
UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

# La mayoría sigue sin ser suficiente

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

**Nicolás Tallar**

Director de tesis: Esteban Mocskos

14 de Agosto de 2023
Buenos Aires, Argentina

Firma

# La mayoría sigue sin ser suficiente


**Resumen**

Las criptomonedas son activos digitales que son mantenidos por la colaboración entre los participantes de una red, denominados nodos. Bitcoin es el principal exponente de esta tecnología en cuanto a adopción y referencia como paradigma de ser un activo digital. Su principal atractivo, y en general de las criptomonedas, es que no está controlada por una entidad central, sino que es un sistema distribuido. La base de la seguridad de esta descentralización son los protocolos ejecutados por miles de nodos para consensuar la historia transaccional.

El almacenamiento de los datos se sustenta en la *blockchain*, una estructura de datos distribuida que se valida por todos los nodos de la red y construida por algunos de estos, los mineros. Los mineros compiten entre sí mediante un *puzzle* computacional para añadir transacciones a la *blockchain*, recibiendo a cambio de su participación una recompensa económica. A partir de la *blockchain* se puede derivar el *ledger*, el libro de cuentas indicando qué transacciones se realizaron y cuánto de la criptomoneda tiene cada usuario.

Al tratarse de una red de máquinas, cada nodo puede poseer una visión distinta del *ledger*, pero mediante protocolos comparten y unifican la información de manera de tender a una misma visión del sistema. Una de las hipótesis que se realizan respecto de estos protocolos descentralizados es que incentivan un comportamiento cooperativo entre los mineros: que su mejor estrategia es compartir inmediatamente toda información nueva que produzcan ellos mismos u obtengan de otros nodos. Sin embargo, aparecieron en la literatura algunos trabajos cuestionando que esta suposición sea verdadera para el caso de Bitcoin y presentaron distintas estrategias de comportamiento más económicamente atractivas que un comportamiento cooperativo. La primera estrategia desarrollada y la más investigada al momento es el *selfish mining*. A diferencia de la estrategia cooperativa, ésta se basa en administrar con astucia la información disponible de la *blockchain*, reteniendo gran parte para ser su único conocedor, y así tener ventaja en la competencia por añadirle transacciones.

Los trabajos mencionados se centran en analizar de forma teórica esta estrategia de minado y otras variantes. En esta tesis, nos centramos en el análisis del ataque de *selfish mining* pero, a diferencia de los trabajos de la literatura, lo hacemos desde el punto de vista operacional: utilizamos una herramienta de simulación que permite acercanos al comportamiento real de Bitcoin para evaluar este ataque. La herramienta de simulación fue desarrollada usando `SimGrid`, lo que nos permite estudiar escenarios de gran escala y tener la seguridad de estar utilizando una herramienta con una fuerte adopción en la comunidad de simulación. Para nuestra experimentación, comenzamos estudiando escenarios controlados de baja escala, cuyos resultados fueron utilizados para verificar el comportamiento del atacante y el resto de los nodos. Luego, gracias a la escalabilidad de `SimGrid`, realizamos simulaciones a gran escala cercanas a Bitcoin: utilizamos topologías de 10000 nodos respetando el tamaño y las propiedades de su red actual.

La primera de las conclusiones a las que llegamos es que, en los experimentos de topologías chicas, los resultados obtenidos en la práctica fueron efectivamente los esperados de acuerdo a la literatura. Por otro lado, para los experimentos a gran escala se observó que realizar el ataque es más rentable que no hacerlo, pero no pudimos concluir si es tan rentable como lo predicho teóricamente, ya que no existen actualmente herramientas para realizar dichas predicciones en sistemas similares al real. Como alternativa, se presentó un estimador de la rentabilidad del atacante en base a propiedades del grafo surgido a partir de las conexiones entre nodos. Transversalmente, exploramos el funcionamiento del ataque realizando un análisis del uso de los recursos de todos los mineros de la red. Esto puso en evidencia una fuente adicional de ineficiencia del sistema cuando el ataque está activo.



**Keywords:** Bitcoin; Blockchain; Selfish mining; Simulación

# Majority is still not enough


**Summary**

Cryptocurrencies are digital assets maintained through the collaboration of participants in a network called nodes. Bitcoin is the dominant exponent of this technology in terms of adoption and reference as a paradigm for being a digital asset. Its main attraction, and generally of cryptocurrencies, is the lack of a central controlling entity but rather a distributed system. The basis for the security of this decentralization is the protocols executed by thousands of nodes to reach a consensus on the transactional history.

The blockchain supports data storage, a distributed data structure validated by all nodes in the network and built by some of these called miners. Miners compete with each other through a computational puzzle to add transactions to the blockchain, receiving an economic reward for their participation. The ledger which is the accounting book indicating which transactions were made and the amount of cryptocurrency each user has, can be derived from the blockchain.

As it is a network of machines, each node may have a different view of the ledger, but through protocols, they share and unify the information in order to tend toward a shared vision of the system. One hypothesis regarding these decentralized protocols is that they incentivize cooperative behavior among miners: their best strategy is to immediately share any new information they produce or obtain from other nodes. However, some studies have questioned whether this assumption holds for Bitcoin and present different behavior strategies that are more economically attractive than cooperative behavior. The first developed and the most researched strategy is selfish mining. Unlike the cooperative strategy, this one is based on cunningly managing the available information on the blockchain, retaining a large part of it to be the only one knowing it, and thus having an advantage in the competition to add transactions.

The mentioned studies focus on theoretically analyzing this mining strategy and other variants. In this thesis, we focus on analyzing the selfish mining attack, but unlike other works of literature, we do it from an operational point of view: we use a simulation tool that allows us to get closer to the actual behavior of Bitcoin to evaluate this attack. The simulation tool is based on SimGrid, which has a strong adoption in the simulation community, allowing us to study large-scale scenarios. For our experiments, we started studying controlled low-scale scenarios, whose results verified the behavior of the attacker and the rest of the nodes. Then, thanks to the scalability of `SimGrid`, we carried out simulations on a large scale close to Bitcoin: we used topologies of 10 000 nodes respecting the size and properties of its current network.

The first of the conclusions we reached is that in the experiments of small topologies, the results obtained in practice were effectively the ones expected according to the literature. On the other hand, for the large-scale experiments, it was observed that carrying out the attack is more profitable than not doing it, but we could not conclude if it is as profitable as theoretically predicted, since there are currently no tools to make such predictions in systems similar to the real one. As an alternative, an estimator of the attacker's profitability was presented based on properties of the graph arising from the connections between nodes. Transversely, we explored the operation of the attack by analyzing the use of resources by all miners in the network, revealing an additional source of inefficiency in the system when the attack is active.


**Keywords**: Bitcoin; Blockchain; Selfish mining; Simulation

# Índice general

# 1

# Conceptos preliminares

## 1.1 Introducción

Las criptomonedas son medios digitales de intercambio, donde el registro de las operaciones se realiza de forma descentralizada, y a través de tecnología criptográfica. Esta naturaleza descentralizada se logra mediante el uso de protocolos e incentivos, que permiten que los nodos (las computadoras conectadas a la red) lleguen a un consenso sobre qué operaciones se realizaron y qué activos tiene cada uno de los usuarios de la moneda.

Una pregunta importante a hacer entonces es: ¿estos protocolos e incentivos garantizan que la criptomoneda sea segura? En las criptomonedas, cualquier persona puede configurar un nodo y conectarse a la red, y, más aún, diferenciar su comportamiento del resto de los nodos para intentar obtener más beneficio propio. Hay un gran interés de la comunidad en responder esta pregunta, tanto desde el análisis de las propiedades garantizadas por los protocolos, como también planteando una variedad de ataques posibles y analizando su impacto. Algunos de los ataques más conocidos son: ataques del 51 %, ataques eclipse, ataques de *double-spending*, *selfish mining*.

Este trabajo de tesis busca contribuir a continuar analizando el impacto de los ataques a criptomonedas, centrándose exclusivamente en el ataque de *selfish mining*, específicamente en cómo éste impactaría en Bitcoin.

### 1.1.1 Glosario

- **bloque *stale***: Bloque que no terminó en la *main chain* de un nodo en particular o de toda la red
- **Coinbase**: La primera transacción de un bloque, que contiene la *reward* que le corresponde al minero por generarlo.
- **dificultad de un bloque**: Medida que representa cuan difícil fue el minado del bloque [1]. Éste es actualizado cada 2016 bloques para mantener un tiempo entre bloques promedio de 10 minutos.
- **dificultad total de un bloque B**: Suma de las dificultades de todos los bloques entre B y el génesis, siguiendo el recorrido planteado a través de la relación padre-hijo entre los bloques.

[1] Para más detalles: `https://en.bitcoin.it/wiki/Difficulty`

- **PoW**: Protocolo de producción de bloques de una blockchain, donde para producir un bloque es necesario hashear su estructura (cambiando algún *nonce*) hasta que el resultado tenga una cantidad ceros esperados como prefijo.
- **hashing power de un nodo**: Cantidad de hashes de PoW que éste puede realizar por segundo. Principal medida del poder de cómputo de un nodo.
- **main chain de un nodo**: La mejor cadena de bloques que posee ese nodo. En el cliente de referencia, la mejor cadena es la de mayor dificultad total, desempatando por la cadena que primero le llegó al nodo.
- **mainnet de Bitcoin**: red donde se intercambian los bitcoins con valor real, a la cual uno implícitamente se refiere al hablar de la red de Bitcoin. Es utilizado en contraste a la *testnet* de Bitcoin, la cual es una red utilizada para pruebas y sin ningún valor monetario.
- **minero**: Nodo de Bitcoin que participa de la producción de bloques.
- **nodo**: Instancia del cliente de Bitcoin (es decir, cualquier programa que cumpla con su protocolo) conectado a la red.
- **reward de un bloque**: Cantidad de bitcoins recibidos por el minero de ese bloque.

### 1.1.2 Breve introducción a Bitcoin

Bitcoin, presentado en 2008 a través del trabajo [Nak08], fue la primera criptomoneda, que busca descentralizar las operaciones monetarias sobre una red *p2p*, sin necesidad de confiar en una tercera parte de confianza (*trusted third-party*).

Para esto, basa su seguridad en tecnologías criptográficas: hash criptográfico, criptografía asimétrica y firmas digitales. Los hashes criptográficos son funciones de tamaños arbitrarios a un tamaño fijo (en general 32 bytes), que permiten generar identificadores únicos de la información hasheada. Por otro lado, la criptografía asimétrica es el uso de una clave pública derivada a partir de una clave privada, que se usa para la encriptación de información; todos los datos encriptados con la clave privada solo pueden ser descencriptados con la clave pública y viceversa. Uno de sus usos es para generar firmas digitales; al adjuntar a un mensaje Alice la encriptación de este mismo mensaje con su clave privada, Bob puede usar la clave pública de Alice para verificar que ésta fue la que autorizó este mensaje. Por ejemplo, en Bitcoin los usuarios autorizan sus transacciones mediante el uso de firmas digitales (con sus claves asimétricas) del hash de estas.

Siendo una red *P2P*, cualquier persona puede levantar alguno de los programas compatibles con la red[2] y contribuir al procesamiento de sus operaciones; llamaremos *nodos* a los participantes. Estos nodos se agrupan en dos redes: la *mainnet*, donde se transfieren los Bitcoins con valor económico y la *testnet*, utilizada exclusivamente para pruebas y sin valor real.

A continuación, se entrará en detalle en cada uno de los aspectos de Bitcoin que consideramos importante tener en cuenta para el resto del trabajo.

#### Estructuras de datos

En esta sección se describirán las principales estructuras de datos involucradas en la operatoria de Bitcoin.

[2] `https://en.bitcoin.it/wiki/Software#Bitcoin_clients`

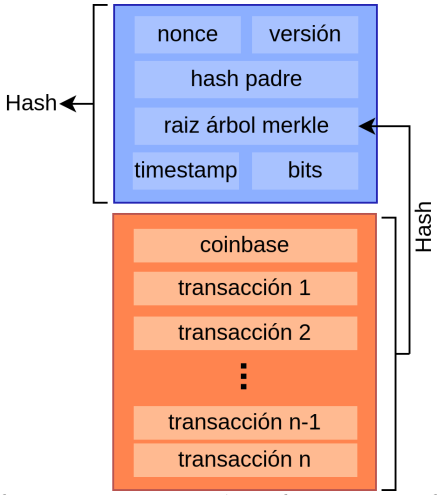
**Transacciones** Las operaciones que se pueden realizar en Bitcoin son de transferencia de cantidades determinadas de la moneda digital. La unidad más pequeña de la moneda digital es llamada **Satoshi** y a 100 millones de satoshis se los denominá 1 Bitcoin (1 BTC), mientras que a las operaciones de transferencia se las llama **transacciones**.

Los usuarios, dueños de bitcoins, operarán con ellos mediante un esquema de clave pública y privada, siendo identificados por su dirección (*address*, el hash de su clave pública). Las transacciones por lo tanto, registran la transferencia de Bitcoins desde su dueño (i.e., *sender*) a sus nuevos dueños (i.e., *receivers*), siendo todos estos identificados por sus direcciones.

No todas las transacciones serán válidas, como podría ser el caso en que el *sender* no posea los bitcoins que está transfiriendo, o que la cantidad recibida por los *receivers* sea mayor a la enviada por el *sender*, entre otros posibles ejemplos.

**Bloques** Dadas las transacciones generadas por los usuarios, los nodos de Bitcoin se ponen de acuerdo en cuáles de ellas ejecutar y en qué orden, la principal estructura utilizada para esto son los *bloques*. Estos se encuentran divididos en *body*, que contiene una lista ordenada de transacciones, y *header*, con metadata. Los campos del header son:

- Un número de versión
- El hash del bloque padre (descrito en la sección 1.1.2)
- El *merkle root* de las transacciones: identifica el *body* correspondiente a este *header*
- El tiempo en segundos donde fue minado el bloque
- La dificultad de minado del bloque (descrito en la sección 1.1.2)
- El *nonce* correspondiente a la PoW (descrito en la sección 1.1.2)




Existe un cierto orden entre los bloques (que se describirá en la siguiente sección), determinando entonces un orden entre todas las transacciones. Por este motivo, los bloques serán la unidad de transferencia de transacciones a incluir en el registro.

También ocurre que no todos los bloques son válidos, lo cual puede ocurrir por ejemplo si posee alguna transacción inválida, o algunos de los campos de su header son inválidos.

**Blockchain** Cada bloque es identificado unívocamente por su hash. Éste se utiliza para establecer un orden entre los bloques mediante una relación de padre-hijo, cada bloque B2 producido deberá ser el siguiente a alguno creado previamente B1: B2 será denominado el hijo de B1 y B1 el padre de B2. Esta relación quedará demarcada en el bloque B2 teniendo al hash de B1 en el campo "hash del bloque padre".

El bloque inicial es denominado el **bloque génesis** y es parte de la configuración del sistema. Entonces, la red de Bitcoin puede ponerse de acuerdo en el estado de las cuentas de los usuarios llegando a un consenso sobre algún bloque. Si la red de Bitcoin se puso de acuerdo en un determinado bloque B, entonces se puede determinar el estado de las cuentas ejecutando las transacciones de cada bloque (en el orden indicado por su *body*) desde el bloque génesis hasta el bloque B, recorriéndolos mediante la relación de padre-hijo. A la estructura que se deriva de la relación padre-hijo entre bloques se la denomina una **blockchain**.

Un valor útil que se puede derivar de ésta, es el orden que posee un bloque respecto al bloque génesis, denominado también la **altura del bloque**. El bloque génesis tiene altura 0, y cada uno de los hijos de un bloque con altura $n$ tendrá altura $n + 1$.

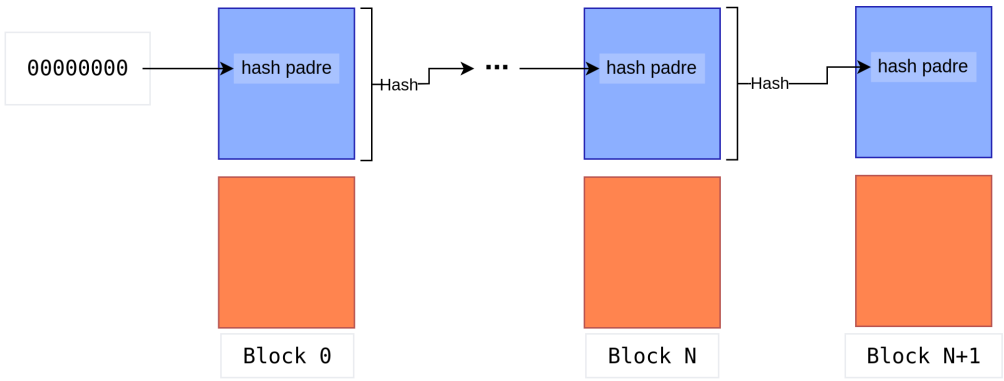



**Figura 1.1:** *Relación padre-hijo entre bloques a través de sus hashes. Cada uno es el hijo del que tiene pegado a su izquierda, salvo el bloque génesis que no tiene padre.*

**Mempool** A las transacciones que no se hayan incluido en bloques aceptados hasta el momento se las llaman **transacciones pendientes**. Cada nodo mantendrá un registro del conjunto de éstas en una estructura denominada **mempool**. Si un nodo procesó hasta un bloque B, el objetivo de esta estructura es poseer toda transacción no incluida en ninguno de los bloques que relacionan a B con el génesis, y que sea considerada con chances de ser incluida en futuros bloques.

Entonces, nuevas transacciones serán añadidas al ser descubiertas mediante su propagación por la red (si cumplen algunas reglas mínimas de validez) o al pertenecer a un bloque que ya no es aceptado por el nodo. Y serán quitadas al pertenecer a un bloque que pasa a ser aceptado por el nodo.

## Protocolos de minado

Todos los nodos de la red que lo deseen pueden participar de la producción de bloques. Debido a que la validación de los bloques de Bitcoin se acompaña con una recompensa en forma de emisión de bitcoins, se asocia este procedimiento con la analogía de la minería tradicional: los nodos productores son llamados **mineros** y al proceso de generación de bloques se lo llama **minado**.

En pocas palabras, los bloques serán generados mediante:

1. La selección del bloque padre del bloque
2. La selección ordenada de transacciones válidas del mempool
3. La creación de la transacción de coinbase
4. Completar la metadata del header (salvo su *nonce*)
5. El cómputo del PoW del bloque

El PoW de un bloque es un *puzzle* computacional (muy costoso de resolver) que ordena su generación. Este *puzzle* computacional se trata de variar el campo *nonce* del *header* del bloque hasta que su *hash* sea menor a un valor deseado y, entonces, tenga una requerida cantidad de ceros como prefijo. Dado que obtener el *hash* de un bloque es la operación más costosa de este proceso, el poder de cómputo de los mineros es medido en la cantidad de *hashs* por segundo que pueden realizar y se lo denomina **hashing power**.

La cantidad de ceros requerida es determinada por la *dificultad* del bloque, la cual es ajustada periódicamente (cada 2016 bloques) para que estos sean minados cada 10 minutos en promedio. Entonces, un aumento del *hashing power* de la red (por la aparición de un nuevo minero o el incremento del poder de

cómputo de uno existente) ocasionará que eventualmente también aumente la dificultad de minado de los bloques, y viceversa.

Existe un mecanismo de incentivos, con el fin de que los participantes de la red actúen como mineros y que quieran sus bloques sean incorporados a la blockchain, con la suposición de que esto incentiva su comportamiento cooperativo. Este mecanismo, es que cada bloque minado incluya la emisión de un monto predeterminado de bitcoins para el minero del bloque, que es llamada **recompensa** o *reward*. Aparte, los usuarios que generan transacciones fijan una comisión, llamada *fee* que deciden darle a los mineros que las incluyan en sus bloques, incentivando así que eso ocurra. Tanto la *reward* como los *fees* son incluidos en la *transacción de coinbase* de los bloques.

## Protocolos de *networking*

Bitcoin funciona sobre una red *peer-to-peer (p2p)*, cada nodo $n$ estará conectado por Internet a un conjunto de otros nodos, a los que llamaremos los *peers* de $n$. Cada nodo se comunicará con sus *peers* mediante mensajes, utilizándolos por ejemplo para propagar las transacciones y los bloques que conoce, haciendo que eventualmente toda la red se entere de éstas.

Existen ciertos protocolos que los nodos tienen que cumplir con sus *peers*: protocolos de *handshaking* para iniciar una conexión, no enviarles a sus *peers* operaciones inválidas ni duplicadas, entre otros. En caso de incumplimiento, un nodo podrá ser castigado con la desconexión y/o impidiendo la reconexión en el corto plazo, limitando así el comportamiento que pueda tener un nodo, y buscando incentivar un comportamiento cooperativo y que sume a la seguridad de la moneda.

## Protocolos de consenso

El hecho de tratarse de una red *p2p* introduce latencias en la distribución de los bloques producidos, esto junto con la naturaleza aleatoria del proceso de minado pueden ocasionar que dos mineros produzcan bloques con el mismo padre. Por ejemplo, en la figura 1.3, la demora en que el **bloque i+2** (producido por el minero 2) sea recibido por el minero 3 ocasiona que éste haya producido al **bloque i+2'** para ese momento.

La blockchain no es, entonces, una cadena de bloques sino un árbol de bloques, siendo cada una de sus ramas denominadas *forks*. Mediante un algoritmo de comparación (descrito más adelante), el mejor *fork* de un nodo será su *main chain*, mientras que el resto de los bloques serán denominados bloques *stale*. Al bloque de mayor altura de la *main chain* lo denominaremos como su **tope**. Los mineros usarán al tope como el bloque padre del nuevo bloque a producir, al menos según los clientes de referencia.

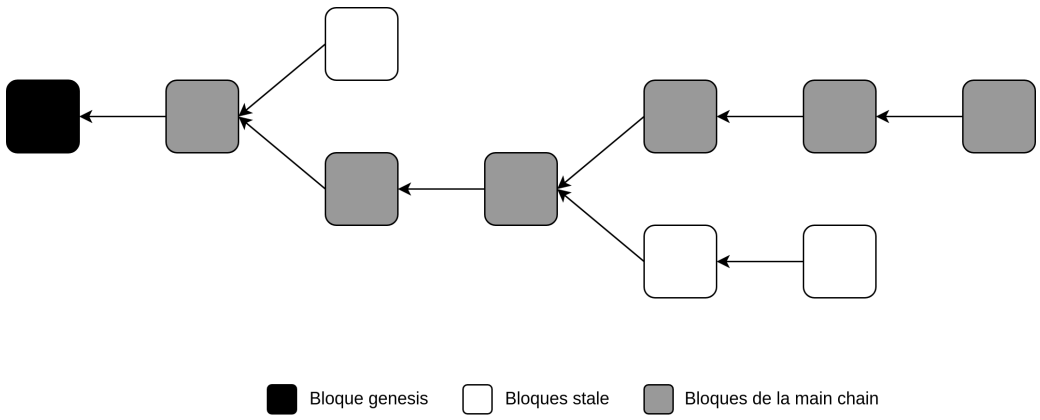



**Figura 1.2:** *La blockchain es un árbol de bloques: el génesis y la* main chain *forman un camino, al cual se conectan los* forks *producidos y abandonados.*

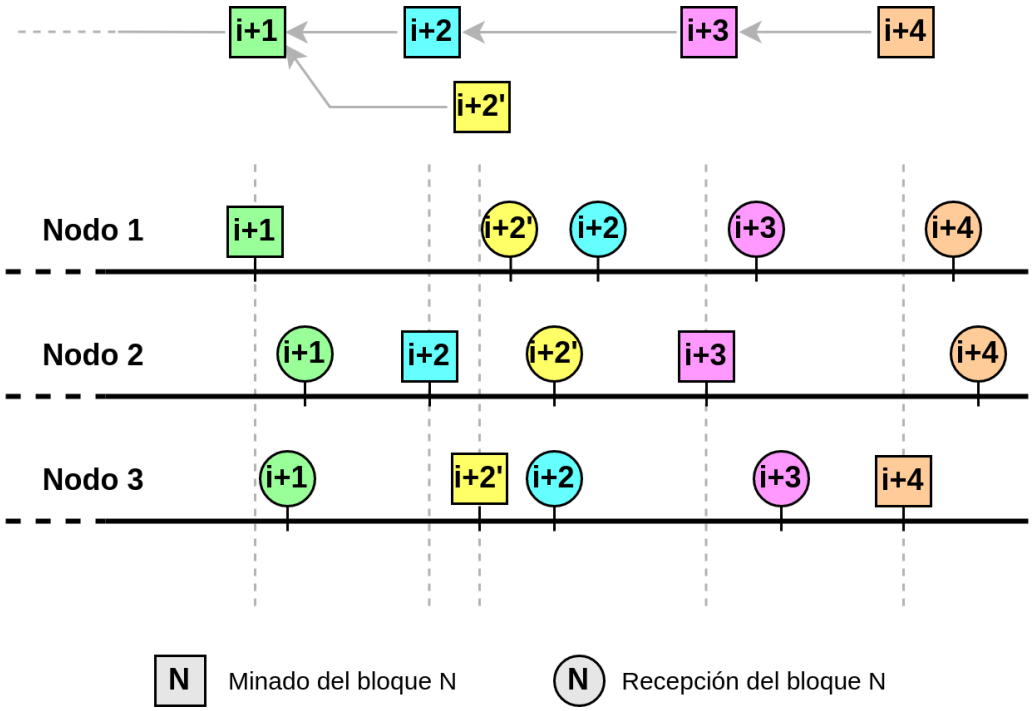



**Figura 1.3:** *Ejemplo de la naturaleza distribuida de la blockchain: los bloques minados por un nodo llegan con diferentes latencias a las distintas partes de la red. Esto puede ocasionar el minado de bloques con el mismo padre como es el caso del* ***bloque i+2*** *y* ***i+2'***.

Dados dos o más *forks*, en los clientes de referencia de Bitcoin[3] el mejor será el que:

1. **Posea la dificultad total más alta**: es decir, que la suma de las dificultades de los bloques de un *fork* sea más alta que la suma del otro

2. **El primero en llegar**: si un nodo se encuentra ante la situación de tener que decidir entre dos *forks* que tienen la misma dificultad total, la forma de desempate será quedarse con el que recibió primero.

El segundo punto ocasiona que aún poseyendo dos nodos los mismos bloques, sus *main chain* (y por lo tanto su vista de la blockchain) puedan ser distintas. A medida que se vayan minando nuevos bloques y propagados por la red, irán cambiando las *main chains* y bloques *stales* de cada nodo.

El hecho de poseer estas reglas en común de comparación de los bloques, y el tiempo relativamente grande de generación de bloques (10 minutos), ocasionará que los nodos eventualmente lleguen a un consenso sobre la blockchain. En términos generales, se garantizará que haya un prefijo largo en común entre las *main chains* de todos los nodos; a los bloques de este prefijo se le denominan bloques confirmados. Por ejemplo, los usuarios de Bitcoin usualmente suponen que los nodos solo varían entre sí en los últimos seis bloques como mucho[4].

### 1.1.3 Breve introducción a *selfish mining*

Pese a las reglas especificadas en la sección 1.1.2, los nodos poseen bastante libertad en su estrategia de minado a usar: pueden elegir el bloque padre que quieran para sus bloques minados, y los pueden transmitir a la red cuando mejor les convenga, entre otras cosas. Pero, se supone que, dada una mayoría del *hashing power* de los mineros con comportamiento cooperativo, el protocolo incentiva que el resto también

[3] `https://en.bitcoin.it/wiki/Software#Bitcoin_clients`
[4] `https://en.bitcoin.it/wiki/Confirmation`

lo sea. Una intuición detrás de esto es que, por ejemplo, siendo minero me conviene propagar un bloque apenas lo mino, para que el resto de los mineros mine sobre él y sea más probable que no se convierta en un bloque *stale*.

El trabajo de Eyal y Gün Sirer [ES18] es uno de los primeros donde se cuestiona que esto sea realmente así, y se presenta una estrategia de minado potencialmente más rentable que el comportamiento cooperativo, aún ante una mayoría cooperativa: el *selfish mining*. Ésta es más rentable (bajo algunas condiciones) por generar mayores *rewards* para un minero *selfish* en comparación con comportarse de forma cooperativa.

Este ataque todavía no se ha registrado en la red pero las posibilidades de que esto se dé son inciertas, así como el impacto ni las complicaciones de que el protocolo de Bitcoin se comporte de esta forma [NKMS16]. Esto abre así un espacio para análisis sobre este ataque, los posibles mecanismos de defensa y otras variantes de ataque que puedan ser más efectivas. En este trabajo nos enfocar en estos aspectos: explorará el comportamiento del ataque y sus posibles consecuencias.

### 1.1.4 Breve introducción a `SimGrid`

`SimGrid` es un *framework* orientado a la simulación de eventos discretos para trabajar sobre sistemas de cómputo distribuido. Puede ser utilizado tanto para evaluar algoritmos abstractos como para realizar *benchmarks* de aplicaciones paralelas basadas en `MPI`. Es por esto que, generalmente, es usado para la simulación en: cluster computing, cloud computing, peer-to-peer systems, entre otros. Siendo Bitcoin (y las criptomonedas en general) redes peer-to-peer, resulta un buen candidato para su modelado y experimentación.

`SimGrid` posee una variedad de interfaces que expone, pero la única utilizada en este trabajo es su interfaz `S4U`. Ésta se basa en el modelo de actores[5]: cada unidad computacional es un actor y se comunica con el resto de los actores del sistema mediante mensajes. Esta interacción se realiza a través de *mailboxes* (colas de mensajes) en los cuales cualquier actor puede encolar y desencolar mensajes.

Este modelo permite representar muy intuitivamente el comportamiento de redes *P2P* (específicamente también el de las criptomonedas): cada nodo es representado por un actor, y la conexiones entre ellos por *mailboxes*. Este modelo progresará entonces mediante cada nodo recibiendo mensajes a través de los *mailboxes* dirigidos a él, procesándolos y enviando otros mensajes a sus *peers* usando los *mailboxes* que parten de él.

Por otro lado, `SimGrid` permite representar la diferencia entre las topologías físicas y lógicas de una aplicación distribuida: la topología física es la obtenida por las conexiones por cables entre las máquinas, mientras que la lógica es la utilizada a nivel de aplicación, también se la conoce como *overlay*. En el caso particular de Bitcoin, la topología física es la internet mientras que la lógica es la red P2P. `SimGrid` llama *plataforma* a la topología física y *deployment* a la topología lógica, y permite representar el enroutamiento y las distancias de cada una por separado.

Este trabajo utilizará *SimGrid* como su modelo de simulación, entraremos en más detalles al respecto en la sección 2.1.

## 1.2 Trabajo Relacionado

Este trabajo complementa principalmente los trabajos de tesis de Marcos Kevin Piotrkowski [Pio19] y Nicolás De Carli [DC19].

Marcos Kevin Piotrkowski [Pio19] explora la utilización de `Simgrid` como *framework* para la simulación de redes de Bitcoin, realizando algunas primeras experimentaciones en base al *selfish mining*. El simulador implementado por él será utilizado como base para las experimentaciones realizadas en este trabajo.

[5] `https://en.wikipedia.org/wiki/Actor_model`

Nicolás De Carli [DC19], al contrario, basa sus análisis en la emulación de redes de Bitcoin (más chicas por ese motivo), centrándose en el tiempo entre bloques. El sistema de *log* y su procesamiento han sido las bases para los utilizados en este trabajo.

Por otro lado, el análisis de medidas de centralidad de grafos realizado por Gustavo Juan Cairo [Cai21] se utilizó de base para la selección de una de ellas para el uso en este trabajo.

### 1.2.1 Sobre grafos y topologías

Dos trabajos sobre topologías resultaron de interés para modelar a Bitcoin, ya que estas propuestas tienen características similares a la red de Bitcoin. Watts y Strogatz [WS11] presentan un modelo para la representación de redes aleatorias de mundo pequeño con *clustering* alto. Albert y Barabási [AB02] presentan un modelo para generar redes aleatorias complejas libres de escala empleando un mecanismo denominado **conexión preferencial**.

Por otro lado, en el trabajo analizaremos la conectividad de nodos en una determinada topología mediante métricas de centralidad. La métrica *load centrality* será la utilizada con este fin, la cual fue presentada por Uoh et al. en [UKK11].

### 1.2.2 Sobre Bitcoin

Miller et al. [MLP+15] crearon `Coinscape`, una herramienta para mejorar la caracterzación de la topología de la *mainnet* de Bitcoin. En su trabajo, analizan los resultados obtenidos por el uso de esta herramienta y las conclusiones que se pueden obtener acerca de las características de la red.

`Coinscape` analiza las conexiones de los nodos mediante la metadata provista por la respuesta al mensaje de *GetAddr*, permitiendo distinguir conexiones entrantes y salientes. Aparte, busca detectar qué nodos de la red son los más influyentes enviando transacciones conflictivas a distintos nodos de la red y viendo cuales terminan apareciendo en un bloque minado.

De los resultados obtenidos se concluye que, en general, los nodos tienen aproximadamente ocho *peers*, lo que se coincide con la cantidad de conexiones salientes que tiene el cliente oficial de Bitcoin[6]. Sin embargo, hay algunos que tienen más de 100 *peers*, los autores indican que estos últimos podrían ser *pools* de minería y otras herramientas de monitoreo de la red.

También se concluye que, si bien la red se asemeja un grafo aleatorio, no lo es puramente. Para llegar a esta conclusión se realizan detecciones de comunidades (mediante el algoritmo de Louvain [BGLL08]) y se comparan los resultados de medidas de conectividad con grafos aleatorios. El resultado es que la media de estas magnitudes se encuentra a dos desvíos estándar de distancia de la media de esos mismos valores para grafos aleatorios, lo que indicaría que la red de Bitcoin no es puramente aleatoria. Los autores proponen que esto se debe a que todos los nodos se empiezan a conectar a partir de los mismos *bootnodes* y de a poco van descubriendo el resto de la red.

Todas estas observaciones serán de utilidad al momento de selección del modelado de la red de Bitcoin.

### 1.2.3 Sobre *selfish mining*

Varios trabajos han analizado el ataque de *selfish mining* y sus distintas variaciones intentando aumentar su efectividad.

Eyal y Gün Sirer [ES18] presentan, por primera vez, el ataque de *selfish mining* y realizan un análisis del impacto sobre Bitcoin. Aparte, presentan una modificación posible al desempate entre *forks* para disminuir la efectividad del ataque.

[6] `https://bitcoin.org/en/full-node#reduce-traffic`

Nayak et al. [NKMS16] generalizan la estrategia original, presentando estrategias *stubborn* que son mejores que el ataque original. Analizan también en ese trabajo como el atacante puede combinar su ataque con ataques de eclipse para amplificar más su ganancia.

Concurrentemente, Sapirshtein et al. [SSZ15] también observan que el *selfish mining* no es el mejor ataque de la familia y presentan una posible generalización distinta.

Mientras todos los trabajos anteriores se centraron en que haya un único atacante, Liu et al. [LRDJ18] exploran la efectividad del ataque en el caso que haya múltiples atacantes concurrentes. Utilizando distintas variantes del ataque (*selfish* y *stubborn mining*) analizan la coexistencia de ellas, las cuales son combinadas con una nueva estrategia *publish-n* que es más eficiente en la presencia de múltiples atacantes.

Por otro lado, otros trabajos se han focalizado en distintos mecanismos para disminuir la efectividad del ataque original y sus variaciones.

Zhang y Preneel [ZP17] proponen un cambio a la comparación entre *forks*, que incentiva la publicación de un bloque apenas sea minado. Resulta más eficiente ante un ataque de *selfish mining* que las defensas propuestas hasta el momento de su publicación.

Nojoumian et al. [NGN+19] analizan el proceso de minado desde el punto de vista de la teoría de juegos y proponen la utilización de reputación de los mineros. Proponen que el comportamiento honesto de los mineros sea un equilibrio de Nash.

## 1.3 Modelo

### 1.3.1 Comportamiento malicioso

Siendo Bitcoin una red p2p, cualquier nuevo integrante de ella puede comportarse con:

- **Comportamiento honesto**: siguiendo un comportamiento completamente cooperativo con el resto de la red. Éste es el que siguen los clientes estándar de Bitcoin, ver `https://en.bitcoin.it/wiki/Software#Bitcoin_clients`.
- **Comportamiento malicioso**: realizando variaciones al comportamiento honesto para beneficio propio o solo para perjudicar a la red. Un ejemplo es decidir no propagar ningún bloque no minado por él.

Para intentar garantizar el correcto funcionamiento de la red, Bitcoin posee:

- **Mecanismos de prevención de ataques**: por ejemplo, procesar de a un mensaje por *peer* para evitar que un solo nodo lo monopolice.
- **Mecanismos de incentivación de comportamiento honesto**: el comportamiento que más convenga seguir a los nodos debería ser el comportamiento honesto. Existen distintas herramientas para eso: penalizaciones por comportamiento indebido, recompensas monetarias distribuidas apropiadamente, entre otras. Un ejemplo son las *rewards* otorgadas a los mineros por producir bloques, que incentivan su participación en la red. Otro ejemplo son las comisiones a los mineros por incluir transacciones, que incentivan a que los bloques tengan la mayor cantidad de transacciones posibles.

Basado en estos mecanismos de incentivación, se hace la suposición de que en Bitcoin mientras haya una mayoría (en términos de *hashing power*) honesta, los incentivos son suficientes como para que a los mineros les convenga tener un comportamiento honesto. Un ejemplo de esta incentivación es que, si un nodo decide no compartir inmediatamente un bloque minado, entonces el resto de la red no va a minar sobre ese bloque, lo cual parecería disminuir las chances de que éste resulte confirmado en la blockchain. En el trabajo de Eyal y Gün Sirer [ES18] se demostró que los incentivos actualmente utilizados no son suficientes, y que pueden ser explotados para hacer un ataque de *selfish mining*.

### 1.3.2 *Selfish mining*

El ataque de *selfish mining* consiste en aprovecharse de la retención de información. Siendo un nodo honesto, uno apenas genera un bloque lo propaga al resto de la red, siendo un minero *selfish*, uno no siempre lo hace; entonces el minero *selfish* conoce más bloques que los honestos, conoce tanto los de ellos como sus bloques privados. Esto le permite, por ejemplo, que si el minero *selfish* minó cinco bloques adicionales a los que conoce el resto de la red y no se los compartió, entonces todo el *hashing power* de la red será desperdiciado minando sobre una cadena inválida, la mejor *chain* será la *chain* privada que solo minero *selfish* conoce. En cambio, todo el *hashing power* del minero *selfish* será útil, esta desproporción le da una gran ventaja sobre el resto de la red.

#### Variaciones del ataque

Dada la estrategia mencionada, existe una familia de posibilidades de cómo pueda funcionar:

- Mediante la determinación de:
  - Qué bloques se propagan
  - Sobre qué bloque se mina
- De acuerdo a distintos eventos:
  - Minero *selfish* minó un bloque
  - El resto de la red minó un bloque

En Eyal y Gün Sirer [ES18], se propone la primera variante, el *selfish mining*, pero existen otras, como por ejemplo el *stubborn mining* presentado en Nayak et al. [NKMS16]. En este trabajo se explora únicamente el *selfish mining*.

#### Parámetros del ataque

El ataque de *selfish mining* posee dos parámetros que controlan la estrategia:

- $\alpha$: porcentaje de *hashing power* que tiene el *selfish miner* respecto al de toda la red. Un atacante podría incrementar el valor de este parámetro mediante la compra de más o mejor hardware y así tener más poder de cómputo. A lo largo del trabajo consideraremos siempre $\alpha < 0{,}5$, para respetar la mayoría del sistema es honesta. Esta restricción es habitual tanto en la práctica como en la mayoría de los trabajos analizando la seguridad de Bitcoin, ya que se conoce que la criptomoneda sería insegura por ser vulnerable al ataque del 51 $\%$[7].
- $\gamma$: parámetro que describe la conectividad del *selfish miner* con el resto de la red. Esta magnitud depende de la topología de la red de Bitcoin. Su cálculo depende del resultado de la competencia entre dos bloques con la misma dificultad total:
  1. Un bloque generado por algún nodo del resto de la red
  2. Y un bloque generado por el *selfish miner*, cuya propagación solo comienza al haber recibido el bloque anterior

  Entonces, $\gamma$ será el porcentaje de la red (en términos de *hashing power*) que recibe primero el mensaje del *selfish miner* antes que el otro bloque. Un atacante podría incrementar el valor de este parámetro mediante una mejor conectividad con la red, como por ejemplo teniendo una mayor cantidad de *peers*.

[7] `https://en.bitcoin.it/wiki/Majority_attack`

## Descripción del ataque

Vamos a suponer que un único minero está realizando el ataque con un comportamiento *selfish* y que el resto de los nodos sigue un comportamiento honesto. Luego, el minero *selfish* mantendrá parte de su *chain* de forma privada (sin propagarla) y minará siempre sobre ésta, mientras que los mineros honestos, siguiendo el protocolo, minarán sobre el *fork* de mayor dificultad total de la red (eventualmente, desempatando por el bloque que les llegara primero).

En la figura 1.4 se presenta la máquina de estados que representa el ataque. Sus estados $s = 0', 0, 1, 2, ...$ representan cuanto más larga es la cadena privada del *selfish miner* respecto de la mejor cadena conocida por el resto de la red. Se empezará entonces en el estado $0$, distinguiendo entre $0$ donde el minero *selfish* tiene su *chain* privada vacía, y $0'$, cuando no. Sus transiciones representan el minado de un bloque por algún subconjunto de los mineros, teniendo una probabilidad asociada de que sucedan. Por ejemplo, desde el estado $0$ hay probabilidad $\alpha$ de que el minero *selfish* mine un bloque, que lo añadirá a su *chain* privada, pasando a tener ventaja de $1$ ($s = 1$) sobre el resto de la red. Por otro lado, el resto del *hashing power* $(1 - \alpha)$ corresponde a los mineros honestos, que con esa probabilidad minarán un bloque, se propagará a toda la red y todos los nodos lo importarán, incluyendo el minero *selfish*, por lo que la ventaja se mantendrá en $0$ ($s = 0$).

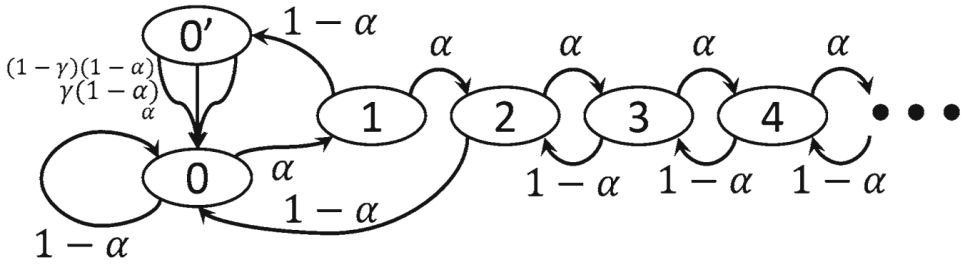



**Figura 1.4:** *Representación teórica del* selfish mining*: sus estados son la ventaja que tiene el atacante sobre la red (en número de bloques) y las transiciones se producen en base a la producción de bloques (tanto del atacante como el resto de la red).*

Entonces, en los estados $s = 0, 1, 2, ...$, si el minero *selfish* mina un bloque (con probabilidad $\alpha$), lo guardará en su *chain* privada y aumentará su ventaja sobre el resto de la red a $s+1$. En los estados $s = 3, 4, ...$ si un minero honesto mina un bloque (con probabilidad $1 - \alpha$), el minero *selfish* mantendrá su *chain* privada pero con una ventaja de $s - 1$. Si lo realizan en el estado $s = 2$ entonces el minero *selfish* propagará toda su *chain* privada para ganarle al resto de la red, y su ventaja pasará a $0$. Si lo hacen en el estado $s = 0$, toda la red importará ese bloque y la ventaja se mantendrá en $0$.

Si esto sucede en el estado $s = 1$, entonces el minero *selfish* y el resto de la red pasarán a estar empatados, el minero *selfish* propagará toda su *chain* privada, generándose una transición al estado $0'$. En éste, los mineros honestos estarán divididos entre los que recibieron primero la cadena del *selfish miner* ($\gamma$ de todos ellos) y los que recibieron primero la *chain* honesta ($1 - \gamma$ de ellos). Entonces, se pueden dar tres transiciones a partir de este estado, todas terminan en el estado $0$:

1. el minero *selfish* mina un bloque (con probabilidad $\alpha$) y gana el empate.

2. los mineros honestos minan un bloque sobre la *chain* del *selfish* (con probabilidad $\gamma(1 - \alpha)$)

3. los mineros honestos minan un bloque sobre la *chain* de los honestos (con probabilidad $(1 - \gamma)(1 - \alpha)$).

**Altura vs dificultad total** Algo interesante de la descripción anterior (que es la misma utilizada por Eyal et al. [ES18]) es que se utiliza la altura de los *forks* para compararlos y determinar cuál es mejor, y no la dificultad total, que es el valor realmente utilizado por los clientes de Bitcoin.

Pero, en el caso en que el *hashing power* de la red sea constante, la dificultad de minado solo va a tener pequeñas variaciones alrededor del valor que corresponda para este poder de cómputo (como se describirá en el experimento de control de la sección 2.3.1). Éste será el caso en los experimentos realizados en este trabajo, así que esta diferencia en la comparación no impactará en los resultados obtenidos.

Pese a ello, el hecho que solo se ajuste la dificultad cada 2016 bloques da la intuición que su potencial impacto en el resultado del ataque sería mínimo.

## Consecuencias teóricas del ataque

Según se mostró teóricamente en Eyal et al. [ES18], la ventaja del *selfish miner* se traduce en una mejora de la proporción de bloques minados por el minero *selfish* de la *main chain* eventualmente confirmada por la red. Si el atacante se hubiese comportado de forma honesta, la proporción de bloques minados por él sería $\alpha$, mientras que comportándose de forma *selfish*, ésta pasará a ser:

$$P = \frac{\alpha(1-\alpha)^2(4\alpha + \gamma(1-2\alpha)) - \alpha^3}{1 - \alpha(1 + (2-\alpha)\alpha)} \tag{1.1}$$

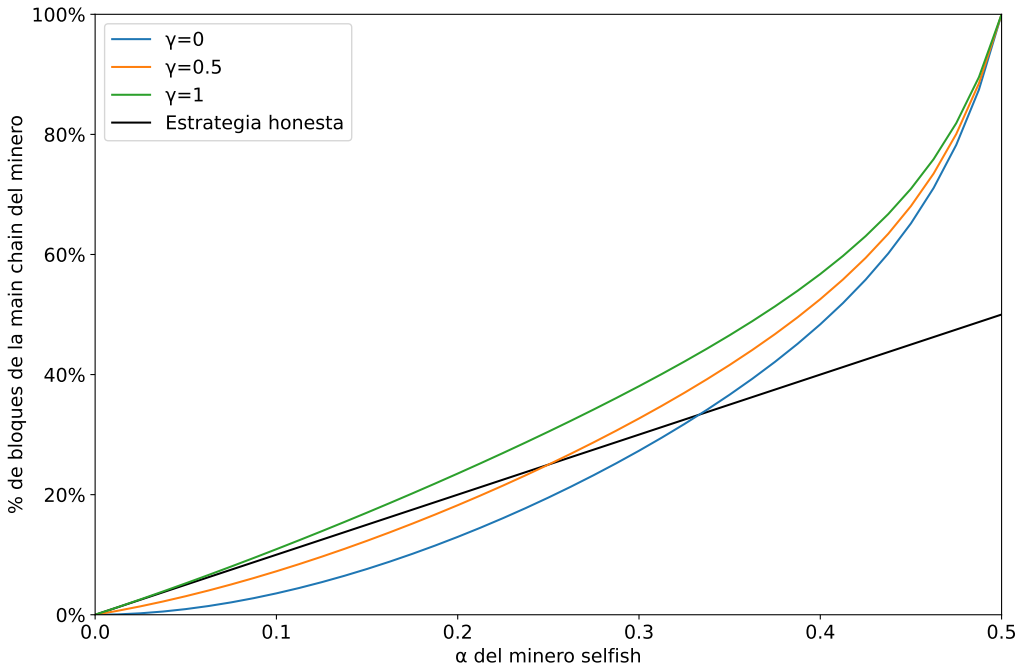



**Figura 1.5:** *Comparación de la proporción de bloques minados siguiendo estrategia honesta y* selfish: *con $\alpha > 1/3$ siempre conviene la estrategia* selfish, *con un valor menor depende de la conectivad a la red (i.e., $\gamma$).*

En la figura 1.5 se presenta de manera gráfica la comparación entre ambas proporciones. Como se puede observar, existen combinaciones de $\alpha$ y $\gamma$ donde la estrategia de *selfish mining* resulta mejor que la estrategia honesta (por ejemplo $\gamma = 0$ y $\alpha > \frac{1}{3}$). En estos casos, el *selfish miner* logra incrementar el control que tiene sobre la criptomoneda, lo cual trae consigo:

- Ganar mayor plata, correspondiente a cada bloque y la comisión de sus transacciones. Es este motivo por el que Eyal et al. [ES18] concluyen que el protocolo de Bitcoin incentiva este comportamiento antes que el comportamiento cooperativo.

- Mayor poder para atentar contra el funcionamiento de la red, como por ejemplo, censurar transacciones al no incluirlas en sus bloques.

# 2

# Metodología

## 2.1 Simulador basado en `SimGrid`

Como se muestra en la figura 2.1, existen varias alternativas posibles para la realización de experimentos sobre redes de Bitcoin. Cada una de ellas consistirá en la decisión de modelado o uso del cliente real para las distintas facetas que componen a los nodos de Bitcoin: el minado de bloques, la aplicación, el protocolo de comunicación de red, y la representación de la red o topología P2P.

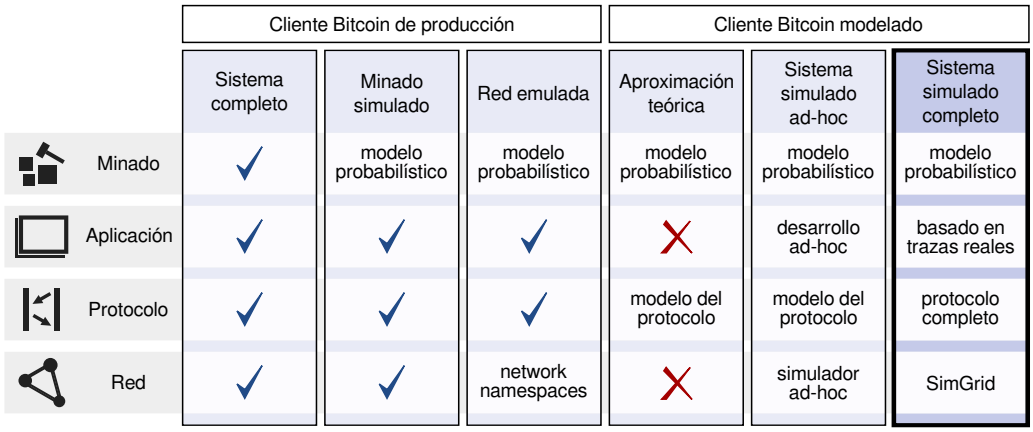



**Figura 2.1:** *Diferencias en el comportamiento de modelos simulados y emulados de la red de Bitcoin, con distintas alternativas para cada uno.*

El primero de los enfoques posibles es el uso de un cliente productivo de Bitcoin, con algunas de sus facetas posiblemente reemplazados por modelos. Algunos de los candidatos a ser reemplazados son el protocolo de minado y la red subyacente entre los nodos de Bitcoin. El segundo de los enfoques plantea partir de un modelo de Bitcoin con cada una de sus facetas abordadas por distintas aproximaciones, más o menos cercanas al comportamiento real de los nodos.

Luego, la diferencia entre las posibilidades planteadas está guiada por cuán cercano se está a la red productiva, y cuán costoso será realizar las simulaciones. Si realizamos experimentos con clientes productivos, los resultados obtenidos serán exactos, pero muy costosos en términos de recursos, limitando el tamaño de los escenarios que se podrán estudiar. En el otro extremo, al utilizar todos modelos, las simulaciones son fácilmente escalables, pero los resultados serán menos fidedignos.

En el caso de este trabajo, el objetivo será realizar simulaciones sobre topologías del tamaño de la red

productiva de Bitcoin. Entonces, iremos por la última de las variantes ilustradas en el esquema 2.1, haciendo uso del *framework* de `SimGrid`.

El simulador utilizado fue desarrollado partiendo del utilizado por Kevin Piotrkowski [Pio19] en su trabajo de tesis, realizándole algunos cambios requeridos para los experimentos planteados en este trabajo. En las siguientes secciones describiremos el comportamiento del simulador, incluyendo tanto la base de la que se comenzó como las alteraciones realizadas.

### 2.1.1 Modelado de Bitcoin

La interfaz del *framework* de simulación `SimGrid` utilizada en el simulador es *S4U*, que plantea como modelo el uso de:

- **Actores**: procesos de computación, con su código y memoria propia (en *S4U* en particular se permite también que compartan memoria). Cada parte del sistema debe ser modelado a través de un actor, los cuales se comunican a través de mensajes.

- ***Mailbox***: es la implementación de una cola de mensajes. Un conjunto de actores producen nuevos mensajes y otros que los consumen, ordenando entonces la comunicación entre todos ellos. Un tradicional caso de uso es asociarle un único actor consumidor a cada *mailbox*, disponibilizando entonces el *mailbox* como la única forma de comunicarse con él.

En el modelado de Bitcoin planteado para este trabajo se representa cada nodo de la red con un actor distinto (y sin más actores que estos), siendo estos de alguna de las siguientes clases:

- *Node*: nodo pasivo que no mina bloques, solo los recibe y los propaga; pero sí se encarga de generar transacciones.

- *Miner*: extiende la clase *Node*, generando también bloques siguiendo el comportamiento de los clientes de referencia.

- *SelfishMiner*: extiende la clase *Miner*, reemplazando su estrategia de minado por el *selfish mining*.

Cada conexión direccional entre dos nodos (es decir de un nodo emisor a uno receptor) tendrá un *mailbox* distinto asociado. Al ser las conexiones en Bitcoin bidireccionales, cada par de nodos tendrá entonces dos *mailboxs* asociados. En la figura 2.2 se puede observar un ejemplo del modelado de Bitcoin mediante *S4U*.

Luego de realizar los cómputos necesarios, cada nodo/actor esperará a que suceda alguno de los siguientes eventos (con *Comm:wait_any_for* para mayor eficiencia, más adelante entramos en más detalles):

1. Recibir un mensaje de alguno de sus *mailboxes*: procesándolo y propagándolo (potencialmente con otros mensajes) de ser necesario, volviendo a esperar nuevos eventos después de eso.

2. *Timeout* para la generación de bloques (en caso de ser *Miner* o *SelfishMiner*) o transacciones. También, luego de la generación y propagación, los nodos vuelven a esperar a recibir nuevos eventos. Los tiempos de generación son pre-calculados al realizar su última actividad, en base a la configuración del nodo (como el *hashing power* para los mineros), modelos de probabilidades asociadas a los eventos y un generador de números aleatorios.

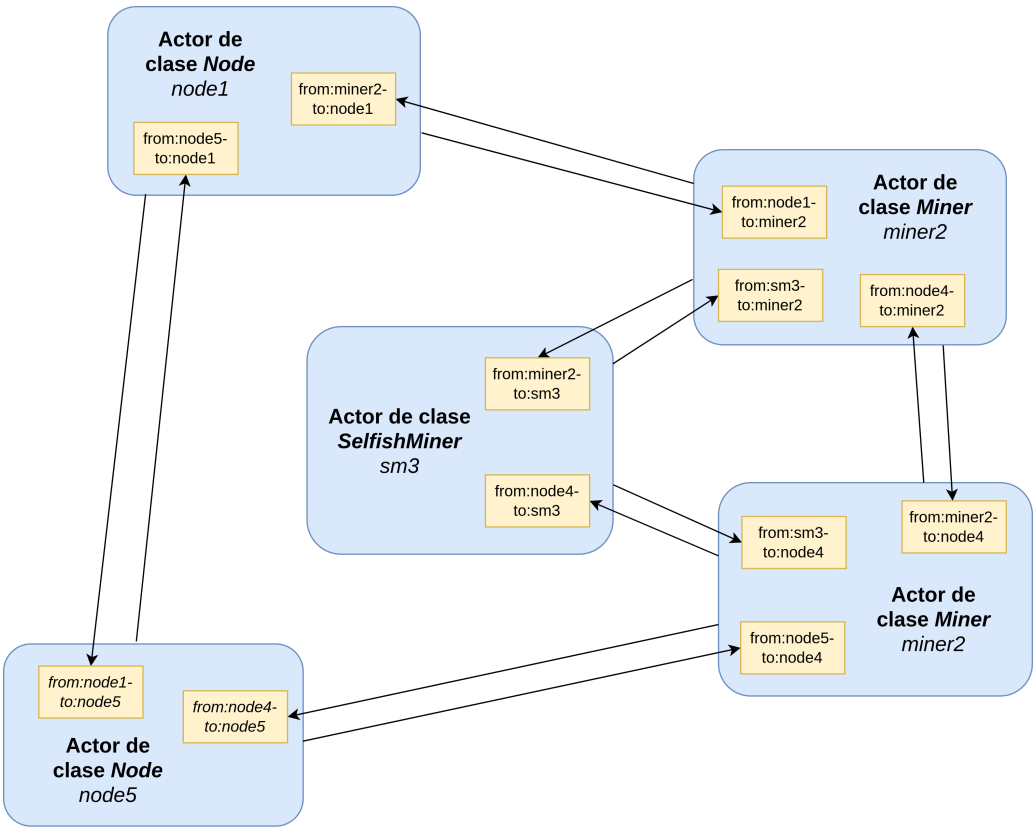



**Figura 2.2:** *Ejemplo de modelado de una red de Bitcoin como actores de S4U: cada actor representa un nodo, conteniendo cada uno diferentes* mailboxs *utilizados para representar las conexiones direccionadas entre nodos.*

### 2.1.2 Utilización

Para la instanciación del modelo planteado en la sección anterior, será necesario plantear la topología física y lógica a utilizar (como se describió en la sección 1.1.4). La topología lógica determina los nodos a utilizar y sus conexiones: es decir los actores y *mailboxs* involucrados; mientras que la topología física determina las latencias de las conexiones lógicas.

La configuración de ambas topologías esta implementada por `SimGrid`, configurable mediante:

- Un archivo de plataforma **platform.xml**: describe la topología física, incluyendo los *hosts* usados (con sus velocidades de procesamiento) y las conexiones (con su *bandwidth* y latencia). Se facilita de *script* **createPlatformXml** para su generación aleatoria a partir de ciertas configuraciones, como el tipo de topología que se desea que tenga el grafo resultante.

- Archivos de *deployment*, que incluyen a:

  - Un archivo **deployment.xml** que especifica los actores a utilizar, sobre qué *host* se ejecutan y qué tipo de clase utilizan (*Node*, *Miner* o *SelfishMiner*).
  - Un archivo **ctg_data** con las configuraciones para la generación de transacciones.
  - Un archivo **node_data-i**, **miner_data-i** o **selfish_miner_data-i** por cada nodo dependiendo de su tipo, con los parámetros específicos del nodo: su *hashing power*, *peers* lógicos, entre otras cosas.

  También se facilita un *script* **createDeploymentXml** para su generación aleatoria dependiendo de ciertos parámetros (como el tipo de topología a usar).

Luego, el uso de nuestro simulador requerirá de los pasos (que se pueden observar en la figura 2.3):

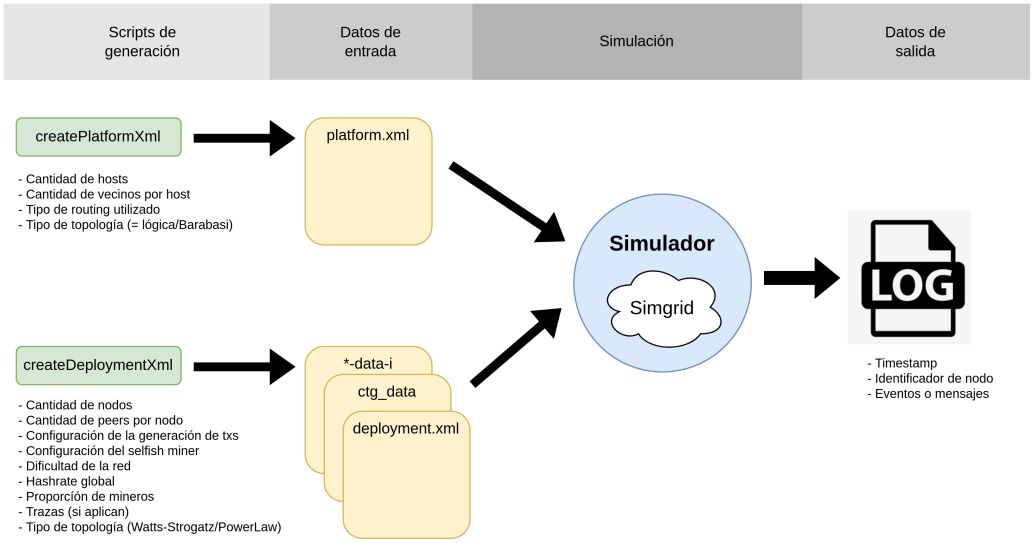



**Figura 2.3:** *Utilización del simulador en base a* `SimGrid`*: corriendo los* scripts *de generación se obtienen los datos de entrada necesarios para la ejecución de la simulación, a partir de la cual se obtiene un archivo de* log *con los eventos transcurridos durante ésta.*

1. Generar la topología física (con *createPlatformXml*) y lógica (con *createDeploymentXml*) deseada.

2. Ejecutar nuestro simulador pasándole las topologías como parámetro, incluyendo también la duración deseada, configuraciones de *logging* y *seed* para alimentar el generador de números aleatorios (esto permite repetir la misma secuencia de eventos, que es útil para ciertos escenarios).

3. Del paso anterior se obtiene un archivo de *logs* con todo lo ocurrido durante la simulación.

### 2.1.3 Otras funcionalidades de interés

A continuación se describen algunas funcionalidades de importancia para el trabajo que fueron añadidas como extensiones al simulador.

**Eventos de las simulaciones** Para facilitar el análisis de lo ocurrido durante una simulación, adoptamos el sistema de *logging* propuesto por De Carli [DC19], con algunas modificaciones. Éste consiste en registrar todos los bloques que procesa la capa de consenso de los nodos, permitiendo así obtener tanto todos los bloques producidos como también derivar cual es la *main chain* de cada uno de los nodos.

De esta forma, cada nodo registra tres eventos:

1. La importación de un bloque minado por el nodo.

2. La recepción de un bloque de los *peers* de un nodo, siempre y cuando no sea huérfano ni previamente conocido.

3. El minado de un bloque de forma *selfish*.

Asociando a cada uno de estos la información:

*Hash del bloque | Altura del bloque | Hash del padre del bloque | Timestamp del minado del bloque*

En los clientes honestos, el minado de un bloque y su posterior importación por la capa de consenso se hacen instantáneamente por lo que no es necesario distinguirlos. Para el caso de los clientes *selfish*, parte su algoritmo de minado implica el guardado de bloques producidos, es decir, se mina el bloque (paso 3), sin importarlo ni compartirlo (paso 1) hasta que se den ciertas condiciones.

**Soporte para *selfish mining*** Se extendió la implementación originalmente planteada por Piotrkowski [Pio19], que tenía comportamiento *selfish* pero no seguía completamente la especificación planteada por Eyal et al. [ES18]. En el capítulo 1.3 entraremos en más detalles acerca del comportamiento implementado.

**Soporte para redes de 10 mil nodos** Se hizo foco en el desarrollo para lograr que sea más eficiente la ejecución de experimentos a gran escala,y que, entonces, éstas requieran menos tiempo. Se tomó como escenario objetivo utilizar una red de 10 mil nodos de tamaño (similar a la red de Bitcoin). A grandes rasgos:

- *Se evaluaron las estructuras de datos utilizadas y su manejo*: Por ejemplo, los nodos mantienen un registro de los bloques que conocen, y utilizan este registro para la búsqueda de bloques conocidos y la inserción de nuevos. Se realizaron optimizaciones sobre ésta y otras estructuras, eligiendo mejores formas de representarlas y mejores algoritmos para procesarlas.
- *Se mejoró la forma en que cada nodo detecta si existe algún mensaje a procesar o alguna acción a realizar*: Cada nodo inicialmente realizaba *polling* sobre sus *mailboxs*, con algunas optimizaciones para ahorrar tiempo cuando todos las consultas de todos nodos resultaban en que no había mensajes a procesar. Ésta fue reemplazada por el uso de la funcionalidad nativa de `SimGrid` de *Comm:wait_any_for*, que permite a cada nodo bloquearse hasta recibir algún mensaje de alguno de sus *mailboxs*. Este bloqueo de forma nativa no solo disminuye los recursos utilizados por cada actor, sino que permite al *framework* saltear los tiempos muertos donde todos los actores se encuentren bloqueados, disminuyendo notoriamente el tiempo total que requiere las simulaciones.

## 2.2 Magnitudes estudiadas

En base a los eventos de las simulaciones descritos en la sección 2.1.3, se utilizarán una serie de métricas para analizar diversos aspectos interesantes de los experimentos.

### 2.2.1 Proporción de bloques minados por un nodo

Esta magnitud describe cuál fue el porcentaje de los bloques de la blockchain fue minado por un determinado nodo. Tal como se presentó en el capítulo 1.3, ésta es la medida que un *selfish miner* está intentando incrementar al realizar su ataque.

**Cálculo** Dado un nodo $N$, y sea $M$ la *chain* a la que se llegó a consenso, la proporción de bloques minados por este nodo se puede calcular como:

$$M_N = \Big\{ b_i | b_i \in M \wedge miner(b_i) == N \Big\} \tag{2.1}$$

$$P_N = \frac{|M_N|}{|M|} \tag{2.2}$$

Primero se calcula $M_N$ (en 2.1), el filtrado de los bloques de $M$ donde su minero es $N$. Teniendo los bloques minados por $N$, se obtiene la proporción $P_N$ respecto del total (en 2.2) mediante la división del tamaño ($|\cdot|$) de ambos conjuntos.

### 2.2.2 *Wasted hashing power* de un subconjunto de los nodos

Como se presentó en el capítulo 1.1.2, los bloques minados pueden terminar estando en la *main chain* o como bloques *stale*. Ambos tipos de bloques requieren de la utilización de *hashing power* para producir una *PoW* correcta; los hashes usados para producir bloques *stale* serán catalogados como *wasted hashing power*, ya que corresponde con poder de cómputo que terminó siendo desperdiciado debido a que el bloque no quedó en la *main chain* y el minero que lo produjo no recibió ninguna *reward*.

En nuestro análisis, esta medida resulta muy útil ya que servirá para entender los alcances del funcionamiento del ataque de *selfish mining*. Los subconjuntos de nodos que analizaremos serán: el *selfish miner*, los mineros honestos y a todos los nodos (el *wasted hashing power* total).

Esta medida fue inicialmente presentada por De Carli [DC19] como parte de su trabajo de tesis y solo ha sido extendida para poder determinar a qué tipo de nodo corresponde el *hashing power*.

**Cálculo** Sea $T$ el conjunto de todos los nodos de la red y $N$ un subconjunto de estos ($N \subseteq T$). Llamaremos $d_i$ al tiempo que el nodo $i$ pasó desperdiciando su *hashing power* y $h_i$ al *hashing power* que éste posee (en hashes por segundo). Como se puede observar en la figura 2.4, $d_i$ será definido como la suma de los tiempos donde:

- **El nodo $i$ mina sobre un bloque viejo**: esto ocurre cuando otro minero ya produjo un bloque de altura $B + 1$, mientras que el nodo $i$ sigue minando sobre el bloque de altura $B$ (generalmente por demora en transferencia de información en la red).
- **El nodo $i$ mina sobre un bloque *stale***: es decir, todo el minado realizado sobre *forks* que no terminan siendo la *main chain*.

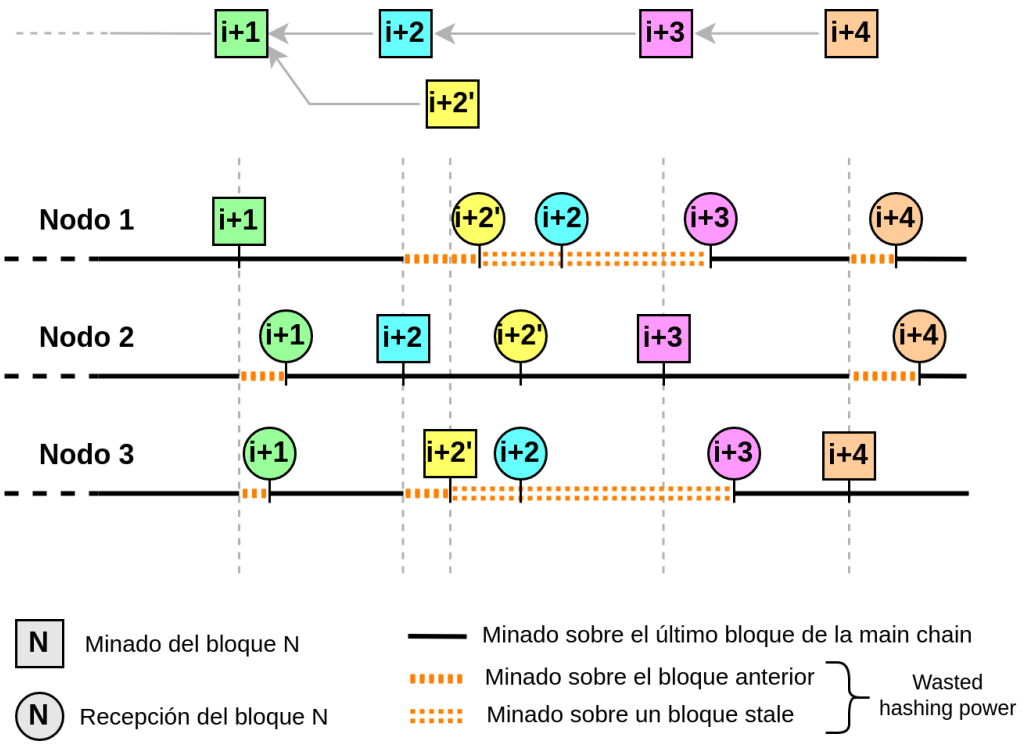



**Figura 2.4:** *Ejemplo del* wasted hashing power *de cada minero (aplicado al ejemplo de la figura 1.3): las líneas en naranja reflejan el único momento donde cada nodo desperdició su* hashing power.

Entonces, calcularemos primero el *hashing power* total de $N$, $H_N$, y a la cantidad de hashes totales desperdiciados por $N$, $D_N$.

$$H_N = \sum_{i \in N} h_i \tag{2.3}$$

$$D_N = \sum_{i \in N} d_i \cdot h_i \tag{2.4}$$

donde $h_i$ es el *hashing power* del nodo $i$, por lo que sumando sus valores en 2.3 obtendremos su valor total para los nodos $N$. Por otro lado, dado que $d_i$ es el tiempo donde el nodo $i$ desperdició su *hashing power*, la multiplicación $d_i \cdot h_i$ da como resultado el número de *hashes* totales desperdiciados por el nodo $i$, resultando entonces en que su suma en 2.4 sea el número de *hashes* desperdiciados por los nodos $N$.

Luego, dado que el tiempo total considerado en el experimento es $T$, el porcentaje de *wasted hashing power* de $N$ ($W_N$) se puede obtener como:

$$W_N = \frac{D_N}{T \cdot H_N} \tag{2.5}$$

### 2.2.3 *Load centrality*

Dado un grafo, la centralidad es una medida que se calcula para cada uno de sus nodos y representa, en pocas palabras, qué tan importante (para cierta definición de *importancia*) es un nodo dentro del grafo. Siendo el grafo la topología de una red, su centralidad será una medida de la conectividad que cada nodo tiene con el resto.

Uno de los parámetros del *selfish mining* es $\gamma$, que es también puede verse como una medida de centralidad (aunque solo para el atacante). Para los casos donde $\gamma$ es desconocido, utilizaremos medidas de centralidad para su estimación. En particular, debido a lo conceptualmente parecidos que son la definición de *load centrality* y la de $\gamma$ (como se explorará más adelante), nos centraremos exclusivamente en esta medida de centralidad.

**Cálculo para cualquier grafo** Dado un grafo, la *load centrality* $C_L$ de un nodo $p$ será calculada utilizando un proceso de flujo:

$$C_L(p) = \sum_{q \neq p \neq r : q \in V \wedge r \in R_q} \rho_{q,r}(p) \tag{2.6}$$

donde $\rho_{q,r}(p)$ es la cantidad de flujo (enviada del nodo $q$ al $r$) que pasa por $p$, $V$ es el conjunto de nodos del grafo, y $R_q$ es el subconjunto de nodos alcanzables desde $q$.

El flujo utilizado parte de $q$ y se transmite entre nodos de la siguiente forma: para cada unidad de flujo $x$ con destino $r$ que llegue a un nodo $v$, se dividirá $x$ en partes iguales entre todos los vecinos que tengan distancia mínima a $r$. Si hay un solo camino con distancia mínima, todo el flujo irá por ahí.

Por ejemplo, si $x = 1$ y $p$ tiene tres vecinos:

- El primer vecino $v_1$ está a la distancia mínima de $r$: 3;
- El segundo vecino $v_2$ también está a distancia 3, lo que significa que hay otro camino mínimo que lo incluye;
- El tercer vecino $v_3$ está a distancia 5 de $r$

Como $v_1$ y $v_2$ están a distancia mínima de $r$, cada uno recibirá $x = 0{,}5$, mientras que $v_3$ recibirá 0 ya que no es parte de un camino mínimo entre $q$ y $r$. Este proceso se repite para cada nodo, y el flujo o suma total que pase por cada nodo $v$ se considerará la *carga* (o sea, la *load centrality* de $v$).

Esto resulta en que la *load centrality* sea un valor entre 0 y 1.

**Cálculo para una topología física y lógica de una red** Dada una red de nodos $N$ con topología física $G_F = (N, E_F)$, siendo sus ejes pesados por alguna medida de distancia entre nodos, y dada su topología lógica $G_L = (N, E_L)$, con ejes no pesados. Calcularemos primero la topología lógica pesada, agregando a cada uno de los ejes de $E_L$ la distancia mínima entre ellos sobre $G_F$:

$$G_{LW} = (N, E_{LW}) \tag{2.7}$$

$$con\ E_{LW} = \left\{ (e_{src}, e_{dst}, minimumDistance(G_F, e_{src}, e_{dst})) \;\middle|\; (e_{src}, e_{dst}) \in E_L \right\} \tag{2.8}$$

Sobre el grafo resultante $G_{LW}$ se calculará para cada nodo la *load centrality* como fue descrita arriba.

## 2.3 Validación

Debido a la cantidad de cambios introducidos al simulador (descritos en la sección 2.1), como primer paso de la validación se repetirán las validaciones realizadas cuando éste fue inicialmente construido en el trabajo de tesis de Piotrkowski [Pio19].

### 2.3.1 Validación del proceso de minado

Tal como se mencionó en la sección 1.1.2, uno de los componentes del funcionamiento de Bitcoin es cómo se asegura que el tiempo entre bloques se mantenga alrededor de un promedio de 10 minutos, sin importar los cambios que se produzcan sobre la red, es decir, más allá de la variación de *hashing power*.

Al utilizar *PoW*, la cantidad de bloques minados por un nodo está relacionada con su *hashing power* y la dificultad de minado. Si aumentase el *hashing power* de los nodos, entonces aumentaría la cantidad de bloques producidos, ocasionando que el tiempo entre bloques disminuya. Esto ocasionaría que, eventualmente, reaccione el mecanismo de ajuste de dificultad y que ésta aumente, buscando que baje la tasa de minado de bloques por los nodos.

En este experimento, se validará que este mecanismo de ajuste de dificultad funcione apropiadamente.

**Diseño experimental** Se realizarán dos simulaciones: una de control en la que el *hashing power* de los nodos se mantiene constante y otra en la que se reducirá el *hashing power* de todos los nodos pero solo temporalmente. El objetivo será entonces observar que, ante un *hashing power* constante, el tiempo de minado se mantiene alrededor de los 10 minutos por bloque. Por otro lado, se buscará observar que que, ante cambios del *hashing power* de la red, el tiempo vuelva a ser de 10 minutos luego del ajuste de dificultad.

| 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) |

Tabla 2.1: Parámetros del experimento para la validación del funcionamiento del proceso de minado

En la simulación con reducción del *hashing power*, a las aproximadamente 1755 horas (durante el día 73) del experimento se reducirá a la mitad el *hashing power* de todos los nodos, que será restituido a su valor original a las 3957 horas (casi a los 164 días).

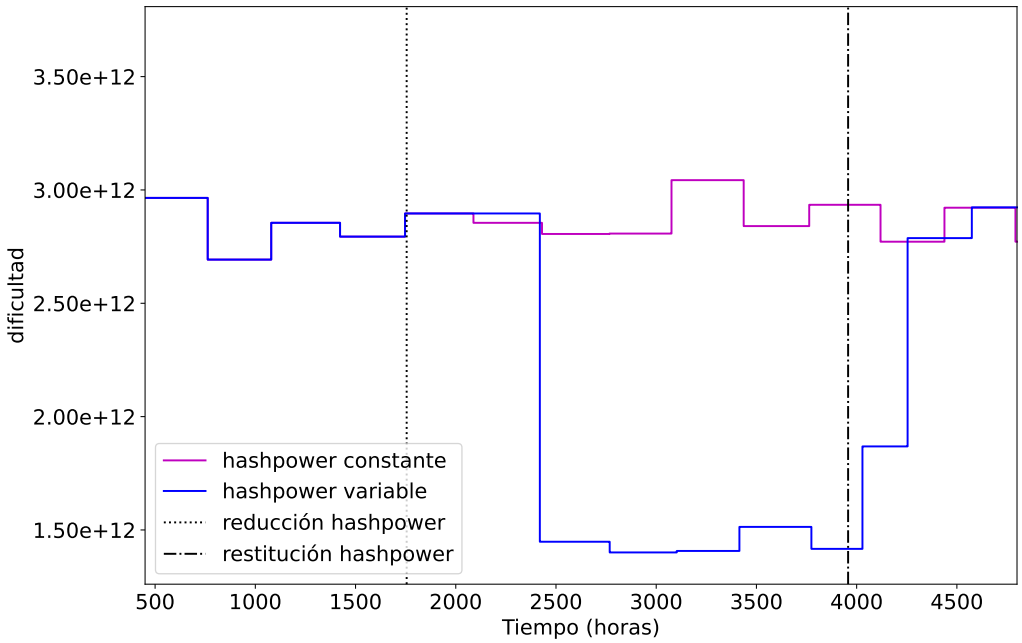



**Figura 2.5:** *Comparación de la evolución del valor de la dificultad de minado de los bloques (a medida que se ejecutan los ajustes de dificultad) respecto de si el* hashing power *de los mineros es constante o pasa a través de una reducción a la mitad y posterior restitución.*

**Variación en la dificultad de minado** En la figura 2.5 se puede observar los valores de la dificultad de minado y como estos fueron impactados por la variación en el *hashing power* de los nodos.

En primer lugar, en la simulación con *hashing power* constante, se puede observar una constante oscilación sin grandes cambios, entre el $2,6^{12}$ y $3,0^{12}$ de dificultad. Esto es esperado para estas condiciones ya que un *hashing power* constante teóricamente indica una dificultad constante, con las oscilaciones observadas siendo consecuencia de pequeñas variaciones probabilisticas de los tiempos de minado.

En segundo lugar, en la simulación con *hashing power* variable se puede observar que la reducción del hash power se hizo apenas después de un ajuste de dificultad. Este ocasionó que en el siguiente ajuste la dificultad cambie a oscilar alrededor de un valor cercano a la mitad del original. Este valor se mantuvo hasta que se restituyó el *hashing power* original de los nodos. Ante este último evento, la dificultad volvió a oscilar alrededor de los valores originales. Los comportamientos descritos previamente son los que uno esperaría de cada una de las dos simulaciones en el cliente de Bitcoin.

Algo interesante a destacar es que, mientras la adaptación de la dificultad fue instantánea al reducirse el *hashing power*, cuando éste se restauró fueron necesarios dos ajustes. Para explicar esto es necesario partir del comportamiento del mecanismo de ajuste de dificultad: cada 2016 bloques, se toman los últimos 2015 bloques[8] y se ajusta la dificultad en base al tiempo requerido para minarlos.

Cuando se restituyó el *hashing power*, no pasó tanto tiempo hasta el siguiente ajuste de dificultad, por lo que de los 2015 bloques considerados la mayoría probablemente hayan tenido entre sí tiempos de 10 minutos, haciendo que el ajuste no sea tan grande como corresponde para los nuevos valores de *hashing power*. Recién al siguiente ajuste de dificultad es que hubo suficientes bloques con tiempo menor a 10 minutos ocasionando un ajuste más severo, llevando la dificultad al valor acorde. Por otro lado, cuando el *hashing power* fue reducido apenas se había dado un ajuste de dificultad, por lo que este impacto en la mayoría de los 2015 bloques considerados para el ajuste, haciendo que sea el único necesario.

[8] Este es un *off-by-one bug* en el código de Bitcoin, se deberían tomar los últimos 2016 bloques

**Variación en el tiempo efectivo entre los bloques** En la figura 2.6 se grafican los tiempos para minar 144 bloques, y cómo estos variaron en la simulación a medida que se redujo y restituyó el *hashing power* de los nodos. Como líneas verticales se presentan los ajustes de dificultad pertinentes al análisis, es decir, todos los que ocurrieron en las inmediaciones de estos cambios en el *hashing power*. Como líneas horizontales se muestran los valores teóricamente esperados para los tiempos de minado de esos bloques, de acuerdo al *hashing power* de la red y la dificultad esperada.

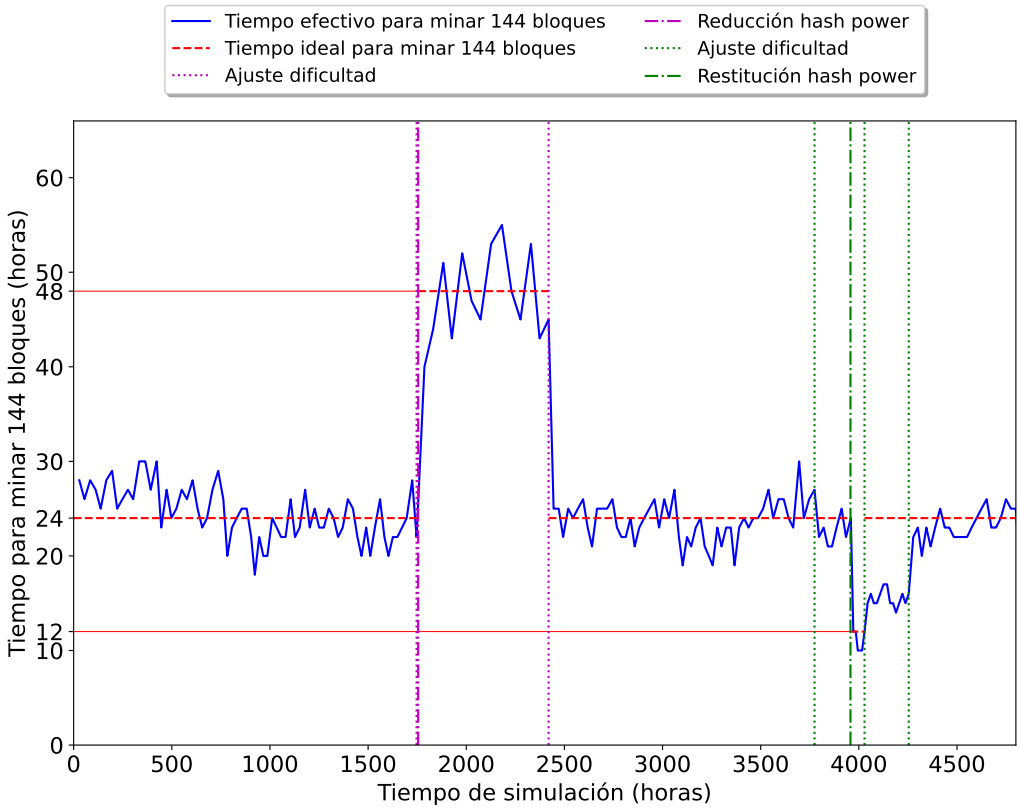



**Figura 2.6:** *Evolución del tiempo requerido para minar 144 bloques, respecto de la reducción a la mitad del* hashing power *de los nodos (a las 1750 horas) y su restitución al valor original (a las 4000 horas). Debido al objetivo de que hayan 10 minutos entre cada bloque, sin cambios de* hashing power, *se esperaría que se produzcan 144 bloques cada 24 horas.*

Tal como se había predicho en el análisis del otro gráfico, se puede observar cómo el tiempo entre bloques en general está oscilando, lo cual tiene como consecuencia las oscilaciones constantes sobre la dificultad de minado.

Esta oscilación es en general sobre las 24 horas, el valor esperado por el tiempo entre bloques configurado de 10 minutos, 144 veces 10 minutos es exactamente 24 horas. Al duplicar el *hashing power* de los nodos, este también se duplicó al valor de 48 horas, mientras que cuando se lo redujo a la mitad, el tiempo se redujo a las 12 horas; siendo entonces los tiempos consistentes con los cambios en el *hashing power*.

Por último, se puede observar en la figura 2.6 el hecho que se requieran dos ajustes para que el tiempo entre bloques vuelva a los 10 minutos después de la restitución del *hashing power*.

Todos estos resultados son consistentes con los visto en la figura 2.5 y con lo esperado para el cliente real.

### 2.3.2 Validación de traza real

La red de Bitcoin progresa ante la ocurrencia de los eventos:

- Un usuario genera una transacción (con cierto tamaño y comisión), que ingresa a la red a través de un nodo.
- Un minero genera un bloque

Dependiendo de su orden se llegará a resultados distintos: variando los bloques aceptados por la red y las transacciones que cada uno de estos contenga. Denominamos como **traza de Bitcoin** a la lista ordenada de estos eventos (conteniendo el momento donde ocurrieron y el nodo donde lo hicieron), con la *main chain* a la que llegó cada nodo luego de su ejecución.

Un mecanismo para evaluar el comportamiento de un simulador es obtener una traza de la red real de Bitcoin, ejecutarla en el simulador y validar que tan cercanos son los resultados esperados con los obtenidos. En esta validación realizaremos justamente eso, contrastando el comportamiento del simulador con la red de Bitcoin a partir de una traza de transacciones y bloques reales.

El foco de la traza utilizada será que los bloques generados posean el número de transacciones observadas en la red real, lo cual validará que se realizó una correcta propagación de ellas desde los usuarios que las generan hasta los mineros que las incluyen en sus bloques.

**Diseño experimental** Los eventos de la traza utilizada se obtuvieron mediante una API provista por Blockcypher[9] que, aparte de la información persistida en la blockchain, contiene fechas aproximadas del momento donde se generaron las transacciones. Siendo el foco de este experimentos las transacciones, simplificaremos la traza elegida, utilizando bloques durante una operatoria de la red sin *forks* (solo se ejecutarán y generarán bloques de la *main chain*) ni congestión, con un promedio de 900 kB de tamaño y 1600 transacciones por bloque.

Para recrear exactamente la traza sería necesario información de la topología (tanto física como lógica) de la red en ese momento, incluyendo la localización de los mineros y de los generadores de cada transacción. Debido a la imposibilidad de tener esta información se usará un tipo de topología similar a la cadena principal de Bitcoin, con todos los nodos distribuyéndose equitativamente la generación de las transacciones (que serán generadas en el momento determinado por la traza).

| 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 2.2: Parámetros del experimento para la validación de inclusión de transacciones según una traza real

**Resultados obtenidos** A continuación se presentan los resultados obtenidos, que comparan el resultado de la simulación con el esperado por la traza ejecutada.

En la figura 2.7 se presenta una comparativa de la cantidad de transacciones que tiene cada bloque, respecto de las esperadas por la traza ejecutada. En general (a excepción de los bloques 7 y 8) no hubo más de una transacción de diferencia. Tal como se observa mejor en 2.8, una de las potenciales explicaciones

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

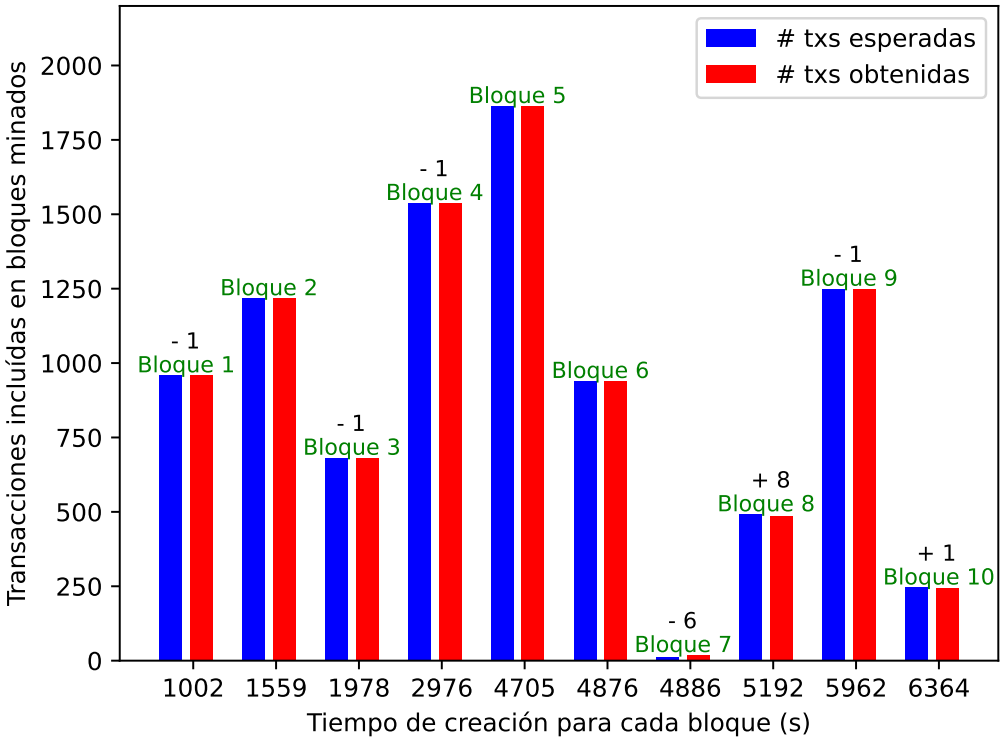



**Figura 2.7:** *Comparación de la cantidad de transacciones por bloque observadas ante la ejecución de la traza real, y la observada en la mainnet de Bitcoin.*

de la mayor diferencia en el bloque 7 y 8 es que estos fueron generados con los menores tiempo desde sus respectivos bloque padre.

En general las diferencias obtenidas pueden ser atribuidas a:

- las fechas de publicación de las transacciones no son exactas, la API utilizada provee la fecha en la que alguno de sus nodos testigos detectó la transacción.
- las diferencias entre las topologías físicas y lógicas usadas, que afectará el tiempo de propagación de las transacciones.

Pese a estas diferencias, en general, comparativamente a la cantidad de transacciones de los bloques, se puede observar una baja diferencia entre la cantidad de transacciones esperadas por la traza real y las efectivamente obtenidas en los bloques de la simulación.

Por otro lado, en la figura 2.8 se presenta la evolución del tamaño del *mempool* de uno de los nodos (estructura presentada en 1.1.2), actuando como representativo de la evolución del *mempool* de todos. Se observa que su tamaño es constantemente creciente, a medida que se generan y propagan las transacciones, a excepción de cuando se procesa un bloque, donde se eliminan del *mempool* todas las transacciones incluidas en éste.

Es importante aclarar que una de las simplificaciones del simulador implementado es que no tiene restricciones al tamaño máximo de su *mempool*, a diferencia de los clientes de Bitcoin donde por defecto es de 300 MB. Esta simplificación no afecta los resultados de la traza ejecutada, ya que el tamaño del *mempool* durante su ejecución se mantuvo lejos de ese tope[10].

[10] `https://jochen-hoenicke.de/queue/#BTC,all,weight,0`

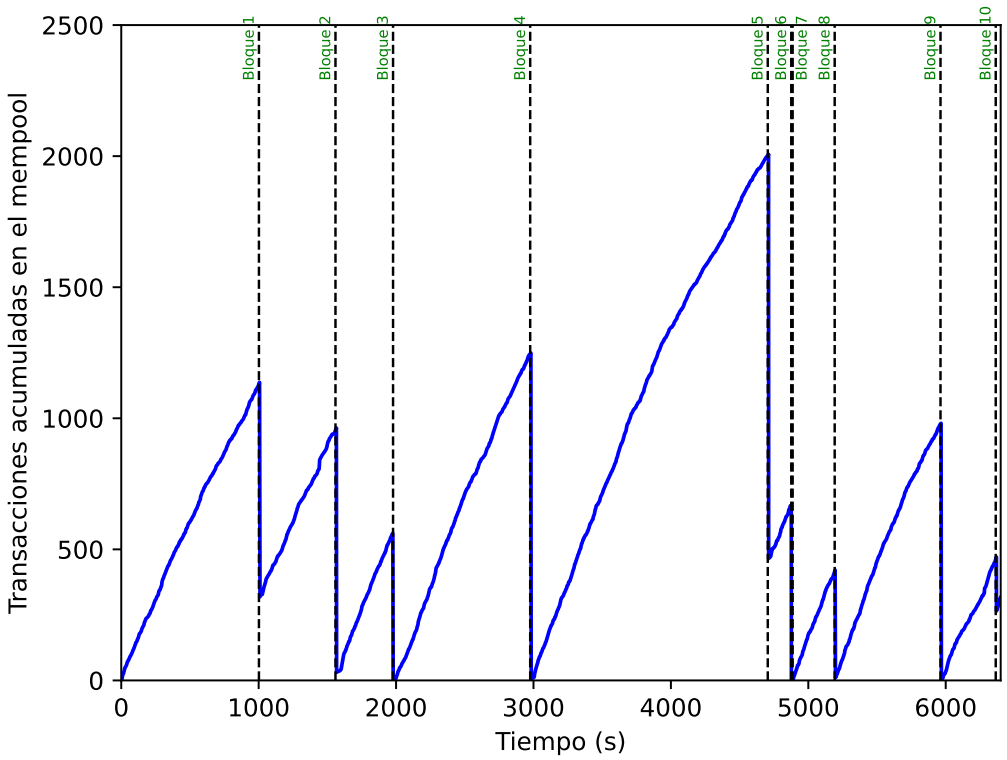



**Figura 2.8:** *Evolución del tamaño del* mempool. *En el mempool se guardan las transacciones que todavía no se incluyeron en ningún bloque, por lo que se espera que su tamaño sea constantemente creciente hasta que se produzca/reciba un bloque.*

### 2.3.3 Validación de la implementación del *selfish miner*

El foco de esta sección es validar que la implementación del *selfish miner* en el simulador sea correcta, es decir, respetando la especificación descrita en Eyal et al. [ES18]. Como se mencionó en el capítulo 1.3, en ese trabajo se describe el comportamiento del sistema ante un *selfish mining* a través de una máquina de estado probabilística (presentada en la figura 1.4).

Recopilando de ese capítulo, las transiciones de esta máquina de estados representan el minado de un bloque por algún subconjunto de los mineros, mientras que el número del estado es la cantidad de bloques que el minero *selfish* le lleva de ventaja al resto de la red.

Existen tres tipos de roles que intervienen en las transiciones (cuya participación depende de los parámetros mencionados en 1.3.2):

- **Minero *selfish***: el minero realizando el ataque. Tiene $\alpha$ probabilidad de minar un bloque en cualquier momento.
- **Mineros honestos cómplices**: solo intervienen en caso que haya un empate entre la cadena honesta y *selfish*; serán los mineros honestos que reciban primero la cadena *selfish* antes que la honesta, y por lo tanto minarán sobre la cadena *selfish*. Serán el $\gamma$ porcentaje de los mineros honestos en promedio, y no siendo necesariamente siempre el mismo subconjunto de los mineros honestos ante cada empate.
- **Mineros honestos inocentes**: el resto de los mineros honestos, teniendo también el resto de las probabilidades de minar un bloque en cualquier momento.

Si se añaden los roles descritos a la máquina de estados, identificando cuál de estos está involucrado en cada una de las transiciones, se obtiene como resultado la figura 2.9.

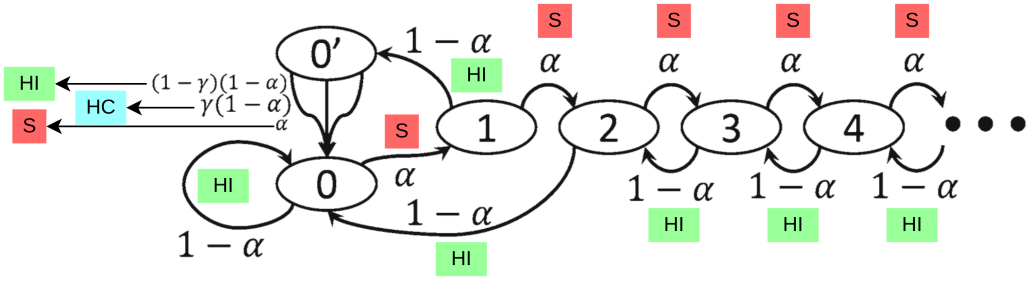



**Figura 2.9:** *Extensión de la figura 1.4, agregándole los roles que poseen mineros que producen los bloques de cada una de las transiciones: HI=minero honesto inocente, S=minero selfish, HC=minero honesto cómplice.*

La idea para la validación del comportamiento del *selfish miner* será entonces comparar la implementación con la máquina de estados presentada:

1. *Generamos trazas sobre la máquina de estados*: estás trazas serán una lista de rondas, con un rol encargado de generar un bloque en cada una, i.e. {Minero *selfish*, Minero honesto inocente, Minero honesto cómplice}

2. *Generamos la main chain esperada*: cada transición de la máquina de estados tiene asociada una modificación de la *main chain* de cada nodo, por lo que esta será calculable a partir de la traza generada.

3. *Ejecutamos la traza generada*

4. *Comparamos el resultado obtenido con el esperado*:

    *a*) Comprobamos que la *main chain* obtenida es la esperada.

    *b*) Verificamos empíricamente que los cálculos de *reward* hechos en Eyal et al. [ES18] sean correctos.

## Diseño experimental

Para esta validación se modificarán los valores de $\alpha$ y $\gamma$ de la misma forma que se realizó en el trabajo de Eyal et al. [ES18]:

- $\alpha$: con valores 0.1, 0.2, 0.3, 0.4, 0.45 y 0.475.
- $\gamma$: con valores 0, 0.5 y 1.

En la siguiente sección entraremos en los detalles pertinentes para explicar la implementación del modelo previamente descrito.

| Configuración del experimento | |
|---|---|
| Topología física | Igual a la lógica |
| Topología lógica | 3 nodos (ver sección sobre *Representación de los roles*) |
| Proporción de mineros | 100 % |
| Longitud de las trazas | 7500 |
| Número de repeticiones | 10 |

Tabla 2.3: Configuración del experimento para validación del *selfish miner*

**Generación de la traza** Se implementó la máquina de estados de la figura 2.9. Con esta, se partirá del estado $0$, y se generarán transiciones aleatoriamente (de acuerdo a las probabilidades de la máquina) hasta obtener la longitud deseada de la traza, obteniendo trazas como la de la figura 2.10.

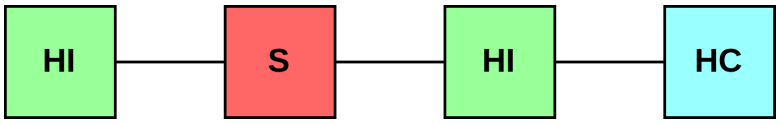



**Figura 2.10:** *Ejemplo de una traza de la 2.9, partiendo del estado* $0$

Como se observa en la figura 2.11, cada ronda de la traza ocasiona una transición de la *main chain* de cada uno de los nodos de cada rol, sea por el minado de un bloque y/o la recepción de uno/varios. En base a ésta, vamos a poder concluir que *main chain* tendrá cada tipo de rol al terminar de ejecutar la traza generada. En el caso de la traza de ejemplo, los nodos de todos los roles terminarán con la misma *main chain* que se ve al tope de la figura, es decir con los bloques $\{1, 2, 3\}$.

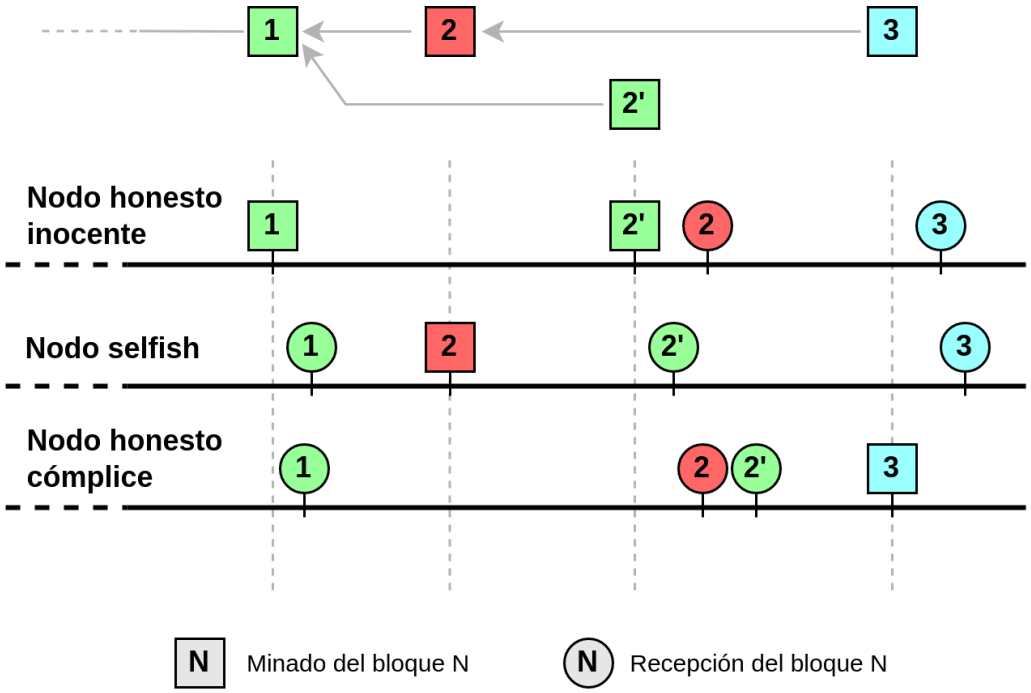



**Figura 2.11:** *Posible ejecución de la traza de la figura 2.10. Se observa como el selfish miner no propaga su bloque 2 hasta dejar de estar en ventaja, y como las diferencias entre las latencias de distribución del bloque 2 y 2' ocasionan que los mineros honestos reciban primero uno u otro (y por lo tanto lo consideren su tope a uno u otro).*

### Ejecución de la traza

**Representación de los roles** Como se busca verificar la implementación del *selfish miner* con respecto a la máquina de estados de la figura 2.9, se plantea una topología que permita representar los tres roles. Desde el punto de vista de la red, el único requerimiento es que ante un empate entre los mineros honestos y *selfish*, los nodos con rol de cómplice reciban primero el bloque del minero *selfish*. La topología presentada en la figura 2.12 cumple con este requerimiento y posee la menor cantidad de nodos (buscando así la representación más simple posible).


**Figura 2.12:** *Topología utilizada para representar de manera simplificada los roles ante un selfish mining. Estando el nodo selfish entre el medio de los honestos le permite controlar totalmente la información que recibe uno u otro.*

El nodo *selfish* será el único que tenga un comportamiento distinto al de los clientes estándar.

**Ejecución** Para la ejecución de las trazas de la máquina de estados se utilizará la herramienta del simulador que permite ejecutar trazas de bloques(presentada originalmente en el experimento de la sección 2.3.2). Con ese objetivo, cada ronda de la traza será asignada al nodo que posee el rol del minero y, para mantener los mismos tiempos de Bitcoin, las rondas estarán separadas por 10 minutos entre cada una.

**Artificialidad del experimento** Es importante aclarar que existen ciertas limitaciones impuestas al modelo que no sucederán en el caso de un ataque real:

- **Simulación artificial**: El minero honesto cómplice interviene en situaciones que dependen del contexto global de la red y no local, solo interviene en caso de un empate. Esto ocasiona que no exista una posible asignación de *hashing power* a los nodos para que se den las trazas esperadas.

- **Roles estáticos**: En un ataque real los nodos con comportamiento honesto irán variando su rol (siendo cómplices o inocentes) dependiendo de quién haya minado el bloque del empate, lo cual no es el caso de este experimento.

Estas restricciones se deben a que el objetivo del experimento fue que se trate de una validación controlada y lo más simple posible del ataque, no que represente precisamente su comportamiento. En el siguiente capítulo se quitarán las limitaciones aquí planteadas y se explorará el comportamiento del ataque en ese contexto.

## Resultados obtenidos

***Main chain*** Se utilizó la *main chain* esperada, de cada simulación, que se obtuvo durante la generación de la traza para comparar con la obtenida al procesar los eventos de los *logs* (presentados en la sección 2.1.3). En todas las simulaciones, la comparación resultó en que ambas *main chains* fueron equivalentes.

**Proporción minada por el minero *selfish*** En la figura 2.13 se observan las proporciones de bloques minados por el atacante y las teóricamente esperadas, siendo las barras de error el desvío estándar. Ambas proporciones fueron siempre muy cercanas, permitiendo concluir que las simulaciones cumplen con los cálculos de *revenue* descritos en el trabajo de [ES18].

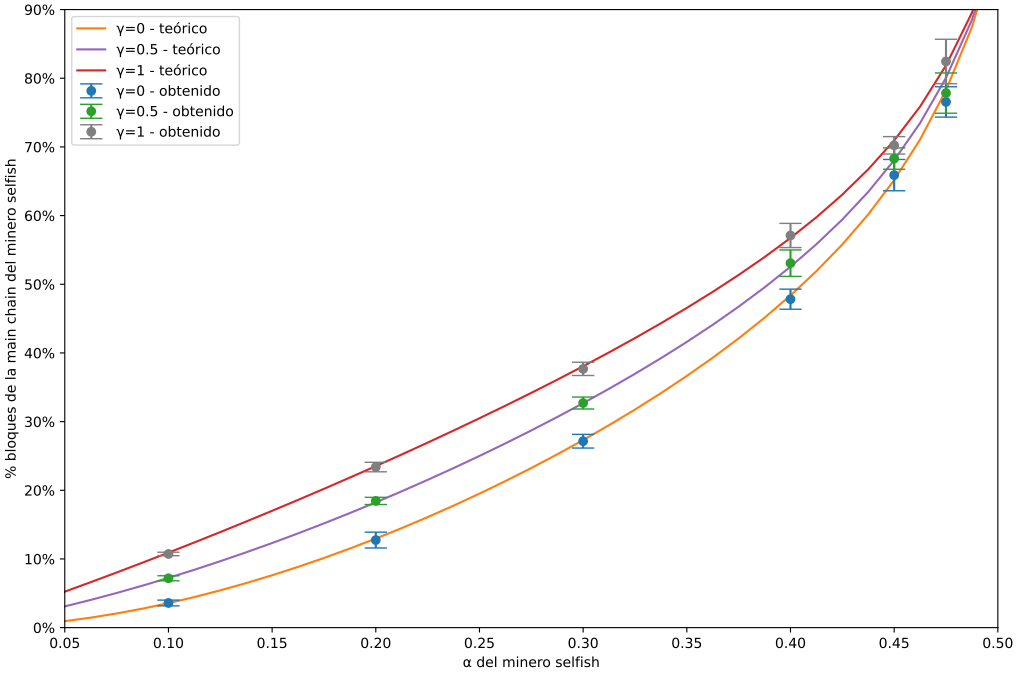



**Figura 2.13:** *Comparación entre la proporción minada por* selfish *(variando el $\alpha$ y $\gamma$) y la teóricamente esperada, donde se observa su gran similitud.*

**Conclusiones generales** Ambas comparaciones dan un gran grado de confianza en que la implementación del *selfish miner* en el simulador respeta la máquina de estados planteada en el trabajo de Eyal et al.

# 3

# Experimentación

En el capítulo previo se realizaron una serie de validaciones para verificar que distintos aspectos del simulador estaban implementado correctamente, incluyendo entre ellos la implementación del *selfish miner*. Pero, los experimentos del *selfish miner* realizados hasta ahora fueron muy controlados y no representativos de un ataque real. En este capítulo se irán quitando restricciones, explorando el comportamiento del ataque para escenarios con complejidad incremental:

1. Experimento con topologías controladas y $\gamma$ conocido, pudiendo así estimar el resultado esperado y compararlo con el obtenido.
2. Experimento con topologías controladas y variando la conectividad, centrandose en el impacto de la conectividad en los resultados del ataque.
3. Experimento a gran escala y variando el $\alpha$ del atacante, donde veremos si los resultados obtenidos en topologías controladas se aplican también en topologías del tamaño de la *mainnet* de Bitcoin.
4. Experimento a gran escala y variando la conectividad, focalizandose nuevamente en el impacto de la conectividad pero en topologías de la escala de Bitcoin.

## 3.1 Experimentos de topologías fijas

En primer lugar, empezaremos evaluando el comportamiento del ataque en topologías simples y chicas para facilitar su análisis, siendo entonces todos los nodos de estas mineros y sin transacciones. Basándose en los valores usados en Eyal et al. [ES18], el parámetro $\alpha$ del *selfish miner* irá variando para ver el impacto, entre: 0.1, 0.2, 0.3, 0.4, 0.45 y 0.475.

En términos de nuestras simulaciones, la topología física de Bitcoin solo es importante por las latencias que introduce en la comunicación a nivel aplicación. Dos nodos podrán estar comunicados en la topología lógica (es decir, son *peers*), pero su latencia variar fuertemente dependiendo del camino que los mensajes tengan que realizar por la capa física. Por simplicidad, en estos experimentos removeremos esta complejidad, siendo la topología física igual a la lógica, con la misma latencia configurada para cualquier par de nodos que sean *peers*.

Por otro lado, cada simulación será de 1200 horas (aproximadamente 7000 bloques) y con 10 repeticiones por configuración, ya que empíricamente ha mostrado tener una baja varianza entre repeticiones. Aparte se mantendrán los 10 minutos promedio entre bloques utilizados por la red de Bitcoin.

| Configuración del experimento | |
|---|---|
| Topología física | Igual a la lógica |
| Topología lógica | Descrito en cada experimento *(con entre 8 y 9 nodos)* |
| Proporción de mineros | 100 % |
| Tiempo simulado | 1200 horas |
| Número de repeticiones | 10 |

Tabla 3.1: Parámetros generales de los experimentos con topologías fijas

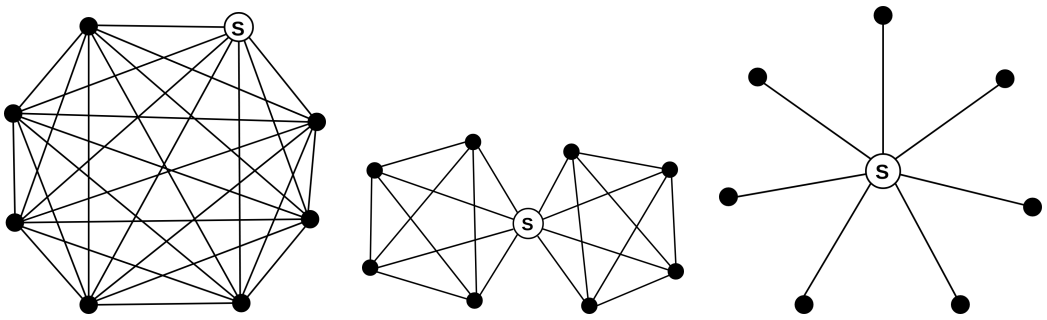
### 3.1.1 Experimento de topologías con $\gamma$ conocido

Una de las dificultades de hacer experimentos respecto del ataque de *selfish mining*, es poder calcular el parámetro $\gamma$ usado (o viceversa, encontrar una topología dado un $\gamma$ elegido). Por este motivo los primeros experimentos serán con topologías donde ya sabemos su $\gamma$ de antemano, permitiendo así variar tanto éste como el parámetro $\alpha$ y observar su impacto en los resultados. Para esto utilizaremos las topologías presentadas en la figura 3.1, explicando a continuación porque poseen el $\gamma$ indicado para cada una. En estas, el nodo marcado con **S** será el único minero *selfish*, con el resto siendo honestos.

Se han elegido estas topologías por su variedad de $\gamma$, siendo casi idénticos a los usados en el análisis de Eyal et al. [ES18]; se reemplazó $\gamma = 1$ por $\gamma \approx 0{,}86$ por ser el primero imposible fuera del marco teórico.

Con una topología tipo clique (figura 3.1(a)), cuando un minero mina un bloque, éste le llegará al mismo tiempo a todos los nodos de la red. En el caso que un minero honesto mine un bloque que ocasiona un empate con un bloque del minero *selfish*, toda la red se enterará al mismo tiempo del bloque honesto; por lo que, para cuando el minero *selfish* propague su bloque, éste será peor por haber sido recibido después. Luego, ningún minero honesto minará sobre la cadena del minero *selfish*, siendo entonces $\gamma = 0$.

En el caso de una topología con dos comunidades (del mismo tamaño) separadas por el *selfish miner* (figura 3.1(b)), éste controlará la comunicación entre ambas. Por lo tanto, si un minero honesto de la comunidad izquierda mina un bloque que ocasiona un empate, toda esta comunidad recibirá primero este bloque (por ser una clique), mientras que el minero *selfish* se asegurará que la comunidad derecha reciba primero su bloque. Entonces, la mitad de los mineros honestos minarán sobre la cadena del minero *selfish* y la mitad sobre la del minero honesto, siendo $\gamma = 0{,}5$




**(a)** *Clique* ($\gamma = 0$) **(b)** *2 islas* ($\gamma = 0{,}5$) **(c)** *Estrella* ($\gamma = \frac{6}{7} \approx 0{,}86$)

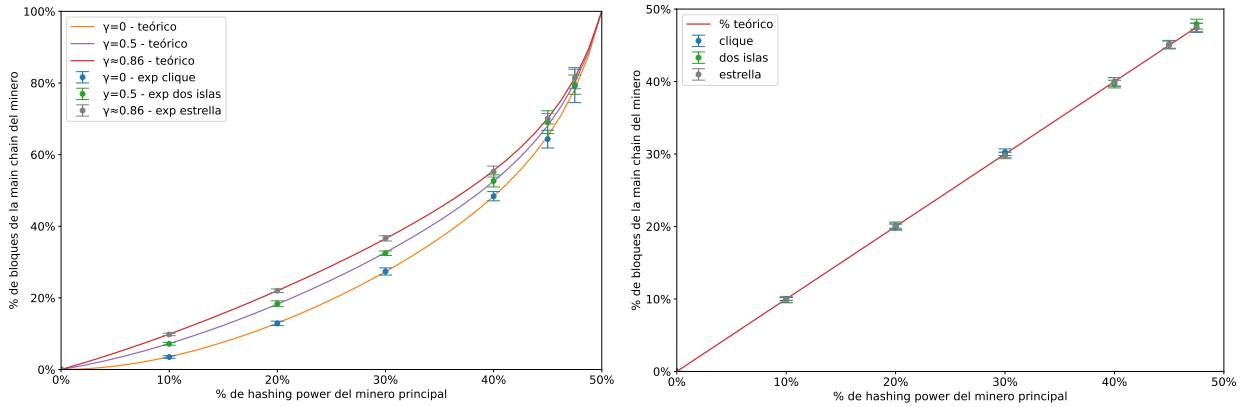
**Figura 3.1:** *Topologías a utilizar: con entre 8 y 9 nodos y variados valores de $\gamma$. El nodo **S** será el único que siga una estrategia selfish, el resto serán honestos.*

Análogamente a lo que sucede en el caso anterior, en la figura 3.1(c), cuando alguno de los mineros honestos de los extremos mine un bloque, el minero *selfish* se enterará primero antes que ningún otro. Luego, ante un caso de empate, solo el minero honesto que generó el bloque lo habrá recibido antes, los seis restantes mineros honestos habrán recibido primero el bloque del minero *selfish*. El $\gamma$ de esta topología será entonces $\frac{6}{7} \approx 0{,}86$.

En la siguiente sección analizaremos los resultados obtenidos al realizar simulaciones variando tanto el $\alpha$ como la topología usada.

Se incluyen también los resultados obtenidos al ejecutar las simulaciones con la misma configuración, exceptuando que el minero utilizará la estrategia honesta en vez de la *selfish*. Es decir, para cada una de las topologías descritas anteriormente, se irá variando el *hashing power* del nodo principal (el que sería el minero selfish en caso contrario). Esto servirá como experimento de control contra el cual contrastar los valores obtenidos para las distintas mediciones.

## Resultados obtenidos




**(a)** *Siguiendo la estrategia **selfish**.* **(b)** *Siguiendo la estrategia **honesta**.*

**Figura 3.2:** *Comparación de la proporción de bloques minados obtenidos y teóricamente esperados, para topologías chicas (número de nodos entre 8 y 9) con diferente conectividad.*

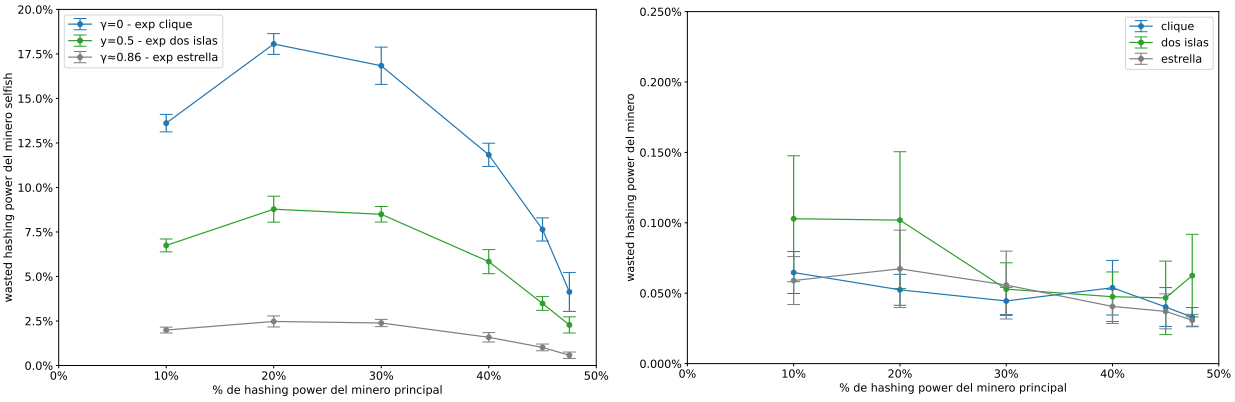
**Proporción minada por el *selfish*** Para el escenario que el minero principal es *selfish*, en la figura 3.2(a) se observa la proporción de bloques minados por el minero *selfish* en las simulaciones, comparándolas con las esperadas según el trabajo de Eyal et al. de acuerdo a los $\alpha$ y $\gamma$ usados. La cercanía entre ambas nos permite concluir que el ataque fue en general tan exitoso como se esperaba teóricamente.

Por otro lado, en la figura 3.2(b) se incluyen los resultados obtenidos para la misma métrica en caso que el minero principal se comporte de forma honesta, contrastando con lo teóricamente esperado, que es que la proporción de bloques minados por el minero sea igual a su proporción de *hashing power*. No se advierten diferencias significativas entre las proporciones obtenidas para las diferentes topologías, y se puede concluir que la métrica se comporta como es esperada, debido a la cercanía de sus valores con los teóricos.

A continuación se presentan los resultados obtenidos al calcular el *wasted hashing power* para tres subconjuntos de nodos: solo el minero principal, para los mineros honestos y todos los mineros. Es importante recordar que definimos *wasted hashing power* de un subconjunto de nodos como el porcentaje de todos los *hashes* realizados por ese conjunto de nodos que ha sido desperdiciado en una simulación. Un *hash* realizado para producir a un bloque $B_{n+1}$ y con padre $B_n$ está desperdiciado si:

- Es sobre un bloque viejo, es decir, ya se produjo algún bloque $B'_{n+1}$ de mayor o igual altura en algún lugar de la red (aunque no le haya llegado al minero en cuestión).
- El bloque $B_n$ sobre el que se está minando va a quedar como *stale*, y entonces no va a ser parte de la *main chain*.

En este experimento nos centraremos en conclusiones más generales e independientes de la conectividad de la red, entraremos en más detalles respecto a éstas en el siguiente experimento.




**(a)** *Siguiendo la estrategia* ***selfish****.*

**(b)** *Siguiendo la estrategia* ***honesta****.*

**Figura 3.3:** *Wasted hashing power del minero principal, para topologías chicas (número de nodos entre 8 y 9) con diferente grado de conectividad. Los porcentajes indican cuanto de los hashes realizados por el minero durante las simulaciones fue desperdiciado.*

***Wasted hashing power* del minero principal** En el experimento donde el minero principal es *selfish*, su *wasted hashing power* nunca superó el 20 %, como se puede observar en la figura 3.3(a). A primer análisis, uno supondría que dado un $\alpha$ o $\gamma$ mayor, el ataque es más exitoso, ocasionando que el *wasted hashing power* del atacante sea menor. Se observa que esto ocurre para $\gamma$ pero no para $\alpha$, el *wasted hashing power* aumenta entre $\alpha = 0{,}1$ y $\alpha = 0{,}2$, mientras que la proporción de bloques minados más que se duplico entre ambos. Una intuición respecto de lo ocurrido podría que el ataque es más exitoso porque, a pesar de que el minero *selfish* malgasta más de su *hashing power*, los mineros honestos lo malgastan aún más. En el siguiente experimento se entrarán en más detalles del comportamiento del ataque que justificarán lo observado.

En contraste, en la figura 3.3(b) del experimento de control se observa que el minero, de haber utilizado una estrategia honesta, hubiese tenido siempre un *wasted hashing power* bastante menor, con promedios siempre menores al 0.25 %. Este desperdicio mucho mayor del atacante obviamente tiene sus consecuencias

en los requerimientos para que el ataque sea rentable, sobre los cuales entraremos en detalle en la última parte de esta sección.

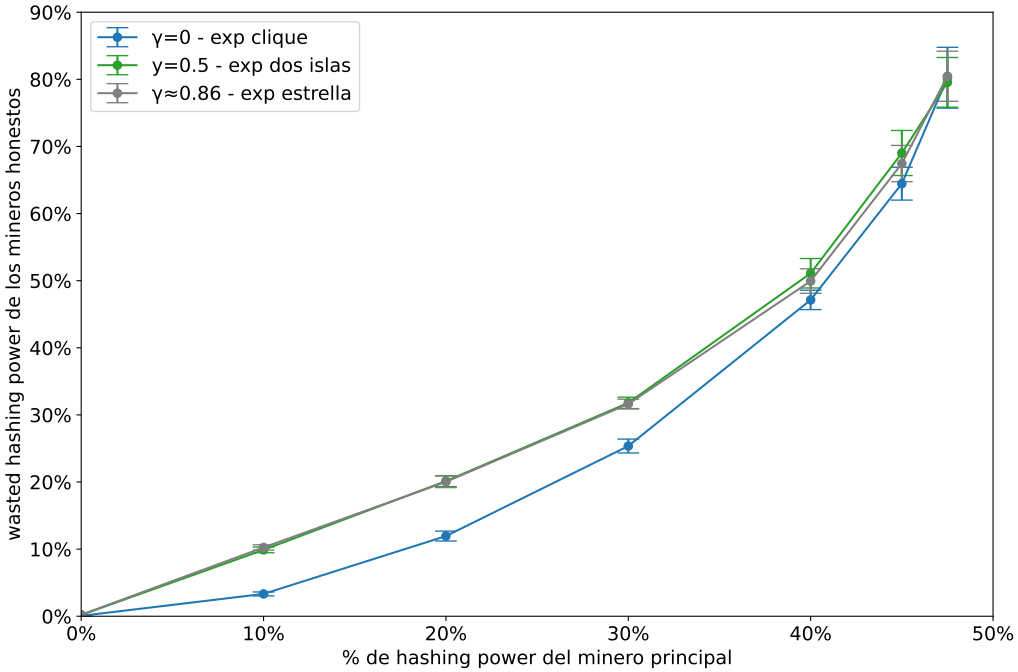



**Figura 3.4:** *Wasted hashing power de los mineros honestos (dado que el minero principal es **selfish**), para topologías chicas (número de nodos entre 8 y 9) con diferente grado de conectividad. Los porcentajes indican la proporción de* hashes *desperdiciados por todos los mineros honestos durante las simulaciones.*

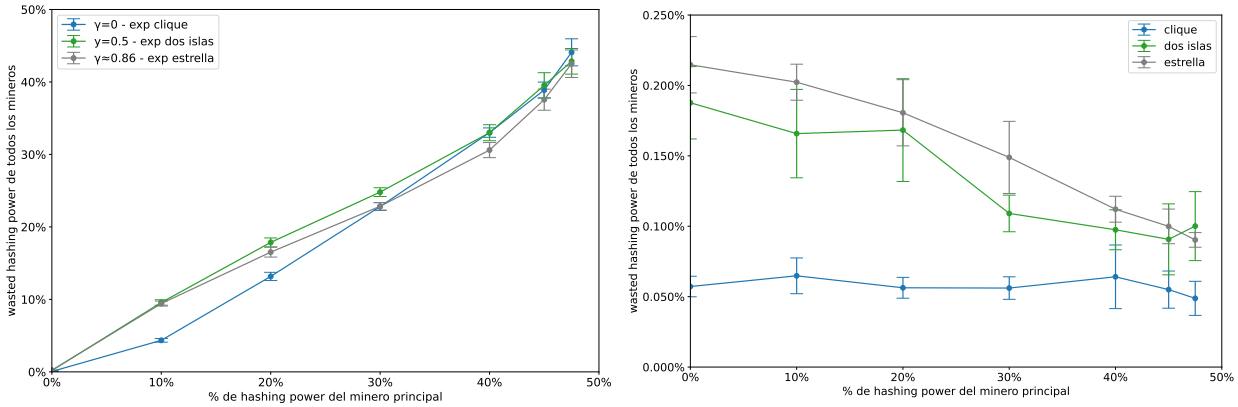
***Wasted hashing power* de los mineros honestos** En el caso que el minero principal sea *selfish*, el *wasted hashing power* de los mineros honestos (es decir todos menos el minero principal) tiene una evolución cercana a una exponencial a medida que aumenta el $\alpha$ del ataque, como se se puede observar esto en la figura 3.4. También se ve en ésta los muy altos valores de esta magnitud para los parámetros más exitosos del ataque, como son $\alpha \geqslant 0{,}4$. La evolución exponencial respecto de $\alpha$ es consistente con la evolución exponencial de la proporción de bloques minados por el atacante. Algo que llama la atención es el hecho que la diferencia de $\gamma$ entre la topología de dos islas y la estrella no se haya trasladado en una diferencia notoria de *wasted hashing power*, sino que solamente lo haga para la topología clique.

No se incluye acá la comparación con el caso que el minero principal sea honesto ya que todos los mineros utilizan la misma estrategia, se deja el resultado de esa magnitud para la siguiente sección.

***Wasted hashing power* de todos los mineros** Para el caso que el minero principal es *selfish*, al observar el *wasted hashing power* total en 3.5(a), se ve que este es constantemente creciente a medida que aumenta el *hashing power* del *selfish miner*, llegando hasta ser casi un 45 %. Por otro lado, un $\gamma$ más alto no necesariamente implica más *wasted hashing power*, lo cual adjudicamos a la integración de un *wasted hashing power* más bajo del *selfish* con el comportamiento no correlacional del de los honestos.

En cambio, para el caso que el minero principal es *honesto* (la figura 3.5(b)) se observan nuevamente valores muy bajos de *wasted hashing power* (todos menores a 0.25 %). Esto indica que los bajos valores previamente observados para el minero principal suceden también para todos los mineros en general.

Comparando los valores de *wasted hashing power* total de ambos casos nos da una noción de lo perjudicial que es el ataque para la calidad de la red: mientras que con un atacante este llega a superar valores del 40 %, sin este (y en el mismo contexto) tiene valores inferiores al 0.25 %.




**(a)** *Dado que el minero principal es* ***selfish****.*

**(b)** *Dado que el minero principal es* ***honesto****.*

**Figura 3.5:** *Wasted hashing power de todos los mineros, para topologías chicas (número de nodos entre 8 y 9) con diferente grado de conectividad. Los porcentajes indican cuanto de los hashes realizados por todos los mineros durante las simulaciones fue desperdiciado. Notar el fuerte cambio de escala entre ambos gráficos.*

**Discusiones generales** Mencionamos, al comparar el *wasted hashing power* del minero principal según su estrategia que el atacante, aún siendo exitoso, tendrá más *wasted hashing power* que si hubiese sido honesto (donde hubiese sido de menos de un 0.25 %). Esto indica que, con un tiempo entre bloques fijo, la cantidad de bloques de la *main chain* (y por lo tanto *rewards*) del minero *selfish* será menor que si se hubiese comportado de forma honesta. Tal como se expresa en Nayak et al. [NKMS16], la rentabilidad del ataque depende de un ajuste de dificultad: para ganar más *rewards* un minero tiene que realizar el ataque por el suficiente tiempo para un ajuste de dificultad reduzca el tiempo entre bloques y recién ahí pasar a ser rentable.

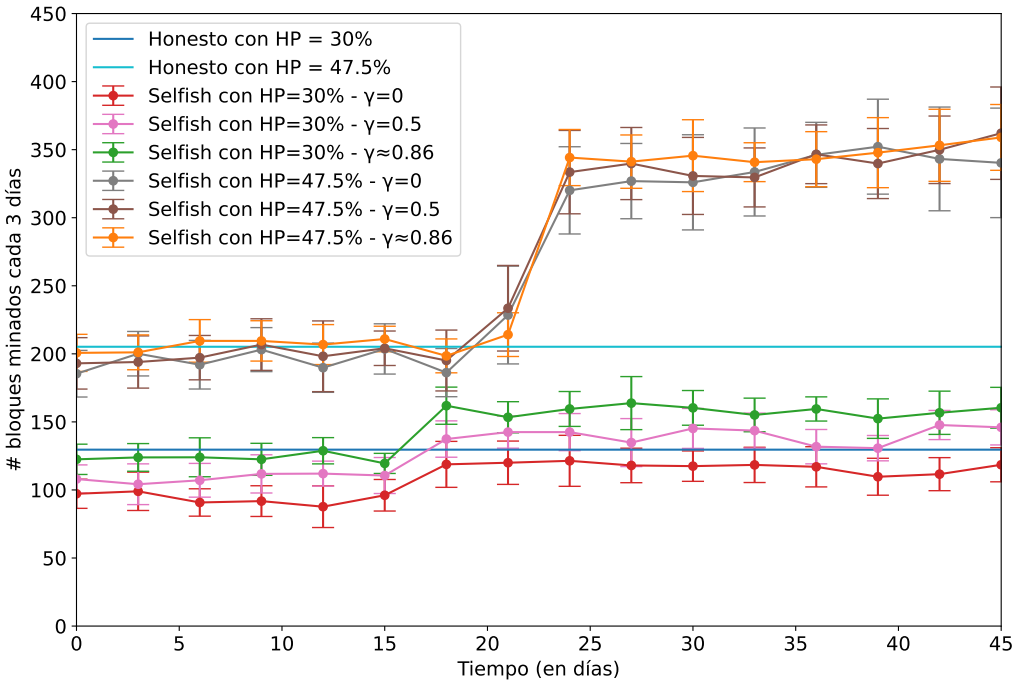



**Figura 3.6:** *Comparación del número de bloques minados (cada tres días simulados) por el minero principal siendo* selfish*, respecto del teóricamente esperado si hubiese sido honesto, para topologías chicas (número de nodos entre 8 y 9) con diferente grado de conectividad. Se muestran los resultados para las simulaciones con* hashing power *del* 30 % *y* 47.5 % *ya que son representativas de los diferentes casos que se pueden dar.*

Lo descrito se puede observar claramente en la figura 3.6 donde se muestra la cantidad de bloques minados por el atacante con variados parámetros, comparando con la cantidad que hubiese minado comportándose de forma honesta. Con $\alpha = 0{,}3$, el atacante minó menos bloques (y por lo tanto recibió menos *reward*) que si hubiese sido honesto hasta el día 18 en promedio (a excepción de $\gamma = 0$ donde el ataque fue siempre no exitoso). Lo mismo sucedió con $\alpha = 0{,}475$ (salvo con $\gamma \approx 0{,}86$), donde recién al día 25 comenzó a ganar más *reward* que comportándose de forma honesta. Obviamente, una vez que la rentabilidad cada tres días supera la rentabilidad siendo honesto, el atacante tendrá que recuperar lo "*perdido*" en los días anteriores antes de considerar el ataque "*rentable*". Estos datos dan una muestra del tiempo durante el cual se tiene que realizar el ataque antes de que éste sea exitoso.

Otro resultado interesante al comparar los *wasted hashing power* con el del minero principal (que es *selfish*) es remarcar el impacto que tiene la gran disparidad de información entre los mineros honestos y el atacante en la importante diferencia entre sus valores. El minero *selfish*, como mucho, tendrá un bloque por *fork* desperdiciado y las latencias normales de la red, mientras que el minero honesto recibirá tarde los bloques del *selfish* y con potencialmente *forks* muy largos desperdiciados. Esta diferencia de *wasted hashing power* explica el funcionamiento y éxito del ataque: el minero *selfish* no gana por minar más bloques que antes (de la *main chain*) sino por hacer que proporcionalmente los honestos malgasten más su *hashing power* y minen mucho menos (de la *main chain*).

### 3.1.2 Experimento variando la conectividad de la red

Habiendo ya analizado en el experimento anterior el comportamiento del ataque con distintos $\alpha$ y $\gamma$, el foco de este experimento será analizar como la conectividad de la red impacta en el éxito del *selfish mining*. Entonces, para las topologías usadas partiremos de la topología estrella e iremos agregándole ejes hasta obtener una topología clique. Debido a que el nodo *selfish* ya esta conectado con el resto de los nodos, estos nuevos ejes ocasionarán una mayor conectividad entre los mineros honestos, representando entonces un menor $\gamma$.

En la figura 3.7 se presentan las topologías utilizadas. El objetivo al agregar ejes fue la simetría:

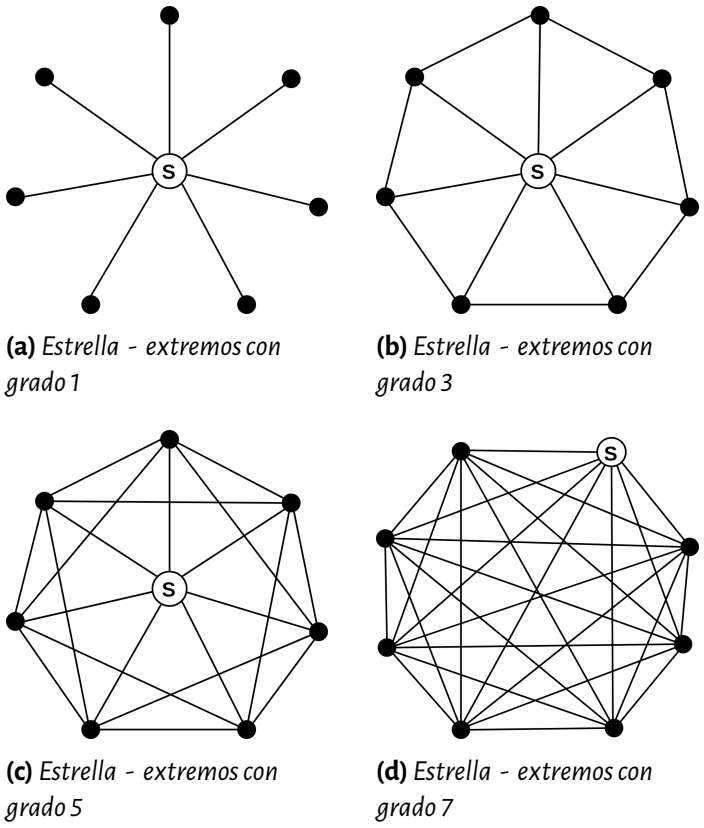
1. *Simetría entre los nodos honestos*: nos simplificará el análisis de los resultados que todos los mineros honestos estén en una situación de igualdad. En cada experimento, todos los mineros honestos tendrán el mismo grado $g$. Sean los mineros honestos $h \in 0, .., 6$, cada uno de ellos tendrá como *peers*:

$$peers(h) = \Big\{\text{minero selfish}\Big\} \cup \Big\{p : p \in \{0, .., 6\} \setminus \{h\} \wedge d(p, h) < \frac{g-1}{2}\Big\} \tag{3.1}$$

$$\text{con } d(p_1, p_2) = min(|p_1 - p_2|, 7 - |p_1 - p_2|). \tag{3.2}$$

   Esto significará que, dado el circulo que forman los nodos honestos alrededor del *selfish*, los *peers* serán los $(g-1)/2$ nodos más cercanos a su ïzquierdaz la misma cantidad a su "derecha".

2. *Simetría entre las topologías*: lo cual esperamos que distancie equitativamente los resultados obtenidos por ellos, es decir que el $\gamma$ aumente aproximadamente lo mismo entre una topología y otra. De una topología a otra, siempre aumentaremos en dos el grado de los nodos honestos, que ocasionará que aumente en siete la cantidad de ejes totales.




**(a)** *Estrella - extremos con grado 1*

**(b)** *Estrella - extremos con grado 3*

**(c)** *Estrella - extremos con grado 5*

**(d)** *Estrella - extremos con grado 7*

**Figura 3.7:** *Topologías a utilizar: con ocho nodos y variada conectividad. El nodo **S** será el único con estrategia* selfish, *siendo el resto nodos honestos.*

## Obtención de la traza a partir de eventos del simulador

Una herramienta que será utilizada para los análisis realizados para este experimento es la categorización de los bloques donde hubieron empates y quien fue el ganador de este. Para esto se partirá de los eventos del simulador (descritos en 2.1.3) y, sabiendo que nodo es el minero *selfish*, se reproducirá la traza sobre la máquina de estados que representa el ataque. Esto permitirá saber en que estado esta estaba para cualquier bloque, y específicamente encontrar todos los bloques donde hubieron empates.

## Resultados obtenidos

A continuación se presentarán los resultados obtenidos al variar el $\alpha$ y la topología usada entre las recientemente presentadas. Al igual que el experimento anterior, se realizó una comparación con el experimento de control en el que el minero principal sigue una estrategia honesta. Al obtenerse resultados similares y sin nuevas conclusiones a mencionar, no se los incluye en esta sección, pero se pueden ver en la sección A.1.1 del apéndice.

**Proporción minada por el minero *selfish*** En la figura 3.8 se presentan las proporciones de bloques minados por el *selfish miner*, comparándolas con el uso de la estrategia de minado honesta. Ésta aparenta tener distancias equitativas entre las distintas topologías, por lo que la *simetría entre topologías* planteada parece haber tenido el resultado esperado.

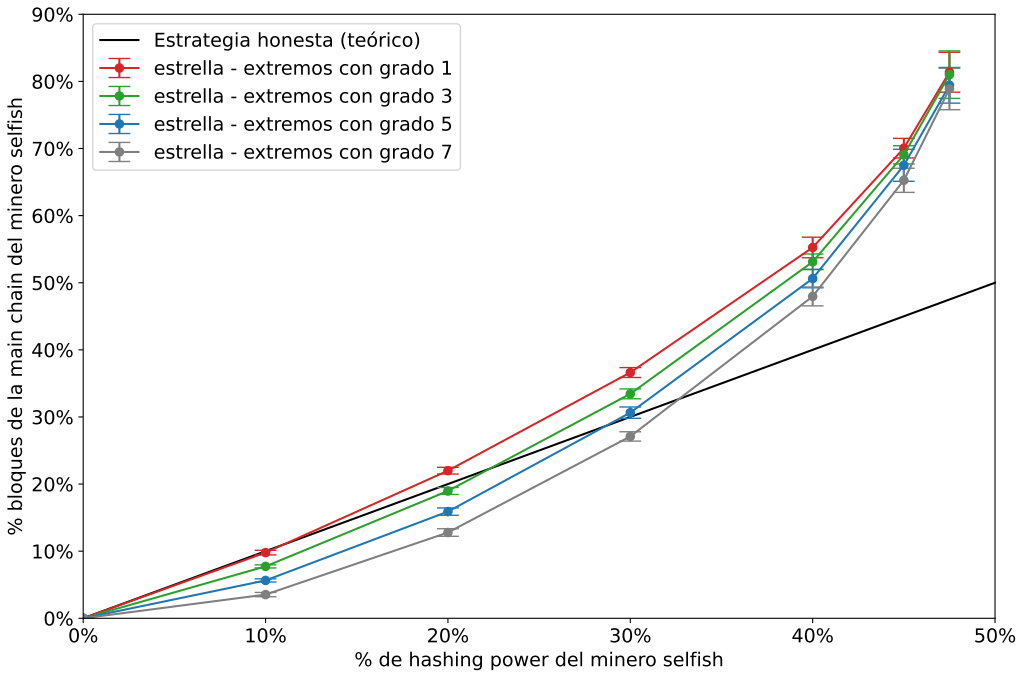



**Figura 3.8:** *Comparación de la proporción de bloques minados por el atacante para cada topología chica (de ocho nodos) de diferente grado de conectividad, y la teóricamente esperada si hubiese sido honesto.*

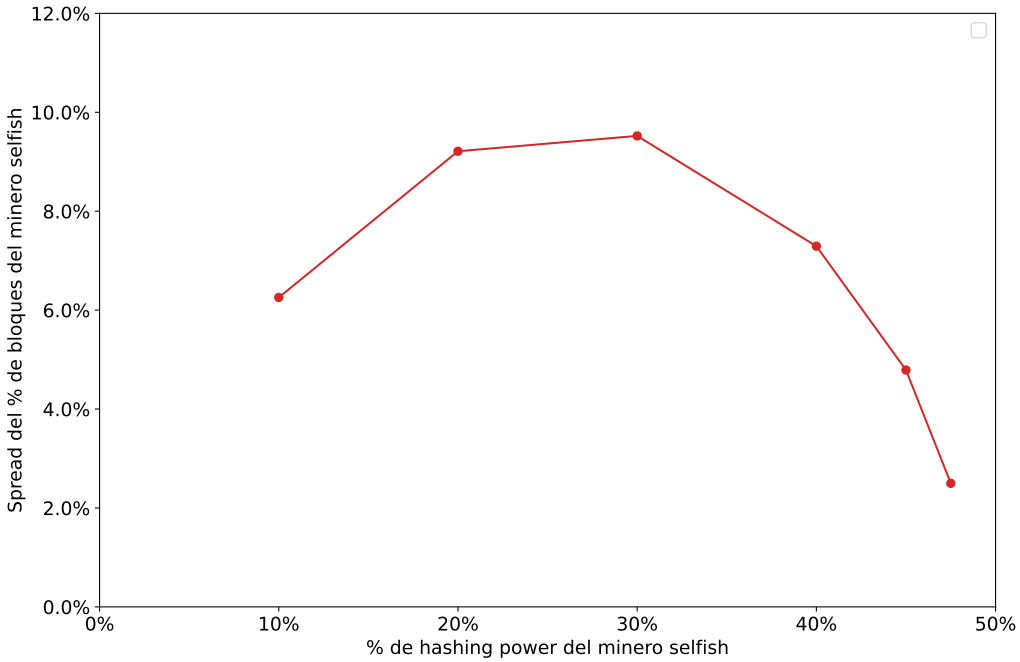



**Figura 3.9:** Spread *(diferencia entre el valor máximo y mínimo de cada* hashing power*) de la proporción minada por el minero* selfish *de la figura 3.8. Por los valores de esa figura, cada punto es la diferencia entre la proporción obtenida para la estrella con extremos de grado 1 y la obtenida para la de extremos de grado 7. Un valor mayor de* spread *indica entonces que el impacto del incremento de la conectividad del atacante (es decir $\gamma$) es mayor.*

En esa misma figura se puede observar cómo el impacto de la conectividad de la red varía bastante dependiendo del $\alpha$ utilizado. Para discutir esto, definimos el *spread* de la proporción de bloques minados (calculado para la figura 3.9) partiendo de los promedios de bloques minados, como la diferencia entre

el máximo valor observado y el mínimo. Para este caso en particular, será siempre entonces la diferencia en la proporción obtenida para un determinado $\alpha$ en la topología estrella y la proporción obtenida en la topología clique para el mismo $\alpha$.

Combinando lo observado en ambas figuras, por un lado podemos entonces ver que con $\alpha = 0{,}2$ y $\alpha = 0{,}3$ el *spread* es de casi un 10 % de bloques minados, siendo aquí la conectividad fundamental para determinar si el ataque fue exitoso (se minaron más que si hubiese sido honesto) o no. Por otro lado, con $\alpha = 0{,}475$, el *spread* es menor al 3 %, lo cual no solo es proporcionalmente mucho menor respecto de la proporción minada (alrededor de un 80 %), sino que tampoco impacta en el éxito del ataque: el ataque siempre fue exitoso con ese $\alpha$.

Entonces, surge analizar por qué el *spread* tiene el comportamiento observado, para lo cual empezaremos remarcando el hecho que éste es consecuencia del impacto que tiene el $\gamma$ sobre una simulación dado un determinado $\alpha$. Al ser el $\gamma$ un parámetro que impacta solo en el resultado de los empates, entonces un impacto mayor de éste requiere que hayan habido una mayor cantidad de empates. Dado un $\alpha$ y $\gamma$, cuánto mayor es el número de empates que se dan, mayor es el número de empates ganados por el *selfish* (el cual es proporcional al $\gamma$), y mayor es el número de bloques minados. En la figura 3.10, se observa la evolución del número de empates que se dieron para los distintos valores de $\alpha$. En primer lugar, como se esperaría, el número de empates es independiente del $\gamma$. En segundo lugar, lo observado gráficamente y el coeficiente Pearson de $0{,}9413$ parecerían confirmar nuestra hipótesis: la evolución del número de empates tiene una fuerte correlación con la evolución del *spread*, justificando esto la evolución del *spread* observado en la figura 3.9.

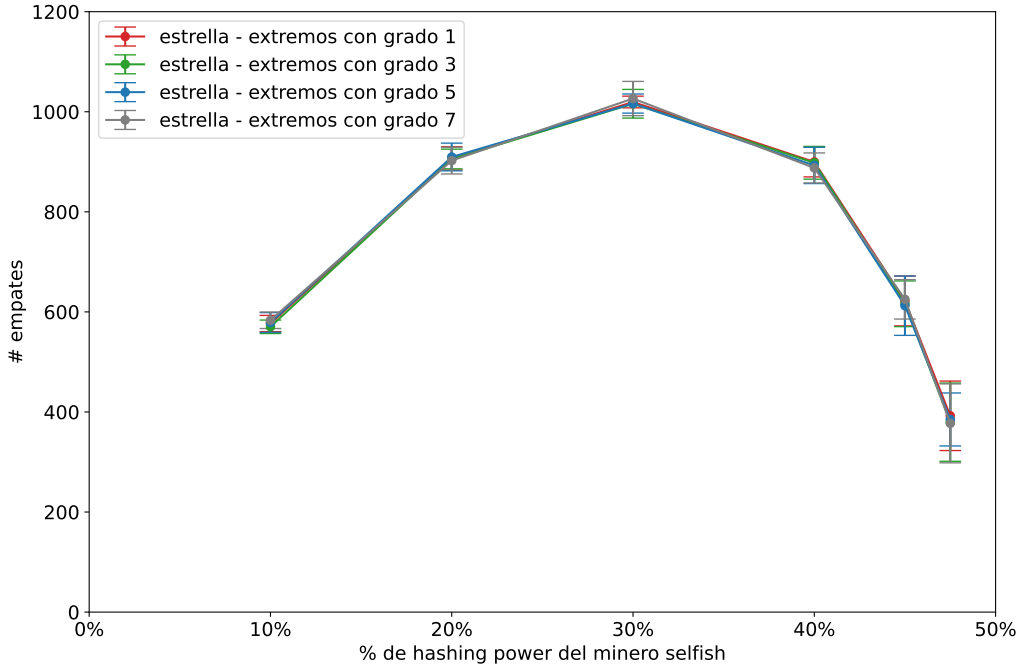
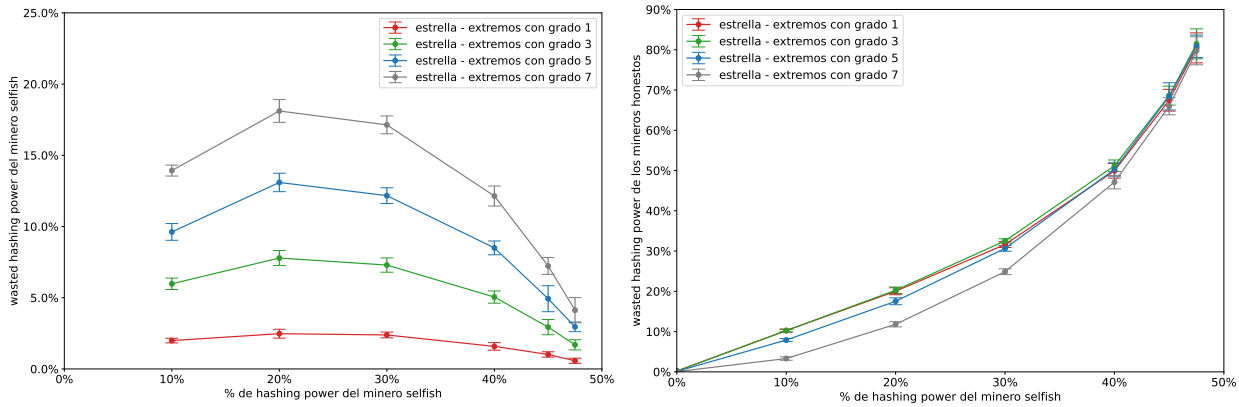



**Figura 3.10:** *Número de empates entre un* fork *producido por el minero* selfish *y otro producido por los honestos, para cada topología chica (de ocho nodos) de diferente grado de conectividad. Dos* forks *se consideran en empate si contienen el mismo número de bloques, y suceden si la máquina de estados del atacante (la figura 1.4) está en el estado 0'. Introducido debido a una potencial correlación con la figura 3.9.*

A continuación analizaremos el *wasted hashing power* de los mismos subconjuntos de nodos del experimento anterior: *selfish miner*, mineros honestos y el total.




**(a)** *Wasted hashing power del minero* ***selfish***.

**(b)** *Wasted hashing power de los mineros* ***honestos***.

**Figura 3.11:** Wasted hashing power *del minero atacante y del resto (los mineros honestos), para topologías chicas (de ocho nodos) con diferente grado de conectividad. Los porcentajes indican la proporción de los hashes desperdiciado por cada tipo de minero en los distintos experimentos.*

***Wasted hashing power* de los mineros *selfish*** En la figura 3.11(a) se observa el *wasted hashing power* del minero *selfish*, que sigue teniendo una evolución parecida al experimento anterior. Como ya se describió, el primer análisis sería esperar que un mayor $\alpha$ implique un ataque más exitoso y, entonces, un *wasted hashing power* constantemente menor. Sin embargo, esto no es lo que sucede. Un análisis más cercano del *wasted hashing power* explicará las razones por lo que esto no ocurre.

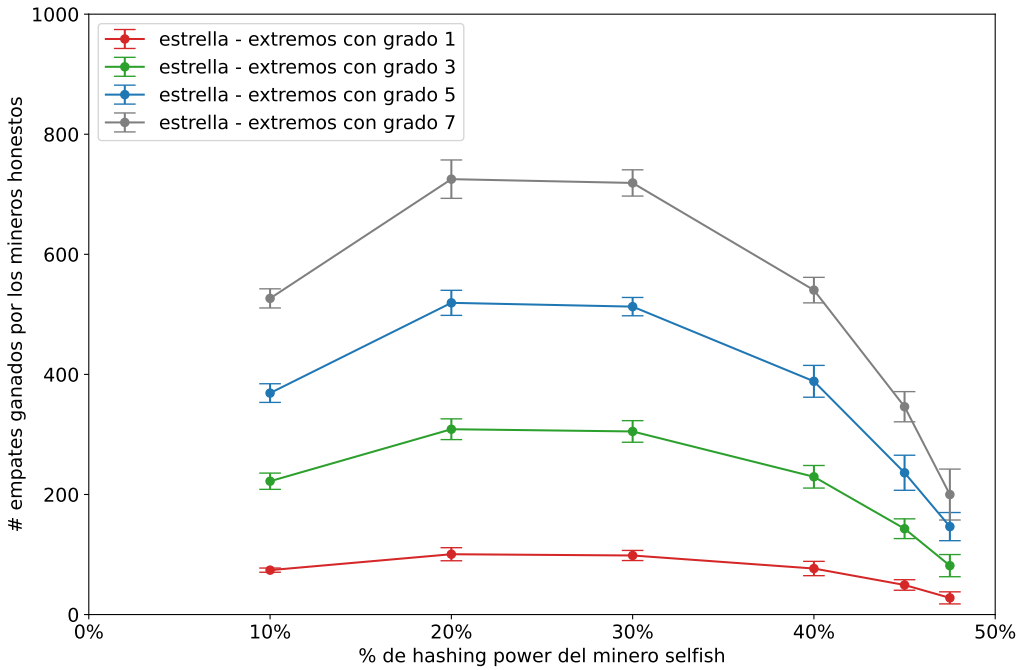



**Figura 3.12:** *Número de empates entre* forks *del minero* selfish *y los honestos que fueron ganados por estos últimos, para topologías chicas (de ocho nodos) con diferente grado de conectividad. Dos* forks *se consideran en empate si contienen el mismo número de bloques, uno de ellos gana cuando los bloques nuevos producidos lo continúan y no al otro. Introducido para explicar una potencial correlación con la figura 3.11(a).*

Al ser tener el minero *selfish* más información sobre los bloques minados que el resto (solo él conoce de su cadena privada), son limitadas las ocasiones donde éste desperdicia su *hashing power*. El único bloque

*stale* sobre el que mina es cuando hay un empate y lo gana un minero honesto, habiendo sido todos los bloques del *fork* aceptado producidos por los mineros honestos. Por lo tanto, una posible explicación de la evolución del *wasted hashing power* es que el número de empates ganados por mineros honestos evoluciona de la misma forma. Comparando la figura 3.12 con la 3.11(a), se puede observar que la explicación parecería ser correcta. Lo mismo indican los coeficientes de Pearson que analizan la correlación entre los valores obtenidos para cada topología:

- **Estrella - extremos con grado 1**: 0.9784
- **Estrella - extremos con grado 3**: 0.9857
- **Estrella - extremos con grado 5**: 0.9890
- **Estrella - extremos con grado 6**: 0.9894

El hecho que con $\alpha = 0{,}2$ o $\alpha = 0{,}3$ los mineros honestos ganen más empates podría resultar raro, si es que uno no considera también la evolución de la cantidad de empates que se dan (que se ve en la figura 3.10). Si en vez de observar el número de empates uno lo hace a través de la proporción de empates ganados, como en la figura 3.13, se obtiene una vista más intuitivamente esperada. La evolución de la proporción en relación al $\alpha$ parecería ser lineal, lo cual es consistente con el hecho de que la probabilidad que un minero honesto gane el empate es $(1 - \alpha)(1 - \gamma)$.

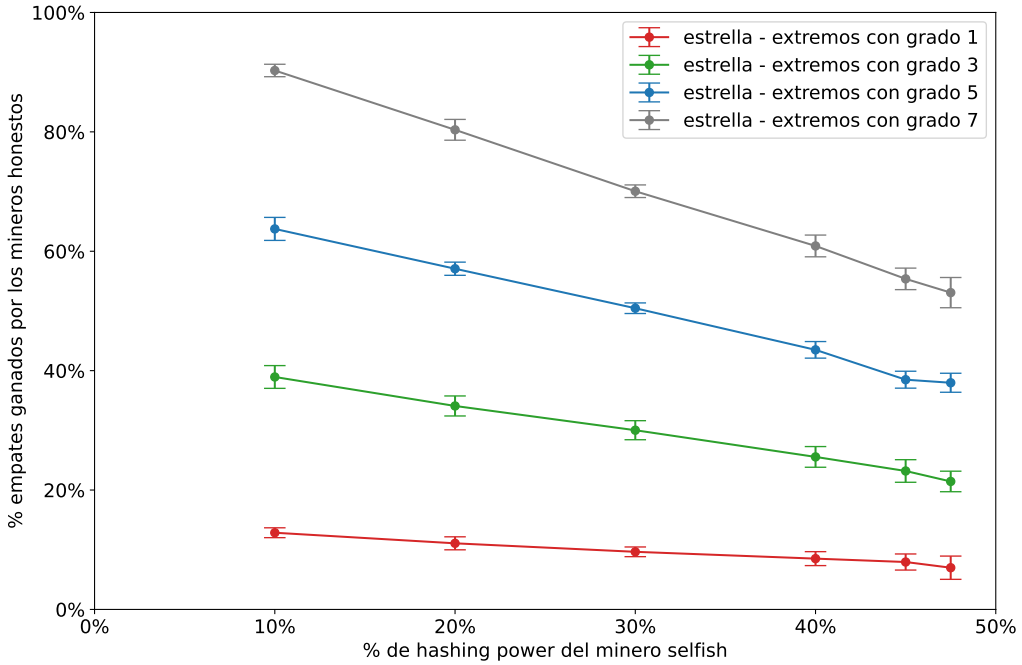



**Figura 3.13:** *Proporción de empates entre* forks *del minero* selfish *y los honestos que fueron ganados por estos últimos, para topologías chicas (de ocho nodos) con diferente grado de conectividad. Dos* forks *se consideran en empate si contienen el mismo número de bloques, uno de ellos gana cuando los bloques nuevos producidos lo continúan y no al otro. Teóricamente, se espera que sea* $(1 - \alpha)(1 - \gamma)$.

***Wasted hashing power* de los mineros honestos** En la figura 3.11(b) se observa un comportamiento parecido al del experimento anterior (figura 3.4), con los dos experimentos de mayor $\gamma$ pareciendo estar con resultados parecidos, sin más diferencias notables para mencionar.

***Wasted hashing power* de todos los mineros** En la figura 3.14 se presenta el *wasted hashing power* de todos los mineros. Éste continúa siendo constantemente creciente a medida que aumenta el $\alpha$, sin diferencias notables respecto del experimento anterior.

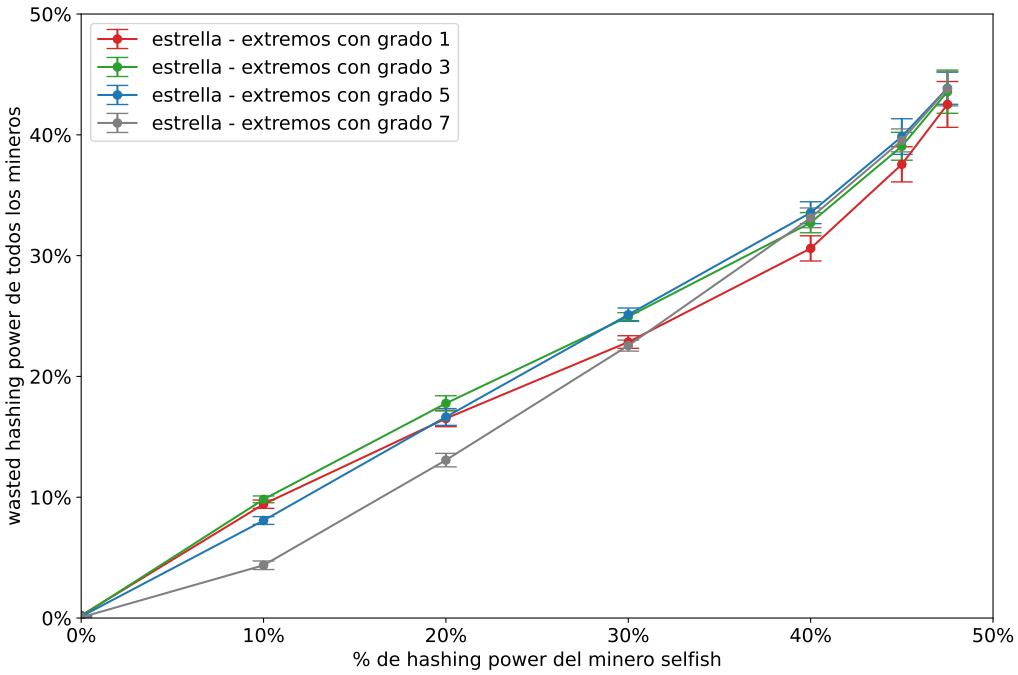



**Figura 3.14:** Wasted hashing power *de todos los mineros, para topologías chicas (de ocho nodos) con diferente grado de conectividad. Los porcentajes indican la proporción de los* hashes *desperdiciados por todos los mineros.*

## 3.2 Experimentos a gran escala

Habiendo analizado y comprobado que el *selfish mining* funciona como es esperado en topologías chicas y controladas, procederemos a verificar los resultados realizando experimentos lo más cercano posibles a la *mainnet* de Bitcoin.

### 3.2.1 Diseño experimental

En la tabla 3.2 se presenta un resumen general de las características de las simulaciones realizadas. En la siguientes secciones las describiremos en más detalle, centrándonos especialmente en las topologías usadas, que se basan fuertemente en las originalmente planteadas en el trabajo de tesis de Piotrkowski [Pio19].

| Configuración del experimento | |
|---|---|
| Topología física | Barabási de 10000 hosts |
| Topología lógica | Watts-Strogatz de 10000 nodos |
| Cantidad de mineros | 15 |
| Tiempo simulado | 600 horas |
| Repeticiones por topología | 3 |
| Topologías por configuración | 3 |

Tabla 3.2: Parámetros generales de los experimentos a gran escala

#### Topología lógica

**Número de nodos** Al momento de de la escritura de este manuscrito, el número de nodos de la *mainnet* de Bitcoin era de 15 000, pero el promedio de los últimos cinco años es 10 000 por lo que éste será el valor utilizado[11].

**Número de *peers* por nodo** Al analizar el trabajo de Miller et al. [MLP+15], se determinó que el grado de conectividad de los nodos en la *mainnet* de Bitcoin era en general ocho, por lo que ésta será la cantidad de *peers* que tendrán los nodos de nuestras simulaciones. Esto es consistente con la configuración por *default* de los clientes que usan ocho *peers* salientes[12].

**Tipo de topología** Esta característica también fue analizada por Miller et al. [MLP+15] en su trabajo, donde determinaron que la topología de Bitcoin cumple las características de ser:

- conexo
- compartir propiedades con las de un grafo aleatorio pero exhibiendo clusters o comunidades

Por lo tanto, la topología Watts-Strogatz (presentada en el trabajo [WS11]) será la utilizada, ya que comparte estas características. En base a este modelo es que se generará, aleatoriamente, la topología lógica usada para cada simulación.

[11] `https://bitnodes.io/dashboard/?days=1825`
[12] `https://bitcoin.org/en/full-node#reduce-traffic`

**Número de mineros** En la red de Bitcoin, debido al gran poder de cómputo requerido para generar bloques, los principales productores de bloques son los denominados *pools de minería*. En estos, mineros de distintos tamaños se agrupan para minar sobre el mismo bloque, repartiéndose entre ellos el *reward* por los bloques producidos. Al generar estos la mayoría de los bloques e interactuar con la red a través de un único nodo, serán los únicos que contabilizaremos para la determinación del número de mineros a utilizar, representando al resto de los mineros chicos con un único minero. Por este motivo utilizaremos 15 mineros[13].

**Dificultad y hashing power global** Tanto la dificultad inicial como el *hashing power* global de la red serán configurados para que se mine un bloque cada 10 minutos como se utiliza en Bitcoin. $\alpha$ del total del *hashing power* estará en manos del *selfish miner*, con el resto distribuido equitativamente entre los mineros honestos.

***Selfish mining*** El atacante será elegido siempre al azar.

## Topología física

**Número de hosts** Tendremos un *host* por nodo así que serán 10 000.

**Tipo de topología** En Bitcoin, la topología física es directamente Internet, que generalmente es modelada mediante una topología Barabási-Albert [AB02], por lo que será también la utilizada por nosotros. En base a este modelo es que se generará aleatoriamente la topología física usada para cada simulación.

**Ruteo entre hosts** Para que un paquete viaje de un host a otro, se usará el camino mínimo, determinado mediante el algoritmo de Dijkstra.

## Duración de las simulaciones y repeticiones realizadas

Por cada configuración se generarán aleatoriamente tres topologías distintas, repitiendo la simulación para cada una tres veces, con un total de nueve repeticiones por configuración. Cada simulación abarcará 600 horas, ocasionando *main chains* de más de 2000 bloques. Estos valores se han elegido debido a su baja varianza en pruebas empíricas.

## Cantidad de transacciones

Nuestras simulaciones se centran en el comportamiento del consenso y los bloques, no en las transacciones de la red, por lo que para simplificar los bloques solo tendrán las transacciones de *coinbase*.

## Ejemplificación de las características del modelado de la red

Para dar una mejor intuición del diseño experimental utilizado, analizaremos a continuación ejemplos de topologías generadas bajo los modelos descritos previamente. Se eligieron cuatro topologías de los experimentos realizados (por su representatividad del resto), para cada uno de los cuales se explorarán algunas de sus características que consideramos más relevantes.

**Topología lógica** Comenzaremos por la topología lógica, que previamente describimos que será modelada con el modelo de Watts-Strogatz y con un grado de conectividad objetivo de ocho nodos.

En la figura 3.15 se puede ver que, para el modelo utilizado, los nodos de la topología lógica tienen en su gran mayoría ocho vecinos. La distribución respecto del resto de los grados tiene un comportamiento similar a lo que se esperaría de una distribución normal alrededor de ocho.

[13] `https://explorer.btc.com/btc/insights-pools`

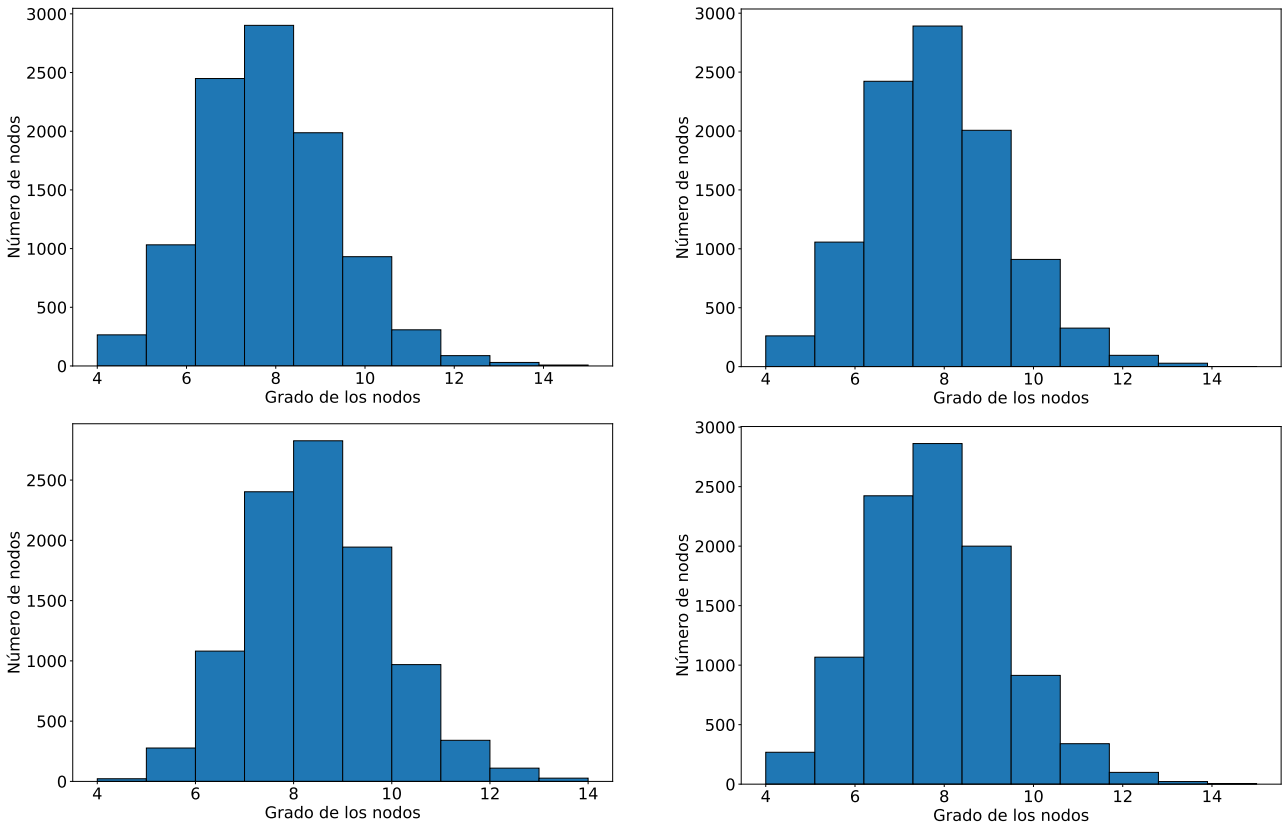
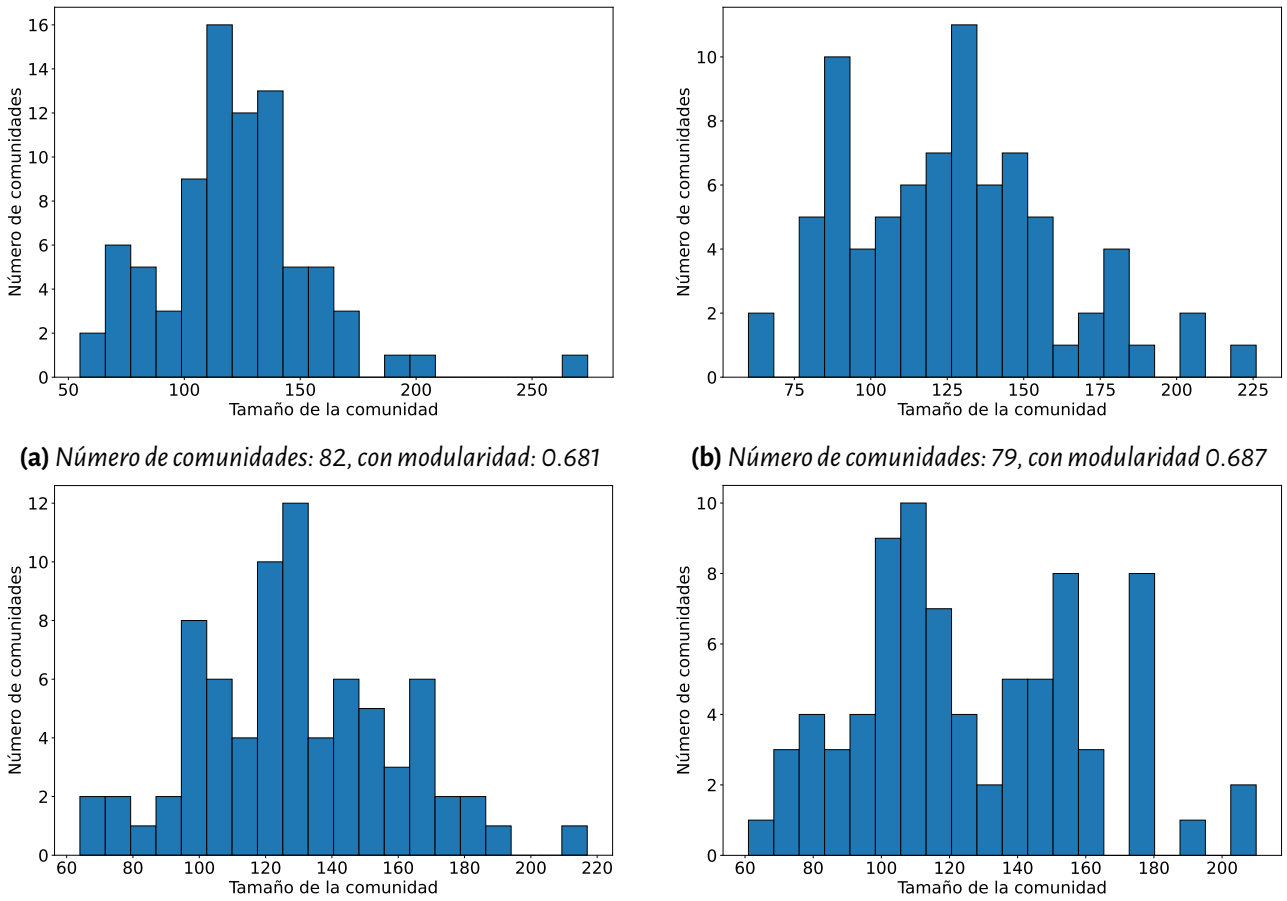



**Figura 3.15:** *Histogramas de las topologías lógicas mostrando la relación entre los grados de los nodos observados dentro del grafo y la cantidad de nodos con ese grado. Al crearlas utilizando el modelo de Watts-Strogatz se determinó que su valor objetivo era ocho.*

Por otro lado, en la figura 3.16 se muestra un análisis de las comunidades de los mismos ejemplos. Para realizarlo, se utilizó el algoritmo de detección de comunidades propuesto por Blondel et al. [BGLL08], que es una heurística basada en la optimización de la medida de modularidad. La modularidad es indicadora de la calidad de las comunidades y fue presentada por primera vez por Newman y Girvan [NG04].

Al presentar esa medida, Newman y Grivan indican que, en la práctica, los valores observados de modularidad suelen estar en el rango entre 0.3 a 0.7. Luego, el hecho de haber obtenido en cada uno de los ejemplos comunidades con modularidad mayor a $0{,}68$ indica que las comunidades observadas son de muy buena calidad.

En este análisis se observa que cada topología tiene alrededor de 80 comunidades, teniendo la mayoría de estas entre 100 y 200 nodos cada una. Esto parecería ser consistente con las características deseadas que fueron planteadas al haber decidido seleccionar Watts-Strogatz como modelo para las topologías lógicas.




**(a)** *Número de comunidades: 82, con modularidad: 0.681*

**(b)** *Número de comunidades: 79, con modularidad 0.687*

**(c)** *Número de comunidades: 77, con modularidad 0.685*

**(d)** *Número de comunidades=79, con modularidad: 0.685*

**Figura 3.16:** *Análisis de las comunidades de la topología lógica: en cada gráfico se muestra el número de comunidades que tiene cada tamaño, junto con la modularidad de ellas, que sirve como medida de su calidad. Las topologías lógicas analizadas fueron generados siguiendo el modelo de Watts-Strogatz, que exhibe clusters o comunidades.*

**Topología física** Para el modelado de la topología física ya habíamos mencionado que utilizaremos el modelo de Barabási-Albert, que es uno de los más utilizados para modelar la red de Internet. Este modelo genera redes *scale-free* aleatorias, siguiendo entonces el grado de los nodos una distribución *power law*.

En la figura 3.17 se ejemplifica cómo el uso del modelo Barabási-Albert ocasiona que las redes físicas generadas sigan una distribución *power law* del grado de sus nodos. Esto ocasiona que la cantidad de nodos baje muy rápidamente a medida que aumenta su grado: la gran mayoría de los nodos poseen menos de 25 vecinos, mientras que hay menos de diez nodos en los intervalos con grado mayor a 200.

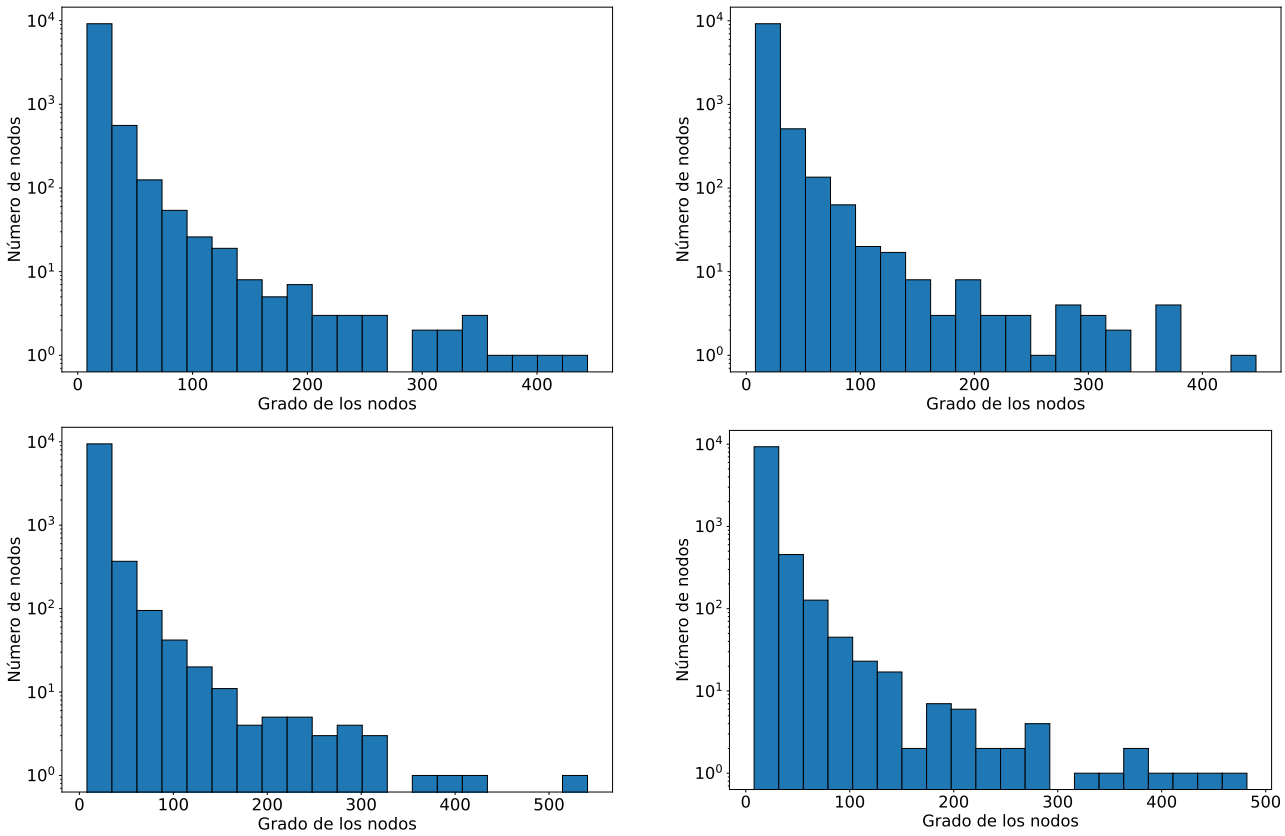



**Figura 3.17:** *Histogramas de las topologías físicas mostrando la relación entre los grados de los nodos observados dentro del grafo y la cantidad de nodos con ese grado, utilizando una escala logarítmica para esta última (el eje Y). Las topologías físicas analizadas fueron generadas utilizando el modelo de Barabási-Albert, el cual sigue una distribución* power law.

**Topología combinada** Habiendo explorado las características del modelo de topología física y lógica utilizadas, nos centraremos en la topología obtenida como combinación de ambas. En particular, mostraremos los valores de centralidad observados para cada nodo, que exhibirán la conectividad que cada uno de estos poseen con el resto de la red.

En la sección 2.2.3 se describió *load centrality*, además de ésta utilizaremos otras dos que miden, de distinta forma, la importancia de un nodo en una red:

1. *degree centrality*: la importancia de un nodo es el número de sus vecinos,

2. *eigenvector centrality*: la importancia de un nodo es dependiente de la importancia de sus vecinos,

3. *load centrality*: la importancia de un nodo depende de la cantidad de caminos mínimos del grafo que pasan a través de él.

Cada una de estas centralidades posee valores en el rango de 0 a 1.

En las figuras 3.18 (de *degree centrality*), 3.19 (de *eigenvector centrality*) y 3.20 (de *load centrality*) se muestran los histogramas de los valores obtenidos para cada una de las tres centralidades descritas. Existen algunas variaciones entre los valores y formas observadas en estas, pero el común denominador entre todas es una centralidad muy baja (menor a $0{,}05$) de todos los nodos, y por lo tanto una baja conectividad de cada uno de ellos.

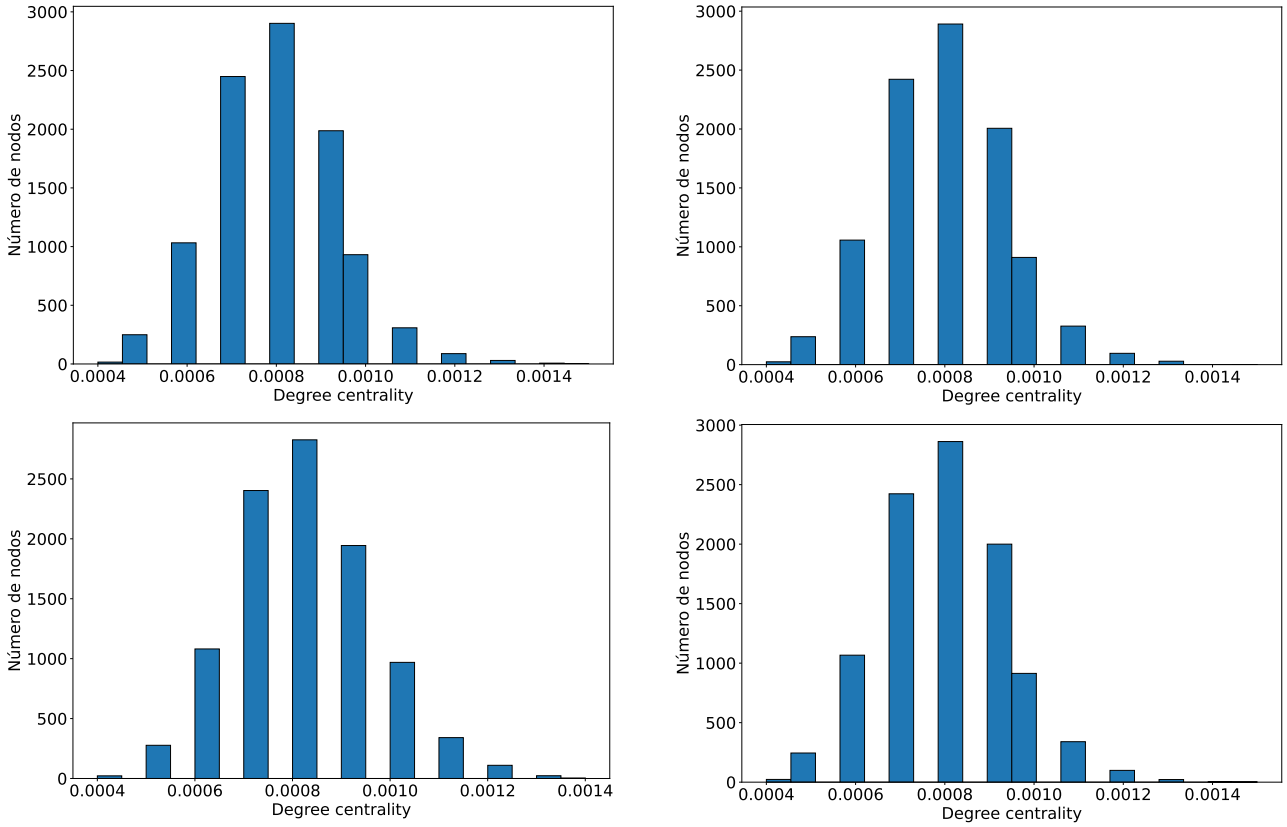



**Figura 3.18:** *Histogramas con el número de nodos de la topología combinada que poseen cierto* degree centrality.

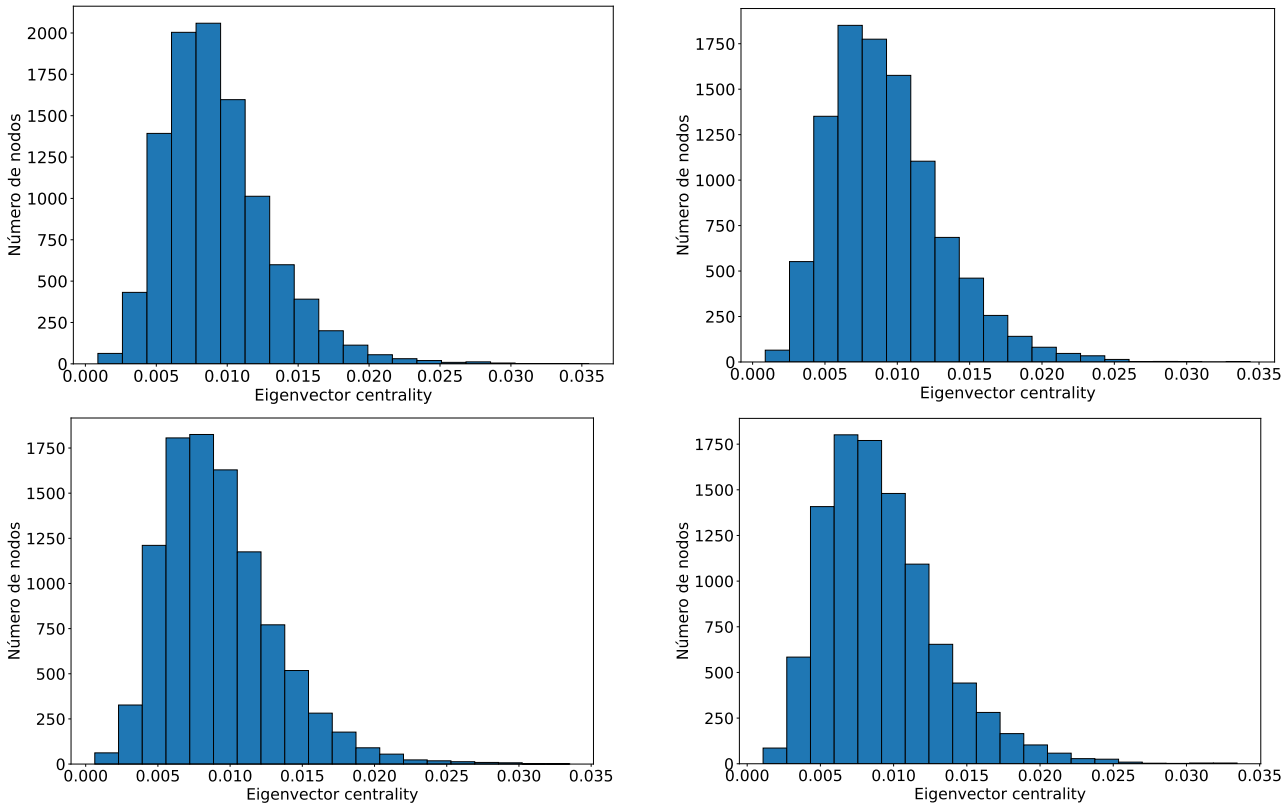



**Figura 3.19:** *Histogramas con el número de nodos de la topología combinada que poseen cierto* eigenvector centrality.

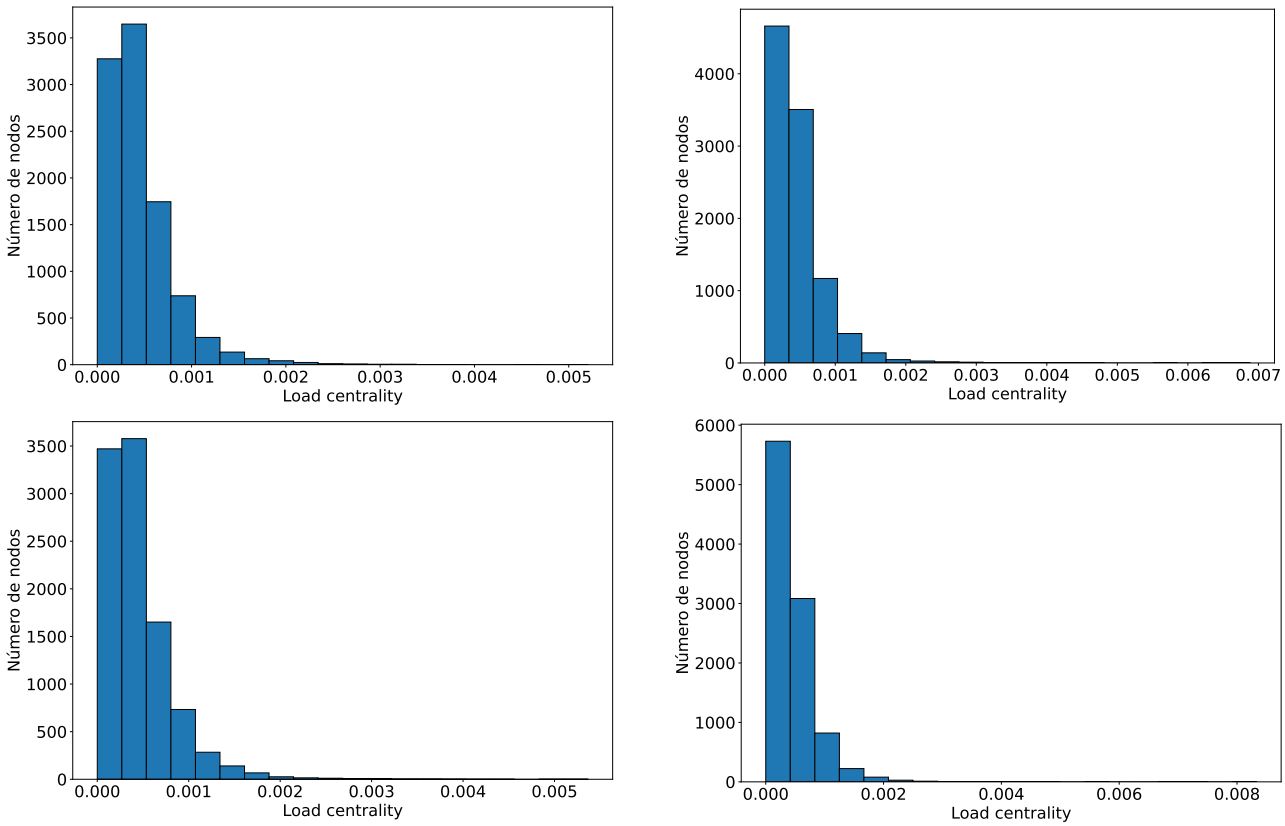


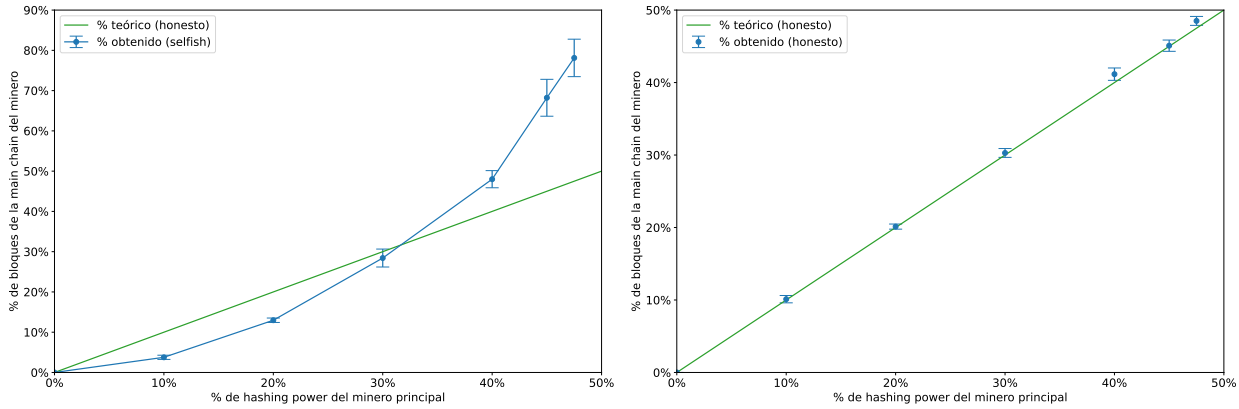

**Figura 3.20:** *Histogramas con el número de nodos de la topología combinada que poseen cierto* load centrality.

### 3.2.2 Experimento variando el *hashing power* del minero *selfish* ($\alpha$)

El primer experimento a gran escala a realizar será utilizar la topología descrita en la sección anterior y evaluarla variando el $\alpha$ del *selfish miner* entre los mismos valores de los anteriores experimentos: 0.1, 0.2, 0.3, 0.4, 0.45 y 0.475.

#### Proporción minada por el minero principal

La figura 3.21(a) presenta la comparación entre la proporción de bloques minados siguiendo la estrategia *selfish* y lo teóricamente esperado si usara la estrategia honesta. Las tendencia observada para la estrategia *selfish* corresponde a la vista en los experimentos anteriores, siendo el ataque exitoso con $\alpha \geqslant 0{,}4$. De esto se puede concluir que el ataque de *selfish mining* sigue siendo igual de exitoso en topologías de gran escala.




**(a)** *Siguiendo la estrategia* ***selfish****.*

**(b)** *Siguiendo la estrategia* ***honesta****.*

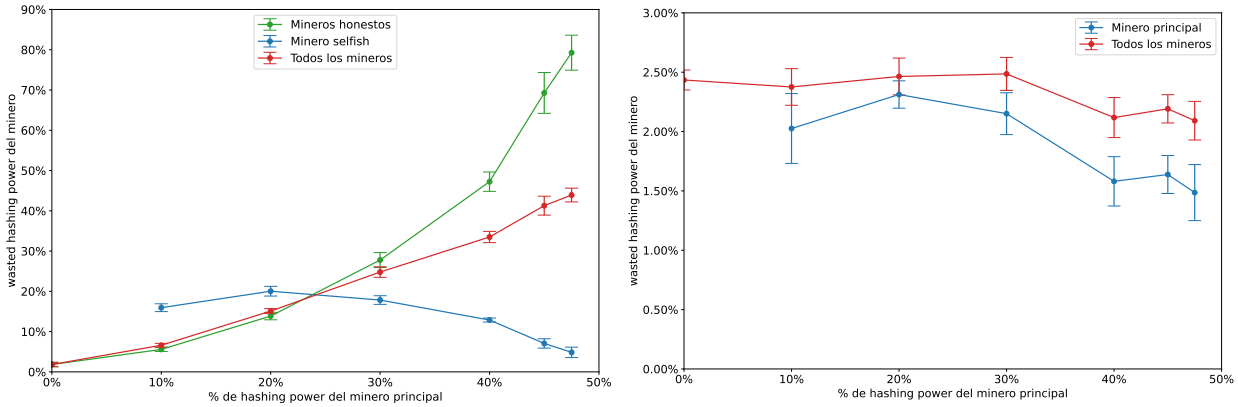
**Figura 3.21:** *Comparación de la proporción de bloques minados por el minero principal para distintos* hashing power *en topologías a gran escala (de* 10 000 *nodos), respecto de lo teóricamente esperado para un minero honesto.*

Por otro lado, en el caso que el minero principal sigue una estrategia honesta se realiza un control de que la proporción teórica previamente mencionada sucede también en la práctica. Los resultados de esto se observan en la figura 3.21(b), donde se ve que la proporción de bloques minados sí respeta lo teóricamente esperado.

#### *Wasted hashing power*

Empezando por el caso en cual el minero principal es *selfish*, se presenta en la figura 3.22(a) la comparación del *wasted hashing power* del minero *selfish*, honesto y el total. Las tendencias observadas para estos son consistentes con las vistas en los experimentos anteriormente realizados.

En contraste, la figura 3.22(b) muestra el valor de esta magnitud para el caso que el minero principal sea honesto. Lo que primero se observa es que los valores de *wasted hashing power* obtenidos (de 1.5 % a 2.5 %) son de una escala alrededor de diez veces más grande que en los experimentos con topologías chicas (donde eran menores a 0.25 %). Esta diferencia se puede explicar por el hecho que las topologías chicas fueron diseñadas con una mayor conectividad, que provoca menores latencias y, entonces, se tienen un menor tiempo de los mineros minando en bloques viejos, es decir, menor *wasted hashing power*. Otra observación interesante es que el *wasted hashing power* del minero principal siempre es menor que el de la totalidad de los mineros. Se requiere la realización de nuevos experimentos para analizar esta diferencia y constantar si es que existe alguna causa para ella o si se trata de un artefacto estadístico.




**(a)** *Dado que el minero principal es* ***selfish****.*

**(b)** *Dado que el minero principal es* ***honesto****.*

**Figura 3.22:** Wasted hashing power *de cada tipo de nodo respecto del* hashing power *del minero principal, para topologías a gran escala (con* 10 000 *nodos). Los porcentajes indican cuántos de los* hashes *realizados durante las simulaciones por los mineros de cada tipo fue desperdiciado.*

Por otro lado, comparando los *wasted hashing powers* de cada estrategia del minero principal se sigue observando una muy amplia diferencia, provocada por lo detrimental que es para la red el ataque de *selfish mining*.

### *Load centrality*

En el gráfico de proporción de bloques minados no se realizó la comparación con los valores esperados como en los experimentos anteriores, ya que desconocemos el $\gamma$ de las topologías utilizadas. No es de nuestro conocimiento que no se hayan hecho hasta ahora análisis sobre el cálculo del $\gamma$ para una topología arbitraria, por lo que utilizaremos una medida suficientemente parecida para su estimar su valor. La medida utilizada será la *load centrality*, que ha sido descrita en detalle en la sección 2.2.3.

El motivo de esto es que, en pocas palabras, la *load centrality* es la cantidad de caminos mínimos entre cualquier par de nodos que pasan por un nodo en particular. Aplicando esto al caso de un empate en el *selfish mining*, se lo podrían considerar al par de nodos como los mineros honestos, el emisario siendo el que minó el bloque que empató y el receptor siendo otro minero, y al nodo en particular siendo el nodo *selfish*. La cantidad de caminos mínimos lo podemos entonces asociar con la probabilidad de que un bloque minado por un honesto sea recibido primero por el minero *selfish* (pudiendo entonces propagar su cadena privada) respecto de que no lo sea, es decir con $\gamma$. Como se observa, ambas medidas parecerían no ser exactamente equivalentes, pero con una misma idea detrás en común.

Se calculó el promedio de *load centrality* del *selfish miner* para las simulaciones con cada valor de $\alpha$. Al ser esta una medida dependiente exclusivamente de la topología usada, es entonces independiente del $\alpha$ utilizado, lo cual explica porque se obtuvieron valores muy parecidos para los distintos valores de $\alpha$. Siempre se obtuvo un valor muy chico (el valor máximo fue $0{,}000556$), lo cual es consistente con la baja centralidad observada para todos los nodos de las topologías ejemplo de la sección 3.2.1. Los resultados de esa sección muestran que aún utilizando el nodo de mayor centralidad de cada topología, su centralidad seguiría siendo muy baja.

Por otro lado, en la figura 3.23 se compara la proporción de bloques minados por el *selfish* con la teóricamente esperada si utilizamos a la *load centrality* como estimador del $\gamma$. El no haber amplias diferencias entre ambas indicaría que la *load centrality* es un buen estimador para el $\gamma$, al menos cuando hay una baja conectividad. Partiendo de la observación anterior, se concluye entonces que $\gamma \approx 0$ para las simulaciones realizadas en este experimento.

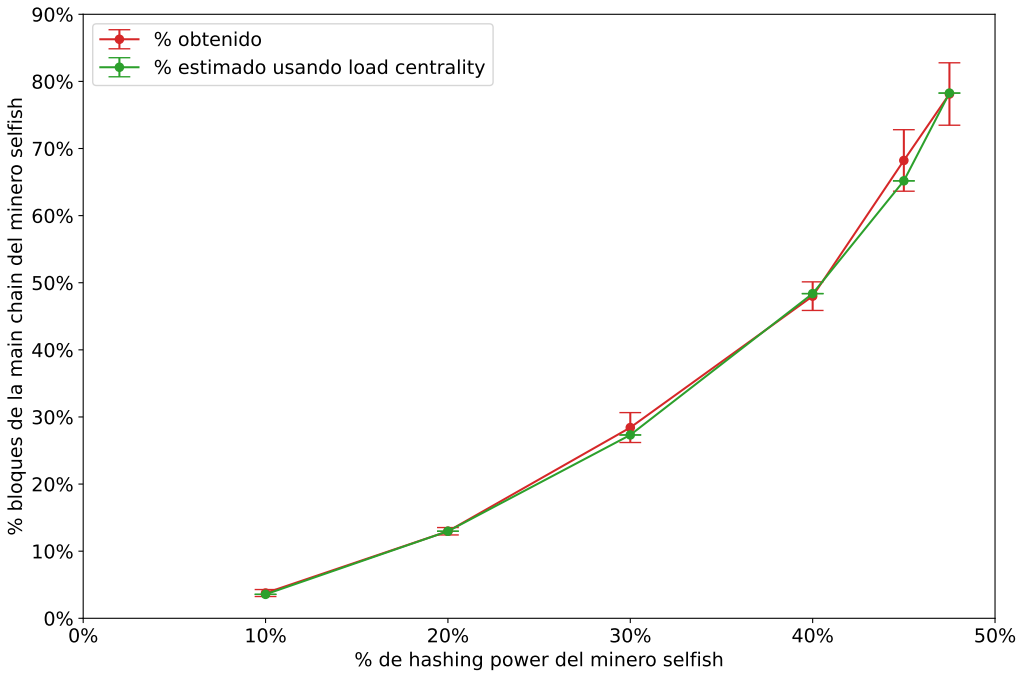



**Figura 3.23:** *Para analizar que tan buena estimación de $\gamma$ es el load centrality se lo utiliza para estimar la proporción de bloques minados del* selfish *reemplazando $\gamma$ por el* load centrality *en la fórmula de proporción esperada que depende tanto de $\alpha$ como $\gamma$ (observada en la ecuación 1.1). Se compara la estimación realizada con la proporción de bloques minados observada dependiendo del* hashing power *del minero principal, realizando todo para topologías a gran escala (de* 10 000 *nodos).*

### 3.2.3 Experimento variando la conectividad

En el experimento anterior concluimos que la topología planteada resultaba en $\gamma \approx 0$, de acuerdo a nuestras estimaciones mediante el *load centrality*. En este experimento entonces exploraremos distintas topologías que incrementen el valor de $\gamma$, modificando para ello la conectividad que el minero *selfish* tiene con el resto de la red.

Con este objetivo agregaremos paulatinamente a la topología del experimento anterior (la planteada en la sección 3.2.1) más conexiones del minero *selfish* al resto de los nodos. Cada simulación será entonces configurada con un número de *peers* extras, que será la cantidad de conexiones que agregaremos del minero *selfish* al resto de los nodos, eligiendo estos de forma aleatoria. Se realizará esto hasta llegar a que el minero *selfish* esté conectado a toda la red, agregando entonces como número de *peers* extras: 0, 250, 500, 1000, 2000, 4000, 8000 y 9991.

A continuación mostramos los resultados obtenidos, en los cuales mantendremos $\alpha = 0{,}4$. Como en anteriores experimentos, se realizó un experimento de control donde se reemplaza al minero principal *selfish* por uno honesto. Los resultados se encuentran en la sección A.1.2, no siendo añadidos aquí por no aportar nuevas conclusiones no discutidas ya en la comparación del anterior experimento.

#### Proporción minada por el *selfish*

En la figura 3.24 observamos la evolución de la proporción de bloques minados por el minero *selfish*. El ataque tiene una tendencia a ser cada vez más efectivo hasta llegar a los 2000 *peers* extras, valor a partir del cual las proporciones obtenidas pasan a ser bastante constantes. Dado que existen solo 15 mineros, y estando estos conectados aleatoriamente a solo ocho nodos, no resulta sorprendente que la diferencia en conectividad sea suficientemente grande respecto del atacante con 2000 *peers* extras, resultando en que

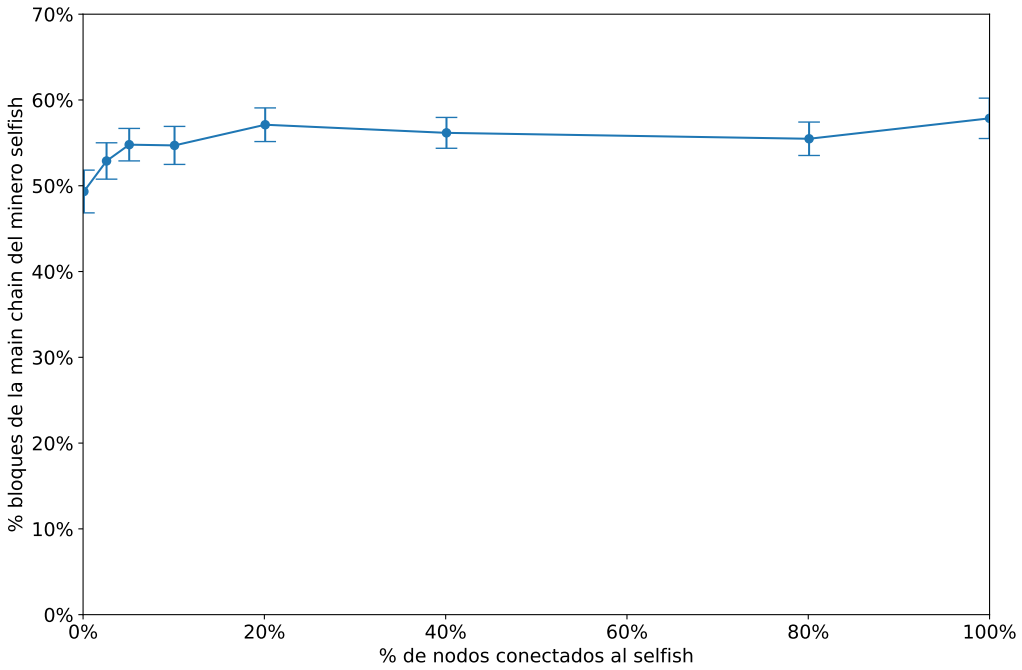



**Figura 3.24:** *Proporción de bloques minados por el minero principal para distintos números de* peers *extras en topologías a gran escala (de* 10 000 *nodos) y con* hashing power *constante del minero principal de* 40 %.

agregando más de estos no modifiquen el éxito del ataque.

### *Wasted hashing power*

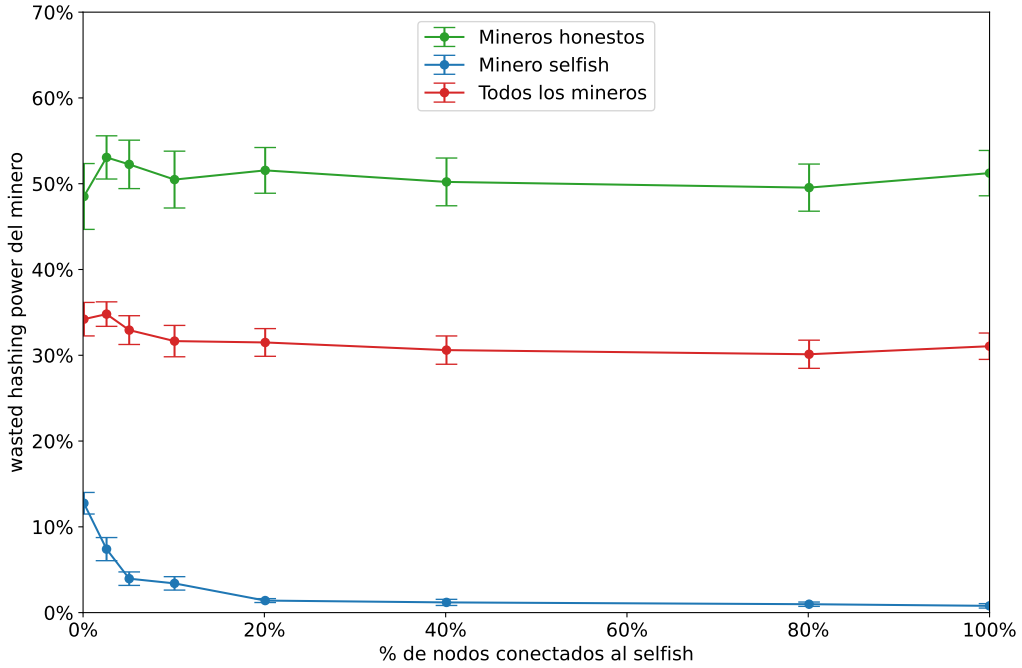
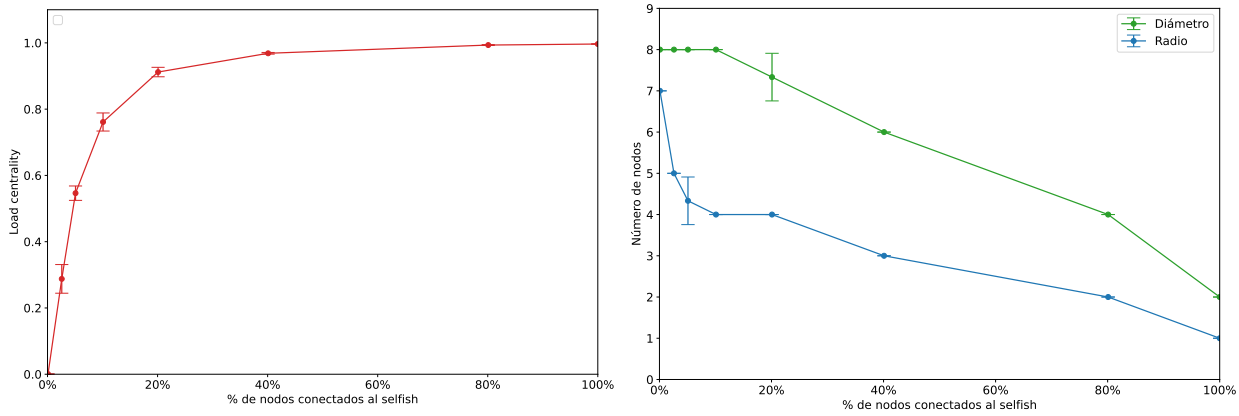



**Figura 3.25:** Wasted hashing power *de cada tipo de nodo respecto del número de* peers *extras, para topologías a gran escala (con* 10 000 *nodos) y con* hashing power *constante del minero principal de* 40 %. *Los porcentajes indican la proporción de los* hashes *realizados por los mineros de cada tipo que fue desperdiciado.*

Se presenta en la figura 3.25 el *wasted hashing power* del minero *selfish*, los mineros honestos y el total. Los resultados obtenidos son consistentes con el gráfico anterior, ya que se mantienen generalmente constante luego de superar los 2000 *peers*. Al igual que habíamos observado en los experimentos menor escala donde también se variaba la conectividad del atacante (en la sección 3.1.2), una mayor conectividad resultó en un *wasted hashing power* del minero *selfish* constantemente decreciente, mientras que en el caso de lo mineros honestos se puede apreciar un poco más de fluctuación.

### *Load centrality*




**(a)** *Load centrality*

**(b)** *Diámetro y radio*

**Figura 3.26:** *Propiedades de las topologías a gran escala (de* 10 000 *nodos), respecto del número de* peers *extras.*

Como se observa en la figura 3.26(a), el *load centrality* evolucionó de forma logarítmica respecto de la cantidad de *peers* extras. Su incremento resulta considerable hasta llegar a los 2000 *peers* extras, donde se supera el valor de $0{,}91$, y llegando casi $1$ (el máximo valor posible) cuando el *selfish miner* logró una conectividad total a la red.

Tal como se presenta en la figura 3.27, utilizando los valores del *load centrality* como estimadores del $\gamma$ para el cálculo de proporción de bloques teóricamente esperados se sigue observando una gran cercania entre lo estimado y lo obtenido. La diferencia entre ambos nunca superó el 3 %, lo cual sigue confirmando que el *load centrality* es un buen estimador del $\gamma$.

Si analizamos entonces la evolución del $\gamma$ dada la evolución de su estimador se ve que efectivamente logramos incrementar el valor de $\gamma$ aumentando la cantidad de *peers* extras que tiene el *selfish miner*. La evolución de esta parecería ser logarítmica, partiendo de $\gamma \approx 0$ sin extra *peers* y llegando a $\gamma \approx 1$ con una conectividad total a la red.

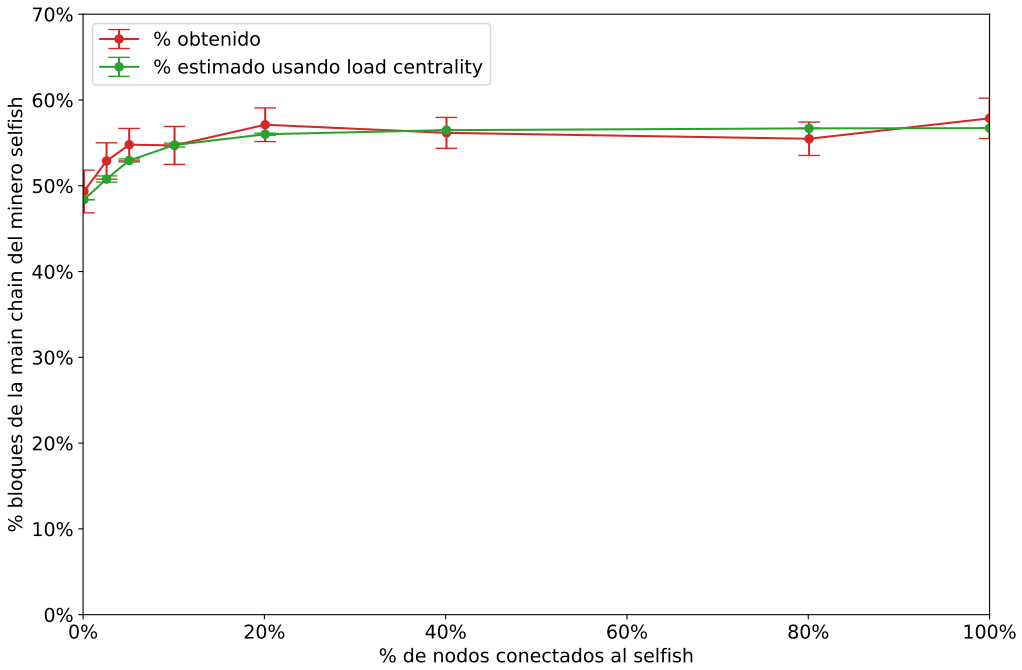



**Figura 3.27:** *Para analizar que tan buena estimación de $\gamma$ es el* load centrality *se lo utiliza para estimar la proporción de bloques minados del* selfish *reemplazando $\gamma$ por el* load centrality *en la fórmula de proporción esperada que depende tanto de $\alpha$ como $\gamma$ (observada en la ecuación 1.1). Se compara la estimación realizada con la proporción de bloques minados observada dependiendo del número de* peers *extras, realizando todo para topologías a gran escala (de* 10 000 *nodos) y con* hashing power *de* 40 $\%$ *del minero principal.*

## Radio y diámetro

Para analizar el impacto de los *peers* extras en los grafos de la topologías utilizadas, un par de métricas interesantes son sus radios y diámetros. En pocas palabras, en un grafo se puede definir su periferia como los nodos más alejados de todos los demás, siendo el máximo de estas distancias el diámetro, y su centro como los nodos más cercanos a todos, siendo el máximo de su distancia el radio. El diámetro corresponderá entonces con la mayor distancia (mínima) que hay entre todos los pares de nodos.

Formalmente, dado un grafo G y siendo $d_G$ una función que calcula la distancia mínima entre dos nodos de G, la funciones de radio y diámetro se definen como (a $\epsilon_G(v)$ se la denomina la excentricidad):

$$\epsilon_G(v) = \max_{u \in V} d_G(v, u) \tag{3.3}$$

$$radius(G) = \min_{v \in V} \epsilon_G(v) \tag{3.4}$$

$$diameter(G) = \max_{v \in V} \epsilon_G(v) \tag{3.5}$$

En la figura 3.26(b) se presentan el radio y diámetro de las topologías a medida que se aumenta la cantidad de *peers* extras agregados. Primero, centrandonos en el caso donde no se añadieron *peers* extras (es decir la topología utilizada como modelo de Bitcoin), se obtuvo un radio de 7 y un diámetro de 8. Esto indica por un lado que la distancia (mínima) entre cualquier par de nodos no es tan grande, como mucho están conectados por un camino de distancia 8, y por el otro que, dado que el radio es tan cercano al diámetro, no existen altas diferencias entre la conectividad de un nodo y otro, sino que están todos en una situación similar en lo que a distancias respecta.

Otro hecho interesante es el cambio brusco en la evolución de a ambas magnitudes cuando el atacante pasa a estar conectado al 20 $\%$ de la red: antes de este punto el diámetro se mantiene constante y el radio se

modifica de forma aparentemente exponencial negativa, pero luego ambos pasan a tener una variación de forma casi lineal. El número de *peers* extras donde se da este cambio es el mismo donde también se modificó la evolución tanto del *wasted hashing power* como de la proporción de bloques minados (ambas pasando a ser casi constantes). Atribuimos esto a que hay un mismo motivo detrás: estando conectado al $20\,\%$ de los nodos, la centralidad del atacante pasó a ser tan alta que tiene un papel fundamental en la conectividad de la red.

**Conclusiones parciales** De los experimentos realizados, se puede apreciar que varias magnitudes (especialmente la proporción de bloques minados) se mantienen constantes a partir de tener más de $2000$ *peers* extra. Se puede concluir que si un atacante (en las condiciones de red plantadas en 3.2.1) quiere aumentar el éxito de su ataque, no es necesario que logre una conectividad total a la red. Al menos para $\alpha = 0{,}4$ con una conectividad al $20\,\%$ de la red resulta suficiente (para otros valores de $\alpha$ habrán otros *thresholds* requeridos).

# 4

# Conclusiones

Al comenzar este trabajo nos planteamos la evaluación desde el punto de vista operacional del ataque de *selfish mining*. El objetivo era poder comprobar este ataque no ya desde el punto de vista teórico, sino acercándose a lo que es el sistema en su funcionamiento y escala real. En el caso de Bitcoin, lograr imitar exactamente este sistema resulta virtualmente imposible por su escala y por la necesidad de cómputo completamente fuera del alcance. Entonces, el planteo metodológico consistió en adaptar una herramienta de simulación de eventos discretos que ya había sido probada en el grupo para un objetivo similar, de manera de poder hacer una aproximación a Bitcoin. Luego de realizado el desarrollo y realizada una serie de validaciones, se pudo usar la herramienta para comprobar que en redes de topologías chicas el ataque es igual de efectivo que lo esperado de acuerdo al trabajo de Eyal y Gün Sirer [ES18].

Luego, se verificó que en topologías modeladas de acuerdo al comportamiento real de Bitcoin, el ataque siempre fue más rentable que la estrategia honesta. Al no poder calcular el parámetro $\gamma$ de estas topologías, tampoco es posible calcular la rentabilidad teóricamente esperada del atacante, por lo que no se realizó la comparación con ella. Como alternativa a este problema, se planteo el uso de la *load centrality* de los grafos subyacentes a las topologías como estimador de este parámetro, comprobándose su efectividad para su aproximación. Esto por una lado provee una herramienta tentativa para realizar futuros análisis sobre topologías con $\gamma$ desconocido y por el otro sugiere la cercanía con la rentabilidad teórica aún en redes a gran escala.

Transversalmente al análisis de rentabilidad, se estudió el impacto sobre el uso de recursos de los mineros, particularmente sobre su *wasted hashing power*. Se observó un decremento global sobre la eficiencia del uso de los recursos, especialmente respecto a los mineros honestos, ocurriendo solo en mucha menor medida sobre el atacante. Se analiza en el trabajo como esta disparidad de impacto exhibe la diferencia de información entre ambos tipos de mineros, lo que explica la rentabilidad del ataque. Por otra parte, individualmente el atacante también es afectado por esta deficiencia global de la red: lo es en mucha menor medida respecto de los mineros honestos, pero implica que sea necesario el ajuste de la dificultad de minado para que su ataque sea rentable.

Volviendo al análisis de rentabilidad y dadas las ventajas de la estrategia de *selfish mining* respecto de la estrategia honesta: ¿por qué entonces nadie lo realizó hasta ahora en Bitcoin? De este y otros trabajos de la literatura (Nayak et al. [NKMS16] brevemente exploran esta pregunta) surgen varios posibles motivos:

- Dado el *hashing power* requerido, es necesaria de una gran inversión para llevar a cabo el ataque.
- Como se mencionó previamente, es un ataque que se debe mantener en el tiempo para ser rentable (a la espera de que se ajuste la dificultad a las nuevas condiciones).

- Es un ataque fácilmente detectable por cualquiera que esté observando la red. Esta detección puede tener como consecuencia que (1) la red se actualise para defenderse (incluyendo *blacklisting* o penalidades hacia el atacante); (2) disminuya el valor de la moneda en su convertibilidad con el dólar (y por lo tanto implique menos *rewards* para el atacante).

Pese a ello, estos motivos no aplican del todo para otras criptomonedas *Proof-of-work* menos populares, que podrían entonces ser vulnerables a este ataque.

# 5

# Trabajo Futuro

Los valores de *wasted hashing power* de los experimentos de control (es decir, con minero principal honesto) se utilizaron principalmente en este trabajo para comparar con el caso de un atacante. Esto ocasionó que no se llegué a profundizar sobre el impacto que tienen distintos factores sobre su evolución, como sea la conectividad, el tamaño de la red, la distribución del *hashing power*. Uno de las principales observaciones que sería importante investigar es sobre el consistente menor *wasted hashing power* del minero principal en comparación con todos los mineros, para el experimento a gran escala variando $\alpha$, y así determinar su causa o si se trata de un artefacto estadístico.

Otro tema a ahondar de este trabajo es el modelado de las topologías (física y lógica) de la *mainnet* de Bitcoin. En este trabajo, se partió de las principales propiedades observadas en las topologías reales y se plantearon modelos consistentes con ellas. Sería significativo para un correcto modelado extender el análisis realizado, comparando los utilizados con otras familias de modelos con otros parámetros. Para esta comparación se podría utilizar los valores obtenidos para distintas métricas en la red de Bitcoin de otros trabajos, como por ejemplo en Miller et al. [MLP+15].

Por otro lado, en este trabajo brevemente se introdujeron las medidas de centralidad y se las relacionó con el parámetro $\gamma$ del ataque, pero no se llegó a profundizar demasiado en el tema. Sería interesante realizar mayores y más detalladas comparaciones entre distintas medidas de centralidad y el $\gamma$ del ataque, idealmente obteniendo alguna forma de calcular el $\gamma$ dada una topología arbitraria.

Asimismo, nos resultó muy interesante el análisis planteado por Liu et al. [LZS+18], respecto del comportamiento de la red ante múltiples atacantes (combinando también variaciones al ataque). Como trabajo futuro se podría intentar replicar los resultados obtenidos en esa publicación con el simulador construido y utilizado para este trabajo.

# A

# Apéndice

## A.1 Otros resultados de los experimentos con la estrategia honesta

### A.1.1 Experimento de topología fija, variando la conectividad

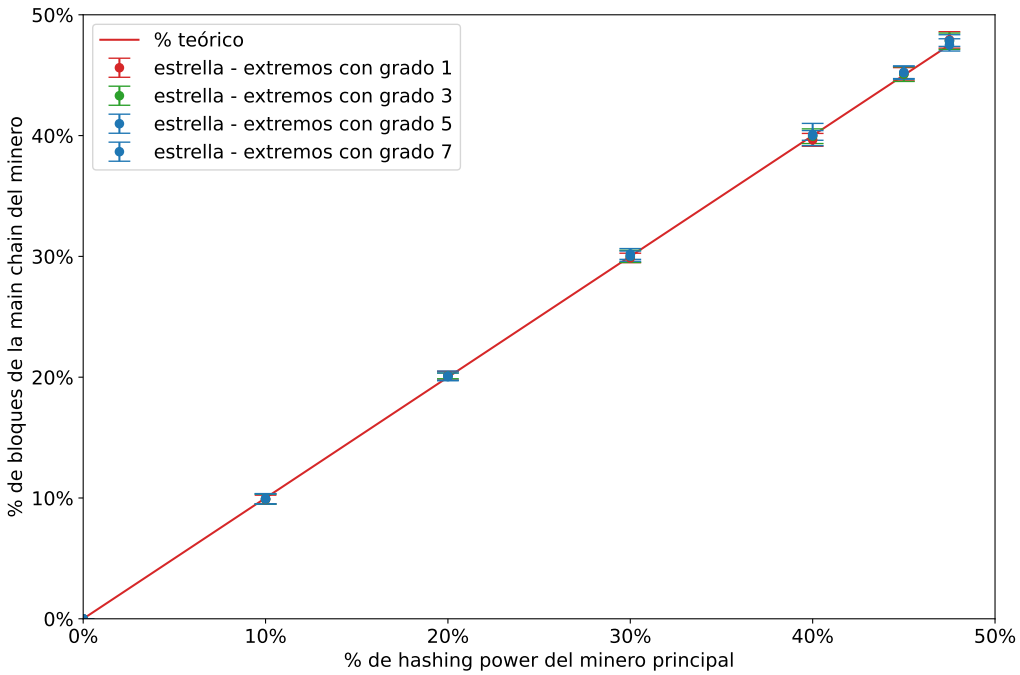



**Figura A.1:** *Comparación de la proporción de bloques minados por el minero principal honesto para cada topología chica (de ocho nodos), respecto de lo teóricamente esperado.*

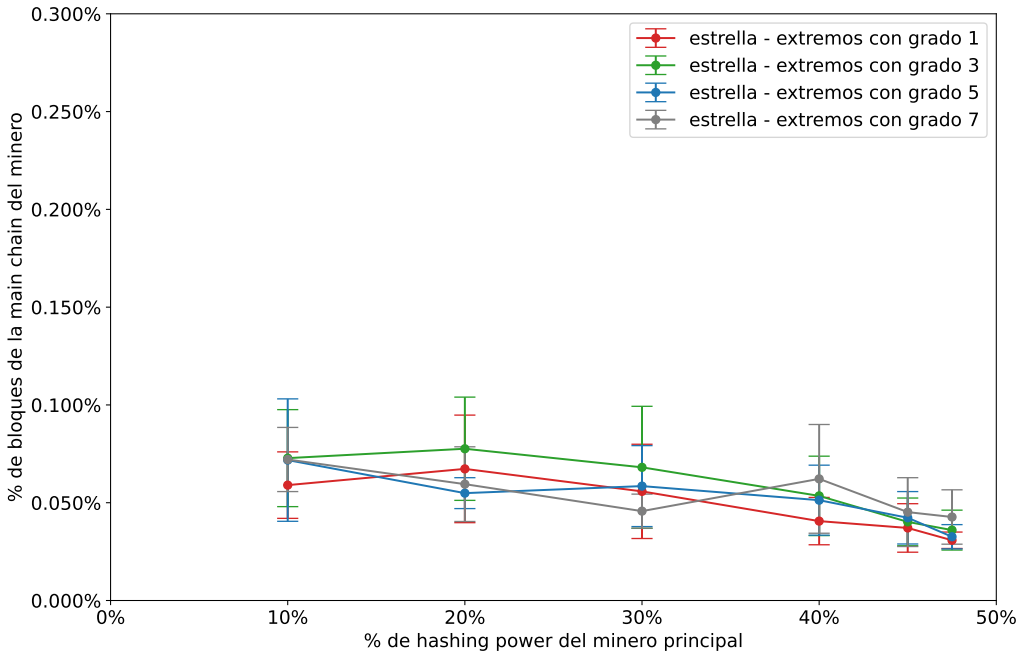



**Figura A.2:** Wasted hashing power *del minero principal honesto, para topologías chicas (de ocho nodos) con diferente conectividad. Los porcentajes indican la proporción de los* hashes *realizados durante las simulaciones por el minero principal que fue desperdiciado.*

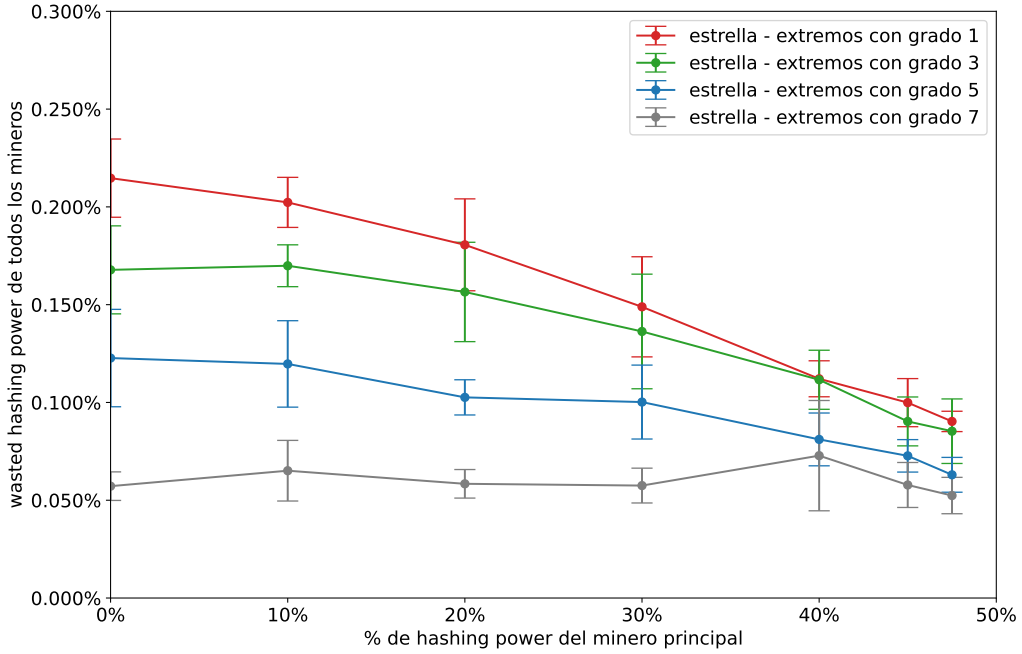



**Figura A.3:** Wasted hashing power *de todos los nodos (dado que el minero principal es honesto), para topologías chicas (de ocho nodos) con diferente conectividad. Los porcentajes indican la proporción de los hashes realizados durante las simulaciones por el minero principal que fue desperdiciado.*

### A.1.2 Experimento a gran escala, variando conectividad

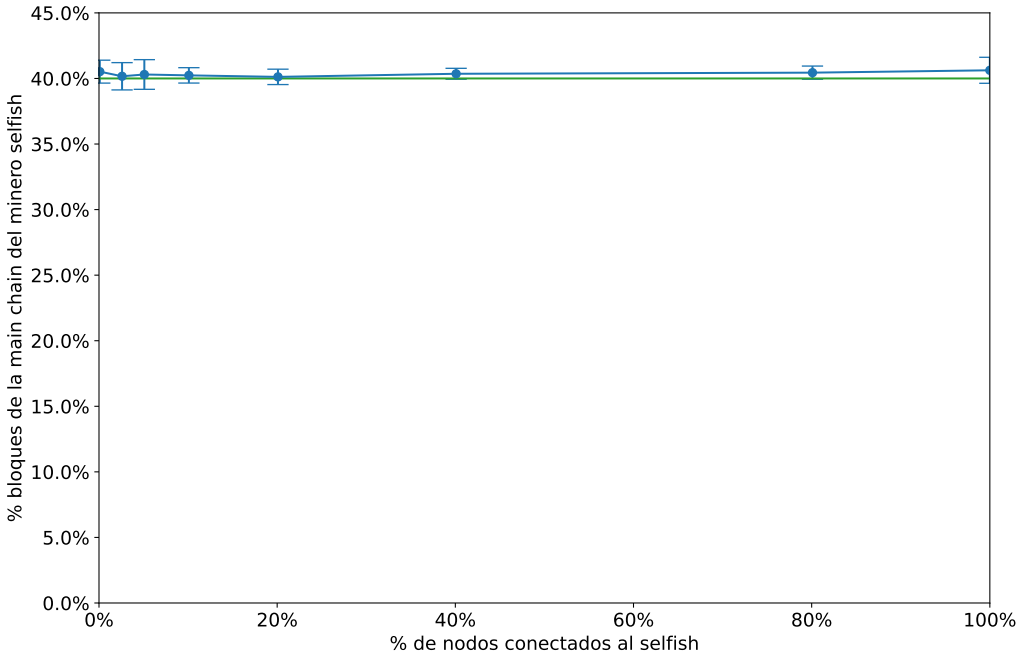



**Figura A.4:** *Proporción de bloques minados por el minero principal (con estrategia honesta) para distintos números de* peers *extras en topologías a gran escala (de* 10 000 *nodos) y con* hashing power *constante del minero principal de* 40 %.

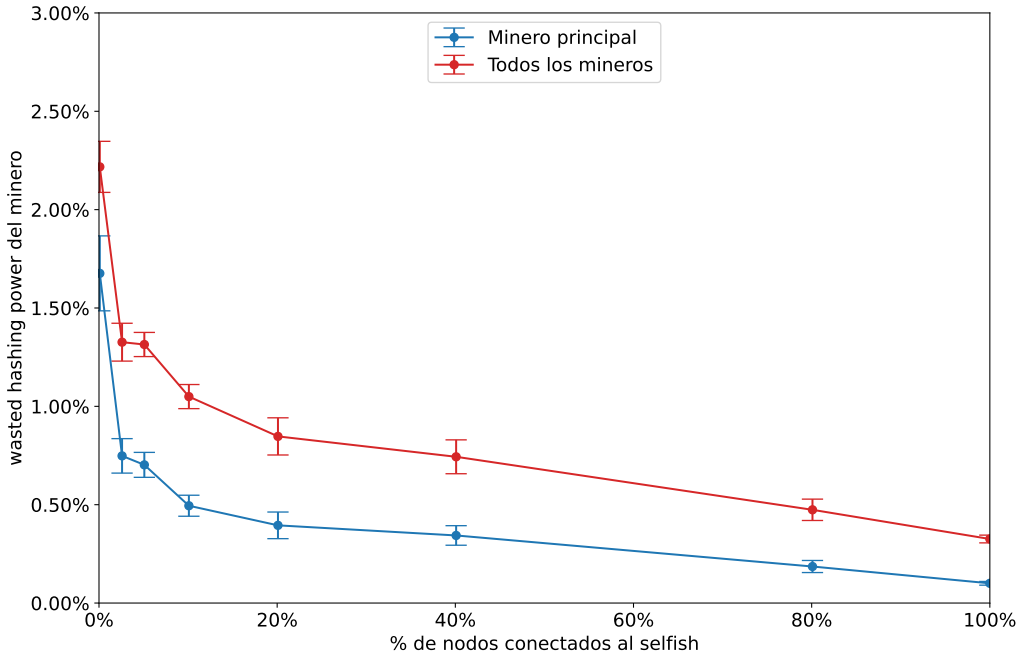



**Figura A.5:** Wasted hashing power *de cada tipo de nodo (dado que el minero principal es honesto) respecto del número de peers extras, para topologías a gran escala (con 10.000 nodos) y con* hashing power *constante del minero principal de* 40 %. *Los porcentajes indican la proporción de los* hashes *realizados durante las simulaciones por los mineros de cada tipo que fue desperdiciado.*

# Bibliografía


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

[BGLL08] Vincent Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of community hierarchies in large networks. *CoRR*, abs/0803.0476, 01 2008.

[Cai21] Gustavo Juan Cairo. Efectos de la topología de redes complejas en el aprendizaje social. Master's thesis, Depto. de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, CABA, Argentina, July 2021.

[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.

[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.

[LRDJ18] Hanqing Liu, Na Ruan, Rongtian Du, and Weijia Jia. On the strategy and behavior of bitcoin mining with n-attackers. pages 357–368, 05 2018.

[LZS+18] Hongyu Li, Liehuang Zhu, Meng Shen, Feng Gao, Xiaoling Tao, and Sheng Liu. Blockchain-based data preservation system for medical data. *Journal of Medical Systems*, 42(8):141, Jun 2018.

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

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

[NG04] Mark Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 69:026113, 03 2004.

[NGN+19] Mehrdad Nojoumian, Arash Golchubian, L. Njilla, K. Kwiat, and C. Kamhoua. *Incentivizing Blockchain Miners to Avoid Dishonest Mining Strategies by a Reputation-Based Paradigm: Proceedings of the 2018 Computing Conference, Volume 2*, pages 1118–1134. 01 2019.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 305 – 320, 2016.

[Pio19] Marcos Kevin Piotrkowski. Bitcoin: todos se benefician por igual, pero... ¿hay algunos más iguales que otros? Master's thesis, Depto. de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, CABA, Argentina, December 2019.

[SSZ15] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *arXiv preprint arXiv:1507.06183*, 2015.

[UKK11] K.-I Uoh, B. Kahng, and D. Kim. *Universal Behavior of Load Distribution in Scale-Free Networks*. 12 2011.

[WS11] Duncan Watts and Steven Strogatz. *Collective dynamics of 'small-world' networks*. 12 2011.

[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.