UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

SIMULACIÓN DE PROTOCOLOS PARA LA OPTIMIZACIÓN DEL TIEMPO DE PROPAGACIÓN DE MENSAJES EN REDES P2P BLOCKCHAIN

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MATEMÁTICAS APLICADAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MATEMÁTICO

FRANCISCO TOMÁS ALIAGA HERNÁNDEZ

PROFESOR GUÍA:
IVÁN RAPAPORT ZIMERMANN

PROFESOR CO-GUÍA:
PEDRO MONTEALEGRE BARBA

COMISIÓN:
AXEL OSSES ALVARADO

Este trabajo ha sido parcialmente financiado por Proyecto Fondecyt Regular 1220142 y CMM ANID BASAL FB210005

SANTIAGO DE CHILE
2023

Resumen

Las redes blockchain son una tecnología que puede tener un amplio campo de aplicación, sin embargo, enfrentan desafíos de escalabilidad: la latencia de propagación de la información a través de las redes blockchain entorpecen su funcionamiento y limitan su capacidad.

En este trabajo se aborda el problema de la latencia de propagación a través de modelar una red blockchain para ser simulada mediante un programa computacional. Se utiliza como base el trabajo del artículo "Perigee: Efficient Peer-to-Peer Network Design for Blockchains" en que se propone usar protocolos de selección de vecinos para optimizar la topología de la red. En el presente trabajo se elabora un modelo propio, añadiendo realismo en la forma de "node churning", limitaciones en las conexiones entre vecinos y la capacidad de realizar una simulación a escala real (de cincuenta mil nodos).

Esta nueva herramienta amplía el dominio en que la técnica presentada en "Perigee" está validada, demostrando más evidencia de que la técnica puede efectivamente mejorar la latencia de propagación y la escalabilidad en redes blockchain reales. Además, el modelado de la red sugiere naturalmente la forma de un algoritmo de selección de vecinos, para el que se demuestra empíricamente su coherencia como protocolo optimizador de topologías de red de pares a pares.

Además, se indaga en métodos estadísticos para analizar redes de escalas similares a la real, pues para ellas, los métodos exhaustivos utilizados para su análisis, producen un excesivo costo computacional.

En memoria de mis abuelos

Agradecimientos

Primero agradecer a mi familia, por el apoyo incondicional e incalculable. En especial, a mi madre Patricia, que es la mejor madre del mundo mundial. También, a mi padre Arturo, que me ha formado para ser lo que soy. Igualmente agradecer especialmente a mis abuelos, Elba, Luis, Jeanette y Arturo, por las enseñanzas fundamentales sobre las que me he edificado. Además agradecer a Constanza por su apoyo como hermana.

Agradecer también a mi profesor guía Iván Rapaport y a mi profesor co-guía Pedro Montealegre, por la oportunidad de estudiar un tema interesante, y más importante aún, por su apoyo y paciencia en este viaje. Al escribir esta tesis he descubierto que uno puede levantar una piedra para mirar debajo, y encontrar un continente. Increíble es el continente de la computación distribuida y las blockchains.

Agradecer a la Facultad de Ciencias Físicas y Matemáticas, y al Departamento de Ingeniería Matemática, por otorgarme una formación contundente y rigurosa. Podría darle las gracias a todos los profesores del departamento, pero me voy a limitar solo a los cursos más importantes de la carrera: los del primer semestre. Agradecer a José Soto por el curso de introducción al álgebra tan completo y formal, siento que en ese curso aprendí matemática verdaderamente, por primera vez. Agradecer a Jorge San Martín por su introducción al cálculo exageradamente motivante, creo que en ese curso aprendí a explorar más allá de lo que nos enseñan, que quizás es de las enseñanzas más importantes.

Agradecer la amistad de mis compañeros y amigos, Javier, Maximiliano, Amanda, Antonia, Cami y Camila, Cristóbal, Martín, Felipe y Foli, Ignacio y Nacho, Francisco, Vicente, Sara. La amistad es lo más valioso que tenemos. Yo estaré siempre para ustedes.

Agradecer a la comunidad de desarrollo de videojuegos (VGDEV) de la Universidad de Chile, por ser un espacio grato para aprender y discutir sobre esa disciplina que es , en parte, artística, y en parte, técnica, y que me permite tanto expresarme como también valorar el poder de las herramientas científicas que se aprenden en la ingeniería y la matemática. Agradecer a Gabriel, Eric, Christopher, Elías, que han aportado para que todo esto funcione, también agradecer a todos quienes participan de la comunidad. Ha sido considerable la motivación por este tema, y me ha llevado a estudiar computación en más profundidad y compartir mis conocimientos con más personas.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

En este trabajo abordamos el problema de reducir la latencia de propagación en una red Peer-to-Peer Blockchain, nuestro enfoque principal es a través de la simulación de un protocolo de optimización de la topología de la red, incorporando características inherentes y realistas de las redes del mundo real: intereses egoístas, intermitencia de participantes y número de participantes a escala real. Además, avanzamos en proporcionar herramientas para el análisis de redes a esta escala.

El texto se organiza de la siguiente manera: En el capítulo 1 se entrega una contextualización del problema, esto comprende el contexto más amplio de las blockchains y el desafío de escalabilidad en el que se encuentra. La escalabilidad es importante para que los sistemas blockchain puedan implementarse más ampliamente, y en este trabajo abordamos un problema de la escalabilidad que tiene que ver con la latencia de propagación de la información. Posteriormente, se profundiza un poco más y se aborda una lista de preliminares que dan un contexto más especializado para el problema.

En el capítulo 2 se propone un modelo que sirve para analizar la red incumbente, y además es suficiente para poder implementar simulaciones de él. Además de proponer el modelo, en el capítulo 2 se realizan análisis sobre él, para dar una perspectiva más precisa de las propiedades de las redes a simular. También, se analiza la motivación de un protocolo original EdgePriority, que aparece de manera natural una vez que se cuenta con el modelo del funcionamiento de la red. El capítulo 2 también aborda temas relacionados a la computación de métricas para la simulación, y cierra con el análisis de complejidad computacional de un programa que la implementa. Algunos detalles más técnicos se relegan a un anexo, en la medida que son resultados de utilidad, pero cuyos técnicos (como las demostraciones) no tienen directa relación con el problema.

En el capítulo 3 se enfoca en los aspectos más concretos: primero se mencionan detalles de la implementación concreta de la simulación, junto con detalles de eficiencia como el multithreading, el uso de estructuras de datos ad-hoc y la abstracción del código. Finalmente, el capítulo 3 muestra el funcionamiento de la simulación y sus resultados mediante varios experimentos, que permiten comparar distintas cualidades de la red y mostrar que los protocolos logran optimizar la red, incluso bajo supuestos más realistas que antes. Adicionalmente, se encuentran instrucciones relativas a la ejecución de simulaciones en el anexo B.1.

En total, el trabajo propone un modelo sólido sobre el que se pueden realizar simulaciones eficientes, y probar distintos aspectos interesantes de la red. Esta nueva herramienta amplía el dominio en que la técnica presentada por [38] está validada, demostrando más evidencia de que la técnica puede efectivamente mejorar la latencia de propagación y la escalabilidad en redes blockchain reales.

1.1. Contexto

1.1.1. Blockchain

Blockchain es una tecnología que sirve para replicar una base de datos de tipo libro (ledger) en una multitud de computadores (red Peer-to-Peer) a lo largo del mundo que permite garantizar seguridad en las réplicas, es decir, con garantías de que se puede confíar que lo que se encuentra en este libro es lo mismo que lo que todo el resto tiene. Su origen es en la red de pagos Bitcoin, inventada por Satoshi Nakamoto [44] y lanzada al mundo en 2008, y desde su aparición ha atraído el interés de tanto científicos de la computación, como también el mundo de las finanzas, gobiernos e incluso empresas productivas, debido a que permiten el reporte de información fidedigna, de manera accesible y transparente [26].

En el contexto de La Ciencia de la Computación, Blockchain se relaciona con el problema de consenso bizantino, un problema clásico de la computación distribuida, que consiste en generar un consenso (acuerdo del valor de una variable de estado) entre una red de computadoras que se pasan mensajes remotamente, aun cuando un porcentaje significativo de la red esté compuesta por computadoras con fallas, o bien computadoras adversariales en las que no se puede confiar. La relación con el problema de consenso bizantino es que las aplicaciones en blockchain son capaces de otorgar una garantía de las variables reportadas en su libro, por lo tanto, se pueden ver como una posible solución al problema de diseñar un sistema distribuído de computadoras capaz de lograr el mutuo acuerdo. En la práctica, la robustez técnica de las redes Blockchain ha sido validada por su uso en un contexto con un gran incentivo adversarial: el traspaso de dinero.

1.1.2. Escalabilidad en Blockchain

A pesar de su éxito, la tecnología blockchain enfrenta el desafío de la escalabilidad. Se dice que un sistema de computación es *escalable* si este es capaz de mantener su funcionamiento incluso frente al crecimiento del trabajo necesario para operarlo, producto de un cambio en sus condiciones, como puede ser: la cantidad de computadoras en la red, el número de usuarios o el número de tareas recibidas. Existen muchos sistemas que no son escalables, en que un crecimiento del número de usuarios puede causar una falla catastrófica en la operación del sistema, comprometiendo el acceso de distintos usuarios o la confiabilidad de los datos disponibles. En muchas redes blockchain (como Bitcoin o Ethereum), la variable relevante para la escalabilidad es el número de transferencias o registros que se pueden escribir en el libro por unidad de tiempo. Para hacer concreto este interés, consideremos que las tarjetas

de crédito procesan decenas de miles de transacciones por segundo; para el gobierno de un estado que busca innovar tecnológicamente, le puede interesar que un sistema blockchain permita procesar los trámites de sus millones de habitantes diariamente; y para una red de empresas productivas, puede ser interesante registrar el inventario de miles de productos en una cadena de suministro. Si se desea implementar una tecnología blockchain para resolver este tipo de problemas del mundo real, la tecnología debe ser escalable en el número de registros que puede procesar en cada momento.

Latencia en redes Peer-to-Peer

Una de las formas de abordar el asunto de la escalabilidad de número de transacciones es perfeccionar la propagación de información entre las computadoras de la red [15], lo que se puede lograr a través de la configuración adecuada de las conexiones entre computadoras de la red. El estudio de Decker y Wattenhofer [17] muestra que en Bitcoin, hay información que se recibe con diferencias de hasta 60 segundos entre pares de la red, y muestra que este tipo de demoras excesivas causan efectos indeseables sobre la red: la latencia de confirmación y los *forks*. La latencia de confirmación es el tiempo que tardan los registros en consolidarse de manera fiable dentro de la red, y es una métrica crítica para la escalabilidad, no solo en Bitcoin, sino a nivel fundamental en las redes blockchain, y está ligada directamente a la latencia de propagación de los mensajes en al red [3]. La latencia de propagación de mensajes es el tiempo máximo entre que un mensaje se emite y se recibe por todo el resto de los participantes de la red; si se piensa la red como un grafo G con pesos en sus aristas dados por el tiempo que tardan en propagarse los mensajes entre un par de nodos, la latencia de propagación puede entenderse como el diámetro de este grafo. En [17] se proponen una serie de medidas para disminuir la latencia de propagación de mensajes, específicamente en redes Blockchain, sin embargo, el problema está lejos de estar resuelto [1].

El problema de generar redes con baja latencia de propagación en redes Peer-to-Peer existe incluso desde el año 2000 [37], desde antes y más allá de las redes Blockchain, por ejemplo, en el contexto de aplicaciones de distribución de archivos [46]. Incluso antes de Blockchain, se ha considerado la importancia de que los protocolos empleados sean consistentes con los intereses egoístas de los participantes [53] [12], al igual que técnicas para la reducción del latencia como *neighbor selection* y sus implicancias en la red [40] [37].

El método de selección de vecinos consiste en medir alguna métrica observable en la red, y en base a ella, decidir sobre la conexión con otros pares de la red. Esta métrica puede ser meramente una variable de red como lo es el tiempo de comunicación, o puede ser más compleja y estar relacionada con los intereses egoístas de cada participante. Debido al deseo de mejorar la latencia de propagación, recientemente han surgido varias investigaciones sobre protocolos de selección de vecinos en Blockchain [38] [1] [59] [**?**] [51].

1.1.3. Perigee y simulaciones

En este trabajo nos hemos basado fuertemente en el paper *Perigee: Efficient Peer-to-Peer Network Design for Blockchains* [38], que sugiere un método de selección de vecinos basado en observaciones que miden la rapidez con la que otros participantes de la red son capaces de propagar la información de las transacciones. Si un participante no es suficientemente eficiente, entonces corre el riesgo de ser desconectado por sus pares, lo cual, sumado a intereses monetarios, estimula el interés por la cooperación en propagar la información de la manera más rápida posible. En [38], demuestran a través de simulaciones que su método es capaz de mejorar significativamente la latencia de propagación, pero, la simulación es solo en una red con $1,000$ nodos participantes. Al día de hoy, la red de Bitcoin cuenta con cerca de $50,000$ participantes [6]. Además, debido al gran número de características de las redes, parece complejo elaborar una teoría matemática para demostrar que este tipo de protocolos optimizan globalmente la latencia de propagación, y muchos de los estudios mencionados se limitan solo a mostrar sus resultados a través de simulaciones. Aunque los resultados son prometedores, estos solo se han limitado a redes de hasta $1,000$ nodos, y varios dejan para trabajo futuro probar los métodos en condiciones más realistas.

En esta oportunidad, tomamos el protocolo de [38] y lo llevamos a la práctica con una simulación de una red a escala real, pudiendo hacer simulaciones de $50,000$, en presencia de *node churning*, un fenómeno de intermitencia en la actividad de los nodos, ocurre en redes del mundo real. El algoritmo que lleva a cabo la simulación da lugar es eficiente, y permite realizar simulaciones con una complejidad $\Theta(T\Delta|V|\log|V|)$ cuando $|V|$ es el número de participantes en la red , Δ el número de conexiones por cada nodo y T es el tiempo simulado de la red. En un computador personal moderno, esto se traduce en una simulación que puede ser llevada a cabo en a lo máximo unas cuantas horas para redes con $|V|$ del tamaño de redes Blockchains reales, y T de la magnitud del transcurso de semanas. Nuestra simulación valida los resultados de [38] en una escala real y en la presencia de *node churn*, y por lo tanto, es un recurso valioso para tomar la decisión de adoptar un protocolo como el de [38] y otros que apuntan por mejorar la latencia de propagación mediante protocolos de selección de vecinos. Además, el código de la simulación es versátil y podría permitir evaluar otro tipo de protocolos de selección, considerar condiciones de internet adversas, y simular otras características de redes blockchain.

Métricas de desempeño

Un método relevante para la evaluación del rendimiento de redes Peer-to-Peer de miles de agentes es la medición de funciones de distribución, en inglés *Cumulative Distribution Functions* o CDFs acerca de la red [53]. En [17] se muestra que la CDF del tiempo de primer reporte de nuevos bloques está directamente conectada con la probabilidad de que se genere un *fork*; y en [38] la métrica de desempeño usada es la curva CDF del tiempo en que los mensajes de cada nodo se propagan a la mayoría ($90\,\%$) de la red. Sin embargo, el calcular la CDF de la métrica de [38] es $\Omega(|V|^2)$, y para $|V| = 50,000$ esto puede tardar días en un computador personal moderno. Para resolver esto, desarrollamos un método estadístico a partir de la desigualdad de Azuma-Hoeffding, que permite estimar la curva usando menos de un $3\,\%$ del tamaño de la red cuando $|V| = 50000$ y alrededor de $10\,\%$ para cuando la red tiene

tamaño $|V| = 10000$, manteniendo el error en una magnitud despreciable, y permitiendo el cálculo de manera rápida.

En otros estudios, como en [40], se ha mostrado interés por los efectos de la economía individual de los nodos sobre el desempeño global de la red. Nosotros proponemos una métrica de desempeño $\mathbb{E}_b[l(T_b)]$, que mide la latencia de propagación en redes blockchain incorporando las heterogeneidades, como la latencia entre los nodos, las conexiones entre pares de nodos, el poder de hash; esta métrica tiene la característica de ser una suma de métricas individuales para cada nodo, coherente con los intereses egoístas de cada uno, y que permite relacionar el desempeño global de la red, con el desempeño local. Aparte de esto, la métrica es sencilla de estimar mediante métodos de Monte-Carlo, y motivó el diseño de un nuevo método de selección de vecinos, al que llamamos *EdgePriority*, un método que tiene eficacia similar a Perigee, y además es bastante más liviano de simular y de implementar.

1.2. Preliminares

Aquí se entregan algunas nociones fundamentales básicas necesarias para entender el problema.

1.2.1. Topología Peer-to-Peer

Una red Peer-to-Peer (o P2P) es un tipo de red de computadores descentralizada donde los computadores (nodos) en la red comparten recursos directamente entre sí sin la necesidad de un servidor centralizado, osea, definida en oposición al clásico modelo de cliente-servidor. En una red P2P, cada nodo tiene la capacidad de enviar y recibir información directamente de otros nodos en la red, como compartir recursos, archivos, ancho de banda y poder de procesamiento.

En la red P2P de Bitcoin, cada usuario mantiene una copia del libro (ledger) público de todas las transacciones de Bitcoin que se han realizado en la red. Cuando se realiza una transacción, esta es anunciada en la red y validada por otros nodos, y después ser agregada en un bloque, proceso que se detalla más adelante.

Latencia de propagación

En redes P2P como Bitcoin, se utiliza una técnica llamada *flooding* para propagar los mensajes y transacciones. En el *flooding*, cada nodo envía un mensaje a todos los demás nodos con los que está conectado. Por limitaciones en los recursos de computación y de internet, un nodo tiene un límite de *vecinos* con los que mantiene conexión. En consecuencia, el *flooding* esparce los mensajes como si fueran *un rumor*, y desde que el mensaje se emite, este pasa por varios nodos hasta llegar, posiblemente a todos. A grandes rasgos, nos referimos a la latencia de propagación como el tiempo máximo desde que un nodo emite una información y

esta información se termina de propagar por toda la red. Más adelante se detallan medidas formales más precisas.

Node Churning

El node churning es un fenómeno que ocurre en las redes P2P en el que los nodos de la red entran y salen de la red con una alta frecuencia. Esto puede deberse a diferentes razones, como la conexión intermitente de los nodos, la adición de nuevos nodos a la red o la salida de nodos existentes.

El node churning puede tener un impacto negativo en la estabilidad y el rendimiento de la red, ya que la entrada y salida frecuentes de los nodos pueden causar fluctuaciones en la disponibilidad y la conectividad de los nodos en la red. Esto puede resultar en un aumento del tiempo de latencia, la pérdida de mensajes o la interrupción de la comunicación en la red.

Según Bitnodes [6], cada día entran y salen aproximadamente 200 nodos mineros de Bitcoin. El impacto de este efecto ha sido medido cuantitativamente en varios estudios [**?**] [**?**].

1.2.2. Blockchain

Una blockchain es una estructura de datos que se utiliza para almacenar datos de una manera segura e inmutable. Se compone de bloques de datos que se unen entre sí de manera secuencial (y por ello forman una cadena de bloques). En Bitcoin, se usa como la base de datos que registra las transacciones.

Hashing cryptográfico

Una función de hashing criptográfico es una función matemática que se utiliza para convertir datos de cualquier tamaño en un valor de longitud fija y única. Esta función toma una entrada de cualquier longitud y devuelve una cadena de bits de longitud fija, que se conoce como hash o valor resumen.

Para ser considerada como una función de hashing criptográfico, debe cumplir con ciertas características importantes. Propiedad (1): debe ser resistente a invertirse, lo que significa que debe ser difícil de invertir para encontrar la entrada original a partir del hash resultante. Propiedad (2): debe tener la propiedad de *no presentar a colisiones*, que es una propiedad similar al la inyectividad: es poco probable que dos entradas diferentes produzcan el mismo hash. Propiedad (3): dado una cadena de longitud fija A, debe ser difícil (requerir mucho trabajo) encontrar una entrada que produzca A.

Hash pointers e integridad de la información

Para lograr la seguridad e inmutabilidad en las blockchain, se utilizan funciones de hash cryptográfico.

Cada bloque contiene un hash que se calcula a partir del contenido del bloque (como las transacciones). Este hash actúa como un valor que identifica de manera única ese bloque en la cadena (Propiedad 2). Además, cada bloque también contiene el hash del bloque anterior en la cadena, lo que crea una cadena de dependencias en hash. En caso de que se intentara cambiar el contenido de un bloque, sería necesario recalcular el hash del bloque, y como el bloque siguiente contiene el hash del bloque actual, sería necesario cambiar el hash de todos los bloques posteriores en la cadena. Lo anterior no se puede evitar debido a la propiedad (3). Esta es una característica importante que garantiza que los datos dentro de la cadena de bloques sea inmutable y resistente a la manipulación, ya que permite validar la integridad de la información de toda la cadena.

Consenso de Nakamoto

En Bitcoin, existe un incentivo monetario para la participación de la red, y este incentivo hace posible que se mantenga un consenso sobre el estado de la red. El mecanismo que logra esto se le llama el consenso de Nakamoto, nombre popularizado por haber escrito el paper original de Bitcoin [44]. Este es el mecanismo permite validar y confirmar las transacciones en la cadena de bloques de manera descentralizada.

Cada nodo de la red, mantiene una copia completa de la cadena de bloques y trabaja en la validación de nuevas transacciones para ser agregadas a la cadena. Cuando un usuario emite una transacción, esta se considera válida si en las transacciones anteriores el usuario suma una cantidad suficiente de fondos a su nombre, y también, si es capaz de verificar la validez de su identidad usando su clave pública. Los nodos que trabajan en la validación se llaman *mineros* y además de validar transacciones, compiten entre sí para resolver un problema criptográfico complejo. El primer nodo que logre resuelve el problema criptográfico es recompensado con una cantidad de Bitcoin a su nombre y puede agregar el nuevo bloque validado a la cadena de bloques.

El problema criptográfico que deben resolver los mineros en el consenso de Nakamoto se llama *prueba de trabajo* (PoW, por sus siglas en inglés). La prueba de trabajo es un problema criptográfico que requiere una gran cantidad de poder computacional para ser resuelto. En el caso de Bitcoin, la prueba de trabajo se basa en la función hash criptográfica SHA-256. Consiste en encontrar un valor hash que cumpla con ciertas condiciones predefinidas, como tener un número específico de ceros al principio del hash. Para encontrar este valor hash, los mineros deben generar y probar diferentes combinaciones de datos y aplicar la función SHA-256 repetidamente hasta que encuentren una combinación que produzca un hash que cumpla con las condiciones predefinidas. Este proceso es fundamental en la seguridad de Bitcoin, ya que garantiza que se requiere una gran cantidad de trabajo para validar y agregar nuevos bloques.

La forma de establecer el consenso es *considerar que una réplica de la blockchain contiene*

los datos consensuados, siempre que sea la que más bloques posee, dentro de todas las otras réplicas que existen en circulación. Para entenderlo, hay que fijarse en el interés monetario de un minero: Si tiene una cantidad moderada de recursos para generar y probar valores de hash, le va a convenir aceptar la cadena más larga como el verdadero estado de la red, y a partir de ella, probar y generar nuevos valores de hash para agregar un nuevo bloque y potencialmente recibir la recompensa. De lo contrario, si elige usar una cadena más corta, estará en desventaja contra los mineros que sí se apegan a la regla de la cadena más larga, ya que esa cadena tiene más probabilidad de que el resto de los mineros trabajen sobre ella, y consigan alargarla aún más. Además, si un nodo *adversarial* deseara comenzar a propagar una versión alterada de la cadena de bloques, se podría demostrar su invalidez al encontrar las diferencias en los valores del hash, y como no es válida, a los mineros no les conviene trabajar en ella, puesto que los otros mineros tampoco lo harán, obligando a esta cadena a ser más corta que el resto. ¿Y qué sucede si hay dos réplicas con el mismo largo? En ese caso, denominado "fork", los nodos eligen arbitrariamente una de las réplicas (puede ser la que primero conocieron) y siguen minando sobre ella hasta obtener un bloque, en ese momento, la cadena puede ser más larga que la de la otra réplica, por lo que los nodos de la otra réplica cesan de trabajar en ella y nuevamente existe un consenso, En total, el mecanismo de consenso de la cadena con más bloques produce incentivos que permiten mantener una réplica válida que permanecerá desarrollándose con nuevas transacciones.

Latencia de confirmación

El proceso de minado de bloques es un proceso aleatorio, en el sentido que el próximo bloque se puede generar en un tiempo aleatorio y por algún nodo aleatorio en la red, dependiendo de qué tan rápido y quién es el que logre resolver el problema criptgoráfico. Aun para un bloque recién minado en la cadena más larga, es posible que aparezca otra cadena sin el bloque, y con muy baja probabilidad, ella logre superar a la primera, produciendo que el bloque mencionado se descarte. En el caso anterior se dice que el bloque no se confirmó, La latencia de confirmación es una medida temporal de cuánto tiempo tiene que pasar para que sea seguro que una transacción, quede en la réplica consensuada a largo plazo: dicho en simple, una transacción de hace 1 segundo tiene una garantía muy débil de seguir para siempre en la réplica consensuada, mientras que una transacción en la réplica desde hace 1 año quedará allí por siempre. Según la definición de [38] "La latencia de confirmación es el tiempo que tiene que transcurrir para que la probabilidad de que una transacción honesta sea borrada, sea suficientemente baja". Evidentemente, esta latencia idealmente debería ser lo más baja posible.

Propagación de la información en blockchain (Artículo de Decker y Wattenhofer)

En el artículo [17] llamado "Information Propagation in the Bitcoin Network", se describe cómo es que el la información se propaga por la red P2P de Bitcoin, y a partir de el modelo postula que los forks ocurren por las demoras de propagación de información en la red.

Específicamente, [17] habla sobre cómo se construye la topología de la red: cada nuevo nodo que se une consulta una tabla con direcciones de otros nodos a servidores voluntarios,

y estos le pasan una lista de conexiones, de la cual el nuevo nodo elige una cantidad d aleatoriamente, y en el futuro pregunta a sus vecinos por listas de conexiones, pudiendo conectarse así a vecinos de vecinos, etc. También se toma del hecho de que la red P2P de Bitcoin propaga sus mensajes por "flooding": Al recibir un bloque, cada nodo se lo repite a sus vecinos en caso de que los vecinos no los tengan, de esta manera, los bloques llegan a todos los nodos. Después, realiza mediciones sobre las demoras de propagación de bloques, y obtiene que hay una gran heterogeneidad en los tiempos que se pueden observar bloques: Si se pone un nodo a medir observaciones de bloques, y las diferencias entre los anuncios de un vecino y otro, se observa que algunos bloques tardan hasta 60 segundos de diferencia en ser anunciados por vecinos distintos. Además, propone un modelo para la probabilidad de que ocurra un "fork" después de originarse un bloque, en términos de la función $f(t) =$ porcentaje de nodos informados del bloque en el tiempo $t \geq 0$ desde el momento inicial. Con esa función, [17] deduce:

$$\mathbb{P}[\text{Se produzca un fork}] = 1 - (1-p)^{\int_0^\infty 1 - f(t)\mathrm{d}t},$$

para $p \in (0, 1)$. En la medida que $f(t)$ se mantiene lejos del 100 % por un largo tiempo, la probabilidad de un fork se vuelve 1. Para eliminar los forks, se debe buscar una manera de que la información fluya lo más rápido posible dentro de la red.

Como mencionamos, cuando se produce un fork, existe más de una réplica en la red, y las transacciones que están (o no) en los bloques de las réplicas en pugna pueden *no confirmarse*, lo cuál significa que pueden revertirse algunos pagos. Eso es indeseable, pero más aún, se conoce que por el mismo fenómeno, la cantidad de transferencias que se pueden hacer disminuye en presencia de forks. Luego, disminuir la latencia de propagación, y la probabilidad de forks es imperante para mejorar la escalabilidad de las redes blockchain, como se discute en el artículo [17].

Perigee

En un artículo de 2020, llamado "Perigee: Efficient Peer-to-Peer Network Design for Blockchains", se propuso un método para optimizar la latencia de propagación en las redes P2P Blockchain, mediante un tipo de algoritmo en que los nodos seleccionan las conexiones con determinados criterios, para optimizar en conjunto el funcionamiento de la red. Lo interesante de ese artículo fue originar un método que usa información que produce la red al operar normalmente, para aprovechar las capacidades de internet y de poder de minado que tiene el resto de los nodos en la red.

El protocolo presentado por ellos toma nota de los tiempos en que otros nodos observan bloques a través de vecinos, y selecciona un subconjunto de vecinos de acuerdo a un puntaje, descartando todo el resto de los nodos, cambiándolos por nuevas conexiones generadas al azar. De los protocolos, los dos más efectivos se llaman SubsetScoring y VanillaScoring, siendo el primero el más potente.

A pesar de eso, los autores identifican algunas necesidades no resueltas. Primero, simular una red en una escala realista, y también frente a la presencia de fenómenos como el "Node

Churning" y con limitaciones de visibilidad al momento de generar conexiones aleatorias. Además, destacan la necesidad de entender mejor el comportamiento de convergencia, por ejemplo, a través de cuantificar qué tan distante es la topología final de una topología ideal, al igual que encontrar una forma de definir qué es una topología ideal para este problema.

En este trabajo el problema de reducir la latencia de propagación con el mismo enfoque del artículo de "Perigee", a través de la simulación de un protocolo de optimización de la topología de la red, pero en esta oportunidad, incorporando características inherentes y realistas de las redes del mundo real: intermitencia de participantes, limitaciones en la visibilidad de nodos y usamos algoritmos eficientes para poder llevar el número de participantes a escala real. Además, avanzamos en proporcionar herramientas para el análisis de redes a esta escala.

Capítulo 2

Modelo de la red

En este capítulo se realizan dos tareas mayores: La primera, cuya sección 2.1 se llama "descripción de los componentes", está encargada de proveer un modelo de la red, en el que sea posible realizar simulaciones y llevar a cabo análisis del comportamiento de la misma. En la segunda mitad 2.2 se realizan análisis del modelo, lo cuál permite tener una perspectiva más profunda, con importancia para realizar simulaciones.

2.1. Descripción de los componentes

En esta sección se detallan un modelo con todos los componentes de la red. Comenzando desde los nodos y cómo de especifica la forma en que se relaciona con los otros nodos de la red, se sigue con las características de generación de nuevas conexiones. Después, se detallan características propios de los nodos y las latencias en la red que proveen a la red de una heterogeneidad necesaria a ser tomada en cuenta para un modelo sobre blockchains. Más tarde, se detalla el modelo de cómo la información se propaga por la red, y cómo se calcula el tiempo de propagación y otras informaciones asociadas. En la sección posterior se explica cómo se puede simular churn dentro de la red. Luego, resume todo lo descrito en un objeto llamado topología P2P. Después viene una subsección llamada dinámica de la topología P2P: bucle principal, donde se describe la ley, o la dinámica por la que la simulación se rige, y que es fundamental para el modelo y la implementación de la simulación. Se termina describiendo las métricas de rendimiento de la red, y la información que recaban los nodos para ejecutar sus protocolos de selección de vecinos.

2.1.1. Nodos, preferencias, conexiones

Se denota $V = \{1, \ldots, |V|\}$ el conjunto de nodos que participan de la red P2P. Cada nodo $v \in V$ posee un conjunto de *vecinos preferentes* $\Gamma_v^o \subseteq V - v$ con los que inicia una conexión para propagar bloques (o mensajes). Esto se codifica en el *grafo de preferencias*:

Definición 2.1 (Grafo de Preferencias) *Dado un conjunto de nodos V y sus vecinos preferentes $(\Gamma_v^o)_{v \in V}$, se define un grafo dirigido $G_{\text{pref}} = (V, E_{\text{pref}})$ en que una arista pertence según:*

$$vu \in E_{\text{pref}} \quad \text{ssi} \quad u \in \Gamma_v^o. \tag{2.1}$$

Debido a que v inicia una conexión con los nodos Γ_v^o, la red P2P puede propagar mensajes desde v hacia algún nodo $u \in \Gamma_v^o$, o bien desde u a v. Al grafo que codifica esta relación bilateral se le define grafo TCP:

Definición 2.2 (Grafo TCP) *A partir del grafo de preferencias G_{pref}, se define el grafo TCP como un grafo bidirigido $G_{\text{TCP}} = (V, E_{\text{TCP}})$, cumpliendo la condición:*

$$uv \in E_{\text{TCP}} \quad \text{ssi} \quad uv \in E_{\text{pref}} \ \vee \ vu \in E_{\text{pref}}. \tag{2.2}$$

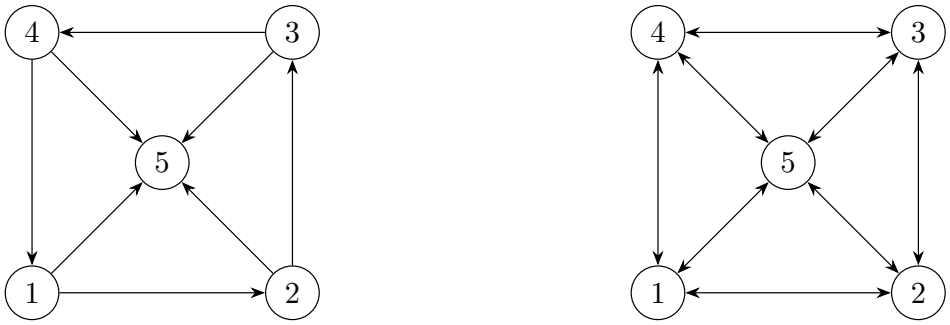


Figura 2.1: Grafos $G_{\text{pref}}, G_{\text{TCP}}$ para $V = \{1, 2, 3, 4\}$ y conjuntos de preferencias $\Gamma_1^o = \{2, 5\}, \Gamma_2^o = \{3, 5\}, \Gamma_3^o = \{4, 5\}, \Gamma_4^o = \{1, 5\}, \Gamma_5^o = \phi$.

Mantener una conexión con otros nodos en una red supone un costo: hay un ancho de banda limitado para transmitir información, y además, las computadoras tienen un número limitado de *sockets* con los que pueden entablar conexiones. Si este no fuera el caso, el problema de optimizar la red P2P se resolvería con conectar a todos los nodos entre todos. Un nodo puede tener a lo más d_{out} vecinos preferentes ($|\Gamma_v^o| \leq d_{out}$) y puede ser el preferido por a lo más d_{in} otros nodos $|\{v \ : \ u \in \Gamma_v^o\}| \leq d_{in} \ \forall u \in V$.

Suposición 2.3 (Limitaciones de número de vecinos de nodos en una red P2P) *El número de vecinos TCP es finito y no escala con V. En específico, existe M, N tal que,*

$$d_{out} \leq M, \quad d_{in} \leq N.$$

En [38] se usa $M = 8, N = 25$, en la realidad, el estándar Bitcoin explicita $d_{G_{\text{TCP}}} \leq 125$.

Se denota Δ el número máximo de vecinos entrantes y salientes del grafo G_{TCP}.

Observación Esto hace al grafo G_{TCP} un grafo *esparso*.

Definición 2.4 (Configuración de conexión) *Es una configuración individual para cada nodo $v \in V$, una tupla de enteros (d^v_{out}, d^v_{in}), y detalla un límite fijado en el número de nodos de conexiones salientes y entrantes permitidas:*

$$\forall v, \quad |\Gamma^o_v| \leq d^v_{out}, \quad |\{u : v \in \Gamma^o_u\}| \leq d^v_{in}. \tag{2.3}$$

Además, se exige que $d^v_{in}, d^v_{out} \geq 1$.

2.1.2. Elección aleatoria de vecinos

Una forma de elegir vecinos es suponer que se tiene acceso a una lista completa de nodos en V, en ese caso, a un nodo v le bastaría elegir al azar los suficientes vecinos en $V - v$ para rellenar d^v_{out}. Sin embargo, hay que considerar que es posible que al elegir nodos en V, se elija un nodo que ya tiene d^v_{in} vecinos entrantes. En ese caso, v puede seguir intentando.

Definición 2.5 (Elección Aleatoria con visibilidad global) *Se define como un método de elección aleatoria en que cada nodo tiene visiblidad de todo el resto. La elección debería ser independiente e idénticamente distribuída en $V - v$. En caso de que*

No es difícil que si $|V| \cdot d^v_{out} < |V| \cdot d^v_{in}$, entonces cada nodo que se proponga buscar nodos aleatoriamente eventualmente logra completar sus conjunto de nodos de salida.

La elección también puede ser con restricciones más realistas. En la red Bitcoin, no todos los nodos se encuentran en una lista pública. Sin embargo, el estándar de Bitcoin [43] estipula que los nodos pueden compartir direcciones de nodos entre sí. Estas direcciones se van almacenando en un gestor de direcciones llamado *addrman*. Una forma de modelar esto es que los nodos tengan acceso a las direcciones de los vecinos de sus vecinos, o bien, la dirección de los vecinos a k saltos.

Definición 2.6 (Vecindad local de k saltos) *Dado un nodo $v \in V$, el grafo G_{TCP} y un entero positivo k, se define,*

$$V(v, k) := \{u \in V \ : \ D(u, v) \leq k\}. \tag{2.4}$$

Aquí $D(u, v)$ es la distancia por saltos en G_{TCP}.

Observación Con esta vecindad local $V(v, k)$ se logra el objetivo planteado por [38] de hacer una simulación con limitaciones de visibilidad.

Conexiones aleatorias

Debido a que se desea implementar una simulación con este modelo, se debe buscar un método (un programa) de cómo llevar a cabo la generación de conexiones aleatorias entre cada v y $V(v, k)$, teniendo en cuenta que el método debe no introducir sesgos en la forma de selección.

Si la generación de conexiones entre nodos fuera totalmente independiente entre cada nodo, se podría realizar de manera simple: Cada nodo v pone una lista con los nodos de L (que puede ser $V - v$ o $V(v,k) - v$ dependiendo del grado de visibilidad), luego genera una permutación aleatoria de L y elige los primeros d_{random} elementos de acuerdo a esta permutación. Este método tiene un problema, y es que como las conexiones entrantes a un nodo son limitadas por d_{in}^v, podría ser que dos nodos u,v elijan aleatoriamente un nodo w que solo tenga para una única conexión más. En ese caso, u o v podrían bloquear la conexión del otro. ¿Cómo decidir la prioridad? En vez de buscar una respuesta definitiva, se plantea el siguiente método:

Se mantiene una estructura de datos R que tiene la posibilidad de extraer elementos al azar, uniformemente. Cada nodo v, independiente del resto, puede publicar en R un elemento llamado **solicitud** (v, v_r) explicitando la identidad del nodo v y el número de conexiones aleatorias que necesita, $r_v \in \mathbb{N}$. No se puede solicitar con $r_v = 0$. Luego, mientras no se estén publicando solicitudes, R extrae un elemento al azar, (v, r_v) y el elemento v gana la posibilidad de elegir un nodo al azar en $V(v,k) \cap \{u : |\{w : u \in \Gamma_w^o\}| < d_{in}^u\}$, actualizándose la solicitud a $(v, r_v - 1)$ si $r_v > 1$, y no se hace nada en el otro caso. Luego R repite el procedimiento hasta que no tenga solicitudes pendientes.

La estructura de datos tarda tiempo $O(\sum_{v \in V} v_r)$ en recibir, y también en procesar todas las solicitudes. Por ahora se prefiere no profundizar en los detalles de esto, pero si se desea, el análisis e implementación está en A.2 y A.2.2 del anexo, y se vuelve a tocar nuevamente más adelante.

Inicialización: Fase Bootstrap

En el estándar de Bitcoin y otras redes peer-to-peer, los nodos entran en conexión con la red a través de una tabla pública en la que seleccionan una cantidad de vecinos al azar para contactarse. Así se forma su conjunto Γ_v^o original. En el modelo, para inicializar la red P2P, se considera que existe una tabla única *NodePool* (R como en la subsección anterior) en la que todos los nodos pueden encontrar a todo el resto de los nodos. Como existe el incentivo para cada nodo de tener buena conexión al resto de la red, la selección original debería cumplir $|\Gamma_v^o| = d_{out}^v$.

En una simulación, es necesario inicializar la red P2P de alguna manera, y en este caso, se usa el esquema descrito de selecciones aleatorias, pero en esta primera inicialziación, se utiliza la visibilidad global. es decir, durante esta fase, cada nodo puede potencialmente conectarse a cualquiera. A esto se le llama fase *Bootstrap*. Después de la inicialización, se considera usar $V(v,k)$ para simular el hecho de que los nodos comparten vecinos entre sí.

Observación Si todos los nodos mantuvieran el mismo $d_{out}^v = k$, y las selecciones de conjuntos son independentes y uniformes, el grafo G_{TCP} tendría la estructura del grafo $G_{V,k-out}$.

En el mundo real, es común encontrar que las redes no son realmente homogeneas, existiendo nodos con una gran cantidad de conexiones mientras que otros con muy bajas conexiones. A la vez, es bastante probable de que, independiente de que la mayoría los nodos no estén conectados entre sí, exista un número pequeño de saltos necesarios para recorrer toda la red.

Este fenómeno se comparte con otros tipos de redes, como las redes sociales, y las redes resultantes se pueden comparar con los modelos de Watts-Strogtaz y de Barabási-Albert.

2.1.3. Heterogeneidad de la red

Al modelar una red blockchain, es necesario incorporar con características que diferencian el rendimiento de cada nodo en la red. Existen nodos que poseen fuerte poder criptográfico, y también nodos que tienen una muy buena conexión a internet. A continuación se incorporan estas características de heterogeneidad, importantes para para describir el comportamiento de la red blockchain.

Suposición 2.7 (Características de los nodos y de la internet) *Se asume que las siguientes condiciones cambian de manera lenta en el tiempo, por lo que se consideran constantes del problema:*

- *Todo nodo $v \in V$ posee un poder computacional para producir bloques $F_v \geq 0$.*
- *La capacidad de un nodo v de ser el que genera un bloque en determinado momento, está dado según su poder criptográfico en comparación con el resto. En concreto, según el valor en una distribución de poder de hash $(f_v \ : \ v \in V)$ en la que $f_v \coloneqq F_v / \sum_{u \in V} F_u$. Es una distribución de probabilidad.*
- *La generación de un bloque sucede en un espacio de probabilidad, que se denota por conveniencia $(\Omega_f, \mathbb{P}_b)$. Los nodos que se originan se identifican con nodos en V, y están dados por variables aleatorias $b_1, \dots b_n, \dots : \Omega_f \longrightarrow V$ independientes e idénticamente distribuidas, según la ley:*

$$\mathbb{P}_b[b_i = v] = f_v \quad \forall v \in V, \forall i \in \mathbb{N}. \tag{2.5}$$

- *Para comunicar un bloque desde un nodo u hacia otro v, ocurre una demora (latencia) especificada por $l_{uv} \geq 0$. No necesariamente se tiene que $l_{vu} = l_{uv}$. En ocasiones, se escribe l_{uv} como $\Delta_u + \partial(u, v)$, descomposición que toma Δ_u como el tiempo inherente que el nodo u tarda en procesar un bloque antes de propagarlo, y $\partial(u, v)$ el tiempo que tarda el mensaje viajar por la internet. Se exige que la función l cumpla:*

$$l_{uv} = l(u, v) \quad l : V \times V \to \mathbb{R}_0^+, \quad l(u, u) = 0 \quad \forall (u, v) \in V \times V. \tag{2.6}$$

  En la práctica y en las simulaciones, l_{uv} debería medirse en milisegundos ([ms]). *En la posterior implementación, se llama a l topología de internet. Cabe notar que no es de la topología de red porque captura los posibles pesos entre todos los nodos de V, e idealmente la topología de red debería aprovecharse eficientemente de la topología de internet.*

2.1.4. Propagación de la información

En el modelo, se incorpora una suposición de cooperación egoísta para la comunicación de bloques: Debido a que existe la presencia de un protocolo de selección de vecinos que prioriza la veloz propagación de la información, los nodos tendrán el incentivo de propagar los bloques a sus vecinos tan pronto como sea posible, pues de lo contrario, actuarían en contra de sus propios intereses de mantenerse bien conectados con el resto de la red, ya que correrían el riesgo de ser desconectados por sus vecinos al no entregar la información a tiempo, siendo comparativamente menos deseables que otros vecinos.

Suposición 2.8 (Propagación de bloques por TCP) *Si dos nodos $u, v \in V$ están conectados por TCP, entonces se propagarán mutuamente los bloques que observen, tan pronto como sea posible (excluyendo propagar b desde u a v si v propagó b a u).*

Observación En las redes P2P reales, este tipo de incentivos se consideran necesarios para que los nodos participantes desempeñen roles colaborativos. Se puede ver el mecanismo de *Choking* en BitTorrent [12], o bien [53] en otras redes más generales.

Con lo anterior, ya se puede definir el tiempo de propagación de un bloque, y otras estructuras en la comunicación.

Definición 2.9 (Tiempo de propagación y predecesores) *Se define $t(b, w)$ como la demora en tiempo de propagación de un bloque b hasta un nodo w:*

$$t(b, w) \coloneqq \min \left\{ \sum_{uv \in E(\overrightarrow{P})} l_{uv} \ : \overrightarrow{P} \text{ es camino en } G_{\mathrm{TCP}} \text{ con extremos } b, w \right\}. \tag{2.7}$$

También se puede interpretar $t(u, v)$ como la demora en tiempo de propagar un bloque entre u hasta v. Notar que la definición descansa en las suposiciones 2.8 y 2.7. Aparte, se asume que el camino $\overrightarrow{P}$ que minimiza (2.7) es único, en la suposición 2.10. Dado que $\overrightarrow{P}$ es único en G_{pref}, para cada b, se dirá que para un par de nodos $u, v \in V$,

u es predecesor de v para b si u fue el primero en propagarle b a v.

Esta relación se denotará por $u \to_b v$, o también $u = \mathrm{pred}_b(v)$. La relación define un grafo dirigido acíclico en V, que se denota T_b. En el caso en que G_{TCP} es conexo, T_b conecta b con todos los nodos, y sus aristas se pueden interpretar como un árbol generador. Por convención, $t(b, v) = +\infty$ y $\mathrm{pred}_b(v) = -1$ si b, v no existe un camino entre ellos a través del grafo G_{TCP}. Además, $\mathrm{pred}_b(v) = v$ siempre que v genera b (y también $t(b, v) = 0$).

Se agrega la suposición de unicidad de $\overrightarrow{P}$:

Suposición 2.10 *Para cada $b \in V, w \in V$, existe un único $\overrightarrow{P}$ que realiza el tiempo de propagación (2.9). Esta es una suposición simplificatoria. Es poco probable que existan dos caminos de conexión que tarden exactamente lo mismo.*

El arreglo $(t(b, u) : u \in V)$ y el grafo T_b se pueden calcular usando el algoritmo de Dikjstra.

2.1.5. Nodos activos e inactivos: simulando Churn

Una forma de aumentar el realismo de la simulación es considerar que existen nodos que no están activos en la red. Nodos que no generan bloques; que si bien, pueden estar en la lista de conexiones preferentes, no transmiten información; y tampoco realizan procesamiento ni selección de vecinos.

La forma de incorporar esto en el modelo, es definir un conjunto $U \subseteq V$ de nodos inactivos, y considerar en las estructuras anteriores a G_{TCP} (2.2) como el grafo (V, E_{TCP}):

$$uv \in E_{\text{TCP}} \quad \text{ssi } (uv \in E_{\text{pref}} \vee vu \in E_{\text{pref}}) \wedge (u \notin U \wedge v \notin U).$$

Es decir, las conexiones transmiten información siempre que algún nodo sea preferencial de otro, y ninguno de los nodos sea inactivo. Aparte de eso, es necesario re-calcular el poder criptográfico que da forma a la ley $\mathbb{P}[b_i = v]$ (2.5) solo considerando que los nodos activos pueden generar bloques:

$$\mathbb{P}[b_i = v] = \begin{cases} F_v / \left(\sum_{u \in V-U} F_u\right) & v \in V - U, \\ 0 & v \in U. \end{cases} \tag{2.8}$$

El conjunto $U \subseteq V$ puede ser cambiado dinámicamente por factores internos o externos. Notar que si $U = \phi$, las definiciones anteriores son idénticas con las nuevas. Una forma de simular *churn* es mediante la entrada y salida de nodos en y hacia fuera de U de forma aleatoria.

2.1.6. Topología P2P

Ya se tiene una descripción de los componentes fundamentales de la red. A partir de esto, se condensa en una sola estructura: Se define la topología P2P como el conjunto de nodos, su estado de actividad, la manera en que están conectados, las demoras de comunicación entre ellos, y la distribución de poder de hashing.

Definición 2.11 (Topología P2P) *Dado V un conjunto de nodos, de los cuales $U \subseteq V$ son inactivos, con el grafo de preferencias G_{pref}, su correspondiente grafo de conexiones G_{TCP} y una colección de latencias par a par $l : V \times V \to \mathbb{R}_0^+$, y una distribución de poder de hash $(f_v : v \in V)$, se define topología P2P como la tupla $\mathcal{N}$:*

$$\mathcal{N} = (V, U, G_{\text{pref}}, G_{\text{TCP}}, (l_{uv})_{(u,v) \in V \times V}, (f_v)_{v \in V}) \tag{2.9}$$

2.1.7. Dinámica de la topologia P2P: Bucle Principal

Ahora se define cómo es que la topología P2P evoluciona en el tiempo. La red está constantemente procesando transacciones, que los nodos consolidan en bloques, y estos bloques se propagan como mensajes por la red. Debido al esquema de *Proof of Work*, los bloques válidos son propagados cada cierto tiempo, en orden de minutos. En [38] se propone que el algoritmo de selección de vecinos utilice los bloques recibidos por los nodos como las observaciones que se usan para decidir la selección de vecinos. Debido a esto, la topología P2P evoluciona a medida que se originan bloques válidos en la red. Hay que considerar que la propagación de bloques tarda minutos; que la ejecución de un algoritmo de selección por parte de un nodo tarda milisegundos; y el establecer o cerrar una conexión tarda un par de segundos. Por ende, es extremadamente poco común que la propagación de un bloque ocurra al mismo tiempo que un vecino se desconecte o se conecte a otro: lo primero ocurre tras varios minutos, mientras que los cambios topológicos ocurren cada a lo más, unos segundos, y tras observarse un nuevo bloque. El hecho de que los bloques se originan en un orden de tiempo mayor a la ejecución de las rutinas de procesamiento, descarte y establecimiento de nuevas conexiones, es posible suponer que la propagación de los bloques se realiza mayoritariamente mientras $G_{\text{pref}}, G_{\text{TCP}}$ están fijos.

Suposición 2.12 (Propagación de bloques es con la topología congelada) *La propagación de bloques ocurre mientras $G_{\text{pref}}, G_{\text{TCP}}$ están congelados.*

Se puede resumir esta dinámica en un bucle infinito que se define como el *bucle principal*:

Definición 2.13 (Bucle Principal) *Mientras la red esté activa, se ejecuta el bucle principal:*

1. *Algún nodo $u \in V$ genera un bloque b_i que se propaga por la red. b_i (o u) está dado por una muestra de las variables aleatorias de (2.5).*
2. *Los nodos en $v \in V$ reciben la información del bloque b_i en tiempo $t(u, v)$ y gracias a sus predecesores en T_u.*
3. *Los nodos $v \in V$ que observan b_i ejecutan su rutina de procesamiento.*
4. *De acuerdo a lo observado, los nodos v podrían elegir desconectarse de algún vecino preferente.*
5. *De acuerdo a sus intereses, los nodos v podrían buscar entablar conexiones con nuevos nodos, seleccionados de manera al azar.*
6. *Repetir desde 1.*

Los pasos 3 y 4 se realizan en cada nodo de manera independiente. A pesar de eso, el paso 5 no puede hacerse de manera independiente entre nodos puesto que dependiendo del orden de conexiones entre nodos, se puede producir un conflicto con la configuración de conexión (2.4).

El bucle principal no queda totalmente definido; es necesario detallar más los pasos. Más adelante, en la sub-sección 2.1.9 se definen posibles mecanismos de cómo es que los nodos realizan sus observaciones en y ejecutan sus rutinas de procesamiento (paso 3) y descarte de vecinos (paso 4) a partir de observaciones; En 2.1.2 se definen mecanismos para la elección aleatoria de vecinos (paso 5).

2.1.8. Métricas

En esta sección se definen formas de evaluar el rendimiento de la topología P2P, en función de qué tan eficiente es para propagar la información a tiempo. Se asume que G_{TCP} es conexo.

Métrica de percentiles

Una métrica importante es la que se usa en [38]. Esta métrica cuantifica el tiempo necesario para que un bloque llegue desde un nodo v a *suficientes* nodos en la red, en el sentido de que esos nodos posean el 90 % del poder criptográfico.

Definición 2.14 (Métrica de percentiles λ_v^α) *Sea $\mathcal{N}$ la topología P2P como en 2.11. Sea $v \in V$, y $\alpha \in (0,1)$ un porcentaje. La métrica objetivo λ_v^α es el tiempo que un bloque de v tarda en propagarse hasta una α porción del hash de la red,*

$$\lambda_v^\alpha \coloneqq \text{mín}\left\{ t \geq 0 \ : \ \sum_{\substack{u \in V: \\ t(v,u) \leq t}} f_u \geq \alpha \right\} \tag{2.10}$$

En esta definicion $t(v,u)$ es la interpretación nodo-nodo de 2.9. Si se escribe λ_v sin especificar α, entonces se asume que $\alpha = 90\,\%$, como la métrica central en [38].

Se define la métrica de percentiles global, λ^α como la CDF *de todos los valores λ_v^α:*

$$\lambda^\alpha \coloneqq \text{CDF}(\lambda_v^\alpha \ : \ v \in V) = (F(t))_{t \in \mathbb{R}} = \left\{ t \longrightarrow F(t) = \frac{1}{|V|} \sum_{v \in V} 1_{\lambda_v^\alpha \leq t} \right\}. \tag{2.11}$$

Idealmente, la curva CDF *debería concentrar su masa de probabilidad en el menor valor posible. Eso significaría que la gran mayoría de los nodos en V tardan poco tiempo en propagar sus mensajes al porcentaje α de hashing en la red.*

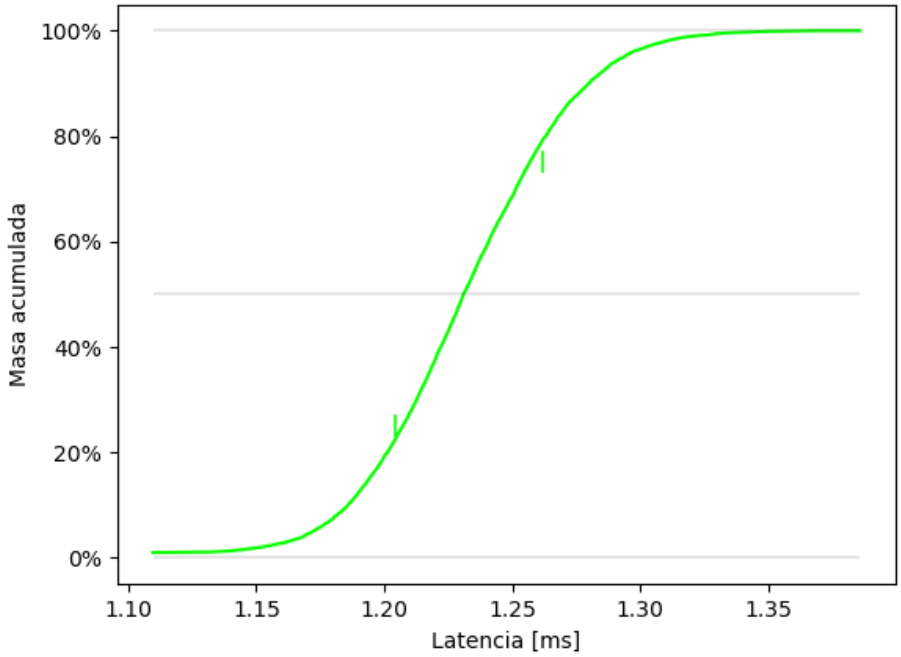
Observación Ya que $\mathbb{P}_b$ es una medida de probabilidad, se puede ver que la métrica λ_v^α es igual a:

$$\lambda_v^\alpha = \text{mín}\left\{ t \geq 0 \ : \ \mathbb{P}_b[W_{t,v}] \geq \alpha \right\}$$

Aquí, $W_{t,v} \subseteq \Omega$ son los los eventos $W_{t,v} \coloneqq \{\omega \in \Omega_f \ : \ t(b_i(\omega), v) \leq t\}$. $\hat{t} = \lambda_v^\alpha$ es el mínimo tiempo en que esta probabilidad sobrepasa α, y esta probabilidad es la de que el siguiente bloque tarde en comunicar la información en tiempo $\hat{t}$. El interés egoísta de minimizar esta función es porque v está minando bloques para ganar la recompensa monetaria, pero esta

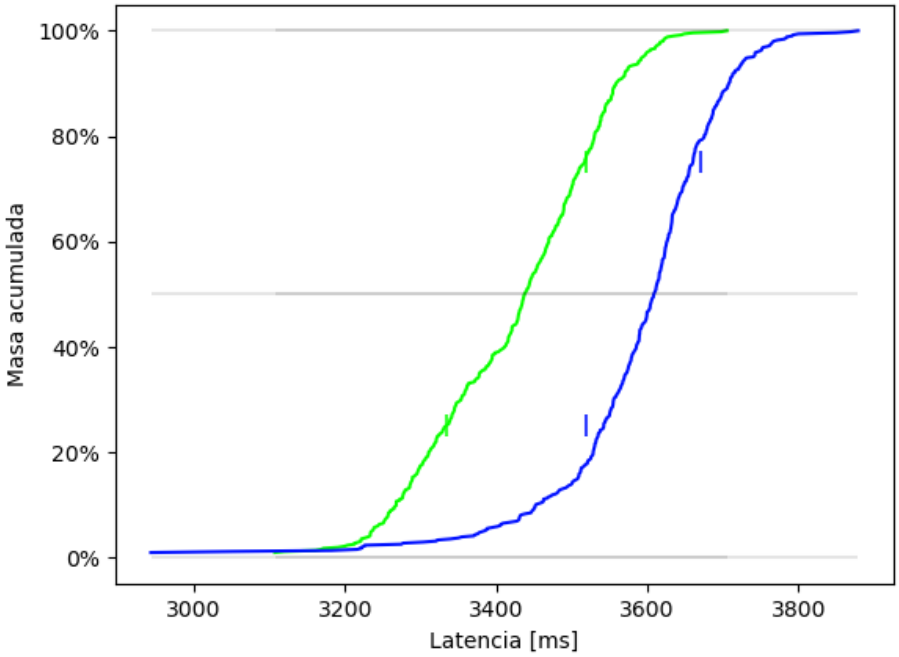
recompensa solo se obtiene si su bloque v es el elegido en la cadena más larga. Entonces, a v le interesa que gran parte de la red (ej: el 90 % del poder de hashing) esté lo suficientemente cerca para que su bloque sea agregado antes de que *otro nodo genere un bloque*. Por otro lado, al informarse en un tiempo bajo, v también evita desperdiciar cálculos en un puzzle criptográfico que puede ya haber sido resuelto en un bloque válido.

Figura 2.2: CDF de la métrica λ^α para una inicializada aleatoriamente, para una red de $V = 10{,}000$ nodos y $\alpha = 90\,\%$. La curva fue calculada por fuerza bruta. Las líneas grises demarcan los percentiles $0, 50$ y 100 y las pequeñas barras verdes son el promedio de la muestra censurada entre los percentiles 0 y 50, y en los percentiles 50 y 100.



La CDF otorga una perspectiva cuantitativa y cualitativa del comportamiento de la red, por ejemplo, en la figura 2.2 se observa la curva para una red inicializada aleatoriamente, sobre una topología en que los nodos se incrustan en $[0, 1]^{20}$ y $l_{uv} = ||x - y||/\sqrt{20}[s]$. Bajo esa topología, cualquier nodo tarda en mandar un mensaje directo a cualquier otro en $1[s]$, porque esa es la latencia máxima posible (la diagonal de $[0, 1]^{20}$), sin embargo, el tiempo para propagar mensajes por la red a el $\alpha = 90\,\%$ de la red, el tiempo puede variar entre $1, 10$ hasta $1, 40[s]$, centrándose con el 50 % de los nodos tardando un poco más de $1, 22[s]$. Además, el gráfico nos muestra que la cantidad λ_v^α distribuye de manera muy similar a una variable aleatoria normal. La figura 2.3 muestra otras dos curvas, la CDF de una red recién inicializada (azul) y la CDF de una red que lleva ejecutando un protocolo de selección de vecinos, por 1,000 bloques. En la figura se puede apreciar que dado que la curva verde está "corrida a la izquierda", produciendo una disminución generalizada de la latencia.

Figura 2.3: Curvas CDFs para contrastar una red inicializada al azar (azul) con la misma red tras 1,000 bloques propagadados bajo protocolo de selección de vecinos (verde). $V = 10{,}000$ nodos y $\alpha = 90\,\%$. La curva fue calculada con una muestra aleatoria de 500 nodos por curva. Las líneas grises demarcan los percentiles $0, 50$ y 100 y las pequeñas barras son el promedio de cada muestra, censurada entre los percentiles 0 y 50, y en los percentiles 50 y 100.



Métricas de tiempo esperado

Otra métrica interesante viene de utilizar el comportamiento en probabilidad. Hay que recordar que $u \to_b v$ denota que el bloque b fue transmitido por primera vez desde u hacia v, entre todos los vecinos de v en G_{TCP}, denotados por $N(v)$.

En base a esto, se puede formar la siguiente métrica para v:

Definición 2.15 (latencia ponderada individual)

$$S_v := \sum_{u \in N(v)} l(u,v)\ \mathbb{P}_b[u \to_b v] = \sum_{u \in N(v)} l(u,v)\ \mathbb{E}_b[1_{u \to_b v}]. \tag{2.12}$$

Notar que S_v es (casi) una combinación convexa de las latencias de sus vecinos.

Observación S_v no es necesariamente una combinación convexa: si la red es conexa, entonces,

$$1 = \mathbb{P}[v \to_b v] + \sum_{u \in N(v)} \mathbb{P}[u \to_b v],$$

y la magnitud $\mathbb{P}[v \to_b v] = f_v$ puede ser positiva.

La latencia ponderada individual es interesante por el siguiente cálculo:

$$\sum_{v\in V} S_v = \mathbb{E}_b\left[\sum_{v\in V}\sum_{u\in N(v)} l_{uv}\ 1_{u\to_b v}\right] \tag{2.13}$$

$$= \mathbb{E}_b\left[\sum_{v\in V}\sum_{u\in V} l_{uv}\ 1_{u\to_b v} 1_{u\in N(v)}\right] \tag{2.14}$$

$$= \mathbb{E}_b\left[\sum_{v\in V}\sum_{u\in V} l_{uv}\ 1_{u\to_b v}\right] \tag{2.15}$$

$$= \mathbb{E}_b\left[\sum_{uv\in E(T_b)} l_{uv}\right] \tag{2.16}$$

$$= \mathbb{E}_b\left[\text{peso de latencias } l \text{ de aristas de } T_b\right] \tag{2.17}$$

Minimizar $\sum_v S_v$ significa globalmente minimizar las latencias de todo el conjunto de aristas que por primera vez informan los bloques a los nodos (en valor esperado sobre los bloques generados).

Definición 2.16 (latencia global esperada)

$$S := \sum_{v\in V} S_v = \mathbb{E}_b\left[\sum_{uv\in E(T_b)} l_{uv}\right]. \tag{2.18}$$

Observación La latencia global esperada contabiliza la latencia sólo de las aristas utilizadas para propagar mensajes por la red, y las pondera por la cantidad de veces que son usadas en por bloques generados según la ley f. Por ejemplo, podría existir una red con una gran cantidad de conexiones innecesarias, es decir, que no sirven como via de primera información de bloques (no están en T_b) y además de muy alta latencia, pero que estas aristas no pesen en la métrica puesto que no serán usadas para propagar mensajes.

Se nota que para cualquier b, T_b es conexo, y por lo tanto, se puede deducir que S es mayor al peso de latencia del árbol generador de peso mínimo (AGPM) de G_{TCP} y también se sabe que el AGPM no depende de b (aleatorio), pese a eso, el peso de AGPM no sirve para reflejar la heterogeneidad en la latencia de propagación que se da por los distintos orígenes aleatorios de b. Es preferible tener un métrica que considere eso, pero puede servir tener una cota inferior. Como corolario de lo anterior, un AGPM del grafo completo K_V con pesos de aristas l_{uv} es cota inferior de S para cualquier G_{TCP}. (Nota: para este párrafo se asumió que $l_{uv} = l_{vu}$, para que AGPM tenga el sentido clásico).

2.1.9. Observaciones y selección de vecinos

La decisión sobre qué nodos contactar es el único poder que tienen los nodos de mejorar su condición en la red, pero ¿Cómo puede un nodo tomar una decisión inteligente para establecer sus preferencias? En [38] abordan el problema a traves de *observaciones*, que permiten a cada nodo informarse del estado de la red y así cambiar sus preferencias. A continuación se definen las observaciones principales de [38], pero también se añaden otras que pueden ser importantes para la formulación (e implementación) de una simulación más general.

Definición 2.17 (Observaciones) *Sea $\mathcal{N}$ una topología (2.11), se considera $T_b, \mathrm{pred}_b(\cdot), t(\cdot,\cdot)$ como en 2.9. Sea v un nodo, y una colección de bloques $B = \{b_1, \ldots, b_n\}$ en la misma componente conexa que v (estos bloques serían los que fueron posibles de percibir por v), se definen algunos arreglos que v puede utilizar para tomar decisiones:*

- *Primeros tiempos: arreglo con el primer momento en que v se entera de cada $b \in B$,*

$$(t_v^b \coloneqq t(b,v) : b \in B) \tag{2.19}$$

  Como b, v están en la misma componente conexa, $t_v^b < +\infty$.

- *Predecesor: arreglo de vecinos que primero propagan $b \in B$ a v,*

$$(\mathrm{pred}_b(v) : b \in B) \tag{2.20}$$

- *Latencias entre vecinos: el arreglo*

$$(l_{uv} : u \in N_{G_{\mathrm{TCP}}}). \tag{2.21}$$

- *Tiempos de los vecinos:*

$$\left(t_{uv}^b : b \in B, u \in N_{G_{\mathrm{TCP}}}(v)\right) \tag{2.22}$$

  Aquí,

$$t_{uv}^b \coloneqq \begin{cases} t(b,u) + l_{uv} & \text{si } \mathrm{pred}_b(u) \neq v \\ +\infty & \text{si } \mathrm{pred}_b(u) = v \end{cases}$$

  Es decir, el tiempo en que los vecinos propagan b hacia v, la condición $\mathrm{pred}_b(u) \neq v$ excluye que el bloque haya sido propagado desde v a un vecino u.

Hay que notar que estas definiciones descansan en la suposición 2.8. Ahora se definen las observaciones normalizadas.

$$\tilde{\mathbb{O}}_v = (t_{uv}^b - t_v^b : u \in \Gamma_v^o, b \in B) \tag{2.23}$$

Comentario: En una red real, los tiempos en que los bloques se observan no cumplirían $t(b,b) = 0$ puesto que los bloques se deberían estampar con el tiempo de origen ($\neq 0$). Un buen modelo de este fenómeno de esto añadiría el tiempo en que se originan $t_b \in \mathbb{R}$ y el tiempo de propagación para así contar con el tiempo real en que los bloques se observan. A pesar de esto, como los protocolos propuestos trabajan con $\tilde{\mathbb{O}}_v$, y por lo tanto, el tiempo t_b se cancelaría en la definición, por lo que no es necesario expresarlo y el modelo queda más simple para trabajar.

A partir de $\tilde{\mathbb{O}}_v$, los nodos v tienen información con la cual comparar las demoras en tiempo que sus vecinos son capaces de propagar bloques. La forma propuesta de comparar esto en [38] es mediante 3 algoritmos: VanillaScoring, UCBScoring, SubsetScoring, de los cuales, se elige no abordar el segundo, porque añade excesiva complejidad al protocolo y porque SubsetScoring lo excede en las ventajas de sus resultados.

Definición 2.18 (Selección de vecinos, protocolos Perigee) *Dado un conjunto de bloques B propagados, un nodo $v \in V$ junto con sus observaciones (2.17) y parámetros $d_{\text{scored}} \in [d_{out}], \alpha \in (0,1)$, se definen los algoritmos* VanillaScoring, SubsetScoring*, que generan una decisión* KeepNeighbors $\in \{0,1\}^{\Gamma_v^o}$ *sobre que vecinos mantener y descartar.*

- VanillaScoring*: Para cada $u \in \Gamma_v^o$ se genera el arreglo $(t_{uv}^b - t_v^b : b \in B)$ y se calcula un puntaje* $\text{score}(u)$*, definido por el percentil α del arreglo. A partir de* $(\text{score}(u) : u \in \Gamma_v^o)$*, se eligen los primeros d_{scored} vecinos. i.e:*

  - $\forall u \in \Gamma_v^o,\ \text{score}(u) \coloneqq \text{percentil}_\alpha(t_{uv}^b - t_v^b : b \in B)$.
  - $\text{KeepNeighbors} \leftarrow (\text{keep}(u) : u \in \Gamma_v^o)$

  Aquí,

  $$\text{keep}(w) := \begin{cases} 1 & \text{si score}(w) \text{ es de los } d_{\text{scored}} \text{ menores de } (\text{score}(u) : u \in \Gamma_v^o), \\ 0 & \text{si no.} \end{cases}$$

  En la práctica, pueden ocurrir empates entre $\text{score}(u)$*. El caso más prevalente, es cuando más de un nodo u tiene* $\text{score}(u) = +\infty$*, en el cual simplemente se decide el empate de manera arbitraria (En la implementación en código, se prefiere el con menor u como número natural). Para $\alpha = 90\,\%$ es una imposibilidad matemática que más de un u cumpla* $\text{score}(u) = 0$*. El otro caso es que* $\text{score}(u)$ *se repita en un número distinto de $\{0, +\infty\}$ pero esto queda descartado por la suposición 2.10, ya que esto necesita de que existan dos caminos distintos de misma latencia.*

- SubsetScoring*:*

  En este caso, en vez de elegir nodos $u \in \Gamma_v^o$ solo según los puntajes de cada uno por separado, se elige de acuerdo a un criterio que tiende a aprovecharse de aquellos nodos que se complementan entre sí. Primero, de forma análoga a VanillaScoring*, se elige un nodo u^* que minimice* $\text{score}(u)$*, luego, se modifican los arreglos de tiempos de tal forma que el siguiente nodo u^{**} a elegir se complemente con u^*, luego el resto de los nodos se complementa bien con $\{u^*, u^{**}\}$ y así...*

  En forma de algoritmo:

  - *seleccionados* $\leftarrow \phi$.
  - *Mientras* $|\textit{seleccionados}| < d_{\text{scored}}$*:*
    - $\forall u \in \Gamma_v^o - \textit{seleccionados}$,
    - $\textit{temp} \leftarrow (\text{mín}\{t_{uv}^b - t_v, \text{mín}\{t_{xv}^b - t_v^b \ :\ x \in \textit{seleccionados}\}\} \ : b \in B)$
    - *Calcular:*

      $$\text{score}(u) \coloneqq \text{percentil}_\alpha(\ \textit{temp}[b] :\ b \in B) \tag{2.24}$$

    - $u^* \leftarrow \text{argmin}\{score(u) : u \in \Gamma_v^o - seleccionados\}$.
    - $seleccionados \leftarrow seleccionados \cup \{u^*\}$.
    - *repetir.*
  - $KeepNeighbors \leftarrow (1_{u \in seleccionados} : u \in \Gamma_v^o)$.

En ambos algoritmos, el número de nodos seleccionados es d_{scored} *y la complejidad temporal es polinomial c/r a* $|B|$,$|\Gamma_v^o|$.

Nuestro protocolo original EdgePriority se define de manera similar:

Definición 2.19 (Selección de vecinos, protocolo EdgePriority) *Dado un conjunto de bloques* B *propagados, un nodo* $v \in V$ *junto con sus observaciones (2.17) y parámetros* $d_{\text{scored}} \in [d_{out}], \alpha \in (0,1)$*, el algoritmo genera una decisión* $\text{KeepNeighbors} \in \{0,1\}^{\Gamma_v^o}$ *sobre que vecinos mantener y descartar.*

- $\forall u \in \Gamma_v^o$,
$$\text{score}(u) \coloneqq \frac{1}{|B|} \sum_{b \in B} 1_{u \to_b v}. \tag{2.25}$$
- $\text{KeepNeighbors} \leftarrow (\text{keep}(u) : u \in \Gamma_v^o)$

Aquí,

$$\text{keep}(w) := \begin{cases} 1 & \textit{si } \text{score}(w) \textit{ es de los } d_{\text{scored}} \textit{ mayores de } (\text{score}(u) : u \in \Gamma_v^o), \\ 0 & \textit{si no.} \end{cases}$$

Se puede notar que si B *es lo suficientemente grande, y fue muestreado según* $\mathbb{P}_b$ *(2.7), la ley de grandes números establece que,*

$$\text{score}(u) = \frac{1}{|B|} \sum_{b \in B} 1_{u \to_b v} \approx \mathbb{E}_b[1_{u \to_b v}] = \mathbb{P}_b[u \to_b v]. \tag{2.26}$$

2.2. Análisis

En esta sección se analizan varias componentes del modelo. Primero, se examina el comportamiento de la selección aleatoria que se propuso anteriormente, con fin de exponer que propiedades tiene y argumentar que sus características son razonables para coordinar la generación de nuevas conexiones. Después, se examina cómo se comporta la conexidad y el diámetro en una red inicializada al azar. Por razones de tamaño, la aleatoriedad genera cierta regularidad, y es importante entenderla porque sus características se relacionan con la latencia de propagación en la red. Seguido de eso, viene la sección más extensa, en que se dan motivaciones para usar EdgePriority, apoyándose en problemas de optimización y el grafo de propagación T_b. Después de eso, vienen secciones más ligadas a la complejidad computacional de las operaciones. Se discute sobre la complejidad computacional de calcular las métricas exhaustivamente, y de cómo mitigar aquella complejidad usando estimaciones estadísticas. Finalmente, se analiza la complejidad computacional de ejecutar una simulación con el modelo planteado.

El procedimiento realizado para generar el procesamiento de nuevas conexiones aleatorias sin bloqueos no es exactamente cómo funciona la realidad, sin embargo, tiene las siguientes propiedades:

No hay bloqueos: Es sencillo notar el hecho de que solo un nodo tenga permiso para formar una conexión aleatoria en durante cada etapa del procedimiento elimina la posibilidad de los bloqueos mencionados.

Se tiene independencia entre las resoluciones de solicitudes: Debido a que las selecciones aleatorias se realizan de manera independiente, el tiempo para v entre una conexión aleatoria y la siguiente es independiente de sus conexiones anteriores

El número esperado de ciclos de R para que v complete sus solicitudes es:

$$T_{v_r}^{N(R)} = \frac{r_v}{r_v + 1}(N(R) + 1), \quad N(R) = \sum_v r_v. \tag{2.27}$$

Los detalles para derivar la fórmula dependen de formular el fenómeno como una cadena de Markov, y se encuentran en el anexo A.2. Intuitivamente, la estructura de datos R es como una urna con bolitas, en la que v pone r_v bolitas que le pertenece, y $T_{v_r}^N$ es el número esperado de bolitas que hay que extraer de la urna (sin reponer) para que v vuelva a ver las r_v bolitas que ingresó. La fórmula (2.27) permite interpretar que R mezcla bastante las bolitas/solicitudes, puesto que si se pone $v_r = 1$, v tendrá que esperar aproximadamente $(N + 1)/2$, aproximadamente que salga la mitad de las solicitudes en la urna antes de ver salir la suya. Profundizando, para k más grande, $T_k^N \approx (N+1)$, osea que si v ingresan muchas solicitudes, será esperable tener que estar hasta casi el final del procesamiento de solicitudes para que se logren realizar todas las solicitudes propias de v. Lo anterior es simétrico para todos los nodos, y por lo tanto, si por ejemplo, todos ingresan $r_v \approx 10$, todos tendrán que esperar, en promedio, el 90 % del proceso total de las solicitudes en la urna para terminar sus solicitudes.

El método no es una respuesta definitiva, pero tiene propiedades razonables para el propósito que se usa: simetría, independencia, y que todos los nodos tardan una cantidad

comparable de tiempo entre ellos para acabar con sus conexiones.

En la sección de implementación se muestra una estructura de datos concreta para realizar el procedimiento.

2.2.1. Garantías de conexidad y diámetro

Es de vital importancia para una red P2P Blockchain que los mensajes se propaguen por toda la red. Más aun, es importante tener garantías sobre la conexidad del grafo G_{TCP} puesto que facilita analizar otras propiedades de la red. En 2.1.2 se explicó que al inicializar la red, los nodos inician conexiones hacia otros, elegidas aleatoriamente, según el parámetro d^v_{out}. Si se define $k := \text{mín}\{d^v_{out} : v \in V\}$, este procedimiento genera un grafo que contiene al grafo aleatorio $G_{V,k-out}$, y por lo tanto, con alta probabilidad, el grafo es $k-$conexo ([24], Teorema 18.2). Por otro lado, en 2.13 y 2.1.2, se detalla que periódicamente los nodos están eligiendo nuevas conexiones. Si la elección de vecinos es con visibilidad global (2.5), entonces la red estaría formando subgrafos del tipo $G_{V,k-out}$, pero con un parámetro menor ($k = \text{mín}\{k_v : v \in V\},\ k_v = d^v_{out} - d_{\text{scored}}$). En el caso en que la visibilidad es local (2.6), no existe tal resultado. A pesar de eso, es posible intuir que el comportamiento de seleccionar vecinos con buenos tiempos de propagación, no debería perjudicar la conexidad, aunque esto debería ser comprobado. El comportamiento de la conexidad en una red con visibilidad local podría ser estudiado mediante simulaciones y algoritmos para calcular conexidad fuerte en digrafos [13].

Más allá de la conexidad, puede ser interesante conocer sobre el tiempo máximo que tarda un mensaje en propagarse por la red bajo la latencia l, que se llama $\text{diam}_l(G_{\text{TCP}})$:

$$\text{diam}_l(G_{\text{TCP}}) := \text{máx}\{t(u,v) : u, v \in V\},$$

Se puede chequear que $\text{diam}_l(G_{\text{TCP}}) = \text{máx}_{v\in V}\, \lambda^\alpha_v$ si $\alpha = 1$ (bajo cierta condición en $(f_v)_v$). Además, se puede notar que si (l_{uv}) está acotada por $L \in \mathbb{R}^+_0$, entonces,

$$\text{diam}_l(G_{\text{TCP}}) \leq L \cdot \text{diam}(G_{\text{TCP}}).$$

Donde, $\text{diam}(G)$ es el diámetro por saltos en G. Para el modelo $G_{V,k-out}$, existe la garantía de que con alta probablidad, $\text{diam}(G_{V,k-out})$ es asimptóticamente $\log_{2k}(V)$. Esto es bueno para los casos de inicialización por bootstrap, y también 2.5, ya que el diámetro está acotado por el diámetro de los subgrafos ($G_{V,k-out}$ como en el análisis de conexidad).

Para el caso de elección de vecinos por visibilidad local, otra forma de pensar en este problema es comparar con otros grafos de redes aleatorias, como el modelo de Barabási-Albert. En ese modelo, se genera un grafo aleatorio paulatinamente incorporando nuevos nodos; al ingresar un nuevo nodo v, este se conecta a otro nodo u con probabilidad $p_u = d_u / \sum_{w\in V} d_w$ [24] [8]. En este modelo, los nodos con más conexiones aumentan su probabilidad de conseguir aún más vecinos, y esto da a lugar una serie de propiedades interesantes. En particular, se ha demostrado ([**?**] [8]) que ese grafo tiene diámetro asimptóticamente $\log |V|$, y versiones aún más realistas tienen diámetro asimptóticamente $\log |V| / \log\log |V|$, con alta probabilidad.

El análisis anterior sobre el diámetro puede sonar un poco contradictorio con lo observado en la realidad. Si se piensa que la latencia $L = \sup_{uv} l_{uv}$ no es tan elevada en el mundo (sea,

2 segundos) y que las redes son de decenas de miles de nodos, pues entonces, teóricamente cada mensaje se podría propagar en $L \log |V| \approx 10[s]$. Según las mediciones de [17], existen mensajes que los nodos se enteran con diferencias de hasta 50 y 60 segundos. El problema está en que no se asume que exista un protocolo de selección de vecinos que los incentive a propagar los mensajes lo más rápido posible, como se asume en 2.8. Por lo tanto, es posible que muchos mensajes/bloques queden a la espera de ser comunicados, al menos en las redes como Bitcoin. Afortundamente, este fenómeno se puede aproximar con nuestro modelo, mediante la alteración arbitraria de l_{uv}. Por ejemplo, podría elegirse que un 50 % de los nodos usen una latencia de propagación de mensajes con una espera, $\hat{l_{uv}} = l_{uv} + 10[s]$, y los resultados de dicha espera podrían ser medidos mediante experimentos.

Para finalizar esta sección, se menciona que el problema de calcular diámetros es medianamente costoso computacionalmente. Si bien, existen algoritmos $O(|V|^2 \log |V|)$ para calcularlos exactamente en grafos esparsos, esto puede tardar bastante para $|V| = 50,000$.

2.2.2. Motivación Edge Priority

Aquí se entregan motivaciones para usar el protocolo **EdgePriority**, definido en 2.19. Se recuerda que en el protocolo cada v selecciona un subconjunto de vecinos (los vecinos salientes/preferentes Γ_v^o) para ser descartado, en la medida que sean los menores al ser evaluados por el puntaje:

$$\text{score}(u) = \frac{1}{|B|} \sum_{b \in B} 1_{u \to_b v} \approx \mathbb{P}_b[u \to_b v].$$

Este puntaje se evalua durante una ronda en que v observa un conjunto B de bloques propagarse por la red. Si $\text{score}(u) = 0$, es casi seguro que u será descartado en la ronda. Todos las conexiones a nodos descartados se reemplazan en seguida por conexiones a nodos seleccionados aleatoriamente.

Problema de optimización individual

Primero, suponer que la red está dada por un grafo G no dirigido, y que $v \notin V$. Considerar la situación en que un nodo v desea conectarse a la red con a lo más p vecinos, y minimizar la latencia promedio en que tarda un bloque generado en llegar a v a través de ellos:

$$P_v) \qquad \min_{U \subseteq V:\ |U| \leq p} \mathbb{E}_b[\min_{u \in U}\ t_G(b,u) + l_{uv}]. \tag{2.28}$$

En este contexto, se denota $u \to_b v$ si u es el minimizador de la expresión dentro de $\mathbb{E}_b$ para v. El problema de optimización 2.28 se asemeja a un par de problemas de optimización combinatoriales en que se desea seleccionar una cantidad de centros que minimicen cierto costo, como lo son el *k-center-problem* y el *p-median problem*, ambos NP-difíciles con respecto del tamaño de $|V|$ para $p > 3$ [14], [11]. Algo que diferencia el problema 2.28 con otros problemas clásicos, es que desde el nodo v no se puede suponer conocimiento a priori del

grafo G, pues para los protocolos, no es posible conocer toda la red. Frente a esto, solo queda la esperanza de utilizar un algoritmo heurístico.

El protocolo EdgePriority, al igual que Perigee, explora soluciones al problema de elegir vecinos mediante la eleccion rutinaria de vecinos aleatorios, y se mantiene explotando un subconjunto de los vecinos que tiene buenas características, pero a diferencia de Perigee, EdgePriority determina la utilidad de u para v en torno a la cantidad $\mathbb{P}_b(u \to_b v)$, la probabilidad de que b propague un bloque más rápidamente que el resto de los vecinos a v.

Claramente, si dado un conjunto U de vecinos de v se cumple que $\mathbb{P}_b[w \to_b v] = 0$, entonces u no tiene utilidad alguna para v, y este puede ser deshechado, abriendo espacio para más vecinos para v. De hecho, si en un momento ocurre que $\mathbb{P}_b[w \to_b v] = 0$ y v decide cambiar w por otro vecino u^+, para el que u^+ obtiene $\mathbb{P}_b[u^+ \to_b v] > 0$ con respecto del conjunto de vecinos $U - w + u^+$, entonces es claro que este nuevo vecino u^+ disminuye la latencia promedio $\mathbb{E}_b[t(b,v)]$. Esto se comprueba sencillamente: primero, notar que $\mathbb{E}_b[\min_{u\in U} t(b,u) + l_{uv}] = \mathbb{E}_b[\min_{u\in U-w} t(b,u) + l_{uv}]$. Se denota $U_1 = U - w$ y se desarrolla con esperanza condicional,

$$\mathbb{E}_b[\min_{u\in U_1+u^+} t(b,u) + l_{uv}] \tag{2.29}$$
$$= \mathbb{E}_b[\quad t(b,u^+) + l_{u^+v} \quad | u^+ \to_b v]\mathbb{P}_b[u^+ \to_b v] \tag{2.30}$$
$$+ \mathbb{E}_b[\min_{u\in U_1-u^+} t(b,u) + l_{uv} | u^+ \not\to_b v]\mathbb{P}_b[u^+ \not\to b] \tag{2.31}$$
$$< \mathbb{E}_b[\min_{u\in U_1-u^+} t(b,u) + l_{uv} | u^+ \to_b v]\mathbb{P}_b[u^+ \to_b v] \tag{2.32}$$
$$+ \mathbb{E}_b[\min_{u\in U_1-u^+} t(b,u) + l_{uv} | u^+ \not\to_b v]\mathbb{P}_b[u^+ \not\to b] \tag{2.33}$$
$$= \mathbb{E}_b[\min_{u\in U_1-u^+} t(b,u) + l_{uv}] \tag{2.34}$$
$$= \mathbb{E}_b[\min_{u\in U-u^+} t(b,u) + l_{uv}] \tag{2.35}$$
$$= \mathbb{E}_b[\min_{u\in U} t(b,u) + l_{uv}] \tag{2.36}$$
$$\tag{2.37}$$

En la desigualdad estricta se usa la definición de $u^+ \to_b v$ dentro de la primera esperanza; la igualdad siguiente se tiene por revertir la esperanza condicional; la siguiente por el hecho básico de conjuntos $U - w - u^+ = U - w$; la última es porque w no contribuye a la esperanza.

Aterrizando lo anterior al protocolo, se sabe con seguridad que si $\text{score}(u) \approx \mathbb{P}_b[u \to b] = 0$, entonces cambiar a u por un vecino nuevo solo puede mejorar la latencia esperada. Eso sí, en general no se tendrá que $\mathbb{P}_b[u \to b]$ sea exactamente cero para algún vecino de v, pero de todas maneras descartar los nodos con las menores probabilidades de informar un bloque es una decisión de bajo riesgo, y razonable en el contexto de una heurística de exploración.

Problema de optimización global

Otra motivación para el algoritmo se encuentra en una visión más global: Cuando toda la red busca optimizar la topología, se está buscando encontrar qué conexiones uv son importantes para la transmisión oportuna de b. En este sentido, el problema a nivel global se puede

plantear como la optimización en el conjunto de aristas $F \subset \binom{V}{2}$ de un grafo $G_F = (V, F)$ de una función, que por ahora se deja explicitada con un ligero abuso de notación: $\mathbb{E}_b[f(G)|F]$. Se llama a esta función la métrica global del grafo G según las aristas F usadas. Así, el problema global es:

$$P_V) \qquad \min_{F \subseteq \binom{V}{2} : F \in \mathcal{F}} \mathbb{E}_b[\max_{v \in V} t_{G_F}(b, v)]. \tag{2.38}$$

$\mathcal{F}$ es simplemente un conjunto que especifica restricciones para las aristas F. $\mathcal{F} \subset \{F : F \text{ conecta } V\}$.

El espíritu de la métrica global $\mathbb{E}_b[f(G)|F]$ es que capture alguna noción que relacione la forma del grafo $G_F = (V, F)$ con la generación de bloques $\mathbb{E}_b$, dando mas importancia a bloques con mayor probabilidad de origen y por cierto, que la métrica se *mejore* si las latencias disminuyen. Varias funciones satisfacen estos criterios, aquí se listan algunas:

$$\mathbb{E}_b[f_1(G)|F] = \mathbb{E}_b \left[\sum_{uv \in F} 1^{G_F}_{u \to_b v} \; l_{uv} \right] \tag{2.39}$$

$$\mathbb{E}_b[f_2(G)|F] = \mathbb{E}_b \left[\max_{v \in V} t_{G_F}(b, v) \right] \tag{2.40}$$

$$\mathbb{E}_b[f_3(G)|F] = \mathbb{E}_b \left[\sum_{v \in V} t_{G_F}(b, v) \right] \tag{2.41}$$

En este contexto, se puede volver a hablar del paradigma heurístico y la exploración-explotación: Si se supone que no se conoce suficiente de las estructuras del problema *a priori*, es decir, no se sabe exactamente f_v ni l_{uv} si no es por haber hecho una exploración anteriormente; la única forma de encontrar un F razonable es mediante la exploración de nuevas aristas y la conservación de aquellas aristas útiles.

Aquí, EdgePriority se relaciona con las métricas global 2.39 ya que si se toma una arista uv que en una configuración F cumple $P_b[u \to_b v] = 0$, entonces es claro que la arista no contribuye a la métrica:

$$\mathbb{E}_b[f(G)|F - uv] = \mathbb{E}_b[f(G)|F] \tag{2.42}$$

Para $f = f_1$ es inmediato de la definición, y para $f \in \{f_2, f_3\}$ basta recordar que $u \to_b v$ siempre y cuando uv participa del grafo T_b. Por otro lado, si al reemplazar la arista uv por otra arista xy (formar $F' = F - uv + xy$), se obtiene $P_b[x \to_b y] > 0$, entonces puede haber mejoras de la métrica, en el caso $f = f_3$, existe al menos un $b \in V$ donde $xy \in T_b$; se puede condicionar en ese b, y se sabe que si un nodo es descendiente de $\text{mín}_{T_b}(x, y)$ en T_b, debe haber mejorado su tiempo en F' con respecto a F, y por eso necesariamente mejora f_3.

Consistencia entre EdgePriority **y la red**

Ahora bien, el protocolo EdgePriority no es un algoritmo centralizado que elimina una arista a la vez, sino que es un proceso distribuído en múltiples nodos que repetidas veces eliminan y agregan aristas de F, de manera no sincronizada, por lo que puede existir la duda si el criterio es razonable más allá de las acciones individuales de cada nodo. Si bien la duda no puede ser despejada completamente, sí se puede argumentar que existe una consistencia entre la información que observa un nodo y su entorno. Las observación de la información en EdgePriority está pauteada principalmente por una visión localizada de T_b, la cuál es bastante reducida: cada nodo solo puede conocer quién es su predecesor. Sin embargo, es información confiable en el sentido de que a un nodo v no se puede mentir sobre quién propagó un bloque antes, pues es una información que se determina en el sustento material no falsificable de que un nodo u propagó la información b antes que cualquier otro. Aparte de no ser falsificable, habla de la importancia de u para v en el grafo, ya que eso significa que u muestra un mejor esfuerzo que el resto de los vecinos para comunicar b: puede ser el caso, en una red real, que otro vecino pudiese propagar b más rápido a v, pero si esto no se dió, es porque no hizo suficiente esfuerzo para ello. Esto, en nuestro modelo simplemente se plasma como que $t(b,u) + l_{uv}$ fue minimizado por u. Lo anterior se puede extender más allá, cuando se considera que para que $u \to_b v$, la arista uv es solo una en el camino de primera propagación $b\overrightarrow{P}v$, luego, mantener la arista uv asegura el mejor camino desde b a v en el grafo actual; si se extrapola a toda la estructura T_b, se puede intuir que cuando todos los nodos se ajustan al protocolo, cada arista de T_b gana un voto para ser mantenida en el futuro. Al menos, esa puede ser una justificación para la consistencia.

En contra de lo escrito en el párrafo anterior, existe una dificultad y es que puede ser que las aristas F de la red cambien demasiado rápido para que, desde un nodo v la evaluación de un nodo u, score(u) capture bien la eficacia *en tiempo presente*, pues a medida que cambia F por descarte y añadido de nuevas aristas, pueden cambiar las rutas óptimas, y esta dificultad se puede agravar aún más si se considera que en una red real no solo hay protocolos de seleccion de vecinos, sino que hay nodos que entran y salen aleatoriamente por el fenómeno de churning. Este no es solo un problema de EdgePriority sino que algo que debería intentar atacarse por cualqueir protocolo de selección de vecinos en la medida que exista variación en la red. Hay literatura que respalda que el churning deteriora la calidad de la red [29], [42] y que es un fenómeno que ocurre continuamente, generando la salida y la entrada de un porcentaje de la red cada día. Por otro lado, en el estado actual de redes blockchain como Bitcoin, el transcurso de una ronda de $|B| = 100$ bloques puede tardar un día entero, por lo que es necesario alguna forma de seleccionar qué instante de información de la ronda es más relevante a la hora de generar decisiones de selección de vecinos. Gracias a la forma *lineal* que adquiere score(u), podría pensarse en usar ponderadores sobre los puntajes:

$$\text{score}_\mu(u) = \sum_{b \in B} \mu_b \cdot 1_{u \to_b v},$$

para $\mu = (\mu_b)_b$ un ponderador que asigna diferentes pesos a distintos instantes. Si $(\mu_b)_b$ suma 1, entonces $\mathbb{E}_b[\text{score}_\mu(u)]$ sigue siendo un estimador de $\mathbb{P}_b[u \to_b v]$. En la práctica, μ_b podría bien darle más importancia a los últimos bloques observados, o bien, al pasado, eso queda como una pregunta de investigación.

Consideraciones adicionales sobre las métricas de latencia

Más atras en el capítulo, en (2.13) y (2.18) se pudo vislumbrar un poco de la importancia del puntaje $\mathbb{P}_b[u \to_b v]$, a continuación se subraya que a partir de la probabilidad, se puede ver a S_v como una buena métrica egoísta, mediante el cálculo del tiempo promedio en que los bloques se propagan hacia v:

$$\mathbb{E}_b[t(b,v)] = \mathbb{E}_b\left[\sum_{u \in N(v) \ \vee \ u=v} 1_{u \to_b v} \ (t(b,u) + l(u,v))\right] \tag{2.43}$$

$$= \sum_{u \in N(v) \ \vee \ u=v} \mathbb{E}_b\left[1_{u \to_b v} \ t(b,u)\right] + \sum_{u \in N(v) \ \vee \ u=v} l(u,v)\mathbb{P}_b[u \to_b v] \tag{2.44}$$

$$= \sum_{u \in N(v) \ \vee \ u=v} \mathbb{E}_b\left[t(b,u) | u \to_b v\right] \mathbb{P}_b[u \to_b v] + \sum_{u \in N(v) \ \vee \ u=v} l(u,v)\mathbb{P}_b[u \to_b v] \tag{2.45}$$

$$= S_v + \sum_{u \in N(v)} \mathbb{E}_b\left[t(b,u) | u \to_b v\right] \mathbb{P}_b[u \to_b v] \tag{2.46}$$

Es decir, se puede descomponer el tiempo promedio en S_v más el tiempo promedio que el bloque llega a los vecinos. Nuevamente, S_v contabiliza (ponderadamente) la latencia exceso entre que un bloque b llega a un vecino u y u lo entrega a v; en virtud de la descomposición de la ecuación, se tiene una noción recursiva: si $\sum_v S_v$ es pequeño en general, entonces un bloque b tarda poco tiempo en salir de b y llegar a $N(b)$; y si $\sum_v S_v$ es pequeño, entonces esto también debería significar que $N(b)$ tarda poco en llegar a $N(N(b))$ y así sucesivamente.

Por otro lado, desde el punto de vista de métrica individual, la descomposición es importante porque S_v puede ser controlado hasta cierto nivel pues v tiene la potestad de descartar vecinos (por ejemplo si tiene l_{uv} elevado), pero $\mathbb{E}_b[t(b,u)|u \to_b v]$ es una cantidad que está fuera del control (directo) de v, pues depende de la topología del resto de la red y la producción de bloques. No obstante, es importante tomar en consideración $\mathbb{E}_b[t(b,u)|u \to_b v]$ cuando $\mathbb{P}_b[u \to_b v]$ sea proporcionalmente grande, y lamentablemente, eso es algo que EdgePriority no contempla. Un desafío pendiente es incorporar más información de las observaciones para un protocolo.

Lamentablemente, se puede leer en todo toda esta sección que los análisis realizados no son tan afirmativos como uno esperaría: hay que someterse a casos bordes para obtener resultados, que muchas veces generan la sensación de no son ser suficientemente fuertes, y todo esto a pesar de que las herramientas usadas para definir el problema son bastante básicas. Vale la pena mencionar que un interés por definir el protocolo fue de tratar de aterrizar el complicado funcionamiento de SubsetScoring (percentiles, observaciones modificadas, subconjuntos, etc...) a estructuras más fundamentales que permitieran entender mejor el problema. Lamentablemente, los resultados de usar $\mathbb{P}_b[u \to_b v]$ parecen no ser suficientes, pero al menos dan un marco para generar intuición e interpretación de los fenómenos de la red. Con esto se cierra el análisis de EdgePriority.

2.2.3. Cálculo de métricas

Para medir el rendimiento en los experimentos que se realicen con el modelo, es necesario realizar cálculos de las métricas. Un problema que tiene tanto el cálculo de λ_v^α como $\mathbb{E}_b[l(T_b)]$ es que el cómputo global (es decir para cada $v \in V$) puede tomar una cantidad de tiempo significativa. Debido a que para ambas métricas se debe ejecutar el algoritmo de Dikjstra $|V|$ veces, que para grafos esparsos, cada ejecución tiene un costo de $\Theta(\Delta \cdot |V| \log_2 |V|)$Si se desea realizar esto para una simulación con $|V| = 50,000, \Delta = 100$, se puede hacer una estimación gruesa de la cantidad de cálculos realizados por una computadora:

$$10^2 \cdot |V|^2 \cdot \log_2 |V| = 10^2 \cdot 5^2 \cdot 10^{4\times 2} \cdot (4 + \log_2(5)) \approx 1,58 \cdot 10^{12}.$$

Para una computadora con un procesador que ejecuta alrededor de $3 \cdot 10^9$ instrucciones por segundo, se puede estimar que la cantidad de trabajo debería procesarse en alrededor de

$$\frac{1,58}{3} 10^{12-9} \ [s] \approx \frac{1}{2} \cdot 10^3 \ [s] \approx 0,14 \ [\text{horas}].$$

Si bien, esta es solo estimación gruesa, empiezan a mostrar el orden de tiempo que tarda calcular métricas en un computador real. En la práctica, para un computador personal con procesador `intel i7-7500U`, de frecuencia 3[GHz] y 4[MB] de memoria caché, estos cálculos tardan aproximadamente una hora.

Lo anterior motiva no calcular las métricas exhaustivamente, si no que usar algún método estadístico, pues en la práctica, las métricas suelen concentrarse. A eso se dedican las siguientes subsecciones.

Se debe mencionar que al momento de redactar este texto, se encontró que el tiempo de cálculo se puede reducir usando montículos de fibonacci en el algoritmo de Dikjstra, pero de todas maneras, el algoritmo es a lo menos $\Omega(|V|^2)$. Si bien el método con montículos de fibonacci puede reducir el costo Δ veces, el orden cuadrático de la simulación seguiría siendo la principal problemática, siendo necesario de todas maneras realizar muestreo estadístico para redes con $|V| \geq 10^5$.

Métricas de latencia esperada

La forma fuerza bruta de calcular $\mathbb{E}_b[l(T_b)]$ se puede realizar mediante la ecuación explícita:

$$\mathbb{E}_b[l(T_b)] = \sum_{b \in V} f_b \sum_{uv \in E(T_b)} l_{uv}, \tag{2.47}$$

pero como se mencionó, esto implica calcular T_b para todo $b \in V$, un cálculo muy costoso. La otra forma sencilla es simplemente hacer un muestreo aleatorio de b usando la ley f_b, generando una muestra aleatoria de bloques iid $B = \{b_1, \ldots, b_n\}$, y entonces,

$$\mathbb{E}_b[l(T_b)] = \mathbb{E}\left[\frac{1}{|B|} \sum_{b \in B} \sum_{uv \in E(T_b)} l_{uv}\right], \tag{2.48}$$

es decir, mediante el método montecarlo clásico. Como se usa el método clásico, la desviación estándar de la estimación será $\sigma/\sqrt{|B|}$, donde σ es la varianza de la variable aleatoria X:

$$X \ : \ \omega \longrightarrow \sum_{uv \in E(T_{b(\omega)})} l_{uv}. \tag{2.49}$$

De esta manera, el costo computacional de calcular será $|B|$ veces computar T_b mediante el algoritmo de Dikjstra, más el cálculo del promedio. En total, eso es $O(|B|\Delta|V|\log_2|V|)$. En los experimentos del capítulo siguiente se determina el tamaño de $|B|$ para que el error de esta desviación sea negligible.

2.2.4. Cálculo de métricas de percentiles

En esta parte se muestran algunas propiedades de la métrica λ_v^α y se discuten métodos estadísticos para su estimación.

Propiedades de la métrica

Proposición 2.20 *Sea $v \in V$, y se asume que la red es conexa.*

$$\lambda_v^\alpha \leq \operatorname{diam}(G_{\mathrm{TCP}}). \tag{2.50}$$

Demostración. Es directo, ya que la red es conexa, entonces todo nodo u de V tiene latencia a lo más $t(v,u) \leq \operatorname{diam}(G_{\mathrm{TCP}}) < \infty$, y luego $\sum_{u\in V} f_u = 1 \geq \alpha$. □

Proposición 2.21 *Consider que $u, v \in V$, y que $u \in N(v)$, entonces:*

$$\lambda_u^\alpha \leq \lambda_v^\alpha + l_{uv}. \tag{2.51}$$

Demostración. Si se usa que $t(u,w) \leq t(v,w) + l_{uv}$, es evidente que:

$$\{w \in V : \ t(v,w) \leq \lambda_v^\alpha\} \subseteq \{w \in V : t(u,w) \leq \lambda_v^\alpha + l_{uv}\}. \tag{2.52}$$

Luego, al sumar $0 \leq f_w$ sobre estos conjuntos, se tiene que la suma supera α:

$$\alpha \underset{(2,10)}{\leq} \sum_{\substack{w\in V:\\ t(v,w)\leq\lambda_v^\alpha}} f_w \leq \sum_{\substack{w\in V:\\ t(u,w)\leq\lambda_v^\alpha + l_{uv}}} f_w. \tag{2.53}$$

Como λ_u^α se define como la mínima que lleva a que la suma supere a α, $\lambda_u^\alpha \leq \lambda_v^\alpha + l_{uv}$. □

Observación La desigualdad de 2.21 no se tiene, en general, con igualdad (como suele ser con distancias de grafos). Sin embargo, sirve para notar que la métrica deberia exhibir variaciones ligeras entre vecinos, similar a una continuidad de tipo *Lipschitz*.

Corolario 2.22

$$|\lambda_u^\alpha - \lambda_v^\alpha| \leq \max\{l_{uv}, l_{vu}\} \qquad \forall v \in V, \forall u \in N(v). \tag{2.54}$$

Corolario 2.23

$$|\lambda_u^\alpha - \lambda_v^\alpha| \leq \text{máx}\{t(u,v), t(v,u)\} \qquad \forall u, v \in V \tag{2.55}$$

También, para el caso en que l_{uv} cumpla desigualdad triangular, es posible obtener una cota inferior sencilla:

Proposición 2.24 *Si l cumple siempre $l_{xy} + l_{yz} \leq l_{xz}$ y L_v^α se define según:*

$$L_v^\alpha := \text{mín}\left\{ t \geq 0 : \sum_{u \in V \,:\, l_{vu} \leq t} f_u \geq \alpha \right\}, \tag{2.56}$$

entonces,

$$L_v^\alpha \leq \lambda_v^\alpha. \tag{2.57}$$

Observación Se nota que la diferencia en la definición entre L_v^α (2.56) y λ_v^α (2.10) reside en que L se toma considerando a todos los vértices que están a *latencia par a par* l_{vu} menor a t, en vez de latencia de propagación $t(v,u) \leq t$. L representa simplemente el poder de hash que podría ser alcanzado si v estuviese conectado a todos los otros nodos.

Demostración. Se observa que si l cumple la desigualdad triangular, entonces, se vuelve evidente que $l_{vu} \leq t(v,u)$. Luego, si $t = \lambda_v^\alpha$, se obtiene:

$$\sum_{u \in V : l_{vu} \leq t} f_u \geq \sum_{u \in V : t(v,u) \leq t} f_u \geq \alpha,$$

y por lo tanto, $\lambda_v^\alpha \geq L_v^\alpha$. □

Esta cota no estaba propuesta en el artículo [38], pues en tamaños como $|V| = 1,000$ no hay mayor problema en calcular todo el arreglo $(t(u,v) \;:\; u, v \in V)$ para el grafo completo $G = (V, \binom{V}{2})$, pero esto no tan es cierto para $|V| = 50,000$, ya que la complejidad de los algoritmos comienza a necesitar de largos tiempos de computación.

Métodos estadísticos

Recordando que las formas de las curvas CDF entregan una perspectiva cuantitativa y cualitativa (en las figuras de ejemplo 2.2 y 2.3), sería interesante desarrollar algún método estadístico para no tener que hacer el costoso cálculo completo de las métricas, pero sí conservar la forma y las cantidades que estas curvas poseen.

Si se pone una medida de probabilidad uniforme en V y se muestrea aleatoriamente vértices iid $\{v_1, v_2, \ldots, v_n\}$, el teorema de Glivenko-Cantelli permite asegurar que la distribución empírica de muestrear $\lambda_{v_i}^\alpha$ converge a la distribución real:

Si

$$F_n(t) := \frac{1}{n}\sum_{i=1}^{n} 1_{\lambda_{v_i}^{\alpha} \leq t}, \tag{2.58}$$

$$F(t) := \frac{1}{|V|}\sum_{v \in V} 1_{\lambda_{v}^{\alpha} \leq t}, \tag{2.59}$$

entonces,

$$||F_n - F||_\infty = \sup_{x \in \mathbb{R}} |F_n(t) - F(t)| \longrightarrow 0 \quad \text{casi seguramente.} \tag{2.60}$$

El teorema de Glivenko-Cantelli da la certeza de que si se usan suficientes muestras, se obtiene algo razonablemente cercano a λ^α, sin embargo, no es inmediato los efectos cuantitativos dado que solo asegura convergencia. Por otro lado, en los experimentos puede ser importante dimensionar si la distribución de λ^α cambió durante un periodo de rondas. Si bien existen pruebas estadísticas como la de Kolmogorov-Smirnoff, la prueba se mide en diferencias de $F_n(t) - F(t)$ pero las variaciones que se quieren dimensionar se "miden en t". Por todo lo anterior, se propone un método ajustado al caso, a partir de la desigualdad de Azuma-Hoeffding. Primero se parte con describir un método general que luego se aterrizará: Considerar muestras aleatorias $X = (X_1, X_2, \ldots, X_n), Y = (Y_1, Y_2, \ldots, Y_n)$ ambas iid, en que cada muestra X_i tiene esperanza μ_X e Y_i tiene esperanza μ_Y, y X_i están acotadas por debajo y arriba por X_-, X_+, similarmente con Y. Se define $\mu = \mu_Y - \mu_X$ y se toma μ_σ real. Se supone que se busca tener seguridad de que μ_X está a cierta distancia de μ_Y, para ello, se toma el promedio empírico. Si se observa una distancia μ_σ entre los promedios empíricos, interesa saber qué tan probable es que no se esté teniendo una visión clara de μ, sino una versión distorsionada μ_σ debido a efectos de la aleatoriedad. Más aún, interesa determinar cuántas mediciones $n = n(\varepsilon)$ se deben realizar para acotar la siguiente probabilidad:

$$\mathbb{P}\left[\frac{1}{n}\sum_{i=1}^{n} Y_i - \frac{1}{n}\sum_{i=1}^{n} X_i \geq \mu_\sigma\right] \leq \varepsilon. \tag{2.61}$$

Si se logra calcular $n(\varepsilon)$, se puede saber hasta qué punto es necesario tomar muestras para tener suficiente seguridad de que lo que se está observando no es fortuito, sino que es el comportamiento subyacente. Para ello, se observa que el evento cuya probabilidad se intenta acotar es:

$$\sum_{i=1}^{n} Y_i - X_i \geq n\mu_\sigma, \tag{2.62}$$

Al desarrollar, restando $n\mu := n(\mu_Y - \mu_X)$ en ambos lados,

$$\sum_{i=1}^{n} (Y_i - X_i) - n\mu \geq n(\mu_\sigma - \mu). \tag{2.63}$$

Si se define Z_n como el lado izquierdo,

$$Z_n \coloneqq \sum_{i=1}^{n} (Y_i - X_i) - n\mu,$$

se tiene que Z_n es una *martingala*. Es directo notar que $\mathbb{E}[Z_n] = E[Z_0] = 0$, y que además, Z_n es una martingala de incrementos acotados:

$$|Z_n - Z_{n-1}| \leq |X_n - X_{n-1}| + |Y_n - Y_{n-1}| + |\mu| \leq |X_+ - X_-| + |Y_+ - Y_-| + |\mu|.$$

Gracias a esto, se puede usar la desigualdad de Azuma-Hoeffding para acotar la probabilidad del evento (2.63), y así:

$$\mathbb{P}\left[\frac{1}{n}\sum_{i=1}^{n} Y_i - \frac{1}{n}\sum_{i=1}^{n} X_i \geq \mu_\sigma\right] \tag{2.64}$$

$$= \mathbb{P}[Z_n \geq n(\mu_\sigma - \mu)] \tag{2.65}$$

$$\leq \exp\left(\frac{-n(\mu_\sigma - \mu)^2}{2(|X_+ - X_-| + |Y_+ - Y_-| + |\mu|)^2}\right). \tag{2.66}$$

Es decir, para acotar por ε, basta con:

$$n(\varepsilon) = 2\ln\left(\frac{1}{\varepsilon}\right)\frac{(|X_+ - X_-| + |Y_+ - Y_-| + |\mu|)^2}{(\mu_\sigma - \mu)^2}. \tag{2.67}$$

A partir de esto, se puede derivar otra cota en probabilidad: Sea $\delta > 0$, entonces la probabilidad:

$$\mathbb{P}\left[\left|\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu_X + \left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) + \mu_Y\right| \geq \delta\right] \leq \varepsilon_1, \tag{2.68}$$

para ε_1 :

$$\varepsilon_1 = 2\exp\left(\frac{-n\delta^2}{2(|X_+ - X_-| + |Y_+ - Y_-| + |\mu|)^2}\right), \tag{2.69}$$

y en consecuencia, con $n = n(\varepsilon_1)$

$$n(\varepsilon_1) = 2\ln\left(\frac{2}{\varepsilon_1}\right)\frac{(|X_+ - X_-| + |Y_+ - Y_-| + |\mu|)^2}{\delta^2}, \tag{2.70}$$

basta para tener la cota (2.68). La prueba de ello se realiza descomponiendo la desigualdad en valor absoluto en la unión de dos desigualdades $\sum_i Y_i - \sum_i X_i - n\mu \geq \pm n\delta$, de donde aparece nuevamente (2.63) con $\mu_\sigma = \mu \pm \delta$, luego basta aplicar la desigualdad y unas cotas básicas de probabilidades.

Ahora se aterriza esto a la estimación de las formas de las CDF. La estimación más gruesa que se puede hacer es determinar la tendencia de la curva X al ser generada para una incialización aleatoria. Para esto, fijar $Y = 0$, $X_i = \lambda^\alpha_{v_i}$. Será generalmente cierto que:

$$\text{máx}\{|X_+|, |X_-|, |\mu_X|\} \leq \max_{x \in V} \lambda^\alpha_x \leq \text{diam}(G_{\text{TCP}}) = O(\log_{2k}|V|).$$

La consideración de orden ocurre porque al ser X de una red aleatoria de características $G_{k,out}$, cumple con las garantías de conexidad, con alta probabilidad. A partir de esto, se usa la ecuación 2.70 y se sabe que:

$$n(\varepsilon_1) = 2\ln\left(\frac{2}{\varepsilon_1}\right)\frac{A\log_{2k}|V|}{\delta^2},$$

para asegurar

$$\left| \frac{1}{n} \sum_{i=1}^{n} X_i - \mu_X \right| \leq \delta,$$

con error de falso positivo menor a ε_1. Esto permite saber que asimptóticamente en el tamaño de la red $|V|$ no serán necesarias muchas muestras, al menos en un comienzo, para determinar cantidades de tendencia en la red. Por otro lado, si la aplicación de los protocolos sobre la red tiende a disminuir X, es razonable usar el mismo $n(\varepsilon_1)$ para chequear esas propiedades sobre las redes optimizadas. Para la curva de 2.2,

$$n(\varepsilon_1) \approx 3 \ln \left(\frac{2}{\varepsilon_1} \right) \frac{1}{\delta^2},$$

Para $\varepsilon = 1/100$, $n \approx 15/\delta^2$, con δ medido en segundos, si se busca descartar que el promedio se desvia por $0,1$ segundos, esto necesita 1,500 muestras, osea el $15\,\%$ del total de $|V|$.

Se hace énfasis en que para $|V| = 10{,}000$ estos resultados no son tan fuertes, pero sí significan un alivio para $|V|$ superiores, como $|V| = 50{,}000$: si hace basta $n(\varepsilon) = O(\log(\varepsilon^{-1}) \log |V|/\delta^2)$ muestras, entonces el costo las muestras para examinar estadísticamente la red solo requiere de una complejidad en tiempo de,

$$O(|V| \log |V|),$$

en vez de $O(|V|^2)$.

Además, el resultado 2.70, poniendo $n(\varepsilon_1)$ en términos de los anchos de rango $X_+ - X_-, Y_+ - Y_-$ y la separación $|\mu| = |\mu_Y - \mu_X|$ permiten aplicar esto no solo para una muestra de $(\lambda^{\alpha}_{v_i})_i$, sino que para los percentiles de la muestra.

Si se pone que p^x_-, p^x_+ son dos percentiles (inferior y superior) de los que se busca observar X, entonces la muestra censurada $\hat{X} = X_i$ si $p_- \leq X_i \leq p^x_+$ puede ser usada en la cota. Definiendo igualmente $p^y_-, p^y_+, \hat{Y}$, y también definiendo los promedios en los percentiles $\mu_{\hat{X}}, \mu_{\hat{Y}}$ como los promedios en las muestras censuradas, el valor para $n(\varepsilon_1)$ queda como:

$$n(\varepsilon_1) = 2 \ln \left(\frac{2}{\varepsilon_1} \right) \frac{(p^x_+ - p^x_- + p^y_+ - p^y_- + |\mu_{\hat{Y}} - \mu_{\hat{X}}|)^2}{\delta^2}, \tag{2.71}$$

Para la figura 2.3 se tienen los siguientes valores para la muestra censurada entre el percentil 0 y el percentil 50:

$$p^x_- = 3109, p^x_+ = 3435, p^y_- = 2943, p^y_+ = 3609, \mu_{\hat{X}} = 3335, \mu_{\hat{Y}} = 3529.$$

Luego,

$$\begin{aligned} n(\varepsilon_1) &\approx 2 \ln \left(\frac{2}{\varepsilon_1} \right) \left(\frac{3435 - 3109 + 3609 - 2943 + 3529 - 3335}{\mu_\sigma - (3529 - 3335)} \right)^2 \\ &\approx 2 \ln \left(\frac{2}{\varepsilon_1} \right) \left(\frac{1186}{\mu_\sigma - 194} \right)^2 \end{aligned}$$

Entonces, la cantidad de muestras para ue erróneamente observar $\mu_\sigma - 194 = 194$ tenga probabilidad $\varepsilon_1 = \frac{1}{100}$, sería:

$$n(\varepsilon_1) = 2\ln(200) \cdot (1186/194)^2 \approx 396.$$

Ahora bien, esta es una muestra censurada (se descarta la mitad de la muestra original al restringir la muestra), por lo que se necesita al menos tomar una muestra de $n \approx 800$. De todos modos, se necesitan pocas muestras para realizar esto, y aunque aparentemente se está probando para un error de estimación $\mu_\sigma - \mu$ que puede parecer muy grueso, las diferencias que inducen los protocolos suelen ser bastante más grandes.

A pesar de lo anterior, teniendo la experiencia de realizar varias simulaciones, se siente que el método propuesto usando la desigualdad de Azuma-Hoeffding, es muy débil, y que en la práctica se necesita una cantidad menor de muestras para que sea evidente la estabilización de las tendencias. Aún así, es un método correcto, y podría usarse robustamente para hacer experimentos a gran escala, o realizar gran número de experimentos y automáticamente detener las simulaciones cuando ya no sea necesario seguir ejecutando los protocolos.

Esto cierra esta sección de las estimaciones.

2.2.5. Complejidad computacional de la simulación

Realizar simulaciones en esta escala es costoso en tiempo de cómputo, debido a que se necesitan realizar cálculos que involucran a todo el grafo de la red de una sola vez, y se debe repetir estos cálculos muchas veces. En la sección 2.2.3 se realizó una estimación que genera una idea de cuánto puede llegar a tomar esto en un computador. Aquí se calcula la complejidad en tiempo para los pasos más importantes de la simulación.

Un ciclo en la simulación corresponde a la propagación de un bloque b como en 2.13, la simulación consiste de varios ciclos en que los nodos observan la propagación de la información de nuevos bloques; tras la generación de suficientes bloques, algunos nodos ejecutan sus protocolos de selección y generan solicitudes para nuevos vecinos. Algunos costos se deben amortizar porque no se realizan en cada ciclo de la simulación. Por ello, se denota $|B|$ el número de ciclos/bloques de la simulación.

Generación del bloque

En primer lugar, se genera un bloque b como una muestra de la variable aleatoria f_v. Asumiendo que f_v se escribe como datos en un arreglo $(f_v \ : \ v \in V)$, existe un algoritmo que permite muestrear un nodo $v \in V$ en $O(\log |V|)$ usando "el método de la transformada inversa" detallado en el anexo A.3.

Cabe destacar que este método requiere un pre-cómputo que tarda $O(|V| \log |V|)$ que involucra ordenar la información a partir de $(f_v)_v$. Para el caso en que no hay "churn", la ley de poder de hash nunca cambia, y por lo tanto el pre-cómputo solo se realiza una vez.

Sin embargo, en simulaciones con churn, la ley está constantemente cambiando, por lo que cada muestra aleatoria de v puede necesitar de un pre-cómputo, elevando el costo temporal a $O(|V| \log |V|)$.

Propagación de información

Tras lo anterior, de acuerdo a las definiciones de la sección 2.1.4 se calcula T_b y los tiempos de observación para los nodos de acuerdo al algoritmo de Dikjstra, tomando una complejidad en tiempo de $O(|E| \log |V|) = O(\Delta |V| \log |V|)$.

Despues, cada nodo, de manera independiente, realiza sus observaciones, que consiste en almacenar algunos de los datos, enumerados en 2.17. Esta es una operación de tiempo constante si se posee T_b de antemano, salvo por las observaciones normalizadas 2.23, que toman Δ, que solo Perigee (SubsetScoring y VanillaScoring) utilizan. En total, al contabilizarse sobre todos los nodos V, estas operaciones pueden ser de tiempo $O(|V|)$, o tiempo $O(\Delta |V|)$, dependiendo del protocolo de los nodos.

Selección de vecinos

Cada nodo $v \in V$, tras una cantidad B_r de bloques propagados, ejecuta su algoritmo de selección de vecinos. De la definición 2.19, se puede notar que el protocolo EdgePriority debe sumar cuántas de las veces cada vecino es el primero en propagar bloques, lo que en total tarda B_r en tiempo. Los protocolos VanillaScoring y SubsetScoring utilizan otros cálculos más sofisticados, pero que en total son solo polinomios de d_{scored} y B_r. En total, todos se pueden resumir como un costo $O(P[B_r, d_{\text{scored}}])$.

Una vez que se deciden que vecinos eliminar, esto se traduce en eliminaciones de aristas en una estructura de datos de un grafo, que y con una estructura de datos como una tabla de hash, puede ser realizado en $O(1)$ para cada eliminación.

Si todos los nodos de la red desean eliminar d_{scored} vecinos cada uno, esto tardaría $O(|V| d_{\text{scored}})$, sin embargo, no todos los nodos se sincronizan para esto, y como la selección se hace cada B_r bloques, el costo amortizado en $|B|$ bloques para cada nodo sería en el orden de:

$$\text{Costo amortizado de selección, por nodo } = O\left(\frac{|B|}{B_r} P[B_r, d_{\text{scored}}]\right).$$

Generación de nuevas conexiones

Anteriormente, en la sección (2.1.2) Se mencionó que para la generación de nuevas conexiones, los nodos publican una lista de solicitudes a una estructura de datos R, que tardaba $O(\sum_v r_v)$ tanto en recibir, como en resolver todas las solicitudes, donde r_v es el número de

vecinos solicitados por cada v. De acuerdo a 2.18 y 2.19, la cantidad de vecinos a remover y agregar es d_{scored}.

De manera similar a la anterior, el costo se amortiza porque solo se realiza cada suficientes bloques B_r.

$$\text{Costo amortizado de conexiones, por nodo } = O\left(\frac{|B|}{B_r} d_{\text{scored}}\right).$$

Se recuerda que además, excepcionalmente, se debe ejecutar la fase de inicialización al comienzo, y eso corresponde a solicitar cerca de Δ nodos por cada nodo, lo que genera un costo de $O(\Delta)$ por cada nodo, al inicio.

Costo total

Considerando una red con $|V|$ nodos, contabilizando $|B|$ bloques propagados, en nodos que tienen d_{scored} vecinos a seleccionar, a través de rondas de B_r bloques, se debe:

1. Inicializar las conexiones de la red, $O(|V|\Delta)$,
2. Realizar el pre-cómputo para el muestreo, $O(|V| \log |V|)$,
3. Propagar bloques $|B|$ veces, repitiendo:
   (a) Generar un bloque en $O(\log |V|)$, (o $|V| \log |V|$ con churn)
   (b) Propagarlo por la red, en $O(\Delta |V| \log |V|)$.
   (c) Cada nodo procesar la información, en $O(|V|)$ total si no se usa Perigee o $O(|V|\Delta)$ en caso contrario.
   (d) Cada nodo procesar la información, en $O(|V| P[d_{\text{scored}}, B_r]/B_r)$.
   (e) Cada nodo solicitar conexiones, en total $O(|V| d_{\text{scored}})$.

Absorbiendo los tiempos que quedan mayorados por otros, se obtiene:

$$\textbf{Complejidad en Tiempo} = O\left(|B||V|\left(\Delta \log |V| + P[d_{\text{scored}}, B_r]/B_r + d_{\text{scored}}/B_r\right)\right). \tag{2.72}$$

Aquí, el término dominante suele ser $|B||V|\Delta \log |V|$.

Capítulo 3

Implementación y Experimentos

En este capítulo se separa en dos secciones. La primera muestran los detalles generales de cómo se implementó el modelo en un programa computacional. En la segunda se muestran algunos experimentos realizados, y se comparan resultados con el artículo [38].

3.1. Implementación

Se implementó el modelo descrito en código C++ Se utilizó la arquitectura llamada **bucle principal**, común en el diseño de motores de simulaciones y videojuegos [28], para secuenciar procesos de la simulación en distintos tiempos, coherentes con el fenómeno real. Además se utilizó **multithreading**. Si se desea saber detalles de cómo usar la simulación, se puede consultar el anexo B.1.

3.1.1. Multithreading

Antes de describir la forma de implementación, se menciona que el programa usa multithreading, pudiendo realizar algunos cómputos en simultáneo de forma de aprovechar mejor el hardware debido a que la simulación realiza cientos de miles de operaciones y esto puede tardar bastante tiempo en completarse. Se llama **hilo** a cada unidad de cómputo paralela. Un hilo se encarga de realizar cierto trabajo en determinado momento, y él podría tentativamente leer y escribir en la memoria principal del computador. En el programa, los hilos aprovecharan el paralelismo con algunas limitaciones: Si bien, dos o más hilos pueden leer la misma ubicación de memoria en simultáneo, no se permitirá que ambos puedan escribir en una ubicación de memoria al mismo tiempo. Más aún, el programa se diseñó de tal forma que los hilos solo escriben en momentos que garantizan que la escritura será en un lugar separado de otros hilos. Por ejemplo, hilos distintos se encargan de procesar las observaciones de nodos distintos, y escriben solo en un lugar asociado a cada nodo, eliminando la posibilidad de encontrarse escribiendo en la misma ubicación. Este estilo de trabajo permite evitar tener que hacer **exclusión mutua**, que es un requerimiento (a veces difícil de implementar) que se debe considerar cuando existen las llamadas **secciones críticas**, que son partes del

código a las que más de un hilo puede acceder con la capacidad de escribir en una ubicación compartida en memoria con otros hilos [48].

3.1.2. Bucle principal

El bucle principal es la secuencia ordenada de pasos que se lleva a cabo basándose en lo definido a en el modelo descrito en el capítulo anterior (específicamente, 2.13). En el bucle principal se implementa la dinámica central de la red para la simulación, y este bucle se define en torno a la propagación de un bloque por la red, frente a esta propagación, los nodos de la red realizan observaciones, toman decisiones mediante sus protocolos, cerrarndo y abriendo conexiones a otros nodos. A continuación se enumeran los tres procedimientos principales.

Primero, se propaga un bloque. Para ello, se elige un nodo al azar entre los nodos activos, de acuerdo a la ley f_v y este se considera el origen del bloque b; a partir de eso, se construye el grafo T_b de propagación, y en base a este, se construyen las observaciones para cada v nodo de la red.

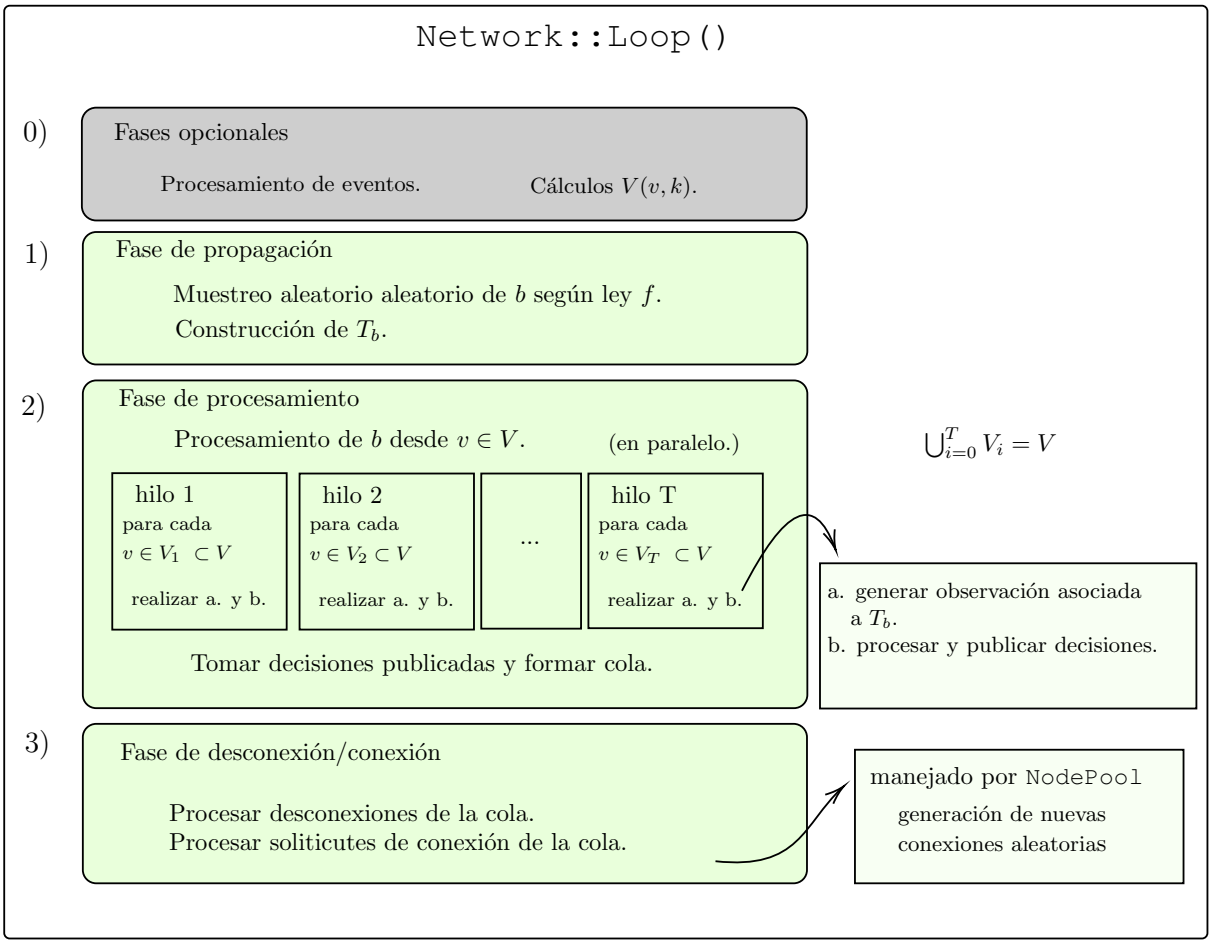
Segundo, a partir de las observaciones, cada nodo v procesa lo recién visto, en un espacio de memoria designado separadado de los demás. Esto se realiza en manera independiente en cada nodo v, en base a leer T_b y datos relacionados a v como Γ_v^o,l_{uv}, etc... Lo recién mencionado permite realizar el procesamiento de los nodos mediante multithreading. Dependiendo de los resultados del procesamiento, cada nodo publica sus acciones, lo cuál genera una cola de acciones a realizar, que incluye remover conexiones y producir solicitudes de conexión entre nodos.

Tercero, a partir de la cola generada en la parte anterior, se realizan las desconexiones y para realizar nuevas conexiones, se ejecuta un proceso en un objeto llamado `NodePool`, que se encarga de producir las conexiones de manera aleatoria, que se describe brevemente en la sección 3.1.3.

El orden de las acciones en el bucle se realiza de esta manera de acuerdo a la definición del modelo 2.13. Luego del tercer paso, se puede iniciar el bucle nuevamente, y de esa manera se simula la propagación de varios bloques. En una simulación, la forma de llevar a cabo esto es usar un ciclo que llama repetidas veces a una función llamada `Network::Loop()`. Opcionalmente, se puede realizar un par de pasos adicionales al comienzo del bucle: procesar *eventos*, que corresponden a acciones externas al bucle (como puede ser, desactivar un nodo) que requieren cómputos adicionales para que se pueda realizar el bucle de nuevo. Por ejemplo, al desactivar un nodo, ya no es capaz de propagar bloques, y por lo tanto, la ley utilizada para elegir un nodo al azar cambia, y es necesario realizar un procedimiento para que eso ocurra (actualizar en (2.8)).

Además, algunos nodos pueden llevar a cabo una *actualización de visibilidad* que involucra calcular nuevamente $V(v, k)$, que es un procedimiento costoso, pero que gracias a ser independiente entre cada nodo (mientras no cambia el grafo), se puede llevar a cabo en los nodos que lo decidan, usando paralelismo al comienzo del bucle. En la figura 3.1 se muestra el trabajo que realiza el bucle principal en `Network::Loop()`y cómo este se divide usando paralelismo por hilos.

Figura 3.1: Diagrama de las fases del bucle principal en la implementación.



3.1.3. Selección Aleatoria: `NodePool`

Se mencionó anteriormente la necesidad de una estructura de datos que se encarga de hacer las conexiones aleatorias en los nodos de la red. En particular, se especificó que la estructura debería:

- Insertar solicitudes de conexión;
- extraer al azar solicitudes, en que la selección se realice con probabilidad uniforme sobre los nodos con solicitudes activas (no cero);
- adicionalmente, actualizar o eliminar solicitudes;

y que además todas estas operaciones sean realizadas en tiempo constante.

La estructura que realiza esto le define por `NodePool`, que por razones de análisis, se denota a veces como R, cuya implementación concreta se encuentra en la sección A.2.2 del anexo.

Para llegar a una estructura con esas características, se consultó textos de algoritmos y simulación estocástica, pero no se encontró material. Sin embargo, buscando en internet se encontró una especificación detallada para una estructura de datos con los requirimientos, que fue encontrada en un sitio web de programación [27].

3.1.4. Abstracción y herencia

La simulación tiene una gran cantidad de componentes: nodos, grafos, objetos que caracterizan las latencias, algoritmos de selección de vecinos, etc. Debido a que algunos de ellos cambian, por ejemplo, cuando se desea simular la ejecución de un protocolo en contraste con otro, pero otros componentes se reutilizan, es que se decidió seguir un paradigma de programación orientada a objetos con C++.

La selección de vecinos tiene un comportamiento casi idéntico entre SubsetScoring y EdgePriority: los dos comportamientos que cambian son que SubsetScoring elige observar más información, y también que ambos protocolos realizan distintos cálculos con la información, pero, el comportamiento en la salida es la misma: ambos producen un subconjunto de vecinos a descartar. Para aprovechar las similitudes, es que se programa una clase *virtual* llamada `ScoringProtocol` con la mayor parte de las características de un protocolo de esas características, y luego se crean dos clases `SubsetScoring` y `EdgePriority` que *heredan* las funcionalidades de `ScoringProtocol`, pero se especializan en las funcionalidades con las que marcan sus diferencias.

De la misma forma que para los protocolos, también se crean clases para representar las topologías de internet, y solo necesitan, a groso modo, una forma de poder calcular una función $(u, v) \to l_{uv}$.

3.2. Experimentos

3.2.1. Estimación de muestras necesarias para la métrica de latencia global

En las secciones 2.2.3 y 2.2.3 se discutió la necesidad y los métodos para estimar $\mathbb{E}_b[l(T_b)]$ y λ^α, en vez de realizar un cálculo exhaustivo. En esta parte se realiza un cálculo exhaustivo para así calcular la magnitud de las desviaciones que se pueden observar al estimar esta cantidad, y se usa los resultados para determinar el número n de muestras necesarias para mantener la magnitud de esas desviaciones a un valor pequeño.

Para estimar $\mathbb{E}_b[l(T_b)]$

Para determinar un porcentaje de muestras satisfactorio para los experimentos posteriores, se toma un escenario básico, con $|V| = 10{,}000$ nodos, con poder de hash uniforme. Se incrusta $v \in V$ dentro de $[0,1]^{10}$ con una x_v muestra al azar uniforme según la medida de lebesgue. La latencia par a par l_{uv} está dada por

$$l_{uv} := \frac{1}{\sqrt{10}}||x_u - x_v||_2 \quad [s].$$

Para la topología $\mathcal{N}$ sobre esta red, se impone $d_{out} \leq 15$, $d_{in} \leq 25$. Se generan cinco redes aleatorias distintas (usando distintas semillas), y para cada una de ellas se se obtiene una distribución para la latencia global $l(T_b)$, calculando el valor $l(T_b)$ de manera exhaustiva para cada $b \in V$. El resultado de la distribución está en la figura 3.2. Luego, cada red aleatoria se entrena con $|B| = 4000$ bloques con nodos ejecutando el protocolo EdgePriority, de rondas de 100 bloques y $d_{\text{scored}} = 12$ ($d_{\text{random}} = 3$), y permitiendo una visibilidad de $k = 2$ para las conexiones aleatorias en vecindades $V(v,k)$. La distribución de $l(T_b)$ está en la figura 3.3.

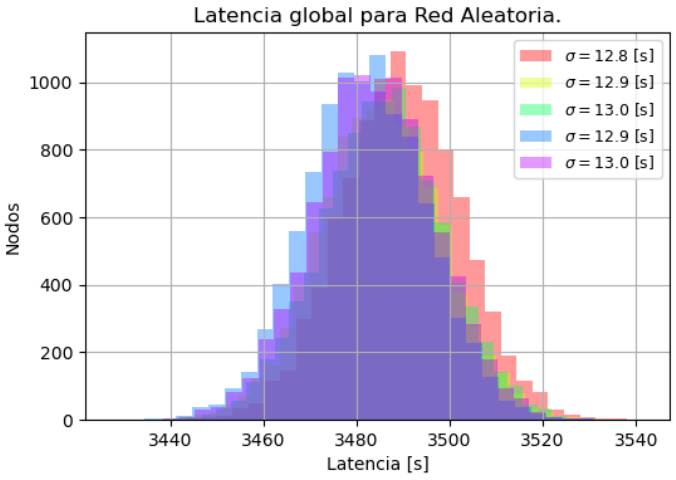


Figura 3.2: Distribución de los valores $l(T_b)$ para $b \in V$, en el caso de una red aleatoria.

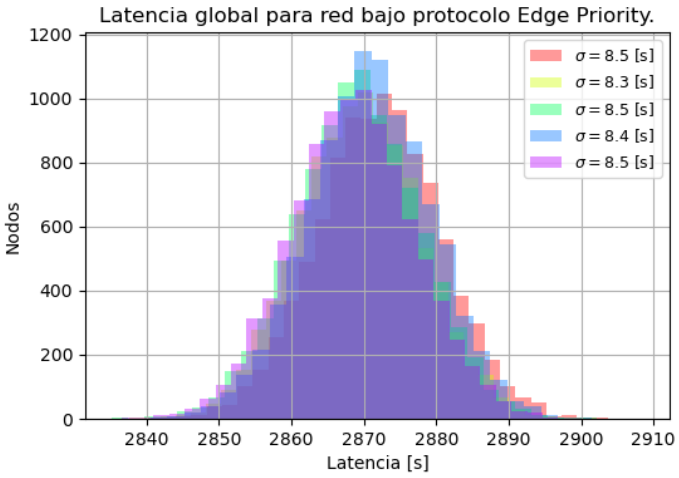


Figura 3.3: Distribución de los valores $l(T_b)$ para $b \in V$, en el caso de una red entrenada por $|B| = 4000$ bloques bajo el protocolo EdgePriority.

Observación La latencia máxima de l_{uv} es $1[s]$. La arista promedio en T_b tendría latencia $\approx 0,35[s]$. El hecho sale de que $\mathbb{E}[l(T_b)]$ está en torno a valores como 3500 y conociendo que T_b debería tener $|V| - 1 = 9999$ aristas dirigidas.

Según los resultados, la desviación estándar de $l(T_b)$ se encuentra en orden de $2 \cdot 10^1$, en comparación al valor del promedio, $3 \cdot 10^4$, esta desviación representa un $0,57\,\%$. Si se desea estimar con un error bajo en comparación a los desplazamientos de $\mathbb{E}_b[l(T_b)]$, se puede tomar el error para que la desviación de estimación 2.48 sea de el $1\,\%$ de la variación de una latencia global a otra. La distancia entre el promedio de ambas campanas es alrededor de $600[s]$, pero para ser más generosos, se considera que es mucho menor, $268[s]$, luego se busca

$$\frac{3 \cdot 2 \cdot 10^1}{\sqrt{n}} = 1\,\% \cdot 268[s].$$

De esto, solo basta tomar $n = 500$ muestras para mantener la desviación $3\sigma/\sqrt{n}$ bajo $1\,\%$ de la diferencia $268[s]$.

Observación Es llamativo que las variaciones sean tan pequeñas entre distintas realizaciones de redes, y que la desviación de $l(T_b)$ sea pequeña además, sin embargo, se vuelve más razonable si se piensa en toda la simetría que posee el espacio $[0,1]^d$; una red aleatoria puede ser muy distinta de otra en los detalles específicos, como las conexiones, pero en escala, se debe comportar muy similar, especialmente la propagación de la información a través de aristas dentro del cubo. Por otra parte, la estabilidad de $l(T_b) = \sum_{uv \in T_b} l_{uv}$ podría deberse a que la propagación tiende a ocurrir por canales/aristas de relativa baja latencia en comparación a la bastia mayoría de aristas, lo cual debería reducir la varianza de la suma en total, o al menos, en apariencia.

Para estimar λ^{α}

Se utilizan los mismos experimentos de la sección anterior, es decir, cinco redes inicializadas aleatoriamente, y esas cinco se entrenan por $|B| = 4000$ bloques usando EdgePriority. En aquellas redes, se toma la muestra exhaustiva de $(\lambda_v^{90\,\%} \ : \ v \in V)$ y se produce la tabla 3.1, con los promedios en diez tramos de diez percentiles. Se identifican las curvas de la red aleatoria con Y y la red entrenada con X. Los bordes de cada tramo se toman como p_-^x, p_+^x en la ecuación 2.71. Se muestra un gráfico de las curvas en la figura 3.4, demarcando los tramos de percentiles las intersecciones con rectas horizontales y los promedios por tramo con demarcaciones verticales colorizadas.

Tramo Percentiles	Promedio Red Entrenada [ms] $\mu_{\hat{X}}$	Promedio Red Aleatoria $\mu_{\hat{Y}}$ [ms]	$\lvert\mu_{\hat{Y}} - \mu_{\hat{X}}\rvert$[ms]
0-10	959	1170	211
10-20	970	1194	224
20-30	982	1206	224
30-40	991	1216	225
40-50	1007	1226	219
50-60	1021	1236	215
60-70	1037	1245	208
70-80	1052	1254	202
80-90	1070	1270	200
90-100	1098	1296	198

Tabla 3.1: Valores promedio en cada percentil para las curvas λ^{α} calculadas exhaustiamente sobre redes aleatorias y redes entrenadas por $|B| = 4000$ bloques bajo EdgePriority.

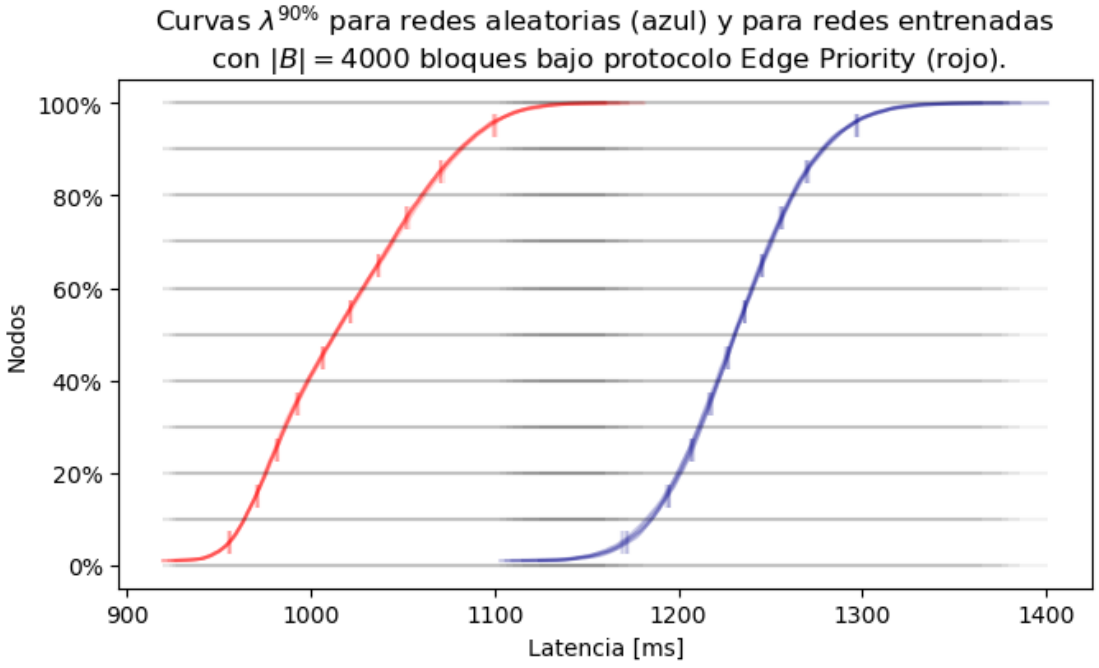


Figura 3.4: Curvas λ^{α} para redes aleatorias y redes entrenadas con $|B| = 4000$ bloques con protocolo EdgePriority. Las líneas grises horizontales demarcan los percentiles, y las marcas de color verticales demarcan el valor de $\mu_{\hat{X}}, \mu_{\hat{Y}}$ para cada uno de los diez tramos. Son 5 curvas rojas, y 5 curvas azules superpuestas, lo cuál es llamativo porque las curvas parecen comportarse exactamente igual.

Si se ingresan los datos obtenidos en el cálculo de $n(\varepsilon)$ de 2.71, con $\varepsilon = 10\,\%$, se necesitan aproximadamente,

$$n(\varepsilon) \approx 2 \ln\left(\frac{2}{\varepsilon}\right)\left(\frac{30+30+200}{\delta}\right)^2.$$

para garantizar que,

$$\left|\frac{1}{n}\sum_i \hat{X}_i - \frac{1}{n}\sum_i \hat{Y}_i - (\mu_{\hat{X}} - \mu_{\hat{Y}})\right| \leq \delta,$$

con probabilidad de falso positivo menor que $10\,\%$. Al elegir $\delta = 60$ [ms], es decir, cerca de un tercio de la separación, se requieren $n(10\,\%) \approx 112$ muestras por cada tramo de percentiles, en total, 1120 muestras aproximadamente, un $11\,\%$ del total de $|V|$, y menos de un $3\,\%$ del total si $|V| = 50000$.

Esto puede ser bastante débil, y en la práctica se observa que las curvas se estabilizan bien con bastantes menos muestras. En las simulaciones puede ser necesario tomar una decisión del número de muestras contextualizandose también por la necesidad de eficiencia de recursos, ya que elevar la cantidad de muestras mucho más allá, genera que las simulaciones tomen mucho tiempo. Para estos mismos datos, usar $\delta = 90$ [ms] permite solo usar $n = 500$ muestras en total. De todas maneras, $\delta = 90$ [ms] es suficiente para diferenciar las dos curvas, que están separadas por al menos 200 [ms].

3.2.2. Simulaciones en varios escenarios distintos

A continuación se muestran los resultados de simular redes bajo distintos escenarios. Se comienza por un escenario por defecto, que sirve de base comparativa para los otros experimentos, que se hacen después, y consisten en variaciones de parámetros de la red. El primer escenario con variación contempla cambios en la latencia l_{uv}, escenario que también fue comparado en [38] con respecto de una simulación base. Similarmente, el siguiente escenario es para evaluar cómo se comporta la red cuando hay una pequeña porción de la red que tiene capacidades mucho más superiores al promedio de los nodos, tanto en poder de hash como en latencias. Estas dos variaciones mencionadas se ven en [38], pero los autores de ese artículo señalan la importancia de entender el comportamiento del protocolo bajo churn, para el que su simulación no estaba preparada. Por ello, a continuación se evalúa el efecto del churn sobre la red usando un modelo simple, en que tras cada bloque se activan y desactivan una cantidad aleatoria de nodos. Los tres escenarios anteriores evalúan el impacto de condiciones externas a los nodos, mientras que los tres siguientes se dedican a explorar qué pasa al cambiar parámetros propios del protocolo o la conectividad. Se realiza una comparativa sobre cómo el protocolo EdgePriority se comporta si se cambia el número de rondas por cada selección de vecinos. Además, se realiza una comparativa de cambiar el número de nodos seleccionados y aleatorios, y finalmente un escenario que muestra un fenómeno interesante al levantar la restricción d_{in}.

Escenario por defecto

Se enumeran características de la red con la que se simula:

1. $|V| = 10{,}000$ nodos,
2. Incrustados en $X = [0,1]^d$,
3. Latencia l_{uv} dada por,

$$l_{uv} := 2000 \cdot \frac{||x_u - x_v||_2}{\sqrt{10}} + 500[\text{ms}].$$

   La latencia se considera como una magnitud aleatoria que varía entre 0 y 2000 [ms], más un valor constante, que se considera un "costo por salto" en la red.

4. $d_{out} \leq 20$ (los nodos tratan siempre de mantener esta cantidad en 20).
5. Se inicializa la red mediante la fase *bootstrap*, generando vecinos aleatorios con visibilidad total ($G_{20,out}$).
6. $d_{in} \leq 80$, pueden tener hasta ochenta vecinos entrantes.
7. La visibilidad que tienen los nodos $V(v,k)$ para generar conexiones nuevas es con $k = 2$ (vecinos de vecinos).
8. Se propagan $|B| = 10{,}000$ bloques en total.

9. Esta misma red se genera una vez para ser entrenada con SubsetScoring, y otra vez (es la misma) para ser entrenada con EdgePriority.

10. Los protocolos usan $d_{\text{scored}} = 16$, $d_{\text{random}} = 4$, y rondas de 100 bloques. En total, los nodos pueden realizar aproximadamente 100 rondas.

11. Se miden las curvas λ^{α} y la latencia $\mathbb{E}_b[l(T_b)]$ al inicio (recién inicializada), y cada $1000 \approx 10 \cdot 100$, osea cada 10 rondas, hasta finalizar. También se guardan propiedades de la red en archivos (listas de adyacencia de grafos, etc.).

Además, el experimento recién enumerado se realiza 3 veces (es decir, 6 simulaciones en total). En cada oportunidad, se cambia la *semilla* de la generación aleatoria de vecinos, generandose redes distintas al comenzar, y conexiones aleatorias distintas entre v y $V(v, k)$ en las fases posteriores.

Adicionalmente, en cada ronda se almacenan los valores de latencia de las aristas ($l_{uv} : uv \in E(G_{\text{TCP}})$), y a partir de ello, se elabora un gráfico con las funciones de distribución (exactas) de los valores l_{uv}. Esto solo se hace para una simulación de SubsetScoring y de EdgePriority.

Resultados del escenario por defecto

Se obtiene que ambos protocolos, tras $|B| = 10000$ bloques, logran optimizar la curva $\lambda := \lambda^{90\,\%}$ por una cantidad considerable. Por ejemplo, en la figura 3.5 se observa que SubsetScoring lleva la mediana de λ desde 3800[ms] a 2900[ms], reduciendo la latencia mediana al 90 % de la red, en aproximadamente un 20 %. EdgePriority también logra optimizar la curva, pero el resultado es más modesto, desde 3800[ms] a 3200[ms] en la mediana, un 16 % aproximadamente. Se contrasta además, que el protocolo SubsetScoring logra una distribución ligeramente más acumulada, con menos desviación, obteniendose que casi el 100 % de los nodos tienen un valor λ_v acotado por 2900 y 3200[ms], teniendo un márgen de 400[ms], mientras que para EdgePriority, las latencias se esparcen entre 3050 y 3400. También SubsetScoring muestra un fenómeno interesante: la masa de λ se acumula paulatinamente en dos centros separados notablemente separados, específicamente, se observa el fenómeno en los primer 20 % de los valores. Este fenómeno parece también ocurrir en EdgePriority, pero con mucha menos preponderancia que en SubsetScoring, y solo en las últimas mediciones, mientras que para subset el fenómeno parte a notarse desde la tercera (cuando $|B| = 2000$).

A pesar de lo anterior, no se puede decir que las simulaciones estabilizan la topología, puesto que al examinar la tendencia de la métrica de latencia global $\mathbb{E}_b[l(T_b)]$ en la figura 3.6, se nota una paulatina disminución a lo largo de las mediciones tomadas, pero no se podría decir con certeza si esta curva ha llegado a su mínimo posible o si continuaría disminuyendo con más rondas. Lo que sí es seguro, es que SubsetScoring es más efectivo que EdgePriority en reducir aquella latencia, pero ambos logran un desempeño razonable para optimizar.

Una de las razones de por qué SubsetScoring podría estar superando a EdgePriority tiene que ver con que el primero logra acumular mayor cantidad de aristas de baja latencia, como se aprecia en la figura 3.7, allí se nota que para la curva de los $|B| = 10000$ bloques,

SubsetScoring obtiene 80 % de las aristas con latencia menor a 1100[ms], mientras que para EdgePriority, el porcentaje de aristas con latencia menor a 1100[ms] ronda por el 50 %. También se nota que ambos protocolos de igual manera retienen aristas de alta latencia, en vez de eliminarlas por completo. La siguiente explicación ya fue notada en [38] para los resultados de las simulaciones de su artículo: El tener una cantidad significativa de aristas de baja latencia podría estar produciendo que las rutas de propagación utilicen aquellas latencias más bajas, y por otro lado, es razonable que existan aristas de alta latencia pues pueden ser más eficientes para conectar regiones alejadas (geográficamente) de la red, que de ser conectadas por aristas más cortas tendrían que dar muchísimos saltos.

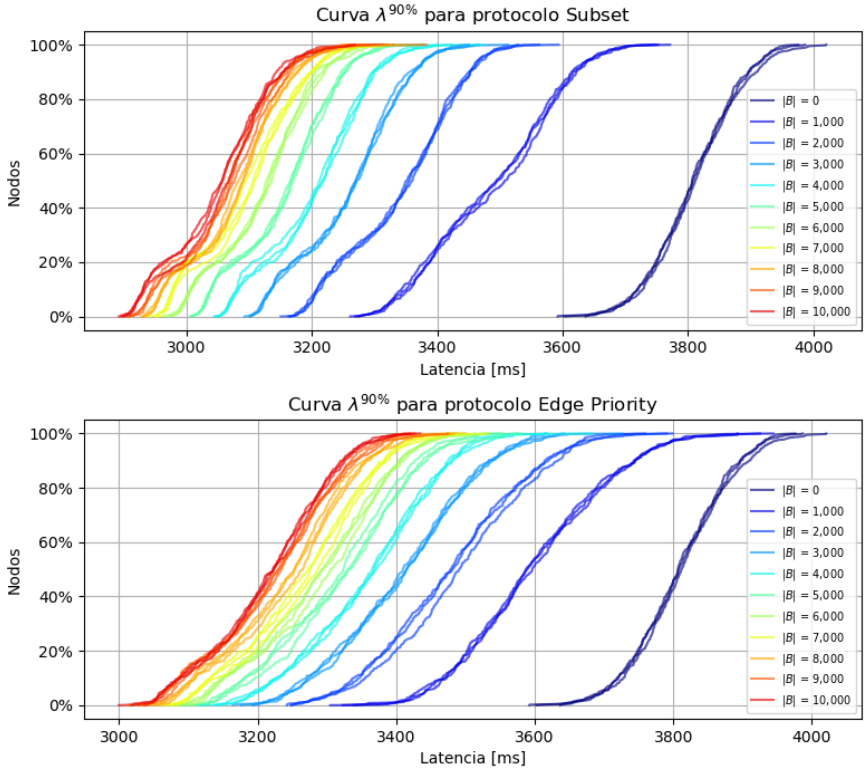


Figura 3.5: Curvas $\lambda^{90\,\%}$ medidas varias veces para varias etapas de propagar bloques usando el protocolo SubsetScoring y EdgePriority. Los colores de las curvas las identifican con el número de bloques La curva $|B| = 0$ corresponde a la red recién iniciada aleatoriamente, y la curva $|B| = 10000$ corresponde a la final.

En general, para esta primera simulación, los resultados se asemejan a los de [38].

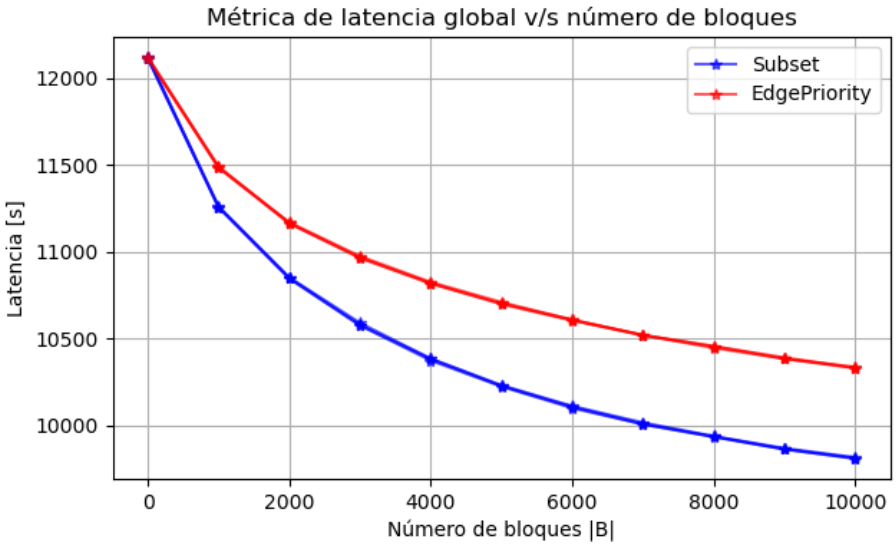


Figura 3.6: Evolución de la latencia global $\mathbb{E}_b[l(T_b)]$ para redes entrenadas con SubsetScoring y EdgePriority a medida que se observan más bloques.

Escenario con variación de constante de la latencia

En este escenario, se cambia l_{uv} por otra versión l^m_{uv}:

$$l^m_{uv} := 2000 \cdot \frac{||x_u - x_v||_2}{10} + m \cdot 500[\text{ms}].$$

Lo que cambia en esta oportunidad es que ahora el valor de la latencia constante o "costo de salto" está ponderado por m. Se realiza el mismo experimento que la versión por defecto, pero se pone la latencia m con $m \in \{1/10, 5/10, 5, 10\}$.

Una forma de entender el interés por este experimento es que a medida que $m \nearrow 10$, la parte de la latencia que depende de la distancia se vuelve menos importante que una latencia que es igual para todos los nodos, independiente de dónde estén. Si se lleva el m a niveles extremos, la latencia debería ser, relativamente, casi la misma para todos los nodos, y el problema de optimización pasaría a ser similar al de optimizar la topología para una latencia constante o uniforme (grafos sin pesos). Al reverso, cuando $m = 1/10$, es una topología donde importa muchísimo más la variabilidad par a par, para la latencia.

Resultados para el escenario de variación de constante en la latencia

Para todas las topologías l^m_{uv}, las redes logran optimizarse hasta cierto grado, pero varía fuertemente el % de diferencia de $\lambda^{90\,\%}$ entre la red aleatoria y el resultado final. Los resultados son muy similares para ambos protocolos, por lo que se hablará principalmente de SubsetScoring. En 3.8 se muestra cómo evoluciona la red bajo SubsetScoring para cada m (EdgePriority está en la figura C.1 del anexo). Cualitativamente, llama la atención que la forma de todas las curvas en cada etapa de la evolución es la misma.

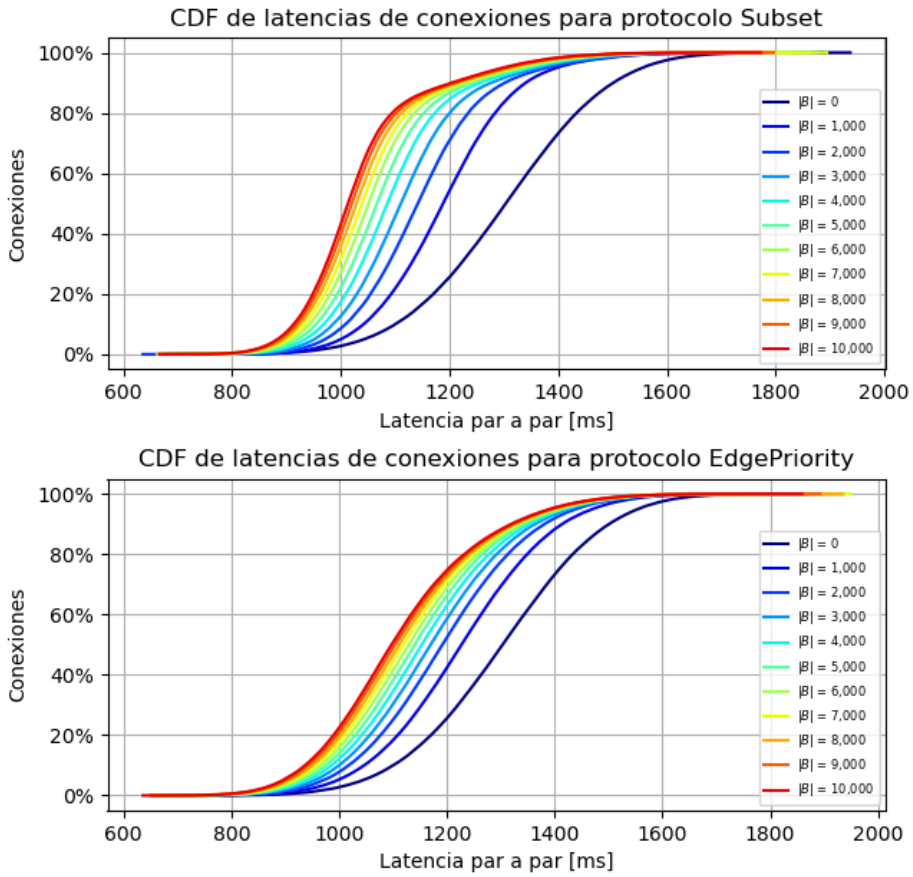


Figura 3.7: Curvas con la distribución de la latencia de las aristas para los protocolos SubsetScoring y EdgePriority, evolucionando a medida que se propagan $|B| = 10000$ bloques. El color de la curva identifica el momento en que se midió.

En la tabla 3.2 (con valores aproximados) se consideran los percentiles 40 % de cada curva aleatoria y la versión más optimizada bajo SubsetScoring. Inmediatamente se nota que mientras más bajo es m, mayor es el porcentaje que el protocolo puede optimizar la latencia $\lambda^{90\,\%}$, lo cuál también puede relacionarse con el hecho de que la diferencia en [ms] no cambia sustantivamente, pues se mantiene cerca de los 750[ms] para todos los casos, lo cual puede atribuirse a que ese valor es un porcentaje fijo de la parte de la latencia que depende de $||x_u - x_v||$. Si ese es el caso, entonces elevar m produciría lo observado. Vale mencionar que este comportamiento es similar a los resultados en el experimento de variar este tipo de latencia en [38]. Algo interesante a destacar es que la distribución de latencia de aristas sigue la misma forma para todas las l^m_{uv}, y solo cambia el desplazamiento a la derecha de la misma, como se ve en la figura 3.9 para $m = 1/10$ y $m = 10$.

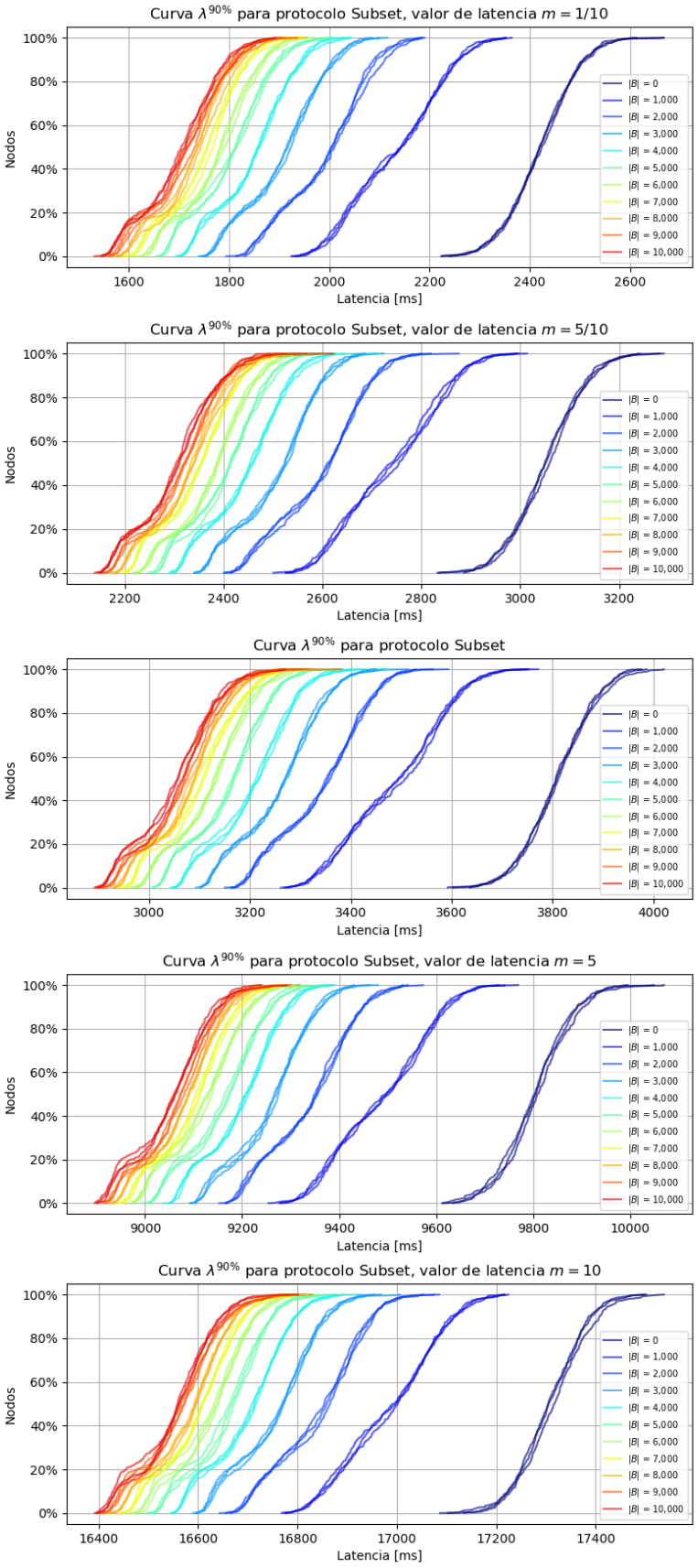


Figura 3.8: Curvas $\lambda^{90\,\%}$ para SubsetScoring en escenarios con variación en la constante de latencia.

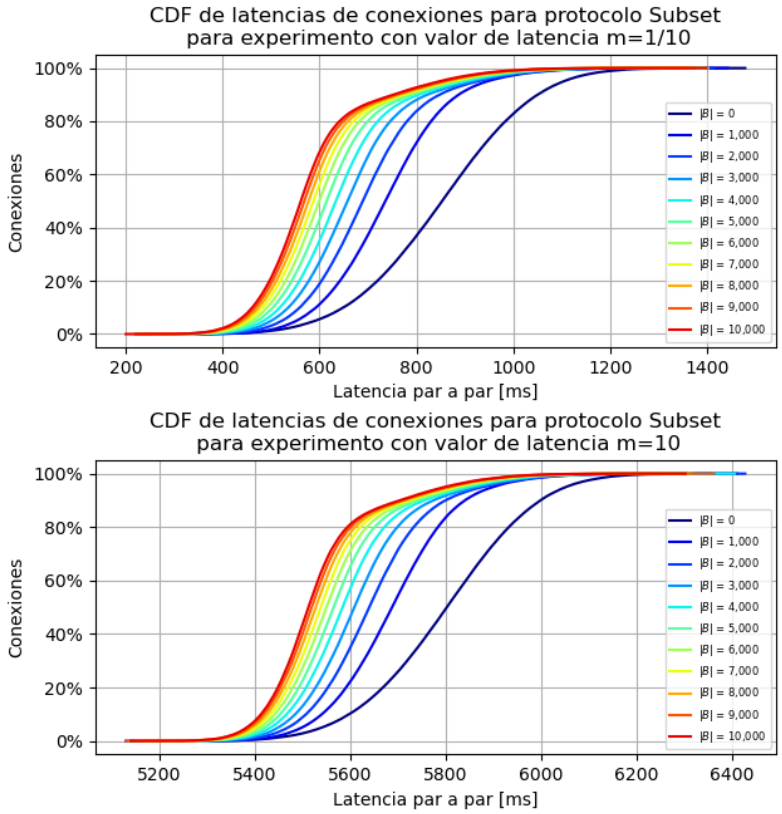


Figura 3.9: Distribución de latencia de las aristas para SubsetScoring actuando sobre topologías de internet l_{uv}^{m}, $m = 1/10$ y $m = 10$.

m	Aleatoria [ms]	Optimizada [ms]	Diferencia [ms]	Porcentaje
1/10	2400	1700	700	29 %
5/10	3050	2300	750	24,5 %
1	3800	3050	750	20 %
5	9800	9040	760	8 %
10	17300	16500	800	4,6 %

Tabla 3.2: Tablas de valores de $\lambda^{90\,\%}$ en el percentil 40 para las curvas inicializadas aleatoriamente y tras la optimización con SubsetScoring, en topologías l^m_{uv} con distintos valores m.

Escenario con presencia de nodos de alto rendimiento

En este escenario, la variación es que en vez de que todos los nodos tengan el mismo poder de hash, se separa en un conjunto V_1 para el que cada nodo $v \in V_1$ tiene $F_v = 9$, y en el complemento, $u \in V - V_1$, $F_u = 1/9$, y $|V_1| = |V|/10$, de modo que el conjunto V_1 tiene el 90 % de poder de hash de la red, y tan solo siendo el 10 % de los nodos de ella.

$$\sum_{v \in V_1} F_v + \sum_{u \in V - V_1} F_u = 10\,\% \cdot 9 + 90\,\% \cdot \frac{1}{9} = 90\,\% + 10\,\% = 100\,\%.$$

Adicionalmente, las conexiones dentro de este grupo se ven beneficiadas: si $(u, v) \in V_1 \times V_1$,

$$l_{uv} := 200 \cdot \frac{||x_u - x_v||_2}{\sqrt{10}} + 500 \text{ [ms]},$$

es decir, se disminuye la latencia del enlace a 1/10 que el valor normal, entre los nodos de V_1 (simula tener buena conexión de internet).

El interés detrás de este tipo de simulación es que en las redes blockchain reales hay nodos que tienen comparativamente mucho mayor poder computacional y conexión de internet que otros, siguiendo leyes de pareto.

Resultados para el escenario con presencia de nodos de alto rendimiento

Primeramente, comparando la figura de los resultados en la red con nodos de alto rendimiento 3.10 con la del escenario de base 3.5, se nota, en la curva de la red aleatoria ($|B| = 0$), que la latencia para llegar al 90 % de la red ha disminuido por lo menos 200[ms] solo por la presencia de estos nodos; además, existe un porcentaje de nodos con latencia menor al resto: notar que la distribución se levanta mucho más a la izquierda. Además, a medida que evoluciona la red, tanto en SubsetScoring como EdgePriority se aprecia que en los nodos con menos latencia $\lambda^{90\,\%}$ se produce una separación: existe un conjunto, que contiene alrededor del 20 % que está aprovechando mejor las capacidades de la red. Lamentablemente, se observa en la figura 3.11 que es probable que la red aún pueda mejorar bajo la evolución de más bloques, debido a que hay una pendiente considerable. Por otro lado, llama la atención que EdgePriority produce aún menos porcentaje aristas de baja latencia (3.12) que en el caso por defecto (3.7), como se aprecia en su percentil 60. Esto se puede deber a que EdgePriority mantiene aristas si estas propagan la información más oportunamente que otras, y la presencia de nodos con mucho poder de hash acumulado puede desincentivar la búsqueda de aristas con menor latencia.

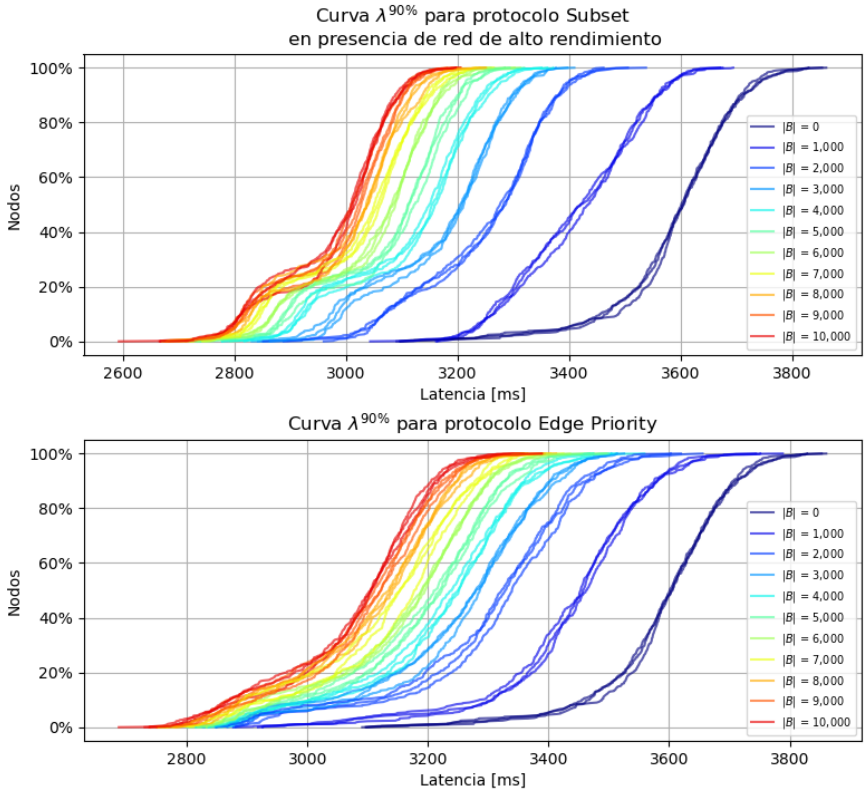


Figura 3.10: Curvas $\lambda^{90\,\%}$ para el escenario con presencia de nodos de alto rendimiento. Notar el levantamiento del 20 % de los nodos con menor latencia.

Escenario con simulación de churn

Esta simulación es como la básica, pero esta vez se comienza con 500 nodos desactivados (5 %), y cada vez que se propaga un bloque b, se elige una cantidad X_b de nodos que se desactivan, y una cantidad Y_b de nodos que se activan. Específicamente, X_b, Y_b son variables binomiales $B(|V|, p)$, por lo que el evento $X_b = k$ equivale a que ocurran k "aciertos", cada uno independiente, de probabilidad p; en este modelo, el "acierto" es que un nodo de los $|V|$ decide desconectarse justo después de que se propaga b. Se usa Y_b con la misma distribución para asegurarse que, en promedio, el número de nodos activos se mantenga en un mismo nivel. El número p es ajustado a datos aproximados del mundo real, $p =$ nodos que se desactivan por día/bloques por día $= 40/100$. Así, si se dice que si n_d es el número de bloques de un día,

$$\text{nodos que se desactivan por día} = n_d \cdot p = \mathbb{E}\left[\sum_{b=1}^{n_d} X_b\right].$$

Los datos fueron extraídos del apartado de churn de [6] pero es posible que se encuentren desactualizados.

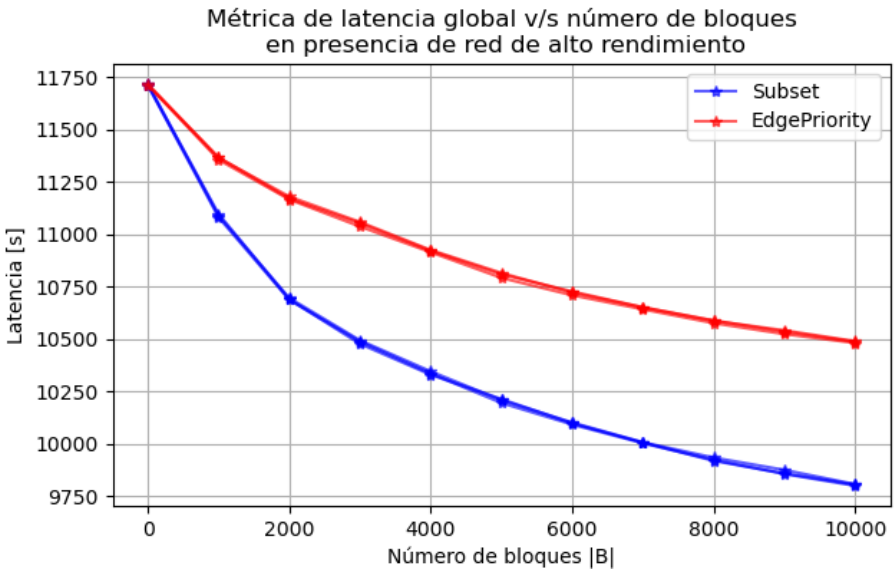


Figura 3.11: Evolución de $\mathbb{E}_b[l(T_b)]$ para el escenario con presencia de nodos de alto rendimiento. En relación a 3.6, en este escenario EdgePriority parece desempeñar peor.

Resultados para el escenario con simulación de churn

Los resultados se muestran en las figuras 3.13, 3.14, 3.15. Es interesante que la mayoría de las métricas se encuentran casi intactas, salvo por un efecto de aleatoriedad introducida por el churn, que se puede notar en que las curvas para distintas simulaciones están dispersas en la figura 3.13, y que la latencia $\mathbb{E}_b[l(T_b)]$ se separa más, pero el efecto en la optimización es importante.

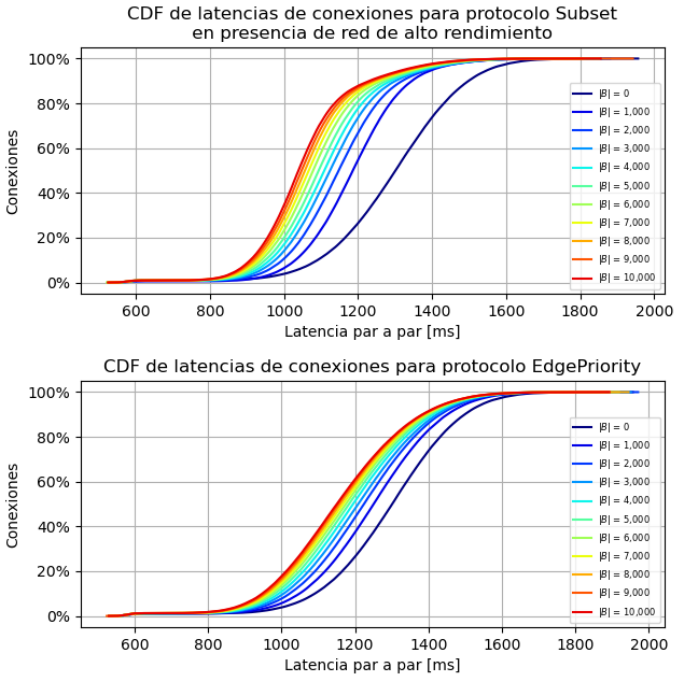


Figura 3.12: Distribuciones de latencias en redes para el escenario con presencia de nodos de alto rendimiento. EdgePriority tiene menos capacidad de obtener aristas de baja latencia que en el caso base 3.7.

Escenarios con rondas de distinta duración

En este conjunto de escenarios, se varía la duración en bloques, y se observa el resultado en cómo evoluciona $\mathbb{E}_b[l(T_b)]$. La duración de las rondas toma un valor r:

$$r \in \{20, 60, 100, 130, 175, 250, 500\},$$

para cada experimento. La cantidad de bloques $|B|$ propagarse se mantiene constante.

Resultados

En la figura 3.16 se aprecia las curvas para cada largo de ronda. Se puede observar que es necesario tener una cantidad mínima de rondas más allá de 20 dado que con ese parámetro, la red no logra hacer nada más allá de cómo se comporta mediante la conexión aleatoria. También se aprecia que existe un efecto negativo en aumentar demasiado el largo de la ronda, ya que para $r \in \{175, 250, 500\}$ se ve un peor rendimiento en comparación a $r \in \{100, 130\}$. Se puede pensar en dos efectos: primero, es necesario que se observe una cantidad no menor de bloques para lograr establecer bien los valores $\mathbb{E}_b[1_{u \to_b v}]$, pero, en sentido contrario, usar demasiadas mediciones no permite que la red explore nuevas aristas lo suficiente. Eso sí, se observa que la curva no se ha estabilizado, por lo que no se asegura si en el largo plazo estas diferencias se mantengan, o incluso se inviertan. Además, se agrega un ejercicio interesante: Si se considera que en $|B| = 8000$, la red con $r = 500$ ha hecho aproximadamente $8000/500 = 16$

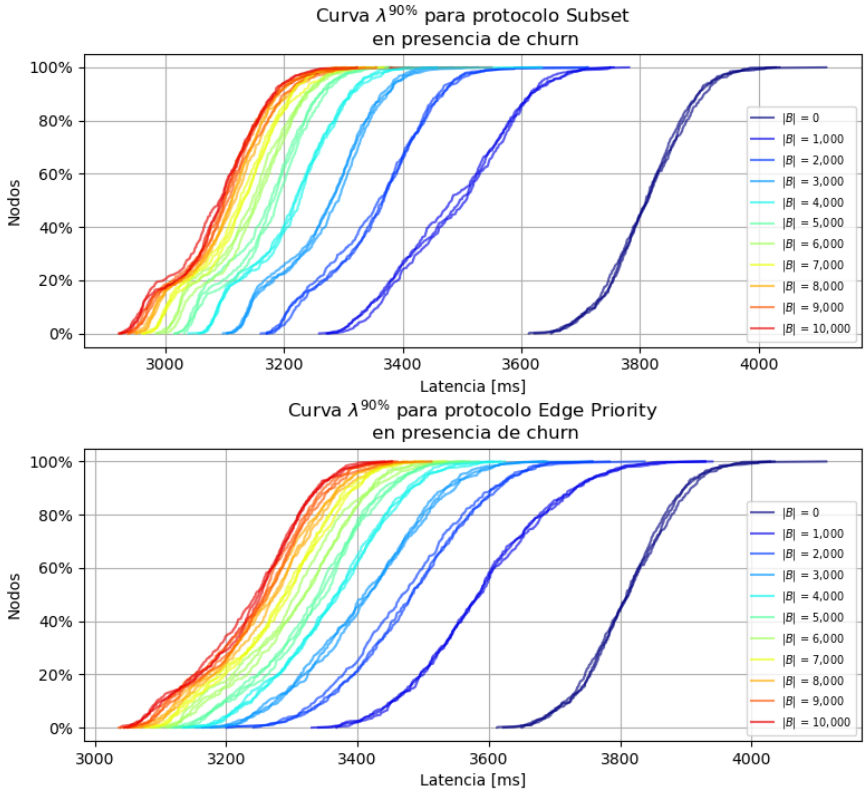


Figura 3.13: Curvas $\lambda^{90\,\%}$ para escenario con churn. El comportamiento es similar al caso base 3.5.

rondas, y en $|B| = 2000$ la red con $r = 100$ ha hecho aproximadamente $2000/100 = 20$ rondas, y se compara el rendimiento de ambas según el número de rondas, se observa que $r = 500$ es superior, en la medida que $\mathbb{E}_b[l(T_b)] \approx 10875$ es menor que lo que obtiene $r = 100$, $\mathbb{E}_b[l(T_b)] \approx 11125$ y con más rondas.

Escenarios variando número de vecinos seleccionados/aleatorios

En este escenario se mantiene fijo el número de vecinos salientes (20), pero se cambia la proporción de vecinos que se mantienen cada ronda debido a obtener alto puntaje, así mismo, esto repercute en la cantidad de vecinos aleatorios que se generan en cada ronda. Se usa el valor d_{scored} y $d_{\text{random}} = 20 - d_{\text{scored}}$:

$$d_{\text{scored}} \in \{5, 10, 13, 16, 19\}.$$

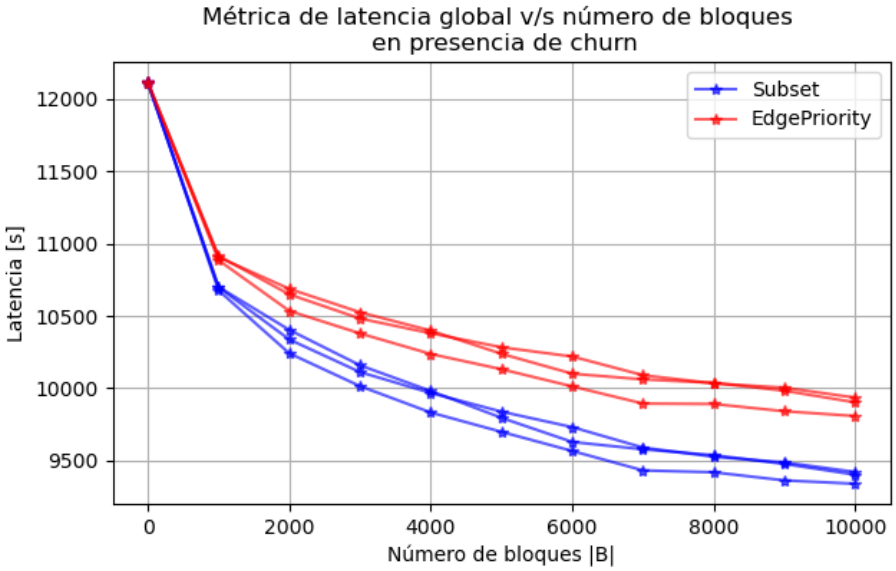


Figura 3.14: Evolución de $\mathbb{E}_b[l(T_b)]$ para escenario con churn.

Resultados

Los resultados se muestran en las figuras 3.17. Se observa que para $d_{\text{scored}} = 5$ ($d_{\text{random}} = 15$) la red comienza a optimizar la latencia en una tasa similar a redes con otros parámetros razonables, pero que rápidamente se estanca, separándose del nivel de optimización de las otras. Se podría decir que $d_{\text{scored}} = 10, d_{\text{random}} = 10$ tiene un comportamiento que parece estancarse, pues se ve superada por la curva $d_{\text{scored}} = 13$, y en tendencia pareciera ser que $d_{\text{scored}} = 16$ también la va a superar. Estos fenómenos parecen indicar que si bien una gran cantidad de vecinos aleatorios permite explorar más aristas, esto no permite aprovechar al máximo el conocimiento sobre los vecinos; desechar tantos vecinos se termina convirtiendo en un desperdicio de potencial. Por otra parte, para $d_{\text{scored}} = 19, d_{\text{random}} = 1$, la tasa de optimización de la latencia es muy lenta, no logrando disminuir ni la mitad, en toda la evolución $|B| = 10000$, lo que otras redes tardan en hacer en $|B| = 2000$.

Levantar restricción d_{in}

En las simulaciones, es común que algunos nodos obtengan una gran cantidad de nodos que generan conexiones hacia ellos, ya que poseer la capacidad de propagar los bloques oportunamente produce un ciclo virtuoso: Si ya se está bien conectado a la red para ver los bloques oportunamente, otros nodos van a mantenerse conectados, y comenzará a ser visto por los vecinos de aquellos nodos, y en el momento que se conecten, probablemente no desearán desconectarse. De esa forma, un nodo bien conectado atrae más nodos conectados. Se decide levantar la restricción de $d_{in} \leq 80$, que aproximadamente dice que un nodo puede recibir cuatro veces la cantidad de conexiones que las que él hace hacia afuera, y simplemente se deja $d_{in} \leq 640$, esperando que sea suficiente para observar algún efecto. Hay que mencionar que levantar esta restricción es irrealista para un nodo común en el mundo real, pueden existir nodos con capacidades similares, pero para un nodo común, que no es un servidor de

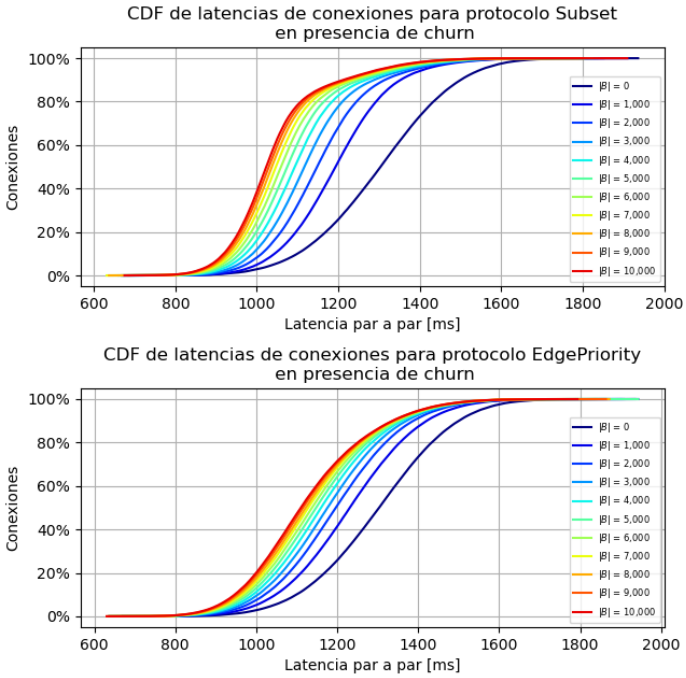


Figura 3.15: Distribución de latencia de aristas para escenario con churn.

alta capacidad, debería haber un límite de conexiones, y además, estar conectado a muchos otros nodos debería deteriorar la capacidad de conectarse. En la simulación solo se quita la restricción, sin añadir un deterioro.

Se entrena EdgePriority para la red con la restricción y la red sin la restricción y se forman histogramas.

Resultados

Primero, se muestra el histograma para la red aleatoria, en 3.18. Se observa que es una campana, centrada alrededor de 20 debido a que al inicio, cada nodo genera 20 conexiones hacia afuera, y el teorema de los saludos permite estimar que en promedio, esa es la cantidad de nodos entrantes. Aun así, hay nodos que varían sus conexiones entrantes desde 5 hasta 40, pero no se alcanza a llegar a la restricción impuesta de $d_{in} \leq 80$.

Después, se observa en histograma para la red entrenada 3.19 que la distribución de los grados cambia radicalmente. Por una parte, hay una basta cantidad de nodos que están por debajo el promedio. luego hay masa acumulada a lo largo de todos los grados entre 20 y 80, y un gran peak en $d_{in} \approx 80$. Como el histograma posee 14 barras, se puede decir que entre $d_{in} = 80 - 80/14 \approx 74$ y $d_{in} = 80$ hay 1100 nodos con esa cantidad de vecinos entrantes, eso equivale al 10 % de la red con altísimo grado.

Ahora, el histograma del último caso revela que solo hay nodos que llegan a tener grado entrante hasta 500 (un 5 % de la red se conecta a ellos), aunque son menos de 10 estos. Por

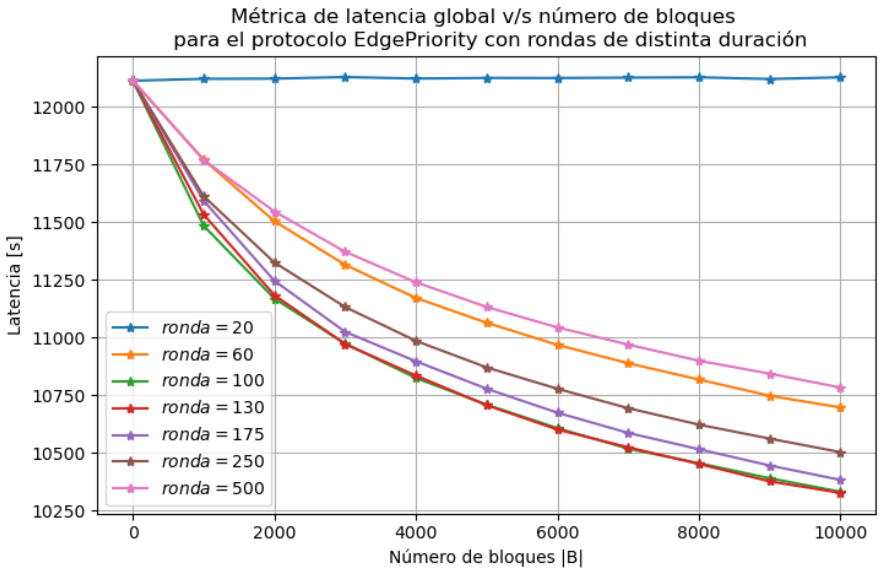


Figura 3.16: Evolución de $\mathbb{E}_b[l(T_b)]$ para rondas de distinta duración.

otra parte, se observa que para todo el rango entre 80 y 500 hay algunos nodos que llenan las barras del histograma; El histograma tiene 14 barras de ancho aproximado 36, y según los datos del histograma, en cada uno de esos anchos hay alrededor de 12 o más nodos con esos grados. Más interesante es qué pasa si se hace un histograma con barras de ancho exponencial, y se muestra el histograma con ambas escalas logarítmicas, como en la figura 3.21, allí se observa una relación afín en la escala logarítmica, de pendiente negativa, aproximadamente -1. Esto significa que la ley de d_{in} es una ley de potencia, de parámetro -1. Es decir que,

$$P_V[d_{in}(v) \in (k-\delta, k+\delta)] \approx O\left(10^4 \cdot \frac{1}{k}\right).$$

Donde P_V es la medida de probabilidad de elegir un nodo de V al azar, uniformemente. Esto revela que la red, con su restricción natural, produce nodos que son tan buenos para comunicar información, que al elevar la restricción, muestran que podrían ser los centros de grandes porciones de la red.

Escenario con red de escala $|V| = 50000$ y churn

Para finalizar, se muestran los resultados de una red similar a la de base, pero con $|V| = 50000$, una latencia l_{uv} dada por

$$l_{uv} := 10000 \cdot \frac{||x_u - x_v||}{\sqrt{20} + 100} \text{ [ms]}.$$

Aquí x_u se incrustan en $[0,1]^{20}$. El largo de las rondas se cambia de 100 a 130 y se mantienen los demás parámetros. Eso sí, se aumenta la cantidad de churn a 200 nodos por ronda diaria.

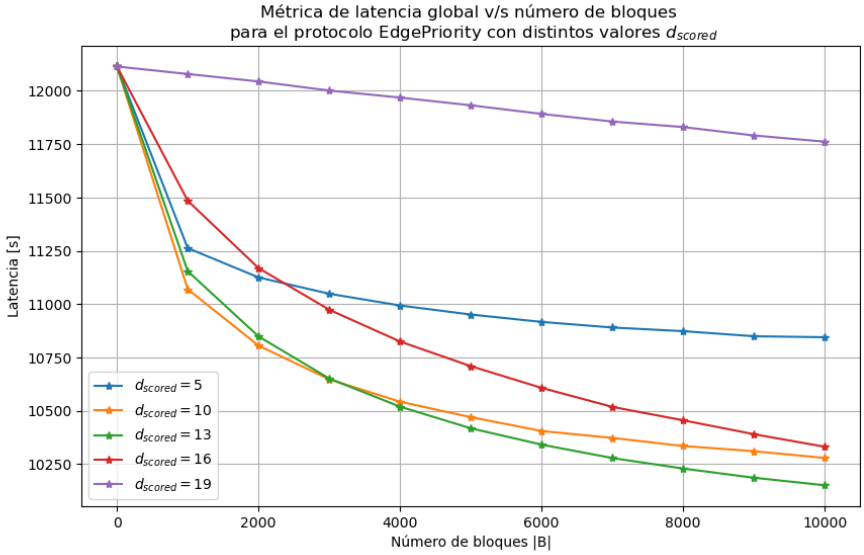


Figura 3.17: Evolución de $\mathbb{E}_b[l(T_b)]$ para distintos d_{scored}.

Resultados para red de escala $|V|$=50000

Los resultados se encuentran en las figuras 3.22, 3.23, 3.24, y son similares a los demás casos. El tamaño de la red y la presencia de churn no afectan al momento de optimizar la red.

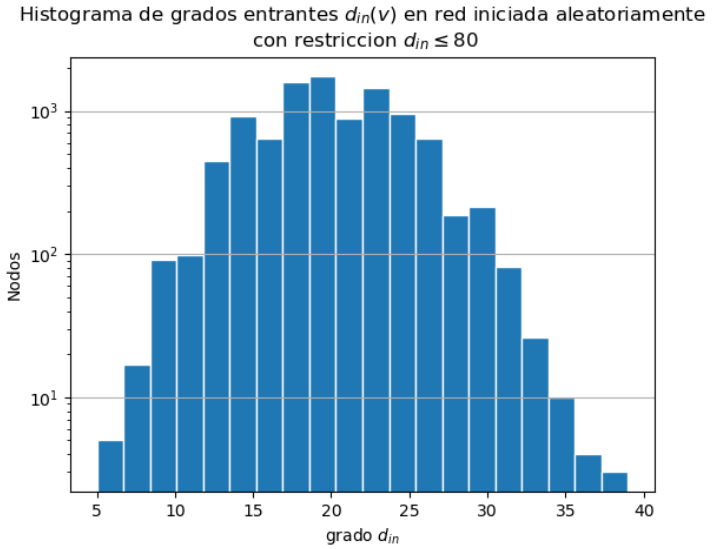


Figura 3.18: Histograma de grado entrante de los nodos, para una red iniciada aleatoriamente.

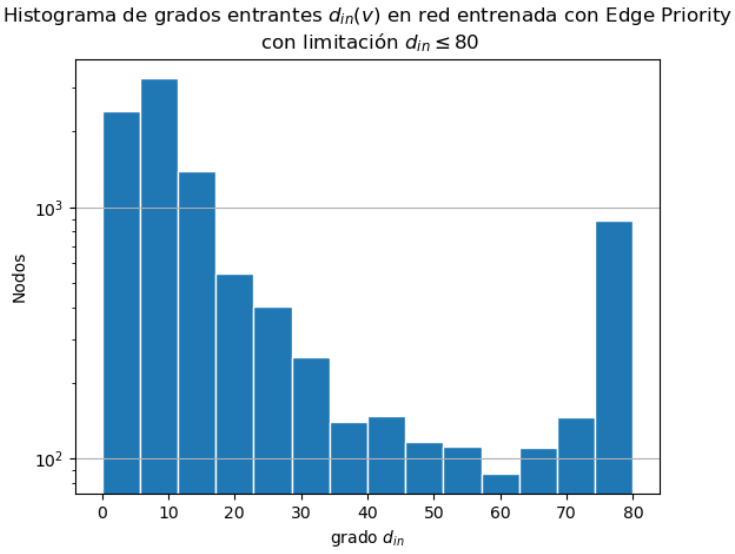


Figura 3.19: Histograma de grado entrante de los nodos, para una red entrenada con EdgePriority por $|B| = 10000$ bloques, bajo la restricción de $d_{in} \leq 80$.

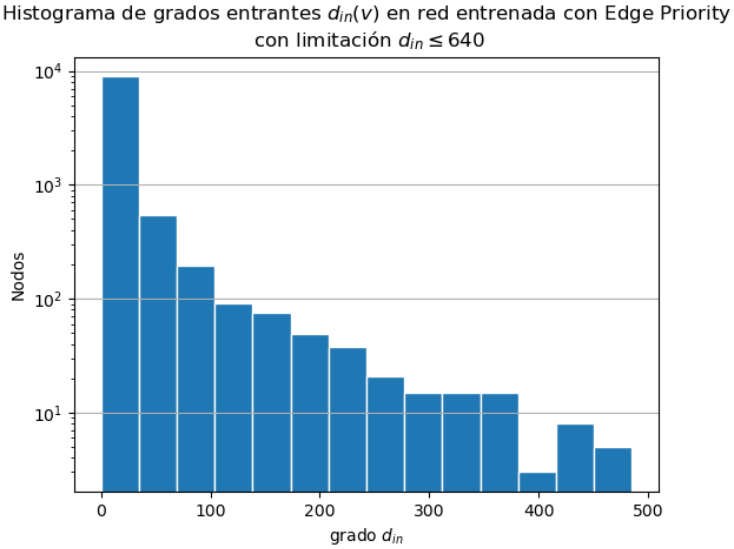


Figura 3.20: Histograma de grado entrante de los nodos, para una red entrenada con EdgePriority por $|B| = 10000$ bloques, bajo la restricción de $d_{in} \leq 640$.

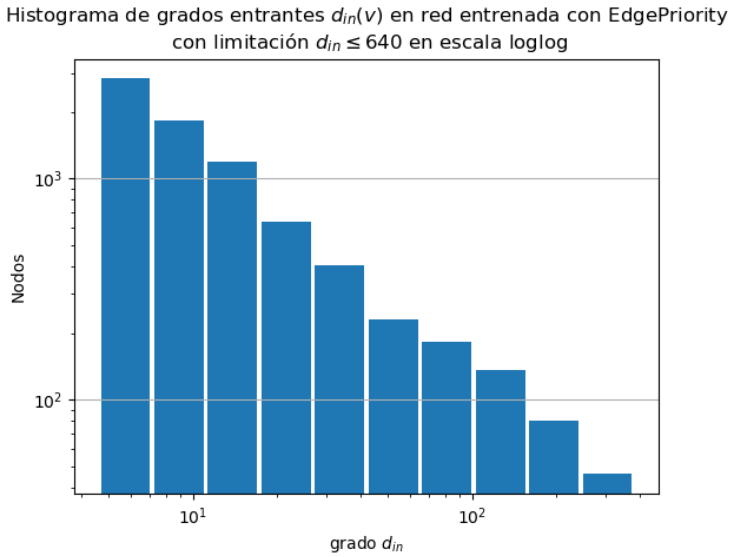


Figura 3.21: Histograma de grado entrante de los nodos, con ambos ejes en escala logarítmica, para una red entrenada con EdgePriority por $|B| = 10000$ bloques, bajo la restricción de $d_{in} \leq 640$.

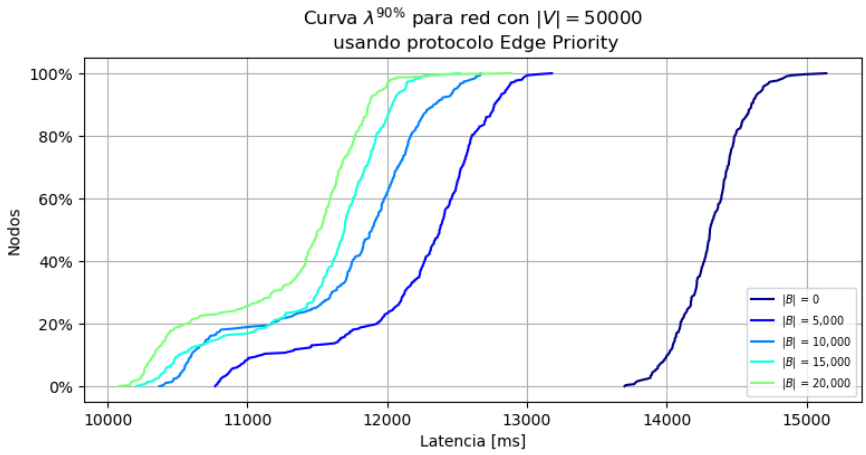


Figura 3.22: Curvas $\lambda^{90\,\%}$ para la simulación con $|V| = 50000$ bajo protocolo EdgePriority.

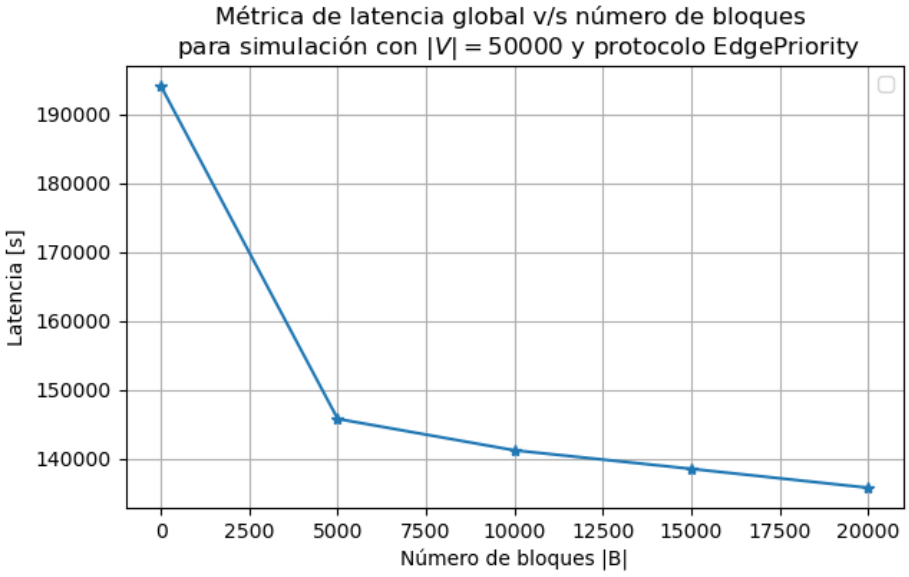


Figura 3.23: Evolución de la latencia global $\mathbb{E}_b[l(T_b)]$ para red de $|V| = 50000$ entrenada con EdgePriority.

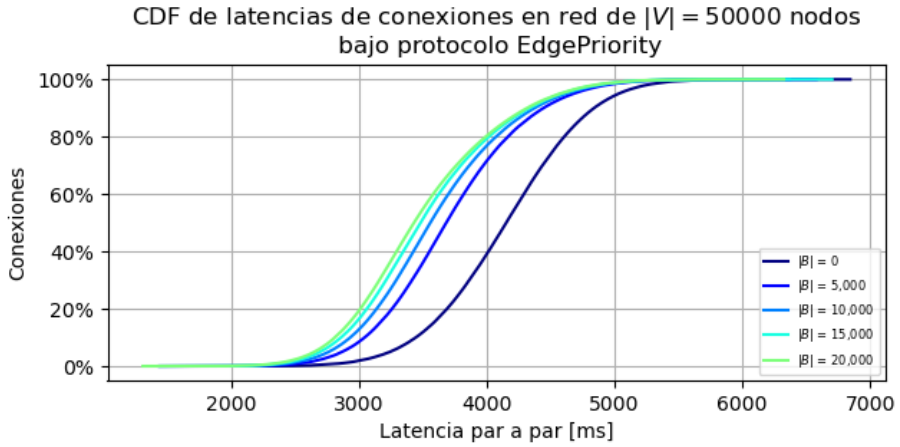


Figura 3.24: Curvas con la distribución de la latencia de las aristas para EdgePriority para simulación a escala $|V| = 50000$.

Capítulo 4

Conclusión

Al terminar este trabajo, se posee la arquitectura de una simulación con características realisticas. De las necesidades planteadas de [38], se logra avanzar en realizar simulaciones a una escala real, con presencia de intermitencia en la actividad de los participantes, y con limitaciones en la visibilidad de otros nodos. Además, el modelo subyacente para simular permite implementar la simulación con algoritmos eficientes, siendo la complejidad lineal en el tiempo que esta se lleva a cabo, y linear-logarítmica en el tamaño de la red. La implementación en código C++ permite realizar simulaciones de decenas de miles de nodos funcionando por cientos de días, en cosa de horas en una computadora personal. Al realizar las simulaciones, se obtienen comportamientos consistentes con los resultados de [38], permitiendo validar independientemente la hipótesis planteada en [38] de que se puede optimizar la topología de red dismiuir la latencia de propagación, pero esta vez a escala real, y también frente a la presencia de churning y visibilidad limitada al momento de generar conexiones. Además, enfrentados a las dificultades que trae evaluar las métricas en tamaños como la escala real, producimos algunas técnicas de medición que pueden ser útiles al momento de evaluar nuevos comportamientos en redes de esta escala. La implementación de la simulación y los métodos desarrollados dejan la puerta abierta para seguir experimentando, e introducir nuevas características para evaluar el rendimiento de los protocolos.

Por otro lado, se avanza en el horizonte teórico, pues a través del modelado de la red fue posible proponer un protocolo nuevo, EdgePriority, basado en heurísticas con interpretaciones de optimizaciones de grafos, que además fue validado con simulaciones. Si bien, no son garantías de optimalidad, el modelo y el nuevo protocolo ofrecen una perspectiva para entender el problema.

Esta nueva herramienta amplía el dominio en que la técnica presentada por [38] está validada, demostrando más evidencia de que la técnica puede efectivamente mejorar la latencia de propagación y la escalabilidad en redes blockchain reales.

Bibliografía

[1] Robert Antwi, James Dzisi Gadze, Eric Tutu Tchao, Axel Sikora, Henry Nunoo-Mensah, Andrew Selasi Agbemenu, Kwame Opunie-Boachie Obour Agyekum, Justice Owusu Agyemang, Dominik Welte, and Eliel Keelson. A survey on network optimization techniques for blockchain systems. *Algorithms*, 15(6), 2022.

[2] John Augustine, Gopal Pandurangan, Peter Robinson, Scott T. Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In Venkatesan Guruswami, editor, *FOCS*, pages 350–369. IEEE Computer Society, 2015.

[3] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 585–602, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] Matthias Bentert and André Nichterlein. Parameterized complexity of diameter. *CoRR*, abs/1802.10048, 2018.

[6] Bitnodes. Bitnodes: estimates the relative size of the bitcoin peer-to-peer network by finding all of its reachable nodes. https://bitnodes.io/nodes/.

[7] Béla Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1):5–34, 2004.

[8] Béla Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2001.

[9] Karl Bringmann and Sebastian Krinninger. A note on hardness of diameter approximation. *Inf. Process. Lett.*, 133:10–15, 2018.

[10] Conrad Burchert, Christian Decker, and Roger Wattenhofer. Scalable funding of bitcoin micropayment channel networks - regular submission. In Paul G. Spirakis and Philippas Tsigas, editors, *SSS*, volume 10616 of *Lecture Notes in Computer Science*, pages 361–377. Springer, 2017.

[11] N. Christofides and J.E. Beasley. A tree search algorithm for the p-median problem. *European Journal of Operational Research*, 10(2):196–204, 1982.

[12] Bram Cohen. Incentives build robustness in BitTorrent. May 2003.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[14] José Alejandro Cornejo Acosta, Jesús García Díaz, Ricardo Menchaca-Méndez, and Rolando Menchaca-Méndez. Solving the capacitated vertex k-center problem through the minimum capacitated dominating set problem. *Mathematics*, 8(9), 2020.

[15] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains - (a position paper). In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2016.

[16] Andrew A. Davidson, Sean Baxter, Michael Garland, and John D. Owens. Work-efficient parallel gpu methods for single-source shortest paths. In *IPDPS*, pages 349–359. IEEE Computer Society, 2014.

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

[18] Anthony Dekker, Hebert Pérez-Rosés, Guillermo Pineda-Villavicencio, and Paul Watters. The maximum degree & diameter-bounded subgraph and its applications. *Journal of Mathematical Modelling and Algorithms*, 11:249–268, 09 2012.

[19] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. *Electron. Notes Discret. Math.*, 38:303–308, 2011.

[20] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. Survey on blockchain networking: Context, state-of-the-art, challenges. *ACM Comput. Surv.*, 54(5), may 2021.

[21] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, jun 2018.

[22] Nikolaos Fountoulakis, Anna Huber, and Konstantinos Panagiotou. Reliable broadcasting in random networks and the effect of density. In *INFOCOM*, pages 2552–2560. IEEE, 2010.

[23] Tobias Friedrich, Thomas Sauerwald, and Alexandre Stauffer. Diameter and broadcast time of random geometric graphs in arbitrary dimensions. *Algorithmica*, 67(1):65–88, 2013.

[24] Alan Frieze and Michał Karoński. *Introduction to Random Graphs*. Cambridge University Press, 2015.

[25] Alan M. Frieze and Wesley Pegden. Traveling in randomly embedded random graphs. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *APPROX-RANDOM*, volume 81 of *LIPIcs*, pages 45:1–45:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[26] Ahmed G. Gad, Diana T. Mosa, Laith Abualigah, and Amr A. Abohany. Emerging trends in blockchain technology and applications: A review and outlook. *Journal of King Saud University - Computer and Information Sciences*, 34(9):6719–6742, 2022.

[27] GeeksForGeeks. Design a data structure that supports insert, delete, search and getrandom in constant time.

[28] Jason Gregory. *Game engine architecture*. Taylor & Francis Ltd., 1 edition, 2009.

[29] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis. Churn in the bitcoin network: Characterization and impact. In *IEEE ICBC*, pages 431–439. IEEE, 2019.

[30] Suhan Jiang and Jie Wu. Approaching an optimal bitcoin mining overlay. *IEEE/ACM Transactions on Networking*, pages 1–14, 2023.

[31] Suhan Jiang and Jie Wu. Approaching an optimal bitcoin mining overlay. *IEEE/ACM Transactions on Networking*, pages 1–14, 2023.

[32] Reiki Kanda and Kazuyuki Shudo. Estimation of data propagation time on the bitcoin network. In *Proceedings of the 15th Asian Internet Engineering Conference*, AINTEC '19, page 47–52, New York, NY, USA, 2019. Association for Computing Machinery.

[33] Lawrence T. Kou, George Markowsky, and Leonard Berman. A fast algorithm for steiner trees. *Acta Inf.*, 15:141–145, 1981.

[34] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Comput.*, 22(4):249–267, 2010.

[35] Fabian Kuhn and Roger Wattenhofer. Dynamic analysis of the arrow distributed protocol. In Phillip B. Gibbons and Micah Adler, editors, *SPAA*, pages 294–301. ACM, 2004.

[36] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, 5th edition, 2009.

[37] Eng Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, 7:72– 93, 04 2006.

[38] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. Perigee: Efficient peer-to-peer network design for blockchains. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 428–437, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Max Mathys, Roland Schmid, Jakub T. Sliwinski, and Roger Wattenhofer. A limitlessly scalable transaction system. In *DPM/CBT@ESORICS*, 2021.

[40] Thomas Moscibroda, Stefan Schmid, and Roger Wattenhofer. Topological implications of selfish neighbor selection in unstructured peer-to-peer networks. *Algorithmica*, 61(2):419–446, 2011.

[41] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. An analytical model for churn process in bitcoin network with ordinary and relay nodes. *Peer Peer Netw. Appl.*, 13(6):1931–1942, 2020.

[42] Saeideh Gholamrezazadeh Motlagh, Jelena V. Misic, and Vojislav B. Misic. Impact of node churn in the bitcoin network. *IEEE Trans. Netw. Sci. Eng.*, 7(3):2104–2113, 2020.

[43] Nakamoto. Bitcoin address manager, 2008.

[44] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.

[45] Amir Nakib, Thibaud Rohmer, El-Ghazali Talbi, and Abdelhamid Nafaa. Dynamic learning optimization algorithm for p2p-vod systems. In *HPCC/SmartCity/DSS*, pages 388–395. IEEE Computer Society, 2017.

[46] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter p2p networks. In *FOCS*, pages 492–499. IEEE Computer Society, 2001.

[47] Christos H. Papadimitriou. Algorithms, games, and the internet. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *STOC*, pages 749–753. ACM, 2001.

[48] M. Posch. *Mastering C++ Multithreading*. Packt Publishing, 2017.

[49] Ou-Yang Rong and Cao Hui. A novel peer selection algorithm to reduce bittorrent-like p2p traffic between networks. In *2009 International Conference on Information Technology and Computer Science*, volume 2, pages 397–401, 2009.

[50] Masoud Sagharichian, Morteza Alipour Langouri, and Hassan Naderi. Calculating exact diameter metric of large static graphs. *J. Univers. Comput. Sci.*, 22(3):302–318, 2016.

[51] Carlos Santiago and Choonhwa Lee. Accelerating message propagation in blockchain networks. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 157–160, 2020.

[52] Carlos Santiago and Choonhwa Lee. Accelerating message propagation in blockchain networks. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 157–160, 2020.

[53] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. Measurement study of peer-to-peer file sharing systems. In *IS&T/SPIE Electronic Imaging*, 2001.

[54] Stefan Schmid and Roger Wattenhofer. Structuring unstructured peer-to-peer networks. In Srinivas Aluru, Manish Parashar, Ramamurthy Badrinath, and Viktor K. Prasanna, editors, *HiPC*, volume 4873 of *Lecture Notes in Computer Science*, pages 432–442. Springer, 2007.

[55] Andreas S. Schulz and Nelson A. Uhan. Sharing supermodular costs. *Operations Research*, 58(4-Part-2):1051–1056, 2010.

[56] Mansi Sood and Osman Yagan. Tight bounds for connectivity of random k-out graphs, 06 2020.

[57] Heike Trautmann, Günter Rudolph, Kathrin Klamroth, Oliver Schütze, Margaret M. Wiecek, Yaochu Jin, and Christian Grimme, editors. *Evolutionary Multi-Criterion Optimization - 9th International Conference, EMO 2017, Münster, Germany, March 19-22, 2017, Proceedings*, volume 10173 of *Lecture Notes in Computer Science*. Springer, 2017.

[58] A. Volgenant. In: Bernhard korte and jens vygen, editors, combinatorial optimization theory and algorithms, second ed., algorithms and combinatorics, vol 21, springer, berlin (2002) isbn 3-540-43154-3 543pp. *Oper. Res. Lett.*, 33(2):216–217, 2005.

[59] Bowen Xue, Yifan Mao, Shaileshh Venkatakrishnan, and Sreeram Kannan. Goldfish: Peer selection using matrix completion in unstructured p2p network, 03 2023.

[60] Yunqi Zhang and Shaileshh Bojja Venkatakrishnan. Kadabra: Adapting kademlia for the decentralized web. *ArXiv*, abs/2210.12858, 2022.

Anexo A

A.1. Características de $[0,1]^d$

Se considera $X_d := [0,1]^d$. La función l_{uv}^d definida para $V \subset X_d$ es

$$l_{uv}^d := \frac{1}{d^{1/p}} ||x - y||_p.$$

donde $p = p(d)$ es alguna función tal que $\lim_d \ln(d)/p(d) = 0$ (puede ser $p(d) = 1 + \ln(d)^{1+\varepsilon}$).

Puede verse que (X_d, l_{uv}^d) tiene una propiedad interesante: la distancia máxima entre puntos de X_d solo se mantiene acotada, pero la cantidad de vecinos que están a una latencia fija tiende a comprimirse (en medida, tiende a cero), o bien, $[0,1]^d$ tiende a dilatarse en comparación. Es fácil notar que $\max_{u,v \in X_d} l_{uv}^d = 1$, con igualdad en $u = 0, v = 1 \in [0,1]^d$.

Para ver lo otro, se define la bola de puntos a latencia $\frac{l}{2}$ de x:

$$n(d,l) = \left\{ y : l_{xy}^d \leq \frac{l}{2} \right\} \subseteq X_d.$$

Se nota que $n(1,1) = [0,1]$. Para fijar ideas se considera $x = \frac{1}{2} \in X_d$ y $l < 1$ (bola centrada en el medio del espacio). A medida que $d \to \infty$, $n(d,l)$ se comporta como otra bola:

$$\begin{aligned} n(d,l) &\approx \left\{ y : ||y - x||_{\lim_d p(d)} \leq \frac{l}{2} \cdot \lim_d d^{1/p(d)} \right\} \\ &\approx \left\{ y : ||y - x||_\infty \leq \frac{l}{2} \cdot 1 \right\}. \end{aligned}$$

y esta última bola, tiene medida de lebesgue $l^d \to 0$, en contraste con la medida de $[0,1]^d$, que permanece constante en 1. Debido a que $p(d)$ necesita crecer extremadamente lento en comparación a d, en las simulaciones se puede usar $p = 2$ para ahorrar flops.

A.2. Modelo de selección aleatoria

El siguiente desarrollo se hizo para entender mejor la dinámica de la estructura R. Primero se realiza un análisis del proceso aleatorio usando la teoría de cadenas de Markov, y después se muestra que esta estructura se puede implementar para inserciones y extracciones de complejidad $O(1)$.

Cabe destacar que no se encontró nada en la literatura estándar de procesos aleatorios.

A.2.1. Propiedades de aleatoriedad

Se tiene una estructura de datos R que recibe una serie de solicitudes $r_v \in \mathbb{N}$ de los nodos $v \in V$, y después, en cada paso extrae aleatoriamente un nodo de $\{u \in V \ : \ r_u > 0\}$ de manera uniforme e independiente, y disminuye r_u en una unidad.

Interesa describir este proceso aleatorio. Se define el espacio de estados $E = \mathbb{N}^V$, se comienza el Proceso en el estado R^0 y se definen los siguientes estados $R^1, R^2, \ldots, R^n, \ldots$ en base a iterar el procedimiento. El proceso $(R^n)_{n\geq 0}$ es una cadena de Markov homogenea con transiciones:

$$\mathbb{P}(r \to r - e_i) = \frac{1}{N(r)} \quad \forall i \in V \ : \ r_i > 0,$$

y estado absorbente $0 \in \mathbb{N}^V$. Aquí $N(r) = \sum_v r_v$. Se pueden definir los tiempos de parada $T_v^k = \text{mín}\{n : R_v^n = k\}$, e interesa particularmente calcular $\tau_x^v = \mathbb{E}[T_v^0 | R^0 = x]$, el número esperado de pasos del proceso para que $R_v^n = 0$.

τ^v es la solución minimal del sistema de ecuaciones:

$$\begin{cases} \tau_x^v = 0 & \forall x : x_v = 0, \\ \tau_x^v = 1 + \sum_y P(x \to y)\tau_y^v & \forall x : x_v > 0. \end{cases} \tag{A.1}$$

Se observa que τ_x^v solo depende de τ_y^v con $y_v = x_v$ o bien $y_v = x_v - 1$, y que $P(x \to y) = \frac{1}{N(x)}$ para estos.

$$\tau_x^v = 1 + \sum_{y:\ x_v = y_v} \frac{1}{N(x)}\tau_y^v + \sum_{y:\ x_v = 1 + y_v} \frac{1}{N(x)}\tau_y^v \tag{A.2}$$

En cualquier caso, $P(x \to y) \neq 0 \implies N(y) = N(x) - 1$. De esto se desprende que la ecuación A.1 tiene una única solución: Toda solución cumple $\tau_x^v = 0$ si $x_v = 0$, y la fórmula determina el valor para $\tau_{x'}^v$ para x' si $N(x') = N(x) + 1$, por lo tanto, usando inducción, la solución está únicamente determinada para todo x con $N(x) = n, n \geq 0$.

Usando que el valor de τ_x^v solo depende de x_v y $N(x)$, Se define S_k^n:

$$S_k^n \coloneqq \tau_x^v, \qquad x_v = k, N(x) = n. \tag{A.3}$$

Se tiene: $S_0^n = 0$, y $S_n^n = n$. Se usa la ecuación A.2 para obtener que si $0 < k < n$,

$$S_k^n = 1 + \frac{n-k}{n} S_k^{n-1} + \frac{k}{n} S_{k-1}^{n-1}$$

Por el mismo argumento usado anteriormente, una ecuación que satisface esto, tiene solución única, y mediante separación de variables $S_k^n = (n+1)B(k)$ se podría desarrollar y obtener que:

$$S_k^n = \frac{k}{k+1}(n+1).$$

Por lo tanto, $\tau_r^v = \frac{r_v}{r_v+1}(N(r)+1)$.

A.2.2. Implementación de la estructura de datos

En esta sección se detalla la implementación concreta de una estructura con la posibilidad de generar inserciones, actualizaciones y eliminaciones de pares (v, r_v), y además, poder generar extracciones aleatorias de probabilidad uniforme sobre el conjunto $\{v \ : \ r_v > 0\}$.

1. Se inicializa una lista L_R vacía. Esta lista admite pares $(v, r_v) \in V \times \mathbb{N}$, representando la identidad de un nodo, y la cantidad de conexiones que está solicitando.
2. Se inicializa un diccionario D_R de tipo `Hashmap` vacío. Este diccionario tiene por llaves, las identidades de nodos $v \in V$ y el valor de la llave v, $D_R[v]$ debería almacenar un índice en la lista, particuarlemente cuál es la ubicación del par (v, r_v) el par está activo en R.

Se recuerdan un par de cosas: Las listas de largo n soportan que los elementos sean accedidos por su índice, un número entre 1 y n. Se puede leer y escribir en un elemento en $O(1)$ si se conoce su índice de antemano. Además, las listas pueden agregar y remover un elemento al final de la lista en $O(1)$. Por otra parte, los hashmaps se pueden implementar para hacer búsquedas, inserciones y actualizaciones en tipo $O(1)$ amortizado. También, que para generar un número aleatorio entero uniforme en un intervalo, basta un procedimiento $O(1)$. Ahora se definen las operaciones:

- Actualización (v, r_v) con $r_v > 0$: Se chequea si D_R tiene la llave v.
  - Si no, entonces se añade el par (v, r_v) a la lista, produciendo que la lista tenga n elementos. Luego, se escribe $D_R[v] \leftarrow n$.
  - Si la tiene, entonces $D_R[v]$ tiene la ubicación del par (v, r) con v, que ahora se sustituye por el nuevo valor $L_R[D_R[v]] \leftarrow (v, r_v)$.
- La actualización en R sirve para hacer inserción. Además, se nota solo se usaron accesos por índice, reescrituras, y búsquedas por hashmap, en total $O(1)$ amortizado.
- Eliminación de v: Si D_R no posee la llave v, entonces no hay nada que hacer. Si D_R posee la llave, entonces se sabe que $L_R[D_R[v]] = (v, r_v)$.

  - Si $D_R[V]$ está al final de la lista, entonces (v, r_v) se remueve de la lista y remueve la información de v en D_R.
  - Si $D_R[v]$ no está en el fin de la lista, no se puede simplemente remover, hay que hacer un paso más: Se busca el elemento al final de la lista (u, r_u) y se pone en $L_R[D_R[v]] \leftarrow (u, r_u)$, y enseguida, se actualiza en el diccionario $D_R[u] \leftarrow D_R[v]$. De esta manera, el diccionarrio sigue conteniendo el índice de u adecuadamente en la lista. Para finalizar, se remueve el último elemento de la lista L_R y se remueve el elemento de llave v en D_R, así, toda la información relativa a v ha sido eliminada de R.

- Nuevamente, todo eso solo utiliza operaciones $O(1)$.

- Extracción aleatoria: a partir de $n =$"largo de la lista L_R", se genera una selección x al azar y uniforme entre 1 y n, y con ese valor se selecciona un par $(v, r_v) = L_R[x]$, obteniendo la muestra aleatoria deseada. Si $r_v - 1 > 0$, entonces simplemente se actualiza el valor $r_v \leftarrow r_v - 1$. Si $r_v - 1 = 0$, entonces se puede hacer directamente el procedimiento de eliminación.

Eso concluye la especificación de la estructura de datos. Del procedimiento de actualización y eliminación, se puede notar que la condición de que el diccionario tenga el índice correcto siempre se cumple adecuadamente (es un invariante). También se desprende que de los elementos (v, r_v) de la lista, cada v puede aparecer una sola vez, y por lo tanto, la selección uniforme descrita es uniforme sobre $\{v : r_v > 0\}$.

A.3. Método de la transformada inversa

Se busca simular una variable aleatoria $X : \omega \to V$ de acuerdo a la ley de probabilidad $\mathbb{P}[X = v] = f_v$.

Primero, se ordena arbitrariamente V de acuerdo a un orden $(V, \leq)$. A partir de este orden, se pre-computa una función distribución $F_X : V \to [0, 1]$ usando la ley f_v:

$$F_X(v) := \sum_{u \in V} 1_{u \leq v} f_u.$$

Luego, $F_X : V \to [0, 1]$ es una función monótona no decreciente con respecto de V. Ahora se necesita añadir un nuevo orden sobre V: $u \leq_X v$ es el órden lexicográfico según primero ver $F_X(u) \leq F_X(v)$, y en caso de que coinciden, se elige como menor, el que tenga $f_x > 0$. Se vuelve a reordenar V según $(V, \leq_X)$. A partir de esto, se construye la "inversa generalizada" $F_X^{-1} : [0, 1] \to V$, seleccionando el primer elemento v según $\leq_X$ tal que $F_X^{-1}(v) \geq U$. Para generar una muestra de X, se genera una variable aleatoria uniforme $U : \omega \longrightarrow [0, 1]$.

El procedimiento de pre-cómputo necesita de ordenar V y de pre-computar F_X sobre los valores de V. Dependiendo el ordenamiento aplicado, esto puede tomar a lo más $O(|V| \log |V|)$ haciendo un cálculo eficiente de $F_x(v)$ en base a valores previamente calculados y una lista

ordenada de V de acuerdo a $\leq$. Después de ello, como V está ordenado, y F_X es una función monótona, por lo que se puede construir un árbol de búsqueda binaria para seleccionar el elemento buscado, tardando a lo más $O(\log |V|)$. Una forma concreta es crear un arreglo indexado por V que anota los valores $F_X(v)$, y aquí el árbol de búsqueda binaria se construye recorriendo sub-arreglos.

Se evalúa si esta función es suficiente para generar variables aleatorias de acuerdo a la ley f_v. Primero notar que si $u \in V$ es un nodo tal que $f_u > 0$ (es decir, tiene probabilidad de ser generado), entonces $F_X(v) < F_X(u)$ para todo $v < u$, y por lo tanto, el método lo puede seleccionar; y al revés, si $f_u = 0$, entonces existirá un elemento $v \in V$ tal que $F_X(v) = F_X(u)$ y $v \leq_X u$, por lo que el método eligirá el elemento con $f_v > 0$ en vez de u. La única excepción es cuando u de $f_u = 0$ es el primer elemento, pero en este caso $F_X(u) = 0$, es decir, para que u sea elegido, sería necesario que $U \leq 0$, evento que tiene probabilidad nula. Para terminar, La probabilidad de elegir un elemento v corresponde a la medida $F_X(v)$ menos la de su antecesor $F_X(v^-)$: se sabe de la definición, que esto es f_v.

En resumen, el método mencionado necesita de un precómputo $O(|V| \log |V|)$, cada muestra toma $O(\log |V|)$ y tiene las propiedades de probabilidad requeridas.

Anexo B

B.1. Cómo usar la simulación

B.1.1. Compilación

Para realizar una simulación, se configura un archivo de código en C++, con extensión `.cpp` que contenga los detalles a simular en la función `main`. Este archivo además debe usar las librerías con el código de fuente de la simulación, que se encuentran en una carpeta de nombre `src`. En los experimentos facilitados, hay varios archivos `simulacion.cpp` de ejemplo. Para compilarlos, basta ejecutar el programa `make` en la carpeta con dicho archivo, o bien compilar con `g++ simulacion.cpp -o simulacion -I/<SRC>` donde `<SRC>` es la carpeta con las librerías de la simulación. Solo es necesario un compilador como `g++` con una versión `c++17`. Una vez lista la compilación, solo basta ejecutar el archivo que resulta de la compilación.

B.1.2. Montaje de una simulación

Las partes de la simulación se ajustan a lo detallado en el modelo de este texto, y en el archivo `.cpp` está cada objeto detallado mediante un comentario explicativo. La función que ejecuta el bucle principal es `run`, o bien, una versión más flexible, `loop`.

B.1.3. Ejecución

Durante la ejecución del programa, se crearán archivos y carpetas en virtud de las mediciones explicitadas en el archivo `.cpp`. Por lo mismo, se recomienda ejecutar el programa en una carpeta vacía, donde no sea molesto la creación de nuevos archivos y carpetas.

B.1.4. Scripts para procesa salidas y hacer gráficos

Junto con la simulación, se incluyen algunos archivos `.py` que se pueden usar para generar los gráficos. Para usarlos, se tiene que ubicar con una consola de comandos en una carpeta *superior* a la carpeta donde se ejecutó la simulación, y se debe ejecutar el comando `python3 <script><nombre de la carpeta(s)>`, donde `<script>` es el archivo de Python en cuestión.

Anexo C

En la próxima página se muestra una figura adicional.

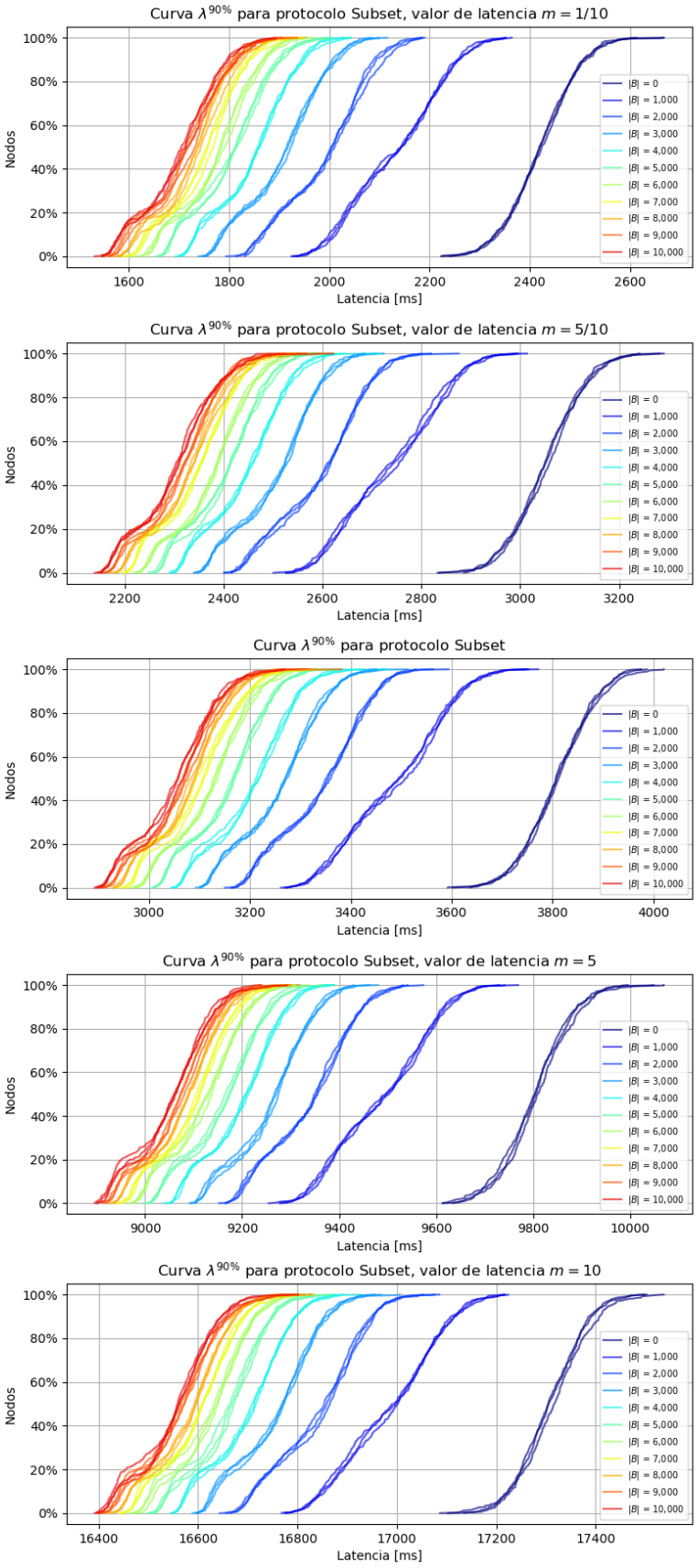


Figura C.1: Curvas $\lambda^{90\,\%}$ para EdgePriority escenarios con variación en la constante de latencia. $r \in \{1/10, 5/10, 1, 5, 10\}$.