State Machine Replication in the Libra Blockchain (中英混合版)

This report describes the algorithmic core of LibraBFT, named LBFT, and discusses next steps in its production. The consensus protocol is responsible for ordering and finalizing transactions. LBFT decentralizes trust among a dynamically reconfigurable set of validators, and remains safe against net-work asynchrony and even if at any particular configuration epoch, a threshold of the participants are Byzantine.
本报告描述了LibraBFT的算法核心，称为LBFT，并讨论了其生产的下一步步骤。共识协议负责命令和最终确定事务。LBFT分散了动态可重构验证器集合之间的信任，并且在网络异步中保持安全，即使在任何特定的配置时代，参与者的阈值是拜占庭的。

LBFT is based on HotStuff, a recent protocol that leverages several decades of scientific advances in Byzantine fault tolerance (BFT) and achieves the strong scalability and security properties required by internet settings. Several novel features distinguish LBFT from HotStuff. LBFT incorporates a novel round synchronization mechanism that provides bounded commit latency under synchrony. It introduces a nil-block vote that allows proposals to commit despite having faulty leaders. It encapsulates the correct behavior by participants in a separable trust zone, allowing it to run within a secure hardware enclave that reduces the attack surface on participants.
LBFT基于HotStuff，这是一种最近的协议，它利用了拜占庭容错(BFT)几十年的科学进步，实现了互联网设置所需的强大可扩展性和安全性。几个新功能将LBFT与HotStuff区分开来。LBFT集成了一种新颖的圆形同步机制，在同步下提供有界提交延迟。它引入了一个无块投票，允许提案尽管有错误的领导者，但仍可提交。它将参与者的正确行为封装在一个可分离的信任区中，允许它在一个安全的硬件飞地中运行，减少参与者的攻击面。

LBFT can reconfigure itself, by embedding configuration-change commands in the sequence. A new configuration epoch may change everything from the validator set to the protocol itself.
LBFT可以通过在序列中嵌入配置更改命令来重新配置自己。一个新的配置时代可能会改变从验证器集到协议本身的一切。

Introduction

The advent of the internet and mobile broadband has connected billions of people globally, providing access to knowledge, free communications, and a wide range of lower-cost, more convenient services. This connectivity has also enabled more people to access the financial ecosystem. Yet, despite this progress, access to financial services is still limited for those who need it most.
互联网和移动宽带的出现连接了全球数十亿人，提供了知识、免费通信和各种更低成本、更便利的服务。这种连接也使更多人能够进入金融生态系统。然而，尽管取得了这一进展，最需要金融服务的人获得金融服务的机会仍然有限。

Blockchains and cryptocurrencies have shown that the latest advances in computer science, cryptography, and economics have the potential to create innovation in financial infrastructure, but existing systems have not yet reached mainstream adoption. As the next step toward this goal, we have designed the Libra Blockchain with the mission to enable a simple global currency and financial infrastructure that empowers billions of people.
区块链和加密货币已经表明，计算机科学、密码学和经济学的最新进展有可能在金融基础设施中创造创新，但现有系统尚未达到主流采用。作为朝着这一目标迈出的下一步，我们设计了天秤座区块链，其使命是实现一个简单的全球货币和金融基础设施，为数十亿人赋能。

At the heart of this new blockchain is a consensus protocol called LBFT— the focus of this report — by which blockchain transactions are ordered and finalized. LBFT decentralizes trust among a set of validators that participate in the consensus protocol and maintain agreement on the history of transactions.
这个新区块链的核心是一个被称为LBFT的共识协议——本报告的重点——通过该协议，区块链交易被命令和最终确定。LBFT分散了参与共识协议并对交易历史保持一致的一组验证者之间的信任。

LBFT encompasses several important design considerations.
LBFT包含几个重要的设计考虑因素。

Permissioned. The first one is a strong commitment for maintaining the Libra blockchain decentralized. Initially, the participating validators will be permitted into the consensus network by an association con- sisting of a geographically distributed and diverse set of Founding Members, which are organizations chosen according to objective membership criteria with a vested interest in bootstrapping the Libra ecosystem. In the literature, this model is referred to as

permissioned, an approach that promotes sustainability without wasting excessive computational power.
第一个是维护天秤座区块链分散的坚定承诺。最初，参与验证者将被一个由地理分布和多样化的创始成员组成的协会允许进入共识网络，这些成员是根据客观的成员标准选择的组织，在引导天秤座生态系统方面有既得利益。在文献中，这种模式被称为许可的，这种方法可以促进可持续性，而不会浪费过多的计算能力。

Over time, membership eligibility will gradually become open while preserving the guarantee of decen- tralization with careful governance. To facilitate dynamic membership, LBFT can reconfigure itself by embedding configuration-change commands that require special credentials in the sequence. A new configu- ration epoch may change every part of the algorithm, from the validator set to the protocol itself.
随着时间的推移，成员资格将逐渐变得开放，同时通过谨慎的治理保留了去中心化的保证。为了促进动态成员资格，LBFT可以通过在序列中嵌入需要特殊凭据的配置更改命令来重新配置自己。一个新的配置时代可能会改变算法的每个部分，从验证器集到协议本身。

Byzantine Fault Tolerance The second consideration is to prevent individual faults from contaminating the entire system. It is worth noting that generally, participants are expected to be credible and the network to have predictable transmission delays. LBFT is designed to mask any deviation from correct behavior in a third of the participants. These cover anything from a benign bit flipping in a node’s storage to fully comprising a server by stealing its secret keys. Additionally, LBFT maintains consistency even during periods of unbounded communication delays or network disruptions. This reflects our belief that consensus protocols whose safety rely on synchrony would be inherently both complex and vulnerable to Denial-of-Service (DoS) attacks on the network.
第二个考虑是防止个别故障污染整个系统。值得注意的是，一般来说，参与者应该是可信的，网络应该有可预测的传输延迟。LBFT旨在掩盖三分之一参与者对正确行为的任何偏差。这些涵盖了从节点存储中的良性比特翻转到通过窃取其秘钥完全构成服务器的任何内容。此外，即使在无界通信延迟或网络中断期间，LBFT也保持一致性。这反映了我们的信念，即安全依赖于同步的共识协议本质上既复杂，又容易受到网络上的拒绝服务(DoS)攻击。

High Assurance The core logic of LBFT allows simple and robust implementation, paralleling that of public blockchains based on Nakamoto consensus [23]. Notably, the protocol is organized around a single communication phase and allows a concise safety argument and a robust implementation. LBFT thus bridges between the simplicity and modularity of public blockchains based on Nakamoto consensus, but decentralizes trust based on permissions.

LBFT的核心逻辑允许简单和健壮的实现，与基于中本聪共识的公共区块链并行[23]。值得注意的是，该协议围绕单一通信阶段组织，允许简洁的安全论点和健壮的实现。LBFT因此在基于中本聪共识的公共区块链的简单性和模块化之间建立了桥梁，但基于权限分散了信任。

LBFT is a cutting edge technology that opens new possibilities in BFT scalability. It is based on the HotStuff algorithm [26] whose key benefit is consensus-linearity. Briefly, this means that under a wide range of settings, consensus decisions cost no more communication than to simply spread the decision to everyone.
LBFT是一项在BFT可扩展性上开辟新可能性的前沿技术，它基于HotStuff算法[26]，其关键好处是共识线性，简而言之，这意味着在广泛的设置下，共识决策的成本不会比简单地将决策传播给每个人更多。

LBFT has various features that distinguish it from HotStuff, some of which are left for future implemen- tation and are discussed only briefly here, in Section 8. The core protocol rotates the leader at every block in the chain in order to provide fairness and symmetry, at no extra communication overhead. It implements a liveness and synchronization mechanism that maintains linearity under a broad set of conditions, as well as provides concrete latency bounds that are discussed formally in [24]. An enhancement to the voting rules allows commits to proceed despite faulty leaders, and planned enhancements will uphold consistency guarantees against higher corruption thresholds than traditional BFT approaches [21]. These features will be described in more detail in future documents.
LBFT有不同于Hot Stuff的各种特性，其中一些是留待将来实现的，这里只在第8节中简单讨论。核心协议在链中的每个区块轮换领导者，以提供公平和对称，没有额外的通信开销。它实现了一种活泼和同步机制，在广泛的条件下保持线性，并提供了[24]中正式讨论的具体延迟边界。对投票规则的改进允许承诺在领导人有错误的情况下继续进行，计划中的改进将维护一致性保证，防止比传统的BFT方法更高的腐败阈值[21]。这些特点将在未来的文件中得到更详细的描述。

Preliminaries

The goal of LBFT is to maintain a database of programmable resources with fault tolerance. At the core, the system is designed around a state-machine-replication (SMR) engine where participants form agreement on a sequence of transactions and apply them in sequence order deterministically to the replicated database.
LBFT的目标是维护具有容错能力的可编程资源数据库。该系统的核心是围绕状态机复制（SMR）引擎设计的，在该引擎中，参与者就一系列交易达成协议，并按确定的顺序将它们确定地应用于复制的数据库。

LBFT is designed for a classical settings in which an initial system consists of a networked system of n participants. The initial set of members is determined when the system is bootstrapped. LBFT can reconfigure itself by embedding configuration-change commands in the sequence. A new configuration epoch may change everything from the validator set to the protocol itself. This allows the system to churn and allow open participation down the line.
LBFT是为经典环境设计的，其中初始系统由n个参与者的联网系统组成。引导系统时，将确定成员的初始集合。LBFT可以通过在序列中嵌入配置更改命令来重新配置自身。一个新的配置时代可能会将所有内容（从验证器集更改为协议本身）进行更改。这允许系统搅动并允许公开参与。

To model failures, we define an adversary that controls the network delays and the behavior of up to a threshold of the participants. In the standard distributed computing model, there are three adversarial settings:
为了对故障进行建模，我们定义了一个对手，该对手控制网络延迟以及行为达到参与者阈值的程度。在标准的分布式计算模型中，存在三种对抗设置：

Synchronous The synchronous model is the one in which the Byzantine consensus problem was originally conceived [25, 20]. In the synchronous model, there exists some known finite time bound Δ, such that the adversary can delay message delivery from an honest origin by at most Δ. The safety of the solution introduced by Lamport et al. relied on synchrony, a dependency that practical systems wish to avoid both due to complexity and because it exposes the system to DoS attacks on safety.
同步模型是最初设想拜占庭共识问题的模型[25，20]。在同步模型中，存在一些已知的有限时间限制，以至于对手最多可以延迟来自诚实原点的消息发送。Lamport等人介绍的解决方案的安全性依赖于同步，这是一个实用系统希望避免的依赖，因为它会使系统面临安全的DoS攻击。

Asynchronous In the Asynchronous model, the adversary can delay message delivery by any finite amount of time even between honest parties. Note that, whereas there is no bound on the delay to message delivery, messages sent between honest parties must eventually be delivered. The celebrated FLP result [17] indicates that in asynchronous settings, any consensus solution that tolerates a single, crash failure must have an infinite execution. In lieu of synchrony assumptions, randomized algorithms, pioneered by Ben- Or [3] guarantee progress with high probability. A line of research gradually improved the scalability of such algorithms, including [15, 22, 7, 1]. LBFT is not designed as a pure asynchronous solution, though in the future, it may incorporate various components of randomized solutions to thwart adaptive attacks.
在异步模型中，对手甚至可以在诚实方之间将邮件传递延迟任何有限的时间。请注意，尽管消息传递的延迟没有限制，但诚实各方之间发送的消息最终必须传递。著名的FLP结果[17]表明，在异步设置中，

任何容许单个崩溃失败的共识解决方案都必须无限执行。代替同步假设，由Ben-Or[3]率先提出的随机算法保证了进步的可能性。一系列研究逐渐改善了此类算法的可伸缩性，包括[15，22，7，1]。LBFT并非设计为纯粹的异步解决方案，尽管在将来，它可能会合并随机解决方案的各种组件以阻止自适应攻击。

Partial synchrony The adversarial model which is assumed for LBFT is a hybrid between synchronous and asynchronous models called partial synchrony. It models practical settings in which the network goes through transient periods of asynchrony (e.g., under attack) and maintains synchrony the rest of the times.
为LBFT假设的对抗模型是同步和异步模型之间的混合，称为部分同步。它对实际设置进行建模，在这些设置中，网络经历了短暂的异步期(例如，在攻击下)，并在其余时间保持同步。

A solution approach for partially synchronous settings introduced by Dwork et al. [13] separates safety (at all times) from liveness (during periods of synchrony). DLS introduced a round-by-round paradigm where each round is driven by a designated leader. Progress is guaranteed during periods of synchrony as soon as an honest leader emerges, and until then, rounds are retired by timeouts. The DLS approach underlies most practical BFT works to date as well as the most successful reliability solutions in the industry, for example, the Google Chubbie lock service [5], Yahoo’s ZooKeeper [19], etcd [14], Google’s Spanner [11], Apache Cassandra [8] and others.
Dwork等人[13]引入的部分同步设置的解决方法将安全性(在任何时候)和活动性(在同步期间)分开。DLS引入了一种逐轮的范式，每轮由指定的领导者驱动。一旦一个诚实的领导者出现，在同步期内的进展就得到保证，在那之前，回合会因超时而退出。DLS方法是迄今为止最实用的BFT工作以及业内最成功的可靠性解决方案的基础，例如谷歌Chubbie锁服务[5]、雅虎的ZooKeeper[19]、etcd[14]、谷歌的Spanner[11]、Apache Cassandra[8]和其他。

Formally, the partial synchrony model assumes a Δ transmission bound similar to synchronous networks, and a special event called GST (Global Stabilization Time) such that:
从形式上讲，部分同步模型假设一个与同步网络相似的RT传输绑定，以及一个被称为GST(全球稳定时间)的特殊事件，这样:

- GST eventually happens after some unknown finite time.
- GST最终会在某个未知的有限时间后发生。

- Every message sent at time t must be delivered by time max{t, GST } + Δ.
- 在时间t发送的每个消息都必须在时间max{t，GST}+Δ之前传递。

An alternative definition for partial synchrony is to assume that there is some finite, but unknown, upper bound on message delivery [13].
部分同步的另一种定义是假设消息传递上有一个有限但未知的上限[13]

Thresholds
There are several key impossibility and lower bounds that need to be taken into consideration. Under the asynchronous (and partially synchronous) model, Fischer et al. [16] showed that Byzantine consensus cannot be solved with more than one third faults, that is, at most f faults where n = 3f + 1. In this settings, a quorum consists of n − f = 2f + 1 validators. Under synchrony, Dolev and Reischuk showed in [12] an Ω(n2) worst case communication complexity.
需要考虑几个关键的不可能和下界。在异步（和部分同步）模型下，Fischer等人。[16]表明，拜占庭共识不能用多于三分之一的错误来解决，也就是说，最多f个错误，其中n=3f+1。在这种情况下，仲裁群体由n－f=2f+1个验证者组成。在同步下，Dolev和Reischuk在[12]中显示了Ω（n2）最坏情况下的通信复杂度。

Technical Background

LBFT is based on a line of replication protocols in the partial synchrony model with Byzantine Fault Tolerance (BFT), e.g., [13, 9, 18, 4, 6, ?, 10]. It embraces cutting-edge techniques from these works to support at scale a replicated database of programmable resources.
LBFT基于拜占庭容错(BFT)部分同步模型中的一系列复制协议，例如[13，9，18，4，6，?，10]。它包含这些工作中的尖端技术，以支持大规模复制的可编程资源数据库。

Round-by-round BFT solutions.
The classical solutions for practical BFT replication share a common, fundamental approach. They operate in a round by round manner. In each round, there is a fixed mapping that designates a leader for the round (e.g., by taking the round modulo n, the number of participants). The leader role is to populate the network with a unique proposal for the round.
实用BFT复制的经典解决方案有一个共同的基本方法。它们以一轮一轮的方式运作。在每一轮中，都有一个固定的映射，为一轮指定一个领导者(例如，通过采用轮模n，参与者的数量)。领导者的作用是在网络中填充一个该轮的独特建议。

The leader is successful if it populates the network with its proposal before honest validators give up on the round and time out. In this case, honest validators participate in the protocol phases for the round. Many classical practical BFT solutions operate in two phases per round. In the first phase, a quorum of validators certifies a unique proposal, forming a quorum certificate, or a QC.

In the second phase, a quorum of votes on a certified proposal drives a commit decision. The leaders of future rounds always wait for a quorum of validators to report about the highest QC they voted for. If a quorum of validators report that they did not vote for a any QC in a round r, then this proves that no proposal was committed at round r.
如果领导者在诚实的验证者放弃回合和超时之前，用它的提议普及网络是成功的。在这种情况下，诚实的验证者参与了这一轮的协议阶段。许多经典实用的BFT解决方案每轮分两个阶段运行。在第一阶段，验证者的法定人数证明一个独特的建议，形成法定人数证书或QC。在第二阶段，对认证提案的法定投票数驱动提交决定。未来几轮的领导人总是等待法定人数的验证者报告他们投票支持的最高质量控制。如果法定人数的验证人报告他们在一轮r中没有投票支持任何QC，那么这证明在一轮r中没有提出任何建议。

HotStuff is a three-phase BFT replication protocol. In HotStuff, the first and second phases of a round are similar to PBFT, but the result of the second phase is a certified certificate, or a QC-of-QC, rather then a commit decision. A commit decision is reached upon getting a quorum of votes on the QC-of-QC (a QC-of-QC-of-QC).

HotStuff是一个三阶段的BFT复制协议。在HotStuff中，一轮的第一阶段和第二阶段类似于PBFT，但第二阶段的结果是一个认证证书，或一个QC-of-QC，而不是然后一个提交决定。提交决定是在获得对QC-of-QC(a QC-of-QC)的法定票数时达成的。

LBFT embraces the three-phase HotStuff paradigm which brings two benefits, optimistic responsiveness and linearity, that are not simultaneously achieved in two-phase solutions. Jointly, these two properties provide consensus linearity without sacrificing asynchrony: An honest leader can prove the safety of a proposal by referencing a single QC (from the highest round). HotStuff was adopted in several subsequent works, including PaLa [10], BFTree [2], and others.
LBFT采用了三阶段HotStuff范式，它带来了两个好处，即乐观响应性和线性度，这是两阶段解决方案中无法同时实现的。这两个属性共同提供了共识线性度，而不牺牲异步性:诚实的领导者可以通过引用单个QC(来自最高回合)来证明提案的安全性。HotStuff在随后的几项作品中被采用，包括PaLa[10]、BFTree[2]和其他作品。

Regardless of the number of phases, a round might not succeed in making progress due to a faulty leader or because validators enter the rounds at different times. The case of unsuccessful rounds is discussed below.
无论阶段的多少，由于领导者的错误或验证者在不同的时间进入回合，回合可能无法成功取得进展。下文讨论了回合不成功的情况。

Chaining LBFT borrows a chaining paradigm that has become popular in blockchain BFT protocols and popularized by HotStuff. In the chaining approach, the three phases for commitment are spread across rounds. More specifically, every phase is carried in a round and contains a new proposal. The leader of round k drives only a single phase of certification of its proposal. In the next round, k + 1, a leader again drives a single phase of certification. This phase has multiple purposes. The k + 1 sends its own k + 1 proposal. However, it also piggybacks the QC for the k proposal. In this way, certifying at round k + 1 generates a QC for k + 1, and a QC-of-QC for k. In the third round, k + 2, the k proposal can become committed, the (k + 1) proposal can obtain a QC-of-QC, and the (k + 2) can obtain a QC.
LBFT借用了一种在区块链BFT协议中已经流行并由HotStuff推广的链式范式。在链式方法中，承诺的三个阶段是分回合进行的。更具体地说，每一个阶段都是一轮进行的，并包含一个新的提案。k轮的领导人只推动了对其提案的单一阶段的认证。下一轮k+1，某领先者再次带动单期认证。这个阶段有多重目的。k+1发出自己的k+1提案。然而，它也支撑了k提案的质量控制。这样，在k+1轮进行认证会为k+1生成一个QC，为k生成一个QC-of-QC。在第三轮k+2中，k提案可以成为承诺，(k+1)提案可以获得QC-of-QC，(k+2)可以获得QC。

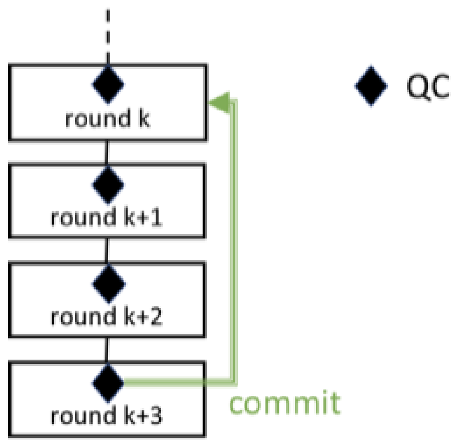


Figure 1: LBFT phases are spread across rounds.

Rounds without certificates Validators may give up on a round by timeout. This may cause transition to the next round without obtaining a QC. The leader of the next round faces a choice as to how to extend the current block-tree it knows. LBFT deviates from HotStuff in that the leader always extend the highest certified leaf with a direct child. The advantage is that the tree of blocks has a uniform structure, every node has a QC for its direct parent.
验证者可能会因超时而放弃回合。这可能会导致过渡到下一轮而没有获得质量控制。下一轮的领导者将面临一个选择，即如何扩展它知道的当前块树。LBFT与HotStuff的不同之处在于，领导者总是通过直系子女延长获得最高认证的叶子。优点是块树具有统一的结构，每个节点的直接父级都有一个QC。

Commit-rule As a consequence of the LBFT chaining approach, the block tree in LBFT may contain “chains” that have gaps in round numbers. Round numbers are explicitly included in blocks, and the resulting commit logic is simple: It requires a 3-chain with contiguous round numbers whose last descendent has been certified.
作为LBFT链式方法的结果，LBFT中的块树可能包含有圆号空隙的“链”。圆号被显式地包含在块中，由此产生的提交逻辑很简单:它需要一个具有连续圆号的3链，其最后一个子代已经被认证。

Round synchronization
In a distributed system, at any moment in time, validators may be in a different state and receive different messages. The issue of bringing validators to enter rounds in approximately the same time is somewhat overlooked in the literature. PBFT gave a theoretical “eventuality” method for round synchronization by doubling the duration of rounds until progress is observed. HotStuff encapsulated the role of advancing rounds in a functionality named PaceMaker, but left its implementation unspecified. In LBFT, when a validator gives up on a certain round (say r), it broadcasts a timeout message carrying a certificate for entering the round. This brings all honest validators to r within the transmission delay bound Δ. When timeout messages are collected from a quorum of validator, they form a timeout certificate (or a TC).
在分布式系统中，在任何时刻，验证者可能处于不同的状态，接收到不同的消息。文献中有些忽略了在大约相同的时间内让验证者进入回合的问题。PBFT给出了一种理论上的回合同步“偶然性”方法，方法是将回合持续时间增加一倍，直到观察到进展。HotStuff在名为PaceMaker的功能中封装了推进回合的角色，但没有具体说明其实现。在LBFT中，当验证器放弃某个回合（例如r）时，它会广播带有进入该回合的证书的超时消息。这将使所有诚实的验证者在传输延迟绑定中r。当超时消息从法定验证器收集时，它们会形成超时证书(或TC)。

LBFT Overview

At a high level, the goal of the LBFT protocol is to commit blocks in sequence.
在高层次上，LBFT协议的目标是按顺序提交块。

The protocol operates in a pipeline of rounds. In each round, a leader proposes a new block. Validators send their votes to the leader of the next round. When a quorum of votes is collected, the leader of the next round forms a quorum certificate (QC) and embeds it in the next proposal. This process continues until three uninterrupted leaders/rounds complete and the tail of a chain has three blocks with consecutive round numbers. Then, the head of the “3-chain” consisting of three consecutive rounds that has formed becomes committed. The entire branch ending with the newly committed block extends the sequence of commits.

将他们的投票发送给下一轮的领导人。当收集到法定人数时，下一轮的领导人形成法定人数证书(QC)，并将其嵌入下一个提案中。这个过程一直持续到三个不间断的领导人/回合完成，一个链条的尾部有三个具有连续圆号的区块。然后，由连续三个回合组成的“3链”的头部成为提交。以新提交区块结束的整个分支扩展了提交的序列。

Figure 2 depicts proposals (blocks), QC’ s, and a commit. The figure displays a tree of blocks, including a fork, rather than a simple chain. Forking can happen for various reasons such as a malicious leader, message losses, and others. For example, the figure shows a Byzantine leader that forked the chain at block k, causing an uncommitted chain to be abandoned. The k block uses the same QC for its parent as the left fork, hence validators vote it. In the depicted scenario, the k block becomes committed, and the left branch is discarded.
图2描绘了提案(块)、QC和提交。该图显示了一棵树块，包括一个叉子，而不是一个简单的链。叉子可能会因为各种原因发生，如恶意领导者、消息损失和其他原因。例如，图中显示了拜占庭领导者在k块分叉了链，导致一个未提交的链被放弃。k块对其父叉使用与左叉相同的QC，因此验证者投票。在所描绘的场景中，k块变成了提交，左分支被丢弃。

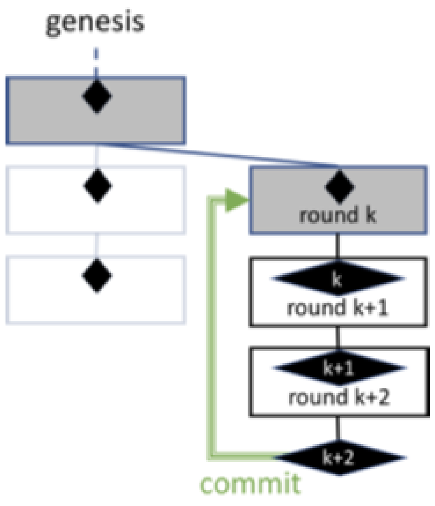


Figure 2: Proposals (blocks) pending in the Block-tree before and after a commit

LBFT guarantees that only one fork becomes committed through a simple voting rule that consists of two ingredients: First, validators vote in strictly increasing rounds. Second, whenever validators receive a block, they maintain a preferred round, defined as the highest known grandparent round. The rule is that validators vote for a block if its parent round is as least the preferred round. In Figure 2, validators that contributed to the formation of a QC for round k + 2 remember k as their preferred round.
LBFT保证只有一个叉子通过一个简单的投票规则被提交，该规则由两个要素组成:首先，验证者在严格增加的回合中投票。第二，每当验证者收到一个区块时，他们都会保持一个首选轮，被定义为已知的最

高祖父轮。规则是，验证者投票支持一个区块，如果它的父轮至少是首选轮。在图2中，有助于形成k+2轮QC的验证者记住k是他们的首选轮。

Consider the scenario depicted in the Figure 3, which is intentionally quite tricky. In the scenario, QCs are formed in rounds k + 1 and k + 2. 2f + 1 validators vote for k + 2 and remember k as their preferred round. Then at round k + 3 the leader experiences a temporary disruption. Round k + 3 times out and another leader, say k + 4 forks the chain at k. This fork obtains 2f + 1 votes because there is no violation of the preferred round rule, and it is followed by k + 5 and k + 6.
考虑图3中描绘的场景，这是故意相当棘手的。在该场景中，QC在k+1和k+2轮中形成。2f+1验证者投票支持k+2，并记住k是他们的首选轮。然后在k+3回合，领先者会经历暂时的中断。回合k+3次出局，另一个领导者，说k+4把链子叉在k。这个叉获得2f+1票，因为没有违反优选轮规则，它后面是k+5和k+6。

In some future round, the leader of round k + 3 becomes leader again and makes use of the QC for k + 2, thus committing the round-k block. Later, another leader may make use of the QC for k + 6 and hence commit k + 4. Again, note that there is no consistency violation, the left-hand fork caused k to commit, and later it is dismissed.
在未来的某个回合中，回合k+3的领导者再次成为领导者，并利用QC为k+2，从而犯了回合k块。后来，另一个领导者可能利用QC为k+6，从而犯了k+4。再次，注意没有违反一致性，左手叉导致k犯了，后来它被驳回。

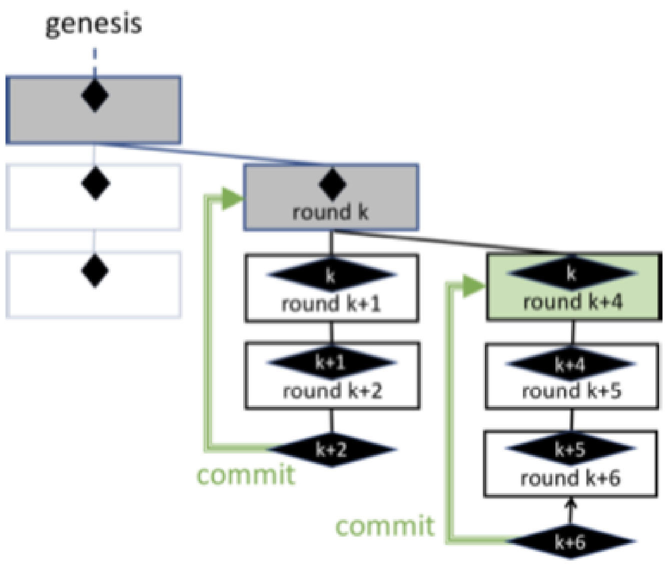


Figure 3: Third block of a three-chain does not necessarily commit

The Protocol in a Nutshell

The logical round-by-round protocol structure above is materialized in the LBFT codebase via a handful of event handlers executed by each validator. Handlers are triggered by either messages or timers.
上面的逻辑逐轮协议结构通过每个验证器执行的少数事件处理程序在LBFT代码库中实现。处理程序由消息或定时器触发。

More specifically, a validator main task is a loop start event processing handling three types of mes- sages, a proposal (block), a vote, and a timeout. It is also triggered when a timer expires.
更具体地说，验证器的主要任务是一个循环启动事件处理，处理三种类型的介绍信、建议（块）、投票和超时。它也会在计时器过期时触发。

For brevity, the description henceforth assumes messages have already been filtered for format and validity of cryptographic values, and that any information indirectly referenced by messages (e.g., a QC for a block) is filled by a lower-level communication substrate.
为了简洁起见，描述假设消息已经被过滤过密码值的格式和有效性，并且消息间接引用的任何信息(例如块的QC)都由较低级别的通信基板填充。

Handling proposals, process proposal msg:
To bootstrap the execution, initially, all validators bootstrap the chain with a fixed genesis block and enter round 1. After the first round, validators enter rounds upon receiving a QC or TC for the previous round; vote and timeout collecting is described below.
为了引导执行，最初，所有验证者用一个固定的创世区块引导链，并进入第一轮。第一轮之后，验证者在收到上一轮的QC或TC时进入一轮；投票和超时收集将在下面描述。

Entering a new round is handled by advance round. Upon entering a new round, the leader for the round extends the chain of blocks with a new proposal and broadcasts it.
进入新一轮由预先一轮处理。进入新一轮后，新一轮的领导者用新提案扩展区块链并广播它。

When a validator receives a leader proposal for the current round k, it invokes process proposal msg to vote for the proposal according to the voting rules.
当验证者收到当前轮k的领先提案时，它会调用进程提案msg，根据投票规则投票支持该提案。

First, it processes the QC or TC for round k－1 the proposal carries, see details below (process certificates). Subsequently, a validator invokes make vote to determine if it can vote for the proposal according to two voting conditions:

首先，它处理提案所携带的k-1轮的QC或TC，详见下文(流程证书)。随后，验证者调用投票，根据以下两种投票条件决定是否可以投票给提案:

1. A validator must vote in monotonically increasing rounds. Each validator maintains and persists a variable last vote round, and votes in round k if it is higher than last vote round. After it sends a vote,
2. 验证者必须以单调递增的方式进行投票。每个验证者在上一轮投票中维持并保持一个变量，如果k值高于上一轮，则在第k轮中投票。在发送投票后，

3. the validator last voting round is advanced to disallow voting again in round k (increase last vote round).
4. 验证者的上一轮投票将提前，以禁止在k轮中再次投票（增加上一轮投票）

When it votes on a block, a validator maintains and persists a variable preferred round, and votes in round k only if the QC inside the k proposal is at least preferred round.
验证者对某个区块投票时，将维持并保留变量的优先回合，并且仅在第k个投标中的QC至少是优先回合时才对第k轮投票。

Last, a validator prepares a speculatively committed state in case block k causes its grandparent to become committed. That is, if block k forms a 3-chain whose rounds numbers are consecutive to the grandparent, then if a QC on k is collected the grandparent will be committed. Therefore, the validator includes in a vote a speculative commitable state commit state id of the grandparent.
最后，如果块k导致其祖父母被提交，验证器将准备一个推测性提交状态。也就是说，如果块k形成3链，其轮数与祖父母连续，则如果收集到k上的QC，则将提交祖父母。因此，验证器在投票中包括祖父母的推测性可提交状态提交状态ID。

Votes are sent to the leader of the next round (k + 1), along with the highest certificate (high qc) known to the validator.
投票连同验证者已知的最高证书（最高qc）一起发送给下一轮的负责人。

Handling votes, process vote msg:
When a validator receives a vote message for round k, it invokes process vote msg to handle it. First, it processes the high qc the vote carries, see details below (process certificates). Second, the Pacemaker aggregates the votes until it collects enough votes to form a QC. The Pacemaker authorizes a transition to the new via advance round, causing the new leader of the new round to generate a proposed block (generate proposal) and to broadcast it.

当验证器收到k轮的投票消息时，它调用进程投票msg来处理它。首先，它处理投票携带的高qc，详见下文(进程证书)。其次，起搏器聚合投票，直到它收集到足够的投票来形成QC。起搏器授权通过提前一轮向新一轮过渡，导致新一轮的新领导人生成一个提议的块(生成提案)并广播它。

Handling timer expiration, local timeout round:
If a validator waits for the leader proposal for round k for a timeout period and it doesn’t arrive, it broadcasts a timeout message carrying its highest certigicate high qc.
如果验证者在超时周期内等待领导者提议的第k轮超时，但它没有到达，它将广播超时消息，该消息带有最高的证书高Qc。

Upon a timer expiration for round k, a validator broadcasts its timeout message to all the participants. A timeout message carries the highest certificate of the validator high qc. Additionally, if the validator entered round k due to a TC, it includes the TC for round k − 1 in the timeout message. Similarly to voting, the last voting round is advanced via increase last vote round to disallow further voting in round k after expiring it.
在第k轮的计时器到期后，验证器会将其超时消息广播给所有参与者。超时消息中包含验证者高Qc的最高证书。另外，如果验证者由于TC进入了回合k，则它将超时消息中包括回合k－1的TC。与投票类似，通过增加最后一轮投票来推进最后一轮投票，以在到期后禁止在第k轮中进行进一步投票。

Handling timeout message, remote timeout:
When a validator receives a timeout message for round k, again it first processes the QC it carries via process certificates, as described below. Second, the leader for round k + 1 collects timeout messages for the preceding round until it forms a TC. Once a leader forms a TC, the leader generates a proposed block via generate proposal and broadcasts it.
当验证器收到k轮的超时消息时，它首先通过流程证书处理它携带的QC，如下文所述。其次，k+1轮的领导者收集上一轮的超时消息，直到它形成TC。一旦领导者形成TC，领导者通过生成提案生成一个提议的块并广播它。

Processing certificates, process certificates:
Upon receiving any message, a validator processes the certificates it carries via process certificates:
在收到任何消息后，验证器通过进程证书处理它携带的证书:

Sync round:
If a validator is currently at the certificate round or lower, it advances to the next round via advance round.
如果验证者当前在证书回合或更低的回合中，则它通过预先回合进入下一回合。

Commit:
If a certificate qc contains a non-empty commitable grandparent state commit state id, then the new committed state is executed and persisted to the ledger store.
如果证书qc包含非空的可提交祖父母状态提交状态ID，则将执行新的提交状态并将其持久保存到分类帐存储中。

Preferred round:
The round of the parent of the certificate becomes “locked” by setting preferred round to it, provided that it is higher than the current preferred round.
证书父级的回合可以通过设置首选回合来“锁定”，前提是该优先级高于当前的首选回合。

LBFT Full Protocol

We proceed with a detailed description of the LBFT protocol, elaborating the data-structures and modules in the implementation. The implementation is broken into the following modules: A Ledger module stores a local, forkable speculative ledger state. A Block-tree module creates proposal blocks and votes. It keeps track of a tree of blocks pending commitment with votes and QC’s on them. A Safety module implements the core consensus safety rules. A Pacemaker module is the liveness keeper that advances rounds. A ProposerElection module maps rounds to leaders. Finally, a Main module is the glue that dispatches messages and timer event handlers.
我们将详细介绍LBFT协议，并详细说明实现中的数据结构和模块。该实现分为以下模块：分类账模块存储本地的，可分叉的投机分类账状态。块树模块创建提案块和投票。它跟踪带有待表决事项的区块树，并对其进行投票和进行质量检查。安全模块执行核心共识安全规则。Pacemaker模块是前进一圈的活力守护者。ProposerElection模块将回合映射到领导者。最后，一个Main模块是分配消息和计时器事件处理程序的粘合剂。

Ledger

Ultimately, the goal of the Libra blockchain is to maintain a database of programmable resources, which the consensus LBFT core replicates. The database is represented by an abstract ledger state. Most of the implementation details of the persistent ledger-store and of the execution VM that applies transactions that mutate ledger state are intentionally left opaque and generic from the point of view of LBFT; in particular, the specifics of Move transaction execution are beyond the scope of this manuscript.

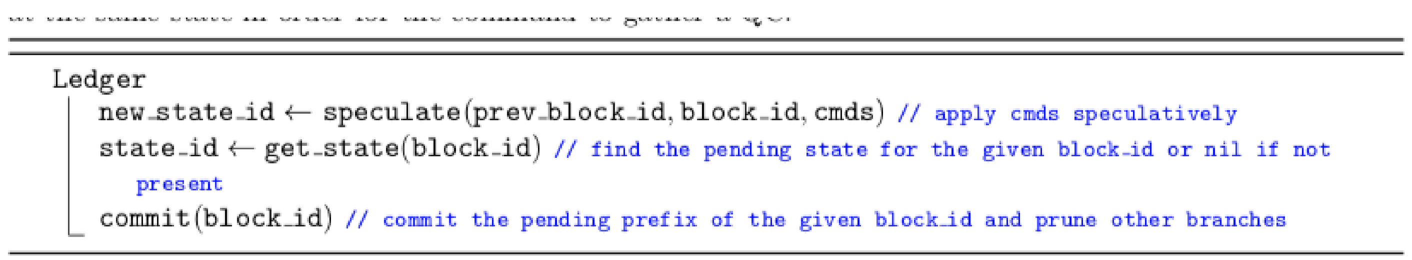
最终，Libra区块链的目标是维护一个可编程资源数据库，共识LBFT核心复制该数据库。该数据库由抽象分类账状态表示。从LBFT的角度来看，持久分类账存储和应用突变分类账状态的事务的执行VM的大多数实现细节都故意留下不透明和通用；特别是，Move事务执行的细节超出了本手稿的范围。

The Ledger module local to each LBFT validator serves as a gateway to the auxiliary ledger-store. It maintains a local pending (potentially branching) speculative ledger state that extends the last committed state. The speculation tree is kept local at the validator in-memory until one branch becomes committed. It provides a mapping from the block-tree (see below), which is pending commitment, to the speculative ledger state that pends commitment.
每个LBFT验证器的本地分类账模块作为辅助分类账存储的网关。它维持一个本地待处理(潜在分支)投机分类账状态，扩展最后一个承诺状态。投机树在内存中的验证器保持本地，直到一个分支成为承诺。它提供了从待定承诺的块树(见下文)到结束承诺的投机分类账状态的映射。

The Ledger.speculate(prev block id, block id, commands) API speculatively executes a block of trans- actions over the previous block state and returns a new ledger state id. Speculative execution potentially branches the ledger state into multiple (conflicting) forks that form a tree of speculative states. Eventually, one branch becomes committed by the consensus engine. The Ledger.commit() API exports to the persistent ledger store a committed branch. Locally, it discards speculated branches that fork from ancestors of the committed state.
Ledger.speculate(prev block id，block id，commands)应用编程接口推测地执行一个对上一个block状态的跨操作块，并返回一个新的分类账状态id。推测执行有可能将分类账状态分支成多个(冲突的)叉，形成推测状态树。最终，一个分支成为共识引擎提交的分支。Ledger.speculate()应用编程接口将一个提交的分支导出到持久分类账存储一个提交的分支。在本地，它丢弃从提交状态祖先分叉的推测分支。

It is important to emphasize that Ledger supports speculative execution in order to enable proper handling of potential non-determinism in execution. If we would build a system that is merely ordering the commands in order to pass them to the execution layer, we would not need to maintain a tree of speculative states because the VM would execute the agreed commands only. However, such a system would not be able to tolerate any non-determinism (e.g., due to a hardware bug): the validators would diverge without LBFT being aware of that. Hence,LBFT goes beyond ordering the commands: it makes sure that the votes certify both the commands and their execution results. There should be at least 2f + 1 honest validators that arrive at the same state in order for the command to gather a QC.
必须强调，分类账支持投机性执行，以便能够正确处理执行中潜在的非决定论。如果我们要构建一个系统，仅仅是命令命令，以便将命令传递给执行层，我们就不需要维护推测状态树，因为虚拟机只会执行商定的命令。然而，这样的系统将无法容忍任何非决定论(例如，由于硬件缺陷):验证者将在LBFT没有意

识到这一点的情况下发生分歧。因此，LBFT不仅仅是命令命令：它确保投票同时证明命令及其执行结果。至少应该有2f+1个诚实的验证者到达相同的状态，以便命令收集一个QC。



```
at the same state in order for the command to gather a QC.

Ledger
  new_state_id ← speculate(prev_block_id, block_id, cmds) // apply cmds speculatively
  state_id ← get_state(block_id) // find the pending state for the given block_id or nil if not
     present
  commit(block_id) // commit the pending prefix of the given block_id and prune other branches
```

Block-tree Module

The Block-tree module consists of two core data-types which the validator protocols is built around, blocks and votes. A third data-type derived from votes is a Quorum Certificate (QC), which consists of a set of votes on a block. An additional data-type concerned solely with timeouts and advancement of rounds is described below in the Pacemaker module. The Block-tree module keeps track of a tree of all blocks pending commitment and the votes they receive.
区块树模块由验证器协议围绕区块和投票构建的两个核心数据类型组成。从投票衍生的第三个数据类型是法定人数证书(QC)，该证书由一组区块的投票组成。起搏器模块下面描述了一个仅涉及超时和回合推进的额外数据类型。区块树模块跟踪所有等待承诺的区块树及其收到的投票。

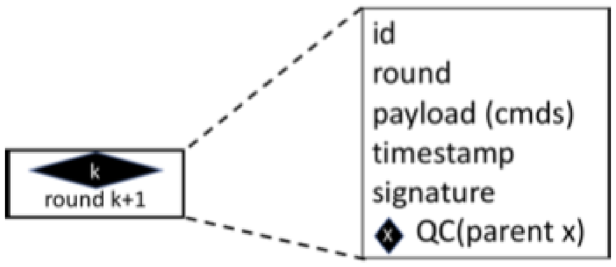


Figure 4: A block in Block-tree

Blocks. The core data structure used by the consensus protocol for forming agreement on ledger transac- tions is a Block. Each block contains as payload a set of proposed Ledger transactions, as well as additional information used for forming consensus decisions.
共识协议用于形成分类账交易协议的核心数据结构是块。每个块作为有效负载包含一组提议的分类账交易，以及用于形成共识决策的额外信息。

Every block b (except for a known genesis block P0) is chained to a parent via b.parent qc, a Quorum Certificate (QC) that consists of a quorum of votes for a parent block. In this way, the blocks pending commitment form a tree of proposed blocks rooted at P0.
每一个b块（除了已知的起源块P0）都通过b. parent qc（一种法定人数证书（qc））链接到父块，该证书由父块的法定人数投票组成。这样，等待承诺的块形成了植根于P0的拟议块树。

```
Block-tree
  Block
    round ; // the round that generated this proposal
    payload ; // proposed transaction(s)
    parent_qc ; // qc for parent block
    id ; // unique digest of round, payload and parent_qc.id
```

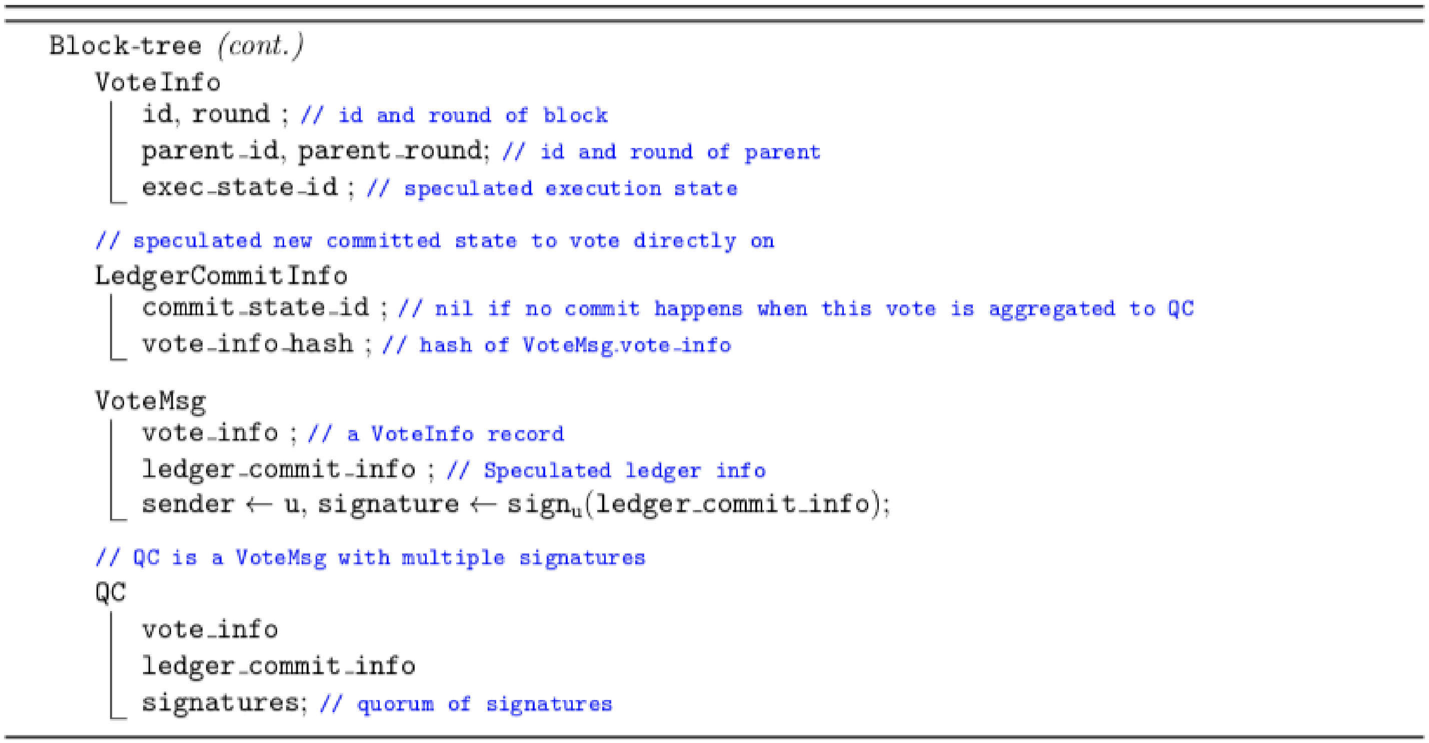
Votes and Certificates. Vote information VoteInfo for a block b must include both the block id id and an execution state id exec state id in order to guarantee a deterministic execution outcome. Additionally and (redundantly), VoteInfo holds the id’ s and rounds of b’ s parent. This redundant information is kept for convenience, allowing to infer commitment from a single block without fetching its ancestors. This also allows to execute the core safety logic of LBFT within a TCB autonomously and determine commitment decisions without IO (see 8).
b块的投票信息VoteInfo必须包括块id和执行状态id exec状态id，以保证确定性的执行结果。此外(冗余)，VoteInfo持有b父的id和回合。为了方便，保留了这些冗余信息，允许从单个块推断承诺，而不需要获取其祖先。这也允许在TCB内自主执行LBFT的核心安全逻辑，并在没有IO的情况下确定承诺决策(见8)。

In addition, a vote message includes LedgerCommitInfo, a speculated committed ledger state, identified by commit state id. When a QC for a block results in a commit of its grandparent, the quorum of votes on the block also serve to certify the new committed ledger state.
此外，投票消息包括LedgerCommitInfo,，这是一个推测的承诺分类账状态，由承诺状态id识别。当一个块的QC导致其祖父母的承诺时，该块上的法定票数也用于证明新的承诺分类账状态。

LedgerCommitInfo serves two purposes. First, it includes the commit state id, which can be given to the clients as a proof of history (in practice commit state id can be a hash of a root of Merkle tree that covers the history of the ledger). Clients should not be aware of the specifics of consensus protocol for as long as they are able to verify that the given ledger state is signed by 2f +1 participants. Second, LedgerCommitInfo includes the hash of VoteInfo. This hash is opaque to the clients and is used by Consensus participants. A validator that signs its vote message is thus

authenticating both the potential LedgerInfo (to be stored by the ledger as a proof of commit) and the actual vote decision (to be used for running the Consensus protocol).
LedgerCommitInfo有两个目的。首先，它包括提交状态id，它可以作为历史证明给予客户端（实际上提交状态id可以是覆盖分类账历史的Merkle树根的哈希）。客户端不应该知道共识协议的细节，只要他们能够验证给定分类账状态是由2f+1参与者签署的。其次，LedgerCommitInfo包括VoteInfo的哈希。这个哈希对客户端是不透明的，并且被共识参与者使用。因此，签名其投票消息的验证器正在认证潜在的Ledger Info（将由分类账存储作为承诺的证明）和实际投票决定（用于运行共识协议）。

Consider, for example, a proposal b in round k with parent b′ in round k − 1 and grandparent b′′ in round k − 2. In case a validator decides to vote for b, it signs a LedgerCommitInfo that includes both the potential commit of b′′ as well as the hash of VoteInfo on b.
例如，考虑k回合中的提案b，k回合1中的父本b′，k回合2中的父本b′。如果验证者决定投票支持b，它会签署一个LedgerCommitInfo，其中包括b′的潜在提交以及b上的VoteInfo的散列。



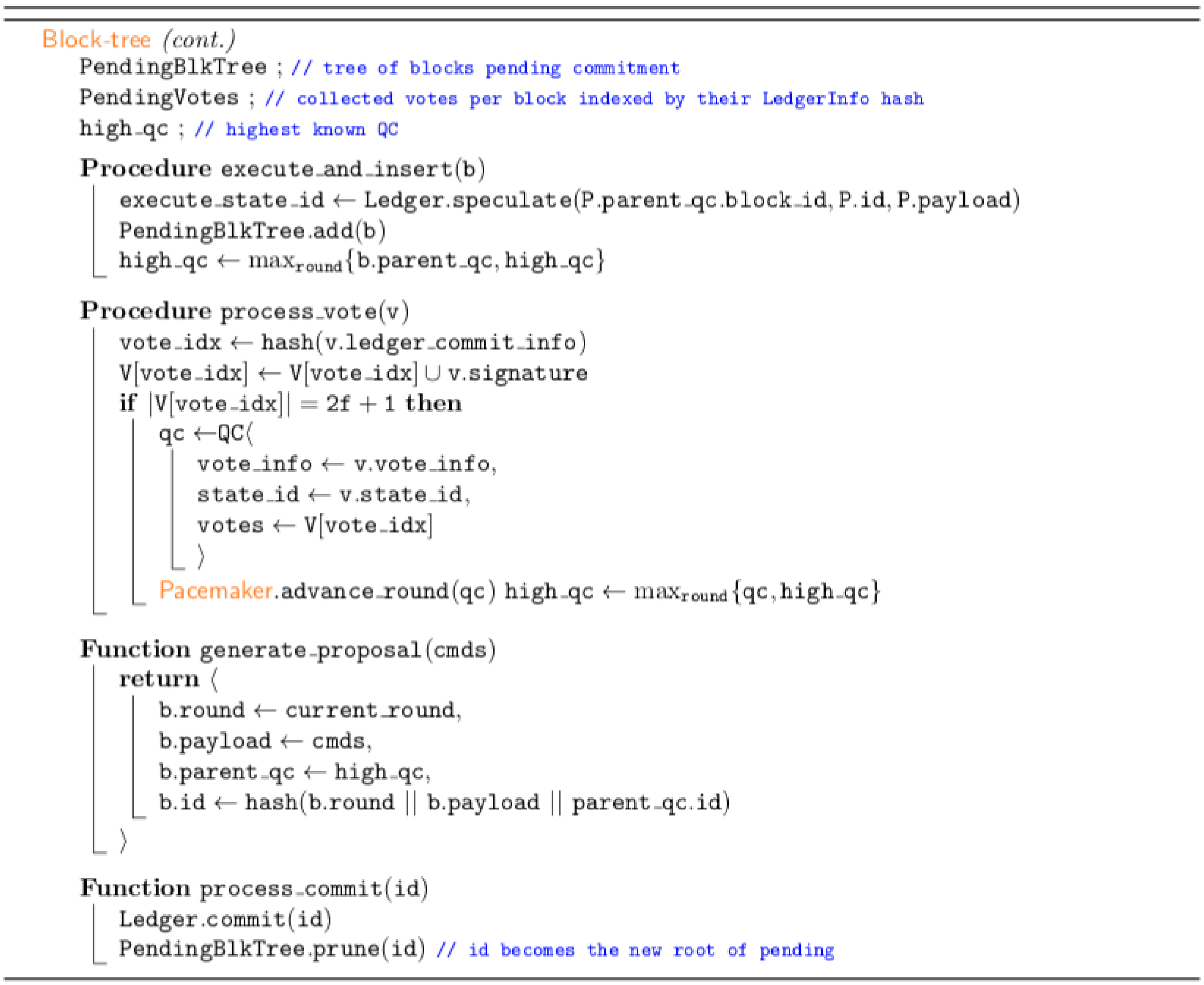
```
Block-tree (cont.)
    VoteInfo
        id, round ; // id and round of block
        parent_id, parent_round; // id and round of parent
        exec_state_id ; // speculated execution state
    // speculated new committed state to vote directly on
    LedgerCommitInfo
        commit_state_id ; // nil if no commit happens when this vote is aggregated to QC
        vote_info_hash ; // hash of VoteMsg.vote_info
    VoteMsg
        vote_info ; // a VoteInfo record
        ledger_commit_info ; // Speculated ledger info
        sender ← u, signature ← sign_u(ledger_commit_info);
    // QC is a VoteMsg with multiple signatures
    QC
        vote_info
        ledger_commit_info
        signatures; // quorum of signatures
```

PendingBlkTree and PendingVotes The Block-tree module tracks blocks pending commitment in PendingBlkTree and votes in PendingVotes. The PendingBlkTree builds a speculative tree of blocks similarly to the Ledger building a speculative tree of states. In fact there is a 1:1 mapping between a block in PendingBlkTree and a block in Ledger. When a new block is added to the PendingBlkTree it is also automatically added to the Ledger.
Block-tree模块跟踪Pending Blk Tree中待承诺的块，并在Pending Votes中投票。Pending Blk Tree构建了一个投机性的块树，类似于Ledger构建一个投机性的状态树。事实上，Pending Blk Tree中的块和

Ledger中的块。当一个新块添加到PendingBlk Tree时，它也会自动添加到Ledger中。

Votes are aggregated in PendingVotes based on the hash of the ledger commit info. Once there are 2f + 1 votes, they form a QC. As mentioned above, ledger commit info includes information both about a potential commit and vote info, hence it is important to make sure that all vote messages refer to the same vote and commit (it would not be enough to aggregate votes just by the id of the proposal).
投票是根据分类账提交信息的散列在Pending Votes中汇总的。一旦有2f+1票，它们就会形成QC。如上所述，分类账提交信息包括关于潜在提交和投票信息的信息，因此确保所有投票信息都指同一个投票和提交是很重要的(仅凭提案的id汇总投票是不够的)。

The algorithm maintains the highest known certified block in high qc, updating it when a new QC is formed or received as part of the proposal. New proposals extend the highest certified block known locally to the validator.
该算法在高质量控制中保持已知的最高认证块，当新质量控制形成或作为建议的一部分收到时，更新该块。新建议将本地已知的最高认证块扩展到验证器。



```
Block-tree (cont.)
    PendingBlkTree ; // tree of blocks pending commitment
    PendingVotes ; // collected votes per block indexed by their LedgerInfo hash
    high_qc ; // highest known QC
    Procedure execute_and_insert(b)
        execute_state_id ← Ledger.speculate(P.parent_qc.block_id, P.id, P.payload)
        PendingBlkTree.add(b)
        high_qc ← $\max_{round}$ {b.parent_qc, high_qc}
    Procedure process_vote(v)
        vote_idx ← hash(v.ledger_commit_info)
        V[vote_idx] ← V[vote_idx] ∪ v.signature
        if |V[vote_idx]| = 2f + 1 then
            qc ←QC⟨
                vote_info ← v.vote_info,
                state_id ← v.state_id,
                votes ← V[vote_idx]
            ⟩
            Pacemaker.advance_round(qc) high_qc ← $\max_{round}$ {qc, high_qc}
    Function generate_proposal(cmds)
        return ⟨
            b.round ← current_round,
            b.payload ← cmds,
            b.parent_qc ← high_qc,
            b.id ← hash(b.round || b.payload || parent_qc.id)
        ⟩
    Function process_commit(id)
        Ledger.commit(id)
        PendingBlkTree.prune(id) // id becomes the new root of pending
```

Safety Module

Pacemaker Module

The advancement of rounds is governed by a module called Pacemaker. The Pacemaker keeps track of votes and of time.
回合的推进由一个叫做起搏器的模块控制。起搏器跟踪投票和时间。

In a “happy path”, the Pacemaker module at each validator observes progress, a leader proposal for the current round, and advances to the next round.
在“快乐的道路”中，每个验证器的起搏器模块观察进展，为当前一轮提出领先建议，并推进到下一轮。

In a “recovery path”, the Pacemaker observes lack of progress in a round.
在“恢复路径”中，起搏器观察到一轮缺乏进展。

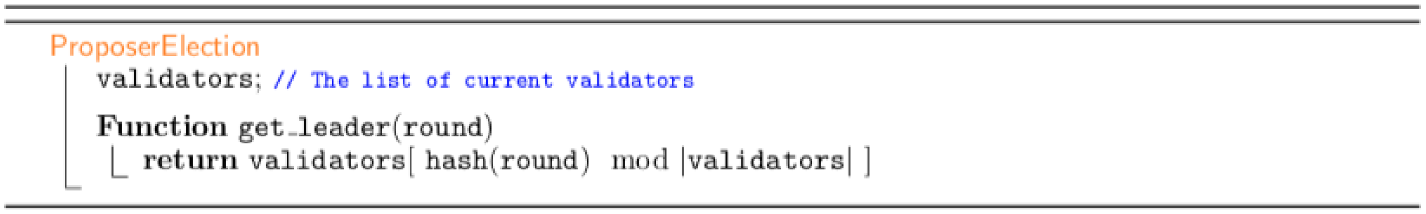
Upon a local round timeout, the Pacemaker broadcasts a TimeoutMsg notification.
在本地轮超时时，起搏器会广播一个Time out Msg通知。

Whenever the Pacemaker receives a certificate, either a quorum certificate (QC) or a timeout certificate (TC), it instructs the validator to advance to the certificate’s round (Main.process new round event). It additionally instructs the validator to forward the QC/TC to the leader of round. This guarantees that upon entering a round with an honest leader, all validators will synchronize to within two network transmission delays.
每当起搏器收到证书时，无论是法定人数证书(QC)还是超时证书 (TC)，它指示验证者推进证书的回合(Main. process new Round事件)。此外，还指示验证者将QC/TC转发给回合的领导者。这保证在与诚实的领导者进入回合时，所有验证者将同步到两个网络传输延迟内。

```
Pacemaker
  current_round; // initially zero
  T[] ; // timeouts per round
  TimeoutMsg
      round ; highqc ← PendingBlkTree.get_high_qc() ; // sender's highest qc added automatically
      sender ← $u$, signature ← sign$_u$(round); // sender and signature added automatically
  Procedure local_timeout_round
      if u $\notin$ T[current_round] then
          Safety.increase_last_vote_round(current_round) // stop voting for current_round
          save_consensus_state()
          broadcast TimeoutMsg(current_round)
          T[current_round] ← T[current_round] ∪ {u}
  Procedure process_remote_timeout(tmo)
      T[tmo.round] ← T[tmo.round] ∪ {tmo.sender}
      // A timeout certificate (TC)
      if |T[tmo.round]| == $2f + 1$ then advance_round(T[tmo.round])
  Procedure advance_round(qc)
      $r$ ← qc.round
      if r < current_round then return
      stop local timer for round r
      current_round ← r + 1
      if u ≠ ProposerElection.get_leader(current_round) then
          send qc to ProposerElection.get_leader(current_round)
      start local timer for round current_round for duration get_round_timer(current_round)
      Main.process_new_round_event()
  Function get_round_timer(r)
      return round timer formula // for example, use $4 \times \Delta$ or $\alpha + \beta^{commit\_gap(r)}$ if $\Delta$ is unknown.
```

ProposerElection Module

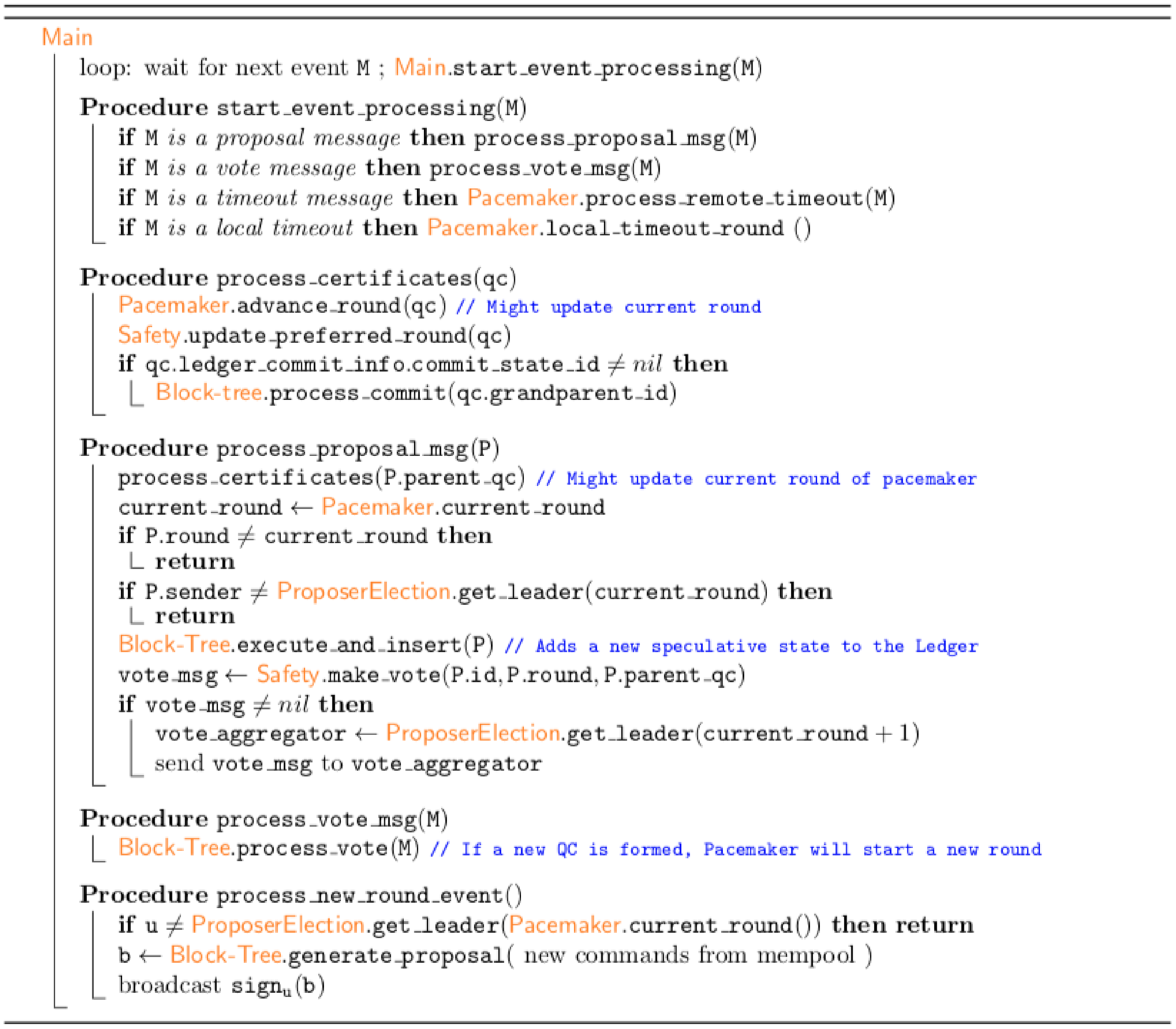
ProposerElection emits a validator for a given round in a deterministic fashion, s.t. different honest par- ticipants would all implicitly agree on the chosen leader per round. The leader is determined by the hash of the given round: as a result different combinations of successive leaders occur in the system with equal probability.
ProposerElection以确定性的方式发出给定回合的验证器，每个回合的诚实参与者都将隐式地同意所选回合的领导者。领导者由给定回合的哈希值决定：结果，系统中连续领导者的不同组合发生的可能性相等。



```
ProposerElection
    validators; // The list of current validators
    Function get_leader(round)
        return validators[ hash(round) mod |validators| ]
```

Main Module

The main body of LBFT is an event-handling switch that runs an endless loop, and invokes appropriate handlers to process messages and events. The following (handful) of high-level events are handled: a propose message, a vote message, a remote timeout message, a local timeout, and a new round (as leader).
LBFT的主体是一个事件处理交换机，它运行一个无穷无尽的循环，并调用适当的处理程序来处理消息和事件。处理以下(少数)高级事件:提议消息、投票消息、远程超时消息、本地超时和新一轮(作为领导者)。



```
Main
    loop: wait for next event M ; Main.start_event_processing(M)
    Procedure start_event_processing(M)
        if M is a proposal message then process_proposal_msg(M)
        if M is a vote message then process_vote_msg(M)
        if M is a timeout message then Pacemaker.process_remote_timeout(M)
        if M is a local timeout then Pacemaker.local_timeout_round ()
    Procedure process_certificates(qc)
        Pacemaker.advance_round(qc) // Might update current round
        Safety.update_preferred_round(qc)
        if qc.ledger_commit_info.commit_state_id $\neq$ $nil$ then
            Block-tree.process_commit(qc.grandparent_id)
    Procedure process_proposal_msg(P)
        process_certificates(P.parent_qc) // Might update current round of pacemaker
        current_round $\leftarrow$ Pacemaker.current_round
        if P.round $\neq$ current_round then
            return
        if P.sender $\neq$ ProposerElection.get_leader(current_round) then
            return
        Block-Tree.execute_and_insert(P) // Adds a new speculative state to the Ledger
        vote_msg $\leftarrow$ Safety.make_vote(P.id, P.round, P.parent_qc)
        if vote_msg $\neq$ $nil$ then
            vote_aggregator $\leftarrow$ ProposerElection.get_leader(current_round + 1)
            send vote_msg to vote_aggregator
    Procedure process_vote_msg(M)
        Block-Tree.process_vote(M) // If a new QC is formed, Pacemaker will start a new round
    Procedure process_new_round_event()
        if u $\neq$ ProposerElection.get_leader(Pacemaker.current_round()) then return
        b $\leftarrow$ Block-Tree.generate_proposal( new commands from mempool )
        broadcast $sign_u$(b)
```

Persisted Validator Information

In addition to the ledger state, which is abstracted away within the Ledger module, a validator needs to per- sist information about its voting history in order to maintain safety. The size of the state that needs to be per- sisted for safety is quite small, and consists of merely a few elements, persisted via save consensus state() as follows:
除了在分类账模块中被抽象化的分类账状态之外，验证者还需要保持其投票历史记录的信息以保持安全。为了安全需要保持的状态的大小相当小，仅由几个元素组成，通过保存共识状态()保持如下：

```
Procedure save_consensus_state()
    persist last_vote_round
    persist prefered_round
    persist PendingBlkTree // make voted information recoverable for liveness, not needed for safety
```

Omitted, gory details

We encapsulate several tasks related to wire-protocol and defer it to a transport substrate that will be described elsewhere. The transport takes care of formatting messages and serializing them for transmission over the wire, for reliably delivering messages, and for retrieving any data referenced by delivered messages, in particular, block ancestors. In particular, when a message is handled in the pseudo code, we assume that its format and signature has been validated, and that the receivers has synced up with all ancestors and any other data referenced by meta-information in the message.
我们封装了与导线协议有关的几个任务，并将其推迟到其他地方将要描述的传输衬底上。传输负责格式化消息并对其进行序列化，以便通过电线进行传输，可靠地传递消息，以及检索所传递的消息（尤其是块祖先）所引用的任何数据。尤其是，当使用伪代码处理消息时，我们假定其格式和签名已经过验证，并且接收者已与所有祖先以及消息中元信息引用的任何其他数据同步。

Liveness Proof and Latency Bounds

Optimistic Responsiveness

We first prove optimistic responsiveness, namely that when an honest leader gets an opportunity to make progress – all honest validators wait for its proposal – they will accept and vote for it. In order to satisfy optimistic responsiveness, we need to demonstrate that at any round, if the leader is honest and a validator receives the leader proposal by the leader without the round timer expiring, then it accepts (votes) for the proposal.
我们首先证明乐观的响应性，即当一个诚实的领导者获得了一个取得进展的机会-所有诚实的验证者等待它的提议-他们将接受并投票支持它为了满足乐观的响应性，我们需要证明在任何一轮中，如果领导者是诚实的，并且验证者在没有回合计时器到期的情况下接受了领导者的提议，那么它就接受（投票）支持该提议。

Property 1. If an honest validator has a preferred round = r0 when it enters round r, then there exist f + 1 honest validators, which had highest 1-chains at round rhqc >= r0 at round r′ < r.

如果诚实验证器在进入r回合时有一个首选的回合=r0，则存在f+1诚实验证器，该验证器在r′<r回合时有最高的1链>=r0。

Indeed, preferred round = r0 implies a 2-chain: there are 2f + 1 validators that voted for a block carrying the QC with round r0. This voting happened in some round r′ < r, hence following our BFT assumptions there exist f + 1 honest validators with highest 1-chain equal or greater than r0 before they enter round r.
事实上，首选的Round=r0意味着一个2链:有2f+1验证器投票支持带有r0的QC区块。这种投票发生在一些r′<r的回合中，因此根据我们的BFT假设，在进入轮r之前，存在f+1诚实验证器，它们的最高1链等于或大于r0。

In order to complete the optimistic responsiveness argument, consider an honest validator starting round r with preferred round r0: by Property 1 some f +1 honest validators had 1-chains at least r0 before entering r. Since the leader gathered the highest 1-chains from 2f + 1 validators before entering r, there should be at least 1 honest validator that sent a 1-chain with round at least r0 to the leader. Thus, a new proposal by the honest leader at round r carries a QC with round at least r0, hence it passes the preferred round rule and should be accepted by this honest validators.
为了完成乐观的响应性论点，考虑一个诚实的验证者从优先的r0开始一轮r:在属性1的时候，一些f+1诚实的验证者在输入r之前至少有1-链。由于领导者在输入r之前从2f+1验证器中收集了最高的1链，因此至少应该有一个诚实的验证器向领导者发送了一个至少r0的1链。因此，在r回合中由诚实的领导者提出的新建议带有至少r0回合的QC，因此它通过了首选的回合规则，应该被这个诚实的验证者接受。

Liveness under Synchrony

We first analyze liveness/latency under synchrony. Assume a known bound Δ on messages transmission delays among honest validators, and let Pacemaker.set round timeout() return a fixed value 4Δ for all rounds.
我们首先分析同步下的活跃度/延迟。假设诚实验证者之间消息传输延迟的已知界为Δ，并让Pacemaker.set Round timeout()为所有回合返回一个固定值4Δ。

Round synchronization. Under synchrony, the Pacemaker maintains round synchronization defined as follows:
在同步状态下，起搏器保持循环同步，定义如下:

Property 2. If round r has an honest leader, then all honest validators enter round r within a period of 2Δ from each other.

如果回合r有一个诚实的领导者，那么所有诚实的验证者都在彼此的2个周期内进入回合r。

Indeed, an honest validator enters round r only upon receiving a QC containing 2f + 1 round-(r − 1) votes or a TC containing 2f + 1 round-(r − 1) timeouts. In the former case, the certificate is sent by the r leader, hence it arrives within Δ to all honest validators. In the latter, the first honest validator to receive a TC for round (r − 1) forwards it to the (honest) leader, hence all remaining honest validators enter round r within 2Δ time.
事实上，诚实验证者只有在收到包含2f+1回合-(r-1)投票的QC或包含2f+1回合-(r-1)超时的TC时才进入r回合。在前一种情况下，证书是由r领导者发送的，因此它在1.2内到达所有诚实验证者。在后者中，第一个收到回合(r-1)的诚实验证者将其转发给(诚实)领导者，因此所有剩余的诚实验证者在2.2时间内进入r回合。

A corollary of Property 2 is that all honest validators overlap in at least 2Δ period at round r.
属性2的推论是，所有诚实验证者在r轮至少2个周期内重叠。

By the corollary to Property 2, an honest leader and all honest validators overlap in at least 2Δ period in the leader round. Due to the optimistic responsiveness property, the honest validators will accept the leader proposal and vote for it.
根据属性2的推论，一个诚实的领导者和所有诚实的验证者在领导者回合中至少有2个周期重叠。由于乐观的响应性属性，诚实的验证者将接受领导者的建议并投票支持它

We call a round three-chainer if it has an honest leader and three subsequent leaders are also honest. Applying the above two properties four times is succession, we get that the three-chainer proposal becomes committed.
如果它有一个诚实的领导者，三个随后的领导者也是诚实的，我们称之为圆形三链。四次应用上述两个属性是继承，我们得到三链提议成为承诺。

We proceed to provide a concrete latency bound under synchrony. The latency to arrive at a new commit is impacted by two factor, the time to bring all honest validators to execute the same round after bad leaders, multiplied by the number of rounds to get a succession of four honest leaders.
我们继续提供一个在同步下绑定的具体延迟。到达新提交的延迟受到两个因素的影响，即在糟糕的领导者之后让所有诚实验证者执行同一轮的时间，乘以获得连续四个诚实领导者的回合数。

Property 3. If rounds r,r+1,r+2,r+3 have honest leaders, where r is the highest round any single honest validator has entered, then r becomes committed within 4 × 6Δ time.如果r、r+1、r+2、r+3轮有诚实的领导者，其中r是任何一个诚实验证者进入的最高一轮，那么r在 4 × 6Δ 时间内被承诺。

By Property 2, all honest validators enter round r within Δ period from each other. Furthermore, no honest validator round timer expires on any of the four rounds r, r + 1, r + 2, r + 3, hence the round proposal commits. Since each round takes 3Δ, accounting for the additional 2Δ skew, the succession takes at most 4×(4Δ+2Δ).
通过属性2，所有诚实验证者在r周期内相互进入回合r。此外，没有诚实验证者回合计时器在四个回合r、r+1、r+2、r+3中的任何一个回合过期，因此回合建议提交。由于每个回合需要3 Δ，占额外的2 Δ，继承最多需要4×(4Δ +2 Δ)。

It is worth noting that the TC broadcast by honest validators is crucial for this latency bound. Without the broadcast, f + 1 validators might advance jointly with f bad validator one round ahead of the remaining f honest, and the f honest never catch up.
值得注意的是，诚实验证者的TC广播对这个延迟约束至关重要。如果没有广播，f+1验证者可能会与f坏验证者共同前进，领先剩余的f诚实者一轮，而f诚实者永远不会赶上。

The expected time e to arrive at an honest succession of four leaders varies and depends largely on the leader rotation regime, the number of actual faults, and their relation to the rotation scheme. For example, if f actual validator faults occur uniformly at random among n, e ≈ 12.
四个领导者诚实接班的预期时间各不相同，主要取决于领导者轮换制度、实际故障数量及其与轮换方案的关系。例如，如果实际验证器故障在n中均匀随机发生，则e≈12。

In total, the expected latency is (e × 3Δ) + (4 × 6Δ). Furthermore, suppose that δ is the actual network latency. We already showed that all honest validators enter the highest round any of them is it within Δ. Once they reach an honest 3-chain, it is easy to show (simply replace Δ above with δ) that is takes only 8δ for the 3-chain to become committed.
总的来说，预期的延迟是(e×3个)+(4×6个)。此外，假设δ是实际的网络延迟。我们已经证明了所有诚实验证者进入最高的一轮，其中任何一轮都是在1以内。一旦他们到达诚实的3链，很容易显示(简单地用δ替换上面的) 3链只需要8δ就可以被承诺。

Note that the above latency bound is incurred in expectation on when the maximal number of faults is reached. Below, we elaborate on an enhancement called “nil blocks” that facilitates faster commits by individual honest leaders 8.
请注意，当达到最大故障数量时，会产生上述延迟约束。下面，我们详细介绍一种名为“零块”的增强，这种增强有助于个人诚实的领导者更快地完成8。

Liveness and Latency under Partial Synchrony.

What happens when Δ is not known (partial synchrony)? Pretty much the same except that validators need to “guess” Δ by gradually increasing their round duration.
当“不知道”（部分同步）时会发生什么？几乎相同，只是验证者需要通过逐渐增加他们的回合持续时间来“猜测”“。

Validators can choose round duration as β+αcommit gap(r), where commit gap(r) is the number of rounds preceding r not known to commit minus 2 (since commits are always delayed two rounds). At any moment in time, different validators may have different views of the last commit, hence their round durations may vary. Eventually, all of them will have round durations at least Δ but the round timeout value is unbounded. That is, all honest validators will guarantee to enter new rounds within 2Δ delay, but the time they spend in a round until a timeout is unbounded. Therefore, the latency bound in a partial synchrony settings is the following property:
验证者可以选择回合持续时间为β+α提交间隙(r)，其中提交间隙(r)是不知道提交前r的回合数减2(因为提交总是延迟两轮)。在任何时刻，不同的验证者可能对最后一次提交有不同的看法，因此他们的回合持续时间可能会有所不同。最终，它们都将至少有圆形持续时间，但圆形超时值是无界的。也就是说，所有诚实的验证者都将保证在延迟2秒内进入新一轮，但他们在一轮中直到超时的时间是无界的。因此，在部分同步设置中绑定的延迟是以下属性：

Again, once validators synchronize at a 3-chain round, they will progress to commit at network speed.
再次，一旦验证者以3链轮同步，他们将以网速进行提交。

Safety Proof

In this section we start by defining some notation that we will use:

- $B_i \longleftarrow^* B_j$ means that the block B_j extends the block B_i.
- $B_i \longleftarrow QC_i \longleftarrow B_{i+1}$ means that the block B_i is certified by the quorum certificate QC_i which is contained in the block B_{i+1}.

- round(B_i) reads as "the round number of block B_i".
- previous_round(B_i) := round(B_{i-1}) for $B_{i-1} \longleftarrow QC_{i-1} \longleftarrow B_i$.
- preferred_round(N, B_i) reads as "the preferred round of validator N after voting for block B_i".

Let's now recapitulate the BFT assumption which we need to prove the safety of our protocol.

Definition 1 (BFT Assumption)**.** For all f dishonest validators, there exist $2f+1$ honest validators.

The following classical lemma can be derived from the assumption.

Lemma 1. Under BFT assumption, for all blocks B, B' and quorum certificates QC, QC', if all the following properties are true

- $B \longleftarrow QC$
- $B' \longleftarrow QC'$
- round(B) = round(B')

then we have that $B = B'$. In other words there can only be one certified block per round.

Proof:

1. By definition, quorum certificates have at least $2f+1$ votes. For a quorum certificate C let's define honest(C) as the set of honest validators that have participated in QC. Due to the *BFT assumption*, For all quorum certificate C we have

$$\text{honest}(C) \geq f+1$$

2. Following *statement 1*, we have

$$|\text{honest}(QC) \cap \text{honest}(QC')| \geq (f+1) + (f+1) - (2f+1) \geq 1$$

3. Following *statement 2*, There exists an honest node hn such that $hn \in QC$ and $hn \in QC'$. Since round(B) = round(B'), by *voting rule 1* hn could have only voted for one block in this round. This implies that $B = B'$.

We can now define what safety means formally.

Definition 2 (Safety)**.** LBFT is **safe** if $\forall\ B, \widetilde{B} \in Blocks$, such that B is committed at round k and $\widetilde{B}$ is committed at round $\widetilde{k} > k$, then $B \longleftarrow^* \widetilde{B}$.

Let's now state and prove the safety theorem.

Theorem 2. *LBFT is safe under BFT assumption.*

Proof:

1. Two blocks B_0, B_0' are committed if there exist two chains:

   - $B_0 \longleftarrow QC_{B_0} \longleftarrow B_1 \longleftarrow QC_{B_1} \longleftarrow B_2 \longleftarrow QC_{B_2}$
   - $\widetilde{B_0} \longleftarrow QC_{\widetilde{B_0}} \longleftarrow \widetilde{B_1} \longleftarrow QC_{\widetilde{B_1}} \longleftarrow \widetilde{B_2} \longleftarrow QC_{\widetilde{B_2}}$

   with both the chains having contiguous rounds, meaning that:

   - round(B_2) = round(B_1) + 1 = round(B_0) + 2
   - round($\widetilde{B_2}$) = round($\widetilde{B_1}$) + 1 = round($\widetilde{B_0}$) + 2

We can assume round($\widetilde{B_0}$) $\geq$ round(B_0) without loss of generality.

2. **It suffices to prove** that $\forall$ B_0 and $\widetilde{B_0}$ such that round($\widetilde{B_0}$) $\geq$ round(B_0) and both blocks are committed, we have $B_0 \longleftarrow^* \widetilde{B_0}$.

3. Let B a certified block, using *lemma 1* the following statements are true:
   - round(B) = round(B_0) $\implies B = B_0$
   - round(B) = round(B_1) $\implies B = B_1$
   - round(B) = round(B_2) $\implies B = B_2$

4. Following *statement 2*, we have either of the **initial states**:
   - round(B_0) $\leq$ round($\widetilde{B_0}$) $\leq$ round(B_2), and by *statement 3* $\widetilde{B_0}$ is either B_0, B_1, or B_2 and $B_0 \longleftarrow^* \widetilde{B_0}$,
   - or round($\widetilde{B_0}$) $>$ round(B_2).

5. Let block $\widetilde{B_i}$ such that round($\widetilde{B_i}$) $>$ round(B_2) and let block $\widetilde{B_{i-1}}$ = previous_round($\widetilde{B_i}$), then either
   - round(B_0) $\leq$ round($\widetilde{B_{i-1}}$) $\leq$ round(B_2), and by *statement 3* $\widetilde{B_{i-1}}$ is either B_0, B_1, or B_2 and $B_0 \longleftarrow^* \widetilde{B_{i-1}}$,
   - or round($\widetilde{B_{i-1}}$) $>$ round(B_2).

   Proof:
   5.1. It suffices to prove that if round($\widetilde{B_i}$) $>$ round(B_2), then round($\widetilde{B_{i-1}}$) $\geq$ round(B_0).

   5.2. Due to quorum certificates having at least $2f + 1$ votes, the intersection of any two quorum certificates has at least one honest validator.

   5.3. By *statement 5.2*, There exists an honest validator hv such that $hv \in QC_{B_2}$ and $hv \in QC_{\widetilde{B_i}}$.

   5.4. By the *voting rule 2*, after hv observed B_2, preferred_round(hv, B_2) $\geq$ round(B_0).

   5.5. Since round($\widetilde{B_i}$) $>$ round(B_2), by the *voting rule 1* hv could only vote for $\widetilde{B_i}$ if

   previous_round($\widetilde{B_i}$) $\geq$ preferred_round(hv, B_2)
   $\Leftrightarrow$ round($\widetilde{B_{i-1}}$) $\geq$ round(B_0).

6. $\forall$ blocks B_i, B_{i-1} such that B_{i-1} = previous_round(B_i), then round(B_{i-1}) $<$ round(B_i).

7. By **induction** using *statements 4, 5 and 6*, $B_0 \longleftarrow^* \widetilde{B_0}$.

Next Steps

In this section, we briefly touch on some roadmap topics that LibraBFT will include in the future. More details about each topic will appear in coming documents.
在本节中，我们简要介绍了LibraBFT未来将包括的一些路线图主题。关于每个主题的更多细节将出现在未来的文件中。

Nil proposals.
If a QC for a round (say k) cannot be formed, a leader of a higher round must present a timeout certificate (TC) from a quorum of validators in order to move into the round. If the TC contains timeout messages from validators, such that each of this quorum of validators did not vote in round k, then it can serve as a QC for an implicit “nil proposal”.
如果不能形成一轮的QC(例如k)，更高一轮的领导者必须提交法定人数验证人的超时证明(TC)才能进入该轮。如果TC包含验证人的超时消息，以至于该法定人数验证人的每个人都没有在k轮投票，那么它可以作为隐含的“无提议”的QC

The benefit of this enhancements is that expired rounds contribute to fast(er) commitment, as depicted in Figure 5. Note that nil proposals can be chained, just like normal blocks.
这种增强的好处是，过期的轮次有助于快速承诺，如图5所示。请注意，可以像普通区块一样，将无提议上链。

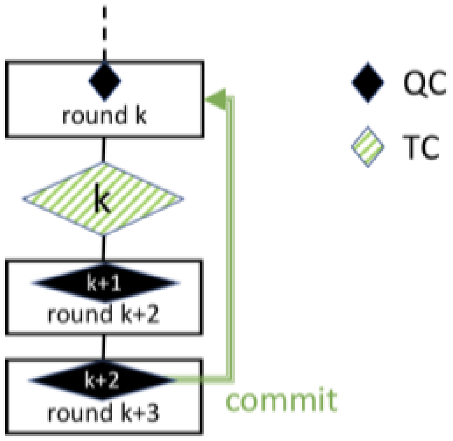


Figure 5: Nil blocks can contribute to fast(er) commitment.

Epoch changes.
LBFT can reconfigure itself by embedding configuration-change commands in the se- quence. Blocks on the branch extending a configuration-change command are proposed and voted on only for the purpose of commiting the reconfiguration, but they must not contain normal transaction payload (nor will transactions be they contain get executed). This is essentially a “stop the world and restart” operation.

LBFT可以通过在se-quence中嵌入配置更改命令来重新配置自己。扩展配置更改命令的分支上的块被提出和表决只是为了进行重新配置，但它们不得包含正常的事务有效负载(事务也不会包含get执行)。这本质上是一个“停止世界和重启”操作。

A new configuration epoch may change every part of the algorithm, from the validator set to the protocol itself. A new epoch starts with an epoch-genesis block proposed and committed by the validators of the new epoch.
一个新的配置时代可能会改变算法的每个部分，从验证器集到协议本身。一个新的时代始于一个新的时代的验证器提出并承诺的一个时代创始块。

Proposer election strategies.
There are several ways to enhance the proposer election mechanism in order to avoid performance hick-ups when a bad leader is elected, or worse, when a succession of bad leaders is elected.
有几种方法可以加强提议人的选举机制，以避免当一个糟糕的领导人当选时，或者更糟糕的是，当一连串糟糕的领导人当选时，表现不佳。

An alternate leader strategy elects an ordered pair of leaders per round. The lower leader delays a certain duration in order to yield to the higher leader. This approach has the benefit of unlocking rounds with a crashed leader quickly, but the risk of creating contention among the leaders. The success of the approach largely hinges on the ability to stagger leaders effectively.
替代领导者策略每回合选择一对有序的领导者。较低的领导者延迟一定的时间，以便屈服于较高的领导者。这种方法的好处是快速解锁与崩溃的领导者的回合，但有可能在领导者之间引起争论。这种方法的成功在很大程度上取决于有效错开领导者的能力。

A different approach optimizes for a stable leader, and provides fairness and load balancing via rotating input generators. In each round, there are two distinct roles, a leader and an input generators. The leader is kept stable so long as progress and good performance are observed. The input generators are designated among the validators based on past performance and availability. In each round, the stable leader has the authority to promote or dismiss an input generator, but if it dismissed generators too frequently, the leader itself will get demoted eventually.
一种不同的方法对稳定的领导者进行优化，并通过旋转输入生成器提供公平和负载均衡。在每个回合中，有两个不同的角色，一个领导者和一个输入生成器。只要观察到进展和良好的性能，领导者就会保持稳定。输入生成器是根据过去的性能和可用性在验证者中指定的。在每个回合中，稳定的领导者有权提升或解雇输入生成器，但如果它解雇生成器过于频繁，领导者自己最终会被降级。

The last approach randomizes leaders using some source of randomness that cannot be predicted in advance.
最后一种方法使用一些无法提前预测的随机性来源对领导者进行随机化。

TCB.

As already mentioned in the Implementation section (Section 5), the Safety module of LBFT captures the rules that must be obeyed by a validator to guarantee agreement on committed proposals. First, the module succinct enough to allow formal verification of code implementations. Second, the interaction of the module with the environment is small: At bootstrap, the information that the validator persisted must be uploaded with freshness proof to the safety module. Thereafter, the module can process proposed blocks autonomously and without any interaction with the environment. This allows to run the entire module within a secure, trusted compute base (TCB), provided that it is backed by a small, freshness-preserving persistent store.
正如在实施部分（第5节）中已经提到的，LBFT的安全模块抓住了验证者必须遵守的规则，以保证对承诺的建议书达成一致。首先，该模块足够简洁，允许对代码实现进行正式验证。第二，模块与环境的交互很小：在引导时，验证器坚持的信息必须上传到安全模块。此后，该模块可以自主地处理提出的块，并且无需与环境进行任何交互。这允许在一个安全、可信的计算基础（TCB）中运行整个模块，前提是它有一个小型的、保持新鲜度的持久存储。

Flexible BFT enhancements.

The Flexible BFT [21] approach allows to support different commit as- surance levels within the same protocol. There are several ways in which this approach may be harnessed to enhance the functionality of LBFT in the future. First, it can gradually increase assurance guarantees for past transactions, thus longer since a transaction has settled, the safer it becomes. This notion is similar to the gradually increasing finality of transactions in Bitcoin and Ethereum. Second, it allows to consider certain blocks as more critical, e.g., periodic checkpoints, high-stake transactions, and reconfiguration com- mands. Third, it supports adapting the commit parameters over time as evidence of corrupt/honest behavior accrues, and likewise, as experience is gathered on network transmission delays.
灵活的BFT[21]方法允许在同一协议中支持不同的提交作为保证级别。有几种方法可以利用这种方法来增强未来LBFT的功能。首先，它可以逐步增加对过去交易的保证担保，因此，交易结算的时间越长，就越安全。这一概念类似于比特币和以太坊交易逐渐增加的终局性。其次，它允许将某些块视为更关键的块，例如定期检查点、高风险交易和重新配置com-mands。第三，它支持随着时间的推移调整提交参数，因为腐败/诚实行为的证据积累，同样，随着网络传输延迟的经验收集。