蜜獾拜占庭协议

Elaine Shi[[1]](#footnote-2) Yu Xia[[2]](#footnote-3) Dawn Song[[3]](#footnote-4) Kyle Croman[[4]](#footnote-5) Andrew Miller[[5]](#footnote-6)

1 介绍

分布式容错协议是关键任务基础设施（如金融交易数据库）的有前途的解决方案。传统上，它们的规模相对较小，通常在一个单一的行政领域内进行，这种领域的普通攻击可能不是主要关注的问题。作为一个典型的例子，Google的容错锁定服务Chubby [14]由五个节点组成，最多可容忍两个崩溃故障。

近年来，分布式系统“加密货币”或“区块链”的一个新的体现出现，比特币的惊人成功开始[44]。这种加密货币系统代表了一个惊人而有效的突破[12]，为我们理解分布式系统打开了新的篇章。

加密货币系统挑战了我们对于容错协议部署环境的传统信念。与经典的“Google内部5个胖乎乎的节点”环境不同的是，加密算法已经揭示并刺激了对广域网上协商一致协议的新需求，在大量互不信任的节点之间，而且网络连接可能很多比传统的局域网设置更难以预测，甚至是敌对的。这种新的设置带来了有趣的新挑战，并呼吁我们重新思考容错协议的设计。

鲁棒性是头等公民。加密货币证明了一个不寻常的操作点的需求和可行性，即将性能优先于鲁棒性。实际上，比特币通过分布式系统标准提供了可怕的性能：交易平均需要10分钟才能完成，整个系统的吞吐量达到每秒10笔交易。然而，与传统的容错部署方案相比，加密货币在高度敌对的环境中茁壮成长，预计出现动机不足和恶意攻击（如果不是普遍的话）。出于这个原因，比特币的许多热心支持者称之为“金钱獾”[41]。我们注意到对稳健性的需求往往与对权力下放的需求密切相关 - 因为权力下放通常需要大量不同参与者参与广域网。

在延迟上的吞吐量。大多数现有的关于可扩展容错协议[6,49]的着作都着重于在由单个管理域控制的LAN环境中优化可伸缩性。由于带宽提供是充足的，所以这些工作通常集中于减少（密码）计算并且在争用期间使响应时间最小化（即请求竞争相同的对象）。

与此相反，区块链则引起了对一类金融应用的兴趣，其中响应时间和争用并不是最关键的因素，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事务中引入延迟，以允许可能的回滚/退款操作。

虽然这些应用不是延迟时间的关键，但是银行和金融机构已经表现出对区块链技术的高吞吐量替代方案的兴趣，能够维持大量的请求。例如，Visa平均处理速度为2,000 tx / sec，最高峰值为59,000 tx / sec [1]。

1.1我们的贡献

时间假设被认为是有害的。大多数现有的拜占庭容错（BFT）系统，即使是那些被称为“鲁棒”的系统，也假设了一些弱同步的变化，大致来说，信息保证在一定的边界Δ之后被传递，Δ可能是时变的或协议设计者未知的。我们认为，基于时间假设的协议不适用于网络链接可能存在的分散式加密货币设置不可靠，网络速度变化迅速，网络延迟甚至可能被敌手诱导。

首先，当预期的时间假设被违反时（例如，由于恶意的网络攻击者），弱同步协议的活性特性可能完全失效。为了证明这一点，我们明确地建立了一个违反假设的敌对“间歇性同步”网络，使得现有的弱同步协议如PBFT [20]会停下来（第3节）。

其次，即使在实践中满足弱同步假设时，当基础网络不可预测时，弱同步协议的吞吐量也会显着下降。理想情况下，我们希望即使在迅速变化的网络条件下，吞吐量也能紧密跟踪网络的性能。不幸的是，弱异步协议需要超时参数，特别是在加密货币应用程序设置中，当选择的超时值太长或太短时，吞吐量可能受到阻碍。作为一个具体的例子，我们表明即使当弱同步假设得到满足时，这样的协议也很难从瞬态网络分区恢复（第3节）。

实用的异步BFT。我们提出了HoneyBadgerBFT，第一个BFT原子广播协议在异步设置中提供最佳的渐近效率。因此，我们直接驳斥普遍的看法，即这些协议必然是不切实际的。

由于Cachin等人，我们对最好的已知异步原子广播协议进行了显着的效率改进。 [15]，这要求每个节点为每个提交的事务传输O（N2）个比特，实质上限制了除了最小网络之外的所有其他吞吐量。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，一个天真的企图消除冗余妥协的公平财产，并允许有针对性的审查攻击。我们发明了一种新的解决方案来解决这个问题，使用阈值公钥加密来防止这些攻击。第二个原因是使用异步公共子集（ACS）子组件的次优实例。我们展示了如何通过结合现有但被忽视的技术来高效地实例化ACS：使用纠删码的有效可靠的广播[18]，以及从多方计算文献[9]中减少ACS到可靠广播。

HoneyBadgerBFT的设计针对加密货币进行了优化，网络带宽是稀缺资源，但计算量比较充足。这使我们能够利用密码构建模块（特别是阈值公钥加密），在经典的容错数据库设置中，这被认为是过于昂贵，其主要目标是即使在争用下也使响应时间最小化。

在异步网络中，消息最终被传递，但是没有其他的时间假设。不同于现有的弱同步协议，参数调整可以挑剔，HonBadgerBFT不在乎。无论网络条件如何波动，HoneyBadgerBFT的吞吐量总是紧密跟踪网络的可用带宽。不准确地说，只要消息最终得到传递，HoneyBadBFT就会取得进展;而且，一旦消息传递就会取得进展。

我们正式证明了我们的HoneyBadBird协议的安全性和活性，并且通过实验证明，即使在最优化的情况下，它也比传统的PBFT协议提供更好的吞吐量。

实施和大规模的实验。我们提供全面的服务，我们将在近期推出免费的开放源码软件HoneyBadgerBFT。我们演示了亚马逊AWS部署的实验结果，分布在5大洲的节点超过100个。为了展示其多功能性和鲁棒性，我们还在Tor匿名中继网络上部署了HoneyBadgerBFT，而不改变任何参数，并呈现吞吐量和延迟结果。

1.2建议的部署方案

在许多可以想象的应用中，我们强调了银行，金融机构和完全分散加密货币的倡导者所追求的两种可能的部署方案。

联邦加密货币。比特币等分散式加密货币的成功激发了银行和金融机构对新的交易处理和结算基础设施的重新审视。 “联邦加密货币”是一个经常被引用的愿景[24,25,48]，其中金融机构的一个联合体共同为拜占庭协议协议做出贡献，以允许快速和稳健的交易结算。这种方式将激化当前缓慢而笨重的银行间结算基础设施。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]。

联邦加密货币需要在广域网上部署BFT协议，可能涉及数百到数千个共识节点。在这种情况下，登记可以很容易地被控制，这样一套共识节点是先验知道的 - 通常被称为“许可”区块链。很明显，HoneyBadgerBFT是在这种联合加密货币中使用的自然候选者。

适用于无权区块链。相比之下，诸如比特币和以太坊这样的分散式加密货币则选择了“无权限”区块链，在这种区块链中，任何人都可以进行注册，节点可以动态和频繁地加入和离开。为了在这种情况下实现安全性，已知的共识协议依靠工作证明来打败Sybil攻击，并且在吞吐量和延迟方面付出巨大代价，例如，比特币每10分钟提交一次交易，吞吐量受到即使在当前块大小最大时也是7 tx / sec。最近的一些作品提出了一个有希望的想法，即利用像比特币这样的较慢的外部区块链或涉及基础货币本身的经济的“证明权益”假设[32,32,35,37]来引导更快的BFT协议，选择一个随机委员会在每个不同的时代进行BFT。这些方法有望实现两全其美，在公开招生中的安全性，分散式网络以及符合传统BFT协议的吞吐量和响应时间。在这里，HoneyBadgerBFT也是一个很自然的选择，因为随机选择的委员会可能在地理上是异构的。

2.背景和相关工作

我们的总体目标是建立一个复制的状态机，客户端生成和提交事务，并由节点网络接收和处理它们。 从应用程序特定的细节（如如何表示状态和计算转换）中抽象出来，只需构建一个完全全局一致的，完全有序的，仅追加的事务日志即可。 传统上，这种基元被称为全序或原子广播[23]; 用比特币的说法，我们称之为区块链。

容错状态机复制协议提供了强大的安全性和活性保证，允许分布式系统提供正确的服务，尽管网络延迟和某些节点的故障。 大量的工作研究了这样的协议，提供不同的性能权衡，容忍不同形式的故障和攻击，并对底层网络做出不同的假设。 我们在下面解释与我们最密切相关的努力。

2.1强大的BFT协议

尽管Paxos [36]，Raft [46]等许多知名协议容忍崩溃错误，但从PBFT [20]开始的拜占庭容错协议（BFT）甚至可以容忍任意（例如恶意）损坏的节点。许多后续的协议提供了改进的性能，通常是通过乐观的执行，当没有错误，客户端不会竞争，网络运行良好，并且至少有一些进步， 33，39，51]。

一般而言，BFT系统在延迟和CPU是瓶颈的部署场景中进行评估[49]，因此最有效的协议可以减少轮次数并将昂贵的加密操作减至最少。

Clement等人[22]通过倡导改善最坏情况的表现，启动了最近的一系列工作[4,6,10,21,22,50]，即使在系统受到攻击时也能提供服务质量保证 - 即使这是在乐观情况下的表现损失。然而，虽然这种“鲁棒BFT”协议优雅地容忍了被攻破的节点，但它们仍然依赖于对底层网络的时间假设。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随机协议

确定性的异步协议对于大多数任务是不可能的[27]。虽然绝大多数实际的BFT协议通过做出时间假设来避免这种不可能性的结果，但随机性（特别是密码学）提供了另一种途径。事实上，我们知道异步BFT协议适用于各种任务，如二进制协议（ABA），可靠广播（RBC）等[13,15,16]。

我们的工作与SINTRA [17]最密切相关，这是一个基于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（ABC）到通用子集协议（ACS）的降低，以及从ACS到多值验证协议（MVBA）的降低。

我们贡献的关键发明是从ABC到ACS的新颖减少，通过批量提供更好的效率（通过O（N）因子），同时使用阈值加密来保持审查弹性（见第4.4节）。通过从文献中挑选更好的子组分实例，我们也获得了更好的效率。尤其是，我们通过使用替代ACS [9]以及高效的RBC [18]来回避昂贵的MVBA原始，如第4.4节所述。

表1总结了HoneyBadgerBFT与其他几种原子广播协议的渐进性能。这里“通信。 “表示每个已提交事务的预期通信复杂度（即传送的总字节数）。由于PBFT依赖于较弱的同步假设，因此在异步网络中可能无法取得进展。协议KS02 [34]和RC05 [47]是乐观的，回落到昂贵的基于MVBA的恢复模式。如提到的Cachin等人的方案（CKPS01）[15]可以使用更高效的ACS进行改进。

表1：原子广播协议的渐近通信复杂度（预期的比特数）

|  |  |  |  |
| --- | --- | --- | --- |
|  | Async? | Comm. compl. | |
| Optim. | Worst |
| PBFT | no | *O*(*N*) | ∞ |
| KS02 [[34]](#_bookmark59)  RC05 [[47]](#_bookmark71) | yes yes | *O*(*N*2)  *O*(*N*) | *O*(*N*3)  *O*(*N*3) |
| CKPS01 [[15]](#_bookmark39)  CKPS01 [[15]+](#_bookmark39) [[9,](#_bookmark33) [18]](#_bookmark41)  **HoneyBadgerBFT (this work)** | yes yes  **yes** | *O*(*N*3)  *O*(*N*2)  **O**(**N**) | *O*(*N*3)  *O*(*N*2)  **O**(**N**) |

建设性意见[9,18] 我们还通过我们的减少获得另一个O（N）的改进。

最后，King和Saia [30,31]最近通过在稀疏图上路由通信来开发了消息数少于二次的协议协议。 但是，将这些结果扩展到异步设置仍然是一个悬而未决的问题。

3.异步和弱同步网络模型之间的差距

几乎所有现代BFT协议都依赖时序假设（如部分或弱同步）来保证活跃性。纯粹的异步BFT协议近年来受到的关注较少。考虑下面这个论点，如果它坚持的话，那么这个狭义的焦点是合理的：

[X]弱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，即使是异步协议也会带来不可接受的性能。

在这一节中，我们提出两个驳斥上述前提的反驳。首先，我们说明异步和弱同步网络模型之间的理论分离。具体而言，我们构建了一个敌手网络调度器，违反了PBFT的弱同步假设（并且确实导致它失败），但是在这个假设下，任何纯粹的异步协议（如HoneyBadgerBFT）都取得了很好的进展。其次，我们做一个实际的观察：即使满足了他们的假设，弱同步协议一旦恢复，网络分区恢复的速度就会很慢，而异步协议只要消息被传递，就会取得进展。

3.1时间假设的多种形式

在继续之前，我们回顾一下各种时间假设的标准形式。在异步网络中，攻击者可以随时以任何顺序发送消息，但是最终必须发送在正确节点之间发送的每条消息。异步网络中的节点实际上对“实时”时钟没有用处，只能根据收到的消息的顺序进行操作。

众所周知的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更强的时间假设。一个方便的（但非常强大的）网络假设是同步的：一个Δ同步网络保证每个发送的消息在最多延迟Δ（其中Δ是实时的量度）。

较弱的时间假设有几种形式。在未知的Δ模型中，协议不能使用延迟界限作为参数。或者，在最终的同步模型中，消息延迟限制Δ只能保证在某个（未知）时刻之后保持，称为“全局稳定时间”。总的来说，这两个模型被称为部分同步[26]。

另一个变化是弱同步[26]，其中延迟边界是随时间变化的，但最终不会比时间多项式函数增长更快[20]。

在可行性方面，上述内容是等同的 - 一个成功的一个协议可以系统地适用于另一个协议。然而就具体性能而言，调整弱同步意味着随着时间逐渐增加超时参数（例如，通过“指数退避”策略）。正如我们稍后所示，从瞬时网络分区中恢复时会导致延迟。

协议通常以超时事件的形式表现这些假设。例如，如果当事人在一定时间间隔内未发现有进展，则采取纠正措施，如选举新领导。异步协议不依赖计时器，并且无论实际的时钟时间如何，每当传递消息时都会取得进展。

在异步网络中计数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Canetti和Rabin [19]所解释的）是让攻击者为每个消息分配一个虚拟的轮数，这个条件是每个（r 1） - 正确的节点之间的消息必须在任何（r + 1） - 消息被发送。

3.2当弱同步失败时

我们现在继续描述为什么当网络条件是敌对的（或不可预测的）时，弱同步BFT协议可能会失败（或者性能下降）。这激发了为什么这样的协议不适合第1节中描述的面向加密货币的应用场景。

阻碍PBFT的网络调度器。我们使用经典的基于领导的BFT协议的实际基本容错（PBFT）[20]，一个代表性的例子来描述敌对网络调度器如何能够引起一类基于领导的BFT协议[4，6， 10，22，33，50]嘎然而止。

在任何时候，指定的领导者都有责任提出下一批交易。如果没有取得进展，无论是因为领导是有缺陷的还是因为网络已经停滞，那么节点试图选出一个新的领导。 PBFT协议严格依赖弱同步网络来实现活跃性。我们构建了一个违背这个假设的对抗调度器，并且实际上阻止了PBFT取得任何进展，但是HonBeyBFT（事实上，任何异步协议）都表现良好。当这些假设被违反时，基于时间假设的协议失败就不足为奇了;然而，展示明确的攻击有助于激励我们的异步构建。

我们调度程序背后的直觉很简单。首先，我们假设单个节点已经崩溃。然后，每当一个正确的节点是领导时，网络就会延迟消息，阻止进展，并导致以循环顺序的下一个节点成为新领导。当崩溃的节点成为下一个成为领导者时，调度器立即修复网络分区，并在诚实的节点之间非常迅速地传递消息;然而，由于领导人坠毁，这里也没有任何进展。

这种攻击违反了弱同步的假设，因为它必须延迟每个周期的消息越来越长，因为PBFT在每次失败的领导者选举之后扩大其超时间隔。另一方面，它提供了越来越大的同步时间。然而，由于这些同步时间在不方便的时候发生，所以PBFT不能使用它们。展望未来，Honey-BadgerBFT，甚至任何异步协议，都将能够在这些同步机会期间取得进展。

为了确认我们的分析，我们实施了这个恶意的调度程序，作为一个代理，截获和推迟所有查看更改消息给新的领导者，并对1200线的PBFT实现进行测试。我们观察到的结果和消息日志与上述分析一致;我们的副本卡在一个循环中，请求视图更改从未成功。在线完整版[42]中，我们给出了PBFT的完整描述，并解释了在这种攻击下它是如何表现的。

从网络分区恢复缓慢。即使弱同步假设最终得到满足，依赖于它的协议也可能很慢从瞬态网络分区恢复。考虑以下情况，它只是上述攻击的有限前缀：一个节点崩溃，网络临时分区持续时间为2DΔ。我们的调度程序治愈

网络划分恰恰是当坠毁节点成为领导者的时候。由于此时的超时间隔现在是2D +1Δ，所以在开始之前协议必须等待另一个2D +1Δ间隔

尽管网络在这个时间间隔内是同步的，但要选出一个新的领导者。鲁棒性和响应性之间的权衡。我们上面所观察到的这种行为并不是PBFT所特有的，而是依靠超时来应对崩溃的协议的基本内在。不管协议变体如何，从业者必须根据一些折衷调整其超时策略。在一个极端（最终同步）中，从业者对网络延迟Δ做出具体的估计。如果估计太低，那么系统可能根本就没有进展;太高，并且不利用可用带宽。在另一个极端（弱同步），从​​业者避免指定任何绝对延迟，但是必须选择影响系统跟踪变化条件的速度的“增益”。异步协议避免了调整这些参数的需要。

4. HoneyBadgerBFT协议

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

4.1问题定义：原子广播

我们首先定义我们的网络模型和原子广播问题。我们的设置涉及N个指定节点的网络，具有不同的众所周知的身份（0到N 1）。节点接收交易作为输入，其目标是就这些交易的排序达成共同的协议。我们的模型特别匹配“权限区块链”的部署场景，其中可以由任意客户端提交事务，但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，例如如何解释事务（以防止重播攻击，例如应用程序可能定义包含特征码和序列号的事务）。就我们的目的而言，交易只是唯一的字符串。在实践中，客户端会生成事务并将它们发送到所有节点，并且在收集来自大多数节点的签名后认为它们已被提交。为了简化我们的表述，我们没有明确地模拟客户，而是假定交易是由对手选择的，并作为输入提供给节点。同样，事务一旦被节点输出就被认为是提交的。

我们的系统模型做出以下假设：

（纯异步网络）我们假设每对节点通过一个可靠的认证的点对点信道连接起来不会丢弃消息。交付时间表完全由攻击者决定，但每个在正确节点之间发送的消息最终都必须交付。我们将根据异步回合的数量来描述协议的运行时间（如第2节所述）。由于网络可以任意延迟地排队消息，因此我们也假定节点具有无限的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）攻击者完全控制多达f个故障节点，其中f是一个协议参数。请注意，3 f + 1 N（我们的协议达到的）是此设置下广播协议的下限。

（受信任的设置）为了便于表示，我们假设节点可以在初始协议特定的设置阶段与可​​信的经销商交互，我们将使用这个阶段来建立公钥和秘密共享。请注意，在真实部署中，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（参见Boldyreva [11]）。我们所知的所有分布式密钥生成协议都依赖于时间假设;幸运的是，这些假设只需要在设置过程中保持不变。

定义1.原子广播协议必须满足以下特性，所有这些特性都应该在异步网络中具有高概率（作为安全参数λ的函数1 negl（λ）），并且不管任意的对手：

（协议）如果任何正确的节点输出一个事务tx，那么每个正确的节点输出tx。

(总顺序）如果一个正确的节点输出了交易序列（tx0，tx1，... tx j），另一个输出 tx0，tx1，... txjjj，则txi = txi代表i min（j，j）。

（审查弹性）如果事务tx被输入到N个正确的节点，那么它最终会被每个正确的节点输出。

审查弹性属性是一种活性属性，​​可防止对手阻止单个事务被提交。这个属性已经被其他名称所引用，例如Cachin等人的“公平性”。 [15]，但我们更喜欢这个描述性的短语。

性能指标。我们主要对分析原子广播协议的效率和事务延迟感兴趣。

（效率）假设每个诚实节点的输入缓冲区是足够满Ω（poly（N，λ））。那么效率是预期的每个节点的通信成本在所有已提交的交易中摊销。

由于每个节点必须输出每个事务，所以O（1）效率（我们的协议达到的）是渐近最优的。效率的上述定义假定网络负载不足，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。由于我们通过批量实现了良好的吞吐量，因此当交易很少发生时，我们的系统在低需求期间使用更多的带宽。如果我们的目标是最大限度地降低成本（例如，基于使用量的计费），没有这种资格的更强的定义将是适当的。

实际上，网络链路的容量是有限的，如果提交的交易数量超过网络处理能力，那么对于确认时间的保证一般是不能保证的。因此，我们定义交易延迟低于相关交易之前输入的交易数量。有限的交易延迟意味着审查弹性。

（事务延迟）假设对手将事务tx作为输入传递给N个正确的节点。假设T是“积压”，即先前输入到任何正确节点的事务总数与已经提交的事务数之间的差值。然后，交易延迟是在每个正确节点输出tx之前作为T的函数的预期异步回合数。

4.2概述和直觉

在HoneyBadgerBFT中，节点接收事务作为输入并将它们存储在它们的（无界）缓冲区中。该协议进入时代，在每个时代之后，新的一批交易被追加到提交日志。在每个纪元开始时，节点在缓冲区中选择一个事务子集（通过我们将要定义的策略），并将它们作为输入提供给一个随机协议协议的实例。在协议协议结束时，选择该时期的最后一组交易。

在这个高层次，我们的方法类似于现有的异步原子广播协议，特别是对于Cachin等人。 [15]，大规模事务处理系统（SINTRA）的基础。和我们一样，Cachin的协议是以异步公共子集（ACS）原语的一个实例为中心的。粗略地说，ACS原语允许每个节点提出一个值，并保证每个节点输出一个包含至少N 2 f个正确节点的输入值的公共向量。从这个原语构建原子广播是微不足道的 - 每个节点只是从它的队列的前面提出一个事务子集，并输出在同意的向量中元素的联合。但是，有两个重要的挑战。

挑战1：实现审查复原力。 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个这样的集合，所以我们可以通过确保节点提出大多数不相交的事务集合来提高整体效率，从而以相同的成本在一批中提交更明确的事务。因此，我们协议中的每个节点不是简单地从缓冲区中选择第一个元素（如CKPS01 [15]），而是提出一个随机选择的样本，每个事务平均仅由一个节点提出。

然而，天真地实施，这种优化将会增强审查弹性，因为ACS原语允许广告人选择最终包含哪些节点的提案。对手可以选择性地审查一个交易，不管哪个节点提出这个交易。我们通过使用阈值加密来避免这个陷阱，这可以防止攻击者知道哪个节点提出了哪个交易，直到达成协议。完整的协议将在4.3节中描述。

挑战2：实际吞吐量。虽然异步ACS和原子广播的理论可行性已经被人们所知[9,15,17]，但它们的实际性能却不是。据我们所知，唯一执行ACS的其他工作是Cachin和Portiz [17]，他们表示在广域网上可以达到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能够在实践中达到高吞吐量。

在本文中，我们展示了通过将精心挑选的子组件拼接在一起，我们可以高效地实例化ACS，并在渐近和实践中获得更大的吞吐量。值得注意的是，我们改进了从O（N2）（如Cachin等[15,17]）到O（1）的ACS的渐近代价（每个节点）。由于组件

我们的cherrypick以前没有一起提交过（据我们所知），我们在4.4节提供了一个完整的结构描述。

模块化协议组成。我们现在准备正式提出我们的建筑。在此之前，我们对我们演讲的风格进行评论。我们以模块化的方式定义我们的协议，每个协议可以运行其他（子）协议的多个实例。外部协议可以提供输入和从子协议接收输出。即使在提供输入之前（例如，如果它从其他节点接收消息），节点也可以开始执行（子）协议。

隔离这样的（子）协议实例是很重要的，以确保与一个实例有关的消息不能在另一个实例中重播。在实践中，通过将每个（子）协议实例与一个唯一的字符串（一个会话标识符）关联起来，用这个标识符标记在这个（子）协议中发送或接收的任何消息，并相应地路由消息。我们在我们的协议描述中隐藏了这些消息标签，以便读取。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，各方之间的异步通信是通过认证的异步通道。实际上，这样的渠道可以使用TLS套接字实例化，例如，我们在第5节中讨论。

为了区分协议内各方之间发送的不同消息类型，我们在打字机字体中使用标签（例如，VAL（m）表示VAL类型的消息m）。

4.3从异步通用子集构造HoneyBadgerBFT

构件块：ACS。我们的主要构件是一个叫做异步公共子集（ACS）的原语。构建ACS的理论可行性已经在几篇论文中得到证实[9,15]。在本节中，我们将介绍ACS的正式定义，并将其用作构建HoneyBadgerBFT的黑盒。在4.4节的后面，我们将通过结合过去被忽略的几个结构来显示，我们可以高效地实例化ACS。更正式地说，ACS协议满足以下性质：

•（有效性）如果正确的节点输出一个集合v，则| v | ≥N-f，v包含至少N - 2个正确节点的输入。 （协议）如果一个正确的节点输出v，那么每个节点输出v。

（总体）如果N个正确的节点接收到一个输入，则所有正确的节点产生一个输出。

构建模块：阈值加密。阈值加密方案TPKE是允许任何一方将消息加密成主公钥的加密原语，使得网络节点必须一起工作来解密它。一旦f + 1正确的节点计算并显示密文的解密份额，明文可以被恢复;直到至少有一个正确的节点显示出其解密份额，攻击者才会对明文没有任何了解。阈值方案提供以下界面：

TPKE.Setup（1λ）PK，SKi生成一个公共加密

密钥PK以及每个参与方的密钥SKi TPKE.Enc（PK，m）C加密消息m TPKE.DecShare（SKi，C）σi产生解密的第i份（或如果C格式错误）

TPKE.Dec（PK，C，i，σi）m合并来自至少第f + 1个成员的一组解密份额i，获得明文m（或者如果C包含无效份额，则无效份额被识别）。

在我们的具体实例中，我们使用Baek和Zheng的门限加密方案[7]。这个方案也是健壮的（根据我们的协议的要求），这意味着即使对于一个敌手生成的密文C，最多也只能得到一个明文（除此之外）。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，该方案满足了明显的正确性和IND-CPA游戏的阈值版本

来自ACS的原子广播。我们现在更详细地描述我们的原子广播协议，在图1中定义。

如前所述，这个协议是以ACS的一个实例为中心的。为了获得可扩展的效率，我们选择一个配料策略。我们让B是一个批量大小，并将在每个时期提交Ω（B）交易。每个节点从其队列中提出B / N事务。为了确保节点主要提出不同的事务，我们从每个队列中的第一个B中随机选择这些事务。

正如我们将在4.4节看到的那样，我们的ACS实例具有O（N2 v +λN3 log N）的总的通信成本，其中v约束任何节点的输入的大小。因此，我们选择批量大小B =Ω（λN2 log N），以便来自每个节点的贡献（B / N）吸收这种附加开销。

为了防止敌手影响结果，我们使用阈值加密方案，如下所述。简而言之，每个节点选择一组事务，然后对其进行加密。每个节点然后将加密作为输入传递给ACS子例程。 ACS的输出因此是密文的向量。一旦ACS完成，密文就被解密。这保证了在对手了解每个节点提出的建议的具体内容之前，交易的集合是完全确定的。这保证了对手在有足够正确节点的队列之前不能有选择地阻止事务被提交。

4.4高效实例化ACS

Cachin等人提出了一个协议，我们称之为CKPS01（暗示）将ACS减少为多值验证的拜占庭协议（MVBA）[15]。粗略地说，MVBA允许节点提出满足谓词的值，最终选择其中的一个。减少很简单：验证谓词表示输出必须是来自至少N个方的签名输入的向量。不幸的是，MVBA原始协议成为一个瓶颈，因为我们所知道的唯一的结构会导致O（N3 v）的开销。

我们通过使用ACS完全避开MVBA的替代实例来避免这个瓶颈。我们使用的实例是由Ben-Or et al。 [9]，我们认为，有些被忽视。事实上，它早于CKPS01 [15]，最初是为了一个非常不相关的目的而开发的（作为实现高效异步多方计算的工具）。该协议是从ACS到可靠广播（RBC）和异步二进制拜占庭协议（ABA）的缩减。直到最近，我们才知道这些子组件的高效构造，我们稍后解释。

在较高的水平上，ACS协议分两个主要阶段进行。在第一阶段中，每个节点使用RBC将其建议值传播给其他节点，然后由ABA决定一个指示哪个RBC成功完成的位向量。

我们现在在更详细地解释Ben-Or协议之前简要解释RBC和ABA结构。

Let *B* = Ω(*λ N*2 log *N*) be the batch size parameter.

**Algorithm HoneyBadgerBFT (for node** *Pi***)**

Let PK be the public key received from TPKE*.*Setup (executed

Let buf := [ ] be a FIFO queue of input transactions.

Proceed in consecutive epochs numbered *r*:

*// Step 1: Random selection and encryption*

by a dealer), and let SK*i* be the secret key for *Pi*.

* let proposed be a random selection of |*B/N*∫ transactions from

the first *B* elements of buf

* encrypt *x* := TPKE*.*Enc(PK*,* proposed)

*// Step 2: Agreement on ciphertexts*

* receive {*v j* } *j*∈*S*, where *S* ⊂ [1*..N*], from ACS[*r*]*// Step 3: Decryption*
* pass *x* as input to ACS[*r*] *//see Figure* [*4*](#_bookmark13)
* for each *j* ∈ *S*:

let *e j* := TPKE*.*DecShare(SK*i, v j* )

* let block*r* := sorted(∪ *j*∈*S*{*y j* }), such that block*r* is sorted in a

multicast DEC(*r, j, i, e j* )

wait to receive at least *f* + 1 messages of the form

DEC(*r, j, k, e j,k* )

decode *y j* := TPKE*.*Dec(PK*,* {(*k, e j,k* )})

canonical order (e.g., lexicographically)

* set buf := buf − block*r*

图1：HoneyBadgerBFT。

通信最优可靠的路演。异步可靠的广播信道满足以下特性：

（协议）如果任何两个正确的节点提供v和vj，则v = vj。

（Totality）如果任何正确的节点传递了v，那么所有正确的节点传递v

（有效性）如果发送者是正确的并且输入v，则所有正确的节点发送v

虽然Bracha [13]经典的可靠广播协议需要O（N2v）比特的全部通信来传播大小为v的消息，但Cachin和Tessaro [18]观察到，消除编码可以将这个成本降低到仅仅是O （N v +λN2 log N），即使在最坏的情况下。这对于大消息（即，当vλN log N）是显着的改进，（回顾到4.3节）指导了我们对批量大小的选择。在这里使用擦除编码诱导

最小的开销常数，等于N N f <3。

如果发送者是正确的，总的运行时间是三（异步）轮;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输出一个值之间最多有两个回合。可靠的广播算法如图2所示。

图2：可靠的广播算法，改编自Bracha的广播[13]，用纠删码来提高效率[18]

**Algorthm RBC (for party** *Pi***, with sender** *P*Sender **)**

* upon input(*v*) (if *Pi* = *P*Sender):

let {*s j* } *j*∈[*N*] be the blocks of an (*N* − 2 *f , N*)-erasure coding

scheme applied to *v*

let *h* be a Merkle tree root computed over {*s j* }

send VAL(*h, b j , s j* ) to each party *P j* , where *b j* is the *jth*

Merkle tree branch

* upon receiving VAL(*h, bi, si*) from *P*Sender,

multicast ECHO(*h, bi, si*)

* upon receiving ECHO(*h, b j , s j* ) from party *P j* ,

check that *b j* is a valid Merkle branch for root *h* and leaf *s j* , and otherwise discard

* upon receiving valid ECHO(*h,* ·*,* ·) messages from *N* − *f* distinct

parties,

**–** interpolate {*s*j*j* } from any *N* − 2 *f* leaves received

**–** if READY(*h*) has not yet been sent, multicast READY(*h*)

**–** recompute Merkle root *h*j and if *h*j ƒ= *h* then abort

* upon receiving *f* + 1 matching READY(*h*) messages, if READY

has not yet been sent, multicast READY(*h*)

* upon receiving 2 *f* + 1 matching READY(*h*) messages, wait for

*N* − 2 *f* ECHO messages, then decode *v*

二元协议。二进制协议是一个标准的原语，允许节点就一个位的价值达成一

致。更正式地说，二元协议保证了三个属性：

（协议）如果任何正确的节点输出位b，则每个正确的节点输出b。

（终止）如果所有正确的节点都接收到输入，则每个正确的节点输出一个位。

（有效性）如果有任何正确的节点输出b，则至少有一个正确的节点接收到b作为输入。

有效性属性意味着一致：如果所有正确的节点都接收到相同的输入值b，则b必须是决定的值。另一方面，如果任何时候两个节点接收不同的输入，

那么对手甚至可以在剩余节点接收输入之前强制决定值。

我们用Moustefaoui等人的一个协议来实例化这个原语。 [43]，这是基于一个密码普通硬币。我们把这个实例化的解释推迟到在线完整版本[42]。它的预期运行时间是O（1），实际上在O（k）轮内以概率1 2 -k完成。每个节点的通信复杂度为O（Nλ），这主要是因为在普通硬币中使用了门限密码。

同意建议值的一个子集。把上述部分放在一起，我们使用Ben-Or等人的协议。 [9]就一组包含至少N f个节点的整个提议的值达成一致。

在较高的水平上，这个协议进行了两个主要阶段。在第一阶段，我使用可靠的广播将其建议值传播给其他节点。在第二阶段，使用N个并发的二进制拜占庭协议实例来表示同意向量{b j} j 1 N，其中b j = 1表示P j的建议值-2∈[..]被包含在最终的集合中。

其实上面的简单描述隐藏了一个微妙的挑战，本 - 奥提供了一个聪明的解决方案。

为了实现上述草图，一个天真的尝试将使每个节点等待第一个（N f）广播完成，然后为与其相对应的二进制协议实例提议1，为所有其他实例提供0。但是，正确的节点可能会以不同的顺序观察完成的广播。由于二进制协议只保证输出为1，如果所有正确的节点一致地建议1，则可能的结果位向量可能是空的。

为了避免这个问题，节点放弃提议0，直到他们确定最终向量将具有至少N f个位。为了给这个协议的流程提供一些直觉，我们叙述

图3中的几种可能的情况。来自Ben-Or等人的算法[9]在图4中给出。预期运行时间为O（log N），因为它必须等待所有二进制协议实例完成。

**RBC2**

**No**

Coin

Coin

Coin

**BA2**

….

**Yes**

**V1**

Coin

Coin

**BA1**

**RBC1**

**Yes**

（a）正常（b）等待慢速广播（c）广播失败

上接图3（ACS执行的例子）我们协议的每个执行包括运行N个并发的可靠广播（RBC）实例，以及N个拜占庭协议（BA），而这个实例又使用预期的常数普通的硬币。 （a）在通常情况下，节点0从索引1处的可靠广播接收到值V1（节点1的建议值）。因此节点0提供了这些实例如何播出输入“是”给BA1，输出“是”。（b）RBC2花费太长时间才能完成，节点0已经接收到（N f）“是”输出，所以它对BA2投出“否”。但是，其他节点已经看到RBC2成功完成，所以BA2导致“是”，节点0必须等待V2。 （c）在RBC3完成之前，BA3以“否”结束。

当使用上述可靠的广播和二进制协议结构实例化时，总的通信复杂度为O（N2 v +λN3 log N），假设v是任何节点输入的最大尺寸。

4.5分析

首先我们观察到协议和总订单属性立即从ACS的定义和TPKE方案的鲁棒性。

定义1.（协议和总订单）。 HoneyBadBirdBFT协议满足协议和总订货性能，除了可忽略的概率。

证明：这两个属性立即从高级原型ACS，TPKE的属性开始。每个ACS实例保证节点在每个时期达成一个密文向量（步骤2）。 TPKE的健壮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）。这就够确保协议和总的秩序。

定理2.（复杂性）假设批量大小为B =Ω（λN2 log N），每个HoneyBadgerBFT时期的运行时间为O（log N），期望通信复杂度为O（B）。

证明： ACS的成本和运行时间在4.4节中解释。阈值解密的N个实例会产生一个附加的轮次和O（λN2）的附加成本，这不会影响整体渐近代价。

Coin

Coin

**RBC3**

**BA3**

HoneyBadgerBFT协议可能在一个时期内承诺B事务。然而，实际的数量可能会少于这个，因为一些正确的节点可能会提出重叠的事务集，其他的可能会响应得太

晚，并且被破坏的节点可能会提出一个空集。幸运的是，我们证明（在线完整版本[42]），假设每个正确节点的队列已满，那么B / 4作为在一个时期提交的预期事务数的下限。

定义3.（效率）。假设每个正确节点的队列至少包含B个不同的事务，那么在一个时期中提交的预期事务数至少为B，从而产生恒定的效率。

最后，我们证明（在线完整版本[42]），该通信不能显着延迟任何事务的提交。

定律4.（审查复原）假设攻击者把事务tx作为输入传递给N个正确的节点。设T是“积压”的大小，即先前输入到任何正确节点的事务总数与已经提交的事务数之间的差值。那么tx在O（T / B +λ）时期内被提交，除了可忽略的概率。

5.实践和评估

在本节中，我们使用HoneyBadBirdBFT协议的原型实现来进行多个实验和性能测量。除非另有说明，否则本节中报告的数字默认为所有节点都表现诚实的乐观情况。

首先，我们演示HoneyBadgerBFT的确可以通过在广域网上进行实验进行扩展，其中包括五大洲的104个节点。即使在这些情况下，Honey-BadgerBFT也可以达到每秒数千次事务的峰值吞吐量。此外，通过与PBFT（代表性的部分同步协议）的比较，HoneyBadgerBFT仅执行一个小的常数因子。最后，我们证明了在Tor匿名通信层上运行异步BFT的可行性。

对于确定性删除编码，我们使用实现了ReedSolomon编码的zfec库[52]。为了实例化常见的硬币原语，我们实现了Boldyreva基于配对的门限签名方案[11]。对于事务的阈值加密，我们使用Baek和Zheng的方案[7]来加密一个256位的临时数据密钥，然后在CBC模式下使用AES256实际有效负载。我们使用PBC库的Charm [3] Python包装来实现这些门限密码方案[38]。对于阈值签名，我们使用提供的MNT224曲线，导致签名（和签名份额）只有65个字节，并启发式地提供112位的安全性。 6我们的门限加密方案需要一个

对称双线性群：因此我们使用SS512群试探性地提供了80位的安全性[45]。

在我们的EC2实验中，我们使用普通的（未经认证的）TCP插座。在真正的部署中，我们将使用两个客户端的TLS和服务器认证，长期增加微不足道的开销生活的会议。同样，在我们的Tor实验中，只有一个端点每个套接字的身份验证（通过“隐藏服务”地址）。

我们的理论模型假设节点具有无限的缓冲区。实际上，可以将更多资源动态添加到节点每当内存消耗达到水印时（例如，永远是75％）虽然我们的原型实现不

但包含此功能。未能提供足够的缓冲区将计入失败预算f。

5.1带宽分解和评估

我们首先分析我们系统的带宽成本。在所有的实验中，我们假定一个恒定的事务大小m T = 250个字节，这将允许一个ECDSA签名，两个公钥，以及应用有效负载（即，近似于典型的比特币交易）。我们的实验使用参数N = 4f，8，各方提出一批B / N交易。至建模最坏的情况，节点以相同的队列开始大小B。我们将运行时间记录为从一开始的时间（N-f）个节点输出一个值的时间。

带宽和故障结果。每个节点消耗的总体带宽也由一个固定的附加开销构成作为交易依赖的开销。对于所有参数值我们考虑到，加法开销主要是O（λN2）项由ABA阶段的门限密码体制造随后的解密阶段。 ABA阶段涉及到每个阶段期望节点发送4N 2个签名份额。只有RBC阶段产生一个交易相关的开销，相当于擦除编码扩展因子r = N/N-2f。该RBC也由于Merkle树，将N 2 logN散列贡献给开销包含在ECHO消息中的分支。总的沟通（每个节点）估计为：

m all = r（Bm T + Nm E）+ N 2（（1 + logN）m H + m D + 4m S）

其中m E和m D分别是密文的大小TPKE方案中的解密共享，并且m S是TSIG的大小签名份额。

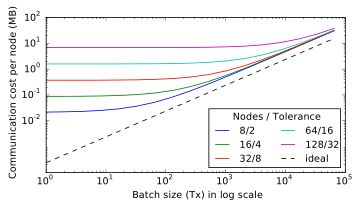


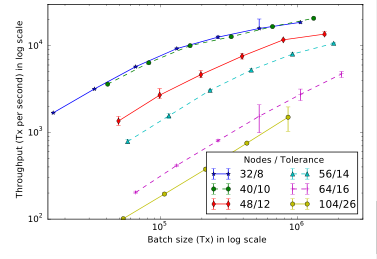
图5：对于不同的批量大小，以兆字节（每个节点）估计的通信成本。 对于小批量，固定成本随O（N2 log N）增长。 在饱和时，开销因子接近N/ N-2f <3。

图6：吞吐量（每秒提交的交易次数）与拟议的交易次数。 误差线表示95％的置信区间。

系统的有效吞吐量随着我们的增加而增加建议批量大小B，这样的交易依赖部分

的成本占主导地位。如图5所示，对于N = 128，批量最大可达1024个交易，独立于交易的带宽仍占主导地位。但是当批量大小的时候达到16384，交易依存部分开始占据主导地位- 很大程度上是由节点传输的RBC.ECHO阶段造成的擦除编码块。

5.2 Amazon EC2上的实验

为了看看我们的设计是如何实用的，我们部署了我们的协议于亚马逊EC2服务并全面测试其性能。我们在2,40,48,56,64和104亚马逊上运行了HoneyBagderBFT EC2 t2.medium实例遍布其8个跨越五大洲的地区。在我们的实验中，我们改变了批量大小使得每个节点建议256,512,1024,2048,4096，8192，16384，32768，65536或131072个交易。

吞吐量。吞吐量被定义为交易的数量

每单位时间承诺。在我们的实验中，我们使用“确认交易每秒“作为我们的测量单位，如果没有指定。图6显示了吞吐量和

所有N方提议的交易总数。错误公差参数设置为f = N / 4。

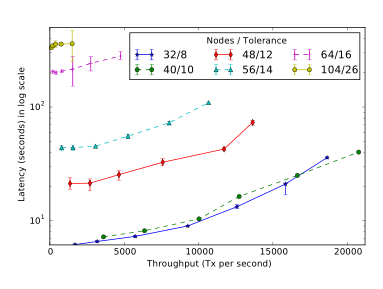


图7：广域网实验中的延迟与吞吐量的关系。 误差线表示95％的置信区间。

发现：从图6可以看出，对于每种情况，随着交易次数的增加，交易量都会增加。对于多达40个节点的中型网络，我们实现每秒吞吐量超过20,000次。对于一个大型的104节点网络，我们每秒可以达到1500次以上的交易。给定一个无限的批量大小，所有的网络规模最终都会收敛到一个共同的上限，仅受可用带宽的限制。虽然网络中消耗的总带宽随着每个附加节点的增加（线性）增加，但是额外的节点也会贡献额外的带宽容量。

吞吐量，延迟和缩放权衡。延迟定义为第一个节点接收到客户端请求和第（N f）个节点完成共识协议之间的时间间隔。这是合理的，因为完成协议的第（Nf）个节点意味着对诚实方达成共识。

图7显示了不同选择的N和f = N / 4时延和吞吐量之间的关系。正斜率表明我们的实验尚未完全饱和可用带宽，即使批量较大，我们也将获得更好的吞吐量。图7还显示，随着节点数量的增加，延迟增加，主要来自协议的ABA阶段。事实上，在N = 104的情况下，对于我们尝试过的批处理量的范围，我们的系统是受CPU限制的，而不是带宽限制的，因为我们的实现是单线程的并且必须验证O（N2）阈值签名。无论如何，我们最大的104个节点的实验在6分钟内完成。

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（具有相等的带宽供应）添加到网络，但是提交一个批次所消耗的最小带宽（以及因此的等待时间）随着O（N2logN）而增加。这种限制意味着对可扩展性的限制，宽度和用户的延迟容忍度。

与PBFT比较。图8显示了与用于部分同步网络的经典BFT协议PBFT协议的比较。我们使用Croman等人的Python实现。 [24]，在分布在亚马逊AWS地区的8,16,32和64个节点上运行。选择批量大小以使网络的可用带宽饱和。

基本上，虽然PBFT和我们的协议有相同的总体上渐近的通信复杂度，我们的协议在网络链路上平均分配这个负载，而PBFT在领头的可用带宽上瓶颈。因此，PBFT的可持续吞吐量随着节点数量的减少而减少，而HoneyBadgerBFT的大致保持不变。

请注意，这个实验只反映乐观的情况，没有故障或网络中断。即使对于小型网络，HoneyBadgerBFT也能在对抗条件下提供明显更好的鲁棒性，如第3节所述。特别是，PBFT将实现零吞吐量而不是敌对的异步调度器，而HoneyBadgerBFT将以常规速率完成时代。

5.3 Tor上的实验

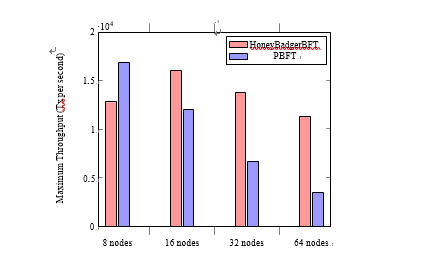
为了演示HoneyBadgerBFT的鲁棒性，我们运行了Tor（最成功的匿名通信网络）上执行的容错共识协议的第一个实例（据我们所知）。与我们原来的AWS部署相比，

图8：与EC2上的PBFT比较

Tor增加了显着的延迟。无论如何，我们显示我们可以运行HoneyBadgerBFT而不调整任何参数。隐藏在Tor背后的HonBygerBFT节点可能会提供更好的鲁棒性。由于它有助于节点隐藏其IP地址，因此可以帮助它们避免有针对性的网络攻击和涉及其物理位置的攻击。

Tor简要背景： Tor网络由大约6,500个继电器组成，这些继电器列在公共目录服务中。 Tor使得“隐藏的服务”成为服务器，通过Tor接受连接，以隐藏他们的位置。当客户建立到隐藏服务的连接时，客户端和服务器都将3跳电路构建到共同的“集合点”。因此，到隐藏服务的每个连接都通过5个随机选择的中继路由数据。 Tor为中继节点公布其容量和利用率提供了一种手段，这些自报的指标由Tor项目汇总。根据这些指标，网络的总容量为145Gbps，目前的利用率为65Gbps。

Tor实验设置。我们设计我们的实验设置，使得我们可以在运行Tor守护进程软件的单台桌面机器上运行所有N个HoneyBadgerBFT节点，同时能够实际反映Tor中继路径。为此，我们配置我们的机器监听N个隐藏的服务（在我们的模拟网络中，每个HoneyBadgerBFT节点都有一个隐藏的服务）。由于每个HoneyBadgerBFT节点与其他节点形成连接，我们每个实验总共构建了N2个Tor电路，从我们的机器开始和结束，并通过5个随机继电器。总之，所有的成对覆盖链路都要经过由随机中继节点组成的真正的Tor电路，这样设计的性能使得所得到的性能代表了一个真实的HoneyBadgerBFT在Tor上的部署（尽管所有模拟节点运行在一台主机上）。

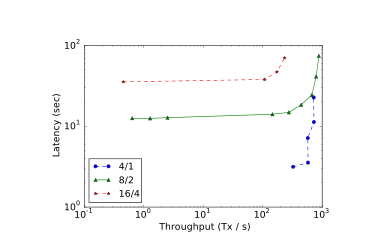


图9：在Tor上运行HonBadgerBFT的实验的延迟vs吞吐量。

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，这一点非常重要。我们只从单一的有利位置形成了连接（从而避免接收），并进行了短时间（几分钟）和小参数的实验（在我们最大的实验中只形成了256个电路）。总的来说，我们的实验涉及通过Tor传输大约五千兆字节的数据 - 小于其日常利用率的1E-5分数。

图9显示了延迟如何随吞吐量而变化。与我们的节点有足够带宽的EC2实验相反，Tor电路受限于电路中最慢的链路。我们实现了每秒Tor超过800次事务的最大吞吐量。

一般来说，通过Tor的中继网络传输的消息往往具有显着且高度可变的延迟。例如，在我们对8方提出每方16384个交易的实验中，一条消息可以延迟316.18秒，延迟方差超过2208，而平均延迟仅为12秒。我们强调，我们的协议不需要像传统的最终同步协议那样针对这种网络条件进行调整。

6.结论

我们已经介绍了HoneyBadgerBFT，第一个高效率和高吞吐量的异步BFT协议。通过我们的实施和实验结果，我们证明了HoneyBadgerBFT可以成为初始加密货币启发的容错事务处理系统部署的合适组件。更一般地说，我们相信我们的工作展示了建立基于异步协议的可靠和事务处理系统的承诺。

**致 谢** 我们感谢Jay Lorch，Jonathan Katz和EminGünSirer提供了有益的建议，尤其是Dominic Williams多次出色的讨论，激励我们解决了这个问题。这项工作部分由NSF拨款CNS-1314857，CNS-1453634，CNS-1518765，CNS-1514261和CNS-1518899支持，DARPA授予N66001-15-C-4066，帕卡德奖学金，斯隆奖学金，两项谷歌教师研究奖，以及VMWare研究奖。这项工作部分完成，而一部分作者访问加州大学伯克利分校的学生，而一部分作者访问了西蒙斯计算理论研究所，由西蒙斯基金会和DIMACS /西蒙斯合作组织支持密码通过NSF授予CNS-1523467。

**参考文献**

[1] How a Visa transaction works. http://apps.usa.visa.com/merchants/become-a-merchant/ how-a-visa-transaction-works.jsp, 2015.

[2] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable byzantine fault-tolerant services. ACM SIGOPS Operating Systems Review, 39(5):59–74, 2005.

[3] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano,

M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. Journal of Cryptographic Engineering, 3(2):111–128, 2013.

[4] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack. Dependable and Secure Computing, IEEE Transactions on, 8(4):564–577, 2011.

[5] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane,

C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling byzantine fault-tolerant replication to wide area networks. Dependable and Secure Computing, IEEE Transactions on, 7(1):80–93, 2010.

[6] P.-L. Aublin, S. Ben Mokhtar, and V. Quéma. Rbft: Redundant byzantine fault tolerance. In Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on, pages 297–306. IEEE, 2013.

[7] J. Baek and Y. Zheng. Simple and efficient threshold cryptosystem from the gap diffie-hellman group. In Global Telecommunications Conference, 2003. GLOBECOM’03. IEEE, volume 3, pages 1491–1495. IEEE, 2003.

[8] M. Ben-Or and R. El-Yaniv. Resilient-optimal interactive consistency in constant time. Distributed Computing, 16(4):249–262, 2003.

[9] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, pages 183–192. ACM, 1994.

[10] A. Bessani, J. Sousa, and E. E. Alchieri. State machine replication for the masses with bft-smart. In Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on, pages 355–362. IEEE, 2014.

[11] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Public key cryptographyâA˘TˇPKC 2003, pages 31–46. Springer, 2002.

[12] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. Kroll, and

E. W. Felten. Research perspectives on bitcoin and second-generation digital currencies. In 2015 IEEE Symposium on Security and Privacy. IEEE, 2015.

[13] G. Bracha. Asynchronous byzantine agreement protocols.

Information and Computation, 75(2):130–143, 1987.

[14] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. In Proceedings of the 7th symposium on Operating systems design and implementation, pages 335–350. USENIX Association, 2006.

[15] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In Advances in Cryptology – Crypto 2001, pages 524–541. Springer, 2001.

[16] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantipole: Practical asynchronous byzantine agreement

using cryptography. In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, pages 123–132. ACM, 2000.

[17] C. Cachin, J. Poritz, et al. Secure intrusion-tolerant replication on the internet. In Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on, pages 167–176. IEEE, 2002.

[18] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium on, pages 191–201. IEEE, 2005.

[19] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, pages 42–51. ACM, 1993.

[20] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In OSDI, volume 99, pages 173–186, 1999.

[21] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi,

M. Dahlin, and T. Riche. Upright cluster services. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pages 277–290. ACM, 2009.

[22] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and

M. Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In NSDI, volume 9, pages 153–168, 2009.

[23] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. Citeseer, 1986.

[24] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels,

A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. W. and. On scaling decentralized blockchains — a position paper. 3rd Bitcoin Research Workshop, 2015.

[25] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. arXiv preprint arXiv:1505.06895, 2015.

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

[27] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM), 32(2):374–382, 1985.

[28] A. Guillevic. Kim-barbulescu variant of the number field sieve to compute discrete logarithms in finite fields. https://ellipticnews.wordpress.com/2016/05/02/kim- barbulescu-variant-of-the-number-field-sieve-to-compute- discrete-logarithms-in-finite-fields/, May 2016.

[29] T. Kim and R. Barbulescu. Extended tower number field sieve: A new complexity for medium prime case. Technical report, IACR Cryptology ePrint Archive, 2015: 1027, 2015.

[30] V. King and J. Saia. From almost everywhere to everywhere: Byzantine agreement with O(n3/2) bits. In Distributed Computing, pages 464–478. Springer, 2009.

[31] V. King and J. Saia. Breaking the O(n2) bit barrier: scalable byzantine agreement with an adaptive adversary. Journal of the ACM (JACM), 58(4):18, 2011.

[32] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi,

L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. arXiv preprint arXiv:1602.06997, 2016.

[33] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. In ACM SIGOPS Operating Systems Review, volume 41, pages 45–58. ACM, 2007.

[34] K. Kursawe and V. Shoup. Optimistic asynchronous atomic broadcast. In in the Proceedings of International Colloqium on Automata, Languages and Programming (ICALP05)(L Caires, GF Italiano, L. Monteiro, Eds.) LNCS 3580. Citeseer, 2001.

[35] J. Kwon. TenderMint: Consensus without Mining, August 2014.

[36] L. Lamport. The part-time parliament. ACM Transactions on Computer Systems (TOCS), 16(2):133–169, 1998.

[37] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and

P. Saxena. Scp: A computationally-scalable byzantine consensus protocol for blockchains. Cryptology ePrint Archive, Report 2015/1168, 2015. http://eprint.iacr.org/.

[38] B. Lynn. On the implementation of pairing-based cryptography. The Department of Computer Science and the Committee on Graduate Studies of Stanford University, 2007.

[39] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: building efficient replicated state machines for wans. In OSDI, volume 8, pages 369–384, 2008.

[40] R. McMillan. Ibm bets big on bitcoin ledger. Wall Street Journal.

[41] R. McMillan. How bitcoin became the honey badger of money. Wired Magazine, http://www.wired.com/2013/12/bitcoin\_honey/, 2013.

[42] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. [Online full version] http://eprint.iacr.org/2016/199, 2016.

[43] A. Mostefaoui, H. Moumen, and M. Raynal. Signature-free asynchronous byzantine consensus with t< n/3 and o (n 2) messages. In Proceedings of the 2014 ACM symposium on Principles of distributed computing, pages 2–9. ACM, 2014.

[44] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcon.org/bitcoin.pdf, 2008.

[45] NIST. Sp 800-37. Guide for the Security Certification and Accreditation of Federal Information Systems, 2004.

[46] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In Proc. USENIX Annual Technical Conference, pages 305–320, 2014.

[47] H. V. Ramasamy and C. Cachin. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In OPODIS, pages 88–102. Springer, 2005.

[48] D. Schwartz, N. Youngs, and A. Britto. The Ripple Protocol Consensus Algorithm, September 2014.

[49] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe. Bft protocols under fire. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI’08, pages 189–204, Berkeley, CA, USA, 2008. USENIX Association.

[50] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung. Spin one’s wheels? byzantine fault tolerance with a spinning primary. In Reliable Distributed Systems, 2009. SRDS’09. 28th IEEE International Symposium on, pages 135–144. IEEE, 2009.

[51] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung. Ebawa: Efficient byzantine agreement for wide-area networks. In High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on, pages 10–19. IEEE, 2010.

[52] Z. Wilcox-O’Hearn. Zfec 1.4. 0. Open source code distribution: http:// pypi.python.org/ pypi/ zfec, 2008.

1. Cornell University [↑](#footnote-ref-2)
2. Tsinghua University [↑](#footnote-ref-3)
3. University of California, Berkeley [↑](#footnote-ref-4)
4. Cornell University [↑](#footnote-ref-5)
5. University of Illinois, Urbana-Champaign

   **摘 要** 加密货币的惊人成功导致了对用于任务关键型应用（比如金融交易）的大规模，高度稳健的拜占庭式容错（BFT）协议的兴趣激增。虽然传统的观点是建立在诸如PBFT（或其变体）之类的（弱）同步协议之上，但是这样的协议严重依赖于网络定时假设，并且只有当网络按预期行事时才保证活跃性。我们认为这些协议不适合这种部署方案。我们提出了另一种方法，即HoneyBadgerBFT，这是第一个实用的异步BFT协议，它保证了活性，而不需要做任何时间假设。我们将我们的解决方案建立在一个新颖的原子广播协议基础上，可以达到最佳的渐近效率。我们提出了一个实现和实验结果，以显示我们的系统可以实现每秒数万个事务的吞吐量，并在广域网上扩展到超过一百个节点。我们甚至可以在Tor上进行BFT实验，而不需要调整任何参数。不同于其他选择，HoneyBadgerBFT根本不受底层网络影响。

   **关键词** HoneyBadgerBFT异步网络 原子广播Tor

   **The Honey Badger of BFT Protocols**

   Elaine Shi1  Yu Xia2  Dawn Song3  Kyle Croman4 Andrew Miller5

   1 Cornell University

   2 Tsinghua University

   3 University of California, Berkeley

   4 Cornell University

   5 University of Illinois, Urbana-Champaign

   **Abstract** The surprising success of cryptocurrencies has led to a surge of inter- est in deploying large scale, highly robust, Byzantine fault tolerant (BFT) protocols for mission-critical applications, such as financial transactions. Although the conventional wisdom is to build atop a (weakly) synchronous protocol such as PBFT (or a variation thereof), such protocols rely critically on network timing assumptions, and only guarantee liveness when the network behaves as expected. We argue these protocols are ill-suited for this deployment scenario.We present an alternative, HoneyBadgerBFT, the first practical asynchronousBFT protocol, which guarantees liveness without mak- ing any timing assumptions. We base our solution on a novel atomic broadcast protocol that achieves optimal asymptotic efficiency. We present an implementation and experimental results to show our system can achieve throughput of tens of thousands of transactions per second, and scales to over a hundred nodes on a wide area net- work. We even conduct BFT experiments over Tor, without needing to tune any parameters. Unlike the alternatives, HoneyBadgerBFT simply does not care about the underlying network.

   **Key words**  HoneyBadgerBFT asynchronous network Atomic Broadcast Tor [↑](#footnote-ref-6)