arXiv:2305.02498v1 [cs.DC] 4 May 2023

ZLB：容忍多数合谋的区块链

Alejandro Ranchal-Pedrosa *
alejandro@protocol.ai

Vincent Gramoli †
vincent.gramoli@sydney.edu.au

2023年5月5日

摘要

拜占庭共识问题一直是设计安全分布式系统的关键。然而，由于拜占庭进程会任意行动，且一般网络中的消息延迟未知，这一问题尤为困难。

在一般情况下，如果对手控制了系统的三分之一，就无法解决共识问题。然而，尽管对手控制着系统中的少数部分，区块链参与者通常仍能“最终”达成共识。通过容忍暂时的不一致，可以突破这一$_{1\ 3}$ 上限：不同参与者可能会为同一索引选择不同的区块，但最终会一致选择同一个区块。迄今为止，还没有区块链能够容忍攻击者控制系统中的多数部分。

众所周知，一旦有 $n/3$ 个拜占庭进程发生故障，安全性和活性都会面临风险；然而，鲜有研究尝试准确区分导致安全性违例的故障与导致终止性违例的故障。

本文提出了关于共识问题可解性的新下界：在我们称为拜占庭-欺骗-良性故障模型的模型中，将违反安全性的欺骗性故障和违反终止性的良性故障，与更一般的拜占庭故障区分开来。我们证明，如果存在 $n \leq 3t + d + 2q$ 个拜占庭进程、 d 个欺诈进程和 q 个良性进程，且满足 t，就无法解决共识问题。

此外，我们通过提出 Basilic 类共识协议，证明该界是紧的；当 $n > 3t + d + 2q$ 时，这些协议能够解决共识问题。这些协议的区别在于，它们在继续执行之前需要等待从多少个进程接收消息。因此，根据良性故障或欺骗性故障的主导程度，这些协议各自更适合

*Protocol Labs、悉尼大学
†Red-Bellow Network，悉尼大学

根据良性故障或欺骗性故障的主导程度，适用于某些应用场景。

然后，我们基于 Basilic 类，提出零损失区块链（ZLB）。这是首个能够容忍对手控制系统一半以上、且其中少于三分之一为拜占庭进程的区块链。ZLB 是一种开放式区块链，结合了可问责拜占庭协议领域的最新理论进展，从而排除无可争议的故障进程。

有趣的是，ZLB 不需要已知的消息延迟上界，却会逐步将故障进程的比例降低到 $1/3$ 以下，并最终达成共识。地理分布式实验表明，ZLB 的性能优于 HōtStuff，并且几乎与可扩展的 Red Belly 区块链一样快，而后者无法容忍 $n/3$ 故障。

1 引言

区块链系统 [59] 承诺在没有中心化机构的情况下追踪资产所有权，因此高度依赖分布式节点就链的下一个索引位置达成一致、共同认可唯一的区块。攻击者可以利用不一致，通过简单地将冲突交易插入相互竞争的区块中来双重支付。

针对这一问题的一些解决方案 [13, 38, 9, 28]通过保证永远不会发生任何不一致（即使是暂时的不一致）来避免分叉。这类解决方案通常采用开放式的许可模型：无许可的客户端可以发起交易，而 n 许可的服务器（或 进程）会将这些交易封装到它们共同认可的区块中。这些解决方案通常假设部分同步 [33] ，或者假设存在一个未知的时间上界，用于限定传递任意消息所需的时间。遗憾的是，众所周知 [62] ，一旦这些进程中有 $1/3$ 遭受拜占庭故障，就无法解决共识问题。更具体地说，在这些区块链中，如果攻击者控制了这些进程的 $1/3$ ，就可以利用不一致来进行双重支付。

经典区块链所推广的其他解决方案 [59, 74, 36, 18], 假设对手仅控制少数进程，通常以算力或质押量来表示。通过接受分叉并容忍暂时的不一致，直到它们最终在“最终”共识中得到解决，系统便能够容忍对手控制超过$1/3$ 但少于$1/2$ 的进程。不幸的是，一旦对手控制了系统的大多数部分，安全性就会遭到破坏。

最近已有研究表明，提供确定性保证的区块链必须解决共识问题 [5]。拜占庭共识问题一直是设计安全分布式系统的关键 [66, 24, 42, 50]。这个问题尤其难以解决，因为拜占庭参与者可以采取任意行为 [46]，而且消息延迟通常无法预测 [33]。在一般情况下，如果拜占庭参与者的数量为$t \geq n/3$ [33]，任何共识协议都会失败，其中 n 是参与者总数。在某些执行过程中，$\lceil n/3 \rceil$ 个拜占庭参与者可以通过停止运行或发送无法理解的消息，阻止共识协议终止。在其他执行过程中，$\lceil n/3 \rceil$ 可以通过发送冲突消息，违反共识协议的一致性属性

通过发送冲突消息。

有趣的是，许多研究致力于通过区分故障类型 [24, 42, 50, 51]，提高封闭网络（例如数据中心）中共识协议的容错能力。一些研究通过容忍更多遗漏故障而非执行故障，突破了 $t < n/3$ 界限 [66, 24]。这些研究天然适用于封闭网络，因为进程受到防火墙保护，免遭入侵：相比被恶意对手破坏，这些进程据推测更可能发生崩溃。从这个意义上说，这些协议更倾向于通过容忍更多故障来保障活性，而非安全性。

遗憾的是，较少有研究致力于探索开放网络（例如区块链）中共识协议的容错能力。在这类环境中，如果参与者能够窃取有价值的资产，他们很可能会造成不一致。鉴于区块链系统中攻击屡见不鲜，这一点令人意外；近期，比特币 Gold 遭受了 $\$70,000$[1] 和 1,800 万美元[2] 的损失，而 Ethereum Classic 遭受了 560 万.美元的损失[3]。相比之下，一些区块链参与者（称为矿工）通常会受到持续监控，以确保他们为所有者带来一定回报，因此终止性较不容易受到阻碍。据我们所知，只有存活但受损（abc）进程 [56]能够刻画违反共识安全性的进程。遗憾的是，abc 进程仅在联盟规模足够大、能够成功造成不一致时才会尝试这样做；而在区块链系统中，这一点无法预测。

我们的结果。为此，我们提出了*拜占庭-欺诈-良性*（BDB）故障模型，引入两种新的进程类型，并根据它们造成的故障来界定。首先，*欺诈性* 进程会发送一些冲突消息（即导致一致性遭到破坏的消息）。其次，*良性* 进程是不会发送任何冲突消息、但会导致无法终止的故障进程。例如，良性进程可能崩溃或发送过期消息，甚至发送矛盾信息，只要其消息不影响一致性即可。这两种故障是共识问题的核心。

我们通过精确探究这两种额外的故障类型（当 $t \geq n/3$ 时，它们会阻止终止性或一致性），给出了拜占庭共识问题可解性的新下界。我们的下界表明，在部分同步模型中 [33]，如果$n \leq 3t + d + 2q$ 存在 t 个拜占庭进程、 d 个欺诈进程和 q 个良性进程，则不存在能够解决共识问题的协议。

此外，我们证明了这一下界是紧的，即提出了能够在 $n > 3t + d + 2q$ 条件下解决共识问题的 Basilic[4] 类协议。Basilic

[1]https://news.bitcoin.com/bitcoin-gold-51-attacked-network-loses-70000-in-double-spends/ [2]https://news.bitcoin.com/bitcoin-gold-hacked-for-18-million/ [3] https://news.bitcoin.com/5-6-million-stolen-as-etc-team-finally-acknowledge-the-51-attack-on-network/[4]“Basilic”这一名称的灵感来源于奥斯曼人用来攻破君士坦丁堡城墙的Basilic大炮。正如这门大炮一样，我们的Basilic 协议提供了一

建立在问责性领域近期进展的基础上，仅在关键消息经过其发送方的经密码学签名后才予以考虑。如果签名有效，接收方会存储这些消息，并推进共识协议的执行。接收方还会基于签名无法伪造这一假设，与其他接收方交叉核对所收到的消息。一旦检测到冲突消息，它们就构成不可否认的欺诈证明，可在继续协议执行前排除故障发送方。得益于这一排除机制，Basilic 满足一项新的性质——主动问责性，该性质保证欺骗进程无法阻止终止性。

Basilic 是一类共识协议，每种协议都由不同的投票阈值 参数化；该阈值表示进程为继续推进而需要从中接收消息的不同进程数量。对于投票阈值 $h \in (n/2, n]$，如果满足 $h \leq n - q - t$，Basilic 就具有终止性；如果满足 $h > \frac{d+t+n}{2}$，则具有一致性。这意味着，只需采用一个阈值（例如 $h = 2n/3$），Basilic 就能容忍故障进程的多种组合：它可以容忍 $t < n/3, q = 0$ 和 $d = 0$；也可以容忍 $t = 0, q < n/3$ 和 $d < n/3$；甚至可以容忍 $t < n/6, q < n/6$ 和 $d < n/6$。应用可以修改此投票阈值，以容忍满足$n > 3t + d + 2q$的任意组合，其中包含 t 个拜占庭进程、 d 个欺骗性进程和 q 个良性进程。

将 Basilic 推广到任意投票阈值 h，使我们能够根据应用需求选择最合适的协议。一方面，如果应用运行在由良性进程主导的封闭网络中（例如数据中心），则应降低阈值以确保终止性。另一方面，如果应用运行在由欺骗进程主导的开放网络中（例如区块链），则应提高阈值以确保一致性。

在介绍 Basilic 之后，我们提出了零损失区块链 （简称ZLB），这是首个能够同时实现以下目标的区块链：在拜占庭对手控制的进程数量少于三分之一的情况下解决共识问题，以及在旨在造成不一致的对手控制超过半数进程的情况下最终解决共识问题，且这两种情况均无需假设同步。ZLB 的关键突破在于，它仅在有限的时间内退回到最终共识，此后便能够再次解决共识问题，直到对手改变其所控制的进程。我们将这一性质称为收敛性。更具体地说，ZLB 能够解决 $t + d < n/3$，同时仅在 $n/3 \leq t + d < 2n/3$ 的有界数量的不利情况下退回到最终共识问题。

为了展示 ZLB 的效率，我们使用比特币交易实现了它，并将其性能与现代区块链系统进行比较。我们表明，在分布于不同大洲的 90 台机器上，ZLB 的性能比启发 Facebook Libra [9], 的 HotStuff [77]状态机复制高出 5.6 倍，并且性能可与近期的 Red Belly 区块链 [28 相媲美。].
我们的实证结果还揭示了一种有趣的现象：

突破拜占庭容错经典界限的工具。

由于消息延迟增加，攻击的影响会随着系统规模扩大而迅速减弱。

此外，我们还基于 ZLB 开发了一个零损失支付应用。该应用保证，由攻击者造成的潜在不一致所导致的财务损失，会由向同一批攻击者收取的存款抵消，以弥补不一致造成的影响。

概要。总而言之，本研究的工作如下：

i）我们提出了新颖的 BDB 故障模型，将其与以往模型进行比较并论证其合理性。ii）我们将拜占庭容错（BFT）共识的经典不可能性界扩展到 BDB 模型。iii）我们引入了 Basilic 类共识协议，并证明它们在 BFT 和 BDB 模型下均具有最优容错能力。iv）我们表明，Basilic 类协议在通信复杂度方面是最优的。v）我们提出了长效区块链（LLB）问题，旨在解决对手能够造成不一致情形下的区块链问题。vi）我们提出了零损失区块链（ZLB），这是首个解决 LLB 问题并采用 Basilic 类协议的区块链。vii）我们构建了 ZLB，并将其性能与最新技术水平进行比较，结果表明，ZLB 比 Facebook 的 Libra 区块链更快，且性能可与近期推出的、但不具问责性的 Red-Belly 区块链相媲美。viii）我们基于 ASMR 构建了一个零损失支付应用，在该应用中，诚实进程或客户端均不会因不一致攻击而损失任何资金。

路线图。第 2 节介绍相关工作。第 3 节介绍我们的 BDB 模型及其他假设。第 4 节给出 BDB 模型中共识的不可能性新界。第 5 节介绍 Basilic 协议类，证明其正确性和复杂度，并证明它还能解决最终共识问题。第 6 节介绍 LLB 问题和 ZLB，展示 ZLB 的正确性证明，并将其与先前工作进行实验评估。第 7 节介绍零损失支付应用，在该应用中，没有诚实进程或客户端会因暂时分歧而损失任何资金。最后，我们在第 8 节总结全文。

2 相关工作

共识。 在 Peer- Review [40] 中，有人提出在分布式系统中引入问责性，尤其是针对 Polygraph [20]中的共识问题。这项工作利用问责性，以新的进程取代欺骗进程。遗憾的是，该方法要求欺骗进程最终停止试图造成不一致。Flexible BFT [56] 提出了一种故障模型和理论结果，用以容忍 $\lceil 2n/3 \rceil - 1$ 存活但受损（abc）进程。其容错能力要求客户端承诺：若要容忍 $\lceil 2n/3 \rceil - 1$ abc故障，就不能容忍单个拜占庭故障；或者，如果客户端决定容忍 $t = \lceil n/3 \rceil - 1$ 拜占庭故障，则不容忍任何 abc 故障。Neu 等人的涨落系统 [60] 在部分同步条件下可用 $t < n/3$ ，并且在同步条件下满足最终性 $t < n/2$。他们近期提出的可用性—问责性两难 [61]也说明了为何需要类似 BDB 的模型。Sheng 等人[65] 描述了多种区块链的取证支持。遗憾的是，这些工作都无法容忍 $q = \lceil \frac{n}{3} \rceil - 1$ 良性故障，甚至无法容忍 $\underline{d} = 1$ 欺骗性故障；也无法容忍 $d = \lceil \frac{n}{3} \rceil - 1$ 甚至 $q = 1$ 良性故障，这是它们不满足主动问责性的直接后果。

Upright [24] 可容忍 $n = 2u + r + 1$ 个故障，其中 u 和 r 分别是执行故障和遗漏故障的数量。Upright 可容忍 $n/3$ 个执行故障，或改为容忍 $n/2$ 个遗漏故障，不及 Basilic 所能容忍的 $q + d < 2n/3$个欺骗性和良性故障，或 $t < n/3$ 个拜占庭故障。Upright 可容忍的执行故障数量也没有超过 BFT 共识的下界。对于最终共识问题，Anceaume 等人 [6] 可容忍 $t < n/2$ 个拜占庭故障，但代价是确定性共识甚至无法容忍 $t = 1$ 个拜占庭故障。如果将初始投票阈值 h_0 设为 $h_0 = \lfloor \frac{n}{2} \rfloor + 1$，我们的 Basilic 类也能容忍这种情况；这属于 Basilic 类的一部分。

据我们所知，得益于主动问责性这一性质，Basilic 是首个能够在 BDB 模型中容忍 $n > 3t + d + 2q$ 的协议。然而，已有研究尝试通过威胁罚没存款、将故障进程移出委员会，或两者兼施，来遏制不当行为。Shamis 等人 [64] 将签名消息存储在专用账本中，以便在进程出现不当行为时对其进行惩罚。Casper [14] 算法会在发生双重投票时施加惩罚，但当 $t < n/3$ 时无法确保终止性。尽管 Tendermint [13] 旨在对进程实施罚没，但它不具备问责性。SUNDR [48] 要求无故障客户端之间进行交叉通信，以检测故障。FairLedger [47] 要求系统同步，才能检测故障进程。Balance [41] 会调整存款金额，以避免抵押过多，但据我们所知，目前没有任何系统实现了这一机制。Polygraph [21] 无需罚没即可解决可问责共识问题。FairLedger [47] 假设系统同步，以便检测故障进程。Sheng 等人 [65] 将取证支持视为使进程能够就其行为向客户端承担责任的能力。我们不考虑他们所展示的这种模型；在该模型中，存在 $2n/3$ 个故障时无法实现问责性。Freitas de Souza 等人 [30] 提供了一种可问责格一致协议的异步实现，该协议会重新配置进程，以实现

检测后达成格一致性。Shamis 等人 [64] 在与我们同期开展的一项研究中，提出将签名消息存储在专用账本中，以便在进程出现不当行为时对其进行惩罚。不过，要保证进展，该方法需要 $2n/3$ 个诚实进程。

区块链。一些研究试图绕过拜占庭故障数量的上限 [46] 以达成一致性。与无许可区块链不同 [59], ，开放式许可区块链试图轮换共识参与者，以便在缺乏完美同步的情况下 [71, 11]应对相互勾结的进程数量不断增加。问责性这一概念最初应用于 PeerReview [40] 中的分布式系统，以及 Polygraph [20], 中的共识协议，但并未用于从不一致中恢复。

故障模型。传统上，封闭式分布式系统认为，遗漏故障（遗漏消息）比执行故障（发送错误消息）更为常见 [24, 42, 51]。Zeno [66] 通过将请求分为弱请求（即可能遭遇重排序的请求）和强请求，实现最终一致性。ZLB 无法建立在 Zeno 之上，因为 Zeno 要求回滚顺序错误的交易，而区块链交易可能产生不可撤销的副作用，例如将货物运送给买家。BFT2F [49] 提供分叉一致性，这迫使对手将正确的客户端限制在同一个分叉中，同时也允许实现问责性。Stewart 等人 [69] 提出了一种与我们的确认阶段类似的最终性工具，但它无法从意见分歧中恢复。Flexible BFT 的 abc 进程 [56] 只有在知道自己能够破坏安全性时才会恶意行事，否则会遵循正确行为。这一假设比我们的欺骗性故障更强。

BAR模型 [3], 及其拜占庭、利他型和理性型分类，源于多个管理域的现实情况，因此比区分 a-b-c 故障更适用于区块链开放网络。各种非区块链系统已进一步细化故障类型，以增强保证。Upright [24] 提出了一种支持 $n = 2u + r + 1$ 故障的系统，其中 u 和 r 分别是执行故障和遗漏故障的数量。该系统可以容忍 $n/3$ 个执行故障，或 $n/2$ 个遗漏故障。Ranchal-Pedrosa等人 [63] 提出了一种博弈论模型和一种共识协议，可容忍最多 $n > \max\left(\frac{3}{2}k + 3t, 2(k + t)\right)$ 其中 t 是拜占庭进程的数量，k 是理性进程的数量；该模型根据理性进程通过在区块链应用中引发不一致所能获得的收益，估算其效用。然而，这一结果以理性进程受这些效用约束为前提，而欺骗进程则始终会试图引发不一致。

一些混合故障模型能够容忍崩溃故障和拜占庭故障，但会阻止拜占庭故障对网络进行分区 [50]。其他模型旨在确保行为良好的法定人数能够及时响应 [51] ，或将崩溃恢复与拜占庭行为相结合，以实现可靠广播 [7]。

3 模型

我们将委员会视为由 $|N| = n$ 个进程组成的集合 $N = \{p_0, ..., p_{n-1}\}$。这些进程在部分同步网络中通信，这意味着在未知的全局稳定时间（GST）之后，通信延迟将保持在一个已知上限 Δ 之内 [33]。进程通过标准的全对全可靠且经认证的通信信道进行通信 [45], ，这意味着消息不会被复制、伪造或丢失，但可能会被重新排序。遵循协议的进程是诚实的。故障进程可能是拜占庭型、欺骗型或良性型，详见第 3.7 节。我们将讨论两个容错界限，分别记为 $t_\ell = \lceil \frac{n}{3} \rceil - 1$ 和 $t_s = \lceil \frac{2n}{3} \rceil - 1$。进程通过私有点对点信道进行通信。

3.1 密码学

我们假设存在一个公钥基础设施（PKI），其中每个参与方都拥有一个公钥和一个私钥，并且所有参与方都知道任意参与方的公钥 [75]。与其他采用这一标准假设 [75, 1], 的协议一样，我们不要求使用撤销列表（我们会将进程移出委员会，但不会从 PKI 中删除其密钥）。我们将 λ 称为安全参数，即密钥的比特数。由于我们的论断和证明需要用到密码学，因此它们成立的概率仅存在 $\epsilon(\lambda)$可忽略概率 [8]之外的例外情况。我们对以安全参数 λ为度量的可忽略函数进行形式化定义，即其渐近下降速度快于任意多项式倒数的函数。形式化而言，如果对于所有 $c > 0$，都存在一个 κ_0，使得对于所有$\kappa > \kappa_0$ [8], 均有 $\epsilon(\kappa) < 1/\kappa^c$，那么函数$\epsilon(\kappa)$就是可忽略函数。

3.2 对手

我们将进程建模为概率多项式时间交互式图灵机（ITM） [52, 16, 15]。进程是由以下协议定义的 ITM：收到传入消息后，进程便会被激活，以执行某些计算、更新其状态、可能生成一些传出消息，然后等待下一次激活。对手 $\mathcal{M}$ 是一个概率 ITM，其运行时间为多项式时间（相对于诚实方生成的消息比特数）。 $\mathcal{M}$ 可以控制网络，以读取或延迟消息，但不能丢弃消息。它还可以接管并腐化一组进程，从而获知这些进程的全部状态（存储的消息、签名等）。它接管这些进程收发所有消息的控制权。此外，它可以即时传递来自诚实进程和用户的消息，而任何诚实进程或用户也可以即时传递它发送的消息。

3.3 发送、接收与传递

消息可以发送和接收，但我们还考虑包含两个函数的广播原语：一个广播函数，允许进程 p_i 通过网络中的多个信道发送

消息；另一个是交付函数，它在广播原语的最后调用，以表示消息接收方已接收并处理该消息。不过，在交付发生之前，可能会进行多次消息交换。由于我们将介绍其中一些广播原语，因此会将协议名称作为前缀加到广播函数和交付函数前，以指代使用该协议广播或交付的消息，例如 AARB 广播、AARB-交付、ABV 广播和 ABV 交付，本文后文将对此进行详细说明。

3.4 共识

由进程委员会执行的协议，若满足以下三项属性，即可解决共识问题：

- **终止性**。每个非故障进程最终都会决定一个值。
- **一致性**。任意两个非故障进程都不会决定不同的值。
- **有效性**。如果所有非故障进程都提出相同的值，就不能决定其他值。

区块链。区块链系统 [59] 是一种分布式系统，它维护由区块组成的序列，其中包含有效的（经密码学签名）且彼此不冲突的交易，用以说明资产如何在账户之间交换。[5]

3.5 拜占庭状态机复制

拜占庭状态机复制（SMR） [17, 44]是一种复制服务，接受客户端发来的确定性命令，并通过共识协议为这些命令确定全序，从而在执行这些命令后，尽管存在*拜占庭* 进程或故障进程，每个诚实进程最终仍会得到相同的状态。共识实例按顺序依次执行，从索引 0 开始。我们将索引为 i 的共识实例称为Γ_i。

传统上，假设诚实进程提出一个值，拜占庭共识问题 [62] 要求每个诚实进程最终决定一个值（共识终止），任意两个诚实进程决定的值都相同（一致性），并且 [57, 31, 28] 决定的值必须是已提议值之一（有效性）。然而，本文考虑拜占庭共识的一种变体（定义 3.1），适用于区块链，其中有效性要求决定的值是所提议值的并集的子集，因此我们能够在每个共识实例中提交更多提议的区块。

[5]请注意，在第 6.4 节中，我们将实现比特币的交易，其中“有效”意味着“不冲突”，因为已请求的交易若其未花费交易输出已被消耗，就不可能有效。

定义 3.1（集合拜占庭共识）。 假设每个诚实进程都提出一个交易集合，集合拜占庭共识 （SBC）问题要求每个进程都决定一个集合，并满足以下性质：

- SBC-终止性：每个诚实进程最终都会决定一个交易集合；
- SBC-一致性：任意两个诚实进程决定的交易集合都相同；
- SBC-有效性：决定的交易集合是从各提议集合的并集中选取的有效交易无冲突集合；
- SBC-非平凡性：如果所有进程都是诚实的，且提议了一个共同的有效无冲突交易集合，那么该集合就是决定的集合。

SBC-终止性和SBC-一致性是许多拜占庭共识定义变体的共同属性；而SBC-有效性指出，只要拜占庭提议者提出的交易有效且无冲突（即从任一账户提取的资产都不超过其余额），这些交易就可能被决定；SBC-非平凡性则是防止那些通过决定预先指定的值来解决问题的平凡算法所必需的。因此，我们认为一个共识实例 Γ_i 输出一个可枚举的决策集合$out(\Gamma_i) = d_i, \quad |d_i| \in \mathbb{N}$，所有 n 进程都会复制该集合。我们将第 i个共识实例 Γ_i 处的状态机复制状态定义为截至第 i个共识实例的所有实例的全部决策。

3.6 问责性

区块链系统中的进程默认不具备问责性，因为它们的故障往往无法被检测到。例如，当某个进程创建分叉时，如果它花费货币的区块链分支之一消失，它就能成功实现双重支付。这自然会使其他进程无法检测到欺诈，也无法追究该进程的不当行为。最近，Poly-graph [20, 21]将可问责共识（定义 3.2）引入为这样一个问题：若发生不一致，则在解决共识的同时 $f < n/3$ 并最终检测出 $f_d \geq n/3$ 故障进程。

定义 3.2（可问责共识）。 可问责共识问题是：（i）当拜占庭故障数量为 $f < n/3$时解决共识；（ii）若两个诚实进程输出不同的决策，则每个诚实进程最终至少输出 $f_d \geq n/3$ 个故障进程。

3.7 拜占庭-欺骗-良性故障模型

本节介绍我们的新型拜占庭—欺骗性—良性（BDB）故障模型所需的形式化定义。

冲突消息。 Basilic 会检测并移除试图造成不一致的故障进程，即使它们未能成功造成不一致也是如此。因此，当进程向不同进程发送不同消息，而原本应向这些进程广播相同消息时，Basilic 必须能够检测到这些进程 [2], 我们将此类消息称为冲突消息。给定一个协议 σ，如果消息或消息集合 msg 由进程 p 发送，并且符合该协议的一次执行 σ_E，那么我们称其符合协议 σ的一次执行, 当且仅当 σ_E 属于所有可能执行的集合，在这些执行中， p 发送了 m，且 p 是诚实进程。此外，如果故障进程 p 发送两条消息 msg、msg'，并且满足以下条件，则称其导致了不一致：存在 σ 的一次执行 σ_E，使得（i）足够多的故障进程分别向互不相交的诚实进程子集发送 msg、msg'（也可能还发送更多消息），每个进程接收一条消息，从而导致不一致；（ii）若不发送 p，则 σ_E 不会导致不一致。若两条消息 msg、msg' 相对于 σ 是冲突的，则：

1. 消息，消息′ 分别在某次执行 σ 中遵循算法 σ_E、$\sigma_{E'}$，2. 不存在某次执行 $\sigma_{E''}$，使得两条消息共同遵循 σ $\sigma_{E''}$，并且 3. 如果将 p 发送的 消息，消息′ 分别发送给互不相交的诚实进程子集中的每个进程，则会导致不一致。

当两条冲突消息合并在一条消息中并由发送方签名时，凭借问责性，它们构成欺诈证明（PoF）。两条冲突消息的一个例子是：故障进程在同一轮次发送两个不同的提案（提议者在每轮次中只能提出一个值）。

我们的冲突消息定义与以往类似概念的不同之处在于，冲突消息允许任何进程 p 验证两条消息是否冲突：诚实进程始终可以仅根据两条冲突消息构造欺诈证明（PoF），但对于所有变异消息 [43], 则p 无法做到，因为还需要获知整个执行；对于由矛盾进程 [22], 发送的消息也无法做到，因为这些消息不一定会导致不一致。

故障模型。 故障进程分为三类，彼此互斥：拜占庭、欺骗性和良性，这就是我们所说的拜占庭-欺骗性-良性 （BDB）故障模型。每个故障进程只属于其中一类。拜占庭、欺骗性和良性进程分别由其可能造成的故障来界定。如果某种故障会导致一致性遭到破坏，那么它就是欺骗性 故障：此类故障会通过发送违反协议的冲突消息，导致两个或更多进程分区产生不一致。我们允许欺骗进程不断发送冲突消息，即使它们未能造成不一致，而是以欺骗行为阻止终止性。由于欺骗进程代表试图破坏一致性的进程，我们还假设，欺骗性故障不会针对在其发送消息时已经终止的协议轮次或阶段发送冲突消息

。欺骗进程可以交替发送冲突消息和遵循协议，但不能以其他任何方式偏离协议。良性 故障是指不会发送冲突消息的任何故障。因此，良性故障仅涵盖可能破坏终止性的故障，例如进程崩溃、发送过期消息等。

与往常一样，拜占庭进程可以任意行动。因此，拜占庭进程可能发生良性故障或欺骗性故障，也可能发生既非欺骗性也非良性的故障。按照这些定义，发送冲突消息后再崩溃的故障，既非良性也非欺骗性。我们分别用 t， d，和 q 表示拜占庭进程、欺骗性进程和良性进程的数量。我们假设对手是静态的，即对手可以在协议开始时选择至多 t个拜占庭进程、 d 个欺骗性进程和 q 个良性进程，且这些选择只有对手知晓。因此，故障进程总数为 $f = t+d+q$。

为了区分良性（分别地，欺骗性）进程与在某次协议执行期间发生良性（分别地，欺骗性）故障的拜占庭进程，我们在 BDB 模型中形式化定义容错。令 $E_\sigma(t,\ d,\ q)$ 表示协议 σ 在至多存在 t 个拜占庭进程、 d 个欺骗性进程和 q 个良性进程的情况下，所有可能执行的集合。若协议 σ 针对特定问题 P满足：对于所有执行$\sigma_E \in E_\sigma(t,\ d,\ q)$， σ都能解决 P，则称其为$(t,\ d,\ q)$容错协议。我们不严格区分术语，将 $(t,\ d,\ q)$-容错协议 σ 称为能够容忍 $t,\ d$ 和 q 个拜占庭、欺骗性和良性进程的协议。

注意，给定协议 σ，根据定义， $E_\sigma(0, d+k, q) \subset E_\sigma(k, d, q)$。因此，如果 σ 是 (k, d, q)-故障容忍的，那么 σ 就是 $(0, d + k, q)$-故障容忍的，同时也是 $(0, d, q + k)$-故障容忍的。然而，反过来不一定成立：一个 $(0, d + k, q)$-故障容忍的协议 σ 不一定是 (k, d, q)-故障容忍的，因为 $E_\sigma(k, d, q) \nsubseteq E_\sigma(0, d + k, q)$，这是由于拜占庭参与者能够造成比欺骗性或良性参与者更多的故障。

与执行故障和遗漏故障相比，需要注意的是，并非所有执行故障都会导致不一致。例如，有些执行故障会广播无效消息，而这些消息可以被丢弃。在我们的 BDB 模型中，这类故障应归类为良性故障，而非欺骗性故障，因为无效消息永远不会导致不一致，反而可能阻止协议终止（例如，只发送会被丢弃的无效消息）。然而，所有遗漏故障都是良性故障，但反过来并不成立（如前述同一示例所示）。与存活但损坏故障模型 [56],相比，欺骗性故障并不局限于只有在确信不一致会成功时才会导致不一致；相反，我们允许它们不断尝试，即使最终未能成功。这意味着，某个协议可能能够容忍 $d < n/3$ 个 abc故障以及 $q < n/3$ 个良性故障，但不一定能容忍 $d < n/3$ 个欺骗性故障以及 $q < n/3$ 个良性故障。反方向则始终成立。

因此，我们认为 BDB 模型更适合用于共识，因为它根据故障所危及的性质类型（欺骗性故障危及一致性，良性故障危及终止性），明确区分了不同的故障类型，同时不将这些故障的行为限制在它们

确信会导致不一致的情形。我们重申，有效性这一性质的定义仅用于排除共识中的平凡解，即所有进程都决定同一个常量；而且，可以在本地检查该性质是否满足。

4 BDB 模型中的共识不可能性

在本节中，我们通过加入欺骗进程和良性进程，扩展了Dwork等人关于在部分同步条件下解决拜占庭共识问题所需诚实进程数量的不可能性结果 [33]。首先，我们在第 4.1 节证明任何共识协议的委员会规模下界。然后，我们在第 4.2 节证明取决于该协议投票阈值的下界，并在同一节中定义该阈值。

4.1 不可能性界

首先，我们考虑 $t = 0$的情形，即只有欺骗进程和良性进程。特别是，我们在引理4.1中证明，如果某协议能够解决共识问题，那么它至多容忍 $d < n - 2q$ 个欺骗进程和 $q < n/2$ 个良性进程。该证明的直觉与部分同步条件下存在 $t_\ell + 1$ 个拜占庭进程时共识不可能性的经典证明类似。引理4.1将BFT 模型的经典下界 [33], 扩展到了BDB 模型，其容忍的对手比经典下界中的对手更强（例如，造成 $d = t_\ell$ 个欺骗性故障和 $q = t_\ell$ 个良性故障的对手）。通过反证法，我们证明，当故障进程数量超过引理4.1所界定的数量时，在某些执行中，如果要维持有效性，那么所有进程要么无法终止，要么无法满足一致性。

引理4.1。 设有一个协议 σ ，且 σ 对于某些 $t,\ d,\ q > 0$，能在所有执行 $\sigma_E \in E_\sigma\ (t,\ d,\ q)$ 中解决共识 $d + t < n - 2\ (q + t)$ 。

证明。 首先，我们像先前关于遗漏故障 [33]的研究那样，通过反证法证明 $q < n/2$ 。假设 $q \geq n/2,\ d = 0,\ t = 0$ ，并考虑将进程划分为不相交划分 $P,\ Q$ ，使得 P 包含1到q 个进程，而 Q 包含 $n - |P|$。首先，考虑情形A： P 中的所有进程都是良性的，其余进程均诚实，且 Q 中的所有进程都提议值0。于是，根据有效性， Q 中的所有进程都决定值0。接着，考虑情形B： Q 中的所有进程都是良性的，其余进程均诚实，且 P 中的所有进程都提议值1。于是，根据有效性， P 中的所有进程都决定值1。现在考虑情形C：没有进程是良性的， P 中的进程都提议值1，而 Q中的进程都提议值0。对于 P 中的进程，情形C与情形B不可区分；对于 Q 中的进程，情形C与情形A不可区分。这就导致矛盾。

由此可得 $q < n/2$。因此，对于 $n = 2$，且由于 $q < 1$，显然，对于 $d + t \geq 2$ ，不可能解决共识问题。因此，我们只需考虑满足 $n \geq 3$ 的 $d + t \geq n - 2\ (q + t)$ 。我们将通过反证法证明这一点。

考虑将进程划分为三个不相交的分区 P、Q、R，使得 P 和 Q 各包含 1 至 $q+t$ 个进程，而 R 包含 1 至 $d+t$ 个进程。首先考虑以下情形 A： P 和 R 中的进程均为诚实进程，并提议值 0； Q 中的进程则发生良性故障。由此可知， $P \cup R$ 必须在某个时刻 T_A 决定值 0；因为如果它们决定值 1，就会存在一种情形，其中 Q 中的进程均为诚实进程，并且也提议值 0，但来自 Q 中进程的消息直到晚于 T_A 的时刻才送达，而此时 $P \cup R$ 中的进程已经决定值 1。这将违反有效性性质。此外，为满足可容忍 $q+t$ 个良性故障时的终止性，它们也必须决定某个值。

现在考虑情形 B： P 中的进程是良性的，而 R 和Q 中的进程是诚实的，并提出值 1。采用相同的方法， $R \cup Q$ 中的进程在时间 T_B决定值 1。

现在考虑情形 C： P 和 Q 中的进程是诚实的，而R 中的进程是欺骗性的；从 Q 中的进程发送的消息会在大于 $\max(T_A, T_B)$ 的时间由 P 中的进程交付，从 P 中的进程发送给 Q中的进程的消息也是如此。 P 中的进程提出 0， Q中的进程提出 1，而 R 中的进程向 P 中的进程提出 0，向 Q中的进程提出 1。于是，对于 P 中的进程而言，该情形与情形 A 完全相同，因此决定 0；而对于 Q 中的进程而言，该情形与情形 B 完全相同，因此决定 1，从而导致不一致。这就产生了矛盾。

□

推论 4.1 （关于 $t=0$条件下共识的不可能性）。 如果 $d \geq n-2q$或 $q \geq n/2$，则不可能存在一个共识协议 σ，能够容忍 d个欺骗性进程和 q个良性进程。

证明。 根据引理4.1可立即得出，因为 σ对于所有执行 $\sigma_E \in E_\sigma(0,d,q)$，若 σ解决 P，则 σ是$(0,d,q)$-容错的。 □

我们通过扩展推论 4.1 的结果来证明定理 4.1 的不可能性结果：除非 $n > 3t+d+2q$，否则无法在存在 t 拜占庭、q 良性和 d 欺诈进程的情况下解决共识。

定理 4.1 （共识的不可能性）。 如果 $n \leq 3t+d+2q$，那么共识协议不可能容忍 t 个拜占庭、 d 个欺骗性和 q 个良性进程。

证明。 根据引理4.1，这显然成立，因为如果 σ 对所有执行 $\sigma_E \in E_\sigma$（t， d， q）都能解决 P，那么 σ 就是 (t， d， q)-容错的。 □

4.2 每个投票阈值的不可能性界

第 4.1 节（以及经典的不可能性结果 [33]）中不可能性结果的证明，推导出了一致性与终止性之间的权衡。在某些情形下，进程必须能够在不接收可能发生良性故障的若干进程所发送的消息的情况下终止。在另一些情形下，进程必须在终止前能够接收足够多进程发送的消息，以确保不会发生由欺骗性

故障导致的不一致。本节将证明依赖于这种权衡的不可能性结果。

因此，在部分同步条件下同时满足一致性和终止性的协议，必须规定一个阈值，表示需要从多少个进程接收消息才能在不损害一致性的情况下实现终止。如果该阈值过小而无法满足一致性，或过大而无法满足终止性，则该协议无法解决共识问题。我们将此阈值称为投票阈值，并以 h 表示。通常，该阈值为 $h = n - t_\ell = \lceil \frac{2n}{3} \rceil$，以容忍 $t_\ell = \lceil \frac{n}{3} \rceil - 1$ 拜占庭故障 [26, 21, 44, 77]。然而，我们在引理 4.2 和推论 4.2 中证明，$h > \frac{-d+t+n}{2}$ 当 $h \in (n/2,\ \overline{n}]$ 时可保证安全性。

引理 4.2（一致性的不可能性（$t = 0$））。设 σ 为一个投票阈值为 $h \in (n/2,\ n]$ 且满足一致性的协议。则 σ 最多可容忍$d < 2h - n$ 个欺骗进程。

证明。界限 $h \in (n/2, n]$ 可以直接得出：如果 $h \leq n/2$，那么两个不包含任何故障进程的子集可以分别针对不同的值达到阈值（引理4.1）。我们计算在什么情况下可能导致不一致。因此，我们有两个诚实进程的不相交划分，使得$|A| + |B| \leq n - d$。假设 A 中的进程和 B 中的进程分别决定了不同的决策 $v_A, v_B, v_A \neq v_B$。这意味着 $|A|+d \geq h$ 和 $|B|+d \geq h$ 必须同时成立。将二者相加，得到 $|A| + |B| + 2d \geq 2h$，又由于 $|A| + |B| \leq n - d$，因此，要发生不一致，必须满足 $n + d \geq 2h$。这意味着，如果 $h > \frac{n+d}{2}$，那么 d 个欺诈进程就不可能导致不一致。

□

引理 4.2 的证明可以直接扩展到包含拜占庭进程，从而得到推论 4.2。

推论 4.2。设 σ 是一个投票阈值为 $h \in (n/2,\ n]$ 且满足一致性的协议。那么，σ 至多容忍 $d + t < 2h - n$ 个欺骗性进程和拜占庭进程。

接下来，在引理 4.3 和推论 4.3 中，我们将给出关于终止性质的类似结果。也就是说，我们将证明，若一个协议在 $t = 0$ 时能够实现终止性，那么它至多容忍 $q \leq n-h$个良性进程，或 $q+t \leq n-h$个良性进程和拜占庭进程。

引理 4.3（终止性的不可能性（$t = 0$））。设 σ 是一个投票阈值为 h 且满足终止性的协议。那么，σ 至多容忍 $q \leq n-h$个良性进程。

证明。如果 $n-q < h$，则无法保证终止性，因为在这种情况下，终止需要获得某些良性进程的投票。如果 $h \leq n-q$，这便不可能，因为它保证阈值低于所有进程数减去 q 个良性进程。

□

推论 4.3. 设 σ 是一个投票阈值为 h 且满足终止性的协议。那么，σ最多可容忍 $q+t \leq n-h$个良性进程和拜占庭进程。

结合推论 4.2 和推论 4.3 的结果，可以推导出给定共识协议投票阈值时的不可能性界限。我们在推论 4.4 中给出这一结果。

推论 4.4。设 σ 是一个在投票阈值为 $h \in (n/2, n]$ 时可解决共识问题的协议。那么，σ 最多可容忍 $d + t < 2h - n$ 个拜占庭进程、$q + t \leq n - h$ 个欺诈进程和良性进程。

我们在图 6 中展示了在可容忍 d 个欺骗进程和 q 个良性进程的情况下，阈值 h 的取值。例如，当阈值为 $h = \lceil \frac{5n}{9} \rceil - 1$ 时，安全性条件为 $d < \frac{n}{9}$，活性条件为 $q < \frac{4n}{9}$，其中 $t = 0$。在 $d = q = 0$ 的情况下，可容忍的拜占庭进程数量上限是两个界中的较小值，例如，对于 $h = \lceil \frac{5n}{9} \rceil - 1$，该上限为 $t < \frac{n}{9}$。在本文余下部分中，假设对手在给定特定投票阈值 h 时，满足 $h \leq n - q - t$ 和 $h > \frac{d+t+n}{2}$ 的弹性最优界。定理 4.1 的结果与投票阈值无关。因此，同时满足 $h \leq n - q - t$ 和 $h > \frac{d+t+n}{2}$ 的协议可以将其投票阈值设为 $h \in (n/2, n]$，从而在 $n > 3t + d + 2q$ 成立的前提下，为任意组合的 t 个拜占庭进程、q 个良性进程和 d 个欺骗进程实现共识。

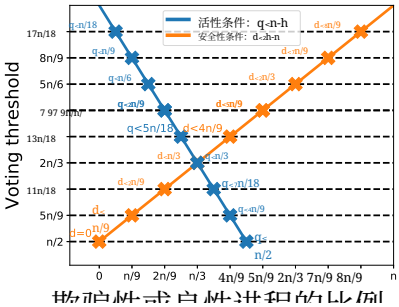


图1：在各投票阈值 h 下，存在 $t = 0$个拜占庭进程时，分别为保证安全性和活性所能容忍的欺骗进程数量 d 和良性进程数量 q。

5 Basilic：BDB 模型中的弹性最优共识

在本节中，我们介绍 Basilic 协议类，这是一类具有弹性且最优的协议，能够在 BDB 模型中针对不同的投票阈值解决主动可问责共识问题。具体而言，Basilic 协议类中的所有协议均可容忍满足 $n > 3t + d + 2q$ 的 t 个拜占庭进程、d 个欺骗性进程和 q 个良性进程；此外，给定该类中由投票阈值 $h \in (n/2, n])$ 唯一确定的特定协议 $\sigma(h)$，则 $\sigma(h)$ 能够容忍数量为 n、满足 $d + t < 2h - n$ 和 $q + t \leq n - h$ 的进程。在本节中，

我们首先需要在第5.1节中介绍几个定义。其次，我们将在第5.2节概述 Basilic 类并展示其组成部分。

5.1 主动可问责共识问题

可问责共识问题 [21] 包含问责性这一性质，以便在欺骗性进程和拜占庭进程设法导致不一致时提供保证。然而，对于 Basilic 而言，这一性质还不够。我们还需要一项额外性质，用于识别并消除所有阻碍终止性的欺骗行为。故障进程可以通过有限数量的冲突消息破坏一致性，但一旦它们发送了一对此类冲突消息，就会留下痕迹，进而可能导致它们被排除出系统。我们的目标是利用这一痕迹，确保欺骗进程无法破坏活性。因此，我们包含主动问责性这一性质，即欺骗性故障不会阻碍协议的终止性。

定义 5.1（主动可问责共识问题）。 具有投票阈值 h 的协议 σ 若满足以下性质，则可解决主动可问责共识问题：

- **终止性**。 每个诚实进程最终都会决定一个值。
- **有效性**。 如果所有诚实进程提议相同的值，则无法决定其他值。
- **一致性**。 如果 $d + t < 2h - n$，那么不会有两个诚实进程决定不同的值。
- **问责性**。如果两个诚实进程输出不一致的决策值，那么所有诚实进程最终都会识别出至少 $2h - n$ 个对该不一致负责的故障进程。
- **主动问责性**。欺骗行为不会妨碍活性。

我们通过纳入投票阈值 h，将先前的问责性定义 [21] 进行了概括。也就是说，先前的问责性定义是我们在本文中针对 $h = 2n/3$ 的标准投票阈值所提出的定义。

5.2 Basilic 内部机制

Basilic 是一类共识协议，所有这些协议都遵循相同的伪代码（算法2–1），但投票阈值各不相同 $h \in (n/2,\ n]$。这些协议的结构遵循从共识问题到二元共识问题的经典归约：前者接受任意有序的输入值集合，后者接受二元输入值。

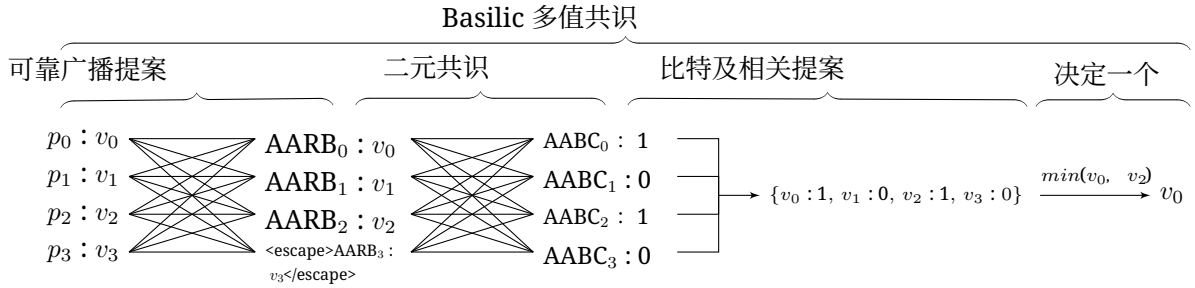


图 2：委员会规模为 $n = 4$ 时 Basilic 的执行示例。首先，每个进程 p_i 选择自己的输入值 v_i，并将其分享给执行各自 $AARB$ 实例 $AARB_i$ 的所有进程。然后，进程执行二元共识协议的一个 AABC$_i$ 实例，以决定是否选择与进程 p_i 关联的输入值。最后，进程在关联的 AABC 实例输出 1 的所有值中，本地处理最小输入值。

5.2.1 Basilic 概述

更具体地说，Basilic 的核心是名为主动可问责二元共识 的二元共识协议，简称 AABC（算法 2–3），其介绍见第5.2.3节。图 2 展示了委员会中有 $n = 4$ 个进程时的一次执行示例。首先，每个进程 p_i 选择自己的输入值 v_i，并通过一个名为主动可问责可靠广播 的可靠广播协议实例（简称 AARB，算法 4）与所有人共享该值。然后，进程针对每个进程 p_i 各执行一个二元共识协议实例 $AABC_i$，以决定是否选择来自进程p_i的对应输入值。最后，各进程在本地处理那些对应的 AABC 实例输出为 1 的值中的最小输入值。

这个 Basilic 二元共识协议与 Polygraph [21],有相似之处，因为它也会检测有罪进程，但它更进一步，会排除这些已检测到的进程，并在运行时调整投票阈值，从而即使在 Polygraph 无法解决共识的情况下也能达成共识（$n/3 \leq t+q+d$）。我们在表 2 中总结了 Basilic 与最新技术水平的比较。与 Polygraph 类似，Basilic 可以执行超级区块优化 [26, 28]，通过在示例中决定两者 v_0 和 v_2 的并集，而不是最小值，从而直接解决 SBC。这使协议的归一化通信复杂度（每次决策）更优。最后，归约的其余部分如算法 1 所示，它会调用 n个主动可问责可靠广播实例，即第 5.2.4 节所述的 AARB（算法 4），随后调用前述 AABC 实例的 n 。

证书与可转移认证。 Basilic 使用证书来验证或丢弃消息，也通过交叉核验证书来检测欺骗进程。证书是先前已传递并签名的消息列表，用于证明证书所附带的消息内容。因此，诚实进程会执行可转移认证 [22]。

也就是说，进程 p_i 可以通过验证消息的签名，从 p_j 传递消息，即使消息 是从 p_k收到的，因为 $k \neq i \neq j$。

检测到的欺骗进程。 Basilic 的一项关键创新是在运行时将检测到的欺骗进程从委员会中移除。因此，我们用 d_r 表示检测到的欺骗进程数量，并定义一个会随检测到的欺骗进程数量变化的投票阈值 $h(d_r)$。因此，进程以初始投票阈值 $h(d_r = 0) = h_0$ 启动 Basilic，例如 $h_0 = \lceil \frac{2n}{3} \rceil$，之后再通过移除检测到的欺骗进程来更新阈值，即 $h(d_r) = h_0 - d_r$。正如我们将要说明的，这样一来，检测到的欺骗进程既不会破坏活性，也不会破坏安全性。证书必须始终包含来自不同进程的 $h(d_r)$ 个签名来证明该消息有效（过滤掉检测到的欺骗进程所提供的最多 d_r 个签名后），否则将被丢弃。回想一下，对手因此受到推论 4.4 中根据投票阈值规定的界限约束。由于 Basilic 使用的阈值会从初始阈值 $h(d_r) = h_0 - d_r$ 开始在运行时更新，我们将这些界限分别改述为适用于初始阈值 $h_0 \leq n - q - t$ 和 $h_0 > \frac{d+t+n}{2}$，或适用于更新后的阈值 $h(d_r) \leq n - q - t - d_r$ 和 $h(d_r) > \frac{d+t+n}{2} - d_r$。

5.2.2 通用 Basilic 协议

我们在算法1中，将 n 个 AABC 二元共识协议实例与 n 个 AARB 可靠广播协议实例结合起来，并展示了通用 Basilic 协议。该协议源自 Polygraph 的通用协议 [19, 21], ，而后者又源自 DBFT 的多值共识协议 [26]。

诚实进程首先启动各自的 AARB 实例（其中每个进程都是源），并在第 2 行提出一个值。交付的提案存储在 $msgs$ 中，索引对应于提案的源。对于每个已记录提案的索引 k，都会以输入值 1 启动索引为 k 的二元共识（第 6 行）。注意，我们可以保证最多对 h（d_r）个提案决定值 1（第 7 行）；其中 d_r 最大可达 d，并由算法 3 中的 update-committee 设置。这意味着，对于标准阈值 h（d_r）$= \lceil \frac{2n}{3} \rceil - d_r$，决定的提案数量最多为 $\lceil \frac{n}{3} \rceil$，因为$d_r < \frac{n}{3}$。一旦诚实进程在至少 h（d_r）个 AABC 实例中决定值 1，诚实进程便会以输入值 0 启动其余 AABC 实例（第 9 行），无需等待各自的值通过 AARB 交付。

最后，当所有 AABC 实例均已终止（第 10 行）后，诚实进程可以输出决策。因此，进程以 AARB 交付的值列表及其对应索引作为输入，并输出一项决策：选择对应索引最低且同一索引的二元共识输出为 1 的 AARB 交付值（第 13 行）。

5.2.3 主动问责的二元共识

我们在算法 2 中展示了初始阈值为 $h_0 \in (n/2,\ n]$ 的 Basilic主动可问责二元共识 (AABC) 协议，以及算法 3 中的一些附加组件和函数。

算法1 带初始阈值 h_0的一般 Basilic。

```
1:  Basilic-gen-propose^{h_0}(v_i):
2:    msgs ← AARB 广播(EST, ⟨v_i, i⟩)                          ▷ Algorithm 4
3:    重复:
4:      如果 (∃v, k : (EST, ⟨v, k⟩) ∈ msgs) 则                  ▷ proposal AARB-delivered
5:        如果 (二进制共识[k] 尚未调用) 则                       ▷ Algorithm 2
6:          bin-decisions[k] ←二进制共识[k].AABC-prop(1)
7:    直到 |bin-decisions[k] = 1| ≥ h(d_r)                      ▷ decide 1 on at least h(d_r)
8:    对所有 k 满足二进制共识[k] 尚未调用 执行
9:      bin-decisions[k] ← 二进制共识[k].AABC-prop(0)
10:   等待直到对所有 k, bin-decisions[k] ≠ ⊥
11:   j ← 最小{k : bin-decisions[k] = 1})
12:   等待直到 ∃v : (EST, ⟨v, j⟩) ∈ msgs
13:   决定 v
```

首先请注意，所有收到的消息都经过正确签名（错误签名的消息会被丢弃），并与所有已发送的消息一同存储在签名消息存储中（详见算法 2 的规则 3）。

Basilic 的 AABC 协议分为两个阶段，之后作出决定。与 Polygraph 的一个关键区别在于，当两个阶段中的某个阶段的计时器达到超时时，如果进程尚不能终止该阶段，那么它会广播该阶段的签名消息集合并重置计时器，具体如规则 4 所述。这使 Basilic 能够防止欺骗进程通过试图造成不一致却始终无法成功来破坏终止性。例如，对于 $n = 4$ 和 $h = \lceil 2n/3 \rceil = 3$，如果 $q = 1$ 和 $d = 1$，欺骗进程可能通过不断向这两个诚实进程发送冲突消息，阻止它们终止，即使这些诚实进程都无法达到不一致值的阈值。因此，一旦计时器到期，进程就会交换其已知的消息集合，并可以更新委员会，移除发送过冲突消息的进程。需要注意的是，进程必须在作出该阶段的决定之前，或在交换签名消息之前等待此计时器，因为只有等待该计时器，才能保证在 GST 之后，所有已发送的消息都会在计时器达到超时之前被接收。每个进程都会维护一个估计值（第 15 行），该值最初由输入给出，随后进入轮次并执行以下阶段:

1. 在第一阶段，每个进程通过可问责二进制值可靠广播（ABV 广播）（第 26 行）广播其估计值（作为输入给出）。我们在算法 3 的第 67–82 行介绍该机制，并在第5.2节.3中进行讨论。除非附有证明其合理性的证书，否则决策消息和 ABV 广播消息都会被丢弃。

该协议还在每一轮中使用轮换协调者（第 23 行），协调者携带一条特殊的 COORD 消息（第 27–29 行）。所有进程都会等待，直到它们至少交付一条来自 abv-broadcast 调用的消息，且计时器（初始设为 Δ）到期（第 30 行）。（请注意，由于 GST 未知，消息延迟的上界仍然未知。）如果某个进程交付了来自协调者的消息（第 33 行），则它会广播一条 ECHO 消息，其中包含 the

Alg算法 2 Basilic 的 AAB　C 初始阈值为 h_0　p_i.

14: AABC-prop$^{h_0}(v_i)$: 15: $est \leftarrow v_i$16: $r \leftarrow 0$17: $timeout \leftarrow 0$18: $cert[0] \leftarrow \emptyset$19: $bin\ \ vals \leftarrow \emptyset$20: 重复： 21: $r \leftarrow r + 1$22:
$timeout \leftarrow \Delta$ ▷ *set timer*23:
$coord \leftarrow ((r-1) mod\, n)\ +1$ ▷ *rotate coordinator*24: ▶ 阶段 1： 25: $timer \leftarrow$start-timer($timeout$) ▷ *start timer*26: abv-broadcast(EST$[r]$, est, $cert[r-1]$, i, $bin\ \ vals$) 27: 如果 ($i = coord$)，则 28: 等待直到 $bin\ \ vals[r] = \{w\}$29: broadcast(COORD$[r]$, w) 30: 等待直到 $bin\ \ vals[r] \neq \emptyset \wedge timer$ 过期 31: ▶ 阶段 2： 32:
$timer \leftarrow timeout$ ▷ *reset timer*33: 如果 ((COORD$[r]$, w) $\in sig\ \ msgs \wedge w \in bin\ \ vals[r]$)，则 34:
$aux \leftarrow \{w\}$ ▷ *prioritize coordinator's value*35: 否则
$aux \leftarrow bin\ \ vals[r]$ ▷ *else use any received value*36: broadcast(ECHO$[r]$, aux) ▷ *broadcast signed* ECHO *message*37: 等待直到 ($vals$ = comp-vals($sig\ \ msgs$, $bin\ \ vals$, aux)) $\neq \emptyset \wedge timer$ 过期 38: ▶ 决策阶段： 39: 如果 ($|vals| = 1$)，则
$est \leftarrow vals[0]$ ▷ *if only one, adopt as estimate*40: 如果 ($est = (r\ mod\ 2)\ \wedge\ p_i$ 之前尚未决定) then 41: decide(est); return$\ est$ ▷ *if parity matches, decide the estimate*42: 否则 $est \leftarrow (r\ mod\ 2)$ ▷ *otherwise, the estimate is the round's parity bit*43: $cert[r] \leftarrow$ compute-cert($vals$, est, r, $bin\ \ vals$, $sig\ \ msgs$) 44: 收到签名消息 $s\ \ msg$时： 45: $pofs \leftarrow$ check-conflicts($\{s\ \ msg\}$, $sig\ \ msgs$) ▷ *returns* $\emptyset$ *or PoFs*46: update-committee($pofs$) ▷ *remove fraudsters*47: 收到证书 $cert\ \ msg$时： 48: $pofs \leftarrow$check-conflicts($cert\ \ msg$, $sig\ \ msgs$) ▷ *returns* $\emptyset$ *or PoFs*49: update-committee($pofs$) ▷ *remove fraudsters*50: 收到欺诈证明列表 $pofs\ \ msg$时： 51: 如果 (verify-pofs($pofs\ \ msg$))，则 ▷ *if proofs are valid then*52: update-committee($pofs\ \ msg$) ▷ *remove fraudsters from committee*53: 规则：1. 发送方未正确签名的每条消息都会被丢弃。2. 第 1 轮之后，abv-broadcast 发送的每条没有有效证书的消息都会被丢弃，但第 2 轮中值为 1 的消息除外。3. 每条收到的签名消息都会存储在 $sig\ \ msgs$ 中，包括证书内的消息。4. 每当计时器达到某个阶段的超时时间，且该阶段无法终止时，进程会广播该阶段当前已交付的签名消息（以及所有收到的未来阶段和轮次的消息），并重置该阶段的计时器。这些消息会加入本地消息集，并在到达时交叉检查欺诈证明。

第二阶段中的协调者值及其签名（第 36 行）。否则，它会将阶段 1 中收到的所有值作为 abv-broadcast 调用的一部分进行回显（第 35 行）。

2. 在第二阶段，进程会等待接收到 $h(d_r)$ ECHO 条消息，如 comp-vals 调用（第 37 行）所示。该调用会返回包含这些 $h(d_r)$ 条已签名 ECHO 消息的值集合。函数 comp-vals 如算法 3（第 83–92 行）所示。随后，进程会在第 39–43 行尝试作出决策。与阶段 1 一样，当阶段 2 的计时器到期时，所有进程都会广播其当前的 ECHO 消息集合。然后，如果进程通过欺诈证明检测到欺骗进程，就会更新其委员会（第 44–52 行），并重新检查是否达到更新后的 $h(d_r)$ 阈值；之后，它们会重置计时器。

3. 在决策阶段，如果 comp-vals 只返回一个值，且该值的奇偶性与当前轮次的奇偶性一致，进程 p_i 就会决定该值（第 41 行），并在 compute-cert 调用中广播相应的证书。如果奇偶性不一致，进程 p_i 则只会将该值作为下一轮的估计值（第 39 行）。如果 comp-vals 返回多个值，p_i 则会将当前轮次的奇偶性作为下一轮的估计值（第 42 行）。当进程在两个值之间犹豫不决时，将奇偶性作为下一轮的估计值有助于下一轮的收敛性。compute-cert 调用（见算法 3 第 93–102 行）会收集用于证明当前估计值合理的签名；如果该估计值在本轮已被决定，则会广播证书。

检测并移除欺诈进程。诚实进程收到一条签名消息后，会调用 check-conflicts（第 45 行）检查该消息是否与存储在 签名消息存储中的先前交付的某条消息冲突。如果没有冲突消息，该函数会返回 $pofs = \emptyset$；否则会返回一份$pofs$欺诈证明列表。随后，在第 46 行，诚实进程调用 update-committee（如算法 3 第 54–66 行所示），在运行时移除 $|pofs|$ 检测出的欺诈进程。在调用 update-committee 时，进程 p_i 会移除所有经新欺诈证明证实为欺诈的进程，并更新委员会 N、其规模 n和投票阈值 h（d_r）。之后，p_i 会重新检查该阶段中所有已交付的消息，以确认在采用新阈值 h（d_r）后是否可以终止该阶段；检查前会过滤掉由 d_r 已移除的欺诈进程 d_r 交付的消息。此操作通过调用算法 3 第 65 行的 recheck-certs-termination() 完成。最后，它调用算法 3 第 66 行的 reset-current-timer()，重置当前阶段的计时器。

Basilic 的 AABC 的终止性和一致性。 我们在引理 5.14 和 5.16 中给出一致性和终止性的详细证明。其思路是，移除欺骗进程不会对一致性产生影响，同时有助于实现终止性，因为阈值 $h(d_r) = h_0 - d_r$ 会随着被移除的欺骗进程数量而降低初始阈值 h_0 。此外，由于所有诚实

算法 3 辅助组件。54：update-committee($new\ pofs$)：▷ *function that removes fraudsters*55：如果（$new\ pofs \neq \emptyset \wedge new\ pofs \not\subseteq local\ pofs$）则 56：
$new\ pofs \leftarrow new\ pofs \backslash local\ pofs$ ▷ *consider only new PoFs*57：
$local\ pofs \leftarrow local\ pofs \cup new\ pofs$ ▷ *store new PoFs*
58 POF $new\ pofs$ ▷ *broadcast new PoFs*：
59 $new\ deceitful \leftarrow new\ pofs.$ ▷ *get deceitful from PoFs*
广播（，）：获取进程（）60：$new\ deceitful \leftarrow new\ deceitful \backslash local\ deceitful$61：
$local\ deceitful \leftarrow local\ deceitful \cup new\ deceitful$62：$N \leftarrow N \backslash \{new\ deceitful\}$；
$n \leftarrow |N|$ ▷ *remove new deceitful*63：
$d_r \leftarrow |local\ deceitful|$ ▷ *update number of detected deceitful*64：$h(d_r)$ ← 重新计算阈值（N，d_r）65：recheck-certs-termination()
▷ *check termination of current phase*66：reset-current-timer() ▷ *reset timer of current phase*67：abv-broadcast(MSG, $val, cert, i, bin\ vals$)：68：广播（BVECHO，$\langle val, cert, i\rangle$）
▷ *broadcast message*69：如果（$r = 3$ 或（$r = 2$ 且 $val = 1$））则丢弃所有未收到有效证书的消息 70：收到（BVECHO，$\langle v, \cdot, j\rangle$）时 71：如果（从 $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ 个进程收到（BVECHO，$\langle v, \cdot, \cdot\rangle$），且 BVECHO、$\langle v, \cdot, i\rangle$ 未广播）则 72：令 $cert$ 为这些消息中收到的任意有效证书 $cert$ 73：广播（BVECHO，$\langle v, cert, i\rangle$）74：如果（从 h（d_r）个进程收到（BVECHO，$\langle v, \cdot, \cdot\rangle$），且（BVREADY，$\langle v, \cdot, \cdot\rangle$）尚未广播）则 75：令 $\overline{cert}$ 为这些消息中收到的任意有效证书 $cert$ 76：构造$bv\ cert$一个证书，其中 h（d_r）由 BVECHO 签名：添加（，$\langle v, cert, j, bv\ cert\rangle$）78：广播（BVREADY，$\langle v, cert, j, bv\ cert\rangle$）79：如果（从 1 个进程收到（BVREADY，$\langle v, cert, j, bv\ cert\rangle$））则 80：$bin\ vals \leftarrow bin\ vals$.添加（BVREADY，$\langle v, cert, j, bv\ cert\rangle$）81：如果（（BVREADY，$\langle v, cert, j, bv\ cert\rangle$）尚未广播）则 82：广播（BVREADY，$\langle val, cert, i, bv\text{-}cert\rangle$）83：comp-vals($msgs, b\ set, aux\ set$)：
▷ *check for termination of phase 2*84：如果 $\exists S \subseteq msgs$，且满足以下条件：85：（i）$|S|$ 包含（h）个不同的 ECHO$[r]$ 消息 86：（ii）$aux\ set$ 等于 S ▷ h（d_r）中的值集合 *with same est*87：则返回（$aux\ set$）88：否则，如果 $\exists S \subseteq msgs$，且满足以下条件：89：（i）$|S|$ 包含（h）个不同的 ECHO$[r]$ 消息 90：（ii）S 中的每个值都属于 $b\ set$ ▷ h（d_r）*messages with different est*91：则返回（$V = S$中的值集合）92：否则返回（$\emptyset$）▷ *else not ready to terminate*93：compute-cert($vals, est, r, bin\ vals, msgs$)：
▷ *compute and send cert*94：如果（$est =$（$r \bmod 2$））则 95：如果（$r > 1$）则 96：$to\ return \leftarrow$（$cert$：（EST$[r]$，$\langle v, cert, \cdot\rangle$）$\in bin\ vals$）97：否则 $to\ return \leftarrow (\emptyset)$ 98：否则 $to\ return \leftarrow$（h（d_r）签名消息，且仅包含 est）99：如果（$vals = \{$（$r \bmod 2$）$\}$ ∧ 没有由 p_i作出的先前决定）则 100：$cert[r] \leftarrow h$（d_r）签名消息，且仅包含 $r \bmod 2$101：广播（est，r，i，$cert[r]$）
▷ *broadcast decision*102：返回（$to\ return$）

进程广播其已交付的欺诈证明；由于账户可问责这一性质，最终所有诚实进程都会对同一组被移除的欺骗进程达成一致。

接着，如果进程 p_i 在广播证书 $cert_i$ 后终止，而另一个进程 p_j 已移除在 $cert_i$中出现的、后来检测到的欺骗进程 $new\ d_r$，那么根据构造，$|cert_i| - new\ d_r \geq h(d_r + new\ d_r)$。因此，要么某个诚实进程终止，此后所有诚实进程都能终止，即使移除了更多欺骗进程；要么诚实进程最终进入这样一种情形：所有欺骗进程都已被检测到 $d_r = d$ 并移除，此后诚实进程便会终止。

请注意，在运行时移除进程可能导致某些轮次的协调者已被移除。为确保正确性，即使该轮次的协调者已被移除，我们也不会更改它。这保证所有诚实进程最终都会进入一个轮次，在该轮次中，它们对同一协调者达成一致，而该协调者是一个诚实进程。如果该轮次是 GST 之后、且所有欺骗进程均已从委员会中移除后的第一个轮次，那么诚实进程将达成一致性。

可问责二值广播。 我们在算法 3 中提出的 ABV 广播受到 Malkhi 等人提出的 E 协议 [55]以及 Polygraph [19, 21]中提出的二值广播的启发。如果诚实进程因 ABV 广播而将值 v 加入 二值集合 （第 77 行和第 80 行），我们称它们对 v执行了 *ABV 交付* 。进程在 ABV 广播期间交换 BVECHO 和 BVREADY消息。 BVECHO 消息经过签名，并且必须附带一个有效证书 $cert_i$ 来证明该值的合理性，如第 68 行和第 73 行所示。BVREADY 消息携带的信息与 BVECHO消息相同，此外还包含一个*bv* 证书，其中包含用于证明BVREADY消息合理性的 $h(d_r)$ BVECHO 消息，并在第 76 行构造。这样，进程一旦收到携带某个值的 BVREADY 消息（第 79 行），就已经获得了 $h(d_r)$ BVECHO 消息，这意味着它可以将该值加入 二值集合，从而对该值执行 ABV 交付（第 77 行和第 80 行）。诚实进程会为自己的估计值（第 68 行）以及所有至少从$\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ 个不同进程收到带签名的BVECHO 消息的值广播带签名的 BVECHO 消息。等待某个值 v 对应的这么多条~~BVE~~CHO 消息， v可确保所有诚实进程都会对其执行 ABV 交付，正如我们在第 5.4 节中所示。

具体而言，我们证明 ABV 广播满足以下性质：(i) ABV 终止性，即每个诚实进程最终都会向二值集合中添加至少一个值；(ii) ABV 一致性，即诚实进程最终会向二值集合中添加相同的值；(iii) ABV 义务性，即如果 $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ 诚实进程对值 v进行 ABV 广播，那么所有诚实进程都会 ABV 交付 v；(iv) ABV 合理性，即如果某个诚实进程 ABV 交付值 v，那么 v 曾由某个诚实进程进行 ABV 广播；(v) ABV 问责性，即每个 ABV 交付的值都包含一个来自上一轮的有效证书。

5.2.4 主动可问责可靠广播

算法 4 展示了 Basilic 的主动可问责可靠广播 (AARB)。该协议与先前工作 [55],中介绍的安全广播类似，不同之处在于，我们还引入了一个计时器，供诚实进程定期广播其已交付 ECHO 消息的集合，以便检测欺骗进程。协议在源广播一条包含其提议值 v 的 INIT消息时启动（第 104 行）。交付该消息后，所有诚实进程也会广播一条带签名的 ECHO 消息，其中包含 v （第 106 行）。随后，当某个进程为同一值 v交付了 p_i 条彼此不同的带签名 ECHO 消息 $h(d_r)$ 时，它会先广播一条 READY 消息（第 109 行），其中包含一份证书，该证书由用于证明 v 合理性的 $h(d_r)$ 条 ECHO 消息构成（在第 108 行构造），然后再 AARB 交付该值（第 110 行）。如果某个进程只交付了一条有效的 READY 消息，且其中包含用于证明该消息合理性的有效证书，则在第 111–115 行也会发生相同的情况。

与算法 2 和 3 中介绍的 Basilic 的 AABC 协议一样，诚实进程会将新收到的签名消息与先前交付的消息进行交叉核对（第 117 行和第 120 行），从而检测出欺骗性故障并更新委员会（第 118 行和第 121 行），通过调用 update-committee 在运行时将其移除。收到欺诈证明列表时也可能发生这种情况（第 122 行）。请注意，这与算法 2 所示 AABC 协议中调用的是同一个函数，因为诚实进程会在整个 Basilic 协议中更新委员会，而不仅仅是在检测到欺诈进程的那个特定 AARB 或 AABC 实例中更新。我们将在第 5.4 节证明，Basilic 的 AARB 协议满足主动可问责可靠广播的以下性质：

- **AARB 唯一性**。 诚实进程最多交付一个值。
- **AARB 有效性**。 如果源先前广播过某个值，诚实进程就会交付该值。

- **AARB 发送**。 如果源是诚实的，并且通过 AARB 广播 v，那么诚实进程会 AARB 交付 v。
- **AARB 接收**。 如果一个诚实进程 AARB 交付 v，那么所有诚实进程都会 AARB 交付 v。
- **AARB 可问责性**。 如果两个诚实进程 AARB 交付了不同的值，那么所有诚实进程都会收到至少 $2h(d_r) - n$ 个进程（包括源）的欺骗行为的欺诈证明。

- **AARB 主动问责性**。欺骗行为不会阻碍活性。

5.3 Basilic 在 BDB 模型中的容错能力

我们在图 3a 中展示了 Basilic 根据初始阈值 h_0所能容忍的拜占庭进程、欺诈进程和良性进程的组合。实线

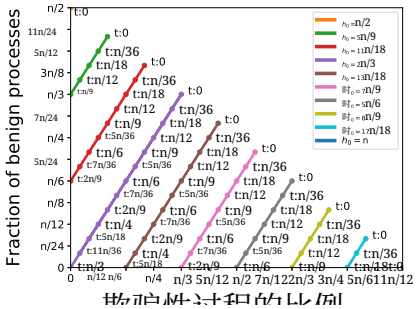
算法 4 Basilic 的 AARB，初始阈值为 h_0。

103: AARB 广播$_{h_0}(v_i)$**:** ▷ *executed by the source*
104: 广播(INIT, v_i) ▷ *broadcast to all*
105: **收到来自** p_j **的**(INIT, v_i) **且尚未发送** ECHO **时:**
106: 广播(ECHO, v, j) ▷ *echo value to all*
107: **收到** $h(d_r)$ (ECHO, v, j) **且尚未发送 a** READY **时:**
108: 构造包含至少 $h(d_r)$ 条已签名消息的 $cert_i$ （ECHO, v, j）
109: 广播(READY, v, $cert_i$, j) ▷ *broadcast certificate*
110: AARB 交付(v, j) ▷ *AARB-deliver value*
111: **收到** (READY, v, $cert$, j) 后，**且尚未发送** READY:
112: **如果** (验证($cert$) $= False$) **，则继续**
113: 将 $cert_i$ 设置为收到的有效证书之一（READY, v, $cert$, j）
114: 广播(READY, v, $cert_i$, j) ▷ *broadcast certificate*
115: AARB 交付(v, j) ▷ *AARB-deliver value*
116: **收到签名消息** s msg **时:**
117: <escape>$pofs \leftarrow$check-conflicts($\{s\ msg\}$, $sig\ msgs$)</escape> ▷ *returns ∅ or PoFs*
118: update-committee($pofs$) ▷ *remove fraudsters*
119: **收到证书** $cert$ msg **时:**
120: $pofs \leftarrow$check-conflicts($cert\ msg$, sig_msgs) ▷ *returns ∅ or PoFs*
121: update-committee($pofs$) ▷ *remove fraudsters*
122: **收到欺诈证明列表**$pofs$ msg**后:**
123: **如果** (verify-pofs($pofs$ msg))**则** ▷ *if proofs are valid then*
124: update-committee($pofs$-msg) ▷ *exclude from committee*
125: **规则:**

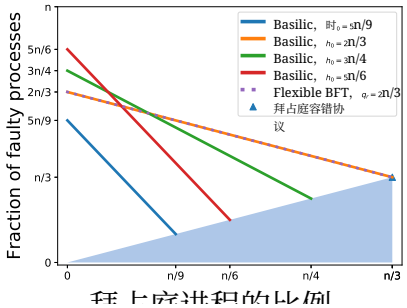
1. 进程在计时器一次
$timer$，初始设为 Δ，达到 0 后，将计时器重置为 Δ。

表示在特定阈值下，随着拜占庭进程数量变化，系统对良性进程和欺诈进程的容忍度变化。例如，对于 $h_0 = \frac{2n}{3}$，如果 $t = 0$，那么 $d < \frac{n}{3}$，且 $q < \frac{n}{3}$。随着 t 增加，例如增至$t = \lceil \frac{n}{6} \rceil - 1$，那么 $d < \frac{n}{6}$，且 $q < \frac{n}{6}$。

我们在图3b中将 Basilic 的容错能力与以往工作的容错能力进行了比较。具体而言，我们展示了 Basilic 的初始阈值$h_0 \in \{5n/9, 2n/3, 3n/4, 5n/6\}$ 的多个取值。首先，我们表明，经典的拜占庭容错（BFT）协议只能容忍图中以蓝色三角形点（ ）表示的情况 $t < n/3$ 。大多数部分同步 BFT 共识协议 [26, 21, 44, 77]都属于这种情况。其次，我们展示了 Flexible BFT [56] 在部分同步条件下、其最大容错能力设置时的情况。可以看到，这种设置与 Basilic 的初始阈值 $h_0 = 2n/3$重叠。然而，区别在于，Basilic 可以容忍实线上 $h_0 = 2n/3$的所有情况，而 Flexible BFT 只能容忍该线上的某个特定点，具体取决于每个客户端的选择。也就是说，Flexible BFT 的客户端必须决定，例如，是容忍 $\lceil 2n/3 \rceil - 1$ 个总故障且其中没有一个是拜占庭故障，还是容忍 $\lceil n/3 \rceil - 1$ 个拜占庭故障且不再容忍任何额外故障。不过，Basilic 可以容忍满足安全性 $h_0 > \frac{n+d+t}{2}$ 和$h_0 \leq n - q - t$活性要求的任意范围，这使得我们的客户端和进程能够针对某个特定阈值 $h_0 \in$ 容忍显著更多的故障组合（$n/2$, n]。因此，我们将 Flexible BFT 的线表示为虚线，而



（a）对于初始阈值 h_0，Basilic 所能容忍的良性、欺骗性和拜占庭进程组合。



（b）对于一般 Basilic 协议的特定初始阈值 h_0，故障进程所占比例与拜占庭进程所占比例的比较，并与其他工作的结果进行比较。

Basilic 的线表示为实线。对于每个初始投票阈值 h_0，Basilic 所能容忍的拜占庭进程最大数量为 $t < \min(2h_0 - n,\ n - h_0)$，该数量通过设置 $q = d = 0$ 并求解安全性和活性的两个界限得到。

5.4 Basilic 的正确性

本节证明 Basilic 的其余性质，包括其 ABV 广播、AABC 和 AARB 协议。

5.4.1 可问责二值广播

我们首先介绍 ABV 广播满足的性质。我们称进程 p_i 对值 v 进行 ABV 广播，是指 p_i 发送一条包含 v 的 BVECHO 消息，并附上证明 v 合理的有效证书。我们在引理 5.1 中证明 ABV 终止性，在引理 5.2 中证明 ABV 一致性，在引理 5.3 中证明 ABV 义务性，在引理 5.4 中证明 ABV 合理性，并在引理 5.5 中证明 ABV 可问责性。

引理 5.1（ABV 终止性）。 每个非故障进程最终都会向二值集合中添加至少一个值。

证明。 请注意，当非故障进程收到带有 v 的 $\lfloor \frac{n-q}{2} \rfloor - d_r + 1$ BVECHO 条消息时，都会广播一条值为 v 的 BVECHO 消息。首先，考虑 $t = d = 0$ 的情况；此时，如果非故障进程收到带有 v 的 $\lfloor \frac{n-q-t}{2} \rfloor + 1$ BVECHO 条消息，就会广播一条带有 v 的 BVECHO 消息。还要记得 $v \in \{0, 1\}$。因此，考虑非故障进程的一个分区 A，$B \subseteq N$，使得 $A \cap B = \emptyset$；并假设 A 中的进程最初发送了一条带有 $v = 0$ 的 BVECHO 消息，而 B 中的进程发送了一条带有 $v = 1$ 的 BVECHO 消息。显然，$|A| + |B| \geq n - q - t$，因此 $|A| \geq \lfloor \frac{n-q-t}{2} \rfloor + 1$ 或 $|B| \geq \lfloor \frac{n-q-t}{2} \rfloor + 1$。不失一般性，假设 $|A| \geq \lfloor \frac{n-q-t}{2} \rfloor + 1$，那么 $|B|$ 中的进程最终会收到足够多的 BVECHO

带有值 $v = 0$ 的消息，也会广播一条带有 $v = 0$ 的 BVECHO 消息。因此，由于 $h(d_r) \leq n - q - t - d_r$，最终所有非故障进程都会收到足够多的 BVECHO 条消息，从而将至少值 0 加入二值集合。

反之，假设 $d > 0$ 和 $t = 0$。那么，如果在某个特定阶段表现出欺骗行为的 $d_r \leq d$ 个欺骗进程足以阻止终止性，这就意味着 d_r 个进程至少向两个非故障进程发送了两条冲突消息。因此，当计时器到期且非故障进程广播其收到的已签名 BVECHO 消息时，所有非故障进程最终都会收到足够多的 BVECHO 消息，从而发送一条 BVECHO 消息。因此，情形 $d > 0$与情形 $d = 0$类似，因为计时器到期时会转发 BVECHO 消息；我们已在前一段证明，在这种情形下能够保证终止性。如果 $t > 0$，也同样可以类比。

此外请注意，如果 d_r 个已检测出的欺骗进程被移除，那么阈值也会按相同的比例 d_r降低，从而保持终止性。

□

引理 5.2（ABV 一致性）。如果一个非故障进程 p_i 将值 v 添加到集合 二值集合 中，那么所有其他非故障进程最终也会将 v 添加到各自的本地集合二值集合中。

证明。该证明十分直接：p_i 在持有$h(d_r)$条由 v签署的消息时，将 v 添加到集合二值集合 中。在这种情况下，它还会利用这些消息构造一个证书***bv*** 证书，并在添加 v 到二值集合之前，将***bv*** 证书 作为 BVREADY的一部分与 v 一起广播。因此，所有其他非故障进程最终都会收到 p_i的 BVREADY 消息，以及包含足够BVECHO 条消息的***bv*** 证书，从而也能将 v 添加到各自的本地二值集合中。最后，回顾一下，所有非故障进程都会在将 v 添加到二值集合之前广播其 BVREADY 消息，这就解决了 p_i 发生故障且仅向部分非故障进程发送 BVREADY 的情况。

□

引理 5.3（ABV 义务性）。如果 $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ 个非故障进程通过 ABV 广播一个值 v，那么所有非故障进程都会 ABV 交付 v。

证明。该证明与引理 5.1 的证明类似。 □

引理 5.4（ABV 合理性）。如果进程 p_i 是非故障进程，并且交付了 ABV 值 v，那么 v 必定已由某个非故障进程进行 ABV 广播。

证明。首先假设 $t = 0$，并假设相反：p_i ABV 交付 v，且所有非故障进程都 ABV 广播 $v', v \neq v'$。由于良性进程可能向部分非故障进程发送 v'，也可能什么都不发送，这意味着仅靠欺骗性进程就能够使p_i ABV 交付 v。但利用界限 $d - d_r < n - h(d_r)$，我们得到$q \geq 2h(d_r) - n$，这与我们对良性故障数量的假设（即界限 $q < 2h(d_r) - n$）相矛盾。因此，至少有一些非故障进程必定 ABV 广播了 v。如果 $t > 0$，证明类似。

□

引理 5.5（ABV 可问责性）。 如果进程 p_i 将值 v 添加到二值集合，那么与 v 相关联的是来自上一轮的有效证书$cert$。

证明。 由于每条没有有效证书的 BVECHO 和 BVREADY 消息都会被丢弃，因此，当值 v 被加入二值集合 时，p_i 可以访问一个有效证书。 -

□

5.4.2 主动问责的可靠广播

在本节中，我们证明 Basilic 的可靠广播 AARB 的性质。我们在引理 5.6 中证明 AARB 唯一性，在引理 5.7 中证明 AARB 有效性，在引理 5.8 中证明 AARB 发送，在引理 5.9 中证明 AARB 接收，在引理 5.10 中证明 AARB 可问责性，并在引理 5.11 中证明 AARB 主动问责性。

引理 5.6 (AARB 唯一性)。 非故障进程最多只能 AARB 交付一个值。

证明。 根据构造，所有非故障进程最多只会 AARB 交付一个值。

□

引理 5.7 (AARB 有效性)。 如果非故障进程 p_i AARB 交付了 v，那么v 曾由 p_s进行 AARB 广播。

证明。 进程 p_i 在接收到 $h(d_r)$ 条消息后，AARB 交付 v。非故障进程只有在收到 $\langle \text{INIT}, v \rangle$ 后才会发送 ECHO 消息（用于 v ）。因此，由于 $d + t < h(d_r)$, p_s 会向至少一个非故障进程进行 AARB 广播 v。

□

引理 5.8（AARB 发送）。 如果 p_s 是非故障的，并且 AARB 广播了 v，那么所有非故障进程最终都会 AARB 交付 v。

证明。 欺骗进程要么广播 v，要么将 v' 组播给分区 A，并将 v 组播给分区 B。在第一种情况下（所有欺骗进程的行为都像非故障进程一样），由于良性进程和拜占庭进程的数量为$q + t \leq n - h(d_r)$，因此至少有 $h(d_r)$ 个非故障进程会回显 v，这足以让所有进程最终 AARB 交付该消息。

反过来考虑这种情况：某些 $d_r \leq d + t$ 欺骗进程通过欺骗性地向两个不同的分区回显不同的消息，而每个分区中至少包含一个非故障进程。随后，当计时器到期且非故障进程交换其已交付的 ECHO消息时，所有进程都会更新其委员会，移除 d_r 已检测到的欺骗进程。因此，由于进程还会在更新委员会后重新计算阈值并重新检查，这种情况就变成了前述没有欺骗进程表现出欺骗行为的情况。如果一个分区通过 AARB 交付了某个值，而另一个分区没有交付该值且计时器到期，也会出现相同情况（引理 5.9）。

□

引理 5.9（AARB 接收）。 如果一个非故障进程从 p_s AARB 交付 v，那么所有非故障进程最终都会从 p_s AARB 交付 v。

证明。 首先，由于 $d + t < 2h(d_r) - n$，欺骗性进程和拜占庭进程无法使两个非故障进程 AARB 交付不同的值（与引理 4.2 类似）。然后，在进程 p_i AARB 交付值 v 之前，它会广播一条 READY 消息，其中包含证明交付 v 合理的证书。因此，当 p_j 收到该 READY 消息时，它也会 AARB 交付 v。

□

引理 5.10 (AARB 可问责性)。 如果两个非故障进程 p_i 和 p_j分别 AARB 交付 v 和 v'，且满足 $v \neq v'$，那么所有非故障进程最终都会收到针对至少 $2h(d_r) - n$个进程（包括 p_s）欺骗行为的欺诈证明。

证明。 非故障进程会广播其 AARB 交付的值的证书，其中包含来自不同进程的 $h(d_r)$条已签名消息。因此，类似于引理 4.2，至少有 $2h(d_r)$ 个进程必须发送过冲突消息，并且在交叉核对冲突证书时会被发现。此外，一些非故障进程必须从 p_s 收到冲突的已签名 INIT 消息，才能达到阈值 $h(d_r)$并 AARB 交付冲突消息，这意味着 p_s 也有故障。

□

引理 5.11 (AARB 主动问责性)。 Basilic 的 AARB 协议满足主动问责性。

证明。 我们在此证明，如果若干故障进程向两个子集 $A, B \subseteq N$发送相互冲突的消息，且每个子集都至少包含一个非故障进程，那么：

- 最终，所有非故障进程都会终止，而不会移除故障进程；或者
- 最终，所有非故障进程都会收到针对这些故障进程的欺诈证明，并将其从委员会中移除；之后，如果源是非故障的，它们就会终止。

不失一般性，我们只考虑 $p_A \in A$ 和 $p_B \in B$。如果它们都在存在冲突消息的情况下终止，我们就完成了。反之，假设只有其中一个（例如 p_A）终止，并 AARB 交付了值 v。那么 p_A 广播一条包含足够数量的 $h(d_r)$ 消息的 READY 消息，证书 $cert$ 中也包含这些消息，从而使 p_B 也能 AARB 交付 v 并终止。不失一般性，我们只考虑一个故障进程 p_i。如果 $cert$ 中来自 p_i 的签名与 p_B存储的来自 p_i 的本地签名冲突，那么 p_B 就会为 p_i构造并广播一份欺诈证明，然后更新委员会和阈值。接着，它重新检查证书，过滤掉 p_i的签名，这样 p_B 也会 AARB 交付 v（因为阈值也相应降低了）。

假设 p_A 和 p_B 都尚未终止。那么，当计时器到期且二者都广播它们所交付的 INIT 和 ECHO 消息时，它们都将能够为 p_i 构造一份欺诈证明（PoF），之后更新委员会和阈值。然后，如果源是非故障的，非故障进程可以按照前一种情况类似地终止。

□

5.4.3 Basilic 二值共识

本节重点讨论 Basilic 的二元共识 AABC 的性质。我们首先在引理 5.12 中证明，如果所有非故障进程以相同的估计值 v 开始第 r 轮，那么所有非故障进程都会在第 r 轮决定 v，或者在第 $r+1$ 轮决定 v。然后，我们在引理 5.13 中证明 AABC 主动问责性，在引理 5.14 中证明 AABC一致性，在引理 5.15 中证明 AABC强有效性，并将 AABC有效性作为引理 5.15 的推论 5.1，在引理 5.16 中证明 AABC- 终止性，以及在引理 5.17 中证明 AABC 问责性。因此，正如我们在定理 5.1 中所展示的，这使 AABC 成为首个具有主动问责性的二元共识协议。

引理 5.12. 假设每个非故障进程在第 r 轮开始时的估计值为 v。那么，每个非故障进程都会在第 r 轮或第 $r+1$ 轮结束时决定 v。

证明。 根据引理 5.3， v 最终会传递给每个非故障进程。根据引理 5.4， v 是传递给每个非故障进程的唯一值。因此， v 是 二值集合中的唯一值，也是非故障进程回显的唯一值，因为阻止终止的欺骗进程会在计时器到期时从委员会中移除（并更新阈值）。这意味着 v 将是 值中的唯一值。如果 $v = r \mod 2$，则所有非故障进程都会决定 v。否则，根据同样的论证，每个非故障进程都会在第 $r+1$ 轮决定 v。

□

我们将在引理 5.13 中证明，Basilic 的 AABC 满足 AABC 主动账户可问责性。

引理 5.13 （AABC 主动问责性）。 Basilic 的 AABC 满足主动问责性。

证明。 我们证明：如果故障进程 p_i 向两个子集 A， $B \subseteq N$ 发送两条冲突消息，且每个子集都至少包含一个诚实进程，那么最终所有诚实进程都会终止；否则，它们会收到针对 p_i 的欺诈证明，并将其从委员会中移除，之后所有进程都会终止。

首先，我们注意到，没有进程会卡在某一轮。根据 ABV 终止性（引理 5.1），每个诚实进程最终都会 ABV 交付一个值，因此进程 p_i 不会卡在阶段 1。

进程也不会卡在等待阶段 2 上。首先，注意，诚实进程的 ECHO 消息中包含的每个值，最终都会交付给 二值集合。其次，所有诚实进程最终都会交付 $h(d_r)$ ECHO 消息；或者，在计时器到期时，进程会交换各自的 ECHO 消息，从而能够构造欺诈证明，并移除阻碍终止的 d_r 欺骗进程。在后一种情况下，从委员会中移除所有欺骗进程并更新阈值后，它们将交付足够的 ECHO 消息以终止阶段 2，因为对于 $d_r = d$， $h(d_r) \leq n - q - t - d$。

接下来，我们证明所有诚实进程始终持有有效证书，以便广播合适的消息；否则，可能会妨碍终止性

阶段 1 的 ABV 广播期间。对于奇偶性与已完成轮次 $r-1$ 相同的估计，进程 p_i 必定已收到该轮次的有效证书（否则它不会在该轮次终止）。如果奇偶性匹配，那么它总能根据轮次 $r-1$ 中交付的估计构造出有效证书。

因此，每一轮中所有进程都会始终无限推进。考虑 GST 之后满足以下条件的第一轮 r ： (i) 协调者是诚实的；(ii) 所有诚实进程都已检测并移除了所有欺骗进程。在这种情况下，每个诚实进程都会优先采用协调者的值，并将其作为自己的 ECHO 消息，且只添加该值。因此，每个进程都会采用相同的值，并在第 r 轮或第 $r+1$ 轮作出决定（根据引理 5.12）。

□

引理 5.14 （AABC一致性）。 若 $d+t \leq 2h-n$，则非故障进程不会作出不同的决定。

证明。 不失一般性，假设非故障进程 p_i 在第 r 轮决定 v 。这意味着 p_i 在第 r 轮收到了 h（d_r）条 ECHO 消息，并且 $vals = \{v\}$。考虑同一轮中非故障进程 p_j 收到的 ECHO 消息。如果 v 位于 p_j 的 $vals$中，那么 p_j 会采用估计值 v ，因为 $v = r$ 模 2。反之，如果 p_j 的 $vals = \{w\}$， $w \neq v$，那么 p_j 收到的 h（d_r）条 ECHO 消息中只包含 w。

类似于引理 4.2， p_j 和 p_i 分别不可能收到 h(d_r)ECHO 条消息，分别对应 v 和 w。然后根据引理 5.12，我们得出结论：每个非故障进程都会在第 $r+1$ 轮或第$r+2$轮决定值 v 。

□

引理 5.15 (AABC强有效性)。 如果某个非故障进程决定了 v，那么某个非故障进程提议了 v。

证明。 该证明与 Polygraph 对强有效性的证明相同 [19, 21]。 □

推论 5.1 (AABC-有效性)。 如果所有进程均无故障且以相同的值开始，那么该值就是唯一的决策值。

引理 5.16 (AABC-终止性). 每个非故障进程最终都会就某个值做出决定。

证明。 该证明直接由引理 5.13 推导得出。 □

引理 5.17 （AABC问责性）。如果两个非故障进程输出了不一致的决策值，那么所有非故障进程最终都会识别出至少$2h-n$ 个应对此不一致负责的故障进程。

证明。此证明与 Polygraph 的问责性证明完全相同 [19, 21]，只是推广到任意阈值 h（d_r），类似于我们在引理 5.10 中所做的推广。

□

定理 5.1。 Basilic 的 AABC 解决了主动问责二元共识问题。

证明。 推论 5.1 以及引理 5.14、5.13、5.16 和 5.17 分别证明了 AABC-有效性、AABC一致性、AABC 主动问责性、AABC-终止性和 AABC-问责性。 □

5.4.4 通用 Basilic 协议

本节汇总所有结果，并给出通用 Basilic 协议的证明。我们在引理 5.21 中证明主动问责性，在引理 5.18 中证明有效性，在推论 5.2 中证明终止性，在引理 5.19 中证明一致性，并在引理 5.21 中证明问责性。最后，我们在定理 5.2 中证明 Basilic 解决了主动可问责共识问题。

推论 5.2 （终止性）。 Basilic 协议满足终止性。

证明。 由引理 5.21 可直接得出。 □

引理 5.18 （有效性）。 Basilic 满足有效性。

证明。 根据推论 5.1 和 AARB 的证明可直接得出。假设所有进程都以值 v 开始运行 Basilic。若所有进程都为非故障进程，则每个由 AARB 交付的提案都由某个非故障进程通过 AARB 发送；又因为所有进程都通过 AARB 发送 v，所以只有 v 会通过 AARB 交付。

由于进程最初只会启动其能够提议 1 的 AABC 实例，这意味着最终所有进程都会启动一个提议 1 的 AABC 实例。根据推论 5.1，该实例将终止，且所有进程都会决定 1。由于其余 AABC 实例最终都会根据引理 5.16 终止，这意味着进程至少会有一个输出 1 的 AABC 实例终止。在计算所有关联位设为 1 的值（它们全都是v）的最小值后，所有进程都会决定 v。 □

引理 5.19 (一致性)。 Basilic 协议满足一致性。

证明。根据引理 5.14 和 5.9，结论显然成立。 □

引理 5.20 （问责性）。 如果两个非故障进程输出了不一致的决策值，那么所有非故障进程最终都会识别出至少 $2h - n$ 个应对这种不一致负责的故障进程。

证明。 该证明可由引理 5.13 和 5.11 直接得出。 □

我们在引理 5.21 中证明，Basilic 满足主动问责性。

引理 5.21 （主动问责性）。Basilic 满足主动问责性。

证明。我们证明，如果某个故障进程 p_i 向两个子集 A， $B \subseteq N$ 发送两条冲突消息，且每个子集都至少包含一个诚实进程，那么最终所有诚实进程都会终止；或者，它们会收到针对 p_i 的欺诈证明，并将其从委员会中移除，之后它们都会终止。

首先，类似于引理 5.13，在 Basilic 中可以发送的所有冲突消息，都是 Basilic 的 AARB 或 AABC 的消息，它们已经满足主动

问责性（见引理 5.13 和 5.11）。这意味着，如果 $d_r > 0$，那么诚实进程最终会更新委员会和阈值，随后重新检查它们是否持有足够的签名消息以终止。接下来，我们证明终止性。根据 AARB-Send 性质（引理 5.8），所有诚实进程最终都会交付来自诚实进程的提案。最终，所有诚实进程都会在索引对应于诚实提议者的所有二元共识实例中提议 1，并根据 AABC-有效性决定 1。由于最终 $h(d_r) \leq n-q-d-t$如果足够的 d_r 阻止终止，因此会被检测并移除，我们可以得出结论：至少有 $h(d_r)$ 个二元共识实例会终止并决定 1。

一旦诚实进程在至少 h（d_r）个提案上决定 1，它们就会对其余提案提议 0；根据 AABC-终止性（引理 5.16），所有剩余的二元共识实例都会终止。接下来，我们证明，对于每个决定为 1 的二元共识实例，至少有一个诚实进程 AARB 交付了与之关联的提案。反证如下：如果没有诚实进程 AARB 交付其关联提案，那么所有诚实进程都会提议 0；这意味着根据 AABC-有效性，该二元共识的最终决定应为 0，而不是 1。因此，根据 AARB 接收性质（引理 5.9），最终所有诚实进程都会交付它们决定为 1 的所有二元共识实例对应的提案。最后，进程决定采用索引较小的提议者所提议的值。

□

我们将所有证明归纳于定理 5.2 的结果中，以证明当 $d+t < 2h_0 - n$ 和 $q + t \leq n - h_0$ 时，初始阈值为 h_0 的 Basilic 协议可以解决共识问题。正如推论 5.3 所示，这一结果意味着，当 $n > 3t + d + 2q$ 时，Basilic 协议类可以解决共识问题。

定理 5.2。当 $d+t < 2h_0 - n$ 和 $q+t \leq n-h_0$ 时，初始阈值为 $h_0 \in (n/2,\ n]$ 的 Basilic 协议解决主动可问责共识问题。

证明。推论 5.2 和引理 5.21、5.18、5.19 和 5.20 分别满足终止性、主动问责性、有效性、一致性和问责性。 □

推论 5.3（推论）。 Basilic 协议类在 $n > 3t + d + 2q$ 的条件下解决主动可问责共识问题。

证明。 在从由 $d + t < 2h_0 - n$ 和 $q + t \leq n - h_0$ 定义的两个不等式构成的系统中移除 h_0 后，该结论可由定理 5.2 直接得出。 □

证明。证明与定理 4.1 类似，不同之处在于，欺诈进程实际上可以通过发送冲突消息来阻止终止性。因此，我们有 $n+t+d \leq 2n-2q-2t-2d$，这意味着 $n > 3(t+d)+2q$。 □

5.5 Basilic 的复杂度

本节将展示 Basilic 的复杂度。我们为每个进程各执行一次 Basilic 的 AARB 可靠广播和 Basilic 的 AABC 二元共识。我们将在第 5.5.3 节证明这些复杂度。

5.5.1 朴素 Basilic

我们在表 1 中总结了三种未经优化的协议的复杂度。

复杂度	AARB	AABC	Basilic
Time	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
消息复杂度	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$
Bit	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(\lambda n^5)$

表 1：GST 后 AARB、AABC 和通用 Basilic 协议的朴素实现的时间复杂度、消息复杂度和比特复杂度。

5.5.2 优化的 Basilic

GST 之后，Basilic 的复杂度与其他近期工作具有相同的渐近复杂度；这些工作并不具备主动问责性， [19, 21, 65], 其中一些甚至不具备问责性， [17], 如表 2 所示。这是因为对手无法阻止任何阶段终止。因此，GST 之后，所有进程都可以在该阶段计时器到期时进入下一阶段或终止协议，其执行等同于 Polygraph 的执行（除了 ABV 广播中多广播一条消息）。在此表中，朴素 Basilic 指的是我们在算法1中展示的协议；而下一行多值 Basilic 展示的是 Polygraph 中所示、同样适用于 Basilic 协议的类似优化， [21]。包含“超级区块”的行指的是应用额外超级区块优化后的结果， [26, 28], 该优化通过求解 SBC（定义 3.1），而非仅仅求解共识，来决定所有 h（d_r）（$\mathcal{O}$（n））关联的 AABC 实例输出为 1 的提案的并集。这项优化仅适用于民主型协议：在这类协议中，所有进程都会提供输入 [26, 21, 67, 68, 72]（即表 2 中的 DBFT、Polygraph 和 Basilic），输出则由这些输入合并得出。经过这些优化后，Basilic 的归一化比特复杂度（即每次决策的复杂度）与其他仅具备问责性、而不具备主动问责性的工作一样低，例如 BFT 取证 [65]或 Polygraph [21]。此外，由于这是实现问责性的最低复杂度， [21], 这也意味着其比特复杂度同样是最优的。请注意，其他工作中已有的优化，例如 BFT 取证在协议 [70], 经过n 次迭代后，每次决策可获得 $\mathcal{O}$（λn^2）的摊销复杂度，也与我们的优化正交，因此同样适用于 Basilic。

Basilic 以及其他民主型协议 [21, 26], 相较于非民主型协议 [65, 70], 的另一项优势在于，提案的分布会将比特分散到网络的多个通道中，而不是使以领导者为发送方或接收方的通道负担过重。也就是说，虽然 BFT 取证将每条网络通道的摊销复杂度标准化为

是 $\Theta(\lambda n)$，因为这是必须通过领导者收发的 $\Theta(n)$条通道发送的比特数；而 Basilic 的则是 $\Theta(\lambda)$，这些比特则分别通过网络中的每条 $\Theta(n^2)$对等通道发送。

最后，表 2 中其余协议不仅不具备主动可问责性，这还意味着，在 BDB 模型中，它们只能解决至多容忍 $t < n/3$ 个故障的共识问题；而初始阈值为 $h_0 = 2n/3$ 的 Basilic 能够解决其中 $d + t < n/3$ 和 $q + t \leq n/3$ 个故障的共识问题，因此在这些工作中能够容忍最强的对手。

表 2：Basilic 与其他工作的复杂度比较。

算法	Msgs	Bits	可问责	主动可问责
PBFT [17]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	✗	✗
Tendermint [12]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	✗	✗
HotStuff [70]	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^2)$	✗	✗
DBFT 超级区块 [26]	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	✗	✗
BFT 取证 [65]	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^3)$	✓	✗
Polygraph 的二进制 [21]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	✓	✗
朴素 Polygraph [21]	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^5)$	✓	✗
Polygraph Multi-v. [21]	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^4)$	✓	✗
Polygraph 超级块 [21]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	✓	✗
Basilic 的 AABC	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	✓	✓
朴素 Basilic	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^5)$	✓	✓
多值 Basilic	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^4)$	✓	✓
Basilic 超级块	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	✓	✓

5.5.3 复杂度证明

本节证明我们在第 5.5 节介绍的 Basilic 的复杂度，以及 Basilic 的 AARB 和 AABC 的复杂度。

引理 5.22（Basilic 的 AARB 复杂度）。GST 之后，若源无故障，则 Basilic 的 AARB 协议的时间复杂度为 $\mathcal{O}(1)$，消息复杂度为 $\mathcal{O}(n^2)$，比特复杂度为 $\mathcal{O}(\lambda \cdot n^3)$。

证明。 GST 之后，当计时器归零时，所有非故障进程都将收到来自每个非故障进程和每个欺骗进程的消息。因此，非故障进程要么可以终止，要么会广播其当前的 ECHO 和 INIT 消息列表，之后移除已检测到的欺骗进程，并且也可以终止。因此，时间复杂度为 $\mathcal{O}(1)$。消息复杂度为 $\mathcal{O}(n^2)$，因为每个非故障进程至少会广播一条 ECHO 和 READY 消息，并且在某些执行中，还会广播一份截至计时器归零时已交付的 ECHO 消息列表。由于该列表和 READY 消息都包含 $\mathcal{O}(n)$个签名，即 $\mathcal{O}(\lambda n)$比特，Basilic 的 AARB 的比特复杂度为 $\mathcal{O}(\lambda n^3)$。

□

引理 5.23（Basilic 的 AABC 复杂度）。 GST 之后，Basilic 的 AABC 协议的时间复杂度为 $\mathcal{O}(n)$，消息复杂度为 $\mathcal{O}(n^3)$，比特复杂度为$\mathcal{O}(\lambda \cdot n^4)$。

证明。GST 之后，Basilic 协议会在第一个满足以下条件的轮次终止：(i) 该轮次的领导者是非故障进程；(ii) 已移除足够多的欺骗性故障，使其无法阻止终止。由于 $t + d + q < n$，我们知道 (i) 在 $\mathcal{O}(n)$ 中成立。每增加一个因欺骗性故障而无法终止的轮次，都会移除非零数量的欺骗性故障，因此我们也知道 (ii) 在 $\mathcal{O}(n)$ 中成立。这意味着 Basilic 会在 $\mathcal{O}(n)$ 轮次内终止。在 AABC 的阶段 1 中，每一轮非故障进程都会执行一次 ABV-广播，发送 $\mathcal{O}(n^2)$，并产生 $\mathcal{O}(n^3)$ 条消息。比特复杂度为 $\mathcal{O}(\lambda n^4)$，因为每条消息最多可包含两个账本，每个账本含有 $\mathcal{O}(n)$ 个签名，即 $\mathcal{O}(\lambda n)$ 比特。阶段 2 的复杂度与阶段 1 相当，可用类似方法得出：如果欺骗性故障阻止阶段 2 终止，非故障进程可能会广播 $\mathcal{O}(n)$ 个签名；如果它们在该轮次作出决定，则可能会广播一份证书。 □

定理 5.3. Basilic 协议的时间复杂度为 $\mathcal{O}(n)$，消息复杂度为 $\mathcal{O}(n^3)$，比特复杂度为 $\mathcal{O}(\lambda \cdot n^5)$。

证明。 由引理 5.23 和引理 5.22 可立即得证，因为 Basilic 会执行 n 个 AARB 实例，随后执行 n 个 AABC 实例。 □

5.6 使用 Basilic 求解最终共识

本节将调整 Basilic，以在 BDB 模型中求解最终共识，并证明 Basilic 协议具有容错最优性。最终共识（◇-共识）抽象 [32]刻画了所有参与者之间的最终一致性。它向每个进程 p_i导出操作 proposeEC_1,proposeEC_2，..

这些操作接受多值参数（非故障进程提议有效值）并返回多值响应。假设对于所有 $j \in N$，每个进程在收到 proposeEC_{j-1}的响应后，都会立即调用 proposeEC_j，则该抽象保证在每个可容许执行中，存在 $k \in N$，使得满足以下性质：

- ◇**-终止性**。 对于所有 $j \in N$，每个非故障进程最终都会对 proposeEC_j 返回响应。
- ◇**-完整性**。 对于所有 $j \in N$，没有任何进程会对 proposeEC_j 响应两次。
- ◇**-有效性**。 对于所有 $j \in N$，对 proposeEC_j 返回的每个值此前都已被提议给 proposeEC_j 。
- ◇**-一致性**。 对于所有 $j \geq k$，不存在两个非故障进程向 proposeEC_j 返回不同的值。

我们详细介绍 ◇*-Basilic*（*BEC*），这是针对 ◇-共识问题对 Basilic 的一种改编。进程 p_i 按照以下步骤执行 ◇-Basilic：

1. BEC 首先执行 Basilic-gen-propose$_{h_0}(v_i)$，其输出由p_i 作为 BEC 的 proposeEC$_0$ 输出返回。

2. 如果 p_i 发现操作 k 和 k'之间不存在不一致，那么对于所有操作 proposeEC$_j$, $k' > j \geq k$，输出都是 proposeEC$_{j-1}$ 的输出。

3. 如果 p_i在操作 j中发现索引为 $r \in [0, n-1]$的新不一致，则：

(a) 如果不一致发生在 AARB 交付的值之间，BEC 按如下方式解决：设 (EST, $\langle u, r\rangle$) 是与本地 AARB 交付的值 (EST, $\langle v, r\rangle$) 不同的值，则对于 proposeEC$_j$, p_i，将 $y = \min(v, u)$ 应用于不一致的值。接下来，如果 proposeEC$_{j-1}$ 的输出为 v, p_i，则用 y替换 AARB 交付的值，并为 proposeEC$_j$输出 y。

(b) 如果 AABC 协议在值 1 和 0 之间出现不一致，则 p_i 将二进制决策$[r]$设为 1。然后，p_i 重新计算添加此二进制决策后最小已决值是否发生变化（即重新执行算法1的第 11–13 行），并将此决策输出给 proposeEC$_j$。

(c) 最后，p_i 广播所有尚未广播的分歧值（及其证书）。

我们在定理 5.4 中证明，如果 $d + t < h_0$ 和 $q + t \leq n - h_0$，那么初始阈值为 h_0 的 ♢-Basilic 可以解决 ♢-共识问题，其中 t, d 和 q 分别是拜占庭、欺骗性和良性进程的数量，而 h_0 是初始阈值。这意味着，正如我们在推论 5.4 中所示，对于拜占庭、欺骗性和良性进程数量分别为 t, d 和 q，且满足 $2t + d + q < n$ 的任意组合，♢-Basilic 协议类都能解决 ♢-共识。

定理 5.4 （♢-共识的阈值）。初始阈值为 h_0 的 ♢-Basilic 协议可以解决 ♢-共识问题，条件是 $d + t < h_0$ 和 $q + t \leq n - h_0$。

证明。♢-完整性显而易见。界限 $q+t \leq n-h_0$ 在推论 4.3 中得到了证明：♢-Basilic 首先执行 Basilic；除非满足 $q + t \leq n - h_0$，否则不会终止，从而满足 ♢- 终止性。♢- 有效性直接源自 Basilic 对有效性的证明（引理 5.18 ）。

我们只剩下证明 ♢-Agreement。如果 $d + t < h_0$，那么所有有效证书都至少包含一个诚实进程。这意味着不一致的数量是有限的。由于诚实进程会广播它们发现的所有不一致（以及相应的有效证书），因此所有诚实进程最终都会发现所有不一致。此外，所有诚实进程都会根据 Basilic 的问责性质（引理 5.21）发现 Basilic 的所有不一致。考虑除 p_i 之外的所有诚实进程都已经发现并处理了所有不一致（如 ♢-Basilic 协议所规定）。假设 p_i 在某个 $k > 0$ 的 proposeEC$_{k-1}$ 操作开始时发现了最后一个不一致。那么，对于所有 $j \geq k$，没有两个诚实进程会在 proposeEC$_k$ 中返回不同的值，从而满足 ♢-Agreement。 □

推论 5.4 (◊-共识)。 Basilic 协议类在 $n > 2t + d + q$ 时解决 ◊-共识。

证明。 从由 $d + t < h_0$ 和 $q + t \leq n - h_0$ 定义的两个不等式构成的系统中移除 h_0 后，结论可由定理 5.4 立即得出。 □

6 零损失区块链

在介绍了我们具有主动问责能力的 Basilic 协议类之后，现在我们来介绍零损失区块链（ZLB）。ZLB 是首个能够容忍对手控制多达 t_s 个进程并试图造成不一致，同时也能够容忍多达 t_l 个拜占庭进程的区块链。ZLB 通过解决暂时分歧并替换经证明存在欺诈行为的进程，实现了这种高水平的容错能力。为此，我们首先详细介绍持久区块链问题，以及针对 ZLB 持久特性而设定的另一项对手假设。

具体而言，解决持久区块链问题，就是在可能时实现共识（$n > 3t + d + 2q$，推论 5.3），并通过排除故障进程、解决共识遭到破坏的情况以及防止此类情况再次发生（$n \leq 3t + d + 2q$），从共识被破坏的状态中恢复（$n > 3t' + d' + 2q$）。

6.1 持久区块链

持久区块链（LLB）是一种拜占庭容错状态机复制，它允许某些共识实例在通过合并由此产生的分叉的各个分支、并使用 SBC（定义 3.1）决定所有过去决策的并集来解决不一致之前，先出现不一致。因此，我们认为共识实例 Φ_i 输出一个可枚举的决策集合 $out(\Phi_i) = s_i, \quad |s_i| \in \mathbb{N}$，所有 n 进程都会复制该集合。我们将状态机复制在第 i 个共识实例 Φ_i 处的状态定义为截至第 i 个共识实例的所有实例的全部决策。

更形式化地说，若状态机复制确保终止性、一致性和收敛性，则它是 LLB：

定义 6.1（持久区块链问题）。 若满足以下所有性质，则状态机复制是 LLB：

1. **终止性：** 对于所有 $k > 0$，共识实例 Φ_k 解决最终共识问题。

2. **一致性：** 如果 $d + t < 2h - n$ 当 Φ_k 启动时，那么执行 Φ_k 的诚实进程就会达成一致性。

3. **收敛性：** 只有有限个共识实例能够解决最终共识，此后所有共识实例都能解决共识。

终止性并不意味着同一个共识实例的首次输出中诚实进程之间达成一致性，但它意味着所有实例都会以一个输出终止，该输出可能发生变化，直至最终达成一致性；而一致性是共识的经典性质。收敛性保证在达成一致性之前，不一致的次数有限。

6.2 慢适应性对手

与单次共识不同，考虑状态机复制需要应对这样一种攻击：对手选择的被攻陷进程会随时间变化。因此，我们认为，控制这些故障进程的对手是适应性的，因为 f 可以随时间变化。不过，我们假设该对手具有慢适应性 [54], ，正如以往具有动态成员资格的区块链系统 [54], 所采用的假设：对手会经历静态时期，在此期间，拜占庭、欺骗性、良性和诚实进程的状态保持不变。为保证收敛性，我们假设这些静态时期足够长，使诚实进程能够发现并替换故障进程。具体而言，每个静态时期 $\mathfrak{t}$ 都分配有一个共识实例 Φ_k ，且 $\mathfrak{t}$ 在 Φ_k 启动时开始。为应对流水线共识实例， $\mathfrak{t}$ 可能不会恰好在 Φ_k结束时终止，因为此时可能还有 Φ_ℓ、.....、 Φ_m 等实例正在运行；在这种情况下，只有当 Φ_ℓ、.....、 Φ_m 和 Φ_k 均已达成一致并结束后，$\mathfrak{t}$才会结束（静态时期 $\mathfrak{t}+1$ 才会开始）。

6.3 候选进程池

由于我们的系统将执行成员变更，排除某些进程并包含新的进程，因此我们通过假设存在一个包含 m 用户的大型池来将所有可以加入的用户建模为系统中的进程，其中至少有 $2n/3$ 是诚实用户（m 可能远大于 n），其余用户则具有欺骗性。该池仅旨在表示这样一个最低要求：在所有曾被提议作为进程包含进系统的用户中，至少有 $2n/3$ 个诚实用户最终会由诚实进程提议。请注意，这一假设明显弱于例如假设诚实进程始终提议包含诚实用户。为简单起见且不失一般性，我们假设在对手的同一静态时期内，如果该池中的某个用户此前已经是进程，则不会再次提议该用户。

6.4 零损失区块链

本节将详细介绍我们的系统。其两个主要理念是：(i) 用新进程替换那些无可争议地应对分叉负责的欺诈进程，从而趋向于能够达成共识的状态；(ii) 对错误决策的冲突交易进行退款。我们将证明，ZLB解决了长期存在的区块链问题。如图3所示，下面介绍ZLB系统的组件，即可问责状态机复制（ASMR）（第 6 节—

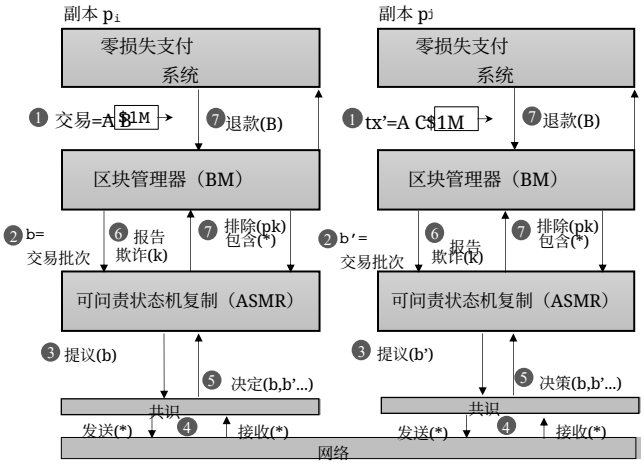


图3：我们的 ZLB 系统的分布式架构依赖于可问责状态机复制（ASMR）、BM 和支付系统。 ❷ 每个进程都会批量处理一些支付请求，图中以一笔从 Alice 的账户 (A) 转至 Bob 的账户 (B) 的 100 万美元转账交易（tx）为例（相应地，tx′ 是从 Alice 的账户 (A) 转至 Carol 的账户 (C) 的 100 万美元转账）。假设 Alice 最初有 100 万美元，并试图通过修改一个由她控制的进程的代码来进行双重支付，以发动合谋攻击。 ❸–❻ ASMR 组件会检测到试图进行双重支付的欺诈进程 p_k 、相关交易 tx 和 tx′ ，以及资金不足的账户 A 。它会使用 A 的余额为交易 tx 提供资金，❻ 通知 BM ❼ 排除或替换进程p_k ，并 ❼ 使用 p_k被罚没的押金为 tx′ 提供资金。

6.4.1）和区块链管理器（BM）（第 6.4.2 节），但我们将零损失支付应用（第7节）留待后文介绍。

只要客户端向某个进程提交新请求，该进程的支付系统组件就会将其转换为支付，并传递给 BM 组件。如图3所示，收到足够多的支付请求后，BM 会向可问责状态机复制（ASMR）发出一批请求；随后，ASMR 将其提交给共识组件。共识组件通过网络交换消息，使诚实进程达成一致。如果检测到不一致，就会罚没欺诈进程的账户。假设 Alice (A) 试图通过以下方式进行双重支付：(i) 分别向 Bob (B) 和 Carol (C) 支付她的 100 万美元，交易分别为 交易 和交易′；(ii) 入侵进程 p_k 的代码，使其产生欺骗性故障，从而导致不一致。ASMR 检测到不一致后，会通知 BM，进程 p_k 会被排除或替换，而 tx' 则由 p_k 被罚没的押金提供资金。

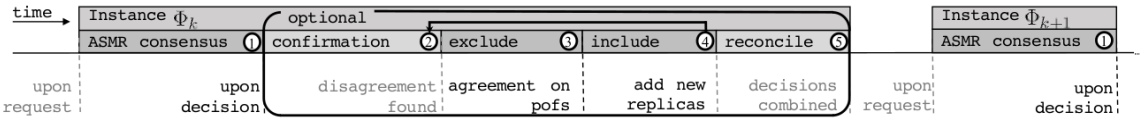


图4：如果存在已入队且等待处理的请求，那么进程会通过参与一个 ASMR 共识阶段 ① 来启动一个新实例 Φ_k；随后可能会进行一系列阶段： ② 进程尝试确认这一决策，以确保没有其他诚实进程对此表示不一致， ③ 如果故障进程导致了不一致，则调用排除协议， ④ 随后包含新进程，以弥补被排除的进程， ⑤ 并合并两个已决交易批次。其中一些阶段在共识终止时完成（黑色），而其他阶段则在收到简单通知时终止（灰色）。进程会在阶段 ②-⑤ 终止之前启动一个新实例 Φ_{k+1}，因为这些阶段并不总能保证终止。

6.4.1 可问责状态机复制（ASMR）

为了检测欺骗进程，我们现在介绍据我们所知首个可问责状态机复制，称为 ASMR。ASMR 通过无限循环执行以下五项操作： ① 主动可问责共识（定义 5.1），尝试对一组新的交易作出决策； ② 确认，旨在确认已达成一致性； ③–④ 成员变更，旨在用新进程替换导致不一致的已检测故障进程；以及 ⑤ 协调阶段，用于合并不一致相关的所有决策，如图4所示。

我们的 ASMR 使用的共识协议是 Basilic，它已能在运行时移除进程。然而，值得注意的是，诚实进程不会永久移除在 Basilic 的某个共识实例中被移除的进程（也就是说，该共识实例终止后，这些进程会在下一个共识实例开始时重新加入）。不过，诚实进程会存储欺诈证明，并最终触发成员变更，以新进程永久替换故障进程。

ASMR 的阶段对于每个索引，ASMR 首先执行主动可问责共识阶段 ①，尝试就一组交易达成一致。随后，它可能会结束接下来的四个阶段 ②–⑤，以从可能的不一致中恢复，或确认并未发生不一致。

①**ASMR 共识：**诚实进程向主动可问责共识协议的 Basilic 类实例提出一组交易，这些交易是它们从客户端收到的，初始阈值为 $h_0 = 2n/3$，希望达成一致性。当共识终止时，所有诚实进程对同一决策达成一致；或者某些诚实进程意见不一致：它们决定不同的交易集合。

② **确认：** 由于当对手控制 $d+t \geq n/3$ 欺骗性和拜占庭进程时，诚实进程可能暂时无法察觉不一致，因此它们会进入确认阶段，等待来自比 BFT 共识所要求的更多不同进程的消息。如果故障进程导致了不一致，那么确认阶段就会终止，并使诚实进程检测到不一致，即诚实进程会收到支持不同决策的证书。否则，该阶段可能不会终止，因为诚实进程需要收到超过 $(\delta + 1/3) \cdot n$ 个进程的消息，其中 δ 是潜在欺骗性故障 $\delta = (d + t)/n$的比例，以此保证欺骗比例为 欺诈比例 δ时不可能出现不一致。具体而言，当 $q = 0$（即 $f = t + d$）时，诚实进程需要收到来自 $n - x$ 个进程的一致消息，满足 $\lfloor (n - x)/(f - x + 1) \rfloor = 1$；这意味着当 $f = \lceil 5n/9 - 1 \rceil$时至少需要 $8n/9$ 个进程，或者当 $f = \lceil 2n/3 \rceil$时需要所有进程，正如我们在定理 6.1 中所示。更具体地说，我们称决策 v 在 Φ_k 时获得了 α-确认，$\alpha \in [0, 2/3]$ 当且仅当只有欺诈比例为 $\delta \geq \alpha$ 的对手才能使诚实进程在 Φ_k时决定 $v' \neq v$。

然而， Φ_k 始终会终止，因为它与确认并行进行，而不等待确认终止。如果确认阶段终止，则要么确认某个区块已不可撤销地最终确定（没有进程提出异议），要么启动成员变更。

③-④ **成员变更：** 我们的成员变更（算法 5）由两个连续的共识算法组成：一个用于排除欺骗进程（第 22 行），另一个用于添加新加入的进程（第 40 行）。我们将包含和排除分为两个共识实例，以避免对同一进程提议排除和包含进程作出决定。进程 p_i 维护一系列变量：当前共识实例 Φ_k、当前进程标识符全集中的欺骗性 进程 C 、一个在运行时更新、供排除协议使用的进程标识符集合 C' 、进程候选池 进程候选池、证书集合 证书、欺诈证明集合欺诈证明 和新的欺诈证明集合新的欺诈证明、检测到的欺骗进程的本地阈值 f_d 、已决定的欺诈证明集合 排除共识 以及已决定的新进程集合 包含共识 。

③ **排除协议：** 如果诚实进程检测到至少 $f_d = 2h_0 - n$ 个（即对于 $h_0 = 2n/3$， $f_d = n/3$ 个）欺骗进程（通过不同的欺诈证明），它们会停止正在进行的 ASMR 共识（第 19 行），然后使用新的进程集合重新启动（第 47 行）。接着，诚实进程通过改用排除了这些进程的更新后委员会C' ，开始成员变更，并忽略来自这些 f_d 个进程的消息（第 20–22 行）。在本文余下部分，我们固定 $f_d = 2h_0 - n$ 。在排除协议开始时，诚实进程会在第 22 行通过调用 Basilic 主动可问责共识协议，提议其欺诈证明集合 ex-propose。我们用 $h'(d'_r) = h'_0 - d'_r$ 表示排除协议的投票阈值，用 C' 表示排除协议的更新后委员会。本文稍后将讨论 h'_0 的具体取值。

我们的排除协议的关键创新在于，进程可以排除其他

算法 5 进程 p_i 处的成员变更，共识 Φ_k

状态：
  Φ_k, k^{th} 参与的 ASMR 共识实例 p_i 。
  C，组成委员会的进程集合
  C'，已更新的进程集合，初始为 $C' = C$
  certificates，在排除期间收到的证书，最初为 $\emptyset$
  pofs，欺诈证明（PoFs）的集合，初始为 $\emptyset$
  new-pofs，新交付的欺诈证明集合，初始为 $\emptyset$
  cons-exclude，共识输出的欺诈证明集合，初始为 $\emptyset$
  cons-include，共识输出的新进程集合，初始为 $\emptyset$
  pool，用于提议新进程的进程候选池
  $deceitful \in I$，已达成共识的欺诈进程的身份，初始为 $\emptyset$
  f_d，要恢复的欺诈证明阈值，$\lceil n/3 \rceil$ 默认为

收到一份欺诈证明列表后 *-pofs*：
  如果 (验证(*pofs*))**则** ▷ *if PoFs are correctly signed*
    $new\_pofs \leftarrow \_pofs \backslash pofs$
    pofs.添加(*new- pofs*) ▷ *add PoFs on distinct processes*
    如果 (ex-propose**尚未开始**) **则**
      若 (大小(*pofs*) $\geq f_d$) **则** ▷ *enough to change members*
        若 (Φ_k 已启动**且未**完成) **则** Φ_k.*stop*()
        $C' \leftarrow C' \backslash pofs$.进程()
        ex-propose.更新委员会(*pofs*) ▷ *update committee*
        ex-propose.启动(*pofs*) ▷ *exclusion consensus*
    else if ($new\_pofs \neq \emptyset$ **and** ex-propose **not** 已完成) **then** $C' \leftarrow C' \backslash new\_pofs$.进程()
      ex-propose。更新委员会(*new pofs*) ▷ *update committee*
      广播(*new pofs*) ▷ *broadcast new PoFs*
      ex-propose。检查证书(*certificates*) ▷ *recheck certificates*

收到一份 *ex-cert* **排除协议的证书：**
  如果 (*ex-cert* $\notin$ *certificates* **并且** 验证证书(*ex-cert*)) **那么** *certificates*.*add*(*ex-cert*)
  ex-propose。检查证书({*ex-cert*}) ▷ *check certificate with current* C'

函数ex-propose。检查证书(*certs*)：
  对所有 *cert* $\in$ *certs* **执行**
    如果(验证证书(*cert*))**则**
      如果($|cert$.进程() $\cap C'| \geq \frac{2|C'|}{3}$)**则** ▷ *current threshold*
        ex-propose。cert decide(*cert*) ▷ *decide certificate's decision*

在决定欺诈证明列表 *cons-exclude* **后，** ex-propose：
  detected-fraud(*cons- exclude*.获取进程()) ▷ *application punishment*
  pofs $\leftarrow$ *pofs* $\setminus$ *cons-exclude*.获取欺诈证明() ▷ *discard the treated pofs*
  $C \leftarrow C \setminus$ *cons-exclude*.获取欺骗性对象() ▷ *exclude deceitful*
  inc-prop $\leftarrow$ *pool*.取用($|$*cons-exclude*$|$) ▷ *take processes from the pool*
  inc-propose。start(*inc-prop*) ▷ *inclusion cons.*

确定要包含在 *cons-include* inc-propose **中的进程列表后：**
  new-processes $\leftarrow$选择($|$*cons-exclude*$|$, *cons-include*) ▷ *deterministic*
  对于所有 *new process* $\in$ *new processes* , **执行** ▷ *for all new to inc.*
    建立连接(*new process*) ▷ *new process joins*
    发送追赶消息(*new-process*) ▷ *get latest state*
  $C \leftarrow C \cup new\_processes$
  若 (Φ_k 已停止) **则跳转至** ① 图4 ▷ *restart cons.*

因此，进程会在运行时收到新的有效欺诈证明后更新其委员会 C'（第 23–24 行）。请注意，Basilic 已会在运行时移除检测到的故障进程，但从 $d'_r = f_d$ 已移除的进程开始，能让诚实进程从一开始就占据优势。因此，在交付证书（第 29 行）后，诚实进程会验证该证书是否包含来自尚未被检测为故障的进程的阈值$h'(d'_r)$个签名（第 33 行），并在第 34 行决定该证书所证明的提案。更新委员会后，诚实进程会重新检查其所有证书（第 26 行），并重新广播其欺诈证明（第 25 行）。由于我们的排除协议解决了 SBC 问题（定义 3.1），它通过一次至少决定 $h'(d'_r)$ 个提案，最大限度地增加被排除的进程数量。

请注意，进程无需等待 f_d 欺诈证明（第 17 行），而可以在检测到一个欺骗进程后立即启动算法 5。不过，等待至少 f_d 份欺诈证明可确保成员变更确有必要，并有助于一次性从同一联盟中移除许多欺骗进程。

④ **纳入协议：**为补偿被排除的进程，纳入协议 inc-propose（第 40 行）在第 39 行从进程候选池（第 6.2 节）中添加新的候选进程。该纳入协议同样是 Basilic 的一个实例，采用与排除协议相同的投票阈值 $h'(d'_r) = h'_0 - d'_r$，但提案的格式和验证方式有所不同：每个提案包含的新进程数与被排除的进程数相同（第 39–40 行）。与排除协议不同，纳入协议使用更新后的委员会（从第 38 行起的 C），该委员会是从成员变更开始时的委员会中排除在排除共识结束时已决议排除的进程后得到的（第 38 行）。由于已决议提案的并集 $h'(d'_r)$ 包含的进程数量足以满足纳入需求，诚实进程会对所有已决议提案的并集应用确定性函数 choose（第 42 行）。该函数通过从所有已决议提案中均匀选取进程，将委员会规模恢复至 n。这确保了 (i) 各项决策中的纳入名额得到公平分配，以及 (ii) 即使所有纳入的进程都是欺骗进程，欺骗比例也不会增加。最后，应用层会惩罚被排除的进程（第 36 行），并纳入新进程（第 42–47 行）。

来自不同分区的诚实进程在执行成员变更时，可能处于不同的共识实例中。因此，即使成员变更已经完成，仍会存在一个短暂的过渡期，在此期间，诚实进程可能会收到包含已被排除进程的证书的区块。这些区块是在其他分区中的诚实进程执行成员变更之前达成决定并进行广播的。然而，需要注意的是，只要 $f < h$ 成立，根据构造，所有证书都至少包含一个诚实进程。因此，所有诚实进程最终都会更新其委员会，并停止生成包含已被排除进程的新证书。

⑤ **协调：** 一旦交付了带有关联有效证书的冲突区块，协调便从合并所有交易开始。

这些交易由在不一致情况下作出决定的不同诚实进程确定。交易通过一个确定性函数进行排序，其简单示例是字典序排序，但也可以通过轮流遍历各实例的索引来实现公平性。

当前实例 Φ_k 终止后，另一个实例 Φ_{k+1} 即可启动，即使它与索引 k 处或更低索引处的确认或协调并发运行。

6.4.2 区块链管理器(BM)

现在，我们介绍区块链管理器（BM）。它基于 ASMR 构建，用于在检测到区块链分叉时合并来自多个分支的区块。一旦识别出分叉，冲突区块不会像经典区块链在发生双重支付时那样被丢弃，而是会被合并。在合并区块的过程中，BM 还会处理冲突交易，例如支付系统中的冲突交易，其方式是使用被排除进程的资金为冲突交易提供资金。关于进程必须在存款中持有多少资金才能保证这一融资机制得以实现，具体细节将在第 7 节中介绍。

与比特币 [59], 类似，BM 接受来自无需许可的用户集合的交易请求。具体而言，这使用户能够使用不同的设备或钱包，发起从同一账户中提取资金的不同交易——这是不具备共识的支付系统所不提供的功能 [25]。与比特币不同，但与近期的区块链 [37, 28], 类似，我们的系统不会激励所有用户参与每个区块的决策过程，而是由一组受限的许可进程负责处理指定区块。这就是 ZLB 提供通常所说的开放式许可区块链的原因 [28]。尽管如此，ASMR 无需进行实质性修改，就能提供具有委员会抽选机制的无需许可区块链 [37] 。

保证各进程之间的一致性 通过构建在能够解决不一致的底层 ASMR 之上，BM 采用区块合并来解决分叉，同时排除已检测到的故障进程并纳入新进程。共识实例可能出现不一致，从而产生多个分支或区块链分叉（定理 6.2）。BM 以 ASMR 的成员变更机制为基础，从分叉中恢复。具体而言，ASMR 每当出现不一致时都会排除 f_d 欺骗进程，这一事实保证欺骗进程的比例 δ 趋向于一个能够保证共识的状态（定理 6.6）。分叉可能产生的最大分支数取决于 q 良性故障的数量、 d 欺骗性故障的数量以及 t 拜占庭故障的数量，也取决于投票阈值；对于状态机复制的历史，这一点已得到证明 [66], ，我们也在定理 6.2 中重述了这一结论。

内存中的交易 ZLB是一个区块链，继承了比特币的相同未花费交易输出（*UTXO*）模型 [59]；每个账户的Balance

算法 6 进程 p_i 的区块合并

```
1: 状态:
2:   Ω, 一个包含以下字段的区块链记录:
3:     deposit, 一个整数, 初始值为 0
4:     inputs-deposit, 一组存款输入, 最初位于第一个存款中
5:     punished-acts, 一组受处罚的账户地址, 初始为 ∅
6:     txs, 一组 UTXO 交易记录, 初始时位于创世区块中
7:     utxos, 一列未花费的输出, 初始时位于创世区块中
8: 收到冲突区块 block时:                                  ▷ merge block
9:   对于 tx 中的每个元素, 执行 block                      ▷ go through all txs
10:    如果 (tx 不在 Ω.txs), 则                            ▷ check inclusion
11:      CommitTxMerge(tx)                                 ▷ merge tx, go to line 17
12:      对于 out, 在 tx.outputs 中执行                    ▷ go through all outputs
13:        如果 (out.account 在 Ω.punished-acts) 则        ▷ if punished
14:          惩罚账户(out.account)                          ▷ punish also this new output
15:  退还输入()                                            ▷ refill deposit, go to line 24
16:  存储区块(block)                                       ▷ write block in blockchain
17: CommitTxMerge(tx):
18:  toFund ← 0
19:  for input in tx.inputs do                             ▷ go through all inputs
20:    如果 (input 不在 Ω.utxos)则                         ▷ not spendable, need to use deposit
21:      Ω.inputs-deposit.添加(input)                      ▷ use deposit to refund
22:      Ω.deposit ← Ω.deposit − input.value               ▷ deposit decreases in value
23:    else Ω.消耗未花费交易输出(input)                    ▷ spendable, normal case
24:  退还输入():
25:  for input in Ω.inputs-deposit do                      ▷ go through inputs that used deposit
26:    如果 (input 属于 Ω.utxos) 则                        ▷ if they are now spendable
27:      Ω.消耗未花费交易输出(input)                       ▷ consume them
28:      Ω.deposit ← Ω.deposit + input.value               ▷ and refill deposit
```

在系统中以UTXO表的形式存储。与比特币不同，为了支持内存优化，系统将维护的未花费交易输出数量保持在最低限度。该表中的每个条目都是一个未花费交易输出，表示某个特定账户（即“输出”）所拥有的一定数量的货币。当一笔从源账户 s_1、……、s_x 向接收方账户r_1、……、r_y 转移的交易执行时，它会检查账户 s_1、……、s_x的未花费交易输出。如果这些账户的未花费交易输出金额充足，则此次执行会尽可能多地消耗未花费交易输出，并生成另一系列未花费交易输出，将转移的金额输出给 r_1、……、r_y，以及可能剩余的金额输出回源账户 s_1、……、s_x。最大化消耗的未花费交易输出数量有助于保持表的紧凑。每个进程通常都可以直接在内存中访问UTXO表，从而更快地执行交易。

区块合并协议 如算法 6 所示，区块链Ω 的状态由一组输入 输入存款 （第 4 行）、一组曾被欺骗进程使用的账户地址受惩罚行为 （第 5 行）、协议使用的存款（第 3 行）、一组交易 以及一个未花费交易输出列表

组成。该算法通过在网络上广播来传播区块，并在收到一个与已知区块冲突的有效区块时启动；它会尝试将所收到区块中的所有交易与区块链 Ω 中的交易合并 Ω （第 11 行）。这是通过调用 CommitTxMerge 函数（第 17–23 行）完成的：输入会被追加到未花费交易输出表中，而冲突输入则由被排除进程的存款（第 22 行）提供资金。我们将在第 7 节说明如何构建一个存款充足的支付系统，以解决成功达成的不一致。

密码学技术 为提供身份认证和完整性，交易使用椭圆曲线数字签名算法（ECDSA）及参数 secp256k1 进行签名，与比特币 [59] 中的做法相同。每个诚实进程都会为其交易分配严格递增的序列号。网络通信在客户端与进程之间使用 gRPC，在进程之间使用原始 TCP套接字，但所有通信通道都通过 SSL 加密。最后，排除协议（算法 5）使用 ECDSA 对导致不一致的消息发送方进行身份认证（即用于欺诈证明）。与 ECDSA 不同，门限加密无法用于追溯故障用户，因为对这些用户的编码所用比特数不足以区分用户；而消息认证码（MAC）不足以提供可转移的身份认证 [23]。

6.5 零损失区块链证明

本节将证明 ZLB 的性质，以解决取决于排除与纳入共识所用投票阈值 h' 的 LLB 问题。我们还将结果推广到 ASMR 共识的投票阈值 h 。接下来，我们将讨论 h'的三种方案，分析其优缺点，并讨论一项额外的理想性质，我们称之为感知性。

6.5.1 α-确认

我们在定理 6.1 中证明，如果一个进程交付了 $c > n - h + \alpha n$ 个不同的证书，那么它要么确认没有大小为 αn 的联盟可能导致不一致，要么发现不一致。

定理 6.1。设 σ 是一个投票阈值为 h 的共识协议，且诚实进程 p_i 在 σ 的一次迭代中决定 v 。如果诚实进程 p_i 从 $c > n - h + \alpha n$ 个不同进程收到证书， $\alpha \in [0, 2/3]$，那么 p_i 要么检测到不一致，要么 α 确认 v。

证明。 p_i 从 $c > n - h + \alpha n$ 个进程处收到证书，这意味着$c - \alpha n > n - h$ 是由诚实进程送达的证书。由于诚实进程总数为 $n - \alpha n$，因此 $x = n - \alpha n - (c - \alpha n) = n - c$ 是 p_i 尚未收到证书的诚实进程数量。若其中某些 x 进程已决定 $v' \neq v$，那么 $x + \alpha n \geq h \iff c \leq n - h + \alpha n$。因此，若 $c > n - h + \alpha n$，则要么 p_i 已经收到证书

对于 v'，要么所有诚实进程都决定了 v，对于 $\delta \leq \alpha$。在后一种情况下，这意味着 v 已得到 α确认。 □

与定理 6.1 相比，我们在定理 6.2 中证明了规模为 $d+t$ 的对手在一个共识实例中能够造成的最大不一致（或分叉分支）数量为 a。

定理 6.2（分支数）。设 σ 是一个投票阈值为 h 的共识协议。假设 $d+t<h$ 个故障进程导致不一致，且令 a 为不一致的决策数。那么，对于 $d+t \geq \frac{ah-n}{a-1}$，$a \leq \frac{n-(d+t)\,h-(d+t)}{}$。

证明。对于 $d+t$ 个故障能够创建 a 个分支，它们必须通过 h 个投票阈值，并且每个阈值都来自诚实进程的 a 个不同不相交划分。由于这些分区互不相交，每个分区包含 $(n-(d+t))/a$ 个进程（不失一般性，假设 n 可被 a 整除）。这意味着，对于攻击者能够生成 a 个分支，必须满足 $\frac{n-(d+t)}{a}+d+t \geq h \iff d+t \geq \frac{ah-n}{a-1}$。用分支数量表示的等价方程为 $a \leq \frac{n-(d+t)\,h-(d+t)}{}$</style}。 □

6.5.2 排除和纳入协议

定理 6.3（排除/纳入协议的共识）。设 ZLB 以投票阈值 h 执行，其中 $f_d = 2h_0 - n$ 是启动成员变更所需检测到的进程的最小数量。若成员变更的排除和纳入协议的初始投票阈值h'_0 满足 $h'_0 > \frac{d+t+n}{2}$ 以保证安全性，并满足 $h'_0 \leq n-f+f_d$ 以保证活性，则这些协议可实现共识。

证明。诚实进程通过问责性检测到故障进程后，首先在本地将其排除 f_d。根据 Basilic 的问责性（引理 5.20），这些进程至少包括 f_d 个被检测到的故障进程。因此，不失一般性，我们假设恰好有 f_d 个进程被检测并排除（如果数量更多，那么由于主动问责性，共识问题会更加容易解决）。据此，我们定义 $d'+t' = d+t-f_d$ 和 $n' = n-f_d$。排除协议执行一个投票阈值为 h'_0 的 Basilic 实例，但实际上它以 $h'(f_d) = h'_0 - f_d$ 开始，因为该实例在检测到 f_d 个故障进程时启动。因此，该 Basilic 实例在安全性方面能够为 $h'(f_d) > \frac{d'+t'+n'}{2}$ 解决共识问题，并在活性方面能够为 $h'(f_d) \leq n'-q-t'$ 解决共识问题（定理 5.2）。

因此，我们首先考虑安全性界限。$h'(f_d) > \frac{d'+t'+n'}{2} \iff h'(f_d) > \frac{d+t+n-2f_d}{2} \iff h'_0 > \frac{d+t+n}{2}$，因为成员变更开始时已具备优势：在启动之前检测到了 $f_d = 2h_0 - n$ 个故障进程。对于活性界限，请注意，成员变更开始时检测到的拜占庭进程最少为 $t-t' \geq f_d - d$。因此，$h'(f_d) \leq n'-q-t' \iff h'(f_d) \leq n-f_d-q-t-d+f_d \iff h'(f_d) \leq n-f \iff h'_0 \leq n-f+f_d$。 □

定理 6.4（◇-排除/纳入协议共识）。令 ZLB 以初始投票阈值 h_0 执行，该阈值为启动成员变更所需检测到的进程的最小数量。那么排除与

成员变更的纳入协议可解决 $\diamondsuit$-共识，前提是其投票阈值 h_0 满足安全性条件 $h'_0 > d + t$ 和活性条件 $h'_0 \leq n - f + f_d$。

证明。 诚实进程通过问责性检测到故障进程后，会在本地将其排除，从而启动排除协议。根据 Basilic 的问责性（引理 5.20），这些至少是 f_d 个已检测到的故障进程。排除协议会运行一个 Basilic 实例，其投票阈值为 h'，该实例可解决 $\diamondsuit$-共识：安全性为 $d' + t' < h'(f_d)$，活性为 $h'(f_d) \leq n' - q - t'$（定理 5.4），其中 $n' = n - f_d$ 且 $d' + t' = d + t - f_d$。

因此，我们首先考察安全性界限。 $d'+t' < h'(f_d) \iff h'(f_d) > d+t-f_d \iff h'_0 > d+t$，这是通过将 $d'+t'$ 替换为 $d+t-f_d$ 得出的；此外，成员变更开始时就已经检测到 $f_d = 2h_0-n$ 个故障进程，因此具有这一优势。对于活性界限，请注意，成员变更开始时至少会检测到 $t-t' \geq f_d-d$ 个拜占庭进程。因此， $h'(f_d) \leq n'-q-t' \iff h'(f_d) \leq n-f_d-q+f_d-t-d \iff h'(f_d) \leq n - f \iff h'_0 \leq n - f + f_d$。 □

对于 $h = 2n/3$ 的 ASMR 共识的标准投票阈值而言，这意味着排除与纳入协议分别存在两个不同的最优投票阈值 h'，具体取决于我们选择让成员变更解决共识还是最终共识。这些阈值分别为：共识对应 $h'_0 = 7n/9 = 2n'/3$，最终共识对应 $h'_0 = 2n/3 = n'/2$。下面我们将讨论这两种选择及其优缺点。此外，我们还提出了一个针对另一种性质——感知性——具有韧性最优性的附加阈值。

解决最终共识的成员变更 界限 $h'_0 = 2n/3 = n'/2$ 意味着排除与纳入协议能够解决最终共识，如定理 6.4 所示。该界限的优势在于，当 $\delta < 2/3$ 时，欺骗性比率 δ 达到最优。对于 $\delta = 2/3 = h/n$，故障进程甚至无需与诚实进程通信，就能造成不一致；这意味着它们可以造成无限次不一致，无法满足收敛性。因此，对于这一投票阈值 h'，可容忍故障的总数为 $f < 2n/3$，且 $q + t \leq n/3$, $d + t < 2n/3$。遗憾的是，由于排除与纳入协议只能解决最终共识，而不能解决共识，这意味着某些进程可能会暂时对要排除和纳入的进程意见不一。不过，最终所有进程都会对要纳入和排除的同一组进程达成一致，如第 5.6 节所示。此外，排除协议上的不一致对对手不利，因为诚实进程最终会排除更多故障进程。纳入协议上的不一致也对对手不利，因为对于要纳入哪些进程的不一致，会迫使对手暴露更多故障进程。相较之下，这些攻击者可以等到 ASMR 共识出现不一致时，再暴露自身为故障进程，这对攻击者更有利。尽管如此，我们现在将展示另一个投票阈值h'，它可以解决这一成员变更漏洞。

通过成员变更解决共识 设置投票阈值 $h'_0 = 7n/9 = 2n'/3$ 使排除与纳入协议能够解决共识，如定理 6.3 所示。与之前的情形相比，该投票阈值使诚实进程能够确定它们在排除与纳入协议的决策上达成一致。这意味着，如果纳入协议仅包含诚实进程，那么在成员变更结束时，对手将无法再造成任何不一致；只要所有诚实进程都已开始成员变更，此后便能保证一致性。

此类方法的缺点是，对于该阈值，可容忍故障的总数为 $f < 5n/9$，其中 $q + t \leq n/3,\ \ d + t < 5n/9$。此外，该投票阈值不足以保证成员变更结束后不会出现不一致，因为一些诚实进程甚至可能尚未感知到成员变更的存在，因此仍可能使用包含 $f < 5n/9$ 个故障进程的过时委员会。基于这一原因，我们定义了一个新的性质，将其称为感知性。

定义 6.2（感知性）。 假设纳入协议仅包含诚实进程。假设成员变更开始。若所有诚实进程都能确定 $k > 0$，使得 Φ_l 将在对手的静态时期内解决共识 $\forall l \geq k$，则 ZLB 满足感知性。

感知性的定义严格强于收敛性的定义，因为仅仅让诚实进程知道它们最终能够解决共识是不够的；它们还必须知道自己何时不再只是解决最终共识，而是开始解决共识（前提是纳入协议不会将欺诈比例恢复到成员变更之前的水平）。感知性也严格强于成员变更的 α确认，因为感知性还保证：那些甚至尚未听说成员变更、因而仍使用过时委员会决定区块的其余诚实进程，在一些诚实进程终止成员变更后，无法再使用过时委员会作出决定。

定理 6.5。ZLB 在 $d + t < h_0 + h'_0 - n$ 时解决感知性问题。

证明。令 O 为尚未听说成员变更的诚实进程集合。若 $|O| + d + t \geq h_0$，则 O 中的进程足以终止共识实例 Φ_k，并决定 v、 $k > 0$。令 O' 为已启动成员变更的诚实进程集合。于是，如果 $|O'| + d + t \geq h'_0$，成员变更便可以终止；并且由于 $h'_0 \geq h_0$，按照构造，一旦 O' 中的进程能够终止成员变更，它们也能够终止共识实例 Φ_k，并决定 v'， $v' \neq v$。因此，我们计算 f 取何值时，$|O| + d + t \geq h_0$ 和 $|O'| + d + t \geq h'_0$不可能同时成立。通过求解方程组可知，要使两者都有可能，则$d + t \geq h_0 + h'_0 - n$，这意味着对于 $d + t < h_0 + h'_0 - n$，要么 O 中的进程能够终止 Φ_k 并决定 v，要么 O' 中的进程能够终止并决定 v'，但两者不能同时发生。

□

根据定理 6.5，投票阈值 $h'_0 = 7n/9 = 2n'/3$ 在解决 $f < 5n/9$ 的排除与纳入协议的共识问题时，仅能满足 $f < 4n/9$ 的感知性。相反，将投票阈值设为 $h'_0 = 5n/6 = 5n'/9$，则可以解决 $f < n/2$ 的排除与纳入协议的共识问题，并满足 $q + t \leq n/3$，$d + t < n/2$，同时满足针对同一对手的感知性。我们将在第 6.6 节中讨论 ZLB 的这三种初始设置，并将其与最新技术水平进行比较。

6.5.3 LLB 的 ZLB 证明

在本节中，我们将证明，无论上一节中展示的三种可能初始参数为何，ZLB 都能解决 LLB。

引理 6.1。 如果 $d+t < \min(h_0, h'_0)$ 且 $h'_0 \leq n-f+f_d$，那么 ZLB 中的每次不一致都会导致一次成员变更，其纳入和排除协议最终会解决共识问题。

证明。 如果 $d+t \geq h_0$，那么故障进程可以在不与诚实进程通信的情况下造成不一致，这意味着这些不一致不会被检测到，成员变更也不会启动。因此可得 $d + t < h_0$。定理 6.4 表明，要使成员变更解决最终共识，必须满足 $d + t < h'_0$ 和 $h'_0 \leq n - f + f_d$。 □

定理 6.6。 若 $q+t \leq n - h_0$、$d + t < \min(h_0, h'_0)$ 且 $h'_0 \leq n - f + f_d$，则 ZLB 对 $f = d+q+t$ 个总故障满足收敛性。

证明。根据引理 6.1，每次成员变更都解决最终共识。剩余界限 $q+t \leq n-h_0$ 源于 ASMR 共识至少必须解决最终共识这一事实（引理 5.21）。如果从一开始就是 $d + t < 2h_0 - n$，那么就不会出现不一致（定理 5.2），因此可以保证收敛性。相反，对于 $2h_0 - n \leq d + t < \min(h'_0, h_0)$ 且如果 $f \leq n - h'_0 + f_d$，根据引理 6.1，每次不一致都会导致一次解决最终共识的成员变更。纳入协议不会增加欺诈比例，因为纳入协议包含的进程数不会超过排除协议排除的进程数（这得益于确定性函数 choose），而且所有被排除的进程都是故障进程。

由于包含共识至少决定 $h'(d_r) = h'_0 - d_r$ 个提案和$d + t < h'_0$（因为我们通过决定所有提案的并集（其关联位被决定为 1）来实现 Basilic，以解决 SBC），因此可以推知，来自诚实进程的一些提案将被决定。由于加入候选者池是有限的，且没有任何进程会被包含多次，因此在最坏情况下，池中的所有故障进程都至少被包含过一次；从此以后，所有诚实进程都只提议包含池中的诚实进程。此时，在慢适应性对手的一个静态时期内，每次新的成员变更都会使欺诈比例下降。

因此，某个包含共识最终会导致欺诈比例 $\delta n < 2h_0 - n$，从此之后共识便得以满足。令 Φ_k 为第一个 ASMR

共识，使得 $\delta n < 2h_0 - n$。由于 $d + t < h_0$ 和 $q + t \leq n - h_0$，所有先前的迭代 $k' < k$ 都解决了最终共识（定理 5.4）。 □

推论 6.1。ZLB 在满足 $d + t < h'_0$ 和 $h'_0 \leq n - f + f_d$ 的任意 h'_0 下，对 $q + t \leq n - h_0$ 实现 h_0 的持久区块链，并对 $d + t < h_0$ 实现持久区块链。

证明。对于 $d + t < h_0$、$q + t \leq n - h_0$，根据定理 5.4，ASMR 共识解决了最终共识问题，满足终止性。若 $d + t < 2h_0 - n$，则根据定理 5.2，ASMR 共识解决了共识问题，满足一致性。最后，定理 6.6 展示了收敛性。 □

6.6 对比表

表 3 展示了 ZLB 与前文提到的成员变更所涉及的三种投票阈值之间的比较。值得注意的是，Basilic 在初始投票阈值为 $h_0 = \frac{2n}{3}$ 时，是唯一能在故障数超过超多数的情况下解决 $\Diamond$-共识的方案，这得益于它在 BDB 模型中对故障的刻画。我们根据成员变更的初始投票阈值 h'_0 列出了 ZLB 的三种配置，但 ASMR 共识的初始投票阈值均设为 $h_0 = \frac{2n}{3}$。这三种配置正是我们在第 6 节 6.5.2 中讨论的配置。不过，在这三种情况下，值得注意的是，只有 Basilic 和 ZLB 既能在 BFT 模型中针对韧性最优的故障数 $3t < n$ 解决共识问题，又能在故障数超过 $3t < n$ 时解决最终共识问题。此外，只有 ZLB 的三种配置能够在总故障数为 $3f \geq n$、其中拜占庭故障最多为 $t = t_\ell$ 时解决 $\Diamond$-共识（如表下注释所示），同时还能在部分同步条件下针对韧性最优界限 $3t < n$ 解决共识问题。正如第 6 节 6.5.2 所述，ZLB 的三种配置在感知性列中有所不同。有些工作在表中占据多行 [60, 56]，因为其故障容忍能力和假设会随初始配置而变化。

6.7 实验评估

本节将回答以下问题：ZLB 在地理分布式环境中能否实现实用性能？当 $f < n/3$ 时，与启发 Facebook Libra [9] 的 HotStuff 状态机复制方案以及近期推出的快速 Red Belly 区块链 [28] 相比，ASMR 的表现如何？大规模联盟攻击会对 ASMR 的恢复造成什么影响？我们将在第 7 节评估零损失支付应用。

选择合适的区块链进行比较。由于我们为开放网络提供解决方案，因此不能依赖其他区块链所采用的同步性假设 [37]。由于我们需要达成共识，必须假设消息延迟存在一个未知上界 [33]，，因此不与终止性尚未得到证明的随机化区块链 [57, 31, 39, 53] [70]进行比较。这就是我们将评估重点放在部分同步区块链上的原因。因此，我们

区块链	N.	共识 Byz.	共识 总计	◇-共识 Byz.	◇-共识 总计	LLB	感知性	Acc.	罚没	Z.l.	Act.
[59, 35]	S.	0	0	$2t < n$	$2f < n$	✗	✗	✓[35]	✓[35]	✗	✗
[9, 28]	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	✗	✗	✗	✗	✗	✗
[60] (P)	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	✗	✗	✗	✗	✗	✗
[60] (S)	S.	$2t < n$	$2f < n$	$2t < n$	$2f < n$	✗	✗	✗	✗	✗	✗
[56] (1)	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	✗	✗	✗	✗	✗	✗
[56] (2)	P.	0	$f < \frac{2n}{3}$	0	$f < \frac{2n}{3}$	✗	✗	✗	✗	✗	✗
[27, 4]。	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	✗	✗	✗	✓	✗	✗
[13, 65, 64]	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	✗	✗	✓	✓	✗	✗
[21]	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	✗	✗	✓	✗	✗	✗
[6]	P.	0	0	$2t < n$	$2f < n$	✗	✗	✓	✗	✗	✗
Basilic ($h_0 = \frac{2n}{3}$)	P.	$3t < n$	$f < \frac{2n}{3}$ †	$3t < n$	$f < n^{\ddagger}$	✗	✗	✓	✗	✗	✓
ZLB ($h'_0 = \frac{7n}{9}$)	P.	$3t < n$	$f < \frac{2n}{3}$ †	$3(q+t) < n$,	$d+t < \frac{5n}{9}$ §	✓¶	$d+t < \frac{4n}{9}$	✓	✓	✓	✓
ZLB ($h'_0 = \frac{2n}{3}$)	P.	$3t < n$	$f < \frac{2n}{3}$ †	$3(q+t) < n$,	$d+t < \frac{2n}{3}$ §	✓	✗	✓	✓	✓	✓
ZLB ($h'_0 = \frac{5n}{6}$)	P.	$3t < n$	$f < \frac{2n}{3}$ †	$3(q+t) < n$,	$2(d+t) < n$§	✓¶	$2(d+t) < n$	✓	✓	✓	✓

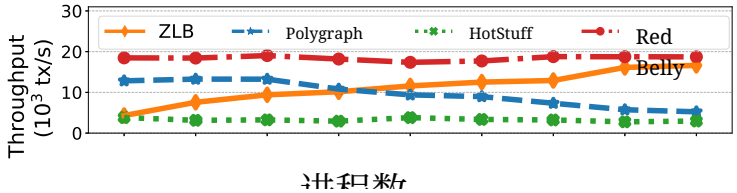
† 实际上，$3(d+t) < n$ 和 $3(q+t) < n$，这意味着如果 $f = \lceil \frac{2n}{3} \rceil - 2$，那么 $t = 0$（图3b）‡ 实际上，$d + t < \frac{2n}{3}$ 和 $3(q+t) < n$，这意味着如果 $f = n - 2$，那么 $t = 0$（定理 5.4）§ 除了 ‡ 之外，它还构建于 Basilic 之上

¶ 成员变更解决的是共识，而不仅仅是 ◇-共识

表3：ZLB 与先前工作的比较表，其中 N. 表示网络假设（S. 表示同步，P. 表示部分同步），Byz. 表示可容忍的拜占庭故障，Acc. 表示准确性，Z.l. 表示零损失，Act. 表示主动问责性。

评估了 Facebook Libra [9], ，但其性能仅为每秒 11 笔交易，原因似乎是 Move 虚拟机带来的开销。因此，我们在此省略这些结果，转而关注其原始状态机复制（SMR）算法 HotStuff，以及其现有的 C++ 代码；此前研究表明，该算法可降低传统拜占庭容错状态机复制的通信复杂度 [77]（我们采用未经修改的原始实现及其默认配置 [76]）。我们还评估了近期推出的可扩展 Red Belly 区块链 [28]（RBB），以及 Polygraph 协议 [21]；据我们所知，它是唯一已实现的可问责共识协议。不过，该协议无法容忍超过 $n/3$ 个故障，因为检测到故障后它无法恢复。

地理分布式实验设置。我们在两种分布式设置中部署这四个系统，使用配备 4 个 vCPU 和 7.5 GiB 内存的 c4.xlarge 亚马逊云科技（AWS）实例：（i）最多包含 100 台机器的局域网，以及（ii）最多包含 90 台机器的广域网。我们评估 ZLB 在故障数量 f 达到 $\lceil \frac{2n}{3} \rceil - 1$ 时的表现；但若未另行说明，我们固定 $f = d = \lceil 5n/9 \rceil - 1$ 和 $q = 0$。由于选择不同的 $h'_0 \in [2n/3, n]$ 对吞吐量的影响可以忽略不计，本节中我们固定 $h'_0 = 7n/9$。请注意，对于该阈值，如果所有欺骗性故障都在成员变更开始时被检测到，即 $f_d = 2n/3$，那么实际上我们可以容忍 $d < 2n/3$ 个欺骗性故障。如果所有攻击者串通起来，尽可能增加其造成不一致的分支数量，就可能出现这种情况；我们的攻击正是如此。所有误差线



进程号数

图5：ZLB 与 Polygraph [21], 、HotStuff [77]和 Red Belly 区块链 [28]的吞吐量对比。

表示 95% 置信区间，图中绘制的值为 3 至 5 次运行的平均值。所有交易均为 ~ 400 字节的比特币交易，并带有 ECDSA 签名 [59]。

6.7.1 ZLB 与 HotStuff、Red Belly 和 Polygraph 的比较

图5比较了 ZLB、RBB、Libra 和 Polygraph 在分布于两大洲、五个可用区的环境中的性能：加利福尼亚州、俄勒冈州、俄亥俄州、法兰克福和爱尔兰（与 Polygraph 实验 [21] 完全相同）。对于 ZLB，我们只展示决策吞吐量，该吞吐量在 $n = 90$ 时达到 16，626 笔交易/秒，因为确认吞吐量与之相近（16，492 笔交易/秒）。由于只有 ZLB 能容忍$f \geq n/3$，因此我们在此比较中固定 $f = 0$。

首先，Red Belly 区块链的吞吐量最高。正如预期，由于缺乏问责性，它的性能优于 ZLB：它无需在消息中附带证书来检测欺诈证明。两种方案都能解决 SBC，因此随着提案数量增加，它们能够决策更多交易（txs），并且每个提案都使用相同的批次大小，即 10，000 笔交易。由此可见，ASMR 的扩展性相当出色：即使在 90 个进程的情况下，容忍 $f \geq n/3$ 个故障的成本似乎也微不足道。

其次，即使 HotStuff 不验证交易，它的吞吐量仍然最低。请注意，HotStuff 使用其专用客户端在默认配置下进行基准测试；这些客户端将提案发送给所有服务器，让服务器之间只交换每笔交易的摘要，从而节省带宽。其性能可以解释为：无论提交了多少交易，HotStuff 每个共识实例只决定一个提案（即一批 10,000 笔交易），这与先前的观察结果一致 [73]。相比之下，随着 n 增大，ZLB 的速度会加快，并在 $n = 90$ 时以 $5.6\times$ 超越 HotStuff，这要归功于超级区块优化，该优化使 ZLB 能够在其多值共识的每个实例中一次决定多个提案 [28]。

最后，在小规模场景下，Polygraph 比 ZLB 更快，因为 Polygraph 的分布式验证和可靠广播实现 [21] 无法追责，因此执行的验证次数更少。当进程数超过 40 时，Polygraph 反而变得更慢，这是因为我们做了优化：例如，它的 RSA 验证比我们的 ECDSA 签名更大，且消耗更多带宽。

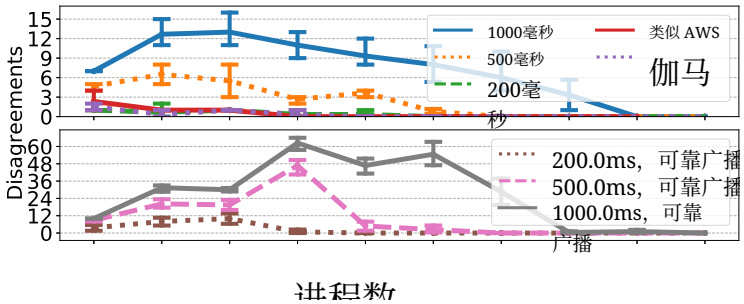


图 6：在投票支持某项决策时进行欺骗（上图），以及在广播提案时进行欺骗（下图）的情况下，对于各种均匀延迟、由伽马分布生成的延迟，以及从观测到的 AWS 延迟中抽样得到的分布，所产生的不一致的决策，适用于 $f = d = \lceil 5n/9 \rceil - 1$。

6.7.2 尽管遭受联盟攻击，ZLB 仍具备可扩展性

为了评估 ZLB 在发生故障时的表现，我们实现了以下两种可能的合谋攻击。在 SBC 问题的解法（定义 3.1）中，故障进程可以组成一个由 $f \geq n/3$ 个进程构成的联盟，通过发送冲突消息，导致诚实进程产生不一致；联盟攻击有以下两种：

1. **可靠广播攻击：** 故障进程在可靠广播期间行为异常，向不同分区发送不同的提案，导致诚实进程最终在同一索引 k 处得到不同的提案。例如，故障进程向一部分 A 诚实进程发送包含交易 tx_a 的区块 b_a，同时向另一部分 B 诚实进程发送包含冲突交易 tx_b 的区块 b_b，$A \cap B = \emptyset$，且两者位于同一索引 k 处。

2. **二元共识攻击：** 故障进程在同一个二元共识中，分别针对两个分区中的每个二元值进行投票，导致诚实进程在其位掩码的同一索引处决定不同的比特；其中，在位掩码索引 k 处决定为 1（相应地为 0）意味着在 ZLB 中包含（相应地不包含）索引为 k 的提案。例如，故障进程向诚实进程的一个子集 A发送决定为 1 和 0 的消息，同时向诚实进程的另一个子集 B 发送决定为 0 和 1 的消息，其中，$A \cap B = \emptyset$分别对应于与包含交易 tx_a 的区块 b_a 以及包含冲突交易 tx_b的区块 b_b相关联的二元共识实例。

请注意，故障进程无法从组合这些攻击中获益：如果两个诚实进程在索引 k处交付了不同的提案，则不一致源于它们在相应的二元共识实例中输出了 1。同样地，只有当两个诚实进程在索引 k处拥有相同的对应提案时，迫使它们在第 k个二元共识中产生不一致才有意义。

为了扰乱诚实进程分区之间的通信，我们根据均匀分布和伽马分布，在分区之间注入随机通信延迟，并采用了先前发表的研究中获得的 AWS 延迟。

测量轨迹 [58, 29, 28]。（攻击者与每个分区正常通信。）

图 6（上）展示了由二元共识攻击导致的不一致数量，即诚实进程决定的不同提案数。首先，我们在两个分区之间选取 200、500 和 1000 毫秒的均匀分布延迟。然后，我们选取参数取自 [58, 29] 的伽马分布延迟，以及一种随机抽取此前测得的 AWS 区域之间固定延迟的分布 [28]。我们自动计算欺诈故障规模能够创建的最大分支数（例如，对于 $d < 5n/9$，为 3 个分支），然后为每个分支创建一个诚实进程分区，并在任意两个分区之间应用这些延迟。

有趣的是，我们观察到一致性属性具有可扩展性：在保持欺诈比例不变的情况下，进程数越多，攻击者就越难以造成不一致。这种可扩展性现象是由于规模扩大时，攻击者之间的通信延迟不可避免地增加，从而为诚实进程的划分提供了相对更多时间来检测欺骗进程，进而限制不一致的数量。采用期望值低于均匀延迟的、更符合实际的网络延迟（伽马分布和 AWS 延迟）时，欺骗进程几乎无法造成哪怕一次不一致。这证实了我们系统的可扩展性。

图 6（下）描绘了可靠广播攻击下的不一致数量。与二元共识攻击期间相比，该攻击期间的不一致数量要高得多。不过，随着系统规模扩大，不一致数量下降得更快，因为攻击者会更早暴露自己。

6.7.3 由故障和延迟导致的不一致

现在，我们评估规模更大的联盟和延迟对 ZLB 的影响。在一个进程数从 20 增至 100 的系统中，我们测量随着欺诈比例和分区延迟增加而产生的不一致数量。请注意，理论上可通过中间人攻击实现这些延迟，但由于矿池的自治系统之间存在直接对等连接，在真实区块链中实现这些延迟极为困难 [34]。

尽管对于现实但并非灾难性的延迟，ZLB 对攻击具有相当强的抵御能力（图 6），但攻击者可以尝试在不同区域之间的网络瘫痪数秒时发动攻击。我们的实验结果见图 8：对于二元共识攻击，当诚实进程的分区之间存在 10 秒的均匀延迟时，攻击者最多可造成 52 个提案不一致；当均匀延迟为 5 秒时，最多可造成 33 个不一致，$n = 100$。进一步的测试表明，在 5 秒的均匀延迟下，可靠广播攻击最多可造成 165 个提案不一致。

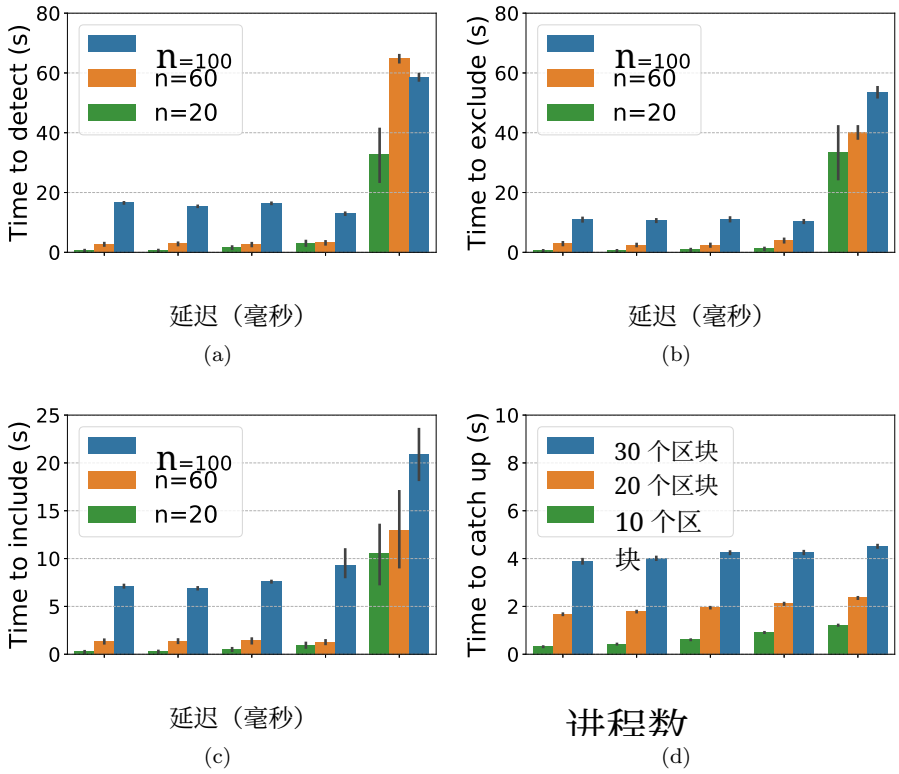


图 7：（从左到右、从上到下）按延迟分布和进程数统计检测$\lceil \frac{n}{3} \rceil$欺骗性进程、排除它们、包含新进程所需的时间；以及按区块数和进程数统计追赶所需的时间，并带有$f = d = \lceil 5n/9 \rceil - 1$。

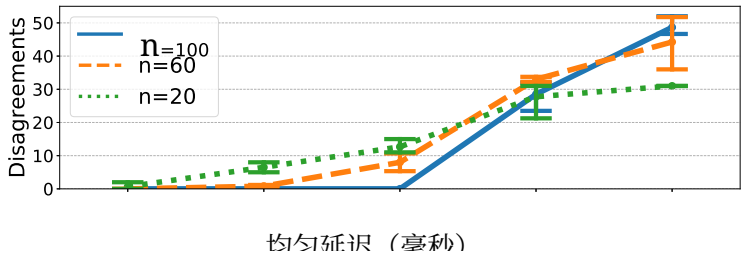


图 8：在二元共识攻击下，不同灾难性均匀延迟对应的不一致决策，针对 $f = d = \lceil 5n/9 \rceil - 1$。

6.7.4 合并区块和更换成员所需的时间

为了更深入地了解 ZLB 延迟的原因，我们测量了合并区块以及通过新进程替换欺骗进程以更换成员所需的时间。这里展示的是在最坏情况下（所有交易都冲突），合并不同大小的两个区块所需的本地时间。这是最坏情况下所需的时间，因为进程可以合并它们同时收到的提案（即，无需暂停共识）。我们的实验表明，合并分别包含 100、1000 和 10000 笔交易的两个区块，所需时间分别为 0.55、4.20 和 41.38 毫秒。显然，与运行共识算法所需的时间相比，本地合并区块所需的时间可以忽略不计。

图 7 展示了在不同延迟和进程数下，检测欺骗进程所需的时间（左上）、运行排除共识所需的时间（右上）以及运行纳入共识所需的时间（左下）。检测时间是指从攻击开始到诚实进程检测到攻击所经过的时间：如果最初的 f_d 个欺骗进程结成联盟并造成不一致，那么检测到第一个欺骗进程和第一个 f_d 个欺骗进程所需的时间是重叠的。（我们会同时检测到所有这些进程。）在通信延迟较大的情况下，排除共识所需的时间（57 秒）明显长于纳入共识（21 秒），这是因为排除共识的提案包含欺诈证明，导致领导进程执行耗时的加密验证。通信延迟较短时，性能便切实可用。最后，图 7（右下）展示了追赶所需的时间如何随提案（即区块）数量变化。正如预期，随着进程数增加，这一时间线性增长，因为追赶需要验证更大的证书；但对于 $n = 100$ 个进程，该时间仍然切实可用。使用证书的优势在于，进程只需验证证书即可追赶，而不必验证证书所指区块中的所有交易。

7 零损失支付应用

本节将介绍如何利用 ZLB 实现一种零损失支付系统，使任何诚实进程都不会损失任何货币。其核心思路是要求共识进程存入足够数量的货币，以便

在遭受攻击时，使用欺骗进程的货币，从而避免任何诚实进程蒙受损失。

7.1 假设

为了衡量合谋攻击以概率 ρ in 通过引导共识走向不一致、从而成功分叉 ZLB 所产生的预期影响，我们首先需要作出以下假设：

1. **可替代资产**。 我们假设用户可以转移资产（例如货币），这些资产具有可替代性，即同一值的任意两个单位可以互换，且无法彼此区分。加密货币就是可替代资产的一个例子。

这一零损失支付系统也可以与非同质化代币（NFT）和智能合约配合使用，但有一个例外：对于同一个 NFT（或不一致的状态）的两个接收方，其中一方会发现自己的 NFT 被收回（或其已返回的状态被撤销），并获得事先约定的补偿，以弥补由此带来的不便。

2. **存款退款**。 为限制一次成功的双重支付对某个区块的影响，ZLB 会将存款保留若干个区块 w后再退还。交易在达到此区块深度 w之前，不应被视为最终确认 （即不可逆）。我们将此称为 w 最终确认区块深度。攻击者可以分叉成 a 个分支，并尝试多次花费金额 $\mathfrak{G}$ （每个区块），我们将其称为收益，从而获得最大收益（$a-1$）$\mathfrak{G}$。每个诚实进程都可以通过累加其已确定区块中所有交易的输出，计算出收益。此外，进程可以通过设计将收益限制在一个上限内，丢弃输出总和超过该上限的区块；也可以允许收益高达全部流通资产。存款 $\mathfrak{D}$ 是收益的一个因子，即 $\mathfrak{D} = b \cdot \mathfrak{G}$。目标是让每个联盟至少存入 $\mathfrak{D}$，而由于每个联盟的规模至少为 $\lceil n/3 \rceil$，这意味着每个进程都必须存入金额 $3b\mathfrak{G}/n$。

3. **网络控制限制**。 一旦故障进程选定遭受不一致的诚实进程互不相交子集（即分区），我们就需要防止故障进程的通信速度无限快于不同分区中的诚实进程。更形式化地，令 X_1 （分别为 X_2）表示随机变量，用于指示同一分区内两个进程之间（分别为来自不同分区的两个诚实进程之间）传递一条消息所需的时间。对于某个 $\varepsilon > 0$，我们有 $E(X_1)/E(X_2) > \varepsilon$。请注意， X_1 的定义也意味着，它是同一分区内两个诚实进程或两个故障进程之间通信时间的随机变量。此概率同步假设与其他区块链（例如比特币）的假设类似，这些区块链保证快速指数收敛；在相同假设下，ZLB 也具有这一结果。接下来，我们将分析重点放在针对每次共识迭代的攻击上；即使分叉只持续很短时间，只要单个共识实例中出现分叉，我们就视为发生了成功的不一致

时间。我们将在第 7.3 节讨论如何使用随机信标进行委员会抽选，以便在部分同步通信网络中实现零损失。

7.2 理论分析

我们证明，攻击者筹集的资金始终不少于其盗取的资金。为便于说明，我们假设成员变更要么在存款退款前启动，要么不启动。因此，该攻击可视为一次伯努利试验，其成功概率为 ρ （每个区块），该概率可由 ε 推导得出。每次攻击尝试中，攻击者通过分叉为 a 个分支，最多可获得 $(a-1)\mathfrak{G}$ 个货币；或者至少损失 $\mathfrak{D}$ 个货币作为惩罚，这些资金可用于填补成功攻击所盗取的资金。

我们引入随机变量 Y，用于衡量攻击成功所需的尝试次数，该变量服从均值为 $E(Y)=\frac{1}{\hat{\rho}\hat{\rho}}$ 的几何分布，其中 $\hat{\rho}=1-\rho$ 是攻击失败的概率。因此，我们定义攻击的预期收益为：$\mathcal{G}(\hat{\rho})=(a-1)\cdot(\mathrm{P}(Y>w)\cdot\mathfrak{G})$，预期惩罚为：$\mathcal{P}(\hat{\rho})=\mathrm{P}(Y\le w)\cdot\mathfrak{D}$。由此，我们可以将每次攻击尝试的预期存款流动定义为两者之差 $\xi=\mathcal{P}(\hat{\rho})-\mathcal{G}(\hat{\rho})$。定理 7.1 给出了 ZLB 实现零损失时对应的取值。

定理 7.1（零损失支付系统）。令 ρ 表示每个区块发生攻击成功的概率，$\mathfrak{D}$ 表示每个联盟的最低存款，以收益上界 $\mathfrak{D}=b\mathfrak{G}$ 的倍数表示；w 表示返还存款所需的最终确认区块深度。若 $g(a,b,\rho,w)=(1-\rho^{w+1})b-(a-1)\rho^{w+1}\ge 0$，则 ZLB 实现零损失支付系统。

证明。回想一下，成功攻击的最大收益为 $\mathfrak{G}\cdot(a-1)$，而在一次不一致尝试中，攻击者的预期收益 $\mathcal{G}(\hat{\rho})$ 和惩罚 $\mathcal{P}(\hat{\rho})$ 如下：

$$\begin{aligned}\mathcal{G}(\hat{\rho}) &= (a-1)\cdot(\mathbb{P}(Y>w)\cdot\mathfrak{G}) = (a-1)\cdot(\rho^{w+1}\cdot\mathfrak{G}),\\ \mathcal{P}(\hat{\rho}) &= \mathbb{P}(Y\le w)\cdot\mathfrak{D} = (1-\rho^{w+1})\mathfrak{D} = (1-\rho^{w+1})b\mathfrak{G}.\end{aligned}$$

因此，存款流量 $\xi=\mathcal{P}(\hat{\rho})-\mathcal{G}(\hat{\rho})$：

$$\xi=\big((1-\rho^{w+1})b-(a-1)\rho^{w+1}\big)\mathfrak{G}=g(a,b,\rho,w)\mathfrak{G}.$$

如果 $\xi<0$，则系统需承担 $\mathcal{G}(\hat{\rho})-\mathcal{P}(\hat{\rho})$ 的成本；否则，惩罚足以为冲突提供资金。由于收益非负 $\mathfrak{G}\ge 0$，可得 $g(a,b,\rho,w)\ge 0$ 对于 $\xi\ge 0$ 成立，从而实现零损失。□

7.2.1 最终确认区块深度和存款规模

设置 $c=\frac{b}{a-1+b}$ 后，我们可以计算 ZLB 在给定最终确认区块深度 w 下所能容忍的攻击成功概率 $\rho\le c^{\frac{1}{w+1}}$，或者计算在概率 ρ 下实现零损失所需的最终确认区块深度 $w\ge\frac{\log(c)}{\log(\rho)}-1$。

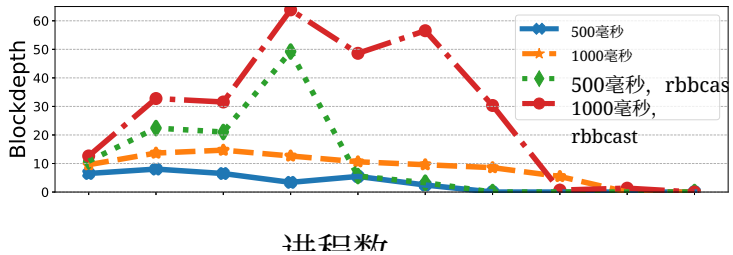


图 9：对于 $\mathfrak{D} = \mathfrak{G}/10$、$f = d = \lceil 5n/9 \rceil - 1$ 和 $q = 0$，获得零损失所需的最小最终确认区块深度为 w。

我们固定存款 $\mathfrak{D}$ 并为收益 $\mathfrak{G}$设定上限。例如，对于 $\delta = 0.5$ 则$a = 3$；若概率为 $\rho = 0.55$，最终确认区块深度为 $w = 4$ 个区块，即使存款仅为最大收益 $\mathfrak{D} = \mathfrak{G}/10$ 的十分之一，也能保证零损失；但若 $\rho = 0.9$ 则 $w = 28$。虽然 a 随ρ多项式增长，但当欺诈比例 δ 接近渐近极限 $2/3$时，它会呈指数增长，导致 $\delta = 0.6$时需要 $w = 37$ 个区块，而 $\delta = 0.64$时则为 $w = 46$，或者 $\delta = 0.66$时为$w = 58$，其中 $\rho = 0.9$ 且 $\mathfrak{D} = \mathfrak{G}/10$。

7.2.2 支付系统的实验评估

基于第 6.7 节的实验结果以及前述理论分析，图 9 展示了在 $\mathfrak{D} = \mathfrak{G}/10$、$f = d = \lceil 5n/9 \rceil - 1$ 和 $q = 0$ 条件下，各种均匀通信延迟所需的最小最终确认区块深度 w。同样可以看到，最终确认区块深度会随着进程数增加而降低，这证实了零损失性质具有良好的扩展性。此外，较小的均匀延迟可以在较小的 w 值下实现零损失；对于所有这些延迟，当 $n > 80$ 时，$w < 5$ 个区块即可实现零损失。尽管图中未展示，但我们的实验表明，即使均匀延迟为 10 秒，在二元共识攻击的情况下设置 $w = 50$ 个区块（在可靠广播攻击的情况下设置 $w = 168$ 个区块），仍可实现零损失。不过，如果网络运行正常，ZLB 将支持较大的 f 值，并且实际上还能从攻击中获益，获得足够多的资金来弥补被盗金额。

7.3 关于概率同步的讨论

我们在第 7 节中假设概率同步，以便引入每次共识迭代中攻击失败的概率。在部分同步条件下，由于在欺诈者被识别之前委员会保持静态，对手可以成功实施攻击，其成功概率为 $\rho = 1$。然而，即使在部分同步条件下，仍有其他因素可能影响概率 ρ。例如，考虑区块深度 $w \geq 1$，采用随机信标 [37]，使委员会在每次迭代时都更换，可以降低攻击成功的概率。在这种情况下，攻击成功的概率取决于随机信标选中

足够多的联盟成员（以及诚实进程各分区中足够多的进程），并使其连续迭代 $w+1$ 次，从而使联盟能够再攻击 w 个区块。设计一种能够容忍规模大于 t_ℓ 的联盟的随机信标并证明其性质，是我们未来的工作之一。

8 结论

在本研究中，我们提出了 BDB 故障模型。随后，我们介绍了 Basilic 协议类，该协议类针对 BFT 和 BDB 模型下的共识问题均具有韧性最优性，并且通信复杂度最优；这得益于主动问责性这一性质，即欺骗行为不会妨碍活性。我们还证明，在 BDB 模型下，Basilic 共识协议类能够解决 $\Diamond$-共识问题，适用于 $d+t<h_0$ 和$q+t\leq n-h_0$，且满足 $h_0\in(n/2,\ n]$ 为初始投票阈值。

接着，我们介绍了 ZLB，这是首个能够容忍多数故障的区块链。为此，我们首先定义了 LLB 问题，随后详细介绍 ZLB 并证明其正确性。此外，我们构建了 ZLB，并在多数攻击者的情况下对其进行了评估，还将其与先前的工作进行了比较，展现出有竞争力的性能。最后，我们介绍了一种基于 ZLB 构建的零损失支付应用，该应用保证诚实进程或用户不会因暂时分歧而损失任何资金。

我们未来的主要研究方向是将本工作提出的进展应用于随机信标问题，以便在本文提出的模型中设计一种随机信标协议。该模型强于一般的 BFT 模型，可用于轮换委员会，并从我们的零损失支付应用中移除概率同步假设。

参考文献

[1] Ittai Abraham、Philipp Jovanovic、Mary Maller、Sarah Meiklejohn、Gilad Stern 和 Alin Tomescu。《异步分布式密钥生成的共识达成》。发表于 *2021*年*ACM*分布式计算原理研讨会，PODC'21，第 363–373 页，2021 年。

[2] Ittai Abraham、Philipp Jovanovic、Mary Maller、Sarah Meiklejohn、Gilad Stern 和 Alin Tomescu。实现异步分布式密钥生成的共识。载于*2021*年*ACM*分布式计算原理研讨会论文集，第363–373页，2021年。

[3] Amitanand S. Aiyer、Lorenzo Alvisi、Allen Clement、Mike Dahlin、Jean- Philippe Martin 和 Carl Porth。面向协作式服务的 BAR 容错。载于第*20*届 *ACM* 操作系统原理研讨会论文集，第45–58页，2005年。

[4] Zachary Amsden 等。《Libra 区块链》。技术报告，Calibra，2019年。9月25日修订版。[5] Emmanuelle Anceaume、Antonella Del Pozzo、Romaric Ludinard、Maria Potop-Butucaru 和 Sara Tucci-Piergiovanni。区块链抽象数据类型。载于第*31*届 *ACM* 并行算法与体系结构研讨会，SPAA '19，第349–358页，纽约，美国，2019年。计算机协会。[6] Emmanuelle Anceaume、Antonella Del Pozzo、Thibault Rieutord 和 Sara Tucci-Piergiovanni。《区块链终局性》。载于 Quentin Bramas、Vincent Gramoli 和 Alessia Milani 编，第*25*届分布式系统原理国际会议（*OPODIS 2021*），第217卷莱布尼茨国际信息学会议录（*LIPIcs*），第6:1–6:19页，德国 Dagstuhl，2022年。Schloss Dagstuhl – 莱布尼茨信息学中心。[7] Michael Backes 和 Christian Cachin。在包含拜占庭故障、崩溃故障和恢复的计算混合模型中实现可靠广播。载于*IEEE/IFIP DSN*，第37–46页，2003年。[8] Michael Backes 和 Christian Cachin。在包含拜占庭故障、崩溃故障和恢复的计算混合模型中实现可靠广播。载于*DSN*，第3卷，第37–46页，2003年。[9] Mathieu Baudet、Avery Ching、Andrey Chursin、George Danezis、Fran¸cois Garillot、Zekun Li、Dahlia Malkhi、Oded Naor、Dmitri Perelman 和 Al- berto Sonnino。Libra 区块链中的状态机复制。*Libra* 协会，技术报告，2019年。[10] Michael Ben-Or、Ran Canetti 和 Oded Goldreich。异步安全计算。载于第二十五届 *ACM* 计算理论研讨会（*STOC*）论文集，第52–61页，1993年。[11] Alysson Bessani、Eduardo Alchieri、Jo ao Sousa、Andr´e Oliveira 和 Fer- nando Pedone。从拜占庭复制到区块链：共识仅仅是起点。载于*IEEE/IFIP DSN*，第424–436页，2020年。[12] Ethan Buchman。Tendermint：区块链时代的拜占庭容错，2016年。硕士论文。[13] Ethan Buchman、Jae Kwon 和 Zarko Milosevic。关于 BFT 共识的最新消息。技术报告 1807.04938，arXiv，2018年。[14] Vitalik Buterin 和 Virgil Griffith。友好的终局性装置 Casper。技术报告 1710.09437v4，arXiv，2019年1月。[15] Christian Cachin、Klaus Kursawe、Frank Petzold 和 Victor Shoup。安全且高效的异步广播协议。载于*CRYPTO*，第524–541页，2001年。

[16] Christian Cachin、Klaus Kursawe 和 Victor Shoup。君士坦丁堡中的随机预言机：利用密码学实现实用异步拜占庭协议。《密码学杂志》，18(3):219–246，2005。[17] Miguel Castro 和 Barbara Liskov。实用拜占庭容错与主动恢复。*ACM Trans. Comput. Syst.*，20(4):398–461，2002。[18] Benjamin Y. Chan 和 Elaine Shi。Streamlet：教科书式精简型区块链。发表于*AFT*，第1–11页，2020。[19] Pierre Civit、Seth Gilbert 和 Vincent Gramoli。Polygraph：可问责拜占庭协议。密码学电子印本档案，报告 2019/587，2019。https://ia.cr/2019/587。[20] Pierre Civit、Seth Gilbert 和 Vincent Gramoli。简短公告：Polygraph：可问责拜占庭协议。发表于*DISC*，第45:1–45:3页，2020。[21] Pierre Civit、Seth Gilbert 和 Vincent Gramoli。Polygraph：可问责拜占庭协议。发表于*2021*年*IEEE*第*41*届分布式计算系统国际会议论文集（*ICDCS*），第403–413页，2021。[22] Allen Clement、Flavio Junqueira、Aniket Kate 和 Rodrigo Rodrigues。论非矛盾陈述的（有限）效力。发表于*2012*年*ACM*分布式计算原理研讨会论文集，PODC '12，第301–308页，美国纽约州纽约，2012。计算机协会。[23] Allen Clement、Flavio Junqueira、Aniket Kate 和 Rodrigo Rodrigues。论非矛盾陈述的（有限）效力。发表于*PODC*，第301–308页，2012。[24]Allen Clement、Manos Kapritsos、Sangmin Lee、Yang Wang、Lorenzo Alvisi、Michael Dahlin 和 Taylor Riche。Upright集群服务。发表于*ACM SOSP*，第277–290页，2009。[25] Daniel Collins、Rachid Guerraoui、Jovan Komatovic、Petr Kuznetsov、Matteo Monti、Matej Pavlovic、Yvonne Anne Pignolet、Dragos-Adrian Seredinschi、Andrei Tonkikh 和 Athanasios Xygkis。仅通过广播消息实现在线支付。发表于*IEEE/IFIP DSN*，第26–38页，2020。[26] Tyler Crain、Vincent Gramoli、Mikel Larrea 和 Michel Raynal。DBFT：高效的无领导者拜占庭共识及其在区块链中的应用。发表于*NCA'18*，第1–8页，2018。[27] Tyler Crain、Vincent Gramoli、Mikel Larrea 和 Michel Raynal。DBFT：高效的无领导者拜占庭共识及其在区块链中的应用。发表于*2018*年*IEEE*第*17*届网络计算与应用国际研讨会（*NCA*），第1–8页，2018。

[28] Tyler Crain、Christopher Natoli 和 Vincent Gramoli。Red Belly：安全、公平且可扩展的开放式区块链。发表于*IEEE S&P*，2021。[29] Mark E Crovella 和 Robert L Carter。互联网中的动态服务器选择。发表于*IEEE HPCS*，1995。[30] Luciano Freitas de Souza、Petr Kuznetsov、Thibault Rieutord 和 Sara Tucci Piergiovanni。简短公告：问责性与重配置——自愈格协议。发表于*DISC*，第54:1–54:5页，2021。[31] Sisi Duan、Michael K. Reiter 和 Haibin Zhang。BEAT：让异步拜占庭容错切实可行。发表于*CCS*，第2028–2041页，2018。[32] Swan Dubois、Rachid Guerraoui、Petr Kuznetsov、Franck Petit 和 Pierre Sens。实现最终一致性的最弱故障检测器。发表于*2015*年*ACM*分布式计算原理研讨会论文集，PODC '15，第375–384页，美国纽约州纽约，2015。计算机协会。[33] Cynthia Dwork、Nancy Lynch 和 Larry Stockmeyer。部分同步条件下的共识。*ACM*期刊（*JACM*），35(2):288–323，1988。[34] Parinya Ekparinya、Vincent Gramoli 和 Guillaume Jourjon。中间人攻击对以太坊的影响。发表于第*37*届*IEEE*可靠分布式系统研讨会（*SRDS*），第11–20页，2018。[35] 以太坊。以太坊2.0阶段0，2021。https://notes.ethereum.org/@djrtwo/Bkn3zpwxB。[36] Juan Garay、Aggelos Kiayias 和 Nikos Leonardos。比特币骨干协议：分析与应用。发表于*EUROCRYPT*，第281–310页，2015。[37] Yossi Gilad、Rotem Hemo、Silvio Micali、Georgios Vlachos 和 Nickolai Zeldovich。Algorand：面向加密货币的可扩展拜占庭协议。发表于*SOSP*，第51–68页，2017。[38] Guy Golan-Gueta、Ittai Abraham、Shelly Grossman、Dahlia Malkhi、Benny Pinkas、Michael K. Reiter、Dragos-Adrian Seredinschi、Orr Tamir 和 Alin Tomescu。SBFT：面向区块链的可扩展去中心化信任基础设施。发表于*IEEE/IFIP DSN*，第568–580页，2019。[39] Bingyong Guo、Zhenliang Lu、Qiang Tang、Jing Xu 和 Zhenfeng Zhang。Dumbo：更快速的异步拜占庭容错协议。发表于*CCS*，第803–818页，2020。[40] A. Haeberlen、P. Kouznetsov 和 P. Druschel。PeerReview：分布式系统的实用问责机制。发表于*SOSP*，2007。

[41] Dominik Harz、Lewis Gudgeon、Arthur Gervais 和 William J. Knotten-belt。Balance：加密货币存款的动态调整。载于 *CCS*，2019 年。[42]Manos Kapritsos、Yang Wang、Vivien Quema、Allen Clement、Lorenzo Alvisi 和 Mike Dahlin。All about eve：面向多核服务器的执行-验证复制。载于 *USENIX OSDI*，第 237–250 页，2012 年。[43] Kim Potter Kihlstrom、Louise E. Moser 和 P. M. Melliar-Smith。用于解决共识的拜占庭故障检测器。《The Computer Journal》，46(1):16–35，2003 年 1 月。[44] Ramakrishna Kotla、Lorenzo Alvisi、Mike Dahlin、Allen Clement 和 Edmund Wong。Zyzzyva：推测式拜占庭容错。载于 *SOSP*，2007 年。[45] Petr Kuznetsov、Andrei Tonkikh 和 Yan X Zhang。重新审视快速拜占庭共识的最优弹性。载于*2021 年 ACM 分布式计算原理研讨会论文集*，PODC'21，第 343–353 页，2021 年。[46] Leslie Lamport、Robert Shostak 和 Marshall Pease。拜占庭将军问题。《ACM Transaction on Programming Languages and Systems》，4(3):382–401，1982 年。[47] Kfir Lev-Ari、Alexander Spiegelman、Idit Keidar 和 Dahlia Malkhi。Fairledger：面向金融机构的公平区块链协议。技术报告 1906.03819，arXiv，2019 年。[48] Jinyuan Li、Maxwell Krohn、David Mazi`eres 和 Dennis Shasha。安全不可信数据存储库（SUNDR）。载于 *USENIX OSDI*，第 9 页，2004 年。[49] Jinyuan Li 和 David Mazi`eres。拜占庭容错系统中超越三分之一故障副本。载于*USENIX NSDI*，第 10 页，2007 年。[50] Shengyun Liu、Paolo Viotti、Christian Cachin、Vivien Qu´ema 和 Marko Vukolic。XFT：超越崩溃故障的实用容错。载于 *USENIX OSDI*，第 485–500 页，2016 年。[51] Marta Lokhava、Giuliano Losa、David Mazi`eres、Graydon Hoare、Nicolas Barry、Eli Gafni、Jonathan Jove、Rafa l Malinowsky 和 Jed McCaleb。借助 Stellar 实现快速、安全的全球支付。载于 *SOSP*，第 80–96 页，2019 年。[52] Yuan Lu、Zhenliang Lu、Qiang Tang 和 Guiling Wang。Dumbo-mvba：最优多值验证异步拜占庭一致性再探。载于第*39*届分布式计算原理研讨会论文集，PODC '20，第 129–138 页，美国纽约州纽约市，2020 年。计算机协会。

[53] Yuan Lu、Zhenliang Lu、Qiang Tang 和 Guiling Wang。Dumbo-MVBA：最优多值验证异步拜占庭一致性再探。载于 *PODC*，第 129–138 页，2020 年。[54]Loi Luu、Viswesh Narayanan、Chaodong Zheng、Kunal Baweja、Seth Gilbert 和 Prateek Saxena。面向开放区块链的安全分片协议。载于 *CCS*，2016 年。[55] Dahlia Malkhi、Michael Merritt 和 Ohad Rodeh。广域网上的安全可靠组播协议。载于第十七届国际分布式计算系统会议论文集，第 87–94 页。IEEE，1997 年。[56] Dahlia Malkhi、Kartik Nayak 和 Ling Ren。灵活拜占庭容错。载于 *CCS*，第 1041–1053 页，2019 年。[57] Andrew Miller、Yu Xia、Kyle Croman、Elaine Shi 和 Dawn Song。拜占庭容错协议中的蜜獾。载于 *CCS*，第 31–42 页，2016 年。[58] Amarnath Mukherjee。关于互联网负载低频分量的动态及其重要性。技术报告 MS-CIS-92-83，宾夕法尼亚大学，1992 年。[59] 中本聪。比特币：一种点对点电子现金系统，2008 年。http://www.bitcoin.org。[60] J. Neu、E. Tas 和 D. Tse。潮起潮落协议：解决可用性与最终性困境。载于*IEEE S&P*，第 446–465 页，2021 年。[61] Joachim Neu、Ertem Nusret Tas 和 David Tse。可用性与问责性困境及其通过问责工具实现的解决方案。*arXiv* 预印本 *arXiv:2105.06075*，2021 年。[62] Marshall Pease、Robert Shostak 和 Leslie Lamport。在存在故障的情况下达成一致。《ACM 杂志》，27(2):228–234，1980 年。[63] Alejandro Ranchal-Pedrosa 和 Vincent Gramoli。Trap：利用理性参与者的诱饵解决拜占庭共识。载于 *2022*年 *ACM* 亚洲计算机与通信安全会议论文集，ASIA CCS '22，第 168–181 页，美国纽约州纽约市，2022 年。计算机协会。[64] Alex Shamis、Peter Pietzuch、Miguel Castro、Edward Ashton、Amaury Chamayou、Sylvan Clebsch、Antoine Delignat-Lavaud、Cedric Fournet、Matthew Kerner、Julien Maffre 等。PAC：CCF 实用问责机制。技术报告，arXiv，2021 年。[65] Peiyao Sheng、Gerui Wang、Kartik Nayak、Sreeram Kannan 和 Pramod Viswanath。拜占庭容错协议取证。载于 *CCS*，2021 年。

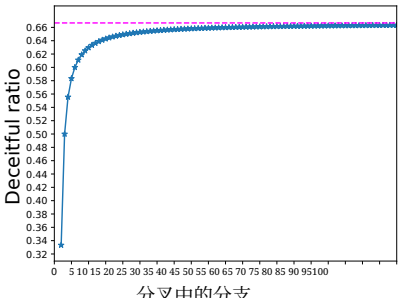
[66] Atul Singh、Pedro Fonseca、Petr Kuznetsov、Rodrigo Rodrigues 和 Pet- ros Maniatis。Zeno：最终一致的拜占庭容错。载于第六届 *USENIX* 网络系统设计与实现研讨会论文集，NSDI'09，第 169–184 页，2009年。[67] Chrysoula Stathakopoulou、Tudor David、Matej Pavlovic 和 Marko Vukoli´c。Mir-bft：适用于去中心化网络的高吞吐量稳健 BFT，2019年。[68] Chrysoula Stathakopoulou、Matej Pavlovic 和 Marko Vukoli´c。让状态机复制的可扩展性变得简单。载于第十七届欧洲计算机系统会议论文集，EuroSys '22，第 17–33 页，美国纽约州纽约市，2022年。计算机协会。[69] Alistair Stewart 和 Eleftherios Kokoris-Kogia。Grandpa：拜占庭最终性工具。技术报告 2007.01560，arXiv，2020年。[70] Pierre Tholoniat 和 Vincent Gramoli。区块链拜占庭容错的形式化验证。载于*FRIDA*，2019年10月。[71] Guillaume Vizier 和 Vincent Gramoli。ComChain：一种支持拜占庭容错重配置的区块链。并发与计算：实践与经验，32(12)，2020年10月。[72] Gauthier Voron 和 Vincent Gramoli。Dispel：采用分布式流水线的拜占庭状态机复制，2019年。[73] Gauthier Voron 和 Vincent Gramoli。Dispel：采用分布式流水线的拜占庭状态机复制。技术报告 1912.10367，arXiv，2019年12月。[74] Gavin Wood。以太坊：一种安全、去中心化的通用交易账本，2015。[75] Yingjie Xue 和 Maurice Herlihy。防范跨链交易中的输不起攻击。载于*2021*年*ACM*分布式计算原理研讨会论文集，PODC'21，第 155–164 页，2021年。[76] Maofan Yin、Dahlia Malkhi、Michael K. Reiter、Guy Golan Gueta 和 Ittai Abraham。HotStuff：从区块链视角看 BFT 共识。技术报告 1803.05069，arXiv，2019年7月。第 6 版（访问于 2019年5月21日）。[77] Maofan Yin、Dahlia Malkhi、Michael K. Reiter、Guy Golan Gueta 和 Ittai Abraham。HotStuff：兼具线性性与响应性的 BFT 共识。载于*PODC*，2019年。

A ZLB 的补充结果

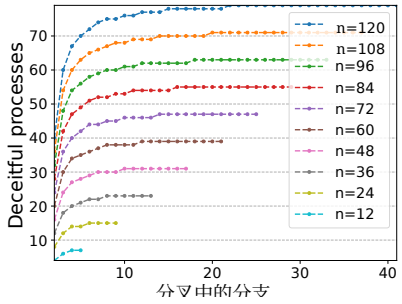
A.1 分支数量与欺诈比例

我们在图1 10中展示了攻击者能够造成至少

a 个分支不一致时所需的最低欺诈比例（左）和欺诈进程数量（右），其中投票阈值为 $h = 2n/3$。值得特别注意的是，不足系统一半的联盟除了双重支付外，无法实施其他攻击；而欺诈比例为 $\delta < 11/18$ 时，最多可以实施六重支付，同时与阈值 $2/3$仅相差 $1/18$ ，非常接近。还请注意， a 的取值范围仅为 1 到 $n/3+1$；当 $t+d=t_s=\lceil 2n/3\rceil-1$ 且$n/3+1$ 个诚实进程分别属于不同分区时，取值为 $n/3+1$。



(a)



(b)

图10：区块链分叉中，要达到投票阈值 $h = 2n/3$，对于分支数 a ，所需的最小欺诈比例 δ （左）和欺骗进程数（右）。

A.2 固定超级区块大小

图11展示了在一个大型广域网中，最多使用300个 AWS c5.4xlarge 实例时 ZLB 的吞吐量，其中所有提案的总大小固定为200，000笔交易。我们展示了实现方案的两种吞吐量结果：一种针对决策，另一种针对确认；对于确认，我们假设采用最严格的 f 可能条件，即在确认某个值之前，必须收到所有回复。我们可以看到，已确认交易的吞吐量略低，因为每个进程

必须等待从每个其他进程接收证书，从而放大了慢速进程的影响。随着进程数量增加，性能会下降，主要原因是证书大小随之增大。我们在图5中省略了确认吞吐量，因为在该设置下，它所需的时间仅比决策多出微不足道的一点，主要是由于瓶颈在于交易验证；对于固定的交易总量，这一点不太明显，如图11所示。

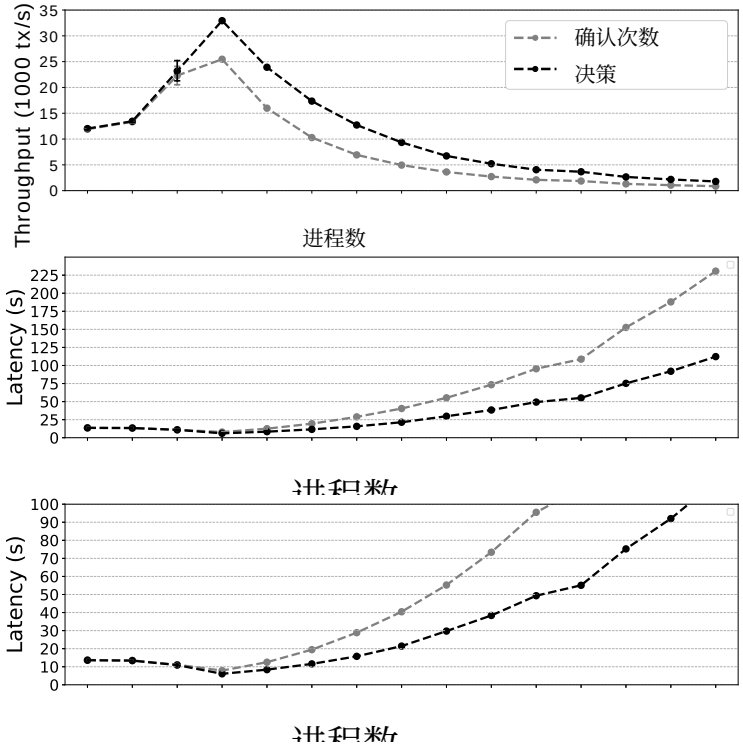


图11：超级区块大小固定为 200，000 笔交易时，ZLB 已决策提案的吞吐量（上）和延迟（中、下），并与已确认提案进行比较。

A.3 二元共识攻击的位掩码

二元共识攻击可以通过引导所有分支采用不同的位掩码，最大化不一致。然而，与诚实进程广播的提案相关联的比特不一致，可能无助于实现攻击者意图发动的特定攻击（例如双重支付）。不过，我们在图12中展示，对于二元共识攻击，即使攻击者在更少的二元共识实例中暴露自身，只要他们不最大化各分支间存在不一致的比特数，不一致的数量就会减少。

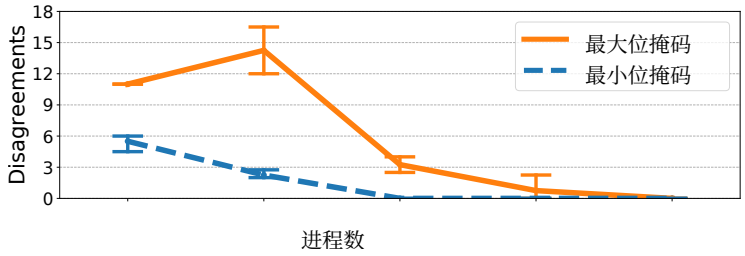


图12：在每个位掩码至少有一个比特、至多所有比特不一致的情况下，二元共识攻击中的不一致决策，适用于 $f = d = \lceil 5n/9 \rceil - 1$。