ABCP(AME Blockchain Consensus Protocol)

0.7

文档状态

0. Abstract		Y
1. Introduction		
2. Related Work		
3. Problem Definition and Assumption	2.1 Problem Definition	Y
	2.2 System Model and Assumptions	Y
	2.3 Adversary model	
	2.4 Key Features of ABCP	
4. ABCP Overview		Y
5. ABCP Details	5.1 Node’ s Local State	Y
	5.2 Potential Committee Selection	Y
	5.3 Final Committee Selection	Y
	5.4 Reach consensus	Y
	5.5 Security Analysis	Y
	5.6 Complexity Analysis	Y
6. Evaluation	6.1 Evaluation Setup	
	6.2 Real Testbed Performance	
7. Conclusion and Future Work		

1. Abstract

AME Blockchain 是一个分布式的 *partial synchronous message-passing system* **[3]**，网络中的每一个节点都可能遭遇各种故障甚至本身是一个 byzantine **[6]**节点。与此同时，网络中的某些节点也可能遭遇外部的恶意攻击，比如 DDos 和 Sybil attack 等。ABCP 是 AME 项目的共识协议模块，也是 AME 项目的核心组件之一。因此，AME Blockchain 的共识协议面临比较大技术和工程挑战，传统分布式系统共识不仅不考虑 byzantine failure，而且通常系统规模较小，工程实践中的 multi-paxos，raft 和 zab 协议，都是依赖相对稳定的 leader（primary）来就一系列 value 达成共识。但在区块链共识中，不能有稳定的 leader，因为这样很容易被针对性攻击或者腐化。ABCP 的设计将利用前沿的技术和领先的架构，并借助 AME AI 生态管控，权衡安全、性能和扩展性等多方需求，本文将概要介绍 ABCP 设计中的取舍权衡和基本流程描述。

和目前主流的其他共识协议相比，ABCP 在设计的过程中，我们不仅借鉴了非常启发式的 ***VRF*** 和 ***random beacon*** 的技术方法，而且我们对以下四个方面进行了深入的思考和创新，最终形成了 ABCP 独特的技术优势：

1) Early-stopping consensus

一直以来，区块链中共识协议的系统模型都是全网有 ***n*** 个节点，最多有 ***t*** 个节点可以同时故障。这样一个理论模型其实是一种比较悲观的假设。相反，ABCP 的设计中，借助于外围 AI 的管控，我们认为 AME blockchain 的网络不会有非常频繁的非计划性节点的 join 和 leave。因此，理想的 ABCP 比较深入地思考一个问题：在多达 t 个故障节点的时候，保证 ABCP 协议的正确性，在实际故障节点 f < t 的情况下，我们希望 ABCP 尽快达成共识，也就是 ABCP 共识算法要具备“early-stopping”特性。

2) Parallel byzantine agreement instances

传统 BFT 算法通常都依赖一个 leader 角色来驱动一次共识实例的运行，一旦 leader 节点本身是一个 byzantine 节点，byzantine leader 节点可能会丢弃、篡改或者延迟消息的发送，导致不能达成共识或者达成共识的效率非常低。ABCP 中采用针对同一个 block proposal，同时运行多个并行的 byzantine agreement instances 来规避 leader 节点可能是 byzantine 节点的问题。总体提升了达成共识的效率，因为最终 byzantine agreement instances 是在一个小范围内运行的，多个实例运行引入的开销是相对较低的。

3) Detect faulty nodes during consensus

ABCP 采用的技术方法是 gradecast，使用 gradecast 来检测 faulty nodes，一旦我们识别出某些 faulty nodes 之后，ABCP 在后续算法和流程中将忽略这些 nodes。通过这个设计，ABCP 秉承

transforms byzantine failures into benign failures 思想，进一步提升共识收敛的效率，同时也为 AI 管控模块提供了 node 运行行为分析的素材。

4) Prior or posterior strategies

区块链共识中，为了提升安全性通常都采用一些 posterior 的技术手段，例如 VRF；相反，为了提升效率和降低复杂度，又需要一些 prior 技术手段，例如公知的分组和 round-robin 规则。ABCP 结合 AME 生态来进行合理取舍，灵活结合 prior 和 posterior 的技术手段，确保整个生态的健康和可持续发展。

2. System Model and problem definition

System Model :

The system consists of *n* nodes, out of which up to $t < \frac{n}{3}$ may be Byzantine, i.e., behave arbitrarily and collude together. Denote by f ≤ t the actual number of faulty nodes in a given run. Communication is assumed to be partially synchronous and is done via message passing.

The Byzantine consensus problem consists of each node *p* having an input value v_p from a finite set V (i.e., $v_p \in V$). Each node *p* also has an output value $o_p \in V$. Two properties should hold:

1) ***Agreement***: $o_p = o_q$ for any two non-faulty nodes p,q (thus we can talk about the output value of the algorithm);

2) ***Validity***: if all non-faulty nodes start with the same input value v, then the output value of the algorithm is v.

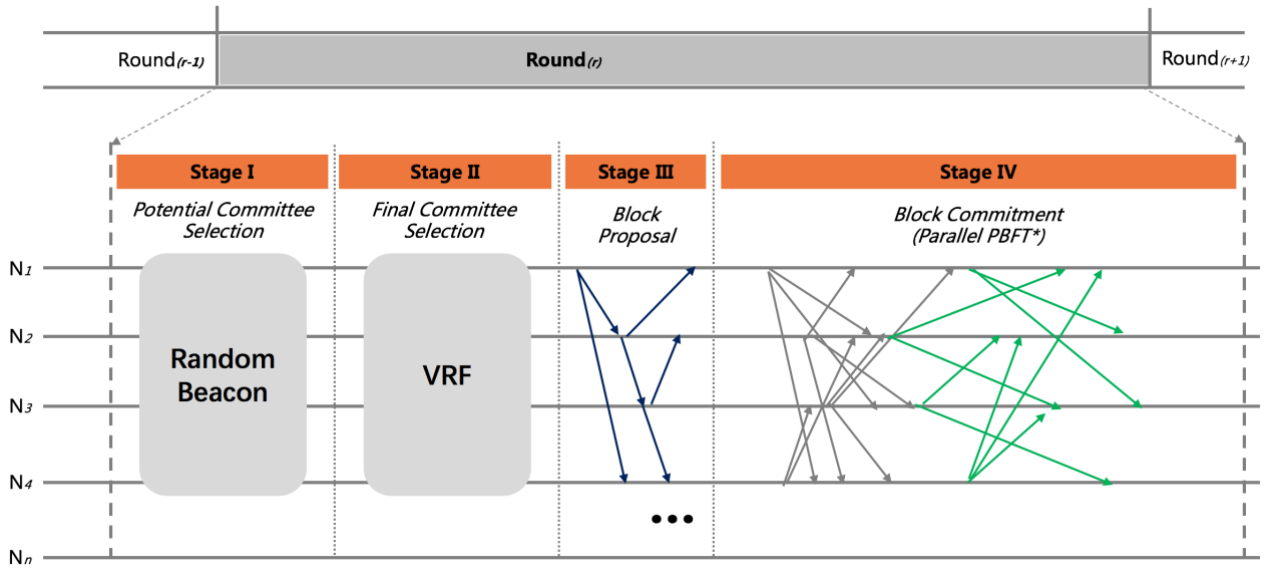
Byzantine Consensus:

Assume a system of n nodes, where each node n_i has a private value v_i, and the following must be achieved:

1) ***Agreement:*** All non-faulty nodes must agree on the same value $v \in \{ v_1, v_2, \dots, v_n \}$.
2) ***Validity:*** If all non-faulty nodes have the same initial value v, then the agreed upon value by all non-faulty nodes is v.
3) ***Termination:*** All non-faulty nodes must eventually decide on a value.

3. ABCP Overview

本节概要介绍 ABCP，我们总体从 AME Blockchain 的参与者和 ABCP 执行过程的两个角度来描述。宏观上，ABCP 的参与者是 AME Blockchain 网络中的 Full Node，在达成共识的过程中，同一个节点可能扮演不同的逻辑角色，比如：Potential Committee（***PC***），Final Committee（***FC***）等。ABCP 的一个完整生命周期包含 4 个 Stages，ABCP 设计的概念和总体流程如下图所示：



图一：**ABCP** 总体示意图

Node: 每一个 Node 代表 AME Blockchain 中的一个 Full Node（N_1，N_2, ..., N_n），知道网络中所有其他 Node 的 public key。接收 client 提交的 transaction，与其他 Nodes 通过 peer-to-peer 的协议进行通讯。诚实的 Node 完全按照 ABCP 的协议运行且遵循 AME 生态的管理策略，byzantine Node 可能违背协议和管理策略，但是一旦被 AME 生态设别，将会收到惩罚。无论 Node 诚实与否，都将面临常规的 benign failure。

Rounds and Stages：ABCP 中把一轮共识过程抽象为一个 round，在同一个 round 中，要么产生一个新的 Block，要么未达成任何共识，产生一个空 Block。每一个 Node 根据本地最新的 Blocks 状态来识别当前的 round number。如图一所示，每一个 round 被划分为 4 分 stages，其中：

Stage I - 用于在整个 AME Blockchain 网络中选出 Potential Committee（***PC***），选择 ***PC***的主要目的是平衡性能和资源利用，避免在全网进行消息广播的开销，这一步也是 ABCP 实现 scalability 的关键举措。stage I 的主要技术方法是基于 Random Beacon，通过一个公用 hash 函

数H_{rb}，每一个 Node 在本地计算全网所有 Nodes 对应的 hash 值（H_{rb}函数的输出），按照预设的阈值N_{pc}选择一定数量的 Nodes 成为 ***PC***的成员。

Stage II - 的主要作用是防止 adaptive attack，因为在 stage I 中产生的 ***PC***成员是全网可知的，存在一定的被攻击风险（安全分析章节我们将结合 AME 生态的管理策略来评估这个潜在风险）。所以 Stage II 通过 Verifiable Random Function（***VRF***）**[33]**的技术方法，确保最终发起 block proposal（$\boldsymbol{B}_i^r$）的 Nodes 的隐蔽性，这样也就避免了针对$\boldsymbol{B}_i^r$ Nodes 的 adaptive attack。最终，在 stage II 中 ABCP 从 ***PC***中选出一个子集成为最终的 Final Committee（***FC***）。这一阶段，***FC***成员向 ***PC***成员广播：（1，$\boldsymbol{B}_i^r$，***sig***（$\boldsymbol{B}_i^r$），$\boldsymbol{\sigma}_i^r$）。

Stage III – 等待$\boldsymbol{\lambda}_{pc}$之后，开始 reduction 流程（详细描述见第 4.4 章节）：首先每一个 ***FC*** Nodes 成员，根据其接收到的$\boldsymbol{B}_{i\in\{0,\dots,fc-1\}}^r$。本地计算选择出一个$\boldsymbol{B}_i^r$ $leader$节点，记为$\boldsymbol{\ell}^r$。具体选择的规则如下：

a. 选择$\boldsymbol{B}_{i\in\{0,\dots,fc-1\}}^r$中包含的事务数目最多的$\boldsymbol{B}_i^r$；
b. 如果多个$\boldsymbol{B}_{i\in\{0,\dots,fc-1\}}^r$所包含的事务数目相同且为最多，则选择 ***H(***$\boldsymbol{\sigma}_{i\in\{0,\dots,fc-1\}}^r$***)***最小的$\boldsymbol{B}_i^r$；

$\boldsymbol{\ell}^r$ 构建消息：包含自己对应的（2，***sig***（$\boldsymbol{B}_{\ell^r}^r$），$\boldsymbol{\sigma}_{\ell^r}^r$）广播给所有的 ***PC***成员。***FC***对 ***FC***成员广播的消息进行相应的处理，等待$\boldsymbol{\lambda}_{fc}$之后，如果某个节点接收到（2，***sig***（$\boldsymbol{B}_{\ell^r}^r$），$\boldsymbol{\sigma}_{\ell^r}^r$）的属于同一个节点，且其计数大于等于$\frac{2.N_{fc}+1}{3}$，则继续广播（3，$\boldsymbol{sig}$（$\boldsymbol{B}_{\ell^r}^r$），$\boldsymbol{\sigma}_{\ell^r}^r$）。等待$\boldsymbol{\lambda}_{fc}$之后，如果某个节点接收到（3，***sig***（$\boldsymbol{B}_{\ell^r}^r$），$\boldsymbol{\sigma}_{\ell^r}^r$）的属于同一个节点，且其计数大于等$\frac{2.N_{fc}+1}{3}$，则Stage III 最终输出有效的 candidate block($\boldsymbol{B}_{cb}^r$)，否则 Stage III 最终输出一个空块$\boldsymbol{B}_{\in}^r$。Stage III 的输出我们统一记为：$\boldsymbol{B}_{pbft\_input}^r$。

Stage IV – ***FC***根据 Stage III 的输出：

对于 FC 中的每一个节点 i，如果 Stage III 输出为$\boldsymbol{B}_{cb}^r$，且 ***H(***$\boldsymbol{\sigma}_i^r$***)***属于 ***H(***$\boldsymbol{\sigma}_{i\in\{0,\dots,fc-1\}}^r$***)***中最小的$N_{\boldsymbol{fc_valid_leader}}$个成员，节点 i 作为 Leader 并行地运行改进后的$\boldsymbol{PBFT}^*$实例；FC 中的所有节点在参与$\boldsymbol{PBFT}^*$的过程中放行本地所见 ***H(***$\boldsymbol{\sigma}_x^r$***)*** 最小的$N_{\boldsymbol{fc_valid_leader}}$实例；

对于 FC 中的每一个节点 i，如果 Stage III 输出为$\boldsymbol{B}_{\in}^r$，且$\boldsymbol{H}(\boldsymbol{\sigma}_i^r)$属于$\boldsymbol{H}\left(\boldsymbol{\sigma}_{i\in\{0,\dots,fc-1\}}^r\right)$中最大的$N_{\boldsymbol{fc_empty_leader}}$个成员，节点 i 作为 Leader 并行地运行改进后的$\boldsymbol{PBFT}^*$实例；FC 中的所有节点在参与$\boldsymbol{PBFT}^*$的过程中放行本地所见 ***H(***$\boldsymbol{\sigma}_x^r$***)*** 最大的$N_{\boldsymbol{fc_empty_leader}}$实例；

FC 集合中的节点，如果任意时间收到某个$\boldsymbol{PBFT}^*$已经达成共识的广播，立即广播共识结论，这种情况 ABCP 称为 Final Consensus；如果一直未收到共识结论，可能是$\boldsymbol{PBFT}^*$未达成共识，也可能是当前节点处于网络分区的情况，等待一个比较长的 SBR 之后广播空块$\boldsymbol{B}_{\epsilon}^{r}$，这种情况 ABCP 称为 Tentative Consensus。（具体描述参见 4.4 章节）

4. ABCP Details

在解决 ABCP 详细算法前，我们先把算法描述过程中使用的符号和含义罗列如下：

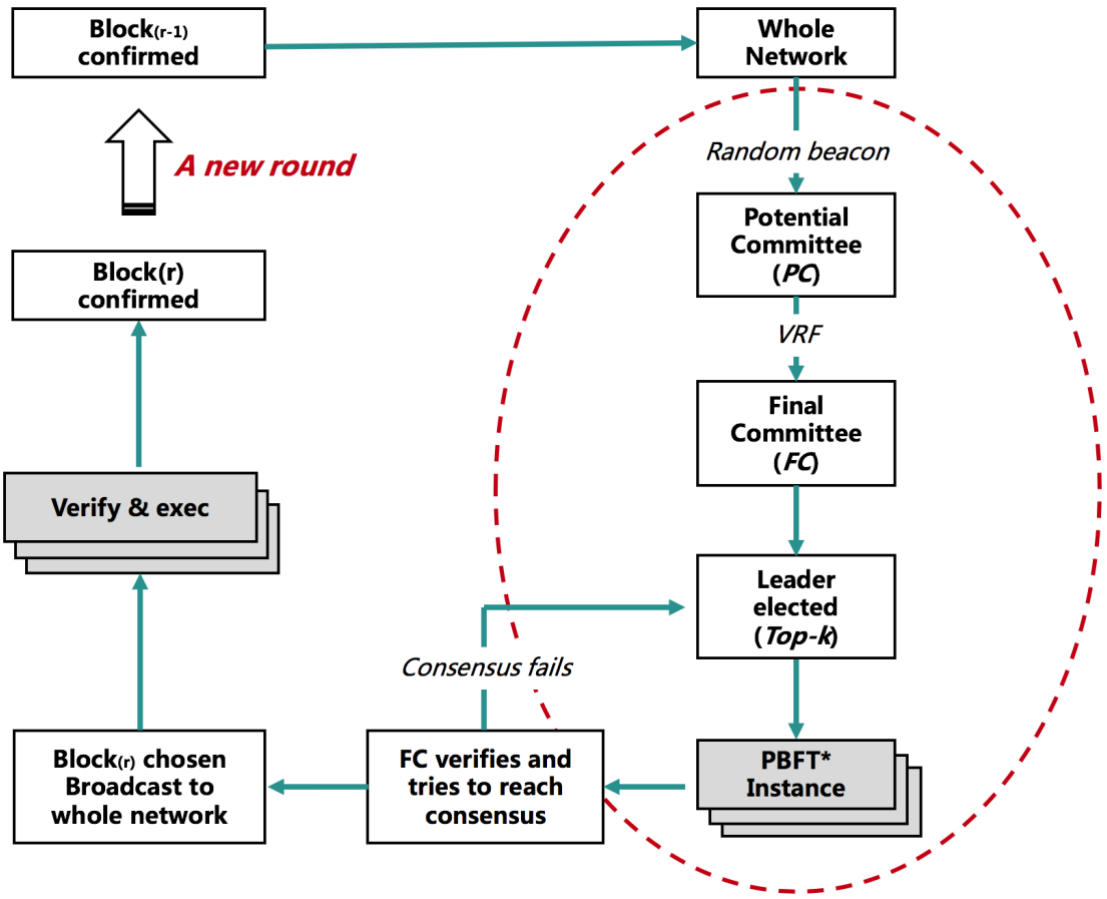
Parameter	Meaning	Value
N_{all}	AME blockchain 全网的节点数目	设计目标: 10w
N_{pc}	被选中为 ***PC***的节点数目	512
N_{fc}	被选中为 ***FC***的节点数目	16
$N_{fc\_valid\_leader}$	共识阶段输入有效区块并行运行 ***PBFT****实例的数目	3
$N_{fc\_empty\_leader}$	共识阶段输入空块并行运行 ***PBFT****实例的数目	2
$\boldsymbol{PK_i}$	节点 i 的 public key	
$\boldsymbol{SK_i}$	节点 i 的 secret key	
$\boldsymbol{SE_i^r}$	节点 i 的 secret string，周期性变化	
$\boldsymbol{RS_r}$	第 r 轮的 random seed	
$\boldsymbol{rep_i^r}$	节点 i 在第 r 轮的 reputation	
$\boldsymbol{pw_i^r}$	节点 i 在第 r 轮的权重	
RI	Secret string refresh interval (# of rounds)	10
$\boldsymbol{\tau_{pc}}$	Expected # of potential committee	
$\boldsymbol{\tau_{fc}}$	Expected # of final committee	
$\boldsymbol{\sigma_i^r}$	节点 i 在第 r 轮的 credential	
$\boldsymbol{\lambda_{pc}}$	Timeout to broadcast to PC committee	500ms
$\boldsymbol{\lambda_{fc}}$	Timeout to broadcast to FC committee	200ms
$\boldsymbol{\lambda_{all}}$	Timeout to broadcast to whole network	3000ms

Parameter	Meaning	Value
B_i^r	节点 i 在第 r 轮的 block proposal	
B_{cb}^r	第 r 轮的 candidate block	
$B_{\in}^r$	第 r 轮的空块	
$B_{pbft\_input}^r$	第 r 轮 pbft 算法的输入 value	
ℓ^r	第 r 轮的 leader	
sig	数字签名	
H	某个 hash 函数	

在对 ABCP 进行详细介绍之前，我们首先说明 AME Blockchain 初始设置的情况。在初始阶段我们通过一种 key generation protocol 来产生所有 Nodes 会使用到的必要信息，我们为每一个 Node 产生三种必要信息：

- PK_i： public key，所有 AME Blockchain 中的 Node 都知晓其他 Node 的 PK
- SK_i： secret key，每一个 Node 私有，用于对消息签名和验证
- SE_i^r： secret string，每一个 Node 私有，且周期性变化，一个固定长度的随机字符串

另外，在 Genesis Block 中，我们需要通过 *VSS Coin Tossing scheme* 产生一个随机种子 Random Seed（ RS_r ）。$Round\ r$ 的 Potential Committee 选择， RS_{r-1}是其中一个影响因素之一，每一个 B_i^r中随机选 t 个节点的 $SE_j^r(j \in \{0, 1, \dots, t-1\})$来作运算以生成当前$B_i^r$的 RS_r。



图二：**ABCP** 流程示意图

如上图所示，每一个 Block 的最终确认，意味中本$\boldsymbol{Round}$的结束，同时也意味着下一个$\boldsymbol{Round}$的开启。在 AME blockchain 中，在同一个$\boldsymbol{Round}$中我们假设参与者不变，这个过程中的 node leave 我们认定其为 failure node。在$\boldsymbol{Round}_{r-1}$和$\boldsymbol{Round}_{r}$之间，根据 AME AI 和生态管理策略来处理 node 的 join 和 leave。一个$\boldsymbol{Round}$从全网选择 ***PC*** 开始，经历选择 ***FC***、确定$\boldsymbol{PBFT}^{*}$ leader、并行运行$\boldsymbol{PBFT}^{*}$、confirm ***BP***、广播 ***BP***，以当前$\boldsymbol{Round}$确认结束。我们使用 timeout 来抽象同步通讯系统中的同步上限，称之为 sync barriers，简称 ***SBR***。如果在等待并行 **PBFT*** 的 ***SBR*** 比较小，可以选择在第一次 ***SBR*** 后未达成共识再重试一次。

选择一次比较大的 ***SBR*** 还是多次间隔较小的 ***SBR***，是运行时可以根据系统的状况灵活决定的，通常可以以一个比较大的 ***SBR*** 间隔开始尝试，直到找到系统平稳运行的最佳值。虽然选择 ***SBR*** 的间隔大小和 ABCP 的正确性无关，但是会对整个系统的的性能带来一些影响，一个较大的 ***SBR*** 值会导致一$\boldsymbol{Round}$的 ABCP 开销变长，缺乏实践意义；相反，一个较小的 ***SBR*** 值可能被把一些正确但仅仅是慢的节点误认为是故障节点，甚至导致一$\boldsymbol{Round}$形成空块。

4.1 *Node*'s Local State

每一个 *Node* 本地维护PK_i和SK_i，PK_i是全网公知的。 SE_i^r是一个固定长度的随机序列串，每一个节点周期性地更新，最终被选中用于产生某个 *Block* 的 *Random Seed*(RS_i)。

4.2 Potential Committee（*PC*）Selection

AME Blockchain 中的节点，通过下面的方式计算所有节点的 *Potential Weight*(*PW*)：

$$PW_i^r \triangleq \frac{H(RS_{r-1}, r, PK_i)}{rep_i^r}$$

其中$i = \{0, 1, \ldots, n-1\}$，其中$RS_{r-1}$是上一个 *block* 的 *random seed*，r为本轮轮次，PK_i是每一个节点的 *public key*，rep_i^r是节点i 在截止$Round\ r-1$ 时，AME 生态赋予的 *reputation*，*H* 是预设的 hash 函数。根据计算的结果并结合全网的阈值τ_{pc}，我们确定 *Potential weight* 最小的 *m* 个节点为 *PC* 成员。

4.3 Final Committee（FC）Selection

Final Committee 的选择我们采用 *VRF* [33]技术方法，*VRF* 中主要涉及到四个函数：

- *VRFGen*：产生一对<*public key, secret key*>，这一步使用前面产生的<SK_i，PK_i>
- *VRFVal*：输入一个数值 *secret key* 和 *input*, 输出是一个随机数 *output*
- *VRFProof*：输入 *secret key* 和 *input*，输出是一个 *proof*，用于验证 *output* 是正确的
- *VRFVerify*：输入 *public key*，*input*，*output*，*proof*，验证 *proof* 的合法性

我们把 *VRFVal* 和 *VRFProof* 统一为一个步骤，称为 *VRFEvaluate*。以RS_{r-1}作为 *VRFEvaluate* 的输入（$input \triangleq RS_{r-1}$）。每一个节点本地计算输出$\sigma_i^r$，然后结合公开的阈值和二项分布，判定自己是否被选中为 *FC*。在广播B_i^r到 *PC* 的同时，包含自己对应的（1，B_i^r，*sig*（B_i^r），σ_i^r）。

在选定 FC 成员后，我们开始 Block proposal 选择的流程。等待λ_{pc}之后开始 reduction 流程，ABCP 的 reduction 过程基于[35]的理论基础，满足下面两个特性：

（相关的 assumption 和模型请参考[35]，这里我们只关注输出的含义）

a. If agreement is alter = true, there are correct processes with different initial values from V. In this case, all correct processes use a predefined default value from V as the result of the following steps.
b. If agreement is alert = false, then all correct processes have the same initial value from V. This value is the result of the following steps.

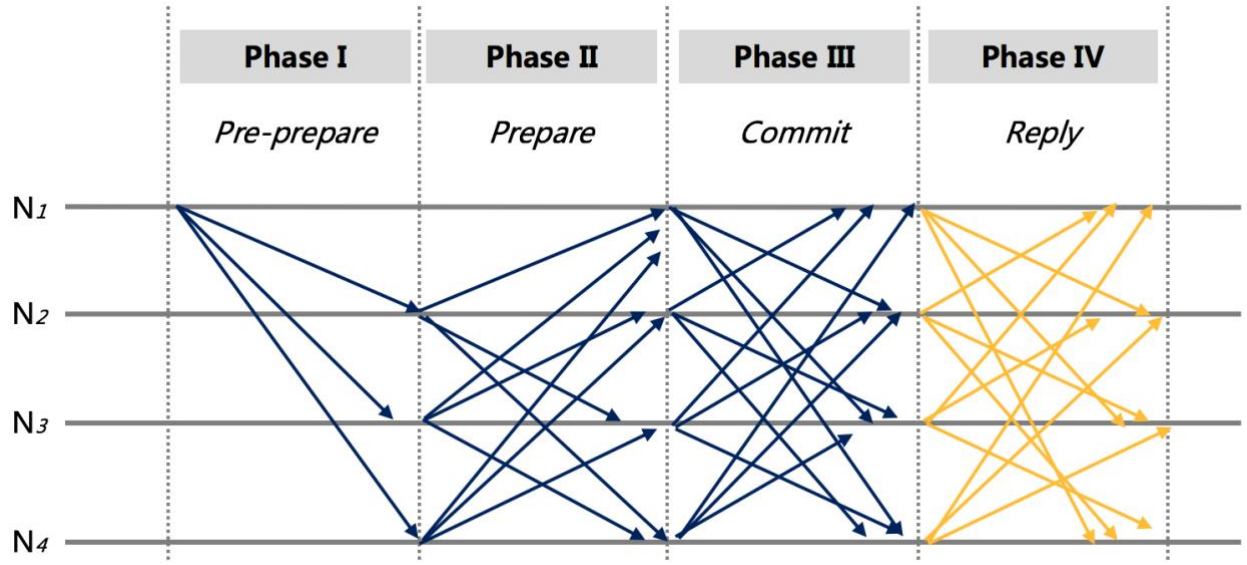
具体的流程如下，首先每一个 ***FC*** Nodes 成员，根据其接收到的$\boldsymbol{B}^r_{i\in\{0,\dots,fc-1\}}$。本地计算选择出一个$\boldsymbol{B}^r_i\ leader$节点，记为$\boldsymbol{\ell}^r$ 。具体选择的规则如下：

a. 选择$\boldsymbol{B}^r_{i\in\{0,\dots,fc-1\}}$中包含的事务数目最多的$\boldsymbol{B}^r_i$；
b. 如果多个$\boldsymbol{B}^r_{i\in\{0,\dots,fc-1\}}$所包含的事务数目相同且为最多，则选择 $\boldsymbol{H(\sigma}^r_{i\in\{0,\dots,fc-1\}})$最小的$\boldsymbol{B}^r_i$；

$\boldsymbol{\ell}^r$ 构建消息：包含自己对应的（2，$\boldsymbol{sig}$（$\boldsymbol{B}^r_{\ell r}$），$\boldsymbol{\sigma}^r_{\ell r}$）广播给所有的 ***PC*** 成员。***FC*** 对 ***FC*** 成员广播的消息进行相应的处理，等待$\boldsymbol{\lambda}_{fc}$之后，如果某个节点接收到（2，$\boldsymbol{sig}$（$\boldsymbol{B}^r_{\ell r}$），$\boldsymbol{\sigma}^r_{\ell r}$）的属于同一个节点，且其计数大于等于$\frac{2.N_{\boldsymbol{fc}}+\boldsymbol{1}}{3}$，则继续广播（3，$\boldsymbol{sig}$（$\boldsymbol{B}^r_{\ell r}$），$\boldsymbol{\sigma}^r_{\ell r}$）。等待$\boldsymbol{\lambda}_{fc}$之后，如果某个节点接收到（3，$\boldsymbol{sig}$（$\boldsymbol{B}^r_{\ell r}$），$\boldsymbol{\sigma}^r_{\ell r}$）的属于同一个节点，且其计数大于等于$\frac{2.N_{\boldsymbol{fc}}+\boldsymbol{1}}{3}$，这种情况对应于 reduction 协议中 alter 等于 false 的情况，则Block proposal 选择最终输出有效的 candidate block($\boldsymbol{B}^r_{cb}$)；否则 Block proposal 选择最终输出一个空块$\boldsymbol{B}^r_{\in}$。这种情况对应于 reduction 协议中 alter 等于 true 的情况。Block proposal 选择的输出我们统一记为：$\boldsymbol{B}^r_{pbft_input}$。

4.4 Reach consensus

在一个异步网络系统中达成共识是不可能的，具体可以参见**[1]**。工程实践中，分布式共识达成都是基于 ***partial synchronous*** 系统的实现，AME Blockchain 网络也是一个半同步消息传递系统，我们假定发送一个消息到另外一个节点的上限是 Δ，不同节点处理速度的上限是 Φ，虽然我们预先并不确定 Δ 和 Φ 是多少。当上一个步骤的选定的 ***FC*** 选定后，下一步就是在这些节点间就 ***BP*** 达成共识。



图三：改进的 **PBFT** 算法 **PBFT***

达成共识阶段，传统的 Byzantine agreement protocol **[2]**都是 leader-based，如果 leader 是一个 byzantine 节点，会严重影响达成共识的效率，甚至导致某些$\boldsymbol{Round}$中的共识流产；如果 byzantine leader 节点只是延迟消息的发送，外围系统非常难于甄别，因此对整个系统有比较大的危害**[15] [22] [24] [29] [32]** 。为了克服这些缺点，ABCP 采用两个主要的技术方法，协议需要 $3f+1$ 个 ***FC*** 以容忍 f 个 byzantine 节点，算法本身不需要 digital signature。下面简单介绍这两个技术方法：

PBFT* ： 传统 PBFT **[2]**算法主要分为四个 phases，分别为 Pre-prepare、Prepare、Commit 和 Reply。在 Reply 阶段 Replica(s)把满足 Commit 条件的结果回复给 client，client 作为 request 的发起者最终学习到对应的 request 是否在当前网络中达成共识。在 client-server 模型中这是一种非常普遍的做法，但是在 AME blockchain 网络中，每一个 Nodes 其实是定位平等的，每一个 Nodes 都需要尽早知道是否就下一个$\boldsymbol{B}^{r}_{pbft_input}$已经达成共识这一结论。因此，我们对 PBFT **[2]**算法的 Reply phase 进行了简单的改进，将之前的 reply to client 修改为在 ***FC*** 范围内广播。这样，

FC 成员中的 Nodes 可以尽早知道下一个$\boldsymbol{B}^{r}_{pbft_input}$的共识是否已经达成，确认$\boldsymbol{B}^{r}_{pbft_input}$达成共识后，***FC*** 成员就可以安全地进入下一步流程。

Parallel Instances：目前大多数的区块链共识算法中，都采用单 ***PBFT*** 实例来试图对下一个$\boldsymbol{B}^{r}_{i}$达成共识。如果遭遇 PBFT Leader 为 byzantine 节点，会严重影响达成共识的效率和网络传输的消息量**[39]**。在 Algorand **[29] [32]**中分析表明，在遭遇 byzantine Leader 的情况下，需要 11 次广播才能达成高可能性的共识结论。ABCP 中，我们通过在 ***FC*** 内选取多个节点，各自作为$\boldsymbol{PBFT}^{*}$Leader 运行独立的$\boldsymbol{PBFT}^{*}$实例，因为 ***FC*** 内部的选取条件的是公知的，虽然不同 ***FC*** 节点所见的选取条件可能会不同，但是这没关系，最坏的情况是导致并行运行$\boldsymbol{PBFT}^{*}$的实例数量不是很精确，不影响协议的正确性。下面详细描述其流程：

a. 在 Stage III 中，每一个 ***FC*** 节点的视角，可能得到的输出是$\boldsymbol{B}^{r}_{\ell r}$或者$\boldsymbol{B}^{r}_{\in}$；

b. 在 ***FC*** 的范围内，分别有多个$\boldsymbol{PBFT}^{*}$实例运行，对于以$\boldsymbol{B}^{r}_{\in}$为输入的$\boldsymbol{PBFT}^{*}$实例，如果本地的 alter 等于 false，则直接拒绝当前实例的运行；

c. 我们把以$\boldsymbol{B}^{r}_{\ell r}$为输入$\boldsymbol{PBFT}^{*}$实例集合称为α，以$\boldsymbol{B}^{r}_{\in}$为输入$\boldsymbol{PBFT}^{*}$实例集合称为 β，1 代表达成共识，0 代表未达成共识。最终的输出有以下三种情况：（1）$\alpha=1$，$\beta=0$；（2）$\alpha=0$，$\beta=1$；（3）$\alpha=0$，$\beta=0$；

   c.1 其中（1）和（2），成功的节点都会广播最终共识结论（在 AME 的正常网络状态，这个共识决定也会迅速散播到 ***FC*** 集合内部的所有正常节点）我们把这种情况称为 ABCP 的 final consensus，后续所有的交易都会基于 final consensus 的 block 进行。广播中通过 consensus type 来区分 final consensus 和 tentative consensus；

   c.2 如果是（3），因为达成共识的条件是 ***FC*** 集合内的法定人数而不是全部节点，所以每一个节点的视角其实无法区分当前是处于（1）、（2）、（3）的哪种状况。由于（1）和（2）的 early-stopping 特性，我们可以在本地未收到任何共识结论前等待一个较长的 ***SBR***。因为（3）的情况，一种原因是因为当前节点处于 network partition 中，另外一种原因是 ***FC*** 节点间无法就输入达成共识，这种情况下当前节点广播$\boldsymbol{B}^{r}_{\in}$作为最终的共识决定，称为 ABCP 的 tentative consensus。

d. Final consensus 本被接收到的节点立即 append 到 AME blockchain 中，这类共识的交易也因此被实时确认；round r 的 tentative consensus block 会被 hold 住，直到后续 round r+1 达成 final consensus 且其前驱 block 是 round r 的 tentative consensus block，才会连同 round r+1 的共识一起 append 到 AME blockchain 中。

e. ABCP 产生 final consensus 和 tentative consensus 两种类型的目的保持 ABCP 协议的 liveness，但在概率极低的 network partition 极端情况下，ABCP 理论上存在产生分叉的可能，反过来分叉又会对协议的 liveness 产生很大的影响。ABCP 的应对策略是：

   e.1 在 ABCP 的运行过程中，密切监控 block proposal 中附带的 previous block hash，如果和本地所见的 last block hash 不一致，则可能存在分叉，通过这种方式来发现可能的分叉。

   e.2 发现存在分叉后选择一些“分叉提案者”（P_{fps}），P_{fps}提议一个空块，空块的 previous hash 是自身所见的上一个 final consensus 区块（$\boldsymbol{B}^{x}_{last_final_block}$）的 hash 值，提案中还包含$\boldsymbol{B}^{x}_{last_final_block}$的高度。

   e.3 后续运行 ABCP 协议来就P_{fps}的提案达成共识，唯一不同的是选择P_{fps}的 leader block proposal 时，选择$\boldsymbol{B}^{x}_{last_final_block}$高度最高的提案，其它部分流程和普通 ABCP 协议流程一致。

在$\boldsymbol{PBFT}^{*}$执行的过程中，每一个 $\boldsymbol{FC}$ 节点在处理消息时都会带上自己的 signature。这不是一致性协议正确性的需要，是为了在 ABCP 达成共识后，$\boldsymbol{FC}$ 会把选定的 block 和相应的 signature 信息一并广播出去，非 $\boldsymbol{FC}$ 节点可以根据这些信息来验证选定 $\boldsymbol{Block\ Proposal}$ 的有效性。最终把选定的 $\boldsymbol{B}^{r}_{cb} \triangleq \boldsymbol{B}^{r}_{\ell^{r}}$或$\boldsymbol{B}^{r}_{\in}$记录到 AME blockchain 中。

ABCP 的这种并行共识有一个好处是一旦 $\boldsymbol{FC}$ 中某个成员接受到某个$\boldsymbol{PBFT}^{*}$实例已经达成共识，它可以放弃其他实例的运行而直接广播结论，不需要等待一个 $\boldsymbol{SBR}$ 超时。这是 ABCP 独特的 early-stopping 特性，在分布式系统中能够有效地提高共识收敛的效率，减少不必要的网络通讯。

The ABCP Protocol

Step 1: Potential Committee Selection (*PC*)

对于 ABCP 中的所有节点，通过下面的方式本地计算所有节点的权重(Potential Weight)：

$$PW_i^r \triangleq \frac{H(RS_{r-1}, r, PK_i)}{rep_i^r}$$

其中i = {0, 1, …, n-1}，其中RS_{r-1}是上一个 block 的 random seed，r为本轮轮次，PK_i是每一个节点的 public key，rep_i^r是节点i 在截止$Round$ r-1 时，AME 生态赋予的 reputation，H 是预设的 hash 函数。根据计算的结果并结合全网的阈值τ_{pc}，我们确定 Potential weight 最小的 m 个节点为 PC 成员。

- 如果本地判断属于 PC 成员，则继续 Step 2；
- 如果本地判断不属于 PC 成员，则立即结束；

Step 2: Final Committee Selection (*FC*)

对于每一个 PC 节点，以$\boldsymbol{RS_{r-1}}$作为 VRF 的输入（$input \triangleq \boldsymbol{RS_{r-1}}$）。每一个节点 i 本地计算输出$\boldsymbol{\sigma_i^r}$，然后结合公知的阈值和二项分布判定：

- 如果本地判断属于 FC 成员，广播（1，$\boldsymbol{B_i^r}$，sig（$\boldsymbol{B_i^r}$）到 PC，然后继续 Step 3；
- 如果本地判断不属于 FC 成员，则立即结束；

Step 3: Candidate Block Proposal Selection

3.1 每一个 FC Nodes 成员，等待$\boldsymbol{\lambda_{pc}}$之后，根据其接收到的$\boldsymbol{B}^r_{i\in\{0,\ldots,fc-1\}}$。本地计算选择出一个$\boldsymbol{B}^r_i$ $leader$节点，记为$\boldsymbol{\ell}^r$ 。具体选择的规则如下：

a. 选择$\boldsymbol{B}^r_{i\in\{0,\ldots,fc-1\}}$中包含的事务数目最多的$\boldsymbol{B}^r_i$；
b. 如果多个$\boldsymbol{B}^r_{i\in\{0,\ldots,fc-1\}}$所包含的事务数目相同且为最多，则选择 $H(\boldsymbol{\sigma}^r_{i\in\{0,\ldots,fc-1\}})$最小的$\boldsymbol{B}^r_i$；

$\boldsymbol{\ell}^r$ 构建消息：包含自己对应的（2，sig（$\boldsymbol{B}^r_{\ell^r}$），$\boldsymbol{\sigma}^r_{\ell^r}$）广播给所有的 PC 成员。

3.2 每一个 FC Nodes 成员，等待$\boldsymbol{\lambda_{fc}}$之后，如果某个节点接收到（2，sig（$\boldsymbol{B}^r_{\ell^r}$），$\boldsymbol{\sigma}^r_{\ell^r}$）的属于同一个节点，且其计数大于等于$\frac{2.N_{fc}+1}{3}$，则继续广播（3，$\boldsymbol{sig}$（$\boldsymbol{B}^r_{\ell^r}$），$\boldsymbol{\sigma}^r_{\ell^r}$）给所有的 PC 成员。

3.3 每一个 FC Nodes 成员，等待$\boldsymbol{\lambda_{fc}}$之后，如果某个节点接收到（3，sig（$\boldsymbol{B}^r_{\ell^r}$），$\boldsymbol{\sigma}^r_{\ell^r}$）的属于同一个节点，且其计数大于等于$\frac{2.N_{fc}+1}{3}$，则Block proposal 选择最终输出有效的 candidate block($\boldsymbol{B}^r_{cb}$)；否则 Block proposal 选择最终输出一个空块$\boldsymbol{B}^r_{\in}$。

Step 4: Reach Consensus

4.1 根据 Step 3 的输出：

- 对于 FC 中的每一个节点 i，如果 Step 3 输出为$\boldsymbol{B}_{cb}^{r}$，且 H($\boldsymbol{\sigma}_{i}^{r}$)属于 H($\boldsymbol{\sigma}_{i\in\{0,\ldots,fc-1\}}^{r}$)中最小的$N_{fc_valid_leader}$个成员，节点 i 作为 Leader 并行地运行改进后的$\boldsymbol{PBFT}^{*}$实例；FC 中的所有节点在参与$\boldsymbol{PBFT}^{*}$的过程中放行本地所见 H($\boldsymbol{\sigma}_{x}^{r}$) 最小的$N_{fc_valid_leader}$实例；

- 对于 FC 中的每一个节点 i，如果 Step 3 输出为$\boldsymbol{B}_{\in}^{r}$，且$\boldsymbol{H}(\boldsymbol{\sigma}_{i}^{r})$属于$\boldsymbol{H}\left(\boldsymbol{\sigma}_{i\in\{0,\ldots,fc-1\}}^{r}\right)$中最大的$N_{fc_empty_leader}$个成员，节点 i 作为 Leader 并行地运行改进后的$\boldsymbol{PBFT}^{*}$实例；FC 中的所有节点在参与$\boldsymbol{PBFT}^{*}$的过程中放行本地所见 H($\boldsymbol{\sigma}_{x}^{r}$) 最大的$N_{fc_empty_leader}$实例；

4.2 开始等待一个较大的 SBR：

- 如果在 SBR 内收到任意一个$\boldsymbol{PBFT}^{*}$实例的共识确认，立即广播其 final consensus 共识结论，区块为$\boldsymbol{B}_{cb}^{r}$或者$\boldsymbol{B}_{\in}^{r}$，取决于达成共识的$\boldsymbol{PBFT}^{*}$实例；

- 如果在 SBR 内未收到$\boldsymbol{PBFT}^{*}$实例的共识确认；

  Case1: 当前节点处于网络分区中，全网可能已经达成共识也可能没有，无从确认；因此广播 tentative consensus，区块为$\boldsymbol{B}_{\in}^{r}$；

  Case2: 全网因为敌对攻击或者任何其它原因没有达成共识；因此广播 tentative consensus，区块为$\boldsymbol{B}_{\in}^{r}$；

4.3 每一个接收节点的视角：

- 如果接收到 Final consensus，立即 append 到 AME blockchain 中；

- 如果接收到 tentative consensus，hold 住当前 round r 的 tentative consensus block，直到后续 round r+1 达成 final consensus 且其前驱 block 是 round r 的 tentative consensus block，连同 round r+1 的共识一起 append 到 AME blockchain 中；

Step R: Recovery Protocol

ABCP 产生 final consensus 和 tentative consensus 两种类型的目的保持 ABCP 协议的 liveness，但在概率极低的 network partition 极端情况下，ABCP 理论上存在产生分叉的可能，反过来分叉又会对协议的 liveness 产生很大的影响。ABCP 的应对策略是：

- 在 ABCP 的运行过程中，密切监控 block proposal 中附带的 previous block hash，如果和本地所见的 last block hash 不一致，则可能存在分叉，通过这种方式来发现可能的分叉；
- 发现存在分叉后，选择一些“分叉提案者”（P_{fps}），P_{fps}提议一个空块，空块的 previous hash 是自身所见的上一个 final consensus 区块（$\boldsymbol{B}^{x}_{last_final_block}$）的 hash 值，提案中还包含$\boldsymbol{B}^{x}_{last_final_block}$的高度；
- 后续运行 ABCP 协议来就P_{fps}的提案达成共识，唯一不同的是选择P_{fps}的 leader block proposal 时，选择$\boldsymbol{B}^{x}_{last_final_block}$高度最高的提案，其它流程和普通 ABCP 协议流程一致；

4.5 Security Analysis

Safety goal. 在假设存在少于 1/3 节点为 Byzantine 节点的情况下，所有诚实节点对交易达成一致性. 更具体的，当一笔交易 A 被一个诚实节点接受之后，未来任意被诚实节点接受的新交易都基于交易 A 之上；即在任意节点的交易 log 里，都应该含有交易 A。

Liveness goal. 在弱同步网络假设下，系统在潜在 Byzantine 恶意攻击环境下，能不断产生新的区块。

ABCP 通过选举出部分节点组成的 ***PC***和 ***FC***来运行 parallel PBFT*算法，现在 committee 中产生共识后广播至整个网络。所以当 ***PC***和 ***FC***的选举能够正常产生以及随机选择的 ***PC***和 ***FC***中的恶意节点不超过 1/3 阈值条件下 safety 和 liveness 的性质可有 PBFT*的协议保证。而 ***PC***是基于阈值签名的 decentralized random beacon 实现，任意节点都无法操控和预知 random seed 的生成，所以无法操控和预知 ***PC***和 ***FC***的生成。在 random seed 产生后所有节点都可以知道 ***PC***的成员身份，以及在 ***FC***主动广播自己的身份后，***PC***中的成员将得知 ***FC***各个成员的身份。因此存在潜在的对 ***PC***和 ***FC***成员进行攻击的行为。下面我们将分析各种攻击行为类型以及攻击行为的 economic incentive 来论证在 AME 网络中无法对 ***PC***中的成员进行大规模攻击。

Types of Attack

Nothing at stake:

Nothing at stake攻击通常指validator由于没有经济利益刺激，而在投票中投出不同的块，使得区块产生过程流产等。为了应对这一攻击，ABCP共识的解决方案是在正常出块情况下，给作为PC和FC成员的validators提供经济激励。

PC激励。PC为第1轮选举中被选出的委员会，其成员通过VRF计算进一步选举产生了FC，付出了算力和劳动，应该获得相应的经济激励。更重要的是，对PC进行激励可以保证其成员不恶意向全网广播FC成员信息，导致FC成员遭受恶意节点攻击。显然，FC进行PBFT共识之前，需要向PC广播其身份以便找到全部的FC成员。此时，PC便知晓了FC的身份。当1个非FC的PC知晓自己本轮不能进入FC，进而没机会获得“挖矿”奖励时，有可能会出现作恶或者消极怠工的行为，因为本轮打包的成功与否跟自身无利益关系。因此，我们提出如果本轮出块成功，下一轮才对本轮的PC进行奖励，这样就保证了每一轮的PC都希望本轮出块成功，从而避免了无经济利益的作恶行为。

PC 节点数相对较多，长期来看，每个节点承担 PC 工作的几率均等。因此，对 PC 的激励是大范围的空投行为，需要采用价值相对较低、且流转受限的系统内代币进行，以保证整个区块链生态不会过分通胀。
FC 激励。FC 为第 2 轮选举产生的最终委员会，其承担着打包、且通过 Leadless PBFT 算法产生共识的责任，该过程也就是传统意义上的“挖矿”。为了激励矿工诚实劳动，减少作恶行为，应该在出块并确认成功后，给予 FC 成员以经济激励。此时的经济激励采用更直接的、且能在交易所进行交易的代币进行发放。
如前文所述，FC成员可分为三个角色：
• Final Miner–成功打包者。
• Final Leader –PBFT中区块的提案者，在Leadless PBFT算法下，通常有N个。
• Final Verifier–PBFT中块的验证者，所有FC成员都是verifier。
根据三方在共识过程中的贡献，定义其收益比例为：Wabc-miner ：Wabc-leader：Wabc-verifier。在一轮奖励总量确定之后，将按照三方比例，分别发放各自的经济奖励。

Selfish mining

Selfish mining是指被选中的打包者优先或者只打包对自身有利的交易，从而更快的完成自身利益相关交易确认的行为。技术层面上来说，由于其打包的都是真实交易，验证节点无法拒绝确认此类的打包行为。另外，由于其区块较小，广播速度更快，更容易被其他节点确认。从这个角度来说，FC节点会倾向于selfish mining，以便优先完成自己利益相关的交易，减少等待时间，更大几率成为final miner。为此，我们将FC成员打包的交易量与其可作为FC所获得的收益关联起来，尝试使用经济手段解决这一问题。
假设AME区块链网络中1个块的容量上限为ClimitMB，Final Miner所提出包的大小为CfinalMB，那么其能获取的ABC奖励由如下公式决定：
Nabc-miner-non-selfish =Nabc-miner *(Climit -1 / (Cfinal + k)) / Climit
其中，k是常数，用来控制收益的收敛速度。
需要注意的是，为了刺激FC所有成员尽量把本地的全部交易打包入区块，需要将FC中其余两个角色（Final Leader和Final Verifier）可获得经济利益的期望均与最终区块的大小进行正向关联。

Sybil Attack：
Sybil Attack 通常通过单个节点拥有多个 identity 来操控网络。AME 网络中的节点将具有唯一的 identity，同时 AME 的 AI 管控模块亦要求各个节点上报对应的 identity，ip 等网络属性，这将使得 Sybil Attack 难以出现在 AME 网络中。

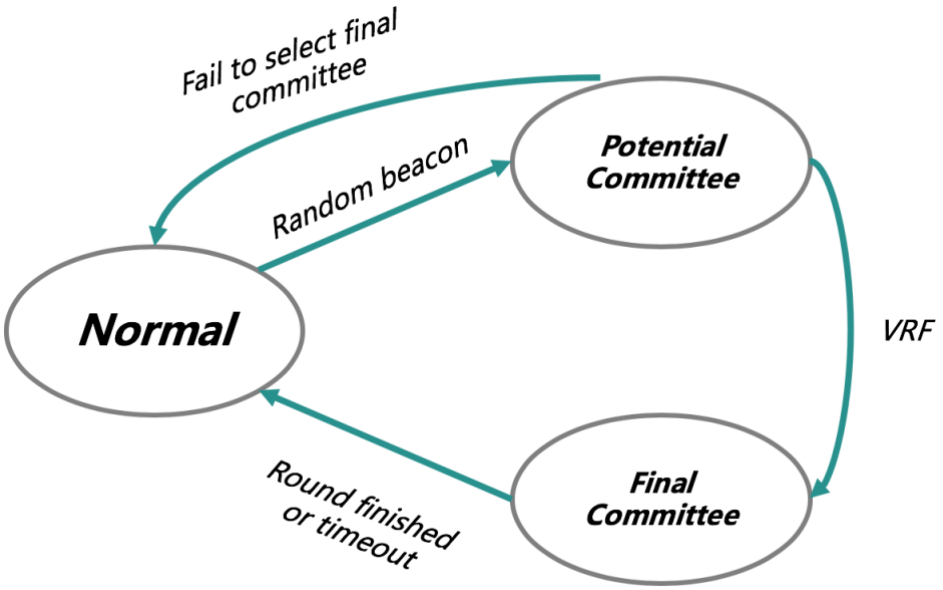
Eclipse Attack：

Eclipse Attack 通过控制单个节点的所有邻居节点，是的节点处于孤立状态，无法获得网络中最新的区块状态等。AME 中的 AI 管控模块要求各个节点上报当前网络节点之间的 interaction graph，并侦测各个节点的 interaction graph 是否异常；同时由于 ABNN 的底层网络结构使得；单个节点有大量的同组邻居节点和各个组对应的 counterpart 节点，因此 eclipse attack 不太可能出现在 AME 网络中。

Adaptive adversary：AME 的 AI 管控模块将根据节点的长期运行过程，调整节点 reputation，而一旦发现节点出现攻击行为将迅速降低节点 reputation 直至剔除网络；因此 AME 中的网络很难出现控制大量节点的恶意行为来达到有足够的恶意节点进入 ***PC***和 ***FC*** 以操控 Block 的生成或者使之长时间无法出块。

Incentive Analysis of Threat

在 ABCP 中，我们用下图来描述每一个 Node 处于的状态，整理流程是一个简单的状态机模型：



图四：**ABCP** 节点的状态转换示意

如上图，ABCP中的的Node处于Normal、Potential Committee和Final Committee三者其一，我们从ecomomic incentive角度分别分析每一种角色状态下节点涉及的安全性风险：

Normal

作为Normal节点存在攻击其他处于Potential Committee和Final Committee

节点的可能，以使得该轮区块流产，重新选择Potential Committee 和Final Committee，便于获利；由于单轮Potential Committee的选择范围在500个节点，而每一轮的Potential Committee都是随机选择，难以短时间内攻击大范围节点，而一旦有较大范围的攻击行为，AME的AI模块将会侦测并降低该节点的reputation直至剔除出网络；而由于Final Committee的选择是通过VRF进行秘密选举，处于Normal节点无法得知具体哪个节点是Final Committee节点，无法进行针对性攻击。

Potential Committee

从全网Normal状态到Potential Committee状态，我们通过全网Nodes总数n和一个阈值k来决定可以被选中为Potential Committee的概率，也就是说借助于AME AI模块的总体管控，我们可以选择最优的k（n通常是会变化的，在AME Blockchain中n不频繁变化）。同时被选中为Potential Committee的Node，基于其长期的运行行为，AME生态会提升其reputation，且若该轮正常成功产生区块可获得相应的ABIT报酬，所以Potential Committee成员没有主动发起攻击的动机。

Final Committee

从Potential Committee状态到Final Committee状态，由于我们采用用VRF的技术方法，每一个被VRF选中的Node只有自己知道自身被选中，其他Nodes是无从知道的，也无法发起任何攻击。一个被选定为Final Committee的Node，根据本地pending transactions提出block proposal，并且附带VRF的output、output_proof和signature，其他Node可以根据公共知晓的VRF函数、output和output_proof验证被选中为Final Committee的真实性。成为Final Committee将根据其担任的角色或多或少均获得AME生态的ABC奖励，所以Final Committee成员没有主动发起攻击的动机。

4.6 Complexity Analysis

本节我们分析 ABCP 的消息复杂度和消息延迟，我们假设某一$\boldsymbol{Round}$中：

N_{all}： AME blockchain 全网的节点数目
N_{pc}： 被选中为 ***PC***的节点数目
N_{fc}： 被选中为 ***FC***的节点数目
$N_{fc\_valid\_leader}$：共识阶段输入有效区块并行运行 ***PBFT****实例的数目
$N_{fc\_empty\_leader}$：共识阶段输入空块并行运行 ***PBFT****实例的数目

一次 ABCP 的共识过程中，在正常情况下，需要如下几次串行的消息传递：***FC***在确认自己被选中后广播 ***BP***以及相应的 proof；***FC***各个节点本地计算出优先级最小的 ***FC***成员，广播其 ***BP***的签名；***FC***各个节点计数本地接收到的 ***BP***签名，如果计数大于等于 2. N_{fc}/3，则继续广播 ***BP***的签名；FC 各个节点本地输出 ***GC***的结果，其中 grade 等于 2 的 ***FC***且其优先级在 ***FC***中前 3/8 的节点，作为 ***PBFT****的 leader 并行运行 ***PBFT****的 4 次消息传递；***BP***被确定后的全网广播 1 次。所以一次 ABCP 过程需要 8 个 message delay。总的消息数量为：

$$N_{pc}{}^{2}+2.N_{fc}{}^{2}+3.N_{fc}{}^{2}.N_{fc\_valid\_leader}+3.N_{fc}{}^{2}.N_{fc\_empty\_leader}+N_{fc}.N_{all}$$

注意：这里假定所有节点都正常，不考虑 p2p 网络传播过程的冗余消息和为了效率进行收敛而精简掉的消息，也不考虑终止转发无效或者 byzantine 消息等情况。

PBFT*有多个实例，每一个对应一个 leader。除非他们全是 fault nodes（无论 benign fault or byzantine fault），整个共识过程将会带来一次额外的等待 ***SBR***，这种情况最终以产生空块B_{ϵ}^{r}为结局；否则都可以在任意一个 ***PBFT**** instances 达成共识后广播$B_{pbft_input}^{r}$，这样最大化达成共识的效率。

5. 探讨与分析

Alogrand 和 Dfinity 是目前主流的两种共识协议，下面分别进行探讨与分析。

5.1 Algorand 共识协议

Algorand 共识协议提出并解决了三个问题：第一个是随机数发生器，第二个是随机出来的提案者如何在不泄露自己身份的情况下证明自己，第三个是如何应对不在线的节点。Algorand 采用了 PoS（Proof of Stake）共识机制的分组方式（包括提案组和验证组），以解决 PoW 共识机制所带来的高能耗问题，并希望同时提升交易性能。

Algorand 的独特创新是 VRF(可验证随机函数)和加密抽签，整体方案具有后验性。具体而言，用户是唯一知道自己是否成为某组（某轮的提案组或某步的验证组）成员的人，他人无法事先获知，只能等其广播后验证，恶意攻击者甚至无法事先知道组成员是谁，因此不能对他们进行贿赂或发起 DDos 攻击，组员间也无法共谋，提高了系统的安全性。但仔细分析其工作机制后，我们发现了下面的问题：

1) 签名数据庞大，造成存储浪费并影响性能。Algorand 使用 VRF 来确定提案组与验证组，这个方式充分发挥了 VRF 的可验证性优势，且后验优势使得 Algorand 的共识体系更安全。但是，Algorand 进入验证阶段，采用的是一种可扩展的拜占庭容错算法，即 BA*算法，参与节点通过 VRF 秘密抽签选出。这一设计使 Algorand 在验证前必须等待凭证（VRF prove）到来，才能知晓参与节点。而且，由于使用了可扩展的拜占庭容错算法，使得 Algorand 的验证组规模必须比较大(2000～4000 人)，这将导致签名数据异常庞大。以 2000 个验证者为例每产生一个区块的验证消息约 300K，不仅造成存储浪费，而且更影响性能。
2) 算法对于网络带宽要求极大，个人用户很难参与，严重影响去中心化特性。如下图所示，对照来自于 Algorand 论文中的公开测试数据，在实验环境中，Algorand 需要让区块达到 10M 大小，才能达到 125 倍比特币的性能。10M 区块大小意味着要求节点的网络带宽峰值至少需要 80M 才能承载，这对目前一般用户非常困难，影响了系统的效率和去中心化特性。
3) 节点网络规模受限，网络通讯不经济并影响性能。当网络规模太小时，将无法组成所需的提案组和验证组，影响系统正常工作。而网络规模太大时，提案块（假设大小 2M）和验证信息（大小约 300K）无法快速传播到后继验证节点，也会影响系统性能。Algorand 即使在验证环境也需要等待 5*10=50s 以上（包括等待潜在提案者的时间延迟及其特殊的拜占庭算法），加上其提案传播时间 10 秒，至少需要 60 秒才能完成一次

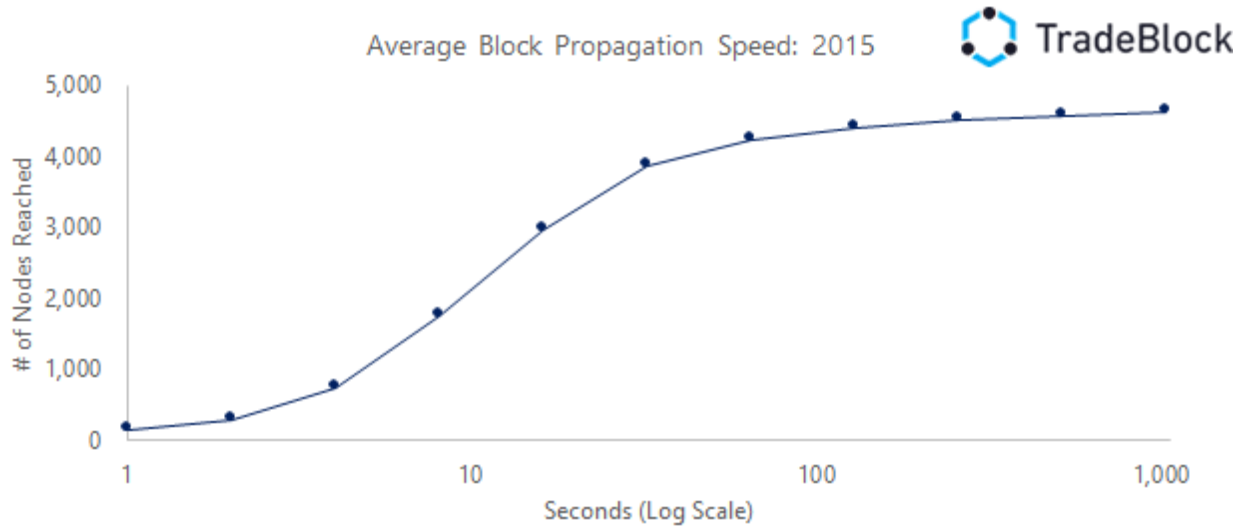
出块上链。假设一个有效存储 1M（去除签名后）的块包含 1 万笔交易，则其 TPS 为 10000/60=167。我们预估在全网真实环境下，这个结果会更糟，离项目方宣称的 1000TPS 差之甚远。

4) 账本数据存储相对较大。如前所述，因其采用多步 BA 共识算法，其签名占据巨大存储空间，虽然提高了安全性，但影响了性能，也浪费了账本存储空间，制约了账本系统的存储容量和系统规模。

我们认为 Algorand 的 VRF 和加密抽签后验性给区块链共识协议的设计提供了很好的启发思想，但其在验证环节的设计更偏单纯的学术化理想化，导致其对网络流量、高效数据通讯等实际工程落地问题思考不够；严重影响了公链运行性能、节点网络规模、账本存储容量和去中心化程度。

5.2 Dfinity 共识协议

另外一个颇受关注的共识协议是 Dfinity，为了解决 PoW 性能低下和高能耗的问题，Dfinity 也采用了 PoS 分组模式。Dfinity 用随机数把矿工分为若干个组，每组 400 人。每次由随机数决定下一出块组，组内使用 BLS 门限签名来达成组内共识。BLS 门限签名要比 Algorand 的可扩展的 BA*算法更高效，无需像 BA*算法那样多轮的验证步骤，不但提高了 TPS 而且还省下了长签名数据入链造成的存储容量问题。其理论上整个出块上限是 10 秒，每隔 2 秒一个提案，且在其 testnet 上
的 TPS 是 2500。Dfinity 在性能方面比 Algorand 更具优势，但是其网络通讯效率不高，像比特币和 Algorand 一样，完全靠 P2P 的传播，没有针对分组模式进行设计。容量虽不像 Algorand 那样浪费，但最大容量仍受限于 full node 的存储空间。



图五：节点规模和传输时延统计参考

1) 不具有后验性，安全性较差。虽然有多个组，但各组的成员事先固定，容易被攻击，也容易组内窜谋，影响安全性。Dfinity 也没有考虑同组的提案人和验证者联手共谋的可能，存在被人为控制的隐患。
2) 网络通讯效率低下。400 个成员的组内通信也是依靠 P2P 的 gossip 协议，通讯量巨大且效率低下，影响出块速度和系统性能。假设有 10 个组，每组 400 个节点，则系统有 4000 个节点，按 TradeBlock 上图所示，1M 的块需要约 18 秒传播到 3000 个节点，远超其出块 timeout 时间，系统无法正常工作，影响系统的稳定性、性能和规模。
3) 存储容量受限于单节点的存储能力。与比特币、以太坊和 Algorand 一样，其系统容量，受限于单节点的存储空间，没有涉及分布式存储技术的应用和优化。

Dfinity 在性能方面因其分组共识和 BLS 组签名有所提高，但其安全性有所下降。其 400 人分组规模的通讯效率也存在一定问题，我们认为其技术架构和宣称的去中心化云服务仍有较大的差距。

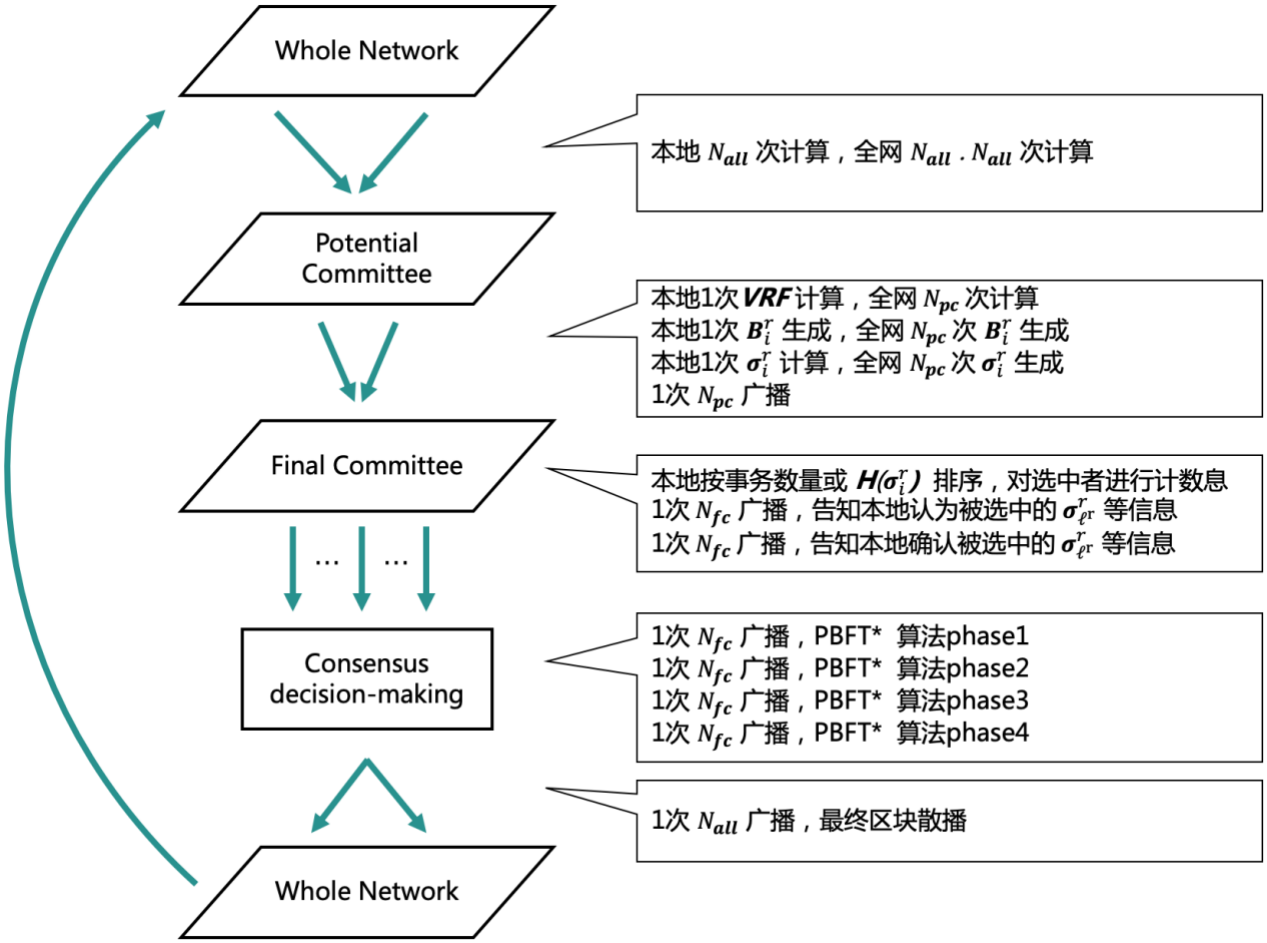
5.3 总结与探讨

综上所述，我们从技术层面分析认为：Algorand 不仅可能无法达到其宣称的高性能，而且由于其带来的巨大存储和带宽开销，提高了节点加入网络的门槛，其去中心化特征也可能存在问题。Dfinity 在安全性和性能上都存在比较多的疑点。同时，无论 Algorand 还是 Dfinity 目前都还没有工程落地，大型分布式系统，尤其是去中心化的大型分布式系统，其理论基础和工程实践之间存在着巨大的鸿沟，实践中的效果如何有待事实来证明。最后，这两个项目目前都还没有 token 激励机制，在一定程度上会影响参与者的积极性，从而影响其整体生态的发展和实用价值。

ABCP 在设计上从理论层面我们兼顾系统的安全性、扩展性和资源消耗，避免走纯粹理论学术化的极端，从系统层面思考出发，确保我们的设计具有非常健壮的工程落地基础。同时，ABCP 结合整个 AME 生态的经济模型和激励机制，具备非常完备的理论、实践和实用的价值！

6. ABCP 性能评估

ABCP 中性能消耗主要在网络传输上，其次是本地计算，ABCP 中涉及计算和广播的流程如下所示：



图六：ABCP 的主要性能消耗

我们估算本地一次全网 hash 值计算耗时 1000ms，一次N_{pc}(512)规模的广播耗时为 500ms，一次N_{fc}(16)规模的广播耗时为 200ms，一次N_{all}(10 万)规模的广播耗时为 3000ms。那么 ABCP 的一次出块保守估算为：

$$1000 + 500 + 6*200 + 3000 = 5700ms = 5.7s$$

如上估算假设 ABCP 节点全球分布，节点间最远距离为 RTT=400ms，那么单次广播耗时 200ms。实际工程中，前期节点规模不会一步到 10 万，加上节点分布一般会好于 RTT=400ms。ABCP 的出块时间理想估算在 3s 左右。下表假设把 ABCP 移植到 bitcoin 和 ethereum 中，估算 AME blockchain 的吞吐量和 latency：

	区块大小	出块时间 3s （tps）	出块时间 5.7s （tps）
bitcoin	4M	1028	541
	8M	2056	1082
ethereum	4M	2608.4	1372.8
	8M	5216.8	2745.6

注意：其中 bitcoin 的事务大小按 680 字节计算，ethereum 的事务大小按 268 字节计算，出块时间即为事务确认 latency。

REFERENCES

[1] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," in *ACM SIGACT-SIGMOD*, 1983.

[2] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398– 461, 2002.

[3] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[4] L. Lamport, "Paxos made simple," *SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.

[5] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference*, 2014.

[6] L. Lamport, R. E. Shostak, and M. C. Pease, "The Byzantine generals problem," *ACM*, vol. 4, no. 3, pp. 382–401, 1982.

[7] A. Boldyreva, "Threshold signatures, multi-signatures and blind signatures based on the Gap-Diffie-Hellman-group signature scheme," in *Workshop on Theory and Practice in Public Key Cryptography (PKC)*,
2003.

[8] Y. Amir, B. A. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE Trans. Dependable Sec. Comput.*, vol. 8, no. 4, pp. 564–577, 2011.

[9] A. Shoker and J.-P. Bahsoun, "BFT for three decades, yet not enough!" 2009. [Online]. Available: http://haslab.uminho.pt/ashoker/files/ shokerbft3decadestr.pdf

[10] C. Ho, "Reducing costs of Byzantine fault tolerant distributed applications," 2011-08-31. [Online]. Available: http://ecommons.library.cornell.edu/ handle/1813/30754

[11] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, Dec. 1990. doi: 10.1145/98163.98167. [Online]. Available: http://doi.acm.org/10.1145/98163.98167

[12] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for byzantine fault tolerant services," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP ' 03. New York, NY, USA: ACM, 2003. doi: 10.1145/945445.945470. ISBN 1-58113-757-5 pp. 253–267. [Online]. Available: http://doi.acm.org/10.1145/945445.945470

[13] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002. doi: 10.1145/571637.571640. [Online]. Available: http://doi.acm.org/10.1145/571637.571640

[14] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," *ACM Trans. Comput. Syst.*, vol. 27, no. 4, pp. 7:1–7:39, Jan. 2010. doi: 10.1145/1658357.1658358. [Online]. Available: http://doi.acm.org/10.1145/1658357.1658358

[15] G. Santos Veronese, M. Correia, A. Bessani, and L. C. Lung, "Spin one's wheels? byzantine fault tolerance with a spinning primary," in *28th IEEE International Symposium on Reliable Distributed Systems, 2009. SRDS '09*, Sep. 2009. doi: 10.1109/SRDS.2009.36 pp. 135–144.

[16] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: building efficient replicated state machines for WANs," in *OSDI*, vol. 8, 2008, pp. 369–384. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855767

[17] Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM 27(2), 228–234 (1980)

[18] Ben-Or,M.:Another advantage of free choice(extended abstract):Completely asynchronous agreement protocols. In: PODC 1983, pp. 27–30. ACM, New York (1983)

[19] Bracha, G.: An asynchronous [(n - 1)/3]-resilient consensus protocol. In: PODC 1984, pp. 154–162. ACM, New York (1984)

[20] Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. JACM 35(2), 288–323 (1988)

[21] Martin, J.-P., Alvisi, L.: Fast Byzantine consensus. IEEE Transactions on Depend- able and Secure Computing 3(3), 202–215 (2006)

[22] Borran, F., Schiper, A.: A Leader-free Byzantine Consensus Algorithm. Technical report, EPFL (2009)

[23] Attiya, H., Welch, J.: Distributed Computing: fundamentals, simulations, and advanced topics. John Wiley & Sons, Chichester (2004)

[24] Moses, Y., Waarts, O.: Coordinated traversal: (t + 1)-round byzantine agreement in polynomial time. In: FOCS, pp. 246–255 (1988)

[25] Rachid Guerraoui and Marko Vukoli ´c. Refined quorum systems. Distributed Computing, 23(1):1–42, 2010.

[26] Leslie B. Lamport. Fast byzantine paxos. United States Patent 7620680, filed August 15, 2002, issued November 17, 2009.

[27] Brian Masao Oki. Viewstamped replication for highly available distributed systems. Technical Report MIT/LCS/TR-423, MIT Laboratory for Computer Science, August 1988. (Ph.D. Thesis).

[28] Lamport,L.,“Generalized Consensus and Paxos”,TechnicalReport, MSR-TR-2005-33,Mar.2005,MicrosoftResearch,59pages.

[29] J. Chen and S. Micali. Algorand. Technical report, 2017. URL http://arxiv.org/abs/1607.01341.

[30] G. Brockman. Stellar, July 2014. https://stripe. com/blog/stellar.

[31] J. R. Douceur. The Sybil attack. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS’ 02), Cambridge, MA, Mar. 2002.

[32] S. Micali. Fast and furious Byzantine agreement. In Proceedings of the Innovations in Theoretical Computer Science (ITCS) Conference, 2017.

[33] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), New York, NY, Oct. 1999.

[34] R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Report 2016/917, 2016. http://eprint. iacr.org/.

[35] R. Turpin and B. A. Coan. Extending binary Byzantine agreement to multivalued Byzantine agreement. Information Processing Letters, 18(2):73–76, Feb. 1984.

[36] P. Berman, J. A. Garay, and K. J. Perry. Bit optimal distributed consensus. Computer science: research and applications, pages 313–321, 1992.

[37] B. A. Coan and J. L. Welch. Modular construction of a byzantine agreement protocol with optimal
message bit complexity. Inf. Comput., 97(1):61–85, 1992.

[38] M. Fitzi and M. Hirt. Optimally efficient multi-valued byzantine agreement. In PODC ' 06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing, pages 163–168, New York, NY, USA, 2006. ACM.

[39] G. Liang and N. Vaidya. Capacity of byzantine agreement: Complete characterization of the four node network. Technical Report, CSL, UIUC, April 2010.

[40] I. Damga ̊rd and M. Koprowski, "Practical threshold RSA signatures without a trusted dealer," in *EUROCRYPT*, 2001.

[41] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *J. ACM*, vol. 34, no. 1, pp. 77–97, Jan. 1987.

[42] M. Correia, N. F. Neves, and P. Ver ´ıssimo, "From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures," *Comput. J.*, vol. 49, no. 1, pp. 82–96, Jan. 2006.