The Honey Badger of BFT Protocols

ABSTRACT

加密货币令人惊讶的成功引领了对部署大规模，高鲁棒性，关键任务应用程序的拜占庭容错（BFT）协议的兴趣的飙升，例如金融交易。虽然传统的认知是建立在一个（弱）同步协议上，如 PBFT（或其变体），但这种协议严重依赖于网络定时假设，而且只有当网络按照预期运行时才能保证其活性。我们认为这些协议不适合这种部署场景。

我们提出一种替代方案，HoneyBadgerBFT，第一个实用*异步（asynchronous）*BFT 协议，它不需要做出任何定时假设就能保证活性。我们将自己的解决方案基于一种新的可以达到最佳的渐近效率（asymptotic efficiency）的原子广播协议之上。我们展示了一种实现和实验结果，以显示我们的系统可以在一个广域网（wide-area network）上超过一百个节点的规模上达到每秒成千上万次交易的吞吐量。我们甚至在不需要任何调参的情况下，在 Tor 上进行 BFT 实验。与其他替代方案不同的是，HoneyBadgerBFT 根本不关心底层网络。

1. INTRODUCTION

分布式容错协议是用于关键任务基础设施（比如金融交易数据库）的有前景的解决方案。传统上，他们被部署在相对小规模的、而且通常在一个单一的、对抗性攻击可能并不是主要问题的管理领域上。作为一个典型的例子，Chubby[14]，谷歌的容错锁服务的一个部署，由五个节点组成，可以承受多至两个崩溃故障。

近些年来，从比特币惊人的成功开始[43]，一种名为“加密货币（cryptocurrencies）”或“区块链（blockchain）”的分布式系统新的具体化表现出现了。这种加密货币系统展现了令人惊讶且有效的突破[12]，并且在我们对分布式系统的理解上开辟了新的篇章。

加密货币系统挑战了我们对于容错协议部署环境的传统理念。与经典的“5 Chubby nodes within Google”环境不同，加密货币揭示并刺激了一种在互不信任的大量节点之间的广域网上对一致性协议的新需求，此外，网络连接可能比传统局域网设置更加无法预测，甚至是敌对的。这种新设置提出了有趣的新挑战，并且需要我们重新考虑容错协议的设计。

鲁棒性是一等公民（Robustness is a first-class citizen）。加密货币显示了一个不同寻常的、即使是以牺牲性能为代价也要把鲁棒性放在最重要位置上的操作点的需求和可行性。事实上，比特币通过分布式系统标准提供了糟糕的性能：一次交易平均需要 10 分钟才能提交，整个系统只能达到大约每秒 10 次交易的吞吐量。然而，与传统的容错部署场景相比，加密货币在一个高度敌对的，良好的和恶意的攻击并存（如果不普遍）的环境中成长起来。出于这个原因，许多比特币的热情支持者称它为“Honey Badger of Money”[41]。我们注意到，对鲁棒性的需求常常与*非集权化（decentralization）*的需求密切相关，因为非集权化通常需要在广域网络中大量不同参与者的参与。

吞吐量优于延迟（Favor throughput over latency）。大多数现有的可扩展容错协议[6, 49]着重于在单个管理域控制的局域网环境中优化可扩展性。由于带宽配置充足，因此

这些工作通常着重于减少（加密的）计算，并在争夺（即对同一对象请求的争夺）的同时最小化响应时间。

相比之下，区块链已经激起了人们对一类响应时间和竞争不是最关键因素的金融应用的兴趣，例如支付和结算网络[1]。事实上，一些金融应用程序故意在提交交易时引入延迟，以允许可能的回滚或是退款操作。

尽管这些应用程序并不是严格需要延迟，但银行和金融机构已经表示对区块链技术的*高吞吐量（high-throughput）*替代方案感兴趣，从而能够维持大量的请求。例如，Visa 平均处理速率为 2000 tx/sec，峰值为 59000 tx/sec[1]。

1.1. Our Contributions

定时假设是有害的（Timing assumptions considered harmful）。大多数现有拜占庭容错（BFT）系统，甚至是那些被称为“鲁棒的（robust）”系统，都要假设弱同步的一些变形，大致上也就是说，消息要保证在某一边界Δ之后被交付，但Δ可能是实时变化的或者是协议设计者未知的。我们认为，基于定时假设的协议不适用于网络链接不可靠的，网速快速变化的，网络延迟甚至可能被敌对诱发的，非集权化的加密设置。

首先，当违背了预期的定时假设（例如，由于一个恶意网络对手）时，弱同步协议的活性属性可能会完全失败。为了说明这一点，我们明确地构建了一个违反假设的对抗性“间歇同步”网络，使得现有的弱同步协议，比如 PBFT [20]，将慢慢终止（第 3 节）。

然后，即使在实际中，弱同步假设得到满足，弱同步协议也会在底层网络不可预测的情况下显著降低吞吐量。理想情况下，我们想要一个即使在网络快速变化的情况下，其吞吐量也能密切跟踪网络性能的协议。不幸的是，弱异步协议需要非常挑剔难调的超时参数，尤其是在加密货币应用程序设置中；当选中的超时值太长或太短时，吞吐量就会受到阻碍。作为一个具体的例子，我们表明，即使在满足弱同步假设的情况下，这样的协议也很慢从瞬态（transient）网络分区中恢复（第 3 节）。

实用异步 BFT（Practical asynchronous BFT）。我们提出了 HoneyBadgerBFT，第一个在异步设置中提供*最优渐近效率（optimal asymptotic efficiency）*的 BFT *原子广播（atomic broadcast）*协议。因此，我们直接驳倒了这类协议必然不切实际的普遍看法。

我们对已知最好的异步原子广播协议（asynchronous atomic broadcast protocol）做出了极大效率的提高。由 Cachin 等人所提出的[15]，要求每个节点为每个已提交的交易发送 $O(N^2)$比特，从而大幅限制了除了最小网络之外的所有网络的吞吐量。这种低效率有两个根本原因。第一个原因是各方之间的冗余工作。然而，消除冗余的天真尝试损害了公平性，并允许了有针对性的审查攻击。我们发明了一种新颖的解决方案，通过使用阈值公钥加密来防止这些攻击以克服这个问题。第二个原因是对异步公共子集 Asynchronous Common Subset (ACS)子组件的次优实例的使用。我们展示如何通过组合现有但被忽视的技术来有效地实例化 ACS：使用擦除代码的有效可靠广播[18]，以及从多方计算文献中的 ACS 到可靠广播的降低[9]。

HoneyBadgerBFT 的设计针对像网络带宽是稀缺资源但计算量相对较大的部署场景下的加密货币进行的优化。这允许我们利用在主要目标是即使在争用下也能最小化响应时间的经典容错数据库设置中被认为太贵的加密构建区块（特别是阈值公钥加密）的优势。

在一个异步网络里，在不需要做出其他定时假设，消息最终都会被送达。不像已有的非常挑剔且难于调参的弱同步假设协议，HoneyBadgerBFT 对此并不关心。无论网络情况如何波动，HoneyBadgerBFT 的吞吐量总是紧密地跟踪网络可用的带宽。不准确地说，只要消息最终被送达，HoneyBadgerBFT 最终都会取得进展；此外，一旦消息被送达，它就会取得进展。

我们正式证明了 HoneyBadgerBFT 协议的安全性和活性，并且通过实验表明，即使在乐观情况下，它提供比传统 PBFT 协议更好的吞吐量[20]。

实现和大规模实验（Implementation and large-scale experiments）。我们提供了一个 HoneyBadgerBFT 完全成熟的实现，我们将在不久之后将其作为免费的开源软件发布。[1] 我们演示了来自 Amazon AWS 分布在五大洲超过一百个节点的部署的实验结果。为了展示它的多功能性和鲁棒性，我们还在 Tor 匿名中继网络上，不改变任何参数地，部署了 HoneyBadgerBFT，并给出吞吐量和延迟结果。

1.2. Suggested Deployment Scenarios

在众多可能的应用程序中，我们强调了两种可能的部署场景，这些场景是由银行、金融机构和主张完全非集权化加密货币的倡导者所寻求的。

同盟加密货币（Confederation cryptocurrencies）。像比特币这样非集权化加密货币的成功，促使银行和金融机构以一种新的眼光来审视其交易处理和结算的基础设施。“同盟加密货币”是一个经常被引用的愿景[24,25,47]，众多金融机构联合起来共同促成了一项拜占庭一致协议，以允许交易的快速鲁棒的结算。人们对这种方法将简化如今缓慢笨拙的银行间结算的基础设施的热情持续高涨。因此，几个新的开源项目旨在为这个设置构建一个合适的 BFT 协议，比如 IBM 的 Open Blockchain 和 Hyperledger 项目[40]。

同盟加密货币将需要在广域网上部署 BFT 协议，可能涉及数百到数千个一致性节点。在这种设置中，可以容易地控制注册，使得这组一致性节点被认为是一个*先验（priori）*—通常被称为“被许可（permissioned）”的区块链。很明显，HoneyBadgerBFT 是在这种同盟加密货币中使用的天然候选人。

权限较少的区块链的适用性（Applicability to permission less blockchains）。相比之下，像比特币和以太坊这样的非集权化的加密货币选择了一个任何人都可以注册，而且节点可以动态频繁地加入和离开的，“无许可（permissionless）”区块链。为了在这种设置中实现安全性，已知的一致性协议依赖于 proofs-of-work 来战胜 Sybil 攻击，并且在吞吐量和延迟方面付出巨大的代价，比如，比特币每大约 10 分钟提交交易，其吞吐量速率限制在 7 tx/sec，即使当前的区块大小是最大的。最近的几项研究都提出了一个很有前途的想法，就是利用一个更慢的外部区块链，比如比特币或经济“proof-of-stake”的假设，包括基础货币本身[32,32,35,37]，通过在每一个不同的时代，选择一个随机的委员会（committee）执行 BFT 协议，从而引导更快的 BFT 协议。这些方法保证在一个开放注册的、非集权化的网络的安全性，以及与经典的 BFT 协议匹配的吞吐量和响应时间两者之间都达到最佳。因为随机选择的委员会可以是地理上异构的，所以这里的 HoneyBadgerBFT 也是一个自然的选择。

[1] https://github.com/amiller/HoneyBadgerBFT

2. BACKGROUND AND RELATED WORK

我们的总体目标是构建这样一个复制状态机，客户端生成并提交交易，而一个节点网络接收并处理它们。从应用程序的具体细节（比如如何表示状态和计算转换）抽象出来，足以构建一个完全全球统一的、完全有序的、只可附加的交易日志。传统上，这种基元（primitive）称为*总序性（total order）*或*原子广播（atomic broadcast）*[23]；用比特币的说法，我们称之为*区块链（blockchain）*。

容错状态机复制协议提供了强大的安全性和活性保证，允许一个分布式系统在网络延迟和一些节点故障的情况下提供正确的服务。大量的工作研究了这些协议，提供了不同的性能权衡，容忍不同形式的故障和攻击，并对底层网络做出了不同的假设。我们在下面解释了与我们最密切相关的工作。

2.1. Robust BFT Protocols

虽然 Paxos[36]、Raft[45]和许多其他众所周知的协议可以容忍宕机故障，但拜占庭容错协议（BFT），从 PBFT[20]开始，甚至可以容忍任意的（例如，恶意的）损坏的节点。许多后续协议经常通过在没有故障，客户端不会有太多争议，网络运行良好的情况下提供出色性能的*乐观执行（optimistic execution）*来提供改进的性能，不然至少也会有一些进展[2,5,33,39,51]。

通常上，BFT 系统在延迟和 CPU 是瓶颈[49]的部署情况下进行评估，因此最有效的协议可以减少回合次数并最小化昂贵的加密操作。

Clement 等人 [22]通过倡导改善*最坏情况（worst-case）*的性能，发起了最近的一系列工作[4,6,10,21,22,50]，即使系统受到攻击时也能提供服务质量的保证—即使这需要在乐观情况下的性能开销。然而，尽管在这种紧要关头，这种 “Robust BFT”协议能够优雅地容忍受损的节点，但它们仍然依赖于关于底层网络的定时假设。我们的工作进一步采用这种方法，即使在完全异步网络中也能保证良好的吞吐量。

2.2. Randomized Agreement

确定性异步协议对于大多数任务来说是不可能的[27]。虽然绝大多数实用的 BFT 协议都通过制定定时假设避免这一不可能的结果，但随机性（尤其是密码学）提供了另一条途径。事实上，我们知道用于各种任务的异步 BFT 协议，例如二进制协议 binary agreement (ABA)，可靠广播 reliable broadcast (RBC)，等等[13,15,16]。

我们的工作与 SINTRA 最为密切[17]，一个基于 Cachin 等人的异步原子广播协议的系统实现（CKPS01）[15]。该协议包括从原子广播 atomic broadcast (ABC)到公共子集协议 common subset agreement (ACS)的简化，以及从 ACS 到多值验证协议 multi-value validated agreement (MVBA)的简化。

我们贡献的关键发明是一种新颖的从 ABC 到 ACS 的简化，通过批处理提供更好的效率（通过一个 O(N)因子），同时使用阈值加密来保护审查弹性（见第 4.4 节）。我们还通过从文献中提取改进的子组件实例来获取更好的效率。特别地，如第 4.4 节所述，我们通过使用替代 ACS [9]以及有效的 RBC [18]来回避昂贵的 MVBA 基元。

表 1 总结了 HoneyBadgerBFT 与其他几种原子广播协议的渐近性能。这里的“Comm.

Compl.”表示每个被提交的交易的预期通信复杂度（即，传送的总字节）。由于 PBFT 依赖于弱同步假设，因此它在一个异步网络中可能根本无法进行。协议 KS02[34]和 RC05[46]都很乐观，回到了基于 MVBA 的昂贵恢复模式。正如上文提到的 Cachin 等人的协议(CKPS01)[15]可以通过一个更高效的 ACS 架构来改进[9,18]。通过我们新颖的简化，我们还获得了另一个 O(N)改进。

最后，King 和 Saia [30,31]最近通过在稀疏图上路由通信开发了具有小于二次数（less-than-quadratic）的消息的一致协议。然而，将这些结果扩展到异步设置仍然是一个开放的问题。

Table 1: Asymptotic communication complexity (bits per transaction, expected) for atomic broadcast protocols

	Async?	Comm. compl.	
		Optim.	Worst
PBFT	no	$O(N)$	∞
KS02 [34]	yes	$O(N^2)$	$O(N^3)$
RC05 [46]	yes	$O(N)$	$O(N^3)$
CKPS01 [15]	yes	$O(N^3)$	$O(N^3)$
CKPS01 [15]+ [9, 18]	yes	$O(N^2)$	$O(N^2)$
HoneyBadgerBFT (this work)	**yes**	**O(N)**	**O(N)**

3. THE GAP BETWEEN ASYNCHRONOUS AND WEAKLY SYNCHRONOUS NETWORK MODELS

几乎所有现代的 BFT 协议都依赖于定时假设（比如*局部 partial* 或*弱同步 weak synchrony*）来保证活性。近年来，纯异步的 BFT 协议受到了相当少的关注。考虑这样的观点，如果它持有的话，证明以下狭窄的关注点：

[X] *弱的同步假设是不可避免的，因为在任何违反这些假设的网络中，即使异步协议也将提供不可接受的性能。*

在这节中，我们提出反驳上述前提的两个论点。首先，我们说明了异步和弱同步网络模型之间的理论分离。具体地说，我们构建了一个对抗性的网络调度器，它违反了 PBFT 的弱同步假设（确实导致它失败），但在这种情况下，任何纯异步协议（如 HoneyBadgerBFT）都取得了良好的进展。然后，我们做了一个实际的观察：即使他们的假设得到满足，弱同步协议一旦恢复，就会很慢再从网络分区中恢复，而一旦消息被送达，异步协议就会得到进展。

3.1. Many Forms of Timing Assumptions

在继续进行之前，我们将回顾各种标准的定时假设。在异步网络中，对手可以随时以任何顺序传递消息，但尽管如此，*最终（eventually）*必须传递正确节点之间发送的所有消息。异步网络中的节点有效地不使用“实时（real time）”时钟，只能根据接收到的消息的*顺序（ordering）*采取行动。

著名的 FLP[27]结果排除了用于原子广播和许多其他任务的确定性异步协议的可能性。因此，确定性协议必须做出一些更严格的定时假设。一个方便的（但是很强的）网络的假设是*同步（synchrony）*：一个Δ-synchronous 网络保证每条发送的消息在最多推迟

Δ（Δ是实时测量）之后被送达。

较弱的定时假设有几种形式。在 unknown-Δ模型中，协议无法使用延迟界限（delay bound）作为一个参数。或者，在*最终同步（eventually synchronous）*模型中，消息延迟界限 Δ 仅被保证在一些（未知）瞬间保持，称为“全局稳定时间（Global Stabilization Time）”。总的来说，这两种模型被称为*局部同步（partial synchrony）*[26]。然而，另一种变化是*弱同步（weak synchrony）*[26]，这里延迟界限是随时间变化的，但最终并不比时间的多项式函数长得快[20]。

在可行性方面，上述两者是等效的—在一个设置中成功的协议可以系统地适应另一个。然而，在具体的性能方面，对于弱同步的调整意味着随时间而逐渐增加超时参数（例如，通过“指数退避（exponential back-off）”策略）。正如我们稍后所示，这导致从瞬态网络分区恢复时的延迟。

协议通常以超时事件的形式显示这些假设。例如，如果各方发现在某个时间间隔内没有取得任何进展，那么他们就会采取纠正措施，比如选举新领导人。异步协议不依赖于计时器，无论何时消息被送达，也不用管实际的时钟时间，都可以取得进展。

在异步网络中进行计数（Counting rounds in asynchronous networks）。尽管最终交付的保证与“实时”的概念是分离的，但仍然需要描述异步协议的运行时间。标准的方法（例如，像 Canetti 和 Rabin 所解释的[19]）是针对对手将每个消息分配一个虚拟轮次，条件是正确节点之间的每个(r - 1) - message 必须在任何(r + 1) - message 发送前送达。

3.2. When Weak Synchrony Fails

我们现在继续描述当网络条件是对抗性的（或不可预测的）的时候，为什么弱同步 BFT 协议可能会失败（或遭受性能下降）。这解释了为什么这样的协议不适用于第 1 节中描述的面向加密货币的应用场景。

一个阻止 PBFT 的网络调度器（A network scheduler that thwarts PBFT）。我们使用实用的拜占庭容错（PBFT）[20]，一个典型的基于领导者的 BFT 协议，一个有代表性的例子来描述一个对抗性网络调度器如何导致一组基于领导的 BFT 协议[4,6,10,22,33,50]来终止工作。

在任何给定的时间，指定的领导负责提出下一批交易。如果没有取得进展，要么是因为领导者有问题，要么是因为网络停滞，然后节点就会试图选举一个新的领导者。PBFT 协议严重依赖于弱同步网络的活性。我们构建了一个违反这个假设的对抗性调度器，并确实阻止了 PBFT 的任何进展，但对于 HoneyBadgerBFT(实际上，任何异步协议)都可以很好地执行。不令人惊讶的是，基于定时假设的协议在这些假设被违背时失效；然而，演示一个显式攻击有助于激励我们的异步构建。

我们的调度器的直觉很简单。首先，我们假设单个节点已经崩溃。然后，当正确的节点是领导者时，网络会延迟消息，阻止进展，并使循环中的下一个节点成为新的领导者。当崩溃的节点成为下一个领导者时，调度器立即恢复网络分区，并在诚实节点之间非常快速地传递消息；但是，由于领导者崩溃，这里也没有任何进展。

由于 PBFT 在每次失败的领导者选举后延长了超时时间间隔，因此这种攻击违反了弱同步假设，因为它每个巡回必须延迟消息越来越长的时间。另一方面，它也提供了越来越大的同步时间。然而，由于这段时间的同步发生在不方便的时间，PBFT 无法利用

它们。向前看，HoneyBadgerBFT，以及任何一个异步协议，都能在这些机会主义的同步过程中取得进展。

为了证实我们的分析，我们将这个恶意调度程序作为一个代理来实现，它拦截并延迟所有给新的领导者的视图更改消息，并用 1200 行 Python 代码实现的 PBFT 对其进行测试。我们观察到的结果和消息日志与上述分析一致：我们的副本被困在一个从未成功的视图更改请求循环中。在附录 A 中，我们给出了 PBFT 的完整描述并解释了它在这次攻击下的行为。

从网络分区缓慢恢复（Slow recovery from network partitions）。即使弱同步假设最终被满足，依赖于它的协议可能也会很慢地从瞬态网络分区中恢复。考虑下面的场景，只是上述攻击的有限前缀：一个节点崩溃，并且网络在 $2^D\Delta$ 的持续时间内被暂时分区。当崩溃节点成为领导者的回合时，我们的调度器可以精确地恢复网络分区。由于此时的超时间隔是 $2^{d+1}\Delta$，协议必须等待另一个 $2^{d+1}\Delta$间隔之后选出一位新的领导者，尽管网络在此间隔内是同步的。

鲁棒性和响应性之间的权衡（The tradeoff between robustness and responsiveness）。我们上面所观察到的这些行为并不是特定于 PBFT 的，而是从根本上继承于依赖超时来应对崩溃的协议。不管协议如何变化，从业者（practitioner）都必须根据某些权衡来调整他们的超时策略。在一种极端情况（最终同步），从业者做出一个特定的关于网络延迟Δ的估计。如果估计值太低，则系统根本无法取得任何进展；如果太高，它则没有利用可用的带宽。在另一个极端情况（弱同步），从业者避免指定任何绝对延迟，但仍然必须选择一个影响系统跟踪不同条件的速度的“增益（gain）”。异步协议避免了对这些参数进行调优的需要。

4. THE HoneyBadgerBFT PROTOCOL

在本节中，我们将介绍 HoneyBadgerBFT，第一个实现最优渐近效率的异步原子广播协议。

4.1. Problem Definition: Atomic Broadcast

首先我们定义我们的网络模型和原子广播问题。我们的设置包括 N 指定节点的网络，以及独特且著名的特性（P_0 through P_{N-1}）。节点接收交易作为输入，它们的目标是在这些交易的排序上达成共识。我们的模型尤其匹配一个这样的“许可区块链（permissioned blockchain）”的部署场景，交易可以由任意客户端提交，但是负责执行协议的节点是固定的。

原子广播原语允许我们抽象出任何特定于应用程序的细节，例如如何解释交易（以防止重播攻击，例如，应用程序可能定义交易以包括签名和序列号）。就我们的目的而言，交易只简单的是唯一的字符串。在实践中，客户端将生成交易并将它们发送到所有节点，并在收集了大多数节点的签名之后再认为它们是被提交了的。为了简化我们的演示，我们不显式地建模客户端，而是假设交易是由对手选择的，并被提供作为对节点的输入。类似地，交易一旦被一个节点输出，就被认为是提交了的。

我们的系统模型做出以下假设：

- *（纯异步网络 Purely asynchronous network）*我们假设每一对节点都由一个不会丢失消

息的可靠的经过验证的点对点通道连接。[2]交付日程（delivery schedule）完全由对手决定，但是在正确的节点之间发送的每一个消息最终都必须被送达。我们感兴趣的是根据异步轮的数量（如第 2 节所述）来描述协议的运行时间。由于网络可能以任意延迟排队消息，我们还假设节点有无界缓冲区，并且能够处理它们接收到的所有消息。

•*（静态拜占庭故障 Static Byzantine faults）*对手可以对多达 f 个故障节点进行完全控制，其中 f 是一个协议参数。请注意，3f + 1 ≤ N（我们的协议实现）是此设置中广播协议的下限。

•*（信任设置 Trusted setup）*为了便于表达，我们假设节点可以在初始协议特定设置阶段与受信任的经销商进行交互，我们将用它来建立公钥和秘密份额。*注意，在实际部署中，如果实际的可信方不可用，则可以使用分布式密钥生成协议来代替（比较Boldyreva[11]）。*我们所知道的所有分布式密钥生成协议都依赖于定时假设；幸运的是，这些假设只需要在设置过程中进行。

定义 1. *原子广播协议必须满足以下属性，所有这些属性都应该在异步网络中,不管任意对手，都以高概率（作为一个函数 1 − negl(λ)的安全参数，λ）保持：*

•*（一致性 Agreement）*如果任何正确的节点输出一个交易 tx，那么每个正确的节点都输出 tx。

•*（总序性 Total Order）*如果一个正确的节点输出交易序列<$tx_0, tx_1, \ldots tx_j$>，而另一个输出<$tx'_0, tx'_1, \ldots tx'_j$>，则对于 i ≤ min(j, j')，有 $tx_i = tx'_i$。

•*（审查弹性 Censorship Resilience）*如果交易 tx 被输入到 N - f 个正确的节点，则它最终由被个正确的节点输出。

审查弹性属性是一种活性属性，防止对手阻止甚至单一交易的提交。这个属性已被其他名字命名，例如 Cachin 等人[15]提出的“公平性（fairness）”，但是我们更喜欢这个更具描述性的词。

性能标准（Performance metrics）。我们将主要关注分析原子广播协议的*效率（efficiency）*和*交易延迟（transaction delay）*。

•*（效率 efficiency）*假设每个诚实节点的输入缓冲区足够完整 Ω(poly(N, λ))。那么*效率*是每个节点在所有承诺交易中摊销的预期通信成本。

由于每个节点必须输出每个交易，所以 O(1)效率（我们的协议所达到的）是渐近最佳的。上述效率的定义假定网络处于*负载之下（under load）*，这反映了我们的主要目标：在充分利用网络可用带宽的同时保持高吞吐量。由于我们通过批处理实现良好的吞吐量，因此当交易不经常有的时候，我们的系统在低需求期间，每个交易使用更多的带宽。如果我们的目标是最小化成本（例如，基于使用计费），则没有符合这种标准的更强的定义将是合适的。

在实践中，网络链路的容量有限，如果提交的交易比网络可以处理的更多，通常上对于确认时间的保证无法维持。因此我们在下面的问题中定义交易*延迟（transaction delay）*与在此次交易*之前（ahead）*输入的交易数量相关。有限的交易延迟意味着审查弹性。

[2] Reliable channels can be emulated on top of unreliable channels by resending transmissions, at the expense of some efficiency.

•*（交易延迟 Transaction delay）*假设一个对手传递交易 tx 作为输入给 N – f 个正确的节点。让 T 成为“backlog”，即以前输入到任何正确节点的交易总数和已提交的交易数之间的差值。那么*交易延迟*是每个正确的节点作为 T 的函数输出 tx 之前的异步回合的预期数量。

4.2. Overview and Intuition

在 HoneyBadgerBFT 中，节点接收交易作为输入，并将它们存储在它们的（无界）缓冲区中。协议在时期（epochs）中进行，其中在每个时期之后，将新的交易批次添加到提交日志。在每个时期的开始，节点选择其缓冲区中的一个交易的子集（通过我们之后将定义的策略），并将它们作为随机共识协议实例的输入。在共识协议结束时，选择该时期的最终交易集。

在这么高的水平上，我们的方法类似于现有的异步原子广播协议，特别是 Cachin 等人的[15]，大规模交易处理系统（SINTRA）的基础。和我们的类似，Cachin 的协议以异步公共子集（Asynchronous Common Subset, ACS）基元的实例为中心。大致来说，ACS 基元允许每个节点提出一个值，并且保证每个节点输出一个包含至少 N - 2f 个正确节点的输入值的公共向量。从这个基元构建原子广播是微不足道的—每个节点只是从前面的队列中提出一个交易子集，并输出商定的向量中的元素的并集。然而，有两个重要的挑战。

挑战 1：达到审查弹性（Achieving censorship resilience）。ACS 的成本直接取决于每个节点提出的交易集的大小。由于输出向量至少包含 N- f 个这样的集合，因此我们可以通过确保节点提出*大部分不相交（mostly disjoint）*的交易集来提高整体效率，从而以相同的成本在一个批次中提交更多的不同的交易。因此，我们的协议中的每个节点都不需要从其缓冲区中简单地选择第一个元素（如 CKPS01 [15]），而是提出了一个随机选择的样本，这样每个交易平均仅由一个节点提出。

然而，简单地实现，这种优化将损害审查弹性，因为 ACS 基元允许对手选择*哪些（which）*节点的提议最终将被包含在内。对手可以选择性地检查一个交易，不论是哪个（些）节点提出的它。我们通过使用阈值加密来避免这种陷阱，从而阻止对手学习是哪些节点提出哪些交易，直到达成共识之后。完整的协议将在第 4.3 节中描述。

挑战 2：实际吞吐量（Practical throughput）。虽然异步 ACS 和原子广播的理论可行性已经被了解[9,15,17]，但它们的实际性能并不如此。据我们所知，实施 ACS 的唯一其他工作是 Cachin 和 Portiz 所做的[17]，他们表明在广域网上他们可以达到 0.4 tx/sec 的吞吐量。因此，一个有趣的问题是这样的协议是否可以在实践中实现高吞吐量。

在本文中，我们将精心挑选的子组件拼接在一起，我们可以有效地实例化 ACS，并渐近地和实际中都获得更大的吞吐量。值得注意的是，我们将 ACS 的渐近成本（每个节点）从 $O(N^2)$（如 Cachin 等人[15,17]）改进到 O(1)。由于我们选择的组件以前没有一起出现（据我们所知），我们在第 4.4 节中提供了对整个结构的独立描述。

模块化协议组合（Modular protocol composition）。我们现在准备正式提出我们的架构。在这样做之前，我们做一个关于我们的展示风格的备注。我们以模块化的方式定义我们的协议，其中每个协议可以运行其他（子）协议的多个实例。外部协议可以从子协议中提供输入并接收输出。一个节点甚至可以在提供给它输入之前就开始执行（子）协议（比如，如果它接收来自其他节点的消息）。

隔离这种（子）协议实例是非常重要的，以确保与一个实例有关的消息不能在另一个实例中重播。这通过将每个（子）协议实例与唯一字符串（会话标识符）相关联来标识用这个标识符在该（子）协议中发送或接收的任何消息并且相应地路由消息来在实际中实现。我们在协议描述中抑制这些消息标签，以方便阅读。我们使用括号来区分子协议的标记实例。例如，RBC[i]表示 RBC 子协议的第 i 个实例。

我们隐含地假设各方之间是通过认证的异步通道进行异步通信的。实际上，可以使用 TLS socket 来实例化这些通道，例如我们在第 5 节中讨论的。

为了区分协议中各方之间发送的不同消息类型，我们使用**打字机字体（typewriter）**中的标签（例如，**VAL(*m*)**表示类型为 **VAL** 的消息 *m*）。

4.3. Constructing HoneyBadgerBFT from Asynchronous Common Subset

构建区块（Building block）：ACS。我们的主要构建区块是称为异步公共子集（ACS）的基元。建立 ACS 的理论可行性已经在几项工作中得到证实[9,15]。在本节中，我们将介绍 ACS 的正式定义，并将其用作黑盒来构建 HoneyBadgerBFT。稍后在第 4.4 节中，我们将介绍，通过组合过去有些被忽略的几个结构，我们可以有效地实例化 ACS！

更正式地，ACS 协议满足以下属性：

•*（有效性 Validity）*如果一个正确的节点输出一个集合 v，则| v | ≥ N – f 而且 v 包含至少 N - 2f 个正确节点的输入。

•*（一致性 Agreement）*如果一个正确的节点输出 v，则每个节点都输出 v。

•*（整体性 Totality）*如果 N - f 个正确的节点接收到一个输入，则所有正确的节点都会产生一个输出。

构建区块（Building block）：阈值加密（threshold encryption）。*阈值加密方*案 TPKE 是允许任何方将消息加密到主公钥的加密基元，使得网络节点必须一起工作以对其进行解密。一旦 f +1 个正确的节点计算出并显示密文的*解密份额（decryption shares）*，就可以恢复明文（plaintext）；直到至少一个正确的节点显示其解密份额以前，攻击者都不会了解到明文。阈值方案提供以下接口：

• TPKE.Setup(1^λ)→PK, {SK_i}生成公共加密密钥 PK，以及各方的秘密密钥 SK_i

• TPKE.Enc(PK, m)→C 加密消息 m

• TPKE.DecShare(Ski, C)→ σ_i 产生解密的第 i 个份额（或者⊥，如果 C 是格式错误的）

•TPKE.Dec(PK, C, {i, σ_i})→m 从获得明文 m 的至少 f + 1 方组合一组解密份额{i, σ_i}（或者，如果 C 包含无效份额，则无效份额是被标识的）。

在我们的具体实例化中，我们使用了 Baek 和 Zheng 的阈值加密方案[7]。这个方案也是鲁棒的（按照我们的协议要求），这意味着即使对于一个对抗性生成的密文 C，也最多可以恢复一个明文（除了⊥之外）。请注意，我们假设 TPKE.Dec 有效地识别输入中的无效解密份额。最后，该方案明显满足正确性，以及 IND-CPA 游戏的阈值版本。[3]

[3] The Baek and Zheng threshold scheme also satisfies (the threshold equivalent of) the stronger IND-CCA game, but this is not required by our protocol.

ACS 的原子广播（Atomic broadcast from ACS）。我们现在更详细地描述我们的原子广播协议，如图 1 所示。

Algorithm HoneyBadgerBFT (for node $\mathcal{P}_i$)

Let $B = \Omega(\lambda N^2 \log N)$ be the batch size parameter.
Let PK be the public key received from TPKE.Setup (executed by a dealer), and let SK_i be the secret key for $\mathcal{P}_i$.
Let buf := [ ] be a FIFO queue of input transactions.
Proceed in consecutive epochs numbered r:

// Step 1: Random selection and encryption

- let proposed be a random selection of $\lfloor B/N \rfloor$ transactions from the first B elements of buf
- encrypt $x := \mathsf{TPKE.Enc}(\mathsf{PK}, \text{proposed})$

// Step 2: Agreement on ciphertexts

- pass x as input to $\mathsf{ACS}[r]$ *//see Figure 4*
- receive $\{v_j\}_{j \in S}$, where $S \subset [1..N]$, from $\mathsf{ACS}[r]$

// Step 3: Decryption

- for each $j \in S$:
  - let $e_j := \mathsf{TPKE.DecShare}(\mathsf{SK}_i, v_j)$
  - multicast $\mathtt{DEC}(r, j, i, e_j)$
  - wait to receive at least $f + 1$ messages of the form $\mathtt{DEC}(r, j, k, e_{j,k})$
  - decode $y_j := \mathsf{TPKE.Dec}(\mathsf{PK}, \{(k, e_{j,k})\})$
- let $\mathsf{block}_r := \mathsf{sorted}(\cup_{j \in S}\{y_j\})$, such that block_r is sorted in a canonical order (e.g., lexicographically)
- set $\mathsf{buf} := \mathsf{buf} - \mathsf{block}_r$

Figure 1: HoneyBadgerBFT.

如上所述，该协议集中围绕 ACS 的实例。为了获得可伸缩的效率，我们选择了批处理策略。我们让 B 作批次大小，并在每个时期提交 Ω(B)个交易。每个节点从其队列中提出 B/N 交易。为了确保节点提出大部分不同的交易，我们从每个队列中的第一个 B 中随机选择这些交易。

如第 4.4 节所述，我们的 ACS 实例化具有 $O(N^2 |v| + \lambda N^3 \log N)$的总通信成本，其中 $|v|$ 限制任何节点的输入的大小。因此，我们选择大小为 $B = \Omega(\lambda N^2 \log N)$的批次，使得每个节点(B/N)的贡献可以吸收这种附加开销。

为了防止对手影响结果，我们使用阈值加密方案，如下所述。简而言之，每个节点选择一组交易，然后对其进行加密。然后，每个节点将加密作为输入传递给 ACS 子程序。因此，ACS 的输出是密文的向量。一旦 ACS 完成，密文就将被解密。这样可以保证在交易对象了解每个节点提出的提议的特定内容之前完全确定一组交易。这保证对手无法有选择地阻止交易一旦在足够的正确节点处位于队列前方就被提交。

4.4. Instantiating ACS Efficiently

Cachin 等人提出一个我们称之为 CKPS01 的协议（隐含地）将 ACS 简化为到多值验证拜占庭协议（MVBA）[15]。粗略来说，MVBA 允许节点提出满足最终被选择的述语（predicate）之一。简化方法很简单：验证述语表示输出必须是至少 N - f 方签名输入的

向量。不幸的是，MVBA 基元协议成为了一个瓶颈，因为我们知道的唯一的构造引起了 $O(N^3|v|)$的开销。

我们通过使用完全回避 MVBA 的 ACS 的一个替代实例来避免这种瓶颈。我们使用的实例是由 Ben-Or 等人提出的[9]，而且在我们看来，在某种程度上有些被忽略了。实际上，它的产生早于 CKPS01 [15]，最初是为了一个大不相关的目的而开发的（作为实现有效的异步多方计算的工具[9]）。该协议是从 ACS 到*可靠广播 reliable broadcast* (RBC)和*异步二进制拜占庭协议 asynchronous binary Byzantine agreement* (ABA)的简化。只是最近我们才知道这些子组件的有效构造，我们马上就会解释。

在高层次上，ACS 协议分两个阶段进行。在第一阶段，每个节点 P_i 使用 RBC 将其提出的值传播给其他节点，随后，ABA 来决定一个指示哪些 RBC 已经成功完成的位向量。

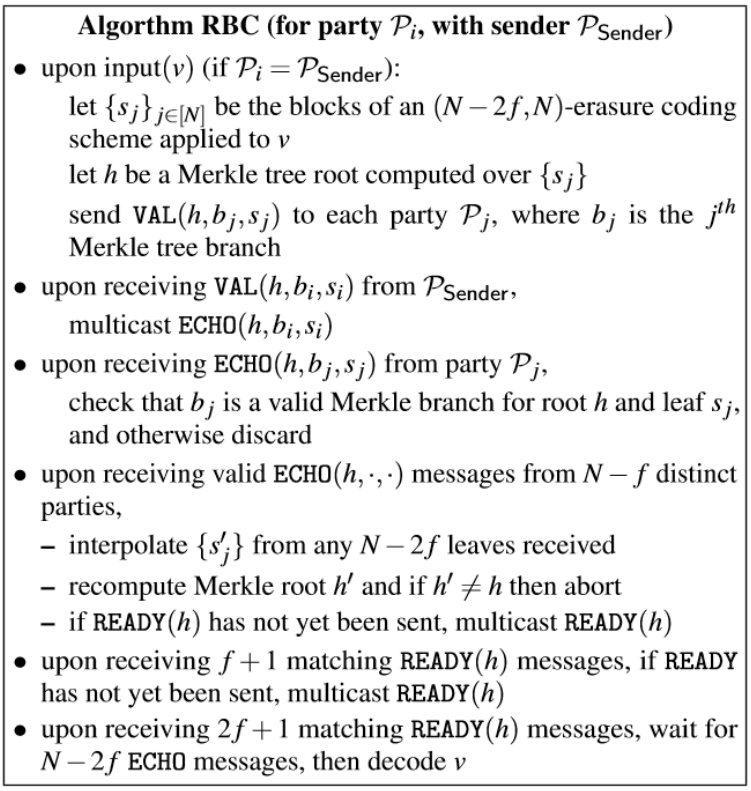
现在，我们在更详细地解释 Ben-Or 协议之前，先简要解释 RBC 和 ABA 结构。

最佳通信可靠广播（Communication-optimal reliable roadcast）。异步可靠广播频道满足以下属性：

- *（一致性 Agreement）*如果任何两个正确的节点传递 v 和 v’，则 v = v’。
- *（整体性 Totality）*如果任何正确的节点传递 v，则所有正确的节点传递 v。
- *（有效性 Validity）*如果发送者是正确的并且输入 v，则所有正确的节点传递 v。

虽然 Bracha 的[13]经典可靠广播协议需要 $O(N^2|v|)$位的总通信量，以便广播大小为 $|v|$的消息，Cachin 和 Tessaro [18]认为，擦除编码可以即便在最坏的情况下也可以将此成本降低到仅仅 $O(N|v| + \lambda N^2 \log N)$。这个对于大量消息（即，当$|v| >> \lambda N \log N$ 时）的重要改进措施，可以指导我们选择批量大小（回看第 4.3 节）。这里的擦除编码的使用最多引起一个等于$\frac{N}{N-2f} < 3$的小常数因子的开销。

如果发送者是正确的，总运行时间是三个（异步）轮次；在任何情况下，在第一个正确的节点输出一个值，和最后一个节点输出一个值之间，最多消耗两轮。可靠广播算法如图 2 所示。



Algorthm RBC (for party $\mathcal{P}_i$, with sender $\mathcal{P}_{\text{Sender}}$)

- upon input(v) (if $\mathcal{P}_i = \mathcal{P}_{\text{Sender}}$):
  let $\{s_j\}_{j\in[N]}$ be the blocks of an $(N-2f,N)$-erasure coding scheme applied to v
  let h be a Merkle tree root computed over $\{s_j\}$
  send $\texttt{VAL}(h,b_j,s_j)$ to each party $\mathcal{P}_j$, where b_j is the j^{th} Merkle tree branch
- upon receiving $\texttt{VAL}(h,b_i,s_i)$ from $\mathcal{P}_{\text{Sender}}$,
  multicast $\texttt{ECHO}(h,b_i,s_i)$
- upon receiving $\texttt{ECHO}(h,b_j,s_j)$ from party $\mathcal{P}_j$,
  check that b_j is a valid Merkle branch for root h and leaf s_j, and otherwise discard
- upon receiving valid $\texttt{ECHO}(h,\cdot,\cdot)$ messages from $N-f$ distinct parties,
  - interpolate $\{s'_j\}$ from any $N-2f$ leaves received
  - recompute Merkle root h' and if $h' \neq h$ then abort
  - if $\texttt{READY}(h)$ has not yet been sent, multicast $\texttt{READY}(h)$
- upon receiving $f+1$ matching $\texttt{READY}(h)$ messages, if READY has not yet been sent, multicast $\texttt{READY}(h)$
- upon receiving $2f+1$ matching $\texttt{READY}(h)$ messages, wait for $N-2f$ ECHO messages, then decode v

Figure 2: Reliable broadcast algorithm, adapted from Bracha's broadcast [13], with erasure codes to improve efficiency [18]

二进制协议（Binary Agreement）。二进制协议是允许节点在单个位的值上达成一致的标准基元。更正式地说，二进制协议保证三个属性：

•*（一致性 Agreement）*如果任何正确的节点输出比特 b，则每个正确的节点输出 b。

•*（终止 Termination）*如果所有正确的节点接收到输入，则每个正确的节点输出一个比特。

•*（有效性 Validity）*如果任何正确的节点输出 b，则至少一个正确的节点接收 b 作为输入。

有效性意味着*一致（unanimity）*：如果所有正确的节点接收到相同的输入值 b，则 b 必须是确定的值。另一方面，如果两个节点在任何点接收不同的输入，那么对手可能会在其余节点接收输入之前对每个值强制执行该决定。

我们用 Moustefaoui 等人的协议[42]实例化了这个基元，它是基于加密 common coin 的。我们将此实例的解释推迟到附录。其预期运行时间为 O(1)，实际上在概率为 $1\text{-}2^{-k}$ 的时候在 O(k)回合内完成。每个节点的通信复杂度是 $O(N\lambda)$，这主要是由于 common coin 中使用的阈值加密。

提议值的一个子集的一致性（Agreeing on a subset of proposed values）。将上述部分放在一起，我们使用 Ben-Or 等人的协议 [9]就包含至少 N - f 个节点的整个提议的一组值达成一致。

在高层次上，这个协议分两个阶段进行。在第一阶段，每个节点 P_i 使用可靠广播向

其他节点传播其提出的值。在第二阶段，使用 N 个二进制拜占庭协议的并发实例对位向量$\{b_j\}_{j\in[1..N]}$达成一致，其中 $b_j = 1$ 表示 P_j 的提议值包含在最终的集合中。

实际上，上面的简单描述掩盖了一个微妙的挑战，Ben-Or 提供了一个聪明的解决方案。

对上述梗概实现的简单尝试将使每个节点等待第一个（N – f 个）广播来完成，然后针对与其相对应的二进制协议实例提出 1，对于所有其他广播则提出 0。然而，正确的节点可能会以不同的顺序观察完成的广播。由于二进制协议仅保证输出为 1，如果所有正确的节点一致提出 1，则可能产生的位向量会是空的。

为了避免这个问题，节点放弃了提议 0，直到确定最终向量将至少具有 N - f 位。为了使得这个协议的流程更加直观，我们在图 3 中叙述了几个可能的情况。Ben-Or 等人提出的算法[9]在图 4 中给出。预期运行时间为 O(logN)，因为它必须要等待所有的二进制协议实例结束。[4]当用上述可靠广播和二进制协议结构来实例化时，总的通信复杂度为 $O(N^2 |v| + \lambda N^3 \log N)$，假设$|v|$是任何节点输入的最大尺寸。

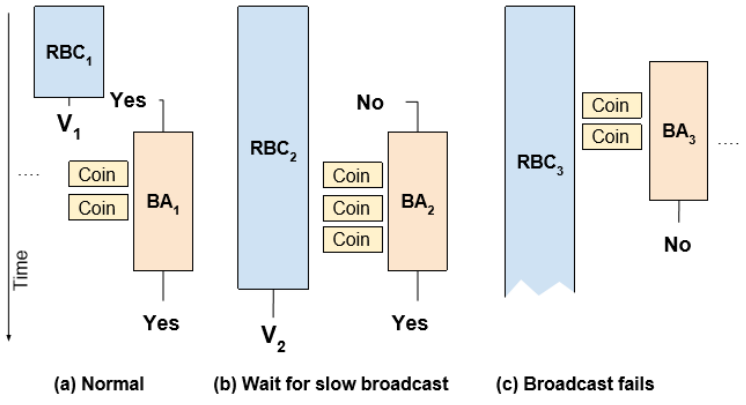


Figure 3: (Illustrated examples of ACS executions.) Each execution of our protocol involves running N concurrent instances of reliable broadcast (RBC), as well as N of byzantine agreement (BA), which in turn use an expected constant number of common coins. We illustrate several possible examples of how these instances play out, from the viewpoint of Node 0. (a) In the ordinary case, Node 0 receives value V_1 (Node 1's proposed value) from the reliable broadcast at index 1. Node 0 therefore provides input "Yes" to BA_1, which outputs "Yes." (b) RBC_2 takes too long to complete, and Node 0 has already received $(N-f)$ "Yes" outputs, so it votes "No" for BA_2. However, other nodes have seen RBC_2 complete successfully, so BA_2 results in "Yes" and Node 0 must wait for V_2. (c) BA_3 concludes with "No" before RBC_3 completes.

[4] The expected running time can be reduced to O(1) (c.f.[8]) by running several instances in parallel, though this comes at the expense of throughput.

Algorithm ACS (for party $\mathcal{P}_i$)

Let $\{\mathsf{RBC}_i\}_N$ refer to N instances of the reliable broadcast protocol, where $\mathcal{P}_i$ is the sender of RBC_i. Let $\{\mathsf{BA}_i\}_N$ refer to N instances of the binary byzantine agreement protocol.

- upon receiving input v_i, input v_i to RBC_i *// See Figure 2*
- upon delivery of v_j from RBC_j, if input has not yet been provided to BA_j, then provide input 1 to BA_j. *See Figure 11*
- upon delivery of value 1 from at least $N - f$ instances of BA, provide input 0 to each instance of BA that has not yet been provided input.
- once all instances of BA have completed, let $C \subset [1..N]$ be the indexes of each BA that delivered 1. Wait for the output v_j for each RBC_j such that $j \in C$. Finally output $\cup_{j \in C} v_j$.

Figure 4: Common Subset Agreement protocol (from Ben-Or et al. [9])

4.5. Analysis

首先，我们在 ACS 的定义和 TPKE 方案的鲁棒性之后紧接着声明一致性和总序性。

理论 1.（一致性和总序性 Agreement and total order）。*HoneyBadgerBFT 协议满足一致性和总序性，除了可忽略不计的可能性之外。*

证明. 这两个属性紧随高水平协议的属性，ACS 和 TPKE。每个 ACS 实例保证节点在每个时期对一个密文向量达成一致（步骤 2）。TPKE 的鲁棒性保证每个正确的节点将这些密文解密为一致的值（步骤 3）。这足以确保一致性和总序性。

理论 2.（复杂度 Complexity）。*假设一个大小为 $B = \Omega(\lambda N^2 \log N)$ 的批次，则预期每个 HoneyBadgerBFT 时期的运行时间为 $O(\log N)$，并且总的预期通信复杂度为 $O(B)$。*

证明. ACS 的成本和运行时间在第 4.4 节中进行了说明。阈值解密的 N 个实例产生一个额外的轮次和额外的 $O(\lambda N^2)$成本，这不会影响整体渐近成本。

HoneyBadgerBFT 协议可以在单个时期提交最多 B 个交易。然而，实际数量可能小于这个值，因为一些正确的节点可能会提出重叠的交易集，另一些可能会响应太晚，而损坏的节点可能会提出一个空集合。幸运的是，我们证明（在附录中）假设每个正确的节点的队列已满，则 B/4 作为在一个时期中提交的交易的预期数量的下限。[5]

理论 3.（效率 Efficiency）。*假设每个正确的节点的队列至少包含 B 个不同的交易，那么在一个时期中提交的交易的预期数量至少是 $B/4$，从而导致持续的效率。*

最后，我们证明（在附录中）对手不能明显地延迟任何交易的提交。

理论 4.（审查弹性 Censorship Resilience）。假设对手将交易 tx 作为输入传递给 N - f 个正确的节点。令 T 为“backlog”的大小，即先前输入到任何正确节点的交易的总数与已经提交的交易数之间的差。那么 tx 在 $O(T/B + \lambda)$个时期内被提交，除了可忽略不计的可能性之外。

[5] The actual bound is $(1-e^{-1/3})B > B/4$, but we use the looser bound B/4 for readability.

5. IMPLEMENTATION AND EVALUATION

在本节中，我们使用 HoneyBadgerBFT 协议的原型实现进行了几项实验和性能测量。除非另有说明，本节中报告的数字是默认所有节点行为都诚实的*乐观（optimistic）*情况。

首先，我们通过在包含在五大洲的多达 104 个节点的广域网中进行实验，证明了 HoneyBadgerBFT 的确是可扩展的。即使在这些条件下，HoneyBadgerBFT 峰值可以达到每秒数千个交易的吞吐量。此外，通过与 PBFT 的比较，一个有代表性的局部同步协议，HoneyBadgerBFT 仅在执行小常数因子时表现更差。最后，我们演示了通过在 Tor 匿名通信层上运行异步 BFT 的可行性。

实现细节（Implementation details）。我们在 Python 中使用 gevent 库进行并发任务，开发了 HoneyBadgerBFT 的原型实现。

对于确定性擦除编码，我们使用了实现 Reed-Solomon 代码的 zfec 库[52]。对于 common coin 基元的实例化，我们实现了 Boldyreva 的基于配对的阈值签名方案[11]。对于交易的阈值加密，我们使用 Baek 和 Zheng 的方案[7]加密一个 256 位的临时密钥，其次是在实际有效载荷的 CBC 模式下的 AES-256。我们使用 PBC 库[38]的 Charm [3] Python 包装器来实现这些阈值加密方案。对于阈值签名，我们使用提供的 MNT224 曲线，产生只有 65 个字节的签名（和签名份额），并启发式提供 112 位的安全性。[6]我们的阈值加密方案需要一个对称双线性组：因此我们使用 SS512 组，它启发式提供了 80 位的安全性[44]。[7]

在我们的 EC2 实验中，我们使用普通的（未认证的）TCP sockets。在真正的部署中，我们将使用带有客户端和服务器身份验证的 TLS，为长时间的会话添加微小的开销。类似地，在我们的 Tor 实验中，每个 socket 只有一个端点被认证（通过“隐藏服务”地址）。

我们的理论模型假设节点有无界缓冲区。实际上，尽管我们的原型实现还没有包括这个特征，但是当存储器消耗达到标记线时（例如，每当 75％满时），可以动态地向节点添加更多的资源。未能提供足够的缓冲区可以反映故障预算 f。

5.1. Bandwidth Breakdown and Evaluation

我们首先分析我们系统的带宽成本。在所有实验中，我们假定大小为每个 $m_T = 250$ 字节承认 ECDSA 签名的常量交易，两个公钥以及一个应用有效负载（即，大致上的典型比特币交易的大小）。我们的实验使用参数 N = 4f，[8]各方提出了一批 B/N 个交易。为了对最坏情况进行建模，节点以大小为 B 的相同队列开始。我们将运行时间记录为从实验开始到第（N - f）个节点输出值的时间。

带宽和分解结果（Bandwidth and breakdown findings）。每个节点消耗的总带宽由一个固定的附加开销和依赖于交易的开销组成。对于我们考虑的所有参数值，附加开销由一个由在它之前的 ABA 阶段和解密阶段的阈值加密产生的 $O(\lambda N^2)$项主导。ABA 阶段使每个节点期望发送 $4N^2$ 的签名份额。只有 RBC 阶段才会导致与擦除编码扩展因子

[6] Earlier reports estimate 112 bits of security for the MNT224 curve [44]; however, recent improvements in computing discrete log suggest larger parameters are required [28,29].

[7] We justify the relatively weak 80-bit security level for our parameters because the secrecy needs are short-lived as the plaintexts are revealed after each batch is committed. To defend against precomputation attacks, the public parameters and keys should be periodically regenerated.

[8] The setting N = 4f is not the maximum fault tolerance, but it is convenient when f divides N.

$r = \frac{N}{N-2f}$相等的依赖交易的开销。由于 ECHO 消息中包含 Merkle 树分支，RBC 阶段也提供 $N^2 \log N$ 哈希开销。（每个节点的）通信成本估计为：

$$m_{\text{all}} = r(Bm_{\text{T}} + Nm_{\text{E}}) + N^2((1 + \log N)m_{\text{H}} + m_{\text{D}} + 4m_{\text{S}})$$

其中 m_E 和 m_D 分别是 TPKE 方案中的密文和解密份额，m_S 是一个 TSIG 签名份额的大小。

随着我们增加建议的批量大小 B，系统的有效吞吐量增加，使得成本的依赖交易部分占主导地位。如图 5 所示，对于 N = 128，最多 1024 个交易的批量大小，与非依赖交易的带宽仍然在总开销中占主导地位。然而，当批量大小达到 16384 时，依赖交易部分开始占主导地位—这主要是因为节点发送擦除编码区块的 RBC.ECHO 阶段。

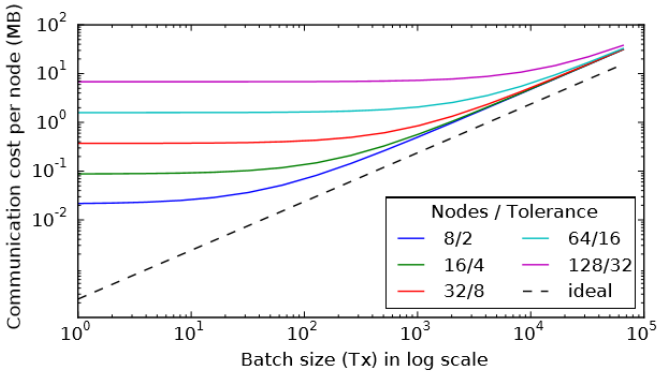


Figure 5: Estimated communication cost in megabytes (per node) for varying batch sizes. For small batch sizes, the fixed cost grows with $O(N^2 \log N)$. At saturation, the overhead factor approaches $\frac{N}{N-2f} < 3$.

5.2. Experiments on Amazon EC2

为了看看我们的设计如何实用，我们在 Amazon EC2 服务上部署了我们的协议，并对其性能进行了全面的测试。我们在 32，40，48，56，64 和 104 个 Amazon EC2 t2.medium 均匀分布在整个五大洲的 8 个地区上的实例上运行 HoneyBagderBFT。在我们的实验中，我们改变了批量大小，使得每个节点提出 256，512，1024，2048，4096，8192，16384，32768，65536 或 131072 个交易。

吞吐量（Throughput）。吞吐量定义为每单位时间提交的交易数量。在我们的实验中，如果没有指定，我们使用“每秒确认交易（confirmed transactions per second）”作为衡量单位。图 6 显示了所有 N 方提议的吞吐量与交易总数之间的关系。容错参数设置为 f = N / 4。

发现. 从图 6 可以看出，对于每个设置，吞吐量随着提议的交易数量的增加而增加。对于多达 40 个节点的中型网络，我们可以实现每秒超过 20000 个交易的吞吐量。对于一个大型 104 节点的网络，我们可以维持每秒超过 1500 个交易。给定一个有限的批量大小，所有的网络大小将最终收敛到一个共同的只受可用带宽限制的上限。虽然网络中消耗的总带宽随着每个附加节点的增加而（线性）增加，但额外的节点也可以提供额外的带宽容量。

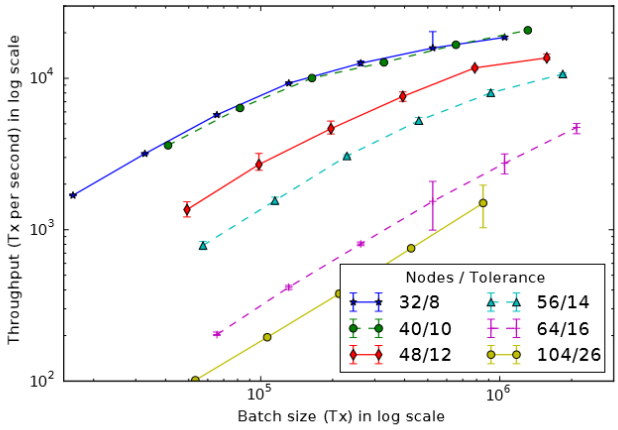


Figure 6: Throughput (transactions committed per second) vs number of transactions proposed. Error bars indicate 95% confidence intervals.

吞吐量，延迟和规模权衡（Throughput, latency, and scale tradeoffs）。延迟定义为第一个节点接收客户端请求的时间与第（N - f）个节点结束一致性协议之间的时间间隔。这是合理的，因为完成协议的第（N - f）个节点意味着实现诚实方的一致性。

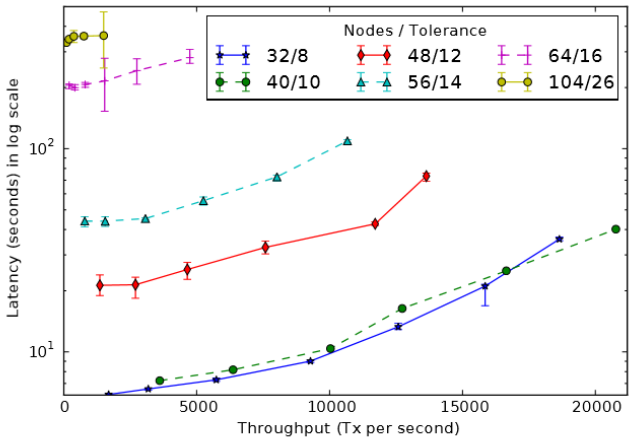


Figure 7: Latency vs. throughput for experiments over wide area networks. Error bars indicate 95% confidence intervals.

图 7 显示了 N 和 f = N / 4 的不同选择的延迟和吞吐量之间的关系。正斜率表明我们的实验尚未使可用带宽完全饱和，即使在更大的批次大小的情况下，我们也可以获得更好的吞吐量。图 7 还显示，随着节点数量的增加，延迟也会增加，这很大程度上源于协议的 ABA 阶段。实际上，在 N = 104 时，对于我们尝试的批量大小的范围，我们的系统是 CPU 限制而不是带宽限制的，因为我们的实现是单线程的，而且必须验证 $O(N^2)$ 阈值签名。即使这样，我们最大的 104 个节点的实验是在 6 分钟内完成的。

尽管可以在不影响最大可达到的吞吐量的情况下将更多的节点（具有相同的带宽配置）添加到网络中，但是提交一个批次所消耗的最小的带宽（和因此而产生的延迟）随 $O(N^2 \log N)$增加。这种约束意味着可扩展性的限制，这取决于带宽成本和用户对延迟的容忍。

与 PBFT 的比较（Comparison with PBFT）。图 8 显示了与 PBFT 协议（用于局部

同步网络的经典 BFT 协议）的比较。我们使用 Croman 等人的 Python 实现 [24]，运行在均匀分布在亚马逊 AWS 区域间的 8，16，32 和 64 个节点上。选择批量大小以使网络的可用带宽饱和。

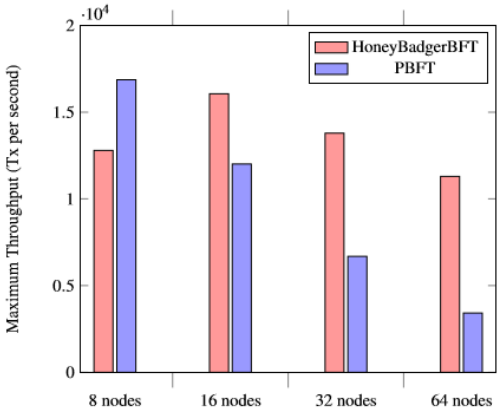


Figure 8: Comparison with PBFT on EC2s

根本地，虽然 PBFT 和我们的协议具有*总共（in total）*相同的渐近通信复杂度，但是我们的协议在网络链路之间均匀地分配了这个负载，而 PBFT 在领导者的可用带宽上陷入瓶颈。因此，PBFT 的可实现吞吐量随着节点数量的减少而减少，而 HoneyBadgerBFT 大致保持恒定。

请注意，本实验仅反映无故障或网络中断的乐观情况。即使对于小型网络，HoneyBadgerBFT 在对抗条件下提供了明显更好的鲁棒性，如第 3 节所述。特别地，PBFT 在对抗异步调度器下出现**零吞吐量（zero throughput）**，而 HoneyBadgerBFT 将以正常速率完成这一时期。

5.3. Experiments over Tor

为了演示 HoneyBadgerBFT 的鲁棒性，我们运行了第一个在 Tor（最成功的匿名通信网络）上实施的容错一致性协议的（我们所知的）。与我们原始的 AWS 部署相比，Tor 增加了显着的和变化的延迟。无论如何，我们证明了我们可以运行 HoneyBadgerBFT 而不调整任何参数。隐藏在 Tor 保护罩后面的 HoneyBadgerBFT 节点可能会提供甚至更好的鲁棒性。由于它有助于节点隐藏其 IP 地址，因此可以帮助他们避免有针对性的网络攻击和涉及其物理位置的攻击。

Tor 的简要背景介绍（Brief background on Tor）。Tor 网络由大约 6500 个列在公共目录服务中的继电器组成。Tor 启用“隐藏服务”，即通过 Tor 接受连接以隐藏其位置的服务器。当客户端建立与隐藏服务的连接时，客户端和服务器都将 3-hop circuits 构建为一个公共的“会合点（rendezvous point）”。因此，与隐藏服务的每个连接都会通过 5 个随机选择的继电器来路由数据。Tor 为继电器节点提供了一种方式来宣传其容量和利用率，这些自我报告的度量由 Tor 项目进行汇总。根据这些指标，[9]网络的总容量为～145Gbps，当前利用率为～65Gbps。

Tor 实验设置（Tor experiment setup）。我们设计实验设置，以便我们可以在运行

[9] https://metrics.torproject.org/bandwidth.html as of Nov 10, 2015

Tor 守护进程软件的单台计算机上运行所有 N 个 HoneyBadgerBFT 节点，同时能够真实地反映 Tor 继电器路径。为此，我们配置我们的机器来监听 N 个隐藏服务（在我们的模拟网络中为每个 HoneyBadgerBFT 节点提供一个隐藏服务）。由于每个 HoneyBadgerBFT 节点形成与其他节点的连接，我们构建了每个实验的总共 N^2 个 Tor 电路，从我们的机器开始并结束，通过 5 个随机继电器。总而言之，所有成对的覆盖链接遍历由*随机（random）*继电器节点组成的真正的 Tor 电路，设计使得获得的性能可以代表 Tor 上真正的 HoneyBadgerBFT 部署（尽管所有模拟节点都在单个主机上运行）。

由于 Tor 为许多用户提供关键的公共服务，因此确保在实时网络上进行的研究实验不会对其产生不利影响是非常重要的。我们只从一个优势点形成连接（从而避免接收），并且以小参数（在我们最大的实验中仅形成 256 个电路）运行短时间（几分钟）的实验。总的来说，我们的实验涉及到通过 Tor 传输大约五千兆字节（gigabytes）的数据—小于每日利用率的 1E-5 分数。

图 9 显示了延迟如何随吞吐量而变化。与我们的节点具有足够带宽的 EC2 实验相比，Tor 电路由电路中最慢的链路限制。在 Tor 中，我们可以达到每秒最多超过 800 次交易的吞吐量。

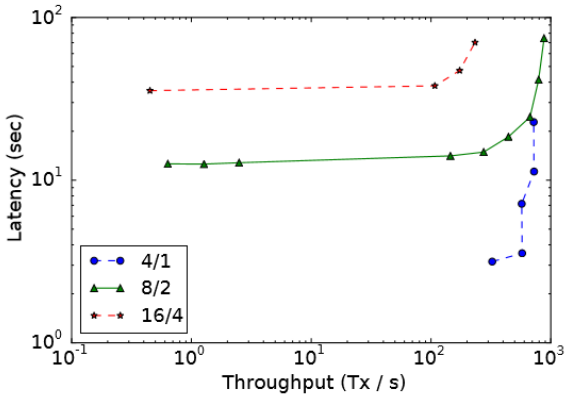


Figure 9: Latency vs throughput for experiments running HoneyBadgerBFT over Tor.

一般来说，传递在 Tor 的中继网络上的消息往往具有显著且高度变化的延迟。例如，在我们在 8 方实验中，每一方提出 16384 个交易，一个信息可以延迟 316.18 秒，延迟差超过 2208，而平均延迟只有 12 秒。我们强调，我们的协议不需要对这样的网络条件进行调整，传统的最终同步协议也是如此。

6. CONCLUSION

我们已经介绍了 HoneyBadgerBFT，第一个有效且具有高吞吐量的异步 BFT 协议。通过我们的实现和实验结果，我们展示了 HoneyBadgerBFT 可以成为容错交易处理系统初始激励加密货币部署中的一个合适的组件。更一般来说，我们相信我们的工作展示了构建基于异步协议的可靠交易处理系统的承诺。