Ambr——无限可扩展性的异步智能合约与跨链平台

Draft 0.6.1

Ambr Team
Support@ambr.org

概要: Ambr 是一个拥有无限可扩展性的异步智能合约与跨链平台。该平台在保证安全性的前提下，具有高灵活、实时、无限可扩展性、异步高并发等特性。Ambr 底层基于有向无环图DAG数据结构和Casper协议。每个地址都有自己的区块链即账户链，在无数账户链并行的情况下，增加不影响性能的验证链来进行验证与审核，完成 Casper 共识与手续费的分发。该方式使得 Ambr 兼具强大的数据处理能力与系统的绝对安全性。Ambr 的 AVM 虚拟机部分兼容了一些 EVM 的特性，并在 ETH 智能合约的基础上进行拓展，使其更加高效实用。并且完美解决了 DAG 结合智能合约的一致性问题。Ambr 的账户链与中继链模型先天完美契合，实现了 Ambr 具有同其他币种间进行安全快速交易的功能，该跨链技术使 Ambr 不但能实现跨链转账，同时也能实现链内多币种转账。除此之外，Ambr 的脱链协议使得 Ambr 拥有无限的交易处理能力，与闪电网络与普通状态通道不同的是，引入观察者角色来惩罚作假人与保证高频次脱链交易时双方的利益。

I. 引言

传统互联网中电子货币的中心化交易处理模型已经体现出诸多问题：高维护成本、可逆转的信息处理、无隐私保护性、效率低下等。所以分布式架构在对传统交易模型的处理上等有了极大提升。在 1998 年，戴伟(Wei Dai)提出的 b-money 虽较好实现了这种思想，然而却仍未完美解决共识问题。Hal Finney 在 2005 年提出了工作量证明（PoW）的概念，随着加密技术的完善，基于 PoW 机制，中本聪（Satoshi Nakamoto）在 2009 年实现了第一种被全球广泛认可的数字加密货币—比特币。

比特币通过链式结构和 PoW 的概念，保证了货币的安全性。随着互联网的发展，比特币在商业活动中得到了广泛的应用。然而随着交易数量的增长，比特币的链式结构已经不能满足爆炸性增长的交易量，交易的实效性不会再得到保证。之后，虽然出现了许多不同共识机制的数字货币，例如 PoS，DPoS 等，但在链式结构下始终都存在各种不同的问题。基于此，本文将介绍一种基于新型数据结构—DAG 和一种新型的共识协议—Casper，在保证安全性的前提下实现实时无延迟的加密货币—Ambr。

II. 背景

区块链技术是过去几年来信息技术最重大的发展之一，被认为是继大型机、个人电脑、互联网之后计算模式的颠覆式创新。联合国、国际货币基金组织，以及世界各地等国家都对区块链的发展给予高度关注，积极探索推动区块链的应用。

区块链的主要特征有:

1.去中心化

由于使用分布式核算和存储，无中心化的硬件或管理机构，任意节点的权利和义务都是均等的，系统中的数据块由整个系统中具有维护功能的节点来共同维护。

2.开放性

系统是开放的，除了交易各方的私有信息被加密外，区块链的数据对所有人公开，任何人都可以通过公开的接口查询区块链数据和开发相关应用，因此整个系统信息高度透明。

3.自治性

区块链采用基于协商一致的规范和协议（比如一套公开透明的算法）使得整个系统中的所有节点能够在去信任的环境自由安全的交换数据，任何人为的干预不起作用。

4.信息不可篡改

一旦信息经过验证并添加至区块链，就会永久的存储起来，除非能够同时控制住系统中超过 51%的节点，否则单个节点上对数据库的修改是无效的，因此区块链的数据稳定性和可靠性极高。

5.匿名性

由于节点之间的交换遵循固定的算法，区块链中的程序规则会自行判断活动是否有效，因此其数据交互是无需信任的。因此交易对手无须通过公开身份的方式让对方对自己产生信任，信用如何累积对信用的累积非常有帮助。

因此，Ambr 旨在创建一套具有新的体系和规则的分布式系统。以上特点是区块链技术必须具有的特点。除此之外，Ambr 会在这些基础上引入大规模的创新特点。

III.设想与展望

目的

随着人们对于信息共享的需求越来越高，业界倾向于越来越需要一个安全的、去中心化的平台，比特币和以太坊应运而生，但是链式结构的设计使得他们性能不足的缺点已经不能满足人们日益增长的需求。

Ambr 被设计为一个通用的去中心化应用平台，采用 DAG 的账本结构，账本中的交易按账户分组。通过分层共识算法 ，将交易的写入和确认解耦，并保障系统的高性能和扩展性。Ambr 的虚拟机部分使用以太坊智能合约语言并在 Solidity 基础上并进行了适当扩展，提供了更为强大的描述能力。另外，全新的基于消息驱动的异步架构，极大提高了系统的吞吐率和扩展性。Ambr 内置了数字代币，为了更好的服务于用户，营造一个健康的、透明的社区环境。本文档对 Ambr 的架构准则和系统实现细节进行了详细的描述。

设想

Ambr 被定义为实现以下需求

- 完整的不可篡改的基于 DAG 的智能合约系统。
- 能并行正确独立执行智能合约。
- 高度的网络自适应性。
- 量子安全
- 快速确认的共识系统
- 高性能，存储友好的系统
- 绿色的共识算法/验证协议。

PoW 共识算法具有很好的安全性，在比特币和以太坊中得到了充分的验证。但这个算法在求解数学难题需要消耗大量计算资源，造成能源浪费；目前，以太坊整体的 TPS 只有 15 左右，完全无法满足去中心化应用的需求

构建高质量的软件是非常具有挑战性的，而且由此带来后续维护、软件升级等问题。

Ambr 项目将严格按照模块化的思路来开发，引出抽象层，严格定义系统边界，同时随时和社区保持同步，以实现一个螺旋状，持续迭代的良好开发模式。来支持开发高质量的软件系统。

为了达到上述要求，我们的设计致力于以下几个方面：

- 一个良好分层的软件体系架构。
- 一个完整的模拟测试系统和数学模型来随时验证。
- 使用函数式编程范式结合面向对象编程范式，使其一方面更更自然地适应分布式和并行处理，另一方面对开发人员友好，以增加可维护性和可阅读性
- 良好的代码复查机制。
- 高性能、鲁棒性、可扩展性。

展望

Ambr 将自己定位在未来区块链技术的领导者，将继续保持在技术上的持续投资，Ambr 未来将会向社区注入总计 30%的代币，以打造一个强大、团结、积极向上的开发者社区。

IV. 共识

区块链系统是一种典型的分布式系统。在完全去中心化的分布式系统中，共识相当于整套系统的基础设施，很大程度上决定了系统的安全性、效率与资源消耗程度。

A. 共识机制

关于共识机制，主流的共识机制有四种:PoW，PoS，DPoS 和 DAG，下文将进行简要分析：

1) PoW:工作量证明，这是一种按照矿机算力来分布权利的去中心化系统。其缺点非常明显:

a）消耗社会资源，算力所计算的结果无实用性导向，且消耗的资源庞大。

b）效率低下，出块速率与安全性成反比，已经无法满足基于区块链的去中心化应用开发。

c）已逐渐失去去中心化的特性，算力资源因矿池而逐渐集中。

d）算力的生态聚拢效应对新型公链不友好，基于 PoW 的公链在早期难以获得大量算力维持安全性。

2)PoS:权益证明机制，为了应对 PoW 所面对的问题，PoS 使用使用了股份来对系统进行去中心化，并假设拥有越多股份越希望系统朝好的方向发展。

特点：

a）节省社会资源，不需要对算力的依赖，杜绝了对资源的浪费。

b)权利与利益相关性较强，拥有越大权利的人同时也拥有着更多的股份。

c)权益证明的转账速度依然较慢。由于每个人都有投票权导致整个的确认速度依然较慢。

d)作恶成本低，虽然利益相关性较强，但是由于缺乏惩罚机制，导致作恶成本并没有那么高。

3）DPoS：委托权益证明机制。为了追求效率，DPoS 牺牲了去中心化的特性，由于超级节点的处理能力非常强，使得整个系统的吞吐量和确认效率非常高，然而其中心化的缺点十分明显。因传播渠道的不同，超级节点的竞选会倒向于市场传播能强的一方。普通节点无记账权，被迫被超级节点所代表。

权利和利益关系的不对等，决策以少部分个人的意志为转移。

4）DAG：有向无环图。由于传统区块链项目基于单链的结构导致吞吐量较低。DAG 将区块链带入了异步时代，其有以下几个特点：

a)吞吐量巨大，远远超过 DPoS。

b）结构复杂容易出错，当交易量较大时，由于链的结构较为复杂，很有可能导致对历史的共识不一致，或者系统无法对交易进行序列化，使得交易永远无法被确认。导致有些项目必须借助于中心化的各种组织才能保证项目稳定运行，这使得对项目变相的进行了中心化。

c）无手续费，表层理解上这是个优点，但实为缺点，无手续费就有可能导致攻击者发送大量的无成本交易对资源进行占用，却几乎没有任何成本。而 DAG 中无手续费多数受制于其数据结构与共识本身的缺点，例如一致性问题。无手续费、不需要大规模挖矿虽导致系统不会通货膨胀，但是同时也导致了维护整个系统正常运行

的节点失去了利益。

d)生态建设难，基于上述可得，既能够无成本地滥用系统，又没有利益驱使人们去维护系统，那么生态系统将会畸形。

B. 共识设计原则

安全性：所有区块链系统的第一特性，Ambr 希望设计出一套绝对安全、完全去中心化的共识系统。

性能：在保证安全性的前提下，拥有超高吞吐量与快速确认的共识。

C. 系统总览

不同于传统的区块链共识，Ambr 使用 Casper 与 DAG 作为共识的基础，使得我们既拥有 Casper 的安全和去中心化的特性，又拥有 DAG 的超高吞吐量的特点。

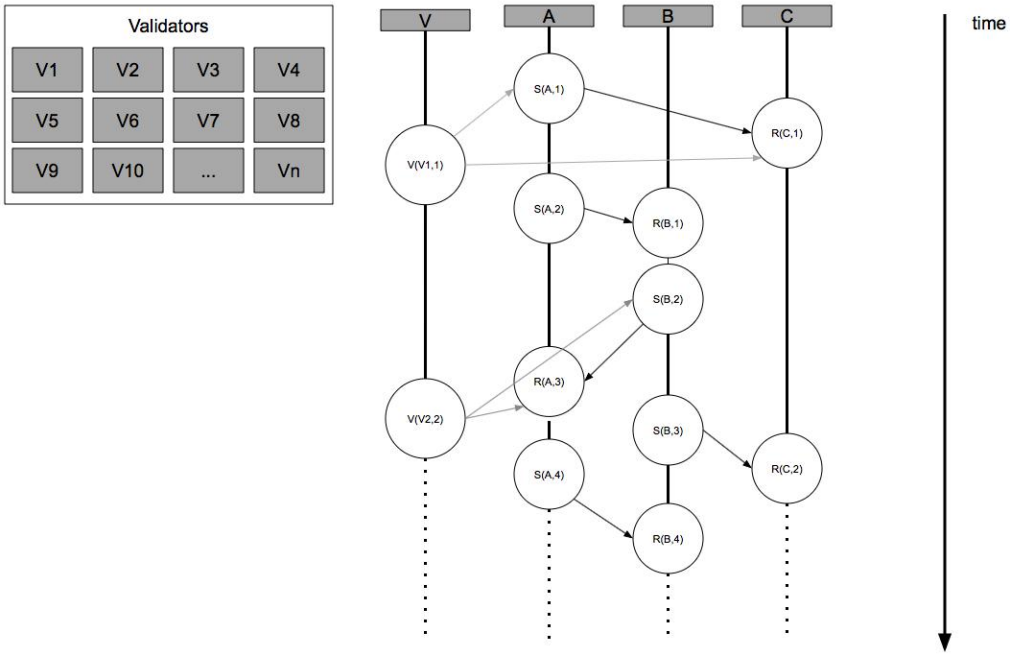

图 1 共识系统总览

整个系统的构成如上图所示，由验证者（V1~Vn)，账户链(A，B，C...)，验证链 V，交易单元（S，R)，验证单元(V1，V2...)组成。

D. 相关概念

账户链:每个账户均有一条属于自己的链，该链上所有数据均同账户相关，唯有账户的所有者能在该条链上添加数据。如图所示 S(A，n) U R(A，n)的交易单元组成了 A 的账户链。

交易单元:交易单元分为交易发送单元 S 与交易接收单元 R，一对 S，R 组成一

笔完整的交易，其中交易发送单元从发送方的余额中扣除金额，交易接收单元将交易发送单元中的发送金额加入到接收单元中。

交易发送单元

```
Unit{
  version:0000...0001，
  type:send，
  balance:9999，
  previous:DC32...1CD1，
  destination:zefd...sieokdf，
  public_key:12ea...df94，
  signature:84ec...edf6，
  hash:57da...96c2，
}
```

交易接收单元

```
Unit{
  version:0000...0001，
  balance:9999，
  type:receive，
  previous:DC32...1CD1，
  source:32C1...5DB4，
  public_key:12ea...df94，
  signature:84ec...edf6，
  hash:57da...96c2，
}
```

以上详细展示了交易的发送和接收单元的结构。其中

- version:版本号，用于未来升级协议，起到辨识作用。
- balance:当前账户余额，每个单元点都会包含该字段。
- type:交易类型
- previous：上一个当前账户的交易单元
- destination：类型为交易发送单元时该字段有效，代表要发送到的账户地址。
- source：类型为交易接收单元时该字段有效，表示与该交易接收单元所配对的交易发送单元。

- public_key，signature:对单元进行签名，保证该单元为账户拥有者所发送。
- hash:对整个交易单元进行 hash 运算，保障交易单元在网络传输的过程中不被修改。

E.验证链

为了对 DAG 进行序列化，以及使得 DAG 在末端形成不可修改的稳定点，我们设计了一条验证链辅助 DAG 进行序列化和达到稳定。

关于这条链的设计有几个基本的问题需要解决。

1、允许谁在验证链上发布单元？

由于见证链是维护整套系统的最重要组成部分之一，那么在验证链上发布单元的人应当同整套系统有较大的利益相关性,具有这样条件又许诺系统维护当中的人，我们称之为验证者。

2、什么时间在验证链上发布单元？

验证人一定是（至少大部分）为了维护整个系统稳定高效而存在的，那么验证者之间的关系应当是合作大于竞争，而不应由于争抢导致无畏的内耗。

3、如何保证对的人在对的时间发布对的节点？

我们不但应该奖励对系统提供了贡献的验证者，同时也要严惩并剔除对系统造成破坏的验证者，另外还需要轻度惩罚以及剔除懒惰或者由于能力不足无法对系统做出贡献的验证者。

4、这个验证链如此核心，如何保证去中心化。

基于以上三种问题，我们对验证链做出如下的设计目标：

1）验证者需要持有较多 token 以保证利益相关性，同时为了增强这种相关性，其所持有的 token 需要有一定的锁仓期。

2）验证单元应该有验证人轮流发布，而非竞争。

3）验证者需要提供保证金，当其对系统做成破坏时或者无作为时没收，削减其保证金的账户。为了鼓励验证者验证更多的交易，需要手续费作为对验证者验证更多交易的奖励。

4）验证者的集合是动态变化的，数目不定的，门槛较低的，以保证系统去中心化的特点。

缴纳保证金后的 2 个验证单元之后，自动加入验证者集合。验证者轮流出块，验证所有未验证的交易，并包含拥有未验证交易账户链的末端单元的哈希值，对其进行见证，并邀请所有验证者进行投票，若投票通过，则发布者获取手续费并与所有验证者平分验证费。若没获得足够的选票则跳过。若此时有验证者作恶，则罚没所有保证金。

下面定义对作恶以及惩罚进行定义：

1、拥有当前发布单元权的验证人没有履行义务，对不合法的交易进行了验证。

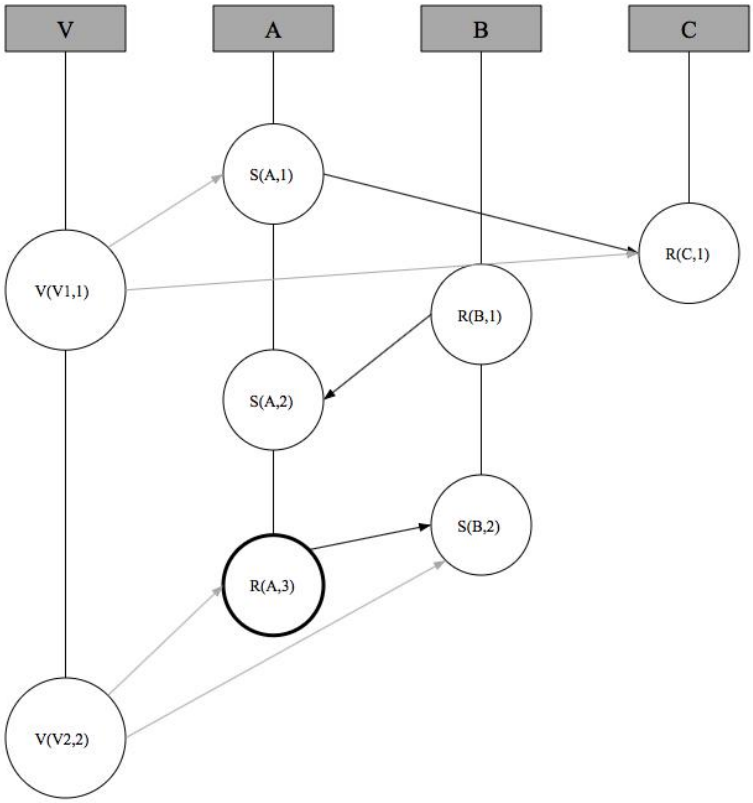

图 2 验证者作恶示意

如图所示,R(A,3)是一个不合法的单元点(例如 S(B,2)只给他转了 10 个 token，R(A，3)却说自己收到了 20 个 token)，但是验证人 V2 却认为 R(A，3)是合法的，进行了验证并发起投票。这种情况验证者 V2 的保证金将会由于其主动作恶被全部扣除。

2、拥有当前发布单元权的验证人连续发布了 2 个验证单元，或者在不该自己发布单元的时候，发布了单元。

假设在一个时间区间我们没收到当值验证者发布的单元，那么有两种可能，a）该验证者不作为，没有履行义务。b）由于网络延时，并没有主动作恶，对整个系统影响不大，那么我们对其进行较轻的出发，且如果连续发生，系统将自动将自动从验证者集合中剔除出验证人。

若在不合适的时间点发布了验证单元，表示其无视共识，恶意发布单元，将会对其进行罚没保证金的处罚。

F.验证单元

验证单元由验证者轮流发布，其主要作用如下：

1）对所有未经验证的交易进行合法性验证。

2）为上一个验证单元收集投票。

3）若上一个验证单元作恶，则发布惩罚。其结构示意如下：

```
Verification{
  version:0000...0001，
  previous:DC32...1CD1，
  heigth:999，
  verify:...，
  punishment:...，
  public_key:12ea...df94，
  signature:84ec...edf6，
  hash:57da...96c2，
}
```

其中

- version:表示版本号。
- previous:指向上一个验证单元。
- height:表示当前验证单元的高度。
- verify:验证过的交易和投票。
- punishment:对未通过投票的作恶验证者进行举证并惩罚。
- public_key，signature:对单元进行签名，保证该单元为验证者发送。
- hash:对整个单元进行 hash 运算，保障交易单元在网络传输的过程中不被修改。

G.相关逻辑

加入验证者集合与离开验证者集合

要成为验证者，首先应符合以下条件:

1)验证者需要具有一定的token，以确保Ambr系统与验证者的利益相关性足够。

2)验证者有能力维护 Ambr 的系统，这指的是拥有一台具有公网 IP 的高性能服务器，长期在线，以保障 Ambr 能够安全稳定运行。

而如何成为验证者就比较简单了，只需要在自己的账户链上添加一个请求加入验证者的单元即可（当然，这里需要交纳保证金），在该单元得到验证链确认后的两个验证单元后，被加入到验证者集合中。

具体的单元结构如下：

```
Unit{
```

```
  version:0000...0001，
  type:JoinValidators，
  balance:9999，
  previous:DC32...1CD1，
  public_key:12ea...df94，
  signature:84ec...edf6，
  hash:57da...96c2，
}
```

同成为验证者类似，离开验证者集合只需要在账户脸上添加一个离开验证者的信息即可，在被验证之后，经过两个验证点变从验证者集合中离开。

具体的单元结构如下：

```
Unit{
  version:0000...0001，
  type:LeaveValidators，
  balance:9999，
  previous:DC32...1CD1，
  public_key:12ea...df94，
  signature:84ec...edf6，
  hash:57da...96c2，
}
```

H.验证者相关责任

验证者的责任就是维护整个系统能够安全，稳定，高速的运行，以此来达到维护自己资产以及获得收益的目的。为达到此目的，验证者应当按照如下规则履行相关责任。

当验证者在任期内发现验证链上新的验证单元的时候，同步信息，并对验证单元的正确性进行判断后投出经过自己签名的一票。

当轮到本验证者进行发布验证单元时，需要做两件事情。

1、收集对上一个验证单元的投票，若上一个验证单元没收到足够的选票，且拥有上一个验证单元作恶证据，则收集证据，对上一个验证单元进行惩罚，将结果放入到当前的验证单元中，等待投票审核。

2、对当前收到的交易进行合法性判断，并对符合合法性要求的进行见证。

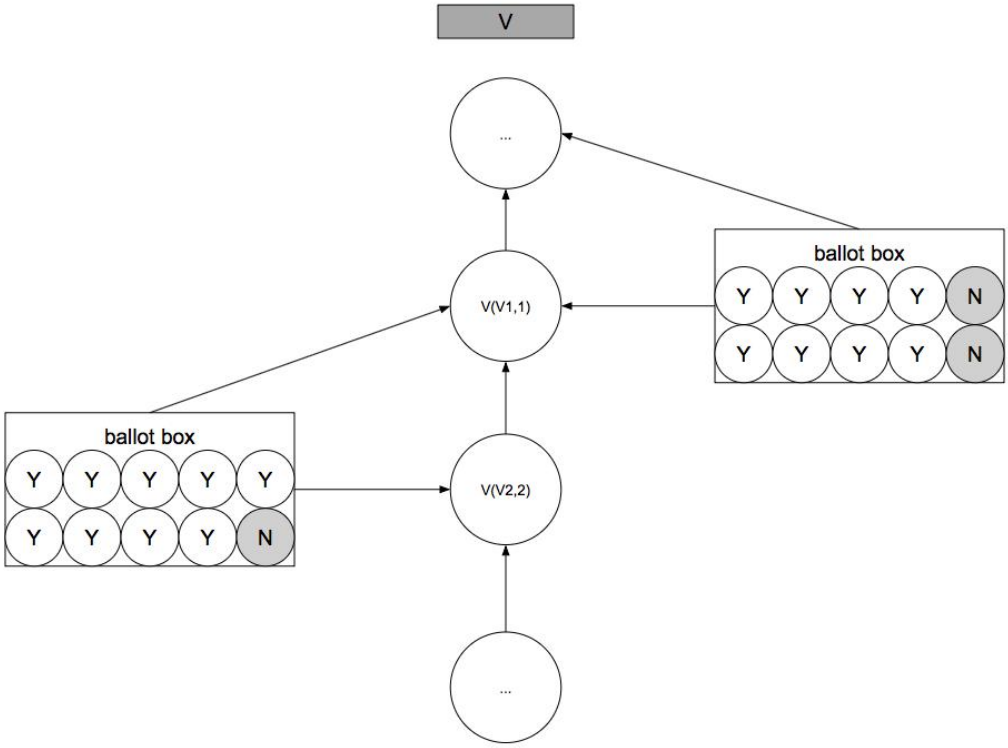

图 3 投票统计图

注意：这里需要强调一个事情，由于我们是去中心化系统，我们希望更多的人加入到我们的验证者，但是上图中可以看出，如果我们要对作恶节点进行惩罚，就必须对其作恶证据进行一定时间的保存，已被举证的时候使用。然而为了存储作恶而保存如此巨量的投票信息，对系统将会是巨大的负担。

这是一个矛盾，为此，我们鼓励所有的见证人动态的保存较新的投票信息，作为举证使用，而之前较早的投票信息则可以裁掉。

I. 维护 Ambr 的共识原则与原因：

原则 1：每产生一个验证单元，验证者应当获取和保证金成正比的收益。

原因 1：为了保证系统的安全，稳定和高效，需要更多的人和资金进入成为验证者。

原则 2：每个验证单元的收入应当被按比例分配。

原因 2：为了保证每个验证者之间的关系为合作而非竞争，从而节省社会资源，并且不会为了争夺而形成利益集团。

原则 3：越多的人和资金成为验证者会使原来的验证人获得更高的收益。也就是 p%的验证者参与了共识博弈，那么他们将得到 f(p)≤p%的收益。

原因 3：为了保证整个验证者集合是开发的，去中心的，而不是被少数利益集团控制的。

原则 4：主动作恶的人将受到严厉惩罚，无作为作恶的人也需要收到惩罚。

原因 4：主动作恶，例如不承认历史，对不合法的交易进行验证。无作为作恶，例如在该发布验证单元时不发布，在该投票时不投票。

原则 5：投票资金越多，所获得的收益也越多。也就是 p%的验证者参与了投票，那么他们将得到 f(p) ≤ p%的收益。若 100%的人参与投票，则收益为 100%。

原因 5：尽量保证所有的验证者参与投票，做为惩罚无作为的人的依据。

V. 智能合约

在一些高级语言系统（例如 java 和.net），为了提高平台兼容和移植能力， 并提高运行性能， 多通过引入一种中间编码,类汇编语言，然后运行时由虚拟机解释执行。 以太坊的 Solidity，RChain 的 RHO，ECO 使用的 WASM。

经过综合评测各种平台，Ambr 决定使用 Solidity 语言作为合约编写语言，它是一种简单的， 经过验证的表现力强大的语言， 并且由于在以太坊社区的使用，积累了大量的社区开发人员，配合由 C 语言开发的强大的 JIT 解释器解释执行，性能非常高。

并且可以利用社区现有的开发工具， 使合约编写变的更为简单方便

虚拟机

AVM（Ambr 虚拟机(Ambr Virtual Machine - AVM)基于栈结构，机器的字大小（以及堆栈中数据的大小）是 256 位。主要是便于执行 Keccak-256 哈希及椭圆曲线计算。内存模型基于字寻址的字节数据。运行模型是基于一个线程堆栈，堆栈的最大深度为 1024。AVM 也有一个独立的存储模型；类似内存但更像一个字节数组,一个基于字寻址的字数组。不像易变的内存，存储是非易变的且是作为系统状态的一部分被维护。所有内存和存储中的数据会初始化为 0。AVM 不是标准的冯诺依曼结构。它通过一个特别的指令把程序代码保存在一个虚拟的可以交互的 ROM 中,而不是保存在一般性可访问的内存或存储中。AVM 在某些情况下会有异常发生，包括堆栈溢出和非法指令。就像燃料缺乏异常那样,不会改变状态。有异常时,AVM 立即停止并告知执行代理(交易的处理者，或执行环境),代理会单独处理异常。

bytecode

bytecode 基本上可以认为是虚拟机的指令码，他是为了加快运行时的执行速度而设计的一种中间代码，类似于汇编指令集，它是一种虚拟的指令，并不会直接在 CPU 上运行，需要 AVM 解释执行。

它具有如下优点：

- 减少文件大小。

- 生成函数原型更快。
- 增加被破解的难度。
- 对源代码轻微的优化。

B. 智能合约特点

智能合约是由以太坊引入，它与平时的代码其实没有什么区别，它运行于一个使用 p2p 网络通信的分布式系统上。这个运行的平台，赋予了这些代码不可变性，确定性，分布式和自校验状态等特点。代码运行过程中状态的存储，是不可变的。每一个人，都可以开一个自己的节点，重放整个系统，将会获得同样的结果。

在 Ambr 中，每个合约都有一个唯一的地址来标识它自己。客户端可以与这个地址进行交互，可以发送接收 Ambr 币，调用函数，查询当前的状态等。
智能合约，本质上来说就是由代码和代码运行后存储到区块链上的状态的这两个元素组成。

C. 图灵完备

在可计算性理论里，如果一系列操作数据的规则（如指令集、编程语言、细胞自动机）可以用来模拟单带图灵机，那么它是图灵完备的。虽然图灵机会受到储存能力的物理限制，图灵完全性通常指“具有无限存储能力的通用物理机器或编程语言”, 而 Solidity 就是一种图灵完备的语言.

执行费用（Gas）

为了激励网络中的节点积极的参与计算,在另一方面，也为了避免恶意代码空耗资源，每段代码在提交到网络中进行运算时均需要提供 Gas 费用。
收费体现在下述 3 个方面:

1)通用指令计算收费，当使用网络中的可计算资源(CPU, GPU 等), 我们通过 bytecode 指令收费;

2)调用子合约和创建合约收费;

3)动态使用中的内存增多收费，对于一个账户的执行，收费的单位是按照 32 字节的整倍数收费，内存访问和寻址也是基于 32 字节对齐的。访问一个大于索引量 32 字节的地址，就会需要额外的内存使用费。地址很容易超过 32 字节的限制。

综上所述，必须能够去管理这些可能发生的事件。存储费用有一个微妙的行为——为最小化存储费用（直接与一个在所有结点上的状态数据通信），清除存储的执行费用指令不仅仅免除, 而且会返回一些费用；因为初始化的存储费用往往比实际使用的多, 在清除存储空间后会有返回费用，这个返回的费用是预先支付的费用中的一部分

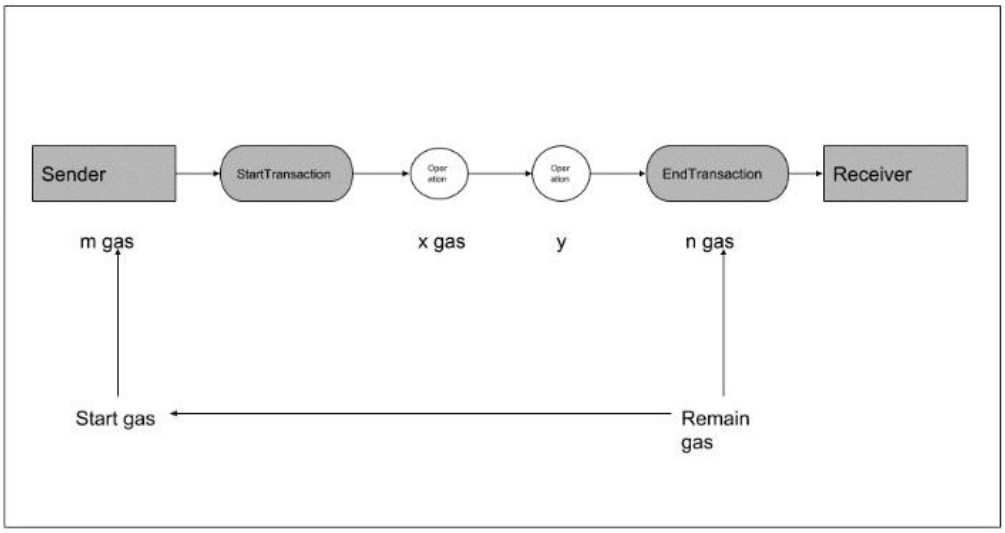

图 4 燃料使用流程

当发送者发送了足够多的 Gas 来执行合约，如果燃料剩余这些燃料将被退还给发送者。同时将执行相关结构同步到接收者账户状态。

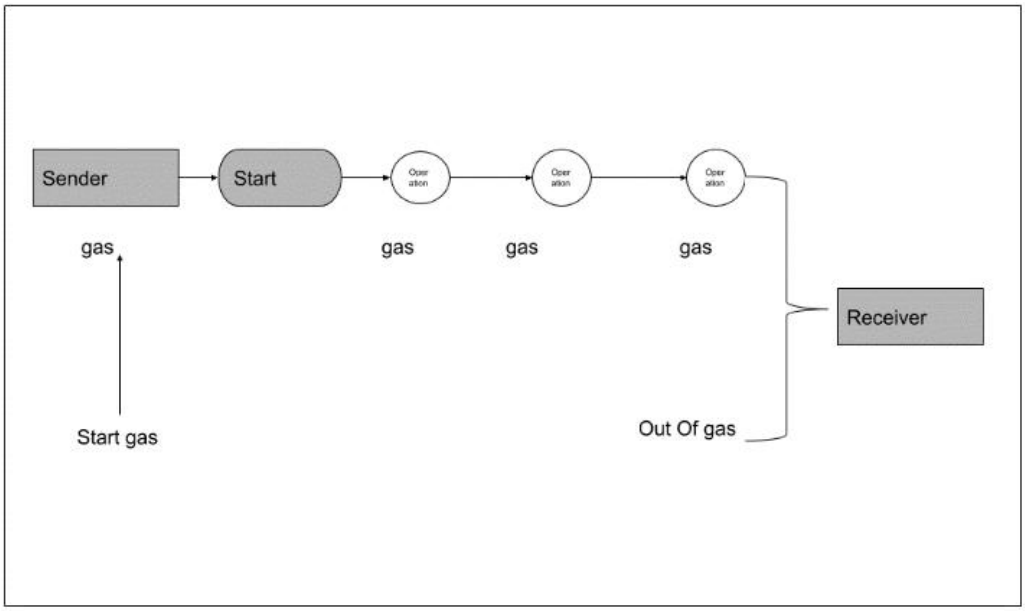

图 5 燃料用尽

当发送者发送的 gas 不足以支付运行此合约时，虚拟机将运行中止指令，跳出合约运行

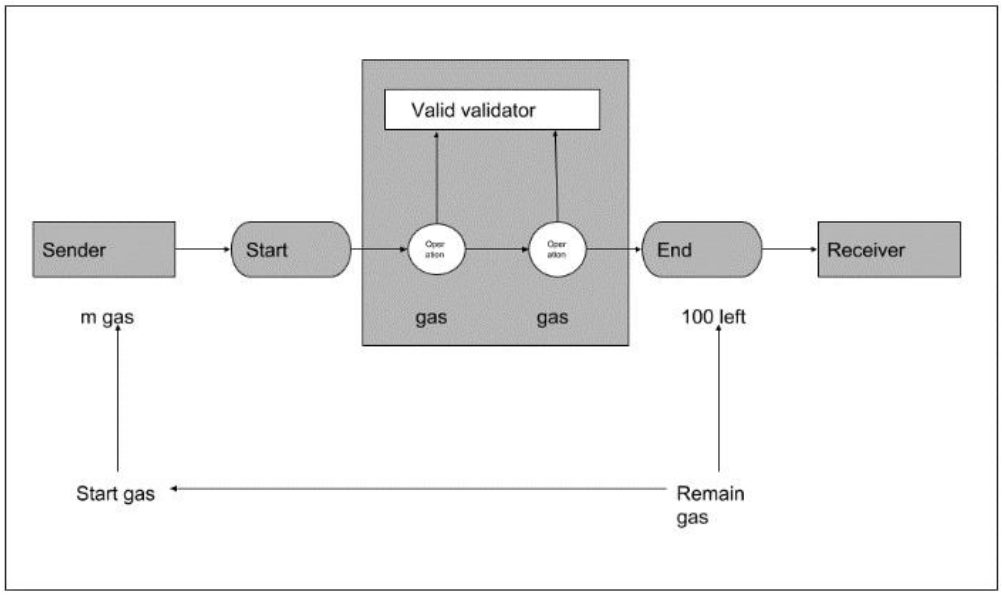

图 6 手续费

上图中描述的消耗的燃料就是手续费，消耗的燃料最终会发给验证者。

E. 合约介绍

Ambr 的智能合约被部署在 DAG 上，通过共识协议和 p2p 网络同步到网络中的其他节点。这样使得它在每一个完整节点都可以被正确执行。

智能合约通过 Solidity 语言编写，然后由编译器编译成 2 进制的字节码文件。最后通过各自节点的 AVM 解释执行。

智能合约示例

```
pragma solidity ^0.4.16

contract Foo {
    function bar(bytes[2])
            public pure {
    }

      function baz(uint32 x，  bool y)
      public pure returns (bool r) {
      r = x > 32 || y;
  }
```

```
}
```

智能合约被编译后， 会被发送到网络中存储，对智能合约会生成一个新的合约账户， 然后基于该账户添加一个 transaction 节点，如下图所示。

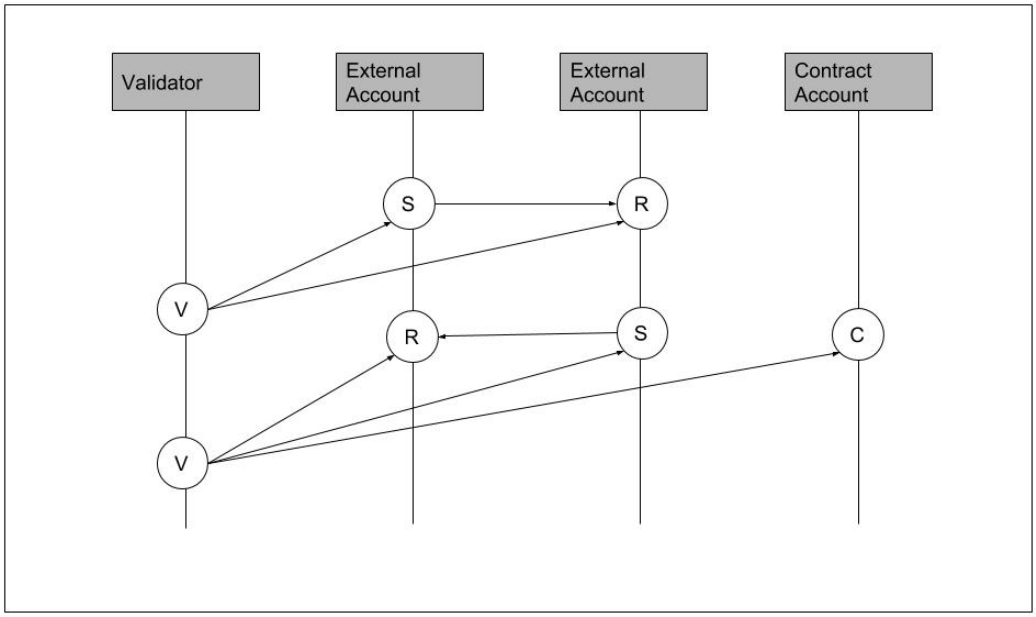

图 7 合约创建流程

存储结构如下:

```
Unit{
  version:0000...0001,
  language:solidity
  type:contract,
  previous:DC32...1CD1,
  public_key:12ea...df94,
  signature:84ec...edf6,
  hash:57da...96c2,
  creator:43ef...999,
  gas: 2334,
  gasprice: 2344,
  value: “233“，//ambr number
  initcode: “abef”,
  contract: “efdae...ef32”,
```

```
//this is optional
metadata: {
   sources:
   {
      “filename.sol”: {
         “keccak256”: “0x123…”,
         “location”: “ef23….”,
      },
      “filename2.sol”: {
         “keccak256”: “0x123…”,
         “content”: “something….”,
      },
   },
   //defines the varients for EVM
   settings:
   {
      libraries: {
         “lib1”: “hashaddress”,
         “lib2”: “hashaddress”,
      },
      target: {
         “filename”: “myfile.exe”,
         “filename2”: “myfile2.exe”,
      },
      parameters: {
         “optimize”: “true”,
      },
   },
   output:
   {
      abi: “”,
      doc: “”,
   },
},
```

```
 },
}
```

激活合约

```
params:[{
  “from”: “0xxxxxxx”,
  “to”: “0x33333”,
  “gas”: 4455555,
  “gasprice”: 234524,
  “value”: “xxxxx”,
  “data”: “0x3ader33”,
}]
```

F.账户与状态

Ambr 内置了 2 种账户:

- 外部账户: 具有一个私钥。
- 合约账户: 拥有代码。

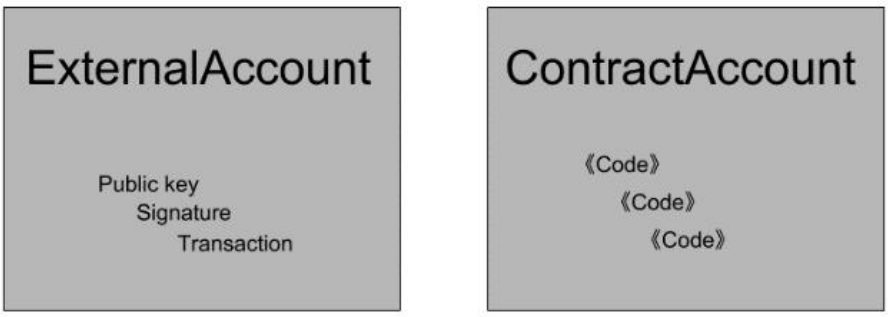

图 8 账户与状态

外部账户拥有一个私钥，可以拥有币，可以通过创建和用自己的私钥来对交易进行签名，来发送消息给另一个外部拥有账户或合约账户。在两个外部拥有账户之间传送的消息只是一个简单的价值转移. 同是从外部账户也可以发送消息，交易到合约账户，从而激活合约账户的代码，允许它执行各种动作。(比如转移代币，写入内部存储，挖出一个新代币，执行一些运算，创建一个新的合约等等)。

合约账户不可以自己发起一个交易。相反，合约账户只有在接收到一个交易之后(从

一个外部拥有账户或另一个合约账户处)，为了响应此交易而触发一个交易。我们将会在“异步消息驱动”章节来了解如何使用异步消息系统，来实现合约与合约之间的通信。

所以，在系统中，所有的交易和消息通信都是由外部账户发起。

世界状态

Ambr 的验证链全局状态机中，状态是由账户地址和相应的状态组成的一颗 Merkle 树结构。

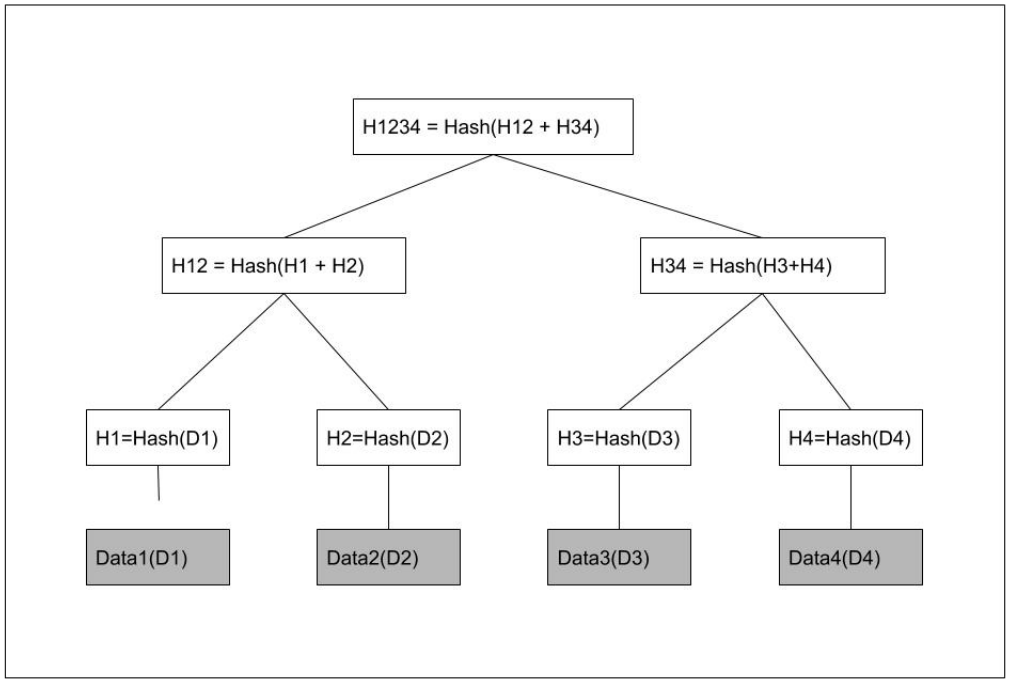

图 9 Merkle 树

如上图所示，Merkle 树是一种二叉树，由一个根节点、一组中间节点和一组叶节点组成。最下面的叶节点包含存储数据或其哈希值，每个中间节点是它的两个孩子节点内容的哈希值，根节点也是由它的两个子节点内容的哈希值组成。

对普通外部账户来说，也就是每次状态都是参与交易双方的账户，主要时存储收，发的账户地址和相应状态（余额等）。对于合约账户而言，它要保存更多的数据：1. 出发交易的账户细节，2. 合约账户的执行结果状态，3. 合约代码影响的状态结果，4. 手续费接收者账户的状态结果。

默克尔树的特点是：底层数据的任何变动都会传递到其父亲节点，一直传到树根。

默克尔树的典型应用场景包括：

- 快速比较大量数据：当两个默克尔树根相同时，则意味着所代表的数据必然相同。

- 快速定位修改：例如上例中，如果 D1 中数据被修改，会影响到 N1，N4 和 Root。因此，沿着 Root --> N4 --> N1，可以快速定位到发生改变的 D1；
- 零知识证明：例如如何证明某个数据（D0……D3）中包括给定内容 D0，很简单，构造一个默克尔树，公布 N0，N1，N4，Root，D0 拥有者可以很容易检测 D0 存在，但不知道其它内容。

根据上述表述，Merkle 树可以用于 SPV 轻节点钱包中，用于减少下载量。 在无需下载全部数据的基础上，就可以实现账户信息数据的比较、 验证。 同时，在 P2P 网络的同步时，减少了数据的传输和重传。

账户状态

- nonce：如果账户是一个外部拥有账户，nonce 代表从此账户地址发送的交易序号。如果账户是一个合约账户，nonce 代表此账户创建的合约序号
- balance：账户余额
- StorageRoot： Merkle 树的根节点 Hash 值。Merkle 树会将此账户存储内容的 Hash 值进行编码，默认是空值
- CodeHash：此账户代码的 hash 值。对于合约账户，就是被 Hash 的代码并作为 CodeHash 保存。对于外部拥有账户，CodeHash 域是一个空字符串的 Hash 值。

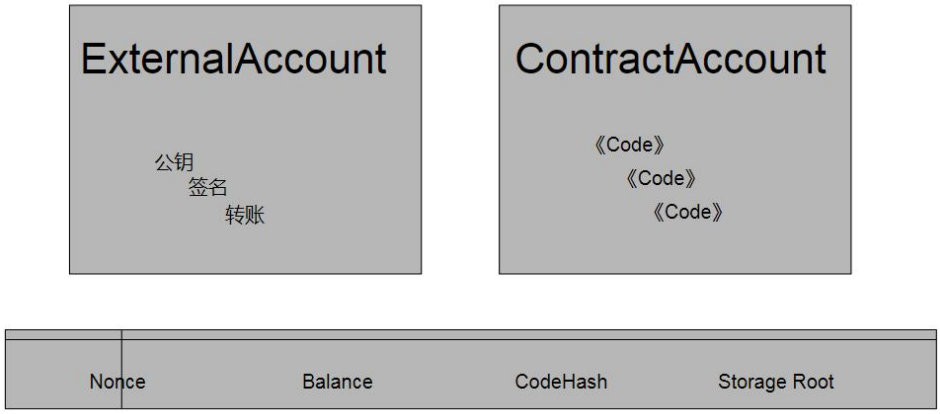

图 10 账户组成

异步消息驱动

Ambr 使用 RPC 来通信， 数据内容格式使用 Json 格式。

RPC(remote process call)，名曰远程过程调用。意思就是两台物理位置不同的服务器，其中一台服务器的应用想调用另一台服务器上某个应用的函数或者方法，由于不在同一个内存空间不能直接调用，因此需要通过网络来表达语义以及传入的参数。RPC 是跨操作系统，跨编程语言的网络通信方式。

RPC 规定在网络传输中参数和返回值均被序列化为二进制数据，这个过程被称为序列化（Serialize）或编组（marshal）。通过寻址和传输将序列化的二进制发送给另一台服务器。另一台服务器收到二进制数据以后会反序列化，恢复为内存中的表达方式，然后找到对应方法调用将返回值仍旧以二进制形式返回给第一台服务器，然后再反序列化读取返回值。

Ambr 工程使用 Google 公司的 GRPC 框架，它基于 http2，提供了强大的性能和扩展性。Json 格式作为明文格式，易于阅读和理解。

在比特币的实现中，使用了一套 JsonRPC。相对于 JsonRPC，GRPC 有 Google 公司强大的社区支持。并且，它时基于 Http2 的，拥有各种平台的统一实现，特别使用与各种设备之间的无缝交互，Http2 的网络复用，使得 GRPC 在多次频繁调用时可以重用已有的 TCP 连接，从而大大提高了效率。因为 TCP 连接的建立和释放是非常消耗资源的。GRPC 同时也对各种数据格式提供了更加强大的支持。Json，ProtoBuffer, XML,等。有利于后续系统的升级，维护和扩展。

合约创建

在 Ambr 的系统中，创建一个合约本质上是指创建一个特殊的交易，它的内容是一段代码，它的目的实际上是创建一个新的合约账户。

Ambr 中创建合约账户流程是先使用一个特殊的公式来声明新账户的地址，然后使用下面的方法来初始化一个账户：

- 设置 nonce 为 0
- 如果发送者通过交易发送了一定量的 Ambr 作为账户的初始余额
- 将存储设置为 0
- 设置合约的 CodeHash 为一个空字符串的 Hash 值

在完成了账户的初始化之后，接着使用交易附带过来的 Init code 执行合约的初始化。Init code 的执行过程是各种各样的。这取决于合约的构造器，可能是更新账户的存储，也可能是创建另一个合约账户等。

当初始化合约的代码被执行之后，会使用 gas。交易不允许使用的 gas 超过剩余 gas。如果它使用的 gas 超过剩余 gas，那么就会发生 gas 不足异常(OOG)并退出。如果一个交易由于 gas 不足异常而退出，那么状态会立刻恢复到交易前的一个点。发送者也不会获得在 gas 用完之前所花费的 gas。

不过，如果发送者随着交易发送了 Ambr(参见 Json 消息结构 Unit-Value)，即使合约创建失败 Ambr 也会被退回来。如果初始化代码成功的执行完成，最后合约创建的花费会被支付。这些是存储成本，与创建的合约代码大小成正比（再一次，没有免费的午餐）。如果没有足够的剩余 gas 来支付最后的花费，那么交易就会再次宣布 gas 不足异常并中断退出。

如果所有的都正常进行没有任何异常出现，那么任何剩余的未使用 gas 都会被退回给原始的交易发送者，现在改变的状态才被允许永久保存。

执行模型

交易的执行:

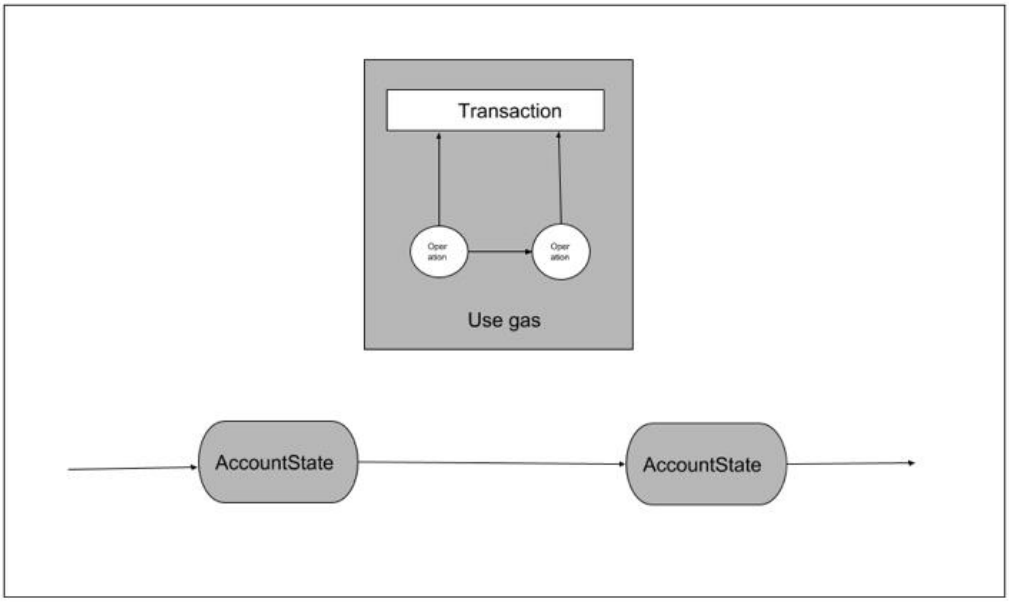

图 11 状态迁移

首先，虚拟机在执行交易前会进行一系列的前期验证：

- 交易代码是有效的可以执行的符合预先规则定义的指令集
- 有效的交易签名。
- 有效的交易序号。账户中的 nonce 就是从此账户发送出去交易的计数。交易序号一定等于发送账户中的 nonce。

交易的预估 gas 一定要等于或者大于交易使用的固有 gas，固有 gas 包括：

1）执行交易预订费用 gas

2）随交易发送的数据的 gas 费用（每字节数据或代码为 0 的费用为 4gas，每个非零字节的数据或代码费用为 68gas）

3）如果是合约创建交易，还需要额外的合约创建 gas

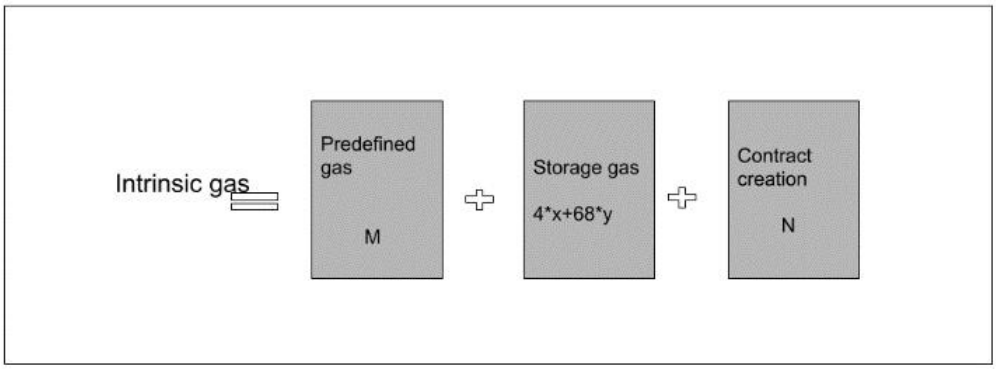

图 12 固有燃料

发送账户余额必须有足够的 Ambr 来支付”前期”gas 费用。前期 gas 费用的计算比较简单：

首先，交易的 gas 数量乘以交易的 gas 价格得到最大的 gas 费用。然后，这个最大的 gas 费用加上从发送方传送给接收方的总值。

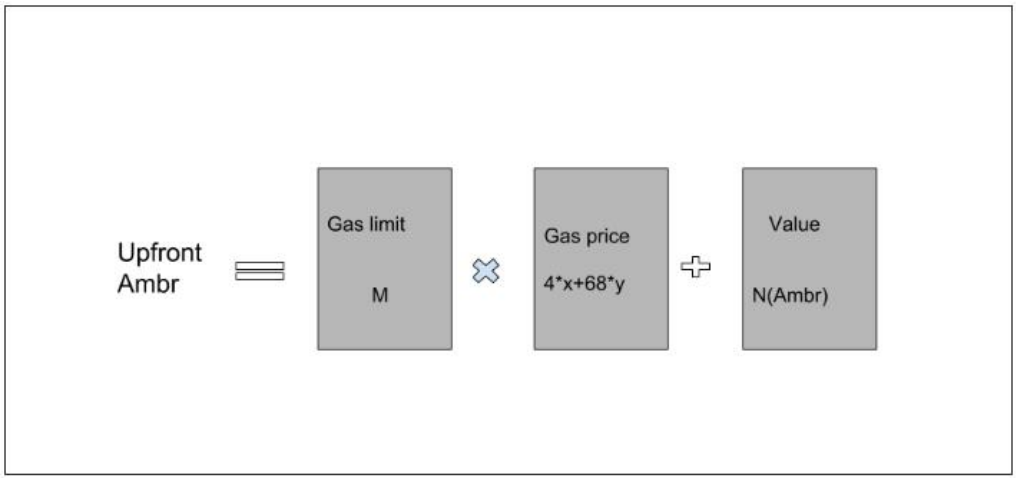

图 13 前置花费

如果交易符合上面的所有要求，那么我们进行下面的步骤。

1)我们从发送者的余额中扣除执行的前期费用，并为当前交易将发送者账户中的 nonce 增加 1。此时，我们可以计算剩余的 gas，将交易的总 gas 减去使用的固有 gas。

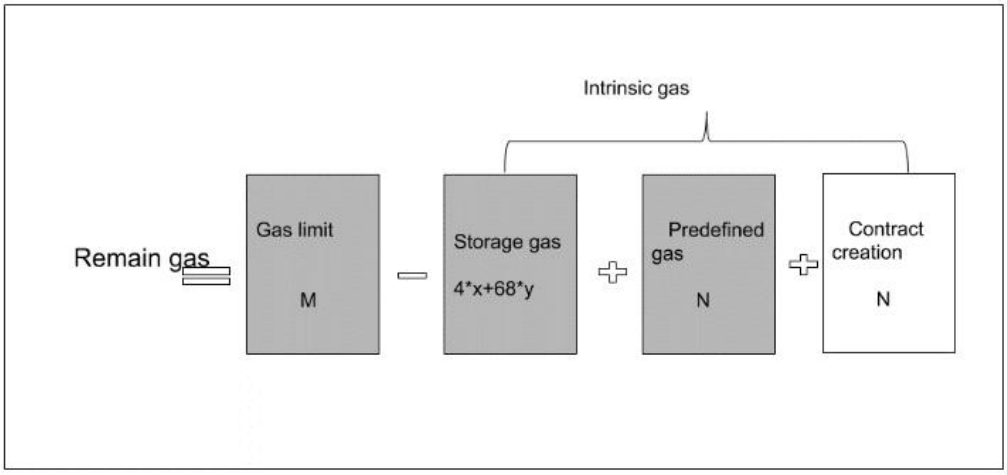

图 14 剩余燃料

2)开始执行交易。在交易执行的整个过程中，系统将保持跟踪“子状态”。子状态是记录在交易中生成的信息的一种方式，当交易完成时会立即需要这些信息。具体来说，它包含：

- 自毁集：在交易完成之后会被丢弃的账户集（如果存在的话）
- 日志系列：虚拟机的代码执行的归档和可检索的检查
- 点退款余额：交易完成之后需要退还给发送账户的总额。回忆一下我们之前提到的系统中的存储需要付费，发送者要是清理了内存就会有退款。使用退款计数进行跟踪退款余额。退款计数从 0 开始并且每当合约删除了一些存储中的东西都会进行增加。

3)交易所需的各种计算开始被处理。

当交易所需的步骤全部处理完成，并假设没有无效状态，通过确定退还给发送者的未使用的 gas 量，最终的状态也被确定。

一旦发送者得到退款之后：

- gas 花费就会发送给验证者
- 交易使用的 gas 会被添加到区块的 gas 计数中（计数一直记录当前区块中所有交易使用的 gas 总量，这对于验证区块时是非常有用的）
- 所有在自毁集中的账户（如果存在的话）都会被删除

最后，我们就有了一个新的状态以及交易创建的一系列日志。

综上，这些构成了 Ambr 的状态机

AVM 执行模型与内存布局

现在我们已经介绍了交易执行的基本知识，让我们再看看合约创建交易和消息通信的一些区别。到目前为止，我们了解了从开始到结束交易的执行必须经历的一系列步骤。现在，我们来看看交易究竟是如何在 Ambr 的虚拟机(AVM)中执行的。

协议实际操作交易处理的部分是在虚拟机中运行的， 有虚拟机解释执行， EVM 是图灵完备虚拟机器。AVM 存在而典型图灵完备机器不存在的唯一限制就是 AVM 本质上是被 gas 束缚。因此，可以完成的计算总量本质上是被提供的 gas 总量限制的。

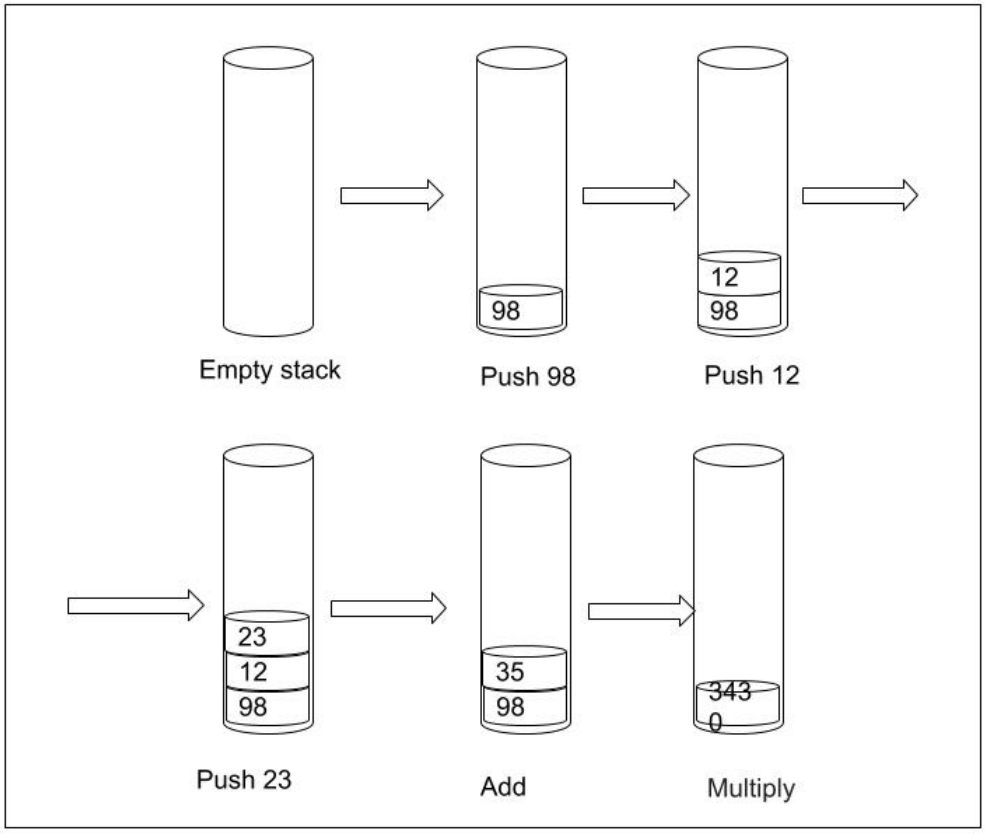

图 15 指令执行

此外，AVM 具有基于堆栈的架构。堆栈机器 就是使用后进先出来保存临时值的计算机。

AVM 中每个堆栈项的大小为 256 位，堆栈有一个最大的大小，为 1024 位。

EVM 有内存，各项按照可寻址字节数组来存储。内存是易失性的，也就是数据是不持久的。

AVM 也有一个存储器。不像内存，存储器是非易失性的，并作为系统状态的一部分进行维护。AVM 分开保存程序代码，在虚拟 ROM 中只能通过特殊指令来访问。

这样的话，AVM 就与典型的冯·诺依曼架构 不同，此架构将程序的代码存储在内存或存储器中。

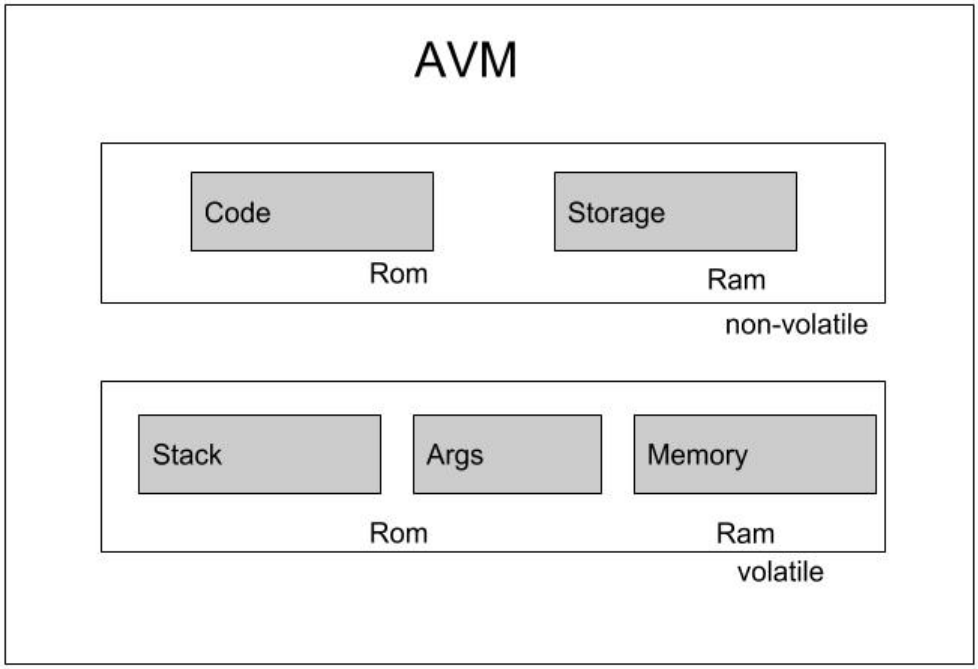

图 16 内存模型

AVM 同样有属于它自己的语言：“AVM 字节码”，当一个程序员比如 Java 程序员， 它通常使用高级语言来编码。 但是它在运行的时候为了提高效率， 是需要编译成一种字节码，我们通常都是用高级语言例如 Solidity 来编写代码。然后我们可以将它编译成 AVM 可以理解的 AVM 字节码。

现在来说执行。

在执行特定的计算之前，处理器会确定下面所说的信息是否有效和是否可获取：系统状态

- 用于计算的剩余 gas
- 拥有执行代码的账户地址
- 原始触发此次执行的交易发送者的地址
- 触发代码执行的账户地址（可能与原始发送者不同）
- 触发此次执行的交易 gas 价格
- 此次执行的输入数据
- Value 作为当前执行的一部分传递给该账户
- 待执行的机器码
- 当前区块的区块头
- 当前消息通信或合约创建堆栈的深度

执行刚开始时，内存和堆栈都是空的，程序计数器为 0。

1 PC: 0 STACK: [] MEM: [], STORAGE: {}

然后 AVM 开始递归的执行交易，为每个循环计算系统状态和机器状态。系统状态也就是 Ambr 的全局状态(Global state)。机器状态包含：

- 可获取的 gas
- 程序计数器
- 内存的内容
- 内存中字的活跃数
- 堆栈的内容
- 堆栈中的项从系列的最左边被删除或者添加。
- 每个循环，剩余的 gas 都会被减少相应的量，程序计数器也会增加。
- 在每个循环的结束，都有三种可能性：
- 机器到达异常状态（例如 gas 不足，无效指令，堆栈项不足，堆栈项会溢，无效的 JUMP/JUMPI 目的地等等）因此停止，并丢弃所有更改
- 进入后续处理下一个循环
- 机器到达了受控停止（到达执行过程的终点）

假设执行没有遇到异常状态，达到一个“可控的”或正常的停止，机器就会产生一个合成状态，执行之后的剩余 gas、产生的子状态、以及组合输出。

DAG 与合约

合约创建

合约创建是由用户发送一个合约创建消息，然后通过这个消息创建一个合约账户， 然后用消息里面所带有的初始化代码初始化这个账户，如果成功，那么这个账户会通过 p2p 网络，同步到其他的分布式节点。

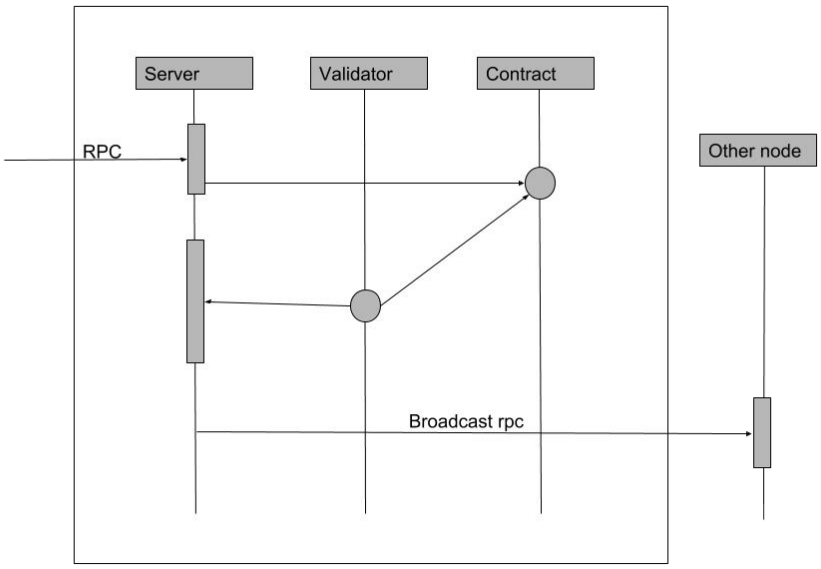

图 17 合约流程

```
Unit{
   version:0000...0001,
   language:solidity
   type:contract,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
   creator:43ef...999,
   gas: 2334,
   gasprice: 2344,
   value: “233“，//ambr number
   initcode: “abef”,
   contract: “efdae...ef32”
}
```

合约创建消息

上述表示一个 json 格式的合约创建指令。其中

version:版本号，用于未来升级协议是做辨识。

language: 合约编程语言， 因为虚拟机执行的是字节码， 所以理论上只要开发出相应的编译器， 它就会支持任何语言

type:合约类型

public_key: 创建者账户的公钥， 用于验证数据的合法性

signature: unit 签名

hash: unit 结构体的 sha256 字符串

creator: 创建者的账户地址

gas: 合约估计使用的 gas 值

gasprice: 发送者愿意支付的 gas 价格(Ambr 计量)

value: Ambr 的数量

initcode: 初始账户代码

contract: Solidity 编译后的智能合约字节码的二进制

DAG+合约的一致性问题

市场是存在一些基于 DAG 技术的产品，例如 IOTA 和 Byteball，但他们都是不支持智能合约的，这是因为就 DAG 架构本身而言，它存在一个很大的隐患——不能完全保证交易状态的原子性和统一性。从时间上来讲，可能存在特定节点（比如远程节点）确认某笔交易的时间无法估计。

从节点上来讲，全网络节点中的某个节点可能无法更新某一时刻的交易信息，即该节点没有被广播到某一时刻的交易信息。这些情况对于很多商业形态来说是一个极大隐患。

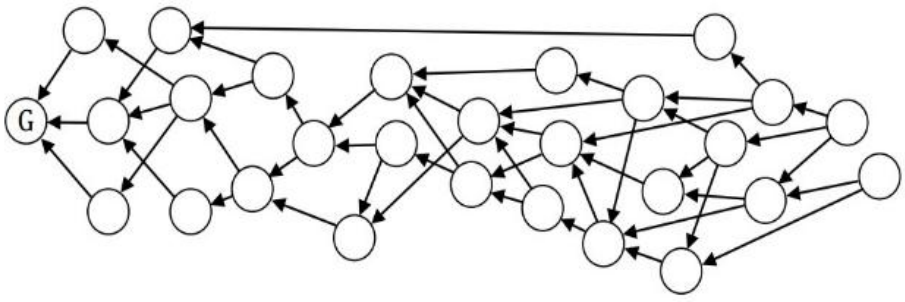

图 18 传统 DAG

如上图所示，所有 DAG 架构中的交易单元，称之为普通交易单元。普通单元中包含签名、交易、父辈哈希值，而每一个交易都是延迟确认的，由于架构实现中存在的不确定性，即交易确认的不确定性，节点同步的不确定性，由于网络延迟所带来的顺序不一致性等，每一个节点都是由后续节点来确认，但是由于后续节点的加入在不同的网络节点上面的插入顺序使不能保证的，使智能合约的实现成为了一

个很大的挑战。

但 Ambr 成功的应对了这一挑战，解决了这一难题，Ambr 引入了基于账户多链结构的 DAG 架构，使得网络中参与者的基本单位，账户独立出来，同时引入了一个验证链，采用 Casper 算法选举，成功的解决了上述不确定性，实现了智能合约，并且将智能合约和普通交易的解决方案保持了一致。

合约验证

我们的共识协议依赖并行的验证链来协助进行交易确认。

当智能合约被创建后，就会在 DAG 图中形成一个拥有一个节点的账户链。当节点被创建后，接下来会被当前验证链上的验证者所验证，同时生成一个新的 DAG 节点。这时候， 新合约会被按照共识协议所定义的规则， 被广播到其他网络节点。

广播消息被通过 p2p 网络发送到其他节点后，节点会拿到消息内部的 public_key、unithash 和 signature 进行验证，如果验证通过，就会将此节点插入到 DAG 上。

验证链合约验证规则如下

通过计算 unit 的关键字段得到 sha256 的 hash 值，然后将它和 unithash 字段对比，以检查是否被修改。

将 public_key 应用椭圆加密算法，对 signature 解密，然后将它和 unithash 字段对比，检查是否由可信发送者发送。

通过 creator 值，得到创建人账户余额，和接收到的消息的 value 值对比，判断是否该用户由足够的余额。

取出账户的合约代码，对代码进行规则完整性验证。看是否是合约的有效指令集。

VII. 脱链协议

比特币网络的堵塞导致如非必要使用者尽量避免于链上对比特币进行转账，而以太坊也曾出现一个游戏应用就导致交易严重堵塞。及时到账已经成为了我们日常生活的一部分，然而传统去中心化的区块链的转账速度确是由几秒到几天，后者严重限制了区块链技术的使用范围。

比特币和以太坊分别提出闪电网络和状态通道等脱链协议提供对即时到账和大规模并发需求的支持。

闪电网络和状态通道的基本逻辑如下

1)A、B 在链上分别将资金放入资金池。

2)A、B 链下高频次的进行交易，每次都需要 A，B 双重签名，且交易附带的序号每次加 1。

3)A、B 一方想关闭通道，则提供拥有双方签名的拥有最高序号交易证据，若另

一方在一定时间内没有提供更高序号(也就是更晚达成共识)的交易，之后按照最高交易序号来进行资金池的资金分配。

4)若有多方参与交易，则两两相连，组成一条长通道进行交易。

此协议优点为将其中频繁交易的过程放到链外进行，从而降低链上交易压力，提高交易速度和吞吐量。然而此协议的缺点也相当明显：

1)一方关闭交易通道，另一方必须在一定时间内观察对方提供的证据是不是最高的交易序号，若不是则需要提供证据，否则有可能由于对方提供了不利于己方的证据而导致损失。

2)高频次交易还有另外一个非常重要的使用场景是围绕着商家进行的发散式高频交易。而闪电网络和状态通道对该类使用场景的支持度明显不够。为此 Ambr 设计对脱链协议进行了如下设计:

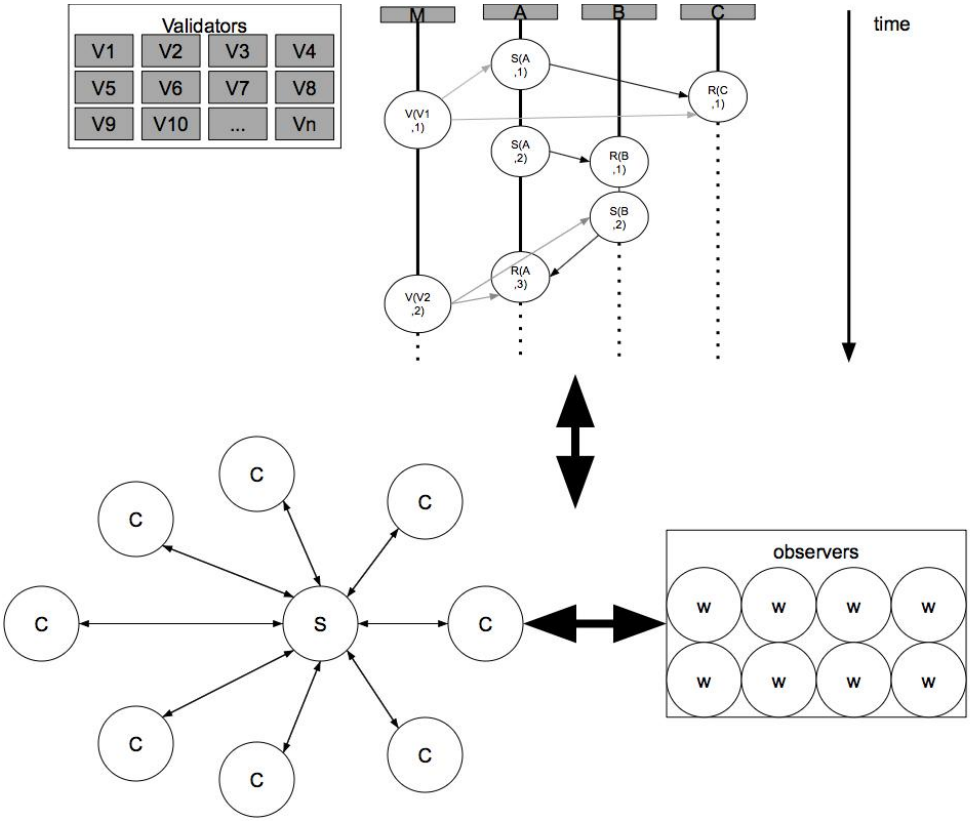

图 19 脱链交易示意图

1)增加了观察者，通道建立的时候，双方均可通过很小的代价选择观察者，用以帮助当自己不在线时，观察者能保证在链上进行举证从而保证自己的利益。

2)对于商家，可能同时再向不同的客户服务，如若被举证作弊，则丢失所有通道中所有保证金，以此来保证通道使用者的诚实。

通过脱链协议，能够将大量高频次的小交易放到链外执行，链上只保存最初和最终的一个结果，从而对系统的吞吐量进行极大程度的扩展。

假设一种场景，两个互不相识的人进行交易，需要经历沟通，付款，发货等流程，每个流程双方都留有证据，而法律要求，如若发生分歧，在一定时期内如果一方证据充足，而另一方未提供更有效的证明，则依照提供证据一方的主张在法院进行判决。

正常的交易流程如下图所示。

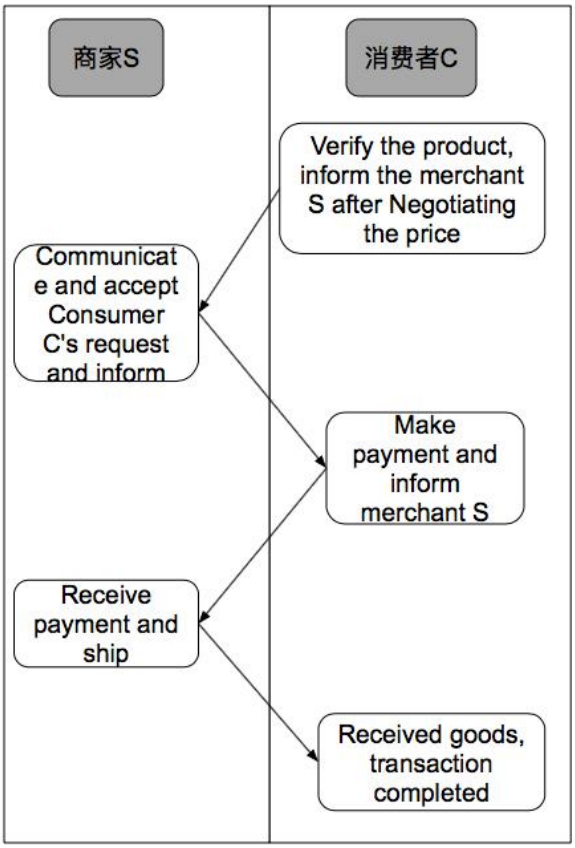

图 20 正常交易示意图

然而消费者很有可能由于购买金额不大，购买商品之后，不会时刻在线关注这笔交易有没有在法院被起诉。这时就留给了商家作恶的空间，如图所示

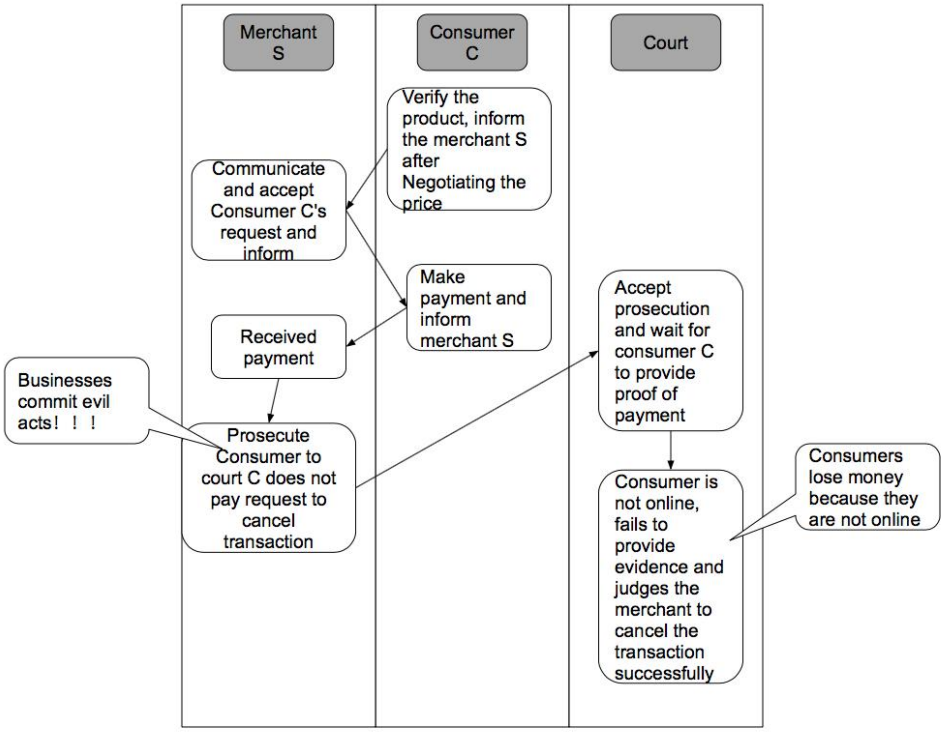

图 21 商家作恶示意图

在这种情况下，消费者为了保障自己的利益，当自己不在线的时候，就需要有知道自己交易细节的角色帮忙观察商家有没有作恶。

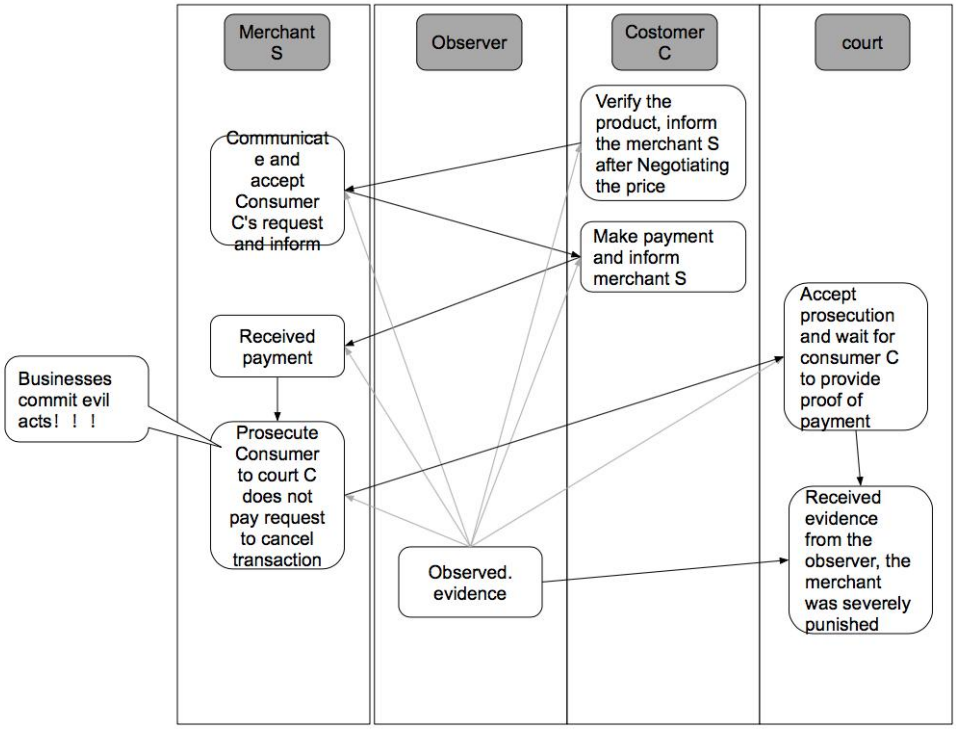

图 22 带观察者的交易作恶示意图

综上，通过增加观察者，以及对作假人的惩罚来保证在高频次的脱链交易双方利益。

VIII. 跨链协议

17 年以来，随着大量的区块链项目被开发与推广，越来越多的人力与资金被投入到这些项目中。如今规模与种类不一的区块链项目数量呈井喷状出现。随之，也诞生了一系列问题。比如在区块链项目数量增长的同时，没有及时的匹配上相应的措施，从而导致了大多数的区块链之间无法通信与连接。换句话说，每个区块链项目都是一个个的”信息孤岛”，这极大地限制了区块链的应用空间。因此，有效的跨链技术是解决这些问题的关键。跨链技术可以把众多的区块链项目从一个个的“信息孤岛”中拯救出来，为它们建立起一个个互通的桥梁。

跨链技术是实现区块链向外拓展的手段,很大程度上决定了区块链项目的发展上限。Ambr 采用的跨链技术主要基于安全性、高效性和实现的难度来设计。

关于跨链技术的历史：早期的跨链技术更多关注资产转移 ，以 Ripple 和 BTC Relay 为代表;现有的跨链技术更多关注跨链基础设施 ,代表有 Polkadot 和 Cosmos。最新出现的 FUSION 实现了多币种智能合约,这意味着可以在整个市场中产生多种跨链金融交易。

A. 公证人机制

主要代表是 Ripple Interledger 协议。它适用于所有记账系统，目标要实现全球统一的支付标准。

B. Corda

Corda 是一种“类区块链”技术架构。交易双方共同选出一个具有高可行度的公证人，让公证人来检验数据的有效性和唯一性。若公证人证实该交易可行，那么交易就会达成，此时交易公证人的账本会进行同步。这么做的好处在于在保证安全性的条件下，使得交易处理得更加高效。

Fabric：新定义的概念包括链、peer、通道和共识服务。Peer 可以参与多个账本使得 Fabric 具有扩展性，Peer 之间具有事务隔离、账本隔离等特点。

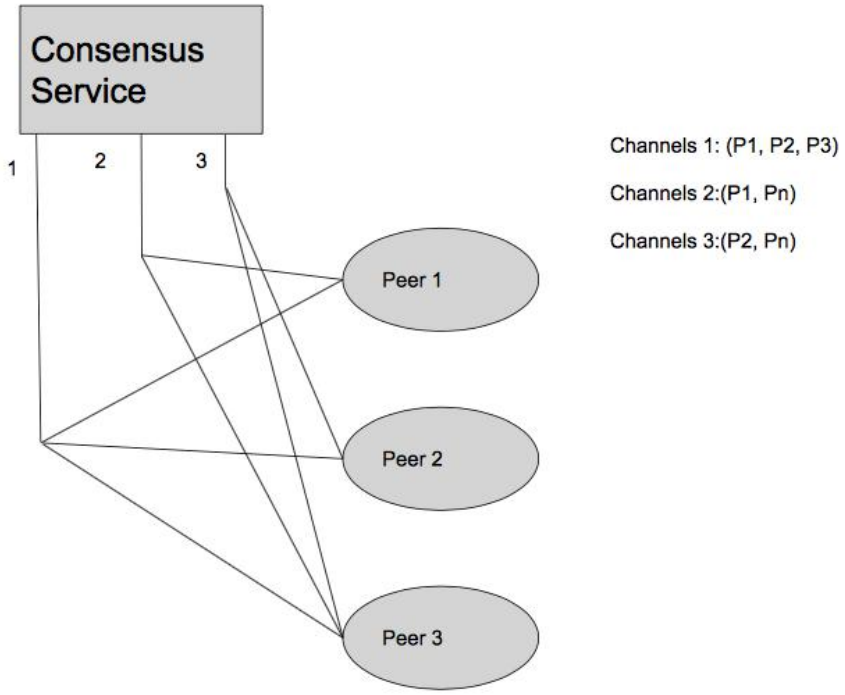

图 23 Fabric 通道

C. Cosmos

Cosmos：(Interchain Foundation 的跨链开源项目)专注于解决跨链资产转移，该区块链网络主要由 Zone 和 Hub 构成

1)Cosmos Zone 是单独的区块链空间。

2)Cosmos Hub 中心是一种多资产权益证明加密货币网络。

Hub 是中继链，由去中心化的验证人组来记账。一个 Hub 与多个 Zone 进行通信，每个 Hub 具有与它相关联的多个 Zone 的账单信息，以此来产生一个多资产中心账本。Hub 保证资产在不同的 Zone 转移的过程中，里面的资产总量保持不变。Cosmos 过程如下：

首先，Hub 和 Zone 的跨链通信由 IBC 协议来达成。我们假设 Zone1 想要和 Zone2 进行跨链交易。

1)Zone1 生成交易信息，并发布在 Hub 上。

2)Hub 生成 Zone1 的跨链信息包存在的证明，并且发布在 Zone2 上。

3)Zone2 受到消息包，并且在 Hub 上发布已收妥的证明信息。

4)Hub 给出 Zone2 已收妥的证明的证明，并且把消息发在 Zone2 上。

Cosmos 优点包括如下：

1)每个 Zone 里面的代币转移都会通过它们共同连接的 Hub，因此每个 Zone 里面的资产都会被记录

2)若有一个 Zone 发生故障，不会使得其他有效的 Zone 产生影响

3)新加入的 Zone 可以轻易地被加入到 Hub 中心里来

D. 跨链交易

跨链技术是实现区块链向外拓展的手段，很大程度上决定了区块链项目的发展上限。关于跨链技术，目前主流的跨链技术包括：公证人机制（Notary schemes）、侧链/中继（Sidechains/relays）、哈希锁定（Hash-locking）、分布式私钥控制（Distributed private key control）。其中主要跨链技术的区别如下图所示：

跨链技术	Notary 公证技术	Relay 中继及侧链技术	Hash-locking 哈希锁定	Ambr 跨链技术
互操作性	所有	所有	只有交叉依赖	所有
信任模型	多种公证人诚实	链不会失败或者受到“51%攻击”	链不会失败或者受到“51%攻击”	链不会失败或者受到“51%攻击”
适用跨链交换	支持	支持	支持	支持
适用跨链资产转移	支持	支持	不支持	支持
适用跨链 Oracles	支持	支持	不直接支持	支持
适用跨链资产抵押	支持	支持	大多数支持但是有难度	支持
实现难度	中等	难	容易	中等
多币种智能合约	困难	困难	不支持	支持
实现案例	Ripple	BTC Relay/Polkadot/COSMOS	Lightning network	Ambr 待实现

E. Ambr 跨链交易

Ambr 的路线图包含了使用中继技术和类似于 Polkadot 和 Cosmos 这样的未来协议，以支持不同加密币种之间的跨链交易。Ambr 采用的跨链技术主要基于安全性、高效性和实现的难度来考虑设计。Ambr 跨链协议的核心技术是 Ambr 中继链，该技术使得 Ambr 不仅具有 Polkadot 的可伸缩性、可扩展性，也具有 Cosmos 的兼容未来区块的特点。

Ambr 的跨链协议支持不同币种之间的跨链交易，允许用户实现 Ambr 币和 BTC、ETC、ZCash 等代币之间的交易。打破各个币种之间的交易障碍，使得一个个“信息孤岛”在 Ambr 中完成无障碍的转移、交易与兑换。Ambr 中继链对于区块链不同币种间的交易与兑换起到巨大的推动作用，对新兴的区块链相关公司提供十分新奇的技术与思想指导，对区块链行业的蓬勃发展有重大创新意义。

Ambr 跨链协议设计原则：

1)安全性：这是 Ambr 跨链设计的基石，Ambr 要在实现跨链的同时具有绝对的安全性。跨链过程中产生的历史数据是极其难以修改的。

2)性能：跨链的效率也是 Ambr 一个很重要的考虑因素。在保证安全性的条件下，尽可能地提高 Ambr 的吞吐量和跨链确认的速度。换句话说，就是让 Ambr 跨链每秒处理的交易笔数达到一定的数量，使得用户享受较好的交易体验。

Ambr 中继链是使用类似于 Polkadot 中继链和 Cosmos 的跨链协议来综合设计的。Ambr 中继链主要起到记录交易地址与交易金额和验证交易是否合法的作用。一条交易链上具有多个交易单元。每笔跨链交易必须由至少一个交易单元进行记录和验证。每种和 Ambr 中继链连通的外币其中产生的合约必须由中继链的中转单元进行审核。而且每个外币的某节点的交易地址肯定在中继链上有一个映射地址。每笔交易金额都会储存在中继链中的中转单元中。

中继链为解决 Ambr 和外币的跨链交易提供了技术和平台支持，也为不同外币之间的跨链交易提供了空间与机会。

中转单元由验证者轮流发布，其主要作用如下：

1)对所有未经验证的交易进行合法性验证。

2)为上一个中转单元收集投票。

3)若上一个中转单元作恶，则发布惩罚。

其结构示意如下

```
Transfer
{
    version:0000...0001，
    previous:DC32...1CD1，
    height:999，
    verify:...，
    punishment:...，
    direction: output/input，
    sourcelink:Ambr，
    targetlink:ETH，
    amount:99ETH，
   public_key:12ea...df94，
    signature:84ec...edf6，
    hash:57da...96c2，
}
```

其中

- version：表示版本号。
- previous：指向上一个中转单元。
- height：表示当前中转单元的高度。
- verify：验证过的交易和投票。
- direction：表示方向，分 output（输出）和 input（输入）。
- sourcelink：表示源链， input 情况下可以是 ETH、BTC 等，output 情况下只能是 Ambr。
- targetlink：表示目标链， output 情况下可以是 ETH、BTC 等， input 情况下只能是 Ambr。
- amount：表示金额，需要自带单位。
- punishment：对未通过投票的作恶验证者进行举证并惩罚。
- public_key，signature：对单元进行签名，保证该单元为验证者发送。
- hash：对整个单元进行 hash 运算，保障交易单元在网络传输的过程中不被修改。

以 Ambr 与以太坊（ETH）跨链交易为例：

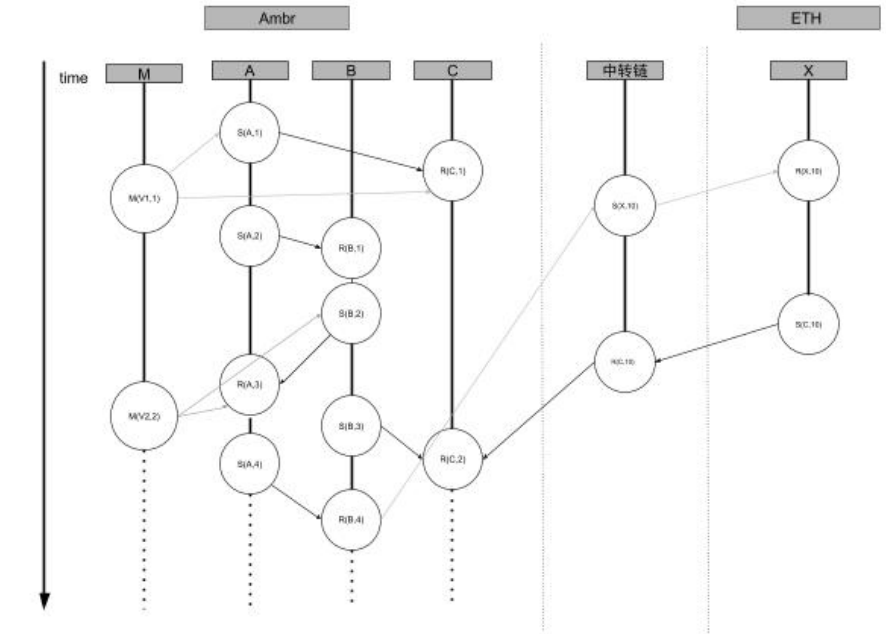

图 24 Ambr 与 ETH 跨链交易

输入交易：

1)假设以太坊上的帐户 X 需要向 Ambr 帐户 C 支付 10ETH。X 需要在以太坊上

申请一份中转合约，中转合约应包含 Amber 账户 C 地址和发送金额，并由账户 C 签名。

2)中继链主动监测到 ETH 有 Ambr 账户相关的合约。

3)中继链中某一转单元记录审核该合约，并由验证者合集数字验证。

4)如果数字验证通过，则在账户 C 末节点增加关于该交易的记录，并在 ETH 中锁定 10ETH， 使之暂时不能在 ETH 中流通；如若不通过，则合约作废，交易失败。

5)最终由 AMBR 验证链中的某一验证单元对账户 C 的末节点进行验证，达成 AMBR 全网共识。

输出交易：

1)假设 Ambr 账户 B 需要向以太坊上的账户 X 支付 10ETH。账户 B 需要向 Ambr 中继链发送一条交易请求，该交易请求应包含 ETH 账户 X 的地址和金额，并由账户 B 签名。

2)中继链接收到 Ambr 账户 B 的请求并记录。

3)中继链中某一中转单元记录审核该交易请求，并由验证者合集数字验证并且签名。

4)若验证通过，则账户 B 的末节点增加一条扣除 10ETH 的交易记录，并由中继链转化账户 X 的地址在 ETH 的映射，并释放 ETH 中锁定的任意 10ETH。

5)最终由合约将该 10ETH 发送给 ETH 中的账户 X。

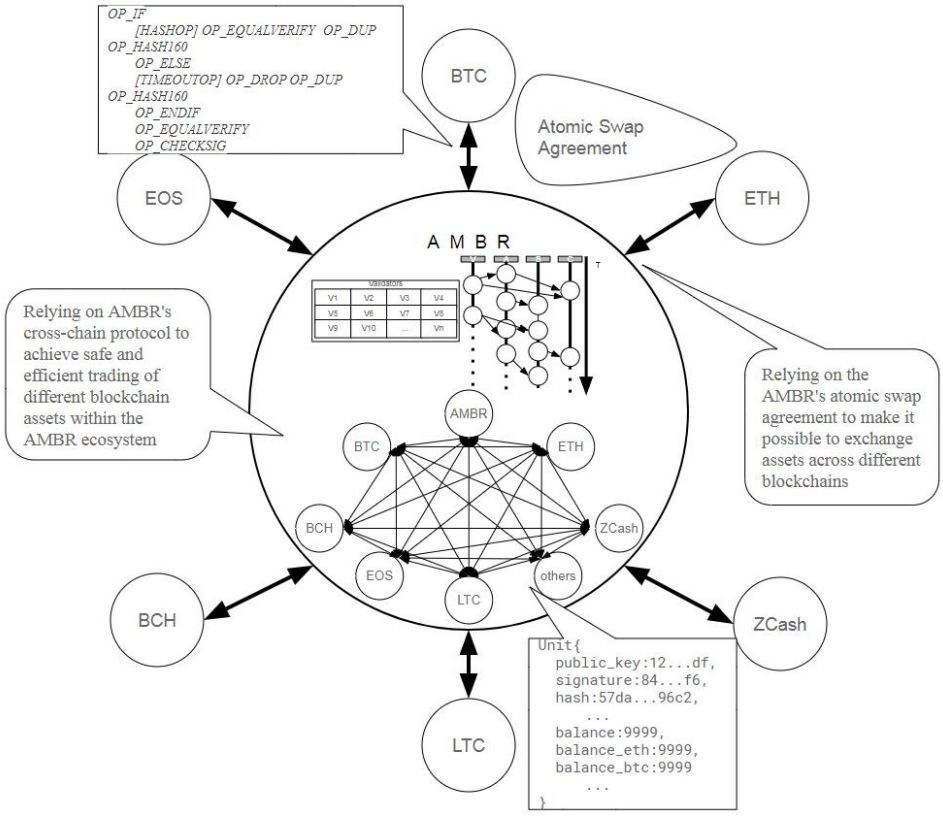

图 25 跨链技术总览图

Ambr的跨链交易技术不仅仅局限在某一单一币种。Ambr的中继链可以与BTC、ZCash、EOS 等衍生空间连通。换句话说，市面上现在的大部分币种衍生出来的区块空间都可以接入到 Ambr 的中继链上。而且任何新诞生的币种只要和 Ambr 签订合约并且达成共识，都可方便快捷地被后续接入到 Ambr 中继链。这样就可以使得 Ambr 实现很大规模的无限扩展，这样子的方式大大满足了全球交易的需求。

不仅如此，现在市场上的大部分币种甚至可以通过 Ambr 中继链进行直接交易，比如 BTC 和 ETH 可以在 Ambr 中继链上直接进行交易。这样就无需在 BTC 和 ETH 直接建立渠道或者平台，大大节省了技术和人力成本。Ambr 中继链可作为市场上一个巨大的跨链平台，实现不同币种之间的交易。

为确保 Ambr 中继链上的跨链交易的即时性、安全性，Ambr 原子互换实现将会被采用。Ambr 原子互换是一种新技术，允许 Ambr 币与在其它类型的数字资产之间实现无需信任的点对点交易。这种交易可以在瞬间完成，任何一方都没有机会违反协议。而且当交易其中一方中途退出的情况下，数字资产会在一个规定的时间后退还给双方。

这种技术对加密数字货币的未来具有重大意义，因为这种无缝的跨区块链加密数字货币互换能力开启了一种新的应用。Ambr 原子互换可以打通各种加密数字货币之间的交易阻碍，确保交易正确。如果用户想要在 Ambr 币与在其它类型加密数字货币之间进行交易，那么这种技术可以让用户完全控制自己的资金。

Ambr 原子互换的工作原理

假设一个例子，M 和 N 是数字资产交易的双方，M 在 Ambr 在有自己的帐户，N 在以太坊、Ambr 上有自己的帐户。现在 M 和 N 通过电话、网络等途径谈妥了一笔交易，也知道了双方在各自的 Ambr 账户。根据达成的交易，M 准备把他在 Ambr 上的 100Ambr 币转让到 N 在 Ambr 的帐户下，N 将通过 Ambr 向 M 支付 200ETC。为了完成 Ambr 币和 ETC 的交易，双方依次执行以下步骤：

- 为了原子性地完成这笔交易，首先 N 自定义一个指令 X，计算得出 V:= Hash(X)，X 现在只有 N 自己知道。
- N 在 Ambr 上发布一笔转账交易，有条件地转让 200ETC 给 M。但和普通的转账交易不同，这笔交易附带一个哈希锁定条件：M 只有在 4000 秒内向 Ambr 出示一个满足 Hash(X') == V 的指令 X'才能将 100ETC 入自己账上（采用账户模型还是 UTXO 模型没有本质区别），若 M 超时未能领取 ETC，则 N 可以通过在 Ambr 上发起一笔退款交易把 100ETC 返回自己账户上。哈希锁定条件中的 V 和超时时间都是公开的，M 当然也看得到。
- M 现在在 Ambr 上看到 N 发起了这样一笔交易，但他不知道解锁指令 X 是什么，

所以他必须向 N 通过 Ambr 支付 100Ambr 币以买到这个指令 X。于是 M 在 Ambr 上给 N 发送一个附带同样哈希锁定的指令转帐，有效期 2000 秒，超时若 N 未领取则转帐金额会自动退款。这个哈希锁定的指令转帐原则上很容易实现，其逻辑是：当 N 点击指令转帐后会弹出一个对话框，要求 N 输入一个满足 Hash(X') == V 的指令 X'，如果输对了，指令转帐中的 Ambr 币会转入 N 在 Ambr 上的账户，同时 Ambr 会给 M 发送一个回复，告知 M 转帐已被领取，且在回复上同时显示 N 输入的指令 X。如果 N 输错了指令 X'，则 N 无法收取转帐金额。

- 现在 N 收到了指令转帐，及时点击指令转帐，且输入了 N 自己知道的指令 X。因为 V == Hash(X)，所以 N 成功拿到了 100Ambr 币。根据程序逻辑，Ambr 给 M 发送一个回复，告知 M 转帐已被收取，且 N 输入的指令是 X，于是 M 知道了指令 X。
- 因为 M 现在知道了指令 X，他现在就可以在 Ambr 上利用指令 X 来提取那 200 悬而未决的 ETC。M 及时进行了操作，就在 Ambr 上拿到了 200ETC。
- 至此，M 拿到了 200ETC，而 N 拿到了 200Ambr 币。在交易过程中，以太坊和 Ambr 完全不需要互相通信，但仍然确保了 Ambr 币和 ETC 的原子互换。

以上是正常流程，在异常流程下互换的原子性仍然是成立的。比如在上面的第 3 步中 M 没有通过 Ambr 发出转帐，N 既然看不到转帐也就不会输入指令 X 给 M，M 拿不到指令 X 也就无法在 Ambr 上提取 ETC，交易的原子性得到保证。

哈希锁定机制在以太坊等区块链上都很容易实现，至于 Ambr 要实现上述哈希锁定的转帐也不会有太多的困难。

此外，由于两种转账都是指定对方的转账，程序可以被设计成可以由第三方提供指令帮助解锁，但资产仍然按原先指定的流转方式。这种设计使得用户在自身钱包失效或暂时无法访问 Ambr 可以安全地委托他人代为操作，因此解锁指令在 Ambr 中公开不仅不会带来安全问题，还会有额外的好处。

IX. 安全相关

通过上文描述的共识机制可以总结一个合法的交易必须有以下特点：

1) 账户发布的交易不可以是已经确认过的交易。这一点可以通过 Hash 值来判断此交易是否存在。

2) 每个交易必须有合法账户签名。

3) 账户中的交易必须呈链式分布，即每个交易必须有之前的交易信息。

4) 由于 Ambr 是异步模型，Receive 模块可以等到签收者签名生效，然而 Sender 模块是立刻生效的，即 Sender 模块广播后，Send 模块发送者的余额会立刻扣除 Send 模块对应的部分。

随着货币种类数量的增多，攻击类型也在不断增多，因此，从安全角度来看考

虑可能的攻击方式是十分必要的，基于以上结论，讨论可能的攻击方式和 Ambr 如何应对攻击。

a. 双花攻击

双花攻击的的典型示例如下：

1)假设 Alice 是买家，Bob 是卖家，Alice 通过 Bitcoin 向 Bob 购买了一件数字物品，Bob 确认后向 Alice 发送物品。

2)此时 Alice 又向 Charlie 购买了一件数字物品，Alice 用同样的付款方式，Charlie 确认后发送物品。

假设上述情况下 Alice 只有 10 元，商品价格也是 10 元，Alice 用 10 元买了两件物品，这就造成了双花问题。

Bitcoin 系统中，在 Bob 确认收到付款后，Bob 需要等待几个生成的 Blocks 后，再向 Alice 发送物品。此机制虽然解决了安全性问题，却也造成了延迟性问题。而 Ambr 的设计模式完美的规避了这个问题。Ambr 的账户链的每个块都显示账户余额。假设 Alice 要达到双花的目的，它的模块链必然是下面情况的一种：

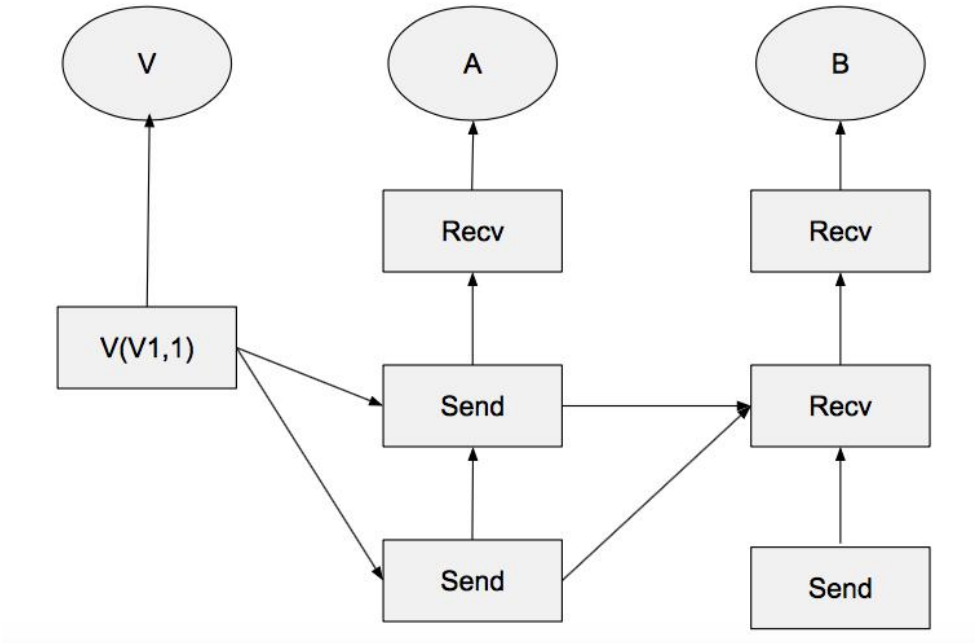

图 26 双花场景 1

如图所示，A 下面有两个连续的 Send 模块。这种情况下两个 Send 模块都是有效的。由于 Send 模块是立即生效的，所以账户余额会立即变动。假设 Alice 付款后余额不足以支付第二个 Send 模块。系统会立即检测，第二个 Send 不会生成。

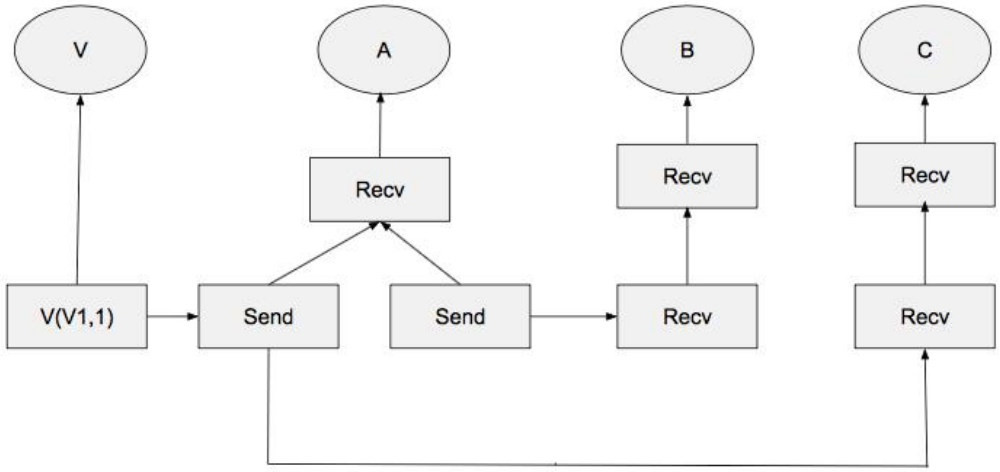

图 27 双花场景 2

如图所示，A 交易链上出现分叉，两个 Send 模块同时指向 A 链中一个 Recv 模块。Ambr 会通过账户的验证者来判断两条支链的有效性。方法是验证者会将两条支链广播到网络中，从收到的回应进行投票，从而判断哪条支链有效。无论投票结果如何，支链总会保留一条而放弃其它的支链。这样双花攻击在 Ambr 系统中不会生效。

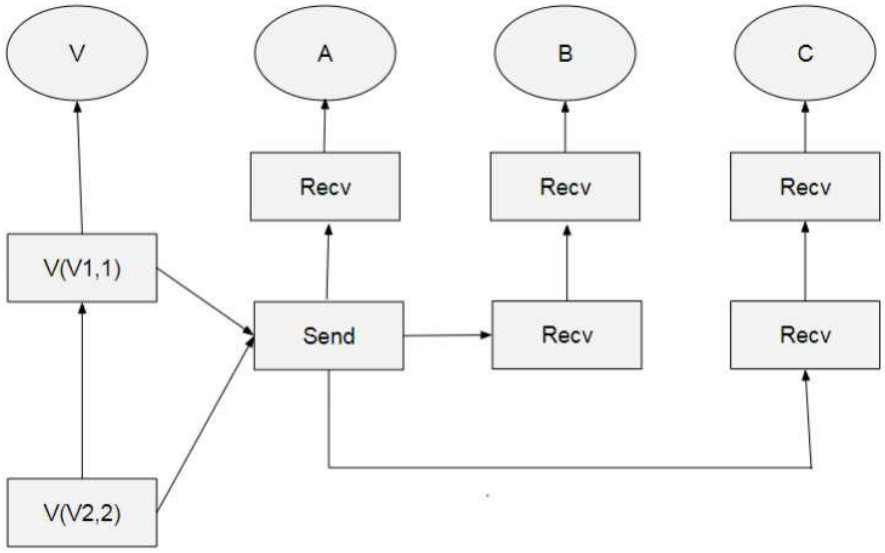

图 28 双花场景 3

3）另一种双花攻击如图所示。假设在 M(V，0，1)验证交易过程中，M 已经验证 A-->B 的交易，并且此交易是有效的。在 M（V，0，2）验证中，M(V，0，2)验证了 A-->C 的交易，并将其标记为有效。这样就完成了双花。从 A，B，C 中难以察觉哪

个交易是有效的，此时可以从验证者链中判断 M(V，0，2)是否作弊。方法是在验证者投票过程中，由于 M(V，0，1)已经验证过交易，M(V，0，2)的交易会被验证者组成集合反对。M(V，0，3)收集投票后可以判断 M(V，0，2)验证的交易不是有效交易，并对 M(V，0，2)进行举证。系统会对 M(V，0，2)进行处罚，没收其存款，并将其踢出验证者集合，并且由 M(V，0，3)对 M(V，0，2)验证的交易进行重新验证。

b.女巫攻击

女巫攻击最早由 Douceur 提出，即在对等网络中，单一节点具有多个身份标识，账户生成大量节点来影响投票的结果。然而 Ambr 的股权分配是由账户拥有的资产分配的，所以女巫攻击对 Ambr 系统没有作用。

c. Block Gap Synchronization

在 Ambr 的设计模式中，为了保持网络传输数据量的最小化，Send 和 Receive 模块是由 UDP 传输的。然而 UDP 传输并不保证可靠性，模块并不保证安全接收。所以 Ambr 系统节点有如下两种选择：

1）丢弃这个模块，并将其作为攻击节点发出的垃圾信息。

2）要求发送者重新传输，此时使用安全可靠的 TCP 传输。

此时攻击者发出大量垃圾模块，就会造成 TCP 传输增多，网络负担加重，对 Ambr 系统的实时性和有效性就会有影响。因此，Ambr 作为如下应对：

1）对收到了垃圾模块由验证者进行广播并对回应进行计数。

2）如果投票低于阈值，系统会将其作为垃圾模块并丢弃。

d.粉尘攻击

粉尘攻击即攻击者在账户之间发出大量没有必要却是有效的垃圾交易，如交易金额为 0 等等，大量的交易使网络拥堵从而对 Ambr 造成影响。在 Ambr 设计模式中，手续费的设计目的就是应对这种攻击方式。攻击者在发布交易时，需要付一定的手续费，才能发布交易。这样的结果是攻击者的攻击成本大大增加使收益远远小于成本。这样就可以降低这种攻击数量。

e. Penny-spend Attack

Penny-spend Attack 是由攻击者向同一节点发出大量交易，从而该节点的存储资源会渐渐耗光，从而不能处理其他交易。这种攻击方式和上述粉尘攻击相似，不同的攻击同一节点。不过和粉尘攻击相同的是，攻击者的成本是一样的。而且从 Ambr 的设计模式中可以看到，用户并不需要关心交易的历史记录，因为余额会随着交易变动。因此节点可以选择删除一些历史记录，例如一些时间比较远的，交易金额比

较小的等等，这样可以使攻击成本大大增加从而达到防范的目的。

f. > 50%攻击

从 Ambr 的设计模式中可以看出，Ambr 是基于股权来进行投票的。因此，如果攻击者拥有大于 50%的股权，那么他就可以可以操纵投票结果，从而达到攻击目的。但是，攻击者要拥有 50%股权有以下几点限制：

1）Ambr 系统是基于股权投票的，那么拥有股权多的人就不会希望 Ambr 系统崩溃。如果 Ambr 系统崩溃，这些人需要承受大量损失。

2）如果攻击者要收购股份，那么它必须收购 Ambr 系统中 50%的股份，收购成本的增加使攻击风险也相应增加。

3）Ambr 的验证者是多样性的，这样使验证者联合攻击的概率大大减小。

4）Ambr 的设计模式会严惩攻击者，一旦确认攻击攻击者资产就会被没收。

因此，攻击者的攻击成本和风险都会大大增加，从而降低攻击者的攻击欲望。

g.拜占庭将军问题

从上述共识中可以看出，验证者所组成的集合可以组成一个分布式系统。假设系统网络延迟为 L，从以前的 BFT 研究中可以得出以下结论：

1）在部分同步的系统中（即 L < d ，d 为一个常数，但是并不知道 d 的边界值），如果有小于 1/3 的节点作恶，或不响应，系统也可以达成共识。如 PBFT 算法可以在小于 1/3 的节点不响应的情况，达成共识。

2）在异步模型中，（即 L 为无穷大），则不存在达成共识的方法。

假设攻击者利用网络攻击（DoS），使至少 1/3 的节点无法响应，这样验证者组成的分布式系统就无法达成共识，交易也就永远无法验证。

基于以上假设，Ambr 设定了一个边界值（1 小时或 1 天），如果在时间超过这个边界值，验证者链的长度也没有增长，系统则会判定验证者集合组成的分布式系统无法达成共识。Ambr 的应对策略如下：

1）Ambr 会对验证者集合进行重新组合，或替换一些没有响应的节点，或组合一个新的组合。

2）系统所有节点会接到提示，告知验证者集合无法达成共识。此时系统设定一个延时时间，假设此时间为 T。

3）节点在收到新的验证模块后开始计时，当收到模块的时间 t > T 时，节点可以确定验证模块有效。

4）系统对原来的验证者集合进行调查，如果调查结果是没有网络攻击，验证者是自身原因没有响应，那么会对验证者处以相应罚金。如果调查结果是攻击者发出了网络攻击，那么攻击者的所有存款都会被没收。

h. Castastrophic Crashes

由于 Ambr 中权重是由投票者所有的资产决定的。假设稳定点需要的投票通过百分比为 k%，即投票数所拥有的资产在总资产的比重大于 k%才能形成稳定点。这样就会衍生另一种攻击方式，攻击者只要攻击拥有资产比较多的节点，使这些节点不能响应，且这些节点的所拥有的资产大于 1-k%。攻击成功后，投票永远不能通过，即稳定点也不会再生成。

这时系统会检测投票不能通过的原因，如果网络不能达成共识，按上述步骤进行。如果投票不能通过，应对策略如下：

1) 系统首先检测不能响应的节点放到一个集合中，并且集合中的节点按照拥有资产和总资产的比重进行排序。

2) 系统会设置一个延迟时间 T。如果当前时间 $t > T$ 时，系统按照集合中的节点顺序取出节点中的资产放到投票对应的资产中，直至投票对应的资产达到通过所需的资产。

3) 系统对集合中的节点进行检查，如果调查结果是没有网络攻击，验证者是自身原因没有响应，那么会对验证者处以取出资产相应百分比的罚金。如果调查结果是攻击者发出了网络攻击，那么攻击者的所有存款都会被没收。

X. 社区工作

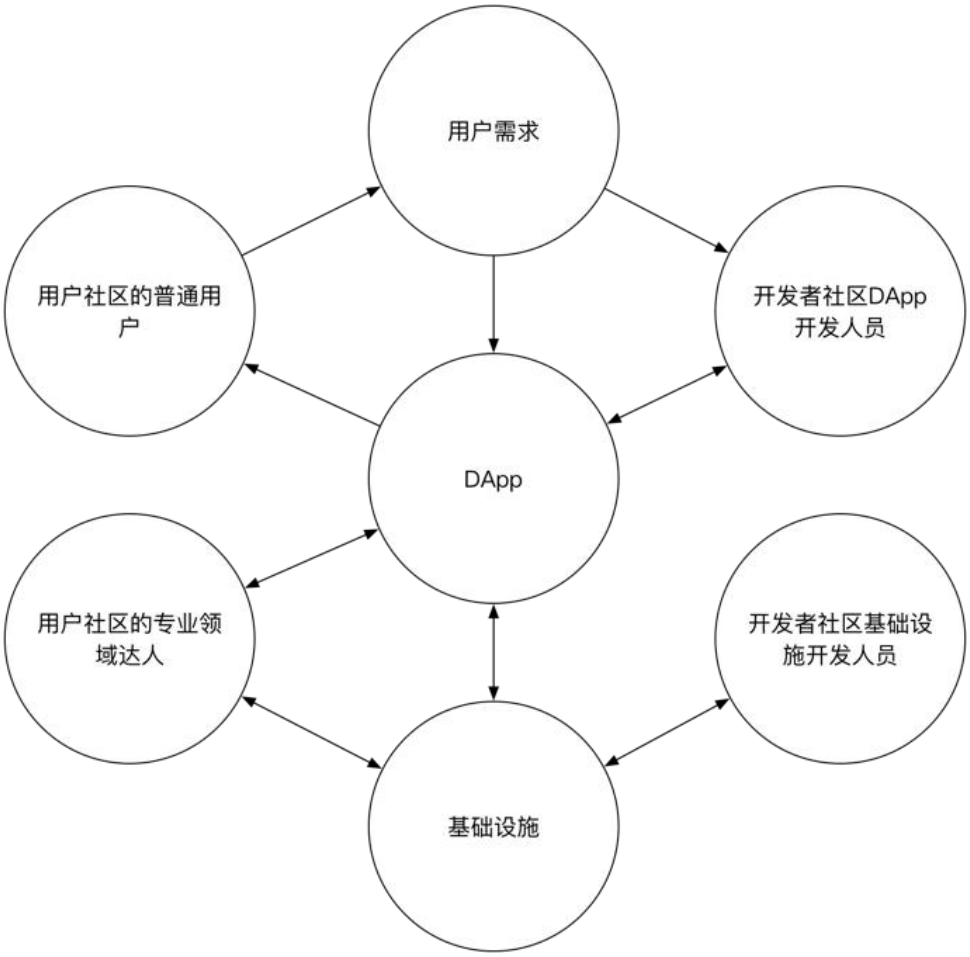

图 29 社区图

Ambr 将投入 30%的 token 完全用于社区建设。这是史无前例的投入力度，Ambr 对社区的重视程度，将超越任何以往区块链项目。

Ambr 希望吸引最出色的开发人员以及最优秀的意见领袖加入。在未来，社区将类似于人类社会中的议会，这里集中了整个区块链生态中最杰出的人群。整个 Ambr 的开发任务，重大决策，发展方向都将完全的依赖社区。

为此，对于 Ambr 来说，建设一个活跃、优秀、团结且有趣的社区将是整个 Ambr 生态的重点。

为了保证用于社区建设的 token 能够公开透明，我们将在链上存证，并在社区公布每一笔用于社区建设的资金的去向。

A. 开发者社区

在 Ambr 中的开发者社区包括 Ambr 公链开发人员和与之相关的 DApp 的开发人员。Ambr 渴望更多的开发人员加入我们的社区共同开发 Ambr。

Ambr 会维持社区的活跃，并且会发布一些带解决的问题向社区寻求帮助，当社区开发人员帮助解决问题后，Ambr 将会对提供帮助的社区开发人员发放 token 奖

励其为社区所作的贡献。

Ambr 欢迎帮助 Ambr 维持技术领先的开发人员。未来 Ambr 为了保证自身技术的领先，会努力尝试各种最新的研究成果，并将最有利于 Ambr 发展的成果并入 Ambr 中，这必须依赖强大的开发者社区对新研究成果的探索。无论开发者的探索成果是否被 Ambr 所采用，都对 Ambr 的发展做出了贡献。我们将对其发放 token 以奖励其为 Ambr 所做的贡献。

Ambr 欢迎基于 Ambr 开发 DApp 的开发人员，我们会对基于 Ambr 进行开发的 DApp 提供全面的技术服务，甚至会采用发放 token 入股的方式进行支持，

B. 用户社区

Ambr 的社区并不仅仅是开发者社区，在 Ambr 的社区中，用户社区同开发者社区同等重要。Ambr 希望更多愿意帮助 Ambr 发展的人员加入。并希望社区中人员能够从哲学，经济学，法学，社会学，政治学，历史学，新闻传播学，数学等各个角度对 Ambr 进行全方位的分析和建议，以帮助 Ambr 能够在各个方面都足够出色。对于这些帮助 Ambr 发展的朋友，将予以 token 奖励其对社区所做的贡献。

Ambr 希望被更多的人所了解，因此无论是在学术期刊还是博客上，对 Ambr 进行宣传的朋友，将会视情况予以 token 奖励。

附录

[1] L. Lamport， “Time， Clocks， and the Ordering of Events in a Distributed System，” Commun. ACM， vol. 21， no. 7， pp. 558–565， 1978.

[2] M Pease， R Shostak， L Lamport. Reaching Agreement in the Presence of Faults. Journal of the ACM， 1980，27(2): 228-234.

[3] M. J. Fischer， N. A. Lynch， and M. S. Paterson， “Impossibility of Distributed Consensus with One Faulty Process，” J. ACM， vol. 32， no. 2， pp. 374–382， 1985.

[4] L. Lamport， “The Part-Time Parliament，” ACM Trans. Comput. Sys-tems， vol. 16， no. 2， pp. 133–169，1998.

[5] M. Castro and B. Liskov，“Practical Byzantine Fault Tolerance，” Proc. Symp. Oper. Syst. Des. Implement.， no. February， pp. 1–14， 1999.

[6] Satoshi Nakamoto，"Bitcoin: A Peer-to-Peer Electronic Cash System，https://bitcoin.org/bitcoin.pdf，2008. A. Back， M. Corallo， L. Dashjr， M. Friedenbach， G. Maxwell， A. Miller， A. Poelstra， J. Timón， and P. Wuille， “Enabling Blockchain Innovations with Pegged Sidechains，” pp. 1–25， 2014.

[7] T. D. Joseph Poon， “The Bitcoin Lightning Network: Scalable Off-Chain Payments， http://lightning.network/lightning-network-paper.pdf，” pp. 1–59， 2016.

[8] Gentry C.， Halevi S.，"Implementing Gentry’s Fully-Homomorphic Encryption Scheme". In: Paterson K.G. (eds) Advances in Cryptology – EUROCRYPT 2011. EUROCRYPT 2011. Lecture Notes in Computer Science， vol 6632. Springer， Berlin， Heidelberg.

[9] van Dijk M.， Gentry C.， Halevi S.， Vaikuntanathan V.， "Fully Homomorphic Encryption over the Integers". In: Gilbert H. (eds) Advances in Cryptology – EUROCRYPT 2010. EUROCRYPT 2010. Lecture Notes in Computer Science， vol 6110. Springer， Berlin， Heidelberg.

[10] López-Alt， Adriana， Eran Tromer， and Vinod Vaikuntanathan. "On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption.". Proceeding STOC '12 Proceedings of the forty-fourth annual ACM symposium on Theory of computing， Pages 1219-1234.

[11] I. Miers，C. Garman，M. Green，and A. D. Rubin，“Zerocoin: Anonymous distributed e-cash from bitcoin，” Proc. - IEEE Symp. Secur. Priv.， pp. 397–411， 2013.

[12] F. Reid and M. Harrigan， “An analysis of anonymity in the bitcoin system，” Secur. Priv. Soc. Networks， pp. 197–223， 2013.

[13] K. Bhargavan， A. Delignat-Lavaud， C. Fournet， A. Gollamudi， G. Gonthier， N. Kobeissi， A. Rastogi，T. Sibut-Pinote， N. Swamy， and S. Zanella-Béguelin， “Formal Verification of Smart Contracts，” 2016.