

RaiBlocks: Uma Criptomoeda com Rede Distribuída sem Taxas

Colin LeMahieu
clemahieu@gmail.com

Resumo—Recentemente, a alta demanda e a escalabilidade limitada aumentaram o custo e o tempo de transação médio em criptomoedas populares, produzindo uma experiência insatisfatória. Aqui introduzimos RaiBlocks, uma criptomoeda com uma arquitetura original denominada *block-lattice*, onde cada conta possui sua própria *blockchain*, possibilitando transações quase instantâneas e escalabilidade ilimitada. Cada usuário possui sua própria *blockchain*, possibilitando-lhes atualizá-la de forma assíncrona ao restante da rede, resultando em rápidas transações com mínima sobrecarga. Transações monitoram saldo de contas em vez de valores transacionados, permitindo uma poda agressiva da base de dados sem comprometer a segurança. Atualmente, a rede RaiBlocks já processou 4,2 milhões de transações com um registro sem poda de apenas 1,7GB. As transações sem custos e em frações de segundo fazem da RaiBlocks a criptomoeda ideal para transações de consumidores.

Index Terms—Criptomoeda, blockchain, raiblocks, registro distribuído, digital, transações.

I. INTRODUÇÃO

DESDE a implementação do Bitcoin em 2009, tem havido uma crescente fuga de moedas tradicionais suportadas por governos e sistemas financeiros em direção a sistemas modernos de pagamento baseados em criptografia, os quais oferecem a habilidade de armazenar e transferir fundos de uma maneira segura e que não requer confiança [1]. Para funcionar efetivamente, uma moeda deve ser facilmente transferível, não-reversível e possuir taxas limitadas ou nenhuma taxa. Os acrescidos tempos de transação, as altas taxas e a questionável escalabilidade da rede têm levantado questões sobre a praticidade do Bitcoin como uma moeda corrente para o dia a dia.

Neste artigo, introduzimos a RaiBlocks, uma criptomoeda de baixa latência, construída em uma arquitetura inovadora *block-lattice* para a estrutura de dados oferecendo escalabilidade ilimitada e sem custo de transações. Por projeto, RaiBlocks é um protocolo simples com o único propósito de ser uma criptomoeda de alta performance. O protocolo da RaiBlocks pode rodar em hardwares de baixa potência, permitindo a ela ser uma criptomoeda prática e descentralizada para o uso cotidiano.

Estatísticas das criptomoedas apresentadas nesse artigo são acuradas na data de publicação.

II. CONTEXTO

Em 2008, um indivíduo anônimo sob o pseudônimo Satoshi Nakamoto publicou um artigo técnico descrevendo a primeira criptomoeda descentralizada no mundo, o Bitcoin [1]. Uma

inovação chave trazida pelo Bitcoin foi a Blockchain, uma estrutura de dados pública, imutável e descentralizada que é usada como um registro para as transações da moeda. Infelizmente, com o amadurecimento do Bitcoin, vários problemas no protocolo tornaram-no inviável para várias aplicações:

- 1) Baixa escalabilidade: Cada bloco na blockchain pode armazenar uma quantidade limitada de dados, o que significa que o sistema pode processar apenas uma quantidade restrita de transações por segundo, transformando o lugar em um bloco em uma *commodity*. Atualmente, a taxa média de uma transação é US\$10,38 [2].
- 2) Alta latência: O tempo médio de confirmação de uma transação é de 164 minutos [3].
- 3) Ineficiência energética: A rede de Bitcoin tem um consumo estimado em 27,28TWh por ano, consumindo em média 260KWh por transação [4].

Bitcoin, e outras criptomoedas, funcionam por meio da obtenção de consenso em seus registros globais a fim de verificar transações legítimas enquanto resistem a atuantes maliciosos. Bitcoin alcança o consenso através de uma medida econômica denominada Prova de Trabalho (PoW, de “Proof of Work”). Em um sistema PoW, participantes competem para computar um número, chamado *nonce*, de forma que o hash do bloco inteiro esteja em um intervalo alvo. A amplitude desse intervalo válido é inversamente proporcional ao poder computacional acumulado em toda a rede do Bitcoin, a fim de manter consistente o tempo médio para se encontrar um *nonce* válido. Àquele que encontra o *nonce*, é permitido adicionar um bloco à blockchain; sendo assim, aqueles que dedicam mais recursos computacionais para computar um *nonce* possuem um papel maior no estado da blockchain. PoW proporciona resistência a um ataque Sybil, no qual uma entidade se comporta como múltiplas entidades a fim de ganhar poder adicional em um sistema descentralizado, e também reduz significativamente condições de corrida que são inerentes ao acesso a uma estrutura de dados global.

Um protocolo alternativo de consenso, Prova de Participação (PoS, de “Proof of Stake”) foi introduzido primeiramente pela Peercoin em 2012 [5]. Em um sistema PoS, participantes votam com peso equivalente à riqueza que eles possuem na dada criptomoeda. Nesse arranjo, àqueles que possuem um maior investimento financeiro é dado mais poder e eles são inerentemente incentivados a manter a honestidade do sistema ou arriscar perder seu investimento. PoS evita os desperdícios da competição de poder computacional, requerendo apenas *softwares* leves rodando em *hardwares* de baixa potência.

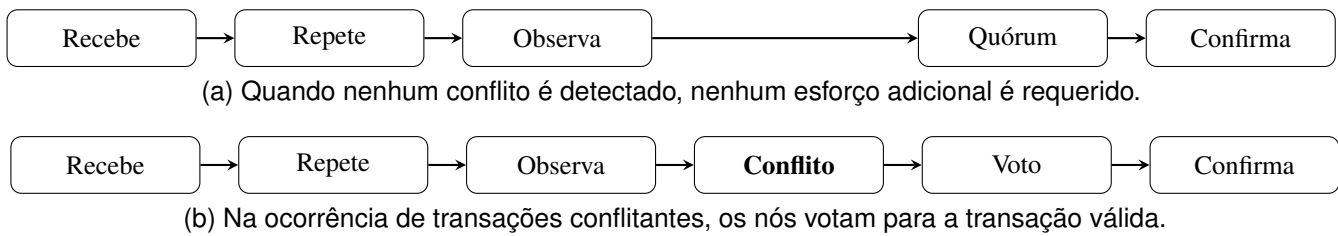


Figura 1. RaiBlocks não requer esforço adicional para transações típicas. Para transações conflitantes, os nós devem votar em qual transação manter.

O artigo original da RaiBlocks e a primeira implementação beta foram publicados em dezembro de 2014, fazendo dela uma das primeiras criptomoedas baseadas em Grafos Acíclicos Dirigidos (DAG) [6]. Pouco depois, outras criptomoedas baseadas em DAG começaram a se desenvolver, mais notavelmente DagCoin/Byteball [7] e IOTA [8]. Essas criptomoedas romperam o molde de blockchain, aprimorando a performance do sistema e sua segurança. Byteball alcança consenso dependendo de uma “cadeia-principal” composta de “testemunhas” honestas, reputadas e confiadas pelos usuários, enquanto IOTA alcança consenso através de PoW acumulado por transações empilhadas. RaiBlocks alcança consenso em transações conflitantes através de um sistema de votação ponderada pelo saldo. Esse sistema de consenso fornece transações mais rápidas e mais determinísticas enquanto ainda mantém um sistema forte e descentralizado. RaiBlocks continua este desenvolvimento e está posicionada como uma das criptomoedas de maior performance.

III. COMPONENTES DA RAIBLOCKS

Antes de descrever a arquitetura geral da RaiBlocks, vamos definir os componentes individuais do sistema

A. Conta

Uma conta é a porção de chave pública do par de chaves de uma assinatura digital. A chave pública, também referida como endereço, é compartilhada com os outros participantes da rede enquanto a chave privada é mantida em segredo. A assinatura digital de um pacote de dados assegura que o conteúdo foi aprovado pelo portador da chave privada. Um usuário pode controlar diversas contas, mas apenas um endereço público pode existir para cada conta.

B. Bloco/Transação

Os termos “bloco” e “transação” muitas vezes são usados indiferentemente, dado que um bloco contém uma única transação. Transação especificamente se refere à ação enquanto bloco refere-se à codificação digital da transação. Transações são assinadas pela chave privada pertencente à conta na qual a transação é feita.

C. Registro

Registro é o conjunto global de contas no qual cada conta tem sua própria cadeia de transações (Figura 2). Esse é um componente chave do projeto que se encaixa na categoria de substituir um acordo de tempo de execução por um acordo de

tempo de projeto; todos concordam via checagem de assinatura que somente o dono da conta pode modificar sua própria cadeia. Isso converte uma estrutura aparentemente compartilhada de dados, um registro distribuído, em um conjunto de estruturas não compartilhadas.

D. Nó

Um nó é uma parte de *software* rodando em um computador que se sujeita ao protocolo RaiBlocks e participa da rede RaiBlocks. O *software* gere o registro e todas as contas que o nó controlar, se alguma. Um nó pode tanto armazenar todo o registro quanto um histórico podado, contendo apenas os últimos blocos da blockchain de cada conta. Ao configurar um novo nó é recomendado verificar o histórico completo e podar localmente.

IV. VISÃO GERAL DO SISTEMA

Diferentemente das blockchains usadas em muitas outras criptomoedas, RaiBlocks usa uma estrutura *block-lattice*. Cada conta possui sua blockchain (cadeia da conta) equivalente ao histórico de transações/saldo da conta (Figura 2). Cada cadeia de conta pode ser atualizada apenas pelo proprietário da conta; isso permite que cada cadeia seja atualizada imediatamente e de forma assíncrona ao resto da *block-lattice*, resultando em transações rápidas. O protocolo da RaiBlocks é extremamente leve; cada transação cabe dentro do tamanho mínimo de pacote UDP para ser transmitido pela internet. Requerimentos de *hardware* para os nós também são mínimos, dado que nós

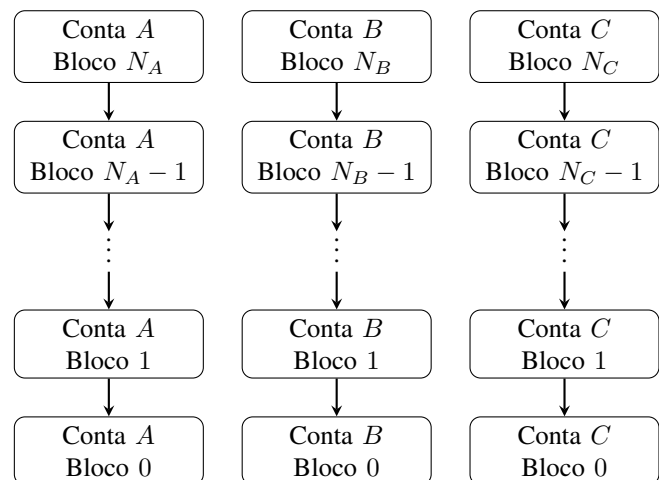


Figura 2. Cada conta tem sua própria blockchain contendo o histórico de saldo da conta. O bloco 0 deve ser uma transação de abertura (Seção IV-B)

apenas têm que gravar e transmitir blocos para a maioria das transações (Figura 1).

O sistema é iniciado com uma *conta genesis* contendo o *saldo genesis*. Este saldo é uma quantidade fixa e nunca pode ser aumentado. O saldo *genesis* é dividido e enviado para outras contas através de transações de envio registradas na cadeia da *conta genesis*. A soma dos saldos de todas as contas nunca excederá o saldo *genesis* inicial, o que dá ao sistema um limite superior na quantidade e não permite aumentá-lo.

Esta seção mostrará como diferentes tipos de transações são construídas e propagadas através da rede.

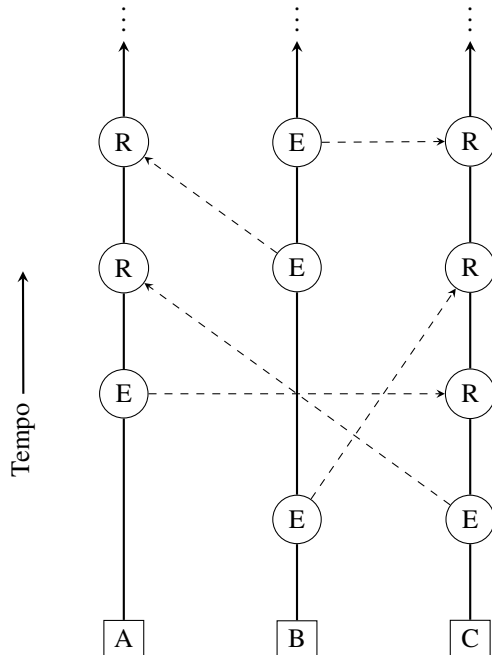


Figura 3. Visualização da block-lattice. Toda transferência de fundos requer uma transação de envio (E) e uma de recebimento (R), cada uma assinada pelo respectivo proprietário da cadeia da conta (A,B,C)

A. Transações

Transferências de fundos de uma conta para outra requerem duas transações: uma de envio (*send*) que deduz a quantia do saldo do remetente e uma de recebimento (*Receive*) que adiciona a quantia ao saldo da conta do receptor (Figura 3).

Transferir quantias como transações separadas nas contas dos remetentes e destinatários serve para alguns propósitos importantes:

- 1) Sequenciar chegadas de transferências que são inerentemente dessincronizadas;
- 2) Manter transações pequenas para caberem em pacotes UDP;
- 3) Facilitar a poda do registro, minimizando o rastro de dados;
- 4) Isolar transações estabelecidas das não estabelecidas.

Mais de uma conta transferindo para a mesma conta de destino é uma operação assíncrona; a latência da rede e as contas remetentes não necessariamente estando em comunicação umas com as outras significa que não há uma maneira

universalmente concordável para saber qual transação ocorreu primeiro. Dado que adição é uma operação associativa, a ordem em que as entradas são sequenciadas não importa e consequentemente precisamos apenas de uma concordância global. Esse é um componente chave do projeto que transforma uma concordância de tempo de execução em uma concordância em tempo de projeto. A conta destinatária tem controle para decidir qual transferência chegou primeiro e expressa isso através da ordem da assinatura dos blocos que chegam.

Se uma conta deseja enviar uma grande transferência a partir de um conjunto de várias pequenas transferências recebidas, nós queremos representar isso de forma que caiba em um pacote UDP. Quando uma conta receptora sequencia as transferências de entrada, ela mantém um saldo corrente tal que a qualquer momento ela é hábil a transferir qualquer quantia com um tamanho de transação fixo. Isso difere do modelo de transação entrada/saída utilizado pelo Bitcoin e outras criptomoedas.

Alguns nós não têm interesse em gastar recursos para armazenar o histórico completo de transações de uma conta; eles apenas estão interessados no saldo atual de cada conta. Quando uma conta faz uma transação, ela codifica o seu saldo acumulado e esses nós precisam apenas monitorar o último bloco, o que permite-lhes descartarem dados históricos mantendo a correção.

Mesmo com foco em concordâncias em tempo de projeto, há uma janela de atraso na validação das transações devido a identificar e lidar com maus atuantes na rede. Dado que concordâncias na RaiBlocks são alcançadas rapidamente, na ordem de milissegundos a segundos, podemos apresentar ao usuário duas categorias familiares de transações: estabelecidas e não estabelecidas. Transações estabelecidas são transações para as quais uma conta gerou blocos de recebimento. Transações não estabelecidas ainda não foram incorporadas ao saldo cumulativo do destinatário. Essa é uma substituição para as métricas de confirmação mais complexas e pouco familiares presentes em outras criptomoedas.

B. Criando uma Conta

Para criar uma conta, é preciso emitir uma transação de abertura (*open* - Figura 4). Uma transação de abertura é sempre a primeira transação da cadeia de uma conta e pode ser criada a partir do primeiro recebimento de fundos. O campo *account* armazena a chave pública (endereço) derivada da chave privada que é utilizada para assinatura. O campo *source* contém o hash da transação que enviou os fundos. Na criação da conta, um representante deve ser escolhido para votar em seu nome; isso pode ser mudado depois (Seção IV-F). A conta pode se declarar como representante de si mesma.

C. Saldo da Conta

O saldo da conta é gravado no registro em si. Em vez de gravar a quantia de uma transação, a verificação (Seção IV-I) exige checar a diferença entre o saldo do bloco de envio e o saldo do bloco predecessor. A conta destinatária pode então incrementar o saldo anterior com o saldo do novo bloco recebido. Isso é feito para aprimorar a velocidade de

```

open {
  account: DC04354B1...AE8FA2661B2,
  source: DC1E2B3F7C...182A0E26B4A,
  representative: xrb_lanr...posrs,
  work: 0000000000000000,
  type: open,
  signature: 83B0...006433265C7B204
}

```

Figura 4. Anatomia de uma transação de abertura (*open*).

processamento no download de grandes volumes de blocos. Quando solicitado o histórico da conta, os saldos já são dados.

D. Enviando de uma Conta

Para enviar de um endereço, este endereço deve já possuir um bloco de abertura e, portanto, um saldo (Figura 5). O campo *previous* contém o hash do bloco anterior na cadeia da conta. O campo *destination* contém a conta para a qual os fundos serão enviados. Um bloco de envio (*send*) é imutável uma vez confirmado. Assim que transmitido para a rede, os fundos são imediatamente deduzidos do saldo da conta do remetente e esperam como pendentes até a parte recebedora assinar o bloco que aceita esses fundos. Fundos pendentes não devem ser considerados como aguardando confirmação, já que eles já foram gastos da conta do remetente, que não pode revogar a transação.

```

send {
  previous: 1967EA355...F2F3E5BF801,
  balance: 010a8044a0...1d49289d88c,
  destination: xrb_3w...m37goeuufdp,
  work: 0000000000000000,
  type: send,
  signature: 83B0...006433265C7B204
}

```

Figura 5. Anatomia de uma transação de envio (*send*).

E. Recebendo uma Transação

Para completar uma transação, o destinatário dos fundos enviados deve criar um bloco de recebimento na própria cadeia da conta (Figura 6). O campo *source* referencia o hash da transação de envio associada. Uma vez que esse bloco é criado e transmitido, o saldo da conta é atualizado e os fundos foram movidos oficialmente para a conta.

F. Atribuindo um Representante

Detentores de contas terem a opção de escolher um representante para votar em seu nome é uma poderosa ferramenta de descentralização que não tem um análogo forte nos protocolos PoW e PoS. Nos sistemas PoS convencionas, o nó do dono da conta deve estar rodando para participar da votação. Rodar um nó continuamente é impraticável para vários usuários; dar

```

receive {
  previous: DC04354B1...AE8FA2661B2,
  source: DC1E2B3F7C6...182A0E26B4A,
  work: 0000000000000000,
  type: receive,
  signature: 83B0...006433265C7B204
}

```

Figura 6. Anatomia de uma transação de recebimento (*receive*).

a um representante o poder de votar em nome de uma conta relaxa esse requerimento. Proprietários de conta possuem a habilidade de reatribuir o consenso para qualquer conta a qualquer momento, uma transação de mudança (*change*) muda o representante de uma conta, subtraindo seu peso de voto do representante antigo e somando-o ao peso do novo representante (Figura 7). Nenhum fundo é movido nessa transação e o representante não tem o poder de gastar os fundos da conta representada.

```

change {
  previous: DC04354B1...AE8FA2661B2,
  representative: xrb_lanrz...posrs,
  work: 0000000000000000,
  type: change,
  signature: 83B0...006433265C7B204
}

```

Figura 7. Anatomia de uma transação de mudança (*change*).

G. Forks e Votação

Um fork (bifurcação) ocorre quando j blocos b_1, b_2, \dots, b_j reivindicam o mesmo bloco como seu predecessor (Figura 8). Esses blocos causam uma visão conflitante no status da conta e devem ser resolvidos. Apenas o dono da conta é hábil a assinar blocos na sua cadeia, então um fork necessariamente é o resultado de programação ruim ou intenções maliciosas (gasto duplo) por parte do dono da conta.

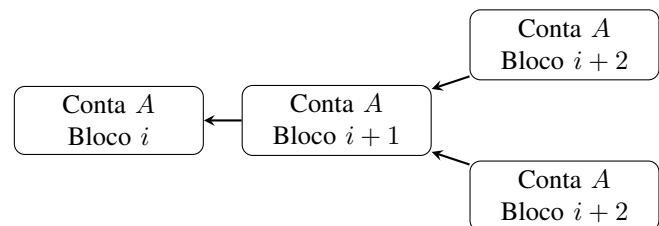


Figura 8. Um fork ocorre quando dois ou mais blocos assinados referenciam o mesmo bloco anterior. Blocos mais antigos estão à esquerda, mais novos à direita

Após detecção, um representante criará um voto referenciando o bloco b_i em seu registro e o transmitirá para a rede. O peso do voto de um nó, w_i , é a soma dos saldos de todas as contas que o nomearam como seu representante. O nó vai observar os votos provenientes dos outros M representantes

online e manter uma contagem acumulada por 4 períodos de votação, total de um minuto, e confirmar o bloco vencedor (Equação 1).

$$v(b_j) = \sum_{i=1}^M w_i \mathbb{1}_{b_i=b_j} \quad (1)$$

$$b^* = \arg \max_{b_j} v(b_j) \quad (2)$$

O bloco b^* que tiver a maioria dos votos será mantido no registro do nó (Equação 2). O(s) bloco(s) perdedor(es) são descartados. Se um representante substitui um bloco em seu registro, isso vai criar um novo voto com um número de sequência maior e transmitir o novo voto para a rede. Este é o **único** cenário onde representantes votam.

Em algumas circunstâncias, problemas breves de conectividade da rede podem causar um bloco transmitido a não ser aceito por todos os pares. Qualquer bloco subsequente nessa conta será ignorado como inválido pelos pares que não viram a transmissão inicial. Uma retransmissão desse bloco será aceita pelo pares restantes e os blocos subsequentes recuperados automaticamente. Mesmo quando um fork ou bloco perdido ocorrem, apenas as contas referenciadas na transação são afetadas; o restante da rede continua processando transações para todas as outras contas.

H. PoW

Todos os quatro tipos de transação tem um campo *work* que deve ser preenchido corretamente. Esse campo permite ao criador da transação computar um *nonce* tal que o hash da concatenação desse *nonce* com o campo *previous* nas transações de recebimento/envio/mudança ou com o campo *account* numa transação de abertura seja inferior a um certo limiar. Diferente do Bitcoin, o PoW na RaiBlocks é utilizado simplesmente como uma ferramenta anti-spam, similar ao Hashcash, e pode ser computado em ordem de segundos [9]. Quando uma transação é enviada, o PoW para o próximo bloco pode ser pré-computado já que o campo *previous* é conhecido; isso fará com que transações pareçam instantâneas para o usuário final desde que o tempo entre as transações seja maior que o tempo necessário para computar o PoW.

I. Verificação de Transação

Para um bloco ser considerado válido, ele deve possuir os seguintes atributos:

- 1) O bloco não deve já estar no registro (transação duplicada);
- 2) Deve ser assinado pelo proprietário da conta.
- 3) O bloco do campo *previous* deve ser a cabeça da cadeia da conta. Se este bloco existe e não é a cabeça, então há um fork.
- 4) A conta deve ter um bloco de abertura;
- 5) O hash computado atende aos requerimentos de limiar do PoW.

Se é um bloco de recebimento, checa-se se o hash do bloco *source* está pendente, ou seja, ainda não foi resgatado. Se é um bloco de envio, o saldo deve ser menor do que o saldo anterior.

V. VETORES DE ATAQUE

RaiBlocks, assim como todas as criptomoedas descentralizadas, pode ser atacada por partes maliciosas em tentativas de ganhos financeiros ou morte do sistema. Nesta seção, descrevemos alguns cenários de ataque, as consequências de tais ataques, e como o protocolo da RaiBlocks toma medidas preventivas.

A. Defasagem de Sincronização de Blocos

Na Seção IV-G, discutimos o cenário no qual um bloco pode não ser propriamente transmitido, levando a rede a ignorar blocos subsequentes. Se um nó observa um bloco que não tem o bloco anterior referido, ele tem duas opções:

- 1) Ignorar o bloco já que este pode ser um bloco com lixo malicioso;
- 2) Requisitar um ressincronização com outro nó.

No caso de ressincronização, uma conexão TCP deve ser formada com um nó de *bootstrapping* para facilitar o tráfego aumentado que uma ressincronização requer. Contudo, se o bloco era realmente um bloco mau, então a ressincronização era desnecessária, e aumentou o tráfego na rede sem necessidade. Esse é um Ataque de Amplificação da Rede e resulta em uma negação de serviço (denial-of-service — DoS).

Para evitar ressincronizações desnecessárias, nós esperarão até um certo limiar de votos ser observado para um bloco potencialmente malicioso antes de iniciar uma conexão com um nó de *bootstrap* para sincronizar. Se um bloco não recebe votos suficientes, pode ser assumido como lixo.

B. Inundação de Transações (Flood)

Uma entidade maliciosa poderia mandar diversas transações desnecessárias porém válidas entre contas sob seu controle numa tentativa de saturar a rede. Sem taxas de transações, eles são hábeis a continuar esse ataque indefinidamente. Contudo, o PoW requerido para cada transação limita a frequência de transações que uma entidade maliciosa pode gerar sem um investimento significativo em recursos computacionais. Mesmo sob tal ataque em uma tentativa de inflar o registro, nós que não possuem o histórico completo são hábeis a podar transações antigas de suas cadeias; isso reduz o uso de armazenamento deste tipo de ataque para quase todos os usuários.

C. Ataque Sybil

Uma entidade poderia criar centenas de nós RaiBlocks em uma única máquina; contudo, dado que o sistema de votos é ponderado pelo saldo da conta, adicionar nós extras à rede não irá fazer com que o atacante ganhe votos extras. Sendo assim, não há vantagem a ser ganha através de um Ataque Sybil.

D. Ataque Penny-Spend

Um Ataque Penny-Spend se dá quando um atacante envia quantias infinitesimais a um grande número de contas, a fim de gastar recursos de armazenamentos dos nós. A publicação de blocos tem sua frequência limitada pelo PoW, então isso

limita em certo nível a criação de contas e transações. Nós que não são de histórico completo podem podar contas abaixo de uma métrica estatística tal que uma conta mais provavelmente não é válida. Finalmente, RaiBlocks é ajustada para usar um espaço mínimo de armazenamento permanente, então o espaço requerido para armazenar uma conta adicional é proporcional ao tamanho de um bloco de abertura + indexação = $96B + 32B = 128B$. Isso equivale a 1GB ser capaz de armazenar 8 milhões de contas de penny-spend. Se nós desejarem podar mais agressivamente, eles podem calcular uma distribuição baseada na frequência de acesso e delegar contas usadas com pouca frequência a armazenamento mais lento.

E. Ataque de PoW Pré-computado

Dado que o proprietário de uma conta será a única entidade adicionando blocos à cadeia da conta, blocos sequenciais podem ser computados, conjuntamente com seu PoW, antes de serem transmitidos para a rede. Aqui o atacante gera uma miríade de blocos sequencias, cada de um valor mínimo, através de um período extenso de tempo. Em certo ponto, o atacante causa uma negação de serviço (DoS), inundando a rede com muitas transações válidas que os nós vão processar e ecoar tão rápido quanto possível. Essa é uma versão avançada da inundação de transações descrita na Seção V-B. Este tipo de ataque apenas funcionaria brevemente, mas poderia ser utilizado em conjunto com outros ataques como um Ataque >50% (Seção V-F) para aumentar sua efetividade. Limitação na frequência de transações e outras técnicas estão sendo investigadas atualmente para mitigar ataques.

F. >50% Ataque

A métrica de consenso para RaiBlocks é um sistema de votação ponderado pelo saldo. Se um atacante for capaz de ganhar mais de 50% da força de votação, ele pode oscilar o consenso na rede e deixar o sistema quebrado. Um atacante é capaz de reduzir a quantidade de saldo que deve perder ao impedir que bons nós venham a votar através de um DoS na rede. RaiBlocks toma as seguintes medidas para evitar esse tipo de ataque:

- 1) A defesa primária contra esse tipo de ataque é o peso dos votos ser amarrado ao investimento no sistema. O titular da conta é inerentemente incentivado a manter a honestidade do sistema para proteger seu investimento. Tentativas de adulterar o registro seriam destrutivas para o sistema como um todo, o que destruiria seu investimento.
- 2) O custo desse ataque é proporcional à capitalização de mercado da RaiBlocks. Nos sistemas PoW, pode ser inventada tecnologia que fornece controle desproporcional em comparação com o investimento monetário e, se o ataque for bem sucedido, essa tecnologia pode ser reaproveitada após o ataque estar completo. Com RaiBlocks, o custo de atacar o sistema escala com o próprio sistema e, se um ataque for bem-sucedido, o investimento no ataque não pode ser recuperado.
- 3) Para manter o quórum máximo de votantes, a próxima linha de defesa é a votação representativa. Os titulares

de contas que não conseguem participar de maneira confiável por motivos de conectividade podem nomear um representante que pode votar com o peso de seus saldos. Maximizar o número e a diversidade de representantes aumenta a resiliência da rede.

- 4) Forks em RaiBlocks nunca são acidentais, então os nós podem adotar políticas sobre como interagir com blocos bifurcados. O único momento em que contas não atacantes são vulneráveis a blocos bifurcados é se elas receberem um saldo de uma conta atacante. Contas procurando segurança em relação a blocos bifurcados podem esperar um pouco ou bastante tempo antes de receber de uma conta que gerou forks ou até optar por nunca receber. Os receptores também podem gerar contas separadas para usar quando recebem fundos de contas duvidosas para isolar outras contas.
- 5) Uma última linha de defesa que ainda não foi implementada é cementação de blocos *block cementing*. RaiBlocks faz grandes esforços para resolver bifurcações nos blocos rapidamente via votação. Os nós podem ser configurados para cementar blocos, o que os impedirá de serem revertidos após um certo período de tempo. A rede está suficientemente segura ao se concentrar no rápido tempo de ajuste para evitar forks ambíguos.

Uma versão mais sofisticada de um ataque > 50% é detalhada na Figura 9. “Offline” é a porcentagem de representantes que foram nomeados mas não estão online para votar. “Stake” é a quantidade de investimento com que o atacante está votando. “Ativo” são representantes que estão online e votando de acordo com o protocolo. Um atacante pode reduzir a quantidade de participação que ele deve empenhar ao derrubar outros votantes, deixando-os offline via um ataque de DoS na rede. Se esse ataque puder ser sustentado, os representantes que serão atacados tornar-se-ão não sincronizados e isso é demonstrado por “Não sinc”. Finalmente, um atacante pode ganhar um breve estouro na força de votação relativa, ao redirecionar o ataque de Negação de Serviço para um novo conjunto de representantes enquanto o antigo conjunto está resincronizando seus registros, isso é demonstrado por “Ataque”.

Offline	Não sinc	Ataque	Ativo	Stake
---------	----------	---------------	-------	-------

Figura 9. Um potencial arranjo de votação que poderia reduzir os requisitos de ataque em 51%.

Se um atacante for capaz de causar Stake > Ativo por uma combinação dessas circunstâncias, ele pode ser capaz de alterar votações no registro à custa de sua participação. Podemos estimar o quanto esse tipo de ataque poderia custar ao examinar a capitalização de mercado de outros sistemas. Se estimarmos 33% dos representantes como offline ou atacados via DoS, um atacante precisará comprar 33% do limite de mercado para atacar o sistema via votação.

G. Intoxicação Bootstrap

Quanto mais tempo um atacante é capaz de manter uma chave privada antiga com um saldo, maior a probabilidade de

que os saldos que existiam naquele momento não tenham representantes participantes porque seus saldos ou representantes foram transferidos para contas mais recentes. Isso significa que se um nó for iniciado para uma representação antiga da rede em que o atacante tenha um quórum de participação na votação em comparação com os representantes àquele momento, ele poderá oscilar as decisões de votação para esse nó. Se esse novo usuário quisesse interagir com qualquer pessoa, além do nó atacante, todas as suas transações seriam negadas, pois eles possuem diferentes blocos de cabeça. O resultado líquido é que os nós podem desperdiçar o tempo de novos nós na rede, alimentando-os com informações ruins. Para evitar isso, os nós podem ser emparelhados com um banco de dados inicial de contas e cabeças de bloco bem conhecidas; este é um substituto para o download do banco de dados inteiro até o bloco genesis. Quanto mais próximo o download é de ser atual, maior a probabilidade de se defender com precisão desse ataque. No final, este ataque provavelmente não é pior do que alimentar dados de lixo para os nós durante o bootstrapping, uma vez que eles não poderiam transacionar com qualquer pessoa que tenha um banco de dados contemporâneo.

VI. IMPLEMENTAÇÃO

Atualmente, a implementação de referência está implementada em C++ e tem produzido lançamentos desde 2014 no Github [10].

A. Características do Projeto

A implementação de RaiBlocks adere à arquitetura apresentada nesse artigo. Especificações adicionais são descritas aqui.

1) *Algoritmo de Assinatura*: RaiBlocks usa um algoritmo de curva elíptica ED25519 modificado com Blake2b para todas as assinaturas digitais [11]. ED25519 foi escolhido para assinaturas e verificações rápidas e alta segurança.

2) *Algoritmo de Hash*: Dado que o algoritmo de hash é utilizado apenas para prevenir spam na rede, a escolha do algoritmo é menos importante quando comparada com criptomoedas baseadas em mineração. Nossa implementação usa Blake2b com um algoritmo de dispersão para os conteúdos dos blocos [12].

3) *Função de Derivação da Chave*: Nas carteira de referência, as chaves são encriptadas por uma senha e a senha é alimentada via uma função de derivação de chave para proteger contra tentativas de quebras por ASIC. Atualmente Argon2 [13] é o vencedor da única competição pública voltada para criar uma função de derivação de chaves resiliente.

4) *Intervalo entre Blocos*: Como cada conta possui sua própria blockchain, atualizações podem ser feitas de forma assíncrona ao estado da rede. Sendo assim, não há intervalos de blocos e transações podem ser enviadas instantaneamente.

5) *Protocolo de Mensagens UDP*: Nosso sistema é projetado para operar indefinidamente usando o mínimo possível de recursos computacionais. Todas as mensagens no sistema foram projetadas para caber dentro de um único pacote UDP. Isto também torna fácil para pares leves com conectividade intermitente participarem da rede sem reestabelecer conexões

TCP de curto prazo. TCP é usada somente para novos pares quando eles desejam iniciar as blockchains de forma massiva.

Nós podem ter certeza de que suas transações foram recebidas pela rede observando tráfego de transmissões de transações de outros nós já que ele deve ver várias cópias ecoadas de volta a si mesmo.

B. IPv6 e Multicast

Para construir em cima de um UDP sem conexão, as implementações futuras usam o multicast IPv6 como uma substituição para inundações tradicionais de transações e transmissão de votos. Isso reduzirá o consumo de largura de banda da rede e dará mais flexibilidade inteligente aos nós no futuro.

C. Performance

No momento da escrita deste artigo, haviam sido processadas 4,2 milhões de transações pela rede RaiBlocks, produzindo uma blockchain de 1,7GB. Tempos de transações são medidos na ordem de segundos. A implementação de referência atual operando em SSDs comerciais pode processar acima de 10.000 transações por segundo sendo vinculada essencialmente ao limite de operações de entrada/saída.

VII. USO DE RECURSOS

Esta é uma visão geral dos recursos utilizados por um nó RaiBlocks. Adicionalmente, vamos passar por ideias de redução de usos de recursos para casos específicos de uso. Nós reduzidos são tipicamente chamados leves (*light*), podados (*pruned*), ou nós de verificação simplificada de pagamentos (SPV).

A. Rede

A quantidade de atividade de rede depende de quanto a rede contribui para a saúde dela.

1) *Representativo*: Um nó representativo requer recursos de rede máximos, pois observa o tráfego de votos de outros representantes e publica seus próprios votos.

2) *Inconfiável*: Um nó sem confiança é semelhante a um nó representativo, mas é apenas um observador, não contém uma chave privada de conta representativa e não publica seus próprios votos.

3) *Confiado*: Um nó de confiança observa o tráfego de votos de um representante em que confia para realizar consenso corretamente. Isso reduz a quantidade de tráfego de votação entrante de representantes que vão para este nó.

4) *Leve*: Um nó leve também é um nó confiado que apenas observa tráfego para contas em que ele está interessado, permitindo o uso mínimo da rede.

5) *Bootstrap*: Um nó bootstrap serve partes ou todo o registro para os nós que estão online. Isso é feito através de uma conexão TCP em vez de UDP, uma vez que envolve uma grande quantidade de dados que requer controle de fluxo avançado.

B. Capacidade de Disco

Dependendo da necessidade do usuário, diferentes configurações de nós requerem diferentes recursos para armazenamento.

- 1) *Histórico*: Um nó interessado em manter um registro histórico completo de todas as transações irá precisar da quantidade máxima de armazenamento.
- 2) *Atual*: Devido ao projeto de manter os saldos acumulados nos blocos, os nós apenas precisam manter o último bloco para cada conta a fim de participar do consenso. Se um nó não possui interesse em manter um histórico completo, pode optar por manter apenas os blocos cabeça das cadeias.
- 3) *Leve*: Um nó leve não mantém localmente dados do registro e participa da rede apenas para observar atividades nas contas em que esteja interessado ou, opcionalmente, criar novas transações com as chaves privadas que este possui.

C. CPU

- 1) *Geração de Transações*: Um nó interessado em criar uma nova transação deve produzir um nonce através de PoW, a fim de passar pelo mecanismo de aceleração da RaiBlocks. A computação de diferentes hardwares é comparada no Apêndice A I.
- 2) *Representante*: Um representante deve verificar assinatura dos blocos, votos, e também produzir as próprias assinaturas para participar no consenso. A quantidade de recursos do CPU para um nó representante é significativamente menor que uma geração de transação e deve funcionar com qualquer CPU em um computador contemporâneo.
- 3) *Observador*: Um nó observador não gera os próprios votos. Como o esforço de assinatura é mínimo, os requisitos de CPU são quase idênticos aos de rodar um nó representante.

VIII. CONCLUSÃO

Neste artigo apresentamos a estrutura de uma criptomoeda de baixa latência, sem taxas e veloz que utiliza uma nova estrutura de block-lattice e votos por Proof of Stake delegada. A rede requer recursos mínimos, nenhum hardware de mineração de alta potência e pode processar uma alta taxa de transferência. Tudo isso é alcançado através de blockchains individuais para cada conta, eliminando problemas de acesso e ineficiências de uma estrutura de dados global. Nós identificamos possíveis vetores de ataque no sistema e apresentamos argumentos sobre como RaiBlocks é resistente a essas formas de ataques.

APÊNDICE A

POW HARDWARE BENCHMARKS

Como mencionado anteriormente, a PoW em RaiBlocks é utilizada para reduzir spam na rede. Nossa implementação de nó fornece aceleração que pode se aproveitar de GPUs compatíveis com OpenCL. A Tabela I fornece comparações reais de vários hardwares. Atualmente, o limiar de PoW é fixado, mas um limiar adaptativo pode ser implementado à medida que o poder computacional médio progride.

Tabela I
PERFORMANCE DE POW POR HARDWARE

Dispositivo	Transações por segundo
Nvidia Tesla V100 (AWS)	6,4
Nvidia Tesla P100 (Google,Cloud)	4,9
Nvidia Tesla K80 (Google,Cloud)	1,64
AMD RX 470 OC	1,59
Nvidia GTX 1060 3GB	1,25
Intel Core i7 4790K AVX2	0,33
Intel Core i7 4790K,WebAssembly (Firefox)	0,14
Google Cloud 4 vCores	0,14-0,16
ARM64 server 4 cores (Scaleway)	0,05-0,07

RECONHECIMENTO

Gostaríamos de agradecer a Brian Pugh por compilar e formatar este documento e a Bruno Garcia, Douglas Braga, Luiz Fabrette e Mariana Carvalho por produzirem a tradução em português brasileiro.

REFERÊNCIAS

- [1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: <http://bitcoin.org/bitcoin.pdf>
- [2] "Bitcoin median transaction fee historical chart." [Online]. Available: https://bitinfocharts.com/comparison/bitcoin-median_transaction_fee.html
- [3] "Bitcoin average confirmation time." [Online]. Available: <https://blockchain.info/charts/avg-confirmation-time>
- [4] "Bitcoin energy consumption index." [Online]. Available: <https://digiconomist.net/bitcoin-energy-consumption>
- [5] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," 2012. [Online]. Available: <https://peercoin.net/assets/paper/peercoin-paper.pdf>
- [6] C. LeMahieu, "Raiblocks distributed ledger network," 2014.
- [7] Y. Ribero and D. Raissar, "Dagcoin whitepaper," 2015.
- [8] S. Popov, "The tangle," 2016.
- [9] A. Back, "Hashcash - a denial of service counter-measure," 2002. [Online]. Available: <http://www.hashcash.org/papers/hashcash.pdf>
- [10] C. LeMahieu, "Raiblocks," 2014. [Online]. Available: <https://github.com/clemahieu/raiblocks>
- [11] D. J. Bernstein, N. Duif, T. Lange, P. Shwabe, and B.-Y. Yang, "High-speed high-security signatures," 2011. [Online]. Available: <http://ed25519.cr.yp.to/ed25519-20110926.pdf>
- [12] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "Blake2: Simpler, smaller, fast as md5," 2012. [Online]. Available: <https://blake2.net/blake2.pdf>
- [13] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: The memory-hard function for password hashing and other applications," 2015. [Online]. Available: <https://password-hashing.net/argon2-specs.pdf>