De Snowflake a Avalanche: Una Nueva Familia de Protocolos Metaestables para Criptomonedas

Team Rocket[†]
t-rocket@protonmail.com
Traducido al español por Azem (azem@azem.es)

Revisión original: 05/16/2018 21:51:26 UTC

Resumen

Este artículo introduce una nueva familia de protocolos sin líder tolerante a fallos bizantinos (*leaderless Byzantine fault tolerance protocols*), construida sobre un mecanismo metaestable. Estos protocolos proporcionan una fuerte garantía de seguridad probabilística en presencia de adversarios bizantinos, mientras que su naturaleza concurrente les permite lograr alto rendimiento y escalabilidad. A diferencia de blockchains que dependen de una prueba de trabajo (*proof-of-work*), son quiescentes y verdes. Sorprendentemente, a diferencia de los protocolos de consenso tradicionales que requieren comunicación $O(n^2)$, su complejidad de comunicación varía de $O(kn \log n)$ a $O\ (kn)$ para algún parámetro de seguridad $k \ll n$.

El artículo describe la familia de protocolos, la ejemplifica en tres protocolos separados, analiza sus garantías y describe cómo se pueden utilizar para construir el núcleo de un sistema de pago electrónico a escala de Internet. El sistema se evalúa en una implementación a gran escala. Los experimentos demuestran que el sistema puede lograr un alto rendimiento (1300 tps), proporcionar una latencia de confirmación baja (4 segundos) y escala bien en comparación con los sistemas existentes que ofrecen similares funcionalidades. Para nuestra implementación y configuración, el cuello de botella del sistema se encuentra en la verificación de la transacción.

1. Introducción

El hecho de lograr un acuerdo entre un conjunto de hosts distribuidos se encuentra en el núcleo de innumerables aplicaciones, que van desde servicios a escala de Internet que atienden a miles de millones de personas [12, 30] a criptomonedas por valor de miles de millones de dólares [1]. Hasta la fecha, ha habido dos familias principales de soluciones a este problema. Los protocolos de consenso tradicionales se basan en la comunicación de todos-a-todos *(all-to-all)* para asegurar que todos los nodos correctos lleguen a las mismas decisiones con absoluta certeza. Debido a que requieren una sobrecarga de comunicación cuadrática y un conocimiento preciso de la membresía, han sido difíciles de escalar a un gran número de participantes.

Por otro lado, los protocolos de consenso de Nakamoto [9, 24, 25, 34, 40–43, 49–51] se han vuelto populares con el auge de Bitcoin. Estos protocolos proporcionan una garantía de seguridad probabilística: las decisiones de consenso de Nakamoto pueden revertirse con cierta probabilidad ϵ. Un parámetro de protocolo permite que esta probabilidad se vuelva arbitrariamente pequeña, lo que permite que los sistemas financieros de alto valor se construyan sobre esta base. Esta familia es un ajuste natural para configuraciones abiertas y sin permisos donde cualquier nodo puede unirse al sistema en cualquier momento. Sin embargo, estos protocolos son bastante costosos, derrochadores, y limitados en el rendimiento. Por construcción, estos protocolos no pueden detenerse: su seguridad se basa en la participación constante de los mineros, incluso cuando no hay decisiones que tomar. Bitcoin actualmente consume alrededor 63.49 TWh/año [22], aproximadamente el doble que en toda Dinamarca [15]. Además, estos protocolos tienen un

cuello de botella de escalabilidad inherente que es difícil de superar mediante una simple reparametrización [18].

Este artículo introduce una nueva familia de protocolos de consenso. Inspirada por los algoritmos de susurro *(gossip algorithms)*, esta nueva familia obtiene su seguridad a través de un mecanismo deliberadamente metaestable. Específicamente, el sistema opera muestreando repetidamente la red al azar y dirigiendo los nodos correctos hacia el mismo resultado. El análisis muestra que la metaestabilidad es una técnica poderosa, aunque no universal: puede mover una red grande a un estado irreversible rápidamente, aunque no siempre se garantiza que lo haga.

Similar al consenso de Nakamoto, esta nueva familia de protocolos proporciona una garantía de seguridad probabilística, utilizando un parámetro de seguridad ajustable que puede hacer que la posibilidad de un error de consenso sea arbitrariamente pequeña. A diferencia del consenso de Nakamoto, los protocolos son verdes, quiescentes y eficientes; no se basan en la prueba de trabajo [23] y no consumen energía cuando no hay decisiones que tomar. La eficiencia de los protocolos proviene de dos fuentes: requieren costos de comunicación que van desde $O(kn \log n)$ a $O(kn)$ para algún pequeño parámetro de seguridad k, y establecen sólo un orden parcial entre las transacciones dependientes.

Esta combinación de las mejores características del consenso tradicional y de Nakamoto implica un compromiso significativo: vitalidad *(liveness)* para las transacciones en conflicto. En concreto, la nueva familia garantiza la vitalidad solo para transacciones virtuosas, es decir, para aquellas emitidas por un cliente correcto y que por tanto garantiza que no habrá conflicto con otras transacciones. En un entorno de criptomonedas, las firmas criptográficas obligan a que solo el propietario de una clave pueda crear una transacción que gaste una moneda en particular. Dado que los clientes correctos siguen el protocolo según lo prescrito, se les garantiza tanto la seguridad como la vitalidad. En contraste, los protocolos no garantizan la vitalidad útil de las transacciones fraudulentas, enviadas por clientes bizantinos, que entran en conflicto entre sí. Estas decisiones pueden detenerse en la red, pero no tienen un impacto de seguridad en las transacciones virtuosas. Mostramos que esta es un compromiso sensato, y que el sistema resultante es suficiente para construir sistemas de pago complejos.

En general, este documento hace una contribución significativa: una nueva familia de protocolos de consenso adecuados para las criptomonedas, basada en el muestreo aleatorio y la decisión metaestable. Los protocolos proporcionan una fuerte garantía de seguridad probabilística y una garantía de vitalidad para los clientes correctos.

La siguiente sección proporciona una intuición acerca de los nuevos protocolos, la Sección 3 proporciona pruebas de seguridad y vitalidad, la Sección 4 describe un sistema de pago similar a Bitcoin, la Sección 5 evalúa el sistema, la Sección 6 presenta el trabajo relacionado y, finalmente, la Sección 7 resume nuestras contribuciones.

2. Enfoque

Comenzamos con un protocolo no bizantino, Slush, y progresivamente construimos Snowflake, Snowball y Avalanche, todos basados en el mismo mecanismo metaestable común. Aunque proporcionamos definiciones para los protocolos, diferimos su análisis formal y las pruebas de sus propiedades a la siguiente sección.

En general, esta familia de protocolos logra sus propiedades haciendo trampa humildemente de tres maneras diferentes. Primero, inspirándonos en Bitcoin, adoptamos un margen de seguridad que es probabilístico. Este margen probabilístico es indistinguible de los márgenes de seguridad tradicionales en la práctica, ya que las elecciones apropiadamente pequeñas de ϵ pueden hacer que sea prácticamente inviable un fallo en el consenso, menos frecuente que los fallos de procesamiento de las CPUs o las colisiones de hashes. En segundo lugar, en vez de un solo modelo de máquina de estado replicada (*replicated state machine*, RSM), donde el sistema determina una secuencia de transacciones totalmente ordenadas T_0, T_1, T_2, . . . emitido por cualquier cliente, adoptamos un modelo de consenso paralelo con clientes autenticados, donde cada cliente interactúa de manera independiente con sus propios RSM y, opcionalmente, transfiere la propiedad de su RSM a otro cliente. El sistema establece solo un orden parcial entre transacciones

dependientes. Por último, no ofrecemos ninguna garantía de vitalidad para los clientes que se comportan de forma indebida, pero nos aseguramos de que los clientes con buen comportamiento eventualmente reciban servicio. Estas técnicas, en conjunto, permiten al sistema implementar una criptomoneda útil similar a Bitcoin, con un rendimiento y escalabilidad drásticamente mejores.

2.1 Modelo, Objetivos, Modelo de Amenaza

Asumimos una colección de nodos, $\mathcal{N}$, compuesta de nodos correctos $\mathcal{C}$ y nodos bizantinos $\mathcal{B}$, donde $n = |\mathcal{N}|$. Adoptamos lo que comúnmente se conoce como el modelo de salida de transacciones no gastadas (UTXO) de Bitcoin. En este modelo, los *clientes* se autentican y emiten transacciones firmadas criptográficamente que consumen completamente un UTXO existente y emiten nuevos UTXO. A diferencia de los nodos, los clientes no participan en el proceso de decisión, sino que solo suministran transacciones a los *nodos* que ejecutan el protocolo de consenso. Dos transacciones entran en *conflicto* si consumen el mismo UTXO y producen resultados diferentes. Los clientes correctos nunca emiten transacciones en conflicto, ni es posible que los clientes bizantinos forjen conflictos con transacciones emitidas por clientes correctos. Sin embargo, los clientes bizantinos pueden emitir varias transacciones que entren en conflicto entre sí, y los clientes correctos solo pueden consumir una de esas transacciones. Así pues, el objetivo de nuestra familia de protocolos de consenso es el de *aceptar* un conjunto de transacciones no conflictivas en presencia de un comportamiento bizantino. Cada cliente puede considerarse como una máquina de estado replicada cuyas transiciones están definidas por una lista totalmente ordenada de transacciones aceptadas.

Nuestra familia de protocolos ofrece las siguientes garantías con alta probabilidad:

P1. Seguridad. No hay dos nodos correctos que acepten transacciones en conflicto.

P2. Vitalidad. Cualquier transacción emitida por un cliente correcto (también conocida como transacción virtuosa) eventualmente será aceptada por cada nodo correcto.

En lugar de un acuerdo incondicional, nuestro enfoque garantiza que la seguridad se mantendrá con una probabilidad $1 - \epsilon$, donde la elección del parámetro de seguridad ϵ está bajo el control del diseñador del sistema y las aplicaciones.

Asumimos un poderoso adversario adaptativo capaz de observar el estado interno y las comunicaciones de cada nodo en la red, pero no capaz de interferir con la comunicación entre los nodos correctos. Nuestro análisis asume una red síncrona, mientras que nuestra implementación y evaluación se realiza en una configuración parcialmente síncrona. Conjeturamos que nuestros resultados se mantienen en redes parcialmente síncrona, pero dejamos la prueba de ello como trabajo futuro. No asumimos que todos los miembros conozcan a todos los miembros de la red, sino que pueden tener temporalmente algunas discrepancias en la vista de la red. Asumimos un sistema de arranque seguro, similar al de Bitcoin, que permite que un nodo se conecte con suficientes nodos correctos para adquirir una visión estadísticamente imparcial de la red. No asumimos una infraestructura de clave pública (*Public Key Infrastructure*, PKI). Realizamos suposiciones criptográficas estándar relacionadas con firmas de clave pública y funciones hash.

2.2 Slush: Introduciendo Metaestabilidad

El núcleo de nuestro enfoque es un protocolo de consenso con un único decreto, inspirado en protocolos de epidemia o susurros.

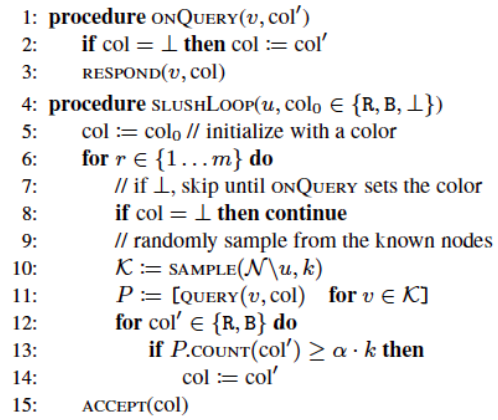

Figura 1. Protocolo Slush. Se omiten los tiempos de espera por legibilidad.

El protocolo metaestable más simple, Slush, es la base de esta familia, que se muestra en la Figura 1. Slush no es tolerante a los fallos bizantinos, pero sirve como ilustración para los protocolos *tolerantes a errores bizantinos* (BFT) que siguen. Para facilitar la exposición, describiremos el funcionamiento de Slush utilizando una decisión entre dos colores en conflicto, rojo y azul.

En Slush, un nodo comienza inicialmente en un estado incoloro. Al recibir una transacción de un cliente, un nodo incoloro actualiza su propio color al que se lleva en la transacción e inicia una consulta. Para realizar una consulta, un nodo selecciona una muestra pequeña, de tamaño constante (k) de la red de manera uniforme y aleatoria, y envía un mensaje de consulta. Al recibir una consulta, un nodo incoloro adopta el color de la consulta, responde con ese color e inicia su propia consulta, mientras que un nodo coloreado simplemente responde con su color actual. Si las respuestas k no se reciben dentro de un límite de tiempo, el nodo selecciona una muestra adicional de los nodos restantes de manera uniforme y las consulta hasta que recopila todas las respuestas. Una vez que el nodo de consulta recopila k respuestas, comprueba si una fracción $\geq \alpha k$ es del mismo color, donde $\alpha > 0.5$ es un parámetro del protocolo. Si se cumple el umbral αk y el color de la muestra difiere del propio color del nodo, el nodo se cambia a ese color. Luego regresa al paso de consulta e inicia la siguiente ronda de consultas, para un total de m rondas. Finalmente, el nodo decide el color con el que terminó en el momento m.

Este protocolo simple tiene algunas propiedades curiosas. Primero, casi *no tiene memoria*: un nodo no retiene ningún estado entre rondas que no sea su color actual, y en particular no mantiene un historial de interacciones con otros. En segundo lugar, a diferencia de los protocolos de consenso tradicionales que consultan a cada participante, cada ronda implica muestrear solo una pequeña porción de la red de tamaño constante al azar. En tercer lugar, incluso si la red comienza en el estado metaestable de una división 50/50 rojo-azul, las perturbaciones aleatorias en el muestreo harán que un color adquiera una ligera ventaja y luego los muestreos repetidos aumentarán y amplificarán ese desequilibrio. Finalmente, si m se elige lo suficientemente alto, Slush se asegura de que todos los nodos tengan un color idéntico con una alta probabilidad. Cada nodo tiene una sobrecarga de comunicación constante y predecible por ronda, y mostraremos que m crece logarítmicamente con n.

Si Slush se implementa en una red con nodos bizantinos, el adversario puede interferir con las decisiones. En particular, si los nodos correctos desarrollan una preferencia por un color, el adversario puede intentar voltear los nodos al contrario para mantener la red en equilibrio. El protocolo Slush se presta al análisis, pero no proporciona, por sí solo, una garantía de seguridad sólida en presencia de nodos bizantinos, porque los nodos carecen de estado. Abordamos esto en nuestro primer protocolo BFT.

2.3 Snowflake: BFT

Snowflake aumenta a Slush con un único contador que captura la fuerza de la convicción de un nodo en su color actual. Este contador por nodo almacena cuántas muestras consecutivas de la

red han obtenido el mismo color. Un nodo acepta el color actual cuando su contador excede β, otro parámetro de seguridad. La figura 2 muestra el protocolo modificado, que incluye las siguientes modificaciones:

1. Cada nodo mantiene un contador *cnt;*
2. Tras cada cambio de color, el nodo restablece *cnt* a 0;
3. Tras cada consulta exitosa que produce $\geq \alpha k$ respuestas para el mismo color que el nodo, el nodo aumenta *cnt.*

Cuando el protocolo está correctamente parametrizado para un umbral dado de nodos bizantinos y una garantía deseada ϵ, puede garantizar tanto la seguridad (P1) como la vitalidad (P2). Como mostraremos más adelante, existe un punto de cambio de fase después del cual los nodos correctos tienen más probabilidades de tender hacia una decisión que un estado bivalente. Además, existe un punto de no retorno después del cual una decisión es inevitable. Los nodos bizantinos pierden el control más allá del cambio de fase, y los nodos correctos comienzan a pasar más allá del punto de no retorno, para adoptar el mismo color con una alta probabilidad.

```
1:  procedure SNOWFLAKELOOP(u, col_0 ∈ {R, B, ⊥})
2:      col := col_0, cnt := 0
3:      while undecided do
4:          if col = ⊥ then continue
5:          K := SAMPLE(N\u, k)
6:          P := [QUERY(v, col)  for v ∈ K]
7:          for col' ∈ {R, B} do
8:              if P.COUNT(col') ≥ α · k then
9:                  if col' ≠ col then
10:                     col := col', cnt := 0
11:                 else
12:                     if ++cnt > β then ACCEPT(col)
```

Figura 2: Snowflake.

2.4 Snowball: Añadiendo Confianza

La noción de estado de Snowflake es efímera: el contador se reinicia con cada cambio de color. Si bien, teóricamente, el protocolo puede ofrecer fuertes garantías con un estado mínimo, ahora mejoraremos el protocolo para que sea más difícil de atacar incorporando una noción más permanente de creencia.

```
1:  procedure SNOWBALLLOOP(u, col_0 ∈ {R, B, ⊥})
2:      col := col_0, lastcol := col_0, cnt := 0
3:      d[R] := 0, d[B] := 0
4:      while undecided do
5:          if col = ⊥ then continue
6:          K := SAMPLE(N\u, k)
7:          P := [QUERY(v, col)  for v ∈ K]
8:          for col' ∈ {R, B} do
9:              if P.COUNT(col') ≥ α · k then
10:                 d[col']++
11:                 if d[col'] > d[col] then
12:                     col := col'
13:                 if col' ≠ lastcol then
14:                     lastcol := col', cnt := 0
15:                 else
16:                     if ++cnt > β then ACCEPT(col)
```

Figura 3: Snowball.

```
1: procedure AVALANCHELOOP
2:     while true do
3:         find T that satisfies T ∈ 𝒯 ∧ T ∉ 𝒬
4:         𝒦 := SAMPLE(𝒩\u, k)
5:         P := Σ_{v∈𝒦} QUERY(v, T)
6:         if P ≥ α · k then
7:             c_T := 1
8:             // update the preference for ancestors
9:             for T' ∈ 𝒯 : T' ←* T do
10:                if d(T') > d(𝒫_{T'}.pref) then
11:                    𝒫_{T'}.pref := T'
12:                if T' ≠ 𝒫_{T'}.last then
13:                    𝒫_{T'}.last := T', 𝒫_{T'}.cnt := 0
14:                else
15:                    ++𝒫_{T'}.cnt
16:        // otherwise, c_T remains 0 forever
17:        𝒬 := 𝒬 ∪ {T}   // mark T as queried
```

Figura 4: Avalanche: el bucle principal.

Snowball aumenta a Snowflake con contadores de confianza que capturan la cantidad de consultas que han generado un resultado de umbral para su color correspondiente (Figura 3). Un nodo decide un color si, durante un cierto número de consultas consecutivas, su confianza para ese color supera la de otros colores. Las diferencias entre Snowflake y Snowball son las siguientes:

1. En cada consulta exitosa, el nodo incrementa su contador de confianza para ese color.

2. Un nodo cambia de color cuando la confianza en su color actual se vuelve más baja que el valor de confianza del nuevo color.

Snowball no solo es más difícil de atacar que Snowflake, sino que se puede generalizar más fácilmente a protocolos de múltiples decretos.

2.5 Avalanche: Añadiendo DAG

Avalanche, nuestro protocolo final, generaliza Snowball y mantiene un gráfico acíclico dirigido (*Directed Acyclic Graph*, DAG) dinámico de solo anexión de todas las transacciones conocidas. El DAG tiene un único sumidero que es *el vértice génesis*. Mantener un DAG proporciona dos beneficios significativos. Primero, mejora la eficiencia, porque un solo voto en un vértice DAG vota implícitamente por todas las transacciones en el camino hacia el vértice génesis. Segundo, también mejora la seguridad, porque el DAG entrelaza el destino de las transacciones, similar a la cadena de bloques de Bitcoin. Esto hace que las decisiones anteriores sean difíciles de deshacer sin la aprobación de los nodos correctos.

Cuando un cliente crea una transacción, nombra a uno o más *padres*, que se incluyen inseparablemente en la transacción y forman los bordes del DAG. Las relaciones padre-hijo codificadas en el DAG pueden, pero no es necesario, corresponder a dependencias específicas de la aplicación; por ejemplo, una transacción secundaria no necesita gastar o tener ninguna relación con los fondos recibidos en la transacción principal. Usamos el término *conjunto de antepasados* para referirnos a todas las transacciones accesibles a través de los bordes primarios en el historial, y la *progenie* para referirnos a todas las transacciones secundarias y sus descendientes.

El desafío principal en el mantenimiento del DAG es elegir entre *transacciones en conflicto*. La noción de conflicto es definida por la aplicación y transitiva, formando una relación de equivalencia. En nuestra aplicación de criptomoneda, las transacciones que gastan los mismos fondos *(doble gasto)* entran en conflicto y forman *un conjunto de conflictos*, de los cuales solo uno puede ser aceptado. Tenga en cuenta que el conjunto de conflictos de una transacción virtuosa es siempre un singleton. Las zonas sombreadas en la Figura 7 indican conjuntos de conflictos.

Avalanche expresa una instancia de Snowball para cada conjunto de conflictos. Mientras que Snowball usa consultas repetidas y múltiples contadores para capturar la confianza acumulada en las transacciones (colores) en conflicto, Avalanche aprovecha la estructura DAG y utiliza la progenie de una transacción. Específicamente, cuando se consulta una transacción T, todas las transacciones a las que se puede acceder desde T siguiendo los bordes del DAG forman parte

implícitamente de la consulta. Un nodo sólo responderá positivamente a la consulta si T y su ascendencia completa son actualmente la opción preferida en sus respectivos conjuntos de conflictos. Si más de un umbral de respondedores vota positivamente, se dice que la transacción recoge una *ficha*, $c_{uT} = 1$, de otra manera, $c_{uT} = 0$. Los nodos luego calculan su *confianza* como la suma de los valores de las fichas en la progenie de esa transacción. Los nodos consultan una transacción solo una vez y se basan en nuevos vértices y fichas, sumados a la progenie, para construir su confianza. Los vínculos se rompen por una preferencia inicial para las transacciones vistas por primera vez.

2.6 Avalanche: Especificación

Cada nodo correcto u realiza un seguimiento de todas las transacciones que ha aprendido en conjunto $T_{u,}$ particionado en conjuntos de conflictos mutuamente excluyentes $\mathcal{P}_{T,} T \in \mathcal{T}_{u.}$ Ya que los conflictos son transitivos, T_i y T_j están en conflicto, entonces $\mathcal{P}_{T_i} = \mathcal{P}_{T_j.}$

Escribimos $T' \leftarrow T$ si T tiene un borde parental para la transacción T', la relación "$\overset{*}{\leftarrow}$" es su cierre transitivo reflexivo, que indica un camino de T a T'. Cada nodo u puede computar un valor confidencial, $d_u(T)$ desde la progenie como sigue:

$$d_u(T) = \sum_{T' \in \mathcal{T}_u, T \overset{*}{\leftarrow} T'} c_{uT'}.$$

Además, mantiene su propia lista local de nodos conocidos. $\mathcal{N}_u \subseteq \mathcal{N}$ que componen el sistema. Para simplificar, asumimos por ahora que $\mathcal{N}_u = \mathcal{N}$ y elide subíndice u en contextos sin ambigüedad. Está garantizada la compatibilidad de los DAG construidos por diferentes nodos. Especialmente, si $T' \leftarrow T$; y por el contrario, si $T' \nleftarrow T$, entonces ningún nodo terminará con $T' \leftarrow T$.

1: **function** isPreferred(T)
2: **return** $T = \mathcal{P}_T$ `.pref`
3: **function** isStronglyPreferred(T)
4: **return** $\forall T' \in \mathcal{T}, T' \overset{*}{\leftarrow} T$: isPreferred(T')
5: **function** isAccepted(T)
6: **return**
$((\forall T' \in \mathcal{T}, T' \leftarrow T$: isAccepted(T'))
$\wedge |\mathcal{P}_T| = 1 \wedge d(T) > \beta_1)$ `// safe early commitment`
$\vee \mathcal{P}_T$.cnt $> \beta_2$ `// consecutive counter`

7: **procedure** onQuery(j, T)
8: onReceiveTx(T)
9: respond(j, isStronglyPreferred(T))

Figura 5: Avalanche: generación de transacciones.

```
1: procedure INIT
2:     𝒯 := ∅  // the set of known transactions
3:     𝒬 := ∅  // the set of queried transactions
4: procedure ONGENERATETX(data)
5:     edges := {T′ ← T : T′ ∈ PARENTSELECTION(𝒯)}
6:     T := TX(data, edges)
7:     ONRECEIVETX(T)
8: procedure ONRECEIVETX(T)
9:     if T ∉ 𝒯 then
10:        if 𝒫_T = ∅ then
11:            𝒫_T := {T}, 𝒫_T.pref := T
12:            𝒫_T.last := T, 𝒫_T.cnt := 0
13:        else 𝒫_T := 𝒫_T ∪ {T}
14:        𝒯 := 𝒯 ∪ {T}, c_T := 0.
```

Figura 6: Avalanche: votación y decisión de primitivas.

Cada nodo implementa una máquina de estados controlada por eventos, centrada alrededor de una consulta que sirve para solicitar votos en cada transacción y, simultáneamente, para notificar a otros nodos la existencia de transacciones recién descubiertas. En particular, cuando el nodo u descubre una transacción T a través de una consulta, se inicia un proceso de consulta de una sola vez por muestreo de clientes al azar. Una consulta comienza agregando T a $\mathcal{T}$, inicializando C_T a 0 y luego enviando un mensaje a los clientes seleccionados.

Nodo u responde a una consulta verificando si cada T' como $T' \overset{*}{\leftarrow} T$ es tal que actualmente se prefiere entre las transacciones de la competencia $\forall T'' \in \mathcal{P}_{T'}$. Si cada antepasado T' cumple con este criterio, se dice que la transacción es *fuertemente preferida* y recibe un voto de sí (1). Un fallo de este criterio en cualquier T' produce un voto negativo (0). Cuando u acumula k respuestas, comprueba si hay αk respuestas positivas para T, y si es así concede la ficha $C_T = 1$ para T. El proceso anterior producirá un etiquetado del DAG con el valor de la ficha C_T y la confianza asociada para cada transacción T. Las fichas son el resultado de muestras únicas y son inmutables, mientras que *d(T)* puede aumentar a medida que el DAG crece. Debido a que los valores de C_T varían de 0 a 1, los valores de confianza son monotónicos.

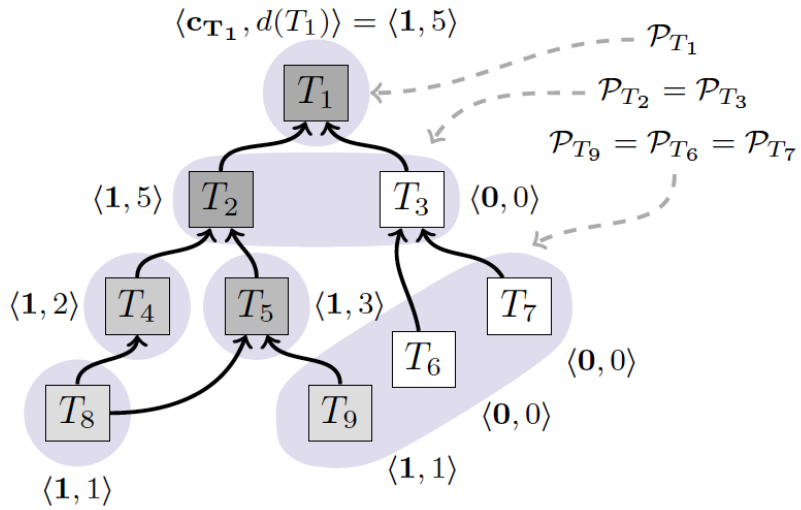

Figura 7: Ejemplo de valores de confianza y fichas, etiquetadas como tuplas en ese orden. Las cajas más oscuras indican transacciones con valores de confianza más altos.

La Figura 7 ilustra un DAG de muestra construido por Avalanche. Al igual que en Snowball, el muestreo en Avalanche creará una retroalimentación positiva para la preferencia de una sola

transacción en su conjunto de conflictos. Por ejemplo, como T_2 tiene mayor confianza que T_3, es más probable que sus descendientes recolecten fichas en el futuro en comparación con T_3.

Similar a Bitcoin, Avalanche deja determinar el punto de aceptación de una transacción a la aplicación. Una aplicación proporciona un predicado isAccepted (aceptado) que puede tener en cuenta el valor en riesgo en la transacción y las posibilidades de que se revierta una decisión para determinar cuándo decidir.

Se puede realizar el compromiso con una transacción a través de un *compromiso temprano seguro*. Para transacciones virtuosas, se acepta T cuando es la única transacción en su conjunto de conflictos y tiene una confianza mayor que el umbral β_1. Como en Snowball, T también puede aceptarse después de un número β_2 de consultas consecutivas exitosas. Si una transacción virtuosa no se acepta debido a un problema de vitalidad de los padres, podría aceptarse si se vuelve a emitir con otros padres.

La Figura 5 muestra cómo Avalanche realiza la selección principal y enreda las transacciones. Debido a que las transacciones que consumen y generan el mismo UTXO no entran en conflicto entre sí, cualquier transacción se puede volver a emitir con distintos padres.

La Figura 4 ilustra el bucle principal del protocolo ejecutado por cada nodo. En cada iteración, el nodo intenta seleccionar una transacción T que aún no se ha consultado. Si no existe tal transacción, el bucle se detendrá hasta que se agregue una nueva transacción a $\mathcal{T}$. Luego selecciona k clientes y consulta a esos clientes. Si más de αk de esos pares devuelven una respuesta positiva, el valor de la ficha se establece en 1. Después de eso, actualiza la transacción preferida de cada conjunto de conflictos de las transacciones en su ascendencia. A continuación, se agrega T al conjunto $\mathcal{Q}$, por lo que el nodo nunca más lo volverá a consultar. El código que selecciona clientes adicionales si algunos de los pares k no responden se omite por simplicidad.

La Figura 6 muestra lo que sucede cuando un nodo recibe una consulta para la transacción T del cliente j. Primero, agrega T a $\mathcal{T}$, a menos que ya lo tenga. Luego determina si T es fuertemente preferido actualmente. Si es así, el nodo devuelve una respuesta positiva al cliente j. De lo contrario, devuelve una respuesta negativa. Observe que, en el pseudocódigo, asumimos que cuando un nodo conoce T, también conoce recursivamente toda la ascendencia de T. Esto se puede lograr posponiendo la entrega de T hasta que toda su ascendencia sea recuperada recursivamente. En la práctica, se usa en paralelo un proceso de susurro adicional que disemina las transacciones, pero no se muestra en pseudocódigo por simplicidad.

3. Análisis

En esta sección, analizamos Slush, Snowflake, Snowball y Avalanche.

Modelo de red Asumimos una red de comunicación síncrona, donde en cada paso de tiempo t, un programador global elige un solo nodo correcto de manera uniforme y aleatoria.

Preliminares Sea $c = |C|$ y b = $|\mathcal{B}|$; sea $u \in C$ el nodo correcto; sea $k \in \mathbb{N}_+$; y sea $\alpha \in \mathbb{R}$ = (1/2, 1]. Dejamos que R ("rojo") y B ("azul") representen dos opciones genéricas conflictivas. Sin pérdida de generalidad, centramos nuestra atención en los conteos de R, Es decir, el número total de nodos que prefieren R.

Cada consulta de red de clientes k corresponde a una muestra sin reemplazo fuera de una red de n nodos, también conocida como muestra hipergeométrica. Dejamos que la variable aleatoria $\mathcal{H}^k_{\mathcal{N},x} \rightarrow [0,k]$ denote los recuentos resultantes de R (que va de 0 a k), donde x es el número total de R en la población. La probabilidad de que la consulta alcance el umbral requerido de αk o más votos viene dada por:

$$P(\mathcal{H}^k_{\mathcal{N},x} \geq \alpha k) = \sum_{j=\alpha k}^{k} \frac{\binom{x}{j}\binom{n-x}{k-j}}{\binom{n}{k}} \qquad (1)$$

Tail Bound (Cota de cola) Podemos reducir algo de la complejidad en la Ecuación 1 b introduciendo un límite en la distribución hipergeométrica inducida por $\mathcal{H}_{C,x}^{k}$. Sea $p = x/c$ la relación de soporte para R en la población. La expectativa de $\mathcal{H}_{C,x}^{k}$ es exactamente kp. Entonces, la probabilidad de que $\mathcal{H}_{C,x}^{k}$ se desvíe de la media en más de una pequeña constante ψ viene dada por la cota de cola de Hoeffding [29], como sigue,

$$P(\mathcal{H}_{C,x}^{k} \leq (p-\psi)k) \leq e^{-k\mathcal{D}(p-\psi,p)} \qquad (2)$$

donde $\mathcal{D}(p-\psi, p)$ es la divergencia de Kullback-Leibler, medida como

$$\mathcal{D}(a,b) = a\log\frac{a}{b} + (1-a)\log\frac{1-a}{1-b} \qquad (3)$$

3.1 Análisis de Slush

Slush opera en una configuración no bizantina; es decir, $\mathcal{B} = \emptyset$ y así $C = \mathcal{N}$ y $c = n$. En esta sección, mostraremos que (R1) Slush converge a un estado donde todos los nodos coinciden en el mismo color en tiempo finito casi con total seguridad (es decir, con probabilidad 1); (R2) proporciona una expresión de forma cerrada para acelerar hacia la convergencia; y (R3) caracteriza el número mínimo de pasos por nodo necesarios para llegar a un acuerdo con probabilidad $\geq 1-\epsilon$.

La versión de procedimiento de Slush en la Figura 1 hizo uso de un parámetro m, el número de rondas que un nodo ejecuta las consultas de Slush. Para derivar este parámetro, transformamos la ejecución del protocolo de una versión de procedimiento y concurrente a una realizada por un planificador, como se muestra en la Figura 8. Lo que finalmente queremos extraer es el número total de rondas ϕ que el planificador deberá ejecutar en orden para garantizar que toda la red es del mismo color con una alta probabilidad.

1: initialize $u.\text{col} \in \{\text{R}, \text{B}\}$ for all $u \in \mathcal{N}$
2: **for** $t = 1$ to ϕ **do**
3: $\quad u := \text{SAMPLE}(\mathcal{N}, 1)$
4: $\quad \mathcal{K} := \text{SAMPLE}(\mathcal{N} \backslash u, k)$
5: $\quad P := [v.\text{col} \quad \textbf{for } v \in \mathcal{K}]$
6: $\quad$ **for** $\text{col}' \in \{\text{R}, \text{B}\}$ **do**
7: $\quad\quad$ **if** $P.\text{COUNT}(\text{col}') \geq \alpha \cdot k$ **then**
8: $\quad\quad\quad u.\text{col} := \text{col}'$

Figura 8: Slush ejecutado por un planificador global.

Analizamos el sistema principalmente utilizando técnicas estándar de cadena de Markov. Sea $\mathbf{X} = \{X_1, X_2, X_3, \ldots X_\psi\}$ una cadena de Markov de tiempo discreto. Sea $s_i, i \in [0, c]$ un estado en esta cadena de Markov, donde la cuenta de R es i y la cuenta de B es $c - i$. Sea $q(s_i) = M(s_i, s_{i-1})$, $p(s_i) = M(s_i, s_{i+1})$ y $r(s_i) = M(s_i, s_i)$, donde $p(s_0) = q(s_0) = p(s_c) = q(s_c) = 0$. Este es un proceso de nacimiento-muerte donde el número de estados es c y los dos puntos finales, s_0 y s_c, son estados absorbentes, donde todos los nodos son rojo y azul, respectivamente.

Para extraer valores concretos, aplicamos las siguientes probabilidades a $q(s_i)$ y $p(s_i)$, que capturan la probabilidad de seleccionar un nodo rojo y muestrear con éxito el azul, y viceversa.

$$p(s_i) = \frac{c-i}{c} P(\mathcal{H}^k_{\mathcal{C},i} \geq \alpha k)$$
$$q(s_i) = \frac{i}{c} P(\mathcal{H}^k_{\mathcal{C},c-i} \geq \alpha k)$$

Lema 1 (R1). *Slush alcanza un estado de absorción en tiempo finito con casi total seguridad.*

Demostración. Sea s_i un estado de inicio donde $i \leq c - ak$. Todos estos estados s_i tienen una probabilidad distinta de cero de alcanzar el estado de absorción s_0 para todos los pasos de tiempo $t \leq i$. Los estados donde $i > c - a$ no tienen una posible mayoría de umbral para B, y todos los pasos de tiempo $t \leq i$ no permiten que suficientes transiciones lleguen a s_0. En estas condiciones, la probabilidad de absorción es estrictamente > 0 y, por lo tanto, Slush converge en pasos finitos.

Lemma 2 (R2). ***Rondas esperadas para decidir B***. *Sea U_{s_i} una variable aleatoria que exprese el número de pasos para alcanzar s_0 desde el estado s_i. Sea*

$$\mathscr{Y}_c(h) = \sum_{l=s_h}^{s_{c-1}} \prod_{j=s_1}^{l} \frac{q(j)}{p(j)} \tag{4}$$

Entonces,

$$\begin{aligned} \mathbb{E}[U_{s_i}] = \mathbb{E}[U_{s_1}] &\sum_{l=s_0}^{s_{i-1}} \prod_{j=s_1}^{l} \frac{q(j)}{p(j)} \\ &- \sum_{l=s_0}^{s_{i-1}} \sum_{j=s_1}^{l} \frac{1}{p(j)} \prod_{v=j}^{l} \frac{q(v+1)}{p(v+1)} \end{aligned} \tag{5}$$

donde

$$\mathbb{E}[U_{s_1}] = \mathscr{Y}_c(0)^{-1} \sum_{l=s_1}^{s_{c-1}} \sum_{j=s_1}^{l} \frac{1}{p(j)} \prod_{v=j}^{l} \frac{q(v+1)}{p(v+1)} \tag{6}$$

Demostración. El tiempo esperado para la absorción en s_0 se puede modelar usando la siguiente ecuación de recurrencia:

$$\begin{aligned} \mathbb{E}[U_{s_i}] = q(s_i)\mathbb{E}[U_{s_{i-1}}] + p(s_i)\mathbb{E}[U_{s_{i+1}}] \\ + r(s_i)\mathbb{E}[U_{s_i}] \end{aligned} \tag{7}$$

para $i \in [1, c-1]$ con condición de contorno $\mathbb{E}[U_{s_0}] = 0$. Resolviendo para la relación de recurrencia se obtiene el resultado.

$\mathcal{C}$ Size	600	1200	2400	4800	9600
Exp. Conv.	12.66	14.39	15.30	16.43	18.61

Tabla 1: Número esperado de iteraciones por nodo para la convergencia a partir de la división de la red $\mathcal{C}$ en el caso más desfavorable (igual), en el caso de $k = 10,\ a = 0.8$. La desviación estándar para todas las muestras es ≤ 2.5.

La Tabla 1 muestra la latencia (es decir, el número de pasos de tiempo por nodo) para alcanzar el estado de decisión B a partir de la división 50/50 metaestable, es decir, el valor $\mathbb{E}\left[U_{s_{c/2}}\right]/c$. A medida que el tamaño de la red c se duplica, la convergencia esperada crece solo linealmente.

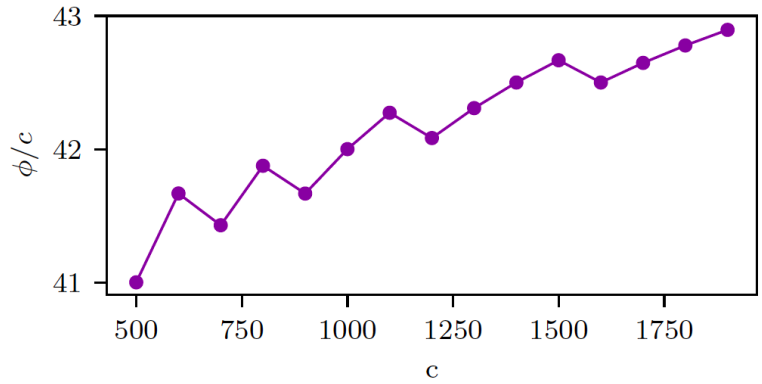

Figura 9: Parámetro necesario $\emptyset/c$ requerido para lograr la ϵ-convergencia en el protocolo Slush, es decir, cada nodo ha alcanzado el mismo color con una alta probabilidad. Fijamos $k = 10, a = 0.8$.

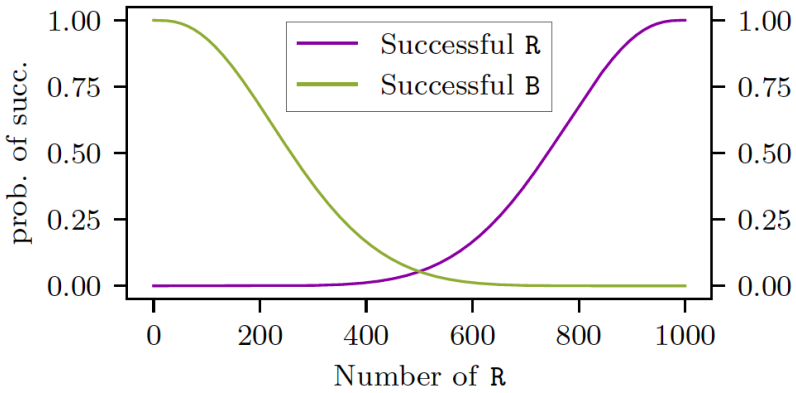

Figura 10. Probabilidad de éxito en comparación con muestras de divisiones de la red.

Finalmente, para demostrar R3, debemos calcular el primer paso de tiempo de la cadena M de Markov que produce una probabilidad $\leq \epsilon$ de todos los estados transitorios. Este cálculo no tiene una simple aproximación de forma cerrada, por lo tanto, debemos aumentar iterativamente la matriz de probabilidad de transición M hasta el primer paso de tiempo t que alcanza la probabilidad $\leq \epsilon$ para todos los estados s_i donde $s_{ak} < s_i < s_{c-ak}$. La Figura 9 muestra el tiempo esperado para alcanzar s_{c-ak} con una alta probabilidad, en varios tamaños de red, con $k = 10, a = 0.8$ fijos.

Los siguientes dos resultados adicionales proporcionan una intuición útil para los procesos estocásticos subyacentes de Slush y los protocolos subsiguientes. La probabilidad de una única

consulta exitosa en función del número total de nodos que soportan R en la red se muestra en la Figura 11. Esta probabilidad disminuye rápidamente a medida que aumenta el valor de k. La red se derrumba más rápido si el valor de k es menor debido a la perturbación por mayor cantidad de consultas aleatorias. De hecho, para Slush, $k = 1$ es óptimo. Más adelante veremos que para el caso bizantino, necesitaremos un valor mayor para k.

La Figura 10 muestra la divergencia de las probabilidades de muestras exitosas en varias divisiones de red. Cuando la red está dividida de manera uniforme, las probabilidades son iguales para cada color, pero divergen muy rápidamente a medida que la red se vuelve menos equilibrada.

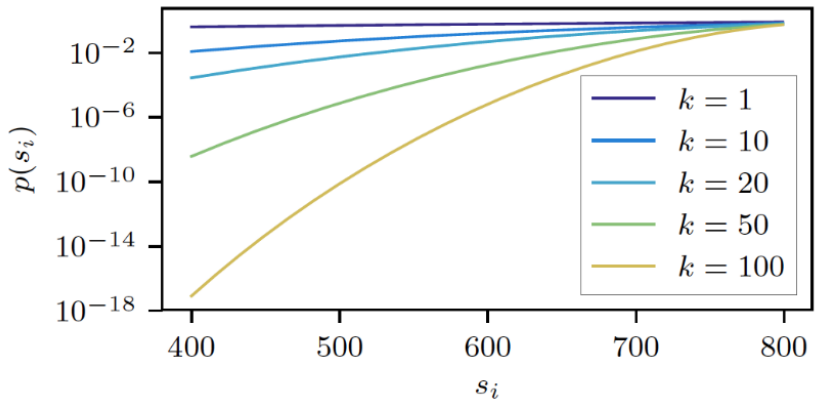

Figura 11: Probabilidad de una única muestra de mayoría roja exitosa con k variable, $a = 0.8$ y éxitos que van desde la mitad de la red hasta el soporte completo.

3.2 Análisis de seguridad de Snowflake

Snowflake, cuyo pseudocódigo se muestra en la Figura 12, es similar a Slush, pero tiene tres diferencias clave. Primero, el conjunto muestreado de nodos incluye nodos bizantinos. En segundo lugar, cada nodo también realiza un seguimiento del número total de veces consecutivas que ha muestreado la mayoría del mismo color. También introducimos una función llamada $\mathcal{A}$, la estrategia de confrontación, que toma como parámetros todo el espacio de los nodos correctos, así como el nodo u elegido en el momento t, y como efecto secundario, modifica el conjunto de nodos $\mathcal{B}$ a una configuración arbitraria de colores.

La cadena de nacimiento-muerte que modela a Snowflake, que se muestra a continuación, es casi idéntica a la de Slush, con la presencia agregada de nodos bizantinos.

$$p(s_i) = \frac{c-i}{c} P(\mathcal{H}^k_{\mathcal{C},i} \geq \alpha k),$$
$$q(s_i) = \frac{i}{c} P(\mathcal{H}^k_{\mathcal{C},c-i+b} \geq \alpha k)$$

En Slush, el estado metaestable de la red era una división del 50/50 en el estado de Markov $s_{c/2}$. En Snowflake, los nodos bizantinos tienen la capacidad de compensar las desviaciones desde $s_{c/2}$ hasta un umbral proporcional a su tamaño en la red. Sea el estado de la cadena de Markov $s_{ps} > s_{c/2}$ el estado a partir del cual la probabilidad de ingresar al estado de absorción de R es mayor que la probabilidad de ingresar al estado de absorción de B dado el peso total de los nodos bizantinos. Simétricamente, existe un punto de cambio de fase para B también. Todos los estados entre estos dos puntos de cambio de fase están controlados por Bizancio, mientras que todos los demás estados son metaestables.

En esta sección, analizamos las dos condiciones que garantizan la seguridad de Snowflake. La primera condición (C1) requiere que se demuestre que existe un punto sin retorno $s_{ps+\Delta}$ después del cual la probabilidad máxima bajo la actividad bizantina completa de devolver el sistema al punto de cambio de fase es menor que ϵ.

La segunda condición (C2) requiere la construcción de un mecanismo para garantizar que un proceso solo pueda comprometerse con un color si el sistema está más allá del punto mínimo de no retorno, con una alta probabilidad. Como mostraremos más adelante, satisfacer estas dos condiciones es necesario y suficiente para garantizar la seguridad de Snowflake con una alta probabilidad.

```
initialize ∀u, u.col,
initialize ∀u, u.cnt := 0
for t = 1 to φ do
    u := SAMPLE(C, 1)
    A(u, C)
    K := SAMPLE(N\u, k)
    P := [v.col   for v ∈ K]
    for col' ∈ {R, B} do
        if P.COUNT(col') ≥ α · k then
            if col' ≠ u.col then
                u.col := col', u.cnt := 0
            else
                ++u.cnt
```

Figura 12: Snowflake ejecutado por un planificador global.

Propiedades del Parámetro de Muestreo k. Antes de construir y analizar las dos condiciones que satisfacen la seguridad de Snowflake, introducimos algunas observaciones importantes del tamaño de la muestra k. La primera observación de interés es que la distribución hipergeométrica se aproxima a la distribución binomial para tamaños de red suficientemente grandes. Si x es una fracción constante del tamaño de red n, entonces:

$$\lim_{n\to\infty} P(\mathcal{H}^k_{\mathcal{N},x} \geq \alpha k) = \sum_{j=\alpha k}^{k} \binom{k}{j} (x/n)^k (1-(x/n))^{k-j} \tag{8}$$

Esto determina que, para tamaños de red suficientemente grandes, la probabilidad de éxito para cualquier muestra individual se convierte en una función que depende solo de los valores de k y α, no se ve afectada por aumentos adicionales en el tamaño de la red. El resultado tiene consecuencias para la escalabilidad de Snowflake y protocolos posteriores. Para k suficientemente grande y fijo, la seguridad del sistema será aproximadamente igual para todos los tamaños de red $n > k$.

Ahora examinamos cómo se comporta el punto de cambio de fase al aumentar k. Mientras que $k = 1$ es óptimo para Slush (es decir, la red se derrumba más rápido cuando $k = 1$), vemos que la seguridad de Snowflake se desplaza al máximo de su posición ideal para $k = 1$, lo que reduce el número de soluciones posibles. Afortunadamente, los pequeños incrementos en k tienen un efecto exponencial en los s_{ps}, creando un conjunto más amplio de soluciones factibles. Más formalmente:

Lema 3. *s_{ps} se acerca $s_{c/2+b/2}$ exponencialmente rápido a medida que k se acerca a n.*

Demostración. En s_{ps}, la ratio $p(s_i)/\,q(s_i)$, es igual a 1 por definición. Sustituyendo la solución $i = c/2 + b/2$ en esta ecuación se obtiene el resultado en $k = n$. Al calcular el diferencial en s_{ps} entre cualquier k y k +1 elegido se obtiene la tendencia exponencial. La Figura 13 muestra

la relación exponencial entre la disminución de k y s_{ps}. El gráfico muestra que los valores pequeños, pero no minúsculos de k conducen a grandes regiones factibles.

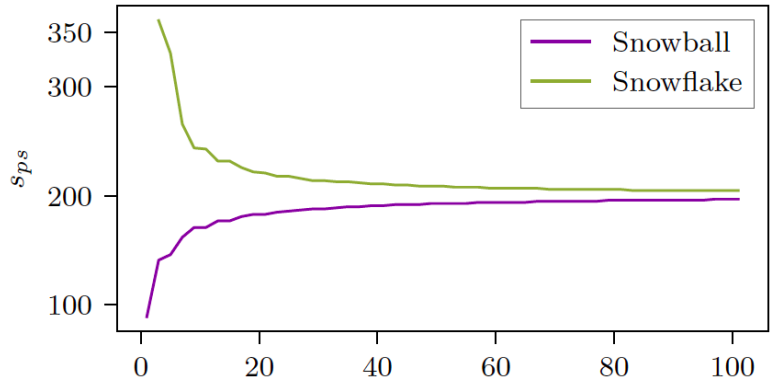

Figura 13: Estado de cambio de fase de Snowflake / Snowball vs. k valores (eje x), con n = 2000, α= 0.8 y b = 20% de n. El cambio de fase se indexa a partir de $c/2$.

Satisfacción de los criterios de seguridad Sea ϵ un parámetro de seguridad del sistema de valor $\leq 2^{-32}$, generalmente denominado probabilidad *insignificante*.

Ahora formalizamos y analizamos la condición de seguridad C1. Nos referimos a todos los estados s_i que satisfacen la condición C1 como *estados de soluciones factibles*. Para que el sistema garantice la seguridad, debe existir algún estado $s_{c/2+\Delta} > s_{c/2+b} > s_{ps}$ que implique ϵ-reversibilidad en la cadena de nacimiento-muerte. En otras palabras, una vez que el sistema alcanza este estado, vuelve con una probabilidad mínima, bajo *cualquier* estrategia Bizantina. La Figura 14 resume la región de interés. Nuestro objetivo es determinar los valores de los parámetros del sistema ajustables para lograr esta condición, dado el tamaño de red objetivo n, el componente bizantino máximo b y el parámetro de seguridad ϵ, fijados por el diseñador del sistema.

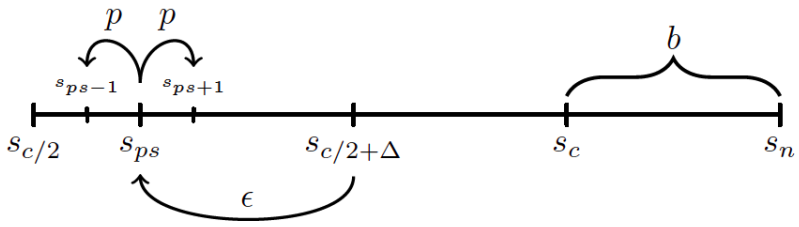

Figura 14: Representación del espacio de soluciones factibles. Fuera del punto de cambio de fase, la red crea un sesgo hacia un extremo. En el punto Δ, este sesgo es irreversible.

Para asegurar C1, debemos encontrar Δ que se cumpla la siguiente condición:

$$\exists \Delta, \quad \text{donde} \quad \begin{array}{c} s_{c/2+\Delta} > s_{ps},\ s.t.\ \forall t \leq \phi : \\ \epsilon \geq (V_c[s_{c/2+\Delta}, s_{ps}])^t \end{array} \tag{9}$$

donde V_c es la matriz de transición de probabilidad (c + 1)^2 para el sistema. En terminología de teoría de la probabilidad, la declaración anterior calcula la probabilidad de acertar en t pasos, donde el estado de inicio es $s_{c/2+\Delta}$ y el estado de acierto es s_{ps}. Satisfacer esta primera restricción, es decir, mover el conjunto de nodos a una decisión, no es suficiente. *Los nodos* deben poder asegurarse de que pueden decidir con seguridad, lo que significa que deben garantizar la condición C2. Si bien hay varias formas de construir el predicado que garantiza esta condición, la que adoptamos en Snowflake implica consultas β exitosas consecutivas, de manera que *P*

(isAccepted $\mid s_i \leq s_{c/2+\Delta}) \leq \epsilon$. Para alcanzar este umbral de probabilidad de fallo para C2, resolvemos la más pequeña β que cumpla con la desigualdad $P(\mathcal{G}(p, \phi/c) \geq \beta) \leq \epsilon$ donde p = $P\ (\mathcal{H}^k_{N,i} \geq \alpha k)$ y $\mathcal{G}(p, \phi/c)$ es una variable aleatoria que cuenta el mayor número de éxitos consecutivos del mismo color para una prueba ϕ/c intentos. Resolver para el mínimo i produce β.

La tarea del diseñador del sistema, finalmente, es derivar valores para α y β, dadas las deseadas n, b, k, ϵ. La elección de α sigue inmediatamente de b y es $(n-b)/n$, el tamaño máximo permitido para tolerar $\leq b$ nodos bizantinos. Resolver Δ con una expresión de forma cerrada sería ideal, pero desafortunadamente es difícil, por lo tanto, empleamos una búsqueda numérica e iterativa. Como k y β son mutuamente dependientes, un diseñador de sistemas normalmente reparará uno y buscará el otro. Debido a que β es un parámetro sintonizable cuya consideración es principalmente la latencia frente a la seguridad, un diseñador puede corregirlo en un valor deseado y buscar la k correspondiente. Como es deseable un k bajo, lo hacemos comenzando en $k = 1$, y evaluando sucesivamente valores mayores de k hasta que se encuentre una solución viable. Dependiendo de b y del ϵ elegido, tal solución puede no existir. Si es así, esto será evidente durante el diseño del sistema.

Finalizamos nuestro análisis componiendo formalmente C1 y C2:

Teorema 4. Si C1 y C2 se satisfacen con los parámetros del sistema elegidos apropiadamente, entonces la probabilidad de que dos nodos u y v decidan sobre R y B respectivamente es estrictamente $< \epsilon$ en todos los pasos de tiempo $0 \leq t \leq \phi$.

Demostración. La prueba sigue de manera directa desde las garantías principales que proporcionan C1 y C2. Sin pérdida de generalidad, suponga que un solo nodo u decide R en el tiempo $t < \phi$, y suponga que la red está en el estado s_j en el momento de la decisión. Por construcción, isAccepted va a ser verdadero con probabilidad ϵ no mayor que para todos los estados de la red $s_i < s_{c/2+\Delta}$. Por lo tanto, ya que u decidió, debe ser el caso que $s_j > s_{c/2+\Delta}$, con alta probabilidad. Por último, dado que el sistema solo volverá a un estado con una mayoría de nodos B con una probabilidad insignificante una vez que haya pasado $s_{c/2+\Delta}$, la probabilidad de que un nodo v se decida por B es estrictamente $< \epsilon$.

Para propósitos de ilustración, examinamos una red con $n = 2000$; $\alpha = 0.8, \epsilon \leq 2^{-32}$ y rendimiento de 10000tps. Si permitimos que este sistema se ejecute durante ~ 4,000 años (lo que corresponde a $\phi = 10^{15}$), elegimos β tal que la probabilidad de que un nodo se comprometa en el estado $s_{c/2+\Delta-1}$ sea ϵ. Entonces ocurre un fallo de seguridad cuando un nodo se compromete con R y el sistema revierte a B. La Figura 15 muestra la probabilidad de fallo y la correspondiente β para diferentes elecciones de k.

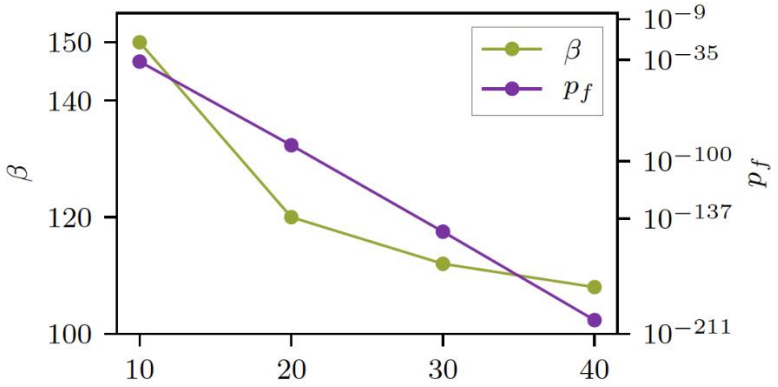

Figura 15: β y probabilidad de fallo p_f vs. k valores (eje x), con $n = 2000$, $\alpha = 0.8$.

Tenga en cuenta que una mayor elección del parámetro α dicta un mayor espacio de soluciones factibles, lo que implica una mayor tolerancia a la presencia bizantina. Si el diseñador

del sistema elige permitir una mayor presencia de nodos bizantinos, puede lograr este objetivo a expensas de la vitalidad.

3.3 Análisis de seguridad de Snowball

En esta sección, demostraremos que Snowball proporciona una seguridad estrictamente mejor que Snowflake. La diferencia clave en la mecánica del protocolo entre Snowflake y Snowball es que los nodos correctos ahora aumentan monótonamente su confianza en R y B. Esto limita el efecto de las perturbaciones aleatorias, así como el impacto del adversario bizantino.

Para comprender de manera intuitiva las diferencias de comportamiento entre Snowball y su predecesor, lo ilustramos con un ejemplo, acompañado de algunos valores numéricos concretos. Supongamos una red de tamaño n donde $b = (1/5)n$, $\alpha = (4/5)k$ (es decir, el umbral máximo permisible para tolerar la presencia de nodos bizantinos $(1/5)\,n$, y donde k se elige lo suficientemente grande como para admitir una solución viable. Supongamos que la red cambia al estado $s_{c/2+\Delta}$, de modo que los nodos correctos tienen una probabilidad no despreciable de decidir R, y supongamos, en aras de la simplicidad, que se elige Δ como el valor más alto posible, específicamente $c/2 + \Delta + b = \alpha n = (4/5)n$, el umbral superior establecido por el diseñador del sistema.

En esta configuración, al menos $(4/5)n - b = (3/5)n$ nodos (correctos) prefieren R. Estos nodos perciben soporte de red para R de a lo sumo $(4/5)n$ si los nodos bizantinos eligen votar también por R, y por lo menos $(3/5)n$ si los nodos bizantinos eligen retirar el soporte para R. A la inversa, el resto de los $(1/5)n$ (correctos) nodos que pueden preferir B perciben el soporte de red para B de al menos $(1/5)n$ y como máximo $(2/5)n$, dependiendo de la estrategia bizantina. Sin embargo, independientemente de la estrategia bizantina, todos los nodos correctos que prefieren R tendrán un crecimiento esperado de la confianza R para ser estrictamente mayor que la confianza B para los nodos que prefieren B. Esto se debe a que el soporte de red para R es al menos $(3/5)n$, que es mayor que el soporte de red máximo para B de $(2/5)n$. Por lo tanto, una vez que la red cambie al estado mínimo requerido, el crecimiento esperado de la confianza R será mayor que el de la confianza B, para todos los nodos correctos y para todas las estrategias bizantinas.

En la sección anterior, proporcionamos las dos condiciones necesarias para garantizar la seguridad de Snowflake. La segunda condición, C2, sigue siendo idéntica en Snowball y, por lo tanto, no necesita ser modificada y analizada nuevamente. A la inversa, la condición C1 (es decir, la que muestra que la red es irreversible con una alta probabilidad) debe volver a analizarse. En esta sección, utilizando la misma intuición proporcionada por el ejemplo anterior, mostramos formalmente cómo Snowball logra la irreversibilidad ante una probabilidad estrictamente mayor que en Snowflake.

Seguridad en Snowball. La Figura 16 ilustra la versión basada en el planificador de Snowball. En Snowball, un nodo u prefiere R si $u.d[\mathrm{R}] > u.d[\mathrm{B}]$, y viceversa. Al inicio del protocolo, estas preferencias se inicializan a $\langle 0,0 \rangle$. Esto contrasta con Snowflake, donde los nodos prefieren un color basado solo en la última muestra exitosa. Podemos modelar el protocolo con una cadena de Markov similar a la de Snowflake, y derivar los parámetros y la viabilidad del protocolo.

```
initialize ∀u, u.col
initialize ∀u, u.lastcol
initialize ∀u, u.cnt := 0
initialize ∀u, ∀col′ ∈ {R, B}, u.d[col′] := 0
for t = 1 to ϕ do
    u := SAMPLE(C, 1)
    A(u, C)
    K := SAMPLE(N\u, k)
    P := [v.col  for v ∈ K]
    for c′ ∈ {R, B} do
        if P.COUNT(col′) ≥ α · k then
            u.d[col′]++
            if u.d[col′] > u.d[col] then
                u.col := col′
            if col′ ≠ u.lastcol then
                u.lastcol := col′, u.cnt := 0
            else
                ++u.cnt
```

Figura 16: Snowball ejecutado por un planificador global.

Arreglamos un Δ tal que, bajo la construcción Markoviana de Snowflake, si el sistema alcanza $s_{c/2+\Delta}$, entonces con una alta probabilidad no se revierte. Demostramos que esta elección de Δ ofrece una garantía aún más sólida una vez que la red cambia el protocolo a Snowball. Sin pérdida de generalidad, agrupamos los nodos correctos en dos grupos, y representamos a cada grupo a través de dos nodos u y v, donde u representa el conjunto de nodos correctos que prefieren rojo y v que prefieren azul. Por último, permitamos que el estado del sistema se inicialice en $s_{c/2+\Delta}$.

La mejor configuración de confianza posible en la que los nodos bizantinos pueden intentar forzar a los nodos correctos es donde todos los nodos que prefieren R, representados por u, se ven obligados a mantener una confianza casi igual entre los dos colores, y donde todos los nodos que prefieren B, representado por v, ganan tanta confianza como sea posible en B.

Sea $u.d[\mathrm{R}] = u.d[\mathrm{B}] + 1$, correspondiente a la mínima diferencia de color viable para los nodos que prefieren el rojo, y sea $u.d[\mathrm{B}] - u.d[\mathrm{R}] = \kappa$ la diferencia para los nodos de preferencia azul. Mientras que $\kappa \geq 0$, entonces la expectativa de crecimientos de preferencia en el tiempo t es:

$$\begin{aligned}
\mathbb{E}[u.d^t[\mathtt{R}]] &= \mathbb{E}[u.d^{t-1}[\mathtt{R}]] + \left(\frac{1}{2} + \frac{\Delta}{c}\right) P(\mathcal{H}^k_{\mathcal{N},c/2+\Delta+b} \geq \alpha k) \\
\mathbb{E}[u.d^t[\mathtt{B}]] &= \mathbb{E}[u.d^{t-1}[\mathtt{B}]] + \left(\frac{1}{2} + \frac{\Delta}{c}\right) P(\mathcal{H}^k_{\mathcal{N},c/2-\Delta+b} \geq \alpha k) \\
\mathbb{E}[v.d^t[\mathtt{R}]] &= \mathbb{E}[v.d^{t-1}[\mathtt{R}]] + \left(\frac{1}{2} - \frac{\Delta}{c}\right) P(\mathcal{H}^k_{\mathcal{N},c/2+\Delta} \geq \alpha k) \\
\mathbb{E}[v.d^t[\mathtt{B}]] &= \mathbb{E}[v.d^{t-1}[\mathtt{B}]] + \left(\frac{1}{2} - \frac{\Delta}{c}\right) P(\mathcal{H}^k_{\mathcal{N},c/2-\Delta+b} \geq \alpha k)
\end{aligned} \tag{10}$$

Tenga en cuenta que estamos utilizando implícitamente una función indicadora al sumar la probabilidad de éxito al crecimiento de confianza esperado, donde el valor esperado del indicador es exactamente la probabilidad de éxito de una muestra. Sustituyendo por la cota de cola de Hoeffding, obtenemos que la velocidad a la que κ decrece por cada paso de tiempo es

$$r = \left(\left(\frac{1}{2} - \frac{\Delta}{c}\right) e^{-k\alpha \log\left(\alpha / \frac{c/2-\Delta+b}{c+b}\right)} \right.$$
$$\left. e^{-k(1-\alpha)\log\left((1-\alpha)/\left(1-\frac{c/2-\Delta+b}{c+b}\right)\right)}\right)$$
$$-\left(\left(\frac{1}{2} - \frac{\Delta}{c}\right) e^{-k\alpha \log\left(\alpha / \frac{c/2+\Delta+b}{c+b}|\right)} \right.$$
$$\left. e^{-k(1-\alpha)\log\left((1-\alpha)/\left(1-\frac{c/2+\Delta+b}{c+b}\right)\right)}\right) \quad (11)$$

Al alcanzar $\kappa = -1$, los nodos que prefieren el azul habrán cambiado a rojo. A partir de entonces, todos los nodos correctos son rojos, y la confianza del rojo crece significativamente más rápido que la del azul. Ahora mostramos que la tasa de crecimiento de R siempre será mayor que la tasa de crecimiento de B, en todos los nodos correctos.

Dejando que $X(u.d^t[\mathrm{R}])$, sea una variable aleatoria que genere la confianza de rojo para u con el paso de tiempo t, y dejando que $\bar{X}(u.d^t[\mathrm{R}]) = X(u.d^1[\mathrm{R}]) + \cdots + X(u.d^t[\mathrm{R}])$, aplicamos la desigualdad de concentración de Hoeffding para obtener esa $P(\bar{X}(u.d^t[\mathrm{R}]) - \mathbb{E}[\bar{X}(u.d^t[\mathrm{R}])] \geq l) \leq e^{-2l^2t}$. En el instante t, la confianza esperada del rojo para u es

$$u.d^0[\mathrm{R}] + t\left(\frac{1}{2} + \frac{\Delta}{c}\right)\left(\frac{\frac{c}{2} + \Delta + b}{c+b}\right) \quad (12)$$

Y a la inversa, la confianza esperada del azul para u es

$$u.d^0[\mathrm{B}] + t\left(\frac{1}{2} + \frac{\Delta}{c}\right)\left(\frac{\frac{c}{2} - \Delta + b}{c+b}\right) \quad (13)$$

Entonces, la probabilidad de que el valor esperado de la confianza roja esté cerca del valor esperado de la confianza azul es

$$e^{-2t^3(1/2+\Delta/c)^2(2\Delta/(c+b))^2} \quad (14)$$

La solución para t que lleva a que tal probabilidad sea despreciable conduce a:

$$t = \left(\frac{\log(\epsilon)}{-2 \cdot (1/2 + \Delta/c)^2(2\Delta/(c+b))^2}\right)^{1/3} \quad (15)$$

Por lo tanto, solo en un pequeño número de pasos de tiempo t, u tiene una confianza en rojo centrada alrededor de su media, con una probabilidad ϵ cercana a la media de la confianza en azul. Se sigue un análisis idéntico para v. En otras palabras, después de un pequeño número de pasos de tiempo, la confianza de u en el rojo crece lo suficiente como para que la probabilidad de que este valor sea cercano al de la confianza en el azul es despreciable. Más pasos de tiempo amplifican esta distancia y disminuyen aún más la probabilidad de que las dos confianzas estén cerca. Concluimos nuestro resultado con el siguiente teorema:

Teorema 5. *Sobre todos los parámetros de red viables n, b, y para todos los parámetros del sistema k elegidos apropiadamente y α, la probabilidad de violar la seguridad en Snowball es estrictamente menor que la probabilidad de violar la seguridad en Snowflake.*

Demostración. Bajo la construcción de Snowball en comparación con Snowflake, vemos que el criterio de seguridad C2 sigue siendo el mismo. En otras palabras, el predicado de decisión de éxitos consecutivos tiene las mismas garantías. Por otro lado, las garantías probabilísticas de C1 cambian, lo que significa que las probabilidades de reversibilidad del sistema son diferentes. Sin embargo, como determinamos en la Ecuación 15, a lo largo de unos pocos pasos u, la diferencia de confianza entre R y B crece lo suficiente como para garantizar que esta diferencia de confianza se revertirá tan solo con una probabilidad insignificante. A medida que avanza el tiempo, la probabilidad de estar cerca disminuye poli-exponencialmente, como se muestra en la Ecuación 14. Los mismos resultados siguen para v, pero a la inversa, lo que significa que las medias se acercan entre sí en lugar de desviarse. Esto continúa hasta que v cambia de color.

Snowball tiene garantías de seguridad estrictamente más fuertes que Snowflake, lo que implica que los parámetros elegidos apropiadamente para Snowflake se aplican automáticamente a Snowball. Usando las mismas técnicas que antes, el diseñador del sistema elige valores k y valores β adecuados que garanticen las garantías de seguridad ϵ deseadas.

3.4 Análisis de seguridad de Avalanche

La diferencia clave entre Avalanche y Snowball es que en Avalanche, las consultas en el DAG sobre la transacción T_i se utilizan para consultar implícitamente toda la ascendencia de T_i. En particular, una transacción T_i es preferida por u *si y solo si* todos los ancestros también son preferidos. Supongamos que T_i y T_j están en el mismo conjunto de conflictos. Ahora podemos inferir dos cosas. Primero, podemos considerar toda la ascendencia de T_i y T_j como una instancia de decisión única de Snowball, donde la ascendencia de T_i puede considerarse la decisión R, y la ascendencia de T_j puede considerarse la decisión B. En segundo lugar, dado que se debe preferir T_i si se prefiere un hijo de T_i, entonces podemos colapsar la progenie de T_i en una sola urna que agrega repetidamente un color R a la confianza cada vez que un hijo de T_i recibe una ficha. En consecuencia, Avalanche se mapea a una instancia de Snowball, con las propiedades de seguridad mostradas anteriormente.

Sin embargo, notamos que, dado que una decisión sobre una transacción virtuosa depende de sus padres, las garantías de vitalidad de Avalanche no reflejan la de Snowball. Nos ocuparemos de esto en las siguientes dos subsecciones.

3.5 Compromiso temprano seguro

Como razonamos anteriormente, cada conflicto establecido en Avalanche puede verse como una instancia de Snowball, donde cada instancia de progenie vota iterativamente por todo el camino de la ascendencia. Esta característica proporciona varios beneficios; sin embargo, también puede llevar a que algunas transacciones virtuosas que dependen de una transacción deshonesta sufran el destino de esta última. En particular, las transacciones fraudulentas pueden interponerse entre transacciones virtuosas y reducir la capacidad de las transacciones virtuosas para alcanzar el predicado isAccepted. Como un experimento mental, suponga que una transacción T_i nombra un conjunto de transacciones primarias que están todas decididas, según la vista local. Si se toman muestras de T_i en un conjunto suficientemente grande de consultas exitosas sin descubrir ningún conflicto, entonces, dado que se asume que se decide toda la ascendencia de T_i, debe implicar que al menos $c/2 + \Delta$ nodos correctos también voten por T_i, logrando la irreversibilidad.

Por ello para medir estadísticamente la seguridad de que T_i haya sido aceptado por un gran porcentaje de nodos correctos sin ningún conflicto, utilizamos un proceso de nacimiento en un solo sentido, donde ocurre un nacimiento cuando un nuevo nodo correcto descubre el conflicto de T_i. Necesariamente, las muertes no pueden existir en este modelo, porque una transacción conflictiva no puede ser invisible una vez que un nodo correcto lo descubre. Sea $t = 0$ el momento en que T_j, que entra en conflicto con T_i, se introduce en un solo nodo correcto u. Sea

s_x, para $x = 1$ a c, el estado donde el número de nodos correctos que conocen T_j es x, y sea $p(s_x)$ la probabilidad de nacimiento en el estado s_x. Entonces, tenemos:

$$p(s_x) = \frac{c-x}{c}\left(1 - \frac{\binom{n-x}{k}}{\binom{n}{k}}\right) \qquad (16)$$

Resolver para el tiempo esperado en alcanzar el estado final de nacimiento proporciona un límite inferior para al parámetro β_1 en la rama de decisión rápida aceptada de isAccepted. La siguiente tabla muestra un ejemplo del análisis para $n = 2000$, $\alpha = 0.8$ y la variable k, donde $\epsilon \ll 10^{-9}$, y donde β es el tamaño mínimo requerido de $d(T_i)$. En general, un número muy pequeño de iteraciones son suficientes para el predicado de compromiso temprano seguro.

k	10	20	30	40
β	10.87625	10.50125	10.37625	10.25125

3.6 Vitalidad

Slush. Slush es un protocolo no tolerante a errores bizantinos, y hemos demostrado que termina dentro de un número finito de rondas casi con total seguridad.

Snowflake y Snowball. Ambos protocolos hacen uso de un contador para realizar un seguimiento del soporte mayoritario consecutivo. Como el adversario no puede forjar un conflicto para una transacción virtuosa, inicialmente, todos los nodos correctos tendrán color rojo o ⊥. Un nodo bizantino no puede responder a ninguna consulta con otra respuesta que no sea roja, ya que no puede forjar conflictos y ⊥ no está permitido por el protocolo. Por lo tanto, el único mal comportamiento del nodo bizantino es negarse a responder. Dado que el nodo correcto volverá a muestrear si la consulta se agota, según la ecuación de convergencia esperada en [2], todos los nodos correctos terminarán con el valor rojo unánime dentro de un número finito de rondas casi con total seguridad.

Avalanche. Avalanche introduce una estructura DAG que enreda el destino de conjuntos de conflictos no relacionados, cada uno de los cuales es una instancia de un solo decreto. Este enredo conlleva una tensión: adjuntar una transacción virtuosa a los padres indecisos ayuda a impulsar las transacciones hacia una decisión, mientras que las transacciones corren el riesgo de sufrir fallos en la vitalidad cuando los padres se vuelven deshonestos. Podemos resolver esta tensión y proporcionar una garantía de vitalidad con la ayuda de los siguientes mecanismos:

Buena ascendencia eventualmente. Las transacciones virtuosas pueden reintentarse eligiendo nuevos padres, seleccionados de un conjunto que es más probable que sea el preferido. En última instancia, siempre se puede adjuntar una transacción a los padres decididos para mitigar completamente este riesgo. Una técnica simple para la selección de los padres es seleccionar nuevos padres para una transacción virtuosa en alturas sucesivamente más bajas en el DAG, avanzando hacia el vértice de la génesis. Se garantiza que este procedimiento terminará con los padres decididos y sin oposición, asegurando que la transacción no pueda sufrir un fallo de vitalidad debido a transacciones deshonestas.

Fichas suficientes. Un mecanismo secundario es necesario para asegurar que las transacciones virtuosas con ascendencia decidida recibirán suficientes fichas. Para garantizar esto, los nodos correctos examinan el DAG en busca de transacciones virtuosas no- NOP que no tienen suficiente progenie y que emiten transacciones NOP para ayudar a aumentar su confianza. Una transacción NOP tiene solo un padre y no tiene efectos secundarios en la aplicación, y puede ser emitida por cualquier nodo. Los nodos bizantinos no pueden abusar de ellos porque, aunque los NOPs activan nuevas consultas, no otorgan fichas automáticamente.

3.7 Mezclar y Ver las Actualizaciones

Cualquier sistema realista necesita acomodar la salida y llegada de nodos. Hasta ahora, simplificamos nuestro análisis al asumir un conocimiento preciso de la membresía de la red. Ahora demostramos que los nodos de Avalanche pueden admitir una cantidad de rotación bien caracterizada, al mostrar cómo seleccionar parámetros para que los nodos de Avalanche puedan diferir en su visión de la red y tomar decisiones de manera segura.

Considere una red cuyo funcionamiento está dividido en épocas de duración τ, y una actualización de vista de la época t a $t+1$ durante la cual los nodos γ se unen a la red y los nodos $\bar{\gamma}$ salen. Bajo nuestra construcción estática, el espacio de estado s de la red tenía un parámetro clave Δ^t en el tiempo t, inducido por c^t, b^t, n^t y los parámetros de seguridad elegidos. Esto puede, en el peor de los casos, impactar en la red agregando γ nodos de color B y eliminar los $\bar{\gamma}$ nodos de color R. En el momento $t+1$, $n^{t+1} = n^t + \gamma - \bar{\gamma}$, mientras que b^{t+1} y c^{t+1} se modificarán en una cantidad $\leq \gamma - \bar{\gamma}$, y así se inducirá un nuevo Δ^{t+1} para los parámetros de seguridad elegidos. Este nuevo Δ^{t+1} tiene que ser elegido tal que $P(s_{c^{t+1}}/2 + \Delta^{t+1} - \gamma \rightarrow s_{ps}) \leq \epsilon$, para asegurar que el sistema converja bajo los supuestos pesimistas anteriores. El diseñador del sistema puede hacer esto fácilmente seleccionando un límite superior en $\gamma, \bar{\gamma}$.

El paso final para garantizar la corrección de un cambio de vista es tener en cuenta una combinación de nodos que se encuentran a ambos lados del límite τ. Nos gustaría que la red evitara un estado inseguro sin importar qué nodos estén usando las vistas antiguas y nuevas. La forma más fácil de hacer esto es determinar Δ^t y Δ^{t+1} para los límites deseados en $\gamma, \bar{\gamma}$, y luego usar el valor conservador Δ^{t+1} durante el momento t. En esencia, esto garantiza que no se realicen compromisos en el espacio de estado s^t a menos que cumplan de manera conservadora los criterios de seguridad en el espacio de estado s^{t+1}. Como resultado, no hay posibilidad de que un nodo decida rojo en el tiempo t, la red pase por un cambio de época y se encuentre a la izquierda del nuevo punto de no retorno Δ^{t+1}.

Este enfoque balancea parte del espacio de factibilidad, para agregar la capacidad de acomodar la rotación de $\gamma, \bar{\gamma}$ nodos por época. En general, si τ excede el tiempo requerido para tomar una decisión (en el orden de minutos a horas), y los nodos están ligeramente sincronizados, pueden agregar o soltar hasta $\gamma, \bar{\gamma}$ nodos en cada época mediante el proceso conservador descrito anteriormente. Dejamos el método preciso para determinar la visualización siguiente para un artículo posterior, y en lugar de eso confiamos en un oráculo de membresía que actúa como secuenciador y limitador de velocidad γ, utilizando tecnologías como Fireflies [31].

3.8 Complejidad de la Comunicación

Como la vitalidad no está garantizada para transacciones fraudulentas, enfocamos nuestro análisis de la complejidad del mensaje únicamente para el caso de transacciones virtuosas. En el caso de transacciones virtuosas, se garantiza que ambos Snowflake y Snowball terminarán después de $O(kn \log n)$ mensajes. Esto se desprende de los resultados conocidos relacionados con los algoritmos epidémicos [21], y se confirma en la Tabla 1.

La complejidad de la comunicación para Avalanche es más sutil. Permitamos que el DAG inducido por Avalanche tenga un factor de bifurcación esperado de p, correspondiente al ancho del DAG, y determinado por el algoritmo de selección principal. Dado el umbral de decisión β, una transacción que acaba de llegar al punto de decisión tendrá una progenie asociada $\mathcal{Y}$. Sea m la profundidad esperada de $\mathcal{Y}$. Si dejáramos que la red de Avalanche avanzara y luego congelamos el DAG a una profundidad y, entonces tendremos aproximadamente py vértices/transacciones, de los cuales $p(y-m)$ se deciden en expectativa. Solo pm de las transacciones recientes carecerían de la progenie requerida para una decisión. Para cada nodo, cada consulta requiere k muestras y, por lo tanto, se espera que el costo total del mensaje por transacción sea $(pky)/(p(y-m)) = ky/(y-m)$. Dado que m es una constante determinada por la región indecisa del DAG a medida que el sistema avanza constantemente, la complejidad del mensaje por nodo es $O(k)$, mientras que la complejidad total es $O(kn)$.

4. Implementación

Hemos portado por completo las transacciones de Bitcoin a Avalanche, para proporcionar un sistema de pago básico. La implementación de una criptomoneda completa implica el arranque, la acuñación, la posesión, la desposesión y el control de la inflación. Si bien tenemos soluciones para estos problemas, su discusión completa está más allá del alcance de este documento. En esta sección, nos centramos en cómo Avalanche puede soportar la primitiva transferencia de valor en el centro de las criptomonedas.

Transacciones UTXO. Además de la estructura DAG en Avalanche, se utiliza un gráfico UTXO que captura la dependencia del gasto para realizar el libro mayor para el sistema de pago. Para evitar la ambigüedad, llamamos *transacciones* a aquellas que codifican las transferencias de dinero, mientras que llamamos *vértices* a las transacciones ($T \in \mathcal{T}$) en la DAG de Avalanche.

Cada transacción representa una transferencia de dinero que toma varias entradas de las cuentas de origen y genera varias salidas a los destinos. Como un sistema basado en UTXO que mantiene un libro mayor descentralizado, los saldos se mantienen por *los resultados no gastados* de las transacciones.

Más específicamente, una transacción TX_a mantiene una lista de entradas: In_{a1}, In_{a2}, ⋯. Cada entrada tiene dos campos: la referencia a un resultado de transacción no gastado y un script de gasto. El resultado de la transacción no gastada se refiere únicamente a un resultado de una transacción realizada anteriormente. El fragmento de secuencia de comandos se adjuntará a la secuencia de comandos de la salida referida formando un cálculo completo. Por lo general, podría ser una prueba criptográfica de validez, pero también podría ser cualquier cálculo Turing-completo en general. Cada salida Out_{a1}, Out_{a2}, ⋯, contiene cierta cantidad de dinero y un script que normalmente contiene una verificación criptográfica que toma la prueba de la entrada futura y verifica la validez.

En nuestro sistema de pago, hay *direcciones* que representan diferentes cuentas mediante claves criptográficas. La clave pública se usa como la identidad para los destinatarios en los scripts de salida, mientras que la clave privada es para crear firmas en los scripts de entrada, gastando los fondos disponibles. Solo el propietario de la clave puede gastar la salida no utilizada creando una entrada con la firma en una nueva transacción.

Debido a la posibilidad de duplicar el gasto por parte del propietario de la clave privada, las criptomonedas como Bitcoin utilizan una cadena de bloques como el registro lineal para rechazar la transacción que aparece más adelante en el registro e intenta gastar algo de producción dos veces. En cambio, en nuestro sistema de pago, usamos Avalanche para resolver los conflictos de doble gasto en cada conjunto de conflictos, sin mantener un registro lineal.

Si pudiéramos asumir que cada transacción solo puede tener una entrada, la inicialización de Avalanche sería sencilla. Permitimos que cada vértice del DAG subyacente sea una transacción. El conjunto de conflicto establecido en Avalanche es el dado por un conjunto de transacciones que intentan gastar el mismo resultado no gastado. Los conjuntos de conflictos son disjuntos porque cada transacción solo tiene una entrada que gasta una salida no utilizada y, por lo tanto, pertenece exactamente a un conjunto.

Las transacciones de múltiples entradas consumen múltiples UTXO, y en Avalanche, pueden aparecer en múltiples conjuntos de conflictos. Para contabilizarlas correctamente, representamos pares de *transacciones de entrada* (por ejemplo, In_{a1}) como un vértice de Avalanche, y utilizamos la conjunción de isAccepted para todas las entradas de una transacción para garantizar que no se aceptará ninguna transacción a menos que se acepten todas sus entradas. Dado que la aceptación de cada par es significativa para el sistema de pago solo si se aceptan todos los pares de la misma transacción, podemos unir el destino de estos pares de la misma transacción mediante una única consulta agrupada: el nodo consultado solo responderá "sí" si todos los pares son muy preferidos según el DAG. Este predicado más conservador no socavará la seguridad porque el mero hecho de introducir transacciones que no reúnan fichas no aumentará el valor de confianza en el protocolo.

La Figura 17 muestra la implementación real donde el DAG se construye en la granularidad de la transacción, mientras que la Figura 18 muestra la lógica equivalente del protocolo subyacente, donde los vértices están en la granularidad de la transacción de entrada.

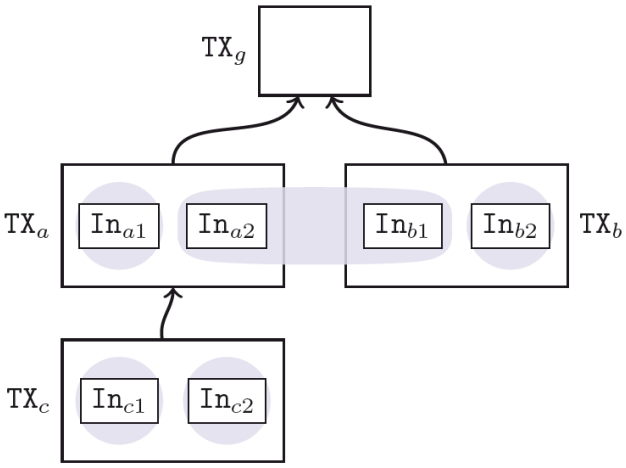

Figura 17: La implementación real de DAG en la granularidad de transición.

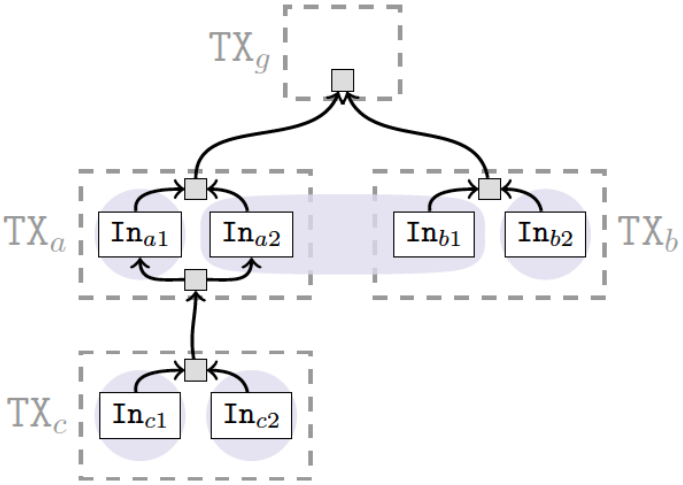

Figura 18: La estructura lógica DAG subyacente utilizada por Avalanche. Los pequeños cuadrados con sombras son vértices ficticios que solo ayudan a formar la topología DAG para mayor claridad, y pueden ser reemplazados por bordes directos. Las regiones grises redondeadas son los conjuntos de conflicto.

Siguiendo esta idea, finalmente implementamos el DAG de pares de transacciones de entrada, de modo que se puedan agrupar varias transacciones por consulta.

Selección de los padres. El objetivo del algoritmo de selección principal es producir un DAG bien estructurado que maximice la probabilidad de que la red acepte rápidamente transacciones virtuosas. Si bien este algoritmo no afecta la seguridad del protocolo, afecta a la vitalidad y desempeña un papel crucial en la determinación de la forma del DAG. Un buen algoritmo de selección de padres hace que el DAG aumente en profundidad con un "ancho" aproximadamente constante. El DAG no debe divergir como un árbol o converger en una cadena, sino que debe proporcionar concurrencia para que los nodos puedan trabajar en múltiples frentes.

Hay ventajas y desventajas inherentes en el algoritmo de selección de padres: la selección de padres bien aceptados hace que sea más probable que una transacción encuentre apoyo, pero puede llevar a la dilución de votos. Además, la selección de padres más recientes en la frontera

del DAG puede conducir a transacciones atascadas, ya que los padres pueden ser deshonestos y permanecer sin apoyo. En la siguiente discusión, ilustramos este dilema. Asumimos que cada transacción seleccionará un número pequeño, p, de padres. Nos centramos en la selección del conjunto padre elegible, desde el cual se puede elegir un subconjunto de tamaño p al azar.

Quizás la idea más simple es acuñar una nueva transacción con los padres elegidos de manera uniforme y aleatoria entre las transacciones que actualmente se prefieren más. Específicamente, podemos adoptar el predicado utilizado en la regla de votación para determinar los padres elegibles sobre los cuales un nodo votaría positivamente, de la siguiente manera:

$$\mathcal{E} = \{T : \forall T \in \mathcal{T}, \text{isStronglyPreferred}(T)\}.$$

Pero esta estrategia producirá grandes conjuntos de padres elegibles, que consisten principalmente en transacciones antiguas e históricas. Cuando un nodo muestrea las transacciones de manera uniforme desde $\mathcal{E}$, el DAG resultante tendrá un abanico de salidas cada vez mayor. Debido a que las nuevas transacciones tendrán poca progenie, el proceso de votación tomará mucho tiempo para construir la confianza requerida de cualquier nueva transacción.

En contraste, los esfuerzos para reducir el despliegue y controlar la forma del DAG seleccionando las transacciones recientes en la frontera de decisión sufren otro problema. Las transacciones más recientes tendrán una confianza muy baja, simplemente porque no tienen suficientes descendientes. Además, es posible que sus conjuntos de conflictos no estén bien distribuidos y sean bien conocidos en toda la red, lo que lleva a un archivo adjunto principal en una transacción que nunca será compatible. Esto significa que las mejores transacciones para elegir se encuentran en algún lugar cerca de la frontera, pero no demasiado profundas en la historia.

El algoritmo de selección de padres adaptativo elige a los padres comenzando en la frontera de DAG y retirándose hacia el vértice de la génesis hasta encontrar un padre elegible.

De lo contrario, el algoritmo intenta con los padres de las transacciones en $\mathcal{E}$, lo que aumenta la posibilidad de encontrar transacciones más estabilizadas a medida que se retira. Se garantiza que la búsqueda en retroceso terminará cuando llegue al vértice de la génesis. Formalmente, los padres seleccionados en este algoritmo de selección adaptativo son:

1: **function** parentSelection($\mathcal{T}$)
2: $\quad\mathcal{E}' := \{T : |\mathcal{P}_T| = 1 \vee d(T) > 0, \forall T \in \mathcal{E}\}.$
3: $\quad$**return** $\{T : T \in \mathcal{E}' \wedge \forall T' \in \mathcal{T}, T \leftarrow T', T' \notin \mathcal{E}'\}.$

Figura 19: Selección de padres adaptativa.

Optimizaciones Implementamos algunas optimizaciones para ayudar a escalar el sistema. Primero, usamos *actualizaciones perezosas* para el DAG, porque la definición recursiva de confianza puede requerir un recorrido costoso del DAG. Mantenemos el valor actual de $d(T)$ para cada vértice activo en el DAG, y lo actualizamos solo cuando un vértice descendente recibe una ficha. Dado que la ruta de búsqueda se puede recortar en vértices aceptados, el coste de una actualización es constante si los vértices rechazados tienen un número limitado de descendientes y la región indecisa del DAG permanece en un tamaño constante. En segundo lugar, el conjunto de conflictos podría ser muy grande en la práctica, porque un cliente no autorizado puede generar un gran volumen de transacciones en conflicto. En lugar de mantener una estructura de contenedor de datos para cada conjunto de conflictos, creamos una asignación de cada UTXO a la transacción preferida que señala al representante de todo el conjunto de conflictos. Esto permite a un nodo determinar rápidamente conflictos futuros y la respuesta adecuada a las consultas. Finalmente, aceleramos el proceso de consulta terminando tan pronto como se alcanza el umbral αk, sin esperar k respuestas.

5. Evaluación

Hemos implementado completamente el sistema de pago propuesto en aproximadamente 5K líneas de código C ++. En esta sección, examinamos su rendimiento, escalabilidad y latencia a través de una implementación a gran escala en Amazon AWS, y ofrecemos una comparación con los resultados conocidos de otros sistemas.

5.1 Preparación

Llevamos a cabo nuestros experimentos en Amazon EC2 ejecutando desde cientos hasta miles de instancias de máquinas virtuales. Utilizamos instancias de c5.large, que proporcionan dos núcleos de CPU virtuales por instancia que se adaptan a dos procesos, cada uno de los cuales simula un nodo individual. AWS proporciona un ancho de banda de hasta 2 Gbps, aunque el protocolo Avalanche utiliza a lo sumo 36 Mbps.

Nuestra implementación utiliza el formato de datos de transacción, la interpretación del contrato y el código de firma secp256k1 directamente desde Bitcoin 0.16. Simulamos un flujo constante de nuevas transacciones de los usuarios mediante la creación de procesos separados de clientes, cada uno de los cuales mantienen carteras separadas, generan transacciones con direcciones nuevas de destinatarios y envían las solicitudes a los nodos de Avalanche. Utilizamos varios de estos procesos de clientes para maximizar la capacidad de nuestro sistema. El número de destinatarios para cada transacción se ajusta para lograr un tamaño de transacción promedio de alrededor de 600 bytes (2–3 entradas/salidas por transacción en promedio y un tamaño de UTXO estable), el tamaño de transacción promedio actual de Bitcoin. Para utilizar la red de manera eficiente, agrupamos hasta 10 transacciones durante una consulta, pero mantenemos los valores de confianza en la granularidad de cada transacción.

Todas las métricas mostradas reflejan mediciones de extremo a extremo tomadas desde la perspectiva de todos los clientes. Es decir, los clientes examinan el número total de transacciones confirmadas por segundo para el rendimiento y, para cada transacción, restan la marca de tiempo inicial de la marca de tiempo de confirmación para la latencia. Cada experimento de rendimiento se repite 5 veces y se indica la desviación estándar en cada figura. Debido a que saturamos la capacidad del sistema en todas las ejecuciones, algunas transacciones tendrán una latencia mucho mayor que la mayoría, usamos la regla de 1.5×IQR que se usa comúnmente para filtrar los valores atípicos. Tome el experimento geo-replicado como un ejemplo, 2.5% de datos son los valores atípicos que tienen una latencia de 13 segundos en promedio. Caen fuera del rango de 1.5×IQR (aproximadamente 3σ) y se filtran. Hay muy pocos valores atípicos cuando el sistema no está saturado. Todas las latencias reportadas (incluido el máximo) son aquellas que no están filtradas. En cuanto a los parámetros de seguridad, seleccionamos $k = 10, \alpha = 0.8, \beta_1 = 11, \beta_2 = 150$, que producen un MTTF de ~10^{24} años.

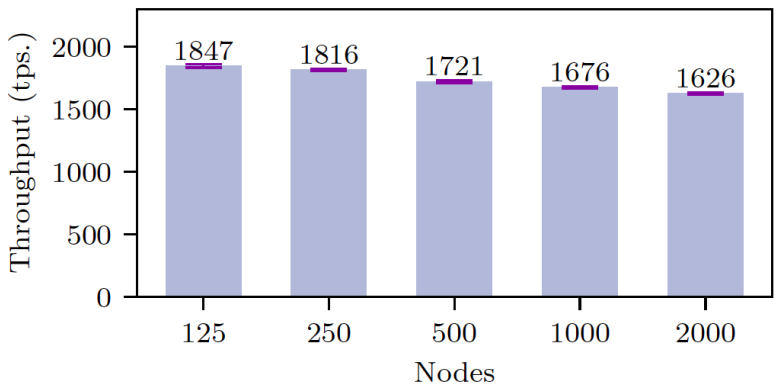

Figura 20: Rendimiento vs. tamaño de red. El eje x contiene el número de nodos y es logarítmico, mientras que el eje y son las transacciones confirmadas por segundo.

5.2 Rendimiento

Primero, medimos el rendimiento del sistema saturándolo con transacciones y examinando la tasa a la cual las transacciones se confirman en el estado estable. Para este experimento, primero ejecutamos Avalanche en 125 nodos (63 máquinas virtuales) con 10 procesos de clientes, cada uno de los cuales mantiene 300 transacciones pendientes en un momento dado.

Como se muestra en la primera barra de la Figura 20, el sistema logra más de 1800 transacciones por segundo (tps). A modo de comparación, el código criptográfico y de transacción que utilizamos de Bitcoin 0.16 solo puede generar 2.4K tps y verificar 12.3K tps, en un solo núcleo.

5.3 Escalabilidad

Para probar si el sistema es escalable en términos de la cantidad de nodos que participan en el consenso de Avalanche, ejecutamos el sistema con configuraciones idénticas y variamos la cantidad de nodos desde 125 hasta 2000.

La Figura 20 muestra que el rendimiento general se degrada entre un 10% y 1626 tps cuando la red crece en un factor de 16 a $n = 2000$. La degradación se debe al aumento de tiempo en las transacciones de susurros *(gossip transactions*). Tenga en cuenta que el eje x es logarítmico y, por lo tanto, la degradación del rendimiento es sublineal.

Mantener un orden parcial que simplemente capture las relaciones de gasto permite una mayor concurrencia en el procesamiento que un sistema de replicación de registro tolerante a errores bizantinos clásico donde todas las transacciones deben ser linealizadas. Además, la falta de un líder naturalmente evita los cuellos de botella.

5.4 Estado latente

La latencia de una transacción es el tiempo que transcurre desde el momento de su envío hasta que se confirma como aceptado. La Figura 21 muestra el histograma de distribución de latencia utilizando la misma configuración que para las mediciones de rendimiento con 2000 nodos. El eje x es el tiempo en segundos, mientras que el eje y es el porcentaje de transacciones que se finalizan dentro del período de tiempo correspondiente.

Este experimento muestra que todas las transacciones se confirman en aproximadamente 1 segundo. La Figura 21 también describe la función de distribución acumulada acumulando el número de transacciones finalizadas a lo largo del tiempo. Las latencias más comunes son alrededor de 620 ms y la varianza es baja, lo que indica que los nodos convergen en el valor final como un grupo al mismo tiempo. La línea vertical muestra la latencia máxima que hemos observado, que es de alrededor de 1.1 segundos.

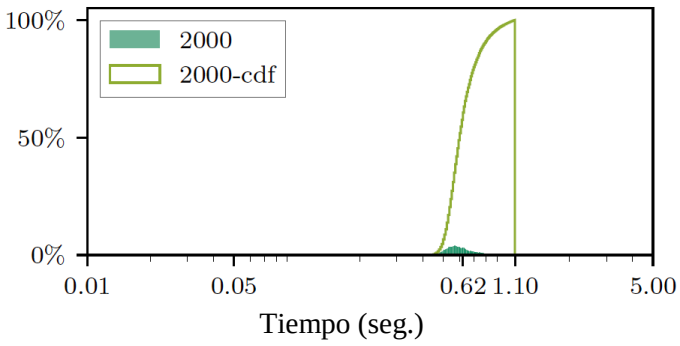

Figura 21: Histograma de latencia de transacción para $n = 2000$. El eje x es la latencia de transacción en segundos con escala de registro, mientras que el eje y es el porcentaje de transacciones que entran en el tiempo de confirmación. Las barras de histograma sólidas muestran la distribución de todas las transacciones, junto con una curva que muestra la CDF de la latencia de la transacción.

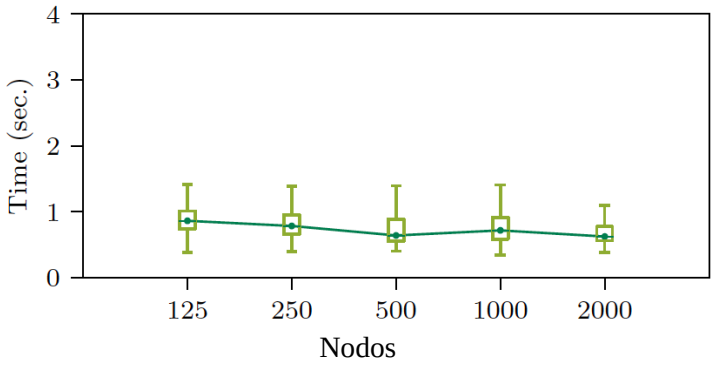

Figura 22: Latencia de transacción vs. tamaño de red.

La Figura 22 muestra las latencias de transacción para diferentes números de nodos. Los bordes horizontales de las cajas representan el mínimo, el primer cuartil, la mediana, el tercer cuartil y la latencia máxima, respectivamente, de abajo hacia arriba. De manera crucial, los datos experimentales muestran que la latencia media es más o menos independiente del tamaño de la red.

5.5 Mal comportamiento de los clientes

A continuación, examinamos cómo las transacciones fraudulentas emitidas por clientes malintencionados que duplican el gasto de los productos no gastados pueden afectar la latencia de las transacciones virtuosas creadas por otros clientes honestos.

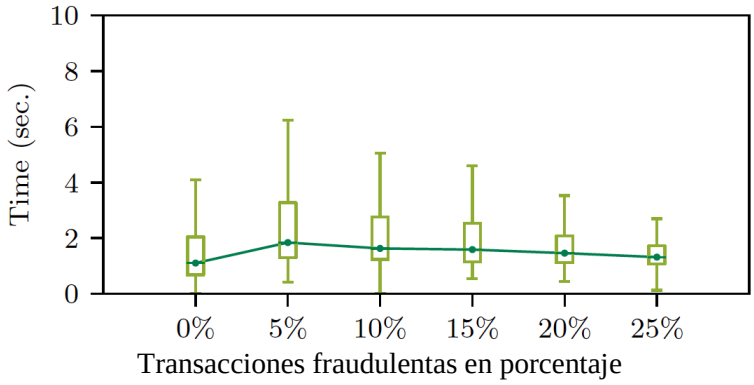

Figura 23: Latencia vs. relación de transacciones fraudulentas.

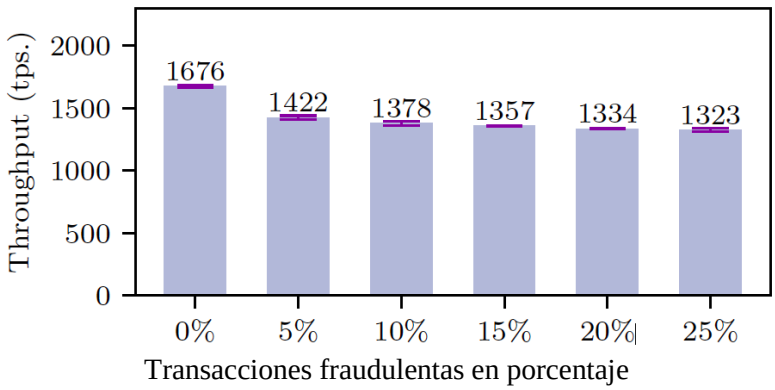

Figura 24: Rendimiento vs. relación de transacciones fraudulentas.

Adoptamos una estrategia para simular el mal comportamiento de los clientes donde una fracción (del 0% al 25%) de las transacciones pendientes entra en conflicto con algunas existentes. Los procesos del cliente logran esto al designar algunos flujos de transacciones de doble gasto entre todas las transacciones pendientes simuladas y enviar las transacciones en conflicto a diferentes nodos. Utilizamos la misma configuración con $n = 1000$ que en los experimentos anteriores, y solo medimos el rendimiento y la latencia de las transacciones confirmadas.

La latencia de Avalanche solo se ve afectada levemente por el mal comportamiento de los clientes, como se muestra en la Figura 23. Sorprendentemente, las latencias disminuyen un poco cuando aumenta el porcentaje de transacciones fraudulentas. Este comportamiento se produce porque, con la introducción de transacciones fraudulentas, se reduce el rendimiento *efectivo* general y, por lo tanto, se alivia la carga del sistema. Esto se confirma en la Figura 24, que muestra que el rendimiento (de transacciones virtuosas) disminuye con la proporción de transacciones fraudulentas. Además, la reducción en el rendimiento parece ser proporcional al número de clientes que se comportan mal, es decir, no se proporciona ningún aprovechamiento a los atacantes.

5.6 Geo-replicación

También evaluamos el sistema de pago en un escenario geo-replicado emulado con latencias significativamente más altas que en medidas anteriores. Seleccionamos 20 ciudades principales que parecen estar cerca de números sustanciales de nodos de Bitcoin alcanzables, según [10]. Las ciudades cubren América del Norte, Europa, Asia occidental, Asia oriental y Oceanía, y también cubren los 10 principales países con el mayor número de nodos accesibles. Utilizamos la latencia y la matriz de fluctuaciones rastreadas desde [53] y emulamos la latencia de paquetes de red en el kernel de Linux usando tc y netem. Los 2000 nodos se distribuyen uniformemente a cada ciudad, sin emulación de latencia de red adicional entre los nodos dentro de la misma ciudad. Asignamos un proceso de cliente a cada ciudad, manteniendo 300 transacciones pendientes por ciudad en cualquier momento.

Nuestras mediciones muestran un rendimiento promedio de 1312 tps, con una desviación estándar de 5 tps. Como se muestra en la Figura 25, la latencia mediana de la transacción es de 4.2 segundos, con una latencia máxima de 5.8 segundos.

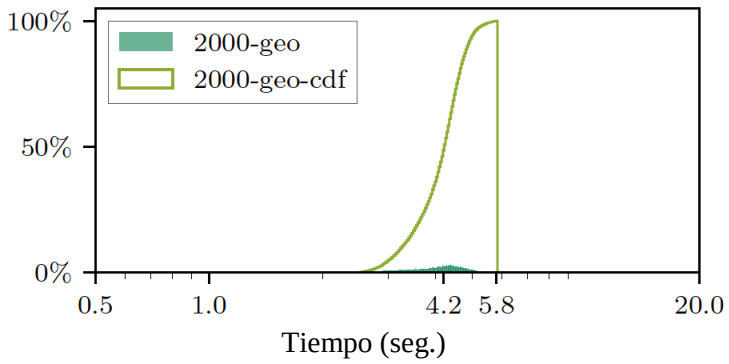

Figura 25: Histograma de latencia para $n = 2000$ en 20 ciudades.

5.7 Procesamiento por lotes

El procesamiento por lotes es una optimización crítica que puede mejorar el rendimiento al amortizar la sobrecarga de consenso sobre un mayor número de transacciones útiles. Avalanche utiliza el procesamiento por lotes por defecto para realizar 10 transacciones por evento de consulta, es decir, por vértice en el DAG.

Para probar el aumento de rendimiento de los procesos por lotes, realizamos un experimento en el que los lotes están desactivados. Sorprendentemente, el rendimiento por lotes es solo 2× tan grande como el caso no consolidado, y el aumento del tamaño del lote no aumenta el rendimiento.

La razón de esto es que la implementación se ve obstaculizada por la verificación de la transacción. Nuestra implementación actual utiliza un modelo basado en eventos para manejar una gran cantidad de mensajes simultáneos de la red. Después de comentar la función verify() en nuestro código, el rendimiento aumenta a 8K tps, lo que demuestra que la interpretación del contrato o las operaciones criptográficas involucradas en la verificación representan el principal cuello de botella para el sistema. Es posible eliminar este cuello de botella mediante la descarga de la verificación de la transacción a una GPU. Incluso sin optimización de GPU, 1312 tps está muy por encima de las cadenas de bloques existentes.

5.8 Comparación con otros sistemas

Los parámetros para nuestros experimentos se eligieron para ser comparables a Algorand [26], y utilizamos Bitcoin [40] como base.

Algorand usa una función aleatoria verificable para elegir comités, y mantiene un registro totalmente ordenado mientras que Avalanche establece solo una orden parcial. Algorand está basado en líderes y realiza un consenso por comité, mientras que Avalanche no tiene líderes. Ambas evaluaciones utilizan una red de decisión de tamaño 2000 en EC2. Nuestra evaluación utiliza c5.large con 2 vCPU, red de 2 Gbps por VM, mientras que Algorand usa m4.2xlarge con 8 vCPU, red de 1 Gbps por VM. Las CPU tienen aproximadamente la misma velocidad, y nuestro sistema no está bloqueado por la red, lo que hace posible la comparación. Los parámetros de seguridad elegidos en nuestros experimentos garantizan una probabilidad de violación de seguridad por debajo de 10^{-9} en presencia de 20% de nodos bizantinos, mientras que la evaluación de Algorand garantiza una probabilidad de violación por debajo de 5×10^{-9} con un 20% de nodos bizantinos.

El rendimiento es de 3-7 tps para Bitcoin, 364 tps para Algorand (con bloques de 10 Mbyte) y 159 tps (con bloques de 2 Mbyte). Por el contrario, Avalanche logra más de 1300 tps de manera constante en hasta 2000 nodos. En cuanto a la latencia, la finalidad es de 10 a 60 minutos para Bitcoin, unos 50 segundos para Algorand con bloques de 10 Mbytes y 22 segundos con bloques de 2 Mbytes y 4.2 segundos para Avalanche.

6. Trabajo Relacionado

Bitcoin [40] es una criptomoneda que utiliza una cadena de bloques basada en prueba de trabajo (PoW) para mantener un libro mayor de transacciones UTXO. Mientras que las técnicas basadas en prueba de trabajo [6, 23], e incluso las criptomonedas con acuñación basada en prueba de trabajo [45, 52] se han explorado antes, Bitcoin fue el primero en incorporar PoW en su proceso de consenso. A diferencia de los protocolos BFT más tradicionales, Bitcoin tiene una garantía de seguridad probabilística y asume una potencia de cómputo mayoritariamente honesta en lugar de una membresía conocida, lo que a su vez ha permitido un protocolo sin permiso a escala de Internet. Si bien no tiene permisos y es resistente a los adversarios, Bitcoin tiene un bajo rendimiento (~ 3 tps) y una alta latencia (~ 5.6 horas para una red con 20% de presencia bizantina y 2^{-32} garantía de seguridad). Además, PoW requiere una cantidad sustancial de potencia de cómputo que se consume solo con el propósito de mantener la seguridad.

Innumerables criptomonedas utilizan PoW [6, 23] para mantener un libro mayor distribuido. Al igual que Bitcoin, sufren cuellos de botella inherentes a la escalabilidad. Existen varias propuestas de protocolos que intentan utilizar mejor el esfuerzo realizado por PoW. Bitcoin-NG [24] y la versión sin permiso de Thunderella [43] utilizan un consenso como el de Nakamoto para elegir a un líder que dicta la escritura del registro replicado durante un tiempo relativamente largo para proporcionar un mayor rendimiento. Además, Thunderella proporciona un límite optimista que, con 3/4 de la potencia computacional honesta y un líder electo honesto, permite que las transacciones se confirmen rápidamente. ByzCoin [34] selecciona periódicamente un pequeño

grupo de participantes y luego ejecuta un protocolo similar a PBFT dentro de los nodos seleccionados. Consigue un rendimiento de 393 tps con aproximadamente 35 segundos de latencia.

Otra categoría de protocolos blockchain propone deshacerse del PoW y reemplazarlo con Proof-of-Stake (PoS). PoS elimina el coste computacional del PoW al requerir que un nodo ponga su dinero en espera a cambio de la participación en el consenso. Los protocolos de consenso basados en PoS reemplazan la potencia de cómputo de la mayoría honesta por los valores de participación de la mayoría honesta. Snow White [19] y Ouroboros [33] son algunos de los primeros protocolos de PoS con seguridad comprobada. Ouroboros utiliza un protocolo seguro de lanzamiento de monedas de múltiples partes para producir aleatoriedad para la elección del líder. El protocolo de seguimiento, Ouroboros Praos [20] proporciona seguridad en presencia de adversarios totalmente adaptables.

Los protocolos basados en el acuerdo bizantino [36,44] suelen hacer uso de quórums y requieren un conocimiento preciso de la membresía. PBFT [14], un representante bien conocido, requiere un número cuadrático de intercambios de mensajes para llegar a un acuerdo. El protocolo Q/U [3] y la replicación HQ [17] utilizan un enfoque basado en quórum para optimizar los casos de operación sin conflictos para lograr el consenso en una sola ronda de comunicación. Sin embargo, aunque estos protocolos mejoran el rendimiento, se degradan muy pobremente en las discusiones. Zyzzyva [35] combina BFT con ejecución especulativa para mejorar el caso de operación libre de fallos. Aliph [28] introduce un protocolo con rendimiento optimizado en varios casos de ejecución, en lugar de uno solo. En contraste, Ardvark [16] sacrifica algo de rendimiento para tolerar la degradación del peor de los casos, proporcionando un perfil de ejecución más uniforme. Este trabajo, en particular, sacrifica las optimizaciones libres de fallos para proporcionar un rendimiento consistente incluso ante un gran número de fallos. Trabajos anteriores en sistemas BFT con permiso generalmente requieren al menos $2b + 1$ réplicas de tolerancia. Sin embargo, CheapBFT [32] mostró cómo aprovechar los componentes de hardware de confianza para construir un protocolo que utiliza $b + 1$ réplicas.

Otros trabajos intentan introducir nuevos protocolos bajo redefiniciones y relajaciones del modelo BFT. BFT a gran escala [46] modifica PBFT para permitir la elección arbitraria del número de réplicas y el umbral de error, proporcionando una garantía probabilística de vitalidad para alguna relación de error pero protegiendo la seguridad con una alta probabilidad. En otra forma de relajación, Zeno [48] introduce un protocolo de replicación de máquina de estado BFT que cambia la consistencia por una alta disponibilidad. Más específicamente, este documento garantiza la consistencia eventual en lugar de la linealización, lo que significa que los participantes pueden ser inconsistentes, pero eventualmente llegar a un acuerdo una vez que la red se estabilice. Al proporcionar una garantía de consistencia aún más débil, llamada consistencia causal de bifurcación-unión, Depot [37] describe un protocolo que garantiza la seguridad en $b + 1$ réplicas.

NOW [27] es, según nuestro entendimiento, el primero en utilizar la idea de los sub-quórums para generar instancias más pequeñas de consenso. La idea de este documento es que se pueden extraer pequeños quórums de tamaño logarítmico de un conjunto potencialmente grande de nodos en la red, lo que permite que instancias más pequeñas de protocolos de consenso se ejecuten en paralelo.

HoneyBadger [39] proporciona una buena vitalidad en una red con latencias heterogéneas y logra más de 341 tps con una latencia de 5 minutos en 104 nodos. Tendermint [11] rota el líder para cada bloque y se ha demostrado con hasta 64 nodos. Ripple [47] tiene una latencia baja al utilizar subredes de confianza colectiva en una red grande. La empresa Ripple proporciona una lista predeterminada de nodos de confianza que cambia lentamente, lo que hace que el sistema sea esencialmente centralizado. En la configuración sincronizada y autenticada, el protocolo en [4] logra una confirmación constante de 3 a 3 vueltas, a costa de una complejidad cuadrática de los mensajes.

Stellar [38] utiliza el acuerdo bizantino federado en el que los *pedazos de quórum* permiten una confianza heterogénea para diferentes nodos. La seguridad está garantizada cuando las transacciones se pueden conectar de forma transitoria mediante pedazos de quórum de confianza.

Algorand [26] utiliza una función aleatoria verificable para seleccionar un comité de nodos que participan en un nuevo protocolo de consenso bizantino. Alcanza más de 360 tps con una latencia de 50 segundos en una red emulada de 2000 nodos de comité (500K usuarios en total) distribuidos entre 20 ciudades. Para evitar los ataques Sybil, utiliza un mecanismo como el Proof-of-Stake que asigna ponderaciones a los participantes en la selección del comité en función del dinero en sus cuentas.

Algunos protocolos utilizan una estructura de Gráficos Acíclicos Dirigidos (DAG) en lugar de una cadena lineal para lograr un consenso. En lugar de elegir la cadena más larga como en Bitcoin, GHOST [50] utiliza una regla de selección de cadena más eficiente que permite tener en cuenta las transacciones que no están en la cadena principal, lo que aumenta la eficiencia. SPECTRE [49] utiliza transacciones en el DAG para votar recursivamente con PoW para lograr un consenso, seguido por PHANTOM [51] que logra un orden lineal entre todos los bloques. Avalanche es diferente de la forma en que el resultado de la votación es una ficha de una sola vez que está determinada por una consulta, mientras que los votos en PHANTOM están puramente determinados por la estructura de la transacción. Similar a Thunderella, Meshcash [9] combina un protocolo lento basado en PoW con un protocolo de consenso rápido que permite una alta tasa de bloques independientemente de la latencia de la red, ofreciendo un tiempo de confirmación rápido. Hashgraph [7] es un protocolo sin líder que crea un DAG a través de susurros aleatorios. Requiere un conocimiento completo de la membresía en todo momento y, al igual que el Ben-Or [8], sufre al menos de una complejidad de mensaje exponencial [5, 13].

7. Conclusión

Este documento introdujo una nueva familia de protocolos BFT sin líder, metaestables y sin PoW. Estos protocolos no incurren en costes de mensaje cuadráticos y pueden funcionar sin un conocimiento preciso de la membresía. Son ligeros, inactivos y ofrecen una sólida garantía de seguridad, aunque logran estas propiedades al no garantizar vitalidad para transacciones en conflicto. Hemos ilustrado cómo se pueden usar para implementar un sistema de pago similar a Bitcoin, que alcanza 1300tps en una configuración geo-replicada.

Referencias

[1] Crypto-currency market capitalizations. https://coinmarketcap.com. Accessed: 2017-02.

[2] Snowflake to Avalanche: A metastable protocol family for cryptocurrencies. https://www.dropbox.com/s/t5h2weaws1vo3c7/paper.pdf, 2018.

[3] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable byzantine fault-tolerant services. In ACM SIGOPS Operating Systems Review, volume 39, pages 59-74. ACM, 2005.

[4] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren. Efficient synchronous byzantine consensus. 2017.

[5] J. Aspnes. Randomized protocols for asynchronous consensus. Distributed Computing, 16(2-3):165–175, 2003.

[6] J. Aspnes, C. Jackson, and A. Krishnamurthy. Exposing computationally-challenged byzantine impostors. 2005.

[7] L. Baird. Hashgraph consensus: fair, fast, byzantine fault tolerance. Technical report, Swirlds Tech Report, 2016.

[8] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In Proceedings of the second annual ACM symposium on Principles of distributed computing, pages 27–30. ACM, 1983.

[9] I. Bentov, P. Hubácek, T. Moran, and A.Nadler. Tortoise and Hares Consensus: the Meshcash framework for incentive-compatible, scalable cryptocurrencies. IACRCryptology ePrint Archive, 2017:300, 2017.

[10] Bitnodes. Global Bitcoin nodes distribution. https://bitnodes.earn.com/. Accedido: 2018-04.

[11] E. Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. PhD thesis, 2016.

[12] M. Burrows. The chubby lock service for looselycoupled distributed systems. In 7th Symposium on Operating Systems Design and Implementation (OSDI'06), November 6-8, Seattle, WA, USA, pages 335–350, 2006.

[13] C. Cachin and M. Vukolic. Blockchain consensus protocols in the wild. CoRR, abs/1707.01873, 2017.

[14] M. Castro and B. Liskov. Practical byzantine fault tolerance. In Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999, pages 173–186, 1999.

[15] Central Intelligence Agency. The world factbook. https://www.cia.gov/library/ publications/the-world-factbook/geos/da.html. Accedido: 2018-04.

[16] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA, pages 153–168, 2009.

[17] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. Hq replication: A hybrid quorum protocol for byzantine fault tolerance. In Proceedings of the 7th symposium on Operating systems design and implementation, pages 177-190. USENIX Association, 2006.

[18] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains - (a position paper). In Financial Cryptography and Data Security - FC 2016 InternationalWorkshops, BITCOIN,VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers, pages 106-125, 2016.

[19] P. Daian, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. https://eprint.iacr.org/2016/919.

[20] B. David, P. Gazi, A. Kiayias, and A.Russell. Ouroboros Praos: An adaptively-secure, semisynchronous proof-of-stake blockchain. In Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II, pages 66–98, 2018.

[21] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, pages 1–12. ACM, 1987.

[22] Digiconomist. Bitcoin energy consumption index. https://digiconomist.net/bitcoinenergy-consumption. Accessed: 2018-04.

[23] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings, pages 139–147, 1992.

[24] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse. Bitcoin-ng: A scalable blockchain protocol. In 13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016, pages 45–59, 2016.

[25] J. A. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and applications. In Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II, pages 281–310, 2015.

[26] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017, pages 51– 68, 2017.

[27] R. Guerraoui, F. Huc, and A.-M.Kermarrec. Highly dynamic distributed computing with byzantine failures. In Proceedings of the 2013 ACM symposium on Principles of distributed computing, pages 176–183. ACM, 2013.

[28] R. Guerraoui, N. Kneževic, V. Quéma, and M. Vukolic. The next 700 bft protocols. In Proceedings of the 5th European conference on Computer systems, pages 363–376. ACM, 2010.
[29] W. Hoeffding. Probability inequalities for sums of bounded random variables. Journal of the American statistical association, 58(301):13–30, 1963.

[30] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internetscale systems. In 2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010, 2010.

[31] H. D. Johansen, R. van Renesse, Y. Vigfusson, and D. Johansen. Fireflies: A secure and scalable membership and gossip service. ACM Trans. Comput. Syst., 33(2):5:1–5:32, 2015.

[32] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel. Cheapbft: resource-efficient byzantine fault tolerance. In Proceedings of the 7th ACM european conference on Computer Systems, pages 295–308. ACM, 2012.

[33] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Advances in Cryptology - CRYPTO2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I, pages 357-388, 2017.

[34] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin security and performance with strong consistency via collective signing. In 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., pages 279–296, 2016.

[35] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. L. Wong. Zyzzyva: Speculative byzantine fault tolerance. ACM Trans. Comput. Syst., 27(4):7:1-7:39, 2009.

[36] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. ACM Trans. Program. Lang. Syst., 4(3):382–401, 1982.

[37] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud storage with minimal trust. ACMTransactions on Computer Systems (TOCS), 29(4):12, 2011.

[38] D. Mazieres. The Stellar consensus protocol: A federated model for internet level consensus. Stellar Development Foundation, 2015.

[39] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The Honey Badger of BFT protocols. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, pages 31-42, 2016.

[40] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[41] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II, pages 643–673, 2017.

[42] R. Pass and E. Shi. Fruitchains: A fair blockchain. IACR Cryptology ePrint Archive, 2016:916, 2016.

[43] R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. In Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II, pages 3–33, 2018.

[44] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. J. ACM, 27(2):228–234, 1980.

[45] R. Rivest and A. Shamir. Payword and micromint: Two simple micropayment schemes. In Security protocols, pages 69–87. Springer, 1997.

[46] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee. Large-scale byzantine fault tolerance: Safe but not always live.

[47] D. Schwartz, N. Youngs, A. Britto, et al. The Ripple protocol consensus algorithm. Ripple Labs Inc White Paper, 5, 2014.

[48] A. Singh, P. Fonseca, P. Kuznetsov, R. Rodrigues, P. Maniatis, et al. Zeno: Eventually consistent byzantine-fault tolerance.

[49] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. IACR Cryptology ePrint Archive, 2016:1159, 2016.

[50] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in Bitcoin. In Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers, pages 507–527, 2015. [51] Y. Sompolinsky and A. Zohar. PHANTOM: A scalable blockdag protocol. IACR Cryptology ePrint Archive, 2018:104, 2018.

[52] V.Vishnumurthy, S. Chandrakumar, and E. G. Sirer. Karma: A secure economic framework for peer-topeer resource sharing.

[53] WonderNetwork. Global ping statistics: Ping times between wondernetwork servers. https://wondernetwork.com/pings.
Accessed: 2018-04.21