[Chinese Simplified] Ethereum 白皮书中文版

Jump to bottom

James Ray edited this page on 4 Mar 2019 · 11 revisions

一个下一代智能合约和去中心化应用平台

> 如想了解更多内容，请关注微信公众号“Corey区块链技术分享”。

2009年中本聪发明的比特币经常被誉为金钱和货币的激进发展，作为数字资产的第一个例子，它同时没有人支持或者内在价值，也没有中心化的发行人或控制人。然而，可以说更重要的比特币实验的另一部分是作为分布式共识工具的底层区块链技术，并且人们的注意力正在迅速开始转向比特币的这个方面。区块链技术的常用可供选择的应用包括使用区块链数字资产来表示自定义货币和金融工具（彩色硬币），底层物理设备的所有权（智能财产），不可替代的资产，如域名（域名币），以及更复杂的应用程序，涉及让数字资产由实现任意规则的代码（智能合约）或甚至基于区块链的去中心化自治组织（DAO）直接控制。以太坊打算提供的是一个内置完全成熟的图灵完整编程语言的区块链，可用于创建“合约”，“合约”可用于编码任意状态转换函数，允许用户创建上面提到的任何系统，以及我们还没有想到的许多其他东西，只需通过在几行代码中编写逻辑。

Contents

比特币和现有概念的介绍

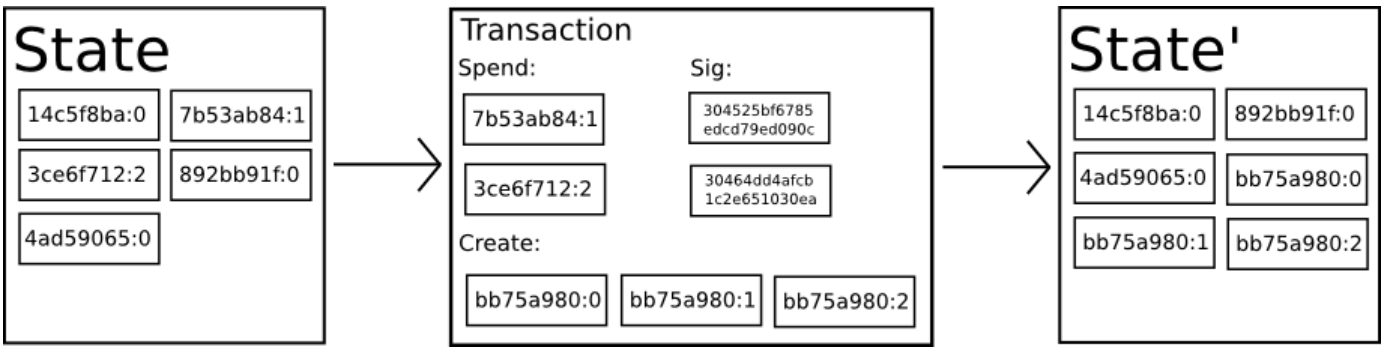
历史

去中心化数字货币以及财产登记等替代应用程序的概念已存在数十年。20世纪80年代和90年代的匿名电子现金协议，主要依赖于称为Chaumian blinding的加密原语，提供了一种具有高度隐私的货币，但协议基本上未能获得欢迎，因为它们依赖于中心化的中介。1998年，Wei Dai的b-money成为第一个通过解决计算难题和去中心化共识来引入创造货币的想法的提案，但该提案很少有关于如何实际实现去中心化的共识的细节。2005年，Hal Finney介绍了一个可重复使用的工作量证明的概念，这个系统使用来自b-money的想法和Adam Back的计算难度很大的Hashcash谜题来创建加密货币的概念，但再次由于依赖于作为后端的可信计算而达不到理想的效果。2009年，中本聪首次在实践中实现了去中心化货币，将通过公钥密码学管理所有权的既定原语与用于跟踪谁拥有币的共识算法相结合，称为“工作量证明”。

工作量证明背后的机制是这个领域的突破，因为它同时解决了两个问题。首先，它提供了一种简单且适度有效的共识算法，允许网络中的节点集体就比特币分类账状态的一组规范更新达成一致。其次，它提供了一种机制，允许自由进入共识过程，解决决定谁影响共识的政治问题，同时防止sybil攻击。它通过把参与的正式障碍，例如要求在特定清单上注册为唯一实体，替换为经济障碍-共识投票过程中单个节点的权重与节点具有的计算能力成正比。从那时起，一种替代方法被提出了，称为股权证明，计算节点的权重与其货币持有量成正比，而不是计算资源;关于这两种方法的相对优点的讨论超出了本文的范围，但应该注意的是，这两种方法都可以用作加密货币的支柱。

这是一篇来自以太坊创始人Vitalik Buterin的博客文章，Ethereum pre-history。这是另一篇讲述了更多历史的博客文章。

作为一种状态转移系统的比特币



从技术角度来看，比特币等加密货币的账本可以被认为是一个状态转换系统，其中存在一个“状态”，包括所有现有比特币的所有权状态和一个“状态转换功能”，这个“状态转换功能”以一个状态和交易为输入，输出一个新的状态作为结果。例如，在标准银行系统中，状态就是一张资产负债表，一笔交易是将$X金额的货币从账户A移动到账户B的请求，状态转换函数将A账户中的值减少$X并将B账户的值增加$X。首先如果A的帐户余额少于$ X，则状态转换函数会返回错误。因此，人们可以正式定义：

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

在上面定义的银行系统中：

```
APPLY({ Alice: $50, Bob: $50 },"send $20 from Alice to Bob") = { Alice: $30,
Bob: $70 }
```

但是：

```
APPLY({ Alice: $50, Bob: $50 },"send $70 from Alice to Bob") = ERROR
```

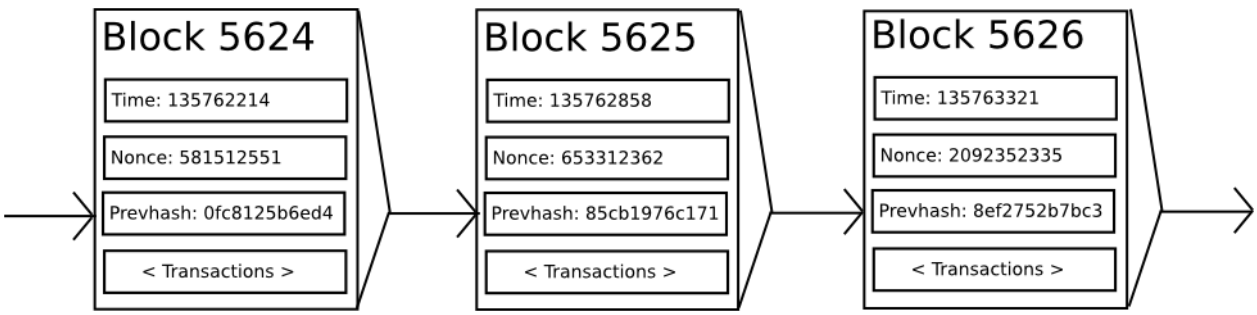
比特币中的“状态”指的是已经被挖矿但尚未花费的所有币（技术上，“未使用的交易输出”或UTXO）的集合，每个UTXO具有面额和所有者（由一个本质上是一个加密公钥的20字节地址定义 fn. 1）。一笔交易包含一个或多个输入，每个输入包含对现有UTXO的引用和由与所有者地址关联的私钥生成的加密签名，每笔交易还包含一个或多个输出，每个输出包含一个要添加到状态的新UTXO。

状态转换函数 `APPLY(S,TX) -> S'` 可以大致定义如下：

1. 对于 `TX` 中的每个输入：
    - 如果引用的UTXO不在 `S` 中，则返回错误。
    - 如果提供的签名与UTXO的所有者不匹配，则返回错误。
2. 如果所有输入UTXO的面额之和小于所有输出UTXO的面额之和，则返回错误。
3. 返回 `S'`，删除所有输入的UTXO并添加所有输出的UTXO。

第一步的前半部分阻止交易发送者花费不存在的币，第一步的后半部分阻止交易发送者花费其他人的币，第二步强制价值守恒。为了将其用于支付，协议如下。假设Alice想要向Bob发送11.7 BTC。首先，Alice将寻找一些她拥有的可用UTXO，总计至少达到11.7 BTC。实际上，Alice将无法获得11.7 BTC;假设她能得到的最小的是6 + 4 + 2 = 12。然后，她使用这三个输入和两个输出创建一笔交易。第一个输出将是11.7 BTC，Bob的地址作为其所有者，第二个输出将是剩余的0.3 BTC“找零”，所有者是Alice自己。

挖矿



如果我们能够访问值得信赖的中心化服务，那么这个系统的实现将是微不足道的;它可以简单地完全按照描述编写代码，使用中心化服务器的硬盘驱动器来跟踪状态。然而，对于比特币，我们正在尝试建立一个去中心化的货币系统，因此我们需要将状态转移系统与一个共识系统相结合，以确保每个人都同意交易订单。比特币的去中心化共识过程要求网络中的节点不断尝试生成被称为“块”的交易包。该网络旨在每十分钟生成大约一个块，每个块包含一个时间戳，一个随机数，对前一个块的引用（例如，Hash值）以及自上一个块以来发生的所有交易的列表。随着时间的推移，这会创建一个持续的，不断增长的“区块链”，不断更新以代表比特币账本的最新状态。

在该范例中表示的用于检查块是否有效的算法如下：

1. 检查块引用的前一个块是否存在且是否有效。

2. 检查块的时间戳是否大于前一个块的时间戳[fn. 2]，以及在未来两小时之内。
3. 检查块上的工作量证明是否有效。
4. 把 `S[0]` 设置为前一个块末尾的状态。
5. 假设 `TX` 是具有 `n` 个交易的块的交易列表。对于 `0...n-1` 中的所有 `i`，设置 `S[i+1] = APPLY(S[i],TX[i])`。如果任何应用程序返回错误，则退出并返回false。
6. 返回true，并将 `S[n]` 注册为该块末尾的状态。

本质上，块中的每个交易必须提供从执行交易之前的规范状态到某个新状态的有效状态转换。请注意，状态不以任何方式编码在块中;它纯粹是一个被验证节点记住的抽象概念，只能通过从genesis（创世区块）状态开始并按顺序应用每个块中的每笔交易来（安全地）计算任意块。另外，请注意矿工将交易纳入区块的顺序很重要;如果一个块中有两笔交易A和B，使得B需要花费一个由A创建的UTXO，那么如果A发生在B之前，则该块将是有效的，否则无效。

上述列表中存在的其他系统中没有的一个有效性条件是“工作量证明”的要求。精确的条件是每个区块的double-SHA256散列值（被视为256位的二进制数）必须小于动态调整的目标，截至本文撰写时，该目标大约为2^{187}。这样做的目的是使区块创建在计算上有“难度”，从而防止sybil攻击者重建对他们有利的整个区块链。因为SHA256被设计为一个完全不可预测的伪随机函数，所以创建一个有效区块的唯一方法就是试错，重复递增nonce随机数并查看新的hash是否匹配。

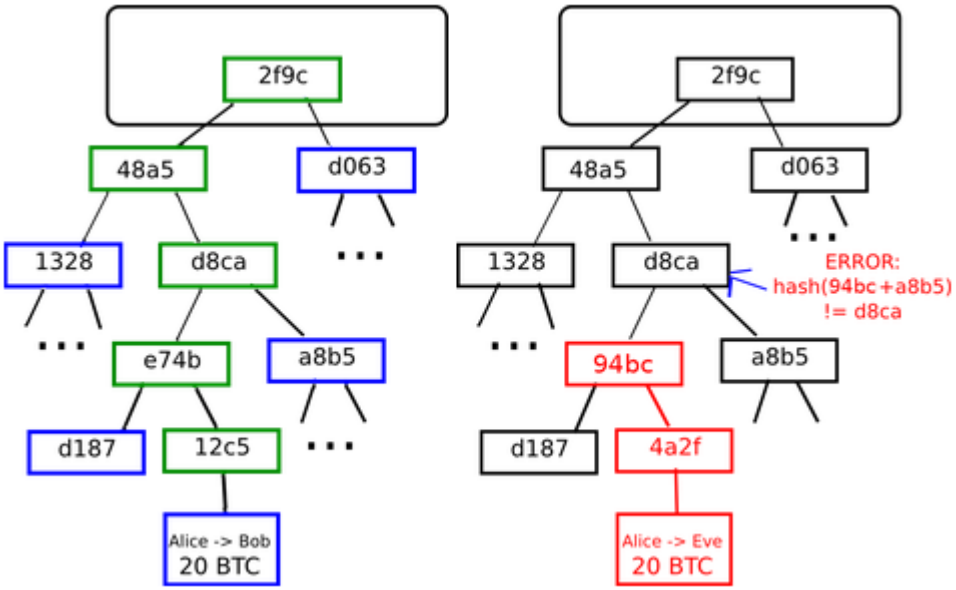
在当前目标 ~2^{187}，网络必须在找到有效区块之前平均进行约 ~2^{69}次尝试;通常，每2016个区块由网络重新校准目标，因此平均每十分钟由网络中的某个节点产生一个新块。为了补偿矿工的这项计算工作，每个区块的矿工都有权包括一个不知从哪里获得12.5BTC的交易。此外，如果任何交易在其输入中的总面额高于其输出中的总面额，则差值也将作为“交易费”给矿工。顺便说一下，这也是BTC发行的唯一机制;创世区块根本没有币。

为了更好地理解挖矿的目的，让我们看看在一个恶意攻击者的事件中会发生什么。由于比特币的底层加密被认为是安全的，因此攻击者将把未直接受加密保护的比特币系统的一部分作为目标：交易顺序。攻击者的策略很简单：

1. 将100 BTC发送给商家以换取某些产品（最好是快速交付的数字商品）
2. 等待产品交付
3. 制作另一笔交易，将相同的100 BTC发送给自己
4. 试着说服网络，发送给他自己的交易是第一笔。

一旦步骤（1）发生，几分钟后，某个矿工将把交易包含在一个区块中，比如区块编号是270.大约一个小时后，在该区块之后将再添加五个区块，每个区块间接指向这笔交易并因此“确认”了它。此时，商家将接受最终确定的付款并交付产品;因为我们假设这是一种数字商品，所以交货是即时的。现在，攻击者创建了另一个将100 BTC发送给自己的交易。如果攻击者只是将其释放到野外，则这笔交易不会被处理;矿工将尝试运行 `APPLY(S,TX)` 并注意到 `TX` 消耗的是不再处于该状态的UTXO。因此，攻击者创建了一个区块链的“分叉”，开始挖掘另一个版本的区块270，该区块指向相同的父区块269，但用新交易代替旧交易。由于区块数据不同，这需要重做工作量证明。此外，攻击者的新版本的区块270具有不同的hash，因此原来的区块271至275没有“指向”它;因此，原始链和攻击者的新链是完全分开的。规则是，在一个分叉中，最长的区块链被认为是事实，因此合法的矿工将在275链上工作，而攻击者单独在270链上工作。攻击者为了让他的区块链最长，他需要拥有比网络剩余部分组合起来的计算能力更多的计算能力才能赶上（因此，“51%攻击”）。

Merkle树



左边：只需在Merkle树中显示少量节点就可以提供分支的有效性证明。

右边：任何改变Merkle树的任何部分的尝试最终都会导致链上某处的不一致。

比特币的一个重要的可扩展性特征是区块存储在多级数据结构中。区块的“hash”实际上只是区块头的hash，大约200字节的数据包含时间戳，随机数，上一个区块的hash和称为Merkle树的数据结构的根hash，Merkle树把所有交易存储在区块上。 Merkle树是一种二叉树，由一组节点组成，在树的底部有大量包含底层数据的叶子节点，以及一组中间节点，每个中间节点是两个子节点的hash，最后是一个根节点，也是由两个子节点的hash形成的，代表树的“顶部”。 Merkle树的目的是允许块中的数据逐个传递：节点只能从一个来源下载一个区块的头部，从另一个来源下载与它们相关的树的一小部分，并且仍然可以保证所有数据都是正确的。这样做的原因是哈希向上传播：如果一个恶意用户试图将假交易交换到Merkle树的底部，这种改变将导致上面的节点发生变化，然后再上面的节点也发生变化，最后改变树的根，从而改变区块的hash，使协议将其注册为完全不同的区块（几乎可以肯定是有无效的工作量证明）。

Merkle树协议对于长期可持续性而言至关重要。比特币网络中的“完整节点”，即存储和处理每个区块的节点，截至2014年4月在比特币网络中占用大约15 GB的磁盘空间，并且每月增长超过1GB字节。目前，这对于某些台式计算机而不是手机是可行的，并且稍后将来只有企业和业余爱好者才能参与。称为“简化支付验证”（SPV）的协议允许存在另一类节点，称为“轻节点”，其下载区块头，在区块头上验证工作量证明，然后仅下载与他们相关的交易相关联的“分支”。这使得轻型节点能够以强有力的安全保证来确定任何比特币交易的状态及其当前的余额，同时仅下载整个区块链的一小部分。

替代的区块链应用程序

采用底层区块链理念并将其应用于其他概念的想法也有很长的历史。1998年，Nick Szabo提出了拥有所有权的安全财产权概念，该文件描述了“复制数据库技术的新进展”将如何允许基于区块链的系统存储谁拥有土地的注册表，创建了精心设计的框架，包括田产，逆权侵占和格鲁吉亚土地税等概念。然而，遗憾的是，当时没有可用的有效复制数据库系统，因此该协议从未在实践中实施。然而，在2009年之后，一旦比特币的去中心化共识得以发展，许多替代应用程序迅速开始出现。

- **Namecoin** - 在2010年创建，Namecoin最好被描述为去中心化名称注册数据库。在像Tor，Bitcoin和BitMessage这样的去中心化协议中，需要有一些识别帐户的方法，以便其他人可以与它们进行交互，但在所有现有解决方案中，唯一可用的标识符是伪随机哈希，如 `1LW79wp5ZBqaHW1jL5TCiBCrhQYtHagUWy` 。理想情况下，人们希望能够拥有一个名为比如“george”的帐户。但问题是，如果一个人可以创建一个名为“george”的帐户，那么其他人也可以使用相同的流程为自己注册“george”并冒充他们。唯一的解决方案是先申请制范例，第一个注册者成功，第二个注册失败 - 这个问题非常适合比特币共识协议.Namecoin是使用这种想法的最古老，最成功的名称注册系统实现。
- **Colored coins** - 彩色硬币的目的是作为一种协议，允许人们创建自己的数字货币 - 或者，以一个单位货币的重要例子，数字令牌。在彩色硬币协议中，通过公开为特定比特币UTXO分配颜色来“发布”新货币，并且协议递归地将其他UTXO的颜色定义为与创建它们的交易的输入的颜色相同。 （一些特殊规则适用于混合颜色输入的场景）。这允许用户维护仅包含特定颜色的UTXO的钱包，并像普通比特币一样发送它们，通过区块链回溯以确定他们收到的任何UTXO的颜色。
- **Metacoins** - metacoin背后的想法是拥有一个存在于比特币之上的协议，使用比特币交易来存储metacoin交易，但具有不同的状态转换功能， `APPLY'` 。由于metacoin协议无法阻止无效的metacoin交易出现在比特币区块链中，因此添加了一条规则，即如果 `APPLY'(S,TX)` 返回错误，则协议默认为 `APPLY'(S,TX) = S` .这提供了用于创建任意加密货币协议的简单机制，可能具有无法在比特币内部实现的高级功能，但由于挖矿和网络的复杂性已经由比特币协议处理，因此开发成本非常低。 Metacoins已被用于实现某些类别的金融合同，名称登记和去中心化交换。

因此，一般而言，建立共识协议有两种方法：构建独立网络，或者在比特币之上构建协议。前一种方法虽然在像Namecoin这样的应用程序中取得了相当的成功，但很难实现;每种单独的实现都需要引导独立的区块链，以及构建和测试所有必要的状态转换和网络代码。此外，我们预测去中心化共识技术的应用集将遵循幂律分布，其中绝大多数应用程序太小而无法保证自己的区块链，并且我们注意到存在大类去中心化应用，特别是去中心化自治组织，需要相互交流。

另一方面，基于比特币的方法存在缺陷，即它不会继承比特币的简化支付验证功能。 SPV适用于比特币，因为它可以使用区块链深度作为有效性的代理;在某些时候，一旦交易的祖先回到足够远的地方，就可以说它们是合法的状态的一部分。另一方面，基于区块链的元协议不能强制区块链不包括在其自己的协议的上下文中无效的事务。因此，完全安全的SPV元协议实现需要一直向后扫描到比特币区块链的开头，以确定特定交易是否有效。目前，基于比特币的元协议的所有“轻量级”实现依赖于可信服务器来提供数据，可以说是非常不理想的结果，尤其是当加密货币的主要目的之一是消除对信任的需求时。

脚本

即使没有任何扩展，比特币协议实际上确实促进了“智能合约”概念的弱版本。比特币中的UTXO不仅可以由公钥拥有，还可以由以简单的基于堆栈的编程语言表示的更复杂的脚本拥有。在这个范例中，花费UTXO的脚本必须提供满足脚本的数据。实际上，即使是基本的公钥所有权机制也是通过脚本实现的：脚本将椭圆曲线签名作为输入，根据交易和拥有UTXO的地址进行验证，如果验证成功则返回1，否则返回0。其他更复杂的脚本存在于各种其他使用场景中。例如，可以构建一个脚本，该脚本需要来自给定三个私钥中的两个的签名来验证（“multisig”），这对于公司帐户，安全储蓄帐户和一些商家托管情况是有用的设置。脚本也可以用来为计算问题的解决方案支付赏金，甚至可以构建一个脚本做这样的事“如果你能提供一个SPV证据证明你发送了这个面额的Dogecoin交易给我，那么这个比特币UTXO就是你的”，本质上允许去中心化的跨加密货币交换。

但是，比特币中实现的脚本语言有几个重要的局限性：

- **缺乏图灵完备性** - 也就是说，虽然比特币脚本语言支持大量计算，但它不支持所有计算。缺少的主要类别是循环。这样做是为了避免在交易验证期间出现无限循环;从理论上讲，它对于脚本程序员来说是一个可以克服的障碍，因为任何循环都可以通过简单地使用if语句多次重复底层代码来模拟，但它会导致脚本的空间效率非常低。例如，实现替代椭圆曲线签名算法可能需要256轮重复的乘法，每一轮乘法都单独包含在代码中。
- **价值盲区** - UTXO脚本无法对可以提取的金额进行细粒度控制。例如，一个强大的oracle合约用例将是一个对冲合约，其中A和B投入价值1000美元的BTC，30天后脚本将价值1000美元的BTC发送给A，其余的发送给B.这需要一个oracle确定1 BTC值多少美元，但即便如此，相对于现在可用的完全中心化解决方案而言，它在信任和基础设施需求方面是一个巨大的改进。然而，因为UTXO是要么全有要么全无的，所以实现这一目标的唯一方法是通过非常低效的侵入来获得不同面额的许多UTXO（例如，每k到30的一个UTXO为2^k）并且选择哪个UTXO发送到A和哪个发送到B.
- **缺乏状态** - UTXO可以被花费或未花费;多阶段合约或脚本没有机会保持任何其他内部状态。这使得很难制定多阶段期权合约，去中心化交换要约或两阶段加密承诺协议（安全计算奖励所必需的）。这也意味着UTXO只能用于构建简单的一次性合约而不是更复杂的“有状态”合约，例如去中心化组织，并且使得元协议难以实现。二元状态与价值盲区相结合也意味着另一个重要的应用，即取款限制，是不可能的。
- **区块链盲区** - UTXO对区块链数据（如随机数，时间戳和先前区块的哈希）不了解。通过剥夺脚本语言可能有价值的随机数来源，这严重限制了赌博和其他几个类别的应用。

因此，我们看到了三种在加密货币之上构建高级应用程序的方法：构建新的区块链，在比特币之上使用脚本，以及在比特币之上构建元协议。构建新的区块链允许在构建功能集时拥有无限的自由，但代价是开发时间，引导工作和安全性。使用脚本很容易实现和标准化，但其功能非常有限，而元协议虽然容易，但却存在可伸缩性方面的缺陷。通过以太坊，我们打算构建一个替代框架，在开发简单性和更强大的轻客户端属性方面提供更大的收益，同时允许应用程序共享经济环境和区块链安全性。

以太坊

以太坊的目的是创建一个替代协议来构建去中心化的应用程序，提供我们认为对于大类去中心化的应用程序非常有用的一组不同的权衡（折衷），特别强调快速开发时间，小型和很少使用的应用程序的安全性，以及不同应用程序非常有效地交互的能力很重要。以太坊通过构建本质上最终的抽象基础层来实现这一目标：具有内置图灵完备编程语言的区块链，允许任何人编写智能合约和去中心化应用程序，在这些应用程序中，他们可以创建自己的任意规则，包括所有权，交易格式和状态转换功能。 Namecoin的简单版本可以用两行代码编写，其他协议如货币和信誉系统可以在二十以下构建。智能合约，包含价值并仅在满足某些条件时解锁的加密“盒子”也可以构建在这个平台之上，其功能远远超过比特币脚本提供的功能，因为图灵完备性的附加能力，价值意识，区块链意识和状态。

哲学

以太坊背后的设计旨在遵循以下原则：

1. **简单性**: 以太坊协议应该尽可能简单，即使以某些数据存储或时间效率低下为代价。[fn. 3]理想情况下，一般程序员应该能够遵循并实现整个规范，[fn. 4]以便充分实现加密货币带来的前所未有的民主化潜力，并进一步将以太坊作为一种向所有人开放的协议的愿景。任何增加复杂性的优化除非提供了非常实质性的好处，否则不应被包括。
2. **普遍性**: 以太坊的设计理念的一个基本部分是以太坊没有“特征”。[fn. 5]相反，以太坊提供了一种内部的图灵完备脚本语言，程序员可以使用它来构建任何可以在数学上定义的智能合约或交易类型。想要发明自己的金融衍生品？有了以太坊，你可以。想制作自己的货币？将其设置为以太坊合约。想要建立一个全面的守护进程或天网？你可能需要有几千个联锁合约，并且一定要慷慨地供给它们，这样做，但有了以太坊，没有什么能阻止你。
3. **模块化**: 以太坊协议的各个部分应设计为尽可能模块化和可分离的。在开发过程中，我们的目标是创建一个程序，如果要在一个地方进行一个小的协议修改，应用程序堆栈将继续运行而无需任何进一步修改。诸如Ethash（参见黄皮书附录或维基文章），修改过的Patricia树（黄皮书，wiki）和RLP（黄皮书，wiki）等创新应该是并且作为单独的功能完整的库实现。这样即使它们在以太坊中使用，即使以太坊不需要某些功能，这些功能在其他协议中仍然可用。应该最大限度地完成以太坊的发展，以便使整个加密货币生态系统受益，而不仅仅是以太坊本身。
4. **敏捷**: 以太坊协议的细节并非一成不变。虽然我们会非常明智地对高级构造进行修改，例如使用分片路线图，抽象执行，使用体现在共识中的数据可用性。稍后在开发过程中的计算测试可能会使我们发现某些修改，例如，协议架构或以太坊虚拟机（EVM）将大大提高可扩展性或安全性。如果发现任何此类机会，我们将利用它们。

5. **无歧视**和**无审查**: 协议不应试图主动限制或阻止特定类别的使用。协议中的所有监管机制都应设计为直接监管危害，而不是试图反对特定的不受欢迎的应用。程序员甚至可以在以太坊上运行无限循环脚本，只要他们愿意持续支付每个计算步骤的交易费用。

以太坊账户

在以太坊中，状态由称为“账户”的对象组成，每个帐户具有20字节的地址，账户之间的价值和信息的直接转移的状态转换。以太坊帐户包含四个字段：

- **nonce（随机数）**，一个用于确保每个交易只能处理一次的计数器
- 帐户的当前**Ether(以太币)余额**
- 帐户的**合约代码**（如果存在）
- 帐户的**存储空间**（默认为空）

“Ether(以太币)”是以太坊的主要内部加密燃料，用于支付交易费用。通常，有两种类型的帐户：**外部拥有的帐户**，由私钥控制，以及**合约帐户**，由合约代码控制。外部拥有的帐户没有代码，可以通过创建和签署交易从外部拥有的帐户发送消息;在合约账户中，每次合约账户收到一条消息，它的代码就激活了，允许其读取和写入内部存储并发送其他消息或依次创建合同。

请注意，以太坊中的“合约”不应被视为应该“履行”或“遵守”的东西;相反，它们更像是存活在以太坊执行环境中的“自治代理”，在被一条消息或交易“戳”时始终执行特定的代码，并且可以直接控制自己的以太币余额和自己的键/值存储以跟踪持久化的变量。

消息和交易

术语“交易”在以太坊中用于表示存储了一条要从外部拥有的帐户发送的消息的签名数据包。交易包含：

- 消息的接收者
- 一个标识发送者的签名
- 从发送者发送到接收者的以太币数量
- 可选的数据字段
- `STARTGAS` 值，表示允许交易执行的最大计算步骤数
- `GASPRICE` 值，表示发送者每个计算步骤支付的费用

前三个是任何加密货币中预期的标准字段。默认情况下，数据字段没有任何功能，但虚拟机具有被合约使用于访问数据的操作码;作为示例用例，如果一个合约作为区块链域名注册服务运行，那么它可能希望将被传递给它的数据解释为包含两个“字段”，第一个字段是要注册的域，第二个字段是该域要注册到的IP地址。合约将从消息数据中读取这些值，并将它们妥善地存储起来。

STARTGAS 和 GASPRICE 字段对以太坊的反拒绝服务模式至关重要。为了防止代码中的意外或敌对无限循环或其他计算浪费，每笔交易都需要设置它可以使用的代码执行的计算步骤数量的限制。计算的基本单位是“gas”;通常，一个计算步骤花费1 gas，但是一些操作会花费更多的gas，因为它们的计算成本更高，或者增加了必须作为状态的一部分存储的数据量。交易数据中的每个字节还需要5 gas费用。费用系统的目的是要求攻击者按比例支付他们消耗的每种资源，包括计算，带宽和存储;因此，导致网络消耗更多这些资源的任何交易必须具有与增量大致成正比的gas费用。

消息

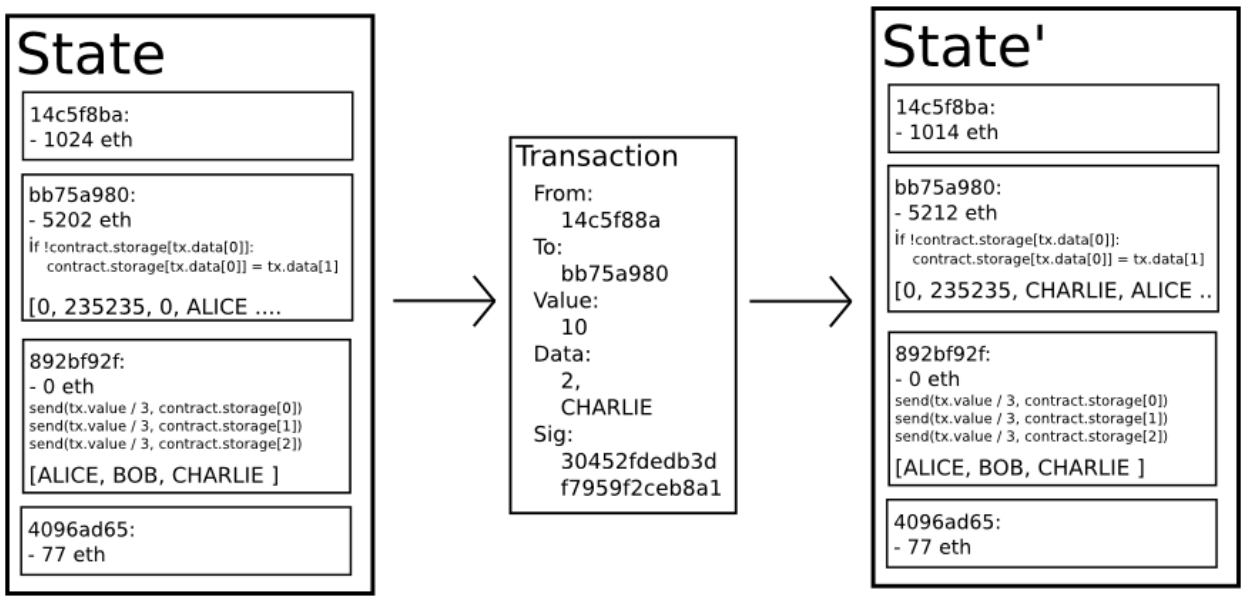
合约具有向其他合约发送“消息”的能力。消息是永远不会序列化的虚拟对象，仅存在于以太坊执行环境中。一条消息包含：

- 消息的发送者（隐含）
- 消息的接收者
- 与消息一起传输的ether数量
- 一个可选的数据字段
- 一个 STARTGAS 值

从本质上讲，消息就像一笔交易，除了它是由合约产生而不是外部账户。当正在执行代码的合约执行 CALL 操作码时会产生一条消息，该操作码生成并执行一条消息。与交易一样，消息会导致接收者帐户运行其代码。因此，合约可以以与外部账户完全相同的方式与其他合约建立关系。

请注意，交易或合约指定的gas限额适用于该交易和所有次级执行所消耗的总gas。例如，如果外部账户A向B发送了一笔1000 gas的交易，并且B在向C发送消息之前消耗了600 gas，并且C的内部执行在返回之前消耗了300 gas，那么B可以在消耗完gas之前再花费100 gas。

以太坊状态转换函数



以太坊状态转换函数 `APPLY(S,TX) -> S'` 可以定义如下：

1. 检查交易是否格式正确（例如，具有正确数量的价值），签名是否有效，以及nonce是否与发送者帐户中的nonce匹配。如果没有，返回错误。
2. 将交易费用计算为 `STARTGAS * GASPRICE` ，并从签名中确定发送地址。从发送者的帐户余额中减去交易费用并增加发送者的nonce。如果没有足够的余额支出，则返回错误。
3. 初始化 `GAS = STARTGAS` ，并在每个字节中取出一定量的gas来支付交易中的字节数。
4. 将交易价值从发送者的帐户转移到接收帐户。如果接收帐户尚不存在，就创建它。如果接收帐户是合约，则运行合约的代码，要么完成，要么直到执行消耗完gas。
5. 如果由于发送者没有足够的资金而导致价值转移失败，或者代码执行耗尽gas，则除了支付交易费用之外回滚所有状态更改，并将交易费用添加到矿工的帐户。
6. 否则，将所有剩余gas的费用退还给发送者，并将消耗的gas费用发送给矿工。

例如，假设合约的代码是：

```
if !self.storage[calldataload(0)]:
    self.storage[calldataload(0)] = calldataload(32)
```

注意，实际上合约代码是用低级EVM代码编写的;这个例子是用Serpent编写的，这是我们的高级语言之一，这是为了清楚起见，可以被编译成EVM代码。假设合约的存储开始是空的，并且发送的交易有10个ether，2000 gas（gas价格是0.001个ether）和64个字节的数据，字节0-31表示数字 `2` ，字节32-63表示字符串 `CHARLIE` 。[fn. 6]在这种情况下，状态转换功能的过程如下：

1. 检查交易是否有效且格式正确。
2. 检查交易发送者是否至少有2000 * 0.001 = 2 ether。如果是，则从发送者的帐户中减去2 ether。
3. 初始化gas= 2000;假设交易是170字节长并且字节费是5 gas，减去850后便剩下1150 gas。
4. 从发送者的帐户中再减去10 ether，并将其添加到合约的帐户中。
5. 运行代码。在这种情况下，这很简单：它检查合约在索引 `2` 处的存储是否被使用，注意到它没有被使用，因此它将索引 `2` 处的存储设置为值 `CHARLIE` 。假设这需要187 gas，因此剩余的gas为1150 - 187 = 963
6. 将963 * 0.001 = 0.963 ether添加回发送者的帐户，并返回结果状态。

如果在交易的接收端没有合约，则总交易费用将简单地等于提供的 `GASPRICE` 乘以交易的长度（以字节为单位），并且与交易一起发送的数据将是无关紧要的。

请注意，消息在交易回滚方面与交易等效地工作：如果消息执行耗尽gas，则该消息的执行以及该执行触发的所有其他执行都会回滚，但父执行不需要回滚。这意味着合约调用另一个合约是“安全的”，好像A用G gas呼叫B然后A的执行保证最多失去G gas。最后，请注意，有一个操作码 `CREATE` ，它创建了一个合约;它的执行机制通常类似于 `CALL` ，但执行的输出确定了一个新创建的合约的代码。

代码执行

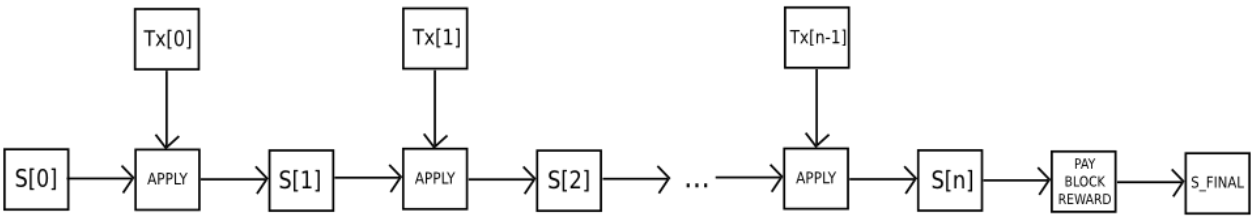
以太坊合约中的代码是用低级的，基于堆栈的字节码语言编写的，称为“以太坊虚拟机代码”或“EVM代码”。代码由一系列字节组成，其中每个字节代表一个操作。通常，代码执行是一个无限循环，包括在当前程序计数器（从零开始）重复执行操作，然后将程序计数器递增1，直到达到代码结束或错误或检测到 `STOP` 或 `RETURN` 指令。这些操作可以访问存储数据的三种类型的空间：

- **堆栈**，可以推入和弹出值的后进先出容器
- **内存**，一个无限可扩展的字节数组
- 合约的长期**存储**，一个键/值存储。与堆栈和内存不同，堆栈和内存在计算结束后重置，存储会长期持久化。

代码还可以访问传入消息的值，发送者和数据，以及区块头数据，代码也可以返回数据的字节数组作为输出。

EVM代码的正式执行模型非常简单。当以太坊虚拟机运行时，其完整的计算状态可以由元组 `(block_state, transaction, message, code, memory, stack, pc, gas)` 定义，其中 `block_state` 是包含所有帐户的全局状态，包括余额和存储。在每轮执行开始时，通过获取 `code` 的第 `pc` 个字节（或者如果 `pc >= len(code)` 则为0）找到当前指令，并且每条指令在其如何影响元组方面都有自己的定义。例如，ADD将两个元素从堆栈中弹出并推入它们的总和，将 `gas` 减少1并将 `pc` 增加1，并且 `SSTORE` 将前两个元素从堆栈中弹出并将第二个元素插入到合约存储中的第一个元素的指定索引处。尽管有许多可以通过即时编译来优化以太坊虚拟机的执行的方法，但以太坊的基本实现可以在几百行代码中完成。

区块链和挖矿



以太坊区块链在许多方面类似于比特币区块链，尽管它确实存在一些差异。关于区块链架构，以太坊和比特币之间的主要区别在于，与比特币（仅包含交易清单的副本）不同，以太坊区块包含交易清单和最新状态的副本。除此之外，区块中还存储了另外两个值，即区块编号和难度。以太坊中的基本区块验证算法如下：

1. 检查引用的前一个区块是否存在且是否有效。
2. 检查区块的时间戳是否大于引用的前一个区块的时间戳，并且在将来不到15分钟内
3. 检查区块编号，难度，交易根，叔根和gas限制（各种底层以太坊特定概念）是否有效。
4. 检查区块上的工作量证明是否有效。

5. 设 `S[0]` 为前一个区块末尾的状态。
6. 设 `TX` 为区块的交易列表，具有 `n` 笔交易。对于 `0...n-1` 中的所有 `i` ，设置 `S[i+1] = APPLY(S[i],TX[i])` 。如果任何应用程序返回错误，或者直到此时区块中消耗的总gas超过 `GASLIMIT` ，则返回错误。
7. 设 `S_FINAL` 为 `S[n]` ，但添加支付给矿工的区块奖励。
8. 检查状态 `S_FINAL` 的Merkle树根是否等于区块头中提供的最终状态根。如果是，则该区块有效;否则，它无效。

这种方法乍一看似乎非常低效，因为它需要存储每个区块的整个状态，但实际上效率应该与比特币的效率相当。原因是状态存储在树结构中，并且在每个区块之后只需要更改树的一小部分。因此，通常，在两个相邻区块之间，绝大多数树应该是相同的，因此数据可以存储一次并使用指针（即，子树的hash）引用两次。一种称为“Patricia树”的特殊树用于实现此目的，包括对Merkle树概念的修改，允许有效地插入和删除节点，而不仅仅是更改节点。此外，由于所有状态信息都是最后一个区块的一部分，因此无需存储整个区块链历史记录 - 如果可以应用于比特币，则是一种可以节省5-20倍空间的策略。

一个常见问题是在物理硬件角度执行合约代码的“位置”。这有一个简单的答案：执行合约代码的过程是状态转换函数定义的一部分，状态转换函数是区块验证算法的一部分，因此如果一笔交易被添加到区块 `B` 中，那么该交易产生的代码执行将是现在和将来，由所有节点执行，下载并验证区块 `B` .

应用

通常，在以太坊之上有三种类型的应用程序。第一类是金融应用程序，为用户提供更强大的方式来管理和使用他们的金钱合约。这包括子货币，金融衍生品，对冲合约，储蓄钱包，遗嘱，以及最终甚至一些类别的全面雇佣合约。第二类是半金融应用，其中涉及金钱，但对于正在进行的工作也存在很多的非货币方面;一个完美的例子是自我实施计算问题解决方案的赏金。最后，有一些应用和金融完全没有关系，如在线投票和去中心化治理。

令牌系统

区块链令牌系统有许多应用程序，从代表资产（如美元或黄金）的子货币到公司股票，代表智能财产的个人代币（令牌），安全的不可伪造的优惠券，甚至与传统价值无关的用作点数激励系统的代币（令牌）系统。在以太坊中，令牌系统非常容易实现。要理解的关键点是货币或令牌系统从根本上说是一个具有一个操作的数据库：从A减去X单位并将X单位赋予B，条件是（1）A在交易前至少有X个单位（2）交易由A批准。实现令牌系统所需的全部是将此逻辑实现到合约中。

在Serpent中实现令牌系统的基本代码如下：

```
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
```

这基本上是本文件中上面进一步描述的“银行系统”状态转换功能的字面实现。需要添加一些额外的代码行来提供起初分配货币单位的初始步骤以及一些其他边缘情况，理想情况下会添加一个函数以让其他合约查询地址的余额。但这就是它的全部。从理论上讲，作为子货币的基于以太坊的令牌系统可能包括基于比特币的链上元货币缺乏的另一个重要特征：直接以该货币支付交易费用的能力。在这种实现方式下，合约会维护一个ether余额，合约可以用这个ether余额来退还被用于支付交易费用的ether给发送者，并且它将通过收集收取费用并在持续的拍卖中转售它们的内部货币单位来补充这种余额。因此，用户需要用ether“激活”他们的帐户，但是一旦ether在那里它就可以重复使用，因为合同每次都会退还。

金融衍生品和稳定价值货币

金融衍生产品是“智能合约”的最常见应用，也是最简单的用代码实现的应用之一。实施金融合约的主要挑战是，其中大多数合约要求参考外部价格代码;例如，一个非常受欢迎的应用是一个智能合约，它针对ether（或另一种加密货币）相对于美元的波动性进行套期保值，但这样做需要合约知道ETH/USD的价值是多少。最简单的方法是通过由特定方（例如纳斯达克）维护的“数据供应”合约，以便该方能够根据需要更新合约，并提供一个接口以便允许其他合约发送一条消息给“数据供应”合约，并且获得提供价格的回复。

鉴于这一关键因素，对冲合约将如下所示：

1. 等待A方输入1000 ether。
2. 等待B方输入1000 ether。
3. 记录通过查询数据供应合约计算出的1000 ether对应的美元值到存储中，比如这是x美元。
4. 30天后，允许A或B“重新激活”合约，以便发送价值x美元的ether（通过再次查询数据供应合约以获得新价格计算）到A，其余为B.

这种合约在密码商业方面具有巨大潜力。加密货币的一个主要问题是它是不稳定的;虽然许多用户和商家可能希望得到处理加密资产的安全性和便利性，但他们可能不希望面临在一天内损失其资金价值23%的预期。到目前为止，最常提出的解决方案是发行人背书的资产;我们的想法是，发行人创建一种子货币，在该子货币中，他们有权发行和撤销单位，并向任何向其提供（离线）一个特定标的资产（例如黄金，美元）的人提供一个单位的货币。然后，发行人承诺向发回一个加密资产单位的任何人提供一个单位的标的资产。该机制允许任何非加密资产被“提升”到加密资产中，前提是发行者可以被信任。

然而，在实践中，发行人并不总是值得信赖，在某些情况下，银行基础设施太弱或太过敌对，无法存在此类服务。金融衍生品提供了另一种选择。在这里，押注加密参考资产（例如ETH）的价格将上涨的投资者的去中心化市场，而不是提供资金来背书一个资产的单一发行人，起到了这个作用。与发行人不同，投资者不能选择在交易方面违约，因为套期保值合约将其资金存放在第三方托管账户中。请注意，这种方法并不是完全去中心化的，因为仍然需要一个可靠的来源来提供价格代码，尽管可以说这在减少基础设施需求（与发行人不同，发行价格供应不需要许可证并且很可能被归类为言论自由）方面仍然是一个巨大的改进并减少欺诈的可能性。

身份和声誉系统

最早的替代加密货币Namecoin试图使用类似比特币的区块链来提供名称注册系统，用户可以在公共数据库中将其名称与其他数据一起注册。主要引用的用例是针对DNS系统，将域名如“bitcoin.org”（或者，在Namecoin的场景下，“bitcoin.bit”）映射到IP地址。其他用例包括电子邮件身份验证和可能更高级的信誉系统。这是在以太坊上提供类似Namecoin的名称注册系统的基本合约：

```
def register(name, value):
    if !self.storage[name]:
        self.storage[name] = value
```

合约很简单;它只是以太坊网络中的一个数据库，可以添加到，但不能修改或删除。任何人都可以注册一个具有某值的名称，然后注册永远持久化。更复杂的名称注册合约还将具有允许其他合约查询它的“函数子句”，以及由名称“所有者”（即第一注册者）用于更改数据或转让所有权的机制。甚至可以基于它添加信誉和信任网络功能。

去中心化文件存储

在过去几年中，出现了许多流行的在线文件存储初创公司，其中最突出的是Dropbox，它试图允许用户上传其硬盘的备份并让服务存储备份并允许用户以付月费的方式访问它。但是，此时文件存储市场有时效率相对较低;粗略看一下现有的各种解决方案表明，特别是在20-200 GB级别的“恐怖谷”，既没有免费配额也没有企业级折扣，主流文件存储成本的月度价格是你支付了比一个月内整个硬盘的成本更多的金钱。以太坊合约可以允许开发去中心化文件存储生态系统，其中个人用户可以通过租用他们自己的硬盘来赚取少量资金，并且可以使用未使用的空间来进一步降低文件存储的成本。

这种设备的关键支撑部分将是我们所谓的“去中心化Dropbox合约”。该合约的工作原理如下。首先，将所需数据分成块，加密每个块以保护隐私，并从中构建Merkle树。然后创建一个合约，具有这样的规则：即每N个块，合约将在Merkle树中选择一个随机索引（使用先前的块哈希，可从合约代码访问，作为随机源），并将X ether提供给第一个为交易提供简化付款验证的实体 - 就像树中特定索引处块的所有权证明一样。当用户想要重新下载他们的文件时，他们可以使用微支付通道协议（例如，每32KB支付1 szabo）来恢复文件;最节省费用的方法是让付款人直到最后才发布交易，而是在每32KB后用相同的nonce替换稍微更有利可图的交易。

该协议的一个重要特征是，虽然看起来似乎是信任许多随机节点不会决定忘记该文件，但可以通过秘密共享将文件分成多个部分，并且观察合约看每个部分仍被某个节点占有来将风险降低到接近零。如果合约仍在支付金钱，则提供加密证据，证明某人仍然在存储该文件。

去中心化自治组织

“去中心化的自治组织”的一般概念是具有某些成员或股东的虚拟实体，其可能具有67%的多数，有权花费该实体的资金并修改其代码。成员将共同决定组织应如何分配资金。分配DAO资金的方法可以从奖金，工资到更为奇特的机制，如奖励工作的内部货币。这基本上复制了传统公司或非营利组织的法律外衣，但仅使用加密区块链技术来强制执行。到目前为止，关于DAO的大部分讨论都围绕着“去中心化自治公司”（DAC）的“资本主义”模式，其中包括股息接收股东和流通股;一种替代方案，也许被称为“去中心化自治社区”，将使所有成员在决策中拥有平等的份额，并要求67%的现有成员同意增加或删除一个成员。一个人只能拥有一个会员资格的要求需要由该集团共同执行。

关于如何编写DAO的一般概述如下。最简单的设计是一段自我修改的代码，如果三分之二的成员同意变更，则会发生变化。尽管代码在理论上是不可变的，但是人们可以轻松解决这个问题，并通过在分开的合同中包含代码块来实现事实上的可变性，并将调用的合约的地址存储在可修改的存储中。在这种DAO合同的简单实现中，将有三种交易类型，由交易中提供的数据区分：

- `[0,i,K,V]` 注册具有索引 `i` 的提议，以将存储索引 `K` 处的地址更改为值 `V`。
- `[1,i]` 注册一个投票赞成提案 `i`
- `[2,i]` 如果已经进行了足够的投票，最终确定提案 `i`

然后合约将为每种交易类型都有子句。它将维护所有开放存储更改的记录，以及投票给他们的人员列表。它还有一个所有成员的清单。当任何存储更改得到三分之二的成员投票时，最终的交易可以执行更改。更复杂的架构也可以为发送交易，添加成员和删除成员等功能提供内置投票功能，甚至可以提供流动民主风格的投票授权（即任何人都可以指派某人帮他投票们，以及指派是传递性的，所以如果A指派B，B指派C，则C确定A的投票）。这种设计将允许DAO作为一个去中心化的社区有机地发展，允许人们最终委派过滤谁是专家的成员的任务，虽然不像“当前系统”专家可以随着时间的推移，随着个体社区成员改变他们的阵营，容易地出现和消失。

另一种模式是去中心化的公司，任何账户都可以拥有零股或多股，做出一个决定需要三分之二以上的股份。完整的架构将涉及资产管理功能，买卖股票报价的能力，以及接受报价的能力（最好在合约中有订单匹配机制）。代表团也将存在流动民主风格，概括了“董事会”的概念。

其他应用

1. 储蓄钱包。假设Alice希望保证她的资金安全，但担心她会丢失或有人会破解她的私钥。她将把ether放入一个与Bob和一家银行订立的合约，具体如下：

- Alice一个人每天最多可以提取1%的资金。
- Bob一个人每天最多可以提取1%的资金，但Alice有能力用她的密钥发起一笔交易来关闭这种能力。
- Alice和Bob一起可以提取任何金额的资金。

通常情况下，每天1%对于Alice就足够了，如果Alice想要提取更多资金，她可以联系Bob寻求帮助。如果Alice的密钥遭到入侵，她会跑到Bob那里将资金转移到一份新合约上。如果她失去了她的密钥，Bob最终将把资金取出来。如果Bob被证明是恶意的，那么Alice可以关闭Bob的提取能力。

2. 农作物保险。人们可以轻松地制定金融衍生品合约，但使用天气的数据供应而不是任何价格指数。如果爱荷华州的一位农民购买的衍生品根据爱荷华州的降水量反向支付，那么如果发生干旱，农民将自动获得资金，如果有足够的降雨，农民会很高兴，因为他们的作物会很好。这一般可以扩展到自然灾害保险。

3. 去中心化的数据供应。对于差异的金融合约，实际上可以通过名为SchellingCoin的协议来使得数据供应去中心化。 SchellingCoin基本上按照如下工作：N方都将给定数据的值（例如ETH/USD价格）放入系统，值被排序，并且第25和第75百分点之间的每个人获得一个通证作为奖励。每个人都有动力提供答案，其他人也会，而且大部分人可以实际认同的唯一价值就是明显的默认：真相。这创建了一个去中心化的协议，理论上可以提供任意数量的值，包括ETH/USD价格，柏林的温度，甚至是特定困难计算的结果。

4. 智能多重签名托管。比特币允许多重签名交易合约，例如，给定五个密钥中的三个就可以花费资金。以太坊允许更细粒度;例如，五分之四可以花费所有资金，五分之三可以每天花费高达10%的资金，五分之二可以每天花费高达0.5%的资金。此外，以太坊多重签名是异步的 - 双方可以在不同时间在区块链上注册他们的签名，最后一个签名将自动发送交易。

5. 云计算。EVM技术还可用于创建可验证的计算环境，允许用户要求其他人执行计算，然后可选地请求证明某些随机选择的检查点的计算正确完成。这允许创建云计算市场，其中任何用户可以用他们的台式机，笔记本电脑或专用服务器来参与，并且可以与安全存款一起抽查来确保系统是可信的（即，节点不能有利地作弊） 。虽然这样的系统可能不适合所有任务;例如，需要高级别进程间通信的任务不能在大型节点云上轻松完成。但是，其他任务更容易并行化;像SETI@home, folding@home和基因遗传算法这样的项目可以很容易地在这样的平台上实现。

6. 点对点赌博。任意数量的点对点赌博协议，如Frank Stajano和Richard Clayton的Cyberdice，都可以在以太坊区块链上实现。最简单的赌博协议实际上只是下一个块hash的差异合约，并且可以从那里建立更高级的协议，创建具有接近零费用且无法作弊的赌博服务。

7. 预测市场。提供了oracle或SchellingCoin，预测市场也很容易实现，预测市场与SchellingCoin一起可能被证明是futarchy主流作为去中心化组织治理协议的第一个主流应用。

8. 链上去中心化市场，以身份和声誉系统为基础。

杂项和担忧

修改的GHOST实现

“Greedy Heaviest Observed Subtree（贪婪的最重要的观察子树）”（GHOST）协议是由Yonatan Sompolinsky和Aviv Zohar于2013年12月首次推出的创新.GHOST背后的动机是具有快速确认时间的区块链由于高过时率而目前遭受降低的安全性 - 因为区块需要一定的时间才能通过网络传播，如果矿工A挖掘一个区块，然后矿工B在矿工A的区块传播到B之前碰巧挖掘了另一个区块，那么矿工B的区块将最终被浪费掉，并且无助于网络安全。此外，存在一个集中问题：如果矿工A是具有30％哈希算力的矿池而B具有10％哈希算力，则A将有70％的时间产生过时区块的风险（因为另外30％的时间A产生了最后一个区块，因此将立即获得挖矿数据）而B将有90％的时间产生过时区块的风险。因此，如果区块间隔足够短以使过时速率变高，则仅通过其尺寸，A将大大提高效率。通过组合这两种效果，快速生成区块的区块链很可能导致一个矿池具有足够大的网络哈希算力百分比，以实际控制挖矿过程。

正如Sompolinsky和Zohar所描述的那样，GHOST通过在计算哪个链“最长”时包含过时区块来解决第一个网络安全丢失问题;也就是说，不仅是一个区块的父级和其他祖先，而且区块的祖先（在以太坊术语中，“叔区块”）的过时后代被添加到计算哪个区块具有最大的工作量证明在支撑它。为了解决中心化偏差的第二个问题，我们超越了Sompolinsky和Zohar所描述的协议，并且还提供了对过时区块的奖励：过时的区块获得其基本奖励的87.5％，并且包含过时区块的侄区块接收剩余的12.5％。但是，交易费不会给予叔区块。

以太坊实现了GHOST的简化版本，只有七个层级。具体来说，定义如下：

- 区块必须指定父区块，并且必须指定0个或更多叔区块
- 区块 `B` 中包含的叔区块必须具有以下属性：
  - 它必须是 `B` 的第 `k` 代祖先的直接子区块，其中 `2 <= k <= 7` 。
  - 它不能是 `B` 的祖先
  - 叔区块必须是有效的区块头，但不需要是先前验证的甚至有效的区块
  - 叔区块必须与先前区块中包含的所有叔区块以及同一区块中包含的所有其他叔区块不同（非双重包含）
- 对于区块 `B` 的每个叔区块 `U` ， `B` 的矿工获得额外3.125％的coinbase奖励， `U` 的矿工获得93.75％的标准coinbase奖励。

由于两个原因，这种包含最多7代的叔区块的有限版本的GHOST被使用。首先，无限的GHOST会在计算给定区块的哪些叔区块有效时包含太多复杂性。其次，在以太坊中使用的无限GHOST补偿消除了矿工在主链而不是公共攻击者链上挖矿的动机。

费用

由于发布到区块链中的每个交易都在网络上强加了需要下载和验证它的成本，因此需要一些监管机制，通常涉及交易费用，以防止滥用。在比特币中使用的默认方法是纯粹自愿收费，依靠矿工担任守门人并设定动态最小值。这种方法在比特币社区得到了非常好的支持，特别是因为它是“基于市场的”，允许矿工和交易发送者之间的供需决定价格。然而，这种推理过程的问题在于交易处理不是市场;虽然将交易处理解释为矿工向发送方提供的服务具有直观的吸引力，但实际上矿工所包含的每笔交易都需要由网络中的每个节点处理，因此绝大部分交易处理的成本都由很多第三方承担，而不是由决定是否包括它的矿工承担。因此，很可能发生公地悲剧问题。

然而，由于在市场机制中发现了这个缺陷，当给出一个特定的不准确的简化假设时，神奇地取消了它自己。论点如下。假设：

1. 一笔交易导致 k 个操作，向任何包含它的矿工提供奖励 kR ，其中R由发送者设置，并且 k 和 R 预先（大致）对矿工可见。
2. 一个操作对任何节点的处理成本为 C （假设所有节点具有相同的效率）
3. 有 N 个挖矿节点，每个节点具有完全相同的处理能力（即总数的 1/N ）
4. 不存在不挖矿的完整节点。

如果预期奖励大于成本，矿工将愿意处理交易。因此，预期奖励是 kR/N ，因为矿工具有处理下一个区块的 1/N 的机会，并且矿工的处理成本仅为 kC 。因此，矿工将包括 kR/N > kC 或 R > NC 的交易。注意， R 是发送方提供的每个操作的费用，因此是发送方从交易中获得的利益的下限， NC 是处理一个操作的整个网络的成本。因此，矿工有动力仅包括总功利利益超过成本的交易。

但是，现实与这些假设有几个重要的偏差：

1. 由于额外的验证时间延迟了区块传播，因此增加了区块过时的可能性，因此矿工确实支付了比其他验证节点更高的处理交易成本。
2. 确实存在不挖掘的完整节点。
3. 在现实中，矿力（挖矿权）分配可能是极端不平等的。
4. 确实存在投机者，政治敌人和疯子，他们的utility函数可能对网络造成损害，并且他们可以巧妙地建立合约，其成本远低于其他验证节点支付的成本。

(1)为矿工提供包含较少交易的趋势(2)增加 NC ；因此，这两种效应至少部分地相互抵消了。How? (3)和(4)是主要问题；为了解决它们，我们只需设置一个浮动上限：没有区块可以比 BLK_LIMIT_FACTOR 乘以长期指数移动平均线更多的操作。特别是：

```
blk.oplimit = floor((blk.parent.oplimit * (EMAFACTOR - 1) + floor(parent.opcount
* BLK_LIMIT_FACTOR)) / EMA_FACTOR)
```

BLK_LIMIT_FACTOR 和 EMA_FACTOR 是暂时设置为65536和1.5的常量，但在进一步分析后可能会更改。

在比特币中存在另一个阻碍大区块大小的因素：大块的区块将需要更长的时间来传播，因此具有更高的成为过时区块的可能性。在以太坊中，消耗很高gas的区块也可能需要更长的时间来传播，因为它们在物理上更大，并且因为它们需要更长的时间来处理交易状态转换以进行验证。这种延迟抑制因素在比特币中是一个重要的考虑因素，但由于GHOST协议，在以太坊中则不那么重要;因此，依靠受监管的区块限制可提供更稳定的基线。

计算和图灵完备性

重要的一点是，以太坊虚拟机是图灵完备的;这意味着EVM代码可以编码任何可以实现的计算，包括无限循环。 EVM代码允许以两种方式循环。首先，有一个 JUMP 指令允许程序跳回到代码中的前一个位置，一个 JUMPI 指令执行条件跳转，允许像 `while x < 27: x = x * 2` 这样的语句.其次，合约可以调用其他合约，可能允许循环递归。这自然会导致一个问题：恶意用户是否可以通过强制矿工和所有完整节点进入无限循环来关闭它们？出现这个问题是因为计算机科学中存在一个被称为暂停问题的问题：在一般情况下，没有办法知道某个程序是否会停止。

如状态转换章节所述，我们的解决方案通过要求交易设置允许的最大计算步骤数来工作，如果执行需要更长的时间，则计算被回滚，但仍然支付费用。消息以相同的方式工作。为了展示我们解决方案背后的动机，请考虑以下示例：

- 攻击者创建一个运行无限循环的合约，然后将一个激活该循环的交易发送给矿工。矿工将处理交易，运行无限循环，并等待它消耗完gas。即使执行耗尽gas并且中途停止，交易仍然有效，并且矿工仍然会对每个计算步骤向攻击者索取费用。
- 攻击者创建了一个非常长的无限循环，其目的是迫使矿工持续计算很长的时间，以至于计算结束时会有一些更多的块出现，矿工将不可能包含交易以索取费用。但是，攻击者将被要求提交 STARTGAS 的值，以限制执行可以采取的计算步骤的数量，因此矿工将提前知道计算将花费过多的步骤。
- 攻击者看到一个包含某种形式代码的合约，如 `send(A,contract.storage[A]); contract.storage[A] = 0` ，并发送一个具有足够gas的交易来运行第一步而不是第二步（比如发起取款而不让余额减少）。合约作者不需要担心防止此类攻击，因为如果执行在中途停止，更改则会被回滚。
- 金融合约的工作原理是获取九个专有数据供应的中位数，以便将风险降至最低。攻击者接管其中一个数据源，它可以通过DAO章节中描述的变量地址调用（variable-address-call）机制进行修改，并将其转换为运行无限循环，从而试图强制从金融合约提取资金以耗尽gas的任何尝试。但是，金融合约可以对消息设置gas限制以防止出现此问题。

图灵完备性的替代方法是图灵不完备性，其中 JUMP 和 JUMPI 指令不存在，并且在任何给定时间只允许每个合约的一个副本在调用堆栈中存在。有了这个系统，所描述的费用系统和我们解决方案有效性的不确定性可能就没有必要了，因为执行合约的成本将受到其规模的限制。此外，图灵不完备性甚至不是那么大的限制;在我们内部构思的所有合约示例中，到目前为止只有一个需要一个循环，这个循环甚至可以通过重复26次一行代码来删除该循环。鉴于图灵完备性的严重影响以及有限的收益，为什么不简单地使用图灵不完备的语言呢？然而，实际上，图灵不完备性远非一个解决问题的巧妙方法。要了解原因，请考虑以下合约：

```
C0: call(C1); call(C1);
C1: call(C2); call(C2);
C2: call(C3); call(C3);
...
C49: call(C50); call(C50);
C50: (run one step of a program and record the change in storage)
```

现在，向A发送一笔交易。因此，在51个交易中，我们有一个合同，需要2^{50}个计算步骤。矿工可以通过为每个合约维护一个值来指定它可以采取的最大计算步数来提前检测这种逻辑炸弹，并为递归调用其他合约的合约计算这个值，但这需要矿工禁止创建其他合约的合约（因为上述所有26个合约的创建和执行可以很容易地归成一个合约）。另一个问题是消息的地址字段是变量，因此通常甚至不可能知道给定合约将提前调用哪些其他合约。因此，总而言之，我们得出一个令人惊讶的结论：图灵完备性非常容易管理，并且非图灵完备性同样令人惊讶地难以管理，除非采取完全相同的控制措施 - 但在这种情况下让协议成为图灵完备的有何不可呢?

货币和发行

以太坊网络包括其自己的内置货币ether，其双重目的是提供主要流动性层，以实现各种类型的数字资产之间的有效交换，更重要的是，提供支付交易费用的机制。为了方便和避免未来的争论（参见比特币当前的mBTC / uBTC / satoshi辩论），面额将被预先标记：

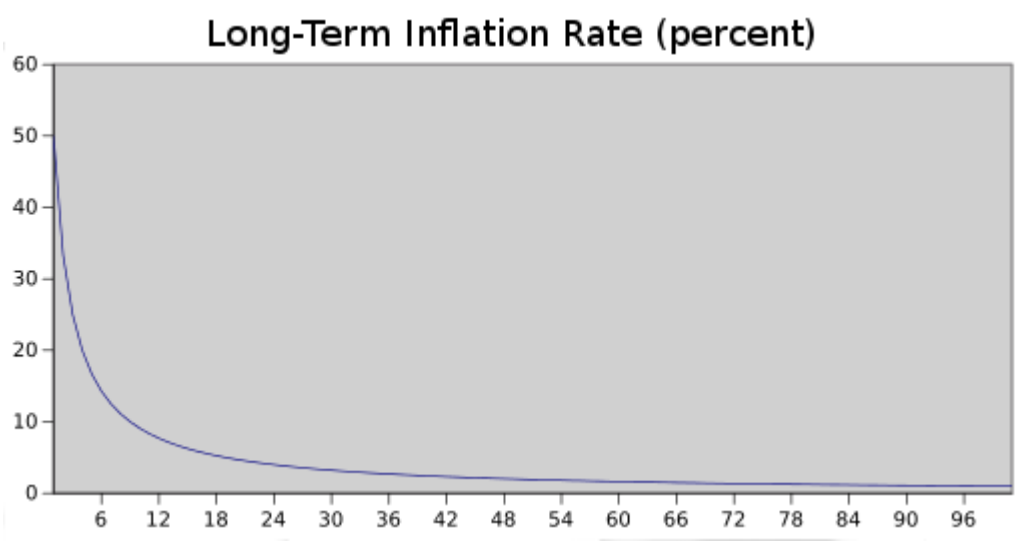
- 1: wei
- 10^{12}: szabo
- 10^{15}: finney
- 10^{18}: ether

这应该被视为“美元”和“美分”或“BTC”和“satoshi”概念的扩展版本。在不久的将来，我们希望“ether”用于普通交易，“finney”用于微交易，“szabo”和“wei”用于围绕费用和协议实现的技术讨论;剩余的面额可能会在以后变得有用，此时不应包括在客户端中。

发行模式如下：

- Ether将以每个BTC 1000-2000ether的价格以货币销售形式发售，该机制旨在为以太坊组织提供资金，并支付其他平台（如Mastercoin和NXT）成功使用的开发费用。较早的买家将受益于更大的折扣。从销售中获得的BTC将完全用于向开发工程师支付工资和奖金，并投资于以太坊和加密货币生态系统中的各种营利性和非营利性项目。
- 销售总额的0.099倍（60102216 ETH）将分配给组织，以补偿早期贡献者并支付在创世区块之前ETH计价的费用。
- 售出总额的0.099倍将作为长期储备。
- 在那个时间点之后，销售总额的0.26倍会被每年永久分配给矿工。

Group	At launch	After 1 year	After 5 years
Currency units	1.198X	1.458X	2.498X
Purchasers	83.5%	68.6%	40.0%
Reserve spent pre-sale	8.26%	6.79%	3.96%
Reserve used post-sale	8.26%	6.79%	3.96%
Miners	0%	17.8%	52.0%



尽管线性货币发行，就像比特币一样随着时间的推移，供应增长率趋于零

上述模型中的两个主要选择是：（1）捐赠池的存在和规模，以及（2）存在永久增长的线性供应，而不是像比特币那样的上限供应。捐赠池的理由如下。如果捐赠池不存在，并且线性发行减少到0.217倍以提供相同的通货膨胀率，那么ether的总量将减少16.5%，因此每个单位的价值将增加19.8%。因此，在均衡状态下，19.8%的ether会在销售中被购买，所以每个单位将再次像以前一样有价值。然后组织也将拥有1.198倍的BTC，可以认为它被分成两个片段：原始BTC和额外的0.198倍。因此，这种情况完全等同于捐赠，但有一个重要的区别：组织纯粹持有BTC，因此没有动力或被激励去支持ether的单位价值。

永久线性供应增长模型降低了某些人认为的在比特币中财富过度集中的风险，并为生活在当前和未来时代的个人提供了获得货币单位的公平机会，同时保留了获取和持有ether的强大动力，因为随着时间的推移，“供给增长率”百分比仍然趋于零。我们还认为，由于粗心，死亡等原因，币总会随着时间的流逝而丢失，而币丢失可以模拟为每年总供应量的百分比，因此流通中的货币总供应量最终将稳定在一个等于年度发行量除以损失率的值（例如，损失率为1%，一旦供应量达到26倍，那么0.26倍将被挖矿，并且每年损失0.26倍，形成均衡状态）。

请注意，在未来，以太坊很可能会转向使用权益证明模型以确保安全性，将发行需求降低到每年0到0.05倍之间。如果以太坊组织丢失资金或因任何其他原因消失，我们将公开“社会合约”：任何人都有权创建未来的以太坊候选版本，唯一的条件是ether的数量必须是最多等于 `60102216 * (1.198 + 0.26 * n)`，其中 `n` 是创世区块后的年数。创作者可以自由地面向大众销售或分配一些或者所有权益证明驱动的供应扩张和最大允许供应扩张之间的差值，以支付开发费用。不符合社会合约的候选人升级可以无可非议地被分叉为合规版本。

挖矿中心化

比特币挖矿算法的工作原理是让矿工一次又一次地对区块头的细微修改版本计算SHA256数百万次，直到最终一个节点出现一个哈希小于目标的版本（目前大约为2^{192}）。但是，这种挖矿算法容易受到两种集中形式的影响。首先，挖矿生态系统已经由ASIC（专用集成电路），专门设计的计算机芯片（使得对于比特币挖矿的特定任务的效率提高了数千倍）所控制和主导。这意味着比特币挖矿不再是高度去中心化的，也不再是平等的追求，二是需要数百万美元的资金才能有效参与。其次，大多数比特币矿工实际上并不在本地进行区块验证;相反，他们依靠集中式矿池来提供区块头。这个问题可能更糟糕：截至撰写本文时，排名前三的矿池间接控制着比特币网络中大约50%的处理能力，尽管当一个矿池或者矿池联盟试图发起51%攻击时矿工可以切换到其他矿池这一事实可以减轻这种影响。

目前以太坊的意图是使用一种挖矿算法，其中矿工需要从状态中获取随机数据，从区块链中的最后N个区块计算一些随机选择的交易，并返回结果的哈希。这有两个重要的好处。首先，以太坊合约可以包括任何类型的计算，因此以太坊ASIC本质上是用于一般计算的ASIC - 比如，一个更好的CPU。其次，挖矿需要访问整个区块链，迫使矿工存储整个区块链，并至少能够验证每笔交易。这消除了对集中式矿池的需求;尽管矿池仍然可以起到平衡奖励分配随机性的合法作用，但是这个功能可以被没有中央控制的点对点矿池同样很好使用。

此模型未经测试，在使用合约执行作为挖矿算法时，在避免某些聪明的优化时可能会遇到困难。然而，该算法的一个特别有趣的特点是，它允许任何人通过在区块链中引入大量专门用于阻碍某些ASIC的合约来“poison the well”。 ASIC制造商有经济刺激，所以使用这种技巧相互攻击。因此，我们正在开发的解决方案最终是一种适应性的经济人类解决方案，而不是纯粹的技术解决方案。

可扩展性

关于以太坊的一个普遍关注点是可扩展性问题。与比特币一样，以太坊也存在缺陷，即每个交易都需要由网络中的每个节点处理。当前比特币区块链的大小约为15 GB，每小时增长约1 MB。如果比特币网络每秒处理Visa的2000笔交易，它将每3秒增加1 MB（每小时1 GB，每年8 TB）。以太坊很可能会遭遇类似的增长模式，甚至可能更糟糕，因为在以太坊区块链之上会有许多应用程序，而不是像比特币那样只是一种货币，但由于以太坊完整节点仅仅需要存储状态而不是整个区块链历史这一事实，这个问题可以得到改善。

如此大的区块链大小（体量）的问题是中心化风险。如果区块链大小增加到100 TB，则可能的情况是只有极少数大型企业会运行完整节点，所有常规用户都使用轻量SPV节点。在这种情况下，可能出现的问题是所有完整节点可以勾结在一起并且都同意以某种有利可图的方式作弊（例如，改变区块奖励，给自己BTC）。轻量节点无法立即检测到这一点。当然，至少有一个诚实的完整节点可能存在，几个小时后，有关欺诈的信息会通过像Reddit这样的渠道流出，但那时为时已晚：这将由普通用户组织起来努力将给定的区块列入黑名单，这是一个巨大且可能不可行的协调问题，其规模与取得成功的51%攻击类似。在比特币的情况下，这是一个目前存在的问题，但存在一个Peter Todd建议采用的区块链修改可以缓解这一问题。

在短期内，以太坊将使用另外两种策略来应对这一问题。首先，由于基于区块链的挖矿算法，至少每个矿工都将被强制为完整节点，从而在完整节点数量上创建下限。其次，更重要的是，我们将在处理每笔交易后在区块链中包含一个中间状态树根。即使区块验证是中心化的，只要存在一个诚实的验证节点，就可以通过验证协议规避中心化问题。如果矿工发布一个无效区块，则该区块必须格式错误，或者状态 `S[n]` 不正确。由于已知 `S[0]` 是正确的，因此必然存在第一个不正确的状态 `S[i]` ，并且 `S[i-1]` 是正确的。验证节点将提供索引 `i` ，以及由需要处理 `APPLY(S[i-1],TX[i]) -> S[i]` 的Patricia树节点的子集组成的“无效证明”。节点将能够使用那些Patricia节点来运行该部分计算，并且看到生成的 `S[i]` 与提供的 `S[i]` 不匹配。

另一个更复杂的攻击将涉及恶意矿工发布不完整的区块，因此甚至不存在完整信息来确定区块是否有效。对此的解决方案是质询 - 响应协议：验证节点以目标交易索引的形式发出“质询”，并且在接收节点时，轻节点将区块视为不可信，直到另一个节点（无论是矿工还是其他验证者）提供Patricia节点的一个子集作为有效性的证明。

结论

以太坊协议最初被设想为加密货币的升级版本，通过高度通用的编程语言提供诸如链上托管，取款限制，金融合约，赌博市场等高级功能。以太坊协议不会直接“支持”任何应用程序，但是图灵完备编程语言的存在意味着理论上可以为任何交易类型或应用程序创建任意合约。然而，以太坊更有趣的是，以太坊协议远远超出了货币。围绕去中心化文件存储，去中心化计算和去中心化预测市场的协议，以及其他许多此类概念，有可能大幅提高计算机行业的效率，并通过首次增加了一个经济层而对其他点对点协议提供大量的推动。最后，还有大量与金钱无关的应用程序。

由以太坊协议实现的任意状态转换功能的概念提供了具有独特潜力的平台;而且，对于数据存储，赌博或金融领域的特定应用程序而言，以太网不是一个封闭式的单一用途协议，而是设计为开放式的，我们相信在未来的几年中，对于大量的金融和非金融协议，以太坊非常适合用作基础服务层。

备注和延伸阅读

备注

1. 一个富有经验的读者可能会注意到，实际上一个比特币地址是椭圆曲线公钥的哈希，而不是公钥本身。但是，事实上，将pubkey哈希称为公钥本身是完全合法的加密术语。这是因为比特币的加密可以被认为是一种自定义数字签名算法，其中公钥由ECC pubkey的哈希组成，签名由与ECC签名连接的ECC pubkey组成，验证算法包括根据作为公钥提供的ECC pubkey哈希检查签名中的ECC pubkey，然后根据ECC pubkey验证ECC签名。
2. 从技术上讲，前11个区块的中位数。
3. 以太坊协议应该尽可能简单，但可能需要具有相当高的复杂度，例如扩展，将存储成本内部化，带宽和I / O，以确保安全性，隐私性，透明性等。在需要复杂性的地方，文档应尽可能清晰，简洁和保持最新，以便完全未受过以太坊教育的人可以学习并成为专家。

4. 请参阅以太坊虚拟机的黄皮书（作为从头开始构建以太坊客户端的规范和参考），同时在以太坊wiki中也有许多主题，例如分片开发，核心开发，dapp开发，研究，Casper R＆D和网络协议。对于研究和未来可能的实现，有ethresear.ch。
5. 另一种表达方式是抽象。最新的路线图计划抽象执行，允许执行引擎不一定必须遵循一个规范，但是例如它可以针对特定应用程序以及分片进行定制。（这种执行引擎的异构性没有在路线图中明确说明。还有被Vlad Zamfir概念化了的异构分片。）
6. 在内部，2和“CHARLIE”都是数字，后者是big-endian base 256表示。数字可以是至少0且至多2^{256}-1。

延伸阅读

1. Intrinsic value: http://bitcoinmagazine.com/8640/an-exploration-of-intrinsic-value-what-it-is-why-bitcoin-doesnt-have-it-and-why-bitcoin-does-have-it/
2. Smart property: https://en.bitcoin.it/wiki/Smart_Property
3. Smart contracts: https://en.bitcoin.it/wiki/Contracts
4. B-money: http://www.weidai.com/bmoney.txt
5. Reusable proofs of work: http://www.finney.org/~hal/rpow/
6. Secure property titles with owner authority: http://szabo.best.vwh.net/securetitle.html
7. Bitcoin whitepaper: http://bitcoin.org/bitcoin.pdf
8. Namecoin: https://namecoin.org/
9. Zooko's triangle: http://en.wikipedia.org/wiki/Zooko's_triangle
10. Colored coins whitepaper: https://docs.google.com/a/buterin.com/document/d/1AnkP_cVZTCMLIzw4DvsW6M8Q2JC0lIzrTLuoWu2z1BE/edit
11. Mastercoin whitepaper: https://github.com/mastercoin-MSC/spec
12. Decentralized autonomous corporations, Bitcoin Magazine: http://bitcoinmagazine.com/7050/bootstrapping-a-decentralized-autonomous-corporation-part-i/
13. Simplified payment verification: https://en.bitcoin.it/wiki/Scalability#Simplifiedpaymentverification
14. Merkle trees: http://en.wikipedia.org/wiki/Merkle_tree
15. Patricia trees: http://en.wikipedia.org/wiki/Patricia_tree
16. GHOST: https://eprint.iacr.org/2013/881.pdf
17. StorJ and Autonomous Agents, Jeff Garzik: http://garzikrants.blogspot.ca/2013/01/storj-and-bitcoin-autonomous-agents.html
18. Mike Hearn on Smart Property at Turing Festival: http://www.youtube.com/watch?v=Pu4PAMFPo5Y
19. Ethereum RLP: https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-RLP