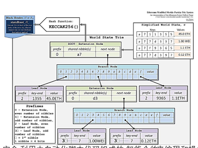


概述 BC 起源和定义:BC 技术通过区块构建 BC 网络,任何达成一致的无信任双方直接交易,不需要第三方中介的参与.**BC 本质**上是一个去中心化的分布式数据库,同时作为 btb 的底层技术,是一串使用密码学方法相关联产生的数据块,每一个数据块中包含了一批次 btb 网络交易的信息,用于验证其信息的有效性(防伪)和生成下一个区块.**BC 定义:**BC 技术是在*不完全可信*的环境中,通过构建*点对点网络*,利用*链式数据结构*来验证与存储数据,借助*分布式共识机制*来确定 BC 结构,利用*密码学*的方式保证数据传输和访问的安全,利用由自动化脚本代码组成的*智能合约*来编程和操作数据.**特点:**开放共识,去中心化,无需信任,无法篡改,可溯源,交易透明,双方匿名.**数据层:**数据层封装了 BC 的底层数据存储和加密技术.每个节点存储的本地 BC 副本(数据账本)可以被看成三个级别的分层数据结构: BC、区块、区块体(封装交易).每个级别都需要不同的加密功能来保证数据的完整性和真实性.**不可变数据:** 只能添加、不能篡改,无法篡改.不可变数据+时间刻度→互联网加上了时间轴.**加密基于密码学体系:**PKI 公钥体系,零知识体系,数字签名,数字指纹,非对称加密**网络层:**网络层封装了 BC 系统的**组网方式、消息传播机制和验证机制**,组网方式通常采用点对点**(P2P)**方式.网格网络(Mesh),权限对等、数据公开,数据分布式、高冗余存储.**共识层:**共识层主要封装 BC 系统使用的各类共识算法.BC 技术的核心优势之一就是能够在决策权高度分散的去中心化系统中使得各节点高效地针对区块数据的有效性达成共识.*常见共识算法*:PoW、PoS、BFT、DPoS 等.早期的 btbBC 采用高度依赖节点算力的工作量证明 (PoW) 机制来保证 btb 网络分布式记账的一致性**激励层:**激励层是将经济因素集成到 BC 技术体系中来,包括经济激励的*发行机制*和*分配机制*等,主要在**公有链**当中出现.在公有链中必须激励遵守规则参与记账的节点,并且惩罚不遵守规则的节点,使得节点最大化自身收益的个体理性行为与保障去中心化的 BC 系统的安全和有效性的整体目标相吻合,才能让整个系统朝着良性循环的方向发展.对私有链系统,则不一定需要进行激励,因为参与记账的节点往往是在链外完成了博弈,通过强制力或自愿来要求参与记账.以 btb 为例: btb 起初由系统奖励给那些创建新区块的矿工,该奖励大约每四年减半.刚开始每记录一个新区块,奖励矿工 50 个 btb,该奖励大约每四年减半.以此类推,到公元 2140 年左右,新创建区块就没有系统所给予的奖励了.届时 btb 全量约为 2100 万个,这就是 btb 的总量,所以不会无限增加下去.另一激励的来源则是交易费.新创建区块没有系统的奖励时,矿工的收益会由系统奖励变为收取交易手续费.交易费将被增加到该区块的激励中.只要既定数量的电子货币已经进入流通,那么激励机制就可以逐渐转换为完全依靠交易费,那么就不必再发行新的货币.**合约层:**合约层封装 BC 系统的各类脚本代码、算法以及由此生成的更为复杂的智能合约.若说数据、网络和共识三个层次作为 BC 底层'虚拟机'分别承担数据表示、数据传播和数据验证功能的话,合约层则是建立在 BC 虚拟机之上的商业逻辑和算法,是实现 BC 系统灵活编程和操作数据的基础.**应用层:**BC 技术是具有普适性的底层技术框架,除可以应用于数字加密货币外,在经济、金融和社会系统中也存在广泛的应用场景.***BC 特征:*去中心,去信任:**BC 由众多节点共同组成一个端到端的网络,不存在中心化的设备和管理机构.节点之间数据交换通过数字签名技术进行验证,无需互相信任,只要按照系统既定的规则进行,节点之间不能也无法欺骗其它节点.**开放,共识:**任何人都可以参与到 BC 网络,每一台设备都能作为一个节点,每个节点都允许获得一份完整的数据库拷贝.节点间基于一套共识机制,通过竞争计算共同维护整个 BC.任一节点失效,其余节点仍能正常工作.**交易透明,双方匿名:**BC 的运行规则是公开透明的,所有的数据信息也是公开的,因此每一笔交易都对所有节点可见.由于节点与节点之间是去信任的,因此节点之间无需公开身份,每个参与的节点都是匿名的.**不可篡改,可追溯:**单个甚至多个节点对数据库的修改无法影响其他节点的数据库,除非能控制整个网络中超过 51%的节点同时修改,这几乎不可能发生.BC 中的每一笔交易都通过密码学方法与相邻两个区块串联,因此可以追溯到任何一笔交易的前世今生.***BC 分类:*按照去中心化程度:公有链:**无官方组织及管理机构,无中心服务器,参与的节点按照系统规则自由接入网络、不受控制,节点间基于共识机制开展工作.**联盟链:**由若干机构联合发起,介于公有链和私有链之间,兼具部分去中心化的特性.**私有链:**建立在某个组织内部,系统的运作规则根据组织要求设定,修改甚至是读取权限仅限于少数节点,同时仍保留着 BC 的真实性和部分去中心化特征.**按照有无许可**: **无许可 BC:**一种完全去中心化的分布式账本技术,允许节点自由加入和退出,无需通过中心节点注册、认证和授权,节点地位平等,共享整个账本. **许可 BC:**存在一个或多个具有较高权限的节点,可以是可信第三方,也可以是协商制定有关规则,其他节点只有经过相应授权后才可访问数据,参与维护.***BC 简史*.BC1.0** 旨在解决交易速度、挖矿公平性、能源消耗、共识方式以及交易匿名等问题,参照物为 btb(BTC).**btb 运行机制:** 1.产生新交易.2.通过 P2P 网络被广播到所有的参与节点.3.各节点都会将新交易进行验证(哈希),并各自形成一个等待上链的区块.4.通过共识算法选出拥有记账权的节点.5.获得记账权的矿工通过 P2P 网络广播它的新区块,全网其它节点核对该区块记账的正确性.6. 超过一定数量的节点验证新区块无误后,就可以将这个区块连接到上一个区块上组成 BC. **BC2.0** 旨在解决数据隐私、数据存储、BC 治理、高吞吐量、域名解析、合约形式化验证等问题,参照物为 ytf(ETH).其最大的优势是加入了**部署智能合约**功能,每个人可以根据需求发布自己的智能合约.ytf 网络中的每笔交易都需要支付一定的*手续费*.无论是转账交易还是部署智能合约,所支付的手续费越高,该交易就越快地被打包进区块中,这也是以太币最主要的价值.**智能合约:**一种旨在以信息化方式传播、验证或执行合同的计算机协议,它允许在没有第三方的情况下进行可信交易,这些交易可追踪且不可逆转.***数字钱包:***数字钱包是一个形象的概念,因为拥有私钥就拥有对应地址的数字,货币,因此人们把管理密钥的软件称为'钱包'.**全节点钱包:**全节点钱包在使用时需要下载所有节点的信息.**轻钱包:**轻钱包在使用时不需要下载节点信息,但这种方式相比于全节点钱包,交易速度会降低.

	公有链	联盟链	私有链
参与者	任何人,无需许可	联盟成员,许可加入	个体或公司内部,许可加入
共识机制	POW/POS/DPOS...	分布式一致性算法	分布式一致性算法
记账人	所有参与者	联盟成员协商确定	内部自定义
激励机制	有	通常无	无
中心化程度	去中心化	多中心	中心化
主要特点	安全性高	效率和成本优化	可控性最强,不对外公开
承载能力	通常为几笔-几十笔/秒	1000-1万笔/秒	1000-10万笔/秒
典型场景	数字货币、智能合约平台	金融、政务、司法等行业	审计、日志等内部场景

大 小	字 段	描 述
4 字节	区块大小	用字节表示的该字段之后的区块大小
80 字节	区块头	组成区块头的几个字段
1~9(可变整数)	交易计数器	交易的数量
可变的	交易	记录在区块里的交易信息

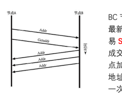
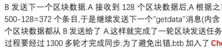
数据项	字节	字段	说明
Version	4	版本	区块版本号,表示本区块遵守的验证规则
Pre-block	32	父区块头哈希值	前一区块的哈希值,使用SHA256(SHA256(父区块头))计算
Merkle-root	32	Merkle根	该区块中交易的Merkle树根的哈希值,同样采用SHA256(SHA256())计算
Timestamp	4	时间戳	该区块产生的近似时间,精确到秒的UNIX时间戳,必须严格大于前11个区块时间的中值,同时全节点也会拒绝那些超出自己2个小时时间戳的区块
Bits	4	难度目标	该区块工作量证明算法的难度目标,压缩格式的当前目标哈希值
Nonce	4	Nonce	32位数字(从0开始),为了找到满足难度目标所设定的随机数

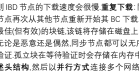
数据项	数据描述	大小
Version No	版本号,目前为 1,表示这笔交易参照的规则	4字节
In-counter	输入数量,正整数 VI = VarInt	1-9字节
list of inputs	输入列表,每区块的第一个交易称为"Coinbase"交易	<in-counter>-许多输入
Out-counter	输出数量,正整数 VI = VarInt	1-9字节
list of outputs	输出列表,每区块第一个交易的输出是给矿工的奖励	<out-counter>-许多输出
lock_time	锁定时间,如果非0并且序列号小于0xFFFFFFFF,则是指块序号;如果交易已经终结,则是指时间戳	4字节

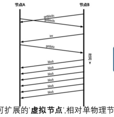
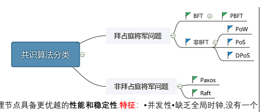
BC 应用场景:多源身份认证,分布式声誉系统,数据协同交换(数据发现、数据交易),分布式流程协作**小结 BC: 定义:** BC 是一个*分布式账本*,一种通过去中心化,去信任的方式集体维护一个可靠数据库的技术方案 **数据角度**: BC 是一种几乎不可能被更改的*分布式数据库*,分布式不仅体现在对数据的分布式存储,也体现在对数据的分布式记录 **业务角度**: BC 是多种技术的整合的结果,通过*新的数据结构、分布式共识机制、哈希加密算法*以及独特的运行机制,使得去中心化的信任机构想成为现实.

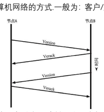
BC 数据层 区块结构*数据区块**是 BC 的基本元素,是一种记录交易的数据结构.区块体只负责记录前一段时间内的所打包交易信息,区块头记录当前区块的元数据.区块类似于账本中的账页,其物理存储形式可以是文件 (如 btb),也可以是数据库(如 ytf).区块头:***btb 系统的区块头主要封装了当前**版本号、前一个区块的地址、 Merkle 根、时间戳、当前区块的目标哈希值**、当前区块 PoW 共识过程的解**随机数**等信息.这些信息大体上可以分为三类: **引用父区块**哈希值的数据 Pre-block;当前 BC 所有交易经过哈希运算后得到的 **Merkle 根**,指向区块体所封装的交易;由**目标哈希值、时间戳与随机数**组成,这些信息都与共识竞争相关,是决定共识难度或者达成共识之后写入区块的信息.**目标哈希值**: 定义了矿工需要进行挖矿的工作量证明的难度值.根据新区块挖掘出的速度,目标值会进行调整 **Nonce**: 初始为 0.当 Nonce 小于目标值则挖矿成功**区块的主标识符**是区块头的哈希值,是使用两次 SHA256 哈希算法之后的结果,可以唯一标识一个区块.**区块高度**,将 BC 看成一个垂直的栈. 也常用来标识一个区块,但可能不唯一.***区块体:***区块体包含了当前区块的交易数量和经过验证的、区块创建过程中生成的所有交易记录.**交易**是 BC 网络中传输的最基本的数据结构,所有有效的交易最终都会被封装到某个区块中,存于 BC 上.***btb 交易过程.*元数据**: 主要存放一些内部处理的信息,包括版本号、交易大小、输入的数量、输出的数量、交易锁定时间,以及标识该交易的哈希值.可以使用该哈希指针指向这个交易.**交易的输入列表**:每笔交易的所有输入排成一个序列,每个输入的格式相同,被序列化成字节流在网上传播.**交易的输出列表**:每笔交易的所有输出也排成一个序列.每个输出的内容分成两部分,一部分是特定数量的 btb,以 '聪'为单位(最小的 btb 单位) ;另一部分是锁定脚本,即提出支付输出所必须被满足的条件以 '锁住'这笔总额.交易的所有输出金额之和必须小于或等于输入金额之和.当输出的总金额小于输入总金额时,二者的差额部分就作为交易费支付给为这笔交易记账的矿工.***UTXO:***借助前一笔交易的哈希指针,所有交易构成了多条以交易为结点的链表,每笔交易都可一直向前追溯至源头的 Coinbase 交易,向后可延展至尚未花费的交易.若一笔交易的输出没有任何另一笔交易的输入与之对应,则说明该输出中的 btb 尚未被花费,这种未花费的交易输出称 UTXO.通过收集当前所有的 UTXO,可以快速验证某交易中的 btb 是否已被花费.通过收集某人所有地址的 UTXO,可以统计他所拥有的 btb 数量.btb 系统中其实并不存在'账户',而只有'地址'(钱包).可以在 btbBC 上开设无限多个钱包地址,某人所拥有的 btb 数量是其所有钱包地址中 btb 的总和.btb 系统并不会把某人的这些地址汇总起来形成其的账户.**btb 甲到乙的转账,就是从甲的一个钱包地址转到乙的一个钱包地址上去.*一个转账交易过程:***用发起方私钥(从一个输出是发送方地址的交易中上一个 UTXO)取出 btb,并用私钥对新交易进行签名.一旦交易完成,这些 btb 就转到接收方的钱包地址中去.接收方钱包中新交易的未使用交易 UTXO 输出,只有接收方的私钥才可以打开.***使用 UTXO 优点:*** 1.UTXO 设计易于确认 btb 的所有权,可以让双重花费更容易验证.只要确认上一个交易的确获得了花费的 btb 即可.通常只要上一个交易是真实的,就的确拥有这些 btb.而 btb 系统中的交易可被认为是真实无误的.采用 UTXO 设计,验证双花只要沿着每个交易的输入逐级向上核查,直到查到这笔 btb 的创币交易即可.2.UTXO 设计与 BC 账本是完全融为一体的.BC 账本存储的是状态.微观地看,每一个 BC 中的交易都是一个状态转换函数,每一个新区块和它之前的所有区块一起形成了一个新的状态,如此重复、持续下去.在确认之后,之前的状态就不可篡改,即不可随意更改.UTXO 是与这种状态的设计相对应的 ***Merkle 树:*** btb 系统采用二叉默克尔树来组织每个区快中的所有交易,可以使用*默克尔路径*快速校验某个区块中是否有特定的交易.**交易类型:生产交易**: 又叫 coinbase 交易,每个区块的第一笔交易都是生产新币的交易.该交易没有输入地址,仅有个输出地址,其作用是将系统新生成的加密货币奖励给创造当前区块的矿工.**通用地址交易**: BC 系统中最常见的交易,由 N 个输入和 M 个输出构成,其中 N,M>0.根据 N 和 M 的不同取值,可以进一步细分为一对一转账交易、一对多分散交易、多对一聚合交易和多对多转账交易.**合成地址交易**: 合成地址交易是一类特殊交易,其接收地址不是通常意义的地址,而是一个以 3 开头的合成地址.合成地址一般 M of 模式的多重签名地址,其中 1≤N≤3、1M≤N,通常选择 N=3.合成地址的交易构造、签名和发送过程与普通交易类似,但其地址创建过程需要三对公钥和私钥,其中公钥用于创建地址、私钥用于签名.例如: 1.若 M=1 且 N=3,则 3 个私钥中任意 1 个都可以签名使用该地址上的币,这种私钥冗余可防止私钥丢失,即使其他 2 个私钥丢失也不会造成损失.2.若 M=2 且 N=3,则 3 个私钥中必须有 2 个同时签名才可使用该地址的币,常见于三方中介交易场景.3.若 M=N=3,则必须 3 个私钥同时签名才可使用该地址的币,常见于多方资产管理场景.***BC 的运行流程:***①源节点创建交易并验证目的节点的地址 ②源节点对交易进行签名加密 ③源节点将该交易广播至全网其他节点④全网节点接收交易并验证其有效性,直到该交易被全网大多数节点验证和接受⑤交易被暂存于节点内存池,并判断是否为孤立交易⑥交易被打包至节点本地区块中⑦全网共识结束后,获胜节点将其本地区块追加到主链⑧交易在主链上被越来越多的后续区块确认.四个环节:**交易生成、网络传播与验证、共识出块、激励分配 交易生成:** 源节点创建交易,将目的节点的公钥作为交易的参数,使用自己的私钥对新交易签名**网络传播与验证:**btb 网络是 P2P 网络,使用Gossip 协议进行交易的传播.每个节点收到交易后都会独立对其有效性进行验证,验证通过后才会中继转发到其他节点.通过验证环节,有效抵御了恶意交易、垃圾信息的传播和拒绝服务攻击.**交易池管理:交易池:** 一个内存池用于存放待确认打包的有效交易.**孤立交易池:** 暂时存放缺失父交易的子交易.交易池导致的交易拥堵和低手续费交易不能及时确认.**交易费与优先级:** 交易优先级=(sum_每个输入对应的 UTXO UTXO 交易额 × UTXO 存在时间)/交易字节长度.交易字节长度=148×输入数+34 输出数+10.btb 系统采用 0.576 作为交易的基准优先级,交易优先级低于该值则会被收费**共识竞争与构建区块:**btb 采用**工作量证明(PoW)**共识算法,其核心思想是通过引入分布式节点的算力竞争来保证数据一致性和共识的安全性.各矿工节点基于各自的计算机算力相互竞争来共同解决一个*求解复杂*但*验证容易*的 SHA256 数学难题(即挖矿) ,最快解决该难题的节点将获得区块记账权和系统自动生成的 btb 奖励.该数学难题可表述为: 根据当前难度值,通过搜索求解一个合适的随机数(Nonce)使得区块头各元数据的 SHA256 哈希值小于或等于目标哈希值.btb 系统通过灵活调整随机数搜索的难度值来控制区块生成的平均时间为 **10 分钟**左右.PoW 共识中每个矿工重复执行以下步骤 1 至 4,最快搜索到符合要求的随机数 Nonce 的矿工取胜并获得记账权: 1.搜集当前时间段的全网未确认交易,并增加一个用于发行新 btb 奖励的 Coinbase 交易,形成当前区块体的交易集合 2.计算区块体交易集合的默克尔根记入区块头,并填写区块头的其他元数据,其中随机数 Nonce 置零 3.随机数 Nonce 加 1,计算当前区块头的双 SHA256 哈希值,若小于或等于目标哈希值,则成功搜索到合适的随机数并获得该区块的记账权;否则继续直到任一节点搜索到合适的随机数为止 4.若一定时间内未成功,则更新时间戳和未确认交易集合、重新计算默克尔根后继续搜索**难度和难度调整机制**:难度是 BC 系统(特别是 PoW 类型的公有链系统)的重要参数,用来度量矿工成功挖到下一个区块的难易程度.符合要求的区块头哈希值通常由多个前导零构成,目标哈希值越小区块头哈希值的前导零越多,成功找到合适的随机数并 '挖'出新区块的难度越大 **目标值计算公式:** $target = coefficient * 256^{exponent-3}$ (十六进制的前两位为幂).当前区块难度值=创世区块 Target/当前区块 Target 创世区块 $Target = 0x00ffff * 256^{0x1d-3} = 65536 * 256^{26}$ 难度的调整在每个节点中独立自动发生.每 2016 个区块产生后,所有节点都会按统一的公式自动调整: **新难度=旧难度×(过去 2016 个区块的实际时间/20160 分钟) 分叉处理与主链判定:**若多个矿工节点在同一时间段内成功搜索到符合要求的随机数并生成区块,则这些节点都可以认为自己在共识竞争中获胜并向 btb 网络中广播其构造的区块,从而产生在同一区块高度出现多个不同的区块的情况,即分叉.为保证 BC 系统中仅有唯一的主链,必须定义合适的主链判定准则来从多个分叉链中选择出唯一的主链,发现孤块的情况.采用该原则,在孤块上的节点也不会得到相应的 btb 奖励.由于孤块的存在,BC 的形状并非单一的 '链条',而是树状结构,其中每个共识轮次对应的时间点上仅有唯一区块是有效的,因而树状结构中也仅有唯一的主链条.**最大工作量原则判断主链:1** 若不同分支的区块高度不同,则选择最长区块链,即选择高度最大的分支作为主链 **2** 若高度一致,则选择难度系数最大的分支作为主链 **3** 若高度和难度系数均相同,则选择接受时间最早的分支为主链 **4** 若上述所有评判系数均相同,则等待新区块产生并连接到某个或者多个分支、区块高度增加后,重复步骤 1-3 直至选出主链.此时,生成新区块的节点即可对其前序分支进行'投票',并链接至最有可能成为主链的分支子链上.***数据层关键技术:时间戳:***维基百科定义时间戳是指格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒(北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒)起至现在的总秒数.时间戳是一份能够表示一份数据在一个特定时间点已经存在的完整的可验证的数据.**功能:**为了记录某件事情的发生日期与时间,以及证明事实存在并保证先后关系.基于文档时间戳的数字公证服务以证明各类电子文档的创建时间,由此保证数据的**可追溯与不可篡改**.时间戳服务器对新建文档、当前时间及指向之前文档签名的哈希指针进行签名,后续文档又对当前文档的签名再进行签名,如此形成了一个基于时间戳的证书链,该链反映了文件创建的先后顺序,且**链中的时间戳极难改.BC 中时间戳:**时间戳服务器: 通过把以区块形式存在的一组 btb 交易实施哈希运算并加盖时间戳,并在btb网络中广播该哈希值.这个时间戳证明在该时间这个交易是存在的,因为只有数据只有在该时间才能得到相应的哈希值.每个时间戳的哈希值包含了前一个时间戳,后续的时间戳是对之前时间戳的增强,形成了一个环环相扣的时间戳链.对其中一个时间戳的篡改的代价极大,需要同时篡改其后生成的所有时间戳.**btb 系统的两个防止节点恶意修改本地时间规则:**①btb 节点会与其连接上的所有其他节点进行时间校正,且要求连接的节点数量至少为 5 个,然后选择这群节点的时间中位数作为时间戳,该中位数时间(称为网络调整时间)与本地系统时间的差别不超过 70 分钟,否则不会更新并会提醒节点更新本机的时间;②合法的时间戳必须大于前 11 个区块时间的中位数并且小于 btb 节点的网络调整时间+2 小时.换言之特币节点会拒绝接收时间戳不在此时间范围内的区块.**哈希函数:**哈希函数的输出值成为哈希值或数字摘要 **哈希碰撞:**理论上哈希碰撞不可避免,实际上输出字符串长度可以使得发生哈希碰撞的概率极低.**技术特征:抗原像:** 也称单向性,即对任意给定的 y 来说,找到任意原像 x 使得 H(x)= y 在计算上是不可行的,即对任何预定义的输出数据,无法反推其输入数据,因此,哈希函数可以看作是一类只有加密过程而没有解密过程的 '单向'加密函数.**抗第二原像或称弱抗碰撞性**: 即给定输入数据 x1 时,寻找其他不等于 x1 的数据 x2,使得 H(x1)=H(x2)在计算上是不可行的.**强抗碰撞**: 寻找任意两个不同的输入 x1 和 x2,使得 H(x1)=H(x2)在计算上是不可行的.**谜题友好**: 对于任意 n 位输出 y 来说,假设 k 是从具有较高不可预测性的高阶最小熵分布中选取的,则无法找到有效方法可在比2的 n次方小很多的时间内找到x,使得H(k||x)=y成立.**雪崩效应**: 输入数据发生任何细微变化,哪怕仅有一个二进制位不同,也会导致输出结果发生明显改变.**定长/定时性**: 不同长度输入数据的哈希过程消耗大约相同的时间且产生固定长度的输出值.**在 BC 中的应用:完整性校验**: 哈希函数的单向性和抗碰撞性通常可以用于校验消息的完整性,以防止消息在传输和存储的过程中出现未经授权的篡改.**数据要素管理**: 哈希函数的抗碰撞性使其可以作为任意数据的 '**数字指纹**',从而可以利用数据的哈希值来对其进行高效管理.例如,BC 系统的公钥、私钥、地址交易 ID、区块 ID 等要素均是通过哈希算法生成并加以标识;BC 交易数据的重要组织方式——默克尔树、BC 系统的数字签名等主要操作也均是利用哈希函数来完成.**共识竞争**: 大多数 BC 系统,特别是基于 PoW 共识的公有链系统,都是利用大量的哈希函数运算来确定共识过程中的记账节点.这主要是利用哈希函数的谜题友好性,使得矿工除了付出大量算力资源执行哈希运算之外,没有其他捷径可以对 PoW 共识过程进行攻击.**默克尔树:**作用: 快速归纳和校验区块数据的存在性和完整性.优点:1.区块头仅需包含根哈希值 2.支持 '简化支付验证',即在不运行完整 BC 网络节点的情况下也能对交易数据进行校验(轻节点)Hash 函数采用 SHA256 算法,二叉 hash 树(*交易数量为奇数时,最后一个交易重复*)通过**默克尔路径**可快速验证某个区块是否存在指定交易 **SPV 节点验证支付:** 基本思路-首先根据待验证交易信息向 BC 网络发起查询请求(称为默克尔区块消息);其他有完整 BC 数据的节点收到该请求后,利用待验证交易信息在其本地 BC 数据库中查询,并将获得的验证路径返回给 SPV 节点;SPV 节点利用该验证路径再做一次校验,若确认无误,即可认为该交易是可信的.**具体步骤:**1) SPV 节点获得待验证交易信息,向 BC 网络发起 Merkle Block Message 查询请求 2) 其他有完整 BC 数据的节点收到请求之后,执行: 1.定位包含该交易的区块 2.检查该区块是否属于整个网络中的最长链 3.取出所有交易生成默克尔树,利用 getProof 方法获得该交易的验证路径 4.将验证路径发送回请求源 SPV 节点 3) SPV 节点获得验证路径后,执行: 1.同步 BC,确保是整个网络中最长的一条 2.先拿默克尔根去 BC 中查找,确保该默克尔根哈希是在链条中 4.利用获得的验证路径,再进行一次默克尔哈希校验,确保验证路径全部合法,则交易真实存在 4.根据该交易所在区块头的位置,确定该交易已经得到多少个确认 **非对称加密:**常用的非对称加密算法是 RSA 算法.它的原理是: 两个超大素数相乘得到的结果几乎无法反过来分解还原得到原本的素数,从而实现可加密而不易破解.**btbBC 网络则采用了椭圆加密算法.BC 中的应用:**私钥证明了用户对于账户的所有权,若用户想要使用某个账户中的 btb,只有用户拥有该账户对应的私钥,才能如愿使用.在登录认证的场景中,用户输入私钥登录,客户端使用私钥加密登录信息后发送给服务器,服务器接收后采用对应的公钥解密认证登录信息.•在 btb 交易中用户使用私钥对交易进行签名,交易信息广播后,验证节点通过公钥对信息进行解密,从而确保信息是由 A 发送的•BC 网络充分利用了非对称加密的特性,一是使用其中一个密钥加密信息后,只有另一个对应的密钥才能解开;二是公钥可以向其他人公开,公钥不能逆推出私钥,保证了账户安全性.**数字签名:**一种证明数字消息、 文档或者资产的真实性的数学方案,作用 1.身份认证,使接收者有理由相信其接收到的消息是由已知的发送者发出的 2.不可抵赖,发送者无法否认其曾经发送过 3.完整性,该内容在传输过程中未被篡改(完整性) **定义**: 附加在数据单元上的一些数据,或是对数据单元所做的密码变换.这种数据和变换允许数据单元的接收者用以确认数据单元来源和数据单元的完整性,并保护数据,防止被人进行伪造.**数字签名方案的模型**通常可以表示为七元组(M,S,SK,PK,Gen,Sign,Verify): ①M: 某字母表中串的集合组成的明文消息空间,即待签名消息内容集合;②S: 可能的签名空间;③SK: 签名私钥空间,即用于生成签名的私钥集合;④PK: 验证密钥空间,即用于验证签名的公钥集合;⑤Gen:N→SK×PK: 密钥生成算法,可生成一对匹配的公钥 pk 和私钥 sk;⑥Sign:M×SK→S: 签名算法,利用私钥 sk 生成消息 m 的签名 s;⑦Verify:M×S×PK→{True,False}: 验证算法,利用公钥 pk 验证消息的签名 s 是否正确.数字签名的模型要素.**多重签名:** btb 系统一般采用'N 选 M'的形式,即该多重签名地址共有 N 个私钥,至少需要其中 M 个私钥共同签名才能从这个地址中转账.

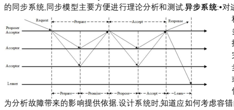
BC 网络层 P2P 网络:定义:在 P2P 网络中,对等节点拥有同等特权,每个节点将其部分资源(如计算、存储、网络带宽)直接提供给其他节点使用,而不需要中央服务器进行中间协调.每台计算机既是服务器,又是客户端;去中心化: 网络中无需中央管理节点的存在.**特点:**•可扩展性: 节点可以自由加入和退出,P2P 网络的自组织、自配置和自动负载均衡特性,破解了 C/S 模式下中心服务器的性能瓶颈问题.•健壮性好: 服务和资源分散于各个节点,不存在第三方,无明显弱节点.•高性价比: 有效利用分散于网络中大量节点上的空闲资源.•私密性: 信息传输无需通过集中节点,所有节点都具备中继转发能力,大幅提高了通信的匿名性,个人隐私得到保护.•均衡性: 资源和处理能力分布于多个节点,避免网络流量过于集中.**P2P 网络分类:混合式对等网络**•C/S 和 P2P 两种模式的混合,反映了早期网络从 C/S 到 P2P 的过渡•混合式对等网络最具影响力的代表是 Napster•早期的 P2P 网络不是完全去中心化的**无结构对等网络**•网络的节点是无固定网络结构图,无中心节点•节点地址没有统一标准•典型的无结构 P2P 网络协议如 Gnutella,它是纯粹意义上的 P2P 网络**结构化对等网络**•一般采用哈希函数将节点地址规范为标准的标识•内容的存储位置与节点标识之间存在映射关系,可以实现有效的节点地址管理•最具代表性的经典模型和应用体系如 Chord、Pastry 等 **P2P 与覆盖网(OverlayNetwork)**•P2P 网络通常构建在更底层的物理网络之上,并为特定应用提供支持,是典型的**覆盖网络**(OverLayNetwork).覆盖网络(OverLayNetwork): 建立在另一个网络上、并为更高层应用提供支持的**中间层网络**.覆盖网络的**作用**: 使得上层应用无需过多考虑与网络有关的诸如安全、资源定位、网络标识及其分配、节点加入与退出、负载均衡等问题,将精力集中在业务功能实现上.btb 网络是一个典型的 P2P 网络,包括系统的组网方式、消息传播机制和验证机制 ***btbBC 网络.*btb 网络节点:**•btb 网络由多种类型的节点组成,其功能集合一般包括网络路由(N)、完整 BC(B)、矿工(M)、钱包(W),每个 BC 节点都参与全网路由,同时也可能包含其他功能.**全节点**: 拥有完整的、最新 BC 数据的节点称为'全节点',这样的节点能够独立自主地校验所有交易 **SPV 节点/轻量级节点**: 只保留区块头数据,通过'简易支付验证'方式完成交易验证的节点称为'SPV 节点',没有 BC 的完整拷贝.**组网方式:**新 BC 节点加入 BC 网络通过以下五种方式: **地址数据库:**网络节点的地址信息由地址管理器存储在地址数据库中.结点启动时由地址管理器载入(节点第一次启用无法使用这种方式)**通过命令行指定:**用户可以通过命令行将指定节点的地址传递给新节点.**DNS 种子,硬编码种子地址,通过其他节点获得**(节点间通过 getaddr 和 addr 交换 IP 地址)**节点发现过程:**1.用户 btb 程序启动时,并不知道任何活跃的全功能节点的 IP 地址 2.为了接入网络和发现这些地址,程序会向 DNS 地址(种子)发出查询请求 3.DNS 服务器返回的响应包(DNS 的 A 记录)中会包含一个或多个全功能节点的 IP 地址***数据传输协议:***节点间采用 TCP 协议通信,端口 8333.一个通用的 **BC 网络**包括如下**核心场景**: ①节点入网建立初始连接②节点地址传播发现③矿工、全节点同步区块数据④客户端创建一笔交易⑤矿工、全节点接受交易⑥矿工、全节点挖出新区块,并广播到网络中⑦矿工、全节点接收广播的区块**消息(Message)**消息是不同节点间信息传输的**基本单位**,协议体现为消息格式的约定和时序.**基本格式**: 消息头+消息体•**起始字符串**: 奇异数(MagicNumber: 0xf9beb4d9),用于标识下一个消息的开始•**命令名**: ASCII 形式的命令码,表示消息体中**消息体大小**: 以字节为单位,消息体的长度•**校验**: 校验值•**消息体 建立初始连接:**•节点 A 通过发送 Version 消息到远端对等节点节点 B 表示连接成功,该消息包括当前节点的版本信息、区块和当前的时间等.•节点 B 收到后检查版本,兼容则建立连接 Verack 消息,也会向节点 A 发送它的 Version 消息•节点 A 收到后检查版本,兼容则回复 Verack 消息,连接成功建立.**地址广播及发现**•成功连接后,新节点 A 向相邻节点 B 发送包含自身 IP 地址的 Addr 消息.相邻节点会将此 Addr 消息再度转发给各自相邻节点,保证新结点 A 可被更多节点获知.•节点 A 可以向其相邻节点 B 发送 Getaddr 消息,节点 B 会回送若干地址信息,如其它节点(Peer)的 IP 地址、端口等数据.**同步区块数据(全节点)**•一个全节点连接到对等节点之后,首先需要构建完整的 BC.该节点向相邻节点发送 version 消息,该消息中包含 BestHeight 字段标示了自己的区块高度通过互相发送 version 消息,对等节点就可得知双方的区块数量.•对等节点还会发送 getblocks 消息,该消息中包含了本节点所有 BC 顶端的区块 hash 值,若一个节点收到的 hash 在自己的 BC 中不属于顶部,那么就代表自己的链比较长.•拥有更长 BC 的节点会识别出其他节点需要补充的块,从而发送库存 inv(inventory)消息,这个消息会包含待发送的区块 hash 值,从而告知其他节点这些块的存在.收到库存消息的节点发现自己缺少块,就会向周围节点(未必是发送库存节点)发送 getdata 消息,来请求具体某些块的数据,从而补全自己.收到 getdata 消息的节点再把节点请求的块数据发送出去.**同步区块数据(SPV 节点)**•SPV 节点同步的不是区块数据,而是区块头.•使用 getheaders 和 headers 消息(类似建立初始连接).**初始区块下载 IBD** 通过下载区块、区块头、以及交易,btb 的所有区块交易在本地就有了副本,成为一个完整的 btb 网络节点,完全跟 btb 网络同步.通常要求下载和验证此时网络中**最长、正确**的区块序列(BC),该链从**编号为 1 的区块开始**;一般**新加入的节点或者节点离线 24 小时以上**,均需执行这个操作,方可接入 btb 网络,才能确认和验证新的交易和新近挖出的区块.btbBC **数据下载方法块优先**假设有两个节点 A 和 B,其中 B 是已经完成区块同步的对等体即是提供数据的正常节点,A 是一个刚启动的节点,A 上除了创世区块以外没有任何其他区块数据.A 使用如下的块优先模式从 B 上同步自己想要的区块数据①A 向 B 发送一个'Getblocks'消息.HeaderHashes: 区块 0 的哈希值(大端模式),表示自己最新的区块是创世区块.StoppingHash:填写全 0,表示想要同步到最新区块.②B 收到消息后,根据要求返回'inv'消息给 A 节点.节点 B 返回有 500(上限)条内容的 inventory 清单集目: Type 字段是 block,表示这是区块,UniqueIdentifier 字段是区块哈希值.**这些区块哈希的顺序很重要,它代表的是区块的顺序**.③IBD 节点 A 使用接收到的清单发送 **getdata 消息**向同步节点 B 请求 128 个块(上限),对于块优先节点,按顺序请求很重要,因为**每个块头均引用前一个块的头哈希**.④收到 getdata 消息后,同步节点 B 将回复所请求的每个块,每个块都以序列化的块格式放入并以单独的 block 消息发送.**发送的第一条消息 block message 如下**.⑤节点 A 对接收到的每个块,对其进行验证,并**保持最多 128 个块**的下载队列.•A 收到这个区块数据后,对这个区块的合法性进行验证,同时等待 B 发送下一个区块数据.A 接收到 128 个区块数据后,A 根据之前接收到的 B 的 inv 消息里的清单还剩下 500-128=372 个条目,于是继续发送下一个"getdata"消息(内含 128 个区块哈希)给 B,**如此循环**一直到 500 个区块数据都从 B 发送给了 A.这样就完成了一轮区块发送任务.•btb 目前的区块数已经达到了 67 万多,这个过程要经过 1300 多轮才可完成同步.为了避免出错,btb 加入了 **Checkpoint 功能**,Checkpoint 就是指定一个区块高度的区块哈希必须等于某个哈希值.块优先的**优点**简单 **缺点** IBD 节点的所有下载都依赖于单个同步节点和块发送的顺序性.这导致: **速度限制**: 所有请求均发送到同步节点,因此,若同步节点的上传带宽有限,则 IBD 节点的下载速度会很慢.**重复下载**: 同步节点可以向 IBD 节点发送非最佳(但有效)的 BC,而也使 IBD 节点再次从其他节点重新下载最佳 BC 下载.**磁盘空间浪费**: 与重复下载密切相关,若同步节点发送了一个非最佳(但有效)的块链,该链将存储在磁盘上,浪费空间,并可能在磁盘驱动器中填充无用的数据.**高内存使用**: 无论是恶意还是偶然,同步节点都可以无序发送块,创建孤立块,直到接收并验证其父块后才能对其进行验证.孤立块在等待验证时会存储在内存中,这可能会导致较大的内存使用.**头优先:**•先尝试下载链中区块的**描述头结构**,然后以**并行方式**连接多个网络节点下载区块.•头优先是 A 使用 **GetHeaders 消息发起**,B **返回 Headers 消息**(最多包含 2000 条清单)•A 在收到 B 的'headers'消息后,验证这些区块哈希的合法性,同时: 1.继续同步剩下的 blockheaders.(目前 btb 区块头大约 50M 左右)2.开始同步区块数据.(使用'getdata'和'data'消息传输区块数据)•btb 默认最大连接数是 8 个,可以同时从 8 个节点并行下载区块数据.并行下载区块的时候,为了在多个节点之间分摊负载,每次从一个节点最多只请求 16 个区块,这就意味着每次最多下载 8*16=128 个区块.btb 还使用最大 1024 个区块的下载移动窗口,这样来最大化下载速度.下载移动窗口之前的区块都是被验证过的合法的区块,而窗口里的区块,只要下载了就马上进行验证,等全部下载完了,基本上也就验证完了.**交易广播** btb 系统的交易数据传播协议核心步骤: ①btb 交易节点将新生成的交易数据使用 **inv 消息**向全网所有节点进行广播②每个节点发送 **getdata 消息**请求 inv 消息中所有交易的完整信息,收到的节点使用 tx 发送交易.接收节点将收集到的交易数据存储到一个区块中③每个节点基于自身算力在区块中找到一个**具有足够难度的工作量证明**④当节点找到区块的工作量证明后,就向全网所有节点广播此区块(block 消息)⑤仅当包含在区块中的所有交易都是有效的且之前未存在过的,其他节点才认同该区块的有效性⑥其他节点接受该数据区块,并在该区块的末尾制造新的区块以延长该链条.**新块广播**•当一个矿工节点发现新的区块后,它需要将此区块在全网尽可能大的范围内广播,有两种方式: **1.主动推送**:•向每个完整节点对等发送一条带有新区块的'block'消息.•矿工使用这种方式可以合理地绕过标准中继方法•它知道所有对等方都没有刚发现的块 **2.区块中继**•不主动推送•发出 **Inv 消息**,在 Inv 中携带新区块消息•由收到 Inv 消息的节点把区块取走.**检测节点存活 ping 消息**用于确认接收方是否仍处于连接状态.通过发送 ping 消息,可以检测节点是否存活.接收方通过回复 pong 消息,告诉发送节点自己仍然存在.•默认情况,任何**超过 20 分钟未响应** ping 消息的节点会被认为该节点已经从网络中断开.***数据验证机制***当新区块在 BC 网络传播时,每个接收到区块的节点都将对区块进行**独立验证**,验证**通过的区块才会进行转发**,从而尽早杜绝无效或者恶意数据在网络传播,预防小部分节点串通作恶导致无效区块被网络接受,尽最大可能保证网络中传播区块的正确性.•以 btb 网络为例,节点收到邻近节点发来的数据后,其首要工作就是**验证该数据的有效性**.矿工节点会收集和验证 P2P 网络中广播的尚未确认的交易数据,并对照预定义的标准清单,从数据结构、语法规范性、输入输出和数字签名等各方面校验交易数据的有效性,并将有效交易打包到当前区块中**数据验证清单**①验证**区块大小**在有效范围②确认**区块数据结构(语法)的有效性**③验证**区块至少含有一条交易**④验证**第一个交易是 coinbase** 交易(PreviousTransactionhash 为 0 且 PreviousTxout-index 为-1),有且仅有一个⑤验证**区块头部有效性**: 1.确认区块版本号是本节点可兼容的 2.区块引用的前一区块是有效的 3.区块包含的所有交易构建的默克尔树是正确的 4.时间戳合理 5.区块难度与本节点计算的相符 6.区块哈希值满足难度要求⑥验证**区块内的交易有效性**,具体检查列表如下: •检查交易语法正确性•确保输入与输出列表都不能为空•lock_time 小于或等于 INT_MAX,或者 nLockTime 和 nSequence 的值满足 MedianTimePast(当前区块之前的 11 个区块时间的中位数)•交易的字节大小大于等于 100•交易中签名数量小于签名操作数量上限(MAX_BLOCK_SIGOPS)•解锁脚本(scriptSig)只能够将数字压入栈中,并且锁定脚本(scriptPubkey)必须要符合 isStandard 的格式(拒绝非标准交易)•对于 coinbase 交易,验证签名长度 2 至 100 字节•每一个输出值,以及总量,必须在规定的范围内(不超过全网总币量,大于 0)•对于每一个输入,若引用的输出存在于内存池中任何的交易,该交易将被拒绝•验证孤立交易: 对于每一个输入,在主分支和内存池中寻找引用的输出交易,若输出交易缺少任何一个输入,该交易将被认为是孤立交易.若与其匹配的交易还没有出现在内存池中,那么将被加入到孤立交易池中•若交易费用太低(低于 minRelayTxFee 设定值)以至于无法进入一个空的区块,则交易将被拒绝•每一个输入的解锁脚本必须依据相应输出的锁定脚本来验证•若不是 coinbase 交易,确认交易输入有效,对于每一个输入: 验证引用的交易存于主链;验证引用的输出存在于交易;若引用的是 coinbase 交易,确认至少获得 COINBASE_MATURITY(100)个确认;确认引用的输出没有被花费;验证交易签名有效;验证引用的输出金额有效;确认输出金额小于等于输入金额(差额即为手续费)•若是 coinbase 交易,确认金额小于等于交易手续费与新区块奖励之和 ***BC 分叉:自然分叉***,机器共识过程产生的临时分叉.btb 网络是一个去中心化的 P2P 网络.固有的节点地域分布、网络传输延迟,造成节点接收新区块存在一定时间差异.当两个不同节点近乎同时发掘出新区块 A、B 并进行广播的时候,就会造成区块 BC 分叉的发生.区块是**并发产生**的,并且具有广播延迟,**高出块率会导致更多的分叉**.分叉浪费网络和处理资源,**降低安全性•最长链规则不再安全**:若最长链上面只有 10%的区块,攻击者只需要超过 10%的算力就可以改变最长链! **同步账本的时间远远小于出块时间,否则就会增加分叉的概率.人为分叉**,人的共识失败产生的分叉(BIP): **'软分叉'是向前兼容的**分叉,新规则下产生的区块可被未升级的旧节点所接受,旧节点只是无法识别、解析新规则.新、旧版本互相兼容,软分叉对整个系统的影响较小.‡ **'硬分叉'是不向前兼容的**,旧版本节点不会接受新版本节点创建的合法区块,于是新旧版本节点开始在不同的 BC 上运行,由于新旧节点可能长期并存,不像软分叉是临时的,硬分叉有可能会*长期存在的*,**分叉链的存在于其算力的大小.软分叉示例**•基于 coinbase 字段的随机数扩展,coinbasedata 大小在 2-100 字节,原本可任意定制,但 BIP-34 要求开始必须在开头包含块高度且更新块版本信息,保证了交易和区块的唯一性,有效的 2.在最近 1000 个区块中超过 75%的版本号是'2' 时,整个系统开始强制要求版本号设置为'2',且要求 coinbasedata 包含块高度,但此时版本号为'1' 的区块仍被接受 3.当最近 1000 个区块中超过 95%的版本号是'2'时,版本号为'1'的区块将不被接受,迫使最后一小部分节点进行升级.**硬分叉**•若原 BC 称为 A 版本,硬分叉产生的同源分叉链称为 B 版本,则具体可以分为如下几种情况: 1.A 版本仍然被广泛支持,B 版本算力不足消亡,即还是保留原链 2.B 版本获得广泛支持,A 版本算力不足消亡,即保留新链.3.A、B 版本都有各自的支持,同时存在,这种情况是最有符合严格意义上的硬分叉,例如 **ETH 与 ETC**,两者都有其代币,这种分叉存在一定的门槛.4.A 版本仍然被广泛支持,B 版本通过代码调整难度,小部分节点也能够让它存活.与 3)的区别在于这种分叉币几乎没有门槛,人人可以分叉.5.B 版本获得支持,A 版本调整代码,小算力也可存活.硬分叉的**过程**一般经历如下 4 个阶段: **软件分叉**: 新的客户端发布,新版本改变规则且不被旧客户端兼容,首先客户端出现了分叉 **网络分叉**: 新版本的节点在网络上运行,其发现的区块将被旧版本节点拒绝,旧版节点断开与这些新版节点的连接,因此进一步网络出现了分叉 **算力分叉**: 运行不同客户端版本的矿工的算力将逐渐出现分叉 **链分叉**: 升级的矿工基于新规则挖矿,而拒绝升级的矿工仍基于旧规则,导致整个 BC 出现了分叉 **社区分叉**: 每一种 BC 的背后都有其对应的社区、开发者、矿工等利益、信仰共同体,链的硬分叉同时也会带来对应社区的分裂: 对于数字货币持有者来说,硬分叉会让他们额外增加一笔财富(分叉链 Token)***第四章:BC 共识层分布式系统模型与共识分布式系统模型:***分布式系统是组件分布在网络计算机上且通过消息传递进行**通信和行为协调**的系统,对外呈现为一个完善的、可扩展的'**虚拟节点**',相对单物理节点具备更优越的**性能和稳定性**.**特征**: •并发性•缺乏全局时钟,没有一个全局正确的时间来协调各组件的行为•组件故障的独立性**系统模型分类(两种):结构模型** 构成系统各部分组件的位置、角色和它们之间的关系,定义了系统的各组件之间相互交互的方式以及它们映射到下面的计算机网络的方式.一般为: 客户/服务器结构、对等结构(客户/服务器模型的变种).•**基础模型** 对体系结构模型中公共属性的一种更为形式化的描述,包括: 交互模型、故障模型和安全模型.**交互模型(时序模型)**•交互模型: 1.进程之间通过消息传递进行交互,实现系统的通信和协作功能.•有较长时间的延迟.•时间是进程间进行协调的基本参照,在分布式系统中,很难有相同的时间概念.2.独立进程之间相互配合的准确性受限于上面两个因素.**同步系统:**•进程执行每一步的时间都有明确的上限和下限.•每一条消息会在已知的时间范围内确定被接收到.•本地时钟与实际时间的漂移率也在已知范围内.•可以根据超时(Timeout)来检测进程的故障.•实际系统很少有真正的同步系统,同步模型主要方便进行理论分析和测试.**异步系统:**•对进程执行速度、消息传递延迟和时钟漂移率都没有限制.•实际的分布式系统大多数是异步系统.**部分同步系统:**•对系统执行时间有一定信息,但不一定准确.•与完全同步和完全异步系统相比,部分同步系统理论还不完善.**故障模型** 计算机或者网络发生故障,会影响服务的正确性.故障模型定义可能出现的故障形式,为分析故障带来的影响提供依据.设计系统时,知道应如何考虑容错的需求.**故障类型: 崩溃故障**•节点正确运行直至崩溃.•节点崩溃后不可恢复,若其他节点可以检测到这种故障则称为'故障-停止',否则称为'崩溃'.•节点崩溃后可以恢复运行,则称为'故障-恢复'.**遗漏故障**•节点或信道未能执行本来的动作导致消息丢失.•根据消息丢失发生的所在步骤位置分为发送遗漏故障、信道遗漏故障和接收遗漏故障.**时序故障**•节点过早或过迟的响应,仅适用于*同步系统假设*.•分为时钟故障、节点性能故障和信道性能故障.**拜占庭故障**•亦称随机故障,指节点可能任意出错、错误地,甚至恶意地执行某些未经许可的动作.**安全模型**•分布式系统的模块特性以及开放性,使得它们暴露在**内部和外部的攻击之下**.•**安全模型的目的**是提供依据,以此分析系统可能受到的侵害,并在设计系统时防止这些侵害的发生.**分布式一致性(Consistency)**•分布式一致性是分布式计算的核心问题.•**定义一**: 指多个节点对某一变量的取值达成一致,一旦达成一致,则变量的本次取值即被确定.•**定义二**: 指分布式系统中的多个服务节点,给定一系列的操作,在约定协议的保障下,使它们对外界呈现的状态是一致的.换句话说,也就是**保证集群中所有服务节点中的数据完全相同并且能够对某个提案达成一致.•一致性分类**: •**强一致性**: 当分布式系统中更新操作完成之后,任何多个进程或线程,访问系统都会获得最新的值.•**弱一致性**: 指系统并不保证后续进程或线程的访问都会返回最新的更新的值.•**最终一致性**: 最终一致性是*弱一致性的特定形式*.系统保证在没有后续更新的前提下,系统最终返回上一次更新操作的值.也就是说,若经过一段时间后要求能访问到更新后的数据,则是最终一致性.**共识(Consensus)**•共识描述了分布式系统中多个节点之间,彼此对某个状态**达成一致结果**的过程.在实践中,要保障系统满足不同程度的一致性,核心过程往往需要通过**共识算法**来达成.•共识和一致性常常被认为是等价的和可互换的.•**共识侧重于分布式系统中各节点达成一致的过程及其算法,是一种手段**,而一致性则更侧重于节点共识过程最终达成的稳定状态,描述的是**结果状态**.达成某种共识并不意味着就保障了一致性(指强一致性),只能说**共识机制,能够实现某种程度上的一致性.FLP 定理和 CAP 定理 FLP 定理**: 在含有多个确定性进程的异步系统中,只要有一个进程可能发生故障,那么就不存在协议能保证有限时间内使所有进程达成一致.**CAP 定理**•网络服务不可能同时保证如下三个特点,最多只能保证两个: **一致性**: (Consistency):指强一致性,分布式系统中的所有数据备份在同一时刻必须保持同样的值.**可用性**: (Availability):集群的部分节点出现故障,系统仍可以处理用户请求,即所有读写请求在一定时间内得到响应,不会一直等待.**分区容错性**: (Partition-tolerance):出现网络分区、不同分区的节点间无法互相通信时被分隔的节点仍能正常对外服务.即允许丢失任意多的从一个节点发往另一个节点的消息.**在满足分区容错的前提下,不能同时满足一致性和可用性,只能兼顾.实际应用:**CAP 定理: 工程实践中,一般会适当放宽对特定性质的假设,例如放宽强一致性要求•弱一致性: 用户读到某一操作对系统特定数据的更新需要一段时间.•最终一致性: 在数据更新操作完成之后的某个时间点,分布式系统的数据最终达成一致.BC 系统中的共识算法也需要满足 CAP 定理: 以 btb 为代表的大多数公有链通常牺牲强一致性,同时满足最终一致性、可用性和分区容错性.•某些联盟链或私有链可能会牺牲可用性来满足强一致性和分区容错性.**拜占庭将军问题:**节点不可靠,信道可靠,同步系统(**两军问题**节点可靠,信道不可靠•经典情形下两军问题是不可解的,并不存在一个能使蓝军一定胜利的通信协议.•TCP 协议三次握手,是两军问题的工程解)注意: 1.发起提议的将军越多,造成的结果越乱 2.进攻时,存在两种可能,将军本身就是叛徒,给不同的军队发出不同指令;或有的军队叛变了,将错误指令传递给别的军队 **问题模型假设:** ◆一个网络中存在n个节点,其中第i个节点发出的消息记为vi◆每个节点都会监听其他节点发送的消息,即v1,v2,⋯,vn◆网络中存在m个*恶意节点***求解问题:**■在存在恶意节点的网络中,诚实节点能对决策问题达成一致 **求解条件:** ●一致性: 每个诚实节点必须接收到相同的消息集合v1,v2,⋯,vn->所有节点角度●正确性: 若节点i是诚实的,其他诚实节点必须以它发送的消息作为vi->单个节点角度●一致性可转化为如下条件:**无论节点i是否诚实,任意两个诚实节点所保存的其消息均为vi**->单个节点角度.转化后的求解条件均从单个节点角度出发,针对单个节点求解算法.**主从节点模式**转化条件: **交互一致性条件**•IC1>>诚实的从节点必须遵守相同的(主节点)消息•IC2>>若主节点诚实,每个诚实的从节点遵守主节点的消息.**信道假设**: A1: 每个消息都会被正确传递 A2: 每个消息接受者都知道消息发送者 A3: 消息的缺失可以被检测到**口头消息(OM)算法**(假设有 m 个非诚实节点)**OM(0)**情况:主节点向每个从节点发送消息每个从节点接收消息,若未接收则默认为缺省值.**OM(m)**,m>0 情况:(1)主节点向每个从节点发送消息v(2)对任意从节点i,其接收的消息vi记为vi,其作为主节点运行 OM(m-1)向剩余n-2 个从节点发送vi(3)对任意i和j≠i,令vj 为 OM(m-1)中从节点i从节点 j 接收的消息,从节点采用消息 majority(v1,v2,⋯,v(n-1)).**算法容错率小于1/3(当三个节点中存在一个非诚实节点时, 诚实节点的行为会不确定).诚实节点对于诚实主节点的消息达成一致,当主节点为恶意节点时,诚实节点消息也能达成一致 BC 解决拜占庭问题:**设定几个规则: 1.一个时间段内只能发起一个消息.比如 10 点到 10 点一刻,只能发出一个签名并盖上时间戳的进攻消息,别的将军想要发起的进攻消息得在 10 点一刻以后才行.2.消息传递出去以后,收到进攻消息的将军必须也要在消息上签名,确认各自的身份,并盖上时间戳,然后把这个信息拷贝下来传递给其他的将军.3.为防止有将军签假名,信息都加了密.系统中各个将军都有一个公用密码和私人密码,公用密码是公开的.A 将消息传递给 B 时,使用 B 的公用密码加密,而 B 则要用他的私人密码才能解密.B 签完名后,所有将军都可以通过他的公用密码来验证他签名的真实性.4.大家都能从各自的电脑上看到信息的传递进度.**BC 解决方式优势:**1.信息里每个将军都要签名验证身份,若有将军篡改了消息,大家就能看到是哪些将军篡改了消息.2.尽管有不一致的消息.但只要一个消息得到了 6 名或 6 名以上将军的同意,那么大家就达成了共识.综上,每个将军都有一个与其他将军实时同步的消息记录,解决了内鬼问题,能让大家轻松地达成共识.**共识过程的主流模型记账节点**通过共识过程选定的记账节点**代表节点**特定算法选举出代表矿工节点参加共识过程**矿工节点**对数据或交易进行验证、打包、更新上链**数据节点**全体数据节点,生产数据或交易**主流模型:(12 为选举共识,34 为主链共识)**1.**选主**: 选主是共识过程的核心,即通过选举、证明、联盟或混合等方式从全体矿工节点中选出记账节点的过程.2.**造块**: 第一阶段选出的记账节点根据特定的策略将当前时间段内全体节点生成的交易或数据打包到一个区块中,并将生成的新区块广播给全体矿工节点或代表节点 3.**验证**: 矿工节点或代表节点收到广播的新区块后,将各自验证区块内封装的交易或者数据的正确性和合理性 4.**上链**: 验证通过后记账节点将新区块添加到主链,形成一条从创世区块到最新区块的完整的、更长的链条.若有分叉,需根据共识算法中的主链判别标准确定主链 **共识算法的分类:算法共识**•研究在特定的网络模型和故障模型的前提下,如何在缺乏中央控制和协调的分布式网络中确保一致性,其实质是一种'机器共识'.•后续提到的 BC 共识算法均指的是'算法共识'.**决策共识**•研究无中心的群体决策中,如何就最优的决策达成一致的问题,例如关于 btb 系统扩容问题和分叉问题的社区讨论与路线选择,其实质是'人的共识' 根据**选主策略**,分为: 选举类、证明类、随机类、联盟类、混合类;根据**容错类型**,分为拜占庭容错(BFT)和非~(CFT);根据**部署方式**,分为公有链、私有链、联盟链•**选举类共识**: 即矿工节点在每一轮共识过程中通过投票选举的方式选出当前轮次的记账节点,首先获得半数以上选票的矿工节点将会获得记账权.例如 Paxos 和 Raft 等.•**证明类共识**: 也可称为 Proof o f X 类共识,即矿工节点在每一轮共识过程中必须证明自己具有某种特定的能力,证明方式通常是竞争性地完成某项难以解决但易于验证的任务,在竞争中胜出的矿工节点将获得记账权.例如 PoW 和 PoS.•**随机类共识**: 即矿工节点根据某种随机方式直接确定每一轮的记账节点.例如 Algorand 和 PoET.•**联盟类共识**: 即矿工节点基于某种特定方式首先选举出*一组代表节点*,而后由代表节点以轮流或者选举的方式依次取得记账权.特点: 代议制,例如 DPoS 等. **通常情况下,公有链基于拜占庭共识实现一致性,联盟链基于非拜占庭共识实现一致性 BC 共识协议:** 1、出块节点选举: 在出块节点选举阶段,某个节点(或多个节点)成为出块节点,提出新区块.由于分布式网络中可能存在的恶意节点及分叉块的影响,其他节点在收到新区块以后不能直接将其加入自己的本地 BC 中.2、所有节点需要利用主链共识对新区块及其构成的主链达成一致.**出块节点选举机制**和**主链共识**共同保证了 BC 数据的**正确性和一致性**,从而为分布式环境中的不可信主体间建立信任关系提供技术支撑 **POW- 工作量证明**,引入了对一个特定值的计算工作.btb 采用的是 POW,矿工们在挖一个新的区块时,必须对 SHA-256 密码散列函数进行运算,区块中的随机数列值以一个或多个 0 开始.随着 0 数目的上升,找到这个解所需要的工作量将呈指数增长,矿工通过反复尝试找到这个解.在这其中,若想要对已出现的区块信息进行修改,攻击者必须完成该区块外加之后所有区块的工作量,并最终赶上和超越诚实节点的工作量.对应数学公式 Y=SHA256(X,a)简单、干净 **POS-权益证明**,试图解决 POW 机制中大量资源被浪费的情况.这种机制通过计算你持有占总币数的百分比,包括你占有币数的时间来决定记账权.Y=HASH(X,N),挖矿难度因人而异,*持币越多则挖矿越容易***分布式一致性算法 Paxos 算法** Paxos 算法解决的问题是在一个可能发生消息延迟、丢失、重复的分布式系统中如何就某个值达成一致,保证不论发生以上任何异常,都不会破坏决议的一致性.**(非拜占庭故障)Paxos 算法节点分为三种角色: 提议者(Proposer)**负责向 acceptor 发起提案**接受者(Acceptor)**负责向提案,对提案进行回应以表示自己接受提案**学习者(Learner)**不参与前面的决策过程,只从别人那里学习已经确定的、达成一致的提案结果.*一个节点可以同时拥有这三种身份,也可以只有部分身份***若一个提案被半数以上 Acceptor 接受,它就被选定了(Chosen),并由 Learner 负责执行选定的提案**.**提案**由两部分组成: **提案编号+提案值.提案编号 id** 由 Proposer 自行选择决定,一般是**相互独立、不可重复的递增序列**.一种可选的方案是 M 个 Proposer 均被分配一个 1 到 M 之间的唯一数字,对于 iProposer,其第 n 次提案的编号可以为 M(n-1)+i.**提案值(Value)是要等待达成共识的数据**.Paxos 要保证在众多被提出的 Value 中,有且仅有一个 Value 被选定,这是该算法的核心**约束条件对 Value**: 若没有 Value 被提出,就不应该有 Value 被选定.A: 只有被提出的值 Value 才可以被选定 B: 只有一个值 Value 可以被选定 C: 除非一个值 Value 被选定,否则它不会被执行 **对提案**: 若只有一个提案被提出的话,那么这个提案应该被最终选定,Acceptor 必须能够接受*多个不同的 Value*.•P1: 每个 Acceptor 必须接受它收到的第一个提案•P2: 若一个提案(其值为 v)已经被选定,那么对于所有编号更大的被选定的提案,它们所提议的值也必须是 v.•P2a: 若一个提案(其值为 v)已经被选定,那么对于任何 Acceptor 接受的编号更大的提案,它们的值也必须是 v.•P2b: 若一个提案(其值为 v)已经被选定,那么对于任何 Proposer 提出的编号更大的提案,它们的值也是 v.•P2c: 对于任意的 v 和 n,若提案(编号为 n,值为 v)被提出,那么存在一个由大多数 Acceptors 构成的集合 S,满足下述两个条件之一: ①没有成员接受过小于 n 的提案;②成员接受过的提案中,编号最大的值为 v.(上述约束条件的关系是 P2<P2a<P2b<P2c,即若满足 P2c就可以确保满足 P2,Paxos 算法就是建立在 P2c 上)**算法流程**两个阶段: **Phase1(1)准备(Prepare)**: 一个 Proposer 创建一个提案(编号 N),并向超过半数的 Acceptors 发送包含提案编号的 Prepare(N)消息(2)**承诺(Promise)**: 每个 Acceptor 收到消息后,检查提案的编号 N 是否大于它曾接受过的所有提案的编号.若是,它会回应以 Promise(Nx,Vx)消息,承诺不会接受任何编号小于 N 的提案;否则它将不予回应.其中 Nx 和 Vx 是它曾接受过的提案中编号最大的提案的编号和值,若没有接受过提案,Nx 和 Vx 为 NULL.**Phase2(1)请求接受(Accept Request)**: 若 Proposer 收到了超过半数 Acceptors 的 Promise 消息,它需要先找到这些消息中最大的提案的值 Vn,然后向这些 Acceptors 发送 Accept(N,Vn)消息;若所有 Promise 消息中 Nx 和 Vx 都为 NULL,则 Proposer 可以选择任意的值作为 V.(2)**接受(Accepted)**: 当 Acceptor 收到 Accept(N,V)消息,它首先检查是否已承诺过编号大于 N 的提案,若答案是否,它就接受该提案 N,并发送 Accepted(N,Vn);否则就拒绝.**执行阶段**当获得半数以上 Accepted 返回后,该提案被选定,并交 Learner 执行.Learner 可以通过三种方式获取被选定的值 value: **方式一**: Acceptor 每接受一个提案,就将该提案发送给所有 Learner.这种方式可以使得 Learner 快速获取被选定的 value,但是由于每个 Acceptor 都要与每个 Learner 通信,所需的通信次数等于二者数量的乘积.**方式二**:Acceptor 每接受一个提案,就将该提案发送给主 Learner;当提案值被最终选定后,再由主 Learner 发送给其他 Learner.这种方式将通信次数降低为 Acceptor 和 Learner 的数量之和,但是主 Learner 可能会发生单点故障问题,降低了系统可靠性.**方式三**:Acceptor 每接受一个提案,就将该提案发送给一个 Learner 集合,该集合中的每个 Learner 都可以将选定的提案值发送给所有的 Learner.方式一和方式二的折中.**Learner 集合的数量越多,系统可靠性就越好,但通信复杂度也相应地越高.一轮 Paxos 算法的过程**•一轮 Paxos 只对一个值达成共识•**Acceptor 本地记录**以下几个值•**minProposal** 自身响应的提案 id 最大 prepare 请求的提案 id•**acceptedProposal** 自身响应的 accept 请求中提案编号最大的提案 id•**acceptValue** 自身响应的 accept 请求中提案编号最大的提案值.考虑单个 proposer(自己也是一个 acceptor): **Prepare 阶段**: •**1.**Proposer 向 Acceptor 发送 prepare 请求.Prepare 消息包含这次提案的 id.•**2.**Acceptor 收到后向 proposer 返回 Promise 消息.Promise 消息包含该 Acceptoraccepted 的拥有最大编号提案的值以及该提案 id(即 acceptedProposal 和 acceptedValue),若没有则留空.然后 Acceptor 将该 prepare 请求的提案 id 记为自己的 minProposal(**Acceptor 在对一个编号为 n 的提案做出响应后,不会再对编号小于 n 的 prepare 请求做出响应,也不会对编号小于 n 的 accept 请求作出响应**.因此若一个 Acceptor 在这之前已经对一个编号大于 n 的提案做出响应,则 Acceptor 在这里不会返回 promise 消息)**Accept 阶段 3.**在 Proposer 收到过半的 Acceptor 的 promise 消息后,进入 propose 阶段,向 Acceptor 发送 Accept 请求.•请求同样要带有提案 id 与提案的值 idn•请求还要带有一个值,这个值为对要达成共识的一个提案: 若之前收到的所有 promise 消息中都没有附带 acceptedValue,则这个值为 Proposer 自己选中的值;若至少有一个收到的 promise 消息中带有 acceptedValue,则从这些消息中找出 acceptedProposal 最大的消息对应的 acceptedValue•**4.**Acceptor 在收到 accept 请求之后: •首先检查这个 accept 请求的提案 idn,若 n 小于自己最后响应的其他 prepare 请求的 id 即 minProposal 则向 Proposer 返回自己的 minProposal 值•否则,该 Acceptor 会接受这个 accept 请求.Acceptor 会记下这个请求的 id 以及其包含的值作为自己的 acceptedProposal(同时也作为 minProposal)和 acceptedValue,然后同样也会向 Propose 返回自己的 minProposal•Proposer 等待收到过半 Acceptor 的响应•**若响应中包含的 minProposal 值大于自己当前的提案 id,则放弃本轮**•否则说明已经得到了过半 Acceptor 的支持,即认为已经达成了一致•**5.**在确认已经达成了一致之后,Proposer 就可以将结果发给 learner 令其记下结果•除了这种方式,也可以让 Acceptor 在每次接受一个值后就告诉 Learner,Learner 在收集到过半的相同 id 的值后就可以确定结果•还有其他方法,具体使用哪种方法取决于对通信代价和速度的取舍 **活锁**上述过程只考虑了一个 Proposer 的情况,所以一般两轮广播+收集就能直接完成.若有多个 Proposer 先后发起请求,**后一个 prepare 请求就会使前一个 prepare 请求后续的 accept 请求失效**,除非这个 accept 请求在后一个 prepare 发出之前就收到了足够的响应达成共识.这种情况可能**连续、交替**发生 proposer 每当自己的请求确认失败后马上用新的请求 id 发起新一轮 prepare,所以**二者的 prepare 不断使对方的请求失效**,这个过程可能一直持续从而形成活锁.这种情况下系统无法继续运行下去.为了解决这个问题,可在集群中**选出一个节点作为 leader,在对每条数据/每条事务达成一致性的时候,只有 leader 会作为 Proposer 而其他节点不会发起提案**.对每条事务都运行一次上述的**一轮 Paxos**,由于此时**只有一个 Proposer**,所以就不会出现活锁的情况.Leader 可以向其他节点定时发送一个心跳包来声明自己存活.一旦其他节点认为 leader 消失了,就可以发起竞选来尝试成为 leader,选出 leader 的过程也可以直接使用一轮 Paxos 算法来选出.**Raft 算法**•适用于非拜占庭容错环境中的分布式一致性算法.Raft 采用了特定的技术设计来提高算法的可理解性.模块化(领导选举、 日志复制、 安全性、 成员变更)状态空间规约**一些概念:复制状态机 RSM(ReplicatedStateMachine)**指多台机器具有完全相同的状态,运行完全相同的确定性状态机,可以解决很多分布式系统中的容错问题.复制状态机通常可以**容忍半数节点故障**.每个服务器节点都有: •**一致性模块•日志•状态机• 基于日志的复制机制**•日志记录了导致状态机中状态转换的命令序列.•状态机的状态可以通过执行日志中的命令序列获得,状态机的当前状态是可以重新计算得到.•保证不同节点间的**日志一致**(即保存有相同顺序的命令序列),即最终可以保证状态机之间的状态一致性.**思路和过程** Raft 算法采用 RSM 复制状态机模型,每个节点服务器都是一个状态机,所有的服务器都以同样的顺序响应客户端的请求.如何保证分布式一致性的问题就转换成如何**保证所有的状态机的日志一致性**的问题.其算法思路: •在集群中通过**领导选举**确定一个领导者,全权负责**复制日志的管理**.•领导者从客户端接收日志条目,将日志条目复制到其他服务器上,并且在保证安全性的时候通知其他服务器将日志条目应用到他们的状态机中.•基于领导选举的机制大大简化了日志复制的管理,例如日志数据可以直接从领导者流向跟随者

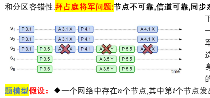
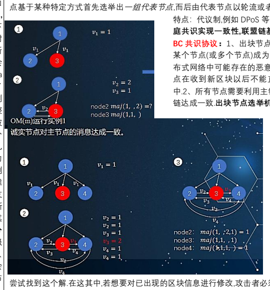



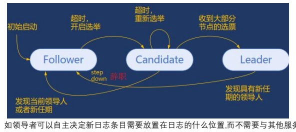


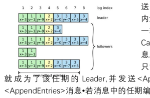



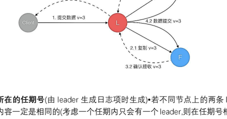




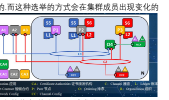



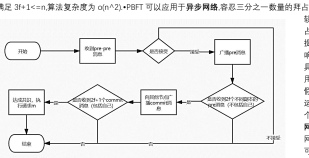






如领导者可以自主决定新日志条目需要放置在日志的什么位置,而不需要与其他服务器商议,日志数据只是从领导者流向其他服务器•当领导者宕机或者与其他服务器断开连接后,集群其他节点会启动领导选举过程并选出新的领导者.Raft 集群节点在任一时刻都处于三种状态之一: **领导者 Leader、跟随者 Follower、领导候选 Candidate**.•一般情况下,系统中会有一个节点称为 Leader,其他节点称为 Follower.由 Leader 向各个 Follower 同步自己的 log,log 中每一个 entry 都代表着系统的一个事务.(**日志复制**)•Leader 需要**定时向 Follower 发送心跳包来证明自己的存活**,一旦 Follower 等待超过了 timeout 时间还没有收到心跳包则认为 Leader 节点已经宕机,此时 Follower 就会将自己的状态变为 Candidate 参与下一轮 Leader 的竞选(**领导选举**)•集群中的成员可能出现变化,一些新的节点会加入,或者一些性能不好的节点会被移出.Raft 实现了联机更新成员的功能.(**成员变更**)**领导选举-任期 Term**Raft 引入了'任期'(Term)概念,将时间分割成**任意长度**的任期,任期由**任期号**标识,**单调递增**.任期从一次领导选举开始.•一个节点在尝试竞选成为 Leader 时,**会指定自己要成为哪个任期的 Leader**.•当任期 n 的 leader 节点宕机,等待超时的 Follower 将自己的状态变为 Candidate,申请成为下一任期的 Leader **领导选举-从每个节点的角度**:所有的服务器**初始**都处于 Follower 状态,没有 Leader,此时各个节点会等待心跳包到超时并参与选举.•Follower 定时会收到 Leader 节点所发送的心跳包以证明 Leader 仍然存活,若它在一个预定的时间周期内没有收到心跳包,表明 Leader 节点已经宕掉,这个节点就会开启一轮新的选举.并且: •将当前的**任期编号+1**•将自己的状态设为 Candidate;•投票给自己并向其他的服务器发送<RequestVote>消息.**领导选举有三种可能**:①赢得选举•在一个给定任期内一个节点只能给一个 Candidate 投票.•若收到大多数节点的选票,该节点就成为了该任期的 Leader,并发送<AppendEntries>心跳包给其他的服务器②若收到一条<AppendEntries>消息•若消息中的任期编号大于当前任期,意味着某个节点已当选为 Leader,则将自己的状态设为 Follower;否则,丢弃该消息.③选举失败•若出现多个节点同时竞选 Leader,可能导致选票分散,任一参选节点都无法获得大多数投票,选举失败.•每个 Candidate 超时后重新启动新的选举,为防止多个 Candidate 节点恰好同时参与竞选,同时超时-同时进行下一轮竞选这样的死循环出现,每个 Candidate 的**超时时间**是在一个范围内的**随机值**.**日志格式**•Raft 的日志 log 的形式上大致如图所示•每一行代表一个节点的 log,行内每一个方格代表一个 logentry.•每一行从左到右 logindex 递增,**每个 logentry 上都包含着自己所在的任期号**(由 leader 生成日志项时生成)•若不同节点上的两条 logentry 有相同的任期号和 index,则其内容一定是相同的(考虑一个任期内只会有一个 leader,则在任期号相同的情况下,两条 logentry 是由相同的 leader 构造的,若其 index 也一样则显然是同一个 entry.领导选举的过程保证一个任期只有一个 leader)每个节点都会维护一个 commitIndex 日志条目索引.**日志复制**•每当 Leader 收到了来自 client 发来的事务请求(1),就将其构造成一条新的 logentry 加入到自己的 log 中.•Leader 会将新 entry 用<AppendEntries>消息发送给各个 Follower(2),若发送失败则会不断重试.当收到大部分 Follower 的响应(表示收到了 entry,图中(3))则确认 entry 以及其之前的 entry 都可以 commit.•leader 在确认一条 entry 可以被 commit 之后,会通知各 follower 节点该 entry 可以被 commit(4.2),并会向 client 发送响应通知(4.1),表示这条事务确实已经被接收.•Entry 被 commit,代表其中事务所对应的具体内容可以被执行 **日志复制-故障**•阶段 1 和阶段 2.1 之前出故障,不影响一致性•阶段 2.1 失败(部分或全部),有一致性问题•阶段 3.1 失败(部分或全部),有一致性问题•阶段 4.1 失败,客户端重新发送请求后提交即可,无一致性问题•某些情况会出现网络分区,而导致双领导,原领导降级为 Follower **远程过程调用 RPC AppendEntriesRPC** 由 Leader 节点发出,用于复制日志.当增加日志条目数为 0 是用作心跳 heartbeat.**RequestVoteRPC**:由 Candidate 节点发出,请求其他节点给自己投票.**集群成员变更**•Raft 中 leader 选举是基于过半成员支持选出的.而这种选举的方式会在集群成员出现变化的时候导致一些问题.•如图,集群原来配置有 S1S2S3 三个节点,新配置中集群里添加了 S4S5 节点.在新配置还没有完全传到所有旧节点时可能出现一些问题.如图中箭头所指的点.•在这个时间点,S1S2 还用着旧配置,于是这两个节点形成了旧配置中的'大多数节点',当此时 S1 请求成为 leader 而 S2 同意,则 S1 会认为自己成为了一个 leader.•而 S3 已经使用新配置,S3S4S5 构成了新配置中的大多数节点,若此时 S3 请求成为 leader 而 S4S5 同意,则 S3 也会觉得自己成为了下一个 leader.•此时就出现了两个 leader.两个相同任期的 leader 就可能导致'**相同 index,相同任期而内容不同**'的一对 logentry 出现从而导致系统出现错误.为**解决**此问题,Raft 算法使用**两个事务**来完成一次成员变更.•当 Leader 收到成员变更的请求(从旧配置 C-old 到新配置 C-new),Leader 会先构造一个介于二者之间的配置 C-old,new,这个配置包含的成员是新旧两个配置的内容.Leader 会先将 **C-old,new** 作为一个新的 entry 按照之前的流程同步到各个 Follower 上.在确认这个中间配置 commit 了之后,各个节点就按照这个中间配置来行事,即节点需要同时获得新配置和旧配置中大多数人的同意才能成为 leader.在这个阶段的 leader 出现了之后,该 leader 则会再将新配置 C-new 作为新 entry 同步给其他节点.当 C-new 配置 commit 了之后,则各个节点开始按照新配置来运行.旧配置的节点自然离开集群.至此,完成成员变更.**安全性选举安全性**在任何时刻只能有一个 leader(term). **领导 Append-Only**•日志项不能被覆盖或删除,只能扩展新日志项.**日志一致性**•不同节点上的两条 logentry 有相同的任期号和 index,则其内容一定是相同的.•该 index 之前的所有日志都是相同的.**领导者完整性**•对于任意给定的任期,领导者都包含了此前各个任期所有被提交的日志条目.**状态机安全性**•若一个服务器已经在一个状态机上应用了一条 logentry,那么其他服务器都不会应用*同一个 logindex 的同一 entry*.**PBFT 共识算法**拜占庭将军问题被证明: 在将军总数大于 3f,背叛者为 f 或者更少时,忠诚的将军可以达成命令上的一致,即 3f+1<=n.算法复杂度为 o(n^(f+1)).而 PBFT 算法将复杂度降低为 o(n^2).•PBFT 可以应用于**异步网络**,容忍三分之一数量的拜占庭节点,由于相较于之前的拜占庭算法大幅提高了系统的响应效率,因此具有较强的实用性.•PBFT 算法假设共识过程运行环境是一个**异步分布式网络**(例如互联网环境).网络中可能发生消息传输失败、延迟、重新发送等问题,且节点有作恶的可能(故意发送错误消息).•PBFT 算法使用**数字签名**来防止欺骗、重发和检测消息完整性.消息包括公钥签名、消息类型编码、哈希产生的消息摘要等.后续的算法演示设**<m>σi 表示消息 m 被节点 i 签名,D(m)**表示消息 m 的摘要.**PBFT 容错率**: 设节点总数是 n,其中作恶节点有 f 个,那么剩下的正确节点为 n-f 个,这意味着只要收到 n-f 个消息就能做出决定.最坏情况下,这 n-f 个消息中有 f 个是由作恶节点冒充的,那么正确的消息就是 n-f-f 个,根据多数一致的原则,正确消息必须占多数,也就是 n-f-f>f,即 n>3f.由于节点必须是整数个,所以 **n 最少是 3f+1 个**.采用 PBFT 算法的系统**并非节点数越多越好**,要根据 f 来计算.•PBFT 中的服务被建模为**状态机**,节点分为**主节点**和**副节点**,每个节点上都保存有状态副本.所有副本的状态变迁通过视图(view)的配置更换来进行.在每一个视图中,只存在一个主节点.主节点可以简单地由**视图编号 mod|节点数量|**来决定.每个节点上的副本状态包括服务状态、消息日志(记录副本收到的消息)和当前视图编号.**两个限定条件**: ①节点是确定性的,给定状态和参数相同的情况下,执行结果相同;②所有节点必须从相同状态开始执行.即使存在失效副本节点,算法对所有非失效副本节点的请求执行总顺序达成一致.•**算法的五个阶段**: ①请求 Request②预准备 Pre-prepare③准备 Prepare④确认 Commit⑤回复 Reply(12 个阶段同一视图中请求发送的时序性,345 确保不同视图之间的确认请求是严格排序的)如图所示,n=4,f=1.①客户端 C 发送请求给主节点 0,②主节点 0 把请求广播给其他副节点 1、2、3.③由于节点 3 失效(故障或作恶),图中红线代表诚实节点的信道),节点 1、2 广播消息.④节点 0、1、2 搜集到 2f(不包括自己)的消息之后,在本地达成了 commit 条件,然后将消息广播给全网.⑤节点 0、1、2 搜集到 2f+1(包括自己)的 commit 的消息,执行请求操作,并将结果发送回客户端.⑥客户端搜集到 3 个消息,大于 f,故请求成功•在算法**开始阶段**,主节点由 p=vmodn 计算得出,随着 v(视图编号)的增长可以看到 p 不断变化.图中,节点 0 为第一轮视图的主节点.**第一阶段(请求)**: 客户端签名发送消息<REQUEST,o,t,c>σc 给主节点 0,o 为操作;t 为时间戳,用于保证 c 请求只被执行一次,也可以用于比较操作执行顺序,例如 t 可以设置为客户端发送请求时的本地时钟;c 为客户端编号**第二阶段(预准备)**: 主节点构造消息<<PRE-PREPARE,v,n,d>σ0,m>广播到集群中的其它节点,同时追加到消息日志中.•PRE-PREPARE 标识当前消息所处的协议阶段.•v 标识当前视图编号.•n 为主节点分配给所广播消息的一个唯一递增序号,m 为客户端发来的消息,d 为 m 的数字摘要.**第三阶段(准备)**: 副本收到主节点请求后,会对消息*有效性*进行检查,检查通过会追加在消息日志中,并广播消息<PREPARE,v,n,d,i>σi,其中 i 是本节点的编号.对消息的有效性有如下检查: 1.检查收到的消息体中摘要 d,是否和自己对 m 生成的摘要一致,确保消息的完整性.2.检查 v 是否和当前视图 v 一致.3.检查 n 是否接收过相同序号 n 和 v,但是不同摘要 d 的消息.4.检查序号 n 是否在水线 h 和 H 之间,避免快速消耗可用序号.(防止作恶节点消耗序号空间)**第四阶段(确认)**: 副本收到 2f(不包括自己)个一致的 PREPARE 消息后,会进入 COMMIT 阶段,并且广播消息<COMMIT,v,n,D(m),i>σi给集群中的其它节点.在收到 PREPARE 消息后,副本同样也会对消息进行有效性检查,包含上一阶段介绍的 1、2、4 三个检查步骤.**第五阶段(回复)**: 副本收到 2f+1(包括自己)个一致的 COMMIT 消息后且已经没有序号小于 n 的请求,则执行 m 中包含的操作(保证多个 m 按照序号 n 从小到大执行),执行完毕后发送消息<REPLY,v,t,c,i,r>σi 给客户 i 端.v 为视图编号,t 为时间戳,c 为客户端编号,i 为节点编号,r 为操作结果.•客户端在收到回复后要进行签名验证、时间戳比较和操作结果 r 比较.当搜集到 f+1 个一致结果的回复之后才能确定操作结果.若在客户端给主节点发送请求之后的一定时间内没有响应,客户端会广播请求,所有副本会进行响应.若响应结果发现主节点失效,则会通过**视图更换**(ViewChange)来切换主节点.**检查点消息(Checkpoint 消息)及其作用**•**Checkpoint(检查点)**: 当前节点处理的最新请求序号.前面已经提到主节点收到请求会给请求编号.比如一个节点正在共识的一个请求编号是 101,那么对于这个节点,它的 checkpoint 就是 101.•**stablecheckpoint(稳定检查点)**: 节点 i 发送<CheckPoint,n,d,i>σi给其他节点,当收到了 2f+1 个验证过的 CheckPoint 消息,比如系统有 4 个节点,三个节点都对 n 是 213 达成了共识,那么 stablecheckpoint=213.•**stablecheckpoint 作用: 最大的目的是减少内存的占用**.因为每个节点的消息日志记录下之前共识过什么请求,随着系统运行,日志数据会越来越大,所以应该有一个机制来实现对数据的删除.例如,现在的稳定检查点是 213,那么代表 213 号之前的记录已经共识过,所以之前的记录就可以删掉了.**ViewChange(视图更改)事件**当主节点挂了(超时无响应)或者副节点集体认为主节点是问题节点时,就会触发 ViewChange 事件,ViewChange 完成后,视图编号将会加 1.**ViewChange 的流程**: 从节点向其他节点广播 view-change 事件<view-change,v+1,n,C,P,i>,判断是否收到 2f 个不同节点的视图编号为 v+1 的 view-change 消息(不包括自己),否则未达成一致,是则向其他节点广播 new-view 消息<new-view,v+1,V,O>,节点验证收到的视图编号为 v+1 的 New-View 消息里的 O 集合是否正确,否结果,是则按照 pbft 流程逐条执行 O 集合里的请求消息,视图编号为 v+1(每个请求执行后不会重新返回消息给客户端),结束(v: 上一个视图编号,n: i 节点的 stablecheckpoint 编号,C: 2f+1 个节点的有效 checkpoint 信息的集合,P: i 节点中上一个 view 中编号大于 n 且到达 prepared 状态的请求消息的集合,V: 新的主节点接收到的有效视图编号为 v+1 的 view-change 消息集合,O: pre-prepare 消息集合) **Raft 与 PBFT 对比** 1Raft 2Pbft: **适用环境**: 私有链,联盟链 **通信复杂度** O(n) O(n^2) **最大故障和容错节点**故障节点: 2f+1≤N 容错节点: : 3f+1≤N **流程对比:Raft**: 初始化 leader 选举,谁快谁当;公识过程;重选 leader 机制 **Pbft**: 初始化 leader 选举(按编号依次轮流做主节点;共识过程;重选 leader 机制)PBFT 是在**联盟链共识节点较少**的情况下 BFT 的一种解决方案.**总结**•PBFT 算法由于每个副本节点都需要和其他节点进行 P2P 的共识同步,因此随着节点的增多,性能会下降的很快,但在较少节点的情况下可以有不错的性能,并且分叉的几率很低.PBFT 主要用于*联盟链*,但是若能够结合类似 DPoS 这样的节点代表选举规则的话也可以应用于公有链,并且可以在一个不可信的网络里解决拜占庭容错问题,TPS(服务器每秒处理的事务数)**PoW 共识算法**工作量证明最早是为**防止服务和资源滥用**,或者拒绝服务攻击等场景而提出的一种经济对策.一般要求证明方在使用服务或资源之前,首先完成具有一定难度或者适当工作量的复杂运算,这种工作量对于**证明方**是'昂贵的'且'没有捷径'的,但对于**验证方**是快速和简单的;BC 系统中的稀缺资源是'**区块记账权**'以及随区块发行的 **btb 奖励**.PoW 共识机制通过引入分布式节点的算力竞争来作为工作量证明,利用其算力来完成大量的哈希函数计算工作,以便选出每个 10 分钟时间窗口的唯一'记账人',从而保证 BC 账本数据的一致性和共识的安全性;工作量证明的核心技术是**哈希**.哈希函数具有单向性、随机性、定长性和定时性等特点,btb 中选择使用 SHA256 哈希算法,矿工需要找到区块头中的一个随机数 Nonce,使区块头的两次哈希结果满足以 n 个 0 开头,这一过程目前没有比**穷举法**更好的算法,因此从概率上,一般需要 16^n 次哈希才能找到结果.**btb 中的 PoW: 三大要素**: PoW 函数、区块信息、难度值;PoW 函数定义为: Fdiff(*BlockHeader*)→SHA256(*SHA256*(*BlockHeader*))<MaxTarget/*diff*,其中 MaxTarget 为 btb 系统的最大目标值,难度值 diff 为正实数,diff 越大,不等号右边的式子值越小,即得 0 越多,计算难度越大;**diff 会被系统动态调整**,使 btb 的出块速度大致稳定在 10 分钟 **btb 中的 PoW 共识过程**: 1.每个 btb 节点搜集当前时间段(最近 10 分钟)的全网未确认交易,并增加一个用于发行新 btb 奖励的 CoinBase 交易,形成当前区块体的交易集合;2.计算区块体交易集合的 Merkle 根记入区块头,并填写区块头的其他元数据,其中随机数 Nonce 置 0.这个区块就是 PoW 函数的输入数据;3.Nonce 加 1;计算当前区块头的两次 SHA256 哈希值,若小于目标哈希值,则成功搜索到合适的随机数并获得该区块的记账权;否则继续步骤 3 直到任一节点搜索到合适的随机数为止;4.若一定时间内未成功,则更新时间戳和未确认交易集合,重新计算 Merkle 根后搜索 统计意义上,**当节点拥有全网 n%的算力时,该节点总是有 n%的概率首先找到合理的随机数**.btbBC 系统的安全性和不可篡改性是由 PoW 共识的强大算力保证的.当区块头目标哈希值越小,成功找到合适随机数并挖出新区块的难度越大,任何对于区块数据的攻击或篡改都必须重新计算该区块以及其后区块的 SHA256 难题,且计算速度必须使得伪造链长度超过主链,这种攻击难度形成的成本将远超其收益.**PoW 的优势** 1.PoW 共识的架构简明扼要、有效可靠;2.PoW 可以实现某种程度上的公平性,即投入算力越多就可以等比例地增加越多的获胜概率;3.PoW 可以有效抵御 51%攻击,攻击者必须拥有超过整个系统 51%的算力,才有可能篡改 btb 账本,这使得攻击成功的成本变得非常高昂,难以实现.**PoW 的缺陷** 1.强大的算力造成了极大的资源浪费(电力);2.



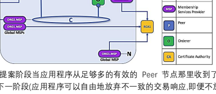



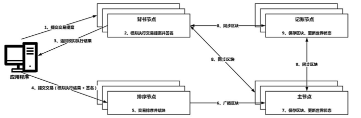
长达 10 分钟的交易确认时间使其相对不适合小额交易的商业应用;3.矿机和矿池的出现,使得算力逐渐集中在大型矿池手上,有违去中心化的初衷 **PoS 共识算法**随着 btb 系统算力的提升和挖矿设备的专业化,特别是 AISC 矿机和大型矿池的出现,使得矿工群体逐渐从持币者群体中独立出来,形成了完全不同的两个群体.同时,PoW 不仅耗能巨大,而且算力中心化问题日益凸显,整个系统的安全性逐渐不是掌握在使用者手中,而是取决于矿工和矿池.因此诞生了 PoS(权益证明)共识算法.PoS 共识中,**具有最高权益的节点**最有可能获得记账权,其权益体现为节点对特定数量货币的所有权.PoS 共识更多地是代表一种理念,实际上有多种不同的表现形式,目前主要有以下**三种 PoS 共识算法**: 1.PoS+PoW 混合共识(PoS1.0)2.纯 PoS 共识(PoS2.0)3.PoS 共识的扩展形式(PoS3.0)**PoS+PoW 混合共识(PeerCoin)**: PeerCoin(点点币)中,PoW 共识用于早期货币发行,随着系统挖矿难度上升,逐渐过渡到 PoS 共识;*两类区块*: PoW 区块和 PoS 区块;**CoinStake(币权)交易**: 为 PoS 特殊设计的交易: 交易输入数量≥1,第一个输入不能为空,称为 **Kernel**;交易输出数量≥2,第一个输出必须置空;PoS 区块的标识是第二笔交易为 CoinStake(所有区块第一笔交易都是 coinbase);PoS 中的挖矿,就是找到合适的 Kernel 交易;**激励机制**: Coinstake 全部输入的币龄总和按一定比例转化为矿工收益;**共识流程**: 节点首先从自己所有的 UTXO 中选定一个作为 Kernel,构造 Coinstake 交易,计算两次 SHA256 哈希值;若不满足上式,则重新构造 Coinstake 交易,重构过程中时间戳 nTime 会改变,同时也可以改变 Kernel,以得到不同的 Coinstake 交易.如此重复下去,知道获得合格区块为止.由上可知,由于时间戳以秒为单位,因此共识的搜索空间大大减少,同时 **Kernel 的币龄**是影响找到合格区块的最大因素;区块生成后,每个区块中的交易都会将其消耗的币龄提交给区块,获得最高消耗币龄的区块将会被选中成为主链;**权益激励**: stakeReward=(0.01×nCoinAge/365)×COIN 其中 nCoinAge 是 Coinstake 所有输入的币龄总和,这部分收益奖励给矿工;**纯 PoS 共识**(Nextcoin 未来币): NXT 是第一个纯粹应用 PoS 机制运行的数字货币,基于**账户结构**而非 UTXO,NXT 中不存在挖矿,10 亿 NXT 一次性创建在第一个区块中,之后网络靠交易费用来维护;NXT 采用透明锻造机制,通过指定锻造下个区块的节点,其余节点将交易发到此节点上,来缩短交易时间;**PoS 函数**: hit<$baseTarget \times effectiveBalance \times elapseTime$.hit: 每个区块有一个生成签名(generationSignature)字段,用户使用自己的私钥对上一区块的 generationSignature 进行签名,结果进行 SHA256 哈希,哈希结果的前 8 字节作为 hit 变量;baseTarget: 全网难度基准值;effectiveBalance: 账户有效余额,账户中具有铸币权利的货币余额;elapseTime: 当前时间与上一区块的时间间隔;显然,当一个新区块生成后,每个用户锻造下一区块的 hit 值已经成为常量,因此用户不需要挖矿,只需等待时间推移至不等式成立即可锻造区块;对于用户,其有效余额越大,同等条件下更有可能获得锻造区块的机会;NXT 规定优先选择最早生成的区块;**区块锻造流程**: 用户必须实时在线,当网络上有新区块产生时,每个账户立即计算自己的 hit,然后根据公式 elapseTime=hit/(baseTarget×effectiveBalance)计算得知自己锻造区块的期望时间值,并将这个时间广播给其他节点.当全网每个节点都知道其他节点的期望时间时,也就知道下个区块优先由哪个节点锻造,并将交易发给该节点.该节点在自己的时间窗口锻造好区块后立即广播全网,其他节点检验新区块是否有效.**优点**: 1.有效降低能源消耗;2.避免算力集中问题;**缺陷**: 1.存在'富者更富'的马太效应问题;2.存在无利害关系等安全性问题;无利害关系攻击,即由于 PoS 中挖矿代价较低,当链出现分叉时,对于一个节点来说,其利益最大化的方法是在所有链上挖矿,因此可能会导致链分叉越来越多 **DPoS 共识算法** DPoS(委托权益证明)即通过共识节点的权益投票将区块数据的记账权和 BC 参数的配置权赋予特定的少数代表节点,从而实现公平和民主的共识过程和 BC 治理,并解决 PoW 的能源消耗和 PoS 的无利害关系攻击等问题.DPoS 通常包含见证人选举和产生区块两个阶段.●**见证人选举**: DPoS 共识过程中,股东节点可以将其持有的股份权益作为选票授予一个代表,称为**见证人**.股东节点的投票权重与其持币数量成正比.DPoS 选择得票数最多且有意愿的 N 个节点进入董事会,轮流对交易进行打包和生成新区块;作为回报,见证人会获得区块奖励、交易费或系统发行的特定奖励,同时,其需要缴纳一定数量的保证金.见证人对股东负责,若其错过签署对应的区块,则股东会撤回选票将其投出董事会,因此,见证人节点通常需保证 99%以上的在线时间以实现盈利目标; DPoS 系统中还会通过选举方式产生一组特定的**授权代表**(Delegates),这些授权代表有权配置和调整 BC 系统的参数,如交易费用、区块大小等.若大多数授权代表都同意变更提案,则所有股东节点有一段时间来审查变更提案,在此期限内可以罢免授权代表并废止提案●**生产区块**: DPoS 共识过程中,见证人节点按照预先定义的顺序轮流产生区块.每个区块被签署生产之前,需要验证前一个区块已经被受信任的见证人签署.每个见证人轮流在一个固定时间内生产一个区块,若见证人没有在其时间段中生产区块,那么该时间段后这个见证人将会被跳过,由下一个见证人生产下一个区块,如此循环.**RPCA 共识算法** Ripple 协议共识算法(RPCA)是 Ripple 系统及其数字加密货币——瑞波币(XRP)所采用的的共识算法.RPCA 通过利用网络中的集体可信的子网络来克服所有节点同步通信的延迟问题,因此具有很低的延迟.●**RPCA 基本概念** 1.服务器: 是指运行 Ripple 服务器软件并参与共识过程的验证节点,是被其他服务器加入到信任列表中的节点.非验证节点不参与共识过程;2.最新关闭区块(账本): 最近被共识过的区块;3.开放区块: 正在被共识的区块,开放区块通过共识后,就成了最新关闭区块;4.可信任节点列表(UNL): 每个服务器会维护一个可信任节点列表 UNL,这里的信任指相信这个列表中的节点不会联合起来作弊.在共识过程中只接受来自信任节点列表中的节点投票 ●**收集交易,形成交易集** 1.收集交易'候选集': 每个服务器节点从网络中收集客户端发起的新交易以及之前共识过程留下来的旧交易,检查后暂时存放在'交易候选集'中;2.做并集: 每个服务器节点对它 UNL 中服务器的交易候选集做并集,检查所有交易的正确性,并将验证通过的交易打包成提案并广播到网络中;3.投票: 当服务器接从网络中接收到一个新的提案后,若发送提案的服务器在自己的 UNL 中,则对既在该提案又在自身提案中的交易投一票;验证服务器不断处理和发送提案,但只有达到一定投票比例的交易会进入下一轮,在最终轮中,超该服务器 UNL 中 80%节点投票的交易会被放入交易集中;●**区块打包** 1.计算哈希: 形成交易集后,服务器对这些交易验证双花,并打包新的区块,对当前区块号、交易集的 Merkle 根哈希、父区块哈希等内容计算一个哈希 2.广播: 每个验证服务器在网络中传播它的区块哈希值;3.阈值: 验证服务器收到它 UNL 广播过来的区块哈希后,只有当其 UNL 中超过 80%的节点对同一区块哈希时,该区块被认为是有效的,区块被关闭;若某服务器的区块哈希与通过共识的哈希不同,则需要去其他节点拉取正确的区块;若没有区块哈希超过设定阈值,则重新开始共识过程

btb 系统•btb 是基于密码学和经济博弈的一种数字加密货币,也是历史上第一个大规模的虚拟货币系统,可以在世界范围内流通的数字货币.•btb 既是构成数字货币生态系统基础概念和技术的总称,又是 btb 网络中参与者存储和传输的货币单位.**btb 四个创新**•一个去中心化的点对点网络(btb 协议)•一本公开的交易总账(BC)•一套可独立校验交易和发行货币的规则(共识规则)•一种通过全球去中心化对有效的 BC 达成共识的机制(工作量证明算法) **btb 网络**•btb 是依据点对点(P2P)网络架构设计的.P2P 网络上每个节点(计算机)都是对等的,所有节点分担提供网络服务的压力.在 P2P 网络中不存在中心化服务,也不存在层级结构.因此以 P2P 网络构建的 btb 具有去中心化的特点.•尽管 btbP2P 网络中的节点彼此对等,但是取决于支持的功能,它们会扮演不同的角色.一个 btb 节点是一个功能集合: 路由、BC 数据库、挖矿和钱包服务.**btb 网络现状**: 现阶段 btb 主网有近万个运行着 btb 内核客户端的的监听节点,以及数以百计的运行着各类 btbP2P 协议的节点组成.网络中的小部分节点也是挖矿节点,它们竞争挖矿,验证交易,并创建新区块.许多大公司运行全节点的客户端来接入 btb 网络,它们保存了 BC 完整副本,但不具备挖矿和钱包功能,这些节点作为网络边缘路由器,通过它们能构建其他服务.**btb 中继网络**: 由于 btb 矿工之间竞争解决 PoW 问题以获得产生新区块的能力,而这个竞争对时间是非常敏感的,若因为网络传播时延导致竞争失败,会严重影响 btb 生态的良好发展.为了解决这个问题,btb 提出了中继网络,*去最小化矿工之间网络时延*,以解决网络时延带来的问题.和矿工和矿池直通相当于高速主干网,减少跳数和延迟•最初由几个托管在全球的 AmazonWebService 的节点组成,连接了绝大多数的矿工和矿池;之后,中继网络更换了区块传播的协议(FIBRE,快速互联网 btb 传播引擎),进一步降低了数据传播量和网络时延;如今,还有处在提案阶段的基于 Falcon 协议改进的中继网络 **连接的建立**: 当一个新节点启动后,为了参与到网络中来,它需要发现网络中的其他节点.为了连接到一个已知的节点,节点间需要建立 TCP 连接(8333),在建立连接时节点会发送一个包含基本版本信息的'握手消息'version: {nVersion-客户端使用的 btbP2P 版本协议 nLocalServices-节点支持的本地服务列表 nTime-当前时间 addrYou-远程节点 IP addrMe-本地 IP subver: -子版本信息,用于展示本节点运行的软件类型 BestHeight-子节点区块高度} 双方节点通过发送 version 信息,会对应地发送确认信息 verack 以表示节点兼容,当双方互相确认过 version 信息之后,节点间的连接正式建立 **节点的发现**: 一个新节点可以通过'DNS 种子'方式来获取已有的 btb 节点的 IP 地址列表.'DNS'种子是一个提供了 btb 节点 IP 地址列表的 DNS 服务器.DNS 种子分为两类,第一类种子提供稳定的 btb 监听节点 IP 地址的静态列表;一类 DNS 种子提供随机的 btb 节点 IP 地址,这些列表通过爬虫或者运行 btb 客户端来得到.由于 btb 网络是一个动态网络,网络中的节点可能随时加入或者离开,新的节点为了使自己的连接变得稳定,需要在接入到网络之后,不断地发现新节点和被新的节点发现.新节点可以通过 Addr 和 Getaddr 操作来完成这一功能.节点通过 Addr 将本地 IP 发送给连接节点,连接节点将这一 IP 传播到别的节点.节点通过 Getaddr 获得连接节点所保存的节点 IP 列表 **节点状态的检测**: 一个节点可以通过 getpeerinfo 指令来列出全部对等节点的连接信息 若一个连接上没有通信,节点会定期发送一个信息以维持这个连接.若一个节点在一个连接上超过 90 分钟未通信,就认为这个节点从连接断开.**全节点和轻节点**: btb 能具有去中心化的特性得益于它的数据是完整备份在 P2P 网络中的对等节点上.**全节点**: 全 BC 节点维护了一份完整的、最新的包含全部交易的 btbBC 副本.这份副本由节点独立构建和检验,从最初的第一个区块(创世区块)开始一直构建到网络中的最新已知区块.一个全 BC 节点可以独立可信地验证任何交易,而无须求助于或者依赖于其他节点或是信息源.全 BC 节点依赖网络来接收关于新交易区块的更新,在验证之后合并到它本地的 BC 副本中.一个全节点连接到网络之后,需要在本地建立一个完整的 BC 副本: 若这是一个全新的节点,它只包含了内置在客户端软件中的创世区块,它需要在网络中同步所有的区块;若这是重新上线的全节点,它需要检查当前最新块高度,并同步缺失的区块 **轻节点**(SPV 节点): 由于 btb 链上数据的快速增长,作为全节点的存储压力较大,不适用于小型设备.因此,btb 更新出了轻节点(SPV 节点),它通过一种简易支付验证(SPV)的方法使得能够在不存储完整的 BC 数据的情况下去验证交易.SPV 节点仅保存了区块头,不保存块内交易,它保存的 BC 信息比完整 BC 小 1000 倍.由于 SPV 节点无法构建所有的可用 UTXO 全貌,所以它的验证方式不同于全节点,它*依赖于对等节点提供的 BC 数据局部视图进行验证*.**SPV 节点对交易的验证**当一个 SPV 节点需要验证一笔交易时,由于它本地不保存交易,它需要向网络中的对等节点请求数据.SPV 节点会用一个 merkle 路径在交易和包含它的区块之间建立联系,并等待接收到交易所在区块和后续验证区块之后再进行验证.当一个 SPV 节点需要验证一笔交易是否存在时,可以通过请求一个 merkle 路径的证明并验证 BC 上的工作量证明,来确认一笔交易在区块中是否存在.**btb 钱包** btb 钱包中只含有密钥,而不是币.每个用户都有一个包含多个密钥的钱包.钱包只包成含私钥/公钥对的密钥链.用户用私钥签名交易,从而证明他们拥有这笔交易的输出(他们的钱币).钱币则以交易输出的形式存储在 BC 中.**分类***基于密钥技术*,可以将 btb 钱包分类为 1.非确定性钱包 2.确定性钱包 *基于运行平台*,可以将 btb 钱包分类为 1.桌面钱包 2.手机钱包 3.网络钱包 4.硬件钱包 5.纸钱包 **非确定性钱包**,又称随机钱包,钱包只是随机生成私钥的集合.每个密钥都是从随机数独立生成的,密钥彼此无关.缺点是钱包丢失,密钥丢失,则钱包所控制的资金即不可避免的丢失了.需要不断备份**确定性钱包** 其中所有的密钥都是从一个主密钥派生出来,这个主密钥即为种子(seed).该类型钱包中所有密钥都相互关联,若有原始种子,则可以再次生成全部密钥.确定性钱包中使用了许多不同的密钥派生方法.最常用的派生方法是使用树状结构,称为分层确定性钱包或 HD 钱包.•除了简单的测试之外,任何场合都不鼓励使用非确定性钱包,备份和使用过于麻烦.通常推荐使用基于行业标准的 HD 钱包,只需要备份种子助记词.**公私钥和地址生成**: **1.**生成私钥•钱包地址可以通过私钥运算出来.选择私钥的过程就是在一个限定范围内,随机选择一个数字.•这个私钥需要满足以下两点要求.•一,私钥必须是 32 个字节(256 位)•二,私钥必须是正数并且小于(略小于 2^256)•以上限制源于 btb 使用的签名算法为 **ECDSA**,限制了私钥的选取范围.**2.**生成公钥•首先,将私钥传入 ECDSA 算法,得到的结果是一个 64 字节的整数,这个数是由长度均为 32 字节的一个点的 X 值和 Y 值拼接到一起得到的•然后在结果的最开头加上 0x04 获得 btb 的完整公钥•上面完整版公钥比较长,还可以使用压缩版公钥.由于随机椭圆曲线的性质是关于 X 轴对称的.故在给定 X 值的情况下,Y 值就只有正负两种取值.具体的做法可以从完整公钥之中取出 X 值,若 Y 值的最后一个字节是偶数,那么在 X 值的开头添加 0x02.若 Y 值的最后一个字节是奇数,则在 X 值的开头添加 0x03 **3.**生成地址•btb 的地址是通过公钥进行了一系列的转换而获得的,其中主要是进行了多重哈希运算.•第一步,对公钥进行加密.公钥既可以是完整版,也可以是压缩版.X1=RIPEMD-160(SHA-256(PKey)) •第二步,对加密版公钥添加网络标识字节.btb 有两个网络: 主网和测试网.X2=FlagByte+X1(主网对应 flagbyte 为 00)•第三步,添加校验值,钱包软件使用校验值很容易判定地址是否填错或损坏了.校验值=(SHA-256(SHA-256(X2)))的前 4 个字节.把这个校验值添加到第二步结果的末尾即得到钱包地址了.•将上面的十六进制数字串进行'Base58'编码后就能得到我们常见的 btb 钱包地址.•**btb 地址**是一个由数字和字母组成的字符串,以数字'1'开头.•btb 地址通常在 btb 交易中以'收款方'出现 **交易流程**: 钱包软件搜集 UTXO→提供正确的解锁脚本→构造支付给接收者的输出(交易的输出会被创建成一个包含这笔数额的脚本形式,只能被引入这个脚本的一个解答后才能兑换)→构造交易完毕,发送→节点收到交易,独立验证,通过则转发(自循环)→被某个挖矿节点验证并加入到区块中 **发起交易**•Alice 打算花费十美金从 Joe 那里购买 btb, Joe 按发送键进行转账.Joe 的移动 btb 钱包构建了一个交易,从 Joe 的钱包将 0.10BTC 发送给 Alice 提供的地址,并用 Joe 的**私钥**对该交易签名.这就告诉 btb 网络,Joe 已经授权将这笔钱转移给 Alice 的新地址.当交易通过点对点协议传输时,这笔交易在网络中快速传播.在不到一秒钟内,网络中大多数连接良好的节点都会接收到这笔交易,并且首次看到 Alice 的地址.•同时,Alice 的钱包不断地'listen'在 btb 网络上发布的交易,寻找与她的钱包中的地址相匹配的任何内容.在 Joe 的钱包发送交易几秒钟后,Alice 的钱包将显示它正在接收 0.10BTC.•起初,Alice 的钱包把与 Joe 的这笔交易显示为'未确认'.这意味着交易已传播到网络,但尚未记录在 btb 交易账簿即 BC 中.**'确认'就是一个交易必须包含在一个区块中,并被添加到 BC,这样的情况平均每 10 分钟发生一次**.一般为了防止分叉,经验性地会在经过多次'确认'后,才最终认定交易完成.**btb 共识-交易验证**共识是用来实现 BC 在 P2P 网络上所有节点数据一致性的关键机制,现有的共识算法可以分为 BFT 类共识、CFT 类共识和 PoX 类共识.btb 采用的共识算法是 PoW,但是从一个'广义'的角度来说,btb 的共识过程不仅仅是计算 PoW 问题的解.**完整的共识过程**包括: 出块节点对收到的交易的验证;矿工计算 PoW;网络节点对区块的验证 **交易的验证** btb 交易的基本单位是交易输出,而所有可用和可支出的输出就是 UTXO.UTXO 不可分割,只能通过花费来生成新的 UTXO.btb 中的余额概念由钱包程序创建,钱包通过扫描 BC 并汇总其密钥可以控制的所有 UTXO 值,来计算余额.交易的输入大于输出,差值的部分就是**交易费**,交易费不显示在交易中,它会在打包时包含在 Coinbase 中;交易费会补偿 btb 矿工以保护网络安全.收费本身也可以用作安全机制,使攻击者在经济上无法通过向网络注入大量垃圾交易来拥痪网络.交易费与交易大小、币龄等因素有关,交易费不是强制的,但是可以优先级有关,当前免费的交易很难被打包.交易输入通过引用来标识使用哪个 UTXO,并通过解锁脚本提供所有权证明.输入的第一部分是通过引用交易哈希和索引(txid,vout)来指向 UTXO 的指针,该索引标识该交易中的特定 UTXO.第二部分是**解锁脚本**,钱包会构造该解锁脚本,以满足 UTXO 中设置的支出条件.**交易输出包含 UTXO 的值和锁定脚本**.**btb 脚本** btb 通过数字签名来在交易中证明交易是由付款者本人操作的,而这个签名的验证过程,是通过锁定和解锁脚本来实现的.每笔交易在生成输出时,会在每个 UTXO 中构造锁定脚本 scriptPubKey,来指定将来要花费它所需要满足的条件,之所以叫 scriptPubKey,是因为它通常包含一个公钥或公钥哈希,可以用来辨别这个 UTXO 的所有者.每个交易的输入都包含一个解锁脚本 scriptSig,它通常包含由用户私钥产生的签名以及公钥,用于满足锁定脚本的条件.若**解锁脚本中的公钥哈希后与锁定脚本中相同,且私钥产生的签名能被锁定脚本中的公钥解锁**,说明这个签名是这个 UTXO 的所有者签署的,花费是合法的.每个 btb 验证节点通过同时执行锁定和解锁脚本来验证交易.每个输入都包含一个解锁脚本,并引用以前存在的 UTXO.验证软件将复制解锁脚本,检索输入引用的 UTXO,然后从该 UTXO 复制锁定脚本,依次执行解锁和锁定脚本.若解锁脚本满足锁定脚本条件,则输入有效.也就是说,解锁和锁定脚本结合起来,才是一个完整的验证过程,但脚本各自单独执行,并在两次执行之间传递堆栈.btb 的脚本语言是基于栈的语言 **解锁脚本中的签名主要有三个作用**: 身份认证(花的钱是我的),交易认证(这笔交易我是承认的),内容校验(交易内容不能更改).因此解锁脚本中的签名,是对当前交易哈希的签名,由于签名也是交易的内容,因此签名无法签署包含签名的交易哈希.通过制作交易副本,并使用对应输出的锁定脚本替换解锁脚本,并做出其他相应改变,再进行签名的.此外,由于不同的交易可能需要签名的部分不同,如一笔交易,付款者只希望其中一笔/部分钱必须支付给某人,剩余的钱支付给谁并不在意,那他只需要对全部输入和某一笔输出进行签名即可.因此每个签名都有一个 SIGHASH 字段,来指示签名的哈希包含了交易中的哪一部分.钱包软件生成交易以后,生成的交易会被发送给 btb 网络中的相邻节点,以便可以在整个 btb 网络中传播.但是,在将交易转发给其邻居之前,每个接受交易的节点都会首先独立验证交易,这样可以确保**仅有有效交易在网络上传播**,而遇到无效交易的第一个节点会将其丢弃.交易在经过节点独立验证后但尚未确认前,会被放入节点的交易池,节点不断监听 BC 网络,在新的区块产生之后,节点会将新块中包含的交易从交易池中剔除.**btb 共识-PoW** 当交易池中的交易达到一定数量后,挖矿节点会将它们打包到一个候选块中,挖矿节点此时会创造一笔特殊的交易 Coinbase,作为区块中的第一笔交易.Coinbase 是对矿工的奖励,该奖励包含区块奖励和该区块中所有交易的交易费用之和(通过计算所有交易的输入减输出得到).Coinbase 交易不消耗 UTXO,取而代之它有一个称为 coinbase 的输入,该交易有一个输出,支付给矿工自己的 btb 地址.交易集合生成后,挖矿节点需要构造区块,填充区块头.btb 的区块平均每 10min 生成一个,这是货币**发行的速率和交易结算速度的基础**,需要在长期内保持恒定;因此难度目标值会根据全网的挖矿能力进行调整,保证 10min 产生一个新区块的周期**区块的验证**挖矿成功时,挖矿节点将块发给所有对等节点,这些节点对新块进行独立验证并传播,一旦一个节点验证了一个新的区块,它会尝试将新的区块连接到现存的 BC;节点该步的验证过程,主要是为了保证矿工不会作弊,比如矿工在 Coinbase 中给自己奖励了 1000 个 btb,则这笔交易在验证时会被认为无效,从而导致整个块无效.这一步验证的内容包括: •区块的数据结构语法有效•区块头散列值小于目标难度(工作量证明要求)•区块时间戳早于验证时刻未来两个小时(允许时间误差)•区块大小在限制范围之内•第一个交易(且只有第一个)是 coinbase 交易•所有交易再次经过之前所示的独立验证有效.由于链存在分叉的可能性,因此一笔交易在上链以后,可能由于最长链的改变成为无效交易,为了防止交易被推翻而造成的损失,交易生效需要一定的**确认数**;一笔交易,当其上链时就会获得一个确认数,之后每产生一个新区块,交易的确认数加一.这一个确定的确认数,在 btb 网络中公认为 6,因为当产生 6 个新区块后,区块被推翻的几率已经可以小到忽略不计.6 个区块只是一个参考数字 **btb 激励**•在矿工将交易打包进区块时,其中的第一笔交易是一笔叫做 coinbase 的特殊交易.与其他常规交易不同,这笔交易不需要 UTXO 作为输入,其输入中只包含一个叫 coinbase 的输入;这笔交易只有一个输出,输出支付到矿工自己的 btb 地址上.•Coinbase 交易中输出的 btb 的量就是矿工打包完成这个区块所拿到的 btb 奖励 **输出的值由以下两部分组成**: •**矿工费/交易手续费**•**区块奖励**•二者相加的结果就是 coinbase 的输出值,也就是矿工拿到的奖励.•在打包区块时,矿工会将块中除了 coinbase 的其他交易的输入输出分别求和再计算差值,这个差值就是矿工打包这个区块收到的手续费的总和.•区块奖励值是一个基于区块高度计算出来的值.btb 特性-分叉为 BTC 和 BCH 两种币 **51 攻击**谁能抢到最长链,谁才拥有记账权.在 btb 网络中,采用 PoW 共识机制来解决如何获得记账权的问题,采用'最长链共识'解决如何记账的问题.所谓 51%的攻击,就是利用 btb 网络采用 PoW 竞争记账权和'最长链共识'的特点,使用算力优势生成一条更长的链'回滚'已经发生的'交易行为'.51%是指算力占全网算力的 51%.btb 网络需要通过哈希碰撞来匹配随机数从而获得记账权,算力衡量的是一台计算机每秒钟能进行哈希碰撞的次数.算力越高,意味着每秒钟能进行越多次的哈希碰撞,即获得记账权的几率越高,在理论上,若掌握了 50%以上的算力,就拥有了获得记账权的绝对优势,可以更快地生成区块,也拥有了篡改 BC 数据的权利.实际上,当恶意攻击者持有 btb 全网占比比较高的算力时,即使尚未达到 51%的比例,也可以制造相应的攻击,比较典型的就是**双花问题**.假设 A 拥有 51%的算力,在区块高度 1127 时,A 转给 B 一个 btb 的记录被矿工打包.待交易确认后 A 依靠 51%的算力优势在区块高度 1126 后重新生成了一条'更长的链',并在区块高度 1127 处又将该 BTC 转给 C 且该交易记录被打包,即该链包含了 A 将一个 btb 转给 C 的记录.根据'最长链共识',包含 C 转账记录的链成为主链,则 A 转给 B 的一个 btb 则为'无效支付'.若掌握了 51%的算力,除了可以修改自己的交易记录外,还可以阻止区块确认部分交易,以及阻止部分矿工获得有效的记账权.但是,拥有 51%的算力也不是万能的,无法修改其他人的交易记录,也不能阻止交易的发出,更不能凭空产生 BTC.**面临的挑战****算力中心化**•随着全网算力的不断上涨、难度的不断升级,单台机器挖到块的概率变得非常的低.于是独立矿工们不得不加入矿池,按算力比例分配收益,以此来保障挖矿的稳定性.•然而随着越来越多的矿工加入到各个矿池中,btb 系统的算力变得越来越中心化.**低吞吐**按照 btb 白皮书的定义,我们很容易计算出 btb 的交易处理时间: •若按照每十分钟产生一个 1MB 大小的区块,每笔交易需要 250B 来存储数据,计算可以知道 1MB 只能存放 4194 个交易数据(Transaction),再除以时间,也就是一秒钟最多处理 7 笔交易,显然这个速度是无法满足正常的交易需求的**资源浪费** **监管风险** btb 具有匿名性、难追溯、跨境流通便利的特点,使其被越来越多地违法分子用来进行诈骗、赌博、洗钱,成为犯罪的新媒介 **BitcoinScript 简介**•btb 在交易中使用脚本(Script)系统,类似 FORTH 语言,脚本是简单的、基于堆栈的、并且从左向右处理,它特意设计成非图灵完备,没有 LOOP 语句.一个脚本本质上是众多指令的列表,这些指令记录在每个交易中,交易的接收者想花掉发送给他的 btb,这些指令就是描述接收者是如何获得这些 btb 的.•逆波兰表达式语言.•无状态.执行脚本之前没有状态,执行脚本之后也不保存状态.执行前后堆栈都是空的.•绝大多数 btb 交易都是基于 P2PKH(pay-to-Public-Key-Hash)脚本的.

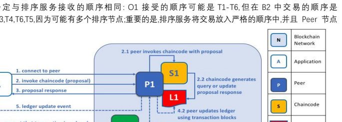
ytf 与智能合约 **智能合约•狭义定义**: 运行在分布式账本上预置规则、具有状态、条件响应的,可封装、验证、执行分布式节点复杂行为,完成信息交换、价值转移和资产管理的计算机程序.•**广义定义**: 无需中介、自我验证、自动执行合约条款的计算机交易协议.按照其设计目的可分为:旨在作为法律的替代和补充的智能法律合约,旨在作为功能型软件的智能软件合约,以及旨在引入新型合约关系的智能替代合约**特性**: 自动执行、安全透明、自治自足 **ytf 概述****产生背景** 1.btb 并不完美,其中协议的扩展性是一项不足,例如 btb 网络里只有一种符号(btb),用户无法自定义另外的符号,这些符号可以是代表公司的股票,②或者是债务凭证等,这就限制了一些功能 2.btb 协议尽管使用了一套基于堆栈的脚本语言,然而却不足以构建更高级的应用,例如去中心化交易所等.ytf 从设计上就是为了解决 btb 扩展性不足的问题 **定义** ytf(Ethereum)是一个开源的有智能合约功能的公共 BC 平台,通过其专用加密货币以太币(Ether)提供去中心化的以太虚拟机(EVM)来处理点对点合约.(以太币①支付算里,价值因素②任何操作和转账所需要支付的代价)**两个特点**: 具有一个全球范围可访问的单体状态;还有一个执行状态更改的虚拟机.从更加实际的角度来看,ytf 是一个开源的、全球去中心化的计算基础架构,可以执行称为智能合约的程序.它使用 BC 同步和保存系统状态,借助**以太币**这种数字货币来计量并控制程序执行的资源开销.**ytf 基本原理** **ytf 模型**•ytf 的本质就是一个基于交易的状态机: 起始于一个创世(Genesis)状态,然后随着交易的执行,状态逐步改变一直到最终状态,这个最终状态就是 ytf 世界的权威版本.**基本组件**•**点对点(P2P)网络**: ytf 运行在 EthereumMainNetwork 上,这是一个通过 TCP30303 端口寻址的网络 •**共识规则**: ytf 的共识规则,由 ytf 黄皮书中的参考标准进行精确定义•**交易**: ytf 交易是一个网络消息,主要包含交易的发送方、接收方、价值和数据载荷.•**状态机**: ytf 的状态转换由 ytf 虚拟机(EVM)处理,这是一个基于栈的虚拟机,执行 bytecode(字节码指令).被称为'智能合约'的 EVM 程序采用高级语言编写,并编译为通过 EVM 执行的字节码.•**数据结构**: ytf 的 BC 以数据库的方式保存在每一个节点之上,BC 内包含了交易和系统的状态,经过哈希处理的状态保存在 MerklePatriciaTree(MPT)数据结构之内.•**共识算法**: ytf 使用 btb 的共识模型 Nakamoto Consensus,它使用顺序单一签名区块,由 PoW 加权重要性来确定最长链,从而确定当前状态.但是,有计划在不久的将来转向代号为 Casper 的 PoS 加权投票系统.**ytf 账户**•ytf 引入了账户的概念以取代 btb 的 UTXO 模型.账户以地址为索引,地址由公钥衍生而来.有**两种类型的账户**: 外部账户(通常简称为'账户')和合约账户.•Nonce: 若账户是一个外部账户,代表从此账户地址发送的交易序号.若账户是一个合约账户,代表此账户创建的合约序号.18•Balance: 余额,即账户拥有的以太币数量,单位为 Wei,1Ether=10eWei.•CodeHash: 代码哈希,与账户关联的 EVM 代码的哈希值,外部账户的 codeHash 为一个空字符串的哈希,创建后不可更改.状态数据库中包含所有代码片段哈希,以便后续使用.•StorageRoot: 存储根节点,账户内容的 MerklePatricia 树根节点的哈希编码.**外部帐户 EOA** 是由用户实际控制的账户,拥有一对公私钥,可以通过使用私钥创建和签署交易,将消息发送到其他外部帐户或合约帐户.两个外部账户之间的消息只是一个价值转移.但是从外部账户到合约账户的消息会激活合约账户的代码,允许它执行各种操作(例如转移 Token,写入内部存储,创建新的 Token,执行一些计算,创建新的合约等).•**合约账户**是一个包含合约代码的账户,由合约代码控制.合约账户不能自行发起新的交易,合约账户只能触发交易以响应其他来自外部帐户或合约账户的交易.**外部账户可对合约或外部账户发起交易,合约账户间可能相互触发** **两种账户对比****账户私钥** 1.一般在 ytf 的钱包客户端创建通过私钥算法创建生成,私钥由账户所有者自己保管存储 2. 没有对应的账户私钥 **账户地址** 1.账户地址基于账户的公钥采用地址生成算法推导得出 2. 账户地址采用智能合约发布者的账户相关信息推导得出 **账户链上生成** 1.通过账户私钥签名一笔交易发往 ytf 的节点创建生成 2. 通过外部账户向 ytf 发布智能合约时创建生成 **账户存储 1.**仅仅存储一个账户对应的交易序号和账户的有效余额 2. 除了外部账户的存储信息外,还存储该账户对应的智能合约数据.这些智能合约的数据按照特定的编码方式进行组织和存储**账户代码** 1.无对应的账户代码 2.保存了智能合约的账户代码内容,该内容即为编写的智能合约代码经过编译器编译后字节码内容 **UTXO 与账户模型的对比** **UTXO 模型的好处**: •*可扩展性*-由于可以同时处理多个 UTXO,因此可以实现并行交易并鼓励可扩展性创新.•*隐私性*-即使 btb 不是一个完全匿名的系统,假设用户每笔交易都使用新地址,那么也能够提供更高级别的隐私.若需要更好的隐私性可以考虑更复杂的方案,例如环形签名.**账户/余额模型的好处**: •*简单性*-ytf 选择了更直观的模型,以使开发人员在开发那些需要状态信息或涉及多方的智能合约时更简单.例如智能合约,它能够直接跟踪状态并根据状态执行不同的任务.UTXO 的无状态模型强制交易包含状态信息,这使得合约的设计复杂化.•*效率*-除简单性外,帐户/余额模型更有效,因为交易只需要验证发送帐户是否有足够的余额.•账户/余额模型的一个**缺点**是无法避免**双重支付攻击**,但通过递增的随机数可以消除这种类型的攻击.在 yt 中,每个帐户都有一个公共可见的随机数,每次进行交易时,随机数增加一.这可以防止同一个交易被多次提交.**ytfBC**ytf 对 btbBC 技术上做了一些调整,ytf 区块主要由区块头、交易列表和叔区块头三部分组成.•**区块头**包含下列信息: 父块的散列值(PrevHash)、叔区块的散列值(UnclesHash)、状态树根散列值(stateRoot)、交易树根散列值(TransactionRoot)、收据树根散列值(ReceiptRoot)、时间戳(Timestamp)、随机数(Nonce)等.ytfBC 上区块数据结构的一个重大改变就是保存了**三棵 Merkle 树根**,分别是状态树、交易树和收据树.存储三棵树可方便账户做更多查询.•**交易列表**是由矿工从交易池中选择打包进区块中的一系列交易.•BC 上的第一个区块称为'创世区块',BC 上除了创世区块以外每个区块都有它的父区块,这些区块连接起来组成一个 BC,出块时间约为 **15 秒**.**ytf 区块结构** **Header 成员**•ParentHash: 指向父区块(parentBlock)的哈希指针.除了创世块(GenesisBlock)外,每个区块有且只有一个父区块.•Coinbase: 挖掘出这个区块的矿工地址.在每次执行交易时系统会给与一定补偿的 Ether,这笔金额就是发给这个地址的.•UncleHash: Block 结构体的成员 uncles 的 RLP 哈希值.uncles 是一个 Header 数组.•Root: StateDB 中的'stateTrie'的根节点的 RLP 哈希值.Block 中,每个账户以 stateObject 对象表示,账户以 Address 为唯一标示,其信息在相关交易(Transaction)的执行中被修改.所有账户对象可以逐个插入一个 Merkle-Patrica-Trie(MPT)结构里,形成'stateTrie'.•TxHash:Block 中'txTrie'的根节点的 RLP 哈希值.Block 的成员变量 transactions 中所有的 tx 对象,被逐个插入一个 MPT 结构,形成'txTrie'.•ReceiptHash: Block 中的"ReceiptTrie"的根节点的 RLP 哈希值.Block 的所有 Transaction 执行完后会生成一个 Receipt 数组,这个数组中的所有 Receipt 被逐个插入一个 MPT 结构中,形成"ReceiptTrie".•Bloom: **Bloom 过滤器(Filter)**,用来快速判断一个参数 Log 对象是否存在于一组已知的 Log 集合中.•Difficulty: 区块的难度.Block 的 Difficulty 由共识算法基于 parentBlock 的 Time 和 Difficulty 计算得出,它会应用在区块的'挖掘'阶段.•Number: 区块的序号.Block 的 Number 等于其父区块 Number+1.•Time: 区块'应该'被创建的时间.由共识算法确定,一般来说,要么等于 parentBlock.Time+10s,要么等于当前系统时间.•GasLimit: 区块内所有 Gas 消耗的上限,打包进该区块的交易的 GasLimit 之和不能大于区块的 GasLimit.该数值在区块创建时设置,与父区块有关.具体来说,根据父区块的 GasUsed 同 GasLimit*2/3 的大小关系来计算得出,用以控制区块可以打包的交易数•GasUsed: 区块内所有 Transaction 执行时所实际消耗的 Gas 总和.•Nonce: 一个 64bit 的哈希数,它被应用在区块的"挖掘"阶段,并且在使用中会被修改.**ytf 交易与消息**•ytf 的'交易'是指一条外部账户发送到 BC 上另一账户的消息的签名数据包,包含了发送者的签名、接收者的地址以及发送者转移给接收者的以太币数量等内容.•ytf 交易数据包还包括 GasLimit 和 GASPRICE.ytf 的每一笔者交易都需要支付一定的费用,用于支付交易执行所需要的计算开销.计算开销的费用并不是以太币直接计算的,而是引入 Gas 作为执行开销的基本单位,通过 GasPrice 与以太币进行换算.•引入 Gas,还可以防止代码的指数型爆炸和无限循环,每笔交易需要对执行代码所引发的计算步骤-包括初始消息和所有执行中引发的消息-做出限制.GasLimit 就是限制,GASPRICE 是每一计算步骤需要支付矿工的费用.若执行交易的过程中,'用完了瓦斯',所有的状态改变恢复原状态,但是已经支付的交易费用不可收回了.若执行交易中止时还剩余瓦斯,那么这些瓦斯将退还给发送者•**创建合约**有单独的交易类型和相应的消息类型;**合约的地址**是基于账号随机数和交易数据的哈希计算出来的.•ytf 的消息在某种程度上类似于 btb 的交易,一个消息就是一个交易•消息包括: 消息发送者、消息的接收者、可选的数据域、合约实际输入数据、gasLimit,同交易.**交易的类型**•交易可以分为三种: •**转账交易**: 是最简单的一种交易,从一个账户向另一个账户发送以太币.发送转账交易只需要指定交易的发送者、接收者、转币的数量.•**创建合约的交易**: 将合约部署到 BC 上的交易,发送者是合约的创建者,接受者为空,交易数据字段中指定合约的二进制代码.•**执行合约的交易**: 调用合约中的方法,需要将交易的接受者指定为要调用的合约的地址,通过交易数据字段指定要调用的方法以及向该方法传递的参数.一段智能合约是被唯一的(合约)地址所标识,该地址有自己的资金余额(以太币),并且一旦有一笔交易发送至该(合约)地址,ytf 网络节点就会执行合约逻辑.**ytf 使用 ytf 虚拟机(EVM)来执行智能合约**.**ytf 状态转换函数(交易的执行)**ytf 的状态转换函数: APPLY(S,TX)->S',定义如下•检查交易的格式是否正确、签名是否有效和随机数是否与发送者账户的随机数匹配.如否,返回错误.•计算交易费用: fee=GASLimit•GASPRICE,并从签名中确定发送者的地址.从发送者的账户中减去交易费用和增加发送者的随机数.若账户余额不足,返回错误.•设定初值 GAS=GASLimit,并根据交易中的字节数减去一定量的瓦斯值.•从发送者的账户转移价值到接收者账户.若接收账户还不存在,创建此账户.若接收账户是一个合约,运行合约的代码,直到代码运行结束或者瓦斯用完.•若因为发送者账户没有足够的钱或者代码执行耗尽瓦斯导致价值转移失败,恢复原来的状态,但是还需要支付交易费用,交易费用加至矿工账户.•否则,将所有剩余的瓦斯归还给发送者,消耗掉的瓦斯作为交易费用发送给矿工.**转账交易或合约调用交易 1.发起者发起转账交易或合约调用交易请求** From: 发送者地址用户 A To: 接受账户地址或合约地址 Value: 转移的金额 Gas: 用来完成交易(执行合约)的 Gas 数量 GasPrice: 交易中心愿意付出的 Gas 的单价 Data: 一段要发送给接受者(合约账户)的消息.经过 client 交易形式化后,增加一个 Nonce: 整数,用户发起的每笔交易都有唯一的编号,表示该交易是用户发起的第几笔交易 **2. 对等节点(可能有多个)检验、存储和转发交易**: 检查交易,计算交易费用,从发送方的账户中减去相应费用,并根据交易中的字节数减去一定量的 Gas 值.将交易请求放在存储池中,并转发给其他节点 **3. 获得记账权的节点打包交易请求并执行合约代码(使用自己的 EVM)** 用户 B 获得新区块记账权,用户 A 的交易区块请求和其他交易一起被打包到区块 **4.生成区块的节点将区块发送到网络中 5. 交易被所有节点保存在本地的 BC** 接收到区块的节点运行智能合约,并对结果进行相互验证.该用户同步到用户 B 挖掘的区块,验证区块的合法性,若合法,将其加到本地的 BC **创建合约交易** 1.发送者发起创建智能合约的交易请求 2.获得记账权的节点打包交易并部署合约 3.获得记账权的节点发送新区块至 ytf4.智能合约被部署在本地 BC 上,接上面的 5 **状态树、交易树和收据树**•每个 ytf 区块头保存了三棵树,这三棵树组成 MPT 树: 状态树、交易树、收据树.ytf 使用了 MerklePatricia 树(又称 MerklePatriciaTrie,MPT)作为其数据组织形式,用来组织管理账户数据、生成交易集合哈希等.MPT 是一种加密认证的数据结构,它融合了 Merkle 树和 Trie 树(前缀树)两种数据类型的优点.**Trie 树**,也叫做 Radix 树,通常用来存储(key,value)对.key 代表的是从树根到对应 value 的一条真实路径.即从根节点开始,key 中的每个字符(从前到后)都代表着从根节点出发寻找相应 value 所要经过的子节点.value 存储在叶节点中,是每条路径的终节点.•假如 key 中的每个字符最多来自一个容量为 N 且所包含的字母互不相同的字母表,那么树中的每个节点最多会有 N 个孩子,树的最大深度便是 key 的最大长度.**Patricia 树**是一种压缩的 Trie 树.•非根节点不是只能存储字符,而是可以存储字符串,也就是路径压缩了的 Trie,因此节省了在内存空间的开销.**MPT 树**结合了 Patricia 树和 Merkle 树的优点,在 Patricia 树中根节点是空的,而 MPT 树可以在根节点保存整棵树的哈希校验和,而校验和的生成则是采用了和一Merkle 树一致的生成方式.ytf 采用 MPT 树来保存交易、交易的收据以及世界状态.ytf 还使用了一种特殊的十六进制前缀(hex-prefix,HP)编码,使得字母表只有 16 个字符(一个字符称为一个 nibble).MPT 树节点分成了四种类型: •空节点(hashNode)•叶节点(valueNode)•分支节点(fullNode)•扩展节点(shortNode)**状态树**中的每个节点最多有 16 个孩子节点,每个叶节点表示一个账户,这些叶节点的父节点由叶节点的散列组成,而这些父节点再组成更高一层的父节点,直至到形成根节点.•状态树包含一个键值映射,其中**键是账户地址,值是账户内容**,主要是{nonce,balance,codeHash,storageRoot}.•状态树的根节点存放在区块头中.**交易树**•每个区块都有一棵独立的交易树.区块中交易的顺序主要由'矿工'决定,在这个块被挖出前这些数据都是未知的.不过'矿工'一般会根据交易的 GasPrice 和 nonce 对交易进行排序.首先会将交易列表中的交易划分到各个发送账户,每个账户的交易根据这些交易的 nonce 来排序.每个账户的交易排序完成后,再通过比较每个账户的第一条交易,选出最高价格的交易,这些是通过一个堆(heap)来实现的.每挖出一个新块,生成一个交易树.•交易树包含的键值对中,其中**每个键是交易的编号,值是交易内容**.**收据树**•每个区块都有自己的收据树,收据树不需要更新,收据树代表每笔交易相应的收据.收据树也包含一个键值映射,其中**键是索引编号**,用来指引这条收据相关交易的位置,**值是收据的内容**.•交易的收据是一个 RLP 编码的数据结构: [medstate,Gas_used,logbloom,logs].其中,medstate 是交易处理后树根的状态;Gas_used 是交易处理后 Gas 的使用量;logs 是表格[address,[topic1,topic2,…] ,data]元素的列表,表格由交易执行期间调用的操作码 LOG0…LOG4 生成(包含主调用和子调用),address 是生成日志的合约地址,topicn 是最多 4 个 32 字节的值,data 是任意字节大小的数组;logbloom 是交易中所有 logs 的 address 和 topic 组成的布隆过滤器(二进制向量数据结构,具有很好的空间和时间效率,被用来检测一个元素是不是集合中的一个成员).区块头中也存在一个布隆过滤器,使用布隆过滤器可以减少查询的工作量,这样的构造使得 ytf 协议对轻客户端尤其的友好.利用存储的三棵 MPT 树,ytf 客户端可以轻松地**查询**以下内容: •某笔交易是否被包含在特定的区块中.•查询某个地址在过去的 30 天中发出某种类型事件的所有实例(例如,一个众筹合约完成了它的目标).•目前某个账户的余额.•一个账户是否存在.•假如在某个合约中进行一笔交易,交易的输出是什么.**数据存储 LevelDB**•高效的键值对数据库,其中键值都是二进制的 •ytf 中共有三个 LevelDB 数据库,分别是 BlockDB、StateDB 和 ExtrasDB.•BlockDB 保存了块的主体内容,包括块头和交易;•StateDB 保存了账户的状态数据;•ExtrasDB 保存了收据信息和其他辅助信息.•可把三个 MPT 树理解为 LevelDB 所存储数据的索引,便于 ytf 系统查询和处理这些数据.**ytf 的共识机制 PoW+PoS**•ytf 共识机制有**四个阶段**,即 Frontier(前沿)、Homestead(家园)、Metropolis(大都会)、Serenity(宁静).ytf 前三个阶段采用的是 PoW 共识机.第四个阶段将采用自己创建的 PoS 机制,名为 Casper 投注共识,这种机制增加了惩罚机制,并基于 PoS 的思想在记账节点中选取验证人.**PoW-Ethash 算法**•Ethash 是 ytf 的 PoW 算法•用 Ethash 算法来代替原有的 PoW 算法,是为了解决挖矿中心化问题.Ethash 算法的特点是挖矿的效率基本与 CPU 无关,而与内存大小、带宽正相关,目的是去除专用硬件的优势,抵抗 ASIC.这*算法的基本流程*: ①对于每一个区块,首先通过扫描区块头的方式计算出一个种子,该种子只和当前区块的信息有关,然后根据种子生成一个 16M 的伪随机数据集(Cache).②基于 Cache 生成一个 1GB 大小的数据集合 DAG(有向无环图),它是一个完整的搜索空间.③挖矿的过程就是从 DAG 中随机选择元素(类似于 btb 挖矿中查找合适 Nonce)再进行哈希运算,可以从 Cache 快速计算 DAG 指定位置的元素,进而哈希验证.挖矿者存储数据集,该数据集周期性更新,每 3000 个块更新一次,并且规定 DAG 的大小随着时间推移线性增长,从 1G 开始,每年大约增长 7G 左右.④验证者能够基于缓存计算得到 DAG 中自己需要的指定位置的元素,然后验证这些指定元素的散列是不是小于某个散列值,也就是验证'矿工'的工作是否符合要求 **PoS-Casper**•ytf 的 PoS 共识协议称为 Casper,采用'权益'(Stake,即以太币)作为记账权背书.•大致的思路是,将 Casper 的应用逻辑通过智能合约来实现,在这个合约中,记账权归属于'验证者'.任何拥有以太币的账户都可以在 Casper 合约中成为验证者,前提是必须要在 Casper 智能合约中抵押一定的以太币(抵押的以太币越多,被选中作为验证者的概率就越高).之后 Casper 合约通过一种随机方式,选出一个验证者集合.被选中的验证者集合按照一定顺序依次验证区块(当然也可以选择放弃),若区块没有问题,就将其添加到 BC 中,同时相应的验证者将会获得一笔与他们的抵押成比例的奖励.若验证者不遵守合约规定的规则,合约就会没收他抵押的以太币作为惩罚.**ytfGhost 协议和叔块(UncleBlock)**•**GHOST** 是一种主链选择协议(不是侧链选择协议).GHOST 协议以**包含子树数目最多**为基本原则.•GHOST 协议,认为孤块也是有价值的,会给发现孤块的矿工以回报,也会给引用孤块的矿工以奖励.在 yt 中,**孤块被称为'叔块'**,它们可以为主链的安全作出贡献.•叔块上的交易不会得到执行.•叔块的好处: 1.ytf 十几秒的出块间隔,大大增加了孤块的产生,并且降低了安全性.通过鼓励引用叔块,使引用主链获得更多的安全保证(因为孤块本身也是合法的)2.btb 中,采矿中心化成为一个问题.给与叔块报酬,可以一定程度上缓解这个问题.•区块可以引用 0-2 个叔块.叔块必须是区块的前 2 层~前 7 层的祖先的直接的子块,被引用过的叔块不能重复引用.•引用叔块的区块,可以获得挖矿报酬的 1/32,也就是 5*1/32=0.15625Ether.最多获得 2*0.15625=0.3125Ether.•被引用的叔块,其矿工的报酬和叔块与引用它的区块之间的间隔层数有关系.**以太币(ETH)**是 ytf 发行的一种数字货币,在公有链上发起任何一笔交易都需要支付一定的以太币.•以太币的总供给及其年发行率由 2014 年的预售决定的,以太币来源包括'矿前+区块奖励+叔块奖励+叔块引用奖励'.具体的分配大致如下: •预付款的贡献者总共有 6000 万个以太币.•挖出一个新区块的矿工奖励为 5 个以太币.•叔块被引用,每个叔块会为矿工产出 4.375 个以太币,矿工每引用一个叔块,可以得到大约 0.15 个以太币(最多引用两个叔区块).**以太币的货币单位 小结**•ytf 通过建立终极的抽象的基础层-内置有图灵完备编程语言的 BC-使得任何人都能够创建合约和去中心化应用,并在其中设立他们自由定义的所有权规则、交易方式和状态转换函数.•**ytf 虚拟机(EVM)**是 ytf 中智能合约的运行环境.它不仅被沙箱封装起来,事实上它被完全隔离,运行在 EVM 内部的代码不能接触到网络、文件系统或者其它进程.甚至智能合约与其它智能合约只有有限的接触.**智能合约**定义•从用户角度看,'智能合约'是根据事先任意制订的规则来自动转移数字资产的系统.•从开发者角度看,'智能合约'就是存储在 BC 上的代码,用以实现执行特定的功能.**使用合约的步骤**编译→创建→部署→调用→监听或销毁 •智能合约是一种无需中介、自我验证、自动执行合约条款的计算机交易协议.•智能合约表示的是一种'无需第三方,通过某种可信环境来执行的,以数字形式存在并且被履行的合约内容'.智能合约就像一种合同,当合同的内容需要被履行的时候,不需要人们按照合同内容进行操作,而是合同自己能够执行需要的操作.•智能合约的行为由合约代码控制,而智能合约的账户则保存了合约的状态.**BC 上的智能合约**•BC 上的智能合约就是存储在 BC 上的代码,用以实现执行特定的功能.•与传统合约相比,BC 智能合约有如下**优势**: 1.逻辑的明确性,不容易产生歧义.2.不可能被篡改.合约执行记录,可以作为永久凭证.3.合约执行的强制力可以保证: ①合约本身在 BC 上,②合约结果也~ **智能合约运行机制**: 矿工可将智能合约和其结果打包到区块中•智能合约一般具有值和状态两个属性,代码中预置了合约条款的相应触发场景和响应规则.•智能合约经多方共同协定、各自签署后随用户发起的交易提交,经 P2P 网络传播、矿工验证后存储在 BC 特定区块中,用户得到返回的合约地址及合约接口等信息后即可通过外部数据发起交易来调用合约.•矿工受系统预设的激励机制激励,将贡献自身算力来验证交易,矿工收到合约创建或调用交易后在本地沙箱执行环境 EVM 中创建合约或执行合约代码.•合约代码根据可信外部数据源(也称为预言机,Oracle)和世界状态的检查信息,自动判断当前所处场景是否满足合约触发条件,以严格执行响应规则并更新世界状态.•交易验证有效后被打包进新的数据区块,新区块经共识算法认证后链接到 BC 主链,所有更新生效.**智能合约的基础架构模型**•*运维层*: 封装了一系列对合约层中静态合约数据的动态操作,包括机制设计、形式化验证、安全性检查、维护更新、自毁等,是保证智能合约能够按照设计者意愿正确、安全、高效运行的关键.•*合约层*: 封装静态的合约数据,可看作是智能合约的静态数据库,封装所有智能合约调用、执行、通信规则.包括各方达成一致的合约条款、合约条款代码化后的情景-应对型规则(if-then,what-if),以及按照平台特性和立契者意愿补充必要的智能合约与用户之间、合约与合约之间的访问权限与通信方式等.•*基础设施层*: 封装了支持智能合约及其衍生应用实现的所有基础设施,包括分布式账本及其关键技术、开发环境和可信数据源等,这些基础设施的选择在一定程度上影响智能合约的设计模式和合约属性.**智能合约结构**合约就像一个类(class),其中包含状态变量、函数、函数修改器、事件、结构和枚举.合约还支持*继承*.**创建和调用智能合约的流程** 1.编写智能合约,编译为字节码,打包后部署到 ytfBC 网络上 2.调用合约,发起交易 **ytf 上的智能合约相关流程**•编译合约: •ytf 上的智能合约首先要进行编写,再用编译器编译成 ytf 虚拟机 EVM 的字节码形式.•部署合约: 需要一个外部账户通过一个部署合约的交易来将合约的代码部署到链上.合约在部署后,会以一个**合约账户**的形式存在,拥有自己的合约地址.需要支付费用和用户输入密码•调用合约: 用户需要构造一笔交易/消息.该消息的接受者是目标合约的合约地址,并在 Data 部分需要加上对应的数据.•监听: 在编写合约的时候,可以在合约中定义各种 event.•在合约的函数中,可以通过 emit<eventname>(args..)的方式来触发事件.•通过 web3.js 中的函数可以指定合约地址以及其 event 进行监听.•销毁•通过调用 selfdestruct 函数,合约就被销毁.只有拥有者才能够销毁合约 **合约使用场景**金融类场景: 对冲合约、储蓄钱包、遗嘱等 非金融类场景: 在线投票、去中心化治理、域名注册 **智能合约-安全性**•所有智能合约都被公开的执行,源码很容易获得,代码中隐藏的 Bug 和漏洞也更容易被恶意攻击者发现.•TheDAO 事件 **ytf 虚拟机**是 ytf 协议和具体操作的核心.•EVM 是一个基于栈的架构,是 ytf 中的堆栈指令解释执行器,提一供图灵完备的执行环境.其基本数据处理单元是 256 位的'字'.•由于智能合约的执行结果需要在 ytf 的网络节点之间进行共识,因此 EVM 的所有指令执行必须具有*确定性的输出*.•*单线程*.•EVM 为了保证交易的可终止性,引入了执行器燃料 Gas,按照执行指令的复杂度、所需要的存储空间对每个步骤进行计价,消耗指定数量的 Gas,当交易所赋予的 Gas 消耗为零时则停止指令的执行,退出执行环境.**执行费用** 1.为了能够激励节点有足够的动力来维护 ytf 网络的稳定运行,对提供节点算力的用户需要支付一定的费用,这种奖励来源于两部分: 一是单纯的区块打包奖励;一是打包交易的手续费.2.交易手续费在 ytf 中通过交易执行的复杂度来定价,这种定价取决于执行交易需要消耗多少算力.ytf 的交易手续费也从一定程度上保证了智能合约不会出现执行后无法终止的情况,通过经济的手段防止了恶意的交易在 BC 网络上传播从而引起网络拥痪.3.ytf 的 VM 除了执行指令消耗 Gas 外,还对存储空间的使用也按 Gas 进行计价.消耗的存储空间越大,那么需要支付的 Gas 费用也会越高.这个存储空间包括内存空间和持久空间,持久空间是指智能合约的 Storage 变量.**堆栈执行器**•ytf 的 EVM 虚拟机是一种堆栈执行器解释执行的执行器,通过从 BC 的数据库中加载智能合约的二进制代码,从头开始执行指令,结合交易请求的相关参数和相关的跳转指令来实现智能合约执行流程.**Solidity** 是一种用于编写 ytf 智能合约的面向对象高级语言 1.Solidity 中的合约与面向对象编程语言中的类很相似,在一个合约中可以声明多种成员,包括状态变量、函数、函数修改器、事件等.同时,一个合约可以继承另一个合约 2.状态变量是永久存储在合约账户存储中的值,用于保存合约的状态.Solidity 语言提供了多种类型的变量.3.函数是合约的执行单位,一个合约可能包含多种功能函数,它们相互调用,共同组成了合约的工作逻辑.4.函数修改器可用于改变函数的行为,比如在函数执行前加入输入条件检查.5.事件用于记录合约执行过程中发生的各种事件和状态变化 **事件 Event**——非常重要的功能.ytf 通过 Logs 实现 Events(事件)功能.日志内容是交易收据(TransactionReceipts)的一部分,整个日志内容,包括 Receipts 的其它内容会生成一个 ReceiptsRoot 存储在区块的头部.日志永远存在.通过设置一些过滤条件,来获取某些特定的事件.可以用于追溯合约的历史状态,设置检查点等.**预言机 Oracle** 定义•BC 外信息写入 BC 内的机制,一般被称为预言机.预言机的**功能**就是将外界信息写入到 BC 内,完成 BC 与现实世界的数据互通.•它允许确定的智能合约对不确定的外部世界作出反应,是*智能合约与外部进行数据交互的唯一途径*,也是 *BC 与现实世界进行数据交互的接口*.•简言之,Oracle 可以视作 SmartContracts 与外部交互的 API.**为什么需要预言机?** •BC 是确定性的环境,它不允许不确定的事情或因素,智能合约不管何时何地运行都必须是一致的结果,所以虚拟机(VM)不能让智能合约有直接的 networkcall(网络调用),否则结果就可能是不确定的.•智能合约无法主动去获取链外的数据,只能被动接受数据.**工作流程**•用户的智能合约把请求给链上 Oracle 合约,通过链下的 API 接口获得外部数据.外部把数据给链上的 Oracle 合约,然后 Oracle 合约再把数据给用户的智能合约.**应用场景**•一切需要与链下进行数据交互的 DApp 都需要预言机.**当前的问题**•中心化预言机是非常易受攻击的.大多数项目选择使用自己搭建的预言机服务,但实际上这会导致单点故障,易受攻击等问题.因此,去中心化预言机在未来将成为更好的方案.•链上是无法生成随机数的,或者说生成的随机数是可以被预测和破解的.比如说,大多数博彩游戏都是在链上生成随机数,所以很容易被预测和破解,导致资金被盗.这时候就需要预言机从外部给智能合约一个不可预测的随机数.**Oracle 分类**•基于 DATASOURCE: •Software oracles.在互联网上处理来自网络的数据,通过浏览这些数据来找到它需要的信息,提取所需的信息,并将其返回到智能合约中•Hardware oracles.通过扫描仪和传感器等设备直接从物理世界收集数据•Humanoracles.依靠人的行为向 BC 系统提供外部数据.Human oracles 为智能合约提供了问题的答案.•基于 TRUSTMODEL: •Centralizedtrustmodel.集中式信任模型依赖于来自单一来源的数据.Centralizedtrustmodel 的效率很高,但其可用性、可访问性和数据有效性仅依赖于一个节点,可能出现单点故障.•Decentralizedtrustmodel.解决了 Centralizedtrustmodel 中的单点失效问题.然而与集中式信任模型相比,在数据处理上有更高的延迟,效率更低.•基于 DESIGNPATTERN•Request-response.当数据空间太大而无法存储在智能合约中,并且预期用户一次只需要整个数据集的一小部分时,将使用请求-响应设计模式.此设计模式的一个适用用例是数据提供程序.这种设计模式可以实现为链上智能合约系统,其中对 oracle 的请求被发起,而链外基础设施用于监视请求、检索和返回数据.•Publish-subscribe.有效地为预期会发生变化的数据(如价格和天气信息)提供广播服务.此模式类似于 RSSfeeds,其中使用新信息更新 oracle,并使用一个标志表示订阅者可以使用新数据.订阅者可以轮询 oracle 以检查最新信息是否已更改,也可以侦听 oracle 契约的更新并在发生时采取行动.•Immediate-read.用于提供即时决策所需的数据,如学位认证和拨号代码.这种类型的 oracle 将数据存储在其合约存储中,并可以从中查找数据并返回.oracle 存储中的数据可以让任何其他智能合约使用对 oracle 合约的请求调用进行查询,也可以让支持 BC 的应用程序直接查询这些数据.•基于 INTERACTION: Inboundoracles.将外部世界的数据插入到 BC 中.一个例子就是资产的价格,当它达到期望的价格时,就可以自动购买.•Outboundoracles.允许智能合约向外部世界发送数据.智能锁是一个例子,当在智能锁的 BC 地址上收到付款时,实体世界中的智能锁会自动解锁.**目前预言机项目和解决方案**•Oraclize: 为 ytf 提供中心化预言机服务.是中心化的,目前只能在 ytf 网络使用,而且 gas 费较高.•Chainlink: ytf 上第一个去中心化预言机解决方案.•Chainlink 的解决方案是通过在链上的智能合约和链下的数据节点,奖励机制和聚合模型的方式,进行数据的请求和馈送.•Chainlink 的链式聚合成本较高,拓展性差,基于声誉系统容易集中化.•欧链 OracleChain: EOS 上的第一个去中心化预言机解决方案.•欧链采用自主的 PoRD 机制(Proof-of-Reputation&Deposit),本质上是一种抵押代币奖励机制的声誉系统.奖励随节点惩罚作恶节点,可以实现 Augur、Gnosis 等预测市场应用的功能,还能支撑对链外数据有更高频率访问需求的智能合约业务.•DOSNetwork: 支持多条主流公链的去中心化预言机服务网络•DOSNetwork 是一个 Layer-2 的预言机解决方案,它通过在链上部署一个轻量级智能合约,在链下是一个 p2p 网络.服务方式的选取和数据验证采用 VRF+阈值签名等技术,保证了去中心化和数据安全,并达到快速反应.•可以适配所有主流公链,比如 ytf、EOS、Tron、ThunderCore、Ultrain 等.**去中心化应用** DApp 是一个大部分或者全部去中心化的应用.•在设计一个去中心化应用的时候,通常会考虑以下方面: 1.后端软件(一般去中心化,放在智能合约上)2.前端软件 3.数据存储 4.消息通信 5.域名解析,在考虑这些可能是中心化的,也可能是去中心化的.•例如,可以将前端开发为运行在中心化的 Web 服务器上,或者开发为在个人设备上运行的移动端应用.后端和数据存储也可以部分运行在中心化的服务器上,或者是都交给智能合约和 P2P 存储.**DApp 优点**•弹性: •得益于用智能合约管理后端逻辑,只要智能合约所在 BC 平台一直稳定工作,DApp 就不会停机.•透明: •DApp 的上链特征使得任何人都可以对代码进行检查,审核功能.去 DApp 的交互历史都保存在 BC 上.•审查阻力: •只要用户能够访问 ytf 节点,就可以一直访问 DApp 而不会受到任何集中式控制力量的干扰.一旦代码被部署到 BC 网络,任何服务商甚至智能合约拥有者都无法对代码进行修改.**DApp 后端(智能合约)开发**•在一个 DApp 中,智能合约用来存储业务逻辑和相关状态信息,在设计时有以下两个主要因素需要考虑: 1.智能合约的大小: •由于 ytf 的任何计算操作都需费用,若计算太过复杂,用户需要提供更多的交易费,降低了参与热情.设计者应当尽量将最关键的需要去中心化的逻辑和关键数据存储使用智能合约实现.2.合约代码的更新能力: •更新合约的一种方法是使用版本控制系统.例如,可以设计一个入口通道合约,它将所有调用转发到合约的最新版本,由可更新的地址参数定义.还可以使用名称注册表,并将其更新为指向最近的合约版本.**DApp 前端(Web 用户界面)开发**•不需要专门学习类似 Solidity 这样的智能合约开发语言,使用基础的 Web 开发技术即可,即 HTML,CSS,JavaScript.•前端通常使用 Web3.js 这个 JavaScript 库和 ytf 上的智能合约进行交互.**DApp 数据存储**•正如前面所说,*在 ytf 的智能合约不适合处理和存储大量的数据*,可以考虑将 DApp 的非关键数据和静态数据,如图片、视频、客户应用程序的前端资源(HTML,CSS,JavaScript)等存储在链下.•链下存储的理想方案是使用去中心化的 P2P 存储系统,例如: •IPFS•星际文件系统,是一个去中心化的、内容可寻址的存储系统.可以通过文件内容的哈希值在任意一个 IPFS 节点上将数据取回.目标是在 Web 应用程序的分发上取代 HTTP 协议•Swarm•功能类似于 IPFS,由 ytf 基金会创建,是 Go-Ethereum 工具链的一部分.**DApp 消息通信和域名解析**•**消息通信**: •DApp 的通信环节既可以使用传统的中心化服务器,也可以考虑使用基于 P2P 网络的去中心化的通信协议.比较经典的,例如 Whisper,也是 ytf 基金会的 Go-Ethereum 工具链的一部分.•**域名解析**: •在传统网络上,DNS(域名系统)服务可以将在浏览器中输入的域名解析为对应的 IP 地址.同样的,在 ytf 中,提供了 **ENS**(ytf 名称服务)来提供去中心化的解决方案.ENS 本质上也是一组实现在 ytf 上的智能合约,为其他智能合约提供服务.通过使用 ENS(将 hash 值转换为域名),用户可以通过易记的域名访问 DApp 或者智能合约.**ytf 客户端**•ytf 客户端是一种软件应用程序,它实现了 ytf 协议规范,并通过点对点网络与其他 ytf 客户端通信.•ytf 协议有六种主要实现方式,六种不同的语言:•可以使用 ytf 客户端**同步**到各个 ytf 网络(主网或者测试网络),同步时可以选择全同步或者快速同步.也可以借助客户端**搭建自己的 ytf 网络**•不建议直接同步到 ytf 主网络中去,因为需要很大的存储空间和带宽资源.•通常先借助 Ganacheytf 节点仿真器进行部署测试,再使用 ytf 客户端同步到测试网络进行部署测试,最后再考虑部署到主网上.**Ganache**•Ganache 是用于个人开发的 ytf 节点仿真器,它提供了图形界面.Ganache 搭建的 BC 节点是运行在内存中的,对于每次交易的返回是实时的,不需要像真实的 BC 那样等待默认出块时间后才能知道交易是否成功写入.这种特性使得 Ganache 特别适合开发者在开发过程中用来测试和调试.**Truffle**•Truffle 本身基于 javascript,是用来快速开发 ytf 智能合约的开发框架.使用 Truffle,可以快速地进行环境的配置、solidity 智能合约的编译、脚本化部署、脚本化测试框架、得到合约抽象接口,从而方便地进行整个应用的开发.•Truffle 提供了多个 DApp 应用开发模板,通过使用 truffleunbox 命令,可以方便、快捷地构建并初始化对应的开发模板.例如使用 truffleunboxreact 命令,即可安装 react.js(一种前端开发框架)开发前端的应用模板,开发模板中包含了 react.js 应用访问 ytf 智能合约的所有依赖项.**案例分析**: 去中心化评课社区**整体架构**•表示层主要提供用户与平台的交互和智能合约交互的可视化图形界面.•业务逻辑层与 BC 层的交互通过调用 web3.js 库中封装的 ytfapi 来实现.•BC 层即 ytf 网络,可以是本地的模拟测试网络,也可以是真实地 ytf 主网络.**智能合约**•在去中心化评课社区中,主要用到两个智能合约,CommentRegistry(平台合约)和 CommentAccount(账户合约).

超级账本 HyperledgerFabric **联盟链**•联盟链是由多个机构共同参与管理的 BC,每个组织或机构管理一个或多个节点,其数据只允许系统内不同的机构进行读写和发送.联盟链的各个节点通常有与之对应的实体机构组织,通过授权后才能加入与退出网络.各机构组织组成利益相关的联盟,共同维护 BC 的健康运转.•联盟链内部会指定多个预选的节点为**记账人**,每个块的生成由所有的预选节点共同决定,其他接入节点可以参与交易,但不过问记账过程,其他第三方可以通过该 BC 开放的 API 进行限定查询.**联盟链的特点**•部分去中心化.与公有链不一样,联盟链在某种程度上只属于联盟内部的成员所有,由于其节点不多,达成共识容易,交易速度自然也就快很多.•可控性较强.公有链的节点一般是海量的(如 btb、ytf),要篡改区块数据,几乎是不可能的,而联盟链,只要所有机构中的大部分达成共识,即可更改区块数据.•数据不默认公开,可以提供更好的隐私保护.不同于公有链,联盟链的数据只限于联盟里的机构及其用户才有权限进行访问.•节点是(基本)可信的.公有链: 匿名的,以出块为激励的 BC.联盟链: 实名的,以其上运行的商业业务价值为激励的 BC.**联盟链平台-超级账本(Hyperledger)**超级账本项目致力为透明、公开、去中心化的企业级分布式账本技术提供开源参考实现,并推动 BC 和分布式账本相关协议、规范和标准的发展.作为一个联合项目,超级账本由面向不同目的和场景的子项目构成.所有项目都遵守 Apachev2 许可,并约定共同遵守如下的基本原则: •重视模块化设计: 包括交易、合同、一致性、身份、存储等技术场景;•重视代码可读性: 保障新功能和模块都可以很容易添加和扩展;•可持续的演化路线: 随着需求的深入和更多的应用场景,不断增加和演化新的项目.**Hyperledger 管理结构**•TechnicalSteeringCommittee-技术委员会•GoverningBoard-管理董事会•LinuxFoundation-Linux 基金会 **Hyperledger 项目管理**•超级账本是推动 BC 跨行业应用的开源项目的总称,组织成员可以发起新的 BC 项目,加入到超级账本项目中,但需要遵循 Hyperledger 的生命周期.•Hyperledger 的**生命周期**分为五个阶段,分别为 Proposal(提案)、Incubation(孵化)、Active(活跃)、Deprecated(过时)、EndofLife(结束).成员发起新项目时,首先发起者撰写草案,草案内容包括项目的目标、开发过程、代码维护等信息,提交给技术委员会进行审核,该阶段即为提案阶段;当技术委员会有三分之二通过,则进入孵化阶段,在孵化期将对项目进行开发、测试,直到项目完成;项目参与者对该项目没有异议,项目将进入活跃阶段;经过几年以上时间后,随着技术的进步,该项目跟不上时代,将进入过时阶段,最后被淘汰,结束整个生命周期.**HyperledgerFabric** 是 Hyperledger 中的 BC 项目之一.与其他 BC 技术一样,它有一个账本,使用智能合约,是一个参与者管理交易的系统.•HyperledgerFabric 是一个许可 BC 平台.与允许未知身份参与网络的开放式非许可系统(需要诸如'工作量证明'之类的协议来验证交易并保护网络)不同,HyperledgerFabric 网络的成员需要从可信赖的**成员服务提供者(MSP)**注册.•**许可 BC** 在一组已知、已识别且经常经过审查的参与者之间运作,这些参与者在产生一定程度信任的治理模式下运作.许可 BC 提供了一种方法来保护具有共同目标但可能不完全信任彼此的一组实体之间的交互.通过依赖参与者的身份,许可 BC 可以使用更传统的崩溃容错(CFT)或拜占庭容错(BFT)共识协议,这些协议不需要昂贵的挖掘.•HyperledgerFabric 经过特别设计,具有**模块化的体系结构**,无论是可插拔共识、可插拔身份管理协议(如 LDAP 或 OpenIDConnect)、密钥管理协议或加密库,该平台的核心都经过了配置,以满足企业用例需求的多样性.•HyperledgerFabric 还提供**创建通道**的功能,允许一组参与者创建各自的交易账本.这对于某些网络而言,这是一个特别重要的选择.这些网络中,一些参与者可能是竞争对手,并且不希望他们做出的每笔交易都被每个参与者知晓,例如,他们只向某些参与者提供的特殊价格,而其他人不是.若两个参与者组成一个通道,那么只有这两个参与者拥有该通道的账本副本,而其他参与者没有.•Fabric 支持用 Java、Go 和 Node.js 等通用编程语言编写的智能合约 **Fabric 系统结构** **Fabric 模型**•资产(Assets): 资产定义为可以在网络上交换的几乎任何有货币价值的东西,可以从有形资产(房地产和硬件)到无形资产(合同和知识产权).•链码(Chaincode): 链码的执行与交易排序分开,限制了跨节点类型所需的信任和验证级别,并优化了网络的可伸缩性和性能.•账本功能(LedgerFeatures): 不可变的共享账本对每个通道的整个交易历史进行编码,包含类似 SQL 的查询功能,以实现高效的审计和争议解决.•隐私(Privacy): 通道和私人数据收集私人和保密的多边交易成为可能,这些交易通常是竞争企业和受监管行业在公共网络上交换资产所需的.•安全和成员服务(Security&MembershipServices): 许可成员资格提供了一个可信的 BC 网络,参与者知道所有交易都可以由授权的监管机构和审计师检测和跟踪.•共识(Consensus): 一种独特的一致性方法可以实现企业所需的灵活性和可伸缩性.**Fabric 应用层**提供了 API 事件、SDK•Fabric 提供了 gRPC 接口,gRPC 是一个跨语言的 RPC 框架,是一个比 http 更有效率的接口.•在 API 的基础上,Fabric 官方给出了针对不同语言的 SDK,如 Node.js、Golang、Python 和 Java 等,开发人员可以利用 SDK 开发基于 BC 的应用.•Fabric 采用*异步通信*的模式进行开发.通过在链码中定义事件,然后使用应用程序去监听这些事件.当某个事件被触发的时候,就可以执行预先定义的回调函数.**Fabric 核心层**•包括三大组件: •BC 服务(Blockchain)•链码服务(Chaincode)•成员权限管理(Membership)•BC 提供一个分布式账本平台,BC 代表的是账本状态数据变更的历史过程.•链码包含所有的处理逻辑: 通过调用链码接口来改变世界状态.•**世界状态**是一个键值数据库,用于存放链码执行过程中涉及到的状态变量.•成员权限管理基于 PKI(公开密钥基础设施),平台可以对接入的节点和客户端的能力进行限制.**安全和成员服务**•超级账本交易网络中的所有参与者都知道自己的身份.•**公钥基础设施**用于生成绑定到组织、网络组件和最终用户或客户端应用程序的加密证书.•**数据访问控制**可以在更广泛的网络和通道级别上进行操作和管理.•超级账本结构的这种'许可'概念,再加上通道的存在和功能,有助于解决隐私和保密性是首要问题的场景.**分布式账本**•账本是 Fabric 中所有状态转换的有序、防篡改的记录.状态转换是由参与方提交的链码调用(称作'交易')的结果.每个交易都会产生一组资产键值对,在创建、更新或删除时提交到账本.•账本组成一个 BC 和一个状态数据库.BC 用于将不可变的、按顺序排列的记录存储在区块中,状态数据库来维护当前 Fabric 的状态.*每个通道有一个账本.每个节点为其所属的每个通道保留一份账本副本*.**Fabric 账本功能**•使用基于键的查找、范围查询和组键查询来查询和更新账本•使用富查询语言进行只读查询(若使用 CouchDB 作为状态数据库)•只读历史查询-查询某个关键字的账本历史,数据溯源•交易包含链码中读取的键/值的版本(readset)和链码中写入的键/值(writeset)•交易包含每个背书节点的签名,并提交给排序服务•交易被按序装入区块,并从排序服务'传递'到通道上的节点•节点根据背书策略验证交易,并执行这些策略•在扩展块之前,执行版本检查,以确保所读取的资产状态在链码执行期间未发生变化•一旦交易被验证和提交,就不可改变•通道的账本包含配置区块的定义策略、访问控制列表和其他相关信息•通道包含成员服务提供程序实例,允许从不同的证书颁发机构派生加密材料 **共识**•针对账本和交易提供的对 BC 服务.比如针对接入层交易提供担保(背书)、针对接入层多个交易做排序、交易排序结果打包成区块再通知给组织的锚节点,然后锚节点再通过 goosip 协议广播给组织内的其他节点、节点收到区块验证通过后再同步到分布式账本中.•**Solo 模式**指整个 Fabric 网络依赖于一个 Order 节点.•**Kafka 模式**依赖于一个 KafkaOrder 集群.•**BFT(拜占庭容错)模式**则是去中心化的 Qrder 集群.**Fabric 交易流程** **Fabric 链码** 超级账本支持主流编程语言的智能合约(链码)设计 **BC 应用**,一般由若干部署在 BC 网络中的智能合约,以及调用这些智能合约的应用程序组成.✓用户专注于与业务本身相关的应用程序,而智能合约则封装了与区块链账本直接交互的相关过程,被应用程序调用 **智能合约直接与账本结构打交道**,处于十分核心的位置.✓智能合约代码本质上是为了对上层业务逻辑进行支持;✓智能合约最终会部署在 BC 网络中与账本进行交互.应用程序通过调用智能合约提供的方法接口来实现业务逻辑.(**CA 给用户的 app 办法签发,app 调用智能合约,从而更新账本,用户还可以订阅账本更新事件**)**Fabric 核心概念**•**身份和 MSP•节点•Fabric 网络•账本•通道•链码•策略认证和身份管理**•在一个 Fabric 网络中,不同参与者包括 Peer 节点、排序节点、客户端应用程序、管理员等都需要确定对资源的确切权限以及对参与者在 BC 网络中拥有的信息的访问权限.•同时,与参与者身份相关联的主体信息,也需要可被验证.•Fabric 采用 MembershipServiceProvider(MSP)来定义管理组织内的有效身份规则,使用 X.509 证书作为身份,采用传统的公钥基础结构(PKI)分层模型实现: ■公钥加密,私钥解密;■私钥签名,公钥验证;■数字签名证书的机构(CA),是负责发放和管理数字证书的权威机构,并作为电子商务交易中受信任的第三方,承担公钥体系中公钥的合法性检验的责任;■CA 通过中间 CA 组建信任链,规避风险;■Fabric 提供了内置了 FabricCA 组件帮助用户在 BC 网络中创建 CA.•根 CA 和中间 CA 构成的认证链,提供了身份证明;Fabric 提供了一个内置的 CA 组件,允许在所形成的 BC 网络中创建 CA.•MembershipServiceProvider(MSP)将为 BC 网络中的参与者提供一个允许的身份列表.•在 BC 网络中,MSP 主要在本地节点(localMSP)和通道(channelMSP)两个域内生效.•每个组织(节点)都必须定义一个本地身份管理服务,定义该节点的管理权和参与权.组织、组织的管理员、节点管理员和节点本身都必须有明确的信任根;•**channelMSP** 定义了通道级别的管理权和参与权,识别谁拥有通道级别的权限•**localMSP** 只在本地节点或用户的文件系统实例化,**channelMSP** 会在通道的每个节点实例化,并通过共识保持同步;•若组织需要加入通道,则需要在通道配置中包含组织成员信任链的 MSP,否则,来自该组织身份的交易将被拒绝;•建议每个参与通道的组织都定义 MSP,以提升权限管理的颗粒度,更好的保证组织和用户隐私;•**系统级别的通道**包含参与服务的所有组织的 MSP **MSP 和 BC 网络** **Fabric 网络**是一个为应用程序提供账本及智能合约(Chaincode)服务的技术基础设施.首先,智能合约被用来生成交易,接下来这些交易会被分发给网络中的每个节点,这些交易会被记录在它们的账本副本上并且是不可篡改的.这个应用程序的用户可能是使用客户端应用的终端用户,或者是一个 BC 网络的管理员.在大多数情况下,多个组织会聚集到一起作为一个联盟来形成一个网络,并且它们的权限是由一套在网络最初配置的时候联盟成员都同意的规则来决定的,并且,网络的规则可以在联盟中的组织同意的情况下随时地被改变.•**通道 Channel** 是用于实现 BC 网络中业务的隔离,一个联盟多个通道,每个通道可代表一项业务,并且对应一套账本.通道内的成员为业务参与方(即联盟内的组织),一个组织可以加入多个通道.通道分为: 系统通道、应用通道•**排序节点**通过系统通道来管理应用通道,用户的交易信息通过应用通道传递,对一般用户来说,通道是指应用通道.•通道也可以理解为子链•1 个通道+1 个账本+N 个成员 **FabricNetwork**A:Application-应用 CA: CertificateAuthorities 证书颁发机构 C: Channel 通道 L: Ledger 账本 S: SmartContract 智能合约 P: Peer 节点 O: Ordering 排序 R: Organization 组织 NC: NetworkConfigCC: ChannelConfig 背景设定•科大、滁州学院、工大和省网中心(R1-R4)共同决定搭建一个 HyperledgerFabric 网络;•省网中心(R4)为网络的初始创建者,负责配置网络的初始版本(网络配置 NC 和排序 O),省网中心不参与网络中的数据交易;•省网中心(R4)指派科大为网络的管理员,并共同管理网络配置(NC)和排序服务(O);•科大(R1)和滁州学院(R2)因'共建共享在线课程'项目(A1)有私有通讯(C1)的需求,并由两个组织协商制定了网络规范(CC1);•滁州学院(R2)和工大(R3)因'E 会学'项目(A3)有私有通讯(C2)的需求,并由两个组织协商制定了网络规范(CC2);•科大、滁州学院和工大使用'线上课程资格证书颁发'项目(A2),A2 同时拥有在 C1 和 C2 通信的权限;•各通道(C1 和 C2)遵循排序规则并使用排序规则(O);•科大、滁州学院、工大分别部署了一个节点(P1-P3),其中 P1 同步了 C1 的账本、P2 同步了 C1 和 C2 的账本、P3 同步了 C2 的账本;•应用(A1-A3)通过智能合约(S)访问账本(L1-L2);•组织之间通过 CA 来辨识用户、应用等的合法性**网络演进** 1.省网中心(R4)配置本机构的证书颁发服务 CA4,为机构节点和管理员签发证书,并初始化 Fabric 网络,完成了网络配置 NC4 和排序服务 O4 2.科大(R1)配置本机构的证书颁发服务 CA1 3. 省网中心(R4)指定科大(R1)为该网络的管理员 4.科大和滁州学院组成联盟 X1,由省网中心或科大在 NC 中进完成配置 5.科大和滁州学院小组内达成共识,并形成通道配置 CC1 和通道 C1,C1 通道的数据服从排序服务的调度 6.科大 (R1)的节点 P1 进入网络,并获得了通道 C1 的账本拷贝 L1,账本逻辑上存于 C1,但物理上还是存在各节点的拷贝 L 上的 7.同样的,滁州学院的节点 P2 加入网络,并同步了账本 L1 的拷贝 8.智能合约 S5 分别被部署在科大和滁州学院的 BC 结点 P1、P2 9.客户端应用 A1A2 通过智能合约与通道 C1 进行交互,智能合约 S5 提供了完整的定义来实现对账本 L1 的查询和更新 10.类似地构建 C2 **策略-Policies**•认证和身份管理解决了 BC 网络上的身份认证和管理、参与权确权问题;•策略是一组定义了如何制定决策和达到特定结果的规则,是 Fabric 基础设施上的管理机制,负责管理网络成员在修改网络、通道或智能合约时如何达成一致;例如,它们描述了在通道中添加或删除成员的标准,更改区块的形成方式,或指定背书智能合约所需的组织数量.•简单的说,Fabric 网络上的任意操作都受策略控制;•策略是 Fabric 与 ytf 和 btb 等其他 BC 有所不同的原因之一: •在 ytf 和 btbBC 中,交易可以由任意节点生成和验证 **Fabric 节点**•Peers: 表示组织中的节点,Peer 以区块的形式从 Orderer 节点接收有序的状态更新,维护状态和账本.Peer 节点必然是确认节点.•**确认节点** Comitter: 验证数据并保存到账本•**背书节点** Endorser: 对结果进行背书,返回提案响应给客户端•**锚节点** Anchor: 通道中每个组织(Org)都有一个锚节点,用于同一通道中不同组织的 Peer 节点发现通道内所有 Peer 节点•**Leader 节点**: 节点代表,连接到 Orderer 节点,从 Orderer 节点的批量区块广播给其他节点.•**排序节点** Orderer: 为网络中所有合法交易进行全局排序,并将一批排序后的交易组合生成区块结构.•**证书颁发机构 CA 节点**: 负责组织内部成员用户生成和颁发数字证书.38 排序节点和排序服务 Ordering•**排序服务**在超级账本 Fabric 网络中起到十分核心的作用.所有交易在发送给 Committer 节点进行验证接受之前,需要先经过排序服务进行全局排序.•因为 Fabric 的设计依赖于确定性的共识算法,所以任何由普通节点验证的块都是最终的和顺序一致的•除此之外,将链码执行(发生在普通节点上)的背书与排序分离,使 Fabric 在性能和可伸缩性方面具有优势,消除了在执行和排序由同一节点执行时可能出现的瓶颈.•节点是 Fabric 的基本元素,可以被创建、启动、停止、重新配置甚至删除.它们暴露了一系列的 API,这就可以让管理者和应用程序同这些 API 提供的服务互动;•节点维护了一份或多份账本,并且每个账本都有一个或多个智能合约//链码使用这个账本•账本数量和访问账本的链码的数量之间没有固定的关系.一个 Peer 节点可能会有很多链码和账本.•节点维护的账本和链码具有冗余性,这是 Fabric 网络为了避免单点失效所采用的策略,也是 Fabric 网络去中心化和分布式的特点 **节点和应用程序** **节点和通道**•通道允许一组特定的节点和应用程序在 BC 网络中彼此通信.•将通道视为一个由物理节点集合构成的逻辑结构,节点提供了对通道的访问和管理的控制点.**节点和组织**•Fabric 网络由多个组织而不是单个组织管理.节点是构建这种分布式网络的核心,因为它们属于这些组织,并且是这些组织的网络连接点.**节点和身份**•网络中的每个节点都由其所属组织的管理员分配一个数字证书,当节点连接到通道时,其数字证书通过通道 MSP 标识其所属组织..P1 和 P2 具有由 CA1 发出的标识.通道 C 根据其通道配置中的策略确定来自 CA1 的标识应使用 Org1.MSP 与 Org1 关联.**节点、应用程序和排序节点**通过交易的三个阶段和之前所讲述的模块,详细介绍**一次交易的流程**•第一阶段: **提案阶段** 1.应用程序 A1 生产了交易 T1 和提案 P 2.T1 和 P 被发送给了通道 C 上的节点 P1 和 P2 (这是因为**背书策略**定义了该交易提案被认可的前提是需要获得 P1 和 P2 的认可,也就是共识)3.P1 使用交易 T1 和 提案 P 来执行链码 S1,生成对交易 T1 的响应 R1,提供背书 E1 4.P2 使用交易 T1 提案 P 执行了链码 S1,生成对于交易 T1 的响应 R2,提供背书 E2(节点通过向提案的响应添加自己的数字签名的方式提供背书,并且使用它的私钥为整个交易提案提供签名.背书 E1 相当于'在账本 L1 上的交易 T1 的响应 R1 已经被 Org1 的 Peer 节点 P1 同意了) 5.提案阶段当应用程序从足够多的有效的 Peer 节点那里收到了签过名的提案响应的时候就结束了,进入下一阶段(应用程序可以自由地放弃不一致的交易响应,即便不放弃,更新账本时也会被拒绝)•第二阶段: **排序和将交易打包到区块** 1.应用程序 A1 向排序节点 O1 发送由 E1 和 E2 背书的交易 T1 2. 同时,应用程序 A2 将 E1 背书的交易 T2 发送给排序节点 O1;3. O1 将来自应用程序 A1 的交易 T1 和来自





应用程序 A2 的交易 T2 以及来自网络中其他应用程序的交易打包到区块 B2 中;(一个区块中交易的顺序不一定与排序服务接收的顺序相同: O1 接受的顺序可能是 T1-T6,但在 B2 中交易的顺序是 T1,T2,T3,T4,T6,T5,因为可能有多个排序节点;重要的是,排序服务将交易放入严格的顺序中,并且 Peer 节点在验证和提交交易时将使用这个顺序.)(在 Fabric 中,排序服务生成的区块是最终的,永远不会被重写或删除,确保了没有**账本分叉**;排序节点只做排序,不判断交易内容)•第三阶段: **验证和提交** 1.排序节点将区块分发给连接到它的所有 Peer 节点(并非每个 Peer 节点都需要连接到一个排序节点,Peer 节点可以使用 gossip 协议将区块关联到其他节点)2.每个节点将独立地以确定的方式验证区块,以确保账本保持一致(每个节点都验证区块中的每笔交易,确保得到了所需组织的背书和背书策略相匹配)3.验证完成后,节点 P1 处理区块 B2,在 P1 上的账本 L1 中添加一个新区块.同时,节点 P2 处理区块 B2,从而将一个新区块添加到 P2 上的账本 L1 中(节点 P1 和 P2 上的账本 L1 就会保持一致的更新,并且每个节点都可以通知与之连接的应用程序交易已经被处理)



账本 ledger•Fabric 中,分类帐由两个不同但相关的部分组成-**世界状态和 BC**•BC 被构造为互连块的顺序日志,其中每个块包含一系列交易,每个交易代表对世界状态的查询或更新.•世界状态被实现为数据库,保存一组账本状态的当前值.账本状态表示为键值对.世界状态可以频繁更改,因为可以创建、更新和删除状态.•世界状态中每次账本更改都会有相应的版本号递增.•应用程序提交获世界状态变化的交易,并且交易提交到分类账 BC,但只有获得所需背书的交易才会导致世界状态更新.若没有得到足够的背书人的签名,则不会导致世界状态的改变.•世界状态中将**业务对象属性的当前值**保存为唯一的账本状态.•BC 则是世界状态中的业务对象如何达到其当前状态的历史记录,记录了每个账本状态的所有以前版本及其更改方式.•B0 是 BC 中的第一个区块,即创世区块;Bn 具有块头 Hn,其中包含 Dn 中所有事务的加密哈希以及 Hn-1 的哈希,并由此将区块前后相接,组成 BC.**区块**•**区块头**: 包含三个字段,区块编号、当前块哈希、上一个块头哈希•**区块数据**: 包含按顺序排列的交易列表,它是在排序服务创建块时写入的.•**区块元数据**: 包含区块创建者的证书和签名,用于被网络节点验证区块.区块确认节点将每个交易的有效/无效指示符添加到位图中,该位图也位于区块元数据中,以及直到并包括该块的累积状态更新的哈希,以便检测状态分叉.与区块头和区块数据不同,*此部分不是区块哈希计算的输入*.**交易**•块 B1 的区块数据 D1 中的交易 T4 由区块头 H4、交易签名 S4、交易提案 P4、交易响应 R4 和背书列表 E4 组成.•**交易头**保存有关交易的一些基本元数据,例如,相关链码的名称及其版本.**智能合约和链码**•智能合约定义了被添加到账本中的新事实的可执行逻辑;•**链码是智能合约的集合**,管理者如何将智能合约打包用于部署;•类比于传统的 http 协议,一个 Method 和支持的 Action 相当于一个智能合约,Method 的组合部署类似于链码•智能合约以编程方式访问账本两个不同的部分: •BC: 记录所有交易的历史,且记录不可篡改•世界状态: 保存这些状态当前值的缓存,是一个经常需要用到的对象的当前值•智能合约主要在**世界状态中将状态写入(put)、读取(get)和删除(delete)**,还可以**查询**不可篡改的 BC 交易记录.•读取(get)操作一般代表的是查询,目的是获取关于交易对象当前状态的信息.•写入(put)操作通常生成一个新的业务对象或者对账本世界状态中现有的业务对象进行修改.•删除(delete)操作代表的是将一个业务对象从账本的当前状态中移除,但不从账本的历史中移除.•**链码和背书策略**•每个链码都有一个背书策略与之相关联,该背书策略适用于此链码中定义的所有智能合约,指明了 BC 网络中哪些组织必须对一个既定智能合约所生成的交易进行签名,以此来宣布该交易有效.•所有的交易,无论是有效的还是无效的,都会被添加到分布式账本中,但仅有效交易会更新世界状态.•**背书策略是 HyperledgerFabric 与 ytf 或 btb 等其他 BC 的区别所在**.**有效交易** **私有数据-PrivateData**•若一个通道上的一个组织需要对该通道上的其他组织保持数据私有,则可以选择创建一个新通道,其中只包含需要访问数据的组织.但是,在每种情况下创建单独的通道会产生额外的管理开销(维护链码版本、策略、MSPs 等),并且在保留一部分数据私有的同时,希望所有通道参与者都看到事务的情况是不允许的,从 v1.2 开始,Fabric 提供了创建私有数据集合的功能,它允许在通道上定义的组织子集能够背书、提交或查询私有数据,而无需创建单独的通道.