Universidad de Valladolid

Escuela de Ingeniería Informática

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Despliegue, configuración y análisis de una red blockchain privada aplicada a IoT

Trabajo de Fin de Grado

Alumno:
Javier Gómez Herrero

Tutor:
César Llamas Bello

A María

Agradecimientos

A mi hermana María y a mis padres Antonio y Amara, a quienes debo todo.
A toda mi familia, por haber estado siempre conmigo.
A los amigos que me han traído hasta aquí desde Miranda, Palencia, Castrillo y Valladolid.
A los amigos que me ha regalado la universidad y me han acompañado estos años.
A los compañeros y amigos del grado, más que necesarios en este camino.
A Columna, Maite, Borja, Sergio, Gamarra, Imaz, Motro, Víctor y en especial CarlosNoé, claves para llegar aquí.
Y a Carlota y sus mofletes.

Resumen

Con la información como la moneda de cambio en el mundo digital, la gestión de la misma es una parte clave de cualquier sistema, y cada situación requiere distintas soluciones. En el caso del Internet de las Cosas (IoT por sus siglas en inglés) ya existen formas de gestión de datos, pero las posibilidades ofrecidas por la tecnología blockchain hacen de este enfoque uno bastante atractivo. Este proyecto explora la aplicación de esta tecnología a IoT. Tras un estudio del estado del arte de ambos campos, se realiza el despliegue de una red blockchain privada mediante la plataforma Ethereum sobre la que se simulan dispositivos IoT. Se realiza una evaluación del rendimiento del sistema y se discuten sus resultados. Tras la ejecución de este proceso, se extraen las conclusiones del trabajo realizado. Estas conclusiones tienen en cuenta tanto el proceso de despliegue como el mantenimiento de la red blockchain privada y su rendimiento. De esta manera se determina la viabilidad de todo el proceso y las posibilidades del trabajo futuro.

Palabras clave: Blockchain, Internet de las Cosas, Ethereum, Nodo, Contrato Inteligente, Transacción.

Abstract

As data is the currency in the digital world, its management is a key part of any system, and every situation requires different solutions. In the case of the Internet of Things (IoT) several paradigms already exist, but the possibilities offered by the blockchain technology make this approach a rather attractive one. This project explores the applications of this technology to the IoT. After a study of the state of the art of both fields, a private blockchain network is deployed through the use of Ethereum and IoT devices are simulated on it. Afterwards, a performance evaluation is conducted and its results discussed. Once this process is finished, conclusions are be drawn. This conclusions take into account the process of deployment as well as maintenance of the private blockchain network and its performance. This way the viability of the whole process is determined, as well as the possibility of future work.

Keywords: Blockchain, Internet of Things, Ethereum, Node, Smart Contract, Transaction.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Este proyecto se lleva a cabo como trabajo final del grado en ingeniería informática de la universidad de Valladolid. Se ejecuta a lo largo de unos dos meses, durante los cuales se llevan a cabo diversas tareas que más adelante serán especificadas.

En la actualidad, la convergencia de tecnologías como el Internet de las Cosas (IoT) y la cadena de bloques (blockchain) ha abierto nuevas posibilidades y desafíos en el ámbito de la industria y la informática. Esas posibilidad y desafíos se entrelazan con la presencia de la actual revolución industrial, llamada industria 4.0 y caracterizada por la interconexión de dispositivos inteligentes y la digitalización de los procesos industriales. Ésta es la tendencia actual hacia la automatización y el intercambio de datos, con objetivos también como optimizar la eficiencia, la transparencia y la seguridad en las operaciones.
El IoT, que se refiere a la red de dispositivos físicos conectados que recopilan y comparten datos, proporciona una infraestructura sólida para capturar información en tiempo real y tomar decisiones basadas en datos. Sin embargo, a medida que el IoT se expande, los datos se crean en cantidades exponencialmente mayores y su gestión plantea desafíos en niveles como la seguridad, la integridad o la confiabilidad entre otros.

Aquí entra en juego la tecnología blockchain, que ofrece una forma descentralizada y segura de almacenar, verificar y compartir información de manera transparente. La cadena de bloques, originalmente desarrollada para respaldar las transacciones de criptomonedas, tiene el potencial de abordar los desafíos del IoT al proporcionar un registro inmutable y distribuido de los datos.

Se vuelve ahora a la Industria 4.0, creadora del concepto de la "fábrica inteligente". Una de las características de este tipo de fábricas es la toma de decisiones de manera descentralizada por parte de los sistemas que las componen. Esta descentralización no solo es útil a la hora de la toma de decisiones, si no que otros procesos pueden aprovechar sus características, como el almacenaje de datos o la interconexión de dispositivos IoT.
Así, este proyecto se centra en la investigación y aplicación de las tecnologías blockchain en el contexto del IoT. Se busca comprender las tecnologías y su estado del arte, aprender

a trabajar con ellas, desarrollar una plataforma de prueba sobre la que ejecutar pruebas y analizar el rendimiento y evaluar los resultados para extraer conclusiones sobre su funcionamiento, viabilidad y escalabilidad.

1.2. Motivación

A nivel personal, la motivación tras la realización de éste proyecto emana, en primera instancia, de la necesidad de realizarlo como colofón del grado de ingeniería informática. Esto se une a la posibilidad de profundizar en un área de la informática relativamente nueva, con un potencial aún por explotar y que ofrece un abanico de posibilidades cada vez más amplio conforme nos adentramos en él. Además, tras la cobertura mediática recibida en los últimos años por el mundo de las criptodivisas, una de las posibilidades que ofrece la tecnología blockchain, y el paso por la asignatura *Tecnologías Distribuidas y Blockchain*, perteneciente al máster en investigación informática de la universidad de Valladolid, la posibilidad de continuar investigando en esta tecnología se presenta como una gran oportunidad.

En el nivel de la propuesta del trabajo y del interés de la temática, la motivación detrás de esta investigación surge de la creciente relevancia que tienen tanto la tecnología blockchain como IoT en el panorama actual. Ambas tecnologías han demostrado tener un gran potencial en diversos ámbitos, y su integración puede generar soluciones innovadoras y beneficios significativos en la industria y otros sectores.
En primer lugar, la tecnología blockchain ofrece características como la inmutabilidad, la transparencia y la descentralización, lo que la sitúa como una herramienta ideal para garantizar la seguridad y la integridad de los datos. Esto es especialmente relevante en un entorno IoT, donde la cantidad de dispositivos y la generación masiva de datos pueden plantear desafíos en términos de confianza y protección de la información.
En segundo lugar, IoT está revolucionando la forma en que interactuamos con el mundo, al permitir la interconexión de dispositivos y la recopilación de datos en tiempo real. Sin embargo, también presenta desafíos en cuanto a la seguridad, la privacidad y la escalabilidad, precisamente puntos fuertes de la tecnología blockchain. La combinación de IoT y blockchain puede abordar estos desafíos al proporcionar un marco seguro y confiable para el intercambio y la gestión de datos en entornos distribuidos.
En tercer lugar, la cuarta revolución industrial, conocida como la Industria 4.0, necesita nuevas tecnologías y formas de manejar, generar y almacenar datos. Esto se debe a que en esta revolución el intercambio de datos de manera masiva y la automatización son puntos clave y conforme se desarrolla los sistemas actuales pueden no solo tener problemas con el nuevo paradigma si no incluso quedarse obsoletos.

La necesidad de explorar y comprender cómo aplicar blockchain a IoT también surge de la búsqueda de soluciones que permitan una mayor confianza, transparencia y trazabilidad en los sistemas IoT. Además, se busca impulsar la investigación y el desarrollo de nuevas aplicaciones y servicios que aprovechen el potencial sin explotar de esta combinación. De esta manera, las prácticas de investigación se plantean como una oportunidad para profundizar en el conocimiento de estas tecnologías y explorar su aplicación conjunta primero en ámbitos reducidos y después, mediante escalabilidad, hasta la Industria 4.0.

Por último, y como debe ser en todo proyecto de investigación, la motivación subyacente es también contribuir al avance de la ciencia y la tecnología, y poder aportar soluciones innovadoras que impulsen el progreso y mejoren la calidad de vida de las personas.

1.3. Objetivos

El objetivo de esta investigación es explorar y estudiar la aplicación de la tecnología blockchain en el contexto de IoT. Este objetivo se puede separar en varios hitos de menor envergadura que forman parte del propósito de este proyecto. Los hitos, presentados a continuación, hacen del trabajo, su planificación y su organización tareas más sencillas.

- Investigación de las tecnologías blockchain e IoT:
  Se llevará a cabo una revisión de literatura científica, artículos, *papers* y otros recursos para comprender las características, aplicaciones, desafíos y estado del arte de estas tecnologías. De esta manera se realiza la introducción en los campos relevantes para la investigación y se garantiza la navegación con cierta facilidad entre los recursos disponibles. También se facilita la comprensión de los campos y sus tendencias y se consigue saber cuales son los siguientes pasos a dar, es decir, como comenzar a ejecutar el siguiente hito.

- Estudio de las plataformas existentes:
  Para aplicar ambas tecnologías es necesario en primer lugar que puedan ser interconectadas. Para facilitar ese proceso existen diversas plataformas que deberán ser consideradas. Así, se analizarán aquellas disponibles en el mercado. Se evaluarán sus características, requisitos, funcionalidades y casos de uso así como nuestras posibilidades y necesidades para seleccionar la plataforma más adecuada para la investigación.

- Implantación de la plataforma seleccionada:
  Se procederá al despliegue de la red blockchain mediante la plataforma elegida, siguiendo los requisitos y procedimientos necesarios para ésta y que previamente habrán sido recopilados mediante su documentación y otros materiales disponibles en la red. Esta red se configurará en un entorno simulado mediante varios ordenadores proporcionados para la investigación, que harán las veces de nodos de la red y de dispositivos IoT.

- Simulación de la red IoT:
  Una vez desplegada la red privada, se configurará la simulación. Esto incluye definir y ejecutar métodos para realizar las siguientes tareas:

  - Almacenaje de datos en la cadena de bloques.
  - Acceso a los datos almacenados en la cadena de bloques.
  - Simulación de los dispositivos IoT.
  - Automatización de la interacción entre los dispositivos IoT y la cadena de bloques.

- Medición de prestaciones:
  Se realizarán mediciones sobre la simulación, con el fin de evaluarlas más adelante. Estas incluirán:
  - Tiempos de ejecución de transacciones.
  - Coste de ejecución de las transacciones.
  - Gasto energético.
  - Almacenamiento consumido.
- Evaluación de prestaciones:
  Tras las mediciones mencionadas en el punto anterior, se analizarán los resultados para determinar distintas características de la red. Los tiempos de ejecución de las transacciones permitirán saber si la velocidad de almacenamiento en la blockchain y su interacción con ella es viable para dispositivos IoT. El gasto energético indicará aproximadamente el gasto que supondría una inversión en esta tecnología. El coste de las transacciones indicaría si una red con un algoritmo de decisión basado en PoW (Proof of Work) es una buena opción. No sólo esto, si no que midiendo estos valores con diferentes números de nodos mineros y simuladores de dispositivos IoT se puede analizar la escalabilidad del sistema.
- Documentación:
  Se realizará el presente informe, documentando el proceso de la investigación con sus diferentes etapas, así como otros puntos de interés como la motivación de su desarrollo, sus objetivos, la planificación o el diseño del sistema entre otros.

Al lograr estos hitos se espera obtener un conocimiento sólido sobre la integración de blockchain e IoT, sobre la plataforma elegida para realizar la implantación y sobre la forma de manejar y analizar redes blockchain. Es decir, tras la ejecución de este proyecto se pretende no sólo conocer el análisis de la red para obtener las conclusiones finales, si no también poder replicar el sistema con facilidad en casos con diferentes requisitos, conocer su funcionamiento y las razones del mismo, y poder operarlo con facilidad.

1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

- **Capítulo 1: Introducción**
  Este capítulo describe aspectos necesarios para la comprensión de posteriores apartados. Estos aspectos incluyen el contexto sobre el que se sitúa la investigación, la motivación tras ella y sus objetivos. Incluye también el presente apartado especificando la estructura del documento.
- **Capítulo 2: Estado del Arte**
  Este capítulo describe el análisis previo de las tecnologías involucradas en el proyecto y su estado del arte. También incluye el análisis y selección del software sobre el que se sustentará el sistema.

- **Capítulo 3: Desarrollo**
  Este capítulo describe los pasos llevados a cabo para ejecutar el proyecto. se divide en las siguientes secciones:

  - Plan:
    La planificación previa al comienzo del trabajo, incluyendo el marco de trabajo, los tiempos propuestos y el presupuesto del proyecto.
  - Análisis:
    El desglose de los requisitos necesarios para ejecutar el proyecto, así como otra información relacionada al mismo.
  - Diseño:
    El diseño del sistema a desarrollar mediante recursos como diagramas SysML y UML.
  - Recursos:
    Los recursos a utilizar. Esto incluye hardware como los ordenadores o el cableado y software como la plataforma blockchain.
  - Implementación:
    El proceso de implementación del sistema, desde la instalación de los recursos necesarios hasta su puesta en marcha funcional.
  - Pruebas:
    Las puebas realizadas para asegurar el correcto funcionamiento del sistema previo a su análisis.
  - Seguimiento:
    El desglose de actividades realizadas, especialmente aquellas no ajustadas a la planificación inicial.

- **Capítulo 4: Resultados**
  Este capítulo describe los resultados obtenidos del análisis del sistema, además de proporcionar una discusión sobre ellos.

- **Capítulo 5: Conclusiones**
  En este capítulo se presentan las conclusiones extraídas del trabajo realizado y los resultados obtenidos, y hace las veces de cierre del informe mediante el broche final de las opciones del trabajo futuro.

- **Anexo A: Manuales**
  Incluye manuales de mantenimiento, de instalación, despliegue, y de uso.

- **Anexo B: Archivos**
  Incluye el contenido de diferentes archivos y *scripts* de interés para la instalación, despliegue y uso del sistema.

- **Anexo C: Resumen de enlaces adicionales**
  Incluye enlaces de interés sobre el proyecto, como el repositorio de código.

Capítulo 2

Estado del Arte

El primer paso a la hora de comenzar con el proyecto de investigación es realizar un análisis detallado de las tecnologías involucradas: el Internet de las Cosas (IoT) y la cadena de bloques (blockchain). Este análisis busca proporcionar comprensión de las tecnologías que respaldan el proyecto y establecer las bases para su aplicación . A través de la exploración de los conceptos fundamentales y los avances más recientes, se podrán identificar sus fortalezas y limitaciones así como comprender su potencial en conjunto.

Para realizar este análisis se comenzarán conociendo los inicios de cada tecnología para después continuar explorando su evolución histórica, sus funcionamientos básicos y sus aplicaciones prácticas en diversos campos. Poco a poco se profundizará en cada uno de los campos hasta conocer sus tecnologías y aplicaciones más punteras, es decir, su estado del arte.

Tras este análisis, es decir, tras conocer el estado del arte de ambas tecnologías, se seleccionarán una serie de plataformas que muestren el mayor potencial a la hora de integraras. Tras ello, se compararán y se elegirá una de ellas, que será la utilizada en adelante para llevar a cabo las siguientes fases del proyecto.

2.1. Internet of Things

El concepto del Internet de las Cosas se origina a finales de la década de 1990. Su creador, Kevin Ashton era un pionero británico en el campo de la tecnología y su trabajo con la identificación por radiofrecuencia (RFID por sus siglas en inglés) le llevó a trabajar para el MIT. Fué él quien acuñó el término Ïnternet of Things.[en] el año 1999. Ashton visualizó un mundo en el que los objetos cotidianos estarían conectados a Internet, permitiendo la recopilación y el intercambio de datos de forma automática.

A medida que avanzaba el siglo XXI, el IoT experimentó un crecimiento significativo en términos de avances tecnológicos y adopción generalizada. Los avances en la miniaturización de los componentes electrónicos, el desarrollo de tecnologías de comunicación inalámbrica y la mejora de la capacidad de procesamiento contribuyeron al rápido desarrollo del IoT, que probaba ser un paradigma más que útil en un mundo cada vez más

hiperconectado.

Figura 2.1: Representación del Internet de las Cosas [40]

Fue esa creciente conectividad junto con el acceso generalizado a Internet los que permitieron que dispositivos previamente aislados se integraran en una red global interconectada. Los sensores, actuadores e incluso dispositivos empotrados se volvieron más accesibles y económicos, lo que facilitó su implementación en una gran cantidad de procesos, sistemas e industrias, así como en la vida cotidiana.

Así, esta tecnología se basa en la interconexión de dispositivos físicos, conocidos como çosas". Estos dispositivos, que pueden variar desde electrodomésticos y wearables hasta maquinaria industrial y sensores ambientales, están equipados con una amplia variedad de sensores, actuadores y capacidades de comunicación. Es de esta manera como pueden recopilar datos de su entorno y enviarlos a través de redes de comunicación a sistemas centrales para su procesamiento y análisis. Aunque esta última mención de la conexión a un sistema principal emana del paradigma centralizado habitual, también pueden comunicarse con otros dispositivos colindantes junto con los que operan y funcionan. Es, de hecho, esa capacidad de comunicarse entre ellos formando un sistema descentralizado una de las bases de este proyecto de investigación.

Su funcionamiento se basa en cuatro componentes principales:

- **Dispositivos**
  Recopilan los datos mediante sensores, y actuán en función de las instrucciones que dictan los actuadores.

- **Conectividad**
  Permite la comunicación entre dispositivos y sistemas centrales u otros dispositivos. Esta se puede llevar a cabo mediante tecnologías como Wi-Fi, Bluetooth o RFID entre otras.

- **Nube**
  Originalmente, proporciona el almacenamiento de información y la capacidad de procesamiento necesarios para el análisis y la gestión de los datos generados.
- **Aplicaciones**
  Permiten a los usuarios interactuar con los dispositivos y acceder a los datos recopilados a través de interfaces intuitivas.

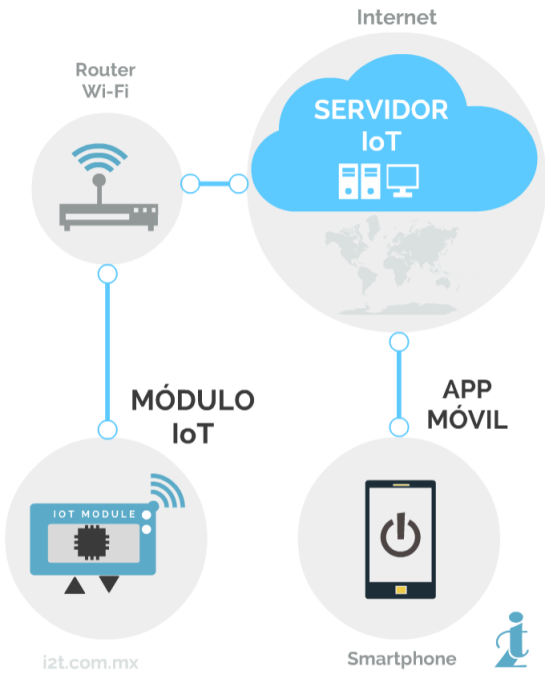

Figura 2.2: Representación del Funcionamiento del Internet de las Cosas [38]

2.1.1. Aplicaciones

Como se ha podido ver hasta ahora, desde su concepción en la década de 1990 el IoT ha evolucionado hasta convertirse en una tecnología onmipresente, generadora de enormes cantidades de información. De hecho, esta proliferación de dispositivos y sensores inteligentes ha llevado al desarrollo de multitud de aplicaciones. Estas abarcan una amplia gama de sectores, como el hogar inteligente, la salud, la agricultura, la logística y la industria. Por ejemplo, en la automatización de los hogares, conocida como domótica, los dispositivos IoT pueden controlar la iluminación, la seguridad y la temperatura, brindando comodidad y eficiencia energética.

En especial, sus aplicaciones en la industria son parcialmente responsables de la llegada de la ya comentada Industria 4.0. Entre estos avances encontramos:

- **Monitorización y mantenimiento predictivo**
  El IoT permite monitorizar constantemente los activos y la maquinaria en tiempo

real. Los sensores integrados en los equipos recopilan datos sobre su rendimiento, consumo e incluso vibraciones y otros parámetros relevantes. Estos datos son analizados y permiten detectar posibles fallos, realizando así un mantenimiento predictivo que evita interrupciones no planificadas y gasto económico y temporal.

- **Optimización de la cadena de suministro**
  El IoT facilita el seguimiento y la gestión de los productos a lo largo de toda la cadena de suministro. Mediante sensores y etiquetas como las de tipo RFID es posible conocer datos como la ubicación, la temperatura, la humedad u otras variables de los productos en tiempo real. Esto mejora la visibilidad de la cadena, permite una planificación más precisa y eficiente, y reduce el riesgo de pérdidas y deterioro de productos.

- **Fabricación inteligente**
  La integración del IoT en las factorías permite la creación de entornos de producción altamente automatizados y conectados. Los sensores en las máquinas y los equipos recopilan datos sobre el rendimiento, la calidad, el consumo de energía y otros parámetros. Estos datos se utilizan para optimizar los procesos, realizar ajustes en tiempo real y mejorar la eficiencia y la calidad de los productos fabricados.

- **Gestión energética y sostenibilidad**
  El IoT se utiliza en la monitorización y control de los sistemas de gestión energética en los edificios industriales. Los sensores y los dispositivos conectados permiten la recopilación de datos sobre el consumo energético, la iluminación, la climatización y otros parámetros. Estos datos se utilizan para identificar oportunidades de ahorro energético, optimizar el uso de los recursos y reducir el impacto ambiental.

- **Seguridad y prevención de riesgos**
  El IoT desempeña un papel crucial en la seguridad y la prevención de riesgos en la industria. Los sensores y las cámaras conectadas permiten la monitorización de condiciones peligrosas como fugas o incendios, y la alerta temprana de situaciones de riesgo. Esto contribuye a garantizar un entorno de trabajo seguro y a minimizar los accidentes y las pérdidas humanas y materiales.

2.1.2. Estado Actual

La rápida evolución de la que hablamos y que el IoT ha experimentado a lo largo de sus dos décadas de vida trae consigo constantes desafíos, habitualmente relacionados con la seguridad, la privacidad y la interoperabilidad. Por ejemplo los estándares y protocolos de seguridad, como el encriptado de datos y la autenticación, se han vuelto fundamentales para proteger la integridad de la información en este entorno y así poner solución al desafío de la seguridad. Sin embargo no es el único. A continuación se definen los desafíos, oportunidades y tendencias que conforman su estado más avanzado, su Estado del Arte.

- **Desafíos**

  - Consumo energético:
    La eficiencia energética es necesaria para prolongar la vida útil de las baterías e incluso los propios dispositivos, además de reducir el impacto ambiental.

- Escalabilidad:
  La gestión de las grandes cantidades de datos generadas plantea desafíos en términos de almacenamiento y procesamiento.
- Interoperabilidad:
  Dado que no existen estándares comunes y las tecnologías disponibles abarcan un abanico de posibilidades muy amplio, la interoperabilidad entre dispositivos y sistemas IoT no siempre es fácil o incluso posible.
- Privacidad y seguridad:
  La protección de datos personales y la seguridad de los dispositivos y redes son un desafío constante a lo largo de todo el mundo digital. IoT no es una excepción, y además las redes pueden plantear diversas vulnerabilidades en función de cada dispositivo, su hardware y su software.

- **Oportunidades**

  - Mejora de la eficiencia operativa:
    El IoT permite la monitorización en tiempo real de activos y procesos, lo que facilita la optimización y mejora de la eficiencia en diversos sectores, como la industria, la logística y la energía. Como hemos visto en el anterior apartado, la industria, entre otros sectores, ya ha visto mejoras. Sin embargo aún existen más y mejores posibilidades.
  - Nuevos modelos de negocio:
    La conectividad y la recopilación de datos en el IoT ofrecen oportunidades para el desarrollo de nuevos modelos de negocio, como los servicios basados en suscripción y los modelos de pago por uso.
  - Transformación digital:
    La adopción de esta tecnología implica una transformación digital en las organizaciones, que pueden aprovechar el potencial de los datos generados por los dispositivos IoT para tomar decisiones más informadas y mejorar la experiencia del cliente.

- **Tendencias**

  - Computación en la nube y *edge computing*:
    Estos conceptos proporcionan capacidad de almacenamiento y procesamiento de datos cercana a los dispositivos, mejorando la eficiencia y reduciendo la latencia.
  - Expansión de la conectividad:
    El número de dispositivos conectados sigue aumentando exponencialmente, abarcando desde dispositivos domésticos hasta infraestructuras industriales complejas.
  - Inteligencia Artificial y análisis de datos:
    La combinación de IoT con técnicas de Inteligencia Artificial (IA) y análisis de datos permite extraer información valiosa y tomar decisiones automatizadas en tiempo real.
  - Seguridad y privacidad:
    La seguridad y la privacidad son preocupaciones crecientes en el ámbito del

IoT. Se está invirtiendo en el desarrollo de soluciones para proteger los datos y garantizar la integridad de los sistemas IoT.

Como se puede ver, la actualidad del IoT refleja un panorama en constante evolución, con tendencias hacia una mayor conectividad, inteligencia artificial, computación en la nube y preocupación por la seguridad y la privacidad. A pesar de los desafíos existentes, el IoT ofrece numerosas oportunidades para mejorar la eficiencia operativa, impulsar nuevos modelos de negocio y lograr una transformación digital en diversos sectores.

2.2. Blockchain

En 1982 el criptógrafo David Chaum publica la disertación "*Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups*. Es aquí donde se propone por primera vez un protocolo basado en una cadena de bloques. A lo largo de los años se ve cierto desarrollo en este área, como el llevado a cabo por la compañía Surety, cuyos creadores introdujeron árboles de hashes en el diseño de la cadena de bloques. No es, sin embargo, hasta 2008 cuando se conceptualiza la primera blockchain descentralizada. Ésta conceptualización es llevada a cabo por parte de Satoshi Nakamoto, que es el pseudónimo utilizado por el creador (o creadores) de Bitcoin, la primera criptodivisa.

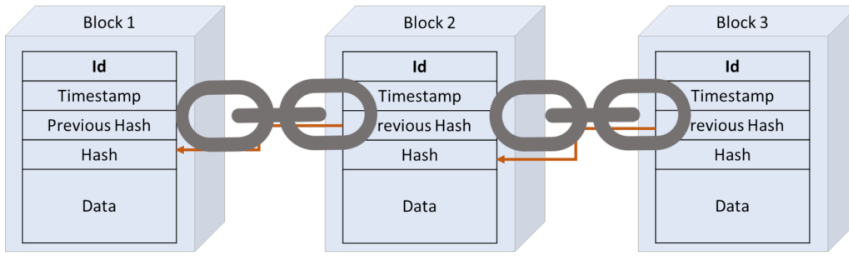

Figura 2.3: Representación de una cadena de bloques [3]

La mejora de diseño de la cadena de bloques introducida por Nakamoto le permite operar sin necesitar que los bloques sean firmados por entidades de confianza. Esto se hace introduciendo dificultad a la hora de crear bloques, lo que añade estabilidad la velocidad a la que dichos bloques se incorporan a la cadena. Esta validación se realiza mediante un método de tipo *hashcash*, e incluye el momento exacto de la validación.

Para comprender las bases y el funcionamiento de esta tecnología, la podemos separar en los siguientes puntos:

1. **Estructura**
   Si es llamada cadena es porque conceptualmente es similar a una. Se compone de una serie de bloques que almacenan información, y cada uno de estos bloques forma un eslabón de la cadena. Para mantener su orden e integridad, cada bloque almacena el *hash* del bloque anterior.

2. **Descentralización**
   Esta cadena de bloques opera mediante una red descentralizada, sin una autoridad de confianza que valide las operaciones llevadas a cabo en ella. Son los nodos los que deben trabajar en conjunto para validad y verificar cada transacción.

3. **Consenso**
   Para trabajar juntos, los nodos deben utilizar algoritmos de consenso. Un método común es Prueba de Trabajo (PoW por sus siglas en inglés), donde los nodos compiten en la resolución de un problema complejo para validar un bloque. Cada vez que se lleva a cabo una validación, ésta se propaga por la red y se pasa a la creación del siguiente nodo. Cuando dos validaciones chocan, es decir, se propagan por la red a la vez al haber sido generadas en un intervalo de tiempo corto, aquella realizada más pronto es al que prevalece.

4. **Seguridad**
   Una vez que un bloque ha sido establecido, es casi imposible de modificar, ya que mantiene información del nodo previo. Para modificar la cadena de bloques, un atacante necesitaría más de la mitad de la capacidad computacional de la red.

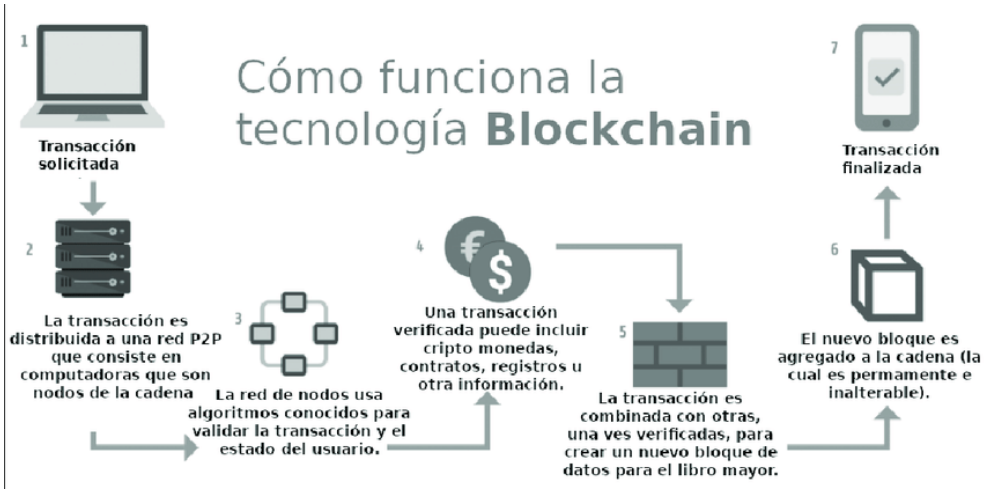

Figura 2.4: Representación del funcionamiento de una cadena de bloques [13]

Tras de la creación de Bitcoin, la tecnología blockchain ha experimentado un rápido desarrollo y una creciente adopción en diversos sectores. A medida que la comprensión y la confianza en la tecnología aumenta, se exploran nuevas aplicaciones y se realizan avances significativos. Es, sin embargo, una tecnología relativamente joven, cuya aplicación no alcanza aún las dos décadas.

Una de las principales evoluciones en la tecnología blockchain fue la introducción de Ethereum en 2015. En este caso ya no se está hablando de una criptodivisa como Bitcoin o muchas otras, sino como una plataforma que permite ejecutar contratos inteligentes y aplicaciones descentraliadas.

Además, se han desarrollado diferentes tipos de blockchains [46], como las públicas, privadas y de consorcio, cada una con sus propias características y casos de uso específicos. También se han propuesto mejoras y soluciones para abordar desafíos como la escalabilidad, la privacidad y la interoperabilidad en la tecnología blockchain.

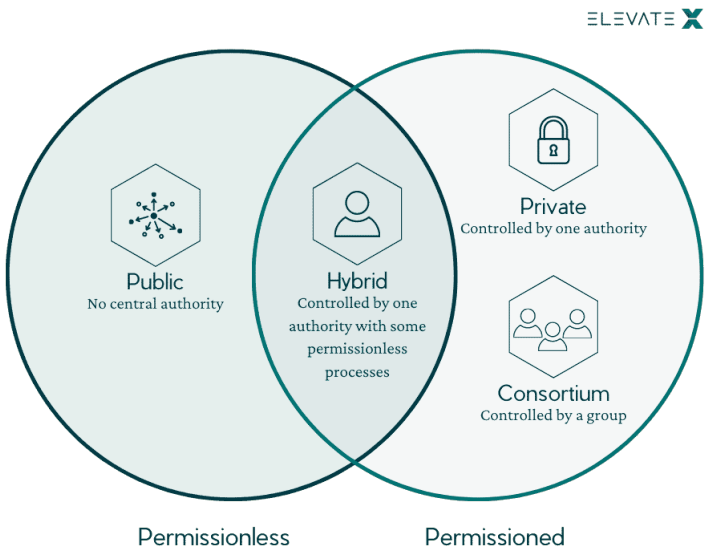

Figura 2.5: Tipos de blockchain [46]

La tecnología blockchain ha evolucionado desde sus inicios con Bitcoin hacia una variedad de herramientas versátiles con múltiples aplicaciones en diversos sectores. Su funcionamiento descentralizado, transparente e inmutable la convierte en una tecnología prometedora para garantizar la seguridad, la confianza y la eficiencia en diferentes ámbitos. Aunque la mayor parte de su uso (y la percepción pública de esta tecnología) tiene que ver con las criptodivisas y un mercado especulativo que ha sido el centro de muchas noticias, como veremos a continuación, sus aplicaciónes llegan mucho más allá.

2.2.1. Aplicaciones

Aunque las aplicaciones de esta tecnología varían y pueden referirse incluso a la domótica de una casa inteligente, sus aplicaciones más importantes y prometedoras se refieren al ámbito de la industria:

- **Gestión de la cadena de suministro**
  La cadena de suministro es un área donde blockchain ha encontrado una amplia aplicación. Mediante el uso de registros inmutables y transparentes, blockchain permite rastrear y auditar el movimiento de productos desde su origen hasta su destino. De esta manera se mejora la trazabilidad, la autenticidad y la seguridad de cada

producto, así como la eficiencia en la gestión de inventario. Esto además se traduce en una reducción de costos.

- **Gestión de derechos de propiedad y activos**
  Mediante la *tokenización* de activos y su inclusión en la blockchain, se pueden representar y transferir derechos de propiedad de manera digital. Esto puede ser aplicado a propiedades inmobiliarias, obras de arte, vehículos y otros activos, simplificando el proceso de compra, venta y transferencia de propiedad. Además, la tecnología blockchain garantiza la autenticidad y evita fraudes en la gestión de activos. Esto ya se ha visto mediante los tokens no fungibles (NFTs por sus siglas en inglés). Sin embargo, como ha ocurrido con las criptodivisas, al situarse en un mercado no regulado, han sido presa de la especulación.

- **Gestión de identidad**
  Mediante el uso de blockchain, es posible crear identidades digitales seguras y verificables, lo que mejora la seguridad y la privacidad en las transacciones y la comunicación digital. Además, blockchain permite a los individuos tener un mayor control y propiedad sobre sus datos personales.

- **Pagos y transacciones financieras**
  Probablemente la aplicación más destacada, ya que la tecnología blockchain ha demostrado su utilidad en la simplificación y agilización de los procesos de pago y transacciones financieras en la industria. Los contratos inteligentes y las criptomonedas basadas en blockchain permiten realizar transacciones directas, seguras y rápidas sin intermediarios, lo que reduce los costos y los tiempos de liquidación. Uno de los desafíos respecto a este punto es, al igual que con el segundo, la inexistencia de un mercado regulado y por ende la posibilidad de que cada criptactivo pueda ver su valor profundamente influenciado por las corrientes de un mercado especulativo.

La blockchain ofrece múltiples aplicaciones en la industria, desde la gestión de la cadena de suministro y la tokenización de activos, hasta la gestión de identidad, los pagos y transacciones financieras. Estas aplicaciones proporcionan mayor eficiencia, transparencia, seguridad y confianza en los procesos industriales, impulsando la transformación y la innovación en cualquier sector susceptible de utilizar esta tecnología.

2.2.2. Estado Actual

Actualmente esta tecnología muestra un panorama dinámico y en constante evolución, con avances significativos en diversos aspectos como es común en un campo de tan corta edad. Algunos de los puntos clave que reflejan el estado actual de la tecnología son los siguientes:

- Escalabilidad:
  Uno de los desafíos principales en el ámbito de la blockchain es la escalabilidad, es decir, la capacidad de procesar un alto volumen de transacciones de manera rápida y eficiente. Actualmente se están explorando diferentes enfoques para mejorar la escalabilidad de las blockchains, como la implementación de protocolos de consenso más eficientes, la adopción de soluciones de capa 2 y el desarrollo de blockchains de alto rendimiento.

- Privacidad y Confidencialidad:
  Éstas son áreas de gran importancia a lo largo de todo el mundo digital. En el caso de la blockchain, se presentan mejoras importantes. Aunque la cadena de bloques es transparente y auditable, existen casos en los que se requiere la protección de datos sensibles. En este sentido, se han propuesto soluciones como las blockchains basadas en contratos inteligentes confidenciales y los sistemas de mezcla de transacciones para preservar la privacidad de los usuarios.

- Interoperabilidad:
  La interoperabilidad entre diferentes blockchains y sistemas es otro desafío a abordar en el estado del arte de la tecnología blockchain. Actualmente se están desarrollando estándares y protocolos para facilitar la comunicación y la transferencia de valor entre distintas plataformas blockchain, lo que permitiría la creación de ecosistemas interconectados y potenciaría la adopción de la tecnología a mucha mayor escala.

- Sostenibilidad y Eficiencia Energética:
  El sistema de consenso basado en PoW requiere gran capacidad computacional y un alto consumo de energía. Esto hace que las plataformas blockchain basadas en PoW generen altos gastos monetarios y energéticos, poniendo en duda sus sostenibilidad. Por estas razones se están investigando y desarrollando soluciones para reducir el impacto ambiental, como el uso de algoritmos de consenso más eficientes y la migración hacia modelos de consenso de prueba de participación (Proof of Stake). Éste último es el caso de Ethereum, una de las plataformas más extendidas.

- Adopción Empresarial:
  En los últimos años, ha habido un creciente interés y adopción de la tecnología blockchain por parte de empresas y organizaciones en diferentes industrias. Se están realizando numerosas pruebas de concepto y proyectos piloto para explorar el potencial de blockchain en áreas como las planteadas en el anterior punto, además de otras como por ejemplo la atención médica. Esta adopción empresarial es clave para impulsar la madurez y la aplicación práctica de la tecnología.

El estado del arte de la tecnología blockchain muestra un progreso continuo en las áreas clave descritas. A medida que se abordan los desafíos existentes y se realizan nuevos avances, se espera que la tecnología blockchain siga evolucionando y desempeñe un papel cada vez más importante en la transformación digital de diversos sectores.

2.3. Blockchain aplicado a IoT

La combinación de blockchain e Internet of Things, las dos tecnologías previamente analizadas, despierta gran interés debido a su potencial para abordar desafíos relacionados con la seguridad, la privacidad, la escalabilidad y la confianza en entornos de IoT. Como prueba de ello se verán a continuación algunas de las aplicaciones existentes de blockchain a IoT, así como su potencial para impulsar la innovación en este campo.

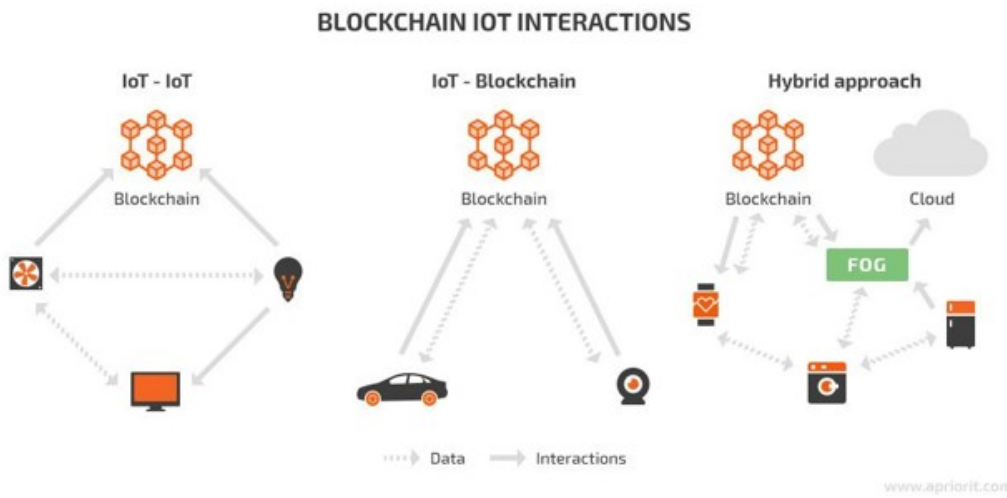

Figura 2.6: Formas de implementar blockchain en IoT [29]

Hasta el momento se han desarrollado varias aplicaciones de blockchain aplicado a IoT en diferentes áreas. Un ejemplo es el ámbito de la gestión de la cadena de suministro, donde la combinación de blockchain e IoT permite rastrear y auditar el movimiento de productos en tiempo real, mejorando la transparencia y la seguridad de la cadena de suministro. Asimismo, se han propuesto soluciones basadas en blockchain para la gestión de la energía en redes inteligentes, donde los dispositivos IoT pueden intercambiar energía de manera segura y eficiente.

El potencial de blockchain en IoT es prometedor y se han identificado varias ventajas clave. En primer lugar, como se ha visto, la naturaleza inmutable y transparente de blockchain proporciona un registro confiable y auditable de los datos generados por los dispositivos IoT, lo que mejora la integridad de los datos y fomenta la confianza entre las partes involucradas. Además, la descentralización de blockchain permite la eliminación de intermediarios, lo que agiliza y simplifica los procesos de transacción y verificación en entornos de IoT.

Aún por encima de esto, la capacidad de blockchain para garantizar la seguridad y la privacidad de los datos es particularmente relevante en IoT, donde la confidencialidad de la información es crucial. En el caso de redes públicas, mediante el uso de técnicas criptográficas y contratos inteligentes, blockchain puede proteger los datos generados por los dispositivos IoT y garantizar que solo las partes autorizadas tengan acceso a ellos. En el caso de redes privadas el uso de técnicas criptográficas puede ser de menor preocupación, ya que el acceso a los datos del público general puede ser restringido mediante otros métodos.

Es importante destacar que el campo de blockchain aplicado a IoT aún está en evolución y existen desafíos por superar. Algunos de estos desafíos incluyen la escalabilidad de las soluciones blockchain en entornos de IoT masivos, la gestión eficiente de los recursos

computacionales y energéticos, así como la interoperabilidad entre diferentes plataformas y dispositivos. Sin embargo, se están llevando a cabo investigaciones y desarrollos activos para abordar estos desafíos y permitir la adopción generalizada de blockchain en IoT.

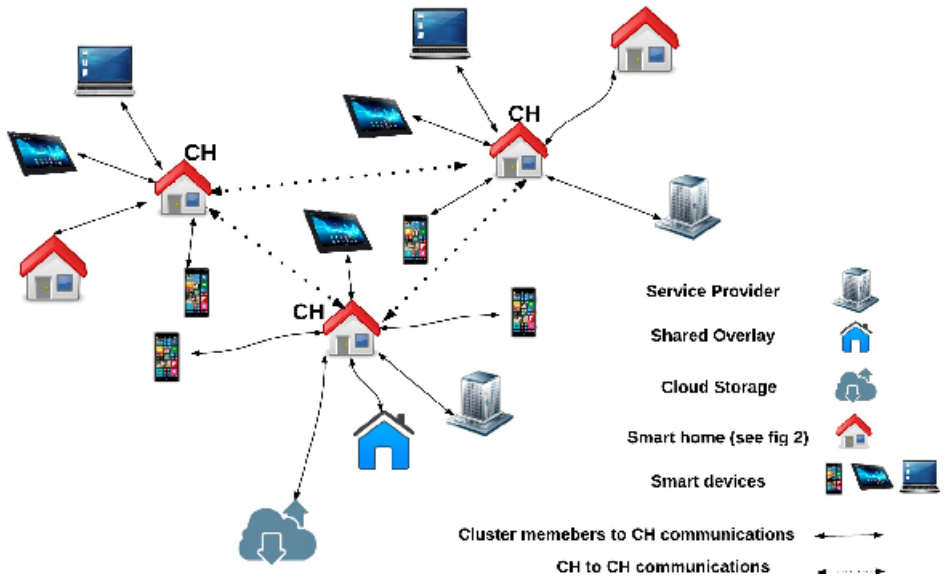

Figura 2.7: Ejemplo de blockchain aplicado a IoT en una *smart home* [10]

La combinación de blockchain e IoT presenta un amplio potencial en diversas áreas, como se ha mencionado en los casos de la gestión de la cadena de suministro y la gestión de la energía. A medida que se superen los desafíos técnicos y se realicen más investigaciones, se espera que blockchain desempeñe un papel fundamental en la innovación y transformación de los sistemas de IoT.

2.4. Plataformas

Para llevar a cabo el proyecto es necesario hacerlo mediante una plataforma que permita implementar una red blockchain sobre IoT, además de otras consideraciones que se abordan más adelante dentro de esta sección. Por ello, tras el estudio se las tecnologías se definen las siguientes como potenciales plataformas:

- **Hyperledger Fabric + IBM Watson IoT Platform**
  Hyperledger Fabric es un framework *open source* de la fundación Linux que permite crear aplicaciones blockchain personalizadas e integrarlas con sistemas IoT. IBM Watson IoT Platform es una plataforma IoT que permite la integración con Hyperledger Fabric y permite así la creación de aplicaciones blockchain personalizadas.

- **IOTA**

Plataforma blockchain sin minería diseñada para permitir la comunicación y transacciones de datos seguras entre dispositivos IoT.

- **Ethereum**
  Plataforma blockchain descentralizada. Permite creación de contratos y aplicaciones distribuidas. Compatible con el protocolo MQTT, lo que la hace adecuada para su integración con sistemas IoT.

- **Filament**
  Plataforma blockchain diseñada para IoT que permite la creación de dispositivos IoT seguros y conectados a las red blockchain.

- **IoTeX**
  Plataforma blockchain centrada en la seguridad y privacidad de los dispositivos IoT. Permite la creación de contratos inteligentes y la integración con otros sistemas blockchain.

- **R3 Corda**
  Framework blockchain *open source* que permite la creación de aplicación blockchain personalizadas. Es compatible con sistemas IoT y permite comunicación y transacciones de datos de forma segura.

Para elegir entre ellas, se tendrán en cuenta los siguientes aspectos:

- **Integración con IoT**
  La plataforma debe, de manera nativa o mediante cualquier otro método, poder trabajar con dispositivos IoT.

- **Madurez**
  La plataforma no debe encontrarse en fases de desarrollo, si no que debe ofrecer un entorno ya funcional y que permite ejecutar el proyecto.

- **Sencillez**
  La plataforma debe poder desplegarse y manejarse sin necesidad de formación excesivamente experta en la misma.

- **Automatización**
  La plataforma debe permitir automatizar el proceso de interacción con los dispositivos IoT.

- **Red privada**
  La plataforma debe permitir implementar una red privada sobre la que operar.

- **Documentación**
  La plataforma debe disponer de amplia documentación mediante la que estudiar y comprender su funcionamiento, además de servir como recurso en caso de fallos inesperados o debidos a inexperiencia.

- **Comunidad**
  La plataforma debe contar con una comunidad de desarrollo amplia. De esta manera se asegura variedad y cantidad de recursos online así como foros a los que poder acceder como último recurso.

- **Disponibilidad**
  La plataforma debe poder ser utilizada. En el mejor de los casos, de manera gratuita y sin licencia, aunque la posibilidad de utilizar una plataforma bajo licencia no se descarta en caso de que el acceso sea sencillo.

Característica	**Plataforma**					
	Hyperledger	IOTA	Ethereum	Filament	IoTeX	R3 Corda
Integración	**Sí**	**Sí**	**Sí**	**Sí**	**Sí**	**Sí**
Madurez	**Sí**	**Sí**	**Sí**	No	No	**Sí**
Sencillez	No	**Sí**	**Sí**	**Sí**	No	**Sí**
Automatización	**Sí**	**Sí**	**Sí**	No	**Sí**	No
Red Privada	**Sí**	**Sí**	**Sí**	**Sí**	**Sí**	**Sí**
Documentación	**Sí**	**Sí**	**Sí**	No	No	**Sí**
Comunidad	No	**Sí**	**Sí**	No	No	No
Disponibilidad	No	**Sí**	**Sí**	No	**Sí**	No

Tabla 2.1: Comparativa de plataformas

Es importante destacar que en esta tabla sólo se han definido como "**Sí**" las características totalmente cumplimentadas. Por ejemplo, Filament y R3 Corda sí que cuentan con documentación, pero su amplitud no es la deseada. En cuanto a la disponibilidad, Hyperledger, Filament y R3 Corda son gratuitas, pero requieren licencia. Por último, aunque todas las plataformas tienen cierta comunidad, sólo en el caso de IOTA y Ethereum son de la envergadura suficienta como para asegurar una cantidad de contenido online (investigación, guías, foros, etc) que permita solucionar potenciales problemas con sencillez.

Así, la selección, aunque disputada por IOTA, es Ethereum. Esto se debe a que no solo cumple todas las características deseadas si no a que lo hace con creces. A continuación se profundiza en todos los aspectos de la plataforma, desde su (corta) historia hasta las posibilidades que ofrece más allá de las vistas en la tabla.

2.4.1. Ethereum

Una de las plataformas más extendidas (por no decir la que más) para el uso de blockchain es Ethereum, propuesta por Vitalik Buterin en 2015. Como se ha mencionado antes brevemente, su fama y extensión se debe a las posibilidad que ofrece mediante los contratos inteligentes y las aplicaciones descentralizadas.

Los contratos inteligentes son programas informáticos autoejecutables. Estos contratos se crean utilizando el lenguaje de programación Solidity, específico de Ethereum, y se despliegan en la cadena de bloques. Una vez desplegados se ejecutan exactamente como se programaron, sin posibilidad de censura, tiempo de inactividad, fraude o interferencia de terceros. Cuando se despliega un contrato inteligente en Ethereum, se le asigna una dirección única en la red. Los participantes de la red pueden interactuar con el contrato enviando transacciones a esa dirección. Además, estas transacciones pueden contener instrucciones para que el contrato realice ciertas operaciones o cambios de estado.

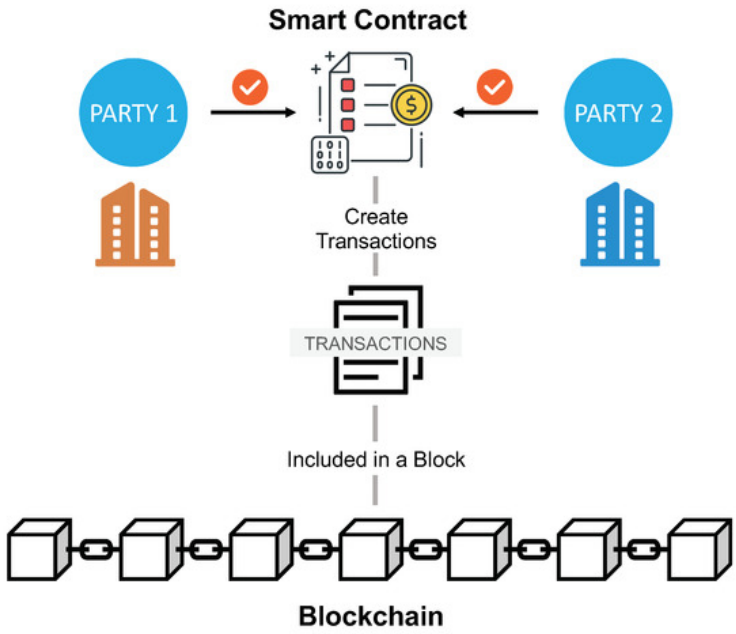

Figura 2.8: Funcionamiento de un contrato inteligente [49]

Además de su funcionalidad básica de transferencia de valor (criptodivisa), Ethereum también proporciona una capa de almacenamiento descentralizado llamada "Ethereum Name Service" (ENS), que permite asignar nombres legibles por humanos a direcciones de Ethereum, facilitando así la interacción con los contratos y aplicaciones distribuidas.

Aunque originalmente Ethereum utilizaba PoW como mecanismo de consenso, en 2023 (a lo largo de este trabajo de investigación), migró a Proof of Stake con la finalidad de reducir el coste energético y facilitar la escalabilidad.

En definitiva, la red descentralizada de nodos de Ethereum, combinada con la capacidad de ejecutar contratos inteligentes, brinda una amplia gama de posibilidades para la creación de soluciones innovadoras en diversas áreas, como las finanzas, la industria, o incluso los juegos.
Algunas de las características más remarcables de Ethereum son las siguientes:

- **Amplia adopción y comunidad activa**
  Como ya se ha comentado, es una de las plataformas blockchain más ampliamente adoptadas y cuenta con una comunidad de desarrolladores y usuarios activa y comprometida. Esta amplia adopción y el apoyo de la comunidad ofrecen una amplia gama de recursos, herramientas y bibliotecas que facilitan la implementación y el desarrollo.

- **Interoperabilidad**
  Ethereum ha sido diseñado para ser compatible con otros protocolos y tecnologías, lo que facilita la interoperabilidad con otros sistemas y la integración con componentes

existentes de IoT. Además sigue estándares *open source* y bien establecidos, lo que garantiza la compatibilidad con otros sistemas.

- **Seguridad**
  A pesar de haber sufrico ciertos incidentes de seguridad en el pasado, Ethereum ha ido mejorado constantemente, mediante la implementación de medidas de seguridad o la auditoría de contratos inteligentes entre otras medidas.

- **Herramientas de desarrollo**
  Aunque se entrará en profundidad en ciertas herramientas más adelante, es pertinente mencionar que Ethereum cuenta con multtud de herramientas que facilitan la creación y el despliegue de aplicaciones blockchain. Por ejemplo su entorno de desarrollo integrado (IDE) Remix, que permite trabajar con Solidity, el lenguaje en el que se escriben los contratos inteligentes de Ethereum.

Estas cualidades ofrecen un entorno propicio no solo para la realización del proyecto si no también para el trabajo futuro de mejora y expansión e incluso la posible implementación en el mundo real. Así, brinda confianza, escalabilidad y flexibilidad. A continuación analizaremos las herramientas que nos han permitido trabajar con Ethereum.

Capítulo 3

Desarrollo

3.1. Plan

Ee primer paso a realizar antes de comenzar el trabajo es el de definir una hoja de ruta y un plan de acción. De esta manera se reparten los recursos (de tiempo, material, etc.) a lo largo del proyecto y se crea una guía a seguir para realizar el proyecto. A lo largo de esta sección se detalla este proceso.

3.1.1. Marco de trabajo

Para este proyecto se utilizará la metodología de cascada, que es un enfoque secuencial para la gestión del trabajo. Esta metodología divide el proyecto en fases y, salvo excepciones, cada fase comienza al terminar la previa. Este enfoque es lineal y rígido, y hace difícil realizar cambios en etapas previas, especialmente cuanto mayor sea la envergadura del proyecto.

Se ha seleccionado esta metodología porque en este proyecto cada uno de los hitos a conseguir depende del anterior. A grandes rasgos, el proyecto cuenta con una primera fase de estudio, sin la cual no se puede realizar el despliegue del sistema, la segunda fase. Además, para analizar el sistema, éste debe encontrarse en correcto funcionamiento. Aunque estas tres fases pueden ser desglosadas en más partes, esas partes también atienden a la necesidad de que las previas hayan sido completadas, y por ello este enfoque es el elegido.

Así, se realiza una planificación inicial del proyecto, cuyo diagrama de Gantt se puede observar en la figura 3.1. La duración cubre siete semanas a jornada completa, es decir, 280 horas. Las 20 horas restantes son asignadas a la elaboración del informe a lo largo de las siete semanas. Además, se presupone la necesidad de horas extra tras la última semana debido a posibles retrasos a lo largo del desarrollo del proyecto. En el anexo C se puede observar el siguiente diagrama en mayor detalle.

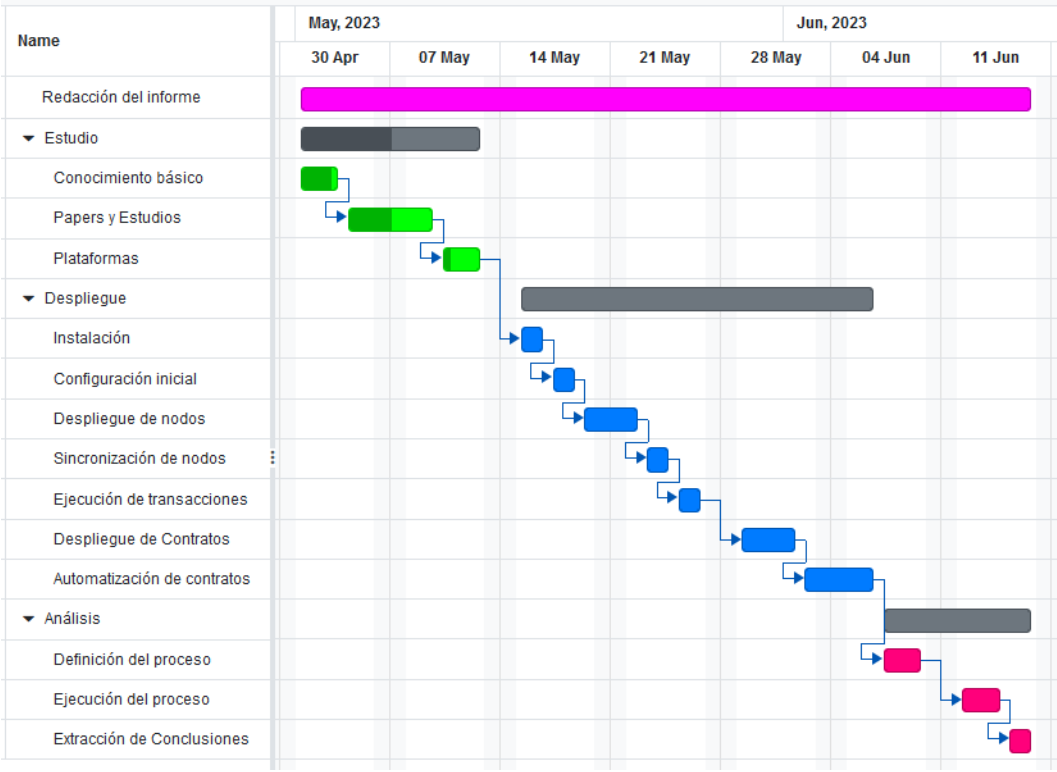

Figura 3.1: Diagrama de Gantt con la planificación inicial

La primera fase, el estudio, dispone de dos semanas. Esto se debe principalmente a la inexperiencia del desarrollador en el manejo de las tecnologías en cuestión. Sin embargo también se considera que un tiempo de estudio amplio puede prevenir problemas y retrasos futuros, ya que se puede realizar un análisis más profundo del software a utilizar, permitiendo así seleccionar el más adecuado. También es la fase donde la presente planificación está sujeta a cambios en función de la visión del desarrollador, cambios que pueden necesitar incluso varias horas para ser realizados correctamente.

La segunda fase se refiere a la implantación de la plataforma, su configuración, uso y pruebas. Es con diferencia a la que más tiempo se le asigna debido a su cantidad de tareas y dificultad, más de tres semanas. La dificultad emana de dos situaciones: inexperiencia y falta de recursos. En el caso de la primera, aunque durante la fase de estudio el desarrollador puede haber llegado a conocer el funcionamiento del software que se utilizará, dos semanas no son tiempo suficiente para adquirir conocimientos profundos sobre como operarlo. La segunda situación es causada por el constante cambio que experimentan las tecnologías a abordar unido a su corta edad, especialmente en el caso de la *blockchain*. Debido a ello, la cantidad de material disponible en la red, desde estudios y *papers* hasta guías y tutoriales, es reducido y, cuando existe, tiende a estar desactualizado.

La tercera fase Se refiere al análisis. Esto incluye definir cuál será el proceso de análisis,

cómo se realizará, ejecutarlo y extraer las conclusiones pertinentes. Dado que el desarrollador ya debe haber alcanzado experiencia suficiente con el manejo de las herramientas y el software, a esta fase se le asigna una semana y media en total. Podría ser un tiempo menor pero se tiene en consideración la necesidad de un cierto estudio sobre los métodos de análisis (es decir, que pruebas realizar sobre el sistema) y la posible necesidad de crear *scripts* y similares en la ejecución del análisis.

A lo largo de estas tres fases se realizará paulatinamente la redacción del informe, conforme se avance con el resto del proyecto. Inicialmente se pretende dedicar a esta tarea entre dos y tres horas por semana adicionales, ajustando conforme a necesidad. De esta manera se cubren las veinte horas restantes hasta las 300 requeridas para el trabajo de fin de grado, como se ha comentado.

3.1.2. Riesgos

A lo largo del proyecto pueden surgir problemas que afecten a su desarrollo. Estos problemas, riesgos, pueden afectar en mayor o menor medida y cada uno de ellos tiene una probabilidad de ocurrir diferente. Aunque algunos nos impredecibles, otros pueden ser conocidos previamente. Esta sección se centra en éstos últimos, en cómo actuar contra ellos y paliar sus efectos mediante un enfoque proactivo.

El primer paso es separar los riesgos en sus diferentes categorías. Aunque hay muchas categorizaciones distintas, se utilizará la más básica y aceptada. Así, tenemos tres tipos de riesgos:

- **Del Negocio**
  Aquellos que hacen peligrar la ejecución del proyecto. Dado si tipo generalmente económico o administrativo no se tendrán en cuenta para este proyecto.

- **Del Proyecto**
  Aquellos que afectan a la planificación del proyecto. Al hacerse realidad, requerirán un mayor coste y esfuerzo.

- **Técnicos**
  Aquellos que afectan a la calidad. Al hacerse realidad, la complejidad del proyecto crece por encima de lo inicialmente estimado.

Para afrontar los riesgos existen dos estrategias. Como se ha comentado anteriormente, en este proyecto se utilizará una estrategia proactiva, es decir, se analizarán los posibles riesgos antes de comenzar, se mide la probabilidad de la ocurrencia de cada uno, se priorizan y se lleva a cabo una planificación tanto de prevención como de contingencia. Como mención, la otra posibilidad es una estrategia reactiva, que requiere esperar a que los riesgos se conviertan en realidad y después afrontarlos. Esta estrategia requiere tener recursos continuamente disponibles en el caso de que un riesgo ocurra.

Una vez elegida la estrategia proactiva, se debe utilizar una matriz de riesgos para, mediante su probabilidad e impacto, determinar la importancia del mismo. Dicha matriz de riesgos se puede ver a continuación:

Impacto	Probabilidad				
	Improbable	**Poco Probable**	**Posible**	**Probable**	**Casi Seguro**
Mínimo	Muy Bajo	Muy Bajo	Bajo	Medio	Medio
Menor	Muy Bajo	Bajo	Medio	Medio	Medio
Moderado	Bajo	Medio	Medio	Alto	Alto
Mayor	Medio	Medio	Alto	Alto	Muy Alto
Maximo	Medio	Alto	Alto	Muy Alto	Muy Alto

Tabla 3.1: Matriz de riesgos

Para terminar, antes de avanzar a la gestión de los riesgos, se deben tener en cuenta las técnicas de respuesta para minimizar el impacto. Estas técnicas pueden ser divididas, de acuerdo a su propósito, en las siguientes categorías:

- **Evasión**
  Aquellas técnicas que se centran en evitar que el riesgo se haga realidad o afecte al proyecto.
- **Atenuación**
  Acciones centradas en la reducción de la probabilidad de que ocurra un riesgo o en la mitigación de su impacto.
- **Transferencia**
  Técnicas que depositan sobre otras entidades la responsabilidad de gestionar el riesgo, como aseguradoras o proveedores.
- **Aceptación**
  Se divide en activa y pasiva. Esta última es la acción de reconocer el riesgo pero no hacer nada por gestionarlo. En el caso de la activa, se preven contingencias y se reservan recursos, de la similar manera a la estrategia reactiva previamente mencionada.

Gestión de Riesgos

A lo largo de esta sección se definen los riesgos detectados, en las tablas siguientes, junto a sus tipos, probabilidades e impactos. También se define un plan de acción para cada uno de ellos.

Riesgo 01: Problema médico		
Tipo De proyecto	**Probabilidad** Poco probable	**Impacto** Moderado
Descripción Retrasos en el desarrollo y ejecución del proyecto debido a una baja médica del desarrollador.		
Plan de acción Replanificación de la siguiente actividad para paliar los retrasos y problemas resultantes.		

Tabla 3.2: Riesgo 01

Riesgo 02: Retrasos de Planificación		
Tipo	**Probabilidad**	**Impacto**
De proyecto	Probable	Mayor
Descripción		
Retrasos en el desarrollo y ejecución del proyecto debido a estimaciones incorrectas o problemas inesperados.		
Plan de acción		
Replanificación de la siguiente actividad para paliar los retrasos y problemas resultantes mediante la adición de horas a las actividades no terminadas.		

Tabla 3.3: Riesgo 02

Riesgo 03: Modificación de objetivos		
Tipo	**Probabilidad**	**Impacto**
De proyecto	Poco Probable	Mayor
Descripción		
Cambio en los requisitos del proyecto debido a la planificación ágil u otros factores..		
Plan de acción		
Replanificación de los requisitos y de las siguientes actividades para dedicar tiempo y recursos a conformarse a ellos.		

Tabla 3.4: Riesgo 03

Riesgo 04: Problemas con tecnologías nuevas		
Tipo	**Probabilidad**	**Impacto**
Técnico	Probable	Mayor
Descripción		
Retrasos y problemas en el desarrollo y ejecución del proyecto debido a la inexperiencia del desarrollador con las tecnologías a utilizar.		
Plan de acción		
Uso de software conocido si es posible. En caso negativo, uso de software con funcionamiento sencillo de entender, amplia documentación y recursos en línea y comunidad activa.		

Tabla 3.5: Riesgo 04

Riesgo 05: Actualizaciones inesperadas del software a utilizar		
Tipo	**Probabilidad**	**Impacto**
Técnico	Poco Probable	Moderado
Descripción		
Retrasos y problemas en el desarrollo y ejecución del proyecto debido a actualizaciones inesperadas del software a utilizar.		
Plan de acción		
Evaluación de la viabilidad del sistema desarrollado dentro de la nueva versión. En caso de no ser viable, evaluación del coste de los cambios. En caso de ser costoso, reversión del software a la última versión viable.		

Tabla 3.6: Riesgo 05

Riesgo 06: Problemas de compatibilidad entre recursos software		
Tipo	**Probabilidad**	**Impacto**
Técnico	Probable	Mayor
Descripción		
Retrasos y problemas en el desarrollo y ejecución del proyecto debido a problemas de compatibilidad entre recursos software sin capacidad de interoperabilidad.		
Plan de acción		
Durante la fase de análisis y estudio de las tecnologías y plataformas, escoger para el desarrollo plataformas que ofrezcan interoperabilidad. En la medida de lo posible, elegir software "plan B".		

Tabla 3.7: Riesgo 06

Riesgo 07: Bajo rendimiento del hardware		
Tipo	**Probabilidad**	**Impacto**
Técnico	Poco probable	Moderado
Descripción		
Retrasos y problemas en el desarrollo y ejecución del proyecto debido a la incapacidad del hardware de mantener el sistema con las especificaciones de potencia y rendimiento deseadas.		
Plan de acción		
Durante la fase de análisis y estudio de las tecnologías y plataformas, escoger para el desarrollo plataformas que ofrezcan buen rendimiento para los recursos disponibles. En caso de no ser posible, adaptar las configuraciones del software y el sistema a las posibilidades del hardware manualmente.		

Tabla 3.8: Riesgo 07

3.1.3. Presupuesto

Con el objetivo de dar una visión realista al proyecto, se estima el coste que el desarrollo del proyecto podría acarrear a una empresa en el caso de llevarlo a cabo. Después se muestra el coste real de la ejecución del proyecto como trabajo de fin de grado mediante el apoyo de la universidad.

Simulado

Para estimar el presupuesto realizamos una división de gastos entre dos categorías, material y personal. El material se refiere a los recursos utilizados y detallados en la sección de recursos, más adelante. El coste personal es el coste del salario del desarrollador que ha llevado a cabo el proyecto.

- **Material**
  Para el desarrollo del proyecto se han utilizado los siguientes recursos, mostrados junto a sus costes:

Producto	Cantidad	Coste	Total
Ordenadores	3	50	150
Pantalla	1	30	30
Teclado + ratón	1	15	15
Switch de red	1	20	20
Cables VGA	3	5	15
Cables de red	3	4	12

Tabla 3.9: Coste del material

  Dado que el software será en su totalidad gratuito y el material de oficina obviado, el precio de los recursos materiales asciende a un total de aproximadamente **242 Euros**.

- **Personal**
  Para el cálculo del gasto de personal se ha utilizado e sueldo medio medio de un desarrollador de software en España: 29.800 Euros brutos al año. Reduciendo esta cifra a sueldo por hora (8 horas diarias, jornada completa) se obtienen 10.2 Euros/hora. Dado que la previsión estima algo más de 300 horas, utilizamos esa aproximación: $300x10{,}2 = 3060$. Es decir, el coste total de personal ascendería a alrededor de **3060 Euros** para la empresa.

- **Total**
  Para calcular el total, realizamos la suma de ambos costes: $242 + 3060 = 3302$. El total del presupuesto simulado para la empresa ascendería a aproximadamente unos **3302** Euros.

Real

Dado que este proyecto no se lleva a cabo en el marco empresarial, el coste real del mismo difiere de la simulación previamente planteada. En primer lugar, el desarrollador

(alumno de la universidad) no percibe ningún tipo de remuneración monetaria, por lo que el coste de personal asciende a 0 Euros. Además, el material utilizado para llevar a cabo el proyecto es también proporcionado por la universidad, por lo que su coste es, una vez más, de 0 Euros. Es decir, el coste final del proyecto es de **0 Euros**.

Se puede argumentar que esto no es del todo cierto, ya que hay cierto coste asociado al gasto energético de los ordenadores y al desgaste del material, sin embargo es tan reducido que se ha obviado a la hora de realizar los cálculos.

3.2. Análisis

A lo largo de esta sección se proporciona un glosario como método de simplificar la navegación por los temas y campos tratados en adelante y después se definen y categorizan los requisitos del sistema.

3.2.1. Glosario

Término	**Descripción**
Blockchain	Estructura de datos basada en una cadena de bloques conectados en serie que almacenan información. Llamada cadena de bloques en español.
Clef	Herramienta de Geth que permite gestionar operaciones relativas a la cuentas de una red blockchain con seguridad.
Consenso (Algoritmo)	Método que permite a una red blockchain operar de manera descentralizada, sin necesidad de la existencia de entidades de confianza en dicha red.
Contrato Inteligente	Programa almacenado en una red blockchain de Ethereum, y por tanto inmutable. Para modificar sus variables es necesario realizar una transacción, para ver la información no. Llamado *smart contract* en inglés.
Criptodivisa	Token de una blockchain.
Cuenta	Entidad que permite almacenar tokens de la red blockchain a la que pertenece. Permite realizar transacciones y se accede a ella mediante una dirección.
Despliegue	Acción de poner en funcionamiento el sistema.
Dirección	Número en base hexadecimal que representa una entidad de la blockchain y permite acceder a ella. Ejemplos de entidades son las cuentas y los Contratos desplegados.
Ether	Criptodivisa de Ethereum. Equivale a 10^6 Wei.
Ethereum	Plataforma de código abierto basada en la tecnología blockchain.
Firma	La acción de validar una transacción para su envío a la blockchain.
Gas	Cantidad de tokens necesaria para ejecutar una transacción.

Tabla 3.10: Glosario I

Término	Descripción
Geth	Implementación en el lenguaje Go de la plataforma Ethereum.
Iot	Red colectiva de dispositivos conectados mediante internet u otros medios.
JavaScript	Lenguaje de programación mediante el que se interactúa con la herramienta Geth.
Nodo	Participante de una red blockchain.
Operación	En el contexto de este trabajo, transacción.
PoS	Algoritmo de consenso basado en la posesión de tokens de los nodos participantes en una red blockchain.
PoW	Algoritmo de consenso basado en la resolución de problemas matemáticos complejos por parte de los nodos de una red blockchain.
Python	Lenguaje de programación mediante el que se realizan las operaciones sobre contratos en este proyecto.
Script	Programa informático, secuencia de comandos.
Solidity	Lenguaje de programación mediante el que se escriben los contratos Ethereum.
Token	Unidad de valor basada en criptografía. Es emitida en una blockchain y sirve como moneda de cambio mediante la que realizar operaciones sobre la misma.
Transacción	Operación sobre al blockchain que actualiza su estado. Puede ser un cambio en una variable de un contrato o un envío de tokens de la red blockchain.
Wei	Criptodivisa de la red Ethereum. Es la unidad mínima.

Tabla 3.11: Glosario II

3.2.2. Requisitos

Los requisitos de sistema en un proyecto como este son las especificaciones que describen las características y funcionalidades que el sistema debe tener para satisfacer las demandas o necesidades del cliente o usuario. En el caso actual, al no haber cliente ni usuario, se refieren a las características necesaria para que el sistema se pueda considerar funcional y puedan realizarse sobre él las tareas de análisis.

Estos requisitos se dividen los siguientes tres grupos:

- **Funcionales**
  De aquí en adelante "FR" por sus siglas en inglés, estos requisitos son aquellos necesarios para la ejecución normal del sistema.

- **No Funcionales**
  De aquí en adelante "NFR" por sus siglas en inglés, estos requisitos son aquellos no necesarios para la ejecución normal del sistema.

- **De Información**
  De aquí en adelante "IR" por sus siglas en inglés, estos requisitos son aquellos referidos a la información que el sistema debe almacenar.

Requisitos Funcionales

- **RF-01**:
  El sistema debe permitir la creación de una red Ethereum privada.
- **RF-02**:
  El sistema debe permitir el registro de nodos en la red privada, con el propósito de que formen parte de ella e interactúen con los demás componentes.
- **RF-03**:
  El sistema debe permitir que un nodo que forme parte de la red valide transacciones manteniendo el consenso.
- **RF-04**:
  El sistema debe permitir que un nodo que forme parte de la red ejecute transacciones e interactúe con la cadena de bloques.
- **RF-05**:
  El sistema debe permitir la creación y gestión de cuentas de la red Ethereum.
- **RF-06**:
  El sistema debe permitir la gestión de las claves criptográficas de cada cuenta.
- **RF-07**:
  El sistema debe permitir realizar transacciones entre cuentas de la red blockchain de Ethereum, utilizando claves criptográficas para la autenticación y la seguridad de las transacciones.
- **RF-08**:
  El sistema debe permitir desplegar contratos inteligentes en la red Ethereum privada.
- **RF-09**:
  El sistema debe permitir la interacción con los contratos desplegados en la red.
- **RF-10**:
  El sistema debe permitir gestionar diferentes dispositivos IoT de maneras distintas mediante los contratos.
- **RF-12**:
  El sistema debe permitir a los dispositivos IoT almacenar información en la red Ethereum privada.
- **RF-13**:
  El sistema debe permitir acceder a la información almacenada en la red Ethereum privada.

Requisitos No Funcionales

- **RNF-01**:
  EL sistema debe ofrecer un rendimiento eficiente y tiempos de respuesta rápidos.

- **RNF-02**:
  EL sistema debe ofrecer eficiencia en cuanto a consumo de recursos energéticos y de almacenamiento.

- **RNF-03**:
  EL sistema debe ser seguro ante ataques y manipulaciones externas.

- **RNF-04**:
  EL sistema debe tener una arquitectura modular para facilitar el desarrollo futuro.

- **RNF-05**:
  EL sistema debe ser escalable para poder manejar grandes números de transacciones y nodos.

- **RNF-06**:
  EL sistema debe disponer de una interfaz de usuario sencilla.

- **RNF-07**:
  EL sistema debe ser compatible con diferentes plataformas y sistemas operativos.

- **RNF-08**:
  EL sistema debe ser compatible con los protocolos de comunicación utilizados en IoT.

- **RNF-09**:
  EL sistema debe contar con un sistema de gestión de cuentas seguro y confiable.

Requisitos De Información

- **RI-01**:
  Cada nodo de la red debe tener un identificador único.

- **RI-02**:
  Cada cuenta de la red debe tener un identificador o dirección único.

- **RI-03**:
  Cada contrato desplegado en la red debe tener un identificador o dirección único.

- **RI-04**:
  Los eventos relevantes como transacciones o despliegues en la red deben ser registrados.

- **RI-05**:
  El almacenamiento de datos en la red mediante contratos debe especificar qué nodo es el responsable de ese almacenaje.

- **RI-06**:
  Cada nodo debe tener un tipo de datos almacenados acorde a sus necesidades.

3.3. Diseño

A lo largo de esta sección se muestra mediante diagramas el diseño del sistema. Éste puede dividirse en dos partes, el hardware y el software, para los que se utilizarán principalmente *SysML* y *UML* como lenguajes de modelado visual. Estas dos partes se pueden ver en las subsecciones siguientes, seguidas por los casos de uso.

3.3.1. Hardware

La parte hardware sobre la que se sustenta el sistema no es especialmente compleja, y, como se puede ver a continuación en la figura 3.2, debe contener nodos que puedan operar de tres maneras:

- Nodo de la red sin conexión a dispositivos IoT.
- Nodo de la red con conexión a 0 o varios dispositivos IoT.
- Nodo de la red desplegado en un dispositivo IoT.

Además, cada uno de los tipos de nodos puede o no participar como minero de transacciones.

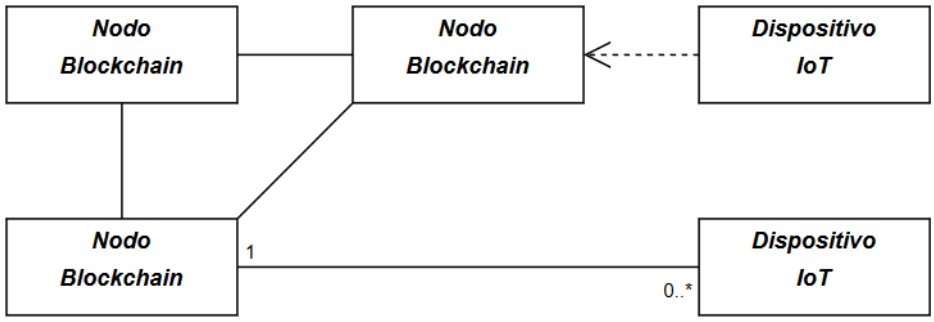

Figura 3.2: Hardware del sistema

En el caso de este proyecto, y como se verá a continuación en la subsección Software, los nodos podrán operar de las tres formas e iterar entre ellas con facilidad. Esto de debe a que los dispositivos IoT serán simulados mediante *scripts* Python, por lo que para cambiar de tipo de nodo sólo será necesario ejecutar o no diferente número de simulaciones.

3.3.2. Software

Para establecer en cada dispositivo un nodo con la versatilidad descrita en el apartado anterior, se definirá un sistema de directorios como el siguiente:

```
eth/
├── nodo/
│   ├── keystore
│   └── blockchain
└── contratos/
    └── contrato01/
        ├── contrato.sol
        └── despliegue.script
```

Figura 3.3: Sistema de directorios

Es importante notar que este es un diagrama sencillo, diseñado para separar la ejecución de dispositivos IoT (simulados bajo la carpeta *contratos*) y el nodo. Está, sin embargo, sujeto a potenciales cambios, especialmente en función de las necesidades del *software* necesario para ejecutar ambas partes, software que se detalla en el apartado 3.4, Recursos.

A continuación podemos ver las interrelaciones de los diferentes componentes de manera sencilla, en especial cómo el cliente elegido para trabajar con la plataforma Ethereum (gestor nodo en el diagrama) obtiene la información necesaria por parte de los dispositivos IoT y la pone en común con el resto de nodos.

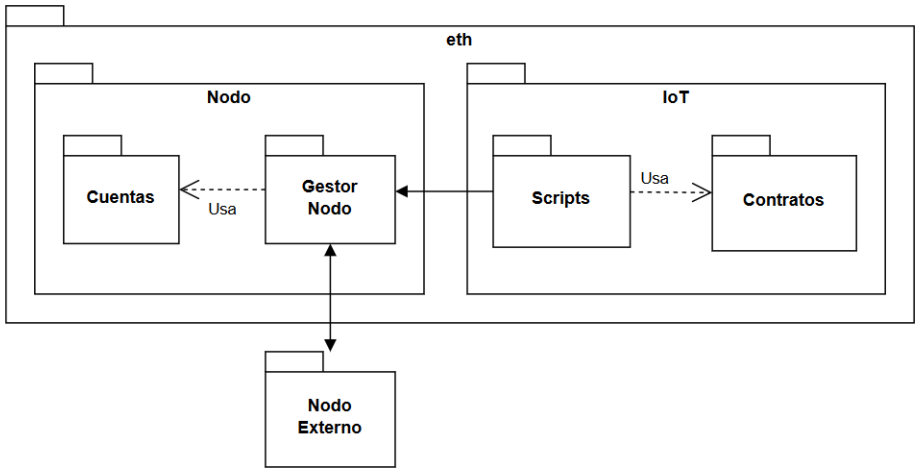

Figura 3.4: Diagrama de paquetes

3.3.3. Casos de Uso

Como se puede ver en el siguiente diagrama UML, se definen siete casos de uso principales.

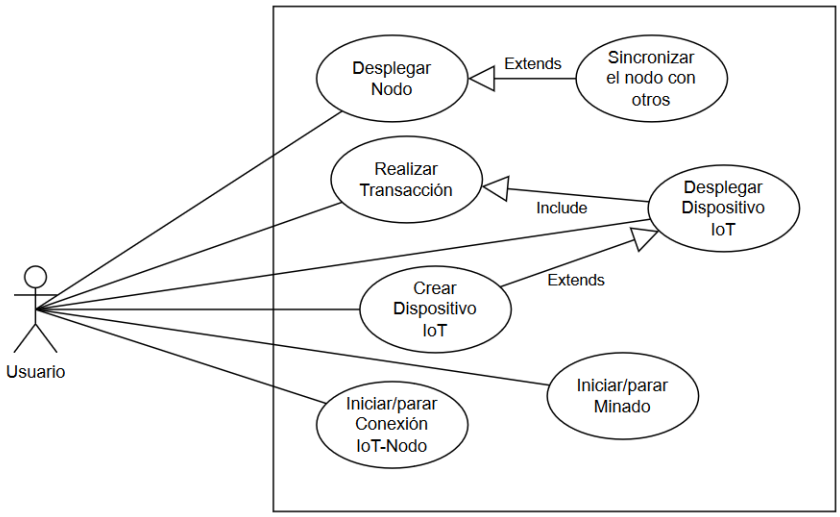

Figura 3.5: Diagrama de casos de uso

El proceso de implementación del sistema será detallado más adelante en la sección 3.5 Implementación, e incluirá todos los casos de uso.

3.4. Recursos

En esta sección se presentan los recursos utilizados, desde el hardware sobre el que se sustenta el sistema hasta el software que lo compone. En este último caso se incluye el sistema operativo sobre el que se ha trabajado hasta el resto de software y herramientas utilizadas para llevar a cabo el proyecto. También se hablará en ciertos casos sobre la selección de determinados recursos sobre otros.

3.4.1. Hardware

Como material sobre el que se sustentará el software se han utilizado unos ordenadores llamados *nooks*. Estos son similares a las tradicionales *torres* excepto por su tamaño, que es mas reducido. En este caso se han utilizado tres unidades.

Además de estos *nooks*, también se incluyen en este apartado las fuentes de alimentación, los periféricos (pantalla, teclado y ratón), un *switch* que permite la conexión cableada a Internet y el diverso cableado que conecta estos componentes.

3.4.2. Software

Sistema Operativo

Como sistema operativo sobre el que mantener la futura blockchain y las simulaciones de dispositivos IoT se ha elegido Ubuntu, más concretamente la última versión disponible a la hora de realizar la instalación, la versión 22.04.2. Se elige la versión de escritorio para

facilitar tareas de desarrollo posteriores mediante herramientas como el editor de texto o el navegador de internet.

Aunque en primera instancia se considera utilizar Windows, las cualidades que ofrece Ubuntu son mejores para llevar a cabo el proyecto:

- **Flexibilidad**
  Ubuntu ofrece alta personalización y adaptabilidad a las necesidades específicas de cada usuario, característica especialmente importante en el caso del desarrollo de software.

- **Compatibilidad**
  Ubuntu es ampliamente reconocido por su soporte de desarrollo, amplia comunidad y una gran cantidad de herramientas y bibliotecas disponibles para los usuarios. Además Linux también demuestra compatibilidad con una amplia gama de tecnologías y protocolos utilizados en los entornos IoT y blockchain, lo que facilitará la posterior integración.

- **Escalabilidad**
  La capacidad de adaptación a cada entorno específico también resulta indispensable en un campo como el de IoT, donde las capacidades de los dispositivos varían y precisan de un sistema que ofrezca versatilidad y robustez.

- **Documentación y comunidad**
  A la hora de trabajar con nuevas tecnologías en las que el desarrollador no tiene demasiada experiencia, es preciso asegurarse de que va a existir material que permita aprender e incluso apoyo de otros desarrolladores con mayor experiencia. En el caso de Ubuntu, así como los sistemas basados en Linux, existe gran cantidad de documentación en internet, así como una gran comunidad de desarrolladores dispuestos a brindar apoyo en multitud de foros y sitios web.

Estas características, sumadas a la gratuidad del sistema, sitúan a Ubuntu como la primera elección de sistema operativo.

BalenaEtcher

Software libre utilizado para grabar el archivo de sistema operativo en una memoria portátil (pen drive) y permitir su instalación en los ordenadores. Se ha seleccionado esta opción por su sencillez.

Blockchain

De acuerdo a las razones detalladas en el capítulo 2.4, Plataformas, la elegida ha sido Ethereum. Dentro de ésta se destacan a continuación Geth, el cliente a utilizar, y Solidity, el lenguaje mediante el que se programan los contratos, que se detallan a continuación.

Geth

Geth, también conocido como Go-Ethereum, es la implementación en el lenguaje de programación Go del protocolo Ethereum. Es un cliente de ejecución de Ethereum, lo que significa que permite realizar transacciones, despliegue y ejecución de contratos, además del despliegue de una red propia. También contiene una máquina incorporada, la Máquina Virtual Ethereum.

Es uno de los clientes más utilizados además de estar respaldado por una comunidad muy activa. Es compatible con los estándares de Ethereum y está actualizado a su par. También tiene un gran rendimiento y una alta eficiencia en el procesamiento de transacciones y sincronización con la red Ethereum. Además, ofrece una amplia gama de posibilidades de configuracion y permite llevar a cabo casi cualquier proyecto con relativa sencillez, lo que lo hace especialmente atractivo.

La versión de Geth elegida es la 1.11.6, la última previa a la migración de Ethereum al protocolo de consenso PoS, con el propósito de que permita trabajar con el método PoW y activar y desactivar el minado de bloques en función de las necesidades a lo largo del proceso de desarrollo. Todo esto además se lleva a cabo desde la consola JavaScript que permite interactuar con el nodo deseado y la red a la que éste esté conectado.

Es también pertinente mencionar que Geth cuenta con herramientas adicionales como Clef, un gestor de cuentas y transacciones en la red blockchain que se utilizará más adelante.

Solidity

Como se ha mencionado con anterioridad, Solidity es el lenguaje de programación utilizado para crear los contratos inteligentes. Utilizar este lenguaje es necesario, ya que son la manera de almacenar en la cadena de bloques los datos generados por los dispositivos IoT. La versión del lenguaje que se utilizará será la 0.8.0, dato que debe ser definido en cada contrato para que pueda ser compilado y se pueda interactuar con él una vez esté desplegado.

Ethereum ofrece un entorno de desarrollo para trabajar con Solidity, llamado Remix IDE. Sin embargo, dado que el desarrollador conoce el lenguaje y los errores pueden ser detectados son sencillez en el proceso de compilación, no se utilizará dicho IDE. La programación se realizará mediante el editor de texto nativo de Ubuntu y la compilación mediante línea de comandos o mediante un *script* Python, como más adelante se explica.

Python

Para la compilación, despliegue e interacción con los contratos se utilizará Python. De esta manera se podrá enlazar con dichos procesos la simulación de un dispositivo IoT. En un mismo *script* se puede compilar un contrato, desplegarlo en la red blockchain, obtener su dirección y utilizarla para realizar transferencias. Esto, sin embargo, precisa de dos paquetes que deberán ser instalados mediante el gestor de paquetes pip: py-solc-x y web3.

- **py-solc-x**
  Solc es el nombre del compilador para Solidity, y py-solc-x la versión adpatada a Python. Para realizar la compilación mediante Python, es preciso utilizar este paquete, que no solo permite realizar el proceso si no que además lo hace de manera sencilla.

- **web3.py**
  web3.py es una librería de Python que permite interactuar con Ethereum. La API original deriva de la API JavaScript Web3.js, pero contiene cambios que hacen su uso más cómodo para su desarrollo en Python.

3.4.3. Otros

Además del hardware y software mencionados, se han puesto a disposición del desarrollador otros recursos:

- **Laboratorio**
  Un entorno equipado con escritorio, material de oficina y otros materiales para hacer el trabajo más sencillo.

- **Pizarra**
  Ha servido como tablero de planificación entre otros y ha permitido llevar un seguimiento del proyecto y su calendario, así como de su planificación. Se ha hecho así de manera física en contraposición a la posibilidad de utilizar software preparado para ello.

3.5. Implementación

A lo largo de esta sección se detallan los pormenores de la implementación del sistema. Se describirán los procesos correspondientes a cada subapartado, de manera que se comprende el proceso necesario para tener un nodo de la red blockchain funcional, sincronizado y con un contrato desplegado. El despliegue de más nodos y/o contratos es sencillo, sólo requiere ejecutar los pasos repetidamente.

3.5.1. Instalación

El primer paso de la instalación se refiere al sistema operativo. Se debe descargar la versión 22.04.2 de escritorio de Ubuntu, disponible en el apartado de descargas de su página web [25].
La instalación es trivial, aunque precisa de un computador funcional y software para realizar la grabación del sistema operativo en un *pen drive*. En la misma página de Ubuntu se proporciona una guía [25]. Una vez realizada la instalación, es recomendable acceder a los ajustes del sistema y desactivar las actualizaciones automáticas, ya que parte del software utilizado no se encuentra en su última versión.

Una vez listo el sistema operativo, se debe instalar Ethereum, Geth, pip, py-solc-x y web3, como se ha podido ver en el capítulo 5, Recursos. Para ello se debe abrir un terminal de comandos y ejecutar los siguientes:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository -y ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install -y ethereum
$ sudo apt install -y python3-pip
$ pip install py-solc-x
$ pip install web3
```

El caso de la instalación de la versión 1.11.6 de Geth, primero debe descargarse directamente desde su sitio web [15]. Una vez descargado, se extraerá y se añadirá al PATH. En este caso, se asume que el usuario se encuentra en el directorio donde se ha descargado el archivo y que Geth y sus herramientas serán guardadas en la carpeta "geth" dentro del directorio /home/user/.

```
$ tar -xf geth_1.11.6.tar.gz
$ mv geth_1.11.6/* /home/user/geth/
$ nano /home/user/.bashrc
```

Tras acceder mediante la última orden al archivo *.bashrc*, se añade al final del achivo la siguiente línea.

```
export PATH="/home/nook/eth/geth:$PATH"
```

Para aplicar los cambios sin reiniciar la máquina, se ejecuta la última orden.

```
$ source ~/.bashrc
```

3.5.2. Creación del Nodo

El primer paso una vez instalado el software necesario es crear el directorio donde se trabajara y, dentro de él, los directorios donde se ubicarán el nodo y los contratos.

```
$ mkdir /home/user/eth
$ mkdir /home/user/eth/node
$ mkdir /home/user/eth/contracts
$ cd /home/user/eth
```

En el directorio de trabajo creamos el fichero génesis, cuyos contenidos deben ser los especificados en el anexo B, Archivos, y lo utilizamos para, mediante Geth, inicializar el nodo en el directorio correspondiente.

```
# Creacion del archivo genesis
$ nano genesis.json

# Inicio del nodo mediante archivo genesis
$ geth --datadir node init genesis.json
```

A continuación, creamos una cuenta mediante la que posteriormente se realizarán las operaciones relativas a la blockchain. También generamos el archivo de configuración del nodo y lo iniciamos.

Es importante mencionar que para iniciar el nodo se utiliza el archivo *start.sh*, cuyo contenido se puede ver (y está explicado) en el anexo B, Archivos. Antes de ejecutar dicho *script*, la dirección de la cuenta creada debe ser incluida en él. Esto se debe al uso del método de consenso PoW, que al minar genera *tokens* y éstos deben poder ser almacenados.

```
$ cd node

# Creacion de cuenta (requiere clave y puede tardar unos segundos)
$ geth --datadir . account new

# Archivo de configuracion
$ geth --datadir . dumpconfig > config.toml

# Creacion y ejecucion del script de inicio
$ nano start.sh
$ sudo chmod +x start.sh
$ ./start.sh
```

Una vez seguidos estos pasos el nodo está en funcionamiento y, de hecho, mediante el último comando, se ha accedido a la consola JavaScript proporcionada por Geth y que permite interactuar con el nodo con una serie de comandos, descritos en el anexo A, Manuales. Como ejemplo, la forma de iniciar y parar el minado se muestra a continuación, aunque no es recomendable hacerlo hasta que se haya ejecutado el proceso de sincronización.

```
// En este caso se utiliza un solo hilo, pero se pueden utilizar mas
// Si todo es correcto, el nodo responde con el mensaje "null"
> miner.start(1)
null
> miner.stop()

// Para salir de la consola.
> exit
```

3.5.3. Sincronización

Para llevar a cabo la sincronización entre nodos, éstos deben ”conocerse”, es decir, deben tener la información de contacto de otro nodo. Esta información de contacto incluye el identificador del nodo, su dirección IP y el puerto en el que está disponible. Es conveniente explicar, aunque se profundiza en ello en el apartado correspondiente del anexo B Archivos, que el puerto por defecto es el 30303, y para ejecutar dos nodos desde la misma máquina es preciso utilizar dos puertos diferentes.

Para extraer la información de un nodo se sigue el proceso mostrado a continuación. Se presupone que el directorio actual es */home/user/eth/node/*.

```
# Direccion IP
$ ./start.sh
```

```
// Identificador y puerto del nodo
> admin.nodeInfo.enode
> exit
```

Una vez extraídos los datos, sustituirse le dirección IP en el identificador del nodo (típicamente *localhost*) por la dirección de la máquina. De esta manera se podrá utilizar posteriormente.

```
// dentificador del nodo
enode://...@127.0.0.1:30303

// Direccion IP
157.88.15.213

// Forma final
enode://...@157.88.15.213:30303
```

Existen dos opciones para conectar dos nodos: hacerlo temporalmente o establecerlos como nodos estáticos para automatizar la conexión. La diferencia es que en el primer caso, cada vez que se para el nodo, se pierde la conexión, mientras que en el segundo caso el nodo se intenta conectar desde el arranque.
En cualquiera de las dos opciones, este proceso debe llevarse a cabo en los dos nodos a conectar, utilizando en cada uno la información del otro.
A continuación se detallan las dos posibilidades, aunque en el caso de este proyecto se ha optado por la segunda.

- **Conexión temporal**
  Ésta se lleva a cabo mediante una orden dentro de la máquina JavaScript de Geth:

```
> admin.addPeer("enode://...@157.88.15.213:30303")
```

- **Conexión permanente**
  Para realizar este tipo de conexión, se debe editar el fichero de configuración *config.toml* generado en el paso de configuración del nodo. Este archivo está compuesto por varias secciones, pero se debe avanzar avanzar hasta la llamada [Node.P2P]. Dento de ella se puede ver la línea StaticNodes = [ ].
  Es aquí donde se pueden definir los nodos a los que conectarse, de la siguiente manera:

```
...
[Node.P2P]
```

```
    ...
    StaticNodes = [
        "enode://...@157.88.15.213:30303",
        "enode://...@157.88.15.228:30305"
    ]
    ...
```

Una vez realizado cualquiera de estos métodos, los nodos se sincronizarán, aunque puede llevarles unos segundos o incluso algún minuto. Se puede proceder a comprobar la sincronización mediante el siguiente comando desde la consola de Geth:

```
// Ver nodos emparejados.
> admin.peers
```

3.5.4. Transacciones

La ejecución de transacciones de manera manual se puede realizar desde la consola JavaScript de Geth, aunque es necesario utilizar una herramienta (incluida en el paquete previamente instalado) llamada Clef.
Clef es una herramienta creada para firmar transacciones en un entorno local, así como para gestionar las cuentas del mismo entorno. En los casos que se tratarán posteriormente, es decir, en la automatización del despliegue e interacción con los contratos, no es necesaria, pero sí lo es en el caso de realizar las transacciones manualmente, ya que éstas deben ser firmadas.

El primer paso es construirlo (si no lo está ya) e inicializarlo.

```
# build
$ make clef

# Inicializar
$ clef init [directorio]
```

Para lanzar la herramienta y utilizarla se proporciona un *script* el anexo B, Archivos, donde se explican los parámetros proporcionados en el inicio.
En la primera ejecución se debe definir una clave maestra para proteger la herramienta, y una vez hecho eso, se puede utilizar.

El funcionamiento es relativamente básico: mientras Clef se ejecuta en un terminal, está esperando a la ejecución de transacciones. Cuando desde la consola de Geth se realiza una de estas, el terminal de Clef pide confirmación para firmar la operación. El usuario da la confirmación, la transacción es firmada y se ejecuta.

Para realizar una transacción se utilizan los siguientes comandos:

```
// Creacion de la transaccion
> var tx = {from: [direccion origen], to: [direccion destino], value:
    web3.toWei(10, "ether")}
```

```
    // Validar creacion de la transaccion desde la consola de Clef

    // Envio de la transaccion
    > eth.sendTransaction(tx)
    // Validar envio de la transaccion desde la consola de Clef
    // Esparar al minado de la transaccion
```

3.5.5. Contrato

La última fase de la implementación del sistema se hace mediante los contratos inteligentes. Éstos actúan como programas desplegados en la red blockchain, con los que se puede interactuar. Para poder ejecutar esas interacciones, el primer paso es crear el contrato mediante el lenguaje de programación Solidity. Después, se compila y se despliega en la red.

A continuación se muestra como ejemplo un contrato sencillo, que permite almacenar un dato, modificarlo y visualizarlo.

```
pragma solidity ^0.8.0;

contract Dato{

  string dato="foo";

  function modificarDato(string memory _dato) public {
     dato=_dato;
  }

  function getDato() public view returns (string memory) {
     return dato;
  }
}
```

En el anexo B, Archivos, se encuentran algunos contratos diseñados para el sistema, y también son explicados más a fondo. Sin embargo con el ejemplo anterior podemos ver el funcionamiento básico de Solidity.

En primer lugar se declara la versión (o versiones) que soportan el contrato. Después, la clase, una variable de clase y las dos funciones que permiten modificarla y visualizarla. En el caso de la visualización, no es necesario ejecutar una transacción, pero para hacer una modificación sí.
Una vez definido el contrato, se debe desplegar. En el caso de este proyecto, ambos despliegue e interacción se hacen mediante Python. Los *scripts* Python utilizados se pueden ver, una vez más, en el anexo B, Archivos. Sin embargo a continuación se describe, por pasos, el proceso.

Despliegue

1. **Compilación**
   Se lee el archivo que contiene el contrato, de extensión *.sol*, y se compila mediante el compilador de solidity para Python, py-solc-x, con la versión adecuada:

```
from solcx import compile_standard, install_solc
install_solc("0.8.0")
compiled_sol = compile_standard({
        "language": "Solidity",
        "sources": {"Storage.sol": {"content": hw_file}},
        "settings": {
            "outputSelection": {
                "*": {
                    "*": ["abi", "metadata", "evm.bytecode",
                        "evm.bytecode.sourceMap"]
                }
            }
        },
},solc_version="0.8.0",)
```

   De esta manera se generan dos archivos, llamados *bin* o *bytecode* y *abi*, mediante los que se desplegará el contrato. Se extraen de la siguiente manera:

```
bytecode = compiled_sol["contracts"]["Storage.sol"]["Storage"]
    ["evm"]["bytecode"]["object"]
abi = json.loads(compiled_sol["contracts"]["Storage.sol"]
    ["Storage"]["metadata"])["output"]["abi"]
```

2. **Conexión a la red**
   Se establece la conexión a la red Ethereum de la siguiente manera:

```
from web3 import Web3

w3 = Web3(Web3.HTTPProvider("http://127.0.0.1:8545"))
# Identificador
chain_id = [Identificador de la blockchain]
# Direccion de la cartera a utilizar
address = [Direccion de la cuenta]
# Clave privada de la cartera. Metodo inseguro
private_key = [Clave Privada]
```

3. **Creación de la transacción de despliegue**
   Se crea la transacción que permitirá introducir el contrato en la red blockchain.

```
# Creacion del contrato
Storage = w3.eth.contract(abi=abi, bytecode=bytecode)
```

```
3
4       # Ultima transaccion para la cartera a usar
5       nonce = w3.eth.get_transaction_count(address)
6
7       # Creacion de la transaccion de despliegue
8       transaction = Storage.constructor().build_transaction(
9           {
10              "chainId": chain_id,
11              "gasPrice": w3.eth.gas_price,
12              "from": address,
13              "nonce": nonce,
14          }
15      )
```

4. **Firma y ejecución de la transacción**
   Mediante las siguientes líneas de código se despliega el contrato y, tras el despliegue, se accede a la dirección en la que ha sido desplegado. Esta dirección será utilizada posteriormente para interactuar con él.

```
1       # Firma de la transaccion
2       sign_transaction = w3.eth.account.sign_transaction(transaction,
            private_key=private_key)
3
4       # Envio de la transaccion
5       transaction_hash =
            w3.eth.send_raw_transaction(sign_transaction.rawTransaction)
6
7       # Esperar al minado de la transaccion
8       transaction_receipt =
            w3.eth.wait_for_transaction_receipt(transaction_hash)
9       print(f"Contrato desplegado. Direccion:
            {transaction_receipt.contractAddress}")
```

Interacción

Para interactuar con el contrato es necesario en primer lugar conocer el previamente mencionado archivo *abi*, por lo que, de realizarse la interacción en un *script* distinto como es nuestro caso, es necesario volver a generarlo. Esto se puede hacer de dos maneras:

- Guardando el archivo a la hora de su compilación y leyéndolo después. Es importante tener en cuanta que el archivo es de tipo JSON.
- Compilando otra vez el contrato y accediendo al *abi* de nuevo. Esta es la manera elegida en este proyecto.

En segundo lugar también es necesario conocer la dirección del contrato, así como los datos ya usados en el despliegue para establecer conexión con la red blockchain. Una vez realizado esto podemos crear una instancia del contrato de la siguiente forma:

```
contrato = w3.eth.contract(address=direccion, abi=abi)
```

Para acceder a la información almacenada mediante un contrato en la red blockchain no es necesario llevar a cabo una transacción, ya qu eno se modifica el estado de la red. Ésto se realiza de la manera mostrada a continuación, tomando como caso el ejemplo de contrato propuesto antes.

```
# Imprimir dato
print("Dato: ", contrato.functions.getDato().call())
```

La modificación del dato sin embargo sí que requiere efectuar una transacción, lo que se hará de la siguiente forma:

```
# Creacion de la transaccion
newDato = instance.functions.modificarDato("bar").build_transaction({
    "chainId": id,
    "from": address,
    "gasPrice": w3.eth.gas_price,
    "nonce": nonce
})

# Firma
firma_newDato = w3.eth.account.sign_transaction(newDato,
    private_key=private_key)

# Envio
envio_newDato =
    w3.eth.send_raw_transaction(firma_newDato.rawTransaction)

# Esperar al minado de la transaccion
recibo_newId =
    w3.eth.wait_for_transaction_receipt(envio_newDato.hex())
print("Transaccion ejecutada")
```

Una vez alcanzado este punto, se puede decir que la red puede operar de forma automática mediante los contratos y los *scripts* de interacción.

3.6. Pruebas

Para corroborar el correcto funcionamiento del sistema, se comprueba la sincronización entre nodos, el acceso a datos de la blockchain, la realización de transacciones y el despliegue y ejecución de contratos. En el caso de que éstos áreas funcionen correctamente se puede asumir con seguridad que el resto de funciones, como el minado o la herramienta Clef, también funcionan con normalidad.

3.6.1. Sincronización

Para comprobar la sincronización simplemente es necesario iniciar los nodos y esperar unos segundos. Después, de ejecuta la siguiente orden en la consola de Geth desde cualquiera de los terminales:

```
1    > admin.peers
```

Tras ejecutar esta orden y ver los datos de los otros nodos de manera correcta se completa esta prueba, obteniendo la siguiente respuesta:

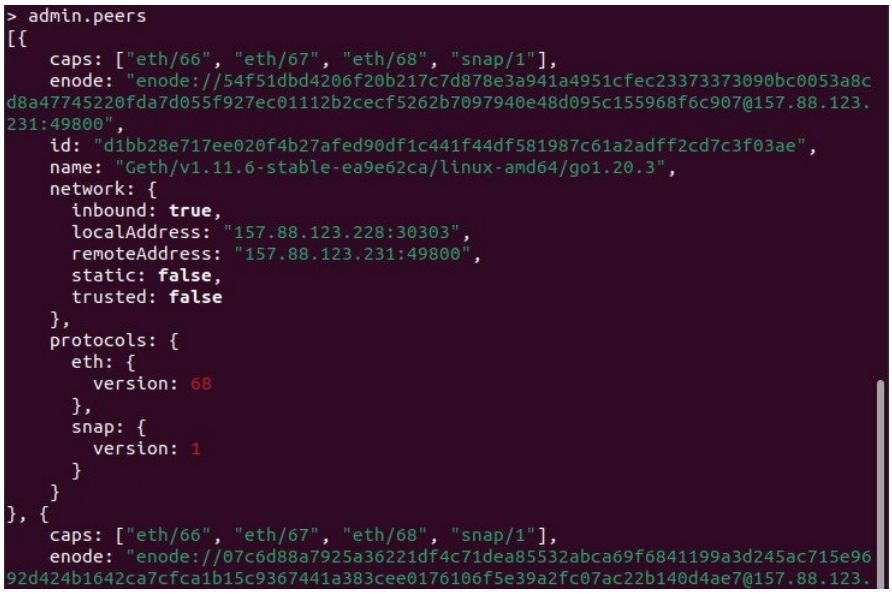

Figura 3.6: Nodos conectados

3.6.2. Acceso a datos

El acceso a datos de la blockchain es presupuesto una vez los nodos están sincronizados, no obstante se comprueba mediante el acceso al balance de varias cuentas.

Desde el nodo A se accede al balance de la primera cuenta de las almacenadas en él y en el nodo B, y desde el nodo B se realiza la misma operación.

Figura 3.7: Acceso a cuentas desde nodo A

```
> eth.accounts[0]
"0x1a22606226c4c3c1f8f8baea1e6ea9b1d774b73d"
> web3.fromWei(eth.getBalance(eth.accounts[0]), 'ether')
1631.367229353
> web3.fromWei(eth.getBalance('0xe93D1031914dF4bfDb478f810D3d762E474ccf70'), 'ether')
3538.691685431
>
```

Figura 3.8: Acceso a cuentas desde nodo B

Además, para acceder a la lista de cuentas mediante el comando *eth.accounts[n]*, es necesaria la confirmación de Clef, como se puede ver a continuación.

```
-------- List Account request--------------
A request has been made to list all accounts.
You can select which accounts the caller can see
  [x] 0xe93D1031914dF4bfDb478f810D3d762E474ccf70
    URL: keystore:///home/nook/eth/node/keystore/UTC--2023-06-02T07-
55-36.833638056Z--e93d1031914df4bfdb478f810d3d762e474ccf70
-------------------------------------------
Request context:
        NA -> ipc -> NA

Additional HTTP header data, provided by the external caller:
        User-Agent: ""
        Origin: ""
Approve? [y/N]:
> y
```

Figura 3.9: Confirmación mediante Clef del acceso a la lista de cuentas

Se confirma así, mediante esta prueba, el acceso de los nodos a la blockchain y el funcionamiento de Clef.

3.6.3. Transacciones

Para comprobar el correcto funcionamiento ded las transacciones se ejecutará primero una transacción utilizando Clef, se comprobará su corrección, y después sin utilizar Clef, donde el resultado esperado es un error.

La primera parte de la prueba consiste en comprobar el balance de las dos cuentas involucradas, crear la transacción, ejecutarla y volver a comprobar los balances.

```
> web3.fromWei(eth.getBalance(eth.accounts[0]), 'ether')
3534.691685431
> web3.fromWei(eth.getBalance('0x06a117E7952Bb1820A784fc8e4F29cB1EA776346'), 'ether')
1905.941085216
> var tx = {from: eth.accounts[0], to: '0x06a117E7952Bb1820A784fc8e4F29cB1EA776346', value:
 web3.toWei(10, "ether")}
undefined
> eth.sendTransaction(tx)
"0xd263e5ef5bccba8e0c552ce666ebe7c52aa6722f7de947b10ded3a26c6484202"
> web3.fromWei(eth.getBalance(eth.accounts[0]), 'ether')

3524.691664431
> web3.fromWei(eth.getBalance('0x06a117E7952Bb1820A784fc8e4F29cB1EA776346'), 'ether')

1915.941085216
>
```

Figura 3.10: Transacción

```
--------- Transaction request------------
to:    0x06a117E7952Bb1820A784fc8e4F29cB1EA776346
from:               0xe93D1031914dF4bfDb478f810D3d762E474ccf70 [chksum ok]
value:              1000000000000000000 wei
gas:                0x5208 (21000)
gasprice: 1000000000 wei
nonce:    0xb1 (177)
chainid:  0x4d2

Request context:
        NA -> ipc -> NA

Additional HTTP header data, provided by the external caller:
        User-Agent: ""
        Origin: ""
-----------------------------------------
Approve? [y/N]:
> y
## Account password

Please enter the password for account 0xe93D1031914dF4bfDb478f810D3d762E474ccf70
>
----------------------
Transaction signed:
 {
    "type": "0x0",
    "nonce": "0xb1",
    "gasPrice": "0x3b9aca00",
    "maxPriorityFeePerGas": null,
    "maxFeePerGas": null,
    "gas": "0x5208",
    "value": "0x8ac7230489e80000",
    "input": "0x",
    "v": "0x9c8",
    "r": "0xab970fec9b80ff3843a997cc06dd03063a3952814588ec75e7e46f07b38932a6",
    "s": "0x4d8389f1df78de0a63efef5e64cd35577912e59f2cb2d4d7b92b3d8e62b433f3",
    "to": "0x06a117e7952bb1820a784fc8e4f29cb1ea776346",
    "hash": "0xd263e5ef5bccba8e0c552ce666ebe7c52aa6722f7de947b10ded3a26c6484202"
  }
```

Figura 3.11: Firma de la transacción mediante Clef

Como se puede ver, los balances reflejan el cambio correcto y la operación es firmada utilizando Clef. Esta herramienta además proporciona los datos de la transacción.
Después, se realiza la misma operación en otro nodo, donde no se utiliza Clef, y se obtiene el siguiente resultado:

```
> var tx = {to: eth.accounts[0], from: '0xe93D1031914dF4bfDb478f810D3d762E474ccf70', val
ue: web3.toWei(10, 'ether')}
undefined
> eth.sendTransaction(tx)

Error: unknown account
        at web3.js:6365:9(39)
        at send (web3.js:5099:62(29))
        at <eval>:1:20(3)

>
```

Figura 3.12: Transacción errónea sin Clef

Viendo que ambos resultados son correctos, se considera esta prueba también exitosa.

3.6.4. Contratos

Para realizar las pruebas de contratos, se crea un contrato sencillo, capaz de almacenar entradas en un *array* y mostrar el mismo, y un *script* Python que permite desplegarlo y ejecutarlo. Esta ejecución consiste concretamente en un bucle, donde cada iteración ejecuta una transacción para guardar en el contrato el tiempo que ha tardado en ejecutarse la transacción anterior. La última transacción, lógicamente, no puede ser guardada. Por último se muestran las 10 entradas.

Ambos archivos se encuentran en el anexo B, Archivos, y la salida que ofrece la ejecución es la siguiente:

```
nook@nookA:~/eth/prueba$ python3 deployment.py
Contrato desplegado. Direccion: 0x8139262C9f9547E0F7E71B1BE3F6Ecc0979b41FA
['9.5367431640625e-07', '4.538058519363403', '1.8434946537017822', '8.8387091159
82056', '16.959954261779785', '5.207155466079712', '2.8018743991851807', '5.7341
77350997925', '11.772989273071289']
```

Figura 3.13: Salida de la ejecución del contrato

3.7. Seguimiento

En la sección 3.1 del presenta capítulo se muestra el plan del proyecto, los tiempos estimados para cada una de las tareas a realizar. También se estudian los riesgos que podrían potencialmente afectar a la ejecución de dicho plan.Este estudio de riesgos no es en vano; la probabilidad de poder ejecutar un proyecto siguiendo con exactitud su planteamiento original es prácticamente nula.

A lo largo de esta sección se mostrarán los problemas encontrados durante el trabajo, sus causas y consecuencias, y, en general, cómo ha sido la ejecución del proyecto.

3.7.1. Estudio

En el planteamiento se asignan dos semanas de trabajo al estudio de los campos y tecnologías a tratar. En la planificación inicial esta fase consta de tres etapas, aunque durante la ejecución del proyecto una breve cuarta etapa fué añadida:

- **Adquisición de conocimiento básico**
  Esta primera fase transcurre con normalidad. Se revisan conceptos clave sobre ambas tecnologías, como la descentralización.
  También se revisan los apuntes de la asignatura Sistemas Distribuidos y Sistemas Empotrados, así como de Redes de Computadoras y Arquitectura de Redes y Servicios, cuyo contenido no está excesivamente ligado a los campos a tratar pero ofrecen un punto de partida sólido.

- **Revisión de estudios y *papers***
  Una vez conocidos y comprendidos los conceptos fundamentales y las bases de Blockchain e IoT, se procede a investigar en profundidad.
  El primer paso dado es la búsqueda de estudios sobre ambas tecnologías, por separado y aplicadas en conjunto. Es en este momento donde se ve que la combinación

de IoT y Blockchain es aún un área con bastante margen de exploración. A pesar de esto, los resultados son suficientes, pero pertenecen en su mayoría a la biblioteca de la IEEE. Aquí se encuentra el primer problema: la falta de una cuenta con acceso a dicha biblioteca. La solución es sencilla, comenzar por aquellos *papers* públicos y acceder más adelante con ayuda de la universidad. Esto sin embargo tampoco fue necesario ya que se utilizó la prueba gratuita ofrecida por la IEEE.

- **Organización de materiales**
  En este punto el proyecto se encuentra llegando a esta fase unas horas antes de lo previsto. Estas horas no son, sin embargo, aprovechadas para ejecutar el estudio de plataformas, si no para organizar los materiales proporcionados para el trabajo. Se organizan los ordenadores, junto con el cableado y periféricos de cada uno, y se comprueba el correcto funcionamiento de cada elemento, desde la pantalla hasta la conexión de red por cable.

- **Estudio de plataformas**
  Este estudio, cuyos resultados pueden observarse en la sección 2.4 Plataformas, se realizó, igual que las dos fases anteriores, con suficiente agilidad para ser terminado unas horas antes de lo previsto. Éstas horas fueron aprovechadas para preparar la primera parte de la siguiente fase, la instalación de software.

3.7.2. Despliegue

Las tres semanas (23 días, concretamente) reservadas para esta fase se separan en siete tareas distintas. Tienen como objetivo obtener un sistema funcional que permita realizar el análisis posterior. A continuación se puede ver el desarrollo real de esta fase. Debido al tiempo que esta fase comprende y la cantidad de tareas y subtareas a realizar en ella, se muestra el avance día a día.

Cada uno de los días se muestra con el siguiente formato:

Fecha	**Tarea Planificada**	**Tarea Ejecutada**
Descripción		

15/06/2023	Instalación	Instalación
Comienzo del proceso de instalación de *software*. Instalación de sistema operativo en las máquinas, configuración del mismo e instalación de Ethereum y Geth. También se instala software adicional, como Truffle para la creación y despliegue de contratos,		

16/06/2023	Instalación	Instalación
Lectura de la documentación del software, principalmente Geth, y configuración inicial y exploración del mismo.		

Tabla 3.12: Seguimiento de la fase de desarrollo I

17/06/2023	Configuración Inicial	Configuración Inicial
Configuración del entorno de trabajo. Creación de cuentas y del archivo *genesis.json*.		

18/06/2023	Configuración Inicial	Despliegue de nodos
Despliegue de nodos locales (en la misma máquina) y sincronización entre ellos, primero temporal, después estática.		

19/06/2023	Despliegue de nodos	Despliegue de nodos
Despliegue de nodos en cada máquina, y sincronización estática entre ellos.		

22/06/2023	Despliegue de nodos	Limpieza
Debido a la cantidad de archivos de archivos basura generados hasta el momento, se decide reiniciar todas las máquinas y realizar un despliegue limpio. Después se comienza a estudiar la realización de transacciones.		

23/06/2023	Sincronización de nodos	Transacciones
Se intentan realizar transacciones entre cuentas mediante el proceso de desbloqueo de cuentas, realización de la transacción y bloqueo de cuentas. Este intento no tiene éxito, ya que el método ha quedado obsoleto en pos de la herramienta Clef. Se inicializa y configura Clef.		

24/06/2023	Sincronización de nodos	Transacciones
Con Clef listo para ser utilizado, se realizan las primeras transacciones y la comprobación de su correctitud. Para simplificar el proceso, se implementan reglas a Clef, dando así luz verde a determinadas transacciones de manera automática.		

25/06/2023	Transacciones	Despliegue de contratos
Exploración de Solidity para crear contratos, y creación de algunos sencillos como prueba y práctica. Se crea el primer proyecto mediante Truffle y se intenta desplegar, sin éxito. El error se debe a la necesidad de Truffle de utilizar Node.js, por lo que se instala junto a npm. Sin embargo los errores prosiguen.		

26/06/2023	Transacciones	Despliegue de contratos
Se continúa intentando solventar el error de Truffle, ésta vez mediante el uso de nvm, un gestor de versiones para npm. Esto es debido a que las últimas versiones de npm presentan problemas de compatibilidad con Truffle y Geth. Aunque los errores varían, el despliegue sigue siendo imposible.		

29/06/2023	Despliegue de contratos	Despliegue de contratos
Dado que la posibilidad de cierto *software* no funcionando correcatmente ya había sido considerada en los riesgos, se ejecuta el plan de contingencia para este caso: el uso de software "plan B". En este caso se ha considerado un despliegue manual mediante JavaScript, utilizando la consola de Geth, junto con una posterior automatización utilizando el mismo lenguaje. El despliegue de esta manera es, sin embargo, fallido una vez más.		

Tabla 3.13: Seguimiento de la fase de desarrollo II

30/06/2023	Despliegue de contratos	Despliegue de contratos
Se intenta de nuevo el despliegue, esta vez utilizando Python. El despliegue es, esta vez, exitoso. Se realizan varios despliegues con varios contratos para confirmar el correcto funcionamiento. Pese a los problemas de esta fase el proyecto continúa por delante de lo previsto.		

31/06/2023	Despliegue de contratos	-
El protocolo Ethereum es actualizado, y el algoritmo de consenso deja de ser PoW y pasa a ser PoS (Proof of Stake). La actualización del *software* se realiza de manera automática y el sistema deja de funcionar. Se ejecuta el plan de acción del riesgo 3.6, y se analiza la posibilidad de actualizar el sistema. Se estima que una reestructuración del mismo para ser adaptado a PoS llevaría alrededor de 5 días, y se desecha la opción.		

01/07/2023	Despliegue de contratos	Instalación
Se reinstala el software en la versión previa y se solucionan los fallos obtenidos tras la actualización. Debido a éstos y al intento de adaptación del sistema al nuevo algoritmo de consenso, se decide realizar una instalación y despliegue limpios, desde cero. De esta manera también se eliminan archivos basura y se limpia el entorno de trabajo.		

02/07/2023	Automatización	Automatización
Se comienzan a ejecutar, mediante Python, interacciones con los contratos desplegados. Éstas sin embargo no son exitosas y no llegan a actuar o siquiera leer la información del contrato. La búsqueda de una solución mediante el *trace* del error es infructuosa, así como lo es mediante el código de error (*'code': -32000, 'message': 'invalid opcode: SHR'*).		

05/07/2023	Automatización	Automatización
Con la ayuda del foro `ethereum.stackexchange.com`, se encuentra el origen del error: la configuración del archivo *genesis.json*. Mientras que ésta permitía crear los nodos y utilizar la red para realizar transacciones, necesitaba cierta configuración adicional para tratar los contratos. Para crear un nuevo archivo génesis se utiliza la herramienta Puppeth. Después, se reinicia la red, se despliega un pcntrato de prueba y se comprueba que, efectivamente, es posible interactuar con él.		

06/07/2023	Automatización	Automatización
Una vez comprobada la viabilidad, se crea un contrato y un *script* python para comprobar la posibilidad de automatización. Se realizan múltiples operaciones y transacciones sin necesidad de intervención y se confirma que funciona.		

07/07/2023	Análisis	Pruebas
Habiendo terminado a tiempo el despliegue y configuración de la red, se deben ejecutar pruebas para comprobar su correcto funcionamiento antes de continuar a la fase de análisis. Éstas se pueden ver en la sección 3.6, Pruebas, y ocupan un día más de lo esperado para ésta fase.		

Tabla 3.14: Seguimiento de fase de desarrollo III

3.7.3. Análisis

Esta última fase incluye la definición del proceso de evaluación, a la que son asignados tres días, la ejecución del mismo, con otros tres días, y dos días mas para extraer conclusiones. Aunque el total es de 8 días, el proyecto cuenta en este punto con un retraso de un día acumulado en la fase anterior, además de encontrarse en un punto de desarrollo del informe notablemente por detrás del estimado.

Así, se comienza realizando la primera parte, en la que se decide cuál va a ser el proceso de análisis. Tras revisar parte del material estudiado al comienzo del proyecto, esta parte es concluida en un día, adelantando así el proyecto y dejando 6 para el resto de la fase.

Las dos tareas restantes terminan ocupando 5 de los 6 días disponibles, con un total de 4 para la ejecución del análisis i 1 para la extracción de conclusiones. Esto se debe a que la ejecución necesita de la creación de contratos y scripts específicos y su despliegue y ejecución, la cual también requiere tiempo. Además, a lo largo de esta tarea se han podido obtener resultados paulatinamente, entre ejecuciones descartadas y exitosas, lo que implica que el proceso de extracción de conclusiones se haya realizado al mismo tiempo.

3.7.4. Informe

A pesar de llegar al punto anterior con un día de ventaja sobre la planificación del proyecto, esta ventaja a sido a expensas del informe, al que se ha dedicado menos tiempo del originalmente planeado. Aunque el retraso en este documento no es pronunciado, sí que acarrea cierto esfuerzo extra. Este esfuerzo se traduce en el día de ventaja que el proyecto llevaba, otros dos para finalizar la redacción y uno más, más adelante, tras haber sido revisado una última vez por el tutor del trabajo.

Capítulo 4

Resultados y Discusión

A lo largo de este capítulo se mostrarán los resultados obtenidos de diferentes pruebas realizadas sobre el sistema, con el fin de conocer sus capacidades y rendimiento. Después se discutirán estos resultados, su significado en términos de posibilidades del sistema y de el éxito de su implementación.

4.1. Resultados

Para obtener una visión sobre el rendimiento de este sistema se analizarán principalmente los tiempos de ejecución de transacciones mediante la interacción con contratos. También, en pruebas separadas, se analizará la pérdida de transacciones en distintas situaciones.

4.1.1. Tiempos

Para obtener los tiempos de ejecución de transacciones se comienza creando el contrato y *script* de despliegue que se pueden ver en la sección B.5, Análisis de Tiempos, del anexo B, Archivos. El contrato permite almacenar una lista de tiempos y otra de valores, mientras que el *script* permite definir el número de transacciones que se realizarán, el archivo donde se guardarán los resultados y, por último, el *payload*.

Se realizarán múltiples ejecuciones de este despliegue, cada una con 100 transacciones, variando los siguientes parámetros:

- **Número de nodos mineros**
  Aunque se mantendrán los tres nodos (uno por cada máquina) activos, conectados y sincronizados entre sí para todas las pruebas, no siempre actuarán como mineros. Así, se iterará entre 1, 2 y 3 mineros.

- **Número de dispositivos IoT simultáneos**
  Se refiere a la cantidad de *scripts* ejecutados a la vez. En este caso variará entre 1 y 3. Éste último caso mantendrá un dispositivo por cada nodo (máquina), minero o no.

- **Payload**
  Como se puede ver en el contrato, la carga de datos se realiza enviando (mediante la transacción correspondiente) un tiempo y un valor. El tiempo, como se ve en el *script*, se refiere a la transacción previa, mientras que el valor es introducido mediante un parámetro en cada ejecución del *script*. Éste valor, al que llamaremos *payload*, tendrá tres variaciones:

  1. *foo*
  2. *mediumpayload*
  3. *heavypayloadusingalongstring*

  Iterando entre estas tres variaciones se puede observar cómo la diferencia de carga puede afectar a los tiempos de ejecución de las transacciones.

Así, permutando las posibilidades de cada variable, obtenemos 18 ejecuciones, desde la más sencilla, con *payload* sencillo, un dispositivo IoT y un nodo minero, hasta la más complicada, con el *payload* largo, 3 dispositivos IoT y 3 nodos mineros.
Como se ha mencionado, se realizarán 100 transacciones por cada ejecución, para obtener así la media de los tiempos que cada una tarda en ser minada.
Como nota antes de abordar los resultados, hay que matizar que, de las 18 ejecuciones mencionadas, la mitad son triples; se ejecutan 3 simulaciones de dispositivos IoT simultáneamente, recibiendo resultados de cada una de las simulaciones.

A continuación se pueden ver los tiempos medios en segundos de cada uno de los casos.

Payload	**Mineros**		
	1	**2**	**3**
1	8.592	6.620	4.767
2	13.594	6.878	5.086
3	20.787	7.583	5.858

Tabla 4.1: Tiempos de ejecución en segundos con un dispositivo IoT

Payload	**Dispositivo**	**Mineros**		
		1	**2**	**3**
1	A	16.426	8.363	6.768
	B	16.433	8.352	6.758
	C	16.417	8.353	6.766
2	A	14.385	8.157	5.950
	B	14.389	8.052	5.956
	C	14.453	8.084	5.909
3	A	21.455	8.206	6.418
	B	20.960	8.354	6.428
	C	20.940	8.293	6.441

Tabla 4.2: Tiempos de ejecución en segundos con tres dispositivos IoT

En el siguiente gráfico se muestra la comparativa de tiempos respecto a los nodos mineros utilizados. En los casos en los que se han ejecutado tres dispositivos simultáneamente, debido a la proximidad de los resultados, se ha utilizado la media total.

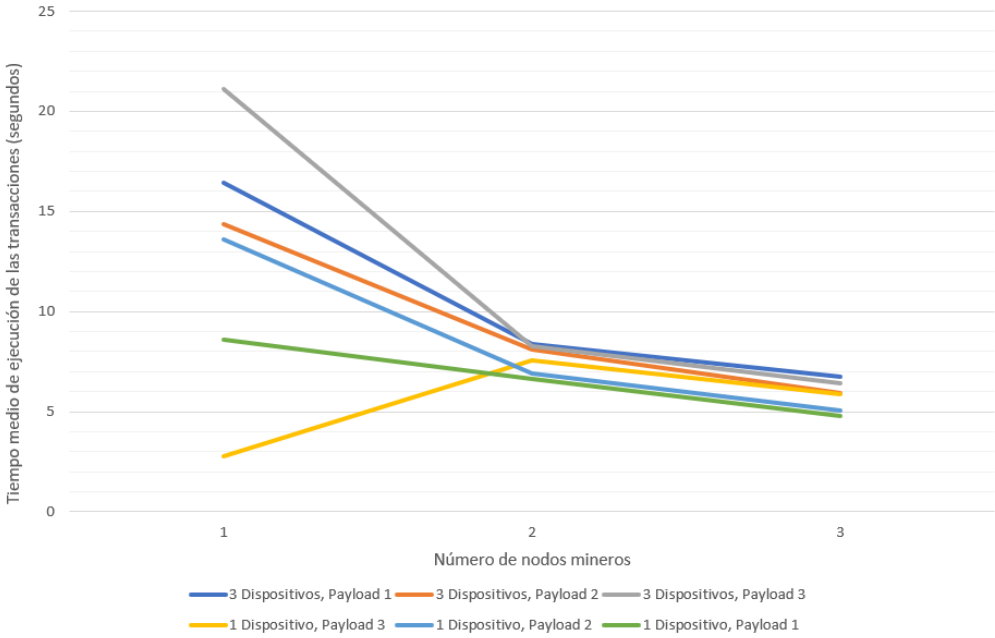

Figura 4.1: Gráfico de tiempos medios respecto a la cantidad de nodos utilizada.

Además de calcular los tiempo medios, se han almacenado también máximos y mínimos.
En el caso de la simulación de un solo dispositivo IoT es remarcable que, en todas las ejecuciones, el tiempo mínimo de transacción es menor de un segundo mientras que el máximo varía más, registrando el mayor pico en 14 segundos para el caso de 3 mineros con el mayor *payload*, y entre 50 y 60 segundos para los casos con un solo minero.
Para las ejecuciones con 3 dispositivos simultáneos los mínimos son similares, por debajo de los 2 segundos en todas las ocasiones y llegando a ser menores a 1 segundo. Los máximos sin embargo varían más: con tres mineros simultáneos se registran picos entre 15 y 30 segundos, mientras que con un solo minero las cifras llegan a sobrepasar el minuto, en varias ocasiones, aunque la mayoría de máximos se sitúan entre los 25 y 50 segundos.

4.1.2. Pérdidas

Mediante el contrato y *script* de despliegue que se pueden ver en la sección B.6, Análisis de Pérdidas, del anexo B, Archivos, se estima la cantidad de transacciones que pueden ser perdidas en situaciones en las que se fuercen las mismas. El contrato permite almacenar una lista de identificadores, mientras que el *script* permite definir el número de transacciones que se realizarán y el tiempo en segundos a esperar entre el lanzamiento de cada iteración.

Se realizarán múltiples ejecuciones de este despliegue, cada una con 100 transacciones, modificando las siguientes variables:

- **Número de nodos mineros**
  Aunque se mantendrán los tres nodos (uno por cada máquina) activos, conectados y sincronizados entre sí para todas las pruebas, no siempre actuarán como mineros. Así, se iterará entre 1 y 3 mineros.

- **Tiempo entre transacciones**
  Se realizarán iteraciones cada 5, 10, 20 y 40 segundos.

Se utilizará sólo un dispositivo y no se modificará su *payload*, al contrario de lo llevado a cabo en la sección anterior, ya que los tiempos de ejecución y las cargas son irrelevantes en este caso. Los problemas derivados del choque entre transacciones de un mismo contrato no variarán en función del número de contratos o la carga de las mismas.

A continuación se pueden ver el número de transacciones perdidas en cada uno de los casos.

Tiempo	**Mineros**		
	1	**2**	**3**
5	12 %	11 %	9 %
10	4 %	7 %	3 %
20	2 %	1 %	4 %
40	0 %	1 %	0 %

Tabla 4.3: Porcentajes de pérdida de transacciones

Tras realizar las ejecuciones, se observa que la pérdida de transacciones ocurre, como era esperado, cuando una no ha sido ejecutada y una nueva llega a "reemplazarla". Esta situación provoca una disputa entre ambas transacciones, eliminando una de ellas e incluso produciendo un incremento del gas necesario para realizar la transacción, en cuyo caso la que haya prevalecido puede fallar también.

4.2. Discusion

Una vez obtenidos los resultados, se puede evaluar su significado. En primer lugar su mera obtención indica que el sistema funciona correctamente, como se ha comprobado previamente en la sección 3.6, Pruebas. Sin embargo estas pruebas, a diferencia de los resultados obtenidos en la sección previa, no permiten analizar otros aspectos de la blockchain desplegada, como su eficiencia o su viabilidad.

En el caso del análisis de tiempos, se pueden ver diferencias pronunciadas entre los tiempos medios. Como ejemplo se pueden ver más de 12 segundos de diferencia entre los diferentes *payloads* de la ejecución de un dispositivo IoT con un solo minero 4.1.
Las diferencias se pronuncian aún más, como es lógico, en el caso de los máximos y los mínimos, aunque éstos, conociendo todos los valores de cada ejecución, se pueden considerar generalmente *outliers*, especialmente en el caso de los máximos. Éstos máximos posan un problema para el que existe solución: el ajuste de los parámetros de la blockchain

(dificultad y algoritmo de consenso). De esta manera todos los tiempos serán menores y las condiciones que permiten a esos máximos aparecer, como la desincronización de los nodos, serán menos habituales o incluso erradicadas.
La existencia de mínimos por debajo de los 2 segundos indica que los parámetros de la red son favorables en ciertas condiciones. Por ende mejorando esos parámetros se pueden asegurar esas condiciones como la situación habitual de la red.

Además, la capacidad computacional de las máquinas utilizadas para desplegar el sistema no dista mucho de la de una amplia gama de dispositivos IoT. Esto nos permite asumir que si cada dispositivo actúa como un nodo la velocidad de la red será baja y estable, como se puede ver en la última columna (la referente a ejecuciones con 3 mineros) de la tabla 4.2. En ella podemos ver cómo el *payload* no provoca variaciones demasiado significativas en los tiempos, que se ven reducidos y aunque presentan máximos y mínimos distantes, son generalmente estables.

De hecho, como se puede observar en la figura 4.1 los tiempos medios se reducen y estabilizan al incluir múltiples nodos mineros en el sistema. Mientras que con uno solo los tiempos son mayores y más dispares, con dos o más convergen. Esto se debe a la dificultad de minado definida para la red, así como a su algoritmo de consenso. Así, aunque la cantidad de dispositivos varíe o la carga de las transacciones aumente, los tiempos de minado seguirán convergiendo hacia el mismo punto, con mayor o menor dificultad.

Las transacciones perdidas, que se pueden observar en la tabla 4.3, deben ser analizadas de otra manera. Aunque un aumento en la velocidad general de ejecución de las transacciones disminuirá la pérdida de las mismas, no asegura que todas sean completadas. Esto se debe a forma de trabajar de Ethereum, síncrona. Así, se deben buscar otros métodos para evitar o enmendar la colisión de transacciones.
La implementación de una cola es un método viable si las transacciones no se generan con más rapidez de la que se ejecutan. En caso de que estos datos se generen de forma muy rápida, pueden ser agrupados en paquetes enviados periódicamente a la red blockchain, o se pueden poner a disposición diferentes métodos para realizar la interacción con el contrato, evitando así las colisiones.

Capítulo 5

Conclusiones

Las primeras valoraciones a extraer sobre este trabajo son relativas al aprendizaje que ha supuesto a nivel teórico y práctico, además de en el ámbito del trabajo de investigación.
A lo largo del grado se abordan las dos tecnologías clave en este proyecto, IoT y blockchain, aunque no se profundiza en ellas. Tras la realización de este trabajo se puede afirmar que se ha adquirido un conocimiento más avanzado de ambas, incluso integral en el caso de blockchain. Se ha entrado en sus bases, la motivación de su existencia, su funcionamiento e incluso su estado del arte, proporcionando así no solo una base sólida para realizar este trabajo, si no también una comprensión suficiente para desarrollar trabajo futuro con relativa facilidad.
Además, mediante la implementación del sistema, se han abordado los componentes del mismo y los procesos que permiten ponerlo en marcha con las condiciones deseadas. Así, se ha aprendido no sólo a desplegar el sistema en cuestión si no a desplegar casi cualquier sistema adaptado a unas condiciones determinadas siempre y cuando se sitúe en el marco de las tecnologías utilizadas en este proyecto.
Por último también se ha aprendido a desarrollar de manera relativamente independiente un proyecto de investigación. Aunque con ayuda puntual del tutor, el alumno ha planteado, planificado y desarrollado todas las fases de este trabajo, y lo ha hecho dentro de los estándares de un proyecto investigativo.
Así, la primera conclusión extraída de este trabajo, como se ha mencionado, son relativas al aprendizaje; se ha aprendido a utilizar las tecnologías IoT y blockchain, y se ha aprendido a hacerlo de manera independiente y bajo el marco de una investigación.

Estas nuevas capacidades permiten al alumno ensanchar sus miras, comprender un poco más y mejor el funcionamiento de las investigación académica y en este caso, debido al atractivo que ésta ha supuesto, considerarla incluso como posibilidad de futuro. Especialmente viendo cómo las tecnologías abordadas aún tienen un largo camino por recorrer, camino debido principalmente al factor de su edad. Al ser tecnologías jóvenes aún no han visto explotadas o incluso consideradas todas su posibilidades, y su carácter constantemente cambiante (especialmente en el caso de la blockchain) no facilita ese desarrollo en investigación.

Respecto al sistema, sus viabilidad en otros entornos como el industrial, su escalabilidad y sus posibilidades, se puede decir que la investigación ha proporcionado resultados positivos, incluso exitosos. Los tiempos vistos en la sección 4.1, Resultados, son estables y relativamente bajos, y como se explica en la sección 4.2, Discusión, mejorables. Así, se puede argumentar que, aunque aún es necesario mas trabajo investigativo, la implementación conjunta de las tecnologías IoT y blockchain obtiene resultados prometedores. Los beneficios de ambas tecnologías se complementan y su aplicación puede demostrar mucha utilidad en ciertos entornos, como el de la Industria 4.0 [2].

Se puede afirmar que este proyecto ha sido no sólo enriquecedor en múltiples niveles, si no también el resultado de una investigación con un amplio abanico de posibilidades futuras.

5.1. Líneas de trabajo futuras

La primera tarea a realizar en el caso de implementar el sistema desarrollado en este proyecto, es afinar sus características y parámetros. Aunque en este caso han sido ajustadas hasta cierto punto, el margen de mejora es amplio y no ha sido abordado por la falta de tiempo.
La otra tarea principal que mejoraría las posibilidades de una red como la planteada es la definición de contratos inteligentes con mayor complejidad y posibilidades, además de estructuras de datos más favorables para un sistema blockchain.

Estos perfeccionamientos pueden ser llevados a cabo mediante la simulación de dispositivos IoT definida en este proyecto, pero otra posibilidad es incluir en el sistema más dispositivos reales. Esto puede ser realizado mediante máquinas enfocadas al ámbito IoT (Raspberry Pi, por ejemplo) y aportaría un grado de realismo mayor. Así, los resultados serían más aproximados a un sistema real y por ende su implementación más cercana.

Por último, aunque en este caso se ha utilizado como plataforma Ethereum, el análisis de otras es más que necesario para comprender cuáles son de mayor utilidad para cada situación. La comparativa entre más plataformas es clave y por tanto una de las principales líneas a seguir en el futuro próximo.

Bibliografía

[1] Muhammed Ali. How to Deploy and Interact with Solidity Contracts with Python and Ganache. https://coinsbench.com/how-to-deploy-and-interact-with-solidity-contracts-with-python-and-ganache-be63334323e6, Mayo 2022.

[2] Tejasvi Alladi, Vinay Chamola, Reza M. Parizi, and Kim-Kwang Raymond Choo. Blockchain applications for industry 4.0 and industrial iot: A review. *IEEE Access*, 7:176935–176951, 2019.

[3] Praveen Bhosale. Blockchain explained and its application to payments. `https://www.paiementor.com/blockchain-explained-application-payments/`, Febrero 2020.

[4] Aaron Bloomfield. Geth Command Summary. `https://aaronbloomfield.github.io/ccc/docs/geth_reference.html`.

[5] D.L. Chaum. *Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups*. University of California, Berkeley, 1982.

[6] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[7] Brian Cockfield. IoT With The Ethereum Blockchain. `https://hackaday.com/2017/11/09/iot-with-the-ethereum-blockchain/`, Noviembre 2017.

[8] Ali Dorri, Salil Kanhere, Raja Jurdak, and Praveen Gauravaram. Blockchain for iot security and privacy: The case study of a smart home. In *Blockchain for IoT Security and Privacy: The Case Study of a Smart Home*, 03 2017.

[9] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Blockchain in internet of things: Challenges and solutions. *IEEE Access*, 2016.

[10] Ali Dorri, Salil S. Kanhere, Raja Jurdak, and Praveen Gauravaram. Blockchain for iot security and privacy: The case study of a smart home. *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 618–623, 2017.

[11] Said Eloudrhiri. Create a private Ethereum blockchain with IoT devices. https://chainskills.com/2017/02/24/create-a-private-ethereum-blockchain-with-iot-devices-16/, Febrero 2017.

[12] Hyperledger Foundation. Documentación de Hyperledger Fabric. https://hyperledger-fabric.readthedocs.io/en/release-2.5/, 2023.

[13] Daniel Fuentes-Morales, Jorge Flores-Velazquez, and Rodrigo Roblero. *La web de las cosas (WoT) para monitoreo y control de biosistemas con aplicaciones prácticas en Agricultura Urbana (AgUrban)*, pages 135–154. ResearchGate, 03 2022.

[14] Omar S. Hiremath. Execute Ethereum Smart Contract on Private Network with Truffle. https://www.edureka.co/blog/ethereum-smart-contract-project#InstallingPrerequisites, Noviembre 2022.

[15] The Go-Ethereum authors. Documentación de Go-Ethereum. https://geth.ethereum.org/docs, Diciembre 2022.

[16] The Go-Ethereum authors. Introducción a Clef. https://geth.ethereum.org/docs/tools/clef/introduction, Diciembre 2022.

[17] IBM. IBM Watson IoT Platform. https://internetofthings.ibmcloud.com/.

[18] IBM. Hyperledger Fabric for Trusted IoT. https://github.com/IBM/Hyperledger-Fabric-for-Trusted-IoT, 2023.

[19] ConsenSys Software Inc. Documentación de Truffle. https://trufflesuite.com/docs/truffle/, 2022.

[20] IOTA. Documentación de IOTA. https://wiki.iota.org/, 2023.

[21] Raj Koshik. Extracción de la clave privada de una cuenta ethereum. https://ethereum.stackexchange.com/questions/12830/how-to-get-private-key-from-account-address-and-password, Junio 2018.

[22] Laphou Lao, Zecheng Li, Songlin Hou, Bin Xiao, Songtao Guo, and Yuanyuan Yang. A survey of iot applications in blockchain systems: Architecture, consensus, and traffic modeling. *ACM Comput. Surv.*, 53(1), feb 2020.

[23] Sarah Laoyan. Qué es la metodología waterfall y cuándo utilizarla. https://asana.com/es/resources/waterfall-project-management-methodology, Septiembre 2022.

[24] Han Liu, Dezhi Han, and Dun Li. Fabric-iot: A blockchain-based access control system in iot. *IEEE Access*, 8:18207–18218, 2020.

[25] Canonical Ltd. Instalación de Ubuntu desktop. https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview, 2022.

[26] Filament PD Ltd. Internet of Things — Filament. https://www.filamentpd.com/internet-of-things, 2021.

[27] Sebastian Ma. Getting Started With Ethereum Private Blockchain. https://dzone.com/refcardz/getting-started-with-etherium-private-blockchain.

[28] Dillion Mejida. Instalación de Node Version Manager. https://www.freecodecamp.org/news/node-version-manager-nvm-install-guide/, Septiembre 2022.

[29] Yuri Musienko. How to implement blockchain in iot? https://merehead.com/blog/implement-blockchain-in-iot/, Marzo 2023.

[30] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2008.

[31] Oscar Novo. Blockchain meets iot: An architecture for scalable access management in iot. *IEEE Access*, 5:1184–1195, 03 2018.

[32] Chukwuebuka Okonkwo. A Guide to Private Ethereum Mining with Geth (Go-Ethereum). https://dev.to/heydamali/a-guide-to-private-ethereum-mining-with-geth-go-ethereum-13ol, Octubre 2021.

[33] Mercury Protocol. How To: Deploy Smart Contracts Onto The Ethereum Blockchain. https://medium.com/mercuryprotocol/dev-highlights-of-this-week-cb33e58c745f, Diciembre 2017.

[34] Matevž Pustišek and Andrej Kos. Approaches to front-end iot application development for the ethereum blockchain. *Procedia Computer Science*, 129:410–419, 2018. 2017 INTERNATIONAL CONFERENCE ON IDENTIFICATION,INFORMATION AND KNOWLEDGEIN THE INTERNET OF THINGS.

[35] Adrián Rabadán. Gestión de riesgos de proyectos software. https://es.wikiversity.org/w/index.php?title=Gesti%C3%B3n_de_riesgos_de_proyectos_software&oldid=154406, 2022.

[36] Kinza Shafique, Bilal A. Khawaja, Farah Sabir, Sameer Qazi, and Muhammad Mustaqim. Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. *IEEE Access*, 8:23022–23040, 2020.

[37] Hardik Shah. Building Industrial IoT with IOTA: Introduction and How IOTA Works. https://www.simform.com/blog/industrial-iot-iota-part-1/, Junio 2022.

[38] I2T Smart. El internet de las cosas. https://i2t.com.mx/internet-de-las-cosas/.

[39] Corwin Smith. Introducción a web3. https://ethereum.org/es/web3/, Junio 2023.

[40] Solbyte. La tendencia hacia el internet de las cosas. https://www.solbyte.com/blog/la-tendencia-hacia-el-internet-de-las-cosas/, 2014.

[41] Rodrigo García Trejo. Creación de una Red Blockchain local de Ethereum con GETH y PUPPETH. https://dev.to/fynio/crear-un-blockchain-local-de-ethereum-con-geth-y-puppeth-fdi, Octubre 2020.

[42] Michael A. Walker, Abhishek Dubey, Aron Laszka, and Douglas C. Schmidt. Platibart: A platform for transactive iot blockchain applications with repeatable testing. In *PlaTIBART: A Platform for Transactive IoT Blockchain Applications with Repeatable Testing*, M4IoT '17, page 17–22, New York, NY, USA, 2017. Association for Computing Machinery.

[43] The web3.js developers. Documentación de web3.js - Ethereum JavaScript API 1.0.0. https://web3js.readthedocs.io/en/v1.10.0/, 2022.

[44] The web3.py developers. Documentación de web3.py 6.5.0. https://web3py.readthedocs.io/en/stable/, 2022.

[45] Wikipedia. Internet de las cosas — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Internet_de_las_cosas&oldid=152279877, 2023.

[46] Elevate X. The 4 blockchain types explained. https://elevatex.de/blog/web3/4-blockchains-types-explained/, Enero 2022.

[47] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, 2014.

[48] Jinxin Zhang and Meng Wu. Blockchain use in iot for privacy-preserving anti-pandemic home quarantine. *Electronics*, 9(10), 2020.

[49] Yujian Zhang and Daifu Liu. Toward vulnerability detection for ethereum smart contracts using graph-matching network. *Future Internet*, 14(11), 2022.

[50] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, jan 2018.

Apéndice A

Manuales

A.1. Manual de instalación y despliegue

Dado que ambos procesos de instalación y despliegue han sido descritos a lo largo de la sección 3.5, Implementación, no es necesario añadir nada en esta sección.

Para mayor detalle sobre los *scripts* y archivos utilizados en la instalación, ver el anexo B, Archivos.

A.2. Geth

A continuación se muestra un resumen de los comandos más comunes de esta herramienta.

- **Bloque actual**

```
1   eth.blockNumber
```

- **Inicio y parado del minado**

```
1   miner.start(numerohilos)
2   miner.stop()
```

- **Cuenta n del nodo**

```
1   eth.account[n]
```

- **Cuenta n del nodo**

```
1   eth.account[n]
```

- **Balance de cuenta (wei)**

```
eth.getBalance(cuenta)
```

- **Balance de cuenta (eth)**

```
web3.fromWei(eth.getBalance(cuenta), "ether")
```

- **Balance de la cuenta destino de los beneficios de minado del nodo**

```
eth.getBalance(eth.coinbase)
```

- **Crear transacción**

```
transaccion = {from: cuentaorigen, to: cuentadestino, value:
    cantidad}
```

- **Enviar transacción**

```
eth.sendTransaction(transaccion)
```

- **Información de transacción**

```
eth.getTransactionReceipt(transaccion)
```

- **Transacciones pendientes**

```
eth.pendingTransactions
```

Se puede encontrar más material en la documentación de Geth [15] y en el resumen realizado por Aaron Bloomfield [4].

A.3. Extracción de Clave Privada

En los *scripts* Python utilizados para el despliegue 3.5 este proyecto se define la clave privada de la cuenta utilizada para firmar las transacciones en texto plano. Aunque éste es un método inseguro, se ha elegido debido al carácter investigativo del trabajo y la privacidad de la misma red blockchain. Para utilizar la clave privada en texto plano, ésta debe ser extraída del archivo apropiado [21].

Para realizar esta extracción, se puede utilizar el siguiente proceso:

```
$ pip install web3
```

```
>>> from web3.auto import w3
>>> with open("~/.eth/node/keystore/UTC--...") as keyfile:
...     encrypted_key = keyfile.read()
...     private_key = w3.eth.account.decrypt(encrypted_key,
    'password')
>>> import binascii
>>> binascii.b2a_hex(private_key)
```

Apéndice B

Archivos

B.1. genesis.json

```
{
    "nonce": "0x0000000000000042",
    "timestamp": "0x00",
    "parentHash":
    "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": "0x00",
    "gasLimit": "0x8000000",
    "difficulty": "0x400",
    "mixhash":
    "0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x3333333333333333333333333333333333333333",
    "alloc": {},
    "config":{
      "chainId": 1234,
      "homesteadBlock":0,
      "byzantiumBlock": 0,
      "constantinopleBlock": 0,
      "eip145Block": 0,
      "eip150Block": 0,
      "eip155Block": 0,
      "eip158Block": 0
    }
}
```

- **config**
  Configuración de la blockchain. Inicio de ciertos bloques necesarios para el despliegue.
  - chainId:

Identificador de la cadena y protección contra ataques *replay*.

- Bloques eip:
  Ethereum Improvement Proposal. Contribuciones de desarrolladores al proyecto Ethereum. El valor 0 indica que se están utilizando.
- Otros bloques:
  Actualizaciones importantes de Ethereum. El valor 0 implica que se están utilizando.

- **difficulty**
  dificultad de minado. Un valor más bajo permite minado más rápido y menor dificultad, que implica menor seguridad.
- **gasLimit**
  El límite del coste de tokens por cada bloque. un valor alto es recomendado para la ejecución de pruebas.
- **alloc**
  Cuentas con saldo desde el inicio. Innecesarias en el caso de este proyecto.
- **coinbase**
  Cantidad de tokens total que se pueden repartir por el minado de bloques a los nodos.

El resto de valores indican una blockchain estándar, con un nonce bajo, el hash previo y los datos extra, vacíos también en este caso.

B.2. start.sh

Este *script* sirve para ejecutar el nodo con las configuraciones necesarias.

```
1  #!/bin/bash
2
3  geth
4      # Directorio del nodo
5      --datadir .
6      # Permitir el desbloqueo de las cuentas a pesar de inseguridad
7      --allow-insecure-unlock
8      # Directorio de las claves
9      --keystore keystore
10     # Nombre del socket para conectarse al nodo
11     --ipcpath node
12     # Desactivar el mecanismo de descubrimiento de pares
13     --nodiscover
14     # Identificador de la red
15     --networkid 1234
16     # Activar el servido HTTP-RPC
17     --http
18     # Dominios desde los que aceptar peticiones de origen cruzado
```

```
    --http.corsdomain "*"
    # APIs de la interfaz HTTP-RPC a utilizar
    --http.api 'personal,eth,net,web3,txpool,miner'
    # Activar minado
    --mine
    # Cuenta donde recibir los beneficios del minado
    --miner.etherbase=[account address]
    # Fichero de configuracion
    --config config.toml
    # Ubicacion del fichero .ipc de Clef.
    --clef=clef/clef.ipc

    # log de la ejecucion
    console 2> node.log
```

Es importante mencionar las *flags* –port y –http.port. Sus valores por defecto son 30303 y 8545 respectivamente, pero en caso de ejecutar dos nodos en una máquina, no pueden ser los mismos. Además, en caso de utilizar la *flag* de Clef, clef debe estar en funcionamiento antes de ejecutar el nodo.

El resto de posibles opciones se pueden ver mediante el comando *geth –help*.

B.3. Clef

La ejecución de Clef se realiza mediante el siguiente *script*:

```
#!/bin/bash

clef
    # Identificador de la blockchain
    --chainid 1234
    # Directorio de las claves
    --keystore keystore
    # Directorio de Clef
    --configdir clef
    # Activacion de HTTP
    --http
```

B.4. Contrato para pruebas

A continuación se muestra el contrato, llamado *Prueba.sol* y el *script* Python utilizados para realizar la prueba vista en la sección 3.6.
Es importante tener en cuenta que las variables como *w3*, *chain_id*, *address* o *private_key* dependen del sistema blockchain sobre el que se ejecuta el contrato.

- **Contrato**

```
//SPDX License-Identifier: GPL 3.0

pragma solidity ^0.8.0;

contract Prueba{

  string[] public hist;

  function addInput(string memory _input) public {
     hist.push(_input);
  }

  function getInfo() public view returns (string[] memory) {
     return hist;
  }
}
```

- ***Script* de despliegue**

```
import json
import calendar
import time
from solcx import compile_standard, install_solc
from web3 import Web3
from web3.exceptions import TimeExhausted

# Lectura del contrato
with open("Prueba.sol", "r") as file:
    hw_file = file.read()

# Compilado
install_solc("0.8.0")
compiled_sol = compile_standard(
    {
        "language": "Solidity",
        "sources": {"Prueba.sol": {"content": hw_file}},
        "settings": {
            "outputSelection": {
                "*": {
                    "*": ["abi", "metadata", "evm.bytecode",
                        "evm.bytecode.sourceMap"]
                }
            }
        },
    },
    solc_version="0.8.0",
```

```
27      )
28
29      # Codigo compilado
30      with open("compiled_code.json", "w") as file:
31          json.dump(compiled_sol, file)
32
33      # Extraccion de archivos bin (bytecode) y abi
34      bytecode = compiled_sol["contracts"]["Prueba.sol"]["Prueba"]
35          ["evm"]["bytecode"]["object"]
36      abi = json.loads(compiled_sol["contracts"]["Prueba.sol"]
37          ["Prueba"]["metadata"])["output"]["abi"]
38
39      # Conexion a la blockchain
40      w3 = Web3(Web3.HTTPProvider("http://127.0.0.1:8545"))
41      chain_id = 1234
42      address = "0x..."
43      private_key = "0x..."
44
45      # Creacion del contrato
46      Storage = w3.eth.contract(abi=abi, bytecode=bytecode)
47      nonce = w3.eth.get_transaction_count(address)
48
49      # Despliegue
50      transaction = Storage.constructor().build_transaction({
51        "chainId": chain_id,
52        "gasPrice": w3.eth.gas_price,
53        "from": address,
54        "nonce": nonce,
55      })
56      sign_transaction = w3.eth.account.sign_transaction(transaction,
            private_key=private_key)
57      transaction_hash =
            w3.eth.send_raw_transaction(sign_transaction.rawTransaction)
58      transaction_receipt =
            w3.eth.wait_for_transaction_receipt(transaction_hash)
59      print(f"Contrato desplegado. Direccion:
            {transaction_receipt.contractAddress}")
60      ca = transaction_receipt.contractAddress
61
62      # Instancia del contrato
63      instance = w3.eth.contract(address=ca, abi=abi)
64
65      # Subida de datos
66      init = time.time()
67      end = time.time()
68      for i in range(0, 9):
69        try:
```

```
70          valor = str(end-init)
71          nonce = w3.eth.get_transaction_count(address)
72          dato =
                instance.functions.addInput(valor).build_transaction({
73              "chainId": chain_id,
74              "from": address,
75              "gasPrice": w3.eth.gas_price,
76              "nonce": nonce
77          })
78          f_dato = w3.eth.account.sign_transaction(dato,
                private_key=private_key)
79          e_dato = w3.eth.send_raw_transaction(f_dato.rawTransaction)
80          init = time.time()
81          r_dato = w3.eth.wait_for_transaction_receipt(e_dato.hex())
82          end = time.time()
83      except TimeExhausted as e:
84          print("TimeExhausted")
85
86      # Visualizacion de datos
87      print(instance.functions.getInfo().call())
```

B.5. Análisis de Tiempos

A continuación se muestra el contrato y el *script* Python utilizados para realizar el análisis de tiempos visto en la sección 4.1. El *script* recibe tres argumentos, el nombre del archivo donde se guardará la salida, el número de iteraciones del bucle donde se realiza la transacción y el payload.

- **Contrato**

```
1  //SPDX License-Identifier: GPL 3.0
2
3  pragma solidity ^0.8.0;
4
5  contract Prueba{
6
7      string[] tiempos;
8      string[] valores;
9
10     function addInput(string memory _tiempo, string memory _valor)
           public {
11         tiempos.push(_tiempo);
12         valores.push(_valor);
13     }
14
15     function getTiempos() public view returns (string[] memory) {
```

```
        return tiempos;
    }

    function getValores() public view returns (string[] memory) {
        return valores;
    }

    function getCount() public view returns (uint count) {
        return valores.length;
    }
}
```

- ***Script* de despliegue** Dado que desde la lectura del contrato hasta su despliegue se realiza de manera idéntica a la realizada en la sección anterior, Contrato para pruebas, éstas no se muestran a continuación aunque son necesarias.

```
    # argv1: output
    # argv2: iteraciones
    # argv3: payload
    import os
    import sys
    import json
    import calendar
    import time
    import array as arr
    from solcx import compile_standard, install_solc
    from web3 import Web3
    from web3.exceptions import TimeExhausted

    # Lectura del contrato
    # Compilado
    # Extraccion de archivos bin (bytecode) y abi
    # Conexion a la blockchain
    # Creacion del contrato
    # Despliegue

    # Instancia del contrato
    instance = w3.eth.contract(address=ca, abi=abi)

    total=float(0)
    max=float(0)
    min=float(50)

    # Subida de datos
    total = 0;
    for i in range(0, int(sys.argv[2])):
      try:
```

```
32          nonce = w3.eth.get_transaction_count(address)
33          dato = instance.functions.addInput(str(end-init),
                str(sys.argv[3])).build_transaction({
34              "chainId": chain_id,
35              "from": address,
36              "gasPrice": w3.eth.gas_price,
37              "nonce": nonce
38          })
39          f_dato = w3.eth.account.sign_transaction(dato,
                private_key=private_key)
40          e_dato = w3.eth.send_raw_transaction(f_dato.rawTransaction)
41          init = time.time()
42          r_dato = w3.eth.wait_for_transaction_receipt(e_dato.hex())
43          end = time.time()
44          cur=(end-init)
45          total+=cur
46          if(cur<min):
47              min = cur
48          if(cur>max):
49              max = cur
50      except TimeExhausted as e:
51          print("TimeExhausted")
52
53  # Visualizacion de datos
54
55  f = open("./results/{}.txt".format(str(sys.argv[1])), "w")
56  f.write(f"{transaction_receipt.contractAddress}")
57  f.write("\nExitosos: ")
58  f.write(str(instance.functions.getCount().call()))
59  f.write("\nMedia: ")
60  f.write(str(float(total/instance.functions.getCount().call())))
61  f.write("\nMinimo: ")
62  f.write(str(min))
63  f.write("\nMaximo: ")
64  f.write(str(max))
65  f.write("\nTodos los tiempos: ")
66  f.write(str(instance.functions.getTiempos().call()))
67  f.write(str(cur))
68  f.close()
69  print("Ejecucion terminada")
```

B.6. Análisis de Pérdidas

A continuación se muestra el contrato y el *script* Python utilizados para realizar el análisis de pérdidas visto en la sección 4.1. El *script* recibe dos argumentos, el número de iteraciones del bucle donde se realiza la transacción y el tiempo en segundos a esperar

entre el lanzamiento de cada iteración.

- **Contrato**

```
//SPDX License-Identifier: GPL 3.0

pragma solidity ^0.8.0;

contract Prueba{

   string[] ids;

   function addInput(string memory _id) public {
      ids.push(_id);
   }

   function getIds() public view returns (string[] memory) {
      return ids;
   }

   function getCount() public view returns (uint count) {
      return ids.length;
   }
}
```

- ***Script* de despliegue** Al igual que en la sección anterior, se han ommitido ciertas partes del *script* que son mostradas en la sección Contato para Pruebas.

```
    # argv1: iteraciones
    # argv2: tiempo entre iteraciones
    import os
    import sys
    import json
    import calendar
    import time
    import array as arr
    from solcx import compile_standard, install_solc
    from web3 import Web3
    from web3.exceptions import TimeExhausted

    # Lectura del contrato
    # Compilado
    # Extraccion de archivos bin (bytecode) y abi
    # Conexion a la blockchain
    # Creacion del contrato
    # Despliegue

    # Instancia del contrato
```

```
21    instance = w3.eth.contract(address=ca, abi=abi)
22
23    # Subida de datos
24    for i in range(0, int(sys.argv[1])):
25      pid = os.fork()
26      if pid > 0:
27         time.sleep(int(sys.argv[2]))
28      else:
29         try:
30            nonce = w3.eth.get_transaction_count(address)
31            dato = instance.functions.addInput(str(i))
32                .build_transaction({
33                    "chainId": chain_id,
34                    "from": address,
35                    "gasPrice": w3.eth.gas_price,
36                    "nonce": nonce
37                })
38            f_dato = w3.eth.account.sign_transaction(dato,
                  private_key=private_key)
39            e_dato =
                  w3.eth.send_raw_transaction(f_dato.rawTransaction)
40            r_dato = w3.eth.wait_for_transaction_receipt(e_dato.hex())
41            print(i)
42         except TimeExhausted as e:
43            print("TimeExhausted")
44         sys.exit()
45
46    # Visualizacion de datos
47
48    f = open("./results/{}s.txt".format(str(sys.argv[2])), "w")
49    f.write(f"{transaction_receipt.contractAddress}")
50    f.write("\nExitosos: ")
51    f.write(str(instance.functions.getCount().call()))
52    f.write("\nTodos: ")
53    f.write(str(instance.functions.getIds().call()))
54    f.close()
55    print("Ejecucion terminada")
```

Apéndice C

Figuras adicionales

C.1. Diagrama de Gantt

A continuación se muestra en mayor detalle el diagrama de Gantt desarrollado en la sección 3.1, Plan.

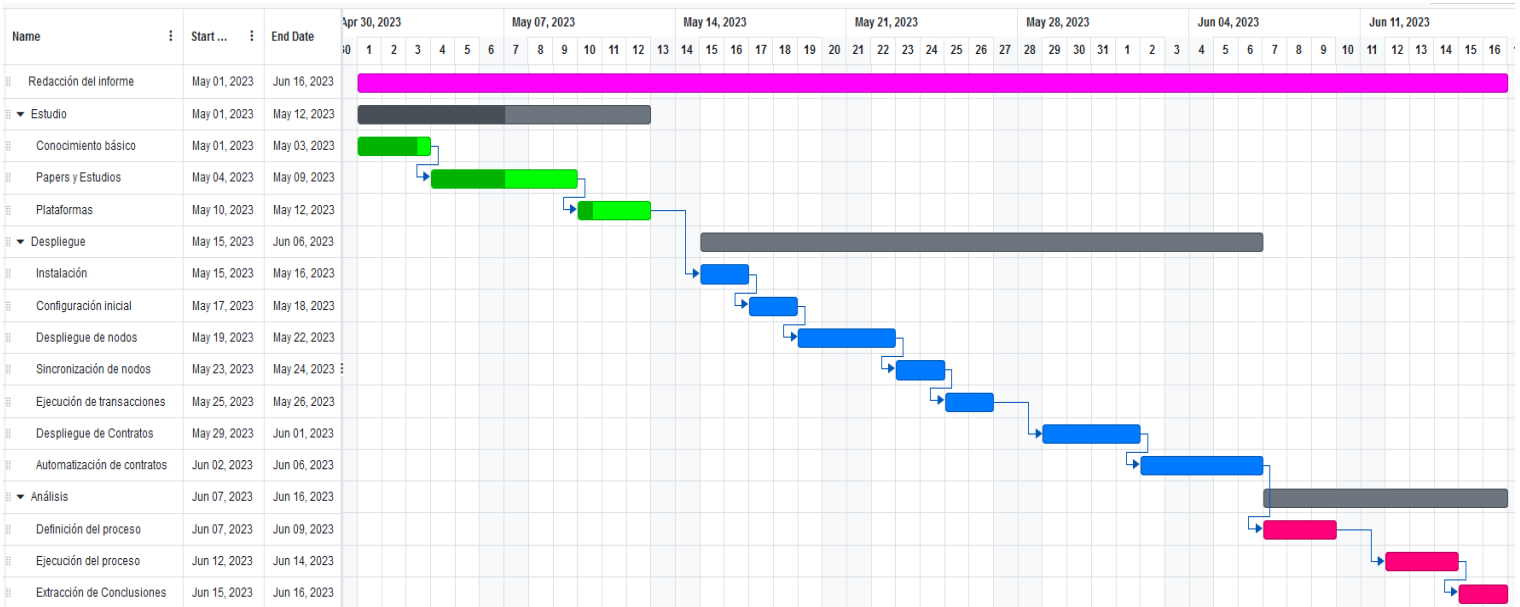

Figura C.1: Diagrama de Gantt detallado