto;
        resource.generacion=0;
        resource.consumo=0;
        return Registry.add(resource)
    });
}

async function crearOferta(IDsecreto){
    let assetRegistry = await bnc2.getAssetRegistry('org.mercado.Oferta')
    let factory = bnd2.getFactory();
    let oferta = await factory.newResource(Mercado,'Oferta',IDsecreto.concat('_ofer'))
    oferta.owner=IDsecreto;
    oferta.cantidad=0;
    oferta.precio=0;
    await assetRegistry.add(oferta)
}

//Se crea el usuario en las cuatro redes, y en cada una de ellas se llama a las funciones para
     crear sus activos correspondientes
async function crearProsumer1 (){
    //console.log("Creando prosumer creacion")
    let participantRegistry = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
    let factory = bnd1.getFactory();
    let prosum = await factory.newResource(Creacion,'Prosumer',ID)
    prosum.IDsecreto = IDsecreto;
    prosum.Nombre = 'Nombre';
    prosum.Apellidos ='Apellidos';
    prosum.Direccion ='Direccion';
    prosum.TipoGeneracion = 'Tipo Generacion';
    prosum.Coins=0;
    prosum.Reputacion =5;
    prosum.Apto=false;
    await participantRegistry.add(prosum)
    }

async function crearProsumer2(){
    //console.log("Creando prosumer mercado")
    let participantRegistry1 = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
    let participantRegistry2 = await bnc2.getParticipantRegistry('org.mercado.Prosumer')
    let prosum1= await participantRegistry1.get(ID);
    await crearOferta(prosum1.IDsecreto);
    let factory = bnd2.getFactory();
    let prosum2 = await factory.newResource(Mercado,'Prosumer',prosum1.IDsecreto)
    prosum2.Apto=false;
    await participantRegistry2.add(prosum2)
}
```

Listado A.31: crearProsumer.js – Programa en node.js que crea un usuario en cada una de las cadenas blockchain con sus activos correspondientes

```
async function crearProsumer3(){
    //console.log("Creando prosumer registro")
    let participantRegistry1 = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
    let prosum1= await participantRegistry1.get(ID)
    await crearInstantaneos(prosum1.IDsecreto);
    await crearAcumulados(prosum1.IDsecreto);
    let participantRegistry3 = await bnc3.getParticipantRegistry('org.registro.Prosumer')
    let factory = bnd3.getFactory();
    let prosum3 = factory.newResource(Registro,'Prosumer',prosum1.IDsecreto)
    prosum3.acumulados=factory.newRelationship(Registro,'Acumulados',prosum1.IDsecreto.concat('_acum'));
    prosum3.instantaneos=factory.newRelationship(Registro,'Instantaneos',prosum1.IDsecreto.concat('_inst'));
    prosum3.Diferencia=0;
    await participantRegistry3.add(prosum3)
}
async function crearProsumer4(){
    //console.log("Creando prosumer liquidacion")
    let participantRegistry1 = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
    let prosum1= await participantRegistry1.get(ID)
    let participantRegistry4 = await bnc4.getParticipantRegistry('org.liquidacion.Prosumer')
    let factory = bnd4.getFactory();
    let prosum4 = factory.newResource(Liquidacion,'Prosumer',prosum1.IDsecreto)
    prosum4.CoinsPeriodo=0;
    prosum4.ReputacionPeriodo=0;
    prosum4.Desajuste=0;
    await participantRegistry4.add(prosum4)
}

//Desde esta función se conecta y se desconectan las conexiones y se llama a la creación de los usuarios
async function crearUsuario(){
    bnd1 = await bnc1.connect(cardCreacion);
    bnd2 = await bnc2.connect(cardMercado);
    bnd3 = await bnc3.connect(cardRegistro);
    bnd4 = await bnc4.connect(cardLiquidacion);
    await crearProsumer1();
    await crearProsumer2();
    await crearProsumer3();
    await crearProsumer4();
    let participantRegistry1 = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
    let prosum1= await participantRegistry1.get(ID)
    prosum1.Coins= coins;
    await participantRegistry1.update(prosum1)
    await bnc1.disconnect(cardCreacion)
    await bnc2.disconnect(cardMercado);
    await bnc3.disconnect(cardRegistro);
    await bnc4.disconnect(cardLiquidacion)
}

crearUsuario();
```

A.4 Cliente en Node.js del gestor del sistema

Listado A.32: ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía

```
const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;
const exec = require('child_process').exec;
var jf = require('jsonfile')
const colors = require('colors');
//Variables auxiliares
let cardCreacion='admin@creacion';
let cardMercado= 'admin@mercado';
let cardRegistro='admin@registro';
let cardLiquidacion = 'admin@liquidacion';
let Creacion = 'org.creacion';
let Mercado = 'org.mercado';
let Liquidacion = 'org.liquidacion';
var ronda = '0';
var ronda1 = '0';
let usuarios = 0;
let ofertas_recibidas = 0;
primera_vez=true;
///Valores a variar en funcion de los parámetros que requiera
//el mercado para aceptar a los prosumidores
var coins_minimos=200;
var rep_minima = 4;

// se crean las conexiones a cada una de las cadenas blockchain y se conectan con la
    identificacion del administrador de cada una
const bnc1 = new BusinessNetworkConnection();
const bnc2 = new BusinessNetworkConnection();
const bnc3 = new BusinessNetworkConnection();
const bnc4 = new BusinessNetworkConnection();

//Los eventos creados por la ejecución de los contratos inteligentes
bnc1.on('event', (event) => {
    if (event.iniciado1) {
        iniciarMercado2();
    }
});

bnc2.on('event', (event) => {
    if (event.oferta) {
        if (ofertas_recibidas==1){setTimeout(function(){
            if(ofertas_recibidas<usuarios){casarMercado();
            console.log('Algún usuario no ha introducido su oferta')}
        },5000)}
        ofertas_recibidas++;
        process.stdout.write('Se ha(n) recibido '+ofertas_recibidas +'\r')
        if(ofertas_recibidas == usuarios){
            process.stdout.write('Se ha(n) recibido '+ofertas_recibidas)
            console.log(' todas recibidas'.green)
            casarMercado()
        }
    }
    if (event.casado) {
        finalizarMercado();
        console.log(' hecho'.green)
    }
});
```

Listado A.33: ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía

```
bnc4.on('event', (event) => {
    if (event.mercado) {
        console.log(' hecho'.green)
        casarDesajuste()
    }
    if (event.casado) {
        console.log(' hecho'.green)
        liquidarDesajuste()
    }
    if (event.desajuste){
        console.log(' hecho'.green)
        transaccionesUtility()
    }
    if (event.utility){
        console.log(' hecho'.green)
        realizarPagos();
    }
});

//Una función auxiliar que se emplea durante todo el programa, lo que hace es imprimir por pantalla la hora exacta
async function marca_tiempo(){
    var d= new Date();
    var hora_actual = d.getUTCHours();
    var minuto = d.getMinutes();
    var segundo =d.getSeconds();
    process.stdout.write(hora_actual+':'+minuto+':'+segundo+' --> ');
}

//La web de ESIOS no tiene una API para node js, por lo tanto se ha tenido que crear un archivo bash de linux para descargar los precios.
//Al archivo bash se se pasa el parámetro de la fecha que se quiere descargar el precio y lo descarga en dos archivos, uno con el precio de compra y otro con el de venta
async function descargayleePrecioMayorista(){
    console.log('Descargando precio mayorista de la web Esios (Solo una vez al día)')
    var d= new Date();
    var dia = d.getDate()
    if (primera_vez==false){dia = d.getDate()+1}
    else {primera_vez=false}
    var mes = d.getMonth()+1;
    var año =d.getFullYear();
    var fecha = año+'-'+mes+'-'+dia
    const file= '../archivos_instalacion/datosEsios.sh ';
    exec(file + fecha, function (error, stdout, stderr) {
            if (error !== null) {
            console.log(error);
            } else {
            console.log('stdout: ' + stdout);
            console.log('stderr: ' + stderr);
            }
        });
    setTimeout(leerPrecioMayorista,5*1000)
}
```

Listado A.34: ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía

```
1
2  //Esta función lee los archivos con el precio mayorista descargados por la función anterior
3  async function leerPrecioMayorista(){
4      var archivo1 = jf.readFileSync('../archivos/PVPCventa.json');
5      var archivo2 = jf.readFileSync('../archivos/PVPCcompra.json');
6      let assetRegistry2 = await bnc2.getAssetRegistry('org.mercado.precioMayorista')
7      let factory2 = await bnd2.getFactory();
8      var array = [];
9      for (let n = 0; n < archivo1.indicator.values.length; n++) {
10         let resource = await factory2.newResource(Mercado,'precioMayorista',archivo1.indicator
               .values[n].datetime_utc);
11         var d = new Date(archivo1.indicator.values[n].datetime_utc);
12         resource.timestamp = d;
13         resource.precioVenta=archivo1.indicator.values[n].value;
14         resource.precioCompra=archivo2.indicator.values[n].value;
15         array.push(resource)
16     }
17     await assetRegistry2.addAll(array);
18     process.stdout.clearLine();
19     console.log('Leido precio')
20 }
21
22 //Esta función crea una transacción de iniciar el mercado en la primera cadena que lo que hace
       es comprobar los coins y la resputación de cada usuario para comprobar si puede
23 //o no participar, si es apto para participar le resta la cantidad de la fianza de sus coins
24 async function iniciarMercado1 (){
25     let factory1 = bnd1.getFactory();
26     let iniciarMercado1 = factory1.newTransaction(Creacion,'iniciarMercado');
27     iniciarMercado1.coins = coins_minimos;
28     iniciarMercado1.reputacion = rep_minima;
29     await bnc1.submitTransaction(iniciarMercado1);
30 }
31
32 //mediante un query comprueba que usuarios son aptos, y se lo comunica al mercado, que
       mediante una transacción en la que recibe sus IDs los admite como participantes
33 async function iniciarMercado2(){
34     var IDs=[];
35     let aptos = await bnc1.query('comprobarApto');
36     usuarios = aptos.length;
37     if (usuarios==0){
38         console.log('No hay ningún usuario apto para introducir ofertas');
39     }
40     else {
41         for (let n = 0; n < aptos.length; n++) {IDs.push(aptos[n].IDsecreto);}
42         let factory2 = bnd2.getFactory();
43         let iniciarMercado2 = factory2.newTransaction(Mercado,'iniciarMercado');
44         iniciarMercado2.prosumers = IDs;
45         await bnc2.submitTransaction(iniciarMercado2);
46         console.log('Se espera a que los '+usuarios+' usuarios inserten sus ofertas antes de
               casar el mercado');
47     }
48 }
```

Listado A.35: ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía

```
async function casarMercado(){
    process.stdout.write('Casando las ofertas recibidas...')
    let factory2 = bnd2.getFactory();
    let casarMercado = factory2.newTransaction(Mercado,'casarMercado');
    await bnc2.submitTransaction(casarMercado);}
//Una vez obtenido el resultado del mercado se borran las pujas y ofertas que han dado ese resultado
//del mercado y se pone a todos los usuarios como no aptos, preparados para una nueva ronda de mercado
async function finalizarMercado(){
    let factory1 = bnd1.getFactory();
    let factory2 = bnd2.getFactory();
    let finalizarMercado1 = factory1.newTransaction(Creacion,'finalizarMercado');
    let finalizarMercado2 = factory2.newTransaction(Mercado,'finalizarMercado');
    await bnc1.submitTransaction(finalizarMercado1);
    await bnc2.submitTransaction(finalizarMercado2);
    marca_tiempo();
    console.log('Mercado finalizado de la ronda de las '+ ronda)
    ofertas_recibidas=0;}
//Inserta los datos de la cadena de registro en la cadena de liquidación
async function insertarDatosContador(){
    process.stdout.write('Cogiendo los datos de los contadores de los usuarios... ')
    let prosumers = await  bnc3.query('prosumers');
    let participantRegistry4 = await bnc4.getParticipantRegistry('org.liquidacion.Prosumer');
    var array = []
    for (let n=0; n<prosumers.length; n++){
        let prosum3=prosumers[n];
        let prosum4 = await participantRegistry4.get(prosum3.ID)
        prosum4.Desajuste=prosum3.Diferencia;
        array.push(prosum4)}
    await participantRegistry4.updateAll(array);
    console.log(' hecho'.green)}
//Se liquidan las transacciones establecidas en la subasta de energía
async function liquidarTransacciones(){
    process.stdout.write('Liquidando las transacciones del mercado...')
    let factory4 = await bnd4.getFactory();
    var d = new Date();
    var month = d.getMonth() + 1; //months from 1-12
    var day = d.getDate();
    var year = d.getFullYear();
    var hora_actual = d.getUTCHours();
    var minuto_actual = d.getMinutes()-1;
    let transacciones = await bnc2.query('resultados',{ periodo: (minuto_actual+'-'+
        hora_actual+'-'+day+'-'+month+'-'+year)});
    var array = []
    for (let n=0; n<transacciones.length; n++){
        let trans = transacciones[n];
        let resource = await factory4.newResource(Liquidacion,'Resultado',trans.resultadoID);
        resource.cantidad = trans.cantidad;
        resource.precio = trans.precio;
        resource.IDcomprador = trans.IDcomprador
        resource.IDvendedor = trans.IDvendedor
        resource.periodo = trans.periodo
        array.push(resource)}
    let assetRegistry4 = await bnc4.getAssetRegistry('org.liquidacion.Resultado')
    await assetRegistry4.addAll(array)
    let transaccion = await factory4.newTransaction(Liquidacion,'transacciones');
    transaccion.mercado = true;
    await bnc4.submitTransaction(transaccion);}
```

Listado A.36: ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía

```
//Se crean unas nuevas transacciones para realizar los pagos por la energía que no había sido
    reflejada en la subasta
async function casarDesajuste(){
    process.stdout.write('Casando a los usuarios con desajuste...')
    var d = new Date();
    let precios = await bnc2.query('precios',{timestamp : d })
    var precioMayorista = precios[0];
    var energiaRequerida = 0;
    var excedentes = await bnc4.query('desajustePositivo');
    var deudores = await bnc4.query('desajusteNegativo');
    for (let n=0; n<deudores.length; n++){
  energiaRequerida = energiaRequerida + Math.abs(deudores[n].Desajuste)
     }
    var energiaOfertada = 0;
    for (let n=0; n<excedentes.length; n++){
  energiaOfertada = energiaOfertada + excedentes[n].Desajuste
    }
    var precio = (precioMayorista.precioCompra * energiaRequerida + precioMayorista.
        precioVenta * energiaOfertada)/(energiaOfertada+energiaRequerida)
    let factory4 = bnd4.getFactory();
    let transaccion = await factory4.newTransaction(Liquidacion,'casarDesajuste');
    transaccion.precio = precio;
    await bnc4.submitTransaction(transaccion);
}

//Transfiere los pagos de la ronda de mercado a los datos personales de los proumidores
async function liquidarDesajuste(){
    process.stdout.write('Liquidando las transacciones de eliminación de desajuste...')
    var d = new Date();
    let factory4 = bnd4.getFactory();
    let transaccion = await factory4.newTransaction(Liquidacion,'transacciones');
    transaccion.mercado = false;
    await bnc4.submitTransaction(transaccion);
}

//Esta funcion crea las transacciones con la utility de los usuarios que no hayan encontrado
//con quien hacer el intercambio dentro de la microrred
async function transaccionesUtility(){
    process.stdout.write('Realizando transacciones con la utility de desajuste...')
    var d = new Date();
    let precios = await bnc2.query('precios',{timestamp : d })
    var precioMayorista = precios[0];
    let factory4 = bnd4.getFactory();
    let transaccion = await factory4.newTransaction(Liquidacion,'transaccionesUtility');
    transaccion.precioCompra = precioMayorista.precioCompra;
    transaccion.precioVenta = precioMayorista.precioVenta;
    await bnc4.submitTransaction(transaccion);
}
```

Listado A.37: ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía

```
//Esta función pone todos los valores de apto a false, dejando la situación para iniciar el mercado nuevamente
async function realizarPagos(){
    process.stdout.write('Transfiriendo pagos y reputación de la ronda al usuario...')
    let participantRegistry1 = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
    let participantRegistry4 = await bnc4.getParticipantRegistry('org.liquidacion.Prosumer')
    var array1=[]
    var array4=[]
    let prosumers = await bnc1.query('prosumers');
    for (let n = 0; n < prosumers.length; n++) {
        let prosum1= prosumers[n];
        let prosum4 = await participantRegistry4.get(prosum1.IDsecreto)
        prosum1.Coins = prosum1.Coins + prosum4.CoinsPeriodo;
        prosum1.Reputacion = prosum1.Reputacion + prosum4.ReputacionPeriodo;
        array1.push(prosum1);
    }
    await participantRegistry1.updateAll(array1)
    //Lo hacemos en dos bucles para asegurarnos que suma las cantidades correspondientes
    //en la cadena 1 antes de borrarlas en la 4
    for (let n = 0; n < prosumers.length; n++) {
        let prosum1= prosumers[n];
        let prosum4 = await participantRegistry4.get(prosum1.IDsecreto)
        prosum4.CoinsPeriodo=0;
        prosum4.ReputacionPeriodo=0;
        array4.push(prosum4);
    }
    await participantRegistry4.updateAll(array4);
    console.log(' hecho'.green)
    await marca_tiempo();
    console.log('Finaliza la fase a posteriori de la ronda '+ronda1)
}
//Desde esta función se conecta y se desconectan las conexiones y se llama a la creación de los usuarios
async function antes(){
    await marca_tiempo();
    var d = new Date();
    var hora_actual = d.getUTCHours();
    var minuto_actual = d.getMinutes()+1;
    ronda = (hora_actual+':'+minuto_actual)
    console.log('Se inicia la fase previa de la ronda de las '+ronda)
    await iniciarMercado1();
}
//Todas las funciones que se ejecutan al finalizar el periodo de mercado, una vez leídos los datos del contador
async function despues(){
    await marca_tiempo();
    var d = new Date();
    var hora_actual = d.getUTCHours();
    var minuto_actual = d.getMinutes()-1;
    ronda1 = (hora_actual+':'+minuto_actual)
    console.log('Se inicia la fase a posteriori de la ronda de las ' +ronda1)
    await insertarDatosContador();
    await liquidarTransacciones();
}
```

Listado A.38: ClienteGestor.js – Programa en node.js que conecta al gestor del sistema con las cadenas de bloques y da las ordenes necearias para crear una subasta de energía

```
1  //cada vez que se entra en la función se comprueban los precios mayoristas
2  //y se ejecuta la fase previa del mercado
3  //35 segundos después se ejecuta la fase a posteriori
4  async function temporizar(){
5      var d = new Date();
6      let precios = await bnc2.query('precios',{timestamp : d });
7      if (precios.length<3){
8          await descargayleePrecioMayorista();
9      }
10     var espera2 = (35)*1000
11     setTimeout(await despues,espera2);
12     await antes();
13 }
14
15 //Se conecta a las redes BC con las tarjetas instaladas correspondientes,
16 //descarga los precios de la web de ESIOS
17 //y se espera a llegar al segundo 30 del siguiente minuto para ejecutar la funcion previa,
18 //y despues del periodo correspondiente temporizar
19 async function principal(){
20     bnd1 = await bnc1.connect(cardCreacion);
21     bnd2 = await bnc2.connect(cardMercado);
22     bnd3 = await bnc3.connect(cardRegistro);
23     bnd4 = await bnc4.connect(cardLiquidacion);
24     var d = new Date();
25     let precios = await bnc2.query('precios',{timestamp : d });
26     if (precios.length==0){
27         await descargayleePrecioMayorista();
28     }
29     await marca_tiempo();//Esto solo es para indicar a qué hora se ejecuta el programa, para
           saber cuanto hay que esperar a que empiece
30     console.log('Se ha conectado el cliente del gestor del sistema. ')
31     var espera1 = 60;
32     var espera_inicio= setInterval(function(){
33         var fecha = new Date();
34         var segundos = fecha.getSeconds();
35         espera1 = 30-segundos;// Se determina el tiempo que falta para llegar al próximo
               segundo 30
36         if (espera1<0){espera1=60+espera1;}
37         if (espera1==9) process.stdout.clearLine();
38         process.stdout.write(espera1+' segundos para iniciar la subasta entre prosumidores\r'
               );
39         if (espera1==0){
40             process.stdout.clearLine();
41             antes();
42             setInterval(temporizar,60*1000);
43             clearInterval(espera_inicio);}
44     },1000);
45
46 }
47
48 principal();
```

A.5 Cliente en Node.js de los prosumidores

Listado A.39: ClienteProsumer.js – Programa en node.js que conecta al usuario prosumidor con las cadenas de bloques, inserta los datos de los parámetros eléctricos y las ofertas para la subasta de energía

```
ID='0';
let Mercado = 'org.mercado';
let Registro = 'org.registro'
var ID = process.argv;
const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;
var jf = require('jsonfile')

// se crean las conexiones a cada una de las redes de blockchain
const bnc1 = new BusinessNetworkConnection();
const bnc2 = new BusinessNetworkConnection();
const bnc3 = new BusinessNetworkConnection();
const bnc4 = new BusinessNetworkConnection();

//cada vez que se inicia el mercado se ejecuta la función ofertas
bnc2.on('event', (event) => {
    if (event.iniciado2) ofertas();
});
//Esta es una funcion auxiliar a la que llamo para que muestre por pantalla
//la hora exacta a la que se realiza la llamada
async function marca_tiempo(){
    var fecha= new Date();
    var hora_actual = fecha.getUTCHours();
    var minuto = fecha.getMinutes();
    var segundo =fecha.getSeconds();
    process.stdout.write(hora_actual+':'+minuto+':'+segundo+' --> ');
}
//extrae los datos de los correspondientes archivos JSON generados por el contador y los
    introducen en la red BC de registro de datos eléctricos
async function insertarAcumulados(){
    await marca_tiempo();
    console.log('Se insertan los datos del contador')
    let participantRegistry1 = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
    let prosum1= await participantRegistry1.get(ID)
    let IDsecreto = prosum1.IDsecreto;
    let assetRegistry = await bnc3.getAssetRegistry('org.registro.Acumulados');
    let acumulados = await assetRegistry.get(IDsecreto.concat('_acum'));
    var archivo = await jf.readFileSync('../cards/'+ID+'/acumulado.json');
    acumulados.generacion=archivo.generacion;
    acumulados.consumo=archivo.consumo;
    await assetRegistry.update(acumulados);
    let factory =bnd3.getFactory();
    let actualizarDiferencia = factory.newTransaction(Registro,'actualizarDiferencia');
    actualizarDiferencia.prosum= factory.newRelationship(Registro, 'Prosumer', IDsecreto);
    await bnc3.submitTransaction(actualizarDiferencia);
}
//Estas dos funciones extraen los datos del archivos JSON generad
//por el EMS y los introduce en la cadena de mercado
async function insertarOferta(IDsecreto){
    let factory =bnd2.getFactory();
    let insertarOferta = factory.newTransaction(Mercado,'insertarOferta');
    var archivo = await jf.readFileSync('../cards/'+ID+'/oferta.json');
    insertarOferta.cantidad=archivo.cantidad;
    insertarOferta.precio=archivo.precio;
    insertarOferta.owner = IDsecreto;
    await bnc2.submitTransaction(insertarOferta);
}
```

Listado A.40: ClienteProsumer.js – Programa en node.js que conecta al usuario prosumidor con las cadenas de bloques, inserta los datos de los parámetros eléctricos y las ofertas para la subasta de energía

```
1
2  //Esta función comprueba si el usuario ha sido admitido en el mercado y en caso afirmativo
       procede a llamar a las funciones para insertar las pujas y/o ofertas
3  async function ofertas(){
4      let participantRegistry1 = await bnc1.getParticipantRegistry('org.creacion.Prosumer')
5      let prosum1= await participantRegistry1.get(ID)
6      let IDsecreto = prosum1.IDsecreto;
7      if (prosum1.Apto){
8          await marca_tiempo();
9          console.log('El usuario ha sido aceptado en el mercado, se inserta la oferta')
10         await insertarOferta(IDsecreto);
11         //console.log('Insertadas las ofertas de compra/venta de energía')
12     }
13     else {await marca_tiempo();console.log('El usuario no puede participar en el mercado')}
14 }
15
16 //Se conecta a las redes BC con las tarjetas instaladas correspondientes, y se espera a llegar
       al segundo 0 para comenzar a introducir los datos del contador
17 //Cada 60 segundos se vuelven a insertar
18 async function conectar(){
19     let cardCreacion=ID.concat('.creacion@creacion');
20     let cardMercado= ID.concat('.mercado@mercado');
21     let cardRegistro=ID.concat('.registro@registro');
22     let cardLiquidacion = ID.concat('.liquidacion@liquidacion');
23     bnd1 = await bnc1.connect(cardCreacion);
24     bnd2 = await bnc2.connect(cardMercado);
25     bnd3 = await bnc3.connect(cardRegistro);
26     bnd4 = await bnc4.connect(cardLiquidacion);
27     marca_tiempo();
28     console.log('Registro correcto en la red con el ID '+ID)
29     var fecha = new Date();
30     var segundos = fecha.getSeconds();
31     espera=(60-segundos)*1000;
32     setTimeout(function(){insertarAcumulados();setInterval(insertarAcumulados,60000)},espera);
33 }
34
35 //Pide el ID por teclado, pero en principio no debería pasar nunca, sólo para pruebas
36 async function leerID (){
37     console.log("Escribe tu ID de usuario");
38     var stdin = process.openStdin();
39     stdin.addListener("data", function(d) {
40         ID=d.toString().trim();
41         conectar()
42   });
43 }
44
45 //si no se encuentra el archivo dentro de la carpeta del prosumer con el ID del usuario,
46 //se pide que se inserte. Si se encuentra se usa
47 async function inicio(){
48     if (ID.length < 3) {
49         ID=0
50         leerID()
51     }
52     else{ID = ID[2]
53     conectar()}
54 }
55
56 inicio();
```

Apéndice B

Configuración y monitorización de *Hyperledger Fabric*

En el presente anexo se describen las técnicas seguidas para la instalación y configuración de *Hyperledger Fabric* en su versión 1.2, *Hyperledger Composer* en su versión 0.20, así como las herramientas empleadas para la monitorización de los recursos consumidos por los contenedores en los que se levantan.

B.1 Descarga e instalación de *Hyperledger Composer*

El primer paso de la instalación de *Hyperledger Composer* consiste en la instalación de los requisitos previos, paso muy sencillo en sistemas operativos Linux, pues únicamente hay que descargar y ejecutar un archivo que contiene la batería de comandos que descargan todas las dependencias necesarias:

```
$ curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh
$ chmod u+x prereqs-ubuntu.sh
$ ./prereqs-ubuntu.sh
```

El siguiente paso consiste en la instalación de la herramienta *composer-cli*, que contiene todas las operaciones esenciales para interaccionar con *Hyperledger Composer*:

```
$ npm install -g composer-cli@0.20
```

Además, instalamos también la herramienta *Playground*, que facilita el desarrollo y prueba de aplicaciones en la red *blockchain* directamente en el navegador web:

```
$ npm install -g composer-playground@0.20
```

El último paso de la instalación consiste en la descarga del directorio con las herramientas para descargar *Hyperledger Fabric* y su posterior instalación:

```
$ mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers
$ curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/
fabric-dev-servers.tar.gz
$ tar -xvf fabric-dev-servers.tar.gz

$ cd ~/fabric-dev-servers
$ export FABRIC_VERSION=hlfv12
$ ./downloadFabric.sh
```

B.2 Control del entorno de desarrollo

Para levantar los contenedores que contienen los nodos, la base de datos, el ordenante y la autoridad certificadora de *Hyperledger Fabric*, se deben ejecutar los siguientes comandos:

```
$ cd ~/fabric-dev-servers
$ export FABRIC_VERSION=hlfv12
$ ./startFabric.sh
$ ./createPeerAdminCard.sh
```

Posteriormente se pueden parar y volver a levantar todas las veces que se quiera mediante los comandos `./stopFabric.sh` y `./startFabric.sh` respectivamente. Cuando se quieran eliminar, se hace con el comando `./teardownFabric.sh`.

Para iniciar la herramienta *Playground* y poder desarrollar y probar aplicaciones directamente en el navegador web, se emplea el comando `composer-playground`, que monta la herramienta en el puerto 8080 del equipo en el que se ejecuta, pero se puede modificar este puerto añadiendo al comando anterior el sufijo `-p` seguido por el número de puerto en el que se quiera montar la herramienta.

Para eliminar todos los contenedores montados en el equipo se ejecutan los siguientes comandos:

```
$ docker kill $(docker ps -q)
$ docker rm $(docker ps -aq)
$ docker rmi $(docker images dev-* -q)
```

B.3 Modificación del número de nodos por defecto

Para configurar una red de *Hyperledger Fabric* con nodos situados en más de un equipo físico se emplea la guía redactada por [45], en la que desarrolla la edición de los archivos necesarios para instalar tres nodos en dos equipos en la versión 1.0 de *Hyperledger Fabric* y se adapta a las variaciones que se han introducido en la versión 1.2 para hacerlo compatible con todo el código desarrollado a lo largo del trabajo. En el primer equipo se instala la autoridad certificadora, el ordenante y dos nodos, y en el segundo equipo se instala un nodo, se puede ser el esquema del sistema completo en la figura 5.6. No se necesita instalar ningún componente adicional, únicamente se modifican los archivos de *Hyperledger Fabric* ya situados en el equipo.

1. En primer lugar, se copian los archivos `docker-compose.yml` y `startFabric.sh`. Las copias se renombran `docker-compose-peer2.yml` y `startFabric-peer2.sh`.

2. En el archivo `crypto-config.yaml` se cambia el número de nodos, de 1 a 3.

3. Al variar la información de configuración del archivo `crypto-config.yaml`, la carpeta con las claves ya no son válidas, por ello, se debe generar una nueva carpeta `crypto-config`. Esto se realiza mediante las siguientes órdenes en el terminal de linux:

   ```
   $ cd "$(dirname "$0")"
   $ ./cryptogen generate --config=./crypto-config.yaml
   $ export FABRIC_CFG_PATH=$PWD
   $ ./configtxgen -profile ComposerOrdererGenesis -outputBlock ./composer-genesis.block
   $ ./configtxgen -profile ComposerChannel -outputCreateChannelTx ./composer-channel.tx
   -channelID composerchannel
   ```

4. Para configurar de manera correcta a la autoridad certificadora, se debe copiar el archivo con la clave privada, situado en: `/crypto-config/peerOrganizations/org1.example.com/ca/`, y cambiarlo por que el que hubiera anteriormente en el archivo `docker-compose.yml`. Además, en

```
OrdererOrgs:
- Name: Orderer
  Domain: example.com
  Specs:
    - Hostname: orderer
PeerOrgs:
- Name: Org1
  Domain: org1.example.com
  Template:
    Count: 3
  Users:
    Count: 0
```

Figura B.1: Modificación del archivo `crypto-config.yaml`.

este mismo archivo, se añade la configuración para añadir un nodo adicional (se denomina peer1) y su correspondiente base de datos (couchdb1):

```
  peer1.org1.example.com:
  container_name: peer1.org1.example.com
  image: hyperledger/fabric-peer:1.2.1
  environment:
  - CORE_LOGGING_LEVEL=debug
  - CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
  - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
  - CORE_PEER_ID=peer1.org1.example.com
  - CORE_PEER_ADDRESS=peer1.org1.example.com:7051
  - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
  - CORE_PEER_LOCALMSPID=Org1MSP
  - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
  - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
  - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start
  ports:
  - 8051:7051
  - 8053:7053
  volumes:
  - /var/run/:/host/var/run/
  - ./:/etc/hyperledger/configtx
  - ./crypto-config/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/msp:
  /etc/hyperledger/peer/msp
  - ./crypto-config/peerOrganizations/org1.example.com/users:/etc/hyperledger/msp/users
  depends_on:
  - orderer.example.com
  - couchdb1

  couchdb1:
  container_name: couchdb1
  image: hyperledger/fabric-couchdb:0.4.10
```

5. En el archivo creado anteriormente `docker-compose-peer2.yml`, se inserta la configuración

para crear los contenedores correspondientes al otro equipo, tanto el nodo como su correspondiente base de datos (peer2 y couchdb2):

```
peer2.org1.example.com:
container_name: peer2.org1.example.com
image: hyperledger/fabric-peer:1.2.1
environment:
- CORE_LOGGING_LEVEL=debug
- CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
- CORE_PEER_ID=peer2.org1.example.com
- CORE_PEER_ADDRESS=peer2.org1.example.com:7051
- CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
- CORE_PEER_LOCALMSPID=Org1MSP
- CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
- CORE_LEDGER_STATE_STATEDATABASE=CouchDB
- CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb2:5984

working_dir: /opt/gopath/src/github.com/hyperledger/fabric
command: peer node start
ports:
- 9051:7051
- 9053:7053
volumes:
- /var/run/:/host/var/run/
- ./:/etc/hyperledger/configtx
- ./crypto-config/peerOrganizations/org1.example.com/peers/peer2.org1.example.com/msp:/etc
/hyperledger/peer/msp
- ./crypto-config/peerOrganizations/org1.example.com/users:/etc/hyperledger/msp/users
depends_on:
- couchdb2

couchdb2:
container_name: couchdb2
image: hyperledger/fabric-couchdb:0.4.10
ports:
- 7984:5984
environment:
DB_URL: http://localhost:7984/member_db
```

6. Una vez creados los archivos con la configuración de todos los contenedores que van a crearse, se procede a modificar los archivos para iniciarlos. En el archivo `startFabric.sh`, hay que añadir la información para unir el nodo 1 al canal creado por el nodo 0. Para ello se añade el siguiente código:

```
# Create the channel
docker exec peer0.org1.example.com peer channel create -o orderer.example.com:7050
-c composerchannel -f /etc/hyperledger/configtx/composer-channel.tx

# Join peer0.org1.example.com to the channel.
docker exec -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/
msp" peer0.org1.example.com peer channel join -b composerchannel.block

# Create the channel
docker exec -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp"
peer1.org1.example.com peer channel fetch config -o orderer.example.com:7050 -c composerchannel

# docker exec peer1.org1.example.com peer channel create -o orderer.example.com:7050
-c composerchannel -f /etc/hyperledger/configtx/composer-channel.tx

# Join peer1.org1.example.com to the channel.
docker exec -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp"
peer1.org1.example.com peer channel join -b composerchannel_config.block
```

En el archivo `startFabric-peer2.sh` se añade la información para unir el nodo 2 al canal creado por el nodo 0:

```
# Create the channel
```

```
docker exec -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com
/msp" peer2.org1.example.com peer channel fetch config
-o orderer.example.com:7050 -c composerchannel

# Join peer1.org1.example.com to the channel.
docker exec -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com
/msp" peer2.org1.example.com peer channel join -b composerchannel_config.block
```

7. También se debe modificar el archivo `createPeerAdminCard.sh`, para que el administrador tenga la información de los nodos que va a haber en la red y de esta forma permitirles conectarse a ella. Además, en este archivo, se debe modificar la clave privada que había sido actualizada en el paso número 3, y si está desactualizada no permitirá actuar como administrador. El archivo queda de la siguiente forma:

```
Usage() {
        echo ""
        echo "Usage: ./createPeerAdminCard.sh [-h host] [-n]"
        echo ""
        echo "Options:"
        echo -e "\t-h or --host:\t\t(Optional) name of the host to specify
        in the connection profile"
        echo -e "\t-n or --noimport:\t(Optional) don't import into card store"
        echo ""
        echo "Example: ./createPeerAdminCard.sh"
        echo ""
        exit 1
}

Parse_Arguments() {
        while [ $# -gt 0 ]; do
        case $1 in
        --help)
        HELPINFO=true
        ;;
        --host | -h)
        shift
        HOST="$1"
        ;;
        --noimport | -n)
        NOIMPORT=true
        ;;
        esac
        shift
        done
}

HOST=192.168.1.168
Parse_Arguments $@

if [ "${HELPINFO}" == "true" ]; then
Usage
fi

# Grab the current directory
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

if [ -z "${HL_COMPOSER_CLI}" ]; then
HL_COMPOSER_CLI=$(which composer)
fi

echo
# check that the composer command exists at a version >v0.16
COMPOSER_VERSION=$("${HL_COMPOSER_CLI}" --version 2>/dev/null)
COMPOSER_RC=$?

if [ $COMPOSER_RC -eq 0 ]; then
AWKRET=$(echo $COMPOSER_VERSION | awk -F. '{if ($2<20) print "1"; else print "0";}')
if [ $AWKRET -eq 1 ]; then
```

```
echo Cannot use $COMPOSER_VERSION version of composer with
fabric 1.2, v0.20 or higher is required
exit 1
else
echo Using composer-cli at $COMPOSER_VERSION
fi
else
echo 'No version of composer-cli has been detected, you need to install
composer-cli at v0.20 or higher'
exit 1
fi

cat << EOF > DevServer_connection.json
{
        "name": "hlfv1",
        "x-type": "hlfv1",
        "x-commitTimeout": 300,
        "version": "1.0.0",
        "client": {
                "organization": "Org1",
                "connection": {
                        "timeout": {
                                "peer": {
                                        "endorser": "300",
                                        "eventHub": "300",
                                        "eventReg": "300"
                                },
                                "orderer": "300"
                        }
                }
        },
        "channels": {
                "composerchannel": {
                        "orderers": ["orderer.example.com"],
                        "peers": {
                                "peer0.org1.example.com": {},
                                "peer1.org1.example.com": {},
                                "peer2.org1.example.com": {}
                        }
                }
        },
        "organizations": {
                "Org1": {
                        "mspid": "Org1MSP",
                        "peers": ["peer0.org1.example.com","peer1.org1.example.com",
                        "peer2.org1.example.com"],
                        "certificateAuthorities": ["ca.org1.example.com"]
                }
        },
        "orderers": {
                "orderer.example.com": {
                        "url": "grpc://${HOST}:7050"
                }
        },
        "peers": {
                "peer0.org1.example.com": {
                        "url": "grpc://${HOST}:7051"
                },
                "peer1.org1.example.com": {
                        "url": "grpc://${HOST}:8051"
                },
                "peer2.org1.example.com": {
                        "url": "grpc://192.168.1.169:9051"
                }
        },
        "certificateAuthorities": {
                "ca.org1.example.com": {
                        "url": "http://${HOST}:7054",
                        "caName": "ca.org1.example.com"
                }
```

```
        }
}
EOF

PRIVATE_KEY="${DIR}"/composer/crypto-config/peerOrganizations/org1.example.com/users/
Admin@org1.example.com/msp/keystore/
4eeb40687c081040aa45161a6523104bcbbf03a35e4ea55ee94526e9ce5ca544_sk
CERT="${DIR}"/composer/crypto-config/peerOrganizations/org1.example.com/users/
Admin@org1.example.com/msp/signcerts/Admin@org1.example.com-cert.pem

if [ "${NOIMPORT}" != "true" ]; then
CARDOUTPUT=/tmp/PeerAdmin@hlfv1.card
else
CARDOUTPUT=PeerAdmin@hlfv1.card
fi

"${HL_COMPOSER_CLI}"  card create -p DevServer_connection.json -u PeerAdmin -c "${CERT}"
-k "${PRIVATE_KEY}" -r PeerAdmin -r ChannelAdmin --file $CARDOUTPUT

if [ "${NOIMPORT}" != "true" ]; then
if "${HL_COMPOSER_CLI}"  card list -c PeerAdmin@hlfv1 > /dev/null; then
"${HL_COMPOSER_CLI}"  card delete -c PeerAdmin@hlfv1
fi

"${HL_COMPOSER_CLI}"  card import --file /tmp/PeerAdmin@hlfv1.card
"${HL_COMPOSER_CLI}"  card list
echo "Hyperledger Composer PeerAdmin card has been imported, host of fabric specified as
'${HOST}'"
rm /tmp/PeerAdmin@hlfv1.card
else
echo "Hyperledger Composer PeerAdmin card has been created, host of fabric specified as
'${HOST}'"
fi
```

8. Hay que modificar la IP del equipo que va a alojar al nodo 2 en el archivo anterior, para que el administrador del sistema lo reconozca y permita su conexión. Además, la IP del primer equipo también debe ser modificada, en lugar de dejar textitlocalhost como viene por defecto, hay que poner la IP correcta, para que el segundo equipo la reconozca.

9. Con todos estos archivos modificados, se copia la carpeta completa y se pega en el equipo que va a alojar al nodo número 2.

10. Se inician los contenedores del primer equipo y se crea la tarjeta de administrador de la red en este equipo mediante las siguientes órdenes en el terminal de linux estando alojados en la carpeta:

```
$ ./teardownFabric.sh && ./startFabric.sh && ./createPeerAdminCard.sh
```

11. En el segundo equipo basta con ejecutar el siguiente comando dentro de la carpeta copiada:

```
$ ./startFabric-peer2.sh
```

Si se quiere modificar el número de nodos una vez creada la red y en funcionamiento con los nodos previamente configurados, se sigue la guía de [46] en la cual se puede comprobar que la principal diferencia con respecto al método para modificar el número de nodos antes de ponerlos en funcionamiento, radica en que no se modifica la carpeta completa con la información criptográfica de la red (`crypto-config`), simplemente se añade la información correspondiente al nuevo nodo mediante el comando:

```
./cryptogen extend --config=./crypto-config.yaml
```

Además, se debe añadir la información del nuevo nodo (especialmente importante la IP del equipo donde se va a alojar) en el archivo `createPeerAdminCard.sh`, y crear una nueva tarjeta de administrador de la red que permita la conexión a este nuevo nodo.

B.4 Monitorización de recursos empleados por contenedores

La monitorización de los recursos consumidos en cada uno de los contenedores *Docker* del equipo, se emplea la herramienta *cAdvisor* desarrollada por *Google* [44], y que reporta la información en tiempo real de las transferencias de datos, almacenamiento, uso de memoria RAM y procesador, entre otros datos de interés. Para utilizar esta herramienta, debe montarse en su propio contenedor mediante los siguientes comandos:

```
sudo docker run \
--volume=/:/rootfs:ro \
--volume=/var/run:/var/run:ro \
--volume=/sys:/sys:ro \
--volume=/var/lib/docker/:/var/lib/docker:ro \
--volume=/dev/disk/:/dev/disk:ro \
--publish=8080:8080 \
--detach=true \
--name=cadvisor \
google/cadvisor:latest
```

Esta batería de comandos, además de crear el contenedor lo publica en el puerto 8080 del equipo en el que se ejecuta. Este puerto se puede modificar si ya está en uso en el equipo.

Apéndice C

Pliego de condiciones

Durante el desarrollo del presente trabajo de fin de máster ha sido necesario el empleo de las diversas herramientas de desarrollo, equipos y recursos para dar forma a una plataforma de intercambio de energía entre usuarios prosumidores basada en la tecnología *blockchain*. Todas estas herramientas y recursos se detallan en este pliego de condiciones junto con su coste estimado en la sección C.1.

Además, se ha realizado un pliego de condiciones en el que se detalla cómo se realizaría el despliegue de la plataforma de intercambios de energía en una microrred compuesta por 50 usuarios prosumidores en un entorno urbano en el que en la sección C.2. Se incluye el coste estimado de este despliegue, diferenciando entre los usuarios prosumidores (usuario final del servicio) y el gestor de la microrred.

C.1 Presupuesto de realización del TFM

En este apartado se detallan las herramientas y recursos empleados para desarrollar este trabajo fin de máster junto con sus costes monetarios (C.1), donde se puede observar el coste nulo de las herramientas de desarrollo, al haber tratado en todo momento de emplear software de código abierto con el objetivo de reducir el coste final y permitir en un futuro el despliegue a un entorno real con un presupuesto moderado.

Concepto	Descripción	Coste/Ud	Coste total
Ordenador 1	Equipo con procesador intel i7-8550U, 8 GB RAM y SSD de 250 GB	1000 €	1000 €
Ordenador 2	Equipo con procesador intel i5-6400, 16 GB RAM y SSD 500 GB	1000 €	1000 €
GNU/Linux	Sistema operativo	—	—
Conexión a Internet	500 Mbps simétricos (3 meses)	50 €	150 €
Hyperledger Fabric	Plataforma de desarrollo de redes blockchain	—	—
Hyperledger Composer	Herramientas de desarrollo de aplicaciones para Hyperledger Fabric	—	—
Visual Studio Code	Entorno de desarrollo de código integrado	—	—
LaTeX	Procesador de textos	—	—
Node.js	Entorno de ejecución de JavaScript en el servidor	—	—
cAdvisor	Monitorizador de recursos	—	—
Trabajo alumno	360 horas de investigación y desarrollo de código	15 €	5400 €
Trabajo tutor	20 horas de dedicación del tutor	30 €	600 €
Total			**8150 €**

Tabla C.1: Presupuesto de las herramientas y recursos empleados para el desarrollo de la plataforma de intercambios de energía entre usuarios prosumidores en una microrred.

C.2 Pliego de condiciones plataforma intercambios de energía

Se ha realizado una estimación de los recursos necesarios para la implementación de la plataforma de intercambios de energía en una microrred urbana (C.2.1) así como una estimación de los costes de cada unos de estos recursos, poder evaluar si sería rentable su instalación. Se diferencia entre los costes que debería asumir el usuario prosumidor (sección C.2.2) y los costes que debería asumir el gestor del sistema (sección C.2.3), que podría ser una cooperativa de los prosumidores de la microrred u otra entidad (con o sin ánimo de lucro).

Se debe tener en cuenta que, en Octubre de 2019, no existe una normativa específica que regule y controle el desarrollo e implantación de las microrredes eléctricas en España y, por lo tanto, el despliegue de este proyecto no podría realizarse si no fuera en forma de proyecto piloto con fines de investigación, hasta que la normativa no lo contemple.

C.2.1 Pliego de condiciones generales

Para el despliegue de la plataforma de intercambios de energía se deben realizar las instalaciones de equipos que se detallan a continuación:

- Instalación de un contador en el punto de conexión de la microrred a la red general de distribución, que sea capaz de medir de manera bidireccional la cantidad de energía que se intercambia con esta, con el objetivo de facturar esta energía a la empresa comercializadora correspondiente.
- Instalación de al menos tres equipos capaces de gestionar los intercambios de energía, para que en caso de que ocurriera un fallo en uno de ellos, bien sea técnico o malintencionado, el sistema siguiera funcionando de manera correcta, y con almacenamiento suficiente para que pueda guardar la base de datos, siempre creciente, durante varios años.
- Comunicación entre el contador del punto de conexión de la microrred y los equipos del gestor del sistema mediante cable Ethernet categoría 5e.
- Instalación de un equipo de bajas prestaciones (se ha desarrollado la plataforma de manera que requiera un bajo consumo de recursos) para cada uno de los usuarios prosumidores de la microrred. Placa de desarrollo tipo Raspberry Pi.
- Comunicación mediante cable Ethernet categoría 5e entre el equipo de cada uno de los usuarios prosumidores de la microrred y su contador inteligente.
- De manera opcional, cada usuario prosumidor podrá añadir en su equipo un dispositivo de almacenamiento en el que se descargaría la base de datos completa, lo que haría aumentar su confianza en el sistema al poder garantizar que no se realizan modificaciones en los datos almacenados.

Se presupone que todos los usuarios disponen de conexión a Internet de alta velocidad, al ser una plataforma de intercambios planteada para un entorno urbano, por lo tanto, no sería necesario el despliegue de cable de telecomunicaciones entre los diferentes usuarios de la plataforma.

C.2.2 Costes prosumidor

Para la realización de estimación de los costes de adquisición e instalación de los dispositivos necesarios para unirse a la plataforma de intercambios de energía de la microrred, se ha presupuesto que el usuario

prosumidor dispone de una conexión a Internet con un ancho de banda suficiente, y que, o bien se une únicamente como comprador de energía, o dispone de un generador y/o almacenador de electricidad ya instalado y operativo en su propiedad.

Concepto	Descripción	Coste/Ud	Coste total
Equipo	Raspberry Pi 4 – 1 GB RAM, quad-core CPU	40 €	40 €
Comunicación contador	15 metros cable Ethernet cat. 5e	9 €	9 €
Instalación	2 horas de mano de obra	50 €	100 €
Total			**149 €**

Tabla C.2: Presupuesto de la adquisición e instalación del equipo necesario para que el prosumidor pueda conectarse a la plataforma de intercambios de energía.

C.2.3 Costes gestor del sistema

Los costes para el gestor del sistema son los derivados de la adquisición de los equipos de altas prestaciones necesarios, su instalación y puesta en marcha de los programas de gestión de la plataforma en su interior y el despliegue de cable de comunicaciones desde los equipos hasta el contador inteligente de la microrred.

Se incluye, en este presupuesto del gestor del sistema, los costes del colegio de ingenieros industriales para el visado del proyecto, que son los especificados para el despliegue de redes de telecomunicaciones.

Concepto	Descripción	Coste/Ud	Coste total
Equipos	Tres equipos con procesador de altas prestaciones disco duro de 10 TB y 16 GB de memoria RAM	1600 €	4800 €
Comunicación contador	Tres rollos de 50 metros cable Ethernet cat. 5e	15 €	45 €
Coste COIIM	Costes visado proyecto	45 €	45 €
Mano de obra	20 horas para la instalación de los equipos	50 €	1000 €
Total			**5890 €**

Tabla C.3: Presupuesto de la adquisición e instalación de los equipos necesarios para la gestión de la plataforma de intercambios de energía

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá