O Emaranhado

Serguei Popov* Tradução por Marco Aurélio Jr†

25 de outubro de 2017. Versão 1.4

Resumo

Neste trabalho, analisamos os fundamentos matemáticos da IOTA, uma criptomoeda para a indústria de Internet das Coisas (IoT). A principal característica desta nova criptomoeda é o *emaranhado*, um grafo acíclico dirigido (DAG) para armazenar transações. O emaranhado naturalmente sucede a blockchain como seu próximo passo evolutivo, e oferece características que são requiridas para estabelecer um sistema de micropagamento máquina-máquina.

Uma contribuição essencial deste artigo é uma família de algoritmos de Monte Carlo via Cadeias de Markov (MCMC). Esses algoritmos selecionam locais de ligação no emaranhado para uma transação que acabou de chegar.

1 Introdução e descrição do sistema

A ascensão e sucesso do Bitcoin durante os últimos seis anos provaram que a tecnologia blockchain tem valor no mundo real. No entanto, esta tecnologia também tem uma série de inconvenientes que impedem que ele seja usado como uma plataforma genérica para criptomoedas em todo o mundo. Uma desvantagem notável é o conceito de uma taxa de transação para transações de qualquer valor. A importância dos micropagamentos aumentará no desenvolvimento rápido da indústria de Internet das Coisas, e pagar uma taxa que é *maior* do que a quantidade de valor sendo transferida não é lógico. Além disso, não é fácil se livrar das taxas na infraestrutura blockchain, uma vez que servem de incentivo para os criadores de blocos. Isso leva a outro problema com a tecnologia de criptomoeda existente, nomeadamente a

*Também conhecido como `mthcl`; Informação de contato do autor: `serguei.popov@iota.org`

†Informação de contato do tradutor: `portellademacedo@hotmail.com`

natureza heterogênea do sistema. Existem dois tipos distintos de participantes no sistema, aqueles que emitem transações e aqueles que aprovam transações.O desenho deste sistema cria uma discriminação inevitável de alguns participantes, que por sua vez, cria conflitos que fazem com que todos os elementos gastem recursos na resolução de disputas. Os problemas acima mencionados justificam uma busca por soluções essencialmente diferentes da tecnologia blockchain, a base para Bitcoin e muitas outras criptomoedas.

Neste artigo, nós discutimos uma abordagem inovadora que não incorpora tecnologia blockchain. Essa abordagem está sendo implementada atualmente em uma criptomoeda chamada *iota* [1], que foi desenhada especificamente para a indústria de IoT. O objetivo deste trabalho é focar nas características gerais do emaranhado, e discutir problemas que surgem quando se tenta se livrar da blockchain e manter um livro-razão distribuído. A implementação concreta do protocolo iota não é discutida.

Em geral, uma criptomoeda baseada em emaranhado funciona da seguinte maneira. Ao invés da blockchain global, há um DAG que nós chamamos de *emaranhado*. As transações emitidas pelos nós constituem o conjunto de locais do grafo do emaranhado, que é o livro-razão para armazenar transações. O conjunto de bordas do emaranhado é obtido da seguinte forma: quando uma nova transação chega, ela deve *aprovar* duas[1] transações anteriores. Essas aprovações são representadas por bordas dirigidas, como mostrado na Figura 1[2]. Se não há uma borda dirigida entre a transação A e a transação B, mas há um caminho dirigido de comprimento de pelo menos dois de A para B, nós dizemos que A *aprova indiretamente* B. Há também a transação "gênese", que é aprovada direta ou indiretamente por todas as outras transações (Figura 2). A gênese é descrita do seguinte modo. No começo do emaranhado, havia um endereço com um saldo que continha todos os tokens. A transação gênese enviou estes tokens para vários outros endereços "fundadores". Vamos salientar que todos os tokens foram criados na transação gênese. Nenhum token será criado no futuro, e não haverá mineração no sentido que mineradores recebam recompensas monetárias "surgidas do nada".

Uma nota rápida sobre terminologia: *locais* são transações representadas no grafo do emaranhado. A rede é composta de *nós*; isto é, nós são entidades que emitem e validam transações.

A ideia principal do emaranhado é a seguinte: para emitir uma transação, usuários devem trabalhar para aprovar outras transações. Portanto, usuários que emitem

[1]Esta é a abordagem mais simples. Pode-se também estudar sistemas similares onde as transações devem aprovar k outras transações para um geral $k \geq 2$, ou ter um conjunto de regras inteiramente diferente.

[2]O tempo sempre aumenta da esquerda para a direita em cada figura.

uma transação estão contribuindo para a segurança da rede. Assume-se que os nós verificam se as transações aprovadas não são conflitantes. Se um nó descobre que uma transação está em conflito com a história do emaranhado, o nó não aprovará a transação conflitante de uma maneira direta ou indireta[3].

Conforme uma transação recebe aprovações adicionais, ela é aceita pelo sistema com um nível maior de confiança. Em outras palavras, será difícil fazer o sistema aceitar uma transação de gasto duplo. É importante observar que nós não *impomos* quaisquer regras para escolha de quais transações um nó irá aprovar. Ao invés disso, nós argumentamos que se um número grande de nós seguir alguma regra de "referência", então para qualquer nó fixo é melhor aderir a uma regra do mesmo tipo[4]. Isso parece um suposição razoável, especialmente no contexto da IoT, onde os nós são chips especializados com firmware pré-instalado.

Para emitir uma transação, um nó faz o seguinte:

- O nó escolhe duas outras transações para aprovar de acordo com um algoritmo. Em geral, estas duas transações podem coincidir.

- O nó checa se as duas transações não estão em conflito, e não aprova transações conflitantes.

- Para um nó emitir uma transação válida, ele deve solucionar um enigma criptográfico similar aqueles na blockchain do Bitcoin. Isto é alcançado ao encontrar um a nonce tal que o hash deste nonce concatenado com alguns dados da transação aprovada tenha uma forma específica. No caso do protocolo Bitcoin, o hash deve ter nos seus primeiros algarismos pelo menos um número pré-definido de zeros.

É importante observar que a rede da iota é assíncrona. em geral, os nós não enxergam necessariamente o mesmo conjunto de transações. Também deve ser notado que o emaranhado pode conter transações conflitantes Os nós não têm que chegar em um consenso sobre quais transações[5] válidas têm o direito de estar no livro-razão, significando que todas elas podem estar no emaranhado. Entretanto, no caso onde há transações conflitantes, os nós precisam decidir quais transações se tornarão órfãs[6]. A regra principal que os nós usam para decidir entre duas transações conflitantes é

[3]Se um nó emite uma nova transação que aprova transações conflitantes, então corre o risco de outros nós não aprovarem sua transação, que cairá no esquecimento.

[4]Nós comentamos mais sobre isso ao final da Seção 4.1

[5]Transações que foram emitidas de acordo com o protocolo.

[6]Transações órfãs não são mais indiretamente aprovadas pelas transações recebidas

a seguinte: um nó roda o algoritmo de seleção de ponta[7] (cf. Seção 4.1) várias vezes, e vê qual das duas transações é mais provável que seja indiretamente aprovada pela ponta selecionada. Por exemplo, se uma transação foi selecionada 97 vezes durante 100 execuções do algoritmo de seleção de ponta, nós dizemos que ela é confirmada com 97% de confiança.

Iremos comentar também a seguinte pergunta (cf. [4]): O que motiva os nós a propagar transações? Cada nó calcula alguma estatísticas, uma das quais é quantas novas transações são recebidas de um vizinho. Se um nó em particular é "muito preguiçoso", ele será abandonado pelos seus vizinhos. Assim sendo, mesmo se um nó não emitir transações, e portanto, não tem incentivo direto para compartilhar novas transações que aprovem sua própria transação, ainda tem incentivo para participar.

Após introduzir alguma notação na Seção 2, nós discutimos algoritmos para escolher as duas transações para aprovar, as regras para medir a aprovação geral da transação (Seção 3, especialmente Seção 3.1), e possíveis cenários de ataque (Seção 4). Também, no improvável evento de que o leitor esteja assustado com as fórmulas, ele pode pular diretamente para as "conclusões" ao final de cada seção.

Deve-se notar que a ideia de usar DAGs no campo das criptomoedas existe há algum tempo, vide [3, 6, 7, 9, 12]. Especificamente, [7] introduz o protocolo GHOST, que propõe uma modificação do protocolo do Bitcoin ao fazer do livro-razão principal uma árvore ao invés da blockchain. É mostrado que tal modificação reduz os tempos de confirmação e melhora a segurança geral da rede. No artigo [9] os autores consideram um modelo de criptomoeda baseada em DAG. O modelo deles é diferente do nosso pelas seguintes razões: os locais do DAG deles são blocos ao invés de transações individuais; Os mineradores no sistema deles competem por taxas de transação; E novos tokens podem ser criados pelos mineradores de blocos. Além disso, observe que uma solução semelhante à nossa foi proposta em [6], embora ela não discuta nenhuma estratégia de aprovação de ponta em particular. Após a primeira versão deste artigo ser publicada, vários outros trabalhos que objetivaram criar um livro-razão distribuído baseado em DAG têm aparecido, e.g. [8]. We também referenciamos outra abordagem [2, 10] que tem por objetivo tornar micropagamentos possíveis no Bitcoin ao estabelecer canais de pagamento ponto-a-ponto.

[7]Como mencionado acima, há uma boa razão para assumir que outros nós irão seguir o mesmo algoritmo para seleção de ponta.

2 Pesos e mais

Nesta seção nos definimos o peso de uma transação, e conceitos relacionados. O peso de uma transação é proporcional a quantidade de trabalho que o nó emitente investiu nela. Na implementação atual da iota, o peso somente pode assumir valores 3^n, onde n é um número inteiro positivo que pertence a algum intervalo não-vazio de valores aceitáveis[8]. De fato, é irrelevante saber como o peso foi obtido na prática. É importante apenas que todas as transações tem um número inteiro positivo, seu peso, ligado a ela. Em geral, a ideia é que uma transação com um peso maior é mais "importante" do que uma transação com um peso menor. Para evitar envio de spam e outros estilos de ataque, é assumido que nenhuma entidade pode gerar uma abundância de transações com pesos "aceitáveis" em um curto período de tempo.

Uma das noções de que precisamos é o *peso acumulado* de uma transação: é definido como o próprio peso de uma transação em particular, mais a soma de pesos próprios de todas as transações que aprovam direta ou indiretamente esta transação. O algoritmo para cálculo do peso acumulado é ilustrado na Figura 1. As caixas representam transações, e o pequeno número no canto SE de cada caixa denota o peso próprio, e o número negrito indica o peso acumulado. Por exemplo, a transação F é direta ou indiretamente aprovada pelas transações A, B, C, E. O peso acumulado de F é $9 = 3+1+3+1+1$, o qual é a soma do peso próprio de F e os pesos próprios de A, B, C, E.

Vamos definir "pontas" como transações não aprovadas no grafo do emaranhado. No retrato superior do emaranhado da Figura 1, as únicas pontas são A e C. Quando a nova transação X chega e aprova A e C no retrato inferior do emaranhado, X se torna a única ponta. O peso acumulado de todas as outras transações aumenta em 3, o peso próprio de X.

Nós necessitamos introduzir duas variáveis adicionais para a discussão de algoritmos de aprovação. Primeiro, para um local de transação no emaranhado, nós introduzimos sua

- *altura*: o comprimento do caminho mais longo orientado para a gênese;
- *profundidade*: o comprimento do caminho reverso mais longo até alguma ponta.

Por exemplo, G tem altura 1 e profundidade 4 na Figura 2 porque o caminho reverso F, D, B, A, enquanto D tem altura 3 e profundidade 2. Além disso, vamos apresentar a noção de *pontuação*. Por definição, a pontuação de uma transação é a

[8]Este intervalo também deve ser finito — ver o "ataque de grande peso" na Seção 4.

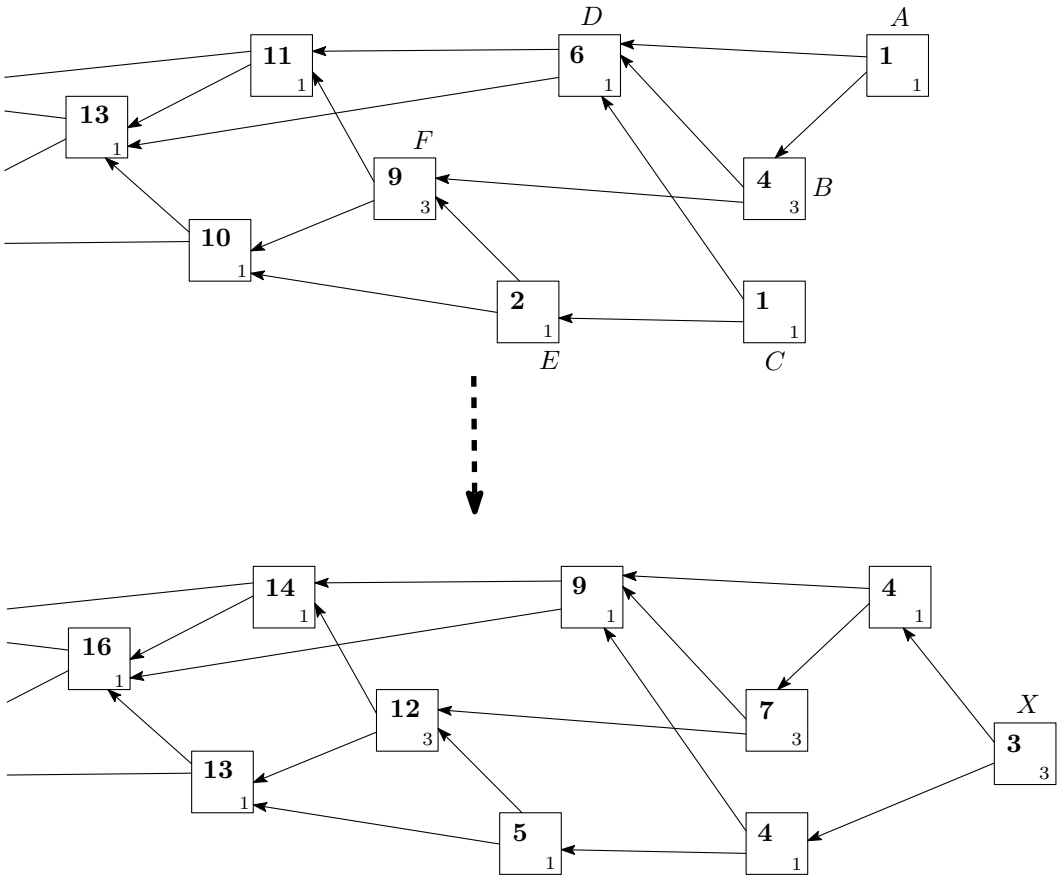

Figura 1: DAG com atribuições de peso antes e depois de uma transação recém-emitida, X. As caixas representam transações, o número pequeno no canto SE de cada caixa denota o peso próprio, e o número denota o peso acumulado.

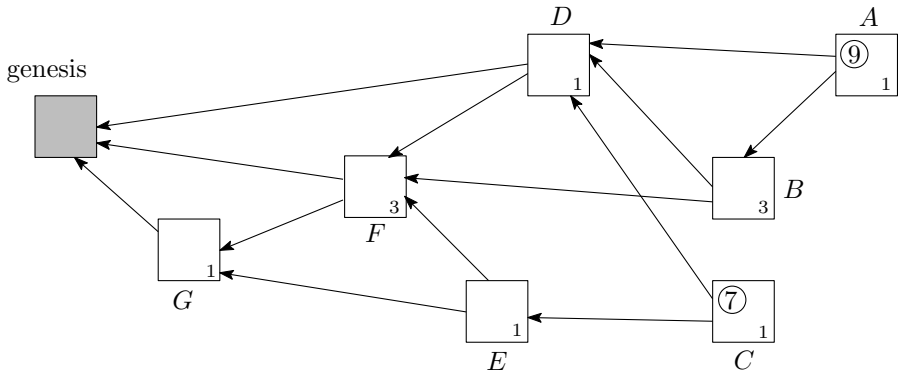

Figura 2: DAG com pesos próprios atribuídos a cada local, e pontuações calculadas para locais A e C.

soma dos próprios pesos de todas as transações aprovadas por esta transação mais o próprio peso da transação propriamente dita. Na Figura 2, as únicas pontas são A e C. A transação A direta or indiretamente aprova as transações B, D, F, G, então a pontuação de A é $1 + 3 + 1 + 3 + 1 = 9$. De forma análoga, a pontuação de C é $1 + 1 + 1 + 3 + 1 = 7$.

Para entender os argumentos apresentados neste artigo, pode-se assumir com segurança que todas as transações têm um peso próprio igual a 1. *De agora em diante, aderimos a essa suposição.* Sob este pressuposto, o peso acumulado da transação X se torna 1 mais o número de transações que direta ou indiretamente aprovam X, e a pontuação se torna 1 mais o número de transações que são direta ou indiretamente aprovadas por X.

Notemos que, entre aqueles definidos nesta seção, o peso acumulado é (de longe!) a métrica mais importante, embora altura, profundidade, e pontuação entrarão brevemente em algumas discussões também.

3 Estabilidade do sistema, e vértices de corte

Deixe $L(t)$ ser o número total de pontas no sistema no tempo t. Espera-se que o processo estocástico $L(t)$ permaneça *estável*[9]. Mais precisamente, espera-se que o processo seja *recorrente positivo*, ver Seções 4.4 e 6.5 do [11] para definições formais. Em particular, a recorrência positiva implica que o limite de $\mathbb{P}[L(t) = k]$ como $t \to \infty$ deve existir e ser positivo para todos $k \geq 1$. Intuitivamente, esperamos que $L(t)$ deva

[9]Sob uma suposição adicional de que o processo é homogêneo no tempo.

flutuar em torno de um valor constante, e não escapar para o infinito. Se $L(t)$ fosse escapar para o infinito, muitas transações não aprovadas seriam deixadas para trás.

Para analisar as propriedades de estabilidade de $L(t)$, nós precisamos fazer algumas suposições. Um pressuposto é que transações são emitidas por um grande número de entidades grosso modo independentes, então o processo de recebimento de transações pode ser modelado por um processo de ponto de Poisson (cf. e.g. Seção 5.3 do [11]). Deixe λ ser a taxa desse processo de Poisson. Por simplicidade, vamos assumir que essa taxa permanece constante no tempo. Suponha que todos os dispositivos tenham aproximadamente o mesmo poder computacional, e deixe h ser o tempo médio que um dispositivo precisa para realizar cálculos que são requeridos para emitir uma transação. Então, vamos *assumir* que todos os nós se comportam da seguinte maneira: para emitir uma transação, um nó escolhe duas pontas aleatoriamente e aprova elas. Deve-se observar que, em geral, *não* é uma boa ideia para os nós "honestos" adotar esta estratégia porque tem uma série de desvantagens práticas. Em particular, não oferece proteção suficiente contra nós "preguiçosos" ou maliciosos (ver Seção 4.1 abaixo). Por outro lado, ainda consideramos este modelo, uma vez que é simples de analisar, e pode fornecer insights sobre o comportamento do sistema para estratégias de seleção de ponta mais complicadas.

Em seguida, fazemos uma nova suposição simplificadora de que qualquer nó,no momento em que ele emite uma transação, observa não o estado atual do emaranhado, mas exatamente h unidades de tempo atrás. Isso significa, em particular, que uma transação ligada ao emaranhado no tempo t somente se torna visível para a rede no tempo $t + h$. Nos também assumimos que o número de pontas permanece grosso modo estacionário no tempo, e é concentrado em torno de um número $L_0 > 0$. Na sequência, iremos calcular L_0 como uma função do λ e h.

Observe que, em um dado instante t nós temos aproximadamente λh "pontas escondidas" (que estavam anexadas no intervalo de tempo $[t-h, t)$ e ainda não estavam visíveis para a rede); também, assumir que normalmente existem r "pontas reveladas" (que estavam anexadas antes do tempo $t-h$), então $L_0 = r+\lambda h$. Por estacionariedade, podemos então assumir que, no momento t também existem λh locais que eram pontas no momento $t - h$, mas não são mais pontas. Agora, pense em uma nova transação que vem neste momento; então, uma transação que ele escolhe aprovar é uma ponta com probabilidade $r/(r + \lambda h)$ (uma vez que existem r pontas conhecidas pelo nó que emitiu a transação, e também há λh transações que não são mais pontas, embora que o nó pense que são), então o número médio de pontas escolhidas é $2r/(r + \lambda h)$. A observação chave agora é que, no regime estacionário, este número médio de pontas escolhidas deve ser igual a 1, uma vez que, em média, uma nova transação não deve mudar o número de pontas. Resolvendo a equação $2r/(r + \lambda h) = 1$ em relação a r,

nós obtemos $r = \lambda h$, e então

$$L_0 = 2\lambda h. \tag{1}$$

Nós também notamos que, se a regra é que uma nova transação referencia k transações ao invés de 2, então um cálculo semelhante dá

$$L_0^{(k)} = \frac{k\lambda h}{k-1}. \tag{2}$$

Isto é, claro, consistente com o fato de que $L_0^{(k)}$ deveria tender a λh como $k \to \infty$ (basicamente, as únicas pontas seriam aquelas ainda desconhecidos para a rede).

Além disso, (retornamos ao caso de duas transações para aprovar) o tempo esperado para uma transação ser aprovada pela primeira vez é aproximadamente $h + L_0/2\lambda = 2h$. Isso ocorre porque, pela nossa suposição, durante as primeiras h unidades de tempo uma transação não pode ser aprovada, e após que o fluxo de Poisson de aprovações a ele tem taxa de aproximadamente $2\lambda/L_0$. (Recordar Proposição 5.3 do [11], que diz que se classificarmos de forma independente cada evento de um processo de Poisson de acordo com uma lista de subtipos possíveis, então os processos de eventos de cada subtipo são processos independentes de Poisson.)

Observe que[10] em qualquer tempo fixo t o conjunto de transações que eram pontas em algum momento $s \in [t, t + h(L_0, N)]$ tipicamente constituem um *vértice de corte*. Qualquer caminho de uma transação emitida no horário $t' > t$ para a gênese deve passar através deste conjunto. É importante que o tamanho de um novo vértice de corte no emaranhado ocasionalmente torna-se pequeno. Pode-se então usar os pequenos vértices de cortes como pontos de verificação para possíveis podas no DAG e outras tarefas.

É importante observar que a estratégia "puramente aleatória" acima não é muito boa na prática porque não encoraja a aprovação de pontas. Um usuário "preguiçoso" poderia sempre aprovar um par fixo de transações muito antigas, portanto, não contribuindo para a aprovação de transações mais recentes, sem ser punido por tal comportamento[11]. Além disso, uma entidade maliciosa pode inflar artificialmente o número de pontas através da emissão de muitas transações que aprovam um par fixo de transações. Isso tornaria possível que as transações futuras selecionem estas pontas com uma probabilidade muito alta, efetivamente abandonando as pontas pertencentes aos nós "honestos". Para evitar problemas desse tipo, é preciso adotar

[10] Pelo menos no caso em que os nós *tentam* aprovar pontas.

[11] Nós lembramos ao leitor que nós não tentamos *reforçar* nenhuma estratégia de seleção de ponta em particular. Um atacante pode escolher pontas de qualquer maneira que achar conveniente.

um estratégia que é tendenciosa para as “ melhores ” pontas. Um examplo de tal estratégia é apresentado na Seção 4.1 abaixo[12].

Antes de iniciar a discussão sobre o tempo esperado para uma transação receber sua primeira aprovação, observe que nós podemos distinguir dois regimes (Figura 3).

- Carga baixa: o número típico de pontas é pequeno e freqüentemente se torna 1. Isso pode acontecer quando o fluxo de transações é tão pequeno que não é provável que várias transações diferentes aprovem a mesma ponta. Além disso, se a latência da rede for muito baixa e os dispositivos calculam rapidamente, é improvável que muitas pontas apareçam. Isso é válido mesmo no caso de o fluxo das transações ser razoavelmente grande. Além disso, temos que assumir que não há atacantes que tentam inflar artificialmente o número de pontas.

- Carga alta: o número típico de pontas é grande. Isto pode acontecer quando o fluxo de transações é amplo, e atrasos computacionais em conjunto com a latência de rede torna provável que várias transações diferentes aprovem a mesma ponta.

Esta divisão é bastante informal, e não há limite claro entre os dois regimes. Mesmo assim, achamos que pode ser instrutivo considerar esses dois extremos diferentes.

A situação no regime de carga baixa é relativamente simples. A primeira aprovação acontece numa acontece numa escala de tempo média de ordem λ^{-1} uma vez que uma vez que uma das primeiras transações recebidas irá aprovar uma dada ponta.

Vamos considerar agora o regime de carga alta, o caso onde L_0 é amplo. Como mencionado acima, pode-se supor que os fluxos de Poisson de aprovação para diferentes pontas são independentes e tem uma taxa aproximada de $2\lambda/L_0$. Assim sendo, o tempo esperado para uma transação para receber sua primeira aprovação é em torno de $L_0/(2\lambda) \approx 1.45h$ (1).

Contudo, vale a pena notar que, para estratégias de aprovação mais elaboradas[13], pode não ser uma boa idéia esperar passivamente muito tempo até uma transação é aprovado pelos outros. Isto é devido ao fato que pontas “melhores” irão continuar aparecendo e irão ser preferidas para aprovação. Em vez disso, no caso de uma

[12] O sentimento do autor é que a estratégia de aprovação da ponta é *o* ingrediente mais importante para construir uma criptomoeda baseada em emaranhado porque os vetores de ataque estão escondidos neste elemento do sistema. Além disso, como geralmente não há maneiras de aplicar uma estratégia específica de aprovação de ponta, deve haver uma boa razão para que os nós escolham voluntariamente seguir uma estratégia em comum. Um possível motivo é saber que uma boa parte de outros nós estão seguindo a mesma estratégia de aprovação da ponta.

[13] Que favorecem pontas de “melhor” qualidade nas futuras implementações da iota.

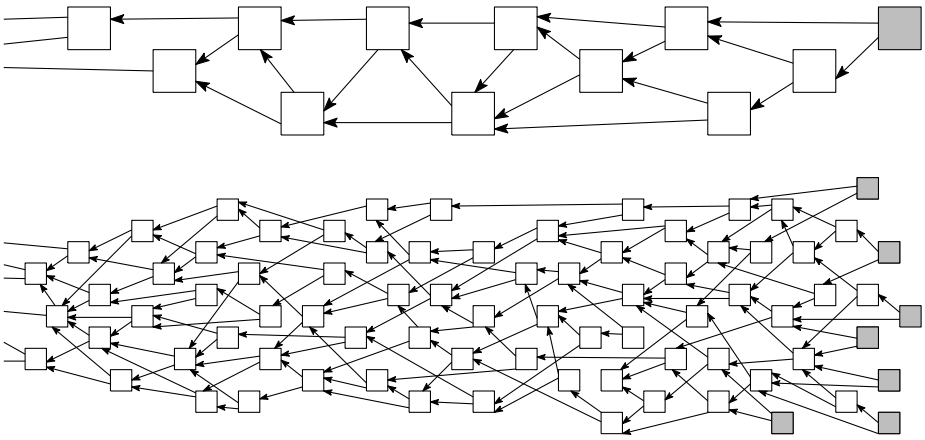

Figura 3: Regimes de carga baixa (superior) e carga alta (inferior) de fluxo de transações recebidas. Quadrados brancos representam locais verificados, enquanto quadrados representam pontas.

transação estar aguardando para aprovação um intervalo de tempo muito maior que $L_0/2\lambda$, uma boa estratégia seria promover esta transação latente com uma transação vazia adicional[14]. Em outras palavras, um nó pode emitir uma transação vazia que aprova sua transação anterior junto com uma das pontas "melhores" para aumentar a probabilidade que uma transação vazia receba aprovação.

Acontece que as estratégias de aprovação baseadas em alturas e pontuações podem ser vulneráveis a um tipo específico de ataques, ver a Seção 4.1. Nós iremos discutir estratégias mais elaboradas[15] para se defender contra tais ataques naquela seção. Enquanto isso, ainda vale a pena considerar a estratégia simples de seleção de ponta, onde uma transação recebida aprova duas pontas aleatórias. Essa estratégia é a mais fácil para analisar, e, portanto, pode prover algum insight sobre o comportamento qualitativo e quantitativo do emaranhado.

Conclusões:

[14]Uma transação vazia é uma transação que não envolve nenhuma transferência de token, mais ainda tem que aprovar duas outras transações. Deve ser observado que gerar uma transação vazia contribui para a segurança da rede.

[15]De fato, o sentimento do autor é que a estratégia de aprovação de ponta é *o* ingrediente mais importante para construir uma criptomoeda baseada em emaranhado. É lá que muitos vetores de ataque estão escondidos. Além disso, uma vez que geralmente não há como *reforçar* uma estratégia de aprovação de ponta em particular, ela deve ser tal que os nós voluntariamente optaram por segui-la. Sabendo que pelo menos uma boa proporção de outros nós faz isso.

1. Nós distinguimos entre dois regimes, carga baixa e carga alta (Figura 3).

2. Existem apenas algumas pontas no regime de carga baixa. Uma ponta é aprovada pela primeira vez em $\Theta(\lambda^{-1})$ unidades de tempo, onde λ é a taxa do fluxo de transações.

3. No regime de carga alta o número típico de pontas depende da estratégia de aprovação da ponta utilizada pela nova transação.

4. Se uma transação usa a estratégia de aprovar de duas pontas aleatórias, o número típico de pontas é dado por (1). Isso pode ser mostrado que essa estratégia é ótima em relação ao típico número de pontas. Entretanto, não é prático adotar essa estratégia porque não encoraja a aprovação de pontas.

5. Estratégias mais elaboradas são necessárias para lidar com ataques e outros problemas de rede. Uma família de tais estratégias é discutida na Seção 4.1.

6. O tempo usual para uma ponta ser aprovada é $\Theta(h)$ no regime de carga alta, onde h é a computação média/tempo de propagação para um nó. No entanto, se a primeira aprovação não ocorrer no intervalo de tempo acima, é uma boa ideia para o emitente e/ou receptor promover esta transação com uma transação vazia adicional.

3.1 Quão rápido o peso acumulado normalmente cresce?

Suponha que a rede esteja no regime de baixa carga. Depois que uma transação for aprovada várias vezes, seu peso acumulado crescerá com velocidade λ porque todas as novas transações irão indiretamente referenciar essa transação[16].

No caso onde a rede está no regime de carga alta, uma transação antiga com um peso acumulado grande irá experenciar um crescimento de peso com velocidade λ porque essencialmente todas as novas transações irão indiretamente referenciar ela. Além disso, quando a transação é adicionada pela primeira vez ao emaranhado, talvez seja necessário aguardar algum tempo para ser aprovada. Neste intervalo de tempo, o peso acumulado da transação comporta-se de forma aleatória. Para caracterizar a velocidade com que o peso acumulado cresce depois da transação receber várias aprovações, vamos definir $H(t)$ como o peso acumulado esperado no momento t (Para

[16]Lembre-se de que assumimos que os pesos próprios de todas as transações são iguais a 1, então o peso acumulado é somente o número de transações que direta ou indiretamente referenciam uma transação mais 1.

simplificar, nós começamos a contar o tempo no momento quando nossa transação for revelada para a rede, i.e., h unidades de tempo depois que foi criada) e $K(t)$ como o número esperado de pontas que aprovam a transação no momento t. Vamos também abreviar $h := h(L_0, N)$. Fazemos uma suposição simplificadora de que o número de pontas permanece aproximadamente constante em um valor de L_0 ao longo do tempo. Nós trabalhamos com a estratégia "aprovar duas pontas aleatórias" nesta seção. É esperado que o comportamento qualitativo irá ser grosso modo o mesmo para outras estratégias razoáveis.

Relembre que uma transação entrando na rede no momento t tipicamente escolhe duas pontas para aprovar baseado no estado do sistema no momento $t-h$ porque o nó deve fazer alguns cálculos e verificações antes de emitir a transação. Não é difícil ver que (assumindo, porém, que $K(\cdot)$ é o número *real* de pontas, não apenas o número esperado) a probabilidade da transação aprovar pelo menos uma de "nossas" pontas no emaranhado é $1 - \big(1 - \frac{K(t-h)}{L_0}\big)^2 = \frac{K(t-h)}{L_0}\big(2 - \frac{K(t-h)}{L_0}\big)$[17]. Análogo ao Exemplo 6.4 do [11], nós podemos escrever para pequeno $\delta > 0$

$$H(t+\delta) = H(t) + \lambda\delta\frac{K(t-h)}{L_0}\Big(2 - \frac{K(t-h)}{L_0}\Big) + o(\delta),$$

e assim deduzir a seguinte equação diferencial

$$\frac{dH(t)}{dt} = \lambda\frac{K(t-h)}{L_0}\Big(2 - \frac{K(t-h)}{L_0}\Big). \tag{3}$$

Para poder usar (3), precisamos primeiro calcular $K(t)$. Esta não é uma tarefa trivial uma vez que uma ponta no tempo $t-h$ pode não ser uma ponta no momento t, e o número total de pontas que aprovam a transação original aumenta em 1 no caso onde uma transação recebida aprova tal ponta. A observação crucial é que a probabilidade de que uma ponta no momento $t-h$ permanece uma ponta no tempo t é aproximadamente $1/2$. (Para verificar isso, relembre a discussão da Seção 3: o número típico de pontas é $2\lambda h$, e durante o intervalo de comprimento h novas λh pontas irão substituir a metade das antigas.) Portanto, no momento t aproximadamente a metade $K(t-h)$ das pontas permanece em no estado de ponta não-confirmada, enquanto a outra metade irá receber pelo menos uma aprovação. Deixe A denotar o conjunto de $K(t-h)/2$ pontas no tempo $t-h$ que ainda são pontas no tempo t, e deixe B denotar o conjunto restante de $K(t-h)/2$ pontas que já foram aprovadas no tempo t. Deixe p_1 ser a probabilidade que uma nova transação aprova pelo menos 1

[17] A expressão no lado esquerdo é 1 menos a probabilidade de que as duas pontas aprovadas não são nossas.

transação de B e não aprova nenhuma transação de A. Além disso, deixe p_2 ser a probabilidade que ambas transações aprovadas pertencem a A. Em outras palavras, p_1 e p_2 são as probabilidades que o número atual de pontas "nossas" aumenta e diminui em 1 acerca da chegada da nova transação. Nós temos

$$p_1 = \Big(\frac{K(t-h)}{2L_0}\Big)^2 + 2 \times \frac{K(t-h)}{2L_0}\Big(1 - \frac{K(t-h)}{L_0}\Big),$$
$$p_2 = \Big(\frac{K(t-h)}{2L_0}\Big)^2.$$

Para obter a primeira expressão, observe que p_1 é igual à probabilidade de ambas as pontas aprovadas pertencem a B mais duas vezes a probabilidade de que a primeira ponta pertença a B e a segunda ponta não pertença a $A \cup B$. Análogo a (3), a equação diferencial para $K(t)$ é:

$$\frac{dK(t)}{dt} = (p_1 - p_2)\lambda = \lambda \frac{K(t-h)}{L_0}\Big(1 - \frac{K(t-h)}{L_0}\Big). \tag{4}$$

É difícil de resolver (4) exatamente, então fazemos suposições simplificadoras adicionais. Em primeiro lugar, observamos que, após o tempo em que $K(t)$ atinge o nível εL_0 por um $\varepsilon > 0$, crescerá muito rapidamente para $(1-\varepsilon)L_0$. Agora, quando $K(t)$ é pequeno em relação a L_0, podemos abandonar o último fator no lado direito de (4)[18]. Nós obtemos uma versão simplificada de (4) ao recordar que $\frac{\lambda h}{L_0} = \frac{1}{2}$:

$$\frac{dK(t)}{dt} \approx \frac{1}{2h} K(t-h), \tag{5}$$

com condição de limite $K(0) = 1$. Procuramos uma solução da forma $K(t) = \exp(c\frac{t}{h})$; depois de substituir isso em (5), nós obtemos

$$\frac{c}{h} \exp\Big(c\frac{t}{h}\Big) \approx \frac{1}{2h} \exp\Big(c\frac{t}{h} - c\Big),$$

portanto

$$K(t) = \exp\Big(W\big(\tfrac{1}{2}\big)\frac{t}{h}\Big) \approx \exp\Big(0.352\frac{t}{h}\Big) \tag{6}$$

é uma solução aproximada, onde $W(\cdot)$ é a chamada Função W de Lambert.[19] Tomando o logaritmo de ambos os lados em (6), achamos que o tempo em que $K(t)$

[18] isto seria uma constante perto de 1, então o lado direito seria equivalente a $\lambda \frac{K(t-h)}{L_0}$.

[19] Também conhecido como a função ômega ou logaritmo do produto; para $x \in [0, +\infty)$ é caracterizada pela relação $x = W(x)\exp(W(x))$.

alcança εL_0 é aproximadamente

$$t_0 \approx \frac{h}{W\left(\frac{1}{2}\right)} \times \left(\ln L_0 - \ln \varepsilon^{-1}\right) \lesssim 2.84 \cdot h \ln L_0. \tag{7}$$

Retornando a (3) e retirando o último termo no lado direito, nós obtemos isso durante o "período de adaptação"(i.e., $t \leq t_0$ com t_0 como em (7)), isso sustenta que

$$\begin{aligned} \frac{dH(t)}{dt} &\approx \frac{2\lambda}{L_0} K(t-h) \\ &\approx \frac{1}{h \exp\left(W(\frac{1}{2})\right)} \exp\left(W\left(\tfrac{1}{2}\right)\frac{t}{h}\right) \\ &= \frac{2W\left(\frac{1}{2}\right)}{h} \exp\left(W\left(\tfrac{1}{2}\right)\frac{t}{h}\right) \end{aligned}$$

e portanto

$$H(t) \approx 2 \exp\left(W\left(\tfrac{1}{2}\right)\frac{t}{h}\right) \approx 2 \exp\left(0.352\frac{t}{h}\right). \tag{8}$$

Vamos relembrar ao leitor que depois do período de adaptação, o peso acumulado $H(t)$ cresce linearmente com velocidade λ. Ressaltamos que o "crescimento exponencial"em (8) não quer dizer que o peso acumulado cresce "muito rapidamente" durante o período de adaptação. Em vez disso, o comportamento é como descrito na Figura 4.

Conclusões:

1. Depois que uma transação for aprovada várias vezes no regime de carga baixa, seu peso acumulado crescerá com velocidade λw, onde w é o peso médio de uma transação genérica.

2. No regime de carga alta, há duas fases de crescimento distintas. Primeiro, o peso acumulado de uma transação $H(t)$ aumenta com velocidade crescente durante o *período de adaptação* de acordo com (8). Depois que o período de adaptação está acabado, o peso acumulado cresce com velocidade λw (Figura 4). De fato, para *qualquer* estratégia razoável o peso acumulado irá crescer com essa velocidade após o fim do período de adaptação porque todas as transações recebidas irão indiretamente aprovar a transação de interesse.

3. Pode-se pensar no período de adaptação de uma transação como o tempo até que a maioria das pontas atuais aprovem indiretamente essa transação. O comprimento típico do período de adaptação é dado por (7).

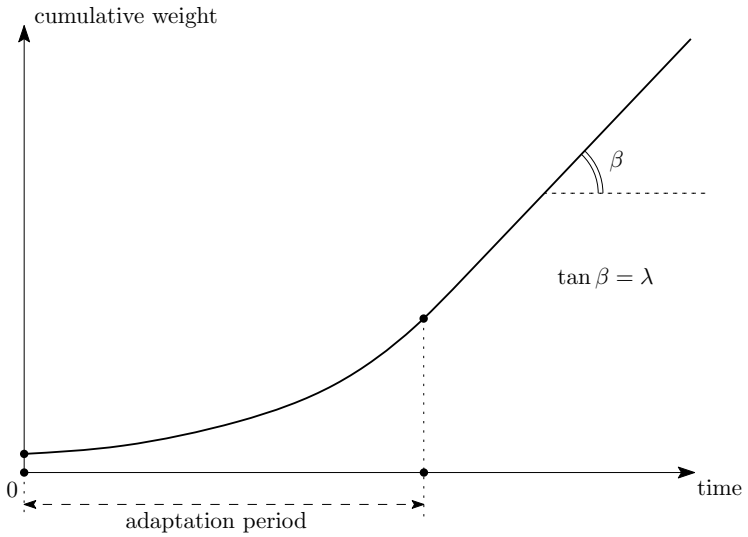

Figura 4: Parcela de peso acumulado em relação ao tempo para o regime de carga alta.

4 Possíveis cenários de ataque

Começamos por discutir um cenário de ataque onde o atacante tenta "ultrapassar" a rede sozinho:

1. Um atacante envia um pagamento a um comerciante e recebe os bens depois que o comerciante decide que a transação tem um peso acumulado suficientemente grande.

2. O atacante emite uma transação de gasto duplo.

3. O atacante usa seu poder computacional para emitir muitas pequenas transações que aprovam a transação de gasto duplo, mas não aprovam a transação original que foi enviada ao comerciante, seja direta ou indiretamente.

4. É possível que o atacante tenha uma infinidade de identidades Sybil que não são exigidas para aprovar pontas.

5. Um método alternativo ao item 3 seria para o atacante emitir uma grande transação de gasto duplo usando todo o seu poder computacional. Esta transação

teria um peso próprio muito grande[20], e aprovaria transações antes da transação legítima usada para pagar o comerciante.

6. O atacante espera que seu o subemaranhado desonesto supere o subemaranhado honesto. Se isso acontecer, o emaranhado principal continua crescendo a partir da transação de gasto duplo, e o ramo legítimo com o pagamento original ao comerciante se torna órfão (Figura 5).

Na verdade, pode-se mostrar que a estratégia de uma grande transação de gasto duplo aumenta a chance do atacante ser bem-sucedido. Na situação "ideal" desse modelo matemático, esse ataque *sempre* tem sucesso.

Deixe $W^{(n)}$ ser o tempo necessário para obter um nonce que dá a transação de gasto duplo um peso de pelo menos 3^n. Pode-se supor que $W^{(n)}$ é uma variável aleatória exponencialmente distribuída com parâmetro[21] $\mu 3^{-n}$, onde μ representa o poder computacional do atacante.

Suponha que o comerciante aceite a transação legítima quando seu peso acumulado torna-se pelo menos w_0, o que acontece t_0 unidades de tempo após a transação original. É razoável esperar que o peso acumulado cresça com velocidade linear λw, onde λ é a taxa global de chegada de transações emitidas na rede por nós honestos, e w é o peso médio de uma transação genérica. O peso total típico do ramo legítimo nesse momento é $w_1 = \lambda w t_0$.

Deixe $\lceil x \rceil$ seja o menor inteiro maior que ou igual a x, define $n_0 = \lceil \frac{\ln w_1}{\ln 3} \rceil$, de modo que $3^{n_0} \geq {w_1}^{22}$. Se o atacante conseguiu obter um nonce que dê à transação de gasto duplo um peso de pelo menos 3^{n_0} durante o intervalo de tempo de comprimento t_0, então o ataque tem êxito. a probabilidade deste evento é

$$\mathbb{P}[W^{(n_0)} < t_0] = 1 - \exp(-t_0 \mu 3^{-n_0}) \approx 1 - \exp(-t_0 \mu w_1^{-1}) \approx \frac{t_0 \mu}{w_1}.$$

Essa aproximação é verdadeira no caso em que $\frac{t_0 \mu}{w_1}$ é pequeno, o que é uma suposição razoável. Se esse ataque " imediato " não for bem-sucedido, o atacante pode continuar a procurar o nonce que dá peso 3^n para $n > n_0$, e esperar que no momento em que o encontrem, o peso total do ramo legítimo seja menor do que 3^n. A probabilidade de este evento ocorrer é

$$\mathbb{P}[\lambda w W^{(n)} < 3^n] = 1 - \exp\big(- \mu 3^{-n_0} \times (3^{n_0}/\lambda w)\big) = 1 - \exp(-\mu/\lambda w) \approx \frac{\mu}{\lambda w}.$$

[20] Aqui assumimos que o peso próprio de uma transação pode variar. Ficará claro na discussão abaixo porque é uma boa idéia deixar o próprio peso variar.

[21] Com expectativa $\mu^{-1} 3^n$.

[22] De fato, $3^{n_0} \approx w_1$ se w_1 é grande.

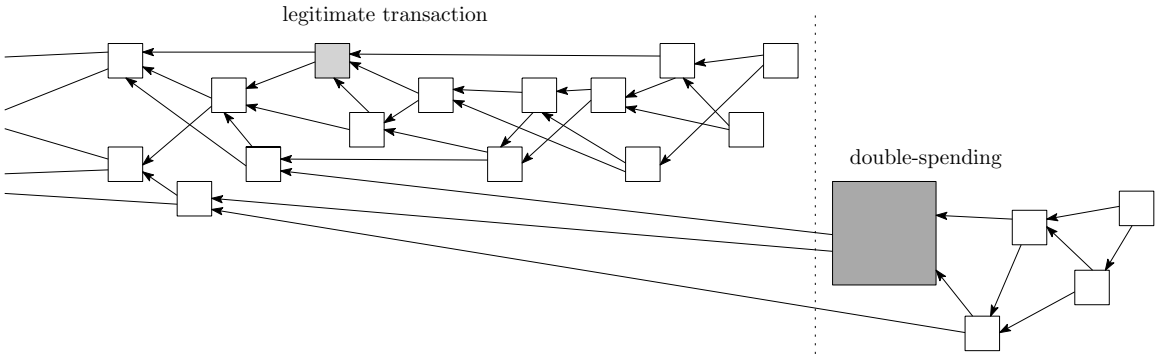

Figura 5: O ataque de “grande peso”

Isto é, embora $\frac{\mu}{\lambda w}$ normalmente deva ser um número pequeno, a cada “ nível ” n o ataque tem sucesso com uma probabilidade constante. Portento, ele terá sucesso. O tempo típico até que ele seja bem sucedido é aproximadamente $3^{\frac{\lambda w}{\mu}}$. Embora esta quantidade possa ser muito grande, a probabilidade de que o “primeiro”[23] ataque tenha sucesso não é insignificante. Portanto, precisamos de contramedidas. Uma dessas contramedidas seria limitar o peso próprio por cima, ou mesmo configurá-lo para um valor constante. Conforme mencionado na Seção 3, o último pode não ser a melhor solução porque não oferece proteção suficiente contra spam.

Agora, iremos discutir a situação em que o peso próprio máximo é limitado a um valor de 1, e estimar a probabilidade de que o ataque tenha êxito.

Suponha que uma dada transação tenha acumulado peso w_0 em t_0 unidades de tempo após o momento quando ela foi emitida, e que o período de adaptação para esta transação acabou. Nesta situação, o peso acumulado da transação aumenta linearmente com velocidade λ. Agora, imagine que o atacante quer fazer um gasto duplo nesta transação. Para fazer isso, o atacante secretamente prepara a transação de gasto duplo e começa a gerar transações *sem sentido* que aprovam a transação de gasto duplo no momento[24] em que a transação *original* foi emitida ao negociante. Se o subemaranhado do atacante ultrapassar o subemaranhado legítimo em algum momento depois que o negociante decide aceitar a transação legítima, o ataque de gasto duplo seria bem-sucedido. Se isso não acontecer, então a transação de gasto duplo não seria aprovada por outros porque a transação legítima iria adquirir mais peso acumulado e, essencialmente, todas as novas pontas iriam aprová-la indiretamente. A transação de gasto duplo seria órfã nesse cenário.

[23]Durante o tempo t_0.

[24]Ou mesmo antes; discutiremos este caso depois.

Como antes, deixe μ representar o poder de computação do atacante. Nós também fazemos uma suposição simplificadora de que as transações se propagam instantaneamente. Deixe $G_1, G_2, G_3, \ldots$ denotar i.i.d. variáveis aleatórias exponenciais com parâmetro μ[25], e definir $V_k = \mu G_k$, $k \geq 1$. Segue que $V_1, V_2, V_3, \ldots$ são i.i.d. variáveis aleatórias exponenciais com parâmetro 1.

Suponha que no momento t_0 o comerciante decida aceitar a transação com peso acumulado w_0. Vamos estimar a probabilidade de o atacante gastar duplamente com sucesso. Deixe que $M(\theta) = (1-\theta)^{-1}$ seja o momento que gera a função da distribuição exponencial com o parâmetro 1 (Seção 7.7 do [14]). Sabe-se[26] que para $\alpha \in (0,1)$ isso sustenta que

$$\mathbb{P}\Big[\sum_{k=1}^{n} V_k \leq \alpha n\Big] \approx \exp\big(-n\varphi(\alpha)\big), \tag{9}$$

onde $\varphi(\alpha) = -\ln\alpha + \alpha - 1$ é a transformada de Legendre de $\ln M(-\theta)$. Como fato geral, isso sustenta que $\varphi(\alpha) > 0$ para $\alpha \in (0,1)$. Relembre que a expectativa de uma variável aleatória exponencial com parâmetro 1 também é igual a 1.

Suponha que $\frac{\mu t_0}{w_0} < 1$, caso contrário, a probabilidade de o subemaranhado do atacante superar eventualmente o subemaranhado legítimo seria próximo de 1. Agora, para superar w_0 no momento t_0, o atacante precisa ser capaz de emitir pelo menos w_0 transações com peso próprio máximo m durante o momento t_0. Portanto, usando (9), encontramos a probabilidade de que a transação de gasto duplo tenha mais peso acumulado ao tempo t_0 é aproximadamente

$$\begin{aligned}
\mathbb{P}\Big[\sum_{k=1}^{w_0/m} G_k < t_0\Big] &= \mathbb{P}\Big[\sum_{k=1}^{w_0} V_k < \mu t_0\Big] \\
&= \mathbb{P}\Big[\sum_{k=1}^{w_0} V_k < w_0 \times \frac{\mu t_0}{w_0}\Big] \\
&\approx \exp\big(-w_0\varphi\big(\tfrac{\mu t_0}{w_0}\big)\big). \qquad (10)
\end{aligned}$$

Para a probabilidade acima ser pequena, $\frac{w_0}{m}$ precisa ser grande e $\varphi\big(\frac{\mu t_0}{w_0}\big)$ não pode ser muito pequeno.

Note que, no tempo $t \geq t_0$, o peso acumulado da transação legítima é aproximadamente $w_0 + \lambda(t - t_0)$ porque nós assumimos que o período de adaptação acabou,

[25] Com valor esperado $1/\mu$.

[26] Isto é uma consequência do chamado Princípio dos Grandes Desvios. Ver o livro geral[13], e a Proposição 5.2 na Seção 8.5 do [14] para uma simples e instrutiva derivação do limite superior, e a Seção 1.9 do [5] para a (não tão simples) derivação do limite inferior.

então o peso acumulado cresce com velocidade λ. Análogo a (10), descobre-se a probabilidade de a transação de gasto duplo ter mais peso acumulado ao tempo $t \geq t_0$ é aproximadamente

$$\exp\big(-(w_0 + \lambda(t - t_0))\varphi\big(\tfrac{\mu t}{w_0 + \lambda(t-t_0)}\big)\big). \tag{11}$$

Então, deve ser verdade que temos $\frac{\mu t_0}{w_0} \geq \frac{\mu}{\lambda}$ uma vez que o peso acumulado cresce com velocidade inferior a λ durante o período de adaptação. Pode-se mostrar que a probabilidade de alcançar um gasto duplo bem-sucedido é de ordem

$$\exp\big(-w_0\varphi\big(\max(\tfrac{\mu t_0}{w_0}, \tfrac{\mu}{\lambda})\big)\big). \tag{12}$$

Por exemplo, deixe$\mu = 2$, $\lambda = 3$ de modo que o poder do atacante seja apenas um pouco menor do que o resto da rede. Suponha que a transação tem um peso acumulado de 32 pelo tempo 12. Então, $\max(\frac{\mu t_0}{w_0}, \frac{\mu}{\lambda}) = \frac{3}{4}$, $\varphi\big(\frac{3}{4}\big) \approx 0.03768$, e (12) então dá o limite superior de aproximadamente 0.29. Se alguém assumir que $\mu = 1$ e mantém todos os outros parâmetros intactos, então $\max(\frac{\mu t_0}{w_0}, \frac{\mu}{\lambda}) = \frac{3}{8}$, $\varphi\big(\frac{3}{8}\big) \approx 0.3558$, e (12) dá aproximadamente 0.00001135, uma mudança bastante drástica.

Da discussão acima é importante reconhecer que a desigualdade $\lambda > \mu$ deve ser verdadeira para que o sistema seja seguro. Em outras palavras, o fluxo de entrada de "transações honestas" deve ser grande em comparação com o poder computacional do atacante. Caso contrário, a estimativa (12) seria inútil. Isso indica a necessidade de medidas de segurança adicionais, como pontos de controle, durante os primeiros dias de um sistema baseado em emaranhado.

Ao escolher uma estratégia para decidir qual das duas transações conflitantes é válido, tem de se ter cuidado ao utilizar peso acumulado como métrica de decisão. Isto é devido ao fato de que o peso acumulado pode ser sujeito a um ataque semelhante ao descrito na Seção 4.1, ou seja, o atacante pode preparar uma transação de gastos duplos com bastante antecedência, criar um subemaranhado secreto referenciando ela e, em seguida, transmitir esse subemaranhado depois que o comerciante aceita a transação legítima. Um método melhor para decidir entre duas transações conflitantes pode ser o único descrito na próxima seção: executar o algoritmo de seleção de ponta e ver qual das duas operações é indiretamente aprovado pela ponta selecionada.

4.1 Um ataque de cadeia parasita e um novo algoritmo de seleção de ponta

Considere o seguinte ataque (Figura 6): um atacante secretamente contrói um subemaranhado que ocasionalmente referencia o emaranhado principal para ganhar uma

pontuação maior. Note que a pontuação das pontas honestas é grosso modo a soma te todos os pesos no emaranhado principal, enquanto a pontuação das pontas do atacante contêm a soma de todos os pesos próprios na cadeia parasita. Como a latência da rede não é um problema para um atacante que constrói um subemaranhado sozinho[27], ele pode ser capaz de dar mais altura às pontas parasita se ele usar um computador que é suficientemente potente. Além disso, o atacante pode aumentar artificialmente sua contagem de pontas no momento do ataque, transmitindo muitas novas transações que aprovam transações que foram emitidas anteriormente na cadeia parasita (Figura 6). Isso dará ao atacante uma vantagem no caso em que os nós honestos usam alguma estratégia de seleção que envolve uma escolha simples entre pontas disponíveis. Para se defender desse estilo de ataque, vamos usar o fato de que o emaranhado principal supostamente deve ter mais poder de hashing ativo do que o atacante. Portanto, o emaranhado principal é capaz de produzir aumentos maiores no peso acumulado para mais transações do que o atacante. A ideia é usar um algoritmo MCMC para selecionar as duas pontas para referenciar.

Deixe $\mathcal{H}_x$ ser o peso acumulado atual de um local. Lembre-se de que assumimos que todos os pesos próprios são iguais a 1. Portanto, o peso acumulado de uma ponta é sempre 1, e o peso acumulado de outros locais é de pelo menos 2.

A idéia é colocar algumas partículas, também conhecidas como caminhantes aleatórios, em locais do emaranhado e deixá-los caminhar em direção às pontas de forma[28] aleatória. As pontas " escolhidas " pelos passeios são então os candidatos para aprovação. O algoritmo é descrito da seguinte maneira:

1. Considere todos os locais no intervalo $[W, 2W]$, onde W é razoavelmente grande[29].

2. Independentemente coloque N partículas em locais neste intervalo[30].

[27]Isto é devido ao fato de que um atacante pode sempre aprovar suas próprias transações sem depender de qualquer informação do resto da rede .

[28]Não há uma fonte de aleatoriedade "canônica". Os nós apenas usar seus próprios (pseudo) geradores de números aleatórios para simular os passeios aleatórios.

[29]A idéia é que a partícula seja colocada " profunda " no emaranhado de modo que não vai chegar a uma ponta de imediato. No entanto, a partícula não deve ser colocada "muito profunda"porque precisa encontrar uma ponta em um tempo razoável. Além disso, o intervalo $[W, 2W]$ é arbitrário. Poderíamos escolher $[W, 5W]$, etc. Existem também outras formas de selecionar os pontos de partida dos caminhantes. Por exemplo, um nó pode simplesmente fazer uma transação aleatória entre t_0 e $2t_0$ unidades de tempo no passado, onde t_0 é um ponto de tempo fixo.

[30]Essa escolha é em grande parte arbitrária. Usamos várias partículas ao invés de apenas duas para segurança adicional. A idéia é que, se uma partícula fosse acidentalmente pular para a cadeia do atacante, que é supostamente longa, então passaria muito tempo lá e outras pontas seriam escolhidas primeiro.

3. Deixe essas partículas realizarem passeios aleatórias em tempo discreto independentes "em direção às pontas", significando que uma transição de x para y é possível se e somente se y aprovar x

4. Os dois caminhantes aleatórios que atingem a ponta inicialmente se sentarão nas duas dicas que serão aprovadas. No entanto, pode ser sábio modificar esta regra da seguinte maneira: primeiro descarte aqueles caminhantes aleatórios que chegaram às dicas *muito rápido* porque eles podem ter terminado em uma das "pontas preguiçosas".

5. As probabilidades de transição dos caminhantes são definidas da seguinte maneira: se y aprova x ($y \rightsquigarrow x$), então a probabilidade de transição P_{xy} é proporcional a $\exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_y)\big)$, que é

$$P_{xy} = \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_y)\big)\Big(\sum_{z:z\rightsquigarrow x} \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_z)\big)\Big)^{-1}, \qquad (13)$$

   onde $\alpha > 0$ é um parâmetro para ser escolhido[31].

Observe que esse algoritmo é "local", o que significa que não é necessário atravessar o emaranhado de volta à gênese para realizar cálculos relevantes. Em particular, observe que não é necessário calcular o peso acumulado para todo o emaranhado. No máximo, é necessário calcular os pesos acumulados para os locais que aprovam indiretamente o ponto de partida do caminhante.

Para verificar se o algoritmo funciona como pretendido, primeiro considere as "pontas preguiçosas". Essas pontas aprovam intencionalmente algumas transações antigas para evitar o trabalho de verificação (Figura 6). Mesmo que a partícula esteja em um local aprovado por uma ponta preguiçosa, não é provável que a ponta preguiçosa seja selecionada porque a diferença entre pesos acumulados seria muito grande e P_{xy} seria pequena.

Em seguida, considere este estilo de ataque alternativo: o atacante constrói secretamente uma cadeia contendo uma transação que esvazia o saldo da conta dele para outra conta sob seu controle, indicada como o círculo vermelho mais à esquerda na Figura 6. Em seguida, o atacante emite uma transação no emaranhado principal, representada pelo círculo vermelho mais à direita, e espera que o comerciante a aceite. A cadeia parasita ocasionalmente referencia o emaranhado principal. No entanto, o peso acumulado não é muito grande na cadeia parasita. Deve-se notar que a cadeia parasita não pode fazer referenciar o emaranhado principal após a transação

[31] Pode-se começar com $\alpha = 1$.

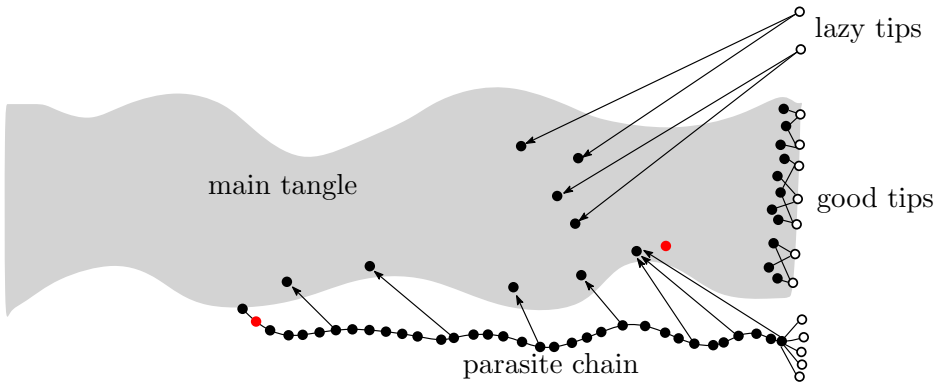

Figura 6: Representação visual do algoritmo de seleção de ponta para pontas honestas, bem como a cadeia parasita. Os dois círculos vermelhos indicam uma tentativa de gasto duplo por um atacante.

do comerciante. Além disso, o atacante poderia tentar inflar artificialmente o número de pontas em sua cadeia parasita no momento do ataque (Figura 6). A ideia do atacante é fazer com que os nós que emitem novas transações façam referência à cadeia parasita para que o ramo honesto do emaranhado se torne órfão.

É fácil ver porque o algoritmo de seleção MCMC não irá selecionar uma das pontas do atacante com alta probabilidade. O raciocínio é idêntico ao cenário da ponta preguiçosa: os locais na cadeia parasita terão um peso acumulado muito menor do que os locais que eles referenciam no emaranhado principal. Portanto, não é provável que o caminhante aleatório sempre salte para a cadeia parasita, a menos que comece lá,e esse evento não é muito provável porque o emaranhado principal contém mais locais.

Como uma medida de proteção adicional, primeiro podemos executar uma caminhada aleatória com um α grande (de modo que é na verdade " quase determinístico ") escolher uma "ponta modelo"; então, use caminhadas aleatórias com α pequeno para real seleção de ponta, mas verificar se (indiretamente) as transações referenciadas são consistentes com a ponta modelo.

Observe também que, para uma caminhada aleatória que *sempre* se move em direção as dicas, é muito simples e rápido calcular a distribuição de probabilidade de saída usando uma recursão direta; isso é algo que nós*não* queremos que os nós façam. No entanto, é possível modificar nossa abordagem da seguinte maneira: em cada etapa, a caminhada aleatória pode retroceder (ou seja, ir a 1 passo mais longe das pontas) com probabilidade (digamos) $\frac{1}{3}$ (e dividir o restante $\frac{2}{3}$ como antes). A

caminhada alcançará as pontas de forma muito rápida de qualquer forma (porque tem uma deriva para as pontas), mas não será tão fácil calcular a medida de saída.

Deixe-nos comentar por que os nós seguiriam esse algoritmo. Lembre-se da Seção 1 que é razoável supor que pelo menos uma proporção "boa"dos nós irá seguir o algoritmo de *referência*. Além disso, devido a atrasos computacionais e de rede, o algoritmo de seleção de ponta preferencialmente funcionaria com um instantâneo anterior do emaranhado em relação ao momento em que uma transação é emitida. *Pode ser uma boa idéia mover intencionalmente esse instantâneo para um ponto de tempo ainda mais no passado*[32] no algoritmo de referência pelas razões que explicamos na sequência. Imagine um nó "egoísta"que apenas deseja maximizar as chances de sua transação ser aprovada rapidamente. O algoritmo MCMC desta seção, que é adotado por uma proporção considerável de nós, define uma distribuição de probabilidade no conjunto de pontas. É claro que uma primeira escolha natural para um nó egoísta seria escolher as pontas onde o máximo dessa distribuição é alcançado. No entanto, se muitos outros nós também se comportam de forma egoísta e use a mesma estratégia, que é uma suposição razoável, então todos perderão. *Muitas* novas transações irão aprovar as mesmas duas pontas quase ao mesmo tempo, gerando, portanto, muita concorrência entre eles para posterior aprovação. Também deve ser claro que os nós não irão "sentir"o aumento de peso acumulado causado por esta aprovação em massa das mesmas duas dicas, uma vez que os nós estão usando um instantâneo passado. Por esse motivo, mesmo um nó egoísta teria que usar algum algoritmo de aprovação de ponta aleatória[33] com uma com uma distribuição de probabilidade para seleção de pontas que é próxima da a distribuição de probabilidade padrão produzida pelo algoritmo de seleção de ponta de referência.

Não afirmamos que essa distribuição de probabilidade "agregada"seria igual à distribuição de probabilidade padrão na presença de nós egoístas. No entanto, o argumento acima mostra que ele deve estar perto disso. Isso significa que a probabilidade de muitos nós tentarem verificar as mesmas dicas "ruins"continuaria pequena. Em todo caso, não há um grande incentivo para que os nós sejam egoístas porque os

[32]Primeiro, o walker aleatório encontra uma antiga ponta em relação a esse instantâneo e, em seguida, continua a caminhar em direção às pontas " reais " no emaranhado atual.

[33]como observado anteriormente, para uma caminhada de retrocesso, parece não haver uma maneira fácil de descobrir quais pontas são melhores (ou seja, mais chanllces de serem selecionadas por nós "honestos") além de executar o MCMC muitas vezes. No entanto, executar o MCRW muitas vezes requer tempo e outros recursos; Depois que alguém gasta algum tempo nisso, o estado do emaranhado já terá mudado, então talvez possamos ter que começar de novo. Isso explica por que os nós não têm motivos para abandonar a estratégia de seleção de pontas do MCMC em favor de outra coisa, pelo menos se eles assumirem que uma proporção considerável dos outros nós segue a estratégia de seleção de pontas padrão.

ganhos possíveis apenas representam uma ligeira diminuição no tempo de confirmação. Isso é inerentemente diferente de outras construções descentralizadas, como o Bitcoin. O fato importante é que os nós não têm motivos para abandonar o algoritmo de seleção de ponta MCMC.

Gostaríamos de mencionar que a definição de probabilidades de transição, conforme indicado em (13), não foi gravada em pedra. Em vez do expoente, pode-se usar uma função diferente que diminua rapidamente, como $f(s) = s^{-3}$. Há também liberdade para escolher W e N também. Neste momento, não está claro se existem argumentos teóricos que mostram exatamente de que forma esses parâmetros devem ser escolhidos. Em suma, achamos que a principal contribuição desta seção é a idéia de usar MCMC para a seleção de ponta.

4.2 Ataque de divisão

Aviv Zohar sugeriu o seguinte esquema de ataque contra o algoritmo MCMC proposto. No regime de alta carga, um atacante pode tentar dividir o emaranhado em dois ramos e manter o equilíbrio entre eles. Isso permitiria que ambas os ramos continuassem crescendo. O atacante deve colocar pelo menos duas transações conflitantes no início da divisão para impedir que um nó honesto de efetivamente unir os ramos ao referenciar ambos simultaneamente. Então, o atacante espera que aproximadamente metade da rede contribua para cada ramo para que eles possam "compensar"as flutuações aleatórias, mesmo com uma quantidade relativamente pequena de poder computacional pessoal. Se esta técnica funcionar, o atacante poderá gastar os mesmos fundos nos dois ramos.

Para se defender contra tal ataque, é preciso usar uma regra de "sharp thresold"Isso torna muito difícil manter o equilíbrio entre os dois ramos. Um exemplo dessa regra é selecionar a cadeia mais longa na rede Bitcoin. Deixe-nos traduzir este conceito para o emaranhado quando se está passando por um ataque de divisão. Suponha que o primeiro ramo tenha um peso total de 537 e o segundo ramo tenha peso total 528. Se um nó honesto selecionar o primeiro ramo com probabilidade muito próxima a 1/2, então o atacante provavelmente seria capaz de manter o equilíbrio entre os ramos. No entanto, se um nó honesto selecionar o primeiro ramo com probabilidade muito maior do que 1/2, então o atacante provavelmente seria incapaz de manter o equilíbrio. A incapacidade de manter o equilíbrio entre os dois ramos no último caso deve-se ao fato de que após uma inevitável flutuação aleatória, a rede rapidamente irá escolher um dos ramos e abandonará o outro. Para que o algoritmo MCMC se comporte dessa maneira, é preciso escolher uma função de decaimento muito rápida f, e iniciar a caminhada aleatória em um nó com grande profundidade

para que seja altamente provável que a caminhada comece antes da bifurcação do ramo. Neste caso, a caminhada aleatória escolheria o ramo "mais pesado"com alta probabilidade, mesmo que a diferença no peso acumulado entre os ramos concorrentes seja pequena.

Vale a pena notar que a tarefa do atacante é muito difícil devido a problemas de sincronização de rede: ele pode não estar ciente de um grande número de transações recentemente emitidas[34]. Outro método eficaz para defender contra um ataque de divisão seria para uma entidade suficientemente poderosa instantaneamente publicar uma grande quantidade de transações em um ramo, mudando assim rapidamente o equilíbrio de forças e tornando difícil para o atacante lidar com essa mudança. Se o atacante conseguir manter a divisão, as transações mais recentes terão apenas 50% confiança de confirmação (Seção 1), e os ramos não irão crescer. Nesse cenário, os nós "honestos"podem decidir começar a dar seletivamente sua aprovação para as transações que ocorreram antes da bifurcação, ignorando a oportunidade de aprovar as transações conflitantes nos ramos divididos.

Pode-se considerar outras versões do algoritmo de seleção de ponta. Por exemplo, se um nó vê dois grandes subemaranhados, ele escolhe aquele com uma soma maior de pesos próprios antes de executar o algoritmo de seleção de ponta MCMC descrito acima.

Pode considerar-se a seguinte ideia para futuras implementações. Poderíamos fazer as probabilidades de transição definidas em (13) dependam de ambos $\mathcal{H}_x - \mathcal{H}_y$ e $\mathcal{H}_x$ de tal forma que o próximo passo da cadeia de Markov é quase determinista quando o caminhante está profundo no emaranhado, mas torna-se mais aleatório quando o caminhante está perto de pontas. Isso ajudará a evitar entrar no ramo mais fraco, assegurando aleatoriedade suficiente quando escolher as duas pontas a serem aprovadas.

Conclusões:

1. Nós consideramos estratégias de ataque para quando um atacante tenta duplo gasto através de “ ultrapassar” o sistema.

2. O ataque de "grande peso"significa que, para gastar duas vezes, o atacante tenta dar um peso muito grande à transação de gasto duplo, de modo que superaria o subemaranhado legítimo. Essa estratégia seria uma ameaça para a rede no caso em que o peso próprio permitido não fosse ilimitado. Como solução, podemos

[34]Os pesos cumulativos "reais"podem ser bastante diferentes do que ele acredita.

limitar o peso próprio de uma transação por cima, ou configurá-lo para um valor constante.

3. Na situação em que o peso próprio máximo de uma transação é m, a melhor estratégia de ataque é gerar transações com peso próprio m que referenciam a transação de gasto duplo. Quando o fluxo de entrada das transações " honestas " é suficientemente grande em comparação com o poder computacional do atacante, a probabilidade de que a transação de gasto duplo tem um peso acumulado maior pode ser estimado usando a fórmula (12) (ver também exemplo abaixo (12)).

4. O método de ataque de construir uma "cadeia parasita"torna as estratégias de aprovação baseadas em altura ou pontuação obsoletas, uma vez que os locais do atacante terão valores maiores para essas métricas quando comparados com o emaranhado legítimo. Por outro lado, o algoritmo MCMC de seleção de ponta descrito na Seção 4.1 parece fornecer proteção contra esse tipo de ataque.

5. O algoritmo MCMC de seleção de ponta também oferece proteção contra nós preguiçosos como um bônus.

5 Resistência a cálculos quânticos

Sabe-se que um computador quântico suficientemente grande[35] pode ser muito eficiente para lidar com problemas que dependem de tentativa e erro para encontrar uma solução. O processo de encontrar um nonce para gerar um bloco Bitcoin é um bom exemplo desse problema. Atualmente, é preciso verificar uma média de 2^{68} nonces para encontrar um hash adequado que permita gerar um novo bloco. Sabe-se (ver e.g. [15]) que um computador quântico precisaria $\Theta(\sqrt{N})$ operações para resolver um problema que é análogo ao desafio Bitcoin mencionado acima. Este mesmo problema precisaria $\Theta(N)$ operações em um computador clássico. Portanto, um computador quântico seria cerca de $\sqrt{2^{68}} = 2^{34} \approx 17$ bilhões de vezes mais eficiente em minerar a blockchain do Bitcoin do que um computador clássico. Além disso, é importante notar que se uma blockchain não aumenta sua dificuldade em resposta para o poder de hashing aumentado, haveria uma taxa aumentada de blocos órfãos.

Pelo mesmo motivo, um ataque de "grande peso"também seria muito mais eficiente em um computador quântico. No entanto, limitando o peso por cima, como

[35] Ainda uma construção hipotética nos dias de hoje.

sugerido na Seção 4, efetivamente evitaria um ataque de computador quântico também. Isto é evidente na iota porque o número de nonces que é preciso verificar a fim de encontrar um hash adequado para a emissão de uma transação não é exageradamente grande. Em média, é em torno de 3^8. O ganho de eficiência para um computador quântico "ideal"seria, portanto, da ordem de $3^4 = 81$, que já é bastante aceitável[36]. Mais importante ainda, o algoritmo usado na implementação iota é estruturado de tal forma que o tempo para encontrar um nonce não é muito maior do que o tempo necessário para outras tarefas que são necessárias para emitir uma transação. Essa última parte é muito mais resistente contra a computação quântica e, portanto, dá ao emaranhado muito mais proteção contra um adversário com um computador quântico quando comparado à blockchain (do Bitcoin).

Agradecimentos

O autor agradece Bartosz Kusmierz, Cyril Grünspan, Olivia Saa, e Toru Kazama que apontaram vários erros em rascunhos anteriores, and James Brogan por suas contribuições para tornar esse artigo mais legível.

Referências

[1] Iota: a cryptocurrency for Internet-of-Things. See `http://www.iotatoken.com/`, and `https://bitcointalk.org/index.php?topic=1216479.0`

[2] bitcoinj. Working with micropayment channels. `https://bitcoinj.github.io/working-with-micropayments`

[3] PEOPLE ON NXTFORUM.ORG (2014) DAG, a generalized blockchain. `https://nxtforum.org/proof-of-stake-algorithm/dag-a-generalized-blockchain/` (registration at `nxtforum.org` required)

[4] MOSHE BABAIOFF, SHAHAR DOBZINSKI, SIGAL OREN, AVIV ZOHAR (2012) On Bitcoin and red balloons. *Proc. 13th ACM Conf. Electronic Commerce*, 56–73.

[5] RICHARD DURRETT (2004) Probability – Theory and Examples. *Duxbury advanced series.*

[36] Note que $\Theta(\sqrt{N})$ poderia facilmente significar $10\sqrt{N}$.

[6] Sergio Demian Lerner (2015) DagCoin: a cryptocurrency without blocks. https://bitslog.wordpress.com/2015/09/11/dagcoin/

[7] Yonatan Sompolinsky, Aviv Zohar (2013) Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. https://eprint.iacr.org/2013/881.pdf

[8] Yonatan Sompolinsky, Yoad Lewenberg, Aviv Zohar (2016) SPECTRE: Serialization of Proof-of-work Events: Confirming Transactions via Recursive Elections. https://eprint.iacr.org/2016/1159.pdf

[9] Yoad Lewenberg, Yonatan Sompolinsky, Aviv Zohar (2015) Inclusive Block Chain Protocols. http://www.cs.huji.ac.il/~avivz/pubs/15/inclusive_btc.pdf

[10] Joseph Poon, Thaddeus Dryja (2016) The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf

[11] Sheldon M. Ross (2012) *Introduction to Probability Models.* 10th ed.

[12] David Vorick (2015) Getting rid of blocks. slides.com/davidvorick/braids

[13] Amir Dembo, Ofer Zeitouni (2010) *Large Deviations Techniques and Applications.* Springer.

[14] Sheldon M. Ross (2009) *A First Course in Probability.* 8th ed.

[15] Gilles Brassard, Peter HØyer, Alain Tapp (1998) Quantum cryptanalysis of hash and claw-free functions. *Lecture Notes in Computer Science* **1380**, 163–169.