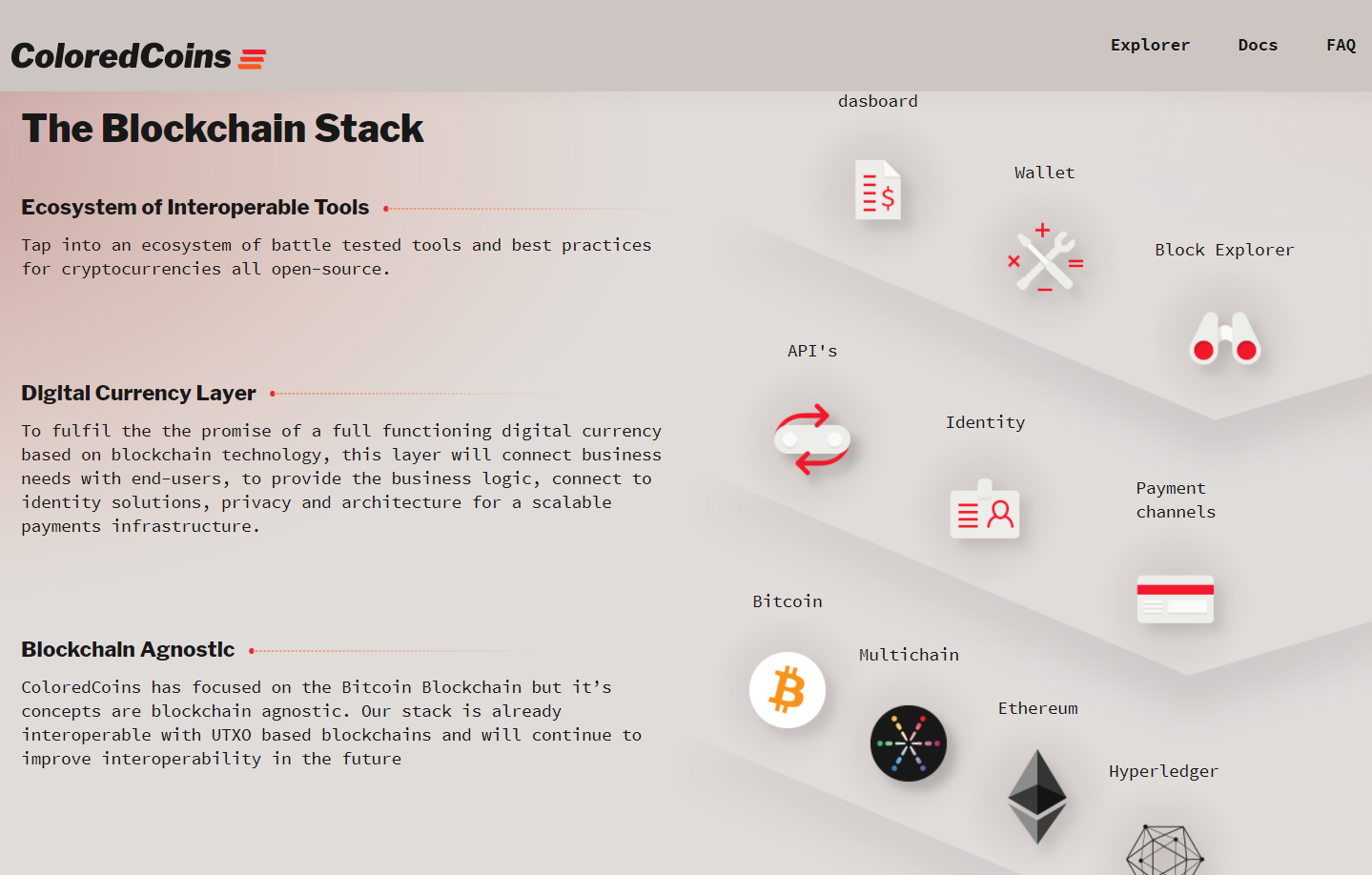
Ethereum

Uma Plataforma para Smart Contracts e Aplicações Decentralizadas da Próxima Geração

O desenvolvimento do Bitcoin por Satoshi Nakamoto em 2009 tem frequentemente sido aclamado como uma revolução do dinheiro e da moeda, sendo o primeiro exemplo de ativo digital que simultaneamente não possui nenhum lastro ou "valor intrínseco (1)" e nenhum emissor/controlador centralizado. Entretanto, outra -- provavelmente mais importante -- parte do experimento do Bitcoin é a tecnologia do blockchain -- o "motor" da criptomoeda -- como uma ferramenta de consenso distribuído, e a atenção está rapidamente começando a mudar em direção a esse outro aspecto do Bitcoin. Dentre as aplicações alternativas da tecnologia do blockchain, normalmente incluem-se a utilização de **ativos digitas** no blockchain para representar moedas personalizadas e instrumentos financeiros ("colored coins" (2)), a propriedade de um dispositivo físico subjacente ("smart property" (3)), ativos não-fungíveis, como domínios na internet ("Namecoin" (4)), tal como aplicações mais complexas envolvendo ter ativos digitais sendo controlados diretamente por um pedaço de código implementando regras arbitrárias ("smart contracts") ou até mesmo "organizações autônomas descentralizadas" (Em inglês, Decentralized Autonomous Organizations -- DAOs) baseadas no blockchain. O que o Ethereum pretende prover é um blockchain com uma linguagem de programação verdadeiramente Turing-completa, que pode ser utilizada para criar "contratos" que podem ser usados para codificar funções arbitrárias de transição de estado, permitindo que os usuários criem qualquer um dos sistemas descritos acima, assim como muitos outros que nós sequer ainda imaginamos, simplesmente escrevendo sua lógica em poucas linhas de código.

1. <https://bitcoinmagazine.com/articles/you-say-bitcoin-has-no-intrinsic-value-twenty-two-reasons-to-think-again-1399454061/>
2. <https://docs.google.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0lIzrTLuoWu2z1BE/edit#heading=h.wxrvzqj8997r>
3. <https://en.bitcoin.it/wiki/Smart_Property>
4. <https://namecoin.org/>



# História

O conceito de moeda digital descentralizada, assim como as aplicações alternativas como registro de propriedade, já existe há décadas. Os protocolos anônimos de e-cash dos anos 1980 e 1990 eram basicamente dependentes de uma primitiva criptográfica conhecida como Chaumian Blinding. Chaumian Blinding provia essas novas moedas com alto grau de privacidade, mas seus protocolos falhavam miseravelmente em ganhar força por causa da sua dependência de um intermediário centralizado. Em 1998, Wei Dai criou o b-money (1), que se tornou a primeira proposta de se introduzir a ideia de criar dinheiro através da resolução de enigmas computacionais e também como consenso descentralizado, mas a proposta possuía poucos detalhes sobre como o consenso descentralizado iria ser implementado de fato. Em 2005, Hal Finney introduziu o conceito de "provas de trabalho reutilizáveis (em inglês, reusable proof-of-work) (2)", um sistema que usa ideias do b-money junto com os computacionalmente custosos enigmas Hashcash -- criados por Adam Back -- para criar um conceito para uma criptomoeda, entretanto, mais uma vez ficou aquém do ideal por depender de computação confiável como backend. Em 2009, uma moeda descentralizada foi implementada pela primeira vez na prática por Satoshi Nakamoto, combinando as conhecidas primitivas para gestão da propriedade através de criptografia de chave pública (public key cryptography) com um algoritmo de consenso para manter o controle de quem possui moedas, conhecido como “proof of work"

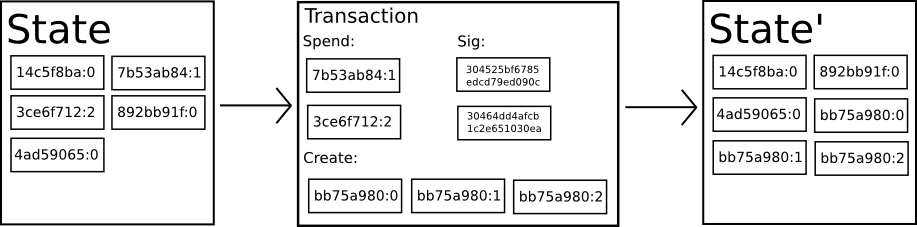
1. <http://www.weidai.com/bmoney.txt>
2. <http://www.finney.org/~hal/rpow/>

O mecanismo por trás do proof of work foi uma revolução porque ele resolveu dois problemas simultaneamente. Primeiro, ele provia um algoritmo de consenso simples e razoavelmente efetivo, permitindo que nós em uma rede coletivamente concordássemos sobre um conjunto de atualizações no estado do livro-contábil do Bitcoin. Segundo, ele provia um mecanismo para permitir a livre entrada no processo de consenso, resolvendo o problema político de decidir quem pode influenciar no consenso, enquanto simultaneamente previne Sybil attacks (1). O proof of work faz isso substituindo uma barreira formal à participação, como o requerimento de ser registrado como uma entidade única em uma lista particular, com uma barreira econômica: o peso de um único nó no processo de votação por consenso é diretamente proporcional ao poder computacional que o mesmo possui. Desde então, uma abordagem alternativa tem sido proposta -- a chamada "prova de participação" (em inglês, proof of stake), calculando o peso de um nó como sendo proporcional as suas posses na criptomoeda, e não a seus recursos computacionais. A discussão a respeito dos méritos relativos das duas abordagens está além do escopo deste paper, mas é importante notar que ambas podem ser usadas para servir como backbone de uma criptomoeda.

1. <https://en.wikipedia.org/wiki/Sybil_attack>

**Conversa: Proof of work ou Proof of Stake?**

# Bitcoin como um Sistema de Transição de Estados



Do ponto de vista técnico, o livro-contábil de uma criptomoeda como o Bitcoin pode ser imaginado como um sistema de transição de estados, onde existe um "estado" consistindo no status de propriedade de todos os bitcoins existentes e uma "função de transição de estado", que recebe um estado e uma transação e gera um novo estado como resultado. Em um sistema bancário tradicional, por exemplo, o estado é o balancete, uma transação é uma requisição de mover $X de A para B, e a função de transição de estados reduz o valor da conta de A em $X e aumenta o valor da conta de B em $X. Se a conta de A tem menos que $X de saldo, a função de transição retorna um erro. Assim, pode-se definir:

APPLY(S,TX) -> S' or ERROR

No sistema bancário definido acima:

APPLY({ Alice: $50, Bob: $50 },"Envie $20 de Alice para Bob") = { Alice: $30, Bob: $70 }

Mas...

APPLY({ Alice: $50, Bob: $50 },"Envie $70 de Alice para Bob") = ERROR

O "estado" no Bitcoin é o conjunto de todas as moedas (tecnicamente, "saídas não-gastas de transações" -- em inglês, unspent transaction outputs ou UTXO) que foram mineradas, mas ainda não gastas, com cada UTXO tendo uma denominação e um dono (definido por um endereço de 20-bytes que é essencialmente uma chave criptográfica pública[1]. Uma transação contém uma ou mais entradas, com cada entrada contendo uma referência para um UTXO existente e uma assinatura criptográfica produzida pela chave privada associada à chave ao endereço do dono, e uma ou mais saídas, com cada uma contendo um novo UTXO para ser adicionado no novo estado.

A função de transição de estado APPLY(S,TX) -> S' pode ser definida aproximadamente como segue:

1. Para cada input em TX:
   * Se o UTXO referenciado não pertence a S, retorne um erro.
   * Se a assinatura fornecida não pertence ao dono do UTXO, retorne um erro.
2. Se a soma das denominações de todas as entradas UTXO é menos que a soma das denominações de todas as saídas UTXO, retorne um erro.
3. Retorne S' com todas as entradas UTXO removidas e todas as saídas UTXO adicionadas.

A primeira metade do primeiro passo previne que o remetente da transação gaste moedas que não existem; a segunda metade previne que o remetente gaste moedas de outras pessoas. O segundo passo força a conservação de valor. Afim de usar isso tudo para pagamento, o protocolo funciona desta forma: suponha que Alice queira enviar 11,7 BTC para Bob. Primeiro, Alice irá procurar por um conjunto dos UTXO disponíveis que ela possua e que juntos somem pelo menos 11,7 BTC. Realisticamente, Alice raramente será capaz de obter exatamente 11,7 BTC; digamos que a menor quantia que ela consegue juntar é 6 + 4 + 2 = 12. Ela então cria uma transação com aquelas três entradas e duas saídas. A primeira saída será os 11.7 BTC com o endereço de Bob como seu dono e a segunda será o "troco" com os 0,3 BTC restantes. Se Alice não reivindicar o troco para si mesma, enviando-o para um endereço que ela possui, o minerador terá o direito de mantê-lo para si ("fique com o troco").

# Mineração



Se nós tivéssemos acesso a um serviço centralizado confiável, este sistema seria trivial de se implementar; ele poderia ser codificado exatamente como descrito, usando o HD do servidor centralizado para manter o controle do estado da rede. Entretanto, com o Bitcoin estamos tentando construir um sistema monetário descentralizado, então precisaremos combinar o sistema de transição de estados com um sistema de consenso de modo a garantir que todos concordem com a ordem das transações. O processo de consenso descentralizado do Bitcoin requer que os nós na rede continuamente tentem produzir pacotes de transações chamados de "blocos". É esperado que a rede crie um novo bloco válido a cada dez minutos aproximadamente, com cada bloco contendo um *timestamp*, um *nonce*, uma referência para (ex.: um *hash* do) o bloco anterior e uma lista de todas as transações que ocorreram desde o último bloco. Com o passar do tempo, isso cria um "blockchain" persistente, que cresce indefinidamente e é continuamente atualizado para representar o último estado do livro-contábil do Bitcoin.

O algoritmo para checar que um bloco é válido, expressado nesse paradigma, é como segue:

1. Checar se o bloco anterior referenciado pelo bloco atual existe e é válido.
2. Checar se o *timestamp* do bloco é maior que o do bloco anterior e menos que 2 horas no futuro.
3. Checar se o *proof of work* do bloco é válido.
4. Deixar que S[0] seja o estado ao fim do bloco anterior.
5. Supor que TX é a lista de transações do bloco com n transações. Para cada i em 0...n-1, faça S[i+1] = APPLY (S[i], TX[i]). Se alguma aplicação retornar um erro, saia e retorne falso.
6. Retornar true e registrar S[n] como o estado ao final do bloco atual.

Essencialmente, cada transação no bloco precisa prover uma transição de estado válida a partir do estado canônico de antes da transação ser executada para um novo estado. Note que o estado não está codificado no bloco de forma alguma; trata-se puramente de uma abstração para ser lembrada pelo nó validador e pode apenas ser (seguramente) computada para qualquer bloco se partindo do estado gênesis e sequencialmente aplicando cada transição em todos os blocos. Adicionalmente, note que a ordem na qual o minerador inclui transações no bloco importa; se há duas transações A e B tal que B gasta um UTXO ("saídas não-gastas de transações") criado por A, então o bloco será válido somente se A vem antes de B, mas não o contrário.

A única condição de validade presente na lista acima que não é encontrada em nenhum outro sistema é o requerimento do *proof of work*. A exata condição é que o duplo-hash-SHA256 de cada bloco -- tratado como um número de 256-bits -- deve ser menor que um *alvo* dinamicamente ajustado, o qual no momento da redação deste texto é aproximadamente 2187. O propósito disto é fazer com que a criação de um bloco seja computacionalmente difícil, assim prevenindo que *Sybil attackers* refaçam todo o blockchain favorecendo-se. Como o SHA256 é projetado para ser uma função pseudoaleatória completamente imprevisível, a única forma de criar um bloco válido é simplesmente por tentativa e erro, repetidamente incrementando o *nonce* e verificando se o novo *hash* satisfaz a condição de ser menor que o *alvo*.

Com o atual *alvo* de ~2187, a rede precisa tentar, em média, ~269 vezes antes de encontrar um bloco válido; em geral, o *alvo* é recalculado pela rede a cada 2016 blocos, para que, na média, um novo bloco seja produzido por algum nó da rede a cada dez minutos. Para compensar os mineradores por esse trabalho computacional, estes possuem o direito de incluir um tipo especial de transação ("coinbase") em cada bloco, creditando -- no momento da redação deste texto -- a eles mesmos 12,5 BTC criados a partir do nada. Além disso, se qualquer transação tem uma denominação total maior em suas entradas que em suas saídas, a diferença também passa a ser de direito do minerador, como uma "taxa de transação". Propositalmente, este é também o único mecanismo pelo qual BTC são emitidos; o estado gênesis não continha nenhuma moeda.

De modo a melhor compreender o propósito da mineração, vamos examinar o que acontece quando existe um atacante malicioso na rede. Como a criptografia subjacente ao Bitcoin é assumida como segura, alguém mal-intencionado irá atacar a única parte do Bitcoin que não é protegido diretamente por criptografia: a ordem das transações. A estratégia do atacante é simples:

1. Enviar 100 BTC para um comerciante em troca de algum produto (preferivelmente, um bem digital de entrega rápida).
2. Esperar pela entrega do produto.
3. Produzir uma outra transação enviando os mesmos 100 BTC para ele mesmo.
4. Tentar convencer a rede que a transação dele para ele mesmo é a que veio primeiro.

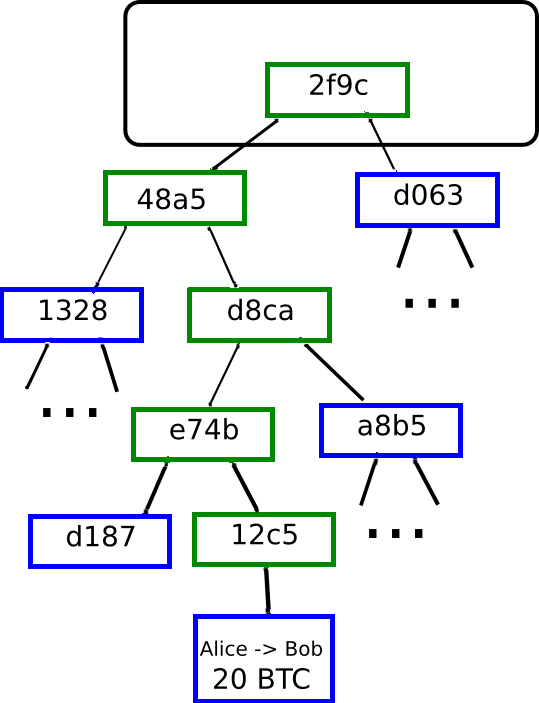
Uma vez que o passo (1) foi realizado, depois de alguns minutos, algum minerador irá incluir a transação em um bloco, digamos o bloco nº 270.000. Depois de aproximadamente uma hora, mais cinco blocos foram adicionados ao blockchain depois daquele bloco, com cada um deles apontando indiretamente para aquela transação e, portanto, "confirmando-a". Neste ponto, o comerciante irá aceitar o pagamento como finalizado e entregar o produto; como estamos assumindo que esta é uma mercadoria digital, a entrega é instantânea. Agora, o atacante cria outra transação enviando 100 BTC para ele mesmo. Se ele simplesmente envia a transação para a rede, ela não será processada; mineradores tentarão rodar APPLY(S,TX) e notar que TX consome um UTXO que não está mais naquele estado S. Então, em vez disso, o atacante cria um *fork* do blockchain, começando por minerar outra versão do bloco 270.000, apontando para o mesmo bloco 269.999 como pais, mas com a nova transação no lugar da original. Visto que os dados do bloco são diferentes, isto requer refazer o *proof of work*. Além disso, a nova versão do bloco 270.000 tem um *hash* diferente, então os blocos 270.001 a 270.005 não "apontam" para ele; deste modo, a corrente original e a nova corrente do atacante estão completamente separadas. A regra é que, no caso de um *fork*, **a cadeia mais longa** do blockchain é assumida como verdadeira e, assim, mineradores honestos irão trabalhar sobre a cadeia com o bloco 270.005, enquanto o atacante estará trabalhando na cadeia com o novo bloco 270.000. A fim de que o atacante consiga fazer seu blockchain o mais longo, ele precisaria ter mais poder computacional que o restante da rede combinada para poder alcançá-la (o que é conhecido como "o ataque dos 51%").

Os blocos dependem do hash de todos os blocos anteriores. Um atacante com imenso poder de computação pode refazer a prova de trabalho (PoW) por uma quantidade considerável de blocos e, eventualmente, pode ganhar muitos bitcoins, mas, como descrito no artigo de Satoshi, a recompensa de mineração de um bloco válido é muito mais do que para interromper a rede.

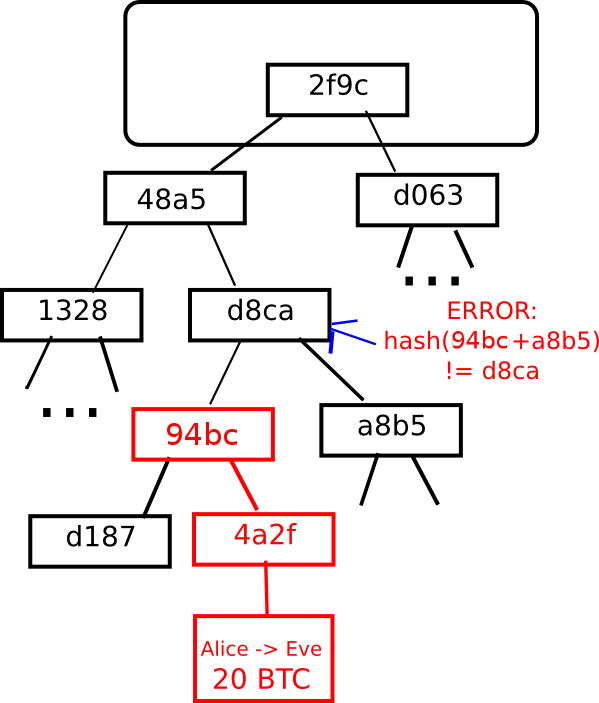
**Conversa: Computação Quântica**

**Conversa: O que vai acontecer em 2.140?**

**Conversa: O que vai acontecer com os custos de GAS no Ethereum?**



01: Basta apresentar apenas um pequeno número de nós em uma árvore Merkle para fornecer uma prova da validade de um ramo.



02: Qualquer tentativa de mudar qualquer parte da árvore Merkle acabará por levar a uma inconsistência em algum lugar da cadeia.

Um importante recurso de escalabilidade do Bitcoin é que o bloco é armazenado em uma estrutura de dados de vários níveis. O "hash" de um bloco é, na verdade, apenas o hash do cabeçalho do bloco, um dado de aproximadamente 200 bytes que contém timestamp, nonce, hash do bloco anterior e o hash da raiz de uma estrutura de dados chamada árvore Merkle armazenando todas as transações No bloco. Uma árvore Merkle é um tipo de árvore binária, composta por um conjunto de nós com um grande número de nós de folha na parte inferior da árvore que contém os dados subjacentes, um conjunto de nós intermediários onde cada nó é o hash de seus dois filhos, E, finalmente, um único nó raiz, também formado a partir do hash de seus dois filhos, representando o "topo" da árvore. O objetivo da árvore Merkle é permitir que os dados em um bloco sejam entregues de forma fragmentada: um nó pode baixar apenas o cabeçalho de um bloco de uma fonte, a parte pequena da árvore relevante para eles de outra fonte e ainda estar seguro que todos os dados estão corretos. A razão pela qual isso funciona é que os hashes se propagam para cima: se um usuário mal-intencionado tentar trocar uma transação falsa no final de uma árvore Merkle, essa alteração causará uma alteração no nó acima e, em seguida, uma mudança no nó acima dessa , Finalmente alterando a raiz da árvore e, portanto, o hash do bloco, fazendo com que o protocolo o registre como um bloco completamente diferente (com certeza com uma prova de trabalho inválida).

O protocolo da árvore Merkle é indiscutivelmente essencial para a sustentabilidade a longo prazo. Um "nó completo" na rede Bitcoin, que armazena e processa a totalidade de cada bloco, ocupa cerca de 15 GB de espaço em disco na rede Bitcoin em abril de 2014 e cresce por mais de um gigabyte por mês.

# Aplicativos alternativos de Blockchain

A iniciativa de levar a idéia do Blockchain subjacente e aplicá-lo a outros conceitos também tem uma longa história. Em 2005, Nick Szabo saiu com o conceito de títulos de propriedade seguros com autoridade proprietária, um documento que descreve como "novos avanços na tecnologia de banco de dados replicado" permitirá um sistema baseado em blocos para armazenar um registro de quem é real proprietário. Criando um quadro elaborado, incluindo conceitos como homesteading, possessão adversa e imposto sobre a terra georgiana. No entanto, infelizmente, nenhum sistema de banco de dados replicado efetivo estava disponível no momento e, portanto, o protocolo nunca foi implementado na prática. Após 2009, no entanto, uma vez que o consenso descentralizado de Bitcoin foi desenvolvido, várias aplicações alternativas começaram rapidamente a surgir.

**Namecoin** - criado em 2010, Namecoin é um banco de dados de registro de nomes descentralizado. Em protocolos descentralizados como Tor, Bitcoin e BitMessage, é necessário que haja alguma maneira de identificar as contas para que outras pessoas possam interagir com elas, mas em todas as soluções existentes, o único tipo de identificador disponível é um hash pseudorandom como 1LW79wp5ZBqaHW1jL5TCiBCrhQYtHagUWy.

Se, alguém gostaria de ter uma conta com um nome como "george". No entanto, o problema é que, se uma pessoa pode criar uma conta chamada "george", outra pessoa pode usar o mesmo processo para se registrar "george" para si mesmo e representá-los. A única solução é um paradigma do primeiro-arquivo, onde o primeiro que realizou o registro é bem-sucedido e o segundo falha - um problema perfeitamente adequado para o protocolo de consenso do Bitcoin. A Namecoin é a implementação mais antiga e mais bem sucedida de um sistema de registro de nomes usando essa idéia.

**Colored coins** - o propósito das moedas coloridas é servir como um protocolo para permitir que as pessoas criem suas próprias moedas digitais. No protocolo de moedas coloridas, esta "emite" uma nova moeda atribuindo publicamente uma cor a um Bitcoin UTXO específico, e o protocolo define de forma recursiva a cor de outros UTXO como sendo a cor das entradas que a transação que os criou gastou (Algumas regras especiais se aplicam no caso de entradas de cores misturadas). Isso permite aos usuários manter carteiras contendo apenas UTXO de uma cor específica e enviá-las em torno de bitcoins regulares, retrocedendo através do bloco de cadeias para determinar a cor de qualquer UTXO que eles recebem.

**Metacoins** - a idéia por trás de uma metacoin é ter um protocolo que atue em cima do Bitcoin, usando transações Bitcoin para armazenar transações de metacoin, mas com uma função de transição de estado diferente, APPLY'. Como o protocolo do metacoin não pode impedir que as transações de metacoin inválidas apareçam na cadeia de bits do Bitcoin, uma regra é adicionada que se APPLY '(S, TX) retornar um erro, o protocolo padrão é APPLY' (S, TX) = S. Isso fornece um Mecanismo fácil para criar um protocolo de criptografia arbitrário, potencialmente com recursos avançados que não podem ser implementados dentro da própria Bitcoin, mas com um custo de desenvolvimento muito baixo, já que as complexidades de mineração e redes já são manipuladas pelo protocolo Bitcoin. As Metacoins foram usadas para implementar algumas classes de contratos financeiros, registro de nomes e troca descentralizada.

Assim, em geral, existem duas abordagens para construir um protocolo de consenso: construir uma rede independente e construir um protocolo acima do Bitcoin. A primeira abordagem, embora bem sucedida no caso de aplicativos como Namecoin, é difícil de implementar.

# Scripts

Mesmo sem extensões, o protocolo Bitcoin realmente facilita uma versão fraca de um conceito de "contratos inteligentes". O UTXO no Bitcoin pode ser possuído não apenas por uma chave pública, mas também por um script complexos, expresso em uma simples linguagem de programação baseada em script. Neste paradigma, uma transação que o UTXO deve fornecer dados que satisfaça o script. Na verdade, mesmo o mecanismo básico de propriedade da chave pública é implementado através de um script: o script leva uma assinatura de curva elíptica como entrada, verifica-o contra a transação e o endereço que possui o UTXO e retorna 1 se a verificação for bem-sucedida e 0 caso contrário. Existem outros scripts mais complicados para vários casos de uso adicionais. Por exemplo, pode-se construir um script que requer assinaturas de duas das três chaves privadas para validar ("multisig"), uma configuração útil para contas corporativas, contas de poupança seguras e algumas situações de custódia do comerciante. Os scripts também podem ser usados ​​para pagar recompensas por soluções para problemas computacionais, e pode-se construir um script que diga algo como "este Bitcoin UTXO é seu se você puder fornecer uma prova SPV de que você enviou uma transação.

No entanto, a linguagem de script implementada no Bitcoin tem várias limitações importantes:

**Falta de Turing** - completude - isto é, enquanto existe um grande subconjunto de computação que a linguagem de script Bitcoin é compatível, ela não dá suporte a tudo. A categoria principal que **falta é** **loops**. Isso é feito para evitar loops infinitos durante a verificação de transações; teoricamente, é um obstáculo superável para programadores de script, uma vez que qualquer loop pode ser simulado simplesmente repetindo o código subjacente muitas vezes com uma instrução if. Por exemplo, a implementação de um algoritmo de assinatura de curva elíptica alternativa provavelmente exigirá 256 rodadas de multiplicação repetidas, todas incluídas individualmente no código.

**Value-blindness** - não há nenhuma maneira para um script UTXO fornecer um controle sobre a quantidade que pode ser retirada. Por exemplo, um caso de uso poderoso de um contrato de oracle seria um contrato de hedge, onde A e B colocavam US $ 1000 em BTC e, após 30 dias, o script envia US $ 1000 de BTC para A e o resto para B. Isso exigiria um Oracle para determinar o valor de 1 BTC em USD.

**Lack of state** (Falta de estado) - UTXO pode ser gasto ou não gasto; não há oportunidade para contratos ou scripts em vários estágios que mantenham qualquer outro estado interno além disso. Isso torna difícil fazer contratos de opções em várias etapas, ofertas de intercâmbio descentralizadas ou protocolos de compromisso criptográficos em dois estágios (necessários para recompensas computacionais seguras). Isso também significa que o UTXO só pode ser usado para construir **contratos simples**.

**Blockchain-blindness** - UTXO são “cegos” para determinados dados do bloco de blocos, como o hash do nonce e do bloco anterior. Isso limita severamente as aplicações, privando a linguagem de script de uma fonte potencialmente valiosa de aleatoriedade.

Assim, vemos três abordagens para a construção de aplicativos avançados em cima da cryptocurrency:

1. Construir uma nova Blockchain
2. Usar scripts encima do Bitcoin
3. Construir um meta-protocolo encima do Bitcoin.

Construir um novo Blockchain permite uma liberdade ilimitada na construção de um conjunto de recursos, mas ao custo do tempo de desenvolvimento, esforço de inicialização e segurança não.

Usar scripts não é fácil de implementar e padronizar, e é muito limitado em suas capacidades.

Os meta-protocolos, embora simples, sofrem falhas na escalabilidade.

**Conversa: Qual Blockchain usar?**

# Ethereum

A intenção da Ethereum é criar um protocolo alternativo para a construção de aplicações descentralizadas, proporcionando um conjunto diferente de compromissos que acreditamos que será muito útil para uma grande variedade de aplicações descentralizadas, com particular ênfase em situações em que o tempo de desenvolvimento, segurança para pequenas e médias empresas. Aplicações raramente usadas, e a capacidade de diferentes aplicações para interagir de forma muito eficiente, são importantes. Ethereum faz isso construindo o que é essencialmente a camada fundacional abstrata final: uma cadeia de blocos com uma linguagem de programação completa de Turing, permitindo que qualquer pessoa escreva contratos inteligentes e aplicativos descentralizados onde eles podem criar suas próprias regras arbitrárias para propriedade, formatos de transação e Funções de transição do estado.

Uma versão da Namecoin e outros protocolos como moedas e sistemas de reputação podem ser construídos podem ser desenvolvidos. Com muito mais poder do que a oferecida pelo script Bitcoin devido aos poderes adicionais de Turing-completeness, Consciência de valor, bloqueio de consciência e estado.

# Contas Ethereum

No Ethereum, o estado é composto de objetos chamados de "contas", com cada conta possuindo um endereço de 20 bytes e transições de estado sendo transferências diretas de valor e informações entre contas. Uma conta Ethereum contém quatro campos:

1. O nonce, um contador usado para garantir que cada transação só possa ser processada uma vez
2. O balanço de éter atual da conta
3. O código do contrato da conta, se presente
4. O armazenamento da conta (vazio por padrão)

"Ether" é o principal combustível da Ethereum, e é usado para pagar taxas de transação. Em geral, existem dois tipos de contas:

1. Contas de propriedade externa, controladas por chaves privadas
2. Contas contratuais, controladas pelo código do contrato.

Uma conta de propriedade externa não tem código, e pode-se enviar mensagens de uma conta de propriedade externa criando e assinando uma transação. Em uma conta de contrato, toda vez que a conta recebe uma mensagem, seu código se ativa, permitindo que ele leia e escreva no armazenamento interno, envie outras mensagens ou crie contratos.

Observe que os "contratos" em Ethereum não devem ser vistos como algo que deve ser "realizado" ou "cumprido". Em vez disso, eles são mais como "agentes autônomos" que vivem dentro do ambiente de execução Ethereum, sempre executando um pedaço específico de código quando "chamado" por uma mensagem ou transação, e ter controle direto sobre seu próprio equilíbrio de éter e sua própria chave.

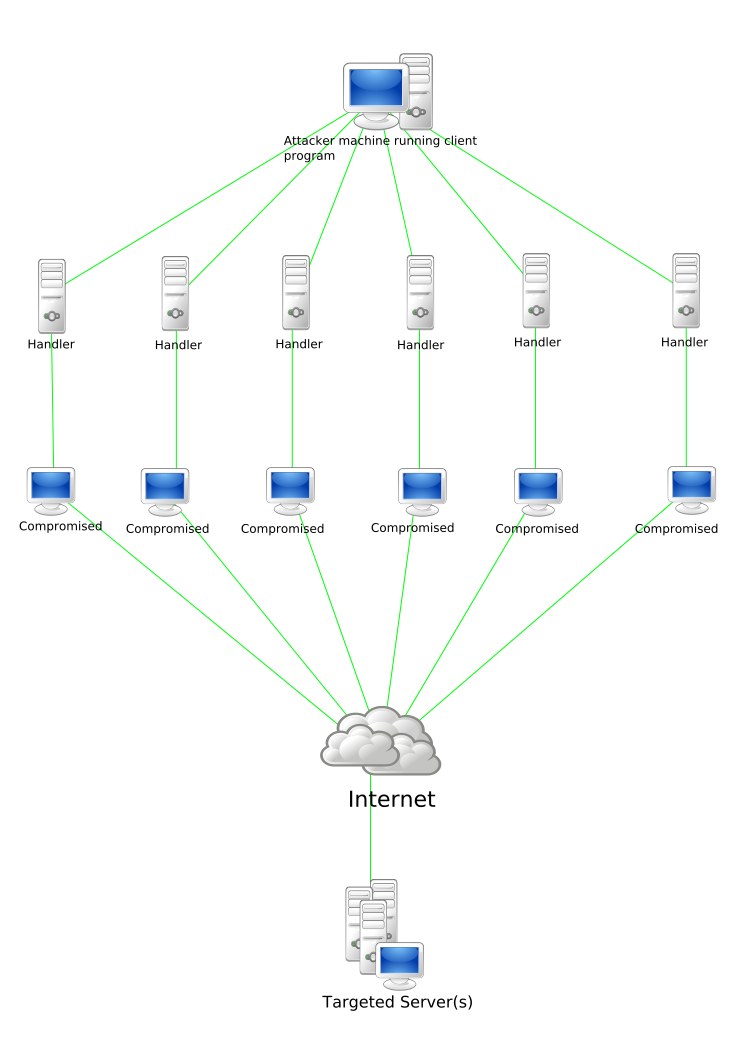
# Mensagens e Transações

O termo "transação" é usado em Ethereum para se referir ao pacote de dados assinado que armazena uma mensagem a ser enviada de uma conta de propriedade externa. As transações contêm:

1. O destinatário da mensagem;
2. Uma assinatura que identifica o remetente;
3. A quantidade de éter para transferência do remetente para o destinatário;
4. Um campo de dados opcional;
5. Um valor de STARTGAS, que representa o número máximo de etapas computacionais, a execução da transação pode ser realizada e
6. Um valor GASPRICE, representando a taxa que a remetente paga por etapa computacional.

Os três primeiros são campos padrão esperados em qualquer cryptocurrency. O campo de dados não tem função por padrão, mas a máquina virtual possui um código operacional com o qual um contrato pode acessar os dados. Como exemplo de caso de uso, se um contrato estiver funcionando como um serviço de registro de domínio, talvez você deseje interpretar os dados que estão sendo transmitidos para ele como contendo dois "campos", sendo o primeiro campo um domínio para registrar e o segundo Campo sendo o endereço IP para registrá-lo. O contrato iria ler esses valores a partir dos dados da mensagem e colocá-los adequadamente no armazenamento.

Os campos STARTGAS e GASPRICE são cruciais para o modelo anti-denial-of-service model de serviço de Ethereum. Para evitar laços infinitos acidentais ou outro desperdício computacional no código, cada transação é necessária para estabelecer um limite para quantos passos computacionais de execução de código podem ser usados.



Um ataque de negação de serviço (ataque DoS) é um ataque cibernético onde o invasor procura fazer com que um recurso de rede ou máquina não esteja disponível para seus usuários pretendidos interrompendo temporariamente ou indefinidamente os serviços de um host conectado à Internet. A negação de serviço normalmente é realizada inundando a máquina ou o recurso visado com solicitações supérfluas na tentativa de sobrecarregar sistemas e evitar que alguns ou todos os pedidos legítimos sejam cumpridos.

A unidade fundamental de computação é "gas"; Geralmente, um passo computacional custa 1 gas, mas algumas operações custam maiores quantidades de gás porque são mais computacionalmente caras ou aumentam a quantidade de dados que devem ser armazenados como parte do estado.

Existe também uma taxa de 5 gas para cada byte nos dados da transação. A intenção do sistema de taxas é exigir que um invasor pague proporcionalmente por cada recurso que eles consomem, incluindo computação, largura de banda e armazenamento; portanto, qualquer transação que leva a rede a consumir uma quantidade maior de qualquer desses recursos deve ter uma taxa de gás aproximadamente proporcional ao incremento.

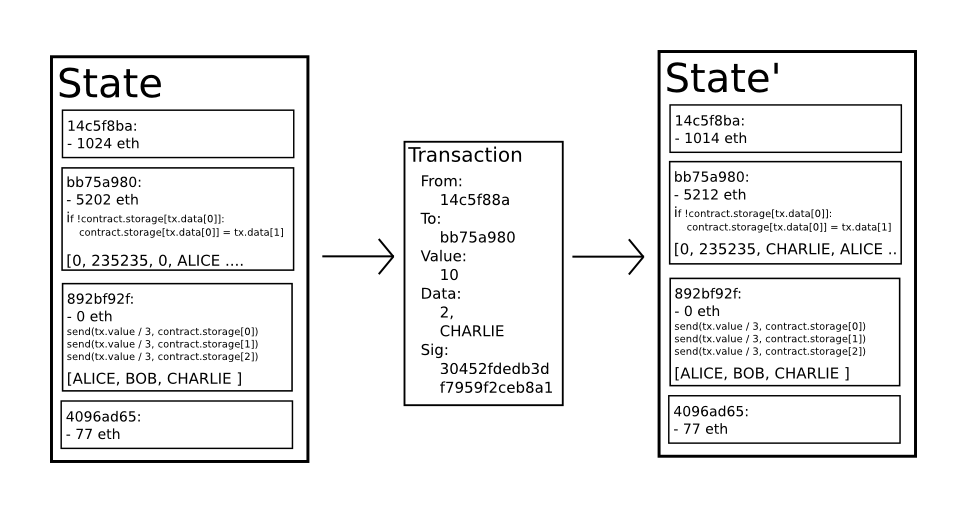
Observe que o subsídio de gas atribuído por uma transação ou contrato aplica-se ao total de gas consumido por essa transação e todas as sub-execuções.

Por exemplo, se a pessoa A envia uma transação para B com 1.000 gas; B consome 600 gas antes de enviar uma mensagem para C e a execução interna de C consome 300 gas antes de retornar; Então, B pode gastar mais 100 gas antes de ficar sem gas.

**Conversa: Porque será que todas as aplicações Ethereum são “objetivas”?**

**Pensando “descentralizado”**

# Função de Transição do Estado Ethereum



A função de transição do estado Ethereum, APPLY (S, TX) -> S 'pode ser definida da seguinte maneira:

Verifique se a transação está bem formada (ou seja, tem o número certo de valores), a assinatura é válida e o nonce corresponde ao nonce na conta do remetente. Caso contrário, devolva um erro.

Calcule a taxa de transação como STARTGAS \* GASPRICE e determine o endereço de envio da assinatura. Subtrair a taxa do saldo da conta do remetente e incrementar a falta do remetente. Se não houver saldo suficiente para gastar, devolva um erro.

Inicialize GAS = STARTGAS e tire uma certa quantidade de gas por byte para pagar os bytes na transação.

Transfira o valor da transação da conta do remetente para a conta de recebimento. Se a conta de recebimento ainda não existir, crie-a. Se a conta de recepção for um contrato, execute o código do contrato até a conclusão ou até a execução ficar sem gas.

Se a transferência de valor falhou porque o remetente não tinha dinheiro suficiente ou a execução do código ficou sem gas, reverte todas as mudanças de estado, exceto o pagamento das taxas, e adicione as taxas à conta do minerador.

Caso contrário, reembolsar os honorários por todo o gas restante ao remetente e enviar as taxas pagas pelo gas consumido para o minerador.

Por exemplo, suponha que o código do contrato seja:

if !self.storage[calldataload(0)]:

self.storage[calldataload(0)] = calldataload(32)

Observe que o código do contrato está escrito no código EVM de baixo nível; este exemplo está escrito em Serpent.

Suponha que o armazenamento do contrato seja iniciado em vazio e uma transação seja enviada com 10 valores de éter, 2000 GAS, 0,001 GASPRICE e 64 bytes de dados, com bytes 0-31 representando o número 2 e bytes 32-63 representando a string CHARLIE. O processo para a função de transição de estado neste caso é o seguinte:

1. Verifique se a transação é válida e bem formada.
2. Verifique se o remetente da transação tem pelo menos 2000 \* 0.001 = 2 éter. Se o fizerem, subtrair 2 éter da conta do remetente.
3. Inicializar gas = 2000; assumindo que a transação é de 170 bytes e a taxa de bytes é de 5, subtrai 850 para que haja mais 1150 gas.
4. Subtrair 10 mais éter da conta do remetente e adicioná-lo à conta do contrato.
5. Execute o código. Neste caso, isso é simples: ele verifica se o armazenamento do contrato no índice 2 é usado, percebe que não é, e assim ele define o armazenamento no índice 2 para o valor CHARLIE. Suponha que isso custe 187 gas, então a quantidade restante de gas é 1150 - 187 = 963.
6. Adicione 963 \* 0.001 = 0.963 éter de volta à conta do remetente e devolva o estado resultante.

Se não houvesse um contrato no final do recebimento da transação, a taxa de transação total seria simplesmente igual ao GASPRICE fornecido multiplicado pelo comprimento da transação em bytes, e os dados enviados ao lado da transação seriam irrelevantes.

Observe que as mensagens funcionam de forma equivalente às transações em termos de reverts: se a execução de uma mensagem ficar sem gas, a execução dessa mensagem e todas as outras execuções desencadeadas por essa execução, reverterão, mas as execuções dos pais não precisam reverter. Isso significa que é seguro para um contrato chamar outro contrato, como se A chama B com gas de G.

# Execução de Código

O código em contratos Ethereum está escrito em uma linguagem bytecode baseada em pilha de baixo nível, denominada "código de máquina virtual Ethereum" ou "código EVM".

O código consiste em uma série de bytes, onde cada byte representa uma operação. Em geral, a execução do código é um ciclo infinito que consiste em executar repetidamente a operação no contador do programa atual (que começa em zero) e, em seguida, incrementar o contador do programa por um, até chegar ao fim do código ou um erro ou STOP ou a instrução RETURN é detectada.

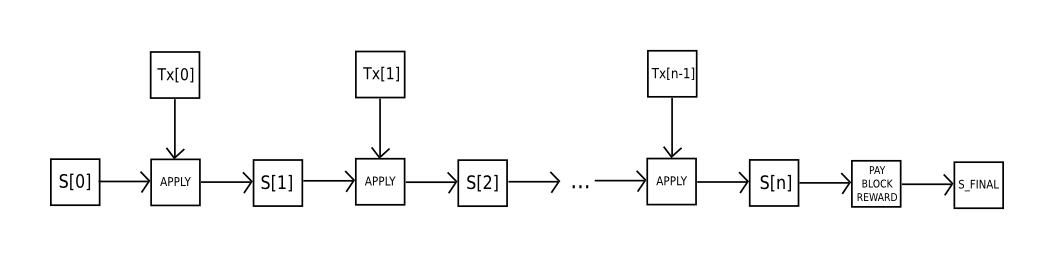
O código também pode acessar o valor, o remetente e os dados da mensagem recebida, bem como os dados do cabeçalho do bloco, e o código também pode retornar uma matriz de dados de bytes como uma saída.

O modelo de execução formal do código EVM é surpreendentemente simples. Enquanto a máquina virtual Ethereum está sendo executada, seu estado computacional completo pode ser definido por:

1. Block\_state
2. Transação
3. Mensagem
4. Código
5. Memória
6. Gas

Onde block\_state é o estado global que contém todas as contas e inclui saldos e armazenamento.

# Blockchain e Mineração



A Blockchain Ethereum é, de muitas maneiras, semelhante à Blockchain do bitcoin, embora tenha algumas diferenças. A principal diferença entre Ethereum e Bitcoin em relação à arquitetura blockchain é que, ao contrário do Bitcoin, os blocos Ethereum contêm uma cópia da lista de transações e do estado mais recente. Além disso, dois outros valores, o número do bloco e a dificuldade, também são armazenados no bloco. O algoritmo de validação de blocos em Ethereum é o seguinte:

1. Verifique se o bloco anterior referenciado existe e é válido.
2. Verifique se o timestamp do bloco é maior do que o bloco referenciado anterior e menos de 15 minutos para o futuro
3. Verifique se o número do bloco, a dificuldade, a raiz da transação e o limite de gás
4. Verifique se a prova de trabalho no bloco é válida.
5. Deixe S [0] ser o estado no final do bloco anterior.
6. Deixe TX ser a lista de transações do bloco, com n transações. Para todos i em 0 ... n-1, defina S [i + 1] = APPLY (S [i], TX [i]). Se algum aplicativo retornar um erro, ou se o gas total consumido no bloco até este ponto exceder o GASLIMIT, retornar um erro.
7. Deixe S\_FINAL ser S [n], mas adicionando a recompensa de bloco paga ao minerador.
8. Verifique se a raiz da árvore Merkle do estado S\_FINAL é igual à raiz de estado final fornecida no cabeçalho do bloco. Se for, o bloco é válido; caso contrário, não é válido.

A abordagem pode parecer altamente ineficiente à primeira vista, porque precisa armazenar todo o estado com cada bloco, mas, na realidade, a eficiência deve ser comparável à do Bitcoin.

A razão é que o estado é armazenado na estrutura da árvore e, após cada bloco, apenas uma pequena parte da árvore precisa ser alterada. Assim, em geral, entre dois blocos adjacentes, a grande maioria da árvore deve ser a mesma e, portanto, os dados podem ser armazenados uma vez e referenciados duas vezes usando ponteiros (ie, hashes of subtrees). Um tipo especial de árvore conhecida como "árvore de Patricia" é usada para realizar isso, incluindo uma modificação no conceito da árvore de Merkle que permite que os nós sejam inseridos e excluídos e não apenas mudados de forma eficiente. Além disso, porque todas as informações do estado fazem parte do último bloco, não é necessário armazenar todo o histórico da cadeia de blocos - uma estratégia que, se pudesse ser aplicada ao Bitcoin, pode ser calculada para fornecer economias 5-20x no espaço.

Uma pergunta comum é o "onde" o código do contrato é executado, em termos de hardware físico. Isso tem uma resposta: o processo de execução do código do contrato faz parte da definição da função de transição do estado, que faz parte do algoritmo de validação do bloco, então, se uma transação for adicionada ao bloco B, a execução do código gerada por essa transação será executada por todos os nós, agora e no futuro, que validam o bloco B.

# Aplicações

Em geral, existem três tipos de aplicações no Ethereum.

A primeira categoria é aplicações **financeiras**, proporcionando aos usuários formas mais poderosas de gerenciar e contratar contratos usando seu dinheiro. Isso inclui sub-moedas, derivativos financeiros, contratos de hedge, carteiras de economia, testamentos e, finalmente, até mesmo algumas classes de contratos de trabalho em grande escala. A segunda categoria é aplicações **semi-financeiras**, onde o dinheiro está envolvido, mas também há um lado não monetário para o que está sendo feito; um exemplo são as recompensas por soluções para problemas computacionais. Finalmente, existem aplicações como votação on-line e governança descentralizada **que não são financeiras**.

# Token Systems

Os sistemas de token em Blockchain têm muitas aplicações que vão desde sub-moedas que representam ativos como USD ou ouro para ações da empresa, tokens individuais representando propriedade inteligente, apólices de seguros e sistemas de token sem vínculo com o valor convencional, usado como Programas de Fidelidade.

Os sistemas Token são surpreendentemente fáceis de implementar no Ethereum. O ponto chave para entender é que uma moeda, ou sistema de token, é apenas um banco de dados com uma operação: subtrair unidades X de A e dar unidades X para B, com a condição de que (i) A tenha pelo menos unidades X antes da transação E (ii) a transação é aprovada por A. Tudo o que é necessário para implementar um sistema de token é implementar essa lógica em um contrato.

O código básico para implementar um sistema token na Serpent é o seguinte:

def send(to, value):

if self.storage[msg.sender] >= value:

self.storage[msg.sender] = self.storage[msg.sender] - value

self.storage[to] = self.storage[to] + value

**Conversa: MyToken**

# Derivados financeiros e moedas de valor estável

Os derivativos financeiros são as aplicações mais comuns de um "contrato inteligente", e um dos mais simples de implementar no código.

O principal desafio na implementação de contratos financeiros é que a maioria deles requer referência a um preço externo; por exemplo, uma aplicação muito desejável é um contrato inteligente que se cobre contra a volatilidade do éter (ou outra criptografia) em relação ao dólar dos EUA, mas isso requer o contrato para saber qual é o valor do ETH / USD.

A maneira mais simples de fazer isso é através de um contrato de "feed de dados" mantido por uma parte específica (por exemplo, NASDAQ) projetada para que essa parte tenha a capacidade de atualizar o contrato conforme necessário e fornecer uma interface que permita que outros contratos enviem uma mensagem para esse contrato e obtenha uma resposta que forneça o preço.

Como ficaria este contrato?

1. Aguarde A para introduzir 1000 éter.
2. Aguarde a B para introduzir 1000 éter.
3. Registre o valor USD de 1000 éter, calculado consultando o contrato de feed de dados, em armazenamento, digamos que isso é $ x.
4. Após 30 dias, permitir que A ou B "reativem" o contrato para enviar o valor de $ x do éter (calculado consultando novamente o contrato de feed de dados para obter o novo preço) para A e o resto para B.

Um dos principais problemas citados sobre criptomoedas é a volatilidade. Embora muitos usuários e comerciantes possam querer a segurança e a conveniência de lidar com ativos criptográficos, eles podem não querer enfrentar essa perspectiva de perder 23% do valor de seus fundos em um único dia.

Na prática, os emissores (oráculos) nem sempre são confiáveis e, em alguns casos, a infraestrutura bancária é muito fraca ou muito hostil para que esses serviços existam. Derivados financeiros fornecem uma alternativa. Aqui, em vez de um único emissor que fornece os fundos para respaldar um ativo, um mercado descentralizado de especuladores, apostando que o preço de um recurso de referência criptográfica (por exemplo, ETH) vai subir, desempenha esse papel. Ao contrário dos emissores, os especuladores não têm opção de inadimplência em seu lado da barganha porque o contrato de hedge mantém seus fundos em custódia. Note-se que esta abordagem não é totalmente descentralizada, porque uma fonte confiável ainda é necessária para fornecer o ticket de preços, embora, sem dúvida, mesmo assim, esta é uma melhoria enorme em termos de redução de requisitos de infraestrutura (ao contrário de ser um emissor, a emissão de uma alimentação de preço não requer licenças e provavelmente pode ser categorizado como open source) e reduzindo o potencial de fraude.

**Conversa: Oráculos, Open Source**

# Sistemas de reputação e identidade

A primeira cryptocurrency alternativa, a Namecoin, tentou usar uma cadeia de blocos tipo Bitcoin para fornecer um sistema de registro de nomes, onde os usuários podem registrar seus nomes em uma base de dados pública, juntamente com outros dados.

O principal caso de uso citado é para um sistema DNS, mapeando nomes de domínio como "bitcoin.org" (ou, no caso de Namecoin, "bitcoin.bit") para um endereço IP. Outros casos de uso incluem autenticação de e-mail e sistemas de reputação potencialmente mais avançados. Aqui está o contrato básico para fornecer um sistema de registro de nome com Namecoin em Ethereum:

def register(name, value):

if !self.storage[name]:

self.storage[name] = value

O contrato é muito simples; tudo isso é um banco de dados dentro da rede Ethereum que pode ser adicionado, mas não modificado ou removido. Qualquer um pode registrar um nome com algum valor, e esse registro então fica para sempre.

# Moeda e Emissão

A rede Ethereum inclui sua própria moeda incorporada, ether, que tem o duplo propósito de fornecer uma camada primária de liquidez para permitir uma troca eficiente entre vários tipos de ativos digitais e, mais importante, fornece um mecanismo para pagar taxas de transação.

* 1: wei
* 1012: szabo
* 1015: finney
* 1018: ether

Isso deve ser tomado como uma versão expandida do conceito de "dólares" e "centavos" ou "BTC" e "satoshi". No futuro próximo, esperamos que o "ether" seja usado para transações comuns, "finney" para microtransações e "szabo" e "wei" para discussões técnicas sobre taxas e implementação de protocolos; As denominações restantes podem tornar-se úteis mais tarde e não devem ser incluídas nos clientes neste momento.

O modelo de emissão será o seguinte:

1. A Ether será lançada em uma venda de moeda no preço de 1000-2000 éter por BTC, um mecanismo destinado a financiar a organização Ethereum e pagar pelo desenvolvimento que tem sido utilizado com sucesso por outras plataformas, como a Mastercoin e a NXT. Os compradores anteriores beneficiarão de descontos maiores. O BTC recebido da venda será usado inteiramente para pagar salários e recompensas aos desenvolvedores e investido em vários projetos com fins lucrativos e sem fins lucrativos no ecossistema Ethereum e cryptocurrency.
2. 0,099x do montante total vendido (60.102.216 ETH) será alocado para a organização para compensar os contribuintes antecipados e pagar as despesas denominadas em ETH antes do bloco gênese.
3. 0,099x do valor total vendido será mantido como uma reserva de longo prazo.
4. 0,26x do valor total vendido será atribuído aos mineradores por ano

**Conversa: Proof of Stake**

# Centralização de Mineração

O algoritmo de mineração Bitcoin funciona fazendo com que os mineradores computem SHA256 em versões ligeiramente modificadas do cabeçalho do bloco milhões de vezes uma e outra vez, até que eventualmente um nó apareça com uma versão cujo hash é menor do que o alvo (atualmente em torno de 2192). No entanto, esse algoritmo de mineração é vulnerável a duas formas de centralização.

Primeiro, o ecossistema de mineração passou a ser dominado por ASICs (circuitos integrados específicos da aplicação), chips de computador projetados e, portanto, milhares de vezes mais eficientes na tarefa específica da mineração Bitcoin. Isso significa que a mineração de Bitcoin não é mais uma busca altamente descentralizada e igualitária, exigindo que milhões de dólares de capital efetivamente participem.

Em segundo lugar, a maioria dos mineradores de Bitcoin na verdade não realiza validação localmente; em vez disso, eles contam com um pool de mineração centralizado para fornecer os cabeçalhos dos blocos. Este problema é indiscutivelmente pior: os três principais pools de mineração controlam indiretamente cerca de 50% do poder de processamento na rede Bitcoin, embora isso seja atenuado pelo fato de que os mineradores podem mudar para outros pools de mineração se um pool ou a coalizão tenta um ataque de 51%.

A intenção atual no Ethereum é usar um algoritmo de mineração onde os mineradores são obrigados a buscar dados aleatórios do estado, calcular algumas transações selecionadas aleatoriamente dos últimos N blocos na cadeia de blocos e retornar o hash do resultado. Isso tem dois benefícios importantes.

Primeiro, os contratos Ethereum podem incluir qualquer tipo de computação, de modo que um ASIC Ethereum seria essencialmente um ASIC para o cálculo geral - ou seja. Uma CPU melhor.

Em segundo lugar, a mineração requer acesso a toda a Blockchain, forçando os mineradores a armazenar toda a Blockchain e, pelo menos, ser capaz de verificar todas as transações. Isso remove a necessidade de pools de mineração centralizados.

# Conclusões

O protocolo Ethereum foi originalmente concebido como uma versão atualizada de uma cryptocurrency, fornecendo recursos avançados, como garantia de Blockchain, limites de retirada, contratos financeiros, mercados de apostas e similares através de uma linguagem de programação altamente generalizada.

O protocolo Ethereum não "suporta" nenhum dos aplicativos diretamente, mas a existência de uma linguagem de programação completa de Turing significa que os contratos arbitrários podem teoricamente ser criados para qualquer tipo de transação ou aplicativo. O que é mais interessante sobre Ethereum, no entanto, é que o protocolo Ethereum se move muito além de apenas moeda.

Os protocolos em torno de armazenamento de arquivos descentralizados, computação descentralizada e mercados de predição descentralizada, entre dezenas de outros conceitos, têm o potencial de aumentar substancialmente a eficiência da indústria computacional e fornecer um impulso maciço a outros protocolos peer-to-peer.

O conceito de uma função de transição de estado arbitrária, conforme implementado pelo protocolo Ethereum, prevê uma plataforma com potencial exclusivo; Em vez de ser um protocolo fechado e de propósito único destinado a uma série específica de aplicativos em armazenamento de dados, jogos de azar ou finanças.

Camada para um número muito grande de protocolos financeiros e não financeiros nos próximos anos.