UNIVERSIDAD DE LA REPÚBLICA - FACULTAD DE INGENIERÍA

Instituto de Computación - InCo

PROYECTO DE GRADO

Comparación de plataformas para smart contracts basadas en blockchain

Autores:

Gladys Cardozo

Pablo Perdomo

Tutores:

Alberto Pardo

Germán Ferrari

Marcos viera

Montevideo, Uruguay - Julio 2020

Agradecimientos

Gracias a nuestros tutores: Germán, Marcos y Alberto por dedicarnos su tiempo, guiarnos durante el proceso y mantenernos enfocados durante la duración del proyecto.

Con mucho cariño queremos agradecer a nuestras familias, parejas y amigos por su compañía, paciencia y apoyo incondicional durante todos los años de carrera. ¡Estamos muy orgullosos de compartir este logro con ustedes!

Resumen

No hay duda de que *Blockchain* es un tema muy relevante en la actualidad y que muchas organizaciones buscan resolver algunos de sus problemas utilizando esta tecnología.

Luego del gran impulso que ha tenido Bitcoin desde 2009, el siguiente gran impacto se vivió en 2015 cuando surgió Ethereum con la implementación de los *smart contracts*, generando el desarrollo de miles de aplicaciones distribuidas que funcionan usando criptomonedas y creando un nuevo mercado de desarrollo e investigación sobre el tema.

En este trabajo se realiza un estudio de plataformas que soportan *smart contracts*, comenzando por un análisis y selección de las características más relevantes que fueron tenidas en cuenta para realizar una comparación que muestre el estado actual de la tecnología. Para poder lograr los objetivos establecidos, se estudian las veinte plataformas mejor rankeadas en uno de los sitios más importantes del mercado de criptomonedas, se estudian plataformas que tienen desarrollo en la región y otras plataformas que no son de acceso público.

Se muestran algunas conclusiones obtenidas durante la comparación así como también posibles áreas de investigación que son interesantes a seguir desarrollando. Se concluye que la elección de la plataforma está muy relacionada a la solución que se quiere implementar, no existiendo una plataforma única que abarque y resuelva todas las situaciones posibles, si no que existen plataformas recomendadas para resolver algunos casos específicos. También se concluye que si bien algunas de estas plataformas ya tienen varios años en el mercado, se siguen realizando investigaciones, probando y analizando sus características con el fin de crear una plataforma que sea más aceptada por la sociedad y el sector empresarial.

Palabras clave: Blockchain, Plataforma Blockchain, Smart contract, comparación, Ethereum, Ethereum Classic, EOS, TRON, Stellar, NEO, Waves, RSK, Quorum, Hyperledger Fabric, Corda.

1. Introducción

Bitcoin y *Blockchain* surgen como respuesta al problema de la no existencia de un instrumento que permitiese realizar pagos a través de un canal de comunicación sin la participación de un tercero de confianza (entidad financiera, bancos). Para poder realizar pagos electrónicos de forma segura, esta tecnología utiliza una base de datos distribuida, redes P2P, mecanismos de consenso y pruebas criptográficas, permitiendo que dos partes interesadas realicen transacciones directamente entre ellas.

Blockchain [NBF+16] es una estructura de bloques unidos mediante punteros de *hash,* donde cada bloque almacena información (*data*), un puntero al bloque anterior como se muestra en la Figura 1 y un valor (*digest*) que permite verificar que la estructura no fue alterada.

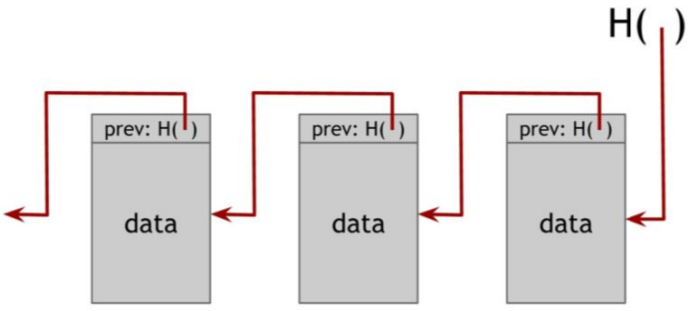

Figura 1: *Blockchain* [NBF+16]

Bitcoin [Nak08] es la primera plataforma que permite transferir monedas digitales, bitcoins (BTC). Utiliza una *Blockchain* pública para registrar el historial completo de las transacciones de la moneda. Los nodos participantes de la red de Bitcoin usan un algoritmo de consenso basado en pruebas criptográficas *Proof-of-Work* para establecer cómo agregar un nuevo bloque de transacciones a la *Blockchain.* Los nodos compiten para generar el siguiente bloque de la cadena. El primer nodo que resuelve dicha prueba gana una recompensa en BTC.

Desde que surge esta forma alternativa de comercializar fuera del sistema bancario, que asegura transacciones seguras y anónimas, el mundo a puesto mucho interés sobre este tema. Ya sea por el pico de precio que ha alcanzado Bitcoin o porque se quieren encontrar formas alternativas de guardar la información de modo inalterable, *Blockchain* está cada vez más presente en nuestro día a día. Según *coinmarketcap* [Cmc19], el sitio con mayor cantidad de consultas sobre criptomonedas, desde 2009 se han creado al menos 2253 criptomonedas y más de 40 proyectos que implementan *smart contracts.*

Actualmente se considera que se está transitando la tercera etapa en el desarrollo de esta tecnología basada en *Blockchain* [AM18]. Dentro de la primera, las plataformas se focalizaron fuertemente en lograr un sistema de intercambio seguro de monedas electrónicas, en la creación de las mismas y en su forma de almacenamiento. Se las denomina como "*Blockchain* de primera generación" y algunos ejemplos son; Bitcoin, Litecoin y Dash. Durante la segunda etapa, surge una nueva generación de plataformas con el objetivo de potenciar todas las características de la primera e implementar lo que definió Nick Szabo en 1994 [Sza94], la creación de *smart contracts*, que entre otras propiedades, permite configurar transacciones de forma programable. Se la denomina como "*Blockchain* de segunda generación" y algunos ejemplos muy conocidos son: Ethereum Classic, Ethereum, NEO y QTUM. Actualmente estamos transitando la etapa tres y es denominada "*Blockchain* de tercera generación". Buscan seguir mejorando la tecnología desarrollada en las etapas anteriores y ser altamente escalables, logrando reducir el tiempo de procesamiento de las transacciones y reorganizar los nodos para lograr la auto-gobernación. Algunas plataformas son: Bitshares, Steem, Lisk, EOS.

Este trabajo se centra en la comparación de plataformas de segunda y tercera generación, no solo comparando los lenguajes que las diferencian, si no, evaluando aspectos más amplios brindando al lector una serie de criterios para la comparación de las plataformas.

Los objetivos que se establecen para alcanzar la meta mencionada son:

- Estudiar las principales características de la tecnología *Blockchain*.
- Identificar y catalogar diferentes plataformas de *Blockchain* de 2ª y 3ª generación.
- Elaborar criterios para la comparación de las plataformas.
- Aplicar los criterios de comparación a un conjunto de plataformas seleccionadas.

A continuación se describe cómo es organizado este documento.

- Capítulo 2: Marco teórico. Se definen los conceptos teóricos más importantes utilizados a lo largo del documento.
- Capítulo 3: Elaboración de los criterios de comparación. Se definen las características que se utilizarán en la comparación de las plataformas.
- Capítulo 4: Análisis de plataformas. Se describen y analizan las plataformas seleccionadas. Se comparan las características de cada una en base a los criterios definidos.
- Capítulo 5: Conclusiones y trabajo futuro. Se describen las conclusiones obtenidas y se proponen líneas de investigación en las cuales se puede seguir profundizando.

2. Marco teórico

Para comprender correctamente la temática planteada en este documento, se presentan algunos conceptos fundamentales, sus características y particularidades.

Uno de los aspectos centrales es *Blockchain*, esta tecnología no surge de forma aislada si no que utiliza y re-implementa otras tecnologías concebidas previamente: pruebas criptográficas, redes P2P, bases de datos distribuidas y mecanismos de consenso, entre otros.

En la literatura relacionada no existe una definición consensuada que describa que es *Blockchain* específicamente ya que cuando Satoshi[Nak08] describió la solución, no acuñó el término. Por lo tanto, abunda el uso de la palabra, pero existen sutilezas y diferencias de acuerdo al contexto en el que se presenta. En base a la información recopilada y a definiciones presentadas en el libro “Bitcoin and Cryptocurrency Technologies” de la Universidad de Princeton [NBF+16], se describen las definiciones de *Blockchain* y plataforma de *Blockchain*.

Blockchain

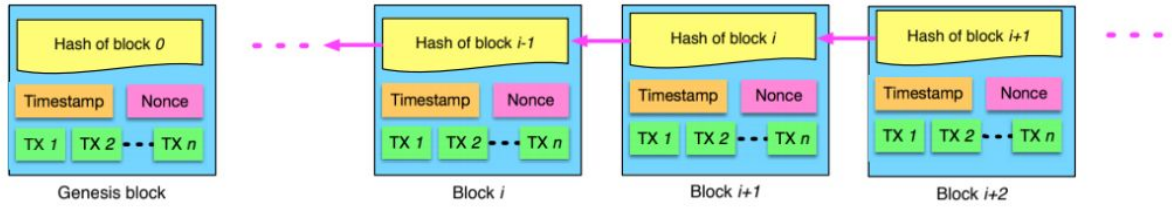

Figura 2: *Estructura Blockchain* [ZXD+18]

Blockchain es una secuencia de bloques que contiene un registro completo de todas las transacciones, a este registro se le conoce como *ledger*. En la Figura 2 se ilustra un ejemplo sobre la estructura de *Blockchain*. El primer bloque, se llama *Genesis Block*, y es el único que no tiene una referencia a un bloque anterior. Todos los bloques subsiguientes tiene un valor de referencia *hash code* que se calcula utilizando datos del propio bloque y del *hash code* del bloque anterior, de esta forma los mismos quedan conectados. Si alguien quisiera alterar la información de un

bloque *i*, debería modificar los hashes de todos los bloques *i+1* para que la cadena sea consistente, a esta acción se le llama comúnmente como *fork* de la *Blockchain*.

Función de Hash

Es una función matemática que transforma cualquier entrada de datos en una nueva serie de datos con una longitud fija como se muestra en la Figura 3. Es decir, independientemente de la longitud de los datos de entrada, la longitud del valor de salida será la misma. Este valor es conocido como *hash code*. Se llaman funciones de hash criptográficas a aquellas funciones de hash que cumplen las siguientes dos propiedades: resistencia a colisiones y ocultamiento. Una colisión ocurre cuando dos datos de entrada distintos producen el mismo hashcode. Una función se dice resistente a colisiones si nadie puede encontrar una colisión, formalmente se define una función H resistente a colisiones si es inviable encontrar dos valores *x* e *y* tales que $x \neq y$ y $H(x)=H(y)$. En cuanto a la segunda propiedad, ocultamiento, formalmente una función de hash H es de ocultamiento si, eligiendo un valor secreto r desde una distribución de probabilidad con alta min-entropía, entonces $H(r \parallel x)$ hace que no sea factible encontrar x. Min-entropía es una medida de cuán predecible es una salida y alta min-entropía captura la idea intuitiva de la que distribución de probabilidad es esparsa. Entonces, cuando elegimos r en esta distribución, no hay mayor probabilidad de que un valor específico ocurra, por lo que hace tener una probabilidad muy baja de encontrar x, $y=H(x)$ [NBF+16].

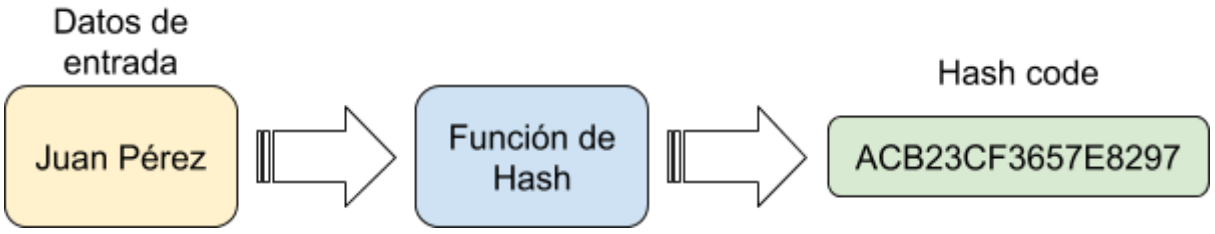

Figura 3: *Ejemplo de función de hash*

Plataforma de *Blockchain*

Una plataforma de *Blockchain* es una red de nodos *peer-to-peer* (P2P) que se encarga de mantener una *Blockchain* distribuida, o *ledger* distribuído, ejecutar un mecanismo de consenso que permita incorporar nuevos bloques al *ledger* en forma segura, ejecutar *smart contracts* si es que lo soporta y pagar los incentivos si es que utiliza. Estos nodos son los participantes de la plataforma y pueden ser individuos u organizaciones. Es común que las plataformas inicien con una *Blockchain* de pruebas y luego liberen una versión en producción de la solución. Incluso algunas además mantienen una versión de desarrollo para que los programadores validen la ejecución de *smart contracts* antes de ser publicados en la red.

Si bien una plataforma de *Blockchain* es una red P2P, el acceso a la misma y el manejo de los bloques en el *ledger*, puede ser controlado por una o más entidades. A partir de esta idea se pueden diferenciar tres tipos de plataformas [VH19]:

Privadas: En este tipo de plataformas el *ledger* es compartido y validado por un grupo predefinido de nodos. El sistema requiere iniciación y/o validación a los nodos que quieran ser parte del sistema. Estos nodos autorizados son responsables de mantener el consenso. Una plataforma de este tipo es adecuada para sistemas cerrados, donde todos los nodos son completamente confiables y el administrador tiene la máxima autoridad para controlar el acceso a los nodos autorizados.

Públicas: Por otro lado se encuentran las que son públicas. Estas permiten, a cualquier persona, acceder y mantener el *ledger* distribuido. Para poder validar la integridad del *ledger* se realiza la ejecución de un mecanismo de consenso. Una plataforma de este tipo es completamente abierta y distribuida; cualquiera puede unirse, participar, y dejar el sistema. Este sistema funciona bajo nodos desconocidos y posiblemente no confiables.

Permisionadas o de Consorcio: Son un híbrido entre las dos plataformas mencionadas anteriormente, en donde, el *ledger* se mantiene público (o parcialmente público) y los nodos participantes en la ejecución del mecanismo de consenso son preseleccionados. Es decir, no todos los nodos de la red pueden

participar en el acuerdo y en la validación de las transacciones. Este tipo de plataforma es adecuada para sistemas que buscan ser más performantes o que buscan establecer un consorcio entre varias empresas teniendo al menos un nodo representante. A pesar de que el sistema no es completamente abierto, aún obtiene algunos beneficios de la descentralización como por ejemplo, el sistema tiene tolerancia a fallas en el caso de que algún nodo actúe maliciosamente.

Transacción

Una transacción es un intercambio de información que se almacena en el *ledger*. Dependiendo de la plataforma de *Blockchain* usada, será un intercambio de activos, criptomonedas o la ejecución de un *smart contract*. En la Figura 4 se muestra un ejemplo de una transacción de Bitcoin, extraída desde un explorador de la plataforma. Allí se puede observar el hash, que identifica la transacción, la entrada y la salida. En este caso desde dos direcciones se transfieren 0,00014566BTC a una sola cuenta. Además se puede identificar la fecha y hora de la misma, si ya fue confirmada por los mineros y cuanto es el costo de la comisión.

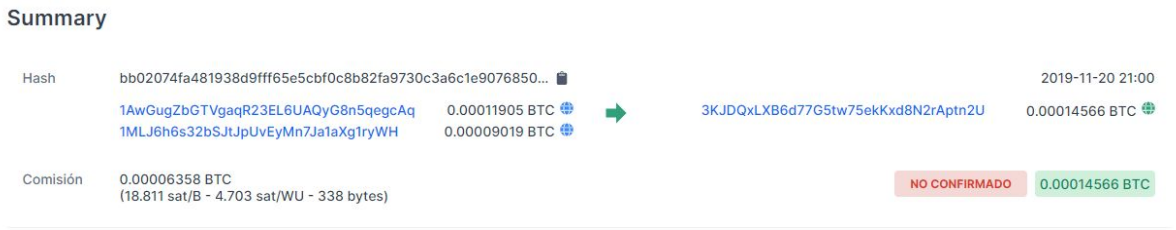

Figura 4: *Transacción en Bitcoin.* [BE19]

Red P2P

Si bien existen varias clasificaciones y definiciones de redes *peer-to-peer* (P2P), se puede decir, de forma general, que es una red de computadoras conectadas a través de internet cuyo objetivo principal es compartir recursos y trabajar de forma colaborativa para poder entregar/ejecutar una tarea específica. Como se muestra en la Figura 5, a diferencia de las redes cliente-servidor, en donde se tiene un servidor centralizado, en este tipo de redes cada nodo actúa como cliente y servidor al mismo tiempo, es decir, cada nodo puede requerir o proveer servicios.

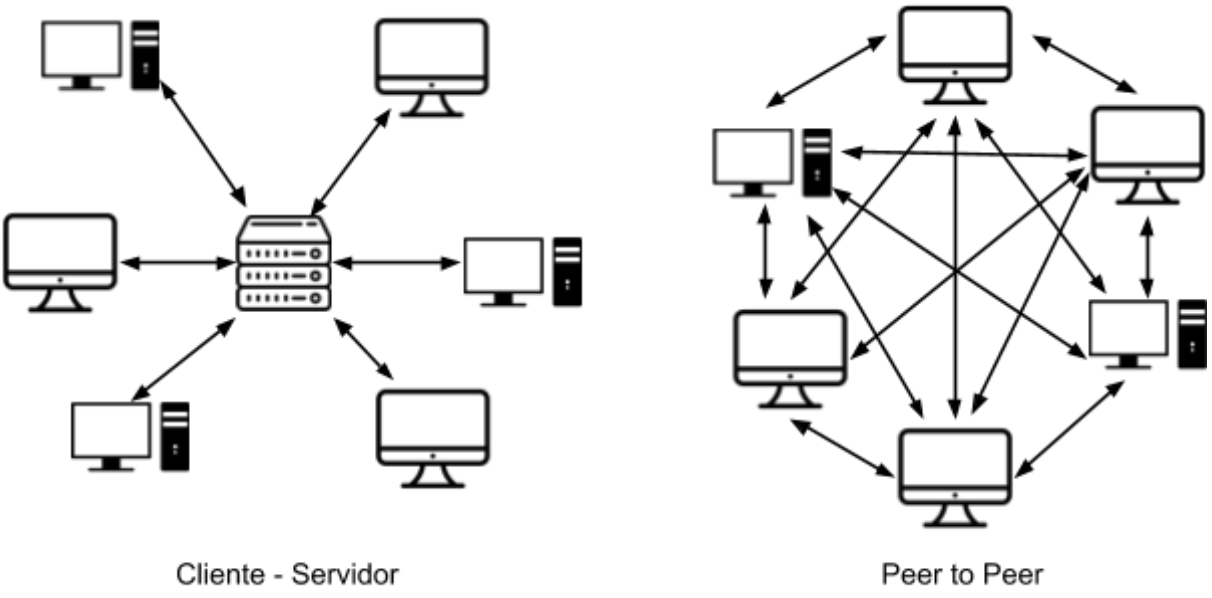

Figura 5: *Redes clásicas versus P2P*

Las redes P2P son tan populares y utilizadas debido a que presentan varias ventajas frente a otro tipo de redes, principalmente aspectos relacionados a la *performance*. Dentro de estas características se destacan las dos siguientes:

- **Escalabilidad**. Contrariamente a lo que sucede en las redes cliente-servidor, donde cuantos más usuarios existan conectados peor será su rendimiento, en las redes P2P es deseable que el número de nodos conectados sea alto. Esto permite tener más distribuida la información, tener más recursos para ejecutar y tener mayor tolerancia a fallos.
- **Robustez**. Al ser una red distribuida y en donde todos los nodos realizan las mismas tareas, es muy difícil que la red deje de funcionar. Es decir, al no existir un punto singular de falla, incluso si algunos de estos nodos se desconectan, la red puede seguir funcionando sin tener problemas de pérdida de información.

Mecanismo de Consenso

Los nodos del sistema utilizan el mecanismo de consenso para acordar el estado del *ledger* cada vez que un nuevo bloque es creado. El problema de lograr un acuerdo entre varios nodos no surge en las plataformas de *Blockchain*, si no que se ha estudiado fuertemente en sistemas distribuidos. En el caso de estas plataformas se establece qué tipo de mecanismo funciona mejor en base a las características deseadas. Por ejemplo, si es una plataforma cerrada, en la que se confía plenamente en los nodos participantes, se pueden utilizar mecanismos que no requieran validaciones adicionales de seguridad. Entonces, tenemos un sistema distribuido donde un conjunto de nodos se comunican entre sí mediante mensajes para ponerse de acuerdo acerca del estado de la plataforma, previendo que pueden existir fallas en los canales de comunicación o comportamiento malicioso por parte de otros nodos (conspirar para que el consenso no ocurra).

En general un mecanismo de consenso debe cumplir las siguientes dos propiedades: el valor propuesto tiene que ser generado por un nodo honesto y el mecanismo debe finalizar en el acuerdo de un valor entre todos los nodos honestos [NBF+16].

El objetivo del mecanismo de consenso, además de acordar el estado actual de la plataforma, nos permite elegir el nodo que propondrá el próximo bloque en forma aleatoria y anónima de una forma que no se exponga al sistema a un ataque *Sybil* [NBF+16]. *Sybils* son copias de nodos que un adversario malicioso puede crear para que parezca que hay muchos participantes diferentes. Este tipo de ataques surgen en cualquier red peer-to-peer, no es específico de las plataformas de blockchain. Debido a que cualquier nodo podría ser elegido, la red se podría llenar de nodos que se ponen de acuerdo para actuar de forma maliciosa e inundar la plataforma de información incorrecta. La forma en la que las plataformas públicas moderan este comportamiento es a través de las incentivos económicos. Con esto, se incentiva a que los nodos mantengan un buen comportamiento sobre la plataforma.

Resumiendo, los principales objetivos del mecanismo de consenso son: primero alcanzar un acuerdo unificado, que no es trivial, debido a que no es un sistema centralizado, sino, que el proceso se basa en la confianza de los participantes; segundo, prevenir el *double-spending,* este es un problema que existe previo a la concepción de *Blockchain* y trata de prevenir la réplica de información, en el caso de las criptomonedas, es evitar que una moneda sea gastada dos veces; tercero, en caso que la plataforma lo provea, alinear con los incentivos económicos, es decir, recompensar a los nodos que validan de forma exitosa y mantienen la plataforma segura evitando así que se generen ataques Sybil.

Algunos mecanismos se diferencian en que cumplen la propiedad de *Byzantine Fault Tolerance,* esta propiedad se basa en el problema de los generales Bizantinos [LSP82], es un problema que busca resolver como los generales de una tropa se ponen de acuerdo en atacar o retirarse, asumiendo que pueden existir generales corruptos o que el mensaje puede ser extraviado en el camino.

Dentro del contexto de *Blockchain*, se busca alcanzar un acuerdo del estado general de la red, asumiendo que existen errores en la misma y que pueden existir nodos que actúan de forma maliciosa. Este tipo de mecanismo, en general organiza a los nodos de forma tal que uno es el principal y los otros funcionan como respaldo.

RAFT es un mecanismo útil para redes cerradas o de consorcio en donde no es necesario implementar un mecanismo a prueba de fallas bizantinas, por lo que resulta más simple. Es del tipo *Crash Fault Tolerance* por lo que delega la confianza a un nodo líder (se asume siempre que es honesto). Si el líder falla por un error de la red, se elegirá de forma automática un nuevo líder. Los bloques son generados únicamente si hay transacciones disponibles. Esto reduce el espacio de almacenamiento de la *Blockchain*. Los bloques son difundidos a toda la red antes de ser agregados a la cadena y no son protegidos por un hash único o por firmas de nodos, por lo que lo hace altamente inseguro el momento de reescribir los bloques de la cadena. Se distinguen tres tipos de nodos; nodo líder (único en toda la red), nodos candidatos y nodos seguidores, que se definen luego de un proceso de votación. Es decir, en primera instancia todos los nodos son de tipo “candidato”,

luego se vota y elige a un líder y el resto de los nodos pasan a ser del tipo "seguidor". El líder es el nodo encargado de crear bloques, agregar las transacciones y compartirlo a la red. En el caso de los seguidores, son nodos que no son capaces de minar bloques nuevos, simplemente pertenecen a la red y ayudan a mantener la *Blockchain*. Minar en este contexto es validar y registrar las transacciones en la cadena de bloques**.**

pBFT *(practical Byzantine Fault Tolerance)* fue diseñado para trabajar de forma eficiente de manera asíncrona. Los nodos participantes de la red son ordenados de forma secuencial, uno de los nodos es denominado como líder de forma aleatoria y el resto como nodos secundarios. El objetivo es que todos los nodos honestos logren el consenso por la regla de la mayoría. Este tipo de mecanismo funciona si el número de nodos maliciosos no supera el tercio de los nodos participantes. El funcionamiento es el siguiente: el cliente envía una solicitud al líder. El líder reenvía la solicitud a los nodos secundarios. Todos los nodos revisan la solicitud y responden al cliente. La solicitud es validada cuando el cliente recibe m+1 respuestas iguales de los nodos, siendo m el máximo número posible de nodos maliciosos.

dBFT *(delegated Byzantine Fault Tolerance)* clasifica a los nodos entre delegados y oradores. El lugar de un general o un líder se tienen diferentes delegados que son elegidos por los nodos de la red y dentro de estos delegados se elige un orador al azar en cada iteración. El orador es responsable de la creación del nuevo bloque a partir de las transacciones pendientes de validar. El bloque propuesto se envía a los delegados para su validación. Los delegados pueden refutar la honestidad del orador en base a la consistencia de la información recibida. Si más de dos tercios de los delegados alcanzan el consenso y lo validan, el bloque es añadido a la cadena. Para afectar el mecanismo de consenso, los delegados maliciosos necesitan controlar al menos dos tercios de la red para corromper los datos de la *Blockchain*.

IBFT *(Istanbul Byzantine Fault Tolerance)* es una implementación de *Practical Byzantine Fault Tolerance* y la diferencia más grande con RAFT o cualquier otro algoritmo del tipo *Crash Fault Tolerance* es que no se confía en lo que indica el

nodo líder, siempre se asume que puede ser un nodo malicioso. Cada bloque requiere de múltiples rondas de votación de un conjunto de nodos validadores para alcanzar el consenso y esto se registra como una colección de firmas dentro del bloque. Utiliza tres fases; *PRE-PREPARE, PREPARE y COMMIT*, y permite tener una tolerancia de hasta un tercio de los nodos validadores defectuosos. Los nodos que son elegidos como validadores deben elegir a un nodo, por ronda, para que proponga un nuevo bloque. Luego de que este nodo propone un nuevo bloque, debe enviarlo a toda la red junto con un mensaje *PRE-PREPARE*. Cuando los nodos validadores reciben el bloque con el mensaje *PRE-PREPARE*, crean el estado *PRE-PREPARED* y lo vuelven a reenviar al resto de los nodos para asegurar que todos están trabajando en el mismo bloque. Después de, al menos dos tercios de los nodos, recibir el mensaje *PRE-PREPARED*, validan el bloque y vuelven a generar un mensaje para reenviarlo a la red, en este caso COMMIT. Finalmente al recibir el estado *COMMIT* por al menos dos tercios de los nodos, los nodos actualizan el estado y agregan el nuevo bloque a la cadena.

Algo importante a destacar de este mecanismo de consenso es que no genera *forks* en la cadena.

Por otro lado, existen algoritmos basados en pruebas, aquí los nodos deben resolver un puzzle criptográfica de forma exitosa para poder agregar nuevos bloques a la cadena y así obtener un incentivo económico (medido en la criptomoneda que sea utilizada). Uno de estos mecanismos es *POW (Proof of Work)* y lo utiliza Bitcoin.

POW de forma simplificada, permite seleccionar un nodo de forma aleatoria, siempre y cuando sea capaz de resolver el puzzle criptográfico. Su funcionamiento está dado por los siguientes pasos: primero se anuncian las nuevas transacciones a todos los nodos, cada nodo elabora un bloque con las nuevas transacciones recibidas, en cada ronda, un nodo aleatorio anuncia su bloque, los otros nodos pueden aceptar el bloque si deciden que sus transacciones son válidas, los nodos aceptan el bloque mediante la inclusión de su hash en la creación del siguiente bloque. Cuantas más confirmaciones se obtienen, mayor será la probabilidad de que

el bloque pertenezca a la *Blockchain*. En este contexto, confirmaciones implica que los nodos han procesado y agregado el bloque a su cadena. Este mecanismo requiere de un gran poder computacional para la resolución del puzzle, lo que resulta en un gran costo de recursos para el nodo que vaya a crear un bloque. Este es uno de los motivos por el cual se agregan incentivos a la creación de los bloques, ya que permite a los nodos participantes recuperar parte de su inversión y a su vez incentivar un comportamiento honesto.

En el caso de ***POS*** (*Proof of Stake*) los nodos pueden minar o validar bloques en función de la cantidad de criptomonedas que posean. Es decir que cuanto mayor sea su riqueza, mayor será la probabilidad de que participe en la generación de un nuevo bloque. Aunque el proceso se mantenga aleatorio, se tiene que tener un mínimo de criptomonedas para participar de este grupo de nodos. Una de las ventajas frente a *POW* es que no necesita capacidad de cómputo para lograr el consenso, por lo que es más amigable con los recursos.

DPOS (*Delegated Proof of Stake*) es una variación del mecanismo mencionado anteriormente. En lugar de nodos o validadores, a los nodos se los conoce como delegados y cualquier nodo puede convertirse en tal mediante la votación de los nodos participantes en la red. El poder de votación de cada nodo depende de su participación dentro de la red. Cada delegado tiene un turno para producir un nuevo bloque y cobrar el incentivo. Este mecanismo es más escalable que los anteriores.

LPOS (*Leased Proof of Stake*) busca resolver el problema de centralización que presenta *POS* al permitir que sólo los nodos pudientes puedan participar del consenso. *LPOS* permite que los nodos arrienden sus monedas a la red y tomen beneficio de ello. Las monedas alquiladas están bloqueadas en la cuenta del usuario y no pueden ser transferidas o intercambiadas. Permanecen en control total del titular de la cuenta y los arrendamientos se pueden cancelar en cualquier momento. Las monedas que se alquilan a un nodo que va a minar se utilizan para aumentar su posición y aumentar las posibilidades de que encuentre el siguiente bloque.

SCP (*Stellar Consensus Protocol*) busca alcanzar el consenso entre subgrupos de nodos participantes llamados quorums. En conjunto los quórums con los nodos globales, alcanzan un consenso entre los conjuntos de transacciones, que luego son confirmadas. Los nodos globales son instituciones financieras y otras fuentes confiables, mientras que cualquier nodo participante puede interactuar dentro del quorum. En cada quórum, participa el nodo y sus nodos de confianza. Mediante el voto federado, el quórum se pone de acuerdo con una declaración. El voto federado funciona de la siguiente forma, en una ronda de votación, cada nodo debe seleccionar uno de los posibles valores. No puede hacer esto hasta que esté seguro de que los otros nodos del quórum elegirán el mismo resultado. Para estar seguros, los nodos intercambian una ráfaga de mensajes entre ellos para que todos confirmen que el quórum de nodos toma la misma decisión. La votación federada resuelve el problema de que la red se ponga de acuerdo, ya que al menos un nodo del quórum pertenece a otro quórums, donde se asume fuertemente que esta superposición logra la cobertura de toda la red. El proceso de votación, aceptación y confirmación es una ronda completa de votación federada. Este protocolo combina muchas de estas rondas para alcanzar el consenso.

Fork

Un *fork* (bifurcación) es una división de la *Blockchain* en dos ramas. Esto ocurre dentro del funcionamiento habitual en las redes que su mecanismo de consenso lo permita, cuando más de un nodo propone el siguiente bloque a introducir en la cadena o cuando se detectan errores a nivel estructural, defectos o existe una necesidad de mejora en la plataforma. En el primer caso, los nodos continúan trabajando sobre la cadena que el mecanismo de consenso considere válida. Podemos distinguir dos tipos de cambios o mejoras, los que producirán un *hard fork* y los que producirán un *soft fork*. Dentro de los *hard fork*, se introducen cambios que previamente se consideraban inválidos, es decir que la nueva versión de software reconocerá a bloques como válidos, que previamente rechazaba. Cuando la mayoría de los nodos están actualizados, la cadena más larga contendrá bloques que son inválidos para los nodos desactualizados. Entonces ellos trabajarán sobre

una rama que excluya a los bloques con la nueva funcionalidad. Hasta que ellos se actualicen, considerarán a su rama (la más corta) como la más larga válida. Este suceso se llama hard fork porque hace que la cadena se divida. Los nodos de la red, continuarán trabajando en una u otra cadena dependiendo de la versión de software que tengan y estas ramas nunca se unirán otra vez. Un segundo tipo de cambio que podemos hacer en Bitcoin es agregar características que hacen que las reglas de validación sean más estrictas. Es decir, restringen el conjunto de transacciones válidas o el conjunto de bloques válidos de manera tal que la versión anterior aceptaría todos los bloques, mientras que la nueva versión rechazaría algunos. Este tipo de cambio se denomina *soft fork* y puede evitar la división permanente que introduce *hard fork*. Estas bifurcaciones, si no son bien recibidas por la comunidad, pueden afectar la confianza en la plataforma y el interés de los inversores.

En agosto de 2017, la comunidad de Bitcoin estaba discutiendo cómo mejorar el rendimiento de las transacciones (escalabilidad de la plataforma). Existían dos opiniones diferentes: aumentar el tamaño del bloque a 8 MB y almacenar los datos dentro de un bloque con un método llamado SegWit. Estas diferencias provocaron la primera bifurcación de Bitcoin: Bitcoin Cash que optó por la primera opción y Bitcoin que ahora usa SegWit.

Otro ejemplo es en 2016, un atacante logró utilizar la vulnerabilidad de la seguridad de Ethereum para su beneficio y muchas personas fueron afectadas perdiendo sus inversiones, este ataque afectó el proyecto conocido como Decentralized Autonomous Organization (DAO) [MCA+17]. Dentro de la comunidad de Ethereum existían dos formas de resolver este problema: revertir las transacciones debido a la falla o no hacer ningún cambio, manteniendo el protocolo original. Finalmente la *Blockchain* se bifurcó en: Ethereum (con la falla corregida) y Ethereum Classic (respetando el protocolo original).

Es claro también que si hay una diferencia en la *Blockchain* pero la comunidad está 100% de acuerdo, la modificación se realiza y no se producen bifurcaciones.

Smart contracts

Los *smart contracts* fueron definidos en 1994 por Nick Szabo [Sza94]. Sin embargo, en esa época era inviable su desarrollo debido a que no se contaba con la infraestructura necesaria. Un *smart contract* (contrato inteligente) es un programa informático que generalmente está almacenado en una *Blockchain y* que es invocado/ejecutado por la parte interesada para lograr que ciertas acciones se lleven a cabo a partir de una serie de condiciones. Es decir, si el contrato es invocado y se cumplen las condiciones pre-establecidas, el *smart contract* ejecuta automáticamente las cláusulas correspondientes.

Los *smart contracts* pueden ser un acuerdo entre más de un interesado. Es decir, se pueden utilizar para pautar un intercambio de forma segura si alguna de las condiciones establecidas es verdadera. Los mismos existen dentro de un ecosistema que no es controlado por ninguna de las partes implicadas ni por un tercero, si no que son conocidos y alcanzables por todos los nodos que tengan acceso a la *Blockchain* donde el contrato está almacenado. Es decir, se programan las condiciones, las partes implicadas firman el contrato (si es requerido) y se aloja en la *Blockchain* para que no pueda ser modificado.

Los objetivos principales de los smart contracts son: agregar mayor seguridad a los contratos tradicionales, reducir costos administrativos, reducir el tiempo asociado a este tipo de interacciones, evitar tener que usar un intermediario. Para este último punto, por ejemplo, en el intercambio de un bien, no se necesitaría un escribano que garantice la transacción.

En otras palabras, buscan mejorar los contratos actuales siendo más seguros, más baratos, ahorrando tiempo y evitando fraudes. Sin embargo, las consecuencias de las malas decisiones de diseño dentro de los lenguajes de programación pueden ser costosas, como es el caso del contrato DAO [MCA+17] en Ethereum.

Debido a que los *smart contracts* son piezas de código guardadas dentro de los bloques, heredan la propiedad de inmutabilidad de la *Blockchain*. Esta propiedad históricamente ha causado problema por fallas de diferentes orígenes, tanto

humanas (errores en el código de programación) como de seguridad (comportamientos maliciosos no deseados), por lo que mediante patrones de diseño o funcionalidades habilitadas únicamente al dueño del contrato, algunas plataformas permiten ciertas flexibilidades. Un patrón que permite hacer esto posible es, alojar en un contrato la dirección de la última versión del contrato ejecutable. De esta forma, en caso de que se quiera actualizar, es posible desplegar un nuevo contrato y modificar la dirección de invocación del primero. El contrato anterior no es borrado de la *Blockchain*, pero se puede invocar una función propia para que sea deshabilitado.

Criptomoneda

Es un medio digital de intercambio que surge dentro de las plataformas de *Blockchain*. Si bien es una moneda que se basa en criptografía, su valor está sujeto a variaciones de precio dependiendo de la oferta y la demanda del mercado.

Las criptomonedas tienden a tener las mismas características que el dinero: son fungibles, divisibles, portátiles y tienen un suministro limitado. Normalmente, las criptomonedas están destinadas a ser utilizadas como el efectivo físico: para pagar por bienes y servicios (aunque la adopción minorista todavía es lenta) o como reserva de valor y ahorro.

Token

Los *tokens* son activos digitales que se pueden usar dentro del ecosistema de un proyecto determinado. Es una unidad de valor que surge a través de una ICO (*initial coin offering*) o relacionado al uso de una *Blockchain*. Los *tokens* se crean sobre el protocolo de una criptomoneda existente en una determinada *Blockchain*.

La mayor parte de los *tokens* se distribuyen a través de una ICO donde los emprendimientos captan dinero o criptomonedas para el desarrollo de su plan de negocios de parte de los inversionistas interesados (*crowdfunding*), quienes a cambio reciben *tokens* que tendrán una utilidad o valor dentro del proyecto. El propósito de los *tokens* es diferente al de las criptomonedas, aunque también se

puedan usar como medio de pago. Por ejemplo, muchos *tokens* se crean para usarse dentro de aplicaciones descentralizadas (DApps) y sus redes.

Completitud de Turing

Se dice que un lenguaje de programación es Turing completo si permite representar cualquier función computable [DSW94]. La propiedad de completitud de Turing en *smart contract*s es asociada generalmente a un conjunto de características del lenguaje, como por ejemplo soportar bucles.

Si bien un lenguaje puede ser Turing completo, dentro de la VM (*virtual machine*) de la *Blockchain* se puede restringir las iteraciones/recursiones, por ejemplo, modelando el consumo de un *token* obligando a que la iteración/recursión termine cuando este se agote, interrumpiendo la ejecución del programa. De esta forma la VM evita la ejecución de bucles arbitrarios. La completitud es inherentemente más poderosa pero también expone más vulnerabilidades.

Oráculos

Los oráculos son proveedores de datos externos, capaces de introducir información a la *Blockchain* para que sea utilizada por los *smart contracts* que la necesiten. Existen varios tipos de oráculos, entre ellos se pueden distinguir, oráculos de *hardware*, de *software*, de consenso, oráculos *inbound* (que permiten agregar información del mundo real a la cadena) y *outbound* (que permiten informar a una entidad fuera de la cadena de un evento que ocurrió dentro de ella). Un aspecto importante que deben resolver los oráculos es que la información debe estar validada y deben proveer un mecanismo que asegure, a quien consume los datos, que los mismos no fueron modificados. También es importante entender que al ser un proveedor de datos para *smart contracts* y ser una fuente centralizada, existe el interés en atacarlos debido al poder que representan. Si el oráculo está comprometido, todos los *smart contracts* que lo utilicen lo estarán.

3. Metodología de investigación

En este capítulo se describen los pasos para identificar y catalogar plataformas de *Blockchain* de 2ª y 3ª generación, aquí se presenta uno de los aportes centrales del proyecto, la selección y definición de criterios de comparación que son aplicados a un conjunto de plataformas de *smart contracts* basadas en *Blockchain*. Estos criterios no buscan determinar la mejor solución, si no que establecen un marco base para analizar las características más importantes que presenta esta tecnología y continuar con su estudio a futuro.

3.1 Estudio de la tecnología y relevamiento estado actual

Para el desarrollo de la metodología de investigación sobre plataformas de *smart contracts* se estudia en profundidad la tecnología *Blockchain*, cubriendo conceptos que son claves para entender cómo funciona, cuáles son sus características más relevantes y el estado actual de la misma. Para lograr este objetivo, uno de los recursos utilizado es el libro “Bitcoin and Cryptocurrency Technologies” de la Universidad de Princeton [NBF+16], citado también en el marco teórico de este documento.

Luego de asentar una base de conocimiento, es necesaria la determinación de las fuentes de información para evaluar el estado actual de *Blockchain* y realizar el relevamiento de las plataformas de estudio. En este documento se utilizan como fuentes de información *CoinMarketCap* [Cmc19] debido a su popularidad entre usuarios y empresas, *GitHub* bajo el tag de *#blockchain*, documentación oficial de cada una de las plataformas y eventos como *Blockchain Summit*, *Meetups* y talleres relacionados.

3.2 Selección de características de comparación

En base al estudio realizado sobre el estado actual de la tecnología es que se puede definir un conjunto de características que permiten comparar las plataformas seleccionadas. A continuación se describen las mismas.

3.2.1 Mecanismo de consenso

Se hace un relevamiento de qué mecanismo de consenso es usado por cada plataforma seleccionada.

Los mecanismos de consenso determinan atributos característicos como: la escalabilidad, el grado de descentralización, la seguridad del sistema mediante la tolerancia a fallos y por ende la consistencia del *ledger*.

Existe una relación directa entre la complejidad de ejecución del mecanismo y el volumen de transacciones soportado. Es decir, el tiempo que se tarda en llegar a un acuerdo para ser impactado en el *ledger*, con la cantidad de transacciones que se pueden procesar, y por ende la escalabilidad de sistema.

3.2.2 Completitud de Turing

Es interesante revisar si los lenguajes soportados por las plataformas cumplen o no con la propiedad de la completitud de Turing, ya que la ausencia de esta característica restringe el espacio de soluciones que se pueden implementar. Algunas plataformas eligen limitar su potencial implementando mecanismos ajenos al lenguaje en busca de un equilibrio entre seguridad y complejidad de los contratos.

La incompletitud de Turing se obtiene cuando no hay loops, recursión general y expresiones del tipo *go-to* dentro del lenguaje. La ausencia de estas estructuras, permite saber de antemano cuánto poder de procesamiento es requerido para la ejecución, es decir, se puede determinar con exactitud el costo computacional que se requiere en la plataforma.

De aquellas plataformas que soportan lenguajes con esta propiedad, es relevante estudiar cuáles son sus vulnerabilidades (por ejemplo *loops* infinitos donde agoten recursos) y qué mecanismos implementan para mitigarlos, ya que al contar con un lenguaje que permita modelar una realidad más compleja se incrementará la probabilidad de introducir errores y fallas de seguridad. Sobre las que utilizan lenguajes no completos, interesa entender qué posibilidades brindan para realizar contratos y qué puntos de seguridad están garantizando.

3.2.3 Lenguajes permitidos

También es relevante comparar cuáles son los lenguajes permitidos en cada plataforma, ya que permiten obtener indicadores sobre cuántos son los desarrolladores que los usan, el tamaño de la comunidad, la documentación que exista al respecto y las características específicas de cada lenguaje. Si bien se considera que tener una mayor cantidad de lenguajes soportados va a impactar positivamente en la incorporación de la plataforma, se entiende que es importante tener un balance entre la cantidad y cuáles son los lenguajes soportados.

3.2.4 Tamaño de la comunidad

Se mide el tamaño de la comunidad ya que indirectamente permite evaluar cuál es la utilización real que tiene una plataforma, cuán difícil es comenzar a trabajar en ella y qué tan segura puede ser. Al tener una comunidad más activa la mejora es continua, por lo que se solucionan problemas constantemente y se proponen mejoras futuras.

Para evaluar el tamaño de la comunidad de cada una de las plataformas se estudia la utilización de las mismas en dos de las fuentes más usadas por los desarrolladores; *StackOverflow* y *GitHub*. Sobre ellas se van a analizar los siguientes tres aspectos:

- La cantidad de recursos que utilizan el *tag*: “nombre plataforma”, dentro de *StackOverflow*.

- La cantidad de preguntas que contienen simultáneamente los siguientes *tags*: “nombre plataforma” y “*Blockchain*”; “nombre plataforma” y “*smart contracts*”, dentro de *StackOverflow*.
- La cantidad de repositorios en *GitHub* que se encuentran a partir de las siguientes búsquedas: “nombre plataforma” y “*Blockchain*”; “nombre plataforma” y “*coin*”; “nombre plataforma” y “*smart contract*”

3.2.5 Tipo de acceso

Se hace un relevamiento sobre el tipo de acceso que ofrecen las plataformas. Las mismas son clasificadas en: públicas, privadas o permisionadas. Las principales diferencias entre estos tipos de *Blockchain* son la forma en la que se comparte el *ledger* y el permiso para participar en el sistema.

Es importante evaluar esta característica en conjunto con las anteriores, ya que el tipo de acceso condiciona varios de los aspectos mencionados en las características anteriores. Por ejemplo, una red privada puede ser más laxa en el mecanismo de consenso y en los lenguajes, y aún así ser segura, debido a que los nodos están interactuando en un sistema distribuido confiable.

3.2.6 Comunicación fuera de *Blockchain* (Oráculos)

Debido a que los *smart contracts* tienen acceso únicamente a sus propias secciones de información y a la información dentro de otros *smart contracts*, es interesante analizar la posibilidad de consumir información del mundo exterior sin importar cómo estos son generados.

Por lo tanto, para la comparación de las plataformas se evalúa si permiten comunicarse con oráculos, cuáles son estos y cuáles son sus principales características, centrándose especialmente en cómo resuelven la validez de la información y los problemas que presenta interactuar con un sistema centralizado de información.

3.2.7 Modificación de *smart contract* desplegado

Analizar si la plataforma brinda la posibilidad de actualizar un smart contract y cómo afecta a la seguridad del sistema. Se entiende que la modificación de un contrato puede mejorar la calidad general, ya que permite corregir errores y/o introducir mejoras en contratos desplegados. Sin embargo, al brindar esta posibilidad, se agrega más vulnerabilidad al sistema porque este recurso puede ser usado de forma maliciosa. Por este motivo se evalúan, en caso de existir, los mecanismos utilizados para ejecutar la actualización y cuáles son sus características principales, por ejemplo patrones de diseño.

4. Análisis de plataformas

En este capítulo se detallan las características principales de las plataformas seleccionadas y el análisis realizado en base a los criterios de comparación definidos.

4.1 Selección plataformas

Para realizar la selección de las plataformas a comparar primero se relevan las plataformas públicas y permisionadas y luego las plataformas privadas.

Se toman las veinte plataformas principales rankeadas en *CoinMarketcap*[Cmc19] como se muestra en la Figura 6 y se descartan aquellas que no cuenten con una solución en producción que soporte *smart contracts*.

#	Name	Symbol	Market Cap	Price	Circulating Supply	Volume (24h)	% 1h	% 24h	% 7d
1	Bitcoin	BTC	$60.681.847.608	$3.464,01	17.517.787	$5.043.937.584	0,19%	-1,15%	-3,04%
2	XRP	XRP	$12.445.862.137	$0,302352	41.163.466.448 *	$438.357.004	-0,01%	-2,08%	-1,76%
3	Ethereum	ETH	$11.254.870.706	$107,49	104.703.610	$2.519.334.757	-0,06%	-1,83%	-4,83%
4	EOS	EOS	$2.152.844.813	$2,38	906.245.118 *	$567.625.069	-0,34%	-0,79%	0,06%
5	Bitcoin Cash	BCH	$2.091.663.603	$118,83	17.602.425	$218.977.856	0,33%	-1,61%	-2,39%
6	Tether	USDT	$2.020.950.151	$1,00	2.020.855.917 *	$3.756.996.124	-0,18%	-0,33%	-0,98%
7	Litecoin	LTC	$2.016.740.848	$33,44	60.310.581	$820.461.891	-0,29%	-1,55%	2,84%
8	TRON	TRX	$1.696.159.966	$0,025442	66.668.553.480	$174.803.441	-0,31%	-1,76%	-11,17%
9	Stellar	XLM	$1.559.217.083	$0,081347	19.167.472.159 *	$106.729.578	0,25%	-1,68%	-14,41%
10	Bitcoin SV	BSV	$1.121.559.772	$63,72	17.601.461	$58.052.365	-0,14%	-2,46%	-9,82%
11	Cardano	ADA	$985.988.168	$0,038029	25.927.070.538	$12.940.762	-0,20%	-2,38%	-6,69%
12	Binance Coin	BNB	$870.405.723	$6,74	129.175.490 *	$59.942.158	0,94%	-1,45%	-4,05%
13	Monero	XMR	$719.406.512	$42,90	16.767.882	$45.359.366	0,26%	-0,37%	-6,83%
14	IOTA	MIOTA	$693.384.830	$0,249461	2.779.530.283 *	$5.980.121	-0,16%	-2,58%	-7,92%
15	Dash	DASH	$576.039.391	$66,90	8.611.031	$161.289.431	0,11%	-1,07%	-5,84%
16	NEO	NEO	$456.476.196	$7,02	65.000.000 *	$113.281.405	0,17%	-2,33%	-5,12%
17	Ethereum Classic	ETC	$423.399.621	$3,92	108.012.792	$156.280.488	0,03%	-1,39%	-6,91%
18	NEM	XEM	$353.382.892	$0,039265	8.999.999.999 *	$13.814.930	-0,18%	-3,89%	-24,96%
19	USD Coin	USDC	$302.531.642	$1,01	298.283.047 *	$26.365.349	0,06%	0,11%	-0,13%
20	Waves	WAVES	$281.634.096	$2,82	100.000.000 *	$23.050.834	-0,54%	3,62%	3,79%

Figura 6: *Listado de plataformas extraídas de CoinMarketCap el 3 de Febrero de 2019*

Las veinte plataformas extraídas son (en orden de popularidad): Bitcoin, XRP, Ethereum, EOS, Bitcoin Cash, Tether, Litecoin, TRON, Stellar, Bitcoin SV, Cardano, Binance Coin, Monero, IOTA, Dash, NEO, Ethereum Classic, NEM, USD Coin, Waves.

Las plataformas que se descartan por no contar con soporte de *smart contracts* o no contar una solución en producción son:

Bitcoin si bien muchas veces se utiliza la variable OP_RETURN para almacenar código programable, Bitcoin no provee un mecanismo para su ejecución debido a que no está diseñada para este fin.

XRP o **Ripple**, fue construido para un uso empresarial, es una red permisionada que ofrece a los bancos y proveedores una forma de gestionar dinero de forma confiable bajo la tecnología de *Blockchain* pero no presenta soporte para *smart contracts*.

Bitcoin Cash es un *hard-fork* de Bitcoin que tuvo efecto en Agosto del 2017, la modificación realizada fue el incremento del tamaño de los bloques a 8MB para ayudar al escalamiento de la plataforma pero sigue sin soportan la tecnología de *smart contracts*.

Tether es una criptomoneda que busca estar a la par con el precio del dólar americano y convertirse en una criptomoneda estable. No soporta *smart contracts*.

LiteCoin es una criptomoneda *peer-to-peer* que se basa en el protocolo de Bitcoin pero que difiere en los términos del algoritmo de *hash* utilizado. No soporta *smart contracts*.

Bitcoin SV es un *hard-fork* de Bitcoin Cash (BCH) originado por diferencias dentro del grupo de desarrolladores de BCH en relación a algunas decisiones técnicas (tamaño de bloques y sobreescritura de algunos scripts de la red). Bitcoin SV (Bitcoin Satoshi Vision) asegura mantener la visión del protocolo propuesto por Satoshi Nakamoto en su *white paper* lanzado en 2008 [Nak08], sigue sin soportar la tecnología de *smart contracts*.

Binance Coin es la criptomoneda de la plataforma de intercambio Binance, se diferencia de Bitcoin por facilitar la compra y venta de otras monedas digitales dentro de su propia plataforma.

Monero es una criptomoneda privada. La plataforma asegura que no se pueden rastrear los usuarios que originan las transacciones. No soporta *smart contracts*.

IOTA es una plataforma *open source* que permite trabajar con *ledgers* distribuidos pero que no trabaja con tecnología *Blockchain*, en su lugar utiliza un grafo acíclico dirigido.

Dash tambíen conocido como Darkcoin es una moneda digital centrada en la privacidad y en las transacciones instantáneas. No soporta *smart contracts*.

USD Coin es una criptomoneda que está respaldada uno a uno con dólares. Es utilizada para transferir fondos electrónicamente con la seguridad que su valor no va a variar. No soporta *smart contracts*.

Nem no es una plataforma que permita desarrollar *smart contracts*, simplemente provee una API para acceder a algunas características predefinidas en la *Blockchain*.

Cardano está elaborando un desarrollo para soportar *smart contracts* pero aún no cuenta con una solución en producción.

Además de la lista obtenida usando el método anterior, se agrega **Rootstock** (RSK), de origen Argentino que aún no tiene influencia en la capitalización del mercado mundial, encontrándose en el puesto número 1756 de *CoinMarketCap*. Se la considera relevante ya que cumple con las características deseadas, tiene gran presencia en la región y se impulsa fuertemente por ser la plataforma que implementa *smart contracts* sobre la tecnología de Bitcoin.

En base a los conocimientos adquiridos, también interesa poder comparar plataformas privadas ya que presentan otro enfoque a la resolución de los problemas presentes en la tecnología *Blockchain*. Debido a que en *CoinMarketCap* no se listan estas plataformas se hace un relevamiento en base a la popularidad

(consultas realizadas en StackOverflow), resultando en: **Hyperledger Fabric**, **Quorum** y **Corda**.

Finalmente el conjunto seleccionado para investigar en profundidad resulta en:

1. Ethereum
2. Ethereum Classic
3. EOS
4. TRON
5. Stellar
6. NEO
7. Waves
8. RSK
9. Quorum
10. Hyperledger Fabric
11. Corda

4.2 Ethereum

Es una plataforma desarrollada en 2015 por Vitalik Buterin, pensada como un método alternativo para la creación de aplicaciones descentralizadas. Es la plataforma que pone el concepto de *smart contracts* sobre la mesa. Es decir, si bien al momento de su creación existían plataformas que permitían agregar *scripts* a sus bloques, Ethereum fue un paso más adelante y creó un protocolo que permite desarrollar aplicaciones y operaciones complejas dentro de la *Blockchain* utilizando lenguajes Turing completos. Si bien la plataforma Ethereum provee un *framework* para cualquier tipo de aplicación distribuida, uno de los principales casos de uso es la automatización de interacciones entre pares y/o la coordinación de un grupo de acciones sobre una red.

Utiliza una criptomoneda propia llamada Ether (ETH), cuya unidad mínima se llama Wei y representa $1x10^{-18}$ Ether. Es utilizada para pagar los costos de enviar transacciones a la plataforma. Cada transacción que es enviada a la red requiere cierta cantidad de Gas, que es lo que representa la potencia computacional que es necesaria para procesar dicha transacción. Para procesar una transacción e incluirla dentro de un bloque, los mineros esperan ser compensados, esto se lleva a cabo

estableciendo el precio por Gas de cada transacción. Las unidades de Gas se definen utilizando la unidad GWei (1 ETH = 1000000000 Gwei). Por ejemplo si se envía ETH de una cuenta a otra (operación que cuesta 21000 Gas) si se define el precio del gas a 1 Gwei, el costo de la operación sería 0.000021 ETH.

El mecanismo de consenso definido es el mismo que utiliza la plataforma Bitcoin y tal como fue explicado en el Capítulo 2, todos los nodos participantes en la red deben resolver un *puzzle* criptográfico para poder generar el nuevo bloque de la cadena y son recompensados cada vez que lo hacen. Si bien es uno de los mecanismos más populares y seguros dentro de esta tecnología, sus propias características impactan negativamente en la *performance* y en los recursos que necesita. Es decir, al tener que realizar pruebas criptográficas tan complejas, los tiempos de creación de nuevos bloques son altos (aproximadamente 16 segundos por bloque) así como el consumo energético.

También presenta otros problemas de escalabilidad relacionados al almacenamiento y los datos para mantener la cadena. Al igual que en Bitcoin, Ethereum se ve afectado por el hecho de que cada transacción debe ser procesada por todos los nodos que participen en la red. En el caso de Bitcoin el tamaño de la cadena crece alrededor de 1MB por hora, por lo que si su tasa de creación de bloques aumentara también lo haría en igual proporción su cadena, llegando a tamaños inmanejables. Este mismo problema podría presentarlo Ethereum, incluso peor si se consideran todas las transacciones generadas entre aplicaciones y no solo las generadas por intercambios de monedas.

La máquina virtual de Ethereum (EVM) es quien se encarga de ejecutar los *smart contracts* sobre la *Blockchain*, está basada en una pila y almacena sus datos de forma *big-endian* con palabras de 256 bits. Una de las diferencias más grandes con el resto de las plataformas al momento de su creación, es que su máquina virtual es Turing completa, es decir, puede codificar cualquier cálculo que pueda llevarse a cabo incluso implementar bucles infinitos.

Smart contracts

Los *smart contracts* son cuentas no controladas por una persona, que ejecutan su código fuente dentro de la EVM cada vez que reciben una transacción (sin importar que esta provenga de una cuenta de usuario o de otro *smart contract*). Estas transacciones pueden tener o no un *payload* que es utilizado para especificar el tipo de operación que el contrato debe ejecutar o para enviar información adicional.

Para el desarrollo de los contratos se utiliza un lenguaje de alto nivel (influenciado por C++, Python y JavaScript) llamado Solidity, que es orientado a objetos y que fue diseñado exclusivamente para implementar estos *smart contracts* sobre la máquina virtual de Ethereum. Es decir, programas que definen el comportamiento de las cuentas dentro del estado de esta plataforma. El lenguaje es fuertemente tipado, permite definir tipos de datos complejos, importar bibliotecas e invocar otros *smart contracts*. También presenta un enorme conjunto de operaciones primitivas que permiten a los desarrolladores implementar casi cualquier tipo de contrato. En la Figura 7 se muestra un ejemplo de cómo es un contrato definido en este lenguaje. Se puede ver que el mismo está compuesto por un constructor (en el cual se guarda la dirección del creador), una función para obtener la dirección (que tiene visibilidad pública) y una función que permite que sea deshabilitado.

```
pragma solidity ^0.5.11;

contract CreatorAddress {

    // stores the address of the contract creator
    address payable creator;

    // contract constructos, it's executed once when the contracts is put into a transaction
    constructor () public {
      creator = msg.sender;
    }

    function getAddress() public view returns (address creatorAdress) {
        return creator;
    }

    function kill() public {
        if (msg.sender == creator)
            selfdestruct(creator);  // kills this contract and sends remaining funds back to creator
    }
}
```

Figura 7: *Smart contract con destrucción, basado en ejemplo RSK*.[Dem15]

Para ejecutar el código en la EVM, el mismo es traducido a un lenguaje de bajo nivel, basado en stack denominado *bytecode*, que dentro del ecosistema Ethereum

se lo referencia como “EVM code”. En la Figura 8 se puede observar un ejemplo de la traducción realizada.

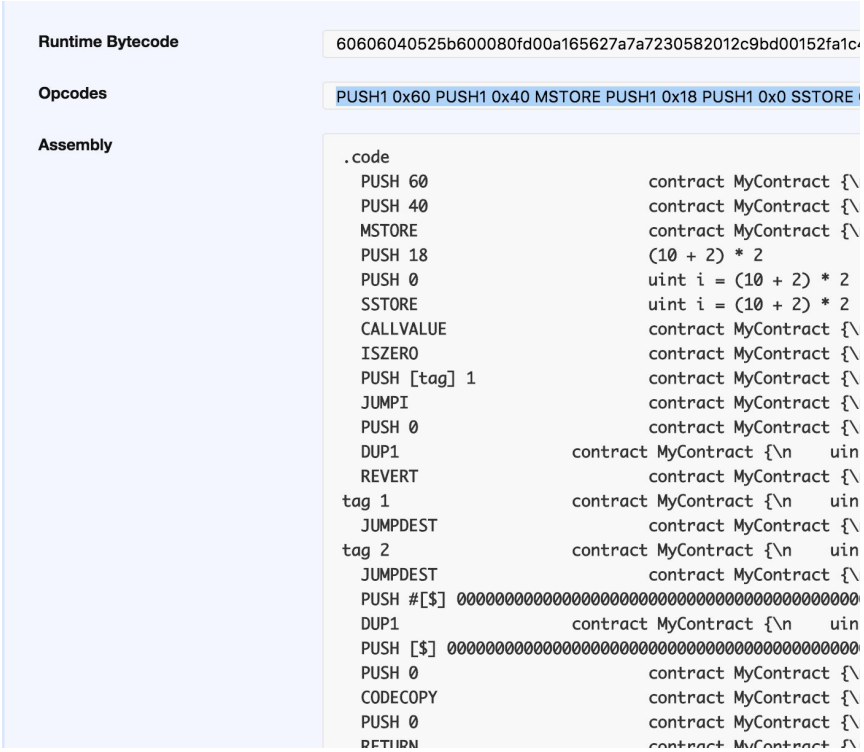

Figura 8: *Ethereum bytecode and opcode*.[Med19]

Estas operaciones tiene acceso a tres tipos distintos de espacio de información:

- una pila de tipo *LIFO (last in first out)* en donde se puede poner y quitar datos
- memoria, un arreglo infinito de *bytes*
- almacenamiento del contrato, es a largo plazo y utiliza un mapa clave-valor.

También pueden acceder a algunos valores que vienen dentro de la invocación; cabezal del bloque, dirección del remitente y datos de entrada (parámetros).

Cada una de estas operaciones requiere que un minero utilice cierta cantidad de capacidad computacional, por lo que debe ser capaz de recibir una recompensa acorde. Es aquí donde se utiliza el concepto de Gas, ya que permite especificar cuál es el esfuerzo computacional que debe hacer un nodo (minero) para ejecutar el *bytecode* que le es solicitado. En la Figura 9, se muestran ejemplos de operaciones y sus respectivos costos en Gas.

Operación	Precio en unidades gas
ADD	3
MUL	5
SUB	3
DIV	5
SDIV	5
MOD	5
SMOD	5
ADDMOD	8
MULMOD	8
EXPBASE	10
EXPBYTE	10
SIGNEXTEND	5

Figura 9: *Relación Gas por operación de smart contract*

Una particularidad del Gas es que el precio lo fija el usuario interesado en ejecutar un contrato. Es decir, quién quiera ejecutar algo dentro de esta plataforma deberá establecer cuánto está dispuesto a pagar por una unidad de Gas (precios establecidos en Gwei).

Otra de las razones por las cuales Ethereum implementa el Gas es para poder controlar que un contrato termine su ejecución. Al invocar un contrato el usuario decide cuánto será el límite de Gas dispuesto a consumir. De esta forma se evita el problema de la no terminación, con cada operación ejecutada se decrementa la cantidad de Gas disponible para seguir ejecutando, llegando eventualmente a cero.

4.3 Ethereum Classic

Es la plataforma creada originalmente como Ethereum pero su inicio ocurre el 20 de julio de 2016, en el bloque número 1.920.000, a partir del problema ocasionado por el ataque que afectó a la *Decentralized Autonomous Organization* (DAO). Es decir, es la plataforma que implementa y mantiene, hasta el día de hoy, la filosofía presentada en el paper de Vitalik Buterin sin tener en cuenta el *fork* generado para la solución del ataque del DAO.

Para resolver este problema se llevaron a cabo ciertas medidas que dividieron a la comunidad de Ethereum en dos partes. Quienes no apoyaron las medidas tomadas

manifestaron varios aspectos del proceso que violan algunos de los principios de Ethereum; inicialmente se generó un *soft fork* simplemente para poder generar una lista negra y poder censurar ciertas transacciones, lo que muchos consideran una decisión apresurada que viola algunos valores y principios del protocolo. Luego se generó un *hard fork* para cambiar las reglas de la plataforma, lo cual también se consideró como una decisión apresurada que generó un cambio irregular en el estado de Ethereum, violando algunas de sus propiedades fundamentales; inmutabilidad y fungibilidad *ledger*. Según la comunidad de ese entonces, estas propiedades no pueden ser violadas en la tecnología *Blockchain*. Si no se tiene la propiedad de inmutabilidad, la validez de todas las transacciones podría cuestionarse, lo que no solo deja transacciones abiertas a fraude si no que puede significar un comportamiento desastroso a cualquier aplicación distribuida. Por otra parte, la propiedad de fungibilidad es una característica asociada al dinero, en donde una unidad es igual a otra, por ejemplo un dólar tiene el mismo valor que otro dólar, y en el caso de Ethereum esto dejaba de ser de esta forma. A partir de esto es que parte de la comunidad de Ethereum se separa y genera lo que llaman "*Ethereum Classic code of Principles*".

Código de principios (*Ethereum Classic code of Principles*)

- el propósito de Ethereum Classic es proporcionar una plataforma descentralizada, que ejecute aplicaciones descentralizadas, que se ejecuten exactamente como se programó sin posibilidad de tiempo de inactividad, censura, fraude o interferencia de terceros.
- el código es ley, no habrá cambios en el código Ethereum Classic que violen las propiedades de inmutabilidad o fungibilidad del ledger, las transacciones o el historial contable no se pueden revertir ni modificar por ningún motivo.
- los *forks* y/o cambios en el protocolo subyacente, sólo se permitirán para actualizar la tecnología en la que Ethereum Classic opera.
- el desarrollo interno del proyecto puede ser financiado por cualquier persona, ya sea a través de un tercero de confianza de su elección o directamente, utilizando la moneda de su elección por proyecto y siguiendo un protocolo de *crowdfunding* transparente, abierto y descentralizado.

- cualquier individuo o grupo de individuos puede proponer mejoras o actualizaciones a los activos de Ethereum Classic existentes o propuestos.
- cualquier individuo o grupo de individuos puede usar la plataforma descentralizada Ethereum Classic para crear aplicaciones descentralizadas, realizar ventas colectivas, crear organizaciones/corporaciones autónomas o para cualquier otro propósito que consideren adecuado

Smart Contracts

Los aspectos técnicos son los mismos que en Ethereum, la gran diferencia está en la filosofía que ambas tienen. Es decir, los *smart contracts* son desarrollados utilizando los mismos lenguajes (Figura 10 *HelloWorld* en Solidity), el proceso para desplegar un contrato o realizar una transacción es la misma, utiliza el concepto de Gas y se tiene un precio asociado a cada uno de las operaciones que se pueden utilizar. Usa el mismo mecanismo de consenso (*PoW*) y posee las mismas características asociadas a la criptomoneda (mismos incentivos, inflación, etc.).

```
pragma solidity >=0.4.22 <0.7.0;

/**
 * @title HelloWorld
 * Pablo
 */
contract HelloWorld {

    string greeting = "Hello World!!";

    /**
     * @dev Return value
     * @return string greeting
     */
    function greet() public view returns (string memory){
        return greeting;
    }
}
```

Figura 10: *Smart contract HelloWorld desarrollado con Solidity*

4.4 EOS

Es una plataforma anunciada en 2017 por la compañía Block.One y liberada como *open-source* en junio de 2018, es concebida como una plataforma para aplicaciones distribuidas que permite resolver algunos problemas presentes en las plataformas Bitcoin y Ethereum, así como también eliminar los costos de utilización.

La criptomoneda que utiliza la plataforma, cuya unidad mínima 1/10000 no tiene nombre, algo que sí sucede en otras plataformas. La misma es utilizada tanto para seleccionar los nodos que van a formar parte del mecanismo de consenso (*DPoS*), como para reservar recursos de ejecución. A diferencia de otras plataformas esta criptomoneda no es creada cuando nuevos bloques son agregados a la cadena, al momento del lanzamiento se crearon aproximadamente 1B (1 billón americano) de monedas. Esta criptomoneda posee inflación, estipulada en 5%, con lo que se espera que luego de agotarse los 1B de monedas en circulación, 50M sean creados.

La plataforma está definida como *feeless* (sin cargo). Esto quiere decir que no se debe pagar para enviar una transacción o por ejecutar un *smart contract*, si no que se deben utilizar los *tokens* para obtener recursos que les permita a los desarrolladores/usuarios ejecutarlos. Estos recursos son: poder computacional (CPU), ancho de banda y almacenamiento de logs (*Disk*) y almacenamiento del estado (RAM). Tanto el ancho de banda como el poder computacional tienen roles muy importantes, por un lado el acceso instantáneo a los datos y por otro el acceso a los datos a largo plazo. Dentro de la *Blockchain* se mantienen todas las acciones realizadas sobre la cadena, que son descargadas por los nodos completos y procesados para crear el estado de cada aplicación. Si el esfuerzo de procesar la cadena crece muy rápido será necesario crear “respaldos” intermedios del estado de la cadena y “desechar” el resto de la información histórica, por lo que es muy importante conocer y manejar el poder computacional de forma correcta.

Para poder utilizar estos recursos, los nodos productores de bloques publican su capacidad disponible (procesamiento, almacenamiento y RAM) y los usuarios interesados pueden reservar cierta capacidad de procesamiento proporcional a la

cantidad de *tokens* que tengan. Si el usuario no desea utilizar toda la capacidad que sus *tokens* le brindan puede delegar o alquilar, parte de su poder de procesamiento o ancho de banda pero no su almacenamiento RAM.

Tiene soporte para roles y permisos. En el caso de las cuentas, existen dos permisos por defecto; *active* y *owner*. El permiso *Owner* (dueño) es el que está asociado a la administración de niveles de operaciones de una cuenta y es el permiso "padre" de el resto de los permisos. El permiso *Active* (activo) está asociado a operaciones comunes, por ejemplo ejecución de *smart contracts*, transferencia de *tokens*, compra de RAM, etc. A su vez se pueden crear nuevos permisos los cuales deben ser controlados apropiadamente dentro de los *smart contracts*. También cuenta con el permiso eosio.code que es utilizado por el *smart contract* para definir si este puede comunicarse con otros contratos. Este permiso debe ser provisto por la cuenta a la cual está asociado el contrato, agregandolo como uno de sus permisos activos.

A diferencia de Bitcoin y Ethereum implementa un mecanismo de consenso *DPoS* (*Delegated Proof of Stake*) que utiliza veintiún nodos productores de bloques, llamados nodos completos, que son elegidos por todos aquellos usuarios que posean *tokens* de la plataforma.

En caso de encontrarse un *smart contract* que se comporte de una manera impredecible o que este pueda hacer un uso irracional de los recursos, el productor de bloques puede revertir esa situación. Al ser estos nodos, quienes deciden qué transacciones son incluidas en los bloques, tienen la habilidad de congelar aquellas cuentas que envíen este tipo de transacciones. Para esto se realiza una votación entre los veintiún nodos y si quince de ellos están de acuerdo, podrán efectuar los cambios en discusión. También pueden, en caso de encontrar una aplicación que corra sin parar, cambiar el código sin tener que realizar un *hard fork* en la *Blockchain* entera (proceso similar al congelado de cuentas).

Smart contracts

EOS utiliza el lenguaje C++ para definir sus *smart contracts* que luego son compilados a *web assembly* para poder ser guardados en su *Blockchain*.

Cada contrato es definido como una clase que hereda de la clase `eosio::contract`, en la cual se pueden definir atributos y métodos que representarán variables persistentes en el estado o acciones que pueden ser invocadas por un usuario u otro *smart contract*, siempre y cuando se tengan los permisos correspondientes (eosio.code). Al heredar de `eosio::contract` los contratos poseen varios atributos que son utilizados para la generación del archivo compilado (WASM) y el código ABI. A diferencia de otras plataformas, cada vez que se invoca una acción, una nueva instancia de un contrato es creada y esta es destruída luego de su ejecución. Por este motivo, si dentro de un contrato es necesario utilizar información que represente el estado, debe ser cargado cuando el contrato comienza a ejecutar, es decir, definirlo en su constructor y debe ser guardado durante su destrucción. Es importante destacar que solo un contrato puede estar asociado a una cuenta, y que pueden ser modificados/actualizados solamente por la cuenta a la cual están asociados. En la Figura 11 se puede observar cómo se utilizan las clases mencionadas previamente.

```
#include <eosio/eosio.hpp>

using namespace eosio;

class [[eosio::contract]] hello : public contract {
  public:
      using contract::contract;

      [[eosio::action]]
      void hi( name user ) {
         print( "Hello, ", user);
      }
};
```

Figura 11: *Hello World implementado en C++*. [Eos19]

4.5 TRON

Es una plataforma fundada en julio de 2017 y lanzada como *open source* en diciembre de ese año. El propósito de este proyecto fue desarrollar una plataforma que permita resolver los problemas de escalabilidad vistos en Bitcoin y Ethereum. Si bien el bloque génesis fue creado en julio de 2018, la utilización, o mejor dicho, comercialización ha posicionado a TRON en el doceavo lugar en *CoinMarketCap* al momento de escribir este documento. El código de la *Blockchain* está implementada en Java y originalmente fue un *fork* de Ethereum.

La plataforma utiliza la criptomoneda TRX, cuya unidad mínima es SUN que vale 1/1000000 TRX. Esta criptomoneda puede ser usada para votar a los nodos super representativos o para obtener puntos de ancho de banda y así realizar distintas transacciones. Estos puntos son consumidos de acuerdo al tamaño en *bytes* de la transacción, es decir si el tamaño son 20 *bytes* se utilizarán 20 puntos. Algo interesante de la plataforma es que cada usuario tiene una cuota gratis de puntos de ancho de banda por día, pasados esos puntos se deberán congelar recursos en TRX para acceder a más puntos o pagar las cuotas de las transacciones utilizando TRX directamente.

Como se detalla en la Figura 12, TRON adopta una arquitectura de 3 capas: almacenamiento, núcleo y aplicación. El protocolo utilizado para comunicarse entre las distintas capas está especificado utilizando Google Protocol Buffers lo que da soporte multi lenguaje (solo para los componentes descritos dentro de la arquitectura, no los *smart contracts*) .

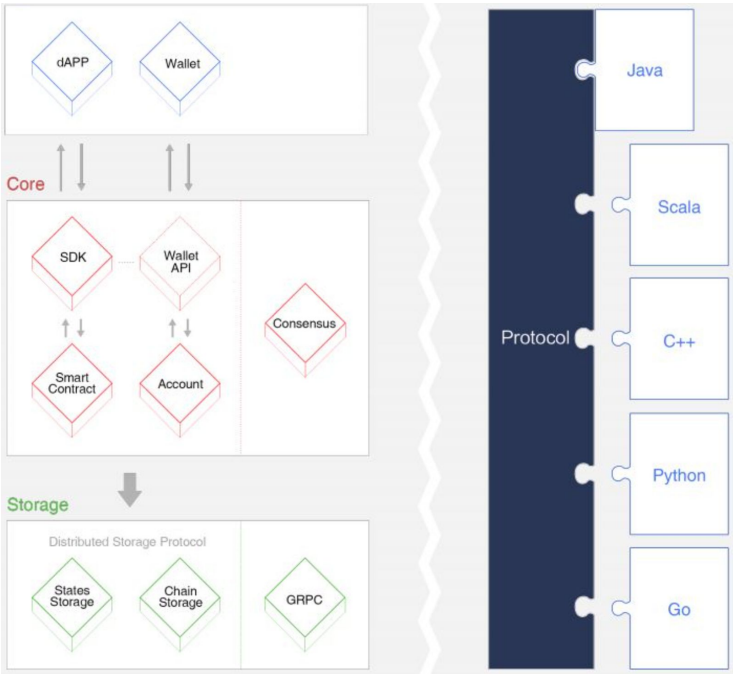

Figura 12: *Arquitectura de TRON*. [TF19].

Dentro del núcleo se incluyen varios módulos: *smart contracts*, manejo de cuentas de usuarios y el mecanismo de consenso. Utiliza una máquina virtual basada en *stack* con un conjunto de instrucciones optimizado. Incorpora Solidity como lenguaje de programación de *smart contracts* y prevé la incorporación de otros lenguajes. El mecanismo de consenso utilizado es *Delegated Proof of Stake* (*DPoS*).

Para la capa de almacenamiento TRON implementa un protocolo único distribuido, que consiste en dos componentes diferentes: por un lado el almacenamiento de los bloques de la cadena y por otro el almacenamiento del estado.

Para el almacenamiento de bloques utiliza LevelDB, una biblioteca de almacenamiento desarrollada por Google, que tiene una alta *performance* y algunas de sus características son que almacena claves y valores en matrices arbitrarias de *bytes* en donde los datos son ordenados por su clave, soporta operaciones por lotes (*GET*, *PUT, DELETE*), presenta iteradores en dos direcciones (adelante/atrás) y puede comprimirse fácilmente usando la librería Snappy de Google.

Para el almacenamiento del estado en los nodos completos utiliza KhaosDB, que es una base de datos en memoria, es capaz de almacenar todos los *forks* de cadenas

que se están generando y presenta mecanismos para cambiar el *fork* activo en períodos muy cortos de tiempo.

TRON utiliza Google Protocol Buffers que es un protocolo de comunicación, de almacenamiento, entre otras características, no asociado a un lenguaje o plataforma, que permite serializar estructuras de datos. En comparación con XML es de 3 a 10 veces más pequeño y entre 20 a 100 veces más rápido. Además presenta una API RESTful Http, es decir que comparte la misma interfaz que Protobuf pero está orientada para utilizarse con clientes Javascript.

Tiene 27 nodos *Super Representatives* (SRs), que son los nodos encargados de producir los bloques para la cadena. Es capaz de producir 2000 transacciones por segundo (TPS) y de generar un nuevo bloque cada tres segundos. El SRs que genere un bloque, ganará 32 TRX. Es decir: 336384000 TRX anualmente para los 27 SRs. Existen tres tipos de nodos: *Witness*, configurados por los SRs y se encargan, principalmente, de producir y proponer bloques; *Full Node*, nodos que proveen APIs y se encargan de difundir todas las transacciones y bloques de la cadena; *Solidity Node*, solamente sincronizan transacciones que ya están validadas. Similar a lo que sucede con Bitcoin, para que una transacción sea considerada válida se deben tener 19 bloques confirmados (el que agrega la transacción y 18 más), es decir, aproximadamente un minuto en validar.

Al igual que en otras plataformas, el modelo de TRON se basa en la utilización de cuentas de usuario, es decir, todas las entidades que participan de la red son cuentas identificadas por su dirección y un conjunto de otras propiedades, por ejemplo: balance de TRX, ancho de banda, etc. Estas cuentas son quienes pueden intercambiar TRX entre sí, implementar y manipular *smart contracts* y aplicar para ser un nodo *Super Representatives* (SRs).

Smart contracts

Al igual que en Ethereum, los *smart contracts* utilizados en la plataforma TRON son escritos en Solidity y bajo el mismo proceso. Es decir, luego de ser escritos y testeados son compilados a *bytecode*, luego desplegados en la red TRON y ejecutado por la máquina virtual. Una vez que los contratos son desplegados, estos

son consumidos a través de su dirección, utilizando la información provista por la interfaz ABI.

Si bien ambas plataformas utilizan Solidity como lenguaje de programación de sus *smart contracts*, en TRON se utiliza una versión modificada. Estas modificaciones sólo involucran cambios para poder soportar las monedas utilizadas en la plataforma (TRX y SUN), por lo que el resto de la sintaxis corresponde a la versión ^0.4.24 (momento en el que se hizo el *fork*).

Para poder utilizar la plataforma TRON de forma activa, es decir enviando transacciones o creando *smart contracts*, es necesario contar con dos tipos de recursos: ancho de banda (*bandwidth*) y energía. El ancho de banda es el recurso que permite a los usuarios participar en la elección de los nodos super representativos y poder enviar transacciones a la red sin necesidad de pagar por ello. La energía es el recurso necesario para poder ejecutar un *smart contract* (concepto similar al Gas de Ethereum).

Para generar ancho de banda o energía el usuario debe, a través de su billetera electrónica, congelar una cierta cantidad de TRX. Esto quiere decir que el usuario decide cuántos de sus TRX van a quedar inutilizados por un período de tres días (no los va a poder intercambiar ni consumir). Cuantos más sean los TRX que se congelen, mayor será la cantidad de ancho de banda o energía que tenga el usuario.

Generalmente no se cobra para la mayoría de las transacciones (*), sin embargo por algunas restricciones del sistema y para ser justo en algunas ocasiones si se hace. El flujo normal a la hora de ejecutar transacciones es el siguiente:

1. Se consume el ancho de banda dado por la cantidad de TRX que la cuenta tiene congelados, si no es suficiente se va al paso siguiente.
2. Se consume el ancho de banda que es grátis por día, si no es suficiente se va al paso siguiente.
3. Se le cobra a la cuenta que envía la transacción de la siguiente forma: tamaño de *bytes* de la transacción * 10 SUN.

**Cualquier transacción de consulta es gratis, no cuesta energía ni ancho de banda.*

4.6 Stellar

Los fundadores de esta plataforma son Jed McCaleb y Joyce Kim. En julio del 2014 lanzaron el proyecto *open source* junto con la creación de *The Stellar Development Foundation* (*SDF*), una organización sin fines de lucro que apoya el desarrollo y crecimiento de la red. *SDF* y Stellar tienen como cometido destrabar la economía mundial mediante la fluidez del dinero, la apertura de los mercados y el empoderamiento de las personas. La fundación se encarga de actualizar el código fuente de Stellar, apoyar las comunidades técnicas y empresariales alrededor de la plataforma, además de realizar alianzas con empresas e instituciones.

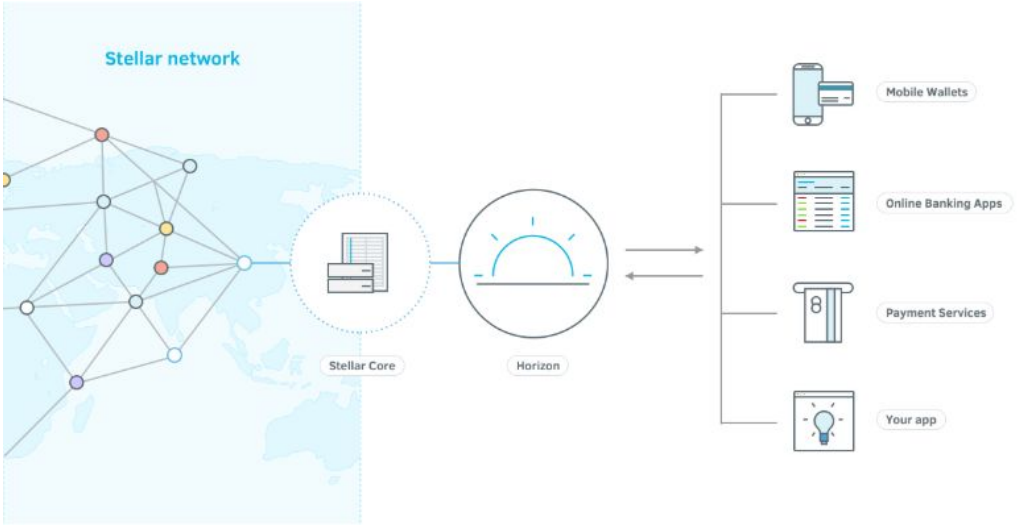

Figura 13: *Arquitectura Stellar*. [Ste19] .

Como se muestra en la Figura 13, la mayoría de las aplicaciones interactúan con Stellar a través de Horizon, un servidor RESTful HTTP API. Esto brinda la posibilidad de realizar transacciones, verificar balances de cuenta y suscribirse a eventos directamente desde un explorador web. Stellar mantiene actualizada los SDK (kit de desarrollo de *software*) para la comunicación con Horizon en los siguientes lenguajes de programación: JavaScript, Java y Go. La comunidad también mantiene actualizados los SDKs en Ruby, Python y C#.

Luego se encuentra el *core*, este *software* realiza el trabajo de validación y consenso del estatus actual de la *Blockchain* con otras instancias del *core* para cada

transacción, usando el protocolo de consenso propio *SCP* (*Stellar Consensus Protocol*).

La plataforma utiliza la criptomoneda Lumen (XLM) y cada transacción que es realizada conlleva un costo de 0.00001 XLM. Esta plataforma a diferencia de Bitcoin, no mina su criptomoneda, desde el principio se crearon 100.000 millones de unidades que están divididos entre: personas aleatorias (50%), empresas u organizaciones del ecosistema (25%), titulares de Bitcoin o Ripple (20%). El 5% restante, se reservan para mantener los costes operativos de la plataforma y seguir con su desarrollo. Para utilizar la plataforma es necesario tener Lumens. Por protocolo, cada cuenta debe tener siempre al menos 0.5 Lumens guardados por cada tipo de *asset* que utiliza. La plataforma justifica el uso de la moneda por dos motivos: disuasión de *SPAM* y liquidez. La primera busca desincentivar el uso de la red con *SPAM*, como en otras redes existen cuentas falsas y comportamiento malicioso, por lo que cobrar por la ejecución de transacciones desmotiva este tipo de comportamiento. Stellar fue construida para el uso de pago en diferentes monedas, por ejemplo, enviar una cantidad en dólares y que la otra persona reciba pesos.

Los puntos que se destacan de esta plataforma son: se puede hacer un intercambio de monedas sin necesidad de usar una entidad de intercambio de divisas como intermediario, ya que la red lo resuelve de forma automática, y puede manejar cualquier tipo de *asset* (puede ser moneda u otro tipo de recurso). El costo por transacción es bajo, lo que habilita el uso de *micropayments*. El tiempo de confirmación de las transacciones también es bajo, entre 2 y 5 segundos. Tiene aliados importantes en el mercado como IBM y Deloitte.

Los *assets* son activos que son usados dentro de la plataforma, pueden ser rastreados, retenidos y transferidos. Cualquier *asset* puede ser intercambiado por otro. Excepto por los Lumens, todos los *assets* tienen un tipo y un *Issuer*, que determina que cuenta creo el activo. Con esta característica es que Stellar no sólo promueve el intercambio de moneda si no también de recursos.

Cuando se emite un *asset*, se pueden establecer ciertas características, una interesante a destacar es imponer el requisitos de *KYC* (*Know Your Customer*). Es decir, antes de que una persona pueda interactuar con el *asset*, tiene que identificarse, esto facilita el conocimiento de los clientes y la confirmación de la identidad, pero rompe con el esquema de anonimización que manejan algunas plataformas de *Blockchain*.

Smart Contracts

A diferencia de Ethereum, los *smart contracts* en Stellar son más limitados. No se pueden programar de forma tal que queden almacenados en la *Blockchain* para ser ejecutados en cualquier momento ya que tienen otro paradigma. En esta plataforma, un *smart contract* es un conjunto de transacciones formadas por operaciones que tienen un efecto sobre el *ledger.* Existen trece posibles operaciones, desde la creación de una cuenta, la transferencia de un *asset* hasta el cálculo de la inflación en la plataforma. Una vez creada la transacción, la misma es firmada por los participantes para autorizar su ejecución. Las transacciones son atómicas, es decir, son exitosas o no, no hay un estado intermedio. Además, cada transacción tiene asignado un número de secuencia, permitiendo que los nodos puedan seguir el orden de ejecución correcto. Luego de N transacciones ejecutadas, la próxima transacción válida es la N+1, cualquier transacción con número de secuencia igual a N o N+2 fallará.

Cuando se quiere implementar un *smart contract* en Stellar, se deben considerar las siguientes cuatro restricciones:

1) El uso de la *multi signature* (multifirma), es decir, prever qué actores son necesarios para la autorización (firma) de la transacción.
2) El *batching*, lote de operaciones que se incluyen en una transacción y el funcionamiento entre sí, debido a que deben ser usadas de forma tal que se cumpla la atomicidad de la transacción, es decir que si una operación falla, todas las operaciones subsiguientes de la transacción fallarán.

3) El número de secuencia, que puede ser manipulado para forzar a que una transacción no suceda si una transacción alternativa es aceptada (*if-then-else*).
4) Los límites temporales, que son establecidos para determinar que una transacción es válida en el tiempo que es ejecutada. Esto se utiliza cuando se quieren definir fideicomisos.

A continuación, en la Figura 14 se muestra un ejemplo de *crowdfunding* donde si se alcanza el objetivo formulado, el dinero es transferido al proyecto recaudador y si no, el dinero es devuelto a los inversores:

Figura 14: *Ejemplo de smart contract.*[Ste19] .

Cuatro cuentas interactúan en este contrato. Las cuentas F y G que sponsorean la recaudación y autorizan las transacciones relativas a la cuenta C, C es la cuenta que se utiliza para guardar el dinero recaudado y S es la cuenta del proyecto en caso de alcanzar la meta fijada.

Las transacciones TXN1 y TXN2 son creadas y enviadas por una de las dos partes que patrocinan la campaña de *crowdfunding*. La transacción TXN1 crea la cuenta de haberes C. Esta cuenta se financia con un saldo inicial para que sea válida en la red. En la transacción TXN2 se configura para que las transacciones realizadas en C sean firmadas por F y G. Este mecanismo de confianza funciona para proteger a los donantes de acciones maliciosas sobre la recaudación.

La transacción TXN3 se crea y se envía a la red para comenzar la campaña de crowdfunding. Crea una oferta en la red que vende los *tokens* de participación a una cierta tasa por *token*. Dado que se crea una cantidad limitada de *tokens* para la campaña de *crowdfunding*, los *tokens* tienen un precio que permite recaudar a través de las ventas.

La transacciones TXN4 y TXN5 están pre-firmadas, pero no han sido ejecutadas aún y están publicadas en la red. Ambas transacciones tienen un tiempo mínimo de finalización del período de *crowdfunding* para evitar que se ejecuten antes de lo acordado por las partes patrocinadoras. TXN4 transfiere el monto recaudado a la cuenta S. TXN5 evita que se vendan todos los *tokens* restantes al cancelar la oferta y permite a los donantes recuperarlos.

4.7 Neo

Es un proyecto que surge en 2014 bajo el nombre de Antshares en China y lanza su *Blockchain* de producción en 2016. Neo busca generar una "economía inteligente" con una red distribuida utilizando la tecnología de *Blockchain,* la identidad digital para digitalizar activos y el uso de *smart contracts* para que estos activos digitales se autogestionen.

Los activos digitales son activos de la vida real que toman forma en la *Blockchain*, esta digitalización es descentralizada, segura y trazable. Los activos que son registrados a través de una identidad digital están protegidos por ley. Existes dos tipos de activos digitales en la plataforma: primero los globales, que se registran en el espacio del sistema y pueden ser utilizados por todos los clientes y *smart contracts*; y segundo los contractuales, que se registran en el área de almacenamiento privado del *smart contract* y requieren que un cliente compatible los reconozca.

La identidad digital se refiere a la información de identidad de individuos, organizaciones y otras entidades que existen en forma electrónica. El sistema de identidad digital más usado se basa en el estándar PKI X.509 (Infraestructura de clave pública). En NEO, se implementa un conjunto de estándares de identidad

digital compatibles con X.509 y el módulo de verificación de identidad, tanto para emitir o para usar identidades digitales incluye: reconocimiento facial, huellas digitales, voz, SMS y otros métodos de autenticación de múltiples factores.

NEO tiene una criptomoneda y un *token* nativos: NEO y NeoGas. El primero se divide en dos, 50 millones de NEOs se distribuyeron de forma proporcional en el momento del *crowdfunding* y los otros 50 millones son reservados para tareas de desarrollo, operaciones y mantenimiento dentro de la plataforma. NeoGas se usa como insumo para la ejecución de los *smart contracts* y transacciones en la red.

El mecanismo de consenso elegido por la plataforma es *dBFT* (*Delegated Byzantine Fault Tolerant*). Además se crean nuevos bloques en un intervalo de 15 a 20 segundos. Este mecanismo combinado con la tecnología de identidad digital, es decir que los nodos *bookkeepers* pueden tener nombres reales de individuos o instituciones, es decir que es posible: detener, revocar o transferir la propiedad debido a decisiones judiciales. Existen dos tipos de nodos: los ordinarios y los *bookkeepers.* Los nodos ordinarios usan el sistema para transferir, intercambiar y aceptar la información del *ledger*. Los nodos *bookkeepers* mantienen el *ledger* actualizado, ejecutando el mecanismo de consenso.

NEO posee una arquitectura distribuida que permite alta redundancia de *storage y* utiliza oráculos como una fuente confiable y externa de información.

Smart contracts

Una de las ventajas de desarrollar *smart contracts* en NEO es que se pueden escribir en C#, Java y otros lenguajes de programación convencionales y luego ser compilados en la máquina virtual NeoVM. Los mismos pueden ser invocados múltiples veces.

Los *smart contracts* se pueden invocar mutuamente pero no de forma recursiva, es decir que el *smart contract* A puede invocar al *smart contract* B, pero el B no al A. La recursión se puede escribir únicamente dentro del contrato. Además la relación entre contratos debe ser establecida de forma estática, no se puede establecer en

tiempo de ejecución. Esto permite que el comportamiento del contrato sea determinista antes de la ejecución.

La invocación del contrato es disparada por una transacción especial, que puede acceder y modificar el estado global del sistema y el área de almacenamiento privada del *smart contract* en tiempo de ejecución. Por ejemplo, es posible crear un activo digital en un contrato, guardar información e incluso crear un nuevo contrato, cuando está siendo ejecutado. Durante la ejecución del *smart contract* se realiza el cobro por instrucción realizada en NeoGas. Si se acaba el monto estipulado, el contrato dará error, se detendrá la ejecución y todos los cambios realizados por el mismo serán revertidos.

Una característica a destacar es *Function Contract*, que es usada para proveer de forma pública funciones que son usadas habitualmente y pueden ser invocadas por diversos *smart contracts*. De esta forma, los desarrolladores pueden reutilizar código. Cada función, cuando es puesta en producción, puede configurar si necesita un área de *storage* privado de lectura/escritura, obteniendo la persistencia del estado. Las funciones deben ser pre-desplegadas en la *Blockchain* para ser invocadas y cuando se quieran inhabilitar, deben ser removidas mediante la invocación de la función *self-destructing*, eliminando así el *storage privado*. En caso que se requiera, el viejo *smart contract* puede ser migrado a otro subcontrato antes de ser destruido utilizando la herramienta de migración de contratos propia de NEO.

4.8 Waves

Es una plataforma lanzada en 2016 por Sasha Ivanov y su equipo de desarrollo, ideada para cubrir las necesidades que tienen los desarrolladores y empresas que quieren explotar algunas de las propiedades de los sistemas que integran tecnologías de *Blockchain*. Dentro de ellas destacan cuatro: seguridad, auditabilidad, verificabilidad y confiabilidad.

Los protocolos de *Blockchain* presentan algunas limitaciones en cuanto a escalabilidad que se basan principalmente en dos grandes desafíos: producción (*throughput*) y latencia. Actualmente estos protocolos no son lo suficientemente

rápidos ni capaces de incluir más transacciones en sus sistemas, lo que lo hace el principal desafío a tratar de resolver. Existen propuestas para mejorar la escalabilidad de Bitcoin por ejemplo, y se basan principalmente en definir un tamaño de bloque apropiado y en prever el posible crecimiento de los mismos. Sin embargo todas estas propuestas presentan un cuello de botella y no importa cuál es el tamaño del bloque, en Bitcoin se podrían producir entre 3 a 7 bloques por segundo, lo que es significativamente inferior a la cantidad de transacciones que puede manejar un sistema de pagos como Visa Net (más de 65000 mensajes de transacción por segundo según su sitio oficial [VISA2019]). En WAVES la cantidad de transacciones soportadas por bloque se establece entre 0 y 6000, siendo el tamaño máximo del bloque 1 MB.

La plataforma utiliza la criptomoneda WAVES, cuya unidad mínima es WAVELET que vale 1/1000000000 WAVES. Esta plataforma a diferencia de Bitcoin, no mina su criptomoneda, desde el principio se crearon 100 millones de unidades que fueron divididos de la siguiente manera: 1 millón para los usuarios precursores, 1 millón para recompensas a involucrados en la *ICO*, 4 millones a socios estratégicos, 9 millones a los desarrolladores y 85 millones en circulación.

Para la utilización de la plataforma es necesario contar con WAVES ya que cada una de las transacciones que se pueden ejecutar tienen un costo específico. Si bien los costos de las transacciones están definidos según la tabla que se muestra en la Figura 15, cada participante puede especificar el costo que va a pagar por una transacción, siempre y cuando este sea superior a lo establecido. Cuanto mayor sea el costo a pagar más rápido se agregará la transacción.

Transaction type	A minimum transaction fee in WAVES	Comments
Alias transaction	0.001	
Burn transaction	0.001	
Data transaction	0.001 per kilobyte	The value is rounded up to the thousandths
Exchange transaction	0.003	
Invoke script transaction	0.005 + B + C + 0.004 × D	If an invoke script transaction is sent from a smart account, then B = 0.004, otherwise B = 0. An invoke script transaction may have a payment attached (only one payment). So, if the invoke script transaction has a payment attached and this payment is a smart asset, then C = 0.004, otherwise C = 0. In addition to this, the invoke script transaction can invoke a transfer of a number of different assets. D represents the number of transfers that are smart assets
Issue transaction	1 for regular token 0.001 for non-fungible token	
Lease cancel transaction	0.001	
Lease transaction	0.001	
Mass transfer transaction	0.001 + 0.0005 × N	N is the number of transfers inside of the transaction. The value of 0.0005 × N in the formula is rounded up to the thousandths
Reissue transaction	1	
Set asset script transaction	1	
Set script transaction	0.01	
Sponsorship transaction	1	
Transfer transaction	0.001	

Figura 15: *Tabla de costos por tipo de transacción.* [Wav19].

A continuación se describen algunas características que destacan esta plataforma. Waves utiliza una arquitectura en dos capas, en donde se diferencian nodos completos y nodos livianos. Estos nodos completos son aquellos que tienen capacidad para almacenar todo el historial de transacciones, el equivalente a un nodo en Bitcoin, y los nodos livianos son aquellos que mediante algún proceso de

verificación pueden almacenar solamente los datos que le son necesarios. Si bien todos los nodos contienen información que parte desde el mismo bloque inicial, podría decirse que ocurre cierto grado de centralización ya que los nodos livianos deben confiar de los completos. Waves no restringe la posibilidad de ser un nodo completo, quien lo desee y tenga capacidad, puede serlo.

Waves elige basarse en el mecanismo de consenso *LPOS (Leased Proof of Stake)* debido principalmente a dos aspectos; al éxito de la plataforma NXT [Nxt20] (la primera plataforma de las llamadas *Blockchain* de segunda generación, que implementó *Proof of Stake*) y a algunos aspectos teóricos de dicho protocolo. A su vez se implementan algunas mejoras para permitir reducir el tiempo entre transacciones y mejorar el *throughput*. En un mecanismo del tipo *POS,* cada nodo que posee una cierta cantidad de *tokens* tiene una oportunidad (proporcional a la cantidad) de producir un nuevo bloque. En una arquitectura de dos capas, es lógico pensar en manejar el procesamiento de los pagos en los nodos completos, sin embargo al utilizar *POS* todos los nodos cuyo balance sea distinto de cero deben ser elegibles a participar.

Waves implementa un sistema de arrendamiento de *tokens*, en donde los nodos livianos pueden arrendar sus *tokens* a los nodos completos para que estos aumenten la posibilidad de generar un nuevo bloque. En este proceso los *tokens* arrendados no se transfieren a los nodos completos, simplemente quedan “suspendidos” en la cuenta del nodo arrendador. Esto permite mejorar en dos aspectos la seguridad del protocolo: por un lado que hayan más *tokens* participando del consenso hará que sea más difícil que exista un nodo con más del 50% de los *tokens* y por otro lado, que un nodo pueda arrendar *tokens* hará que no tenga que comprometer todos sus *tokens* en el consenso si no que simplemente puede tener un balance mínimo y arrendar el resto. Al reducir la cantidad de nodos que potencialmente pueden producir bloques permite que los tiempos de confirmación sean más rápidos, que haya menor latencia y por ende un mayor rendimiento del sistema.

Cada vez que un nuevo bloque es creado, el nodo creador recibe cierta cantidad de waves, algunos como recompensa por minar el nuevo bloque y otros por los costos de transacción. Esta recompensa es definida y discutida cada 100000 bloques, es decir, cada vez que se crean 100000 bloques empieza un período de votación, en donde todos los nodos completos votan si quieren incrementar, decrementar o mantener la recompensa. Este período de votación dura 10000 bloques y el resultado de los votos, es agregado a *waves.rewards.desired* en cada uno de los bloques generados. Al finalizar el proceso se analizan los 10000 bloques y si más de la mitad de ellos indican incrementar/decrementar se cambia el valor por 0.5 waves, si esto no ocurre se mantiene el valor.

Waves provee una entidad descentralizada de intercambio distribuido (DEX) que permite el intercambio de diferentes activos entre usuarios sin la necesidad de usar una entidad externa y centralizada. Para lograr este propósito los *tokens* deben ser intercambiados en una ubicación pública, así compradores y vendedores pueden hacer sus órdenes. El intercambio en tiempo real se puede producir gracias al componente *Matcher*, entidad que vincula órdenes con transacciones a gran velocidad. Estas velocidades se logran gracias a que no es necesario validar que el nuevo bloque haya sido agregado a la *Blockchain*. Las órdenes son unidas de a pares y enviadas al *Matcher* para que este chequee que los precios fijados son correctos para ambas partes. Una vez chequeado, el *Matcher* genera una transacción de tipo *exchange*, la firma y la envía a los mineros para que estos la validen. Una vez validada, la misma es puesta en la *Blockchain* y los balances de todas las partes son actualizados. Algo que se destaca de este sistema de intercambio, es que los fondos son transferidos recién cuando la transacción queda registrada en la cadena. En la Figura 16 se muestra un diagrama sobre este proceso, en el cual hay dos interesados en comprar y un interesado en vender. Por los montos que se manejan en el ejemplo, se puede ver que el *Matcher* cobra un *fee* proporcional a la cantidad de unidades intercambiadas.

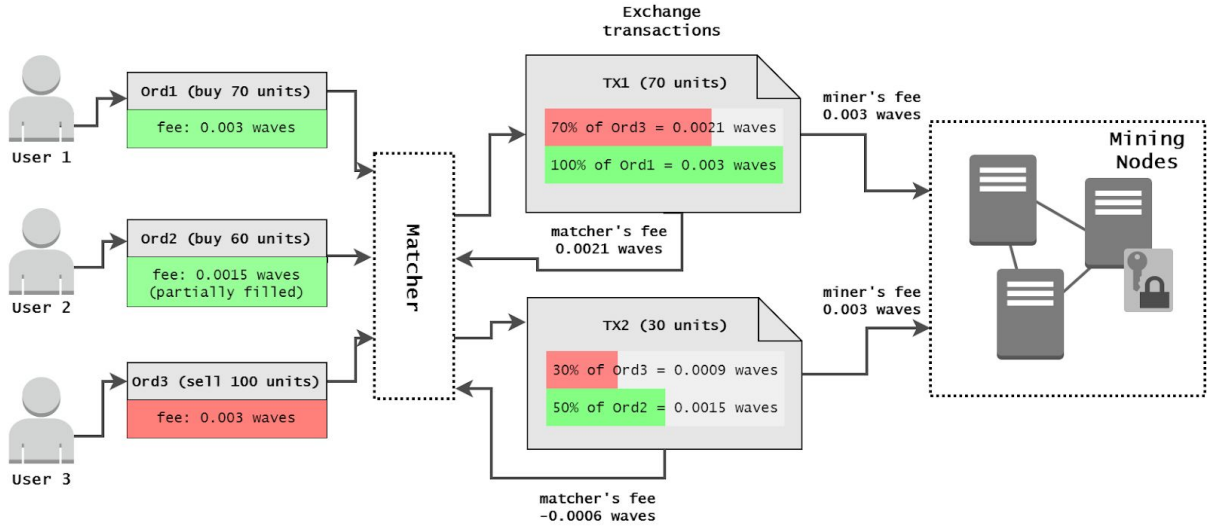

Figura 16: *Intercambio distribuido (DEX)*. [Wav16]

Smart Contracts

A diferencia de otras plataformas el concepto de "*smart*" siempre está asociado a una cuenta o un recurso, es decir, no existe una entidad que separa a uno de estos dos, algo que sí sucede en Ethereum por ejemplo. Al asociar un *script*, escrito en el lenguaje Ride [BS18], a cualquiera de estos dos componentes, el usuario puede modificar el comportamiento por defecto de los mismos. Esto permite poder agregar lógica que valide o chequee ciertas condiciones que sean de interés para el desarrollador, por ejemplo se puede chequear que una transacción ocurra solo si varios usuarios están firmando la misma. Como se representa en la Figura 17, en el caso de las cuentas inteligentes, el *script* se ejecuta cada vez que la cuenta quiera enviar una transacción, no así en los recursos inteligentes, donde el *script* se ejecuta cada vez que un usuario quiere interactuar con un recurso.

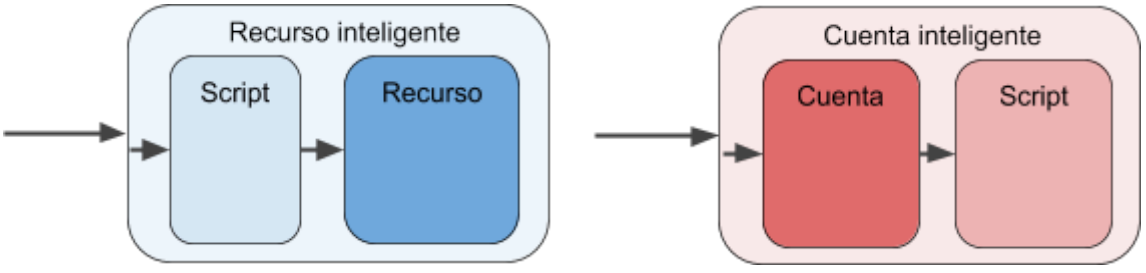

Figura 17: *Smart Account y Smart Asset*. [Wav16]

Otra diferencia frente al resto de las plataformas es que utiliza un lenguaje desarrollado por ellos mismos para implementar los distintos *smart contracts*. El lenguaje utilizado se llama Ride y tiene algunas particularidades frente a otros: es un lenguaje funcional muy similar a F#, es fuertemente tipado, no tiene loops, ni recursión ni expresiones del estilo *go-to* y no es Turing Completo.

4.9 RSK

RSK [Dem15] surge como proyecto en 2015 en Argentina y lanza su *Blockchain* en producción en enero de 2018. Los fundadores justifican la creación de la plataforma, principalmente para resolver la carencia de implementación de *smart contracts* en la red de Bitcoin. RSK es una Bitcoin-*sidechain*, es decir, RSK tiene su propia *Blockchain* pero no su propia criptomoneda. Tiene una comunicación bidireccional con Bitcoin, cuando los bitcoins son transferidos a la *Blockchain* de RSK, "se transforman" en SmartBitcoins, este *token* no es transferido en realidad si no que los bitcoins quedan bloqueados del lado de Bitcoin y se habilita su uso del lado de RSK, pueden ser devueltos a la red original en cualquier momento y no hay costos adicionales más que los específicos de costo de transacción propia de cada *Blockchain*. Con esta particularidad, RSK complementa a Bitcoin, teniendo transacciones más rápidas y logrando mayor escalabilidad. Un pago de RSK requiere una quinta parte del tamaño de un pago estándar de Bitcoin. Usando el protocolo de compresión de transacciones (*LTCP*), el tamaño de la transacción puede reducirse a 1/50 del tamaño de una transacción de Bitcoin. Esto lleva un aumento sustancial en la capacidad de volumen de transacciones a cambio de un menor nivel de privacidad en la *Blockchain*. Tiene un tiempo estimado de confirmación de bloque de 30 segundos y se generan entre 10 y 20 transacciones por segundo dependiendo del origen de la misma.

RSK es un mix de QixCoin[Qix13] y Ethereum, QixCoin fue una criptomoneda creada en 2013 por los mismos fundadores de RSK. QixCoin introducía el concepto de pago por ejecución, similar al Gas. Además mantiene algunos conceptos inherentes a Ethereum como el formato de las cuentas, la *virtual machine* que utiliza y la interfaz con web3. Por esto último, Ethereum y RSK son altamente compatibles.

RSK utiliza el mecanismo de consenso *Proof-of-Work* con un adicional, soporta *merge-mining*, esto es que el trabajo que realizan los mineros para resolver el acertijo y ganar bitcoins en Bitcoin, también sirve para el minado sobre la *Blockchain* de RSK.

Algunas particularidades de esta plataforma son: el método de *merge-mining* con Bitcoin, la *two-way peg sidechain* y la protección del *selfish-mining* usando el protocolo DECOR+[Dem15].

Merge-mining es un mecanismo que permite a los mineros de Bitcoin minar otras criptomonedas simultáneamente con un costo marginal casi nulo. Se utiliza la misma infraestructura, *software* y *hardware*. Para RSK se incluye una referencia al bloque de esta *Blockchain*, cada vez que un minero encuentra una solución, la compara con las dificultades (del puzzle) de ambas redes y ocurre una de esas tres casuísticas:

1) la solución satisface la dificultad de la red Bitcoin. Por lo tanto, se envía el bloque a la red. La referencia de minería de RSK será incluida e ignorada por la red Bitcoin. Dado que la dificultad de la red de RSK es menor que Bitcoin, esta solución también funcionará para RSK y también puede enviarse a la red.
2) La solución no satisface la dificultad de la red Bitcoin pero sí satisface la dificultad de la red RSK. Como consecuencia, la solución se enviará a la red RSK y no a la red Bitcoin.
3) La solución solo satisface la dificultad del grupo (que es muchas veces menor que la dificultad de la red Bitcoin o RSK) y no se envía a ninguna red.

Para modelar la *two-way peg sidechain*, es decir, el intercambio de criptomoneda entre Bitcoin y RSK, esta última tiene un *smart contract* llamado *Bridge*, que es utilizado por la Federación de RSK para hacer transferencia de criptomoneda. Este contrato podría ser utilizado por cualquier persona, por lo que garantiza la descentralización del proceso. Para transferir de RSK hacia Bitcoin, primero se tienen que enviar los SBTC a una dirección especial del *Bridge*, como Bitcoin no puede acceder ni verificar esta transacción, la Federación de RSK interviene para

liberar bitcoins en Bitcoin. Se utiliza un modelo de seguridad por *hardware* (*HSM*) que impide que los miembros de la Federación accedan a la información de las claves privadas, por lo que es imposible que exista el robo de moneda. Cuando se colecta la cantidad suficiente de firmas en la transacción, los BTC son envían a las direcciones correspondientes que iniciaron el intercambio.

El proceso de intercambio entre BTC y SBTC demora alrededor de 15 horas (100 confirmaciones de bloque en Bitcoin) como precaución para evitar la pérdida de fondos debido a una reorganización de la cadena. Desde 2019, adicionalmente se requiere pertenecer a una lista blanca de usuarios, el flujo de RSK a Bitcoin no tiene esta restricción. Este último requisito generó que algunos *exchanges* (intermediarios de compra y venta de moneda), tengan a la venta SBTC para los usuarios que necesiten realizar el cambio y no puedan esperar las 15 horas de confirmación. La existencia de la Federación es justificada debido a que Bitcoin no soporta la ejecución de *smart contract* para realizar la liberación del dinero. [Dem16]

En la red de Bitcoin, cuando dos o más mineros han resuelto bloques al mismo nivel, surge un conflicto de intereses. Cada minero competidor quiere que su bloque sea seleccionado por los mineros restantes. Los mineros y usuarios honestos prefieren que se elija el mismo bloque, ya que reduce la probabilidad de reversión del bloqueo. DECOR+ establece los incentivos económicos correctos para una elección convergente, sin requerir mayor interacción entre los mineros. El conflicto se resuelve para que: se maximicen las ganancias de los mineros participantes cuando la recompensa es mayor al promedio y para que se reduzca el poder de los mineros para censurar otros bloques o transacciones cuando la recompensa es cercana al promedio.

Smart contracts

Como se mencionó anteriormente, los *smart contracts* en RSK se ejecutan dentro de la máquina virtual que se basa en Ethereum, la RVM y utilizan el mismo lenguaje de desarrollo, Solidity. En la sección **4.1** se puede consultar un ejemplo.

4.10 Quorum

Es una plataforma desarrollada por la compañía JP Morgan a partir de un *soft fork* de Ethereum pensada para el sector empresarial. Es decir, es una plataforma que se basa casi totalmente en Ethereum pero que agrega algunas funcionalidades de privacidad y protección de información, transacciones y *smart contracts* privados, que la hacen más adecuada para ser utilizada entre empresas. Es creada a partir de Ethereum ya que consideran importante los siguientes aspectos de esta; es la primera plataforma de *smart contracts* en estar en producción, tiene uno de los ecosistemas de desarrolladores más grandes y la red pública de Ethereum protege más de $1B Ether.

Maneja permisos para la red y los nodos, es decir, la red no es abierta para que cualquiera pueda participar, solo algunos nodos lo podrán hacer

Establece privacidad en transacciones y contratos, es decir, dejando de lado su naturaleza que maneja permisos, Quorum maneja el concepto de transacciones y contratos privados. Las transacciones de Quorum pueden tener una lista de claves públicas que identifican las entidades involucradas y son quienes tendrán acceso a dicha transacción, en caso de crear una transacción privada. Los datos que deban ser privados son encriptados y colocados dentro del *payload* de una transacción habitual de Ethereum que luego es enviada a la red Quorum.

Los contratos privados son aquellos creados a partir de una transacción privada y que cuentan con su propia estructura de almacenamiento. Por este motivo un contrato privado no puede ser creado con una transacción pública ya que el estado del nuevo contrato será guardado dentro de la estructura destinada a contratos públicos.

La performance genera cientos de transacciones por segundo. Esto se logra principalmente por la simplicidad que tienen sus mecanismos de consenso, ya que al trabajar sobre una red permisionada pueden omitir ciertos controles. En [BSK+18] se puede encontrar un estudio donde se analiza la performance de Quorum utilizando sus dos mecanismos de consenso, que demuestra que puede escalar

más que otras plataformas.

Quorum no tiene criptomoneda propia si no que utiliza Ether. Al ser una plataforma desarrollada como un *fork* de Ethereum que utiliza la misma tecnología base no es necesario crear una criptomoneda propia.

A diferencia de otras plataformas, presenta dos mecanismos de consenso que pueden ser seleccionados a la hora de comenzar un nodo; *Raft-based* e *Istanbul Byzantine Fault Tolerance (IBFT).*

Tiene una arquitectura basada en dos capas: todos los nodos son clientes basados en go-Ethereum [GET20] con una capa superior que modifica los mecanismos de consenso utilizados y agrega las funcionalidades necesarias para ejecutar e intercambiar transacciones privadas. Dentro de esta capa se encuentran los siguientes componentes:

- manejador de transacciones (*Transaction Manager*), que brinda acceso a transacciones encriptadas, maneja la información local y permite la comunicación con otros manejadores de transacciones.
- *crypto enclave*, es quien maneja las claves privadas y se encarga de encriptar/desencriptar la información de las transacciones privadas.
- QuorumChain, es el protocolo que permite, utilizando el *core* de Ethereum, verificar y propagar los votos dentro de la red
- Manejador de red (*Network Manager)*, es el encargado de controlar el acceso a la red, permitiendo la creación de una red permisionada.

Como se muestra en la Figura 18, Quorum utiliza una sola *Blockchain* para almacenar tanto los contratos privados como los públicos, siendo los nodos de la red los encargados de validar las distintas transacciones, utilizando algunos de los componentes mencionados anteriormente. Tanto los *smart contracts* que sean de tipo privados como las transacciones son expuestas solo a las entidades involucradas.

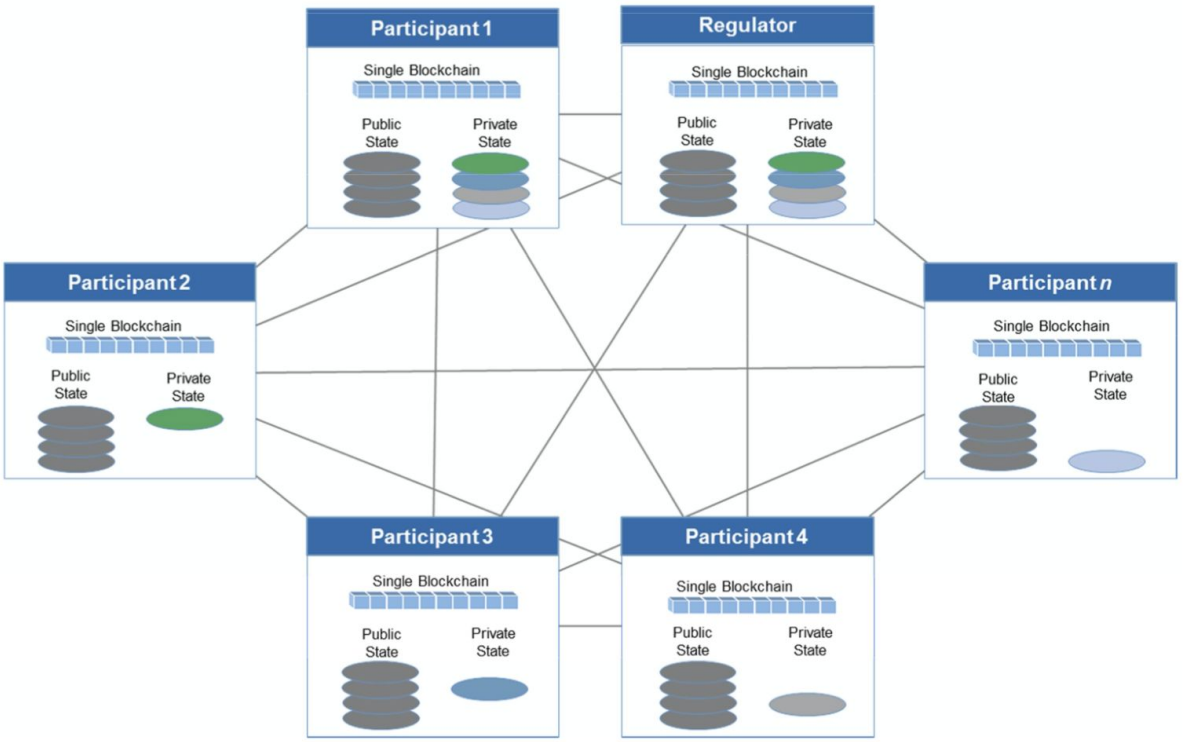

Figura 18: *Red Quorum*. [Mor17]

Smart contracts

Como se mencionó anteriormente, al ser una plataforma desarrollada como un *fork* de Ethereum, utiliza la misma tecnología base, los *smart contracts* dentro de Quorum son desarrollados utilizando Solidity y se ejecutan sobre la misma máquina virtual (*EVM)*. Por este motivo no se estudian las características dentro de la plataforma, ya que las mismas se encuentran detalladas en la sección **4.1**.

4.11 Hyperledger Fabric

Es una plataforma de *Blockchain* privada, *open source*, orientada al uso empresarial que surge en 2015 como un proyecto de la Fundación de Linux y es mantenida por una gran comunidad de desarrolladores (cerca de 35 organizaciones y 200 programadores). Tiene una arquitectura modular y configurable. Fue concebida para que se adapte a las necesidades de cada organización debido a que las *Blockchain* públicas no son elegidas por las empresas que necesitan manejar otro nivel de privacidad y confidencialidad en el manejo de las transacciones. Además los participantes de la red deben ser identificables, se deja de lado el anonimato. No

tiene definida una criptomoneda y no se asigna un costo por transacción. Es posible personalizar el mecanismo de consenso que se quiere implementar, lo que hace que la plataforma pueda alcanzar un alto rendimiento en las transacciones, es decir, mayor cantidad de confirmación de transacciones en menor tiempo.

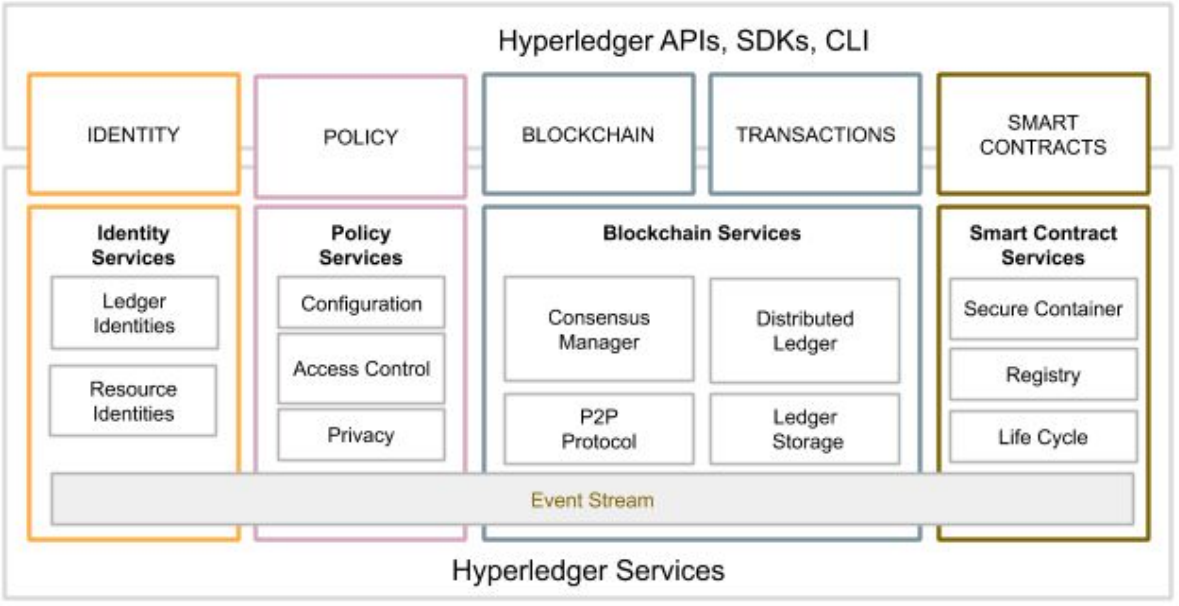

Figura 19: *Arquitectura de Hyperledger Fabric.* [Hyp16]

La Figura 19 muestra un diseño lógico de los diferentes módulos que podemos encontrar en esta plataforma. Se detallan a continuación los más relevantes para la ejecución de los *smart contracts*.

Dentro de los servicios de Identidad (*Identity Services*) se gestionan y se emiten los certificados que autorizan a cada miembro de la organización a ser participante de la red o el consorcio establecido. Los nodos que participan en las organizaciones son clasificados en tres tipos. Los *orderers* son responsables de la distribución de los bloques, mantienen la consistencia entre los diferentes nodos de la red, ejecutan el mecanismo de consenso y notifican a los *anchor peers* de los bloques que le son inherentes. Los *peers* mantienen los datos del *ledger* sincronizados en la red, cada uno tiene su propia copia de los siguientes componentes: el *Transaction Log* (*Distributed Ledger* en la Figura 19) que guarda un registro de todas las transacciones que se producen en el canal y el *State DB* (*Ledger Storage* en la Figura 19) que guarda un listado del estado de los *assets* en cada momento. Cada

organización tendrá tantos *peers* como considere necesario y se pueden clasificar en cuatro roles que son representados en la Figura 20. Los *anchor peer* son los intermediarios con otros elementos fuera de la organización. Los *leader peer* son los encargados de recibir los bloques del *orderer* y de distribuirlos al resto de *peers* de su organización. Los *regular peer* son los nodos más básicos, mantienen el *ledger* actualizado y se sincronizan con el resto de *peers*. Finalmente los *endorsing peers* son los encargados de recibir las transacciones de los clientes o aplicaciones, se encargan de simular la ejecución de la transacción pero sin dejar registro de la misma en el *StateDB*. En caso de que la simulación se haya producido satisfactoriamente, entonces se manda al *ordering service* (donde están los *orderer peers*) para que se encargue de reunirlas y transmitirlas al resto de la red en un bloque. Esta simulación se produce para evitar un ataque intencionado de un cliente contra la red y para evitar errores de configuración. El último tipo de nodo consiste en los *clients* que son responsables de enviar las transacciones. Su función principal es la de generar una solicitud de transacción contra un *endorsing peer*.

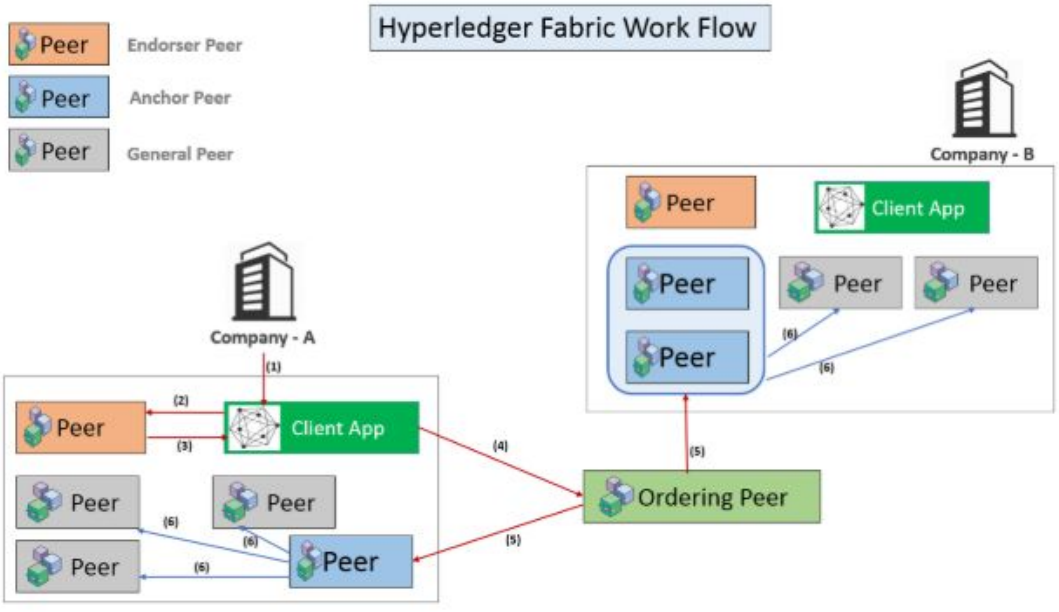

Figura 20:*Intercambio entre nodos.* [Coi19]

En la documentación oficial se indica que los términos *smart contract* y *chaincode* se pueden usar de forma indistinta pero tienen algunas sutilezas. Un *smart contract* define la lógica de la transacción que controla el ciclo de vida de un objeto contenido en el estado global mientras que un chaincode almacena uno o varios smart contract para ser desplegados en la Blockchain. Cada *chaincode* lleva asociado un *endorsement policy*. Estas son políticas que se definen a la hora de desplegar el *chaincode* en el canal. Las aplicaciones deberán conocer, las direcciones de los *endorsing peers* a los que deben enviar estas solicitudes de transacciones. Cuando el cliente recibe la respuesta de todos los *endorsing peers* o tiene el número suficiente de respuestas válidas se las envía al *orderer* para que siga el flujo de transacción.

Smart contracts

En Hyperledger Fabric, los *smart contracts* son instalados en un *endorsement peer* de la organización y son invocados por una aplicación cliente externa a la *Blockchain* mediante el envío de una propuesta de transacción al *endorsement* peer, que es tomada como entrada para la ejecución del *smart contract* y una vez finalizado, se le enviará la respuesta firmada, que interactúa sólo con el *StateDB* del *ledger*.

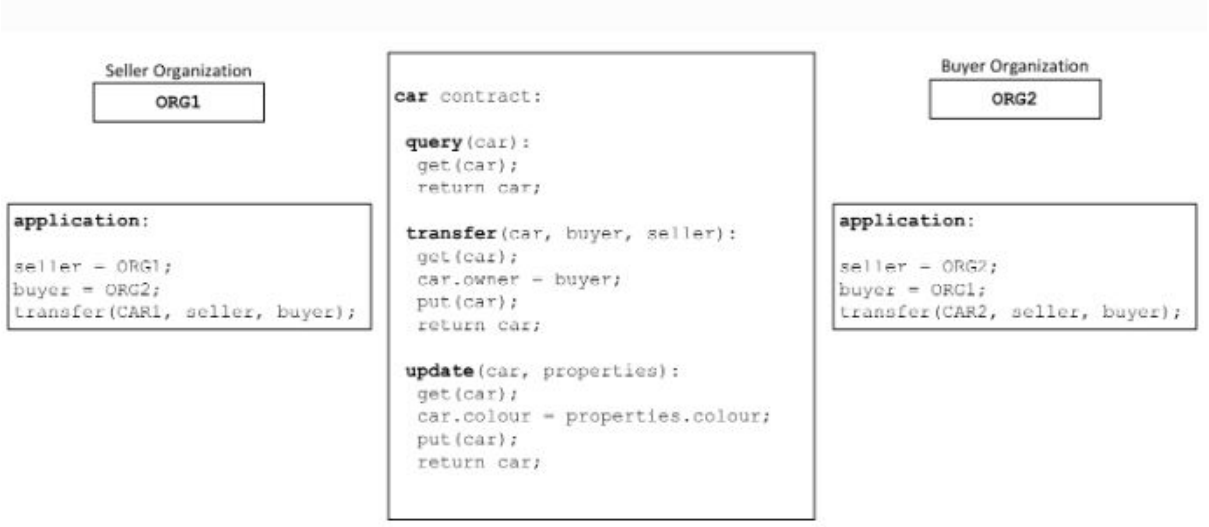

Figura 21: *Ejemplo de smart contract*.[Hyp19].

En la Figura 21, se detalla cómo dos organizaciones ORG1 y ORG2 definieron un *smart contract* para consultar, transferir y actualizar autos.

Los *smart contracts* son definidos dentro de un *chaincode.* Muchos contratos pueden ser definidos dentro del mismo *chaincode* y una vez que el mismo es puesto en producción, todos los contratos quedan disponibles para ser ejecutados por las aplicaciones. Además, para cada *chaincode* se configura una *endorsement policy* que rige para todos los contratos que son contenidos por ella. En la misma, se configura, cuantos *endorsement peers* deben firmar una transacción para ser declarada como válida. Un ejemplo de *endorsement policy* es definir que al menos 3 de cuatro organizaciones participantes de una *Blockchain*, deban firmar la transacción para que sea válida. Todas las transacciones, independiente de si son válidas o no, se almacenan en el *ledger* distribuido, pero sólo las transacciones válidas manipulan el *StateDB* de los objetos.

Los *smart contracts* tienen funciones: *PUT, GET y DELETE* que modifican el estado global de objetos del *ledger* y también pueden consultar el registro de transacciones. Mientras que las transacciones de lectura, actualización, borrado y creación, modifican los objetos del *StateDB*, la *Blockchain* mantiene un registro inmutable de estos cambios. Es importante aclarar que la modificación de los objetos del *StateDB* no se realiza cuando se ejecutan los contratos, si no que es realizada, luego de que los ordered nodes colocan la transacción en un bloque y es enviada a la Blockchain. En el ejemplo que se muestra en la Figura 21, *car* representa el *smart contract*, que cambia mediante la operación *transfer* la propiedad propiedad *car.owner,* es decir que actualiza quién es dueño del vehículo. Se muestra el uso de algunas operaciones como PUT y GET. También se muestra la endorsement policy que indica que ORG1 y ORG2 tienen que firmar la transacción para que se la considere válida.

Otro aspecto a destacar en esta plataforma son los channels que permiten que las organizaciones participen de forma simultánea en múltiples Blockchain separadas. Es una forma eficiente de compartir infraestructura sin comprometer la privacidad de la comunicación y la información. Los smart contracts pueden invocar a otros smart

contract dentro del mismo channel o en diferentes, lo que les permite consultar y modificar StateDB de otros channels.

4.12 Corda

La plataforma Corda [Hea16] surge en 2016 como producto de R3, un consorcio formado por empresas interesadas en la tecnología de *Blockchain*. El objetivo es crear una solución en la que empresas e individuos puedan ejecutar *smart contracts* de forma directa y respetando la privacidad de las partes interesadas, es decir que el *ledger* no es público. Funciona como una plataforma de *ledgers* distribuidos (*DLT*) y si bien, tiene las funciones de cualquier plataforma de *Blockchain*, no es una plataforma de *Blockchain* propiamente dicha, ya que no encapsula las transacciones en bloques y las concatena, si no que el *ledger* directamente las almacena. Una de las razones por la que las transacciones son almacenadas en bloques en otras plataformas, es para que sea eficiente la distribución de la información a lo largo de la red, además que el uso de *Proof-of-Work* requiere que los nodos tengan uniformidad en cuanto a la información de cuál es el último bloque y generar espacios de tiempo para que la red se actualice y se pueda crear el siguiente bloque. En Corda, los nodos no mantienen una *Blockchain* si no, que almacenan un copia del ledger que comparten con otros nodos. Es un sistema *open source* privado usado principalmente por el sector financiero.

El *state object* es un objeto inmutable que representa un hecho conocido por uno o más nodos en un momento específico de tiempo y puede representar hechos de cualquier tipo, como: stock, préstamos, información de identidad, etc..es un objeto inmutable que representa un hecho conocido por uno o más nodos en un momento específico de tiempo y puede representar hechos de cualquier tipo, como: stock, préstamos, información de identidad, etc... Para actualizarlo es necesario realizar una transacción que consumirá el *state object* anterior y dejará un nuevo *state object* con la información actualizada.

Las redes Corda utilizan mensajes punto a punto en lugar de una transmisión global. Esto significa que la coordinación de una actualización del *ledger* requiere que los

participantes de la red especifiquen exactamente qué información debe enviarse, a qué contrapartes y en qué orden. El framework de flujo automatiza esta secuencia de pasos y es ejecutado únicamente por los nodos involucrados y no por toda la red.

Para determinar que una transacción propuesta genera un actualización válida del ledger, se tienen que alcanzar dos tipos de consensos: de validez, cada nodo que tiene que firmar, verifica que la transacción es válida antes de firmala; y de unicidad, que es revisado únicamente por el servicio de notarios.

El consenso de validez es el proceso de verificar que las siguientes condiciones se cumplan, tanto para la transacción propuesta como para cada transacción en la cadena de transacciones que generó las entradas a la transacción propuesta: la transacción es aceptada por los contratos de cada *state object* de entrada y salida; la transacción tiene todas las firmas requeridas.

El consenso de unicidad es el revisa que ninguna de las entradas a una transacción propuesta ya se haya consumido en otra transacción. Si una o más de las entradas ya se han consumido en otra transacción se conoce como *double spending* y la propuesta de transacción se considera inválida. El grupo de notarios hará una de las siguientes dos acciones: firmará la transacción si verifica que no existen otras transacciones que consumen cualquiera de los estados de entrada de la transacción propuesta o rechazará la transacción y la marcará como un intento de *double spending*.

Corda tiene *pluggable* mecanismos de consenso, lo que permite que los grupos de notarios elijan un algoritmo de consenso basado en sus requerimientos en términos de privacidad, escalabilidad, compatibilidad del sistema legal y agilidad algorítmica. Un grupo notario puede ser un solo nodo, varios nodos de confianza mutua o varios nodos de desconfianza mutua. Dentro de las características del grupo de notarios, se puede puede optar por ejecutar un algoritmo de alta velocidad y que requiere alta confianza como RAFT, o un algoritmo de baja velocidad y que requiera bajo nivel de confianza entre los nodos como BFT o cualquier otro algoritmo de consenso que se elija.

Las transacciones en Corda se procesan haciendo que cada participante en la transacción ejecute el mismo código de manera determinista para verificar las actualizaciones propuestas al *ledger* y son compartidas únicamente a los interesados.

Corda no tiene una una moneda nativa como tienen otras plataformas, pero es posible modelar *tokens* y *assets* utilizando *state objects*, contratos y flujos.

Una transacción es válida si está firmada digitalmente por todos los participantes requeridos y es contractualmente válida, se tienen que cumplir ambas condiciones para que sea impactada en el *ledger*. La validez contractual se define de la siguiente forma: cada estado de transacción especifica el tipo de contrato, el contrato toma una transacción como entrada y establece si la transacción se considera válida en base a las reglas del contrato. Una transacción es válida si el contrato de cada estado de entrada y cada estado de salida se considera válido.

Corda se destaca en varios aspectos: las partes tienen garantía sobre la identidad de los participantes en la red; las únicas partes que tienen acceso a los detalles de una transacción son aquellos que participan en la transacción y aquellos que necesitan asegurarse de la procedencia de la misma; las ofertas registradas por el *ledger* pueden ser, por contrato, aceptadas como pruebas admisibles y legalmente vinculantes por todas las partes en cualquier disputa.

La arquitectura de esta plataforma sostiene la escalabilidad del sistema, la red se ampliará para admitir miles de millones de transacciones diarias; longevidad, debido a que diferentes versiones de Corda podrán coexistir en la misma red y las aplicaciones continuarán ejecutándose en versiones posteriores; interoperabilidad, la plataforma está diseñada para permitir que múltiples aplicaciones coexistan e interoperen en la misma red; se incluye un conjunto estandarizado de interfaces para contratos para maximizar la interoperabilidad de una amplia gama de proveedores.

Corda permite que se especifiquen límites de tiempo arbitrariamente precisos en las transacciones (que deben ser confirmadas por un *timestamper* confiable) en lugar

de depender del momento en que se extrae un bloque. Esto es importante dado que muchos tipos de *smart contracts* requieren precisión en el tiempo.

Smart contracts

Los nodos verifican el contenido de una transacción antes de firmarla. Un nodo no tiene la obligación de firmar una transacción solo porque es válida contractualmente. A veces, la validez de la transacción dependerá de información externa, como un tipo de cambio de monedas. En estos casos, se requiere de la consulta al oráculo.

Existen dos formas de gestionar las actualizaciones de un smart contract en Corda. De forma implícita, que significa autorizar previamente múltiples implementaciones del contrato con anticipación, utilizando restricciones; y de forma explícita, al crear una transacción de actualización de contrato especial y lograr que todos los participantes de un estado lo firmen utilizando los flujos de actualización de contrato.

Corda soporta *smart contracts*, que coinciden con la definición de Clack, Bakshi, Braine[CBB17]; *smart contract* es un acuerdo cuya ejecución se puede automatizar mediante un código de computadora que trabaja con entrada y control humano, y cuyos derechos y obligaciones, tal como se expresan en prosa legal, son legalmente exigibles: la lógica de *smart contracts* especifica restricciones que aseguran que las transiciones de estado sean válidas de acuerdo con las reglas acordadas previamente, descritas en el código del contrato como parte de CorDapps; provee servicios de singularidad y de *timestamping* conocidos como grupos de notarios para ordenar transacciones temporalmente y eliminar conflictos.

Los nodos descargarán y ejecutarán contratos dentro de un *sandbox*. La máquina virtual que se ha seleccionado para la ejecución y validación de *smart contracts* es la máquina virtual de Java (JVM), ya que cuenta con una gran cantidad de bibliotecas existentes, y la reutilización de un estándar de la industria hace que sea más fácil para los usuarios reutilizar su código existente dentro de los smart contracts.

La JVM que se utiliza es modificada para que la ejecución de los contratos sea determinista. Para garantizar que el contrato sea completamente determinista se

utiliza el sandbox que realiza un análisis estático del *bytecode* cargado y permite auditar el comportamiento del mismo. El análisis estático de *bytecode* realiza las siguientes verificaciones (entre otras): invocaciones dinámicas, métodos nativos, *breakpoints*, versiones no soportadas de la API y excepciones como *ThreadDeath* y *Threshold Violation*. Por otro lado se utiliza una estrategia de instrumentación de costos que está diseñada para garantizar que se terminen los bucles infinitos y que si el costo de verificar una transacción se vuelve inesperadamente grande (por ejemplo, contiene algoritmos con complejidad exponencial en el tamaño de la transacción), todos los nodos acuerdan exactamente cuándo abandonar. Esto no pretende ser una protección contra ataques de denegación de servicio, pero en el caso de encontrar nodos que propician transacciones de este tipo, el mismo resulta bloqueado en la red. Otra necesidad consiste en restringir el uso de la memoria. El *sandbox* impone una cuota en *bytes* asignados para controlar este punto.

4.13 Análisis comparativo

En esta sección se describe el análisis obtenido en base a la investigación realizada. En la Figura 22 se detalla un resumen con las plataformas y los criterios definidos.

Plataforma	Mecanismo de Concenso	Turing complete	Lenguajes	Tipo de acceso	Oráculo	Modificación de contratos
Ethereum	PoW	SI	Solidity, LLL, Serpent	Público	SI	SI
EOS	DPoS	SI	C++	Público	SI	NO
TRON	DPoS	SI	Solidity	Público	SI	SI
Stellar	SCP	NO	Javascript, Java, Go	Permisionada	NO	NO
NEO	dBFT	SI	C#, Python, Java, JavaScript, Go	Público	SI	SI
Ethereum Classic	PoW	SI	Solidity, Vyper	Público	SI	SI
Waves	LPoS	NO	RIDE	Público	SI	SI
RSK	PoW (DECOR+)	SI	Solidity	Pública	SI	SI
Quorum	RAFT, IBFT	SI	Solidity	Permisionada	SI	SI
Hyperledger Fabric	Pluggable (estandarizado)	SI	Go, NodeJS, Java	Privada	SI	NO
Corda	Pluggable (estandarizado)	SI	Java, Kotlin	Privada	SI	SI

Figura 22: *Cuadro comparativo.*

Mecanismos de consenso

El mecanismos de consenso *POW* es comúnmente utilizado por las plataformas de primera generación de *Blockchain* ya que garantiza la consistencia de un sistema descentralizado de forma segura, logrando que sea usado por todos los participantes. En la investigación conducida, se concluye que este mecanismo, a pesar de los problemas que tiene (escalabilidad y uso de recursos eléctricos) es el más seguro para plataformas públicas, es decir en sistemas donde todos los nodos participan activamente de la validación del *ledger* de forma anónima. A medida que los sistemas acotan la cantidad de usuarios validadores de la red y/o que se añaden otros supuestos al sistema, se viabiliza la implementación de nuevos mecanismos. En la búsqueda de generar confirmación de transacciones en menor tiempo es que surge *POS* y *dPOS*, ambos mecanismos funcionan de forma más eficiente en el consumo de recursos, es decir que son computacionalmente escalables, debido a que no se necesita el poder de cómputo para el cálculo de la prueba criptográfica y acotan la cantidad de usuarios participantes, el primero eligiendo entre los nodos con mayor cantidad de tokens almacenados y el segundo designando una cantidad fija de participantes aleatorios.

Una de las características a considerar para elegir de forma adecuada el mecanismo de consenso por la plataforma de *Blockchain*, es el tipo de acceso que tiene la plataforma. *POW* funciona muy bien para la participación pública de los nodos. Los mecanismos *BFT* requieren de una selección de nodos conocidos y confiables para su participación. En cambio *POS* puede ser utilizado en ambos tipos de sistemas respetando algunos supuestos.

El tipo de mecanismo usado también condiciona el hecho de que se puedan generar forks en la *Blockchain,* es decir que la decisión de agregar el bloque en la cadena sea no determinista. En *POW* múltiples bloques se minan al mismo tiempo, lo cual conlleva a que dependiendo de la latencia de red, exista el riesgo de que se forme una bifurcación. En *POS* los fork pueden coexistir por pequeños períodos de tiempo si los validadores, votan en paralelo múltiples cadenas. Adicionalmente el mecanismo puede imponer penalidades adicionales para que esto no ocurra. En

cambio en los *BFT* se determina de forma inmediata el consenso en el próximo bloque, por lo que es completamente determinista.

Alcanzar un acuerdo de forma determinista afecta directamente el ratio de las transacciones, y por ende la escalabilidad de la plataforma. *POW* tiene un ratio que se calcula de forma probabilística, de acuerdo a la dificultad en resolver el puzzle criptográfico. Este modelo tiene alta latencia y bajo ratio de transacciones. *BFT* y *POS* pueden alcanzar el consenso en menos tiempo, por ende tienen un mayor ratio de transacciones. Otro punto relacionado a la escalabilidad, es como se comportan los mecanismos cuando la cantidad de nodos participantes en la red se incrementa. Excepto en pBFT que tiene que mantener acotada la cantidad de nodos verificadores, para no generar un desborde de mensajes de confirmación, todos los mecanismos son escalables en este sentido.

Otra variable que se suma es el grado de confianza de los nodos en la plataforma. *POW* y *POS* se pueden usar en sistemas donde los nodos son desconocidos, *BFT*, necesita asegurar cierto nivel de confianza para efectivamente llegar al consenso, por eso es que plataformas permisionadas y privadas pueden utilizar estos mecanismos.

Finalmente en la descripción de las plataformas se menciona si la plataforma utiliza un *token*. Las plataformas que tienen mecanismos basados en *POW* y *POS* necesitan de la existencia del *token* para su funcionamiento.

Turing complete

Del estudio realizado se puede concluir que la mayoría de las plataformas que implementan *smart contracts* utilizan lenguajes que son Turing completos (nueve de las once plataformas estudiadas). En todos los casos que utilizan lenguajes de este tipo se justifica su utilización en que permiten implementar cualquier tipo de contrato sin importar el fin con que se vaya a utilizar (financieros, intercambio de bienes o servicios, aplicación distribuida). Una particularidad que implementan todas las plataformas que tiene esta propiedad y no son privadas o permisionadas, es la utilización de un *token* como forma de pago por cómputo de procesamiento. Es decir, así como en Ethereum se implementa el concepto de Gas, otras plataformas

implementan la misma idea con distinto nombre, con el objetivo de poder limitar el tiempo de ejecución de un contrato. Esto garantiza que sin importar cómo esté programado el *smart contract*, su ejecución siempre va a terminar (en el peor de los casos agotando todos los recursos que le hayan sido provistos). Para aquellas plataformas que son Turing completas y de tipo privadas o permisionadas, no se requiere la evaluación del costo computacional ya que existe una validación previa entre algunos de los nodos de la red o es el propio motor quien restringe su ejecución.

Finalmente existen plataformas que no cumplen con esta característica, no utilizan lenguajes Turing completos para la implementación de *smart contract* basándose en que los contratos son programados con un propósito específico y que no es necesario que todos los problemas puedan ser modelados. Por ende, utilizan lenguajes de dominio específico, que por un lado restringen la variedad de vulnerabilidades posibles, como ataques o errores por loops infinitos, y por otro permiten determinar con exactitud cuál es la complejidad del *smart contract* a ejecutar (costo de ejecución).

Lenguajes soportados

Para el estudio de los lenguajes que utilizan las distintas plataformas se generan los siguientes dos gráficos que reflejan claramente las grandes diferencias que existen entre ellas.

Figura 23: *lenguajes clasificados por propósito* Figura 24: *lenguajes clasificados por paradigma*

En relación al propósito del lenguaje, como se observa en la Figura 23, la mayoría de las plataformas utilizan un lenguaje diseñado específicamente para los *smart contracts*. Es decir, en lugar de utilizar un lenguaje de propósito general como por ejemplo: Java, Python, C++, JavaScript o C#, se crea o utiliza un lenguaje desarrollado pura y exclusivamente con este fin. Esto puede verse, en primera instancia, como una mejora de seguridad, ya que contar con un lenguaje con estas características permite tener un conjunto de operaciones acotado que sólo permita implementar ciertos escenarios específicos. Sin embargo, se puede ver que al utilizar un lenguaje que tiene muchos años en uso, que se ha utilizado en una enorme cantidad de proyectos y cuya comunidad es muy grande, garantiza que el lenguaje ha sido probado, que se conocen sus vulnerabilidades y que no presenta grandes riesgos de seguridad y/o performance.

Otro de los argumentos que da el propio creador de Ethereum, Vitalik Buterin, es que al generar su propio lenguaje y compilador, se puede optimizar el tamaño del código generado y así minimizar el espacio requerido para almacenar el *smart contract* en la *Blockchain*, algo que es muy costoso.

Si se analiza el paradigma que siguen los lenguajes que utilizan las plataformas estudiadas se puede ver que en su mayoría optan por un lenguaje orientado a objetos mientras que en algunos casos se opta por un lenguaje de tipo funcional. En el caso de las plataformas seleccionadas, como se observa en la Figura 24, diez plataformas utilizan un lenguaje orientado a objetos y solamente una un lenguaje funcional. Al igual de lo que sucede en otras discusiones sobre ventajas y desventajas de estos lenguajes, no se puede concluir si uno es mejor al otro, simplemente se analiza cuál es la característica por la que se los elige.

En el caso de los lenguajes funcionales queda claro que se los elige ya que brindan la posibilidad de conocer exactamente cómo se va a ejecutar una función sin importar cuál sea su estado. Esto garantiza cuál va a ser el costo computacional asociado a la ejecución de un contrato, lo que permite definir si se quiere o no agregar a la *Blockchain* y/o en el caso de cobrar por su uso, definir cuánto va a ser su costo real.

En el caso de los lenguajes que son orientados a objetos, estos son elegidos principalmente por el tamaño de su comunidad y por su simplicidad a la hora de modelar un problema de la vida real. Es decir, se opta por elegir lenguajes que son muy utilizados por desarrolladores de forma de captar un mayor interés en el desarrollo de esta tecnología.

Tamaño de la comunidad

Se mide el tamaño de la comunidad ya que indirectamente permite evaluar cuál es la utilización real que tiene una plataforma, cuán difícil es comenzar a trabajar en ella y qué tan segura puede ser. Al tener una comunidad más activa la mejora es continua, por lo que se solucionan problemas constantemente y se proponen mejoras futuras.

Como se observa en la Figura 25, se realiza una compilación de los siguientes tres aspectos: (a) la cantidad de recursos que tiene el ***tag*** asociado al nombre de la plataforma dentro de *StackOverflow*; (b) la **cantidad de repositorios** en *GitHub* que se encuentran a partir de las siguientes búsquedas: "nombre plataforma" y "*Blockchain*", "nombre plataforma" y "*coin*", "nombre plataforma" y "*smart contract*"; y finalmente (c) el **total de la cantidad de preguntas** que contienen simultáneamente los siguientes *tags*: "nombre plataforma" y "*Blockchain*", "nombre plataforma" y "*smart contracts*", dentro de *StackOverflow*.

Plataforma	Búsqueda	#tags *	#repositorios	#preguntas		
			project: plataforma	is:question [plataforma] and [blockchain]	is:question [plataforma] and [smartcontracts]	TOTAL
Ethereum	ethereum	2517	214	526	315	841
EOS	eos	60	98	5	7	12
TRON	tron	30	42	8	2	10
Stellar	stellar	25	60	8	0	8
Neo	neo	0	19	0	0	0
Ethereum Classic	ethereum-classic / ethereumclassic	4	19	0	1	1
Waves	wavesplatform	69	125	19	16	35
RSK	rootstock	0	63	0	0	0
Quorum	quorum	98	38	31	2	33
Hyperledger Fabric	hyperledger-fabric	4373	130	644	18	662
Corda	corda	1732	91	82	4	86

Figura 25: *Cálculo para indicador de tamaño de la comunidad, obtenido 10 de Noviembre de 2019*

En base a esta información, se grafican las tres variables como se observa en la Figura 26, donde se muestra comparativamente qué plataformas tienen más representación por parte de la comunidad.

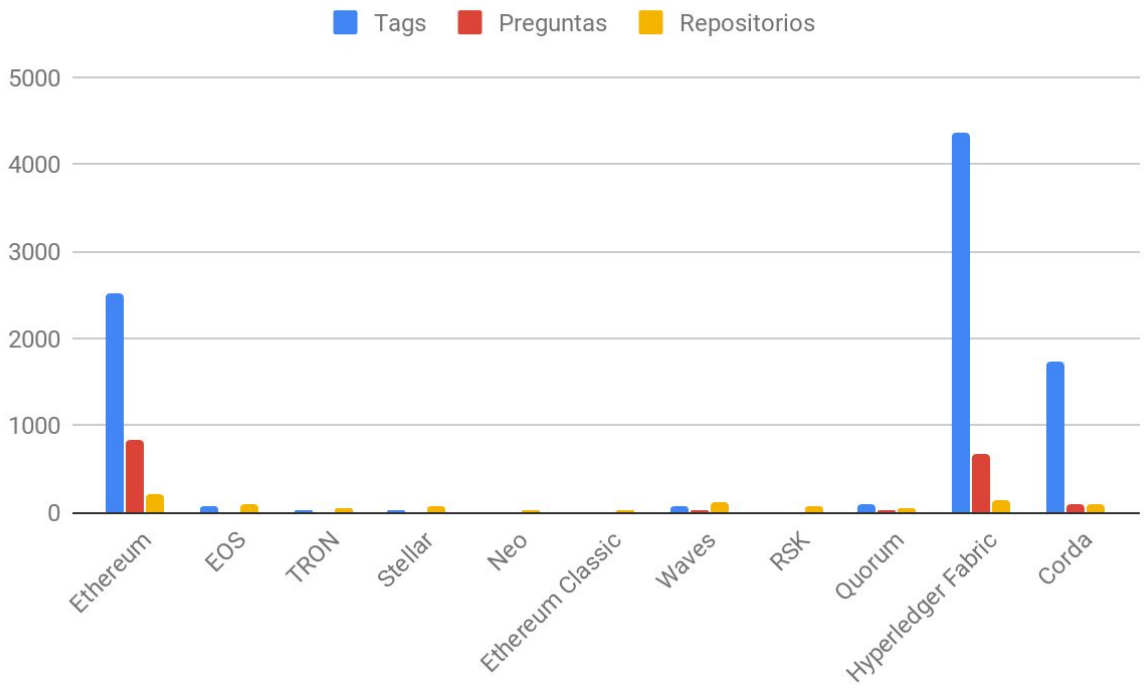

Figura 26: *Gráfico de tamaño de la comunidad, obtenido 10 de Noviembre de 2019*

Ethereum, Hyperledger Fabric y Corda son las más consultadas, actualizadas y representadas por parte de la comunidad de programadores. Se observa además que la cantidad de repositorios es distribuida de forma más uniforme, no así con las otras dos variables. En la Figura 27 se muestra la distribución de *tags* y total de preguntas. Observamos que tanto Ethereum como Hyperledger Fabric son las más consultadas, lo cual tiene sentido dado que ambas son las plataformas que entendemos que tienen más presencia en ambos sectores, público y privado (de uso empresarial).

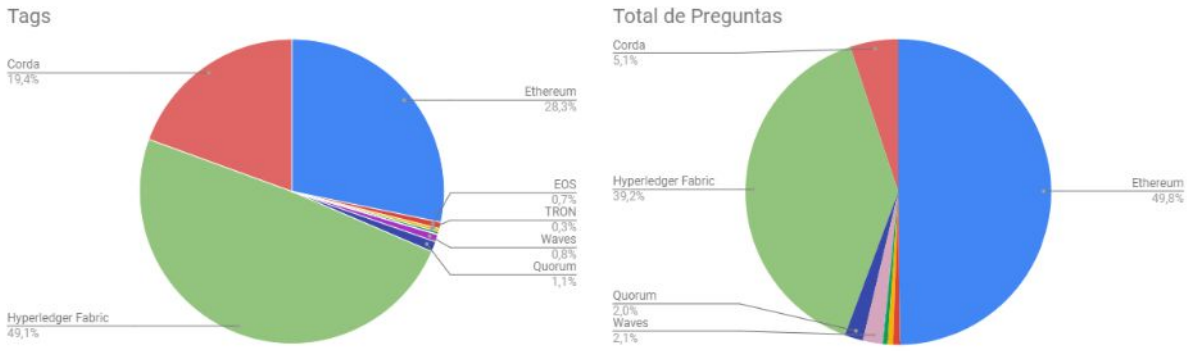

Figura 27: *Gráfico de Tags y Preguntas*

Tipo de acceso

Es importante evaluar esta característica en conjunto con las anteriores, ya que el tipo de acceso condiciona varios de los aspectos mencionados en las características anteriores. En la Figura 28 se observa la relación entre el tipo de acceso y el mecanismo utilizado.

	PoW	PoS y var	BFT	Total
Pública	3	3	1	7
Permisionada			2	2
Privada		x	x	2
Total	3	4	4	11

Figura 28: *Tipo de acceso/Mecanismo de consenso*

En base a las once plataformas estudiadas se concluye que las plataformas públicas buscan asegurar la consistencia del *ledger* mediante mecanismos más robustos como son: *PoW*, *PoS* y variantes. Por otro lado, para utilizar los mecanismos basados en *BFT* se necesita supuestos adicionales, como los que brindan las plataformas permisionadas, donde se conoce qué nodos son los responsables de la ejecución del mecanismo. Las plataformas privadas tienen la libertad de elegir qué mecanismo se adecua más a cada organización, sin embargo desestiman el uso de *PoW* por el consumo de recursos que conlleva. Un caso particular es la plataforma NEO ya que siendo pública utiliza un mecanismo que normalmente es utilizado en plataformas privadas (*dBFT)*. Esto lo puede debido a que gestiona la identidad de las personas/organizaciones que utilizan la red,

evitando así una red anónima de participantes que requiera una componente adicional de seguridad

	Solidity	Otros	Funcional	Total
Pública	4	2	1	7
Permisionada	1	1		2
Privada		2		2
Total	5	5	1	11

Figura 29: *Tipo de acceso/Lenguajes*

En cuanto a los lenguajes utilizados, como se mostró en las Figura 23, en nuestra muestra de once plataformas, ver Figura 29, se encuentra dividido de forma pareja entre lenguajes generales (Java, C++,etc) y específicos (Solidity). Es peculiar que para las plataformas públicas prime Solidity pero está justificado en que las plataformas estudiadas toman a Ethereum como base. Por otro lado, como las privadas pueden garantizar la seguridad debido a otras restricciones ya mencionadas, permiten que el usuario elija en qué lenguaje codificar los *smart contracts*. Waves al contrario, es una plataforma pública que utiliza *LPoS* y favorece la seguridad mediante las restricciones del lenguaje.

En cuanto al uso de Oráculos y la actualización de los *smart contracts*, no está condicionado al tipo de acceso que tengas configuradas las plataformas.

Oráculos

Debido a que los *smart contracts* tienen acceso únicamente a sus propias secciones de información y a la información dentro de otros *smart contracts*, es interesante analizar la posibilidad de consumir información del mundo exterior sin importar cómo estos son generados.

Se concluye que únicamente Stellar no permite la invocación debido a su implementación. No tiene habilitado dentro del conjunto de operaciones permitidas, una operación para invocar un oráculo. Al profundizar en este aspecto, la plataforma utiliza *smart contracts* para optimizar y automatizar transacciones, no tiene como

objetivo utilizarlos con otro fin como puede ser construir una *dApp*, entonces, es entendible esta restricción.

Modificación de contratos

Puede resultar contradictorio hablar de *smart contracts* modificables en el contexto de la tecnología *Blockchain*, ya que una de sus características principales es que la información es inmutable. Lo que sucede en la mayoría de las plataformas estudiadas es que implementan algún mecanismo, dentro de sus *smart contracts*, para habilitar esta característica. Algo importante es que el código del contrato desplegado en la *Blockchain* no se modifica, lo que se hace es aplicar algún patrón de diseño, específico para cada lenguaje y/o plataforma, que brinda esta posibilidad. En la Figura 30 se muestra la base de un ejemplo bastante usado en Solidity, que es la implementación de un patrón proxy, es decir, se despliega un contrato que va a recibir todas las invocaciones y las va a redireccionar a un contrato específico utilizando la dirección que tiene almacenada en su memoria (estado).

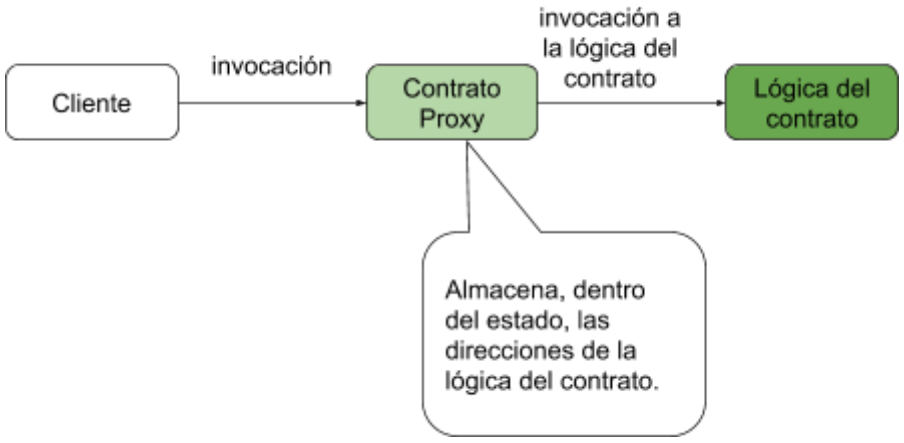

Figura 30: *Diagrama patrón proxy utilizado comúnmente en Solidity*

En algunos casos las plataformas también implementan operaciones que permiten deshabilitar ciertos contratos. Es decir, el código del contrato sigue estando almacenado dentro de la *Blockchain* pero el mismo no puede ser consumido por ningún cliente. Esta validación es realizada por los nodos que intervienen en la red y su utilización se muestra a continuación en la Figura 31.

```
pragma solidity ^0.5.11;

// To demonstrate how to force sending Ether to another contract:
// 1. Deploy the SelfDestruct contract, funding 1 Ether.
// 2. Deploy the Target contract.
// 3. Execute kill function in SelfDestruct, passing the address of Target as input.
// 4. Check the balance of Target contract. It should now have 1 Ether.

contract SelfDestruct {
    constructor() public payable {
    }

    function kill(address payable to) public {
        selfdestruct(to);
    }
}

contract Target {
    // Notice this contract does not have a payable fallback,
    // so we should not be able to send Ether to this contract...

    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

Figura 31: *Destrucción de Smart Contract con Solidity*

5. Conclusiones y trabajo futuro

En este trabajo se realizó una comparación de plataformas para *smart contracts* basadas en *Blockchain*, utilizando una serie de criterios que permiten mostrar las características más relevantes de esta tecnología. Para desarrollar este estudio se establecieron algunos objetivos que permitieron orientar la investigación y facilitaron el entendimiento: estudiar las principales características de la tecnología *Blockchain*, identificar y catalogar diferentes plataformas de *Blockchain* de 2ª y 3ª generación, determinar los criterios para la comparación de las plataformas y aplicarlos a un conjunto de plataformas seleccionadas.

En este capítulo se muestra el resultado de la comparación, se muestran las conclusiones obtenidas a partir del estudio realizado y se proponen algunas líneas futuras de investigación que permitan profundizar algunos conceptos y tecnologías tratadas.

Durante el transcurso del proyecto, hemos observado cómo esta tecnología está viva y su popularidad ha crecido enormemente, tanto es así que el ranking de las primeras veinte plataformas de CoinMarketCap ha cambiado radicalmente desde el día de comienzo de esta investigación. Algo que sucede en este rubro es que las plataformas continúan reinventándose buscando proporcionar lo que antes no hacían. Se observa cómo algunos cambios no son tan simples de impactar. Por ejemplo desde 2018 está planteado el proyecto de cambiar el mecanismo de consenso de Ethereum, la migración de *PoW* a *PoS.* Sin embargo dicha modificación ha sido postergada y todavía no se conoce la fecha de su liberación a producción. Por otro lado, Waves no soportaba el acceso a Oráculos y este año publicó tal funcionalidad. RSK sigue buscando su consolidación, encontrando nuevos aliados como RIFOS y volcándose a nuevos proyectos como *iovlabs.org.*

Si bien estas plataformas mutan y se fortalecen, el estado actual de la tecnología no permite que las plataformas se pueda utilizar como un sistema de pagos a nivel mundial, es decir, si se usan como plataforma de pagos pero no compiten con Visa,

al menos no hasta que mejore la velocidad de procesamiento de transacciones. Visa de acuerdo a su último reporte procesan más 65.000 mensajes de transacciones por segundo.

Por otro lado, existe un factor de utilización en estas tecnologías que es la moda, muchas veces se utiliza esta tecnología y no es necesario. Encontramos que en varias conferencias y medios de difusión, *Blockchain* se presenta como la gran solución a muchos problemas, pero realmente, no es la más conveniente en la mayoría de los casos. Una de las características fundamentales de *Blockchain*, en el contexto Bitcoin, es que funciona como una base de datos distribuida en una red no segura sin la necesidad de que exista un tercero con el rol de administrador. Para lograr esto es que utiliza una serie de pruebas criptográficas de alta complejidad, que garantizan que no todos los nodos participantes puedan validar los bloques. Esto provoca que los tiempos de procesamiento y validación sean muy largos, lo que no será de gran utilidad a empresas que necesiten procesar datos a altas velocidades. Para este caso lo importante será definir si es que se necesita tener un *ledger* inmutable compartido o si será suficiente una base de datos. En caso de optar por un *ledger*, será imprescindible estudiar cuál será el mecanismo de consenso que permita tales velocidades.

Otros datos importantes, basados en el último Survey de StackOverflow [Sos19], se encuentran en la consulta a los programadores de la comunidad sobre su opinión acerca de la tecnología de *Blockchain*, Figura 32.

En general, los usuarios más jóvenes y con menor experiencia son los más optimistas acerca su uso, sin embargo los usuarios con mayor experiencia consideran que el uso de esta tecnología es un desperdicio de recursos.

El resultado de la encuesta es que en un total de 60.165 participantes: un 29.2% consideran que es útil en muchos contextos y que puede cambiar algunos aspectos de nuestra vida; el 26.2% la considera útil para el registro de información que no tiene que ser modificada (no hablando de moneda); el 16.8% considera que es una

moda pasajera; el 15.6% considera que se hace un uso irresponsable de recursos y el 12.2% considera que es útil para manejar monedas distribuidas (Bitcoin).

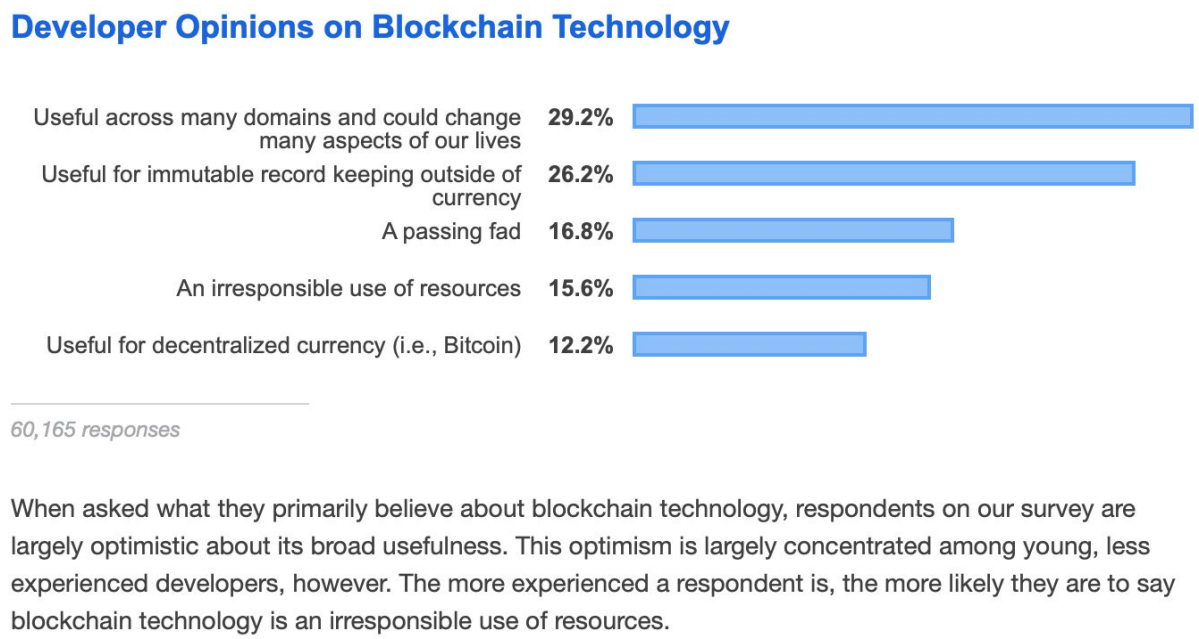

Figura 32: *Opinión acerca de Blockchain* [Sos19]

Por otro lado también se le consultó a los usuarios como las organizaciones utilizan realmente la tecnología de *Blockchain*, Figura 33. La mayoría de las respuesta muestra como no se está usando o implementando soluciones con esta tecnología, y cuando si se usa, no es para las transacciones relacionadas al intercambio de monedas. Curiosamente, los desarrolladores que indicaron que sus organizaciones están utilizando esta tecnología están en India.

Sobre un total de 48.175 respuestas el resultado de la encuesta es que: 80% indicaron que no la usan para nada; 12.7% la utilizan pero no con fines de intercambio de moneda; 4% desarrollan productos basados en el intercambio de moneda, 2.1% aceptan Bitcoin u otras criptomonedas como medio de pago y 1.3% implementan su propia criptomoneda.

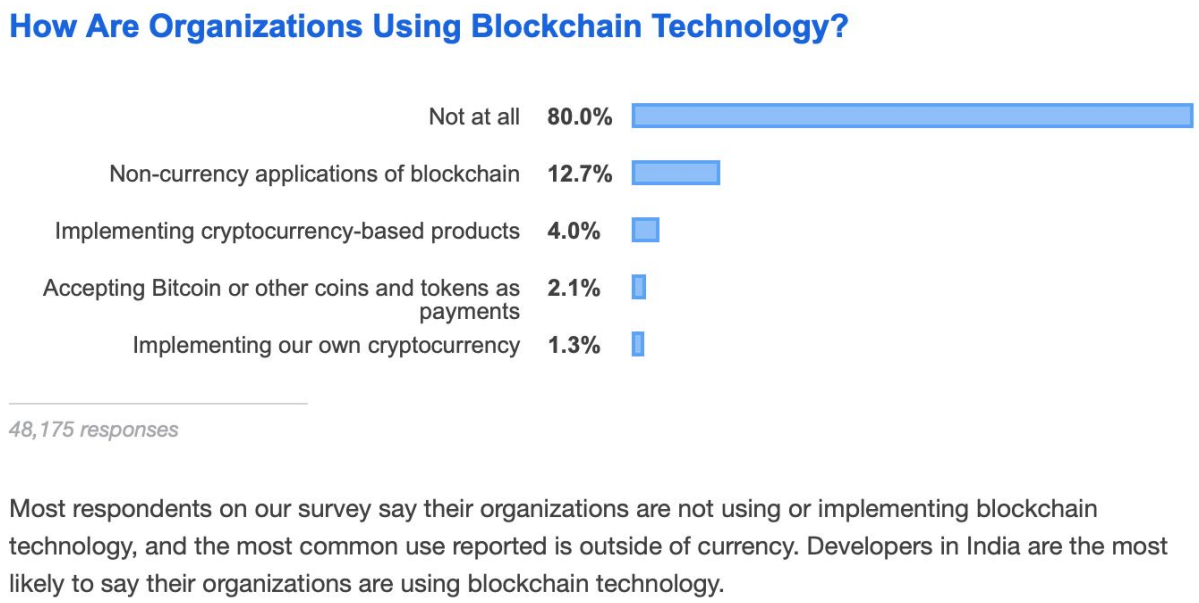

Figura 33: *Opinión acerca de cómo se usa Blockchain* [Sos19]

Algo de lo que también se habla es si los smart contracts van a reemplazar la asesorías legales, o sea, de si esta descentralización conlleva a la modificación de los marcos regulatorios. Surge entonces la pregunta de quién es realmente responsable por las fallas contractuales y los vacíos legales. Lo que se concluye es que por la madurez de la tecnología, la asesoría legal no puede ser reemplazada aunque por otro lado se abren nuevos paradigmas de acuerdos e intercambios entre personas. Los abogados y notarios tienen que ayornarse a esta tecnología para ofrecer mejores servicios y competir en su mercado. El ideal es un equipo de trabajo mixto, entre ingenieros y abogados para escribir *smart contracts* que resulten legalmente válidos.

A continuación se proponen algunos lineamientos de trabajo a futuro.

Debido a que las plataformas de *Blockchain* buscan posicionarse como líderes en el sistema de intercambio de divisas y pagos online, es interesante investigar qué componentes (mecanismos de consenso, Virtual Machine, lenguajes permitidos, tipo de acceso) permiten garantizar mayor seguridad y optimizar los tiempos de procesamiento.

Siguiendo esta línea es interesante investigar en profundidad la migración de Ethereum a su versión 2.0, Figura 34. Dentro de esta versión se va a modificar el

mecanismo de consenso (se deja de usar *PoW* para usar *PoS*), se modifica la máquina virtual con la que se ejecutan los *smart contracts* para lograr mejorar la velocidad de ejecución del código (deja de ser EVM para usar eWASM) y se agrega soporte a permisos/privacidad, entre otros.

Figura 34: *Roadmap de Ethereum* [Con19]

Queda por fuera del alcance de este proyecto la investigación de Open Zeppelin [Ope19], una empresa que tiene por objetivo auditar y otorgar ciertas garantías a los contratos que revisan. Proveen una librería con distintas funcionalidades relacionadas a la calidad. Es interesante evaluar qué características prueban, a qué plataformas se les ofrece, cuanto es usado en la comunidad mundial, entre otras.

Finalmente se espera que esta investigación sea de utilidad tanto para comprender los principales conceptos de las plataformas de Blockchain como las características relevantes al momento de elegir una para el desarrollo de los *smart contracts.*

6. Referencias bibliográficas

[AM18] Ackermann, J., Meier, M. (2018). *Blockchain 3.0: The next generation of blockchain systems*. Advanced Seminar Blockchain Technologies, Summer Term 2018, Technical UniversityMunich

[BE19] Blockchain Luxembourg S.A | Blockchain Explorer (2019). Disponible: https://www.blockchain.com/explorer?view=btc_transactions. Consultado: 4/11/2019

[BP17] Bartoletti, M., & Pompianu, L. (2017). *An empirical analysis of smart contracts: platforms, applications, and design patterns. In International conference on financial cryptography and data security. Lecture Notes in Computer Science vol. 10323, pp. 494-509.* Springer, Cham.

[BS18] Begicheva, A., & Smagin, I. (2018). RIDE: a Smart Contract Language for Waves. Technical report, Waves, 2018. URL https://wavesplatform. com/files/docs/white_paper_waves_smart_contracts.pdf.

[BSA+17] Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., & Danezis, G. (2019). *SoK: Consensus in the age of blockchains*. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies (pp. 183-198).

[BSK+18] Baliga, A., Subhod, I., Kamat, P., & Chatterjee, S. (2018). *Performance evaluation of the quorum blockchain platform*. arXiv preprint arXiv:1809.03421.

[But13] Buterin, V. (2013). *Ethereum white paper*. GitHub repository, 22-23. Disponible: https://github.com/ethereum/wiki/wiki/White-Paper. Consultado: 20/11/2019

[CBB17] Clack, C., Bakshi, V., Braine, B. (2017) *Smart Contract Templates: foundations, design landscape and research directions*. https://arxiv.org/pdf/1608.00771 Zugriffsdatum, 2018, vol. 27.

[Cdm13] CoinDesk Digital Media | CoinDesk (2013). Disponible:https://www.coindesk.com/. Consultado: 20/07/2019

[Cmc19] Cryptocurrency Market Capitalizations | CoinMarketCap (2019). Disponible: https://coinmarketcap.com/. Consultado: 25/09/2019

[Coi19] CoinMonk | Medium (2019). Disponible: https://medium.com/coinmonks/how-does-hyperledger-fabric-works-cdb68e6066f5. Consultado: 15/12/2019

[Con19] Consensys media (2019). Disponible: https://media.consensys.net/the-roadmap-to-serenity-bc25d5807268. Consultado: 12/12/2019

[Dem15] Demian, S. (2015) *RSK White Paper*. Disponible: https://docs.rsk.co/RSK_White_Paper-Overview.pdf. Consultado: 6/10/2019

[Dem16] Demian, S. (2016) *Drivechains, Sidechains and Hybrid 2-way peg Designs*. Disponible:https://docs.rsk.co/Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf. Consultado: 10/11/2019

[DSW94] Davis, M.,Sigal R.,Weyuker E., "Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science", Second Edition, Academic Press,1994.

[Eos17] IO, E. (2017). *EOS. IO technical white paper*. *EOS.* Disponible: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md. Consultado:19/4/2019

[Eos19] EOS Developer Site (2019). Disponible: https://developers.eos.io/. Consultado:15/12/2019

[ETC16a] Ethereum Classic. (2016). *Ethereum Classic*. Disponible: https://ethereumclassic.github.io. Consultado: 6/10/2019

[ETC16b] Ethereum Classic. (2016). *The Ethereum Classic declaration of independence*. Disponible:

https://ethereumclassic.github.io/assets/ETC_Declaration_of_Independence.pdf. Consultado: 6/10/2019

[GET20] Go Ethereum. (2020). *Go Ethereum*. Disponible: https://github.com/ethereum/go-ethereum. Consultado: 9/5//2020

[Hea16] Hearn, M. (2016). *Corda: A distributed ledger. Corda Technical White Paper.* Disponible: https://docs.corda.net/releases/release-V3.1/_static/corda-technical-whitepaper.pdf. Consultado: 6/10/2019

[Hyp16] Hyperledge Fabric. (2016). *Hyperledge Fabric White Paper.* Disponible: http://blockchainlab.com/pdf/Hyperledger%20Whitepaper.pdf. Consultado: 6/10/2019

[Hyp19] Hyperledge Fabric SmartContracs (2019) Disponible: https://hyperledger-fabric.readthedocs.io/en/release-1.4/smartcontract/smartcontract.html. Consultado: 15/12/2019

[LSP82] Lamport, L., Shostak, R., Pease, M. *The Byzantine Generals Problem*. Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, Pages 382-40

[Maz16] Mazieres, D (2016). *Stellar White Paper.* Disponible: https://www.stellar.org/papers/stellar-consensus-protocol.pdf. Consultado:19/04/2019

[MCA+17] Mehar, M., Shier, C., Giambattista, A., Gong, E., Fletcher, G., Snayhie, R., Kim, H., Laskowski, M. (2017) *Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack.* Journal of Cases on Information Technology 21(1) 19-32.

[Med19] Repositorio de Medium (2019). *Ethereum bytecode*. Disponible: https://miro.medium.com/max/2796/1*JxvsdWRFQfByRNy7xDEQ5w.png. Consultado: 19/11/2019

[MO17] McDonald, J., Oliverio, J. (2017). *NEM White Paper*. Disponible: https://nem.io/wp-content/themes/nem/files/ApostilleWhitePaper.pdf. Consultado: 19/04/2019

[Mor17] Morgan,JP. (2017) *Quorum White Paper.* Disponible: https://github.com/jpmorganchase/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf. Consultado: 6/10/2019

[Nak08] Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system*.

[NBF+16] Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press.

[Neo18] (2018). *NEO White Paper*. Disponible: https://docs.neo.org/docs/en-us/basic/whitepaper.html. Consultado: 19/04/2019

[Nxt20] NXT (2020). Disponible: https://www.jelurida.com/nxt. Consultado: 10/5/2020

[Ope19] Open Zeppelin (2019). Disponible: https://openzeppelin.com/contracts/. Consultado: 15/12/2019

[Qix13] Sitio oficial de QixCoin. Disponible: http://qixcoin.com/. Consultado: 10/05/2020

[Sos19] StackOverflow Survey 2019. Disponible: https://insights.stackoverflow.com/survey/2019#blockchain-in-the-real-world. Consultado: 5/12/2019

[SSS17] Sankar, L. S., Sindhu, M., & Sethumadhavan, M. (2017, January). *Survey of consensus protocols on blockchain applications*. In 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 1-5). IEEE.

[Ste19] Stellar Developer Guides 2019. Disponible: https://www.stellar.org/developers/guides/get-started/. Consultado: 15/12/2019

[Sza94] Szabo, N. (1994). *Smart Contracts*. Disponible: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html. Consultado: 15/04/2019

[Sza96] Szabo, N. (1996). *Smart Contracts: Building Blocks for Digital Markets.* Disponible: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. Consultado: 8/05/2019

[TF18] Tron Fundation (2018). *TRON White Paper*. Disponible: https://tron.network/static/doc/white_paper_v_2_0.pdf. Consultado: 19/04/2019

[TF19] Tron Fundation (2019). *TRON Network*. Disponible: https://tron.network/. Consultado: 15/04/2019

[Tnw17] The Next Web (2017). Disponible: https://answers.thenextweb.com/s/vitalik-buterin-13gxQB. Consultado: 30/11/2019

[VISA19] Visa Facts (2019). Disponible: https://usa.visa.com/dam/VCOM/global/about-visa/documents/visa-fact-sheet-july-2019.pdf. Consultado: 2/11/2019

[VH19] Viriyasitavat, W., & Hoonsopon, D. (2019). *Blockchain characteristics and consensus in modern business processes. Journal of Industrial Information Integration*, 13, 32-39.

[VS17] Valenta, M., & Sandner, P. (2017). *Comparison of ethereum, hyperledger fabric and corda*. [ebook] Frankfurt School, Blockchain Center. Disponible: https://pdfs.semanticscholar.org/00c7/5699db7c5f2196ab0ae92be0430be4b291b4.pdf. Consultado:6/10/2019

[Wav16] (2016) *Waves White Paper.* Disponible: https://blog.wavesplatform.com/waves-whitepaper-164dd6ca6a23. Consultado: 19/04/2019

[Wav19] (2019) Waves official documentation. Disponible: https://docs.wavesplatform.com/en/. Consultado: 19/10/2019

[WZ18] Wohrer, M., & Zdun, U. (2018, March). *Smart contracts: security patterns in the ethereum ecosystem and solidity*. In 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE) (pp. 2-8). IEEE.

[ZXD+17] Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017, June). An overview of blockchain technology: *Architecture, consensus, and future trends*. In 2017 IEEE International Congress on Big Data (BigData Congress) (pp. 557-564). IEEE.

[ZXD+18] Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). *Blockchain challenges and opportunities: A survey*. International Journal of Web and Grid Services, 14(4), 352-375.