Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

Sobre los límites del tiempo entre bloques en Bitcoin

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Nicolás De Carli

Director: David Alejandro González Márquez

Codirector: Maximiliano Iván Geier

Buenos Aires, 2019

Resumen

Desde su aparición en 2008, Bitcoin luchó por ganar adopción en el sistema financiero mundial. Ostentando la principal novedad de poder operar sin la necesidad de una autoridad central, se mostró como una alternativa moderna a los esquemas bancarios tradicionales. Uno de los grandes problemas observados durante el auge de las criptomonedas, fue su limitada capacidad en cuanto al volumen de transacciones que podía procesar, especialmente comparado con los sistemas de pago convencionales.

En este trabajo exploramos la posibilidad de modificar los parámetros del protocolo de Bitcoin, con el fin de incrementar la cantidad de transacciones por segundo que el sistema puede procesar.

Se modificó el cliente oficial de Bitcoin para ejecutar experimentos en ambientes controlados. Además, se presenta un modelo de red a escala, que recrea la distribución geográfica y de *hashing power* de los clientes la red de Bitcoin real. Se logró emular, con los recursos disponibles, redes de más de 240 nodos, distribuidos a través de 37 países. Dicha infraestructura permite estudiar el comportamiento de la red para distintos volúmenes de carga. Para evaluar estas configuraciones, se introduce una magnitud que permite cuantificar el porcentaje de recursos desperdiciados en la red: *wasted hashing power*.

Los resultados obtenidos indican que las opciones de configuración de la red Bitcoin no resultan adecuadas para maximizar la cantidad de transacciones a procesar, siendo posible mejorar la cantidad de transacciones con un *wasted hashing power* equivalente al sistema actual.

Abstract

Since it was launched back in 2008, Bitcoin strove to gain adoption within the global financial system. By flaunting its novel feature of being able to operate without a central authority, it has been advertised as a modern alternative to the traditional banking systems. A serious problem that arose during the boom of cryptocurrencies was their inability to process huge amounts of transactions, specially when compared to conventional payment systems.

Throughout this work we explore the feasibility of increasing the amount of transactions per second the system can process by applying modifications to the Bitcoin's protocol.

A series of modifications were made to Bitcoin's official client that allowed us to perform experiments in controlled environments. A scaled Bitcoin network model is introduced, which reproduces the geographic and hashing power properties of the clients of the real network. We achieved the emulation of scenarios containing more than 240 nodes distributed across 37 countries. Such infraestructure gives the capability of studying the behaviour of the network under various load levels. In order to assess these configurations, we introduce a metric called *wasted hashing power* that estimates the percentage of the network's resources that were wasted.

Results suggest Bitcoin's current difficulty configuration is not optimal regarding to maximizing the network's throughput, as by tweaking such configuration it is possible to increase the amount of transactions per second the system processes while maintaining similar levels of wasted hashing power.

Índice general

Índice de figuras

Capítulo 1

Introducción

Desde su aparición en 2008, Bitcoin luchó por ganar adopción en el sistema financiero mundial. Ostentando la principal novedad de poder operar sin la necesidad de una autoridad central, se mostró como una alternativa moderna a los esquemas bancarios tradicionales. En los últimos años, las criptomonedas basadas en *blockchain* lograron acaparar una enorme atención, haciendo que su capitalización de mercado conjunta alcance los 835,6 mil millones de dólares el 7 de Enero de 2018. Exactamente un año después, este valor se vio reducido a 104 mil millones de USD.

Uno de los grandes problemas observados durante el auge de las criptomonedas fue la incapacidad de competir con sistemas de pago convencionales. Entre Diciembre del 2017 y Enero del 2018, se registró el mayor uso sostenido de Bitcoin de la historia, en cada segundo miles de operaciones nuevas eran solicitadas por usuarios alrededor del mundo. Sin embargo, el sistema nunca logró llegar a procesar ni siquiera diez transacciones por segundo, haciendo que el 24/01/2018 los pagos emitidos tarden en promedio 33,15 horas en ser confirmados[1].

Además, al haber muchas operaciones pendientes de confirmación, la mediana de las comisiones de transacciones llegó a ser 34 USD a fines de Diciembre del 2017, teniendo como

[1] https://cnb.cx/2DxQexO

consecuencia que sea imposible usar Bitcoin como un medio de pago para consumos de bajo valor. Por otro lado, sistemas de pagos basados en tarjetas bancarias procesan cotidianamente alrededor de 2000 transacciones por segundo, mientras mantienen una comisión fija menor al 4 % del valor de la compra. Esta disparidad entre los modelos mencionados deja mucho trabajo de investigación para realizar, teniendo como fin medir cuán posible es achicar la brecha sin recaer en un esquema centralizado.

Implementamos en el cliente de Bitcoin un modelo de minado simulado que nos permite reproducir el comportamiento de la red real sin utilizar grandes cantidades de poder de cómputo. También desarrollamos un modelo a escala que emula la topología de la red real de Bitcoin. De esta manera, utilizando pocos recursos de hardware, recreamos escenarios complejos con rasgos similares a los que podrían darse en la red real.

1.1 Sobre Bitcoin

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

La transferencia de una moneda digital puede pensarse como el endosado de un cheque. Una persona puede escribir en el dorso del documento el nombre del destinatario, el cual podría endosarlo nuevamente si así lo desease. También es posible saber si la última persona en endosar el cheque es el dueño original del documento. 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, validando 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. En la misma 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 acreedor y un tercero.

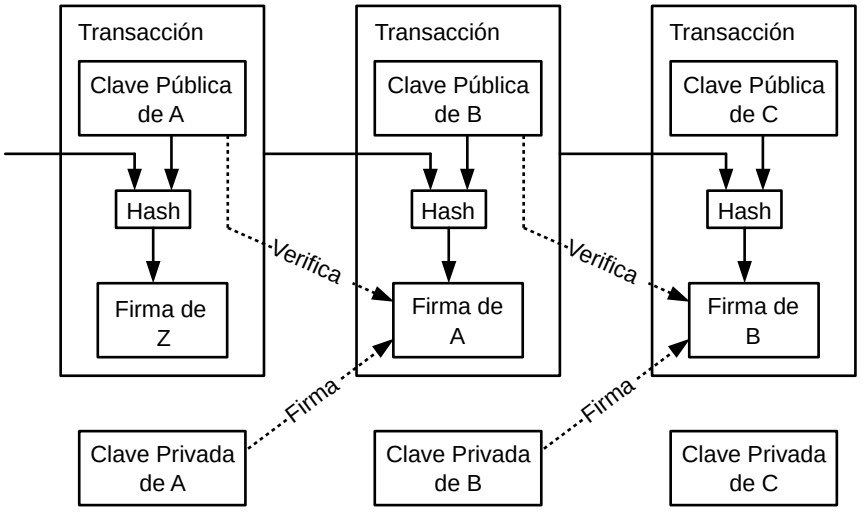


Figura 1.1: Esquema de transacciones en una criptomoneda [Nak08].

Esto deja lugar a un problema conocido como *double spending (doble gasto)*: el emisor podría crear más de una transacción a partir de una misma operación previa. En este escenario sólo una de las nuevas transacciones debería ser válida, ya que de lo contrario el emisor estaría multiplicando dinero. Una solución a este problema podría ser contar con una entidad de confianza que se encargue de validar todas las transacciones.

El sistema de Bitcoin fue propuesto en el 2008 por una persona bajo el seudónimo «Satoshi Nakamoto» [Nak08]. Fue la primer criptomoneda descentralizada que propuso una solución al problema de *double spending* sin necesidad de que haya una *tercera parte de confianza (trusted third-party)*. La idea principal consiste en que exista un único registro de todas las transacciones que fueron procesadas, en el cual cualquier usuario puede agregar operaciones. Todos los nodos de la red van a guardar una copia de dicho registro.

Este registro replicado y distribuido se conoce como **Blockchain**, el cual consiste en un árbol que contiene transacciones en cada uno de sus nodos. Luego, si un usuario introduce una nueva transacción a la red, es fácilmente detectable si en la Blockchain existe otra

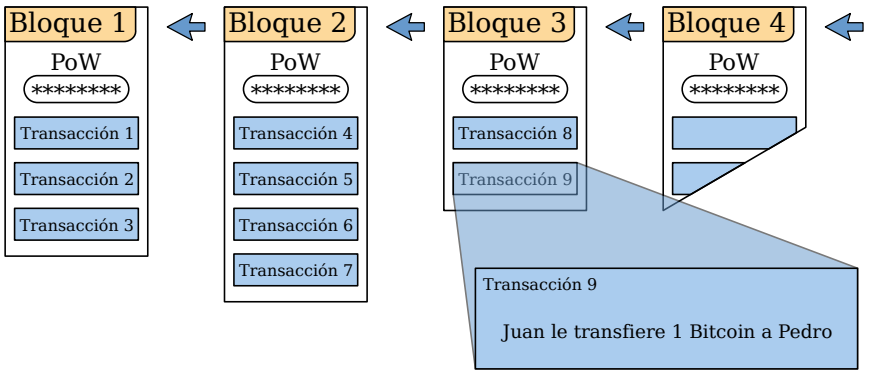


Figura 1.2: Esquema de blockchain con transacciones.

operación que hace referencia a la misma transacción «padre». Si esto último ocurre, se considera que el usuario está intentando hacer double spending y la transacción es inválidada por los nodos de la red. En la figura 1.2 se puede apreciar una esquematización de la blockchain.

1.1.1 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 los conceptos detrás de ellas.

1.1.1.1 Transacciones

Una transacción en Bitcoin está definida por una lista de *inputs* y una de *outputs*. Un output consiste en un par *(cantidad, dirección)*, 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. Las definiciones de las propiedades son las siguientes:

- 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, cada input es una referencia a un output anterior en la blockchain. Para verificar que el uso del output es legítimo, se calcula el hash de la clave pública y se verifica que sea igual a la que figura en el output usado. Luego, basta con verificar la firma digital con esa clave pública para asegurarnos la autenticidad de la operación.

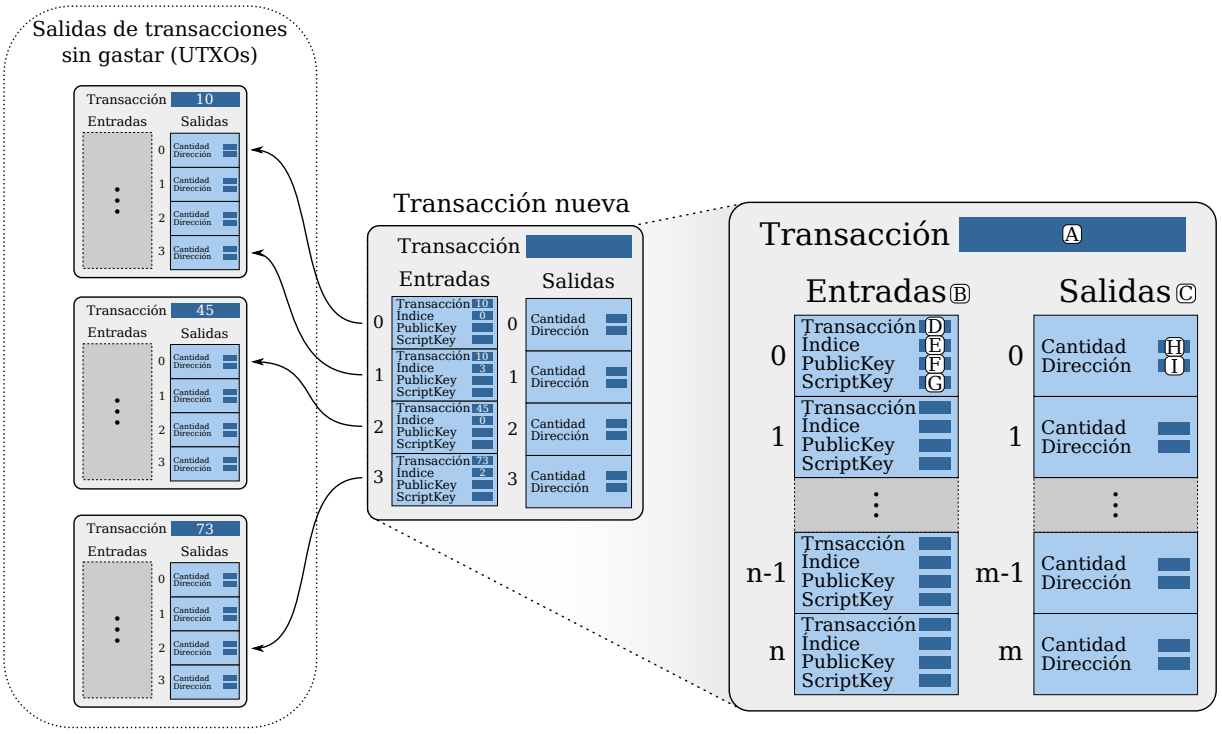


Figura 1.3: Ejemplo de una transacción en Bitcoin, sus inputs hacen referencias a outpus de transacciones anteriores.

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

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

(B) La lista de inputs de la transacción, donde cada elemento hace referencia a un output de alguna transacción previa que todavía no haya sido 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 utilizado de la transacción referenciada.

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

(G) La firma digital que demuestra que se tiene control sobre los fondos utilizados.

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

(I) La dirección destino, es decir, 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 válida, el script debe ejecutarse satisfactoriamente. Cabe mencionar que estos scripts están limitados en el sentido en que no tienen ciclos, restringiendo el poder de expresividad.

Para poder afirmar que una transacción es válida, se debe verificar que todos los inputs también lo son, y que la suma de los outputs referenciados por esos inputs es mayor a la suma de los outputs de la transacción. La diferencia entre ambas sumas podrá ser reclamada por quien agregue la transacción a la Blockchain. Este dinero extra es lo que se conoce como *«transaction fees» (comisión).*

Una vez que un output de una transacción fue utilizado, ya no se puede volver a usar. Es decir, sólo se pueden incluir en nuevas transacciones outputs que no fueron utilizados previamente. Estos últimos se conocen como *«Unspent Transaction Outputs» (UTXOs).*

1.1.1.2 Bloques

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

Visto desde otro modo, si tomamos la Blockchain como un árbol, los bloques son sus nodos.

Cada bloque está compuesto por:

- El *hash* del bloque anterior (su padre).
- Tiempo en segundos en que fue minado, contados a partir del *epoch*[2].
- La dificultad con la que fue hallado.
- `nonce`: Un campo en el que se puede poner cualquier número, permitiendo así modificar el resultado del cálculo del hash del bloque.
- Un conjunto de transacciones ordenadas.
- El número de versión.

No toda combinación de valores en estos parámetros resulta en un bloque válido. Para que un bloque sea válido y aceptado por la red debe contar con una prueba de trabajo, la cual se conoce como *Proof of Work (PoW)*. Ésta debe ser difícil de calcular pero fácil de verificar. El mecanismo que se usa para hallar y validar el `PoW` se conoce como *proceso de minado*, y será explicado más adelante.

Los bloques se separan en *header* y *body*. En Bitcoin, el body consiste en un *Merkle tree* cuyas hojas son los hashes de las transacciones contenidas en el bloque. Por otro lado, el header incluye el resto de los campos mencionados anteriormente, además del hash de la raíz del Merkle Tree del body.

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 1.4 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.

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

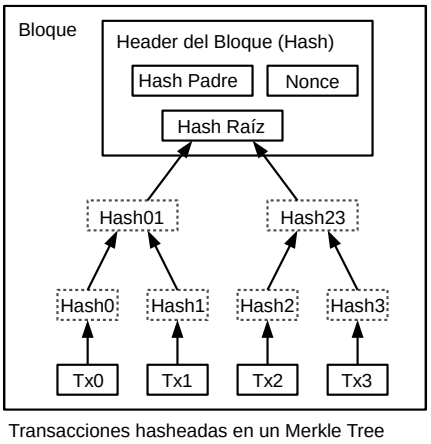


Figura 1.4: Ejemplo de Bloque con Merkle Tree [Nak08].

1.1.1.3 Blockchain

La estructura de la Blockchain puede verse como un árbol de Bloques. No puede ser considerada una lista, ya que podría suceder que dos bloques tengan el mismo padre, dando lugar así a ramificaciones. La Blockchain tiene una cadena[3] 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 de Bitcoin es un bloque especial llamado *Bloque Génesis*, el cual fue minado el 3 de enero de 2009. Este bloque contiene en uno de sus campos un titular del diario *«The Times»* de esa fecha[4], para dar prueba que este bloque no fue creado antes de ese día.

El número de un bloque es la altura del bloque desde la raíz de la Blockchain. Los nodos que construyen bloques son denominados *nodos mineros*. Cualquier cliente de la red puede tomar este rol. Al recibir un bloque, los nodos lo validan y lo agregan a su Blockchain. Si el nuevo bloque tiene como padre a un bloque stale, puede cambiar la cadena que el nodo considera como principal. Este escenario ocurre si la adición de dicho bloque genera una cadena con mayor dificultad total que la previamente considerada como principal.

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

1.1.2 Proceso de minado y consenso

Los mineros obtienen una recompensa en Bitcoins cuando agregan un bloque a la cadena principal de la Blockchain. Un minero debe calcular satisfactoriamente la PoW de un nuevo bloque para poder incluirlo en la Blockchain, gastando poder de cómputo en el proceso. La Proof of Work del bloque es un número tal que **SHA256(SHA256(header))**[5], tiene una cantidad dada de 0's en sus bits más significativos. En bloques válidos, esta cantidad es mayor o igual a un número que se deriva del campo *dificultad* del *header* del bloque.[6]

La motivación de este proceso yace en el costo computacional de obtener un hash que cumpla los requisitos pedidos. SHA es una función criptográfica con la propiedad de que no existe una manera directa de calcular la función inversa, y por lo tanto la única forma de encontrar un hash válido es recalcularlo varias veces cambiando valores del header, hasta encontrarlo. Los hashes calculados están uniformemente distribuidos, por lo cual, la probabilidad de que un hash (256 bits) comience con d ceros, es $\frac{2^{256-d}}{2^{256}}$.

El esfuerzo necesario para armar un bloque que cumpla con la dificultad de la red crece exponencialmente con la cantidad de ceros requerida. Esto se debe a que agregar un cero extra a la dificultad disminuye a la mitad la cantidad de hashes que cumplan con esa restricción. Sin embargo, para verificar que un bloque cumple con esta propiedad, alcanza con calcular dos veces el `SHA256` del *header*. De esta forma, se puede comprobar fácilmente que un minero realizó una cierta cantidad de trabajo para hallar un bloque.

Es importante destacar que una PoW solo sirve para un bloque, ya que ésta 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 antelación.

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

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

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

la dificultad que deben cumplir los bloques para ser válidos. Para esto, la red define un *target*, que es el tiempo promedio en que se espera que aparezcan nuevos bloques en la red (en Bitcoin es 10 minutos). Cada 2016[7] bloques, se cuenta cuántos aparecieron en menos tiempo que el target, y en función de esto se aumenta o disminuye la dificultad hasta cuatro veces el valor anterior. Esto se traduce en que el hash del header del bloque comenzará con hasta dos ceros más o menos que el padre, según aumente o disminuya la dificultad, respectivamente.

Bitcoin provee incentivos para que los mineros colaboren con la red. Por un lado, el minero que construye un bloque válido es acreedor de todas las comisiones de las transacciones incluidas en él. Por el otro, cada bloque cuenta con una transacción especial llamada *coinbase*, la cual tiene un solo output y carece de inputs. El minero tiene control total sobre los campos de esta transacción. Por lo tanto, el nodo que descubre un bloque puede asignar una dirección de su billetera al output de la *coinbase*, convirtiéndose así en acreedor del valor de la transacción. Esta es la única forma de crear Bitcoins, y es por eso que se conoce al proceso como «Minar Bitcoins», ya que el minero en realidad está introduciendo nuevos Bitcoins al sistema. El output de la *coinbase* sólo puede ser usado luego de que hayan aparecido 100 bloques por encima del que la contiene. Esta política trata de evitar que se obtenga dinero a partir de bloques *stale*. Sin embargo, esta decisión de diseño no es parte del protocolo de consenso de la red. Un usuario puede gastar ese crédito, sabiendo que en un futuro cercano podría quedar invalidado.

La cantidad de Bitcoins otorgada por minar bloques disminuye en función del número de bloque. Cada 210000 bloques (4 años aproximadamente) se divide a la mitad, comenzando con un valor de 50BTC por bloque. Esta política de reducción resulta en que existe una cota sobre la cantidad total de Bitcoin que pueden existir en el sistema, la cual es aproximadamente 21 millones. A abril de 2019, la recompensa por hallar un bloque es de 12,5BTC (equivalente a U$S66000).

Cuando un minero encuentra un hash que hace válido a un nuevo bloque, propaga dicho

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

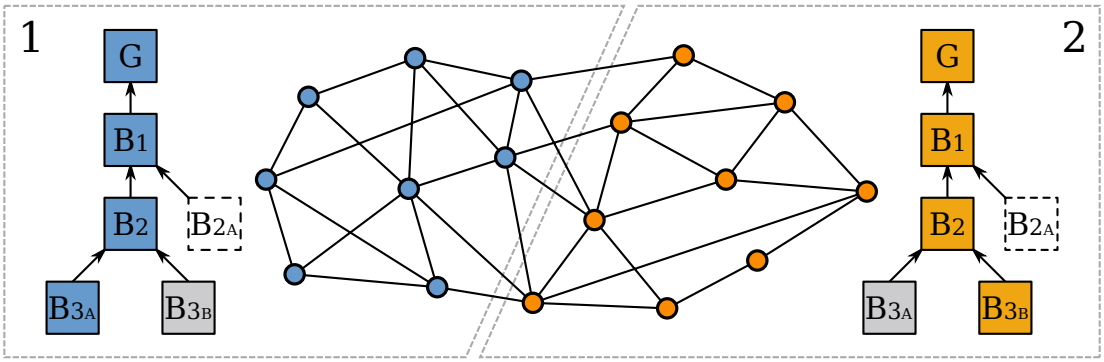


Figura 1.5: Ejemplo de una partición de los nodos de la red. El grupo de nodos 1 considera al bloque B_{3A} como bloque de su cadena principal, mientras que el grupo 2 considera a B_{3B}. El bloque B_{2A} quedó *stale* luego de que se minaron los bloques B_{3A} y B_{3B}.

hash por toda la red. Luego los demás nodos piden el bloque completo, lo validan, lo agregan a la Blockchain y lo propagan de la misma manera. [8]

Debido a la naturaleza distribuida de Bitcoin, es posible que dos nodos distintos de la red encuentren bloques válidos diferentes con la misma altura dentro de la Blockchain. Esto sucede cuando un minero genera un bloque antes de que sea notificado de la existencia de otro bloque con la misma altura. Ante esta situación, los nodos de la red deberán decidir cuál de los dos bloques forma parte de la cadena principal. 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*). Esto es lo que se conoce como *Protocolo de Consenso*.

Esta situación deriva en una partición en los nodos de la red, como se puede observar en la figura 1.5, 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 ésta, tendrá una mayor dificultad acumulada y, por lo tanto, será la aceptada

[8] 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*.

por 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 agregar un bloque cuyo padre sea el mismo que el del bloque que la contiene y, adicionalmente, crear tantos bloques como los que hayan sido agregados en la cadena original. 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 no se conocen mecanismos para mitigarlo.

Realizar cambios al protocolo require el consenso de los nodos de la red, ya que cada cliente funciona de manera autónoma. Ante una nueva versión del software que introduce este tipo de cambios, todos los clientes deberían actualizarse para poder funcionar bajo las nuevas reglas. Si solo una parte de los nodos lo hiciese, se podría producir una partición en la red entre los nodos que actualizaron y los que no lo hicieron. A esta situación se la conoce como *hard fork*.

1.2 Sobre el throughput de Bitcoin

En esta sección mostramos que el protocolo de Bitcoin impone un límite sobre la cantidad de transacciones por segundo que puede procesar. Luego exploramos posibles formas de

aumentar dicho límite, y fundamentamos la elección de la forma estudiada en el presente trabajo. Por último, explicamos en detalle por qué particularmente decidimos no elegir aumentar el tamaño de bloque como forma de incrementar el *throughput* del sistema.

1.2.1 Limitaciones del sistema

Como vimos en secciones anteriores, Bitcoin espera que aparezca un bloque nuevo en la red cada diez minutos. Existe una máxima cantidad de transacciones que cada uno puede contener, ya que el tamaño de estos bloques está acotado. Actualmente, un bloque puede incluir hasta 7483 transacciones, restringiendo solo a las más pequeñas que se pueden construir. Luego, es fácil ver que el sistema de Bitcoin, como máximo, puede esperar procesar $7483/600 = 12{,}47$ transacciones por segundo. Además, este escenario es posible si y solo si todas las operaciones utilizadas son de tamaño mínimo, que es una situación más bien inusual en la vida cotidiana. Por lo tanto, la cota de uso real es todavía menor.

Es claro entonces, que para poder agregar a la blockchain más de 12,47 transacciones por segundo, es necesario realizar cambios en el protocolo.

1.2.2 Analizando las variables del sistema

En Bitcoin hay tres parámetros del protocolo que son elegidos por la comunidad: el tiempo entre bloques esperado, el tamaño máximo de bloque y el tamaño de las transacciones. El primero se establece indirectamente mediante la dificultad de la red elegida, los otros dos son límites acordados y fijados en el software que implementa el protocolo.

Además de los parámetros, existen características que determinan la topología de la red. Principalmente, estos factores son: la cantidad de nodos, la manera en que estos se conectan, y la latencia entre ellos. En la sección 2.3 presentamos un modelo de red a escala que captura la conectividad y la latencia de la red real de Bitcoin. Utilizamos este modelo para la realización de los experimentos, de forma que la topología de red tenga rasgos similares a los de la realidad.

Los parámetros y características que se mencionan en los últimos dos párrafos pueden ser vistas como variables primitivas de la red. También es posible definir variables de red a partir de otras variables ya descriptas. Por ejemplo, podemos definir una variable como la máxima cantidad de transacciones que entran en un bloque. Esta métrica se define a partir de una fórmula cerrada, ya que resulta de dividir el tamaño máximo de bloque por el tamaño mínimo de las transacciones.

También podemos definir variables a partir de fenómenos observables, por ejemplo, podemos establecer "Tiempo de propagación" (descripto en detalle en la sección 2.4.2) como el tiempo que transcurre entre que un bloque es descubierto y los nodos de la red lo aceptan. Esta variable depende de otras dos: la cantidad de transacciones por bloque y la topología de la red. La primera dependencia se genera porque el protocolo de Bitcoin envía las transacciones de bloques en mensajes individuales. Mientras que la segunda se debe a que, teóricamente, los tiempos de propagación de la red están dados por dos cosas: la cantidad de información que hay que transmitir y la red subyacente donde se transmite. Si bien podemos establecer estas dependencias, conociendo la topología de red y la cantidad de transacciones de un bloque, no es posible determinar una expresión para calcular cuál será el tiempo de propagación de ese bloque. Esto sucede porque hay factores dinámicos de la red que también influyen en los tiempos de propagación.

El máximo *throughput* esperado puede definirse como la mayor cantidad de transacciones por segundo que el sistema puede aspirar a incorporar a la blockchain de manera sostenida. Esta variable deriva del tiempo entre bloques y de la máxima cantidad de transacciones que entran en un bloque.

Visto de esta forma, podemos modificar algunos de los parámetros del protocolo para que indirectamente aumenten el *throughput* del sistema, a pesar de que no dependa directamente de las variables alteradas. Sin embargo, al cambiar los parámetros del protocolo también pueden verse modificadas otras variables, resultando en un deterioro de distintos aspectos del sistema. Por ejemplo, la probabilidad de que haya un *fork* en la *mainchain* depende de los tiempos de propagación y del tiempo entre bloques. Como veremos en detalle en la

sección 2.4.3, la cantidad de *forks* y los tiempos de propagación van a determinar el *wasted hashing power* de la red, es decir, el tiempo en el que los mineros consumieron energía pero no aportaron a la convergencia de la red.

En la figura 1.6 podemos ver un resumen de todas las variables y sus dependencias.

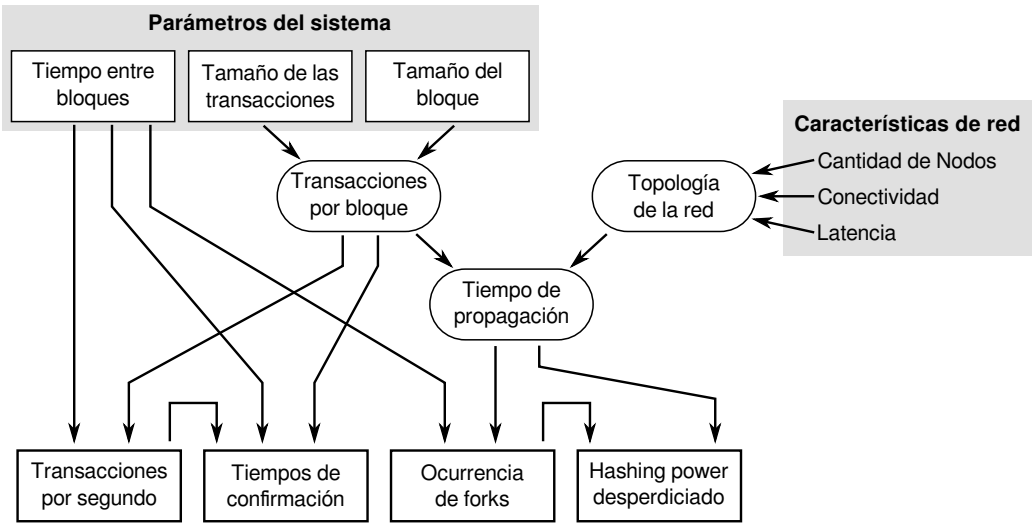


Figura 1.6: Diagrama de dependencias de variables en el sistema de Bitcoin

1.2.3 Explorando posibles soluciones

En los últimos años surgieron diversas criptomonedas con el objetivo principal e ofrecer un *throughput* mayor al de Bitcoin. Éstas proponen sistemas de pagos descentralizados estructuralmente distintos. Por ejemplo, `Ripple` promete poder procesar 1500 transacciones por segundo.

Este trabajo no apunta a explorar nuevos esquemas, sino a evaluar distintas configuraciones del modelo actual de Bitcoin. Es decir, vamos a experimentar cambiando distintos parámetros del protocolo de Bitcoin vigente. Como la cantidad de transacciones por segundo que procesa el sistema se desprende de la cantidad de transacciones por bloque y del tiempo entre bloques, cualquiera de los siguientes cambios resultaría en un aumento del *throughput* del modelo:

1 Aumentar el máximo tamaño de bloque.

2 Disminuir el tamaño de las transacciones.

3 Reducir el tiempo entre bloques.

La primera opción, aumentar el máximo tamaño de bloque, ya fue propuesta hace unos años y no tuvo una buena recepción por parte de la comunidad. Además, implementar esta opción dificultaría la escalibilidad del sistema desde el punto de vista de software. En la sección 1.2.4 se discuten estos puntos.

Hay dos maneras de abordar el segundo caso: (i) Se puede rediseñar la estructura de las transacciones para que ocupen menos, o (ii) Se puede establecer un límite menor para el tamaño máximo de una transacción. Plantear una nueva estructura para las transacciones requiere el estudio y la aplicación de técnicas criptográficas para garantizar la seguridad de las mismas. Dicha reingeniería se encuentra fuera del alcance de este trabajo. Con respecto a disminuir el tamaño máximo de la transacción, en la práctica esto resultaría en reducir la cantidad máxima de inputs y outputs que pueden convivir en la misma. Sin embargo, recordemos que el *throughput* máximo teórico se calcula a partir de transacciones de tamaño mínimo, que consisten en una persona enviándole dinero a otra. Por lo tanto, disminuir el tamaño máximo de transacción no impacta en el *throughput* máximo teórico del sistema Bitcoin. Además, reducir la cantidad máxima de inputs y outputs posibles en una transacción cambia el conjunto de operaciones que se pueden realizar en Bitcoin, ya que algunas de las características utilizadas actualmente por entidades bancarias requieren una alta cantidad de inputs y outputs, resultando en transacciones de tamaño cercano al máximo. Como apuntamos a incrementar el throughput máximo teórico de Bitcoin, y como no queremos sacrificar parte de la funcionalidad del sistema Bitcoin, la segunda opción no resulta adecuada para el presente trabajo.

La tercera opción es atractiva en la teoría, ya que trae consigo dos consecuencias favorables. Dividir el tiempo entre bloques por un número x, resulta en multiplicar el *throughput* máximo del modelo por x. Además, disminuir el tiempo entre bloques trae un beneficio, que aumentar la cantidad de transacciones por bloque no: Se reducen los tiempos de con-

firmación. Es decir, reducir el tiempo entre bloques no solo tiene el potencial de aumentar la escalabilidad del sistema, sino que además podría mejorar la experiencia de usuario.

Sin embargo, con respecto a consecuencias negativas, no es claro cómo la disminución del tiempo entre bloques afecta a la cantidad de *forks* y al *wasted hashing power*. En el presente trabajo nos inclinamos por la opción 3, es decir, experimentar con distintas reducciones al tiempo entre bloques, con el objetivo de estudiar cómo se comportan distintos aspectos de la red bajo los cambios propuestos. De esta manera vamos a analizar cuán viable es aumentar el *throughput* de Bitcoin mediante la reducción del tiempo entre bloques.

1.2.4 Sobre la viabilidad de aumentar el tamaño de bloque

Con el fin de incrementar el *throughput* de Bitcoin, a mediados del 2017 se planteó formalmente aumentar el máximo tamaño de bloque permitido. La propuesta generó mucha polémica ya que fue recibida de maneras muy contrastantes, una gran parte de la comunidad quería adoptar esta nueva medida, mientras que los demás se oponían firmemente a su implementación. Finalmente, el primero de Agosto de 2017 se produjo un *hard fork* de la blockchain. Bitcoin continuó funcionando normalmente sin cambios en su protocolo, pero una gran parte de sus mineros empezaron a operar utilizando el nuevo tamaño máximo de bloques. *Bitcoin Cash - BCH* fue el nombre elegido para la criptomoneda del sistema comprendido por los nodos que adoptaron las nuevas normas.

Al momento de la creación de BCH, su capitalización de mercado fue de aproximadamente 7 mil millones de dólares. Por otro lado Bitcoin tenía esa cifra en alrededor de 45 mil millones de USD. En todo momento, desde su creación hasta Febrero de 2019, BCH tuvo un valor menor al 30 % del de BTC. Además, la última vez que un BCH cotizaba a más del 20 % de un BTC fue en Diciembre del 2017, y la última vez que un BCH valía más que el 10 % de un BTC fue en Agosto de 2018. Es decir, no solo que la comunidad respondió dándole mucho más respaldo económico al protocolo tradicional de Bitcoin, sino que el valor de BCH con respecto al de BTC fue decrementando en el tiempo. Al 1 de Febrero de 2019, 1 BTC equivale a aproximadamente 30 BCH.

Actualmente, la media de tiempo entre bloques en Bitcoin tiende a 10 minutos. Sin embargo, como los tiempos entre bloques siguen una distribución exponencial, la mediana de tiempo entre bloques tiende a $\ln(2) \cdot 10 \approx 6{,}93$ minutos. Es decir, se espera que la mitad de todos los bloques tarden menos de 7 minutos en aparecer. Otro dato a tener en cuenta, es que se espera que más del 63 % de todos los tiempos entre bloques sean menores a 10 minutos. El hecho de que muchos bloques aparezcan antes de los 10 minutos sugiere que tiene que haber algunos que tardan mucho más en ser minados, ya que si no la media estaría por debajo de los 10 minutos. De hecho, el percentil 90 de los tiempos entre bloques es apenas más de 23 minutos. Visto de otro modo, se espera que un décimo de todos los bloques tarden más de 23 minutos en aparecer.

Bitcoin espera que sean minados 144 bloques por día. Si ordenamos los tiempos en que cada uno tardó en aparecer, se espera que el 6^{to} tiempo más alto tienda a ser aproximadamente 31,8 minutos. Es decir, 6 veces por día (o 1 vez cada 4 horas), debería haber un intervalo de más de media hora sin bloques nuevos. Más aún, realizando el mismo análisis con el máximo valor de los 144 tiempos entre bloques, llegamos a la conclusión de que se espera que una vez por día haya un bloque que tarde más de 49,6 minutos en aparecer.

Pensemos ahora en el escenario en el que el sistema se encuentra operando cerca de su *throughput* máximo teórico, y para algún número x, pasan $x \cdot 10$ minutos sin que aparezcan bloques. En dicho caso, el cliente va a tener que almacenar en memoria aproximadamente x veces la cantidad de transacciones que entran en un bloque. O sea que necesariamente el cliente actual tiene que ser diseñado para poder operar teniendo como pendientes 5 veces la cantidad de transacciones que entran en un bloque, ya que en condiciones de uso intenso, esa situación debería suceder una vez por día. Por otro lado, cada vez que un cliente recibe una transacción, tiene que compararla con todas las que tiene almacenadas hasta ese momento, para asegurarse de no incluir dos transacciones incompatibles en el bloque que está minando. Por lo tanto, el costo computacional de incorporar una nueva transacción aumenta linealmente con la cantidad de transacciones pendientes.

Notemos que, si se duplicara el tamaño máximo, entonces al menos una vez al día se espera

que el cliente tenga como pendientes diez veces la cantidad de transacciones que actualmente entran en un bloque. Obviamente, el cliente tendría que estar diseñado para poder soportar esa cantidad de transacciones pendientes, lo que podría significar un aumento del requerimiento de almacenamiento del programa. Además, se esperaría que, al menos una vez al día, el tiempo que tarde el cliente en incorporar una nueva transacción llegue a aumentar en un orden de magnitud. Es decir, aumentar el tamaño máximo de bloque dificulta la escalabilidad del software de los clientes. Por otro lado, disminuir el tiempo entre bloques resulta en una menor probabilidad de ver tiempos entre bloques tan altos, y por lo tanto facilita la escalabilidad desde el punto de vista del software.

En resumen, aumentar el tamaño de bloques introduce complicaciones de escalibilidad, que es lo contrario a lo que queremos lograr. Además, ya se probó hacerlo, y la comunidad se mantuvo fiel al tamaño de bloque tradicional. Por lo tanto, no parece interesante explorar la posibilidad de aumentar el *throughput* del protocolo incrementando el tamaño de bloque.

1.2.5 Redes de retransmisión

En los últimos años aparecieron lo que se conoce como *"Redes de retransmisión"*. Éstas se componen por un grupo reducido de nodos que se dedican únicamente a retransmitir, a todos sus *peers*, los bloques y transacciones que les llegan del resto de la red. Las redes de retransmisión apuntan a que todos los clientes de la red estén conectados a alguno de sus nodos. Si se sigue esta estrategia, se reduce drásticamente el diámetro de la red, el cual influye en los tiempos de propagación, que a su vez influyen en la aparición de *forks*.

Un ejemplo notorio es la llamada *"Red Fibre"*, la cual está compuesta por seis servidores distribuidos a través del hemisferio Norte. Fibre funciona como un agregado a la tradicional red P2P de Bitcoin, es decir, los clientes que se conectan a alguno de los nodos de Fibre no dejan de usar el método estándar para establecer conexiones con otros clientes de la red. Entonces, si en algún momento la red Fibre dejara de funcionar, los clientes no perderían su conectividad, ya que estarían en condiciones de utilizar los enlaces originales.

La introducción de Fibre trajo resultados positivos: los tiempos de propagación se redujeron en un orden magnitud. Sin embargo, su utilización es polémica, principalmente por dos motivos. Por un lado, la red impone diferentes condiciones de latencia dependiendo del lugar del mundo desde donde operan los clientes. Como todos los servidores de Fibre están en el hemisferio Norte, los bloques de un cliente minando en el hemisferio Sur necesariamente van a tener tiempos de propagación mayores a los de bloques descubiertos en el hemisferio Norte. Por otro lado, las redes de retransmisión agregan un cierto grado de centralización a la red de Bitcoin, ya que una misma persona es dueña de todos los servidores de la red Fibre. Esta situación no implica una centralización completa, porque si Fibre descarta algunos bloques de la red, estos bloques eventualmente llegan a destino mediante la red P2P tradicional. Sin embargo, entre otras cosas, Fibre podría retrasar (hasta cierto punto) la retransmisión de bloques, y nadie se podría dar cuenta, ya que la red P2P original es mucho más lenta. Entonces, el dueño de Fibre podría decidir empeorar las condiciones de red para un cliente en particular, retrasando solo la retransmisión de sus bloques. Esta disparidad de poder hace que la red no sea completamente descentralizada, en contraposición a la idea original de Bitcoin.

Como queremos estudiar posibles incrementos de *throughput* manteniendo completamente el aspecto descentralizado de Bitcoin, en los experimentos no vamos a incluir una red de retransmisión como la descripta.

1.3 Trabajo relacionado

Decker y Wattenhofer [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. En base a la información recopilada, analizan el comportamiento de la red durante 10,000 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,

concluyen que el tiempo de propagación de los bloques es la principal causa de *forks* en la red y hallan una correlación fuerte entre el tiempo de propagación de los bloques y el tamaño de los mismos.

Además proponen cambios para mejorar el tiempo de propagación de bloques: tener mayor conectividad entre nodos, propagar las notificaciones de bloque nuevo antes de recibir el bloque completo, 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. [DPSHJ14] también mide el tiempo de propagación de bloques, pero haciendo foco en la topología de la red y buscando obtener la distribución de direcciones IPs de los nodos Bitcoin por país. Para ello, su cliente envía 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 mayor a 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, ésta no es lineal.
- Reportan la métrica de qué porcentaje de bloques se propagó antes de un tiempo dado a un porcentaje de nodos.
- Encuentran 87.000 IPs distintas de nodos de Bitcoin, de las cuales solo 6.000 mantuvieron constantes a lo largo del experimento.

Los autores publicaron junto con el trabajo la distribución de IPs por país que luego utilizaremos para armar topologías virtuales para nuestros experimentos.

Miller et al [MJ15] 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 y las comparan con la cantidad de comunidades de grafos aleatorios. El resultado es que la media de estas medidas se encuentra a dos desvíos estándares de distancia de la media de las métricas en grafos aleatorios. Los autores proponen que esto se debe a que los nodos se empiezan a conectar a partir de *«boot nodes»* y de a poco se van expandiendo.

Con respecto a los nodos influyentes, el 2 % de los nodos es responsable de 75 % 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 éstos tienen gran poder de cómputo y serían nodos influyentes.

Croman et al. [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`, 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, que es una propuesta para mejorar el tiempo de propagación de bloques enviando menos información. La solución aprobada se denomina «Compact Blocks» [Cor], dejando rechazada a la propuesta de utilizar «XThin Blocks» [CRS+].

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 los trabajos de Kiayias et al. [KP15, GKL16] se analiza la seguridad de blockchains con intervalos muy cortos entre bloques, incluyendo el efecto del protocolo de consenso usado por Ethereum.

Marco Vanotti en [Van16] analiza los efectos de decrementar el tiempo entre bloques y aumentar el diámetro de la red respecto a la generación de *forks* en la red. Utiliza `Mininet` y `Maxinet` para emular las topologías de red descriptas en los trabajos de Donet et al. [DPSHJ14] y Miller et al. [MJ15].

Silvio Vileriño [Vn17] diseña una metodología para el estudio de sistemas basados en blockchain. Luego utiliza la misma para experimentar sobre los límites del funcionamiento de Ethereum cuando se reduce el tiempo entre bloques.

Dihn et al. [DWC+17] desarrolla una herramienta llamada *Blockbench* que permite integrarse a cualquier sistema de blockchain por medio de una API. El foco del trabajo es el uso de redes privadas de blockchain para el desarrollo de aplicaciones usualmente montadas sobre bases de datos estándar, como por ejemplo aplicaciones bancarias. El framework

presentado mide distintos aspectos de performance, escalabilidad y tolerancia a fallos utilizando como entrada datos recolectados de clientes reales y datos generados sintéticamente. Realizan experimentos sobre Ethereum, Parity y Hyperledger Fabric y concluyen que las redes blockchain actuales no son adecuadas para procesamiento de datos a gran escala.

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

Capítulo 2

Metodología

Este capítulo está dedicado a explicar las herramientas y métodos utilizados en la realización de los experimentos de este trabajo.

Primero, en la sección 2.1, vamos a detallar el proceso de minado simulado utilizado en los experimentos. El mismo nos permite emular el proceso de minado real sin gastar poder de cómputo.

La sección 2.2 está dedicada al cliente de Bitcoin usado en los experimentos. Tomando el cliente oficial como punto de inicio, modificamos algunos aspectos de su funcionamiento interno y le agregamos nuevas funcionalidades.

Luego vamos a presentar el modelo de red de Bitcoin utilizado en el presente trabajo. En la sección 2.3 fundamentamos las decisiones tomadas en dicho modelo, explicamos cómo construirlo y detallamos los datos usados en el proceso.

En la sección 2.4 presentamos y discutimos las magnitudes a medir en los experimentos. Por último, en la sección 2.5, mostramos los métodos y materiales usados para asegurar la fidelidad de los resultados obtenidos en los experimentos.

2.1 Minado simulado

Planteamos un modelo estadístico que simula el comportamiento de minado de bloques. Esto nos permitió emular el funcionamiento de la red de Bitcoin sin tener que invertir poder de cómputo real.

Como los hashes calculados durante el minado de bloques son criptográficamente seguros, podemos suponer que están distribuidos de manera uniforme. Entonces, cada vez que computamos un hash tenemos la misma probabilidad de que sea un valor que cumpla con la dificultad de la red. Por lo tanto, la acción de calcular un hash puede modelarse como un ensayo de *Bernoulli*.

Dado que minar consiste en la repetición de estos ensayos en intervalos cortos de tiempo, el acto de minado de un cliente puede pensarse como un proceso de *Poisson* caracterizado por el parámetro λ_i, que indica la cantidad de bloques que el nodo i genera por unidad de tiempo. Si la red tuviera n mineros independientes en la red, habría n procesos Poisson independientes.

Como indica la figura 2.1, siendo $\lambda_1, \ldots, \lambda_n$ los parámetros para cada uno de los mineros de nuestra red, en la visión global donde todos los mineros están ejecutando en forma paralela, el proceso de minado es un proceso de Poisson con parámetro $\lambda_g = \sum_{i=1}^{n} \lambda_i$.

En un proceso de Poisson de parámetro λ_g, la cantidad de tiempo que transcurre entre cada par de eventos sigue una distribución exponencial con media $\frac{1}{\lambda_g}$. Entonces, si queremos que el tiempo entre bloques promedio de la red sea de 600 segundos, tendremos que $\frac{1}{\lambda_g} = 600$, resultando en que $\lambda_g = \frac{1}{600}$. El modelo no tiene restricciones sobre la forma en que distribuimos el hashing power de la red entre los mineros que actúan.

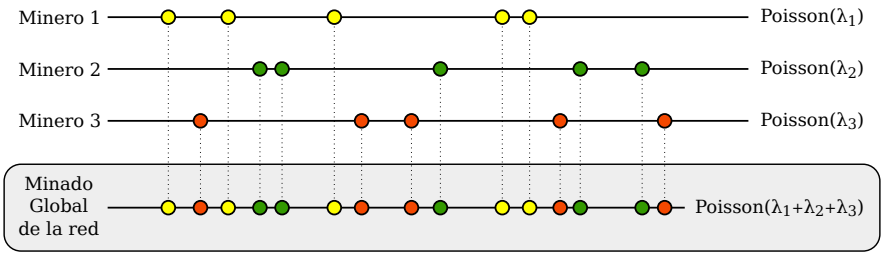


Figura 2.1: Suma de tres procesos de Poisson independientes. Cada línea simula el proceso de minado a lo largo del tiempo. El resultado es, también, un proceso de Poisson donde su parámetro característico es la suma del de los tres procesos de Poisson que lo componen.

2.2 Instrumentalización del cliente

Modificamos el cliente BTC Core v0.17 con el fin de poder realizar distintos experimentos. Elegimos Bitcoin Core por ser ser el programa más usado por los nodos de la red. Además, usamos la versión 0.17 por ser la última al momento de hacer el trabajo.

Modificamos el código del programa para poder especificar la dificultad inicial de la red cuando se inicializa el cliente. En la sección 2.2.1 explicamos esta nueva funcionalidad.

También realizamos diversas modificaciones a la parte del programa que procesa transacciones. En la sección 2.2.2 detallamos todos estos cambios, así como también las motivaciones para realizarlos.

Implementamos, dentro del cliente, un módulo de manejo de *logs* que registra todos los datos necesarios para reconstruir las métricas de los experimentos. Para realizarlo agregamos nuevo código al programa. En la sección 2.2.3 mostramos la funcionalidad de este nuevo módulo.

Por último, realizamos cambios de protocolo a la red *regtest* del cliente que es una red privada incorporada en el cliente oficial. Esta red tiene parámetros predeterminados distintos a la *mainnet* (la red de producción), y se usa principalmente para hacer pruebas y experimentos. Como en nuestros experimentos queremos emular a la red real, en la sección 2.2.4 listamos todos los cambios hechos para que la red *regtest* de nuestro cliente se comporte como la *mainnet*.

2.2.1 Selección de dificultad

Agregamos la posibilidad de elegir el orden de magnitud de la dificultad inicial de la red. Al inicializar el cliente, podemos especificar un valor x entre 0 y 255, luego la dificultad de la red empezará siendo $2^{256-x} - 1$. Bajo este esquema, los hashes de los primeros bloques minados van a satisfacer la dificultad si y solo si sus x bits más significativos son 0.

Para implementar este modelo, el identificador `dificultad` fue agregado a la lista de comandos válidos del programa. Luego, el usuario puede especificar una dificultad x agregando el argumento `-dificultad=x` a la inicialización del cliente. Cuando esta opción es usada, se genera un número de 256 bits que representa la dificultad deseada y se almacena entre los parámetros de la blockchain.

Por otro lado, en BTC Core existe la función `GetNextWorkRequired`, la cual determina cuál debe ser la dificultad del próximo bloque. Ésta es usada principalmente para validar que los bloques que llegan de los *peers* tengan un hash que cumpla con la dificultad de la red. El método también es utilizado para determinar la dificultad actual de la red al momento de minar nuevos bloques.

Modificamos `GetNextWorkRequired` agregando una cláusula al principio: si el próximo bloque tiene numeración 1, entonces la dificultad requerida es la establecida por el usuario al iniciar el programa. De esta manera inyectamos la dificultad inicial deseada, sin modificar la lógica de recalculado de dificultad. Tampoco fue necesario cambiar aspectos de la generación y validación de bloques, por lo que la blockchain construida conserva todas las características propias del sistema Bitcoin. Por último, la red real tiene una dificultad mínima determinada, que es utilizada cuando el ajuste de dificultad resulta en un valor menor o igual. En el cliente cambiamos este número a $2^{256} - 1$, esto permite asignar dificultades menores a la mínima posible en la red real.

Para implementar el proceso de minado simulado descripto en la sección 2.1, necesitamos poder generar bloques válidos en tiempo constante. Es por esto que utilizamos el método presentado en esta sección para asignar dificultad inicial 0 a la red. De esta manera, el

primer hash calculado para cualquier bloque va a cumplir con los requisitos de dificultad, y por lo tanto, minar un bloque válido va a solo ser una cuestión de agregar transacciones y llenar cada campo del header correctamente.

2.2.2 Aumento del throughput de transacciones

Cada nodo de la red cuenta con un conjunto de transacciónes sin confirmar, éstas son válidas, pero no forman parte de un bloque que está en la blockchain. La estructura que contiene estas transacciones válidas sin confirmar es llamada *mempool*. Cuando un nodo mina un nuevo bloque, incluye un subconjunto de transacciones de su *mempool*. Al momento de armar el bloque, agrega una a una las transacciones del *mempool* hasta que el bloque se llena o el *mempool* se vacía. Cabe destacar que en el *mempool* pueden convivir dos transacciones incompatibles entre sí. Si este fuera el caso, solo una de ellas va a ser incluida en el próximo bloque, incluso si hubiera espacio suficiente para ambas.

Como vimos en la sección 1.2.1, existe una cota superior teórica sobre el *throughput* de Bitcoin, es decir, la cantidad de transacciones por segundo que el sistema puede incorporar a la blockchain. Esta limitación está dada por parámetros del protocolo, específicamente, por el tiempo entre bloques promedio y el máximo tamaño de bloque posible. En principio, modificar estos parámetros debería resultar en que el sistema pueda funcionar con un *throughput* mayor a la cota actual.

Sin embargo, notamos que, incluso si cambiábamos los parámetros de la red, no podíamos lograr un *throughput* superior a 30 transacciones por segundo. Luego de realizar experimentos para buscar los motivos de este comportamiento, encontramos una limitación de software en el cliente oficial de Bitcoin. Al momento de procesar una nueva transacción, el cliente BTC Core utiliza un mecanismo de sincronización (i.e. *mutex*) para asegurarse exclusividad en su procesamiento. La variable del programa utilizada se llama `cs_main`. Pero, el código no solo utiliza usa este *mutex* al procesar una nueva transacción sino que también lo hace al minar y cuando valida los bloques que llegan de sus *peers*.

Esto trae las siguientes dos consecuencias: por un lado las transacciones se procesan de manera completamente serial, no es posible comenzar con una nueva, hasta no asentar la anterior. Además, durante los más de 30 ms que tarda en validarse y almacenarse cada transacción, el cliente no puede empezar el proceso de minado ni recibir bloques nuevos. Esto afecta de manera inherente el *throughput* del cliente, llegando a un máximo de 30 transacciones por segundo.

Esta limitación provoca que el cliente sin modificaciones no se pueda usar para las condiciones experimentales en las que se busque una mayor cantidad de transacciones por segundo. Por este motivo, nos vimos forzados a realizar algunas modificaciones al cliente que permiten elevar la cantidad de transacciones por segundo que puede procesar.

Al momento de validar una transacción, el cliente la compara contra todas las que ya están confirmadas en la blockchain, para certificar que es compatible. Este chequeo requiere tiempo lineal con respecto a la cantidad de transacciones en la blockchain. En nuestros experimentos nunca enviamos operaciones que puedan generar conflicto con otras anteriores, lo que nos permite deshabilitar esta validación y reducir el tiempo necesario para procesar transacciones.

Otra actividad que impactaba en el *throughput* de transacciones es la copia a disco del *mempool*. Cada vez que una transacción nueva ingresa al *mempool*, éste es volcado a disco en su totalidad. Modificamos este comportamiento para que el volcado se realice sólo si se llena la capacidad del *mempool* en memoria principal. La necesidad de resguardar el *mempool* en disco se basa en el diseño del sistema para asegurar la persistencia de los datos cuando se funciona en producción. Como realizamos los experimentos en un ambiente controlado, no nos tenemos que preocupar por imprevistos como que se apague la máquina donde está corriendo el cliente.

BTC Core tiene un sistema de manejo de *logs* de eventos en el cual registra todo lo que va haciendo. Si en algún momento ocurre una falla, es posible recurrir a este registro para *debuggear* el programa. Como este mecanismo agrega *overhead* a cada operación que el

cliente hace, esta funcionalidad fue deshabilitada.

El cliente de Bitcoin cuenta con una llamada RPC que devuelve todos los *outputs* sin gastar de sus direcciones. El resultado de esta función es una lista de archivos `jsons`, en la cual cada uno contiene 11 atributos. Esta estructura crece con la cantidad de *outputs* disponibles y puede llegar a ser muy grande. Por otro lado, en nuestros experimentos, de cada *output* disponible solo usamos cuatro de esos 11 atributos. Es por esto que agregamos otra llamada RPC que solamente prepara y devuelve para cada *output* disponible, solo los cuatro atributos que necesitamos. Esta nueva llamada RPC es un mecanismo mucho más rápido que el original y, por lo tanto, agrega una menor carga al proceso de generación de transacciones. Las propiedades que dejamos son solo aquellas necesarias para poder usar el *output* que está siendo detallado.

Por último, realizamos modificaciones a la cache del cliente para reducir el volumen de datos que éste escribe a disco. Cambiamos el código del programa para que el tamaño por defecto del *mempool* aumente de 300 MB a 800 MB. Esta nueva capacidad sigue siendo baja para la cantidad de memoria principal del hardware utilizado, por lo tanto podemos utilizar esta nueva configuración sin incurrir en problemas. El incremento del tamaño del *mempool* permite almacenar más transacciones antes de que sea necesario bajarlas a disco. Además del *mempool*, el cliente cuenta con una cache de propósito general que se usa para almacenar el estado de distintas estructuras del programa. Sin embargo, esta estructura cuenta originalmente con límites de tamaño específicos para cada estructura, por ejemplo, no podían usarse más de 2 MB para almacenar bloques. Aumentamos los límites de todas las estructuras para minimizar el uso de disco del cliente.

Todos estos cambios lograron que el cliente alcance un *throughput* de, al menos, 120 transacciones por segundo. Esta cantidad resulta más que suficiente ya que, en todos los experimentos que realizamos y debido a la configuración del protocolo de red utilizada, la cota superior teórica de *throughput* resulta menor a 55 transacciones por segundo. Por lo tanto, con las modificaciones realizadas, podemos usar el cliente en nuestros experimentos sin condicionarlos por una limitación de software.

Por último, cabe aclarar que, si bien la solución propuesta incluye modificar el funcionamiento del cliente, es posible lograr incrementar el límite de *throughput* al nivel requerido por nuestros experimentos, modificando sustancialmente la arquitectura del cliente. Una solución posible es dividir el *mempool* en varias estructuras equivalentes, y crear un *pipeline* para la validación de transacciones. Utilizando un diseño de esas características, el *throughput* del cliente tiene el potencial de aumentar a alrededor de 150 transacciones por segundo, que resulta superior a la cota actual del cliente utilizado en el trabajo.

2.2.3 Manejo de logs

Como mencionamos en la sección anterior, el cliente posee un sistema de manejo de *logs* con fines de *debugging*. Éste recolecta mucha información irrelevante para nuestros experimentos y no registra datos que sí usamos, por lo tanto dicho sistema fue deshabilitado.

Como hay datos de la red que sí necesitamos, agregamos al cliente un nuevo sistema de manejo de *logs* para poder medir el comportamiento de la red mientras se realizan los experimentos. Éste consiste en dejar registrada información en un archivo cada vez que ocurre un evento relevante. Cuando el cliente se entera mediante un *peer* que apareció un nuevo bloque, deja registrado su hash. Luego, al momento de asentar ese bloque en la blockchain local, en una nueva entrada se vuelve a escribir el hash, acompañado del hash del padre. Por último, si el programa mina un nuevo bloque, se agrega al *log* la cantidad de transacciones que contiene, su hash y el hash del padre. Además de la información mencionada, cada entrada del *log* tiene un *timestamp* que indica en qué momento ocurrió el suceso.

En resumen, las tres entradas posibles en el *log* son las siguientes:

- Al minar un nuevo bloque:
  Hash del bloque descubierto | Hash del ancestro | Cantidad de transacciones incluidas en el bloque | Timestamp del momento de la creación
- Al asentar un nuevo bloque proveniente de la red:

Hash del bloque asentado | Hash del ancestro | Timestamp del momento de la aceptación

· Al descubrir un hash de un nuevo bloque:
Hash descubierto | Timestamp del momento del descubrimiento

Utilizando los hashes de bloques y los ancestros podemos reconstruir la blockchain. Luego, teniendo ya el árbol completo, podemos calcular la información relacionada a *forks* detallada en la sección 2.4.1

Por otro lado, usando los *timestamps* de creación, descubrimiento y aceptación, podemos saber los tiempos de propagación de cada bloque. Los mismos son detallados en la sección 2.4.2

Además, usando los *timestamps* de creación podemos saber cuál fue el tiempo entre bloques promedio. Esta magnitud nos sirve como control (se detalla en la sección 2.5.1).

Por último, la cantidad de transacciones por bloque, sumado al tiempo entre bloques, nos permite saber cuál fue el *throughput* de la red. Este dato también es mencionado en la sección 2.5.1, ya que forma parte de la validación de los experimentos.

2.2.4 Acondicionamiento de la red Regtest

Cuando se inicia el cliente, se le puede indicar que use una de las tres redes disponibles: *mainnet*, *testnest* o *regtest*. La primera es la red oficial, en la cual cada bitcoin cotiza en el mercado. La segunda es una red pública global paralela a la *mainnet*, los bitcoins de esta red no tienen un valor de conversión a monedas reales. La última es una red privada local, la cual se usa para testear y *debuggear* la mayoría de los aspectos del cliente. En nuestros experimentos vamos a usar la red *regtest* que, al ser privada y local, nos da un ambiente controlado.

Como la red *regtest* está pensada para hacer tests, viene configurada con parámetros distintos a *mainnet*. Por ejemplo, la frecuencia con la que la recompensa de minado es dividida

por dos es diferente. Como queremos emular la red real, todos los parámetros de configuración fueron elegidos con los mismos valores que en la red *mainnet*.

Además de la discrepancia en los parámetros de la red, existían diferencias en el funcionamiento del programa cuando se ejecuta usando *regtest* en lugar de *mainnet*. En *regtest*, si no aparecieron bloques en los últimos 20 minutos, la dificultad bajaba a la mínima posible. Esta lógica fue deshabilitada, ya que no sucede lo mismo en *mainnet*. También, en *regtest*, cada vez que se validaba una transacción, no solo se realizaban los chequeos relacionados a la nueva operación, sino que también se verificaban todas las que ya estaban asentadas en la blockchain. El mismo mecanismo cuadrático se efectuaba en la validación de bloques, verificando todos los anteriores. Estos chequeos fueron deshabilitados ya que tampoco se efectúan cuando el cliente utiliza *mainnet*.

Habiendo realizado los cambios mencionados, podemos usar *regtest* para realizar experimentos, y obtener resultados válidos para la red real.

2.3 Modelo de Red

En esta sección vamos a presentar un modelo a escala que representa a la red de Bitcoin real. El tamaño de la escala es variable, es decir, dado un número n, vamos a mostrar cómo construir una red con n clientes que sigue el modelo propuesto.

En la sección 2.3.1 vamos a analizar tres características de la red real: Tiempos de propagación, disposición geográfica de los nodos y distribución del hashing power de la red. Veremos que la recreación de estas características es crucial para una buena representación de la red real.

Luego, en la sección 2.3.2, pasamos a detallar todos los datos concretos que van a ser utilizados en la construcción del modelo. Por otro lado, en la sección 2.3.3, mostramos cómo van a ser usados estos datos.

En la sección 2.3.4, describimos un modelo de red IP que asigna regiones del mundo a cada cliente de la red. Además, bajo dicho modelo, las latencias entre nodos son definidas basán-

dose únicamente en los lugares asignados a cada uno de ellos. Este modelo fue introducido en [Vn17].

En la sección 2.3.5, mostramos la metodología para decidir la región del mundo de cada cliente, así como también su *hashing power*.

Teniendo definidos todos los clientes de la red, en la sección 2.3.6, vamos a ver el procedimiento para establecer conexiones entre ellos.

2.3.1 Representando a la red real

La idea del modelo es representar a la red real, en una escala reducida. Ésta debe poseer características similares, revelando un comportamiento equivalente al real. Sin embargo, no todas las características de la red son igual de relevantes. En esta sección mostramos los rasgos de la red real que vamos a recrear: los tiempos de propagación, la disposición geográfica de los nodos y la distribución del *hashing power* a través de los mineros.

2.3.1.1 Sobre los tiempos de propagación

Como Bitcoin es un sistema distribuido, los eventos producidos por cada nodo no se propagan de manera instantánea. Al tiempo que transcurre entre que un nodo genera un nuevo evento (por ejemplo mina un bloque) y dicho evento llega al resto de los clientes de la red, se lo conoce como *tiempo de propagación*.

Desde que un nodo x mina un bloque, hasta que dicho bloque le llega a otro nodo y (es decir, durante el tiempo de propagación del bloque), todo el esfuerzo dedicado al minado por parte de y es desperdiciado, ya que el bloque que se está buscando ya fue descubierto por x. Además, si durante el tiempo de propagación, y descubriera el bloque ya minado por x, se crearía un *fork* en la blockchain. Por lo tanto, es claro que en el sistema Bitcoin, los tiempos de propagación afectan directamente a su funcionamiento.

Los tiempos de propagación están determinados principalmente por dos factores: la topología de la red de Bitcoin y la latencia entre clientes en la red IP subyacente. El primero

determina el camino entre cada par de nodos de la red, y el segundo indica cuánto lleva recorrer las aristas de dicho camino.

En las secciones 2.3.5 y 2.3.6 mostramos cómo se genera la topología de la red de Bitcoin del modelo, la cual recrea la topología de la red real.

En la *mainnet*, la latencia entre clientes está determinada principalmente por sus ubicaciones geográficas. Es por esto que en el modelo vamos a hacer hincapié en la distribución geográfica de los nodos en el mundo.

2.3.1.2 Diferenciando nodos por su ubicación

En el modelo propuesto, le asignamos una zona geográfica a cada cliente, representando que ese nodo opera desde ese lugar del mundo. Luego aseguramos que dados dos clientes, la latencia entre ellos depende únicamente de las ubicaciones que le fueron designadas a cada uno.

De esta manera, utilizando valores de *delay* intra-regiones e inter-regiones similares a los de la red IP global, recreamos las condiciones de latencia que hay entre clientes reales que están corriendo en distintas regiones del mundo.

2.3.1.3 Distribución del hashing power

Como vimos en la sección 1.1.2, en Bitcoin se utiliza un sistema de consenso para decidir qué bloques pertenecen a la blockchain y cuáles no. Dicho consenso se logra a través del proceso de minado, en el cual los nodos usan poder de cómputo para decidir cuál va a ser el siguiente bloque de la blockchain.

El resultado de esta votación está influenciado por la distribución del *hash rate* de la red. Esto es porque la probabilidad de que un nodo descubra un nuevo bloque antes de un determinado tiempo depende exclusivamente del *hash rate* de dicho nodo (suponiendo que mina de manera continua sin parar). Por lo tanto, la probabilidad de que un determinado nodo descubra un nuevo bloque antes que los demás, depende exclusivamente de los *hash*

rates de cada uno de los mineros de la red. En otras palabras, la convergencia de la blockchain está dada por la distribución del *hash rate* de la red.

Por último, vamos a poner el foco en un rasgo de la red de Bitcoin real: el *hash rate* está distribuido de manera desigual. Menos del 1 % de los nodos de la red controla más del 85 % del *hash rate* de toda la red. Estos nodos con gran poder de cómputo son llamados *pooles de minería*, ya que varios de ellos consisten en la unión de muchos clientes mineros que no están conectados de manera directa a la red. Dichos clientes distribuyen el espacio de búsqueda de la *proof of work*. De esa manera, como además introducen bloques a la red desde un mismo conjunto reducido de IPs, para los demás nodos de la red, este conjunto de clientes es visto como un único nodo que tiene una gran parte del *hashing power* de la red.

El porcentaje de *hashing power* de la red que es controlado por *pools* de minería varía en el tiempo. En algunos casos llegó a ser más del 95 % del total.

En el modelo vamos a recrear una distribución de *hashing power* desigual. También vamos a definir nodos que representan *pools* de minería.

2.3.2 Fuentes de datos

Para poder construir el modelo propuesto necesitamos saber principalmente tres cosas: (i) la distribución geográfica de los nodos de la red real de Bitcoin, (ii) las latencias entre los lugares donde se encuentran los clientes y (iii) los detalles de todos los *pools* de minería de Bitcoin.

Con respecto a la distribución de los clientes a través del mundo, utilizamos datos del sitio `bitnodes.earn.com` para saber, dado un país, cuántos nodos de Bitcoin operan desde ahí. Los países grandes fueron subdivididos en regiones, ya que vamos a usar la categorización geográfica para establecer latencias entre nodos, y por ejemplo, un cliente en el Oeste de EEUU va a tener latencias inter-país distintas a las de un cliente operando desde el Este de EEUU. Utilizamos también `bitnodes.earn.com` para distribuir los nodos a través de

distintas regiones del mismo país. Los países chicos fueron tenidos en cuenta como una sola región.

A fin de introducir las características de conectividad dentro del modelo, utilizamos el sitio wondernetwork.com/pings como referencia de las latencias entre distintas ciudades del mundo. Nos fijamos las latencias entre ciudades de la misma región y las tomamos como latencias intra-regiones. Es decir, las latencias asignadas a ciudades serán las latencias de las regiones en donde se ubican.

Por último, utilizamos los datos mostrados en https://btc.com/stats/pool?pool_mode=month para saber qué *pools* existen en la red y qué porcentaje de *hashrate* controla cada uno. La página estima el *hashing power* de cada *pool* fijándose qué porcentaje de los bloques fue minado por cada *pool* durante los últimos 30 días. Luego, nos referimos a las páginas oficiales de cada *pool* para conocer qué nodos públicos poseen en la red de Bitcoin, y en qué lugar del mundo se ubica cada uno. Todos estos datos fueron obtenidos el 1/12/2018 y con ellos se construye el modelo propuesto.

2.3.3 Cómo se utilizan los datos

Con respecto a la distribución geográfica de los nodos de la red, vamos a calcular el porcentaje de clientes en cada lugar del mundo, y recrear una proporción similar en el modelo.

Luego, utilizando la topología IP detallada en la sección 2.3.4, vamos a hacer que la latencia entre dos nodos dependa únicamente de sus ubicaciones en el mundo. Usando este esquema, sumado a que utilizamos los datos de latencia reales mencionados anteriormente, esperamos que los tiempos de propagación en el modelo resulten similares a las de la red real.

Por otro lado, con respecto al *hashing power* de la red, vamos a hacer una distinción especial entre *pools* de minería y los demás nodos. Es decir, vamos a procesar de una manera a aquellos nodos que poseen un porcentaje despreciable del *hashing power* total de la red, y de otra diferente a los que no.

Usando los datos de *pools* mencionados anteriormente, vamos a armar una lista que contiene

todos los *pools* de minería identificables de la red. Para cada uno, enumeramos todos los nodos que posee en la red y, para cada uno de estos nodos, se detalla su ubicación geográfica y su porcentaje de *hashing power* de la red. La estrategia utilizada consiste en agregar al esquema de red un cliente por cada nodo de *pool* que figura en el registro, respetando su *hashing power* y zona del mundo.

Cabe notar que, según la información que tenemos, un poco más del 14 % del *hash rate* de la red no es controlado por *pools* conocidos públicamente. Además, no existe un cliente que, individualmente, aporte una parte significativa de dicho porcentaje. En otras palabras, en la red real, efectivamente un grupo de clientes con *hash rate* despreciable, en conjunto, aportan el *hash rate* que consideramos como de nodos que no están asociados a *pools*. Sin embargo, podría existir un *pool* privado que controle muchos clientes en la red y, de esta manera, aportar una gran parte del *hash rate* que nosotros consideramos que no es de *pools*. Pero, para los demás nodos que están en la red, un *pool* privado con estas características sería percibido de la misma manera que un grupo de nodos minando independientemente, ya que no existe un único cliente que aporte un porcentaje no despreciable de los bloques de la blockchain. Por lo tanto, si efectivamente existiera un *pool* privado en la red, en nuestro modelo éste sería correctamente representado por un subconjunto de los nodos que no están en *pools*.

2.3.4 Modelo de red IP

Vamos a describir la topología de la red utilizando un grafo G, el cual pasamos a especificar de manera constructiva.

Sea L la lista de zonas geográficas a incluir en el modelo. En nuestros experimentos L es una serie de 37 ciudades. G comienza siendo un grafo completo $K_{|L|}$, en el cual cada nodo representa un elemento de L, y cada arista (v, w) contiene la latencia medida en Internet entre los lugares representados por v y w. Dado un host h_i que deseamos incluir en la red, llamamos r_i al nodo que representa la ubicación asignada a h_i. Luego, para cada h_i, agregamos un nodo u_i al grafo, el cual solo es adyacente a r_i. Además, cada (u_i, r_i) tiene

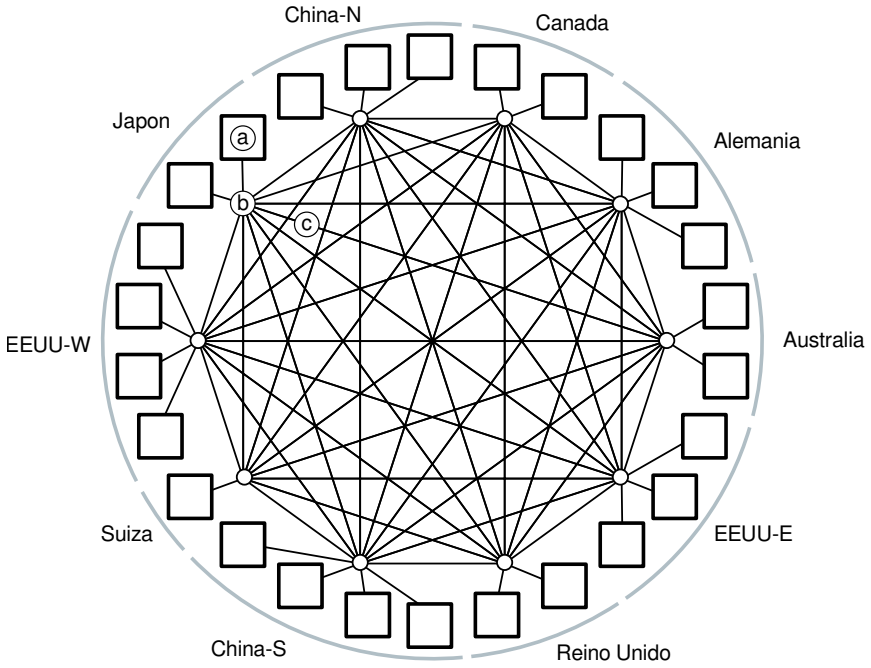


Figura 2.2: Modelo de red IP.

como valor la mitad de la latencia intra-región de la zona representada por r_i. De esta manera, logramos que 2 hosts en la misma ubicación geográfica tengan una latencia similar a la experimentada en la vida real. Por último, a cada arista (k_i, k_j) perteneciente a la clique máxima inducida por G, le restamos una vez el valor que tienen las aristas incidentes a k_i que no pertenecen a la clique. Esto último es necesario porque el número asignado originalmente incluía la mitad de la latencia intra-región de los lugares representados por k_i y k_j. En resumen, la topología puede verse como una clique con rayos. Los nodos de grado 1 son los hosts que van a correr el cliente de Bitcoin, mientras que los vértices de la clique son los puntos de acceso de cada región.

En la figura 2.2 se ve un ejemplo de del modelo. El objeto marcado con una "a" representa un host de la red IP que corre un cliente de Bitcoin. Uno de los hosts pertenecientes a la clique, y que por lo tanto no corre clientes de Bitcoin, está identificado con la letra "b". Por último, la "c" marca uno de los enlaces entre regiones.

2.3.5 Agregando clientes de Bitcoin al modelo

Ya teniendo el modelo de topología IP definido, lo que resta decidir es qué regiones del mundo vamos a incluir y cuántos clientes pertenecen a cada una. Estos dos valores se van a desprender de los datos mencionados y de la cantidad de nodos deseada. Suponiendo que queremos crear una red de n nodos. En primer lugar, vamos a agregar un cliente c_i al modelo por cada pool de minería p_i que tenemos registrado. La ubicación geográfica de c_i y su porcentaje de *hashing power* con respecto a la red van a ser iguales a los de p_i en *mainnet*. Una restricción del modelo es que la red a crear contenga al menos a todos los *pools* registrados. Basta con crear una red de al menos 50 nodos para contenerlos a todos.

Habiendo ya agregado todos los clientes que representan *pools* de minería, pasamos a procesar los demás nodos. Si tomamos t como la cantidad de *pools* que incorporamos al modelo, notamos que faltan agregar $m = n - t$ clientes. Además, llamando h a la suma de porciones de *hashing power* de cada p_i, vemos que queda distribuir $g = 1 - h$ del total de la red. Como todos los nodos que procesemos de ahora en más tienen un *hashing power* muy bajo en la red real, no tenemos datos precisos de su *hashing power*. Es por esto que decidimos establecer $l = (g/m) \cdot 100$ como porcentaje de *hashing power* a todos los clientes que agreguemos al modelo a partir de ese momento. De esta forma, distribuimos *hashing power* restante de manera uniforme entre los nodos que quedan por agregar, que son aquellos que no pertenecen a algún *pool* de minería.

Este esquema se comporta bien cuando m es grande, ya que eso implica que l termine siendo chico, como queremos. Con respecto a valores aceptables para n, sabemos que t está acotado por 50, si asignamos a n valores mayores a 150, el modelo funciona como esperamos.

Ahora vamos a enfocarnos en la distribución geográfica de la red, empezando por calcular para cada región r_i el porcentaje de clientes de la red real que posee; a este número lo llamamos d_i. Para obtener un valor aproximado inicial de cuántos nodos debería haber en r_i, introducimos la variable $x_i = n \cdot d_i$. El siguiente paso consiste en armar un vector A

que contiene todos los x_i y ordenarlo de manera ascendente. Llamamos q_j a la región del valor almacenado en la posición j de A. Una vez que tenemos los datos como queremos, vamos a iterar el arreglo de menor a mayor. Para cada A_j, en principio queremos hacer que q_j tenga $\lfloor A_j \rfloor$ clientes. Para lograr esto tenemos que considerar a los nodos que procesamos al principio, los cuales representan *pools* de la red real. Por lo tanto, llamando y_j a la cantidad de clientes que ya incorporamos a q_j, agregamos al modelo $z = \lfloor A_j \rfloor - y_j$ nodos localizados en q_j y actualizamos A_j asignándole z.

Cabe notar que z puede resultar negativo, en ese caso no vamos a incorporar clientes nuevos al modelo, pero A_j sí queda menor a 0. Este último escenario ocurre cuando hay un *pool* ubicado en un lugar con muy pocos nodos en la red. Si esto pasa varias veces, también puede ocurrir que eventualmente, luego de calcular un z, si agregásemos z nodos a la red, ésta quedaría con más de n clientes. Si eso sucede, en vez de agregar z nodos, insertamos al modelo la cantidad máxima posible para llegar al límite n. Como en la práctica hay menos de cinco *pools* localizados en lugares con pocos nodos en la red, y como usamos $\lfloor A_j \rfloor$, la situación de tener que agregar menos nodos de lo deseado porque llegamos al límite es extremadamente rara.

Luego de haber iterado todo el arreglo, como utilizamos la parte entera de cada valor, probablemente todavía no hayamos llegado a agregar n clientes. La idea ahora es usar esos lugares disponibles para incorporar regiones que todavía no utilizamos y así enriquecer el modelo. Sea k el índice tal que, antes de modificar el vector en la iteración anterior, $\lfloor A_k \rfloor = 0$ y $\lfloor A_{k+1} \rfloor > 0$. Es decir, identificamos la región q_k que en base a su porcentaje de clientes en la red real, no aportó nodos al modelo, y además de todos los lugares en esta situación, fue el que más cerca estuvo de hacerlo. Es fácil encontrar k mientras se realiza la primera iteración sobre A, por lo que su utilización no incrementa la complejidad del armado del modelo. El siguiente paso es recorrer A de manera decreciente empezando por k, hasta iterar sobre una posición cuya región geográfica posee menos de del 0,2 % de nodos de la red real. Este límite inferior está puesto para no agregar clientes a un lugar del mundo que tenga un porcentaje completamente despreciable de la distribución de nodos

de *mainnet*. Para cada posición i visitada de esta forma, si $A_i > 0$, agregamos un cliente ubicado en q_i y decrementamos A_i en 1. La condición de que A_i sea positivo filtra las regiones que aportaron clientes como *pools* de minería, a pesar de tener poco porcentaje de los nodos de la red. Luego de agregar algún nodo nuevo, si hay exactamente n clientes en el modelo, entonces se corta la iteración en ese momento y no incorporamos más clientes.

Si terminamos la última iteración y aún no se llegan a completar las n posiciones, vamos a ordenar A de manera decreciente. De esta forma logramos que al principio del arreglo se ubiquen los porcentajes de regiones que estuvieron más cerca de aportar un nodo más al modelo. Por lo tanto, ahora recorremos A desde la primera posición a la última hasta que el modelo cuente con n nodos, agregando un cliente a cada región de los porcentajes visitados de esta manera. En la práctica, esta última iteración por lo general no es necesaria, y si sucede, muy pocos nodos son agregados hasta llegar al límite n.

2.3.6 Estableciendo conexiones entre clientes

Ya teniendo establecidos todos los nodos del modelo, cada uno con su *hashing power* y zona geográfica, pasamos a determinar cómo están conectados los clientes de Bitcoin.

En el caso del cliente original, cuando un nuevo nodo corriendo BTC Core se conecta a la *mainnet*, se registra en algunos servidores de nombre (i.e. DNS), los cuales le responden enviándole una lista de direcciones de otros nodos en funcionamiento. El cliente, entonces, usa esa información para establecer conexiones con *peers* de la red. Vamos a basarnos en este mecanismo para desarrollar un algoritmo aleatorio, el cual usamos para establecer las conexiones entre clientes en el modelo.

Para empezar, creamos un vector B que sea una permutación aleatoria de todos los identificadores de nodos actualmente en el modelo. Adicionalmente, definimos una lista vacía C. Luego, iteramos B y para cada b_i, agregamos X copias de b_i a C, elegimos de manera aleatoria Y elementos c_1, c_2 de C distintos a b_i, los eliminamos de la lista, y hacemos que b_i esté conectado con c_1 y c_2 en el modelo. Esto resulta en que cada nodo tenga entre 1 y

$Y + X$ peers, ya que pueden haberse obtenido entre 1 e Y nodos distintos al momento de elegir Y de manera aleatoria y, luego, el cliente puede ser seleccionado por hasta X nodos futuros.

Naturalmente, dependiendo de los valores que se le asigne a n, X e Y, va a variar la cantidad de aristas que hay en el grafo. Utilizamos $n = 240$, $X = 6$ e $Y = 2$ para nuestros experimentos. En los resultados de los mismos observamos que los percentiles de los tiempos de propagación de bloques eran del mismo orden de magnitud que los reportados [DW13]. Además, también fue similar la cantidad de *forks* observada cuando se usó 600 s como *target time*.

Por último, como vimos en la sección 1.2.5, es posible utilizar redes de retransmisión centralizadas para disminuir los tiempos de propagación de la red. Nosotros decidimos no incluir una red de retransmisión para mantener el aspecto descentralizado de Bitcoin. Sin embargo, como las redes de retransmisión operan en paralelo a la red P2P tradicional, en futuros trabajos es posible usar el modelo presentado con el agregado de una red de retransmisión.

2.4 Medidas utilizadas

Como vimos en la sección 1.2.3, reducir el tiempo entre bloques trae diversas consecuencias negativas. Dichas consecuencias no son efectos directos que puedan ser calculados mediante expresiones matemáticas, como sí lo es el *throughput* máximo del sistema. Por lo tanto, las métricas utilizadas en el presente trabajo apuntan a medir la magnitud de los efectos adversos que resultan de reducir el tiempo entre bloques en el sistema de Bitcoin.

Primero presentamos métricas simples relacionadas a dos fenómenos observables: La cantidad de *forks* en la blockchain y los tiempos de propagación de la red. Luego, en base a estos dos factores definimos el concepto de *wasted hashing power* de un nodo, es decir, el tiempo en que un determinado nodo está gastando energía en calcular hashes sin contribuir a la convergencia de la red. Como minar bloques en la red real es un proceso que consume

energía, desperdiciar *hashing power* se traduce en elevar el gasto de dinero, sin aumentar la posibilidad de incrementar las ganancias. La última métrica presentada apunta a calcular el *wasted hashing power* de la red, que es lo que termina impactando a los inversores de manera directa.

2.4.1 Forks

Mostramos la cantidad de *forks* que hubo en la blockchain. Además para cada uno identificamos su altura, es decir, su distancia a la *mainchain*.

2.4.2 Tiempos de propagación

Cada vez que aparece un bloque nuevo en la blockchain, se anuncia mediante la distribución del hash de su *header*. Luego, cuando cada minero tiene toda la información correspondiente al bloque, lo acepta, es decir, lo asienta en su copia local de la blockchain. En los experimentos, para cada bloque de la blockchain, calculamos el tiempo de propagación de los dos eventos. Es decir, calculamos cuánto tardó en llegar el hash de su *header* a cada minero, y cuánto tardó cada minero en aceptar el bloque. Teniendo ya todos los tiempos de propagación para cada bloque, calculamos los percentiles [10, 100] de cada evento. Por último, para cada percentil *n-ésimo*, calculamos el promedio de todos los percentiles *n-ésimo* de los bloques.

2.4.3 Wasted Hashing Power

2.4.3.1 Motivación

Si bien las medidas anteriores dan una noción de la calidad de la red, lo hacen desde un punto de vista puramente técnico. Si aumenta la cantidad de *forks* o los tiempos de propagación, se degrada la convergencia de la red, es decir, aumenta el esfuerzo requerido para minar una determinada cantidad de bloques. Sin embargo, no es trivial cuantificar el impacto económico que producen las variaciones en los aspectos técnicos de la red. A partir

de las métricas simples presentadas anteriormente, presentamos *wasted hashing power*, que captura el porcentaje de *hashing power* que no fue utilizado para la convergencia de la red. Utilizando esta medida, es posible determinar qué proporción de una inversión económica es realmente usada.

2.4.3.2 Definición

En la sección 1.1.1.2 detallamos los atributos de los bloques en Bitcoin. Algunos de estos atributos, como la dificultad de la red, necesitan ser congruentes con la *mainchain*. Es decir, si por ejemplo un minero introduce a la red un nuevo bloque, que tiene como dificultad un valor distinto al determinado por la red, los demás nodos van a rechazar dicho bloque.

Los mineros utilizan el último bloque conocido para determinar cuál es la dificultad de la red. Si el cliente efectivamente tiene el extremo (último bloque) de la *mainchain*, también conoce la dificultad de la red actual, y equivocarse al establecerla en el bloque sería un error evitable. Si, en cambio, el cliente no tiene el extremo de la *mainchain* correcto, el escenario es más complejo. Aunque un cliente actúe de manera correcta y escriba como ancestro al extremo de su copia local de la *mainchain*, como Bitcoin es un sistema distribuido, puede suceder que el cliente desconozca que ya haya aparecido un bloque en la red con ese mismo ancestro. En este caso, mientras al cliente no le hayan llegado todos los datos necesarios para poder asentar el bloque en su blockchain, lo mejor que puede hacer es seguir calculando hashes con el extremo de la blockchain que conoce. En esa situación, el campo ancestro se establece de manera errónea, pero el cliente no puede evitar equivocarse. Lo que queremos remarcar es que puede haber errores inevitables en el proceso de minado producidos por la naturaleza distribuida del sistema Bitcoin.

Como explicamos en la sección 1.1.2, en el proceso de minado cada nodo aporta su *hashing power* para que, en conjunto, minen un nuevo bloque cada diez minutos en promedio. Sin embargo, notemos que la probabilidad de que un hash cumpla con la dificultad de la red es independiente de la validez de los demás parámetros. Por lo tanto, si un minero logra calcular un hash que cumpla con la dificultad requerida, y luego el bloque termina siendo

inválido por otros motivos, la red va a tener que esperar en promedio diez minutos para generar un nuevo bloque. En este caso, el cliente que generó el bloque inválido desperdició su *hashing power* mientras minaba para dicho bloque, ya que no lo estaba usando para contribuir al objetivo grupal de generar un bloque válido cada diez minutos. Dicho de otra manera, vamos a considerar que un cliente está desperdiciando su *hashing power* cuando genera hashes de un bloque que tiene algún campo inválido.

En el presente trabajo suponemos que todos los nodos actúan de manera óptima, es decir, no vamos a tener en cuenta errores evitables. Todos los campos que tienen que ser consistentes con la red dependen del último bloque de la *mainchain*. Por lo tanto, si generamos un hash válido para un bloque nuevo que tiene el campo ancestro correcto, entonces el bloque va a ser válido. Lo que sucede a veces es que la correctitud del campo ancestro puede ser determinada luego de que transcurre bastante tiempo desde la generación del bloque. Por ejemplo, cuando ocurre un *fork* en la red, recién luego de que se determine cuál de los bloques con misma altura termine formando parte de la *mainchain*, cada nodo va a saber si estaba ayudando a minar sobre la *mainchain* o no.

En la práctica, cada cliente desperdicia su *hashing power* en dos situaciones: (i) durante el tiempo que transcurre entre que se genera un bloque nuevo y el cliente lo acepta; y (ii) cuando se produce un *fork* en la blockchain y el cliente mina sobre un bloque que no termina en la *mainchain*. La sumatoria de todos esos tiempos para un cliente dado es el tiempo en el que dicho cliente desperdició su *hashing power*. Llamemos d_i al tiempo que el nodo i pasó desperdiciando su hashing power, y $h_i \in [0, 1]$ a su proporción de *hashing power* con respecto al total de la red. Entonces, definimos al tiempo total desperdiciado de un experimento x como $D_x = \sum_{i \in nodos} d_i \cdot h_i$. Luego, dado que el tiempo total en el que el experimento x transcurrió fue T_x, el porcentaje de *wasted hashing power* de x se puede calcular como $(D_x/T_x) \cdot 100$.

La figura 2.3 ilustra una fracción de la línea de tiempo relacionada al proceso minado de la blockchain. La misma incluye los eventos relacionados a cuatro clientes de la red, y remarca los momentos en los que dichos clientes desperdiciaron su *hashing power*.

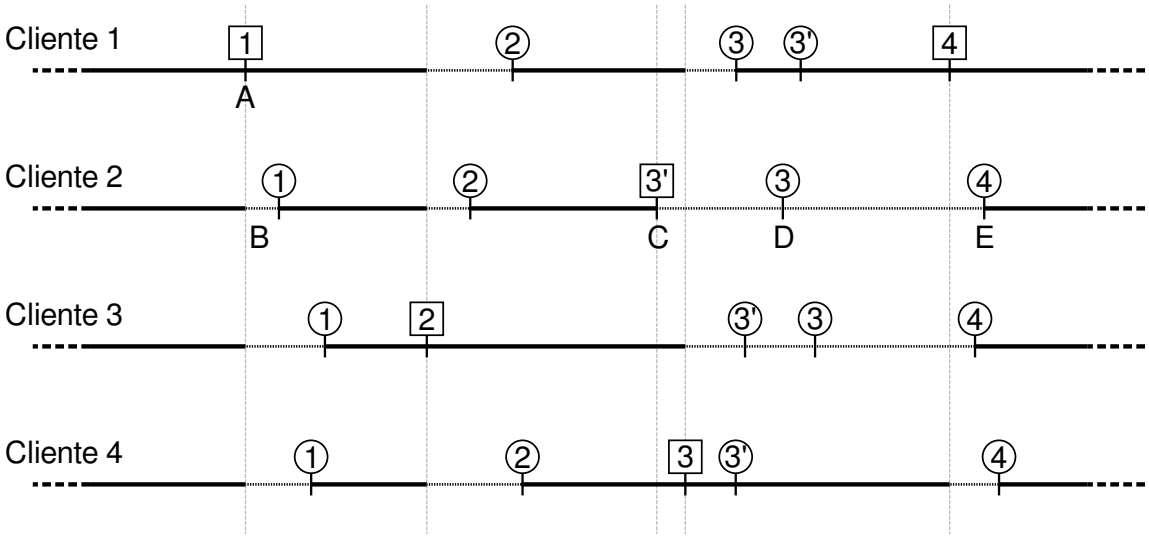


Figura 2.3: Diagrama de dependencias de variables en el sistema de Bitcoin

En la sección de resultados vamos a utilizar como métrica el porcentaje de *wasted hashing power* de un experimento. De esta manera vamos a medir el impacto monetario que resulta de aumentar el *throughput* mediante la reducción del tiempo entre bloques.

2.4.4 Tiempos de confirmación

Como vimos en la sección 1, cuando Bitcoin tuvo una suba en la cantidad de transacciones por segundo enviadas por sus usuarios, éstas llegaron a tardar en promedio 33,15 horas hasta ser incluidas en un bloque. A su vez, esta situación de congestión hizo que incrementen las comisiones de las transacciones. En nuestros experimentos vamos a llamar *Tiempo de confirmación* al tiempo que transcurre desde que una transacción es introducida al sistema hasta que forma parte de un bloque. Utilizando esta medida, podemos analizar cómo afectan los distintos cambios a los parámetros del protocolo desde el punto de vista del usuario del sistema.

2.5 Control Experimental

2.5.1 Validación del experimento

En cada experimento le establecemos valores a dos variables: el tiempo entre bloques objetivo y la cantidad de transacciones que son introducidas al sistema por segundo. Con el fin de controlar que el sistema se haya comportado acorde a los valores deseados, controlamos que ciertas mediciones obtenidas estén en un rango esperable.

Por un lado, calculamos el tiempo entre bloques promedio observado, teniendo en cuenta también los bloques que no forman parte de la *mainchain*. De esta manera, controlamos que el proceso de minado simulado se comporte como esperamos.

Por otro lado, utilizamos la cantidad de transacciones en cada bloque para determinar cuántas, en promedio, tuvieron los bloques que quedaron en la *mainchain* del experimento. Este valor, junto al promedio de tiempo entre bloques de la *mainchain*, nos indica la cantidad de transacciones por segundo promedio que hubo durante el experimento. Además, el componente que genera las transacciones está programado para que, si en algún momento no puede introducir transacciones al sistema, deje registrado el período de tiempo de inactividad. Luego, para cada *engine* calculamos la suma de todos los períodos de inactividad. Por último, verificamos que todas esas sumas de tiempo sean menores al 0,05 % del tiempo total del experimento para asegurarnos de que efectivamente se hayan introducido transacciones al sistema de manera constante.

2.5.2 Limitaciones de Hardware

La red es un sistema dinámico que modifica su estado en función del tiempo y la carga que esté soportando. En los experimentos emulamos una red de 240 nodos utilizando 4 servidores. Estos últimos tienen sus propias limitaciones, que si se alcanzaran, afectarían negativamente los resultados de la emulación.

Creamos dos herramientas a fin de determinar que el sistema se encuentra en un rango

de funcionamiento aceptable. Por un lado, un aplicación que mide el *bandwidth* (ancho de banda) usado de la placa de red del sistema. Una instancia de este proceso es iniciada en cada host físico.

Por otro lado, en cada host virtual iniciamos un *engine* que cada un período de tiempo le envía un paquete de 18 bytes a otro host virtual. El tiempo en que permanece dormido cada vez es aleatorio, siguiendo una distribución uniforme con media 1.5 s y varianza 0.5 s. La selección del host destino también es aleatoria y uniforme entre todos los posibles destinatarios.

Con la primer herramienta podemos ver que no se haya saturado el ancho de banda de la red, mientras que con la segunda podemos medir la varianza de la latencia de la red a través del tiempo.

Finalmente, cabe aclarar que el *overhead* que introduce la utilización de estas herramientas no afecta los resultados de los experimentos ya que el consumo de CPU por estas herramientas es despreciable. Con respecto a la red, ésta es utilizada solamente por el *engine* de ping, el cual en promedio agrega 3.2 kB/s al tráfico de la red. Como los equipos donde realizamos los experimentos tienen enlaces de red de 1 Gbit/s, 3.2 kB/s de datos no resulta en un *overhead* perceptible.

Capítulo 3

Experimentos

En este capítulo se detallarán los experimentos realizados, haciendo foco en el diseño experimental, los parámetros utilizados y los procesos involucrados en su ejecución.

Empezamos la sección 3.1 describiendo el diseño experimental utilizado. Luego pasamos a analizar varias configuraciones de protocolo que podrían ser utilizadas en experimentos. Por último detallamos todos los experimentos realizados, explicando por qué elegimos las configuraciones que se usaron en cada uno de ellos.

Para que un experimento pueda realizarse, es necesario llevar a cabo varios procesos que generan el escenario requerido. Primero hay que crear los hosts virtuales y agregar enlaces de red entre ellos para construir la topología IP deseada. Luego tenemos que inicializar todos los clientes de Bitcoin que forman parte del experimento, conectándolos acorde al modelo utilizado. Finalmente debemos instanciar un cliente extra que minará 576000 bloques, haciendo que el experimento se realice sobre un escenario similar al de la *mainnet*. En la sección 3.2 describimos todos estos procesos de inicialización, detallando las tecnologías usadas y los módulos de software desarrollados para este propósito. En la misma sección también especificamos programas auxiliares que escribimos para poder ejecutar los experimentos correctamente. Cada cliente de Bitcoin estará emparejado con una instancia de estos programas, los cuales estarán a cargo de introducir bloques y transacciones al sistema

acorde a la configuración de protocolo usada.

Por último, en la sección 3.3, especificamos el hardware utilizado para la realización de los experimentos.

3.1 Diseño experimental

Los experimentos de este trabajo consisten en una red de Bitcoin de 240 clientes, los cuales están minando y enviando transacciones nuevas a sus *peers* en todo momento. Luego de que la *mainchain* alcanza 1215 bloques de altura, detenemos los clientes y guardamos todos los datos estadísticos generados. La red de Bitcoin utilizada sigue el modelo propuesto en la sección 2.3, y los nodos utilizan el cliente descripto en la sección 2.2. En cada experimento vamos a establecer dos variables: (i) el tiempo entre bloques esperado, y (ii) la cantidad de transacciones que hay en promedio en cada bloque.

Modificamos el tiempo entre bloques para probar al sistema bajo distintos condiciones, tal como definimos en la sección 1.2.3. Además, para cada tiempo entre bloques evaluado, realizamos distintos experimentos, cada uno con una cantidad diferente de transacciones por bloque promedio. Esto nos permitió ver el funcionamiento de cada configuración de tiempo entre bloques bajo distintas condiciones de carga del sistema. Actualmente, el protocolo de Bitcoin establece un tiempo entre bloques esperado de 600 segundos y, usualmente, cada bloque contiene en promedio 2000 transacciones. Hicimos un experimento usando esta configuración y, también, reduciendo el tiempo entre bloques esperado a la mitad, un cuarto, un octavo y un dieciseisavo. Además de utilizar el valor promedio normal de 2000 transacciones en cada bloque, hicimos experimentos aumentando la carga del sistema a 4000 transacciones por bloque promedio, y disminuyéndola a 1000. Adicionalmente, bajo algunas configuraciones de tiempo entre bloques probamos el sistema procesando 6000 transacciones por bloque promedio, que es poco más del 80 % del máximo posible bajo los tamaños de bloque permitidos actualmente. Este escenario nos mostró el comportamiento de Bitcoin operando con un *throughput* cercano a su cota teórica.

Intentamos realizar experimentos en los que la cantidad de transacciones incluidas en cada bloque era, en promedio, entre un 90 % y 95 % del máximo posible. Sin embargo, en estos escenarios nunca se llegó a registrar que, en promedio, haya más transacciones por bloque que un 80 % de la mayor cantidad concebible. Esto sucedía porque, de a momentos, el desempeño del cliente se veía deteriorada producto de la alta cantidad de transacciones pendientes de confirmación en el sistema. Este deterioro reducía la capacidad del cliente para procesar nuevas transacciones, haciendo que se incorporen al sistema menos de las requeridas para llegar a la cantidad por bloque promedio deseada. Recordemos que, como vimos en la sección 1.2.4, debido a la naturaleza exponencial del proceso de minado, frecuentemente hay bloques que tardan en aparecer entre el triple y el quintuple de lo esperado, lo que causa que haya una gran cantidad de transacciones sin confirmar. En el capítulo 4, veremos que al procesar 6000 transacciones por bloque, ya se registra un deterioro notable en la calidad de la red, sugiriendo así que esta configuración resulta suficiente para probar el sistema bajo un escenario de congestión.

Al establecer un tiempo entre bloques y una cantidad de transacciones por bloque promedio, indirectamente también se establece la cantidad de transacciones por segundo que el sistema incorpora a la blockchain en promedio (es decir, el *throughput*).

En la tabla 3.1 podemos ver la cantidad de transacciones por segundo que procesa el sistema en promedio, bajo distintas combinaciones de configuraciones. Realizamos experimentos con combinaciones de configuraciones representados por casilleros con letra azul en la tabla. La configuración de 600 segundos con 1000 transacciones por bloque no fue considerada porque tiene un *throughput* muy bajo, y nos interesa ver cómo se comporta el sistema con cargas en el rango entre normales y altas. No probamos los parámetros que resultan en *throughputs* mayores a 60 Tx/s, porque podríamos encontrarnos con limitaciones de software que afecten los resultados. Adicionalmente a los casos que hay en la tabla, hicimos un experimento con tiempo entre bloques de 150 segundos y 1000 transacciones por bloque. Este último aporta un caso más del sistema funcionando con un *throughput* de $3,33$ Tx/s.

Como veremos en detalle en el capítulo 4, el conjunto de experimentos realizado nos per-

Carga del sistema		Transacciones por Bloque			
		1000	2000	4000	6000
Tiempo Entre Bloques (s)	37,5	26.67	53.33	106.67	160.00
	75	13.33	26.67	53.33	80.00
	150	6.67	13.33	26.67	40.00
	300	3.33	6.67	13.33	20.00
	600	1.67	3.33	6.67	10.00

Cuadro 3.1: *Throughput del sistema para distintas configuraciones de protocolo*

mite ver cómo se ven alteradas las medidas de la sección 2.4 al disminuir el tiempo entre bloques bajo dos condiciones: (i) La cantidad de transacciones por bloque está fija, y (ii) El *throughput* del sistema está fijo. El primer escenario se puede ver al comparar los experimentos que compartan columna en la tabla. El segundo se observa cuando se contrastan resultados de experimentos que están en una misma diagonal descendiente de la tabla.

3.2 Automatización de los experimentos

En [GM18] se introduce el lenguaje *fog*, que es usado para describir una topología de red IP y definir comandos a ejecutar en los hosts de dicha red. En el mismo trabajo se presenta `SherlockFog`, un programa que toma un script escrito en *fog* y ejecuta comandos de *shell* que logran una emulación de la topología deseada y ejecutan en los hosts virtuales lo especificado en el script. Cada nodo de la red generada es emulado mediante un *container* de Linux, que es un ambiente aislado adecuado para realizar experimentos. Utilizamos *SherlockFog* para emular la red IP y correr, de manera automatizada, los módulos necesarios para inicializar los experimentos.

El programa tiene la capacidad de utilizar varios hosts físicos para emular la red, en nuestro caso utilizaremos los cuatro servers descriptos en la sección 3.3.

Construimos un programa que toma un JSON que representa una instancia del modelo descripto en la sección 2.3, luego crea un script de de formato *fog*, y ejecuta `Sherlock Fog` usándolo como entrada. El script está diseñado para que `SherlockFog` inicialice un experimento con el escenario solicitado.

En los párrafos subsiguientes detallamos el proceso que automatiza los experimentos, los programas y módulos mencionados son lanzados por `SherlockFog` en los hosts emulados.

En primer lugar, se inicializan todos los clientes con dificultad 0 y cada uno agrega como peer a los nodos de Bitcoin requeridos para generar la topología deseada. Luego, una instancia más del cliente también con dificultad 0 es iniciada, ésta se conecta a todos los demás nodos de la red de Bitcoin del experimento y mina 576000 bloques. De esta manera, creamos un escenario más realista para los experimentos, ya que los clientes van a operar con una blockchain de tamaño similar al de la *mainnet*. Además, como los bloques iniciales van a ser generados para direcciones de los clientes del experimento, éstas van a tener muchos outputs y crédito para gastar durante el experimento.

Luego, para cada instancia de Bitcoin, se inicia un programa. Éste consiste en dos *threads* que interactúan con el cliente, de los cuales uno genera bloques y el otro transacciones.

El fin del *thread* 1 es implementar la simulación del proceso de minado de bloques mencionada en la sección 2.1. Para lograr esto cuenta con un ciclo infinito que: (i) genera un número al azar que representa una cantidad de milisegundos y se duerme por ese tiempo, (ii) le pide al cliente asignado que mine un bloque, el cual es generado inmediatamente. Para poder simular el comportamiento del escenario deseado, los números aleatorios generados por el *engine* del cliente siguen una distribución exponencial con parámetro $\lambda = h/t$. Siendo h la proporción de *hashing power* que tiene el cliente, y t el tiempo entre bloques esperado.

Por otro lado, para generar transacciones, el *thread* 2 del programa también ejecuta un ciclo infinito que se duerme por una cantidad aleatoria de milisegundos e interactúa con el cliente para introducir una transacción al sistema. Los números aleatorios generados en este procedimiento siguen una distribución uniforme. La media y varianza de la misma van a ser configuradas para que la cantidad de transacciones por bloque promedio del experimento se corresponda con los valores deseados.

En la figura 3.1 podemos ver retratada la interacción entre los *threads* del programa y el

cliente de Bitcoin.

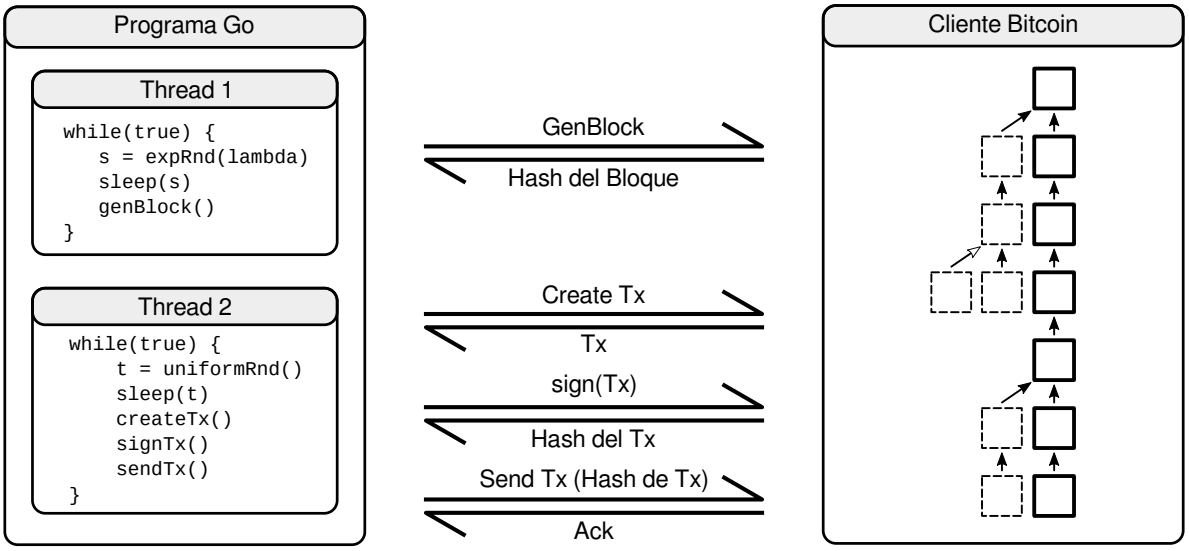


Figura 3.1: Diagrama de dependencias de variables en el sistema de Bitcoin

Como explicamos en la introducción, Bitcoin necesita aumentar drásticamente su *throughput* máximo posible para equipararse con los sistemas de pago tradicionales. Estos sistemas solamente utilizan operaciones de una persona a otra. En todos nuestros experimentos las transacciones tienen un solo input y un solo output, representando así el funcionamiento de un sistema de pago tradicional.

Al inicializar cada cliente de Bitcoin, generamos dos direcciones de pago nuevas vinculadas a su billetera digital. Al momento de construir la blockchain base inicial, cada bloque es generado de manera tal que la recompensa de minado es dada a alguna de estas $2 \cdot n$ direcciones. La distribución de crédito inicial se hace de manera equitativa, resultando en que todos los clientes tengan la misma cantidad de dinero disponible. Luego, cada vez que se genera una transacción mediante un cliente determinado, se asigna el output a una de las direcciones del mismo. Además, como input de la transacción se utiliza un output de la otra dirección del cliente. En cada operación se usa la menor cantidad posible de *fee*, para hacer así que el crédito inicial alcance para todo el experimento.

3.3 Infraestructura Utilizada

Todos los experimentos se fueron realizados en cuatro servidores `Dell PowerEdge C6145`, cada uno con las siguientes características:

- CPU: Cuatro procesadores AMD Opteron 6276 de 16 núcleos cada uno.
- RAM: 128 GB DDR3.
- SO: Debian GNU/Linux 7.
- Kernel: Linux 4,19.
- Interfaz de red: Ethernet de 1 Gbit/s.

En total, se utilizaron 256 núcleos para emular redes de Bitcoin con 240 mineros, los cuales tenían a su disposición un poco más de 2 GB de RAM cada uno.

Como detallamos en la sección 3.2, usamos `SherlockFog` para emular la topología de red IP, que permite distribuir los hosts de dicha red a través de distintas máquinas físicas. En nuestros experimentos utilizamos cada servidor para emular exactamente 60 mineros, es decir, repartimos equitativamente un cuarto de la emulación a cada uno.

Capítulo 4

Resultados

Comenzamos este capítulo analizando, para cada experimento realizado, los valores obtenidos de las medidas presentadas en la sección 2.4. Luego, demostramos que el tiempo de confirmación esperado de una transacción crece de manera exponencial a medida que sube la carga del sistema. Después agrupamos todos los experimentos en cuatro escenarios e hicimos comparaciones entre las mediciones registradas en experimentos que comparten un mismo grupo. Los experimentos fueron agrupados según su nivel de carga, todos los escenarios que comparten un mismo grupo procesaron la misma cantidad de transacciones por segundo. El primer grupo está compuesto por todos los experimentos que tuvieron una carga similar a la normal observada en la *mainnet*. El segundo comprende aquellos experimentos que funcionaron con un nivel de carga considerado como elevado para los estándares de Bitcoin actuales. Los últimos dos representan escenarios con niveles de carga que no son alcanzables con la configuración de protocolo de la *mainnet*.

Como vimos en la sección 3.1, cada experimento realizado se corresponde con una de las combinaciones de configuraciones de las entradas de color azul de la tabla 3.1. Adicionalmente, también realizamos un experimento con 150 segundos de tiempo entre bloques y 500 transacciones por bloque.

Carga del sistema		Transacciones por Bloque			
		1000	2000	4000	6000
Tiempo Entre Bloques (s)	37,5	26.67	53.33	106.67	160.00
	75	13.33	26.67	53.33	80.00
	150	6.67	13.33	26.67	40.00
	300	3.33	6.67	13.33	20.00
	600	1.67	3.33	6.67	10.00

Carga del sistema para distintas configuraciones de protocolo - *Cuadro 3.1 de la sección 3.1*

Como discutimos en la sección 1.2.3, disminuir el tiempo entre bloques esperado indirectamente altera otras propiedades del sistema. En resumen, reducir el tiempo entre bloques aumenta la probabilidad de ocurrencia de *forks*, los cuales incrementan el *wasted hashing power* de la red. Veamos ahora cómo se comportaron las magnitudes presentadas en la sección 2.4, para cada uno de los experimentos realizados.

En la cuarta tabla del cuadro 4.1 se aprecia que los tiempos de confirmación de transacciones están fuertemente correlacionados con el tiempo entre bloques. En el cuadro 4.2 se muestran los valores de la última tabla del cuadro 4.1 como porcentajes del tiempo entre bloques del experimento al que pertenecen. Esta información sugiere que los tiempos de confirmación están correlacionados con la cantidad de transacciones que, en promedio, se incluyen en cada bloque. Además, esta relación parece tener un carácter exponencial.

En la sección 1.2.4 se observó que, frecuentemente, hay bloques que tardan en aparecer mucho más de lo esperado. De hecho, supongamos que se están introduciendo, en promedio, 6000 transacciones nuevas al sistema entre cada bloque. Además, sea t el tiempo entre bloques esperado, como en un bloque entran hasta 7483 transacciones, si alguno tarda en aparecer más de $z = 7483/6000 \cdot t$ segundos, inevitablemente quedarán transacciones pendientes que tendrán que ser procesadas en el próximo bloque. Debido a que los tiempos entre bloques siguen una distribución exponencial, la probabilidad de que un bloque tarde en más de z segundos en aparecer es 0,28732. Entonces, la probabilidad de que dos bloques seguidos tarden cada uno más de z segundos en aparecer es $0{,}28732^2 \simeq 0{,}08255$. Esta situación se espera que pase aproximadamente una vez cada 12 bloques. Dado que nuestros experimentos consisten en el minado de 1215 bloques, este escenario de dos bloques satu-

Tiempos de Aceptación de Bloques (Percentil 90)		Transacciones por Bloque				
		500	1000	2000	4000	6000
Tiempo Entre Bloques (s)	37,5		28.08	43.07		
	75		10.55	31.75	42.55	
	150	2.15	3.10	13.42	34.48	35.90
	300		2.53	6.00	17.82	24.03
	600			3.41	9.47	19.59

Cantidad de Forks		Transacciones por Bloque				
		500	1000	2000	4000	6000
Tiempo Entre Bloques (s)	37,5		573	843		
	75		139	360	469	
	150	14	18	92	201	202
	300		10	22	64	80
	600			10	26	32

Porcentaje de Wasted Hashing Power		Transacciones por Bloque				
		500	1000	2000	4000	6000
Tiempo Entre Bloques (s)	37,5		24.13	31.33		
	75		5.27	15.03	19.56	
	150	0.92	1.05	3.80	9.31	9.39
	300		0.54	0.76	2.39	3.59
	600			0.30	0.76	1.81

Tiempos de confirmación de transacciones (Seg)		Transacciones por Bloque				
		500	1000	2000	4000	6000
Tiempo Entre Bloques (s)	37,5		40.49	44.17		
	75		85.75	82.47	82.18	
	150	151.87	151.03	153.46	192.67	191.17
	300		292.22	322.20	416.64	599.64
	600			602.85	788.10	1460.97

Cuadro 4.1: *Tablas de resultados seleccionados*

		Transacciones por Bloque				
		500	1000	2000	4000	6000
Tiempo Entre Bloques (s)	37,5		107,97 %	117,79 %		
	75		114,33 %	109,96 %	109,57 %	
	150	101,25 %	100,69 %	102,31 %	128,45 %	127,45 %
	300		97,41 %	107,41 %	138,88 %	199,88 %
	600			100,48 %	131,35 %	243,50 %

Cuadro 4.2: *Tiempos de confirmación de transacciones, como porcentaje del tiempo entre bloques, para distintas configuraciones de protocolo*

rados seguidos debería darse más de 100 veces. Por lo tanto era esperable que los tiempos de confirmación se hubiesen incrementado significativamente en los experimentos de 6000 transacciones por bloque.

Veamos ahora que los tiempos de confirmación de las transacciones aumentan de manera exponencial a medida que se incrementa la carga del sistema. Primero, recordemos que se pueden incluir a lo sumo 7483 transacciones en un bloque. Supongamos ahora que están siendo enviadas al sistema X operaciones nuevas por segundo. Entonces, en $7483/X = Z$ segundos, ingresan al sistema la máxima cantidad de transacciones que entran en un bloque. Por lo tanto, si un bloque tarda más de Z segundos en aparecer, inevitablemente se va a saturar, es decir, no va a poder contener todas las transacciones que hay sin confirmar en el sistema. Por otro lado, supongamos que una nueva transacción c entra al sistema. Definamos a la variable aleatoria K como la cantidad de bloques que aparecen hasta que uno incluye a c. $P(K = n)$ es la probabilidad de que haya exactamente $n - 1$ bloques saturados luego de que c entra al sistema. Estos $n - 1$ bloques contendrán operaciones que fueron enviadas al sistema antes que c. Para que $n - 1$ bloques se llenen de transacciones previas a c, tiene que haber al menos $(n - 1) \cdot 7483$ transacciones pendientes en el momento que c es introducida. Llamemos S_m a la sumatoria de los tiempos que tardaron en aparecer los últimos m bloques, y definamos E como el tiempo que transcurrió desde el minado del último bloque. Definiendo $R_m = S_m + E$, si $R_m \geq (m + n - 1) \cdot 7483/X$, entonces inevitablemente va a haber como mínimo $(n-1) \cdot 7483$ transacciones pendientes de confirmación. R_m sigue una distribución exponencial, ya que es la sumatoria de variables aleatorias exponenciales. Entonces podemos afirmar que K también es una variable aleatoria exponencial, ya que tomando a m como la cantidad de bloques que aparecieron desde el último que no estuvo saturado, $P(K = n) = P(R_m \geq (m+n-1) \cdot 7483/X)$. Además, como X se encuentra dividiendo en la expresión anterior, $P(K = n)$ aumenta de manera exponencial a medida que se incrementa la carga del sistema. Si una transacción tiene que esperar al minado de n bloques para ser confirmada, entonces su tiempo de confirmación va a ser la sumatoria de tiempos que tardaron en aparecer cada uno de los n bloques. Por

lo tanto, si denominamos T al tiempo entre bloques promedio, el tiempo de confirmación esperado de una transacción va a ser $\sum_{i=0}^{\infty} P(K = i) \cdot T \cdot i$. Como este valor es una sumatoria de variables que aumentan exponencialmente a medida que se incrementa X, concluimos que el tiempo de confirmación esperado de una transacción crece de manera exponencial a medida que sube la carga del sistema.

El análisis del párrafo anterior supone que las transacciones son incluidas a la blockchain de manera *FIFO*, que es lo que sucede cuando todas ofrecen la misma comisión. Analizar el sistema funcionando con transacciones que tienen distintas comisiones está fuera del alcance del presente trabajo. Sin embargo, en el capítulo 1 contamos que cuando Bitcoin experimentó una fuerte suba en la cantidad de transacciones enviadas al sistema, los tiempos de confirmación llegaron a ser más de 30 horas (es decir, un 19890 % del tiempo entre bloques). Esta situación refleja el comportamiento exponencial esperado.

Por otro lado, observemos que en las primeras tres tablas del cuadro 4.1 se cumple la siguiente propiedad: Para todas las submatrices m que poseen un valor numérico n en su posición inferior izquierda, n es menor estricto a los demás valores de m. Es decir, en los cuadros mencionados, los valores incrementan en sentido hacia la esquina superior derecha. Esta misma propiedad es válida en el cuadro 3.1, apoyando la teoría de que existe una correlación entre el *throughput* y las medidas mencionadas. En las siguientes secciones vamos a comparar los valores obtenidos en los experimentos que tienen la misma carga, pero distinto tiempo entre bloques.

4.1 Throughput normal de Bitcoin

Veamos primero resultados de experimentos con configuraciones que resultan en un *throughput* de 3,33 Tx/s, que es la carga normal de Bitcoin. De esta forma podemos ver cómo se verían alteradas las métricas de la sección 2.4, si se cambiase el tiempo entre bloques bajo el contexto actual.

Los tres escenarios analizados en esta sección son:

- 600 segundos de tiempo entre bloques, 2000 transacciones promedio por bloque.
- 300 segundos de tiempo entre bloques, 1000 transacciones promedio por bloque.
- 150 segundos de tiempo entre bloques, 500 transacciones promedio por bloque.

En la figura 4.3 se puede ver que la cantidad de *forks* fue la misma en los experimentos de 600 y 300 segundos. Mientras que, para el caso de 150 segundos, se registró una suba del 40 %.

En las figuras 4.1 y 4.2, vemos que los tiempos de propagación de la red fueron mayores en el experimento de 600 segundos.

En los casos de 150 y 300 segundos, los bloques demoraron una cantidad de tiempo similar en llegar al 70 % de los nodos de la red. Luego, con respecto al 30 % restante, el experimento de 300 segundos registró mayores tiempos de propagación que el de 150.

Recordemos que un *fork* ocurre cuando un nodo mina un bloque durante el tiempo de propagación de otro bloque. Es decir, si un nodo genera un bloque b, se va a producir un *fork* si otro nodo descubre un bloque antes de conocer la existencia de b. Dicho de otro modo, entre que se genera b hasta que un nodo v se entera de su existencia, hay una ventana de tiempo en la que podría ocurrir un *fork*.

Los datos de tiempos de propagación son congruentes con los *forks* que hubo en cada experimento. En los experimentos de 150 y 300 segundos, las ventanas de tiempo en las que podían ocurrir *forks* eran similares, pero un tiempo entre bloques menor resulta en una mayor probabilidad de que se mine un bloque durante ese tiempo. Entonces, es esperable que el escenario con tiempo entre bloques de 150 segundos haya generado más *forks* que el de 300 segundos. Por otro lado, el experimento de 600 segundos tuvo mayores tiempos de propagación que el de 300, es decir, la ventana de tiempo en la cual se podían generar *forks* fue mayor. Sin embargo, como la aparición bloques en un mismo período de tiempo corto es menos probable en el escenario de 600 segundos, es razonable que se hayan registrado

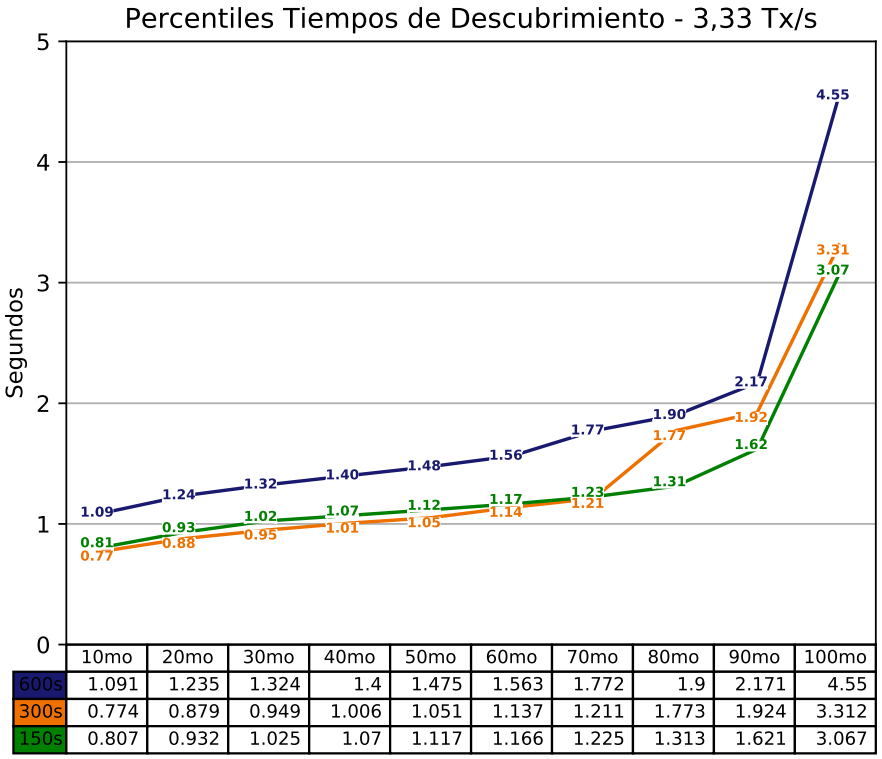
la misma cantidad de *forks* en ambos escenarios.

Por último, en la figura 4.4 vemos que el *wasted hashing power* aumenta a medida que decrece el tiempo entre bloques. Como los experimentos de 150 y 300 segundos tuvieron tiempos de propagación similares, en esos escenarios se desperdiciaba una cantidad similar de *hashing power* en la generación de cada bloque. Sin embargo, este desperdicio ocurría, en promedio, una vez cada 150 segundos en un experimento y una vez cada 300 segundos en otro.

Es decir, cada 300 segundos, el escenario con tiempo entre bloques de 150 segundos pasaba más tiempo desperdiciando *hashing power* que el caso de 300 segundos. Además, el experimento de 150 segundos duró en total aproximadamente $150 \cdot 1215$ segundos, mientras que el de 300 segundos duró aproximadamente el doble. Por lo tanto, desperdiciar un segundo de *hashing power* pesa más en el experimento de 150 segundos que en el de 300. También, en el experimento de 150 segundos hubo más *forks* que en el de 300, haciendo que suba más el *wasted hashing power*. Todo esto se condice con lo registrado: el experimento de 150 segundos desperdició un 71 % más de *hashing power* que el de 150 segundos. Con respecto al experimento de 600 segundos, tuvo igual cantidad de *forks* que el de 300 segundos, y los tiempos de propagación fueron mayores que el de 300 segundos. Sin embargo, para que el *wasted hashing power* sea igual en los experimentos de 300 y 600 segundos, este último debería haber tenido tiempos de propagación cercanos al cuádruple del caso de 300 segundos. Como el incremento de los tiempos de propagación fue mucho menor que un 300 % más, el experimento de 300 segundos registró un un 77 % más de *wasted hashing power* que el de 600 segundos.

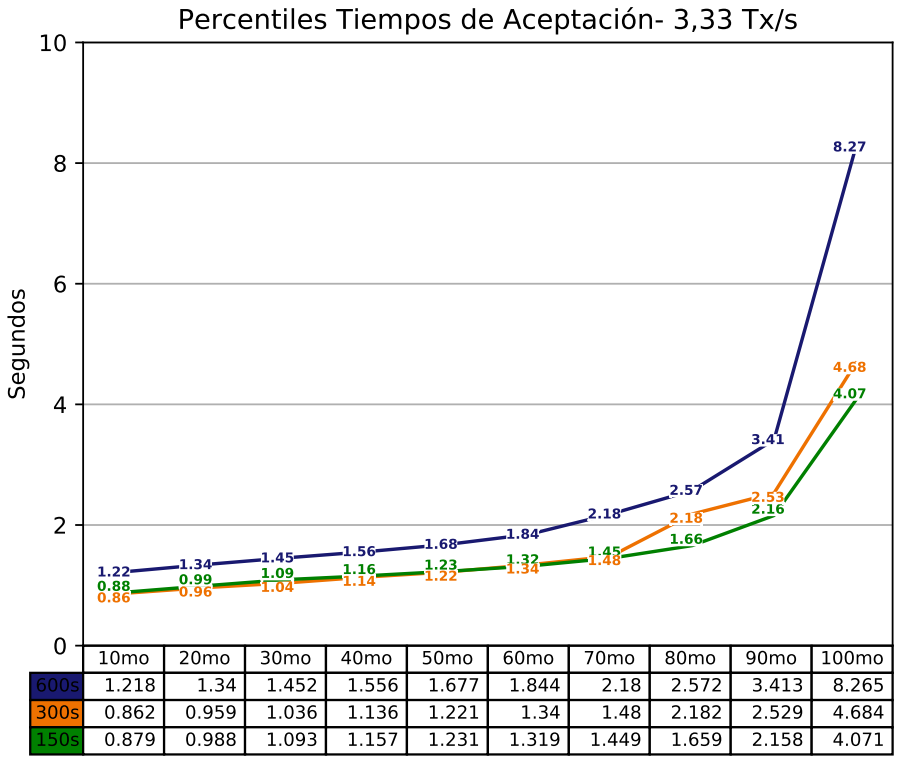
Todos los valores observados en esta sección son del mismo orden de magnitud que los registrados en la *mainnet*. Por lo tanto, la red de Bitcoin podría funcionar perfectamente con las tres configuraciones de protocolo utilizadas en estos experimentos. Reducir el tiempo entre bloques disminuye los tiempos de confirmación de las transacciones, ya que en promedio hay que esperar menos para la aparición de futuros bloques. Pero también incrementa el *wasted hashing power*, elevando así el costo económico de mantener funcionando

la red. El hecho de si conviene pagar más para tener tiempos de confirmación menores es una decisión de negocios, ya que reducir el tiempo de confirmación resulta en una mejor experiencia de usuario.



	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
600s	1.091	1.235	1.324	1.4	1.475	1.563	1.772	1.9	2.171	4.55
300s	0.774	0.879	0.949	1.006	1.051	1.137	1.211	1.773	1.924	3.312
150s	0.807	0.932	1.025	1.07	1.117	1.166	1.225	1.313	1.621	3.067

Figura 4.1: Percentiles de tiempos de descubrimiento de bloques en experimentos con una carga de 3,33 Tx/s



	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
600s	1.218	1.34	1.452	1.556	1.677	1.844	2.18	2.572	3.413	8.265
300s	0.862	0.959	1.036	1.136	1.221	1.34	1.48	2.182	2.529	4.684
150s	0.879	0.988	1.093	1.157	1.231	1.319	1.449	1.659	2.158	4.071

Figura 4.2: Percentiles de tiempos de aceptación de bloques en experimentos con una carga de 3,33 Tx/s

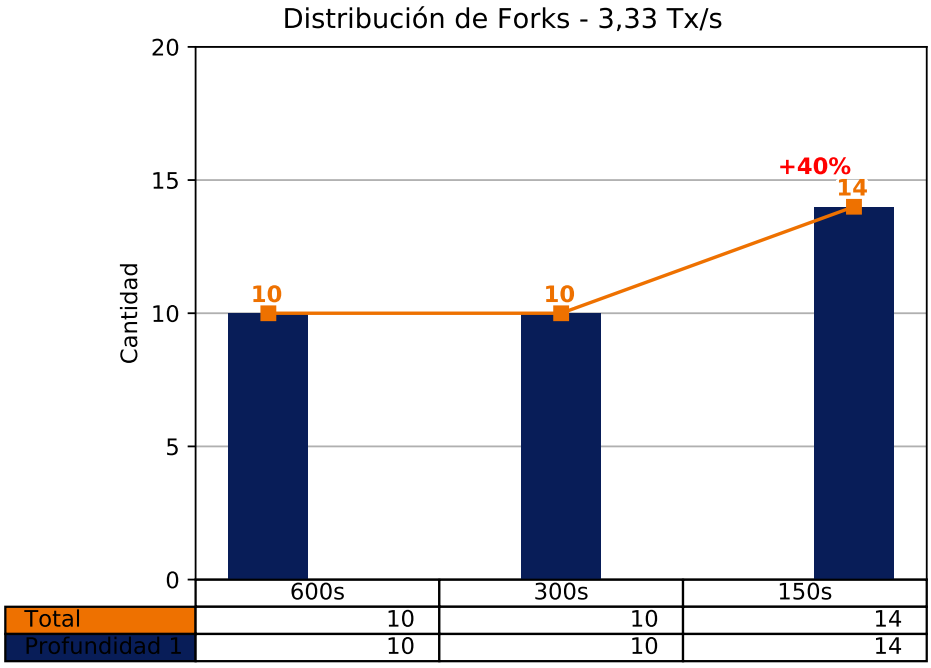


Figura 4.3: Distribución de forks en la blockchain de experimentos con una carga de 3,33 Tx/s

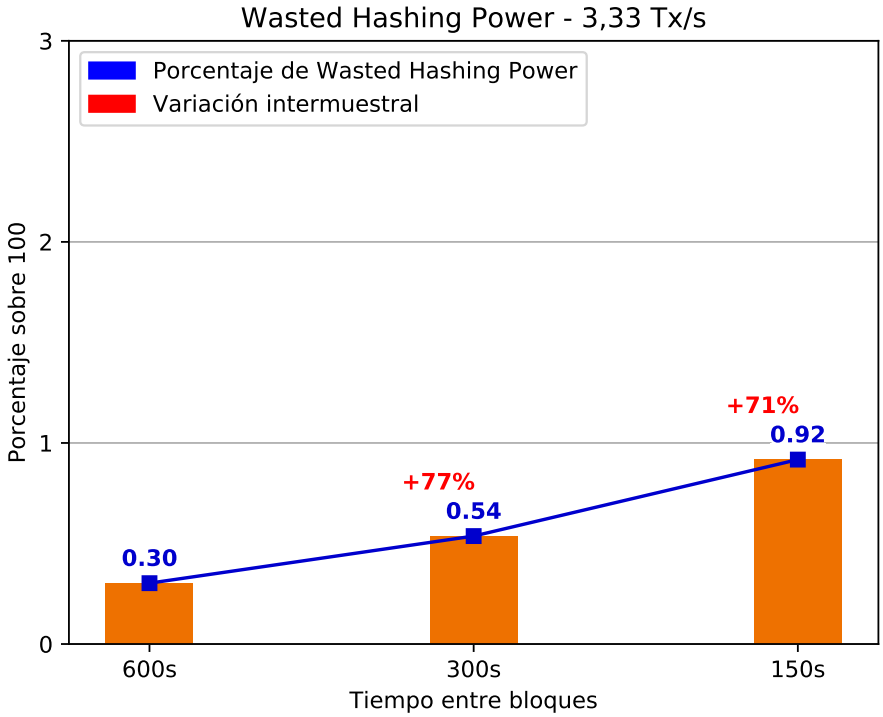


Figura 4.4: *Wasted hashing power* en experimentos con una carga de 3,33 Tx/s

4.2 Límites de funcionamiento estable

En la sección 1.2.1 vimos que el máximo *throughput* teórico de Bitcoin es de 12,47 transacciones por segundo. Sin embargo, en el cuadro 4.2 vemos que, con 600 segundos de tiempo entre bloques y 4000 transacciones por bloque, los tiempos de confirmación de las transacciones fueron un 31,35 % mayores al tiempo entre bloques. Es decir, usando el tiempo entre bloques actual de la *mainnet* y con el sistema bajo una carga de 6,67 Tx/s, ya percibimos una leve suba en los tiempos de confirmación de las transacciones. Si bien este incremento en los tiempos de confirmación es tolerable por los usuarios de la red, un *throughput* mayor bajo el mismo tiempo entre bloques resultaría en un deterioro no despreciable de la calidad del servicio.

Catalogamos como "alto" a un *throughput* de 6,67 Tx/s bajo un tiempo entre bloques de 600 segundos, ya que está cerca del límite del buen funcionamiento de la red con respecto a los tiempos de confirmación. En esta sección vamos a comparar distintas configuraciones de protocolo operando con una carga de 6,67 Tx/s. De esta manera, todos los escenarios que evaluemos no van a estar saturados, asegurándonos así que las demás métricas no estén alteradas por situaciones de congestión.

Los tres escenarios comparados en esta sección son:

- 600 segundos de tiempo entre bloques, 4000 transacciones promedio por bloque.
- 300 segundos de tiempo entre bloques, 2000 transacciones promedio por bloque.
- 150 segundos de tiempo entre bloques, 1000 transacciones promedio por bloque.

En la figura 4.7 vemos un patrón distinto al observado en la sección 4.1, ya que la cantidad de *forks* decrece a medida que disminuye el tiempo entre bloques usado. Sin embargo, el único experimento que registró un *fork* de profundidad 2 fue el de 150 segundos.

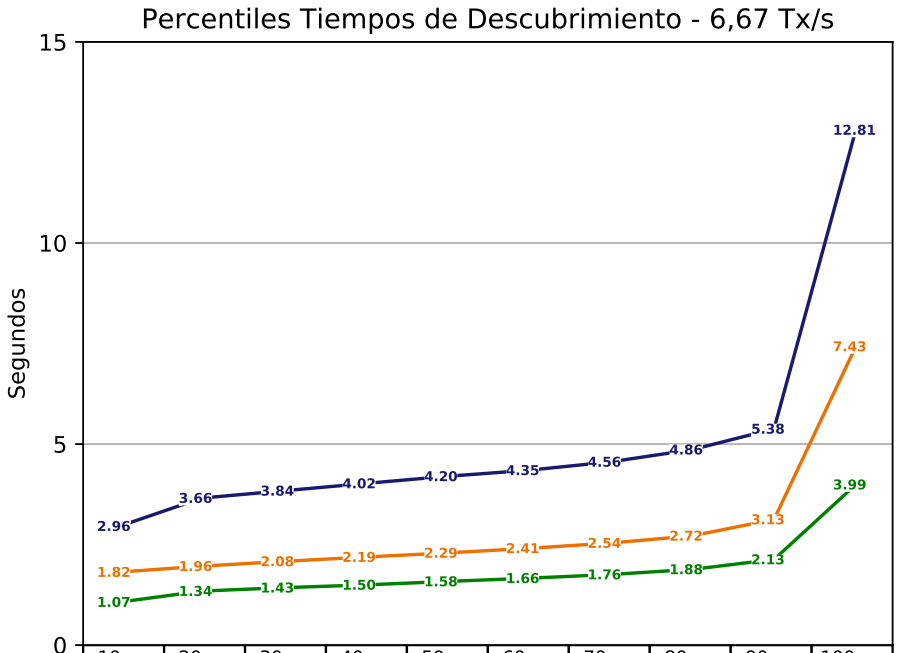
Un minero debe tener todas las transacciones incluidas en un bloque para poder aceptarlo, y lo anuncia a sus *peers* una vez que lo acepta. Además, cada vez que disminuye el tiempo entre bloques también decrementa acorde la cantidad de transacciones en cada bloque (para

mantener la misma carga). Entonces, es esperable que los tiempos de propagación crezcan a medida de que aumenta el tiempo entre bloques, tal como se observa en las figuras 4.5 y 4.6. También podemos apreciar que el incremento entre los experimentos de 150 y 300 segundos fue menor que el observado entre los escenarios de 300 y 600 segundos.

En la figura 4.8 vemos que el *wasted hashing power* se mantuvo más estable que en el conjunto de experimentos de la sección 4.1. Además, se observan valores similares de *wasted hashing power* en los casos de 600 y 300 segundos. El experimento con tiempo entre bloques de 150 segundos desperdició un 38 % más de *hashing power* que el de 300 segundos, lo cual representa la mitad de aumento comparado al observado en la sección 4.1.

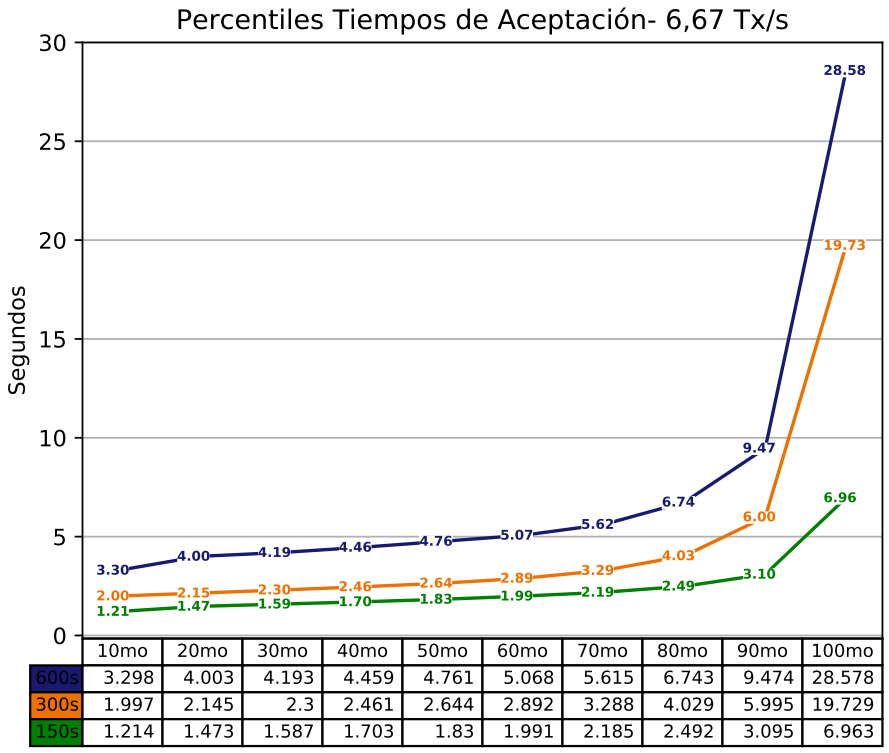
Los valores de *wasted hashing power* registrados se pueden fundamentar con las siguientes dos observaciones. En primer lugar, la cantidad de *forks* bajó a medida que se decrementó el tiempo entre bloques, lo cual resulta en un descenso del porcentaje de *hashing power* que se desperdició resolviendo forks. Por otro lado, el aumento intermuestral de los tiempos de propagación fue mayor al observado en la sección 4.1, elevando así el porcentaje de *hashing power* desperdiciado esperando que se transmitan los bloques.

Los datos arrojados por los experimentos sugieren que, bajo una carga de 6,67 Tx/s, conviene utilizar un tiempo entre bloques de 300 segundos. El costo económico de mantener la red en funcionamiento con esta configuración es similar al de usar 600 segundos como tiempo entre bloques. Sin embargo, en el experimento de 600 segundos, cada bloque incluía en promedio 4000 transacciones, lo que significa que no se podría aumentar mucho la carga sin caer en un escenario de saturación. En cambio, con un tiempo entre bloques de 300 segundos, se incluyen en promedio 2000 transacciones por bloque, que es un valor lejano al punto de saturación del sistema. Esto significa que con un tiempo entre bloques de 300 segundos, se podría duplicar la carga, manteniendo a su vez el buen funcionamiento del sistema y llegando así a un valor de *throughput* mayor que el límite teórico cuando se usa 600 segundos de tiempo entre bloques.



	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
600s	2.959	3.657	3.838	4.02	4.198	4.349	4.557	4.861	5.378	12.811
300s	1.817	1.963	2.082	2.19	2.293	2.407	2.538	2.718	3.126	7.428
150s	1.071	1.344	1.428	1.497	1.583	1.664	1.756	1.884	2.13	3.992

Figura 4.5: Percentiles de tiempos de descubrimiento de bloques en experimentos con una carga de 6,67 Tx/s



	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
600s	3.298	4.003	4.193	4.459	4.761	5.068	5.615	6.743	9.474	28.578
300s	1.997	2.145	2.3	2.461	2.644	2.892	3.288	4.029	5.995	19.729
150s	1.214	1.473	1.587	1.703	1.83	1.991	2.185	2.492	3.095	6.963

Figura 4.6: Percentiles de tiempos de aceptación de bloques en experimentos con una carga de 6,67 Tx/s

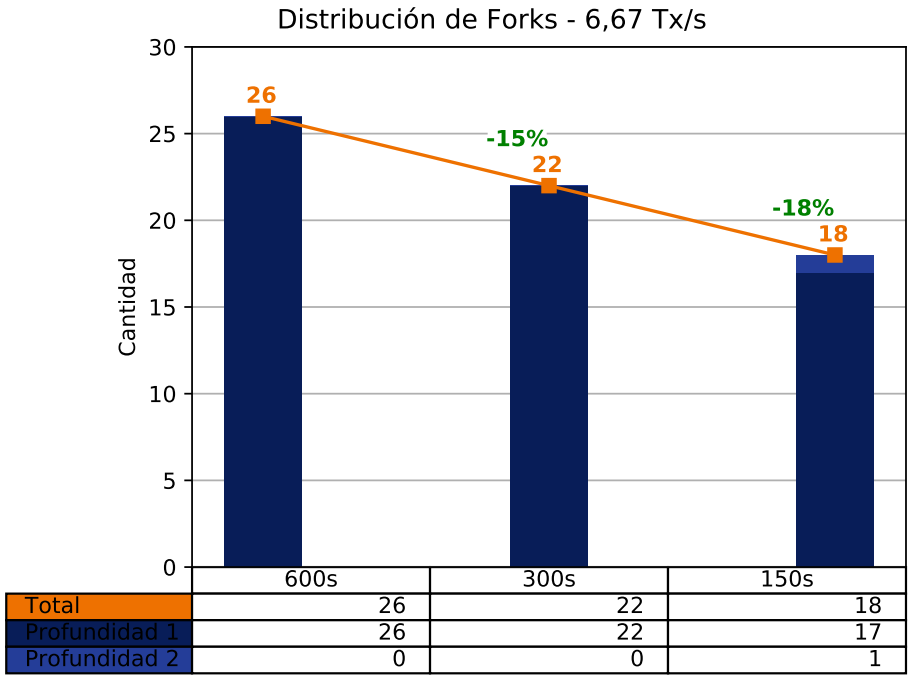


Figura 4.7: Distribución de forks en la blockchain de experimentos con una carga de 6,67 Tx/s

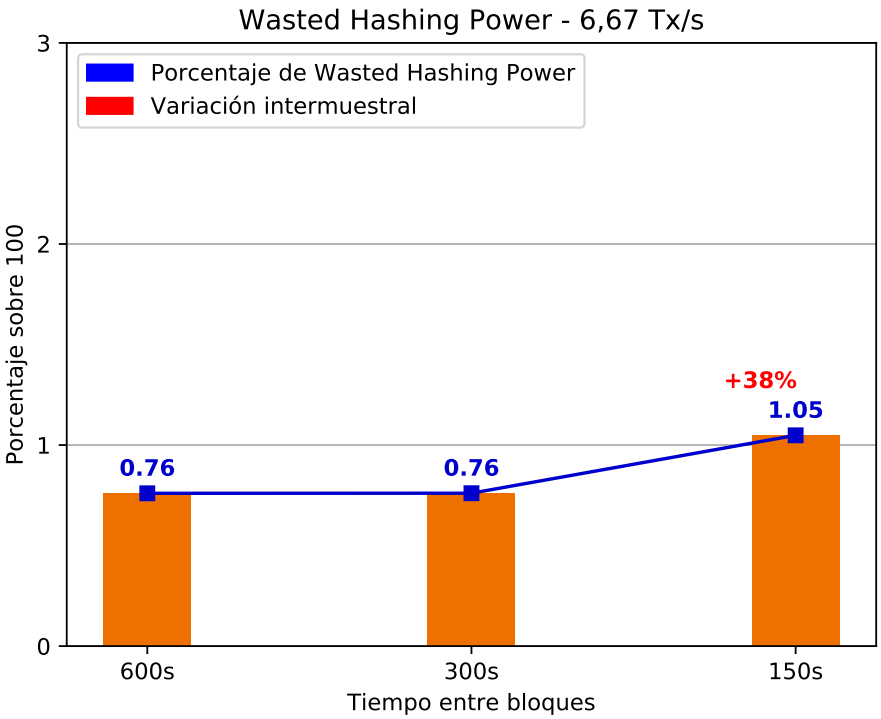


Figura 4.8: *Wasted hashing power* en experimentos con una carga de 6,67 Tx/s

4.3 Throughputs mayores al límite teórico actual

En esta sección analizaremos dos conjuntos de experimentos que plantean escenarios con el sistema de Bitcoin funcionando bajo condiciones que no son alcanzables usando la configuración de protocolo actual de la *mainnet*. El primer conjunto está comprendido por experimentos con configuraciones que procesan, en promedio, $13,33$ transacciones por segundo. El segundo contiene experimentos con el sistema funcionando bajo una carga de $26,67$ Tx/s. En ambos casos vamos a evaluar cuán posible es lograr procesar más transacciones por segundo que el límite teórico actual, manteniendo un nivel de calidad de red aceptable para los usuarios.

Los tres escenarios del primer conjunto analizado en esta sección son:

- 300 segundos de tiempo entre bloques, 4000 transacciones promedio por bloque.
- 150 segundos de tiempo entre bloques, 2000 transacciones promedio por bloque.
- 75 segundos de tiempo entre bloques, 1000 transacciones promedio por bloque.

El segundo conjunto contiene experimentos con las siguientes tres configuraciones:

- 150 segundos de tiempo entre bloques, 4000 transacciones promedio por bloque.
- 75 segundos de tiempo entre bloques, 2000 transacciones promedio por bloque.
- 37,5 segundos de tiempo entre bloques, 1000 transacciones promedio por bloque.

En la figura 4.9 podemos ver que se registraron muchos más *forks* que en los experimentos evaluados en las secciones anteriores, llegando a haber un orden de magnitud más en el caso de 75 segundos de tiempo entre bloques. Al igual que en los casos de la sección 4.1, vemos que la cantidad de *forks* aumentó a medida que disminuyó el tiempo entre bloques.

En los experimentos de $13,33$ Tx/s, los tiempos de propagación también aumentaron a medida que disminuyó el tiempo entre bloques. El incremento registrado entre los escenarios con 150 y 300 segundos de tiempo entre bloques fue similar al observado entre los experimentos de 300 y 600 segundos.

En la figura 4.12 observamos que, en todos los escenarios con una carga de 13, 33 Tx/s, el *wasted hashing power* de la red fue un orden de magnitud superior al registrado con los parámetros actuales de la *mainnet*. Al igual que en las secciones anteriores, vemos que el *wasted hashing power* tiende a incrementar a medida de disminuye el tiempo entre bloques.

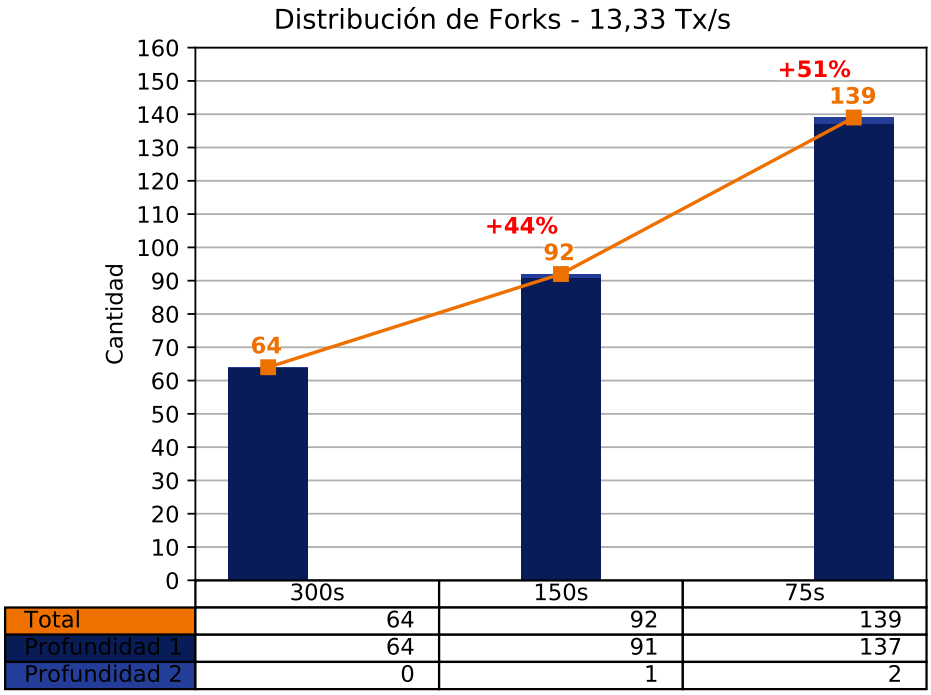
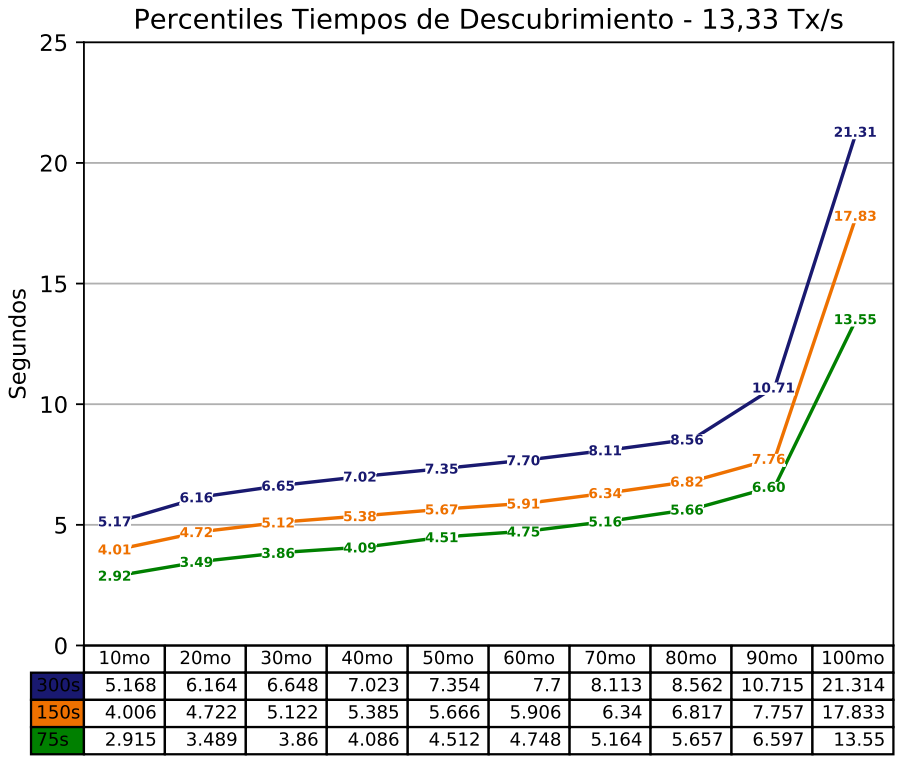


Figura 4.9: Distribución de forks en la blockchain de experimentos con una carga de 13,33 Tx/s

Con respecto a las configuraciones que resultan en una carga de 26,67 Tx/s, vemos en la figura 4.13 que todos los experimentos registraron una alta cantidad de *forks*. En comparación al escenario configurado con los parámetros y la carga normal de la *mainnet*, el experimento de tiempo entre bloques de 150 segundos tuvo aproximadamente 20 veces más *forks*. Siguiendo la misma comparación, los experimentos con tiempo entre bloques de 75 y 37,5 segundos registraron aproximadamente 36 y 57 veces más *forks* respectivamente. Es decir, en todos los escenarios hubo un orden de magnitud más de forks que en el experimento realizado con los parámetros y carga normal de la *mainnet*.

En las figuras 4.15 y 4.16 se observan tiempos de propagación similares en los casos con



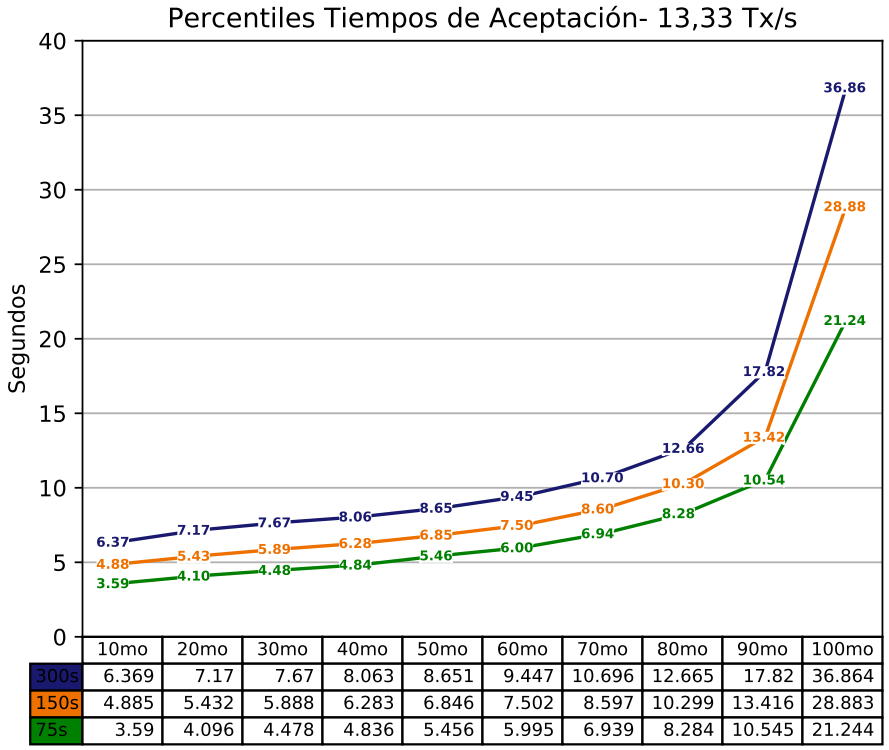
	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
300s	5.168	6.164	6.648	7.023	7.354	7.7	8.113	8.562	10.715	21.314
150s	4.006	4.722	5.122	5.385	5.666	5.906	6.34	6.817	7.757	17.833
75s	2.915	3.489	3.86	4.086	4.512	4.748	5.164	5.657	6.597	13.55

Figura 4.10: Percentiles de tiempos de descubrimiento de bloques en experimentos con una carga de 13,33 Tx/s

150 y 75 segundos de tiempo entre bloques. En cambio, en el escenario con 37,5 segundos de tiempo entre bloques, los tiempos de propagación fueron levemente menores a los registrados en los dos experimentos anteriores.

Se registró una gran cantidad de *wasted hashing power* en todos los experimentos realizados bajo una carga de 26,67 Tx/s. En la figura 4.14, vemos que los nodos del escenario con 150 segundos de tiempo entre bloques desperdiciaron 9,31 % de su *hashing power*. También podemos observar que el *wasted hashing power* se incrementó en un 61 % cada vez que se redujo el tiempo entre bloques a la mitad. Esto resultó en que los experimentos con tiempos entre bloques de 75 y 37,5 segundos tuvieran un *wasted hashing power* de 15,03 % y 24,13 % respectivamente.

Estos resultados sugieren que no resulta viable mantener a Bitcoin funcionando bajo una carga de 26,67 Tx/s, ya que la calidad de la red sufre un gran deterioro.



	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
300s	6.369	7.17	7.67	8.063	8.651	9.447	10.696	12.665	17.82	36.864
150s	4.885	5.432	5.888	6.283	6.846	7.502	8.597	10.299	13.416	28.883
75s	3.59	4.096	4.478	4.836	5.456	5.995	6.939	8.284	10.545	21.244

Figura 4.11: Percentiles de tiempos de aceptación de bloques en experimentos con una carga de 13,33 Tx/s

Los valores obtenidos en los experimentos con una carga de 13,33 Tx/s también son elevadas, aunque quizás tolerables en el escenario de 300 segundos de tiempo entre bloques. Sin embargo, otros factores ponen en duda que la red pueda funcionar procesando esa cantidad de transacciones por segundo. En el experimento con tiempo entre bloques de 300 segundos, en promedio hay 4000 transacciones en cada bloque. Como vimos al principio de este capítulo, que los bloques tengan esa cantidad de transacciones resulta en un deterioro del tiempo de confirmación de las mismas. Por otro lado, los experimentos de 150 y 75 segundos de tiempo entre bloques presentan signos de deterioro no despreciables. Ambos registraron *forks* de profundidad 2, y en los dos el *wasted hashing power* fue de más de diez veces superior al observado en el experimento realizado con los parámetros y carga normal de la *mainnet*. Por lo tanto, si bien no es imposible mantener el sistema funcionando con una carga de 13,33 Tx/s, podemos afirmar que en dicho caso estaríamos cerca del límite del máximo *throughput* alcanzable mediante la modificación de los parámetros de la red.

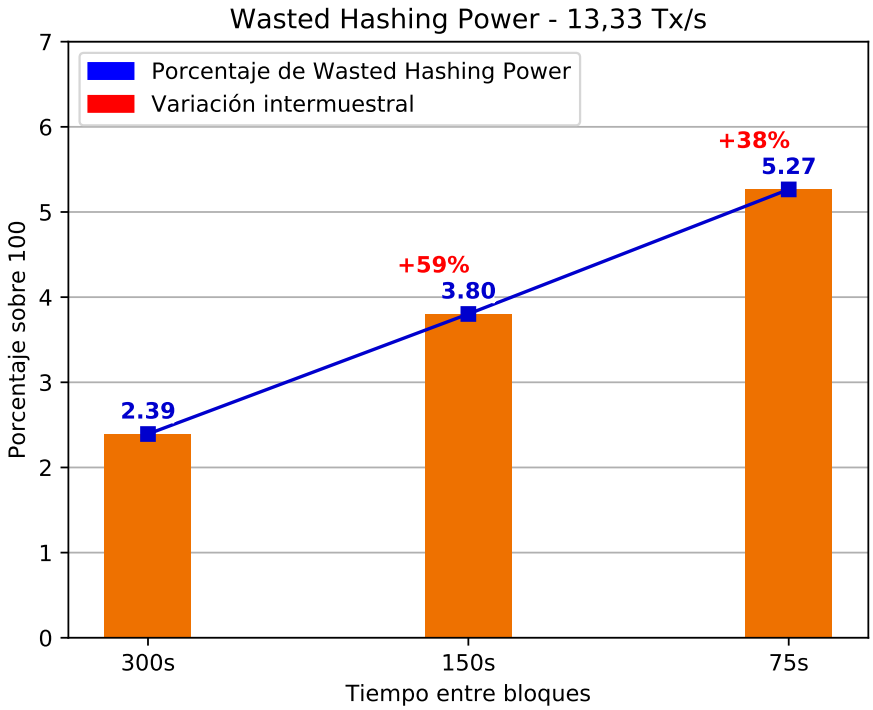
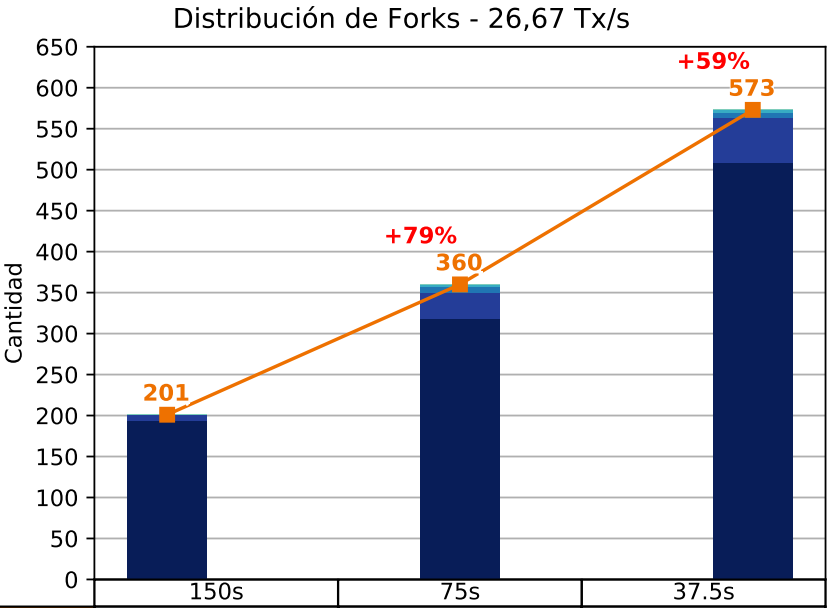


Figura 4.12: *Wasted hashing power* en experimentos con una carga de 13,33 Tx/s

Adicionalmente, en los experimentos que utilizaron configuraciones de tiempo entre bloques de 75 y 37,5 segundos, se observó un gran deterioro de la calidad del sistema, el cual se manifestó, incluso, cuando se procesaron únicamente 1000 transacciones en cada bloque (un 13,36 % de la capacidad de los mismos). En ambos escenarios se registraron *forks* con altura mayor a uno y el *wasted hashing power* fue de un orden de magnitud superior al que presenta la *mainnet* normalmente. Estos resultados sugieren que existe un límite inferior del tiempo entre bloques con el que el sistema puede ser configurado, sin sufrir un gran deterioro de la calidad de la red y sin que se genere un incremento significativo en el costo económico de mantenerlo en funcionamiento.



	150s	75s	37.5s
Total	201	360	573
Profundidad 1	194	318	509
Profundidad 2	7	32	54
Profundidad 3	0	7	7
Profundidad 4	0	2	3
Profundidad 5	0	1	0

Figura 4.13: Distribución de forks en la blockchain de experimentos con una carga de 26,67 Tx/s

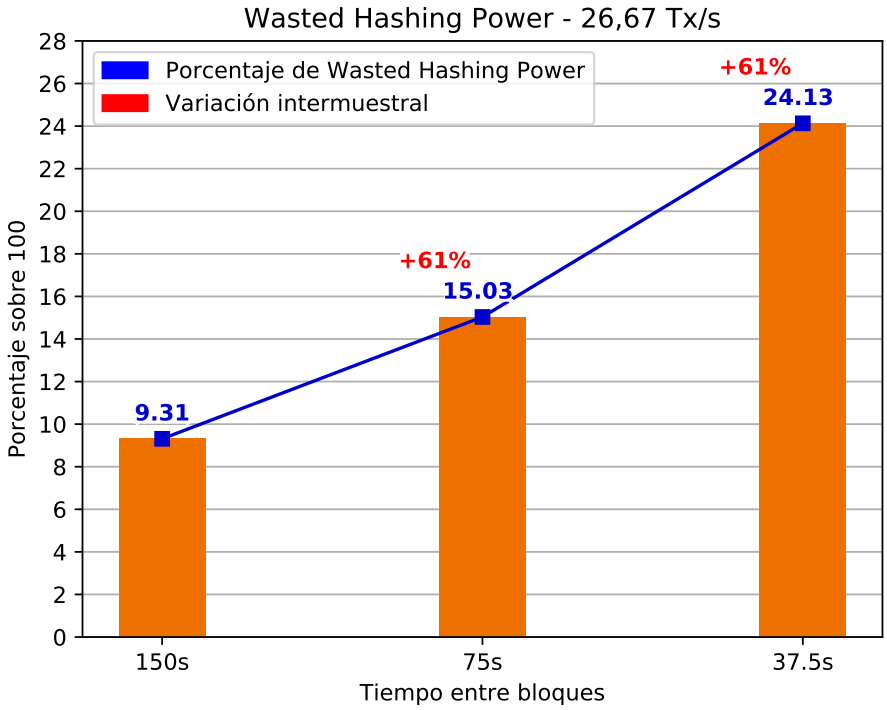
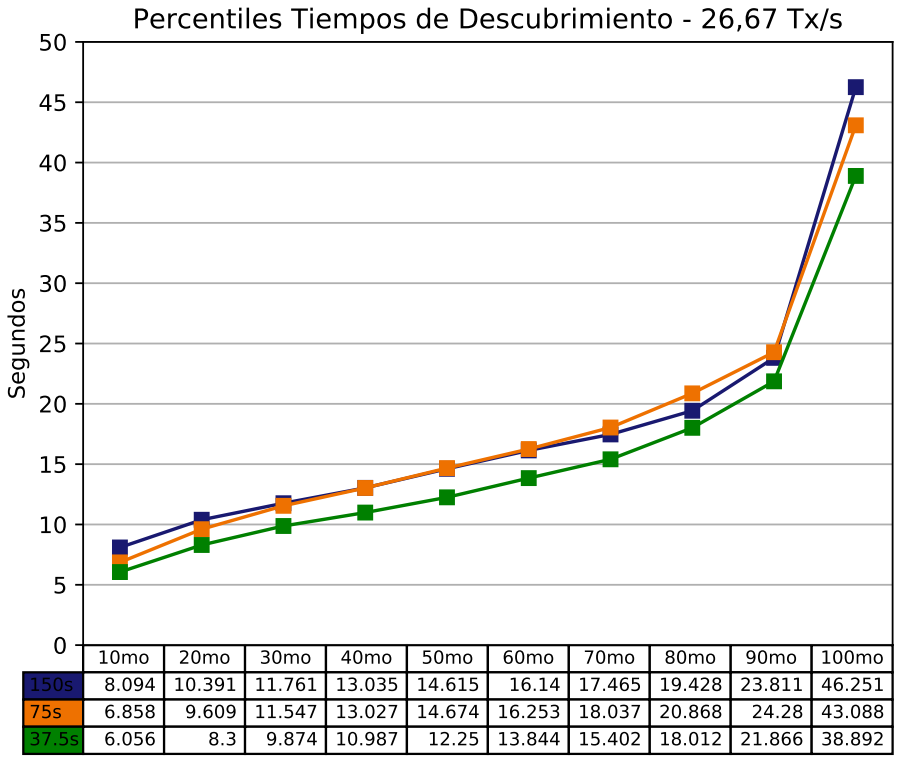
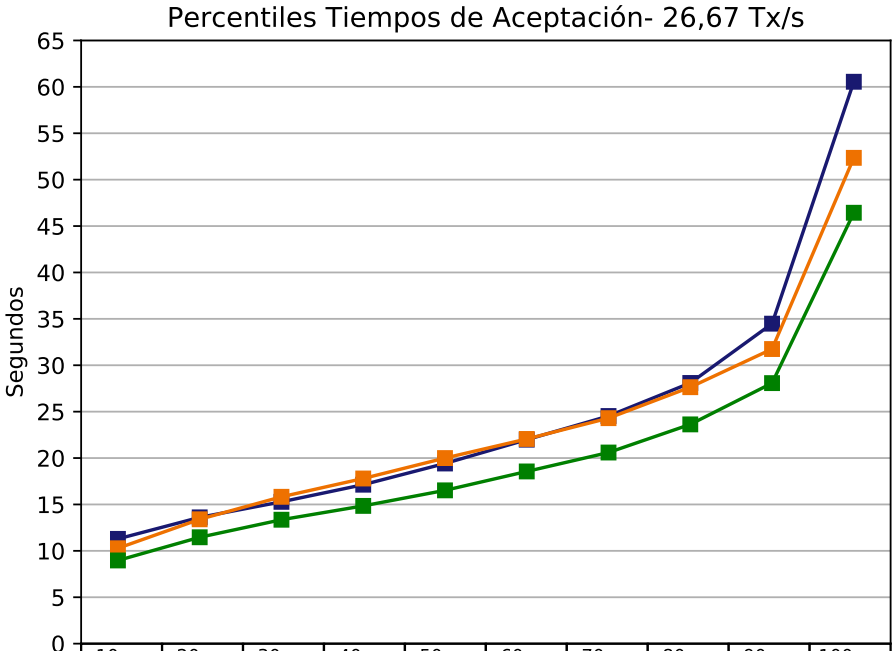


Figura 4.14: *Wasted hashing power* en experimentos con una carga de 26,67 Tx/s



	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
150s	8.094	10.391	11.761	13.035	14.615	16.14	17.465	19.428	23.811	46.251
75s	6.858	9.609	11.547	13.027	14.674	16.253	18.037	20.868	24.28	43.088
37.5s	6.056	8.3	9.874	10.987	12.25	13.844	15.402	18.012	21.866	38.892

Figura 4.15: Percentiles de tiempos de descubrimiento de bloques en experimentos con una carga de 26,67 Tx/s



	10mo	20mo	30mo	40mo	50mo	60mo	70mo	80mo	90mo	100mo
150s	11.289	13.609	15.282	17.129	19.414	21.98	24.529	28.09	34.481	60.549
75s	10.279	13.401	15.824	17.796	19.998	22.056	24.299	27.636	31.748	52.351
37.5s	8.965	11.463	13.354	14.844	16.519	18.556	20.595	23.627	28.075	46.432

Figura 4.16: Percentiles de tiempos de aceptación de bloques en experimentos con una carga de 26,67 Tx/s

Capítulo 5

Conclusiones y trabajo futuro

5.1 Conclusiones generales

En este trabajo se evaluaron distintas configuraciones del modelo actual de Bitcoin, cambiando parámetros del protocolo, en busca de aumentar el *throughput* de la red. Se optó por alterar el tiempo entre bloques, ya que este parámetro influye proporcionalmente sobre el máximo *throughput* alcanzable. Además, disminuir el tiempo entre bloques trae como beneficio la reducción de los tiempos de confirmación y potencialmente el aumento en la capacidad de escalar del sistema.

Modificamos el cliente de Bitcoin, incorporando mejoras en su eficiencia que lograron aumentar la cantidad de transacciones por segundo que el sistema puede soportar. Además, se agregó un sistema de *logging* que registra todos los datos necesarios para reconstruir los valores de las métricas definidas. Por último, logramos que las redes locales que genera el cliente posean las mismas características de la *mainnet*, generando así entornos experimentales adecuados para la realización de este trabajo. Este cliente modificado fue utilizado para la realización de todos los experimentos de la tesis. Esto nos permitió realizar experimentos en escenarios configurados con parámetros superiores a los soportados por el cliente original.

La red física y lógica utilizada se basa en un modelo que busca representar a la red real, en una escala reducida. Ésta posee características similares, revelando un comportamiento equivalente al real. Para poder construir este modelo se utilizaron datos de la distribución geográfica de los nodos de la red real de Bitcoin, las latencias entre los lugares donde se encuentran los clientes y los detalles de todos los *pools* de minería de Bitcoin. Se diseñó un algoritmo que, en base a esta información, recrea un modelo a escala de la red de Bitcoin real.

Las mediciones realizadas en una red Bitcoin, como *forks* o los tiempos de propagación, dan una noción de la calidad de la red, pero desde un punto de vista técnico. Se definió una nueva magnitud que permite cuantificar el impacto económico que producen las variaciones en la red. La misma captura el porcentaje de *hashing power* que no fue utilizado para la convergencia de la red. Utilizando esta medida, fue posible determinar qué proporción de una inversión económica es realmente utilizada y cuál es desperdiciada.

Los experimentos de este trabajo consisten en una red de Bitcoin de 240 clientes, distribuidos a lo largo de 37 países, los cuales están minando y enviando transacciones nuevas a sus *peers* en todo momento. En cada uno establecimos dos variables: el tiempo entre bloques esperado, y la cantidad promedio de transacciones incluida en cada bloque.

Actualmente, el protocolo de Bitcoin establece un tiempo entre bloques esperado de 600 segundos, y usualmente cada bloque contiene en promedio 2000 transacciones. Hicimos un experimento usando esta configuración, y también reduciendo el tiempo entre bloques esperado a la mitad, un cuarto, un octavo y un dieciseisavo. Además de utilizar el valor promedio normal de 2000 transacciones en cada bloque, hicimos experimentos aumentando la carga del sistema a 4000 transacciones por bloque promedio, y disminuyéndola a 1000.

Los resultados obtenidos fueron clasificados según el *throughput* alcanzado para cada escenario con una carga del sistema y tiempo entre bloques dados.

En los experimentos donde el *throughput* alcanzó 3,33 Tx/s, es decir, el *throughput* normal con el que la red de Bitcoin opera actualmente, todos los valores observados fueron del

mismo orden de magnitud que los registrados en la *mainnet*. Estos resultados sugieren que es posible utilizar tiempos entre bloques de tanto 600, 300 como 150 segundos y mantener el buen funcionamiento de la red bajo el *throughput* mencionado. No obstante, reducir el tiempo entre bloques disminuye los tiempos de confirmación de las transacciones, ya que en promedio hay que esperar menos para la aparición de futuros bloques. Sin embargo, también incrementa el *wasted hashing power*, elevando así el costo económico de mantener la red en funcionamiento. El hecho de si conviene pagar más para tener tiempos de confirmación menores es una decisión de negocios, ya que reducir el tiempo de confirmación resulta en una mejor experiencia de usuario.

Cuando se aumenta la carga a 6,67 Tx/s, y operando con un tiempo entre bloques de 600 segundos, el sistema se acerca al límite del buen funcionamiento de la red con respecto a los tiempos de confirmación. Los datos arrojados bajo esta carga sugieren que es conveniente configurar la red para que el tiempo entre bloques esperado sea de 300 segundos, diferente al de la red actual de Bitcoin. Al utilizar esta configuración, y bajo la carga mencionada, el costo económico de mantener el sistema en funcionamiento es similar al observado cuando la red tiene un tiempo entre bloques esperado de 600 segundos. Sin embargo, en el experimento de 600 segundos, cada bloque incluía en promedio 4000 transacciones, lo que significa que no se podría aumentar mucho la carga sin caer en un escenario de saturación. En cambio, con un tiempo entre bloques de 300 segundos, se incluyen en promedio 2000 transacciones por bloque, que es un valor lejano al punto de saturación del sistema. Esto significa que con un tiempo entre bloques de 300 segundos, se podría duplicar la carga, manteniendo a su vez el buen funcionamiento del sistema y llegando así a un valor de *throughput* mayor que el límite teórico cuando se usa 600 segundos de tiempo entre bloques.

Utilizando los parámetros actuales de la *mainnet*, el límite teórico del *throughput* que el sistema puede alcanzar es 12,47 Tx/s. Los resultados sobre escenarios funcionando bajo condiciones que no son alcanzables usando la configuración actual del protocolo de la *mainnet* sugieren que no resulta viable mantener a Bitcoin funcionando bajo una carga superior a 26,67 Tx/s, ya que la calidad de la red sufre un gran deterioro.

Los valores obtenidos en los experimentos con una carga de 13,33 Tx/s también son elevadas, aunque quizás tolerables en el escenario de 300 segundos de tiempo entre bloques. Sin embargo, otros factores ponen en duda que la red pueda funcionar procesando esa cantidad de transacciones por segundo. En el experimento con tiempo entre bloques de 300 segundos, en promedio hay 4000 transacciones en cada bloque, resultando en un deterioro del tiempo de confirmación de las mismas. Por otro lado, los experimentos de 150 y 75 segundos de tiempo entre bloques presentan signos de deterioro no despreciables. Ambos registraron *forks* de profundidad 2, y en los dos el *wasted hashing power* fue de más de diez veces superior al observado en el experimento realizado con los parámetros y carga normal de la *mainnet*. Por lo tanto, si bien no es imposible mantener el sistema funcionando con una carga de 13,33 Tx/s, podemos afirmar que en dicho caso estaríamos cerca del límite del máximo *throughput* alcanzable mediante la modificación de los parámetros de la red.

Adicionalmente, los resultados obtenidos en experimentos con tiempos entre bloques bajos sugieren que existe un límite inferior para este valor, sin que el sistema sufra un gran deterioro de la calidad de la red y sin que se genere un incremento significativo en el costo económico de mantenerlo en funcionamiento.

Bajo algunas configuraciones de tiempo entre bloques probamos el sistema procesando 6000 transacciones por bloque promedio, que es poco más del 80 % del máximo posible bajo los tamaños de bloque permitidos actualmente. Este escenario nos permitió analizar el comportamiento de Bitcoin operando con un *throughput* cercano a su cota teórica. Los resultados obtenidos mostraron que los tiempos de confirmación crecen significativamente, resultando en un gran deterioro de la experiencia de usuario.

Adicionalmente, demostramos que los tiempos de confirmación de las transacciones aumentan de manera exponencial a medida que se incrementa la carga del sistema, indicando que Bitcoin necesariamente debe operar con una carga reducida para alcanzar una buena experiencia de usuario.

Bitcoin es un sistema distribuido, y como tal, está sujeto a reglas de funcionamiento basa-

das en la propagación de información. Dichas reglas establecen límites en cómo se puede configurar el protocolo de comunicación. Por ejemplo, el tiempo entre bloques no podría nunca ser de 1 segundo.

Los parámetros del protocolo generan límites teóricos, los cuales, en la práctica, el sistema real nunca puede alcanzar. Por ejemplo, utilizar los parámetros de la *mainnet* determina que el sistema nunca podría superar un *throughput* máximo de 12,47 Tx/s. Sin embargo, ya cuando este es superior a aproximadamente el 60 % del límite, se observa un crecimiento significativo en los tiempos de propagación que vuelve el sistema inusable. Esto es válido para cualquier valor de tiempo entre bloques que pudiera ser configurado.

En cualquier criptomoneda, la falta de una tercera parte de confianza implica que se deben realizar verificaciones adicionales para garantizar la seguridad y la integridad del sistema. Esto redunda en un incremento en la complejidad computacional de los algoritmos involucrados.

Esta tesis deja de manifiesto que todas las limitaciones mencionadas anteriormente tienen como consecuencia que Bitcoin no puede acercarse al rendimiento de sistemas de pago tradicionales a menos que resigne en algún grado su carácter descentralizado.

5.2 Trabajo futuro

En esta sección se proponen algunas líneas de investigación que pueden servir como continuación a este trabajo. Las mismas están enfocadas a seguir desarrollando el modelo presentado y a analizar el funcionamiento de Bitcoin bajo escenarios que quedaron fuera del alcance de la tesis.

Redes de retransmisión

El modelo de red propuesto en este trabajo contempló únicamente nodos Bitcoin que operan en la red P2P. En la sección 1.2.5 mencionamos que las llamadas *redes de retransmisión* forman parte de una estrategia para reducir los tiempos de transmisión que se observan en

el sistema. Las mismas modifican el camino de los mensajes en algunos casos, resultando en patrones de comunicación diferentes al sistema original. El modelo de red presentado podría ser extendido, agregando al modelo una red de retransmisión que modifique la forma en que los clientes se conectan, y de esta manera realizar experimentos para analizar cómo se ven alterados los tiempos de propagación cuando se suma al sistema una red de ese tipo.

Adicionalmente, resulta interesante estudiar escenarios en los que se busque determinar el nivel de centralización que tiene el sistema al funcionar en conjunto a una red de retransmisión.

Tamaño del modelo

Si bien el modelo propuesto permite estudiar una red Bitcoin a escala, una pregunta que surge de su definición es cuál es la mínima cantidad de clientes que se debe utilizar, de modo que la red emulada presente un comportamiento similar al sistema real. Este estudio se podría llevar a cabo experimentando con escenarios de menor tamaño, utilizando en ellos el protocolo de la *mainnet*. Los resultados de este análisis ayudarían a determinar la cantidad mínima de hardware requerido para poder aplicar exitosamente la metodología presentada.

Otro posible trabajo consiste en estudiar cómo se comportan las métricas de un escenario a medida que se incrementa la cantidad de nodos contenidos en él. Este análisis ayudaría a determinar cuán redituable resulta agregar recursos de hardware a la plataforma experimental para evaluar escenarios de mayor tamaño.

Comparación del rendimiento económico de diferentes sistemas

En el presente trabajo definimos la métrica *wasted hashing power*, la cual apunta a medir el rendimiento económico del sistema abstrayéndose de las características del protocolo. Es posible definir esta métrica para diferentes criptomonedas, generando así una metodología para hacer comparaciones entre ellas. La misma podría ser utilizada incluso para evaluar nuevos sistemas basados en *blockchain*.

Condiciones de red variables

En el modelo presentado, las latencias están modeladas de manera estática, es decir, durante el transcurso de un experimento, la propagación de información entre dos clientes dados tarda lo mismo en todo momento. Sin embargo, en una red real, la latencia entre dos nodos varía con el tiempo. El modelo podría extenderse de modo que la latencia y otras características de los enlaces tomen valores dinámicos que cambian en función del tiempo.

Utilizando esta nueva funcionalidad, podrían plantearse dos escenarios con condiciones de red irregulares. El primero consiste en que haya breves momentos en los que la latencia de ciertos enlaces se incremente significativamente, mientras que el segundo consta de períodos en los que el enlace no está disponible, pudiendo generar así particiones en la red.

Experimentar con el cliente funcionando en estos escenarios daría una noción del comportamiento del sistema bajo condiciones de red desfavorables.

Bibliografía

[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.

[Cor] Matt Corallo. Bip 152 - compact blocks. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki. Accessed: 3 de octubre de 2019.

[CRS+] Andrew Clifford, Peter R. Rizun, Andrea Suisani, Andrew Stone, and Peter Tschipper. Towards massive on-chain scaling: Presenting our block propagation results with xthin. https://medium.com/@peter_r/towards-massive-with-xthin-da54e55dc0e4#.kk1znlg1v. 3 de octubre de 2019.

[DPSHJ14] Joan Antoni 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.

[DWC+17] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains.

In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1085–1100, New York, NY, USA, 2017. ACM.

[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.

[GKL16] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. Cryptology ePrint Archive, Report 2016/1048, 2016. https://eprint.iacr.org/2016/1048.

[GM18] Maximiliano Geier and Esteban Mocskos. Sherlockfog: Finding opportunities for mpi applications in fog and edge computing. In Esteban Mocskos and Sergio Nesmachnow, editors, *High Performance Computing*, pages 185–199, Cham, 2018. Springer International Publishing.

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

[MJ15] Andrew Miller and Rob Jansen. Shadow-bitcoin: Scalable simulation via direct execution of multi-threaded applications. In *Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test*, CSET'15, pages 7–7, Berkeley, CA, USA, 2015. USENIX Association.

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

[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.

[Van16] Marco Vanotti. Un avance hacia entornos de gran escala para experimentos con criptomonedas. Master's thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2016.

[Vn17] Silvio Vileriño. Estudio de los límites de generación de bloques en blockchain. Master's thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2017.