UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Bitcoin: todos se benefician por igual, pero... ¿Hay algunos más iguales que otros?

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

Marcos Kevin Piotrkowski

Director de tesis: Dr. David González Márquez
Codirector de tesis: Dr. Maximiliano Geier

Fecha y lugar de defensa
xx de xxxx de 2019
Buenos Aires, Argentina

Firma

Bitcoin: todos se benefician por igual, pero... ¿Hay algunos más iguales que otros?

Resumen

La aparición y el auge de Bitcoin plantean nuevos posibilidades sobre la forma tradicional de hacer transacciones, así como también nuevos desafós a resolver. Para atacar los mismos es indispensable contar con herramientas que nos permitan evaluar aspectos de la red frente a determinados escenarios y también el impacto que puedan traer modificaciones al protocolo.

Desde distintos grupos académicos y de la industria del software se está trabajando en herramientas que permitan evaluar el comportamiento de la red bajo diferentes condiciones. Luego de relevar las mismas entendimos que aún hay espacio para el desarrollo de aquellas que nos permitan analizar ciertos comportamientos de la red que solo son observables al considerar escalas de la red comparables a la real con decenas de miles de nodos.

En este trabajo presentamos una herramienta para simular la red Bitcoin utilizando el framework provisto por SimGrid, un simulador de eventos discretos.

El foco está puesto en los aspectos que permiten llegar a un consenso distribuido sobre el historial de transacciones que ocurrieron en la red.

Vamos a tener dos tipos de actores: nodos y mineros. Los primeros encargados de generar transacciones y los segundos un tipo especial de nodos que además son capaces de generar bloques. Las transacciones incluidas en dichos bloques serán consideran como confirmadas.

Todos los actores van a seguir el comportamiento del cliente de referencia de Bitcoin para transmitir, recibir y validar los mensajes en la red simulada.

La herramienta provista permite analizar como se comporta la red ante una traza de actividad determinada así como el comportamiento ante cambios de parámetros en el protocolo de consenso o de red, y en particular será utilizada en nuestro trabajo para identificar ventajas que puedan obtener los actores mediante una mejora en su conectividad o en el poder de cómputo disponible.

Quiero agradecer al LICAR por brindarme un espacio que me permitió avanzar en la tesis, del que pude conocer los entretelones de la investigación científica y del que me llevo una experiencia que excede el alcance de este trabajo.
Al DC por organizar las jornadas de investigación para estudiantes de computación donde pude conocer en que trabaja cada área del departamento y me sirvió para contactar a los que luego serían mis directores de tesis.
Pero, principalmente, a Esteban, David y Maxi por todo lo que hicieron por el departamento en general y, en particular, por ayudar a tesistas como yo a dar un cierre que esté al nivel de lo que se espera de esta carrera. Su apoyo fue fundamental para poder investigar sobre un tema que me apasiona sin sentirlo como una obligación.

Índice general

1

Introducción

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

Bitcoin ofrece distintas funcionalidades, entre las que podemos mencionar:

- **Peer discovery**: para encontrar otros nodos de Bitcoin disponibles a los cuales consultar sobre el estado de la Blockchain y a los cuales comunicar las transacciones generadas por el nodo local.

- **Message signature**: para firmar mensajes con la clave privada de una dirección de Bitcoin con el fin de probar a otros miembros que uno es el dueño de dicha cuenta.

- **Protocolo de consenso de Blockchain**: esta tecnología representa el mayor aporte innovador de Bitcoin. Este protocolo permite llegar al consenso entre pares sin necesidad de confiar en una entidad centralizada.

Desde el nacimiento de esta tecnología, ha habido importantes esfuerzos por extender su utilización a diversos ámbitos más allá de los relacionados a la transferencia monetaria. Entre los que nos parece interesante mencionar encontramos: el registro distribuido de DNS, la creación de organizaciones autónomas, el registro de eventos en la Blockchain como prueba de la existencia de un suceso, entre otros.

Hay un gran interés por parte de la comunidad de criptomonedas por mejorar la tecnología de transferencias para competir con actores tradicionales como tarjetas de crédito o bancos, así como de encontrar nuevos problemas que puedan ser resueltos mediante la tecnología de Blockchain. Una gran parte estas iniciativas se verían fuertemente beneficiadas si se pudiera evaluar su comportamiento en una escala real, es decir, contar con una forma de analizar cómo interactúan miles de agentes.

Este trabajo de tesis busca proveer una herramienta para simular una red completa de Bitcoin basándose en el comportamiento del cliente de referencia. Este cliente de referencia está basado en la versión original publicada por Nakamoto [Nak08] y es mantenido por una gran comunidad de desarrolladores, algunos de los cuales participan activamente desde el nacimiento de esta tecnología.

En este trabajo buscamos enfocarnos principalmente en el protocolo de consenso. El objetivo principal es generar una herramienta que permita analizar cómo se comporta la red ante una secuencia de eventos determinada así como el comportamiento ante cambios de parámetros en el protocolo de consenso o de red, por ejemplo: cantidad de *peers* por default, latencia entre nodos, tamaño máximo de un bloque, tiempo promedio de generación de los mismos, tiempo de validación de transacciones y bloques, etc.

1.1 Glosario de términos

A continuación definimos una serie de términos que serán utilizados a lo largo de este trabajo.

- **address** (dirección): un hash de 20 B para que los usuarios puedan intercambiar información de pago.
- **tx**: abreviación de transacción.
- **Bloque**: es una estructura compuesta por un encabezado, una lista de una o más transacciones y protegida por un mecanismo de **prueba de trabajo**.
- **Proof of Work** (prueba de trabajo): el protocolo de Bitcoin utiliza un mecanismo que asegura que estadísticamente los mineros hayan tenido que realizar una cierta cantidad de trabajo para lograr incorporar un bloque a la Blockchain.
- **Coinbase tx**: es la primera transacción en un bloque y siempre es creada por un minero. Le permite a éste obtener la recompensa otorgada por la red por haber encontrado un bloque válido.
- **Nodo**: proceso que sigue el protocolo de la red de Bitcoin y está conectado a través de dicha red con otros nodos. Puede generar transacciones, informarlas a sus *peers*, y validar las transacciones y bloques que recibe de ellos.
- **Minero**: es un nodo especializado que además de poder comportarse como este también puede generar bloques y comunicarlo a sus *peers*.
- **Mainnet**: red de producción del sistema Bitcoin.
- **Testnet**: red Bitcoin destinada al desarrollo de nuevas funcionalidades de clientes y todas las pruebas de desarrollo de la comunidad, se rige por otras reglas que la *Mainnet* y tiene una escala mucho menor.
- **Reference Implementation**: se refiere al cliente `bitcoind`, que es la implementación más usada en la red Bitcoin.
- **Blockchain**: registro público replicado y descentralizado de todas las transacciones confirmadas, representado en una estructura de árbol que contiene todos los bloques válidos (incluyendo huérfanos).
- **Dificultad**: medida de cuan difícil es encontrar un nuevo bloque. Hay un valor mínimo de cuan fácil puede ser y la dificultad esperada para cada bloque se reajusta periódicamente para que estadísticamente aparezcan cada diez minutos. Los mineros deben incluir en sus bloques una **prueba de trabajo** que asegure que estos bloques cumplen con la dificultad esperada por la red.
- **Main Chain**: cadena de bloques de la Blockchain que un nodo considera como aquella con la mayor dificultad.
- **Fork**: refiere a una bifurcación en la Blockchain, cuando dos partes de la red ven *Main Chain* diferentes. Será descripto más ampliamente en la sección 1.2.2 (Proceso de minado y consenso).
- **Bloque huérfano**: un bloque válido que no forma parte de la *Main Chain*.
- **Mempool** representa la colección de transacciones aún no confirmadas que son almacenadas por los nodos hasta que expiran o son incluídas en la *Main Chain*. Por default, el cliente de referencia considera como expiradas las transacciones después de 24 horas y las elimina del *mempool*.

- **Reorganización**: ante una situación de *forks*, dos o más partes de la red considerarán *Main Chains* diferentes. Eventualmente, la partición de la red con el mayor *hash power* conseguirá una cadena con mayor dificultad, la cual será adoptada por el resto de la red que deberá reorganizar los bloques que consideraba válidos y las transacciones que consideraba confirmadas. Los bloques que no formen parte de la nueva *Main Chain* pasará a ser considerados *bloques huérfanos* y las transacciones de dichos bloques serán consideradas inválidas (por ejemplo si referencian fondos ya utilizados por otras transacciones de la *Main Chain*) o serán movidas al *mempool*.

- **Transaction Fee** (comisión por transacción): es un monto en bitcoins que el autor de una transacción paga al minero que la incluya en un bloque. El minero puede basar la elección de qué transacciones agregar a un bloque en obtener la más alta comisión, con lo que una transacción con baja comisión corre el riesgo de tener un tiempo de confirmación muy largo.

1.2 Breve introducción a Bitcoin

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

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

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

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

En la figura 1.1 se presenta una esquematización de este proceso en donde se realizan tres transacciones de una criptomoneda. Se puede ver que el dueño de la primera clave privada firma el *hash* que hace referencia a la transacción anterior y a la clave pública del nuevo destinatario. De esta manera crea una nueva transacción. Este proceso se repite una vez más entre el nuevo destinatario y una tercera persona.

Este procedimiento deja lugar a un problema conocido como *double spending*: el emisor podría crear todas las transacciones que quisiera a partir de una misma moneda, sin embargo para que el sistema funcione, una sola de estas transacciones debería ser válida. Una solución a este problema (y la tradicional adoptada en el sistema financiero hace mucho tiempo) sería contar con una entidad de confianza a la que se encargue la función de validar todas las transacciones realizadas. De esta manera, esta entidad centralizada podría verificar todas las transacciones recibidas y detectar aquellos casos en los que el dinero intenta ser utilizado (i.e. gastado) más de una vez.

Bitcoin propone una solución diferente al problema de *double spending* y no requiere tener que confiar en una tercera parte. La idea principal es que todos los nodos de la red lleven un registro de todas las transacciones que fueron procesadas y que cualquier nodo pueda publicar transacciones para ser incluidas en dicho registro.

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

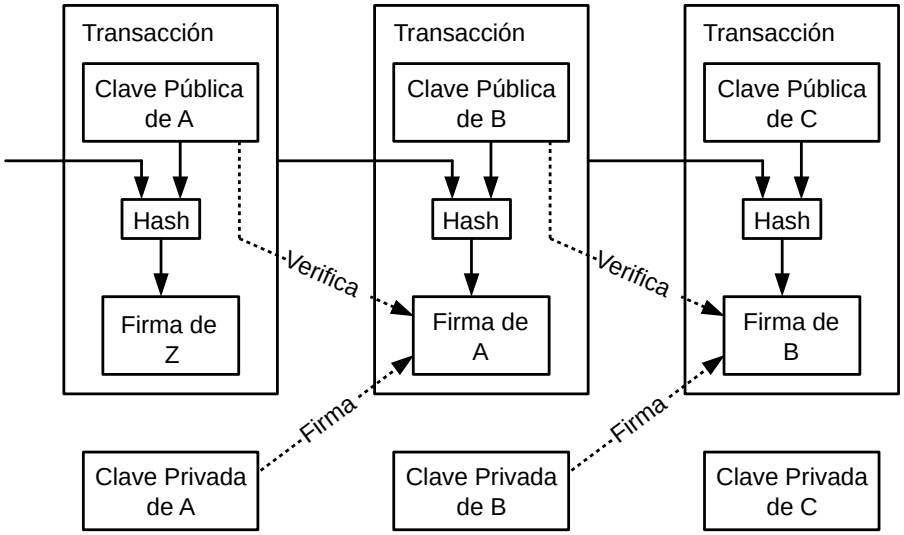


Figura 1.1: *Esquema de transacciones en una criptomoneda. Fuente: [Nak08]*

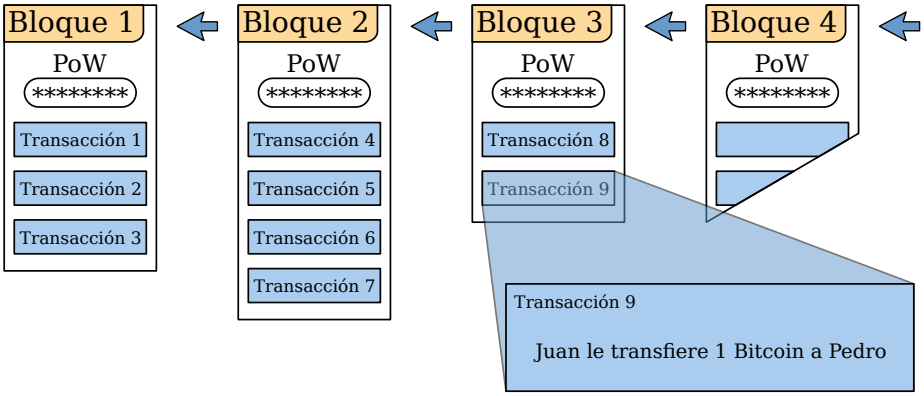


Figura 1.2: *Esquema de blockchain con transacciones.*

1.2.1 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 las ideas detrás de ellas.

Transacciones

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

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

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

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

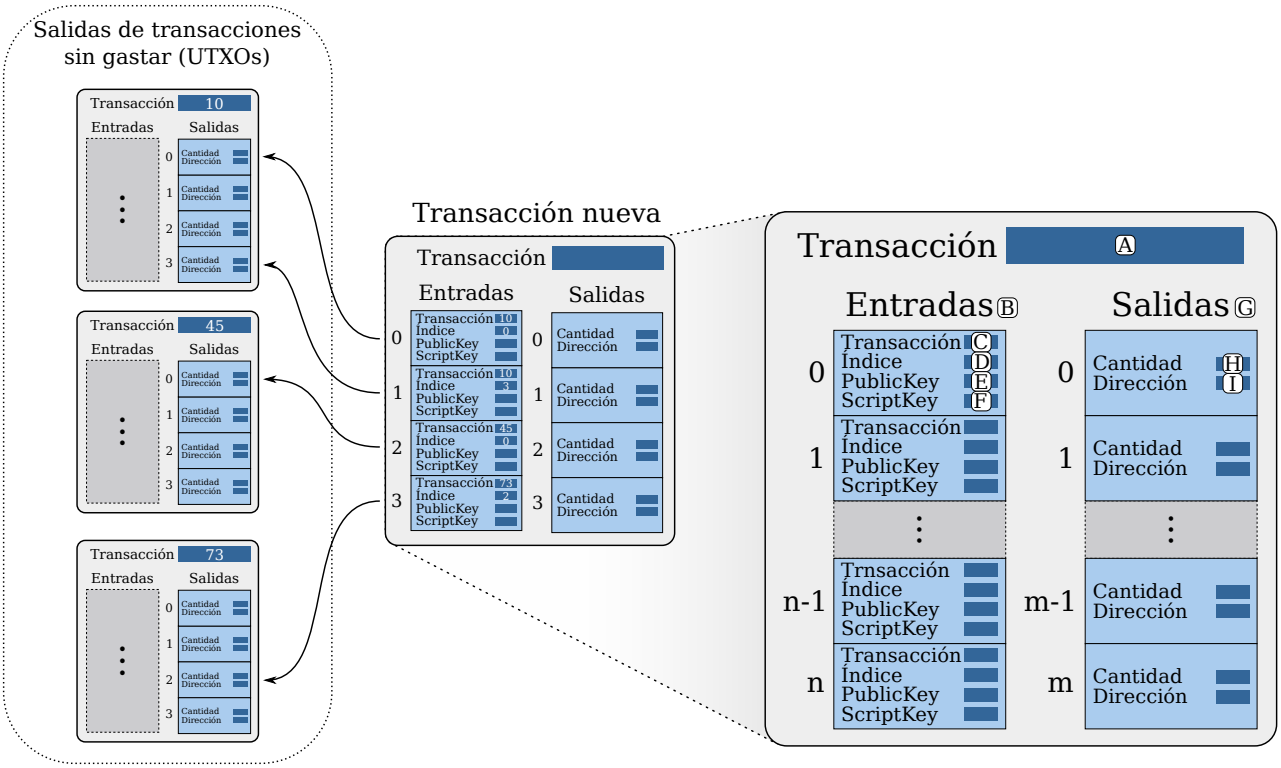


Figura 1.3: *Ejemplo de una transacción en Bitcoin, sus* inputs *hacen referencias a* outputs *de transacciones anteriores.*

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

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

B) La lista de *inputs* de la transacción y por cada uno de ellos:

   C) El *hash* de la transacción a la que ese *input* hace referencia.

   D) El índice del *output* dentro de la lista de *outputs* de la transacción referenciada (tiene que ser uno que no haya sido utilizado previamente en otra transacción).

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

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

G) La lista de *outputs* de la transacción y por cada uno de ellos:

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

   I) La dirección (*address*) destino determinada por 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.

Estos scripts están limitados en su poder expresividad en el sentido en que no permiten ciclos. Dicha limitación es para reducir la complejidad posible de dichos scripts considerando que los mismos deben ser ejecutados por todos los nodos de la red con el fin de validar las transacciones. De otra forma podría haber scripts con ciclos infinitos o que requiriesen un gran poder de cómputo para ser validados, lo cual sería un vector de ataque que podría ser abusado por actores maliciosos. Otras criptomonedas que sí permiten scripts con ciclos deben incluír otras limitaciones para prevenir este vector de ataque.

Para verificar que una transacción sea válida, se debe verificar que todos los *inputs* son válidos, y que la suma de los *outputs* referenciados por esos *inputs* es mayor o igual a la suma de los *outputs* de la transacción. Si la suma de los *inputs* es mayor a la de los *outputs*, el excedente podrá ser reclamado por quien agregue la transacción a la Blockchain.

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

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

Bloques

Cada bloque está compuesto por:

- El *hash* del bloque anterior (su padre).
- Tiempo en segundos en que fue minado, contados a partir del *epoch*[1].
- La dificultad con la que ese bloque fue hallado.
- `nonce`: La prueba de trabajo, que demuestra que el bloque cumple con la dificultad que dice tener.
- Un conjunto de transacciones ordenadas.
- Un número de versión.

No todos los bloques son válidos. Para que un bloque sea válido y aceptado por la red, debe contar con lo que se conoce como *Proof of Work* (prueba de trabajo).

Se denomina *proof of work* a un tipo de mecanismo particular por el cual se trata de disuadir ataques de denegación de servicios u otros abusos a un servicio como el envío de *spam*. Dicha prueba de trabajo está pensada para ser resuelta por una máquina y una propiedad clave con la que debe contar un mecanismo de este tipo es su asimetría con respecto a la dificultad: debe ser computacionalmente costoso para una máquina generar la prueba de trabajo esperada pero a su vez debe ser fácil para terceros validarla.

Para el caso de Bitcoin, los mineros deberán incluir dicha prueba de trabajo en un campo especial de sus bloques lo cual será abordado más adelante en la sección de proceso de minado.

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

Un *Merkle Tree* es un árbol binario de hashes criptográficos, en donde cada nodo contiene el hash de la concatenación de sus dos hijos. En la figura 1.4 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.

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

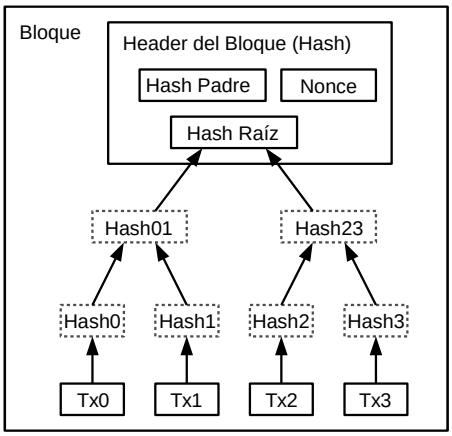


Figura 1.4: *Ejemplo de Bloque con Merkle Tree. Fuente: [Nak08]*

Blockchain

La estructura de la Blockchain puede verse como un árbol de bloques. Se diferencia de una lista pues puede suceder que dos bloques tengan el mismo padre, dando lugar a ramificaciones.

La raíz de la Blockchain es un bloque especial llamado *Bloque Génesis*. En Bitcoin, este bloque fue minado el 3 de enero de 2009, y además, en uno de sus campos contenía un titular del diario «*The Times*» de esa fecha[2], para dar prueba que este bloque no fue creado con gran antelación a ese día.

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

Cualquier nodo de la red puede construir bloques, denominándose **nodo minero**. Al recibir un bloque, los nodos lo validan y lo agregan a su Blockchain. Es posible que los nodos cambien la cadena que considera como principal si el nuevo bloque pertenece a una cadena previamente *stale*, pero que pasa a representar mayor dificultad total que la cadena principal. Para que un bloque sea válido y aceptado por los nodos de la red, debe contener una prueba de trabajo, que valide que el nodo minero hizo uso de CPU para poder construir el bloque.

1.2.2 Proceso de minado y consenso

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

El campo `nonce` del *header* del bloque contiene la *Proof of Work (PoW)*. Su contenido es un número tal que **SHA256(SHA256(header))**[3] represente un valor menor al requerido por la dificultad con la que opere la red en dicho momento, expresada con número con precisión de 256 bit. De manera simplificada, se puede explicar que los mineros van modificando el `nonce` hasta encontrar uno para el cual su hash cumpla con el valor esperado y mientras mayor sea la cantidad de 0s con la que empiece mayor será la probabilidad de que quede por debajo del umbral de corte y el bloque sea considerado válido.

[2] «The Times 03/Jan/2009 Chancellor on brink of second bailout for banks»

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

Podemos explicar el costo computacional si notamos que los hashes están uniformemente distribuidos y la probabilidad de que un hash (256 bit) comience con d ceros, es $\frac{2^{256-d}-1}{2^{256}}$. Además, el hecho de modificar incluso un único bit del *input* implica cambiar completamente el hash calculado, lo cual fuerza a probar, mediante fuerza bruta, distintas entradas hasta encontrar una cuyo hash resultante cumpla con la dificultad esperada.

Para verificar que un bloque cumple con esta propiedad, alcanza con calcular dos veces el `SHA256` del *header*, mientras que la dificultad de armar un bloque que sea válido crece exponencialmente con la cantidad de 0's requerida[4]. De esta forma, se cuenta con un proceso que demuestra fácilmente que un minero hizo una cantidad razonable de trabajo para hallar un bloque.

Es importante destacar que una *PoW* solo sirve para un bloque, ya que 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 anticipación.

Debido a que el poder cómputo de la red puede fluctuar mucho con el ingreso y egreso de nodos que participan, Bitcoin incorpora un sistema de **ajuste de dificultad**, que aumenta o disminuye la dificultad que deben cumplir los bloques para ser válidos. Para esto, la red define un *target*, que es el tiempo promedio en que deben aparecer nuevos bloques en la red (en Bitcoin es diez minutos). Cada 2016 bloques, se cuenta cuántos de esos bloques[5] se construyeron en menos tiempo que el target, y en función de eso se aumenta o disminuye la dificultad hasta cuatro veces el valor anterior. Esto se traduce en que el hash del *header* del bloque debe comenzar con hasta dos ceros más o menos, según aumente o disminuya la dificultad, respectivamente.

Bitcoin provee ciertos incentivos para que los mineros colaboren con la red. Por un lado, el minero que construye un bloque se queda con todos las *fees* de las transacciones de ese bloque. Por otro lado, Bitcoin recompensa a los mineros, permitiéndoles ser los beneficiarios de la transacción *coinbase* del bloque, una transacción especial que otorga nuevos bitcoins, o sea no referenciados en el conjunto de *outputs* no gastados. Esta es la única forma de crear bitcoins, y es por eso que se conoce al proceso como «minar bitcoins», pues el minero en realidad está haciendo aparecer nuevos bitcoins.

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

La cantidad de bitcoins otorgada por minar bloques disminuye en función del número de bloque. Cada 210 mil bloques (cuatro años aproximadamente) se disminuye a la mitad. Comenzando con un valor de 50 BTC por bloque. Esto da un total de aproximadamente 21 millones de bitcoins. Al 2 de noviembre de 2019, la recompensa por hallar un bloque en Bitcoin es de $12{,}5$ BTC.

Cuando un minero encuentra un *nonce* que hace válido al bloque, propaga dicho bloque a sus nodos vecinos, éstos lo validan, agregan a la Blockchain y continúan su propagación[6].

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

En esta situación, los nodos de la red que reciben un bloque competitivo al bloque sobre el que están minando, tienen que decidir si ajustan o no su cadena principal. Esto es lo que se conoce como **Protocolo de Consenso**. En Bitcoin, ante dos ramas competitivas, gana la que tiene mayor dificultad total (suma de las

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

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

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

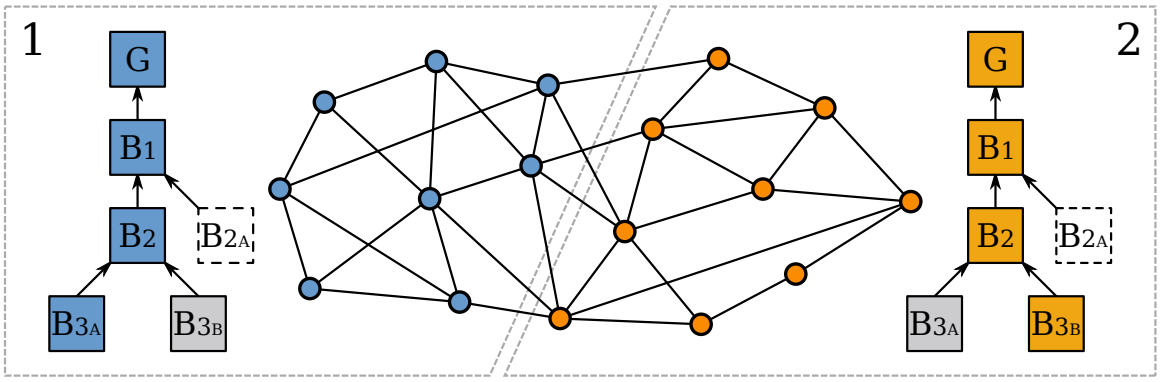


Figura 1.5: *Ejemplo de una partición de los nodos de la red. El grupo de nodos 1 considera al bloque B_{3A} como bloque de su cadena principal, mientras que el grupo 2 considera a B_{3B}. El bloque B_{2A} quedó* stale *luego de que se minaron los bloques B_{3A} y B_{3B}.*

dificultades de los bloques), y ante un empate, cada nodo se queda con la que recibió primero (*first-seen rule*).

Esta situación deriva en una partición en los nodos de la red, como se puede observar en la 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 más dificultad y por lo tanto será la aceptada para toda la red.

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

Esto indica que, para tener cierta garantía de no ser víctimas de un ataque de *double spending*, simplemente se debe esperar a que haya más bloques minados sobre el bloque en donde aparece la transacción de interés. Si un atacante quisiese revertir esa transacción, debería minar un bloque cuyo padre sea el mismo que el del bloque que la contiene y, además, crear tantos bloques como bloques hayan sido minados a continuación. La posibilidad práctica de revertir una transacción decrece con cada bloque que aparece por el esfuerzo computacional que implicaría recrear una cadena paralela más larga mientras el resto de la red sigue incrementando la longitud de la *Main Chain* actual.

Bitcoin tiene como premisa que más de la mitad de los nodos actúan de manera honesta, pero si esto no ocurriese y un atacante contase con mayor poder de cómputo que el resto de la red, podría efectivamente deshacer tantos bloques de la *Main Chain* actual como quisiese y reemplazarlos por otros propios. Un escenario de estas características es conocido como un **ataque del** $51\,\%$ e implicaría una situación ante la cual la red no podría recuperarse, donde los usuarios no tendrían garantías de la validez de las transacciones y que probablemente implicaría una reducción parcial o total del valor económico de esta criptomoneda.

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

1.2.3 Protocolo

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

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

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

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

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

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

En el cliente de referencia de Bitcoin los nodos tratan de establecer conexiones con ocho *peers* activos, lo que se conoce como conexiones salientes. Cuando otros *peers* son los que intentan establecer una conexión con el nodo actual se denomina conexión entrante. El cliente permite en total 125 conexiones entre las de tipo entrantes y salientes.

1.3 Descripción de los aspectos a simular

Luego de haber hecho una breve introducción a Bitcoin presentaremos los aspectos del sistema que fueron modelados.

Dado que la novedad de Bitcoin es la prevención del *double spending* en un sistema descentralizado, enfocamos el desarrollo de la herramienta en poder cubrir los aspectos del protocolo que hacen esto posible. En consecuencia, otros aspectos del cliente de referencia como *peer discovery*, que permite a un nodo encontrar a otros activos al momento de unirse a la red, no van a ser modelados. En particular vamos a suponer una configuración estática de la red donde los nodos participantes sean los mismos durante la duración de una simulación.

1.3.1 Estructuras de datos

En esta sección se describirán en detalle las estructuras de datos utilizadas en la simulación. Como se describió previamente, los mensajes principales del protocolo de red son **GetAddr, Addr, Inv, GetData, GetHeaders, GetBlock, Block, Headers, Tx, Version y VerAck**. Para cubrir los aspectos del protocolo que hacen posible la prevención del *double spending* verificamos que los mensajes relevantes a modelar son **Inv, GetData, Block, Tx**.

En nuestra simulación, un **Mensaje** es una estructura básica que, luego, es extendida por cualquiera de las otras para especializar su comportamiento. Un mensaje incluye:

- Un identificador para el mensaje.
- El tamaño en bytes.

El identificador es importante dado todas las transacciones y bloques tienen uno y, sin éste, un nodo no podría determinar si ya lo recibió previamente o tiene que pedirlo a sus *peers*.

El tamaño (en bytes) se provee a las primitivas de envío de mensajes de SimGrid para poder calcular el tiempo de comunicación considerando, además, el *bandwidth* (ancho de banda) y la latencia de los links que debe atravesar para ir del nodo origen al nodo destino.

Transacciones

Cada transacción extiende a la clase Mensaje agregando los siguientes datos:

- El *fee* por byte que paga al minero que la incluya en un bloque.
- El momento a partir del cual puede ser confirmada.

En Bitcoin, cada minero es libre de elegir qué transacciones incluir en los bloques que genera. Es por esto que cada nodo puede incluir en las transacciones que genere un *fee* o recompensa que podrá ser reclamada por el minero que la incluya dentro de un bloque.

El segundo dato que incluimos en nuestras transacciones es información adicional indicando el momento a partir del cual puede ser confirmada. Este valor lo utilizaremos en el simulador cuando necesitemos reproducir una traza real dado que, en ocasiones, necesitamos modelar la situación en la que una transacción fue conocida por todos pero aun así no es agregada al primer bloque luego de su creación.

Bloques

Cada bloque incluye la información base provista por la clase **Mensaje** y además:

- La altura que le corresponde en la *Blockchain*.
- El momento en el que fue minado.
- El identificador del bloque padre.
- La dificultad de la red al momento de ser minado.
- La dificultad acumulada de la red, contando desde el bloque génesis hasta éste.
- Las transacciones que están siendo incluidas en este bloque.

La altura y dificultad acumulada son datos que pueden reconstruirse por cada nodo leyendo la información de la Blockchain local, pero es más práctico en la simulación tener estos datos fácilmente disponibles en los bloques mismos a fin de simplificar la lógica necesaria.

El momento en el que fue minado es necesario con el fin de poder simular fehacientemente el cálculo de la nueva dificultad de la red. Este cálculo es realizado cada 2016 bloques y se toma en cuenta el tiempo que ocurrió entre los últimos 2015 bloques (un bug de Bitcoin tiene un *off-by-one-error* y por eso tenemos que tomar 2015 en vez de 2016). La nueva dificultad se ajustará de manera proporcional con respecto al desfajase del *target* ideal de un bloque cada diez minutos. Ejemplo, si los bloques aparecieron en promedio cada 20 minutos, la nueva dificultad será la mitad de la actual, pero si los bloques aparecieron cada cinco minutos será el doble.

De manera similar, la dificultad de la red que había al momento de minar este bloque es práctica incluirla para simplificar la lógica del escenario de una reorganización de bloques que ocurre cuando un *fork* se transforma en la cadena principal y, además, ocurre un re-cálculo del *target* de la dificultad entre ambas cadenas.

El identificador del bloque padre es esencial para la Blockchain dado que cada bloque creado por un minero debería estar construyendo sobre la cadena de bloques con el mayor trabajo hasta el momento, dado que es la que va a ser considerada por los nodos como referencia para determinar las transacciones válidas. Si el identificador del bloque padre es distinto al tope de la cadena con más trabajo actual estaríamos en presencia de un *fork* en la red. Si la dificultad acumulada de este *fork* es mayor a la representada por la cadena actual entonces pasa a reemplazarla, de manera contraria queda en una estructura de bloques conocidos por si otros mineros siguen construyendo sobre la misma para convertirla en la cadena con mayor trabajo.

Por último, las transacciones que incluye este bloque son aquellas que pasan a ser consideradas como confirmadas por los nodos que reciben dicho bloque. Dado que las reglas del protocolo siguen a las del cliente de referencia, y éste impone un límite de 1 MB por bloque, la heurística elegida para que un minero decida cuales transacciones incluir es la de maximizar la cantidad de **fees** o recompensas.

Inv Es el mensaje que envía un nodo a sus *peers* para avisarles de nuevo inventario de objetos, donde un objecto es una transacción o un bloque. Cada **Inv** incluye la información base de un **Mensaje** y además un mapa de objetos. Éste es un mapa donde la clave es el identificador de un objeto y el valor el tipo del mismo. Este mensaje es luego, opcionalmente, respondido por cada *peer* con un mensaje **GetData** para requerir el subconjunto de objetos del cual no tiene conocimiento actualmente.

GetData Es el mensaje que envía un nodo al *peer* que le notificó primero con un mensaje de **Inv** sobre los objetos que conocía. En este nuevo mensaje también se incluye un mapa de objetos donde la clave es un identificador y el valor es el tipo de objeto requerido (ej: transacción).

1.4 Breve introducción a SimGrid

`SimGrid` es un *framework* orientado a la simulación de eventos discretos para estudiar sistemas de cómputo distribuidos. Permite realizar estudios en los ámbitos de *cluster computing*, *cloud computing*, sistemas *peer-to-peer*, entre otros.

Algunas de su aplicaciones comunes incluyen el análisis de algoritmos abstractos, mediciones de performance y depuración de aplicaciones distribuidas.

	Cliente Bitcoin de producción			Cliente Bitcoin modelado		
	Sistema completo	Minado simulado	Red emulada	Aproximación teórica	Sistema simulado ad-hoc	Sistema simulado completo
Minado	✓	modelo probabilístico	modelo probabilístico	modelo probabilístico	modelo probabilístico	modelo probabilístico
Aplicación	✓	✓	✓	✗	desarrollo ad-hoc	basado en trazas reales
Protocolo	✓	✓	✓	modelo del protocolo	modelo del protocolo	protocolo completo
Red	✓	✓	network namespaces	✗	simulador ad-hoc	Simgrid

Figura 1.6: *Relación entre aspectos del cliente de producción y formas de evaluar los mismos en distintos modelos de emulación o simulación*

En la figura 1.6 podemos apreciar por cada fila una característica del ecosistema de Bitcoin que resulte relevante de ser analizada, como por ejemplo los aspectos de minado o el protocolo en sí. Cada una de estas características puede ser evaluada partiendo desde dos enfoques: uno en el cual contemos con el cliente real de Bitcoin y otro en el cual lo hagamos por medio de un modelo del mismo.

Al considerar el primero de estos enfoques, podemos a su vez observar distintos aspectos experimentales en los que el cliente de Bitcoin puede utilizarse, desde la consideración de una configuración en la cual contemos con un sistema completo, hacia otros en los cuales contaremos con modelos que reemplacen algunos de los componentes reales.

Al considerar el segundo de los enfoques podemos observar que puede ser abordado también desde distintas aproximaciones, por ejemplo desde una que sea puramente teórica y en la cual nos abstraigamos de la existencia de una aplicación o de una red subyacente hasta uno donde contemplemos una simulación del sistema completo.

La elección del enfoque a utilizar debe tener en cuenta ciertos factores como lo son el alto costo computacional del minado y la dimensión de la topología de red de Bitcoin. Dichos factores vuelven inviable en la práctica ciertas experimentaciones si las mismas tuviesen que hacerse sobre la base de un sistema real completo, como lo sería la primera aproximación ilustrada en la imagen previamente mencionada. Con el fin de poder analizar fenómenos que ocurran en una red de tamaño comparable la real de Bitcoin, optamos por desarrollar la experimentación sobre la base de un sistema completamente simulado, lo cual se corresponde con la última de las variantes ilustradas, donde se hace uso del *framework* de SimGrid.

En la sección 3.1 (SimGrid: framework de simulación) expandimos los conceptos técnicos de este *framework* así como las decisiones de configuración que utilizamos para nuestros experimentos.

2

Trabajo Relacionado

Este trabajo de tesis complementa, principalmente, los trabajos de tesis presentadas en años anteriores por Marco Vanotti en [Van16] y Silvio Vileriño en [Vn17].

Mientras en los trabajos mencionados se estudió el comportamiento de criptomonedas mediante la emulación de las redes involucradas, en éste se analizan mediante la simulación con la cual buscamos poder hacer experimentos de mayor escala. En particular, buscamos poder analizar topologías comparables a las red de Bitcoin que cuenta con alrededor de 10000 nodos, mientras que en los trabajos con emulación previamente descriptos se pudo llegar a considerar escenarios con cientos de nodos, lo cual atribuímos principalmente al uso más intensivo de recursos por la necesidad de ejecutar el código real.

Marco Vanotti [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]. También se realiza una validación entre Mininet y Maxinet con experimentos basados en la criptomoneda *RSK*.

Silvio Vileriño [Vn17] se basa en el estudio previo de Marco Vanotti para proponer una metodología para analizar diversos aspectos de las redes Blockchain.

Este trabajo también aprovecha enfoques de análisis utilizados en trabajos de tesis previos. Por ejemplo, en la sección de experimentación utilizamos la métrica de *hash power* desperdiciado que fue presentada originalmente en el trabajo de Nicolás De Carli [DC19].

2.1 Sobre SimGrid

SimGrid aparece como una opción adecuada para encarar este tipo de simulación dado los antecedentes del uso de la misma en experimentos a gran escala.

En Degomme et al. [DLM+17] desarrollaron `SMPI`, un simulador para aplicaciones MPI, con énfasis en poder analizar el comportamiento de tareas de High Performance Computing (Computación de Alto Desempeño). Dicho simulador se basa en la reimplementación del standard MPI (*Message Passing Interface*) sobre la API provista por SimGrid.

Mahmoodi Khorandi y Sharifi [MKS17] utilizaron SimGrid para estudiar los efectos de co-recursos compartidos en el contexto de high performance virtual clusters. Con SimGrid pudieron simular fehacientemente máquinas virtuales, procesos, tareas, comunicaciones, migración de procesos y de máquinas virtuales.

Kolberg et al. [KdBMA+13] desarrollaron un simulador evaluar algoritmos que usan MapReduce. Lo logran recreando el flujo de MapReduce sobre SimGrid y proveen mecanismos para ajustar y evaluar aquellos parámetros que afectan el comportamiento de estos algoritmos, como lo son las políticas de *scheduling* y la distribución de datos.

2.2 Sobre Bitcoin

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

Decker y Wattenhofer [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. El mismo busca calcular la distribución de IPs de 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, pero solo $6{,}000$ IPs se 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.

Además del trabajo anterior, la comunidad puede obtener información de la geolocalización de nodos en el globo en un sitio especializado [Yeo].

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 magnitudes se encuentra a dos desvíos estándares de distancia de la media de esos mismos valores para 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.

Gervais et al. [GKW+16] diseñan un simulador sobre `ns-3` para analizar la seguridad de la red de Bitcoin ante variaciones en parámetros de la red, como tiempo entre bloques o el algoritmo de consenso. Se modelan las Blockchains y protocolos de Bitcoin, Ethereum y otras criptomonedas, así como también algunos ataques conocidos.

Apostolaki et al. [MAL17] analizan posibles ataques a Bitcoin alterando el comportamiento del protocolo de ruteo de internet o interceptando tráfico. Encuentran que un atacante puede aislar al 50 % del *hash power* de la red, tomando el control de menos de 100 prefijos BGP[1]. Aislando partes de la red o ralentizando los tiempos de propagación, el daño realizado a Bitcoin no es despreciable, ya que puede causar que gran parte del *hash power* se desperdicie o permitir el uso de *exploits* para realizar ataques de double spending. Finalmente proponen medidas para contrarrestar el efecto de estos ataques.

Existen trabajos acerca de ataques de minado egoísta a Bitcoin, así como también propuestas para mejorar la resistencia a éstos. Se presentan algunos de ellos a continuación.

Eyal y Gün Sirer [ES18] introducen la noción de minero egoísta y explican sus impactos sobre Bitcoin. Finalmente proponen una modificación al protocolo para disminuir la exposición a estos ataques.

Göobel et al. [GKKT16] estudian el porcentaje mínimo de poder de cómputo necesario para llevar a cabo un ataque en el cual un minero o grupo de ellos se beneficien ocultando del resto de la red una cadena alternativa mejor que la representada por la *main chain*. El objetivo de dicho ataque es invalidar el trabajo realizado por los mineros honestos y conseguir una tasa de inclusión de bloques propios en la *main chain* mayor a la que obtendrían publicando sus propios bloques al momento de minarlos.

Karame et al. [KAC12] analizan la posibilidad tener éxito al ejecutar un ataque de tipo *double spending* considerando escenarios con distintos porcentajes de *hash power* para el atacante y de cantidad de bloques que sea necesario revertir.

Zhang et al. [ZP17] proponen cambios al protocolo de Bitcoin para mejorar la defensa a ataques de minado egoísta. La modificación propuesta incluye un cambio en el protocolo de consenso, beneficiando a bloques que referencien bloques anteriores y sean enviados a tiempo. De esta forma, eliminan el incentivo de no compartir los bloques instantáneamente luego de minarlos.

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

[1] `https://en.wikipedia.org/wiki/Border_Gateway_Protocol`

3

Metodología

En este capítulo explicaremos la metodología general para la creación de la herramienta de simulación y la realización de experimentos, mientras que los detalles particulares del diseño del modelo serán abordados en el siguiente capítulo.

Presentamos el desarrollo de una herramienta que permite modelar una red de Bitcoin en base al *framework* de SimGrid. El modelo fue realizado en base al cliente de referencia de Bitcoin, sobre el cual se medieron los tiempos incurridos en procesar transacciones y bloques.

Para corroborar el adecuado comportamiento del modelo se contrastaron trazas reales con las simuladas por éste. El resultado y análisis de dichas métricas así como la comparación de trazas será abordado en el capítulo 5.

A continuación vamos a describir las características del *framework* utilizado y las opciones de configuración elegidas. Luego, se detallarán las opciones de configuración que incorporamos a la herramienta desarrollada a fin de generar los experimentos propuestos. Por último, se explicarán los mecanismos adoptados para realizar mediciones el resultado de la simulación.

3.1 SimGrid: framework de simulación

SimGrid es un *framework* para construir simuladores de eventos discretos. Éste provee una biblioteca de funciones para modelar simulaciones de aplicaciones distribuidas.

La interfaz expuesta por SimGrid permite declarar una red de nodos donde cada uno puede ejecutar tareas independientes. Éstas se comunican utilizando un mecanismo basado en *mailboxs*, los cuales explicaremos en la siguiente sección. El motor de simulación de SimGrid es serial y los procesos son ejecutados en un solo *core* de forma colaborativa. Al ser serial, hay ciertas diferencias con respecto al comportamiento real que estamos modelando, por ejemplo:

- La serialización que hace SimGrid es una de las posibles dado un conjunto de eventos concurrentes.

- Si dos tareas de distintos actores están listas para ser ejecutadas en el mismo instante, el motor elegirá alguna de ellas primero seguida de la otra.
  Si bien ambos actores verán que sus tareas empezaron en el mismo instante simulado, en la práctica haber empezado a simular una antes que la otra podría implicar que la primera tarea use un recurso (como un *link*) por el que luego deba esperar la segunda.

- Puede suceder que ambas tareas impliquen el envío de un mensaje a un tercer actor y el orden en que este nuevo actor los reciba termine determinado por la elección inicial que hizo SimGrid[1].

Como beneficio de este modelo serial, mono-*core* y colaborativo podemos mencionar la facilidad en la reproducción de trazas si queremos repetir un experimento, dado que la serialización es determinística para una misma simulación. Si la simulación no siguiese dicho modelo y, por ejemplo, ocurriese en distintos *cores* podría haber condiciones de carrera que alterasen el resultado en ejecuciones para la misma entrada de datos. También podría ser que evitar dicha condición de carrera implicase más lógica del lado del simulador para coordinar la actividad de los *cores*. Si, por ejemplo, fuese necesario sincronizar eventos con un reloj distribuido ello implicaría costos significativos de implementación y performance en la simulación que a su vez degradarían los beneficios de un modelo de cómputo concurrente.

Una tarea es desalojada del motor de SimGrid cada vez que ejecuta una función que implica el paso de tiempo en el modelo. El tiempo de cómputo es simulado y se genera por medio de eventos en el sistema. En este sentido, el código ejecutado por el programa de usuario desarrollado para modelar el comportamiento de la tarea en cuestión no implica el paso de tiempo en el modelo. Una explicación más detallada se describe en 3.1.2 (Utilización).

Para comprender las decisiones de diseño del modelo, se detallarán partes del funcionamiento a bajo nivel de SimGrid.

3.1.1 Conceptos Principales

- **Actor**: representa un nodo de la red.
  Cada uno tiene código y datos privados asociados.
  Ejecutan *tasks* en un *host* determinado.

- **Task**: representa un proceso, en particular el cómputo a realizar y los datos a transferir.

- **Mailbox**: son puntos de encuentro entre actores.
  Los mensajes producidos por los actores se envían y reciben hacia/desde los mailboxes.
  Se pueden establecer estos puntos de encuentro independientemente de la ubicación en la red dado que SimGrid se hará cargo del *routing* necesario.
  Los mailboxes se identifican por medio de *strings*, por ejemplo: `host:puerto` o cualquier otra identificación que tenga sentido en el dominio del problema.

Características del modelo de red

SimGrid soporta distintas configuraciones para el modelo de red. Todas ellas comparten las siguientes características generales:

- cada mensaje es transmitido por una lista de *links*.

- cada *link* entre dos *hosts* tiene un *bandwidth* y una latencia asociada.

- el *link* puede ser unidireccional o bidireccional.

- el uso de un *link* puede ser compartido entre todos los mensajes a enviados a través del mismo o exclusivo para cada mensaje puntual.

- SimGrid elige el camino entre los *links* de los actores que están enviando/recibiendo mensajes hacia/-desde cada *mailbox*. Dicho camino no necesariamente es siempre el mínimo.

[1] Nota: los conceptos de actores y tareas se detallarán en la sección 3.1.1

Limitaciones principales del modelo de red

Para simular el tiempo de transmisión de un mensaje SimGrid tiene en cuenta el tamaño del mensaje, las características de la lista de *links* por las que debe ser transmitido y los posibles mensajes que estén utilizando la capacidad de dichos *links* en ese momento.

Las limitaciones de SimGrid son las siguientes:

- el modelo del mecanismo de *slow start* de TCP puede no ajustarse al experimento.
  Permite configurar un modelo de red donde se ajustan las latencias para hacerlo más realista pero solo es recomendable si los mensajes a enviar son mayores a 100 kB. En caso contrario, uno observa una latencia hasta 13 veces mayor que en el escenario real.

- no modela congestión de la red ni pérdida de paquetes.

- los posibles caminos entre *hosts* por los que pueden transmitirse los mensajes son precomputados al momento de armar la red. En tiempo de ejecución el *routing* es estático, es decir, que SimGrid elige siempre el mismo camino para transmitir un mensaje entre *hosts*.

Características del modelo de cómputo

Podemos destacar el hecho de que cada *host* tiene una cierta velocidad medida en FLOPS (Floating Point Operations per second) y, a su vez, cada tarea de cómputo a realizar indica los FLOPS que requiere. Luego, SimGrid simula el tiempo correspondiente de acuerdo a las tareas que están siendo ejecutadas simultáneamente en dicho *host* según la capacidad de cómputo de este último.

3.1.2 Utilización

Conceptualmente, una simulación está compuesta de un conjunto de recursos que proveen ciertas capacidades (ej: los *links* en la red deben transmitir paquetes, las CPUs deben realizar cómputos, etc) y de una aplicación que utiliza estos recursos para llevar a cabo una serie de acciones.

SimGrid provee APIs para que los actores puedan interactuar con los recursos simulados. Es el desarrollador quien debe hacer uso de dichas interfaces para implementar la simulación de su problema.

Cada recurso está administrado por un modelo que indica cuándo una acción será completada, por ej: el modelo de red determina el tiempo que demorará transmitir un mensaje por un *link*. SimGrid consulta a todos los modelos el tiempo de finalización de los recursos siendo utilizados y se queda con el mínimo de estos para avanzar el reloj interno de la simulación. De acuerdo al tiempo mínimo para completar una tarea luego queda determinado el próximo actor que actuará. En dicho momento se ejecuta el código de usuario asociado a dicho actor. El inicio de nueva acción (comunicación o cómputo) vuelve a bloquear al código de usuario y recomienza el mismo ciclo previamente descripto.

El motor de simulación es colaborativo sin desalojo, lo que implica que las tareas ejecutan hasta que sucede un evento que desencadena nuevos eventos de otras tareas. El *scheduler* de SimGrid siempre espera que una tarea termine o genere un evento para continuar y dar el control a otra tarea.

Utilización, funcionamiento y extracción de resultados

A continuación damos una breve introducción a conceptos centrales en nuestras simulaciones. Dichos conceptos luego serán abordados en más detalle en sus propias secciones.

- **Datos de entrada**, compuestos por:

  - ***Platform***: un archivo con la información de los recursos físicos disponibles a simular: los *hosts* que proveen capacidad de cómputo y los *links* entre *hosts* que proveen conectividad.

- ***Deployment***: la declaración de los actores involucrados en la simulación, los *hosts* que van a estar utilizando, así como cualquier información extra de inicialización necesaria para los mismos.

- **Aplicación**: el código de usuario que hace uso de las APIs provistas por el framework de SimGrid, tanto para simulación como para *logging*.

- **Salida**: es un registro de eventos con el tiempo, evento ocurrido y el nodo donde sucedió. Se puede configurar un nivel de criticidad para controlar el tipo de eventos a incluir, así como también configurar el formato de las lineas que componen esta salida.
  En base al registro de eventos provisto por este *log* se puede extraer la información necesaria para analizar el comportamiento de la simulación.

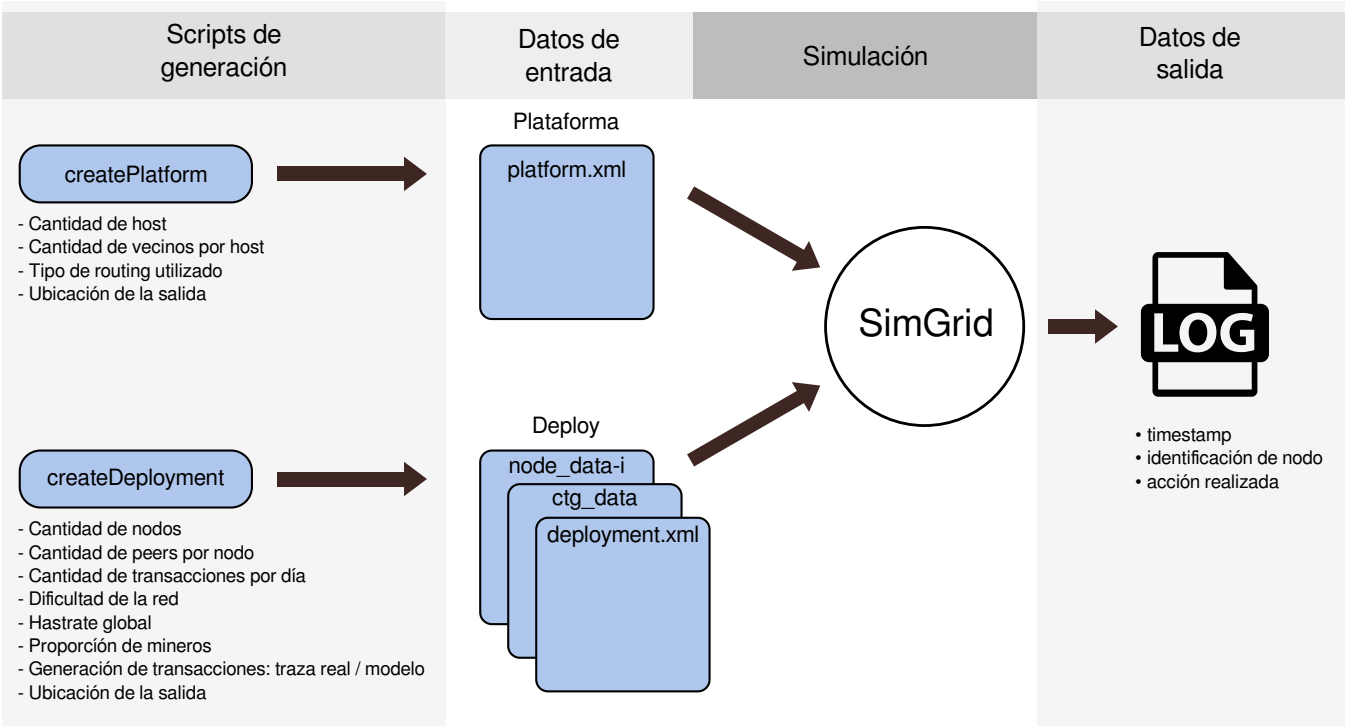


Figura 3.1: *Relación del simulador con datos de entrada y salida*

En la figura 3.1 introducimos la relación del simulador con los datos de entrada, así como también la de estos últimos con las herramientas auxiliares para generarlos y, por último, el *log* resultante de la simulación. Las interacciones que ocurren al interior de SimGrid serán abordadas en el capítulo 4 (Modelo) en la figura 4.2 mientras que a continuación explicaremos detalladamente los elementos mencionados en esta figura.

Datos de entrada Al utilizar SimGrid es necesario proveerle como argumentos al simulador los datos de *platform* así como los de *deployment* específicos para la ejecución.

Platform se refiere a los recursos físicos que vamos a considerar para una simulación, entre los que podemos mencionar:

- *host*: representa el recurso en el que un actor puede ejecutarse. Algunos atributos que pueden configurarse para este recurso son la velocidad (en FLOPS), la cantidad de *cores* y el estado inicial (prendido/apagado).

- *cluster*: como su nombre lo indica, representa un *cluster* de cómputo. Algunos de sus atributos son la cantidad de *hosts* dentro del mismo, así como el ancho de banda y latencia entre dichos *hosts* internos.
- *link*: representa una conexión de red. Entre sus atributos principales están el ancho de banda, latencia y política de red (ej: si el ancho de banda es compartido entre distintos paquetes de red o asignado completamente a uno).

Dicha configuración es estática y no varía en el tiempo.

Por otro lado, la plataforma lógica determinada por los datos de *deployment* se corresponde con conceptos propios de la aplicación simulada (en nuestro caso, conceptos como *peers*, transacciones por día, dificultad de la red, etc). Si bien dichos conceptos serán propiamente abordados en la sección 4.1 (Simulación del cliente), a continuación mencionamos los mismos con el fin de mostrar su relación con los archivos de entrada.

Archivos de Entrada y Salida

- Archivo `platform.xml`: representa los datos de la plataforma física de nuestra simulación.
- Archivo `deployment.xml` y auxiliares: representan la plataforma lógica de nuestra simulación y están dados por:
  - Un archivo `JSON` por cada nodo lógico de la aplicación donde se indica si dicho nodo genera transacciones, los otros nodos que va a tener como *peers* y la traza de transacciones a seguir (si no se eligió la opción de generarlas en base a un modelo). En el caso de que el nodo sea un minero se incluye también el *hash power* que tiene disponible.
  - Un archivo `JSON` global llamado `ctg_data` con el tipo de generación de transacciones: modelo o traza real.
    Si se sigue un modelo, se indica si el tipo de generación es uniforme para todos los nodos o sigue una distribución exponencial y en dicho caso el valor del λ correspondiente.
    Si se sigue una traza real entonces los datos para la generación de cada evento junto con el nodo que debe producirlo (ej: hora de creación de una transacción, tamaño y nodo que debe hacer el *broadcast* al resto de la red).
  - Un archivo `XML` llamado `deployment.xml` donde por cada nodo lógico se referencia el *host* descripto por el archivo `platform.xml` donde dicho nodo debe simularse.
- Archivo de *log*: es la única salida producida por nuestro simulador con todos los eventos que ocurrieron. En cada línea de dicho *log* se incluye: la hora del evento, el nodo interviniente y el tipo de acción realizada.

El *log* de salida se implementó utilizando las capacidades provistas por SimGrid para registrar los eventos ocurridos durante la simulación. Mediante estos métodos obtenemos por salida estándar la información de los eventos que ocurren en cada uno de los actores, la cual puede ser fácilmente redireccionada a un archivo para su persistencia y conveniente posterior análisis. En la sección 3.1.2 se describe más detalladamente el formato de salida.

Aplicación Para la implementación del protocolo de red se analizó la documentación de `https://en.bitcoin.it/wiki/Protocol_documentation` dejando de lado aspectos como *peer discovery*, por considerarlo fuera del alance de este trabajo.

Para la simulación de los tiempos de validación de transacciones y bloques se tomaron muestras reales del cliente de referencia. En la sección 4.1 describimos más detalladamente dichas mediciones.

Salida: extracción de resultados Hacemos uso de la API de log de SimGrid para registrar los eventos durante una simulación donde cada uno tiene el siguiente formato:

```
<tiempo>  <nodo>: <acci\'on realizada>
```

Un ejemplo concreto es el siguiente:

```
436.533838  node-296: informing 225 of 1 objects in our inventory
462.817250   node-72: creating tx 8386373514783821425
462.817250   node-72: creating block 1587103434935014363 with 3 txs. height: 1, parent
462.817250   node-72: next activity will be 18475.793139
462.817250   node-72: received a new block 1587103434935014363 from 72 with 3 txs
462.817250   node-72: confirmed tx 8079046571094085839 in block 1587103434935014363
462.817250   node-72: confirmed tx 8386373514783821425 in block 1587103434935014363
```

En el mismo encontramos el tiempo de simulación, seguido del actor encargado de ejecutar la acción y por último un mensaje correspondiente a la acción realizada. En el anexo 9.2.2 se describe la operatoria para configurar algunos de los escenarios más complejos.

3.2 Generación de casos de prueba

Los experimentos que llevamos a cabo necesitan de los archivos de entrada mencionados en la sección 3.1.2. Dado que generarlos manualmente sería complejo por las múltiples variables a tener en cuenta, incluimos herramientas adicionales encargadas de crear los archivos necesarios para las plataformas físicas y lógicas, los cuales luego son utilizados por el simulador. En el anexo 9.2.1 listamos dichas herramientas y describimos su uso.

4

Modelo

Este capítulo detalla en 4.1 la decisiones de diseño para simular un cliente Bitcoin y a continuación en 4.2 se explican los aspectos de simulación de la red subyacente.

4.1 Simulación del cliente

La simulación del cliente no consiste solamente en emular el comportamiento del mismo. Sino que es necesario tener en cuenta el tiempo que demora el cliente en realizar determinadas acciones.

A fin de poder simular el proceso de validación de transacciones y bloques, buscamos la forma de aproximarlo, considerando los aspectos más relevantes del cliente de referencia y la precisión buscada en nuestro modelo. Decidimos aproximar este tiempo que demoran estos procesos, en relación al tamaño de las transacciones. En nuestro simulador de eventos discretos, las transacciones que se crean tienen un tamaño fijo y, a su vez, los bloques que se envían están compuestos por transacciones de tamaño fijo. Es decir, el tamaño de un bloque solamente depende de la cantidad de transacciones que contiene.

Para tomar tiempos reales, instrumentalizamos el cliente de referencia, configurando dicho cliente a la red de *testnet*. Dicha red representa la versión de *testing* de Bitcoin, utilizando el mismo protocolo, que además de validar transacciones nos permite generar transacciones propias de manera gratuita.

Se recopilaron $220{,}000$ muestras de transacciones y, por cada una, se grabó el tiempo necesario para validarla junto con el tamaño de la misma. En la tabla 4.1 describimos los detalles de la plataforma para dichas mediciones.

Configuración del ambiente de pruebas	
Sistema Operativo	Linux
Distribución	Ubuntu 16.04.6 LTS
Procesador(es)	Intel(R) Core(TM) i5-6300U CPU @ 2.40 GHz ($\times 4$)
Memoria RAM	16 GB
Disco Rígido	451 GB
Versión de Bitcoind	0.16

Tabla 4.1: Configuración del ambiente de pruebas

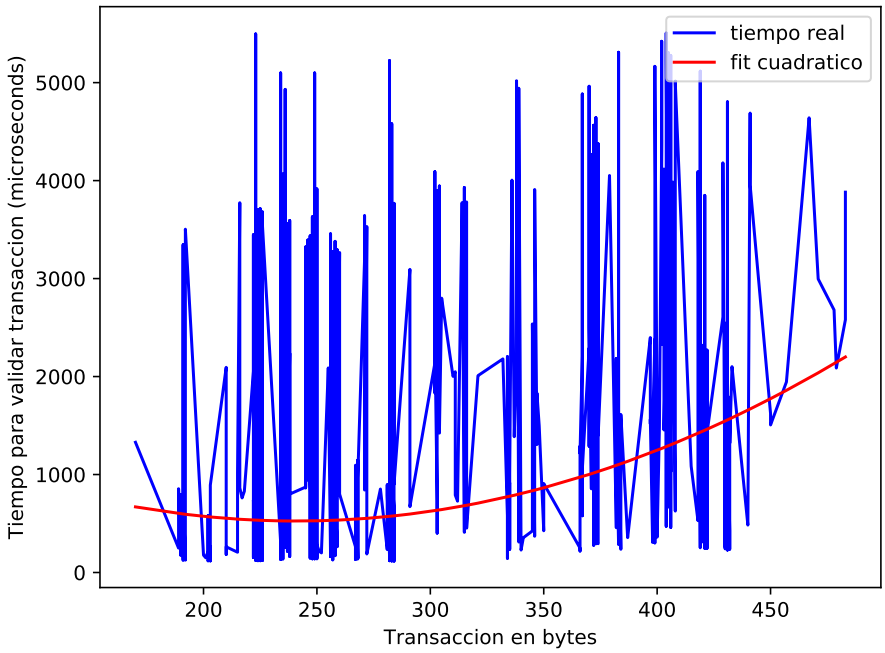


Figura 4.1: *Tiempo de validación y aproximación por función cuadrática*

En la figura 4.1 graficamos los tiempos observados de entre las $220{,}000$ transacciones recopiladas. Dado que hay indicios en la bibliografía de crecimiento $O(N^2)$ con respecto a la cantidad de *inputs* de una transacción[1], decidimos hacer un *fit* de una función cuadrática con respecto a los tiempos analizados, el cual indicamos en la figura con la curva "fit cuadrático". Notamos una amplia varianza entre los tiempos observados y estimamos que esto se debe a las particularidades propias de cada transacción donde los tiempos de validación crecen $O(N^2)$ en relación al tamaño solo en casos patológicos. En [2] se describen características para llegar a estas cotas superiores en lo que a tiempo de validación respecta.

Para simular los tiempos de transacciones utilizamos los coeficientes del *fit* cuadrático mencionado previamente según el tamaño de cada una de ellas, mientras que para simular el tiempo de los bloques sumamos los tiempos de las transacciones que lo componen.

4.1.1 Interrelación Aplicación - Framework de SimGrid

A continuación, en la figura 4.2, detallamos los componentes que interactúan en SimGrid, tanto la parte de aplicación encargada de modelar el cliente de Bitcoin así como la plataforma simulada provista por el *framework*, y también la relación de ambos con los archivos de entrada y salida. Esto complementa la figura 3.1 vista en el capítulo 3 (Metodología) donde habíamos mostrado la relación de la entrada y salida con el simulador.

[1] `https://bitslog.wordpress.com/2017/01/08/a-bitcoin-transaction-that-takes-5-hours-to-verify/`

[2] `https://scalingbitcoin.org/montreal2015/presentations/Day2/11-Friedenbach-scaling-bitcoin.pdf`

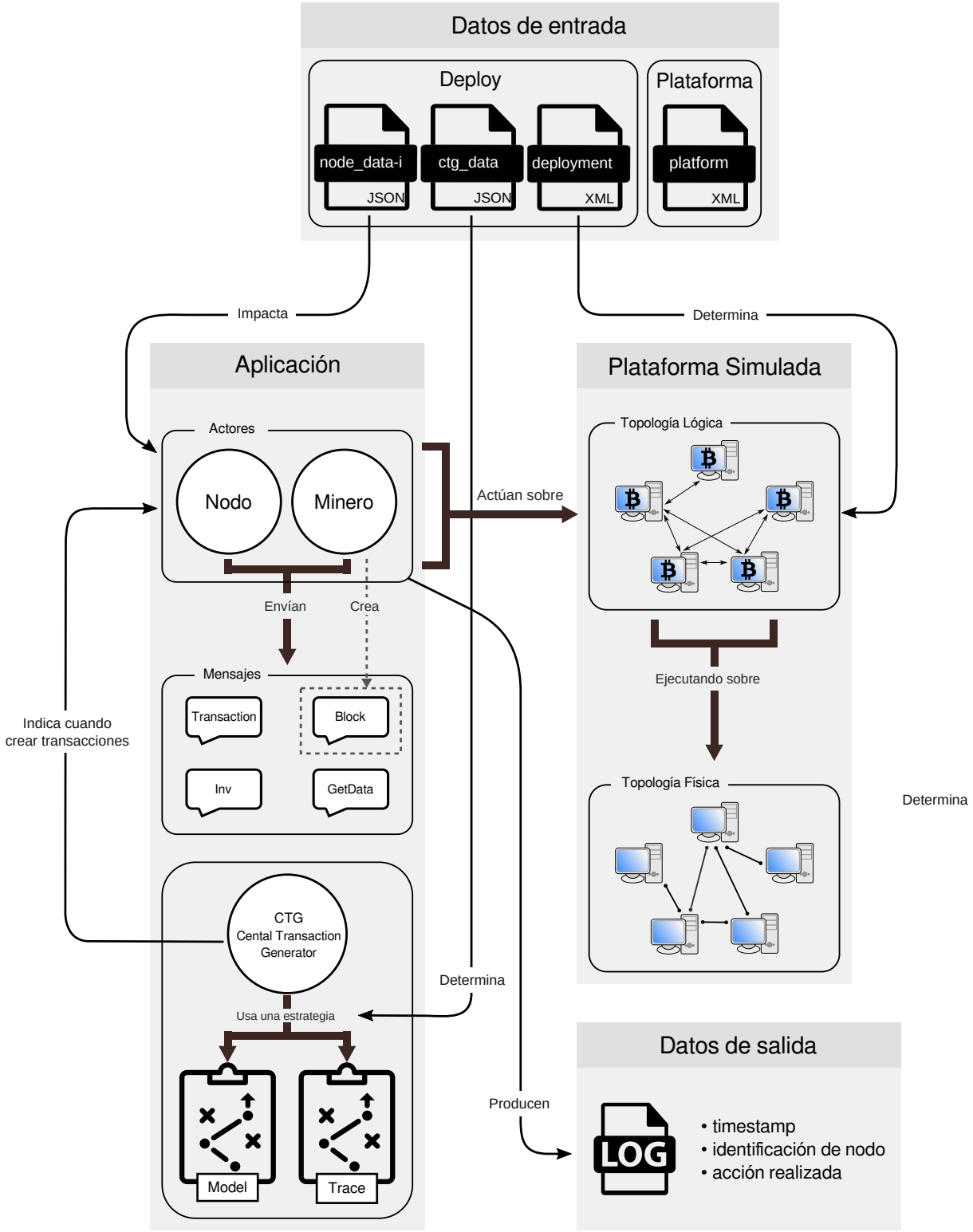


Figura 4.2: *Relación entre datos de entrada, aplicación, plataforma simulada y salida*

Podemos observar en la figura 4.2 cómo los datos de entrada son consumidos tanto por el framework de SimGrid como por el código de aplicación. Los archivos `XML` son los utilizados por SimGrid para definir las topologías lógicas y físicas a simular, mientras que los archivos de tipo `JSON` son utilizados para la configuración de los actores (nodos y mineros).

La tarea de los actores consiste en la creación, envío y validación de mensajes siguiendo el protocolo de red con el fin de simular dicho aspecto de la red Bitcoin. Los mensajes *Transaction*, *Block*, *Inv* y *GetData* son el

subconjunto modelado del protocolo de red de Bitcoin, cuya finalidad será abordada más adelante. En particular, podemos observar cómo los mensajes de tipo *Block* son solo creados por actores de tipo Minero. Los mensajes de tipo *Transaction* son creados tanto por ambos actores pero la frecuencia de aparición de dichos mensajes será establecida por un componente CTG ("Central Transaction Generator") que describiremos a continuación.

Por último, podemos destacar como la salida está dada por un archivo de *log* producto de las acciones realizadas por los actores simulados en la aplicación.

4.1.2 Actor central de generación de transacciones (CTG)

A fin de modelar la aparición de transacciones en la red se optó por definir un actor central encargado de indicar a cada nodo simulado en qué momento debe crearlas y con qué tamaño. Esta decisión se debe a que la aparición de dichas transacciones en el "mundo real" responde a comportamientos arbitrarios de los distintos agentes que utilizan la Blockchain, o sea, no es el propio cliente de Bitcoin quien decide el momento de creación de una transacción sino agentes externos como usuarios finales que usan una billetera virtual o *exchanges* de criptomonedas. El nuevo actor implementado es, entonces, una abstracción de las acciones de agentes externos que operan con el cliente de referencia.

Para evitar afectar la simulación, el actor central interactúa con los nodos simulados a nivel host y no a nivel del protocolo (ej: este actor no envía ni recibe mensajes usando la red simulada por Simgrid). Un ejemplo práctico para ilustrar la relación de este actor con los otros puede darse si consideramos que un nodo está simulando una aplicación de billetera de Bitcoin:

1. El usuario de la aplicación de billetera virtual decide hacer una transacción para comprar un bien o pagar un servicio. Este evento es causado por una acción humana, es decir ocurre fuera del protocolo Bitcoin.

2. La acción genera una interacción entre nodos Bitcoin que siguen el mismo protocolo de comunicación, validación y consenso. Estas interacciones sí, son las que estamos interesados en modelar.

3. A raíz de las interacciones entre los nodos, una transacción fue agregada a un bloque y éste minado. Validando así la operación con la billetera virtual.

El actor central representa, entonces, todo el comportamiento externo que no es capturado por el protocolo Bitcoin, y lo hace sin afectar el modelo de simulación, debido a que se comunica directa con el nodo que debe llevar a cabo la acción indicada.

Cómo se determina el próximo evento de creación de transacción para un nodo En la rutina de inicialización de un nodo, éste le consulta al actor central de transacciones el instante en el futuro en que tendrá que generar una transacción. Antes de empezar el ciclo de ejecución principal, el nodo compara el instante actual con el correspondiente al de la próxima transacción a generar, si el mismo fue alcanzado, entonces genera dicha transacción. Esta transacción luego será comunicada a sus *peers* durante el ciclo de ejecución principal.

De no haber aún alcanzado el instante en el tiempo necesario para la generación de la próxima transacción entonces opera normalmente, ejecutando el ciclo principal y luego se mantiene en inactividad durante los siguientes lapsos de tiempo, según el caso:

- **100 ms**: Si no hay mensajes que ya estén listos provenientes de los *peers* del nodo.

- **0 ms**: En caso que la validación previa indique la existencia de mensajes pendientes de ser consumidos de sus *peers*. En este escenario el nodo no pasa a estar inactivo, sino que reinicia el ciclo.

- **tiempo hasta próxima transacción**: en caso que el evento de generación de transacción esté contenido en los próximos 100 ms. Por este motivo, en lugar de no entrar en inactividad durante 100 ms, se mantiene inactivo justo lo necesario para poder generar y comunicar la transacción en el momento que le fue indicado por el actor central de transacciones.

El cliente de referencia indica seguir las primeras dos directivas para el tiempo de inactividad, mientras que la tercera, relacionada a la generación de una transacción, fue agregada a modo de simular lo más fielmente posible el comportamiento del cliente de referencia (en el cual hay un *thread* separado encargado de generar y transmitir una transacción cuando la misma es indicada por el usuario final). Al considerar ese tercer escenario para el tiempo de inactividad, logramos simular fehacientemente el instante de creación y comunicación, dado que mantenemos al nodo inactivo el tiempo necesario.

Es importante señalar que ninguna de las validaciones o interacciones entre un nodo y el actor central, afectan la simulación. Ninguna de estas interacciones se consiera como paso del tiempo en la simulación, ya que no produce envíos de mensajes (ej: transmitir una transacción), o agrega tiempo de ejecución simulado (ej: validación de un bloque), o tiempo agregado de inactividad adicional (ej: esperar por un tiempo fijo).

Por último, una vez que el nodo llega al momento en que debe generar una transacción, se vuelve a consultar al actor central el instante del próximo evento. De esta forma se reinicia el ciclo de eventos con el actor central.

4.1.3 Ciclo de ejecución principal

Todo nodo de Bitcoin basado en el cliente de referencia cuenta con un ciclo principal de ejecución, el cual buscamos modelar fielmente. En esta sección describimos en detalle cómo fue modelado dicho ciclo.

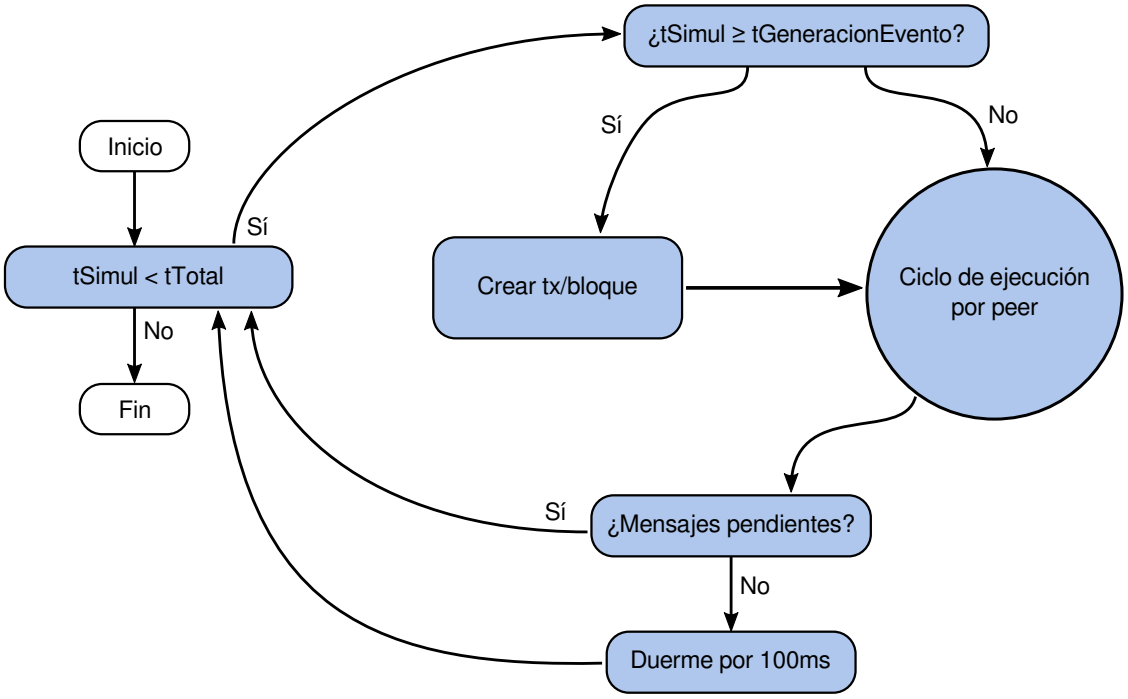


Figura 4.3: *Ciclo principal de ejecución*

En la figura 4.3 podemos apreciar una vista esquemática del ciclo principal de ejecución. Dicha ejecución continuará siempre que el tiempo actual de simulación `tSimul` sea menor al total `tTotal`. Dentro se comienza verificando si se está en condiciones de generar un evento, es decir, si se debe crear una transacción

o un bloque. A fin de determinar el instante `tGeneracionEvento` en que se debe generar el evento, se realiza el siguiente procedimiento, el cual depende del tipo de objeto a crear:

- **Transacción**: Interviene el actor central, dicho actor es el encargado de indicar a cada nodo el momento de crear una transacción.
- **Bloque**: Se determina de acuerdo a la dificultad actual de la red y al *hash power* del minero.

Luego de la etapa de generación de actividad por parte del nodo, se continúa con de envío y recepción de mensajes hacia y desde los *peers* del nodo (detallado en la sección 4.1.4).

Respetando el comportamiento del cliente de referencia, se le envía a cada *peer* todos aquellos mensajes que se encuentren pendientes. Por otro lado, para los mensajes entrantes, se procesa por cada *peer* a lo sumo uno de los mensajes que estén listos en su correspondiente cola de pendientes. Este es un mecanismo implementado desde el cliente de referencia para evitar ataques de tipo *denial of service*, donde un solo *peer* nos impediría seguir atendiendo el procesamiento de mensajes hacia el resto.

Finalmente, verificamos si luego de haber recorrido todos nuestros *peers* hay aún trabajo pendiente listo para ser procesado. De no ser el caso, se espera durante 100 ms antes de recomenzar el ciclo nuevamente. Este tiempo de espera, parte del comportamiento del cliente de referencia.

4.1.4 Ciclo de ejecución por *peer*

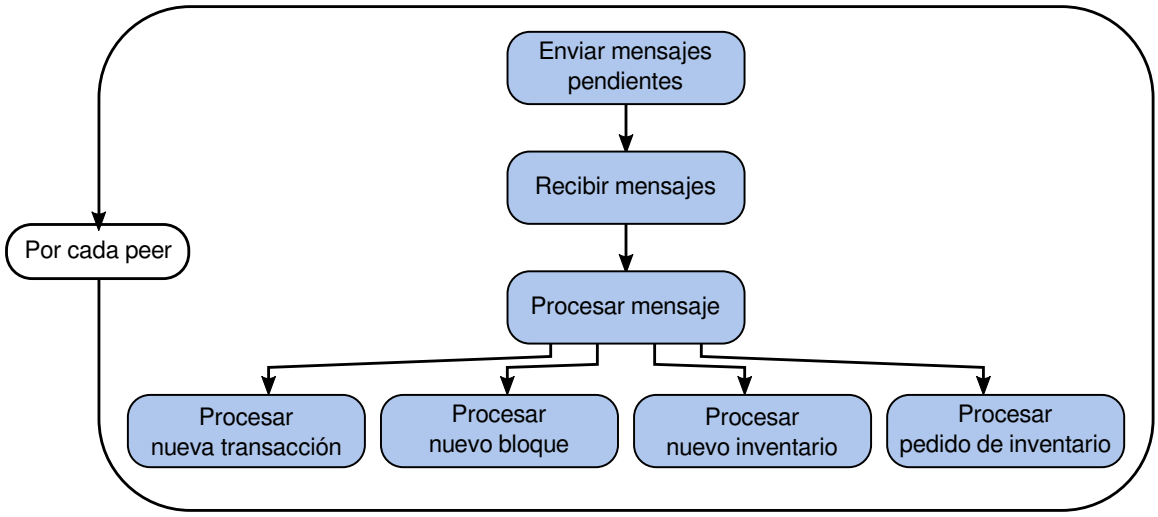


Figura 4.4: *Procesamiento / envío de mensajes de / hacia un* peer

En la figura 4.4 podemos observar que por cada *peer* que un nodo tenga como vecino se realiza un procesamiento en dos etapas:

- **Primera etapa**: Si el nodo tiene mensajes disponibles para enviar al *peer* en la cola correspondiente, se envían dichos mensajes.
- **Segunda etapa**: El nodo procesa, a lo sumo, uno de los mensajes que tenga listos para consumir provenientes del *peer*.

Si en la segunda etapa hubiese un mensaje a procesar, se utilizará una rutina diferente para cada tipo:

- `Inv` (nuevo inventario): El *peer* nos informa los hashes de objetos (transacciones o bloques) que conoce. Si el nodo que recibe este mensaje desconoce uno o más hashes, entonces responderá con un mensaje de `GetData` para obtener el objeto completo.

- `GetData` (pedido de inventario): El *peer* nos solicita los objetos completos en base un aviso de inventario (`Inv`) que le comunicamos previamente.

- `TX` (transacción): El nodo nos informa de una transacción que pudo haber sido creada por el *peer* o por otro nodo de la red. Este mensaje ocurre como consecuencia de uno previo de tipo `GetData` donde se provee el hash de la transacción de la cual se solicitará el objeto completo.

- `Block` (bloque): El nodo nos informa de un bloque que pudo haber sido creado por el *peer* o por otro nodo de la red. Este mensaje suele ocurrir generalmente como respuesta a uno de tipo `GetData` en el cual se provee el hash del bloque del cual se solicitará el objeto completo. Una excepción a esta modalidad se da cuando un minero informa los bloques que minó. Dicho minero le envía a sus *peers* mensajes de tipo `Block` para evitar el *roundtrip* que implicaría la secuencia:

  · Minero $\rightarrow$ Peer: `Inv` | el minero le informa a uno de sus *peers* el hash del bloque recientemente minado
  · Peer $\rightarrow$ Minero: `GetData` | el nodo le pide al minero el contenido completo de dicho bloque
  · Minero $\rightarrow$ Peer: `Block` | el minero le envía el contenido completo del bloque a su *peer*

4.2 Topologías y redes privadas

Comenzaremos esta sección introduciendo conceptos acerca de topologías físicas y lógicas. Éstas hacen referencia a distintos niveles dentro de una red. Más adelante hablaremos de Blockchains privadas sobre Bitcoin y en particular, sobre la generación de topologías virtuales relevantes a nuestro trabajo.

4.2.1 Tipos de topologías

Definimos dos tipos de topologías según el nivel de la red donde se haga foco. Cuando nos referimos a la topología lógica de un experimento, estamos hablando de la relación de *peers* entre nodos en la red de la criptomoneda. Este nivel de red existe solamente en el contexto de la aplicación.

Al referirnos a la topología simulada estamos hablando de la red de *hosts* físicos que son simulados utilizando SimGrid. La topología simulada entonces representa una red de computadoras de N *hosts*.

La tabla 4.2 describe una posible configuración válida.

Topología lógica	Los nodos siguen el protocolo Bitcoin. Ejemplo: Si la topología fuese una clique entonces todos los nodos en la red de Bitcoin tienen a todos los demás como *peers*.
Topología simulada	La red de hosts físicos que son simulados con SimGrid. Ejemplo: Si la topología fuese un anillo entonces el host i-ésimo solo tiene conectado al host $i + 1$-ésimo o al primero.

Tabla 4.2: Ejemplo de configuración de topologías

Utilizando el ejemplo de la configuración en la tabla 4.2, cuando un nodo en Bitcoin realiza un *broadcast* ve al resto de la red como *peers*, pero en la topología subyacente las conexiones no necesariamente son directas.

En este caso, un mensaje enviado desde el nodo 1 al nodo n, es un paso en la topología lógica, pero tiene que atravesar $n - 1$ links de red física simulada subyacente. Dicho ejemplo se ilustra en la imagen 4.5 para una red de cinco nodos lógicos sobre una red física subyacente de cinco *hosts*.

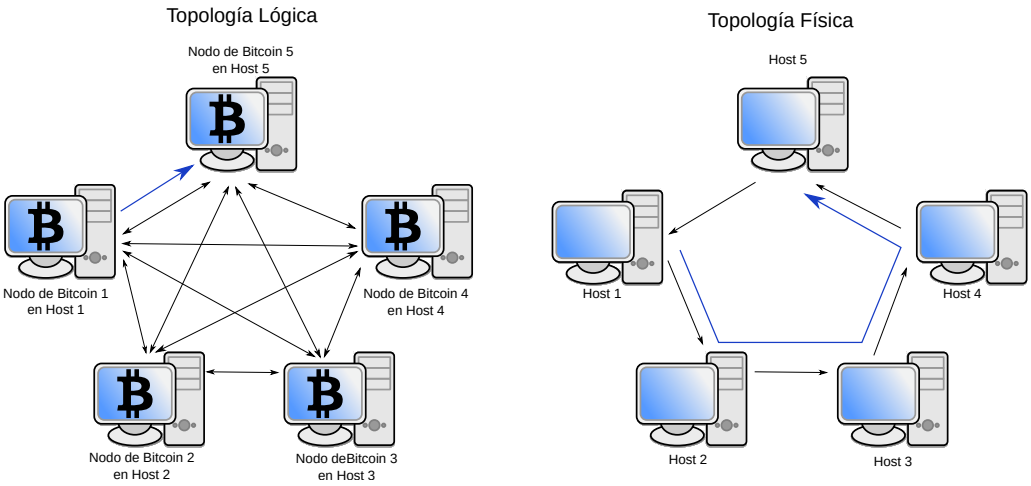


Figura 4.5: *Comparativa de envío de un mensaje según la topología de la red*

4.2.2 Modelo de Red de SimGrid

SimGrid puede ajustarse a particularidades de la red y emular su comportamiento. Para esto implementa técnicas tales como el *slow start* al simular una conexión TCP o modifica su funcionamiento global dependiendo del tamaño del mensaje a enviar. Estos ajustes se incluyen en distintos modelos de red disponibles[3], entre los que podemos destacar:

- **CM02**: Es un modelo antigüo sin factores de corrección para *slow start*. La simulación para mensajes pequeños es poco realista.
- **LV08**: Es el modelo por defecto, en el cual *slow start* tiene factores de corrección sobre la latencia y ancho de banda.
- **SMPI**: Es una especialización para modelar más fehacientemente el comportamiento de un *cluster* para *high performance computing*. Está basado en el modelo *LV08* pero en vez de usar un factor de corrección igual para todas las comunicaciones lo adapta según el tamaño del mensaje e enviar.

En la sección 5.3 (Validación de protocolos de red y consenso) presentamos pruebas de validación de nuestro simulador contra una traza real de transacciones y bloques. Durante la realización de dichas pruebas constatamos que el modelo *SPMI* era el que brindaba la mejor correlación entre la traza simulada y la real y por ende fue el elegido para nuestro trabajo.

[3] `http://simgrid.gforge.inria.fr/simgrid/3.20/doc/options.html`

5

Validación

5.1 Motivación

En este capítulo mostraremos experimentos para validar la metodología previamente presentada. Se busca verificar con los mismos que la simulación siga el comportamiento del cliente de referencia y que los resultados obtenidos en base a la reconstrucción de una traza real sean consistentes con los observados en la red de Bitcoin.

5.2 Validación del ajuste de dificultad

El protocolo de Bitcoin incorpora un mecanismo de reajuste de dificultad de la red que ocurre una vez cada 2016 bloques con el fin de mantener una taza de aparición de un bloque, en promedio, cada diez minutos. De haber más mineros en la red, esto aumentaría el *hash power* global de la red y, de no mediar ningún ajuste, los bloques empezarían a aparecer más seguido. A su vez, la situación opuesta en la cual disminuye el poder de cómputo global generaría mayor tiempo de espera entre bloque y bloque, incrementando el tiempo de confirmación de las transacciones.

Este aspecto fue modelado en nuestro simulador por ser una parte crítica del protocolo y, si bien no fue necesario utilizarlo en la parte de experimentación, aportaría valor para futuros trabajos en los cuales se necesite evaluar períodos de tiempo más extensos.

Presentaremos experimentos para validar el correcto modelado de este aspecto. Analizaremos la variación de la dificultad para una red simulada con dos escenarios: uno estático donde el *hash power* global disponible se mantiene constante (es decir: misma cantidad de mineros con mismo *hash power* durante la simulación), y un segundo experimento donde modificaremos el *hash power* global en forma dinámica durante el mismo. En la tabla 5.1 indicamos las características generales de los experimentos realizados.

Configuración del experimento	
Topología física	15 hosts
Topología lógica	15 mineros
Distribución de *hash power*	Obtenida de https://www.blockchain.com/pools
Porcentaje mineros	100 % de la red (no se consideraron nodos comunes)
Tiempo simulado	280 días (20 períodos de ajuste de dificultad)
Tiempo entre bloques	600 segundos (tiempo estándar)

Tabla 5.1: Configuración del experimento para la validación del ajuste de dificultad

Variación en los tiempos de ajuste de dificultad En la figura 5.1 se consideraron los dos experimentos previamente mencionados para ilustrar la variación en los tiempos de ajuste de la dificultad de la red.

- el eje x indica la cantidad de horas que duraron los experimentos.
- el eje y indica la dificultad de la red.
- las dos series representadas indican el tiempo que duró cada período de ajuste en los dos escenarios considerados.
- la serie "hash power constante" indica el primero de los escenarios considerados, donde el poder de cómputo global en la red se mantiene.
- la serie "hash power variable" indica el segundo escenario dinámico donde se modifica el poder de cómputo global durante el experimento determinado por los siguiente eventos:
  - la linea vertical punteada "reducción hash power" representa el instante donde se reduce el *hash power* global luego de cinco períodos de ajuste (es decir: cinco períodos de 2016 bloques)
  - la línea vertical "restitución hash power" indica el instante en el que se restablece el *hash power* global original luego de cinco períodos de ajuste durante los cuales la red operó con la mitad del poder de minado.

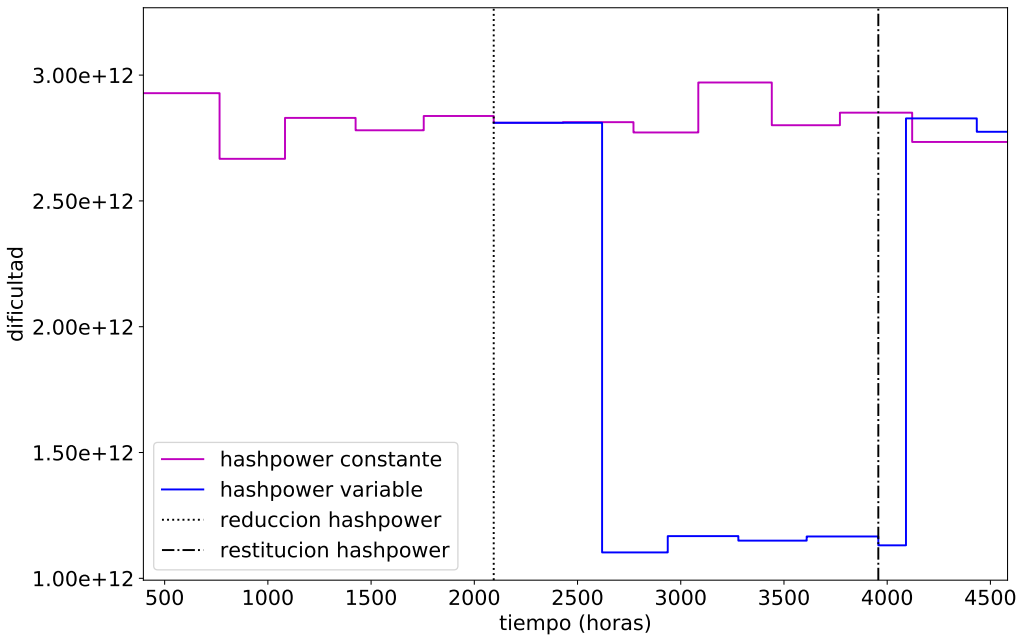


Figura 5.1: *Ajuste de dificultad ante cambios en el* hash power *global*

Análisis de la figura 5.1

Luego de una dificultad inicial un poco mayor a la correspondiente al *hash power global*, la misma converge con un par de períodos de ajuste.

Para el experimento de *hash power global* constante, la dificultad de la red no varía considerablemente durante el experimento, aunque por la naturaleza probabilística de la aparición de bloques siempre es esperable una pequeña diferencia luego de cada período.

Para el experimento de *hash power global* variable, luego de reducir el poder de cómputo a la mitad al cabo del quinto período, el siguiente período tiene una duración de aproximadamente el doble en comparación con el experimento constante.

Esto se debe a que la dificultad todavía sigue siendo la anterior y la red necesita en promedio el doble de tiempo para minar los próximos 2016 bloques. Una vez que esos bloques fueron minados, la dificultad converge rápidamente a un nuevo valor para asegurar la aparición de bloques individuales cada diez minutos.

Luego de cinco períodos, volvemos a restituir el *global hash power* y ahora vemos que el período siguiente de ajuste es muy breve dado que la red opera con un *hash power* efectivo que es el doble del estimado, es decir que los 2016 bloques de dicho período se encuentran mucho antes que las dos semanas ideales en las que debería ocurrir en condiciones normales.

Variación en tiempos de generación de bloques En la figura 5.2 se ilustra como varía el tiempo en encontrar 144 bloques considerando el experimento dinámico en el cual se modifica el *hash power* global.

Dado que el minado es probabilístico y puede haber mucha varianza entre el tiempo de aparición de un bloque y el siguiente, se eligió graficar el tiempo de aparición entre 144 bloques. Dicha cantidad responde al escenario ideal donde, en condiciones normales, la red debería minar un bloque cada diez minutos, lo cual da un total de 144 durante un día.

Con la línea punteada horizontal mostramos el tiempo ideal de 24 horas durante el cual deberían generarse 144 bloques. A su vez, mostramos dos grupos de líneas verticales, para cada uno de ellos primero indicamos el instante en que hubo un ajuste de dificultad e inmediatamente después varió el *hash power* global (por como armamos el experimento), y luego indicamos el ajuste inmediatamente posterior donde la red continuó operando con el mismo *hash power* global del instante previo.

Cabe aclarar que con el fin de mostrar solo la información más importante solo gráficamos los cuatro ajustes de dificultad más relevantes pero hay que considerar que durante el experimento hubieron doce ajustes de dificultad en total.

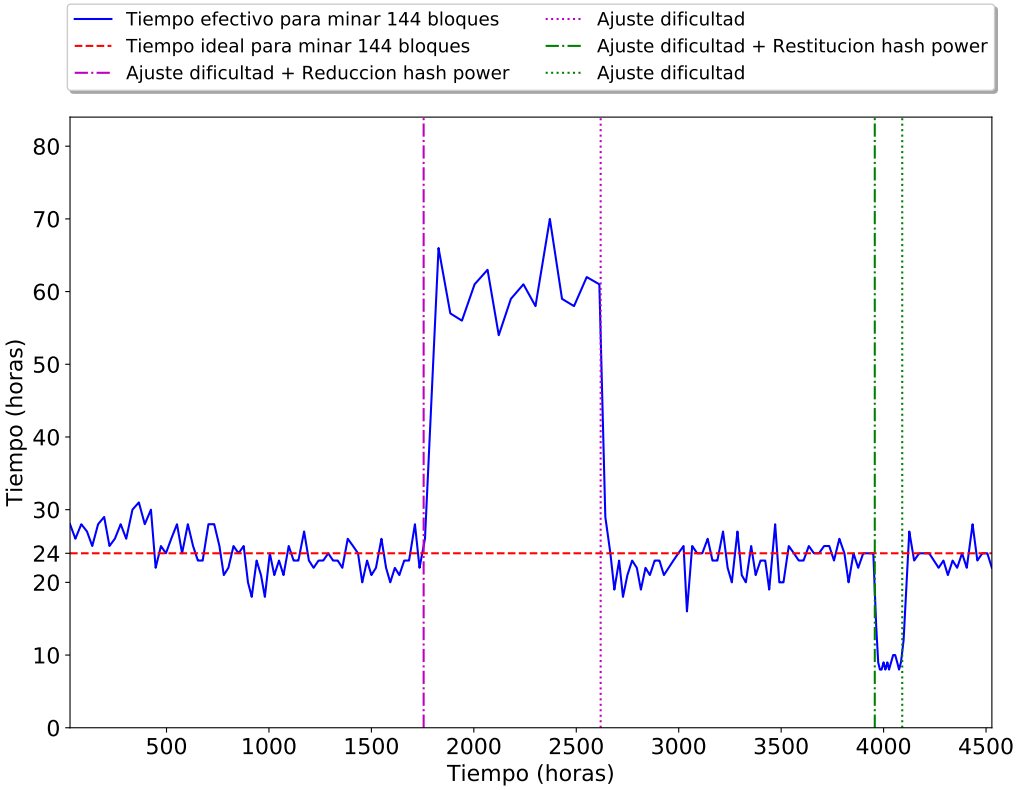


Figura 5.2: *Variación en tiempos de aparición de bloques ante cambios en el* hash power *global*

Análisis de la figura 5.2 Si consideramos aquellos períodos donde la red operó con *hash power global* constante vemos que la aparición de 144 bloques oscila alrededor de las 24 horas ideales. Dicha oscilación la atribuimos a los distintos instantes de ajustes donde la dificultad crece o disminuye para adaptarse a lo que tardaron en encontrarse los 2.016 bloques previos.

Dado que el minado es probabilístico, podría suceder que encontrar 2016 bloques tarde menos de las dos semanas ideales, lo cual redundará en una dificultad mayor para el siguiente período. De igual manera, podría suceder que se tardase más de dos semanas, lo cual haría operar a la red bajo una menor dificultad en el siguiente período.

Observamos que luego del instante "Ajuste de dificultad + Reducción *hash power*" hay un incremento significativo de los tiempos de minado para 144 bloques. Dichos tiempos son un poco superiores a las 48 horas que podrían esperarse de una reducción del $50\,\%$ en el poner de minado y lo atribuimos al período inmediatamente anterior donde los 2016 previos pudieron haberse minado antes que las 2 semanas ideales, lo cual redundó en una mayor dificultad a la ideal para el período que estábamos analizando.

Observamos que luego del reajuste siguiente la red vuelve a oscilar en torno al ideal de 24 horas, esta vez operando con la mitad de poder de minado del inicio del experimento.

Luego del instante "Ajuste de dificultad + Restitución *hash power*" observamos que los tiempos para minar 144 bloques caen a alrededor de 10 horas. Esto se debe a que estamos operando con la misma dificultad pero aumentamos el poder de cómputo 100 % restituyéndolo a los valores iniciales. De manera similar al análisis anterior, el hecho de que durante dicho período la red tarde diez horas en vez de las doce ideales que le "corresponderían" para ese poder de cómputo, lo atribuimos a la dificultad inicial del período la cual en este caso sería un poco menor debido a haber tardado en el período previo más de lo esperado en encontrar los 2016 bloques. Finalmente, observamos que luego del ajuste de dificultad inmediatamente siguiente a haber restituido el poder de minado original la red vuelve a oscilar en torno al tiempo ideal.

El comportamiento descripto es el esperado por el cliente real, el modelo de simulación logra reproducir

el protocolo de ajuste de dificultad cuando este es necesario.

5.3 Validación de protocolos de red y consenso

A continuación, detallamos un experimento en el cual reproducimos una traza real obtenida de la Blockchain de Bitcoin. Los bloques de la traza se eligieron durante una operatoria normal de la red sin *forks* ni congestión, con un promedio de 1600 transacciones por bloque y un tamaño promedio de 900 kB por bloque.

En este experimento queremos validar que nuestro simulador está modelando correctamente los protocolos de red y de consenso de Bitcoin. A dicho fin primero obtuvimos los siguientes datos de una traza real de la Blockchain:

- **transacciones**: el tamaño en bytes y el momento en que fueron publicadas, es decir el instante en que fueron comunicadas desde el nodo que la generó a otros *peers*.
- **bloques**: el momento en que se crearon y las transacciones que se incluyeron en los mismos (es decir: pasaron a ser transacciones confirmadas). Es importante aclarar que no contamos con información de otros bloques competitivos de la red, sino que estamos considerando solo aquellos que terminaron en la *main chain*.

Para conseguir dichos datos no es suficiente inspeccionar la Blockchain ya que en la misma solo aparece el historial de bloques y transacciones incluidas en los mismos, pero no el momento en que dichas transacciones aparecieron en la red. A fin entonces de conseguir dichos instantes, utilizamos una API provista por `Blockcypher`[1] que ofrece información complementaria a la persistida en la Blockchain. Por ejemplo, el instante aproximado en que una transacción fue publicada a la red. Dicha información es aproximada debido a que no se puede tener certeza del **momento exacto** en un nodo compartió la misma a sus *peers*, pero gracias al sistema de nodos geográficamente distribuidos que provee la API anteriormente mencionada, podemos obtener el tiempo más temprano entre los informados por estos nodos testigo.

[1] `https://www.blockcypher.com/dev/bitcoin/`

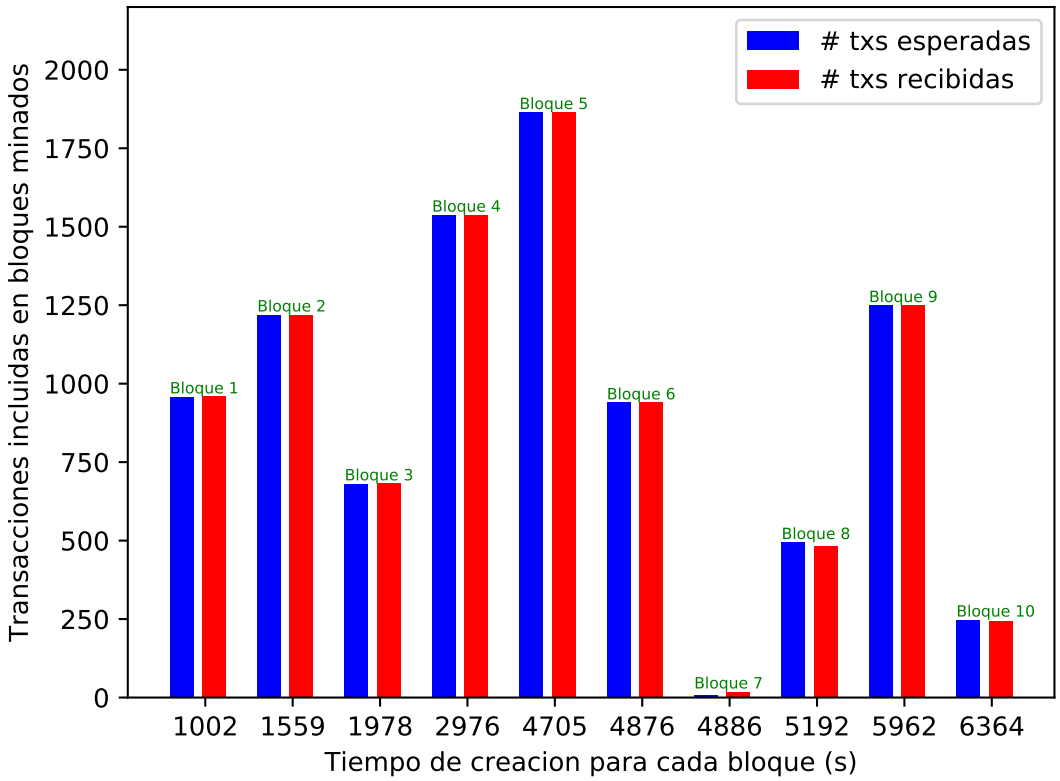


Figura 5.3: *Comparación entre la cantidad de transacciones incluidas en una traza real y su correspondiente simulación*

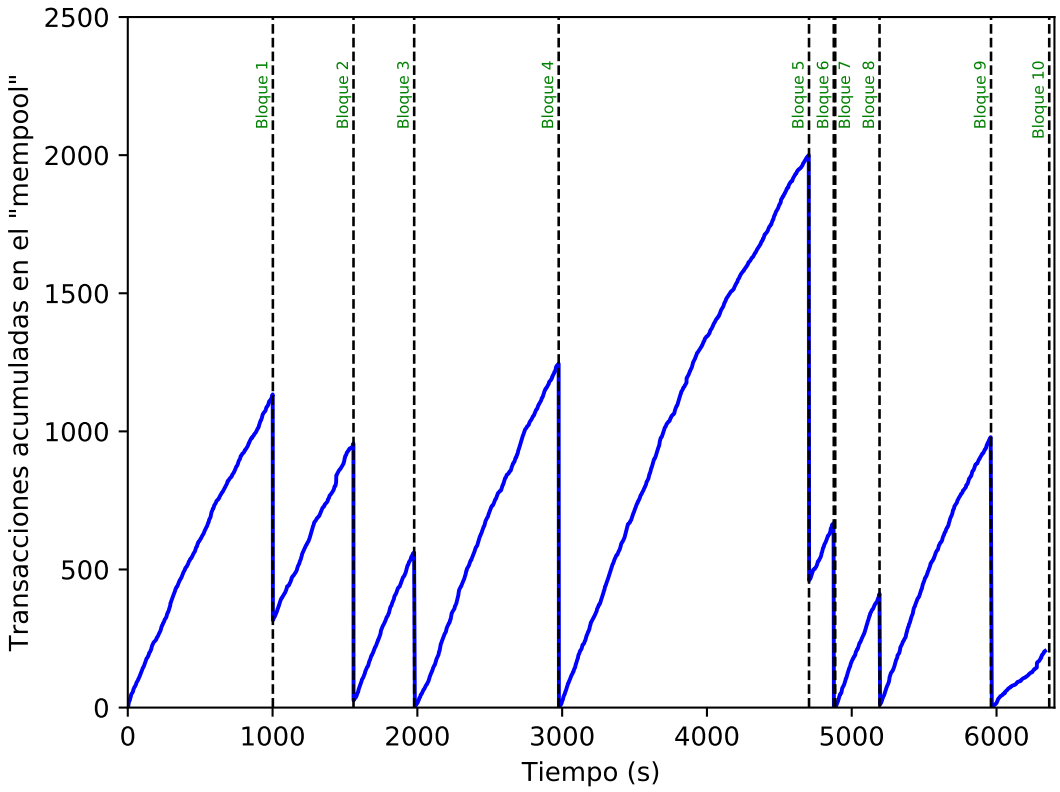


Figura 5.4: *Transacciones acumuladas en el* mempool *durante la simulación de una traza real*

La figura 5.3 ilustra la cantidad de transacciones incluidas en cada bloque, tanto para la simulación como en la traza real. Es esperable que la cantidad de transacciones que sean agregadas por la simulación en cada bloque sea la misma que en la traza real. Se pueden notar sutiles diferencias en algunos casos, por ejemplo con los bloques 6 y 7, las cuales atribuimos a los siguientes factores:

- dichos bloques fueron minados muy próximos en el tiempo (lo cual puede observarse en la figura 5.4) y algunas de las transacciones que en la traza real llegaron a incluirse en el bloque 6, en la simulación lo hicieron en el 7.
- la fecha de publicación de una transacción, como mencionamos previamente, es una aproximación que nos provee la API utilizada, donde en realidad es la fecha en que primero fue detectada por alguno de los nodos testigo de dicho servicio.
- las diferencias entre las topologías física y lógica de la red real en comparación con las simuladas pueden afectar los tiempos de propagación en algunos casos.

En general podemos observar una buena correlación entre las cantidad de transacciones esperadas en cada bloque y las efectivamente incluidas en el mismo.

En la figura 5.4 se grafica la acumulación promedio de transacciones en el *mempool* de cada nodo donde permanecerán hasta ser confirmadas en un bloque.

En el protocolo de Bitcoin las transacciones creadas pueden opcionalmente incluir una comisión para el minero que las incluya en un bloque. Mientras mayor sea esta comisión, mayor será el incentivo del minero para incluirla. Además, el protocolo impone un límite de 1 MB por bloque, con lo cual queda restringida la cantidad de transacciones que un minero puede incluir. En la práctica, la comisión de una transacción será proporcional al tamaño de la misma dado que mientras mayor sea este menor será la cantidad de transacciones de este tipo que el minero pueda incluir en un bloque. Durante períodos de congestión donde los nodos generan más transacciones de las que pueden ser incluidas en un bloque, los mineros racionales van a maximizar su beneficio eligiendo del *mempool* local aquellas que le reporten la mejor relación costo / beneficio. Si bien dichos períodos de congestión ocurren en Bitcoin, para esta validación elegimos una traza correspondiente a un período de baja generación de transacciones. La estrategia utilizada por los mineros para simular el comportamiento durante un período sin congestión consistió en que estos incorporasen directamente aquellas transacciones de las cuales tenían conocimiento en su *mempool* local.

Configuración del experimento	
Topología física	Barabási 1.000 hosts
Topología lógica	Barabási 1.000 nodos
Cantidad de mineros	15 (representando a los *pools* más grandes)
Distribución de *hash power*	Obtenida de `https://www.blockchain.com/pools`
API utilizada	`https://www.blockcypher.com/dev/bitcoin/`

Tabla 5.2: Configuración del experimento para validación de inclusión de transacciones según una traza real

6

Experimentación

En las secciones previas hicimos una introducción de Bitcoin y SimGrid, explicamos el modelo utilizado para la implementación de nuestro simulador de Bitcoin, así como también describimos los experimentos llevados a cabo para validar los aspectos de propagacion de transacciones, minado de bloques y dinamica de ajuste de dificultad, tal como lo hace el cliente de referencia.

La importancia de poder simular una red de Bitcoin a escala real radica en poder analizar el impacto que tienen los cambios en el protocolo (ej: cambios en el *target time*) o en los actores de la red (ej: cambios en la distribución de *hash power* entre los mineros).

En esta sección buscaremos explorar algunos de los escenarios que podemos analizar con el simulador de Bitcoin implementado.

Una posibilidad que nos brinda este simulador es la de analizar en topologías comparables a la red real de Bitcoin el impacto que pueden tener cambios en el protocolo (ej: cambios en el *target time*) o en los actores de la red (ej: cambios en la distribución de *hash power* entre los mineros).

6.1 Diseño experimental

Antes de presentar cada uno de los experimentos destacaremos algunas características comunes a cada uno de los mismos.

Con respecto a la elección del código a simular, se consideró como cliente de referencia a *Bitcoin Core* por ser el más utilizado según `https://coin.dance/nodes`, utilizando la versión 0.14 por ser la más reciente del mismo al comienzo de este análisis.

Para SimGrid se utilizó la versión 3.21 por ser también la última disponible en el momento de realización de este trabajo y, de las distintas APIs provistas por dicho *framework*, se eligió S4U[1]. La utilización de dicha API implica que el código fue escrito en C++.

6.2 Impacto del *hash power* en el minado

El objetivo de este experimento es evaluar qué impacto tiene el *hash power* en el proceso de minado. Para ello vamos a designar a un nodo de la red como "superminero", el cual tendrá un porcentaje del *hash power* global distinto al resto de los mineros. Dicho *hash power* restante será repartido de forma uniforme entre el resto de los mineros.

[1] `http://simgrid.gforge.inria.fr/simgrid/latest/doc/group__s4u__api.html`

Se considerarán ocho escenarios en los cuales el "superminero" designado tenga respectivamente, el $3\,\%, 6\,\%, 11\,\%, 16\,\%, 24\,\%, 33\,\%, 41\,\%$ y $50\,\%$ del *hash power* global, y además serán en base a cuatro *target times* distintos: 75 segundos, $2\frac{1}{2}$ minutos, 5 minutos y 10 minutos.

A su vez, queremos analizar la eficiencia que puede obtener dicho "superminero", no solo operando de manera "justa", que será definido a continuación, sino también tratando de explotar algunos ataques donde se suplanta una cadena que era considerada la *main chain* por otra alternativa producida por este actor. En la sección 2 (Trabajo Relacionado) mencionamos los trabajos de Eyal et al. y Karame et al., que analizan ataques de minado considerando distintos escenarios de *hash power* del atacante.

Replicaremos entonces cada uno de los escenarios, considerando las siguientes tres posibles estrategias a usar por el "superminero":

- **Opera de manera justa**: en esta modalidad el "superminero" va a minar bloques siguiendo el comportamiento esperado, que es el de aceptar bloques con mayor prueba de trabajo y anunciar bloques propios al resto lo antes posible.

- **Retrasa aceptación**: en esta modalidad el "superminero" va a retrasar la aceptación de bloques externos hasta que haya transcurrido la mitad del *target time* o hasta que aparezca un nuevo bloque externo que construya sobre el anterior no aceptado previamente. Lo que buscaría un minero con esta estrategia es tratar de que su bloque sea el que termine imponiéndose en la *Main Chain* y gane sobre aquel bloque externo para el cual retrasó la aceptación.

- ***Selfish Mining***: en esta última modalidad el "superminero" va a retrasar la publicación de los bloques propios, tratando de construir una mejor *Main Chain* en privado e ir publicando los bloques necesarios de la misma con el fin de invalidar bloques competitivos de otros mineros. De esta forma el "superminero" busca forzar al resto a trabajar en vano minando bloques que resultarán huérfanos.

Vamos a describir las características de la red con la que hicimos los escenarios de simulación:

- Tamaño de la red: 5000 nodos.
  Dicho valor fue tomado en función de lograr una simulación representativa de una red real, la cual tiene alrededor de 10000 nodos activos[2]
- Cantidad de nodos mineros: 15.
  Cada uno de estos *pools* de minería representa típicamente a varios mineros individuales que colaboran para encontrar un bloque y repartirse la recompensa proporcionalmente según el *hash power* que haya aportado cada uno al *pool*. Dichos mineros individuales no interactúan con otros nodos de Bitcoin, sino solo con su respectivo *pool*. Dado que nos interesa modelar el comportamiento observable de la red elegimos representar los actores mineros que sean *pools*, usando una distribución de *hash power* entre ellos similar a la existente en la red real[3].
- Cantidad de transacciones: solo las transacciones *coinbase* de cada bloque.
  No modelamos otro tipo de transacciones ya que estamos interesados en evaluar solo la cantidad de bloques por minero que terminan quedando en la *Main Chain*. Además, modelando solo ese tipo de transacciones logramos mayor tiempo simulado en un tiempo real dado, lo que permite obtener un mayor volumen de resultados para analizar.

[2] `https://bitnodes.earn.com/`. Tenemos entonces alrededor de la mitad de nodos y conexiones totales entre nodos en comparación a lo que ocurre en la red real. Consideramos que esto nos permite obtener métricas representativas de carga de mensajes en la red, de latencia en la comunicación y de tiempos de propagación de transacciones y bloques. Evitamos simular los 10000 nodos porque el tiempo de simulación y la memoria requerida restringían la cantidad de experimentos que podíamos llevar a cabo en paralelo.

[3] `https://www.blockchain.com/pools`

- Topología física: utilizamos una red con distribución Barabási.
  Réka y Barabási [AB02] explican que ciertos sistemas como Internet, redes sociales o la WWW son aproximaciones de redes libres de escala y contienen algunos pocos nodos con un alto grado de conectividad. Elegimos la topología Barabási por ser una aproximación a este tipos de sistemas y ser Internet el ámbito donde ocurrirán las conexiones físicas entre los nodos.
- Topología lógica: cada nodo está conectado en promedio a ocho vecinos siguiendo el comportamiento del cliente de referencia.
  Esto lo implementamos por medio de una distribución Watts-Strogatz que resulta en un grafo con las siguientes características:
  - es conexo, al igual que la red Bitcoin donde los nodos tienen el grado de conectividad indicada por parámetro. En este caso, ocho para seguir el comportamiento del cliente de referencia.
  - comparte propiedades con las de un grafo aleatorio pero exhibiendo *clusters* o comunidades. Dichas comunidades parecen producirse en la topología real de Bitcoin según [MJ15].
- Bloques totales: consideramos 1000 bloques en nuestras simulaciones por considerarla suficiente para obtener una distribución entre los *pools* que sea representativa del *hash power* de cada uno y para poder observar situaciones de bifurcaciones donde la red no esté operando en consenso.

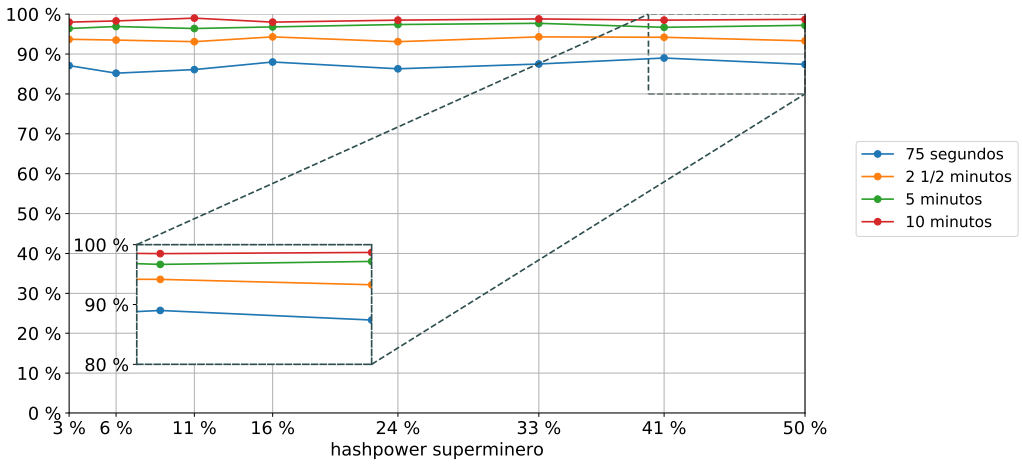
A continuación analizaremos distintas métricas extraídas para este experimento y para cada una de las mismas haremos una comparación según el tipo de estrategia utilizada por el "superminero" y según la proporción de *hash power* que tenga el mismo sobre el global.

6.2.1 Impacto en la proporción de bloques válidos

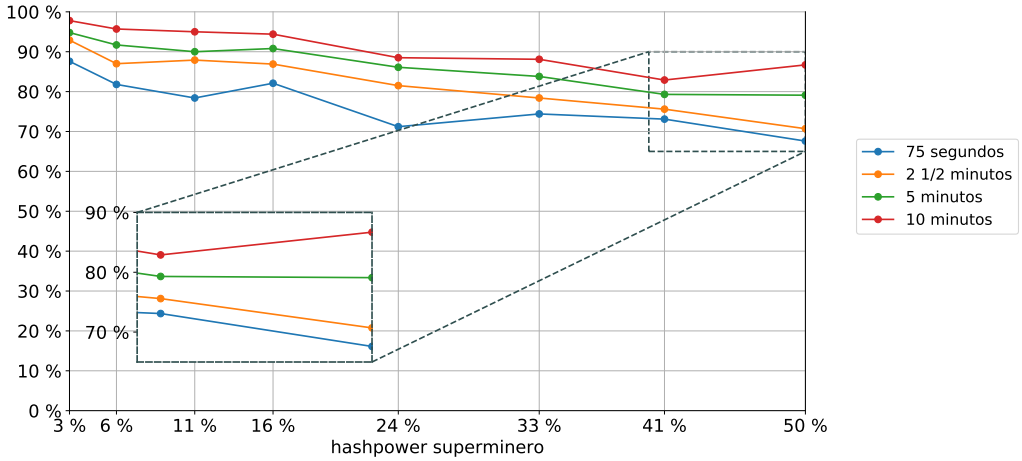
En las figuras que se presentan a continuación mostraremos cómo impacta la presencia de un "superminero" en el porcentaje de bloques válidos sobre los bloques totales minados de la red.

Más formalmente, cada serie representa para cada uno de los *target time* considerados la siguiente expresión:

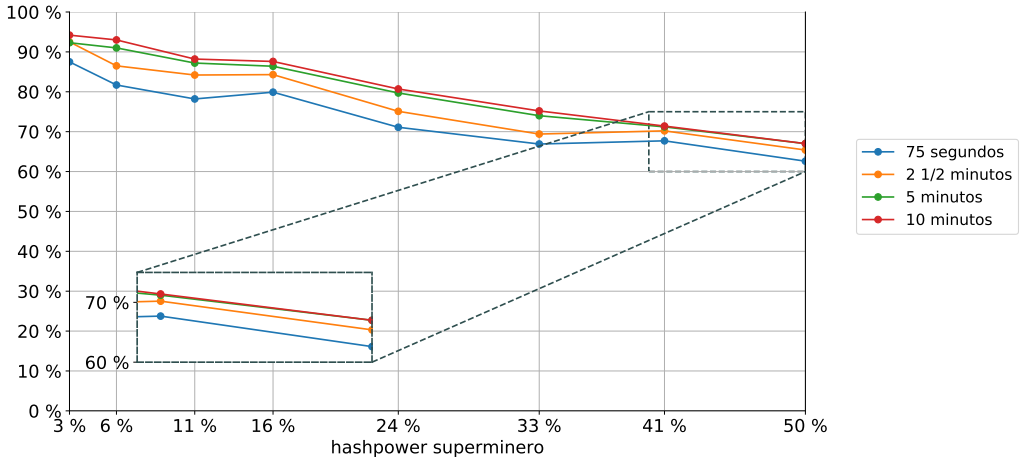
$$\frac{\text{Bloques Válidos}}{\text{Bloques Minados}} = \frac{\sum\limits_{i \in \text{mineros}} BV(i)}{\sum\limits_{i \in \text{mineros}} BM(i)} \tag{6.1}$$



(a) *Opera de manera justa*



(b) *Retrasa aceptación*



(c) Selfish Mining

Figura 6.1: *Variación en la proporción de bloques válidos en la red*

Análisis Cuando el "superminero" opera de manera "justa" **6.1a** no se aprecia un impacto significativo en la red.

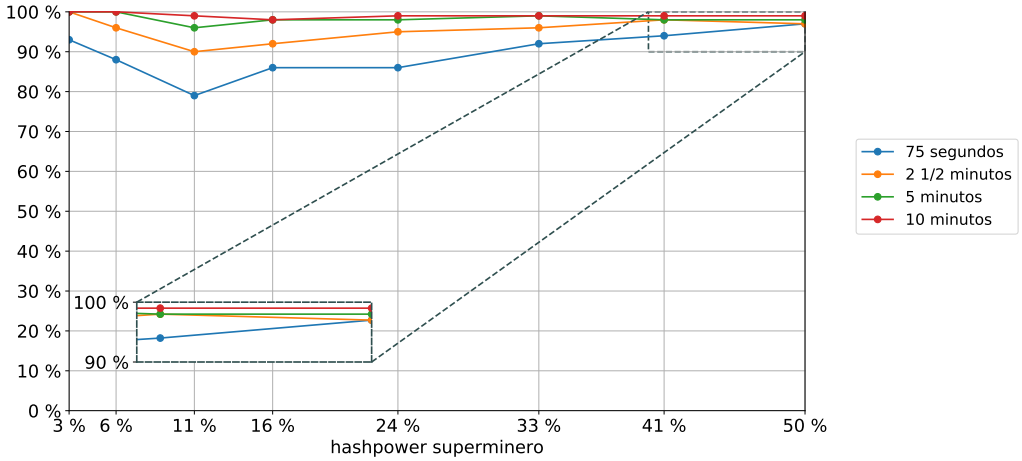
Si evaluamos su incidencia en la red al utilizar una estrategia de retraso en la aceptación de bloques **6.1b** podemos verificar una degradación en el porcentaje de bloques válidos a nivel global a medida que este minero tiene una proporción mayor del *hash power* global.

Al considerar la última estrategia **6.1c** podemos constatar el mismo efecto que en el caso previo pero de manera más apreciable aún, lo cual indica el efecto adverso para la red que trae un actor con alta proporción del *hash power* global al forzar trabajo improductivo en el resto de los mineros.

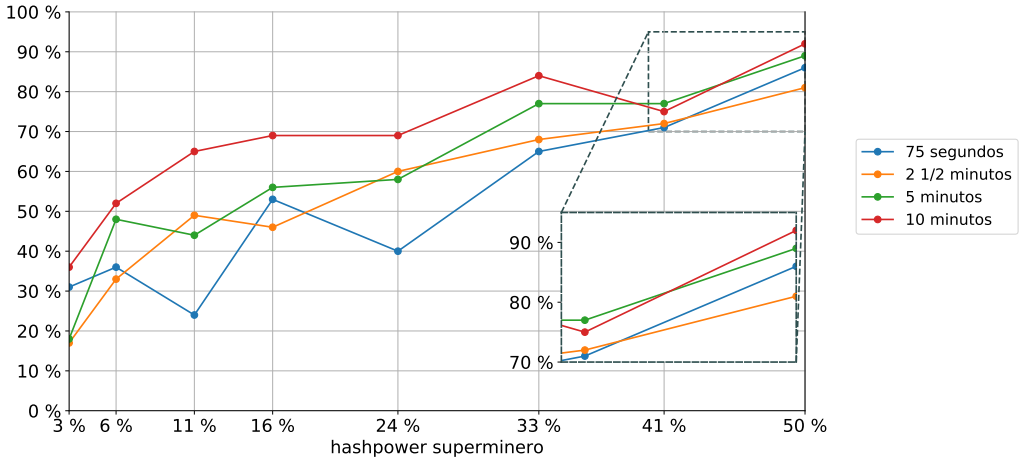
6.2.2 Impacto en la proporción de bloques válidos para el "superminero"

En este caso, repetimos el análisis utilizando la siguiente fórmula que describe la incidencia del "superminero" con respecto a sus propios bloques:

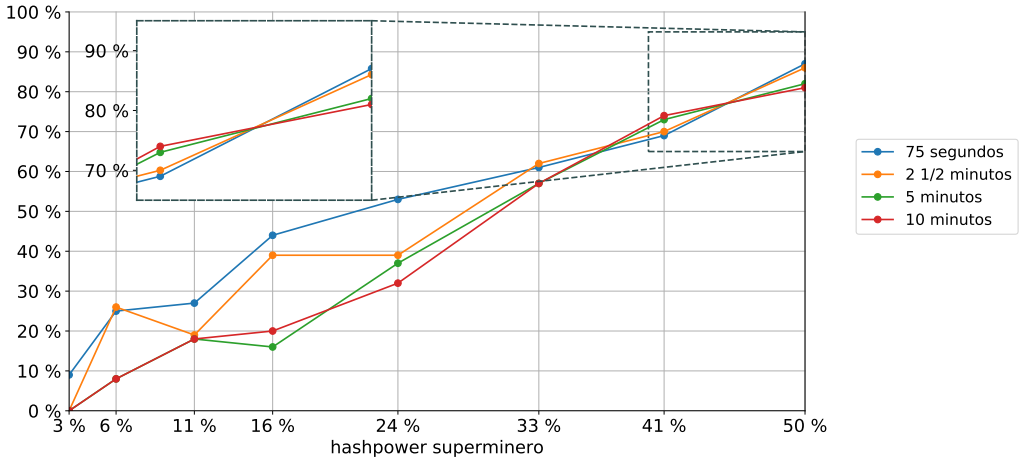
$$\frac{\text{Bloques Válidos Superminero}}{\text{Bloques Minados Superminero}} = \frac{BV(\text{superminero})}{BM(\text{superminero})} \tag{6.2}$$



(a) *Opera de manera justa*



(b) *Retrasa aceptación*



(c) Selfish Mining

Figura 6.2: *Variación en la proporción de bloques válidos considerando los producidos por el superminero*

Análisis Apreciamos que aún operando de manera "justa" en la figura **6.2a**, el "superminero" mejora su proporción de bloques válidos con respecto a los minados a medida que aumenta su *hash power*.

Por otro lado, observamos una degradación muy importante para la estrategia de retraso de bloques incluida en la figura **6.2b** si se cuenta con una baja proporción del poder de cómputo global.

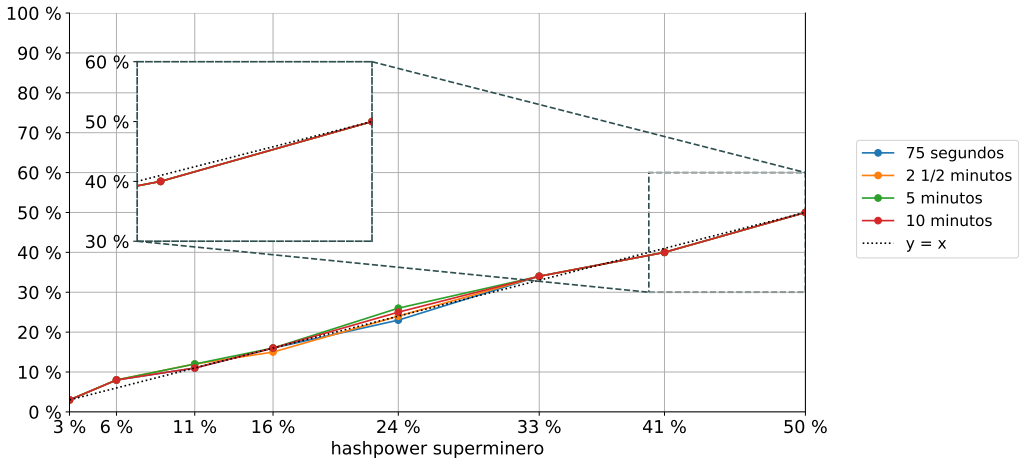
Esta degradación se ve aún más acentuada al utilizar una estrategia de *Selfish Mining* como se presenta en la figura **6.2c**. De esta última observación puede deducirse que dicha estrategia no parece ser recomendable para pequeños *pools* de minería dada la ínfima proporción de bloques que lograrán incorporar a la *Main Chain*.

6.2.3 Impacto del *hash power* en bloques minados

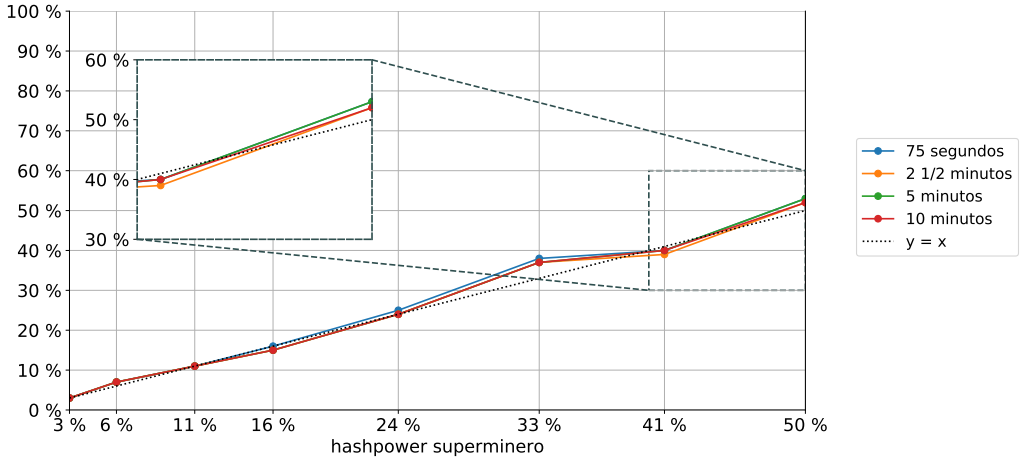
En este caso, repetimos el análisis utilizando la siguiente fórmula que describe la relación de bloques minados por el "superminero" con respecto a la cantidad de bloques minados en total:

$$\frac{\text{Bloques Minados Superminero}}{\text{Bloques Minados}} = \frac{BM(\text{superminero})}{\sum\limits_{i \in \text{mineros}} BM(i)} \tag{6.3}$$

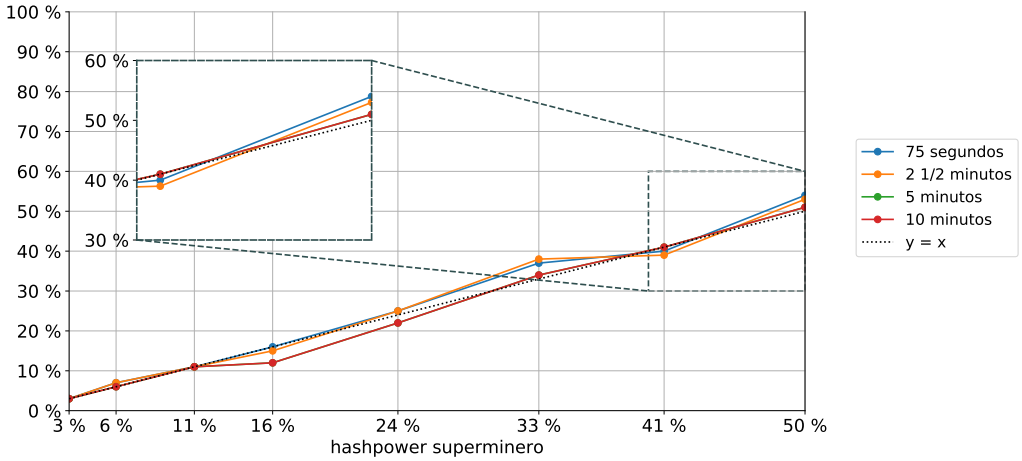
Para mayor claridad incluimos la recta $x = y$ que denota el caso ideal donde un minero crea una cantidad de bloques proporcional a su *hash power*.



(a) *Opera de manera justa*



(b) *Retrasa aceptación*



(c) Selfish Mining

Figura 6.3: *Proporción de bloques producidos producidos por el superminero con respecto al total*

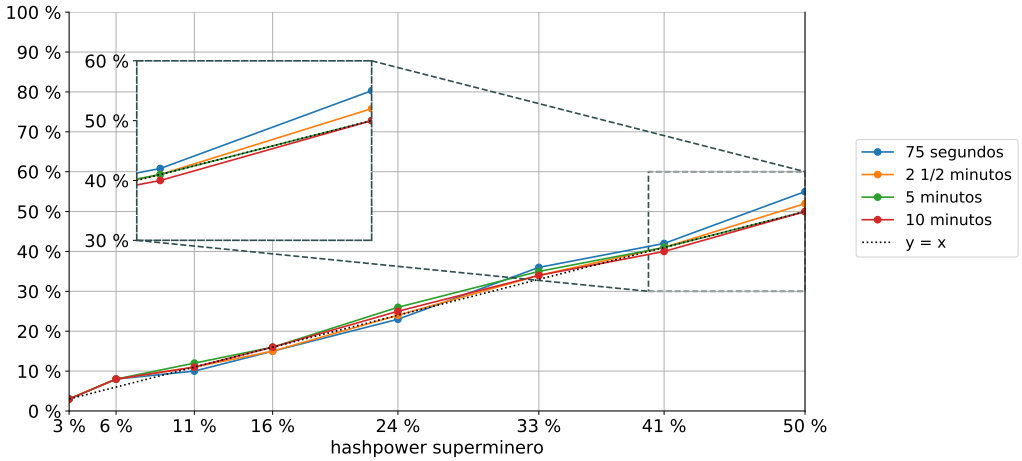
Se puede observar en los distintos casos de la figura 6.3 que, para los distintos *target times*, la cantidad de bloques minados se coincide con el *hash power* del escenario considerado, y esto ocurre así independientemente de la estrategia utilizada. Esto es esperado ya que aún no estamos considerando si dichos bloques minados pasan a formar parte de la *Main Chain* o bien pasan a ser "bloques huérfanos".

6.2.4 Impacto del *hash power* en bloques válidos

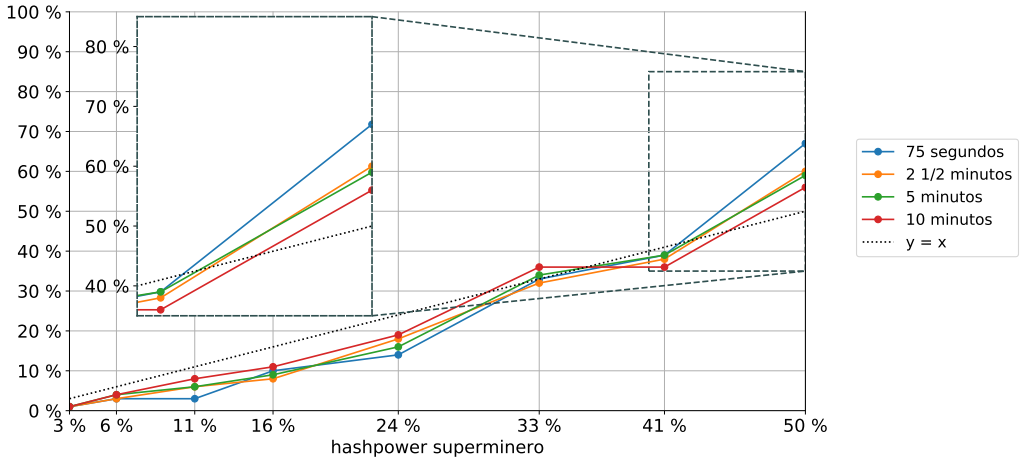
En este caso, repetimos el análisis utilizando la siguiente fórmula que describe la relación de bloques válidos del "superminero" con respecto a la cantidad de bloques válidos en total:

$$\frac{\text{Bloques Válidos Superminero}}{\text{Bloques Válidos}} = \frac{BV(\text{superminero})}{\sum\limits_{i \in \text{ mineros}} BV(i)} \tag{6.4}$$

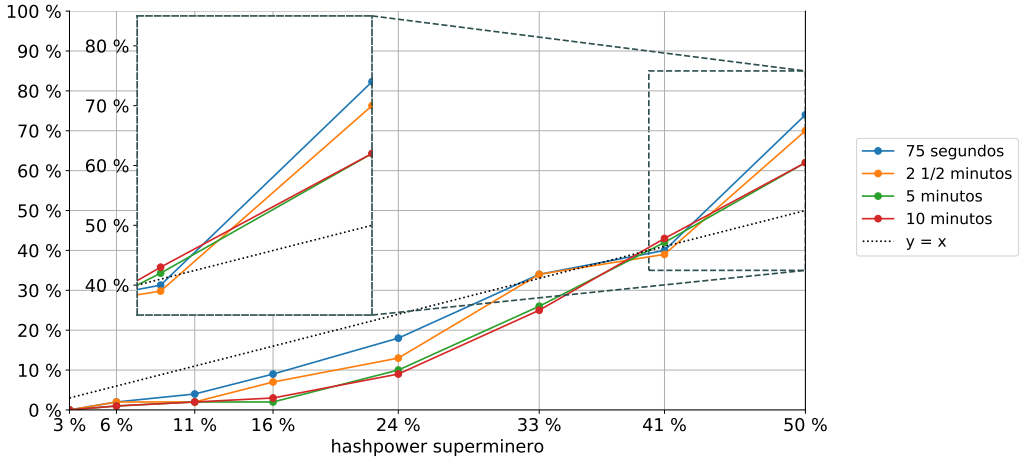
Con este escenario, buscamos evaluar cuál es la participación efectiva del "superminero" en la Blockchain resultante. Para mayor claridad, incluimos la recta $x = y$, que denota el caso ideal donde un minero logra incorporar a la Blockchain una cantidad de bloques proporcional a su *hash power*.



(a) *Opera de manera justa*



(b) *Retrasa aceptación*



(c) Selfish Mining

Figura 6.4: *Proporción de bloques válidos del superminero con respecto a los válidos en la* Blockchain

Análisis

- **Figura 6.4a**: El "superminero" logra incorporar una cantidad de bloques proporcional a su *hash power*, excepto en el último escenario, donde cuenta con el $50\,\%$ de poder de cómputo global y en el cual se aprecia una leve ganancia con respecto al valor esperado de bloques válidos. Esta ganancia es más pronunciada para *target times* menores, debido a una mayor cantidad de *forks* en la red, que son las ventanas de oportunidad para obtener un beneficio/detrimento con respecto al escenario ideal de asignación proporcional por poder de cómputo.
  Es destacable observar que aquí el minero está ganando aún siguiendo de manera "justa" las reglas (no hace abuso de su posición dominante).

- **Figura 6.4b**: Observamos que la estrategia de retrasar la aceptación de bloques externos perjudica al "superminero" en todos los escenarios excepto el último, donde cuenta con la mitad de poder de cómputo global. En dicho escenario termina muy favorecido con respecto a la modalidad de operatoria "justa" analizada previamente.

- **Figura 6.4c**: La estrategia de *Selfish Mining* empieza ofreciendo peor performance que en el caso anterior, pero a diferencia de esta última ya se vuelve competitiva con el 41 % del *hash power* (alcanza

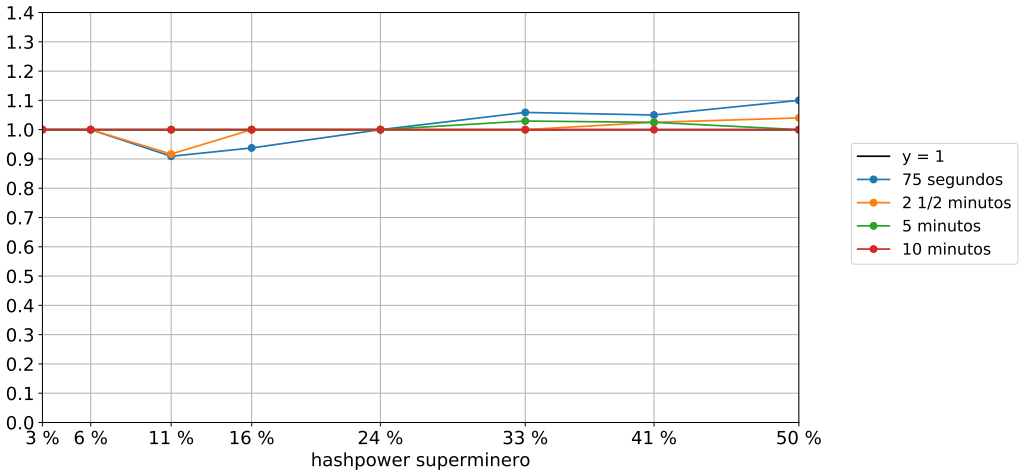
la performance de la modalidad “justa”) y ofrece un mejor beneficio aún en el caso del $50\,\%$ (supera la ganancia obtenida de la modalidad donde se retrasa la aceptación). Estos resultados son consistentes con el análisis realizado por Gobel et al. [GKKT16] sobre ataques de tipo *Selfish Mining*.

6.2.5 Performance de la estrategia utilizada por el “superminero”

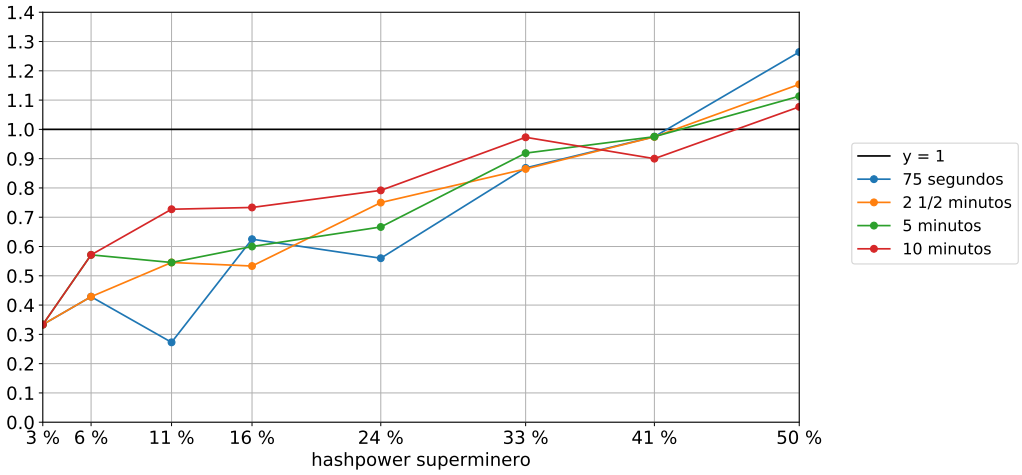
En este caso, repetimos el análisis utilizando la siguiente fórmula que describe el comportamiento observado con respecto al ideal si la Blockchain resultante fuese producto de bloques repartidos proporcionalmente entre los mineros de acuerdo a su *hash power*:

$$\frac{\dfrac{\text{Bloques Válidos Superminero}}{\text{Bloques Válidos}}}{\dfrac{\text{Bloques Minados Superminero}}{\text{Bloques Minados}}} = \frac{\dfrac{BV(\text{superminero})}{\sum\limits_{i \in \text{ mineros}} BV(i)}}{\dfrac{BM(\text{superminero})}{\sum\limits_{i \in \text{ mineros}} BM(i)}} \tag{6.5}$$

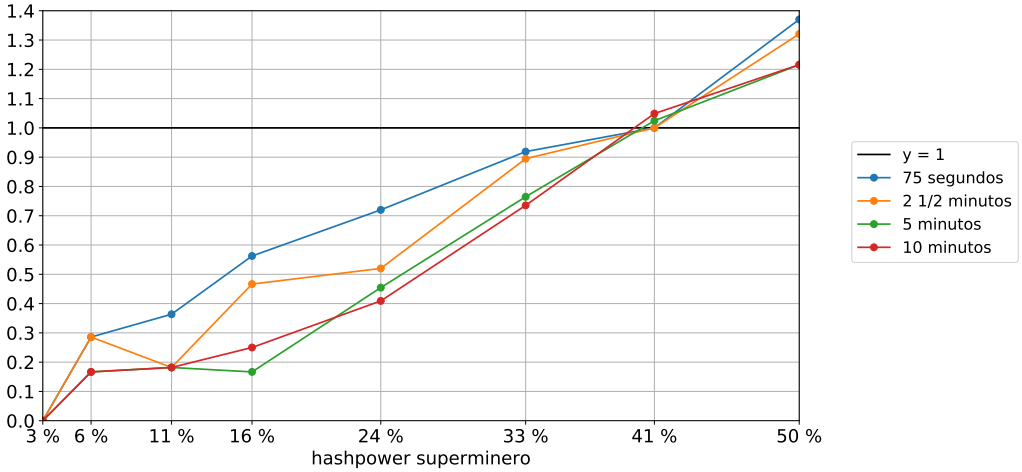
Con esta expresión buscamos reflejar la relación entre dos proporciones. Por una lado, aquella correspondiente a los bloques que el “superminero” logra incluir en la *Main Chain* y, por otro, aquella correspondiente a los bloques producidos por el “superminero” con respecto al total. Para mayor claridad incluimos la serie $y = 1$ que indica el escenario ideal en el que un minero logra incluir bloques exactamente de acuerdo a su *hash power*.



(a) *Opera de manera justa*



(b) *Retrasa aceptación*



(c) Selfish Mining

Figura 6.5: *Comparación entre la variación de bloques válidos con respecto a la variación de bloques totales*

Análisis Al considerar la primera estrategia, mostrada en la figura **6.5a**, podemos observar una oscilación no concluyente del $\pm 10\,\%$ con respecto a la recta $y = 1$.

Si evaluamos las otras estrategias, constatamos un claro deterioro en la performance del minero si el mismo cuenta con una baja proporción del poder de cómputo global. La estrategia de retrasar la aceptación de bloques (presentada en la figura **6.5b**) se torna claramente favorable para el "superminero" en el último tramo cuando posee la mitad del *hash power*.

Por último, un "superminero" que esté siguiendo una modalidad de tipo *Selfish Mining* (figura **6.5c**) puede ver resultados favorables a partir del $41\,\%$ y mejores valores que las otras estrategias al considerar un escenario con el $50\,\%$ del *hash power*.

Los mineros verán los mejores rendimientos utilizando la última estrategia en aquellos casos en los que cuenten una alta proporción del poder de cómputo global, pero de no ser el caso deberían optar en cambio por seguir el comportamiento "justo" como lo muestra la figura **6.5a**.

6.2.6 *Hash power* desperdiciado

El *hash power* desperdiciado representa la proporción de tiempo durante el cual los mineros trabajan en buscar el próximo bloque sobre una versión local desactualizada de la *Blockchain* por alguno de los siguientes motivos:

- Aún no recibieron de sus *peers* información sobre el o los nuevos bloques.
- Recibieron dicha información pero corresponde a bloques competitivos con la misma altura que los propios.
- No recibieron la información porque un minero con posición dominante está haciendo un ataque de *selfish mining* donde especula con la publicación de una cadena que resulte mejor a la actual.

La sumatoria de todos esos tiempos para un cliente dado es el tiempo en el que dicho cliente desperdició su hash power. Llamemos d_i al tiempo que el nodo i pasó desperdiciando su *hash power*, y $h_i \in [0, 1]$ a su proporción de hash power con respecto al total de la red. Entonces, definimos al tiempo total desperdiciado de un experimento x como $D_x = \sum_{i \in \text{mineros}} d_i \cdot h_i$. Luego, dado que el tiempo total en el que el experimento x transcurrió fue T_x, el porcentaje de *wasted hash power* de x se puede calcular como $(D_x/T_x) \cdot 100$.

La figura 6.6 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 *hash power*.

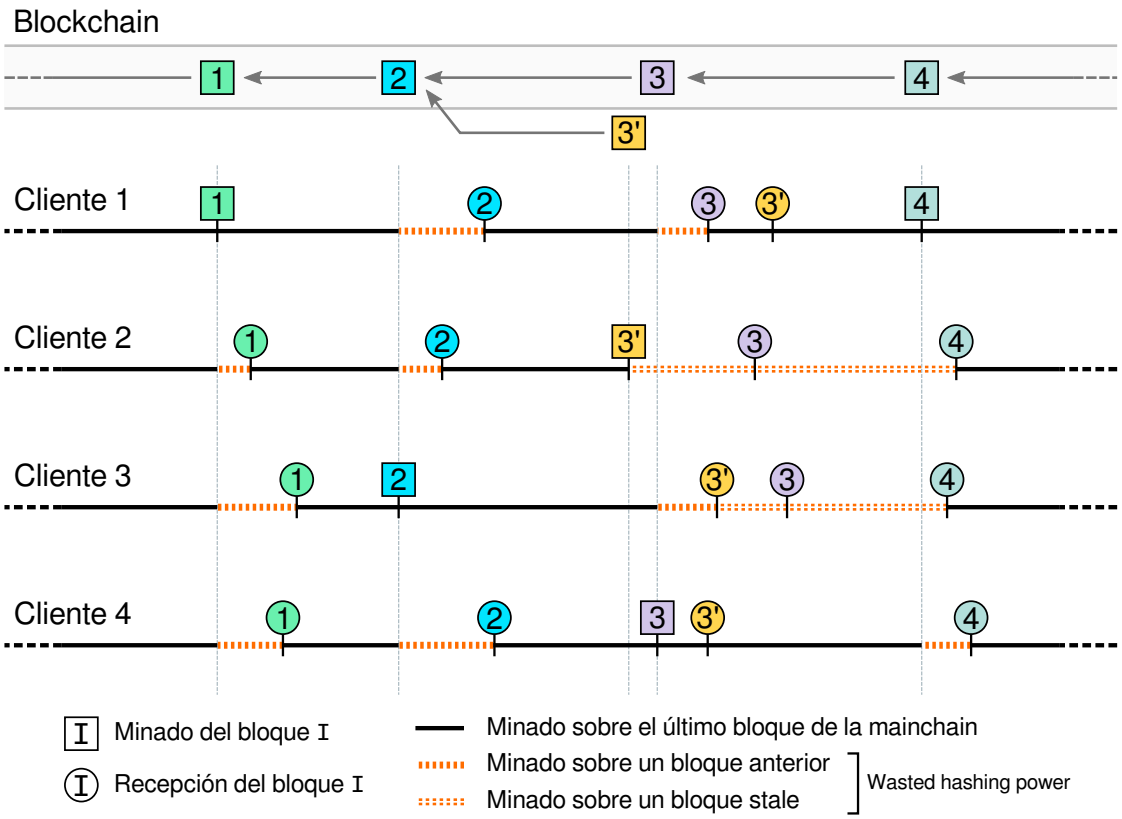
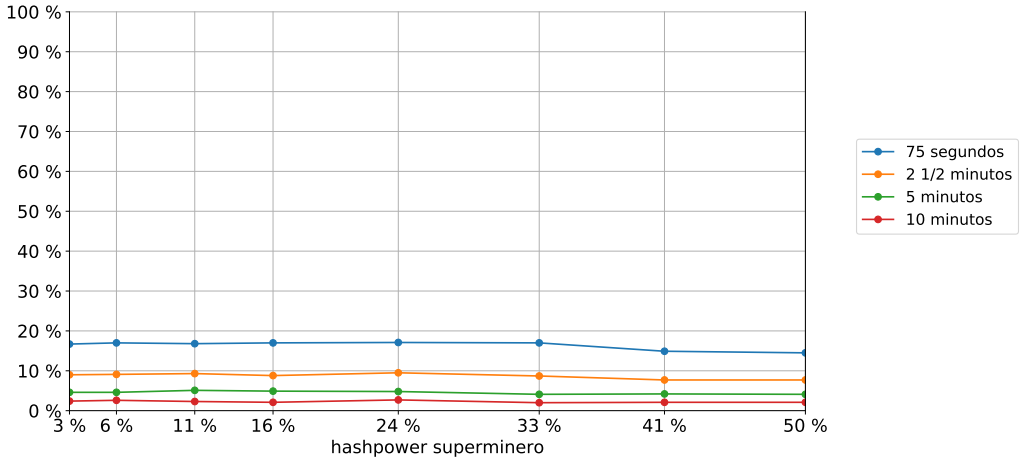
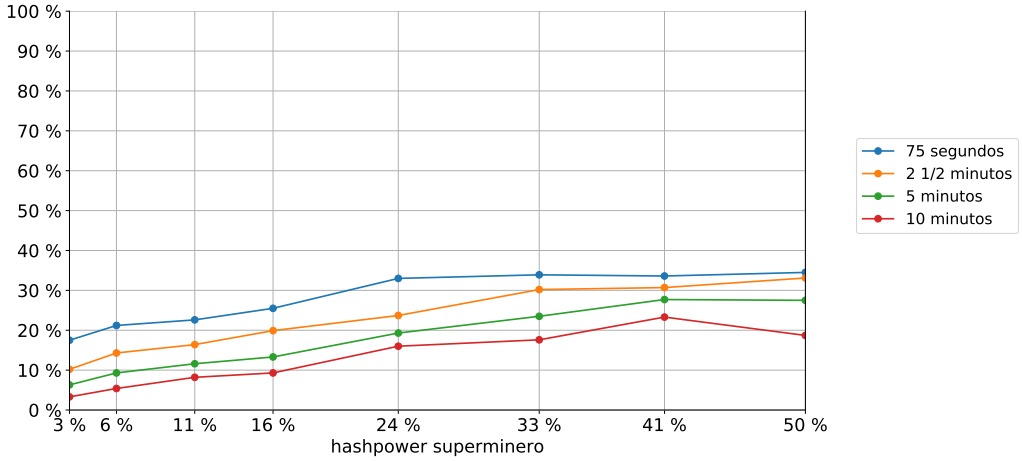


Figura 6.6: *Diagrama de dependencias de variables en el sistema de Bitcoin*

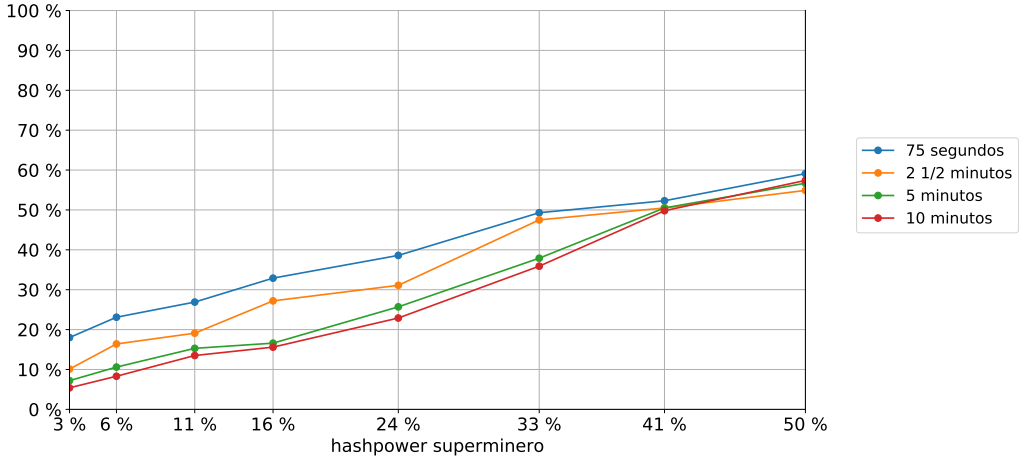
En las distintas figuras incluidas en la 6.7, ilustramos el impacto del *hash power* del "superminero" en relación *hash power* que se desperdicia en la red.



(a) *Opera de manera justa*



(b) *Retrasa aceptación*



(c) Selfish Mining

Figura 6.7: *Variación del* hash power *desperdiciado a nivel global*

Análisis

- **6.7a**: Observamos una leve reducción en el desperdicio de *hash power* a nivel global a medida que el "superminero" cobra más relevancia. Esto lo adjudicamos al beneficio de centralizar la generación de bloques en un actor que opera de manera "justa", lo cual redunda en reducciones de *forks* al momento de establecer la cadena principal, dado que la mitad de los bloques son del mismo nodo (que son conocidos por el mismo de manera inmediata), pero además porque este actor acepta bloques externos si los mismos representan una mejor cadena.

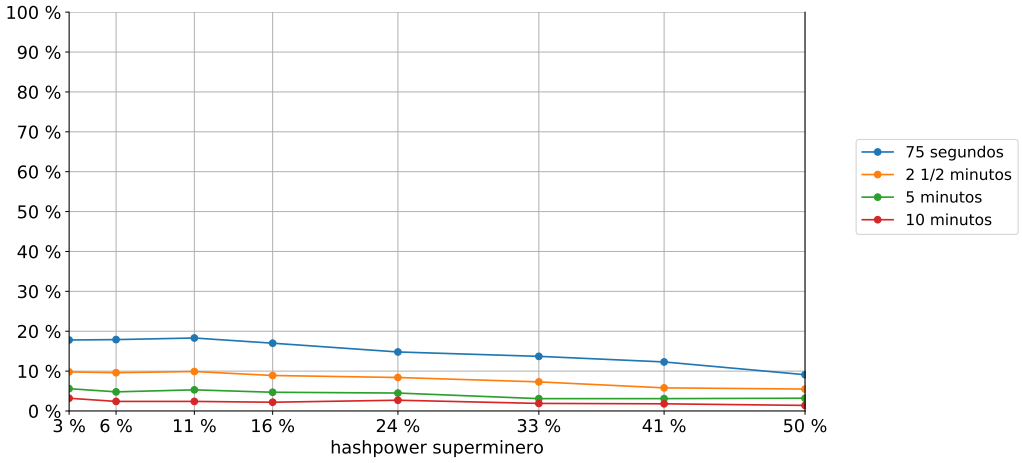
- **6.7b**: Apreciamos un incremento importante en el desperdicio del *hash power* en la primera fase de incremento de la participación del "superminero" para luego estabilizarse. Este comportamiento puede adjudicarse al perjuicio que tiene retrasar la aceptación de bloques externos ya que redunda en mayores *forks*, pero por otro lado el impacto se ve mitigado cuando este actor cobra más relevancia debido a los beneficios de centralizar la red. De todas formas, la red muestra en este caso una peor performance con respecto a la primera metodología.

- **6.7c**: Constatamos un impacto negativo en la red que se incrementado a medida que lo hace la proporción del *hash power* del "superminero" sobre el total. Eso lo atribuimos a la posibilidad cada vez más certera de reescribir la historia con sus propios bloques a medida que tiene un porcentaje mayor del cómputo global. Este escenario tal vez sea el más perverso para un sistema descentralizado porque el mayor beneficio tiene que ir aparejado de un detrimento en la usabilidad de la red (ej: mayor variabilidad en la aparición de bloques válidos, tiempos de confirmación más largos para las transacciones, penalidad para los mineros que operan de manera "justa").

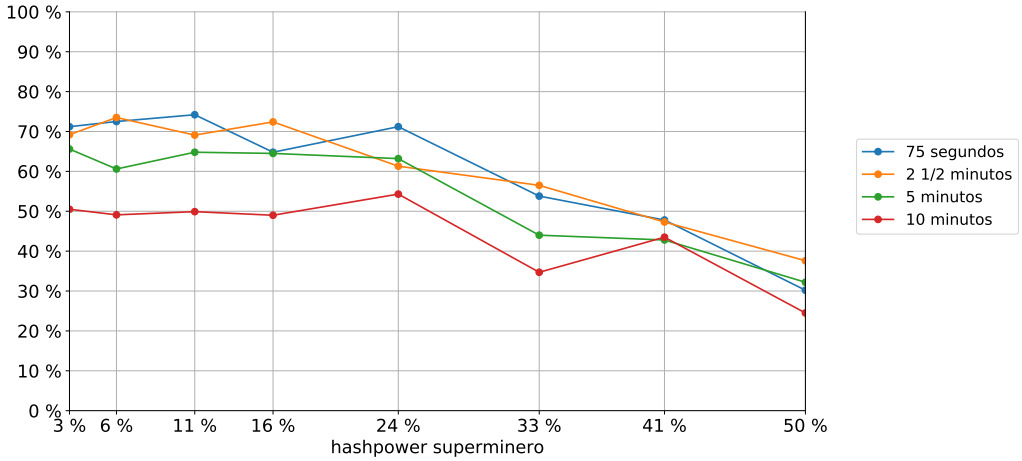
6.2.7 *Hash Power* desperdiciado del "superminero"

En este caso, repetimos el análisis utilizando la siguiente expresión que describe la proporción de tiempo durante el cual el "superminero" hizo trabajo no productivo:

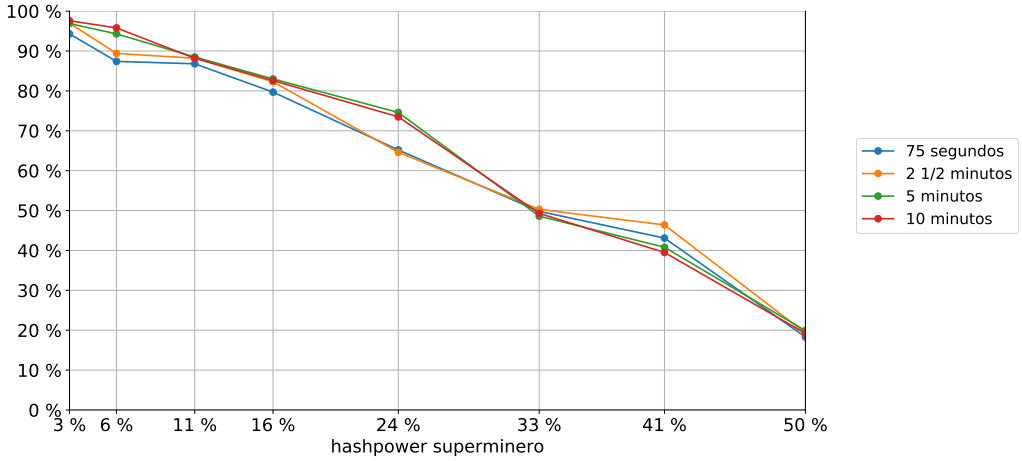
$$\frac{\sum_{i \in \text{intervalos}} WastedHP(\text{superminero}, i)}{\text{Tiempo total}} \tag{6.6}$$



(a) *Opera de manera justa*



(b) *Retrasa aceptación*



(c) Selfish Mining

Figura 6.8: *Variación del* hash power *desperdiciado propio*

Análisis

- **Figura 6.8a**: El "superminero" que opera de manera justa empieza con una proporción de *hash power* desperdiciado similar a la que vimos en la figura 6.7, pero luego la misma se reduce hasta llegar a cifras de un dígito para cualquiera de los *target times* considerados.
- **Figura 6.8b**: La estrategia de retrasar la aceptación de bloques externos resulta muy costosa para un minero con poca proporción del poder de cómputo global. A medida que aumenta dicha proporción, el "superminero" reduce sustancialmente la cantidad de *hash power* desperdiciado, aunque no obstante siguen en una peor posición con respecto al promedio global.
- **Figura 6.8c**: Al utilizar la estrategia de *selfish mining* observamos que es prohibitiva su aplicación para mineros con baja proporción del poder de cómputo global dado que siempre van a estar desperdiciando la totalidad de su *hash power*. Por el contrario, a medida que el poder de cómputo del superminero aumenta, este logra mejorar sustancialmente su situación particular en detrimento del desempeño global de la red. Esto lo atribuimos a que la "apuesta" de ocultar bloques propios empieza a redituar en detrimento del desempeño de todos los otros actores que tienen que desechar trabajo que previamente pertenecía a una mejor cadena.

6.3 Impacto de la conectividad en el minado

Para este experimento buscamos evaluar el posible impacto de la conectividad en el proceso de minado. Para ello vamos a designar a un nodo de la red como "superminero" el cual tendrá una conectividad distinta a la establecida por el cliente de referencia, mientras que los otros mineros sí seguirán dicho comportamiento.

Las características de la red para este experimento son las mismas que las utilizadas para el escenario 6.2 (Impacto del *hash power* en el minado) con las siguientes diferencias:

- *Hash power*: el mismo estará distribuido de manera uniforme entre los 15 *pools* de minería simulados.
- Topología lógica: cada nodo estará conectado en promedio a 8 vecinos siguiendo el comportamiento del cliente de referencia excepto aquel designado como "superminero" para el cual consideraremos distintos escenarios en los cuales estará conectado con 4, 8, 16, 32, 64, 128 y 500 vecinos.

A continuación analizaremos distintos escenarios según la conectividad del "superminero" y, además, en base a cuatro *target times* distintos en los que operó la red: 75 segundos, 2 ½ minutos, 5 minutos y 10 minutos.

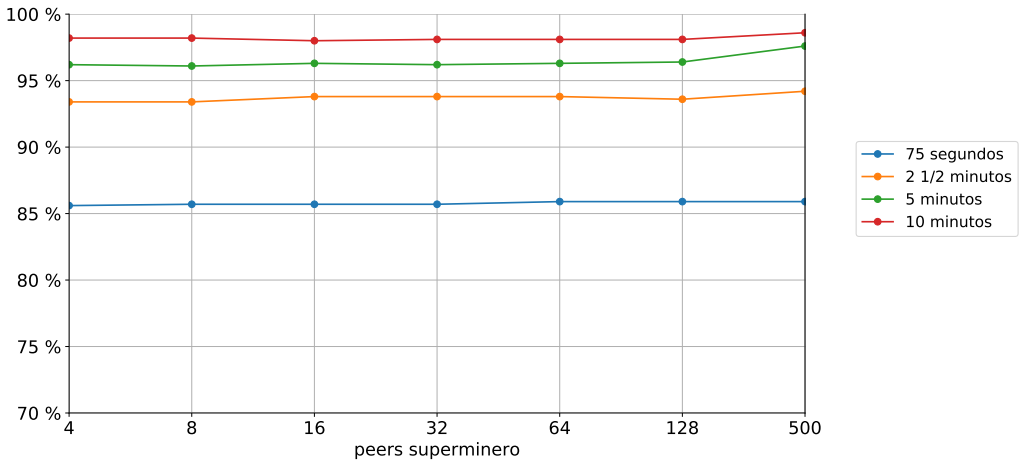
6.3.1 Impacto de la conectividad en bloques válidos

En las siguientes figuras mostraremos la variación en la proporción de bloques válidos con respecto al total de bloques minados, tanto a nivel global (caso **6.9a**) donde la variación estará dada por:

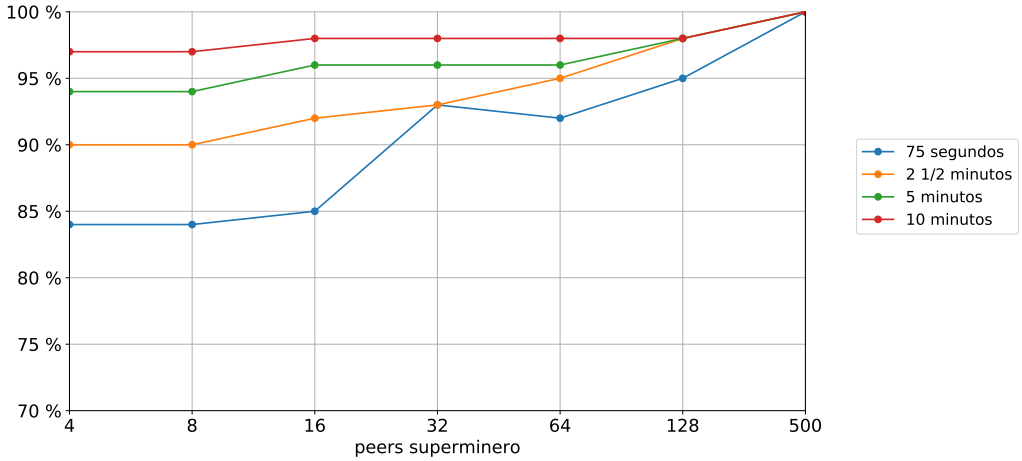
$$\frac{\text{Bloques Válidos}}{\text{Bloques Minados}} = \frac{\sum\limits_{i \in \text{mineros}} BV(i)}{\sum\limits_{i \in \text{mineros}} BM(i)} \tag{6.7}$$

Para el superminero, caso incluido en la figura **6.9b**, la variación estará dada por esta expresión:

$$\frac{\text{Bloques Válidos Superminero}}{\text{Bloques Minados Superminero}} = \frac{BV(\text{superminero})}{BM(\text{superminero})} \tag{6.8}$$



(a) *Proporción de bloques válidos con respecto al total (todos los mineros)*



(b) *Proporción de bloques válidos con respecto al total (superminero)*

Figura 6.9: *Variación en la proporción de bloques válidos*

Análisis

- **Figura 6.9a**: Observamos un comportamiento estable para todas las series correspondientes a los distintos *target times* con una mejora en la proporción de bloques que son considerados válidos cuando existe un "superminero" conectado a 500 *peers*. Creemos que esto se debe al "servicio" que brinda dicho nodo al resto de la red, agilizando la comunicación de nuevos bloques para que sean recibidos por los otros mineros.

- **Figura 6.9b**: Observamos una mejora sistemática en la proporción de bloques que terminan siendo válidos para el "superminero". En los últimos escenarios de 128 y 500 *peers*, el "superminero" pudo agregar prácticamente todos los bloques minados propios a la *Blockchain*. Esto lo relacionamos con la capacidad que tiene este nodo de conocer rápidamente los nuevos bloques externos (y evitar minar sobre una cadena *stale*) así como por su capacidad de informar rápidamente sobre sus propios bloques (y evitar que los mismos se vuelvan huérfanos).

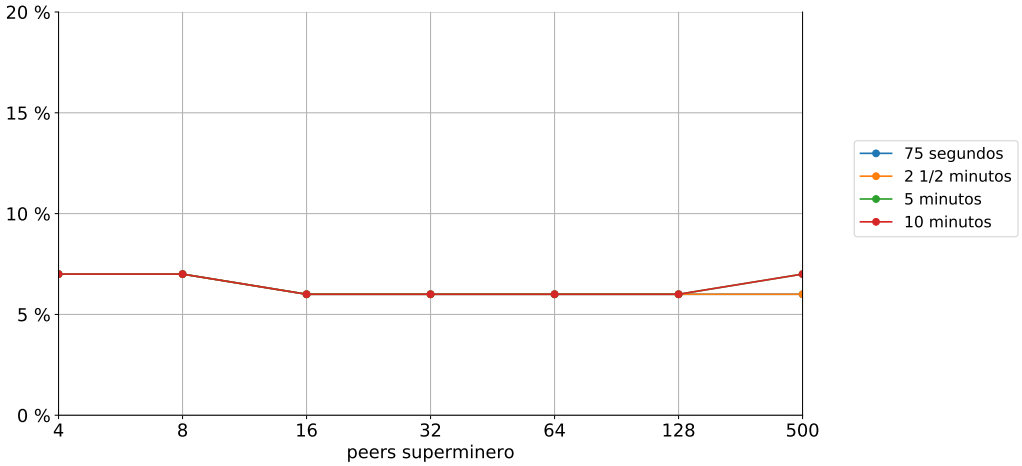
6.3.2 Impacto de la conectividad en bloques del "superminero" con respecto al total

Mostraremos a continuación la variación en la proporción de bloques válidos del "superminero" con respecto al total de válidos global (caso de la figura 6.10a), la cual estará dada por la siguiente expresión:

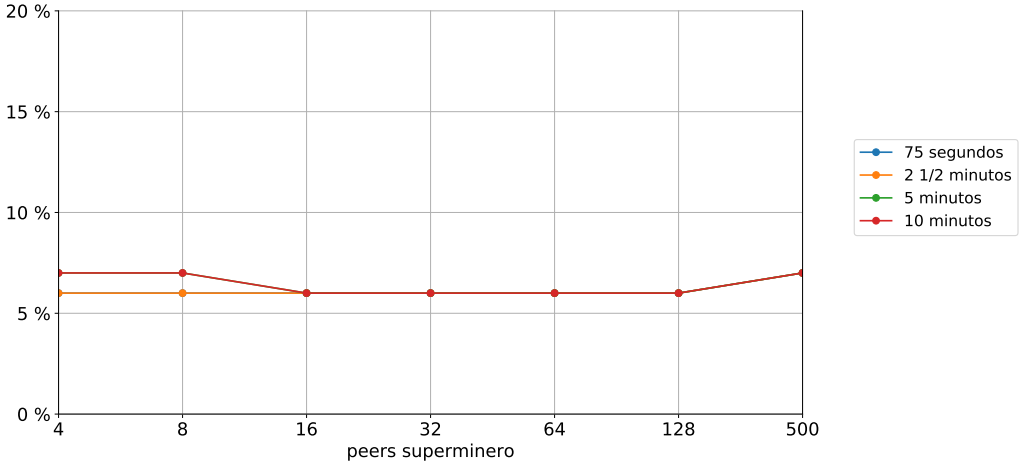
$$\frac{\text{Bloques Válidos Superminero}}{\text{Bloques Válidos}} = \frac{BV(\text{superminero})}{\sum\limits_{i \in \text{mineros}} BV(i)} \tag{6.9}$$

Así como también mostraremos la variación en la proporción de bloques totales del "superminero" con respecto al total de minados global (caso de la figura 6.10b), la cual estará dada por:

$$\frac{\text{Bloques Minados Superminero}}{\text{Bloques Minados}} = \frac{BM(\text{superminero})}{\sum\limits_{i \in \text{mineros}} BM(i)} \tag{6.10}$$



(a) *Proporción de bloques minados por el "superminero" con respecto al total*



(b) *Proporción de bloques válidos del "superminero" con respecto al total de bloques válidos*

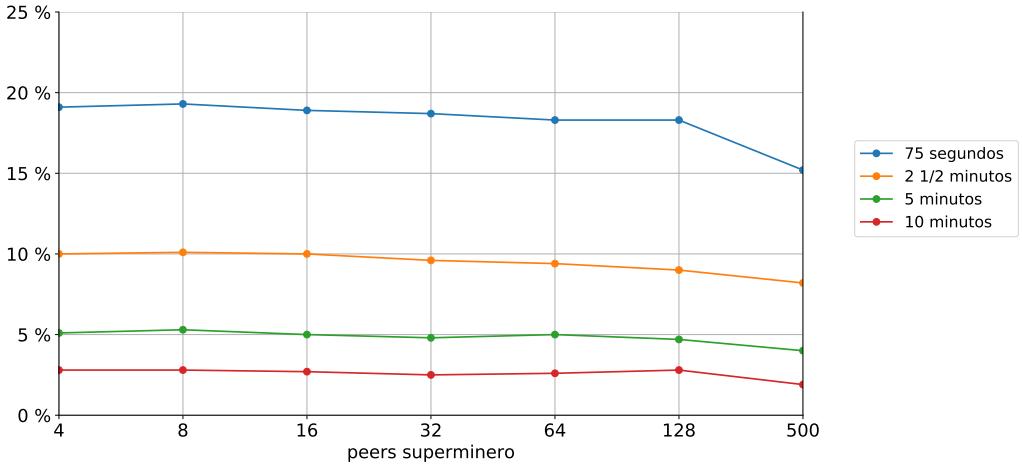
Figura 6.10: *Variación en la proporción de bloques válidos o totales del superminero con respecto al global*

Análisis En ambos casos notamos una tendencia constante lo cual es razonable ya que el "superminero" tiene el mismo *hash power* que el resto de los mineros y además opera de manera "justa", con lo cual es de esperar que su porcentaje de bloques válidos y totales se mantenga parejo con respecto a su *hash power*.

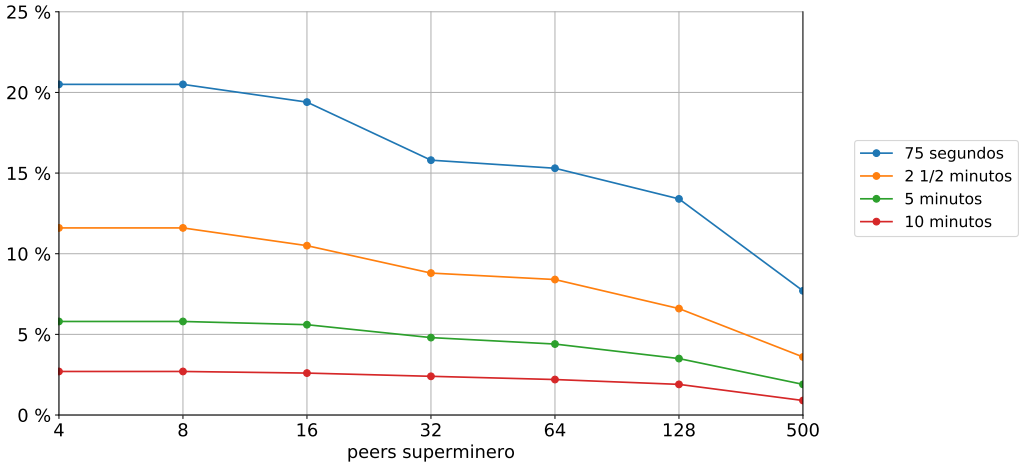
Entendemos que los beneficios observados en el panel 6.9 aquí no tienen mucho impacto debido a la baja incidencia de este actor en el cómputo global ya que solo posee 1/15 del *hash power* total de la red.

6.3.3 *Hash power* desperdiciado

En la figuras que presentaremos a continuación observaremos el impacto de la presencia de un "superminero" tanto a nivel global (caso de la figura **6.11a**) así como el impacto para su caso particular (caso **6.11b**).



(a) Hash power *desperdiciado global*



(b) Hash power *desperdiciado "superminero"*

Figura 6.11: *Variación en el* hash power *desperdiciado global y del "superminero"*

Análisis A nivel global, en la figura **6.11a** solo podemos apreciar una clara incidencia cuando contamos con un "superminero" conectado a 500 *peers*.

Si tenemos en cuenta que hay 5000 nodos en la red simulada, es relevante contar con un nodo que facilite el intercambio de mensajes entre el $10\ \%$ de la red y, si el mismo opera de manera "justa" (como es el caso), se entiende que favorezca al correcto funcionamiento del protocolo. Al considerar solo la situación particular del "superminero" de la figura **6.11b**, podemos observar reducciones sistemáticas en el *hash power* desperdiciado a medida que éste está más conectado.

Dichos resultados manifiestan la importancia para los mineros de contar con una buena conectividad para operar de manera competitiva en la red.

6.4 Impacto de la conectividad en los nodos

En el experimento presentado a continuación evaluaremos el posible impacto que tenga la presencia de "supernodos" en la red en relación al tiempo promedio de recepción de las transacciones.

En Miller et al. [MJ15] indican que, si bien el cliente de referencia está configurado para conectarse con ocho *peers* por defecto, en la práctica se observa la existencia de un reducido conjunto de los clientes conectados a cientos o miles de nodos.

Evaluaremos distintos tamaños de red y para cada uno de los mismos obtendremos el tiempo promedio entre la generación de una transacción y el tiempo de recepción de la misma por el minero que termina confirmándola en un bloque.

Para cada tamaño de la red consideraremos dos escenarios: considerando la existencia de "supernodos", o sea aquellos que están conectados a más de ocho *peers*, y por otro lado los contrastaremos con el escenario donde todos los nodos tendrán la conectividad indicada por el cliente de referencia.

Para el sub-experimento donde consideramos la existencia de "supernodos", evaluamos el promedio de recepción para las transacciones para todos los nodos y además el promedio considerando solo los "supernodos", lo cual permite analizar el impacto (beneficio) de estar más conectado en relación al tiempo promedio esperado para que las transacciones generadas lleguen al minero que termina confirmándolas.

A continuación describiremos las características de la red utilizada para construir los escenarios simulados: Las características de la red para este experimento son las mismas que las utilizadas para el análisis del "Impacto del *hash power* en el minado" con las siguientes diferencias:

- Tamaño de la red: diez escenarios distintos considerando entre 500 y $5{,}000$ nodos en incrementos de 500.
  Se buscó evaluar el impacto de la conectividad a medida que crecía el diámetro de la red hasta una escala comparable con la red real, la cual cuenta con alrededor de 10.000 nodos activos[4].

- Hash power: el mismo será distribuído de manera uniforme entre los 15 *pools* de minería simulados.

- Cantidad de transacciones: 200.000 por día.
  Queremos modelar un caso de uso real y esta cantidad de transacciones es representativa de lo que sucede en la red actual de Bitcoin en un día promedio[5].

- Bloques totales: buscamos simular una hora de la evolución del sistema por lo que consideramos 6 bloques en este experimento por una restricción en el tiempo de uso total disponible de los recursos de cómputo utilizados para correr las simulaciones.

[4] `https://bitnodes.earn.com/`
[5] `https://www.blockchain.com/charts`

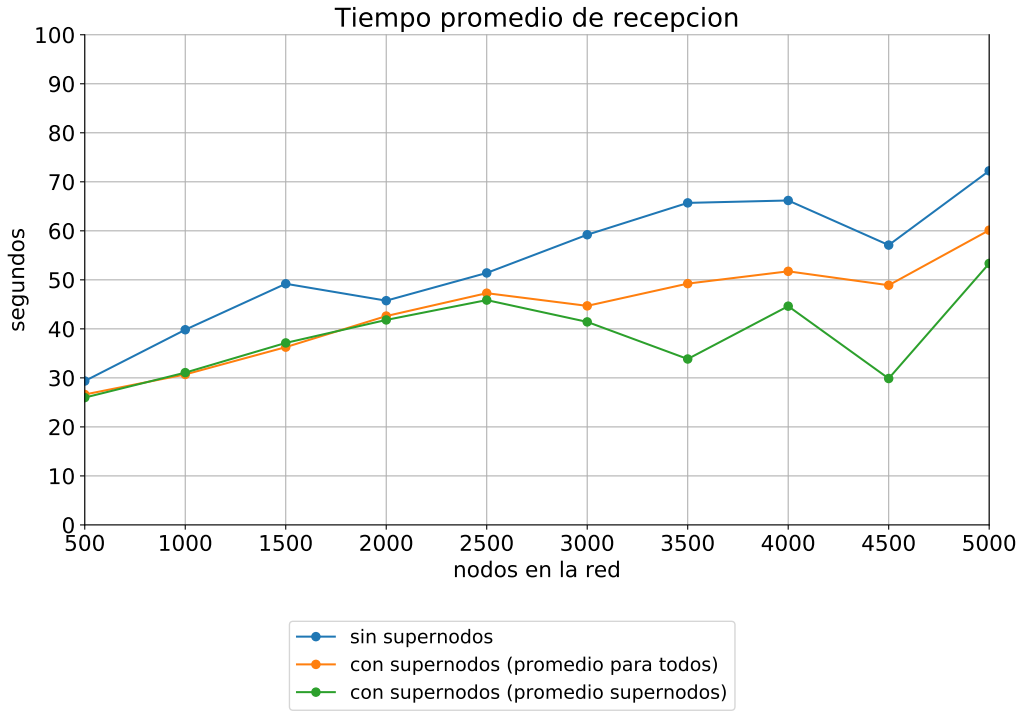


Figura 6.12: *Tiempo de recepción promedio para transacciones para distintos tamaños de la red*

En la figura 6.12 podemos apreciar claramente como crece el tiempo promedio para que una transacción sea recibida por aquel minero que la terminará confirmando al incluirla en un bloque.

También podemos apreciar como, si bien el impacto entre contar o no con “supernodos” no aumenta concluyentemente con el tamaño de la red, sí podemos ver que para todo tamaño dado la red siempre se comportará mejor (menores tiempos de confirmación) al contar con “supernodos” que sin ellos.

Finalmente, cuando solo nos quedamos con las 2 series que representan tiempos de confirmación con “supernodos”, podemos ver como estos últimos terminan obteniendo una leve ventaja a medida que el diámetro de la red se expande. Podemos concluir a partir de estos resultados que los “supernodos”, por el solo hecho de estar más conectados que el resto, logran que sus propias transacciones lleguen antes que el resto a los mineros que terminarán por confirmarlas.

7

Conclusiones

En este trabajo describimos la implementación de un simulador del protocolo Bitcoin que permite generar redes configurando parámetros como la topología deseada, número de nodos, mineros, latencia, ocurrencia de transacciones, *target time*, así como otros específicos al tipo de escenario bajo estudio. Detallamos los aspectos del protocolo que esperamos capturar con dicho simulador, cómo se condice el comportamiento simulado de dichos aspectos con respecto a la red real, y describimos cómo se pueden preparar distintos experimentos y extraer resultados de los mismos.

Al contar con un modelo que captura adecuadamente los aspectos del cliente real y al mismo tiempo nos permite modelar distintas topologías de red, podemos simular la interacción de miles de estos clientes dedicando menos recursos que si ejecutásemos el código real o tuviésemos que replicar en *hardware* la topología real.

Al contar con un simulador que permite recrear redes comparables en su escala a la real, con los consiguientes períodos de bifurcaciones en la *Main Chain*, definimos entonces experimentos donde se exploraron las ventajas que ciertos nodos o mineros podrían obtener en escenarios donde contasen con mayor conectividad o *hash power* que el resto.

Examinamos en dichos experimentos que estas ventajas son observables sin necesidad de cambiar el comportamiento "justo" de cada actor, esto es, sin que ninguno de ellos modifique su accionar en la red para *explotar* sus ventajas comparativas. Pero también observamos como dichas ventajas se vuelvan más significativas cuando estos actores utilizan distintas heurísticas para beneficiarse. También apreciamos que estas estrategias solo reditúan luego de superado un cierto umbral y, en caso contrario, se puede incluso tener un peor desempeño que el del resto de los actores.

Observamos escenarios donde la red no es equitativa para todos los actores que participan en ella. La proporción de bloques efectivos que logran incorporar los mineros no está dada solo por su *hash power* sino que a mayor porcentaje del mismo sobre el total obtienen ganancias mayores a las que corresponderían proporcionalmente. Asimismo, aquellos que están mejor conectados lograr incorporar con más frecuencia sus bloques a la *Main Chain* cuando hay ocurrencia de *forks*. Por último, exhibimos un experimento donde los nodos que no minan también pueden beneficiarse más que el resto al lograr que sus transacciones lleguen antes a los mineros si cuentan con una mejor conectividad que la media.

Durante este trabajo apreciamos el valor del simulador implementado para comprobar como ciertos actores de la red Bitcoin pueden verse beneficiados más que otros pero el alcance de la herramienta desarrollada no solo está acotado a los escenarios evaluados, sino que también podría utilizarse para evaluar otros aspectos del protocolo como mostraremos en la sección 8 (Trabajo Futuro).

8

Trabajo Futuro

En este trabajo hemos mostrado un simulador del cliente de referencia de Bitcoin centrado en modelar el protocolo de consenso e inequidades inherentes que pueden ocurrir entre los actores participantes. Creemos que en trabajos futuros se podría utilizar el mismo para evaluar escenarios no considerados en esta tesis, entre los que mencionamos algunos a continuación.

Saturación de bloques Analizar escenarios de saturación donde por la cantidad de transacciones generadas, las mismas no puedan ser incluidas en los bloques de 1 MB que permite la red. Si dicho análisis lo asociáramos a un modelo de comisión por transacción y analizáramos el tamaño del *mempool* promedio de cada nodo, podríamos evaluar la incidencia que tiene la saturación de los bloques en el costo de comisión por transacción, tiempo de confirmación y tamaño del *mempool*.

Ataques de seguridad Simular ataques de tipo Sybil o Eclipse [1] donde un atacante utilice múltiples nodos con el fin de conectarse a otros actores de la red y proveerles información alternativa sobre la red. Un ataque de este tipo podría ser utilizado para aislar a un minero de la red, proveerle información sobre una *main chain* alternativa y de esta forma lograr que sus bloques queden huérfanos.

Evaluar en más profundidad ataques de tipo *selfish mining*, de manera similar a lo analizado en la sección 6 (6) pero con variaciones en la estrategia a utilizar.

Modelar otros aspectos del cliente real Se podría mejorar el simulador para modelar algunos aspectos de *peer discovery* y evaluar el comportamiento de la red con una topología que cambie durante el transcurso del experimento.

Asimismo, se podrían modelar otros aspectos de una transacción. En el cliente real las transacciones envían bitcoins dividiéndolo entre un conjunto de uno o más *outputs*. Nuevas transacciones eligen como sus *inputs*, uno o más *outputs* de una o más transacciones previas.

Modelar estos aspectos podría permitir evaluar mejores heurísticas con respecto a las transacciones que a un minero le conviene incluir en un bloque. Por ejemplo, una heurística simple podría ser la de incluir aquellas que tengan la mejor relación comisión / tamaño en bytes, pero en la práctica puede convenir incluir transacciones con una peor relación si la misma referencia como *outputs* los *inputs* de otras transacciones que están esperando a ser confirmadas y ofrecen buenas comisiones.

[1] `https://medium.com/chainrift-research/bitcoins-attack-vectors-sybil-eclipse-attacks-d1b6679963e5`

9

Anexo

9.1 Impacto del *target-time* en el *orphan-rate* de los bloques

En 2017 Silvio Vileriño presentó en su tesis "Estudio de los límites de generación de bloques en blockchain" [Vn17] un experimento donde analizaba la propagación de bloques y porcentaje de bloques huérfanos (ie: que no terminaban en la cadena principal) en una red emulada de Ethereum.

Dado que contamos con dichos resultados, nos pareció valioso hacer una comparación utilizando nuestro simulador de Bitcoin dado que si bien son protocolos distintos, ambos mantienen igualmente los conceptos de transacciones y bloques así como la utilización de una Blockchain para la validación de transacciones y lograr consenso distribuido entre nodos.

9.1.1 Configuración del experimento

Se consideró una topología de red y escenarios idénticos a los utilizados en la tesis de Silvio Vileriño.

Para todos los escenarios se utilizó una misma topología lógica base generada aleatoriamente de 60 nodos. Cada nodo tiene un país asignado y 2 o 3 *peers*.

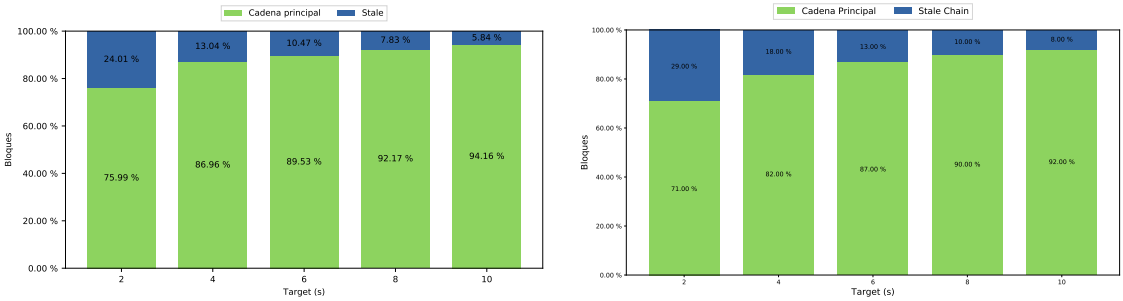
Para construir dicha topología aleatoria se generó un grafo conexo aleatorio donde cada nodo tiene entre i y j vecinos. Luego se le asigna a cada nodo un país de forma aleatoria siguiendo la distribución del trabajo de Donet et al. [DPSHJ14] y para finalizar se asignan la latencia entre enlaces de países según los datos que extraídos del trabajo de Miller et al. [MJ15].

Un aspecto importante a mencionar sobre el trabajo de Vileriño fue la limitación en la escala de los experimentos que era factible realizar debido al hardware y herramientas disponibles así como a la naturaleza de la experimentación emulando Ethereum en vez de simulándolo. Dicha emulación implicaba la utilización de clientes reales de Ethereum con los recursos de procesador, red y memoria que ello implica. Teniendo en cuenta lo anterior, para poder caracterizar redes de mayor diámetro se consideraron escenarios sobre varios conjuntos de latencias: Latencia real definida por los datos previamente mencionamos, Latencia 2X que duplica los valores anteriores, y finalmente Latencia 200 ms. que satura todos los links a una latencia de 200 ms.

Para los distintos escenarios de este experimento, la asignación de mineros en la topología lógica fue una elección aleatoria del 50 % de los nodos de la red, todos ellos con igual *hash power*. Se ejecutaron experimentos variando el tiempo entre bloques entre {2, 4, 6, 8, 10} segundos y para el análisis de todos los experimentos se consideran los primeros 1.000 bloques de la red.

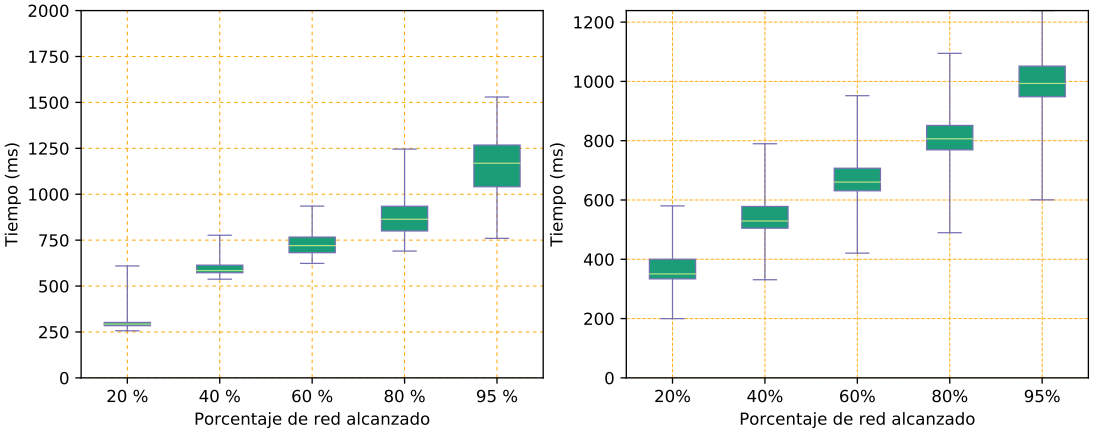
En las próximas secciones presentamos los resultados para las diferentes combinaciones de latencias y tiempo entre bloques.

9.1.2 Latencia Real

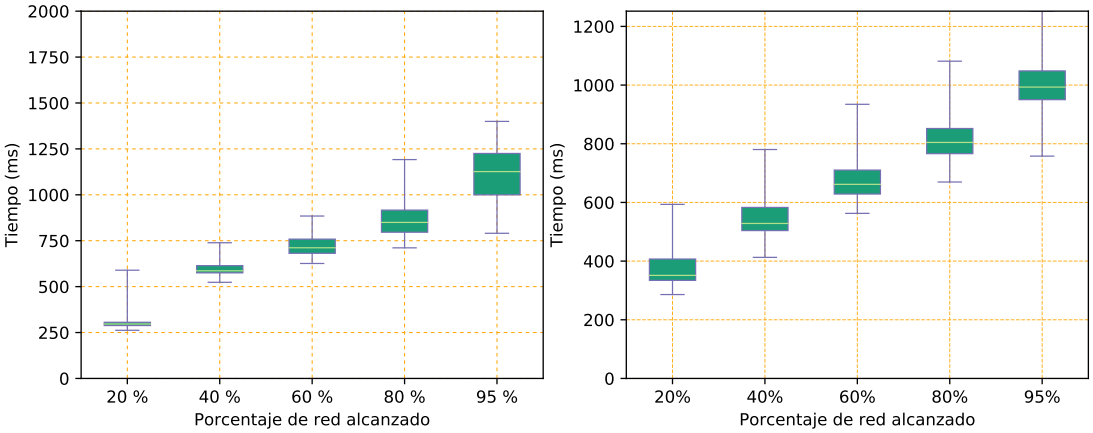


(a) *Blockchain Ethereum Emulada* **(b)** *Blockchain Bitcoin Simulada*

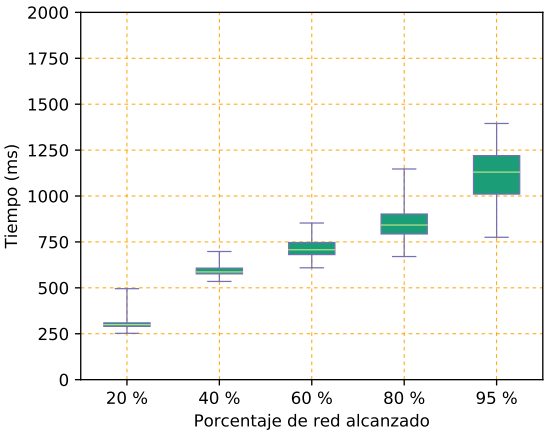
Figura 9.1: *Rendimiento de la blockchain - Topología emulada con latencias reales.*



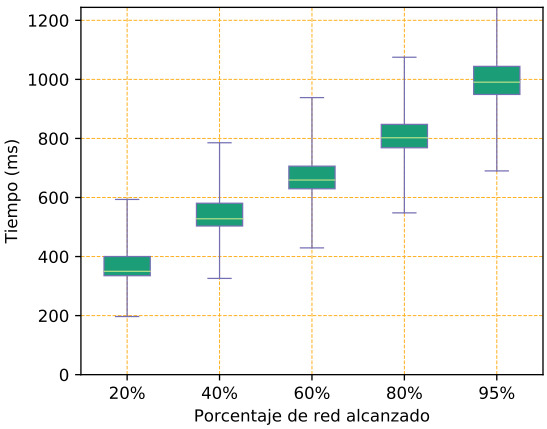
(a) *Target 2s - Blockchain Emulada* **(b)** *Target 2s - Blockchain Simulada*



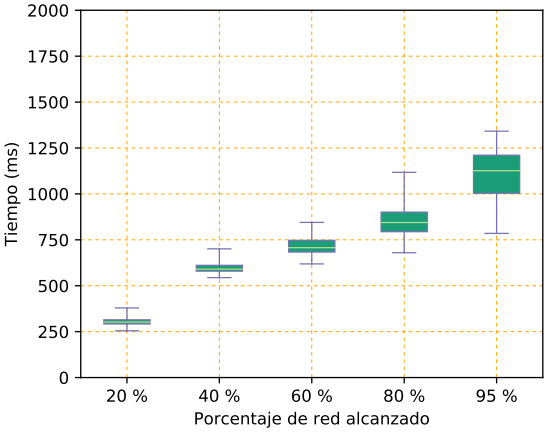
(a) *Target 4s - Blockchain Emulada* **(b)** *Target 4s - Blockchain Simulada*



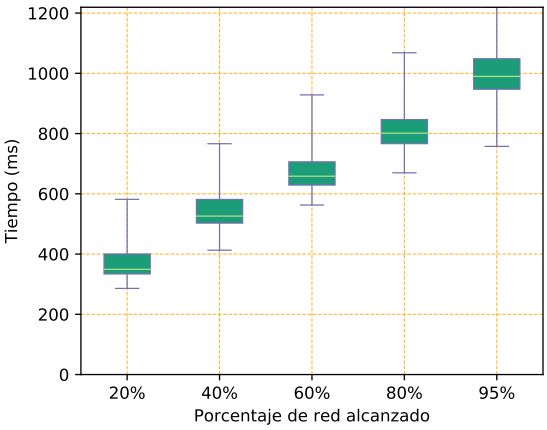
(a) *Target 6s - Blockchain Emulada*



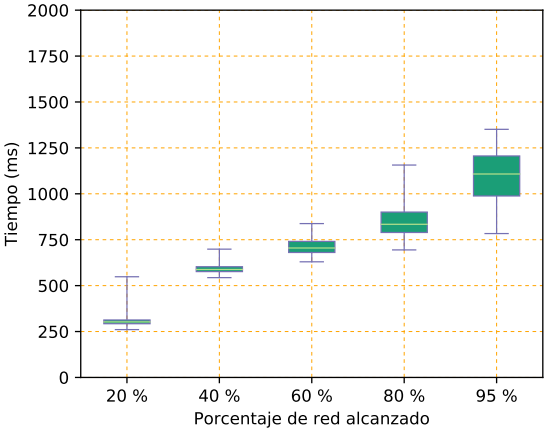
(b) *Target 6s - Blockchain Simulada*



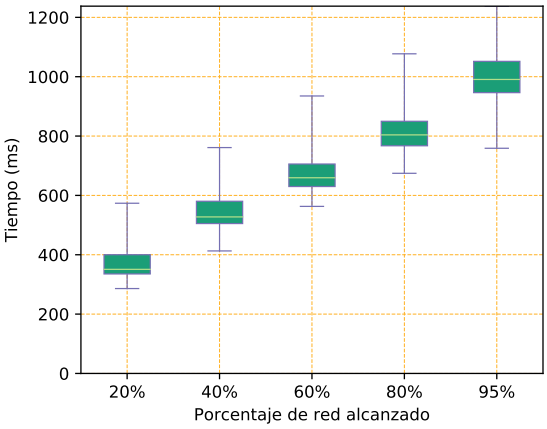
(a) *Target 8s - Blockchain Emulada*



(b) *Target 8s - Blockchain Simulada*



(a) *Target 10s - Blockchain Emulada*



(b) *Target 10s - Blockchain Simulada*

Cantidad de Bloques Competitivos - Emulación Ethreum					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	>4 bloques
10	94,10 %	5,70 %	0,20 %	0,00 %	0,00 %
8	91,70 %	8,20 %	0,10 %	0,00 %	0,00 %
6	89,00 %	10,40 %	0,60 %	0,00 %	0,00 %
4	86,10 %	12,90 %	1,00 %	0,00 %	0,00 %
2	72,20 %	24,50 %	2,90 %	0,40 %	0,00 %

Cantidad de Bloques Competitivos - Simulación Bitcoin					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	>4 bloques
10	89,67 %	10,05 %	0,28 %	0,00 %	0,00 %
8	87,71 %	12,01 %	0,28 %	0,00 %	0,00 %
6	82,91 %	16,82 %	0,27 %	0,00 %	0,00 %
4	74,74 %	23,69 %	1,57 %	0,00 %	0,00 %
2	58,95 %	33,78 %	6,33 %	0,94 %	0,00 %

Tabla 9.1: Estadísticas de ancho de forks.

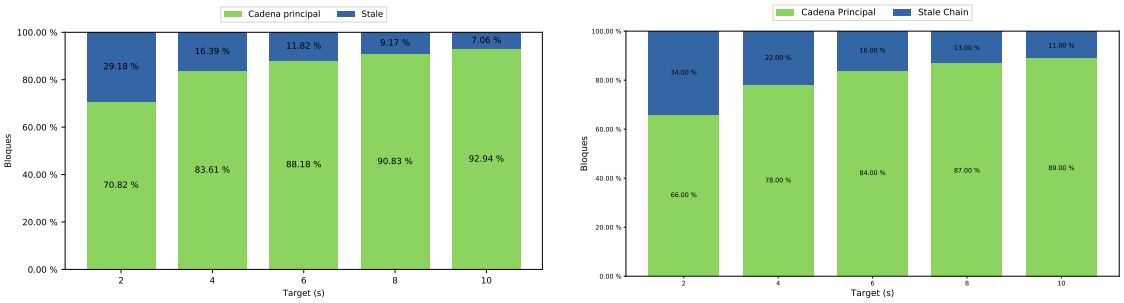
Profundidad de forks - Emulación Ethereum				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	57	2	0	0
8	84	0	0	0
6	102	7	0	0
4	127	11	0	0
2	234	31	6	1

Profundidad de forks - Simulación Bitcoin				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	53	1	0	0
8	64	1	0	0
6	86	4	0	0
4	139	5	0	0
2	230	22	2	0

Confiabilidad
Alta
Media
Baja

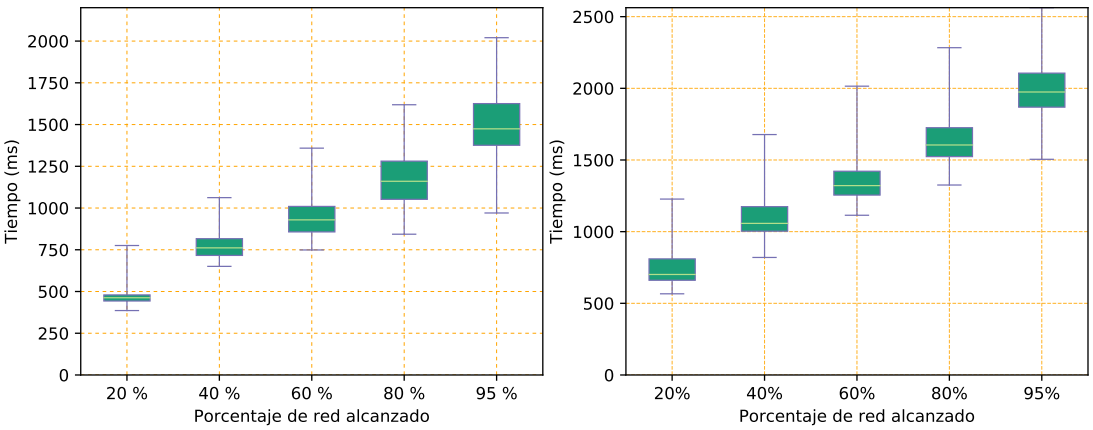
Tabla 9.2: Estadísticas de profundidad forks - Mapa de color de la confiabilidad de la Blockchain.

9.1.3 Latencia Duplicada: real x 2

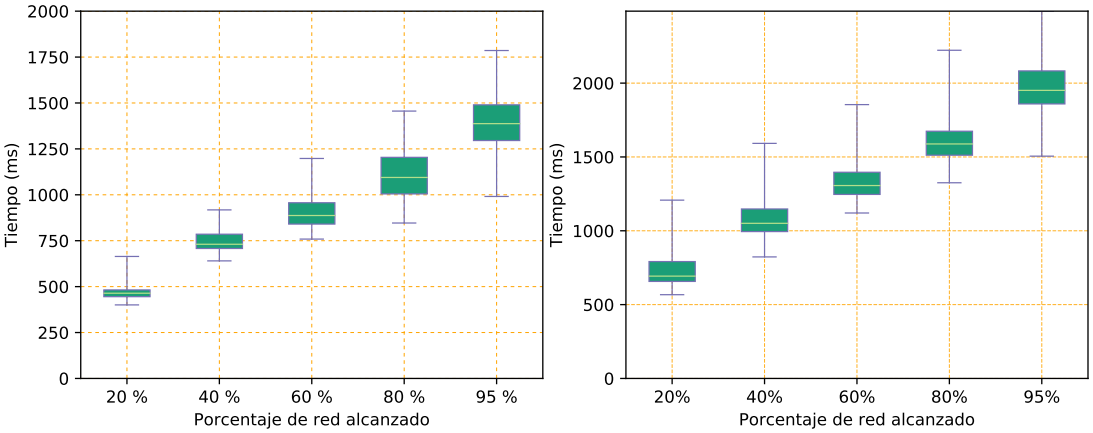


(a) *Blockchain Ethereum Emulada* **(b)** *Blockchain Bitcoin Simulada*

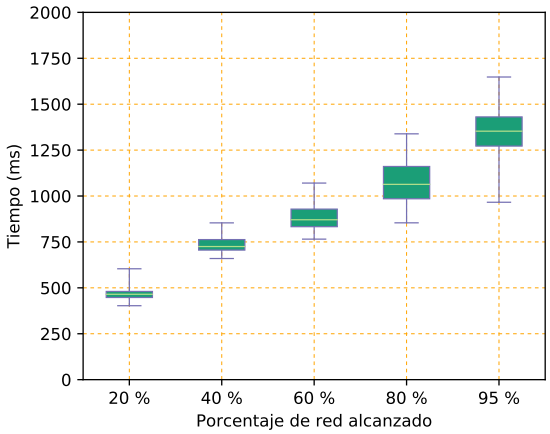
Figura 9.7: *Rendimiento de la blockchain - Topología emulada con latencias reales.*



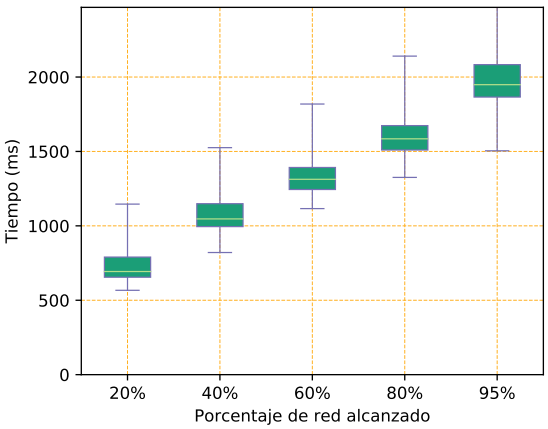
(a) *Target 2s - Blockchain Emulada* **(b)** *Target 2s - Blockchain Simulada*



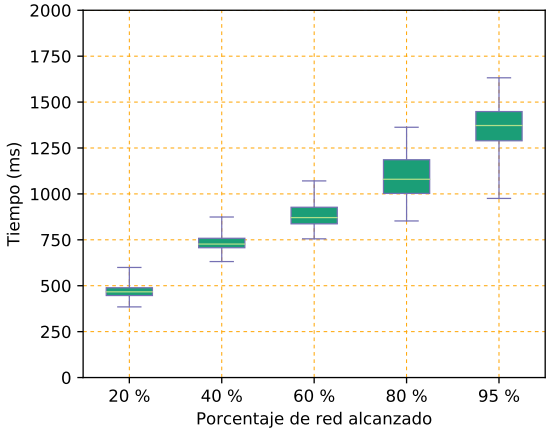
(a) *Target 4s - Blockchain Emulada* **(b)** *Target 4s - Blockchain Simulada*



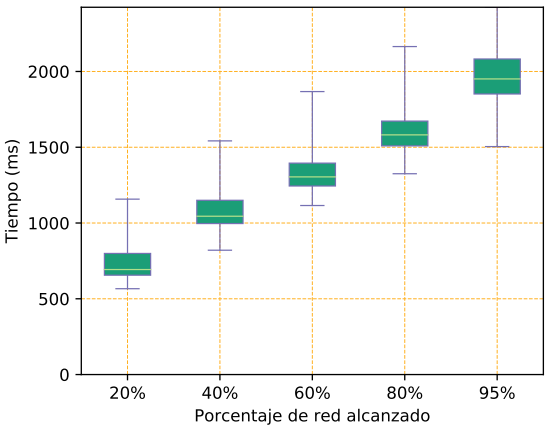
(a) *Target 6s - Blockchain Emulada*



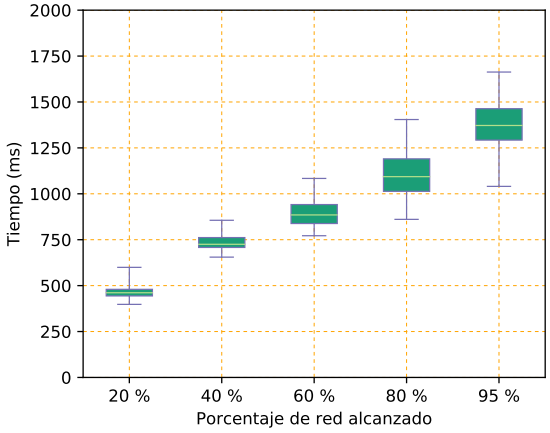
(b) *Target 6s - Blockchain Simulada*



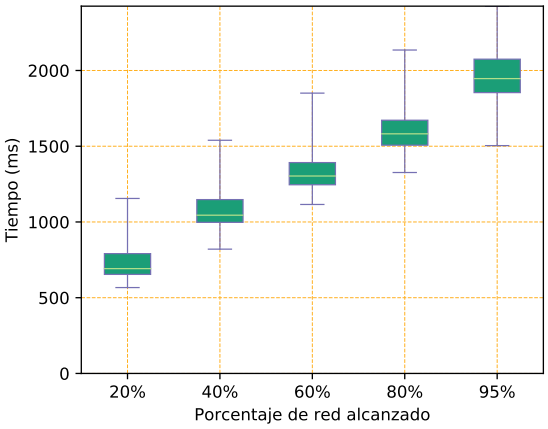
(a) *Target 8s - Blockchain Emulada*



(b) *Target 8s - Blockchain Simulada*



(a) *Target 10s - Blockchain Emulada*



(b) *Target 10s - Blockchain Simulada*

Cantidad de Bloques Competitivos - Emulación Ethereum					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	>4 bloques
10	92,60 %	7,30 %	0,10 %	0,00 %	0,00 %
8	90,40 %	9,20 %	0,40 %	0,00 %	0,00 %
6	87,50 %	11,70 %	0,80 %	0,00 %	0,00 %
4	82,60 %	15,40 %	1,90 %	0,10 %	0,00 %
2	65,50 %	28,30 %	5,80 %	0,40 %	0,00 %

Cantidad de Bloques Competitivos - Simulación Bitcoin					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	>4 bloques
10	79,64 %	19,55 %	0,81 %	0,00 %	0,00 %
8	75,42 %	23,27 %	1,31 %	0,00 %	0,00 %
6	68,85 %	28,38 %	2,77 %	0,00 %	0,00 %
4	58,8 %	34,56 %	5,39 %	1,25 %	0,00 %
2	36,78 %	45,05 %	15,23 %	2,6 %	0,33 %

Tabla 9.3: Estadísticas de ancho de forks.

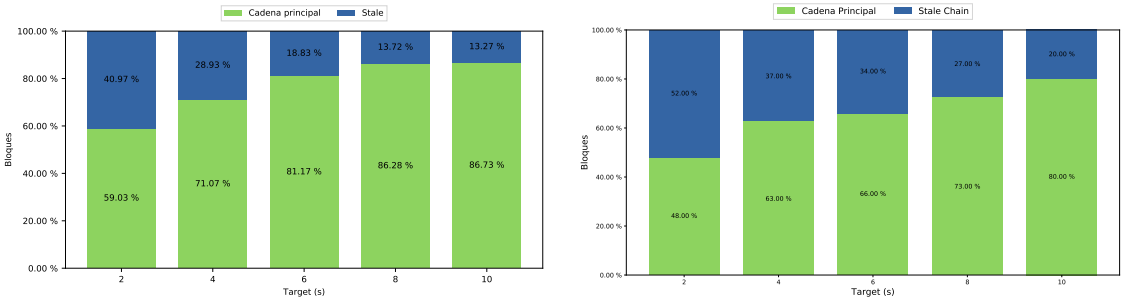
Profundidad de forks - Emulación Ethereum				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	69	3	0	0
8	90	5	0	0
6	121	6	0	0
4	165	12	2	0
2	281	57	6	3

Profundidad de forks - Simulación Bitcoin				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	107	4	0	0
8	134	5	0	0
6	167	11	1	0
4	233	20	2	0
2	362	68	10	5

Confiabilidad
Alta
Media
Baja

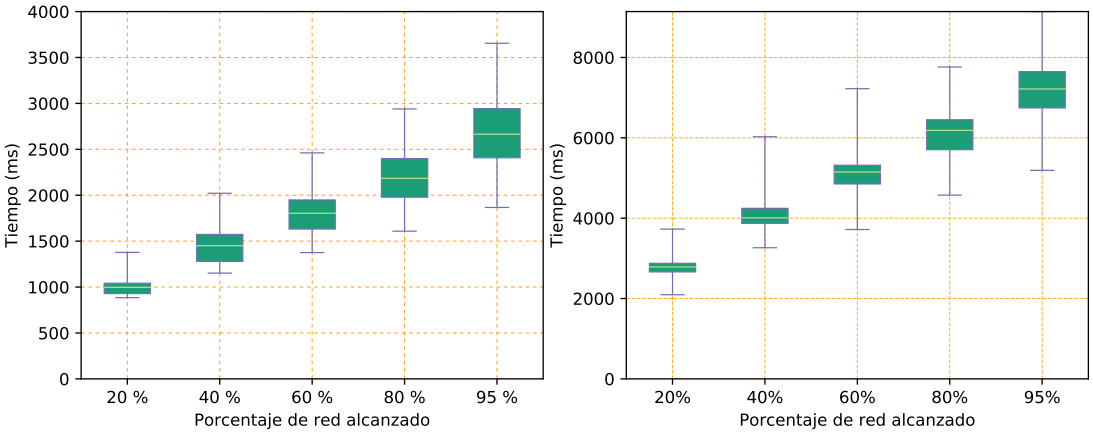
Tabla 9.4: Estadísticas de profundidad forks - Mapa de color de la confiabilidad de la Blockchain.

9.1.4 Latencia Constante: 200ms

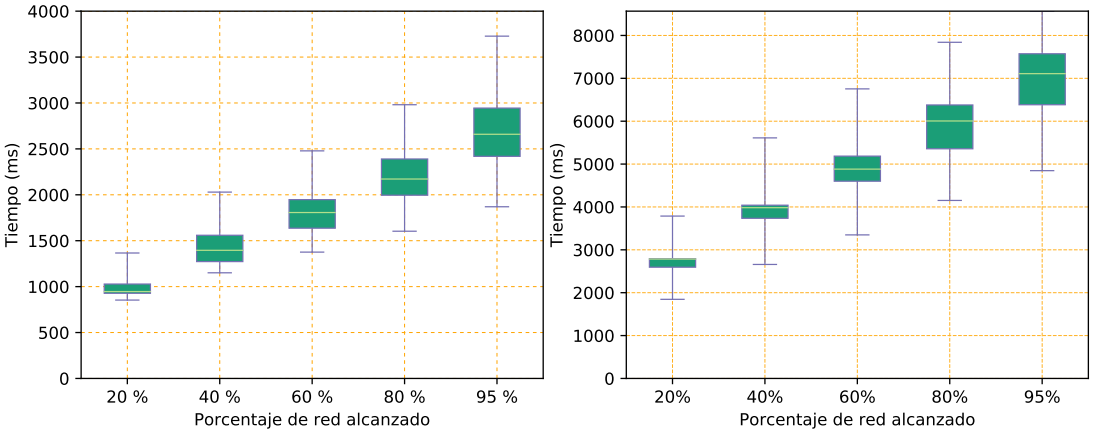


(a) *Blockchain Ethereum Emulada* **(b)** *Blockchain Bitcoin Simulada*

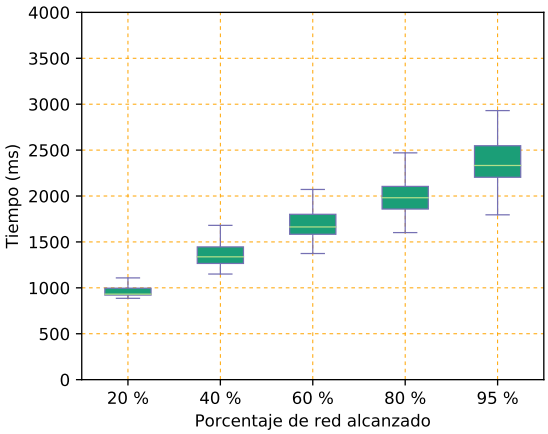
Figura 9.13: *Rendimiento de la blockchain - Topología emulada con latencias reales.*



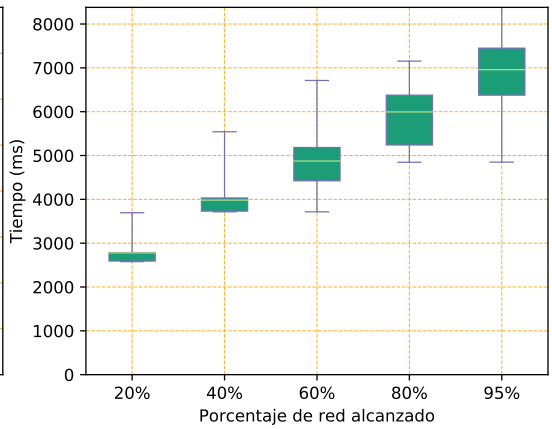
(a) *Target 2s - Blockchain Emulada* **(b)** *Target 2s - Blockchain Simulada*



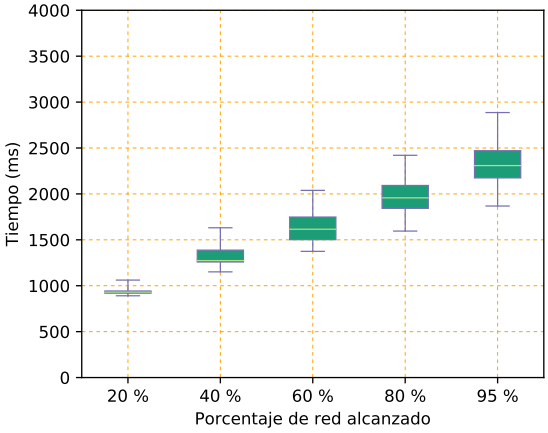
(a) *Target 4s - Blockchain Emulada* **(b)** *Target 4s - Blockchain Simulada*



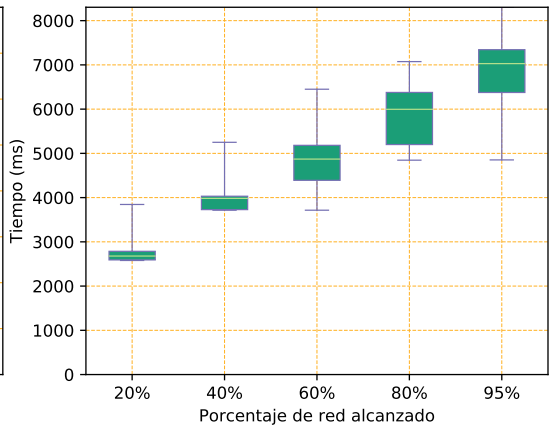
(a) *Target 6s - Blockchain Emulada*



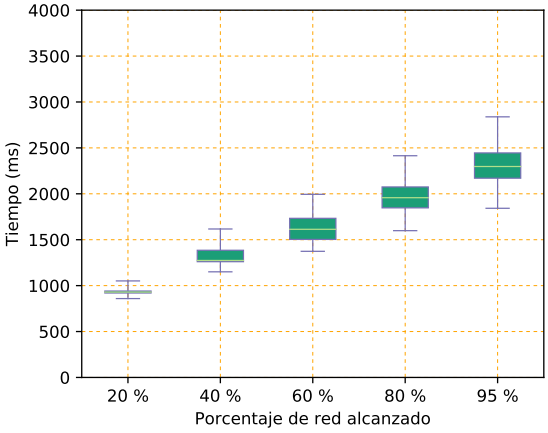
(b) *Target 6s - Blockchain Simulada*



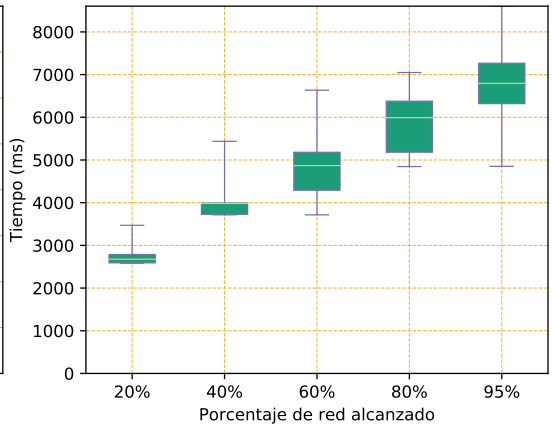
(a) *Target 8s - Blockchain Emulada*



(b) *Target 8s - Blockchain Simulada*



(a) *Target 10s - Blockchain Emulada*



(b) *Target 10s - Blockchain Simulada*

Cantidad de Bloques Competitivos - Emulación Ethereum					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	>4 bloques
10	86,00 %	12,80 %	1,20 %	0,00 %	0,00 %
8	85,40 %	13,40 %	1,20 %	0,00 %	0,00 %
6	79,30 %	18,40 %	2,20 %	0,10 %	0,00 %
4	65,50 %	29,90 %	4,60 %	0,50 %	0,00 %
2	47,20 %	39,40 %	10,70 %	2,30 %	0,40 %

Cantidad de Bloques Competitivos - Simulación Bitcoin					
Target (segs)	1 bloque	2 bloques	3 bloques	4 bloques	>4 bloques
10	48,52 %	40,49 %	9,54 %	1,45 %	0,00 %
8	42,57 %	43,33 %	12,17 %	1,93 %	0,00 %
6	30,51 %	47,43 %	18,38 %	3,68 %	0,00 %
4	20,31 %	44,78 %	23,83 %	8,96 %	2,13 %
2	6,62 %	28,45 %	31,86 %	24,14 %	8,42 %

Tabla 9.5: Estadísticas de ancho de forks.

Profundidad de forks - Emulación Ethereum				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	135	9	0	0
8	140	5	2	0
6	189	21	1	0
4	276	38	14	2
2	393	97	30	9

Profundidad de forks - Simulación Bitcoin				
Target (segs)	Prof = 1	Prof = 2	Prof = 3	Prof >3
10	297	34	6	1
8	346	40	10	0
6	395	99	13	5
4	469	128	40	19
2	563	228	101	104

Confiabilidad
Alta
Media
Baja

Tabla 9.6: Estadísticas de profundidad forks - Mapa de color de la confiabilidad de la Blockchain.

9.1.5 Discusión

En las secciones anteriores observamos que existe una relación entre el *target-time* y la aparición de forks en la red. Mientras más corto sea el tiempo entre bloques, mayor cantidad de bloques huérfanos ocurrirán, afectando el rendimiento y la confiabilidad de la red.
Notamos una correlación entre los resultados obtenidos en el escenario emulado sobre Ethereum y los obtenidos en el escenario simulado sobre Bitcoin. Observamos no obstante siempre un mayor porcentaje de bloques huérfanos para el caso de Bitcoin así como también una mayor incidencia de la latencia en la propagación de bloques en la red (es decir: en Ethereum la propagación no se ve tan afectada por el aumento de la latencia). Consideramos que esta mayor incidencia observada para el caso de Bitcoin se puede deber a los siguientes factores:

- incremento en el *orphan-rate*: el cual genera bifurcaciones temporales en la red que llevan más tiempo resolver.
- diferencias respecto al protocolo de Ethereum: siendo que el mismos está pensado para generar bloques cada 15 segundos con las optimizaciones que ello conlleva mientras que Bitcoin el *targe-time* es de 600 segundos el cual está muy alejado de los escenarios de {2, 4, 6, 8 y 10} segundos.
- simulación del modelo de red realizado por SimGrid: en la sección 4.2.2 mencionamos que al momento de validar nuestro simulador elegimos el modelo *SPMI* por ser el que mejor modelaba la latencia en el *slow start* de TCP. Sin embargo, no podemos descartar que dicho modelo sea insuficiente para representar fielmente la conexión real y dado que en los tres escenarios evaluados (latencial real, duplicada y constante) vimos la incidencia de la latencia en los mismos, cualquier deficiencia en la simulación de la misma es de esperarse que afecte los resultados obtenidos.

9.2 Herramientas Adicionales

9.2.1 Generación de archivos de entrada

Implementamos en Python las herramientas `createPlatformXml` y `createDeploymentXml`. Las mismas son determinísticas y su salida consiste en la generación de archivos para un escenario puntual de simulación. Su comportamiento está dado por los argumentos de entrada que detallamos a continuación.

- `utils/createPlatformXml`: permite generar una plataforma física indicando la cantidad de hosts y links entre los mismos así como la estrategia de routing que debe utilizar SimGrid para enviar los mensajes entre hosts.
- `utils/createDeploymentXml`: permite generar los datos de entrada para los actores de simulación entre los que podemos destacar:
  - cantidad de nodos de Bitcoin.
  - número de *peers* por nodo.
  - tipo de simulación. Permite elegir si queremos crear transacciones siguiendo un modelo de generación o bien reproducir una traza real de Bitcoin que recibimos como argumento de entrada.
  - *hashrate* global. Permite indicar el poder de cómputo global de todos los mineros en la red.
  - porcentaje de mineros por sobre los nodos totales.
  - tipo de distribución de transacciones. Permite elegir si vamos a distribuir uniformemente las transacciones o bien seguir una distribución exponencial.

Con estas herramientas podemos generar distintas topologías físicas y lógicas así como distintos parámetros que definirán el comportamiento de los actores simulados.

Dadas las opciones previamente detalladas, podemos utilizar estas herramientas auxiliares para preparar distintos escenarios de experimentación. Ejemplificamos a continuación algunos de los posibles escenarios a simular:

- **Supernodos**: consideramos que un porcentaje de los nodos de la red tienen una conectividad mayor a la indicada por el cliente de referencia, que es de 8 *peers* por nodo.
- **Supermineros**: consideramos que un porcentaje de los mineros tiene mayor *hash power* que el resto.
- **Mineros más conectados**: consideramos que un porcentaje de los mineros tiene mayor conectividad al resto de los mineros.
- **Cambios en *target time***: permite evaluar el comportamiento de la red ante bloques que se generan estadísticamente antes que los 10 minutos indicados por el cliente de referencia.
- **Reproducción de una traza real**: permite simular la aparición de transacciones y bloques con los mismos tamaños y en el mismo instante que ocurrieron en una traza real.

Se incluyen *scripts* auxiliares y especializaciones para las herramientas previamente mencionadas con el fin de poder realizar experimentos con escenarios más complejos. Entre las funcionalidades adicionales agregadas en dichas especializaciones podemos mencionar:

- para el experimento de "Supernodos" fue necesario poder configurar la cantidad de *peers* que iba a tener cada uno de los nodos. Esto se implementó mediante el agregado de una nueva opción en la herramienta `createDeployment`.
- para el experimento de "SmallWorld" fue necesario agregar la posibilidad de generar una topología estrella a la herramienta `createPlatform`.

9.2.2 Utilización de las herramientas para la realización de experimentos

Tiempo de confirmación de transacciones
Para conocer el tiempo promedio que tardan las transacciones en incluirse en un bloque podemos:

1. Tomar la fecha de creación de cada una en base a estos mensajes:

```
462.817250   node-72: creating tx 8386373514783821425
```

2. Tomar la fecha de inclusión de la misma en un bloque en base a estos mensajes:

```
462.817250   node-72: confirmed tx 8079046571094085839 in block 1587103434935014363
```

3. Tomar la diferencia entre los tiempos de ambos eventos para cada unas de las transacciones generadas y luego simplemente obtener el promedio.

Impacto de la conectividad en el minado

Para analizar el impacto que tiene la conectividad de un minero en la cantidad de bloques efectivos que mina, tenemos que cruzar los datos de la topología lógica con los del log de una simulación.
En el directorio `additional_experiments/supernodes` creamos una versión alternativa de la herramienta `createDeployment` que nos permite definir nodos más conectados que otros.

1. Para analizar el impacto de la conectividad entonces empezaríamos creando un escenario como el siguiente donde tenemos 10 mineros con el mismo ***hash power*** pero donde uno de ellos tiene 100 *peers* en vez de los 8 que tiene el resto:

```
additional_experiments/supernodes/createDeploymentXml \\
--nodes_count=1000 --peers_count=8 --miners_count=10 \\
--supernodes_count=0 --supernodes_peers=0 \\
--superminers_count=1 --superminers_peers=100 \\
--data_dir=/tmp/experiment_with_superminer \\
--difficulty=3462542391191 --global_hashrate=25130091717
```

2. Luego de correr una simulación para el escenario previamente creado, deberíamos extraer del log resultante las líneas como las siguientes:

```
46.817250   node-72: creating block 1587103434935014363 with 3 txs. height: 1...
```

3. Deberíamos quedarnos con los bloques que fueron parte de la *main chain*, es decir que no son parte de *forks*. Esto lo podemos lograr mediante estos mensajes que indican que un bloque fue satisfactoriamente aceptado por toda la red:

```
465.417250  node-274: BLOCK_RECEIVED_BY_ALL 1587103434935014363
```

4. En base a la plataforma lógica definida en el punto **1** podemos identificar el minero con mayor conectividad que el resto.
En base a los mensajes mencionados en el punto **2** podemos obtener la cantidad de bloques creados por cada minero.
En base a los mensaje del punto **3** podemos averiguar los bloques efectivamente aceptados por la red.
Con esta información podemos entonces identificar el impacto (si hubiese alguno) de la conectividad de un minero en relación a la cantidad efectiva de los bloques que se incluyen en la *Blockchain*.

Bibliografía

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, January 2002.

[DC19] Nicolás De Carli. Sobre los límites del tiempo entre bloques en bitcoin. Master's thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2019.

[DLM+17] Augustin Degomme, Arnaud Legrand, George S. Markomanolis, Martin Quinson, Mark Stillwell, and Frédéric Suter. Simulating mpi applications: The smpi approach. *IEEE Transactions on Parallel and Distributed Systems*, 28(8):2387–2400, August 2017.

[DPSHJ14] Joan Antoni Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The bitcoin P2P network. In *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 87–102, Berlin, Heidelberg, 2014. Springer International Publishing.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Proceedings of the IEEE P2P 2013*, pages 1–10. Institute of Electrical and Electronics Engineers, Inc, September 2013.

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, June 2018.

[GKKT16] Johannes Gobel, H. Paul Keeler, Anthony Krzesinski, and Peter Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, 104:23 – 41, 2016.

[GKW+16] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 3–16, New York, NY, USA, 2016. ACM.

[KAC12] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 906–917, New York, NY, USA, 2012. Association for Computing Machinery.

[KdBMA+13] Wagner Kolberg, Pedro de B. Marcos, Julio C.S. Anjos, Alexandre K.S. Miyazaki, Claudio R. Geyer, and Luciana B. Arantes. Mrsg – a mapreduce simulator over simgrid. *Parallel Computing*, 39(4):233 – 244, April 2013.

[MAL17] Apostolaki Maria, Zohar Aviv, and Vanbever Laurent. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392. Institute of Electrical and Electronics Engineers, Inc, May 2017.

[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.

[MKS17] Sina Mahmoodi Khorandi and Mohsen Sharifi. Scheduling of online compute-intensive synchronized jobs on high performance virtual clusters. *Journal of Computer and System Sciences*, 85:1 – 17, 2017.

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

[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.

[Yeo] Addy Yeow. Global bitcoin nodes distribution. `https://https://bitnodes.earn.com`. Accessed: 25 de noviembre de 2019.

[ZP17] Ren Zhang and Bart Preneel. Publish or perish: A backward-compatible defense against selfish mining in bitcoin. In Helena Handschuh, editor, *Topics in Cryptology – CT-RSA 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 277–292, Cham, Switzerland, 2017. Springer International Publishing.