Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

Un avance hacia entornos de gran escala para experimentos con criptomonedas

Tesis presentada para optar al título de

Licenciado en Ciencias de la Computación

Marco Vanotti

Director: David Alejandro González Márquez

Codirector: Maximiliano Iván Geier

Buenos Aires, 2016

RESUMEN

En los últimos años las criptomonedas descentralizadas, tales como Bitcoin, tomaron gran impulso en el mundo de la computación. Éstas permiten realizar pagos de forma segura sin depender de una tercera parte de confianza. Sin embargo, sus implementaciones presentan problemas para el uso a gran escala, ya que los tiempos necesarios para procesar y validar transacciones son muy altos.

Para mitigar estos problemas, existen una gran variedad de propuestas. RSK propone reducir el tiempo entre bloques a 10 segundos junto con cambios en los protocolos de consenso y red.

En este trabajo se presenta una metodología para el análisis y evaluación de cambios en sistemas de criptomonedas. La misma consiste en instrumentar el cliente de la criptomoneda sobre una red emulada, obteniendo métricas que reflejen el estado del sistema.

Se analizó el efecto de la variación del tiempo entre bloques y se simuló un minado con pools de minería usando merged-mining.

Se pudo corroborar que disminuyendo el tiempo de generación de bloques a valores muy pequeños, los tiempos de propagación tienen un efecto significativo sobre la cantidad de forks en la red.

Además, el análisis sobre merged-mining arrojó que, bajo la configuración utilizada, tener un tiempo entre bloques de 10 segundos resulta inviable.

Finalmente la herramienta utilizada permite realizar experimentos de gran tamaño ya que la misma escala horizontalmente utilizando hardware convencional.

ABSTRACT

In recent years there has been a great rise in popularity of decentralized cryptocurrencies, such as Bitcoin. These currencies allow secure payments without depending on a trusted third party. However, their current implementations don't allow massive usage, because of high processing and validation times.

There is a wide variety of proposals to address these problems. RSK proposes a 10-second interval between blocks and changes in consensus and network protocols.

In this work we present a tool for analysis and evaluation of changes in cryptocurrency systems. In order to get metrics that reflect the system state, we instrumented the cryptocurrency client and ran it over an emulated network.

We analyzed the effect of changing the mean time between blocks and also simulated a network with mining pools using merged-mining.

We could verify that by decreasing the interval between blocks to very small values, the block propagation time has a significant impact over the amount of forks in the network.

Regarding the mining pool simulation with merged-mining, our analysis showed that using a specific hardware configuration, having a block generation time of 10 seconds is not viable.

Finally, the proposed tool allows for large-scale experiments as it scales horizontally using commodity hardware.

En primer lugar, quiero agradecer a Claudio y Christian, los jurados, por tomarse el tiempo y trabajo de corregir esta tesis, haciendo un esfuerzo por ajustarse al estricto cronograma con el que contábamos.

Al Departamento de Computación y a la Universidad de Buenos Aires, por permitirme acceder a una educación de primer nivel de forma gratuita.

A la gente de RSK Labs: Sergio, Ruben, Adrian y Diego, por darme la oportunidad de hacer la tesis con ellos e introducirme en el mundo de las criptomonedas. También a Martin Medina, quien cometió el error de influír en esa decisión.

A David y Maxi, mis directores, por haber aceptado dirigir esta tesis aún cuando no era su area de interés. Por su infinita paciencia, predisposición y meticuloso detalle. No me van a alcanzar los días de mi vida para agradecerles por el excelente trabajo que hicieron y la dedicación que pusieron. Este trabajo es tanto mio como de ellos.

A Esteban Mocskos, por aceptarme en el laboratorio incluso antes de haber comenzado la tesis. Por sus valiosos comentarios sobre el trabajo, y por sobre todo, por prestarme a dos de sus mejores tesistas para dirigir esta tesis.

A la gente del LSQ y las chicas de secretaría, por todas las tardes y mates compartidos, por todo el aliento que pusieron en las últimas semanas para que llegue a terminar a tiempo la tesis.

A mis compañeros de cursada, de estudio y de grupos a lo largo de la carrera. Si llegué a hacer la tesis, es porque antes tuve excelentes grupos con los que aprobé las materias.

A la gente de Orga2, la mejor materia de la carrera, donde pude formarme como docente y conocer excelentes personas.

A Juan José Miranda Bront, por darme una mano cuando más la necesitaba.

A mis amigos que me acompañan desde antes de comenzar la carrera, hasta el final: Martin Ventura, Matías Marquez y Patricio Palladino.

A mi familia, mis padres, mi hermano, mis primos, mis tíos y abuelas, por estar siempre presentes y ser la mejor familia que uno puede esperar.

A Nadia, mi compañera de vida en estos últimos años, por aguantarme, soportarme y acompañarme todo este tiempo.

A la abuela Inés, por alentarme siempre...

Índice general

1. INTRODUCCIÓN

Con el nacimiento de *Bitcoin* en 2008 [Nak08], las criptomonedas tomaron impulso en el mundo de la computación. Una criptomoneda es un medio de cambio digital que utiliza tecnología criptográfica para asegurar la veracidad de las transacciones. Una transacción es una transferencia de dinero entre dos pares.

Hay un gran interés por parte de la comunidad del mundo de las criptomonedas por tener un sistema competitivo contra sistemas de pago online como Visa o Paypal. Estos sistemas pueden procesar gran cantidad de transacciones por segundo, y dar tiempos de confirmación instantáneos. Para lograr esto en criptomonedas como Bitcoin se debe aumentar la cantidad de transacciones que se pueden procesar por segundo, como así también minimizar el tiempo necesario para tener garantía de que esta transacción se encuentra procesada.

Este trabajo de tesis tiene dos objetivos:

I) Generar un método para evaluar posibles cambios a aplicar sobre criptomonedas, con foco en los protocolos de red.

II) Analizar cambios sobre la criptomoneda *RSK*, utilizando el mismo código que se pondrá en producción, y usando topologías similares a las utilizadas por criptomonedas.

En este capítulo se introduce el funcionamiento de Bitcoin y de las tecnologías que utiliza. Además, se presentan dos criptomonedas adicionales: Ethereum y RSK. Ethereum provee la funcionalidad de *smart contracts*, mientras que RSK es la criptomoneda sobre la que está basado este trabajo.

Inicialmente se presenta una breve introducción a Bitcoin, que da lugar a las estructuras fundamentales de las criptomonedas: Blockchain, Bloques y Transacciones, como así también de los protocolos y procesos: Minado, Consenso e intercambio de mensajes.[1]

[1] El libro «Mastering Bitcoin» [Ant14] contiene una descripción más técnica y detallada de todos los temas aquí presentados.

Luego se menciona brevemente el concepto *smart contracts* y a Ethereum, haciendo énfasis en las diferencias de esta criptomoneda con Bitcoin.

También se presenta RSK, explicando su motivación y relación con Ethereum y Bitcoin, explicando algunas de las tecnologías claves que esta criptomoneda presenta.

El capítulo concluye con una explicación de qué son los *Pools de Minería* y cómo se relacionan con Bitcoin y RSK.

1.1. Breve introducción a Bitcoin

Bitcoin fue propuesta en el 2008 por una persona bajo el seudónimo «Satoshi Nakamoto» [Nak08]. Fue la primer criptomoneda descentralizada que resolvía el problema de *double spending* sin necesidad de confiar en una *tercera parte de confianza (trusted third-party)*.

Una criptomoneda es un medio de cambio digital que utiliza tecnología criptográfica para asegurar la veracidad de las transacciones.

Se puede ver a la transferencia de una moneda digital como el endosado de un cheque: una persona escribe en el dorso el destinatario del dinero, y este puede a su vez endosarlo nuevamente. También se puede saber si la persona que se lo dio o bien fue el dueño del cheque, o bien fue el último en endosarlo antes que él.

En el mundo electrónico, se puede lograr algo similar con firmas digitales y *hashes criptográficos*: cuando una persona quiere transferir dinero digital a otra, se crea una transacción, que no es más que la firma digital del *hash criptográfico* de la transacción anterior que usó ese dinero y la clave pública del destinatario. De esta forma, el destinatario puede verificar que el emisor era realmente el dueño del dinero, verificando la firma digital contra la transacción con el hash dado, y además, puede volver a transferirla usando él su clave privada.

En la figura 1.1 se presenta una esquematización de este proceso en donde se realizan tres transacciones de una criptomoneda, se puede ver que el dueño de la primer clave privada firma el hash que hace referencia a la transacción anterior y a la clave pública del nuevo destinatario,

creando una nueva transacción. Este proceso se repite una vez más entre el nuevo destinatario y una tercer persona.

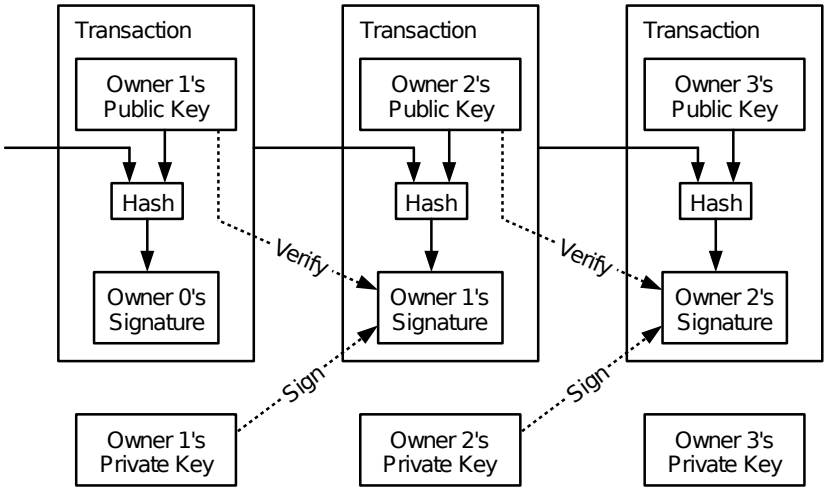


Fig. 1.1: Esquema de transacciones en una criptomoneda. Fuente: [Nak08]

Esto deja lugar a un problema conocido como *doble gasto (double spending)*: el emisor puede crear todas las transacciones que quiera a partir de una moneda, pero sólo una de estas debería ser válida. Una solución a este problema podría ser contar con una entidad de confianza, que se encargue de validar todas las transacciones.

Bitcoin propone una solución diferente al problema de double spending, sin necesidad de confiar en una tercera parte. La idea principal es que todos los nodos de la red tengan copia de un registro de todas las transacciones que fueron procesadas, y que cualquiera pueda escribir en ese registro.

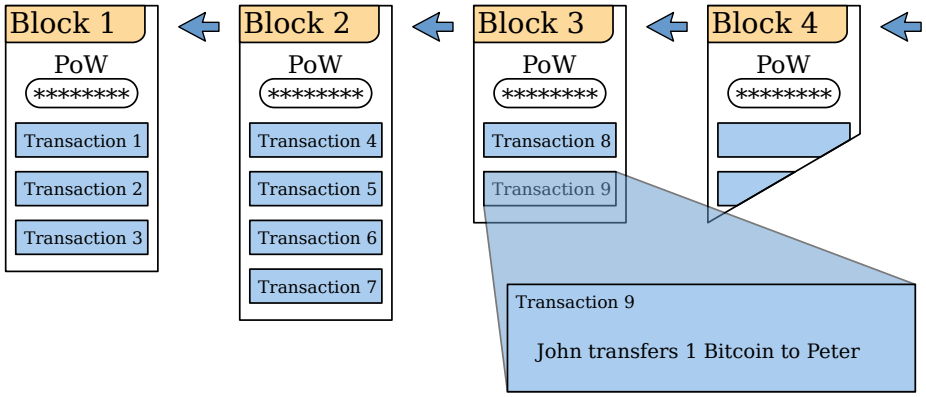


Fig. 1.2: Esquema de blockchain con transacciones.

Este registro replicado y distribuido se conoce como **Blockchain**. Luego, no puede ocurrir double spending, ya que una transacción es inválida si en la Blockchain existe otra transacción que hace referencia a la misma transacción «padre» que ésta. En la figura 1.2 se puede apreciar una esquematización de esta estructura.

1.2. Estructuras de datos

En esta sección se describirán en detalle las estructuras de datos utilizadas en Bitcoin. Estas estructuras son la base de todas las criptomonedas descentralizadas, por lo cual es importante analizar las ideas detrás de ellas.

1.2.1. Transacciones

Una transacción en Bitcoin, está definida por una lista de *inputs* y una lista de *outputs*. Un *output* consiste en un par de $(cantidad,\ hash)$ donde el primer valor es la cantidad de Bitcoins que el destinatario va a recibir y el segundo es el hash criptográfico de la clave pública del destinatario.

La lista de inputs $(hash,\ indice,\ clave,\ firma)$ hace referencia a outputs de transacciones anteriores. Un input contiene:

- El hash de la transacción a la que hace referencia.
- El índice del output deseado dentro de la transacción.
- La clave pública correspondiente a dicho output.
- Una firma digital del hash de la transacción usando la clave privada correspondiente a la clave pública del input.

De esta forma, el input hace uso de un output anterior. Para verificar que este uso es válido, se calcula el hash de la clave pública y verifica que sea igual a la que figura en el output usado, luego basta con verificar la firma digital con esa clave pública.

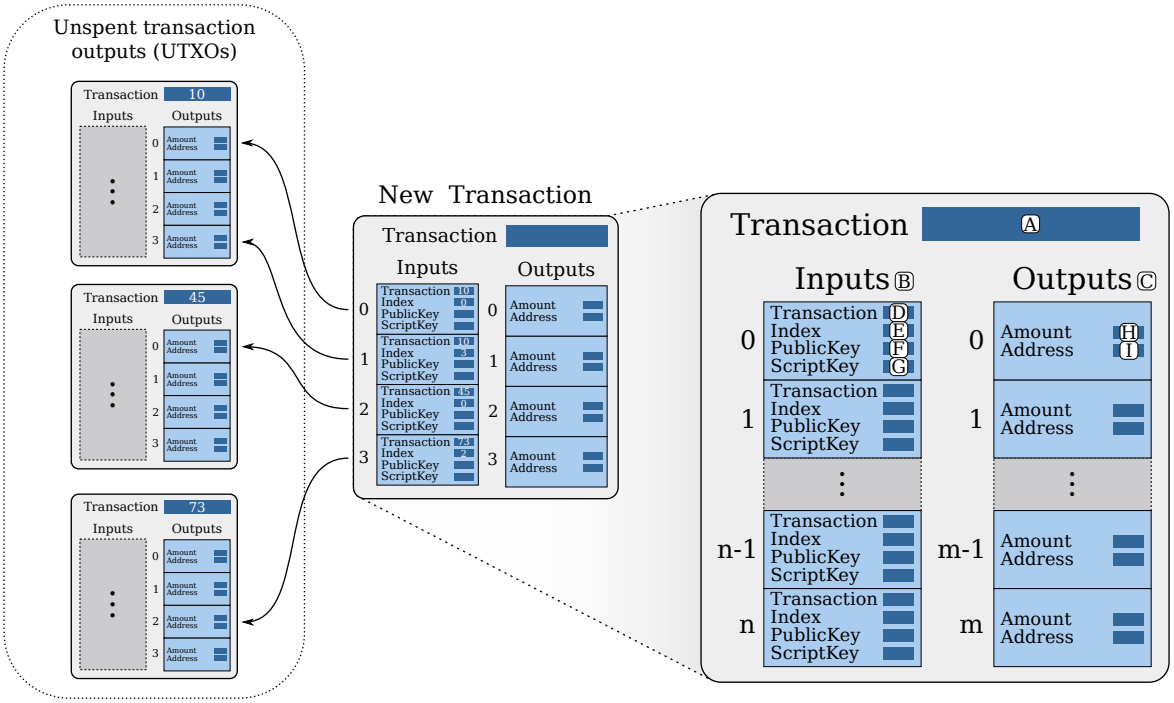


Fig. 1.3: Ejemplo de una transacción en Bitcoin, sus inputs hacen referencias a outpus de transacciones anteriores.

En la figura 1.3 se puede ver un ejemplo de una transacción con varios inputs. Del lado derecho, se destacan ciertos campos:

(A) El hash que identifica de forma unívoca a la transacción.

(B) La lista de inputs de la transacción, cada elemento hace referencia a un output de alguna transacción anterior que todavía no fue usado.

(C) La lista de outputs de la transacción.

(D) El hash de la transacción a la que ese input hace referencia.

(E) El índice del output dentro de la lista de outputs de la transacción referenciada.

(F) La clave pública que se corresponde con el *address* del output.

(G) Una firma digital con la clave privada correspondiente a la clave pública, para demostrar que se tiene control sobre la misma y por lo tanto, se puede usar ese output.

(H) La cantidad de Bitcoins a transferir a ese output.

(I) La dirección destino: el hash de la clave pública del destinatario.

Esta es la versión más sencilla de transacciones. Bitcoin soporta un mecanismo llamado *Pay To Script Hash (P2SH)* que permite dar un hash de un script que es almacenado como address en el output. Luego para usar este output, se deben proveer parámetros para ejecutar el script correspondiente al hash del output. Para que esta transacción sea valida, el script debe ejecutarse satisfactoriamente.

Estos scripts están limitados en el sentido en que no tienen ciclos, restringiendo el poder de expresividad. Esto tomará mayor importancia más adelante, cuando se hable de Ethereum y Smart Contracts.

Para verificar que una transacción es válida, se debe verificar que todos los inputs son validos, y que la suma de los outputs referenciados por esos inputs es mayor o igual a la suma de los outputs de la transacción. Si la suma de los inputs es mayor a la de los outputs, el excedente podrá ser reclamado por quien agregue la transacción a la Blockchain, este dinero extra es lo que se conoce como *«transaction fees»*.

Una vez que un output de una transacción fue usado, ya no se puede volver a usar. Es decir, sólo se pueden considerar los outputs de transacciones que no fueron utilizados previamente, a estos se los conoce como *«Unspent Transaction Outputs» (UTXOs)*.

Para que una transacción válida sea agregada a la Blockchain, ésta debe formar parte de un *Bloque*. Los bloques agrupan transacciones y son la unidad básica de la Blockchain.

1.2.2. Bloques

En Bitcoin, cada bloque está compuesto por:

- El *hash* del bloque anterior (su padre).
- Tiempo en segundos en que fue minado, desde *epoch*[2].

[2] 1 de Enero de 1970, a las 00:00:00 UTC

- La dificultad con la que ese bloque fue hallado.[3]
- `nonce`: La prueba de trabajo, que demuestra que el bloque cumple con la dificultad que dice tener.
- Un número de versión.
- Un conjunto de transacciones ordenadas.

No todos los bloques son válidos. Para que un bloque sea válido y aceptado por la red, debe contar con lo que se conoce como *Proof of Work (PoW)*. El `nonce` permite verificar de forma sencilla que el bloque cumple con la prueba de trabajo requerida. El mecanismo que se usa para hallar el `nonce` y para validarlo se conoce como proceso de minado, y será explicado más adelante.

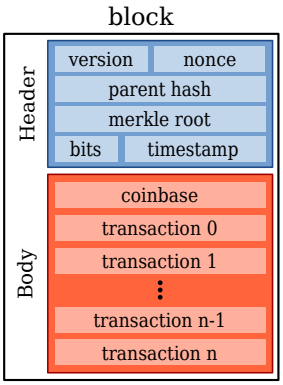


Fig. 1.4: Bloque Bitcoin

Los bloques se separan en *header* y *body*, como puede verse en la figura 1.4. En Bitcoin, el header contiene todos los campos mencionados anteriormente, salvo las transacciones, que se encuentran en el body. El header cuenta con el hash de un *Merkle Tree* que contiene las transacciones.

Un Merkle Tree es un árbol binario de hashes criptográficos, en donde cada nodo contiene el hash de la concatenación de sus dos hijos. En la figura 1.5 se puede ver cómo se forma la raíz de este árbol de hashes, usando todas las transacciones que se encuentran en el body del bloque.

[3] Esta dificultad se encuentra reflejada en el campo «bits» del header del bloque (ver figura 1.4).

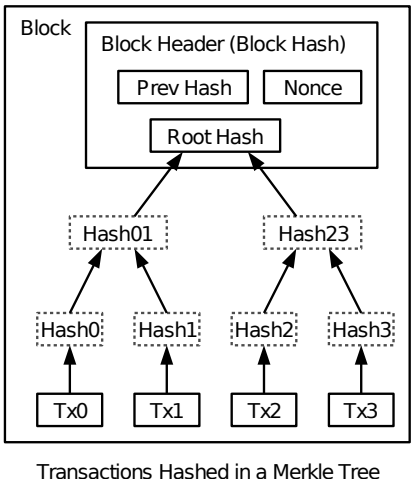


Fig. 1.5: Ejemplo de Bloque con Merkle Tree. Fuente: [Nak08]

1.2.3. Blockchain

La estructura de la Blockchain puede verse como un árbol de Bloques, se diferencia de una lista pues puede suceder que dos bloques tengan el mismo padre, dando lugar a ramificaciones. La Blockchain tiene una cadena[4] principal, llamada *main chain*, que es la cadena con mayor dificultad total (suma de dificultades de todos los bloques que la componen). Los bloques que no pertenecen a la *main chain* son llamados bloques *«stale»*.

La raíz de la Blockchain es un bloque especial llamado *Bloque Génesis*. En Bitcoin, este bloque fue minado el 3 de enero de 2009, y además, en uno de sus campos contenía un titular del diario *«The Times»* de esa fecha[5], para dar prueba que este bloque no fue creado con gran antelación a ese día.

A la Blockchain se agregan bloques, que contienen transacciones. El número de un bloque es la altura del bloque dentro de la Blockchain. Si un bloque contiene transacciones cuyos inputs hacen referencia a outputs que ya fueron gastados en un bloque de menor número dentro de su cadena, el bloque es inválido. De esta forma, en una cadena no se pueden agregar bloques que hagan *double spending*.

Cualquier nodo de la red puede construir bloques, denominándose *nodo minero*. Al recibir un

[4] Se entiende como cadena a la lista de bloques desde un nodo hasta la raíz del árbol.
[5] «The Times 03/Jan/2009 Chancellor on brink of second bailout for banks»

bloque, los nodos lo validan y lo agregan a su Blockchain. Es posible que cambien su cadena principal, si el nuevo bloque pertenece a una cadena previamente *stale* y esta ahora tiene mayor dificultad total que la cadena principal. Para que un bloque sea válido y aceptado por los nodos de la red, debe contener una prueba de trabajo (PoW), que valide que el nodo minero hizo uso de CPU para poder construir el bloque.

En la figura 1.6 se puede ver un panorama general del funcionamiento de la Blockchain:

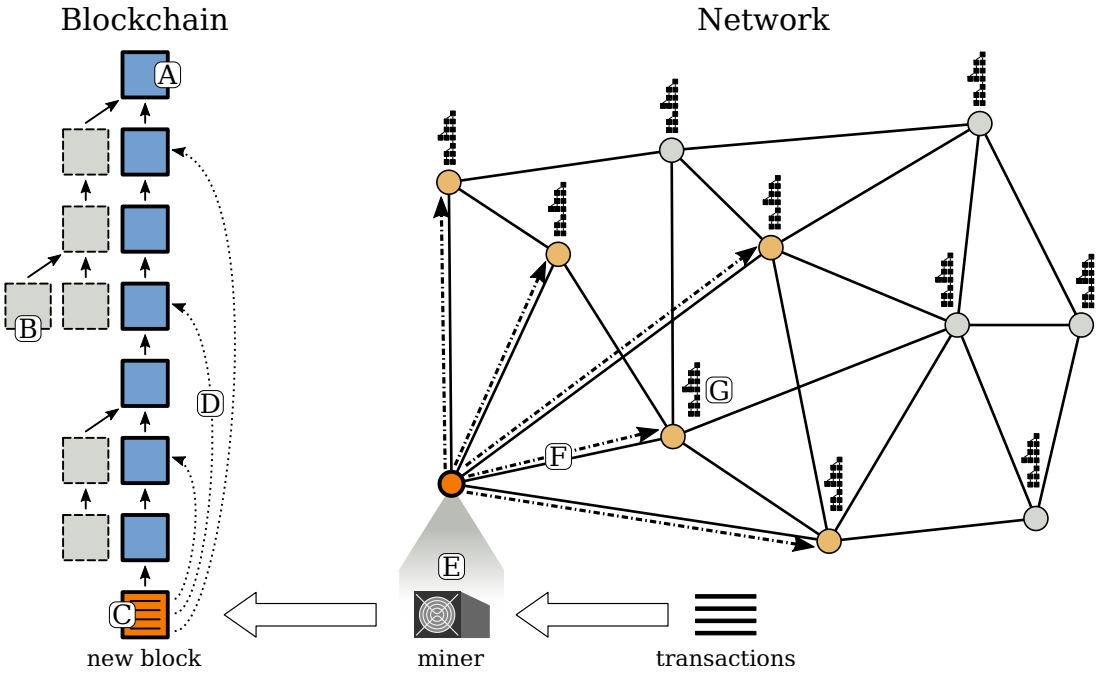


Fig. 1.6: Esquema de sistema de criptomoneda descentralizada.

(A) La Blockchain es un árbol de bloques que cuenta con una cadena principal. El bloque raíz se conoce como bloque génesis.

(B) Los bloques que no pertenecen a la cadena principal son bloques stale y no son tenidos en cuenta al momento de validar transacciones.

(C) Un nuevo bloque se agrega a continuación de algún bloque de la Blockchain. En este caso el bloque forma parte de la cadena principal. Las transacciones que agrega este bloque ocurren después de todas las transacciones que fueron agregadas en bloques anteriores en su cadena.

(D) Los inputs de las transacciones de un bloque hacen referencia a outputs de transacciones

anteriores que todavía no fueron gastados.

(E) Un nodo minero agrupa transacciones para crear un nuevo bloque, teniendo que hallar una PoW para que este sea válido.

(F) Al hallar un nuevo bloque, el minero lo propaga por la red, y el resto de los nodos al recibirlo y validarlo también lo propagan.

(G) Todos los nodos de la red tienen una copia de la Blockchain, que se usa para validar transacciones y bloques. Actualizan su copia con cada bloque válido recibido.

1.3. Proceso de minado y consenso

En Bitcoin los nodos de la red deciden en conjunto cuál será el próximo bloque de la Blockchain, votando mediante el uso de CPU. Para esto, los mineros proveen una prueba de que realizaron una cierta cantidad de trabajo para crear un bloque, que luego los demás nodos validan previo a agregar el bloque a su Blockchain. A esta prueba se la conoce como *Proof of Work (PoW)*. Los mineros tienen el incentivo de una recompensa en Bitcoins si su bloque forma parte de la cadena principal de la Blockchain.

El campo `nonce` del *header* del bloque contiene la Proof of Work. Su contenido es un número tal que **SHA256(SHA256(header))**[6], tiene una cantidad dada de 0's en sus bits más significativos. Esta cantidad es mayor o igual a un número que se deriva del campo *dificultad* del *header* del bloque.[7]

La motivación de este proceso es que obtener un hash que cumpla la propiedad pedida es computacionalmente costoso: cambiando un bit del input, el hash calculado cambia totalmente. Los hashes calculados están uniformemente distribuidos, por lo cual, la probabilidad de que un hash (256 bits) comience con d ceros, es $\frac{2^{256-d}-1}{2^{256}}$.

Para verificar que un bloque cumple con esta propiedad, alcanza con calcular dos veces el

[6] `SHA256` es una función de hash criptográfico.

[7] Técnicamente del campo dificultad es un número de 256 bits, el hash del bloque debe ser menor que este número. Decir que debe empezar con una cierta cantidad de ceros es una simplificación del proceso.

`SHA256` del *header*, mientras que la dificultad de armar un bloque que sea válido crece exponencialmente con la cantidad de 0's requerida.[8] De esta forma, se cuenta con un proceso que demuestra fácilmente que un minero hizo una cantidad razonable de trabajo para hallar un bloque.

Es importante destacar que una PoW solo sirve para un bloque, ya que esta depende del contenido del mismo. Si un bloque cambia cualquiera de sus campos (padre, transacciones, fecha), la PoW deja de ser válida. Además, como cada bloque tiene el hash del bloque padre, no es posible calcular la PoW con anticipación.

Debido a que el poder cómputo de la red puede fluctuar mucho con el ingreso y egreso de nodos, Bitcoin incorpora un sistema de *ajuste de dificultad*, que aumenta o disminuye la dificultad que deben cumplir los bloques para ser válidos. Para esto, la red define un *target*, que es el tiempo promedio en que deben aparecer nuevos bloques en la red (en Bitcoin es 10 minutos). Cada 2016 bloques, se cuenta cuántos de esos bloques[9] se construyeron en menos tiempo que el target, y en función de eso se aumenta o disminuye la dificultad hasta cuatro veces el valor anterior. Esto se traduce en que el hash del header del bloque debe comenzar con hasta dos ceros más o menos, según aumente o disminuya la dificultad, respectivamente.

Bitcoin provee ciertos incentivos para que los mineros colaboren con la red. Por un lado, el minero que construye un bloque se queda con todos los fees de las transacciones de ese bloque. Por otro lado, Bitcoin recompensa a los mineros, permitiéndoles ser los beneficiarios de la transacción *coinbase* del bloque, una transacción especial que crea Bitcoins de la nada. Esta es la única forma de crear Bitcoins, y es por eso que se conoce al proceso como «Minar Bitcoins», pues el minero en realidad está haciendo aparecer nuevos Bitcoins.

La transacción *coinbase* debe ser la primera transacción del bloque. Además, el minero tiene control total sobre los campos de esta transacción. El output de esta transacción sólo puede ser usado luego de que haya 100 bloques por encima del bloque minado. Esto se basa en evitar que se obtenga dinero a partir de bloques *stale*. Esta decisión de diseño es parte del protocolo

[8] Esto se debe a que cada 0 extra pedido disminuye a la mitad la cantidad de hashes que cumplan con esa restricción.

[9] Debido a un error de software, en Bitcoin solo se revisan los últimos 2015 bloques

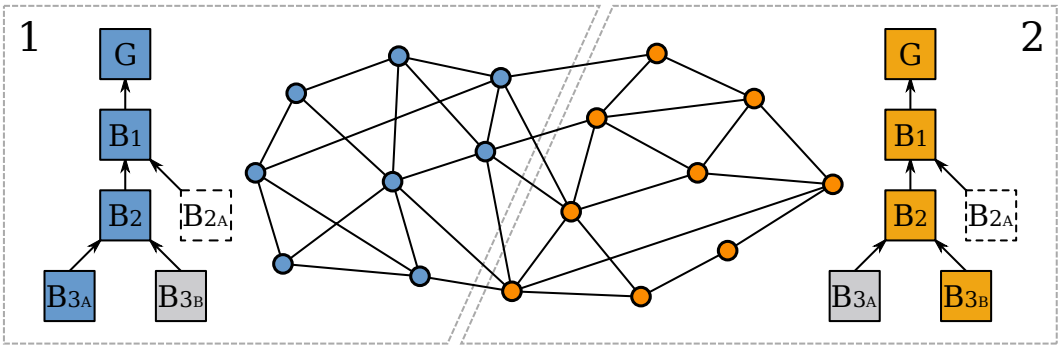


Fig. 1.7: Ejemplo de una partición de los nodos de la red. El grupo de nodos 1 considera al bloque B_{3A} como bloque de su cadena principal, mientras que el grupo 2 considera a B_{3B}. El bloque B_{2A} quedó *stale* luego de que se minaron los bloques B_{3A} y B_{3B}.

de consenso de la red.

La cantidad de Bitcoins otorgada por minar bloques disminuye en función del número de bloque. Cada 210000 bloques (4 años aproximadamente) se disminuye a la mitad. Comenzando con un valor de 50BTC por bloque. Esto da un total de, aproximadamente, 21 millones de Bitcoins. Al 8 de noviembre de 2016, la recompensa por hallar un bloque en Bitcoin es de 12,5BTC (cerca de US$8750).

Cuando un minero encuentra un nonce que hace válido al bloque, lo propaga por toda la red y los demás nodos lo validan, agregan a la Blockchain y lo propagan.[10]

Debido a la naturaleza distribuida de Bitcoin, es posible que cuando un minero encuentra un bloque y éste se propague al resto de la red, otro minero haya encontrado también un bloque válido a la misma altura dentro de la Blockchain.

En esta situación, los nodos de la red que reciben un bloque competitivo al bloque sobre el que están minando, tienen que decidir si ajustan o no su cadena principal. Esto es lo que se conoce como *Protocolo de Consenso*. En Bitcoin, ante dos ramas competitivas, gana la que tiene mayor dificultad total (suma de las dificultades de los bloques), y ante un empate, cada nodo se queda con la que recibió primero (*first-seen rule*).

Esta situación deriva en una partición en los nodos de la red, como se puede observar en la

[10] En la práctica no basta sólo con incrementar el nonce, ya que este campo sólo cuenta con 32 bits, es posible que se itere completamente y no se cumpla con la dificultad pedida. Se modifican otros campos también, como el tiempo del bloque y la transacción *coinbase*.

figura 1.7, donde un grupo de nodos tiene una cadena principal, y otro grupo de nodos otra. Cada grupo de nodos minará sobre la que considere su cadena principal. A esta situación se la conoce como un *fork* de la red. Eventualmente, sucederá que una de las dos cadenas tenga una dificultad total mayor (porque se minaron más bloques en esa cadena), y de a poco todos los mineros van a aceptar dicha cadena como su cadena principal. Luego, al minar sobre esta, tendrá mas dificultad y por lo tanto será la aceptada para toda la red.

Cuando un bloque es agregado a la Blockchain, cada bloque que se agregue por encima de él (es decir, que lo tenga como un ancestro), aumenta la probabilidad de que ese bloque pertenezca a la mejor cadena. Por ejemplo, en la red Bitcoin la divergencia más grande que ocurrió fue de cuatro bloques (sin contar los casos que se debieron a fallas de software), y se recomienda que para estar seguros de que el bloque va a pertenecer a la mejor cadena, se espere a que se minen por lo menos seis bloques sobre él (una hora considerando un *target* de diez minutos).

Esto indica que, para tener cierta garantía de no ser víctimas de un ataque de *double spending*, simplemente se debe esperar a que haya más bloques minados sobre el bloque en donde aparece la transacción de interés. Si un atacante quisiese revertir esa transacción, debería minar un bloque cuyo padre sea el mismo padre que el del bloque que la contiene y, además, crear tantos bloques como bloques hayan sido minados después. Es decir, el atacante tiene que controlar más de la mitad del poder de cómputo de la red para poder crear una cadena que sea más larga que la creada por los nodos honestos. A este ataque se lo conoce como un ataque del 51 % y, ante esta situación, no hay nada que se pueda hacer. Bitcoin tiene como premisa que más de la mitad de los nodos son **nodos honestos**.

En Bitcoin, realizar grandes cambios sobre los protocolos es realmente complicado: debido a la heterogeneidad de la red, lograr que todos los nodos actualicen su versión y estén de acuerdo con cambios implica un enorme esfuerzo. Si solo una parte de los nodos actualiza, y esta actualización cambia reglas del protocolo, se podría producir un *fork* en la red y separar en dos redes: una con los nodos que actualizaron y otra con los nodos que no actualizaron. A este tipo de fork se lo conoce como *hard fork*.

1.4. Protocolo e infraestructura

Los nodos de Bitcoin se comunican entre sí mediante un protocolo binario. Para poder saber a qué nodos conectarse, el cliente de Bitcoin cuenta con un archivo que contiene una serie de *«boot nodes»* a los que un nodo se puede conectar para pedir direcciones de otros nodos.

Los principales mensajes utilizados por los nodos de Bitcoin son: **GetAddr, Addr, Inv, GetData, GetHeaders, GetBlock, Block, Headers, Tx, Version y VerAck**.

`Version` y `VerAck` son los dos primeros mensajes intercambiados entre los nodos al conectarse, en el cual se informan qué versión del protocolo están usando, y cuál fue el último bloque recibido, entre otras cosas.

Un nodo puede pedirle a uno de sus pares una lista de conocidos para aumentar su cantidad de conexiones. Esto lo hace mediante el mensaje `GetAddr`, que pide una lista de nodos que se consideren activos, es decir, que hayan mandado un mensaje en las últimas tres horas. La respuesta a este mensaje es un mensaje `Addr` que contiene una lista de nodos activos elegidos al azar dentro de sus conocidos.

Cuando un nodo encuentra un bloque nuevo, este envía un mensaje de `Inv` a todos sus peers, informándoles del hash del bloque hallado. Estos le piden información del bloque mediante el mensaje `GetData`, y el nodo responde con un mensaje `Block`. Algo similar ocurre cuando un nodo recibe una transacción, la propaga mediante un mensaje `Inv`, y al recibir un `GetData`, envía la transacción en un mensaje `Tx`.

Cuando un nodo nuevo ingresa a la red, este debe sincronizar toda su Blockchain, traer todos los bloques que le faltan. Para ello, puede pedir varios bloques mediante los mensajes `GetBlocks` y `GetHeaders`.

En Bitcoin, los nodos tratan de mantener una lista de 8 peers activos en conexiones salientes, sin embargo, no tienen límite sobre la cantidad de conexiones que aceptan.

1.5. Ethereum

Hemos mencionado que, en Bitcoin, los outputs de las transacciones podían contener el hash de un script. Al momento de usar un input, se debía proveer el script correspondiente a ese hash y los datos necesarios para que se ejecute satisfactoriamente. Con esto, se pueden hacer condiciones de pago complejas, como requerir múltiples firmas digitales para poder utilizar el output de una transacción.

Los scripts que pueden ser ejecutados en Bitcoin tienen una limitación muy grande: no pueden contener ciclos ni saltos hacia atrás. Esto impide la creación de condiciones de pago más complejas, motivando el uso de *Smart Contracts*.

Ethereum es una criptomoneda que agrega contratos inteligentes *(Smart Contracts)* a las transacciones. Los contratos inteligentes permiten ejecutar código sin las restricciones de Bitcoin, permitiendo, además, llamar a funciones de otros contratos y crear nuevos.

Ethereum tiene la noción de cuentas, que es un concepto análogo a las direcciones en Bitcoin. El estado de las cuentas se actualiza luego de la ejecución de las transacciones de cada bloque.

Si bien Ethereum puede ejecutar transacciones más complejas que Bitcoin (con ciclos y llamadas recursivas), la ejecución de estas transacciones está acotada por el dinero total a disposición en las cuentas. Cada instrucción ejecutada, byte transferido o almacenado en memoria, tiene un costo, denominado *«gas»*. Cuando se crea una transacción se define el valor de los campos *«GasPrice»*, y *«GasLimit»*. *«GasLimit»* determina cuánto puede ejecutar esa transacción como máximo. *«GasPrice»* hace referencia a la conversión entre *«Ether»*, la moneda de Ethereum, y el *«gas»* usado por la transacción.

1.5.1. Protocolo, infraestructura y consenso

A diferencia de Bitcoin, cada nodo en Ethereum trata de tener 25 peers, obteniendo una red de menor diámetro, lo que facilita la propagación de mensajes. También ocurre que el tiempo entre bloques (target) en Ethereum es de quince segundos (a diferencia de los diez minutos de

Bitcoin).

El protocolo de envío de mensajes cuenta con varias versiones, siendo la más reciente la versión `ETH63`. Al iniciar una conexión entre dos nodos, éstos realizan un *«handshake»*, envíando un mensaje de `Status`, en el que comunican la máxima versión del protocolo que soportan.

Las diferencias entre esas versiones son mayormente sobre cómo sincronizar las Blockchains de forma más eficiente. El protocolo básico se detalla a continuación.

En Ethereum hay dos formas de notificar sobre un nuevo bloque: `NewBlockMessage` que envía un bloque entero o usando `NewBlockHashes` que envía sólo el hash del nuevo bloque.

Un nodo puede pedir el header de un bloque (o de varios), mediante el mensaje `GetBlockHeaders`, y varios bodies mediante el mensaje `GetBlockBodies`, estos mensajes son respondidos con `BlockHeaders` y `BlockBodies` respectivamente.

En Ethereum no hay un mecanismo para pedir transacciones, éstas sólo se envían mediante el mensaje `Transaction`, que se propaga por la red.

El mecanismo de propagación de bloques es diferente que en Bitcoin. Cuando un nodo recibe un `NewBlockMessage`, éste lo valida y agrega a su Blockchain, y lo reenvía a la raíz cuadrada de sus peers. Al resto de los peers les envía un mensaje de `NewBlockHashes`. La motivación, detrás de esta política, es que el bloque se propague rápido por la red sin sobrecargarla mandando bloques a todos los nodos.

Como prueba de trabajo, Ethereum usa `SHA3`, sumado a técnicas para evitar que sea eficiente minar usando hardware dedicado. Este método no es compatible con el hardware de minado de Bitcoin.

Como el tiempo de generación de bloques es considerablemente más corto en Ethereum que en Bitcoin, la probabilidad de que se generen bloques competitivos es mucho mayor, más allá de que mejore el tiempo de propagación. Es por esto que en Ethereum adoptan el protocolo de consenso `GHOST`. Este protocolo incluye la noción de **uncles**.

Los mineros, al armar un bloque para minar, tienen un campo en el que pueden hacer referencia

a uno de los bloques *stale*. Ethereum recompensa tanto al que mina el bloque como al bloque *stale* que es referenciado, este último será un bloque *uncle* del bloque minado. La motivación de este mecanismo de consenso es que a todos los nodos les convenga seguirlo, aunque eso implique dejar de minar sobre su propio bloque.

La dificultad de un bloque con uncles está compuesta por la dificultad del bloque y la suma de dificultades de los bloques uncles. De esta forma, si un bloque hace referencia a uncles, tiene mayor probabilidad de quedar en la cadena principal.

Esto fue una breve descripción de la criptomoneda Ethereum, y sus similitudes y diferencias con Bitcoin. Presentamos Ethereum meramente como una introducción y motivación para RSK, que se basa en esta criptomoneda.

1.6. RSK

Son indudables los beneficios de los *smart contracts*, dando un gran nivel de expresividad y control sobre el dinero digital. La comunidad de Bitcoin muestra gran interés por incluir una funcionalidad semejante a la de Ethereum, pero sería un cambio muy grande para hacer sobre Bitcoin.

RSK es una criptomoneda que propone acercar la funcionalidad de *smart contracts* provista por Ethereum a Bitcoin, haciendo uso de ideas nuevas en criptomonedas.

En primer lugar, RSK es una criptomoneda que no genera un subsidio: la recompensa por minar bloques se desprende únicamente de los fees de las transacciones, no se crea una moneda de la nada como ocurre en Bitcoin o Ethereum. La forma de conseguir monedas de RSK es mediante un sistema transparente de intercambio de dinero entre las Blockchains de RSK y Bitcoin, denominado *2-Way Peg*. Este cambio de dinero se realiza de forma automática utilizando nuevos *opcodes* de scripts en Bitcoin propuestos por RSK, y mediante *Smart Contracts* del lado de RSK. Los detalles de este mecanismo son bastante complejos y exceden al alcance de este trabajo.

Como en RSK no se generan nuevas monedas, salvo transfiriendo de Bitcoin, la relación entre

el precio de RSK y Bitcoin se mantiene constante. Esto ofrece la posibilidad de transferir fondos libremente a RSK, correr *smart contracts*, y cuando se desee, retirar dichos fondos para transferirlos de regreso a Bitcoin.

Por otro lado, RSK aprovecha el hecho de que existe una gran inversión de dinero en hardware específico para minar en Bitcoin, facilitando la adopción. Por un lado obtiene poder de cómputo de forma sencilla y, por el otro, evita la necesidad de reinvertir dinero en hardware especial para minar RSK. Esto se logra mediante un proceso conocido como *merged-mining* que será detallado más adelante. Básicamente, consiste en reutilizar los bloques de Bitcoin para proveer PoW para bloques de RSK.

Con estas dos funcionalidades, RSK se acopla a Bitcoin permitiendo el libre movimiento entre Blockchains.

En lo que a software respecta, el cliente de RSK está basado en el de Ethereum, por lo cual muchas funcionalidades son parecidas, como la compatibilidad de los contratos o la topología de la red.

1.6.1. Protocolo y consenso

En lo que respecta al protocolo de la red, RSK tiene algunas modificaciones con respecto a Ethereum. No cuenta con toda la funcionalidad provista por `ETH63`, ya que provee solamente los siguientes mensajes:

- **Block** que envía un bloque completo, header y body.
- **GetBlock** que pide un bloque particular.
- **NewBlockHashes** que envía el hash de un bloque nuevo.
- **GetBlockHeaders** que pide los headers de algunos bloques.
- **BlockHeadersMsg** en respuesta a **GetBlockHeaders**.
- **Transaction** que envía una transacción completa.

Para la primer versión de RSK, se decidieron implementar esos mensajes para tener un protocolo simple, pero a futuro se espera poder agregar mejoras que permitan optimizar el tiempo de propagación, para así bajar el tiempo entre bloques sin aumentar la cantidad de forks.

Con respecto al protocolo de consenso, RSK actualmente tiene un sistema de uncles como Ethereum, pero sin dar recompensas por minarlos. En un futuro se espera poder implementar el protocolo de consenso `DECOR+`, que se adapta mejor a la política sin subsidio de RSK.

El software de RSK (RootstockJ) está basado en la versión de Java de Ethereum (EthereumJ), la cual a diferencia de implementaciones en otros lenguajes, carece de ciertas funcionalidades, necesarias para la realización de los experimentos propuestos en este trabajo. Entre los cambios realizados, cabe destacar los siguientes:

- Se reescribió el módulo que maneja los mensajes del protocolo de red, implementando la funcionalidad de cada uno de los mensajes aquí mencionados.
- Se arregló gran parte del código que mantiene el estado de los peers de un nodo, pues ocurrían desconexiones esporádicas.
- Se reescribió el código que maneja el servidor de minado, ya que este tenía problemas de concurrencia.
- Se implementó un cliente de minado simulado, del cual se hablará más adelante.
- Se reescribió el módulo encargado de manejar la Blockchain.

1.6.2. Merged mining

En RSK no existe el minado de bloques como en Bitcoin. Se utiliza un proceso llamado *merged-mining*, que permite aprovechar el minado en otras Blockchains para dar *proofs of work* para RSK.

La idea principal es que los mineros además de minar bloques en Bitcoin, minen bloques RSK sin dejar de minar Bitcoin. Recordemos que un bloque es aceptado si éste cuenta con un *Proof of Work* válido para una dificultad dada. Si un bloque de Bitcoin, en alguno de sus campos,

hace referencia a un hash de un bloque de RSK, la prueba de trabajo de ese bloque es válida para el bloque de RSK también, ya que no sería sencillo forjar un bloque de Bitcoin que haga referencia al bloque de RSK y además cumpla con el requisito de dificultad.

Entonces, si un minero desea realizar merged-mining entre Bitcoin y RSK, basta con que agregue el hash del bloque de RSK en algún campo del bloque de Bitcoin. El campo elegido para esto es uno de los inputs de la transacción coinbase (la transacción que recompensa al minero por minar el bloque). Dado que el minero tiene control sobre esa transacción, y que puede agregarle datos sin perder dinero, agrega sobre el final de esa transacción, un tag especial que hace referencia a RSK y el hash del bloque.

Luego, dado que la dificultad del bloque Bitcoin siempre será mayor a la dificultad del bloque de RSK[11], al computar hashes de los bloques, hay solo tres posibilidades:

i) El hash no cumple la dificultad del bloque de RSK ni la dificultad del bloque de Bitcoin.

ii) El hash cumple la dificultad del bloque de RSK, pero no con la del bloque Bitcoin.

iii) El hash cumple con la dificultad de Bitcoin y con la del bloque RSK.

En el primer caso, el minero sigue calculando hashes, ya que el bloque todavía no es válido para ninguna de las dos Blockchains. En el segundo caso, el bloque no le sirve para Bitcoin, pero sí es un bloque válido para RSK, por lo cual, lo envía a la Blockchain de RSK para que sea agregado. Finalmente, en el último caso, el minero envía el bloque a Bitcoin y a RSK, obteniendo ganancia por ambas partes.

El incentivo para minar usando merged-mining es que, sin perder performance en el minado de Bitcoin, se puede obtener la recompensa para el bloque de la otra Blockchain. Desde el punto de vista de RSK, al tener merged-mining se facilita la incorporación de nuevos mineros, ya que los mismos no tienen costo de oportunidad alguno para sumarse a la red de RSK.

[11] En Bitcoin el tiempo entre bloques es de 10 minutos, mientras que en RSK es 10 segundos, de por sí la dificultad de Bitcoin es mucho mayor, pero además, los mineros de RSK son un subconjunto de los mineros de Bitcoin, así que el poder de cómputo en RSK también será menor.

Validación de bloques por Merged-Mining (SPV Proof)

Para validar que un bloque minado por merged-mining hace referencia a un bloque RSK, hacen falta las siguientes comprobaciones:

- Que el bloque Bitcoin cumpla con la dificultad pedida por RSK para ese bloque.
- Que efectivamente se encuentre el hash del bloque RSK en el input de la transacción coinbase.
- Que no se haga referencia más de un bloque RSK.

Con esto, para validar el bloque RSK se podría contar con el bloque Bitcoin entero, pero ocuparía mucha memoria: cada bloque Bitcoin puede ocupar hasta 1MB, y esto sería usado sólo para la validación.

Sin embargo, para validar la dificultad de un bloque Bitcoin, basta con analizar sólo el header del bloque, por lo cual esto nos bastaría para cumplir esa restricción. En el header, además, se encuentra el hash de la raíz del Merkle Tree que contiene las transacciones del bloque. Con este Merkle Tree se pueden validar las transacciones que están en el cuerpo del bloque, calculando todo el árbol binario de hashes hasta la raíz.

Un ejemplo de Merkle Tree se puede ver en la figura 1.8. Para verificar que un elemento está en árbol, basta con hacer todo el camino de hashes desde ese subárbol hasta la raíz. Siguiendo con el ejemplo de la figura, para verificar que L1 está en el árbol, habría que calcular el hash de L1 (Hash 0-0), luego el hash de Hash 0-0 y Hash 1-0 (Hash 0), y finalmente el hash de Hash 0 y Hash 1 (Top Hash). De esta forma, teniendo solo L1, Hash 0-1 y Hash 1, es posible verificar que L1 pertenece al bloque referenciado por el header dado. Esto es, log_2(#transacciones) hashes + L1.

L1 sería la transacción coinbase, también puede ocupar mucho tamaño (más de 1KB de datos). Pero no hace falta tenerlo completo, solo interesa poder calcular su hash, y saber que referencia al bloque de RSK. Es por eso que el hash de RSK se encuentra al final del último input, posicionándolo en los últimos 128 bytes de la transacción. De esta forma, se puede «comprimir»

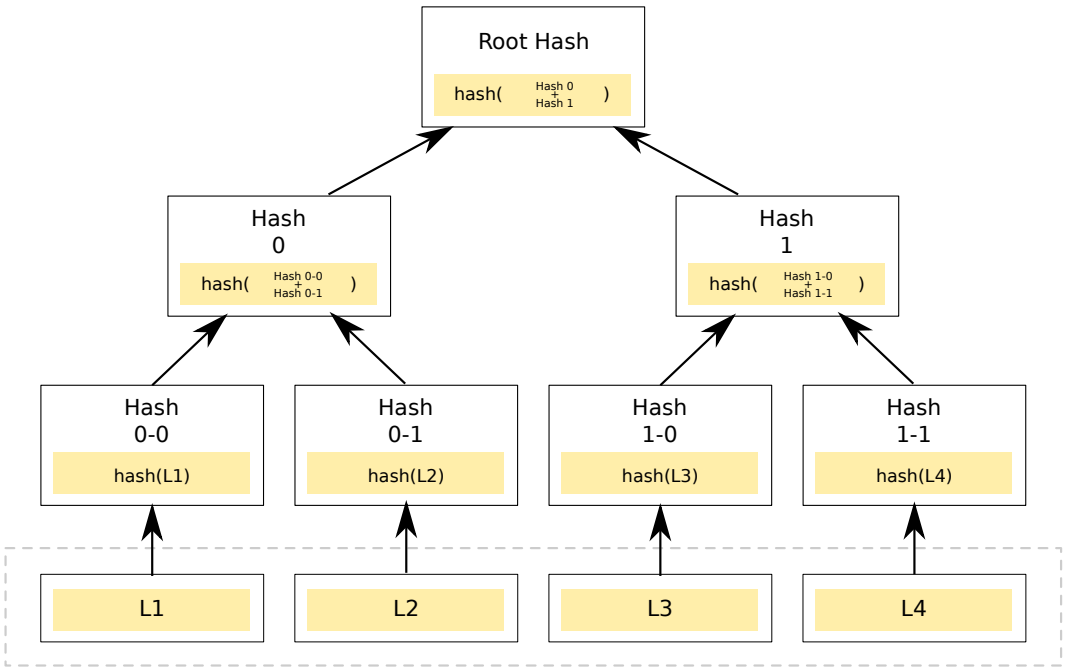


Fig. 1.8: Un árbol binario de hashes (Merkle Tree). Cada nodo tiene el hash de sus hijos concatenados, salvo las hojas.

la transacción coinbase usando el «mid-state» del `SHA256`[12], dejando sin procesar la última parte.

Entonces, el bloque de RSK para validarse, debe tener además:

- El header del bloque de Bitcoin. Para verificar la dificultad y tener la raíz del Merkle Tree
- El mid-state del `SHA256` de la primer parte de la transacción coinbase, dejando sin procesar la última parte que incluye los 128 bytes en donde se encuentra el hash del bloque RSK.
- La última parte de la transacción coinbase, que incluye los 128 bytes finales, más a lo sumo 63 bytes que hayan quedado sin poder procesarse por el algoritmo `SHA256`.
- Los hashes del Merkle Tree necesarios para poder llegar desde la hoja de la coinbase hasta la raíz del Merkle Tree.

[12] SHA256 procesa el input de a bloques, el mid-state de `SHA256` es el estado interno de todas las variables en un momento dado, de forma tal que se pueda frenar el proceso y continuar más adelante desde ese estado sin tener que volver a procesar el input ya visto

A estos campos se los conoce como *«Prueba SPV»*. Y para validar un bloque, un nodo debe:

- Validar la dificultad del header del bloque.
- Validar que esté el hash del bloque de RSK.
- Terminar de calcular el hash de la transacción coinbase partiendo del mid-state y agregando lo que quedó sin procesar.
- «Subir» por el arbol de hashes hasta llegar a la raíz y ver que esta es igual a la que se encuentra en el header del bloque.

1.7. Pools de minería

Para muchos mineros, el sistema de recompensas de Bitcoin es injusto: solo recibe recompensa quien encuentra un bloque, y no todos los que aportaron poder de cómputo para hallarlo. Es por esto que surgió el sistema de *pools de minería*. Un pool de minería agrupa a un conjunto de mineros, comportándose como un único minero de caras al resto de la red, pero internamente, divide las recompensas ganadas teniendo en cuenta cuánto aportó cada minero del pool a encontrar cada bloque. Los mineros ni siquiera tienen que estar conectados a la red real, solo es necesario que se comuniquen con el sistema del pool de minería. En la figura 1.9 se puede ver un esquema de cómo está organizado un pool de minería.

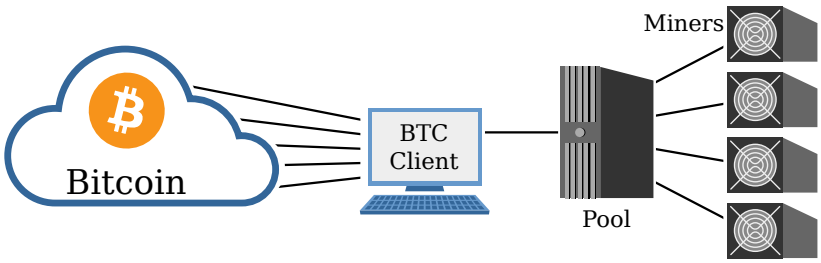


Fig. 1.9: Ejemplo de estructura de un pool de minería. El pool se conecta a un nodo de Bitcoin y varios mineros se conectan a este pool para recibir ordenes.

En líneas generales, la forma en que los pools cuantifican el trabajo realizado por los mineros es la siguiente: el sistema del pool de minería es quien genera el bloque para los mineros, completando todos los campos (incluso la transacción coinbase, para que la recompensa vaya

a parar al pool de minería). Los mineros luego sólo tienen que preocuparse por modificar los campos que cambian el hash del bloque para poder encontrar un bloque que cumpla con la dificultad requerida por el pool de minería.

El pool define una dificultad menor a la dificultad real del bloque, haciendo que efectivamente los mineros envíen soluciones que no necesariamente son soluciones para el bloque real. A estas soluciones parciales se las conoce como *shares*. Esto permite a los pools de minería estimar cuánto aportó cada minero para hallar un bloque, considerando la cantidad de *shares* que reportaron.

Cuando el pool de minería recibe un *share* que satisface la dificultad real del bloque, este lo envia al resto de la red (como si fuese un bloque nuevo), y al cobrar la recompensa, la distribuye entre los mineros que aportaron *shares* para dicho bloque.

Para la interacción con los mineros, muchos pools de minería utilizan el protocolo *Stratum* [slu]. Cada vez que un nuevo mejor bloque aparece en la red, el pool les avisa a todos los mineros mediante un mensaje de *MiningNotify* con el parámetro *cleanJobs* seteado en Verdadero. Esto hace que todos los mineros descarten el trabajo que estén haciendo y empiecen de cero con el nuevo trabajo (ya que al haber un nuevo mejor bloque, deberían empezar a minar sobre este).

Debido a que en promedio, se tardan 10 minutos en hallar un nuevo bloque en Bitcoin, los pools pueden decidir realizar cambios en los bloques que le envían a los mineros, agregando o cambiando las transacciones para maximizar la ganancia obtenida mediante fees. Esta notificación también ocurre periódicamente mediante un *MiningNotify*, pero con el parámetro *cleanJobs* en seteado en Falso, ya que no se requiere que los mineros cancelen todo lo que estén haciendo.

Merged mining con pools de minería en RSK

Al hacer merged-mining entre dos Blockchains (BTC y RSK), los pools deben agregarle al bloque de Bitcoin para minar, el hash del bloque de RSK. Para ello, necesitan conectarse tanto a un nodo Bitcoin como a un nodo RSK. Se puede ver una esquematización de esta

estructura en la figura 1.10.

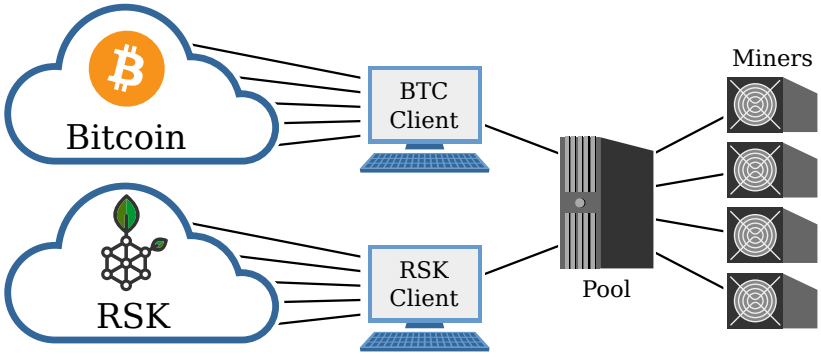


Fig. 1.10: Ejemplo de estructura de un pool de minería conectado a dos Blockchains distintas (BTC y RSK). El pool se conecta a un nodo de Bitcoin y a un nodo RSK, y varios mineros se conectan a este pool para recibir ordenes.

Es importante notar que para incentivar la adopción del sistema de merged-mining entre los mineros, se desea garantizar que el minado de Bitcoin no se ve perjudicado por realizar merged-mining. Sin embargo, al agregar una nueva Blockchain, se genera una nueva fuente de bloques y actualizaciones: Ahora cada vez que llega un nuevo bloque de RSK el pool de minería debe evaluar cómo le va a avisar a los mineros.

Los bloques de Bitcoin ocurren cada 10 minutos en promedio, mientras que los de RSK ocurren cada 10 segundos en promedio. Esto significa que cada 10 segundos aproximadamente, el pool de minería debe estar actualizándoles el trabajo a los mineros. Esta actualización se realiza mediante *MiningNotify*, pero ahora el efecto del parámetro *cleanJobs* cambia bastante el resultado.

Si se envía *cleanJobs* seteado en Verdadero, los mineros descartarían el trabajo que vienen haciendo para Bitcoin, lo cual resulta en una disminución de los shares reportados.

Por otro lado, si *cleanJobs* se envía en Falso, los mineros actualizarán su trabajo cuando lo consideren necesario. Qué tan frecuentemente los mineros actualizan el trabajo depende del hardware utilizado y de la configuración del software. El efecto que esto causa para los bloques de RSK es que se minen bloques viejos, que van a terminar siendo uncles y no formando parte de la cadena principal. Esta política no va a tener ningún efecto para Bitcoin.

Las pruebas experimentales con un minero *AntMiner S4* ([ant]) mostraron que estos actuali-

zaban su trabajo cada 30 segundos. En el capítulo 4 se analizará el efecto que tendría tener todos mineros de ese tipo en la red.

2. TRABAJO RELACIONADO

Gran parte del análisis que se realiza en Bitcoin para descubrir la topología y propiedades de la red, se realiza utilizando clientes instrumentados, que se conectan a varios nodos de la red y recolectan información sobre sus pares.

Decker y Wattenhofer en [DW13] utilizan un cliente instrumentado para analizar el tiempo de propagación de bloques y generación de forks en la red. Su cliente se conecta a una gran cantidad de nodos de forma pasiva, es decir, que no reenvía los mensajes que recibe de los nodos a los que está conectado. Gracias a la información recopilada, analizan el comportamiento de la red durante 10000 bloques, obteniendo los siguientes resultados:

- El tiempo de propagación promedio de bloque en Bitcoin es de 12,6 segundos.
- Estiman la frecuencia de generación de forks en función del tiempo de propagación de bloques
- Hallan una correlación fuerte entre el tiempo de propagación de los bloques y el tamaño de los mismos.

Proponen cambios para mejorar el tiempo de propagación de bloques: tener mayor conectividad entre nodos, propagar las notificaciones de nuevo bloque antes de recibir el bloque entero y ser menos estrictos en la verificación de bloques antes de propagarlos. Tomaron nuevas mediciones con estos cambios y obtuvieron mejoras en los tiempos de propagación. Este trabajo es uno de los primeros acerca del efecto del tiempo de propagación en la generación de forks en la red.

Por otro lado, Donet et al. en [DPSHJ14] también mide el tiempo de propagación de bloques, pero haciendo foco en la topología de la red, busca calcular la distribución de IPs de nodos Bitcoin por país. Para ello, su cliente envió periódicamente mensajes de `GetAddr` a fin de obtener las IPs de nodos de la red, realizando consultas recursivas y obteniendo nuevos nodos en cada paso. Este experimento se realizó durante un período de más de un mes. Para el

análisis de los tiempos de propagación, sólo tomaron mediciones de 710 de bloques (cinco días).

Los resultados de este trabajo fueron:

- Si bien hay una correlación entre el tamaño de bloque y el tiempo de propagación, esta correlación no es lineal.
- Reportan la métrica de qué porcentaje de bloques se propagó antes de un tiempo dado a un porcentaje de nodos.
- Encuentran 87000 IPs distintas de nodos de Bitcoin, pero solo 6000 IPs se mantuvieron constantes a lo largo del experimento.

En el trabajo, publican la distribución de IPs por país, que luego utilizaremos para armar la topología de nuestros experimentos.

Miller et al [MLP+15] crearon *Coinscope*, una herramienta para mejorar la caracterización de la distribución de nodos en la red. Ésta utiliza metadatos provistos por la respuesta de `GetAddr` y diferencia conexiones entrantes de salientes. Por otro lado, busca detectar qué nodos de la red son los más «influyentes». Para esto particiona la red y envía transacciones conflictivas a distintos nodos de la red. Luego, analiza qué transacciones terminan apareciendo en un bloque minado.

Los resultados obtenidos fueron que, en general, los nodos tienen aproximadamente ocho peers, lo que se coincide con el cliente oficial de Bitcoin, sin embargo, hay algunos que tienen más de 100 peers. Los autores indican que estos últimos podrían ser pools de minería.

Si bien la red se asemeja un grafo aleatorio, el trabajo menciona una prueba de «comunidades» (Louvain) sobre los datos obtenidos. La prueba separa los nodos en n conjuntos, $A_1, \ldots, A_n$, que cumplen la propiedad de que $\forall A_i,\ i \in [1, n]$ vale que $\forall x \in A_i,\ \forall j \in [1, n],\ j \neq i,\ \#peers(x,\ A_i) > \#peers(x,\ A_j)$, donde $\#peers(x,\ A)$ es la cantidad de *peers* que tiene x en el conjunto A.

Usando Coinscope toman muestras de la topología de la red, contando las comunidades y las

comparan con la cantidad de comunidades de grafos aleatorios, el resultado es que la media de estas métricas se encuentra a 2 desvíos estándares de distancia de la media de las métricas en grafos aleatorios. Los autores sugieren que esto se debe a que los nodos se empiezan a conectar a partir de *«bootnodes»* y de a poco se van expandiendo.

Con respecto a los nodos influyentes, el 2 % de los nodos es responsable de $\frac{3}{4}$ de las transacciones que aparecen en los bloques: si se le envía una transacción a esos nodos, es muy probable que aparezca en un bloque, por más que haya otra transacción conflictiva en la red. Esto sirve para detectar pools de minería, ya que estos tienen gran poder de cómputo y serían nodos influyentes.

Croman et al. en [CDE+16] analiza distintos problemas de escalabilidad en Bitcoin. Parte desde distintos enfoques como protocolos de red, *«throughput»* de transacciones, consenso y almacenamiento de datos. Detalla que uno de los cuellos de botella es el tiempo de propagación de bloques, sugiriendo como mejora posible, disminuir el diámetro de la red. Sin embargo, concluye que se requiere un rediseño de los protocolos y procesos como única solución viable a largo plazo.

Sompolinsky y Zohar en [SZ13] analizan el throughput de transacciones en Bitcoin y obtienen una cota superior para este valor, regida por el diseño del protocolo de consenso. Presentan como solución el protocolo GHOST (que es usado por Ethereum), el cual introduce el concepto de *«bloques uncles»* para disminuir la cantidad de forks y mejorar el *«throughput»*.

En septiembre de 2016 fue aprobado el Bitcoin Improvement Proposal 512, una propuesta para mejorar el tiempo de propagación de bloques enviando menos información. La solución aprobada se denomina «Compact Blocks» ([cmp]), dejando rechazada a la propuesta de utilizar «XThin Blocks» ([xth]).

«XThin Blocks» trata de evitar el problema de enviar dos veces las transacciones de un bloque: una vez cuando se reciben y otra vez junto con el bloque entero. Para esto, en respuesta al mensaje `Inv` envían `GetData` con un «Bloom Filter» que contiene información sobre las transacciones que tienen. Al momento de enviar el bloque, se envían solo las transacciones que

no aparecen en ese Bloom Filter. Se presenta un análisis instrumentando nodos en distintos lugares del mundo, haciendo énfasis en el efecto que tiene el gran firewall de China en los tiempos de propagación de mensajes ya que gran parte del hashing power se encuentra detrás de este firewall.

«Compact Blocks» reduce significativamente el tiempo de propagación. En lugar de enviar un mensaje de `Inv` se envía un bloque compacto, que contiene el header del bloque (80 bytes), un prefijo de los hashes de las transacciones del bloque lo suficientemente largo como para que el nodo que lo recibe pueda identificarlas de forma unívoca, y las transacciones enteras que estima que el nodo receptor no tiene. Para esto, cada nodo debe llevar constancia de qué transacciones cree que tienen sus pares.

Existen también numerosos análisis teóricos sobre el protocolo de Bitcoin y su seguridad. Uno de los más relevantes, es el de Garay et al [GKL15] en donde se analiza la seguridad del protocolo al tener una menor relación entre tiempo de generación de bloques y tiempo de propagación, entre otras cosas. En [KP15] y [KP16] se analiza la seguridad de blockchains con intervalos muy cortos entre bloques, incluyendo el efecto del protocolo de consenso GHOST usado por Ethereum.

En el aspecto de las simulaciones, Miller y Hopper en [JH12] presentan Shadow, un simulador de eventos discretos para Bitcoin, logrando ejecutar el código de Bitcoin nativo pero reemplazando las llamadas al sistema operativo para las operaciones de red y de tiempo, permitiendo así simular redes con distintas propiedades. Este simulador fue diseñado originalmente para simular la red de Tor, y luego se modificó para Bitcoin. El principal resultado de este trabajo, fue que hallaron un posible ataque de denegación de servicio sobre el cliente oficial de Bitcoin. En el trabajo, utilizan una topología de red obtenida por la aplicación Coinscope (mencionada previamente), complementándola con información sobre latencia a partir de un dataset compaginado con información de CAIDA y NetIndex. En esta tesis se utilizará un modelo de red basado en estas ideas.

Fadhil et al. en [FOAa] proponen un método para capturar información de latencias entre nodos y tiempos de propagación para armar un simulador de eventos discretos: con un cliente

de Bitcoin modificado se conectan a varios nodos y capturan la latencia desde su cliente hacia estos nodos. A partir de esa información arman la topología de su simulación. En [FOAb] muestran que bajo esta topología, en su simulación, el tiempo de propagación se puede mejorar destinando algunos nodos como superpeers, es decir, nodos con muchas conexiones. Este resultado coincide con lo propuesto por Decker y Wattenhofer.

Gervais et al. en [GKW+] diseñan un simulador sobre `ns-3` para analizar la seguridad de la red de Bitcoin ante variaciones en parámetros de la red, como tiempo entre bloques o el algoritmo de consenso. Se modelan las Blockchains y protocolos de Bitcoin, Ethereum y otras criptomonedas, como así también algunos ataques conocidos. Este trabajo tiene foco sobre la seguridad, mostrando que, por ejemplo, Ethereum requiere 37 bloques en la mejor cadena para lograr la misma seguridad que logra Bitcoin con solo 6, suponiendo que existe un atacante que controla un 30 % del poder de cómputo de la red.

Todos estos trabajos muestran el interés de la comunidad por entender y mejorar la eficiencia de las criptomonedas. En esta tesis se utilizarán varias de las ideas mencionadas y de los datos provistos en estas publicaciones.

3. METODOLOGÍA

Este capítulo detalla la metodología utilizada para la realización de experimentos. Se divide en dos grandes partes: la primera describe las herramientas que fueron consideradas y las que terminaron utilizándose. La segunda parte entra en detalle sobre los pormenores que se tuvieron en cuenta para la toma y análisis de datos.

3.1. Herramientas

En general, para realizar pruebas sobre sistemas distribuidos de gran escala como Bitcoin, se suelen plantear dos caminos: o bien se realizan simulaciones sobre modelos simplificados, como ser un simulador de eventos discretos que capture una dinámica particular del sistema, o bien se utilizan modelos más realistas sobre entornos controlados.

En este trabajo, se trata de analizar la dinámica general del sistema con un fuerte enfoque en el comportamiento de la red, por lo cual es necesario tener una buena simulación de la misma. También se busca que el código ejecutado sea lo más cercano al código real posible.

Estas necesidades se tuvieron en cuenta al momento de elegir las herramientas a utilizar: por un lado, poder ejecutar el cliente de RootstockJ sin mayores modificaciones, sin tener que trabajar sobre un modelo simplificado del sistema, lo cual tiene la ventaja de poder evaluar cambios en la plataforma en producción con mayor facilidad. Por otro lado, la red resultante debería acercarse lo más posible al comportamiento real del sistema.

Se contempló la posibilidad de utilizar un simulador de eventos discretos para este trabajo, las opciones consideradas se detallan a continuación.

3.1.1. Simuladores de eventos discretos

El simulador de eventos discretos `ns-3` permite simular con gran precisión el comportamiento de red de un sistema distribuido, sin embargo, no hay mucho trabajo hecho sobre cómo ejecutar código `Java` desde `ns-3`, tampoco es sencillo ejecutar código nativo. Para poder utilizar `ns-3` hubiera sido necesario reescribir un modelo simplificado del sistema en `C++`, lo que implicaría no correr el código real de RootstockJ.

Por otro lado, las simulaciones en `ns-3` son costosas ya que su modelo de simulación es muy detallado. Simular un sistema tan grande como Bitcoin ($\sim$ 6000 nodos), podría ser impracticable en términos de tiempo real, pues un segundo de tiempo de simulación podría insumir varios minutos de tiempo real.

`SimGrid` es otro simulador de eventos discretos, orientado a simular sistemas de gran escala, con un simplificado modelo de red que puede ser configurado según las necesidades del experimento. `SimGrid` tiene una interfaz que permite ejecutar código en `Java`, teniendo que reemplazar las primitivas de envío de paquetes por red por las que provee `SimGrid`. El resultado de utilizar `SimGrid` fue sumamente prometedor: se logró correr `SimGrid` con el mismo código del cliente RootstockJ haciendo mínimos cambios, sin embargo, el modelo de red usado no resultó ser muy realista. En general, `SimGrid` requiere gran nivel de ajuste manual de parámetros hasta lograr un sistema realista. Por cuestiones de tiempo y alcance se descartó usar `SimGrid` para este trabajo, pero es un prometedor camino a seguir en trabajos futuros.

Luego de analizar estas herramientas, se determinó que éstas no llegan a cumplir con los requerimientos necesarios. Si bien `ns-3` provee una simulación detallada de la red, no permite correr el código real, e implicaría tener que reescribir gran parte del mismo. `SimGrid`, por el contrario, permite correr el código de RoostockJ con modificaciones menores, pero no se logró ajustar el modelo de red a nuestras necesidades.

Finalmente, se optó por no utilizar un simulador de eventos discretos, sino contar con una red definida por software, en la cual se puedan ejecutar instancias del cliente RootstockJ. Estas redes definidas por software permiten ajustar parámetros como latencia y conectividad entre

nodos, algo que no es tan fácil de realizar en redes reales.

3.1.2. Redes definidas por software

Mininet

La aplicación «Mininet», creada por Lantz, Hellen y McKeown en [LHM10] permite simular una red de mediano tamaño dentro de una sola computadora. Esta aplicación permite definir una red usando hosts, switches y enlaces que los conecten.

Mininet aísla los elementos de la red usando *«namespaces»* del sistema operativo Linux. Estos namespaces permiten aislar y virtualizar recursos del sistema para conjuntos de procesos. Por ejemplo, se podría tener dos procesos en namespaces distintos y que cada uno de ellos no pueda saber de la existencia del otro. Por cada host y switch del sistema, Mininet crea un namespace de red, cada uno de estos tiene su propio stack de red. Al agregar un enlace entre dos elementos de la red, se crea una interfaz de red nueva en los dos namespaces, y dos interfaces de red virtuales en el namespace global que las une. A cada enlace se le pueden cambiar propiedades como ancho de banda, latencia, `jitter` (varianza de la latencia) y porcentaje de paquetes perdidos. Esto lo hace Mininet mediante el comando `netem` que brinda el subsistema de control de tráfico de Linux.

En definitiva, Mininet orquesta funcionalidades que brinda Linux para poder crear redes emuladas de forma sencilla.

Cada nodo y switch de la red debe saber cómo redireccionar los paquetes que recibe. Para esto, Mininet delega la responsabilidad en un controlador de red. Este controlador les define a los switches *«flujos»* con información sobre cómo reenviar sus paquetes. Para esto, el controlador recibe cada paquete para los cuales los switches no tienen un flujo ya definido.

Si bien en Mininet cada host puede correr aplicaciones nativas sin ningún tipo de virtualización, la gran limitación de Mininet es que corre sobre una sola computadora. Debido a esto, no es una opción válida como herramienta para simular redes de gran tamaño. En particular, cada

nodo RSK corre sobre la máquina virtual de `Java` y requiere una gran cantidad de memoria. Esto es cierto no solo para RSK si no también para muchas aplicaciones no triviales.

Como solución a esta limitación, el equipo de desarrollo de Mininet comenzó a trabajar en «Mininet Cluster», que permite distribuir una red de Mininet en un cluster homogéneo. Lamentablemente, al momento de realizar este trabajo, esta aplicación todavía se encuentra incompleta e inestable.

Maxinet

Maxinet fue creada en Julio del 2014 por Wette et al. [WDS14], y surge como una solución que permite distribuir una topología de Mininet en varias computadoras. Para ello, utiliza `METIS` [KK95] para particionar una topología de Mininet en varias topologías más pequeñas que correrán en distintas máquinas.

Cada una de las máquinas corre una instancia de Mininet con la subtopología que le corresponde.

Para conectar switches que se encuentran entre máquinas distintas, pero que en la topología original compartían un enlace, Maxinet crea túneles `GRE` de Linux entre esas máquinas. Estos túneles encapsulan tráfico entre dos máquinas sobre `IP`.

Maxinet tiene la ventaja de que permite escalar horizontalmente la red, usando computadoras de baja gama para simular experimentos grandes. Por otro lado, Maxinet es relativamente nuevo y no cuenta con el mismo uso y análisis que Mininet. Por esto último, se decidió realizar un experimento comparando Mininet y Maxinet, para ver si la simulación de Maxinet era realista. Los resultados de estas pruebas serán presentadas más adelante, en el capítulo 4.

Límites de Mininet y Maxinet

Tanto Mininet como Maxinet dependen de un software que se llama «Controlador de Red», que se encarga de administrar las rutas de los switches de la red. Por ser un elemento crítico

en la simulación, se probaron varios controladores distintos a fin de encontrar el que mejor se adapte a las necesidades de los experimentos. Todos los controladores evaluados resultaron ser un cuello de botella para la simulación de la red. Se terminó optando por floodlight [flo], que fue el que mejor rendimiento mostró en nuestra plataforma. Uno de los problemas principales de este esquema de red, es que los switches demoraban un tiempo no despreciable en aprender las rutas que debían tomar, y estas rutas variaban, pues eran redefinidas por el controlador. Por esto, la cantidad de mensajes al controlador excede la capacidad que puede procesar, lo que genera que termine descartando muchos paquetes. Se pueden usar varios controladores, pero ni Mininet ni Maxinet proveen una forma sencilla de hacerlo.

Dado que en nuestros experimentos la topología usada no cambia a lo largo de una corrida, una posibilidad sería utilizar rutas estáticas para facilitar el trabajo de los controladores y permitir que la red escale. Para poder realizar esto en Maxinet, sería necesario hacer cambios en el código del programa, para permitir que los puertos de cada switch de la red se respeten luego de que esta se particiona para distribuirse entre todas las máquinas. Floodlight permite establecer rutas estáticas mediante un módulo llamado *Static Entry Pusher*.

3.2. Decisiones de diseño

Todos los experimentos realizados en este trabajo parten de la utilización del cliente de RSK real implementado en `Java` (RootstockJ). Esta pieza de software es la misma que la utilizada en producción, por lo que respeta el comportamiento de la aplicación en un entorno real.

Si bien se busca utilizar el cliente de RSK real es necesario instrumentalizarlo a fin de poder tomar mediciones del sistema.

Se instrumentó el cliente RootstockJ para que al realizar ciertos eventos los registre en un archivo, junto con el *timestamp* de la máquina. Esto se implementó utilizando la primitiva `System.currentTimeMillis()` de `Java`. Todas las máquinas tienen sus relojes sincronizados mediante `NTP` a un servidor en la misma red local. Esta sincronización es realizada periódicamente, por lo que se decidió analizar el impacto de este ajuste en las mediciones. Los resultados

se encuentran en el capítulo 4.

Con los eventos logueados fue posible establecer un orden total entre ellos. Sin tener los relojes sincronizados, no hubiera sido posible comparar el orden de eventos entre distintas máquinas.

Los eventos que se instrumentaron fueron los siguientes:

- `broadcastBlock` cuando un nodo propaga un bloque a todos sus peers. Se loguea: el hash del bloque, el número, la dificultad, el hash del bloque padre, y la lista de nodos uncles, con el hash, numero y dificultad de cada uno.
- `newBlock` cuando un nodo recibe un bloque de uno de sus peers. Se loguea: el hash del bloque, el número, el hash del bloque padre y el id del nodo que lo envió.
- `receivedBytes` la cantidad de bytes por cada mensaje recibido en el protocolo.

Cada evento además, contiene el id del nodo que lo emite.

3.2.1. Topología de red

Si bien la red de Bitcoin utiliza nodos distribuidos por Internet, esta distribución no necesariamente es uniforme. Es importante poder caracterizar esta topología para que los resultados sean comparables a la realidad.

El enfoque utilizado para caracterizar la red fue determinar en dónde se ubican los nodos de la red de Bitcoin en el globo y cómo estos están conectados entre sí.

En el capítulo 2 se habló del trabajo de Donet et al. [DPSHJ14], en donde se hace una lista de las `IPs` activas de los nodos de la red Bitcoin. Sobre esta lista de `IPs` los autores utilizaron un servicio de geolocalización de `IPs`, que resultó en una distribución de `IPs` por país. Estas corresponden tanto a la lista de nodos que aceptan conexiones entrantes como a los nodos que se conectan detrás de `NATs`.

Otra opción considerada fue utilizar la `API` propuesta por [bit], que lista sólo los nodos alcanzables, es decir, que aceptan conexiones entrantes. Sin embargo, se decidió en favor del trabajo de Donet, debido a que también se busca considerar los nodos detrás de `NATs`.

Además de la lista de cantidad de nodos por país, era necesario tener información sobre la conectividad entre estos nodos. Para ello utilizamos el mismo set de datos que utilizaron Miller et al. [JH12], el cual es un archivo en formato `GraphML` que contiene información de latencia, jitter y porcentaje de perdida de paquetes entre nodos distribuidos en diferentes países.

Luego, para construir una topología de red experimental de una cantidad fija de nodos, se siguió la siguiente regla. Siguiendo una distribución especifica, dada por el trabajo de Donet, se eligieron un conjunto de nodos por país. A continuación se tomó del archivo `GraphML` usado por Miller et al. la información de las latencias entre todos los nodos elegidos.

La red de Bitcoin es una red heterogénea a la cual se le conectan y desconectan nodos a lo largo del tiempo. Este fenómeno en redes P2P es comúnmente denominado *churn*. Sin embargo, en este trabajo se optó por tener una topología estática, sin cambios en las conexiones ni en los nodos, debido a que la duración de las pruebas es corta (menor a 12 horas).

Si bien Miller et al. mostraron que la topología de red de Bitcoin no es precisamente un grafo aleatorio, en este trabajo se supone que sí lo es, ya que al usar el mismo proceso de *node discovery* que usa RSK en una red tan chica, resulta en una topología con gran densidad de aristas, que dista de la real. En los experimentos, los nodos se conectan solo a una lista de `IPs` precargada, y no se desconectan a lo largo de toda la prueba. Esta lista de peers es elegida de forma aleatoria tal que la red final sea un grafo conexo. Cada nodo tiene una cantidad de aristas acotada entre dos números dados.

Esto describe cómo son las conexiones entre nodos dentro de la aplicación de RootstockJ. Sin embargo, todavía resta definir la topología de red subyacente para ser provista a las herramientas de simulación.

La topología de Mininet consiste en: cada nodo RootstockJ corre sobre un host distinto, y tiene un switch asignado. Entre cada par de switches existe un enlace con los detalles de la conexión (latencia, jitter y packet drop %). En la figura 3.1 se encuentra una esquema de esta topología.

Cuando esta topología se lleva a Maxinet, lo que ocurre es que los hosts y switches quedan

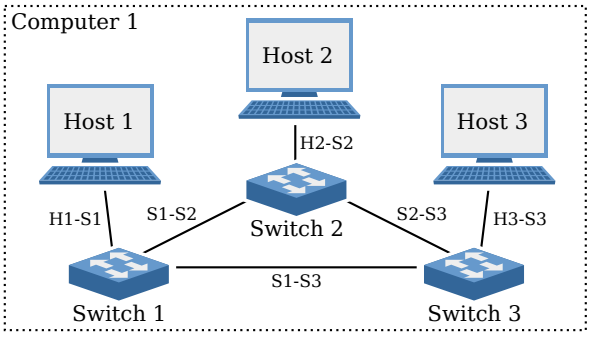


Fig. 3.1: Ejemplo de Topología Mininet. En una misma máquina corren tres hosts virtuales, cada uno conectado a un switch. Entre cada par de switches hay un enlace, con las propiedades de latencia, jitter y packet drop % que corresponderían a la conexión entre sus hosts.

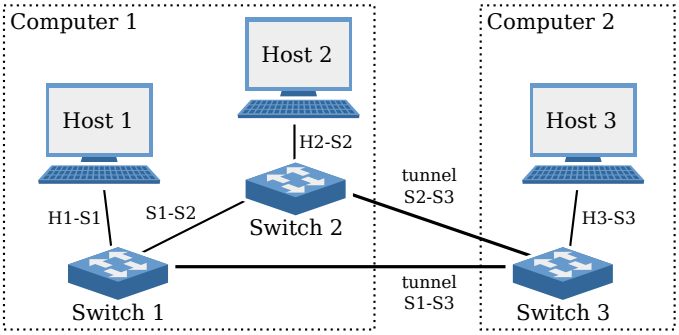


Fig. 3.2: Ejemplo de Topología Maxinet. En una máquina corren 2 hosts y 2 switches, y en otra máquina corre un host y un switch. Cada switch pertenece a la misma máquina que su host, y están conectados mediante un enlace. Entre máquinas físicas, las conexiones entre switches son túneles GRE de Linux.

distribuidos entre todas las máquinas del cluster, uniéndose mediante túneles GRE. Esto se puede ver en la figura 3.2.

Entonces, con Maxinet se tiene definida la topología de red subyacente, que sigue la misma distribución de nodos por país que la red de Bitcoin, esta topología tiene una cantidad cuadrática de enlaces. Esto puede ser un problema para redes más grandes, dado que la topología de red es estática a lo largo de los experimentos, y fijada de antemano, se pueden eliminar los enlaces que no se corresponden a conexiones entre nodos de RSK a fin de obtener una mejor perfomance en la red.

Esta plataforma de experimentación y topología cumple que la distribución geográfica de los nodos sigue las proporciones reportadas por Donet et al., y la conexión entre pares de nodos cuenta con las propiedades de valor de latencia, varianza y pérdida de paquetes que reflejan

las variaciones que se pueden tener entre dos nodos conectados por Internet con rutas que varían. Por todo esto consideramos que es una plataforma representativa de la topología real de Bitcoin.

3.2.2. Proceso de Minado Simulado

En Bitcoin se especifica un tiempo medio entre bloques (llamado *target*), lo cual significa que, en promedio, debe aparecer un bloque nuevo en la red cada *target* segundos. Si resulta que debido al *hashing power* de la red los bloques aparecen más rápido que lo indicado por el tiempo target, la dificultad del bloque se incrementa. Si, por el otro lado, los bloques aparecen con menor frecuencia, la dificultad disminuye. De esta forma Bitcoin se adapta a variaciones del *hashing power* subyacente de la red. Este ajuste se hace cada 2016 bloques.

Uno de nuestros objetivos es analizar aspectos de la red para distintos valores de *target*, y para ello deberíamos esperar a que la dificultad converja para la configuración de mineros de cada experimento, este proceso puede demorar bastante tiempo. Por otro lado, el proceso de minado es intensivo en procesamiento, y dificulta que se hagan comparaciones entre dos sistemas distintos (con hashing power distintos). Queremos buscar una forma de simular el minado de Bitcoin, sin tener que depender del procesamiento, ni de la dificultad.

El criterio para que un bloque Bitcoin sea aceptado, es que el hash del bloque (interpretado como un entero sin signo) sea menor a la dificultad. Esta dificultad varía para ajustarse a los cambios de *hashing power* en la red. Se supone que esta dificultad está fija (pues en nuestras pruebas no se agregan ni remueven nodos a la red y el poder de cómputo de los nodos de la red no varía). El bloque cuenta con un campo *nonce* que puede ser incrementado para obtener distintos hashes.

Entonces, un *minero Bitcoin* al tener un bloque que minar, calcula hashes (`SHA256`) incrementando el nonce hasta obtener algún hash que sea menor a la dificultad esperada. Como los hashes son criptográficamente seguros, se puede asumir una distribución uniforme en el rango $[0, 2^{256})$. Esto es un *ensayo de Bernoulli*, donde la probabilidad de éxito es $\frac{\text{dificultad}}{2^{256}}$.

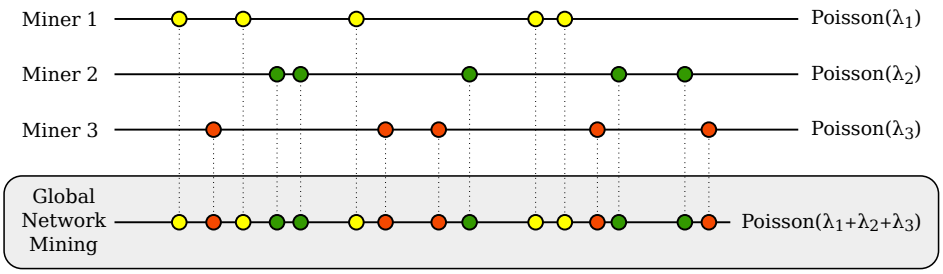


Fig. 3.3: Suma de tres procesos de Poisson independientes. Cada línea simula el proceso de minado a lo largo del tiempo. El resultado es también un proceso de Poisson.

Debido a que se realizan muchos de estos experimentos en intervalos de tiempo muy cortos, se lo puede ver como un proceso de *Poisson* con algún parámetro λ. Entonces, teniendo n mineros, tenemos $\lambda_1, ..., \lambda_n$ parámetros para n procesos de *Poisson* independientes.

La suma de n procesos de *Poisson* independientes es también un proceso de *Poisson*, con parámetro $\sum_{i=1}^{n} \lambda_i$. Esto significa que el proceso de minado de toda la red se puede ver también como un proceso de *Poisson* (ver figura 3.3).

El tiempo entre dos eventos en un proceso de *Poisson* de parámetro λ sigue una distribución *Exponencial* con media $\frac{1}{\lambda}$, por lo cual si queremos que el tiempo medio entre bloques sea de 10 segundos, tendremos que $\frac{1}{\lambda} = 10$, entonces $\lambda = \frac{1}{10}$

Luego, sabemos que este λ es la suma de los λ_i, por lo cual es posible distribuirlo acorde al escenario que resulte de interés, configurando el porcentaje del hashing power de acuerdo a si se prefiere tener una distribución homogénea o una distribución particular.

Algoritmo de minado simulado

Cada minero va a recibir su parámetro λ_i que le determina su hashing power. El proceso de minado es el siguiente:

1. Sea X un número aleatorio tomado de una distribución *Exponencial* (con media λ_i)
2. Dormir por X segundos.
3. Obtener el próximo bloque a minar.
4. Calcular Hashes suponiendo que la dificultad del bloque es 1.

5. Publicar el bloque minado.

Es importante notar que, dado que cada hash es independiente del anterior, no importa si mientras dormimos llega un nuevo bloque sobre el cual minar, dado que el momento en el que deberíamos encontrar el bloque, es inmediatamente posterior al que terminamos de dormir. Por esta razón, recién después de dormir se toma un bloque para minar.

Ajuste de dificultad

Como fue mencionado en el capítulo 1, en Bitcoin el ajuste de dificultad se realiza cada 2016 bloques. Este ajuste puede ser de un factor hasta 4. En este proceso, uno puede suponer que el proceso de minado entre un bloque y el anterior es independiente (pues la dificultad no depende del anterior, salvo una vez cada 2016 bloques).

Por otro lado, en *Ethereum* el ajuste se produce bloque a bloque, de manera más gradual: suponiendo que el tiempo de bloque esperado es D, si el bloque número n fue minado con timestamp t_n, al momento de armar el bloque $n+1$, si el timestamp t_{n+1} (la hora actual en la que se arma el bloque), se encuentra a más de D segundos de t_n, entonces el bloque $n+1$ deberá tener $\frac{1}{2048}$ menor dificultad que el bloque n. En cambio, si $t_n - t_{n+1} \leq D$, el bloque $n+1$ deberá tener $\frac{1}{2048}$ mayor dificultad.

Este ajuste de dificultad produce un cambio más gradual en la dificultad, a costa de fluctuaciones constantes en la dificultad. Como mencionamos anteriormente, el minado simulado que se usa en este trabajo no cuenta con ajuste de dificultad, para simplificar el proceso.

4. RESULTADOS

En este capítulo se presenta el conjunto más relevante de experimentos realizados con el software de RSK instrumentado en las redes simuladas. Se comienza con una sección preliminar que trata sobre experimentos que validan la posibilidad de distribuir los nodos en varias computadoras utilizando Maxinet. A continuación se realiza un análisis sobre el efecto que tiene el tiempo de generación de bloques (target) en la cantidad de forks de la red, siendo este el experimento principal del trabajo. Finalmente se presentan resultados sobre simulación del comportamiento de pools de minería con merged-mining. El capítulo concluye con una breve discusión sobre los resultados y el uso de Maxinet.

4.1. Experimentos Preliminares

En el capítulo anterior se mencionó que para determinar la viabilidad de distribuir el trabajo en varias máquinas era necesario garantizar que, por un lado, el ajuste de reloj de los nodos no fuera significativo como para cambiar el orden de los eventos del sistema, y por otro lado, que la red definida por Maxinet se comporte de forma similar a la red de Mininet.

Para analizar el impacto del ajuste de reloj realizado por `NTP`, se agregó a cada medición el valor de la primitiva `System.nanoTime()` de `Java`, que devuelve el tiempo relativo en nanosegundos desde que comenzó el programa. Este valor no se ve afectado por cambios en el reloj del sistema.

Para cada par de mediciones consecutivas en cada nodo, se comparó la diferencia entre los valores de `System.currentTimeMillis()` y la diferencia entre los valores de `System.nanoTime()`, es decir, por un lado la diferencia entre el tiempo percibido de dos eventos (que se ve afectado por la sincronización de `NTP`) y la diferencia de tiempo real entre ambos eventos. Este valor indica qué tan grandes son los ajustes de reloj.

Luego de una corrida de 10 horas, en un nodo que registró 27784 eventos, el ajuste de tiempo promedio fue 0,3831 ms, la mediana 0,3796 ms, con un desvío estándar de 0,2111. Para el resto de los 31 nodos de esa corrida, los valores fueron muy similares, y entre todos el máximo ajuste fue de 4 segundos.

Esto muestra que el ajuste de tiempo producido por `NTP` es muy gradual, y no afecta el orden de los eventos, ya que la latencia entre dos máquinas es varios órdenes de magnitud mayor.

4.1.1. Mininet vs Maxinet

Con el objetivo de poder utilizar Maxinet como remplazo de Mininet, es necesario validar que ambas herramientas se comportan de forma equivalente. Esta comprobación es muy difícil de realizar midiendo propiedades genéricas de la red, tales como latencia y ancho de banda entre enlaces, pues no ofrece garantías para todos los escenarios posibles. Es decir, medir propiedades sobre la red y compararlas, no permite garantizar que para un determinado uso estas herramientas se comporten de manera similar, incluso a pesar de que los valores medidos sean iguales.

Considerando esto, se decidió realizar un experimento representativo del uso esperado de la aplicación para medir el impacto de reemplazar Mininet por Maxinet para las métricas definidas en este trabajo.

Para esta prueba, se utilizó Maxinet y Mininet, con una topología que respeta las siguientes características: una red de 32 nodos[1], de los cuales 10 eran mineros, minando con minado simulado e igual hashing power. Cada nodo contaba con entre 5 y 7 conexiones y el tiempo entre bloques (target) fue de 10 segundos. Los valores elegidos fueron determinados de manera proporcional a los utilizados en la red de producción de RSK.

Es importante destacar que en esta prueba el minado simulado es muy relevante, ya que permite comparar la dinámica del sistema en dos infraestructuras de hardware sumamente

[1] La decisión de utilizar esta cantidad de nodos se debió a la limitación de memoria en la infraestructura donde se ejecutaron los experimentos de Mininet.

distintas. El minado simulado permite ignorar el poder de cómputo de las máquinas, dado que la aplicación RootstockJ en sí no hace uso intensivo del CPU.

Para las pruebas con Mininet, se contó con una **PC Dell PowerEdge C6145** equipada con:

- 4 x Octa core AMD Opteron 6276s (64 cores)
- 128 GB RAM DDR3.
- 1 x HDD: 500GB 7200RPM SATA 6Gb/s
- 2 x Intel 82576 Ethernet Gigabit Network Card
- 2 x Placa de red Infiniband Mellanox MT26428 IB QDR 10GigE Red de baja latencia.

Para las pruebas con Maxinet, se utilizó el clúster «Museo» del CECAR, que cuenta con 20 computadoras equipadas con:

- 2 Procesadores Intel(R) Xeon(TM) CPU 2.66GHz con 2 cores cada uno y HyperThreading.
- 2 GB RAM DDR3
- 1 x HDD: 80GB 7200RPM SATA
- 1 x Placa de red Infiniband Mellanox MT25208 IB 10GigE Red de baja latencia.

La infraestructura mencionada se utilizó para todos los experimentos de este trabajo.

Métricas

A continuación se describirán las métricas que consideramos relevantes para los experimentos. Estas mediciones son útiles para entender el comportamiento de la red.

La primera de ellas es sobre estadísticas de tiempos de generación de bloques. Un ejemplo de esto es la tabla 4.1. Esta métrica da información sobre cómo se crean nuevos bloques en la red para un target establecido. El campo *«Bloques»* tiene, por un lado, el total de bloques que

terminaron en la cadena principal, y por otro el total de todos los bloques que hubo en la red, incluyendo bloques *stale*.

Dos de las métricas más importantes hablan sobre los forks en la red. En este trabajo analizamos los forks desde dos perspectivas distintas: por un lado la cantidad de bloques competitivos, y por otro los forks por profundidad. En la figura 4.1 se esquematizan ambos enfoques.

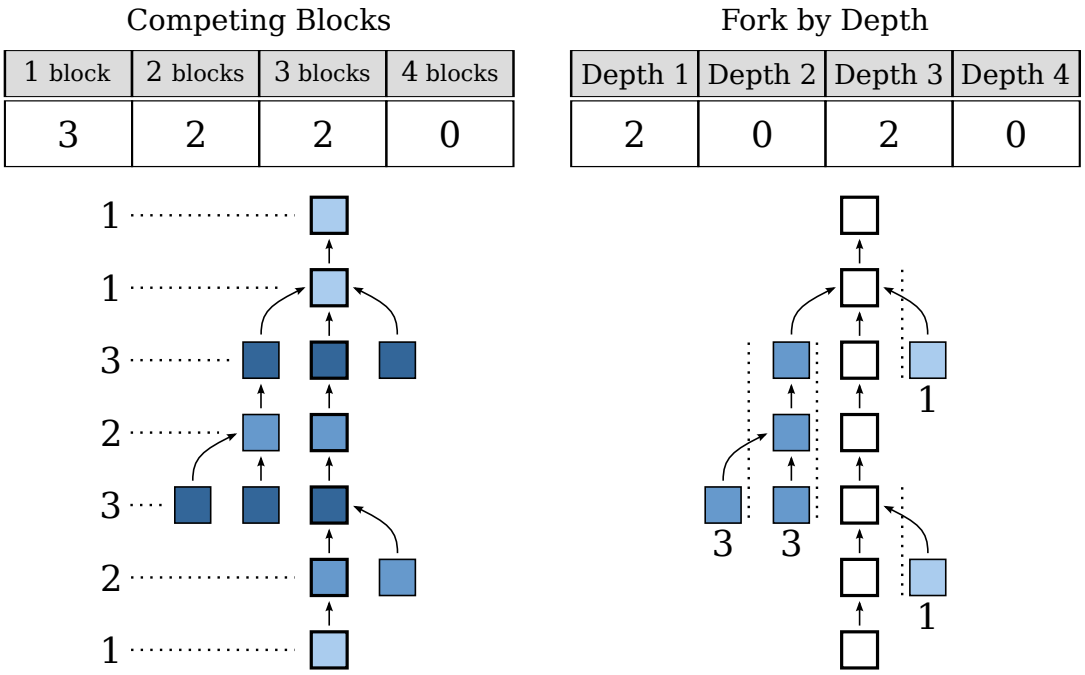


Fig. 4.1: Ejemplo mostrando dos perspectivas distintas para analizar los forks de la red: por bloques competitivos (izquierda) y por profundidad de los forks (derecha).

En la figura se muestra la misma Blockchain con algunos forks. A la izquierda se cuentan, para cada bloque, cuantos bloques competitivos tuvo, agrupando luego por estas cantidades. Por ejemplo, en dos ocasiones hubo 3 bloques competitivos. Esto representaría a 3 mineros hallando un bloque antes de enterarse del bloque que halló otro minero[2].

A la derecha de la figura, se cuentan los forks según su profundidad. La profundidad de un fork es la distancia desde el bloque de mayor altura hasta algún bloque de la cadena principal. Esta métrica nos aporta información acerca de cuánto estuvo dividido el poder de cómputo de la red.

[2] decimos que minaron el bloque «a la vez».

Por otro lado, medimos tiempos de propagación de bloques, es decir cuánto tardan en promedio los bloques en llegar a cierto porcentaje de nodos de la red. En la figura 4.2 se puede ver una esquematización de lo que se busca capturar.

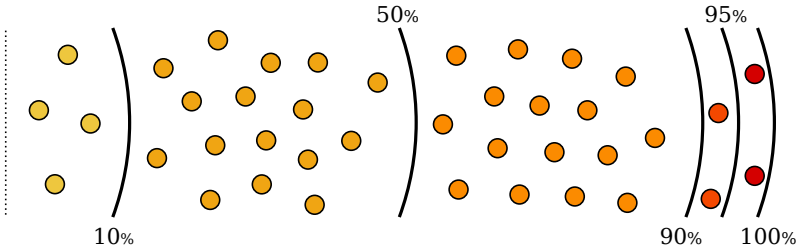


Fig. 4.2: Esquematización sobre propagación de información en la red: se busca capturar cuánto tiempo se tarda, en promedio, en propagar un mensaje a distintos porcentajes de nodos de la red.

Resultados

La naturaleza de este tipo de experimentos, donde la aparición de bloques depende de un proceso con pérdida de memoria[3], y las características del experimento no varían a lo largo del tiempo, permite que en lugar de repetir el mismo varias veces, se pueda tomar una muestra lo suficientemente grande, y que esta última sea representativa.

A raíz de lo expuesto anteriormente, el experimento realizado tuvo una duración de 10 horas, y se analizaron los primeros 3600 bloques del mismo. En Bitcoin, donde el tiempo entre bloques es de 10 minutos, esto sería equivalente a un experimento de 25 días de duración. Para el resto de los experimentos analizamos una cantidad menor de bloques.

En lo que corresponde a generación de bloques, como ambos usaron minado simulado, los tiempos de generación de bloques fueron similares:

De la tabla 4.1 se puede ver que la diferencia entre cantidad de bloques es ínfima: en Mininet sólo fueron minados 9 bloques más que en Maxinet, es decir, es un 0,24 % más de bloques.

En cuanto a la creación de forks en la red, tampoco se notaron grandes diferencias. En la tabla 4.2 se muestra la cantidad de forks por profundidad. Puede apreciarse que la cantidad de forks no varía tanto. Además, la profundidad máxima es la misma. En Mininet se hallaron

[3] Los bloques anteriores no influyen sobre los próximos bloques a minar

Tiempo de Generación de Bloques			
Experimento	Media (ms)	Mediana (ms)	Bloques
Mininet	10346,09	7502,50	3600/3719
Maxinet	10302,24	7309,50	3600/3710

Tab. 4.1: Tiempos de generación de bloques. El campo total de bloques tiene por un lado el total de bloques que terminaron en la cadena principal, y por otro el total de todos los bloques que hubo en la red, esto incluye bloques *stale*.

9 bloques más que en Maxinet, estos corresponden a 7 bloques que terminaron en forks de longitud 1 (113 en Mininet contra 106 en Maxinet), y a 2 bloques que terminaron en una cadena de longitud 2.

Generación de forks (por profundidad)		
Experimento	Profundidad 1	Profundidad 2
Mininet	113	2
Maxinet	106	1

Tab. 4.2: Generación de forks por profundidad. Es la cantidad de bloques que pertenecen a una cadena de longitud 1 o 2 y que no son parte de la cadena principal.

Por otro lado, en la tabla 4.3 se puede apreciar que la cantidad de bloques competitivos es levemente mayor en Mininet. La clasificación de los bloques muestra cuántos mineros hallaron bloques a la vez. Sin embargo, los valores para una corrida de más de 10 horas son similares y la diferencia se debe meramente a la naturaleza aleatoria del proceso de minado.

Cantidad de bloques competitivos			
Experimento	1 bloque	2 bloques	3 bloques
Mininet	3485	115	1
Maxinet	3493	108	0

Tab. 4.3: Cantidad de bloques competitivos. Es la cantidad de bloques agrupados por cuántos tienen el mismo número. Por ejemplo, la columna 2 para Mininet muestra que hubieron 115 bloques de distinto número, para los cuales exactamente dos mineros los encontraron a la vez.

El valor en el que sí se encontró una diferencia apreciable es el tiempo de propagación de bloques. En general, correr los experimentos sobre Maxinet, que termina utilizando hardware

real para comunicación inter-máquinas, lleva a un mayor tiempo de propagación y mayor varianza.

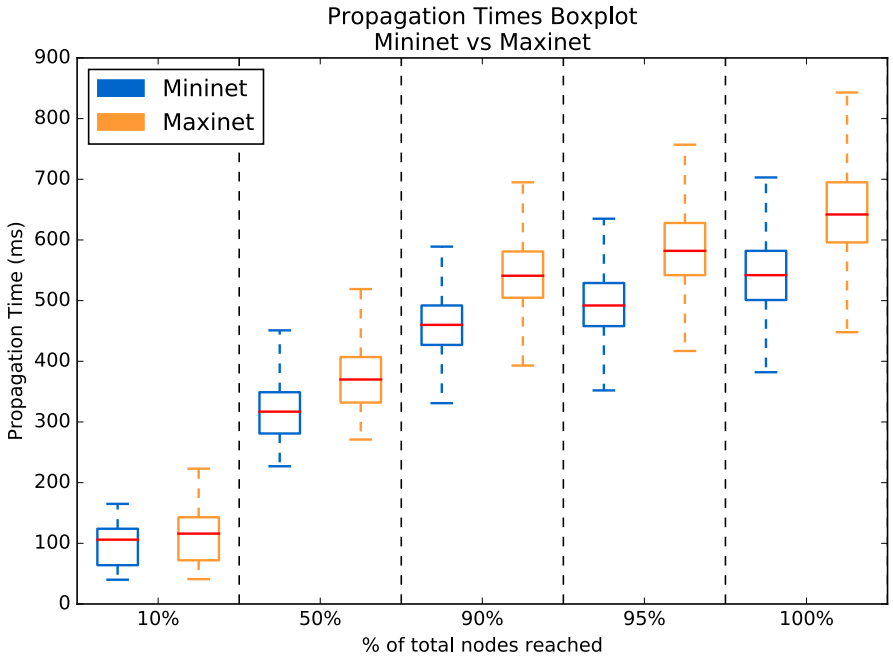


Fig. 4.3: Boxplot comparando tiempos de propagación para distintos tiempos de generación de bloques. En cada sección se ve un resumen del tiempo de propagación de Mininet y Maxinet a un cierto porcentaje de nodos.

En la figura 4.3 se muestra un boxplot de los tiempos de propagación de Mininet y Maxinet a distintos porcentajes de nodos. De este gráfico se desprende que, en general, se tarda más en propagar los bloques sobre Maxinet. Creemos que esto se debe al *overhead* de usar una red física en lugar de correr todo en un entorno virtual dentro de la misma máquina.

En la tabla 4.4 y la tabla 4.5 se encuentra información sobre los tiempos de propagación a distintos porcentajes de nodos de la red. Estos valores se corresponden con los que muestra la figura 4.3. Las diferencias más marcadas se encuentran en los tiempos de propagación entre el 90 % y el 100 % de los nodos. Para el 90 % de los nodos, en promedio Mininet demora 459 ms, y Maxinet 541, una diferencia de 82 ms. Para llegar al 100 % de los nodos la diferencia es de 100 ms. Cabe destacar que para porcentajes menores de nodos, la diferencia es significativamente más pequeña.

Estadísticas de propagación de bloques en Mininet (ms)					
% de Nodos	Media	Mediana	Max	Mín	σ
10	96	106	150	40	31
50	318	317	440	227	45
90	459	460	606	321	48
95	493	492	640	348	52
100	541	542	696	361	57

Tab. 4.4: Tiempo de Propagación de bloques para los experimentos con Mininet. La primer columna representa el porcentaje de nodos alcanzado. Por ejemplo, en promedio, en 318 ms se llega a propagar un bloque al 50 % de los nodos

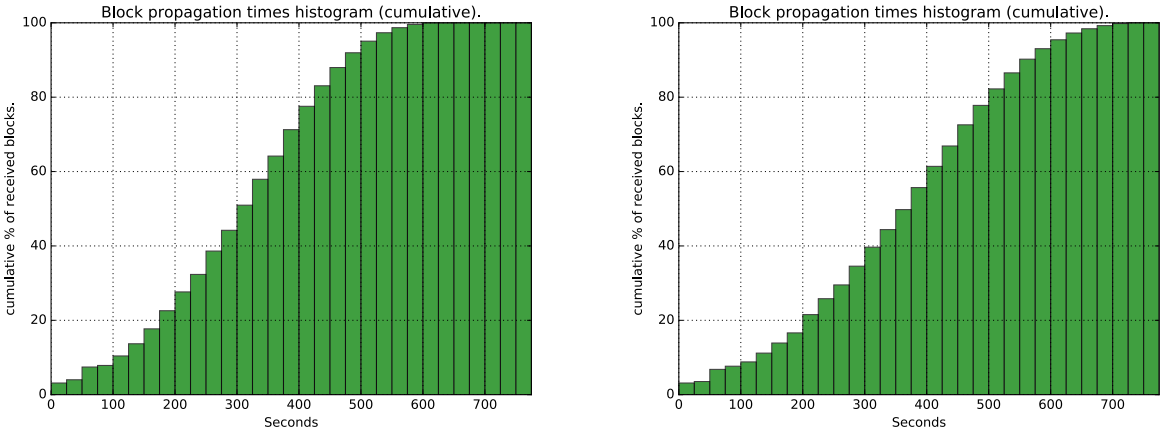
Estadísticas de propagación de bloques en Maxinet (ms)					
% de Nodos	Media	Mediana	Max	Min	σ
10	108	115	174	41	46
50	371	369	512	271	50
90	541	541	711	351	57
95	583	582	773	377	62
100	643	642	843	435	70

Tab. 4.5: Tiempo de Propagación de bloques para los experimentos con Maxinet. La primer columna representa el porcentaje de nodos alcanzado. Por ejemplo, en promedio, en 371 ms se llega a propagar un bloque al 50 % de los nodos

Otro aspecto para analizar los tiempos de propagación de bloques se puede observar en las figuras 4.4a y 4.4b, donde se muestra que en Mininet los bloques se propagan más rápido que en Maxinet. Por ejemplo, en Mininet a los 600 ms ya se propagó el 100 % de los bloques, mientras que en Maxinet eso ocurre recién a los 700 ms.

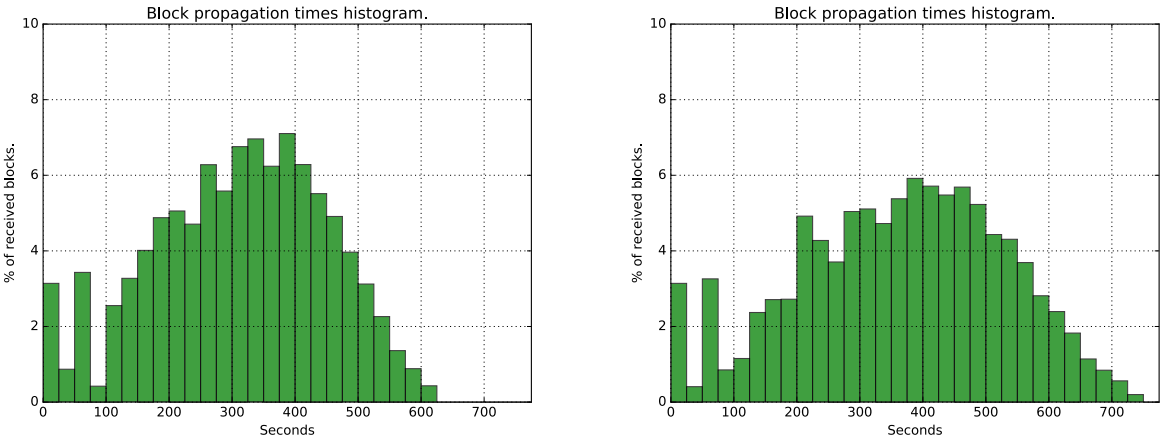
En las figuras 4.5a y 4.5b se presenta el mismo histograma pero de forma no acumulativa. En estos últimos, se observa un leve corrimiento de la media y un aumento en la varianza en Maxinet. Sin embargo, se puede ver que las distribuciones son similares. En un análisis más profundo se requeriría la utilización de tests estadísticos para corroborar nuestra hipótesis.

Es importante destacar la diferencia entre las figuras 4.5a y 4.5b y las tablas 4.4 y 4.5: los histogramas son sobre el total de bloques enviados, sin distinguir a qué porcentaje de nodos



(a) Mininet *(b)* Maxinet

Fig. 4.4: Histograma acumulativo de propagación de bloques



(a) Mininet *(b)* Maxinet

Fig. 4.5: Histogramas de propagación de bloques

llegó, mientras que las tablas miden el porcentaje de nodos alcanzado.

A partir de lo expuesto, entendemos que Maxinet se comporta de forma similar a Mininet. Sin embargo, hay que tener en cuenta que en Maxinet el tiempo de propagación será mayor y tendrá mayor varianza, algo característico de redes heterogéneas como Internet.

4.2. Variación de Tiempo entre bloques

La principal motivación de este trabajo fue analizar el impacto que tiene el tiempo entre bloques (target) en la creación de forks en la red. Esta sección detalla los experimentos realizados

y presenta los resultados obtenidos.

Se presentan tres experimentos. Entre ellos, sólo se altera la latencia entre enlaces, con el objetivo de simular redes de mayor diámetro. Dado que por limitaciones de hardware no fue posible agregar más nodos a la red, se propuso aumentar la latencia entre enlaces como forma de simular redes con mayor tiempo de propagación.

El primer experimento se denomina «Latencia Normal» ya que la latencia no fue modificada con respecto al valor provisto por el set de datos de Miller et al [JH12]. Estos valores de latencias son los mismos que fueron utilizados en los experimentos anteriores. En el segundo experimento, denominado «Latencia 2x» se analiza el sistema usando latencias al doble del valor normal. Por último, el experimento «200ms de Latencia», todas las latencias de todos los enlaces fueron fijadas en 200 ms, un valor que excede bastante la latencia promedio de los enlaces de nuestro dataset (96,60 ms).

En todos los experimentos se varía el *target* (tiempo entre bloques) y se analiza cómo esto afecta la red. Los valores de target son de $\{10,\ 8,\ 6,\ 4,\ 2\}$ segundos. La red usada en los experimentos cuenta con 64 nodos y 10 de ellos siendo mineros, con minado simulado e igual hashing power. Cada nodo tiene entre 3 y 5 conexiones.

Notar que en estos experimentos se usa minado simulado, sin ajuste de dificultad. Por cómo funciona el algoritmo de dificultad de RSK, explicado en el capítulo 1, podría haber ocurrido que la dificultad baje o suba, impactando en la cantidad de forks creados. Consideramos que analizar este efecto está fuera del alcance de este trabajo, pero sería interesante estudiarlo como trabajo futuro.

4.2.1. Latencia normal

Como se mencionó anteriormente, este experimento consiste en analizar el comportamiento de la red de nuestro sistema en función del tiempo entre bloques, dejando las latencias intactas.

Es esperable que por más que se varíe el tiempo entre bloques, los tiempos de propagación de la red se mantengan intactos. En la figura 4.6 se puede ver que el tiempo de propagación

de bloques no varía significativamente al cambiar el target. En esa figura, se agrupan los tiempos de propagación por porcentaje de nodos alcanzados y luego por experimento. Se puede observar que para cada porcentaje de nodos, el tiempo de propagación entre experimentos es muy similar. En la tabla 4.6 se detalla la estadística de los tiempos de propagación para el target de 10 segundos.

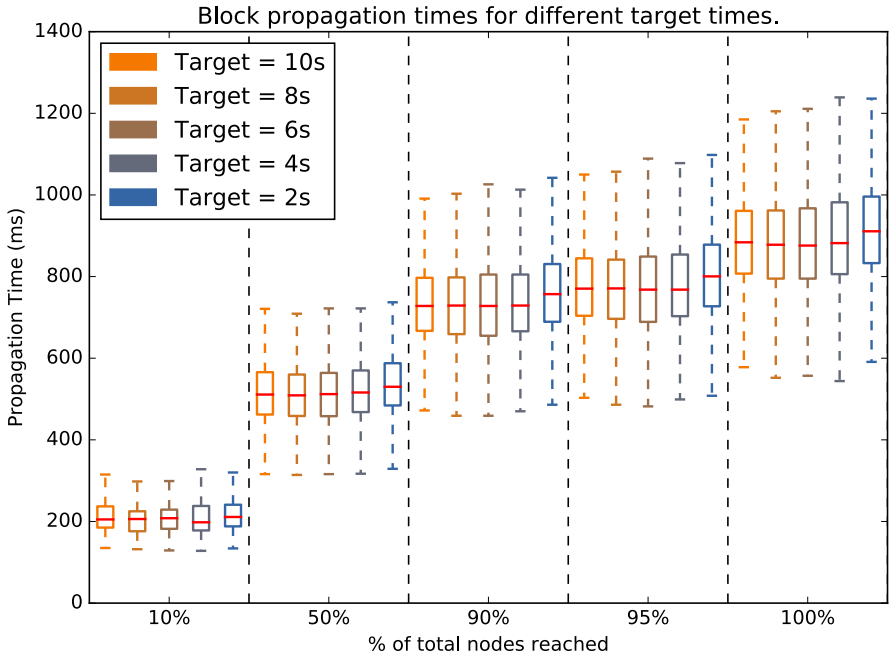


Fig. 4.6: Boxplot comparando tiempos de propagación para distintos valores de target. En cada sección se ve un resumen del tiempo de propagación a un porcentaje dado de nodos, para distintos valores de target. La figura corresponde a un experimento con 64 nodos con conectividad de entre 3 y 5 nodos, 10 nodos mineros y latencia normal.

Con respecto al tiempo de generación de bloques, en la tabla 4.7 se puede ver el tiempo efectivo de generación de nuevos bloques en función del target establecido. Notar que este tiempo sólo tiene en cuenta los bloques nuevos, y no los que quedan como *stale*. Es por este motivo que dicho valor dista un poco con respecto al target.

Acerca de la creación de forks en la red, la tabla 4.8 muestra, como es esperable, que a medida que el tiempo entre bloques disminuye, aumenta la cantidad de forks en la red. Cabe destacar que el primer fork de profundidad 3 se produce con 4 segundos de target. Esto es significativo pues es un evento muy poco probable. Luego, alcanzando los 2 segundos de target, se producen

una gran cantidad de forks de mayor profundidad.

Analizando la cantidad de bloques competitivos que muestra la figura 4.9, se ve que si bien al disminuir el tiempo entre bloques aumenta la cantidad de bloques competitivos, no es tan marcado el aumento de cantidad de mineros que hallan bloques a la vez. Por ejemplo, para el target de 10 segundos, sucedió que 4 mineros hallaron un bloque a la vez, y eso no se volvió a repetir hasta el caso de 4 segundos de target.

Si bien el cambio en el target generó una mayor cantidad de forks, no fueron tantos como se esperaba. Creemos que esto se debe al tiempo de propagación de los bloques. En el siguiente experimento se analiza el efecto de aumentar el tiempo de propagación aumentando la latencia.

Estadísticas de propagación de bloques					
% de nodos	Media (ms)	Mediana (ms)	Max (ms)	Min (ms)	σ
10	211	204	339	135	41
50	513	510	740	316	83
90	730	727	1025	472	107
95	773	769	1088	486	111
100	885	883	1233	554	121

Tab. 4.6: Tiempos de propagación de bloques para un target de 10 segundos. La primera columna representa el porcentaje de nodos alcanzado.

Tiempo de Generación de Bloques			
Target (segs)	Media (ms)	Mediana (ms)	Bloques
10	10589,17	7220,00	900/946
8	8645,46	6215,50	900/961
6	6675,18	4776,50	900/987
4	4513,95	3395,50	900/1004
2	2543,77	1808,00	900/1147

Tab. 4.7: Tiempos de generación de bloques para distintos valores de target.

Generación de forks por profundidad				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	42	1	0	0
8	55	2	0	0
6	85	0	0	0
4	95	2	1	0
2	193	14	7	1

Tab. 4.8: Generación de forks por profundidad, para distintos valores de target. Cantidad de bloques que pertenecen a cadenas de distinta longitud y que no son parte de la cadena principal.

Cantidad de Bloques Competitivos					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	más de 4 bloques
10	860	39	1	1	0
8	846	51	4	0	0
6	818	81	2	0	0
4	804	93	3	1	0
2	695	177	25	2	2

Tab. 4.9: Cantidad de bloques competitivos, para distintos valores de target. Es la cantidad de bloques agrupados por cuántos tienen el mismo número. Por ejemplo, la fila 2 columna 4 muestra que en 4 ocasiones distintas, 3 mineros hallaron un bloque a la vez.

4.2.2. Latencia 2x

El segundo experimento de esta serie consistió en aumentar la latencia al doble del experimento de «Latencia Normal». Al igual que en el caso anterior, al variar el tiempo entre bloques, los tiempos de propagación se mantienen igual. En la figura 4.7 se puede hallar un boxplot análogo al de la figura 4.6, que muestra un resumen de la distribución de los tiempos de propagación.

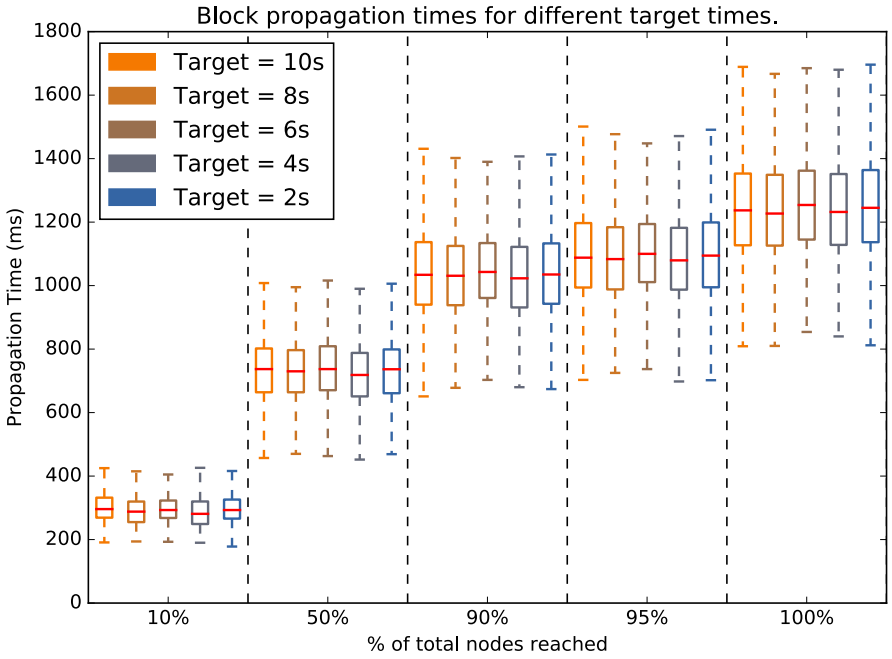


Fig. 4.7: Boxplot comparando tiempos de propagación para distintos valores de target. En cada sección se ve un resumen del tiempo de propagación a un porcentaje dado de nodos, para distintos valores de target. La figura corresponde a un experimento con 64 nodos con conectividad de entre 3 y 5 nodos, 10 nodos mineros y latencia 2x.

Al aumentar la latencia, es esperable que el tiempo de propagación de bloques también aumente. En la tabla 4.10 se muestran los tiempos de propagación promedio para un target de 10 segundos. En comparación con el experimento anterior (ver tabla 4.6), la latencia promedio para llegar al 50 % de los nodos se incrementa en 219 ms (un incremento del 42,77 %).

En cuanto a los tiempos entre bloques, en la tabla 4.11 se puede ver un leve incremento en los tiempos de generación con respecto al experimento con latencia normal. Esto se puede deber a que, al haber más forks en la red, los tiempos de generación de nuevos bloques aumentan.

Como es esperable, la cantidad de forks también aumenta en función del tiempo entre bloques, en la tabla 4.12 se muestran los forks categorizados por profundidad. También ocurre que en los 4 segundos de target se llega a un fork de profundidad 3, lo cual es poco frecuente.

Por otro lado, en la tabla 4.13 se ve la cantidad de bloques competitivos. Comparando estas dos tablas con sus análogas en el experimento «Latencia Normal» (tabla 4.12 y tabla 4.13), se puede ver un aumento marcado en la cantidad de forks, lo cual es esperable pues hubo un aumento en el tiempo de propagación. Además, para targets chicos, la cantidad de forks aumenta mucho más en este experimento que en el anterior.

Con este experimento se vio cómo aumenta la cantidad de forks de la red teniendo un mayor tiempo de propagación, y comparándolo con el experimento «Latencia Normal» , se ve que a medida que el target disminuye, la cantidad de forks aumenta más en este experimento que en el anterior. Por esto, se decidió aumentar aún más la latencia entre nodos. El análisis puede verse en la siguiente sección.

Estadísticas de propagación de bloques (ms)					
% de Nodos	Media	Mediana	Max	Min	σ
10	302	295	494	191	57
50	732	736	1062	441	121
90	1037	1033	1499	640	158
95	1094	1087	1573	664	165
100	1244	1236	1789	809	176

Tab. 4.10: Tiempos de propagación de bloques para un target de 10 segundos. La primer columna representa el porcentaje de nodos alcanzado.

Tiempo de Generación de Bloques			
Target (segs)	Media (ms)	Mediana (ms)	Bloques
10	11135,99	7769,50	900/971
8	8396,47	5851,00	900/979
6	6431,34	4820,00	900/992
4	4708,13	3398,50	900/1036
2	2601,34	2028,50	900/1139

Tab. 4.11: Tiempos de generación de bloques para distintos valores de target.

Generación de forks (por profundidad)				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	65	2	0	0
8	75	1	0	0
6	88	1	0	0
4	121	5	1	0
2	213	31	8	1

Tab. 4.12: Generación de forks por profundidad, para distintos valores de target. Es la cantidad de bloques que pertenecen a cadenas de distinta longitud y que no son parte de la cadena principal.

Cantidad de bloques competitivos					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	más de 4 bloques
10	834	65	3	0	0
8	828	70	2	1	0
6	813	86	2	0	0
4	779	112	9	0	1
2	646	226	21	6	2

Tab. 4.13: Cantidad de bloques competitivos, para distintos valores de target. Es la cantidad de bloques agrupados por cuántos tienen el mismo número. Por ejemplo, la fila 2 columna 4 muestra que 3 mineros hallaron un bloque a la vez.

4.2.3. Latencia 200ms

Aumentar aún más la latencia por un factor constante generaba gran varianza en las mismas y algunas conexiones tenían latencias demasiado altas como para que RootstockJ funcione correctamente. Por ello, se decidió subirlas hasta un tope de 200 ms. Cada enlace entre nodos cuenta con una latencia de 200 ms.

Como en los experimentos anteriores, los tiempos de propagación no varían significativamente en función del target. Esto se puede ver en el boxplot de la figura 4.8.

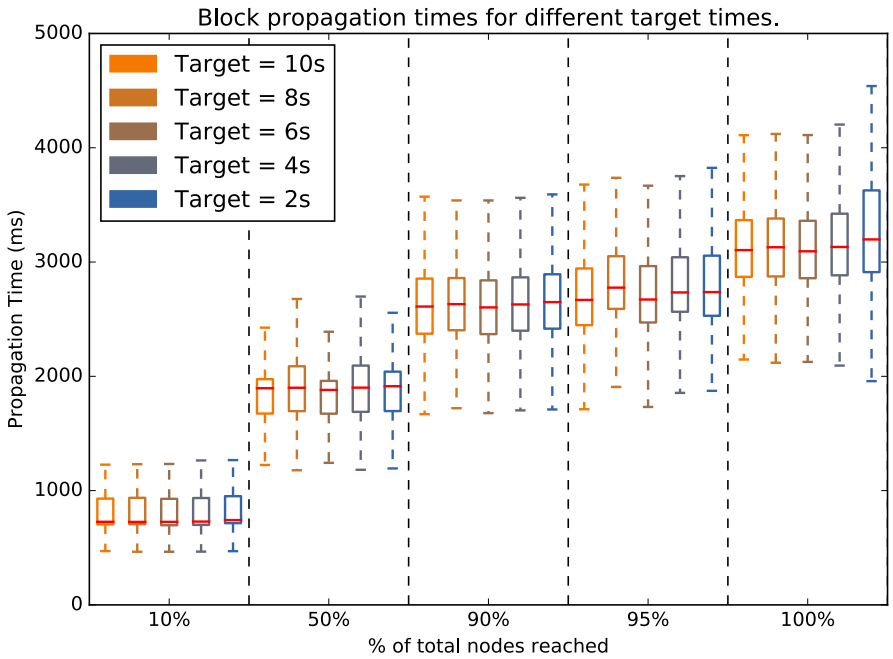


Fig. 4.8: Boxplot comparando tiempos de propagación para distintos valores de target. En cada sección se ve un resumen del tiempo de propagación a un porcentaje dado de nodos, para distintos valores de target. La figura corresponde a un experimento con 64 nodos con conectividad de entre 3 y 5 nodos, 10 nodos mineros y latencia 200 ms.

En lo que respecta al tiempo promedio de generación de nuevos bloques, el mismo es significativamente mayor al target en cada caso. Esto puede verse en la tabla 4.15 donde la diferencia entre dichos valores es de aproximadamente 1,5 segundos.

Los tiempos de propagación para el target de 10 segundos se pueden ver en la tabla 4.14. En comparación con el experimento «Latencia Normal» , dichos valores son mucho mayores. El

tiempo de propagación al 95 % de los nodos del primer experimento, es menor al tiempo de propagación al 10 % de los nodos de este experimento.

Por otro lado, podemos ver cómo incrementa notablemente la cantidad de forks, tanto en cantidad como en profundidad. En la tabla 4.16 se puede ver que a diferencia de los experimentos anteriores, aquí ya se alcanzan forks de profundidad 3 a los 8 segundos, mientras que esto ocurría recién con un target de 4 segundos para los experimentos de «Latencia Normal» y «Latencia 2x».

En la tabla 4.17, se puede ver que una gran cantidad de los bloques minados fueron bloques competitivos. En particular, para un tiempo de 2 segundos, más de la mitad de los bloques fueron bloques competitivos, que terminaron en forks.

Estadísticas de propagación de bloques (ms)					
Procentaje % de Nodos	Media	Mediana	Max	Min	σ
10	793	727	1210	471	164
50	1837	1894	2660	1194	311
90	2569	2610	3616	1669	379
95	2717	2668	3810	1692	389
100	3100	3102	4337	1942	421

Tab. 4.14: Tiempos de propagación de bloques para un target de 10 segundos. La primera columna representa el porcentaje de nodos alcanzado.

Tiempo de Generación de Bloques			
Target (segs)	Media (ms)	Mediana (ms)	Total de Bloques
10	11399,47	8169,50	900/1059
8	9569,75	7086,50	900/1094
6	7595,54	5721,00	900/1140
4	5474,95	4430,00	900/1283
2	3516,32	3046,50	900/1547

Tab. 4.15: Tiempos de generación de bloques para distintos valores de target.

Generación de forks por profundidad				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	135	11	0	0
8	166	12	1	0
6	190	22	3	0
4	290	34	8	0
2	395	90	23	4

Tab. 4.16: Generación de forks por profundidad, para distintos valores de target. Es la cantidad de bloques que pertenecen a cadenas de distinta longitud y que no son parte de la cadena principal.

Cantidad de bloques competitivos					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	más de 4 bloques
10	757	132	11	1	0
8	723	164	14	0	0
6	696	175	28	1	1
4	569	287	41	4	0
2	410	361	109	18	3

Tab. 4.17: Cantidad de bloques competitivos, para distintos valores de target. Es la cantidad de bloques agrupados por cuántos tienen el mismo número. Por ejemplo, en la fila 2, columna 4, se muestra que en 14 ocasiones distintas, 3 mineros hallaron un bloque a la vez.

Esto cierra los experimentos con 200 ms de latencia. En este caso, al ser más marcado el aumento en la latencia, fue más evidente el impacto del tiempo de propagación sobre la cantidad de forks. También es más evidente cómo influye el tiempo de propagación sobre el target: en general, al disminuir el target, la cantidad y profundidad de forks aumenta más si el tiempo de propagación es mayor.

En la siguiente sección se verá un análisis del efecto de tener merged-mining con pools de minería que usan mineros por hardware.

4.3. Pools de Minería

En el capítulo 1 se explicó el proceso de minado, merged-mining y los pools de minería. También se mencionó que RSK funciona haciendo merged-mining, y que era importante garantizarle a los mineros que usar esta técnica no perjudicaría sus ganancias de Bitcoin. Además, se comentó un problema hallado al analizar el proceso de merged-mining entre RSK y Bitcoin: el tiempo de bloque de RSK es mucho menor al tiempo en el que los mineros actualizan voluntariamente su trabajo.

En una red pequeña de RSK en producción, con un solo minero *AntMiner S4*[4], se vió que esto causaba comportamientos atípicos en la red, como por ejemplo, que casi todos los bloques hallados fueran *stale*.

Resultó interesante, entonces, analizar el efecto que tendría tener merged-mining donde todos los mineros utilizaran esta política de actualización en una red más grande. Para ello, se modificó el cliente de *RootstockJ* para que en lugar de minar sobre el último bloque recibido, actualice el bloque sobre el cual minar cada 30 segundos. Esto logra el mismo efecto que se encontró en los mineros por hardware: el trabajo sobre el cual minar se actualiza periódicamente.

La primera prueba se realizó con minado simulado, sin tener en cuenta el ajuste de dificultad. Los parámetros eran: 32 nodos, 10 mineros minando con minado simulado e igual hashing

[4] Hardware diseñado específicamente para minería.

power, 10 segundos de target, conectividad entre 2 y 5 nodos. Se analizaron los primeros 1300 bloques de este experimento.

El tiempo promedio de generación de bloques obtenido fue mayor al doble del target establecido de 10 segundos (23,25 segundos), con una gran cantidad de forks, como se puede ver en la tabla 4.18

Media (ms)	Mediana (ms)	Bloques
23251,32	21501,50	1300/3008

Tab. 4.18: Tiempo de Generación de Bloques. El total de bloques contiene por un lado la cantidad de bloques que formaron parte de la cadena principal, y por otro la cantidad total de bloques, que incluye a los bloques *stale*.

De 3008 bloques, sólo 1300 formaron parte de la cadena principal. Esto significa que un 56,78 % de los bloques minados pertenecían a forks y terminaron siendo *stale*.

Analizando más minusciosamente la generación de forks, en la tabla 4.19 se muestran los forks categorizados por profundidad. Se puede ver que la mayoría de estos son de profundidad 1, y luego hay algunos de mayor profundidad. Creemos que esto último se debe a que al actualizar los bloques para minar de forma tan espaciada, se dan situaciones en donde los conflictos tardan más en resolverse, dando lugar a forks de gran profundidad.

Profundidad 1	Profundidad 2	Profundidad 3	Profundidad 4
1520	91	18	2

Tab. 4.19: Generación de forks (por profundidad)

Por otro lado, en la tabla 4.20 se muestra la cantidad de bloques de la red que son competitivos. Por ejemplo, en 18 ocasiones, más de la mitad de la red estaba compitiendo por un bloque minado. Esta cantidad de bloques competitivos excede por mucho los valores observados en los experimentos de la sección anterior. Notar que esta es una red de menor diámetro y la cantidad de bloques analizados no difiere mucho.

A pesar de la alarmante cantidad de forks, este experimento sirvió para arrojar luz sobre otro problema: debido a que en promedio cada bloque se generaba cada 23 segundos, debería haber

1 bloque	2 bloques	3 bloques	4 bloques	5 bloques	más de 5 bloques
413	409	252	140	69	18

Tab. 4.20: Cantidad de bloques competitivos

ocurrido un ajuste de dificultad para bajarla (recordar que en minado simulado no se tenía en cuenta el ajuste de dificultad). Pero, por otro lado, bajar la dificultad en estas condiciones no hace que se generen nuevos bloques más rápido, ya que el tiempo de actualización de bloques está fijo. Esto causaría que la dificultad baje hasta el mínimo posible definido por la red y aumente la cantidad de forks y bloques *stale* de forma drástica.

En respuesta a esto, se decidió analizar el efecto que tendría un target de 30 segundos, lo que es consistente con el tiempo de refresco de trabajo de los mineros *S4*[5].

Se realizó una corrida igual a la anterior, pero sin minado simulado, con ajuste de dificultad, y con un target de 30 segundos. Se analizaron los primeros 2000 bloques generados.

El resultado fue que la cantidad de bloques *stale* fue muy elevada, pero que la dificultad se mantuvo estable.

La información sobre tiempos de generación de bloques que aparece en la tabla 4.21 muestra que en promedio, se generaron bloques nuevos cada 32,85 segundos, y que también hubo una gran cantidad de forks.

Media (ms)	Mediana (ms)	Bloques Totales
32850,00	28876,0000	2000/3674

Tab. 4.21: Tiempo de Generación de Bloques con minado real y target de 30 segundos. El total de bloques contiene por un lado la cantidad de bloques que formaron parte de la cadena principal, y por otro la cantidad total de bloques, que incluye a los bloques *stale*.

Con respecto a los forks, en la tabla tabla 4.21 se muestra que 1674 bloques terminaron formando parte de forks. Dicho valor representa un 45,56 % de los bloques.

En la tabla 4.22, que muestra los forks categorizados por profundidad, se puede ver que aún aquí hubo gran cantidad de forks.

[5] Este valor fue determinado experimentalmente

De la tabla 4.23 se obtiene un resultado similar: hay muchos menos bloques competitivos con respecto al experimento anterior, pero su cantidad sigue siendo elevada.

Profundidad 1	Profundidad 2	Profundidad 3	Profundidad 4
1557	59	34	0

Tab. 4.22: Generación de forks (por profundidad)

1 bloque	2 bloques	3 bloques	4 bloques	5 bloques	más de 5 bloques
901	691	293	79	29	8

Tab. 4.23: Cantidad de bloques competitivos

Lo importante de este experimento fue ver el efecto que tenía el ajuste de dificultad sobre este entorno. En la figura 4.9 se muestra el tiempo de generación de cada uno de los bloques, y cómo varía la dificultad en función del número de bloque.

Se puede ver que al principio la dificultad aumenta hasta que el promedio de tiempo entre bloques supera al target, y que la dificultad fluctúa pero no se va a hacia el mínimo, que es el problema que ocasionaba tener menor target.

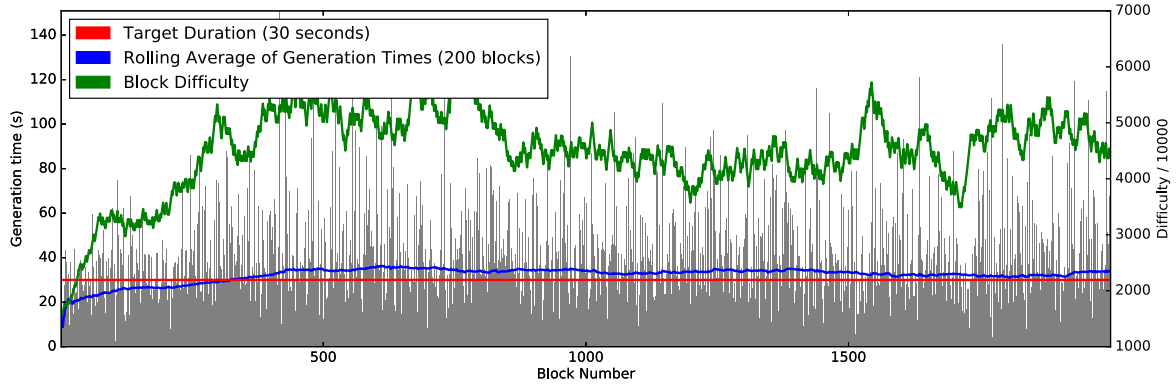


Fig. 4.9: Tiempo de generación de cada bloque con su dificultad para una muestra de 2000 bloques.

Habiendo considerado estos resultados, se decidió cambiar los parámetros de la red de RSK en producción para tener un tiempo de bloque de 30 segundos. Sin embargo, se contemplaron también otra soluciones a largo plazo:

- Considerar a los uncles para el ajuste de dificultad, haciendo que si se generan muchos

no disminuya la dificultad.

- Hacer un mejor análisis sobre el hardware utilizado por los pools de minería. El minero sobre el que se hicieron las pruebas (*AntMiner S4*) ya era obsoleto al momento en que se realizó este trabajo. Es probable que en los dispositivos más nuevos el proceso de actualización ocurra con mayor frecuencia.

4.4. Discusión

Los resultados presentados en la sección de experimentos preliminares muestran que Maxinet es una herramienta viable para distribuir los experimentos entre varias computadoras. Esto nos permitió hacer análisis en redes de mayor tamaño, utilizando el hardware que teníamos a nuestra disposición.

Esta herramienta permite analizar el efecto de realizar cambios en el software de los nodos, sin necesidad de modificar los nodos de la red en producción, dando mayor confianza sobre las posibles mejoras que se puedan realizar sobre los protocolos.

Con respecto al análisis de variación del target, contrariamente a lo esperado, el primer experimento no mostró un gran aumento en la cantidad de forks a medida que el tiempo entre bloques disminuía. Pensamos que esto puede deberse a que el tiempo de propagación era demasiado chico como para influir en red pequeña.

De los siguientes dos experimentos: «Latencia 2x» y «Latencia 200ms» , se pudo ver que al incrementar el tiempo de propagación, el impacto de disminuir el tiempo entre bloques era mayor. De esto se concluye que, contando con una red de mayor tamaño, con un tiempo de propagación comparable con el de la red de Bitcoin o Ethereum, se podría caracterizar mejor el *tradeoff* el target y los forks de la red.

Lo último que se analizó fue el efecto de tener merged-mining con pools de minería que usen mineros con hardware dedicado. La política de refresco de trabajo con la que operaban estos mineros no era compatible con el tiempo entre bloques de RSK. Más allá de la cantidad de forks, el ajuste de dificultad que propone la red la hubiera llevado al colapso. En la red de

RSK se decidió tomar un target de 30 segundos como solución de compromiso, hasta que se pudiera encontrar una solución a largo plazo. Es importante destacar que debido a que los mineros con hardware dedicado están pensados especialmente para minar Bitcoin, que tiene un tiempo entre bloque de 10 minutos, no hay motivo por el cual estos cambien su política de refresco si no resulta más beneficioso económicamente.

5. CONCLUSIONES Y TRABAJO FUTURO

Este trabajo propone una metodología para realizar análisis del funcionamiento de redes de criptomonedas descentralizadas. Para ello, se instrumentó la aplicación de RootstockJ con el objetivo de tomar métricas que ayuden a mejorar el entendimiento del comportamiento de la red.

Entre los experimentos realizados, se analizaron distintas propiedades de la red, como el efecto de la variación del tiempo entre bloques y la simulación de pools de minería con merged-mining.

Corroboramos que disminuyendo el tiempo de generación de bloques a valores muy pequeños, el tiempo de propagación tiene un efecto significativo sobre la cantidad de forks. Por otro lado, la simulación de pools de minería con merged-mining llevó a realizar cambios los parámetros de la red de RSK en producción.

Este trabajo sienta las bases para realizar experimentos de mucho mayor tamaño ya que la herramienta escala horizontalmente.

En nuestros experimentos se analizó la dinámica de la creación de bloques sobre la blockchain, pero no se indagó acerca de los distintos tipos de transacciones dentro de los bloques. Las transacciones pueden ser de distintos tamaños y complejidades, de las cuales los *smart-contracts* son un claro ejemplo.

Nuestro análisis no fue hecho desde el punto de vista de la seguridad. Asumimos que todos los participantes se comportan de forma leal y ejecutan todos el mismo código. Esta plataforma permite realizar experimentos teniendo en cuenta nodos atacantes. Algunos ejemplos de esto son *selfish mining*[1], o ataques de denegación de servicio. Consideramos que esta plataforma es apropiada para validar posibles ataques teóricos, asumiendo que un cierto porcentaje de la red se propone atacarla.

[1] Se dice que un nodo mina de forma egoísta cuando no publica inmediatamente los bloques que mina.

La topología de red utilizada para todos los experimentos se basó en trabajos previos sobre Bitcoin. Resulta interesante realizar un análisis similar sobre las redes de Ethereum y RSK, teniendo en cuenta el efecto del *churn* en la red. Por otro lado, en cuanto a la aplicación, se podrían instrumentar otros eventos, como ser los tiempos de procesamiento de transacciones, bloques, etc. También se podría modificar el proceso de minado para que incluya el ajuste de dificultad real de la red.

El análisis fue hecho sobre la criptomoneda RSK. Mediante esta herramienta es posible realizar el mismo estudio sobre otras criptomonedas más populares, como Bitcoin o Ethereum. Dado que el cliente RootstockJ está basado en EthereumJ, instrumentar este último no supone un gran esfuerzo adicional. Extender este trabajo para que se use el cliente de Bitcoin puede requerir modificaciones más profundas, ya que la arquitectura de la aplicación es muy diferente.

La línea de trabajo de RSK propone algunos cambios en los protocolos de consenso y red para disminuir la cantidad de forks y mejorar los tiempos de propagación. Estos cambios implicarían implementar *«DECOR+»* y *«headers first»*.

En nuestras pruebas, el hashing power estuvo distribuido equitativamente, y la cantidad de mineros fue proporcional al tamaño de la red. Para obtener un modelo más completo del funcionamiento total de la red, se debería contar con una mejor caracterización de la distribución de mineros y el hashing power.

Finalmente, se realizó una prueba de concepto para simular redes de gran tamaño utilizando `SimGrid`. Este enfoque quedó fuera del alcance del trabajo, pero creemos que es una línea interesante para explorar, ya que los resultados preliminares obtenidos fueron muy prometedores. Como siguiente paso, sería necesario ajustar los parámetros del modelo de red y validarlos repitiendo los experimentos de este trabajo.

Bibliografía

[ant] Antminer s4 review. https://www.cryptocoinsnews.com/bitmain-antminer-s4-review-2ths-bitcoin-asic-miner-psu-replacement/. Accessed: 2016-11-31.

[Ant14] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.

[bit] Global bitcoin nodes distribution. https://bitnodes.21.co/. Accessed: 2016-09-30.

[CDE+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, and Emin Gün. On scaling decentralized blockchains. In *Proc. 3rd Workshop on Bitcoin and Blockchain Research*, 2016.

[cmp] Bip 152 - compact blocks. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki. Accessed: 2016-10-10.

[DPSHJ14] Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The bitcoin p2p network. In *International Conference on Financial Cryptography and Data Security*, pages 87–102. Springer, 2014.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[flo] Project floodlight. http://www.projectfloodlight.org/floodlight/. Accessed: 2016-09-30.

[FOAa] Muntadher Fadhil, Gareth Owen, and Mo Adda. Bitcoin network measurements for simulation validation and parameterisation.

[FOAb] Muntadher Fadhil, Gareth Owenson, and Mo Adda. A bitcoin model for evaluation of clustering to improve propagation delay in bitcoin network.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[GKW+] Arthur Gervais, Ghassan O Karame, Karl Wust, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains.

[JH12] Rob Jansen and Nicholas Hopper. Shadow: Running tor in a box for accurate and efficient experimentation. In *Proceedings of the 19th Symposium on Network and Distributed System Security (NDSS)*. Internet Society, February 2012.

[KK95] George Karypis and Vipin Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[KP15] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Technical report, IACR: Cryptology ePrint Archive, 2015.

[KP16] Aggelos Kiayias and Giorgos Panagiotakos. On trees, chains and fast transactions in the blockchain. 2016.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[MLP+15] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering bitcoin's public topology and influential nodes, 2015.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[slu] Stratum protocol - slush pool. https://slushpool.com/help/#!/manual/stratum-protocol. Accessed: 2016-11-31.

[SZ13] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *IACR Cryptology ePrint Archive*, 2013:881, 2013.

[WDS14] Philip Wette, Martin Dräxler, and Arne Schwabe. Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[xth] Towards massive on-chain scaling: Presenting our block propagation results with xthin. https://medium.com/@peter_r/towards-massive-with-xthin-da54e55dc0e4#.kk1znlg1v. Accessed: 2016-10-10.