uc3m | Universidad Carlos III de Madrid

Máster Universitario en Ingeniería Informática
2018-2019

Trabajo Fin de Máster

"Diseño e implementación de un modelo de tokenización de acceso a red basado en Ethereum Blockchain"

César Rodríguez Cerro

Tutor

Juan Miguel Gómez Berbis

1.2.C16 – 26.09.2019

"La mente humana no es capaz de concebir la cuarta dimensión, así que, ¿cómo puede concebir a Dios? Para quien mil años y mil dimensiones son sólo una"

Albert Einstein

Resumen

Tras la publicación del artículo en el que se presentó Bitcoin, allá por el año 2008, la tecnología Blockchain irrumpió con fuerza en nuestra sociedad tecnológica. Los beneficios que aporta esta tecnología a las soluciones que la incorporan son múltiples, pero podrían resumirse en inmutabilidad, trazabilidad y transparencia de la información. Entre las capacidades que quizás más valor aportan se encuentra la tokenización, mediante la cual, se puede representar digitalmente cualquier activo de la vida real. De esta manera, el valor digital del activo puede ser administrado en una red Blockchain manteniendo siempre el control de la propiedad.

En este contexto surge el presente proyecto, con el que se pretende realizar un estudio profundo del ecosistema Blockchain para posteriormente generar una solución que permita comprender si la tecnología aporta un valor real. En concreto se propone una prueba de concepto en la que se introduce la capacidad de tokenización en la gestión del acceso y monitorización de las redes de ordenadores con el respaldo de la plataforma de Blockchain de Ethereum.

Abstract

Since the publication of the Bitcoin whitepaper back in 2008, Blockchain technology has burst onto our technological society. This technology brings multiple benefits to it, mainly immutability, traceability and transparency of the information. Among the capabilities of the technology that perhaps provide a grater value is tokenization, through which it is posible to digitally represent any real-life asset. In this way, the digital value of the asset can be managed in a Blockchain network maintaining always its ownership control.

In this context, the present project's aim is to carry out a deep study of the Blockchain ecosystem in order to later generate a solution that allows to understand if the technology provides a real value. Specifically, a proof of concept is proposed that introduces the capacity of tokenization in the management of access and monitoring of computer networks with the support of the Ethereum Blockchain platform.

Índice de contenidos

Índice de tablas

Índice de ilustraciones

Capítulo 1: Introducción

El primer capítulo incluye una visión general del contenido del trabajo, las principales motivaciones que han llevado a su realización y una descripción de los objetivos generales.

1.1 Visión general

Es evidente que la tecnología es uno de los motores principales de la sociedad actual. Hablar de tecnología es hablar de invención, técnica, innovación, eficiencia y otros muchos términos que, en definitiva, describen el proceso de aprovechar recursos existentes para transformarlos en otros nuevos.

La sociedad del mundo actual es la sociedad de la información, un ecosistema que se caracteriza por el uso de las tecnologías de la información y la comunicación (TIC) que permiten gestionar la gran cantidad de datos generada día a día. Los datos son uno de los pilares fundamentales de las actividades socioculturales y económicas actuales y, de hecho, son considerados en muchos casos "el petróleo del siglo XXI".

Para explotar el nuevo petróleo, han surgido novedosas e innovadoras tecnologías que tienen como objetivo aprovechar al máximo los recursos de información mejorando así las diferentes actividades y procesos actuales. Tecnologías como la inteligencia artificial (AI), Big Data, el Internet de las Cosas (IoT) o la computación cuántica son los motores principales de la nueva revolución tecnológica después de la llegada de Internet.

En este caldo de cultivo surge también la tecnología Blockchain. Allá por el año 2008, un grupo de personas bajo el pseudónimo de Satoshi Nakamoto publican un artículo en el que describen Bitcoin, la primera criptomoneda. Bitcoin es una red de pagos "Peer to Peer" (P2P) en el que las transacciones no dependen de la supervisión de ningún intermediario y cuya base tecnológica es Blockchain. Bitcoin logra entonces ser la primera criptomoneda con aceptación y difusión global, iniciando una etapa de revolución no solo económica sino en también en muchos otros ámbitos de la sociedad actual.

El fin del presente proyecto es el estudio en profundidad del ecosistema Blockchain para conocer su verdadero potencial en soluciones productivas. Además, se busca aplicar este tipo de tecnología para la construcción de un sistema descentralizado y automático en el campo relacionado con las redes de ordenadores.

En concreto, se propone una prueba de concepto que incluye la implementación de un prototipo para gestionar el acceso, consumo y facturación de Internet como servicio ofrecido a través de un punto de acceso inalámbrico. Para lograr el objetivo se combinarán las capacidades que ofrece Blockchain con las de los dispositivos del Internet de las Cosas.

1.2 Motivación

Tras graduarme como ingeniero informático me surgió la inquietud por seguir avanzando en mi carrera formativa y, por lo tanto, tomé la decisión de inscribirme en el máster que completaba mi formación de grado. Esta decisión fue en gran parte tomada por mi interés hacia las nuevas tecnologías. A lo largo del máster pudimos documentarnos y llevar a cabo algunos proyectos de tecnologías tan novedosas como lo son el aprendizaje automático, la robótica, la realidad virtual o el Big Data.

Fue en este periodo cuando comencé a interesarme por Blockchain, una tecnología disruptiva que surgió inicialmente dando soporte a un sistema de pago alternativo al sistema bancario tradicional. Desde que conocí de su existencia, me llamaron la atención algunos de sus fundamentos como la descentralización, la seguridad y la transparencia porque creo que son en gran parte, principios que la sociedad actual ha perdido.

La palabra Blockchain estaba entonces en boca de todo el mundo y, de hecho, la empresa consultora y de investigación Gartner, ya en 2016 incorporaba Blockchain en su "Hype Cycle for Emerging Technologies", lo que significa que la percibía como una de las tecnologías disruptivas en el periodo de los próximos 5 a 10 años con mayor expectativa [1].

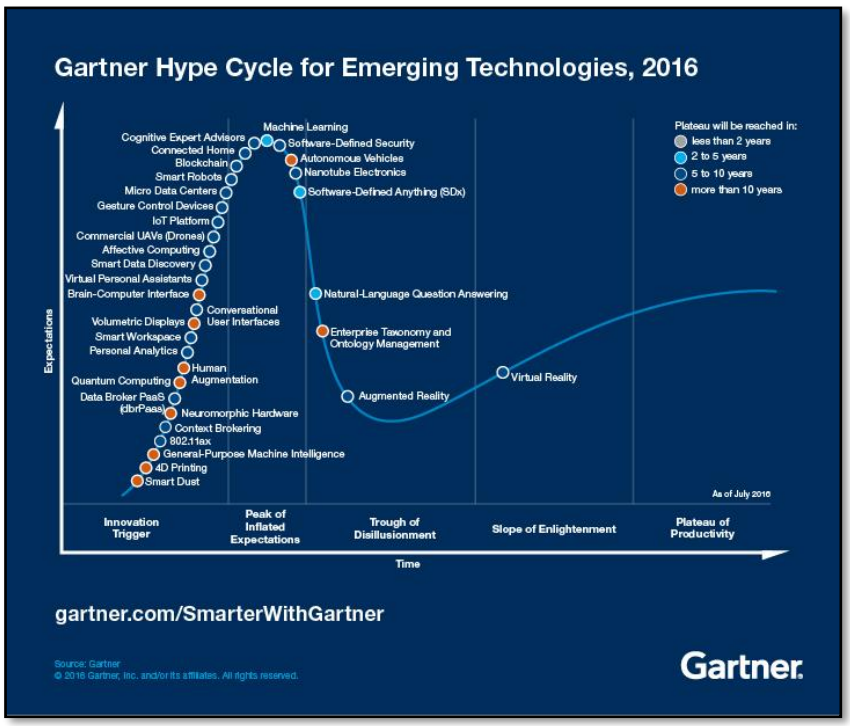


Ilustración 1. Ciclo del Hype de Gartner en 2016

Fue por todos estos motivos por los que empecé a formarme en esta tecnología de manera autodidacta. La realización de un proyecto como el presente conlleva documentarse de todo el ecosistema Blockchain y a profundizar en conocimientos técnicos que desde mi punto de vista son completamente valiosos. Además, creo que la mejor forma de aprender es materializar o llevar a la práctica un caso de uso real, todo esto con el valor añadido de que, perfectamente, podría ser productivo.

1.3 Objetivos

El objetivo de este proyecto es el estudio de la tecnología Blockchain para verificar si su aplicación en soluciones de entornos productivos aporta un valor añadido. Una vez realizado el estudio, se procederá a implementar una prueba de concepto a través de un prototipo funcional que valide el objetivo principal.

En concreto, la prueba de concepto pretende introducir de la criptoeconomía en la gestión del acceso y monitorización en redes de ordenadores a través de la representación digital de activos que permite la tecnología Blockchain. En profundidad, se desarrollará un modelo de acceso a red sobre la plataforma de Ethereum, que permita gestionar la misma en términos de prestación de servicio, consumo y facturación. El sistema deberá gestionarse de manera automática, descentralizada, segura y sin intermediarios a través de la tokenización del ancho de banda ofrecido por un punto de acceso a Internet. A través de la solución propuesta se pretende que cualquier empresa con la infraestructura necesaria pudiera ser prestadora del servicio y que cualquier cliente tenga la opción de comprar tokens para consumir el servicio. La tecnología Blockchain aporta las características ideales de seguridad, trazabilidad y transparencia que dotan de la confianza necesaria a los sistemas de pago.

Para lograr el objetivo, es necesario seguir un proceso que en sus primeras fases será de investigación para continuar con el diseño de la solución y acabar con su implementación. Por lo tanto, para la realización del proyecto se plantean los siguientes objetivos:

- Estudio del ecosistema Blockchain.
- Estudio de la plataforma de Ethereum.
- Análisis de la prueba de concepto para entender cómo introducir la criptoeconomía en el modelo de acceso a red planteado.
- Diseño de un prototipo que valide el valor real que aporta la tecnología.
- Implementación del prototipo a través de las tecnologías seleccionadas.
- Estudio de las posibles mejoras o líneas futuras del sistema desarrollado.

1.4 Estructura del documento

De aquí en adelante, el documento se estructura por capítulos de la siguiente manera.

Los capítulos 2 y 3 corresponden a un profundo estudio de la tecnología Blockchain y en concreto de la plataforma de Ethereum. El capítulo 4 está dedicado al análisis del estudio para definir los objetivos de la prueba de concepto que incluye esta tecnología. En el quinto capítulo se describen todos los módulos que formarán en conjunto el sistema, se eligen las tecnologías más apropiadas para su desarrollo y se define su arquitectura. El capítulo 6 recorre la preparación del entorno de trabajo y el desarrollo de los diferentes módulos que forman el prototipo. En el capítulo 7 se muestran los resultados de la implementación que validan el correcto funcionamiento de la misma. En el capítulo 8 se muestra la planificación del proyecto (primero se expone la planificación que se planteó al inicio y después se muestra el tiempo real dedicado al proyecto). Finalmente, en el capítulo 9 se detallan las conclusiones obtenidas y se proponen algunas mejoras que enriquecerían el sistema en siguientes iteraciones.

En su última parte, el documento finaliza con dos anexos en los que se detallan procesos técnicos necesarios para el desarrollo de la solución.

Capítulo 2: Estado de la cuestión

El segundo capítulo comienza con una pequeña introducción al ecosistema Blockchain que da paso a un análisis completo desde los fundamentos hasta las aplicaciones más novedosas pasando por las ventajas, limitaciones y principales plataformas.

2.1 Introducción a Blockchain

Blockchain es una base de datos distribuida en una red de nodos en la que la información se almacena en forma de bloques. En esta base de datos o "ledger" distribuido, quedan registradas todas las transacciones que realizan los participantes de manera inmutable una vez que se alcanza el consenso en toda la red mediante ciertos protocolos. El libro es mantenido por todos los participantes sin la necesidad de gestión de terceros, formando de esta manera un sistema distribuido y descentralizado.

Blockchain tiene una base criptográfica muy fuerte, los bloques de información están relacionados criptográficamente unos con otros formando una cadena de bloques en orden cronológico. Cada uno de los bloques referencia al bloque anterior mediante un hash criptográfico y gracias a las propiedades de éste, se puede asegurar la inmutabilidad de la cadena.

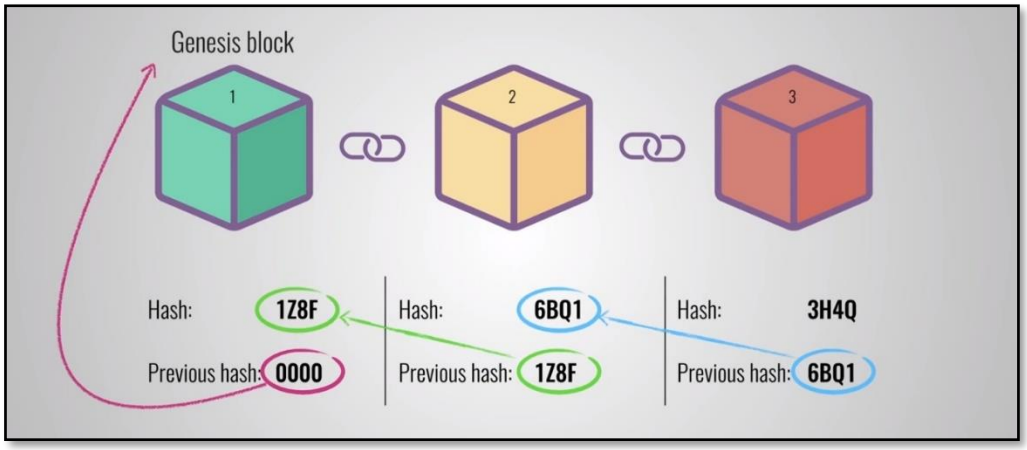


Ilustración 2. Estructura de la cadena de bloques

La modificación de la información de un bloque hará que el hash sea diferente y, por lo tanto, a partir de ese momento la cadena quedará invalidada. Los bloques almacenan transacciones y en función del tipo de Blockchain, la generación de bloques se realizará cada cierto periodo de tiempo o cada cierto número de transacciones. En el caso de Bitcoin se genera un bloque cada 10 minutos.

2.2 Sistemas distribuidos y descentralizados

En muchos casos, estos dos términos son confusos para la mayoría de la gente, sin embargo, tienen dos significados completamente diferentes que tienen que ver con la localización y el control.

Un sistema distribuido es aquel que se distribuye entre una gran cantidad de máquinas o nodos normalmente en diferentes ubicaciones. Esto quiere decir que para lograr el objetivo del sistema es necesaria la participación coordinada de todos los

subsistemas por los que está compuesto. Las principales ventajas de los sistemas distribuidos son:

- **Redundancia:** La información se encuentra replicada en los N nodos que formen parte del sistema. En el caso de que una máquina perdiese su información, podría obtener fácilmente una copia de cualquier otro participante.
- **Resistencia a fallos:** Aunque una parte o subsistema falle, el sistema puede seguir funcionando porque existen otros participantes que pueden realizar la misma función.
- **Escalabilidad:** En el caso de que se desee más capacidad del sistema, solo es necesario añadir más equipos.
- **Bajo coste:** La carga de trabajo se puede dividir eficientemente entre las partes del sistema utilizando máquinas mucho más baratas que si se tratase de un único y potente servidor.

Por el contrario, un sistema descentralizado es aquel en el que la propiedad del mismo no pertenece en su totalidad a una entidad central, es decir, debe existir una separación de poderes que no permita que un participante pueda realizar actualizaciones de forma unilateral sin el apoyo de la mayoría. Los principios de los sistemas distribuidos se pueden resumir como a continuación:

- **Transparencia:** El hecho de que todos los participantes tienen una copia de la misma información hace que sea un sistema transparente y de confianza.
- **Diversidad:** Se generan ecosistemas de diferente naturaleza que aportan funcionalidades complementarias y necesarias para el funcionamiento del sistema en su totalidad.
- **Falta de jerarquía:** Todos los participantes se encuentran en el mismo nivel jerárquico, evitando así concentraciones de poder y recursos.
- **Acceso libre:** Cualquiera con el mínimo de recursos necesario puede entrar a formar parte del sistema.

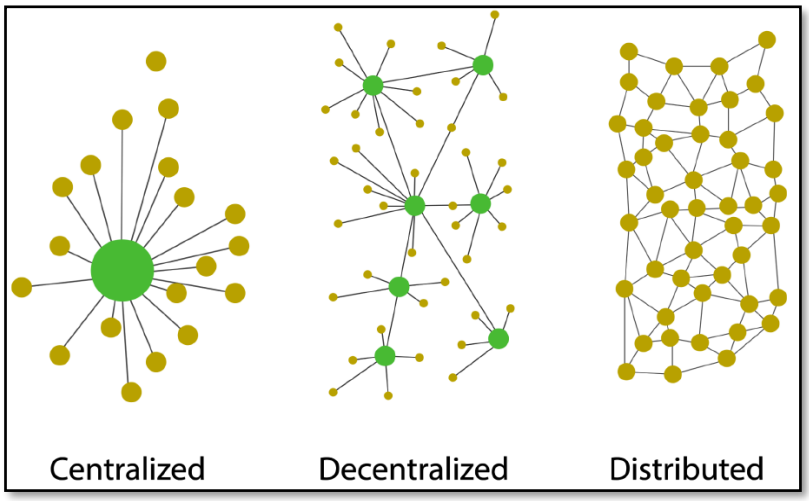


Ilustración 3. Sistemas descentralizados y distribuidos

2.3 Tipos de Blockchain

Como ya se ha expuesto anteriormente, la red Blockchain es una red descentralizada y distribuida entre diferentes nodos o participantes. Este tipo de redes se denominan *"Peer to Peer"* (P2P), ya que la comunicación es directa entre los participantes de la misma. En el apartado anterior se expusieron las ventajas de los sistemas distribuidos y descentralizados frente a los tradicionales en los que el control total de la red pertenece a una entidad central. Aun así y aunque compartan características básicas, existen diferentes tipos de redes Blockchain:

- **Redes públicas:** En este tipo de red, cualquiera puede entrar a formar parte de ella y convertirse en un nodo que envía, lee y verifica transacciones. Se trata de un *ledger* completamente abierto en el que la información puede ser accesible por cualquiera. Son las primeras redes Blockchain que aparecieron, como por ejemplo Bitcoin (BTC) y Ethereum (ETH), las más famosas, pero también existen otras como por ejemplo Litecoin (LTC), Monero (XMR), IOTA (MIOTA) o la incipiente Blockchain de Facebook en la que se basará su propia criptomoneda Libra. Este tipo de red también se denomina no permisionada.
- **Redes privadas:** En oposición a las Blockchains públicas, los permisos para participar en una red están restringidos a una entidad u organización. La escritura de datos y la verificación de las transacciones tampoco es abierta, si no que está destinada solo a ciertos nodos. Generalmente, este tipo de redes son las utilizadas en el entorno empresarial, en el que las organizaciones no desean que su información sea pública y accesible por todos, como es obvio. De las más famosas son Hyperledger Fabric, Corda, Quorum o Ripple.

- **Redes permisionadas:** En las redes permisionadas el poder de escritura y de validación de transacciones está destinado a un conjunto de nodos en función de la política que se defina. En el caso de las lecturas de información pueden ser públicas o permisionadas.

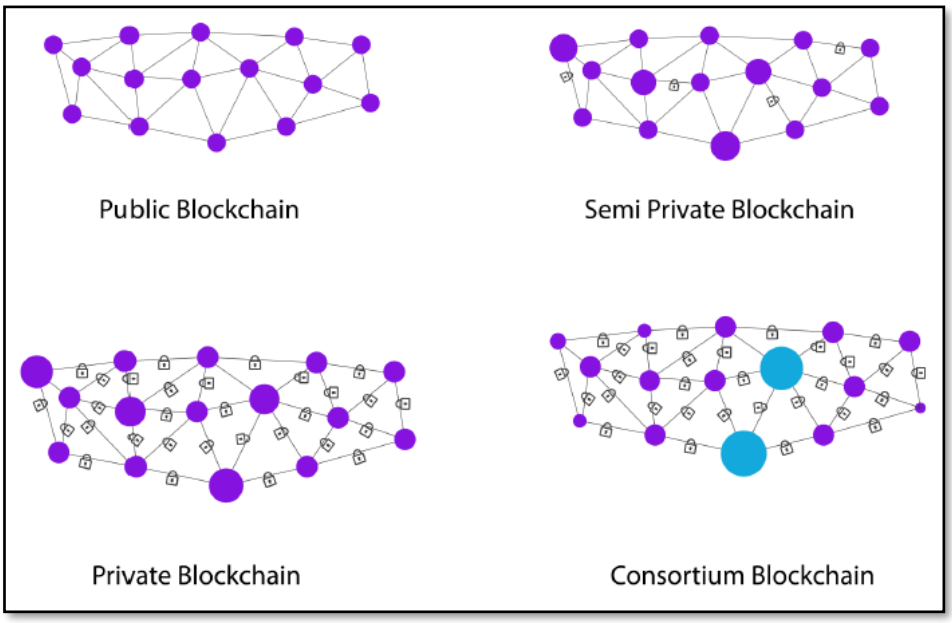


Ilustración 4. Tipos de redes Blockchain

2.4 Consenso

Probablemente, llegados a este punto hayan surgido preguntas como ¿cómo se verifica una transacción?, ¿cómo se logra que toda la red llegue a la misma conclusión?, ¿quién es el encargado de validar un bloque para que entre a formar parte de la cadena? Para dar respuesta a todas estas preguntas se expone a continuación un problema clásico de redes de computadoras: El Problema de los Generales Bizantinos.

Imaginemos un escenario en el que diferentes generales Bizantinos al frente de sus tropas van a atacar una ciudad desde diferentes posiciones estratégicamente elegidas. Todos los generales deben ponerse de acuerdo para atacar la ciudad al mismo tiempo y para concentrar sus fuerzas y lograr el objetivo. El problema es que alguno de los generales puede ser un traidor y emitir un mensaje corrupto para provocar inestabilidad en el ataque. Para resolver la situación, se deben aplicar una serie de algoritmos para alcanzar un consenso entre todas las partes.

A través del consenso se busca alcanzar un acuerdo común entre todos los participantes de la red. Cada Blockchain tiene su propio mecanismo de consenso que permite decidir qué bloques son añadidos a la cadena y cuáles no. A continuación, se definen los mecanismos de consenso más populares en la actualidad:

- **Proof of Work (PoW):** Fue el primero en ser empleado por una Blockchain (Bitcoin) y es el mecanismo de consenso más utilizado hoy en día. Antes de la incorporación de un bloque a la cadena, todos los participantes de la red encargados de validar transacciones se ponen a realizar un acertijo criptográfico con cierto grado de dificultad. Este acertijo requiere de gran capacidad computacional para ser resuelto. Concretamente, deben calcular un valor tal que, combinado con el resto de información del bloque, genere un hash concreto. De esta manera, el primero que logre el resultado es el encargado de añadir el bloque a la cadena después de que la mayoría de los participantes comprueben que la operación es correcta. Por encontrar la solución al problema se ofrece una recompensa al participante. Si la resolución del problema fuese demasiado fácil, la red quedaría expuesta a vulnerabilidades, pero con un coste computacional alto, se evitan comportamientos indeseados, puesto que el gasto de energía que hay que emplear para solucionar el problema es elevado. El proceso de añadir un nuevo bloque a la cadena es lo que se conoce como como minería, y los encargados de hacerlo son los mineros. Este algoritmo de consenso mantiene la integridad de la red siempre y cuando no haya ninguna entidad que controle la mayoría de la red, problema conocido como el ataque del 51% [3].
- **Proof of Stake (PoS):** Este mecanismo surgió para intentar resolver el problema de consumo de energía que conlleva ejecutar PoW. En PoS, cada nodo que desee añadir un nuevo bloque a la cadena hace una apuesta o *stake* de sus fondos para validar el mismo. Cuanto mayor sea la apuesta mayor será la posibilidad de que el participante sea el encargado de añadir el nuevo bloque. El proceso de selección es semi-aleatorio entre los participantes con mayores fondos, lo que plantea el problema de que siempre generen los nuevos bloques los nodos con más fondos, tendiendo así a la centralización.
- **Proof of Stake Anonymous (PoSA):** Se trata de una variación del PoS en la que el origen y el destino de las transacciones quedan ocultos por otros nodos que también reciben recompensa por ayudar en la generación de nuevos bloques.
- **Delegated Proof of Stake (DPoS):** Utiliza una combinación de la cantidad de fondos que tiene el usuario con un sistema de voto entre los participantes. De esta manera, cualquiera que tenga fondos, realiza una votación hacia otros participantes, con el objetivo de delegar en ellos para que generen y añadan el nuevo bloque a la cadena.

- **Proof of Importance (POI):** En este mecanismo cada participante recibe una puntuación de importancia en función de la cantidad de fondos que otorgue. La selección se realiza en proporción a la puntuación que mide la contribución del participante a la red en conjunto con otras variables como, por ejemplo, el número de transacciones en los últimos treinta días [4].

2.5 Limitaciones de la tecnología

En los apartados anteriores se han expuesto las principales ventajas de la tecnología Blockchain pública al ser un sistema distribuido y descentralizado. Entre las que se ha señalado que no es necesaria la participación de un tercero que tenga el papel de intermediario, el control de la red pertenece a los participantes, la inmutabilidad de la información se garantiza gracias a la criptografía, el mecanismo de consenso vela por la confianza de la red y la transparencia y trazabilidad de la información es pública. Sin embargo, no es oro todo lo que reluce, y esta tecnología también tiene ciertas limitaciones que son necesarias comentar.

El principal inconveniente de las tecnologías públicas es en términos de rendimiento y confianza entre las partes involucradas en un proceso de negocio. Los mecanismos de consenso implican tareas computacionales que requieren de cierto tiempo para ser resueltas, por ejemplo, en Bitcoin se genera un nuevo bloque cada diez minutos que incluye aproximadamente 2000 transacciones, lo que penaliza de forma considerable el rendimiento. Por otro lado, en los procesos de negocio en los que se encuentran varias organizaciones involucradas, es posible que no confíen entre ellas por el mero hecho de tener diferentes intereses.

A continuación, se exponen las principales limitaciones de la tecnología:

- **Alto consumo:** Como ya se ha expuesto anteriormente existen mecanismos de validación o consenso como “Proof of Work” que necesitan un enorme poder computacional, repercutiendo así en el gasto de energía correspondiente y penalizando el rendimiento de la red.
- **Bajo rendimiento y mala escalabilidad:** La red Bitcoin es capaz de procesar una media de cuatro transacciones por segundo y la red Ethereum quince, sin embargo, estos números están muy lejos de los alcanzados por las entidades financieras actuales. Queda claro que este aspecto supone una limitación en la implementación de servicios productivos. Por otro lado, la cadena de bloques crece continuamente, algo que supondrá problemas de almacenamiento en un futuro. El tamaño de la cadena de datos de Bitcoin actualmente alcanza los 236MB [5].

- **Información accesible públicamente:** Los datos de la cadena son accesibles por cualquier participante que forme parte de la red y es por ello que en ciertos casos en los que se manejen datos sensibles o en el ámbito empresarial pueda suponer un grave problema.
- **Descentralización y disponibilidad de la red:** La descentralización también puede suponer un problema en el sentido de que los datos se encuentran almacenados permanentemente en diferentes ubicaciones con diferentes regulaciones legales. La disponibilidad de la red no está garantizada, y de hecho, se han dado casos en los que la red se ha congestionado quedando así fuera de servicio.
- **Uso implícito de criptomonedas:** Para hacer uso de las plataformas Blockchain es necesario hacer uso de sus criptomonedas para sufragar las tasas asociadas a la generación de transacciones. Además, el valor de las criptomonedas es muy volátil haciendo que las tasas también fluctúen y sea muy complicado estimar costes.

2.6 Aplicaciones de Blockchain

Las ventajas de Blockchain han llamado la atención de miles de empresas que quieren incorporar esta tecnología en su negocio porque han identificado el valor que les aporta. Los sectores o ámbitos en los que se aplica Blockchain son múltiples, pero a continuación se muestran aquellos en los que más se reconocen sus ventajas:

- **Ecosistema financiero:** Por supuesto, es el sector en el que más encaja debido a su naturaleza y demás, existen en la actualidad miles de instituciones financieras y de seguros que hacen uso de Blockchain para evitar situaciones fraudulentas o aumentar la confianza que proyectan al cliente final, por ejemplo.
- **Cadena de suministro:** La trazabilidad de todo el proceso en la cadena de suministro de bienes de cualquier tipo ahorra muchos costes a las empresas y además aporta un valor diferencial para los consumidores finales ya que aumenta la transparencia, el tracking y la prevención de falsificaciones.
- **Internet de las cosas:** A priori parecen perfectos compañeros puesto que cada dispositivo podría identificarse y enviar datos punto a punto de manera segura, pero también es necesario conocer que existen ciertas limitaciones debido a la gran cantidad de información que generan este tipo de dispositivos.
- **Certificación de documentos:** Tiene mucho sentido hacer uso de Blockchain en los ámbitos en los que se deben gestionar documentación, contratos o

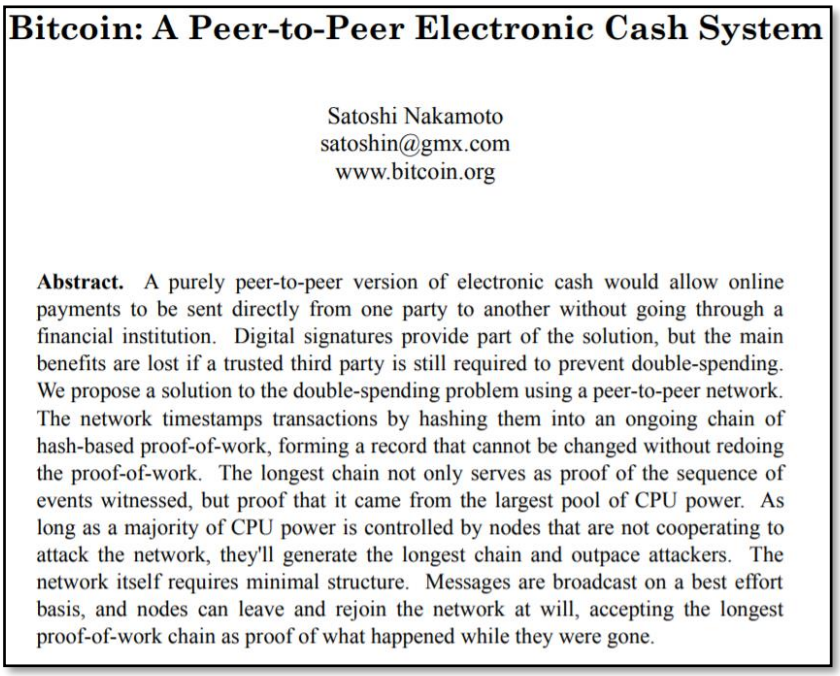
certificados. De hecho, algunas universidades como la Universidad Carlos III de Madrid (UC3M) o la Internacional de La Rioja (UNIR) ya se han sumado a la tendencia en la entrega de diplomas y títulos.

- **Sistemas de votación:** A través de la tecnología se elimina la posibilidad de amaño de votaciones. Además, el usuario quedaría completamente anonimizado y podría votar digitalmente, algo que seguro aumentaría la tasa de participación.
- **Sanidad:** En este sector aporta un valor extra muy tangible, por ejemplo, se puede acceder al historial médico de un paciente de manera descentralizada y segura, así cualquier centro dispondría de la información inmediatamente. Además, es posible tener una trazabilidad completa de la venta de medicamentos y evitar estafas en las pólizas.
- **Media:** A través de Blockchain se puede garantizar la propiedad intelectual de cualquier activo y los derechos de autor.
- **Energía:** El sector energético actualmente ya se está beneficiando de aspectos como eliminar intermediarios entre productor y consumidor, garantizar las fuentes limpias de la energía renovable o el trading de energía entre compañías.

2.7 Principales plataformas Blockchain

Como se mencionó al principio del documento Bitcoin es considerado el primer sistema que adapta Blockchain como base tecnológica y que tuvo una aceptación global. A partir de su aparición han surgido muchas otras criptomonedas y redes Blockchain siguiendo la misma filosofía que Bitcoin e intentando aportar nuevas características. A continuación, se resumen las principales plataformas Blockchain más populares y utilizadas hoy en día comenzando por Bitcoin.

- **Bitcoin:** En 2008 un individuo o grupo de individuos bajo el pseudónimo de Satoshi Nakamoto publican “Bitcoin: A Peer-to-Peer Electronic Cash System”, un artículo en el que se introducía un sistema de pagos electrónicos en el que las transacciones no necesitan de ningún tipo de intermediario financiero gracias al uso de técnicas criptográficas [6].



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Ilustración 5. Abstract del artículo de Bitcoin

Hoy en día es la red Blockchain más utilizada y con mayor capitalización de mercado (181.849.815.885 dólares en el momento en el que se está escribiendo el presente documento con un valor de 10.161,23$ por cada Bitcoin). Gran cantidad de plataformas de diferentes ámbitos aceptan Bitcoin como forma de pago entre las que se encuentran por ejemplo Microsoft USA, Wikipedia, Paypal, Geenpeace, Gear Best o WikiLeaks [7].

- **Ethereum:** En 2013, el investigador y desarrollador Vitalik Buterin, publica un artículo en el que conceptualiza Ethereum, una plataforma Blockchain que surge con el objetivo de que no solo los sistemas de pago se beneficien de esta tecnología, sino ampliar su aplicación también en otros sectores. En 2015 y tras conseguir la financiación necesaria se mina el primer bloque de lo que hoy se conoce como la red principal o “mainnet” de Ethereum. La característica principal de esta plataforma es que permite ejecutar lógica de negocio en la red a través de lo que se conoce como contratos inteligentes o “smart contracts”. Ether es su criptomoneda y actualmente es la plataforma con la segunda mayor capitalización de mercado (20.524.336.398$ en el momento en el que se está escribiendo el presente documento con un valor de 190,99$ por cada Ether).
- **Ripple:** Fundada en 2012 por Chris Larsen y Jed McCaleb, es una Blockchain privada que ofrece un sistema de liquidación bruta en tiempo real caracterizada por la velocidad de las transacciones, la seguridad y el bajo

coste. Denominada “la moneda de los bancos” no pretende eliminar el sistema tradicional, sino ser un mecanismo dentro del sistema bancario que aporte pagos rápidos y fiables. También tiene su propia criptomoneda, XRP, siendo la tercera con mayor capitalización de mercado (11.714.803.683$ en el momento en el que se está escribiendo el presente documento con un valor de 0,273012$ por cada XRP) [8].

- **Hyperledger:** Quizá sea el proyecto colaborativo más utilizado en el entorno empresarial. Fundado en 2015 por la Linux Foundation, busca combinar las ventajas de la tecnología Blockchain en el ámbito privado para acelerar los procesos de negocio siempre de manera fiable y transparente. Entre las empresas que participan en el proyecto se encuentran algunas como IBM, J.P Morgan, Cisco o Accenture. Existen diferentes proyectos dentro de Hyperledger, siendo Hyperledger Fabric el más utilizado en proyectos de ámbito empresarial [9].
- **Corda:** Se trata de una plataforma distribuida bajo el consorcio R3 de naturaleza privada que se centra principalmente en el ecosistema financiero y que pretende ofrecer capacidades de desarrollo en dicha infraestructura implementando aplicaciones distribuidas y restricciones en el acceso a la información de las transacciones generadas. En 2016 se produjo la liberación de su código que es mantenido por miles de desarrolladores. Actualmente es una de las plataformas más utilizadas en el entorno empresarial [10].

2.8 Tokenización de activos

Una de las aplicaciones estrella de la tecnología Blockchain es la tokenización de activos. Esta capacidad permite representar digitalmente cualquier activo del mundo real, ya sea un bien físico como la propiedad de un patrimonio o un activo intangible como el esfuerzo dedicado a una tarea por parte de una persona. Así, el valor digital del activo se representa a través de tokens transferibles que pueden ser administrados (creados, emitidos, transferidos, bloqueados, etc) en una red Blockchain manteniendo en todo momento el control absoluto de la propiedad. Los tokens representan una enésima parte del valor del activo que debe ser evaluado y auditado [11].

Imaginemos, por ejemplo, que se desea tokenizar la cantidad de energía renovable generada por un productor para que, de esta manera, sea posible incentivar la generación de este tipo de energía e intercambiarla fácilmente para su consumo. En primer lugar, sería necesario realizar una valoración del activo para decidir cuántos kilovatios de energía se corresponden con un token y una vez que se ha tokenizado ya se encuentra disponible para todo aquel que desee invertir y es posible gestionar su

valor a través de la red Blockchain. Así, por ejemplo, un usuario productor con placas en su vivienda podría transferir los tokens de energía renovable que no consumiera de manera sencilla y fiable, obteniendo el correspondiente beneficio monetario.

La criptoeconomía introducida a través de la tokenización ofrece un ecosistema financiero más justo y eficiente que reduce de manera considerable las fricciones existentes en la compraventa de valores o títulos. A través del ejemplo anterior es posible darse cuenta de las múltiples ventajas que ofrece la tokenización de activos, pero a continuación se detallan las principales:

- **Liquidez:** Los tokens son comercializados en mercados secundarios elegidos por el emisor del mismo y que son muy cotizados por miles de inversores. De esta manera los compradores tienen libertad a la hora de elegir en qué mercado invertir y los vendedores obtienen liquidez instantánea.
- **Rapidez y ahorro de costes en las transacciones:** Puesto que gestión de los tokens se realizan completamente en la red a través de los llamados *smart contracts* en los que se define toda la lógica, los procesos de intercambio se automatizan al máximo aumentando la velocidad de los mismos. Esta automatización repercute en una reducción de los costes administrativos ya que muy pocos intermediarios son necesarios en el proceso (una entidad de cambio en la que cambiar el dinero real por el token o la criptomoneda, por ejemplo) y que las tasas asociadas son muy asumibles.
- **Transparencia:** Al ser Blockchain la base tecnológica, transmite directamente la transparencia que le caracteriza. Evita el fraude almacenando en el propio token las responsabilidades y derechos del comprador a través del registro inmutable de la propiedad que puede ser consultado en todo momento por todos los participantes.
- **Accesibilidad:** Para realizar una inversión no se necesitan grandes cantidades de dinero puesto que el valor de un token representa un pequeño porcentaje del valor del activo y que la automatización del proceso permite una reducción del coste de inversión. La participación queda abierta a un abanico de público mucho más amplio.

Actualmente la tokenización de activos se está utilizando en gran cantidad de casos de uso reales. CryptoKitties es una plataforma en la que se pueden crear, transferir y en definitiva, coleccionar gatos digitales que son únicos y que pertenecen 100% a su comprador a través de un contrato inteligente en Ethereum [12]. Puede parecer un juego de niños, pero tras su lanzamiento en 2017 alcanzó tal popularidad que logró congestionar la red entera por la gran cantidad de transacciones que se generaban. El

valor real se encontraba detrás de estas queridas mascotas, era la primera vez que se permitía el intercambio de activos digitales únicos entre personas.

En el mundo del arte también se ha hecho uso de la tokenización, ya que permite dividir el valor de una obra original y única en diferentes participaciones de manera que ofrece la posibilidad de revender las mismas en el momento que la obra se vea revalorizada en un futuro.

Tutellus es una plataforma de formación en Internet en la que se ofrecen miles de cursos. Puedes participar tanto como estudiante, profesor o como promotor de cursos. Esta plataforma tiene su propio token, TUT, mediante el que puedes ser recompensado si generas contenido u ofreces una docencia, por ejemplo. También puedes cobrar por aprender, puesto que cuanto más estudias, más tokens y visibilidad ganas dentro del ecosistema en el que las empresas acuden en busca de talento.

Como se puede observar a través de los casos de uso reales que ya emplean la tokenización de activos, las posibilidades que ofrece esta capacidad son infinitas ya que permite comercializar activos digitales y establecer nuevos mercados.

Capítulo 3: Ethereum

Ethereum es la plataforma principal utilizada en el proyecto, en este tercer capítulo se realiza una explicación de sus fundamentos y de los conceptos necesarios para poder implementar una solución funcional.

3.1 Introducción a Ethereum

Como se ha expuesto en el capítulo anterior, Ethereum es una plataforma basada en Blockchain con una capacidad especial que permite desarrollar programas denominados contratos inteligentes o *smart contracts*. Estos contratos contienen lógica de aplicación y hacen posible la implementación de lo que se denominan aplicaciones distribuidas o *DApps* como soluciones verticales.

Ethereum dispone de un lenguaje de programación denominado Solidity. Se trata de un lenguaje "Turing completo", esto es, que tiene capacidad para resolver cualquier problema computacionalmente complejo. Los *smart contracts* se escriben en Solidity y se despliegan en la red para poder interactuar con ellos [13].

La red de Ethereum fue liberada el 30 de Julio de 2015 (en ese momento, se minó el primer bloque de la red principal o *mainnet*) a través de la Ethereum Foundation y dispone de su propia criptomoneda, el Ether (ETH).

Ethereum hereda el funcionamiento de las Blockchains públicas, se trata de un conjunto de nodos operando en red en función de las reglas específicas para alcanzar el consenso para crear, añadir y validar nuevos bloques de información. En cada nodo, reside una copia de la cadena de bloques en la que se tiene un control exacto de las transacciones que se han generado desde el primer bloque o bloque génesis. La cadena es mantenida por los participantes mineros que son los encargados de incorporar nuevas transacciones. El algoritmo de consenso actual de Ethereum es *Proof of Work* pero, con el objetivo de aumentar la escalabilidad del sistema, se baraja la posibilidad de cambiar en breve a *Proof of Stake* con el denominado protocolo Casper.

3.2 Las cuentas de Ethereum

Al igual que existen cuentas bancarias en el sistema financiero, en Ethereum sucede lo mismo. Las cuentas se identifican por una dirección pública que es el identificador para poder interactuar con ellas. Toda la información de las transacciones que ha realizado una cuenta se encuentra almacenada dentro de la misma, pero como se expone a continuación, también dispone de otros datos.

Existen dos tipos de cuentas: Cuentas de propiedad externa o e*xternally owned accounts* (EOAs) que se aseguran y gestionan a través de claves privadas, y cuentas de contrato que se controlan a través del código del propio contrato. Cada cuenta de usuario dispone de un balance y puede realizar transacciones hacia otras cuentas. Las cuentas de contrato disponen de un balance de la criptomoneda (Ether) y del código del contrato para, que cuando se reciba una interacción, pueda realizar las operaciones de lectura o escritura correspondientes en su espacio de memoria.

Los campos que contiene una cuenta son:

- **Nonce:** En el caso de que sea una cuenta del tipo EOA este parámetro representa el número de transacciones en las que ha participado como emisor. Si por el contrario, es del tipo contrato, representa el número de contratos creados por la cuenta.
- **Balance:** Como se había comentado, las cuentas tienen asociada una cantidad de Ether que se verá modificado cuando se realicen transacciones. En concreto el balance se representa en *Wei* que se corresponde con 10^{-18} Ether.
- **Storage root:** Para garantizar la validez de las transacciones, Ethereum utiliza una estructura de control criptográfico denominada *Merkle Patricia Tree*, una variación de *Merkle Tree* más compleja que la utilizada en Bitcoin y que está diseñado para mejorar las inserciones y borrados optimizando la memoria. En ella, se compone un hash a partir del hash de todas las transacciones, de forma que si alguna transacción es modificada el hash es completamente diferente. En este campo se almacena la raíz del *Merkle Patricia Tree* de transacciones.
- **Code hash:** Es el hash generado a partir del código del contrato.

Las cuentas EOA solo disponen de los dos primeros campos, mientras que las cuentas de contrato incluyen los cuatro.

3.3 Smart Contracts

Una de las primeras definiciones del concepto *smart contract* los describía como un conjunto de promesas especificadas digitalmente por todas las partes implicadas que se cumplen automáticamente gracias a la implementación de ciertas reglas o protocolos.

Es necesario dedicar tiempo y otros recursos en la revisión y mantenimiento de los contratos legales tradicionales, lo que implica directamente grandes costes asociados. En muchos casos no se cumplen las políticas definidas en estos contratos y son ejecutados de manera centralizada por la organización correspondiente. El principal problema es que estos contratos son gestionados por humanos susceptibles de cometer errores o de corromperse por ciertos incentivos. Sin embargo, un s*mart contract* se ejecuta de manera autónoma, descentralizada y sin que se pueda influir en su comportamiento ni modificar la información que almacena [14].

Como ya se ha adelantado, la principal ventaja de los s*mart contracts* es la simplicidad que ofrecen en la gestión y resolución de un acuerdo entre diferentes partes,

sobre todo, en aquellos escenarios en los que se transfieren activos que conllevan valor monetario. Además, en este tipo de transacciones la confianza cobra un papel primordial para todas las partes y, puesto que el código de los contratos inteligentes es público, se trata de una característica asegurada a través de Ethereum.

En la práctica un *smart contract* es un programa que se compone de variables, métodos y eventos. Un *smart contract*, igual que cualquier otro programa en ejecución, dispone de su propia memoria. Este espacio de memoria se denomina *storage* y es el lugar en el que se almacena toda la información de variables y que es posible consultar públicamente cuando el contrato se despliega y ejecuta en la red. Una vez que el contrato es desplegado, se identifica con un *address* que será necesario para poder interactuar con él.

La implementación del s*mart contract* dependerá por supuesto del caso de uso, pero existen algunos muy extendidos y famosos como pueden ser los relacionados con la tokenización: los contratos *Ethereum Request for Comments* o ERC. Este tipo de contratos son estándares propuestos por los mismos *developers* de Ethereum cuyo objetivo no es solo impulsar el uso de su plataforma, si no añadir interoperabilidad en el sistema ofreciendo capacidad a las aplicaciones que integran tokens [15].

- **ERC-20:** Ofrece las capacidades de crear, transferir y en definitiva operar tokens creando fácilmente valor transferible en la red. Este contrato implementa funcionalidades como consultar el balance de una cuenta o transferir tokens desde una cuenta hacia otra. Para crear un token de este tipo, simplemente es necesario especificar el nombre del token, su símbolo, los decimales que tendrá (un máximo de 18) y el balance total.
- **ERC-777:** Se trata de una versión avanzada del ERC-20 que añade funcionalidades como la capacidad de aceptación o rechazo de una cantidad de tokens por parte del destinatario o la eliminación de tokens en circulación. También permite que los denominados *operators* tengan poder para transferir tokens en nombre del propietario de la cuenta.
- **ERC-721:** Es el protocolo para crear tokens no fungibles. Este concepto se refiere a un activo que es único y no sustituible por otro, como por ejemplo, un Ferrari modelo especial como el Superamerica 45. A menudo se denominan tokens coleccionables, ya que se puede definir su valor en función de sus propiedades y tienen mucho sentido en entornos en los que se valora la originalidad o la singularidad como puede ser por ejemplo el mundo de las obras de arte. La plataforma CryptoKitties explicada más adelante, fue una de las primeras en implementar este estándar.

Por supuesto, el código de todos estos estándares es abierto y consultable en el repositorio del proyecto *OpenZeppelin* (https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token).

3.4 El Ether y el gas en Ethereum

El Ether, la criptomoneda nativa de Ethereum, es necesaria para pagar el despliegue de *smart contracts* en la red y la ejecución de transacciones. Uno de los conceptos más importantes en Ethereum es el gas, medida utilizada para dimensionar la carga computacional de ejecutar cualquier transacción en la red que consuma recursos. Por lo tanto, para ejecutar ciertas transacciones o crear cuentas y *smart contracts* en Ethereum es necesario disponer de gas.

Cada transacción tiene asociado un límite de gas (*gas limit*) y un precio de gas (*gas price*) expresado en unidades de Ether por unidad de gas, que define las tasas por ejecutarlas en la red [16]. Estas comisiones compensan a los mineros por el coste computacional de ejecutar la transacción y añadir un nuevo bloque. En el caso que se especificase un *gas price* muy bajo, la transacción tardaría mucho en ejecutarse puesto que a ningún minero le interesaría una recompensa tan baja.

Es posible que existan bucles infinitos en el código, de manera que la red podría quedar colapsada. A través del gas se evitan este tipo de situaciones, ya que el coste sería infinito para la cuenta asociada que ejecute el código. Existe una relación directa entre la complejidad de una transacción y su coste en gas, algo que limita en gran aspecto la opción de ataque. Las únicas funcionalidades que no implican un coste asociado son las de consulta o lectura de datos. A continuación, se muestra un ejemplo de lo que supondría la ejecución de una transacción:

- Una transacción que implica n instrucciones cuesta 17500 gas
- El precio del gas es de 0.0002 Ether por cada 100 unidades de gas
- El coste total de la transacción es de: (17500/100) x 0.0002 = 0,035 Ether

En este punto, puede surgir la duda de ¿por qué no pagar directamente en Ether? La razón es que el gas es una medida estable y sin embargo el valor del Ether es muy volátil y lo que hoy costase 0,005 Ether mañana podría costar 0,05, por ejemplo. Es por ello por lo que se utiliza el gas, cuyo valor se va ajustando en función de la volatilidad del Ether. El precio del gas se cuantifica en Gwei que son 10^{-9} Ether. A través de la web Eth Gas Station (https://ethgasstation.info/) es posible obtener una recomendación del *gas price* que hay que especificar para que una transacción se ejecute con cierta velocidad.

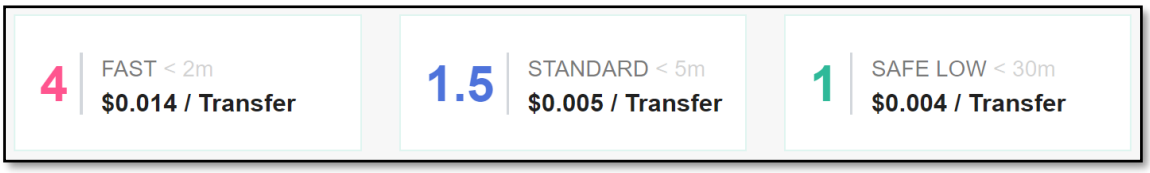


Ilustración 6. Precio del gas en Gwei para la ejecución de una transacción

3.5 La máquina virtual de Ethereum

Todos los participantes de la red hacen uso de una máquina virtual (EVM o *Ethereum Virtual Machine*) para poder ejecutar el código de los *smart contracts*. Cuando un contrato es compilado y desplegado en la red, se traduce su contenido a *bytecode,* que es el tipo de instrucciones que entiende la máquina virtual. Es una máquina de ejecución de pila que ofrece un entorno de ejecución independiente del sistema operativo en el que corra y, que permite implementar contratos inteligentes de manera totalmente desacoplada al entorno en el que se ejecutan.

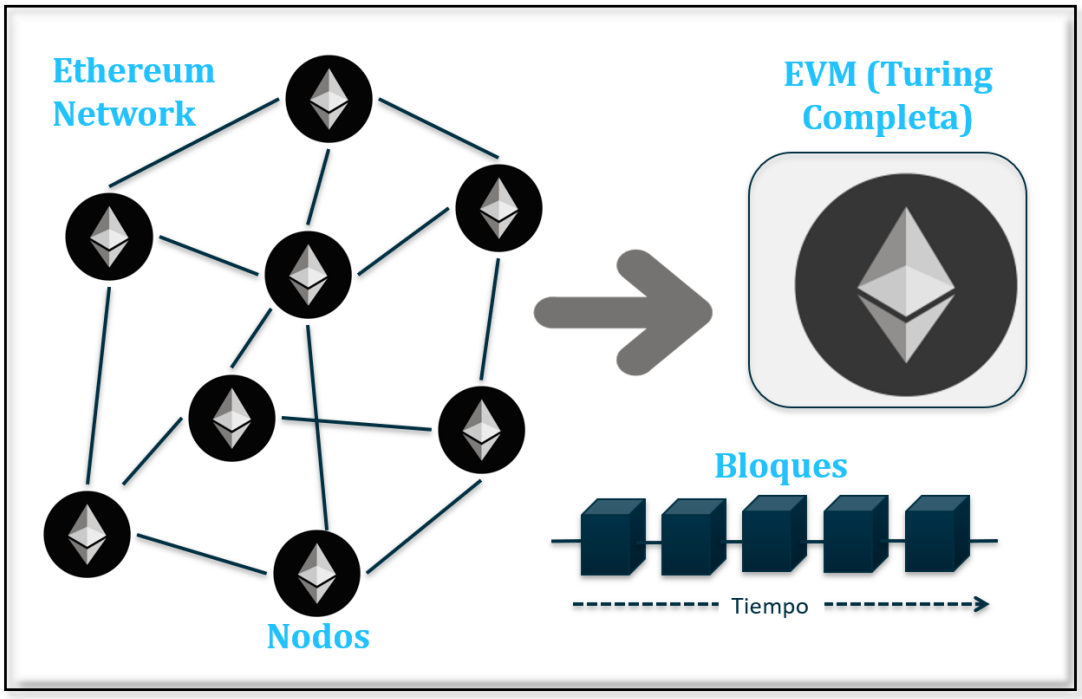


Ilustración 7. Ethereum virtual machine

El lenguaje comprendido por la EVM es un conjunto de instrucciones de bajo nivel como lo son por ejemplo las del lenguaje de ensamblador. Estas instrucciones incluyen operaciones aritméticas, de acceso a memoria, lógicas y de control, etc. Además, la EVM se encarga de gestionar toda la información de las cuentas en Ethereum y los parámetros de la red. A continuación, se muestra toda la información que requiere la EVM para poder proporcionar el entorno de ejecución:

- **Estado de las cuentas** detallado en el apartado anterior.
- **Estado del mundo:** Almacena la información de direcciones y estados de cuentas en un *Merkle Patricia Tree* para garantizar la integridad de la información.
- **Estado de almacenamiento:** Información acerca de la cuenta que está siendo gestionada por la EVM en tiempo real.
- **Información del tiempo de ejecución para tramitar las transacciones:** Esto incluye el precio de las tasas para ejecutar la transacción, el tamaño de esta y la dirección que la ejecuta.
- **Información correspondiente al bloque:** Se incluyen parámetros como el hash del último bloque, el *timestamp* o marca temporal del nuevo generado y su número.

3.6 Los bloques en Ethereum

Un grave problema en este tipo de redes es el conocido como doble gasto y se refiere a la posibilidad de que una moneda digital o criptomoneda se gaste dos veces creando un conflicto entre el registro de las transacciones y el balance disponible. Para solucionar este problema, las transacciones se agrupan en lo que se denomina bloque. Un bloque es un conjunto de transacciones en la red que se ejecutan y distribuyen entre todos los nodos de la red. En el caso de que dos transacciones se contradigan, la segunda que haya ocurrido será eliminada y no entrará a formar parte del bloque.

Los bloques son las partes que componen la cadena, ya que forman una secuencia lineal y temporal. En el caso de Ethereum se forma añade un nuevo bloque cada diecisiete segundos y los elementos principales que lo conforman son la referencia al bloque anterior, la cabecera de bloque y la raíz del *Merkle Patricia Tree* formado por las transacciones. Además, se dedican otros *Merkle Patricia Trees* para los *receipts* que registran el resultado de cada transacción y para los estados.

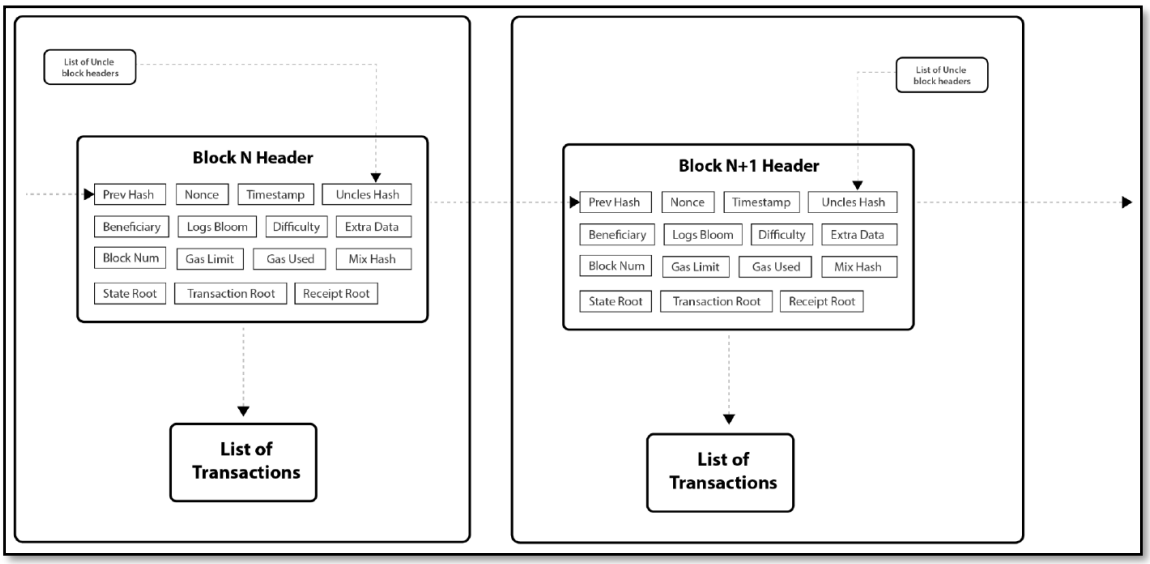


Ilustración 8. Estructura de los bloques en Ethereum

Como se puede observar en la imagen la cabecera contiene un conjunto de metadatos de la red bastante extenso. Quizás los más importantes sean:

- ***Parent hash:*** El hash correspondiente a la cabecera del bloque anterior.
- ***Beneficiary:*** La dirección a la que se envían todas las tasas correspondientes al minado de ese bloque.
- ***State root:*** El hash de la raíz del árbol de estados.
- ***Transaction root:*** El hash de la raíz del árbol de transacciones.
- ***Receipts root:*** El hash de la raíz del árbol de recibos.
- ***Number:*** El número de bloque.
- ***Gas limit:*** El límite de consumo de gas del bloque.
- ***Gas used:*** El gas consumido por todas las transacciones incluidas en el bloque.
- ***Timestamp:*** La marca temporal Unix.
- ***Nonce:*** Un hash que demuestra que la validación del bloque es correcta.

El tipo de hash que utiliza Ethereum es Keccak de 256 bits.

Después de que un bloque sea minado, es necesario que pase por ciertas comprobaciones para ser considerado válido y poder añadirse a la cadena. Los principales *checks* que se realizan son la existencia y validez del bloque anterior, y la validez de la marca temporal del bloque. Por supuesto, debe ser mayor que la del bloque padre y, además, ambos nodos deben separarse como mucho quince minutos. Si cualquiera de estas comprobaciones fallase, el bloque quedaría descartado.

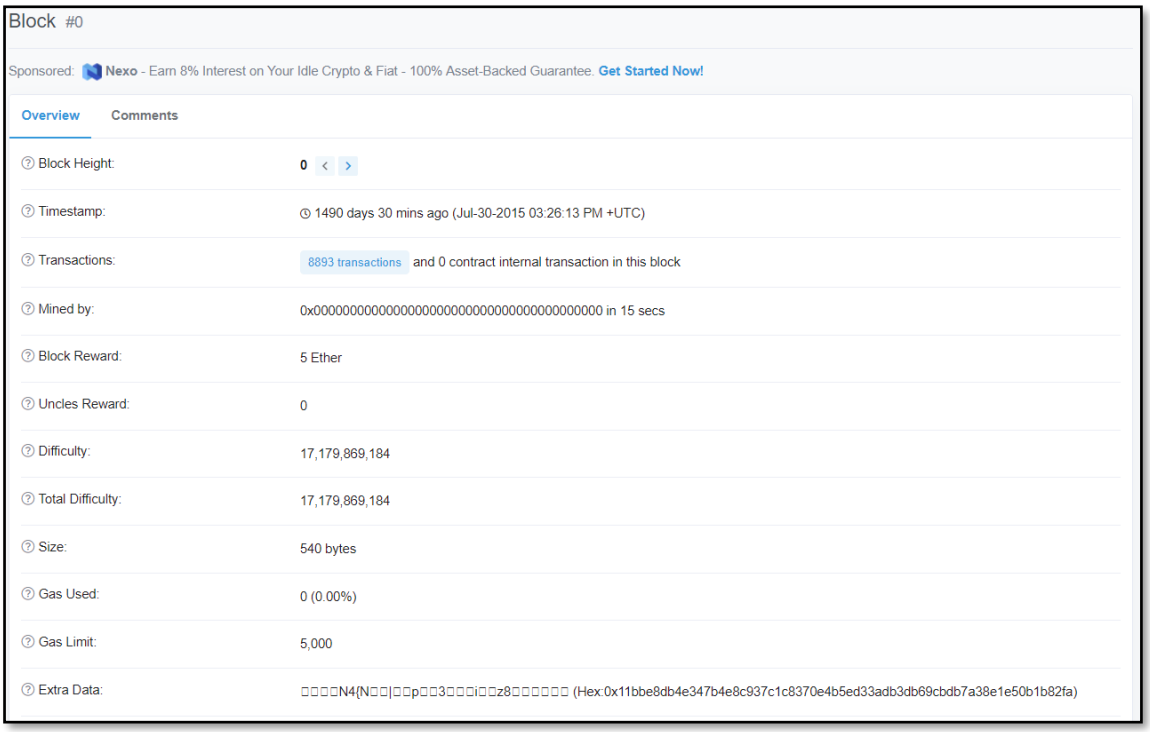


Ilustración 9. Bloque Génesis de Ethereum

3.7 Las redes de Ethereum

La red de Ethereum no es única, si no que en la actualidad existen diferentes redes que implementan el protocolo de Ethereum. Todas ellas son públicas y accesibles a través de un nodo cliente, pero para que dos nodos puedan ser conectados, deben compartir el mismo bloque génesis y el mismo identificador de red. En Ethereum existen tres tipos de redes:

- **Red principal o *mainnet:*** Se trata de la red utilizada para proyectos en producción en la que las transacciones con valor económico real.
- **Redes de pruebas o *testnet:*** Es la red utilizada en los entornos de test de los proyectos. Se usan tokens sin valor económico y las transacciones y sus tasas son simuladas. Excluyendo este aspecto, la red se comporta de forma parecida a la *mainnet.*
- **Redes privadas:** Son utilizadas para crear redes permisionadas dentro de entidades.

Ciertos equipos u organizaciones han dedicado esfuerzo en la creación de redes de prueba para que los desarrolladores puedan testear sus soluciones [17]. Las principales y más utilizadas hoy en día son:

- **Rinkeby:** Surgida en 2017, se trata de una Blockchain de *Proof of Authority.* Solo es compatible con clientes de tipo Geth*.* Sus desarrolladores son el *Ethereum Team*.
- **Kovan:** Su algoritmo de consenso es también *Proof of Authority* y solo es compatible con clientes de tipo Parity*.* Fue lanzada en 2017 por el *Parity Team.*
- **Ropsten:** Surgió en 2016, su algoritmo es de tipo *Proof of Work* y es compatible con ambos clientes: Geth y Parity*.*
- **Görli:** También es *Proof of Authority* y fue lanzada en 2018. Incluye múltiples clientes como Nethermind, Geth o Pantheon.

Para lograr fondos en estas redes existen lo que se denominan *faucets*, plataformas que emiten criptomonedas para que sea posible hacer frente a las tasas simuladas de las redes de prueba. Es decir, especificando una dirección de Ethereum, transfieren tokens hacia la misma (en este caso es necesario Ether). Recordemos que en estas redes las criptomonedas no tienen valor real. Por ejemplo, el *faucet* de *Rinkeby* es https://faucet.rinkeby.io/.

3.8 Los clientes de Ethereum

Los clientes son implementaciones de la Blockchain de Ethereum y son necesarios para poder participar en la red. A través de un cliente se realizan todas las operaciones que permite la red tales como la implementación, compilación y despliegue de contratos, la interacción con ellos, el almacenamiento de saldo en una cuenta, la minería de bloques, etc.

Existen varios clientes programados en diferentes lenguajes, algunos de ellos, como en el caso de las *testnets,* son desarrollados y mantenidos por los equipos oficiales, pero también hay otras implementaciones. Los clientes más empleados en la actualidad son:

- **Geth o go-ethereum:** Programado en Go por parte del equipo de Ethereum.
- **Parity:** Implementado en Rust por Ethcore.
- **Pyethapp:** Desarrollado en Python también por el equipo oficial de Ethereum.
- **thereumjs-lib:** Basado en Javascript.

3.9 Solidity

Solidity es el lenguaje de programación que permite implementar s*mart contracts* para crear aplicaciones descentralizadas de cualquier tipo sobre la red. Se dice

que es un lenguaje orientado a contratos y tipado que se ejecuta a través de la EVM. Surgió en 2014 y fue desarrollado por un equipo de Ethereum formado para ello. Existen otros lenguajes orientados a programar contratos como Mutan, Vyper o Serpent, pero sin duda Solidity es el más extendido [18].

Más adelante se exponen los mejores Entornos de Desarrollo Integrados (Integrate Development Environment en inglés o IDE) para desarrollar en Solidity, pero también es posible hacerlo instalando el compilador y a través de la línea de comandos. La instalación se puede realizar a través del paquete npm y el comando que hay que ejecutar es:

```
$ npm install -g solc
```

A continuación, se muestra el ejemplo de contrato que aparece en el repositorio *Github* de Solidity (https://github.com/ethereum/solidity):

```
pragma solidity ^0.5.0;

contract HelloWorld {
  function helloWorld() external pure returns (string memory) {
    return "Hello, World!";
  }
}
```

Ilustración 10. Contrato "HelloWorld" en Solidity

La primera línea indica cuál es la versión del compilador que se va a utilizar. El método *"helloWorld"* simplemente devuelve el *string "Hello, World!".*

Para compilar este s*mart contract* y obtener el *bytecode* que entiende la EVM se ejecuta el siguiente comando si establecemos el nombre del fichero como "helloWorld.sol" (.sol es la extensión de los contratos escritos en Solidity):

```
$ solc -o outputDirectory --opcodes helloWorld.sol
```

Para generar el binario a partir del *bytecode* se ejecuta:

```
$ solc -o BytecodeOutputDir --bin Example.sol
```

El *bytecode* resultante es el siguiente:

```
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0
DUP1 REVERT JUMPDEST POP PUSH2 0x11A DUP1 PUSH2 0x20 PUSH1 0x0 CODECOPY PUSH1
0x0 RETURN INVALID PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT
PUSH1 0x22 JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR PUSH4 0xFFFFFFFF AND
DUP1 PUSH4 0xC605F76C EQ PUSH1 0x27 JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT
JUMPDEST CALLVALUE DUP1 ISZERO PUSH1 0x32 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST
POP PUSH1 0x39 PUSH1 0xB1 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20
ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2 MLOAD DUP2 MSTORE PUSH1 0x20 ADD
SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0
JUMPDEST DUP4 DUP2 LT ISZERO PUSH1 0x77 JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5
ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH1 0x5E JUMP JUMPDEST POP POP POP
POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1 ISZERO PUSH1 0xA3 JUMPI
DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT
AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1
0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1 0x60 PUSH1 0x40 DUP1 MLOAD
SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0xD DUP2 MSTORE PUSH1 0x20 ADD
PUSH32 0x48656C6C6F2C20576F726C642100000000000000000000000000000000000000 DUP2
MSTORE POP SWAP1 POP SWAP1 JUMP INVALID LOG1 PUSH6 0x627A7A723058 KECCAK256 DUP5
0x5d 0xdc SHR SWAP6 DUP12 0xea 0xa9 MOD 0x2d OR 0xde PUSH18 0xF8A9C1A1D00F53C6A0A1C00712218B3EF4D8
0x4a STOP 0x29
```

Ilustración 11. Bytecode del contrato "HelloWorld" en Solidity

3.10 DApps

Gracias a la plataforma Ethereum y Solidity, es posible implementar aplicaciones descentralizadas (DApps) que interactúan con los *smart contracts* para ejecutar lógica de negocio sin ningún tipo de intermediarios y de manera descentralizada. La principal ventaja de estas aplicaciones es el nivel de seguridad, ya que todos los nodos de la red deben validar la información que se genera en el proceso sin que ninguna entidad central tenga el control.

Como ya se expuso anteriormente, los nodos cliente, son el punto de acceso a la Blockchain y, a través de ellos, las DApps interactúan con los contratos generando transacciones para solicitar las funciones que tengan implementadas.

A continuación, se resumen algunas de las características principales de las aplicaciones descentralizadas:

- La aplicación debe ser 100% *open source.*
- Su funcionamiento debe ser autónomo y sin intermediarios. Los cambios y mejoras que se realicen deben ser decididos por consenso de todos los participantes.
- La aplicación debe agrupar los intereses de todos los usuarios.
- Toda la información generada por la aplicación debe almacenarse en una plataforma descentralizada y abierta en la que sea posible realizar auditorías por parte de cualquier entidad.

Esto no quiere decir que todas las aplicaciones que se ejecutan como verticales en la plataforma de Ethereum sean DApps. De hecho, actualmente las empresas empiezan a tener aplicaciones Blockchain en producción que son de código abierto y

distribuidas, pero cuyo control reside completamente en la organización. La ya mencionada aplicación “CryptoKitties”, es un ejemplo de aplicación distribuida.

Es importante destacar que todos los nodos en los que se ejecuta una DApp necesitan dedicar recursos de computación en mantenerla ejecutando de manera segura y actualizada y es por ello, que la mayoría de DApps que han tenido éxito disponen de su propia criptomoneda interna con un valor que estará definido por la cantidad de usuarios que demanden la aplicación. El principal problema aquí es que este tipo de aplicaciones son en su mayoría de pago.

Arquitectónicamente, la principal diferencia entre las aplicaciones distribuidas y las aplicaciones tradicionales es que estas últimas implementan su *back-end* en lenguajes como Java, NodeJs, Pythono Go, y las aplicaciones distribuidas lo programan en forma de *smart contract*.

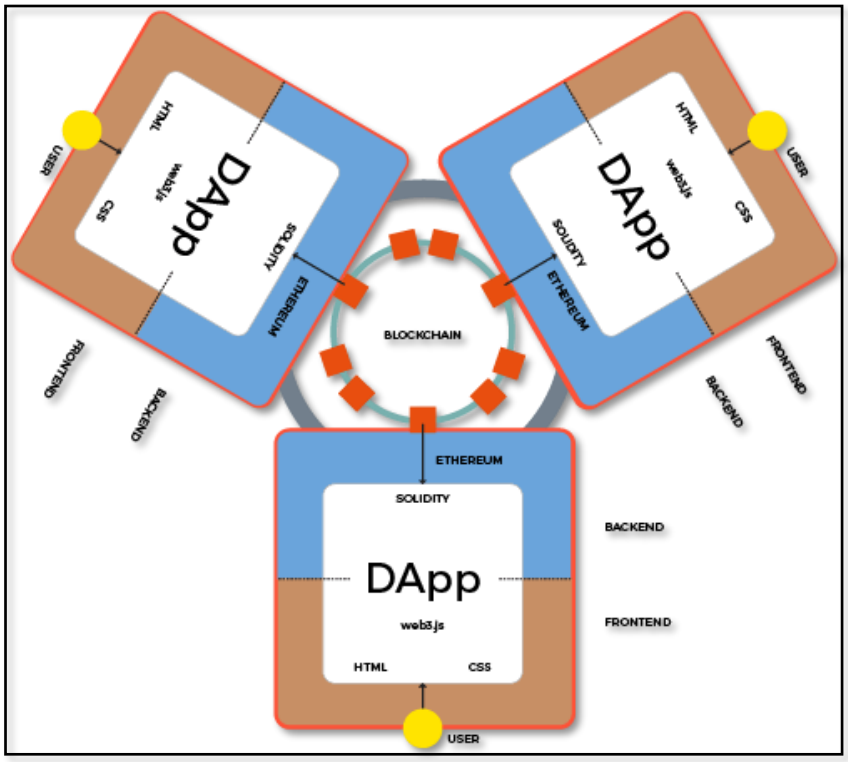


Ilustración 12. Arquitectura de una DApp

3.11 Entornos de programación y herramientas para Solidity

Como ya se ha explicado, es necesario compilar y desplegar un s*mart contract* en Ethereum para poder interactuar con él. Aunque el ecosistema de desarrollo de Solidity no cuenta aún con todas las herramientas o servicios con los que cuentan otros lenguajes, existen diferentes entornos de desarrollo integrados o *integrate development*

environments (IDEs) que soportan la programación y compilación de código escrito en Solidity y otros que están desarrollados exclusivamente para él.

IDEs como *Visual Studio Code* o *Jetbrains,* incluyen extensiones muy completas que incluyen *solc*, el compilador de Solidity [19].

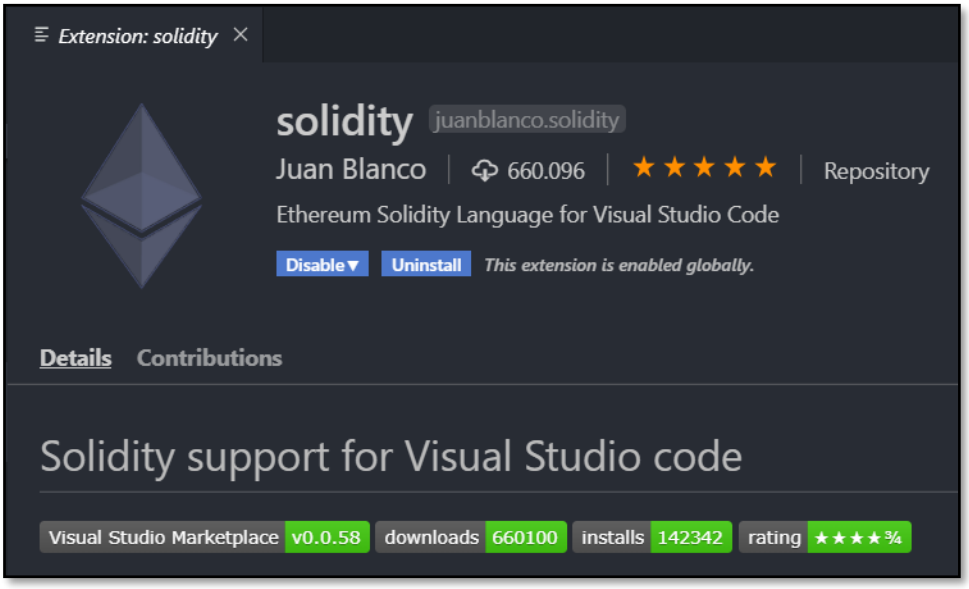


Ilustración 13. Extensión de Solidity para Visual Studio Code

- **EthFiddle:** Es un entorno de desarrollo lanzado por *Loom*, una empresa que apuesta por una Blockchain específica para *DApps* de alto rendimiento, que permite la compilación de código online a través de un navegador. Ofrece la capacidad de depuración y de simular el despliegue en su red (https://ethfiddle.com/).
- **Remix:** Este IDE es el más popular y con mejores capacidades. Desarrollado por el equipo de Ethereum, proporciona un entorno en el que implementar, compilar, depurar y desplegar los *smart contracts* en diferentes redes de prueba como *Rinkeby*, *Kovan* o *Ropsten.* Es accesible a través del navegador (https://remix.ethereum.org) pero también es posible instalarlo en local [20].

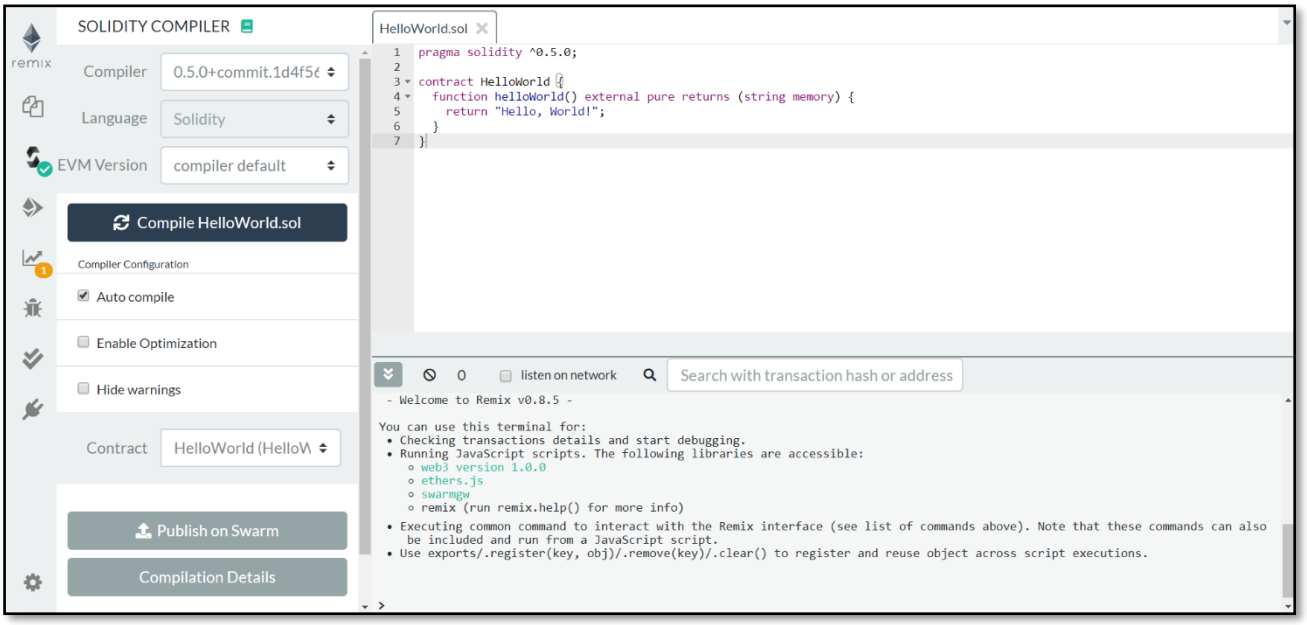


Ilustración 14. Remix IDE

- **Metamask:** Se trata de una extensión disponible para todos los basados en Chrome y Firefox que instala un *wallet* conectado a la red de Ethereum con el que es posible gestionar tus cuentas y balances en Ethereum. Es posible transferir balance a otras cuentas e integrar otros tipos de tokens diferentes al Ether. Tiene integración con el IDE Remix para gestionar las transferencias necesarias en el despliegue e interacción con los contratos inteligentes desplegados. Además, tiene integración con la red principal de Ethereum, pero también con *testnets* como *Ropsten, Kovan, Rinkeby* y *Goerli* [21].

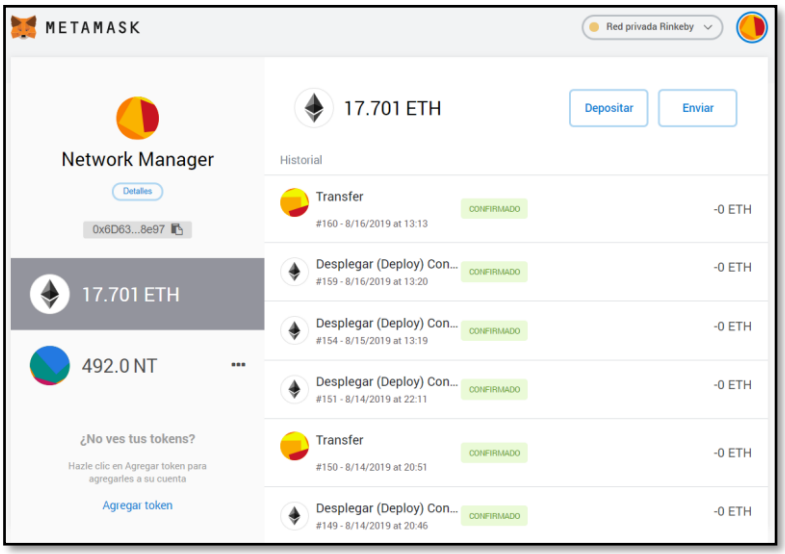


Ilustración 15. Metamask

- **Ganache:** Se trata de un *framework* desarrollado por Truffle que genera un entorno Blockhain simulado desplegando una red en local con 10 cuentas disponibles. En ella es posible desplegar contratos inteligentes, así como testearlos o depurarlos. Es una herramienta muy útil en la fase de desarrollo [22].

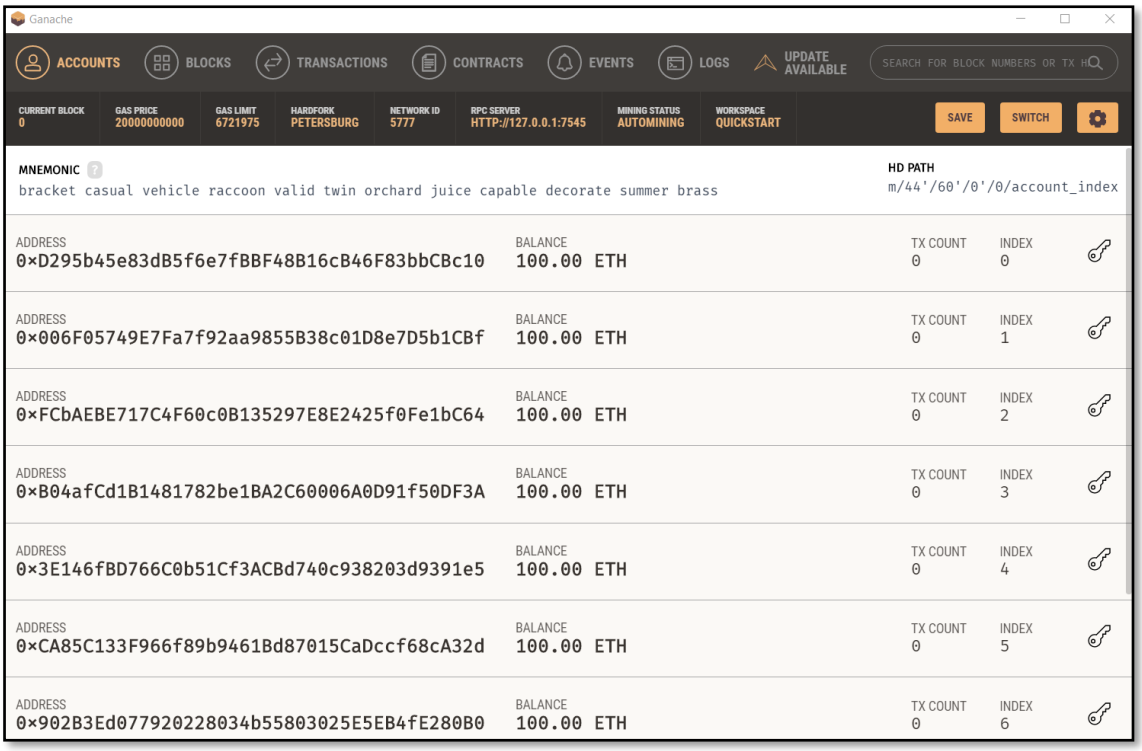


Ilustración 16. Ganache

- **CryptoZombies:** Es una plataforma online también desarrollada por *Loom,* que ofrece un curso interactivo de Solidity a través de un juego en el que debes construir un ejército de zombies. Empieza en el nivel básico y va aumentando la dificultad progresivamente y es ideal para desarrolladores principiantes [22]. Actualmente, están trabajando en ofrecer el curso para aprender a programar en *Libra Core*, la implementación del protocolo de la criptomoneda que lanzó Facebook, Libra [23].
- **Etherscan:** Es lo que se denomina un explorador de bloques de Ethereum a través del cual se pueden consultar bloques, transacciones, direcciones, o *smart contracts* desplegados. En la siguiente dirección se pueden consultar, por ejemplo, los 20 tokens ERC-721 más famosos, entre los que se encuentra por supuesto *CryptoKitties* (CK): https://etherscan.io/tokens-nft. Es posible acceder al contrato y ver toda la información asociada incluyendo las transacciones que se están realizando en tiempo real.

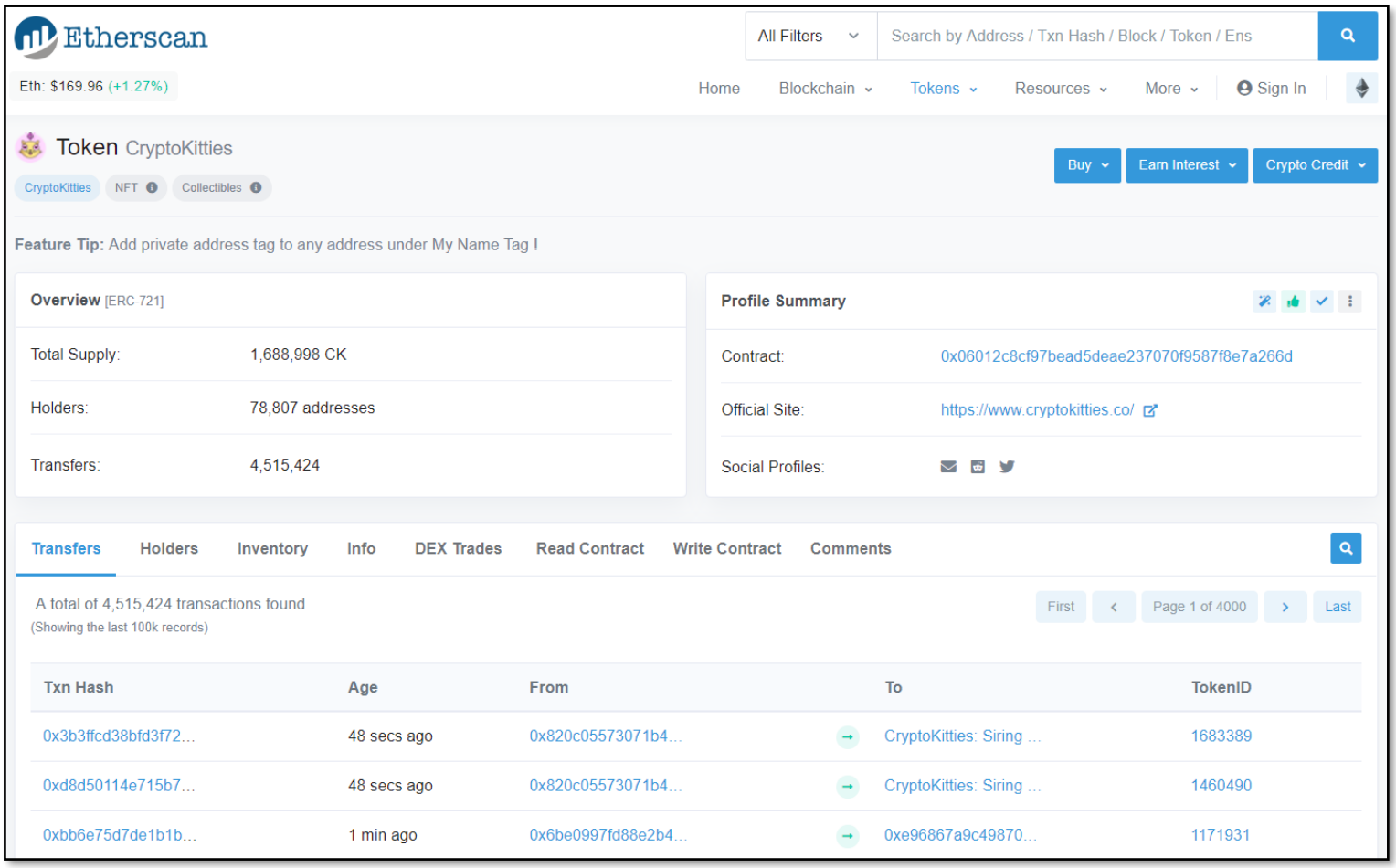


Ilustración 17. El contrato de CryptoKitties en Etherscan

Capítulo 4: Análisis

En el cuarto capítulo se analiza y propone una prueba de concepto que implemente un prototipo para verificar si tiene sentido aplicar la tecnología Blockchain en el desarrollo de una idea que pueda ser explotada en un entorno productivo.

Tras la investigación realizada en los dos capítulos anteriores, queda claro que la tokenización de activos abre un amplio abanico de posibilidades gracias a que se produce una representación digital cuya información asociada es incorruptible. Esta propiedad hace de la tokenización una capacidad muy interesante en entornos en los que se necesita transferir valor de manera fiable, segura y transparente.

La tokenización también es especialmente interesante cuando se deben gestionar recursos escasos como por ejemplo, la energía renovable o el poder computacional de un clúster. A partir del concepto de que casi cualquier activo de la vida real puede tokenizarse, solo es necesario estudiar y modelar el caso para poder aplicar esta capacidad.

En el caso del presente proyecto, se deseaba aplicar la tokenización en el ámbito de redes de ordenadores de cualquier tipo. Barajando otras posibilidades se tomó consciencia de que existen muchos recursos limitados en este entorno como pueden ser la memoria, el poder de cómputo o el ancho de banda y de que se podría generar una prueba de concepto muy interesante. De esta manera, se decidió dedicar la prueba de concepto a desarrollar un sistema que gestionara el ancho de banda a través de su tokenización, ya que puede aportar valor en escenarios muy diferentes.

A continuación, se define la solución propuesta que se desarrollará a través de un prototipo. No se trata de una solución concreta a un problema, sino de una prueba de concepto que se puede aplicar en diferentes escenarios de uso y puede dar lugar a múltiples aplicaciones.

En la prueba de concepto se propone un modelo de acceso a red que permita gestionar la prestación del servicio, el consumo y la facturación; todo ello respaldado en la plataforma de Ethereum. En concreto, el servicio será ofrecido por un *gateway point* o punto de acceso que ofrezca conexión inalámbrica a Internet y que, además, forme parte de la red de Ethereum como nodo para poder interactuar con los *smart contracts* que definan la lógica de la aplicación. Con este sistema se pretende que cualquier usuario pueda intercambiar dinero real por *"Internet tokens"* que representan la capacidad de consumir ancho de banda y son emitidos por la entidad o entidades que prestan el servicio de forma descentralizada. En función del consumo que estén realizando y de la política definida, se realizará automáticamente todo el proceso de facturación, transfiriendo los fondos correspondientes desde los clientes hacia la entidad. Incluso ofrece la posibilidad de que los usuarios vendan sus tokens a otros usuarios o directamente lo vuelvan a intercambiar por dinero real (en el caso de que, por ejemplo, no los fueran a consumir). En definitiva, se introduce la criptoeconomía en el servicio de acceso a Internet permitiendo el *trading* del token asociado.

Un aspecto clave de la solución propuesta, es que puede ser aplicada en escenarios muy diferentes. Por ejemplo, podría ser útil en un complejo hotelero que desee incentivar ciertos comportamientos del consumo del servicio. Reparten a sus clientes el identificador de la red y en vez de la contraseña, les transfieren los tokens que éstos hayan decidido comprar. Si, por ejemplo, desean penalizar el consumo de *streaming* porque perjudica a los usuarios que estén realizando tareas laborales, en función de la política que hayan definido, se produce una facturación mayor.

Otro escenario posible es aquel en el que usuarios con servicio de Internet contratado, ofrezcan conexión a otros usuarios a cambio de cierta cantidad de tokens. Desde el punto de vista del consumidor, es posible disfrutar de una conexión privada a través de cualquier punto de acceso en cualquier lugar del mundo. Desde el punto de vista del prestador del servicio, obtiene beneficio por ofrecer parte de su conexión.

En el siguiente apartado se diseña una de las múltiples aplicaciones que pueden identificarse en este ecosistema.

4.1 Requisitos de usuario

A lo largo de este subapartado se especifican los principales requisitos de usuario que indican las características o funcionalidades que debe incluir la solución. Serán representados en tablas con el siguiente formato:

- **Identificador:** Identifica al requisito de forma única cuya nomenclatura es ***RU_XX*** donde ***XX*** se sustituye por el número que le corresponda.
- **Descripción:** En ella se detalla el requisito de manera sencilla y precisa.
- **Prioridad:** Indica el grado de importancia del requisito dentro de la solución. Los valores que adopta son “Baja”, “Media” y “Alta.

Identificador: RU_01	
Descripción	La solución debe introducir la criptoeconomía en la prestación del servicio de acceso a Internet
Prioridad	Alta

Tabla 1. Requisito de usuario RU_01

Identificador: RU_02	
Descripción	La solución debe gestionar la prestación del servicio de manera automática
Prioridad	Alta

Tabla 2. Requisito de usuario RU_02

Identificador: RU_03	
Descripción	La solución debe gestionar el consumo del servicio de manera automática
Prioridad	Alta

Tabla 3. Requisito de usuario RU_03

Identificador: RU_04	
Descripción	La solución debe gestionar la facturación del servicio de manera automática
Prioridad	Alta

Tabla 4. Requisito de usuario RU_04

Identificador: RU_05	
Descripción	La solución debe ofrecer el servicio a través de un *gateway* o punto de acceso inalámbrico
Prioridad	Media

Tabla 5. Requisito de usuario RU_05

Identificador: RU_06	
Descripción	La solución debe respaldarse en la plataforma Blockchain de Ethereum
Prioridad	Alta

Tabla 6. Requisito de usuario RU_06

Identificador: RU_07	
Descripción	El dispositivo *gateway* debe ser un nodo de la red de Ethereum
Prioridad	Alta

Tabla 7. Requisito de usuario RU_07

Identificador: RU_08	
Descripción	La solución debe almacenar toda la información generada en Ethereum de forma transparente
Prioridad	Alta

Tabla 8. Requisito de usuario RU_08

Identificador: RU_09	
Descripción	La solución mostrará la información asociada a monitorización de la red
Prioridad	Media

Tabla 9. Requisito de usuario RU_09

Identificador: RU_10	
Descripción	La solución mostrará la información asociada al usuario
Prioridad	Media

Tabla 10. Requisito de usuario RU_10

Identificador: RU_11	
Descripción	La solución permitirá que el usuario haga una oferta en tokens para poder consumir el servicio
Prioridad	Alta

Tabla 11. Requisito de usuario RU_11

Identificador: RU_12	
Descripción	La solución debe permitir a los usuarios abonar el servicio mediante tokens previamente adquiridos
Prioridad	Alta

Tabla 12. Requisito de usuario RU_12

Identificador: RU_13	
Descripción	La solución mostrará información acerca de todas las transacciones sobre Blockchain que se generen en el sistema
Prioridad	Media

Tabla 13. Requisito de usuario RU_13

4.2 Requisitos del sistema

Tras definir los requisitos de usuario es necesario detallar los requisitos del sistema, cuyo objetivo es especificar de manera más concreta las propiedades de la solución, es decir, cómo se van a cumplir los requisitos de usuario. El formato va a ser análogo al de los requisitos de usuario exceptuando que la nomenclatura es ***RS_XX*** donde ***XX*** se sustituye por el número que le corresponda y se añade el campo "Origen" en el que se especifica qué requisito de usuario recoge.

Identificador: RS_01	
Descripción	Desplegar un smart contract ERC-20 para tokenizar la prestación del servicio de acceso a Internet
Prioridad	Alta
Origen	RU_01

Tabla 14. Requisito del sistema RS_01

Identificador: RS_02	
Descripción	Implementar módulo software para gestionar la prestación, el consumo y la facturación del servicio
Prioridad	Alta
Origen	RU_02, RU_03 y RU_04

Tabla 15. Requisito del sistema RS_02

Identificador: RS_03	
Descripción	Utilizar un dispositivo Raspberry Pi 3 Model B+ como *gateway* o punto de acceso inalámbrico del servicio
Prioridad	Media
Origen	RU_02, RU_03, RU_04, RU_05

Tabla 16. Requisito del sistema RS_03

Identificador: RS_04	
Descripción	Instalar un cliente de Ethereum tipo Geth para poder interactuar con la red de Ethereum
Prioridad	Alta
Origen	RU_06, RU_07

Tabla 17. Requisito del sistema RS_04

Identificador: RS_05	
Descripción	Desplegar un smart contract en Ethereum que gestione toda la información de la monitorización del servicio
Prioridad	Alta
Origen	RU_08

Tabla 18. Requisito del sistema RS_05

Identificador: RS_06	
Descripción	Implementar una interfaz de usuario para visualizar la información de monitorización de la red y de usuario
Prioridad	Alta
Origen	RU_09, RU_10

Tabla 19. Requisito del sistema RS_06

Identificador: RS_07	
Descripción	Instalar el plugin de Metamask en el navegador
Prioridad	Alta
Origen	RU_11, RU_12 y RU_13

Tabla 20. Requisito del sistema RS_07

Identificador: RS_08	
Descripción	Integrar Metamask en la interfaz de usuario
Prioridad	Alta
Origen	RU_11, RU_12 y RU_13

Tabla 21. Requisito del sistema RS_08

Capítulo 5: Diseño

En este capítulo se realiza un recorrido por el entorno, las plataformas y las herramientas de desarrollo existentes en Blockchain que han sido seleccionadas para el desarrollo del prototipo de la prueba de concepto.

5.1 Hardware

La prueba de concepto se basa en un *gateway point* o punto de acceso que ofrezca conexión inalámbrica a Internet y que, además, forme parte de la red de Ethereum como nodo para poder interactuar con los *smart contracts* que definen la lógica de la aplicación. Desde la fase inicial del proyecto se decidió que se haría uso de un dispositivo IoT para lograr el objetivo. La decisión fue tomada en base a dos aspectos:

- Las características que ofrecen este tipo de dispositivos son ideales para la solución propuesta ya que, son dispositivos de tamaño reducido y cuya capacidad computacional de recopilación y procesamiento de datos, es más que suficiente para lo que han sido diseñados.
- Su naturaleza encaja perfectamente con la estructura descentralizada de las plataformas Blockchain, ya que permiten un sencillo escalado añadiendo más dispositivos que ejecuten un cliente, contribuyendo así a la descentralización de la red. Además, este tipo de dispositivos vienen equipados con un conjunto muy completo de puertos y accesorios, lo cuales les hacen ideales para soluciones en las que la conectividad cobra un papel importante.

En base a estos conceptos, se podría incluso plantear una red permisionada solo dedicada al acceso tokenizado a Internet entre las diferentes empresas de telecomunicaciones. Solo sería necesario instalar el cliente de Ethereum y la aplicación propuesta en *n* dispositivos, donde *n* es un número suficientemente grande para garantizar la descentralización de la red.

Los dispositivos barajados para implementar el prototipo fueron la Raspberry Pi 3B+ [25] y el Arduino UNO WiFi Rev2 [26]. A continuación, se hace una comparativa entre ambos:

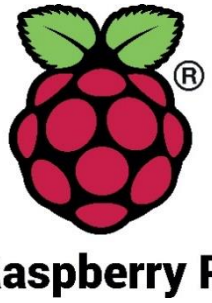
	
CARACTERÍSTICAS TÉCNICAS	
ARDUINO UNO WIFI REV2	RASPBERRY PI 3 MODEL B+
Microcontrolador ATmega 4809 de 8 bits y 20MHz	Microprocesador BCM2837B0, Cortex-A53 64-bit (ARMv8) 1.4Ghz
14 E/S digitales y 6 analógicas	40 E/S digitales o analógicas
Memoria Flash de 48KB y 6KB de SRAM	Memoria RAM de 1GB
Conexión WiFi 2.4 GHz USB, SPI e I2C	Conexión WiFi 2.4 GHz, USB (4), Micro SD, HDMI, DSI, CSI, Bluetooth 4.2
No soporta ningún sistema operativo, el programa se carga directamente a la placa vía serial	Sistema operativo oficial Raspbian
38,90 € (Web Arduino)	35,75€ (Amazon)

Tabla 22. Comparativa de selección de dispositivo IoT

Finalmente se decidió optar por la Raspberry Pi porque se trata de un dispositivo mucho más potente que la placa Arduino. En el siguiente apartado se detallan los motivos principales de la elección.

5.2 Software

El prototipo desarrollado consta de varios módulos software que, por su naturaleza, deben ser implementadas con diferentes tecnologías e integradas entre ellas en lo que se puede denominar sistema o aplicación:

- ***Network monitor:*** Es el módulo encargado de gestionar la red. Permite el acceso a la red en base a la política definida y recoge todos los parámetros de monitorización de la red como por ejemplo el ancho de banda ofrecido o el consumo por parte de los usuarios. Genera la información necesaria para el funcionamiento del sistema y es ejecutado permanentemente para monitorizar la red en tiempo real.
- ***Ethereum Registry Monitor:*** Módulo dedicado al registro de la información producida por el *network monitor* en la plataforma de Ethereum. Es invocado por el *network monitor* cada vez que genera información nueva.
- ***Smart contracts:*** Implementan la lógica de la solución propuesta. Son dos los contratos implementados, cada uno de ellos con una lógica diferente:
  - **Network Monitoring:** Es el encargado de gestionar el registro de la información en Ethereum.
  - **ERC20:** Token que representa una unidad de "capacidad de uso del servicio" y que es la base del funcionamiento del sistema. Es necesario tener balance para poder disfrutar de conexión a Internet. Este contrato automatiza las transferencias de tokens entre los usuarios y el proveedor de servicio en función de la política implementada.
- **Aplicación web:** Coordina la ejecución de todos los módulos y presenta la información. Se compone de una interfaz de usuario (*front-end*) para visualizar toda la información y un controlador (*back-end*) que lee la información de la red almacenada en Ethereum y hace los cálculos necesarios para la automatización de la facturación del servicio, sirviendo todos los datos a la interfaz.

5.3 Tecnología Hardware seleccionada

La elección fue la Raspberry Pi puesto que es capaz de correr una versión de Linux oficial (Raspbian) y porque sus características son muchísimo mejores sobre todo en lo referido a la potencia del microprocesador y la memoria RAM. Además, tiene memoria extensible a través de tarjeta Micro SD por lo que los requisitos de almacenamiento quedan satisfechos. Es un ordenador de placa reducida capaz de instalar y gestionar la

ejecución de un nodo cliente de Ethereum sin problema y que además, soporta todas las necesidades de programación de los diferentes módulos del sistema. Se trata de un dispositivo que es ya más potente que algunos ordenadores de hace algunos años.

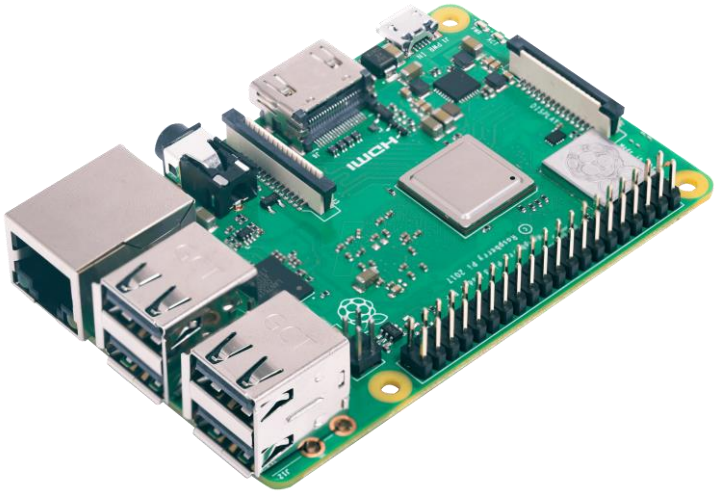


Ilustración 18. Raspberry Pi 3B+

5.4 Tecnología Software seleccionada

Para desarrollar el prototipo se han seleccionado las siguientes tecnologías software:

Python

Es un lenguaje de programación orientado a objetos e interpretado, lo que quiere decir que su código no se compila, sino que se traduce en tiempo de ejecución. También es un lenguaje dinámico (una misma variable puede ser de dos tipos diferentes en dos momentos diferentes) y capaz de ejecutar en múltiples plataformas [27]. Es de naturaleza *open source* y fue lanzado en 1991 por el holandés Guido van Rossum. Se trata de un sistema potente, con gran popularidad en ámbitos como web o procesamiento de datos y con una importante comunidad [28].

Además, tiene una buena librería de escaneo de paquetes de red o *network sniffing* denominada "*Scapy*" muy útil para la prueba de concepto. Por todos estos aspectos, fue el lenguaje elegido para la programación de los dos módulos monitores.

JavaScript

También es un lenguaje de programación interpretado, dinámico y orientado a objetos. Conocido como JS, generalmente se utiliza en proyectos web del lado del cliente (*client-side*) aportando una experiencia positiva en las páginas web interactivas y con un buen rendimiento [29]. Además, es posible utilizar JavaScript en el lado del servidor (*server-side*) y alcanza mejor rendimiento que la implementación de otros servidores

como Apache o PHP [30]. Una de las implementaciones más extendidas en el lado del servidor es Node.js. Es de naturaleza *open source y* genera un entorno de ejecución JavaScript cuyo objetivo es mejorar la eficiencia y el rendimiento del servidor web [31].

Node.js fue el *framework* elegido para la implementación del *back-end* de la aplicación web propuesta en el prototipo.

React

Denominado también ReactJS o React.js se trata de una biblioteca open source para implementar interfaces de usuario de una sola página, es decir interfaces que cargan todos los recursos necesarios al principio ofreciendo así una experiencia más ligera y fluida [32]. Fue lanzado en 2013 por Facebook y está completamente extendido entre los desarrolladores web. React hace uso de lo que se denomina componente, una entidad que tiene estado y vista. Los componentes tienen entre sí una relación jerárquica padre-hijo en la que el flujo de datos es unidireccional desde el padre hacia el hijo. Básicamente, cuando cambia el estado de un componente se actualiza su vista.

ReactJS es por tanto la biblioteca utilizada para desarrollar la interfaz de usuario del prototipo.

Web3.js

Es la librería necesaria para interactuar con la plataforma de Ethereum. Ofrece una *Application Programming Interface* (API) para invocar las funciones de un *smart contract* desplegado en la red y está disponible tanto para JavaScript [33] como para Python [34].

Nodo Geth y smart contracts

El cliente de Ethereum seleccionado para la creación del prototipo es Geth puesto que es el implementado por los desarrolladores de Ethereum. Como ya se adelantó, al seleccionar este tipo de cliente, la única opción *testnet* que soporta es Parity, también desarrollada por el equipo de Ethereum.

Para implementar los contratos inteligentes en la red pública de Ethereum, la única opción es hacerlo a través de Solidity.

5.5 Esquema del prototipo

Por lo tanto, el esquema a grandes rasgos del sistema quedaría de la siguiente manera:

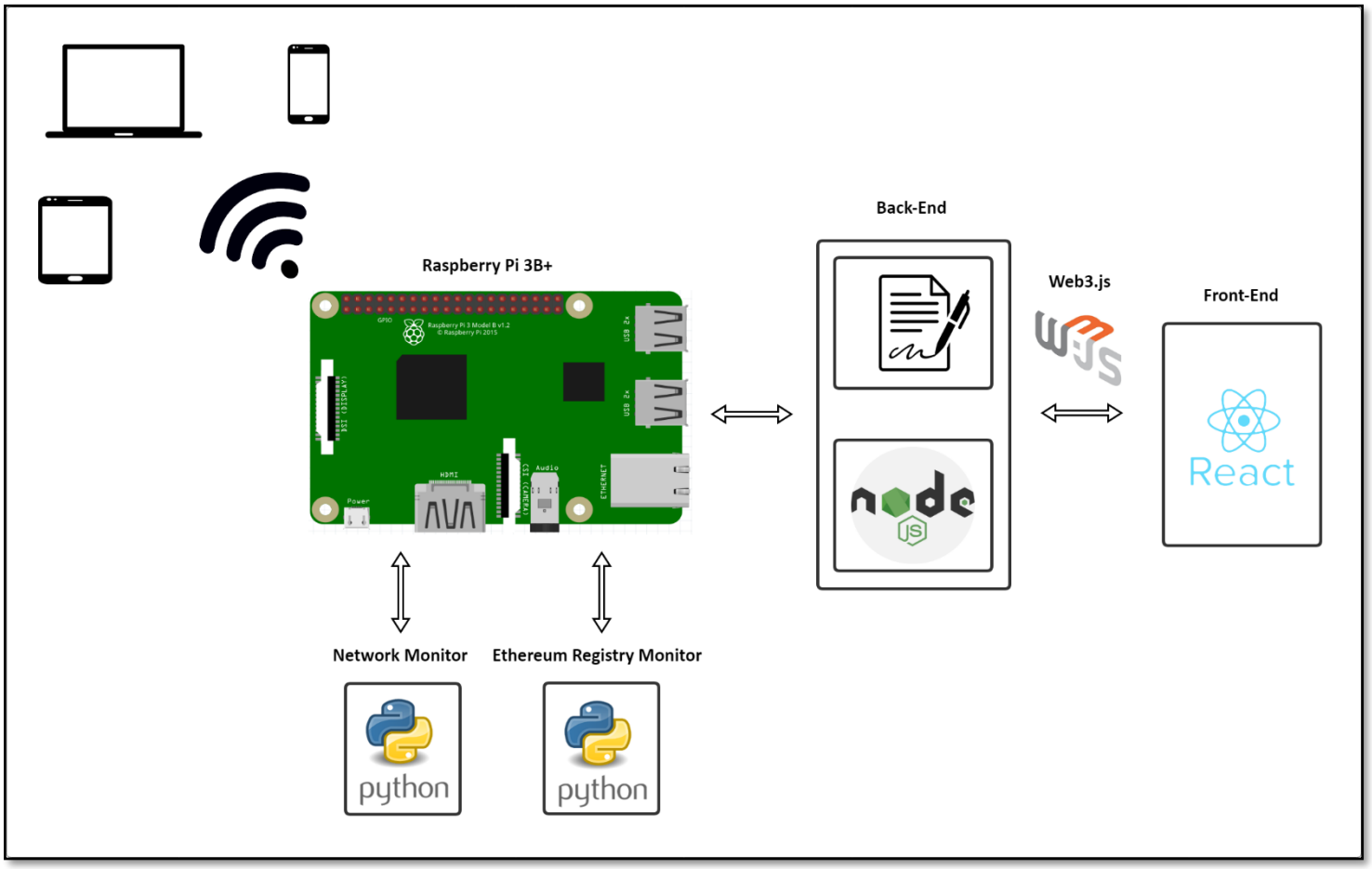


Ilustración 19. Esquema del prototipo

5.6 Arquitectura del sistema

Al tratarse de una aplicación en la que se incorpora Blockchain, la arquitectura del sistema es un poco especial. Podría decirse que el modelo al que más se ajusta es el clásico Modelo-Vista-Controlador (MVC) en el que las diferentes partes están compuestas por:

- **Modelo:** Está compuesto por los módulos Network Monitor y Ethereum Registry Monitor, encargados de enviar los datos a los *smart contracts* para que los registren en la red Blockchain en función de la lógica de negocio que definan. Es por ello que ambos ambos *smart contracts* (Network Monitoring e Internet Token) forman parte también del modelo.

- **Vista:** Se compone de la interfaz de usuario encargada de mostrar toda la información que se genera en el sistema.
- **Controlador:** Lo compone el módulo *controller* del *back-end*, cuya función se basa en demandar datos a los *smart contracts* una vez que le llegan peticiones desde la interfaz de usuario.

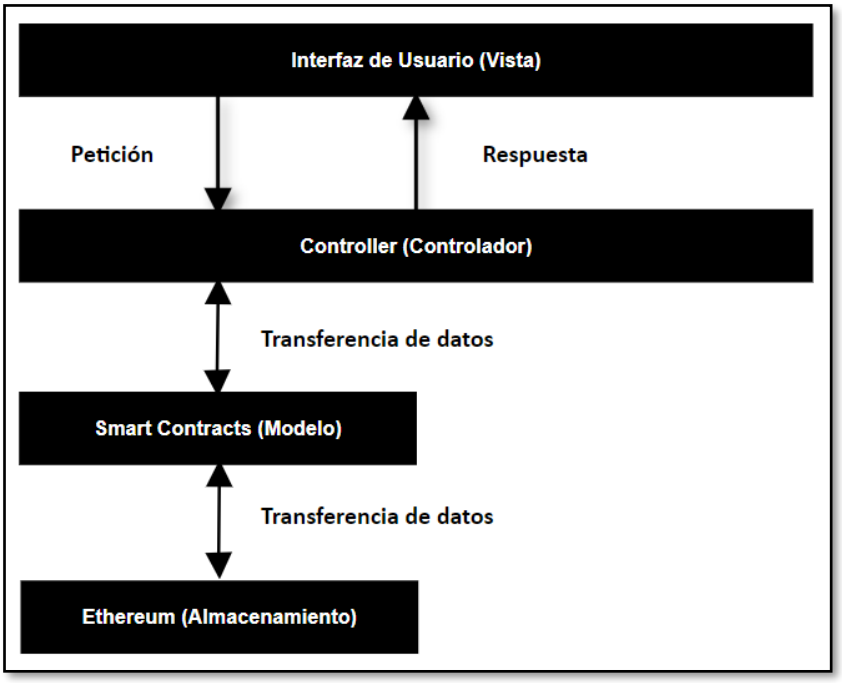


Ilustración 20. Arquiectura del sistema

5.7 Diseño del servidor (*back-end*)

El *back-end* se compone de tanto de los *smart contracts* (Network Monitoring y ERC20, ya explicados en el subapartado de software del presente capítulo) encargados de gestionar toda la lógica Blockchain como de un controlador que permite acceder a los datos almacenados en Ethereum a través de invocaciones a los *smart contracts*. El controlador implementa las funciones necesarias para pedir datos y realizar los cálculos necesarios y estará ejecutando continuamente a la espera de peticiones por parte de la interfaz de usuario.

A continuación, se puede observar un diagrama de secuencia de una petición que llega desde la interfaz de usuario y llega hasta el contrato inteligente que es el que sirve los datos:

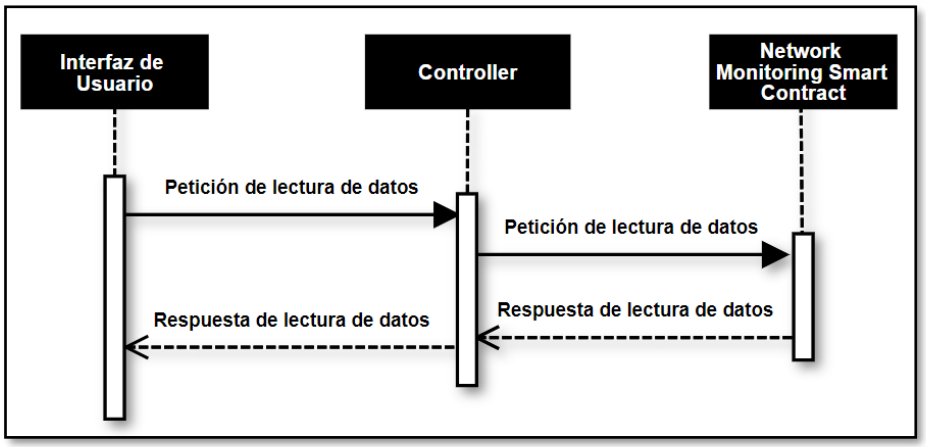


Ilustración 21. Diagrama de secuencia de una petición de datos

5.8 Diseño de la interfaz de usuario (*front-end*)

El *front-end* necesita una fase de diseño en la que se definan los componentes visuales que va a incluir la interfaz. Es importante hacer este tipo de diseño para lograr que la interfaz sea agradable, sencilla y fácil de usar.

Para lograr una versión aproximada de la interfaz se hará uso de *wireframes* que representarán el esqueleto de las páginas.

De esta manera, la interfaz se compone de dos vistas:

- **Pantalla de entrada a la aplicación:** Es la pantalla de presentación que carga cuando se arranca la aplicación. Contendrá el título de la plataforma *"Network Tokenized Access"* junto con un icono y un botón que abre paso a la pantalla principal de la aplicación.
- **Pantalla principal de la aplicación:** Contiene los elementos principales de la plataforma. Se subdivide en dos componentes:
  - **Network Monitor Panel:** En la parte superior de la pantalla, se compone de los elementos que muestran el estado de la red monitorizada por el *gateway*. Los elementos principales que se muestran son las direcciones IP correspondientes a los dispositivos conectados al punto de acceso, la ocupación de la red y el precio de acceso al servicio.
  - **User Panel:** A través de este componente los usuarios solicitan acceso a la red. Como se explicó anteriormente, los usuarios deben hacer una oferta en tokens para que el gestor de la red se decante por un usuario u otro en función del valor de la suma de la oferta del usuario y el precio de acceso. Se propone una vista con un campo de texto y

un botón para introducir la cantidad de tokens de la oferta y enviar la petición de acceso al monitor.

A continuación, se muestran ambos *wireframes:*

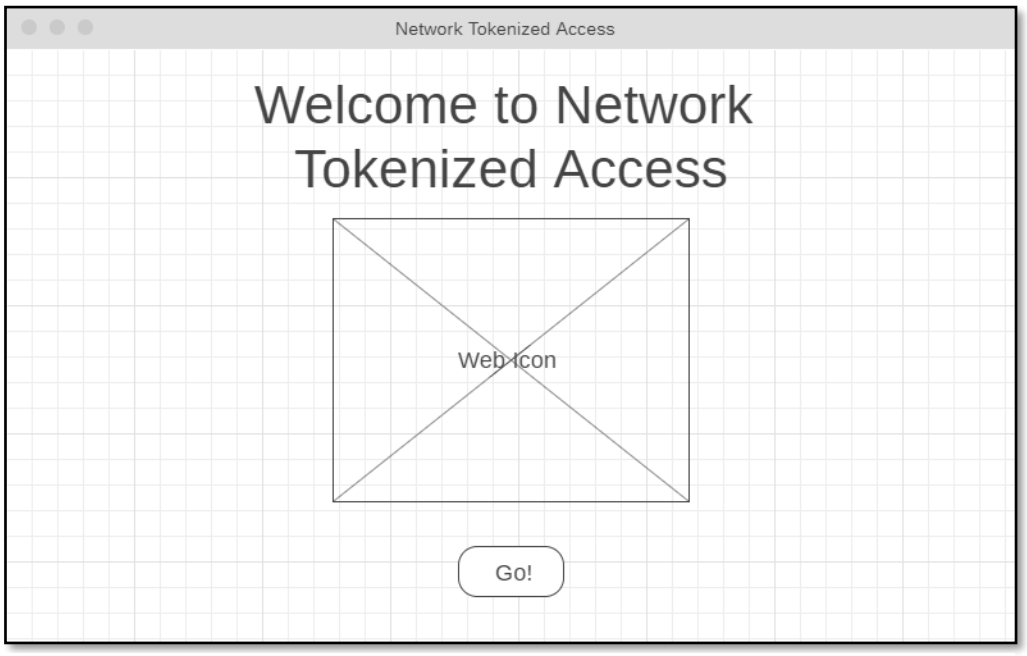


Ilustración 22. Wireframe pantalla inicio

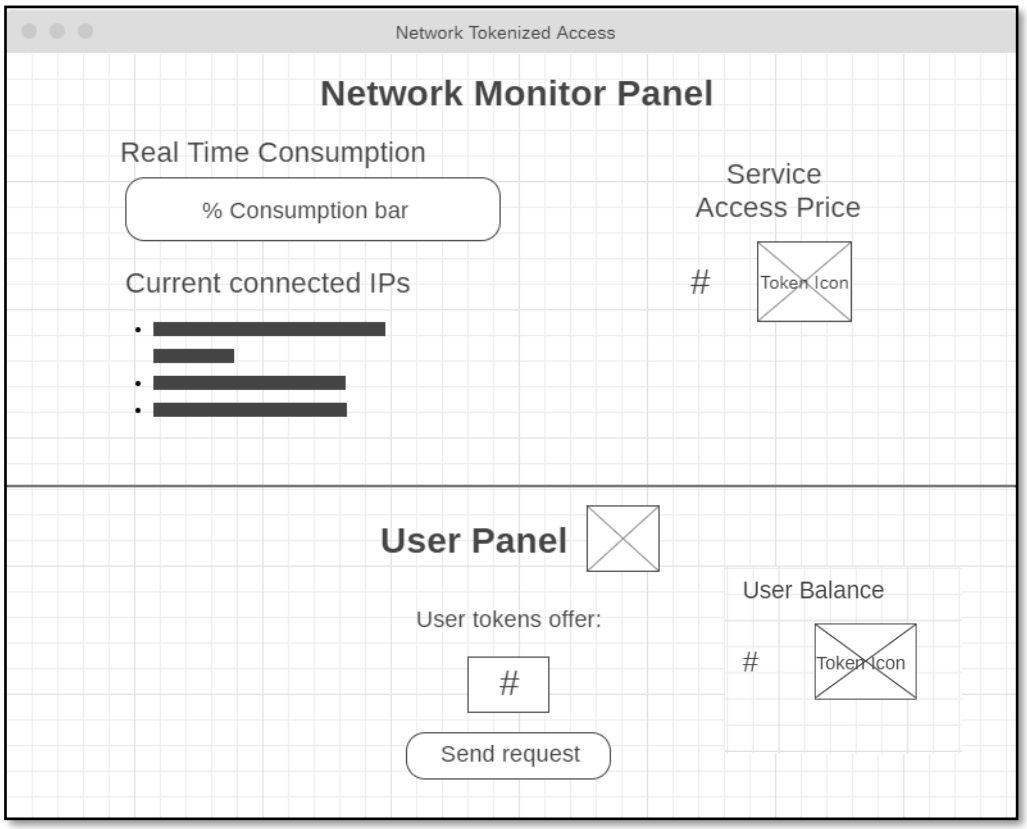


Ilustración 23. Wireframe pantalla principal

Capítulo 6: Implementación

En este apartado, se describe el desarrollo técnico de los diferentes módulos que conforman el prototipo propuesto, haciendo hincapié en todos aquellos aspectos complejos, más significativos o de mayor relevancia.

6.1 Preparación del entorno de trabajo

Para poder implementar el prototipo planteado, es necesario primero configurar un entorno de trabajo con todas las herramientas necesarias. El dispositivo Raspberry será el encargado tanto de desplegar y ejecutar el nodo de Ethereum como de albergar los módulos software de monitorización y la aplicación web.

La programación del dispositivo se realizó siempre a través de conexión remota gracias al protocolo *Secure Shell* (ssh) y gestionando los cambios a través del software de control de versiones Git (la plataforma de control de versiones utilizada fue Github). En primer lugar, es necesario habilitar la conexión ssh en el dispositivo a través de los siguientes comandos:

```
$ sudo systemctl enable ssh
$ sudo systemctl start ssh
```

Una vez habilitada, es posible conectarse remotamente al dispositivo. El programa utilizado para la conexión fue *MobaXterm*.

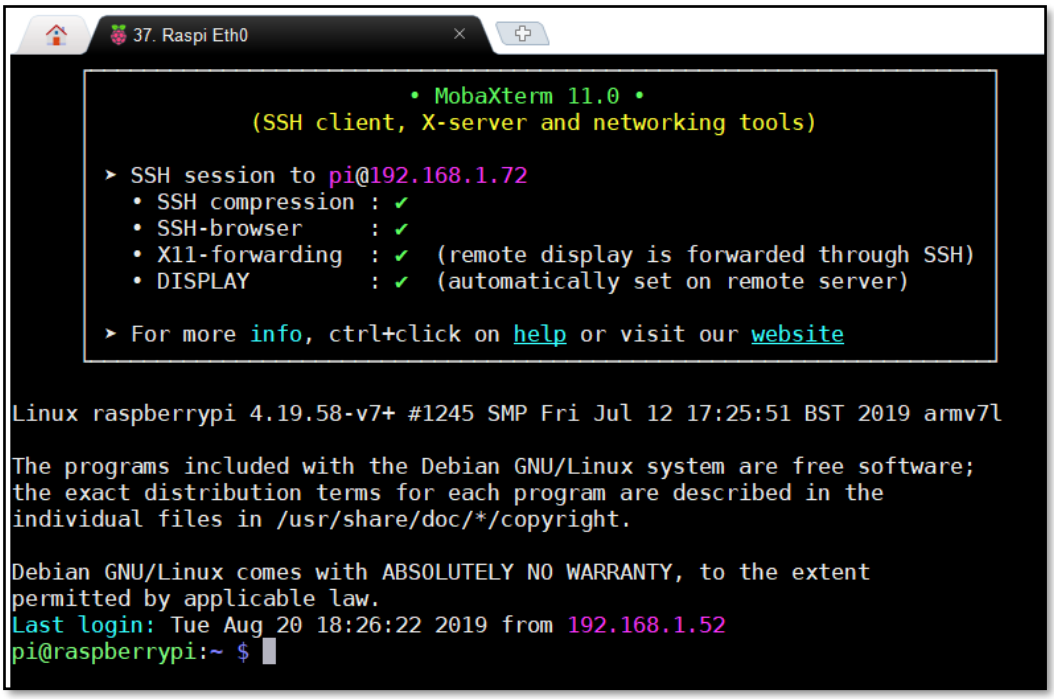


Ilustración 24. Conexión ssh con Raspberry a través de MobaXterm

El primer paso fue configurar el dispositivo Raspberry como *gateway* o punto de acceso a Internet para que haga el papel de *router* y poder monitorizar todo el tráfico que fluye a través del mismo. El proceso de configuración se encuentra detallado en el Anexo A: Configuración de Raspberry Pi 3B+ como gateway de acceso inalámbrico a red.

A continuación, se procedió a la instalación y ejecución del cliente de Ethereum *Geth,* mediante el cual, el dispositivo empezará a formar parte de la red de Ethereum. Puesto que se trata del desarrollo de una prueba de concepto, la red utilizada no es la principal, sino la *testnet* de *Rinkeby*. Todo este proceso se encuentra explicado en el Anexo B: Instalación y arranque de un nodo Geth en Raspberry Pi 3B+.

6.2 Network Monitor

Como ya se adelantó, la programación de este módulo se realizó en Python. Su funcionalidad principal es escanear los paquetes IP que fluyen a través de la interfaz utilizada.

```
def trafficMonitorCb(packet):
    if IP in packet:
        packet = packet[IP]
        consumption.update({tuple(sorted(map(atol, (packet.src, packet.dst)))): packet.len})


def networkMonitoring():
    print("Sniffing the wireless access point for 60 seconds...")

    # sniff IP packets
    sniff(count =0, iface=interface, prn=trafficMonitorCb, store=False,
        timeout=60)

    # set the consumption by IP address
    for (source, destination), total in consumption.most_common():
        source, destination = map(ltoa, (source, destination))
        for host in (source, destination):
            if host not in hosts:
                try:
                    rhost = socket.gethostbyaddr(host)
                    hosts[host] = rhost[0]
                except:
                    hosts[host] = None
```

Ilustración 25. Porción de código de Network monitor

Esta porción de código escanea el tráfico IP de la interfaz con la frecuencia deseada (en este caso cada 60 segundos) y almacena en una estructura de tipo *counter* la una tupla para la dirección IP y el consumo de cada dispositivo conectado a la red.

También se encarga de establecer el ancho de banda de la red que se va a ofrecer (en este caso se limita a 500Mb) mediante el programa tcconfig [35]:

```
sudoPassword = 'password'
command = 'tcset wlan0 --rate 500Mbps --overwrite'
p = os.system('echo %s|sudo -S %s' % (sudoPassword, command))
```

Ilustración 26. Configuración del ancho de banda del gateway

Cuando recopila todos los datos de consumo de ancho de banda llama al módulo *Ethereum Registry Monitor* para que vuelque en la red de Ethereum toda la información.

6.3 Ethereum Registry Monitor

Este módulo realiza la conexión con el nodo Geth que se encuentra ejecutando en el dispositivo e invoca al *smart contract* encargado de gestionar el registro de la información en Blockchain. Es invocado por el módulo *Network Monitor* cuando tiene datos de monitorización.

```
address = "0xb02e3c1294618A50eb3e04772CACB19E55eb3c2D"
account = "0x6D63c734cfF8b067E8D34752A59EEA1eB93a8e97"
privateKey = "Private key"

def sendInfoToEthereum(consumeByIpN):
    # connection to Ethereum node
    web3 = Web3(Web3.HTTPProvider("http://127.0.0.1:8545"))

    # get the abi of the smart contract
    with open('NMContract.json', 'r') as abi_definition:
        abi = json.load(abi_definition)
    # instantiate the smart contract
    contract = web3.eth.contract(address=address, abi=abi)

    # unlock the account to operate with
    try:
        senderAccount = web3.eth.account.privateKeyToAccount(privateKey)
    except:
        print("ERROR: Not a valid private key")
        sys.exit(1)

    # for each ip send a transaction with its consumption
    for ip, consumption in consumeByIpN.items():
        if consumption != 0:
            # estimate the gas needed to send the transaction
            estimateGas = contract.functions.setUserConsumption(ip, consumption).estimateGas()
            # build the transaction
            tx = contract.functions.setUserConsumption(ip, consumption).buildTransaction({
                'gas': estimateGas, 'nonce': web3.eth.getTransactionCount(senderAccount.address)
            })
            # sign the transaction
            signed = web3.eth.account.signTransaction(tx, senderAccount.privateKey)
            # send the transaction and wait for response
            txHash = web3.eth.sendRawTransaction(signed.rawTransaction)
            while (not txHash):
                print("Sending transaction...")
            print("Succesfully sent transaction ", {web3.toHex(txHash)})
```

Ilustración 27. Porción de código de Ethereum Registry Monitor

Como se puede observar en el código para registrar información son necesarios 4 parámetros:

- **La dirección del *smart contract*:** Con el formato *"0x…"* la dirección identifica a un contrato desplegado en la red y es necesaria para poder interactuar con

él. En este caso la dirección del contrato que gestiona el registro de la información en Ethereum es *"0xb02e3c1294618A50eb3e04772CACB19E55eb3c2D",* y se puede observar su información en un explorador de la red como *Rinkeby* (https://rinkeby.etherscan.io/address/0xb02e3c1294618A50eb3e04772CACB19E55eb3c2D)

- **La cuenta de Ethereum:** Se trata de la cuenta con la que se están realizando todas las transacciones en Ethereum y se corresponde con "0x6D63c734cfF8b067E8D34752A59EEA1eB93a8e97".
- **La clave privada de la cuenta:** Es la clave que desbloquea la cuenta anterior. Esta clave nunca debe ser revelada, puesto que en caso contrario cualquiera podría acceder y operar con ella.
- **El ABI del contrato:** Cada contrato tiene un ABI asociado, esto es una traducción del *smart contract* en formato *JavaScript Object Notation* (JSON), el formato entendido por la librería Web3. Por lo tanto, para poder interactuar con el contrato a través de Web3 es necesario proporcionar el ABI del contrato. En este caso el ABI se especifica en un archivo denominado "NMContract.json".

La librería Web3.js dispone de varios módulos para interactuar con la red de Ethereum, pero en este caso el que se va a utilizar es el módulo *"eth"* ya que es el que contiene toda la funcionalidad referida a la interacción con las cuentas y *smart contracts* [36]. Este módulo contiene la implementación del objeto "*contract"* a través del cual se realizará la interacción con los contratos inteligentes como si se tratase de objetos JavaScript. La funcionalidad más importante quizás sea la que genera un objeto de tipo transacción, y en la versión para Python se define de la siguiente manera:

```
myContract.functions.myMethod(args).buildTransaction(transaction)
```

Opcionalmente, se puede especificar el gas estimado y un *nonce* que servirá como diccionario de transacciones.

Las operaciones principales que realiza este módulo son por orden:

- Crear una instancia de web3 conectándose al nodo Geth que ejecuta en esa dirección y puerto asociados:

  ```
  web3 = Web3(Web3.HTTPProvider("http://127.0.0.1:8545"))
  ```

- Leer el ABI del contrato.

- Instanciar un objeto de tipo s*mart contract* especificando su dirección y su ABI:

```
contract = web3.eth.contract(address=address, abi=abi)
```

- Desbloqueo de la cuenta a través de la clave privada para poder operar con ella:

```
senderAccount = web3.eth.account.privateKeyToAccount(privateKey)
```

- Por cada tupla IP-Consumo, generar y enviar una transacción a Ethereum:
    - Estimación del gas consumido en la transacción:

```
estimateGas=contract.functions.setUserConsumption(ip, consumption).estimateGas()
```

    - Generación de la transacción. Como se puede observar se llama al método "*setUserConsumption"* del *smart contract* y se le pasan los argumentos necesarios, en este caso IP y consumo:

```
tx=contract.functions.setUserConsumption(ip, consumption).buildTransaction({'gas': estimateGas, 'nonce': web3.eth.getTransactionCount(senderAccount.address)})
```

    - Firma de la transacción para identificar la cuenta desde la que se realiza:

```
signed=web3.eth.account.signTransaction(tx, senderAccount.privateKey)
```

    - Envío de la transacción:

```
txHash = web3.eth.sendRawTransaction(signed.rawTransaction)
```

6.4 Smart contract: Network Monitoring

La primera línea de un *smart contract* debe especificar la versión de Solidity con la que se compilará el programa, en este caso se utilizó la versión 0.5.7.

```
pragma solidity ^0.5.7;

contract NetworkMonitoring{

    address public owner;
    string public name;
    uint128 networkBandwidth;
    uint128 totalConsumption;
    mapping (string => uint128) private _userConsumption;

    constructor(string memory _name, uint128 _networkBandwidth) public{
        owner = msg.sender;
        name = _name;
        networkBandwidth = _networkBandwidth;
    }
```

Ilustración 28. Variables y constructor del contrato Network monitoring

A continuación, empieza la declaración del *smart contract* como tal, seguida de las variables que va a necesitar y el constructor del mismo. En este caso, las variables utilizadas son:

- ***Owner:*** Se identifica como la cuenta de Ethereum que va a desplegar el contrato.
- ***Name:*** Nombre del contrato.
- ***networkBandwidth:*** Entero sin signo de 128 bits que representa el ancho de banda total ofrecido por la red en bytes.
- ***totalConsumption:*** Entero sin signo de 128 bits que representa el consumo total de la red en bytes.
- ***userConsumption:*** Estructura de tipo *mapping* que relaciona la IP (string) de cada dispositivo conectado a la red con el consumo (uint128) que está realizando.

En el constructor se definen los valores de las variables *owner, name* y *networkBandwidth.*

A continuación, se definen las funciones del contrato encargadas de gestionar el registro de información:

- ***setTotalConsumption***: Establece el consumo total que todos los usuarios están generando en la red.
- ***getTotalConsumption*:** Devuelve el consumo total.

```
function setTotalConsumption(uint128 _consumption) public{
    totalConsumption = _consumption;
}

function getTotalConsumption() public view returns (uint128){
    return totalConsumption;
}
```

Ilustración 29. Registro del consumo total del contrato Network Monitor

- ***setUserConsumption:*** Establece el consumo de cada IP de los dispositivos que están conectados a Internet a través del *gateway.*
- ***getUserConsumption:*** Especificando la IP del dispositivo, devuelve el consumo asociado.

```
function setUserConsumption(string memory _ipAddress, uint128 _consumption) public{
    _userConsumption[_ipAddress] = _consumption;
}

function getUserConsumption(string memory _ipAddress) public view returns (uint128){
    return _userConsumption[_ipAddress];
}
```

Ilustración 30. Registro del consumo de usuario del contrato Network Monitor

6.5 Smart contract: Internet Token

Es un token ERC-20 al que se le han añadido ciertas variables y un constructor para poder definir sus parámetros.

```
pragma solidity ^0.5.7;

import "github.com/OpenZeppelin/openzeppelin-solidity/contracts/token/ERC20/ERC20.sol";

contract InternetToken is ERC20 {
    address public owner;
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 private _totalSupply;
    mapping (address => uint256) private balances;

    constructor(string memory _name, string memory _symbol, uint8 _decimals, uint128 _initialTotalSupply) public{
        owner = msg.sender;
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        _totalSupply = _initialTotalSupply;
        balances[owner] = _totalSupply;
        emit Transfer(address(0), owner, _totalSupply);
    }
```

Ilustración 31. Variables y constructor del contrato Internet Token

En la segunda línea se puede observar que se especifica que el contrato es de tipo ERC20, lo que quiere decir que implementa todas sus funciones. Las variables que han sido añadidas y que se inicializan en el constructor son:

- ***owner:*** Se identifica como la cuenta de Ethereum que va a desplegar el contrato.
- ***name:*** Nombre del token.
- ***symbol:*** Símbolo utilizado para el token.
- ***decimals:*** Número de decimales que va a tener el token.
- ***totalSupply:*** Suministro de tokens que va a ser lanzado en el despliegue.
- ***balances:*** Estructura de tipo *mapping* (address, uint256) para identificar los balances correspondientes a cada cuenta que disponga de tokens.

Las funciones más destacables y de mayor importancia de este s*mart contract* para la implementación de la solución se definen a continuación:

```
function totalSupply() public view returns (uint256) {
        return _totalSupply;
}

function balanceOf(address _tokenOwner) public view returns (uint256) {
    return balances[_tokenOwner];
}

function transfer(address _to, uint256 _value) public returns (bool) {
    require(_value <= balances[msg.sender]);
    require(_to != address(0));

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
}
```

Ilustración 32. Funciones principales del contrato Internet Token

- **totalSupply():** Devuelve el suministro total de tokens en circulación.
- **balanceOf():** Devuelve el balance asociado a una cuenta.
- **transfer():** Transfiere una cantidad de tokens a una cuenta específica.

6.6 Compilación y despliegue de los *smart contracts*

Con la lógica de los contratos implementada, para interactuar con ellos es necesario compilarlos y desplegarlos en la red. Como ya se explicó anteriormente, es

posible hacerlo en una red de pruebas local como *Ganache* por ejemplo, pero es recomendable hacerlo en una *testnet* pública ya que es la que más se asemeja a la red principal o *mainnet* en todos los aspectos, desde las tasas de transferencias, hasta el rendimiento de la red.

Es posible compilar y desplegar los contratos vía línea de comandos a través de la consola de *web3.js* pero en este caso se hará uso del IDE Remix, puesto que ya proporciona un ecosistema de desarrollo online con diferentes funcionalidades y además, dispone de integración con el *wallet* de Metamask, eso sí, debe estar instalado en el navegador.

Se va a realizar la explicación tomando como ejemplo el contrato *Network Monitoring* ya que el proceso es análogo para todos los *smart contracts.*

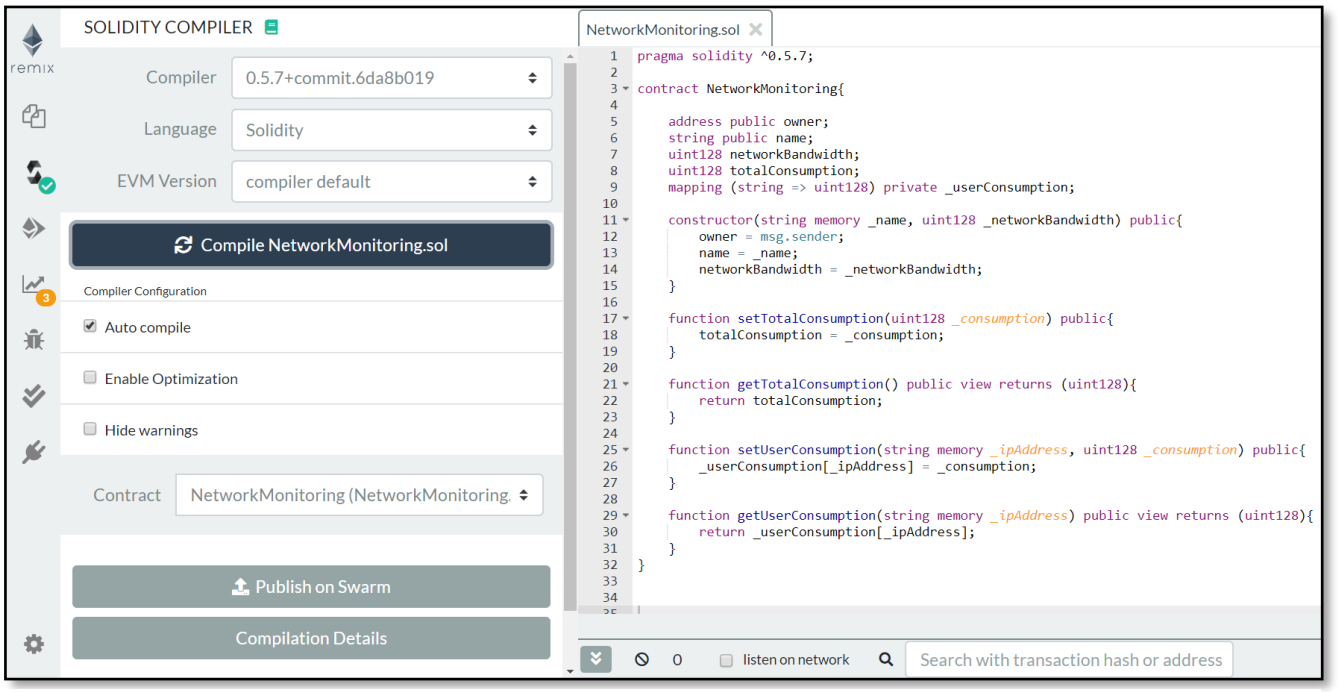


Ilustración 33. Compilación de smart contract en Remix

Se debe seleccionar la versión del compilador y de la EVM que se va a utilizar. Si no devuelve ningún error, significa que el código se ha compilado correctamente.

Ahora solo queda realizar el despliegue del contrato en la red especificando los parámetros que necesita el constructor, que en este caso son el nombre del contrato y el ancho de banda que va a ofrecer el *gateway.* También hay que especificar el entorno (web3) y la cuenta de Ethereum con la que se va a realizar el despliegue (la mía personal en este caso).

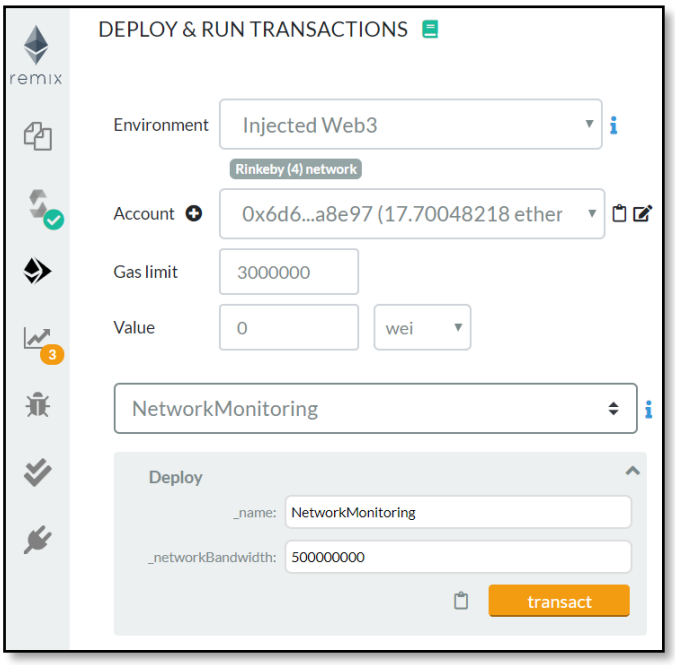


Ilustración 34. Despliegue de smart contract en Remix

Una vez se pulsa el botón *"transact"* se genera la transacción de despliegue y se abre una ventana de *Metamask* en la que se muestra el gas que se va a gastar en realizar la transacción de despliegue que en este caso se corresponde con 0.000556 Ether.

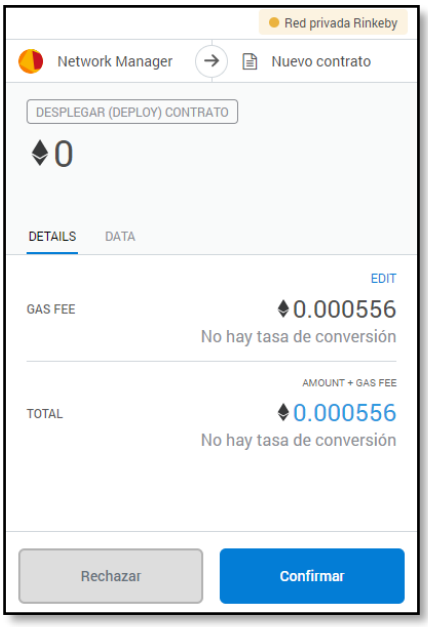


Ilustración 35. Confirmación de la transacción de despliegue en Metamask

Si se confirma la transacción, es enviada y se carga el coste a la cuenta especificada. Una vez se verifique el bloque que incluye la transacción el contrato se encuentra desplegado y puede confirmarse en un explorador de bloques como *Etherscan* introduciendo el *hash* de la transacción

(https://rinkeby.etherscan.io/tx/0xd3bf4423a8cec008da88fbcb2c2c341af4c460ab2525c60abccd07facbb57c7f).

Como se puede observar en la siguiente imagen, una vez desplegado, es posible interactuar con el contrato desde Remix invocando sus funciones. En este caso la invocación se realizará desde el *Ethereum Registry Monitor* (funciones de escritura, tienen tasa asociada y aparecen en naranja) y desde el *back-end* del servidor (funciones de lectura, no tienen tasa asociada y aparecen en azul).

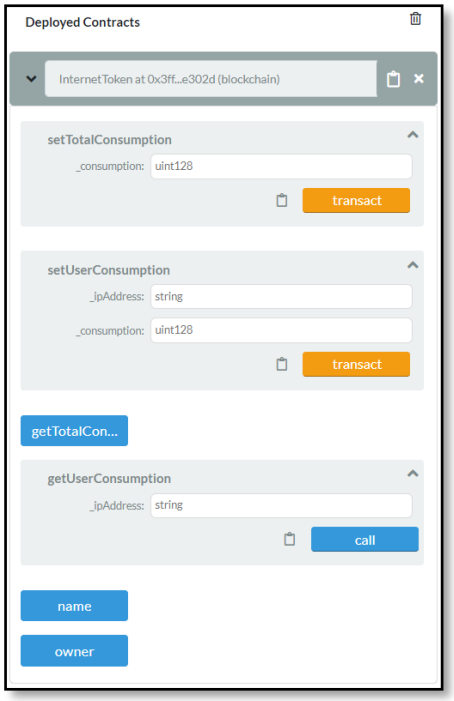


Ilustración 36. Funciones del smart contract desplegado en Remix

6.7 Aplicación web

Tal y como se indicó en el apartado de diseño, la aplicación cuenta con dos módulos, el servidor (*back-end*) y una interfaz de usuario (*front-end*)

Servidor (*back-end*)

Toda la lógica la implementa un controlador denominado *controller.js* y su función es leer la información a través de la interacción con el contrato inteligente que el *Ethereum Registry Monitor* almacenó en Ethereum y aplicar la lógica de la política definida para automatizar la gestión de la red y facturar el servicio.

Al tratarse de programación en JavaScript es necesario utilizar la versión de Web3.js correspondiente. Este controlador realizará invocaciones a ambos contratos, tanto al *Network Monitoring* para leer la información del *gateway* que da servicio como al *Internet Token* para facturar el servicio.

De la misma manera que en el caso del *Ethereum Registry Monitor,* hay que crear una instancia de Web3.js para poder hacer uso del módulo *"Contract"* con el que interactuar con los contratos.

```
var web3 = new Web3( new Web3.providers.HttpProvider("http://127.0.0.1:8545"))
exports.web3 = web3
web3.eth.defaultAccount = "0x6D63c734cfF8b067E8D34752A59EEA1eB93a8e97"

var logger = require('../../logger').logger.nta
const NMAbi = JSON.parse(config.ethereum.networMonitor.ABI)
const ERC20Abi = JSON.parse(config.ethereum.ERC20.ABI)
const NMContract = new web3.eth.Contract(NMAbi, config.ethereum.networMonitor.Address)
const ERC20Contract = new web3.eth.Contract(ERC20Abi, config.ethereum.ERC20.Address)
```

Ilustración 37. Variables del controller

Para ello se especifican los ABIs de ambos contratos y se instancian. También se proporciona un *logger* para tener un control visual de los resultados de las llamadas al servidor.

Es necesario que la aplicación tenga integración con *Metamask* para que el usuario que está realizando una transacción disfrute de una interacción visual a través de una ventana de login en la que da el permiso de utilización de sus cuentas por parte del sistema y acepta o rechaza la transacción. Para integrar *Metamask* es necesario indicar la siguiente instrucción:

```
// Metamask login window. Gets all accounts
const metamaskAccounts = await window.ethereum.enable()
```

Ilustración 38. Integración del controller con Metamask

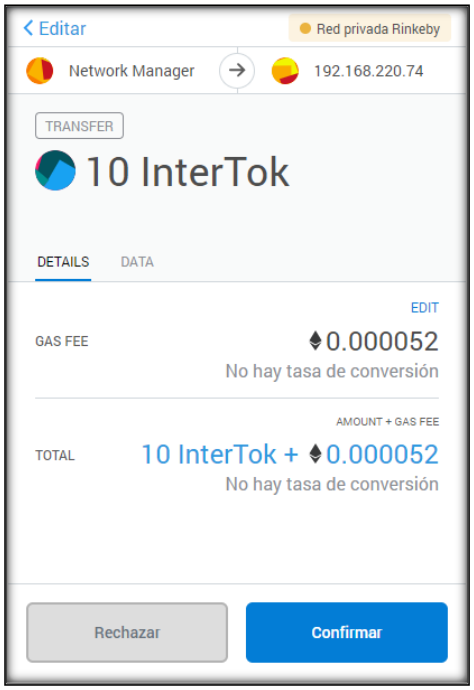


Ilustración 39. Ejemplo de confirmación de transacción desde Metamask

A continuación, se explican todas las funciones implementadas en el *controller.js*:

```
// gets the total network consumption
exports.getTotalConsumption = function (req, res) {
    logger.info('[GET-TOTAL-CONSUMPTION] Getting total network consumption...')
    NMContract.methods.getTotalConsumption().call().then(result => {
        logger.info(`[GET-TOTAL-CONSUMPTION] Smart Contract called successfully`)
        res.status(200).send({output: web3.utils.hexToNumber(result)})
    }).catch(err => {
        logger.error('[GET-TOTAL-CONSUMPTION] ' + err.message)
        res.status(500).send({ error: '[GET-TOTAL-CONSUMPTION] ' +  err.message })
    })
}
```

Ilustración 40. Función getTotalConsumption

getTotalConsumption invoca al método con el mismo nombre del contrato *Network Monitoring* para obtener el total del consumo de ancho de banda que están realizando todos los dispositivos conectados.

```
// gets percentage of network ocupation
exports.getNetworkOcupation = function (req, res) {
    logger.info('[NETWORK-OCUPATION] Getting total network consumption...')
    NMContract.methods.getTotalConsumption().call().then(result => {
        logger.info(`[NETWORK-OCUPATION] Smart Contract called successfully`)
        var totalConsumption = web3.utils.hexToNumber(result)
        var networkOcupation = Math.round((totalConsumption/config.networkBandwidth) * 100) / 100
        res.status(200).send({networkOcupation: networkOcupation})
    }).catch(err => {
        logger.error('[NETWORK-OCUPATION] ' + err.message)
        res.status(500).send({ error: '[NETWORK-OCUPATION] ' +  err.message })
    })
}
```

Ilustración 41. Función getNetworkOcupation

getNetworkOcupation calcula el porcentaje de ocupación de la red en base al consumo total.

```
// gets network consumption by user
exports.getUserConsumption = function (req, res) {
    logger.info('[GET-USER-CONSUMPTION] Getting total network consumption...')
    NMContract.methods.getUserConsumption(req.params.userIp).call().then(result => {
        logger.info(`[GET-USER-CONSUMPTION] Smart Contract called successfully`)
        res.status(200).send({output: web3.utils.hexToNumber(result)})
    }).catch(err => {
        logger.error('[GET-USER-CONSUMPTION] ' + err.message)
        res.status(500).send({ error: '[GET-USER-CONSUMPTION] ' +  err.message })
    })
}
```

Ilustración 42. Función getUserConsumption

getUserConsumption invoca al método con el mismo nombre del contrato *Network Monitoring* para obtener el consumo de ancho de banda que están realizando la IP del dispositivo que se proporciona por parámetro.

```
//  get the network access price
exports.accessPrice = function (req, res) {
    logger.info('[ACCESS-PRICE] Getting total network consumption...')
    NMContract.methods.getTotalConsumption().call().then(result => {
        logger.info(`[ACCESS-PRICE] Smart Contract called successfully`)
        // set the access price
        var accessPrice = Math.round(1/(1-(totalConsumption/config.networkBandwidth)) * 100) / 100
        res.status(200).send({accessPrice: accessPrice})
    }).catch(err => {
        logger.error('[ACCESS-PRICE] ' + err.message)
        res.status(500).send({ error: '[ACCESS-PRICE] ' +  err.message })
    })
}
```

Ilustración 43. Función accessPrice

accessPrice calcula el precio en tokens que un usuario debe pagar para conectarse al *gateway* y disfrutar de conexión a Internet. El precio se calcula en base a la ocupación de la red, de manera que cuanto mayor sea, mayor será el precio. La fórmula utilizada es:

$$Base\ Price = \frac{1}{1 - (totalConsumption/networkBandwidth)}$$

No es suficiente solo con calcular este valor, porque en el caso de que varios dispositivos soliciten el servicio a la vez, se valorará la selección teniendo en cuenta otro criterio para la selección: la oferta del usuario. De manera que se selecciona aquellos dispositivos que con mayor valor de:

$$Access\ Price = Base\ Price + User\ Offer$$

Por ejemplo, si el *base price* está en 10 tokens y hay dos usuarios, uno que oferta 1 token y otro que oferta 2 tokens, se seleccionará el que oferta 2 tokens.

```
// calculates the cost for an user and automates the invoicing
exports.costPerUser = function (req, res) {
    logger.info('[COST-PER-USER] Getting consumption for user ' + req.params.userIp)
    userIp = req.params.userIp
    NMContract.methods.getUserConsumption(userIp).call().then(result => {
        logger.info(`[COST-PER-USER] Smart Contract called successfully`)
        var userConsumption = web3.utils.hexToNumber(result)
        var userOcupation = Math.round((userConsumption/config.networkBandwidth) * 100) / 100
        var costPerUser = 0
        if(userOcupation < 0.1){
            costPerUser = 1
        }else if (userOcupation >= 0.5 && userOcupation <= 0.75){
            costPerUser = Math.round(((userOcupation * 1.5)) * 100) / 100
        }else if (userOcupation >= 0.75 && userOcupation <= 0.90){
            costPerUser = Math.round(((userOcupation * 2)) * 100) / 100
        }else if (userOcupation >= 0.90){
            costPerUser = Math.round(((userOcupation * 4)) * 100) / 100
        }else{
            costPerUser = Math.round((userOcupation) * 100) / 100
        }
        logger.info('[COST-PER-USER] Unlocking account to transfer network tokens...')
        web3.eth.personal.unlockAccount(config.devices_address[userIp], "password", 0x000000000000000000000000000014)
        .then(result => {
            logger.info('[COST-PER-USER] Unlocked account?: ' + result)
            ERC20Contract.methods.transfer(config.ethereum.account, costPerUser).send({
                from: config.devices_address[userIp], gasPrice: '3000000', gas: '4000000'
            })
            .on('transactionHash', (tx) => {
                logger.info(`[COST-PER-USER] Smart Contract called successfully`)
                res.status(200).send({costPerUser: costPerUser, transferTransaction: tx})
            })
            .on('error', (err) => {
            })
        }).catch(err => {
            logger.error('[COST-PER-USER] Error unlocking the account' + err)
            res.status(500).send({error: '[COST-PER-USER] Error unlocking the account' + err.message})
        })
    }).catch(err => {
        logger.error('[COST-PER-USER] ' + err.message)
        res.status(500).send({ error: '[COST-PER-USER] ' +  err.message })
    })
}
```

Ilustración 44. Función costPerUser

costPerUser calcula la facturación que se le debe aplicar a cada usuario en función del consumo de ancho de banda que están realizando para automatizar las transferencias de tokens hacia la cuenta del proveedor de servicios.

Aquí se ha definido una política de facturación sencilla, pero podría definirse la que más conviniese:

- Si el consumo del usuario es inferior al 10% de la red, se le factura 1 token.
- Si el consumo del usuario es entre el 50% y el 75%, se le multiplica un recargo de 1.5 tokens.
- Si el consumo del usuario es entre el 75% y el 90%, se le multiplica un recargo de 2 tokens.
- Si la ocupación del usuario es mayor del 90% al 10%, se le multiplica un recargo de 4 tokens.

Una vez se establece la tarifa de facturación del usuario, se procede a realizar la trasferencia desde la dirección de la cuenta asociada a la dirección IP hacia la cuenta del proveedor de servicio. Para ello se desbloquea la cuenta a través de la contraseña mediante la instrucción:

```
web3.eth.personal.unlockAccount(config.devices_address[userIp], "password").
```

A continuación, se invoca a la función *transfer* del contrato Internet Token para enviar los fondos correspondientes a la tarifa aplicada desde la cuenta del dispositivo hacia la cuenta del proveedor de servicio y se espera por la respuesta de confirmación:

```
ERC20Contract.methods.transfer(config.ethereum.account, costPerUser).send({from: config.devices_address[userIp], gasPrice: '3000000', gas: '4000000'})
```

En este caso, se especifican unos valores de *gasPrice y gas* suficientes para que la transacción se haga efectiva.

```
// gets the token balance for an user
exports.userBalance = function (req, res) {
    logger.info('[USER-BALANCE] Getting user balance...')
    userIp = req.params.userIp
    ERC20Contract.methods.balanceOf(config.devices_address[userIp]).call().then(result => {
        result = result / 10
        logger.info('[USER-BALANCE] User balance: ' + result)
        res.status(200).send({userBalance: result})
    }).catch(err => {
        logger.error('[USER-BALANCE] ' + err)
        res.status(500).send({error: '[USER-BALANCE] ' + err.message})
    })
}
```

Ilustración 45. Función userBalance

Por último, el método *userBalance* devuelve el balance de la dirección asociada con la IP del dispositivo que se especifica por parámetro llamando al método *balanceOf* del *smart contract* Internet Token.

El servidor es ejecutado en local en el puerto 9092 y todas las funciones anteriores son expuestas en diferentes rutas o *paths* a partir de la raíz: *http://localhost:9092.* Por ejemplo, para invocar a la función *userBalance* hay que hacer una petición *a* http://localhost:9092/userBalance/:userIP donde *"userIP"* es la dirección IP del usuario del que se quiere conocer el balance. Todas las rutas se especifican en el archivo *routes.js:*

```
module.exports = function (app) {
    var controller = require('../controllers/controller')

    /* Index route */
    app.get('/', function (req, res) {
        res.send(' <h1 style="font-size:300%;"> Network Tokenized Access! </h1>')
    })

    app.get('/totalConsumption', controller.getTotalConsumption)
    app.get('/userConsumption/:userIP', controller.getUserConsumption)
    app.get('/networkOcupation', controller.getNetworkOcupation)
    app.get('/accessPrice', controller.accessPrice)
    app.get('/userConsumption/:userIP', controller.getUserConsumption)
    app.get('/userCost/:userIP', controller.costPerUser)
    app.get('/userBalance/:userIP', controller.userBalance)
```

Ilustración 46. Rutas de las funciones en el controller

Interfaz de usuario (*front-end*)

Como se adelantó en el apartado de diseño, la interfaz de usuario se va a desarrollar utilizando la biblioteca de React.js. Recordando el diseño, la interfaz se compone de dos pantallas, una de inicio que presenta la aplicación, y otra principal en la que se muestra toda la información del sistema.

Existen cuatro componentes principales encargados de visualizar las pantallas de los cuales dos de ellos, son subcomponentes de uno.

Welcome component

Denominado *Home.js,* se trata del componente que carga la pantalla de inicio por la que se accede a la aplicación. Presenta un mensaje de bienvenida junto con el icono de la presentación y un botón de acceso a la página principal.

Main component

Llamado *Main.js* en la práctica, se encarga de visualizar tanto el componente *NetworkMonitorPanel.js,* que muestra la información correspondiente con el estado del servicio, como el componente *UserPanel.js,* que muestra la oferta del usuario para entrar a la red y permite realizar la petición de acceso al servicio. Por lo tanto, se trata de un componente padre que tiene dos componentes hijos a los que les envía las propiedades para que sean utilizadas en el nivel inferior.

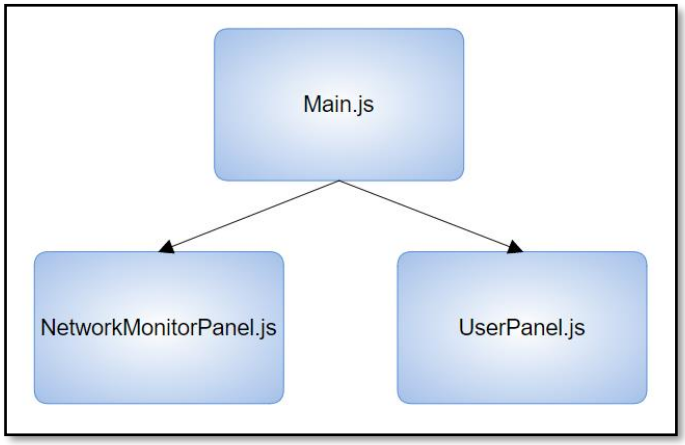


Ilustración 47. Relación jerárquica padre-hijos entre componentes

Las propiedades que define el componente son:

```
export default class Main extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            totalConsumption: "",
            networkOcupation: "",
            accessPrice: "",
            userConsumption: "",
            userCost: "",
            userBalance: ""
        }
    }
```

Ilustración 48. Propiedades del componente main.js

Este componente llama a las funciones que contienen la lógica del sistema y que se encuentran implementadas en el servidor (*back-end*). Las funciones son invocadas a través de peticiones http y para ello, es necesario importar en este componente la librería *"axios"* un cliente ligero que proporciona una forma sencilla de enviar peticiones.

A continuación, se muestra una llamada a la función del *getTotalConsumption* del *back-end,* pero es análoga a todas las demás:

```
async getTotalConsumption() {
    axios.get(`${config.url_local_request}/totalConsumption`)
        .then(res => {
            if (res.error == undefined) {
                this.setState({
                    totalConsumption: `The network total consume is ${res.data.totalConsumption} MBytes`
                })
                console.log(this.state)
            }
        }).catch(err => console.log(err))
}
```

Ilustración 49. Invocación a la función getTotalConsumption desde el front-end

El estilo de todos los componentes se define en el archivo *App.css* en el que se definen todas las clases existentes.

Capítulo 7: Resultados

Los resultados son la comprobación de una correcta implementación. Indicarán si se han cubierto los objetivos principales y darán paso a nuevas ideas para futuras iteraciones de la solución. En este capítulo se muestra una batería de resultados que validan el funcionamiento de todos los módulos del sistema.

Con el prototipo implementado de manera completamente funcional, se muestran a lo largo de todo el capítulo los resultados obtenidos. Primero se mostrará el funcionamiento de los diferentes módulos que conforman la solución para finalmente, mostrar la aplicación web que se encarga de orquestar todo el sistema.

7.1 Nodo Geth en acción

Una vez instalado y configurado el cliente de Ethereum como se detalla en el Anexo A: Configuración de Raspberry Pi 3B+ como gateway de acceso inalámbrico a red, se procedió al arranque del mismo como se puede observar en la siguiente imagen:

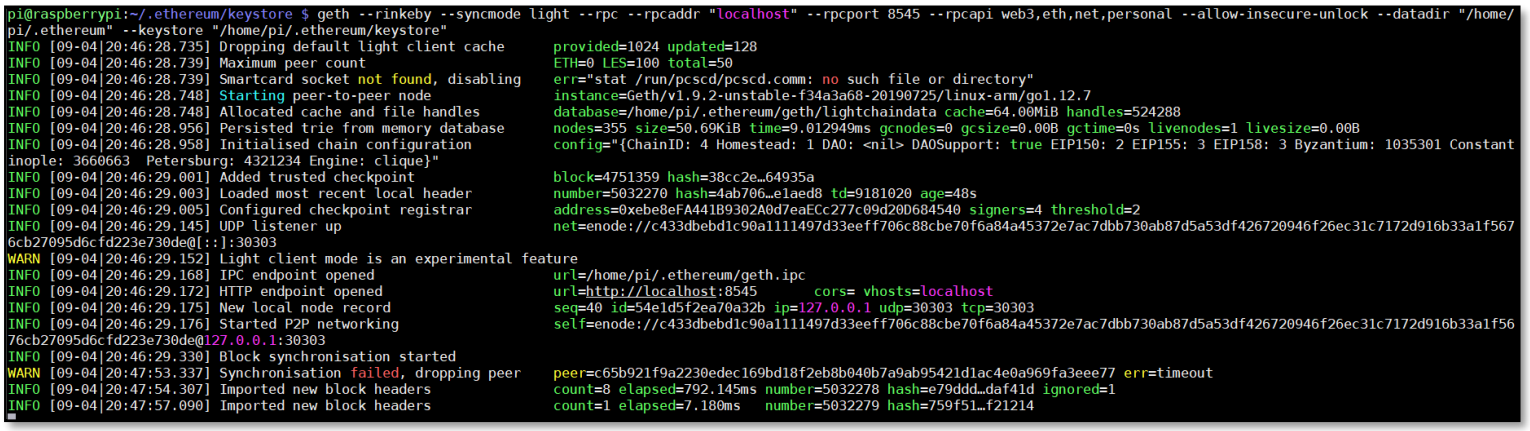


Ilustración 50. Arranque nodo Geth en Raspberry Pi

Al ser la puerta de comunicación con los *smart contracts* desplegados en la red, se puede observar en el *log* del nodo la llegada de una transacción generada, por ejemplo, por el monitor de registro que vuelca la información en Ethereum a través del correspondiente contrato.

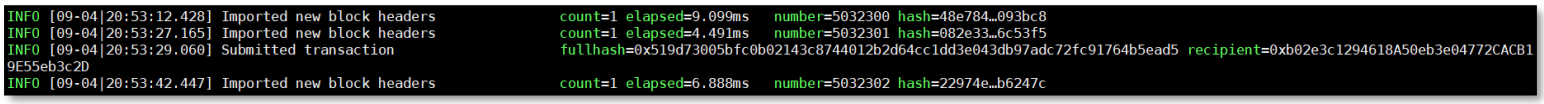


Ilustración 51. Gestión de una transacción por parte del cliente de Ethereum

Como se puede observar la transacción se identifica por el hash de la misma (*fullhash*) y el contrato al que se dirige (*recipient*).

7.2 Network Monitor y Ethereum Registry Monitor en acción

Recordando el funcionamiento de ambos módulos, *Network Monitor* se ejecuta de manera indefinida para monitorizar la red en tiempo real, esto es, gestionar el acceso al servicio por parte de los usuarios y obtener el consumo de ancho de banda que consume cada uno de ellos. Una vez dispone de la información necesaria, llama al

Ethereum Registry Monitor para que vuelque la información en Ethereum a través del contrato *Network Monitoring.* A continuación, se muestra un ejemplo de ejecución en el que se escanea la red durante 30 segundos y se realizan las transacciones de registro en Ethereum.

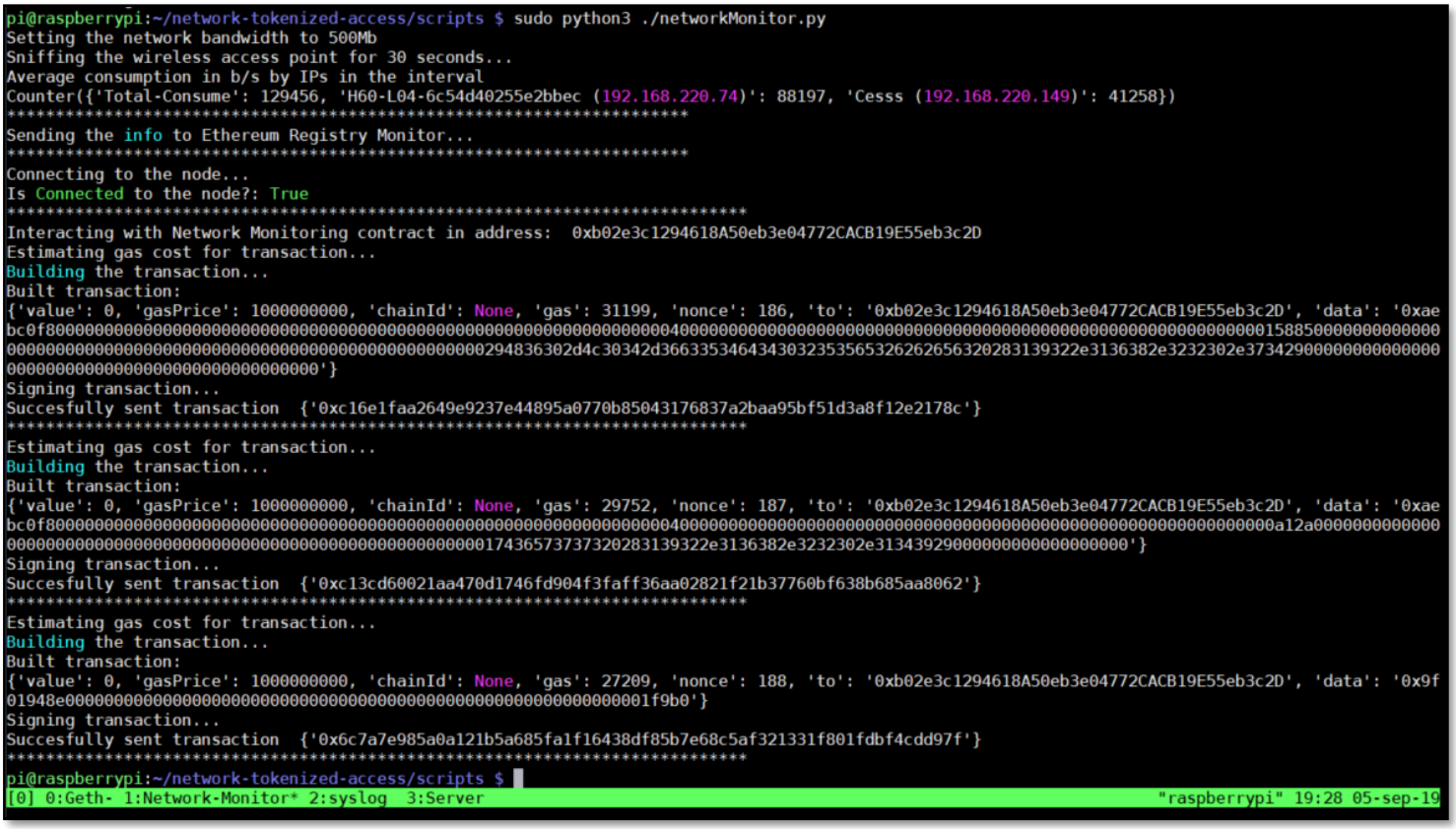


Ilustración 52. Log de ejecución de Network Monitor

Como se puede observar en la imagen, hay dos dispositivos conectados al *gateway* y que están consumiendo ancho de banda: *'H60-L04-6c54d40255e2bbec (192.168.220.74)'* ha consumido una media de 88197 bytes por segundo y *'Cesss (192.168.220.149)'* 41258 bytes por segundo. Cuando llega la información al *Ethereum Registry Monitor* realiza una petición por cada dispositivo para almacenar en Ethereum su correspondiente consumo, lo mismo hace con el consumo total.

7.3 Back-end en acción

El *back-end* encargado de leer la información de la red almacenada en Ethereum y hacer los cálculos necesarios, se ejecuta también de forma permanente, quedando a la espera de las peticiones que llegan desde el *front-end* para obtener datos.

A continuación, se muestra un pantallazo del *log* en el que se pueden observar las *request* provenientes de la interfaz de usuario.

```
pi@raspberrypi:~/network-tokenized-access/app/src/backend/controllers $ npm run dev
npm WARN npm npm does not support Node.js v10.15.2
npm WARN npm You should probably upgrade to a newer version of node as we
npm WARN npm can't make any promises that npm will work with this version.
npm WARN npm Supported releases of Node.js are the latest release of 4, 6, 7, 8, 9.
npm WARN npm You can find the latest version at https://nodejs.org/

> nta@0.1.0 dev /home/pi/network-tokenized-access/app
> nodemon ./src/server/nta.js

[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node ./src/server/nta.js`
Server listening HTTP on port 9092
2019-09-05T17:51:57.470Z [info]: [NETWORK-TOKENIZED-ACCESS] - [GET-TOTAL-CONSUMPTION] Getting total network consumption...
2019-09-05T17:51:57.907Z [info]: [NETWORK-TOKENIZED-ACCESS] - [GET-TOTAL-CONSUMPTION] Smart Contract called successfully
2019-09-05T17:51:59.728Z [info]: [NETWORK-TOKENIZED-ACCESS] - [NETWORK-OCUPATION] Getting total network consumption...
2019-09-05T17:51:59.763Z [info]: [NETWORK-TOKENIZED-ACCESS] - [NETWORK-OCUPATION] Smart Contract called successfully
2019-09-05T17:52:01.816Z [info]: [NETWORK-TOKENIZED-ACCESS] - [GET-USER-CONSUMPTION] Getting total network consumption...
2019-09-05T17:52:01.871Z [info]: [NETWORK-TOKENIZED-ACCESS] - [GET-USER-CONSUMPTION] Smart Contract called successfully
2019-09-05T17:52:04.256Z [info]: [NETWORK-TOKENIZED-ACCESS] - [ACCESS-PRICE] Getting total network consumption...
2019-09-05T17:52:04.292Z [info]: [NETWORK-TOKENIZED-ACCESS] - [ACCESS-PRICE] Smart Contract called successfully
2019-09-05T17:52:06.952Z [info]: [NETWORK-TOKENIZED-ACCESS] - [COST-PER-USER] Getting consumption for user H60-L04-6c54d40255e2bbec (192.168.220.74)
2019-09-05T17:52:07.169Z [info]: [NETWORK-TOKENIZED-ACCESS] - [COST-PER-USER] Smart Contract called successfully
2019-09-05T17:52:07.171Z [info]: [NETWORK-TOKENIZED-ACCESS] - [COST-PER-USER] Unlocking account to transfer network tokens...
2019-09-05T17:52:08.968Z [info]: [NETWORK-TOKENIZED-ACCESS] - [USER-BALANCE] Getting user balance...
2019-09-05T17:52:09.162Z [info]: [NETWORK-TOKENIZED-ACCESS] - [USER-BALANCE] User balance: 8
2019-09-05T17:52:14.526Z [info]: [NETWORK-TOKENIZED-ACCESS] - [COST-PER-USER] Unlocked account?: true
2019-09-05T17:52:14.526Z [info]: [NETWORK-TOKENIZED-ACCESS] - [COST-PER-USER] Transfering 1.1 tokens to network manager...
2019-09-05T17:52:14.832Z [info]: [NETWORK-TOKENIZED-ACCESS] - [COST-PER-USER] Successfully transfer from 0x10eCbce9DDC5aC0B7DC39b1Ff7cF4aCE580353dd
to network manager
[0] 0:Geth  1:Network-Monitor  2:syslog-  3:Server*                                    "raspberrypi" 19:52 05-sep-19
```

Ilustración 53. Log del back-end

7.4 Interfaz de usuario en acción

Finalmente, y gracias al funcionamiento de todos los módulos que conforman el prototipo, se muestra la interfaz de usuario que se encarga de representar visualmente el funcionamiento del sistema.

La pantalla de inicio de la aplicación se muestra a continuación.

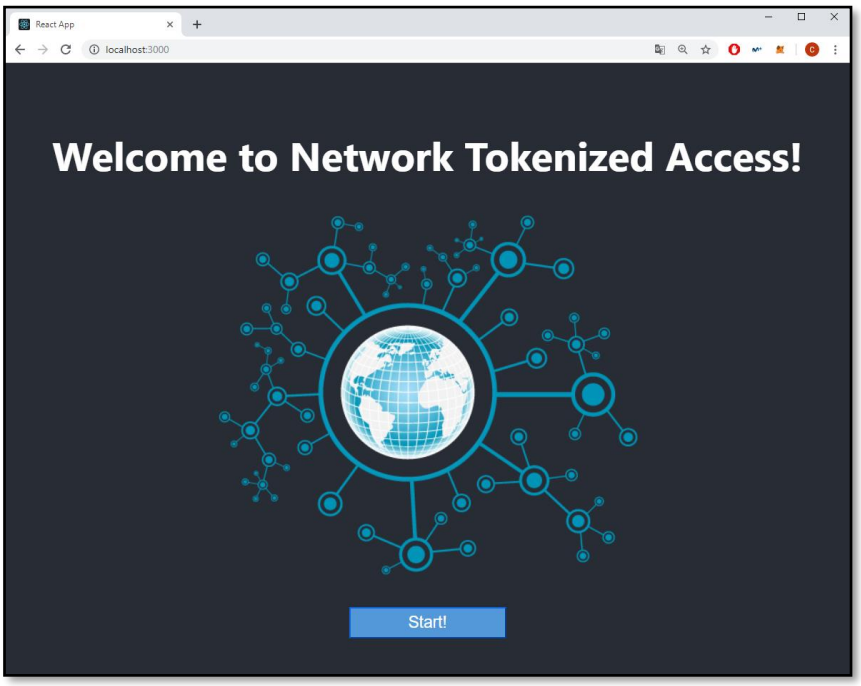


Ilustración 54. Pantalla de inicio de la aplicación

Una vez se accede a la pantalla principal, el resultado es el siguiente:

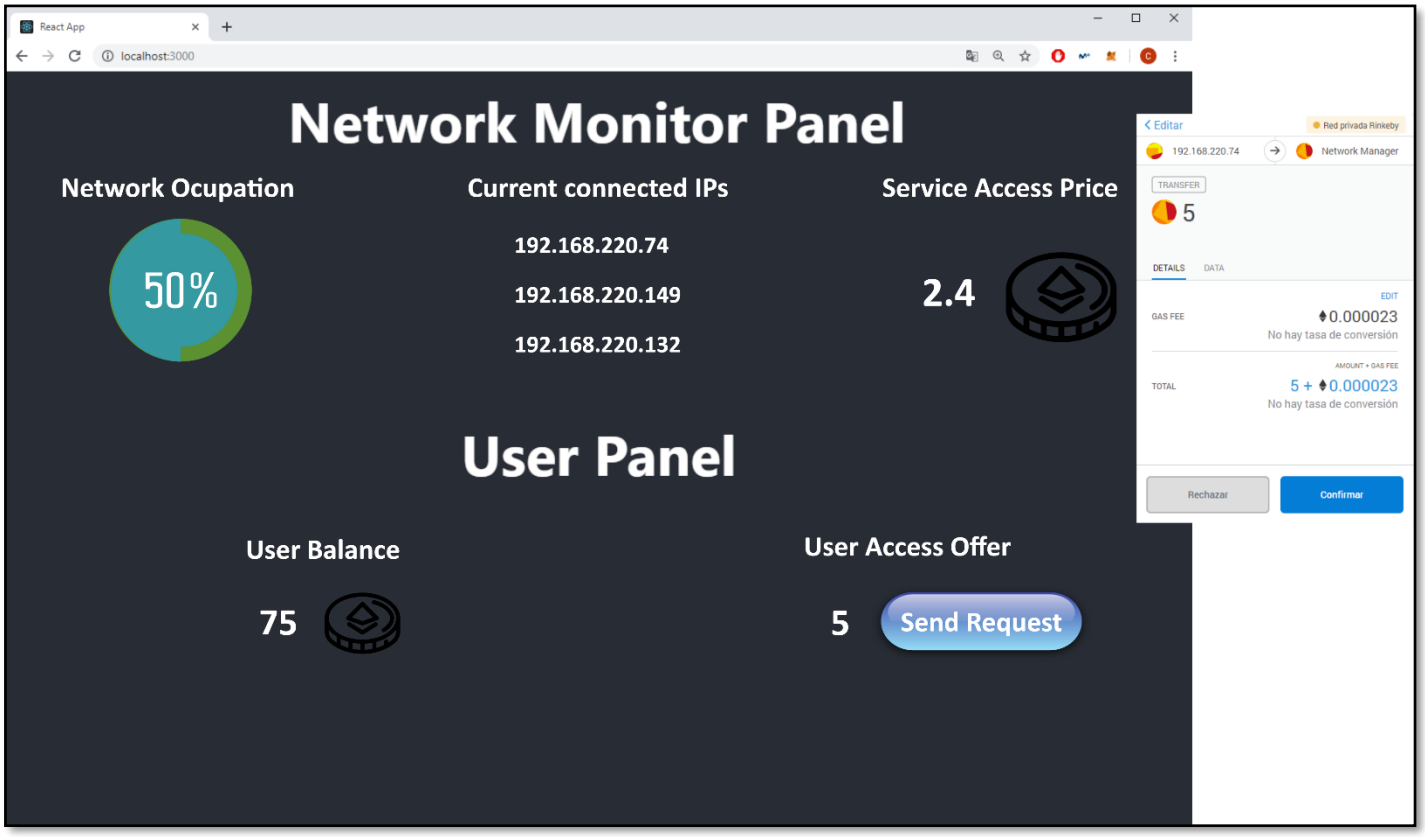


Ilustración 55. Pantalla principal de la aplicación

También se puede observar la integración con el wallet *Metamask* que representa de una forma sencilla y amigable las transacciones que se deben realizar para la facturación del servicio en tokens. En la presentación adjunta, se muestra un vídeo del funcionamiento completo de la interfaz de usuario.

Capítulo 8: Planificación

Uno de los aspectos clave en la supervivencia y éxito de un proyecto es su planificación. En este capítulo se muestra una planificación inicial en la que se estimaron los tiempos de trabajo junto con la dedicación real al proyecto.

En primer lugar y para conocer las dimensiones del proyecto, se realizó una estimación de los tiempos necesarios para desarrollar cada una de sus partes. Para lograr una correcta visualización del tiempo invertido en las actividades de la vida del proyecto se hace uso de diagramas de Gantt.

Las actividades identificadas al comienzo y que se han especificado en el diagrama son:

- **Proposición del proyecto:** Esta fase se corresponde con la búsqueda del objetivo general del proyecto y la presentación al tutor, finalizando con la aprobación de la idea por parte del mismo.
- **Estudio del estado del arte:** Se trata de la fase en la que se estudia el estado actual de la tecnología. Es posible dividir esta fase en otras dos:
  - Estudio del ecosistema Blockchain desde sus características principales hasta sus aplicaciones en el mundo actual.
  - Estudio profundo de la plataforma Ethereum.
- **Análisis:** Consiste en la proposición de un prototipo que incorpore la tecnología Blockchain y que valide el propósito del proyecto.
- **Diseño:** En diseño del prototipo se identifican por primera vez todos los módulos que forman el prototipo, así como las tecnologías que van a ser utilizadas. Se puede diferenciar en dos fases:
  - Estudio y selección de las tecnologías hardware utilizadas
  - Estudio y selección de las tecnologías software utilizadas
- **Implementación:** Es la materialización práctica del prototipo propuesto en la prueba de concepto. Consta de varias partes:
  - Preparación del entorno de trabajo.
  - Implementación del módulo *Network Monitor*.
  - Implementación del módulo *Ethereum Registry Monitor*.
  - Implementación de los *smart contracts*
  - Implementación de la aplicación web.
- **Generación de la documentación:** Elaboración de la memoria escrita que recoge toda la realización del proyecto.

El tiempo dedicado al proyecto ha sido aproximadamente de 3 meses comenzando el 1 de julio. El día 15 de agosto se tomó de descanso por ser fiesta nacional. Se trabajaron también los fines de semana.

Actividad	Fecha de Inicio	Fecha de Fin	Duración
Proposición del proyecto	01/07/2019	04/07/2019	5
Estudio del estado del arte	05/07/2019	20/07/2019	16
Estudio del ecosistema Blockchain	05/07/2019	12/07/2019	8
Estudio de la plataforma de Ethereum	13/07/2019	20/07/2019	8
Análisis	21/07/2019	25/07/2019	5
Diseño	26/07/2019	31/08/2019	6
Diseño hardware	26/07/2019	28/08/2019	3
Diseño software	29/08/2019	31/08/2019	3
Implementación	01/08/2019	10/08/2019	10
Preparación del entorno de trabajo	01/08/2019	02/08/2019	2
Implementación Network Monitor	03/08/2019	04/08/2019	2
Implementación Ethereum Registry	05/08/2019	06/08/2019	2
Implementación Smart Contracts	07/08/2019	08/08/2019	2
Implementación Aplicación Web	09/08/2019	10/08/2019	2
Documentación	11/08/2019	31/09/2019	20
TOTAL			**62**

Tabla 23. Planificación del proyecto

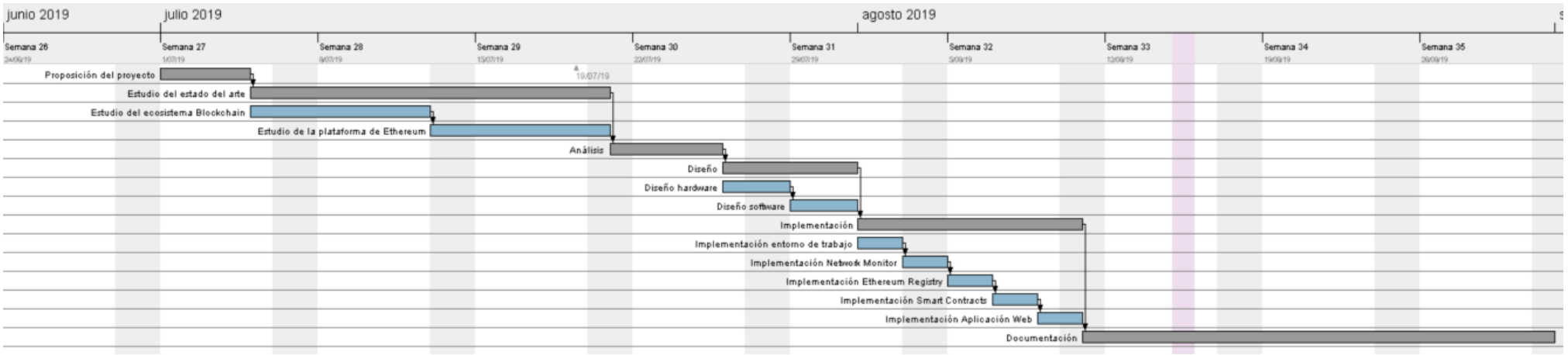


Ilustración 56. Diagrama de Gantt de planificación

Actividad	Fecha de Inicio	Fecha de Fin	Duración
Proposición del proyecto	01/07/2019	04/07/2019	5
Estudio del estado del arte	05/07/2019	24/07/2019	20
Estudio del ecosistema Blockchain	05/07/2019	12/07/2019	8
Estudio de la plataforma de Ethereum	13/07/2019	24/07/2019	12
Análisis	25/07/2019	27/07/2019	3
Diseño	28/07/2019	02/08/2019	6
Diseño hardware	28/07/2019	29/07/2019	2
Diseño software	30/07/2019	02/08/2019	4
Implementación	03/08/2019	14/08/2019	12
Preparación del entorno de trabajo	03/08/2019	04/08/2019	2
Implementación Network Monitor	05/08/2019	07/08/2019	3
Implementación Ethereum Registry	08/08/2019	10/08/2019	3
Implementación Smart Contracts	11/08/2019	12/08/2019	2
Implementación Aplicación Web	13/08/2019	14/08/2019	2
Documentación	16/08/2019	07/09/2019	23
TOTAL			**69**

Tabla 24. Tiempo real dedicado al proyecto

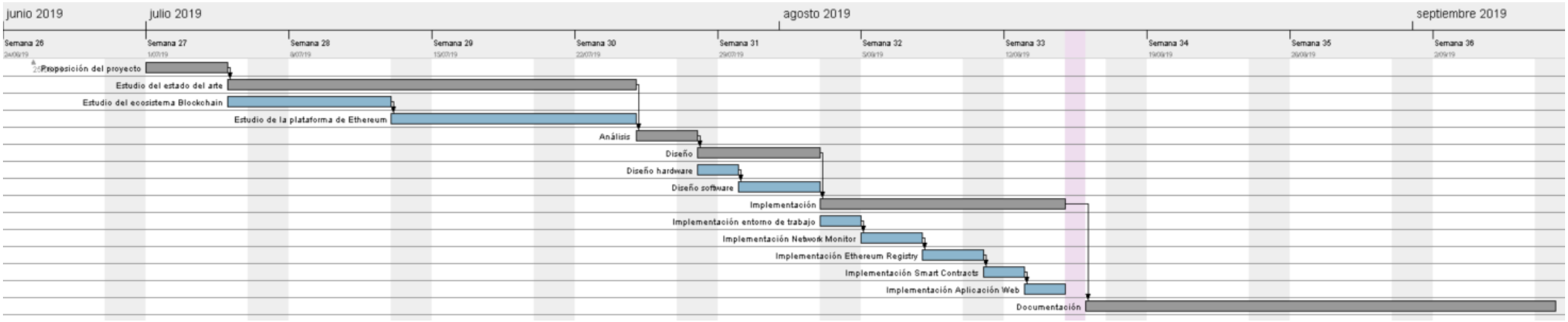


Ilustración 57.Diagrama de Gantt tiempo real invertido

Aunque no son muchas, existen ciertas diferencias entre la planificación inicial del proyecto y el tiempo real dedicado. A continuación, se detallan los motivos:

- La documentación y el estudio del arte llevó cuatro días más de lo previsto. El estudio de la plataforma de Ethereum fue la que consumió más tiempo debido a que la complejidad de esta tecnología es elevada y que hubo que bajar hasta el plano más técnico para poder reflejarlo en la memoria de manera sencilla.
- La fase de análisis se redujo puesto que mientras se realizaba el estudio del ecosistema Blockchain, el objetivo del caso de uso propuesto iba cogiendo forma y, una vez en esta fase, solo hubo que detallar el mismo.
- Se necesitó más tiempo para implementar los módulos de *Network Monitor* y *Ethereum Registry* debido a dificultades encontradas en la integración con la librería Web3.js.
- Finalmente, se dedicaron 3 días más de los previstos a la elaboración de la memoria que fueron básicamente dedicados a pulir detalles y dar formato al documento.

Capítulo 9: Conclusiones y líneas futuras

En el último capítulo se exponen las conclusiones obtenidas tras la realización del proyecto tanto desde un punto de vista objetivo como desde el personal. Además, se identifican los aspectos susceptibles de evolucionar que aporten mayor valor y funcionalidad a próximas versiones de la solución.

Llegados a este punto, es posible asegurar que los objetivos iniciales han sido cumplidos satisfactoriamente. Se ha realizado un estudio profundo de la tecnología Blockchain que ha permitido asentar los conocimientos fundamentales, pero también, los conocimientos necesarios para desarrollar una solución funcional. Además se ha realizado un recorrido por toda la Blockchain de Ethereum y ahora es posible afirmar que se trata de la plataforma pública más completa para desarrollar soluciones, gracias a la posibilidad de incluir lógica de negocio en ellas.

La elección de la prueba de concepto que se ha desarrollado en el trabajo ha resultado muy positiva, ya que representa muy bien cómo introducir la criptoeconomía para comercializar con un activo como el servicio de conexión a Internet. Además, la prueba de concepto ha fomentado el uso de tecnología relacionada con Blockchain y se ha tenido la posibilidad de programar en Solidity y de manejar otro tipo de herramientas como Remix o Metamask, indispensables en este ecosistema.

Es necesario comentar que, no se buscaba una solución concreta para un problema en concreto, si no el desarrollo de un prototipo que sirviera para identificar otros casos en los que introducir la tokenización, la criptoeconomía o cualquier otra ventaja que aporta Blockchain. En este sentido también se ha logrado el objetivo ya que, después de leer el presente proyecto es muy factible que en el lector, hayan surgido opiniones de mejoras o nuevas ideas en las que aplicar esta tecnología. De hecho, durante el desarrollo del proyecto han aparecido ideas que pueden ser aplicadas en futuras versiones del prototipo y que se describen a continuación.

Como se adelantó en el capítulo de análisis, uno de los puntos fuertes del sistema es la posibilidad de variar la política de facturación del sistema, de manera que se pueden incentivar ciertos comportamientos en el consumo del servicio. Aquí se definió una política sencilla en la que consumes más tokens cuanto más ancho de banda se esté consumiendo, se podría desarrollar un algoritmo que, por ejemplo, penalice el consumo de contenido para mayores de dieciocho años en redes públicas dedicadas a la educación. También se puede orientar la prestación del servicio en términos de velocidad, cuantos más tokens dispongas, mayor velocidad de conexión disfrutarás. Asimismo, se podría recompensar a los usuarios con buenos comportamientos en la red. En definitiva, es posible aumentar la potencia de la solución, ya que se puede implementar la política que sea en función del objetivo que se desee.

El sistema desarrollado, también puede favorecer la técnica *WiFi Offloadding* mediante la que se pretende descongestionar la red móvil a través de un cambio de conexión hacia redes WiFi. Generalmente este tipo de redes son redes públicas con una seguridad más que cuestionable. A través del modelo propuesto, cualquiera con un

servicio de Internet contratado puede ofrecer conexiones obteniendo un beneficio por ello mediante el intercambio de tokens. Una línea futura podría ser la evolución del sistema para fomentar la técnica *WiFi Offloadding* en las próximas redes 5G.

En el caso de que se desease comercializar el prototipo propuesto, sería necesario que el sistema incorporase la capacidad de gestionar la identidad digital tanto de usuarios como de prestadores de servicios para verificar y autorizar todas las operaciones que realicen, evitando así el fraude. De esta manera se podría desarrollar un *wallet* del servicio que gestionase la identidad del usuario y sus cuentas asociadas.

Como conclusión personal me gustaría destacar que este proyecto me parece un trabajo muy completo ya que he tenido la oportunidad de aprender tanto conocimientos teóricos como técnicos de esta tecnología y he podido aplicarlos de forma satisfactoria en el desarrollo de una solución. En definitiva, la realización del proyecto final de máster ha supuesto un reto personal y una motivación para seguir formándome en el mundo Blockchain.

Bibliografía

[1] Gartner, *"3 Trends Appear in the Gartner Hype Cycle for Emerging Technologies, 2016"*. **[En línea]. Disponible en**: https://www.gartner.com/smarterwithgartner/3-trends-appear-in-the-gartner-hype-cycle-for-emerging-technologies-2016/

[2] B. Hill, S. Chopra, P. Valencourt, N. Prusty, *"Blockchain Developer's Guide"*.

[3] Cointelegraph**,** *"Prueba de Trabajo (PoW)"*. **[En línea]. Disponible en:** https://es.cointelegraph.com/explained/proof-of-work-explained

[4] SCIntelligence, *"Proof of importance (POI)"*. **[En línea]. Disponible en**: https://sci.smithandcrown.com/glossary/proof-of-importance

[5] Blockchain, *"Blockchain size"*. **[En línea]. Disponible en**: https://www.blockchain.com/es/charts/blocks-size

[6] Bitcoin **[En línea]. Disponible en**: https://bitcoin.org/es/

[7] Academy, *"¿Qué puedes comprar con Bitcoin?"*. **[En línea]. Disponible en**: https://academy.bit2me.com/que-puedes-comprar-bitcoin/

[8] Cointelegraph, *"¿Qué es Ripple y cómo funciona la moneda de los bancos?"*. **[En línea]. Disponible en**: https://es.cointelegraph.com/explained/what-is-ripple-and-how-it-works

[9] Hyperledger, The Linux Foundation Projects. **[En línea]. Disponible en**: https://www.hyperledger.org

[10] Corda, *"An Open Source Blockchain Platform for Businesses"*. **[En línea]. Disponible en**: https://www.corda.net/

[11] Deloitte, *"The tokenization of assets is disrupting the financial industry. Are you ready?"*. **[En línea]. Disponible en**: https://www2.deloitte.com/content/dam/Deloitte/lu/Documents/financial-services/lu-tokenization-of-assets-disrupting-financial-industry.pdf

[12] CryptoKitties. **[En línea]. Disponible en**: https://www.cryptokitties.co

[13] Ethereum. **[En línea]. Disponible en**: https://www.ethereum.org/

[14] Solidity, *"Introducción a los contratos inteligentes"*. **[En línea]. Disponible en**: https://solidity-es.readthedocs.io/es/latest/introduction-to-smart-contracts.html

[15] Open Zeppelin contracts. **[En línea]. Disponible en**: https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token

[16] My Ether wallet, *"What is gas?"*. **[En línea]. Disponible en**: https://kb.myetherwallet.com/en/transactions/what-is-gas/

[17] Ethereum Stack Exchange, *"Comparison of the different TestNets"*. **[En línea]. Disponible en**: https://ethereum.stackexchange.com/questions/27048/comparison-of-the-different-testnets

[18] Solidity documentation. **[En línea]. Disponible en**: https://solidity.readthedocs.io/en/v0.5.11/

[19] Ethereum Solidity Language for Visual Studio Code. **[En línea]. Disponible en**: https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity

[20] Remix, *"Welcome to Remix documentation"*. **[En línea]. Disponible en**: https://remix-ide.readthedocs.io/en/latest/

[21] Metamask, *"Brings Ethereum to your browser"*. **[En línea]. Disponible en**: https://metamask.io/

[22] Ganache, *"One Click Blockchain"*. **[En línea]. Disponible en**: https://www.trufflesuite.com/ganache

[23] Cryptozombies Solidity, *"Learn to code Ethereum DApps by building your own game"*. **[En línea]. Disponible en**: https://cryptozombies.io/

[24] Cryptozombies Libra, *"Masters the Basics of Libra Blockchain Development – Before everyone else"*. **[En línea]. Disponible en**: https://cryptozombies.io/libra/

[25] Raspberry Pi Org, Raspberry Pi 3 Model B+. **[En línea]. Disponible en**: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[26] Arduino Store, Arduino uno WiFi Rev2. **[En línea]. Disponible en**: https://store.arduino.cc/arduino-uno-wifi-rev2

[27] Wikipedia, *"Python lenguaje de programación"*. **[En línea]. Disponible en**: https://es.wikipedia.org/wiki/Python

[28] Python, *"The official home of the Python Programming Language"*. **[En línea]. Disponible en**: https://www.python.org/

[29] Wikipedia, *"JavaScript"*. **[En línea]. Disponible en**: https://es.wikipedia.org/wiki/JavaScript

[30] JavaScript web. **[En línea]. Disponible en**: https://www.javascript.com/

[31] Node.js web. **[En línea]. Disponible en**: https://nodejs.org/es/

[32] React web, *"Una biblioteca de Javascript para construir interfaces de usuario"*. **[En línea]. Disponible en**: https://es.reactjs.org/

[33] Web3 JavaScript Library. **[En línea]. Disponible en**: https://web3js.readthedocs.io/en/v1.2.1/

[34] Web3 Python Library. **[En línea]. Disponible en**: https://web3py.readthedocs.io/en/stable/index.html

[35] tcconfig, *"Welcome to tcconfig's documentation"*. **[En línea]. Disponible en**: https://tcconfig.readthedocs.io/en/latest/index.html

[36] web3.eth module. **[En línea]. Disponible en**: https://web3js.readthedocs.io/en/v1.2.1/web3-eth.html

[37] Raspberry Pi Org, *"Success with access point on Raspbian Stretch"*. **[En línea]. Disponible en**: https://www.raspberrypi.org/forums/viewtopic.php?t=238573

[38] Hostapd Gentoo Wiki. **[En línea]. Disponible en**: https://wiki.gentoo.org/wiki/Hostapd

[39] Wiki Archlinux, *"dnsmasq"*. **[En línea]. Disponible en**: https://wiki.archlinux.org/index.php/Dnsmasq_(Espa%C3%B1ol)

[40] Go-Ethereum, *"Official Go implementation of the Ethereum protocol"*. **[En línea]. Disponible en**: https://github.com/ethereum/go-ethereum

[41] Wikipedia, *"tmux"*. **[En línea]. Disponible en**: https://es.wikipedia.org/wiki/Tmux

Anexo A: Configuración de Raspberry Pi 3B+ como gateway de acceso inalámbrico a red

Una de las capacidades que Raspberry Pi ofrece es la configuración como punto de acceso inalámbrico a la red. A continuación, se muestra el proceso de configuración necesario [37]:

En primer lugar, es necesario instalar dos paquetes de software que harán la configuración más sencilla:

- **Hostapd:** *Host access point Daemon* es un software gratuito y *open source* para GNU/Linux que permite convertir las tarjetas de interfaz de red en puntos de acceso inalámbricos y con autenticación. El único requisito es que la tarjeta sea compatible con el modo Access Point (AP) [38].
- **Dnsmasq:** Un paquete software *open source* que proporciona un servidor Domain Name Server (DNS) destinado a resolver los nombres y direcciones IP de los dispositivos que se conecten al punto de acceso. También proporciona un servidor Dynamic Host Configuration Protocol (DHCP) para la asignación dinámica de IPs en función de la configuración de red [39].

Para la instalación de ambos paquetes se deben ejecutar los siguientes comandos:

```
$ sudo apt-get install hostapd
```

```
$ sudo apt-get install dnsmasq
```

Una vez instalados, se paran ambos procesos puesto que hay que editar sus correspondientes archivos de configuración:

```
$ sudo systemctl stop hostapd
```

```
$ sudo systemctl stop dnsmasq
```

Es interesante que el enrutador tenga una dirección IP que sea estática, puesto que, en otro caso, habría que modificar la configuración constantemente. Para asignar

una dirección estática al dispositivo hay que editar el archivo `/etc/dhcpcd.conf.` En este caso, se asigna la dirección IP estándar de la red local 192.168.220.1 a la interfaz wlan0 que es la correspondiente al adaptador WiFi interno del dispositivo. Las líneas que hay que añadir al archivo, por lo tanto, son:

```
interface wlan0
    static ip_address=192.168.220.1/24
```

A continuación, se reinicia el servicio dhcpcd para que la interfaz wlan0 obtenga su nueva dirección IP fija:

```
$ sudo systemctl restart dhcpcd
```

Llegados a este punto, en el que el servidor DNS se encuentra establecido, se procede a configurar el punto de acceso inalámbrico. Para ello se debe editar el archivo `/etc/hostapd/hostapd.conf` añadiendo las siguientes líneas:

```
interface=wlan0
driver=nl80211
hw_mode=g
channel=6
ieee80211n=1
wmm_enabled=0
macaddr_acl=0
ignore_broadcast_ssid=0
auth_algs=1
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
ssid=Raspi-Brain
wpa_passphrase= Raspi-Brain-Password
```

Los parámetros más destacables en este archivo de configuración son:

- **Driver=nl80211:** Es el controlador que corresponde al adaptador WiFi interno de la placa Raspberry por el que se va a ofrecer la conexión inalámbrica.

- **ieee80211n=1:** Indica que se sirve soporte para el estándar o más popularmente conocido como WiFi 4.
- **Wpa=2:** Se habilita el sistema de protección de redes inalámbricas WiFi Protected Access 2.
- **ssid=Raspi-Brain:** Es el nombre de la red inalámbrica que se está configurando.
- **wpa_passphrase= Raspi-Brain-Password:** Es la contraseña de acceso a WiFi.

Una vez editado se especifica al sistema la ubicación del archivo de configuración:

```
$ sudo vim /etc/default/hostapd
```

Añadiendo esta línea al archivo:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

```
$ sudo vim /etc/init.d/hostapd
```

Añadiendo esta línea al archivo:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Llegados a este punto, se establece el servidor DHCP para la asignación dinámica de IP en función de los parámetros especificados para la configuración de red. El archivo de configuración se localiza en /etc/dnsmasq.conf.orig:

```
interfaz = wlan0
  server=8.8.8.8
  dhcp-range = 192.168.220.50,192.168.220.150,12h
```

Con estos parámetros se indica que la asignación de valores de IPs para los dispositivos que se conecten a la interfaz wlan1 serán los comprendidos entre 192.168.220.50 y 192.168.220.150 y se renovarán cada 12 horas. El servidor DNS es el correspondiente al de Google (8.8.8.8).

Seguidamente, hay que configurar el reenvío de tráfico de datos a través de la toma Ethernet (interfaz eth0) existente entre el dispositivo y el módem que se realizará cuando se conecten los diferentes dispositivos de manera inalámbrica. El archivo a editar es `/etc/sysctl.conf`, en el que hay que descomentar la línea "`net.ipv4.ip_forward=1`". Este parámetro indica la activación del reenvío IP versión 4 en el dispositivo.

Para enmascarar el tráfico IP del tráfico saliente por el cable Ethernet (interfaz eth0) se crea y guarda una regla en el firewall de Linux, iptables:

```
$ sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
$ sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
$ sudo vim /etc/rc.local
$ iptables-restore < /etc/iptables.ipv4.nat
```

Esto es necesario, puesto que se ha definido que la Raspberry tenga un rango de IP local fijo y distinto al del resto de la red, por lo que el router al que está conectado el dispositivo no reconoce como propios los paquetes que salen de ella. Es por ello que se enmascaran los paquetes salientes con el rango correspondiente al router (concretamente a la interfaz eth0) que sí que tiene una IP pública.

El último comando hace que se cargue la regla en el arranque del dispositivo. Tras esta configuración ya es posible arrancar los servicios de hostapd y dnsmasq:

```
$ sudo service hostapd start
$ sudo service dnsmasq start
```

También es necesario especificar que estos dos servicios se inicien en el arranque del dispositivo:

```
$ sudo update-rc.d hostapd defaults
$ sudo update-rc.d hostapd enable
$ sudo update-rc.d dnsmasq defaults
$ sudo update-rc.d dnsmasq enable
```

El último paso es reiniciar el dispositivo:

```
$ sudo reboot
```

Una vez reiniciado la Raspberry los dipositivos ya observan el punto de acceso inalámbrico recién configurado:

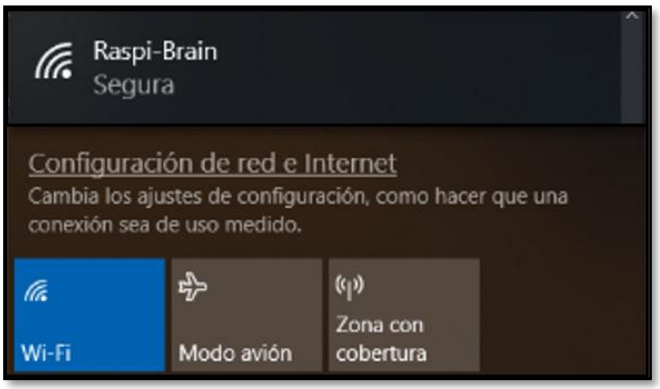


Ilustración 58. Punto de acceso del dispositivo Raspberry

Anexo B: Instalación y arranque de un nodo Geth en Raspberry Pi 3B+

Como se mencionó en los apartados de diseño e implementación por motivos de rendimiento se ha decidido instalar un nodo Geth en modo ligero para que no sea necesario descargar y sincronizar la cadena de bloques en su totalidad. La ejecución del nodo en modo ligero supone directamente un ahorro en el consumo de recursos del dispositivo puesto que solo se descargarán las cabeceras de los bloques.

En primer lugar, previo a la instalación del nodo Geth es necesario instalar una serie de dependencias:

- **Git:** Software de control de versiones que permite la gestión de repositorios de código.
- **libgmp3-dev:** Herramientas de desarrollo de biblioteca aritmética multiprecisión.

Para ello primero hay que asegurarse de que todos los paquetes de software están actualizados:

```
$ sudo apt-get update
$ sudo apt-get upgrade -y
```

El nodo también necesita la instalación de la distribución de Golang (el lenguaje de programación de Google) puesto que su implementación del nodo está escrita en este lenguaje [40]:

```
$ wget https://storage.googleapis.com/golang/go1.10.1.linux-armv6l.tar.gz
$ sudo tar -zxvf go1.10.1.linux-armv6l.tar.gz -C /usr/local/
$ echo 'export GOROOT=/usr/local/go' >> ~/.bashrc
$ echo 'export GOPATH=$HOME/go' >> ~/.bashrc
$ echo 'export PATH=$PATH:$GOROOT/bin:$GOPATH/bin' >> ~/.bashrc
```

```
$ go version
    go version go1.10.1 linux/amd64
```

En este punto se procede a la descarga de la implementación del nodo desde su repositorio:

```
$ git clone https://github.com/ethereum/go-ethereum
$ cd go-ethereum
$ make geth
$ sudo cp build/bin/geth /usr/local/bin/
```

Una vez instalado y compilado el nodo es necesario asociar una cuenta de usuario con fondos de Ethereum (Ether) para poder desplegar e interactuar con los smart contracts que se desee. El nodo va a formar parte de la *testnet* de Rinkeby y la importación de una cuenta existente se realiza desde MetaMask a través de Web3. Para ello se necesita especificar la clave privada de la cuenta y una nueva contraseña:

```
$ geth console –rinkeby
$     web3.personal.importRawKey("<Private     Key>","<New Password>")
```

El último paso es el arranque del nodo y para ello, se ejecuta la interfaz de línea de comando "geth" seguido de unos parámetros de configuración:

```
$ geth --rinkeby --syncmode light --rpc --rpcaddr "localhost" --rpcport 8545 --rpcapi web3,eth,net,personal
```

- **--rinkeby:** Rinkeby network: pre-configured proof-of-authority test network
- **--syncmode:** Blockchain sync mode ("fast", "full", or "light")
- **--rpc:** Enable the HTTP-RPC server
- **--rpcaddr value:** HTTP-RPC server listening interface (default: "localhost")
- **--rpcport value:** HTTP-RPC server listening port (default: 8545)
- **--rpcapi value:** API's offered over the HTTP-RPC interface

Tras el arranque del nodo comienza la sincronización de bloques que, como ya se había adelantado consiste sólo en la descarga de las cabeceras.

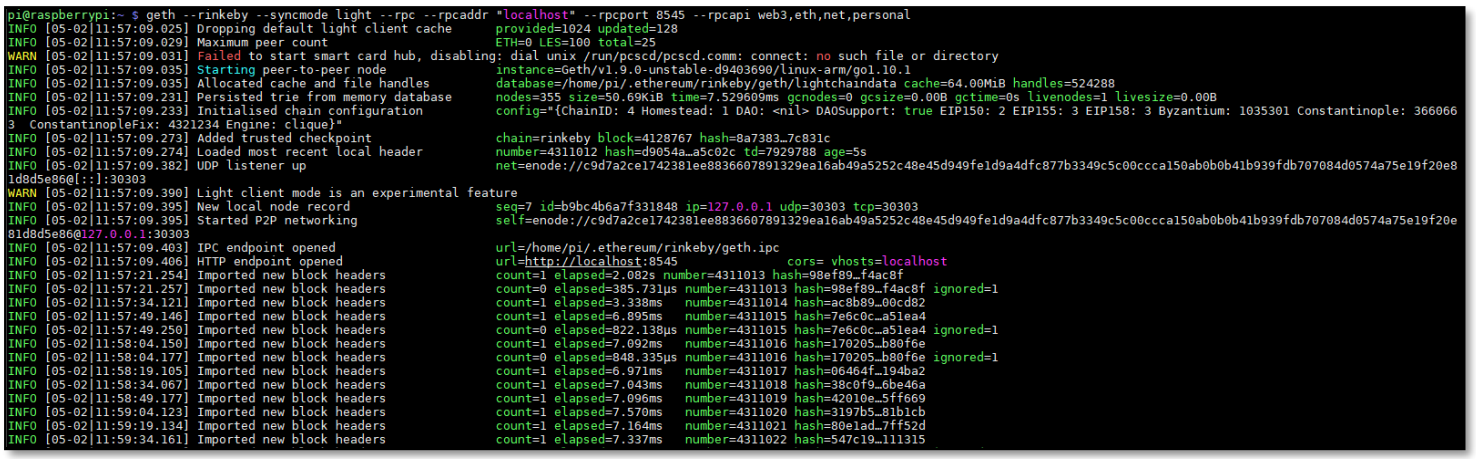


Ilustración 59. Arranque del nodo Geth en Raspberry Pi

El nodo Geth debe estar continuamente en ejecución puesto que será el punto de conexión a la red Ethereum mediante el cual se realizarán todas las interacciones con los contratos inteligentes. Para mantener el nodo en ejecución se ha optado por utilizar tmux, un multiplexor de terminal para sistemas tipo unix, que permite lanzar múltiples terminales dentro de única sesión de consola [41]. Además, las sesiones en esta herramienta son persistentes, lo que significa que, aunque cerremos el propio proceso tmux, los procesos que alberga siguen en ejecución. Para su instalación se deben ejecutar los siguientes comandos:

```
$ sudo apt-get install tmux
```

Para arrancar la herramienta simplemente se ejecuta el comando *"tmux"*. Una vez dentro, se nombra la sesión como se desee ejecutando la combinación *"Ctrl + b + ,"*. En este caso el nombre elegido es *"Geth"*. En este punto se ejecuta el comando para arrancar el proceso del nodo que se quedará ejecutando de manera persistente:

```
$ geth --rinkeby --syncmode light --rpc --rpcaddr "localhost" --rpcport 8545 --rpcapi web3,eth,net,personal
```