

区块链技术与应用

计算机科学与技术学院 李京

03章 区块链网络层

目录

• 3.1 P2P网络

• 3.2 比特币区块链网络

• 3.3 数据传播协议

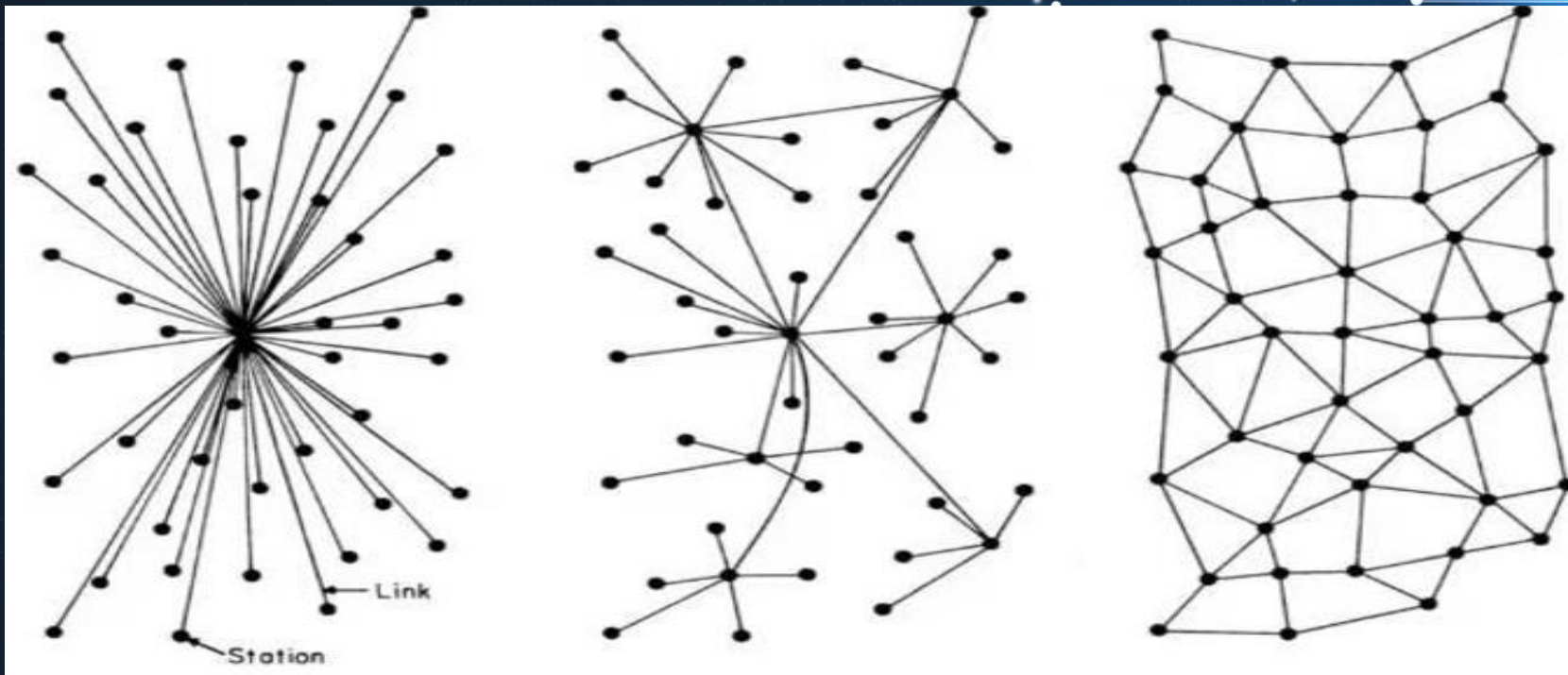
3.4 数据验证机制

3.5 矿池网络协议

3.6 区块链分叉

3.1 P2P网络

- 分布式系统的网络拓扑结构



(a) 中心化
私有链

(b) 多中心化
联盟链或私有链

(c) 去中心化
公有链

什么是P2P网络?

- P2P网络 (Peer-to-Peer Network) 即对等网络或对等计算机网络, 也称“点对点”或“端对端”网络, 是一种在对等网络实体 (Peer) 之间分配资源、任务和工作负载的分布式应用框架;
- 在P2P网络中, 对等节点拥有同等特权, 每个节点将其部分资源 (如计算、存储、网络带宽) 直接提供给其他节点使用, 而不需要中央服务器进行中间协调。每台计算机既是服务器, 又是客户端;
- 去中心化: 网络中无需中央管理节点的存在。

P2P网络的特点

- **可扩展性强**：节点可以自由加入和退出，P2P网络的自组织、自配置和自动负载均衡特性，破解了C/S模式下中心服务器的性能瓶颈问题。
- **健壮性好**：服务和资源分散于各个节点，不经由第三方，无明显弱节点。
- **高性价比**：有效利用分散于网络中大量节点上的空闲资源。
- **私密性**：信息传输无需通过集中节点，所有节点都具备中继转发能力，大幅提高了通信的匿名性，个人隐私得到保护。
- **均衡性**：资源和处理能力分布于多个节点，避免网络流量过于集中。

P2P网络的分类

混合式对等网络

- C/S和P2P两种模式的混合，反映了早期网络从C/S到P2P的过渡
- 混合式对等网络最具影响力的代表是Napster
- 早期的P2P网络不是完全去中心化的

无结构对等网络

- 网络的特点是无固定网络结构图，无中心节点
- 节点地址没有统一标准
- 典型的无结构P2P网络协议如Gnutella，它是纯粹意义上的P2P网络

结构化对等网络

- 一般采用哈希函数将节点地址规范为标准的标识
- 内容的存储位置与节点标识之间存在映射关系，可以实现有效的节点地址管理
- 最具代表性的经典模型和应用体系如Chord、Pastry等

P2P与覆盖网 (Overlay Network)

- P2P网络通常构建在更底层的物理网络之上，并为特定应用提供支持，是典型的覆盖网络 (OverLay Network)。
- 覆盖网络 (OverLay Network)：建立在另一个网络上、并为更高层应用提供支持的中间层网络。
- 覆盖网络的作用：使得上层应用无需过多考虑与网络有关的对等实体发现、直接通信、数据安全、资源定位、网络标识及其分配、节点加入与退出、负载均衡等问题，将精力集中在业务功能实现上。

P2P网络应用经典案例

Napster——P2P网络的先驱

- Napster是P2P技术在文件分享领域的最先尝试
- 创新点在于通过构建存储音乐文件索引与存放位置的信息的Napster服务器
- 用户需要某个音乐文件时，首先与Napster服务器相连，检索信息，根据返回的存放节点择优再进行下载
- 中心化终归是隐患，不可避免地带来了系统瓶颈、单点故障等问题

BitTorrent——Napster衍生强化版

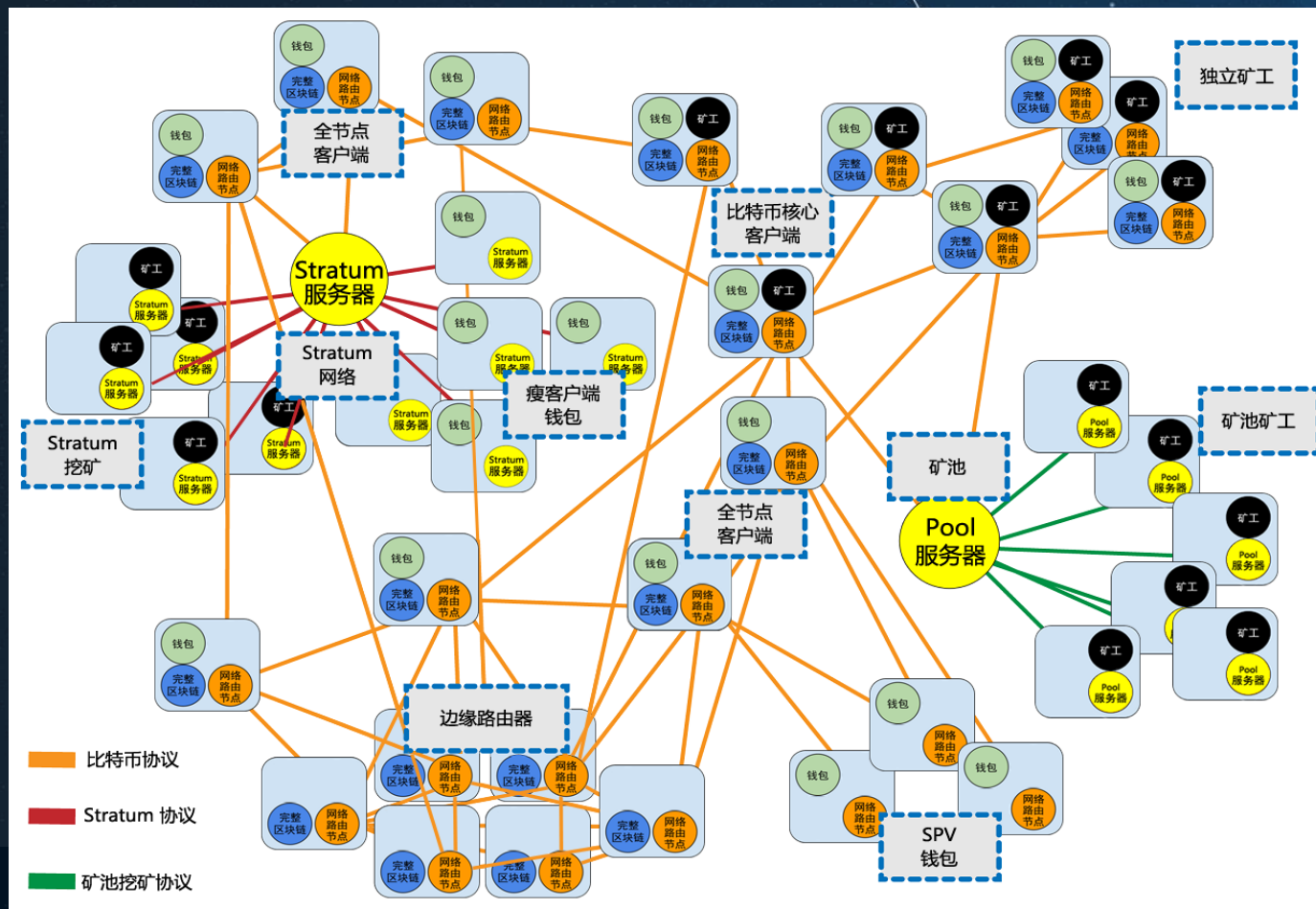
- BitTorrent网络中共享同一文件的用户形成一个独立的子网，从而将服务端分散化了，不会因为单点故障影响整体网络。
- 文件的持有者将文件发送给其中一名用户，再由这名用户转发给其它用户，用户之间相互转发自己所拥有的文件部分，直到每个用户的下载都全部完成。

区块链——去中心化和去中介化

- 区块链网络层大多是选择P2P模式作为其组网模型，其理念就是去中心化和去中介化
- P2P网络天然的全网均等的属性与区块链不谋而合
- 再加上P2P已经是发展成熟、经过考验的技术，二者的结合几乎是必然的

3.2 比特币区块链网络

- 比特币网络是一个典型的P2P网络，包括系统的组网方式、消息传播机制和验证机制。



比特币网络节点

- 比特币网络由多种类型的节点组成，其功能集合一般包括网络路由(Network Route, 简称为N)、完整区块链(Full Blockchain, 简称为B)、矿工(Miner, 简称为M)、钱包(Wallet, 简称为W)。每个区块链节点都参与全网路由，同时也可能包含其他功能。



全节点和轻节点

- 全节点：拥有完整的、最新区块链数据的节点称为“全节点”，这样的节点能够独立自主地校验所有交易；
- SPV节点/轻量级节点：只保留区块头数据，通过“简易支付验证”方式完成交易验证的节点称为“SPV节点”或“轻量级节点”，它们没有区块链的完整拷贝。

比特币网络组网方式

新区块链节点加入区块链网络通过以下五种方式：

地址数据库

- 网络节点的地址信息会由地址管理器 (Address manager) 存储在地址数据库 peers.dat 中。
- 节点启动时，由 address manager 载入。
- 节点第一次启动时，无法使用这种方式。

通过命令行指定

- 用户可以通过命令行方式将指定节点的地址传递给新节点。
- 命令行传递参数格式形如：
-addnode <ip>
或者 -connect <ip>。

DNS种子

- seed.bitcoin.sipa.be
- dnsseed.bluema.tt.me
- dnsseed.bitcoin.dashjr.org
- seed.bitcoinstats.com
- seed.bitcoin.jonasschnelli.ch

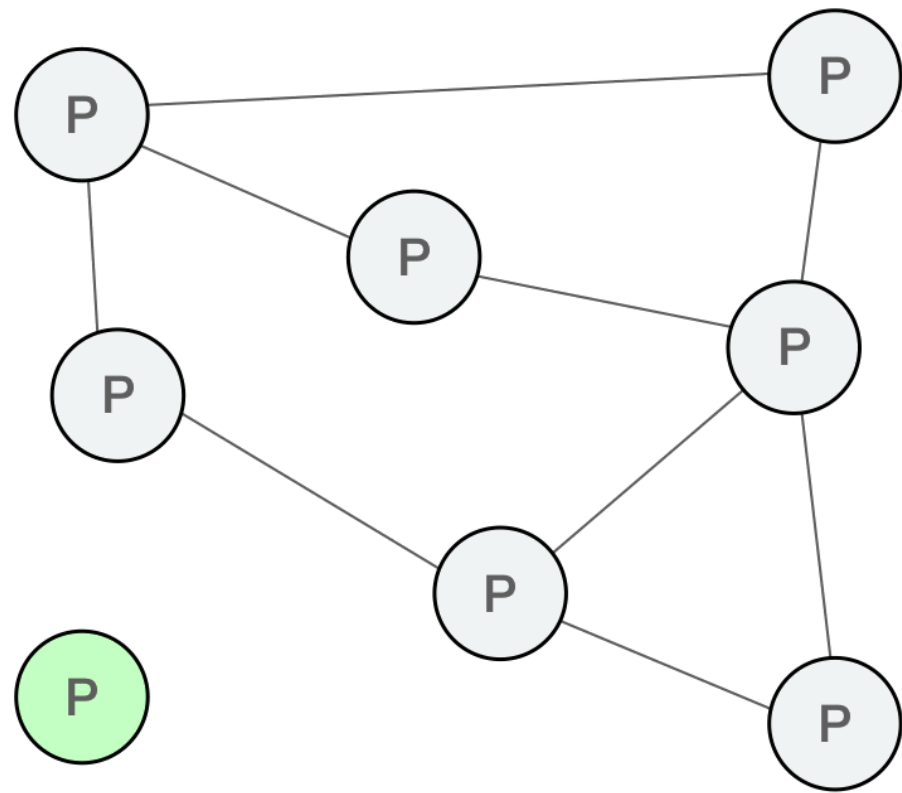
硬编码地址

- 如果 DNS 种子方式失败，还有最后的手段，即硬编码地址。
- 需要注意的是，需要避免 DNS 种子和硬编码种子节点的过载。
- 使用后应该断开与这些种子节点的连接

通过其他节点获得

- 节点间通过 getaddr 消息和 addr 消息交换 IP 地址信息

新节点加入网络



节点发现过程

1

- 用户比特币程序启动时，并不知道任何活跃的全功能节点的IP地址;

2

- 为了接入网络 and 发现这些地址，程序会向DNS地址(种子)发出查询请求;

3

- DNS服务器返回的响应包（DNS的A记录）中会包含一个或多个全功能节点的IP地址

DNS种子服务器例子

```
// From: https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp
```

```
vSeeds.emplace_back("seed.bitcoin.sipa.be"); // Pieter Wuille, only supports x1, x5  
vSeeds.emplace_back("dnsseed.bluematt.me"); // Matt Corallo, only supports x9  
vSeeds.emplace_back("dnsseed.bitcoin.dashjr.org"); // Luke Dashjr  
vSeeds.emplace_back("seed.bitcoinstats.com"); // Christian Decker, supports x1 - xf  
vSeeds.emplace_back("seed.bitcoin.jonasschnelli.ch"); // Jonas Schnelli, only supports x1, x5  
vSeeds.emplace_back("seed.btc.petertodd.org"); // Peter Todd, only supports x1, x5,  
vSeeds.emplace_back("seed.bitcoin.sprovoost.nl"); // Sjors Provoost  
vSeeds.emplace_back("dnsseed.emzy.de"); // Stephan Oeste
```


DNS种子服务器返回的信息

- 用Linux的dig命令查看某个DNS种子服务器返回的信息

```
$ dig seed.bitcoin.sipa.be

; <<>> DiG 9.10.6 <<>> seed.bitcoin.sipa.be
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64217
;; flags: qr rd ra; QUERY: 1, ANSWER: 25, AUTHORITY: 0, ADDITIONAL: 1

;; ANSWER SECTION:
seed.bitcoin.sipa.be.      3337    IN      A       214.99.158.195
seed.bitcoin.sipa.be.      3337    IN      A       202.230.171.5
seed.bitcoin.sipa.be.      3337    IN      A       20.79.84.168
seed.bitcoin.sipa.be.      3337    IN      A       91.173.63.7
seed.bitcoin.sipa.be.      3337    IN      A       45.74.113.110
seed.bitcoin.sipa.be.      3337    IN      A       168.160.113.156
seed.bitcoin.sipa.be.      3337    IN      A       29.239.106.205
seed.bitcoin.sipa.be.      3337    IN      A       25.142.230.178
seed.bitcoin.sipa.be.      3337    IN      A       146.73.72.54
# ... and so on.
```

3.3 数据传播协议

- 节点间采用TCP协议通信，端口是8333。一个通用的区块链网络一般包括如下核心场景：
 - ① 节点入网建立初始连接
 - ② 节点地址传播发现
 - ③ 矿工、全节点同步区块数据
 - ④ 客户端创建一笔交易
 - ⑤ 矿工、全节点接受交易
 - ⑥ 矿工、全节点挖出新区块，并广播到网络中
 - ⑦ 矿工、全节点接收广播的区块

参考：“知乎：比特币源码解读 - 消息处理”

<https://zhuanlan.zhihu.com/p/121791191>

消息(Message)

- 消息是不同节点间信息传输的基本单位，协议体现为消息格式的约定和时序。
- 比特币消息基本格式：消息头+消息体



- 起始字符串：奇异数 (Magic Number: 0xf9beb4d9) ，用于标识下一个消息的开始
- 命令名：ASCII形式的命令码，命令内容在消息体中
- 消息体大小：以字节为单位，消息体的长度
- 校验：校验值

消息示例

f9beb4d976657273696f6e00000000650005f1a69d2721101000100000066000
bc8f5e5400000010000000000000c61b640928d01000000000000000006ffcb0071
c0208d128035cbc97953f80f2f5361746f7368693302e392e332fcf5650001

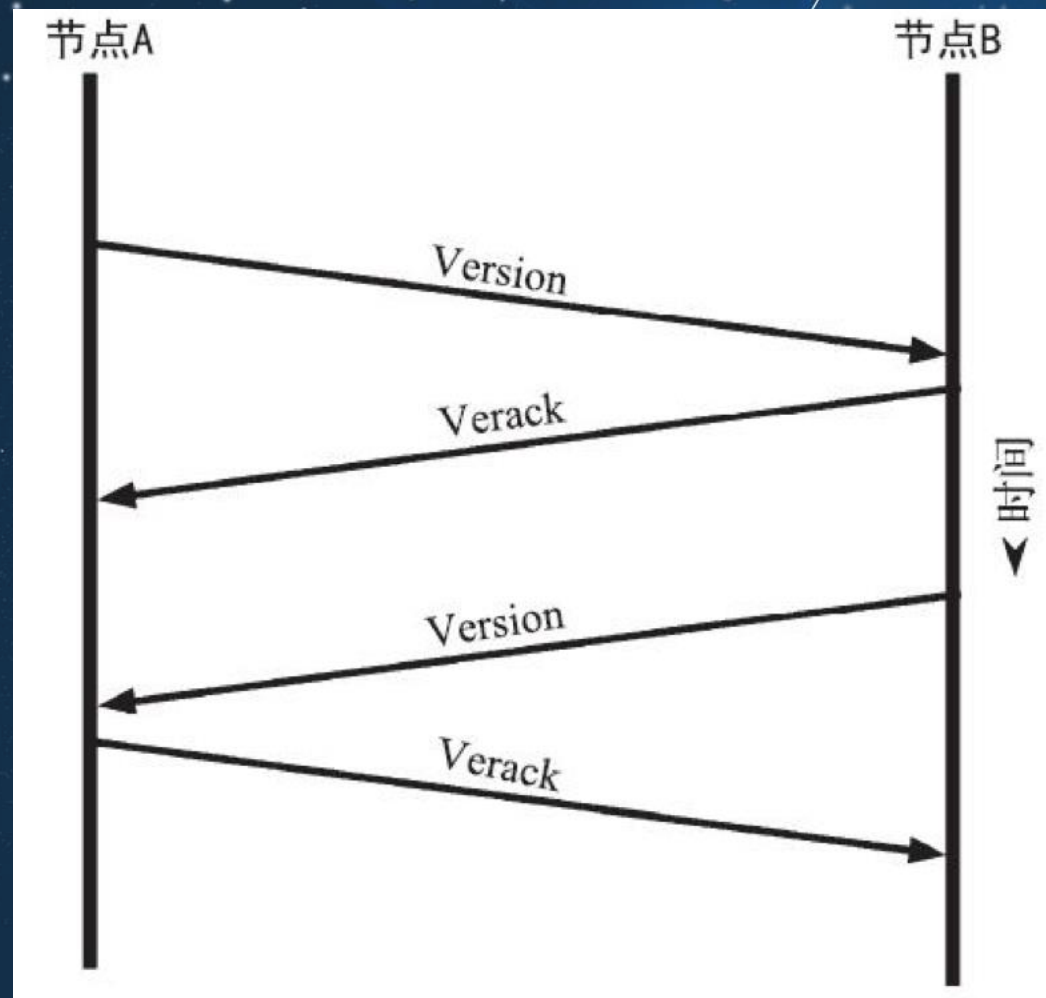
- f9beb4d9 -network magic (always 0xf9beb4d9 for mainnet)
- 76657273696f6e000000 -command,12 bytes human-readable
- 65000000 -pay Load Length, 4 bytes, Little-endian
- 5f1a69d2 -payload checksum, first 4 bytes of hash256 of the pay Load
- 7211.. -pay Load

常用消息的类型

消息	作用
version	宣告自己的版本
addr	地址消息，发送地址信息
inv	库存宣告消息，宣告自己的各种库存
getdata	请求块，交易等各种数据
getblocks	用于比较谁比较长的消息
tx	交易传播消息
review	用于检查公钥能否解锁签名
block	块传播消息
getaddr	地址请求消息
checkorder	获得一个公钥脚本的
submitorder	提交交易到钱包
reply	用于对请求或命令进行回应

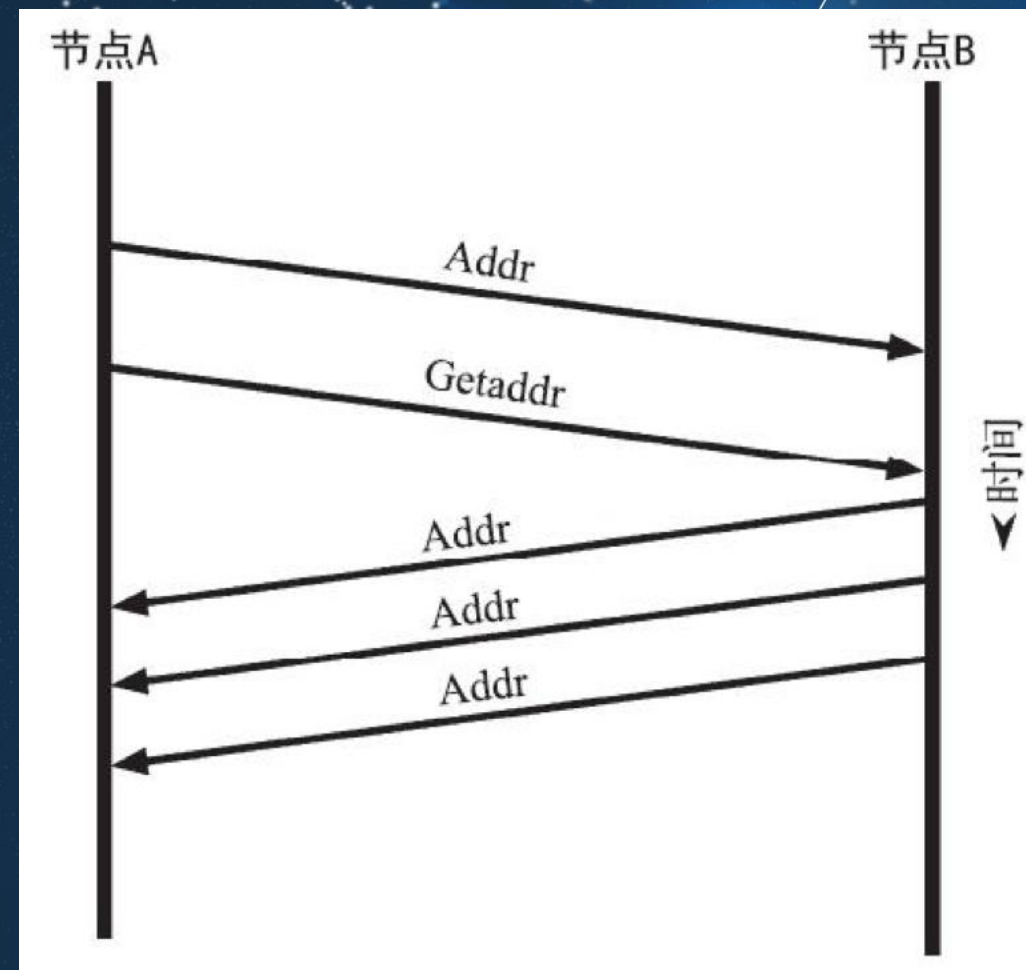
数据传播协议-建立初始连接

- 节点A通过发送Version消息到远端对等节点节点B表示连接成功，该消息包括当前节点的版本消息、区块和当前的时间；
- 节点B收到后检查兼容性，兼容则确定连接，返回Verack消息，也会向节点A发送它的Version消息；
- 节点A收到后检查兼容性，兼容则返回Verack消息，连接成功建立。



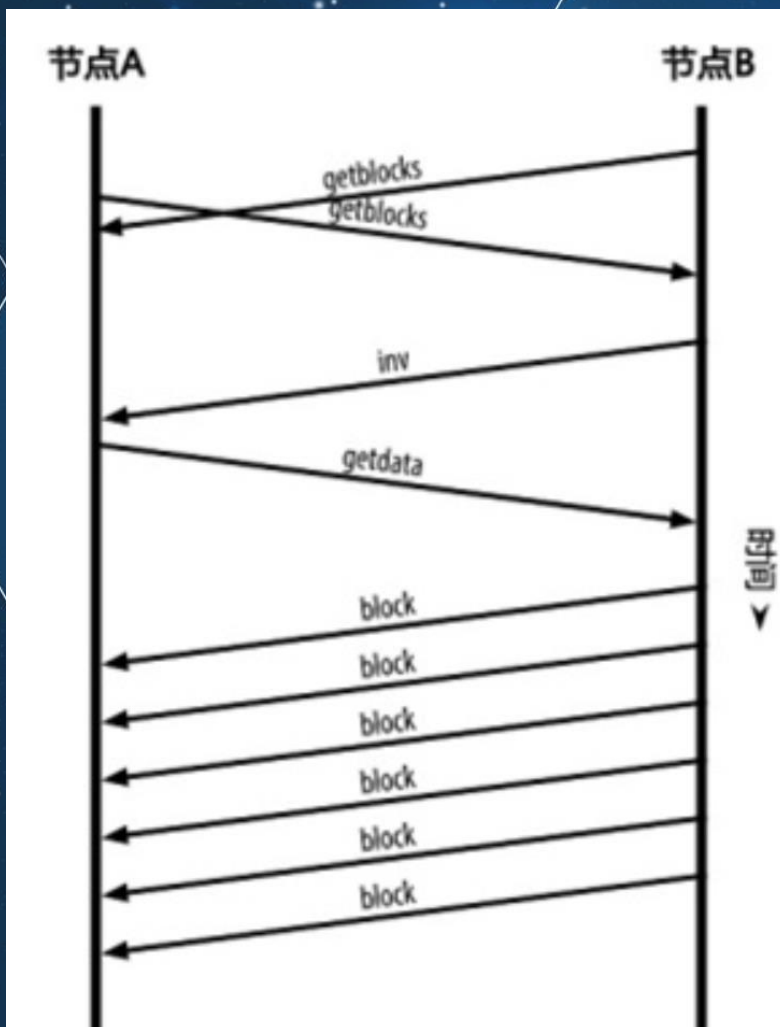
数据传播协议-地址广播及发现

- 成功连接后，新节点 A 向相邻节点 B 发送包含自身 IP 地址的 Addr 消息。相邻节点会将此 Addr 消息再度转发给各自相邻节点，保证新节点 A 可被更多节点获知。
- 节点 A 可以向其相邻节点 B 发送 Getaddr 消息，节点 B 会回送若干地址信息，如其它节点 (Peer) 的 IP 地址、端口等数据。



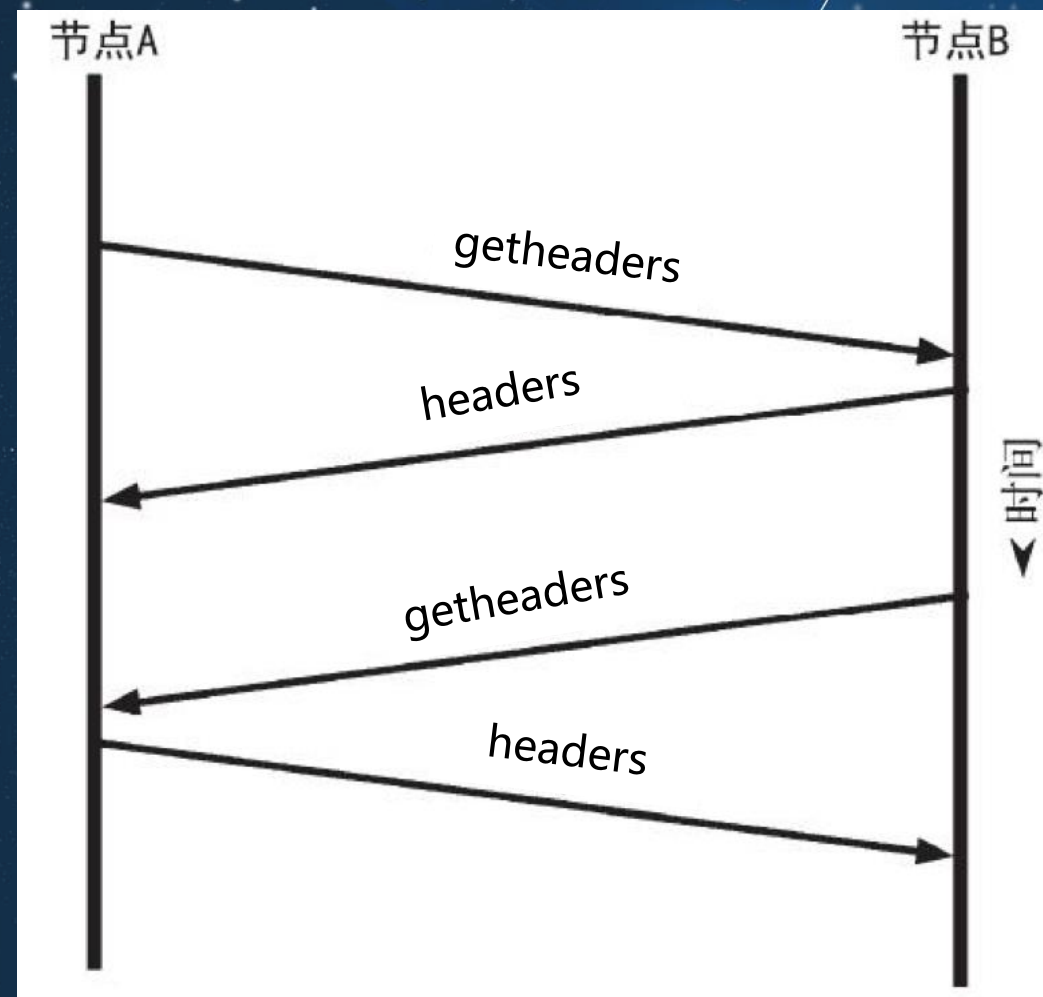
数据传播协议-同步区块数据 (全节点)

- 一个全节点连接到对等节点之后，首先需要构建完整的区块链。该节点向相邻节点发送version消息，该消息中包含BestHeight字段标示了自己的区块高度。通过互相发送version消息，对等节点就可得知双方的区块数量。
- 对等节点还会发送getblocks消息，该消息中包含了本节点保存的区块链顶端的区块hash值，如果一个节点收到的hash在自己的区块链中不属于顶部，那么就代表自己的链比较长。
- 拥有更长区块链的节点会识别出其他节点需要补充的块，从而发送库存inv(inventory)消息，这个消息会包含块的hash值，从而告知其他节点这些块的存在。收到库存消息的节点发现自己缺少块，就会向周围节点(未必是发送库存节点)发送getdata消息，来请求具体某些块的数据，从而补全自己。收到getdata消息的节点再把节点请求的块数据发送出去。



数据传播协议-同步区块数据 (SPV节点)

- SPV节点同步的不是区块数据，而是区块头。
- 使用getheaders和headers消息。



初始化区块下载IBD

- 初始区块下载IBD (Initial Block Download)，通过下载区块、区块头、以及交易，比特币的所有的区块交易在本地就有了副本，成为一个完整的比特币网络节点，完全跟比特币网络同步。
- 通常要求下载并验证此时网络中最长、最正确的区块序列（区块链），该链从编号为1的区块开始；
- 一般新加入的节点或者节点离线24小时以上，均需执行这个操作，方可接入比特币网络，才能验证未确认的交易或新近挖出的区块。

初识区块下载方法

- 比特币的区块链数据下载有两种方式：
 - 块优先Blocks-First
 - 头优先Headers-First



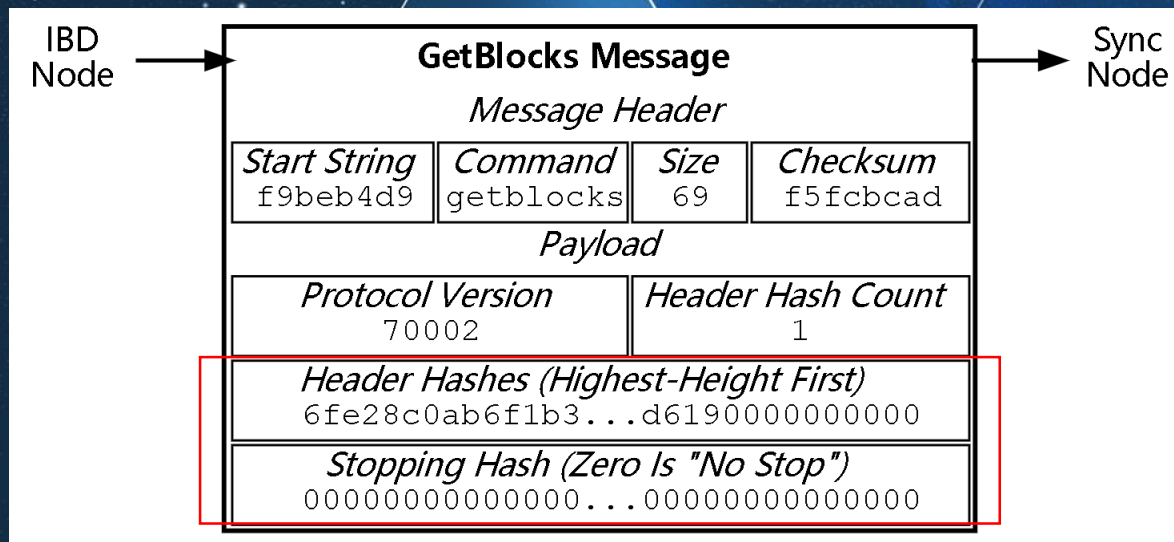
初始化区块下载-块优先

假设有两个节点A和B，其中B是已经完成区块同步的对外提供服务的正常节点，A是一个刚启动的节点，A上除了创世区块以外没有任何其他区块数据。那么A怎么用Blocks-First的模式从B上同步到自己想要的区块数据呢？

① A向B发送一个“Getblocks”消息

GetBlocks消息结构：

```
type MsgGetBlocks struct {  
    ProtocolVersion uint32  
    BlockLocatorHashes []*chainhash.Hash  
    HashStop chainhash.Hash  
}
```

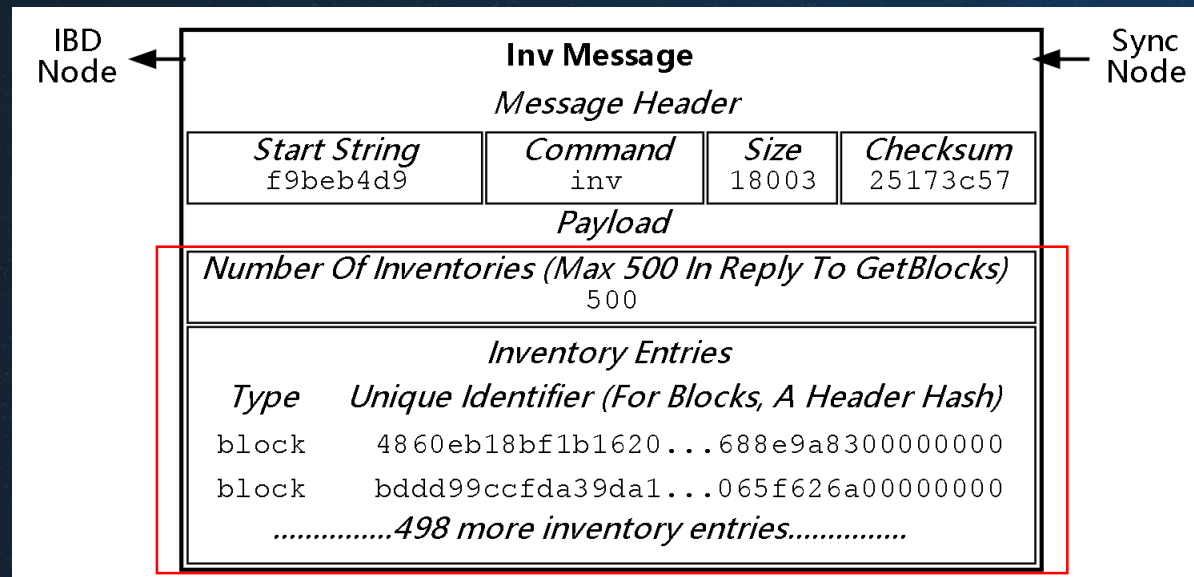


Header Hashes: 区块0的哈希值（大端模式），表示自己最新的区块是创世区块

Stopping Hash: 填写全0，表示想要同步到最新区块。

初始化区块下载-块优先

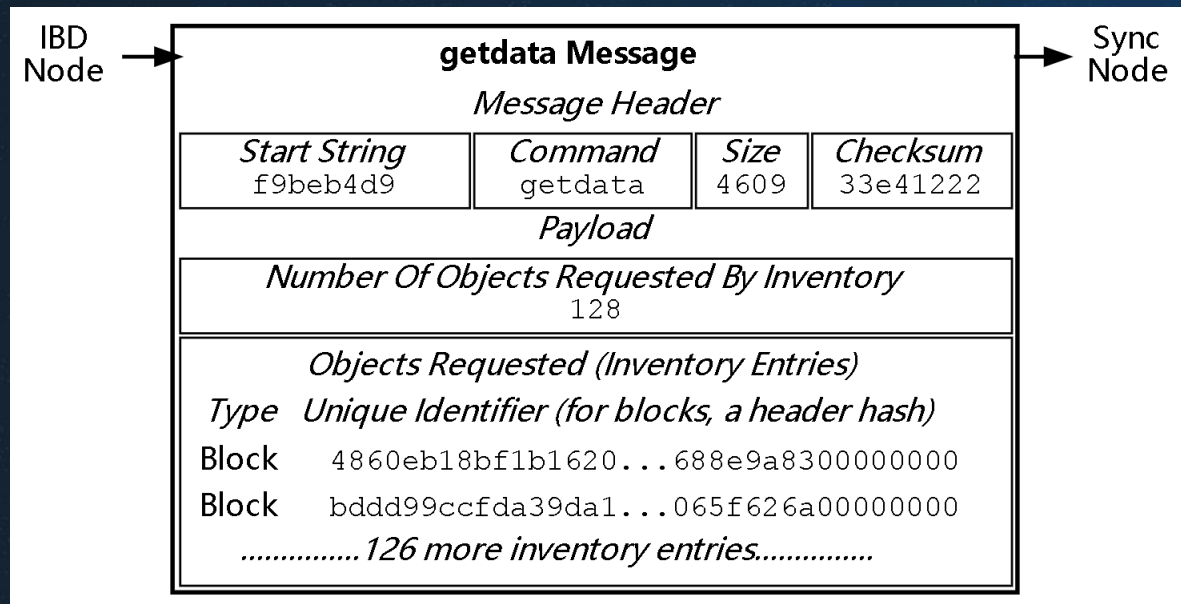
② B收到消息后，根据要求返回“inv”消息给A节点。



节点B返回有500（上限）条内容的inventory清单
条目：Type字段是block，表示这是区块，Unique Identifier字段是区块哈希值。这些区块哈希的顺序很重要，它代表的是区块的顺序。

初始化区块下载-块优先

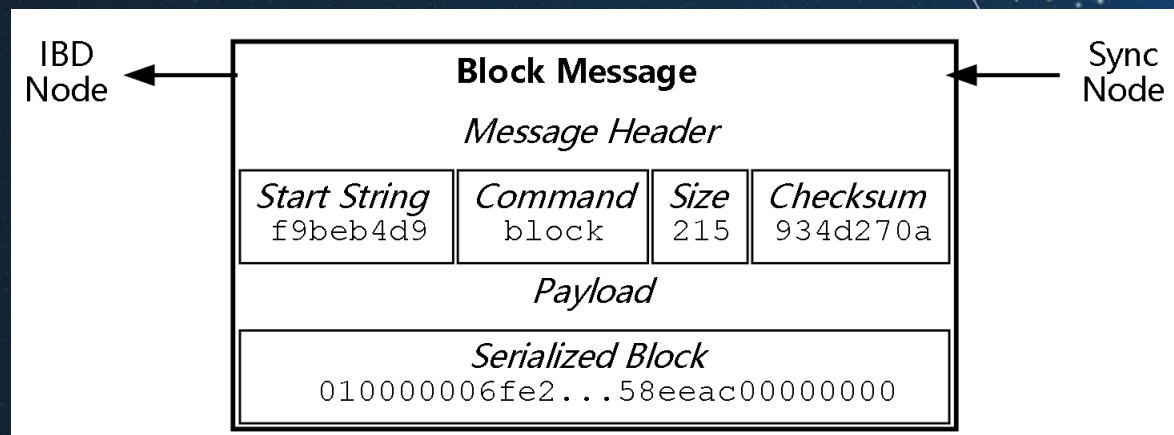
③ IBD节点A使用接收到的清单发送getdata消息向同步节点B请求128个块（上限）。



对于块优先节点，按顺序请求很重要，因为每个块头均引用前一个块的头哈希。

初始化区块下载-块优先

- ④ 收到getdata消息后，同步节点B将回复所请求的每个块。每个块都以序列化的块格式放入并以单独的block消息发送。block发送的第一条消息（针对块1）如下所示。



- ⑤ 节点A对接收到的每个块，对其进行验证，并保持最多128个块的下载队列。

初始化区块下载-块优先

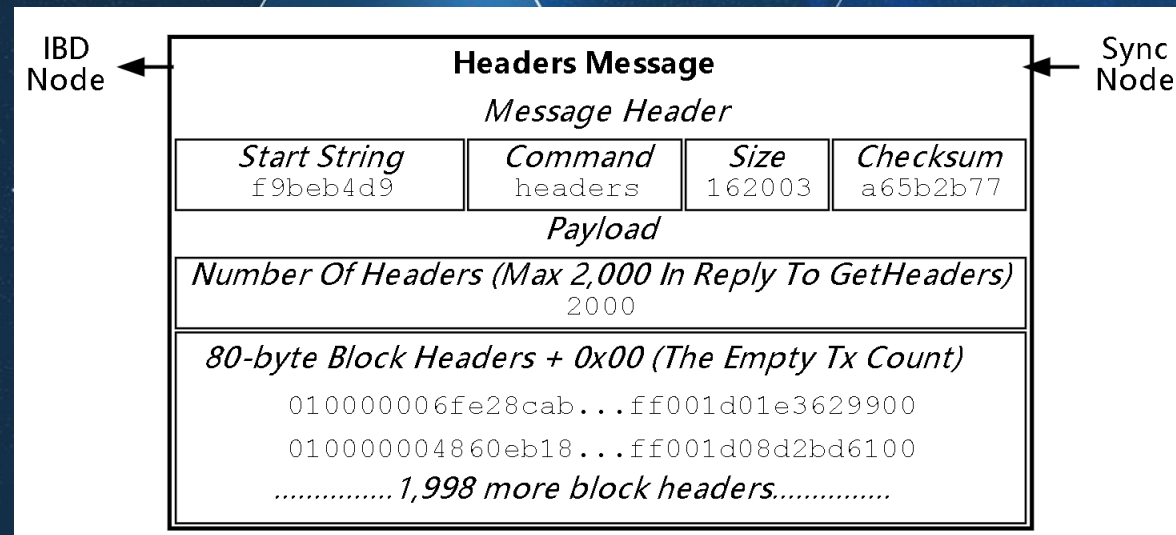
- A收到这个区块数据，对这个区块的合法性进行验证，同时等待B发送下一个区块数据。A接收到128个区块数据后，A根据之前接收到的B的"inv"消息里的清单还剩下 $500-128=372$ 个条目，于是继续发送下一个"getdata"消息（内含128个区块哈希）给B，如此循环一直到500个区块数据都从B发送给了A。这样就完成了一轮区块发送任务。
- 比特币的区块高度已经达到了67万多，这个过程要经过1300多轮才完成同步。为了避免出错，比特币加入了Checkpoint功能，Checkpoint就是指定一个区块高度的区块哈希必须等于某个哈希值。

初始化区块下载-块优先

- 块优先的优点是简单
- 主要缺点是，IBD节点的所有下载都依赖于单个同步节点和块发送的顺序性。这导致：
 - 速度限制：所有请求均发送到同步节点，因此，如果同步节点的上传带宽有限，则IBD节点的下载速度会很慢。
 - 重复下载：同步节点可以向IBD节点发送非最佳（但有效）的区块链，从而迫使IBD节点再次从其他节点重新开始其区块链下载。
 - 磁盘空间浪费：与重复下载密切相关，如果同步节点发送了一个非最佳（但有效）的块链，该链将存储在磁盘上，浪费空间，并可能在磁盘驱动器中填充无用的数据。
 - 高内存使用：无论是恶意还是偶然，同步节点都可以无序发送块，创建孤立块，直到接收并验证其父节点后才能对其进行验证。孤立块在等待验证时会存储在内存中，这可能会导致大量内存使用。

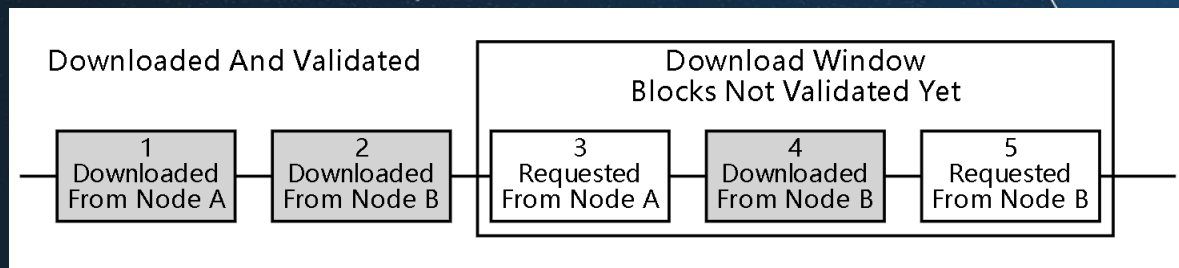
初始化区块下载-头优先

- 先尝试下载链中区块的描述头结构，然后以并行方式连接多个网络节点下载区块。
- 头优先是A使用GetHeaders消息发起，B返回Headers消息（最多包含2000条清单）



初始化区块下载-头优先

- A在收到B的“headers”消息后，验证这些区块哈希合法性，同时：
 - 继续同步剩下的block headers。（目前比特币区块头大致50M左右）
 - 开始同步区块数据。（使用“getdata”和“data”消息传输区块数据）
- 比特币默认最大连接数是8个，可以同时从8个节点并行下载区块数据。并行下载区块的时候，为了在多个节点之间分摊负载，每次从一个节点最多只请求16个区块，这就意味着每次最多下载 $8 \times 16 = 128$ 个区块。比特币还使用最大1024个区块的下载移动窗口，这样来最大化下载速度。下载移动窗口之前的区块都是被验证过的合法的区块，而窗口里的区块，只要下载了就马上进行验证，等全部下载完了，基本上也就验证完了。



示意图（真实窗口是1024）

数据传播协议-交易广播

• 比特币系统的交易数据传播协议核心步骤:

- ① 比特币交易节点将新生成的交易数据使用inv消息向全网所有节点进行广播;
- ② 每个节点发送getdata消息请求inv消息中所有交易的完整信息, 收到的节点使用tx发送交易。接收节点将收集到的交易数据存储到一个区块中;
- ③ 每个节点基于自身算力在区块中找到一个具有足够难度的工作量证明;
- ④ 当节点找到区块的工作量证明后, 就向全网所有节点广播此区块 (block消息);
- ⑤ 仅当包含在区块中的所有交易都是有效的且之前未存在过的, 其他节点才认同该区块的有效性;
- ⑥ 其他节点接受该数据区块, 并在该区块的末尾制造新的区块以延长该链条。

节点A

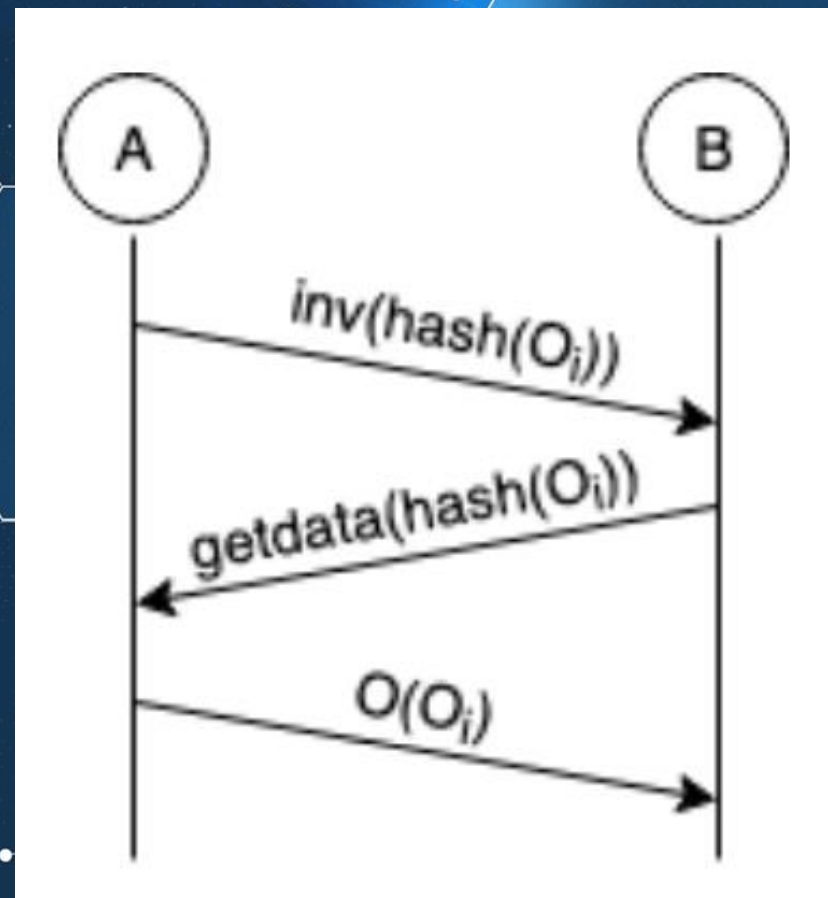


节点B



比特币的新块广播

- 当一个矿工节点发现新的区块后，它需要将此区块在全网尽可能大的范围内广播。有两种方式：
 - **1. 主动推送：**
 - 向每个完整节点对等发送一条带有新区块的 “block” 消息；
 - 矿工使用这种方式合理地绕过标准中继方法；
 - 它不能判断其对等方已经拥有刚发现的块。
 - **2. 区块中继**
 - 不主动推送；
 - 发出Inv消息，在Inv中携带新区块消息；
 - 由收到Inv消息的节点把区块取走。



数据传播协议-检测节点存活

- ping消息用于确认接收方是否仍处于连接状态。通过发送ping消息，可以检测节点是否存活。
- 接收方通过回复pong消息，告诉发送节点自己仍然存在。
- 默认情况，任何超过20分钟未响应ping消息的节点会被认为该节点已经从网络中断开。

3.4 数据验证机制

- 当新区块在区块链网络传播时，每个接收到区块的节点都将对区块进行独立验证，验证通过的区块才会进行转发，从而尽早杜绝无效或者恶意数据在网间传播，预防小部分节点串通作恶导致无效区块被网络接受，尽最大可能保证网络中传播区块的正确性。
- 以比特币网络为例，节点接收到邻近节点发来的数据后，其首要工作就是验证该数据的有效性。矿工节点会收集和验证P2P网络中广播的尚未确认的交易数据，并对照预定义的标准清单，从数据结构、语法规范性、输入输出和数字签名等各方面校验交易数据的有效性，并将有效交易打包到当前区块中。

数据验证清单

- ① 验证区块大小在有效范畴;
- ② 确认区块数据结构（语法）的有效性;
- ③ 验证区块至少含有一条交易;
- ④ 验证第一个交易是coinbase交易（Previous Transaction hash为0且Previous Txout-index为-1），有且仅有一个;
- ⑤ 验证区块头部有效性
 - 确认区块版本号是本节点可兼容的
 - 区块引用的前一区块是有效的
 - 区块包含的所有交易构建的默克尔树是正确的
 - 时间戳合理
 - 区块难度与本节点计算的相符
 - 区块哈希值满足难度要求

数据验证清单

⑥ 验证区块内的交易有效性，具体检查列表如下：

- 检查交易语法正确性
- 确保输入与输出列表都不能为空
- lock time小于或等于INT MAX，或者nLockTime和nSequence的值满足MedianTimePast（当前区块之前的11个区块时间的中位数）
- 交易的字节大小大于等于100
- 交易中签名数量小于签名操作数量上限（MAX_BLOCK_SIGOPS）
- 解锁脚本（scriptSig）只能够将数字压入栈中，并且锁定脚本（scriptPubkey）必须要符合isStandard的格式（拒绝非标准交易）
- 对于coinbase交易，验证签名长度2至100字节
- 每一个输出值，以及总量，必须在规定值的范围内（不超过全网总币量，大于0）
- 对于每一个输入，如果引用的输出存在于内存池中任何的交易，该交易将被拒绝
- 验证孤立交易：对于每一个输入，在主分支和内存池中寻找引用的输出交易，如果输出交易缺少任何一个输入，该交易将被认为是孤立交易。如果与其匹配的交易还没有出现在内存池中，那么将被加入到孤立交易池中
- 如果交易费用太低（低于minRelayTxFee设定值）以至于无法进入一个空的区块，则交易将被拒绝
- 每一个输入的解锁脚本必须依据相应输出的锁定脚本来验证

数据验证清单

⑥ 验证区块内的交易有效性，具体检查列表如下：（继续）

- 如果不是coinbase交易，确认交易输入有效，对于每一个输入：
 - 验证引用的交易存于主链
 - 验证引用的输出存于交易
 - 如果引用的是coinbase交易，确认至少获得COINBASE_MATURITY(100)个确认
 - 确认引用的输出没有被花费
 - 验证交易签名有效
 - 验证引用的输出金额有效
 - 确认输出金额小于等于输入金额（差额即为手续费）
- 如果是coinbase交易，确认金额小于等于交易手续费与新区块奖励之和

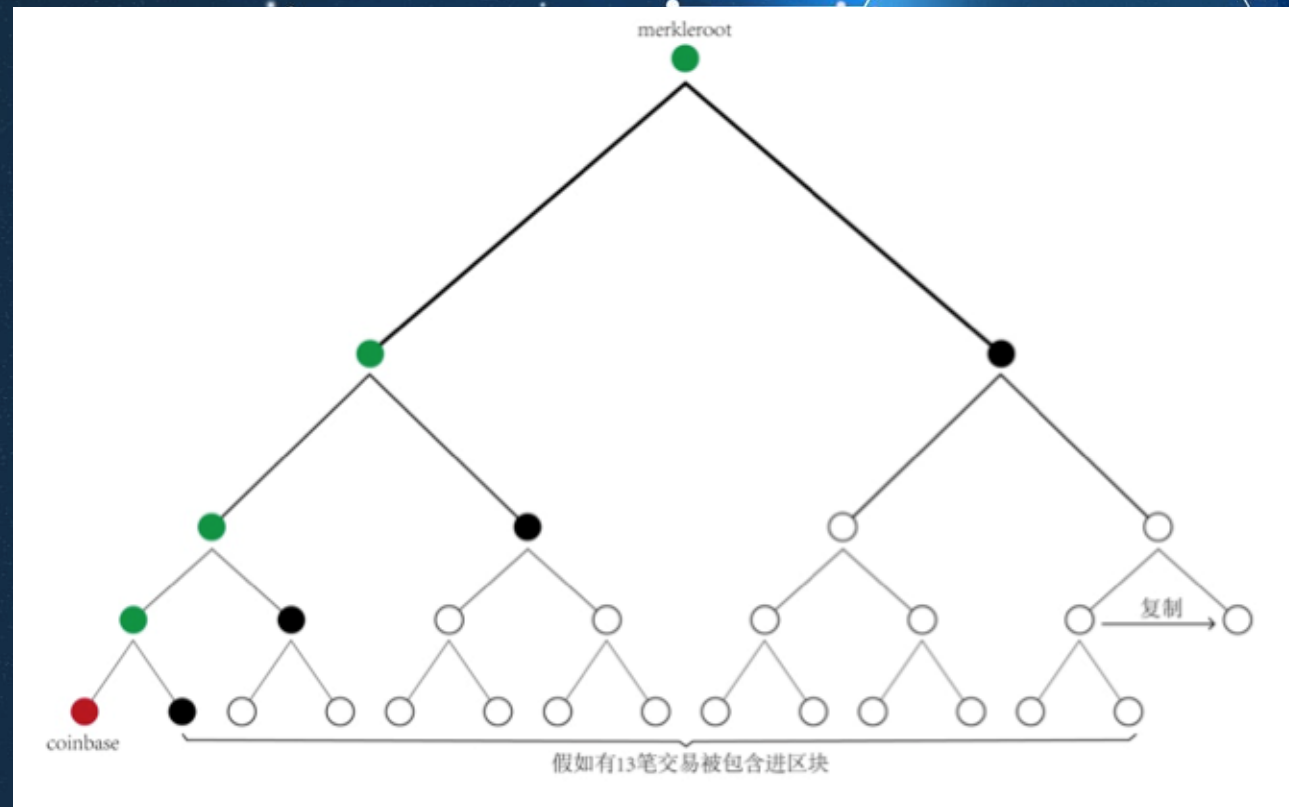
3.5 矿池网络协议



- Getwork协议可以认为是最早的挖矿协议
- 拥有完整数据的节点构造区块头, 即提供 Version, Prev-block, Bits和Merkle-root 这4个字段
- 挖矿程序主要是递增遍历Nonce
- 对于Getwork而言, 矿工对区块一无所知, 只知道修改Nonce这4个字节, 共计 2^{32} 大小的搜索空间
- 如今比特币和莱特币节点都已经禁用 Getwork 协议, 转向更高效的 Getblocktemplate协议

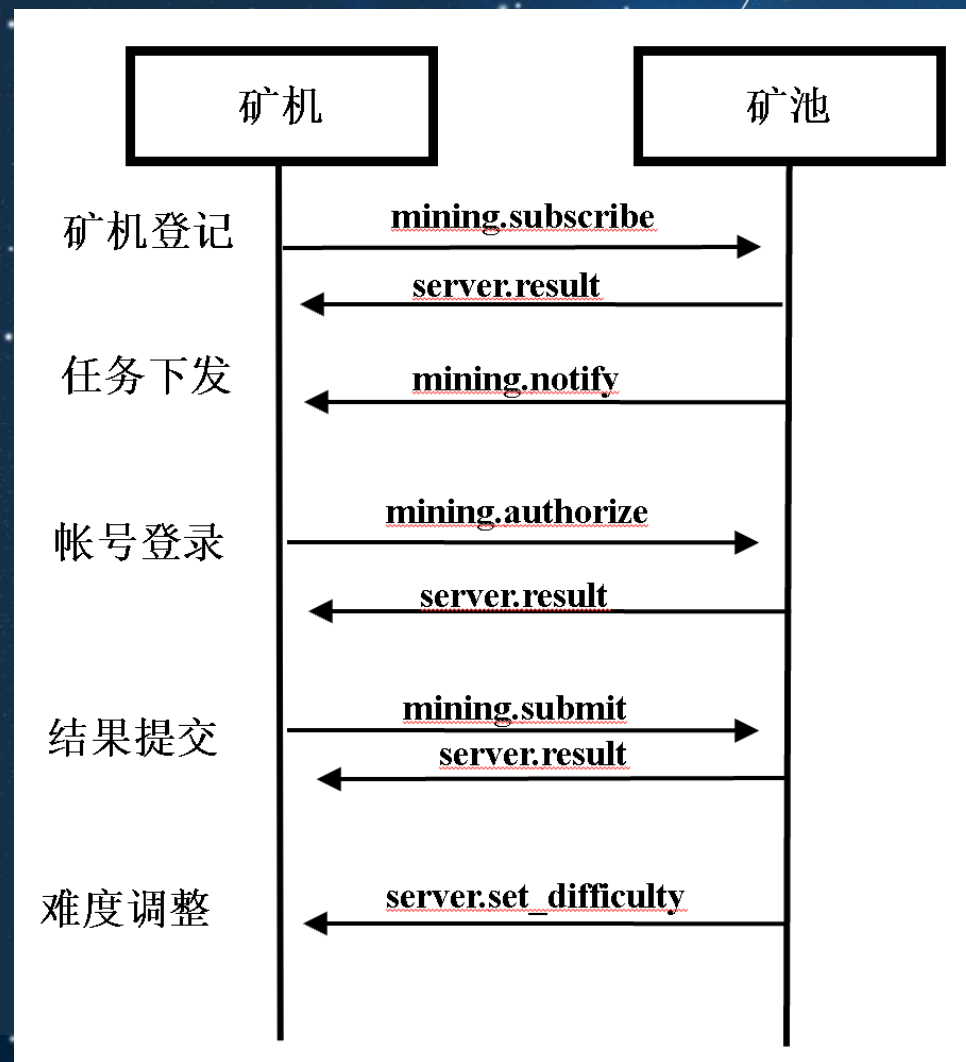
矿池网络协议-Getblocktemplate

- Getblocktemplate协议，2012年诞生
- 其最大的不同点是：Getblocktemplate协议让矿工自行构造区块
- 因为由矿工构建coinbase交易，这种方式所带来的搜索空间巨大
- Getblocktemplate协议虽然扩大了搜索空间，但数据负载过大，每次调用约返回1.5M左右数据。



矿池网络协议-Stratum

- Stratum协议是为了扩展支持矿池挖矿而编写的挖矿协议，2012年底出现，是目前最常用的矿机和矿池之间的TCP通信协议之一，
- 数据采用JSON格式封装，矿机与矿池通信过程如右图所示。Stratum协议利用Merkle树结构特性，从coinbase构造hashmerkleroot，无须全部交易，只要把与coinbase涉及的默克尔路径上的hash值返回即可。假如区块包含N笔交易，这种方式数据规模将压缩至 $\log_2(N)$ ，大大降低了矿池和矿工交互的数据量。



3.6 区块链分叉

自然分叉，机器共识过程产生的临时分叉

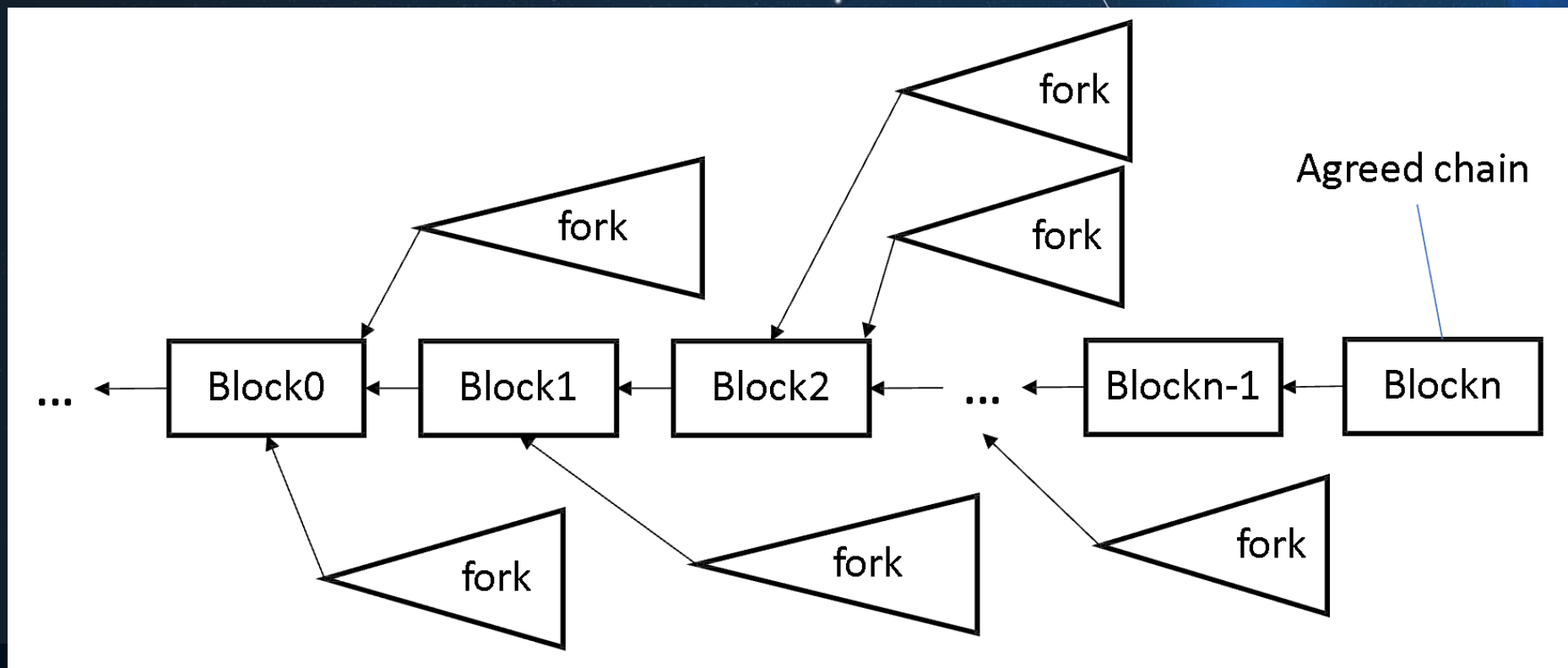
人为分叉，人的共识失败产生的分叉

自然分叉

- 自然分叉，机器共识过程产生的临时分叉。
 - ▣ 比特币网络是一个去中心化的P2P网络。
 - ▣ 固有的节点地域分布、网络传输延迟，造成节点接收新区块存在一定的时间差异。
 - ▣ 当两个不同节点近乎同时发掘出新区块A、B并进行广播的时候，就会造成后继区块链分叉的发生。

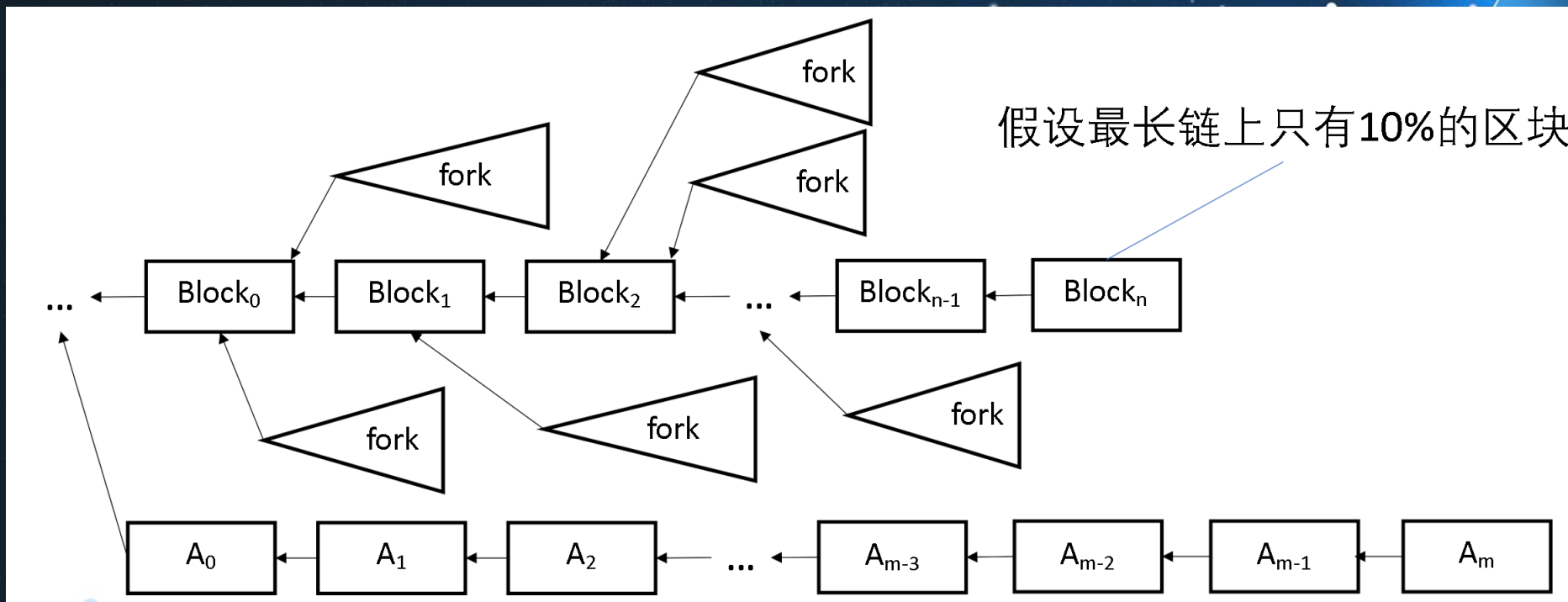
补充

- 区块是并发产生的，并且具有广播延迟，高出块率会导致更多的分叉
- 分叉浪费网络和处理资源，分叉降低安全性



补充

- 最长链规则不再安全



攻击者只需要超过 10% 的算力就可以改变最长链!

同步账本的时间应该远小于出块时间, 否则就会增加分叉的概率。

人为分叉

- 人为分叉，人的共识失败产生的分叉（BIP）

- “软分叉”是向前兼容的分叉。新规则下产生的区块可被未升级的旧节点所接受，旧节点只是无法识别、解析新规则。新、旧版本互相兼容，软分叉对整个系统的影响较小。
- “硬分叉”是不向前兼容的，旧版本节点不会接受新版本节点创建的合法区块，于是新旧版本节点开始在不同的区块链上运行，由于新旧节点可能长期并存，不像软分叉是临时的，硬分叉是有可能长期存在的，分叉链的存活在于其算力的大小。

区块链分叉-软分叉示例

- 基于coinbase字段的随机数扩展，coinbase data大小在2-100字节，原本可任意定制，但BIP-34要求开始必须在开头包含块高度且更新块版本信息，保证了交易和区块的唯一性，来帮助区块验证。其升级过程如下：

初始矿工将块版本号设置为“2”，表示其准备好升级，但此刻并不要求coinbase data包含块高度；

当最近1000个区块中超过75%的版本号是“2”时，整个系统开始强制要求版本号设置为“2”，且要求coinbase data包含块高度，但此时版本号为“1”的区块仍被接受；

当最近1000个区块中超过95%的版本号是“2”时，版本号为“1”的区块将不被接受，迫使最后一小部分节点进行升级。

硬分叉

- 如果原区块链称为A版本，硬分叉产生的同源分叉链称为B版本，则具体可以分为如下几种情况：

A版本仍然被广泛支持，B版本算力不足消亡，即还是保留原链。

B版本获得广泛支持，A版本算力不足消亡，即保留新链。

A、B版本都有相当的支持，同时并存，这种情况是最为符合严格意义上的硬分叉，例如ETH与ETC，两者都有其代币，这种分叉存在一定的门槛。

A版本仍然被广泛支持，B版本通过代码调整难度，小部分节点也能够让它存活。与3) 的区别在于这种分叉币几乎没有门槛，人人可以分叉。

B版本获得支持，A版本调整代码，小算力也可存活。

硬分叉

• 硬分叉的过程一般经历如下几个阶段：

软件分叉

- 新的客户端发布，新版本改变规则且不被旧客户端兼容，首先客户端出现了分叉

网络分叉

- 接受新版的节点在网络上运行，其发现的区块将被旧版节点拒绝，旧版节点断开与这些新版节点的连接，因此进一步网络出现了分叉

算力分叉

- 运行不同客户端版本的矿工的算力将逐渐出现分叉

链分叉

- 升级的矿工基于新规则挖矿，而拒绝升级的矿工仍基于旧规则，导致整个区块链出现了分叉

社区分叉

- 每一种区块链的背后都有其对应的社区、开发者、矿工等利益、信仰共同体，链的硬分叉同时也会带来对应社区的分裂：

2016年因 “The DAO事件”
出现的以太坊经典

2016年以太坊一个知名项目The DAO被黑客攻击，损失了价值超过6000万美金的ETH，随后以太坊团队通过硬分叉的方式（变相回滚）“追回”了被黑客盗取资产，一部分社区成员认为此举有违区块链不可回滚、不可篡改的基本精神仍旧坚持维护旧链，自此分裂出——以太坊（ETH）和以太坊经典（ETC）两个独立的区块链项目，对应不同的价值观理念。

2018年比特币现金的进一步分叉

2017年催生出的比特币现金

BCH是从BTC硬分叉而来，当时它把比特币的区块上限由1M修改为8M，目前已经升级为32M。

2018年11月16日BCH正式硬分叉为BCHABC和BCHSV。

对于数字货币持有者来说，硬分叉会让他们额外增加一笔财富（分叉链 Token）不分叉怎么搞钱，一个比特币变成一千多种虚拟币不都是为了两个字，利益

扩展阅读

- <https://zhuanlan.zhihu.com/p/121791191>
- <https://blog.csdn.net/maxdaic/article/details/106450613>
- [Bitcoin's P2P Network](#)
- [Bitcoin DevGuide:P2P Network](#)



The background is a deep blue gradient. It features a complex network of thin, white, intersecting lines that create a sense of depth and connectivity. Scattered throughout this network are numerous circular dots of varying sizes. Some dots are a bright, glowing light blue, while others are a muted, semi-transparent grey. The overall effect is reminiscent of a digital network, a constellation of stars, or a microscopic view of a material structure.

谢谢!