**Cripto como em criptomoeda?**

A criptografia pode ser a base para um novo sistema financeiro? É isso que as criptomoedas vêm tentando responder desde pelo menos 2008, quando o Bitcoin foi proposto por Satoshi Nakamoto (que até hoje não revelou sua identidade, ou identidades). Antes disso, o termo *crypto* era sempre usado em referência ao campo da criptografia. Mas, desde a criação do Bitcoin, tenho visto seu significado mudar rapidamente, sendo agora também usado para se referir às criptomoedas. Os entusiastas de criptomoedas, por sua vez, têm se interessado cada vez mais em aprender sobre criptografia. Isso faz sentido, já que a criptografia está no núcleo das criptomoedas.

O que é uma criptomoeda? São duas coisas:

* É uma moeda digital. Simplificando, permite que as pessoas transacionem moeda eletronicamente. Às vezes, usa-se uma moeda respaldada por um governo (como o dólar americano), e às vezes usa-se uma moeda inventada (como o bitcoin). Você provavelmente já usa moedas digitais — sempre que envia dinheiro para alguém na internet ou usa uma conta corrente, você está usando uma moeda digital! De fato, não é mais necessário enviar dinheiro em espécie pelo correio, e a maioria das transações financeiras hoje são apenas atualizações de linhas em bancos de dados.
* É uma moeda que depende fortemente da criptografia para evitar o uso de uma terceira parte confiável e para fornecer transparência. Numa criptomoeda, não há uma autoridade central em quem se deve confiar cegamente, como um governo ou um banco. Frequentemente, falamos dessa propriedade como descentralização (como em “estamos descentralizando a confiança”). Assim, como você verá neste capítulo, as criptomoedas são projetadas para tolerar. Assim, como você verá neste capítulo, as criptomoedas são projetadas para tolerar um certo número de atores maliciosos e permitir que as pessoas verifiquem que elas funcionam corretamente.

As criptomoedas são relativamente novas, já que o primeiro experimento bem-sucedido foi o Bitcoin, proposto em 2008, no meio de uma crise financeira global. Embora a crise tenha começado nos EUA, rapidamente se espalhou pelo resto do mundo, corroendo a confiança que as pessoas tinham nos sistemas financeiros e fornecendo uma plataforma para iniciativas mais transparentes como o Bitcoin. Naquela época, muitas pessoas começaram a perceber que o status quo para transações financeiras era ineficiente, caro de manter e opaco para a maioria das pessoas. O resto é história, e acredito que este livro seja o primeiro livro de criptografia a incluir um capítulo sobre criptomoedas.

**12.1 Uma introdução suave aos algoritmos de consenso tolerantes a falhas bizantinas (BFT)**

Imagine que você queira criar uma nova moeda digital. Não é tão complicado construir algo que funcione. Você poderia configurar um banco de dados em um servidor dedicado, que seria usado para rastrear usuários e seus saldos. Com isso, você fornece uma interface para as pessoas consultarem seu saldo ou permitirem que enviem pagamentos, o que reduziria seu saldo no banco de dados e aumentaria o saldo em outra linha. Inicialmente, você também poderia atribuir aleatoriamente alguma quantidade de sua moeda inventada para seus amigos, para que eles possam começar a transferir dinheiro para o seu sistema. Mas um sistema tão simples tem várias falhas.

**12.1.1 Um problema de resiliência: Protocolos distribuídos ao resgate**

O sistema que acabamos de ver é um único ponto de falha. Se você perder a eletricidade, seus usuários não poderão usar o sistema. Pior ainda, se algum desastre natural destruir inesperadamente seu servidor, todos podem perder permanentemente seus saldos. Para lidar com essa questão, existem técnicas bem conhecidas que você pode usar para fornecer mais resiliência ao seu sistema. O campo de sistemas distribuídos estuda tais técnicas.

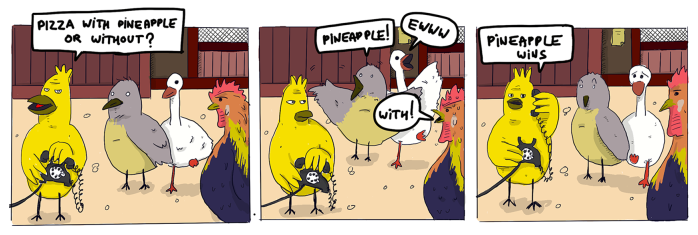
Nesse caso, a solução usual usada pela maioria das grandes aplicações é replicar o conteúdo do seu banco de dados em (mais ou menos) tempo real para outros servidores de backup. Esses servidores podem então ser distribuídos por várias localizações geográficas, prontos para serem usados como backup ou até mesmo para assumir o controle se o seu servidor principal falhar. Isso é chamado de alta disponibilidade. Agora você tem um banco de dados distribuído.

Para sistemas grandes que atendem muitas consultas, é comum que esses bancos de dados de backup não estejam apenas de prontidão esperando para serem úteis, mas sim, sejam usados para fornecer leituras do estado. É difícil ter mais de um banco de dados aceitando escritas e atualizações porque, nesse caso, você poderia ter conflitos (da mesma forma que duas pessoas editando o mesmo documento pode ser perigoso). Assim, geralmente você quer que um único banco de dados atue como líder e ordene todas as escritas e atualizações no banco de dados, enquanto os outros podem ser usados para ler o estado.

A replicação do conteúdo do banco de dados pode ser lenta, e espera-se que alguns dos seus bancos de dados fiquem para trás do líder até que se atualizem. Isso é especialmente verdade se estiverem situados em locais mais distantes no mundo ou estiverem sofrendo atrasos na rede por algum motivo. Esse atraso se torna um problema quando os bancos de dados replicados são usados para ler o estado. (Imagine que você veja um saldo de conta diferente do seu amigo porque ambos estão consultando servidores diferentes.)

Nesses casos, as aplicações são frequentemente escritas para tolerar esse atraso. Isso é chamado de consistência eventual porque, eventualmente, os estados dos bancos de dados tornam-se consistentes. (Existem modelos de consistência mais fortes, mas geralmente são lentos e impraticáveis.) Tais sistemas também têm outros problemas: se o banco de dados principal falhar, qual deles deve assumir como banco de dados principal? Outro problema é: se os bancos de dados de backup estavam atrasados quando o banco de dados principal falhou, perderemos algumas das últimas alterações?

É aqui que algoritmos mais fortes — algoritmos de consenso (também chamados de replicação de log, replicação de máquina de estados ou transmissões atômicas) — entram em cena quando você precisa que todo o sistema concorde (ou chegue a um consenso) sobre alguma decisão. Imagine que um algoritmo de consenso resolve o problema de um grupo de pessoas tentando decidir qual pizza pedir. É fácil ver o que a maioria quer se todos estiverem na mesma sala. Mas se todos estão se comunicando através da rede onde as mensagens podem ser atrasadas, perdidas, interceptadas e modificadas, então é necessário um protocolo mais complicado.



Vamos ver como o consenso pode ser usado para responder às duas perguntas anteriores. A primeira pergunta, de qual banco de dados deve assumir no caso de uma falha, é chamada de eleição de líder, e um algoritmo de consenso é frequentemente usado para determinar qual será o próximo líder. A segunda pergunta é frequentemente resolvida visualizando as mudanças no banco de dados em duas etapas diferentes: pendente e confirmada. As mudanças no estado do banco de dados são sempre pendentes no início e só podem ser definidas como confirmadas se suficientes bancos de dados concordarem em confirmá-las (é aqui que um protocolo de consenso também pode ser usado). Uma vez confirmada, a atualização no estado não pode ser facilmente perdida, já que a maioria dos participantes do banco de dados confirmou a alteração.

Alguns algoritmos de consenso bem conhecidos incluem Paxos (publicado por Lamport em 1989) e sua simplificação subsequente, Raft (publicado por Ongaro e Ousterhout em 2013). Você pode usar esses algoritmos na maioria dos sistemas de banco de dados distribuídos para resolver diferentes problemas. (Para uma ótima explicação interativa sobre o Raft, confira <https://thesecretlivesofdata.com/raft>.)

**12.1.2 Um problema de confiança? A descentralização ajuda**

Os sistemas distribuídos (do ponto de vista operacional) oferecem uma alternativa resiliente a sistemas que atuam como um único ponto de falha. Os algoritmos de consenso usados na maioria dos sistemas de banco de dados distribuídos não toleram falhas muito bem. Assim que as máquinas começam a falhar, ou a se comportar mal devido a falhas de hardware, ou começam a se desconectar de algumas das outras máquinas, como em partições de rede, surgem problemas. Além disso, não há como detectar isso da perspectiva do usuário, o que é ainda mais problemático se os servidores forem comprometidos.

Se eu consultar um servidor e ele me disser que Alice tem 5 bilhões de dólares em sua conta, eu simplesmente tenho que confiar nele. Se o servidor incluir em sua resposta todas as transferências de dinheiro que ela recebeu e enviou desde o início dos tempos e somar tudo, eu poderia verificar que, de fato, o saldo de 5 bilhões de dólares em sua conta está correto. Mas o que me garante que o servidor não mentiu para mim? Talvez, quando Bob consulta um servidor diferente, ele receba um saldo e/ou histórico completamente diferente para a conta de Alice. Chamamos isso de *fork* (dois estados contraditórios apresentados como válidos), um ramo na história que nunca deveria ter acontecido. E, assim, você pode imaginar que o comprometimento de um dos bancos de dados replicados pode levar a consequências bastante devastadoras.

No capítulo 9, mencionei a transparência de certificados, um protocolo de *gossip* que visa detectar tais *forks* na infraestrutura de chaves públicas (PKI) da web. O problema com dinheiro é que apenas detectar não é suficiente. Você quer evitar que os *forks* aconteçam em primeiro lugar! Em 1982, Lamport, o autor do algoritmo de consenso Paxos, introduziu a ideia dos algoritmos de consenso tolerantes a falhas bizantinas (BFT).

Imaginamos que várias divisões do exército Bizantino estão acampadas fora de uma cidade inimiga, cada divisão comandada por seu próprio general. Os generais podem se comunicar entre si apenas por mensageiro. Após observar o inimigo, eles devem decidir um plano comum de ação. No entanto, alguns dos generais podem ser traidores, tentando impedir que os generais leais cheguem a um acordo.

— Lamport et al. ("The Byzantine Generals Problem", 1982)

Com sua analogia Bizantina, Lamport iniciou o campo dos algoritmos de consenso BFT, com o objetivo de impedir que participantes maliciosos de consenso criem diferentes visões conflitantes de um sistema ao concordarem sobre uma decisão. Esses algoritmos de consenso BFT se assemelham bastante aos algoritmos de consenso anteriores como Paxos e Raft, exceto pelo fato de que os bancos de dados replicados (os participantes do protocolo) não confiam cegamente uns nos outros. Protocolos BFT geralmente fazem uso intenso da criptografia para autenticar mensagens e decisões, o que, por sua vez, pode ser usado por outros para validar criptograficamente as decisões produzidas pelo protocolo de consenso.

Esses protocolos de consenso BFT são, portanto, soluções tanto para os nossos problemas de resiliência quanto de confiança. Os diferentes bancos de dados replicados podem executar esses algoritmos BFT para concordar com novos estados do sistema (por exemplo, saldos de usuários), enquanto se policiam mutuamente verificando que as transições de estado (transações entre usuários) são válidas e foram acordadas pela maioria dos participantes. Dizemos que a confiança agora está descentralizada.

O primeiro algoritmo de consenso BFT do mundo real foi o *Practical BFT* (PBFT), publicado em 1999. O PBFT é um algoritmo baseado em líder, semelhante ao Paxos e Raft, onde um líder é responsável por fazer propostas enquanto os demais tentam concordar com as propostas. Infelizmente, o PBFT é bastante complexo, lento e não escala bem além de uma dúzia de participantes. Hoje, a maioria das criptomoedas modernas usa variantes mais eficientes do PBFT. Por exemplo, o Diem, a criptomoeda introduzida pelo Facebook em 2019, é baseado no HotStuff, um protocolo inspirado no PBFT.

**12.1.3 Um problema de escala: Redes permissionadas e resistentes à censura**

Uma limitação desses algoritmos de consenso baseados em PBFT é que todos eles exigem um conjunto conhecido e fixo de participantes. Mais problemático ainda, além de certo número de participantes, eles começam a se degradar: a complexidade de comunicação aumenta drasticamente, eles se tornam extremamente lentos, eleger um líder fica complicado, etc.

Como uma criptomoeda decide quem são os participantes do consenso? Existem várias maneiras, mas as duas mais comuns são:

* **Prova de autoridade (PoA)** — Os participantes do consenso são decididos antecipadamente.
* **Prova de participação (PoS)** — Os participantes do consenso são escolhidos dinamicamente, com base em quem tem mais em jogo (e, portanto, tem menos incentivo para atacar o protocolo). Em geral, criptomoedas baseadas em PoS elegem participantes com base na quantidade de moeda digital que possuem.

Dito isso, nem todos os protocolos de consenso são protocolos clássicos de consenso BFT. O Bitcoin, por exemplo, adotou uma abordagem diferente ao propor um mecanismo de consenso que não tinha uma lista conhecida de participantes. Essa foi uma ideia bastante nova na época, e o Bitcoin conseguiu isso relaxando as restrições dos protocolos de consenso BFT clássicos. Como você verá mais adiante neste capítulo, por causa dessa abordagem, o Bitcoin pode sofrer *forks*, e isso introduz seus próprios conjuntos de desafios.

Sem participantes definidos, como você escolhe um líder? Você poderia usar um sistema de PoS (por exemplo, o protocolo de consenso Ouroboros faz isso). Em vez disso, o consenso do Bitcoin baseia-se em um mecanismo probabilístico chamado prova de trabalho (*proof of work*, PoW). No Bitcoin, isso se traduz em pessoas tentando encontrar soluções para quebra-cabeças a fim de se tornarem participantes e líderes. O quebra-cabeça é criptográfico, como você verá mais adiante neste capítulo.

Devido à ausência de participantes conhecidos, o Bitcoin é chamado de rede *permissionless* (sem permissão). Em uma rede *permissionless*, você não precisa de permissões adicionais para participar do consenso; qualquer um pode participar. Isso contrasta com redes *permissioned* (com permissão), que têm um conjunto fixo de participantes. Resumo alguns desses novos conceitos na figura 12.1.

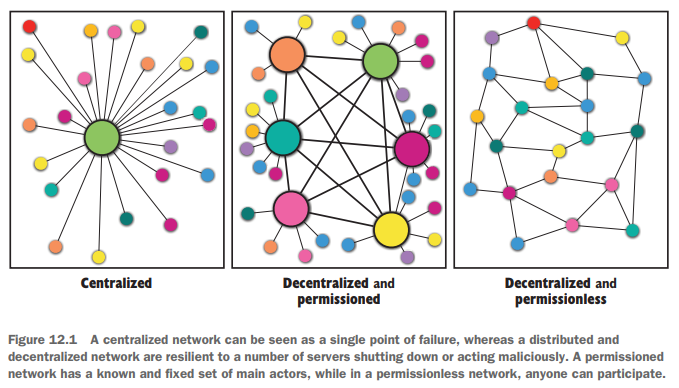


Figura 12.1 Uma rede centralizada pode ser vista como um ponto único de falha, enquanto uma rede distribuída e descentralizada é resiliente a uma série de servidores que podem ser desligados ou agir de forma maliciosa. Uma rede com permissão tem um conjunto conhecido e fixo de atores principais, enquanto em uma rede sem permissão, qualquer pessoa pode participar.

Até recentemente, não se sabia como usar protocolos de consenso BFT clássicos em uma rede *permissionless*, onde qualquer um pode participar. Hoje, existem várias abordagens usando PoS para selecionar dinamicamente um subconjunto menor de participantes como participantes do consenso. Um dos exemplos mais notáveis é o Algorand, publicado em 2017, que seleciona dinamicamente participantes e líderes com base em quanto da moeda eles possuem.

O Bitcoin também afirma ser resistente à censura porque não se pode saber com antecedência quem será o próximo líder e, portanto, não se pode impedir o sistema de eleger um novo líder. Não está tão claro se isso é possível em sistemas PoS, onde pode ser mais fácil descobrir as identidades por trás de grandes somas de moeda.

Devo mencionar que nem todos os protocolos de consenso BFT são baseados em líderes. Alguns são *leaderless* (sem líder), eles não funcionam por meio de líderes eleitos que decidem as próximas transições de estado. Em 2019, a Avalanche lançou uma criptomoeda desse tipo que permitia que qualquer um propusesse mudanças e participasse do consenso.

Por fim, se você achava que o consenso era absolutamente necessário para um sistema de pagamento descentralizado, isso também não é exatamente verdade. Protocolos sem consenso foram propostos em 2018 em *"AT2: Asynchronous Trustworthy Transfers"* por Guerraoui, Kuznetsov, Monti, Pavlovic e Seredinschi. Com isso em mente, não falarei de protocolos sem consenso neste capítulo, pois eles ainda são relativamente novos e não foram suficientemente testados em campo. No restante deste capítulo, abordarei duas criptomoedas diferentes para demonstrar aspectos variados do campo:

* Bitcoin — A criptomoeda mais popular baseada em PoW, introduzida em 2008.
* Diem — Uma criptomoeda baseada no protocolo de consenso BFT, anunciada pelo Facebook e um grupo de outras empresas em 2019.

**12.2 Como o Bitcoin funciona?**

Em 31 de outubro de 2008, um(s) pesquisador(es) anônimo(s) publicou(aram) *"Bitcoin: A Peer-to-Peer Electronic Cash System"* sob o pseudônimo Satoshi Nakamoto. Até hoje, permanece desconhecido quem é Satoshi Nakamoto. Pouco tempo depois, “eles” lançaram o cliente principal do Bitcoin, um software que qualquer um pode executar para ingressar e participar na rede Bitcoin. Isso era tudo o que o Bitcoin precisava: usuários suficientes para executarem o mesmo software ou, pelo menos, o mesmo algoritmo. A primeira criptomoeda da história nasceu — o bitcoin (ou BTC).

O Bitcoin é uma verdadeira história de sucesso. A criptomoeda está funcionando há mais de uma década (no momento em que escrevo isso) e tem permitido que usuários de todo o mundo realizem transações utilizando a moeda digital. Em 2010, Laszlo Hanyecz, um desenvolvedor, comprou duas pizzas por 10.000 BTCs. Enquanto escrevo estas linhas (fevereiro de 2021), um BTC vale quase 57.000 dólares. Assim, já se pode concluir que as criptomoedas podem, às vezes, ser extremamente voláteis.

**12.2.1 Como o Bitcoin gerencia saldos de usuários e transações**

Vamos mergulhar mais fundo nos bastidores do Bitcoin, primeiro analisando como o Bitcoin gerencia os saldos dos usuários e as transações. Como usuário do Bitcoin, você lida diretamente com criptografia. Você não tem um nome de usuário e senha para acessar um site como em qualquer banco; em vez disso, você possui um par de chaves ECDSA (*Elliptic Curve Digital Signature Algorithm*) que você mesmo gera. O saldo de um usuário é simplesmente uma quantidade de BTC associada a uma chave pública e, como tal, para receber BTCs, você simplesmente compartilha sua chave pública com os outros.

Para usar seus BTCs, você assina uma transação com sua chave privada. Uma transação basicamente diz o que você pensa que ela diz: "Eu envio X BTC para a chave pública Y", ignorando alguns detalhes que explicarei mais adiante.

NOTA: No capítulo 7, mencionei que o Bitcoin usa a curva secp256k1 com ECDSA. A curva não deve ser confundida com a curva P-256 do NIST, que é conhecida como secp256r1.

A segurança dos seus fundos está diretamente ligada à segurança da sua chave privada. E, como você sabe, o gerenciamento de chaves é difícil. Na última década, problemas de gerenciamento de chaves em criptomoedas levaram à perda (ou roubo) acidental de chaves no valor de milhões de dólares. Seja cuidadoso!

Existem diferentes tipos de transações no Bitcoin, e a maioria das transações vistas na rede, na verdade, ocultam a chave pública do destinatário através de hashing. Nesses casos, o hash de uma chave pública é chamado de endereço de uma conta. (Por exemplo, este é o meu endereço Bitcoin: bc1q8y6p4x3rp32dz80etpyffh6764ray9842egchy.) Um endereço efetivamente esconde a chave pública real da conta até que o proprietário da conta decida gastar os BTCs (nesse caso, a pré-imagem do endereço precisa ser revelada para que outros possam verificar a assinatura). Isso torna os endereços mais curtos em tamanho e impede que alguém recupere sua chave privada caso o ECDSA um dia seja quebrado.

O fato de existirem diferentes tipos de transações é um detalhe interessante do Bitcoin. As transações não são apenas cargas úteis contendo algumas informações; elas são, na verdade, pequenos scripts escritos em um conjunto de instruções inventado e bastante limitado. Quando uma transação é processada, o script precisa ser executado antes que a saída produzida possa determinar se a transação é válida e, se for, quais etapas devem ser tomadas para modificar o estado de todas as contas.

Criptomoedas como o Ethereum levaram essa ideia de *scripting* ao limite permitindo que programas muito mais complexos (os chamados contratos inteligentes) sejam executados quando uma transação é realizada. Existem algumas coisas aqui que ainda não abordei:

* O que há em uma transação?
* O que significa uma transação ser executada? E quem a executa?

Explicarei o segundo item na próxima seção. Por enquanto, vamos analisar o que há dentro de uma transação.

Uma particularidade do Bitcoin é que não há um banco de dados real de saldos de contas. Em vez disso, um usuário possui "bolsos" de BTCs que estão disponíveis para serem gastos, chamados de Saídas de Transação Não Gastas (*Unspent Transaction Outputs* — UTXOs). Você pode pensar no conceito de UTXOs como uma grande tigela, visível para todos, cheia de moedas que apenas seus proprietários podem gastar. Quando uma transação gasta algumas das moedas, elas desaparecem da tigela, e novas aparecem para os beneficiários dessa mesma transação. Essas novas moedas são simplesmente as saídas listadas na transação.

Para saber quantos BTCs você possui em sua conta, você teria que contar todos os UTXOs que estão atribuídos ao seu endereço. Em outras palavras, você teria que somar todo o dinheiro que foi enviado para você e que ainda não foi gasto. A figura 12.2 dá um exemplo que ilustra como os UTXOs são usados em transações.

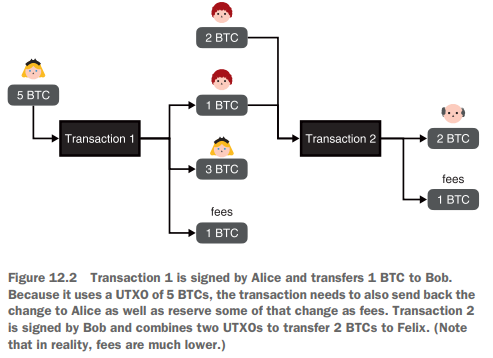


Figura 12.2 A transação 1 é assinada por Alice e transfere 1 BTC para Bob. Como utiliza um UTXO de 5 BTC, a transação também precisa enviar o troco de volta para Alice, além de reservar parte desse troco como taxas. A transação 2 é assinada por Bob e combina dois UTXOs para transferir 2 BTC para Felix. (Observe que, na realidade, as taxas são muito menores.)

Agora surge uma pergunta tipo "ovo e galinha": de onde vieram os primeiros UTXOs? Isso eu responderei na próxima seção.

**12.2.2 Minerando BTCs na era digital do ouro**

Agora você entende o que há em uma transação de Bitcoin e como gerenciar sua conta ou descobrir o saldo de alguém. Mas quem realmente mantém o controle de todas essas transações? A resposta é: todos!

De fato, usar o Bitcoin significa que cada transação deve ser compartilhada publicamente e registrada na história. O Bitcoin é um livro contábil somente de acréscimos (*append-only ledger*) — um livro de transações onde cada página está conectada à anterior. Quero enfatizar aqui que *append-only* significa que você não pode voltar atrás e alterar uma página no livro. Observe também que, como cada transação é pública, a única aparência de anonimato que você tem é o fato de que pode ser difícil descobrir quem é quem (em outras palavras, qual chave pública está ligada a qual pessoa na vida real).

Qualquer um pode inspecionar qualquer transação que tenha ocorrido desde o início do Bitcoin baixando um cliente Bitcoin e usando-o para baixar toda a história. Ao fazer isso, você se torna parte da rede e deve reexecutar cada transação de acordo com as regras codificadas no cliente Bitcoin. Claro, a história do Bitcoin é bastante massiva: no momento em que escrevo, possui cerca de 300 GB, e pode levar dias, dependendo de sua conexão, para baixar todo o livro contábil do Bitcoin. Você pode inspecionar as transações de forma mais fácil usando um serviço online que faz o trabalho pesado por você (desde que confie no serviço online). Dou um exemplo desses chamados exploradores de blockchain na figura 12.3.

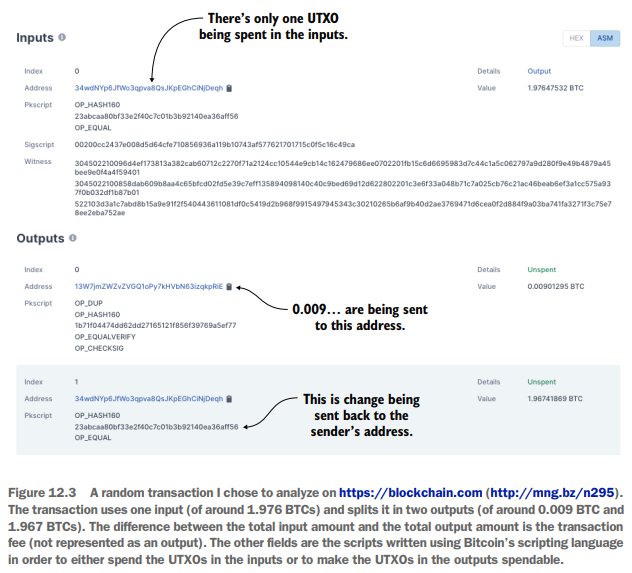


Figura 12.3 Uma transação aleatória que escolhi analisar em https://blockchain.com (http://mng.bz/n295). A transação usa uma entrada (de aproximadamente 1,976 BTCs) e a divide em duas saídas (de aproximadamente 0,009 BTCs e 1,967 BTCs). A diferença entre o valor total da entrada e o valor total da saída é a taxa de transação (não representada como uma saída). Os outros campos são os scripts escritos usando a linguagem de script do Bitcoin para gastar os UTXOs nas entradas ou para tornar os UTXOs nas saídas gastáveis.

O Bitcoin é, na realidade, apenas uma lista de todas as transações que foram processadas desde sua origem (chamamos isso de *genesis*) até agora. Isso deve fazer você se perguntar: quem é responsável por escolher e ordenar as transações neste livro contábil?

Para chegar a um acordo sobre a ordenação das transações, o Bitcoin permite que qualquer um (até mesmo você) proponha uma lista de transações para serem incluídas na próxima página do livro contábil. Essa proposta contendo uma lista de transações é chamada de *bloco* em termos de Bitcoin. Mas permitir que qualquer um proponha um bloco seria um desastre, já que há muitos participantes no Bitcoin. Em vez disso, queremos que apenas uma pessoa faça uma proposta para o próximo bloco de transações. Para fazer isso, o Bitcoin faz com que todos trabalhem em um quebra-cabeça probabilístico, e somente permite que aquele que resolver o quebra-cabeça primeiro proponha seu bloco. Este é o mecanismo de prova de trabalho (*proof of work* — PoW) de que falei anteriormente. O PoW do Bitcoin baseia-se em encontrar um bloco que, ao ser *hashado*, resulte em um *digest* menor que um determinado valor. Em outras palavras, o *digest* do bloco deve ter uma representação binária começando com uma certa quantidade de zeros.

Além das transações que você deseja incluir, o bloco deve conter o *hash* do bloco anterior. Assim, o livro contábil do Bitcoin é realmente uma sucessão de blocos, onde cada bloco se refere ao anterior, até o primeiro bloco, o *genesis block*. É isso que o Bitcoin chama de *blockchain*. A beleza do *blockchain* é que a menor modificação em um bloco tornaria a cadeia inválida, já que o *digest* do bloco também mudaria e, consequentemente, quebraria a referência do próximo bloco a ele.

Observe que, como participante que busca propor o próximo bloco, você não precisa mudar muito em seu bloco para derivar um novo *hash* a partir dele. Você pode fixar a maior parte de seu conteúdo primeiro (as transações que inclui, o *hash* do bloco que ele estende, etc.) e então modificar apenas um campo (chamado de *nonce* do bloco) para impactar o *hash* do bloco. Você pode tratar esse campo como um contador, incrementando o valor até encontrar um *digest* que atenda às regras do jogo, ou pode gerar um valor aleatório. Eu ilustro essa ideia de *blockchain* na figura 12.4.

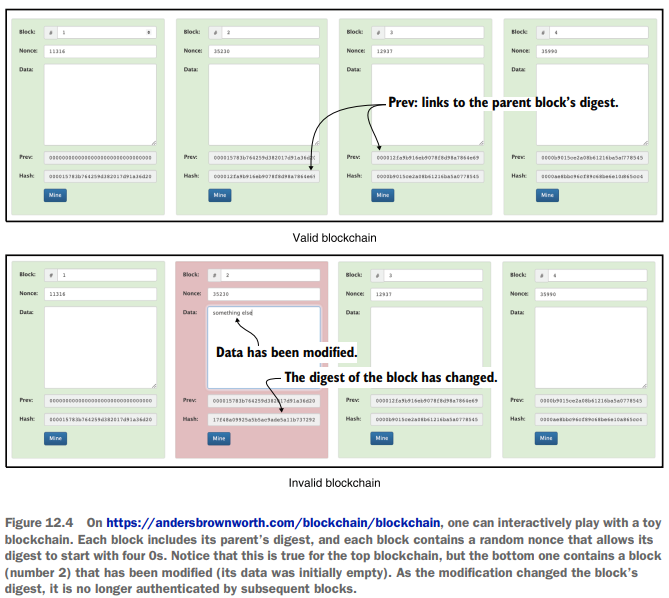


Figura 12.4 Em https://andersbrownworth.com/blockchain/blockchain, é possível brincar interativamente com uma blockchain de brinquedo. Cada bloco inclui o resumo de seu bloco pai e cada bloco contém um nonce aleatório que permite que seu resumo comece com quatro 0s. Observe que isso se aplica à blockchain superior, mas a inferior contém um bloco (número 2) que foi modificado (seus dados estavam inicialmente vazios). Como a modificação alterou o resumo do bloco, ele não é mais autenticado pelos blocos subsequentes.

Tudo isso funciona porque todos estão executando o mesmo protocolo usando as mesmas regras. Quando você se sincroniza com o *blockchain*, você baixa cada bloco de outros pares e verifica que:

* *Hashando* cada bloco, de fato, resulta em um *digest* que é menor do que algum valor esperado.
* Cada bloco se refere ao bloco anterior na história.

Nem todos precisam propor blocos, mas você pode, se quiser. Se o fizer, você é chamado de minerador (*miner*). Isso significa que, para que suas transações entrem no *blockchain*, você precisa da ajuda dos mineradores (como ilustrado na figura 12.5).

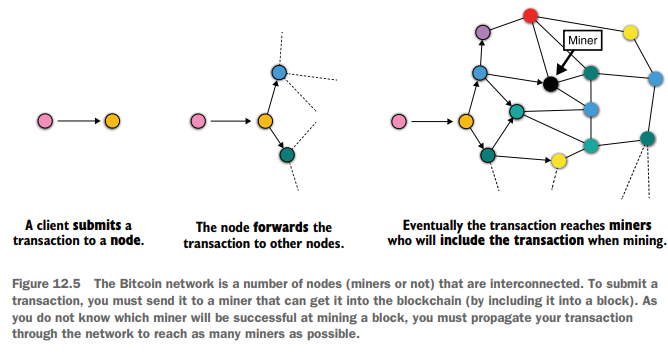


Figura 12.5 A rede Bitcoin é composta por vários nós (mineradores ou não) interconectados. Para enviar uma transação, você deve enviá-la a um minerador que possa inseri-la na blockchain (incluindo-a em um bloco). Como você não sabe qual minerador terá sucesso na mineração de um bloco, você deve propagar sua transação pela rede para alcançar o maior número possível de mineradores.

Os mineradores não trabalham de graça. Se um minerador encontrar um bloco, ele coleta:

* **Uma recompensa** — Um número fixo de BTCs será criado e enviado para seu endereço. No início, os mineradores recebiam 50 BTCs por bloco minerado. Mas o valor da recompensa reduz pela metade a cada 210.000 blocos e será eventualmente reduzido a 0, limitando o total de BTCs que podem ser criados a 21 milhões.
* **Todas as taxas de transação contidas no bloco** — Por isso, aumentar as taxas em suas transações permite que elas sejam aceitas mais rapidamente, pois os mineradores tendem a incluir transações com taxas mais altas nos blocos que mineram.

É assim que os usuários do Bitcoin são incentivados a fazer o protocolo avançar. Um bloco sempre contém o que é chamado de *coinbase*, que é o endereço que coleta a recompensa e as taxas. O minerador geralmente define o *coinbase* para o próprio endereço.

Agora podemos responder à pergunta que fizemos no início da seção: de onde vieram os primeiros UTXOs? A resposta é que todos os BTCs na história foram, em algum momento, criados como parte da recompensa de bloco para os mineradores.

**12.2.3 Inferno dos forks! Resolvendo conflitos na mineração**

O Bitcoin distribui a tarefa de escolher o próximo conjunto de transações a serem processadas via um sistema baseado em PoW. Sua chance de minerar um bloco está diretamente correlacionada à quantidade de *hashes* que você pode computar e, portanto, à quantidade de computação que você pode produzir. Muito poder computacional hoje em dia é direcionado à mineração de blocos no Bitcoin ou em outras criptomoedas baseadas em PoW.

NOTA: O PoW pode ser visto como a forma do Bitcoin de lidar com ataques *sybil*, que são ataques que se aproveitam do fato de que você pode criar quantas contas quiser em um protocolo, dando-lhe uma vantagem assimétrica sobre participantes desonestos. No Bitcoin, a única maneira de obter mais poder é realmente comprar mais hardware para computar *hashes*, não criar mais endereços na rede.

Ainda há um problema: a dificuldade de encontrar um *hash* que seja menor do que um certo valor não pode ser muito fácil. Se for, então a rede terá muitos participantes minerando um bloco válido ao mesmo tempo. E, se isso acontecer, qual bloco minerado é o próximo bloco legítimo na cadeia? É essencialmente o que chamamos de *fork* (bifurcação).

Para resolver *forks*, o Bitcoin possui dois mecanismos. O primeiro é manter a dificuldade do PoW. Se os blocos forem minerados muito rapidamente ou muito lentamente, o algoritmo do Bitcoin que todos estão executando adapta dinamicamente as condições da rede e aumenta ou diminui a dificuldade do PoW. Simplificando, os mineradores precisam encontrar um *digest* de bloco que tenha mais ou menos zeros.

NOTA: Se a dificuldade determina que um *digest* de bloco precisa começar com 1 byte 0, espera-se que você tente 2⁸ blocos diferentes (mais especificamente diferentes *nonces*, como explicado anteriormente) até encontrar um *digest* válido. Eleve isso a 2 bytes e agora espera-se que você tente 2¹⁶ blocos diferentes. O tempo que leva para você chegar lá depende da quantidade de poder que você tem e se possui hardware especializado para computar esses *hashes* mais rapidamente. Atualmente, o algoritmo do Bitcoin altera dinamicamente a dificuldade para que um bloco seja minerado a cada 10 minutos.

Nosso segundo mecanismo é garantir que todos tenham a mesma maneira de seguir em frente caso um *fork* aconteça. Para isso, a regra é seguir a cadeia com a maior quantidade de trabalho. O artigo do Bitcoin de 2008 afirmou: “a cadeia mais longa não apenas serve como prova da sequência de eventos presenciada, mas como prova de que ela veio do maior grupo de poder de CPU”, ditando que os participantes devem respeitar o que veem como a cadeia mais longa. O protocolo foi posteriormente atualizado para seguir a cadeia com a maior quantidade acumulada de trabalho, mas essa distinção não importa muito aqui. Eu ilustro isso na figura 12.6.

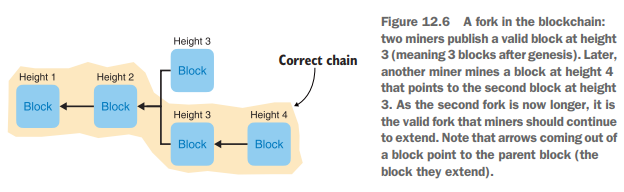


Figura 12.6 Uma bifurcação na blockchain: dois mineradores publicam um bloco válido na altura 3 (ou seja, 3 blocos após a gênese). Posteriormente, outro minerador minera um bloco na altura 4 que aponta para o segundo bloco na altura 3. Como a segunda bifurcação agora é mais longa, é a bifurcação válida que os mineradores devem continuar a estender. Observe que as setas que saem de um bloco apontam para o bloco pai (o bloco que eles estendem).

Disse anteriormente que o algoritmo de consenso do Bitcoin não é um protocolo BFT. Isso ocorre porque o algoritmo de consenso permite tais *forks*. Assim, se você estiver aguardando o processamento de sua transação, absolutamente não deve confiar simplesmente em observar sua transação sendo incluída em um bloco! O bloco observado pode, na verdade, ser um *fork*, e um *fork* perdedor (para um *fork* mais longo) ainda por cima.

Você precisa de mais garantias para decidir quando sua transação foi realmente processada. A maioria das carteiras e plataformas de câmbio espera que um número de blocos de confirmação seja minerado sobre o seu bloco. Quanto mais blocos sobre o bloco que inclui sua transação, menor a chance de que essa cadeia seja reorganizada para outra, devido à existência de um *fork* mais longo.

O número de confirmações geralmente é definido como 6 blocos, o que faz com que o tempo de confirmação da sua transação seja de cerca de uma hora. Dito isso, o Bitcoin ainda não fornece 100% de garantia de que um *fork* após 6 blocos nunca ocorrerá. Se a dificuldade de mineração estiver bem ajustada, então deveria estar tudo bem, e temos motivos para acreditar que isso é verdade para o Bitcoin.

A dificuldade do PoW do Bitcoin aumentou gradualmente ao longo do tempo à medida que a criptomoeda se torna mais popular. A dificuldade agora é tão alta que a maioria das pessoas não pode pagar o hardware necessário para ter uma chance de minerar um bloco. Hoje, a maioria dos mineradores se junta no que são chamados de *mining pools* (pools de mineração) para distribuir o trabalho necessário para minerar um bloco. Eles então compartilham a recompensa.

Com o bloco 632874 [...] o trabalho acumulado esperado na blockchain do Bitcoin ultrapassou 292 *double-SHA256 hashes*.

— Pieter Wuille (2020, [http://mng.bz/aZNJ](http://mng.bz/aZNJ" \t "_new))

Para entender por que os *forks* são disruptivos, vamos imaginar o seguinte cenário. Alice compra uma garrafa de vinho de você, e você estava esperando que ela lhe enviasse os 5 BTCs que possui em sua conta. Finalmente, você observa um novo bloco na altura 10 (ou seja, 10 blocos após o *genesis*) que inclui a transação dela. Sendo cauteloso, você decide esperar que mais 6 blocos sejam adicionados sobre aquele. Após algum tempo, você finalmente vê um bloco na altura 16 que estende a cadeia contendo seu bloco na altura 10. Você envia a garrafa de vinho para Alice e dá o dia por encerrado. Mas essa não é a história completa.

Mais tarde, um bloco na altura 30 aparece do nada, estendendo uma cadeia diferente que se ramificou apenas um bloco antes da sua (na altura 9). Como a nova cadeia é mais longa, ela acaba sendo aceita por todos como a cadeia legítima. A cadeia anterior na qual você estava (a partir do seu bloco na altura 10) é descartada, e os participantes da rede simplesmente reorganizam suas cadeias para agora apontar para a nova mais longa. E como você pode imaginar, essa nova cadeia não tem nenhum bloco que inclua a transação de Alice. Em vez disso, inclui uma transação movendo todos os fundos dela para outro endereço, impedindo você de republicar a transação original que movia os fundos dela para o seu endereço. Alice efetivamente gastou duas vezes seu dinheiro.

Esse é um ataque de 51%. O nome vem da quantidade de poder computacional que Alice precisou para realizar o ataque; ela precisou de apenas um pouco mais do que todos os demais. (<https://crypto51.app> possui uma tabela interessante que lista o custo de realizar um ataque de 51% em diferentes criptomoedas baseadas em PoW.) Este não é apenas um ataque teórico! Ataques de 51% acontecem no mundo real. Por exemplo, em 2018, um atacante conseguiu gastar duas vezes uma quantidade de fundos em um ataque de 51% na moeda Vertcoin.

O atacante essencialmente reescreveu parte da história do livro contábil e então, usando seu poder de *hashing* dominante para produzir a cadeia mais longa, convenceu o restante dos mineradores a validar essa nova versão do *blockchain*. Com isso, ele ou ela pôde cometer o crime supremo das criptomoedas: um gasto duplo de transações anteriores, deixando os recebedores anteriores com moedas invalidadas.

— Michael J. Casey ("Vertcoin’s Struggle Is Real: Why the Latest Crypto 51% Attack Matters", 2018)

Em 2019, a mesma coisa aconteceu com o Ethereum Classic (uma variante do Ethereum), causando perdas de mais de 1 milhão de dólares na época com várias reorganizações de mais de 100 blocos de profundidade. Em 2020, o Bitcoin Gold (uma variante do Bitcoin) também sofreu um ataque de 51%, removendo 29 blocos da história da criptomoeda e gastando duas vezes mais de 70.000 dólares em menos de dois dias.

**12.2.4 Reduzindo o tamanho de um bloco usando árvores de Merkle**

Um último aspecto interessante do Bitcoin que quero abordar é como ele comprime algumas das informações disponíveis. Um bloco no Bitcoin, na verdade, não contém nenhuma transação! As transações são compartilhadas separadamente e, em vez disso, um bloco contém um único *digest* que autentica uma lista de transações. Esse *digest* poderia simplesmente ser o *hash* de todas as transações contidas no bloco, mas é um pouco mais engenhoso do que isso. Em vez disso, o *digest* é a raiz de uma árvore de Merkle.

O que é uma árvore de Merkle? Simplificando, é uma árvore (estrutura de dados) onde os nós internos são *hashes* de seus filhos. Isso pode ser um pouco confuso, e uma imagem vale mais que mil palavras, então veja a figura 12.7.

Árvores de Merkle são estruturas úteis, e você as encontrará em todos os tipos de protocolos do mundo real. Elas podem comprimir uma grande quantidade de dados em um valor pequeno e de tamanho fixo — a raiz da árvore. Não só isso, você não precisa necessariamente de todas as folhas para reconstruir a raiz.

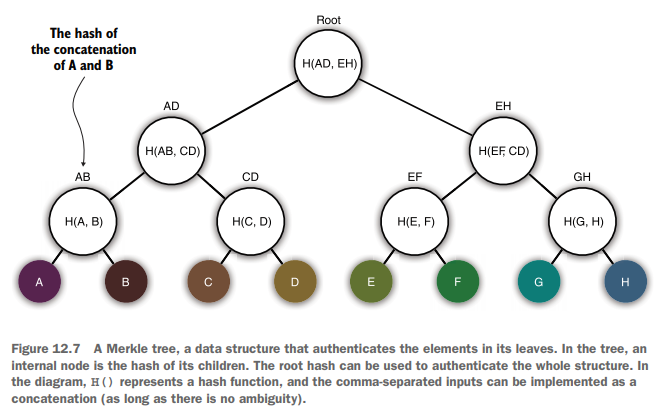


Figura 12.7 Uma árvore de Merkle, uma estrutura de dados que autentica os elementos em suas folhas. Na árvore, um nó interno é o hash de seus filhos. O hash raiz pode ser usado para autenticar toda a estrutura. No diagrama, H() representa uma função de hash, e as entradas separadas por vírgula podem ser implementadas como uma concatenação (desde que não haja ambiguidade).

Por exemplo, imagine que você conhece a raiz da árvore de Merkle devido à sua inclusão em um bloco de Bitcoin, e você quer saber se uma transação (uma folha da árvore) está incluída no bloco. Se ela estiver na árvore, o que posso fazer é compartilhar com você os nós vizinhos no caminho até a raiz como uma prova de pertencimento. (Uma prova que é logarítmica na profundidade da árvore em tamanho.) O que sobra para você é computar os nós internos até a raiz da árvore, *hasheando* cada par no caminho. É um pouco complicado explicar isso por escrito, então ilustro a prova na figura 12.8.

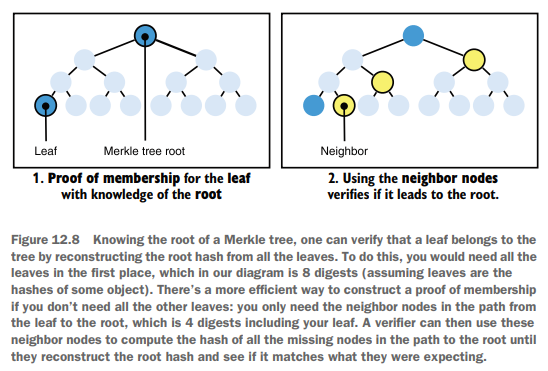


Figura 12.8 Conhecendo a raiz de uma árvore de Merkle, pode-se verificar se uma folha pertence à árvore reconstruindo o hash da raiz a partir de todas as folhas. Para isso, você precisaria de todas as folhas em primeiro lugar, o que em nosso diagrama são 8 resumos (assumindo que as folhas são os hashes de algum objeto). Há uma maneira mais eficiente de construir uma prova de pertencimento se você não precisar de todas as outras folhas: você precisa apenas dos nós vizinhos no caminho da folha até a raiz, o que são 4 resumos incluindo sua folha. Um verificador pode então usar esses nós vizinhos para calcular o hash de todos os nós ausentes no caminho até a raiz até reconstruir o hash da raiz e verificar se ele corresponde ao esperado.

O motivo de se usar árvores de Merkle em um bloco, em vez de listar todas as transações diretamente, é tornar mais leve a quantidade de informações que precisam ser baixadas para realizar consultas simples no *blockchain*. Por exemplo, imagine que você queira verificar se sua transação recente está incluída em um bloco sem precisar baixar toda a história do *blockchain* do Bitcoin. O que você pode fazer é baixar apenas os cabeçalhos dos blocos, que são mais leves, pois não contêm as transações, e uma vez que tenha isso, pedir a um par para lhe informar qual bloco incluiu sua transação. Se houver tal bloco, eles devem ser capazes de lhe fornecer uma prova de que sua transação está na árvore autenticada pelo *digest* que você possui no cabeçalho do bloco.

Há muito mais a dizer sobre o Bitcoin, mas há apenas algumas páginas restantes neste livro. Em vez disso, usarei o espaço restante deste capítulo para dar um passeio pelo campo e explicar como funcionam os protocolos clássicos de consenso BFT.

**12.3 Um passeio pelas criptomoedas**

O Bitcoin é a primeira criptomoeda bem-sucedida e permanece sendo a criptomoeda com a maior participação de mercado e valor, apesar de centenas de outras criptomoedas terem sido criadas. O que é interessante é que o Bitcoin tinha, e ainda tem, muitos problemas que outras criptomoedas tentaram resolver (e algumas com sucesso). Ainda mais interessante, o campo das criptomoedas fez uso de muitos primitivos criptográficos que até agora não tinham muitas aplicações práticas ou sequer existiam! Assim, sem mais delongas, as seções a seguir listam os problemas que têm sido pesquisados desde o surgimento do Bitcoin.

**12.3.1 Volatilidade**

Atualmente, a maioria das pessoas usa criptomoedas como veículos de especulação. O preço do Bitcoin obviamente contribui para essa história, já que mostrou que pode facilmente oscilar milhares de dólares para cima ou para baixo em um único dia. Algumas pessoas afirmam que a estabilidade virá com o tempo, mas o fato é que o Bitcoin não é utilizável como moeda nos dias de hoje. Outras criptomoedas têm experimentado o conceito de *stablecoin*, atrelando o preço de seu token a uma moeda fiduciária existente (como o dólar americano).

**12.3.2 Latência**

Você pode medir a eficiência de uma criptomoeda de várias maneiras. A *throughput* de uma criptomoeda é o número de transações por segundo que ela pode processar. A *throughput* do Bitcoin, por exemplo, é bastante baixa, com apenas 7 transações por segundo.

Por outro lado, a *finality* é o tempo que leva para sua transação ser considerada finalizada uma vez que ela é incluída no *blockchain*. Devido aos *forks*, a *finality* do Bitcoin nunca é completamente alcançada. Considera-se que, pelo menos uma hora após uma transação ser incluída em um novo bloco, a probabilidade de a transação ser revertida torna-se aceitável. Ambos os números impactam grandemente a latência, que é o tempo que leva para uma transação ser finalizada do ponto de vista do usuário.

No Bitcoin, a latência inclui a criação da transação, o tempo que leva para propagá-la através da rede, o tempo que leva para ser incluída em um bloco, e, por fim, o tempo de espera para o bloco ser confirmado.

A solução para esses problemas de velocidade pode ser resolvida por protocolos BFT, que geralmente oferecem *finality* de poucos segundos com garantia de que não há *forks*, bem como *throughput* na ordem de milhares de transações por segundo. Ainda assim, às vezes isso não é suficiente, e diferentes tecnologias estão sendo exploradas. Os chamados protocolos de camada 2 (*layer 2*) tentam fornecer soluções adicionais que possam permitir pagamentos mais rápidos *off-chain* (fora da cadeia), enquanto salvam o progresso periodicamente no *blockchain* principal (referido como camada 1 em comparação).

**12.3.3 Tamanho do blockchain**

Outro problema comum com o Bitcoin e outras criptomoedas é que o tamanho do *blockchain* pode rapidamente crescer para tamanhos impraticáveis. Isso cria problemas de usabilidade quando usuários que desejam usar a criptomoeda (por exemplo, para consultar o saldo de sua conta) precisam primeiro baixar toda a cadeia para interagir com a rede. Criptomoedas baseadas em BFT que processam um grande número de transações por segundo devem facilmente atingir terabytes de dados dentro de meses ou até semanas.

Várias tentativas existem para resolver isso.

Uma das mais interessantes é o Mina, que não exige que você baixe toda a história do *blockchain* para obter o estado mais recente. Em vez disso, o Mina usa provas de conhecimento zero (*zero-knowledge proofs*, ZKPs), mencionadas no capítulo 7 e que abordarei com mais profundidade no capítulo 15, para comprimir toda a história em uma prova de tamanho fixo de 11 KB. Isso é especialmente útil para clientes mais leves, como celulares, que normalmente precisam confiar em servidores de terceiros para consultar o *blockchain*.

**12.3.4 Confidencialidade**

O Bitcoin fornece pseudo-anonimato na medida em que as contas estão apenas vinculadas a chaves públicas. Enquanto ninguém puder vincular uma chave pública a uma pessoa, a conta associada permanece anônima. Lembre-se de que todas as transações de e para aquela conta são publicamente disponíveis, e grafos sociais ainda podem ser criados para entender quem tende a negociar mais frequentemente com quem, e quem possui quanto da moeda.

Existem muitas criptomoedas que tentam resolver essas questões usando ZKPs ou outras técnicas. O Zcash é uma das criptomoedas confidenciais mais conhecidas, pois suas transações podem criptografar o endereço do remetente, o endereço do destinatário e o valor sendo transacionado. Tudo isso usando ZKPs!

**12.3.5 Eficiência energética**

O Bitcoin tem sido fortemente criticado por consumir muita eletricidade. De fato, a Universidade de Cambridge avaliou recentemente que toda a energia gasta minerando BTCs coloca o Bitcoin entre os 30 maiores consumidores de energia do mundo (se visto como um país), consumindo mais energia em um ano do que um país como a Argentina (fevereiro de 2021; <https://cbeci.org/>).

Os protocolos BFT, por outro lado, não dependem de PoW e assim evitam essa sobrecarga pesada. É quase certo que por isso qualquer criptomoeda moderna evita um consenso baseado em PoW, e até mesmo criptomoedas importantes baseadas em PoW como o Ethereum anunciaram planos de migrar para protocolos de consenso mais ecológicos. Antes de passar para o próximo capítulo, vamos dar uma olhada nessas criptomoedas baseadas em protocolos de consenso BFT.

**12.4 DiemBFT: Um protocolo de consenso tolerante a falhas bizantinas (BFT)**

Muitas criptomoedas modernas abandonaram o aspecto de PoW do Bitcoin por protocolos de consenso mais ecológicos e eficientes. A maioria desses protocolos de consenso baseia-se nos protocolos clássicos de consenso BFT, que são, em sua maioria, variantes do protocolo PBFT original. Nesta última seção, usarei o Diem para ilustrar tais protocolos BFT.

O Diem (anteriormente chamado Libra) é uma moeda digital inicialmente anunciada pelo Facebook em 2019, e governada pela Diem Association, uma organização de empresas, universidades e organizações sem fins lucrativos que buscam impulsionar uma rede de pagamentos aberta e global. Uma particularidade do Diem é que ele é respaldado por dinheiro real, usando uma reserva de moedas fiduciárias. Isso permite que a moeda digital seja estável, ao contrário de seu primo mais antigo, o Bitcoin. Para operar a rede de pagamentos de forma segura e aberta, é usado um protocolo de consenso BFT chamado DiemBFT, que é uma variante do HotStuff. Nesta seção, veremos como o DiemBFT funciona.

**12.4.1 Segurança e vitalidade: As duas propriedades de um protocolo de consenso BFT**

Um protocolo de consenso BFT busca alcançar duas propriedades, mesmo na presença de uma porcentagem tolerada de participantes maliciosos. Essas propriedades incluem:

* **Segurança (Safety)** — Nenhum estado contraditório pode ser acordado, o que significa que *forks* não devem acontecer (ou acontecem com probabilidade negligenciável).
* **Vitalidade (Liveness)** — Quando as pessoas enviam transações, o estado acabará por processá-las. Em outras palavras, ninguém pode impedir o protocolo de continuar funcionando.

Observe que um participante geralmente é considerado malicioso (também chamado de bizantino) se não se comportar de acordo com o protocolo. Isso pode significar que ele não está fazendo nada, ou que não está seguindo as etapas do protocolo na ordem correta, ou que não está respeitando alguma regra obrigatória destinada a garantir que não haja *fork*, e assim por diante.

Normalmente, é relativamente direto para os protocolos BFT de consenso alcançarem segurança, enquanto a vitalidade é conhecida por ser mais difícil. De fato, existe um famoso resultado de impossibilidade de Fischer, Lynch e Paterson (*"Impossibility of distributed consensus with one faulty process"*) datado de 1985 e vinculado aos protocolos BFT, que afirma que:

nenhum protocolo determinístico de consenso pode tolerar falhas em uma rede assíncrona (onde as mensagens podem levar quanto tempo quiserem para chegar).

A maioria dos protocolos BFT evita esse resultado de impossibilidade considerando que a rede é um tanto síncrona (e, de fato, nenhum protocolo é útil se sua rede ficar fora do ar por um longo período) ou introduzindo aleatoriedade no algoritmo.

Por essa razão, o DiemBFT nunca sofre *forks*, mesmo sob condições extremas de rede. Além disso, ele sempre faz progresso, mesmo quando há partições na rede onde diferentes partes da rede não conseguem alcançar outras partes, contanto que a rede acabe se recuperando e estabilizando por um período suficientemente longo.

**12.4.2 Um turno no protocolo DiemBFT**

O Diem opera em um ambiente *permissioned* onde os participantes (chamados validadores) são conhecidos antecipadamente. O protocolo avança em rodadas estritamente crescentes (rodada 1, 2, 3, etc.), durante as quais os validadores se revezam para propor blocos de transações. Em cada rodada:

1. O validador escolhido para liderar (deterministicamente) coleta um número de transações, agrupa-as em um novo bloco que estende o *blockchain*, então assina o bloco e o envia a todos os outros validadores.
2. Ao receber o bloco proposto, outros validadores podem votar para certificá-lo, assinando-o e enviando a assinatura para o líder da próxima rodada.
3. Se o líder da próxima rodada receber votos suficientes para aquele bloco, pode agrupá-los em um chamado certificado de quórum (*quorum certificate*, QC), que certifica o bloco, e usar o QC para propor um novo bloco (na rodada seguinte) estendendo o bloco agora certificado.

Outra maneira de enxergar isso é que, enquanto no Bitcoin um bloco contém apenas o *hash* do bloco que ele estende, no DiemBFT, um bloco também contém um número de assinaturas sobre esse *hash*. (O número de assinaturas é importante, mas falarei disso mais adiante.)

Observe que, se os validadores não virem uma proposta durante uma rodada (porque o líder está ausente, por exemplo), eles podem marcar o tempo como esgotado e alertar os outros validadores de que nada aconteceu. Nesse caso, a próxima rodada é acionada e o proponente pode estender o bloco certificado de maior altura que tenha visto. Recapitulo isso na figura 12.9.

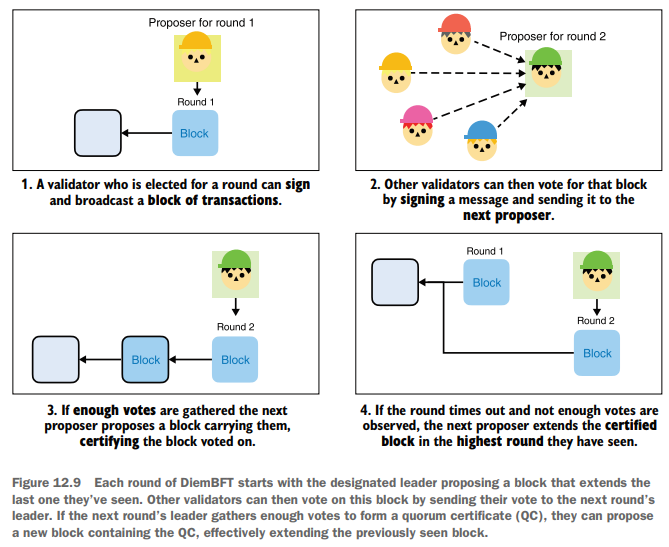


Figura 12.9 Cada rodada do DiemBFT começa com o líder designado propondo um bloco que estende o último bloco visto. Outros validadores podem então votar neste bloco enviando seu voto ao líder da próxima rodada. Se o líder da próxima rodada reunir votos suficientes para formar um certificado de quórum (CQ), ele pode propor um novo bloco contendo o CQ, efetivamente estendendo o bloco visto anteriormente.

**12.4.3 Quanto de desonestidade o protocolo pode tolerar?**

Vamos imaginar que queremos ser capazes de tolerar no máximo *f* validadores maliciosos (mesmo que todos colaborem entre si), então o DiemBFT estabelece que deve haver pelo menos **3f + 1** validadores participando do protocolo (em outras palavras, para *f* validadores maliciosos, deve haver pelo menos **2f + 1** validadores honestos). Enquanto essa suposição for verdadeira, o protocolo garante segurança e vitalidade.

Com isso em mente, os QCs (certificados de quórum) só podem ser formados com uma maioria de votos de validadores honestos, o que corresponde a **2f + 1** assinaturas se houver **3f + 1** participantes. Esses números podem ser um pouco difíceis de visualizar, então mostro como eles impactam a confiança nos votos que observamos na figura 12.10.

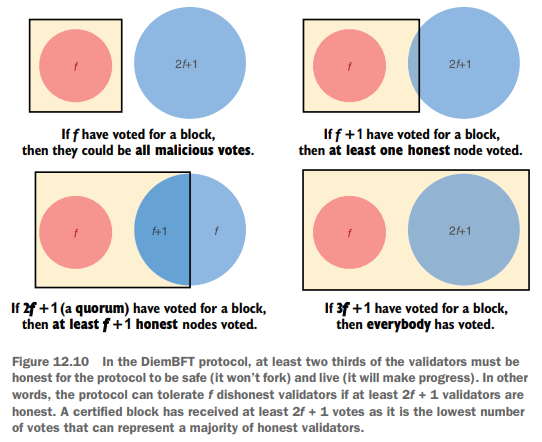


Figura 12.10 No protocolo DiemBFT, pelo menos dois terços dos validadores devem ser honestos para que o protocolo seja seguro (não bifurque) e funcione (progrida). Em outras palavras, o protocolo pode tolerar f validadores desonestos se pelo menos 2f + 1 validadores forem honestos. Um bloco certificado recebeu pelo menos 2f + 1 votos, pois é o menor número de votos que pode representar a maioria dos validadores honestos.

**12.4.4 As regras de votação do DiemBFT**

Os validadores devem seguir duas regras de votação o tempo todo; caso contrário, são considerados bizantinos:

1. Eles não podem votar no passado (por exemplo, se você acabou de votar na rodada 3, só pode votar a partir da rodada 4 em diante).
2. Eles só podem votar em um bloco que estenda um bloco de sua *rodada preferida* ou superior.

O que é uma *rodada preferida*? Por padrão, é 0, mas se você votar em um bloco que estende um bloco que, por sua vez, estende outro bloco (ou seja, você votou em um bloco que tem um bloco avô), então a rodada do bloco avô torna-se sua rodada preferida, a menos que sua rodada preferida anterior fosse mais alta. Complicado? Eu sei, por isso fiz a figura 12.11.

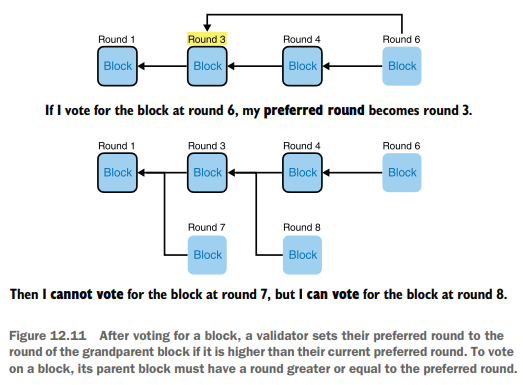


Figura 12.11 Após votar em um bloco, um validador define sua rodada preferencial como a rodada do bloco pai, caso esta seja superior à sua rodada preferencial atual. Para votar em um bloco, seu bloco pai deve ter uma rodada maior ou igual à rodada preferencial.

**12.4.5 Quando as transações são consideradas finalizadas?**

Note que blocos que são certificados ainda não estão finalizados, ou como também dizemos, *committed*. Ninguém deve assumir que as transações contidas em blocos pendentes não serão revertidas. Os blocos e as transações que contêm só podem ser considerados finalizados quando a regra de *commit* for acionada. A regra de *commit* (ilustrada na figura 12.12) diz que um bloco e todos os blocos pendentes que ele estende tornam-se *committed* se:

* O bloco inicia uma cadeia de 3 blocos que são propostos em rodadas contíguas (por exemplo, nas rodadas 1, 2 e 3).
* O último bloco da cadeia de 3 blocos torna-se certificado.

E é basicamente isso que há no protocolo em um nível alto. Mas, claro, mais uma vez, o diabo está nos detalhes.

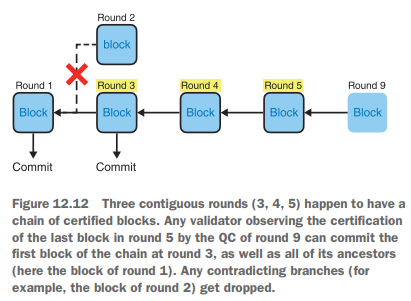


Figura 12.12 Três rodadas contíguas (3, 4, 5) possuem uma cadeia de blocos certificados. Qualquer validador que observe a certificação do último bloco da rodada 5 pelo QC da rodada 9 pode confirmar o primeiro bloco da cadeia na rodada 3, bem como todos os seus ancestrais (neste caso, o bloco da rodada 1). Quaisquer ramificações contraditórias (por exemplo, o bloco da rodada 2) são descartadas.

**12.4.6 As intuições por trás da segurança do DiemBFT**

Embora eu o encoraje a ler a prova de segurança de uma página no artigo do DiemBFT, quero usar algumas páginas aqui para lhe dar uma intuição de por que ele funciona. Primeiro, notamos que dois blocos diferentes não podem ser certificados durante a mesma rodada. Esta é uma propriedade importante, que explico visualmente na figura 12.13.

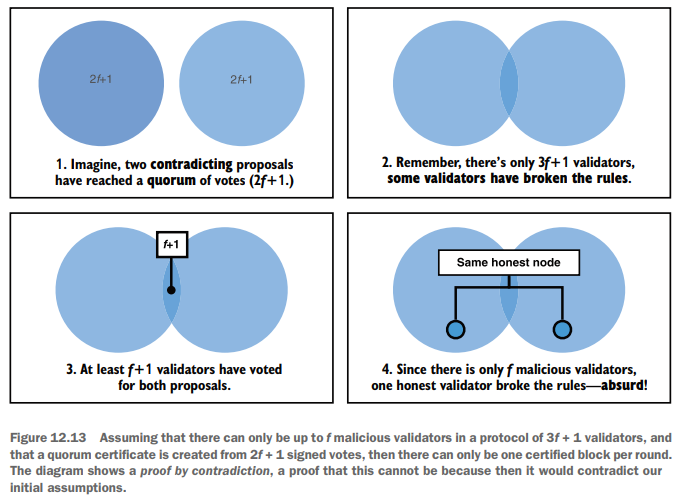


Figura 12.13 Supondo que só pode haver até f validadores maliciosos em um protocolo de 3f + 1 validadores, e que um certificado de quorum é criado a partir de 2f + 1 votos assinados, então só pode haver um bloco certificado por rodada. O diagrama mostra uma prova por contradição, uma prova de que isso não pode ocorrer, pois contradiria nossas suposições iniciais.

Usando a propriedade de que apenas um bloco pode ser certificado em uma rodada dada, podemos simplificar como falamos sobre os blocos: o bloco 3 está na rodada 3, o bloco 6 está na rodada 6, e assim por diante. Agora, observe a figura 12.14 e dedique um momento para descobrir por que um bloco certificado, ou dois blocos certificados, ou três blocos certificados em rodadas não contíguas não podem levar a um *commit* sem o risco de um *fork*.

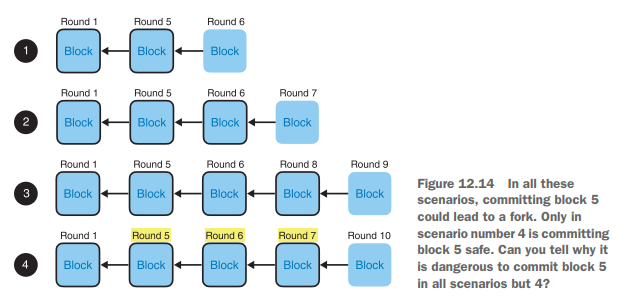


Figura 12.14 Em todos esses cenários, a confirmação do bloco 5 pode levar a uma bifurcação. Somente no cenário 4 é seguro confirmar o bloco 5. Você consegue explicar por que é perigoso confirmar o bloco 5 em todos os cenários, exceto no 4?

Você conseguiu encontrar as respostas para todos os cenários? A resposta curta é que todos os cenários, exceto o último, deixam espaço para que um bloco estenda a rodada 1. Esse bloco tardio efetivamente cria um ramo e pode ser estendido posteriormente de acordo com as regras do protocolo de consenso. Se isso acontecer, o bloco 5 e os outros blocos que o estendem serão descartados, à medida que outro ramo anterior for *committed*. Para os cenários 1 e 2, isso pode ocorrer devido ao proponente não ter visto os blocos anteriores. No cenário 3, um bloco anterior pode aparecer mais tarde do que o esperado, talvez devido a atrasos de rede ou, pior, devido a um validador retê-lo até o momento oportuno. Explico isso mais detalhadamente na figura 12.15.

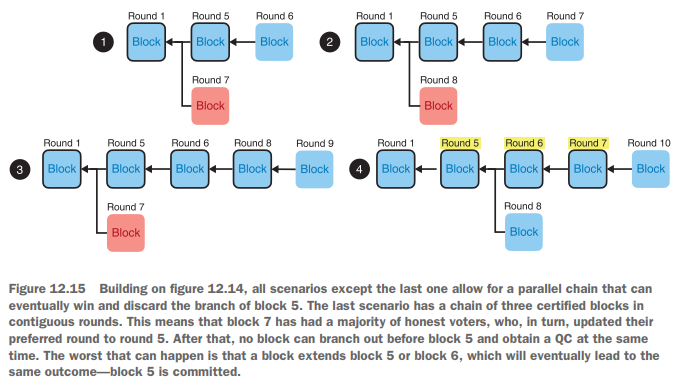


Figura 12.15 Com base na Figura 12.14, todos os cenários, exceto o último, permitem uma cadeia paralela que pode eventualmente vencer e descartar a ramificação do bloco 5. O último cenário tem uma cadeia de três blocos certificados em rodadas contíguas. Isso significa que o bloco 7 teve a maioria dos votantes honestos, que, por sua vez, atualizaram sua rodada preferida para a rodada 5. Depois disso, nenhum bloco pode ramificar antes do bloco 5 e obter um QC ao mesmo tempo. O pior que pode acontecer é que um bloco estenda o bloco 5 ou o bloco 6, o que eventualmente levará ao mesmo resultado — o bloco 5 é confirmado.

**Resumo**

* As criptomoedas consistem em descentralizar uma rede de pagamentos para evitar um ponto único de falha.
* Para que todos concordem com o estado de uma criptomoeda, podemos usar algoritmos de consenso.
* Protocolos de consenso tolerantes a falhas bizantinas (BFT) foram inventados em 1982 e evoluíram para se tornarem mais rápidos e mais simples de entender.
* Protocolos de consenso BFT precisam de um conjunto conhecido e fixo de participantes para funcionar (rede permissionada). Tais protocolos podem decidir quem faz parte desse conjunto de participantes (prova de autoridade ou PoA) ou eleger dinamicamente o conjunto de participantes com base na quantidade de moeda que possuem (prova de participação ou PoS).
* O algoritmo de consenso do Bitcoin (o consenso de Nakamoto) usa prova de trabalho (PoW) para validar a cadeia correta e permitir que qualquer pessoa participe (rede permissionless).
* O PoW do Bitcoin faz com que os participantes (chamados mineradores) computem muitos *hashes* para encontrar alguns com prefixos específicos. Encontrar com sucesso um *digest* válido permite que um minerador decida o próximo bloco de transações e colete uma recompensa, bem como taxas de transação.
* As contas no Bitcoin são simplesmente pares de chaves ECDSA usando a curva secp256k1. Um usuário sabe quanto BTC sua conta possui verificando todas as saídas de transações não gastas (UTXOs). Uma transação é, assim, uma mensagem assinada autorizando a movimentação de um número de saídas de transações anteriores para novas saídas, gastáveis por diferentes chaves públicas.
* O Bitcoin usa árvores de Merkle para comprimir o tamanho de um bloco e permitir que a verificação de inclusão de transações seja pequena em tamanho.
* *Stablecoins* são criptomoedas que tentam estabilizar seus valores, geralmente atrelando seu token ao valor de uma moeda fiduciária como o dólar americano.
* As criptomoedas usam os chamados protocolos de camada 2 para reduzir sua latência processando transações *off-chain* e salvando o progresso *on-chain* periodicamente.
* Provas de conhecimento zero (ZKPs) são usadas em muitas aplicações diferentes de *blockchain* (por exemplo, no Zcash para fornecer confidencialidade e no Coda para comprimir todo o *blockchain* em uma prova curta de validade).
* O Diem é uma *stablecoin* que usa um protocolo de consenso BFT chamado DiemBFT. Ele permanece seguro (sem *forks*) e vivo (o progresso sempre é feito) enquanto não houver mais do que *f* participantes maliciosos de um total de 3f + 1 participantes.
* O DiemBFT funciona por meio de rodadas em que um participante propõe um bloco de transações que estende um bloco anterior. Outros participantes podem então votar no bloco, potencialmente criando um certificado de quórum (QC) se votos suficientes forem reunidos (2f + 1).
* No DiemBFT, os blocos e suas transações são finalizados quando a regra de *commit* (uma cadeia de 3 blocos certificados em rodadas contíguas) é acionada. Quando isso acontece, o primeiro bloco da cadeia e os blocos que ele estende são *committed*.