恒星共识协议：互联网级共识的联邦模型

David Mazières
恒星发展基金会

摘　　要

本文介绍了一种新型的共识方法——联邦拜占庭一致性协议 (Federated Byzantine Agreement, FBA)。FBA 通过群体切片来保证健壮性——即由单个节点的个体信任决策共同决定系统级别的群体。这些切片联结系统的做法和当今独立网络对等直连、转发决定统一化互联网的方式非常相似。

我们还提出了一种 FBA 的构造方法: 恒星共识协议 (Stellar Consensus Protocol, SCP)。和所有拜占庭一致性协议一样，SCP 不做任何关于攻击者理性行为的假设。此前的拜占庭一致性模型预设一个被全体一致接受的成员关系名单；和它们不同的是，SCP 允许开放的成员关系，这促进了有机的网络式成长。较于去中心化的工作量证明 (PoW) 及权益证明 (PoS) 机制，SCP 对计算能力以及经济成本消耗要求适度，降低了进入门槛并潜在地把金融系统开放给新的参与者。

1　介绍

目前的金融基础设施由一堆混乱的封闭系统组成。这些系统之间的差异不仅造成了交易费用偏高 [13]，同时也导致了跨政治和地理边界的资产转移缓慢 [1, 6]。这种摩擦制约了金融服务的发展，使得数十亿人所受金融服务不足。

为了解决这些问题，我们需要建立一种金融技术设施，让它支持我们从互联网那里看到的有机发展和创新的特色，同时仍然保持金融交易的完整性 (integrity)。历史上，我们依赖于高进入门槛以保证完整性。我们相信知名的金融机构并尽力去规范化它们的行为。然而这种排他性和有机增长的目标矛盾。增长需要新的创新型参与者，而它们可能只拥有不多的财力和计算资源。

我们需要一个向任何人开放的世界性金融网络，这样新的参与者可以加入并扩展未享有服务社群的金融渠道。构建这种网络的挑战在于确保参与者准确记录交易。由于进入门槛较低，用户毋需相信服务提供商来监管他们自己；由于范围遍及全世界，提供商也毋需相信某个单一实体来运营网络。一种倍受瞩目的方案是建立去中心化的系统: 参与者通过对另一参与者的交易有效性达成共识来确保完整性。这种协定取决于世界范围的共识机制。

本文提出了联邦拜占庭协议 (FBA)——一个适用于世界范围的共识模型。在 FBA 中，每个参与者了解它认为重要的其它成员。任何交易在参与者认为已结算之前将等待其它参与者中的绝大多数认可该交易。反之，那些重要的参与者将不认可这笔交易，直到有他们认为重要的其它参与者也认可，以此类推。最终网络中将有足够多的节点接受某笔交易，这使得攻击者无法回滚这一交易。直到那时，所有的参与者才认为本次交易结算完成。FBA 的共识可以确保金融网络的完整性。它的去中心化控制可以激励有机增长。

本文进一步提出了一种 FBA 的构造方法: 恒星共识协议 (SCP)。我们证明 SCP 的安全性对于一个异步的协议是最优的，这是因为它确保了在承认这样该保障的任何节点错误情景下的一致性。我们还将说明，除非参与节点出现错误使得信任依赖无法满足, 否则 SCP 不会引发阻塞状态 (这些状态中不可能存在共识)。SCP 是第一个同时满足下述四个关键属性且被证明安全的共识机制：

- **去中心化控制。**任何人都可以参与，并且不需要中心化权威机构认定 “对共识来说谁的认可是必须的”。
- **低延迟。**实际情形中，节点可以在人们对互联网或者支付交易所期待的时间范围内 (即最多几秒) 达成共识。
- **灵活的信任。**用户有信任他们认为合适的任意团体组合的自由。例如，一个小型非营利组织可能在维持比之大很多的机构的诚实性方面起关键作用。
- **渐进安全性。**安全性取决于数字签名和哈希函数族，其参数能被切实调节到对抗具有难以想像超强计算能力的对手。

机制	去中心化控制	低延迟	灵活的信任	渐进安全性
工作量证明	✓			
权益证明	✓	或许有		或许有
拜占庭一致性		✓	✓	✓
Tendermint	✓	✓		✓
SCP(本工作)	✓	✓	✓	✓

图 1: 不同共识机制的性质比较

在金融市场之外，SCP 也有确保组织诚实履行关键职能的应用。一个不错的例子是数字证书认证机构 (CA)–可认为他们持有互联网网络的钥匙。经验表明 CA 在少数场合签署了不正确的认证 [4, 5]。一些研究 [27, 2, 16, 33] 建议通过认证透明性 (Certificate Transparency) 解决这个问题。SCP 在认证透明性核心问题、增强持久认证历史方面可能会用武之地。因为要求在一个去中心话的审计者中对认证历史达到全球范围内的共识，这将使得退回或者重写之前颁发的认证变得更加困难。

下一节讨论以往的共识方法。第3节定义了联邦拜占庭一致性 (FBA)，同时阐述了 FBA 模型中的安全性和存活性的概念。第4节讨论 FBA 系统中的最优化错误恢复，从而建立了 SCP 的安全性目标。第5节构建联邦选举机制，这是 SCP 协议中关键的一个构建环节。第6节阐述了 SCP 本身，证明了它是安全的并且不会陷入阻塞状态。第7节讨论 SCP 的局限性。最后，第8节概述结论。对于不熟悉数学符号的读者，附录A定义了文中使用的一些符号。

2 相关工作

图1总结了 SCP 和以往的共识机制的区别。最有名的去中心化共识机制是由比特币 [34] 发展而来的工作量证明 (Proof-of-Work，PoW) 模式。比特币采用两方面措施来达成共识。首先，它为理性参与者提供了激励机制以期望他们表现良好。其次，它通过工作量证明 [24] 的算法来结算交易，这一设计保护系统免受没有绝大多数计算能力恶意行为者的影响。比特币强烈地表明了人们去中心化共识的迫切希求 [18]。

然而工作量证明方式是存在局限的。首先，它浪费资源：通过 2014 年一次估计，比特币消耗的电力或许和爱尔兰整个国家的消耗相当 [36] 。第二，安全交易结算需要忍受几分钟甚至几十分钟的预期延时 [26] 。最后，和传统的加密协议不同，工作量证明算法不提供渐进安全。考虑非理性攻击者或者破坏共识的外在激励，少量的计算优势可以使得安全假设失效，导致交易历史被所谓的 “51%” 攻击重写。更糟的是，最初拥有少于 50% 算力的攻击者可以利用系统向加入他们的成员提供不成比例的奖励。作为拥有最大算力支持的标杆数字货币，比特币拥有一定程度面对 51% 攻击的保护能力。稍小一点的系统则已经成了牺牲品 [12, 3]，暴露了任何不基于比特币区块链的工作量证明系统的一个问题。

工作量证明的一个替代方案是权益证明 (Proof-of-State，PoS)，这种系统中的共识依赖提供抵押的各方。类似于工作量证明，奖励鼓励理性参与者遵守协议；有些系统还惩罚作恶 [10, 7] 。权益证明使得所谓的 “零成本” 攻击成为可能，其中之前提供抵押但后来兑换并花掉的各方可以回来重写他们仍然拥有权益时的历史。为了减轻这种攻击的危害，系统把工作量证明结合到权益证明——正比于权益降低所需工作量——或者延缓抵押归还足够长时间直到其它 (有时是非正式的) 共识机制建立不可逆转的检查点。

还有另一个共识方式是拜占庭协议 [39, 31]，其最知名的变形是 PBFT [20]。拜占庭一致性在存在一部分参与者的随意 (包括非理性) 行为的情况下可以确保共识。这方式有两个令人瞩目的属性。首先，共识能够快速而且高效。第二，信任和资源所有权解耦，这使得小的非营利机构帮助更强力的组织比如银行或者认证中心保持诚实成为可能。然而麻烦的是各方必须在精确的参与者列表上达成一致。还有必须防止攻击者多次加入系统或超出系统容错范围，即所谓的 “女巫攻击” [22]。BFT-CUP [14] 容许未知参与者，但仍然预设一个避免女巫的中心化的准入控制机制。

一般而言，拜占庭一致性系统中的成员关系是通过中心权威或者封闭式协商来设定的。之前去中心化准入尝试放弃了一些优势。Ripple 采用的方式是发布一个参与者他们自己可以修改的 “启动器” 成员列表，并希望人们的修改微不足道或者被绝大多数参与者所重现。不幸的是因为不同的列表导致安全担保无效 [8]，实际情形下用户不愿意编辑这个列表，于是巨大的权力最终集中到启动器列表的维护者。另一种方式为 Tendermint 所采用 [11]，它基于权益证明的成员资格。然而这样做又把信任捆绑到资源所有权上了。SCP 是第一个给参与者最大化的自由来选择信任哪些其它参与者组合的拜占庭一致性协议。

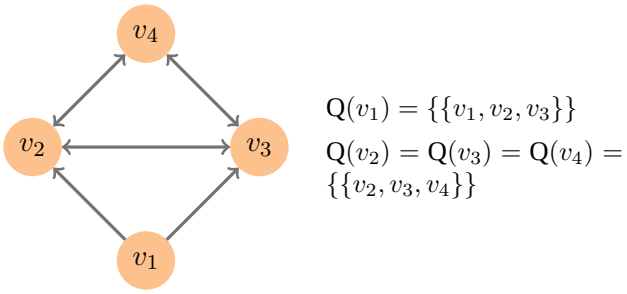

图 2: 不含 v_4 的 v_1 的群体切片不是一个群体

3 联邦拜占庭一致性系统

本节介绍联邦拜占庭一致性 (Federated Byzantine Agreement, FBA) 模型。和非联邦一致性协议一样，FBA 需要解决更新重复的[1]状态的问题，例如事务分类帐或认证树。通过认可应用什么样的更新，节点避免了冲突而不可协调的状态。我们通过一个唯一的存储单元 (slot) 来识别更新，这使得我们可以推断出更新依赖。例如，在一个顺序记录的日志中存储单元可以是连续标号的位置。

FBA 系统运行一个可以确保节点认可存储单元的内容的共识协议。节点 v 在它已经安全地更新了存储单元i 所依赖的所有存储单元之后可以在存储单元i 处安全地更新 x；另外，它相信所有正确工作的节点最终将会认可存储单元i 处的更新 x。外部系统会以不可逆的方式对具体的值作出反应，因此一个节点不能够在之后改变这些值。

FBA 系统的一大挑战是恶意的团体可能参与多次并在数量上超过诚实节点。因此，传统的基于占优数量的群体机制无法适用。取而代之的是，FBA 采用一种分布式的方式来决定群体，这是通过每个节点选择所谓的 “群体切片” 来实现的。接下来的子章节讨论了一些例子。最后，我们定义一个共识协议应当希望满足的关键属性：安全性和完整性的。

3.1 群体切片

在一个共识协议中，节点交换消息来对关于存储单元的陈述进行断言。我们假设这些断言无法被伪造——如果节点通过公钥命名并且数字签名了这些消息那么这是可以保证的。当一个节点侦听到足够多的节点断言了某个陈述之后，它将假定不会有工作节点否定这一陈述。我们称这样的一个足够的集合为**群体切片**，或简称**切片**。为了允许在节点失败的情形下系统仍然能够推进，一个节点可能有多个切片，它的任意一个切片都足够让它相信某个陈述。从较高层次来看，一个 FBA 系统由松散的节点邦联组成，其中的每个节点包含一个或多个切片。更形式化地讲：

定义 3.1 (FBAS) *一个联邦拜占庭一致性系统，或简称* FBAS，*是一个二元组* $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$，*它包含节点集合* $\boldsymbol{V}$ *和群体函数* $\boldsymbol{Q} : \boldsymbol{V} \to 2^{2^{\boldsymbol{V}}} \setminus \{\emptyset\}$，*后者用于指定每个节点的一个或多个切片，这里一个节点属于所有它自己的切片——即，*$\forall v \in \boldsymbol{Q}, \forall q \in \boldsymbol{Q}(v), v \in q$(*注意* 2^X *指的是* X *的幂集*)。

定义 3.2 (**群体**) FBAS$\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ *中的节点集合* $U \subseteq \boldsymbol{V}$ *是一个***群体***当且仅当* $U \neq \emptyset$ *且* U *包含每个节点的一个切片——即，*$\forall v \in U, \exists q \in \boldsymbol{Q}(v), s.t. q \subseteq U$。

群体是足够达成一致的节点集合。切片是说服某一特定节点认可的群体子集。一个群体切片可能小于群体。考虑图2中的四节点系统，每个节点包含单一切片而箭头指向切片中的其它成员。节点 v_1 的切片 $\{v_1, v_2, v_3\}$ 足以说服 v_1 认可。但是 v_2 和 v_3 的切片包含 v_4，这意味着 v_2 和 v_3 都不能在没有 v_4 同意的情况下断言某个陈述。因而，没有 v_4 的参与就不可能达成一致，而唯一的包含 v_1 的群体是所有节点的集合 $\{v_1, v_2, v_3, v_4\}$。

传统非联邦共识要求所有的节点接受相同的切片，即 $\forall v_1, v_2, \boldsymbol{Q}(v_1) = \boldsymbol{Q}(v_2)$。因为任意的切片都足够使得所有的节点相信所有节点的某一个陈述，中心化系统不加区分切片和群体。缺点在于成员关系和群体必须事先规定好，妨碍了开放式成员关系和去中心化控制。传统的系统，例如 PBFT [20]，通常有 $3f + 1$ 个节点，它们中的任意 $2f + 1$ 个节点组成一个群体。这里 f 是拜占庭错误的最大值——意味着节点的随意行为都能保证系统能够存活。

本文中介绍的 FBA 一般化了中心化共识以使得它可以容纳更广泛的环境。FBA 的关键不同在于每个节点选择它自己的群体切片集合 $\boldsymbol{Q}(v)$。从而系统范围内的群体由每个节点的独立决策产生。节

[1]译注：副本 (replica) 是分布式系统中为数据或服务提供的冗余。

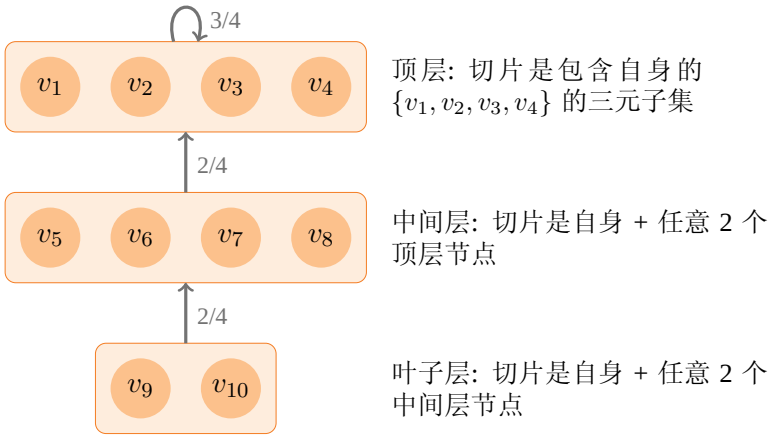

图 3: 层状群体结构举例

点可以根据任意准则选择切片，例如节点的信誉或财务安排。在一些环境下，可能对任何节点来说记录整个系统的所有节点集合 $\boldsymbol{V}$ 是不现实的，然而仍然可以达成共识。

3.2 例子和讨论

图3展示了一个层状系统，系统中的不同节点有着截然不同的切片集合，这可能只有在 FBA 才能实现。顶层由 $v_1, \ldots, v_4$ 组成，其结构类似于 PBFT 中 $f = 1$ 的情形，这意味着只要其它三个节点可达并正常工作，该系统就可以容忍一个拜占庭错误。节点 $v_5, \ldots, v_8$ 组成中间层不相互依赖，而是依赖于顶层。中间层节点只需要两个顶层节点就可以形成切片。(假定顶层只有最多一个拜占庭错误，那么除非整个系统出错否则两个顶层节点不会同时出现错误。) 节点 v_9 和 v_{10} 在叶子层，其切片由任意两个中间层节点组成。注意这里 v_9 和 v_{10} 可能选择不相交的切片集合，例如 $\{v_5, v_6\}$ 和 $\{v_7, v_8\}$；然而，两者都会间接依赖于顶层节点。

实际情形中，顶层可能包含来自各处的四到几十个广为人知并可信的金融机构。当顶层的大小增长时，可能不再有关于它的成员关系的准确一致性，但是大多数团体对顶层节点的意识将会重叠。另外，还可以想象很多的中间层，例如每个代表一个国家或者地理区域。

这种分层系统和域名间的网络路由系统十分类似。当今的网络是由独立的对等直连和网络对间的传输关系共同组成的。没有中央权威机构来指派或仲裁这些安排。然而这些成对的关系也已足够创建出实际意义上的第一层结构——网络服务提供商 (ISP) [35]。尽管英特网的可达性受防火墙影响，但传递可达性几乎是完全的——例如，某个防火墙可能会阻塞纽约时报，但如果它允许 Google 访问，而 Google 能够访问纽约时报，那么纽约时报也间传递性地可达。传递可达性对网站来说或许是受限的设施，但是这对共识至关重要；等价的例子是 Google 仅当在纽约时报接受某个陈述的时候才接受该陈述。

如果我们把群体切片看成类似网络可达性，而把群体看成是传递可达性，那么网络近乎完全的传递可达性启示我们同样也可以利用 FBA 达成世界范围内的共识。在很多方面，共识比网际间的路由选择要容易得多。传输消耗资源且花费资金，但包含切片的过程仅仅要求检查数字签名。因此，FBA 节点可以在包含切片的那端报告错误，相比较常见的在对等直连和传输约定场景见到的那样，这可以用更为相互依赖且冗余的方式来建立保守的切片。

另外，正如图4中展示的那样，中心化共识对有环依赖结构也无能为力。这样一个环状结构不会被有意生成出来，然而当独立的节点选择它们自己的切片时，这可能导致整个网络最终被植入了环状依赖。更大的问题是，相比较传统的拜占庭一致性来说，一个 FBA 的协议必须解决远为多样的群体结构。

3.3 安全性和存活性

我们把节点分为*良性行为的*和*恶性行为的*。一个良性行为的节点选择可感知的群体切片 (将在第 4 节深入讨论) 并且遵守规则，这包括最终回应所有的请求。一个恶性行为的节点不是这样。恶性行为的节点受拜占庭错误的影响，这意味着它们可能会有随意的行为。例如，一个恶性节点可能会被入侵而它的所有者可能恶意地修改软件，或者它可能已经崩溃。

拜占庭一致性的目标是要确保良性行为的节点要在恶性行为节点存在的情形下对外界表现出相同的值。这个目标包含两个方面。首先，我们希望防止节点分叉并对同一存储单元向外界显示不同的

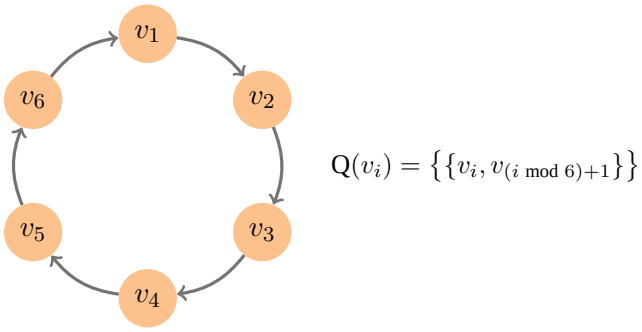

图 4: 环形群体结构举例

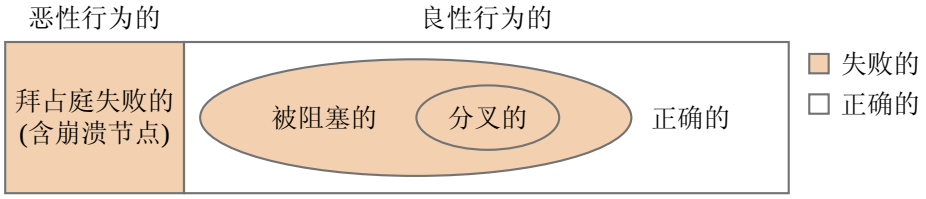

图 5: 节点失败情形的维恩图

值。其次，我们需要确保节点的确会得到值，而不是在某个死的状态下阻塞，这种情形下的共识变得不再可能。我们为这些属性引入下面两个术语：

定义 3.3 (安全性) *FBAS 中的一组节点集合，如果当中的任意两个节点都不会向外界表现出不同的值，那么它享有* **安全性**。

定义 3.4 (存活性) *在 FBAS 中，如果给予合适的消息发送和时间选择节点就能够向外界表现出新值的话，那么该节点享有* **存活性**。

我们称这些既有安全性又有存活性节点是*正确的*。不正确的节点被称为*有错误的*。一个恶性行为的节点是有错误的，然而一个良性行为的节点也可能是有错误的——它可能无限等待某恶性行为节点的消息，或者更为严重的是它的状态被恶性行为节点用错误的信息给破坏了。

图5强调了这样一种可能的节点错误。左边是拜占庭错误——恶性行为的节点。右边是两类良性行为的、但有错误的节点。不具有存活性的节点被标成了*阻塞的*，而那些不具有安全性的节点被标成了*分叉的*。破坏安全性的攻击严格地比破坏存活性的攻击更为强大。因此我们把分叉节点归为阻塞节点的子集。

我们对*存活性*的定义是比较弱的，这是因为它承认*持久性优先抢占*，在这样一种状态下总是可能达成共识，但是网络持续地用错误的方式延迟或调整重要信息的顺序来阻碍共识的形成。持久性优先抢占在一个纯异步、确定的、并能够免受节点错误干扰的系统里面是不可避免的 [25]。幸运的是，抢占是暂时的。它并不意味着节点错误，因为系统可能会在任何时刻恢复。协议可以通过随机性 (接入节点总是选择随机性的候选值直到有足够多节点正好选择了同一个节点 [17, 19]) 或对消息延迟的现实情况下的假设 [23] 来缓解这个问题。当希望限制执行时间时后一类协议更为实用。当然，结束性要求 (termination) 而不是安全性要求依赖于消息超时设置。

4 最优恢复

节点是否安全且可活依赖于多方面因素：它们选择了什么样的群体切片，哪些节点是恶性行为的，当然还有具体的共识协议和网络行为。像对待通常的异步系统一样，我们假设系统最终将在良性行为的节点之间传递消息，但是可能会随意性地延迟或重新排序消息。

本节回答下面这个问题：给定一个特定的 $\langle \boldsymbol{Q}, \boldsymbol{V} \rangle$ 和恶性行为节点子集 $\boldsymbol{V}$，在忽略网络因素的情形下任何联邦拜占庭一致性协议能够确保的最佳安全性和存活性是什么？我们首先讨论群体交这一属性，没有它安全性是无法保证的。接着我们引入可去集的概念——这是一个错误节点的集合，不考虑它们系统仍然有可能保持安全性和存活性。

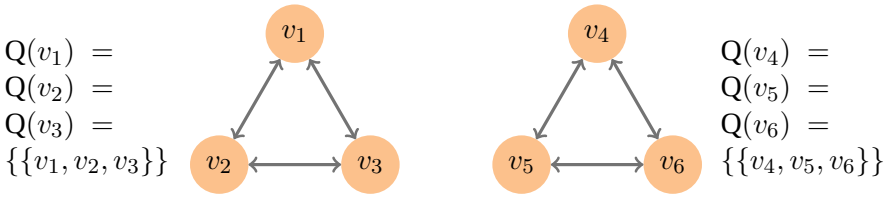

图 6: 不含群体交的 FBAS

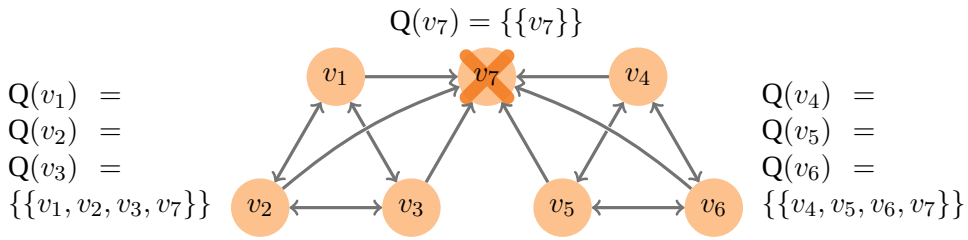

图 7: 恶性行为节点 v_7 可能破坏群体交

4.1 群体交

仅当用函数 $\boldsymbol{Q}$ 表示的群体切片满足一种有效性属性的时候协议才能保证可达到一致，我们把这一属性称为群体交。

定义 4.1 (群体交) 一个 FBAS 包含**群体交**当且仅当它的任意两个群体共享一个节点，即 $\forall U_1 \in \boldsymbol{Q}, U_2 \in \boldsymbol{Q}, U_1 \cap U_2 \neq \emptyset$。

图6展示了一个不具有此特性的系统，这里 $\boldsymbol{Q}$ 允许两个群体$\{v_1, v_2, v_3\}$ 和 $\{v_4, v_5, v_6\}$，它们彼此不相交。不相交的集合可能会独立地认可相互矛盾的陈述，暗中破坏系统范围内的一致性。当很多的群体存在时，如果存在两个不相交的话那么群体交将会失败。例如，图6中的节点集合 $\{v_1, \ldots, v_6\}$ 是一个和另外两个群体相交的群体，但是系统本身仍然缺少群体交。

没有一种协议能够在没有群体交的情形下保证安全性，因为这样的配置可能会以两个对对方一无所知的 FBAS 系统的方式运行着。然而即使有群体交，在存在恶性行为节点的情形下安全性也可能无法保证。比较有两个不相交集合的图6和在恶性行为的节点 V_7 处有交的两个群体的图7。如果 v_7 产生对左右两个群体产生两个不相容的陈述，其结果等价于不相交群体。

事实上，由于恶性节点对安全性毫无用处，任何一个协议必须在良性行为节点在它们自己那里享有群体交的情形下才能保证安全性。毕竟，在安全性最坏的场景下，所有的恶性行为的节点都可能持续性地做出不一致的陈述来使得群体产生分歧。两个仅在恶性节点处相重叠的群体由于恶性节点的欺骗同样会像运行在两个不同的 FBAS 系统里一样。总之，FBAS$\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 可以在节点集合 $B \subseteq \boldsymbol{V}$ 的拜占庭错误中生存下来当且仅当 $\langle \boldsymbol{Q}, \boldsymbol{V} \rangle$ 在从 $\boldsymbol{V}$ 和 $\boldsymbol{Q}$ 中的所有切片中删除了 B 中的节点之后仍然有群体交。更形式化的描述是：

定义 4.2 (删除) 如果 $\langle \boldsymbol{Q}, \boldsymbol{V} \rangle$ 是一个 FBAS 并且 $B \subseteq \boldsymbol{V}$ 是一个节点集合，那么从 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 中删除 B(记作 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$)，意味着计算修改过后的 FBAS $\langle \boldsymbol{V} \backslash B, \boldsymbol{Q}^B \rangle$，这里 $\boldsymbol{Q}^B(v) = \{q \backslash B | q \in \boldsymbol{Q}(v)\}$。

每个节点 v 有责任确保 $\boldsymbol{Q}(v)$ 不会违背群体交。一种做法是选择保守的切片，这会导致较大的群体。当然，一个恶意的 v 可能会故意地选择能够违反群体交的 $\boldsymbol{Q}$。然而一个恶意的 v 还可能对 $\boldsymbol{Q}(v)$ 的值撒谎或者忽略 $\boldsymbol{Q}(v)$ 来做出随意的断言。总之，当 v 是恶性行为的时候 $\boldsymbol{Q}(v)$ 的值是没有意义的。这就是为什么安全性的必要属性——在删除恶性行为节点之后的良性行为节点仍然有群体交——是不受恶性行为节点的切片影响的。

假定图6是由一个三节点 v_1, v_2, v_3 含群体交的 FBAS 系统进化成的一个六节点无群体交的 FBAS 系统。当 v_4, v_5, v_6 加入时，它们恶意地选择切片来破坏群体交，这样没有节点能够为 $\boldsymbol{V}$ 确保安全性。幸运的是，删除这些节点得到 $\langle \boldsymbol{Q}, \boldsymbol{V} \rangle^{\{v_4, v_5, v_6\}}$ 重新恢复群体交属性，这意味着至少 $\{v_1, v_2, v_3\}$ 能够保证安全性。注意这里的删除是概念性的，用于表述最优安全性。一个协议应当为 v_1, v_2, v_3 保证安全性而不需要它们意识到 v_4, v_5, v_6 是恶性行为的。

良性 / 恶性行为的	节点的局部属性，独立于其它节点 (除非用于切片选择验证)。
完好的 / 被污染的	给定节点群体切片和特定恶性行为节点集合下的属性。被污染的节点是恶性行为的或者 (可能是间接地) 依赖于很多恶性行为节点。
正确的 / 失败的	给定节点的群体切片、具体协议及实际网络行为下的节点属性。共识协议的目标就在于在任何只要可能的情形下保证节点的正确性。

图 8: FBAS 节点的关键属性

4.2 可去集 ($DSet$)

我们用可去集 (或 $DSet$) 捕捉节点在切片选择时的容错能力。通俗地讲，在一个 $DSet$ 之外的安全性和存活性是可以保证的，而不需要考虑 $DSet$ 中的节点的行为。换句话说，在一个最优可恢复的 FBAS 中，如果一个 $DSet$ 包含每个恶性行为节点，它也会包含每个错误节点；而反之所有在 $DSet$ 之外的节点都是正确的。举例来说，在一个有 $3f+1$ 个节点且群体大小为 $2f+1$ 的中心化 FBPT 系统中，任意小于等于 f 个节点可以组成一个 $DSet$。由于 FBPT 事实上能够在有 f 个错误节点的情形下存活下来，它的健壮性是最优的。

在不那么典型的图3的例子中，$\{v_1\}$ 是一个 $DSet$，这是因为顶层的节点可能会出现错误而不影响系统的其余部分。$\{v_9\}$ 也是一个 $DSet$，这是因为其它的节点的正确性都不依赖 v_9。$\{v_6,\ldots,v_{10}\}$ 是一个 $DSet$，这是因为 v_5 和顶层节点都不会依赖于任意其它 5 个节点。$\{v_5,v_6\}$ 不是一个 $DSet$，因为它是 v_9 和 v_{10} 的一个切片，因此如果它们是完全恶意的话，它可能对 v_9 和 v_{10} 撒谎，并且让它们断言相互矛盾或和系统内其它节点矛盾的陈述。

为了防止一个错误行为的 $DSet$ 影响其它节点的正确性，系统必须满足两个性质。对于安全性，删除 $DSet$ 不会暗中破坏群体交。对于存活性，$DSet$ 不能够否认其它节点的工作群体。这使得我们有以下定义：

定义 4.3 (DSet) 令 $\langle \boldsymbol{V},\boldsymbol{Q}\rangle$ 是一个 FBAS 而 $B\subseteq \boldsymbol{Q}$ 是一个节点集合，我们称集合 B 是可去集 (或 $DSet$) 当且仅当：

1. (除 B 群体可交性) $\langle \boldsymbol{V},\boldsymbol{Q}\rangle^B$ 是群体可交的；
2. (除 B 群体可达性) 要么 $\boldsymbol{Q}\backslash B$ 是 $\langle \boldsymbol{V},\boldsymbol{Q}\rangle$ 中的一个群体，要么 $B=\boldsymbol{V}$。

除 B 群体可达性防止 B 中的节点拒绝回复请求或阻塞其它节点运行。除 B 群体可交性防止对立的情形——B 中的节点制造矛盾的断言而让其它的节点对同一个存储单元对外产生不一致的值。在切片选择时节点必须权衡这两类威胁。其它条件相同的情况下，大一点的切片会产生大一点的群体从而会有更多的重叠，这意味着更少的错误节点集合 B 在删除时将会破坏群体交。另一方面，大一点的切片更可能包含错误节点，这会危及到群体的可达性。

最小的包含所有恶性行为的节点的 $DSet$ 可能也会包含良性行为的节点；这反映了这样一个事实：一个足够大的恶性行为的节点集合可能会导致良性行为的节点出现错误。例如，在图3中，包含 v_5 和 v_6 的最小 $DSet$ 是 $\{v_5,v_6,v_9,v_{10}\}$。在一种特殊的情形下，$\boldsymbol{V}$ 是 $DSet$。这个特殊情形的动机在于，如果所有的节点都出现了错误，那么剩余的 (零个) 节点自然是正确的。给定足够多的恶性行为的节点，包含所有节点的 $\boldsymbol{V}$ 可能是包含所有恶性行为节点的最小 $DSet$，这意味着没有协议能够保证有比整个系统失败更好的结果了。

一个 FBAS 系统中的 $DSet$ 是由群体函数 $\boldsymbol{Q}$ 事先决定的。而哪个节点的行为是良性或是恶性的由运行时行为决定，例如机器被入侵了。我们关心的 $DSet$ 是那些包含恶性行为的节点，因为它们能够帮助我们把可以保证正确的节点从不能保证正确的节点中辨别出来。有鉴于此，我们引入下面的术语：

定义 4.4 (完好的) FBAS 中的节点 v 是**完好的**当且仅当存在一个包含所有恶性行为的 $DSet$ 集合 B 且 $v\notin B$。

定义 4.5 (被污染的) FBAS 中的节点 v 是**被污染的**当且仅当它不是完好的。

即使一个节点 v 本身是良性行为的，当它被足够多的错误节点所包围而被阻塞进程或状态被破坏时，它也是被污染的。FBAS 都不能保证被污染节点的正确性。图8概述了节点的关键属性。下面的定理减轻了分析的困难，它们表明被污染节点集合总是 FBAS 中的一个有群体交的 $DSet$。

定理 1 令 U 是 FBAS $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 中的一个群体，$B \subseteq \boldsymbol{Q}$ 是节点集合，并令 $U' = U \backslash B$。若 $U' \neq \emptyset$，则 U' 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 中的一个群体。

证明 1 因为 U 是一个群体，每个 U 中的节点 v 都有一个 $q \in \boldsymbol{Q}(v)$ 使得 $q \subseteq U$。由于 $U' \subseteq U$，则有对每个节点 $v \in U'$ 存在 $q \in \boldsymbol{Q}(v)$ 使得 $q \backslash B \subseteq U'$。重新使用删除记号书写即得 $\forall v \in U', \exists q \in \boldsymbol{Q}^B(v)$ 使得 $q \subseteq U'$；因为 $U' \subseteq \boldsymbol{Q} \backslash B$，这表示 U' 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 的一个群体。

定理 2 如果 B_1 和 B_2 是 FBAS $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 的一个有群体交的 $DSet$，那么 $B = B_1 \cap B_2$ 也是一个 $DSet$。

证明 2 令 $U_1 = \boldsymbol{Q} \backslash B_1$ 且 $U_2 = \boldsymbol{Q} \backslash B_2$。如果 $U_1 = \emptyset$，那么 $B_1 = \boldsymbol{V}$，有 $B = B_2$(是一个 $DSet$)，结论成立。类似地，如果 $U_2 = \emptyset$，那么 $B = B_1$，结论成立。否则，注意到除 $DSet$ B_1 和 B_2 的群体可达性，U_1 和 U_2 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 中的群体。由定义可知，两个群体的并仍然是群体[2]。因此 $\boldsymbol{Q} \backslash B = U_1 \cup U_2$ 是一个群体且它有除 B 群体可达性[3]。

我们需要说明除 B 群体可交性。令 U_a 和 U_b 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 的任意两个群体，$U = U_1 \cap U_2 = U_2 \backslash B_1$。根据 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 的群体交性质，$U = U_1 \cap U_2 \neq \emptyset$。但由定理1，$U = U_2 \backslash B_1$ 必为 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^{B_1}$ 的一个群体。考虑到 $U_a \backslash B_1$ 和 $U_a \backslash B_2$ 不可能同时为空集，否则 $U_a \backslash B$ 为空与 U_a 是群体的定义矛盾。因此，由定理1，要么 $U_a \backslash B_1$ 是 $(\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B)^{B_1} = \langle \boldsymbol{V}, \boldsymbol{Q} \rangle^{B_1}$ 的一个群体，要么 $U_a \backslash B_2$ 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^{B_2}$ 的一个群体，或者两者都成立。在前一种情形下，注意到假如 $U_a \backslash B_1$ 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^{B_1}$ 的群体，那么由 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^{B_1}$ 群体可交性可得 $(U_a \backslash B_1) \cap U \neq \emptyset$；由于 $(U_a \backslash B_1) \cap U = (U_a \backslash B_1) \backslash B_2$，可知 $U_a \backslash B_2 \neq \emptyset$，从而 $U_a \backslash B_2$ 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^{B_2}$ 中的一个群体 (定理1)。类似地，$U_b \backslash B_2$ 也必是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^{B_2}$ 的一个群体。但除 B 群体可交性告诉我们 $(U_a \backslash B_2) \cap (U_b \backslash B_2) \neq \emptyset$，这仅在 $U_a \cap U_b \neq \emptyset$ 时可能。

定理 3 在一个有群体交的 FBAS 中，被污染的集合是一个 $DSet$。

证明 3 令 B_{min} 是一个包含所有恶性行为节点的 $DSet$ 的交集。由**完整性**的定义可知一个节点 v 是完好的当且仅当 $v \notin B_{min}$[4]。因此 B_{min} 就是被污染节点集合。由定理2可知，$DSet$ 在交的意义下是闭的，因此 B_{min} 也是一个 $DSet$。

5 联邦选举系统

本节开发了一套可供 FBAS 中的节点对某个陈述认可的联邦选举技术。从高层次来看，对某个陈述 a 认可的过程涉及节点交换两个集合的消息。首先，节点为 a 投票。其次，如果投票成功，节点确认 a，在第一次投票成功的基础上有效地举行第二次投票。

在每个节点看来，两个回合的消息把对陈述 a 的认可分成了三个阶段：不知道的，认可的和确认的。最初，a 的状态对一个节点来说是一无所知的——a 可能最终为真，为假，甚或卡在了一个长期不确定的状态上。如果第一次投票成功，v 可能会接受 a。两个完好节点永远不会接受相互矛盾的陈述，因此如果 v 是完好的并且接受了 a 那么 a 就不可能是假的。

然而由于两种原因，v 接受 a 并不能说明 a 为真。首先，v 接受 a 这一事实并未表明所有完好节点都能接受 a。其次，如果 v 是被污染的，那么接受 a 不能说明任何问题——a 可能在完整节点处被认为是假的。然而即使 v 被污染了——v 并不知道这一点——系统仍然可以有良性行为节点的群体可交性，在这种情形下为了确保最优安全性 v 需要对 a 断言有更高的保障≪TODO (语句不通)≫。举行第二次选举用来解决这两个问题。如果第二次选举成功，v 转为已确认状态，这时它可以最终认为 a 为真并作用于 a 上。

下面的子章节阐述了联邦选举过程的细节。因为投票并没有去除陈述被阻塞的可能性，第5.6节讨论如何处理它们。在第6节里我们将联邦选举传统转化成一个共识协议，它可以避免完整节点的被阻塞存储单元的可能性。

5.1 开放式成员关系下的投票系统

在拜占庭一致性系统中，一个正确的节点仅会在它知道其它的正确节点永远不会接受一个和 a 相冲突的陈述的情形下接受 a。大多数协议为实现这一目标采用了投票方式。良性行为节点仅当在 a 有效时才会给 a 投赞成票。良性行为节点也从不改变它们的投票结果。因此，在一个中心化拜占庭一致性中，如果绝大多数的良性节点 (即群体) 投票赞成 a 的话，那么接受 a 是安全的。当一个陈述收到必要的赞成票之后，我们称这样的陈述是被批准的。

[2]译注：设 $U = U_1 \cup U_2, \forall v \in U$，必有 $v \in U_1$ 或 $v \in U_2$；不妨设 $v \in U_1$，由 U_1 定义可知，$\exists q \in \boldsymbol{Q}(v), s.t., q \subseteq U_1 \subseteq U$。
[3]译注：因为 $(\boldsymbol{Q} \backslash B) \backslash B = \boldsymbol{Q} \backslash B$ 是群体。
[4]译注：$v \notin B_{min} = \bigcap B_i \iff \exists B_k \ s.t., v \notin B_k$，后者正是 v 是完好节点的定义。

在一个联邦的环境中，我们必须调整选举机制以让它适应开放式成员关系。一个不同之处是群体不再对应于大多数良性行为节点。然而，多数性要求首先用于确保良性节点的群体交，在第4.1节我们已经调整它使之适用于 FBA。开放成员关系还暗示了另一个问题需要解决：节点必须在选举过程中发现哪些节点组成了一个群体。为了发现群体，一个协议在所有来自 v 的消息中包含 $\boldsymbol{Q}(v)$，或提供其它一些办法来向 v 查询 $\boldsymbol{Q}(v)$。

定义 5.1 **(投票)** 一个节点 v **投票**赞成一个 (抽象) 陈述 a 当且仅当

1. v 断言 a 是有效的，并且和其它已被 v 接受的陈述相一致；且
2. v 断言 v 从未投票反对 a——即，投票赞成一个和 a 相冲突的陈述——且 v 承诺在未来决不会给 a 投反对票。

定义 5.2 **(批准)** 一个群体U_a **批准**一个陈述 a 当且仅当每个 U_a 中的成员都投票赞成 a。一个节点**批准** a 当且仅当 v 是一个批准 a 的群体U_a 中的成员。

定理 4 两个冲突的陈述 a 和 $\bar{a}$ 不会在一个在群体交且不含恶性行为节点的 FBAS 中同时被批准。

证明 4 反证法。假定群体U_1 批准了 a 而群体U_2 批准了 $\bar{a}$。根据群体可交性，$\exists v \in U_1 \cap U_2$。这样的节点 v 必然已经非法地投票赞成了 a 和 $\bar{a}$，和没有恶性行为节点的假设相冲突。

定理 5 令 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 是一个有除 B 群体可交性的 FBAS 系统并假定 B 包含恶性行为的节点。设 v_1 和 v_2 是两个不在 B 中的节点，a 和 $\bar{a}$ 是相互冲突的陈述。则有，如果 v_1 批准了 a 那么 v_2 不会批准 $\bar{a}$。

证明 5 反证法。假设 v_1 批准了 a 且 v_2 批准了 $\bar{a}$。由定义可得，存在一个包含 v_1 的群体U_1 批准了 a，存在一个包含 v_2 的群体U_2 批准了 $\bar{a}$。由定理1，由于 $U_1 \backslash B \neq \emptyset$ 且 $U_2 \backslash B \neq \emptyset$。两者必是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$，这意味着它们各自在 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 批准了 a 和 $\bar{a}$。然而 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 具有群体可交性且不含恶性行为节点，由定理4可知 a 和 $\bar{a}$ 不可能同时被批准。

定理 6 一个具有群体交的 FBAS 中的两个完整节点不可能批准相互冲突的陈述。

证明 6 设 B 是被污染的节点集。由定理3，B 是一个 $DSet$。由 $DSet$ 的定义，$\langle \boldsymbol{V}, \boldsymbol{Q}, \rangle$ 有除 B 群体可交性。由定理5可得不在 B 中的节点不可能批准冲突的陈述。

5.2 阻塞集

在中心化共识中，存活性是系统的一个或者都有或者都没有的属性。要么一个全体一致的良性行为群体存在，要么恶性行为节点会阻塞系统的其它部分接受新的陈述。在 FBA 中则不然，存活性可能在不同的节点间并不相同。例如，在图 3的分层群体例子中，如果中间层的 v_6, v_7, v_8 崩溃了，叶子层将会被阻塞，而顶层节点及 v_5 将继续具有存活性。

仅当 $\boldsymbol{Q}(v)$ 至少包含一个仅由正确节点组成的群体切片时，FBA 协议可以保证对一个节点 v 的存活性。如果错误节点集合 B 至少包含每个 v 的切片的一个成员的话那么 B 会破坏这一属性。我们称集合 B 是一个v–阻塞，因为它会阻塞 v 的进展。

定义 5.3 **(v–阻塞)** 设 $v \in \boldsymbol{V}$ 是 FBAS $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 中的一个节点。集合 $B \subseteq \boldsymbol{Q}$ 是v–阻塞的当且仅当它和 v 的每个切片都有交集——即，$\forall q \in \boldsymbol{Q}(v), q \cap B \neq \emptyset$。

定理 7 设 $B \subseteq \boldsymbol{V}$ 是 FBAS $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 的一个节点集合。$\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 具有除 B 群体可达性当且仅当对任何 $\boldsymbol{V} \backslash B$ 来说 B 不是v–阻塞。

证明 7 "$\forall v \in \boldsymbol{V} \backslash B, B$ 不是v–阻塞" 和 "$\forall v \in \boldsymbol{V} \backslash B, \exists q \in \boldsymbol{Q}(v)$ 使得 $q \subseteq \boldsymbol{V} \backslash B$" 等价。由群体的定义，后者成立当且仅当 $\boldsymbol{V} \backslash B$ 是一个群体或者 $B = \boldsymbol{V}$，亦即除 B 群体可达性的严格定义。

作为结论，对任何完好节点 v 来说，被污染节点的 $DSet$ 不是v–阻塞的。

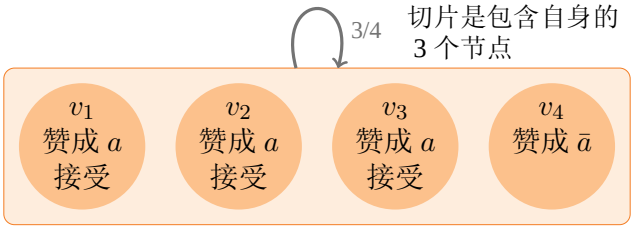

图 9: v_4 投票赞成 $\bar{a}$, 而它和已被批准的陈述 a 矛盾。

5.3 接受陈述

当一个完好节点 v 意识到它已经批准了一个陈述之后，定理6告诉 v 其它完好节点不会批准与之相冲突的陈述。这是让 v 接受 a 的充分条件，但我们不能让它也是必要的。批准一个陈述要求给它投票，而一些节点可能已经赞成了和 a 相冲突的陈述。例如在图9中，v_4 在意识到其它几个节点赞成 a 之前投票赞成了 $\bar{a}$。尽管 v_4 不能够赞成 a，我们仍然希望它接受 a 以和其它节点相一致。

一个关键的认识是，如果一个节点 v 是完好的，那么不存在一个完全包含被污染节点的v–阻塞集 B。现在假定 B 是一个v–阻塞集而 B 中每个成员都要求接受陈述 a。如果 v 是完好的，B 中至少有一个元素也是完好的。完好节点不会对接受陈述 a 撒谎；因此，a 为真并且 v 能够接受它。当然，如果 v 是被污染的，那么 a 就可能不是真的。但是一个被污染的节点可能会接受任何陈述而显然不影响完好节点的安全性。

定义 5.4 (接受) *一个 FBAS 的节点 v **接受**一个陈述 a 当且仅当它从未接受过和 a 相冲突的陈述且它决定下述条件之一:*

1. *存在一个群体U 使得 $v \in U$ 且 U 中的每个成员要么投票赞成了 a 要么声称将接受 a;*
2. *每个v–阻塞的集合声称接受 a。*

尽管一个良性行为的节点不会投票赞成相互矛盾的陈述，上面的条件2允许一个节点投票赞成一个陈述但之后接受一个相冲突的陈述。

定理 8 *在一个有群体交的 FBAS 系统中的两个完整节点不会接受相互冲突的陈述。*

证明 8 *设 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 是有群体交的 FBAS，B 是被污染节点的 $DSet$(其存在性由定理3给出)。假定一个完好节点接受陈述 a。设 v 是第一个接受 a 的节点。在 v 接受 a 时，只有 B 中的被污染节点能够声称接受它。由定理7的结论，B 不会是v–阻塞的，则必是通过条件1来接受 a 的。因此，v 可以找到一个群体U 使得其中的每个节点声称投票赞成或接受 a，而由于 v 是第一个接受 a 的完好节点，这必意味着 $U \backslash B$ 中的节点都投票赞成 a。换句话说，v 在 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 中批准了 a。一般化而言，任何一个被 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 中完好节点所接受的陈述必定在 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 中被批准。由于 B 是一个 DSet，$\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 具有群体交。另外由于 B 包含所有的恶性行为节点，定理4过滤掉了那些对相互冲突的节点的批准。*

5.4 接受是不够的

不幸的是，假定被接受陈述的正确性在一个联邦共识协议中带来的安全性和存活性不是最优的。我们依次来讨论安全性和存活性。给定一些情景，我们然后讨论为什么在 FBA 中的问题比中心化拜占庭一致性中的更为棘手。

5.4.1 安全性

考虑一个仅包含唯一的、全体一致同意的群体的 FBAS $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$——即，$\forall v, \boldsymbol{Q}(v) = \{\boldsymbol{V}\}$。这应当是一个安全性的保守选择——一个节点只有在每个节点都同意之后才会做一件事。然而由于对任一 v 来说每个节点都是v–阻塞的，任何节点都能够独立地说服其它节点接受随意的陈述。

问题在于只有在完好节点中被接受陈述才是安全的。但正如第4.1讨论的那样，保证正确性的唯一的必要条件是良性行为节点的群体交，这有可能在一些良性行为节点被污染的情形下也会成立。特别地，当 $\boldsymbol{Q}(v) = \boldsymbol{V}$ 时，仅有的 $DSet$ 是 $\emptyset$ 和 $\boldsymbol{V}$，这意味着任何一个节点错误将污染整个系统。然而，除 $B \subseteq \boldsymbol{V}$ 群体可交性仍然成立。

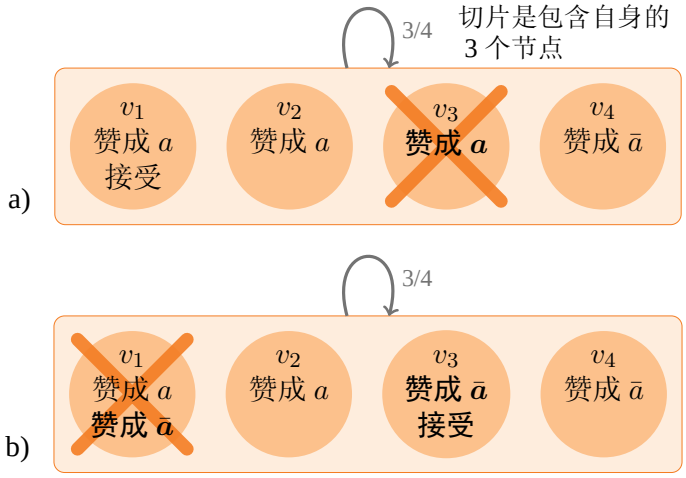

图 10: v_2 当 v_2 没有看到加粗消息时 v_2 无法区分的两种情形

5.4.2 存活性

另一个被接受陈述的局限性是，其它的完好节点可能不会接受它们。对存活性来说，这种可能性使得依赖于被接受的陈述是有问题的。如果一个节点由于它接受了一个陈述而作用于该陈述上，其它的节点可能无法以类似的方式处理该陈述。

考虑图10a 的情形，其中 v_3 在帮助批准 v_1 并接受陈述 a 之后崩溃了。尽管 v_1 接受了 a，v_2 和 v_4 却不能。特别地，在 v_2 看来，所描述的情景和图10b 是一样的，后者 v_3 赞成 $\bar{a}$ 且是良性行为的但回复较慢，而 v_1 是恶性行为的并在向 v_3 发送对 $\bar{a}$ 的赞成票 (因此导致 v_3 接受 $\bar{a}$) 的同时又向 v_2 发送对 a 的赞成票。

为了支持像图10a 中协议层次的存活性概念，v_1 需要一种方式来确保每个其它的完好节点最终能够在作用于 a 之前接受 a。一旦是这种情形，则说系统在 a 上达成了一致就合情合理了。

定义 5.5 **(认可)** 一个 FBAS $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ **认可**陈述 a 当且仅当无论接着会发生什么，一旦足够多的消息被发送且处理了，每个完好节点将接受 a。

5.4.3 和中心化选举的比较

为了理解上述问题为何会出现在联邦选举中，考虑一个由 N 个节点组成且群体大小为 T 的中心化拜占庭系统。这样一个系统在含不大于 $f_L = N - T$ 个节点错误的情况下具有群体可达性。由于任意两个群体共享至少 $2T - N$ 个节点，良性行为的节点的群体交持最多可容纳 $f_S = 2T - N - 1$ 个拜占庭错误。

中心化拜占庭一致性系统通常设置 $N = 3f + 1, T = 2f + 1$ 来使得 $f_L = f_S = f$，该处的平衡点的安全性和存活性有相同的容错能力。如果安全性比存活性更为重要，一些协议增加 T 以使得 $f_S > f_L$ [32]。在 FBA 中由于群体有机成长，系统发现并使得自身处于平衡状态的可能性不大，这让在不谈存活性的情况下维持安全性变得更为重要。

现在考虑在中心化系统中一个节点 v 反对一个已被批准的陈述 a 的情形。如果 v 侦听到 $f_S + 1$ 个节点批准了 a，v 知道要么它们当中的一个是良性行为的，要么所有的安全性保证都崩溃了。不论哪种情形，v 能在不损失安全性的条件下立即作用于 a。FBA 的等价的情景是侦听集合 B，若 B 被删除则会损害良性节点的群体交。因为下面三个原因辨别出这样一个 B 是需要技巧的：第一，群体是动态被发现的；第二，恶性行为的节点可能会对切片撒谎；第三，v 不知道哪个节点是良性行为的。取而代之的做法是，我们在v–阻塞接受 a 时定义联邦选举以接受 a。v–阻塞属性有易于被检测的优势，但等价于在中心化系统中当我们确实需要 $f_S + 1$ 安全能力时侦听 $f_L + 1$ 个节点。

为了确保在中心化系统中获得所有良性行为节点的认可，只需要 $f_L + f_S + 1$ 个节点承认一个陈述被批准了。如果它们当中的多于 f_L 出现错误，我们不再期望存活性。如果 f_L 或更少的节点出现了错误，那么我们知道 $f_S + 1$ 个节点仍然愿意作证批准，这反过来说服所有其它的良性行为节点。然而 f_S 的可恢复性在 FBA 模型中也没有简单的对等情形。

换句话说，在某一时刻一个节点需要强烈信任某个陈述以依赖于它对安全性的真假。一个中心化的系统为陈述 a 提供两种途径来达到这个状态：直接批准 a，或者从 $f_S + 1$ 个声明 a 被批准的节点上回溯推断，如果发现它们都撒谎了则认为安全性是不可期望的。FBA 缺少后面一种方法；仅有的在良

性行为节点上用于安全性的工具是直接批准。由于节点仍然需要有一种方式来克服给已被批准的陈述投反对票的情形，我们引入了接受的概念，但是它提供了一个限于完好节点的稍弱的相容性保障。

5.5 陈述确认

两类被接受陈述的局限性都来源于这一事实：一个完好节点 v 可能会给一个已经被批准的陈述 a 投反对票。在反对 a 之后，它不会再投赞成票，这使得 v 将不可能批准 a。为了给 v 提供一种在给 a 投反对票之后仍然能够批准 a 的方法，接受的定义给出了一个基于v–阻塞的第二准则。但是这个第二准则弱于批准，对有群体交的被污染节点没有任何保障。

现在如果一个陈述 a 有 “从来没有任何完好节点反对它” 这样一个属性，那么我们没有必要去接受它。完好节点可以简单地批准 a，而我们可以在作用于 a 之前要求它们这样做。我们成这样的陈述是不可驳斥的。

定义 5.6 (不可驳斥) *当没有完好节点可以反对一个陈述 a 时，称这条陈述在 FBAS 中是*不可驳斥的。

定理8告诉我们两个完好节点不可能接受相互冲突的陈述。因此，尽管一些节点可能会反对某个被完好节点接受的陈述 a，对 “一个完好节点接受了 a” 这一陈述是不可反驳的。这建议我们举行第二次投票来批准 “一个完好节点接受了 a” 这一事实。

定义 5.7 (确认) *在 FBAS 中一个群体U_a* **确认***一个陈述 a 当且仅当 $\forall v \in U_a$，v 声称承认 a。一个节点* **确认** *a 当且仅当它在这样一个群体中。*

节点通过声称 “$accept(a)$” 的接受陈述 a，这是 “一个完好节点接受 a” 的缩略形式的陈述。一个良性行为的节点 v 仅当在接受了 a 之后才能投票赞成 $accept(a)$，这是因为 v 不能假定其它的任何节点都是完好的。如果 v 本身是被污染的，$accept(a)$ 可能会失败，在这种情形投票赞成它可能会牺牲 v 的存活性，然而无论如何一个被污染的节点对存活性是没有任何保证的。

下一个定理表明节点能够依赖被确认陈述而不损失最佳安全性。定理10则说明被确认陈述符合第5.4.2节中认可的定义，这意味着节点可以依赖被确认陈述而不会破坏完好节点的存活性。

定理 9 *设 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 是一个满足除 B 群体可交性的 FBAS，且假定 B 包含了所有的恶性行为节点。令 v_1 和 v_2 是不在 B 中的两个节点。设 a 和 $\bar{a}$ 是相互冲突的陈述。则有，若 v_1 确认了 a，则 v_2 不会确认 $\bar{a}$。*

证明 9 *首先注意到 $accept(a)$ 和 $accept(\bar{a})$ 相互冲突——没有一个良性行为的节点会同时投票赞成两者。更进一步，注意到 v_1 必须批准 $accept(a)$ 以确认 a。由定理5，v_2 不会批准 $accept(\bar{a})$ 因此不会确认 $\bar{a}$。*

定理 10 *如果一个含群体交的 FBAS $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle$ 确认了一个陈述 a，那么不论接下来会发生什么，一旦足够的消息被发送并处理了，每个完好节点将会接受并确认 a。*

证明 10 *令 B 是被污染节点的 $DSet$，并令 $U_a \not\subseteq B$ 是一个群体，一个完好节点通过它确认 a。让在 $U_a \backslash B$ 中的节点广播 $accept(a)$。根据定义，任何节点 v，不论它是如何投票的，在从一个v–阻塞集合中接受了 $accept(a)$ 陈述之后将接受 a。因此，这些消息会说服额外的节点去接受 a。让这些额外节点反过来广播 $accept(a)$ 直到到达这样一个时刻：不论进一步的通信如何，没有更多的完好节点会接受 a。一旦这个过程完成，设 V^+ 是已经接受 a 的所有完好节点集合，设 $V^- = (\boldsymbol{V} \backslash B) \backslash V^+$ 是那些不能接受 a 的完好节点集。为证明所有的完好节点都接受了 a，我们必须证明 $V^- = \emptyset$。*

由定理1，$U_a \backslash B$ 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 中的一个群体。由于对于任何 $v \in V^-$ 来说 V^+ 都不是v–阻塞的，那么由定理7 要么 $V^- = \emptyset$ 要么 V^- 是 $\langle \boldsymbol{V}, \boldsymbol{Q} \rangle^B$ 中的一个群体。后一种情形导致这样一个矛盾：由于 V^- 只包含完好的 (因此也是良性行为的) 节点，它们当中的没有能够在首先真正接受 a 的情况下声明 $accept(a)$，这意味着 $U_a \backslash B \cap V^- = \emptyset$。然而这是不可能的，因为除 B(一个 $DSet$) 群体可交性告诉我们它的反面：$U_a \backslash B \cap V^- \neq \emptyset$。剩下只可能是 $V^- = \emptyset$。一旦每个节点都接受了 a，所有的都会投票赞成确认 a。因为完好节点构成了一个群体，这些投票将会成功。

图11总结了一个完好节点为了确认 a 可以采用的路径。在不知情的状态下，v 可能会赞成 a 或者与之冲突的 $\bar{a}$。如果 v 投票赞成 $\bar{a}$，它之后不会再赞成 a；但是如果一个v–阻塞集合接受了 a 那么 v 却可以接受 a。接下来确认消息的群体允许 v 去确认 a，由定理10这意味着系统认可 a。

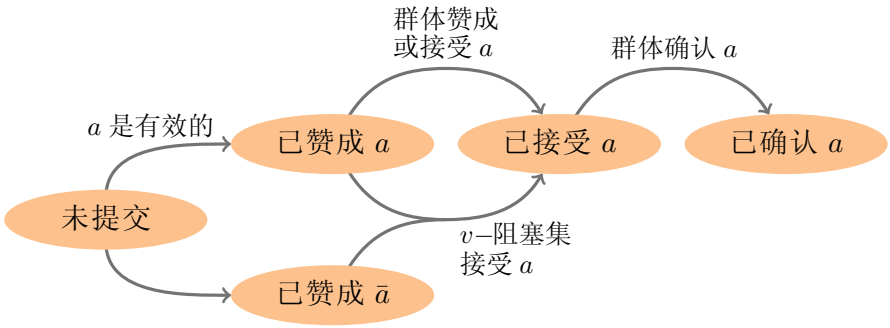

图 11: 一个被接受的陈述 a 在节点 v 处可能的状态

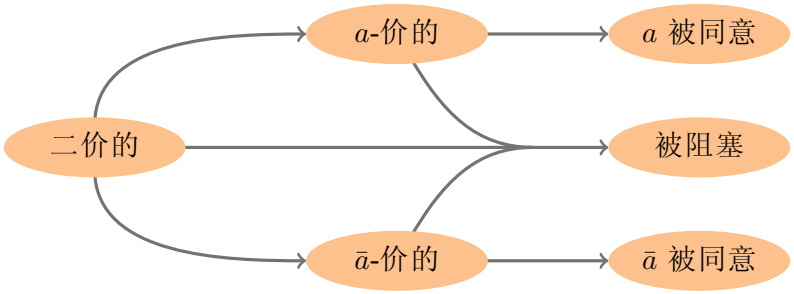

图 12: 陈述 a 在系统范围内可能的状态

5.6 存活性和中和

不论中心化与否，分布式共识的主要挑战是：在系统对它的一致性认可之前，一个陈述可能卡在某个长期不确定的状态中。因此，一个协议不会尝试直接去批准外部表现的值≪TODO 语句不通 ≫。一旦 “存储单元i 处的值是 x” 这一陈述被阻塞了，系统将会永久性的地不能够在存储单元i 处达成一致，从而失去存活性。解决方案是仔细地巧妙修改投票中的陈述。在我们真正关心的问题上 (即关于存储单元内容) 切断一个被阻塞陈述必然是可能的。我们称废弃一个被阻塞的陈述的过程为中和。

更具体地将，图12描述了一个系统范围内一个陈述 a 潜在可能的状态。最初，系统是二价的。我们用它来说明：存在一个可能的事件序列，所有的完好节点会接受 a；而存在另一个序列，所有的完好节点会拒绝 a(即接受和 a 相冲突的陈述 $\bar{a}$)。在某些时刻，两种输出的其中一个可能会不再可能。如果没有完好节点可以拒绝 a，我们称它是 a–价的；反之，如果没有节点可以接受 a，我们称它是 $\bar{a}$–价的。

在一个 FBAS 系统从二价态转化成 a–价时，存在一种可能的结果：所有的完好节点都接受了 a。然而这种情形并不总是成立。考虑像 PBFT 这样的四节点系统 $\{v_1, \ldots, v_4\}$，其中任意三个节点构成了一个群体。如果 v_1 和 v_2 赞成 a，系统变成了 a–价的；没有哪三个节点可以批准一个相冲突的陈述。然而，如果 v_3 和 v_4 最终投票赞成了和 a 冲突的 $\bar{a}$，它也不可能批准 a。在这种情形，a 的状态是永远不确定的，或称之为被卡住的。

正如在图10a 中所看到的那样，甚至一旦一个完好节点接受 a，系统可能还是会在达到系统范围内对 a 的认可上失败。然而，根据定理10，一旦一个完好节点确认 a，所有的完好节点最终都会接受它；因此系统对 a 认可。图13总结了完好节点从它们自己关于一个陈述的局部状态可知的全局状态。

局部状态	a 系统范围的状态
未提交	未知 (对任何节点)
已赞成 a	未知 (对任何节点)
已赞成 $\bar{a}$	未知 (对任何节点)
已接受 a	被阻塞, a-价的, or a 被同意
已确认 a	a 被同意

图 13: 对于陈述 a 的状态一个完好节点所知道的信息

为了保持共识的可能性，一个协议必须确保每个陈述是不可辩驳因而不会被卡住，或者是可中和的因而如果被卡住不会阻塞进展。有两种流行的处理可中和陈述的方法：基于视图的方法，由视图采样副本 [37] 最先提出并得到 PBFT 系统 [20] 的青睐；基于表决的方法，由 Paxos [28] 发明。基于表决的方法可能更难以理解 [38]。更为混乱的是，人们经常称基于视图复制的方法为 “Paxos 算法” 或者断言两种算法是相同的而事实上它们不是 [40]。

基于视图的协议将投票中的存储单元和单调递增的视图数联系相关联。万一在视图 n 处共识卡在了第 i 个存储单元上，节点通过认可视图 n 包含少于 i 个有意义的存储单元并且转向一个大一点的视图数。基于表决的方法将投票中的值和单调递增的表决数相关联。万一一个表决被卡住了，节点用一个更大的表决数重新尝试，仔细选择和之前任何的不确定表决相冲突的值。

本文的工作采用了基于表决的方法，因为这样做更加简化了消除显著的基础节点 (或称领导节点) 的过程。例如存在赶上领导节点的行为 [30] 的可能。

6 SCP：一种联邦拜占庭一致性协议

本节给出了恒星共识协议 (Stellar Consensus Protocol，SCP)。高层次上来讲，SCP 包含两个子协议：提名协议和表决协议。表决协议产生用于存储单元的候选值。如果运行时间足够长，它将在每个完好节点处产生相同的候选值集合，这意味着节点可以用一种确定性的方式为存储单元产生合成值。然而这里有两大注意事项。首先，节点无法知道提名协议何时达到收敛点。其次，即使在收敛之后，恶性行为节点或许有能力多次重置提名过程。

当节点猜测到提名协议已经收敛后，它们执行表决协议，这采用了联邦选举来提交或终止与合成值相关联的表决。当节点同意提交一个表决时，将会为讨论中的存储单元具体化和表决相关联的值。当它们同意终止一个表决时，表决的值变得不再相关。如果一个表决在某个状态下被卡住了 (在这种情形下一个或多个完好节点不能够提交或终止该表决)，那么节点尝试重新使用更高的表决；它们将新的表决和被卡住的表决一样的值相关联，以防止任何节点误认为被卡住的表决被提交了。直观上来说，安全性来源于确保所有被卡住的或已提交的表决和相同的值相关联。存活性由这一事实保证：一个被卡住的表决可以通过转向更高的表决来中和。

本节的剩余部分给出了提名协议和表决协议。它们各自先以概念性语句描述，然后用带有表示一系列概念性语句的消息的具体协议描述。最终，第6.3说明了协议的正确性。SCP 认为每个存储单元是完全独立的且可被看成是一个单一存储单元共识协议中的许多分开的实例 (和 Paxos 算法 [28] 中的 “单一法令议会”(single-decree synod) 类似)。尽管我们的大多讨论隐式地表述存储单元，我们必须总在一个特定的存储单元的环境下解释例如候选值和表决等概念。

6.1 提名协议

因为仅仅要求存储单元是偏序的，一些 SCP 的应用讲对每个存储单元来说将只有一个貌似可信的表决。例如，在认证透明性中，每个中心认证机构可能有一系列它自己的存储单元并且对每个存储单元只签署唯一一份认证树。然而，其它一些应用承认一个存储单元有多个貌似可信的值，这种情形对减少可能的输入值是有帮助的。我们的策略是从一个基于某种超时假设的可以获得共识的同步提名协议出发，并把提名协议的输出运用到一个安全性不依赖于时间的异步表决协议中 [29]。这样一个初始同步的阶段有时候被称为调解者 (conciliator，[15])。

提名协议通过 “关于一个存储单元的候选值趋于相同” 得以运作。之后节点确定性地将这些候选值组合起来成存储单元的一个单独合成值。具体如何组合这些值取决于应用场景。举例而言，恒星网络使用 SCP 为每个存储单元选择了一个事务集合和一个分类帐时间戳。为了组合这些候选值，恒星网络选择了这些事务集的并以及它们的时间戳的最大值。(含有无效的时间戳的值不会接受足够多的提名，从而不会变成候选者。) 其它可能的方法包括，通过取集合交来组合集合，或者简单地选择那些有最大哈希值的候选者。

节点通过对陈述 $nominate\ x$ 进行联邦选举产生出一个候选值 x。

定义 6.1 (候选的) *当一个节点 v 确认了关于值 x 的陈述 $nominate\ x$——即 v 批准了 $accept(nominate\ x)$，它则认为 x 是候选的。*

只要节点 v 没有候选值，v 就有可能投票赞成 $nominate\ x$，这里的 x 可以是任何通过了应用层有效性检验 (例如时间戳不能是未来时间点) 的值。事实上，一般而言 v 应当重新提名它可观察到的其它节点提名的任何值，不过会有一些防止候选者激增的限制 (下文将会讨论)。一旦 v 有了候选值之后，它必须停止对任何新的 x 的值的陈述 $nominate\ x$ 进行投票。但是如联邦选举过程规定的那样，它仍然应当接受为新的值产生的 $nominate$ 陈述 (当被v–阻塞集合接受时) 并确认新的 $nominate$ 陈述。

当一个系统有完好节点的时候 (这意味着它已避免了完全的错误) 提名协议有一些属性。特别地，对每个存储单元来说：

1. 完好节点至少可以产生一个候选值。
2. 在某一时刻，可能的候选值集合停止增长。
3. 如果任意一个完好节点认为 x 是一个候选值的话，那么最终所有的完好节点都会认为 x 是一个候选值。

现在考虑提名协议是如何达到这三个属性的。属性1可以满足是因为 $nominate$ 陈述是不可辩驳的。节点从不会投票反对提名某个特别的值，且直到第一个候选值被确认为止完好节点可以提名任何值。只要有任何值通过了应用层的有效性检查，完好节点就可以投票赞成并确认 $nominate\ x$。属性2可以保证，这是因为每个完好节点至少可以确认一个候选值——这会在一个有限时间内发生——不会有任何节点投票赞成新的值。因此，可能会成为候选值的只有那些已经被那些完好节点投票赞成的值。属性3是定理10的直接结果。

如果参与的值组合变少那么提名过程将会更为高效。因此，我们给节点指派一个暂时的优先级，并且如果可能的话让每个节点提名相同值的节点为更高优先级节点。更具体地讲，设 H 是一个加密哈希函数[5]，其值域是一个整数集合 $\{0, \ldots, h_{max} - 1\}$。($H$ 可能是 SHA-256 [9]，在这种情形下 $h_{max} = 2^{256}$。) 令 $G_i(m) = H(i, x_{i-1}, m)$ 是一个特别用于存储单元的哈希函数，这里的 x_{i-1} 是为 i 之前的存储单元选用的值 (或者当存储单元存在偏序时存储单元i 直接依赖的值的有序集合)。给定一个存储单元i 和一个轮计数值 n，每个节点按照如下公式计算它的*邻居*以及为它的每个邻居计算*优先级*。

$$weight(v, v') = \frac{|\{q | q \in \boldsymbol{Q}(v) \wedge v' \in q\}|}{|\boldsymbol{Q}(v)|}$$

$$neighbors(v, n) = \{v' | G_i(N, n, v') < h_{max} \cdot weight(v, v')\}$$

$$priority(n, v') = G_i(P, n, v')$$

N 和 P 是生成两个不同哈希函数的常量。函数 $weight(v, v')$ 返回 $\boldsymbol{Q}$ 中包含 v' 的切片数占总切片数的百分比。通过考量 v' 的权重作为概率的方式作用于 n 产生邻居 $neighbors(v, n)$，我们也减少了不具很多信任的节点仍然可以支配某一轮的可能性。

每个节点 v 最初应当找到一个节点 $v_0 \in neighbors(v, 0)$ 使得它可以在它可以通信的节点间最大化 $priority(0, v_0)$，然后投票赞成 $nominate$ 和 v_0 相同的值。只有当 $v = v_0$ 时 v 应当引入一个新的值来提名。v 应当使用超时来决定新的用于投票赞成的 $nominate$ 陈述。n 次超时后，v 应当找到一个节点 $v_n \in neighbors(v, n)$ 来最大化 $priority(n, v_n)$，且提名每个 v_n 所提名的值。

定理 11 *最终所有的完好节点将有相同的合成值。*

证明 11 *这一定理的证明可由提名协议的三个性质而得。每个完好节点只会提名一个有限的表决。在没有恶性行为节点的情形下，完好节点将会在一个相同的候选值集合 Z 上收敛。为了阻止该收敛，恶性行为节点可能会引入新的候选值，在某个时间点上它可能是某些但不是所有的完好节点的候选者。这些候选者将需要获得从完好节点那里的选票，然而这使得它们是一个有限集合。最终恶性行为节点或者会停止扰乱系统，或者用尽它们注入的新候选值，在这两种情形下完好节点会在 Z 上收敛。*

6.1.1 具体的提名协议

图14列出了一个节点 v 必须为每个存储单元维护的提名协议状态。X 是 v 已投票赞成 $nominate\ x$ 的值 x 的集合，Y 是 v 已接受 $nominate\ x$ 的值集合，Z 是候选值集合——即含 v 群体已经声明 $accept(nominate\ x)$ 的所有值。最后，v 维护 N——来自每个节点的最新值。(技术上来讲，X，Y 和 Z 可以从 N 的值中重新算出来，但能直接引用它们的值是方便的。) 所有的这些域最初都被初始化为空集。注意到 X，Y 和 Z 都总随时间的推移而增长——节点不会从这些集合中删除元素。

图15说明了一个包含提名协议的具体消息。由于 X 和 Y 随时间单调递增，决定在不考虑网络发送顺序的情况下来自相同节点的多个 NOMINATE 消息哪个是最新的是可能的——只要 D 不改变中间提名 (mid-nomination；否则必须记录 D 的版本信息)。只有一个远程过程调用 (RPC) 对提名来说是必要的——参数是发送者的最新 NOMINATE 消息而返回值是接受者的 (最新 NOMINATE 消

[5]译注：加密哈希函数需要保证下面两者在计算上不可行：1) 找到被加密消息 (原像)；2) 找到和原像不同但有相同哈希值的消息。常见的算法有 MD5，SHA-1，SHA-2，SHA256 等。

变量	含义
X	节点 v 已提名的值集合
Y	节点 v 已承认是为已提名的值集合
Z	节点 v 认为是候选值的集合
N	接收自每个节点的最新的提名类消息 each node

图 14: 节点 v 为每个存储单元维护的提名状态

提名 $v\ i\ X\ Y\ D$

这是一个从节点 v 为存储单元 i 提名值的消息。D 是 v 的群体切片 $\boldsymbol{Q}(v)$ 或是 $\boldsymbol{Q}(v)$ 的无碰撞冲突的哈希。X 和 Y 来自 v 的状态。这些具体的消息编码了下面抽象的消息:

- $\{\,\text{nominate } x \mid x \in X\,\}$ (投票赞成 X 中的每个值)
- $\{\,\text{accept}(\text{nominate } x) \mid x \in Y\,\}$

图 15: 提名协议中的消息

息≪TODO 可能是 accept 消息,不确定 ≫)。如果 D 或被提名值是加密哈希值,必要时第二个 RPC 应当允许获取没有缓存的哈希原像。

由于节点无法通过任何手段知道提名协议何时完成,SCP 必须在不同的节点处处理不同的合成值。则有这样一种优化,节点可在它们有候选值之前尝试预测最终的合成值。为此,合成值可以在 $Z \neq \emptyset$ 时设为 $combine(Z)$,否则在 $Y \neq \emptyset$ 时设为 $combine(Z)$,否则在 $X \neq \emptyset$ 时设为 $combine(X)$。这意味着高优先级节点可以在提名的同时初始化表决,在它的第一个 NOMINATE 消息上加上第一个表决消息 PREPARE(表述参见下文)。

6.2 表决协议

尽管提名过程会继续更新合成值,一旦节点有了合成值它们将参与表决协议。一个表决 b 是一个形如 $b = \langle n, x\rangle$ 的二元组,这里 $x \neq \perp$ 是一个值,而 b 是对当前存储单元具体化的公决 (referendum)。$n \geq 1$ 是一个用于确保大的表决数总可访问的计数器。我们使用类 C 语言的标记 $b.n$ 和 $b.x$ 来表示表决 b 的计数和合成值的域,因而有 $b = \langle b.n, b.x\rangle$。表决是全序的,且 $b.n$ 比 $b.x$ 更为重要[6]。为了方便起见,一个特殊的无效空表决 $\mathbf{0} = \langle 0, \perp\rangle$ 小于其它任何表决,而一个特别的计数器 ∞ 大于其它所有计数器。

我们分别用 “提交表决 b” 或 “终止表决 b” 分别代称使用联邦选举来对语句 $commib\ b$ 和 $abort\ b$ 进行认可。对于给定的表决,$commit$ 和 $abort$ 是相互冲突的,因此一个良性行为的节点最多投票赞成两者之一。在第5节的标注系统下,$commit\ b$ 的反是 $\overline{commit\ b}$,但使用 $abort\ b$ 更为直观。

由于对某个存储单元至多只有一个值被选用,所有提交的和被卡住的表决必须包含相同的值。粗略地说,这意味着如果提交类陈述和更小的非终止表决相冲突的话那么它是无效的。

定义 6.2 (相容的) 两个表决是**相容的** (记作 $b_1 \sim b_2$) 当且仅当 $b_1.x = b_2.x$; 它们是**不相容的** (记作 $b_1 \not\sim b_2$) 当且仅当 $b_1.x \neq b_2.x$。我们还将 $b_1 \leq b_2$(或等价地,$b_2 \geq b_1$) 且 $b_1 \sim b_2$ 记作 $b_1 \lesssim b_2 (b_2 \gtrsim b_1)$。类似地,$b_1 \lneq b_2$ 或 $b_2 \gtrsim b_1$ 意味着 $b_1 \leq b_2$(或等价地 $b_2 \geq 1$) 且 $b_1 \not\sim b_2$。

定义 6.3 (就绪的) 一个表决 b 是**就绪的**当且仅当下面集合中的每个陈述都为真:$\{abort\ b_{old} | b_{old} \lesssim b\}$。

更准确地说,如果 b 被确认是就绪的,那么 $commit\ b$ 对投赞成票来说是有效的,节点通过对应的 $abort$ 陈述的联邦选举保障它。全体一致地对这些陈述进行投票是方便的,因此不论我们在哪里认定≪TODO 使用准确表述 ≫“b 就绪”,周围的环境将应用于 $abort$ 陈述的整个集合中。特别地,一个节点投票赞成、接受或确认 b 就绪当且仅当它分别投票赞成、接受或确认它们全部终止了。

为了提交一个表决并向外界展示它的值 $b.x$,SCP 节点首先接受并确认 b 已经就绪,然后接受并确认 $commit\ b$。在第一个完好节点投票赞成 $commit\ b$ 之前,经由联邦投票的准备步骤确保所有完好节

[6]译注:指二元组 $b_1 \preceq b_2$ 当且仅当 1) $b_1.n \leq b_2.n$;或 2) $b_1.n = b_2.n \wedge b_1.x \leq b_2.x$。

变量	含义
φ	当前阶段: *就绪*, *确认*, *具体化*三者之一
b	节点 v 正尝试准备或提交的当前表决 ($b \neq 0$)
p', p	最高的两个被接受为*就绪*的最高表决，满足 $p' \lesssim p$, 这里 $p' = 0$ 或 $p = p' = 0$ (如果没有这样的表决的话)
c, h	处于*就绪*态时: h 是被确认为就绪的最大表决, 如果没有这样的表决则为 0; 如果 $c \neq 0$, 那么 c 和 h 分别是最低和最高的 v 投票提交且未接受终止的表决。 处于*确认*态时: v 所接受提交的最低和最高表决。 处于*具体化*态时: v 所确认提交 **不变量**: 若 $c \neq 0$, 则 $c \lesssim h \lesssim b$.
z	在下一个表决中用到的值。如果 $h = 0$, 那么 $z = \text{combine}(Z)$ (参见图14); 否则按一定概率更新为 $z = h.x$ (参见6.2.2小节).
M	每个节点所能看到的最新表决消息集合

图 16: 每个节点 v 为每个存储单元维护的表决状态

点最终可以确认 b 是就绪的。当一个完好节点 v 接受 *commit* b 时，意味着 $b.x$ 最终将会被选中。然而，正如第5.4.1中所讨论的那样，为了防止 v 被污染，v 必须在作用于它之前确认提交类陈述。

6.2.1 具体的表决协议

图16强调了由每个节点维护的每一存储单元的状态。一个节点 v 存储了：它当前的表决 b; 两个最近的已经认定就绪的且不相容的表决对 (p, p'); 它必须投票提交的 (或在后续阶段需要确认提交的) 最小表决 c(如果存在的话)，对此它还没有接着接受到终止类陈述; 已确认就绪的最高表决 P; 从每个节点 (M) 处接受到的最新消息; 以及状态 φ。表决 b, p, p' 和 P 在同一个阶段里是不减的。另外，如果 $c \neq \mathbf{0}$——意味着 v 可能参与了批准 *commit* c——代码必须确保 $c \lesssim P \lesssim b$。这一不变量保证了节点总是可以投票为当前的表决 b 做好准备。

图17展示了协议消息。注意 $a \vee accept(a)$ 是每个节点需要为一个群体所断言的，使得它们按照接受定义中的第1种方式接受 a。每个节点通过设置 $b \leftarrow \langle 1, combine(Z)\rangle, p \leftarrow \mathbf{0}, p' \leftarrow \mathbf{0}, P \leftarrow \mathbf{0}, c \leftarrow \mathbf{0}$, $M \leftarrow \emptyset$ 及 $\varphi \leftarrow$ PREPARE 的方式初始化存储单元的状态。之后节点在同类间重复地交换消息，发送由 φ 表明的任何消息。一旦给 M 添加了一个新近接受的消息，一个节点 v 按照下面的方式添加它的状态：

1. 如果 $\varphi =$ PREPARE 且接受的信息让 v 接受新表决是就绪的，更新 p 和 p'。之后，如果 $c \neq \mathbf{0}$ 且 $p \gtrsim P$ 或 $p' \gtrsim P$，设置 $c \leftarrow \mathbf{0}$。
2. 如果 $\varphi =$ PREPARE 且 v 确认新表决是就绪的，增加 P。之后，如果 $c = \mathbf{0}, P \geq b$，且 $p \gtrsim P$ 或 $p' \gtrsim P$ 都不成立，则设 $c \leftarrow P$ 且 $b \leftarrow P$(尽管通常 $b = P$ 已经成立)。
3. 如果 $\varphi =$ PREPARE，且 v 接受一个或多个相容表决的提交类消息。设 c 为最小的这类表决，P 设为最大的使得 "v 能够接受所有的 $\{commit\ b' | c \lesssim b' \lesssim P\}$, $b \leftarrow \langle \infty, c.x\rangle$ 和 $\varphi \leftarrow$ CONFIRM" 的表决。
4. 如果 $\varphi =$ CONFIRM，且接受的消息让 v 接受新表决为就绪的，则提升 p 至最高已被接受为就绪态的、且满足 $p \sim c$ 的表决。
5. 如果 $\varphi =$ CONFIRM 且 v 接受更多的相容的提交类消息，提升 p 至最高的 "使得 v 接受所有的 $\{commit\ b' | c \lesssim b' \lesssim P\}$" 的表决。
6. 如果 $\varphi =$ CONFIRM 且 v 对任意 c' 确认 *commit* c'，设置 c 和 P 为最低和最高的这类表决，设 $\varphi \leftarrow$ EXTERNALIZE，具体化 $c.x$ 并结束。

当 $c = \mathbf{0}$ 时，上述协议实施联邦选举来确认 b 已经就绪。一旦 $c \neq \mathbf{0}$，该协议对 *commit* c (实际上是介于 c 和 P 之间的相容的表决) 实施联邦选举。对确认阶段来说，一旦一个良性行为的节点 v 接受了 *commit* c，该节点就不会接受或尝试确认任何满足 $c' \not\sim c$ 的 *commit* c'。因此直观上说，一旦一个提交被确认了，只要节点具有群体交属性那么具体化它的值就是安全的。

准备 $v\ i\ b\ p\ p'\ c.n\ h.x\ D$
这是一个来自 v 的关于存储单元 i 的消息 D 指定了 $\boldsymbol{Q}(v)$。其他域表征 v 的状态。当 $c.n \neq 0$ 时 $c.x$ 和 $h.x$ 略作 $c.x = h.x = b.x$。这一具体消息编码了这样的一类抽象消息，具体如下:

- $\{\,\text{abort } b' \ \vee\ \text{accept}(\text{abort } b') \mid b' \lesssim b\,\}$ (使得 b 进入就绪的投票)
- $\{\,\text{accept}(\text{abort } b') \mid b' \lesssim p\,\}$ (确认 p 处于就绪态的投票)
- $\{\,\text{accept}(\text{abort } b') \mid b' \lesssim p'\,\}$ (确认 p' 处于就绪态的投票)
- $\{\,\text{commit } b' \mid c.n \neq 0\ \wedge\ c \lesssim b' \lesssim h\,\}$ (在 $c \neq 0$ 的条件下提交 $c, \ldots, h$ 的投票)

确认 $v\ i\ b\ p.n\ c.n\ h.n\ D$
在一个提交之后由 v 发送的试图为存储单元i 具体化 $b.x$ 的消息。方便起见，我们也称 $p' = 0$ (接受提交之后 p' 就不再相关)。同样，D 指定了 $\mathrm{Q}(v)$。编码包含:

- 由 “准备 $v\ i\ \langle\infty, b.x\rangle\ p\ 0\ c.n\ \infty\ D$” 蕴含的所有消息
- $\{\,\text{accept}(\text{commit } b') \mid c \lesssim b' \lesssim h\,\}$ (确认提交 $c, \ldots, h$ 的投票)

具体化 $v\ i\ x\ c.n\ h.n\ D$
在 v 确认为存储单元i 提交 $\langle c.n, x\rangle$ 且具体化值 x 之后，这一消息帮助其他节点具体化 x. 这蕴含了 $c = \langle c.n, x\rangle$ 且 $h = \langle n_h, x\rangle$. 方便期间，我们也称 $b = p = h = \langle\infty, x\rangle$，且 $p' = 0$。编码包含:

- “确认 $v\ i\ x\ \infty\ c.n\ \infty\ D$” 蕴含的所有消息
- “确认 $v\ i\ x\ \infty\ c.n\ h.n\ \{\{v\}\}$” 蕴含的所有消息

图 17: SCP 表决协议中的消息

所有来自一个节点的消息在元组 $\langle \varphi, b, p, p', P \rangle$ 的定义之下是全序的，这里 φ 是最重要的域而 P 最不重要。所有的 PREPARE 消息都在 CONFIRM 消息之前，转而对于给定的存储单元来说在单独的 EXTERNALIZE 消息之前。PREPARE 信息显式地包含这四个域，而 CONFIRM 和 EXTERNALIZE 包含图17中所描述的值。这一序关系使得 M 值包含来自每个节点的最新表决而不依赖于时间来排序消息成为可能，这是因为网络环境可能会对消息重新排序。

一些协议的细节需要解释。形如 "$abort\ b' \vee accpet(abort\ b')$" 的由 PREPARE 所蕴含的陈述并没有指明 v 是否赞成或确认 $abort\ b'$。对于接受的定义来说这种区分并不重要。掩盖这种区分使得 v 忘记了旧的它投票提交 (因此不能够投票终止) 的表决——只要它为这些表决接受一个终止类消息的话。

为了确保节点收敛于 P, p 和 p' 都是必需的，这是因为定理10要求节点重新广播它们已经接受的消息。从就绪的定义可知，"一个节点接受为就绪态的、两个不相容的最高表决" 蕴含了 "所有该节点接受为就绪态的表决"。

在 v 发出 ENTERNALIZE 消息的时候它实际上已经接受了一个区间内的提交类消息 $\{commit\ b' | b' \gtrsim c\}$。然而，$v$ 设置 P 来断言 "只有它确认提交的表决是可接受的"，而不是在隐式的 CONFIRM 消息中设置 $P.n = \infty$ 从而对每个 $b' \gtrsim c$ 断言 $accept(commit\ b')$。这样做是足够的，因为一旦一个单独的完好节点确认了 $commit\ c$，定理10告诉我们所有的完好节点也将确认它。把关注点集中在已被确认的表决上有额外的好处：EXTERNALIZE 消息仅断言 v 已经批准的信息，从而使得 $\boldsymbol{Q}(v)$ 不再相关。这意味着一个独立静态的 EXTERNALIZE 消息对未来任意远处的想赶上进度的节点来说都是有用的，即使群体切片与此同时已经改变了很多。

交换表决消息只需要一个 RPC。参数是发送者最新的消息而返回值是接受者最新的消息。对于 NOMINATE，如果 D 或在表决中的值 x 是加密哈希，那么为了取回没有被缓存的哈希原像需要一个单独的 RPC。

6.2.2 表决选择

如果所有的节点都设置 b 为相同的表决，那么在≪TODO 页 xxx≫的第 1-6 步≪TODO xxx≫足够认可表决的值并且具体化它。然而，在表决协议开始时提名协议并不一定已经在每处都产生了相同的值，在这种情形下节点可能会在它们尝试提交第一个表决的时候失败。如果一个表决失败了或花费了太多的时间那么它可能因为没有响应的节点而失败，那么该节点必须增加它的表决计数器而重新用一个更高的表决来尝试。

当转用新的表决时，节点 v 设置 $b_v \leftarrow \langle b_v.n + 1, z \rangle$，这里 z 是由这个决定的：如果 $P \neq \mathbf{0}$，那么 $z = P.x$；否则，z 是合成值 $combine(Z)$，这里 Z 是第6.1.1节提到的候选值。由于 c 总是由 $P = b$ 初始化而来，提高 $P.x$ 的优先级确保在每个节点处有不变量：如果 $c \neq \mathbf{0}$ 那么 $c \lesssim P \lesssim b$。

方便起见，我们用下标来区分属于特别的节点或消息的域。如果 v 是一个节点，那么用 $b_v, p_v, p'_v \ldots$ 表示图16中描述的节点 v 的状态值 $b, p, p' \ldots$。我们还设置 z_v 为上文提到的 v 的下一个表决——即，如果 $P \neq \mathbf{0}$ 那么 z_v 为 $P_v.x$ 否则是 v 的合成值。类似地，令 b_m, p_m, p'_m, P_m, c_m 表示在图17中所描述的由一个网络消息 m 包含的对应的域。

定义 6.4 (自我验证) *称在节点 v 处的消息集合 $S \subseteq M_v$ 是自我验证的，当 v 是 S 中的某个发送者而 S 的发送者集合是一个在消息的 D 域下、声明的群体集合之下的一个群体。*

一个节点应当在 $\varphi =$ PREPARE 且下面条件之一满足的情况下放弃当前的表决 b_v 而转向一个更高的表决：

1. M_v 包含一个来自发送者的$v-$阻塞集合的消息集合 S，使得 $\forall m \in S, b_v.n < b_m.n$ 且要么 $b_m.n = \infty$ 要么 $P_v \lesssim c_m$ (意味着 v 的表决计数器已经落后了)。
2. 自从 M_v 第一次包含一个自我验证的集合 S(满足 $\forall m \in S, b_v.n < b_m.n$) 之后已经超时过期了 (意味着 b_v 可能将不会提交陈述，且 v 希望它已经收到了足够多的消息来选择一个更好的新表决)。

为了确保协议最终能够结束，条件2必须足够长以使得所有良性行为节点能够交换数轮消息。为了在不需要预测网络延迟的情形下达成这个具体的实现，超时值应当设置为随着 $b_v.n$ 的增长而增长。同样需要注意，当节点 v 已经落后时，并不是遍历测试每个计数器，而是可以简单地增加 $b_v.n$ 到使得条件1为假的最小值。

6.3 正确性

一个节点只有在已经许诺确认所有小编号的表决的终止类陈述之后才能担保确认 $commit\ b$ 陈述。因为一个良性行为的节点不能够接受 (因此担保确认) 相冲突的陈述，这意味着对于给定的 $\boldsymbol{V}, \boldsymbol{Q}$，

定理5确保一个良性行为节点集合 S 只要享有除 $\boldsymbol{V}\backslash S$ 群体可交性则不会产生相互冲突的值。如果 $\boldsymbol{V}$ 和 $\boldsymbol{Q}$ 只在存储单元间改变的话那么安全性仍然成立，但如果它们在存储单元中 (mid-slot) 改变呢 (例如应对节点崩溃)? 为了分析在重新配置的情形下的安全性，我们保守地对旧的和新的群体切片集合进行交操作；这反映了这样一个事实：节点可能依据来自不同时期的消息的组合来作出决定。因此很保守地讲，一个节点只有在当前存储单元用到的每个配置下都是完好的我们才说它是完好节点。但是我们可以放松要求：如果一个节点在最近的配置中都是完好的并且在以往的配置中从未接受过来自全部由恶性行为节点的v–阻塞集合发来的消息，那么我们称该节点是完好的。

定理 12 *令 $\langle \boldsymbol{V}_1, \boldsymbol{Q}_1\rangle, \ldots, \langle \boldsymbol{V}_k, \boldsymbol{Q}_k\rangle$ 是 FBAS 在协商某单独存储单元的时候经历过的配置集合。令 $\boldsymbol{V} = \boldsymbol{V}_1 \cup \cdots \cup \boldsymbol{V}_k$ 且 $\boldsymbol{Q}(v) = \{q | \exists j, v \in \boldsymbol{V}_j \wedge q \in \boldsymbol{Q}_j(v)\}$。令 $B \subseteq \boldsymbol{V}$ 是一个集合，满足 B 包含所有已经发送了非法消息的恶性行为节点——尽管 $\boldsymbol{V}\backslash B$ 可能仍然包含崩溃 (不响应) 的节点。假设 $v_1 \notin B$ 具体化了 x_1，而 $v_2 \notin B$ 具体化了 x_2。则如果 $\langle \boldsymbol{V}, \boldsymbol{Q}\rangle^B$ 有群体交，那么 $x_1 = x_2$。*

证明 12 *为了让 v_1 产生具体的 x_1，它必然已经和一个伪群体 $U_1 \subseteq \boldsymbol{V}$ 共同批准了 $accept(commit(\langle n_1, x_1\rangle))$。我们称伪群体是因为 U_1 对任意特殊的 j 来讲可能都不是 $\langle \boldsymbol{V}_j, \boldsymbol{Q}_j\rangle$ 的群体≪TODO 很可能理解错误≫，这是由于批准可能已经涉及包含多个配置下的消息。然而，为了使得批准成功，$\forall v \in U_1, \exists j, \exists q \in \boldsymbol{Q}_j(v)$ 使得 $q \subseteq U_1$。由 $\boldsymbol{Q}$ 的构造方式可知 $q \in \boldsymbol{Q}$。因此 U_1 是 $\langle \boldsymbol{V}, \boldsymbol{Q}\rangle$。类似地，一个伪群体必然已经批准了 $accept(commit\langle n_2, x_2\rangle)$，且 U_2 一定是 $\langle \boldsymbol{V}, \boldsymbol{Q}\rangle$ 的一个群体。根据 $\langle \boldsymbol{V}, \boldsymbol{Q}\rangle^B$ 的群体交，必然存在某个 $v \in \boldsymbol{V}\backslash B$ 使得 $v \in U_1 \cap U_2$。根据假设，这样的 $v \notin B$ 不能声称接受不相容的表决。由于 v 确认接受值为 x_1 和 x_2 的表决提交，那么必然有 $x_1 = x_2$。*

对于节点 v 的存活性，当一个 FBAS 对一个单独节点经历了一系列的重新配置 $\langle \boldsymbol{V}_1, \boldsymbol{Q}_1\rangle, \ldots, \langle \boldsymbol{V}_k, \boldsymbol{Q}_k\rangle$ 时我们需要考虑一些东西。首先，定理12的安全性前提条件必须对 v 以及 v 关心的节点成立，因为违反安全性会破坏定理10中所要求的群体交属性。其次，最新状态下的恶性行为节点集合 $\langle \boldsymbol{V}_k, \boldsymbol{Q}_k\rangle$ 必须不能是v–阻塞的，这是因为这会否定 v 的一个群体而阻止它批准陈述。最后，v 的状态不能够被一个v–阻塞集合 “错误地声称接受 $\langle \boldsymbol{V}_1, \boldsymbol{Q}_1\rangle \cdots \langle \boldsymbol{V}_{k-1}, \boldsymbol{Q}_{k-1}\rangle$ 的中一个陈述” 所污染。总之，我们认为节点 v 在下面的条件成立的情况下是完好的。首先，在恶性行为节点被删除时对当前存储单元的过去配置的并必须有群体交。其次，v 必须在最新的视图 $\langle \boldsymbol{V}_k, \boldsymbol{Q}_k\rangle$ 下依据以往的静态配置标准还是完好的。最后，v 必须从未接受过以往的配置 $\langle \boldsymbol{V}_1, \boldsymbol{Q}_1\rangle \cdots \langle \boldsymbol{V}_{k-1}, \boldsymbol{Q}_{k-1}\rangle$ 中的全由恶性行为节点组成的v–阻塞集合中接受过消息。

定理 13 *在一个有群体交的 FBAS 中，如果一个表决计数器为 $b.n = n \neq \infty$ 的完好节点开始前页中的条件2中的计时器，那么所有的节点能够将它们的表决计数器提升至 n 而不需要任何计时器过期。*

证明 13 *一个节点只有在它看到了来自群体U 的消息时才开始计时，这时每个 U 中的成员都有一个等于或大于 n 的计数值。如果 U 中包含了所有的完好节点，结论已然成立。否则，令 $U' \neq \emptyset$ 是 U 的完好节点子集。根据除被污染 $DSet$ 群体可交性，必存在至少一个完好节点 $v \notin U$ 使得 U' 是v–阻塞的。(否则定理10的证明给出矛盾。)*

如果 $\forall v' \in U', \varphi =$ PREPARE，意味着 $\forall v' \in U', b_{v'}.n \neq \infty$，那么由≪TODO 页 xxx≫的条件1，$v$ 将会简单地提升 b_v 直到 $b_v.n \geq n$。否则，假设 $\exists v' \in U'$ 使得 $\varphi_{v'} =$ PRIVATE。这意味着 v'(或其它完好节点) 确认 $c_{v'}$ 就绪了且接受了 $commit\ c_{v'}$。因此，由定理10，v 也将在它接受到足够多的其它完好节点的消息之后确认 $c_{v'}$ 是就绪的。此外，根据定理8任何完好节点都不会接受任何 $b' \gtrsim c_{v'}$ 的表决为就绪的。这意味着一旦 $P_v \gtrsim c_{v'}$ 成立，它将永远继续成立下去，再次由条件1得 v 将增加 b_v 直到 $b_v.n \geq n$。

既然 $b_v.n \geq n$，如果完好节点中还有一个小的表决计数器那么在前两个段落中提到的过程将至少提升这样节点中的一个，这同样要求没有超时限制。这个过程将会重复直到每个完好节点的表决计数器都大于等于 n。

定理 14 *在一个有群体交且有足够时间供完好节点来交换足够多消息的 FBAS 中，所有的完好节点最终将会产生一个值。*

证明 14 *根据定理14，所有的节点最终将会有候选值的相容集合 Z。假设已经经历了这个时刻并且每个完好节点都有相同的合成值 $z = combine(Z)$。如果没有节点在 $b.x = z$ 不满足的情形下从未确认任何表决 b 是就绪的，那么所有新的完好节点的表决将会有值 z。令 $b = \langle n, z\rangle$ 为任何完好节点中的最高含 z 表决。最终其它节点将会超时而赶上 b，直到一个群体赶上而开始设置新的计时器。如果时间足够长，那么根据定理13其它节点将会赶上而它们将会完成协议，确认 $commit\ b$。*

否则，给定足够长的时间，根据定理10最终完好节点将收敛于相同的 P。这时，如果我们将 $z = combine(Z)$ 替换为 $z = P.x$ 的话那么和前段中一样的参数也会成立。

值得注意的是如果我们去除了 “足够超时限制” 这一条件的话那么定理14不再为真。特别地，存在这样一个时间段，一个完好节点集合已经进行了足够多的投票来确认一个表决就绪了，而此时此刻节点意识到该表决确实确认就绪了。在有足够长的消息延迟的情况下，节点可以潜在地在不同的 $P.x$ 值之间选择——即在任何节点意识到 P 已经确认就绪了之前，每个节点已经前移了并投票确认 $P' \gtrsim P$ 已经就绪。根据著名的不可能性结果 [25]，一些持久性抢占情景是不可避免的。因此，我们最可能希望的是在一个半异步 (partial synchrony) [23] 的假设下的存活性，这正式定理14给我们保证的。

7 局限性

SCP 只有在节点选用了足够多的群体切片的情况下能够保证安全性。第3.2讨论了为何我们能够合理地寄希望于它们可以做到。然而，当安全依赖于用户可配置的参数时，总存在人们把它们设置错了的可能。

甚至当人们正确设置了群体切片且 SCP 确保了安全性 (safety)，安全性本身并未剔除可能在联邦系统中的其它安全 (security) 问题。例如，在一个金融市场上，被广泛信任的节点可能会改变它们在网络中的角色来获取一些信息，这些信息可能会被用于超前交易或者其它不道德的行为。

拜占庭节点可能会在 SCP 的输入端尝试过滤一些交易而另一方面产生正确的输出。如果良性行为的节点接受所有的交易，结合函数取所有交易的并；而由于存在完好节点，那么这种过滤将不会成功地让受害者交易以概率 1 被阻塞而可能表现出延迟。

尽管 SCP 的安全性是最优的，然而它的性能和通信延迟却不是。通常情况下节点之前没有投票提交和当前表决相互冲突的表决，这时减少一轮通信是可能的。早期的某个 SCP 版本是这样做的，但是协议的描述更为复杂。首先，它要求节点缓存并转发由之前失败的节点发送的签名消息。其次，它不能够掩盖在 PREPARE 消息中投票和终止陈述确认的差别。因此，节点不得不发送一个潜在的无上限的列表给它们的终止节点。

不幸的是，如果一个良性行为节点 v 经历过了一个完全恶意的串通好的v–阻塞集合，改变存储单元中切片来适应错误节点对存活性来说是有问题的。好消息是定理12保证了即使任何有除 $\boldsymbol{V}\backslash S$ 群体可交性的 S 包含被污染的成员，其安全性也是可以保障的。坏消息是如果 v 被戏弄来投票赞成一个坏的提交消息，更新 $\boldsymbol{Q}$ 以解除 v 的阻塞可能是不够的。在这种情形下 v 需要否认过去的投票，而它只能以新的节点 $v' \neq v$ 的身份重新加入系统。可能存在一种自动化这种恢复的方法，例如让其它的节点可以辨认出被重命名的节点并自动将切片中的 v 替换成 v'。

FBA 模型要求参与者对时间的连续性。一旦所有的节点同时长期地离开，重启共识将需要中心化的合作以及人工层次的协商。相反地，一个类似比特币的工作量证明系统有可能经受住完全的颠覆但几乎不要人工干预就能继续执行。另一方面，如果节点确实回来了，FBAS 系统可以从任意长的断供期中恢复过来，然而工作量证明系统将可能面临攻击者在断供期在一个分叉上工作的可能性。

一个令人着迷的可能性是通过对改变配置参数投票或者升级应用协议来权衡 SCP 的调节斗争 (tussle) [21] 的方式。一种实现方式是提名更新参数的特殊消息。候选值可以是一个值集合，也可以是一个参数更新集合。这一做法的一大局限性是：一个大小可以否定系统的一个群体但是不可以破坏安全性的恶意节点集合可能导致随意的配置更改 (通过撒谎并将从未被批准的配置改变放到 Y 中)。如何以一种要求全体群体同意但是不危及存活性的方式对参数更改进行投票仍然是一个有待解决的问题。

8 总结

拜占庭一致性已经长期使得分布式系统获得高效的共识、标准化的加密安全性以及设计可信参与者的灵活性。更近一些发生的是，比特币引入了去中心化共识的革命性观念，带来了许多系统和研究的挑战。本文介绍了联邦拜占庭一致性 (FBA)，一种保留传统拜占庭一致性带来的便利且能达成去中心化共识的模型。FBA 和以往的拜占庭共识系统的关键不同在于 FBA 从参与者的个人的信任决策生成群体，使得类似于互联网的有机生长模型成为可能。恒星共识协议 (SCP) 是一种最佳的对抗恶性行为参与者使得系统恢复的构建方法。

9 致谢

Jed McCaleb 激发了这一工作的灵感并提供的反馈、术语使用建议，同时他还帮忙考虑了多种假想情形。Jessica Collier 协助撰写了本文。Stan Polu 完成了 SCP 第一个实现并在过程中提供了非常宝贵的修改、建议、简化和反馈。Jelle van den Hooff 提供了在群体交和联邦投票章节上的重新调整结构的关键思路，同时还包括术语、组织和演讲等其它方面上重要的建议。Nicolas Barry 在他实现这

一协议的时候发现了多处错误，并指出了必要的澄清。Ken Birman, Bekki Bolthouse, Joseph Bonneau, Mike Hamburg, Graydon Hoare, Joyce Kim, Tim Makarios, Mark Moir, Robert Morris, Lucas Ryan, 以及 Katherine Tom 辛勤地校对论文草稿，找出了多处错误以及会引起困惑的地方，同时还提供了有益的建议。Eva Gantz 提供了实用的写作动机和引用文献。Winnie Lim 在图表上提供了指导。Reddit 社区和 Tahoe-LAFS 小组指出了早期版本的 SCP 中审查的脆弱性，带来了增强版的提名协议。最后，作者还感谢整个恒星团队提供的支持、反馈和鼓励。

10 免责声明

Mazières 教授是作为带薪顾问为本文作出贡献的，这并非他在斯坦福大学的职责。

A 符号表

符号	名字	定义
iff		当且仅当
$f : A \to B$	函数	函数 f 将集合 A 中的每个元素映射为集合 B 中的元素
$f(x)$	函数应用	f 作用于参数 x 上的计算值
$\bar{a}$	补	字母上加一横条表明它的对立面，即 $\bar{a}$ 是 a 的反面
$\langle a_1, \ldots, a_n \rangle$	元组	一个包含域值 $a_1, \ldots, a_n$ 的结构 (复合值)
$A \wedge B$	逻辑与	A 和 B 都为真
$A \vee B$	逻辑或	A 和 B 中至少有一个为真
$\exists e, C(e)$	存在	至少存在一个值 e 使得条件 $C(e)$ 为真
$\forall e, C(e)$	所有	对每一个 e，$C(e)$ 为真
$\{a, b, \ldots\}$	集合	集合包含所列元素 $(a, b, \ldots)$
$\{\, e \mid C(e) \,\}$	集合的结构式	使得 $C(e)$ 为真的所有元素 e
$\emptyset$	空集	不含任何元素的集合
$\lvert S \rvert$	势	集合 S 中的元素个数
$e \in S$	元素	元素 e 是集合 S 中的一个成员
$A \subseteq B$	子集	集合 A 中的每个元素也是集合 B 的元素
$A \subsetneq B$	真子集	$A \subseteq B$ 且 $A \neq B$
2^A	幂集	包含集合 A 中元素所有可能组合的集合，即 $2^A = \{\, B \mid B \subseteq A \,\}$
$A \cup B$	并	包含集合 A 或集合 B 中所有元素的集合，即 $A \cup B = \{\, e \mid e \in A \ \vee \ e \in B \,\}$
$A \cap B$	交	包含在集合 A 中且在集合 B 中所有元素的集合，即 $A \cap B = \{\, e \mid e \in A \ \wedge \ e \in B \,\}$
$A \setminus B$	差	包含在集合 A 中而不在 B 中的所有元素的集合，即 $A \setminus B = \{\, e \mid e \in A \ \wedge \ e \notin B \,\}$
$/$	反	否定某个符号的意义，例如 $e \notin A$ 表示 $e \in A$ 为假，而 $\nexists e, C(e)$ 表示不存在这样的 e 使得 $C(e)$ 为真

参考文献

[1] Boatfuls of cash: how do you get money into fragile states? `http://www.theguardian.com/global-development-professionals-network/2015/feb/19/boatfuls-of-cash-how-do-you-get-money-into-fragile-states`. February, 2015.

[2] Certificate transparency. http://tools.ietf.org/html/rfc6962. June, 2013.

[3] Feathercoin hit by massive attack. http://www.coindesk.com/feathercoin-hit-by-massive-attack/. June, 2013.

[4] Fraudulent digital certificates could allow spoofing. https://technet.microsoft.com/en-us/library/security/2798897.aspx. January, 2010.

[5] Maintaining digital certificate security. http://googleonlinesecurity.blogspot.sg/2015/03/maintaining-digital-certificate-security.html. March, 2015.

[6] Making money transfers work for microfinance institutions. http://www.cgap.org/sites/default/files/CGAP-Technical-Guide-Making-Money-Transfers-Work-for-Microfinance-Ins pdf. March, 2008.

[7] Neucoin: the first secure, cost-efficient and decentralized cryptocurrency. http://www.neucoin.org/en/whitepaper/download. March, 2015.

[8] The ripple protocol consensus algorithm. https://ripple.com/files/ripple_consensus_whitepaper.pdf. 2014.

[9] Secure hash standard (shs). http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf. March, 2012.

[10] Slasher: A punitive proof-of-stake algorithm. https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/. January, 2014.

[11] Tendermint: Consensus without mining. http://tendermint.com/docs/tendermint.pdf. 2014.

[12] Terracoin attack over 1.2th confirmed. https://bitcointalk.org/index.php?topic=261986.0. July, 2013.

[13] Why do africans pay the most to send money home? http://www.theguardian.com/global-development/2013/jan/30/africans-pay-most-send-money. January, 2013.

[14] Eduardo A. Alchieri, Alysson Neves Bessani, Joni Silva Fraga, and Fabíola Greve. Byzantine consensus with unknown participants. In Proceedings of the 12th International Conference on Principles of Distributed Systems, OPODIS '08, pages 22–40, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] James Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. In Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '10, pages 460–467, New York, NY, USA, 2010. ACM.

[16] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: Attack Resilient Public-Key Infrastructure. Attack Resilient Public-Key Infrastructure. ACM, New York, New York, USA, November 2014.

[17] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, PODC '83, pages 27–30, New York, NY, USA, 1983. ACM.

[18] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. IEEE Symposium on Security and Privacy, pages 104–121, 2015.

[19] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. J. ACM, 32(4):824–840, October 1985.

[20] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. pages 173–186. USENIX Association, 1999.

[21] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: Defining tomorrow's internet. IEEE/ACM Trans. Netw., 13(3):462–475, June 2005.

[22] John R. Douceur. The sybil attack. In Revised Papers from the First International Workshop on Peer-to-Peer Systems, IPTPS '01, pages 251–260, London, UK, UK, 2002. Springer-Verlag.

[23] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. J. ACM, 35(2):288–323, April 1988.

[24] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

[25] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. J. ACM, 32(2):374–382, April 1985.

[26] Ghassan Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. ACM Conference on Computer and Communications Security, pages 906–917, 2012.

[27] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil D Gligor. Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure. WWW, pages 679–690, 2013.

[28] Leslie Lamport. The part-time parliament. ACM Trans. Comput. Syst., 16(2):133–169, May 1998.

[29] Leslie Lamport. Brief announcement: Leaderless byzantine paxos. In Proceedings of the 25th International Conference on Distributed Computing, DISC'11, pages 141–142, Berlin, Heidelberg, 2011. Springer-Verlag.

[30] Leslie Lamport. Byzantizing paxos by refinement. In Proceedings of the 25th International Conference on Distributed Computing, DISC'11, pages 211–224, Berlin, Heidelberg, 2011. Springer-Verlag.

[31] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. ACM Trans. Program. Lang. Syst., 4(3):382–401, July 1982.

[32] Jinyuan Li and David Maziéres. Beyond one-third faulty replicas in byzantine fault tolerant systems. In Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation, NSDI'07, pages 10–10, Berkeley, CA, USA, 2007. USENIX Association.

[33] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. Coniks: Bringing key transparency to end users. Cryptology ePrint Archive, Report 2014/1004, 2014. `http://eprint.iacr.org/`.

[34] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf`.

[35] William B. Norton. `http://drpeering.net/white-papers/Art-Of-Peering-The-Peering-Playbook.html`. August, 2010.

[36] K.J. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. In Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET, pages 280–285, June 2014.

[37] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC '88, pages 8–17, New York, NY, USA, 1988. ACM.

[38] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

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

[40] R. Van Renesse, N. Schiper, and F.B. Schneider. Vive la différence: Paxos vs. viewstamped replication vs. zab. Dependable and Secure Computing, IEEE Transactions on, 12(4):472–484, July 2015.