Algorand：Scaling Byzantine Agreements for Cryptocurrencies

Yossi Gilad;Rotem Hemo;Silvio Micali;Georgios Vlachos;Nickolai Zeldovich

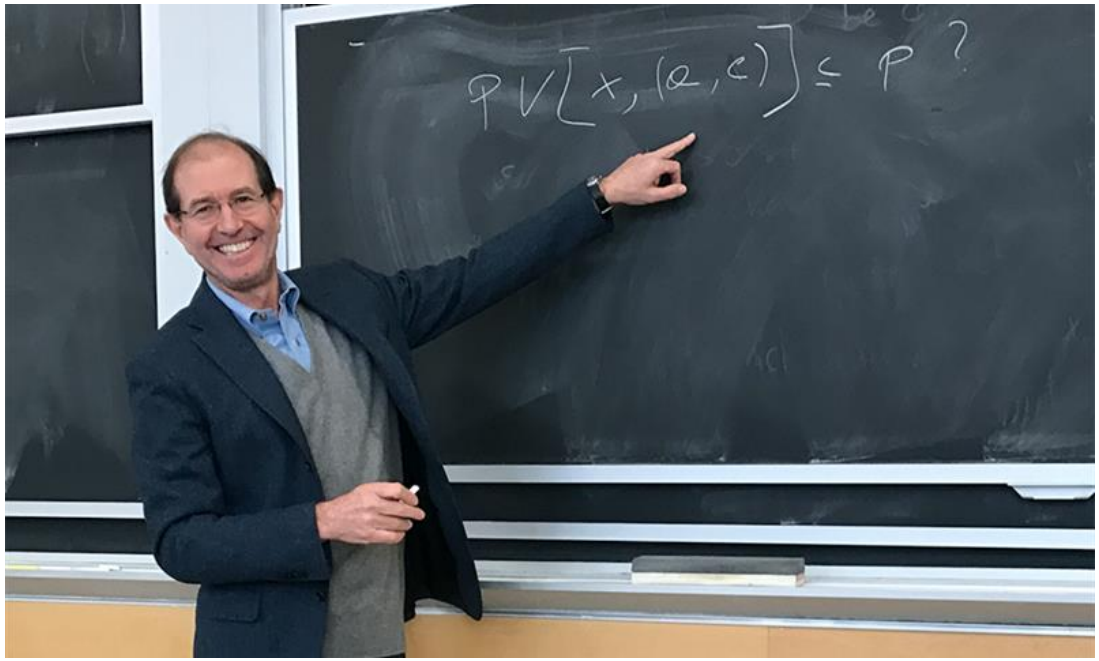
MIT Computer Science and Artificial Intelligence Laboratory

Outline

- Abstract
- Introduction
- Related work
- Cryptographic Sortition
- Block Proposal
- BA ⋆
- Evaluation

Abstract

- Algorand，一种新的加密货币，旨在扩展到多用户情况下能够一分钟内确认交易。Algorand的核心使用称为BA⋆的拜占庭协议，能以低延迟在新块上达成共识，并且没有分叉的可能性。



Introduction -- 比特币区块链系统的几个核心缺陷

- 第一，工作量证明共识机制需要消耗大量计算资源和能源。
- 第二，要求50%以上的计算资源掌握在诚实用户的手中。
- 第三，容易出现分叉
- 第四，可拓展性比较差。

Introduction -- Algorand

- Algorand目标
  - 一分钟内确认交易
  - 杜绝分叉可能
  - 可扩展性
- Algorand面临三大挑战。
  - Algorand必须避免Sybil攻击。
  - BA⋆必须扩展到数百万用户。
  - Algorand必须能够抵御拒绝服务攻击。

权重用户 Weighted users

- 为了防止Sybil攻击，Algorand为每个用户分配权重。 只要用户的加权分数（大于2/3的常数）是诚实的，BA⋆就能保证共识。
- 在Algorand，我们根据用户账户中的资金来分配权重。因此，只要有一小部分（超过2/3）的钱由诚实用户拥有，Algorand就可以避免分叉和双重支出。

委员会的共识 Consensus by committee

- BA⋆通过选择一个委员会（从全部用户中随机选择的一小组代表）来实现可扩展性，以执行其协议的每一步。
- 所有其他用户观察协议消息，从而允许他们学习商定的块。BA根据用户的权重在所有用户中随机选择委员会成员。 这使得Algorand可以确保有足够的委员会成员诚实。

加密抽签 Cryptographic sortition

- 为了防止敌人以委员会成员为目标，BA⋆以私人和非互动的方式选择委员会成员，每个用户可以独立确定他是否被选在委员会中。
- 如果函数指示用户被选中，它将返回一个短字符串，该字符串向其他用户证明此用户的委员会成员资格。
- 由于会员选择是非交互式的，因此在用户开始参与BA⋆之前，攻击者不知道哪个用户被选为委员。

参与者更换 Participant replacement

- 最后，一旦该成员在BA中发送消息，攻击者就能以委员会成员为目标。 BA⋆通过要求委员会成员只参与一次会话来减少这种攻击。
- 因此，一旦委员会成员发送了他的信息（将他的身份暴露给对手），委员会成员就与BA⋆无关。在下一轮会话中将选出新的委员。

Related work

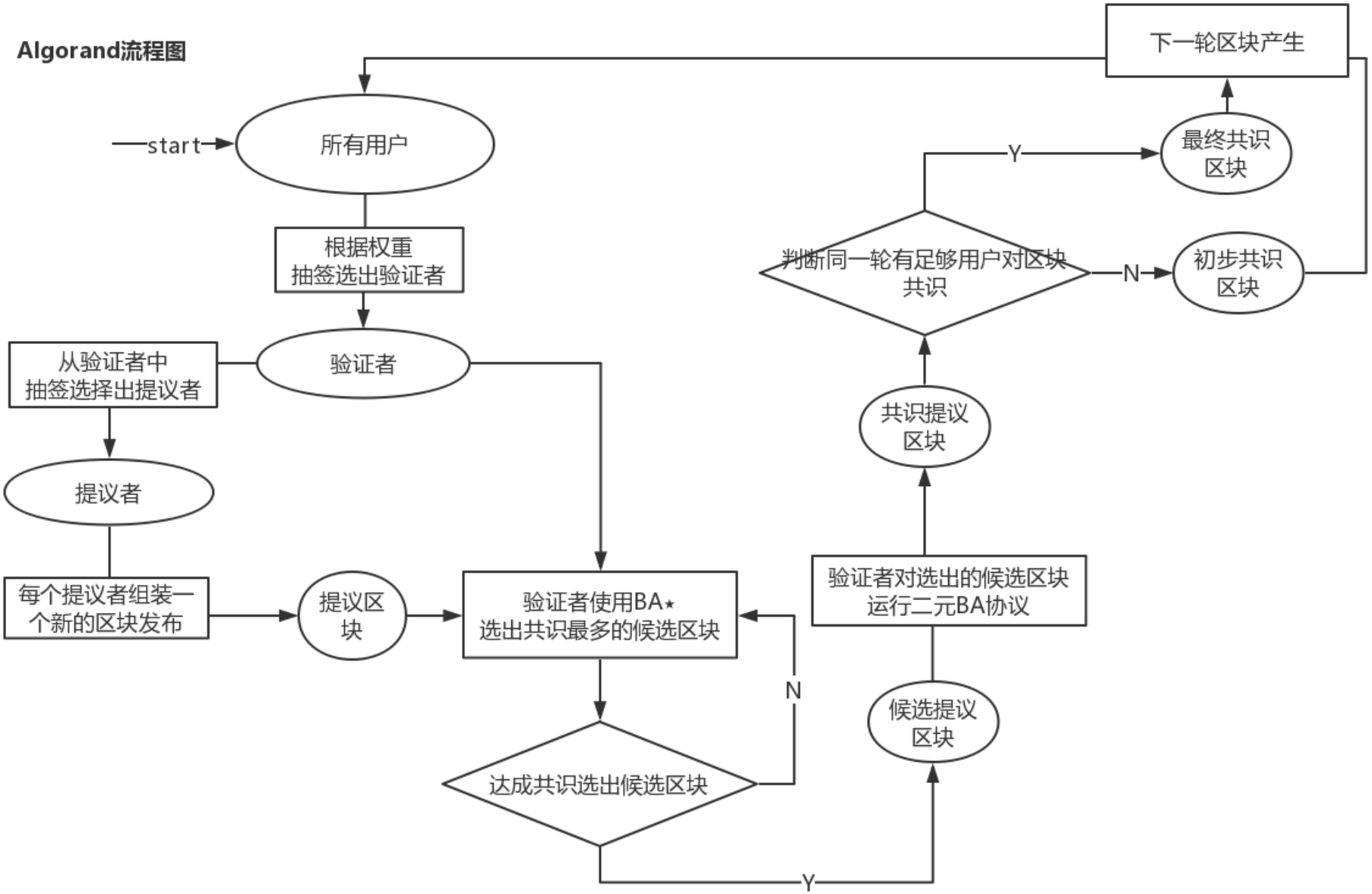
- Pow
- 拜占庭共识，PBFT（拜占庭容错算法）
- Honey Badger（拜占庭共识，指定一组服务器）
- Bitcoin-NG（Pow共识，选举领导者）
- Proof of Stake （股权证明启发）

比特币 PoW

- 比特币和其他加密货币使用PoW （工作量证明）来确保每个人都同意一组已批准的交易，解决了双重支付这个问题，其中用户必须重复计算哈希来增长区块链，最长的链被认为是权威的。 PoW确保对手不会通过创建假名获得任何优势。
- 但是，PoW允许分叉的可能性，其中两个不同的区块链具有相同的长度，并且两个区块链都不相互替代。
- 避免分叉需要两个不幸的牺牲：将链条增长一个区块的时间必须相当长（例如比特币为10分钟），并且应用程序必须等待几个区块才能确保其交易保持在权威链上（在比特币中推荐6个区块 ）。 这就导致，确认比特币交易大约需要一个小时。

拜占庭共识

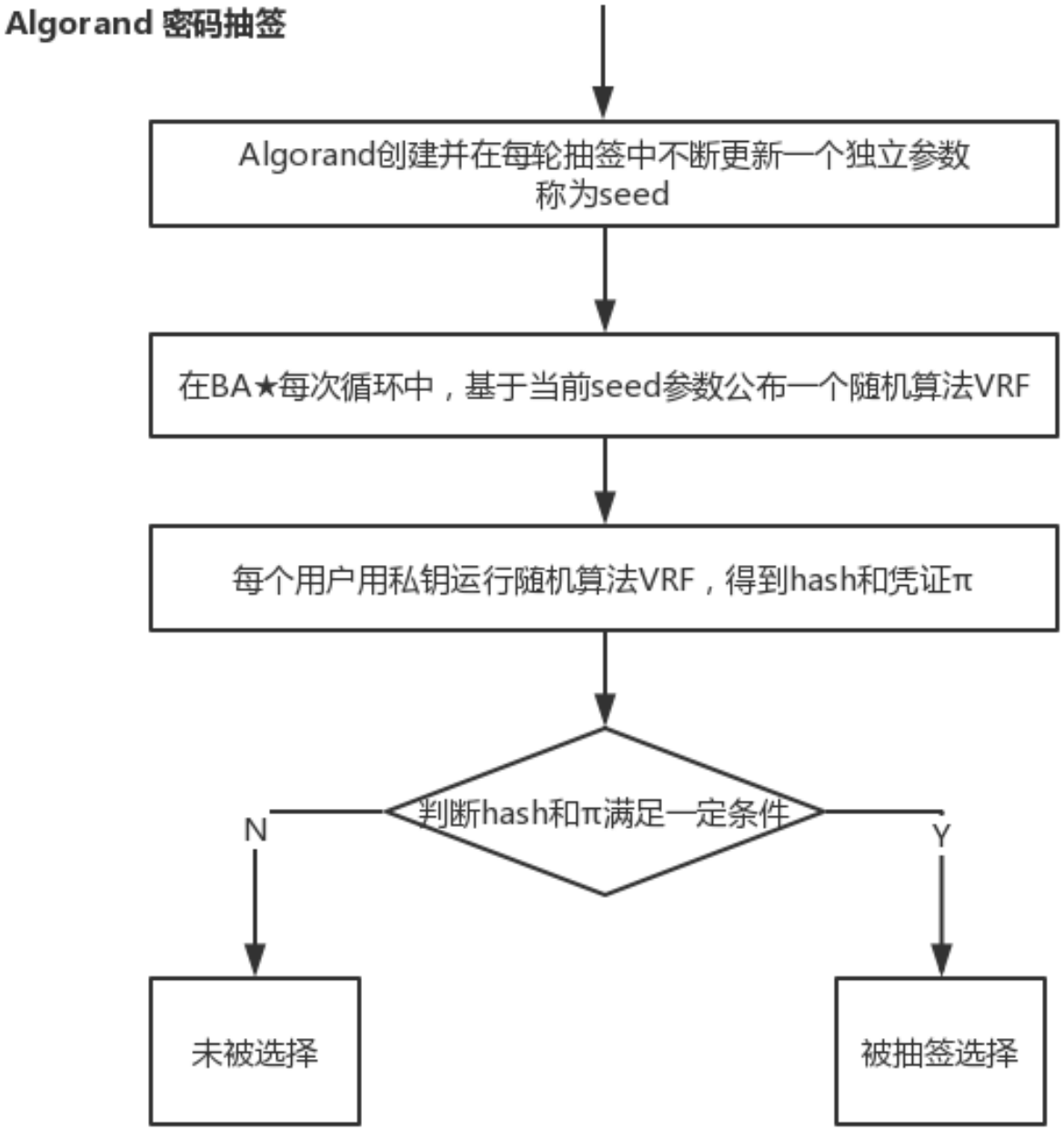
- 依靠拜占庭协议，Algorand消除了分叉的可能性，这种系统可以容忍高达**1/3**的攻击者。这样，交易就能在一分钟内按顺序被确认。
- BA⋆是一个拜占庭式的共识协议，它不依赖于固定的集合服务器，避免了在知名服务器上发生针对性攻击的可能性。
- 通过根据用户的货币余额对用户进行权衡，BA⋆允许用户加入加密货币而不会冒Sybil攻击风险，只要诚实用户持有的钱的比例至少为常数大于2/3。


Algorand流程图
下一轮区块产生
start
所有用户
根据权重
抽签选出验证者
验证者
从验证者中
抽签选择出提议者
提议者
每个提议者组装一
个新的区块发布
提议区
块
验证者使用BA⋆
选出共识最多的候选区块
达成共识选出候选区块
N
Y
候选提议
区块
验证者对选出的候选区块
运行二元BA协议
共识提议
区块
判断同一轮有足够用户对区块
共识
Y
最终共识
区块
N
初步共识
区块

密码抽签 -- 抽签过程

- 首先，Algorand创建并不断更新一个独立参数，称为“种子”。“种子”参数不仅不可能被“敌对者”预测，也不能被其操纵。
- 其次，在BA★每次循环中，Algorand基于当前 “种子”参数构建并公布一个随机算法
- 接着，每个用户使用自己的私钥运行系统公布的随机算法，得到自己的凭证（credential）。凭证值满足一定条件的用户就是这一轮的“验证者”（verifiers）。“验证者”组装一个新区块并连同自己的凭证一起对外发出。
- 最后，所有“验证者”基于“领导者”组装的新区块运行拜占庭协议BA★。在BA★的每次循环中的每一个子步骤中，被选中的“验证者”都是不同的。这样能有效防止验证权力集中在某些用户手中，避免“敌对者”通过腐化这些用户来攻击区块链。

密码


Algorand 密码抽签
Algorand创建并在每轮抽签中不断更新一个独立参数
称为seed
在BA★每次循环中，基于当前seed参数公布一个随机算法VRF
每个用户用私钥运行随机算法VRF，得到hash和凭证π
判断hash和π满足一定条件
N
Y
未被选择
被抽签选择

密码抽签 -- 抽签/验证过程

- sk 用户私钥
- seed 随机种子
- τ 阈值，确定为该role选择用户的预期数量
- r ole 用于区分用户可能被选择的不同角色，验证者或领导者
- w 每个用户的权重
- W 所有用户的权重和
- j 表示此用户被选择了多少次
- hash 返回的hash值
- π 返回的证明值

procedure Sortition($sk, seed, \tau, role, w, W$):

$\langle hash, \pi \rangle \leftarrow \mathrm{VRF}_{sk}(seed||role)$

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

while $\frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k;w,p), \sum_{k=0}^{j+1} B(k;w,p)\right)$ **do**

  j++

return $\langle hash, \pi, j \rangle$

Algorithm 1: The cryptographic sortition algorithm.

procedure VerifySort($pk, hash, \pi, seed, \tau, role, w, W$):

if $\neg VerifyVRF_{pk}(hash, \pi, seed||role)$ **then return** 0;

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

while $\frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k;w,p), \sum_{k=0}^{j+1} B(k;w,p)\right)$ **do**

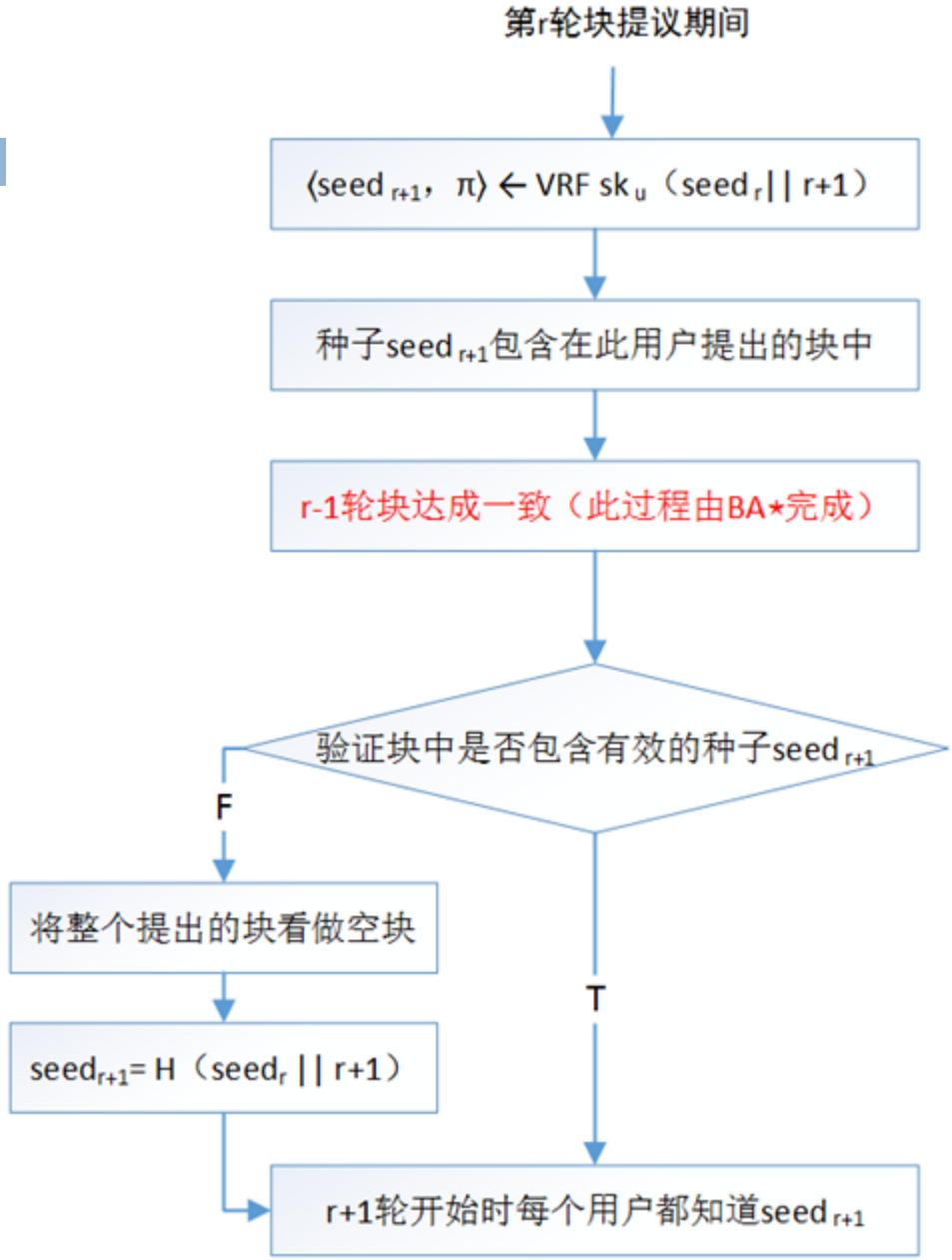
  j++

return j

Algorithm 2: Pseudocode for verifying sortition of a user with public key pk.

密码抽 -- 种子选择

- 在r轮的块提议期间，为第r+1轮块提议计算种子seed $_{r+1}$。
- 如果块中seed无效，则使用seed$_{r-1}$生成新的seed$_{r+1}$



块提议

- 每个提议者提议一个块。然后连同哈希和证明一起传播到网络上。
- 之后，这一轮的所有委员会成员接到消息后，通过BA⋆协议对块达成共识（暂定或最终共识）。如果这一轮是暂定共识，则只有在当后续块（后几轮的块）有达成最终共识的情况下才能确认前几轮的暂定共识块。

块提议 -- 减小块传输

- 为确保每轮都提出一些区块，Algorand为区块角色提议者数目设置为阈值τ大于1，附录实验证明了 τ = 26 可以保证一个合理的提议者数量（1~70）
- 减少不必要的块传输
  - 选择多个提议者的风险是每个人都会传播他们自己的提议块。对于大块（比如1兆字节），这可能会产生显着的通信成本。
  - 为了降低这个成本，使用抽签哈希来区分提议块的优先级，所有块提议者选择的子用户的最高优先级是块的优先级。
  - Algorand传播两种消息：一种仅包含所选块提议者的优先级和证明（来自抽签），另一块包含整个块
  - 这些消息使大多数用户能够了解谁是最高优先级的提议者，并因此迅速抛弃其他提议的块。

块提议 -- 等待时间

- 每个用户都必须等待一定的时间才能通过gossip协议接收块提议，选择此时间间隔不会影响Algorand的安全保证，但对性能很重要。
  - 等待很短的时间将意味着没有收到block提议。
  - 等待时间过长会收到所有block建议，也会不必要地增加确认延迟

块提议 -- 恶意提议者

- 即使某些块提议者是恶意的，最坏的情况是他们欺骗不同的Algorand用户使用不同的块来初始化BA⋆。 这可能会导致Algorand在空白区块达成共识，并可能采取额外措施。
- 然而，事实证明，这种情况相对不太可能。
  - 特别是，如果敌手不是一轮中最高优先级的提案者，那么优先级最高的提案者会向其所有用户传递一致的块。
  - 如果对手是一轮中最高优先级的提议者，他们可以提出空白块，从而防止任何实际交易被确认。 然而，根据Algorand的假设，至少h > 2/3的加权用户是诚实的，这种情况发生的概率至多为1-h。

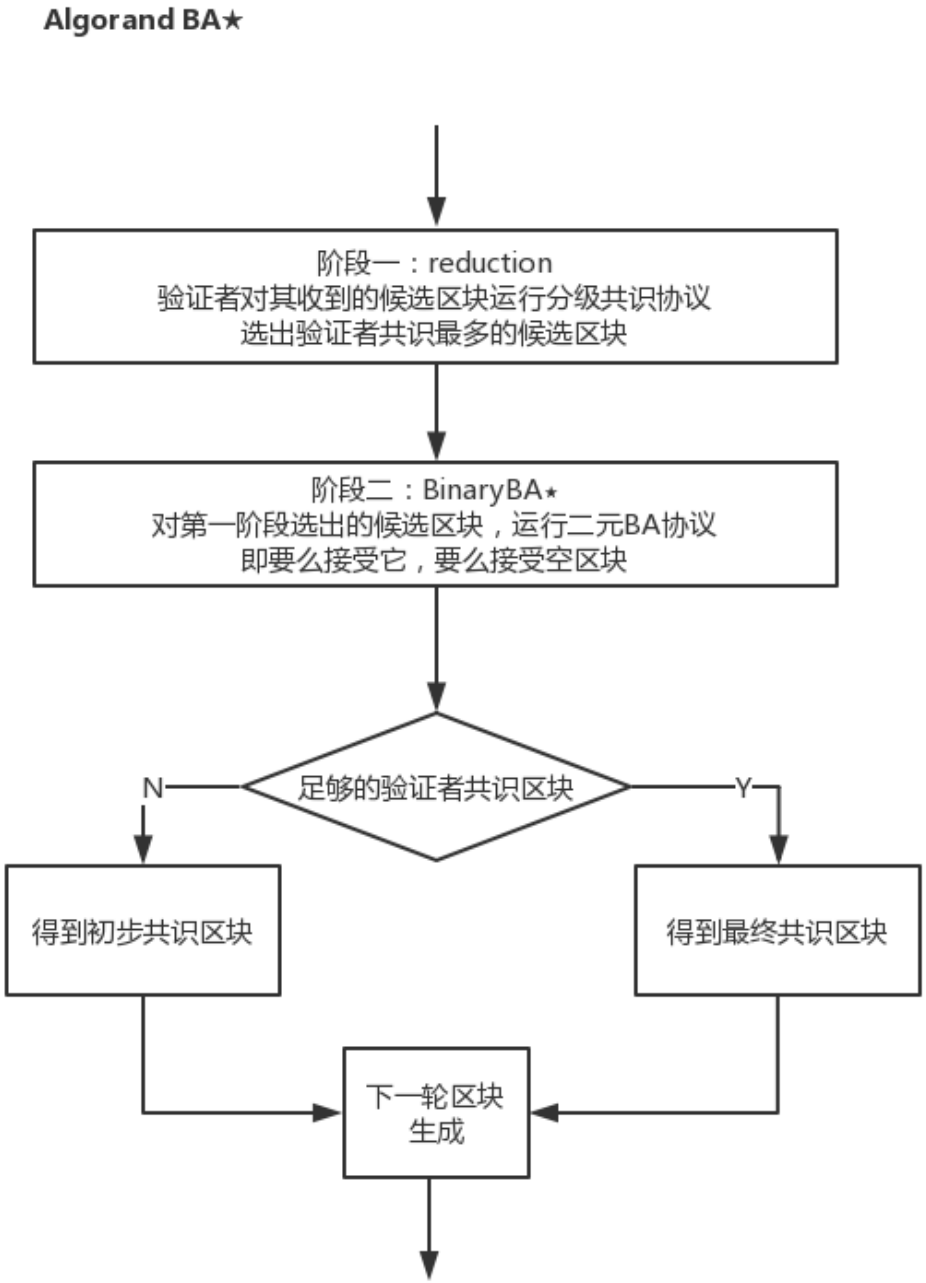
BA⋆

- 块提议不保证所有用户都收到相同的块。
- 为了在一个单独的块上达成共识，Algorand使用BA⋆，可以确保区块链不会分叉。每个用户使用他们接收的最高优先级块初始化BA⋆。

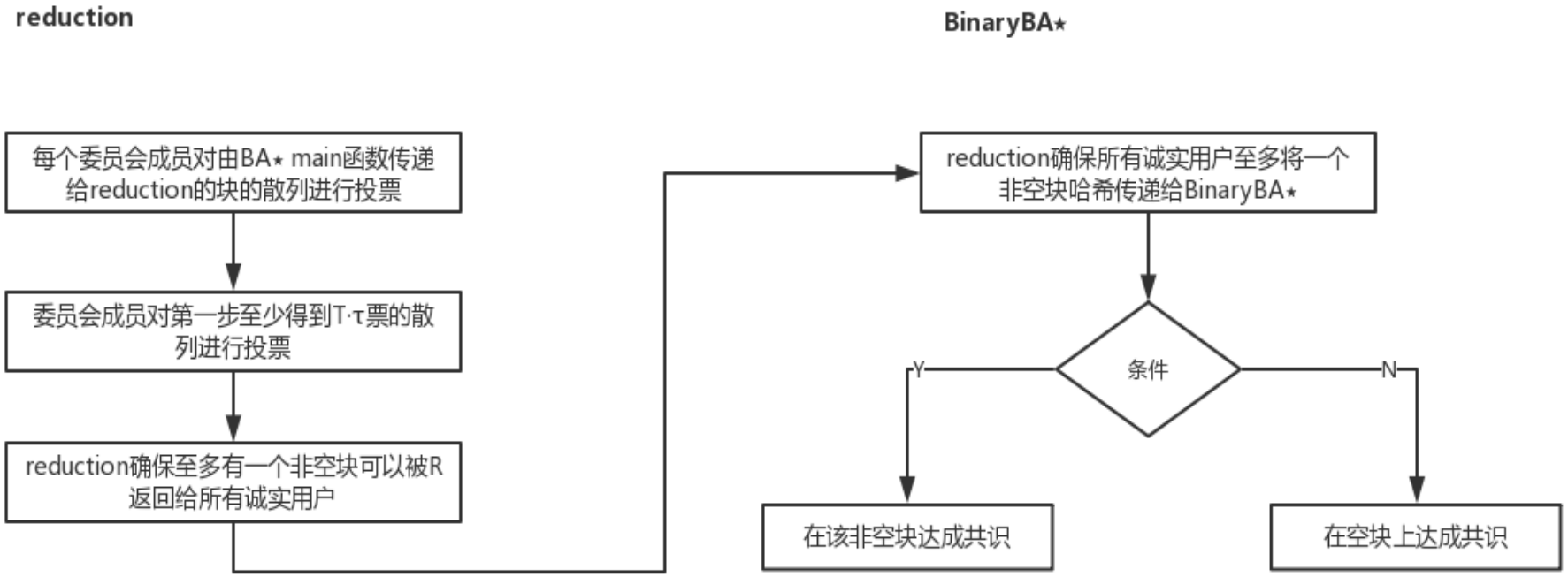
BA⋆

- BA★相当于一个两阶段的投票机制：
  - 第一阶段，将在任意一个块的散列上达成一致的问题转换为在两个值达成一致，这两个值是特定建议的块散列，或者是空块的散列。 程序里是用Reduction这个函数。
  - 在第二阶段，BA⋆就两个选择块之一达成一致：要么同意一个建议的区块，要么同意一个空的区块。程序里是用BinaryBA⋆这个函数。

BA⋆


Algorand BA⋆
阶段一：reduction
验证者对其收到的候选区块运行分级共识协议
选出验证者共识最多的候选区块
阶段二：BinaryBA⋆
对第一阶段选出的候选区块，运行二元BA协议
即要么接受它，要么接受空区块
足够的验证者共识区块
N
Y
得到初步共识区块
得到最终共识区块
下一轮区块
生成

BA⋆


reduction
每个委员会成员对由BA⋆ main函数传递给reduction的块的散列进行投票
委员会成员对第一步至少得到T·τ票的散列进行投票
reduction确保至多有一个非空块可以被R返回给所有诚实用户
BinaryBA⋆
reduction确保所有诚实用户至多将一个非空块哈希传递给BinaryBA⋆
条件
Y
N
在该非空块达成共识
在空块上达成共识

BA⋆

procedure $BA\star(ctx, round, block)$:

$hblock \leftarrow \text{Reduction}(ctx, round, H(block))$
$hblock_\star \leftarrow \text{Binary}BA\star(ctx, round, hblock)$
// Check if we reached "final" or "tentative" consensus
$r \leftarrow \text{CountVotes}(ctx, round, \text{FINAL}, T_{\text{FINAL}}, \tau_{\text{FINAL}}, \lambda_{\text{STEP}})$
if $hblock_\star = r$ **then**
  return $\langle \text{FINAL}, \text{BlockOfHash}(hblock_\star) \rangle$
else
  return $\langle \text{TENTATIVE}, \text{BlockOfHash}(hblock_\star) \rangle$

Algorithm 3: Running *BA⋆* for the next *round*, with a proposed *block*. H is a cryptographic hash function.

BA⋆ -- Reduction函数

- 1.对块的散列进行投票
- 2.选择出至少得到T·τ票的散列，如果没有则返回空块的散列

procedure Reduction(*ctx*, *round*, *hblock*):

// step 1: gossip the block hash
CommitteeVote(*ctx*, *round*, REDUCTION_ONE, τ_{STEP}, *hblock*)
// other users might still be waiting for block proposals,
// so set timeout for $\lambda_{\text{BLOCK}} + \lambda_{\text{STEP}}$
$hblock_1 \leftarrow$ CountVotes(*ctx*, *round*, REDUCTION_ONE, $T_{\text{STEP}}, \tau_{\text{STEP}}, \lambda_{\text{BLOCK}} + \lambda_{\text{STEP}}$)
// step 2: re-gossip the popular block hash
$empty\_hash \leftarrow H(\text{Empty}(round, H(ctx.last\_block)))$
if $hblock_1 = TIMEOUT$ **then**
    CommitteeVote(*ctx*, *round*, REDUCTION_TWO, τ_{STEP}, *empty_hash*)
else
    CommitteeVote(*ctx*, *round*, REDUCTION_TWO, τ_{STEP}, $hblock_1$)
$hblock_2 \leftarrow$ CountVotes(*ctx*, *round*, REDUCTION_TWO, $T_{\text{STEP}}, \tau_{\text{STEP}}, \lambda_{\text{STEP}}$)
if $hblock_2 = TIMEOUT$ **then return** *empty_hash* ;
else return $hblock_2$;

Algorithm 7: The two-step reduction.

BA⋆ -- 发送选票函数

procedure CommitteeVote($ctx, round, step, \tau, value$):

// check if user is in committee using Sortition (Alg. 1)

$role \leftarrow \langle$"committee", $round, step\rangle$

$\langle sorthash, \pi, j\rangle \leftarrow$ Sortition($user.sk, ctx.seed, \tau, role,$
$ctx.weight[user.pk], ctx.W$)

// only committee members originate a message

if $j > 0$ **then**

  Gossip($\langle user.pk, \text{Signed}_{user.sk}(round, step,$
  $sorthash, \pi, H(ctx.last\_block), value)\rangle$)

Algorithm 4: Voting for *value* by committee members. *user.sk* and *user.pk* are the user's private and public keys.

BA⋆ -- 计票函数

- 1.接收投票消息
- 2.统计选票，达到T·τ则返回value
- 3.时间窗口内未收到足够消息则产生超时

procedure CountVotes($ctx, round, step, T, \tau, \lambda$):

```
start ← Time()
counts ← {}     // hash table, new keys mapped to 0
voters ← {}
msgs ← incomingMsgs[round, step].iterator()
while TRUE do
    m ← msgs.next()
    if m = ⊥ then
        if Time() > start + λ then return TIMEOUT;
    else
        ⟨votes, value, sorthash⟩ ← ProcessMsg(ctx, τ, m)
        if pk ∈ voters or votes = 0 then continue;
        voters ∪ = {pk}
        counts[value] + = votes
        // if we got enough votes, then output this value
        if counts[value] > T · τ then
            return value
```

Algorithm 5: Counting votes for *round* and *step*.

BA⋆ --二元拜占庭

```
procedure BinaryBA⋆(ctx, round, block_hash):
step ← 1
r ← block_hash
empty_hash ← H(Empty(round, H(ctx.last_block)))
while step < MaxSteps do
    CommitteeVote(ctx, round, step, τ_STEP, r)
    r ← CountVotes(ctx, round, step, T_STEP, τ_STEP, λ_STEP)
    if r = TIMEOUT then
        r ← block_hash
    else if r ≠ empty_hash then
        for step < s' ≤ step+3 do
            CommitteeVote(ctx, round, s', τ_STEP, r)
        if step = 1 then
            CommitteeVote(ctx, round, FINAL, τ_FINAL, r)
        return r
    step++
    CommitteeVote(ctx, round, step, τ_STEP, r)
    r ← CountVotes(ctx, round, step, T_STEP, τ_STEP, λ_STEP)
    if r = TIMEOUT then
        r ← empty_hash
    else if r = empty_hash then
        for step < s' ≤ step+3 do
            CommitteeVote(ctx, round, s', τ_STEP, r)
        return r
    step++

    CommitteeVote(ctx, round, step, τ_STEP, r)
    r ← CountVotes(ctx, round, step, T_STEP, τ_STEP, λ_STEP)
    if r = TIMEOUT then
        if CommonCoin(ctx, round, step, τ_STEP) = 0 then
            r ← block_hash
        else
            r ← empty_hash
    step++
// No consensus after MaxSteps; assume network
// problem, and rely on §8.2 to recover liveness.
HangForever()
```

Algorithm 8: Binary*BA⋆* executes until consensus is reached on either *block_hash* or *empty_hash*.

实验评估

- 测试在亚马逊云上进行，使用了1000台EC2虚拟机，每个虚拟机具有8个内核和高达1 Gbps的网络吞吐量
- 默认情况下，我们为每个VM运行50个用户，并且用户建议1 MB的块
- 并将每台机器分配到全球20个主要城市之一
- 为每个用户分配相同的资金份额

实验评估 -- 时延

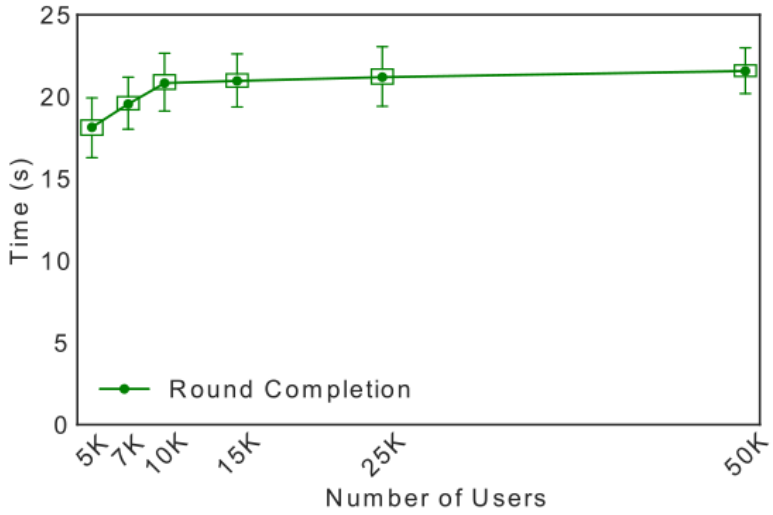


Figure 5: Latency for one round of Algorand, with 5,000 to 50,000 users.

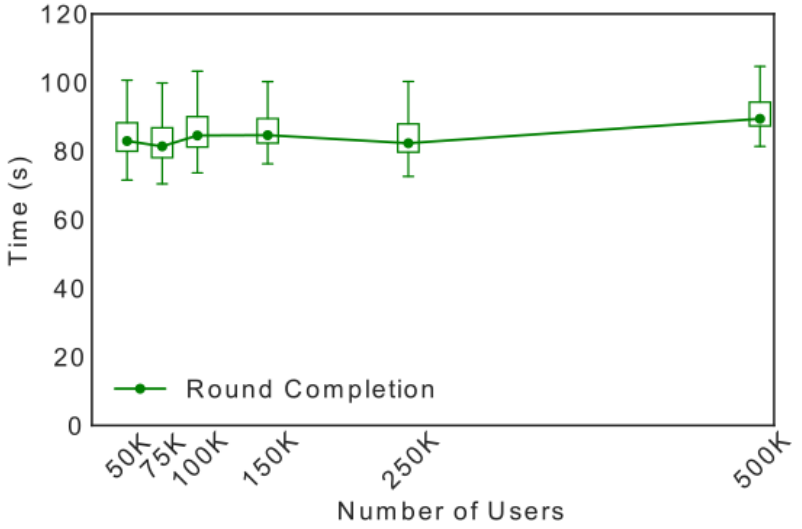


Figure 6: Latency for one round of Algorand in a configuration with 500 users per VM, using 100 to 1,000 VMs.

实验评估 -- 吞吐量

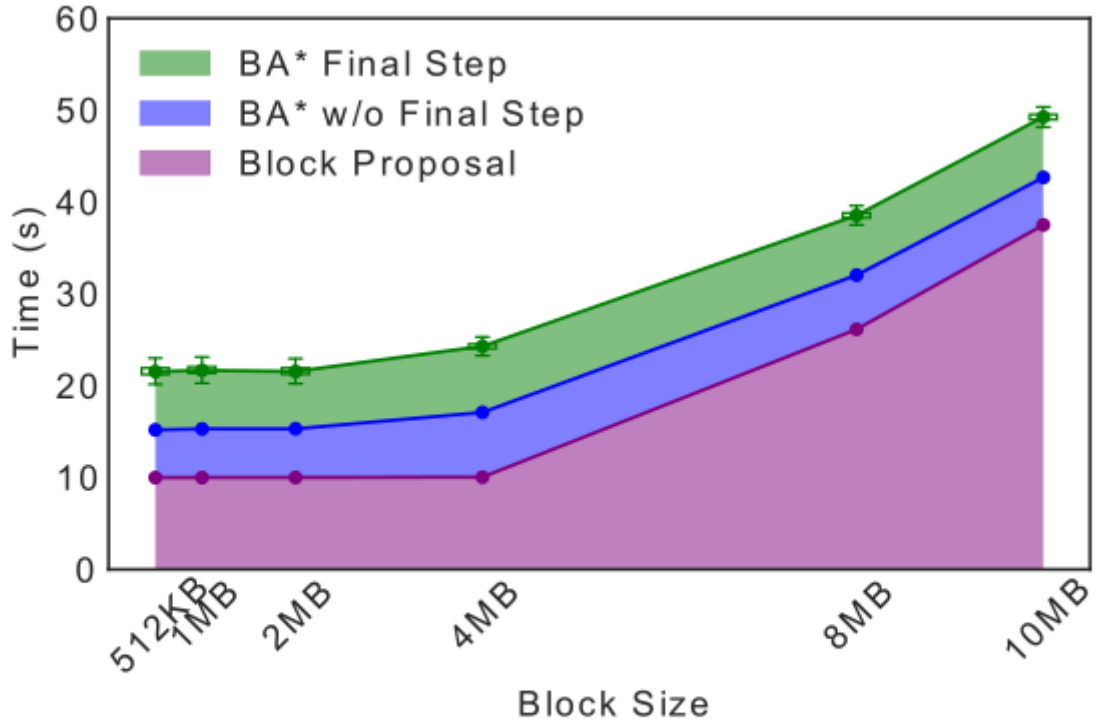


Figure 7: Latency for one round of Algorand as a function of the block size.

实验评估 -- 运行Algorand的开销

- 每个用户的通信成本与运行Algorand的用户数量无关，因为用户拥有预期的固定数量的邻居，他们的通信消息，并且共识协议中的消息数量取决于委员会的规模（而不是用户总数）。
- 在存储成本方面，Algorand存储了块证书，以向新用户证明已经提交了块。 这种存储成本与块本身无关。

实验评估 -- 恶意用户

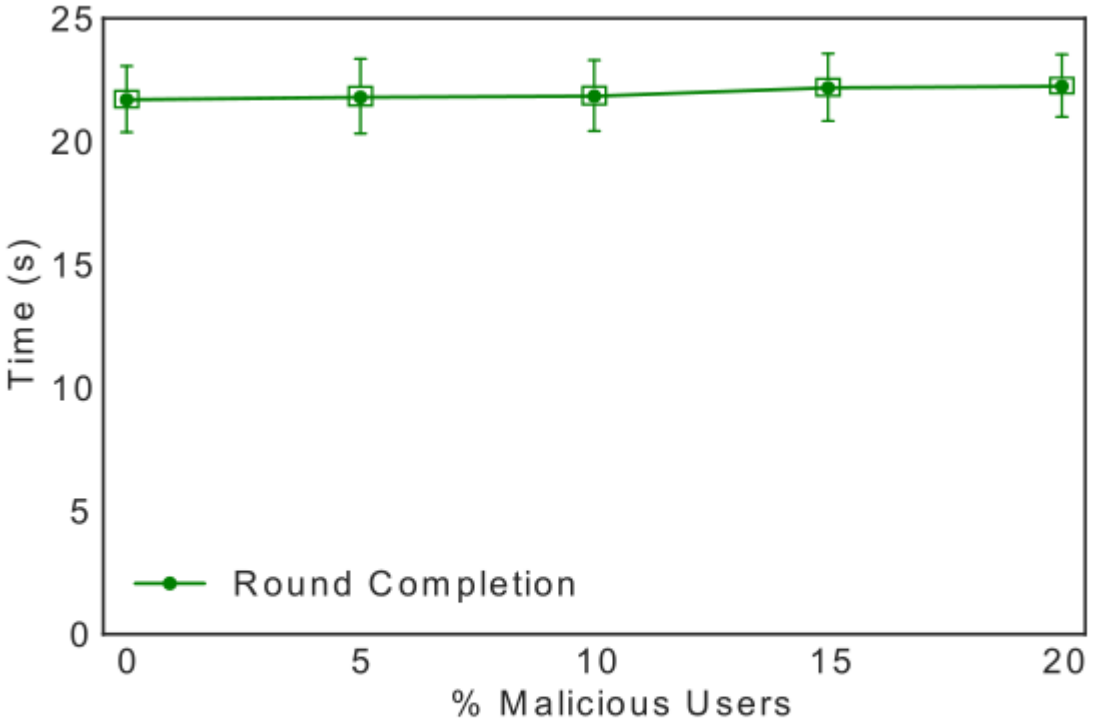
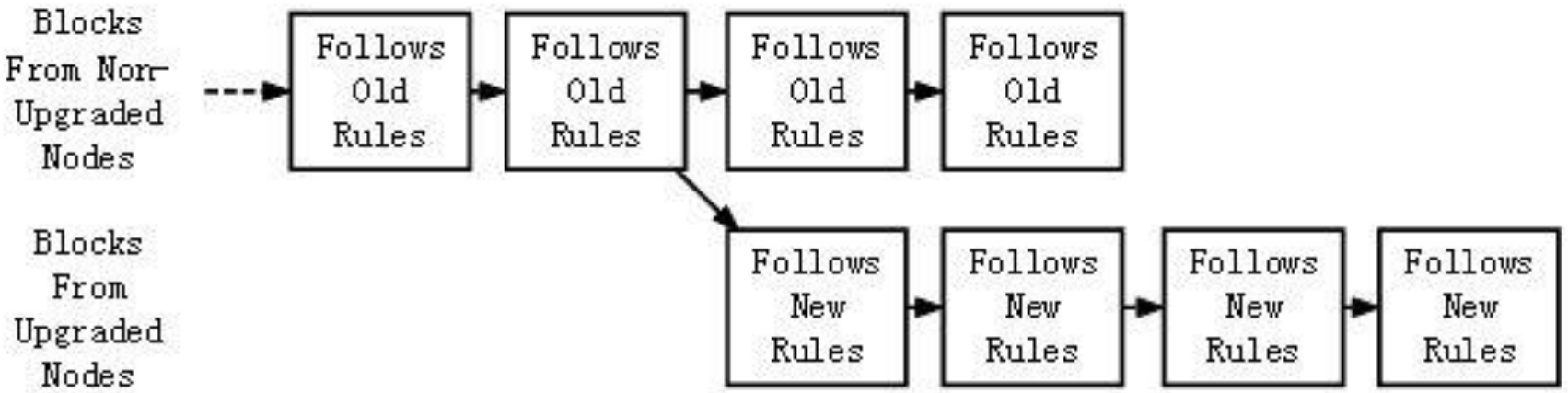


Figure 8: Latency for one round of Algorand with a varying fraction of malicious users, out of a total of 50,000 users.

Thank you!

硬分叉

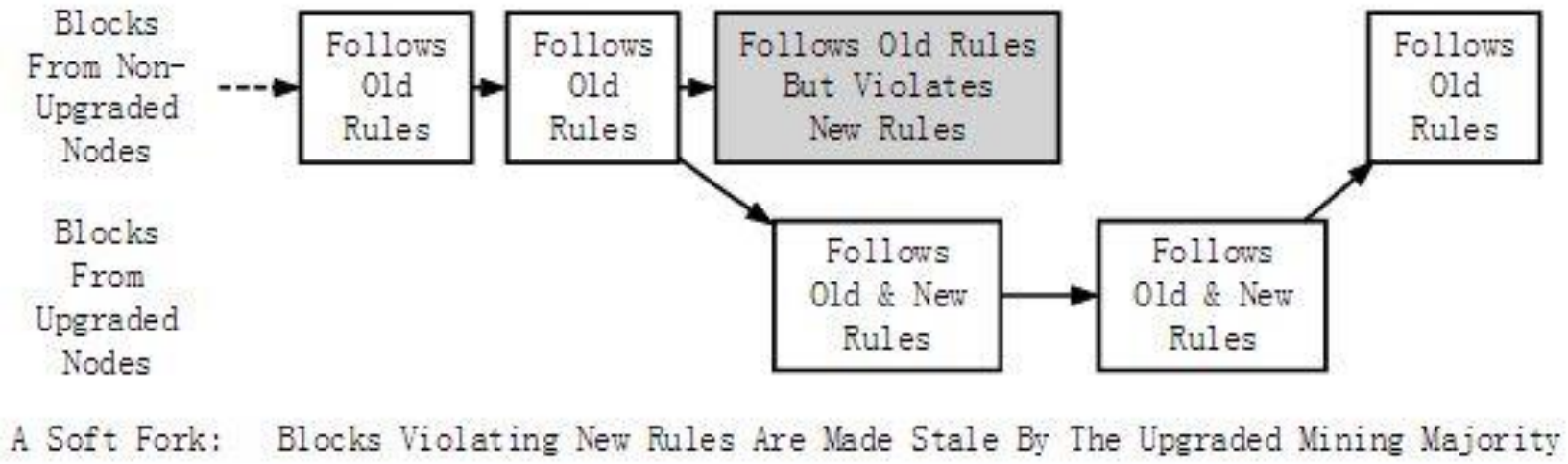
- 简单地说，硬分叉就是旧节点不接受新节点产生的区块，导致网络分裂为新链和旧链的分叉。



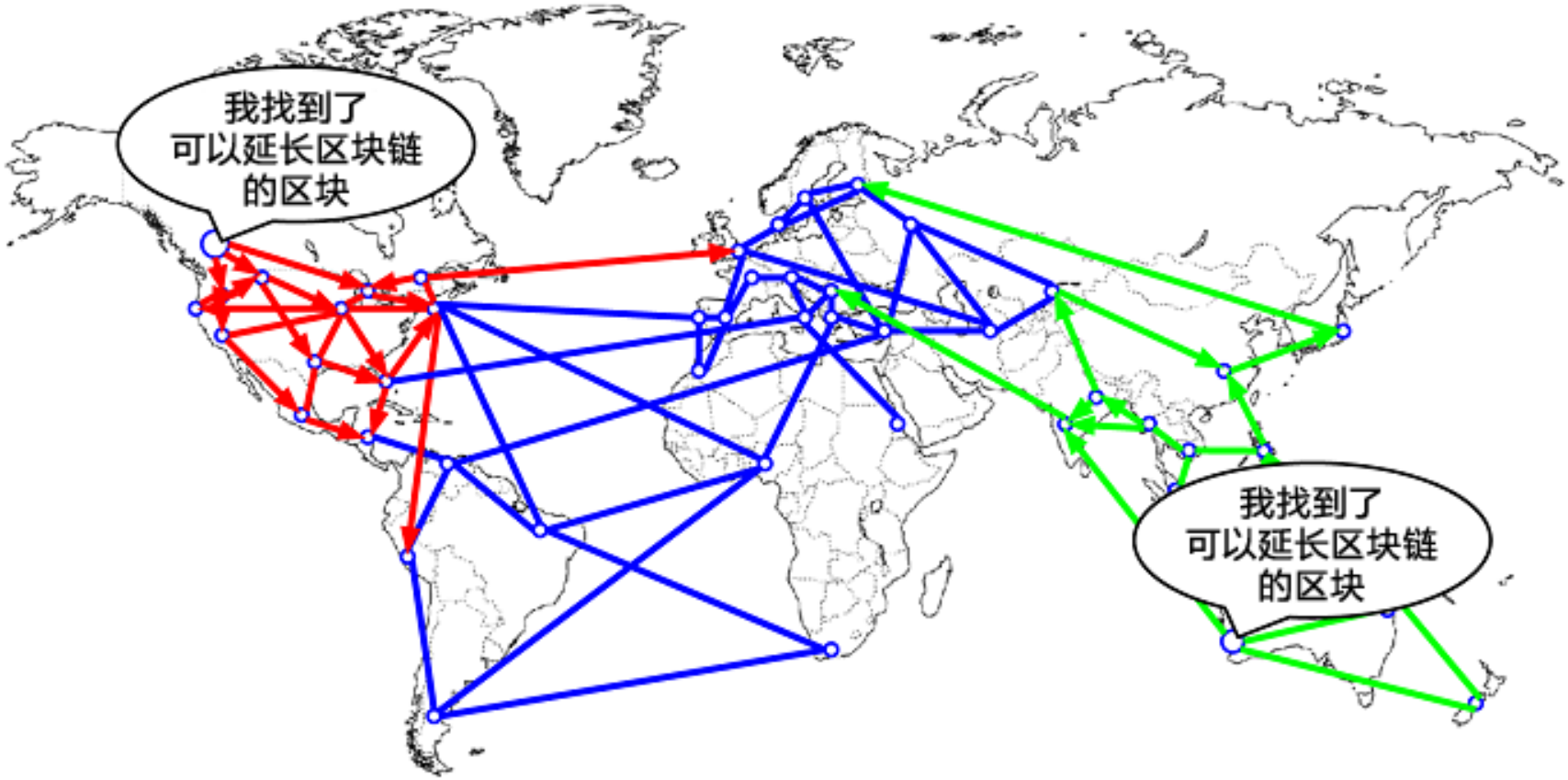
A Hard Fork: Non-Upgraded Nodes Reject The New Rules, Diverging The Chain

软分叉

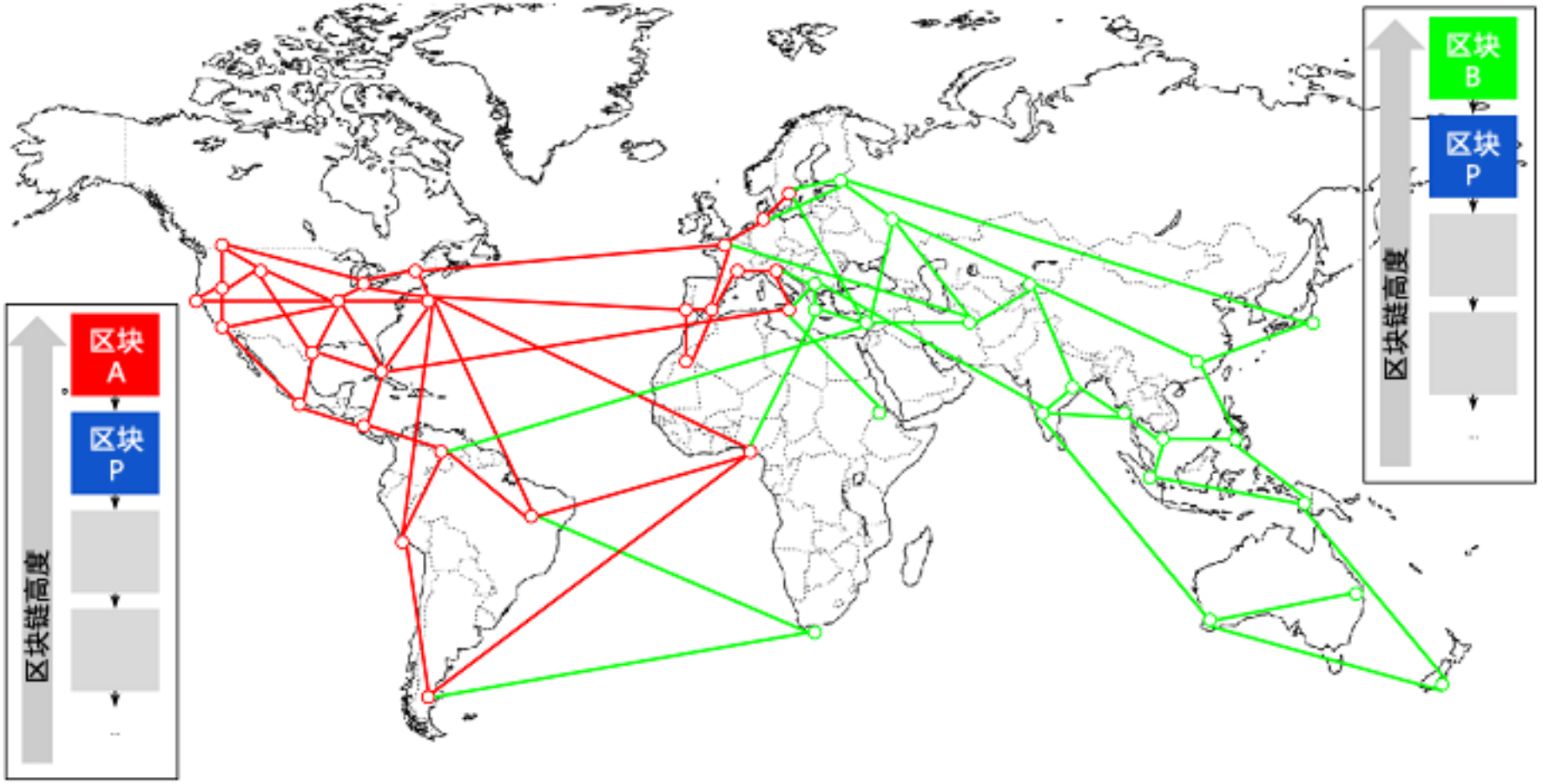
- 软分叉则是旧节点接受新节点产生的区块（虽然可能有某种潜在风险），如新节点算力占优，则分支博弈会导致网络最后归一于新链。



分叉



分叉



分叉