# 跟踪CAsyncSocketEx类族

## 类图：



## 二、各类关系及主要工作：

CEncryptStreamSocket主要负责数据的加密、解密，加密过程的协商等事务，CEMSocket主要负责EDonkey协议数据的发送与接收，而CServerSocket主要负责接收到的数据的处理，以及拟发送数据的封包

## 三、服务器连接过程

### (一)、程序启动后先进行一些，程序初始化工作：

#### 1、初始化本机IP

CeMuleAPP::InitInstance()🡪CServerConnect::CServerConnect()🡪InLocalIP(),

m\_nLocalIP=192.168.1.115

#### 2、初始化服务器列表

CServerListCtrl::OnNmCustomDraw()根据服务器列表反复调用🡪CServerConnect::GetCurrentServer()，

#### 3、启动监听窗口，等待其它客户端连接请求，步骤1.3和1.4与连接服务器无关

eMuleDlg::StartupTimer()🡪CListenSocket::StartListen()

3.1、新建Listen SOCKET

🡪CAsyncSocketEx::Create (UINT nSocketPort /\*=0\*/, int nSocketType /\*=SOCK\_STREAM\*/,long lEvent /\*=FD\_READ | FD\_WRITE | FD\_OOB | FD\_ACCEPT | FD\_CONNECT | FD\_CLOSE\*/, LPCSTR lpszSocketAddress /\*=NULL\*/, BOOL bReuseAddr /\*=FALSE\*/ )

3.1.1、🡪CAsyncSocketEx::InitAsyncSocketExInstance() { m\_spAsyncSocketExThreadDataList==NULL,执行else分支},

3.1.2、将Socket添加到Helper窗口

🡪CAsyncSocketEx::AttachHandle(SOCKET hSocket)🡪 CAsyncSocketExHelperWindow::AddSocket(CAsyncSocketEx \*pSocket, int &nSocketIndex){ m\_nWindowDataSize=512, nSocketIndex=-1, m\_nSocketCount=0, m\_nWindowDataPos=0, m\_pAsyncSocketExWindowData[i % m\_nWindowDataSize].m\_pSocket == NULL,执行if语句块一次，为各变量赋值，nSocketIndex=0, m\_nSocketCount=1, m\_nWindowDataPos=1, m\_pAsyncSocketExWindowData[i % m\_nWindowDataSize].m\_pSocket = 0x0494b3a8 {CListenSocket}, return true;}

3.1.3、选择监听事件

🡪CAsyncSocketEx::AsyncSelect(long lEvent /\*= FD\_READ | FD\_WRITE | FD\_OOB | FD\_ACCEPT | FD\_CONNECT | FD\_CLOSE\*/=8(FD\_ACCEPT)){ m\_lEvent=8(FD\_ACCEPT)，WSAAsyncSelect(m\_SocketData.hSocket, GetHelperWindowHandle(), m\_SocketData.nSocketIndex + WM\_SOCKETEX\_NOTIFY, lEvent)，return true}

3.1.4、 绑定IP

🡪bReuseAddr=0🡪 CAsyncSocketEx::Bind(UINT nSocketPort=55358, LPCSTR lpszSocketAddress=NULL){ sockAddr.sin\_addr.saddr=0.0.0.0}

(2)开始监听

🡪CAsyncSocketEx::Listen(int nConnectionBacklog=5)

#### 4、定时器保证连接处于活跃状态

CUploadQueue::UploadTimer()🡪CServerConnect::KeepConnectionAlive(){if(false)},定时调用，保持连接

### (二)、服务器连接开始

通过点击“连接”按钮，或根据选项设置，程序自动启动连接，首先读取服务器列表中的服务器 ，停止当前正在进行的连接，开始连接尝试

CemuleDlg::OnBnClickedConnect()🡪StartConnect()🡪 CServerConnect:: ConnectToAnyServer（UINT startAt=0, bool prioSort=true, bool isAuto=true, bool bNoCrypt=false）🡪StopConnectionTry(){m\_idRetryTimer=0,for不执行}

🡪Disconnect()(if(false),return false；)

🡪TryAnotherConnectionRequest(){next\_server=first Sever in list}

🡪ConnectToServer(CServer\* server=next\_server, bool multiconnect=true, bool bNoCrypt=true)

#### 1、新建ServerSocket

🡪CAsyncSocketEx::Create(UINT nSocketPort=0, int nSocketType=1/\*=SOCK\_STREAM\*/,

long lEvent=51 /\*=FD\_READ | FD\_WRITE | FD\_CONNECT | FD\_CLOSE\*/,

LPCSTR lpszSocketAddress=NULL /\*=NULL\*/, BOOL bReuseAddr=0 /\*=FALSE\*/ )

1.1🡪 CAsyncSocketEx::InitAsyncSocketExInstance(){ if(m\_spAsyncSocketExThreadDataList)==true,执行if分支,if(!pList)==false}

1.2同（一）3.1.2 m\_nWindowDataPos=2

1.3同（一）3.1.3 lEvent=51

1.4 同（一）3.1.4 nSocketPort=0

#### 2、向服务器发出连接请求

🡪 CServerConnect::ConnectTo(CServer\* server, bool bNoCrypt)

根据bNoCrypt是否为true，分别发起加密连接或未加密连接。当Enable protocol obfuscation未选中时，IsServerCryptLayerTCPRequested()返回false，m\_bTryObfuscated为false。

##### 2.1、未加密连接

###### 2.1.1发起连接

🡪 ConnectTo(CServer\* server=0x04652650, bool bNoCrypt=true)

###### 2.1.2 设置连接加密状态为m\_StreamCryptState = ECS\_NONE

🡪 CEncryptedStreamSocket::SetConnectionEncryption(bool bEnabled=false, const uchar\* pTargetClientHash=NULL, bool bServerConnection=true){m\_streamCryptState=ECS\_UNKNOWN,执行最后的else块，置**m\_StreamCryptState = ECS\_NONE;**}

###### 2.1.3 设置连接状态为CS\_CONNECTING

🡪SetConnectionState(**CS\_CONNECTING=1**){if,else if均为false,直接返回};

###### 2.1.4 调用socket类发起connect

🡪CEMSocket::Connect(LPCSTR lpszHostAddress=0x05115568, UINT nHostPort=5041)🡪 InitProxySupport(){直接返回}🡪 CAsyncSocketEx::Connect(LPCSTR lpszHostAddress=0x05115568, UINT nHostPort=5041)🡪 connect(m\_SocketData.hSocket=1136, lpSockAddr, nSockAddrLen=16)

###### 2.1.5、处理服务器返回连接信息

如果服务器回应了连接请求，HelperWinodw的FD\_CONNECT事件触发，CServerSocket::OnConnect()被调用，根据OnConnect中返回的nErrorCode，如果成功，设置连接状态为等待登录（CS\_WAITFORLOGIN）,调用ConnectionEstablished，如果失败，设置连接状态为CS\_SERVERDEAD或CS——SERVERFATAL，调用ConnectionFailed

(1)、连接成功

①、设置连接状态为等待登录（**CS\_WAITFORLOGIN**）, ConnectionEstablished函数被调用，准备登录信息，调用SendPacket向服务器发送登录信息包，信息包存入待发送队列等待发送, 由UploadBandwidthThrottler根据带宽情况进行择机发送。：

🡪CAsyncSocketExHelperWindow::WindowsProc(message=1284>=WM\_SOCKETEX\_NOTIFY){ hSocket=wParam=1136,nEvent=16(FD\_CONNECT)}🡪CServerSocket::OnConnect(nErrorCode=0)🡪 SetConnectionState(CS\_WAITFORLOGIN)

//SetConnectionState()调用CServerConnect::ConnectionEstablished()，处理CS\_WAITFORLOGIN分支：

🡪 CServerConnect::ConnectionEstablished(CServerSocket\* sender=0x050c5d10)🡪InitLocalIP(){m\_nLocalIP=192.168.1.115}

🡪if(sender->GetConnectionState() == CS\_WAITFORLOGIN),SendPacket(Packet\* packet,bool delpacket= true, CServerSocket\* sender)🡪 CServerSocket::SendPacket(Packet\* packet, bool delpacket=true, bool controlpacket=true, uint32 actualPayloadSize=0, bool bForceImmediateSend=false)🡪

CEMSocket:: SendPacket(Packet\* packet, bool delpacket=true, bool controlpacket=true, uint32 actualPayloadSize=0, bool bForceImmediateSend=false){if(controlpacket){ controlpacket\_queue.AddTail(packet); // queue up for controlpacket

theApp.uploadBandwidthThrottler->QueueForSendingControlPacket(this, HasSent());},if(bForceImmediateSend)==false}



②、可以发送，UploadBandwidthThrottler发送信息包，触发OnSend()

因为是未加密连接，CEncryptedStreamSocket::OnSend()未做出相应动作，没有启动加密连接协商。这边需要理清一下，为什么先发一个9字节的控制包，再发送一个33800字节的控制包，需要跟踪一下发送列表的包的详细情况

？？？///snow:下面这两部分需要厘清

UploadBandwidthThrottler::RunProc(LPVOID pParam)启动🡪RunInternal()🡪CEMSocket:: SendControlData(uint32 maxNumberOfBytesToSend=9, uint32 minFragSize=536)🡪Send(uint32 maxNumberOfBytesToSend=9, uint32 minFragSize=536, bool onlyAllowedToSendControlPacket=true)

🡪CAsyncSocketExHelperWindow::WindowProc(){case FD\_WRITE}🡪CEMSocket::OnSend(0)

🡪 CEncryptedStreamSocket::OnSend(int nErrorCode=0){nothing to do}

byConnected = ES\_CONNECTED;

2.2.3、

UploadBandwidthThrottler::RunProc(LPVOID pParam)启动🡪RunInternal()🡪CEMSocket:: SendControlData(uint32 maxNumberOfBytesToSend=33800, uint32 minFragSize=1300)🡪Send(uint32 maxNumberOfBytesToSend=33800, uint32 minFragSize=1300, bool onlyAllowedToSendControlPacket=true)🡪CEncryptStreamSocket::Send(const \*lpBuf=0x051fbc40,int nBuflen=86,int nFlags=0)🡪CAsyncSocketEx:Send(const \*lpBuf=0x051fbc40,int nBuflen=86,int nFlags=0)

？？？

③、接收服务器返回信息，触发OnReceive()

🡪CAsyncSocketExHelperWindow::WindowsProc(message=1284>=WM\_SOCKETEX\_NOTIFY){ hSocket=wParam=1136,nEvent=2(FD\_READ)}🡪CServerSocket::OnReceive(nErrorCode=0)🡪 CEMSocket::OnReceive(nErrorCode)

判断是否存在下载速度限制，在未超速的情况下，OnReceive()调用Receive()接收当前情况下可以接收的最大字节数，存入[GlobalReadBuffer + pendingHeaderSize]处。

逐级调用Receive()，最终调用socket函数recv函数，接收数据

🡪 CEMSocket::Receive(GlobalReadBuffer + pendingHeaderSize, readMax);

因为是未加密连接，所以不需要进行解密处理

🡪 CEncryptedStreamSocket::Receive(lpBuf,nBufLen,nFlags){case ECS\_NONE:return m\_nObfuscationBytesReceived;}🡪 CAsyncSocketEx::Receive(lpBuf, nBufLen, nFlags);🡪 recv(m\_SocketData.hSocket, (LPSTR)lpBuf, nBufLen, nFlags);

④、接收到数据后，OnReceive()调用PacketReceive()，因为CEMSocket::PacketReceived()是个虚函数，所以调用CServerSocket::PacketReceived()进行处理

⑤、PacketReceived()先判断包是否进行打包了，若是先进行解包；然后调用CServerSocket::ProcessPacket()对包进行处理

⑥、ProcessPacket()根据opcode对包分别进行处理，在连接时主要是OP\_IDCHANGE、OP\_SERVERMESSAGE等

⑦、OP\_IDCHANGE处理分支设置连接状态为CS\_CONNECTED ,调用SetConnectionState(CS\_CONNECTED)

⑧、SetConnectionState()调用CServerSocket::ConnectionEstablished()，处理CS\_CONNECTED分支，向服务器发送共享文件列表，根据选项中的“从服务器更新服务器列表”，发送请求服务器列表信息包。这里一共两次向服务器发送数据。

⑨、重复③∽⑥，只是⑥中处理opcode的分支不同。

(2)、连接失败

连接失败可能发生在三个阶段：OnHostNameResolved、OnConnect、OnClose中，OnHostNameResolved中返回的是CS\_ERROR，OnConnect中返回的是CS\_FATALERROR、CS\_SERVERDEAD，OnClose中返回的是CS\_DISCONNECTED、CS\_SERVERFULL、CS\_NOTCONNECTED。它们全部通过SetConnectionState设置，然后调用CServerConnect::ConnectionFailed()进行处理，根据情况重新发起连接尝试：

①、case CS\_FATALERROR:暂停30秒，从下一服务器开始连接尝试

case CS\_DISCONNECTED:从服务器列表开始重新连接尝试

②、case CS\_ERROR:

case CS\_NOTCONNECTED:直接退出

③、case CS\_SERVERDEAD:

case CS\_SERVERFULL:如果是单个连接且是加密连接，试着进行非加密连接，否则从下一服务器开始连接尝试。

##### 2.2、加密连接

* 客户端进行加密连接的准备已就绪，Enable protocol obfuscation选项被选中，第三项Disable support for obfuscated connections未选，第二项只跟客户端与客户端之间的连接有关。ConnectToAnyServer （）中bNoCrypt参数默认为false，优先进行乱序加密连接。
* 

###### 2.1.1发起连接

🡪 ConnectTo(CServer\* server=0x04652650, bool bNoCrypt=false)

if ( !bNoCrypt && thePrefs.IsServerCryptLayerTCPRequested() && server->GetObfuscationPortTCP() != 0 && server->SupportsObfuscationTCP()){

nPort = cur\_server->GetObfuscationPortTCP();

SetConnectionEncryption(true, NULL, true);

}

###### 2.1.2 设置连接加密状态为m\_StreamCryptState = ECS\_NONE

🡪 CEncryptedStreamSocket::SetConnectionEncryption(bool bEnabled=true, const uchar\* pTargetClientHash=NULL, bool bServerConnection=true){m\_streamCryptState=ECS\_UNKNOWN,执行else if块，m\_bServerCrypt =true;**m\_StreamCryptState = ECS\_PENDING\_SERVER;**}

###### 2.1.3 设置连接状态为CS\_CONNECTING

🡪SetConnectionState(**CS\_CONNECTING=1**){if,else if均为false,直接返回};

###### 2.1.4 调用socket类发起connect

🡪CEMSocket::Connect(LPCSTR lpszHostAddress=0x05115568, UINT nHostPort=5041)🡪 InitProxySupport(){直接返回}🡪 CAsyncSocketEx::Connect(LPCSTR lpszHostAddress=0x05115568, UINT nHostPort=5041)🡪 connect(m\_SocketData.hSocket=1136, lpSockAddr, nSockAddrLen=16)

###### 2.1.5、处理服务器返回连接信息



如果服务器回应了连接请求，HelperWinodw的FD\_CONNECT事件触发，CServerSocket:: OnConnect() 被调用，根据OnConnect中返回的nErrorCode，如果成功，设置连接状态为等待登录（CS\_WAITFORLOGIN）,调用ConnectionEstablished，如果失败，设置连接状态为CS\_SERVERDEAD或CS——SERVERFATAL，调用ConnectionFailed

(1)、连接成功

①、设置连接状态为等待登录（**CS\_WAITFORLOGIN**）, ConnectionEstablished函数被调用，准备登录信息，调用SendPacket向服务器发送登录信息包，信息包存入待发送队列等待发送, 由UploadBandwidthThrottler根据带宽情况进行择机发送。：

🡪CAsyncSocketExHelperWindow::WindowsProc(message=1284>=WM\_SOCKETEX\_NOTIFY){ hSocket=wParam=1136,nEvent=16(FD\_CONNECT)}🡪CServerSocket::OnConnect(nErrorCode=0)🡪 SetConnectionState(CS\_WAITFORLOGIN)

//在SetConnectionState()中设置连接状态（CS\_WAITFORLOGIN）后，根据连接状态调用CServerConnect::ConnectionEstablished()，处理CS\_WAITFORLOGIN分支：

🡪 CServerConnect::ConnectionEstablished(CServerSocket\* sender=0x050c5d10) 🡪InitLocalIP(){m\_nLocalIP=192.168.1.115}

🡪if(sender->GetConnectionState() == CS\_WAITFORLOGIN),

准备登录信息包，然后调用SendPacket将包添加到发送队列末尾

🡪SendPacket(Packet\* packet,bool delpacket= true, CServerSocket\* sender)🡪 CServerSocket::SendPacket(Packet\* packet, bool delpacket=true, bool controlpacket=true, uint32 actualPayloadSize=0, bool bForceImmediateSend=false)🡪

CEMSocket:: SendPacket(Packet\* packet, bool delpacket=true, bool controlpacket=true, uint32 actualPayloadSize=0, bool bForceImmediateSend=false){if(controlpacket){ controlpacket\_queue.AddTail(packet); // queue up for controlpacket

theApp.uploadBandwidthThrottler->QueueForSendingControlPacket(this, HasSent());},if(bForceImmediateSend)==false}

到这里，OnConnect()处理完成。



②、可以发送，UploadBandwidthThrottler发送信息包，触发OnSend()

在数据被正式发送出去之前，OnSend()首先触发！！！先于send()。问题是OnSend()谁来触发？？？connect()会不会触发OnSend()?看起来不会，有待下一步研究。

FD\_WRITE在两种情况下会被触发：一是connect成功之后会被触发，表示通路接通了，可以发送数据了；另一种是在异步通信中，send()函数返回WSAEWOULDBLOCK，数据包被阻塞了，当阻塞情况解除，数据可以重新发送了，触发FD\_WRITE!

因为是加密连接，CEncryptedStreamSocket::OnSend()执行m\_StreamCryptState == ECS\_PENDING\_SERVER语句块，🡪StartNegotiation(true);启动加密连接协商。同样，在StartNegotiation()函数中,执行m\_StreamCryptState == ECS\_PENDING\_SERVER语句块，准备Client端的握手协商报文，设置

m\_NegotiatingState = ONS\_BASIC\_SERVER\_DHANSWER;

m\_StreamCryptState = ECS\_NEGOTIATING;

m\_nReceiveBytesWanted = 96; 需要获取的字节数

然后调用🡪 SendNegotiatingData()，bDelaySend参数未赋值，默认为false。nBufLen – nStartCryptFromByte=0，不加密数据，直接调用CAsyncSocketEx::Send()发送。

这边需要理清一下，为什么先发一个9字节的控制包，再发送一个33800字节的控制包，需要跟踪一下发送列表的包的详细情况

？？？///snow:下面这两部分需要厘清

UploadBandwidthThrottler::RunProc(LPVOID pParam)启动🡪RunInternal()🡪CEMSocket:: SendControlData(uint32 maxNumberOfBytesToSend=9, uint32 minFragSize=536)🡪Send(uint32 maxNumberOfBytesToSend=9, uint32 minFragSize=536, bool onlyAllowedToSendControlPacket=true)

🡪CAsyncSocketExHelperWindow::WindowProc(){case FD\_WRITE}🡪CEMSocket::OnSend(0)

🡪 CEncryptedStreamSocket::OnSend(int nErrorCode=0){startNegotiation()}

byConnected = ES\_CONNECTED;

2.2.3、

UploadBandwidthThrottler::RunProc(LPVOID pParam)启动🡪RunInternal()🡪CEMSocket:: SendControlData(uint32 maxNumberOfBytesToSend=33800, uint32 minFragSize=1300)🡪Send(uint32 maxNumberOfBytesToSend=33800, uint32 minFragSize=1300, bool onlyAllowedToSendControlPacket=true)🡪CEncryptStreamSocket::Send(const \*lpBuf=0x051fbc40,int nBuflen=86,int nFlags=0)🡪CAsyncSocketEx:Send(const \*lpBuf=0x051fbc40,int nBuflen=86,int nFlags=0)

？？？

③、接收服务器返回信息，触发OnReceive()

🡪CAsyncSocketExHelperWindow::WindowsProc(message=1284>=WM\_SOCKETEX\_NOTIFY){ hSocket=wParam=1136,nEvent=2(FD\_READ)}🡪CServerSocket::OnReceive(nErrorCode=0)🡪 CEMSocket::OnReceive(nErrorCode)

判断是否存在下载速度限制，在未超速的情况下，OnReceive()调用Receive()接收当前情况下可以接收的最大字节数，存入[GlobalReadBuffer + pendingHeaderSize]处。

逐级调用Receive()，最终调用socket函数recv函数，接收数据

🡪 CEMSocket::Receive(GlobalReadBuffer + pendingHeaderSize, readMax);

🡪 CEncryptedStreamSocket::Receive(lpBuf,nBufLen,nFlags){ 🡪 CAsyncSocketEx::Receive(lpBuf, nBufLen, nFlags);🡪 recv(m\_SocketData.hSocket, (LPSTR)lpBuf, nBufLen, nFlags);

因为在StartNegotiation()时，m\_StreamCryptState = ECS\_NEGOTIATING;所以Receive函数执行case ECS\_NEGOTIATING：语句块，首先调用Negotiate()，Negotiate()首先是个while循环，根据需要读取的字节数，在发送来的报文上反复读取：

while (m\_NegotiatingState != ONS\_COMPLETE && m\_nReceiveBytesWanted > 0)

{

第一次：nReceiveBytesWanted=96，m\_NegotiatingState = ONS\_BASIC\_SERVER\_DHANSWER;

在switch中执行case ONS\_BASIC\_SERVER\_DHANSWER:语句块，读取DH密钥（96）字节，并设置m\_NegotiatingState = ONS\_BASIC\_SERVER\_MAGICVALUE; nReceiveBytesWanted=4;

第二次：执行ONS\_BASIC\_SERVER\_MAGICVALUE语句块，读取MAGICVALUE, (MAGICVALUE\_SYNC 0x835E6FC4), 并设置m\_NegotiatingState = ONS\_BASIC\_SERVER\_METHODTAGSPADLEN; nReceiveBytesWanted=3;

第三次：执行ONS\_BASIC\_SERVER\_METHODTAGSPADLEN语句块，读取的值是00 00 09，前两个字节表示METHOD值（ENM\_OBFUSCATION 0），字节（09）表示后面填充的随机数是9字节，设置m\_NegotiatingState = ONS\_BASIC\_SERVER\_PADDING; nReceiveBytesWanted=9；

第四次：执行ONS\_BASIC\_SERVER\_PADDING语句块，准备确认报文，SendNegotiatingData(fileResponse.GetBuffer(), (uint32)fileResponse.GetLength(), 0, true);bDelaySend=true,表示延迟发回数据到服务器，设置m\_NegotiatingState = ONS\_BASIC\_SERVER\_DELAYEDSENDING; m\_StreamCryptState = ECS\_ENCRYPTING;

}

SendNegotiatingData()发送流程：if(lpBuf!=NULL){ 因为参数nStartCryptFromByte =0，所以调用RC4Crypt()加密数据，}，同时m\_pfiSendBuffer==NULL，执行if(…||BDelaySend)语句块，将pBuffer写入m\_pfiSendBuffer，return result=0。

SendNegotiatingData()执行完时，数据并未被发送出去，而是被存入缓冲区，等待下一次需要发送数据的时候，再一起发送出去。具体实现在Send()函数中：

if (m\_bServerCrypt && m\_StreamCryptState == ECS\_ENCRYPTING && m\_pfiSendBuffer != NULL) {

ASSERT( m\_NegotiatingState == ONS\_BASIC\_SERVER\_DELAYEDSENDING );

// handshakedata was delayed to put it into one frame with the first paypload to the server

// do so now with the payload attached

int nRes = SendNegotiatingData(lpBuf, nBufLen, nBufLen);///snow:这里的调用发生在Negotiate()调用SendNegotiatingData之后，是第二次调用SendNegotiatingData，这次将真正的发送出数据

}

Send()再次调用SendNegotiatingData()，而SendNegotiatingData()中，if (m\_pfiSendBuffer != NULL)为真，语句块将被执行，m\_NegotiatingState = ONS\_COMPLETE; m\_pfiSendBuffer->Write(pBuffer, nBufLen);///snow:将Send()时要发送的数据pBuffer附加到m\_pfiSendBuffer后面，两个数据一起发送！！！bProcess=true;这时, bDelaySend为false，CAsyncSocketEx::Send(pBuffer, nBufLen);被调用，数据真正发送出去!

到这里，协商部分终于完成了，状态也设置为ONS\_COMPLETE了，可以正式的发送数据了！

但是上面接收到的数据还没处理完呢！

④、接收到数据后，OnReceive()调用PacketReceive()，因为CEMSocket::PacketReceived()是个虚函数，所以调用CServerSocket::PacketReceived()进行处理

⑤、PacketReceived()先判断包是否进行打包了，若是先进行解包；然后调用CServerSocket::ProcessPacket()对包进行处理

⑥、ProcessPacket()根据opcode对包分别进行处理，在连接时主要是OP\_IDCHANGE、OP\_SERVERMESSAGE等

⑦、OP\_IDCHANGE处理分支设置连接状态为CS\_CONNECTED ,调用SetConnectionState(CS\_CONNECTED)

⑧、SetConnectionState()调用CServerSocket::ConnectionEstablished()，处理CS\_CONNECTED分支，向服务器发送共享文件列表，根据选项中的“从服务器更新服务器列表”，发送请求服务器列表信息包。这里一共两次向服务器发送数据。

⑨、重复③∽⑥，只是⑥中处理opcode的分支不同。

(2)、连接失败

连接失败可能发生在三个阶段：OnHostNameResolved、OnConnect、OnClose中，OnHostNameResolved中返回的是CS\_ERROR，OnConnect中返回的是CS\_FATALERROR、CS\_SERVERDEAD，OnClose中返回的是CS\_DISCONNECTED、CS\_SERVERFULL、CS\_NOTCONNECTED。它们全部通过SetConnectionState设置，然后调用CServerConnect::ConnectionFailed()进行处理，根据情况重新发起连接尝试：

①、case CS\_FATALERROR:暂停30秒，从下一服务器开始连接尝试

case CS\_DISCONNECTED:从服务器列表开始重新连接尝试

②、case CS\_ERROR:

case CS\_NOTCONNECTED:直接退出

③、case CS\_SERVERDEAD:

case CS\_SERVERFULL:如果是单个连接且是加密连接，试着进行非加密连接，否则从下一服务器开始连接尝试。

附：加密连接过程：

18:13:26 Connecting

18:13:28 Connecting to TV Underground (176.103.48.36:4184 - using Protocol Obfuscation) ...

18:13:28 snow:CServerSocket:ConnectTo begin

18:13:28 snow:CAsyncSocketEx:before Connect

18:13:28 snow:CAsyncSocketEx:after connect

18:13:28 snow:CAsyncSocketExHelperWindows::WindowProc FD\_CONNECT

18:13:28 snow:CServerSocket:OnConnect start

18:13:28 snow:CServerSocket:OnConnect SetConnectionState(CS\_WAITFORLOGIN)

///在这之前已经有四个控制包了，两个来回，分别是TCP握手3个包，服务器再发了一个包，第四个包，Window Update

18:13:28 Connected to TV Underground (176.103.48.36:4184), sending login request

18:13:28 >>> Sending OP\_\_LoginRequest

///准备登录信息

18:13:28 snow:CServerSocket:SendPacket size: 80 content:9C431EB71E0E142BF1BCF52D41C16FB0000000003ED804000000020100011800687474703A2F2F656D756C652D70726F6A6563742E6E6574030100113C0000000301002019070000030100FB00C80000铪铪

18:13:28 snow:CEMSocket:Send start

18:13:28 snow:CEMSocket:Send end

18:13:28 snow:CEMSocket:Send start

18:13:28 snow:CEMSocket:Send end

18:13:28 snow:CServerSocket:OnConnect end

18:13:28 snow:CAsyncSocketExHelperWindows::WindowProc FD\_WRITE

///触发OnSend

18:13:28 CEMSocket:OnSend start

18:13:28 snow:CEncryptedStreamSocket:OnSend start

18:13:28 snow:CEncryptedStreamSocket:StartNegotiation m\_StreamCryptState == ECS\_PENDING\_SERVER

18:13:28 snow:CEncryptedStreamSocket:SendNegotiatingData start

///发送的第一个数据包：Client端的握手协商报文

18:13:28 snow:CAsyncSocketEx::Send before send,size : 111 , content : 6F1DF429F4C1E545E9996F6B3F40830A35D279ACDE1B88C0B4811B55788E0C120F1BDA1C7B2F2E81563409ECCE64989CD9AADFD9BBC3E0C89F085D3848EABE36262EB0E8833E07B7AA54AA6F932BF66E25E9D99365E81F94012316AC43122698B30DF5CD15DB67CCE2B32EE963E859铪铪铪

18:13:28 snow:CAsyncSocketEx::Send，after send

18:13:28 snow:CEncryptedStreamSocket:OnSend StartNegotiation(true) ///在这里出现是因为Log语句写在了执行语句之后！

18:13:28 CEMSocket:OnSend end

18:13:28 snow:CEMSocket:Send start

18:13:28 snow:CEMSocket:Send end

....

18:13:28 snow:CEMSocket:Send start

18:13:28 snow:CEMSocket:Send end

///snow:完成乱序加密协商 下面这两个是Log，不是verboseLog，所以先出现了，其实是在后面才会出现的！！

18:13:29 Received proper magic value after DH-Agreement from Serverconnection IP: 176.103.48.36

18:13:29 CEncryptedStreamSocket: Finished DH Obufscation handshake with Server 176.103.48.36

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

18:13:29 snow:CServerSocket:OnReceive start

18:13:29 CEMSocket:OnReceive start

18:13:29 CEMSocket:Receive start

18:13:29 snow:CEncryptedStreamSocket:Receive start

///snow:加密建立之后收到的第一个数据包，服务器发来的DH握手报文，总共112字节，第一次读取96字节，第二次读取4字节，第三次读取3字节，第四次读取9字节

18:13:29 snow:CEncryptedStreamSocket:Receive before DeCrypt size:112 content:59C2DDB81DC6D986B57D2AB42F240416D513CCF2526065D3D995CADE55FC38264B46019202103460BA2806B1FCA28DFF4058A45B0F013BED2171EDD1500B7417AA9633C5FC008022346645DFD332B7C2E68FCF2CF8C3BAC63A42259238DFF824852CB00A2B27B6334823E3CAFF8B5742铪铪

18:13:29 snow:CEncryptedStreamSocket:Receive ECS\_NEGOTIATING

18:13:29 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_SERVER\_DHANSWER

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEncryptedStreamSocket:Negotiate before decrypt size:4 content:852CB00A铪铪

///snow:MAGICVALUE\_SYNC 835E6FC4

18:13:29 snow:CEncryptedStreamSocket:Negotiate after decrypt size:4 content:C46F5E83铪铪

18:13:29 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_SERVER\_MAGICVALUE

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEncryptedStreamSocket:Negotiate before decrypt size:3 content:2B27B6铪铪铪

///snow:ENM\_OBFUSCATION 0 前两个字节是00 00，表示支持DH乱序加密（目前只支持这种方法），第三个字节是09，表示下次准备读取的字节数是9

18:13:29 snow:CEncryptedStreamSocket:Negotiate after decrypt size:3 content:000009铪铪铪

18:13:29 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_SERVER\_METHODTAGSPADLEN

18:13:29 snow:CEncryptedStreamSocket:Negotiate before decrypt size:9 content:334823E3CAFF8B5742铪

18:13:29 snow:CEncryptedStreamSocket:Negotiate after decrypt size:9 content:2908A07BAD447EBB24铪

18:13:29 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_SERVER\_PADDING

///没有单独发送，整合到发送的第二个包了，见第二个包的前6字节

18:13:29 snow:CEncryptedStreamSocket:SendNegotiatingData start

18:13:29 snow:CEncryptedStreamSocket:SendNegotiatingData before crypt size:6 content:CDCDCDCDCDCD

18:13:29 snow:CEncryptedStreamSocket:SendNegotiatingData after crypt size:6 content:88F1796B6BF8

18:13:29 snow:CServerSocket:OnReceive end

///snow:有数据需要发送了，Send()被调用，因为上一环节有延迟发送的数据，Send()再次调用SendNegotiatingData，将本次要发送的数据附加到上次延迟发送的数据后一起发送

18:13:29 snow:CEMSocket:Send start

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEMSocket:Send start

18:13:29 CEncryptedStreamSocket:Send start

18:13:29 snow:CEncryptedStreamSocket:SendNegotiatingData start

///N0.30 Length 146 发的第二个数据包

18:13:29 snow:CAsyncSocketEx::Send before send,size : 92 , content : 88F1796B6BF8A45BEB2735F593F0B1E4BB65A044CC6C456600AF28DD345107FEDCCBDDC3E75AA31D54196266CC11040A1243151287DF4D4979B2C6FF2347DC6E9120111A533178AA9DBB3798788FA33FA112DD0BFD3DBDDB69389624铪铪

18:13:29 snow:CAsyncSocketEx::Send，after send

18:13:29 snow:CEMSocket:Send end

18:13:29 snow:CEMSocket:Send start

.......

18:13:30 snow:CEMSocket:Send start

18:13:30 snow:CEMSocket:Send end

///snow:服务器同时向本机发起连接请求，发送了OP\_Hello，本机回复了OP\_HelloAnswer

18:13:30 snow:CAsyncSocketExHelperWindows::WindowProc FD\_WRITE

18:13:30 CEMSocket:OnSend start

18:13:30 snow:CEncryptedStreamSocket:OnSend start

18:13:30 snow:CEncryptedStreamSocket:OnSend end

18:13:30 CEMSocket:OnSend end

18:13:30 snow:CEMSocket:Send start

18:13:30 snow:CEMSocket:Send end

......

18:13:30 snow:CEMSocket:Send start

18:13:30 snow:CEMSocket:Send end

18:13:31 OP\_Hello from 176.103.48.36 'eServer' (eMule v0.47c [eserver],None/None/None)

18:13:31 Hash=AD7727272E0EA7D2ABE16B2551136F69 (eMule) UserID=607152048 (176.103.48.36) Port=4662 Tags=4

Name='eServer'

Version=60

ClientVer=0.47.2.0 Comptbl=0

ModID=eserver

Server=0.0.0.0:0

18:13:31 >>> OP\_\_HelloAnswer to 176.103.48.36 'eServer' (eMule v0.47c [eserver],None/None/None)

18:13:31 snow:CEMSocket:Send start

18:13:31 snow:CEMSocket:Send end

18:13:31 snow:CEMSocket:Send start

18:13:31 snow:CEMSocket:Send end

18:13:31 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

18:13:31 CEMSocket:OnReceive start

18:13:31 CEMSocket:Receive start

18:13:31 snow:CEncryptedStreamSocket:Receive start

18:13:31 snow:CEncryptedStreamSocket:Receive before DeCrypt size:64 content:43B02C8C4C323E51A329CD2E5BC670613BFC96EF70AAB15825E1C4333FAECF39D870B7BD8FD3A058BA542445E08A9EA6752165F90EB69BAAE8B3FB730592B1D3铪铪铪铪铪铪

18:13:31 snow:CEncryptedStreamSocket:Receive ECS\_ENCRYPTING

18:13:31 snow:CEncryptedStreamSocket:Receive after DeCrypt size:64 content:E33B0000000110AD7727272E0EA7D2ABE16B2551136F69B067302436120400000097016553657276657289113C88FB00BD975565736572766572000000000000铪铪铪铪铪铪

18:13:31 snow:CEMSocket:Send start

18:13:31 CEncryptedStreamSocket:Send start

18:13:31 snow:CEncryptedStreamSocket:Send end

18:13:31 snow:CAsyncSocketEx::Send before send,size : 108 , content : 742299CA30F7FE30C0A4DBF610227EEA9D2E78A76FBE61467DBED121D65C98BBDA1E0DB5C5A9557C72545579066E1EA80A5C92E4456A4AD1FDB1B9F6043C7F630A402A9EDC55F58A9CD752BBFF4BE45C73CE9F9A8DD4760BBAB13F9FA3EAC52598E27CFE2D504CB7F43ABA72铪铪

18:13:31 snow:CAsyncSocketEx::Send，after send

18:13:31 snow:CEMSocket:Send end

18:13:31 CEMSocket:OnReceive end

18:13:31 snow:CEMSocket:Send start

18:13:31 snow:CEMSocket:Send end

.....

18:13:31 snow:CEMSocket:Send start

18:13:31 snow:CEMSocket:Send end

18:13:31 ServerMsg - OP\_IDChange

18:13:31 TCP Flags=0x000017f9 \*\*\*UnkBits=0x00001220 Compression=1 NewTags=1 Unicode=1 RelatedSearch=1 IntTypeTags=1 LargeFiles=1 TCP\_Obfscation=1

18:13:31 Obfuscated connection established on: TV Underground (176.103.48.36:4184)

18:13:31 >>> Sending OP\_\_OfferFiles(compressed); uncompr size=334 compr size=322 files=3

18:13:31 Server, Sendlist: Packet size:322

18:13:31 Possible IP Change - Checking for expired Server UDP-Keys: 15 UDP Keys total, 0 UDP Keys expired, 0 immediate UDP Pings forced, 0 delayed UDP Pings forced

18:13:31 New client ID is 71042416

18:13:31 ServerMsg - OP\_ServerStatus

18:13:32 ServerMsg - OP\_ServerMessage

18:13:32 server version 17.15 (lugdunum)

18:13:32 --- Deleted client 176.103.48.36 'eServer' (eMule v0.47c [eserver],None/None/None); Reason=CClientReqSocket::Disconnect(): Close

18:13:32 snow:CEMSocket:Send start

18:13:32 snow:CEMSocket:Send end

18:13:32 snow:CEMSocket:Send start

18:13:32 snow:CEMSocket:Send end

///snow:接收到的第二个数据包 No. 49, length :297

18:13:32 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

18:13:32 snow:CServerSocket:OnReceive start

18:13:32 CEMSocket:OnReceive start

18:13:32 CEMSocket:Receive start

18:13:32 snow:CEncryptedStreamSocket:Receive start

18:13:32 snow:CEncryptedStreamSocket:Receive before DeCrypt size:243 content:44D0AFAEBBBBA2576AFDF00A018D5AC028C5D3E45ED77E51204EEA33BB5232ECA6110CD00BDB383BF93DEC3E57A237FABAC58FF50C8000DA9BC85E62EE4E62CEE2E6C3607BD4A36A43438A6F19881AE715C929C135CF4967E3F97E21B6A001E4DA5F737B27ABC6BC49F4BECAD43FF4CD427BE2A388806A9B9BE7C2E1119A600AA1B7AD920449C3F7E12E9DADEF9489CAA2F02D8E144574A56A159163B677123CED85055C75EC246F92EE776108D8CF5B5040E5A200EF165EE9CE8AC00F27EEABE16FFEEBD8BAF1BDB7AD7AF52F4F4F3797BA06456874824C17D46BD79B125584EB4C1E8BF94E097E4DD616D9CB4BCF7A5C8349铪铪铪

18:13:32 snow:CEncryptedStreamSocket:Receive ECS\_ENCRYPTING

18:13:32 snow:CEncryptedStreamSocket:Receive after DeCrypt size:243 content:E3110000004070053C04F91700005810000070053C04E30900000034F57A010097069601D4CA0000003878DA5D8FCB6AC3400C45E9D65F719729149B2C4AA11FD0751769F6138F3C33608F1C6964C8DF57267DD02E047ADE7B747850928D041E5AB8E2F8D21F9F71982D45ABB63C76EF3305259C8B9686DCDAFA3A0C1A260A352A5B8D99B5F5232F985830660A2B7C04A5D184C0D3A4993DD11C84E2B0156916E621522C636814B1DF979ABA532E8A2F983D5BB9AA1F465C6E389DF151234992DDF109CA58EF58DB0F963A57DBEC77AD6749BD58F7E65CA1DE70357227578537D452FA2EFF48B54CFFECF6C76CD14FF7FA6DA8铪铪铪

///snow:分割成数据包（packet) 为什么size总是多一个字节？因为程序故意将字节数+1了，在构建Packet时又把它减掉了Packet::Packet(char\* header){size = head->packetlength-1; ///snow:这边把读取到的size减1了...}

1、E3110000004070053C04F91700005810000070053C04 6+16=22字节

E3（prot OP\_EDONKEYPROT) 11000000(size：17-1字节，） 40(opcode OP\_IDChange) 7005 3C04（4个字节,ID：71042416，0x043C0570) F917 0000 (4个字节 TCP Flags)5810 0000（4184 dwObfuscationTCPPort） 70053C04（dwServerReportedIP）

TCP Flags=0x000017f9（01 01 11 11 11 10 01） \*\*\*UnkBits=0x00001220 （1220=01011111111001&~10111011001=17F9&~05D9(dwKnownBits）

从右往左 第1位Compression=1 第4位NewTags=1 第5位Unicode=1 第7位RelatedSearch=1 第8位IntTypeTags=1 第9位LargeFiles=1 第11位TCP\_Obfscation=1

2、E30900000034F57A010097069601 6+8=14字节

E3（prot OP\_EDONKEYPROT) 09000000(size：9-1字节，） 34(opcode OP\_SERVERSTATUS) F57A 0100 (UserCount) 9706 9601(FileCount)

3、 6+201=207字节D4CA0000003878DA5D8FCB6AC3400C45E9D65F719729149B2C4AA11FD0751769F6138F3C33608F1C6964C8DF57267DD02E047ADE7B747850928D041E5AB8E2F8D21F9F71982D45ABB63C76EF3305259C8B9686DCDAFA3A0C1A260A352A5B8D99B5F5232F985830660A2B7C04A5D184C0D3A4993DD11C84E2B0156916E621522C636814B1DF979ABA532E8A2F983D5BB9AA1F465C6E389DF151234992DDF109CA58EF58DB0F963A57DBEC77AD6749BD58F7E65CA1DE70357227578537D452FA2EFF48B54CFFECF6C76CD14FF7FA6DA8

D4(prot OP\_PACKEDPROT) CA000000(size:201-1字节） 38（opcode OP\_SERVERMESSAGE)

server version 17.15 (lugdunum)

18:13:32 snow:CSharedFileList::SendListToServer before Pack size:334 content

09:41:38 >>> Sending OP\_\_OfferFiles(compressed); uncompr size=334 compr size=322 files=3

///snow:03000000 三个文件

///FCFCFCFCFCFC PartiaFile

///FBFBFBFBFBFB

18:13:32 snow:CEMSocket:Send start

18:13:32 snow:CEMSocket:Send end

.....

18:13:32 snow:CEMSocket:Send start

18:13:32 snow:CEMSocket:Send end

18:13:32 snow:CServerSocket:SendPacket 压缩后的字节流 size: 322 content

OP\_\_OfferFiles(compressed); uncompr size=334 compr size=322 files=3

///snow：加上包头

09:41:38 snow:CEncryptedStreamSocket:CryptPrepareSendData before Crypt size:328 content铪铪

///snow:D4（OP\_PACKPROT) 43010000（323-1字节） 15(OP\_OFFERFILES)

18:13:32 snow:CEMSocket:Send start

18:13:32 CEncryptedStreamSocket:Send start

18:13:32 snow:CEncryptedStreamSocket:Send end

///snow:发送的第三个数据包 No. 58 length 382 加密后的数据 本机共享的文件列表

18:13:32 snow:CAsyncSocketEx::Send before send,size : 328 , content铪铪

18:13:32 snow:CAsyncSocketEx::Send，after send

18:13:32 snow:CEMSocket:Send end

///snow:发送的第四个数据包

18:13:32 >>> Sending OP\_\_GetServerList

18:13:32 snow:CServerSocket:SendPacket size: 0 content:铪铪

18:13:32 snow:CEMSocket:Send start

18:13:32 snow:CEncryptedStreamSocket:CryptPrepareSendData before Crypt size:6 content:E30100000014

18:13:32 snow:CEncryptedStreamSocket:CryptPrepareSendData after Crypt size:6 content:03981ABF7AD3铪铪铪铪

18:13:32 CEncryptedStreamSocket:Send start

18:13:32 snow:CEncryptedStreamSocket:Send end

18:13:32 snow:CAsyncSocketEx::Send before send,size : 6 , content : CBA98C31FB8A

18:13:32 snow:CAsyncSocketEx::Send，after send

18:13:32 snow:CEMSocket:Send end

18:13:32 CEMSocket:OnReceive end

18:13:32 snow:CServerSocket:OnReceive end

///snow:E3 01000000(0字节） 14(OP\_\_GetServerList)

18:13:32 snow:CAsyncSocketExHelperWindows::WindowProc FD\_CLOSE

18:13:33 ServerMsg - OP\_ServerList

18:13:33 ServerMsg - OP\_ServerIdent

18:13:33 Hash=AD7727272E14A7D2ABE16B2551133269 (Unknown) IP=176.103.48.36:4184 Tags=2 Name=TV Underground Desc=Operated by TVUnderground.org.ru

18:13:33 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

18:13:33 snow:CServerSocket:OnReceive start

18:13:33 CEMSocket:OnReceive start

18:13:33 CEMSocket:Receive start

18:13:33 snow:CEncryptedStreamSocket:Receive start

///snow:接收的第三个数据包 No. 68 length 211 （OP\_SERVERLIST）与（OP\_SERVERIDENT）

18:13:33 snow:CEncryptedStreamSocket:Receive before DeCrypt size:157 content:D8F565C546447B0EB275034C2E49FA6E144494B8CDBB64EA73A3A5533CA26BCFD9D142497AF7911B9AAF06DCE892AEA446C36DCF757704248F932E217DDF3EC1BD96029644037F19A7493313045745A1A2CCE2CE8DC313F595B69D78EC6FE4327AD7D1967B7803C9B254E3F373EDEAA83C25C9895F0CF204F094AA1ACEB0E570D556B61FFC6B63961B400A11B187A9B4F30892FD472600197443E7667F铪

18:13:33 snow:CEncryptedStreamSocket:Receive ECS\_ENCRYPTING

18:13:33 snow:CEncryptedStreamSocket:Receive after DeCrypt size:157 content:E33E000000320AD453B898C71BB06738628A09B06738878A09B067302458105BD26A8588105F4318C38BF3DE288E03889C55CC32748810C39A5305C71BD598A8BD9121E35500000041AD7727272E14A7D2ABE16B2551133269B0673024581002000000020100010E00545620556E64657267726F756E640201000B20004F70657261746564206279205456556E64657267726F756E642E6F72672E7275铪

/// E3(OP\_EDONKEYPROT) 3E000000(62-1字节） 32（OP\_SERVERLIST））0A(10个服务器）每个服务器（4字节IP+2字节Port)，6\*10共60字节

E3(OP\_EDONKEYPROT) 55000000（85-1字节） 41 （OP\_SERVERIDENT） AD7727272E14A7D2ABE16B2551133269（HASH) B0673024(serverip 176.103.48.36) 5810(port:4184)

02000000(Tags） Name=TV Underground Desc=Operated by TVUnderground.org.ru

18:13:33 CEMSocket:OnReceive end

18:13:33 snow:CServerSocket:OnReceive end

18:13:37 snow:CEMSocket:Send start

18:13:37 CEncryptedStreamSocket:Send start

18:13:37 snow:CEncryptedStreamSocket:Send end

///snow:发送的第四个数据包 No.95 length 114 这个包又是什么呢？

18:13:37 snow:CAsyncSocketEx::Send before send,size : 60 , content : 1DD62125F43C36A712F6E699ECB7D800D16D341DDE58E8C9E93F299E042419CF724249CD72EB4F5BC3960999EC03F9ED316C3945C9217C93C36BA0DD铪铪

18:13:37 snow:CAsyncSocketEx::Send，after send

18:13:37 snow:CEMSocket:Send end

18:13:38 ServerMsg - OP\_FoundSources\_OBFU; Sources=1 File=This Ain't The Expendables XXX 3D 2012 BDRip halfSBS 1080p (IgorekSh).mkv

18:13:38 SXRecv: Server source response; Count=1, Dropped=0, PossibleSources=1, File="This Ain't The Expendables XXX 3D 2012 BDRip halfSBS 1080p (IgorekSh).mkv"

18:13:38 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

18:13:38 snow:CServerSocket:OnReceive start

18:13:38 CEMSocket:OnReceive start

18:13:38 CEMSocket:Receive start

18:13:38 snow:CEncryptedStreamSocket:Receive start

///snow:接收的第四个数据包

18:13:38 snow:CEncryptedStreamSocket:Receive before DeCrypt size:46 content:A55F553411CC10A63F0D9D76B94DEE02AC2BF967F8B0982E3BEEEE138EC83049AB6C9B95A628BB3B423EC05C6736

18:13:38 snow:CEncryptedStreamSocket:Receive ECS\_ENCRYPTING

18:13:38 snow:CEncryptedStreamSocket:Receive after DeCrypt size:46 content:E32900000044AAE4E22D13DFBF768780B443CA8D3F85015CB085D14639831FCFE2DC9E0E876106938E6D8F716F39

18:13:38 CEMSocket:OnReceive end

18:13:38 snow:CServerSocket:OnReceive end

///snow:E3 29000000(41-1字节） 44（OP\_FoundSources\_OBFU） Sources=1 File=This Ain't The Expendables XXX 3D 2012 BDRip halfSBS 1080p (IgorekSh).mkv

///数据包的接收到止为止，后面还有4个控制包，其中3个应该是断开连接的

18:15:30 Closing eMule

18:15:30 Deleted public IP

18:15:30 Stopping Kademlia

18:15:30 Wrote 191 contacts to file.

18:15:31 Wrote 3 source, 6084 keyword, and 6 load entries

18:15:31 Saving known files list file "known.met"

18:15:31 Saving known files list file "cancelled.met"

18:15:31 Stored 0 open search for restoring on next start

18:15:31 Aborted any possible UPnP StartDiscoveryThread

18:15:31 No UPnP Mappings to remove, aborting

## 四、客户端呼入连接过程

1、有客户端呼入请求连接，CAsyncSocketExHelperWindows::WindowProc FD\_ACCEPT事件触发，调用OnAccept()函数，CListenSocket::OnAccept启动，根据是否设置了条件接入，分别调用WSAAccept或accept接受连接，调用AsyncSelect(FD\_WRITE | FD\_READ | FD\_CLOSE); ///snow:在建立的Socket上接收FD\_WRITE | FD\_READ | FD\_CLOSE事件

2、FD\_READ事件触发，顺序调用CClientReqSocket::OnReceive() 🡪CEMSocket::OnReceive() 🡪Receive() 🡪CEncryptedStreamSocket::Receive()

3、在CEncryptedStreamSocket::Receive()根据m\_StreamCryptState分别启动加密联接（ECS\_UNKNOWN，StartNegotiation(false)）或无加密连接（ECS\_NONE，直接返回接收字节数）

处理过程跟服务器连接相似，

（1）加密连接：

StartNegotiation()执行if(!bOutgoing){

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_RANDOMPART;

m\_StreamCryptState = ECS\_NEGOTIATING;

m\_nReceiveBytesWanted = 4;

}

Negotiate():

While{

case ONS\_BASIC\_CLIENTA\_RANDOMPART:

准备ClinetB Key

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_MAGICVALUE;

m\_nReceiveBytesWanted = 4;

case ONS\_BASIC\_CLIENTA\_MAGICVALUE:{

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_METHODTAGSPADLEN;

m\_nReceiveBytesWanted = 3;

case ONS\_BASIC\_CLIENTA\_METHODTAGSPADLEN:

m\_nReceiveBytesWanted = m\_pfiReceiveBuffer->ReadUInt8();

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_PADDING;

case ONS\_BASIC\_CLIENTA\_PADDING:

准备HandShake ClientB报文

SendNegotiatingData(fileResponse.GetBuffer(), (uint32)fileResponse.GetLength());

m\_NegotiatingState = ONS\_COMPLETE;

m\_StreamCryptState = ECS\_ENCRYPTING;

}

协商完毕后，加密连接已经建立，开始处理接收到的信息包：

只是调用PacketReceived时实际调用的是CClientReqSocket类中的PacketReceived()，然后再根据是否设置NO\_USE\_CLIENT\_TCP\_CATCH\_ALL\_HANDLER分别调用PacketReceivedCppEH或PacketReceivedSEH(还是调用PacketReceivedCppEH，只是多了异常处理部分)。

3、PacketReceivedCppEH()根据包的协议属性分别处理：

OP\_EDONKEYPROT ： ProcessPacket处理，呼入连接发出的OP\_Hello属于OP\_EDONKEYPROT

OP\_PACKEDPROT：先解压，如果是OP\_EMULEPROT，ProcessExtPacket处理，否则报错

OP\_EMULEPROT：ProcessExtPacket处理

4、ProcessPacket()对各种信息包进行处理，在客户端呼入连接中主要处理OP\_HELLO、OP\_HELLOANSWER信息

**5、在**OP\_HELLO语句块中，构造CUpDownClient对象client，然后调用成员函数client->ProcessHelloPacket(packet,size)🡪ProcessHelloTypePacket(CSafeMemFile\* data)，将OP\_HELLO报文分解为Hash，UserID，Port，Tags以及Additional data

// \*) eDonkeyHybrid 0.40 - 1.2 sends an additional Int32. (Since 1.3 they don't send it any longer.)

// \*) MLdonkey sends an additional Int32

Tags种类：

CT\_NAME、CT\_VERSION、CT\_PORT、CT\_MOD\_VERSION、CT\_EMULE\_UDPPORTS、CT\_EMULE\_BUDDYUDP、CT\_EMULE\_BUDDYIP、CT\_EMULE\_MISCOPTIONS1、CT\_EMULE\_MISCOPTIONS2、CT\_EMULE\_VERSION。

然后读取客户端的ID，信用记录，朋友列表，判断是不是eMule客户端（CT\_EMULE\_UDPPORTS、CT\_EMULE\_MISCOPTIONS1、 CT\_EMULE\_MISCOPTIONS2、CT\_EMULE\_VERSION四个标志改变dwEmuleTags的值，但判断bIsMule只根据Tags中是否存在CT\_EMULE\_VERSION）

**6、与列表中已存在的client对象比较，是否已存在同一client,如果有，更新旧的client信息，如果没有，则添加新client.**

**7、如果client的HASH中包含SO\_EMULE标记，但不包含**CT\_EMULE\_VERSION标签，SendMuleInfoPacket(false)，false表示不是回答报文

8、回发HelloAnswer报文

9、调用ConnectionEstablished()处理连接

10、处理到客户端的KAD连接

## 五、客户端呼出连接过程

1、有客户端呼入请求连接，CAsyncSocketExHelperWindows::WindowProc FD\_ACCEPT事件触发，调用OnAccept()函数，CListenSocket::OnAccept启动，根据是否设置了条件接入，分别调用WSAAccept或accept接受连接，调用AsyncSelect(FD\_WRITE | FD\_READ | FD\_CLOSE); ///snow:在建立的Socket上接收FD\_WRITE | FD\_READ | FD\_CLOSE事件

2、FD\_READ事件触发，顺序调用CClientReqSocket::OnReceive() 🡪CEMSocket::OnReceive() 🡪Receive() 🡪CEncryptedStreamSocket::Receive()

3、在CEncryptedStreamSocket::Receive()根据m\_StreamCryptState分别启动加密联接（ECS\_UNKNOWN，StartNegotiation(false)）或无加密连接（ECS\_NONE，直接返回接收字节数）

处理过程跟服务器连接相似，

（1）加密连接：

StartNegotiation()执行if(!bOutgoing){

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_RANDOMPART;

m\_StreamCryptState = ECS\_NEGOTIATING;

m\_nReceiveBytesWanted = 4;

}

Negotiate():

While{

case ONS\_BASIC\_CLIENTA\_RANDOMPART:

准备ClinetB Key

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_MAGICVALUE;

m\_nReceiveBytesWanted = 4;

case ONS\_BASIC\_CLIENTA\_MAGICVALUE:{

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_METHODTAGSPADLEN;

m\_nReceiveBytesWanted = 3;

case ONS\_BASIC\_CLIENTA\_METHODTAGSPADLEN:

m\_nReceiveBytesWanted = m\_pfiReceiveBuffer->ReadUInt8();

m\_NegotiatingState = ONS\_BASIC\_CLIENTA\_PADDING;

case ONS\_BASIC\_CLIENTA\_PADDING:

准备HandShake ClientB报文

SendNegotiatingData(fileResponse.GetBuffer(), (uint32)fileResponse.GetLength());

m\_NegotiatingState = ONS\_COMPLETE;

m\_StreamCryptState = ECS\_ENCRYPTING;

}

协商完毕后，加密连接已经建立，开始处理接收到的信息包：

只是调用PacketReceived时实际调用的是CClientReqSocket类中的PacketReceived()，然后再根据是否设置NO\_USE\_CLIENT\_TCP\_CATCH\_ALL\_HANDLER分别调用PacketReceivedCppEH或PacketReceivedSEH(还是调用PacketReceivedCppEH，只是多了异常处理部分)。

3、PacketReceivedCppEH()根据包的协议属性分别处理：

OP\_EDONKEYPROT ： ProcessPacket处理，呼入连接发出的OP\_Hello属于OP\_EDONKEYPROT

OP\_PACKEDPROT：先解压，如果是OP\_EMULEPROT，ProcessExtPacket处理，否则报错

OP\_EMULEPROT：ProcessExtPacket处理

4、ProcessPacket()对各种信息包进行处理，在客户端呼入连接中主要处理OP\_HELLO、OP\_HELLOANSWER信息

**5、在**OP\_HELLO语句块中，构造CUpDownClient对象client，然后调用成员函数client->ProcessHelloPacket(packet,size)🡪ProcessHelloTypePacket(CSafeMemFile\* data)，将OP\_HELLO报文分解为Hash，UserID，Port，Tags以及Additional data

// \*) eDonkeyHybrid 0.40 - 1.2 sends an additional Int32. (Since 1.3 they don't send it any longer.)

// \*) MLdonkey sends an additional Int32

Tags种类：

CT\_NAME、CT\_VERSION、CT\_PORT、CT\_MOD\_VERSION、CT\_EMULE\_UDPPORTS、CT\_EMULE\_BUDDYUDP、CT\_EMULE\_BUDDYIP、CT\_EMULE\_MISCOPTIONS1、CT\_EMULE\_MISCOPTIONS2、CT\_EMULE\_VERSION。

然后读取客户端的ID，信用记录，朋友列表，判断是不是eMule客户端（CT\_EMULE\_UDPPORTS、CT\_EMULE\_MISCOPTIONS1、 CT\_EMULE\_MISCOPTIONS2、CT\_EMULE\_VERSION四个标志改变dwEmuleTags的值，但判断bIsMule只根据Tags中是否存在CT\_EMULE\_VERSION）

**6、与列表中已存在的client对象比较，是否已存在同一client,如果有，更新旧的client信息，如果没有，则添加新client.**

**7、如果client的HASH中包含SO\_EMULE标记，但不包含**CT\_EMULE\_VERSION标签，SendMuleInfoPacket(false)，false表示不是回答报文

8、回发HelloAnswer报文

9、调用ConnectionEstablished()处理连接

10、处理到客户端的KAD连接

# CclientReqSocket::ProcessPacket处理OP\_Code

#define OP\_HELLO 0x01 // 0x10<HASH 16><ID 4><PORT 2><1 Tag\_set>

#define OP\_SENDINGPART 0x46 // <HASH 16><von 4><bis 4><Daten len:(von-bis)>

#define OP\_REQUESTPARTS 0x47 // <HASH 16><von[3] 4\*3><bis[3] 4\*3>

#define OP\_FILEREQANSNOFIL 0x48 // <HASH 16>

#define OP\_END\_OF\_DOWNLOAD 0x49 // <HASH 16>

#define OP\_ASKSHAREDFILES 0x4A // (null)

#define OP\_ASKSHAREDFILESANSWER 0x4B // <count 4>(<HASH 16><ID 4><PORT 2><1 Tag\_set>)[count]

#define OP\_HELLOANSWER 0x4C // <HASH 16><ID 4><PORT 2><1 Tag\_set><SERVER\_IP 4><SERVER\_PORT 2>

#define OP\_CHANGE\_CLIENT\_ID 0x4D // <ID\_old 4><ID\_new 4>

#define OP\_MESSAGE 0x4E // <len 2><Message len>

#define OP\_SETREQFILEID 0x4F // <HASH 16>

#define OP\_FILESTATUS 0x50 // <HASH 16><count 2><status(bit array) len:((count+7)/8)>

#define OP\_HASHSETREQUEST 0x51 // \*DEPRECATED\* <HASH 16>

#define OP\_HASHSETANSWER 0x52 // \*DEPRECATED\* <count 2><HASH[count] 16\*count>

#define OP\_STARTUPLOADREQ 0x54 // <HASH 16>

#define OP\_ACCEPTUPLOADREQ 0x55 // (null)

#define OP\_CANCELTRANSFER 0x56 // (null)

#define OP\_OUTOFPARTREQS 0x57 // (null)

#define OP\_REQUESTFILENAME 0x58 // <HASH 16> (more correctly file\_name\_request)

#define OP\_REQFILENAMEANSWER 0x59 // <HASH 16><len 4><NAME len>

#define OP\_CHANGE\_SLOT 0x5B // <HASH 16>

#define OP\_QUEUERANK 0x5C // <wert 4> (slot index of the request)

#define OP\_ASKSHAREDDIRS 0x5D // (null)

#define OP\_ASKSHAREDFILESDIR 0x5E // <len 2><Directory len>

#define OP\_ASKSHAREDDIRSANS 0x5F // <count 4>(<len 2><Directory len>)[count]

#define OP\_ASKSHAREDFILESDIRANS 0x60 // <len 2><Directory len><count 4>(<HASH 16><ID 4><PORT 2><1 Tag\_set>)[count]

#define OP\_ASKSHAREDDENIEDANS 0x61 // (null)

# CclientReqSocket::ProcessExtPacket处理OP\_Code

// extened prot client <-> extened prot client

#define OP\_EMULEINFO 0x01 //

#define OP\_EMULEINFOANSWER 0x02 //

#define OP\_COMPRESSEDPART 0x40 // <HASH 16><von 4><size 4><Daten len:size>

#define OP\_QUEUERANKING 0x60 // <RANG 2>

#define OP\_FILEDESC 0x61 // <len 2><NAME len>

#define OP\_REQUESTSOURCES 0x81 // \*DEPRECATED\* <HASH 16>

#define OP\_ANSWERSOURCES 0x82 // \*DEPRECATED\*

#define OP\_REQUESTSOURCES2 0x83 // <HASH 16><Version 1><Options 2>

#define OP\_ANSWERSOURCES2 0x84 // <Version 1>[content]

#define OP\_PUBLICKEY 0x85 // <len 1><pubkey len>

#define OP\_SIGNATURE 0x86 // v1: <len 1><signature len> v2:<len 1><signature len><sigIPused 1>

#define OP\_SECIDENTSTATE 0x87 // <state 1><rndchallenge 4>

#define OP\_REQUESTPREVIEW 0x90 // <HASH 16>

#define OP\_PREVIEWANSWER 0x91 // <HASH 16><frames 1>{frames \* <len 4><frame len>}

#define OP\_MULTIPACKET 0x92 // \*DEPRECATED\*

#define OP\_MULTIPACKETANSWER 0x93 // \*DEPRECATED\*

#define OP\_PEERCACHE\_QUERY 0x94 // \*DEPRECATED\*

#define OP\_PEERCACHE\_ANSWER 0x95 // \*DEPRECATED\*

#define OP\_PEERCACHE\_ACK 0x96 // \*DEPRECATED\*

#define OP\_PUBLICIP\_REQ 0x97

#define OP\_PUBLICIP\_ANSWER 0x98

#define OP\_CALLBACK 0x99 // <HASH 16><HASH 16><uint 16>

#define OP\_REASKCALLBACKTCP 0x9A

#define OP\_AICHREQUEST 0x9B // \*DEPRECATED\* <HASH 16><uint16><HASH aichhashlen>

#define OP\_AICHANSWER 0x9C // \*DEPRECATED\* <HASH 16><uint16><HASH aichhashlen> <data>

#define OP\_AICHFILEHASHANS 0x9D

#define OP\_AICHFILEHASHREQ 0x9E

#define OP\_BUDDYPING 0x9F

#define OP\_BUDDYPONG 0xA0

#define OP\_COMPRESSEDPART\_I64 0xA1 // <HASH 16><von 8><size 4><Daten len:size>

#define OP\_SENDINGPART\_I64 0xA2 // <HASH 16><von 8><bis 8><Daten len:(von-bis)>

#define OP\_REQUESTPARTS\_I64 0xA3 // <HASH 16><von[3] 8\*3><bis[3] 8\*3>

#define OP\_MULTIPACKET\_EXT 0xA4 // \*DEPRECATED\*

#define OP\_CHATCAPTCHAREQ 0xA5 // <tags 1>[tags]<Captcha BITMAP>

#define OP\_CHATCAPTCHARES 0xA6 // <status 1>

#define OP\_FWCHECKUDPREQ 0xA7 // <Inter\_Port 2><Extern\_Port 2><KadUDPKey 4> \*Support required for Kadversion >= 6

#define OP\_KAD\_FWTCPCHECK\_ACK 0xA8 // (null/reserved), replaces KADEMLIA\_FIREWALLED\_ACK\_RES, \*Support required for Kadversion >= 7

#define OP\_MULTIPACKET\_EXT2 0xA9 // <FileIdentifier> ...

#define OP\_MULTIPACKETANSWER\_EXT2 0xB0 // <FileIdentifier> ...

#define OP\_HASHSETREQUEST2 0xB1 // <FileIdentifier><Options 1>

#define OP\_HASHSETANSWER2 0xB2 // <FileIdentifier><Options 1>[<HashSets> Options]

#define OP\_PORTTEST 0xFE // Connection Test

# CClientUDPSocket::ProcessPacket处理Op\_Code

// extened prot client <-> extened prot client UDP

#define OP\_REASKFILEPING 0x90 // <HASH 16>

#define OP\_REASKACK 0x91 // <RANG 2>

#define OP\_FILENOTFOUND 0x92 // (null)

#define OP\_QUEUEFULL 0x93 // (null)

#define OP\_REASKCALLBACKUDP 0x94

#define OP\_DIRECTCALLBACKREQ 0x95 // <TCPPort 2><Userhash 16><ConnectionOptions 1>

#define OP\_PORTTEST 0xFE // Connection Test

# emule中节点加入Kad网络过程（源代码详解）【对原文部分改进】

emule中节点加入Kad网络过程（源代码详解）

程序启动：

**EmuleDlg.cpp中函数**BOOL CemuleDlg::OnInitDialog()**，此函数用于对话框的初始化，在这个函数里添加了定时器**：VERIFY( (m\_hTimer = ::SetTimer(NULL, NULL, 300, StartupTimer)) != NULL );

**在这里添加了函数**void CALLBACK CemuleDlg::**StartupTimer**(HWND /\*hwnd\*/, UINT /\*uiMsg\*/, UINT /\*idEvent\*/, DWORD /\*dwTime\*/)，

case 2:

theApp.Kad\_Dlg->status++;

if(!theApp.listensocket->StartListening())

ASSERT(0);

if(!theApp.clientudp->Create())

ASSERT(0);

theApp.Kad\_Dlg->status++;

break;

**[PS: 现在已经不是这样了，没有了Kad\_Dlg, 在cemuleDlg.cpp的2087行调用了Kad的Start()函数]**

**在StartupTimer这个函数里，添加了一个ListenSocket的侦听端，并且在本地节点创建了一个CClientUDPSocket\* clientudp;**

然后程序启动。

顺便说一句，在CEmule类中定义了许多的类的实例，这都在今后使用到：

UploadBandwidthThrottler\* uploadBandwidthThrottler;

CClientList\* clientlist;

CClientUDPSocket\* clientudp;

CListenSocket\* listensocket;

CSharedFileList\* sharedfiles;

CDownloadQueue\* downloadqueue;

CUploadQueue\* uploadqueue;

CServerList\* serverlist;

LastCommonRouteFinder\* lastCommonRouteFinder;

CServerConnect\* serverconnect;

CIPFilter\* ipfilter;

CClientCreditsList\* clientcredits;

CSearchList\* searchlist;

CKnownFileList\* knownfiles;

CMMServer\* mmserver;

AppState m\_app\_state; // defines application state for shutdown

CMutex hashing\_mut;

CString m\_strCurVersionLong;

CPeerCacheFinder\* m\_pPeerCache;

CFriendList\* friendlist;

CFirewallOpener\* m\_pFirewallOpener;//hyper added

节点加入网络：

**[emuledlg.cpp的：2087行 ]**

**Emule连接Kad网络时**，调用函数：Kademlia::CKademlia::Start(); Start()这个函数没有做什么实际意义上的事情，主要是new了几个类：

m\_pInstance = new CKademlia();

m\_pInstance->m\_pPrefs = pPrefs;

m\_pInstance->m\_pUDPListener = NULL;

m\_pInstance->m\_pRoutingZone = NULL;

m\_pInstance->m\_pIndexed = new CIndexed();

m\_pInstance->m\_pRoutingZone = new CRoutingZone();

m\_pInstance->m\_pUDPListener = new CKademliaUDPListener();

并且更改了几个定时器的时间。

**接着程序转入到routingzone.cpp中执行**。

在上面那部分的Start ()函数体内部初始化了CRoutingZone这个类，这个类的构造函数CRoutingZone::CRoutingZone()体中调用函数 Init(NULL, 0, CUInt128((ULONG)0));来初始化根节点（应该就是本地节点）。

             // Can only create routing zone after prefs

             // Set our KadID for creating the contact tree

             CKademlia ::GetPrefs ()-> GetKadID(& uMe );

             m\_sFilename = szFilename ;

             // Init our root node.

             Init (NULL , 0, CUInt128(( ULONG )0));

在void CRoutingZone::Init(CRoutingZone \*pSuper\_zone, int iLevel, const CUInt128 &uZone\_index)函数体内部创建了一个新的m\_pBin = new CRoutingBin();

             // Init all Zone vars

             // Set this zones parent

             m\_pSuperZone = pSuper\_zone ;

             // Set this zones level

             m\_uLevel = iLevel ;

             // Set this zones CUInt128 Index

             m\_uZoneIndex = uZone\_index ;

             // Mark this zone has having now leafs.

             m\_pSubZones [0] = NULL ;

             m\_pSubZones [1] = NULL ;

             // Create a new contact bin as this is a leaf.

             m\_pBin = new CRoutingBin();

             // Set timer so that zones closer to the root are processed earlier.

             m\_tNextSmallTimer = time ( NULL) + m\_uZoneIndex .Get32BitChunk (3);

             // Start this zone.

             StartTimer ();

             // If we are initializing the root node, read in our saved contact list.

             if ((m\_pSuperZone == NULL) && ( m\_sFilename .GetLength () > 0))

                         ReadFile ();

**接着调用函数StartTime（）**，用来开始这个区域。在StartTime（）函数内部添加事件CKademlia::AddEvent(this);

             time\_t tNow = time( NULL );

             // Start filling the tree, closest bins first.

             m\_tNextBigTimer = tNow + SEC(10);

             CKademlia ::AddEvent ( this);

在调用完函数StartTime（）函数后，从文件中读取以前保存的联系人。

在调用完函数Kademlia::CKademlia::Start();之后，Kademlia开始处理，转入函数**Kademlia:: CKademlia::Process()**开始执行，在函数void CKademlia::Process()中调用函数**pZone->OnSmallTimer();即CRoutingZone中 OnSmallTimer().。**

line 274:

if (pZone -> m\_tNextSmallTimer <= tNow )

                        {

                                     pZone ->OnSmallTimer ();

                                     pZone ->m\_tNextSmallTimer = MIN2S(1) + tNow ;

                        }

**CRoutingZone中OnSmallTimer()**，在此函数体内，当判断联系人为非空时，调用函数 CKademlia::GetUDPListener()->SendMyDetails\_KADEMLIA2(KADEMLIA2\_HELLO\_REQ, pContact->GetIPAddress(), pContact->GetUDPPort());来发送本地节点的一些信息，其中函数的第一个参数是消息的类型， KADEMLIA2\_HELLO\_REQ表明是Kademlia 2.0网络的加入请求，相当于TCP/IP中的ACK，即表明这个消息是用来加入网络的。第二个参数是本地节点的IP，第三个节点是本地节点的端口。

             if (pContact != NULL)

            {

                         pContact ->CheckingType ();

                         if (pContact -> GetVersion() >= 6){ /\*48b\*/

                                     if (thePrefs . GetDebugClientKadUDPLevel() > 0)

                                                 DebugSend ("KADEMLIA2\_HELLO\_REQ" , pContact ->GetIPAddress (), pContact-> GetUDPPort ());

                                     CUInt128 uClientID = pContact-> GetClientID ();

                                     CKademlia ::GetUDPListener ()-> SendMyDetails( KADEMLIA2\_HELLO\_REQ , pContact ->GetIPAddress (), pContact-> GetUDPPort (), pContact -> GetVersion(), pContact ->GetUDPKey (), & uClientID, false );

                                     if (pContact -> GetVersion() >= KADEMLIA\_VERSION8\_49b ){

                                                 // FIXME:

                                                 // This is a bit of a work arround for statistic values. Normally we only count values from incoming HELLO\_REQs for

                                                 // the firewalled statistics in order to get numbers from nodes which have us on their routing table,

                                                 // however if we send a HELLO due to the timer, the remote node won't send a HELLO\_REQ itself anymore (but

                                                 // a HELLO\_RES which we don't count), so count those statistics here. This isn't really accurate, but it should

                                                 // do fair enough. Maybe improve it later for example by putting a flag into the contact and make the answer count

                                                 CKademlia ::GetPrefs ()-> StatsIncUDPFirewalledNodes( false );

                                                 CKademlia ::GetPrefs ()-> StatsIncTCPFirewalledNodes( false );

                                    }

**接着转入**KademliaUDPListener.cpp中函数void CKademliaUDPListener::SendMyDetails\_KADEMLIA2(byte byOpcode, uint32 uIP, uint16 uUDPPort)运行，主要是调用函数SendPacket(byPacket, uLen, uIP, uUDPPort);，SendPacket(byPacket, uLen, uIP, uUDPPort);函数在KademliaUDPListener.cpp内部，此函数体内部调用函数theApp.clientudp-> SendPacket(pPacket, ntohl(uDestinationHost), uDestinationPort);来发送包。

uint32 uLen = sizeof( byPacket ) - byteIOResponse . GetAvailable();

                         if (byKadVersion >= KADEMLIA\_VERSION6\_49aBETA){

                                     if (isnulmd4 ( uCryptTargetID-> GetDataPtr ())){

                                                 DebugLogWarning (\_T ( "Sending hello response to crypt enabled Kad Node which provided an empty NodeID: %s (%u)"), ipstr (ntohl ( uIP)), byKadVersion );

                                                 SendPacket (byPacket , uLen,  uIP , uUDPPort , targetUDPKey, NULL );

                                    }

                                     else

                                                 SendPacket (byPacket , uLen,  uIP , uUDPPort , targetUDPKey, uCryptTargetID );

                        }

                         else {

                                     SendPacket (byPacket , uLen,  uIP , uUDPPort , 0, NULL);

                                     ASSERT ( targetUDPKey . IsEmpty() );

                        }

KademliaUDPListener.cpp内部CKademliaUDPListener ::SendPacket之一：

{

             if (uLenData < 2) {

                         ASSERT (0);

                         return ;

            }

             AddTrackedOutPacket (uDestinationHost , pbyData[1]);

             Packet \* pPacket = new Packet (OP\_KADEMLIAHEADER );

             pPacket ->opcode = pbyData[1];

             pPacket ->pBuffer = new char [uLenData +8];

             memcpy (pPacket -> pBuffer, pbyData +2, uLenData -2);

             pPacket ->size = uLenData-2;

             if ( uLenData > 200 )

                         pPacket ->PackPacket ();

             theStats .AddUpDataOverheadKad ( pPacket-> size );

**theApp .clientudp -> SendPacket( pPacket , ntohl ( uDestinationHost), uDestinationPort , true**

**, ( uCryptTargetID != NULL ) ? uCryptTargetID-> GetData () : NULL**

**, true , targetUDPKey . GetKeyValue( theApp .GetPublicIP ( false)));**

}

**ClientUDPSocket.cpp中（565line）函数**theApp.clientudp->SendPacket(pPacket, ntohl(uDestinationHost), uDestinationPort);体内部将刚才的消息包（或者叫数据包）加入到controlpacket\_queue的队尾，

**controlpacket\_queue.AddTail(newpending); // line586**

 controlpacket\_queue是一个链表，类型是CTypedPtrList<CPtrList, UDPPack\*> controlpacket\_queue;，

CTypedPtrList <CPtrList , UDPPack\*> controlpacket\_queue ;

// ZZ:UploadBandWithThrottler (UDP) -->

    sendLocker. Lock ();

             controlpacket\_queue .AddTail ( newpending);

    sendLocker. Unlock ();

    theApp. uploadBandwidthThrottler ->QueueForSendingControlPacket ( this);

             return true ;

// <-- ZZ:UploadBandWithThrottler (UDP)

是通过**模板**来实现的。接着继续调用函数theApp.uploadBandwidthThrottler- >QueueForSendingControlPacket(this);此时数据包在链表UploadBandwidthThrottler\* uploadBandwidthThrottler;中排队。

**类UploadBandwidthThrottler继承自CWinThread类，主要是作为线程来运行的。**

类在初始化，在构造函数中调用函数 UINT AFX\_CDECL UploadBandwidthThrottler::RunProc(LPVOID pParam)，

UploadBandwidthThrottler ::UploadBandwidthThrottler ( void) {

             m\_SentBytesSinceLastCall = 0;

             m\_SentBytesSinceLastCallOverhead = 0;

    m\_highestNumberOfFullyActivatedSlots = 0;

             threadEndedEvent = new CEvent(0, 1);

             pauseEvent = new CEvent( TRUE , TRUE );

             doRun = true ;

             AfxBeginThread (RunProc , ( LPVOID) this );

}

UINT AFX\_CDECL UploadBandwidthThrottler:: RunProc (LPVOID pParam) {

             DbgSetThreadName ("UploadBandwidthThrottler" );

             InitThreadLocale ();

             UploadBandwidthThrottler \* uploadBandwidthThrottler = ( UploadBandwidthThrottler\*) pParam ;

             return uploadBandwidthThrottler -> RunInternal();

}

这个函数调用uploadBandwidthThrottler->RunInternal();，RunInternal()函 数主要用来发送来自socket的数据包，函数体内调用两个函数：

SocketSentBytes socketSentBytes = socket->SendControlData(allowedDataRate > 0?(UINT)(bytesToSpend - spentBytes):1, minFragSize);

以及

   if( socket != NULL ) {

                    SocketSentBytes socketSentBytes = socket-> SendControlData (allowedDataRate > 0?(UINT )(bytesToSpend - spentBytes):1, minFragSize );

                                                    uint32 lastSpentBytes = socketSentBytes .sentBytesControlPackets + socketSentBytes. sentBytesStandardPackets ;

                                                    spentBytes += lastSpentBytes ;

                                                    spentOverhead += socketSentBytes . sentBytesControlPackets;

                                        }

  if( neededBytes > 0) {

                                                                            SocketSentBytes socketSentBytes = socket ->SendFileAndControlData ( neededBytes, minFragSize );

                                                                            uint32 lastSpentBytes = socketSentBytes .sentBytesControlPackets + socketSentBytes. sentBytesStandardPackets ;

                                                                            spentBytes += lastSpentBytes ;

                                                                            spentOverhead += socketSentBytes .sentBytesControlPackets ;

                            if (lastSpentBytes > 0 && slotCounter < m\_highestNumberOfFullyActivatedSlots ) {

                                m\_highestNumberOfFullyActivatedSlots = slotCounter ;

                            }

                                                                }

SocketSentBytes socketSentBytes = socket->SendFileAndControlData(neededBytes, minFragSize);

其中的socket类型是ThrottledFileSocket\*，在类ThrottledFileSocket中这两个函数被定义为虚函数，

class ThrottledFileSocket : public ThrottledControlSocket

{

public :

    virtual SocketSentBytes SendFileAndControlData ( uint32 maxNumberOfBytesToSend , uint32 minFragSize ) = 0;

    virtual DWORD GetLastCalledSend () = 0;

    virtual uint32   GetNeededBytes () = 0;

             virtual bool           IsBusy () const = 0;

    virtual bool     HasQueues () const = 0;

             virtual bool           UseBigSendBuffer ()                                                                                            { return false ; }

};

而 且在这个类内部没有具体实现，它们的实现在类CClientUDPSocket中，类CClientUDPSocket继承自**CAsyncSocket**以 及**ThrottledControlSocket**，如下代码：

class CClientUDPSocket : public CAsyncSocket, public ThrottledControlSocket // ZZ:UploadBandWithThrottler (UDP)。

socket->SendControlData(allowedDataRate > 0?(UINT)(bytesToSpend - spentBytes):1, minFragSize);

class CClientUDPSocket : public CAsyncSocket , public CEncryptedDatagramSocket, public ThrottledControlSocket // ZZ:UploadBandWithThrottler (UDP)

{

public :

             CClientUDPSocket ();

             virtual ~CClientUDPSocket ();

             bool       Create ();

             bool       Rebind ();

             uint16    GetConnectedPort ()                               { return m\_port ; }

             bool       SendPacket ( Packet\* packet , uint32 dwIP, uint16 nPort , bool bEncrypt , const uchar \* pachTargetClientHash );

    SocketSentBytes  SendControlData (uint32 maxNumberOfBytesToSend, uint32 minFragSize ); // ZZ:UploadBandWithThrottler (UDP)

protected :

以及

SocketSentBytes socketSentBytes = socket->**SendFileAndControlData**(neededBytes, minFragSize);的实现体在**ClientUDPSocket**.cpp中424行：[ps:newversion中可能没这个了]

SocketSentBytes CClientUDPSocket::**SendControlData**(uint32 maxNumberOfBytesToSend, uint32 /\*minFragSize\*/){ // ZZ:UploadBandWithThrottler (UDP)

**在它们内部调用了函数SendTo**，if (!SendTo(sendbuffer, cur\_packet->packet->size+2, cur\_packet->dwIP, cur\_packet->nPort))（在ClientUDPSocket.cpp中528行）。这个函数是类CClientUDPSocket 的成员函数。int CClientUDPSocket::SendTo(char\* lpBuf,int nBufLen,uint32 dwIP, uint16 nPort)，在这个函数体内调用类CAsyncSocket的成员函数uint32 result = CAsyncSocket::SendTo(lpBuf,nBufLen,nPort,ipstr(dwIP));，类CAsyncSocket是MFC 的类库中的一个类。【NND，终于找到头了】

  if (! SendTo ((char \*) sendbuffer, nLen , cur\_packet -> dwIP, cur\_packet ->nPort )){

                sentBytes += nLen ; // ZZ:UploadBandWithThrottler (UDP)

                                                 controlpacket\_queue .RemoveHead ();

                                                 delete cur\_packet -> packet;

                                                 delete cur\_packet ;

            }

int CClientUDPSocket :: SendTo( char \* lpBuf , int nBufLen ,uint32 dwIP, uint16 nPort ){

             // NOTE: \*\*\* This function is invoked from a \*different\* thread!

**uint32 result = CAsyncSocket:: SendTo (lpBuf , nBufLen, nPort ,ipstr ( dwIP));**

             if (result == ( uint32) SOCKET\_ERROR ){

                         uint32 error = GetLastError();

                         if (error == WSAEWOULDBLOCK){

                                     m\_bWouldBlock = true ;

                                     return -1;

                        }

                         if (thePrefs . GetVerbose())

                                     DebugLogError (\_T ( "Error: Client UDP socket, failed to send data to %s:%u: %s"), ipstr( dwIP ), nPort , GetErrorMessage( error , 1));

            }

             return 0;

}

至此，本地节点加入网络的请求就发送完毕。

* 下面讲述本地节点在接收到来自其他节点的回应后在本地采取的一些措施从而把自己加入到网络内。

**当网络事件发生时（即本地网卡接收到数据包），“socket窗口”接收WM\_SOCKET\_NOTIFY消息，消息处理函数OnSocketNotify被调用，。“socket窗口”的定义和消息处理是MFC实现的，其中OnSocketNotify函数定义如下：**

LRESULT CSocketWnd::OnSocketNotify(WPARAM wParam, LPARAM lParam)

{

CSocket::AuxQueueAdd(WM\_SOCKET\_NOTIFY, wParam, lParam);

CSocket::ProcessAuxQueue();

return 0L;

}

**在CSocket::ProcessAuxQueue();函数中回调CAsyncSocket的成员函数DoCallBack，DoCallBack调用事件处理函数OnReceive。**

int PASCAL CSocket::ProcessAuxQueue()

{

……………………//省略部分

if (pMsg->message == WM\_SOCKET\_NOTIFY)

{

CAsyncSocket::DoCallBack(pMsg->wParam, pMsg->lParam);

}

………………//省略部分

return nCount;

}

void PASCAL CAsyncSocket::DoCallBack(WPARAM wParam, LPARAM lParam)

{

……………………//省略部分

pSocket->OnReceive(nErrorCode);

**/\*pSocket类型是：CClientUDPSocket，因为类CClientUDPSocket继承了类 CAsyncSocket，而OnReceive在CAsyncSocket定义的虚函数，OnReceive在CClientUDPSocket中重新 做了实现，因此调用的时候会转到CClientUDPSocket中OnReceive执行。\*/**

}

void CClientUDPSocket::OnReceive(int nErrorCode)

{

……………………

case OP\_KADEMLIAHEADER:

{

// theStats.AddDownDataOverheadKad(length);

if (length >= 2)

Kademlia::CKademlia::ProcessPacket(buffer, length, ntohl(sockAddr.sin\_addr.S\_un.S\_addr), ntohs(sockAddr.sin\_port));

else

throw CString(\_T("Kad packet too short"));

break;

}

……………………

}

**接着调用在kademlia.cpp中定义的函数ProcessPacket。**

void CKademlia::ProcessPacket(const byte \*pbyData, uint32 uLenData, uint32 uIP, uint16 uPort)

{

if( m\_pInstance && m\_pInstance->m\_pUDPListener )

m\_pInstance->m\_pUDPListener->ProcessPacket( pbyData, uLenData, uIP, uPort);

}

**转入KademliaUDPListener类中ProcessPacket函数运行。**

void CKademliaUDPListener::ProcessPacket(const byte\* pbyData, uint32 uLenData, uint32 uIP, uint16 uUDPPort)

{

//………………………………省略部分

switch (byOpcode)

{

………………………………//省略部分

case KADEMLIA\_RES:

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

DebugRecv("KADEMLIA\_RES", uIP, uUDPPort);

Process\_KADEMLIA\_RES(pbyPacketData, uLenPacket, uIP, uUDPPort);

break;

………………………………//省略部分

}

}

**转入函数Process\_KADEMLIA\_RES(pbyPacketData, uLenPacket, uIP, uUDPPort);执行：**

**void CKademliaUDPListener::Process\_KADEMLIA\_RES (const byte \*pbyPacketData, uint32 uLenPacket, uint32 uIP, uint16 uUDPPort) 【我拦截它就ok了】**

{

//……………………

if(CKademlia::GetPrefs()->GetRecheckIP())

{

FirewalledCheck(uIP, uUDPPort);

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

DebugSend("KADEMLIA\_HELLO\_REQ", uIP, uUDPPort);

SendMyDetails(KADEMLIA\_HELLO\_REQ, uIP, uUDPPort);

}

if(::IsGoodIPPort(ntohl(uIPResult),uUDPPortResult))

{

pRoutingZone->Add(uIDResult, uIPResult, uUDPPortResult, uTCPPortResult, 0);

pResults->push\_back(new CContact(uIDResult, uIPResult, uUDPPortResult, uTCPPortResult, uTarget, 0));

}

}

}

CSearchManager::ProcessResponse(uTarget, uIP, uUDPPort, pResults);

}

在这个函数体内部主要包括对4个函数的调用，分别是：

SendMyDetails(KADEMLIA\_HELLO\_REQ, uIP, uUDPPort);

pRoutingZone->Add(uIDResult, uIPResult, uUDPPortResult, uTCPPortResult, 0);

pResults->push\_back(new CContact(uIDResult, uIPResult, uUDPPortResult, uTCPPortResult, uTarget, 0));

CSearchManager::ProcessResponse(uTarget, uIP, uUDPPort, pResults);

其中第一个函数是在判断自己在防火墙或者NAT之后重新发送本地节点信息的函数，包括重新得到的IP地址以及端口。

第二和第三个函数用来添加此节点作为联系人之一。

**第三个函数是将此消息转入到CSearchManager中相应处理响应的函数进行处理。**

void CSearchManager::ProcessResponse(const CUInt128 &uTarget, uint32 uFromIP, uint16 uFromPort, ContactList \*plistResults)

{

pSearch->ProcessResponse(uFromIP, uFromPort, plistResults);// pSearch是 CSearch类的指针

}

**进一步转入到pSearch->ProcessResponse(uFromIP, uFromPort, plistResults)中执行。**

void CSearch::ProcessResponse(uint32 uFromIP, uint16 uFromPort, ContactList \*plistResults)

{

// Not interested in responses for FIND\_NODE.

// Once we get a results we stop the search.

// These contacts are added to contacts by UDPListener.

if (m\_uType == NODE)

{

// Note we got an answer

**m\_uAnswers++;**

// We clear the possible list to force the search to stop.

// We do this so the user has time to visually see the results.

m\_mapPossible.clear();

delete plistResults;

// Update search on the GUI.

//IMPREVIEW theApp.emuledlg->kademliawnd->searchList->SearchRef(this);

return;

}

}

在这个函数内部我们将响应的节点数目增加一。

后面陆续接收到的消息处理流程与上述情形相似，只是对于不同的消息采取的响应以及动作并不相同。