比特币网络

一、网络协议

比特币网络中的节点之间可以发送、接收消息进行通信，网络协议基于TCP协议，支持IPV4、IPV6、TOR网络。

可以通过设置参数”-onlynet"指明只支持哪种网络。

一、消息

网络通信以消息为单位，消息格式为：消息头 + 数据。

1、消息头

消息头的类为CMessageHeader，定义了消息头的格式。

消息头的格式：

格式	长度（字节）
消息开始字符串（MessageStart）	4
命令（Command）	12
消息大小（MessageSize）	无符号整型
校验和（Checksum）	无符号整型

1. 消息开始字符串

定义了消息标识，采用在正常数据中极少出现的字符，不用大写ASCII码字符，任意对齐，长度为4字节，在UTF-8中无效。

在不同的环境中，消息开始字符串不同。

环境类型	消息开始字符串
主类型（MAIN）	0xd9b4bef9
测试网络（TESTNET）	0x0709110b
回归测试（REGTEST）	0xdab6bffa

2. 命令

定义了通信中的各种命令，长度为12个字节的字符串，由0x20~0x7F之间的字符串构成，其他字符无效。

各种命令及其详解见后面的命令章节。

3. 消息大小

定义了消息的大小，不包含消息头的大小，是无符号整型，最大值是32M（0x02000000）。

4. 校验和

消息的校验和，不包含消息头的数据，是无符号整型。

把消息数据经过2次SHA256算法运算得到校验和。

计算过程如下：

```
template<typename T1>
inline uint256 Hash(const T1 pbegin, const T1 pend)
```

```
{
    static unsigned char pblank[1];
    uint256 hash1;
    SHA256((pbegin == pend ? pblank : (unsigned char*)&pbegin[0]), (pend - pbegin) *
sizeof(pbegin[0]), (unsigned char*)&hash1);
    uint256 hash2;
    SHA256((unsigned char*)&hash1, sizeof(hash1), (unsigned char*)&hash2);
    return hash2;
}
```

5. 有效性

    满足以下3点才是有效的消息：

    1. 消息开始字符串与环境参数定义的字符串相同。
    2. 命令字符串第一个字符为0时，其余字符必须全部为0；不为0时，必须由0x20~0x7F之间的字符串构成。
    3. 消息大小不能超过32M。

2、消息类型

协议定义了3种消息。

消息类型	意义
MSG_TX	交易信息
MSG_BLOCK	区块
MSG_FILTERED_BLOCK	过滤的区块

二、命令

在消息的基础上增加命令，可以完成各种通信。

1、命令

目前网络协议版本号为70002。

命令详解：

命令字符串	含义
addr	网络节点地址
alert	警告
block	区块
filteradd	添加过滤交易信息
filterclear	清理过滤器
filterload	加载过滤器
getaddr	获取地址
getblocks	获取区块
getdata	获取数据
getheaders	获取区块头
headers	区块头
inv	库存清单
mempool	内存池
merkleblock	merkle块
notfound	没有获取相匹配的数据
ping	判断网络是否连通
pong	ping回应
reject	拒绝
tx	交易信息
verack	版本信息回应
version	获取版本信息

疑惑：命令码为什么不定义成宏定义、16进制值。

1、地址信息（addr）

发送端用这个命令广播地址，接收端收到此命令后把接收到的地址添加到节点的地址管理器中。

发送、接收的地址数量最多1000个。

节点地址管理器中的地址数量超过1000个时，不再添加老版本的地址，即源节点的版本号小于CADDR_TIME_VERSION（31402）。

如果源节点是网络节点（节点的fNetworkNode判断），则每隔20分钟更新节点的IP地址管理器（CAddrMan）中地址信息（CAddrInfo）的时间（nTime）。

2、警告（alert）

发送端可以发送警告命令，接收端接收到警告命令后，处理警告命令，且传播下去。

警告消息是格式化的字符串，避免发送其他字符串造假警告消息。

警告消息字符串：

strprintf("CAlert(\n nVersion = %d\n nRelayUntil = %"PRId64"\n nExpiration = %"PRId64"\n nID = %d\n nCancel = %d\n setCancel = %s\n nMinVer = %d\n nMaxVer = %d\n setSubVer = %s\n nPriority = %d\n strComment = \"%s\"\n strStatusBar = \"%s\"\n)\n", nVersion, nRelayUntil, nExpiration, nID, nCancel, strSetCancel, nMinVer, nMaxVer, strSetSubVer, nPriority, strComment, strStatusBar);

警告消息是带签名的，要校验签名的有效性，不处理无效的签名。

处理时先取消上一个警告消息，再处理这个警告消息。

传播警告消息有2种方式：

- 接收到警告消息后，遍历节点数组，给每个节点发送警告消息。
- 接收到”version”命令时，发送警告消息给源节点。

3、区块（block）

当发送端发送获取数据命令时（”getdata"），接收端从硬盘读取区块，把类型为MSG_BLOCK的区块反馈回去。

接收端收到区块命令后，当节点不在导入、重建索引状态时，处理区块（ProcessBlock）。

如果是DoS攻击，则向源节点发送拒绝消息（reject）。

4、添加过滤交易信息（filteradd）

添加过滤交易信息到源节点的过滤器中（pfilter），过滤信息最大是520字节。

5、清理过滤器（filterclear）

清理源节点的海量过滤器（pfilter），删除旧的海量过滤器，新建海量过滤器（CBloomFilter），允许传播交易信息（fRelayTxes）。

6、加载过滤器（filterload）

加载接收的过滤交易信息到新的海量过滤器中（CBloomFilter），更新过滤器的满、空的状态，允许传播交易信息（fRelayTxes）。

过滤器最大是36000字节，哈希函数的数量最大是50个。

7、获取地址（getaddr）

主动获取地址时发送此命令，接收到此命令后把节点的地址管理器中的地址返回给发送端。

先清空源节点的发送地址数组（vAddrToSend）。再把节点的IP地址管理器（addrman）中的地址（CAddress）发送给源节点。

8、获取区块（getblocks）

区块同步时，发送此命令，发送时需要指定区块范围（PushGetBlocks）。

接收到此命令后，根据区块范围，获取相应的区块，反馈回去。

接收的数据中包含区块范围的开始区块的定位信息（CBlockLocator）、结束区块的索引，从开始区块的下一个区块开始。

每次最多获取500个区块信息。满500个时，记录获取的最后一个区块的索引值，保存到源节点的hashContinue中。

9、获取数据（getdata）

发送端请求获取数据时，发送此命令。接收端收到此命令，获取指定的区块范围，从硬盘上读取数据反馈回去（ProcessGetData）。

每次最多获取50000个。

如果源节点是网络节点（节点的fNetworkNode判断），则每隔20分钟更新节点的IP地址管理器（CAddrMan）中地址信息（CAddrInfo）的时间（nTime）。

10、获取区块头（getheaders）

需要获取区块头时，发送此命令，发送时需要制定区块范围。

接收到此命令后，获取指定的范围的区块的头，发送headers消息发送给源节点。

数据中包含区块范围的开始区块的定位信息（CBlockLocator）、结束区块的索引，如果开始区块为空，则只获取结束区块的头部信息。

最多获取2000个区块头部信息。

11、获取inventory（inv）

当需要获取inventory时，发送此命令，发送时，需要指定范围。

接收到此命令后，按指定范围获取inventory数据（PushGetBlocks）。

每次最多获取50000个。

如果源节点是网络节点（节点的fNetworkNode判断），则每隔20分钟更新节点的IP地址管理器（CAddrMan）中地址信息（CAddrInfo）的时间（nTime）。

12、区块头（headers）

当接收到获取区块头命令时，获取区块头信息，发送回去。

接收到此命令后，如何处理，当前代码中没有。（我没有获取最新的代码）

13、内存池（mempool）

需要获取内存池中的交易信息时，发送此命令。接收到此命令后，获取内存池中的交易信息，反馈回去。

交易信息是MSG_TX类型的。

发送的交易信息必须不在源节点的海量过滤器中。

每次最多发送50000个。

14、merkle块（merkleblock）

获取数据时（ProcessGetData），获取到了需要过滤的块，则向源节点返回此命令。

接收到此命令的处理代码没有。（我没有获取最新的代码）

15、未找到（notfound）

获取数据时（ProcessGetData），没有找到相匹配的数据，则向源节点返回此命令，仅仅SPV客户端接收此命令。

16、ping

当用户在RPC中请求ping命令（节点的fPingQueued为TRUE），或者距离上一次发送命令超过了30分钟，且发送消息队列为空，则发送ping命令。

接收到此命令后，当发送端的版本号大于BIP0031_VERSION，则给发送端反馈带随机值的pong命令。不处理版本号低的。

如果源节点是网络节点（节点的fNetworkNode判断），则每隔20分钟更新节点的IP地址管理器（CAddrMan）中地址信息（CAddrInfo）的时间（nTime）。

17、pong

接收到ping命令后，返回pong命令。

接收到pong命令后，更新节点的ping花费时间（nPingUsecTime），所花费的时间为当前时间与节点的ping开始时间（nPingUsecStart）的差。

不处理源节点不带随机数的pong命令，即nPingNonceSent为0。

当接收到的命令的随机值与节点保存的随机值（nPingNonceSent）相同时，才是有效的pong消息。ping异步时，这2个值不相同，返回的随机值为0时，此时只是完成ping，但不更新时间。不为0时，表示ping失败。

最多处理8个pong消息，后面的直接完成。

18、拒绝（reject）

当发生特殊情况时，发送此命令，表示拒绝服务。接收到此命令后，在调试时写日志，记录拒绝服务的原因。

特殊情况如下：

- 重复发送获取版本信息的命令（”version”）。
- 发送端的版本号小于最小版本号（MIN_PEER_PROTO_VERSION = 209）。
- 接收到DoS攻击。
- 处理消息时发生异常（ProcessMessage）。

发送拒绝命令时，带上参数，表示拒绝的原因。

拒绝原因	数值	含义
REJECT_MALFORMED	0x01	处理消息时发生异常（ProcessMessage）
REJECT_INVALID	0x10	块、交易信息无效
REJECT_OBSOLETE	0x11	块版本、发送端版本过期
REJECT_DUPLICATE	0x12	重复发送获取版本命令、交易信息
REJECT_NONSTANDARD	0x40	交易信息不标准
REJECT_DUST	0x41	无
REJECT_INSUFFICIENTFEE	0x42	交易费不足
REJECT_CHECKPOINT	0x43	与校验点有关的错误

19、交易信息（tx）

当获取数据时（ProcessGetData），数据中的交易信息以此命令发送出去。接收到此命令后，把接收到的交易信息添加到内存池中（AcceptToMemoryPool）。

如果不能添加到内存池中，则添加到孤块交易信息中，如果孤块数量超过了限制（MAX_ORPHAN_TRANSACTIONS），则记录日志。

添加到内存池后，遍历孤块交易信息（mapOrphanTransactionsByPrev），把孤块信息添加到内存池中。如果添加失败，且区块无效或者交易费太小，则记录日志。最后删除孤块交易信息（EraseOrphanTx）。

当添加到内存池时返回的状态是DoS攻击，则发送拒绝消息（reject）给发送端。

20、版本回应（verack）

接收到获取版本命令时，发送版本回应命令回去。接收到版本回应命令后，设置节点的接收版本。

设置接收版本号，节点的接收版本（nRecvVersion）、接收消息的报头流的版本号、接收消息数据流的版本号（nVersion）都要设置。

新版本号为回应的的版本号（nVersion）与PROTOCOL_VERSION（70002）的最小值。

21、获取版本信息（version）

节点初始化时（类CNode的构造函数），发送获取版本信息的命令。接收到此命令，回应版本信息，同时把所有警告也反馈回去。

发送端只能发送一次获取版本的命令，重复发送时，回应拒绝命令（reject）。

源节点的版本号不能小于MIN_PEER_PROTO_VERSION（209），否则也回应拒绝消息，且断开连接。

连接自己时，将断开连接。接收信息中的随机数与本地主机的随机数（nLocalHostNonce）相同时，就是在连接自己。

发送版本回应命令，新版本为发送端的版本号与PROTOCOL_VERSION（70002）中的最小值。

发送命令后，把源节点的地址信息添加到节点的地址管理器中。

最后设置源节点成功连接标记（fSuccessfullyConnected），更新源节点地址时间。

如果源节点是网络节点（节点的fNetworkNode判断），则每隔20分钟更新节点的IP地址管理器（CAddrMan）中地址信息（CAddrInfo）的时间（nTime）。

二、网络节点

比特币网络是纯P2P体系，每一个节点既是服务器，也是客户端。节点既要监听网络，又要连接其他节点；既要接收消息，又要发送消息。

节点允许的最大连接数为125个，可以通过参数”-maxconnections"修改。

系统定义了信号量（CSemaphore），用于控制网络连接时的最大数量，信号量的最大值为连接数的最大值。

一、节点

运行了比特币软件的计算机就是一个节点，维系着比特币网络。

1、节点类

节点的信息定义在类CNode中，包含了节点需要的socket、消息、地址、ping数据等。

系统定义了节点数组（vector<CNode*> vNodes），包含了连接的所有节点。当节点连接上，则把此节点添加到节点数组中；断开连接后，从节点数组中移除此节点。

类CNode的定义：

```
class CNode
{
public:
    // socket
    uint64_t nServices;
    SOCKET hSocket;
    CDataStream ssSend;
    size_t nSendSize; // total size of all vSendMsg entries
    size_t nSendOffset; // offset inside the first vSendMsg already sent
    uint64_t nSendBytes;
    std::deque<CSerializeData> vSendMsg;
    CCriticalSection cs_vSend;

    std::deque<CInv> vRecvGetData;
    std::deque<CNetMessage> vRecvMsg;
    CCriticalSection cs_vRecvMsg;
    uint64_t nRecvBytes;
    int nRecvVersion;

    int64_t nLastSend;
    int64_t nLastRecv;
    int64_t nLastSendEmpty;
    int64_t nTimeConnected;
    CAddress addr;
    std::string addrName;
```

```
    CService addrLocal;
    int nVersion;
    // strSubVer is whatever byte array we read from the wire. However, this field is
intended
    // to be printed out, displayed to humans in various forms and so on. So we
sanitize it and
    // store the sanitized version in cleanSubVer. The original should be used when
dealing with
    // the network or wire types and the cleaned string used when displayed or
logged.
    std::string strSubVer, cleanSubVer;
    bool fOneShot;
    bool fClient;
    bool fInbound;
    bool fNetworkNode;
    bool fSuccessfullyConnected;
    bool fDisconnect;
    // We use fRelayTxes for two purposes -
    // a) it allows us to not relay tx invs before receiving the peer's version message
    // b) the peer may tell us in their version message that we should not relay tx invs
    //    until they have initialized their bloom filter.
    bool fRelayTxes;
    CSemaphoreGrant grantOutbound;
    CCriticalSection cs_filter;
    CBloomFilter* pfilter;
    int nRefCount;
    NodeId id;
protected:

    // Denial-of-service detection/prevention
    // Key is IP address, value is banned-until-time
    static std::map<CNetAddr, int64_t> setBanned;
    static CCriticalSection cs_setBanned;

public:
    uint256 hashContinue;
    CBlockIndex* pindexLastGetBlocksBegin;
    uint256 hashLastGetBlocksEnd;
```

```
    int nStartingHeight;
    bool fStartSync;

    // flood relay
    std::vector<CAddress> vAddrToSend;
    std::set<CAddress> setAddrKnown;
    bool fGetAddr;
    std::set<uint256> setKnown;

    // inventory based relay
    mruset<CInv> setInventoryKnown;
    std::vector<CInv> vInventoryToSend;
    CCriticalSection cs_inventory;
    std::multimap<int64_t, CInv> mapAskFor;

    // Ping time measurement
    uint64_t nPingNonceSent;
    int64_t nPingUsecStart;
    int64_t nPingUsecTime;
    bool fPingQueued;

private:
    // Network usage totals
    static CCriticalSection cs_totalBytesRecv;
    static CCriticalSection cs_totalBytesSent;
    static uint64_t nTotalBytesRecv;
    static uint64_t nTotalBytesSent;
}
```

hSocket是连接的socket句柄。

vSendMsg是发送消息数组。

vRecvMsg是接收的消息数组。

addr是节点地址信息。

pfilter是海量过滤器。

id是节点ID。

其余部分就不再解释。

类CNode包含了几个关于消息、Inventory等的函数。

函数ReceiveMsgBytes用来解析接收到的消息数据。
函数PushMessage用来发送消息。
函数PushVersion用来发送版本信息。
函数PushInventory用来发送Inventory。
函数PushAddress用来发送地址。
函数SetRecvVersion用来设置接收版本。
其余函数不再介绍。

2、构造发送消息

发送消息时，先在数据流中（CDataStream）中构造完整的消息，然后插入到消息队列中（CSerializeData），最后把消息逐个发送出去（SocketSendData）。

CSerializeData的定义为：

```
typedef std::vector<char, zero_after_free_allocator<char> > CSerializeData;
```

在数据流中（CDataStream）中构造消息时（PushMessage），先存放带有命令字符串的消息头（BeginMessage），再存放消息数据，一次最多填充9个消息数据，最后填充消息头中的大小、校验和（EndMessage）。

3、解析接收消息

在线程ThreadSocketHandler中接收其他节点发送过来的数据，把这些数据解析成单个消息（ReceiveMsgBytes），添加到节点的消息队列中，在线程ThreadMessageHandler中处理消息。

消息类CNetMessage的定义：

```
class CNetMessage {
public:
    bool in_data;                   // parsing header (false) or data (true)

    CDataStream hdrbuf;             // partially received header
    CMessageHeader hdr;             // complete header
    unsigned int nHdrPos;

    CDataStream vRecv;              // received message data
    unsigned int nDataPos;
};
```

先解析消息头（readHeader），再解析数据（readData）。

先把接收到的数据的开始部分复制到消息头数据流中（hdrbuf），再反格式化成消息头（hdr）。

消息的数据部分复制到消息数据流中（vRecv）。

消息数据最大为MAX_SIZE（0x02000000）。

4、消息缓冲区

接收消息的缓冲区默认是5*1000*1000 = 5000000 ≈ 4.7M。可以通过参数”-maxreceivebuffer”修改。可以通过函数ReceiveFloodSize()获取。

发送消息的缓冲区默认是1*1000*1000 = 1000000 ≈ 976K。可以通过参数”-maxsendbuffer”修改。可以通过函数SendBufferSize()获取。

疑惑：缓冲区大小为什么不定义成1024字节的整数倍?

二、节点线程

每个节点创建8个线程，用于地址、UPNP、监听、消息等，节点开始时创建这些线程（StartNode）。

1、获取主机IP

系统定义了几种本地地址类型，优先采用值大的类型（GetLocal函数）。

```
enum
{
    LOCAL_NONE,     // unknown
    LOCAL_IF,       // address a local interface listens on
    LOCAL_BIND,     // address explicit bound to
    LOCAL_UPNP,     // address reported by UPnP
    LOCAL_HTTP,     // address reported by whatismyip.com and similar
    LOCAL_MANUAL,   // address explicitly specified (-externalip=)

    LOCAL_MAX
};
```

节点启用时，获取主机名称、IP地址，添加到地址、服务对应的数组中（mapLocalHost）。获取本地地址，保存到节点的本地地址（addrLocal），且向节点网络广播地址（AdvertizeLocal）。这样获取的地址用于监听（LOCAL_IF）。

当系统支持IPv4时，则创建此线程，利用第三方获取主机名称、IP地址等。当只支持IPv6时（-onlynet=“IPv6”），则不用创建此线程。这样获取的地址类型为LOCAK_HTTP。

线程名为”bitcoin-ext-ip"。主函数是ThreadGetMyExternalIP。

对支持查询DNS、IP地址的网站发送HTTP请求，从返回的数据中解析IP地址。先用网站IP地址查询，查询失败，再用网站域名查询（GetMyExternalIP）。

BITCOIN中查询2个网站。

网址	IP地址	发送请求	解析关键字
checkip.dyndns.org	91.198.22.70	"GET / HTTP/1.1\r\n” "Host: checkip.dyndns.org\r\n” "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)\r\n” "Connection: close\r\n” "\r\n"	"Address:"
www.showmyip.com	74.208.43.192	"GET /simple/ HTTP/1.1\r\n” "Host: www.showmyip.com\r\n” "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)\r\n” "Connection: close\r\n” "\r\n"	

2、加载DNS中的节点地址

当系统启用了从DNS中加载地址时，则创建此线程。默认是启用，但可以通过设置参数”-dnsseed"来禁用。线程名为”bitcoin-dnsseed"。线程主函数是ThreadDNSAddressSeed。

在系统参数中保存了5个地址信息，保存在数组中（CDNSSeedData）。

名称	主机
bitcoin.sipa.be	seed.bitcoin.sipa.be
bluematt.me	dnsseed.bluematt.me
dashjr.org	dnsseed.bitcoin.dashjr.org
bitcoinstats.com	seed.bitcoinstats.com
xf2.org	bitseed.xf2.org

获取DNS节点数组中地址信息，查找主机的IP，构造成CAddress，添加到地址管理器中。

地址的时间设置为过去3～7天的随机值。

3、UPnP映射端口

当系统启用UPnP做端口映射时，创建此线程。默认是启用UPnP，但可以通过修改参数”-upnp”禁用UPnP。

线程名为”bitcoin-upnp”。

系统使用了开源的miniupnpc库，支持1.5、1.6版本。

首先要获取监听端口，系统定义了默认端口，在不同的网络中，默认端口不同，详细列表如下：

网络类型	端口号
主网络	8333
测试网络	18333
回归测试	18444

监听端口可以通过设置参数”-port”来设置。

再获取局域网内的所有UPnP设备（upnpDiscover），然后获取UPnP列表内的有效IGD设备（UPNP_GetValidIGD），再获取设备的外网IP（UPNP_GetExternalIPAddress），保存到节点中，且广播出去，最后基于TCP协议映射IP地址、端口、设备等。每隔20分钟刷新映射。

如果出现异常，则删除端口映射（UPNP_DeletePortMapping），释放UPnP设备列表（freeUPNPDevlist），释放UPnP的URL资源（FreeUPNPUrls）。

4、监听连接

此线程主要是监听网络，连接节点，接收、发送消息。线程名为”bitcoin-net"。主函数是ThreadSocketHandler。

此线程为无限循环，每隔10毫秒循环一次。

遍历节点数组，当节点标识断开连接（fDisconnect），或者没有任何引用、发送接收消息，则移除节点，关闭socket，添加到断开连接节点数组（vNodesDisconnected）。

遍历断开连接节点数组，当节点没有引用，且能获取节点的相关锁，则移除节点，删除此节点。

重新设置节点数量。

遍历监听SOCKET数组（vhListenSocket）、节点数组，为每个节点的socket设置发送、接收fd_set。

调用select函数监听socket。

遍历监听socket数组，当接收到数据，且socket有效时，接受连接，新建节点，添加到节点数组。

遍历节点数组，增加节点的引用次数。

遍历节点数组，接收网络数据，解析成消息，添加到节点的接收消息数组（vRecvMsg）。当发送集合有数据时，把节点的发送消息（vSendMsg）发送出去。更新节点的上一次发送全部消息的时间（nLastSendEmpty），最后检查不活跃的socke连接，节点需要断开连接（fDisconnect）。

当节点连接上超过1分钟，满足以下几点是不活跃的socke连接。

- 没有任何发送、接收消息（nLastSend、nLastRecv）。
- 距离上一次发送消息超过了90分钟（nLastSend），且距离上一次把全部消息发送出去也超过了90分钟（nLastSendEmpty）。
- 距离上一次接收消息超过了90分钟（nLastRecv）。

5、主动连接外部节点（-addnode）

此线程用于连接用户在RPC中在参数’-addnode’中指定的外部节点，线程名为”bitcoin-addcon"。主函数是ThreadOpenAddedConnections。

此线程也是无限循环，每隔2分钟连接一次。

把参数”-addnode”中的节点字符串添加到节点数组（vAddedNodes）。

当系统有代理域名时（nameproxyInfo.second），连接节点数组中的地址字符串（OpenNetworkConnection）。

没有代理服务器时，获取添加节点数组中的地址的IP地址，且移除节点数组（vNodes）中已经存在的地址，连接数组中的地址。

每隔500毫秒连接一个地址数组中的地址。。

不连接本地地址、已存在的节点地址。

连接时，如果没有代理服务器，则直接连接地址（ConnectSocket）；否则先连接代理服务器（ConnectSocketDirectly），再做socket握手，支持socket版本（Socks4、Socks5）。

连接上节点后，设置连接socket为非阻塞的。然后创建节点，添加到节点数组。

6、主动连接外部节点

此线程用于连接指定地址，线程名为”bitcoin-opencon"。主函数是ThreadOpenConnections。

此线程也是无限循环，每隔500毫秒连接一次。

当指定了参数”-connect”时，连接地址字符串。

从地址管理器中选择一个地址，连接此地址（OpenNetworkConnection）。

连接地址时注意以下几点：

- 不连接无效的、地址数组中的外部连接地址中已有的、本地的地址。
- 最多尝试连接100个地址。
- 不连接受限制的地址。
- 尝试连接次数达到30次时，才尝试连接距离上一次尝试连接10分钟以内的地址。
- 尝试连接次数达到50次时，才尝试连接地址端口不是参数默认端口的地址。

不同的网络类型，默认端口不同。对照表如下：

网络类型	端口号
主网络	8333
测试网络	18333
回归测试	18444

7、发送、接收消息

此线程主要用于处理节点的发送、接收到的消息，线程名为”bitcoin-msghand”。主函数是ThreadMessageHandler。

处理接收消息线程的优先级为THREAD_PRIORITY_BELOW_NORMAL。

此线程为无限循环，每隔0.1秒循环一次。

遍历节点数组，增加节点引用次数，检查节点是否需要同步。遍历节点数组，处理接收的消息，发送消息。遍历节点数组，减少节点引用次数。

当发送消息队列不满时，接收获取数据时，或者接收到消息，且第一个消息时完整的，则不用等待0.1秒。

1、处理接收消息（ProcessMessages）

节点接收消息后，循环遍历节点的消息缓冲区，解析消息头、数据，校验消息的有效性，再处理消息数据，处理过程中如果发生异常，则节点发送拒绝消息，此节点停止发送、接收消息。

消息的有效性检查主要检查以下4项：

1. 接收到的消息是否完整。

   类CNetMessage的complete()函数校验完整性，in_data为TRUE，且消息大小与消息数据偏移相等，则消息是完整的。
2. 消息头开始字符串是否与环境参数中的字符串相同。
3. 消息头是否有效。
4. 消息头校验和是否正确。

   重新计算数据的校验和，检验与消息头中的校验和是否相等。

2、发送消息（SendMessages）

首先检验节点的版本信息，如果版本信息为0，则不能发送消息。节点的版本信息保存在节点（CNode）的nVersion中。

发送消息时，构造需要的消息，插入到节点的发送消息队列中，然后发送出去。如果仅有1个消息，则立即发送；否则在ThreadSocketHandler线程中每隔0.01秒发送全部消息。

发送消息时，检验5个命令是否需要发送，分别是：ping、addr、getblocks、inv、getdata。

当用户在RPC中请求ping命令（节点的fPingQueued为TRUE），或者距离上一次发送命令超过了30分钟，且发送消息队列为空，则发送ping命令。当节点的版本大于BIP0031_VERSION时，设置发送ping命令的开始时间，发送ping命令时带随机数，随机数保存在节点的nPingNonceSent中。低于此版本时，不设置开始时间，不带随机数。

当钱包区块没有同步，且距离上一次广播地址已经有24小时，则广播地址，即把地址添加到全部节点的发送地址数组中（vAddrToSend）。节点必须不在监听状态。地址是节点的本地地址，且地址必须是有效路由的。遍历节点时，如果曾经广播过地址，则清空已知地址集（setAddrKnown）。

随机选择一个节点，发送addr消息发送节点的发送地址数组中的地址，同时把地址插入到已知地址集中，每次最多发送1000个地址。发送完毕后，清空节点的发送地址数组。

当节点需要同步时，且不处于导入、重建索引，则发送getblocks消息获取区块。获取区块时，指定开始区块、结束区块。同时保存开始的区块索引到节点的pindexLastGetBlocksBegin中，保存结束区块索引值到节点hashLastGetBlocksEnd

中。当获取区块时，检查这2项，如果已经获取了最新的区块，则不再获取区块，避免重复发送getblocks消息。

当旧的钱包交易信息确认了，但尚未加入到区块中，且节点区块不处于导入、重建索引、同步区块，则重新广播钱包交易信息，发送inv消息。

遍历节点的发送Inventory数组，发送inv消息。发送的Inventory必须时尚未处理的，即不在节点已知Inventory集中（setInventoryKnown），发送后添加到已知Inventory集中。发送inv消息时最多发送1000个inventory。如果是随机节点，且inventory的类型时MSG_TX，则只发送1/4的inventory，先计算随机值，随机值的低2bit位为0的inventory发送出去，不为0的inventory保留在节点的发送inventory数组（vInventoryToSend）中。

如果节点存在比当前时间早的延迟的inventory，则发送getdata消息。延迟的inventory保存在节点的mapAskFor数组。发送的inventory必须是节点中没有的。每次发送getdata消息时最多发送1000个inventory。同时删除节点的mapAskFor中的项。

8、保存网路地址

此线程用于把网络地址保存到文件中，线程名为”bitcoin-dumpaddr"。主函数是DumpAddresses。

先把地址数据库（CAddrDB）中的地址添加到地址管理器中（CAddrMan），再把地址保存到文件中，文件名为”peers.dat"。

保存地址时，先写入参数中的消息开始字符串（Params().MessageStart()），再写入地址，最后写入校验和，校验和是地址数据的哈希值。

三、网络初始化、关闭

1、初始化

在比特币客户端初始化（AppInit2）过程中，第6步进行网络初始化，第11步启用节点。

1、网络初始化

网络初始化，注册消息处理信号（RegisterNodeSignals），绑定消息处理函数，信号、处理函数如下：

信号	信号	处理函数
获取高度	GetHeight	GetHeight
处理接收消息	ProcessMessages	ProcessMessages
发送消息	SendMessages	SendMessages
初始化节点	InitializeNode	InitializeNode
结束节点	FinalizeNode	FinalizeNode

获取参数”-socks”的值，默认是5，系统只支持sock 4、5版本。 检验sock版本是否有效。

如果指定了参数”-onlynet”，则设置限制指定的网络类型。如果系统支持IPv6，但禁用了IPv6，则设置限制IPv6。

获取参数”-proxy”的值，设置指定的代理服务器。默认是设置IPv4的代理服务器，但如果sock版本为5，则也设置IPv6的代理服务器。

如果指定了参数”-tor”，则设置NET_TOR网络的代理服务器。

绑定监听端口。

如果指定了参数”-externalip”，则添加扩展IP地址。

如果指定了参数”-seednode”，则保存种子节点，以备后续连接。

2、启用节点

启用节点（StartNode），先创建信号量，再初始化以”127.0.0.1”为地址的节点，然后搜索地址，创建7个线程，使用UPnP映射端口。

2、关闭

系统关闭时（Shutdown），停止节点（StopNode），反注册消息处理信号（UnregisterNodeSignals）。

停止节点时，先停止端口映射线程（MapPort），再把网络地址保存到文件中（DumpAddresses）。

反注册消息处理信号时，断开发送、接收消息等信号处理函数。