蜜獾BFT协议

Andrew Miller1) 于霞2） Kyle Croman3)  Elaine Shi3) Dawn Song4)

1)伊利诺伊大学 厄巴纳-尚佩恩分校 厄巴纳-尚佩恩 美国 61801

2)清华大学 北京 100084

3)康奈尔大学 纽约 美国 14853

4)加州大学伯克利分校 加州 美国 04720

摘 要 \* 虚拟货币的惊人成功引起了一股将大规模，高强度的拜占庭容错（BFT）协议部署于关键任务应用的潮流，如财务交易。传统的做法是在顶部建立一个（弱）同步协议，如PBFT协议（实用拜占庭容错）或其变体，但这些协议严重依赖于网络时间假设，只有当网络按照预期运行时才能保证活跃。我们认为这些协议不适合这种部署情景。

我们提出一个替代品，蜜獾BTF协议，第一个保证在不制造任何时间假设的情况下运行的实用异步BFT协议。我们将解决方案建立在一个实现了最佳渐近效率的原子广播协议上。我们提出实现方式和实验结果来证明我们的系统可以实现每秒数万个交易的吞吐量，并可以拓展到广域网上的一百多个节点。我们甚至无需调整任何参数就可以在Tor（洋葱路由器）上进行BFT实验。与其他协议不同，蜜獾BFT协议根本不在乎底层网络。

关键词 \*拜占庭容错协议；原子广播协议；异步BFT协议；最佳渐进效率；吞吐量；

TheHoneyBadgerofBFTProtocols

Andrew Miller1) Yu Xia2) Kyle Croma3) Elaine Shi3) Dawn Song4)

1)University of Illinois, Urbana-Champaign Urbana-Champaign America 61801

2)Tsinghua University Beijing 100084

3)Cornell University NewYork America 14853

4)University of California, Berkeley California America 04720

Abstract \* The surprising success of cryptocurrencies has led to a surge of interest 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 asynchronous BFT protocol, which guarantees liveness without making 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 network.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 \*

# 引言

分布式容错协议是一种有前途的解决方案，可用于解决关键任务的基础设施，如金融交易数据库。传统上，它们的部署规模相对较小，并且通常在敌对攻击不是主要问题的单一管理域。一个典型的例子就是谷歌的容错分布式锁定服务的部署——Chubby，它由五个节点构成，可容忍最多两个崩溃错误。

近年来，随着比特币的惊人成功，被称为“虚拟货币”和“区块链”的分布式系统的新形式出现了。这种虚拟货币系统代表着一种惊人而有效的突破，并且为我们对分布式系统的理解打开了新篇章。

虚拟货币系统挑战了我们对于容错协议的部署环境的传统观念。与经典的“谷歌里的五个Chubby节点”不同，虚拟货币已经解释并激发了对广域网上彼此不信任的节点之间的共识协议的新需求。并且，网络连接会比传统的局域网设置更加难以预测，甚至是与其对立的。这种新的设置既带来了有趣的新挑战，也呼吁我们重新思考容错协议的设计。

**稳定性是第一考虑因素。**虚拟货币显示了一个不同的操作点的需求和可能性，它把稳定性视作高于一切的要素，即使需要牺牲性能。事实上，比特币的分布式系统标准提供了糟糕的性能：一个交易平均需要十分钟才能完成，系统整体的吞吐量只有每秒十个交易。然而，与传统的容错部署方案相比，虚拟货币在高度敌对的环境下迅速发展，可能会出现动机强烈的恶意攻击（如果没有广泛普及）。出于这个原因，许多比特币的热心支持者称之为“金钱蜜獾”。我们注意到对稳定性的需求尝尝与对权利下放的需求密切相关——权力下放通常需要来自广域网的大量不同用户的参与。

**在延迟基础上的吞吐量。**大多数现有的可伸缩容错协议着重于优化在单一管理域控制下的局域网环境中的可伸缩性。由于宽带供应很充足，这些工程往往侧重于减少（虚拟货币的）计算并尽量减少争用（既对于相同物品的竞争的需求）的响应时间。相反，区块链引起了一些金融应用的兴趣，在这些应用中响应时间和争用不是最关键的因素，例如支付和结算网络。事实上，一些金融应用程序故意在提交交易时引入延迟来允许可能的回滚/退款操作。虽然在这些应用程序中延迟不是关键，但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，它们需要以此来满足大量的交易请求。例如，Visa平均处理速度为2000tx/sec，最高峰值为59000tx/sec。

## 1.1我们的贡献

1.1.1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，即使是那些被认为“稳定”的系统，也假定了一些弱同步的变化，大致来说，就是保证消息在一定的间隔Δ之后被传递，但Δ可能是随时间变化的或者对于协议设计者来说未知的。我们认为基于时间假设的协议不适用于分散式的虚拟货币设置，因为在这种设置中，网络链接可能是不可靠的，网络速度变化很快，甚至网络延迟可能被恶意诱导。

首先，当预先的时间假设不成立（例如由于一个恶意的网络攻击），弱同步协议的活性特征可能完全失效。为了证明这一点，我们清晰地构建了一个违反假设的恶意“间歇性同步”网络，使现有的弱同步协议，如PBFT[20]停止（第3节）。

其次，即使在实践中满足弱同步协议假设的情况下，当基础网络不可预测时，弱同步协议的吞吐量也会显著降低。理想情况下，我们希望即使在快速变化的网络条件下，吞吐量也能最大限度地发挥网络的性能。不幸的是，弱异步协议需要严格调整后的超时参数，尤其是在虚拟货币应用程序设置中；当选择的超时值太长或太短时，吞吐量可能会减少。一个具体的例子中，我们表明即使当弱同步假设满足时，这种协议也很难快速的从瞬时网络分区恢复（第三节）。

### 1.1.2实用的异步BFT。

我们提出了蜜獾BFT，第一个在异步设置中提供最佳渐进效率的BFT原子广播协议。因此，我们直接驳斥了认为这些协议是不切实际的普遍看法。

我们对已知最好的原子广播系统进行了效率上的显著改进，这个系统基于Cachin等人的成果，该过程要求每个节点为每个承诺的交易提交O（N2）码，这实际上限制了除了最小网络外其它网络的吞吐量。这种低效有两个根本原因。第一个原因是是双方的冗余工作。然而，一个试图消除冗余的天真尝试损害了交易的公平性，并且允许有针对性的审查攻击。我们发明了一种新的方案使用门限公钥加密防止这些攻击，解决了这个问题。第二个原因是使用了异步公共子集（ACS）组件的一个次优的实例。我们展示了如何通过结合已有的被忽略的技术来搞笑的实例化ACS：使用纠删码的高效可靠的广播，以及参考了多方 计算文献的从ACS到可靠广播的删减。

在虚拟货币的部署场景中，网络宽带是稀缺资源，但计算资源相对充足，蜜獾BFT协议的设计针对这种部署场景进行了优化。这使我们能够利用密码构建模块（尤其是阈值公钥加密），这在主要目标是即使在争用下也尽量最小化响应时间的经典容错数据库设置中被认为是过于昂贵的。

在异步网络中，消息在没有做出其它时间假设的情况下被传递。不同于现有的对于参数调整十分严格的弱同步协议，蜜獾BFT协议不在乎这些。无论网络条件如何波动，蜜獾BFT协议的吞吐量总是能最大限度发挥网络的可用带宽。准确的说，只要消息被传递，蜜獾BFT协议就会发挥作用；而且，它会在消息被传递时迅速发挥作用。

我们正式的证明了我们的蜜獾BFT协议的安全性和可用性，并且通过实验表明了即使在乐观的情况下，它也能比传统的PNFT协议提供更好的吞吐量。

### 1.1.3实行和大规模试验

我们提供了一个完整的蜜獾BFT协议的实现过程，我们将在近期将其作为免费的开源软件发布。1我们用分布在五大洲的超过100个节点在亚马逊AWS上部署并获得了实验结果。为了展示它的多功能性和稳定性，我们还在洋葱匿名中继网络上部署了蜜獾BFT协议，我们没有改变任何参数而实现了吞吐量和延迟结果。

## 建议的部署方案

在许多可以想象到的应用中，我们着重强调了在银行，金融机构和实现完全分散式虚拟货币的两种可能的部署方案。

### 联邦虚拟货币

比特币等分散式虚拟货币的成功激发了银行和金融机构从新的角度审视他们的交易过程和结算基础设施。“联邦虚拟货币”是一个经常被引用的场景【24，25，48】，其中金融机构的一个联合体共同为拜占庭协议做出贡献来允许快速稳定的交易结算。由于这种方式将简化当前用于银行间结算的缓慢而笨重的基础设施，他们的热情十分高涨。因此，几个新的开源项目致力于为这种设置构建一个合适的BFT协议，例如IBM的Open Blockchain 和超级帐本项目【40】。

联邦虚拟货币需要在广域网上部署BFT协议，这可能涉及到数千个共识节点。在这种情况下，注册可以很容易的被控制，这样一套共识节点就是已知的——通常被成为“已许可的”区块链。显然，蜜獾BFT协议实在这种联邦加密货币中使用的自然选择。

### 1.2.2在无权区块链中的适用性。

相比之下，像比特币和以太坊这样的分散式虚拟货币则选择了“无许可的”区块链，在这种区块链中，任何人都可以进行注册，节点可以活跃而频繁的加入和离开。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女巫攻击，并为此在吞吐量和延迟方面付出了巨大的代价，例如，比特币每10分钟提交一次交易，并且即使在当前区块的尺寸最大时吞吐量也限制为7tx/sec。最近的一些研究提出了一个有前景的想法，即利用像比特币这样较慢的外部区块链或涉及基础货币本身的经济“权益证明”假设【32，32，35，37】来引导更快的BFT协议，通过选择一个随机的委员会在每个不同的时期进行BFT协议。这些方法有希望在两方面都达到最优，即在公开注册的分散式网络中的安全性，和与传统BFT协议相匹配的吞吐量和响应时间。在这些情景中，由于随机委员会可能是地理上互不相同的，蜜獾BFT也是一个很自然的选择。

# 2.背景和相关的研究

我们的总体目标是构建一个复制的状态机，其中，客户端生成并提交交易，节点网络接受并处理它们。它从应用程序特定的细节（例如怎样表示状态和计算转换）中抽象出来，只需要构建一个总体上全局一致的，完全有序的，仅附加的交易日志。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可用性保证，这允许了分布式系统在存在网络延迟和某些节点的故障的情况下提供正确的服务。现在已经有大量针对这些协议的研究，它们提供了不同的性能这种来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研究。[[1]](#footnote-1)

## 2.1 稳定的BFT协议

虽然Paxos[36],Raft[46]以及许多其他众所周知的协议都可以容忍崩溃错误，但从PBFT[20]开始的拜占庭容错[[2]](#footnote-2)协议（BFT）甚至可以容忍任意（如恶意的）损坏的节点。许多后续的协议提供了改进的性能，但它们通常是通过在没有错误，客户端没有太大的竞争，网络运行良好的情况下的乐观执行来实现优良性能的提供，否则至少会取得一些进展[2,5,33,39,51]。

一般来说，BFT系统会在延迟和CPU是瓶颈的部署场景[49]中被重新评估，因此最有效的协议应减少回合的次数并最小化昂贵的加密操作。

Clement等人[22]希望改善最差的情况下的性能，他们进行了一系列研究[4,6,10,21]，使系统即使在受到攻击时也能提供服务质量保证——即使这在乐观情况下是以牺牲性能为代价的。然而，虽然这种“稳定BFT”协议稳定的承受了被攻破的节点，但它们仍然依赖于对底层网络的时间假设。我们的研究进一步发展了这种方法，使其即使在完全异步网络中也能保证良好的吞吐量。

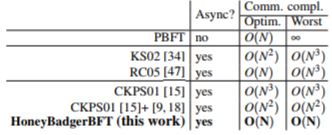
## 2.2 随机协议

确定的异步协议对于大多数任务来说时不可能的[27]。虽然大多数实际的BFT协议都是通过做出时间假设来消除这种不可能性，但随机性（尤其是密码学）提供了另一种方法。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]。

我们的研究与SINTRA[17]关系最为密切，这是一个基于Cachin等人的异步原子广播协议（CKPS01）[15]的系统的实现。该协议包括从原子广播（ABC）到通过子集协议（ACS）的缩减，以及从ACS到多值验证协议（MVBA）的缩减。

我们改进的关键发明是从ABC到ABS的创新删减，它批量提供了更好的效率（通过O(N)因子），同时使用阈值加密来保持审查弹性（见4.4节）。我们也通过精选文献获得了更好的效率。更具体的说，我们通过使用替代的ACS[9]和高效的RBC[18]来会比昂贵的MVBA原始型别，如4.4节所述。

表1总结了蜜獾BFT协议与其他几种原子广播协议的渐进性能。这里’Comm.compl”表示每个被提交的交易的与其通信复杂度（即传送的总字节数）。由于PBFT依赖于弱同步假设，因此在异步网络中可能无法发挥作用。协议KS02[34]和RC05[47]是乐观的，它回落到了基于MVBA的昂贵的恢复模式。像前文提到的Cachin等人的方案（CKPS01）[15]可以改进为使用更有效的ACS结构[9,18]。我们还可以通过我们的创新删减得到另一个O(N)的改进。



表一：原子广播通信的复杂性（每笔交易的预期位数）

最终，King和Saia[30.31]最近通过在稀疏图上的路由通信开发了消息树少于二次的协议。但是，如何将这些成果拓展到异步协议上仍然是一个问题。

# 3.异步和弱同步网络模型之间的差距

几乎所有现代BFT协议都依赖时间假设（例如部分或者弱同步）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，如果它成立的话，将证明这个狭义的观点：

[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难以接受的性能.

在这一节中，我们提出两个驳斥上述前提的反驳首先，我们阐述了异步和弱同步网络模型之间的理论差别。具体而言，我们构建了一个违反PBFT弱同步假设的恶意网络调度器（而且确实使其失效了），但是在这种情况下，所有纯粹的异步协议（如蜜獾BFT协议）都发挥良好。其次，我们观察到：即使满足了它们的假设，当弱同步协议恢复，网络分区的恢复速度很慢，而异步协议只要消息被传递就会立刻发挥作用。

## 3.1时间假设的多种形式

在继续研究之前，我们回顾一下时间假设的各种标准形式。在异步网络中，攻击者可以随意以任何顺序发送消息，但最终必须在正确的节点之间发送每条消息。对异步网络中的节点来说，“实时”时钟没有用处，它们只能根据接收到消息的顺序进行操作。

众所周知的FLP[27]研究结果排除了原子广播和许多其他任务使用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。一个便捷（但非常强大）的网络假设是同步的：一个Δ同步网络保证发送的每个消息在最多在延迟Δ（其中Δ是实时量度）之后被传送。

弱时间假设有几种形式。在Δ未知的模型中，协议不能使用延迟界限作为参数。或者说，在最终的同步模型中，消息延迟限制Δ只能保证在某个（未知的）时刻后保持不变，被称为“全局稳定时间”。这两个模型统称为部分同步[26]。现在的另一个变体是弱同步[26]，其中延迟界限是随时间变化的，但最终不会比时间多项式函数增长更快[20].

就可行性而言，上述内容是等价的——在一个设置中成功的协议就可以被系统的适用于另一个设置。然而就具体性能而言，对弱同步协议的调整意味着主键增加超时参数（例如，通过“指数后退”政策）。正如我们稍后展示的那样，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，如果各方在一定时间间隔内未发现有进展，则采取纠正措施，如选出新的领导者。异步协议不依赖于定时器，无论实际的时钟时间如何，在消息传递时都会取得进展。

### 3.1.1 在异步网络中计数

尽管消息传递的保证与“实时”的概念是分开的，但仍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，这个条件是每个正确节点之间的（r-1）消息必须在（r+1）消息被传送之前送达。

## 3.2 当弱同步失效时

我们现在继续描述为什么当网络条件时敌对的（或不可预知的）时，弱同步BFT协议可能会失效（或性能下降）。这引出了为什么这样的协议不适用于第一节中描述的面向虚拟货币的应用场景。

### 3.2.1 阻碍PBFT的网络调度器

我们使用实用的拜占庭容错（PBFT）协议[20]，经典的基于领导者的BFT协议，作为一个代表性的例子来描述一个恶意网络调度器如何导致一类基于领导者的BFT协议[4,6,10,22,33,50]戛然而止。

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，无论是因为领导者是有缺陷的还是因为网络已经停滞，节点都哦会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可行性。我们构建了一个违反这个假设的恶意调度器，这的确使PBFT取得任何进展，而蜜獾BFT协议（实时上任何异步协议都可以）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，展示清晰地攻击有助于激励我们构建异步协议。

我们的调度程序背后的原理很简单。首先，我们假设某个节点已经崩溃。然后，每当一个正确的节点是领导者时，网络就会延迟，阻止其取得进展，并导致循环顺序的下一个节点成为新的领导者。当崩溃的节点成为新的领导者时，调度器立即修复网络分区，并在正确节点之间非常迅速的传送消息；然而，由于领导节点是崩溃的，仍然无法取得任何进展。

这种攻击违反了弱同步的假设，因为它必须把每个周期的消息延迟的越来越长，因为PBFT会在每次选则领导者失败后扩大其超时间隔。另一方面，它也提供了越来越长的同步时间。然而，由于这些同步时间在不方便的时期出现，所以PBFT无法使用它们。展望未来，蜜獾BFT协议，甚至任何异步协议，都能够趁这些同步时期的机会取得进展。

为了证实我们的分析，我们将这个恶意的调度器作为一个代理，它截获并延迟了了所有视图更改消息给新的领导者，并针对1200线的PBFT的Python实体进行测试。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，并且试图更改从未成功。在线完整版本[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。

### 3.2.2 从网络分区恢复缓慢

即使弱同步假设最终满足，依赖它的协议也可能会很慢的熊瞬间网络分区中恢复。考虑以下情况，它只是上述攻击的有限前缀：一个节点崩溃，网络暂时在2DΔ的持续时间内被分隔。我们的调度程序正是在崩溃的节点成为领导者时精确地修复了网络分区。由于此时的超时间间隔时2D+1Δ，所以在开始选择新的领导者之前，协议必须等待另一个2D+1间隔，尽管在这个间隔内网络是同步的。

### 3.2.3 稳定性和响应性之间的权衡

以上这些我们所观察到的行为并不是PBFT所特有的，而是依赖于超时来处理崩溃的协议的基本内容。不管协议的变体如何，从业者必须用一些折衷手段调整其超时策略。在一个极端情况（最终同步）下，从业者会对网络延迟Δ做出具体的估计。如果估计太低，那么系统可能会根本没有进展；如果太高，就无法完全利用可用宽带。在另一个极端情况（弱同步）下，从业者要避免指定具体的延迟，但是必须选择影响系统跟踪变化条件的速度的“增长值”。异步协议避免了调整这些参数的必要。

# 4. 蜜獾BFT协议

在本节中，我们将介绍蜜獾BFT协议，这是第一个实现最有渐进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（P0到PN-1）。节点接受交易作为输入，其目标是就这些交易的排序达成共识。我们的模型特别匹配“已授权区块链”的部署场景，它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，例如交易如何被解释（以防止重复的攻击，例如，应用程序可能定义一个是无赖包含签名和序列号）。对于我们的目的来说，交易只是唯一的字符串。在实际操作中，客户端会生成交易并将它们发送到所有的节点，并且在收集到大多数节点的签名后认为它们承认了。为了简化我们的表述，我们没有明确的模拟客户端，而是假设交易是被恶意选择的，并作为输入提供给节点。

我们的系统模型做出以下假设：

假设1（纯粹的异步网络），我们假设每对节点之间通过一个可靠的认证过的点对点通道34连接，不会丢失信息。2传递的时间表是完全被恶意决定的，但是[[3]](#footnote-3)在正确节点之间发送的每个信息都必须最终被传达。我们希望根据异步回合的数量来描述协议间的运行时间（如第2节所述）。由于网络可以用任意延迟来排列消息，因此我们也假定节点具有无限的缓冲时间并且能够处理他们接收到的所有消息。

假设2（静态拜占庭故障）攻击者可以完全控制多达f个故障节点，其中f是一个协议参数。注意，3f+1≤N（我们的协议所实现的）是这个设置中广播协议的下限。

假设3（受信任的设置）为了便于表达，我们假设节点可能会在初始协议待定的设置阶段与可信的经销商进行交互，我们将利用这一点来建立公钥和秘密共[[4]](#endnote-2)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]）。我们已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，这些假设只需要在安装过程中保持稳定。

定理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以高概率（按安全参数λ的函数1-neg（λ））保持，尽管存在任意的恶性攻击：

（协议）如果任何正确的节点输出交易tx，则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,…txj,>,另一个输出了< tx0’，tx1’,…txj’’,>,那么当i≤min(j,j’)时，txi=txi’.

（审查弹性）如果交易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，它可以防止攻击者阻止单个交易被提交。这个属性已经有了其它名称，例如Cachin等人的“公平性”。[15],但我们倾向于这个更具描述性的名称。

### 4.1.1性能指标

我们主要对分析院子广播协议的效率和事物延迟感兴趣。

定义（效率）假设每个诚实节点对缓冲区的输入充分满足Ω(poly(N, λ))。那么效率就是在所有已承诺的交易中分摊的每个节点的预期通信成本。

由于每个节点必须输出每项交易，所以O(1)效率(我们的协议所达到的)是渐近最优的。上述对于效率的定义假设了网络负载不足，这反映了我们的主要目标：在充分利用网络可用宽带的同时保持高吞吐量。由于我们达到了批量实现良好的吞吐量，当交易很少到达时，我们的系统可以在低需求时期使每项已提交的交易使用更多的宽带。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），那么不具有这种特点的更强的定义才是适用的。

实际上，网络连接的容量是有限的，如果提交的交易量超过网络处理能力，那么确认时间的保证一般是不能达成的。因此，我们定义的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[[5]](#endnote-3)。

定义(交易延迟) 假设攻击者将交易tx作为输入传递给N-f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交易总数与已经提交的交易数之间的差值。那么交易延迟是在每个正确节点输出tx之前作为T的函数的异步回合的期望数量。

# 4.2 概述和思路

在蜜獾BFT中，节点接受交易作为输入并将它们储存在它们的（无界）缓冲区中。该协议进入一个时期，在每个时期之后，新的一批交易被追加到被承认的日志中。在每个时期开始时，节点在缓冲区中选择一个交易子集（通过我们将定义的方法），并将它们作为输入提供给一个认同协议的实例。在认同协议结束时，这个时期的最后一组交易将被选择。

在这种高层次上，我们的方法类似于现有的异步原子广播协议，特别是Cachin等人的研究。[15],大规模交易处理系统（SINTRA）的基础。和我们一样，Cachin的协议是以异步通用子集（ACS）原语为中心的。大体上说，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-2f个正确的节点的输入值。从这个原语构建原子广播是微不足道的——每个节点只是从其队列的前面提出一个交易子集，并输出已认同的向量的元素的集合。但是，这里有两个重要的挑战。

挑战1：实现审查复原能力。ACS的成本直接取决于每个节点提出的交易组的大小。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的是绝大多数不相交的交易集合来提高整体效率，从而以相同的成本在同一批中提交更多确定的交易。因此，我们的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），二十提出随机选择的样本，每项交易平均只由一个节点提出。

然而，如果天真的去实施，这种优化会降低审查弹性，因为ACS基元允许攻击者选择哪一个节点所提交的交易被包括在内。攻击者可以选择性的审查一个交易，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项交易由哪个节点提出，直到协议达成。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]，但它们的实际性能还未可知。就我们所知，实施ACS的唯一其他研究就是Cachin和Portiz的研究，它们在广域网上可以达到0.4tx/sec的吞吐量。因此，一个有趣的问题是协议是否可以在实践中达到高吞吐量。

在本文中，我们展示了尽心挑选的一组子组件，我们可以高效的实施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ACS的渐近成本（每个节点），使其从O(N2)（如Cachin等人的研究[15,17]）改变为O（1）。由于我们挑选的组件在之前（据我们所知）并没有一同被呈现，因此我们在4.4接中提供了对整个构建的独立描述。

### 4.2.1 模块化协议组成

我们现在准备正式提出我们的构建。在此之前，我们对我们展示的风格进行介绍。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它（子）协议的多个实例。外部协议可以提供输入并从子协议接收输出。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）协议实例以确保属于一个实例的消息不能在另一个实例中重播是非常重要的。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）相关联， 在这个（子）协议中用这个标识符发送或接收的消息，并相应地路由消息。 我们在协议描述中抑制这些消息标签以便于阅读。 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。

我们暗中假设，各方之间的异步通信是通过已认证的异步通道。 实际上，这样的渠道可以使用TLS套接字实例化，就像我们在第5节中讨论的那样。

为了区分协议中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构建蜜獾BFT异步通用子集

蜜獾协议算法（用于节点Pi）

设B =Ω（λN2logN）为批量大小参数。

设PK是从TPKE.Setup（由经销商执行）接收到的公钥，并且SKi是Pi的密钥。

设buf：= [ ]为输入交易的FIFO队列。

进行编号为r的连续时期：

//步骤1：随机选择和加密

•设提取值位为从缓冲区的前B个元素中随机选择的[B / N]项交易

•加密x：= TPKE.Enc（PK，提取值）

//步骤2：密文协议

•将x作为输入传递给ACS [r] //参见图4

•从ACS [r]接收{v j}j∈S，其中S⊂[1..N]

//步骤3：解密

•对于每个j∈S：使e j：= TPKE. DecShare（SKi，v j）

广播DEC（r，j，i，e j）

等待接收形式为DEC（r，j，k，e j，k）的至少f + 1个消息，

解码y j：= TPKE.Dec（PK，{（k，e j，k）}）

•使blockr：= sorted（∪j∈S{y j}），使得blockr以规范的顺序排列（例如按照字典顺序）

•设置buf：= buf -blockr

**构件模块：ACS。** 我们的主要构建模块是一个叫做异步公共子集（ACS）的原语。 构建ACS的理论可行性已在多篇论文中得到了证明[9,15]。本节将介绍ACS的形式化定义，并将其用作构建蜜獾BFT协议的黑盒。 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，我们可以有效地实例化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 f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 - f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。** 一个阈值加密方案TPKE是一个密码原语，它允许任何一方将消息加密成一个主公钥，这样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一个密文的解密份额， 明文可以被恢复; 在至少有一个正确的节点显示出其解密份额前，攻击者对明文没有任何了解。 阈值方案提供以下界面：

TPKE.Setup（1^λ）→PK，{SKi}生成一个公共密钥PK以及每个SKi的密钥

•TPKE.Enc（PK，m）→C加密消息m

•TPKE.DecShare（SKi，C）→σi产生解密的第i份（如果C格式错误，则产生⊥）

•TPKE.Dec（PK，C，{i，σi}）→m合并来自至少f+1方的一组解密份额{i，σi}以获得明文m（或者，如果C包含无效份额，那么无效份额会被明确）。

在我们的具体实例中，我们使用Baek和Zheng的阈值加密方案[7]。 这个方案也是稳定的（根据我们的协议要求），这意味着即使对于一个攻击者生成的密文C，最多也只发现一个明文（除了⊥）。注意，我们假设TPKE.Dec有效地识别了无效的解密份额输入。 最后，该方案满足明显的正确性属性，以及IND-CPA游戏的阈值版[[6]](#footnote-4)本。4

### 4.3.1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中进行定义。

图一：蜜獾BFT协议

如前所述，这个协议是以ACS实例为中心的。为了获得可扩展的效率，我们选择一个批量处理策略。我们让B作为一个批量处理大小，并且将在每个时期提交Ω（B）项交易。 每个节点从其队列中提取B / N项交易。 为了确保大多数节点提取不同的事务，我们随机选择每个队列的前B项交易。

正如我们将在4.4节看到的那样，我们的ACS实施具有的总通信成本为O（N2 | v | +λN3logN），其中|v| 限制了任何节点输入的大小。 因此，我们选择一个批量大小为B =Ω（λN2logN），这样每个节点的贡献（B / N）吸收这种附加开销。

为了防止攻击者影响结果，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。 然后每个节点将密码作为输入传递给ACS子例程。 因此ACS的输出是密文的向量。 一旦ACS完成，密文就被解密。 这保证了在攻击者了解每个节点提交的具体内容之前，交易集合就完全确定了。 这也保证了攻击者不能有选择地阻止交易被提交，除非它排在足够的正确节点的队列之前。

## 4.4高效实例化ACS

Cachin等人提出了一个我们称之为CKPS01的协议（暗中）将ACS简化为多值验证的拜占庭协议（MVBA）[15]。粗略地说，MVBA允许节点提出满足描述的值，最终选择其中的一个。它的删减很简单：用于验证的描述表示输出必须是来自至少N-f方的签名输入的向量。 不幸的是，MVBA原始协议成为了一个瓶颈，因为我们所知道的唯一的结构会导致O（N3 |v|）的开销。

我们通过使用完全避开MVBA的ACS的替代实例来避免这个瓶颈。 我们使用的实例是由Ben-Or等人提出的， [9]并且在我们看来，它有些被忽视。事实上，它早于CKPS01 [15]，并且最初是为了一个非常不相关的目的而开发的（作为实现高效的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结构，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，每个节点Pi使用RBC将其建议值传播给其他节点，然后由ABA决定一位向量，它将显示指示哪些RBC已成功完成。

在更详细地解释Ben-Or协议之前，我们简要解释RBC和ABA结构。

### 4.4.1通信最优的可靠广播

一个异步的可靠广播频道满足以下属性：

（一致性）如果任何两个正确的节点提供v和v’，则v = v’。

（总体性）如果任何正确的节点传递了v，则所有正确的节点传递v

（有效性）如果传送的节点是正确的并且输入了v，那么所有正确的节点传递v。

尽管Bracha [13]的经典可靠广播协议广播大小为| v |的消息需要O（N2 |v|）比特，Cachin和Tessaro [18]观察到纠删码可以将这个代价降低到仅仅O（N |v| +λN2logN），即使在最坏的情况下。 这对于大型消息（即当| v |>>λNlogN）来说是一个重大的改进，（回顾4.3节）引导了了我们对批量大小的选择。 在这里使用纠删码最多会引起一个小的恒定开销因子，相当于

如果发送的节点是正确的，那么总运行时间是三（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正确节点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。

RBC算法（用于Pi部分，通过传送者PSender）

输入（v）（如果Pi = PSender）：

设{sj}j∈[N]为应用于v的一个（N-2f，N）纠删码方案的模块。

设h为在{sj}上计算出的Merkle树根

将VAL（h，bj，sj）发送给每个部分Pj，其中bj是第j个Merkle树分支

在收到来自PSender的VAL（h，bi，si）后，

广播ECHO（h，bi，si）

从Pj接收到ECHO（h，bi，si）后，

确认bj是根h和叶sj的有效Merkle分支，否则丢弃

在接收到来自N-f个不同部分的有效ECHO（h，·，·）消息后，

- 从接收到的任意N-2f个叶中插入{s’j}

- 重新计算Merkle根h’，如果h’ ≠ h则中止

- 如果READY（h）尚未发送，则广播READY（h）

在接收到f + 1条匹配的READY（h）消息时，如果READY尚未发送，则广播READY（h）

接收到2 f +1条匹配的READY（h）消息后，等待N-2f个ECHO消息，然后解码v

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]

### 4.4.2二元协议

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：

（一致性）如果任何正确的节点输出位b，则每个正确的节点输出b。

•（终止性）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。

有效性意味着一致性：如果所有正确的节点都接收到相同的输入值b，则b一定是决定的值。另一方面，如果在任何一点两个节点接收到不同的输入，那么攻击者甚至可能在剩余的节点接收输入之前强制决定一个值。

我们用Moustefaoui等人的一个协议来实现这个原语。 [43]，它基于一个加密的普通硬币。 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），这主要是因为在这个普通硬币中使用了阈值密码。

### 4.4.3就建议值的一个子集达成一致

把上述部分整合起来，我们使用Ben-Or等人的一个协议 [9]来就包含至少N – f个节点的整个协议的一组值达成一致。

在较高水平上，该协议分两个主要阶段进行。 在第一阶段，每个节点Pi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约定一位向量{bj}j∈[1..N]，其中bj= 1表示Pj的建议值被包括在最终集合中。

其实上面的简单描述隐含了一个微妙的挑战，Ben-Or提供了一个聪明的解决方案。

如果试图天真的实现上述方案，那么每个节点都要等待前（N-f）个广播完成，然后为与之对应的二进制协议实例提出1，为所有其他节点提出0。 但是，正确的节点可能会以不同的顺序观测广播。 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提出1，则得到的位向量可能是空的。

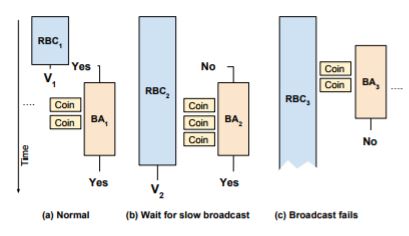
为了避免这个问题，节点放弃提出0，直到他们确定最终的向量将具有至少N-f个比特的集。为了给这个协议的流程提供一些思路，我们在图3中叙述了几种可能的情况。 来自Ben-Or等人[9]的算法如图4所示。预期运行时间为O（logN），因为它必须等待所有二进制协议实例完成。 4当用上述可靠的广播和二进制协议结构来实施时，假设| v | 是任何节点输入的最大尺寸，总的通信复杂度为O（N2|v|+λN3log[[7]](#footnote-5)N）。

图3（ACS执行的例子）

我们的协议的每次执行都包括运行N个可靠广播（RBC）的并发实例，以及N个拜占庭协议（BA），后者又使用预期的常数个硬币。 从节点0的角度，我们举例说明了这些实例如何发挥的几个可能的例子。

（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向BA1提供输入“是”，使其输出“是”。

（b）RBC2花费太长时间完成，而节点0已经收到（N - f）个“是”输出，因此它对BA2投票为“否”。 但是，其他节点已经看到RBC2成功完成，所以BA2的结果为“是”，节点0必须等待V2。

（c）在RBC3完成之前，BA3以“否”结束。[[8]](#footnote-6)

ACS算法（对于Pi部分）

设{RBCi} N表示可靠的广播协议的N个实例，其中Pi是RBCi的发送者。 设{BAi} N表示二进制拜占庭协议协议的N个实例。

接收到输入vi后，输入vi到RBCi//参见图2

在从RBCj传递vj时，如果尚未向BAj提供输入，则将输入1提供给BAj。 查看在线完整版[42]

从BA的至少N-f个实例传送值1时，向尚未提供输入的BA的每个实例提供输入0。

一旦BA的所有实例都已经完成，设C [1..N]为每个发送1的BA的索引。等待每个RBCj的输出vj使得j∈C。最后输出∪j∈Cvj。

图4：通用子集一致协议（来自Ben-Or等[9]）

## 4.5分析

首先我们观察到协议和总秩序属性紧跟着ACS的定义和TPKE方案的稳定性。

定理1.（协议和总秩序）。除去可忽略的概率，蜜獾BFT协议满足一致性和总秩序属性。

证明.这两个属性与高级协议，ACS，TPKE的属性极其相近。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的稳定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秩序。

定理2.（复杂性） 假设批量大小为B =Ω（λN2logN），则每个蜜獾BFT协议时期的运行时间为O（logN），总预期通信复杂度为O（B）。

证明. 第4.4节解释了ACS的成本和运行时间。 阈值解密的N个实例需要额外的一轮和O（λN2）的附加成本，这不影响整体渐近成本。

蜜獾BFT协议可能在一个时期内承诺B项交易。 但是，实际的数量可能会少于这个数字，因为一些正确的节点可能会提出重复的的交易集合，另一些节点可能会响应得太晚，并且被破坏的节点可能会提出一个空集合。幸运的是，我们证明了（在线完整版本[42]） 假设每个正确节点的队列已满，则B/4为一个时期中提交的预期交易数的下限。5

定理3.（效率）。 假设每个正确节点的队列至少包含B项不同的交易，那么在一个时期中提交的交易的预期数量至少是，从而产生恒定的效率。

最后，我们证明（在线完整版本[42]），攻击者不能显着延迟任何交易的提交。

定理4.（审查复原） 假设攻击者把交易tx作为输入传递给N - f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交易总数与已经提交的交易数之间的差值。 那么tx将在O（T / B +λ）时期内被提交，除了可忽略的概率。

# 5.实施和评估

在本节中，我们使用蜜獾BFT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在所有节点都表现诚实的乐观情况下。

首先，我们演示了蜜獾BFT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，蜜獾BFT也可以达到每秒数千次交易的峰值吞吐量。 此外，通过与代表性部分同步协议PBFT的比较，蜜獾BFT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。

### 5.0.1实施细节

我们在Python中开发了蜜獾BFT的原型实例，使用gevent库来执行并发任务。

对于确定性纠删码，我们使用实现Reed-Solomon编码的zfec库[52]。为了实例化普通硬币原语，我们实现了Boldyreva的基于配对的门限签名方案[11]。对于交易的阈值加密，我们使用Baek和Zheng的方案[7]来加密一个256位的短暂密钥，然后在CBC模式下对AES-256进行加密。我们使用 PBC图书馆的Charm [3] Python封套[38]。对于阈值签名，我们使用提供的MNT224曲线，得到只有65字节的签名（和签名份额），并试探性地提供112位安全性7。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，它启发式地提供了80位的安全性[45]。8

在我们的EC2实验中，我们使用普通的（未经认证的）TCP套接字。 在真正的部署中，我们将使用TLS进行客户端和服务器身份验证，这为长寿命会话增加了微不足道的开销。 同样，在我们的Tor实验中，每个套接字只有一个端点被认证（通过“隐藏服务”地址）。

我们的理论模型假定节点具有无限的缓冲区。在实践中，只要内存消耗达到水印，就可以动态地将更多的资源添加到节点（例如，只要满了75％），尽管我们的原型实现还没有包括这个特征。 如果未能提供足够的缓冲区，则会计入失败预算f。[[9]](#footnote-7)

## 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，我们假定每个mT= 250字节的恒定交易大小，其将承认ECDSA签名，两个公共密钥以及应用负载（即，典型比特币交易的大小）。我们的实验使用参数N = 4f，9各方提出一批B/N交易。为了模拟最坏的情况，节点从大小相同的队列B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。

### 5.1.1带宽和故障结果

每个节点消耗的总体带宽包括一个固定的附加开销以及一个交易相关的开销。 对于我们考虑的所有参数值，加法开销由ABA阶段中的阈值密码和随后的解密阶段产生的O（λN2）产生。 ABA阶段涉及每个节点传送4N2签名份额的期望。 只有RBC阶段产生与交易相关的开销，等于纠删码扩展因子r=。 由于ECHO消息中包含的Merkle树分支，RBC阶段也对开销贡献了N2logN哈希值。 总的通信成本（每个节点）估计为：

Mall = r(BmT +NmE) +N2 ((1+logN)mH+mD+4mS)

其中mE和mD分别是TPKE方案中的密文和解密份额的大小，mS是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，使得成本的交易相关部分占主导地位。 如图5所示，当N = 128，当批量大小多达1024个交易，交易无关的带宽仍占支配地位。 然而，当批量大小达到16384时，交易相关部分开始占主导地位 ——主要是由节点传输纠删码的RBC.ECHO阶段产生的。

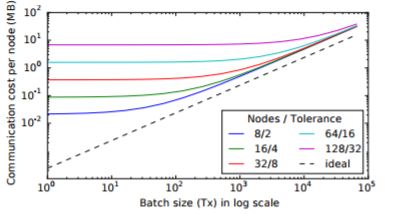


图5：针对不同的批量大小以兆字节（每个节点）为单位的估计通信成本。 对于小批量，固定成本随O（N2logN）增长。在饱和时，开销因子接近<3。

## 5.2 Amazon EC上的实验

为了看看我们的设计有多实用，我们在Amazon EC2服务上部署了我们的协议，并全面测试了它的性能。我们在32个，40个，48个，56个，64个和104个Amazon EC2 t2.medium实例上运行了HoneyBagderBFT， 跨越五大洲。 在我们的实验中，我们改变了批量大小，使得每个节点提出了256,512,1024,2048,4096,8192,16384,32768,65536或131072项交易。

### 5.2.1吞吐量

吞吐量被定义为每单位时间提交的交易数量。 在我们的实验中，如果没有另外指定，我们使用“每秒确认的交易次数”作为度量单位。图6显示了吞吐量与所有N方提出的交易总数之间的关系。 容错参数设置为f = N / 4。

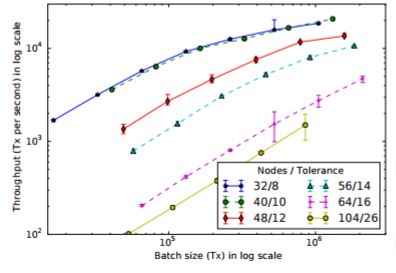


图6：吞吐量（每秒交易次数）与建议交易次数。 误差线表示95％的置信区间。

从图6我们可以看到，对于每个设置，吞吐量随着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。 尽管网络中消耗的总带宽随着每个附加节点的增加（线性）增加，但附加节点也提供额外的带宽容量。

### 5.2.2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请求的时间与第（N-f）个节点完成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）个节点意味着为诚实方达成共识。

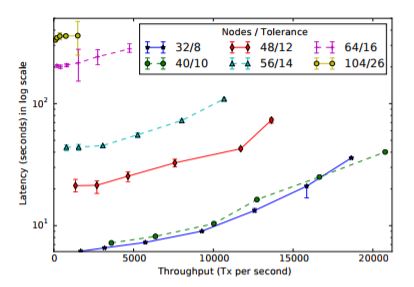
图7显示了N和f = N / 4的不同选择时延迟和吞吐量之间的关系。 正斜率表明我们的实验尚未完全充分利用可用带宽，即使批量较大，我们也能获得更好的吞吐量。 图7还显示，随着节点数量的增加，延迟增加，主要来自协议的ABA阶段。 实际上，在N = 104的情况下，对于我们尝试的批量处理范围，我们的系统是受CPU限制的，而不是带宽限制的，因为我们的实例是单线程的并且必须验证O（N2）阈值签名。 无论如何，我们最大的104个节点的实验在6分钟内完成了。

图7：在广域网上进行实验的延迟与吞吐量的关系。 误差线表示95％的置信区间。

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，但是承诺一个批次所消耗的最小带宽（以及因此等待时间）会随着O（N2logN）而增加。 这个约束意味着可扩展性的限制，它取决于带宽和用户的延迟容忍的成本。

### 5.2.3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实例。 [24]在8,16,32和64个节点上运行，均匀分布在亚马逊AWS地区之间。调节批量大小以使网络的可用带宽饱和。

从根本上说，虽然PBFT和我们的协议总体上具有相同的渐近通信复杂度，但是我们的协议在网络链路中均匀分配这个负载，而PBFT则卡在在领导者的可用带宽上。 因此，PBFT的可达到的吞吐量随着节点的数量而减少，而HoneyBadgerBFT的大致保持不变。

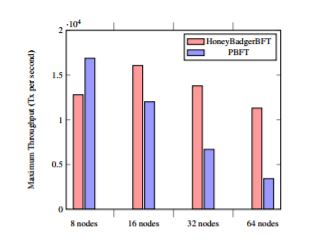


图8：与PBFT在EC2s上比较

请注意，这个实验仅反映乐观的情况，没有故障或网络中断。 即使对于小型网络，蜜獾 BFT协议在恶意条件下也能提供明显更好的稳定性，如第3节所述。特别是，PBFT在恶意的异步调度器下将实现零吞吐量，而蜜獾BFT协议将以常规速率完成这个时期。

## 5.3 Tor上的实验

为了演示蜜獾BFT协议的稳定性，我们运行了一个在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，Tor增加了显着的延迟。 无论如何，我们显示我们可以运行蜜獾BFT协议而不调整任何参数。 隐藏在Tor背后的蜜獾BFT协议节点会提供更好的稳定性。 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。

### 5.3.1Tor简要背景

Tor网络由大约6500个中继组成，这些中继被列在公共目录服务中.Tor使得“隐藏服务”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，每个到隐藏服务的连接通过5个随机选择的中继来路由数据。 Tor为中继节点公布其容量和利用率提供了一种手段，这些自我汇报的指标由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。

### 5.3.2Tor实验设置

我们设计了实验设置，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个蜜獾BFT节点，同时能够真实地反映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，每个蜜獾BFT节点都有一个隐藏的服务）。 由于每个蜜獾BFT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。 总而言之，所有的成对覆盖链路都通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获得的性能代表了Tor上真正的蜜獾BFT部署（尽管所有仿真节点都运行在单个主机上）。

由于Tor为许多用户提供关键的公共服务，因此要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。

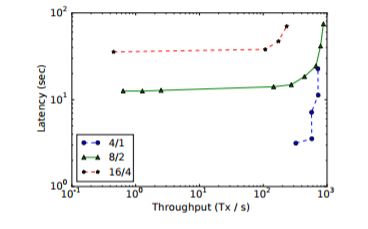
图9显示了延迟如何随吞吐量而变化。 与我们的节点有足够带宽的EC2实验相反，Tor电路受限于电路中最慢的链路。 我们实现了每秒在Tor上超过800次交易的最大吞吐量。

图9：在Tor上运行蜜獾BFT协议的实验的延迟与吞吐量。

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，单个消息可以延迟316.18秒，延迟方差超过2208，而平均延迟只有12秒。我们强调我们的协议不需要像传统的最终同步协议一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，我们证明了蜜獾BFT可以成为用于虚拟货币初期启发的容错交易处理系统部署的合适组件。更普遍的地说，我们相信我们的研究证实了构建可靠并可用于交易处理的基于异步协议的系统的承诺。

## 6.1致谢

我们感谢Jay Lorch，Jonathan Katz和EminGünSirer提供了有益的建议，特别是Dominic Williams为我们提供了一些很好的讨论，这些启发我们解决了这个问题。本文的研究部分由NSF拨款CNS-1314857，CNS-1453634，CNS-1518765，CNS-1514261和CNS- 1518899，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。 这项研究部分完成，而一部分作者访问加州大学伯克利分校的学生，一部分作者访问了西蒙斯计算理论研究所，在西蒙斯基金会的支持下，以及通过NSF授予CNS-1523467。

# 7.参考文献

[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˘TPKC 2003 ˇ , 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 agreementusing 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.

[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(n 3/2 ) bits. In Distributed Computing, pages 464–478. Springer, 2009.

[31] V. King and J. Saia. Breaking the O(n 2 ) 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. https://github.com/amiller/HoneyBadgerBFT [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)
3. 2 可靠的渠道可以通过重新传输而在不可靠的渠道上被模拟，这样会牺牲一些效率。 [↑](#footnote-ref-3)
4. [↑](#endnote-ref-2)
5. [↑](#endnote-ref-3)
6. BAEK和郑的阈值方案也满足（阈值等价的）更强的IND-CCA游戏，但这不是我们的协议所需要的。 [↑](#footnote-ref-4)
7. 实际的界限是（1-e1/3）B> B / 4，但是为了可读性我们使用宽松界限B / 4。 [↑](#footnote-ref-5)
8. 通过并行运行多个实例，期望的运行时间可以减少到O（1）（c.f. [8]），尽管这是以吞吐量为代价的。 [↑](#footnote-ref-6)
9. 早期的报告估计了MNT224曲线的112位安全性[45]。 然而，最近计算离散对数的改进意味着需要更大的参数[28,29]。

   8 对于我们的参数，我们可以证明80位的安全级别相对较弱，因为在每个批次提交之后，明文就会显示出来，所以保密性需求是短暂的。 为了抵御预计算攻击，公共参数和密钥应该定期重新生成。

   9 设置中N = 4 f不是最大容错，但是用f分开N时很方便。 [↑](#footnote-ref-7)