8比特币网络

8.1 P2P网络架构

Bitcoin is structured as a peer-to-peer network architecture on top of the internet. The term peer-to-peer, or P2P, means that the computers that participate in the network are peers to each other, that they are all equal, that there are no "special" nodes, and that all nodes share the burden of providing network services. The network nodes interconnect in a mesh network with a "flat" topology. There is no server, no centralized service, and no hierarchy within the network. Nodes in a P2P network both provide and consume services at the same time with reciprocity acting as the incentive for participation.
比特币是在互联网之上的一个P2P网络架构。
P2P是指参与这个网络的计算机之间彼此对等，没有特殊的节点，所有节点一起承担提供网络服务。
网络节点之间互连成网状网络，具有扁平拓扑。
在这个网络中，没有服务器、没有集中服务、没有层次。
P2P网络的节点同时提供服务和接受服务，以互惠作为参与的激励。

P2P networks are inherently resilient, decentralized, and open. A preeminent example of a P2P network architecture was the early internet itself, where nodes on the IP network were equal. Today’s internet architecture is more hierarchical, but the Internet Protocol still retains its flat-topology essence.
P2P网络天然地具有弹性、去中心化、开放性。
早期的互连网就是P2P网络架构的一个典型用例：IP网络中的节点都是平等的。
当今的互联网架构具有更多层次性，但是IP协议仍然保留了扁平拓扑的本质。

Beyond bitcoin, the largest and most successful application of P2P technologies is file sharing, with Napster as the pioneer and BitTorrent as the most recent evolution of the architecture.
除了比特币，P2P技术的最大和最成功的应用是在文件共享，
Napster是该领域的先锋，BitTorrent是这个架构的最新演进。

Bitcoin’s P2P network architecture is much more than a topology choice. Bitcoin is a P2P digital cash system by design, and the network architecture is both a reflection and a foundation of that core characteristic. Decentralization of control is a core design principle that can only be achieved and maintained by a flat, decentralized P2P consensus network.
比特币的P2P网络架构不仅仅是选择一个拓扑。
比特币被设计为一种P2P的数字现金系统，它的网络架构是这个核心特性的反映和基础。
去中心化控制是核心的设计原则，这个原则只能通过“一个扁平的、去中心化的P2P共识网络”来实现和维护。

The term "bitcoin network" refers to the collection of nodes running the bitcoin P2P protocol. In addition to the bitcoin P2P protocol, there are other protocols such as Stratum that are used for mining and lightweight or mobile wallets. These additional protocols are provided by gateway routing servers that access the bitcoin network using the bitcoin P2P protocol and then extend that network to nodes running other protocols. For example, Stratum servers connect Stratum mining nodes via the Stratum protocol to the main bitcoin network and bridge the Stratum protocol to the bitcoin P2P protocol. We use the term "extended bitcoin network" to refer to the overall network that includes the bitcoin P2P protocol, pool-mining protocols, the Stratum protocol, and any other related protocols connecting the components of the bitcoin system.
“比特币网络”指运行比特币P2P协议的节点的集合。

除了比特币P2P协议之外，还有其它协议，例如Stratum，它用于挖矿、轻量级或移动端钱包。
这些额外的协议是由网关路由服务器提供的，它们使用比特币P2P协议访问比特币网络，然后把网络扩展到运行其它协议的节点。
例如，“Stratum服务器”通过“Stratum协议”将“Stratum挖矿节点”连接到主比特币网络，并将“Stratum协议”桥接到“比特币P2P协议”。
我们使用“扩展比特币网络”指整个网络，它包括比特币P2P协议、矿池协议、Stratum 协议，以及其它相关协议（连接比特币系统的组件）。

8.2 节点类型和角色

Although nodes in the bitcoin P2P network are equal, they may take on different roles depending on the functionality they are supporting. A bitcoin node is a collection of functions: routing, the blockchain database, mining, and wallet services. A full node with all four of these functions is shown in A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing.
虽然比特币P2P网络中的节点都是平等的，但它们根据支持的功能提供了有不同的角色。
比特币节点是下列功能的一个集合：路由、区块链数据库、挖矿、钱包服务。
下图显示了具有全部四种功能的全节点。

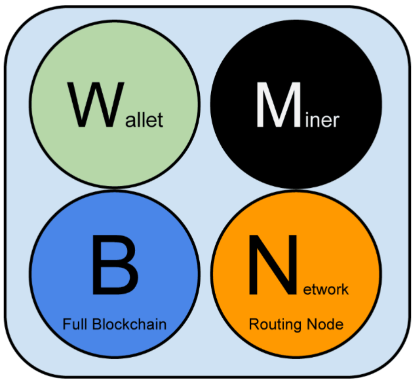


Figure 1. A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing
图8-1具有全部四个功能的比特币网络节点：钱包、矿工、完整的区块链数据库、网络路由

All nodes include the routing function to participate in the network and might include other functionality. All nodes validate and propagate transactions and blocks, and discover and maintain connections to peers. In the full-node example in A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing, the routing function is indicated by an orange circle named "Network Routing Node" or with the letter "N."
所有节点都有路由功能，以参与到网络中，其它功能是可选的。
所有节点都验证和传播交易和区块，并发现和维持与peer的连接。
在图1中，N表示路由功能。

Some nodes, called full nodes, also maintain a complete and up-to-date copy of the blockchain. Full nodes can autonomously and authoritatively verify any transaction without external reference. Some nodes maintain only a subset of the blockchain and verify transactions using a method called *simplified payment verification*, or SPV. These nodes are known as SPV nodes or lightweight nodes. In the full-node example in the figure, the full-node blockchain database function is indicated by a circle called "Full Blockchain" or the letter "B." In The extended bitcoin network showing various node types, gateways, and protocols, SPV nodes are drawn without the "B" circle, showing that they do not have a full copy of the blockchain.
全节点还维护一份完整、最新的区块链。全节点能够自主和可信地验证任何交易，不需要参考外部。

有些节点只维护区块链的一部分，它们使用SPV方法来验证交易。这种节点称为“SPV节点”或“轻量级节点”。
在图1中，B表示全节点区块链数据库功能。
在图3中，SPV节点没有蓝色圆圈，表示它们没有完整的区块链。

Mining nodes compete to create new blocks by running specialized hardware to solve the Proof-of-Work algorithm. Some mining nodes are also full nodes, maintaining a full copy of the blockchain, while others are lightweight nodes participating in pool mining and depending on a pool server to maintain a full node. The mining function is shown in the full node as a black circle called "Miner" or the letter "M."
挖矿节点通过运行特殊硬件来解决工作量证明算法，以赢得创建新区块的竞赛。
有些挖矿节点也是全节点，维护一个完整的区块链；
有些矿池节点是轻量级节点，它们参与矿池，依赖矿池服务器来维护一个全节点。
在图1中，M表示挖矿功能。

User wallets might be part of a full node, as is usually the case with desktop bitcoin clients. Increasingly, many user wallets, especially those running on resource-constrained devices such as smartphones, are SPV nodes. The wallet function is shown in A bitcoin network node with all four functions: wallet, miner, full blockchain database, and network routing as a green circle called "Wallet" or the letter "W."
用户钱包可以是全节点的一部分，桌面比特币客户端同行就是这样。
越来越多的用户钱包是SPV节点，尤其是运行于诸如手机等资源受限设备上的钱包。
在图1中，W表示钱包功能。

比特币P2P协议上的节点的功能	说明	节点类型
N路由	验证和传播交易和区块，发现和维持与peer的连接。	所有节点都有这个功能
B完整区块链数据库	能够自主和可信地验证任何交易，不需要参考外部	全节点、SPV节点
M挖矿	通过解决工作量证明算法，赢得创建新区块的竞赛。	全节点、矿池矿工
W钱包	用户钱包功能	全节点、SPV节点

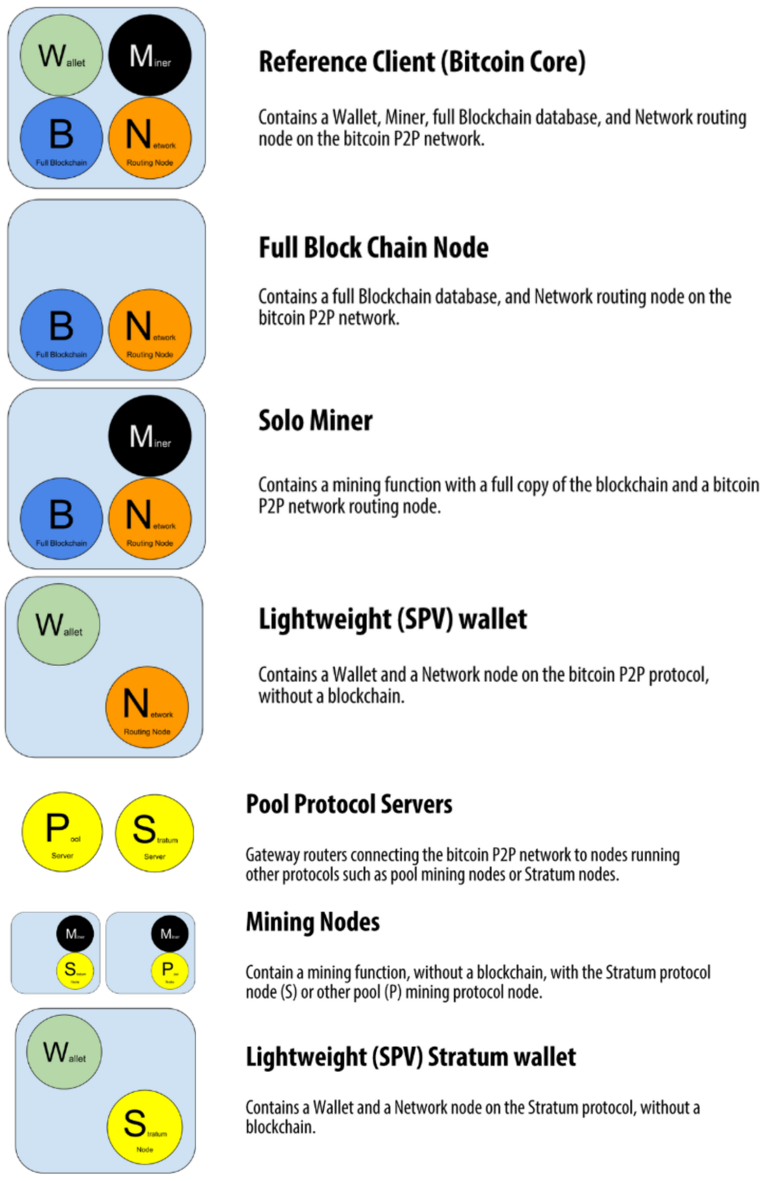
In addition to the main node types on the bitcoin P2P protocol, there are servers and nodes running other protocols, such as specialized mining pool protocols and lightweight client-access protocols.
除了比特币P2P协议上的主要节点类型之外，还有一些服务器和节点运行其它协议，例如：专门的矿池协议、轻量级客户端访问协议。

Different types of nodes on the extended bitcoin network shows the most common node types on the extended bitcoin network.
图2显示了“扩展比特币网络”中最为常见的节点类型。

Figure 2. Different types of nodes on the extended bitcoin network
图2 扩展比特币网络上的节点类型

节点类型	有的功能	没有的功能
参考客户端: Bitcoin Core	路由、完整区块链、挖矿、钱包	
全区块链节点	路由、完整区块链	钱包，挖矿
独立矿工	路由、完整区块链、挖矿	钱包
SPV钱包	路由、钱包	完整区块链、挖矿

矿池协议服务器	网关路由服务器，连接比特币P2P网络到运行其它协议的节点，例如矿池挖矿节点或Stratum节点	
矿机	挖矿，用Stratum协议节点，或其它矿池挖矿协议节点	区块链
SPV Stratum钱包	包含Stratum协议上的一个钱包和一个网络节点	区块链



Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.

Full Block Chain Node

Contains a full Blockchain database, and Network routing node on the bitcoin P2P network.

Solo Miner

Contains a mining function with a full copy of the blockchain and a bitcoin P2P network routing node.

Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.

Pool Protocol Servers

Gateway routers connecting the bitcoin P2P network to nodes running other protocols such as pool mining nodes or Stratum nodes.

Mining Nodes

Contain a mining function, without a blockchain, with the Stratum protocol node (S) or other pool (P) mining protocol node.

Lightweight (SPV) Stratum wallet

Contains a Wallet and a Network node on the Stratum protocol, without a blockchain.

8.3 扩展比特币网络

The main bitcoin network, running the bitcoin P2P protocol, consists of between 5,000 and 8,000 listening nodes running various versions of the bitcoin reference client (Bitcoin Core) and a few hundred nodes running various other implementations of the bitcoin P2P protocol, such as Bitcoin Classic, Bitcoin Unlimited, BitcoinJ, Libbitcoin, btcd, and bcoin.
主比特币网络运行比特币P2P协议,包含:

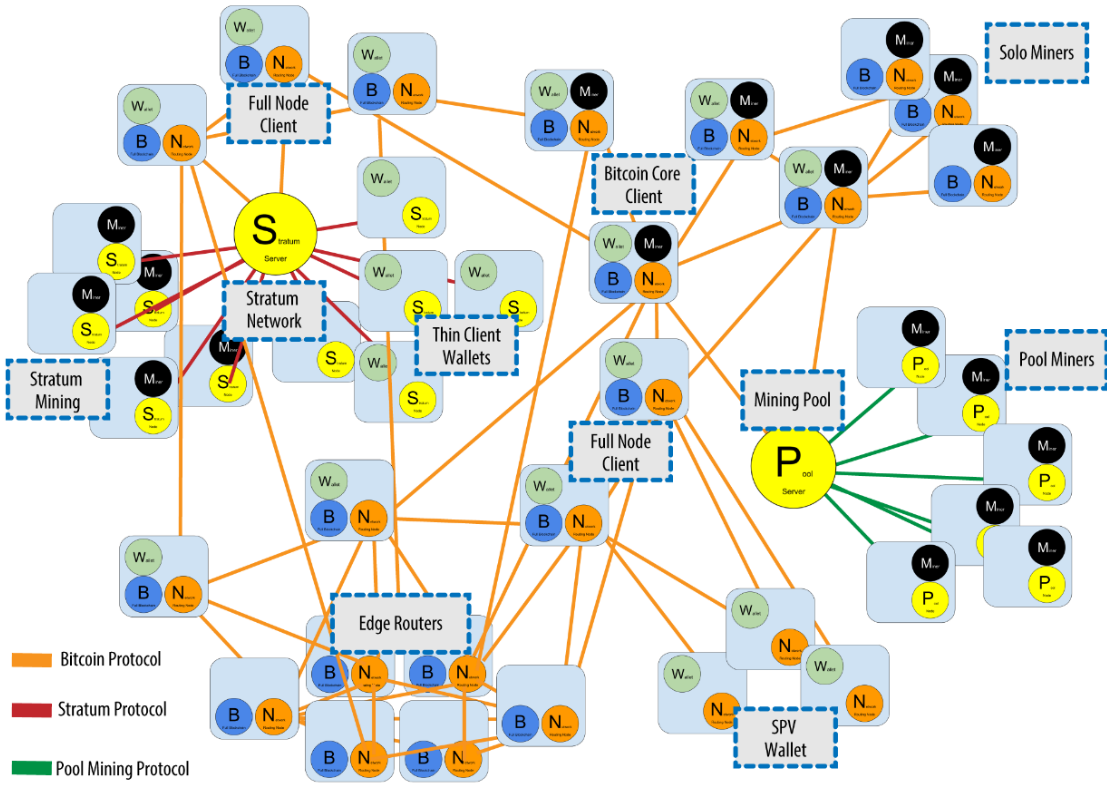
- 大约5000-8000个监听节点：运行各种版本Bitcoin Core
- 几百个节点：运行比特币P2P协议的各种其它实现，例如Bitcoin Classic, Bitcoin Unlimited, BitcoinJ, Libbitcoin, btcd, bcoin等。

A small percentage of the nodes on the bitcoin P2P network are also mining nodes, competing in the mining process, validating transactions, and creating new blocks. Various large companies interface with the bitcoin network by running full-node clients based on the Bitcoin Core client, with full copies of the blockchain and a network node, but without mining or wallet functions. These nodes act as network edge routers, allowing various other services (exchanges, wallets, block explorers, merchant payment processing) to be built on top.
比特币P2P网络中的一小部分节点也是挖矿节点，它们竞争挖矿、验证交易、创建新区块。
各种大公司通过运行基于Bitcoin Core客户端的全节点连接到比特币网络，有完整的区块链和一个网络节点，但没有挖矿或钱包功能。
这些节点作为网络边缘路由器，可以在它们上面构建各种其它服务，例如：交易所、钱包、区块浏览器、商家支付处理、等。

The extended bitcoin network includes the network running the bitcoin P2P protocol, described earlier, as well as nodes running specialized protocols. Attached to the main bitcoin P2P network are a number of pool servers and protocol gateways that connect nodes running other protocols. These other protocol nodes are mostly pool mining nodes (see [mining]) and lightweight wallet clients, which do not carry a full copy of the blockchain.
扩展比特币网络既包括运行比特币P2P协议的网络，还包含运行专门协议的网络节点。
一些矿池服务器和协议网关连接到主比特币P2P网络，它们连接运行其它协议的节点。
这些其它协议节点多数是矿池挖矿节点和轻量级钱包客户端，它们没有完整的区块链。

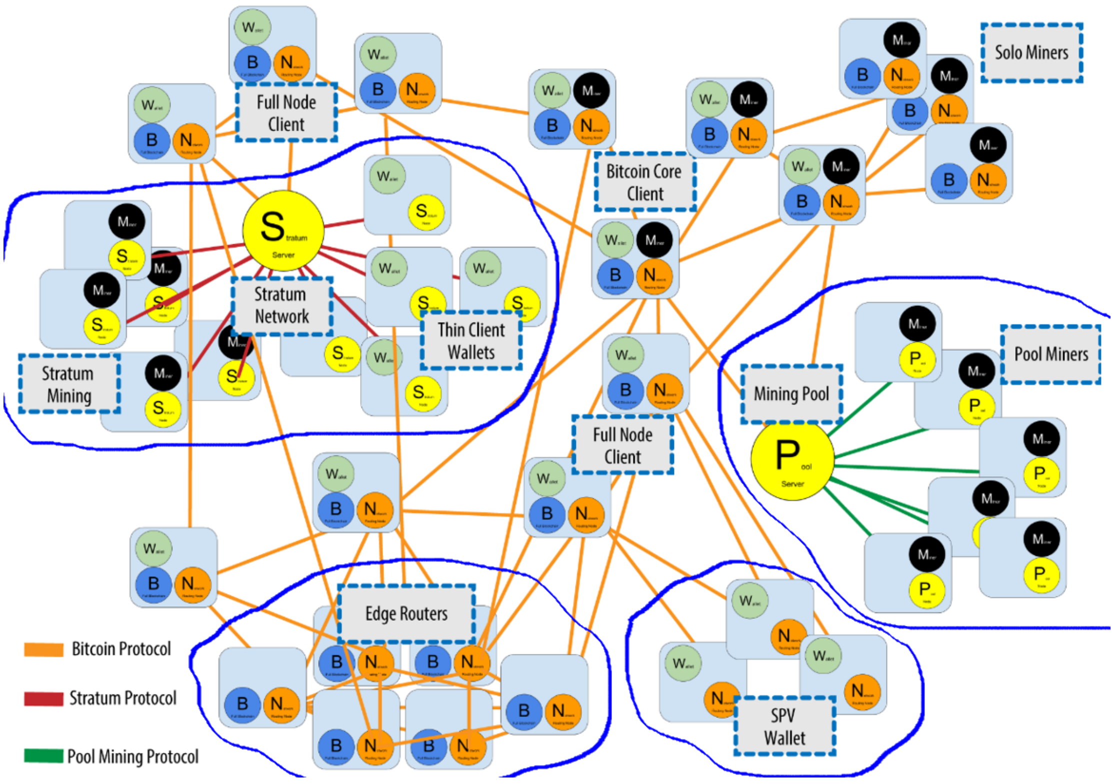
The extended bitcoin network showing various node types, gateways, and protocols shows the extended bitcoin network with the various types of nodes, gateway servers, edge routers, and wallet clients and the various protocols they use to connect to each other.
图3显示了扩展比特币网络，它包括：各种节点类型、网关服务器、边缘路由器、钱包客户端，以及连接它们的各类协议。

Figure 3. The extended bitcoin network showing various node types, gateways, and protocols
图3：扩展比特币网络，有各种节点类型、网关和协议



ddk说明：

- Stratum协议网络：
  - 组成：Stratum协议、服务器、矿机、瘦客户端钱包
  - Stratum：矿机与矿池软件之间的通讯协议
  - 矿机和瘦客户端钱包都通过Stratum协议单独与服务器连接
  - 服务器通过比特币协议与全节点连接
- 矿池协议网络：（与Stratum协议网络类似）
  - 组成：矿池挖矿协议、挖矿服务器、矿池矿工
  - 矿池挖矿协议：矿池服务器与矿池矿工之间的通讯协议
  - 矿池矿工都通过协议单独与矿池服务器连接
  - 矿池服务器通过比特币协议与全节点连接
- 比特币网络
  - SPV钱包（NW）：单独与全节点连接
  - 边缘路由器（NB）：其它四种彼此互连
  - 全节点客户端（NBW）
  - 独立矿工（NBM）
  - Bitcoin Core客户端（NBWM）


Full Node Client
Solo Miners
Bitcoin Core Client
Stratum Server
Stratum Network
Thin Client Wallets
Stratum Mining
Mining Pool
Pool Miners
Pool Server
Full Node Client
Edge Routers
SPV Wallet
Bitcoin Protocol
Stratum Protocol
Pool Mining Protocol

8.4比特币传播网络

While the bitcoin P2P network serves the general needs of a broad variety of node types, it exhibits too high network latency for the specialized needs of bitcoin mining nodes.
虽然比特币P2P网络为各种节点类型的一般需求提供服务，
但对比特币挖矿节点的专门需求来说，它的网络延迟太高了。

Bitcoin miners are engaged in a time-sensitive competition to solve the Proof-of-Work problem and extend the blockchain (see [mining]). While participating in this competition, bitcoin miners must minimize the time between the propagation of a winning block and the beginning of the next round of competition. In mining, network latency is directly related to profit margins.
比特币矿工进行的是争分夺秒的竞争，以解决工作量证明问题和扩展区块链。
在参加这个比赛时，比特币矿工必须尽可能缩短"获胜区块的传播"与"下一轮比赛开始"之间的时间。对挖矿来说，网络延迟与利润率直接相关。

A *Bitcoin Relay Network* is a network that attempts to minimize the latency in the transmission of blocks between miners. The original Bitcoin Relay Network was created by core developer Matt Corallo in 2015 to enable fast synchronization of blocks between miners with very low latency. The network consisted of several specialized nodes hosted on the Amazon Web Services infrastructure around the world and served to connect the majority of miners and mining pools.
比特币传播网络（BRN）企图尽可能缩短矿工之间传输区块的延迟。
最初的BRN是由核心开发者Matt Corallo于2015年创建的，目的是用非常短的延迟在矿工之间快速同步区块。这个网络包含几个专门节点，由世界各地的亚马逊Web服务基础架构托管，用于连接大多数矿工和矿池。

The original Bitcoin Relay Network was replaced in 2016 with the introduction of the *Fast Internet Bitcoin Relay Engine* or *FIBRE*, also created by core developer Matt Corallo. FIBRE is a UDP-based relay network that relays blocks within a network of nodes. FIBRE implements *compact block* optimization to further reduce the amount of data transmitted and the network latency.
最初的BRN在2016年被FIBRE替代，它也由Matt Corallo创建的。
FIBRE是一种基于UDP的传播网络，它在节点网络内传播区块。
FIBER实现了紧凑区块优化，以进一步减少传输的数据量和网络延迟。

Another relay network (still in the proposal phase) is *Falcon*, based on research at Cornell University. Falcon uses "cut-through-routing" instead of "store-and-forward" to reduce latency by propagating parts of blocks as they are received rather than waiting until a complete block is received.
另一个传播网络（仍在建议阶段）是 Falcon，它是基于康奈尔大学的研究。
Falcon使用"直通路由"（不是"存储转发"）来减少延迟，它传播接收到的区块的部分，而不是等着接收完一个完整区块才传播。

Relay networks are not replacements for bitcoin's P2P network. Instead they are overlay networks that provide additional connectivity between nodes with specialized needs. Like freeways are not replacements for rural roads, but rather shortcuts between two points with heavy traffic, you still need small roads to connect to the freeways.
传播网络不是要替代比特币P2P网络。
它们是上层（overlay）网络，在有特殊需要的节点之间提供额外的连接性。
就像高速公路不能替代乡村道路一样，高速公路是在交通繁忙的两点之间的快捷方式，但你仍然需要小路连接到高速公路。

8.5 网络发现

When a new node boots up, it must discover other bitcoin nodes on the network in order to participate. To start this process, a new node must discover at least one existing node on the network and connect to it. The geographic location of other nodes is irrelevant; the bitcoin network topology is not geographically defined. Therefore, any existing bitcoin nodes can be selected at random.
当一个新的网络节点启动时，为了能够参与，它必须发现网络上的其它比特币节点。
为了开始这个过程，新节点必须发现网络上至少一个现有节点，并与它连接。
其它节点的地理位置无关紧要，比特币网络拓扑不是由地理位置定义。
因此，可以随机选择任何现有节点。

To connect to a known peer, nodes establish a TCP connection, usually to port 8333 (the port generally known as the one used by bitcoin), or an alternative port if one is provided. Upon establishing a connection, the node will start a "handshake" (see The initial handshake between peers) by transmitting a version message, which contains basic identifying information, including:
为了连接到一个已知的peer，节点建立一个TCP连接，通常是到端口8333，或者节点提供的其它端口。
一旦建立了连接，这个节点会启动一个“握手”，方法是传输一个version消息，它包含基本的验证信息：

version消息	说明
nVersion	The bitcoin P2P protocol version the client "speaks" (e.g., 70002) 比特币P2P协议版本，例如70002
nLocalServices	A list of local services supported by the node, currently just NODE_NETWORK 节点支持的本地服务列表，目前只有NODE_NETWORK
nTime	The current time 当前时间
addrYou	The IP address of the remote node as seen from this node 远端节点的IP地址
addrMe	The IP address of the local node, as discovered by the local node 本地节点的IP地址
subver	A sub-version showing the type of software running on this node (e.g., /Satoshi:0.9.2.1/) 子版本，显示本节点运行的软件的类型，例如：/Satoshi:0.9.2.1/
BestHeight	The block height of this node’s blockchain 本节点的区块链的区块高度

(See GitHub for an example of the version network message.)
version网络消息的例子参见GitHub

The version message is always the first message sent by any peer to another peer. The local peer receiving a version message will examine the remote peer’s reported nVersion and decide if the remote peer is compatible. If the remote peer is compatible, the local peer will acknowledge the version message and establish a connection by sending a verack.
version消息是peer之间发送的第一个消息。
接收到一个version消息后，会检查远端peer的nVersion，确定与远端peer是否兼容。
如果兼容，则本地peer会确认这个version消息，通过发送一个verack来建立连接。

How does a new node find peers?
一个新节点如何找到其它peer呢?

The first method is to query DNS using a number of "DNS seeds," which are DNS servers that provide a list of IP addresses of bitcoin nodes. Some of those DNS seeds provide a static list of IP addresses of stable bitcoin listening nodes. Some of the DNS seeds are custom implementations of BIND (Berkeley Internet Name Daemon) that return a random subset from a list of bitcoin node addresses collected by a crawler or a long-running bitcoin node.
第1种方法：使用一些“DNS种子”来询问DNS，这些“DNS种子”是DNS服务器，它们提供比特币节点的IP地址列表。
一些“DNS种子”提供了“稳定的比特币侦听节点”的静态IP地址列表。
一些“DNS种子”是BIND的定制实现，它返回比特币节点地址列表的一个随机子集，这些地址由爬虫或长期运行的比特币节点收集的。

The Bitcoin Core client contains the names of five different DNS seeds. The diversity of ownership and diversity of implementation of the different DNS seeds offers a high level of reliability for the initial bootstrapping process. In the Bitcoin Core client, the option to use the DNS seeds is controlled by the option switch -dnsseed (set to 1 by default, to use the DNS seed).
Bitcoin Core客户端包含五个DNS种子的名称。
不同DNS种子的所有权差异和实现差异，为初始启动过程提供了很高的可靠性。
在Bitcoin Core客户端中，使用“DNS种子”的选项是由选项开关-dnsseed控制（默认是1即使用DNS种子）。

Alternatively, a bootstrapping node that knows nothing of the network must be given the IP address of at least one bitcoin node, after which it can establish connections through further introductions. The command-line argument -seednode can be used to connect to one node just for introductions using it as a seed. After the initial seed node is used to form introductions, the client will disconnect from it and use the newly discovered peers.
第2种方法：给新节点提供至少一个比特币节点的IP地址，然后它可以进一步介绍来建立更多连接。
命令行参数-seednode可用于连接到一个节点，把它当做种子。
在使用初始种子节点完成介绍后，客户端与种子节点断开连接，使用新发现的peer。

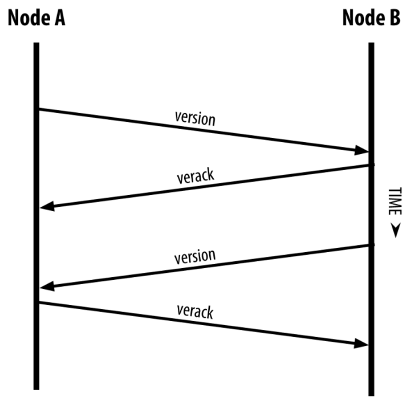


图8-4 对等体之间的初始握手

Figure 4. The initial handshake between peers
图4：peer之间的初始握手

Once one or more connections are established, the new node will send an addr message containing its own IP address to its neighbors. The neighbors will, in turn, forward the addr message to their neighbors, ensuring that the newly connected node becomes well known and better connected.
当建立了一个或多个连接，新节点会发送一个addr消息，包含它的IP地址。
邻居再把这个addr消息转发给它的邻居，从而保证新节点被大家知道，并被更好地连接。

Additionally, the newly connected node can send getaddr to the neighbors, asking them to return a list of IP addresses of other peers. That way, a node can find peers to connect to and advertise its existence on the network for other nodes to find it. Address propagation and discovery shows the address discovery protocol.
此外，新节点可以给邻居发送getaddr消息，要求它们返回其它peer的IP地址列表。
这样，一个节点可以找到要连接的peer，并向其它节点通告它存在于网络上。
图5显示了这个地址发现协议。

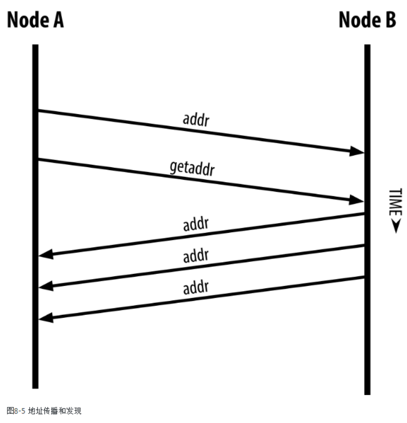


Figure 5. Address propagation and discovery

A node must connect to a few different peers in order to establish diverse paths into the bitcoin network. Paths are not reliable—nodes come and go—and so the node must continue to discover new nodes as it loses old connections as well as assist other nodes when they bootstrap.
一个节点必须连接到一些不同的peer，以便建立到比特币网络的不同路径。
由于节点可以随时加入和离开，所以路径是不可靠的，因此，节点在失去旧连接时必须继续发现新节点，并且在其它节点启动时帮助它们。

Only one connection is needed to bootstrap, because the first node can offer introductions to its peer nodes and those peers can offer further introductions. It's also unnecessary and wasteful of network resources to connect to more than a handful of nodes. After bootstrapping, a node will remember its most recent successful peer connections, so that if it is rebooted it can quickly reestablish connections with its former peer network. If none of the former peers respond to its connection request, the node can use the seed nodes to bootstrap again.
节点启动只需一个连接，因为第一个节点可以将它引荐给自己的peer，而这些peer可以提供进一步的引荐。一个节点如果连接很多节点，既没必要，也是浪费网络资源。
在启动后，节点会记住它最近成功连接的peer；当重新启动后，它可以迅速与先前的peer网络建立连接。
如果先前的peer没有一个响应它的连接请求，这个节点可以使用"种子节点"再次启动。

On a node running the Bitcoin Core client, you can list the peer connections with the command getpeerinfo:
在运行Bitcoin Core客户端的一个节点上，可以用 getpeerinfo 命令列出peer连接信息：

$ bitcoin-cli getpeerinfo

```
{
    "addr"          : "85.213.199.39:8333",
    "services"      : "00000001",
    "lastsend"      : 1405634126,
    "lastrecv"      : 1405634127,
    "bytessent"     : 23487651,
    "bytesrecv"     : 138679099,
    "conntime"      : 1405021768,
    "pingtime"      : 0.00000000,
    "version"    : 70002,
    "subver"     : "/Satoshi:0.9.2.1/",
    "inbound"    : false,
    "startingheight" : 310131,
    "banscore"   : 0,
    "syncnode"   : true
},
{
    "addr"       : "58.23.244.20:8333",
    "services"   : "00000001",
    "lastsend"   : 1405634127,
    "lastrecv"   : 1405634124,
    "bytessent"  : 4460918,
    "bytesrecv"  : 8903575,
    "conntime"   : 1405559628,
    "pingtime"   : 0.00000000,
    "version"    : 70001,
    "subver"     : "/Satoshi:0.8.6/",
    "inbound"    : false,
    "startingheight" : 311074,
    "banscore"   : 0,
    "syncnode"   : false
}
```

To override the automatic management of peers and to specify a list of IP addresses, users can provide the option -connect=<IPAddress> and specify one or more IP addresses. If this option is used, the node will only connect to the selected IP addresses, instead of discovering and maintaining the peer connections automatically.
为了覆盖自动管理的peer，指定一个IP地址列表，用户可以提供选项 -connect=<IP地址> ，指定一个或多个IP地址。
如果使用了这个选项，这个节点只连接到选定的IP地址，不会自动发现和维护peer连接。

If there is no traffic on a connection, nodes will periodically send a message to maintain the connection. If a node has not communicated on a connection for more than 90 minutes, it is assumed to be disconnected and a new peer will be sought. Thus, the network dynamically adjusts to transient nodes and network problems, and can organically grow and shrink as needed without any central control.
如果连接上没有流量，节点会定期发送一个信息来维持这个连接。
如果节点在某个连接上超过90分钟没有任何通信，就认为与这个peer断开了，会找一个新的peer。
因此，这个网络动态适应瞬态节点和网络问题，可以根据需要有机地生长和收缩，而没有任何中央控制。

8.6 全节点

Full nodes are nodes that maintain a full blockchain with all transactions. More accurately, they probably should be called "full blockchain nodes." In the early years of bitcoin, all nodes were full nodes and currently the Bitcoin Core client is a full blockchain node. In the past two years, however, new forms of bitcoin clients have been introduced that do not maintain a full blockchain but run as lightweight clients. We’ll examine these in more detail in the next section.
“全节点”维持完整的区块链，有所有的交易，更加准确地说，应当称为“全区块链节点”。
在比特币的早期，所有节点都是全节点；现在，Bitcoin Core客户端是全区块链节点。
但在过去两年，出现了新型的比特币客户端，它们没有维持完整的区块链，而是作为轻量级客户端运行。

Full blockchain nodes maintain a complete and up-to-date copy of the bitcoin blockchain with all the transactions, which they independently build and verify, starting with the very first block (genesis block) and building up to the latest known block in the network. A full blockchain node can independently and authoritatively verify any transaction without recourse or reliance on any other node or source of information. The full blockchain node relies on the network to receive updates about new blocks of transactions, which it then verifies and incorporates into its local copy of the blockchain.
全区块链节点维持一个完整的、最新的比特币区块链，有所有的交易，它们独立地构建和验证这个区块链，从第一个区块开始，构建到网络中最新的区块。
全区块链节点可以独立和自主地验证任何交易，而不需要求助或依靠任何其它节点或信息源。
全区块链节点依赖这个网络来接收有关交易的新区块的更新，对它进行验证，把它计入本地区块链中。

Running a full blockchain node gives you the pure bitcoin experience: independent verification of all transactions without the need to rely on, or trust, any other systems. It’s easy to tell if you’re running a full node because it requires more than one hundred gigabytes of persistent storage (disk space) to store the full blockchain. If you need a lot of disk and it takes two to three days to sync to the network, you are running a full node. That is the price of complete independence and freedom from central authority.
运行一个全区块链节点给你一个纯粹的比特币体验：独立地验证所有的交易，不需要依赖或信任任何其它系统。
很容易知道你是否在运行一个全节点：它需要100多G的磁盘空间，来存储完整的区块链。
如果你需要大量磁盘，花了好几天来与网络同步，你就是在运行一个全节点。
这就是摆脱中心权威，获得完全的独立和自由所要付出的代价。

There are a few alternative implementations of full blockchain bitcoin clients, built using different programming languages and software architectures. However, the most common implementation is the reference client Bitcoin Core, also known as the Satoshi client. More than 75% of the nodes on the bitcoin network run various versions of Bitcoin Core. It is identified as "Satoshi" in the sub-version string sent in the version message and shown by the command getpeerinfo as we saw earlier; for example, /Satoshi:0.8.6/.
全区块链比特币客户端也有一些其它实现，它们使用不同的编程语言和软件架构。
但是，最常见的实现参考客户端Bitcoin Core，也称为“中本聪客户端”。
比特币网络中超过75%的节点运行各种版本的Bitcoin Core。
在version消息的sub-version字段，使用"Satoshi"来表示Bitcoin core，可以用getpeerinfo命令看到看它，例如：/Satoshi:0.8.6/。

8.7 交换Inventory

The first thing a full node will do once it connects to peers is try to construct a complete blockchain. If it is a brand-new node and has no blockchain at all, it only knows one block, the genesis block, which is statically embedded in the client software. Starting with block #0 (the genesis block), the new node will have to download hundreds of thousands of blocks to synchronize with the network and reestablish the full blockchain.
当全节点连接到peer时，做的第一件事是尝试构建一个完整的区块链。
如果它是一个全新的节点，它只知道一个区块，创世区块，这个区块是写在客户端软件中的。
从区块链0（创世区块）开始，新节点要下载几十万个区块，才能与网络同步，并重建完整的区块链。

The process of syncing the blockchain starts with the version message, because that contains BestHeight, a node’s current blockchain height (number of blocks). A node will see the version messages from its peers, know how many blocks they each have, and be able to compare to how many blocks it has in its own blockchain. Peered nodes will exchange a getblocks message that contains the hash (fingerprint) of the top block on their local blockchain. One of the peers will be able to identify the received hash as belonging to a block that is not at the top, but rather belongs to an older block, thus deducing that its own local blockchain is longer than its peer’s.
同步区块链的过程从version消息开始，因为它有BestHeight字段，表示本节点的当前区块链的区块高度（区块数量）。
收到version消息的节点知道了对方有多少个区块，能够比较自己的区块链中有多少个区块。
这个两个节点就会交换一个getblocks消息，其中包含本地区块链的顶端区块的哈希（指纹）。
其中一个节点就能识别出，收到的哈希不是顶端上区块，而是一个老的区块，因此推断出自己的区块链比peer的区块链更长。

The peer that has the longer blockchain has more blocks than the other node and can identify which blocks the other node needs in order to "catch up." It will identify the first 500 blocks to share and transmit their hashes using an inv (inventory) message. The node missing these blocks will then retrieve them, by issuing a series of getdata messages requesting the full block data and identifying the requested blocks using the hashes from the inv message.
具有更长区块链的节点比它的peer有更多的区块，并且能够识别出peer需要哪些区块才能赶上来。
它就识别出要共享的前500个区块，使用一个inv消息传送它们的哈希。
然后，缺少这些区块的节点会获取这些区块：使用来自inv消息的哈希来识别需要请求的区块；发出一系列getdata消息来，请求完整的区块数据。

Let’s assume, for example, that a node only has the genesis block. It will then receive an inv message from its peers containing the hashes of the next 500 blocks in the chain. It will start requesting blocks from all of its connected peers, spreading the load and ensuring that it doesn’t overwhelm any peer with requests. The node keeps track of how many blocks are "in transit" per peer connection, meaning blocks that it has requested but not received, checking that it does not exceed a limit (MAX_BLOCKS_IN_TRANSIT_PER_PEER). This way, if it needs a lot of blocks, it will only request new ones as previous requests are fulfilled, allowing the peers to control the pace of updates and not overwhelm the network. As each block is received, it is added to the blockchain, as we will see in [blockchain]. As the local blockchain is gradually built up, more blocks are requested and received, and the process continues until the node catches up to the rest of the network.
例如，一个节点只有创世区块。
它收到了peer的inv消息，其中包含了区块链中下500个区块的哈希。
于是，它开始向所有与之相连的peer请求这些区块，使用负载分担，保证不会孤独使用某个peer。
这个节点追踪每个peer连接上“正在传输”的区块数量（即它已经发出了请求但还没有接收到），并且检查这个数量没有超过限制（ MAX_BLOCKS_IN_TRANSIT_PER_PEER ）。
这样，如果一个节点需要大量区块，它只会在先前请求被满足之后，才会请求新的区块，这样，就允许节点控制更新的速度，不至于压垮网络。

当收到一个区块时，就把它加入区块链。当本地区块链逐渐构建起来时，就请求和接收更多的区块，持续这个过程，知道本节点赶上这个网络的其它节点。

This process of comparing the local blockchain with the peers and retrieving any missing blocks happens any time a node goes offline for any period of time. Whether a node has been offline for a few minutes and is missing a few blocks, or a month and is missing a few thousand blocks, it starts by sending getblocks, gets an inv response, and starts downloading the missing blocks. Node synchronizing the blockchain by retrieving blocks from a peer shows the inventory and block propagation protocol.
当一个节点离线任意一段时间，又重新连接时，会再次执行这个过程：与peer比较本地区块链，获取任何本地缺失的区块。
节点发送getblocks，获取一个inv响应，然后开始下载缺失的区块。
图6显示了这个库存和区块传播协议。

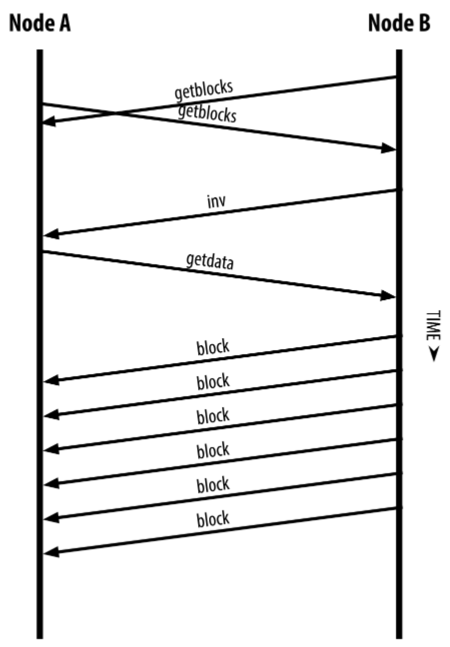


图8-6节点通过从对等体检索区块同步区块链

Figure 6. Node synchronizing the blockchain by retrieving blocks from a peer
图6：节点通过从peer获取区块来同步区块链

8.8 SPV节点

Not all nodes have the ability to store the full blockchain. Many bitcoin clients are designed to run on space- and power-constrained devices, such as smartphones, tablets, or embedded systems. For such devices, a *simplified payment verification* (SPV) method is used to allow them to operate without storing the full blockchain. These types of clients are called SPV clients or lightweight clients. As bitcoin adoption surges, the SPV node is becoming the most common form of bitcoin node, especially for bitcoin wallets.
并非所有节点都有能力储存完整的区块链。
许多比特币客户端运行在空间和电力有限的设备上，例如手机、平板电脑、嵌入式系统等。
对于这种设备，使用SPV方法可以在不必存储完整区块链的情况下工作。
这种类型的客端称为SPV客户端，或轻量级客户端。
随着比特币的使用兴起，SPV节点正在成为比特币节点的最常见形式，特别是比特币钱包。

SPV nodes download only the block headers and do not download the transactions included in each block. The resulting chain of blocks, without transactions, is 1,000 times smaller than the full blockchain. SPV nodes cannot construct a full picture of all the UTXOs that are available for spending because they do not know about all the transactions on the network. SPV nodes verify transactions using a slightly different method that relies on peers to provide partial views of relevant parts of the blockchain on demand.
SPV节点下载区块头，但不下载每个区块中包含的交易。
对于这种没有交易信息的区块链，是完整区块链的1/1000。
SPV节点不能构建所有可用于花费的UTXO的完整视图，因为它们不知道网络上的所有交易。
SPV节点使用另一种方法来验证交易，这个方法依赖peer根据需要提供区块链的相关部门的部分视图。

As an analogy, a full node is like a tourist in a strange city, equipped with a detailed map of every street and every address. By comparison, an SPV node is like a tourist in a strange city asking random strangers for turn-by-turn directions while knowing only one main avenue. Although both tourists can verify the existence of a street by visiting it, the tourist without a map doesn’t know what lies down any of the side streets and doesn’t know what other streets exist. Positioned in front of 23 Church Street, the tourist without a map cannot know if there are a dozen other "23 Church Street" addresses in the city and whether this is the right one. The mapless tourist’s best chance is to ask enough people and hope some of them are not trying to mug him.
打个比方，我们把全节点和SPV节点比作一个城市的两名游客。
全节点带着一张详细地图，包含每条街道和每个地址的详细信息。
SPV节点只知道一条主干道的名字，在这种情况下，他要询问当地人。
虽然两名游客都可以通过实地考察来验证一条街道是否存在，但SPV节点不知道每个巷子里有哪些街道，也不知道附近还有什么其它街道。他在“23号教堂街”的前面，并不知道这个城市里是否还有其它“23号教堂街”，也不知道面前的这个街道是否是他要找的那个街道。对他来说，最好的方式就是向很多人问路，并且希望其中一部分人不会骗他。

SPV verifies transactions by reference to their *depth* in the blockchain instead of their *height*. Whereas a full blockchain node will construct a fully verified chain of thousands of blocks and transactions reaching down the blockchain (back in time) all the way to the genesis block, an SPV node will verify the chain of all blocks (but not all transactions) and link that chain to the transaction of interest.
SPV验证交易的方法是参考交易在区块链中的深度，而不是在区块链中的高度。
全区块链节点会构造有成千上万的区块和交易的一个完全验证的链，这样一路回溯，可回到创世区块。
SPV节点会验证所有区块的链（不是所有交易），并把这个链与感兴趣的交易联系起来。

For example, when examining a transaction in block 300,000, a full node links all 300,000 blocks down to the genesis block and builds a full database of UTXO, establishing the validity of the transaction by confirming that the UTXO remains unspent. An SPV node cannot validate whether the UTXO is unspent. Instead, the SPV node will establish a link between the transaction and the block that contains it, using a *merkle path* (see [merkle_trees]). Then, the SPV node waits until it sees the six blocks 300,001

through 300,006 piled on top of the block containing the transaction and verifies it by establishing its depth under blocks 300,006 to 300,001. The fact that other nodes on the network accepted block 300,000 and then did the necessary work to produce six more blocks on top of it is proof, by proxy, that the transaction was not a double-spend.
例如，当检查区块300,000中的一个交易时，全节点会链接所有300,000个区块，一路回到创世区块，并构建UTXO的一个完整数据库，通过确认这个UTXO还未被花费，来完成对这个交易的验证。
SPV节点不能验证这个UTXO是否未被花费。SPV节点使用一个merkle路径，来建立这个交易与区块300,000之间的联系。
然后，SPV节点等待，直到它看到区块300,001到300,006的六个区块出现在区块300,000之上，通过建立它的深度来验证这个交易。
这个事实证明了：网络上的其它节点接受了区块300,000，并且又生成了6个区块；通过代理，证明这个交易没有被双花。

An SPV node cannot be persuaded that a transaction exists in a block when the transaction does not in fact exist. The SPV node establishes the existence of a transaction in a block by requesting a merkle path proof and by validating the Proof-of-Work in the chain of blocks. However, a transaction’s existence can be "hidden" from an SPV node. An SPV node can definitely prove that a transaction exists but cannot verify that a transaction, such as a double-spend of the same UTXO, doesn’t exist because it doesn’t have a record of all transactions.
如果一个交易实际上不存在，SPV节点不会认为该交易存在于一个区块中。
SPV节点认为交易存在的方法是：请求一个merkle路径证明，验证区块链中的工作量证明。
但是，SPV节点可能看不到某个交易的存在。
SPV节点可以确定地证明某个交易存在，但它不能验证某个交易不存在，因为它没有记录所有的交易。

This vulnerability can be used in a denial-of-service attack or for a double-spending attack against SPV nodes. To defend against this, an SPV node needs to connect randomly to several nodes, to increase the probability that it is in contact with at least one honest node. This need to randomly connect means that SPV nodes also are vulnerable to network partitioning attacks or Sybil attacks, where they are connected to fake nodes or fake networks and do not have access to honest nodes or the real bitcoin network.
这个漏洞可能被针对SPV节点的DoS攻击或双花攻击利用。
为了防御这些攻击，SPV节点需要随机连接多个节点，以增加与至少一个可靠节点相连接的可能性。
需要随机连接也意味着，SPV节点还容易受到网络分区攻击或Sybil攻击，这时它们被连接到假节点或假网络，没有通向可靠节点或真实比特币网络的连接。

For most practical purposes, well-connected SPV nodes are secure enough, striking a balance between resource needs, practicality, and security. For infallible security, however, nothing beats running a full blockchain node.
为了最实用的目的，具有良好连接的SPV节点是足够安全的，它在资源需求、实用性和安全性之间实现一个平衡。
但是，为了绝对安全，要运行一个全区块链节点。

Tip：A full blockchain node verifies a transaction by checking the entire chain of thousands of blocks below it in order to guarantee that the UTXO is not spent, whereas an SPV node checks how deep the block is buried by a handful of blocks above it.
提示：全区块链节点验证一个交易的方法是检查这个交易所在区块之下的完整区块链，从而保证这个UTXO未被花费。而SPV节点检查这个区块的深度，即这个区块之上有一些区块。

To get the block headers, SPV nodes use a getheaders message instead of getblocks. The responding peer will send up to 2,000 block headers using a single headers message. The process is otherwise the same as that used by a full node to retrieve full blocks.
为了获得区块头，SPV节点使用getheaders消息，而不是getblocks消息。
peer发送一个headers 消息，包含最多2000个区块头。
这个过程与全节点获取完整区块的过程没什么区别。

SPV nodes also set a filter on the connection to peers, to filter the stream of future blocks and transactions sent by the peers. Any transactions of interest are retrieved using a

getdata request. The peer generates a tx message containing the transactions, in response. SPV node synchronizing the block headers shows the synchronization of block headers.
SPV节点还在连接上设置了一个过滤器，用以过滤peer以后发送的区块和交易。
使用getdata请求获取任何感兴趣的交易。peer使用tx消息包含这些交易。
图7显示了区块头的同步。

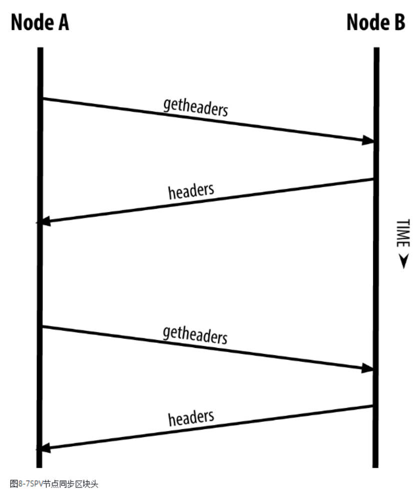


Figure 7. SPV node synchronizing the block headers
图7：同步区块头的SPV节点

Because SPV nodes need to retrieve specific transactions in order to selectively verify them, they also create a privacy risk. Unlike full blockchain nodes, which collect all transactions within each block, the SPV node’s requests for specific data can inadvertently reveal the addresses in their wallet. For example, a third party monitoring a network could keep track of all the transactions requested by a wallet on an SPV node and use those to associate bitcoin addresses with the user of that wallet, destroying the user’s privacy.
因为SPV节点需要获取特定的交易，从而选择性地验证它们，这样也造成了一个隐私风险。
全区块链节点收集每个区块中的所有交易，SPV节点只请求特定数据，这样，无意中就会泄露钱包中的地址。
例如，第三方可能监控网络，跟踪SPV节点上的钱包请求的所有交易，使用这些信息把比特币地址域钱包用户关联起来，从而破坏了用户的隐私。

Shortly after the introduction of SPV/lightweight nodes, bitcoin developers added a feature called *bloom filters* to address the privacy risks of SPV nodes. Bloom filters allow SPV nodes to receive a subset of the transactions without revealing precisely which addresses they are interested in, through a filtering mechanism that uses probabilities rather than fixed patterns.
在引入SPV/轻量级节点之后不久，比特币开发人员增加了一个功能（bloom过滤器），用来解决SPV节点的隐私风险。
bloom过滤器允许SPV节点获取交易的一个子集，而不会精确揭示SPV节点对哪些地址感兴趣；
它通过一个过滤机制，这个机制使用概率，而不是固定模式。

8.9 bloom过滤器

A bloom filter is a probabilistic search filter, a way to describe a desired pattern without specifying it exactly. Bloom filters offer an efficient way to express a search pattern while protecting privacy. They are used by SPV nodes to ask their peers for transactions matching a specific pattern, without revealing exactly which addresses, keys, or transactions they are searching for.
bloom过滤器是一个概率搜索过滤器，它描述了一个希望的模式，但没有精确地描述这个模式。
bloom过滤器提供了一个高效的方法，来表达一个搜索模式，同时保护了隐私。
SPV节点使用bloom过滤器向peer请求与特定模式匹配的交易，但没有完全暴露它在搜索的地址、密钥或交易。

In our previous analogy, a tourist without a map is asking for directions to a specific address, "23 Church St." If she asks strangers for directions to this street, she inadvertently reveals her destination. A bloom filter is like asking, "Are there any streets in this neighborhood whose name ends in R-C-H?" A question like that reveals slightly less about the desired destination than asking for "23 Church St." Using this technique, a tourist could specify the desired address in more detail such as "ending in U-R-C-H" or less detail as "ending in H." By varying the precision of the search, the tourist reveals more or less information, at the expense of getting more or less specific results. If she asks a less specific pattern, she gets a lot more possible addresses and better privacy, but many of the results are irrelevant. If she asks for a very specific pattern, she gets fewer results but loses privacy.
用前面的例子中，没有地图的游客需要询问去“23 Church St”的方向。
如果他想别人问到那条街的方向，他就泄露了他的目的地。
而bloom过滤器是这样问：附近是否有以“R–C–H”结尾的街道?
这样的问法就比前面的问法泄露的信息要少。
使用这个技术，游客可以提供有关目的地的更多信息（例如以“U–R–C–H”结尾的街道），或者提供更少的信息（例如以“H”开头的街道）。
通过改变搜索的精确度，游客透漏了更多或更少的信息，从而得到更精确或更不精确的结果。
如果他问的不精确，得到了更多的可能地址，也更好地保护了隐私，但许多结果是无关的。
如果他问的很精确，得到了更少的结果，但会暴露隐私。

Bloom filters serve this function by allowing an SPV node to specify a search pattern for transactions that can be tuned toward precision or privacy. A more specific bloom filter will produce accurate results, but at the expense of revealing what patterns the SPV node is interested in, thus revealing the addresses owned by the user’s wallet. A less specific bloom filter will produce more data about more transactions, many irrelevant to the node, but will allow the node to maintain better privacy.
bloom过滤器提供这个功能的方法是：允许SPV节点指定交易的搜索模式，这个模式可以基于准确性或私密的考虑进行调节。
一个更具体的bloom过滤器会产生更准确的结果，但也会泄露这个SPV节点感兴趣的模式，因此会泄露用户钱包所拥有的地址。
一个更不具体的bloom过滤器会产生更多的交易数据，对SPV节点来说很多数据是无关的，但SPV节点可以维持更好的隐私。

8.9.1 bloom过滤器工作原理

Bloom filters are implemented as a variable-size array of N binary digits (a bit field) and a variable number of M hash functions. The hash functions are designed to always produce an output that is between 1 and N, corresponding to the array of binary digits. The hash functions are generated deterministically, so that any node implementing a bloom filter will always use the same hash functions and get the same results for a specific input. By

choosing different length (N) bloom filters and a different number (M) of hash functions, the bloom filter can be tuned, varying the level of accuracy and therefore privacy.
bloom过滤器的实现是：N个比特的数组，M个哈希函数。
这些哈希函数的输出值总是在1~N之间，对应于这个比特数组。
这些哈希函数的产生是确定性的，所以，任何实现一个bloom过滤器的节点总是使用相同的哈希函数，并对特定输入得到相同的结果。
通过选择不同长度的bloom过滤器，以及不同数量的哈希函数，可以调整bloom过滤器，从而改变精确度和隐私。

In An example of a simplistic bloom filter, with a 16-bit field and three hash functions, we use a very small array of 16 bits and a set of three hash functions to demonstrate how bloom filters work.
在图8中，我们使用一个非常小的数组（16个比特），以及3个哈希函数，来演示bloom过滤器的工作原理。

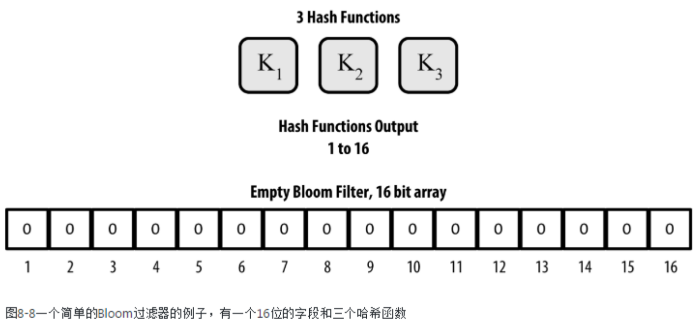


Figure 8. An example of a simplistic bloom filter, with a 16-bit field and three hash functions
图8：一个简单的bloom过滤器，有16个比特，3个哈希函数

The bloom filter is initialized so that the array of bits is all zeros. To add a pattern to the bloom filter, the pattern is hashed by each hash function in turn. Applying the first hash function to the input results in a number between 1 and N. The corresponding bit in the array (indexed from 1 to N) is found and set to 1, thereby recording the output of the hash function. Then, the next hash function is used to set another bit and so on. Once all M hash functions have been applied, the search pattern will be "recorded" in the bloom filter as M bits that have been changed from 0 to 1.
这个bloom过滤器被初始化，数组被初始化为零。
为了给这个bloom过滤器增加一个模式，依次用每个哈希函数对这个模式计算哈希。
使用第一个哈希函数会产生一个数字，范围是1~N。
把数组中对应的比特（索引为1~N）设置为1，这样，就记录了这个哈希函数的输出。
然后用第二个哈希函数来设置另一个比特，以此类推。
一旦M个哈希函数都使用了，搜索模式就被记录在这个过滤器中，就是比特数组的M个比特被设置为1。

Adding a pattern "A" to our simple bloom filter is an example of adding a pattern "A" to the simple bloom filter shown in An example of a simplistic bloom filter, with a 16-bit field and three hash functions.
图9是把模式“A”添加到这个简单的bloom过滤器中。

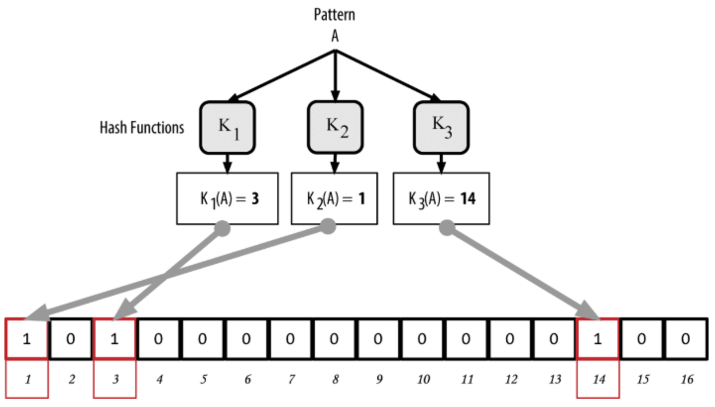


图8-9向简易Bloom过滤器添加关键词"A"

Figure 9. Adding a pattern "A" to our simple bloom filter

Adding a second pattern is as simple as repeating this process. The pattern is hashed by each hash function in turn and the result is recorded by setting the bits to 1. Note that as a bloom filter is filled with more patterns, a hash function result might coincide with a bit that is already set to 1, in which case the bit is not changed. In essence, as more patterns record on overlapping bits, the bloom filter starts to become saturated with more bits set to 1 and the accuracy of the filter decreases. This is why the filter is a probabilistic data structure—it gets less accurate as more patterns are added. The accuracy depends on the number of patterns added versus the size of the bit array (N) and number of hash functions (M). A larger bit array and more hash functions can record more patterns with higher accuracy. A smaller bit array or fewer hash functions will record fewer patterns and produce less accuracy.
增加一个模式就是重复这个过程。
注意，因为一个bloom过滤器可以填入多个模式，所以，一个哈希函数的结果位可能已经被设置为1。
大体上，随着模式的增加，bloom过滤器开始饱和，这个过滤器的精确性降低。
因此，这个过滤器是一个概率数据结构，它随着模式的增加会降低精确性。
精确性依赖于“模式的数量”与“N和M的大小”。
N和M越大，能更精确地记录更多的模式。
N和M越小，记录的模式更少，产生的精确性更低。

Adding a second pattern "B" to our simple bloom filter is an example of adding a second pattern "B" to the simple bloom filter.
图10是在这个简单的bloom过滤器上增加了模式“B”。

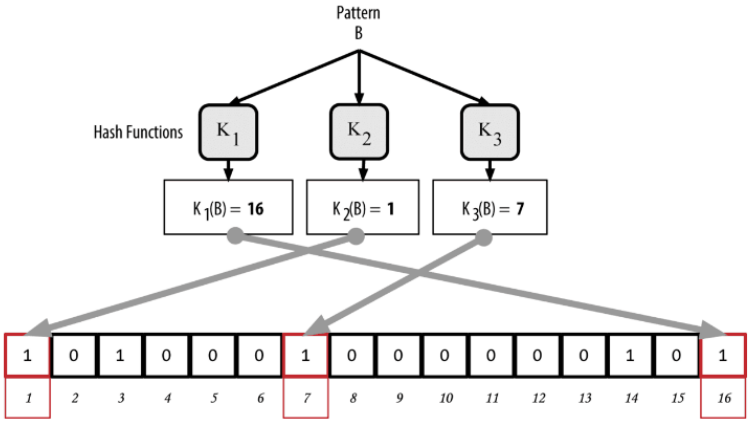


图8-10向该简易Bloom过滤器里增加第二个关键词"B"

Figure 10. Adding a second pattern "B" to our simple bloom filter
图10：在这个简单的bloom过滤器上增加了模式“B”

To test if a pattern is part of a bloom filter, the pattern is hashed by each hash function and the resulting bit pattern is tested against the bit array. If all the bits indexed by the hash functions are set to 1, then the pattern is *probably* recorded in the bloom filter.

Because the bits may be set because of overlap from multiple patterns, the answer is not certain, but is rather probabilistic. In simple terms, a bloom filter positive match is a "Maybe, Yes."
为了测试一个模式是否是一个过滤器的一部分，就用每个哈希函数计算这个模式，把产生的比特模式与比特数组对比。
如果所有比特都是1，表示这个模式可能被记录在这个bloom过滤器中。
这个回答不是确定的，但有相当的概率。
简单的说，bloom过滤器的正匹配的意思是“可能是”。

Testing the existence of pattern "X" in the bloom filter. The result is a probabilistic positive match, meaning "Maybe." is an example of testing the existence of pattern "X" in the simple bloom filter. The corresponding bits are set to 1, so the pattern is probably a match.
图11是一个例子，用这个简单的om过滤器测试模式“X”。
对应的比特都是1，所以这个模式可能是一个匹配。

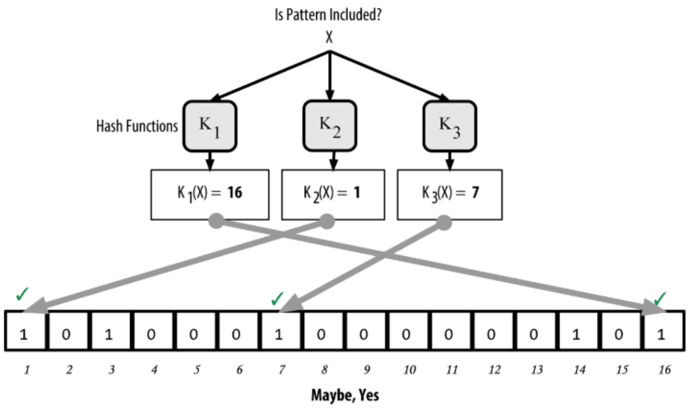


图8-11在bloom过滤器中测试模式"X"的存在。 结果是一个概率正匹配，意思是"也许"

Figure 11. Testing the existence of pattern "X" in the bloom filter. The result is a probabilistic positive match, meaning "Maybe."
图11：用这个bloom过滤器测试模式“X”，结果是概率正匹配，意味着“可能是”

On the contrary, if a pattern is tested against the bloom filter and any one of the bits is set to 0, this proves that the pattern was not recorded in the bloom filter. A negative result is not a probability, it is a certainty. In simple terms, a negative match on a bloom filter is a "Definitely Not!"
相反，如果测试结果中，有一个比特是0，就证明这个bloom过滤器没有记录这个模式。
这个结果不是概率，而是确定的。
简单的说，bloom过滤器的负匹配的意思是“肯定不是”

Testing the existence of pattern "Y" in the bloom filter. The result is a definitive negative match, meaning "Definitely Not!" is an example of testing the existence of pattern "Y" in the simple bloom filter. One of the corresponding bits is set to 0, so the pattern is definitely not a match.
图12是测试模式“Y”，其中一个比特是0，所以这个模式肯定不匹配。

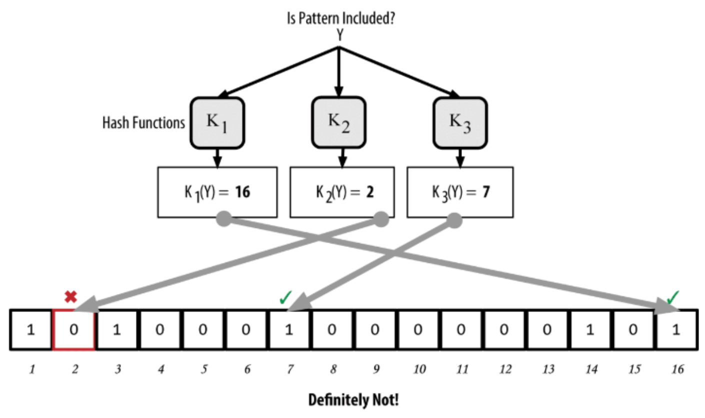


图8-12在bloom过滤器中测试模式"Y"的存在。 结果是一个明确的否定匹配，意思是"绝对不！"

Figure 12. Testing the existence of pattern "Y" in the bloom filter. The result is a definitive negative match, meaning "Definitely Not!"

图12：测试模式“Y”，结果是一个确定的负匹配，即“肯定不是”。

8.10 SPV节点如何使用bloom过滤器

Bloom filters are used to filter the transactions (and blocks containing them) that an SPV node receives from its peers, selecting only transactions of interest to the SPV node without revealing which addresses or keys it is interested in.
bloom过滤器用于过滤SPV节点从peer那里接收的交易（和包含交易的区块），只选择SPV节点感兴趣的交易，而不会泄露SPV节点感兴趣的地址或密钥。

An SPV node will initialize a bloom filter as "empty"; in that state the bloom filter will not match any patterns. The SPV node will then make a list of all the addresses, keys, and hashes that it is interested in. It will do this by extracting the public key hash and script hash and transaction IDs from any UTXO controlled by its wallet. The SPV node then adds each of these to the bloom filter, so that the bloom filter will "match" if these patterns are present in a transaction, without revealing the patterns themselves.
SPV节点把一个bloom过滤器初始化为“空”;在该状态下，bloom过滤器不匹配任何模式。
然后，SPV节点会制作一个列表，包含它感兴趣的所有地址、密钥和哈希。
实现方法是：提取公钥哈希，以及它的钱包控制的任何UTXO中的交易ID和脚本哈希。
然后，SPV节点把它们加入这个bloom过滤器，这样，如果一个交易中存在这些模式，这个bloom过滤器就会匹配，但不会泄露模式本身。

The SPV node will then send a filterload message to the peer, containing the bloom filter to use on the connection. On the peer, bloom filters are checked against each incoming transaction. The full node checks several parts of the transaction against the bloom filter, looking for a match including:

- The transaction ID
- The data components from the locking scripts of each of the transaction outputs (every key and hash in the script)
- Each of the transaction inputs
- Each of the input signature data components (or witness scripts)

然后，SPV节点向peer发送一个filterload消息，包含这个bloom过滤器，用在这个连接上。
在peer上，针对每个进入的交易检查bloom过滤器。
全节点针对这个bloom过滤器检查交易的多个部分，寻找一个匹配，包括：

- 交易ID
- 每个交易输出的锁定脚本的数据部分（脚本中的每个密钥和哈希）
- 每个交易输入
- 每个输入签名数据部分（或见证脚本）

By checking against all these components, bloom filters can be used to match public key hashes, scripts, OP_RETURN values, public keys in signatures, or any future component of a smart contract or complex script.
通过检查这些部分，bloom过滤器可用于匹配：公钥哈希、脚本、OP_RETURN值、签名中的公钥，或智能合约或复杂脚本中的任何未来部分。

After a filter is established, the peer will then test each transaction’s outputs against the bloom filter. Only transactions that match the filter are sent to the node.
在建立了一个过滤器之后，peer测试每个交易的输出。
只有与过滤器匹配的交易才会发送给SPV节点。

In response to a getdata message from the node, peers will send a merkleblock message that contains only block headers for blocks matching the filter and a merkle path (see [merkle_trees]) for each matching transaction. The peer will then also send tx messages containing the transactions matched by the filter.
为了响应getdata消息，peer会发送一个merkleblock消息，它只包含与过滤器匹配的区块的区块头，以及每个匹配交易的merkle路径。
然后，peer发送tx消息，包含与过滤器匹配的这个交易。

As the full node sends transactions to the SPV node, the SPV node discards any false positives and uses the correctly matched transactions to update its UTXO set and wallet balance. As it updates its own view of the UTXO set, it also modifies the bloom filter to match any future transactions referencing the UTXO it just found. The full node then uses the new bloom filter to match new transactions and the whole process repeats.
因为全节点给SPV节点发送交易，SPV节点丢弃任何误报，使用正确匹配的交易来更新它的UTXO集和钱包余额。
当它更新了自己的UTXO集视图，它还会修改这个bloom过滤器，以匹配任何引用刚发现的UTXO的交易。
然后，全节点使用新的bloom过滤器来匹配新的交易，整个过程不断重复。

The node setting the bloom filter can interactively add patterns to the filter by sending a filteradd message. To clear the bloom filter, the node can send a filterclear message. Because it is not possible to remove a pattern from a bloom filter, a node has to clear and resend a new bloom filter if a pattern is no longer desired.
设置bloom过滤器的节点可以交互地向过滤器增加模式，方法是发送filteradd消息。
为了清除这个bloom过滤器，节点可以发送filterclear消息。
因为不可能从bloom过滤器中删除一个模式，所以，如果希望重新设置模式，节点必须清除和重新发送一个新的bloom过滤器。

The network protocol and bloom filter mechanism for SPV nodes is defined in [BIP-37 (Peer Services)]().
BIP-37定义了SPV节点的网络协议和bloom过滤器机制。
BIP-37 Peer Services

8.10.1 ddk总结：SPV原理

SPV原理：（不能十分确定）
1. SPV节点接收最长区块链的所有区块头
2. SPV节点对于要验证的交易（即它关心的支付），在peer上建立一个bloom过滤器。
3. 当peer收到一个区块时，就根据bloom过滤器测试区块中的每个交易，然后发送一个merkleblock消息，包含区块头、感兴趣的交易、哈希路径。
4. SPV节点收到后，先确认这个交易是自己感兴趣的交易，然后使用merkle路径确认这个交易在这个区块中，最后确认这个区块在区块链中。
   如果交易所在的已被确认过6次，则表示交易是有效的。

参考：
- 《精通比特币》 8.8 SPV节点；9.8 Merkle树和SPV
- https://www.reddit.com/r/Bitcoin/comments/3uhi3a/detecting_bloom_filter_requests_from_spv_wallets/
- 《区块链 从数字货币到信用社会》 3.1 SPV

8.11 SPV节点和隐私

Nodes that implement SPV have weaker privacy than a full node. A full node receives all transactions and therefore reveals no information about whether it is using some address in its wallet. An SPV node receives a filtered list of transactions related to the addresses that are in its wallet. As a result, it reduces the privacy of the owner.
SPV节点的隐私比全节点要弱。
全节点接收所有交易，因此，不会泄露钱包中是否使用了某个地址。
SPV节点接收与地址相关的经过过滤的交易列表，因此，它降低了自身的隐私性。

Bloom filters are a way to reduce the loss of privacy. Without them, an SPV node would have to explicitly list the addresses it was interested in, creating a serious breach of privacy. However, even with bloom filters, an adversary monitoring the traffic of an SPV client or connected to it directly as a node in the P2P network can collect enough information over time to learn the addresses in the wallet of the SPV client.
bloom过滤器是减少隐私损失的一种方法。
如果没有bloom过滤器，SPV节点必须明确列出它感兴趣的地址，这会造成严重的隐私泄露。
但是，即使有了过滤器，监视SPV客户端流量的对手，或作为一个节点直接与SPV客户端连接，都可能收集到足够的信息，从而知道SPV客户端的钱包中的地址。

8.12加密的和认证的连接

Most new users of bitcoin assume that the network communications of a bitcoin node are encrypted. In fact, the original implementation of bitcoin communicates entirely in the clear. While this is not a major privacy concern for full nodes, it is a big problem for SPV nodes.
比特币的多数新用户认为：比特币节点的网络通信是加密的。
实际上，比特币的原始实现以完全透明的方式进行通信。
虽然对于全节点来说，这不是一个主要的隐私问题，但对SPV节点来说是一个大问题。

As a way to increase the privacy and security of the bitcoin P2P network, there are two solutions that provide encryption of the communications: *Tor Transport* and *P2P Authentication and Encryption* with BIP-150/151.
作为增加比特币P2P网络的隐私和安全性的一种方法，
有两个方案提供了通信的加密：

- BIP-150 Tor传输
- BIP-151 P2P认证和加密

8.12.1 Tor传输

Tor, which stands for *The Onion Routing network*, is a software project and network that offers encryption and encapsulation of data through randomized network paths that offer anonymity, untraceability and privacy.
Tor代表“洋葱路由网络”，它是一个软件项目和网络，通过随机网络路径提供数据的加密和封装，这提供了匿名性、不可追踪性、隐私性。

Bitcoin Core offers several configuration options that allow you to run a bitcoin node with its traffic transported over the Tor network. In addition, Bitcoin Core can also offer a Tor hidden service allowing other Tor nodes to connect to your node directly over Tor.
Bitcoin Core提供了几个配置选项，允许你让比特币节点的流量在Tor网络上传输。
此外，Bitcoin Core还能提供一个Tor隐藏服务，它允许其它Tor节点在Tor上直接连接你的节点。

As of Bitcoin Core version 0.12, a node will offer a hidden Tor service automatically if it is able to connect to a local Tor service. If you have Tor installed and the Bitcoin Core process runs as a user with adequate permissions to access the Tor authentication cookie, it should work automatically. Use the debug flag to turn on Bitcoin Core’s debugging for the Tor service like this:
从Bitcoin Core 0.12开始，如果节点能连接到一个本地Tor服务，这个节点就自动提供一个隐藏Tor服务。
如果你安装了Tor，Bitcoin Core进程作为一个用户运行，它有足够的权限访问这个Tor认证cookie，它应该能够自动工作。
使用debug标志为Tor服务打开Bitcoin Core的debuggin，如下：

```
$ bitcoind --daemon --debug=tor
```

You should see "tor: ADD_ONION successful" in the logs, indicating that Bitcoin Core has added a hidden service to the Tor network.
你应该在日志中看到：tor：ADD_ONION success
表示Bitcoin Core已经向这个Tor网络添加了一个隐藏服务。

You can find more instructions on running Bitcoin Core as a Tor hidden service in the Bitcoin Core documentation (*docs/tor.md*) and various online tutorials.
你可以在下列找到描述，把Bitcoin Core作为一个Tor隐藏服务来运行：

- Bitcoin Core文档：docs/tor.md

- 各种在线教程

8.12.2 P2P认证和加密

Two Bitcoin Improvement Proposals, BIP-150 and BIP-151, add support for P2P authentication and encryption in the bitcoin P2P network. These two BIPs define optional services that may be offered by compatible bitcoin nodes.
BIP-150和BIP-151为比特币P2P网络增加了如下支持：P2P认证和加密。
这两个BIP定义了可选服务，可由兼容的比特币节点提供这些服务。

BIP-151 enables negotiated encryption for all communications between two nodes that support BIP-151. BIP-150 offers optional peer authentication that allows nodes to authenticate each other’s identity using ECDSA and private keys. BIP-150 requires that prior to authentication the two nodes have established encrypted communications as per BIP-151.
BIP-151为两个节点之间的所有通信支持协商的加密。
BIP-150提供可选的peer认证，允许节点使用ECDSA和私钥来认证对方的身份。
BIP-150要求：在认证之前，两个节点按照BIP-151建立了加密通信。

As of January 2017, BIP-150 and BIP-151 are not implemented in Bitcoin Core. However, the two proposals have been implemented by at least one alternative bitcoin client named bcoin.
在2017年1月时，BIP-150和BIP-151还没在Bitcoin Core中实现。
但是，这两个建议已被一个比特币客户端（bcion）实现。

BIP-150 and BIP-151 allow users to run SPV clients that connect to a trusted full node, using encryption and authentication to protect the privacy of the SPV client.
BIP-150和BIP-151允许用户运行SPV节点，它连接到一个信任的全节点，使用加密和认证来保护SPV客户端的隐私。

Additionally, authentication can be used to create networks of trusted bitcoin nodes and prevent Man-in-the-Middle attacks. Finally, P2P encryption, if deployed broadly, would strengthen the resistance of bitcoin to traffic analysis and privacy-eroding surveillance, especially in totalitarian countries where internet use is heavily controlled and monitored.
此外，认证可用于创建信任比特币节点的网络，防止中间人攻击。
最后，如果广泛部署了P2P加密，能加强比特币对流量分析和隐私监视的阻力，特别是在互联网使用受到严格控制和监控的极权主义国家。

The standard is defined in BIP-150 (Peer Authentication) and BIP-151 (Peer-to-Peer Communication Encryption).
这个标准定义在：

- BIP-150 Peer Authentication
- BIP-151 Peer-to-Peer Communication Encryption

8.13交易池

Almost every node on the bitcoin network maintains a temporary list of unconfirmed transactions called the *memory pool*, *mempool*, or *transaction pool*. Nodes use this pool to keep track of transactions that are known to the network but are not yet included in the blockchain. For example, a wallet node will use the transaction pool to track incoming payments to the user's wallet that have been received on the network but are not yet confirmed.
比特币网络中几乎每个节点都会维护未确认交易的一个临时列表，称为“内存池”或“交易池”。
节点利用这个交易池来追踪交易，这些交易已经被网络知道，但还没有包含在区块链中。
例如，钱包节点会使用这个交易池跟踪给用户钱包的支付，它已经被网络收到，但还没有被确认。

As transactions are received and verified, they are added to the transaction pool and relayed to the neighboring nodes to propagate on the network.
当交易被收到和验证时，它们被添加到交易池，并传播给相邻节点，从而传播到网络中。

Some node implementations also maintain a separate pool of orphaned transactions. If a transaction's inputs refer to a transaction that is not yet known, such as a missing parent, the orphan transaction will be stored temporarily in the orphan pool until the parent transaction arrives.
有些节点实现还维护一个单独的孤儿交易池。
如果一个交易的输入引用的交易还不知道（例如一个缺失的父交易），则这个孤儿交易会临时存储在孤儿交易池中，直到收到它的父交易。

When a transaction is added to the transaction pool, the orphan pool is checked for any orphans that reference this transaction's outputs (its children). Any matching orphans are then validated. If valid, they are removed from the orphan pool and added to the transaction pool, completing the chain that started with the parent transaction.
当一个交易被添加到交易池中，会检查孤儿交易池，看看否有某个孤儿交易引用了这个交易的输出。
任何匹配的孤立交易都会被验证。如果有效，就从孤儿交易池中删除它，并添加到交易池中，从而使以其父交易开始的链变得完整。

In light of the newly added transaction, which is no longer an orphan, the process is repeated recursively looking for any further descendants, until no more descendants are found. Through this process, the arrival of a parent transaction triggers a cascade reconstruction of an entire chain of interdependent transactions by re-uniting the orphans with their parents all the way down the chain.
对新加入交易池的交易来说，它不再是孤儿交易，递归重复这个过程，查找更多的后代，直到没有发现更多的后代。
通过这个过程，父交易的到来会触发互相依赖的交易的完整链条的级联重构，方法是把孤儿交易和父交易一路连接起来。

Both the transaction pool and orphan pool (where implemented) are stored in local memory and are not saved on persistent storage; rather, they are dynamically populated from incoming network messages. When a node starts, both pools are empty and are gradually populated with new transactions received on the network.
交易池和孤儿池都是存储在本地内存中，不是存储在永久存储器上，它们是用进入的网络消息动态填充。
当节点启动时，这两个池都是空的，随着接收到新的交易，不断填充这两个池。

Some implementations of the bitcoin client also maintain an UTXO database or pool, which is the set of all unspent outputs on the blockchain. Although the name "UTXO pool" sounds similar to the transaction pool, it represents a different set of data. Unlike the transaction and orphan pools, the UTXO pool is not initialized empty but instead contains millions of entries of unspent transaction outputs, everything that is unspent from all the way back to the genesis block. The UTXO pool may be housed in local memory or as an indexed database table on persistent storage.
有些比特币客户端的实现还维护一个UTXO数据库（或UTXO池），它是区块链中所有未花费输出的集合。

虽然“UTXO 池”的名字听上去与交易池相似，但它表示一个不同的数据集。
UTXO池不是初始为空，而是包含了数百万的未花费交易输出，即所有未花费输出，一路回到创世区块。
UTXO池可以放在本地内存中，或作为一个索引数据库表放在永久存储器中。

Whereas the transaction and orphan pools represent a single node’s local perspective and might vary significantly from node to node depending upon when the node was started or restarted, the UTXO pool represents the emergent consensus of the network and therefore will vary little between nodes. Furthermore, the transaction and orphan pools only contain unconfirmed transactions, while the UTXO pool only contains confirmed outputs.
交易池和孤儿池表示一个节点的本地视图，不同节点可能区别很大，依赖于节点何时启动或重启。
UTXO池表示了网络的自发共识，因此，节点之间相差不大。
此外，交易池和孤儿池只包含未确认的交易，而UTXO池只只包含已确认的交易。