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-os atualizá-la de forma dessincronizada do restante da rede, resultando em rápidas transações com mínima sobrecarga. Transações monitoram saldo de contas ao invés 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, há uma crescente fuga de moedas tradicionais suportadas por governos e sistemas financeiros em direção a sistemas modernos de pagamentos baseados em criptografia, os quais oferecem a habilidade de armazenar e transferir fundos de uma maneira segura e [1]. Para funcionar efetivamente, uma moeda deve ser facilmente transferível, não-reversível e ter 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 inovativa arquitetura de dados *block-lattice* oferecendo escalabilidade ilimitada e sem custo de transações. Por projeto, RaiBlocks é um simples protocolo com o único propósito de ser uma criptomoeda de alta performance. O protocolo da RaiBlocks pode rodar em hardwares de baixa potência, permitindo-a ser uma prática e decentralizada criptomoeda para o uso cotidiano.

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

II. Contexto

Em 2008, um indivíduo sob o pseudônimo Satoshi Nakamoto publicou um artigo técnico descrevendo a primeira criptomoeda descentralizada no mundo, o Bitcoin [1]. A inovação mais importante trazida pelo Bitcoin foi a Blockchain, uma estrutura de dados pública, imutável e descentralizada, usada como um registro para as transações da moeda. Infelizmente, com o amadurecimento do Bitcoin, vários problemas no protocolo o transformaram impraticável 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 de transações por segundo, transformando o lugar em um bloco em uma *commodity*. Atualmente, o custo mediano de 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 27,28 TWh por ano, usando em média 260KWh por transação [4]

Bitcoin, e outras criptomoedas, funcionam através de alcançar consenso em seus registros globais a fim de verificar transações legítimas enquanto resiste a atuantes maliciosos. Bitcoin alcança o consenso através de uma medida econômica denominada Prova de Trabalho (PoW). Em um sistema PoW, participantes competem para computar um número, chamado *nonce*, cuja hash do bloco inteiro esteja em seu intervalo alvo. A amplitude desse intervalo válido é inversamente proporcional ao poder computacional acumulado em toda rede do Bitcoin, a fim de manter consistente o tempo médio para se encontrar uma *nonce* válida. À quem encontra a *nonce*, é permitido adicionar o 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 à um ataque Sybil, aonde uma entidade se comporta como múltiplas entidades a fim de ganhar poder adicional em um sistema decentralizado, e também reduz significativamente condições de corrida que são inerentes ao acesso à uma estrutura de dados global.

Um protocolo de consenso alternativo, o Proof of Stake (PoS) 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. Com essa disposição, àqueles que possuem um maior investimento financeiro é dado mais poder e são inerentemente incentivados à manter a honestidade do sistema ou arriscar perder seu investimento. PoS evita a a competição por desperdício de poder computacional, requerendo apenas *softwares* de baixa peso rodando em *hardwares* de baixa potência.

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

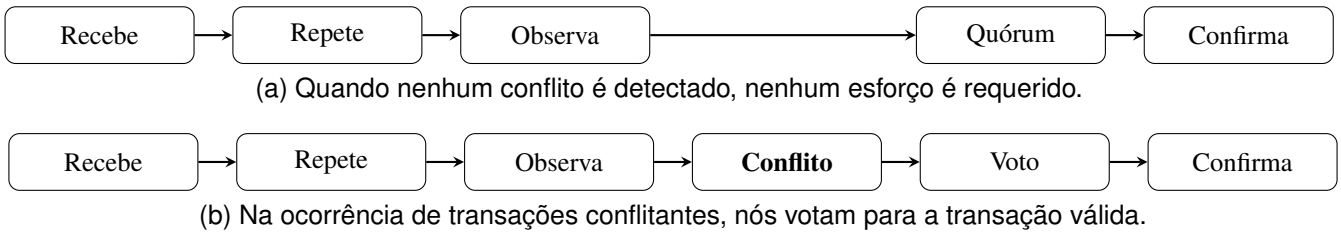

Figura 1. RaiBlocks não requer esforço adicional para transações típicas. Para transações conflitantes, nós devem votar em qual manter.

Direcionados (DAG) [6]. Pouco depois, outras criptomoedas baseadas em DAG começaram a se desenvolver, sendo mais nováveis DagCoin/Byteball [7] e IOTA [8]. Essas criptomoedas quebraram o modelo de blockchain, aprimorando a performance do sistema e sua segurança. Byteball alcança consenso dependendo de uma "cadeia-principal"compreendida de "testemunhas"honestas, reputáveis e confiáveis pelos usuários, enquanto IOTA alcança consenso através de PoW acumulado por transações. Raiblocks alcança consenso através de um sistema de votos ponderados em transações conflitantes. Esses sistema de consenso fornece transações mais rápidas e determinísticas enquanto ainda mantém um sistema forte e decentralizado. RaiBlocks continua seu desenvolvimento e está posicionada como uma das criptomoedas de maior performance.

III. Componentes da RaiBlocks

Antes de definir a arquitetura geral da RaiBlocks, nós definimos os componentes individuais que compõem o sistema

A. Conta

Uma conta é a porção de chave pública de uma 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. Um pacote de dados assinado digitalmente garante que o conteúdo foi aprovado pelo portador da chave privada. Um usuário pode controlar diversas contas, mas apenas um endereço público pode existir por 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 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 onde 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 através de checagem de assinatura que somente o dono da conta pode modificar sua própria cadeia. Isso converte de algo parecido à uma estrutura de dados compartilhada, um registro distribuído, em um conjunto de não registros compartilhados.

D. Nó

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

IV. Descrição do Sistema

Diferentemente das blockchains usadas usadas em muitas outras criptomoedas, RaiBlocks usa uma estrutura de grade de blocos (*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 cada cadeia ser atualizada imediatamente e dessincronizada do restante 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 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 divido e enviado para outras contas através de transações de envio registradas na cadeia da

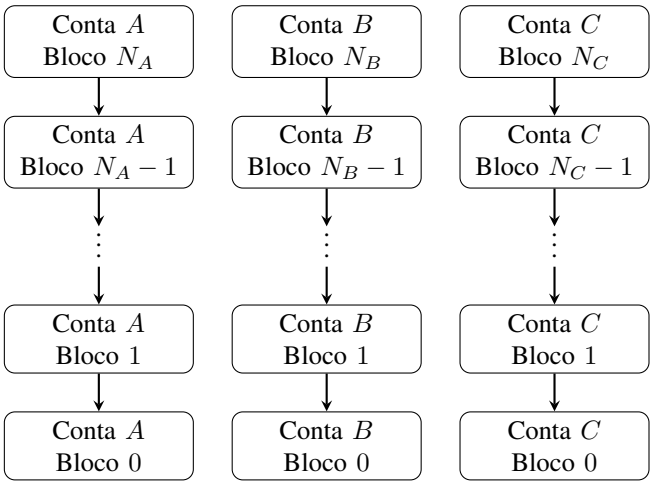

Figura 2. Cada conda tem sua própria blockchain contendo o histórico de saldo da conta. Bloco 0 deve ser uma transação de abertura (Seção IV-B)

conta *genesis*. A soma do saldo de todas as conta nunca vai exceder o saldo *genesis* inicial, o que dá ao sistema um limite superior na quantidade e não o permite aumentá-la.

Esta seção vai mostrar como diferentes tipos de transações são contruídas e propagadas através da rede.

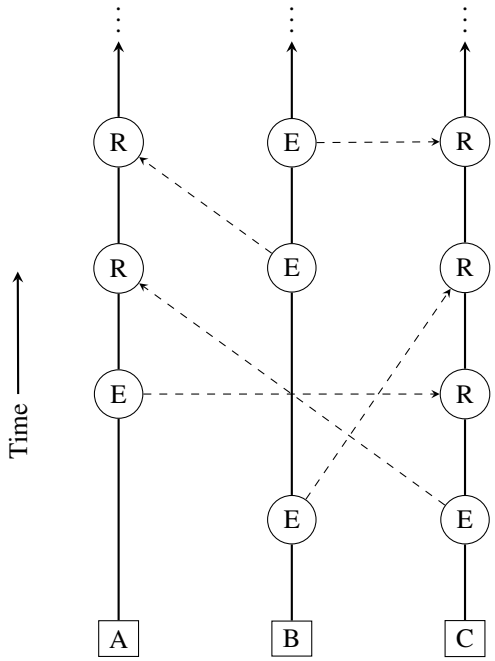

Figura 3. Visualização da block-lattice. Toda transferência de fundos requer uma transação de envio (E) e uma de recebimento (R), cada uma assinada pelo proprietário da própria cadeia da conta (A,B,C)

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 adicona a quantia ao saldo da conta do recebedor (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 dessincronizada; a latência da rede e as contas remetentes não necessariamente estarem em comunicação uma com as outras significa que não há uma maneira de concordância universal para saber qual transação ocorreu primeiro. Dado que adição é uma operação associativa, a ordem que as entradas são sequenciadas não importa e consequentemente apenas precisamos de uma concordância global. Esse é o componente chave do projeto que transforma uma concordância de tempo de corrida em uma concordância de 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 fazer uma grande transferência que é recebida como um conjunto de muitas pequenas transferências, nós queremos representar isso de forma que caiba em um pacote UDP. Quando uma conta recebedora 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 fixa. Isso difere no modelo de transação entrada/saída utilizado pelo Bitcoin e outras criptomoedas.

Alguns nós não tem interesse em gastar recursos para armazenar um 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 esse nós precisam apenas monitorar o último bloco, o que permite eles descartarem dados históricos mantendo informações corretas.

Mesmo com focus em concordâncias por tempo de projeto, há uma janela de atraso na validação das transações devido à identificar e lidar com maus atuantes na rede. Dado que concordâncias na RaiBlocks são alcançadas rapidamente, na ordem de milisegundos a segundos, nós podemos apresentar ao usuário duas categorias familiares de transações: estabelecidas e não estabelecidas. Transações estabelecidas são transações onde uma conta gerou blocos de recebimento. Transações não estabelecidas ainda não foram incorporadas ao saldo cumulativo. 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 recibo de fundos. O campo *account* armazena a chave pública (endereço) derivada da chave privada que é utilizada para assinatura. O campo *source* contém a hash da transação que enviou os fundos. Na criação da conta, um representante deve ser encolhido para votar em seu nome; isso pode ser mudado depois (Seção IV-F). A conta pode se declarar como o próprio representante.

```
open {
   account: DC04354B1...AE8FA2661B2,
   source: DC1E2B3F7C...182A0E26B4A,
   representative: xrb_1anr...posrs,
   work: 0000000000000000,
   type: open,
   signature: 83B0...006433265C7B204
}
```

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

C. Saldo da Conta

O saldo da conta é gravado dentro do próprio registro. Ao invés de gravar a quantia de uma transação, a verificação (Seção IV-I) exige checar a diferença entre o saldo do bloco de envio com 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 processamento no download de grandes volumes de blocos. Quando solicitado o histórico da conta, as quantias já são dadas.

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 quais 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 o parte recebedor 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 a hash da transação de envio associada. Uma vez que esse bloco é criado e transmitido, o saldo da conta é atualizado e os fundos movidos oficialmente para a conta.

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

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

F. Atribuindo um Representante

Detentores de contas terem a opção de escolher um representante para votar em seu nome é uma poderosa ferramenta de decentralização e não tem um forte análogo 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 a um representante o poder de votar em nome de uma conta relaxa esse requerimento. Proprietários de conta possuem a habilidade de reassentar consenso em qualquer conta em qualquer hora, uma transação de mudança (*change*) muda o representante de uma conta, subtraindo o peso de voto do representante antigo e somando ao peso do novo representante (Figura 7). Nenhum fundo é movido nessa transação e o representante não tem o poder de mover os fundos da conta.

```
change {
   previous: DC04354B1...AE8FA2661B2,
   representative: xrb_1anrz...posrs,
   work: 0000000000000000,
   type: change,
   signature: 83B0...006433265C7B204
}
```

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

G. Forks e Votos

Um fork ocorre quando j blocos $b_1, b_2, \ldots, b_j$ reivnidicam o mesmo bloco como seu predecessor (Figura 8). Esses blocos causam uma visualização de conflito 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 resultando de uma programação ruim ou intenções maliciosas (gasto duplo) pelo dono da conta.

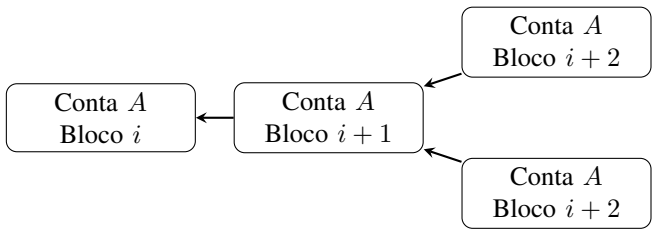

Figura 8. Um fork ocorre quando dois ou mais blocos assinados referenciam o mesmo bloco anterior. Blocos mais antigos estão a esquerda, mais novos a direita

Após a detecção, um representante vai criar um voto referenciando o bloco $\hat{b}_i$ em seu registro e transmitir para a rede. O peso de 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).

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

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

O bloco b^* que tiver a maioria dos votos vai ser 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 uma 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 de um bloco transmitido não ser aceito por todos os pares. Qualquer bloco subsequente nessa conta vai ser ignorado como inválido pelos pares que não virem a transmissão inicial. Uma retransmissão desse bloco vai ser aceita pelo pares restantes e os blocos subsequentes recuperados automaticamente. Mesmo quando um fork ou bloco perdido ocorre, apenas as contas referenciadas na transação são afetadas; o restante da rede procede com processamento de 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 a hash desse *nonce* concatenada com o campo *previous* nas transações de recebimento/envio/mudança ou o campo *account* numa transação de abertura é inferior a um certo limiar. Diferente do Bitcoin, o PoW na RaiBlocks é simplesmente utilizado 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 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, este deve ter os seguintes atributos:

1) O bloco deve não 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 a has do bloco *source* está pendente, ou seja, ainda não foi resgatada. Se é um bloco de envio, o saldo deve ser menor que o saldo anterior.

V. Vetores de ataque

RaiBlocks, assim como todas criptomoedas decentralizadas, pode ser atacada por partes maliciosas para tentativas de ganhos financeiros ou morte do sistema. Nessa seção, descrevemos alguns poucos 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 quando um bloco pode não ser propriamente transmitido, causando a rede a ignorar blocos subsequentes. Se um nó observa um bloco que não tem o bloco anterior referido, este 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 *bootstraping* para facilitar o tráfego aumentado que requer uma ressincronização. Contudo, se o bloco era realmente um bloco mau, então a ressincronização era desnecessária, assim como o aumento de tráfego na rede. Esse é um Ataque de Amplificação da Rede e resulta em uma negação de serviço (denial-of-service).

Para evitar ressincronizações desnecessárias, nós vão observar até um certo limiar de votos sejam observados para um bloco potencialmente malicioso antes de inciar 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 desnecessárias mas válidas transações entre as contas sob seu controle numa tentativa de saturar a rede. Sem custos de transações, eles são hábeis a continuar esse ataque indefinidamente. Contudo, o PoW requerido para cada transação limita a taxa de transação que uma entidade maliciosa pode gerar sem um investimento significante em recursos computacionais. Mesmo sob esse tipo 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 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 é quando um atacante envia quantias infinitesimais a um grande número de contas, a fim de gastar recursos de armazenamentos dos nós. Publicação de blocos tem a taxa limitada pelo PoW, então isso limita de alguma forma a criação de contas e transações. Nós que não

são de histórico completo podem podar contas abaixo de uma métrica estatística tal qual a conta é provavelmente uma conta não 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 à 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 poucas frequência à armazenamento mais lento.

E. Ataque de PoW Pré-computado

Dado que o proprietário de uma conta vai ser a única entidade adicionando blocos para a cadeia da conta, blocos sequencial podem ser computados, conjuntamente com seu PoW, antes de serem transmitidos para a rede. Aqui o atacante gera inumeráveis blocos sequencias, cada de um valor mínimo, através de um período extenso de tempo. Em certo ponto, o atacante performa uma negação de serviço (DoS), inundando a red com muitas transações válidas que os nós vão processar e ecoar tão rápido quanto possível. Essa é uma versão avançada da inundação de transação 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 dos limites 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, eles podem fazer com que a rede possa oscilar o consenso de tornar o sistema quebrado. Um invasor é capaz de reduzir a quantidade de equilíbrio que deve perder ao impedir que bons nós venham a votar através de uma rede DoS. RaiBlocks toma as seguintes medidas para evitar esse tipo de ataque:

1) A principal defesa contra este tipo de ataque é o peso do voto, vinculado ao investimento no sistema. O titular da conta é inerentemente incentivado a manter a honestidade do sistema para proteger seu investimento. Tentando virar o livro maior seria destrutivo para o sistema como um todo, o que destruiria seu investimento.
2) O custo desse ataque é proporcional à capitalização de mercado do RaiBlocks. Nos sistemas PoW, a tecnologia pode ser inventada, que dá um controle desproporcional em comparação com o investimento monetário e, se o ataque for bem sucedido, essa tecnologia pode ser repassada após o ataque estar completo. Com RaiBlocks, o custo de atacar o sistema se equilibra com o próprio sistema e, se um ataque for bem-sucedido, o investimento no ataque não pode ser recuperado.
3) Para manter o quorum máximo dos eleitores, 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 seu saldo. Maximizar o número e a diversidade de representantes aumenta a resiliência da rede.
4) Forks em RaiBlocks nunca são acidentais, então os nós podem tomar decisões políticas sobre como interagir com blocos bifurcados. A única vez que as contas não atacantes são vulneráveis a garfos de bloqueios é se eles receberem um saldo de uma conta atacante. As contas que desejam ser seguras dos garfos de blocos podem aguardar um pouco ou muito mais tempo antes de receber uma conta que gerou garfos ou 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 *block cementing*. RaiBlocks faz grandes esforços para sedimentar bloqueios rapidamente através da votação. Os nós podem ser configurados para blocos de cimento, 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 garfos 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 quem o atacante está votando. “Ativo"são representantes que estão online e votando de acordo com o protocolo. Um atacante pode compensar a quantidade de participação que eles devem perder ao bater outros eleitores fora de linha através de um ataque DoS de rede. Se esse ataque pode ser sustentado, os representantes que serão atacados tornar-se-ão não sincronizado e isso é demonstrado por “Não sinc."Finalmente, um atacante pode ganhar um breve estouro na força de votação relativa, alterando o ataque de Negação de Serviço para um novo conjunto de representantes enquanto o antigo conjunto está re-sincronizando sua razão, isso é demonstrado pelo “Ataque."

Offline	Não sinc	**Ataque**	Ativo	Parada

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

Se um invasor for capaz de causar Stake>Active por uma combinação dessas circunstâncias, eles poderão apontar os votos no ledger à custa de sua participação. Podemos estimar o quanto esse tipo de ataque poderia custar ao examinar o limite de mercado de outros sistemas. Se estimarmos 33% dos representantes estão offline ou atacados via DoS, um invasor precisará comprar 33% do limite de mercado para atacar o sistema através da 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 terão 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 quorum de participação na votação em comparação com os representantes nesse momento, eles poderão 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 denegadas, pois eles possuíam diferentes blocos de cabeça. O resultado líquido é que os nós podem desperdiçar o tempo de novos nós na rede, alimentando-lhes 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 todo o caminho de volta para o bloqueio da gênese. Quanto mais próximo o download é para ser atual, maior a probabilidade de defender com precisão contra esse ataque. No final, este ataque provavelmente não é pior do que alimentar dados de lixo para 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 está produzindo lançamentos desde 2014 no Github [10].

A. Características do Projeto

A implementação de RaiBlocks se adere à arquitetura apresentada nesse artigo. Especificações adicional 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 alimentas 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 dessincronizada 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 de recursos computacionais possíveis. Todas mensagens no sistema foram projetadas para caber dentro de um único pacote UDP. Isto também torna-o fácil para pares leves com conectividade intermitente participarem da rede sem reestabelecer conexões TCP curtas. TCP é usada somente para novos pares quando eles desejam iniciar as blockchain de forma massiva.

Nós podem ter certeza se suas transações foram recebidas pela rede observando tráfego de transmissões de transações de outros nós já que ele deve ver diversas cópias ecoando para 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 que se encaminham para a frente.

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 podem processar acima de 10.000 transações por segundos sendo principalmente limitadas por 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 de pagamentos simplificados (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) Confiável: Um nó de confiança observa o tráfego de voto de um representante que confia para realizar um 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 confiável 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 toda a razão para nós que estamos 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 precisas da quantidade máxima de armazenamento.
2) *Atual:* Devido ao projeto de manter os saldos acumulados nos blocos 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 o cabeçalho dos blocos.
3) *Leve:* Um nó leve não mantém registro local e apenas participa da rede para observar atividades em contas que esteja interessados ou, opcionalmente, criar novas transações para as chaves privadas que este possui.

C. CPU

1) *Geração de Transações:* Um nó interessado em criar uma nova transação deve produzir uma 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 custo, 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, eliminado 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 à 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. Tabela I fornece comparações reais de vários hardwares. Atualmente, o limiar de PoW é fixado, mas um limiar adaptativo pode ser implementado a 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 Brian Pugh por compilar e formatar este documento.

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