ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

ANÁLISIS DE FACTIBILIDAD EN LA IMPLEMENTACIÓN DE BLOCKCHAIN SOBRE UN PROTOTIPO DE INFRAESTRUCTURA IOT-MQTT.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL GRADO DE MAGISTER EN SOFTWARE, MENCIÓN SEGURIDAD

GUILLERMO GABRIEL ANDRADE SALINAS

guillermo.andrade@epn.edu.ec

DIRECTOR: ING. GUSTAVO DAVID SALAZAR CHACÓN, MSC.

gustavo.salazar@epn.edu.ec

CO-DIRECTORA: ING. LUZ MARINA VINTIMILLA JARAMILLO, MSC.

marina.vintimilla@epn.edu.ec

Quito, marzo 2020

APROBACIÓN

Como director y codirector del trabajo de titulación “ANÁLISIS DE FACTIBILIDAD EN LA IMPLEMENTACIÓN DE *BLOCKCHAIN* SOBRE UN PROTOTIPO DE INFRAESTRUCTURA IOT-MQTT” desarrollado por Guillermo Gabriel Andrade Salinas, estudiante de la Maestría en Software, Mención: Seguridad, habiendo supervisado la realización de este trabajo y realizado las correcciones correspondientes, damos por aprobada la redacción final del documento escrito para que prosiga con los trámites correspondientes a la sustentación de la Defensa Oral.

Certificamos que el presente trabajo fue desarrollado por Guillermo Gabriel Andrade Salinas, bajo nuestra supervisión.

Ing. Gustavo Salazar, MSc

DIRECTOR DEL PROYECTO

Ing. Luz Marina Vintimilla, MSc

CO-DIRECTORA DEL PROYECTO

DECLARACIÓN

Yo Guillermo Gabriel Andrade Salinas, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Guillermo Gabriel Andrade Salinas

DEDICATORIA

A mi Mami, a mi Papi ...

A mi Hermano y Hermana …

Y a todos mis amigos.

Guillermo

AGRADECIMIENTOS

Agradezco a mis padres Wilian Andrade y Nancy Salinas, mis hermanos Paúl y Cristina, mi cuñada Yajaira y mi cuñado Andrés que me han apoyado en cada decisión de mi vida, compartido sus conocimientos y experiencia para hacer de mí una mejor persona.

A mi sobrino Isaac y mis sobrinas Micaela y Naomi que, desde su llegada, llenan de alegría mi vida.

A mis amigos Andrés Rodríguez y Vinicio Aroca quienes han demostrado una gran amistad y apoyo en momento difíciles de mi vida.

De manera muy especial a mis tutores Gustavo Salazar y Luz Marina Vintimilla, por guiarme, compartir sus conocimientos y experiencias, y brindarme apoyo en la realización de este proyecto.

Guillermo

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El Internet de las cosas (IoT) ha presentado un crecimiento exponencial en los últimos años. Sin embargo, debido a los limitados recursos de los dispositivos y su arquitectura centralizada, existen problemas difíciles de resolver, tales como sobrecarga del servidor centralizado, punto único de falla y la posibilidad de uso malicioso de la información. *Blockchain* presenta muchas características únicas, tales como mecanismo de consenso, comunicación entre pares, implementación de confianza sin un tercero confiable y transacciones basadas en contratos inteligentes. Estas características parecen ser adecuadas para construir un sistema de IoT distribuido y autónomo que supere los problemas mencionados. *Blockchain* requiere un alto rendimiento computacional, alto ancho de banda y produce retrasos, escenarios no adecuados para IoT, pero comunes en *Blockchain*. Este documento expone el estudio de la factibilidad de implementar la plataforma *Hyperledger Fabric* considerando los dispositivos de IoT como nodos *Peer* de la red. Para esto se llevaron a cabo mediciones de ancho de banda y pruebas de rendimiento en diferentes escenarios transaccionales. Adicionalmente, se propone un modelo de alto nivel para la implementación de la plataforma *Hyperledger Fabric* en dispositivos IoT.

Palabras Clave: Seguridad en IoT, Raspberry Pi, Aplicaciones Blockchain, Contratos Inteligentes, REST, SDK Fabric, Design Research, Orderer.

ABSTRACT

The Internet of Things (IoT) has shown exponential growth in recent years. However, due to the limited resources and centralized architecture, there are difficult-to-solve problems such as centralized server overload, single point of failure, and possibility of malicious use of information. Blockchain presents unique features such as consensus mechanism, peer communication, trusted implementation without a reliable third party and smart contract-based transactions. These features seem to be suited to build a distributed and autonomous IoT system in order to overcome IoT issues. High processor performance, high bandwidth, and delays are not suitable scenarios for IoT, but are common in Blockchain applications. This research presents the feasibility of deploying Blockchain by considering IoT devices as Peer nodes on a Hyperledger Fabric network. Bandwidth measurements and performance tests were performed in transactional scenarios. Additionally, a high-level model for implementation of Hyperledger Fabric platform on IoT devices is proposed.

Keywords: *Hyperledger Fabric*, Internet of Things (IoT), IoT security, Raspberry Pi, *Blockchain* applications, *Smart Contract*.

TÍTULO

Integration of IoT devices as *Peer* nodes in a *Blockchain* network based on *Hyperledger Fabric*.

INTRODUCCIÓN

El crecimiento acelerado del Internet de las Cosas (Internet of Things - IoT) ha impulsado una transformación digital de la industria, incluyendo campos como medicina (monitoreo de cirugías), automóviles (rastreo), hogar (casas inteligentes), entre otros. Estadísticas publicadas por *Statista Research Department* [1], muestran que alrededor del mundo existen cerca de 30.73 billones de dispositivos conectados y se prevé para el 2025 existan más de 75 billones. Desafortunadamente, las redes existentes de IoT tienen desventajas significativas, como tiempo de respuesta bajos y problemas de seguridad y privacidad.

Minhaj [2] propone *Blockchain* como una solución prometedora para abordar los problemas antes mencionados y potenciar esta tecnología con los valores comerciales y operativos de IoT. A pesar de las características prometedoras de *Blockchain*, los equipos de IoT tienen algunas limitaciones para su completa integración. Procesadores, capacidad de memoria ROM y ancho de banda son las principales limitantes. Por ello, existen pocas soluciones de *Blockchain* que utilicen un equipo IoT como un nodo completo de la red *Blockchain*. Minoli [3] y Fakhri [4] han realizado implementaciones de *Blockchain* en equipos IoT acoplándolos como usuarios de la red, mas no como miembros *Peer*.

En la actualidad la tecnología *Blockchain* es ampliamente aceptada e implementada en muchas industrias, siendo la base del Bitcoin, moneda reconocida internacionalmente. *Blockchain*, aparte de su uso en transacciones de criptomonedas, se ha utilizado en varios campos, debido a las ventajas de las cadenas de bloques, como la verificación descentralizada de la información y la resistencia a la manipulación. Por ello, su aplicación en las industrias que incluyen logística, sistemas financieros (micro pagos / sistemas de pago móviles), registros médicos, recopilación y verificación de datos en IoT, distribución de productos digitales, entre muchas otras aplicaciones [5]. Se espera que la tecnología *Blockchain* pueda garantizar que los sistemas desarrollados en base a esta tecnología mantengan la seguridad y privacidad de la información, gracias a que mantiene un registro detallado de las transacciones (información). La integración de *Blockchain* e IoT está creando nuevos niveles de confianza, una red descentralizada y la verificabilidad de transacciones [6].

Esta investigación plantea analizar la factibilidad de implementar una plataforma privada *Blockchain* (*Hyperledger*) en una red IoT que se comunica por medio del protocolo

Message Queue Telemetry Transport (MQTT), utilizando un Raspberry Pi 2B+ como dispositivo IoT e integrándolo a la red *Blockchain* como un nodo completo.

Pregunta de Investigación

El presente estudio comprende el análisis de la viabilidad de implementar la tecnología *Blockchain* en una red IoT que se comunica a través del protocolo MQTT, usando un Raspberry Pi 2B+ como dispositivo IoT que funciona como un nodo *Peer* que mantiene una copia del *Ledger*. Por lo cual se han planteado la siguiente pregunta:

¿Es posible integrar la tecnología *Blockchain* en una infraestructura IoT para mejorar la seguridad y privacidad de la información, empleando comunicación bajo el protocolo MQTT implementada en un prototipo de red?

Objetivo General

Analizar la factibilidad en la implementación de *Blockchain* sobre un prototipo de infraestructura IoT-MQTT para la implementación de esta tecnología en la protección de la información.

Objetivos Específicos

- Identificar los principios de funcionamiento de las tecnologías *Blockchain*.
- Implementar un prototipo de red básico IoT con las tecnologías *Blockchain* y MQTT.
- Probar la funcionalidad de la implementación de *Blockchain* en equipos IoT-MQTT mediante mediciones y pruebas de rendimiento del procesador del equipo, inyectores de tráfico y pruebas de funcionamiento de *Blockchain*.
- Proponer un esquema de implementación genérico de *Blockchain* para redes IoT.

1. REFERENCIAL TEÓRICO

En este primer acápite se abordará conceptos de las tecnologías utilizadas en el proyecto de investigación, tales como Internet de las cosas (Internet of Things - IOT), Blockchain y su plataforma privada "*Hyperledger Fabric*".

1.1. Internet of Things

De acuerdo a Fakhri [4], "*IoT es un sistema de dispositivos interrelacionados; es decir, la interconexión e interacción de varios dispositivos con la ayuda de sistemas embebidos, sensores, programas e inteligencia artificial, proporcionando información a través de Internet sin que el ser humano intervenga en su operatividad"*. IoT permite aplicaciones que

proporcionan mejoras en las actividades tanto de la vida cotidiana de las personas como los cambiantes entornos empresariales.

IoT facilita la conexión de objetos inteligentes a Internet, permitiendo que se conecten dispositivos informáticos tradicionales y no tradicionales, desarrollando una comunicación máquina a máquina (machine-to-machine M2M). Esta comunicación ha permitido las implementaciones de casas inteligentes, vehículos autónomos, equipos controlados remotamente, entre otras aplicaciones que convierten objetos de uso diario en equipos inteligentes [7], [8].

Alrededor del planeta se está ampliando el uso de más dispositivos en las redes de IoT, presentando así, un incremento considerable para futuros años, abriendo una mayor posibilidad que se lleven a cabo gran cantidad de ataques cibernéticos. Por lo que, usuarios y fabricantes deben buscar tecnologías que impidan interferencias maliciosas que comprometan la privacidad y seguridad de los dispositivos IoT y de la información que estos almacenan.

Singh [9] menciona que la vulnerabilidad a los Ataques Cibernéticos es uno de los principales problemas de seguridad en IoT, debido a que los dispositivos son fabricados para disponer de costos accesibles, por lo tanto, violan políticas de seguridad que tienen el propósito de evitar ataques invasivos y no invasivos, tales como Side Channel Attack, Voltage Attacks, Temperature Attacks, por consiguiente haciendo difícil disponer de alta seguridad en las redes IoT.

Atlam [10] lista el cómo la tecnología *Blockchain* puede afrontar los problemas de IoT; entre los de mayor consideración se encuentran:

Tabla 1: Soluciones Blockchain para problemas de IoT.

Problemas IoT	**Solución basada en *Blockchain***
Seguridad	*Blockchain* provee un ambiente seguro e inmutable para varios tipos de dispositivos IoT.
Único punto de falla	*Blockchain* utiliza una comunicación descentralizada y distribuida entre los nodos participantes.
Susceptible a manipulación	La modificación de la información en *Blockchain* se realiza luego de una aprobación por consenso, por lo que puede detectar y prevenir cualquier acción maliciosa
Autenticación y control de acceso	Los Contratos Inteligentes tienen la capacidad de proveer reglas y lógica de autenticación descentralizadas que pueden permitir una autenticación eficiente de los dispositivos IoT.
Privacidad	La información almacenada en una red *Blockchain* puede ser anónima y protegida contra modificaciones.

Los entornos de IoT presentan desafíos en gestión, interoperabilidad, comunicaciones, seguridad y falta de estandarización adecuada [11]. Se ha utilizado varios protocolos en la capa de aplicación del modelo TCP/IP (capa 5, 6 y 7 del modelo OSI) para buscar mejorar la seguridad en entornos IoT, entre ellos se encuentra el protocolo MQTT [12]. MQTT surgió como un protocolo que resuelve muchos de estos problemas, debido a su modelo publicación/suscripción, así como a características de calidad de servicio (*Quality of Service* - QoS), cifrado, seguridad y la posibilidad de establecer comunicaciones confiables y no confiables [11]. Sin embargo, si estas características no se las configura adecuadamente pueden dejar vulnerable a la red. Zamfir [13] demuestra una tendencia en asegurar MQTT, utilizando la capa de transporte con certificación de nivel de socket seguro (SSL). La implementación realizada por Gustavo [8], permite demostrar las ventajas en la seguridad y funcionalidad de la utilización del protocolo MQTT en redes IoT.

En investigaciones sobre seguridad en redes IoT realizadas por Biswas [14], Dorri [7], Reyna [15] y Banerjee [16], se abordan varias propuestas para mejorar la seguridad, presentando un mayor enfoque en la utilización de Blockchain junto a protocolos de comunicación.

1.2. *Blockchain*

Atlam [17] define *Blockchain* como un libro de contabilidad o libro mayor (*Ledger*) distribuido y descentralizado, utilizado para administrar un conjunto de registros. Para almacenar una transacción en el *Ledger*, esta debe ser aceptada y registrar el consentimiento de los nodos participantes en la red *Blockchain*. Un conjunto de transacciones se agrupa y se asigna en un bloque del *Ledger*. Para vincular los bloques entre sí, cada uno registra una marca de tiempo y el hash del bloque anterior. El hash valida la integridad y el no rechazo de los datos dentro del bloque [18]. Además, cada usuario dispone de una copia de *Ledger* y todos los nodos se sincronizan y se actualizan con los nuevos cambios.

Blockchain, según Singh [9], está construido en base a cuatro pilares, que representan su gran valor en la seguridad de la información:

1) Consenso (*Consensus*), todos los miembros de la red verifican cada acción,
2) Libro mayor (*Ledger*), registra un detalle completo de las transacciones efectuadas,
3) Criptografía (*Cryptography*), asegura que todos los datos se encuentran cifrados; y finalmente,
4) Contrato inteligente (*Smart Contract*), utilizado para verificar y validar a los miembros y las transacciones de la red.

Blockchain almacena un registro de todas las transacciones en los nodos participantes, en lugar de un servidor central. Los principales usos de *Blockchain* se encuentran en el mundo financiero y últimamente se está utilizando en soluciones de IoT, por lo tanto, *Blockchain* puede ser una gran ayuda para lograr IoT descentralizada, permitiendo transacciones y coordinación entre los dispositivos que interactúan. *Blockchain* es de gran ayuda para resolver los problemas de escalabilidad, seguridad, privacidad y confianza en IoT. Con *Blockchain* los dispositivos pueden comunicarse a través de la red de manera directa, segura y confiable, sin intermediarios. *Blockchain* permite verificar, validar, rastrear y almacenar todo tipo de información, desde certificados digitales, sistemas de votación democráticos, servicios de logística y mensajería, contratos inteligentes, y por supuesto dinero y transacciones financieras [19].

Entre las plataformas disponibles para la implementación de *Blockchain* se dispone de algunos tipos [20]–[22]:

- ***Privadas o permissioned*** presentan control en el acceso a la red estableciendo normas o acuerdos entre los participantes, únicamente aplicativos autorizados pueden participar en la red. La cadena de bloques se define por consorcio, el acuerdo puede realizarse dentro de una organización. La información no es pública, no es accesible ni puede ser consultada por cualquier usuario de la red. Algunas plataformas privadas son: *Hyperledger, Tembusu y CryptoCorp*
- ***Públicas o permisionless*** son aquellas donde cualquiera con un equipo y acceso a la red se puede unir, es necesario configurar los equipos en base a los criterios de la cadena de bloques. En estas plataformas la información es pública y puede ser consultada por externos. En este tipo de plataformas se preserva el anonimato. Algunas plataformas públicas son: Bitcoin, Ethereum o Litecoin.
- ***Híbridas*** resultan de una combinación entre las plataformas públicas y privadas. En una plataforma híbrida los nodos participantes deben ser invitados, pero las transacciones son públicas, es decir, los nodos participan en el mantenimiento y seguridad, pero las transacciones son visibles para todo el mundo. Algunas plataformas hibridas son: BigchainDB o Evernym.
- ***Blockchain as a Service (BaaS),*** son aquellas que ofertan sus servicios de *Blockchain* en la nube. Estos servicios no sólo consisten en almacenamiento de información, además, ofrecen un aumento en la seguridad, la no necesidad de invertir en hardware y la posibilidad de un entorno más amigable con el que trabajar. Presentan la posibilidad de crear un canal de *Blockchain* sin necesidad de programar. Algunas empresas que ofertan plataformas BaaS son: IBM

(*Hyperledger Fabric*), Amazon (Digital Currency Group) y Microsoft (R3, *Hyperledger Fabric* o Quorum).

Las plataformas públicas como Bitcoin han definido 3 tipos de nodos [23]:

- Nodos completos son los responsables que mantener y distribuir copias del *Ledger* a todos los miembros de la red. Como tal, juegan un papel vital en la red, ya que son el punto de referencia para validar la cadena de bloques.
- Nodos ligeros realizan una función similar a los nodos completos, en lugar de contener una copia completa del *Ledger*, solo contienen una parte de ella.
- Nodos mineros son los encargados de generar los bloques, utilizando la información de las transacciones y algoritmos criptográficos.

1.3. *Hyperledger Fabric*

Hyperledger Fabric [24] es una plataforma de código abierto de *Blockchain* mantenido por *The Linux Fundation*, diseñado para apoyar transacciones empresariales globales. Una red *Hyperledger Fabric* cuenta con una diferenciación entre los nodos miembros, la misma que se basa en el grado de participación del nodo [25]:

- **Cliente** (*cliente*), es un nodo que envía una invocación de transacción, representa la identidad tras un usuario final.
- **Nodo par** (*Peer*), es un nodo que puede invocar transacciones, pero su principal función es mantener el estado y una copia del *Ledger*. Un nodo *Peer* puede tomar un rol adicional conocido como par verificador (*endorsing Peer*). Endorsing *Peer* es un nodo encargado de aprobar las transacciones utilizando el *Smart Contract* instalado.
- **Nodo de orden** (*Orderer*), es un nodo que ejecuta el servicio de comunicación que implementa una garantía de entrega. Este nodo genera un canal de comunicación entre los clientes (*client*) y los pares (*Peer*), ofreciendo un servicio de difusión de mensajes que contienen las transacciones.

Es necesario identificar que en la plataforma *Hyperledger*, el nodo cliente se puede distinguir como un nodo ligero; el nodo *Peer* se considera como un nodo completo, y el servicio *Orderer* se asigna a la función de los nodos mineros.

Hyperledger dispone de una Autoridad Certificadora (*Certificate Authority* - CA), necesaria para la generación de identidades y certificados para cada Peer.

Los *Peers*, *Chaincode* (o *Smart Contract*), *Ledger* y el servicio *Orderer* son los elementos de mayor importancia para el desarrollo de una red *Blockchain* basada en la plataforma *Hyperledger Fabric*. A continuación, se explica en grandes rasgos cada uno de ellos [25].

1.3.1. Peers

En la Figura 1 se puede observar la iteración entre las aplicaciones desarrolladas (Servidor REST) con los *Peers* [26]. Las consultas al *Ledger* implican un proceso de tres pasos entre una aplicación y un *Peer*, mientras que las actualizaciones del *Ledger* requieren dos pasos adicionales.

Las aplicaciones siempre se conectan con los Peers cuando necesitan acceder al *Ledger* y Chaincodes. A través de una conexión entre pares, las aplicaciones ejecutan los Chaincodes para consultar o actualizar el *Ledger*. El resultado de una consulta al *Ledger* se devuelve de inmediato. Los Peers y los *Orderer* se aseguran de que *Ledger* este actualizado en cada *Peer*.

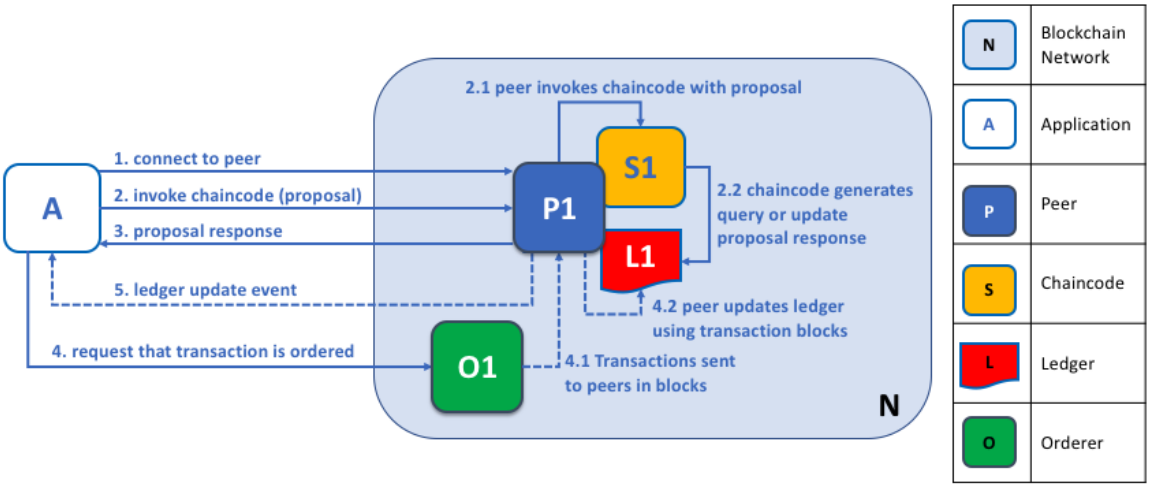


Figura 1: Proceso de aprobación de transacciones y creación de bloques [18].

En este ejemplo, A se conecta a P1 e invoca el Chaincode S1 para consultar o actualizar el *Ledger* L1. P1 invoca S1 generando una respuesta de propuesta que contenga un resultado de consulta o una actualización del *Ledger* propuesta; la aplicación A recibe la respuesta de la propuesta. Para consultas el proceso está completo. Para las actualizaciones, A construye una transacción a partir de las respuestas, que la envía al *Orderer* O1, el que recopila transacciones de toda la red en bloques y las distribuye a todos los *Peer*. P1 valida la transacción antes de actualizar L1. Una vez que L1 se actualiza, P1 genera un evento, recibido por A, para indicar la finalización.

1.3.2. Smart Contract y Chaincode

El *Smart Contract* [27] define la lógica ejecutable que genera nuevos datos para agregarlos al *Ledger*. El *Smart Contract* define una seria de términos, datos, reglas, procesos, entre otros, que definen el modelo que rige las interacciones entre las partes en las transacciones. El Chaincode es generalmente usado por los administradores para agrupar los *Smart Contract* relacionados, sin embargo, pueden ser utilizados para programación del sistema de Fabric en bajo nivel.

Smart Contract accede a dos sectores del *Ledger*, un *Blockchain* que llega el registro inmutable de todas las transacciones y al WorldState que contiene un cache del valor actual.

Un *Smart Contract* principalmente realiza 3 acciones en el WorldState que son PUT, GET, DELETE. PUT es la creación o modificación de un objeto en el *Ledger*. GET representa la consulta para obtener el estado actual de un objeto. DELETE representa la eliminación de un objeto del WorldState, pero no de su historial.

Para la aprobación de una transacción, *Smart Contract* se encarga de validar que cumpla las políticas definidas en este para declarar la transacciones como válidas y firmarlas. Todas las transacciones válidas y no válidas se añaden al historial de *Blockchain*, pero solo las válidas actualizan los valores del WorldState.

1.3.3. Ledger

El *Ledger* [28] está conformado por dos sectores: un *WorldState* y el *Blockchain*. Cada uno de estos tiene su relación con los objetos y su estado actual.

El sector *WorldState* es una base de datos que tiene un cache de los valores actuales de un grupo de objetos. Facilita lo obtenido de los valores actuales, sin tener que calcular toda la travesía de las transacciones realizadas. Los objetos del WorldState puede ser creados, actualizados o eliminados, por lo que el *WorldState* se encuentra en constante cambio.

El sector *Blockchain* es un registro de todas las transacciones realizadas en el *WorldState*. Todas las transacciones se colectan en bloques, permitiendo tener un historial de cambios hasta llegar al valor actual del *WorldState*. La estructura de datos del sector *Blockchain* es muy diferente al *WorldState*, debido a que una vez escritos cambios, estos no se pueden modificar (son inmutables).

1.3.4. Servicio Orderer

El servicio *Orderer* [29] está compuesto por 1 o más nodos, es el encargado de realizar el consenso. En plataformas públicas como Bitcoin o Ethereum cualquier nodo miembro puede participar en el consenso, ordenar las transacciones y crear un bloque. Debido a esto, las plataformas públicas utilizan algoritmos de consensos probabilísticos. *Hyperledger Fabric* trabaja de forma diferente, dispone de un servicio *Orderer* que se encarga de ordenar las transacciones y generar el bloque, utilizando un algoritmo de consenso determinístico donde todo bloque generado por el servicio *Orderer* se garantiza que es final y correcto.

Hyperledger dispone de 3 tipos de servicios *Orderer*:

- **Solo**, se realiza con un solo nodo y no es tolerante a falla. Esta configuración se la toma para testeo de configuraciones, pruebas de *Smart Contract* y pruebas de concepto.
- **Raft**, se configura con varios nodos de la red, es un servicio con Tolerancia a falla (Crash Fault Tolerant - CFT) basado en el protocolo Raft y funciona con el modelo "líder y seguidor", donde se elige un nodo líder y que replica las decisiones a los seguidores.
- **Kafka**, similar a la configuración Raft, está basado en Apache Kafka. Es un servicio con CFT y funcional con el modelo "líder y seguidor". Kafka utiliza la unidad ZooKeeper para administra la coordinación de tareas, miembros, control de acceso, entre otras configuraciones del servicio.

Raft y Kafka son CFT, es decir, si existen 3 nodos en el canal, esta puede resistir la pérdida de un nodo, dejando dos nodos activos. Si existen 5 nodos en el canal, se puede perder dos nodos, dejando 3 nodos activos. La pérdida puede ser un nodo miembro o el líder. En el caso que el líder sea el nodo perdido, los nodos restantes seleccionaran un nuevo líder para continuar con las operaciones normales.

1.3.5. SDK Fabric

El Kit de desarrollo de software (SDK) de *Fabric* facilita que los programadores conecten sus aplicativos con los Peers, invoquen transacciones, envíen transacciones, y recibirán eventos cuando se cumple el proceso.

2. ASPECTOS METODOLÓGICOS

La Investigación de Sistemas de Información [30] (Information Systems Research - ISR) es utilizada como métodos de investigación para explorar fenómenos de interés. Existen varios métodos asociados a ISR, todos ellos han sido desarrollados por miembros de la comunidad de Sistemas de Información expertos en áreas particulares.

La Investigación de la Ciencia de Diseño (Design Research - Design Science Research), parte de ISR, es utilizada para la presente investigación. Como resultado de esta metodología se obtiene un artefacto. Vaishnavi [31] define a los artefactos como "*Algoritmos, interfaces humano/computador y metodologías o lenguajes de diseño de sistemas, logrado como resultado de la investigación con Design Science Research"*.

La investigación tiene como objetivo concluir la factibilidad de implementar la tecnología *Blockchain* en redes IoT y proponer un modelo de alto nivel para la implementación de la tecnología Blockchain. Siguiendo la metodología "Design Science Research Methodology (DSRM)" [32] (Figura 2) se realizó la creación y posterior evaluación de un artefacto de TI, manteniendo como objetivo principal la creación de un artefacto útil y efectivo.

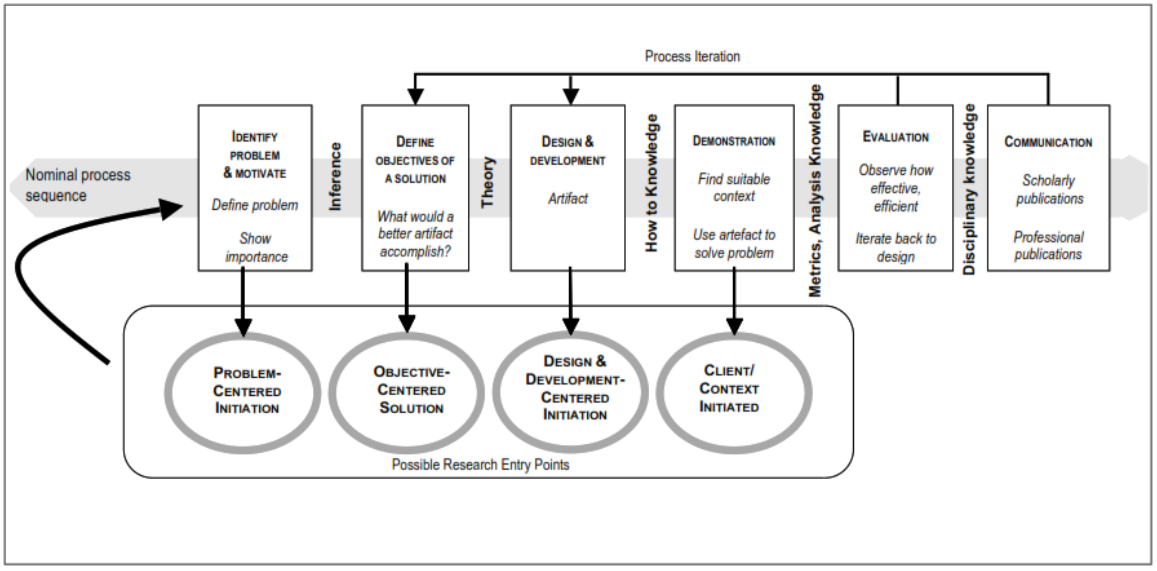


Figura 2: Design Science Research Methodology (DSRM) [32].

Fase 1: "Problem Identification and Motivation"

La investigación se enfocó en el siguiente problema: "Los dispositivos Raspberry Pi (IoT) soportan la integración a la red *Blockchain* como nodos aprobadores de transacciones".

Dorri [33] consideró Blockchain como una solución a los problemas de seguridad y privacidad en redes IoT, debido a que la tecnología utiliza criptografía para asegurar la información y Contratos Inteligentes (*Smart Contracts*) para identificar a usuarios.

Fase 2: "Objectives of a Solution"

El objetivo de esta investigación es analizar la factibilidad de la implementación de la plataforma *Hyperledger Fabric* en redes IoT. Para esto se comprende las siguientes actividades:

- Identificar los principios de funcionamiento de la tecnología *Blockchain*.
- Describir el funcionamiento básico de la plataforma *Hyperledger Fabric*.
- Implementar un prototipo IoT con las tecnologías *Blockchain* y MQTT.
- Comprobar la funcionalidad de la plataforma *Hyperledger* en la red IoT-MQTT mediante mediciones y pruebas de rendimiento de procesador de los equipos, utilización de inyectores de tráfico, pruebas de funcionamiento y pruebas de estrés de transacciones.
- Proponer un esquema de implementación de alto nivel utilizando la plataforma *Hyperledger Fabric* para redes IoT.

Fase 3: "Design and Development"

Gracias a la revisión literaria realizada y la experiencia de profesionales en el campo, compartida por medio de las publicaciones de artículos científicos, libros y otros materiales relacionados a sus investigaciones, fueron de gran ayuda para el diseño y desarrollo de la presente investigación. La etapa de diseño y desarrollo se enfocó en las siguientes subetapas:

- Identificación de las plataformas Blockchain aplicables a los dispositivos IoT.
- Selección y aplicación de una plataforma Blockchain en una red de IoT compuesta por un Raspberry Pi 2B+ y servidores de Amazon Web Services (AWS).

La plataforma *Hyperledger* fue seleccionada para ser implementada en el desarrollo de la investigación, principalmente debido a dos ventajas relacionadas con los objetivos de esta investigación:

- Plataforma de código abierto, organizada por *The Linux Fundation* y desarrollada por una amplia comunidad.

- Multi-arquitectura, *Hyperledger Fabric* dispone de soporte para varias arquitecturas desde equipos de escritorio hasta dispositivos IoT tales como amd64, s390x, ppc64le y armv7.

Hyperledger tiene como objetivo implementar Blockchain en operaciones de negocio, considerando que su desarrollo no está orientado hacia criptomonedas, como lo están gran parte de las plataformas Blockchain. Por lo que se realizó adicionalmente el siguiente proceso:

- Identificación de los módulos de la plataforma *Hyperledger*.
- Selección de los módulos necesarios.
- Aplicación de la plataforma *Hyperledger* en una red IoT.

Fase 4: "Demonstration"

El propósito de esta fase fue demostrar si *Hyperledger* se puede implementar en un equipo IoT (Raspberry Pi 2B+) como un nodo Peer de la red *Blockchain* en la plataforma Hyperledger Fabric. Para cumplir con esta etapa, se obtuvo información que demuestra que la plataforma *Hyperledger* se encontraba funcionando acorde a la configuración propuesta. Se realizaron transacciones entre los nodos miembros de la red. Posteriormente se verificó que la red se mantenga estable y funcional al generar las transacciones.

Fase 5: "Evaluation"

Cleven [34] propone que para obtener utilidad de un artefacto desarrollado en base a Design Science, es necesario dos requisitos fundamentales, que son: "relevancia" y "rigor". La relevancia del artefacto atiende una necesidad comercial real, mientras que el rigor requiere que el investigador aplique adecuadamente el conocimiento existente.

La relevancia del modelo de alto nivel (artefacto), resultado de esta investigación, atiende a la necesidad de implementar Blockchain para proteger la información y descentralizar las redes IoT. De esta manera se protege los servicios que ofrece IoT, de ataques a un solo servidor (un solo punto de falla), además de distribuir la carga computacional, a través de miles de dispositivos; el rigor del artefacto se cumple al implementar los conocimientos obtenidos de las tecnologías Blockchain, IoT, MQTT en la creación del artefacto.

Para evaluar el artefacto se realizó el análisis de los datos recabados para verificar que la implementación cumpla con los requerimientos de la investigación.

En la evaluación del artefacto se recabó los siguientes datos:

- Logs que demuestren la comunicación y funcionamiento de la red *Blockchain* y del protocolo MQTT.
- Captura de paquetes intercambiados entre los miembros de la red tanto para MQTT como para la red *Blockchain*, utilizando el software sniffer de red "*TShark"*.
- Mediciones de ancho de banda, utilizando el software *iPerf*.
- Medición del rendimiento del procesador de los equipos, utilizando el software *Htop*.

Para las pruebas realizadas se tomó en cuenta distintas configuraciones que se pueden realizar en la red *Blockchain* de la plataforma *Hyperledger*. Entre las principales configuraciones se tomaron las siguientes:

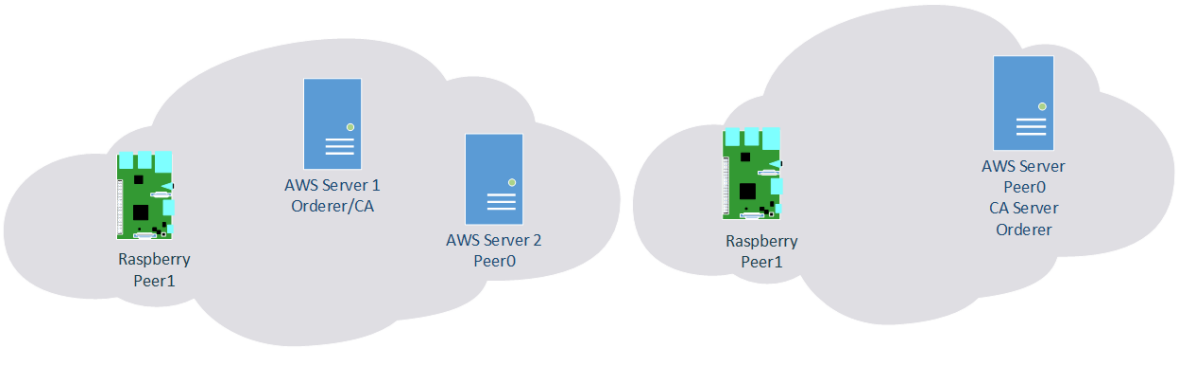
- El tiempo de espera para la creación de un nuevo bloque.
- El número máximo de transacciones en cada bloque.

Se realizó pruebas de estrés, al incrementar el número de transacciones por segundo que reciben los nodos, hasta encontrar un máximo de transacciones, antes que el servicio presente problemas o deje de responder.

Fase 6: "Communication"

La publicación de los resultados de la investigación se ha expuesto en la publicación realizada en ICAT2019 [35], y un artículo enviado a revisión a *International Journal of Information Security and Privacy (IJISP)*.

2.1. Sistema Implementado



a) b)

Figura 3: Modelo Base.

La Figura 3 expone los sistemas utilizados en la investigación. Se realizó un modelo comparativo para comprobar la carga de los servicios *Peer* y *Orderer*. Para el primer

modelo, el servidor AWS correrá los servicios de *Peer*, *Orderer* y CA, mientras que, para el segundo modelo, se separa los servicios de *Peer* y *Orderer*, con el propósito de comprobar la carga de cada servicio.

Los servidores en AWS cuentan con las siguientes características:

- Servidor 1: Es un EC2 t2.micro, que dispone de 1 CPU virtual de hasta 3.3 GHz, 1 GiB de RAM
- Servidor 2: Es un EC2 t2.medium, que dispone de 2 CPU virtual de hasta 3.3 GHz, 4 GiB de RAM.

El servidor 2 se utilizó en ambos modelos para correr el servicio *Orderer*. En el primer modelo el servidor 2 corría adicionalmente el servicio *Peer*.

Para generar las transacciones en el equipo IoT (Raspberry PI 2B+) se utilizó el software Node-Red, que permite la programación por bloques para la obtención de la información y la posterior conexión con el servidor REST. El servidor REST se encarga de distribuir las peticiones a todos los *Peers*.

El dispositivo Raspberry Pi 2B+, adicionalmente a cumplir su rol *Peer* en la red *Blockchain*, trabajó como un dispositivo IoT que enciende/apaga una lámpara a través de los puertos de entrada/salida de uso general (GPIO).

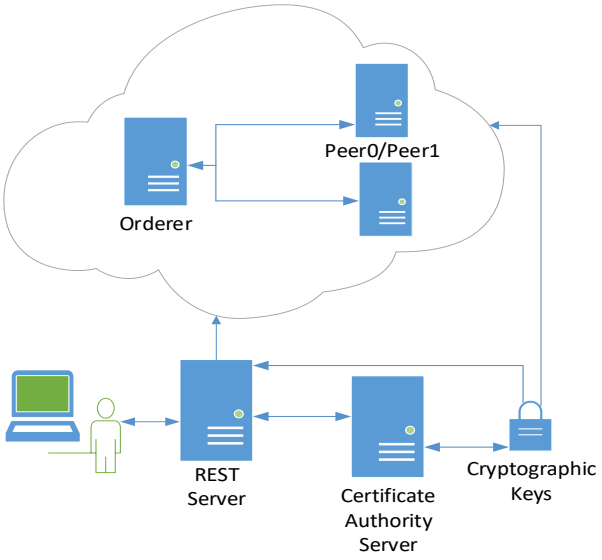


Figura 4: Modelo Red Blockchain.

En la Figura 4 se observa la arquitectura *Blockchain* utilizada en la plataforma *Hyperledger Fabric*. La plataforma *Fabric* permite implementar *Blockchain* utilizando imágenes Docker para levantar sus servicios.

El usuario se conecta por medio del servicio REST con los nodos verificadores para invocar transacciones o consultar información. El servicio REST, en la presente investigación, presenta una invocación y una consulta. La invocación agrega o modifica información a través de transacciones. La consulta permite obtener el estado actual de la cadena de bloques.

Un *MQTT bróker* está configurado en el servidor de AWS para gestionar la comunicación entre Raspberry PI y el servidor de AWS. Se aplicó una protección comparativa para proteger los cambios no deseados en MQTT. Comparando los estados del *MQTT bróker* y *Blockchain* para evitar cambios no autorizados en el estado de la lámpara.

Los servicios de la plataforma *Hyperledger Fabric* disponen de configuraciones para el inicio de los contenedores (Docker) y en las características del canal (Channel) y el *Smart Contract* (o Chaincode).

En las configuraciones de los contenedores se coloca las rutas de los archivos generados para las configuraciones de las demás características, IPs de los servicios importantes de la red como el *Orderer* y CA.

En las configuraciones del canal se define los servicios *Orderer* y las Organizaciones que van a formar parte de la red. Y dentro de las Organizaciones se definen los Peers. Se genera los certificados por medio del servicio CA para realizar todas las transacciones de la red y las firmas digitales que se guardaran en los bloques.

En la investigación se realizó una comparativa entre dos escenarios, utilizando los modelos definidos al inicio de esta sección y las configuraciones del canal, siendo los siguientes:

- Modelo 1 utilizando las configuraciones predeterminadas: generando un nuevo bloque cada 2 segundos con un máximo de 100 transacciones por bloque.
- Modelo 2 utilizando configuraciones personalizadas, generando un nuevo bloque cada 10 minutos (600 segundos) o un máximo de 200 transacciones por bloque.

El escenario 2 se basa en la red Bitcoin, en la que se generan los bloques cada 10 minutos.

3. RESULTADOS Y DISCUSIÓN

La implementación y las pruebas realizadas exponen los resultados obtenidos que se detallan en el presente acápite.

3.1. Implementación

La compilación e implementación de Hyperledger Fabric en sus versiones actuales (>1.2), en un Raspberry Pi 2B+, presentan varias dificultades. Se debe a que las nuevas versiones del dispositivo son desarrolladas para la utilización en sistemas con arquitectura de 64 bits; por lo tanto, un dispositivo con arquitectura armv7l de 32 bits presenta problemas en su aplicación.

La implementación se realizó con Fabric v1.0 compilada por Joe Motacek [36]; Joe en su sitio web, expone una solución para implementar Hyperledger Fabric en dispositivos Raspberry Pi con arquitectura armv7l, la que se adaptó para las necesidades de la presente investigación.

Figura 5: Estado encendido/apagado.

La programación por bloques de Node-Red en AWS se observa en la Figura 6. Los bloques se encargan de generar la información de la transacción obtenida desde una interfaz gráfica, se envía la petición HTTP al servidor REST y procesa la respuesta. Si la respuesta es afirmativa se realiza el cambio en el broker MQTT.

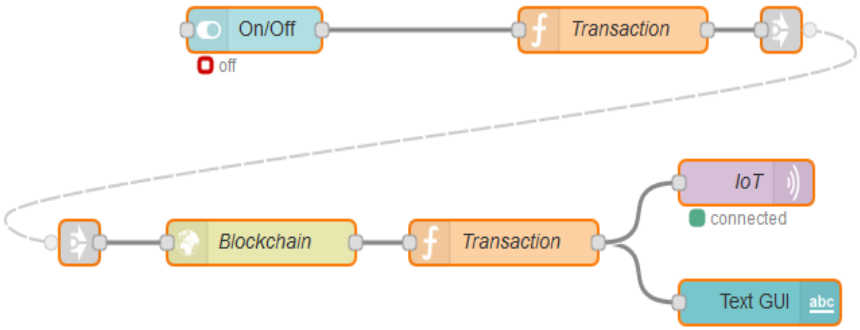


Figura 6: Programación Node-Red en AWS

En el lado del Raspberry, la programación de bloques en Node-Red (ver Figura 7) recibe el cambio introducido en el broker MQTT y realiza una petición HTTP al servidor REST para conocer el estado actual de Blockchain; efectúa una comparativa entre MQTT y

Blockchain para finalmente realizar los cambios en el puerto GPIO. La comparativa se realiza con el fin de evitar que cambios no autorizados se produzcan en el estado de los puertos GPIO.

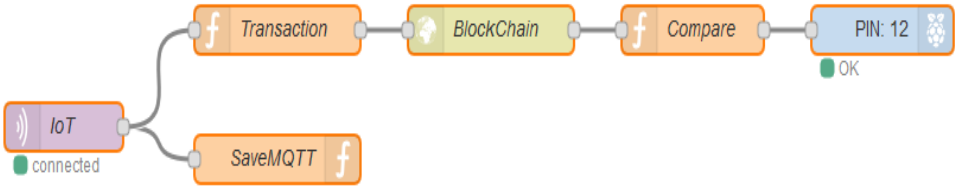


Figura 7: Programación Node-Red en Raspberry

3.2. Operación básica

El funcionamiento de las redes Blockchain e IoT-MQTT se demostró mediante paquetes de red capturados con la herramienta TShark, donde se muestra el paquete de comunicación entre nodos (Figura 8).

No.	Time	Source	Destination	Protocol	Port	Info
131	9.220806735	3.17.180.159	10.10.0.212	TCP	7051	7051 → 42444 [PSH, ACK] Seq=5750 Ac…
132	9.221175849	10.10.0.212	3.17.180.159	TCP	42444	42444 → 7051 [ACK] Seq=5758 Ack=614…
133	9.308426901	3.17.180.159	10.10.0.212	TCP	7051	7051 → 42444 [ACK] Seq=6148 Ack=569…
134	9.383814810	3.17.180.159	10.10.0.212	TCP	7051	7051 → 42444 [ACK] Seq=6148 Ack=575…
135	9.406026101	3.17.180.159	10.10.0.212	MQTT	1883	Publish Message (id=51571) [tesis/1…
136	9.407163702	10.10.0.212	3.17.180.159	MQTT	37674	Publish Received (id=51571)
137	9.531000900	3.17.180.159	10.10.0.212	TCP	1883	1883 → 37674 [ACK] Seq=24 Ack=5 Win…
138	9.531019441	3.17.180.159	10.10.0.212	MQTT	1883	Publish Release (id=51571)
139	9.533489486	10.10.0.212	3.17.180.159	MQTT	37674	Publish Complete (id=51571)
140	9.576925508	10.10.0.212	3.17.180.159	TCP	42444	42444 → 7051 [PSH, ACK] Seq=5758 Ac…
141	9.692590593	3.17.180.159	10.10.0.212	TCP	1883	1883 → 37674 [ACK] Seq=28 Ack=9 Win…
142	9.696898081	3.17.180.159	10.10.0.212	TCP	7051	7051 → 42444 [ACK] Seq=6148 Ack=580…
143	9.696914852	3.17.180.159	10.10.0.212	TCP	7051	7051 → 42444 [PSH, ACK] Seq=6148 Ac…
144	9.697319017	10.10.0.212	3.17.180.159	TCP	42444	42444 → 7051 [ACK] Seq=5802 Ack=621…
145	9.776120927	3.17.180.159	10.10.0.212	TCP	7051	7051 → 42444 [PSH, ACK] Seq=6213 Ac…

No.	Time	Source	Destination	Protocol	Port	Info
3552882	311295.81457…	10.10.0.212	3.17.180.159	TCP	40584	40584 → 7051 [PSH, ACK] Seq=9492815…
3552883	311295.93369…	3.17.180.159	10.10.0.212	TCP	7051	7051 → 40584 [ACK] Seq=94266971 Ack…
3552884	311295.94995…	10.10.0.212	3.17.180.159	TCP	40584	40584 → 7051 [PSH, ACK] Seq=9492825…
3552885	311296.06893…	3.17.180.159	10.10.0.212	TCP	7051	7051 → 40584 [ACK] Seq=94266971 Ack…
3552886	311296.14224…	3.17.180.159	10.10.0.212	MQTT	1883	Publish Message (id=3875) [tesis/la…
3552887	311296.14341…	10.10.0.212	3.17.180.159	MQTT	54084	Publish Received (id=3875)
3552888	311296.25920…	3.17.180.159	10.10.0.212	TCP	1883	1883 → 54084 [ACK] Seq=107110 Ack=3…
3552889	311296.25922…	3.17.180.159	10.10.0.212	MQTT	1883	Publish Release (id=3875)
3552890	311296.26140…	10.10.0.212	3.17.180.159	MQTT	54084	Publish Complete (id=3875)
3552891	311296.32077…	3.17.180.159	10.10.0.212	TCP	7051	7051 → 40584 [PSH, ACK] Seq=9426697…
3552892	311296.32119…	10.10.0.212	3.17.180.159	TCP	40584	40584 → 7051 [ACK] Seq=94928531 Ack…
3552893	311296.34274…	10.10.0.212	3.17.180.159	TCP	40584	40584 → 7051 [PSH, ACK] Seq=9492853…
3552894	311296.39472…	10.10.0.212	3.17.180.159	TCP	40584	40584 → 7051 [PSH, ACK] Seq=9492857…
3552895	311296.41869…	3.17.180.159	10.10.0.212	TCP	1883	1883 → 54084 [ACK] Seq=107114 Ack=3…
3552896	311296.45843…	3.17.180.159	10.10.0.212	TCP	7051	7051 → 40584 [ACK] Seq=94267262 Ack…

Figura 8: Paquetes de red Blockchain y MQTT.

3.3. Rendimiento

En base a los resultados obtenidos, se ha generado gráficas y tablas para el análisis de la información, las mismas que se presentan a continuación.

3.3.1. Primer escenario

La Figura 9 y la Figura 10 muestran el uso del procesador en función del tiempo de los dispositivos Raspberry y AWS respectivamente, en cada una de las pruebas de transacciones realizadas.

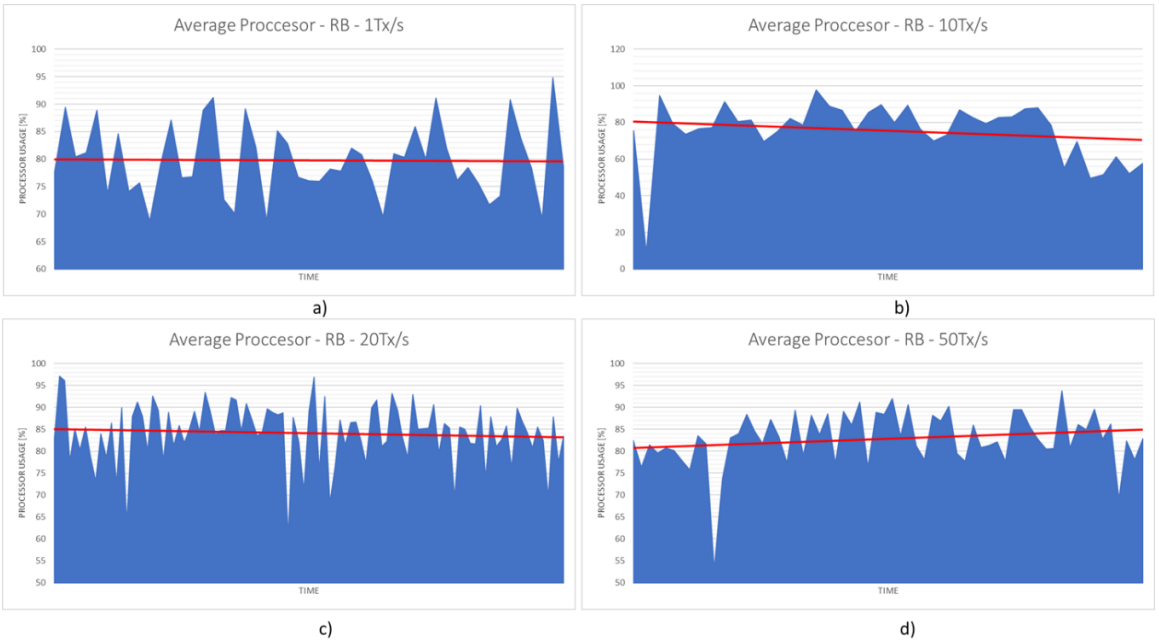


Figura 9: Uso procesador del Raspberry Pi en primer escenario.

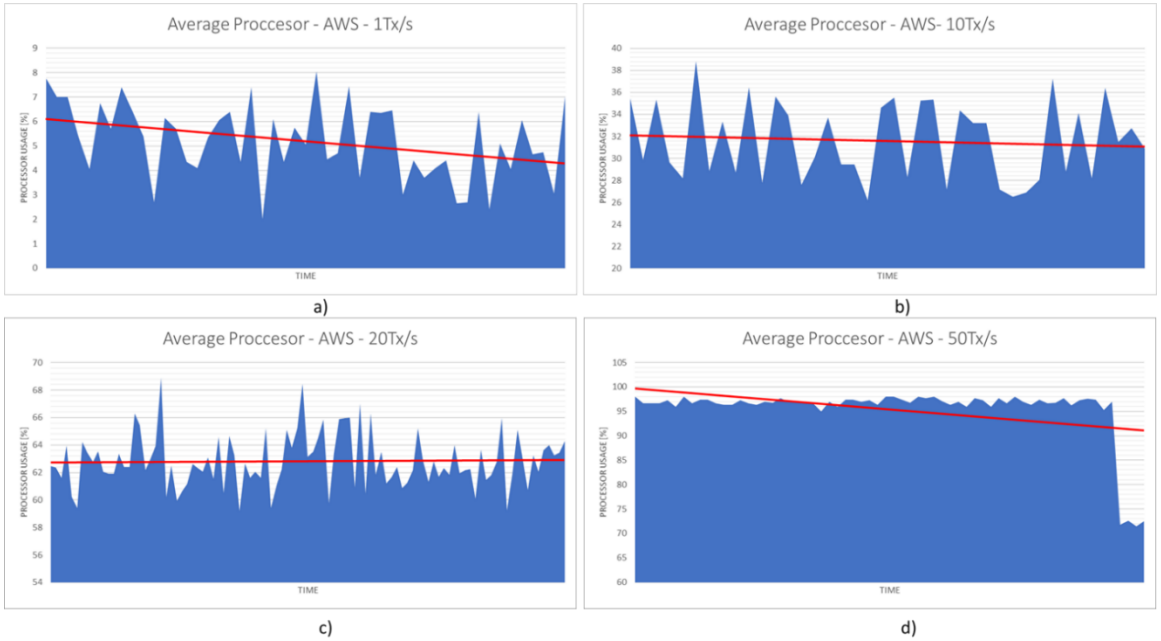


Figura 10: Uso procesador de AWS en primer escenario.

La Tabla 2 expone los valores promedio del rendimiento del procesador en el primer escenario definido. En la tabla se puede apreciar que el dispositivo Raspberry, en

promedio, utiliza sus procesadores en una cantidad similar para cada prueba. Esto se debe a la cantidad de bloques que se generan por segundo. En la configuración se generaba un bloque cada 2 segundos con un máximo de 100 transacciones, por lo que la carga que llegaba al equipo era grande, además de procesar la aprobación de las transacciones en conjunto con el *Peer* de AWS. De igual manera, a medida que se aumentaron las transacciones, también se puede apreciar que Raspberry utiliza la memoria SWAP, lo que le permite mejorar su funcionamiento al disminuir la carga en la memoria RAM.

El servidor AWS presenta un aumento en la carga del procesador directamente proporcional al aumento de transacciones que llegan hasta las 50 transacciones por segundo, es cuando los servicios de *Fabric* que se encontraban en este servidor se reiniciaron debido a la sobrecarga del procesador.

Tabla 2: Rendimiento – primera configuración.

Tx/s	**Raspberry Pi (*Peer*)**			**AWS (*Peer* – *Orderer*)**	
	Procesador	**RAM**	**Swap**	**Procesador**	**RAM**
1	Promedio 80%	31.70%	0%	14%	14.31%
10		36.69%	16.07%	33%	17.09%
20		36.43%	39.55%	63%	18.22%
50	Servicios de *Hyperledger* fallaron				

3.3.2. Segundo escenario

La Figura 11, Figura 12 y Figura 13 exponen el uso del procesador en función del tiempo de los dispositivos Raspberry y los servidores AWS-*Peer* y AWS-*Orderer* respectivamente, en cada una de las pruebas de transacciones realizadas.

En las Figura 11 y Figura 12 se presenta el rendimiento en los equipos que tienen los servicios Peers, y se observa que en las pruebas de 1 Tx/s existen picos en los que el procesamiento sube en gran medida; se debe a que en esos tiempos se generaron nuevos bloques y se aumentó el procesamiento ya que recibe el bloque, lo procesa y lo almacena.

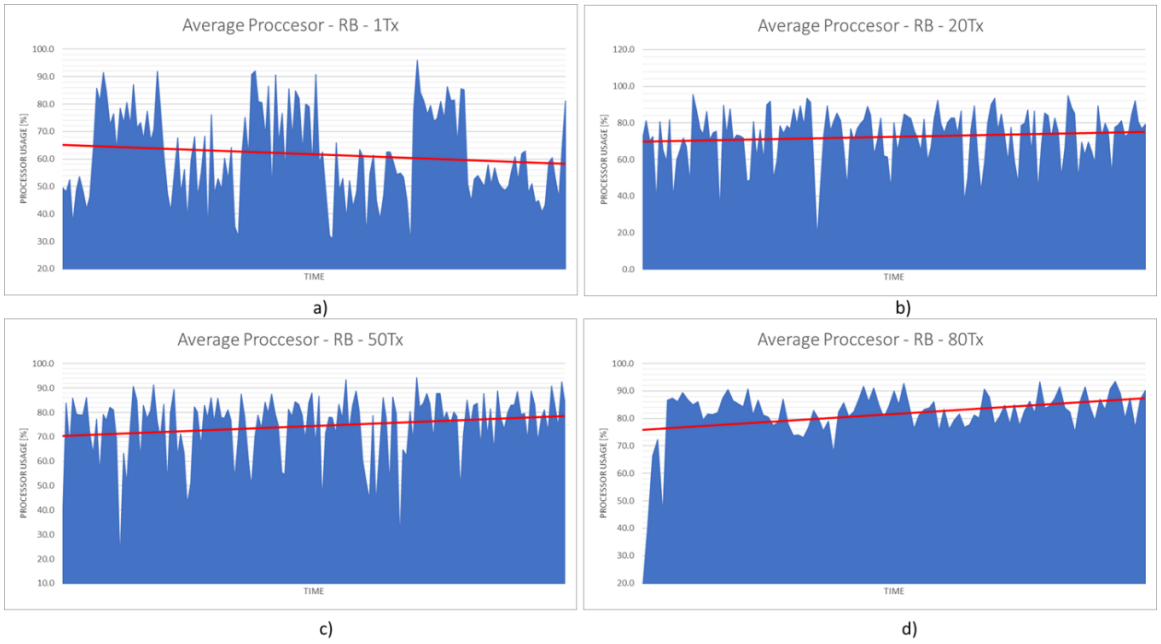


Figura 11: Uso procesador de Raspberry en segundo escenario.

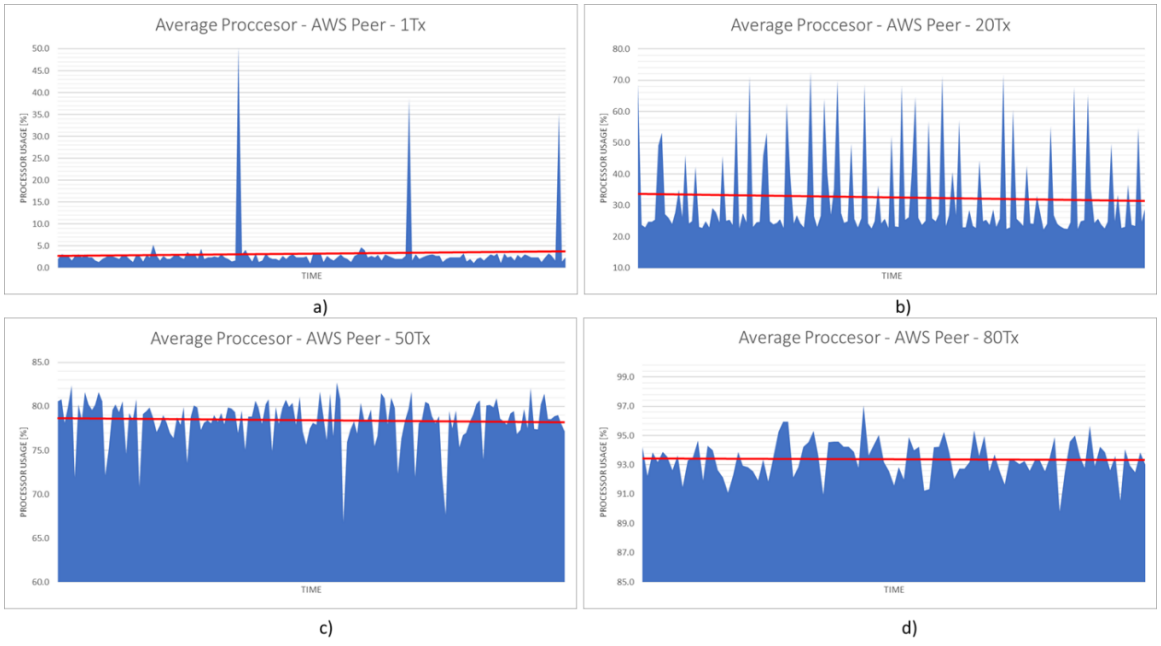


Figura 12: Uso procesador de AWS-Peer en segundo escenario.

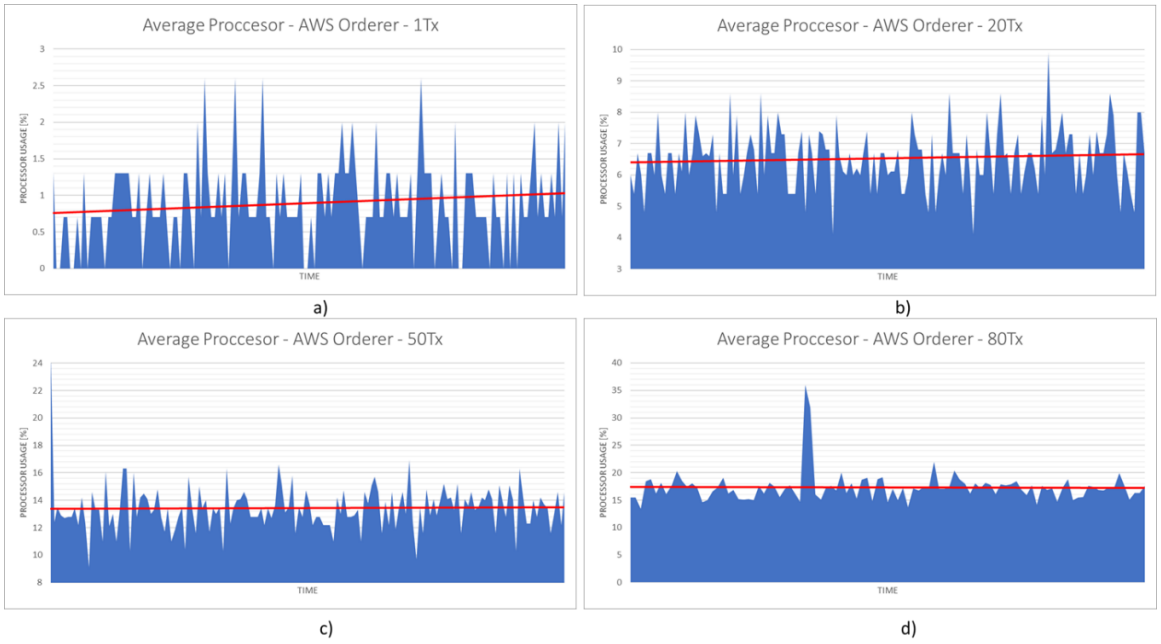


Figura 13: Uso procesador de AWS-Orderer en segundo escenario.

La Tabla 3 muestra los valores promedio del rendimiento del procesador en el segundo escenario, en el que los servicios de *Peer* y *Orderer* se encuentran divididos en diferentes servidores, además que la configuración cambia en la generación de bloques.

La carga computacional en el Raspberry es mucho menor que en el primer escenario, debido a que se reduce drásticamente la cantidad de bloques que se procesan por segundo y el equipo se encarga de procesar las aprobaciones de las transacciones de la red, más no procesar los bloques recibidos desde el *Orderer*.

El servicio *Peer* en AWS presenta un aumento exponencial en el uso de su procesador, llegando a soportar hasta 80 transacciones por segundo en la red. Por lo que se observa una mejora en su rendimiento en comparación del primer escenario.

El servidor con los servicios *Orderer* y CA no presenta una carga mayor al procesador, a pesar de tener menores características computacionales que el AWS *Peer*. Debido a que la generación de los bloques no es grande por los tiempos de generación de los bloques establecidos.

En las pruebas de 50 y 80 transacciones por segundo, se puede apreciar un aumento grande, pero aún no se llega a ocupar ni la mitad de los recursos de procesador. En estas pruebas realizadas, por la cantidad de transacciones, el tiempo de generación de bloques baja a 4 y 3 segundos respectivamente, ya que cumple la condición de 200 transacciones máximo por bloque y se genera un bloque con esa cantidad de transacciones.

Tabla 3: Rendimiento – segunda configuración.

Tx/s	Raspberry Pi (*Peer*)			AWS (*Peer*)		AWS (*Orderer*)	
	Procesador	RAM	SWAP	Procesador	RAM	Procesador	RAM
1	61.68%	33.96%	74.49%	3.24%	14.91%	1.08%	33.60%
20	72.43%	42.72%	78.39%	32.54%	19.96%	6.53%	34.00%
50	74.52%	52.38%	92.11%	78.41%	25.93%	13.44%	34.66%
80	81.60%	55.40%	68.42%	93.40%	30.73%	17.33%	34.36%

En general, se demuestra una disminución significativa en el uso de los recursos de procesamiento de los dispositivos, lo que permitiría llegar a aumentar las transacciones por segundo procesadas en gran número.

Finalmente, el servidor AWS con los servicios *Orderer* y CA no presenta gran carga, debido a que este servicio se encarga exclusivamente de guardar las transacciones hasta que cumpla una de las condiciones y genere un bloque. Con la primera configuración el servicio generaba bloques cada 2 segundos y los distribuía en la red, con la segunda configuración la generación de bloques se reduce drásticamente y la carga computacional es ínfima.

3.4. Ancho de Banda

Las mediciones de ancho de banda se realizaron en conjunto con las pruebas de rendimiento. Lo resultados obtenidos se muestran en las Figura 14 y Figura 15, en la que se grafica el ancho de banda en función del tiempo.

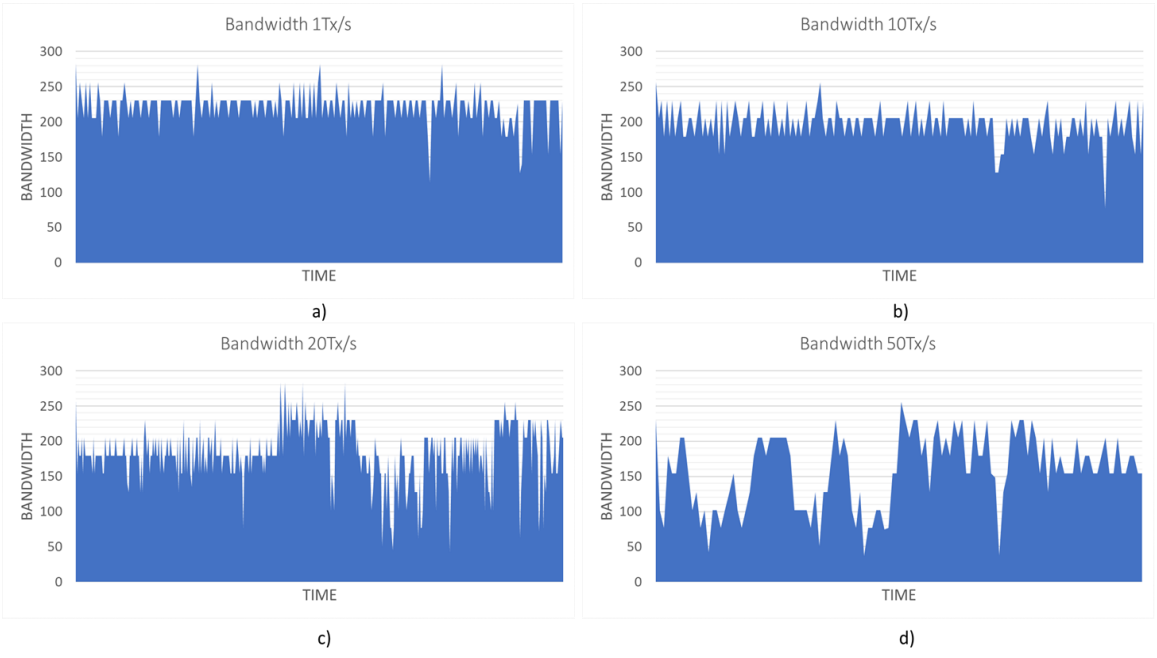


Figura 14: Ancho de banda obtenido en primer escenario.

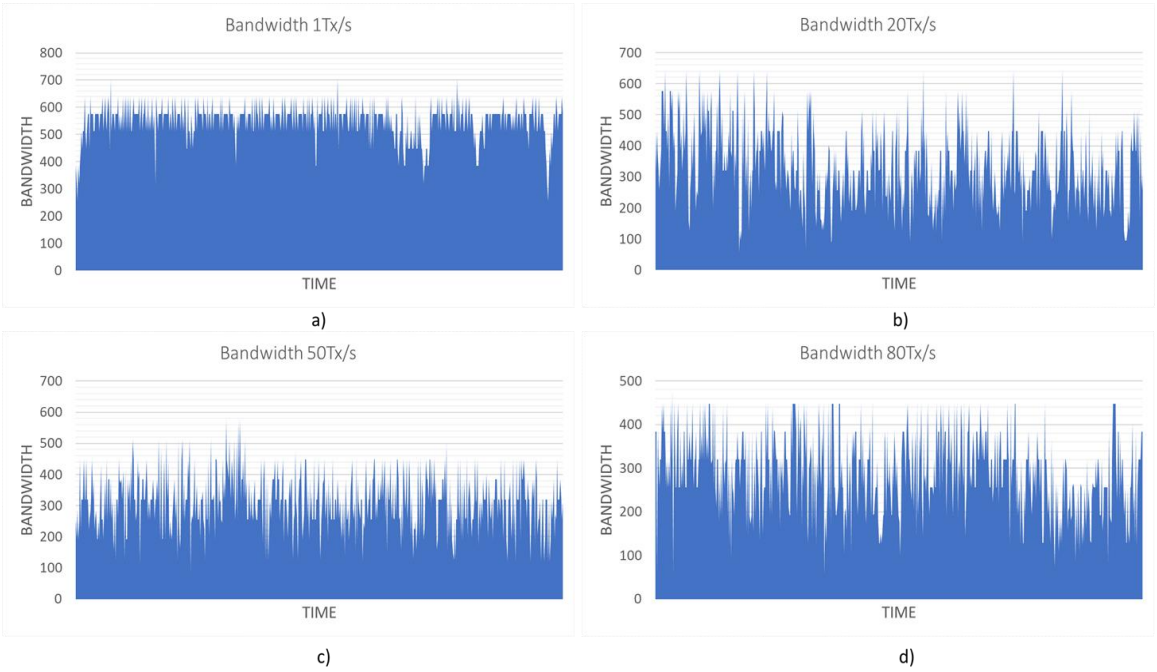


Figura 15: Ancho de banda obtenido en segundo escenario.

Se ha realizado un resumen comparativo en la Tabla 4, en la tabla se presenta los valores máximo, promedio y mínimo de ancho de banda en cada prueba. Se puede apreciar que el ancho de banda disponible aumenta con la segunda configuración; el aumento en el ancho de banda disponible se debe a que existe menor cantidad de intercambio de paquetes de red con información de bloques y notificaciones en la red *Blockchain* por parte del servicio *Orderer*.

Tabla 4: Medición ancho de banda, configuración 1.

	Modelo 1					**Modelo 2**			
	<1Tx/s	1 Tx/s	10 Tx/s	20 Tx/s	50 Tx/s	1 Tx/s	20 Tx/s	50 Tx/s	80 Tx/s
Máximo [Kbps]	282	282	256	282	256	704	640	576	473
Promedio [Kbps]	230	221	196	183	158	550	338	315	291
Mínimo [Kbps]	205	114	76.8	42.7	37.2	256	59	90	58.1

3.5. Memoria ROM

El tamaño de los bloques generados por el servicio *Orderer* depende de la cantidad de transacciones que se almacenen en cada bloque. El tamaño no presenta una tendencia lineal, y el tamaño de cada bloque dependerá de las transacciones, firmas de los Peers, encabezados y toda la información que se guardará en el archivo del bloque.

Entre los bloques obtenidos en las pruebas, se logra encontrar los siguientes valores de tamaño.

- Un bloque con 1 transacción ocupa aproximadamente 5Kb.
- Un bloque con 40 transacciones ocupa aproximadamente 115Kb.
- Un bloque con 200 transacciones ocupa aproximadamente 560Kb.

En la tabla 4 se presenta un aproximado de transacciones y bloques con las que se podría llenar una memoria SD (utilizadas en Raspberry) en sus diferentes capacidades. Para disponer una referencia de la cantidad de información que se podría almacenar en una red *Blockchain* con la limitante de la memoria ROM de un equipo IoT actual.

Tabla 5: Transacciones y bloques aproximados para ocupar completamente memorias de 16Gb, 32Gb y 64Gb.

Tx / tamaño bloque	**13 Gb**		**28 Gb**		**56 Gb**	
	Tx	**Bloques**	**Tx**	**Bloques**	**Tx**	**Bloques**
1Tx / 5 Kb	2.726.298	2.726.298	5.872.026	5.872.026	11.744.051	11.744.051
40Tx / 115 Kb	4.741.387	118.535	10.212.218	255.305	20.424.437	510.611
200Tx / 560 Kb	4.868.389	24.342	10.485.760	52.429	20.971.520	104.858

3.6. Modelo

Como resultado de la investigación se propone un esquema de alto nivel para implementaciones de la plataforma *Hyperledger Fabric* en redes IoT con dispositivos de capacidades tecnológicas similares o mejores a los dispositivos utilizados en el desarrollo de la investigación (ver Figura 16). Donde el usuario interactúa por medio de un aplicativo con el servidor REST, el mismo que realizara las consultas a la red Blockchain. El modelo presenta una propuesta en la que cada servicio de Hyperledger Fabric (Peer, Orderer, CA) sea desplegado en servidores o dispositivos IoT con la finalidad de distribuir la carga computacional, evitando así centralizar todo el procesamiento en un servidor central. En el modelo definido, se debe especificar que tanto el servicio Orderer como el servidor CA pueden ser desplegados como clúster de servidores en función de sus configuraciones. De igual manera se puede conectar varios aplicativos o servidores REST, cada uno se debe registrar como una identidad nueva en el servidor CA para que se autorice la interacción con la red Blockchain.

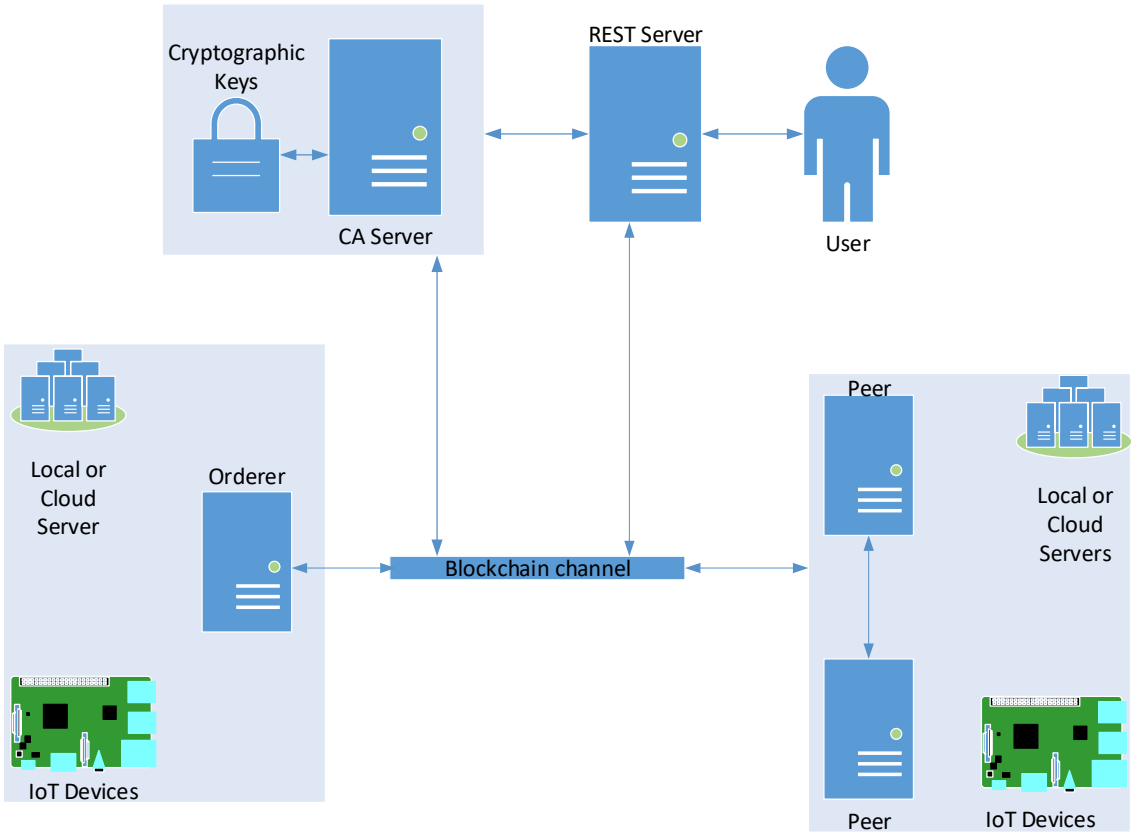


Figura 16: Modelo de alto nivel para implementar Hyperledger en un equipo Raspberry Pi

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

El dispositivo Raspberry Pi 2B+ utilizado en el desarrollo de la investigación dispone de la potencia computacional necesaria para formar parte de una red *Blockchain* como un nodo aprobador de transacciones.

Raspberry Pi 2B+ tiene un gran consumo de recursos de procesador, incluso en las pruebas básicas del escenario, con generación de bloques cada 2 segundos. Por lo cual, los ventiladores o algún sistema de enfriamiento son necesarios para evitar el sobrecalentamiento del procesador y evitar el reinicio o daños en el dispositivo.

El uso de recursos de procesador, ancho de banda, memoria RAM y memoria ROM de los dispositivos presentan un mejor rendimiento en el escenario con generación de bloques cada 10 minutos, gracias a las distintas configuraciones, lo que tiene como consecuencia una carga menor en la red *Blockchain*. El dispositivo IoT no era responsable de crear bloques de datos para conceder seguridad de la información, de hecho, el dispositivo IoT sólo estaba respaldando al Peer y manteniendo una copia de Ledger.

Se debe considerar que mientras el bloque no sea generado, las transacciones se encuentran aprobadas por los Peers, pero no actualizadas en la red *Blockchain*; es decir, se ha comprobado que se puede realizar la transacción, pero no se encuentran actualizados los valores. Por ello, al realizar una consulta de los valores actuales se obtienen los que están actualizados en el último bloque generado.

Debe considerarse que todos los servicios de Hyperledger se ejecutan en máquinas virtuales Docker, lo que representa una seguridad adicional en la implementación, gracias a las ventajas que dispone como: un rápido desarrollo, portabilidad, mínimo uso de recursos y facilidad de despliegue. Adicionalmente, permite reproducir investigaciones de manera fácil con la compartición de las imágenes compiladas.

4.2. Recomendaciones

Hyperledger en sus servicios dispone de varias configuraciones para el método de aprobación de las transacciones en servicio *Orderer*. En la actual investigación se utilizó la configuración “Solo”, la que permite un único servidor *Orderer*. Actualmente *Hyperledger* dispone de las configuraciones adicionales de “Raft” y “Kafka”, las que disponen de tolerancia a las fallas (Crash Fault Tolerant - CFT) y permite distribuir el servicio *Orderer* entre varios dispositivos. La distribución en la carga computacional para la aprobación de las transacciones se puede distribuir a una mayor cantidad de nodos Peers, por lo tanto, se recomienda a futuro investigar una red con un número mayor de dispositivos IoT (Raspberry o similar) para distribuir la carga computacional, al repartir las peticiones entre todos los miembros de la red y soportar un mayor número de transacciones por segundo.

Para realizar una investigación sin tener problemas significativos de las imágenes Docker utilizadas, se recomienda compilar las imágenes en cada dispositivo o en un dispositivo representante si el conjunto de dispositivos consta de la misma arquitectura y librerías instaladas. Con lo que se asegura que la compilación sea para las librerías disponibles del equipo (por arquitectura y versión), y no exista ningún problema al momento de montar las imágenes.

En función de la implementación en la que se requiera utilizar la tecnología *Blockchain* en una red IoT, se deberá estudiar la mejor configuración en el tiempo y cantidad de transacciones por bloque que dispondrá la red, debido a que estas configuraciones afectan en gran medida al rendimiento de los dispositivos.

BIBLIOGRAFÍA

[1] The Statistics Portal, “IoT: number of connected devices worldwide 2012-2025.” [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. [Accessed: 06-Feb-2019].

[2] M. A. Khan and K. Salah, “IoT security: Review, blockchain solutions, and open challenges,” *Futur. Gener. Comput. Syst.*, vol. 82, pp. 395–411, May 2018, doi: 10.1016/j.future.2017.11.022.

[3] D. Minoli and B. Occhiogrosso, “Blockchain mechanisms for IoT security,” *Internet of Things*, vol. 1–2, pp. 1–13, Sep. 2018, doi: 10.1016/j.iot.2018.05.002.

[4] D. Fakhri and K. Mutijarsa, “Secure IoT Communication using Blockchain Technology,” in *2018 International Symposium on Electronics and Smart Devices (ISESD)*, 2018, pp. 1–6, doi: 10.1109/ISESD.2018.8605485.

[5] G.-H. Hwang, P.-H. Chen, C.-H. Lu, C. Chiu, H.-C. Lin, and A.-J. Jheng, “InfiniteChain: A Multi-chain Architecture with Distributed Auditing of Sidechains for Public Blockchains,” 2018, pp. 47–60.

[6] X. Fan, “Faster Dual-Key Stealth Address for Blockchain-Based Internet of Things Systems,” 2018, pp. 127–138.

[7] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, “Blockchain for IoT security and privacy: The case study of a smart home,” in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 618–623, doi: 10.1109/PERCOMW.2017.7917634.

[8] G. D. Salazar Ch, C. Venegas, and L. Marrone, “MQTT-Based Prototype Rover with Vision-As-A-Service (VAAS)in an IoT Dual-Stack Scenario,” in *2019 Sixth International Conference on eDemocracy & eGovernment (ICEDEG)*, 2019, pp. 344–349, doi: 10.1109/ICEDEG.2019.8734341.

[9] M. Singh, A. Singh, and S. Kim, “Blockchain: A game changer for securing IoT data,” in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, 2018, vol. 2018-Janua, pp. 51–55, doi: 10.1109/WF-IoT.2018.8355182.

[10] H. F. Atlam and G. B. Wills, “Intersections between IoT and distributed ledger,” 2019, pp. 73–113.

[11] G. D. Salazar Ch., C. Venegas, M. Baca, I. Rodriguez, and L. Marrone, “Open

Middleware proposal for IoT focused on Industry 4.0," in *2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)*, 2018, pp. 1–6, doi: 10.1109/CCRA.2018.8588117.

[12] S. Zamfir, T. Balan, I. Iliescu, and F. Sandu, "A security analysis on standard IoT protocols," in *2016 International Conference on Applied and Theoretical Electricity (ICATE)*, 2016, pp. 1–6, doi: 10.1109/ICATE.2016.7754665.

[13] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, and A. Panya, "Authorization mechanism for MQTT-based Internet of Things," in *2016 IEEE International Conference on Communications Workshops (ICC)*, 2016, pp. 290–295, doi: 10.1109/ICCW.2016.7503802.

[14] K. Biswas and V. Muthukkumarasamy, "Securing Smart Cities Using Blockchain Technology," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2016, pp. 1392–1393, doi: 10.1109/HPCC-SmartCity-DSS.2016.0198.

[15] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Futur. Gener. Comput. Syst.*, vol. 88, no. 2018, pp. 173–190, Nov. 2018, doi: 10.1016/j.future.2018.05.046.

[16] M. Banerjee, J. Lee, and K.-K. R. Choo, "A blockchain future for internet of things security: a position paper," *Digit. Commun. Networks*, vol. 4, no. 3, pp. 149–160, Aug. 2018, doi: 10.1016/j.dcan.2017.10.006.

[17] H. F. Atlam, A. Alenezi, M. O. Alassafi, and G. B. Wills, "Blockchain with Internet of Things: Benefits, challenges, and future directions," *Int. J. Intell. Syst. Appl.*, 2018, doi: 10.5815/ijisa.2018.06.05.

[18] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016, doi: 10.1109/ACCESS.2016.2566339.

[19] C. Pastorino, "Blockchain: qué es, cómo funciona y cómo se está usando en el mercado," *WeLiveSecurity by ESET*, 2018. [Online]. Available: https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/. [Accessed: 08-Feb-2020].

[20] V. Kalinov and S. Voshmgir, “Blockchain A Beginners guide,” *BlockchainHub*, 2017.

[21] A. Preukschat, “Los tipos de Blockchain: públicas, privadas e híbridas (y II),” 2017. [Online]. Available: https://www.iecisa.com/es/blog/Post/Los-tipos-de-Blockchain-publicas-privadas-e-hibridas-y-II/. [Accessed: 09-Feb-2020].

[22] M. Allende López and V. Colina Unda, “Conoce los distintos tipos de blockchain,” 2018. [Online]. Available: https://blogs.iadb.org/conocimiento-abierto/es/tipos-de-blockchain/. [Accessed: 09-Feb-2020].

[23] M. Beedham, “All you need to know about Bitcoin network nodes,” 2019. [Online]. Available: https://thenextweb.com/hardfork/2019/03/01/bitcoin-blockchain-nodes-network/. [Accessed: 16-Feb-2020].

[24] The Linux Foundation Projects, “Hyperledger – Open Source Blockchain Technologies,” 2016. [Online]. Available: https://www.hyperledger.org/. [Accessed: 15-May-2019].

[25] Hyperledger, “Hyperledger Fabric - Master documentation,” 2019. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/key_concepts.html. [Accessed: 15-May-2019].

[26] Hyperledger, “Peers.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.0/peers/peers.html. [Accessed: 27-Feb-2020].

[27] Hyperledger, “Smart Contracts and Chaincode.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.0/smartcontract/smartcontract.html. [Accessed: 27-Feb-2020].

[28] Hyperledger, “Ledger.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.0/ledger/ledger.html. [Accessed: 27-Feb-2020].

[29] Hyperledger, “The Ordering Service.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.0/orderer/ordering_service.html. [Accessed: 27-Feb-2020].

[30] Association for Information Systems (AIS), “IS Research, Methods, and Theories,” 2010. [Online]. Available: https://aisnet.org/page/ISResearch. [Accessed: 12-Sep-2019].

[31] V. Vaishnavi, B. Kuechler, and S. Petter, *Design Science Research in Information Systems. Advances in Theory and Practice*, vol. 7286. Berlin, Heidelberg: Springer

Berlin Heidelberg, 2012.

[32] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee, “A Design Science Research Methodology for Information Systems Research,” *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–78, 2007, doi: 10.2753/MIS0742-1222240302.

[33] A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak, “BlockChain: A Distributed Solution to Automotive Security and Privacy,” *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 119–125, Dec. 2017, doi: 10.1109/MCOM.2017.1700879.

[34] A. Cleven, P. Gubler, and K. M. Hüner, “Design alternatives for the evaluation of design science research artifacts,” p. 1, 2009, doi: 10.1145/1555619.1555645.

[35] G. Andrade-Salinas, G. Salazar-Chacon, and L.-M. Vintimilla, “Integration of IoT Equipment as Transactional Endorsing Peers over a Hyperledger-Fabric Blockchain Network: Feasibility Study,” in *Applied Technologies*, 2020, pp. 95–109, doi: 10.1007/978-3-030-42517-3_8.

[36] J. Motacek, “Hyperledger Fabric v1.0 on a Raspberry Pi Docker Swarm.” [Online]. Available: https://www.joemotacek.com/hyperledger-fabric-v1-0-on-a-raspberry-pi-docker-swarm-part-1/. [Accessed: 23-Sep-2019].

ANEXOS

ANEXO I Configuración de topología de Hyperledger.

Documento digital: ***configtx.yaml***

```
Organizations:
    - &OrdererOrg
        Name: OrdererOrg
        ID: OrdererMSP
        MSPDir: crypto-config/ordererOrganizations/example.com/msp
    - &Org1
        Name: Org1MSP
        ID: Org1MSP
        MSPDir: crypto-config/peerOrganizations/org1.example.com/msp
        AnchorPeers:
            - Host: peer0.org1.example.com
              Port: 7051
Orderer: &OrdererDefaults
    OrdererType: solo
    Addresses:
        - orderer.example.com:7050
    BatchTimeout: 600s
    BatchSize:
        MaxMessageCount: 200
        AbsoluteMaxBytes: 99 MB
        PreferredMaxBytes: 10 MB
    Kafka:
        Brokers:
            - 127.0.0.1:9092
    Organizations:

Application: &ApplicationDefaults
    Organizations:

Profiles:
    OneOrgOrdererGenesis:
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *Org1
    OneOrgChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1
```

ANEXO II Configuración para generación de Certificados de Organizaciones y Peers.

Documento digital: **crypto-config.yaml**

```
OrdererOrgs:
  - Name: Orderer
    Domain: example.com
    Specs:
      - Hostname: orderer
PeerOrgs:
  - Name: Org1
    Domain: org1.example.com
    Template:
      Count: 2
    Users:
      Count: 1
```

ANEXO III Código del Smart Contract

Documento digital: ***redlan.go***
Documento muy amplio, se incluye una parte del código.

```
package main

import (
	"encoding/json"
	"fmt"
	"strconv"

	"github.com/hyperledger/fabric/core/chaincode/shim"
	pb "github.com/hyperledger/fabric/protos/peer"
)

// Define la estructura del SmartContract
type SmartContract struct {
}

// Define la estructura de Lampara
type Lampara struct {
	LampId		string		`json:"LampId"`
	Estado		bool		`json:"Estado"`
	DespCambio	string		`json:"DespCambio"`
}

/*
 * El metodo Init se llama cuando el Smart Contract se instancia por la red blockchain
 */

func (s *SmartContract) Init(stub shim.ChaincodeStubInterface) pb.Response {
	return shim.Success(nil)
}

/*
 * El metodo Invoke se llama como resultado de ejecutar el Smart Contract
 */

func (s *SmartContract) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
	function, args := stub.GetFunctionAndParameters()

	if function == "queryLampara" {
	 return s.queryLampara(stub, args)
	} else if function == "createLampara" {
	 return s.createLampara(stub, args)
	} else if function == "cambiarEstadoLamp" {
	 return s.cambiarEstadoLamp(stub, args)
	}

	 return shim.Error("Nombre de funcion del SmartContract invalido o inexistente.")
}
```

ANEXO IV Configuración Docker

Documento digital: docker-compose.yaml
Documento muy amplio, se incluye una parte del código.

```
version: '3'
services:
  ca:
    image: hyperledger/fabric-ca:x86_64-1.0.5
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca.org1.example.com
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=redlan
    ports:
      - 7054:7054
    command: sh -c 'fabric-ca-server start --ca.certfile /etc/hyperledger/fabric-ca-server-config/ca.org1.example.com-cert.pem --ca.keyfile /etc/hyperledger/fabric-ca-server-config/ff6e14a70a8998d4734f5c16b1b6a133db82ce6ad5687efe7d0b0167cf9d485a_sk -b admin:adminpw -d'
    volumes:
      - /home/ubuntu/tesis/crypto-config/peerOrganizations/org1.example.com/ca/:/etc/hyperledger/fabric-ca-server-config
    hostname: ca.org1.example.com
    networks:
      redlan:
        aliases:
          - ca.org1.example.com
    extra_hosts:
      - "couchdb1:190.154.218.252"
      - "peer1.org1.example.com:190.154.218.252"
      - "couchdb0:3.19.110.249"
      - "peer0.org1.example.com:3.19.110.249"
    deploy:
      placement:
        constraints:
          - node.hostname == ip-172-31-22-9
```

ANEXO V Configuración para la conexión con Hyperledger Fabric.

Documento digital: ConnectionProfile.yml

```
name: "Network"
version: "1.0"
client:
  organization: Org1MSP
  credentialStore:
    path: "./hfc-key-store"
    cryptoStore:
      path: "./hfc-key-store"
  connection:
    options:
      grpc.keepalive_time_ms: 120000
channels:
  mychannel:
    orderers:
      - orderer.example.com
    peers:
      peer0.org1.example.com:
        endorsingPeer: true
        chaincodeQuery: true
        ledgerQuery: true
        eventSource: true
      peer1.org1.example.com:
        endorsingPeer: false
        chaincodeQuery: false
        ledgerQuery: false
        eventSource: false
organizations:
  Org1MSP:
    mspid: Org1MSP
    peers:
      - peer0.org1.example.com
      - peer1.org1.example.com
    certificateAuthorities:
      - ca.org1.example.com
    adminPrivateKey:
      path: /home/ubuntu/tesis/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/47558becb298f16a0002d09451891e5326467745e1b5ce7e357c537f4878487f_sk
    signedCert:
      path: /home/ubuntu/tesis/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/signcerts/Admin@org1.example.com-cert.pem
orderers:
  orderer.example.com:
    url: grpc://3.12.17.129:7050
    grpcOptions:
      grpc-max-send-message-length: 4194304
peers:
  peer0.org1.example.com:
```

```
    url: grpc://localhost:7051
    eventUrl: grpc://localhost:7053
    grpcOptions:
      grpc.keepalive_time_ms: 600000
  peer1.org1.example.com:
    url: grpc://190.154.218.252:8051
    eventUrl: grpc://190.154.218.252:8053
    grpcOptions:
      grpc.keepalive_time_ms: 600000
certificateAuthorities:
  ca.org1.example.com:
    url: http://3.12.17.129:7054
    httpOptions:
      verify: false
    registrar:
      - enrollId: admin
        enrollSecret: adminpw
    caName: ca.org1.example.com
```

ANEXO VI Codigo NodeJS de servidor REST

Documento digital: ***server.js***

```
//==========MODULOS==============
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var rest = require('request');
var RedFabric = require('./redFabric.js');

//==========VARIABLES==============
app.set('port', 8080);
app.set('rest',rest);
const redFabric = new RedFabric('admin');

//==========INICIACION=============
app.use(function(req, res, next) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Credentials", "true");
    res.header("Access-Control-Allow-Methods", "GET, PUT");
    res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type,
Accept, Token");
    next();
});
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
app.use(express.static('public'));

//==========RUTAS================
require("./routes.js")(app, redFabric);

//===========RUN==============
// Lanza el servidor
app.listen(app.get('port'), function() {
    console.log("Servidor activo en el puerto: 8080");
});
```

ANEXO VII Codigo NodeJS de modulo RedFabric

Documento digital: ***redFabric.js***

```
'use strict';
var fabricClient = require('./config/FabricClient');
class RedFabric {
  constructor(userName) {
    this.currentUser;
    this.issuer;
    this.userName = userName;
    this.connection = fabricClient;
  }
  init() {
    var isAdmin = false;
    if (this.userName == "admin") {
      isAdmin = true;
    }
    return this.connection.initCredentialStores().then(() => {
      return this.connection.getUserContext(this.userName, true)
    }).then((user) => {
      this.issuer = user;
      if (isAdmin) {
        return user;}
      return this.ping();
    }).then((user) => {
      this.currentUser = user;
      return user;
    })
  }
  queryLamp(id) {
    var tx_id = this.connection.newTransactionID();
    var requestData = {
      chaincodeId: 'redlan',
      fcn: 'queryLampara',
      args: [id],
      txId: tx_id};
    return this.connection.query(requestData)
  }
  cambiarEstadoLamp(lamparas) {
    var tx_id = this.connection.newTransactionID();
    var requestData = {
      chaincodeId: 'redlan',
      fcn: 'cambiarEstadoLamp',
      args: [lamparas.gid,lamparas.estado,lamparas.descripcion],
      txId: tx_id };
    return this.connection.submitTransaction(requestData) }
}
module.exports = RedFabric;
```

ANEXO VIII Código NodeJs del modelo para la conexión Cliente con Hyperledger Fabric

Documento digital: ***FabricClient.js***

```
var FabricClient = require('fabric-client');
var fs = require('fs');
var path = require('path');
var configFilePath = path.join(__dirname, './ConnectionProfile.yml');
const CONFIG = fs.readFileSync(configFilePath, 'utf8')

class FBClient extends FabricClient {
   constructor(props) {
      super(props);  }

   submitTransaction(requestData) {
      var _this = this;
      var channel = this.getChannel();
      var peers = this.getPeersForOrg();
                  //var selecpeer = Math.floor(Math.random() * peers.length);
                  //var event_hub = channel.newChannelEventHub(peers[selecpeer].getName());
                  var event_hub = channel.newChannelEventHub(peers[0].getName());
      return channel.sendTransactionProposal(requestData).then(function (results) {
         var proposalResponses = results[0];
         var proposal = results[1];
         let isProposalGood = false;
         if (proposalResponses && proposalResponses[0].response &&
            proposalResponses[0].response.status === 200) {
            isProposalGood = true;
            //console.log('Transaction proposal was good');
         } else {
            throw new Error(results[0][0].details);
            console.error('Transaction proposal was bad');
         }
         if (isProposalGood) {
            //console.log(
               'Successfully sent Proposal and received ProposalResponse: Status - %s, message - "%s"',
               proposalResponses[0].response.status, proposalResponses[0].response.message);
            var request = {
               proposalResponses: proposalResponses,
               proposal: proposal
            };
            var transaction_id_string = requestData.txId.getTransactionID();
            var promises = [];
            var sendPromise = channel.sendTransaction(request);
            promises.push(sendPromise);
            let txPromise = new Promise((resolve, reject) => {
               let handle = setTimeout(() => {
                  event_hub.unregisterTxEvent(transaction_id_string);
                  resolve({ event_status: 'TIMEOUT' });
               }, 2000);
```

```
                event_hub.registerTxEvent(transaction_id_string, (tx, code) => {
                    clearTimeout(handle);
                    event_hub.unregisterTxEvent(transaction_id_string);
                    event_hub.disconnect();
                    var return_status = { event_status: code, tx_id: transaction_id_string };
                    if (code !== 'VALID') {
                        //console.error('The transaction was invalid, code = ' + code);
                        resolve(return_status);
                    } else {
                        //console.log('The transaction has been committed on peer ' +
event_hub._ep._endpoint.addr);
                        resolve(return_status);
                    }
                }, (err) => {
                    //console.log(err)
                    reject(new Error('There was a problem with the eventhub ::' + err));
                });
            });
            promises.push(txPromise);
            return Promise.all(promises);
        } else {
            //console.error('Failed to send Proposal or receive valid response. Response null
or status is not 200. exiting...');
            throw new Error('Failed to send Proposal or receive valid response. Response
null or status is not 200. exiting...');
        }
    }).then((results) => {
        //console.log('Send transaction promise and event listener promise have
completed');
        if (results && results[0] && results[0].status === 'SUCCESS') {
            //console.log('Successfully sent transaction to the orderer.');
        } else {
            console.error('Failed to order the transaction. Error code: ' + response.status);
        }
    }).then(function () {
        return 'SUCCESS';
    })
}
query(requestData) {
    var channel = this.getChannel();
    return channel.queryByChaincode(requestData).then((response_payloads) => {
        var resultData = JSON.parse(response_payloads.toString());
        return resultData;
    }).then(function(resultData) {
        if (resultData.constructor === Array) {
            resultData = resultData.map(function (item, index) {
                if (item.data) {
                    return item.data
                } else {
                    return item;
                }
            })
        }
        return resultData;
```

```
        });
    }
}

var fabricClient = new FBClient();
fabricClient.loadFromConfig(configFilePath);
module.exports = fabricClient;
```

ANEXO IX Codigo NodeJS de modulo rutas.

Documento digital: ***routes.js***

```
module.exports = function (app, redFabric) {

    /**
     * GET lampara : id
     */
    app.get("/api/lampara/:id", function (req, res) {
      var id = req.params.id;
      redFabric.init().then(function() {
        return redFabric.queryLamp(id)
      }).then(function (data) {
        res.status(200).json(data)
      }).catch(function(err) {
        res.status(500).json({error: err.toString()})
      })
           });

    /**
     * PUT -> update lampara
     */
    app.put("/api/lampara/:id", function (req, res) {
      var id = req.params.id;
      var lamparas = {
        gid : id,
        estado : req.body.estado,
        descripcion : req.body.descripcion
      };
      redFabric.init().then(function() {
        return redFabric.cambiarEstadoLamp(lamparas)
      }).then(function (data) {
        res.status(200).json(data)
      }).catch(function(err) {
        res.status(500).json({error: err.toString()})
      })
    });
};
```

ANEXO X Script inicialización Blockchain

Documento digital: ***script.sh***
Documento muy amplio, se incluye una parte del código.

```
#!/bin/bash

CHANNEL_NAME="$1"
: ${CHANNEL_NAME:="mychannel"}
: ${TIMEOUT:="120"}
CHAINCODE="redlan"
COUNTER=1
MAX_RETRY=5
ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
echo "Channel name : "$CHANNEL_NAME

setGlobals () {
        if [ $1 -eq 0 -o $1 -eq 1 ] ; then
                CORE_PEER_LOCALMSPID="Org1MSP"

        CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

        CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

        CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt

        CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
                if [ $1 -eq 0 ]; then
                        CORE_PEER_ADDRESS=peer0.org1.example.com:7051
                else
                        CORE_PEER_ADDRESS=peer1.org1.example.com:8051
                fi
        fi

        env |grep CORE
}
```