
# De centralization

# Demos 白皮书 v0.1

## 摘要

从技术的角度而言，区块链的应用落地离不开智能合约的大规模应用，这就要求智能合约必须降低使用者的门槛。

不论是基于哪种机制的智能合约，都应该把用户体验放在首位。因此，智能合平台应该要考虑到：多链兼容、向后拓展。

优秀的智能合约平台不仅要支持内部的智能合约需求，也应该支持第三方公链的智能合约需求。Demos 是所有 PoW 公链的拓展链，可以帮助第三方公链完成产业升级和应用落地，即多链兼容，

没有一项技术不需要升级，包括区块链。对于传统基础链而言，向后拓展是一个亟待解决的问题。我们在每个区块数据中加入了版本号，用来统一协调版本问题，解决硬分叉导致的用户流失。

Demos 是由一条基础链和一条拓展链构成的区块链平台，在确保去中心化的同时突破智能合约瓶颈。

我们想让更多的人参与到 Demos 的建设当中，解决安全性和一致性等问题后，Demos 的拓展链是没有难度的。我们将考虑核心放在这个节点的稳定性上，一台普通的电脑就可以进行拓展链的打包工作，并且获得由 PoW 机制生产的 Demos 基础链 Token 作为打包奖励。

**关键词：**智能合约、拓展链、移动端、PoW+PoS 共识机制、闪电网络、原子互换、防止硬分叉、社区自治、Demos 构架

# 目 录

# 1. 简介

“智能合约”（Smart contract）这个术语至少可以追溯到 1995 年，由计算机科学家、密码学家尼克·萨博（Nick Szabo）首次提出：“一个智能合约是一套以数字形式定义的承诺（commitment），包括合约参与方可以在上面执行这些承诺的协议。”

下面将详细描述智能合约通过 P2P 网络扩散并存入区块链的过程，包括如下步骤：

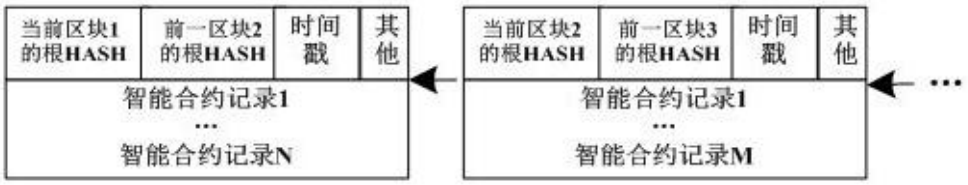
**步骤 1**：合约通过 P2P 的方式在区块链全网中扩散，每个节点都会收到一份。区块链中的验证节点会将收到的合约先保存到内存中，等待新一轮的共识时间，触发对该份合约的共识和处理；

**步骤 2**：到共识时间，验证节点将把最近一段时间内保存的所有合约打包成一个合约集合（set），并算出这个合约集合的 Hash 值，最后将这个合约集合的 Hash 值组装成一个区块结构，扩散到全网；

其它验证节点接收到这个区块结构后，会把里面包含的合约集合的 Hash 取出来，与自己保存的合约集合进行比较，同时发送一份自己认可的合约集合给其它的验证节点；

通过这种多轮的发送和比较，所有的验证节点最终在规定的时间内对最新的合约集合达成一致。

**步骤 3**：最新达成的合约集合会以区块的形式扩散到全网，如下图所示。




**合约区块链示意图**

每个区块包含以下信息：当前区块的 Hash 值、前一区块的 Hash 值、达成共识的时间戳、其它描述信息；

同时，区块链最重要的信息是带有一组已经达成共识的合约集。收到合约集的节点，都会对每条合约进行验证，验证通过的合约才会最终写入区块链中，要验证的内容主要为合约参与者的私钥签名是否与账户匹配。

下面是区块链中智能合约自动执行的过程，包括如下几步：

**第一步：**智能合约会定期检查自动机状态，逐条遍历每个合约内包含的状态机、事务以及触发条件；将条件满足的事务推送到待验证的队列中，等待共识；未满足触发条件的事务将继续存放在区块链上。

**第二步：**进入最新轮验证的事务，会扩散到每一个验证节点，与普通区块链交易或事务一样，验证节点首先进行签名验证，确保事务的有效性；验证通过的事务会进入待共识集合，等大多数验证节点达成共识后，事务会成功执行并通知用户。

**第三步：**事务执行成功后，智能合约自带的状态机会判断所属合约的状态，当合约包括的所有事务都按顺序执行完后，状态机会将合约的状态标记为“完成”，并从最新的区块中移除该合约；

反之将标记为“进行中”，继续保存在最新的区块中等待下一轮处理，直到处理完毕。整个事务和状态的处理都由区块链底层内置的智能合约系统自动完成，全程透明、不可攥改。

由此我们可以了解，智能合约在存证、溯源、数字资产等领域有着广泛的应用场景。但一些著名的提供智能合约的区块链平台，也存在一些不合理之处，阻碍了智能合约的大规模应用。例如以太坊和 ROS。

以太坊使用的是 PoW 共识机制，因为链上数据体积的原因不能进行快速打包，导致智能合约的 TPS 并不能满足大型商用的需求，并且每次调用智能合约都需要客户端用户支付 Gas，这样的设计让用户体验感极差。但 PoW 共识机制节点分布比 PoS 共识机制要多，安全性有保障。

EOS 智能合约也有商用上的局限。虽然 EOS 解决了以太坊存在的 Gas 费用需要用户支付的成本问题，但 EOS 系统中的 RAM 抵押方法却增加了用户的学习成本。

我们认为成本应该分为两部分：使用成本和学习成本。EOS 为了达到了 3000~4000 的 TPS，在“去中心化”方面做出了巨大的让步。相对于比特币和以太坊全网上拥有的万个节点，EOS 全网只有 21 个超级节点产块，在“去中心化”方面受到很大的质疑。

Demos 基础链采用 PoW+PoS 共识机制，拓展链将采用 Demos 工作证明机制。二者职责分布清晰，基础链主要负责 DOS 产出、DOS 的转账等工作，拓展链则主要提供智能合约的使用。

这样的结构可以在保证基础链足够安全的同时，保证拓展链的性能不受基础链

影响，我们将在白皮书的后续部分来阐述这个巧妙的设计。

# 2.Demos 是什么？

Demos 来源于古希腊语，意为民主。在今天也有集合的意思，我们想给越来越多的 PoW 公链提供帮助，在帮助的过程中，集合公链。

Demos 的主体由一条基础链和一条拓展链构成，它的形状类似于 DNA 的双螺旋结构，是一个无数组对称数据交叉记录交叉校验的结构。在兼顾性能和安全的同时，Demos 做到了完全的去中心化。

**Demos 结构示意图**

Demos 是完全开放的、去中心化的应用，内置有 DOS 基础链代币，通过 PoW+PoS 混合共识机制产出。

DOS 可以通过双向校验的方式与拓展链 DAPP 进行交互，作为所有 DAPP 之间的桥梁和媒介。

DOS 还是兑换拓展链 Gas 的途径。一般情况下，Gas 费用的产生有两方面的原因：一是激励机制，矿工可以通过打包数据获取奖励；二是利用手续费的概念防范流量攻击。

# 3.Demos 面向哪些用户？

Demos 除了提供一些基础服务以外，还提供技术和工具上的支持，主要面对开发者、企业、普通用户。

## 一、面向开发者

Demos 中的开发者可以使用开发工具快速开发 DAPP 并且进行发布，DAPP 可以运行在浏览器和移动端和 PC 端。

对此，我们提供了非常丰富的钱包工具，可以进行原生接入，也可以进行网页接入。我们还会将一些优秀的 DAPP 录入到官方钱包，我们非常看重移动端支持和生态建设。

## 二、面向企业和第三方公链

Demos 中的企业用户可以使用 Demos 提供的开源拓展链，拓展链可单独运行也可以构建联盟链。

我们在后续会推出部署工具，让企业部署起来更为方便，使用官方提供的 DAPP 开发工具就可以快速开发、发布基于企业私有链/联盟链的 DAPP。

他们还可以使用原子交换的方式与 Demos 进行交互。

对于 POW 公链来说，未来的节点数量骤减是不可避免的，Demos 的出现会改善这一状况，我们在后续会推出各公链兼容拓展链的开源代码，帮助 POW 公链应对节点流失和缺少应用场景等问题。

## 三、面向普通用户

Demos 中的普通用户可以参与到整个 Demos 的治理中。

我们提供了基于 PoS 的投票自治的系统，用户可以对 Demos 中每个社区决策进行干预，也可以提出建设性的意见指引 Demos 的方向。

我们设立了社区基金会，基金会将奖励每一个对社区做出贡献或者提出宝贵意见的人。普通用户还可以通过移动端、应用商店、网页等入口体验使用基于 Demos 开发的 DAPP。

# 4.设计理论

## 一、图灵完备和图灵不完备

图灵完备意味着你的语言可以做到能用图灵机做到的所有事情，可以解决所有的可计算问题。图灵不完备也不是没有意义，有些场景我们需要限制语言本身。如限制循环和递归，以保证该语言能写的程序一定是终止的。

简单来说，图灵完备的语言，有循环执行语句，判断分支语句等，理论上能解决任何算法，但有进入死循环而程序崩溃的可能。

图灵不完备，应该是不允许或限制循环。可以保证，每段程序都不会死循环，都有运行完的时候。

比特币脚本语言考虑了安全等问题，设计为图灵不完备的；而以太坊采用的是智能合约语言是图灵完备的。

图灵完备语言最显著的一个特点是支持循环，所谓循环，就是程序能不断执行下去。那么在区块链支撑的分布式环境下，矿工如何判断一个程序何时结束呢？

图灵计算理论，已有人证明过，要证明一个程序能不能终止是不可能的（图灵停机问题），所以这种“智能合约”语言需要保证所写出的程序不能存在死循环。

这也是为什么以太坊语言会加入 Gas（燃气）的原因，通过加入 Gas，使程序每个运算过程都会消耗一定成本，从而不会无限制地执行下去。

作为智能合约机，使用图灵完备的语言会比图灵不完备语言更为优秀，在开发过程中对循环的依赖非常高。

## 二、UTXO 模型和账户模型

未花费的交易输出（Unspent Transaction Output）Transaction 被简称为 TX。

比特币的交易由交易输入和交易输出组成，每一笔交易都要花费（spend）一笔输入，产生一笔输出（output），而其所产生的输出，就是“未花费过的交易输出”，也就是 UTXO。

UTXO 的主要优势在于很强的私密性，每笔交易都可以生产新的地址，从而无法追踪，对于货币来说这是好事，但是对于 DAPP 来说更多的是弊端，很多需要跨链的公链都遇见了这个问题，并且基本没有得到很好的解决。

Demos 使用了两套模型，即 UTXO 模型和账户模型，并在账户模型中添加了权限属性。我们用这种方式去兼顾隐私安全和智能合约开发。

在 Demos 中，你在基础链上的交互是采用的是 UTXO 模型，而你在拓展链上的交互采用的是账户模型。

## 三、去中心化、安全和高性能

DCR 被人津津乐道的地方，是在于其 PoW+PoS 的共识机制，这个共识机制确保了安全性和去中心化的最大实现，可以说将二者的兼顾做到了近乎极致。但由于 DCR 是 UTXO 模型与比特币一致，而智能合约需要使用图灵完备语言进行开发，所以 DCR 想要完成智能合约机的兼容非常困难。

Demos 高度赞成 PoW+PoS 的共识机制，并且在 Demos 基础链中保留了这些属性，在高度自治的去中心化基础链中加入了可相互校验的高性能拓展链，兼容了 EVM 智能合约机。Demos 在确保了高性能智能合约的执行的同时，还确保了交易的安全性和去中心化。

# 5.Demos 的特点

## 一、基础链

### （1）PoW+PoS 共识机制

在 Demos 中，每个区块产生一定数量的 DOS 作为奖励，但是这个奖励并不会全额发放给 PoW 矿工。30%的奖励由 PoS 投票者获得，60%的奖励由 PoW 矿工获得，剩余 10%的奖励会进入到社区账户中作为社区基金。社区基金将会用于奖励每一个对社区提供有效意见或对社区做出贡献的人。

在 PoW 方面，Demos 使用的是 Blake256 算法，需要显卡的硬计算能力，该算法在 x86-64 架构上面有着更好的性能表现，而且安全性更高。

在 PoS 方面，Demos 上的 PoS 协议使得用户可以通过持票（ticket）来为 Demos 的某项改进提案（DCC，类似于比特币里面的 BIP 与以太坊里面的 EIP）进行投

票，比如：开发团队是否需要在某项新功能的开发上投入时间，是否要激活某项功能，如何使用经费，等等。

为了参与 PoS，用户需要锁定自己的一部分 DOS 代币来换取 ticket，每张 ticket 可以进行一次投票，投票完成之后，用户解锁相应的 DOS 代币并能够获得一定的奖励，这个过程我们称之为 PoS 挖矿。

每张 ticket 会被随机选举以参加投票，平均时间大约是 28 天，但最高可能会达到 142 天，并有 0.5%的机率选票被作废。

PoW 矿工生成每个区块时，至少需要有 5 张选票进行确认，如果少于 5 张，PoW 矿工的奖励会减少以示惩罚，同时最多可以有 20 张选票。

PoS 的难度调整算法将每隔 144 个区块调整一次 ticket 的价格，以维持票池里的票数在 40960 左右。

用户参与 PoS 挖矿需要支付选票费给 PoW 矿工，以便 PoW 矿工将自己的 ticket 打包放入新挖的区块之中。当 ticket 被选中进行投票的时候，用户的钱包应当在线以行使自己的投票权，如果无法投票，则用户不能获得奖励。

为了方便无法保持全天在线的用户，用户可以选择 PoS 矿池代为投票，这需要向 PoS 矿池支付一定的代理费用。

## （2）社区百分之百自治

在 Demos 主网中，用户可以发起提案和参与提案投票，这个投票系统我们把它叫做 demos Convention Center，简称为 DCC，是一个脱离区块链存储版本和时间戳数据的系统，本质上是“git，一个流行的版本控制系统，外加时间戳”。

我们并不是试图把所有与 Demos 管理有关的数据在区块链上存储起来，而是选择创建一个脱离区块链存储的数据，并将其锚定在 Demos 的区块链中，从而最大限度地减少链上的占用空间。

用户可以在版本控制和时间戳环境中创建和维护任意数据。DCC 可以脱离 Demos 而被使用，尽管它依赖 Demostime 服务器来创建时间戳。

因为提交收录的数据可能存在格式错误、包含不恰当的内容或者在纳入 DCC 之前需要满足其他限制条件，DCC 需要通过人工审查提交的提案。因此，我们为新的提案创建了两个状态：未审查和审查。此分类非常有用，

每个提案提交都有相应的审查令牌，允许提案提交者公开证明他们的提案被审查通过，如果未审查通过，管理者将删除它。这个审查标记能够让 DCC 避免现代社交媒体网站中一个更隐秘和常见的做法--即在这些网站上数据被内部审

查。通过使用审查令牌，DCC 创建了一个透明的审查程序，所以当被发现不合适的审查行为时，管理员将被公开对其审查行为问责。

## （3）防止硬分叉

Demos 不会发生像比特币那样的分叉问题。

首先，我们有一个链上投票机制，以表决与硬分叉相关的问题；

其次，Demos 分叉绝不会像比特现金或 Segwit2x 那样，这是因为 PoW 矿工产生的新区块需要与 Demos 股东（PoS 矿工）的意愿相吻合。

如果 PoW 旷工拒绝遵从集体政策，那么 PoS 股东可以判定新区块无效。比如，80%的股东支持激活闪电网络，而 20%的股东拒绝，那么这 80%的股东就可以直接投票让旧链上的新区块无效。这样一来，PoW 矿工就会直接从旧链转移到激活了闪电网络的新链上，旧链也就无效了。

这一部分的设计也包含在我们的拓展链当中，用来保证双链的一致性和未来 Demos 发展的民主自治。

传统基础链对硬分叉问题非常无奈，基础链为了维护自身的先进与合理，在某些时候不得不做出分叉行为，但是这对于以应用为主的基础链来说就等同于用户流失，恶劣影响是长远存在的。

## （4）原子交换

2013 年，Tier Nolan 在 Bitcoin Talk 论坛上首次提出了原子交换。Nolan 通过使用不同类型区块链上的简单加密货币交易，概述了跨链加密货币互换的基本原则。

像 Coinbase Pro，Binance 或 Gemini 这样的中心化交易平台，可以说是加密货币世界的金融机构。如果没有这些平台，大多数加密交易者和投资者都很难交换加密货币。实际上，集中交换已成为加密货币交易的守门人。

那么，原子交换是如何工作的呢？原子交换使用一种特殊类型的智能合约，称为哈希时间锁合约（hash timelock contract，简称 HTCL）。

这就像一个需要两个特殊钥匙的“虚拟保险箱”：

1. 一个是 HashLock 密钥：只有当所有各方都在各自的交易上签名时，才会将已交易的加密货币分发给交易者；

2. 一个是 TimeLock 密钥：一种安全机制，如果交易没有在指定的时间内完成，它会将交易的加密货币返还给交易者。

要进行原子交换，第一方将创建一个 HTCL 地址，然后存入加密货币。之后，创建此加密货币的密码，这被称为原像，随后进行哈希加密（一个“锁住”原像的过程）。

接着，将这个哈希加密后的原像转发给另一方，后者将验证加密货币的资金是否已存入。然后，第二个交易者将把他们的交易资金存入一个新地址，这个新地址是用相同哈希创建的。

第一方使用用于存放初始交易资金的密码，解锁由第二方存入的交易资金。然后，第二方可以解锁第一方存入的交易资金。这就是进行了原子交换。

## 这种方式的优点在于：

- **做到了真正的去中心化**

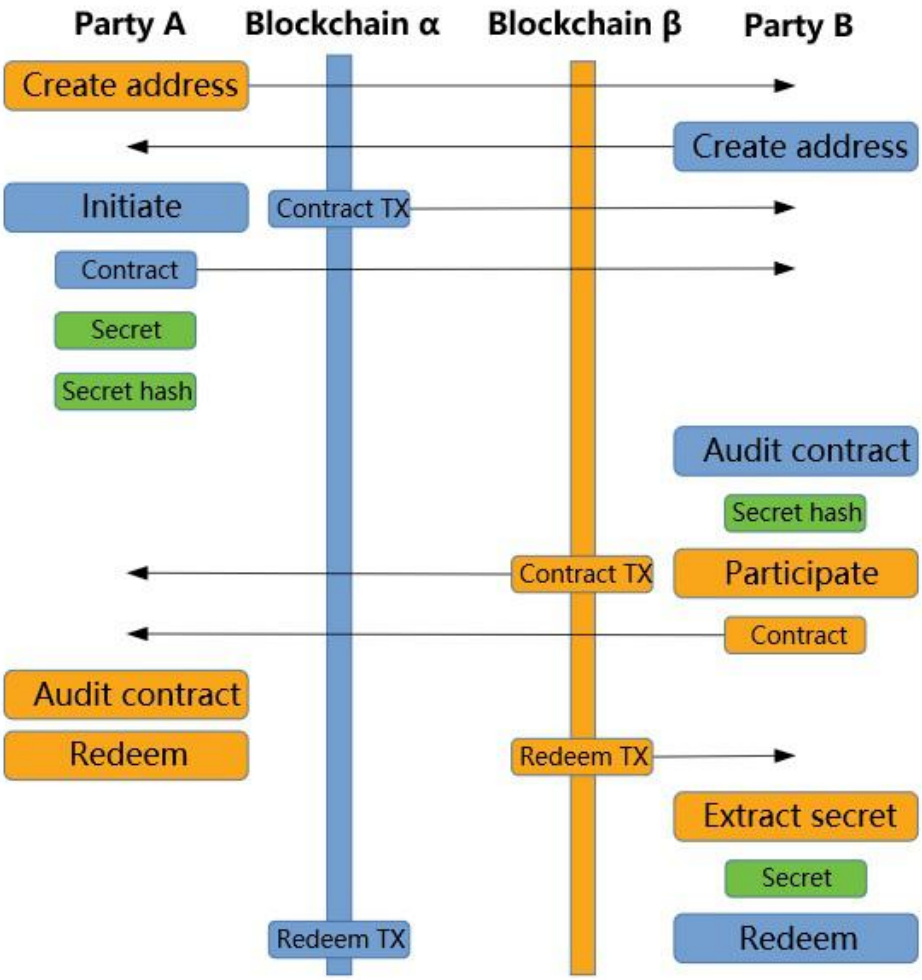
原子交换允许加密货币交易者交换数字资产，无需第三方仲裁，没有交易对手的风险。

- **降低了投资者风险**

原子交换使加密货币持有者始终完全控制其加密货币，消除了中心化交易所带来的黑客风险。

- **降低了交易费用**

与原子交换相关的唯一费用是标准的区块链交易费用，使原子交换成为免费的交易方式。




## （5）闪电网络

闪电网络（Lightning Network）简称 LN，是一个去中心化的系统。它的卓越之处在于，无需信任对方或第三方即可实现实时的海量交易。主要作为用于即时、高容量的微支付。

闪电网络起源于比特币的扩容问题，状态通道的典型应用，也是一个分布式网络，利用区块链的特性消除将资金托管给第三方带来的风险。

闪电网络的目的是实现安全的链下交易，本质上是使用了哈希时间锁智能合约来安全地进行 0 确认交易的一种机制，通过设置巧妙的智能合约，完善链下通道，使得用户可以在闪电网络上进行 0 确认交易。

下面引用一个简单易懂的例子：

A 每天都要在包子铺买早餐，但是 A 觉的每天都用加密货币结算太麻烦，不仅到账速度慢，而且每天都产生手续费。消耗的手续费甚至比包子的价格还要高，所以 A 决定和商店老板 B 说，能不能在你这里挂个帐月底结算。老板 B 说，

不行说一旦你跑了怎么办，你可以预付给我一个月的钱然后我给你记账。这样 A 也觉得不妥。最后两人决定找一个箱子，然后 A 把币放在箱子里，箱子里上了两把锁，只有 A 和 B 同时有钥匙才能打开，A 和 B 有一个共同的账本，每一笔交易都会记录在账本上，每一笔交易只要双方签个字就能确认这笔交易，只要箱子里的钱还够这个支付就会一直进行下去。

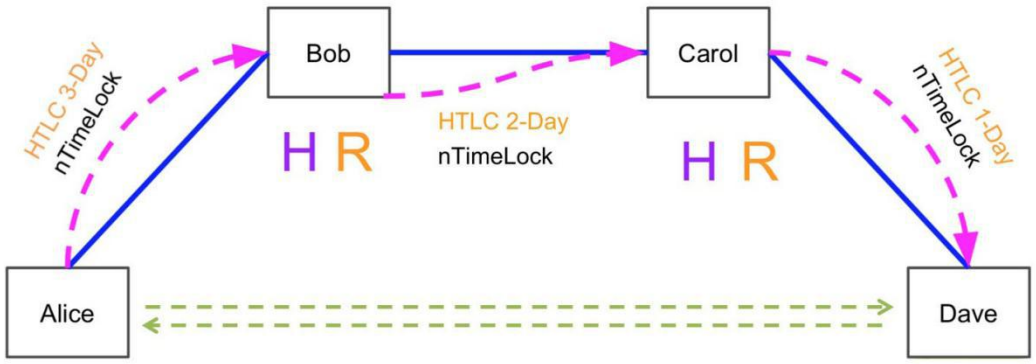
这样的过程就是相当于形成了之前说的链下通道，全部是在链下进行的，只有最后结算的时候才会放到基础链上结算，计入区块。

从头到尾只有两条信息是计入公链上的，一个是将币放在箱子里的锁币信息，还有一个就是结算时候的信息。

闪电网络真正厉害的地方是利用这种通道让没有建立关系的两个人建立通道：

假设有 ABC 三个人，A 和 B 之间有通道，B 和 C 之间有通道，A 想要把币转给 C，就可以通过 B，但是其中产生了信用问题，如果 B 在中间把钱拿跑了怎么办？

为了解决这种问题，闪电网络引用了链下暗语，相当于公钥私钥。A 把箱子给 B，箱子上带有 A 设定的暗语，B 是无法拿出来，当 B 把箱子发给 C 后，A 会把暗语告诉给 C，这样 C 就能拿到箱子里的币了。




## 二、拓展链

在基于安全、稳定、去中心化之上的高性能智能合约实现方案。

## （1）Demos 共识机制

Demos 拓展链使用的共识机制是结合了 PoW 和 PoB 共识机制新型共识机制。

传统 PoW 共识机制在发展过程中逐渐从原先的注重节点数量演变成了全球的算力比拼，这完全背离了中本聪设计比特币的初衷。

而 PoS 包括 PoS 的延伸共识机制也变成了一场全球的财力比拼。

Demos 想用一种方式将共识机制回归至本源，让节点数量越来越多，并且安全可靠，去中心化的体现越来越强，并且兼顾性能。

Demos 共识机制与其他工作证明机制最大的区别是它的证明关键点是前置的。也就是说，你要参与这个工作就必须要证明你有这方面的能力，为此我们构建了信誉漏斗系统，这个系统会负责矿工列表的筛选和维护。

### Demos 的工作原理是这样的：

Demos 共识机制是没有挖矿难度的。

信誉漏斗的结构类似于内存池，内存池主要的工作就是筛选和维护交易列表，而信誉漏斗的工作是筛选和维护矿工列表。

信誉漏斗分成三个部分：优质节点、普通节点、劣质节点。

优质节点进行日常打包工作，普通节点为待选节点、劣质节点则永远不会纳入普通节点列表。

信誉漏斗会将连接时长大于 1 天的普通节点加入到优质列表当中，参与打包工作。

如果优质节点未在当前打包时间内完成打包则将清空连接时长，并将其移至普通节点列表，也就是说，未能及时进行打包并广播的节点会有一天的等待时间作为惩罚。

优质节点进行广播时，数据校验不通过的节点将会被加入到劣质节点中，不可参与继续挖矿工作。

## ● Demos 工作流程

每次有交易产生，Demos 网络就会进行如下共识过程：

1）每个满足连接时长大于 1 的节点都将列入优质节点名单，并进行编号。每个优质节点都将接收到不断从网络上发送过来的交易，通过与本地账本数据验证后，不合法的交易直接丢弃，合法的交易将汇总成交易候选集（candidate set）；

2）每隔一段时间进行一次打包节点的抽取，矿工在抽中打包时必须在产块间隔上限内产块，如无法及时产块，则会选择下一个矿工进行产块，并将该节点移动至普通节点列表中；

3）打包节点的抽取是根据上一个区块的哈希决定的，如该节点没有响应，则由优质列表中另外节点担任打包工作；

4）产块间隔上限决定了矿工的响应时间，这就等同于算力。在 Demos 共识机制中，根据当前产块间隔上限，算出一周理想出块数量。当到达一周理想出块高度时，进行时间戳的对比，计算出实际消耗时间的平均值是否大于当前产块间隔上限，如果大于产块间隔上限，则增加产块间隔上限，反之减少。

下面描述了产块间隔上限：

A（产块时间上限），604800000(一周毫秒)，time1(起始时间戳)，time2(结束时间戳)

理想时间内出块数量：
604,800,000/A

实际出块时间隔：
(time2 - time1) /（604800000/A）

假如 A=100 毫秒：

（1573371246000-1572766446000）/（604800000/100）=100

新产块间隔=A*（(time2 - time1)/604800000）

4）如矿工广播时交易是非法交易，则将该节点移动至劣质节点，不再参与打包工；

5）参与区块打包工作的矿工会获得交易费用的奖励，获得特殊的产块奖励。

这么做可以将 Demos 拓展链的性能发挥至最大，并且合理评估了 Demos 整个拓展链的性能情况，做出最合理的性能调度规划，避免无理由的浪费资源。

Demos 共识机制是没有挖矿难度的。如果需要确保被选中时及时产块，主要考虑的是网络状况、硬盘容量等因素，而不是算力，这样才可以将节点提供的资源合理运用起来。

因为有这样的设计，在未来硬件和网络得到提升时，Demos 可以自我进行升级。

- **选择打包节点**

打包节点的选择，是通过目前优质列表中的节点数量去截取上一个区块哈希头的数值来选择的。比如目前的矿工数量是 100 个节点，则截取上一个区块哈希头的末尾 2 位。如果这个节点未响应，则由列表中该序列的下一个节点进行打包。

## （2）账户和以太坊智能合约机集成

OP_CREATE  将合约字节码指令构建脚本

OP_CALL  将执行合约数据指令构建脚本

OP_GRANT  将执行合约数据指令构建脚本

拓展链交易也是套用 UTXO 结构，使用输入输出构建，虽然加入了权限部分，但是整体结构还是以 UTXO 为主。

### 部署合约

调用地址的 nonce 值构建输入的 outpoint，输出为合约地址，使用 OP_CREATE 指令和合约字节码构建脚本。

### 调用合约

调用地址的 nonce 值构建输入的 outpoint，输出为合约地址，使用 OP_CALL 指令和合约接口数据构建脚本。

### 合约监听转账

拓展链矿工打包新区块时会处理基础链上最新区块的交易，交易输出存在合约，则处理合约监听接口调用。

### 合约调用基础链转账

需要设置合约的 active 权限为 dosd.code，拓展链账户模型加入 EOS 的权限分配。

合约调用基础链转账接口来实现基础链转账，但合约调用接口，虚拟机将生成内部交易，由 dosd.code 权限来签名，生成成功并加入基础链交易池之后，合约调用才算成功，否则合约调用失败。

以太坊底层通过 EVM 模块支持合约的执行与调用，调用时根据合约地址获取到代码，生成环境后载入到 EVM 中运行。通常智能合约的开发流程是用 solidlity 编写逻辑代码，再通过编译器编译元数据，最后再发布到以太坊上。

因为 Demos 基础链是 UTXO 模型，想要和账户模型打通非常困难，我们在这里加入了类似 EOS 的账户权限体系，用户授权给 dosd.code， 通过 dosd.code 将合约中需要调用的基础链转账放入内存池中。

拓展链矿工先收到一笔内部交易，这笔内部交易调用了基础链转账，这时候拓展链矿工先对其校验，验证这笔交易是否合法。校验通过后广播并且将这笔交易发送给 Post office module，Post office module 处理好交易后，再次放入内存池，基础链矿工在接到这笔交易后会反向对拓展链上的这笔交易做校验。这里我们打了个时间差，拓展链上的交易速度远超于基础链，所以这个交易如果是合法的，它将很快被确认，也就是说，基础链矿工如果要校验这笔交易，只需要查看拓展链上的这笔交易是否真实存在即可，这样做既确保了安全性的，又兼顾了去中心化。

## （3）新增交易类型

部署新的智能合约的参考代码如下：

```
1: the version of the VM
[Contract EVM bytecode]
OP_CREATE
```

向链上合约发送资金的脚本为：

```
1: the version of the VM
[Data to send to the contract]
[ripemd-160 hash of contract transaction id]
OP_CALL
```

账号授权
1：the version of the VM
[grant name bytecode]
[grant value bytecode]
OP_GRANT

## （4）合约类型

为了满足不同的需要，比如拓展链 Gas 费用的抵押和赎回等，我们设置了三个级别的合约类型。

### system-level（系统级别）合约

这个类型的合约需要进行社区投票来添加和删除，对应的会有版本升级。

系统级别的合约是不需要缴纳 Gas 费用的，这个级别的合约主要是负责一些系统工作，比如 Gas 抵押的余额查询等。如果社区对于拓展链系统级别合约有新的需求或者好的提议，将进行投票并且由 Demos 开发团队进行开发维护。

该级别的合约伴随黑洞地址使用，没有任何第三方能在不符合运行逻辑的情况下私自盗取账户资产。

如果 system 合约是由第三方公链发起的，比如 BTC 的 Demos 拓展链 system 合约，用户可以将 BTC 转入该合约中，进行使用，可以在去中心交易所交易，也可以用来支付购物、游戏等费用。原理和 Demos 基础链转到 Demos 拓展链一样，第三方公链可以在该合约中设置 Gas 的兑换比例，Demos 拓展链上支持使用第三方公链代币支付 Gas 费用。

每个做基础链和拓展链之间交互的的 system 合约都必须要设置一个手续费用，用来奖励矿工。

每次进行基础链和拓展链转账时，将会扣除千分之二的手续费，全部用来奖励矿工，可以在 Demos 源码中看到。这个千分之二会随着业务量的增加一直持续扩大，它和 PoW 挖矿是反比，PoW 随着减产奖励会越来越少。

通常我们设置为一年周期发放奖励。随着业务量增加，每笔矿工手续费都会增加。按照当前收取到的手续费/365 天产块数量奖励给每个矿工。在 Demos 良好发展的前提下，这个奖励会越来越多，而收取的每笔手续费是恒定的。

### Ark- level（方舟级别）合约

这个级别的智能合约，是完全向用户免费提供的智能合约，Gas 完全由合约开

发者账户进行支付。Ark 级别的智能合约是 Demeos 团队设计之初就一直坚持的智能合约类型，也是我们最为推崇的智能合约类型。

### Base-level（基础级别）合约

在合约类型中，我们也提供了用户付费的合约类型，这个类型的合约是为了保证合约方在受到流量攻击的时候可以快速切换进行防御。

如果一个 Ark- level 合约受到流量攻击，Ark-level 合约方可以重新部署合约，将合约类型转变成 Base-level 来应对流量攻击。

我们非常建议合约开发者试用 Ark- level 的智能合约，它极大的降低了使用者的上手门槛，对于用户来说非常友好。

## （5）Gas 模型

Gas 翻译成中文就是“燃气”，它支撑起 Demos 网络生态系统的正常运行。Gas 用来衡量执行某些动作需要多少“工作量”，这些“工作量”就是为了执行该动作支付给网络的费用额。

通俗理解，Gas 是给矿工的佣金，并且是以 DOS 支付。无论是交易、执行智能合约并启动 DApp，还是支付数据存储费用，都需要用到 Gas。

Demos 还支持其他第三方公链代币抵扣 Gas 费用，在非 system 级别合约中可以设置每次抵扣的代币种类，如果该币种是 system 级别合约支持的币种，将调用成功。

Demos 的 Gas 消耗模型是在以太坊的基础上添加不同的支付方案，一种是用户支付，一种是合约方支付。

标准的转账交易的基本费用为 21000Gas。

附带 data 的字节数长度所消耗的 Gas，如附带了 0x4920676f74206f6e6520626 16e616e61（对应“I got one banana”的十六进制）这个消息，长度为 16 个字节，需要消费 16*68 个 Gas（每个非 0 字节消耗 68Gas，16 个字节就 16*68）。

总消耗的 Gas 就是 21000+16*68=22088Gas。

Demos 中 Gas 是需要保证绝对充足的，所以我们使用抵押的方式来进行预先充值。每次运行合约所消耗的 Gas 会换算成 DOS 进行扣除，并奖励给矿工。

Base-level 的合约需要用户预先将 DOS 抵押到拓展链的系统级别 Gas 合约中，

在合约监听到用户转账后会记录用户的抵押额度，用于之后智能合约调用的开销。

用户也可以通过系统合约回退 Gas，系统合约将会把剩余部分的 DOS 全部退还到用户账户上。这里我们将智能合约的接入成本降低，任何第三方钱包都可以接入 Demos 钱包库，调用智能合约完成 Gas 的抵押和回退。

Ark- level 的合约需要开发商预先将 DOS 抵押到 Gas 合约中去，每一次用户调用合约都将在 Ark- level 的合约账户上扣除 DOS。用户不需要支付 Gas，甚至不需要持有 DOS。

依据此，开发商可以完全独立地开发自己的 DAPP，并且依照自己的运行逻辑去对 DAPP 进行构架。比如，开发商发行了 A123 代币，这个合约通过监听 ETH 代币的转账进行 A123 代币的发放兑换。这个过程中 Demos 只是作为代币的基础创建平台，而没有要求 A123 的代币持有者持有 DOS，用户可以无感体验智能合约。

用户只需要符合开发商的 DAPP 运行逻辑去使用 DAPP 即可，这个构架将大大降低使用者门槛，并衍生出更广泛的 DAPP 应用场景。

用户调用智能合约时，因 Gas 不足而调用失败是不会扣除 DOS 的。

## 指令 Gas 费用清单:

```
GasLimitBoundDivisor uint64 = 1024    // The bound divisor of the Gas limit, used in update calculations.
MinGasLimit          uint64 = 5000    // Minimum the Gas limit may ever be.
GenesisGasLimit      uint64 = 4712388 // Gas limit of the Genesis block.
ExpByteGas              uint64 = 10     // Times ceil(log256(exponent)) for the EXP instruction.
SloadGas              uint64 = 50    // Multiplied by the number of 32-byte words that are copied (round up) for any *COPY operation and added.
CallValueTransferGas   uint64 = 9000   // Paid for CALL when the value transfer is non-zero.
CallNewAccountGas      uint64 = 25000 // Paid for CALL when the destination address didn't exist prior.
TxGas                     uint64 = 21000 // Per transaction not creating a contract. NOTE: Not payable on data of calls between transactions.
TxGasContractCreation uint64 = 53000 // Per transaction that creates a contract. NOTE: Not payable on data of calls between transactions.
TxDataZeroGas           uint64 = 4      // Per byte of data attached to a transaction that equals zero. NOTE: Not payable on data of calls between transactions.
```

LogDataGas uint64 = 8 // Per byte in a LOG* operation's data.
Sha3Gas uint64 = 30 // Once per SHA3 operation.
Sha3WordGas uint64 = 6 // Once per word of the SHA3 operation's data.
SstoreSetGas uint64 = 20000 // Once per SLOAD operation.
SstoreResetGas uint64 = 5000 // Once per SSTORE operation if the zeroness changes from zero.
SstoreClearGas uint64 = 5000 // Once per SSTORE operation if the zeroness doesn't change.
SstoreRefundGas uint64 = 15000 // Once per SSTORE operation if the zeroness changes to zero.
NetSstoreNoopGas uint64 = 200 // Once per SSTORE operation if the value doesn't change.
NetSstoreInitGas uint64 = 20000 // Once per SSTORE operation from clean zero.
NetSstoreCleanGas uint64 = 5000 // Once per SSTORE operation from clean non-zero.
NetSstoreDirtyGas uint64 = 200 // Once per SSTORE operation from dirty.
SstoreSentryGasEIP2200 uint64 = 2300 // Minimum Gas required to be present for an SSTORE call, not consumed
SstoreNoopGasEIP2200 uint64 = 800 // Once per SSTORE operation if the value doesn't change.
SstoreDirtyGasEIP2200 uint64 = 800 // Once per SSTORE operation if a dirty value is changed.
SstoreInitGasEIP2200 uint64 = 20000 // Once per SSTORE operation from clean zero to non-zero
SstoreCleanGasEIP2200 uint64 = 5000 // Once per SSTORE operation from clean non-zero to something else
JumpdestGas uint64 = 1 // Once per JUMPDEST operation.
CreateDataGas uint64 = 200 //
ExpGas uint64 = 10 // Once per EXP instruction
LogGas uint64 = 375 // Per LOG* operation.
CopyGas uint64 = 3 //
TierStepGas uint64 = 0 // Once per operation, for a selection of them.
LogTopicGas uint64 = 375 // Multiplied by the * of the LOG*, per LOG transaction. e.g. LOG0 incurs 0 * c_txLogTopicGas, LOG4 incurs 4 * c_txLogTopicGas.
CreateGas uint64 = 32000 // Once per CREATE operation & contract-creation transaction.
Create2Gas uint64 = 32000 // Once per CREATE2 operation
SelfdestructRefundGas uint64 = 24000 // Refunded following a selfdestruct operation.
MemoryGas uint64 = 3 // Times the address of the (highest referenced byte in memory + 1). NOTE: referencing happens on read,

```
write and in
TxDataNonZeroGasFrontier uint64 = 68    // Per byte of data attached to a transaction that is not equal to zero. NOTE: Not payable on data of calls between
TxDataNonZeroGasEIP2028  uint64 = 16     // Per byte of non zero data attached to a transaction after EIP 2028 (part in Istanbul)
CallGasFrontier                  uint64 = 40  // Once per CALL operation & message call transaction.
CallGasEIP150                    uint64 = 700 // Static portion of Gas for CALL-derivates after EIP 150 (Tangerine)
BalanceGasFrontier               uint64 = 20  // The cost of a BALANCE operation
BalanceGasEIP150                 uint64 = 400 // The cost of a BALANCE operation after Tangerine
BalanceGasEIP1884                uint64 = 700 // The cost of a BALANCE operation after EIP 1884 (part of Istanbul)
ExtcodeSizeGasFrontier       uint64 = 20  // Cost of EXTCODESIZE before EIP 150 (Tangerine)
ExtcodeSizeGasEIP150         uint64 = 700 // Cost of EXTCODESIZE after EIP 150 (Tangerine)
SloadGasFrontier             uint64 = 50
SloadGasEIP150               uint64 = 200
SloadGasEIP1884              uint64 = 800  // Cost of SLOAD after EIP 1884 (part of Istanbul)
ExtcodeHashGasConstantinople  uint64 = 400    // Cost of EXTCODEHASH (introduced in Constantinople)
ExtcodeHashGasEIP1884        uint64 = 700  // Cost of EXTCODEHASH after EIP 1884 (part in Istanbul)
SelfdestructGasEIP150        uint64 = 5000 // Cost of SELFDESTRUCT post EIP 150 (Tangerine)
// CreateBySelfdestructGas is used when the refunded account is one that does
CreateBySelfdestructGas uint64 = 25000
EcrecoverGas          uint64 = 3000 // Elliptic curve sender recovery Gas price
Sha256BaseGas       uint64 = 60   // Base price for a SHA256 operation
Sha256PerWordGas    uint64 = 12   // Per-word price for a SHA256 operation
Ripemd160BaseGas    uint64 = 600  // Base price for a RIPEMD160 operation
Ripemd160PerWordGas  uint64 = 120   // Per-word price for a RIPEMD160 operation
IdentityBaseGas     uint64 = 15   // Base price for a data copy operation
IdentityPerWordGas   uint64 = 3      // Per-work price for a data copy operation
Bn256AddGasByzantium              uint64 = 500    // Byzantium Gas needed
```

```
for an elliptic curve addition
Bn256AddGasIstanbul                 uint64 = 150     // Gas needed for an
elliptic curve addition
Bn256ScalarMulGasByzantium         uint64 = 40000   // Byzantium Gas needed
for an elliptic curve scalar multiplication
Bn256ScalarMulGasIstanbul           uint64 = 6000    // Gas needed for an
elliptic curve scalar multiplication
Bn256PairingBaseGasByzantium       uint64 = 100000 // Byzantium base price
for an elliptic curve pairing check
Bn256PairingBaseGasIstanbul         uint64 = 45000   // Base price for an
elliptic curve pairing check
Bn256PairingPerPointGasByzantium uint64 = 80000   // Byzantium per-point
price for an elliptic curve pairing check
Bn256PairingPerPointGasIstanbul  uint64 = 34000  // Per-point price for an
elliptic curve pairing check
```

## （6）抵押退还

已抵押未花销的 DOS，可以通过调用 Gas 合约来完成 DOS 的退还。Gas 合约提供退还接口，直接调用即可，退还地址为抵押地址，DOS 会在扣除已使用的 Gas 后原路返回。

## （7）版本控制

和基础连相同，Demos 拓展链中也加入了版本控制，以此来做向后拓展。版本号和基础链同步，每一个基础链版本更新，拓展链如果有改动并且被社区认可，则会同步更新，同样依赖于社区自治系统。

# 三、兼容

Demos 的兼容体现在两方面：一方面是对于传统 PoW 公链的兼容，另一方面是对于企业、政府联盟链的兼容。

为此 Demos 中有两套兼容机制：

一种是强兼容，通过授权一种特殊签名来和基础链交互，这里需要使用到系统级别的合约，双方都要做兼容方面的调整；

另一种则是弱兼容，以原子交换的方法做到跨链交互，这种方法不需要去做双

向牟定。

Demos 基础链与拓展链使用的是第一种交互方式，对于去中心化的公链而言，这样的方式显然更好，但是对于有行业机密的联盟链和私有链来说，则推荐使用弱兼容的交互。

## 四、基础链和拓展链的依赖关系

Demos 基础链和拓展链之间的关系是密不可分的：基础链记录了拓展链每一个区块的交易 hash，用来保证两条链的一致性和加强不可篡改性。

双链之间的交互部分，由不同分工的矿工完成数据校验，保证双链交互的去中心化和安全。

Demos 的基础是 PoW+PoS 机制，这样的机制带来的好处是社区高度自治，并且保护 Demos 的完整性。而 Demos 机制又很好地将多节点、高性能的需要引导出来。

DOS 作为双链的流通资产，既是链上资产储存的关键，又是智能合约使用的必要储备资产。

这样的设计的一个优势是，在基础链因为算力增加导致节点数量减少时，由拓展链矿工来稳定节点数量并且得到整体的节点数量稳步增加，将去中心化贯彻到底。

要打包拓展链区块就必须要同步基础链数据，无形中就增加了基础链区块的数据备份。而基础链矿工在打包拓展链合约发起的主交易时，也需要对拓展链区块进行校验，这时候同样需要基础链矿工对拓展链区块进行同步。

Demos 需要最为直观地将 DOS 作用于基础链和拓展链上，以此真正实现以应用落地为基础的区块链价值升级。

我们设计了全新的共识机制和合约体系来降低应用落地的门槛，基础链就像一个保险箱，牢牢的将用户资产保管好，而这些资产不仅仅使用在支付环节中，它还支撑起 Demos 整个应用生态的运行。

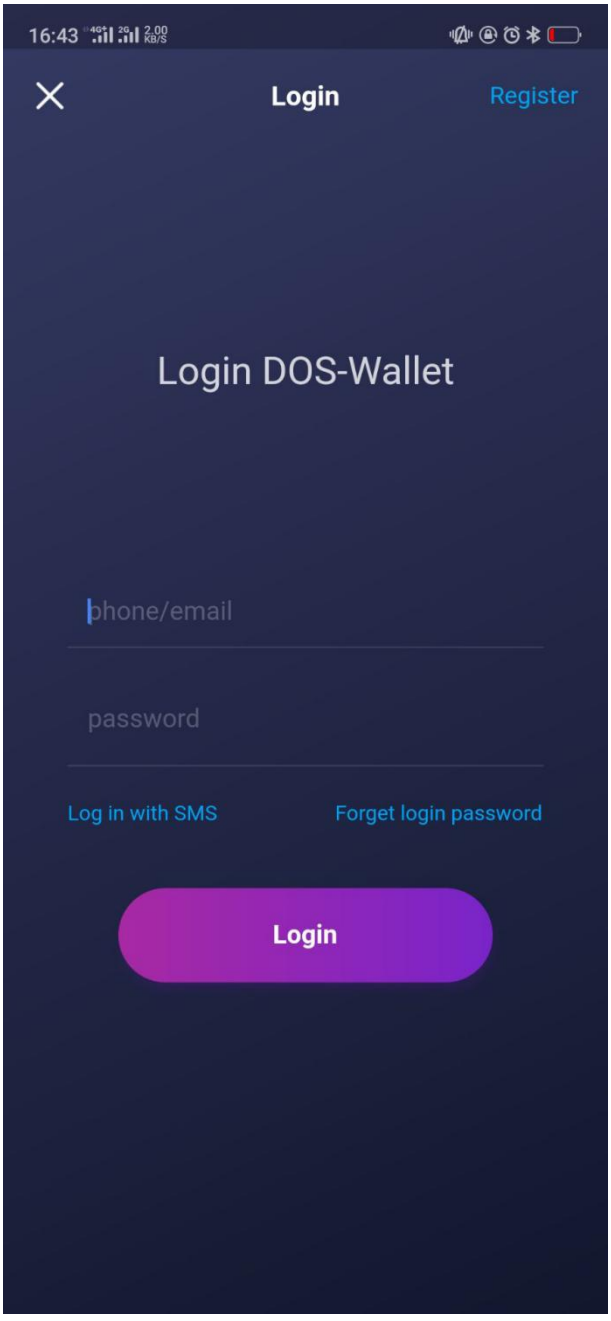
## 五、移动端支持

在白皮书中着重提到这个部分，因为我们不仅考虑到了公链结构，更是以应用落地为核心，去开发相关产品。虽然这个过程很难，但是我们依然坚持要这么做，可以负责任的说，Demos 就是为应用落地而生。

我们比以往任何一个以 PoW+PoS 共识机制为基础的公链都要注重移动端的支持，为此我们设计并开发出一整套移动端钱包库，以满足投票提案、投票挖矿、智能合约调用等功能。

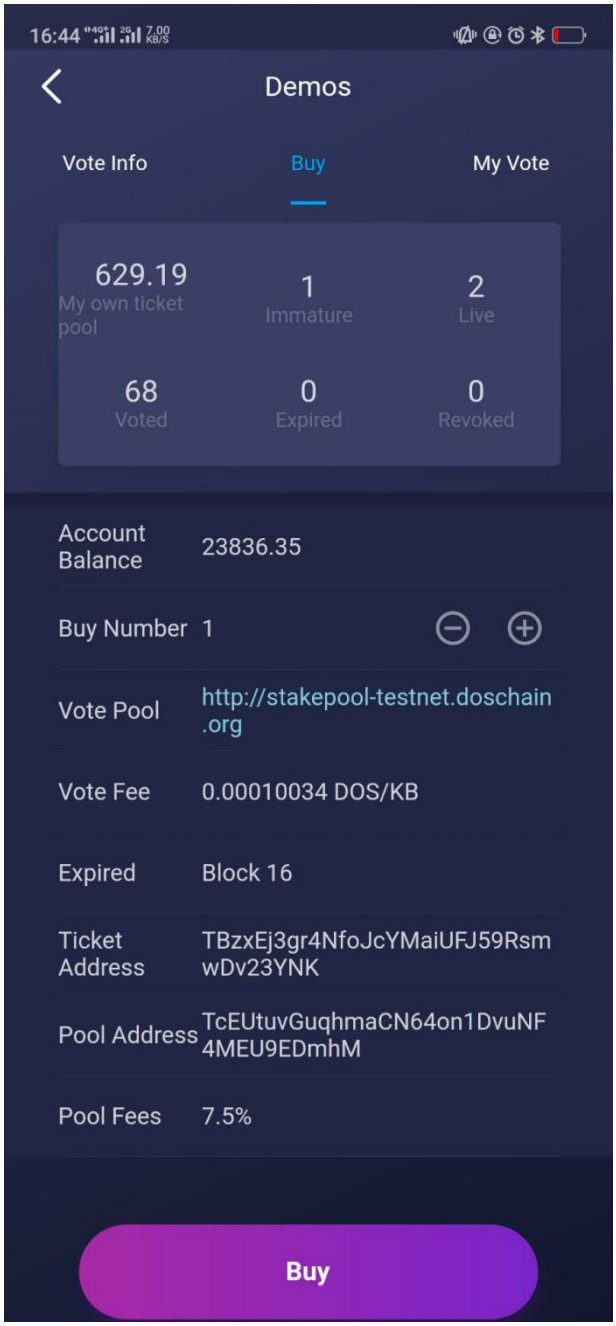
我们将所有的移动端操作都设计得简单且安全，下面是我们的移动端钱包测试网版本，我们会在官网上开放水龙头让用户进行体验，并开源整个项目。

我们希望反哺 DCR 这样开创 PoW+PoS 共识机制的优秀区块链项目，DCR 社区可以借助我们开源的钱包完成整个项目用户体验方面的巨大提升。



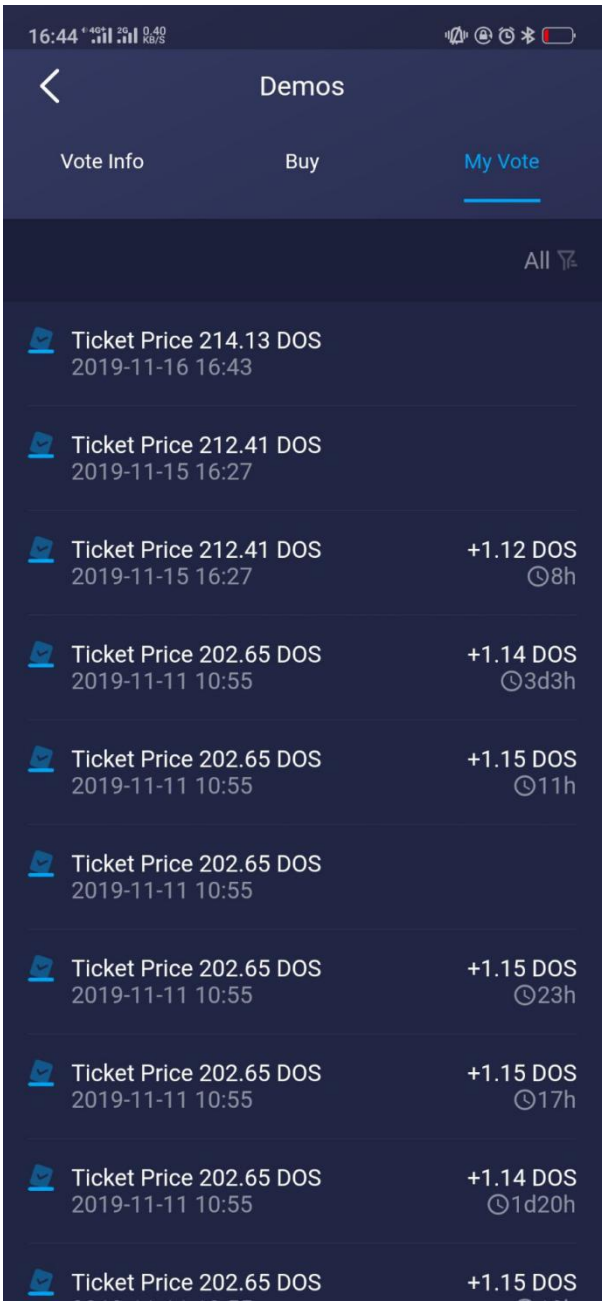

**云账户注册**

PoS 矿池和提案系统的统一注册，相比其他 PoS+PoW 的项目需要单独注册，我们做出了很大的优化，这个账户并不会保留用户的私钥等信息。



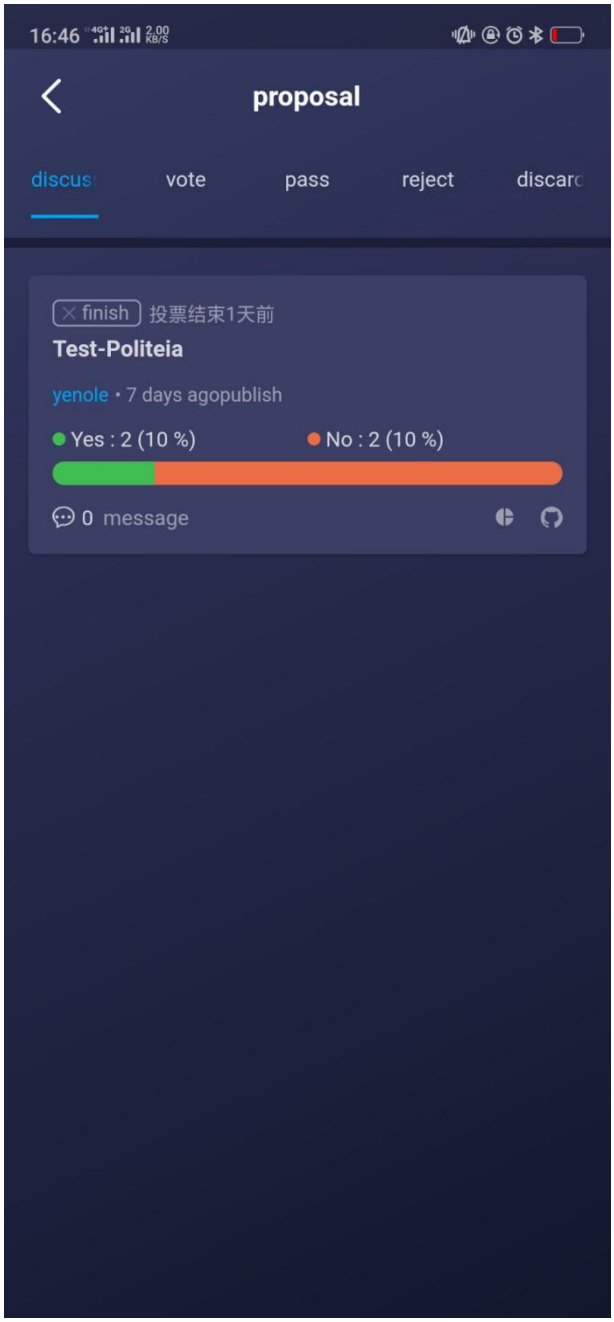

**移动端 PoS 购票**

用户只要注册了 Demos 的云账户就可以在移动端进行 PoS 挖矿，云账户会主动提供 APIkey，无需用户自行填写绑定，只需授权即可。




**PoS 票务清单**

PoS 挖矿的相关数据可以在移动端直接查看，用户每一个投票所得到的收益都能一目了然。




**移动端提案系统**

在此页面，用户可以直接对社区的发展进行干预。

其余各移动端支持，请下载测试网版本自行查看。

# 6. Demos 的阶段性任务

Demos 团队思考了中本聪的初衷等问题，得到以下结论：

1.区块链的未来一定是应用落地，比特币也是一种区块链的应用。

2.社区的完全自治是抵御硬分叉的唯一合理方式。

3.区块链项目不应该只做前瞻性的设计，也要有对应时代变化的机制。

4.要推动区块链产业发展，就必须要做出大胆的创新性尝试，前提是这样的尝试是合理的，并且具备以上三点。

为此我们设置了两个时代，来应对和期待区块链技术的未来。

# 一、基础时代

在这个阶段，Demos 会在主网上线后通过定期出售的形式将 DOS 投入市场。

我们不做私募，也不会在主网上线前以任何一种形式出售任何一枚 DOS，我们清楚私募对于开发团队的重要性，但是我们宁愿艰苦地走完开发路程，也不愿意让用户与开发团队共担风险。

Demos 会借助优秀的经济模型去推动整个 Demos 的发展进程，并试图将速度加快。

与交易所和钱包、第三方支付等平台打通，并推行 Demos 的经济模型使其稳定。

推动社区高度自治的普及，来达到共识的稳定与良性发展。

方舟级别的智能合约类型的推荐，建立方舟级别智能合约的使用习惯。

加强区块链应用商业化落地的建设，为企业乃至政府提供有效的区块链技术支持，我们欢迎更多的开发者加入到 Demos 开发者团队当中，社区自治基金会对开发者进行奖励。

Demos 社区的稳定建设，并为方舟时代的来临做准备。

# 二、方舟时代

在这个阶段，Demos 的主要工作重心放在与第三方 PoW 公链结合方面，我们已经构建足够稳定社区，并拥有覆盖全球的节点，使我们成为全球最大的去中心化公链。

我们会主动研发并推出第三方 PoW 公链的兼容开源版本，以此来帮助第三方公链完成公链升级，并且主动提供节点支持，加速第三方公链和 Demos 拓展链的融合。

并在这个阶段，推动 Demos 拓展链的 Demos 共识机制+PoS 共识落实。以此来赋予融合进来的 PoW 公链以权力。使其在融合当中获取到对于 Demos 拓展链的社区权力，让其真正成为 Demos 不可分割的成员。

在此阶段，会有很多公链与 Demos 拓展链形成 DNA 螺旋结构，围绕着拓展链的发展做出贡献。

Demos 开发团队会将社区基金账户永久关闭，并改变基础链 DOS 产出的分配机制，将 PoW 收益从 60%提升到 70%。

将社区基金账户中的所有资金全部空投给每一个 Demos 账户，以感谢每个人对于 Demos 的支持。

Demos 社区将不再以奖励驱动，所有人包括第三方公链、开发者、DAPP 项目方、普通用户都将呈现一种“人人为我、我为人人”的状态。

Demos 开发团队的使命也在这个时代的开始时完成，无须再持有社区基金账户。也许那个时候 Demos 开发团队已经为 Demos 奉献出了一生，甚至是好几代的 Demos 开发者也已经为其做出宝贵贡献，但是只要这一天最终来临，所有的付出都将是值得的!

战胜“不可能三角”的不是技术，而是人类本身。

# 7. 结论

Demos 提供了全新的智能合约解决方案，我们详细介绍了 PoW+PoS 共识机制如何使社区高度自治，并且提高攻击成本，此外我们还解释了 Demos 共识机制是如何帮助 Demos 实现自我升级，并保持工作效率的。UTXO 模型和账户授权体系的结合，让 Demos 的跨链交互安全且去中心化。

多种级别的智能合约可满足不同开发者的需求，提供全新的运营思路和可能性。Demos 非常重视移动端的支持，并在项目初期就将这个工作提上日程，同步进行开发。

目前，在国家政府的大力推动下，智能合约的普及与应用已推上日程，区块链应用的大规模落地即将到来。

Demos 以去中心化、安全、高性能为项目基础，移动端的便捷、安全、多功能为核心，必将把智能合约的应用落地推向一个前所未有的高度，并为区块链技术的普及起到至关重要的作用。