阅读前请了解

Kaspa 新共识算法《DAG KNIGHT》缺乏简易的中文译本，对于许多想要研究 Kaspa 相关技术的国人造成了不小麻烦，特尝试翻译。

当前版本包含原论文 1-6 部分。

翻译工作借助 Google、ChatGPT 等工具，根据中文语境调整得来，可能与原文有一定出入。

受限于本人英语水平以及相关专业知识储备，翻译、解释、语境存在大量歧义、错误，相关格式、排版也不能让人满意，请谅解。

欢迎提出任意形式的建议、批评、指正。

阅读说明

翻译文章分为两栏：左边是原文，右边是对应的翻译以及个人注释

其中注释部分的颜色有如下含义：

绿色部分表示想法来源有依据，可信度较高。

蓝色部分表示对原文的一些分析和补充，可信度中等。

红色部分表示对于文章晦涩部分没有充分理解并给出通俗的翻译或者找到明确的、广知的资料，抑或是单纯吐槽。

反馈

如果您在阅读过程中发现了任何错误，抑或是想要进行学术交流，您可以通过以下方式联系我：

翻译论文极其耗费精力和时间，如果您觉得本翻译对您有帮助，请考虑捐赠任意数量的Kas。

kaspa:qqqn774xzqkkuv4gs07dj5m9mesgt286nnmn9ys52pc2jf2xraft70mawq58p

The DAG KNIGHT Protocol: A Parameterless Generalization of Nakamoto Consensus

DAG KNIGHT 协议：中本聪共识的无参数推广

ABSTRACT

In 2008 Satoshi wrote the first permissionless consensus protocol, known as Nakamoto Consensus (NC), and implemented in Bitcoin.

A large body of research was dedicated since to modify and extend NC, in various aspects: speed, throughput, energy consumption, computation model, and more.

One line of work focused on alleviating the security-speed tradeoff which NC suffers from by generalizing Satoshi's blockchain into a directed acyclic graph of blocks, a block DAG.

Indeed, the block creation rate in Bitcoin must be suppressed in order to ensure that the block interval is much smaller than the worst case latency in the network.

In contrast, the block DAG paradigm allows for arbitrarily high block creation rate and block sizes, as long as the capacity of nodes and of the network backbone are not exceeded.

摘要

2008 年，中本聪编写了第一个无需许可的共识协议，称为 Nakamoto Consensus(简称 NC)，并将其实现为比特币

自那以后，大量研究致力于在速度、吞吐量、能耗、计算模型等方面改进和拓展 NC 协议。

其中一项工作的重点是把中本聪的区块链用块的有向无环图来表示，即 blockDAG，这样做可以一定程度上解决由于 NC 协议所带来的安全速度权衡问题。

实际应用时，为了保证区块间隔远小于最坏情况下网络的延迟，不得不约束比特币的区块创建速度

区块链系统最首要的性能瓶颈是区块数据的广播延迟造成的，本质上受限于互联网的带宽和通讯延迟，在出下一个区块之前，都需要保证前一个区块在全网有一定的同步率，从而约束了每个区块不能太大，出块频率也不能太高。

相比之下，blockDAG 模式允许任意高的块创建速度和块大小，只要不超过节点和网络主干的容量即可。

Still, these protocols, as well as other permissionless protocols, assume an a priori bound on the worst case latency, and hardcode a corresponding parameter in the protocol. Confirmation times then depend on this worst case bound, even when the network is healthy and messages propagate very fast.

可惜尽管如此，为了应对最坏情况，这些协议以及其他无需许可的协议都预先设置了限制，并在协议中硬编码了相应的参数。即使在网络良好而且消息传输极快的情况下，区块确认时间仍然取决于预想最坏情况的上限。

传统 DAG 无法做到根据网络情况动态调整相关参数，不能从根本上解决问题。

In this work we set out to alleviate this constraint, and create the first permissionless protocol which contains no a priori in-protocol bound over latency.

在这项工作中，我们致力于改进这种限制，并创建第一个没有预先进行延迟限制的无许可协议。

无许可协议可简单理解为公链协议

KNIGHT is thus responsive to network conditions, while tolerating a corruption of up to 50% of the computational power (hashrate) in the network.

因此，KNIGHT 会对网络条件做出响应，同时承受网络中高达 50%的计算能力（哈希率）的攻击行为。

To circumvent an impossibility result by Pass and Shi, we require that the client specifies locally an upper bound over the maximum adversarial recent latency in the network.

为了解决 Pass 和 Shi 的论文中提出的不可能结果，我们要求客户端在本地指定网络中最大敌对最近延迟的上限。

在这句话中，Pass 和 Shi 是指两位计算机科学家，分别是 Rafael Pass 和 Elaine Shi 提出了的一个不可能性结果。

具体而言，Pass 和 Shi 在他们的论文中证明，对于任何异步、没有固定时间界限的网络，都不存在一个完美的、不受拜占庭故障影响的共识协议。

KNIGHT is an evolution of the PHANTOM paradigm, which is a parameterized generalization of NC

KNIGHT 是 PHANTOM 范式的进化版本，PHANTOM 是 NC 的参数化概括。

1 INTRODUCTION

The first permissionless consensus protocol, Nakamoto Consensus(NC), was created in 2008 by Bitcoin's originator Satoshi Nakamoto.

Permissionless is defined as an environment where the set of participants is not a priori known and fixed.

Since its introduction, the research community offered many variants that improve upon NC in terms of speed, throughput, energy consumption, computation model, and more.

One line of work focused on alleviating the speed-security tradeoff, by generalizing Satoshi's blockchain into a directed acyclic graph of blocks – a block DAG.

Whereas in NC each block references a single predecessor, and a single chain within the resulting tree is extended, in DAG based constructions blocks reference multiple predecessors.

Blocks are thus created much more frequently than Bitcoin's 10 minutes interval, typically multiple blocks per one unit of network delay.

This asynchronous operation mode opens up the possibility of conflicts across blocks created in parallel.

The heart of the consensus protocol is its conflict resolution rule, which is written in the form of a DAG ordering algorithm—each nodes runs locally a procedure that takes as input the blockDAG visible to it and returns a linear ordering over its blocks, and by implication over its transactions.

1 简介

比特币的鼻祖中本聪于 2008 年创建了第一个无需许可的共识协议——Nakamoto Consensus (简称 NC)。

无许可是这样定义的：

a) 参与者事先未知

b) 没有固定的参与环境。

去中心化

自 NC 算法推出以来，研究界做出了许多努力，试图在速度、吞吐量、能耗、计算模型等方面改进 NC 算法。

上述研究中，有一项工作侧重于通过将中本聪的区块链概括为有向无环块图 blockDAG，希望借此来解决速度与安全的权衡问题。

相比于基于链式结构的 NC，DAG（有向无环图）的构建方式有很大不同：每个区块会引用多个前置区块，从而扩展出多条链。

因此，相比起比特币的 10 分钟才创建一个区块，blockDAG 无疑要快得多，通常情况下，每单位网络延迟能创建多个区块。

这种异步操作模式提高了并行创建的块之间发生冲突的几率。

冲突解决规则是共识协议的核心，它以 DAG 排序算法的形式给出——每个节点在本地运行一个过程，该过程以其可见的块 DAG 为输入，并返回其块的线性顺序，其中就蕴含了交易的顺序。

拓扑排序：将一个有向无环图进行排序进而得到一个有序的线性序列。

上面的内容是对DAG进行拓扑排序，返回的结果便是随时间线性的交易块数据

This ordering ensures and recovers consistency: The first of any set of conflicting transactions is accepted, and the rest are ignored and skipped over.

这种排序能确保和维持一致性：对于任何一组冲突的交易，只有第一个交易会被接受，其余的交易会被忽略并跳过。

As any consensus protocol, this procedure must satisfy the property that all nodes eventually agree on the ordering.

和其他任何共识协议一样，这个过程必须满足一个条件：所有节点最后都同意该排序。

KNIGHT is a parameterless DAG based consensus—the protocol assumes no upper bound on the network's latency.

KNIGHT 是一种基于 DAG 无参数共识——该协议假定网络延迟没有上限。

In other words, the ordering procedure of KNIGHT does not take as input parameters representing the network's assumed latency.

换句话说，在 KNIGHT 的排序过程中，输入数据不含关于网络假定延迟的参数。

To the best of our knowledge, KNIGHT is the first permissionless parameterless consensus protocol that is secure against any attacker with less than 50% of the computational power in the network.

据我们所知，KNIGHT 是第一个无许可的无参数共识协议，它可以安全地抵御网络中任何算力低于50%的攻击者。

These properties put KNIGHT at an inherently stronger spot than its counterparts: It is both faster and more secure, since it makes fewer assumptions and operates properly despite varying network conditions.

这些特性使得 KNIGHT 比其同类产品有着决定性的优势：更快、更安全，要求的假设更少。它在复杂的网络环境下它也能正常运行。

一般来说，一种算法所要求的假设越多就越难走出实验室

1.1 KNIGHT optimization framework

Conceptually, KNIGHT is an evolution of the PHANTOM optimization framework, which in turn is an evolution of NC.

In NC, the longest chain of blocks within the tree is selected and extended.

PHANTOM generalizes the longest chain rule: Rather than selecting the longest chain, it selects the largest sufficiently connected subset of blocks.

The following definition from captures "well-connectedness":

Definition 1.

$Given\ a\ DAG\ G = (C, E), a\ subset\ S \subseteq C\ is\ called\ a\ k-cluster, if\ \forall B \in S: |anticone(B) \cap S| \leq k$

Here, the anticone of a block is the set of blocks whose order with respect to it is not dictated by the DAG topology; see Figure 1.

1.1 KNIGHT 优化框架

从概念上讲，KNIGHT 是 PHANTOM 优化框架的演变，而 PHANTOM 优化框架又是 NC 的演变。

你可以在《GHOST DAG》里面找到 PHANTOM

在 NC 中，算法会选择并扩展树中最长的块链。

BTC 的最长链原则

PHANTOM 将最长链规则泛化：与其关心哪条链是最长的，还不如选择那些最大的、连通性好的块子集

以下定义阐释了“良好连接性”的概念：

定义 1

假设有一个 DAG 图$G = (C, E)$，其中 C 表示节点集合，E 表示边集合。

对于 C 集合一个子集 S，如果对于集合 S 的一个任意子集 B，满足：$anticone(B)$与集合 S 的交集的大小不超过 k，我们将拥有如上性质的集合 S 称为一个$k-cluster$。

数学语言：

给定一个DAG，其中$G = (C, E)$，若$\exists S \subseteq C$，则称 S 是一个$k-cluster$，当且仅当 $\forall B \in S, s.t. |anticone(B) \cap S| \leq k$

请留意，为了方便，接下来都将使用数学语言书写

其中$\forall$表示任意，$\exists$表示存在，$s.t.$是 satisfy 的缩写，表示满足

在这里，算子$anticone$的作用如下：对于给定的一个块，求出其在有向无环图(DAG)拓扑结构下没有明确先后关系的其他块的集合；请参见图 1

简单地说，函数$anticone$的作用就是找出与给定块（我们称为 A）无关的块（称为B_i），这里的无关定义如下：

- *从B_i出发不能到达 A*
- *从 A 不能到达B_i*

图 1 请参见《GHOST DAG》

Formally, PHANTOM solves the following optimization problem:

PHANTOM Optimization: Maximum k-cluster sub-DAG (MCK_k)
Input: $DAG\ G = (C, E)$,k
Output: $A\ subset\ S^* \subset C\ of\ maximum\ size\ s.t. anticone(B) \cap S^* \leq k\ for\ all\ B \in S^*$

Similarly to other parameterized consensus protocols, the parameter k of PHANTOM represents an upper bound on the network's latency (technically, on the number of blocks per one unit of delay, with high probability).

Observe that for $k = 0$, PHANTOM coincides with NC, as the longest chain is the largest 0-cluster.

Indeed, when the block interval is large (e.g., Bitcoin's 10 minutes per block), the latency parameter k can be set to 0.

In contrast, a system enjoying a high block creation rate would require setting k to be much larger.

For instance, in Kaspa, a cryptocurrency based on PHANTOM, the block interval was set to 1 second, and k was hardcoded with a value of 18, reflecting an assumption of $D \leq 10$ seconds; see for a live visualization of the live DAG of Kaspa.

In contrast, KNIGHT offers an alternative optimization framework, which does not pre-assume a latency bound:

PHANTOM Optimization: Minimal k Majority Cluster sub-DAG ($MkMC$)
Input: $DAG\ G = (C, E)$
Output: $A\ subset\ S^* = MCS_k(G), s.t.\ k\ is\ minimal\ and\ |S^*| \geq \frac{|C|}{2}$

形式上，PHANTOM 解决了以下优化问题：

PHANTOM 优化：求出最大的 $k-cluster \subset DAG(MCK_k)$
输入：$DAG\ G = (C, E)$,k
输出：最大的子集 $S^* \subset C\ s.t. anticone(B) \cap S^* \leq k, \forall B \in S^*$

与其他参数化共识协议类似，PHANTOM 的参数 k 表示网络延迟的上限（从技术上讲，这是指高概率下每单位延迟的块数）。

如果$k = 0$，那么 PHANTOM 与 NC 没有差别，因为最长的链就是最大的$0-cluster$。

实际上，当块间隔很大时（例如，比特币$10min$一个区块）延迟参数 k 可以设置为 0。

相比之下，享有高块创建速度的系统需要将k设置得更大。

例如，在基于 PHANTOM 的加密货币 Kaspa 中，区块间隔被设置为 1 秒，k被硬编码为 18，这里是假设了$D \leq 10$秒。

其中参数D表示网络延迟的上限，你可以在《GHOST DAG》中找到

相比之下，KNIGHT 提供了另一种优化框架，即不预先假定网络延迟的范围：

PHANTOM 优化：找到最小的 k，同时 $k-cluster$要尽可能覆盖 DAG
输入：$DAG\ G = (C, E)$
输出：S^*，S^*满足如下性质：

- $\forall B \in S^*, s.t. |anticone(B) \cap S^*| \leq k$
- $|S^*| \geq \frac{|C|}{2}$
- k是最小的

在 G 中找到覆盖超过一半的子集 s，记为 S。这相当容易；S 是包含集合的集合；

不妨预先给定一个较大的 k，并在 S 中找到所有满足k-cluster 性质的 s，记为S^*；

维持S^*的性质，并使得 k 尽可能小；

满足条件的 k 是唯一的

That is, rather than selecting the largest k-cluster for one predetermined value of k, we select the largest k-cluster for each value of k, and pick the minimal k whose maximizing cluster covers 50% of the DAG.

We thus utilize the honest majority assumption to recognize a subset of blocks that are sufficient to counter an attack.

In this way, we avoid the need to know k in advance, and allow the protocol to self-adjust to the real time latency.

The actual KNIGHT protocol contains more components than the optimization problem $MkMC$, not merely for efficiency but also for security considerations – primarily, natural or malicious changes in the latency1 – as will be described formally in Section 2.

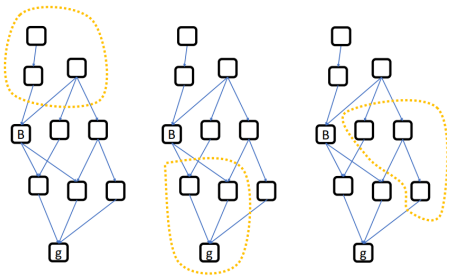

Figure 1: The topology of a blockDAG induces a partial ordering over blocks.

The figure on the left marks blocks provably created after block B, which are called its future set.

Similarly, the figure on the middle marks blocks provably created before B, its past set.

The right-most figure marks blocks whose ordering with respect to B is ambiguous and must be dictated and agreed by the consensus protocol.

换句话说，与其根据预先确定的k选择最大的k-cluster，还不如为每个 k 选择最大的k-cluster，然后利用如下策略选取最终的 k：

- k-cluster覆盖DAG至少50%；
- k 最小

因此，在多数节点都是诚实的假设下，我们找到了能够足以抵御攻击的区块子集。

通过这种方式，我们避开了需要预先知道k的麻烦，并允许协议根据延迟实时调整参数。

*现在，即使你完全不理解上面在说什么，你也可以自信地说出 DK 协议下的 blockDAG 一大优势——**自适应区块创建速度***

出于效率、安全的考虑，尤其是在网络被有意或无意改变延迟的情况下，实际的 KNIGHT 协议比优化问题 MkMC 要复杂得多，而这一部分将在第二节正式描述。

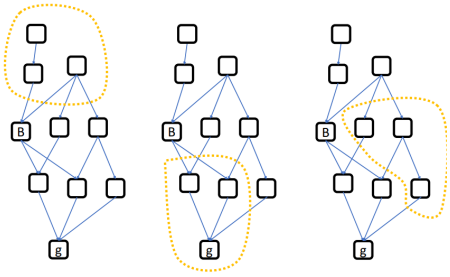

图 1：块 DAG 的拓扑结构使块之间存在偏序关系。

左图标注了在块 B 之后被创建的块，称为其未来集合。

同样地，中间的图标注了在块 B 之前被创建的块，称为其过去集合。

右侧的图标注了在块 B 相对顺序上存在歧义的块，这部分必须由共识协议来确定和达成一致。

请留意，时间方向与图中箭头方向相反，即向上。

在《GHOST DAG》中，左图可用 future(B) 表示，中间可用 past(B) 表示；正如上文确定k-cluster所说，右图被圈住的部分正是anticone(B)，这些块无法到达 B，B 也无法到达这些块。

1.2 Parameterlessness

KNIGHT differs from previous work on proof-of-work-based consensus protocols which typically operate in the synchronous setup and assume an a priori upper bound over D, either explicitly or implicitly.

For instance, Bitcoin's difficulty adjustment algorithm is targeting a block creation rate of $\lambda = 1/600$ blocks per second, which reflects an underlying assumption that $D \ll 600$ seconds.

Similarly, when instantiating the PHANTOM protocol, one must preconfigure the protocol's k parameter which represents the expected number of blocks created in one unit of delay, reflecting an assumption that $D << \frac{k+1}{\lambda}$.

Parameterlessness has two implications. First, confirmation times in parameterized protocols are typically limited by their parameter—they are a function of the hardcoded parameter, regardless of the network's actual latency.

Thus, even when the actual latency of blocks in Bitcoin is 1 or 2 seconds (as is the situation for most of the time), the protocol's convergence times is in the order of tens of minutes.

Similarly, Kaspa's convergence time remains in the order of tens of seconds even when its latency is way below 10 seconds.

As a parameterless protocol, KNIGHT avoids this shortcoming and allows the network to converge according to its actual conditions.

1.2 无参数

KNIGHT 与以前基于工作证明的共识协议的研究不同，后者通常在同步设置中运行，而这些协议都直接或间接预先设定了参数 D。

参数 D 表示网络延迟的上限

例如，比特币的难度调整算法将区块创建速度锁定在$\lambda = 1/600$区块/秒，这反映了$D << 600$秒的基本假设。

同样地，在实例化 PHANTOM 协议时，同样需要预先配置协议的参数 k，该参数表示在一个单位网络延迟中预期创建的块数，反映了一个假设，即$D << \frac{k+1}{\lambda}$。

无参数化有两个含义。首先，参数化协议中的确认时间通常受其参数限制——它们是硬编码参数的函数，也就是说这些协议不关心网络的实际延迟。

因此，即使比特币中块的实际延迟为 1 或 2 秒（大多数时候都是这种情况），协议也要花数十分钟的时间才能完成收敛工作

收敛工作：指在一个分布式系统中，所有节点都达成共识。这里指在六次确认或更高次数的确认后，BTC 主网才确认这些新的交易是有效的

同样地，即使在网络延迟远低于 10 秒的情况下，Kaspa 的收敛时间仍然能保持在数十秒这个数量级

收敛时间：此处指的是等待足够多的节点确认所需的时间

作为一个无参数协议，KNIGHT 可以避免这种缺陷，并允许主网根据其实际条件进行收敛工作。

Thus, when the network's adversarial latency is very low, the ordering of KNIGHT will converge immediately, allowing clients to confirm transactions within a few Internet RTTs (Round Trip Times); and when the network is slow and clogged, the ordering will take longer to converge and transactions longer to confirm.

因此，当网络的敌对延迟非常低时，KNIGHT 能立刻完成排序收敛工作，使客户端能够在几个 RTT 内确认交易；当网络缓慢拥塞时，排序将花费更长时间才能收敛，交易也需要更长时间才能确认。

敌对延迟：在区块链中，有时会出现恶意节点故意延迟向网络广播自己的新块，或故意广播无效块的情况，这被称为“敌对延迟”。这些节点的行为会影响网络的延迟和性能

RTT(Round Trip Times)：从发送端发送数据开始，到发送端收到来自接收端的确认，总共经历的时延

Crucially, we emphasize that this responsivenss is with respect to the worst-case adversarial latency; in Subsection 1.4 we distill this nuance.

重要的是，我们强调这种响应是为了针对最坏情况下的敌对延迟；在第 1.4 小节中我们会进一步阐释其中的原理。

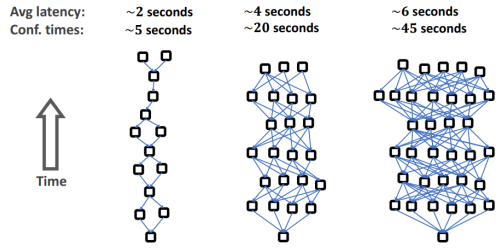

Figure 2: Confirmation times of KNIGHT under various network conditions.

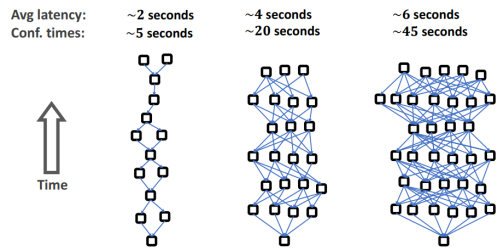

图 2：在不同网络条件下，KNIGHT 的确认时间。

Since it is parameterless, KNIGHT performs according to the (client's bound over the maximum) network's latency, which is typically proportional to the DAG's average width, allowing it to confirm transactions very fast when the network is healthy and speedy.

由于 KNIGHT 是无参数的，因此它的表现通常取决于网络延迟（客户端容忍的最大延迟），网络延迟通常与 DAG 的平均宽度成正比，当网络良好时能够非常快地确认交易。

The confirmation times correspond to a 20% attacker, and confidence parameter $\epsilon = 0.1$.

确认时间对应于一个 20%的攻击者，置信度参数$\epsilon = 0.1$。

模型假设网络中至多有 20%的恶意节点（攻击者）。

置信度参数，即显著性水平 $\alpha = 0.1$，对于大多数正常情况下的节点来说，它们有$1 - \alpha$（90%）的把握认为块的排序是正确的（其备择假设为：块的排序是错误的）。

在统计学中，此部分是假设检验的内容，显著性水平一般约定为 0.05。显著性越小，表示犯错误（此处指选择了备择假设）的概率越低，但代价就是更强的约束条件。此类显著性不能武断评价好坏。

Figures 2 and 3 demonstrate this effect.

In the former, a DAG of various "widths" is presented, corresponding to different network latencies.

When the network is speedy, miners are aware of almost all blocks created by their peers, blocks enjoy small anticones (or "gaps") of size 1 at most, and transactions can be confirmed quickly.

On the other extreme, many blocks are created in parallel, blocks suffer from larger anticones, and transactions take longer to confirm.

This scenario represents either a slow down in the network, or a system intentionally parameterized with a high block rate, e.g., $\lambda = 10$ blocks per second.

Figure 3 further compares the effect of varying network conditions on parameterized protocols (e.g., NC, PHANTOM) and parameterless ones (e.g., KNIGHT).

The confirmation times in the former protocols are constant, accounting to the hardcoded latency-dependent parameter; worse yet, when the network suffers an anomaly, and message delays violate the bound, transactions cannot be confirmed altogether.

图 2 和图 3 展示了这种影响。

在前者中，呈现了不同网络延迟下各种"宽度"的 DAG。

当网络速度很快时，矿工几乎能够知道所有由其同行创建的块，每个块至多有 1 个 anticone 块（或"间隙"），交易可以很快得到确认。

当网络状况堪忧时，有许多块是并行创建的，每个块有更多的 anticone 块，交易需要更长时间才能确认。

这种情况表示网络速度变慢，或者是某一个系统故意设置了高块创建速度，例如$\lambda = 10\ block/sec$。

在网络较差的情况下，可能会发生网络拥塞或丢包等情况，导致传输的区块被延迟或者丢失，这时如果有多个节点同时创建区块并向网络广播，就会出现多个区块并行存在的情况。

而在网络良好的情况下，传输的区块能够及时到达其他节点，因此不太可能同时有多个节点创建出新的区块，区块的创建相对顺畅。

图 3 进一步比较了不同网络条件对参数化协议（例如 NC、PHANTOM）和无参数协议（例如 KNIGHT）的影响。

前一种协议的确认时间是恒定的，取决于硬编码与延迟相关的参数；更糟糕的是，当网络发生异常，并且消息延迟超过了可接受范围时，交易可能根本无法得到确认。

In contrast, the confirmation times of parameterless protocols correspond to the (bound of the client over the maximum) current latency in the network, and, in particular, the network remains fully operational, yet slow, during periods of network anomaly.

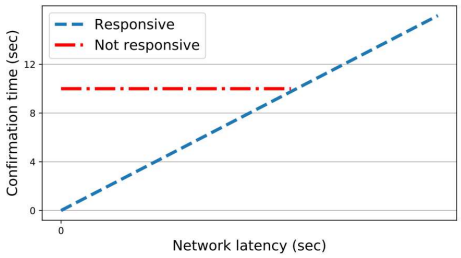

Figure 3: A qualitative comparison of the confirmation time behaviour of parameterized, non-responsive protocols and parameterless, responsive ones.

Confirmation times in the latter case are fast when the network is smooth and speedy, whereas in the former confirmation time is still limited by the constant hardcoded worst case bound.

Additionally, when the bound of a parameterized is violated, transactions may not be confirmed safely, whereas parameterless responsive protocols adapt to the anomaly and allow confirming transactions more slowly than usual, yet safely.

A second implication of parameterlessness is added security: KNIGHT enjoys a stronger security than existing permissionless protocols, as network hiccups do not interrupt consensus, because they do not violate assumptions necessary for its proper operation.

相比之下，无参数协议的确认时间取决于当前网络中的最大延迟（客户端容忍的最大延迟），特别是在网络异常期间，网络仍然可以完全运行，但速度变慢。

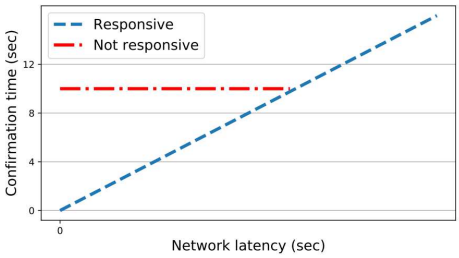

图 3：参数化、无响应式协议（简称红线）和无参数、响应式协议（简称蓝线）的确认时间行为的定性比较。

对于蓝线来说，当网络平稳迅速时，确认时间很快，而对于红线来说，确认时间仍受恒定的硬编码最坏情况限制。

也就是红蓝两线交点的左侧，可以看到蓝线耗费的时间少于红线

此外，当网络情况超过参数化协议的容忍界限时，可能无法安全地确认交易，而无参数、响应式协议会适应异常情况，虽然以比平常更慢的速度确认交易，但是安全性得到了保证。

参数无关性的第二个含义是增加了安全性：KNIGHT 比现有的无许可协议具有更强的安全性，因为网络的小波动并不会打断共识，如果按照正常操作的水准来看，网络卡顿不算什么。

网络波动几乎不会造成损失

1.3 Partial Synchrony

Traditionally, a consensus protocol is said to be partially synchronous if an upper bound on the network latency exists but is unknown to the protocol.

However, proof-of-work consensus introduces some ambiguity into this classification, as it decouples the transaction ordering protocol from the finality protocol: The core of consensus is the transaction ordering rule (e.g., Bitcoin's longest chain rule, KNIGHT's DAG ordering).

This is the canonical algorithm which defines the system, and which all participants run in the same way, including adversarial nodes—individual interpretations of the ledger which differ from the canonical procedure are pointless.

In contrast, transaction finality (e.g., the confirmation count in Bitcoin) is a non-binding procedure which each client or user configures and calculates locally according to their own beliefs and needs, and bears the consequence.

1.3 部分同步

一般地，如果存在一个网络延迟上限，但共识协议却不知道这个上限的存在，我们就把这种协议称为部分同步协议。

然而，工作量证明共识对这种分类带来了一些不确定性，因为它将交易排序协议与终态协议分离开来：共识的核心是交易排序规则（例如，比特币的最长链规则，KNIGHT 的 DAG 排序）。

在工作量证明中，交易排序协议和终态协议是分离的，它们之间没有先后顺序。

交易排序协议定义了如何对交易进行排序，并在区块链上构建一个最长的链或者一个 DAG，而终态协议则决定了区块链上的交易是否可以被认为是最终的、不可更改的状态。两个协议同时运行，但其处理的内容是不同的，它们在整个共识过程中起着不同的作用。

二者分开可以更好地应对网络延迟和攻击等问题。

这里的不确定性是指工作量证明将交易排序协议与终态协议分离开来，导致确认时间取决于网络延迟的问题变得有些含糊不清。

简单地说，就是不知道把锅甩给谁。

这是定义系统的经典算法，所有参与者都必须遵守一个共识，包括敌对节点，如果一个节点的账本数据与整个网络所共识的数据不一致，那么这个节点的账本数据就是没有意义的

相比之下，交易的确定过程并非强制（例如，比特币的确认计数），每个客户端或用户根据自己的想法和需求进行本地配置和计算，同时承担相应的后果。

E.g., a Bitcoin user who believes that malicious miners possess currently less than 33% of the hash rate will confirm transactions faster than one who believes the bound to be 49%, and a 34% attacker will harm the former but not the latter.

例如，如果一个比特币用户（称为A）认为恶意矿工目前拥有的算力低于总算力的33%，此时有另一个用户（称为B）认为恶意矿工目前拥有的算力低于总算力的49%，那么A能比B更快完成交易。但是，如果攻击者拥有的算力大于总算力的34%，A就麻烦了，但B没有受到影响。换句话说，交易的确认速度取决于用户对攻击者拥有算力的估计。

Another example is SPECTRE, wherein the user needs to additionally specify her belief on the latency bound.

另一个例子是SPECTRE，用户需要额外给出她对网络延迟上限的预估。

SPECTRE是一种基于DAG的参数化共识算法，需要指定网络延迟上限

This parameter is configured individually and is inconsequential – and, in fact, not-communicated – to the rest of the network.

此参数由用户单独配置，对于其他网络部分是无关紧要的，实际上也不会传达给网络的其他节点。

在SPECTRE中，参数用于计算确认阈值，它指定了节点在哪个时间点将某个区块视为已确认。

然而，这个参数只影响节点的本地视图，而不会通过网络广播。其他节点不会知道一个节点选择了哪个参数值，因此参数值的变化不会对整个网络造成影响。

因此，可以说这个参数对于其他网络部分是无关紧要的。

Therefore, in this paper we refer to a consensus system whose transaction-ordering rule is agnostic to latency as partially synchronous or parameterless, interchangeably, even if transaction-finality depends on a latency bound (configured by the user locally).

因此，在本文中，我们将把那些交易排序规则与网络延迟无关的共识系统称作部分同步或无参数系统，尽管这些系统的交易确认时间可能会受到网络延迟（由用户本地配置）的限制。部分同步和无参数这两个术语可以互换使用。

这里的互换使用背后的含义是指：这些术语都用来描述在共识系统中，节点之间的通信受到某些限制或不确定性，而这种限制或不确定性是由于无法精确地估计网络延迟所致。

两个术语都表明了这种共识系统对于网络延迟的掌握是有限的

We believe that this terminology is most fitting to the proof-of-work model.

我们认为这个术语最适合工作量证明模型。

1.4 Responsiveness

In the lack of an a priori latency assumption, confirmation times in a parameterless consensus system correspond to the real network latency.

However, “real latency” has two profoundly different interpretations: the observable latency in the network, and the worst case latency that an attacker may cause.

Indeed, a capable attacker may allow – or even assist – the network to operate smoothly, selectively, and disrupt it during a later stage of the attack.

A protocol that has the strong property of confirming transactions according to the observable latency is called responsive .

Despite being parameterless, KNIGHT is not responsive in this sense of performing tightly with the current observable latency, rather is responsive in the weaker sense of performing tightly with the current maximum latency causable by an adversary.

Indeed, in KNIGHT, it is not enough for the client to set a local bound on the observable latency, rather the bound should reflect the maximum latency that may be caused by the attacker.

That is, even if messages currently propagate fully within 1 or 2 seconds, if an attacker may disrupt the network and cause messages to take up to 30 seconds to go through, D should be set by the client to 30 seconds.

This limitation of KNIGHT is unavoidable, since no parameterless protocol with 50% byznatine tolerance threshold can be responsive:

1.4 响应性

在没有预先假定网络延迟的情况下，参数化共识系统中的确认时间与实际网络延迟是相对应的。

然而，“实际延迟”有两种不同的解释：

- 实际测量到的网络延迟
- 可能由攻击者引起的，最坏情况下的延迟。

事实上，一个有想法的攻击者可以先默认或甚至协助网络顺畅运行（这也是攻击的一部分），然后在攻击的后期对主网进行干扰。

如果一个协议能根据可观察到的延迟确认交易，那我们称之为响应式协议。

尽管 KNIGHT 是无参数的，但它并不会对实际测量到的网络延迟做出反应，而是对那些可能是由攻击者造成的网络延迟做出响应。因此，KNIGHT 不是普通的响应式协议。

实际上，在 KNIGHT 中，客户端不能只设置实际网络延迟的范围，还应该考虑攻击者的存在，并设置可能的最大网络延迟。

也就是说，即使当前消息可以在 1 或 2 秒内传播出去，考虑到攻击者的破坏行为导致消息延迟 30 秒，客户端应该要把参数 D 设置为 30 秒。

D 是网络延迟的上限。

KNIGHT 无法克服这一点，如果一个协议想要既具有响应式的特性（能够及时适应网络的延迟），又要具有 50%的拜占庭容错阈值（即能够容忍半数的节点是恶意的），那么这个协议就不能是无参数协议。鱼和熊掌不可得兼。

Theorem 14 (Responsive protocols cannot tolerate 1/3 corruption)[Pass and Shi].

No secure permissionless consensus protocol that is also responsive can tolerate 1/3 or more corruption.

定理 14（响应式协议不能容忍 1/3 的恶意节点）[Pass 和 Shi 著]。

没有一个安全的无许可共识协议同时满足如下条件：

- 是响应式
- 能容忍 1/3 及以上的恶意行为

"Pass and Shi" 指的是研究人员 Rafael Pass and Elaine Shi，他们是分布式系统和密码学领域的专家。

1.5 Consensus protocols, principal categories

Consensus protocols are generally classified and compared according to the following aspects:

- What are the assumptions made by the protocol on the underlying network and behaviour of nodes.
  The stronger the assumptions the weaker the protocol.
- When its assumptions are preserved, how does it perform, specifically, how fast is consensus reached.
- When its assumptions are violated, does the protocol recover, and how fast it recovers. A protocol guaranteed to recover from past failures is called self-stabilizing.
- If the underlying system is used to settle a live queue of transactions, we also ask: How many transactions can the protocol serve, i.e., what constraint on the transaction throughput the protocol imposes or its assumptions require.

Through these categories we now survey, with some brevity, KNIGHT 's main properties:

1.5 共识协议，主要分类

共识协议通常根据以下方面进行分类和比较：

协议对底层网络和节点行为做出了什么假设。如果假设越强，协议应用面就很窄。

在保持其假设的情况下，它的性能如何，具体来说，达成共识的速度有多快。

当其假设被违反时，协议是否能够恢复工作，以及恢复的速度有多快。如果一个协议能够从过去的故障中恢复，我们称之为自稳定协议。

如果一个底层系统可用于实时解决交易队列，我们还会问：协议可以处理多少交易，即协议对交易吞吐量有无限制或者其假设的限制是什么。

通过这些类别，我们现在简要地研究一下 KNIGHT 的主要属性：

1.5.1 Model assumptions.

KNIGHT 's fault model is the byzantine setup, which allows the attacker to deviate arbitrarily from the protocol's rules.

We follow the proof-of-work model which assumes a computationally bounded attacker which possesses less than 50% of the computational power in the network.

This assumption is considered to be weaker (hence more secure) compared to traditional permissioned setups which require a priori knowledge of participating nodes, and compared to proof-of-stake which typically requires a fixed and identifiable set of nodes at the beginning of each epoch.

1.5.1 模型假设

KNIGHT 的故障模型是拜占庭式的，此模式下攻击者可以执行任何可能的攻击行为。

我们遵循工作量证明模型，该模型假设攻击者算力有限，并且在网络中拥有的算力少于 50%。

相比起与那些需要先验知识的传统许可设置，抑或是那些通常需要在每个 epoch 的开头固定可识别节点的权益证明，这个假设无异更加宽松（当然也更加安全）。

先验知识的传统许可设置：传统的许可设置是指在区块链网络中需要预先确定参与的节点，这些节点被认为是可信的，并且它们的身份和权力都是在网络部署之前就已经确定的。这种设置需要先验知识来确定网络中的节点身份和权力，通常由中心化的实体或权威机构来管理和验证。

epoch 用于定义区块链上特定事件发生的时间纪元（era of time），比如激励支付的时间，新的验证者组负责验证交易的时间

1.5.2 Confirmation times (asymptotic)

The lack of a hardcoded latency (or latency dependent) parameter in a partially synchronous protocol is tightly related to its speed of confirmation: Transaction confirmation times are a function of the actual latency in the network (Subsection 1.3 contains an important reservation of this statement in our context).

Performance is commonly discussed in two modes – optimistic and pessimistic.

The former accounts to the scenario where all participating nodes behave properly, and there is no visible attack.

1.5.2 确认时间（渐近）

如果部分同步协议缺乏硬编码的延迟参数（或依赖于延迟的参数），这将会对其交易的确认速度造成影响：对于交易确认时间与网络实际延迟二者来说，它们构成函数关系（在第 1.3 小节，我们给出了相关的具体说明）

性能通常以两种模式讨论：乐观和悲观。

前者考虑的是所有节点都是正常的，不存在攻击者的情况。

In this optimistic scenario, KNIGHT confirms transactions in

$$O((\frac{\ln(1/\epsilon)}{\lambda}+D)/(1-2\alpha)+D^2\lambda)$$

seconds, where λ is the block creation rate in units of blocks/sec (adjusted via a "difficulty adjustment" algorithm adapted from Bitcoin),

在这种乐观的情况下，$KNIGHT$能够以

$$O((\frac{\ln(1/\epsilon)}{\lambda}+D)/(1-2\alpha)+D^2\lambda)$$

的效率确认交易。

在计算机编程中，使用$O(F[n])$表示算法复杂度的渐进上界，这个算法的运行时间不会超过$F[n]$的一个常数倍α，而且$\alpha \overset{n\to+\infty}{\to} 1$。

比如是如果算法复杂度是 O(1)，那么无论怎么输入，计算机只需一次计算便能得到结果。

其中：

参数λ表示以$block/sec$为单位的区块创建速度（通过使用从比特币那里改编的“难度调整”算法来进行调整）。

难度调整算法是一种在区块链中用于控制区块创建速率的机制，它会根据网络上的矿工数量和计算能力的变化来自动调整区块的难度，以保持区块链的安全性和稳定性。

D is an upper bound on the recent delay in the network, $0 \leq \alpha < 1/2$ is the attacker's size, and $0 \leq \epsilon < 1/2$ is the required confidence.

D是当前网络延迟的上限，$0 \leq \alpha < 1/2$ 为攻击者的规模，$0 \leq \epsilon < 1/2$ 为所需置信度。

什么是置信度？前面有说……

As in any proof-of-work protocol, the parameters α and ϵ are set by the client.

Uniquely to KNIGHT (and to SPECTRE), the parameter D too is set by the client—an underestimation by the client will lead to her premature acceptance of transactions, whereas an overestimation will cause her to wait more time than necessary before confirming.

In the pessimistic scenario, a visible manipulation of the DAG is ongoing, and confirmation times are significantly slowed down.

与任何工作量证明协议一样，参数 α 和 ϵ 由客户端设置。

与其他协议不同的是，对于 KNIGHT（包括 SPECTRE）来说，参数 D 也由客户端设置——如果参数 D 偏小，交易就会过早成交；如果参数 D 偏大，交易则需要比正常交易更多的时间才能完成。

在悲观情况下，DAG 正在遭受非常明显的攻击行为，确认时间显着减慢。

Analyzing the convergence time in this case in a tight manner is intractable, and we are thus left with an exponential bound on confirmation times: $O(\frac{1}{\lambda}(\exp(c \cdot D\lambda/(1-2\alpha)) + \ln(1/\epsilon)/(1-2\alpha)))$ seconds.

We emphasize, however, that this bound is far from tight, assumes an unrealistically strong attacker, and furthermore payments of honest users can still be confirmed in quadratic time as in the optimistic case.

在这种情况下严格分析收敛时间是不可行的，因此我们只能得到一个指数级的交易确认时间上限 $O(\frac{1}{\lambda}(\exp(c \cdot D\lambda/(1-2\alpha)) + \ln(1/\epsilon)/(1-2\alpha)))$ 。

然而，需要注意的是，由于假设了一个实际中不太可能存在的高算力攻击者，这个上限没有得到缜密的证明，但即使是这样，那些来自诚实节点的交易仍然可以像乐观情况下所假设的那样，能够在$O(n^2)$内成交。

Indeed, as long as the user did not publish a visible conflict (aka double spend), her transaction is commutative with other recent transactions in the DAG, hence the receiving client may confirm it despite the ordering still converging.

实际上，只要用户没有使用明显的攻击行为（比如双花攻击），那么她的交易就可以和 DAG 中其他最近的交易完成确认，因此，接收客户端可以在交易排序仍在逐步完善的情况下确认该交易。

双花攻击（Double Spending Attack）指的是在某个区块链网络中，一个参与者试图多次使用同一份数字资产，也就是同一个交易输出，从而欺骗其他节点相信这些资产已被多次使用。这种攻击在区块链网络中是一种常见的欺诈行为，可以通过一些防御措施来预防。

1.5.3 Self stabilization.

Similarly to NC and other proof-of-work consensus protocols, KNIGHT is self-stabilizing: If the 50% threshold was violated at some point in the past, KNIGHT recovers and transactions may be confirmed safely once the conditions are met; the recovery time is linear in the length of the violation phase.

Similarly, the latency parameter D which is set by each client locally should correspond to the recent delay in the network, and need not account for the worst case historical latency.

1.5.3 自稳定

与 NC 以及其他工作量证明共识协议类似，KNIGHT 是自我稳定的：如果在过去的某个时间点，有 50%的节点（这已经是临界点了）违反了协议规定，那么 KNIGHT 就会启动自我修复，一旦满足可运行条件，交易便可安全地确认；恢复所需时间与违规节点数目呈线性关系。

同理，每个客户端在本地设置的延迟参数应该与网络中最近的延迟相对应，而且不需要考虑到最坏情况下的历史网络延迟。

Contrast these properties to proof-of-stake protocols, which rely heavily on finality, and which fail therefore to recover from historical catastrophes.

将这些属性与 POW 协议进行对比，这些 POW 协议严重依赖最终性，一旦发生历史性灾难，就无法从中恢复。

最终性：指的是一个区块被确认之后，它的确认状态不会被更改的概率足够高，即该区块已经被永久性地确认。

1.5.4 Throughput.

In contrast to NC, and similar to other DAG-based consensus protocols, KNIGHT remains secure under arbitrarily high throughput configurations—the block rate, and the block size, should be constrained only according to the capacity of nodes' hardware and that of the network's backbone.

All in all, in this work we propose a novel proof-of-work based parameterless consensus protocol.

As far as we are aware, KNIGHT is the first proof-of-work protocol to achieve this property under the partially synchronous model; the only other protocol to operate under this model is SPECTRE, which solves a weaker version of the consensus problem ("weak liveness"), and which supports therefore only the use case of payments where transactions of honest users are commutative.

Our protocol generalizes PHANTOM in that the two coincide when the delay is constant; when the delay is negligible relative to the block creation rate, the protocol further coincides with NC.

1.5.4 吞吐量

与 NC 不同，但与其他基于 DAG 算法的共识协议相似的是，KNIGHT 能在任意高的吞吐量配置下保持安全的块创建速度和块创建大小，而主网能做到这一点应该只需要如下条件来约束：

- 节点硬件条件
- 网络的硬件容量

总而言之，在这项工作中，我们提出了一种新式的基于工作量证明的无参数共识协议。

据我们所知，目前 KNIGHT 是第一个在部分同步模型下实现此属性的工作量证明协议；而另一个在部分同步模型下运行的协议是 SPECTRE，它解决了共识问题的一个较弱版本（"weak liveness"），而且只有在诚实节点的交易是可交换的情况下才能保证交易的成功确认。

"weak liveness"：指在一个共识协议中，只要存在足够多的诚实节点，就能保证最终有某个节点的提议被接受，即系统最终会进入一个共识状态。

"strong liveness"：指系统必须在有限时间内达成共识，即不允许永久性阻塞

我们的协议成功将 PHANTOM 推广到延迟恒定的情况下，当延迟相对于块创建速率非常小，可以忽略不计时，KNIGHT 协议与 NC 协议类似

1.6 Structure of this paper

The remainder of this paper is organized as follows.

Section 2 contains the full description of the KNIGHT protocol.

Section 3 formalizes the model and the statement of KNIGHT's properties.

Section 4 discusses confirmation procedure for clients, and confirmation times.

In Section 5 we present implementation details.

We conclude with surveying related work in Section 6.

1.6 本文的结构

本文的其余部分组织如下。

第 2 节是对 KNIGHT 协议的完整描述。

第 3 节规范了模型和 KNIGHT 属性的阐述。

第 4 节讨论客户端的确认过程和确认时间。

第 5 节中，我们介绍了实现细节。

最后，在第 6 节中，我们将回顾和讨论其他与 KNIGHT 相关的工作。

2 THE DAG KNIGHT PROTOCOL

2.1 Preliminaries

The following terminology is used extensively throughout this paper.

We follow terminology established by previous works concerning DAG protocols.

In a block DAG $G = (C, E)$,C represents blocks and E represents hash references to previous blocks—edges thus point backwards in time.

$past(B, G)$ denotes the set of blocks reachable from B, and $future(B, G)$ the set of blocks from which B is reachable; these blocks were provably created before and after B, correspondingly.

$anticone(B, G)$ denotes the set of blocks outside $past(B, G)$ and $future(B, G)$; the time-relation between B and blocks in its anticone cannot be derived explicitly from the DAG topology.

See Figure 1. When context is clear, we write $anticone(B)$ instead of $anticone(B, G)$.

We denote by $tips(G)$ the set of blocks with indegree 0, that is, which are not referenced by any other block in the DAG.

2 DAG KNIGHT 协议

2.1 预览

本文将广泛使用以下术语。

我们将沿用以前关于 DAG 协议的工作所建立的术语。

即 PHANTOM DAG 协议

在 block DAG $G = (C, E)$中，C 表示块，E 表示对先前块的哈希引用，因此边向后指向时间。

边向后指向时间:在 block DAG 协议中，一个块通过包含上一个块的哈希值来引用先前的块。因此，块DAG中的边是指向过去的，也就是边向后指向时间。

简单说：块箭头的方向与时间方向相反

$past(B, G)$表示从 B 出发，可到达的块的集合；$future(B, G)$表示可到达 B 的块的集合；这些块在 B 之前和之后被证明创建。

$anticone(B, G)$ 表示 $past(B, G)$ 和 $future(B, G)$之外的块集合；因此，不能从 DAG 拓扑结构中明确地推导出 B 与 $anticone(B, G)$中的块之间的时间关系。

简单地说，就是 B 无法到达 $anticone(B, G)$，$anticone(B, G)$ 也无法到达 B。可以理解为不相关的块集合。

见图 1。当上下文清晰没有歧义时，我们用$anticone(B)$代指$anticone(B, G)$。

图 1 在 Part 1

我们用$tips(G)$表示入度为 0 的块集合，这意味着在 DAG 中，$tips(G)$没有被任何其他块引用。

The system is initialized with some known block $genesis$; if a $sub\text{-}DAG\ G' \subseteq G$ has only one block with out-degree 0,we denote it by $genesis(G')$.

For convenience, we additionally regard the virtual block of the DAG, $virtual(G)$, which is a hypothetical (un-mined) block which points to the DAG's tips as its parents. Thus, $past(virtual(G)) = G$.

Essentially, $virtual(G)$ represent the block template for the next block to be created by the miner, if it is honest.

系统使用一些已知块$genesis$进行初始化；我们定义$genesis(G')$，它的含义如下：$sub\text{-}DAG\ G' \subseteq G$只有一个出度为 0 的块

$genesis$译作起源，可以理解为起始块

为了方便起见，我们还将 DAG 的虚拟块$virtual(G)$视为一个假设的块（未挖掘），它将 DAG 的 tips 作为其父块。因此，$past(virtual(G)) = G$。

实际上，如果节点是诚实的，$virtual(G)$就代表矿工下一个要创建的块的模板。

$virtual(G)$是最新的块，因此$virtual(G)$的过去块自然就是整个DAG，即$past(virtual(G)) = G$。

2.2 PHANTOM optimization paradigm

The PHANTOM protocol proposed an optimization problem as a generalization of NC (see box in Section 1).

The optimization targets the largest k-cluster, for a predetermined fixed parameter k which is a function of the worst case latency in the network.

In a $k\text{-}cluster$, each block is connected via the DAG topology to all but at most k blocks.

Since honest nodes possess a majority of the hashrate, and since blocks created by honest nodes reference one another, the largest k-cluster contains recent honest blocks with high probability, which suffices to secure the ordering.

2.2 PHANTOM 优化模式

PHANTOM 协议提出了一个优化问题作为 NC 的一般化（见第 1 节的框）。

该优化问题的目标是寻找最大的$k\text{-}cluster$，其中 k 是关于网络最坏情况下延迟的函数，它是预先确定的固定参数。

在$k\text{-}cluster$中，通过 DAG 拓扑结构，每个块至多与 k 个块相连。

由于诚实节点拥有大多数的算力，并且由于诚实节点创建的块相互引用，因此最大的$k\text{-}cluster$有高概率包含最近的诚实块，这足以确保排序的正确。

诚实节点创建的块相互引用：诚实的节点在创建新块时会参考之前由其他诚实节点创建的块，将其作为新块的先前引用。

2.3 KNIGHT optimization paradigm

KNIGHT adds another layer to the optimization problem (as presented in Section 1).

Rather than assuming k as an input to the problem, KNIGHT searches for the minimal k such that the largest $k\text{-}cluster$ covers at least 50% of the DAG.

This dual minmax optimization (min k, max $k\text{-}cluster$) allows us to tolerate just enough latency and disconnectivity among the selected set of blocks: Intuitively, selecting the cluster of a smaller k would compromise safety, exposing the ordering to manipulations by a minority attacker whose blocks do not cover 50% of the graph; selecting the cluster of a larger k would compromise liveness, as it would allow adversary blocks to inject themselves into the order even after honest blocks have settled.

Building on this parameterless optimization paradigm, we are able to devise a secure consensus DAG ordering rule that is responsive to the network's actual adversarial latency and is not constrained to a priori hardcoded bounds on the adversarial latency which require large safety margins and perform suboptimally.

2.3 KNIGHT 优化范式

KNIGHT 协议对优化问题（正如第 1 节中所述）添加了另一个层面。

KNIGHT 不再将 k 作为问题的输入，而是寻找最小的 k，同时$k-cluster$应该满足如下性质：

- $k\text{-}cluster$ 是最大的；
- $k\text{-}cluster$ 至少覆盖 DAG 的 50%；

这个双重 $minmax$ 优化（min k，max $k\text{-}cluster$）允许我们能够在选定的块集合中容忍一定程度的延迟和断开连接：直观地说，当选择较小的k时，选定的块集合可能无法覆盖 DAG 的 50%，这将使得攻击者能够通过操纵未覆盖 50%图形的区块排序来破坏系统的安全性；如果选择一个较大的k，那么选定的块集合将包含较多的块，其中可能有攻击者的块，这将导致在诚实块已经确定之后，攻击者的块注入到排序中，从而破坏系统的存活性。

存活性：指系统在某些条件下能够继续正常运行的性质，例如即使出现了一些故障或攻击，系统仍然可以继续处理请求和达成共识等。

通过对min k和最大max k-cluster进行双重优化，可以在安全性和存活性之间找到一个平衡点，以容忍足够的延迟和断开连接。

在这个基于无参数的优化模型的基础上，我们能够设计一个安全的共识 DAG 排序规则，该规则能够响应网络的实际敌对延迟，同时不受预设硬编码边界的限制，这些硬编码边界为了较大的安全保证而妥协，不得不表现出次优性能。

这代表 DAG KNIGHT 不会因为预设硬编码边界损失性能。

We reiterate, however, that KNIGHT is not responsive in the strong sense of performing according to the network's observable latency, rather according to the maximum latency that an adversary may cause in the current network; still, under normal Internet conditions, and with sufficient redundancy between peers, this should be in the order of a few seconds at most.

In fact, no protocol that is secure against corruption of up to 50% of the nodes can achieve responsiveness in this strong sense, as was proven by Pass and Shi.

需要明确的是，KNIGHT 协议并非完全响应网络的实际可测量延迟，而是根据当前网络中敌对方可能引起的最大网络延迟来执行相关策略；在正常的互联网环境下，如果节点之间具有足够的冗余，那么这个延迟最多只有几秒钟。

足够的冗余：有足够多的节点

文中还指出，没有一种协议可以保证在 50%节点被损坏的情况下同时实现安全性和强响应性，这是由 Pass 和 Shi 证明的。

强响应性：能够根据网络的实际延迟进行快速决策

Pass 和 Shi 是之前提到的两位密码学家

2.4 KNIGHT – Strawman version

Turning KNIGHT's optimization paradigm into a DAG ordering rule seems straightforward:

Algorithm 1 Naïve ordering algorithm

Input: G – the DAG to order
Output: Ordering of G

1. **function** Order-$DAG(G)$
2. $\boldsymbol{for}\ k = 0,1 \cdots \infty\ \boldsymbol{do}$
3. $S \leftarrow MCS_k$
4. $\boldsymbol{if}\ |S| \geq \frac{|C|}{2}\ \boldsymbol{then}$
5. Order G according to some (deterministic) topological sort that gives precedence to S
6. **return** the ordered DAG

2.4 KNIGHT—— 初始版本

将 KNIGHT 的优化范式转化为 DAG 排序规则似乎很简单：

算法 1 朴素排序算法

输入： G – the DAG to order
输出： Ordering of G

1. **function** Order-$DAG(G)$
2. $\boldsymbol{for}\ k = 0,1 \cdots \infty\ \boldsymbol{do}$
3. $S \leftarrow MCS_k$
4. $\boldsymbol{if}\ |S| \geq \frac{|C|}{2}\ \boldsymbol{then}$
5. 根据优先于S的某种（确定性的）拓扑排序对G进行排序
6. **return** the ordered DAG

目的：

输入：一个 DAG G

输出：排序后的 DAG

思想：

从最小的k-cluster 开始，逐步增加$k-cluster$ 的大小。（MCS_k 就是 k-cluster）

对于每个 S，如果 S 包含至少一半的节点，则使用某种确定性的拓扑排序对 DAG 进行排序，该排序将优先考虑 S 中的节点。最后返回排序后的 DAG。

此处优先考虑 S 中的节点是为了提高 S 中节点的影响力，从而提高协议的安全性。

However, line 3 involves solving an $NP\text{-}hard$ problem, and, additionally, Algorithm 1 is secure only in setups with constant latency and a naïve attacker.

可惜，第 3 行涉及解决一个$NP\text{-}hard$，此外，算法 1 只在恒定网络延迟和初级攻击者的前提下才能保证安全。

寻找一个大小为 k 的最大 clique（完全子图）是一个NP-hard 问题。

Wiki 分团问题

$NP-hard$ 问题是一类非常困难的问题，通常认为没有多项式时间的算法可以解决它们。

We will shortly describe the full version of KNIGHT, which is both efficient (quadratic in $D\lambda$) and secure against spontaneous or malicious changes in network latency.

我们将很快描述 KNIGHT 的完整版本，该版本既高效（二次的$D\ \lambda$）又能够安全地应对网络延迟的自发或恶意变化。

二次的$D\lambda$：指 DAG 的大小和层数成二次关系的计算复杂度。

具体来说，$D\lambda$是指 DAG 中节点的总数和最长路径的长度的乘积。

But first we wish to provide visual insight on the different behaviour of PHANTOM vs KNIGHT's optimization paradigm:

Figure 4 illustrates a ~ 35% attacker attempting a "freeloading" manipulation on the respective protocols.

但首先，我们希望提供有关 PHANTOM 与 KNIGHT 优化模型在不同行为下的直观观察：

图 4 展示了一个占大约 35%算力的攻击者试图对相应的协议进行"免费乘车"操作。

"免费乘车"是指一种对共享资源的恶意利用方式，即攻击者不付出任何代价，却能从网络中获得收益。

在区块链中，攻击者可能通过不挖矿或少挖矿的方式，从而获得比其他参与者更多的收益。

这种攻击方式通常是在共识机制和网络拓扑结构存在漏洞的情况下进行的。

Consider the case where PHANTOM was parameterized with $k = 5$, say, and where the honest network enjoys a period of extreme connectivity in the network such that its blocks form a chain (a $0\text{-}cluster$, in PHANTOM terminology).

假设 PHANTOM 的参数设置为$k = 5$，而诚实节点在网络中拥有极高的连接性，在此期间其块形成一条链（PHANTOM 术语：$0\text{-}cluster$）。

PHANTOM then considers these blocks as part of the largest 5-*cluster*, and they will precede the second half of the honest chain in the final ordering. KNIGHT, in contrast, is not easily misled—it will recognize that $k = 0$ suffices to cover the majority of the DAG.

The same intuition applies generally to any scenario where the network's current (adversarial) latency is smaller than the worst case (adversarial) latency.

Moreover, if the network suffers excessive delays due to some anomaly, and PHANTOM's latency bound is violated, transactions may not be confirmed.

KNIGHT's operation, in contrast, will remain intact, albeit inevitably slower.

PHANTOM 会将这些块视为最大的 5-*cluster* 的一部分，并且它们将在最终排序中排在诚实链的后半部分之前。相比之下，KNIGHT 不容易被误导——它将认识到当 k=0 时就足以覆盖大多数 DAG。

同样的直觉适用于任何当前网络延迟比最坏情况（敌对）延迟小的情况。

此外，如果由于某些异常情况而导致网络遭受过多的延迟，以至于违反了 PHANTOM 的延迟限制，则可能无法确认交易。

相反，即使速度不可避免地缓慢，KNIGHT 的仍然能保持相关的操作。

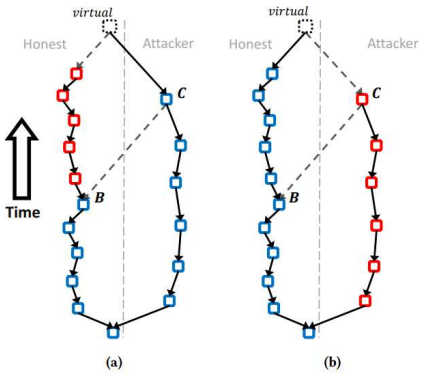

Figure 4: 4a shows a successful freeloading scheme against PHANTOM with $k = 5$. The largest 5-*cluster* contains (also) attacker blocks which were withheld till now, and excludes honest blocks which were mined correctly above B and published immediately.

4b demonstrates the failure of this scheme against KNIGHT. The protocol recognizes that the largest $k = 0$ suffices to cover a majority of the DAG, selects the entire 0-*cluster* (=chain) of B, and excludes the attack blocks.

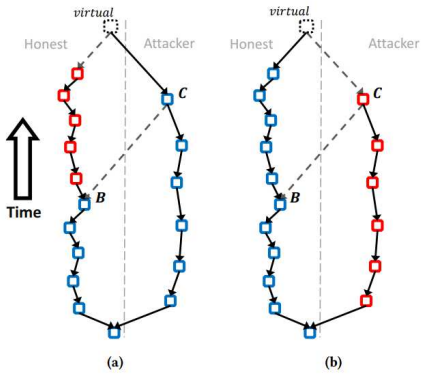

图 4： 4a 展示了一个对 PHANTOM 进行免费乘车攻击的成功方案，其中$k = 5$。

最大的5-*cluster* 包含攻击者块，这些块一直被扣留，同时取代了那些正确地在B上面挖掘并立即发布的诚实块。

4b 演示了这种方案在 KNIGHT 上的失败。该协议认识到最大的$k = 0$足以覆盖大多数 DAG，选择B的整个0-*cluster*（即单链），并排除攻击块。

攻击者可以选择在 DAG 中放置自己的恶意区块，但是这些恶意区块可能与诚实区块混合在一起，从而使得它们能够被 PHANTOM 协议确认。

Algorithm 2 KNIGHT DAG ordering algorithm

Input: $G - a\ block\ DAG$

Output: Selected tip of G, Ordering over G's blocks

1. **function** $O_{RDER} - DAG(G)$
2. **if** G is $\{genesis\}$ **then**
3. return $genesis, [genesis]$
4. **for** $B \in tips(G)$ **do**
5. $chain\text{-}parent(B)$, $order_B \leftarrow O_{RDER}\text{-}DAG(past(B))$
6. $\mathcal{P} \leftarrow tips(G)$
7. **while** $|\mathcal{P}| > 1$ **do**
8. $g \leftarrow$ latest common chain ancestor of all $B \in \mathcal{P}$
9. Partition $\mathcal{P}$ into maximal disjoint sets $\mathcal{P}_1,\cdots,\mathcal{P}_n \subset \mathcal{P}\ s.t.$ latest common chain ancestor of $\mathcal{P}_i$ is in $future(g)$
10. **for** $\mathcal{P}_i \in \{\mathcal{P}_1,\cdots,\mathcal{P}_n\}$ **do**
11. $rank_i \leftarrow C_{ALCULATE}\text{-}R_{ANK}(\mathcal{P}_i, future(g))$
12. $rank_{G,g} \leftarrow min_{i\in\{1,\cdots,n\}} rank_i$
13. $\mathcal{P} \leftarrow T_{IE}\text{-}B_{REAKING}(future(g), \{\mathcal{P}_i : rank_i = rank_{G,g}\})$
14. $\mathcal{p} \leftarrow the\ single\ element\ in\ \mathcal{P}$
15. **return** $\mathcal{p}, [order_{\mathcal{p}} || \mathcal{p} || anticone(\mathcal{p})]$

▷ operator || is sequence concatenation; $anticone(\mathcal{p})$ is iterated in hash-based bottom-up topological order

算法 2 KNIGHT DAG 排序算法

输入：G（一个 DAG）。

输出：选出 G 中的一个 tip, 对 G 中所有的区块进行排序。

在进一步阐释该算法的思想时，我们必须先要了解如下函数的意义：

$$C_{ALCULATE}\text{-}R_{ANK} \quad (1)$$

$$T_{IE}\text{-}B_{REAKING} \quad (2)$$

$$K\text{-}C_{OLOURING} \quad (3)$$

$$UMC\text{-}V_{OTING} \quad (4)$$

其中（1）、（2）位于算法二，（3）、（4）位于算法二的子函数中，即算法三、四、五、六。

当然算法 2 的一些必要先验知识需要说明：

- $genesis$：初始块，也称为创世块。
- latest common chain ancestor：最近公共祖先。
- $\mathcal{P}_1,\cdots,\mathcal{P}_n$：集合$\mathcal{P}$按照一定的规则分成多个不相交的子集$\mathcal{P}_1,\cdots,\mathcal{P}_n \subset \mathcal{P}$，使得每个子集$\mathcal{P}_i$中的所有块的最近公共链祖先都在$future(g)$上
- operator || …… topological order：运算符||表示序列拼接；$anticone(\mathcal{p})$指的是以哈希为基础的自下而上的拓扑排序。

下面，我们将开始相关函数的解释工作。

请留意，DK 协议的主算法已经超出了译者的知识水平，如果各位读者在计算机技术、图论方面颇有建树，欢迎指正。

*此部分的补充说明仍然用**绿色字体**标注，**但这并不代表说明是准确的**。*

如果读者觉得这部分内容很有挑战性，可以直接跳转到第 38 页——回顾 KNIGHT 算法的各部分。

请放心，即使不在意算法实现，也不会对本文的掌握产生较大影响。

Algorithm 3 Rank calculation procedure

Input: G – a block DAG, $\mathcal{P}$ – a set of blocks in G (typically $P \subset tips(G)$)

Output: The rank of $\mathcal{P}$ in G

1. **function** $C_{ALCULATE}$-$R_{ANK}(\mathcal{P}, G)$
2. **for** $k = 0,1 \ldots \infty$ **do**
3. **for** $r \in reps_G(\mathcal{P})$ **do**
4. $C_k(r), \_ \leftarrow K$-$C_{OLOURING}(r, G, k, \boldsymbol{false})$
5. **if** UMC-$V_{OTING}(G \setminus future(r), C_k(r), g(k)) > 0$ **then**
6. **return** k

算法 3 排名计算程序

输入：G（DAG），$\mathcal{P}$（G 中的一组块，更一般来说是$tips(G)$中的一组块）；

输出：$\mathcal{P}$在 G 中的排名；

排名：指的是$\mathcal{P}$中所有块的高度的最大值，也就是这些块中最长的链的长度。

算法思想：

从 0 到 k 进行枚举，紧接着对于每个包含$\mathcal{P}$中所有块的代表 r，进行一次K-$C_{OLOURING}$操作，将 r 以及 r 的所有未来子孙节点染成 k 种颜色，得到一个颜色方案$C_k(r)$。

$reps_G(\mathcal{P})$：$\mathcal{P}$的代表集合，包含所有$\mathcal{P}$中块的最小子集合。注意是包含集合的集合

现在开始验证该颜色方案是否有效，按照如下流程：

下述时间概念是针对$\mathcal{P}$来说的

1. 找到未来未着色的最小块集合$\mathcal{F}$，而且$\mathcal{F}$中至少要有一个块没有被染色。
2. 找到过去已着色的最大块集合$\mathcal{P}'$，使得在$\mathcal{P}'$中每个块的颜色都与代表集合的颜色相同。

对于$\mathcal{P}'$中的每个块，我们可以找到所有已着色的块与之存在父子关系的最大块，然后将这些块的颜色全部设为相同的颜色，这样就可以确定其他未着色块的颜色了。

3. 找到未来未着色的最大块集合$\mathcal{R}$，使得在$\mathcal{R}$中每个块的颜色都与代表集合的颜色相同，同时排除$\mathcal{F}$中的块。

对于 R 中的每个块，我们可以将其父节点着色为代表集合的颜色，然后再递归着色其所有子节点，直到所有未着色块都被着色。

最大块集合$\mathcal{P}'$和最大块集合 R 都是为了确定其他未着色块的颜色，但是它们的作用分别是在已着色块中找到一个与代表集合颜色相同的最大块集合，和在未着色块中找到一个与代表集合颜色相同的最大块集合，并且排除 F 中的块。

如果$\mathcal{F} \cup \mathcal{P}' \cup \mathcal{R}$中存在一个块与另一个块具有相同的颜色，则颜色方案无效，需要继续计算下一个$C_k(r)$。否则，颜色方案是合格的，算法返回该方案的迭代次数 k 作为块集合的排名。

Algorithm 4 Rank tie-breaking procedure

Input: G – a block DAG, $\mathcal{P}_1, \ldots, \mathcal{P}_m \subset tips(G)$

Output: A set of tips $\mathcal{P}_i$ wining the tie-breaking

1. **function** $T_{IE} - B_{REAKING}(G, \mathcal{P}_1, \ldots, \mathcal{P}_m)$
2. $k \leftarrow$ the mutual rank of $\mathcal{P}_1, \ldots, \mathcal{P}_m$ in G
3. $F, \_ \leftarrow K\text{-}C_{OLOURING}(virtual(G), G, g(k), \boldsymbol{true})$
4. **for** $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ **do**
5. **for** $k' \in \{\lfloor k/2 \rfloor, \ldots, k\}$ **do**
6. $\_, chain_{i,k'} \leftarrow K\text{-}C_{OLOURING}(virtual(G), G, k', \boldsymbol{false})$ conditioned[4] on $virtual(G)$ agreeing with $\mathcal{P}_i$
7. $C_i \leftarrow \cup_{k'}\{B \in G : anticone(B) \cap chain_{i,k'} > k'\}$
8. $j \leftarrow argmin_{i \in 1,\ldots,m} max\{\mathcal{F} \cap C_i\}$
9. **return** $\mathcal{P}_j$

如果在$\mathcal{F} \cup \mathcal{P}' \cup \mathcal{R}$中存在一个块与另一个块具有相同的颜色，则说明这个颜色方案无法给这两个块分配不同的颜色，即存在冲突。

由于这个颜色方案是经过精心选择的，且在$\mathcal{F} \cup \mathcal{P}' \cup \mathcal{R}$中已经尽可能地使用了不同的颜色，因此如果存在冲突，则说明这个颜色方案不可能满足要求，也就是无效的。

算法 4 排名打破平局程序

输入：G（一个 DAG），$\mathcal{P}_1, \ldots, \mathcal{P}_m$（并行区块）

输出：从并行区块$\mathcal{P}_1, \ldots, \mathcal{P}_m$中选出最终的区块$\mathcal{P}_i$

当存在多个节点需要染同一种颜色时，算法将根据它们的排名信息，选择最优的节点进行染色，从而避免了节点间颜色冲突的问题。

$tips(G)$表示入度为 0 的块集合

算法思想：

确定所有给定$\mathcal{P}_1, \ldots, \mathcal{P}_m$之间的相对顺序，称为它们的“相互排名”。

排序方法：以$\mathcal{P}_i$到 DAG 中根节点的距离为判断标准，按照距离递增的顺序进行排序。

使用$K\text{-}C_{OLOURING}$函数对$virtual(G)$进行染色，并确定所有节点的排名 k。

$K\text{-}C_{OLOURING}$函数返回的染色信息中包含了每个节点的颜色，颜色的值就是节点在 DAG 中的排名 k。

这里的排名是指节点在 DAG 中的拓扑排序编号，即第几个被遍历到的节点。

此处的$free\text{-}search$参数需要设置为$true$，算法能够自由地选择颜色，而不受参数 k 的限制。这样可以增加算法的搜索空间，从而有可能找到更优的解决方案。当然，这可能会导致算法的搜索时间更长。

对于给定的$\mathcal{P}_i$，算法会在一系列的深度k'下，使用$K\text{-}C_{OLOURING}$算法计算出$virtual(G)$中$\mathcal{P}_i$的所有祖先的颜色以及它们所在的最长链。

$K\text{-}C_{OLOURING}$的$free\text{-}search$参数需要设置为$false$。这能保证算法只从当前可用的 k-颜色集合中选择颜色，并且能够找到不超过 k-颜色的解决方案。

现在针对每个 $\mathcal{P}_i$ 我们可以计算 C_i，对于之前计算出来的最长链，C_i 包含了所有与 $\mathcal{P}_i$ 不相交的节点，但与该最长链有交的节点。

最后，选择具有最大 $\mathcal{F} \cap C_i$ 交集的 $\mathcal{P}_i$ 并返回。

如果有多个 $\mathcal{P}_i$ 满足条件，那就根据哈希值来决策。

Algorithm 5 k-colouring algorithm

Input: G – a block DAG, C – a block in G, k – a non-negative integer, $free\text{-}search$ – a Boolean indicating if the search can maximize freely

Output: k-colouring of $past_G(C)$, k-chain of $past_G(C)$

1. **function** $K\text{-}C_{OLOURING}(C, G, k, free\text{-}search)$
2. **if** $past_G(C) = \emptyset$ **then**
3. **return** $\emptyset, \emptyset$
4. $\mathcal{P} \leftarrow \emptyset$
5. **for** $B \in parents(C)$ **do**
6. **if** B agrees with C **then**
7. $blues_B, chain_B \leftarrow K\text{-}C_{OLOURING}(B, past(B) \cap G, k, free\text{-}search)$
8. $\mathcal{P} \leftarrow \mathcal{P} \cup B$
9. **else if** $free\text{-}search$ **OR** $k > rank_G(C)$ **then**
10. $blues_B, chain_B \leftarrow K\text{-}C_{OLOURING}(B, past(B) \cap G, k, \boldsymbol{true})$
11. $\mathcal{P} \leftarrow \mathcal{P} \cup B$
12. $B_{max} \leftarrow argmax\{|bluesB|: B \in P\}$ (break ties according to hash)
13. $blues_G, chain_G \leftarrow blues_{B_{max}} \cup \{B_{max}\}, chain_{B_{max}} \cup \{B_{max}\}$
14. **for** $B \in anticone(B_{max}, G)$ **do** in hash-based topological ordering
15. **if** $|chain_G \cap anticone(B)| \leq k$ **AND** $blues_G \cap anticone(B_{max}) < k$ **then**
16. $blues_G \leftarrow blues_G \cup \{B\}$
17. **return** $blues_G, chain_G$

算法 5 k-colouring 算法

输入：G（一个 DAG）；C（G 中的一个块）；k（非负整数）；$free\text{-}search$（布尔变量，决定搜索是否可以自由最大化）

输出：返回 $past_G(C)$ 的 k-colouring 和 k-chain

名词解释：

k-colouring：一种染色方案，用于标记 DAG 中的节点，要求相邻的节点颜色不同；

k-chain：一个长度为 k 的链，即 k 个节点依次相邻的路径，用于记录已经染色的节点，以确保新染色的节点与已经染色的节点颜色不同；

算法思想：

如果 $past_G(C) = \emptyset$，则直接返回空结果。

否则，从 $parents(C)$ 中筛选出与 C 相同的块，对这些块进行染色，并将它们加入 $\mathcal{P}$ 集合中。

集合 $\mathcal{P}$ 是临时变量，起中转作用

如果参数 $free\text{-}search$ 为真或者当前节点 C 的排名小于 k，则在当前节点 C 的所有父节点中寻找可以染色的节点，并将它们加入 $\mathcal{P}$ 集合中。

从 $\mathcal{P}$ 集合中选择一个最大的块 B_{max}，并将 B_{max} 添加到 $blues_G$ 和 $chain_G$ 中。如果有多个块大小相同，则按照哈希值进行比较。

也就是函数 $argmax\{\}$ 的作用

遍历 B，$B \in anticone(B_{max}, G)$，并按照哈希基于拓扑排序的顺序处理每个块 B，如果 B 满足以下两个条件，那就把它添加到 $blues_G$ 中：

- $|chain_G \cap anticone(B)| \leq k$

  $anticone(B)$ 中的已选择块数不超过 k
- $blues_G \cap anticone(B_{max}) < k$

$blues_G$ 中已选择的块数不超过 k 它的作用是尝试将尽可能多的块添加到颜色集合 $blues_G$ 中，以便在不超过 k 个颜色的限制下对 DAG 进行着色。

返回$blues_G$和$chain_G$，即k-colouring 和 k-chain

参数 k 的含义：

用于控制染色的颜色数量，限制每个块可以染色的颜色数量，避免相邻的节点颜色相同

用于控制染色的顺序，限制染色过程中已经染过色的节点与即将染色的节点的关系，避免相邻节点颜色相同

k-colouring 是染色方案，作用于所有的节点；而 k-chain 是记录已经染色的节点，作用范围只限于染色过程中的局部节点。

参数 free-search 的含义：

如果 free-search 参数为 true，则算法可以自由地选择颜色，而不受参数 k 的限制。

这意味着算法将尝试选择与它当前可用的颜色不同的任何颜色。这种自由搜索可以增加算法的搜索空间，从而可能找到更优的解决方案。

如果 free-search 参数为 false,则算法只能从当前可用的 k-颜色集合中选择颜色。这将限制算法的搜索空间，但也可以保证算法的解决方案不会超过参数 k 的限制。此时算法可能无法找到全局最优解，但是它可以保证找到一个不超过 k-颜色的解决方案。

因此，free-search 参数提供了一种权衡搜索空间和解决方案质量的方法。在需要最优解时，可以将 free-search 参数设置为 false，以保证算法找到的解决方案不超过 k-颜色限制。在需要探索更大的搜索空间时，可以将 free-search 参数设置为 true，以便算法可以自由搜索，并可能找到更优的解决方案。

Algorithm 6 UMC cascade voting procedure

Input: G – a block DAG, $U \subseteq G$ (typically a k-colouring), d – a non-negative integer representing the deficit threshold

Output: The voting result $vote \in \{-1,1\}\ of\ U \subseteq G$

1. **function** UMC-$V_{OTING}(G, U, d)$
2. $\mathcal{v} \leftarrow \sum_{B \in U} UMC\text{-}V_{OTING}(future(B), U \cap future(B), d)$
3. **return** $sign(\mathcal{v} - |G \setminus U| + d)$

▷ $sign(x) := \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$

算法 6 UMC（无权重多数着色）投票程序

输入：G（一个 DAG），U（G 的一个子集，通常是k-colouring），d（表示赤字阈值的非负整数）

d：一个非负整数，用来控制投票结果的偏见程度。

参数 d 提供了一个平衡考虑 DAG 的整体结构和颜色集合 U 中块的分布的方法，可以控制投票算法对整个 DAG 和颜色集合 U 中块的分布的关注程度。

输出：返回投票的结果，返回值为 1 表示给定的染色方案合法，返回值为-1 表示给定的染色方案不合法。

算法思想：

以下流程均在递归中完成

- 对于每个块$B \in U$，程序计算$future(B)$中的多数颜色，并将其作为块 B 的投票结果。
- 将所有块的投票结果相加，得到颜色集合 U 的总投票结果$\mathcal{v}$。
- 通过$\mathcal{v} - |G \setminus U| + d$的结果我们便能确定该颜色方案是否合理。

$G \setminus U$：G 与 U 的差集，定义如下：

$$G \setminus U = \{x | x \in G, x \notin U\}$$

为什么是$G \setminus U$：程序需要计算与颜色集合 U 中的块在未来中相关联的其他块的数量，这可以通过计算与$G \setminus U$相关联的块的数量来实现。

这些相关联的块是指那些在未来中与颜色集合 U 中的块通过有向边相连的块。

因此，与$G \setminus U$相关联的块的数量可以用来计算投票结果所需的块数。

解释具体的投票过程较为复杂，略过

非常好！

现在我们已经了解了各个函数的基本作用，接下来我们终于可以对算法 2 做出总结。

Algorithm 2 KNIGHT DAG ordering algorithm

Input: $G - a\ block\ DAG$

Output: Selected tip of G, Ordering over G's blocks

1. **function** $O_{RDER} - DAG(G)$
2. **if** G is $\{genesis\}$ **then**
3. return $genesis, [genesis]$
4. **for** $B \in tips(G)$ **do**
5. $chain\text{-}parent(B)$, $order_B \leftarrow O_{RDER}\text{-}DAG(past(B))$
6. $\mathcal{P} \leftarrow tips(G)$
7. **while** $|\mathcal{P}| > 1$ **do**
8. $g \leftarrow$ latest common chain ancestor of all $B \in \mathcal{P}$
9. Partition $\mathcal{P}$ into maximal disjoint sets $\mathcal{P}_1,\cdots,\mathcal{P}_n \subset \mathcal{P}\ s.t.$ latest common chain ancestor of $\mathcal{P}_i$ is in $future(g)$
10. **for** $\mathcal{P}_i \in \{\mathcal{P}_1,\cdots,\mathcal{P}_n\}$ **do**
11. $rank_i \leftarrow C_{ALCULATE}\text{-}R_{ANK}(\mathcal{P}_i, future(g))$
12. $rank_{G,g} \leftarrow min_{i\in\{1,\cdots,n\}} rank_i$
13. $\mathcal{P} \leftarrow T_{IE}\text{-}B_{REAKING}(future(g), \{\mathcal{P}_i : rank_i = rank_{G,g}\})$
14. $\mathcal{p} \leftarrow the\ single\ element\ in\ \mathcal{P}$
15. **return** $\mathcal{p}, [order_{\mathcal{p}} || \mathcal{p} || anticone(\mathcal{p})]$

▷ operator || is sequence concatenation; $anticone(\mathcal{p})$ is iterated in hash-based bottom-up topological order

算法 2 KNIGHT DAG 排序算法

输入：G（一个 DAG）。

输出：选出 G 中的一个 tip, 对 G 中所有的区块进行排序。

tip：没有后继节点的节点，入度为 0。

算法思想：对一个 a block DAG 进行排序，同时选择一个 tip 用于 block DAG 中新创建的块的父节点。

算法逻辑：

首先检查 DAG 是否只包含$genesis$，如果是，则返回$genesis$作为唯一排序结果，算法结束。

如果 DAG 中有多个 tip，那么选择一个 tip，以该 tip 的$past(tip)$为根节点，递归执行算法 2，得到该子图的排序结果$order_B$和子图中所有 tip 的$past(tip)$所在链的连接形式 $chain\text{-}parent(B)$。

$past(B)$ 等价于 $parent(B)$

将 DAG 中所有 tip 存入集合 $\mathcal{P}$中。$\mathcal{P}$中的节点按照一定的顺序排列，但是在这个阶段，它们的相对顺序并没有确定。

重复执行以下步骤，直到集合 $\mathcal{P}$中只剩下一个节点：

1. 找到集合 $\mathcal{P}$中所有节点的最近公共祖先 g，也就是它们所在的链中最后一个公共节点。

对于一个 DAG 中的所有节点，它们都可以向上回溯到某个共同的祖先节点，因为 DAG 是无环的。

2. 将集合 $\mathcal{P}$按照最近公共祖先g划分为若干个相互独立的子集 $\mathcal{P}_1,\cdots,\mathcal{P}_n$。

请注意这些子集是并行的

3. 对于每个子集 $\mathcal{P}_i$，计算它们在 DAG 中的排序$rank_i$，这个排序可以使用 $C_{ALCULATE}\text{-}R_{ANK}$ 算法计算得出。
4. 对于所有子集 $\mathcal{P}_i$的排序结果$rank_i$，取其中的最小值作为 DAG 的排序结果$rank_{G,g}$。

假设有两个并列的 tip P 和 Q，它们的 rank 分别为 $rank_P$ 和 $rank_Q$，我们想要选择一个最佳的 tip。由于这两个 tip 是并列的，它们与其他区块的关系相同。如果

$rank_P < rank_Q$，那么 P 比 Q 更接近最近公共祖先，即 P 所在的链更长。

因此，P 更可能被其他区块选择作为它们的父区块。因此，选择 rank 值更小的 tip 可以使得区块链的生长更加健康，有利于整个系统的稳定性和安全性。

5. 针对所有满足$rank_i = rank_{G,\mathcal{g}}$条件的$\mathcal{P}_i$，使用$T_{IE}$-$B_{REAKING}$函数选出理想的$\mathcal{P}_i$，同时更新参数$\mathcal{P}$。

当集合$\mathcal{P}$中只剩下一个节点时，该节点就是 DAG 的排序结果。

最后将$\mathcal{P}$赋值给$\mathcal{g}$，$\mathcal{g}$代表 DAG 的选中顶点，即我们心念的 tip。

最后返回$\mathcal{g}$和$order_{\mathcal{g}}||\mathcal{g}||anticone(\mathcal{g})$，即最终的 tip 和 DAG 的排序结果

$order_{\mathcal{g}}||\mathcal{g}||anticone(\mathcal{g})$：将如下三个对象按照操作符 || 拼接在一起：

- *$order_{\mathcal{g}}$：以$\mathcal{g}$为根的子 DAG 中所有节点的拓扑排序结果；*
- *$\mathcal{g}$：选中的 tip；*
- *$anticone(\mathcal{g})$：$\mathcal{g}$的所有anticone块，以哈希为基础，自下而上的拓扑顺序排序；*

$anticone(\alpha, G)$：G 中与α无关的块，我们这样定义无关：

- *从$anticone(\alpha, G)$不能顺着箭头到α；*
- *从 α 不能顺着箭头到$anticone(\alpha, G)$*

接下来的我们将恢复说明部分所用颜色的含义

2.5 KNIGHT – Formal specification

Algorithm 2 is quite more involved than merely solving one optimization problem.

Below we will review and motivate each of its components, but first let us present the holy grail of our efforts, in Algorithm 2.

We begin with some necessary notation and definitions. Throughout the rest of the paper we use G to denote a $sub\text{-}DAG$ as well.

Definition 2. For a block B, $chain\text{-}parent(B)$ is a unique parent of B, set by KNIGHT's chain-selection rule (line 5 in Algorithm 2).

The chain of B is defined recursively by $chain(B) := (chain\text{-}parent(B), chain\text{-}parent(chain\text{-}parent(B)), \ldots, genesis)$.

Observe that $past(B)$ fully determines $chain - parent(B)$.

Definition 3. $For\ U \subseteq G, d \geq 0$, we say that U is a $d\text{-}UMC$ of G (Uniform Majority Coverage), $if\ \forall B \in U, future(B) \cap U + d \geq future(B) \cap (G \setminus U)$

Definition 4. A set of blocks $X \subset G$ is said to agree in G if their latest common chain ancestor is a chain-descendant of $g = genesis(G): \exists g' : g \in chain(g') \wedge \forall B \in X : g' \in chain(B)$.

2.5 KNIGHT——形式规范

算法 2 比仅仅解决一个优化问题要复杂得多。

在接下来，我们将回顾和解释其各个组成部分，但首先先让我们在算法 2 中展示我们努力的成果。

我们首先介绍一些必要的符号和定义。在本文的其余部分中，我们使用G来表示$sub\text{-}DAG$。

$sub\text{-}DAG$：DAG的子集

定义 2： 对于区块B，$chain\text{-}parent(B)$是由 KNIGHT 的链选择法则（算法 2 中的第五行）设置的唯一父级。

B 的链的递归定义如下：

$chain(B)$:

$= (chain\text{-}parent(B), chain\text{-}parent(chain\text{-}parent(B)), \ldots, genesis)$.

注意，$past(B)$完全确定了$chain\text{-}parent(B)$。

如果我们知道一个区块的所有过去区块的信息，我们就可以唯一确定它的链父块。

可以认为$chain\text{-}parent(B)$是$past(B)$的子集。

定义 3：对于给定的$U \subseteq G, d \geq 0$，我们称 U 是 G 的一个$d\text{-}UMC$（统一多数覆盖），如果 U 满足如下约束：

$\forall B \in U, future(B) \cap U + d \geq future(B) \cap (G \setminus U)$

$d\text{-}UMC$：这意味着在未来 d 个区块内，U 中的区块被多数节点所接受和确认。

定义 4：如果一组区块$X \subset G$在G中达成共识，则它们的最近共同链祖先是$g = genesis(G)$的链后代，我们这样定义$g = genesis(G)$：

$\exists g' : g \in chain(g') \wedge \forall B \in X : g' \in chain(B)$

这个公式的含义是：

- *g'是g的祖先块；*
- *对于X中的每一个块B，g'也必须*

Intuitively, two blocks agree in G if they agree on g's successor in the chain.

是 B 的祖先块，即g'在$chain(B)$:中出现；

直观地说，如果两个区块在G中达成一致，则它们必须在g的后继上达成共识。

g的后继：g是创世块，g的后继代表在g之后添加到区块链中的块。

Definition 5. For a set $X \subset tips(G)$ agreeing in G, the $setreps_G(X)$ (representatives) is defined by

$\{x \in \overline{past}(X) \setminus past(tips(G) \setminus X): x\ agree\ with\ X\}^2$

定义 5：对于在 G 中达成共识的块集合 $X \subset tips(G)$，我们这样定义其代表集合 $setreps_G(X)$：

$\{x \in \overline{past}(X) \setminus past(tips(G) \setminus X): x\ agree\ with\ X\}^2$

其中：

$\overline{past}(X)$：X 的历史祖先块的集合；

$past(tips(G) \setminus X)$：不在 X 中，G 中所有入度为 0 的历史祖先块的集合；

$x\ agree\ with\ X$：x 在 X 中达成共识；

因此我们可以这样理解，$setreps_G(X)$中的所有块满足如下条件：

- *在 X 中达成共识；*
- *是 X 的祖先块；*
- *与$past(tips(G) \setminus X)$不相交；*

Definition 6. For a block B and $chain\text{-}ancestor g \in chain(B)\ s.t. \exists p1, p2 \in parents(B)$ which do not agree over $future(g)$, $rank_{future(g)}(B)$ is defined to be the rank calculated by KNIGHT ordering when recursively executing $O_{RDER}\text{-}DAG(past(B))$ and for the iteration of the While loop where g was found (line 12 in Algorithm 2).

定义 6：对于一个块 B 和链祖先$g \in chain(B)$，它们满足如下性质：

$\exists p1, p2 \in parents(B)$，$p1, p2$ 无法在$future(g)$中达成共识，我们定义参数 $rank_{future(g)}(B)$，它表示通过执行递归执行 $O_{RDER}\text{-}DAG(past(B))$，利用 KNIGHT 排序算法计算出的 rank 值，而且此时 KNIGHT 排序算法在找到g时停止迭代（在算法 2 的第 12 行处）。

For non-negative integer k, $g(k)=o(k)$ is a function returning a non-negative integer, used throughout the protocol. We set $g(k):=\lfloor\sqrt{k}\rfloor$.

对于非负整数 k，$g(k)=o(k)$是一个返回非负整数的函数，我们将在整个协议中使用它，另外设定$g(k):=\lfloor\sqrt{k}\rfloor$。

When applied to sets of blocks, max and min operators represent topology relations.

如果要应用于块集合，max 和 min 运算符表示拓扑关系。

That is, if $B=\max G$ then $future(B) \cap G=\emptyset$, and likewise if $B=\max G$ then $past(B) \cap G=\emptyset$.

也就是说，如果 $B=\max G$，则 $future(B) \cap G=\emptyset$，同理，如果 $B=\max G$则有$past(B) \cap G=\emptyset$。

2.6 Run time

The algorithms specified in the previous article terminate in polynomial time:

Proposition 1. Algorithm 2 terminates in polynomial time in $|G|$, and returns a tip and an ordering of G.

P_{ROOF}. Observe the following facts:

- The while loop in line 7 decreases the size of $\mathcal{P}$ at each iteration.
- Following line 13 it remains that $\mathcal{P} \neq \emptyset$, thus, after the loop, $\mathcal{P} = 1$ (line 14);thus, the return argument is not null, and is an element in $\mathcal{P}$.
- The overall recursion (line 5) terminates since $\forall B \in tips(G), past(B) \subsetneq G$.
- The procedure $C_{ALCULATE}$-R_{ANK} terminates in polynomial time, since the output of K-$C_{OLOURING}(C, G, k, \cdot)$, for any $block\ C \in G$, returns a k-UMC, for $k = |G|$, since all blocks in G belong to its largest $|G|$-$cluster$ (there are, obviously, much tighter arguments).

In a future version of this paper, we will present an equivalent specification that takes as input two blocks and returns their respective ordering.

This procedure is useful for certain types of clients (e.g., what are known as "liteclients"), and can be shown to terminate within a constant (in time) number of steps, concretely, in $O(D \cdot \lambda)^2$ steps.

2.6 运行时间

前文中指定的算法能在多项式时间内完成：

指算法 2。

命题 1：算法 2 在$|G|$的多项式时间内完成，并返回 G 的一个 tip 和排序。

证明：观察如下事实：

- 在第 7 行的 while 循环中，每次迭代都会减小 $\mathcal{P}$ 的大小。

$\mathcal{P}$ 的大小是指集合$\mathcal{P}$元素的个数大小。

- 接着第 13 行之后，$\mathcal{P}$ 仍然不为空，因此在循环之后，$\mathcal{P} = 1$ (第 14 行)；因此返回参数不为空，并且是 $\mathcal{P}$ 中的一个元素。
- 当满足$\forall B \in tips(G), past(B) \subsetneq G$的条件时，整体递归(第 5 行)终止。

此处$\subsetneq$表示真子集。

原文是$\subseteq$，实际上$\subsetneq$更为常见。？

- 只要 G 中的所有块都属于其最大的$|G|$-$cluster$，那么对于任意的块$C \in G$，K-$C_{OLOURING}(C, G, k, \cdot)$都会返回一个$k$-$UMC$。

  出于以上原因，我们可以说函数$C_{ALCULATE}$-R_{ANK}能够在多项式时间内完成（显然，此后还有更严密的论证）。

在本文的 2.0 版本中，我们将会提供一个等价的规范，这个版本能同时输入两个块并返回它们的排序。

等价的规范：功能相同，但实现方式可能不同。

这个过程对于某些类型的客户端(例如所谓的"轻量级客户端")很有用，而且可以在常数步骤内终止，具体而言，是在$O(D \cdot \lambda)^2$内。

2.7 Reviewing the components of KNIGHT

The algorithms presented above are admittedly involved.

In this subsection we review their core components.

The full version, which will appear online, includes a line by line explanation of the three procedures.

2.7.1 *Greedy maximization.* To cope with the intractable nature of finding the maximal k-cluster, we take an approach similar to where the NP-hard version was replaced with a greedy procedure, called therein GHOSTDAG.

We thus limit the search to extensions of k-clusters of the previous tips of the DAG ($K\text{-}C_{OLOURING}$, line 7).

2.7.2 *Revisiting the Majority condition.* Instead of requiring that the selected k-cluster covers a majority of the DAG (equiv., the majority of the future set of the genesis block), we check whether it covers a majority of the future set of each of its member blocks, including genesis.

Blocks whose future the cluster fails to cover by majority are cast out as outliers, and the procedure counts them outside the cluster.

2.7 回顾 KNIGHT 算法的各部分

上面介绍的算法确实很复杂。

在这一小节中，我们将回顾它们的核心部分。

完整版本将在网上发布，其中包括三个程序的逐行解释。

2.7.1 *贪心最大化。*为了解决如何找到最大的k-cluster 这一问题，我们采取了类似于 GHOSTDAG 中使用的方法——贪心。

前文提过，找到最大的k-cluster 是一个 NP-hard 问题。

因此，我们将搜索集中于扩展自 DAG 的前一级的k-cluster 上（$K\text{-}C_{OLOURING}$，第七行）。

在寻找最大的k-cluster 时，算法会先从 DAG 的第一层寻找包含 genesis 的k-cluster，然后拓展到第二层继续寻找更大的k-cluster，直到无法找到更大的k-cluster 为止。简单说，算法不会在整个 DAG 中寻找最大的k-cluster。

这里的层次关系是指相连，比如第二层包含了与第一层（即 genesis）直接相连的区块。

2.7.2 *重新审视主要条件。*与其要求所选的k-cluster 能够覆盖 DAG 的大多数（也就是未来的$genesis$的大多数），我们只需要检查k-cluster 是否覆盖了每个成员块的未来的大多数，包括$genesis$。

大多数：占比超过 50%。

一般来说，2.7.2 中的翻译内容中的"大多数"是指当前区块所在的k-cluster 的大多数区块。读者也可以简单理解为 51%算力。

对于某个成员块，如果当前的k-cluster 没有覆盖它的未来的大多数，我们视其为异常，并把它排除在当前的k-cluster 之外。

如果覆盖就继续遍历下一个成员块，这样就可以排除不包含在当前k-cluster 中的块，从而选择了那些能够覆盖大多数成员块的未来块的k-cluster 作为最后的结果。

The procedure induces a cascading majority vote from recent blocks down to the genesis block, and the latter's vote dictates whether the majority cover is satisfactory.

从最近的块到创世块，程序引入联级多数投票算法，后者的投票决定了此次大多数覆盖是否令人满意。

每个成员块都投票以表明其未来是否被当前选择的k-cluster 覆盖，如果被覆盖，就投赞成票；否则，就投反对票。如果一个成员块的未来被当前选择的k-cluster 所覆盖的块的数量超过了半数，那么这个成员块就投赞成票；否则，就投反对票。

最终，我们对所有成员块的投票结果进行统计，并将它们传递到创世块。

如果创世块投票结果表明当前选择的k-cluster 覆盖了大多数成员块的未来，那么这个k-cluster 就被认为是满足大多数覆盖要求的。

很显然，这是一个递归过程。

By extending the majority coverage requirement from genesis to any (non-outlier) block in the k-cluster, we recover the "Markovian" nature of the coverage property: Any new honest block "resets" the process by posing an additional challenge to the attacker, namely, to cover the majority of this new block.

通过将大多数覆盖要求从创世块拓展到k-cluster 中的任何非异常块，可以使区块链的覆盖属性具有"Markovian"性质：每当一个新的诚实区块被挖掘出来后，就会对攻击者提出额外的挑战——覆盖新块的大多数。

每次挖出一个新的诚实区块都会重置攻击者的攻击过程，攻击者需要重新控制大多数算力才能攻击区块链。

Markovian：马尔科夫链，它有如下性质：某个事件的概率只与当前状态有关，与之前的历史状态无关。

本文中，如果一个新的诚实块被添加到了区块链中，那么攻击者需要重新攻击这个新块的大多数来掌控区块链，这就是一个"Markovian"过程。

攻击者需要面对新的状态，即新加入的块，如果试图掌控区块链，攻击者很快就会发现之前所做的努力和之前的状态已经不再重要，也不会对当前状态产

生影响。

Indeed, honest miners possess a majority of the hashrate, and blocks of honest miners are referenced by their honest counterparts after at most D seconds, after which honest blocks are expected to win the block race with high probability.

实际上，诚实的矿工拥有大多数的算力资源，而且诚实矿工的区块能够很快被其他的诚实的同行引用，这个过程不会超过 D 秒，因此，这些诚实的区块有极大可能被确认。

To account for these D seconds, we allow the cluster to cover almost a majority—a deficit of $g(k)$ blocks is permitted (line 5 in $C_{ALCULATE}$-R_{ANK}).

为了保证能在 D 秒内完成，我们允许有一些块的不足（即$g(k)$，在$C_{ALCULATE}$-R_{ANK}中的第五行），只要这个k-cluster 覆盖几乎大部分的区块即可。

2.7.3 *Decouple ordering from colouring.* Recall that the partially synchronous model allows for an attacker to control the propagation time of any message in the network up to some (unknown) bound D.

2.7.3 *将排序与着色分离。*回想一下，部分同步模型允许攻击者控制网络中任何消息的传播时间，直到某个（未知的）上限 D。

部分同步模型中，网络中的消息在传输时可能会遇到不确定的延迟，也就是说，消息可能在不同的时间被不同的节点接收到。

在这种情况下，攻击者可以通过控制消息的发送时间来影响消息的传播时间，使得某些节点接收到消息的时间比其他节点更早或更晚。

具体来说，攻击者可以将消息发送时间延迟一段时间，使得消息到达某些节点时已经过时，从而影响这些节点的决策。这个延迟的上限就是文中提到的未知界限D。

It follows that the largest k-cluster, for $k \approx 2 \cdot D \cdot \lambda$ (which bounds the expected size of an honest block's natural anticone) is expected to satisfy the majority coverage property (k-UMC).

因此，最大的k-cluster，对于$k \approx 2 \cdot D \cdot \lambda$（它限制了诚实块的自然 anticone 的预期大小），k 能满足大多数覆盖的性质。

我们重新了解一下 anticone：

在区块链中，每个区块都是由一组已经被确认的交易组成的，这些交易被打包进区块中并由矿工进行验证和添加到区块链中。

每个区块都可以被看作是一个有向无环图（DAG）中的一个节点，而交易则是 DAG 中的边（当然直接用块来表示也没问题）。

当存在多个区块被挖掘时，这些区块之间会形成一个 anticone，其中每个区块都没有被其他区块确认。这与之前的解释并不矛盾。

需要留意的是 anticone 与并行块是两个东西。

本文中，我们假设 anticone 的大小是 $k \approx 2 \cdot D \cdot \lambda$，其中 D 时最大的消息延迟时间，λ 是交易的平均生成速率，此时最大的 k-cluster 能覆盖大多数的诚实节点，从而保证 DAG 的收敛性。

One would expect, therefore, that the following procedure would suffice to secure the ordering: Find the minimal k for which the largest k-cluster satisfy the k-UMC property, and order the DAG according to that cluster.

因此，人们可能会认为以下过程足以维持排序的安全性：找到最小的k，使得最大的k-cluster 满足k-UMC 属性，并根据该簇对 DAG 进行排序。

Albeit, this approach would undermine the stability of the ordering: If the network's latency changes, spontaneously or maliciously, from $d \ll D$ to D the ordering of the DAG would change retroactively from the largest $k(d)\text{-}cluster$ to the largest $k(D)\text{-}cluster$, undermining the convergence guarantee.

可惜，这种方法无法保证排序的稳定性：如果网络的延迟从$d \ll D$突然或者恶意地被更改为D，则 DAG 的排序将从最大$k(d)\text{-}cluster$的排序更改为最大$k(D)\text{-}cluster$的排序，从而破坏排序收敛保证。

如果我们坚持使用上述算法，无法保证曾经使用的k-cluster 是最大的。

To cope with this challenge, we (re)introduce the notion of a main chain, and order the DAG according to this chain.

为了解决这个麻烦，我们（重新）引入主链的概念，并根据这条链对 DAG 进行排序。

We show this chain to be robust even under changes of delays, rendering the ordering robust.

我们证明了这条链即使在延迟变化的情况下也是稳健的，这使得排序也是稳健的。

robust 一般译作鲁棒性。

The chain is formed as follows: Each block picks as its chain predecessor the block which minimizes its own k, or more accurately, its rank ($O_{ODER}\text{-}DAG$, line 11).

主链按照如下方式生成：每个区块选择其链前驱，这个前驱是最小化其自身的 k，更准确地说，是它的排名（$O_{ODER}\text{-}DAG$中的第 11 行）。

这个前驱是一个块，我们要求这个块的 k 最小（即它对应的 anticone 最小）。

这意味着每个区块都会选择一个相对较小的前驱，以便能够更快地在链中达成共识。

That is, we utilize the optimization problem of KNIGHT to select the chain-predecessor of each block rather than to order the entire historical DAG.

也就是说，我们利用 KNIGHT 的优化问题来选择每个块的链前驱，而不是对整个历史 DAG 进行排序来确认区块的顺序。

This decoupling allows the chain to "represent" different k's along its growth, which correspond to the effective latency at the time.

这种解耦允许链在其增长过程中“表示”不同的k，这些k对应于当时的有效延迟。

解耦：将两个或多个系统、过程或功能之间的联系减少到最小限度，以便它们可以独立地运作，不会互相影响或干扰。

文中提到的解耦是指将选择每个块的链前驱这一过程与整个历史 DAG 的排序过

程分开进行，使得每个块可以独立地代表自身的延迟情况，并且可以更灵活地响应网络中可能出现的变化。

更详细地说，通过使用 KNIGHT 算法来选择每个块的链前驱，使得每个块都可以代表不同的k值，这些k值对应于该块的延迟。

这种解耦的方式，使得整个链可以代表不同的延迟情况，并且每个块可以独立地代表其自身的延迟情况，而不受之前或之后的块的影响。

For example, if at chain-level 200 the attacker exposed a side-DAG that required increasing k from 5 to 7, the ordering of past blocks would still be dictated by the chain below level 198, say.

例如，如果在链级别 200 处，攻击者暴露了一个侧面 DAG 并将k从 5 增加到 7，那么在链级别 200 之前（例如链级别低于 198），块的排序仍然由原来的链控制，不受侧面 DAG 的影响。

暴露：指的是攻击者在网络中发起攻击并创建了一个侧面 DAG，从而破坏了原有的链顺序。

在基于 PoW 的区块链中，如果攻击者拥有足够的算力，他们可以创建一个分支的 DAG，从而影响整个区块链的顺序和稳定性。在这种情况下，攻击者暴露了侧面 DAG，即揭示了这种攻击行为，也就是在文中所说的"暴露"。

这意味着，即使在网络中出现了攻击，链的整体顺序不会被完全打乱，而是会在攻击点的位置之前保持原有的顺序。

This decoupling of cluster-selection from DAG-ordering spawns an intricate design space with different inter-dependencies between cluster-selection and chain-ordering.

将集群选择与 DAG 的排序分开来考虑会使得设计空间变得更加复杂，这是由于集群选择和链式排序之间存在不同的相互依赖关系。

设计空间：指在设计过程中可以采用的所有可能的设计方案和技术手段的集合，也可以理解为所有设计决策的潜在空间。

集群选择：指在区块链系统中，选取一组区块来处理新的交易或生成新的区块的过程。

链式排序：将 DAG 中的节点按照一定规则进行排序，以便确定最终的主链。

Interestingly, some natural candidates turn out to be insecure, erring either on over-stability (thereby compromising liveness) or on over-flexibility (compromising safety).

有意思的是，一些看似自然的候选方案被证明是不安全的，这些方案要么过于稳定（这样就不利于区块链活性），要么过于灵活（这样就不利于区块链安全性）。

We strike a balance between these two necessary objectives by allowing the cluster-selection to deviate from the chain-selection of lower-ranked blocks (K-$C_{OLOURING}$, line 9).

我们通过允许集群选择与较低排名块的链式选择有所偏离来平衡这两个必要的目标（K-$C_{OLOURING}$第九行）。

有所偏离是指在集群选择和链式选择之间引入一定程度的灵活性，使得集群选择可以与较低排名块的链式选择有所不同。

在传统的区块链系统中，集群选择和链式选择之间通常是严格耦合的，集群选择决定了链式选择的结果。

允许一定的偏离在一定程度上可以平衡区块链系统的稳定性和安全性，并且可以更好地适应不同的场景和需求。

2.7.4 *Tie-breaking for recovery.* Consider a temporary anomaly ("Poisson burst") in the block creation process which led to an abnormally high rank K.

After the network resumes normal operation, we would like to recover the normal rank, denoted $k^\star$, or otherwise liveness would be compromised (the waiting time for liveness depends on a non-diverging upper bound over the rank);

we thus must guarantee healthy growth of the $k^\star$-$cluster$, even when the current rank K is excessively high.

In this context the tie-breaking rule between two chain-tip candidates of the same rank turns out to be crucial.

2.7.4 *恢复时的打破平局。*考虑到在块创建过程中出现临时异常导致的异常高的等级 k（这个现象称为泊松突发）。

当网络恢复正常操作后，我们需要恢复正常等级，不妨称为$k^\star$，不然就会危害网络活跃性（等待区块链网络达到活跃的时间取决于排名的非发散上界）；

网络活跃性：通常指的是网络中交易被广播、确认和处理的速度和能力。

因此，即使当前排名 K 非常高，我们也必须保证 $k^\star$-$cluster$ 的健康增长。

在这种情况下，相同排名的两个 chain-tip 候选者之间的选择极为重要。

在一个区块链网络中，不同的节点可能会产生不同的区块，这些区块可能具有相同的排名。

当出现这种情况时，需要选择一个区块作为链的继承者。

A naïve rule would prefer the larger K-cluster, yet such a design would allow an attacker to keep the network at its current rank and prevent it from recovering towards $k^\star$.

一个简单的想法就是中意更大的K-cluster，但这样的策略将允许攻击者使网络保持当前的排名，从而阻止网络恢复到$k^\star$

如果选择规则过于简单，比如只优先选择具有更大的 K 簇的 chain-tip 作为继承者，那么这种简单的规则容易被攻击者利用来阻止网络恢复到正常状态。

比如说，一个攻击者不断产生大量的区块，使得当前的排名非常高，那么简单

的选择规则就会不断选择这些由攻击者产生的大K-cluster 的区块作为继承者，而不选择由其他节点产生的排名更低的区块。

这样，攻击者就可以控制网络，阻止它恢复到正常状态，从而破坏整个区块链网络的安全性和稳定性。

Instead, we identify the tip whose cluster utilized the excessive rank latest, and prefer its counterpart (T_{IE}-$B_{REAKING}$ algorithm, line 8).

相反，我们确定了一个 tip，它的 cluster 的等级是最高的，然后选中这个 tip 对应的 chain（T_{IE}-$B_{REAKING}$算法，第 8 行）。

The resulting chain-selection rule forces a tie-preserving attacker to compete on ranks much lower than the current one, and eventually to compete on the natural rank, $k^{\star}$.

由此产生的链选择规则将迫使那些 试图保持平局的攻击者 在比当前排名低得多的排名上竞争，最终在自然排名 $k^{\star}$上竞争。

tie-preserving attacker：译文中译作试图保持平局的攻击者。

其准确含义如下：攻击者试图通过恶意行为来保持分叉状态，从而让网络处于一个无法决定哪个分支是主链的状态。这种攻击者不会选择明确的分支，而是会尽可能地让多个分支保持相同的权重，以便他们能够继续进行攻击并从中获益。

本文中的防御策略可以强制让攻击者在比当前等级更低的等级上进行竞争，最终只有在自然等级 $k^{\star}$ 上才能获胜，从而使得攻击者难以保持分叉状态并攻击网络。

2.7.5 *Adaptiveness to long-term delay changes.* A partially synchronous protocol performs according to the actual (adversarial) latency, as discussed in Section 1.

2.7.5 *适应长期延迟变化。*部分同步协议的执行取决于实际（敌对）延迟，正如第 1 节所讨论的。

However, in the context of a consensus protocol that serves a continuous queue of transactions, the latency might change with time.

然而，在为连续交易队列提供服务的共识协议中，延迟可能会随时间变化。

例如，在网络流量较小时，交易的传输延迟可能会很低，而在网络流量高峰期，传输延迟可能会很高。

It would then be undesirable if the protocol performs according to the worst case historical latency rather than the recent latency in the network.

因此，如果协议单纯地按照历史最坏情况的延迟来执行，而不考虑网络实时的延迟，那么协议将变得难以接受。

We formalize this requirement in Section 3.

我们在第 3 节中用公式表达了这个要求。

To achieve this property, the protocol defines a conflict hierarchy, eliminates iteratively the losing candidates, and selects the final survival as the chain-predecessor.

为了实现适应长期延迟变化的属性，共识协议采用了一系列策略：

- 定义了一个冲突层次结构，用于确定交易之间的优先级和先后顺序；
- 逐步消除那些被认为是失败的候选项，直到只剩下一个幸存者；

根据以上策略，协议最后选出了一个“幸存者”，它将作为链的前身，即下一个区块的父区块。

This logic is implemented in the While loop in O_{RDER}-DAG (line 7).

这个逻辑可以在O_{RDER}-DAG（第 7 行）的中的 While 循环中找到。

2.7.6 *Representatives and monotonicity.* In theory, an attacker may attempt to artificially increase the rank of honest blocks by wasting part of her hashrate to mine blocks that agree with honest blocks but which do not belong to their k-cluster, where k is the current rank of honest blocks.

2.7.6 *代表和单调性。*理论上，攻击者可能会试图通过浪费一部分算力来挖掘与诚实块一致但不在它们的k-cluster 的块（其中 k 是诚实块的当前等级），从而人为地增加诚实块的排名。

排名是指一个区块在 DAG 图中所处的层数。

While this scheme can be shown (yet, at the expense of further complication of the analysis) to be overall suboptimal on her part, it does undermine a desired “monotonous” behaviour of the protocol.

虽然这种方案可以证明（当然，这需要进一步分析复杂性），但它确实破坏了协议所希望的“单调”行为。

“单调”行为：指的是区块链共识协议应该具有的单调性质，即在新的区块被添加到 DAG 图中时，协议选择的主链应该保持不变或者向前推进，而不会后退。

攻击者人为地增加诚实块的排名可能会导致协议选择的主链向后退，因此破坏了协议的单调行为。

Consider a DAG G with two tips B and C, and assume that B “won” and is G’s selected chain tip.

考虑一个 DAG G 中的两个 tip，B 和 C，假设 B“获胜”并成为 G 的选定 chain tip。

即下一个区块的父区块。

Consider the effect of adding to G a new block E, which references B only.

考虑将一个新块 E 添加到 G 中，该块仅包含对 B 的引用。

Since E acknowledges B but not C, one would expect the addition of E to only increase the chance of B to win over C, and definitely not to harm it.

由于 E 承认 B 而不承认 C，因此可以预料的是添加 E 只会增加 B 获胜的机会，而不会拖 B 的后腿。

Alas, if a set of disconnected E's are added to B's future in this manner, they may increase the rank of their part of the DAG, and in particular may flip the choice and lead to the chain going through C.

To recover the desired monotonous behaviour (thereby simplifying our security analysis, as a byproduct), we dictate that C competes with B even if B is no longer a tip of G(!).

Thus, to win the chain over E, C must enjoy a rank lower than E (the new tip) but also of all blocks in E's past (which are not in C's past), and B in particular; this exemplifies the role of the representative set (Definition 5) used in line 3 of Calculate-Rank.

但是，如果以这种方式向 B 的未来添加一组断开的 E，则它们可能会增加 DAG 某一部分的排名，并且特别需要注意的是，这样做可能会反转选择并导致主链通过 C。

断开：可以理解为 B 的一组 anticone 块，即不相连的块。

为了恢复所需的单调行为（作为补充说明，安全分析不会太详细），我们规定即使 B 不再是 G 的 tip，C 也要与 B 竞争。

因此，为了比 E 更赢得主链的“芳心”，C 必须满足比 E（新加入的 tip）更低的等级，而且还要低于 E 的所有过去块（这些块不在 C 的过去），特别是比 B 低。

这就是$C_{ALCULATE}$-R_{ANK}中第 3 行使用的代表集（定义 5）的作用。

3 MODEL AND FORMAL STATEMENT

We follow the prevalent models for a proof-of-work governed network and its extensions to the block DAG framework.

A network of nodes (or miners) is denoted $\mathcal{N}$, each node u maintaining a replica of the DAG observable to it G_t^u.

The set $\mathcal{H}$ denotes nodes that follow the mining protocol, which dictates that every new block references all tips of the DAG observable to its miner at its creation, and is broadcast by it immediately to the network.

The attacker deviates arbitrarily from the mining protocol, and can further accelerate or delay messages from or to honest nodes up to D_t seconds (D_t depends on time since network conditions and connectivity might change with time); we denote by D_{max}, or simply D, the maximal D_t across $t \in [0, \infty)$.

Importantly, $D = D_{max}$ is a function of the block size limit denoted $block\_size\_limit(KB)$, since large messages take longer to propagate.

For brevity, we ignore this parameter, and regard the block size as fixed.

We emphasize that $block\_size\_limit$ can be increased in the same manner than the block rate λ may be increased, as discussed in Subsection 5.1.

3 模型和正式声明

我们遵循如下流行模型：

- 网络基于工作量证明；
- 在 block DAG 框架中拓展；

节点（或矿工）的网络用$\mathcal{N}$表示，每个节点u维护其可观察到的 DAG 的一个副本G_t^u。

可观察：指节点u能够从其它节点获取到该网络的部分信息，并在本地存储中维护它所看到的网络状态，即G_t^u。

我们用集合$\mathcal{H}$表示遵循挖矿协议的节点，该协议规定每个新块在创建时引用其矿工的可观察 DAG 的所有 tips，并立即向网络中广播。

攻击者会随意偏离挖矿协议，并且可以进一步加速或延迟那些来自或发往诚实节点的消息，影响的最长时间为D_t秒（D_t并非固定，因为网络条件和连接可能会随时间改变）；我们用D_{max}或简单地用D来表示$t \in [0, \infty)$上的最大D_t。

D_t的更详细意义：某一时刻t下，攻击者可以最多延迟或加速消息的时间。

重要的是，$D = D_{max}$是一个关于块大小限制(下面统称$block_size_limit(KB)$)的函数，因为复杂的消息需要更长时间才能传播。

为了简洁起见，我们忽略了这个参数，并将块大小视为固定值。

即视区块搭载的信息量恒定。

我们强调，增加$block_size_limit$可以采取与增加块创建速率λ相同的方式来实现，这一点在 5.1 小节中有讨论。

The proof-of-work mechanism targets a certain block creation rate of λ blocks per second, kept (roughly) constant via a difficulty adjustment algorithm, similarly to Bitcoin.

工作量证明机制旨在以每秒λ个块的特定速率创建块，通过难度调整算法（类似于比特币）保持（大致）恒定。

We denote the proof-of-work protocol by $pow(\lambda)$.

我们用$pow(\lambda)$表示工作量证明协议

Block creation thus follows a Poisson process with parameter λ, and the next block in the network is created by an honest node with probability $1-\alpha_t$, for some (unknown, potentially dynamic) $0 \leq \alpha_t < \alpha$ (for $t \in [0,\infty)$).

块创建遵循参数为λ的泊松过程，并且网络中的下一个块由一个诚实节点以概率$1-\alpha_t$创建，其中$0 \leq \alpha_t < \alpha$（对于$t \in [0,\infty)$，α是未知的，可能是动态的）。

如果我们在一段连续的时间内考察泊松分布，我们便得到了泊松过程。

参数为λ的泊松过程：一类随机过程，其中λ是一个固定的正数，通常称为强度，如果给定时间区间$[t,t+\tau]$，则时间区间之中事件发生的数目随机变量$N(t+\tau)-N(t)$呈现泊松分布，其参数为$\lambda\tau$。

在$[t,t+\tau]$内发生事件的数目的概率分布为：

$$P[(N(t+\tau)-N(t))=k]=\frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!}$$

完整的定义很有意思，感兴趣的读者可自行查阅 wiki。

If this inequality is guaranteed to hold for some range $t \geq s$, We say that α is an $s\text{-}updated$ bound over the attacker's computational power; this definition is used below to emphasize a self-stabilizing property which allows to recover from "51% attacks".

如果这个不等式在某个$t \geq s$范围内得到保证，我们说α是攻击者计算能力的$s\text{-}updated$上限；这个定义用于强调一种自稳定特性，它允许从"51%攻击"中恢复过来。

在这个时间范围内，攻击者即使拥有超过α的计算能力，也无法改变诚实节点创建新区块的概率，因为这个概率已经被确定下来，并且不受攻击者控制。

概率一旦确定，攻击者便很难控制网络。

The DAG ordering rule ORD is an algorithm that takes as input a DAG of blocks and returns a linear ordering over its blocks.

DAG 排序规则ORD是一种算法，它以 block DAG 作为输入，并返回其块的线性排序。

In our partially synchronous model, the algorithm may take no parameters as input arguments (such as D, α, k, etc.).

在我们的部分同步模型中，该算法可能不需要作为输入参数（例如D，α，k等）。

D：D_{max}，上文中提到的攻击者可以最多延迟或加速消息的时间。

α：攻击者计算能力的$s\text{-}updated$上限。

k：完全子图的等级，即$k\text{-}cluster$。

We require that all blocks in the DAG were mined correctly according to $pow(\lambda)$.

If block a admits a path to block b in the DAG, a was necessarily created after b.

The DAG topology induces therefore a natural partial ordering, and the gist of the ordering rule is to extend this to a full ordering over the DAG.

我们要求 DAG 中的所有区块都根据$pow(\lambda)$的要求进行挖掘。

如果DAG中的区块a可以通过路径连接到区块b，则区块a必须在区块b之后创建。

因此，DAG 的拓扑结构会导致一种自然的部分排序，排序规则的主要目的是将其扩展为 DAG 上的完全排序。

如果DAG中的区块a可以通过路径连接到区块b，则区块a必须在区块b之后创建，这就建立了一种自然的部分排序关系。

但是，由于 DAG 的性质，不同的区块可能存在并行的创建关系，因此需要对这些并行创建的区块进行额外的排序，以建立完全的排序关系。

3.1 Convergence of the ordering

The following definitions adapt and extend the model from PHANTOM:

Property 1. An ordering rule ORD is said to be:

- Parameterless if its only input argument is a block DAG G; all blocks in G must be mined correctly according to the proof-of-work protocol $pow(\lambda)$.
- $(1-\alpha)$-convergent , $if\ \forall t > 0, \forall u \in \mathcal{H}\ and\ \forall b \in G_t^u$:

  $$\lim_{r\to 0} risk(b,t,r) = 0$$

  even when a fraction of at most α of the mining power is byzantine; the convergence rate of $risk(\cdot)$ should be in $\mathcal{O}(f(D,\lambda,\alpha))$,for some function f, and, in particular, may not grow indefinitely with t.

3.1 收敛性排序

以下定义调整并扩展了 PHANTOM 中的模型：

性质 1：如果排序规则ORD满足以下条件，则称其为收敛性排序：

无参数化 如果其唯一输入参数为一个 block DAG G，则该协议是无参数的；G中的所有块都必须根据工作量证明协议$pow(\lambda)$的规定来挖掘。

$(1-\alpha)$-收敛 若$\forall t > 0, \forall u \in \mathcal{H}$且$\forall b \in G_t^u$，满足：

$$\lim_{r\to 0} risk(b,t,r) = 0$$

我么称之为$(1-\alpha)$-收敛。

即使最多有α%的挖矿算力是拜占庭式的，$risk(\cdot)$的收敛速度应为$\mathcal{O}(f(D,\lambda,\alpha))$，其中$f$是某个函数，特别地，它不应随着$t$的增加而无限增长。

最多有比例为α的节点不可信任，这些节点可能会故意产生错误或不遵守规则。我们称之为有α%的挖矿算力是拜占庭式的。

下面我们重新审视该公式。

$\mathcal{H}$：遵循挖矿协议的节点的集合；

G_t^u：节点u在时间 t 维护的可观察到的 DAG 的一个副本；

r：一个区块被认为是最终确定的程度，或者说是它的“确认深度”。如果一个区块被认为是在链上的，那么它的确认深度就是链上已经确认的区块数量；

$\lim_{r\to 0} risk(b,t,r) = 0$要求对于任何给定的时间$t$和区块$b$，在一个足够小的$r$的范围内，风险（$risk$）的值可以被控制在足够小的范围内，以保证算法的正确性。

可以类比成完备空间下的柯西收敛列。

- Scalable if there exists a constant $\alpha > 0$ such that it $(1-\alpha)$-$converges$ for all $\lambda > 0$; the maximal such α is called the security threshold of ORD.

可拓展性 如果存在一个常数$\alpha > 0$，$\forall \lambda > 0$，满足$(1-\alpha)$-收敛，则它是可扩展的。α的最大值称为 ORD 的安全阈值。

- Self stabilizing if the security threshold of ORD depends on the t-$updated$ bound over the attacker's computational power.

自稳定性 如果 ORD 的安全阈值取决于攻击者计算能力的t-$updated$上限，则该协议是自稳定的。

网络的安全阈值能够根据攻击者计算能力的更新来自动调整。系统可以根据最近的攻击行为，例如攻击者掌握的算力量、攻击发生的频率等来判断攻击者的计算能力水平，然后自动调整安全阈值来保护主网的安全。

- Adaptive if the convergence rate of $risk(b,t,r)$ depends on the recent delay rather than the historical delay; formally, if it is in $\mathcal{O}(max_{s\geq t}((g(s-t,\alpha)\cdot f(D_s,\lambda,\alpha))$.
  The function g represents the "memory" of the process, i.e., how far into the past current values of $D(D_s)$ impact convergence.

自适应性 如果$risk(b,t,r)$的收敛速度取决于最近的延迟而不是历史延迟，则该协议是自适应的；形式上，它的复杂度为 $\mathcal{O}(max_{s\geq t}((g(s-t,\alpha)\cdot f(D_s,\lambda,\alpha))$中。

函数 g 表示运算过程的“内存”，即当前$D(D_s)$值对收敛的影响有多大。

函数 g 反映了在过去多久的时间内，延迟的大小会对$risk(b,t,r)$的收敛速度产生影响。

这个历史状态可以被看作是函数“记忆”或“内存”的一部分，因为它们会在一定程度上影响函数的行为。

Here, $risk(b,t,r)$ is the probability that the ordering between b and any other block c changes between time t and $t+r$.

3.2 Formal statement

We are finally ready to formally state the achievement of the KNIGHT protocol:

T_{HEOREM} 2. KNIGHT's ordering rule (Algorithm 2) is parameterless, scalable, self-stabilizing, and adaptive.

To the best of our knowledge, KNIGHT is the first proof-of-work based protocol to satisfy all of these properties.

For some comparisons: NC is not scalable, since its security threshold deteriorates as the block creation rate λ grows; PHANTOM is not parameterless, since its ordering rule takes as input k, corresponding to the network's worst case latency, and for the same reason it is not adaptive. SPECTRE does not guarantee convergence altogether.

In Section 4 we will further shed light on the convergence rate of KNIGHT, specifically, on the order of the functions f and g.

In Appendix A we will provide a rigorous proof of Theorem 2.

在这里，$risk(b,t,r)$表示在时间范围t到$t+r$内，区块 b 与任何区块 c 的顺序发生变化的概率。

换句话说，它是一个表示在一段时间内区块链中区块之间顺序稳定性的概率分布函数。

3.2 正式声明

我们终于可以正式陈述 KNIGHT 协议的成就：

定理 2. KNIGHT 的排序规则（算法 2）是无需参数、可扩展、自我稳定和自适应的。

据我们所知，KNIGHT 是第一个满足所有这些属性的 POW 协议。

DK 协议解决了虚拟货币“三难问题”!

横向比较：NC 协议不可扩展，因为其安全阈值随着区块创建速率λ的增长而恶化；PHANTOM 不是无需参数的，因为其排序规则需要输入 k，k 对应于网络的最坏延迟，出于同样的原因它也不是自适应的。SPECTRE 不能保证收敛。

NC 协议：BTC 协议，协议为了安全性考虑，不得不把块创建速率约束为 1 block/10min。

PHANTOM 协议：当前 Kaspa 正在使用的共识协议（指在 DK 协议上线前），参数 k 是根据最差网络情况而预先设定好的。

由于译者不了解 SPECTRE 协议，不能给出说明，希望各位读者补充。

在第 4 节中，我们将进一步阐明 KNIGHT 的收敛速度，具体而言，是根据函数f和g之间的顺序来说明的。

在附录 A 中，我们将提供定理 2 的严格证明。

4 CONFIRMATION TIMES

As common in proof-of-work protocols, the procedure for determining the robustness of the ordering – i.e., evaluating the function $risk$ – is done by the client locally, outside the context of consensus.

The performance of the protocol in terms of speed is captured by the convergence rate of $risk$.

This metric should arguably be dissected into two modes, optimistic and pessimistic.

In the former scenario, all participating nodes (miners) seem to behave properly, and in particular there is no visible split in the DAG; formally: all blocks agree on and amplify the entire chain selection, save perhaps a constant-size suffix.

In this optimistic scenario, KNIGHT performs very fast, and transactions may be safely confirmed after at most $\mathcal{O}((\ln(1/\epsilon) + D \cdot \lambda)/(1-2\alpha) + (D \cdot \lambda)^2)$ steps, or $\mathcal{O}((\frac{\ln(1/\epsilon)}{\lambda} + D)/(1-2\alpha) + D^2 \cdot \lambda)$ seconds.

In terms of Definition 1, the latter expression describes the asymptotic behaviour of the function f.

4 确认时间

正如我们在 POW 协议中的普遍做法一样，确定排序的鲁棒性——即评估函数 $risk$ 的过程是由客户端在共识阶段之外，即本地完成的。

在共识阶段，节点不需要相互交流来计算$risk$的值，每个节点都可以独立地进行计算。

协议在速度方面的表现由 $risk$ 的收敛速率决定。

这个度量标准可以被分解为乐观和悲观两种模式。

在乐观情况下，所有参与的节点（矿工）看起来表现正常，特别是在 DAG 中没有可见的分裂；形式上，所有块都同意并增强整个链选择，可能保存一个常数大小的后缀。

参与网络的所有节点都同意当前的区块链选择，并按照这个选择增加新的区块。在这种情况下，整个 DAG 是连通的，没有分叉。

但可能存在一个常数大小的后缀，也就是一些块没有被所有矿工包含进来。这里的后缀并非孤儿块，更大程度是指那些不属于最长链的块，它们的数量有限。

在这种乐观的情况下，KNIGHT 的性能非常快，交易可以在最多 $\mathcal{O}((\ln(1/\epsilon) + D \cdot \lambda)/(1-2\alpha) + (D \cdot \lambda)^2)$，或是 $\mathcal{O}((\frac{\ln(1/\epsilon)}{\lambda} + D)/(1-2\alpha) + D^2 \cdot \lambda)$ 内安全确认。

$\mathcal{O}(\mathcal{F}[x])$ 表示算法运行复杂度，$\mathcal{F}[x]$ 表示运行时间的渐进上界。

本文中的算法时间复杂度已经非常理想了。

根据定义 1，后一项表达式描述了函数 f 的渐进行为。

函数有界。

The function g defined therein can be shown to decay exponentially fast in its argument, implying that confirmation times are highly dependent on the recent worst-case latency in the network, and are insensitive to past or future network hiccups.

可以证明的是，函数 g 在定义域内表现出指数级衰减的性质，这意味着确认时间高度依赖于网络中最近的最坏延迟，并且对于过去或未来的网络故障不敏感。

In the pessimistic case, where an attacker continuously publishes late blocks and thereby slows down chain solidification, our bounds over confirmation times present an order-of-magnitude slow down: $\mathcal{O}(\exp(c \cdot D \cdot \lambda/(1-2\cdot\alpha)) + \ln(1/\epsilon)/(1-2\alpha)))$ steps.

在悲观情况下，攻击者持续发布延迟的块，从而减缓链的拓展速度，确认时间的上限出现了极大的变化：$\mathcal{O}(\exp(c \cdot D \cdot \lambda/(1-2\cdot\alpha)) + \ln(1/\epsilon)/(1-2\alpha)))$。

This describes the asymptotic behaviour of f in the pessimistic scenario (the behaviour of g remains the same).

这描述了悲观情况下函数f的渐进性质（函数g的性质保持不变）。

We stress that these bounds are far from tight—they result from the intractability of analyzing the chain solidification under the most sophisticated attack, and further grant the attacker unrealistic communication capabilities.

我们强调这些边界远非紧密——因为分析最复杂的攻击下链的巩固是难以计算的，同时还授予攻击者不切实际的算力。

巩固：当一个区块被足够多的后继区块所确认引用时，这个区块就被认为已经被“巩固”，也就是被确定下来了，不太可能被改变。

分析的时候，我们必须考虑最差的情况（虽然绝大多数时候并不是这样）——攻击者的算力令人绝望，这样得到的上限往往比实际情况更保守。

To overcome the intractability, and inspired by a technique from PHANTOM paper, our analysis waits for a rare event in which the honest network mined $Z \cdot D \cdot \lambda$ consecutive blocks in a chain, for some predetermined constant Z.

为了解决分析的不可解性，我们参考了 PHANTOM 论文中的一种技术。我们的分析需要一个罕见的出发点——诚实的节点网络挖掘了$Z \cdot D \cdot \lambda$个连续块，其中 Z 是一个预先设定的常数。

This event is guaranteed to happen within a constant number of steps.

可以保证的是，这个事件能够在有限步骤内发生。

While this condition is an overkill, relaxing it and tightening the confirmation times is a complex task, and we defer it to future work.

相关条件有些苛刻，如何放缩确认时间是一项复杂的任务，我们将新建一个文件夹，未来再解决。

Notwithstanding, an attacker cannot slow down the confirmation times of regular transactions, even if it carries out a visible attack.

As long as the user did not publish an explicit visible conflict to her transaction, its receiver will be able to accept it in the same order-of-magnitude as in the optimistic scenario.

Indeed, in this case, the ordering between the published transaction and other transactions would be commutative, and thus the pending chain solidification would be inconsequential to this transaction.

Admittedly, in the case of trading against a smart contract, this commutative property might not hold.

然而，即使攻击者发动了显著的攻击，这并不会减缓正常交易的确认时间。

只要用户没有向其交易显式地发布可见冲突，接收者就能够和在乐观情况下相同的时间范围内接受这笔交易。

可见冲突：两个或多个交易在同一时间被提交到区块链，并且有一个区块只能包含其中一个交易。

实际上，在这种情况下，已发布交易与其他交易之间的顺序是可交换的，因此挂起的链巩固对这笔交易没有影响。

挂起的链巩固：指的是在网络中未达成共识的交易或块形成的一条分支，它处于等待进一步的确认和巩固的状态。

一旦这条分支上的交易或块被进一步确认，并加入到整个链中，那么这个“挂起的链”就被“固化”了，也就是加入到了整个链的选择中。

诚然，若基于智能合约进行交易，这种可交换性可能不成立。

5 IMPLEMENTATION DETAILS

An implementation of Algorithm 2 will be made available online.

5 实现细节

算法 2 的实现细节将在网上公开。

	Visible attack	**No visible attack**
NC	$\mathcal{O}(\frac{(\ln(1/\epsilon)+D_t\lambda)}{max\{0,\frac{1-\alpha}{1+D_t\cdot\lambda}-\alpha\}})$	(same as the visible attack case, asymptotically)
PHANTOM	$\mathcal{O}(\exp(c_1\frac{D_{max}\lambda}{1-2\alpha})+\frac{\ln(1/\epsilon)}{1-2\alpha})$	$\mathcal{O}(\frac{\ln(1/\epsilon)+D_{max}\lambda}{1-2\alpha})$
SPECTRE	∞	$\mathcal{O}(\frac{\ln(1/\epsilon)+D_t\lambda}{1-2\alpha})$
KNIGHT	$\mathcal{O}(\exp(c_2\frac{D_t\lambda}{1-2\alpha})+\frac{\ln(1/\epsilon)}{1-2\alpha}))$	$\mathcal{O}(\frac{\ln(1/\epsilon)+D_t\lambda}{1-2\alpha}+(D_t\lambda)^2)$

Table 1: A comparison of the convergence rates of different proof-of-work protocols, in terms of time-steps (equiv., number of blocks), in the presence of a visible ongoing liveness attack (left column) and when no such attack is carried visibly (right column).

D_{max} denotes an a priori upper bound on the worst case latency, whereas D_t denotes an upper bound on the current latency (including possible delays by an adversary).

To get expected confirmation times in seconds, multiply each expression by the expected block interval λ^{-1}.

表格 1：可见的持续活跃攻击（左列）；没有可见攻击（右列），不同工作量证明协议的收敛速率比较，以时间步数（即块数）表示。

D_{max}是最坏情况下网络延迟的先验上界，而D_t是当前延迟的上界（包括可能由攻击者引起的延迟）。

如果想要得到预期的确认时间（秒），请将每个表达式乘以预期的块间隔λ^{-1}。

5.1 Block size

So far we treated synchronous protocols as assuming a bound on latency D.

In fact, increasing D and decreasing λ by the same multiplicative factor has no effect and could be regarded as mere change in units.

Thus, in truth, the latency assumption takes the form of a bound over $D \cdot \lambda$.

Recall that D depends on the size of messages $block\_size\_limit$ (Section 3).

Thus, increasing the block size would have a similar effect to that of increasing the block rate λ.

5.1 块大小

到目前为止，我们假设同步协议存在对延迟 D 的限制。

假设同步协议中存在一个对网络延迟的限制D，也就是一个上界，来保证协议的正确性和安全性。

实际上，将 D 增加并将λ减小相同的乘法因子不会产生任何影响，可以视为仅仅是单位的变化。

如果将 D 增加 10 倍，同时将λ减小 10 倍，那么实际上一个区块被生成所需的时间仍然是相同的，只是这个时间被表示的单位不同了。

因此，事实上，延迟假设采用了对$D \cdot \lambda$的限制。

简单说延迟由D和λ两部分决定。

请注意，D取决于消息块大小$block_size_limit$（第 3 节）。

因此，增加块大小将具有类似于增加块速率λ的效果。

Consequently, in the same manner in which scalable protocols (Definition 1) remain secure under any λ, they remain secure under any $block\_size\_limit$.

In the following subsection we discuss whether scalable partially synchronous protocols, such as KNIGHT, need to limit λ or $block\_size\_limit$.

因此，在任意的λ下，可扩展协议（定义 1）能够保持安全性，相同的是，协议也能在任意的$block_size_limit$下保持安全性。

在下一小节中，我们将讨论可扩展部分同步协议，比如在 KNIGHT 中，讨论是否需要限制λ或$block_size_limit$。

5.2 Difficulty Adjustment Algorithm (DAA)

5.2 难度调整算法（DAA）

NC and other proof-of-work protocols employ a DAA that increases the difficulty-target of creating new blocks when the computational power contributed to block creation (aka hashrate) increases, and vice versa when it decreases.

It is common to ascribe the Sybil-resiliency of the system to this mechanism. However, in truth, proof-of-work suffices to protect against Sybil-nodes even without any DAA.

NC 和其他 POW 协议采用了一种 DAA，当用于创建新块的计算能力（也称为哈希率）增加时，DAA 会增加创建新块的难度，反之亦然。

*此处有删减，原文提及了一篇论文：*PHANTOM and GHOSTDAG: A scalable generalization of nakamoto consensus

通常将系统的 Sybil 弹性归因于此机制。然而，实际上，即使没有任何 DAA，工作量证明也足以保护免受 Sybil 节点的攻击。

Sybil：一种恶意行为，攻击者通过控制多个虚假身份（也称为“Sybil 节点”）来欺骗系统，从而获取不当利益或破坏系统的安全性和可靠性。

Sybil 弹性：指一个系统对于 Sybil 攻击的弹性或者韧性。

In fact, even if nodes were free to choose the difficulty of their own blocks, one could devise a secure consensus protocol by granting each block a weight, or “voting power”, in proportion to its difficulty.

事实上，节点可以自由选择其自己块的难度，换个角度，可以通过按比例为每个块授予权重或“投票权”，从而设计出一种安全的共识协议。

每个块的投票权与其难度成比例，因此攻击者不能通过创建大量的伪造节点来控制投票权，因为他们只能产生低难度的块，其投票权很小，不能影响整个网络的共识。因此，即使没有 DAA，也可以通过这种方式保护系统免受 Sybil 攻击的影响。

这里的难度是指工作量证明中的难度目标，是一个数字，它规定了一个新块的哈希值必须满足的特定条件。

Instead, the motivation for DAA is threefold:

- Existing protocols operate in the synchronous setup which assumes an a priori bound over the number of blocks created per one unit of delay, i.e., $D \cdot \lambda$. For instance, NC assumes $D \cdot \lambda \ll 1$, and PHANTOM assumes $D \cdot \lambda \ll k + 1$.
  To preserve these bounds and keep the protocol secure, λ cannot increase indefinitely, and must be regulated by the protocol.
- DoS prevention: The capacity of the network and of nodes is limited. The DAA throttles the block creation rate and ensures that the maximum capacity is not exceeded.
- Some application considerations necessitate access to absolute time, such as the regulation of minting, or timelocks. These applications use the block count as a proxy for absolute time.

The first consideration above is irrelevant to KNIGHT, which can cope with dynamic D and λ (and $D \cdot \lambda$).

While KNIGHT still requires DAA for the latter considerations-particularly DoS prevention - it could be satisfied perhaps with relaxed versions of DAA.

we hope that this discussion spurs new ideas for proof-of-work system designs in the partially synchronous setup.

相反，DAA 的目的有三个：

现有协议在同步设置中运行，假设对于每个延迟单位创建的块数有一个先验限制，上限为$D \cdot \lambda$。

延迟单位：某个特定的时间间隔。

例如，NC 假设 $D \cdot \lambda \ll 1$，PHANTOM 假设$D \cdot \lambda \ll k + 1$。

为了维持限制并保持协议的安全性，λ不能无限增加，必须由协议进行调节。

防止 DoS 攻击：网络和节点的容量是有限的。DAA 将会限制块创建率，同时确保（块）不超过最大容量。

DoS 攻击：“拒绝服务”攻击，是指攻击者使用各种手段，通过占用网络带宽、耗尽服务器资源等方式，使得服务无法正常提供，从而导致服务不可用或系统崩溃。

考虑到一些应用程序有着绝对时间的需求，例如铸造货币或定时锁。这些应用程序使用块计数作为绝对时间的代理。

在这些应用中，需要某种方式来表示时间，而区块链中的区块可以被视为时间的一个近似值。

上述第一点和 KNIGHT 没关系，因为它可以应对动态的D和λ（包括$D \cdot \lambda$）。

KNIGHT 无需预先设定参数。

虽然 KNIGHT 仍然需要考虑 DAA 的后两个目的，特别是 DoS 防御，但它可能会满足弱化版本的 DAA。

我们希望这种讨论能够激发新的想法，用于在部分同步设置下的工作量证明系统设计。

6 RELATED WORK

We conclude this paper with a survey of related work. DAG-based protocols have been mentioned extensively throughput the paper, see for example Table 1.

Additional relevant protocols include GHOST, which is an alternative chain-selection rule to NC's longest chain, and which performs similarly (in qualitative terms) to NC.

Thunderella is a permissionless protocol that is responsive in the strong sense of performing according to the network's actual latency; it requires a super majority of 75% to be honest for this optimistic mode (compared to KNIGHT's 51% majority), as well as the pre-selection of a special "accelerator" node, which compromises the permissionless property of the system.

The works in [3, 7, 21] maintain k parallel NC chains, where each block is assigned in random to one of these chains.

The ordering rule must then specify the respective ordering between blocks in different chains.

6 相关工作

本文在相关工作方面进行了调查。DAG 协议在本文中广泛提及，例如表 1。

其他相关协议，包括 GHOST，它是 NC 最长链的另一种选择规则，其表现与 NC 相似（从质量方面来讲）。

Thunderella 是一种无需许可的协议，以强响应的方式来响应网络的实际延迟；当然也有代价，即使是在乐观情况下，它也需要 75%的节点是诚实的（相对来说，KNIGHT 仅需要 51%），以及需要预选一个特殊“加速器”节点，这会影响系统的无需许可属性。

permissionless protocol：指的是在网络中没有中心化的机构或实体可以控制或限制参与者的加入或退出，任何人都可以自由地参与其中。

无需许可协议与无参数协议有一定区别。

Thunderella 需要预先确定一个节点，这破坏了它的无需许可性质。

论文[3，7，21]的工作是维护 k 个平行 NC 链，其中每个块随机分配给这些链中的某一个。

论文 3 Prism: Deconstructing the blockchain to approach physical limits.

论文 7 Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition.

论文 21 Ohie: Blockchain scaling made simple.

排序规则必须指定不同链中块之间的相应顺序。

These works operate in the synchronous setup, as they pre-assume k so as to ensure that each chain grows with negligible latency; conceptually, as observed by [19], these protocols require $D \cdot \lambda/(k+1) \ll 1$.

这些工作在同步设置中进行，它们需要预先假定 k，以确保每个链的增长所带来的延迟是可以忽略的；从概念上讲，正如[19]所说，这些协议需要一个前提：$D \cdot \lambda/(k+1) \ll 1$。

论文 19 PHANTOM and GHOSTDAG: A scalable generalization of nakamoto consensus.

Prism claim a confirmation time of $\mathcal{O}(max(c1(\alpha) \cdot D, c2(\alpha) \cdot B_v \cdot \ln(1/\epsilon)))$ seconds; here, B_v/C effectively represents the number of blocks per second (λ, in our work).

Prism 声称确认时间为 $\mathcal{O}(max(c1(\alpha) \cdot D, c2(\alpha) \cdot B_v \cdot \ln(1/\epsilon)))$ 秒；这里，B_v/C实际上表示每秒块的数量（当然我们更喜欢用λ来表示）。

Prism：可能是某种共识协议。

Importantly, in the above term D stands as a function that depends only on network latency and does not depend on the block message size (denoted B_v , and in our work $block\_size\_limit$).

重要的是，在上述术语中，D作为一个仅取决于网络延迟而不取决于块消息大小（表示为 B_v，本文中称为 $block\_size\_limit$）的函数。

We find this claim questionable, and argue that if indeed D does not depend on $block\_size\_limit$, then "proposer blocks" and "voter blocks" in Prism do not in fact attest that "transaction blocks" referenced by them have been fully published, which opens up data availability attacks.

我们很怀疑这种说法，如果的确如此，那么 Prism 中的"提议块"和"投票块"实际上并没有证明它们所引用的"交易块"已被完全发布，这会导致数据可用性攻击。

提议块：提出新块的节点（即提议者）在区块链中创建的块。

投票块：确认提议块而创建的块，也可以理解为是一个由验证节点共同建立的确认区块链，这些块将包含对提议块的投票结果。

交易块：指包含实际交易信息的块，通常由提议块和投票块引用。

数据可用性攻击：一种网络攻击，在这种攻击中，攻击者试图阻止其他节点能够获取完整的区块数据，从而影响区块链网络的正常运行。

这种攻击通常涉及到在不公开完整数据的情况下进行验证，导致节点无法确认交易是否真的已经被记录在区块链上，因此可能会导致双重支付等问题。

文中提到交易块不能保证完全发布，即没有被多数节点确认，这可能导致攻击。

The number of chains in Prism further depends on the parameter ϵ.

Prism 中的链数还取决于参数ϵ。

The work in [17] proposes a series of protocols, Slush, Snowflake, and Snowball, which use a network sampling technique to resolve conflicts between nodes.

论文[17]的工作提及了一系列协议，Slush、Snowflake 和 Snowball，它们使用一种网络采样技术来解决节点之间的冲突。

论文 17 Scalable and probabilistic leaderless bft consensus through metastability.

The paper claims very fast confirmation times (1.35 seconds).

该论文声称确认时间非常快（1.35 秒）。

Yet, these protocols operate in the synchronous model (see Section 2, “Achieving Liveness”), and thus confirmation times in the pessimistic case are not responsive to the network’s latency.

然而，这些协议在同步模型下运行（参见第 2 节“实现活性”），因此在悲观情况下，确认时间不会响应网络的延迟。

The protocols are further limited to a fixed confidence parameter ϵ (see e.g. Subsection 3.2 therein), similarly to Prism.

和 Prism 类似，这些协议进一步受限于固定的置信度参数ϵ（例如，在其中的第 3.2 小节中）。

Finally, this line of work builds on novel assumptions on nodes’ability to sample the network.

最后，这一工作方向基于一个新假设——节点对整个网络进行采样。

在传统的共识协议中，节点通常只与其它几个邻居节点进行通信，而不是整个网络。

Our work was motivated by [15], which provide possibility and impossibility results regarding responsive consensus protocols.

我们的工作受到[15]的启发，其提供了关于响应式共识协议的可能性和不可能性结果。

To circumvent their 34% threshold bound for responsive parameterless protocols, we focused on a relaxed property that aims to be responsive to the maximal latency causable by an adversary.

为了规避先前研究关于响应参数协议 34%的阈值限制，我们把目光投向一种放宽的属性，旨在对攻击者可能造成的最大延迟做出响应。

KNIGHT respects the bound proven by Pass and Shi in that it is responsive to the current worst-case adversarial latency (Δ, in their model) but not to the actual observable one (δ therein).

KNIGHT 满足 Pass 和 Shi 所证明的相关约束，即它对当前最坏的敌对延迟（在他们的模型中用Δ表示）做出响应，但不对实际可观察到的延迟（表示为δ）做出响应。

Their impossibility result (Section 9.2) relies directly on the attacker increasing the delay from δ to Δ after transactions have been confirmed.

他们的不可能性结果（第 9.2 节）直接依赖于攻击者在交易确认后将延迟从δ增加到Δ。

In our model, however, transaction confirmation times depend on Δ (D_t,in our notation).

然而，在我们的模型中，交易确认时间取决于Δ（本文中记为D_t）。

论文 15 提到了一个不可能性结果，即在没有特殊假设的情况下，不存在一个“完全响应式”的共识协议，能够同时满足以下两个条件：

- *对于任意合法的网络延迟δ，当存在一个“快节点”时（即网络延迟比δ小一定比例的节点），共识协议需要在有限的时间内完成。*
- *当存在一个最坏情况下的对手攻击时（即网络延迟达到某个上限Δ），共识协议需要在有限的时间内完成。*

Pass 和 Shi 的证明过程中，使用了一种技巧，即他们通过假设攻击者可以在交易被确认后增加延迟，来证明这个不可能性结果。

而 KNIGHT 协议的特点在于，它的交易确认时间确实取决于最坏情况下的网络延迟Δ，而不是攻击者可能对已确认的交易进行的任何延迟操作。因此，KNIGHT 协议在一定程度上规避了论文[15]中的不可能性结果。

For discussion of tighter transaction confirmation policies, which employ absolute time in addition to the ledger state, see [13].

关于更严格的交易确认政策的讨论，它们除了使用账本状态外还使用绝对时间，可以参见[13]。

论文 13 An analysis of acceptance policies for blockchain transactions.

The results therein apply, qualitatively, to KNIGHT as well.

其中的结果也适用于 KNIGHT，从定性上来说。