Matrix 共识算法设计1.0 （整理版）

V 1.1

Crane Lv

2018-07-16

目录

[Matrix 共识算法设计1.0 （整理版） 1](#_Toc519506504)

[交易修改 3](#_Toc519506505)

[第一版要求 3](#_Toc519506506)

[第一版设计方案 3](#_Toc519506507)

[定时交易 5](#_Toc519506508)

[设计目的： 5](#_Toc519506509)

[延时状态机 6](#_Toc519506510)

[定时交易设计 6](#_Toc519506511)

[策略交易（交易事务） 6](#_Toc519506512)

[增加策略状态机【废弃】 6](#_Toc519506513)

[策略智能合约 6](#_Toc519506514)

[策略结果加密方案 6](#_Toc519506515)

[策略交易设计 7](#_Toc519506516)

[策略交易流程 7](#_Toc519506517)

[区块结构定义 8](#_Toc519506518)

[区块结构增加 8](#_Toc519506519)

[顶层节点交易广播 9](#_Toc519506520)

[顶层节点交易广播步骤 9](#_Toc519506521)

[交易编号设计 9](#_Toc519506522)

[网络拓扑结构生成方式 9](#_Toc519506523)

[网络拓扑图维护方法 9](#_Toc519506524)

[超级节点 10](#_Toc519506525)

[顶层节点运行方案 10](#_Toc519506526)

[BT下载 11](#_Toc519506527)

[网络拓扑图广播方法 11](#_Toc519506528)

[网络拓扑结构运行方式 11](#_Toc519506529)

[网络节点在线时长统计机制 11](#_Toc519506530)

[网络拓扑结构 11](#_Toc519506531)

[钱包如何连接叶子节点并发送交易 12](#_Toc519506532)

[Matrix共识算法 12](#_Toc519506533)

[共识算法流程 12](#_Toc519506534)

[交易广播及交易验证 12](#_Toc519506535)

[区块验证及挖矿 12](#_Toc519506536)

[Matrix Validator换届期交易处理方法 14](#_Toc519506537)

[将Validator的交易洪泛与交易验证分开 14](#_Toc519506538)

# 交易修改

## 第一版要求

1. 避免以太坊回放攻击，两种方案 ：1. 修改chainID 2. nonce最高bit为1，
   * 1. 【废弃】目前暂定选chainID为101。(chainID在代码中修改，不在交易结构内)
   1. 选第二个方案，最高位为1
      1. 在系统中，每个交易验证前，完成(uint64)Nonce = 0x80…00 | nonce
      2. 网络发送交易时，为了减少网络流量，计算nonce = nonce&0x7F…FF
2. 添加1对多交易
3. 1对1交易与以太坊结构兼容
4. 添加交易类型，默认交易类型为以太坊交易
5. 实现交易区间定时
6. 考虑智能合约集成
7. 1对多交易gas收费标准：
   1. 相当于n个交易：每个交易的gas按照eth的计算方法，gas累加
   2. 交易费优惠，0.7+0.3\*pow(0.9,(n-1))

|  |  |  |  |
| --- | --- | --- | --- |
| 交易数量 | gas折扣 | 普通交易gas | 优惠gas |
| 1 | 1 | 21000 | 0 |
| 2 | 0.97 | 40740 | 1260 |
| 3 | 0.943 | 59409 | 3591 |
| 4 | 0.9187 | 77170.8 | 6829.2 |
| 5 | 0.89683 | 94167.15 | 10832.85 |
| 10 | 0.816226 | 171407.5 | 38592.51 |
| 20 | 0.740526 | 311020.7 | 108979.3 |
| 30 | 0.71413 | 449902.1 | 180097.9 |
| 40 | 0.704927 | 592138.6 | 247861.4 |
| 50 | 0.701718 | 736803.8 | 313196.2 |
| 60 | 0.700599 | 882754.7 | 377245.3 |
| 70 | 0.700209 | 1029307 | 440693 |
| 80 | 0.700073 | 1176122 | 503877.7 |
| 90 | 0.700025 | 1323048 | 566952 |
| 100 | 0.700009 | 1470019 | 629981.4 |

## 第一版设计方案

想法：第一版的做法是扩展签名数据R,S,V的v值，以太坊的v值小于50

设计方案：

1. 交易分两种，一种为原以太坊交易，交易数据为txdata，第二种为Matrix交易，交易数据为txdata\_mx（扩展方法可参考下面，也可重新设计）
2. 以太坊的v值小于50,v += 交易类型\*128，交易类型 0 为原以太坊，1为Matrix
3. 修改代码，EncodeRLP与DecodeRLP。NewEIP155Signer中Hash函数。判断交易类型，如果类型为0，使用原以太坊数据结构以及函数。如果交易类型为1，则需要打包扩展数据，加密hash也要包括扩展数据，同时修改v值
4. RLP处理方法（仅供参考）
   1. Matrix\_Extra拥有三个字段，TxType，LockTime，ExtraTo
   2. txdata后加一个Extra []Matrix\_Extra，标记为`rlp:”tail”`，这样，Extra为空时(非nil)不会写多余字节，有数据时，会添加到尾部, 如果Matrix\_Extra这三个数据有值，则给txdata.Extra赋值，否则tadata.Extra长度为0
5. 时间锁(Timelock)【废弃】
   1. 时间锁主要用于延迟支付，即指定某笔交易在某一个区块高度或时间戳才进行支付或可以被使用。
   2. "locktime"参数，用于表示该笔交易在何时被广播到网络中
      1. 0 < locktime < 500million：代表区块高度（block height)
      2. locktime >= 500million：代表时间戳（从1970-1-1开始的Unix时间戳）
      3. locktime=0表示即时上链，没有延迟

以太坊交易：

|  |
| --- |
| *type txdata struct {*  *AccountNonce uint64 `json:"nonce" gencodec:"required"`*  *Price \*big.Int `json:"gasPrice" gencodec:"required"`*  *GasLimit uint64 `json:"gas" gencodec:"required"`*  *Recipient \*common.Address `json:"to" rlp:"nil"` // nil means contract creation*  *Amount \*big.Int `json:"value" gencodec:"required"`*  *Payload []byte `json:"input" gencodec:"required"`*  *// Signature values*  *V \*big.Int `json:"v" gencodec:"required"`*  *R \*big.Int `json:"r" gencodec:"required"`*  *S \*big.Int `json:"s" gencodec:"required"`*  *// This is only used when marshaling to JSON.*  *Hash \*common.Hash `json:"hash" rlp:"-"`*  *}* |

Matrix交易

|  |
| --- |
| *Type Tx\_to struct{*  *Recipient \*common.Address `json:"to" rlp:"nil"` // nil means contract creation*  *Amount \*big.Int `json:"value" gencodec:"required"`*  *Payload []byte `json:"input" gencodec:"required"`*  *}*  *Type Matrix\_Extra struct{*  *TxType Byte `json:"txType" gencodec:"required"`*  *LockTime Uint32 `json:"lockTime" gencodec:"required"`*  *ExtraTo []Tx\_to `json:"extra\_to" gencodec:"required"`*  *}*  *type txdata struct {*  *AccountNonce uint64 `json:"nonce" gencodec:"required"`*  *Price \*big.Int `json:"gasPrice" gencodec:"required"`*  *GasLimit uint64 `json:"gas" gencodec:"required"`*  *Recipient \*common.Address `json:"to" rlp:"nil"` // nil means contract creation*  *Amount \*big.Int `json:"value" gencodec:"required"`*  *Payload []byte `json:"input" gencodec:"required"`*  *// Signature values*  *V \*big.Int `json:"v" gencodec:"required"`*  *R \*big.Int `json:"r" gencodec:"required"`*  *S \*big.Int `json:"s" gencodec:"required"`*  *// This is only used when marshaling to JSON.*  *Hash \*common.Hash `json:"hash" rlp:"-"`*  *Extra []Matrix\_Extra ` rlp:"tail"`*  *}* |

修改函数

|  |
| --- |
| *EncodeRLP，Decode不需要修改了*  *修改创建交易newTransaction函数即可：*   1. *如果txtype=0,locktime=0,len(extraTo)=0,则 txdata.Extra = []* 2. *否则，txdata.Extra 需要添加参数* |
| *func (s EIP155Signer) Hash(tx \*Transaction) common.Hash {*  *if tx.data.Txtype == 0 // 以太坊*  *return rlpHash([]interface{}{*  *tx.data.AccountNonce,*  *tx.data.Price,*  *tx.data.GasLimit,*  *tx.data.Recipient,*  *tx.data.Amount,*  *tx.data.Payload,*  *s.chainId, uint(0), uint(0),*  *})*  *else //Matrix*  *return rlpHash([]interface{}{*  *tx.data.AccountNonce,*  *tx.data.Price,*  *tx.data.GasLimit,*  *tx.data.Recipient,*  *tx.data.Amount,*  *tx.data.Payload,*  *tx.data.Extra.TxType*  *tx.data.Extra.LockTime*  *tx.data.Extra.ExtraTo*  *s.chainId, uint(0), uint(0),*  *})*  *}* |
| *Web3接口*  *type SendTxArgs struct {*  *From common.Address `json:"from"`*  *To \*common.Address `json:"to"`*  *Gas \*hexutil.Uint64 `json:"gas"`*  *GasPrice \*hexutil.Big `json:"gasPrice"`*  *Value \*hexutil.Big `json:"value"`*  *Nonce \*hexutil.Uint64 `json:"nonce"`*  *// We accept "data" and "input" for backwards-compatibility reasons. "input" is the*  *// newer name and should be preferred by clients.*  *Data \*hexutil.Bytes `json:"data"`*  *Input \*hexutil.Bytes `json:"input"`*  *TxType Byte `json:"txType"`*  *LockTime Uint32 `json:"lockTime"`*  *ExtraTo[] {to,value,input}json:"extra\_to"`*  *}* |

## 定时交易

### 设计目的：

1. 解决定时交易问题：保证在固定区块高度上线交易
2. 解决交易策略问题：发送多个定时交易，得到组合策略（后面详细解释）
3. 存储区块扣存储gas，计算默认gas+存储gas。
4. 执行区块扣执行gas，默认gas=10000，再加上执行步骤的gas
5. 交易中填的Gas要大于等于以上两步的gas的和，如果执行gas超过交易填的gas，报错，gas不够（同以太坊）

### 延时状态机

首先设计一个延时状态机，保存在某个区块高度上链的交易hash，按照区块高度索引，独立于以太坊的状态机。延时状态机的Mekle根要保留到区块头中

1. 延时状态机以区块高度为索引，根据区块高度得到所有交易hash

### 定时交易设计

1. 定时交易发送即可上链，验证交易签名与交易数据的正确性，不验证账户状态的正确性，扣除gas费用，但不执行交易内容，即不改变账户的当前状态。
2. 上链交易包括状态执行成功或不成功（以太坊源代码有），同时包括未执行状态
3. 验证者出块时，规定延时状态机中的交易先执行，先读取延时状态机中的交易，并执行交易内容，设置交易状态为成功或不成功，成功则修改账户的当前状态。扣执行gas
4. 删除延时交易状态机中的旧交易hash，修改延时交易状态树Mekle根

## 策略交易（交易事务）

### 增加策略状态机【废弃】

1. 增加策略状态机，即一个statedb，mekle根也放在区块头中
2. Key值设计方法为hash(address+key)
3. 每一个地址可提供(key,value)对
4. 每一个策略添加[Key,Num],[key0,value0],[key1,value1]…[keyn,valuen]
5. 每个策略有锁定高度，达到锁定高度后，策略不能增加或修改
6. 每个策略拥有一个策略私钥，[keyPrivate，p]
7. 另一种方法是用合约（黄志支持用合约）
8. 我们写一个简单的策略合约。复杂合约用户可参照我们做

### 策略智能合约

1. 做一个可支持简单策略交易的智能合约，用户刻在上面发策略交易
2. 给出策略交易智能合约模板，用户可参考制作合约
3. 定制期货、博彩、挂单等智能合约

### 策略结果加密方案

1. 加密算法
   1. Cindy的随机生成私钥为p，公钥为C
   2. Bob生成的数据为m，Bob的私钥为b，公钥为B
      1. Bob计算私钥为hash(b\*C)
      2. Bob生成密文cipher={m+ hash(b\*C)G}
      3. Bob广播密文cipher
   3. Cindy解密结果
      1. Cindy计算私钥为hash(p\*B) = hash(b\*C)
      2. Cindy解密密文 m = m+hash(b\*C)\*G-hash(p\*B)
      3. Cindy公布Bob的结果

### 策略交易设计

1. 策略交易占用策略状态机，所以gas费用要高一些
2. Alice发送策略交易，需要向Cindy请求一个公钥K，Cindy为此次策略交易的公证人，Cindy生成私钥公钥对[k,K]
3. Alice发送策略交易，需要：
   1. 锁定账户LockAcount
   2. 公钥K
   3. 策略交易Tx-strategy, to为nil或者给定特殊地址（合约地址）
   4. 退回交易tx-refund
   5. 状态锁定高度
4. 其他用户Bob在状态锁定高度前发送解决问题交易，用公钥签名，签名方式见上
5. 状态锁定高度出块后，Cindy公布私钥k
6. 首先到达策略交易高度，Tx-strategy用私钥k解出所有答案，有正确答案则交易验证成功
7. 到达退回交易，Tx-refund验证账户余额，如果余额满足条件，交易验证通过

### 策略交易流程

1. 假设最常见的交易，Alice需要完成任务A，预付1000MAN，在10个区块期间如果有人算出，则得到1000MAN，否则，退回给Alice。则Alice发送交易策略：
   1. 向Cindy，请求公钥K（交易？）
   2. 交易1：from:Alice, to:lockAccount, value:1000
   3. 2：1：10000
   4. 交易2：from:lockAccount, to:nil，value:1000，key= Result，value =hash( A)，lockTime = n+15,strategyLock = n+10。表示10块后状态锁定，不能添加或修改结果，15块后交易2运行
   5. 交易3：from:lockAccount, to:Alice, value:1000,lockTime: n+20，表示20块后refund
2. 如果Bob在10个块之内完成任务A，则发送交易
   1. 交易1：From:Bob，to:lockAccount，data: key= Result，value=cipher(A)
3. 10个块以后
   1. Cindy发送私钥k交易
4. 15个块后
   1. Alice的交易2(Tx-strategy)执行验证，遍历lockAccount下策略数据，使用私钥k解密策略数据
   2. 如果Bob策略正确，则交易2验证成功，Bob得到1000MAN
5. 20个块后
   1. 如果lockAccount有1000MAN，则Alice的交易3验证成功
   2. 否则，Alice的交易3状态验证失败。

# 区块结构定义

## 区块结构增加

### 区块头增加

1. Validator Leader的账户地址：Leader
2. 网络拓扑结构的变化：NetTopology
3. 顶层节点选举的结果及变化：TopNode
4. Validator的区块签名集：*Signatures*

### 区块增加

1. 投票结果区，包括：
   1. 节点上线，掉线投票

## 区块结构定义

### 区块头定义

*type Signature [65]byte*

*type NetTopology struct{*

*address common.Address*

*Position Byte*

*}*

*type Header struct {*

*ParentHash common.Hash `json:"parentHash" gencodec:"required"`*

*UncleHash common.Hash `json:"sha3Uncles" gencodec:"required"`*

*Leader common.Address `json:"leader" gencodec:"required"`*

*Coinbase common.Address `json:"miner" gencodec:"required"`*

*Root common.Hash `json:"stateRoot" gencodec:"required"`*

*TxHash common.Hash `json:"transactionsRoot" gencodec:"required"`*

*ReceiptHash common.Hash `json:"receiptsRoot" gencodec:"required"`*

*Bloom Bloom `json:"logsBloom" gencodec:"required"`*

*Difficulty \*big.Int `json:"difficulty" gencodec:"required"`*

*Number \*big.Int `json:"number" gencodec:"required"`*

*GasLimit uint64 `json:"gasLimit" gencodec:"required"`*

*GasUsed uint64 `json:"gasUsed" gencodec:"required"`*

*Time \*big.Int `json:"timestamp" gencodec:"required"`*

*NetTopology []NetTopology `json:"NetTopology" gencodec:"required"`*

*TopNode []NetTopology `json:"TopNode" gencodec:"required"`*

*Signatures []Signature `json:"signatures " gencodec:"required"`*

*Extra []byte `json:"extraData" gencodec:"required"`*

*MixDigest common.Hash `json:"mixHash" gencodec:"required"`*

*Nonce BlockNonce `json:"nonce" gencodec:"required"`*

*}*

### 区块结构定义

*Type Vote struct{*

*Proposal interface{}*

*Signatures []Signature*

*}*

*type storageblock struct {*

*Header \*Header*

*Votes []\*Vote*

*Txs []\*Transaction*

*Uncles []\*Header*

*TD \*big.Int*

*}*

# 顶层节点交易广播

## 顶层节点交易广播步骤

* 1. 第一步，发送者发送交易签名中的S值及编号N，打包发送给接收者
  2. 第二步，接收者收到交易后，检查交易是否存在，如果全部存在则结束
  3. 第三步，接收者将缺少的交易编号N打包发送给发送者
  4. 第四步，发送者将上一步请求的交易数据去掉S，加上编号N，发送给请求的接收者
  5. 每个节点维护map<s,tx>,map<N,tx>两个map，如果N存在而S不同，也需要重新请求交易

## 交易编号设计

1. 由于负载均衡的需要，每秒每个节点最大发送200Tx，20个区块后没有验证通过的交易会被丢弃，则编号生存最大数量为200\*15\*20=54000
2. 验证节点和矿工节点顶级节点最大个数为128个
3. 用3byte完成上面的编号设计，节点编号共128个，交易序号共131072个

00000000 1111111 1 11111111 11111111

交易序号共131072个

节点编号共128个

1. 矿工和验证者每个节点分配一个唯一的节点编号，可用nodeId排序编号实现
2. 每个节点发送交易的编号从0开始累加，每次加1，已有编号的交易不重新编号
3. 编号发生重复时，旧交易已经丢弃，小概率没有丢弃的，S值也不相同，修改编号下的交易就行

# 网络拓扑结构生成方式

## 网络拓扑图维护方法

1. 网络拓扑图只保留层级概念，按照二分堆的大小，每下一层节点是上一层的两倍
2. 第一次生成网络拓扑图在genesis block中生成
3. 每过一段时间，需要生成新的网络拓扑图时，需要确定
   1. 重新选举主节点
   2. 新加入的节点
   3. 退出的节点
4. 首先完成主节点更替
5. 将退出的节点的位置以及新选为主节点的位置记录下来
6. 按照在线时长以及抵押财产的权重进行替补
   1. 如果第4步的空位没有占满，用末位补全的方式，从叶子节点补上
   2. 如果第4步的空位占满，还有剩余节点，则在树的末尾添加新节点
7. 如果一个上层节点断线率很高，则与叶子节点中短线率最低的叶子交换
8. 每个节点维护一个网络拓扑图状态结构

## 超级节点

1. 基金会布置3-5台超级节点，提供全节点下载功能
2. 超级节点使用主备服务器的方式，主服务器参与顶层节点广播，只需要收到区块头和交易编号集合就可以形成新的区块，有效减少validator节点的网络压力
3. 主服务器如果出现问题，不能正常从顶层节点同步数据，或者备用服务器不能下载数据，则自动切换到一个备用服务器，备用服务器提升为主服务器，以前的主服务器降为备用服务器。
4. 每个节点都用全网广播的区块头验证区块的正确性
5. 超级节点的另一个作用是validator如果被攻击，则接管区块链验证。
   1. Validator默认最多一次掉1/3，如果出现异常，则超级节点接管
   2. Validator上线后，如果在三个区块高度都有心跳，则加入Validator集合
   3. Validator数量达到要求，则超级节点退出接管

## 顶层节点运行方案

1. 每次选举网络节点时，形成网络节点选举队列，5倍长度，即validator为11\*5个，miner为21\*5个，上链，写入区块头
2. 每届的顶层节点运行时，50%的备份节点，validator为11+5，miner为21+10，从选举队列按顺序替补
3. 顶层节点掉线后，备份节点按顺序顶上，成为顶层节点。后续队列插入到备份节点，备份节点始终为5个。
4. 如果掉线节点再次上线，则排到5个备份节点之后，等待进入备份节点。
5. 备份节点从超级节点的服务器同步数据，不从validator同步数据。
6. 顶层节点不参加算例分配网络拓扑。
7. 顶层节点和备份节点的在线验证
   1. 考虑到矿工和备份节点不需要验证区块，统一设计为节点心跳机制，发给所有验证者（11个），不需要太长（5-10秒）
   2. 连续3个心跳没有收到，则证明该节点掉线
   3. Leader发出节点掉线共识，其它节点签名2/3+剩余权益的75%签名即可通过
   4. 签名掉线共识上区块头
8. 顶层节点和备份节点掉线再次上线
   1. 顶层节点和备份节点上线后，连续发送15个心跳，表示上线
   2. 上线节点自己发送掉线共识(Leader可能不会发送)，其它节点签名2/3+剩余权益的75%签名即可通过
   3. 签名上线共识上区块头
9. 算例分配网络拓扑现在先不考虑，等算例分配方案出来再实现，因为不影响前面的工作
10. 问题
    1. 网络极其差怎么办？例如：11个validator掉了9个，或者作恶，扩容
    2. 网络分片了，validator变成了两组网络怎么办？
    3. 超级节点白名单托管

## BT下载

1. 超级节点的主服务器不做bt服务器，也不提供bt下载
2. 超级节点的备份服务器作为bt服务器，提供bt种子
3. 顶层的validator备份节点提供bt下载，但不提供bt种子

## 网络拓扑图广播方法

1. 前提条件：
   1. 所有的节点必须下载Matrix链
   2. 非Validator主节点不需要下载全节点，下载轻节点即可
2. 网络拓扑图的差值放在区块链的Header里
   1. 所有的节点位置都有统一的编号
   2. Header里保存新变化的[Adress,no]，退出者的编号为0
   3. 目前只有层号，编号为层号即可，一个byte
3. 节点编号方法【废弃】
   1. 节点编号为一个Int
   2. 最高Byte为树编号[1-255]
   3. 剩下的为位置编号

位置编号

树编号

11111111 11111111 11111111 11111111

# 网络拓扑结构运行方式

## 网络节点在线时长统计机制

1. 每个区块都要求一部分节点发一个在线交易(正常收取gas)
2. 奇数块按照每块节点总数/32个节点的速度轮流发送交易，节点排序
3. 偶数快按照上一块的Hash值与节点地址(address)计算：hash ADD address ADD 32 =32
4. 每个轮次为64个区块，如果在64个区块内却一笔交易，则表示该轮次该节点未在线
5. 每个交易上线验证高度为当前高度+5

## 网络拓扑结构

1. 网络拓扑现在分为两个拓扑图：任务分发拓扑图及交易发送拓扑图
2. 任务分发拓扑图自上而下发起，每个父节点负责联通自己的子节点，任务由父节点向子节点方向传递下去【以后考虑】
3. 交易发送拓扑图自下而上发起，系统只给出至上而下的层级关系，每个子节点负责连接上层的父节点，建立3个链接，交易由子节点发送给父节点，必须是下层传给上层

## 钱包如何连接叶子节点并发送交易

1. 钱包连接任意公开的主节点，程序按照一定负载均衡原则返回叶子节点
2. 钱包连接给定叶子节点，并向叶子节点发交易
3. 叶子节点向三个父节点发送交易，父节点随机向自己的三个父节点中的一个向上发送交易
4. 叶子节点用UDP向验证节点广播交易

# Matrix共识算法

## 共识算法流程

主节点共识过程分为两个线程

1. 线程1：交易广播及交易验证
2. 线程2：区块验证及挖矿算法

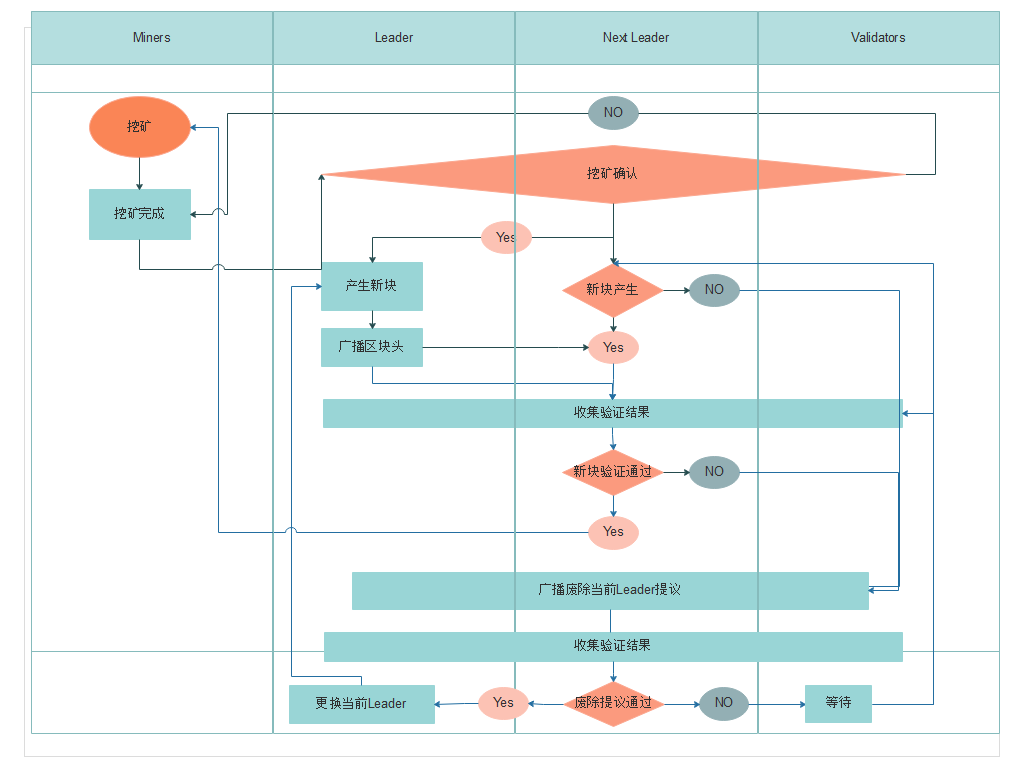
### 交易广播及交易验证

修改内容：验证者只发验证失败的交易

1. 矿工用广播方案广播本节点交易到所有的验证者
2. 验证者用广播方案广播本节点交易到其他验证者
3. 对于每一个交易消息包，验证者挑选出验证失败的交易编号，打包广播给其他验证者
4. 验证者收到75%权益+6节点的失败交易后，从本节点删除该交易

### 区块验证及挖矿

1. Leader发起区块验证，向所有Validator广播新区快，包括区块头及交易编号集。如果在给定时间内没有发起区块验证，则跳入步骤5.
2. 每个Validator向所有Validator广播区块验证结果，Timeout时没发消息的Validator为验证失败。如果在给定时间内没有完成区块验证，则跳入步骤5
   * 1. Var TRUE = 0; FALSE = 1; // true = 0 保证true的时候为原签名
     2. r = k\*G //k为随机数，G为椭圆曲线基点
     3. s = [(hash+ture / false) + r.x\*p]/k //p为私钥
     4. v = r.y % 2 + (ture/false)\*2 //1byte
     5. 签名消息为[r.x,s,v] 共65byte
3. 任何一个Validator收集到满足共识条件的验证签名集合，就可以广播区块头给矿工，矿工开始挖矿
4. 矿工出块方法
5. 矿工出块目前为POW算法
6. 矿工挖矿成功后，将nonce，coinBase，blockHash传给Validator，共(8+20+32)=60
7. 基金会布置备份矿工，当矿工在指定时间内挖不出矿时间(20s)，Validator可以选择备份矿工的挖矿结果。备份矿工与普通矿工同时挖矿，但是不收矿工费
8. 为了保证挖矿时间，每20秒后，挖矿难度折半
9. 每个矿工可提前提供下一个难度（即折半难度）的计算结果，如果在给定时间内，当前难度没有计算出结果，Validator可以马上选择折半难度的计算结果。
10. 出块失败共识
11. Leader接到最新块后广播心跳消息，等待时间为1秒
12. Leader广播心跳消息后，广播新区块验证消息，等待时间为0.5秒
13. Validator接到心跳消息，则等0.5秒超时，否则等1.5秒超时
14. Leader验证签名没有得到足够的签名或失败签名，需要等待1秒，超时
15. 下一个Leader优先发起出块失败共识，等待时间为2秒
16. 出块失败共识未达成，则Leader等1秒再次发起共识
17. 出块失败共识达成，则回到b)。
18. 如果e)超时，则顺推下一个leader发起两个出块失败的共识，依次类推（t0+3）（t1+3）
19. 出块共识消息包括
20. Leader差距值：发起Leader高度-出块失败Leader高度
21. 等待开始消息(上一个区块出块、心跳、新区快验证)
22. 以上消息的签名
23. 其他Validator需要验证
24. 等待开始消息是否正确
25. 时间间隔是否正确
26. 是否满足出块失败的条件
27. 签名消息包括
28. 上一个消息的hash+true/false
29. 计算方法同区块签名验证
30. 如果是区块验证签名时的等待，如果收集了满足共识的签名，则附上自己收集区块正确的签名
31. 出块失败共识 会不会死锁，【通过，不通过，没有达成共识】
32. 首先要满足拜占庭问题的前提，即2/3以上诚实节点
33. TimeOut的问题，多等一轮就达成共识了
34. 区块验签没有达到2/3而等待，就是Timeout问题
35. 区块验签达到2/3，共识失败，肯定有Validator发给矿工挖矿了，为了保证不再次发起出块失败共识，发给矿工挖矿的Validator需要签名的时候附上自己收集区块正确的签名
36. 区块验证以及挖矿流程图



# Matrix Validator换届期交易处理方法

## 将Validator的交易洪泛与交易验证分开

1. 交易网络提前5-10个块切换
2. 切换后新的交易发送给新的validator集合，即下一届validator
3. 新的Validator与旧的Validator一起洪泛广播交易
4. 旧的Validator验证交易

说明：

1. 在换届选举过程中，提前5-10个块，将要退出的Validator就不接收交易了，因为切换了新的交易网络。
2. 新加入的Validator接收交易，与其他Validator一样均分交易压力
3. 整个实现只需要提前切换交易网络即可完成
4. 所有的Validator洪泛交易，接收交易的Validator的tps为99不变，不接收交易的Validator的tps暂定为30%，即30tps，所以理论上是总tps增加了，按照本例的6上6下来算，增加了30\*6=180tps，如果TPS=850，总量25\*6+850 = 1000，网络流量增加了0.5M
5. 验证交易及出块还用旧的Validator不变

网络流量统计

正常情况，1000tps，每个Validator为99tps

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| tps | V-tps | V-R | M-R | Vnum | Mnum | Max Data | Av Data |
| 1000 | 99 | 0.1 | 0.8 | 11 | 32 | 9934985.6 | 5968506 |

换届期间（6上6下）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| tps | V-tps | V-R | M-R | Vnum | Mnum | newV | Max Data | Av Data |
| 1000 | 99 | 0.1 | 0.8 | 11 | 32 | 6 | 11545943.1 | 7579463 |

网络流量增加1.6M

如果按照850TPS算

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| tps | V-tps | V-R | M-R | Vnum | Mnum | newV | Max Data | Av Data |
| 850 | 84.7 | 0.1 | 0.8 | 11 | 32 | 6 | 10420468.4 | 6453988 |

网络流量增加0.5M