# 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（DH密钥交换）

### (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

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

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

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

///snow:MAGICVALUE\_SYNC 835E6FC4

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

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

///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

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

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

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

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

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

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

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

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

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

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

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

///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

///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

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:

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

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

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

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

/// 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

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（USERHASH）

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

09:41:44 snow:CAsyncSocketExHelperWindows::WindowProc FD\_ACCEPT

09:41:44 snow:UploadBandwidthThrottler::QueueForSendingControlPacket m\_TempControlQueue\_list size:1

09:41:44 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

09:41:44 CEMSocket:OnReceive start

09:41:44 CEMSocket:Receive start

09:41:44 snow:CEncryptedStreamSocket:Receive start

09:41:44 snow:CEncryptedStreamSocket:Receive before DeCrypt size:104 content:34E3FBFBFE D7C93DC6 D5B019 ED7A451E353D96B3352D47864F9FB7C4998ECAC12A39F460647D1864F5C90DD608DD14AF64CC86FD954216B5995885E516C9E4FA01DB04A2D12EB59D962A0636E65BC24CDF396293BFDCAB07EBA6A22761A5D14DB66F291FC7FFFD8B

09:41:44 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_CLIENTA\_RANDOMPART

09:41:44 snow:CEncryptedStreamSocket:Negotiate before decrypt size:4 content:D7C93DC6

09:41:44 snow:CEncryptedStreamSocket:Negotiate after decrypt size:4 content:C46F5E83

09:41:44 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_CLIENTA\_MAGICVALUE

09:41:44 snow:CEncryptedStreamSocket:Negotiate before decrypt size:3 content:D5B019

09:41:44 snow:CEncryptedStreamSocket:Negotiate after decrypt size:3 content:00005C

09:41:44 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_CLIENTA\_METHODTAGSPADLEN

09:41:44 snow:CEncryptedStreamSocket:Negotiate before decrypt size:92 content:ED7A451E353D96B3352D47864F9FB7C4998ECAC12A39F460647D1864F5C90DD608DD14AF64CC86FD954216B5995885E516C9E4FA01DB04A2D12EB59D962A0636E65BC24CDF396293BFDCAB07EBA6A22761A5D14DB66F291FC7FFFD8B

///snow:这92个字节并没有用，是无用的随机数

09:41:44 snow:CEncryptedStreamSocket:Negotiate after decrypt size:92 content:20F2C93FA84BCF625FF623AC37BCB1DF8CF9BBF697D109C542D482E3A2F5CC3FE8DEE1B937E1F6870BF2B6302EDA600D9F5DA972544352760FFEF1D97FAF737E1B7F6F7F3AC8AC2F9AA4EE976717A04D066EC9A0C74786B1B1C47330

09:41:44 snow:CEncryptedStreamSocket:Negotiate ONS\_BASIC\_CLIENTA\_PADDING

///snow:HandShake ClientB报文

09:41:44 snow:CEncryptedStreamSocket:SendNegotiatingData start

09:41:44 snow:CEncryptedStreamSocket:SendNegotiatingData before crypt size:32 content:C46F5E83001AA5D91543C21CBC4136BC2B4F3D55979CDB285402157AA4896037

///snow: <MagicValue 4> C46F5E83<EncryptionMethodsSelected 1>00<PaddingLen 1>1A(26) <RandomBytes PaddingLen%max 256> A5D91543 C21CBC41 36BC2B4F 3D55979C DB285402 157AA489 6037

///snow:有用的是前6个字节，后26个字节并没有用

09:41:44 snow:CEncryptedStreamSocket:SendNegotiatingData after crypt size:32 content:2396B71DC7991EF5589CF7BDB4518ECCA86F0A7A5880055480E8C361551BD108

09:41:44 snow:CAsyncSocketEx::Send before send,size : 32 , content : 2396B71DC7991EF5589CF7BDB4518ECCA86F0A7A5880055480E8C361551BD108

09:41:44 snow:CAsyncSocketEx::Send，after send

09:41:44 snow:CEncryptedStreamSocket:Receive ECS\_UNKNOWN if !bNormalHeader

09:41:45 OP\_Hello from 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

09:41:45 Hash=B2C02E84100ED1A8DA8B732310076F5F (eMule) UserID=701042767 (79.16.201.41) Port=7869 Tags=6

Name='http://emule-project.net'

Version=60

KadPort=7879 UDPPort=7879

PeerCache=1 UDPVer=4 DataComp=1 SecIdent=3 SrcExchg=4 ExtReq=2 Commnt=1 Preview=0 NoViewFiles=1 Unicode=1

KadVersion=9, LargeFiles=1 ExtMultiPacket=1 CryptLayerSupport=1 CryptLayerRequest=1 CryptLayerRequires=0 SupportsSourceEx2=1 SupportsCaptcha=1 DirectUDPCallback=0

ClientVer=0.50.0.0 Comptbl=0

Server=85.204.50.116:4232

09:41:45 >>> OP\_\_HelloAnswer to 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

09:41:45 >>> OP\_\_SecIdentState to 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

////snow:OP\_HELLO报文

09:41:45 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

09:41:45 CEMSocket:OnReceive start

09:41:45 CEMSocket:Receive start

09:41:45 snow:CEncryptedStreamSocket:Receive start

09:41:45 snow:CEncryptedStreamSocket:Receive before DeCrypt size:109 content:38F8B02C80601D8B65DBD86E8EC004875AB0F0AE9A0E91B0053A4B17936C27F0AE2DE933173E15BE892A66130139611F00C2301FBB435FEEA5C9902CEEA08C57CEEA4ECF9406467B4D411F97F706F58533EF60F4AECF3048F2BAFE4CF2A4FC62258444B29AD0EDCC8C03491D88

09:41:45 snow:CEncryptedStreamSocket:Receive ECS\_ENCRYPTING

09:41:45 snow:CEncryptedStreamSocket:Receive after DeCrypt size:109 content:E3680000000110B2C02E84100ED1A8DA8B732310076F5F4F10C929BD1E06000000020100011800687474703A2F2F656D756C652D70726F6A6563742E6E6574030100113C000000030100F9C71EC71E030100FA1E421334030100FEB92D0000030100FB00C8000055CC32748810

///snow:Hello报文内容：

///snow:

////snow:OP\_HELLOANSWER报文

09:41:45 snow:CEncryptedStreamSocket:CryptPrepareSendData before Crypt size:108 content:E3670000004C9C431EB71E0E142BF1BCF52D41C16FB070053C043ED806000000020100011800687474703A2F2F656D756C652D70726F6A6563742E6E6574030100113C000000030100F97C637C63030100FA1E421334030100FEB92D0000030100FB00C80000B06730245810

09:41:45 snow:CEncryptedStreamSocket:CryptPrepareSendData after Crypt size:108 content:7D1E106B8412E6E5C625817D14DFC2EA01C56BD80594ABD49F6901500B8808C9817F3B24F8D2F70150DE27E7F32C094E7EE5C7FF1E4A5CBD1F909C921DA462BE9379A8499AADBB6623EBFE3C4FB16B2C75C9F3979416AEE3DBD0C2BFEE0A614E176AAE43C38A5760E85A85CA

09:41:45 CEncryptedStreamSocket:Send start

09:41:45 snow:CEncryptedStreamSocket:Send end

09:41:45 snow:CAsyncSocketEx::Send before send,size : 108 , content : 7D1E106B8412E6E5C625817D14DFC2EA01C56BD80594ABD49F6901500B8808C9817F3B24F8D2F70150DE27E7F32C094E7EE5C7FF1E4A5CBD1F909C921DA462BE9379A8499AADBB6623EBFE3C4FB16B2C75C9F3979416AEE3DBD0C2BFEE0A614E176AAE43C38A5760E85A85CA

09:41:45 snow:CAsyncSocketEx::Send，after send

///snow: OP\_SecIdentState报文

09:41:45 snow:CEncryptedStreamSocket:CryptPrepareSendData before Crypt size:11 content:C50600000087027D490000

09:41:45 snow:CEncryptedStreamSocket:CryptPrepareSendData after Crypt size:11 content:6C2BCA9A5B79612F723129

09:41:45 CEncryptedStreamSocket:Send start

09:41:45 snow:CEncryptedStreamSocket:Send end

09:41:45 snow:CAsyncSocketEx::Send before send,size : 11 , content : 6C2BCA9A5B79612F723129

09:41:45 snow:CAsyncSocketEx::Send，after send

09:41:46 OP\_SecIdentState from 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

09:41:46 >>> OP\_PublicKey to 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

09:41:46 OP\_PublicKey from 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

09:41:46 >>> OP\_Signature to 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

///snow: Receive OP\_SECIDENTSTATE

09:41:46 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

09:41:46 CEMSocket:OnReceive start

09:41:46 CEMSocket:Receive start

09:41:46 snow:CEncryptedStreamSocket:Receive start

09:41:46 snow:CEncryptedStreamSocket:Receive before DeCrypt size:94 content:460B04D94229E524C0A24F598A84440CA201BCCCAE00AA0FF7DF94F6910ED00604ADB662A30D93730B8A7FFCE152F510C7D2A87B218B6F0723346AAA4239EA7A6D3D709936D9BEB3CFAA7E1B0F5B71C74611AAA2C7DFEAE5651DB016F04C

09:41:46 snow:CEncryptedStreamSocket:Receive ECS\_ENCRYPTING

09:41:46 snow:CEncryptedStreamSocket:Receive after DeCrypt size:94 content:C50600000087022E090000C54E000000854C304A300D06092A864886F70D01010105000339003036023100AFA55174DFEDB3A958443242C2F97FF24307750A7467CE613F03DBEA6213F2B0A09AB6702E57ED91A8A6D7CA82C4A5C1020111

///snow: send OP\_PUBLICKEY

09:41:46 snow:CEMSocket::SendPacket controlpacket\_queue size:1

09:41:46 snow:CEMSocket::SendPacket QueueForSendingControlPacket socket:1456

09:41:46 snow:UploadBandwidthThrottler::QueueForSendingControlPacket m\_TempControlQueueFirst\_list size:1

09:41:46 snow:CEMSocket:Send start

09:41:46 snow:CEncryptedStreamSocket:CryptPrepareSendData before Crypt size:83 content:C54E000000854C304A300D06092A864886F70D01010105000339003036023100A82A2C77C430E0FD525FF8616133940D90F4B50371AEF6177D757F1CB780484A5813140B67B450B1E549F8C33E4ECC6D020111

09:41:46 snow:CEncryptedStreamSocket:CryptPrepareSendData after Crypt size:83 content:AFFFF6BC0744C65680EE2D6A944BBC0B390AFB9625A3EC21110DAE8EA29558993052658164488D9D6904A8F93B26F722C47B15E31EA7DA0CD8ABA271EBBFE26D1763E6EFDE68FF926BE2360484A237E72DA345

09:41:46 CEncryptedStreamSocket:Send start

09:41:46 snow:CEncryptedStreamSocket:Send end

09:41:46 snow:CAsyncSocketEx::Send before send,size : 83 , content : AFFFF6BC0744C65680EE2D6A944BBC0B390AFB9625A3EC21110DAE8EA29558993052658164488D9D6904A8F93B26F722C47B15E31EA7DA0CD8ABA271EBBFE26D1763E6EFDE68FF926BE2360484A237E72DA345

///snow: send OP\_SIGNATURE

09:41:46 snow:CEMSocket:Send start

09:41:46 snow:CEncryptedStreamSocket:CryptPrepareSendData before Crypt size:55 content:C532000000863085ADD5AD6774AF16B4528CB12020F49E0F8A190E9B9EEF34487A24166E3C7E5FD0392EF636EB692ADFA76191167FF41E

09:41:46 snow:CEncryptedStreamSocket:CryptPrepareSendData after Crypt size:55 content:345A401F906E078F69FD97E2AAA4AB0D3053F3D2B0A82B6873CF058C6080ADD0F78A25A8CABFB4DAD91F6B1662A5387AA068A9515DAA43

09:41:46 CEncryptedStreamSocket:Send start

09:41:46 snow:CEncryptedStreamSocket:Send end

09:41:46 snow:CAsyncSocketEx::Send before send,size : 55 , content : 345A401F906E078F69FD97E2AAA4AB0D3053F3D2B0A82B6873CF058C6080ADD0F78A25A8CABFB4DAD91F6B1662A5387AA068A9515DAA43

09:41:46 snow:CAsyncSocketEx::Send，after send

09:41:48 OP\_KAD\_FWTCPCHECK\_ACK from 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

09:41:48 OP\_Signature from 79.16.201.41 'http://emule-project.net' (eMule v0.50a,None/None/None)

///snow:Receive OP\_KAD\_FWTCPCHECK\_ACK && OP\_Signature 报文

09:41:48 snow:CAsyncSocketExHelperWindows::WindowProc FD\_READ

09:41:48 CEMSocket:OnReceive start

09:41:48 CEMSocket:Receive start

09:41:48 snow:CEncryptedStreamSocket:Receive start

09:41:48 snow:CEncryptedStreamSocket:Receive before DeCrypt size:61 content:742E84C6007D50560078D096CD4E6D712EC9AC334EFA1AF447330BADC6DF10C4E2A67029B5F147249EC6B97DDD9C9B8EBBDDF9F49461635AE8D7290E0A铪

09:41:48 snow:CEncryptedStreamSocket:Receive ECS\_ENCRYPTING

09:41:48 snow:CEncryptedStreamSocket:Receive after DeCrypt size:61 content:C501000000A8C532000000863024C454FDC435A8B72B35C6F021AAC2F88936ACBC046E65EA3E6699815B60CC9771D009C513887E195753F360ACF58090铪

09:41:48 snow:CEMSocket:Send start

09:41:48 CEncryptedStreamSocket:Send start

09:41:48 snow:CEncryptedStreamSocket:Send end

09:41:48 snow:CAsyncSocketEx::Send before send,size : 55 , content : C53200000086302D17C781663EB3E986F8DB23D0E465A943033C2FDD94FF4174A94F905EB955A09027B00FB469F0579819A8A9A6321F52铪铪铪

## 五、客户端呼出连接过程(USERHASH)

一、几种情况下启动呼出连接：

CUpDownClient::RequestSharedFileList() 请求共享文件列表

CUpDownClient::SafeConnectAndSendPacket 安全连接

CClientList::Process 客户端列表处理时

case KS\_QUEUED\_FWCHECK:

case KS\_QUEUED\_FWCHECK\_UDP:

case KS\_QUEUED\_BUDDY:

CClientUDPSocket::ProcessPacket case OP\_DIRECTCALLBACKREQ: 直接回呼请求

CUpDownClient::AskForDownload 下载文件

CFriend::TryToConnect

CFriend::KadSearchIPByNodeIDResult

CClientReqSocket::ProcessExtPacket case OP\_CALLBACK: 请求回呼

CServerSocket::ProcessPacket case OP\_CALLBACKREQUESTED:服务器回呼请求

CUploadQueue::AddUpNextClient 上传文件

CUrlClient::TryToConnect

1、TryToConnect()在客户端具有高ID的情况下，可以直接发起连接，调用Connect()，如果设置了加密连接，并且对方客户端支持加密连接，就发起加密连接，SetConnectionEncryption(true, GetUserHash(), false)，如果不支持，SetConnectionEncryption(false, NULL, false)，然后调用CEMSocked::Connect()发起连接；

2、CAsyncSocketExHelperWindows::WindowProc FD\_CONNECT事件触发，调用CClientReqSocket::OnConnect ()🡪CEMSocket::OnConnect()启动，

3、对方客户端响应Connect请求，Socket接通，FD\_WRITE事件触发，顺序调用CClientReqSocket::OnSend() 🡪CEMSocket::OnSend() 🡪CEncryptedStreamSocket:: OnSend()

4、CEncryptedStreamSocket::OnSend()根据m\_StreamCryptState状态：启动加密联接（m\_StreamCryptState == ECS\_PENDING，StartNegotiation(true)）

如果是不加密连接，则OnSend()啥都不做。

（1）加密连接：

StartNegotiation()执行if(m\_StreamCryptState == ECS\_PENDING){

准备ClientA报文

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

m\_StreamCryptState = ECS\_NEGOTIATING;

m\_nReceiveBytesWanted = 4;

SendNegotiatingData(fileRequest.GetBuffer(), (uint32)fileRequest.GetLength(), 5);

}

5、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)

{

case ONS\_BASIC\_CLIENTB\_MAGICVALUE:{

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

m\_nReceiveBytesWanted = 2;

case ONS\_BASIC\_CLIENTB\_METHODTAGSPADLEN:

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

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

case ONS\_BASIC\_CLIENTB\_PADDING:

m\_NegotiatingState = ONS\_COMPLETE;

m\_StreamCryptState = ECS\_ENCRYPTING;

}

协商完毕后，加密连接已经建立。

6、Connect()调用SendHelloPacket()发出OP\_HELLO信息包。

7，FD\_READ事件触发，顺序调用CClientReqSocket::OnReceive() 🡪CEMSocket::OnReceive() 🡪Receive() 🡪CEncryptedStreamSocket::Receive()，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\_HELLOANSWER语句块中，

client->ProcessHelloAnswer(packet,size);

client->InfoPacketsReceived();

client->ConnectionEstablished();

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)

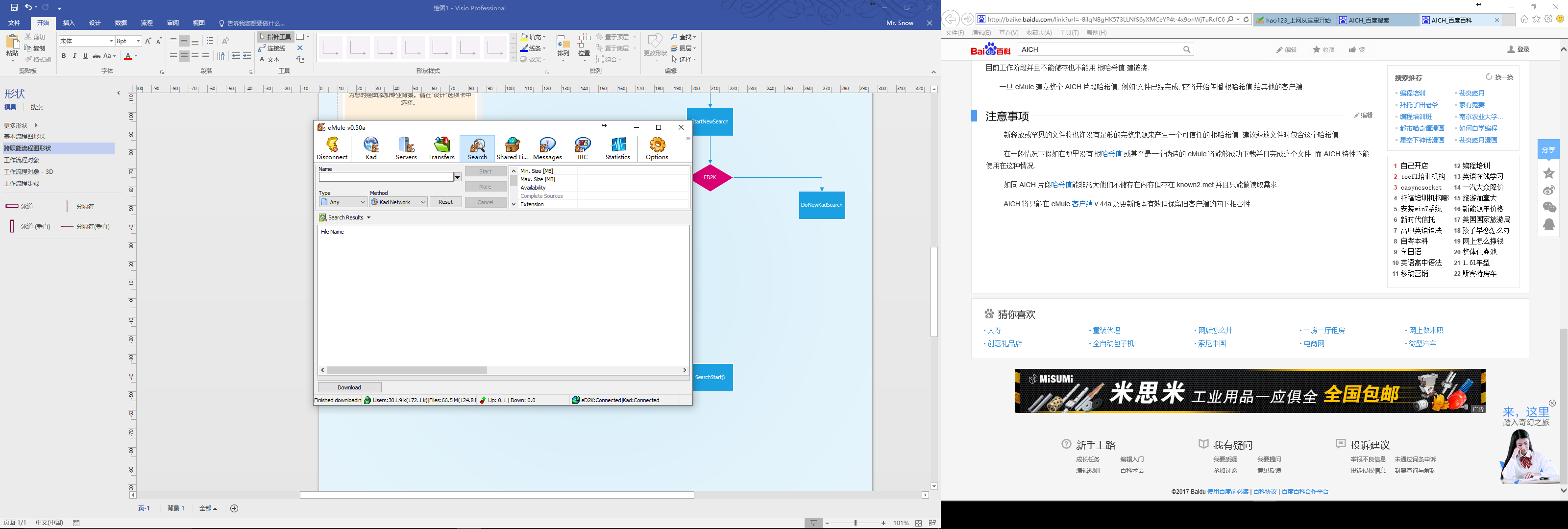
#define OP\_REASKCALLBACKUDP 0x94

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

#define OP\_PORTTEST 0xFE // Connection Test

# 文件搜索 SearchFile

## 一、搜索参数



SearchDlg包含两部分：CsearchParamsWnd和CsearchResultsWnd。当start按钮点击的时候，CsearchParamsWnd::OnBnClickStart()触发，调用CsearchResultsWnd::StartSearch()🡪StartNewSearch()。

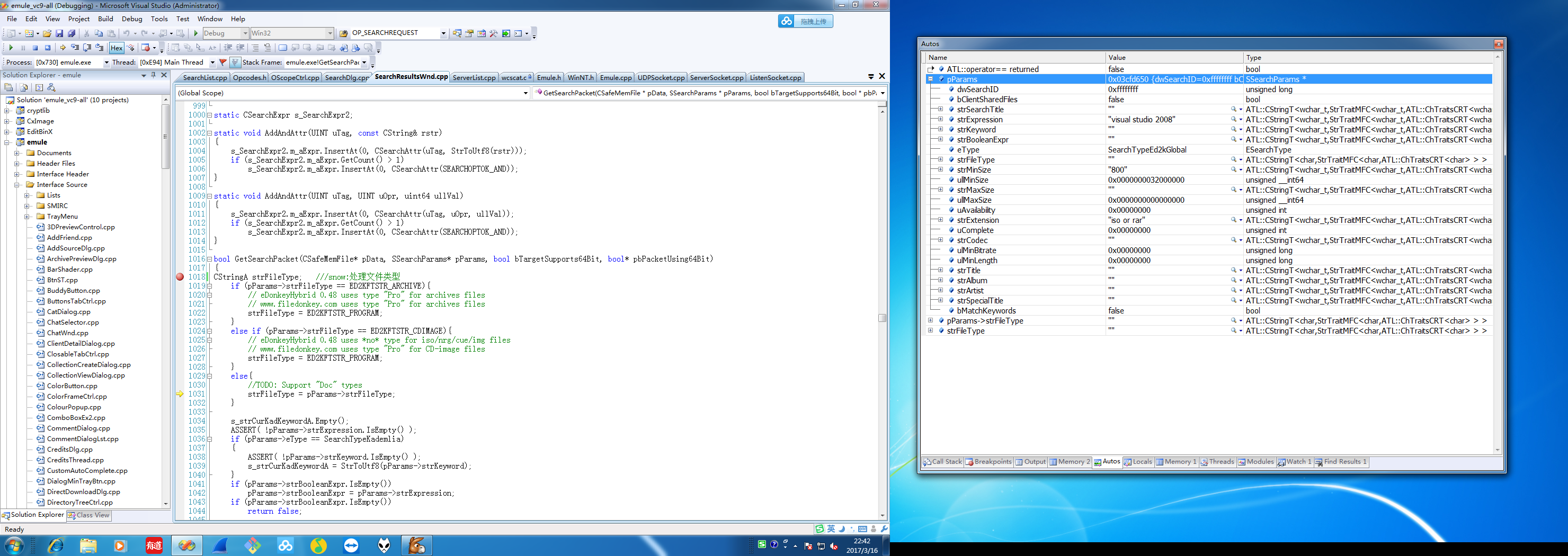
判断Method是Kad还是ED2K，分别调用DoNewED2Ksearch()和DoNewKadSearch()，然后再调用SearchStarted()。

搜索主要分三种：Server、GlobalServer、Kad

Server：向当前连接的服务器发出TCP搜索请求OP\_SEARCHREQUEST，返回OP\_SEARCHRESULT

GlobalServer: 向当前连接的服务器发出TCP搜索请求，向服务器列表中的其它服务器发出UDP搜索请求

（OP\_GLOBSEARCHREQ、 OP\_GLOBSEARCHREQ2、 OP\_GLOBSEARCHREQ3）

Kad：向Kad节点发出UDP搜索请求（见Kademlia搜索部分）

### 1、struct SSearchParams

struct SSearchParams

{

SSearchParams()；

SSearchParams(CFileDataIO& rFile)；

DWORD dwSearchID;

bool bClientSharedFiles;

CString strSearchTitle;

CString strExpression;

CString strKeyword;

CString strBooleanExpr;

ESearchType eType;

CStringA strFileType;

CString strMinSize;

uint64 ullMinSize;

CString strMaxSize;

uint64 ullMaxSize;

UINT uAvailability;

CString strExtension;

UINT uComplete;

CString strCodec;

ULONG ulMinBitrate;

ULONG ulMinLength;

CString strTitle;

CString strAlbum;

CString strArtist;

CString strSpecialTitle;

bool bMatchKeywords;

void StorePartially(CFileDataIO& rFile) const；

};

### 2、GetParameters

SSearchParams\* CSearchParamsWnd::GetParameters()

{

CString strExpression;

m\_ctlName.GetWindowText(strExpression);

strExpression.Trim();

if (!IsValidEd2kString(strExpression)){

AfxMessageBox(GetResString(IDS\_SEARCH\_EXPRERROR) + \_T("\n\n") + GetResString(IDS\_SEARCH\_INVALIDCHAR), MB\_ICONWARNING | MB\_HELP, eMule\_FAQ\_Search - HID\_BASE\_PROMPT);

return NULL;

}

CStringA strFileType;

int iItem = m\_ctlFileType.GetCurSel();

if (iItem != CB\_ERR)

{

LPCSTR pszED2KFileType = (LPCSTR)m\_ctlFileType.GetItemDataPtr(iItem);

ASSERT( pszED2KFileType != NULL );

strFileType = pszED2KFileType;

}

CString strMinSize = m\_ctlOpts.GetItemText(orMinSize, 1);

uint64 ullMinSize = GetSearchAttrSize(strMinSize);

if (ullMinSize == (uint64)-1) {

CString strError;

strError.Format(GetResString(IDS\_SEARCH\_ATTRERR), m\_ctlOpts.GetItemText(orMinSize, 0));

AfxMessageBox(GetResString(IDS\_SEARCH\_EXPRERROR) + \_T("\n\n") + strError, MB\_ICONWARNING | MB\_HELP, eMule\_FAQ\_Search - HID\_BASE\_PROMPT);

return NULL;

}

CString strMaxSize = m\_ctlOpts.GetItemText(orMaxSize, 1);

uint64 ullMaxSize = GetSearchAttrSize(strMaxSize);

if (ullMaxSize == (uint64)-1) {

CString strError;

strError.Format(GetResString(IDS\_SEARCH\_ATTRERR), m\_ctlOpts.GetItemText(orMaxSize, 0));

AfxMessageBox(GetResString(IDS\_SEARCH\_EXPRERROR) + \_T("\n\n") + strError, MB\_ICONWARNING | MB\_HELP, eMule\_FAQ\_Search - HID\_BASE\_PROMPT);

return NULL;

}

if (ullMaxSize < ullMinSize){

ullMaxSize = 0; // TODO: Create a message box for that

m\_ctlOpts.SetItemText(orMaxSize, 1, \_T(""));

}

CString strExtension;

if ((m\_ctlOpts.GetItemData(orExtension) & 1) == 0)

{

strExtension = m\_ctlOpts.GetItemText(orExtension, 1);

strExtension.Trim();

if (!strExtension.IsEmpty() && strExtension[0] == \_T('.'))

{

strExtension = strExtension.Mid(1);

m\_ctlOpts.SetItemText(orExtension, 1, strExtension);

}

}

UINT uAvailability = 0;

if ((m\_ctlOpts.GetItemData(orAvailability) & 1) == 0)

{

CString strAvailability = m\_ctlOpts.GetItemText(orAvailability, 1);

uAvailability = GetSearchAttrNumber(strAvailability);

if (uAvailability == (UINT)-1)

{

CString strError;

strError.Format(GetResString(IDS\_SEARCH\_ATTRERR), m\_ctlOpts.GetItemText(orAvailability, 0));

AfxMessageBox(GetResString(IDS\_SEARCH\_EXPRERROR) + \_T("\n\n") + strError, MB\_ICONWARNING | MB\_HELP, eMule\_FAQ\_Search - HID\_BASE\_PROMPT);

return NULL;

}

else if (uAvailability > 1000000)

{

uAvailability = 1000000;

strAvailability.Format(\_T("%u"), uAvailability);

m\_ctlOpts.SetItemText(orAvailability, 1, strAvailability);

}

}

UINT uComplete = 0;

if ((m\_ctlOpts.GetItemData(orCompleteSources) & 1) == 0)

{

CString strComplete = m\_ctlOpts.GetItemText(orCompleteSources, 1);

uComplete = GetSearchAttrNumber(strComplete);

if (uComplete == (UINT)-1)

{

CString strError;

strError.Format(GetResString(IDS\_SEARCH\_ATTRERR), m\_ctlOpts.GetItemText(orCompleteSources, 0));

AfxMessageBox(GetResString(IDS\_SEARCH\_EXPRERROR) + \_T("\n\n") + strError, MB\_ICONWARNING | MB\_HELP, eMule\_FAQ\_Search - HID\_BASE\_PROMPT);

return NULL;

}

else if (uComplete > 1000000)

{

uComplete = 1000000;

strComplete.Format(\_T("%u"), uComplete);

m\_ctlOpts.SetItemText(orCompleteSources, 1, strComplete);

}

}

CString strCodec;

if ((m\_ctlOpts.GetItemData(orCodec) & 1) == 0)

strCodec = m\_ctlOpts.GetItemText(orCodec, 1);

strCodec.Trim();

ULONG ulMinBitrate = 0;

if ((m\_ctlOpts.GetItemData(orBitrate) & 1) == 0)

{

CString strMinBitrate = m\_ctlOpts.GetItemText(orBitrate, 1);

ulMinBitrate = GetSearchAttrNumber(strMinBitrate);

if (ulMinBitrate == (ULONG)-1)

{

CString strError;

strError.Format(GetResString(IDS\_SEARCH\_ATTRERR), m\_ctlOpts.GetItemText(orBitrate, 0));

AfxMessageBox(GetResString(IDS\_SEARCH\_EXPRERROR) + \_T("\n\n") + strError, MB\_ICONWARNING | MB\_HELP, eMule\_FAQ\_Search - HID\_BASE\_PROMPT);

return NULL;

}

else if (ulMinBitrate > 1000000)

{

ulMinBitrate = 1000000;

strMinBitrate.Format(\_T("%u"), ulMinBitrate);

m\_ctlOpts.SetItemText(orBitrate, 1, strMinBitrate);

}

}

ULONG ulMinLength = 0;

if ((m\_ctlOpts.GetItemData(orLength) & 1) == 0)

{

CString strMinLength = m\_ctlOpts.GetItemText(orLength, 1);

ulMinLength = GetSearchAttrLength(strMinLength);

if (ulMinLength == (ULONG)-1)

{

CString strError;

strError.Format(GetResString(IDS\_SEARCH\_ATTRERR), m\_ctlOpts.GetItemText(orLength, 0));

AfxMessageBox(GetResString(IDS\_SEARCH\_EXPRERROR) + \_T("\n\n") + strError, MB\_ICONWARNING | MB\_HELP, eMule\_FAQ\_Search - HID\_BASE\_PROMPT);

return NULL;

}

else if (ulMinLength > 3600\*24)

{

ulMinLength = 3600\*24;

CString strValue;

SecToTimeLength(ulMinLength, strValue);

m\_ctlOpts.SetItemText(orLength, 1, strValue);

}

}

SSearchParams\* pParams = new SSearchParams;

pParams->strExpression = strExpression;

pParams->eType = (ESearchType)m\_ctlMethod.GetCurSel();

pParams->strFileType = strFileType;

pParams->strMinSize = strMinSize;

pParams->ullMinSize = ullMinSize;

pParams->strMaxSize = strMaxSize;

pParams->ullMaxSize = ullMaxSize;

pParams->uAvailability = uAvailability;

pParams->strExtension = strExtension;

//pParams->bMatchKeywords = IsDlgButtonChecked(IDC\_MATCH\_KEYWORDS)!=0;

pParams->uComplete = uComplete;

pParams->strCodec = strCodec;

pParams->ulMinBitrate = ulMinBitrate;

pParams->ulMinLength = ulMinLength;

if ((m\_ctlOpts.GetItemData(orTitle) & 1) == 0)

{

pParams->strTitle = m\_ctlOpts.GetItemText(orTitle, 1);

pParams->strTitle.Trim();

}

if ((m\_ctlOpts.GetItemData(orAlbum) & 1) == 0)

{

pParams->strAlbum = m\_ctlOpts.GetItemText(orAlbum, 1);

pParams->strAlbum.Trim();

}

if ((m\_ctlOpts.GetItemData(orArtist) & 1) == 0)

{

pParams->strArtist = m\_ctlOpts.GetItemText(orArtist, 1);

pParams->strArtist.Trim();

}

return pParams;

}

## 二、流程

### 1、OnBnClickedStart

void CSearchParamsWnd::OnBnClickedStart()

{

m\_ctlMore.EnableWindow(FALSE);

if (m\_ctlOpts.GetEditCtrl()->GetSafeHwnd())

m\_ctlOpts.CommitEditCtrl();

SSearchParams\* pParams = GetParameters();

if (pParams)

{

if (!pParams->strExpression.IsEmpty())

{

if (m\_pacSearchString && m\_pacSearchString->IsBound())

m\_pacSearchString->AddItem(pParams->strExpression, 0);

m\_searchdlg->StartSearch(pParams);

}

else

delete pParams;

}

}

### 2、StartSearch

void CSearchResultsWnd::StartSearch(SSearchParams\* pParams)

{

switch (pParams->eType)

{

case SearchTypeAutomatic:

case SearchTypeEd2kServer:

case SearchTypeEd2kGlobal:

case SearchTypeKademlia:

StartNewSearch(pParams);

break;

case SearchTypeFileDonkey:

ShellOpenFile(CreateWebQuery(pParams));

delete pParams;

return;

default:

ASSERT(0);

delete pParams;

}

}

### 3、StartNewSearch

bool CSearchResultsWnd::StartNewSearch(SSearchParams\* pParams)

{

if (pParams->eType == SearchTypeAutomatic){ ///snow:自动情况下的处理

// select between kad and server

// its easy if we are connected to one network only anyway

if (!theApp.serverconnect->IsConnected() && Kademlia::CKademlia::IsRunning() && Kademlia::CKademlia::IsConnected()) ///snow:Kad连接，server未连接

pParams->eType = SearchTypeKademlia;

else if (theApp.serverconnect->IsConnected() && (!Kademlia::CKademlia::IsRunning() || !Kademlia::CKademlia::IsConnected())) ///snow:server连接,Kad未连接

pParams->eType = SearchTypeEd2kServer;

else if (!theApp.serverconnect->IsConnected() && (!Kademlia::CKademlia::IsRunning() || !Kademlia::CKademlia::IsConnected())){ ///snow:都未连接

AfxMessageBox(GetResString(IDS\_NOTCONNECTEDANY), MB\_ICONWARNING);

delete pParams;

return false;

}

else { ///snow:KAD和SERVER两者都连接，当连接的服务器是静态的，或者服务器连接的客户端超过万少于万，文件数大于万，服务器列表的服务器少于个，使用server搜索；否则，进行KAD搜索

// connected to both

// We choose Kad, except

// - if we are connected to a static server

// - or a server with more than 40k and less than 2mio users connected, more than 5 mio files and if our serverlist contains less than

// 40 servers (otherwise we have assume that its polluted with fake server and we might just as well be connected to one)

// might be further optmized in the future

if (theApp.serverconnect->IsConnected() && theApp.serverconnect->GetCurrentServer() != NULL

&& (theApp.serverconnect->GetCurrentServer()->IsStaticMember()

|| (theApp.serverconnect->GetCurrentServer()->GetUsers() > 40000 && theApp.serverlist->GetServerCount() < 40

&& theApp.serverconnect->GetCurrentServer()->GetUsers() < 5000000

&& theApp.serverconnect->GetCurrentServer()->GetFiles() > 5000000)))

{

pParams->eType = SearchTypeEd2kServer;

}

else

pParams->eType = SearchTypeKademlia;

}

}

ESearchType eSearchType = pParams->eType;

if (eSearchType == SearchTypeEd2kServer || eSearchType == SearchTypeEd2kGlobal) ///snow:Ed2k服务器搜索或全局服务器搜索

{

if (!theApp.serverconnect->IsConnected()) {

AfxMessageBox(GetResString(IDS\_ERR\_NOTCONNECTED), MB\_ICONWARNING);

delete pParams;

//if (!theApp.serverconnect->IsConnecting() && !theApp.serverconnect->IsConnected())

// theApp.serverconnect->ConnectToAnyServer();

return false;

}

try

{

if (!DoNewEd2kSearch(pParams)) { ///snow:启动server搜索

delete pParams;

return false;

}

}

catch (CMsgBoxException\* ex)

{

AfxMessageBox(ex->m\_strMsg, ex->m\_uType, ex->m\_uHelpID);

ex->Delete();

delete pParams;

return false;

}

SearchStarted();

return true;

}

if (eSearchType == SearchTypeKademlia){…}

ASSERT(0);

delete pParams;

return false;

}

### 4、DoNewEd2kSearch

向服务器发送OP\_SearchRequest信息包，如果服务器返回的结果少于100，且用户选择发起globalserver search，则分三种情况分别再发送OP\_GLOBSEARCHREQ3、OP\_GLOBSEARCHREQ2、OP\_GLOBSEARCHREQ信息包

bool CSearchResultsWnd::DoNewEd2kSearch(SSearchParams\* pParams)

{

if (!theApp.serverconnect->IsConnected())

return false;

bool bServerSupports64Bit = theApp.serverconnect->GetCurrentServer() != NULL

&& (theApp.serverconnect->GetCurrentServer()->GetTCPFlags() & SRV\_TCPFLG\_LARGEFILES);

bool bPacketUsing64Bit = false;

CSafeMemFile data(100);

///snow:根据搜索参数构造searchpacket包体

if (!GetSearchPacket(&data, pParams, bServerSupports64Bit, &bPacketUsing64Bit) || data.GetLength() == 0)

return false;

///snow:searchpacket示例(包体部分，不包括包头)：

/// 0000 01 12 00 76697375616C 20 73747564696F 20 32303038 03 FFFFFF 3101010002

/// 关键字长度 visual 空格 studio 空格 2008

CancelEd2kSearch();

CStringA strResultType = pParams->strFileType;

if (strResultType == ED2KFTSTR\_PROGRAM)

strResultType.Empty();

m\_nEd2kSearchID++;

pParams->dwSearchID = m\_nEd2kSearchID;

theApp.searchlist->NewSearch(&searchlistctrl, strResultType, m\_nEd2kSearchID, pParams->eType, pParams->strExpression); ///snow:将当前搜索添加到搜索历史中

canceld = false;

///snow:取消定时器

if (m\_uTimerLocalServer){

VERIFY( KillTimer(m\_uTimerLocalServer) );

m\_uTimerLocalServer = 0;

}

// once we've sent a new search request, any previously received 'More' gets invalid.

CWnd\* pWndFocus = GetFocus();

m\_pwndParams->m\_ctlMore.EnableWindow(FALSE);

if (pWndFocus && pWndFocus->m\_hWnd == m\_pwndParams->m\_ctlMore.m\_hWnd)

m\_pwndParams->m\_ctlCancel.SetFocus();

m\_iSentMoreReq = 0;

Packet\* packet = new Packet(&data);

packet->opcode = OP\_SEARCHREQUEST;

if (thePrefs.GetDebugServerTCPLevel() > 0)

Debug(\_T(">>> Sending OP\_\_SearchRequest\n"));

theStats.AddUpDataOverheadServer(packet->size);

theApp.serverconnect->SendPacket(packet,false); ///snow:向服务器发送搜索请求

if (pParams->eType == SearchTypeEd2kGlobal && theApp.serverconnect->IsUDPSocketAvailable()) ///snow:全局服务器搜索

{

// set timeout timer for local server

m\_uTimerLocalServer = SetTimer(TimerServerTimeout, 50000, NULL); ///snow:每秒触发一次WM\_TIMER，m\_uTimerLocalServer=1，OnTimer()被调用

if (thePrefs.GetUseServerPriorities())

theApp.serverlist->ResetSearchServerPos();

if (globsearch){

delete searchpacket;

searchpacket = NULL;

m\_b64BitSearchPacket = false;

}

searchpacket = packet;

searchpacket->opcode = OP\_GLOBSEARCHREQ; // will be changed later when actually sending the packet!!

///snow:在OnTimer()中发送 分三种情况：searchpacket->opcode = OP\_GLOBSEARCHREQ3、OP\_GLOBSEARCHREQ2、OP\_GLOBSEARCHREQ; theApp.serverconnect->SendUDPPacket(searchpacket, toask, false);

m\_b64BitSearchPacket = bPacketUsing64Bit;

servercount = 0;

searchprogress.SetRange32(0, theApp.serverlist->GetServerCount() - 1); ///snow:搜索进度条

globsearch = true;

}

else{

globsearch = false;

delete packet;

}

CreateNewTab(pParams);

return true;

}

### 5、ProcessPacket

接收服务器传回的信息，并加以处理

bool CServerSocket::ProcessPacket(const BYTE\* packet, uint32 size, uint8 opcode)

{

try

{

switch (opcode) ///snow:在Opcodes.h定义opcode包含中packet中

{

///snow:服务器返回服务器相关信息

case OP\_SEARCHRESULT:{ ///snow:使用server或globalserver搜索时，返回搜索结果

if (thePrefs.GetDebugServerTCPLevel() > 0)

Debug(\_T("ServerMsg - OP\_SearchResult\n"));

CServer\* cur\_srv = (serverconnect) ? serverconnect->GetCurrentServer() : NULL;

CServer\* pServer = cur\_srv ? theApp.serverlist->GetServerByAddress(cur\_srv->GetAddress(), cur\_srv->GetPort()) : NULL;

(void)pServer;

bool bMoreResultsAvailable;

///snow:调用CSearchList::ProcessSearchAnswer()对搜索结果进行处理

UINT uSearchResults = theApp.searchlist->ProcessSearchAnswer(packet, size, true/\*pServer ? pServer->GetUnicodeSupport() : false\*/, cur\_srv ? cur\_srv->GetIP() : 0, cur\_srv ? cur\_srv->GetPort() : (uint16)0, &bMoreResultsAvailable);

theApp.emuledlg->searchwnd->LocalEd2kSearchEnd(uSearchResults, bMoreResultsAvailable); ///snow:决定是否进行GlobalServer Search

break;

}

### 6、ProcessSearchAnswer

///snow:在CServerSocket::ProcessPacket()中case OP\_SEARCHRESULT分支调用，由服务器返回，处理使用Server搜索时返回的结果，信息包的最后一个字节是或者，表示是否还有更多的搜索结果

UINT CSearchList::ProcessSearchAnswer(const uchar\* in\_packet, uint32 size, bool bOptUTF8,

uint32 nServerIP, uint16 nServerPort, bool\* pbMoreResultsAvailable)

{

CSafeMemFile packet(in\_packet, size);

UINT results = packet.ReadUInt32(); ///前四个字节表示搜索记录数，x012C=300

for (UINT i = 0; i < results; i++){

CSearchFile\* toadd = new CSearchFile(&packet, bOptUTF8, m\_nCurED2KSearchID);

toadd->SetClientServerIP(nServerIP);

toadd->SetClientServerPort(nServerPort);

if (nServerIP && nServerPort){

CSearchFile::SServer server(nServerIP, nServerPort, false);

server.m\_uAvail = toadd->GetIntTagValue(FT\_SOURCES);

toadd->AddServer(server);

}

AddToList(toadd, false);

}

if (m\_MobilMuleSearch)

theApp.mmserver->SearchFinished(false);

m\_MobilMuleSearch = false;

if (pbMoreResultsAvailable)

\*pbMoreResultsAvailable = false;

int iAddData = (int)(packet.GetLength() - packet.GetPosition()); ///snow:信息包的最后一个字节是或者，表示是否还有更多的搜索结果

if (iAddData == 1){

uint8 ucMore = packet.ReadUInt8();

if (ucMore == 0x00 || ucMore == 0x01){

if (pbMoreResultsAvailable)

\*pbMoreResultsAvailable = ucMore!=0;

if (thePrefs.GetDebugServerTCPLevel() > 0)

Debug(\_T(" Search answer(Server %s:%u): More=%u\n"), ipstr(nServerIP), nServerPort, ucMore);

}

else{

if (thePrefs.GetDebugServerTCPLevel() > 0)

Debug(\_T("\*\*\* NOTE: ProcessSearchAnswer(Server %s:%u): \*\*\*AddData: 1 byte: 0x%02x\n"), ipstr(nServerIP), nServerPort, ucMore);

}

}

else if (iAddData > 0){

if (thePrefs.GetDebugServerTCPLevel() > 0){

Debug(\_T("\*\*\* NOTE: ProcessSearchAnswer(Server %s:%u): \*\*\*AddData: %u bytes\n"), ipstr(nServerIP), nServerPort, iAddData);

DebugHexDump(in\_packet + packet.GetPosition(), iAddData);

}

}

packet.Close();

return GetED2KResultCount();

}

### 7、CSearchFile

CSearchFile::CSearchFile(CFileDataIO\* in\_data, bool bOptUTF8,

uint32 nSearchID, uint32 nServerIP, uint16 nServerPort, LPCTSTR pszDirectory, bool bKademlia, bool bServerUDPAnswer)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*snow:start\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* CSearchFile字节流示例

98 00 00 00 c2 70 df 6f 09 9f 44 da d7 6a fb 52 01 3f ce 8c 54 78 c9 9d 43 67 07 00 00 00

82 01 2c 00 49 72 6f 6e 20 4d 61 6e 20 33 20 64 76 64 39 20 50 41 4c 20 28 50 61 72 63 68

65 61 64 61 29 20 56 54 53 5f 30 36 5f 31 2e 56 4f 42 83 02 00 e0 5e 57 89 15 04 89 30 04

88 d4 83 04 94 d5 78 76 69 64 88 d3 46 27

98 00 00 00 搜索到个文件

第一个文件信息

c2 70 df 6f 09 9f 44 da d7 6a fb 52 01 3f ce 8c 文件Hash 16字节

54 78 c9 9d clientIP

43 67 clientPort

07 00 00 00 tag数7个

tag1:文件名

82 01 2C 00 TAGTYPE\_STRING FT\_FILENAME 44字节

tag2: 文件长度

83 02 00 e0 5e 57 TAGTYPE\_UINT32 5G

tag3:可用源数

89 15 04 TAGTYPE\_UINT8 FT\_SOURCES 4个源

tag4：完整源数

89 30 04 UINT8 FT\_COMPLETE\_SOURCES 4个完整源

tag5：比特率

88 d4 83 04 UINT16 d4=FT\_MEDIA\_BITRATE 0x0483=1155

tag6：codec

94 d5 78 76 69 64 (m\_uType(0x14) >= TAGTYPE\_STR1(0x11) && m\_uType <= TAGTYPE\_STR16(0x20) d5=FT\_MEDIA\_CODEC 78 76 69 64 = xvid

tag7:影片长度

88 d3 46 27 UINT16 d3=FT\_MEDIA\_LENGTH 0x2746=10054

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*snow:end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

m\_bMultipleAICHFound = false;

m\_bKademlia = bKademlia;

m\_bServerUDPAnswer = bServerUDPAnswer;

m\_nSearchID = nSearchID;

m\_FileIdentifier.SetMD4Hash(in\_data); ///snow:读取字节的Hash

m\_nClientID = in\_data->ReadUInt32(); ///snow:读取字节的IP

m\_nClientPort = in\_data->ReadUInt16(); ///snow:读取两字节的Port

if ((m\_nClientID || m\_nClientPort) && !IsValidSearchResultClientIPPort(m\_nClientID, m\_nClientPort)){

if (thePrefs.GetDebugServerSearchesLevel() > 1)

Debug(\_T("Filtered source from search result %s:%u\n"), DbgGetClientID(m\_nClientID), m\_nClientPort);

m\_nClientID = 0;

m\_nClientPort = 0;

}

UINT tagcount = in\_data->ReadUInt32(); ///snow:4字节的tag数

// NSERVER2.EXE (lugdunum v16.38 patched for Win32) returns the ClientIP+Port of the client which offered that

// file, even if that client has not filled the according fields in the OP\_OFFERFILES packet with its IP+Port.

//

// 16.38.p73 (lugdunum) (propenprinz)

// \*) does not return ClientIP+Port if the OP\_OFFERFILES packet does not also contain it.

// \*) if the OP\_OFFERFILES packet does contain our HighID and Port the server returns that data at least when

// returning search results via TCP.

if (thePrefs.GetDebugServerSearchesLevel() > 1)

Debug(\_T("Search Result: %s Client=%u.%u.%u.%u:%u Tags=%u\n"), md4str(m\_FileIdentifier.GetMD4Hash()), (uint8)m\_nClientID,(uint8)(m\_nClientID>>8),(uint8)(m\_nClientID>>16),(uint8)(m\_nClientID>>24), m\_nClientPort, tagcount);

// Copy/Convert ED2K-server tags to local tags

///snow:处理tag

for (UINT i = 0; i < tagcount; i++)

{

CTag\* tag = new CTag(in\_data, bOptUTF8);

if (thePrefs.GetDebugServerSearchesLevel() > 1)

Debug(\_T(" %s\n"), tag->GetFullInfo(DbgGetFileMetaTagName));

///snow:如果存在字符型的Tag名字，转换成代码

ConvertED2KTag(tag);

if (tag)

{

// Convert ED2K-server file rating tag

//

// NOTE: Feel free to do more with the received numbers here, but please do not add that particular

// received tag to the local tag list with the received tag format (packed rating). Either create

// a local tag with an eMule known rating value and drop the percentage (which is currently done),

// or add a second tag which holds the percentage in addition to the eMule-known rating value.

// Be aware, that adding that tag in packed-rating format will create troubles in other code parts!

if (tag->GetNameID() == FT\_FILERATING && tag->IsInt())

{

uint16 nPackedRating = (uint16)tag->GetInt();

// Percent of clients (related to 'Availability') which rated on that file

UINT uPercentClientRatings = HIBYTE(nPackedRating);

(void)uPercentClientRatings;

// Average rating used by clients

UINT uAvgRating = LOBYTE(nPackedRating);

m\_uUserRating = uAvgRating / (255/5/\*RatingExcellent\*/);

tag->SetInt(m\_uUserRating);

}

else if (tag->GetNameID() == FT\_AICH\_HASH && tag->IsStr()) ///snow:处理AICH标签

{

CAICHHash hash;

if (DecodeBase32(tag->GetStr(),hash) == (UINT)CAICHHash::GetHashSize())

m\_FileIdentifier.SetAICHHash(hash);

else

ASSERT( false );

delete tag;

tag = NULL;

continue;

}

taglist.Add(tag);

}

}

// here we have two choices

// - if the server/client sent us a filetype, we could use it (though it could be wrong)

// - we always trust our filetype list and determine the filetype by the extension of the file

//

// if we received a filetype from server, we use it.

// if we did not receive a filetype, we determine it by examining the file's extension.

//

// but, in no case, we will use the receive file type when adding this search result to the download queue, to avoid

// that we are using 'wrong' file types in part files. (this has to be handled when creating the part files)

const CString& rstrFileType = GetStrTagValue(FT\_FILETYPE); ///获取文件类型

SetFileName(GetStrTagValue(FT\_FILENAME), false, rstrFileType.IsEmpty(), true); ///snow:对文件名进行规范化

uint64 ui64FileSize = 0;

CTag\* pTagFileSize = GetTag(FT\_FILESIZE); ///snow:获取文件长度

if (pTagFileSize)

{

if (pTagFileSize->IsInt())

{

ui64FileSize = pTagFileSize->GetInt();

CTag\* pTagFileSizeHi = GetTag(FT\_FILESIZE\_HI);

if (pTagFileSizeHi) {

if (pTagFileSizeHi->IsInt())

ui64FileSize |= (uint64)pTagFileSizeHi->GetInt() << 32;

DeleteTag(pTagFileSizeHi);

}

pTagFileSize->SetInt64(ui64FileSize);

}

else if (pTagFileSize->IsInt64(false))

{

ui64FileSize = pTagFileSize->GetInt64();

DeleteTag(FT\_FILESIZE\_HI);

}

}

SetFileSize(ui64FileSize);

if (!rstrFileType.IsEmpty())

{

if (\_tcscmp(rstrFileType, \_T(ED2KFTSTR\_PROGRAM))==0)

{

CString strDetailFileType = GetFileTypeByName(GetFileName());

if (!strDetailFileType.IsEmpty())

SetFileType(strDetailFileType);

else

SetFileType(rstrFileType);

}

else

SetFileType(rstrFileType);

}

///snow:添加服务器到CSimpleArray<SServer> m\_aServers中

///snow:应该是两个用途，、把服务器添加到服务器列表中 2、spam过滤

m\_nClientServerIP = nServerIP;

m\_nClientServerPort = nServerPort;

if (m\_nClientServerIP && m\_nClientServerPort){

SServer server(m\_nClientServerIP, m\_nClientServerPort, bServerUDPAnswer);

server.m\_uAvail = GetIntTagValue(FT\_SOURCES);

AddServer(server);

}

m\_pszDirectory = pszDirectory ? \_tcsdup(pszDirectory) : NULL;

m\_list\_bExpanded = false;

m\_list\_parent = NULL;

m\_list\_childcount = 0;

m\_bPreviewPossible = false;

m\_eKnown = NotDetermined;

m\_nSpamRating = 0;

m\_nKadPublishInfo = 0;

}

### 8、AddToList

首先对待添加的CSearchFile的文件名进行核查，然后通过SearchID取得SearchList，遍历该list，检查list中不是子记录的条目，是否存在与待添加记录相同FileHash的，

1）、如果有，先检查该记录是否有子记录，如果没有，先将原记录自己复制为子记录，添加到list中，标记为原记录的子记录。然后再次遍历list，找出同一parent记录下的同一文件名的不同child记录，

A、如果有，增加其源数(FT\_SOURES)，

B、如果没有，添加toadd到list

2）、如果在list中尚没有同一fileHash的搜索记录，将toadd添加到队尾

bool CSearchList::AddToList(CSearchFile\* toadd, bool bClientResponse, uint32 dwFromUDPServerIP)

{

if (!bClientResponse && !m\_strResultFileType.IsEmpty() && \_tcscmp(m\_strResultFileType, toadd->GetFileType()) != 0)

{

delete toadd;

return false;

}

SearchList\* list = GetSearchListForID(toadd->GetSearchID());

// Spamfilter: Calculate the filename without any used keywords (and seperators) for later use

CString strNameWithoutKeyword;

CString strName = toadd->GetFileName();

strName.MakeLower();

int nPos = 0;

CString strToken = strName.Tokenize(\_T(".[]()!-'\_ "), nPos);

bool bFound;

while (!strToken.IsEmpty()){

bFound = false;

if (!bClientResponse && toadd->GetSearchID() == m\_nCurED2KSearchID){

for (int i = 0; i < m\_astrSpamCheckCurSearchExp.GetCount(); i++){

if (strToken.Compare(m\_astrSpamCheckCurSearchExp[i]) == 0){

bFound = true;

break;

}

}

}

if (!bFound){

if (!strNameWithoutKeyword.IsEmpty())

strNameWithoutKeyword += \_T(" ");

strNameWithoutKeyword += strToken;

}

strToken = strName.Tokenize(\_T(".[]()!-'\_ "), nPos);

}

toadd->SetNameWithoutKeyword(strNameWithoutKeyword);

// search for a 'parent' with same filehash and search-id as the new search result entry

for (POSITION pos = list->GetHeadPosition(); pos != NULL; )

{

CSearchFile\* parent = list->GetNext(pos);

if ( parent->GetListParent() == NULL

&& md4cmp(parent->GetFileHash(), toadd->GetFileHash()) == 0)

{

// if this parent does not yet have any child entries, create one child entry

// which is equal to the current parent entry (needed for GUI when expanding the child list).

if (parent->GetListChildCount() == 0)

{

CSearchFile\* child = new CSearchFile(parent);

child->SetListParent(parent);

int iSources = parent->GetIntTagValue(FT\_SOURCES);

if (iSources == 0)

iSources = 1;

child->SetListChildCount(iSources);

list->AddTail(child);

parent->SetListChildCount(1);

}

// get the 'Availability' of the new search result entry

UINT uAvail;

if (bClientResponse) {

// If this is a response from a client ("View Shared Files"), we set the "Availability" at least to 1.

if (!toadd->GetIntTagValue(FT\_SOURCES, uAvail) || uAvail==0)

uAvail = 1;

}

else

uAvail = toadd->GetIntTagValue(FT\_SOURCES);

// get 'Complete Sources' of the new search result entry

uint32 uCompleteSources = (uint32)-1;

bool bHasCompleteSources = toadd->GetIntTagValue(FT\_COMPLETE\_SOURCES, uCompleteSources);

bool bFound = false;

if (thePrefs.GetDebugSearchResultDetailLevel() >= 1)

{

; // for debugging: do not merge search results

}

else

{

// check if that parent already has a child with same filename as the new search result entry

for (POSITION pos2 = list->GetHeadPosition(); pos2 != NULL && !bFound; )

{

CSearchFile\* child = list->GetNext(pos2);

if ( child != toadd // not the same object

&& child->GetListParent() == parent // is a child of our result (one filehash)

&& toadd->GetFileName().CompareNoCase(child->GetFileName()) == 0) // same name

{

bFound = true;

// add properties of new search result entry to the already available child entry (with same filename)

// ed2k: use the sum of all values, kad: use the max. values

if (toadd->IsKademlia()) {

if (uAvail > child->GetListChildCount())

child->SetListChildCount(uAvail);

}

else {

child->AddListChildCount(uAvail);

}

child->AddSources(uAvail);

if (bHasCompleteSources)

child->AddCompleteSources(uCompleteSources);

// Check AICH Hash - if they differ, clear it (see KademliaSearchKeyword)

// if we didn't have a hash yet, take it over

if (toadd->GetFileIdentifier().HasAICHHash())

{

if (child->GetFileIdentifier().HasAICHHash())

{

if (parent->GetFileIdentifier().GetAICHHash() != toadd->GetFileIdentifier().GetAICHHash())

{

DEBUG\_ONLY(DebugLogWarning(\_T("Kad: SearchList: AddToList: Received searchresult with different AICH hash than existing one, ignoring AICH for result %s"), child->GetFileName()) );

child->SetFoundMultipleAICH();

child->GetFileIdentifier().ClearAICHHash();

}

}

else if (!child->DidFoundMultipleAICH())

{

DEBUG\_ONLY(DebugLog(\_T("Kad: SearchList: AddToList: Received searchresult with new AICH hash %s, taking over to existing result. Entry: %s"), toadd->GetFileIdentifier().GetAICHHash().GetString(), child->GetFileName()) );

child->GetFileIdentifier().SetAICHHash(toadd->GetFileIdentifier().GetAICHHash());

}

}

break;

}

}

}

if (!bFound)

{

// the parent which we had found does not yet have a child with that new search result's entry name,

// add the new entry as a new child

//

toadd->SetListParent(parent);

toadd->SetListChildCount(uAvail);

parent->AddListChildCount(1);

list->AddHead(toadd);

}

// copy possible available sources from new search result entry to parent

if (toadd->GetClientID() && toadd->GetClientPort())

{

if (IsValidSearchResultClientIPPort(toadd->GetClientID(), toadd->GetClientPort()))

{

// pre-filter sources which would be dropped in CPartFile::AddSources

if (CPartFile::CanAddSource(toadd->GetClientID(), toadd->GetClientPort(), toadd->GetClientServerIP(), toadd->GetClientServerPort()))

{

CSearchFile::SClient client(toadd->GetClientID(), toadd->GetClientPort(),

toadd->GetClientServerIP(), toadd->GetClientServerPort());

if (parent->GetClients().Find(client) == -1)

parent->AddClient(client);

}

}

else

{

if (thePrefs.GetDebugServerSearchesLevel() > 1)

{

uint32 nIP = toadd->GetClientID();

Debug(\_T("Filtered source from search result %s:%u\n"), DbgGetClientID(nIP), toadd->GetClientPort());

}

}

}

// copy possible available servers from new search result entry to parent

// will be used in future

if (toadd->GetClientServerIP() && toadd->GetClientServerPort())

{

CSearchFile::SServer server(toadd->GetClientServerIP(), toadd->GetClientServerPort(), toadd->IsServerUDPAnswer());

int iFound = parent->GetServers().Find(server);

if (iFound == -1) {

server.m\_uAvail = uAvail;

parent->AddServer(server);

}

else

parent->GetServerAt(iFound).m\_uAvail += uAvail;

}

UINT uAllChildsSourceCount = 0; // ed2k: sum of all sources, kad: the max. sources found

UINT uAllChildsCompleteSourceCount = 0; // ed2k: sum of all sources, kad: the max. sources found

UINT uDifferentNames = 0; // max known different names

UINT uPublishersKnown = 0; // max publishers known (might be changed to median)

UINT uTrustValue = 0; // average trust value (might be changed to median)

uint32 nPublishInfoTags = 0;

const CSearchFile\* bestEntry = NULL;

bool bHasMultipleAICHHashs = false;

CAICHHash aichHash;

bool bAICHHashValid = false;

for (POSITION pos2 = list->GetHeadPosition(); pos2 != NULL; )

{

const CSearchFile\* child = list->GetNext(pos2);

if (child->GetListParent() == parent)

{

// figure out if the childs of different AICH hashs

if (child->GetFileIdentifierC().HasAICHHash())

{

if (bAICHHashValid && aichHash != child->GetFileIdentifierC().GetAICHHash())

bHasMultipleAICHHashs = true;

else if (!bAICHHashValid)

{

aichHash = child->GetFileIdentifierC().GetAICHHash();

bAICHHashValid = true;

}

}

else if (child->DidFoundMultipleAICH())

bHasMultipleAICHHashs = true;

if (parent->IsKademlia())

{

if (child->GetListChildCount() > uAllChildsSourceCount)

uAllChildsSourceCount = child->GetListChildCount();

/\*if (child->GetCompleteSourceCount() > uAllChildsCompleteSourceCount) // not yet supported

uAllChildsCompleteSourceCount = child->GetCompleteSourceCount();\*/

if (child->GetKadPublishInfo() != 0){

nPublishInfoTags++;

uDifferentNames = max(uDifferentNames, ((child->GetKadPublishInfo() & 0xFF000000) >> 24));

uPublishersKnown = max (uPublishersKnown, ((child->GetKadPublishInfo() & 0x00FF0000) >> 16));

uTrustValue += child->GetKadPublishInfo() & 0x0000FFFF;

}

}

else

{

uAllChildsSourceCount += child->GetListChildCount();

uAllChildsCompleteSourceCount += child->GetCompleteSourceCount();

}

if (bestEntry == NULL)

bestEntry = child;

else if (child->GetListChildCount() > bestEntry->GetListChildCount())

bestEntry = child;

}

}

if (bestEntry)

{

parent->SetFileSize(bestEntry->GetFileSize());

parent->SetFileName(bestEntry->GetFileName());

parent->SetFileType(bestEntry->GetFileType());

parent->ClearTags();

parent->CopyTags(bestEntry->GetTags());

parent->SetIntTagValue(FT\_SOURCES, uAllChildsSourceCount);

parent->SetIntTagValue(FT\_COMPLETE\_SOURCES, uAllChildsCompleteSourceCount);

if (uTrustValue > 0 && nPublishInfoTags > 0)

uTrustValue = uTrustValue / nPublishInfoTags;

parent->SetKadPublishInfo(((uDifferentNames & 0xFF) << 24) | ((uPublishersKnown & 0xFF) << 16) | ((uTrustValue & 0xFFFF) << 0));

// if all childs have the same AICH hash (or none), set the parent hash to it, otherwise clear it (see KademliaSearchKeyword)

if (bHasMultipleAICHHashs || !bAICHHashValid)

parent->GetFileIdentifier().ClearAICHHash();

else if (bAICHHashValid)

parent->GetFileIdentifier().SetAICHHash(aichHash);

}

// recalculate spamrating

DoSpamRating(parent, bClientResponse, false, false, false, dwFromUDPServerIP);

// add the 'Availability' of the new search result entry to the total search result count for this search

AddResultCount(parent->GetSearchID(), parent->GetFileHash(), uAvail, parent->IsConsideredSpam());

// update parent in GUI

if (outputwnd && !m\_MobilMuleSearch)

outputwnd->UpdateSources(parent);

if (bFound)

delete toadd;

return true;

}

}

// no bounded result found yet -> add as parent to list

toadd->SetListParent(NULL);

UINT uAvail = 0;

if (list->AddTail(toadd))

{

UINT tempValue = 0;

VERIFY( m\_foundFilesCount.Lookup(toadd->GetSearchID(), tempValue) );

m\_foundFilesCount.SetAt(toadd->GetSearchID(), tempValue + 1);

// get the 'Availability' of this new search result entry

if (bClientResponse) {

// If this is a response from a client ("View Shared Files"), we set the "Availability" at least to 1.

if (!toadd->GetIntTagValue(FT\_SOURCES, uAvail) || uAvail==0)

uAvail = 1;

toadd->AddSources(uAvail);

}

else

uAvail = toadd->GetIntTagValue(FT\_SOURCES);

}

if (thePrefs.GetDebugSearchResultDetailLevel() >= 1)

toadd->SetListExpanded(true);

// calculate spamrating

DoSpamRating(toadd, bClientResponse, false, false, false, dwFromUDPServerIP);

// add the 'Availability' of this new search result entry to the total search result count for this search

AddResultCount(toadd->GetSearchID(), toadd->GetFileHash(), uAvail, toadd->IsConsideredSpam());

// add parent in GUI

if (outputwnd && !m\_MobilMuleSearch)

outputwnd->AddResult(toadd);

return true;

}

### 9、LocalEd2kSearchEnd

在CServerSocket::ProcessPacket()中调用，决定是否进行GlobalServer Search

void CSearchResultsWnd::LocalEd2kSearchEnd(UINT count, bool bMoreResultsAvailable)

{

// local server has answered, kill the timeout timer

if (m\_uTimerLocalServer) {

VERIFY( KillTimer(m\_uTimerLocalServer) );

m\_uTimerLocalServer = 0;

}

if (!canceld && count > MAX\_RESULTS) ///snow:当搜索记录大于时，中止搜索，所以global server 搜索并没开始，只有记录数小于时，才会启动！

CancelEd2kSearch();

if (!canceld) {

if (!globsearch)

SearchCanceled(m\_nEd2kSearchID);

else

VERIFY( (global\_search\_timer = SetTimer(TimerGlobalSearch, 750, 0)) != NULL );

}

m\_pwndParams->m\_ctlMore.EnableWindow(bMoreResultsAvailable && m\_iSentMoreReq < MAX\_MORE\_SEARCH\_REQ); ///snow:More按钮是否可用

}

### 10、OnTimer

///snow:在LocalEd2kSearchEnd()中，设定当返回搜索记录数大于100时，中止搜索，定时器被终止掉，所以OnTimer()未被调用，当记录小于100时，启动OnTimer

void CSearchResultsWnd::OnTimer(UINT nIDEvent)

{

CResizableFormView::OnTimer(nIDEvent);

///snow:在DoNewEd2KSearch()中global search分支下设置m\_uTimerLocalServer=SetTimer(TimerServerTimeout, 50000, 0) m\_uTimerLocalServer=nIDEvent=1

if (m\_uTimerLocalServer != 0 && nIDEvent == m\_uTimerLocalServer)

{ ///snow:此分支处理定时器，把自己取消，如果启用globsearch，设置定时器

if (thePrefs.GetDebugServerSearchesLevel() > 0)

Debug(\_T("Timeout waiting on search results of local server\n"));

// the local server did not answer within the timeout

VERIFY( KillTimer(m\_uTimerLocalServer) ); ///snow:取消DoNewEd2KSearch()中global search分支下设置的定时器，也就是说定时器只使用一次

m\_uTimerLocalServer = 0;

// start the global search

if (globsearch)

{

if (global\_search\_timer == 0)

VERIFY( (global\_search\_timer = SetTimer(TimerGlobalSearch, 750, 0)) != NULL ); ///snow:设置定时器，定时器的取消在CancelEd2kSearch()中

}

else

CancelEd2kSearch(); ///snow:取消定时器、，GlobalSearch等

}

else if (nIDEvent == global\_search\_timer)

{ ///snow:处理定时器分支，每次取一个服务器，向该服务器发送搜索请求

if (theApp.serverconnect->IsConnected())

{

CServer\* pConnectedServer = theApp.serverconnect->GetCurrentServer();

if (pConnectedServer)

pConnectedServer = theApp.serverlist->GetServerByAddress(pConnectedServer->GetAddress(), pConnectedServer->GetPort());

CServer\* toask = NULL;

while (servercount < theApp.serverlist->GetServerCount()-1)

{

servercount++;

searchprogress.StepIt();

toask = theApp.serverlist->GetNextSearchServer();

if (toask == NULL)

break;

if (toask == pConnectedServer) { ///snow:当前服务器

toask = NULL;

continue;

}

if (toask->GetFailedCount() >= thePrefs.GetDeadServerRetries()) { ///snow:连接失败次数超过死服务器重试次数的

toask = NULL;

continue;

}

break;

}

if (toask)

{ ///snow:找到可用服务器了，根据服务器的UDPFlags和是否支持LargeFilesUDP，分三种情况处理

///1、支持大文件UDP且有UDP标记 OP\_GLOBSEARCHREQ3

///2、不支持大文件UDP但有UDP标记 OP\_GLOBSEARCHREQ2;

///3、

bool bRequestSent = false;

if (toask->SupportsLargeFilesUDP() && (toask->GetUDPFlags() & SRV\_UDPFLG\_EXT\_GETFILES))

{ ///snow:服务器标记了UDPFlags且运行大文件UDP

CSafeMemFile data(50);

uint32 nTagCount = 1;

data.WriteUInt32(nTagCount);

CTag tagFlags(CT\_SERVER\_UDPSEARCH\_FLAGS, SRVCAP\_UDP\_NEWTAGS\_LARGEFILES);

tagFlags.WriteNewEd2kTag(&data);

///snow:将DoNewED2KSearch()中的searchpacket封包，改变Opcode

Packet\* pExtSearchPacket = new Packet(OP\_GLOBSEARCHREQ3, searchpacket->size + (uint32)data.GetLength());

data.SeekToBegin();

data.Read(pExtSearchPacket->pBuffer, (uint32)data.GetLength());

memcpy(pExtSearchPacket->pBuffer+(uint32)data.GetLength(), searchpacket->pBuffer, searchpacket->size);

theStats.AddUpDataOverheadServer(pExtSearchPacket->size);

theApp.serverconnect->SendUDPPacket(pExtSearchPacket, toask, true);

bRequestSent = true;

if (thePrefs.GetDebugServerUDPLevel() > 0)

Debug(\_T(">>> Sending %s to server %-21s (%3u of %3u)\n"), \_T("OP\_\_GlobSearchReq3"), ipstr(toask->GetAddress(), toask->GetPort()), servercount, theApp.serverlist->GetServerCount());

}

else if (toask->GetUDPFlags() & SRV\_UDPFLG\_EXT\_GETFILES)

{ ///snow:标记了UDPflags

if (!m\_b64BitSearchPacket || toask->SupportsLargeFilesUDP()){

///snow:将DoNewED2KSearch()中的searchpacket封包，改变Opcode

searchpacket->opcode = OP\_GLOBSEARCHREQ2;

if (thePrefs.GetDebugServerUDPLevel() > 0)

Debug(\_T(">>> Sending %s to server %-21s (%3u of %3u)\n"), \_T("OP\_\_GlobSearchReq2"), ipstr(toask->GetAddress(), toask->GetPort()), servercount, theApp.serverlist->GetServerCount());

theStats.AddUpDataOverheadServer(searchpacket->size);

theApp.serverconnect->SendUDPPacket(searchpacket, toask, false);

bRequestSent = true;

}

else{

if (thePrefs.GetDebugServerUDPLevel() > 0)

Debug(\_T(">>> Skipped UDP search on server %-21s (%3u of %3u): No large file support\n"), ipstr(toask->GetAddress(), toask->GetPort()), servercount, theApp.serverlist->GetServerCount());

}

}

else

{ ///snow:UDPFlags未标记

if (!m\_b64BitSearchPacket || toask->SupportsLargeFilesUDP()){

searchpacket->opcode = OP\_GLOBSEARCHREQ;

if (thePrefs.GetDebugServerUDPLevel() > 0)

Debug(\_T(">>> Sending %s to server %-21s (%3u of %3u)\n"), \_T("OP\_\_GlobSearchReq1"), ipstr(toask->GetAddress(), toask->GetPort()), servercount, theApp.serverlist->GetServerCount());

theStats.AddUpDataOverheadServer(searchpacket->size);

theApp.serverconnect->SendUDPPacket(searchpacket, toask, false);

bRequestSent = true;

}

else{

if (thePrefs.GetDebugServerUDPLevel() > 0)

Debug(\_T(">>> Skipped UDP search on server %-21s (%3u of %3u): No large file support\n"), ipstr(toask->GetAddress(), toask->GetPort()), servercount, theApp.serverlist->GetServerCount());

}

}

if (bRequestSent)

theApp.searchlist->SentUDPRequestNotification(m\_nEd2kSearchID, toask->GetIP());

}

else

CancelEd2kSearch();

}

else

CancelEd2kSearch();

}

else

ASSERT( 0 );

}

### 11、SearchMore

bool CSearchResultsWnd::SearchMore()

{

if (!theApp.serverconnect->IsConnected())

return false;

SetActiveSearchResultsIcon(m\_nEd2kSearchID);

canceld = false;

Packet\* packet = new Packet();

packet->opcode = OP\_QUERY\_MORE\_RESULT;

if (thePrefs.GetDebugServerTCPLevel() > 0)

Debug(\_T(">>> Sending OP\_\_QueryMoreResults\n"));

theStats.AddUpDataOverheadServer(packet->size);

theApp.serverconnect->SendPacket(packet);

m\_iSentMoreReq++;

return true;

}

## 三、搜索记录的结构

1、CTypedPtrList<CPtrList, SearchListsStruct\*> m\_listFileLists 所有搜索文件列表的指针集合，一个搜索关键字一个列表

2、SearchListsStruct列表为一个包含m\_nSearchID和listSearchFiles的 struct结构，m\_nSearchID为搜索关键字的ID，listSearchFiles为该ID对应的SearchList

typedef struct {

uint32 m\_nSearchID;

SearchList m\_listSearchFiles;

} SearchListsStruct;

3、typedef CTypedPtrList<CPtrList, CSearchFile\*> SearchList;

SearchList为包含每一个CSearchFile对象的指针列表

综合起来就是一个一对多的关系，一个SearchID对应多个SearchFile

4、SearchFile结构：

1、ED2K部分

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*snow:start\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* CSearchFile字节流示例

98 00 00 00 c2 70 df 6f 09 9f 44 da d7 6a fb 52 01 3f ce 8c 54 78 c9 9d 43 67 07 00 00 00

82 01 2c 00 49 72 6f 6e 20 4d 61 6e 20 33 20 64 76 64 39 20 50 41 4c 20 28 50 61 72 63 68

65 61 64 61 29 20 56 54 53 5f 30 36 5f 31 2e 56 4f 42 83 02 00 e0 5e 57 89 15 04 89 30 04

88 d4 83 04 94 d5 78 76 69 64 88 d3 46 27

98 00 00 00 搜索到个文件

第一个文件信息

c2 70 df 6f 09 9f 44 da d7 6a fb 52 01 3f ce 8c 文件Hash 16字节

54 78 c9 9d clientIP

43 67 clientPort

07 00 00 00 tag数7个

tag1:文件名

82 01 2C 00 TAGTYPE\_STRING FT\_FILENAME 44字节

tag2: 文件长度

83 02 00 e0 5e 57 TAGTYPE\_UINT32 5G

tag3:可用源数

89 15 04 TAGTYPE\_UINT8 FT\_SOURCES 4个源

tag4：完整源数

89 30 04 UINT8 FT\_COMPLETE\_SOURCES 4个完整源

tag5：比特率

88 d4 83 04 UINT16 d4=FT\_MEDIA\_BITRATE 0x0483=1155

tag6：codec

94 d5 78 76 69 64 (m\_uType(0x14) >= TAGTYPE\_STR1(0x11) && m\_uType <= TAGTYPE\_STR16(0x20) d5=FT\_MEDIA\_CODEC 78 76 69 64 = xvid

tag7:影片长度

88 d3 46 27 UINT16 d3=FT\_MEDIA\_LENGTH 0x2746=10054

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*snow:end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2、KAD部分

搜索记录怎么处理？在KademliaSearchKeyword中处理，同ED2K处理流程与顺序一致

# 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。

# 文件下载

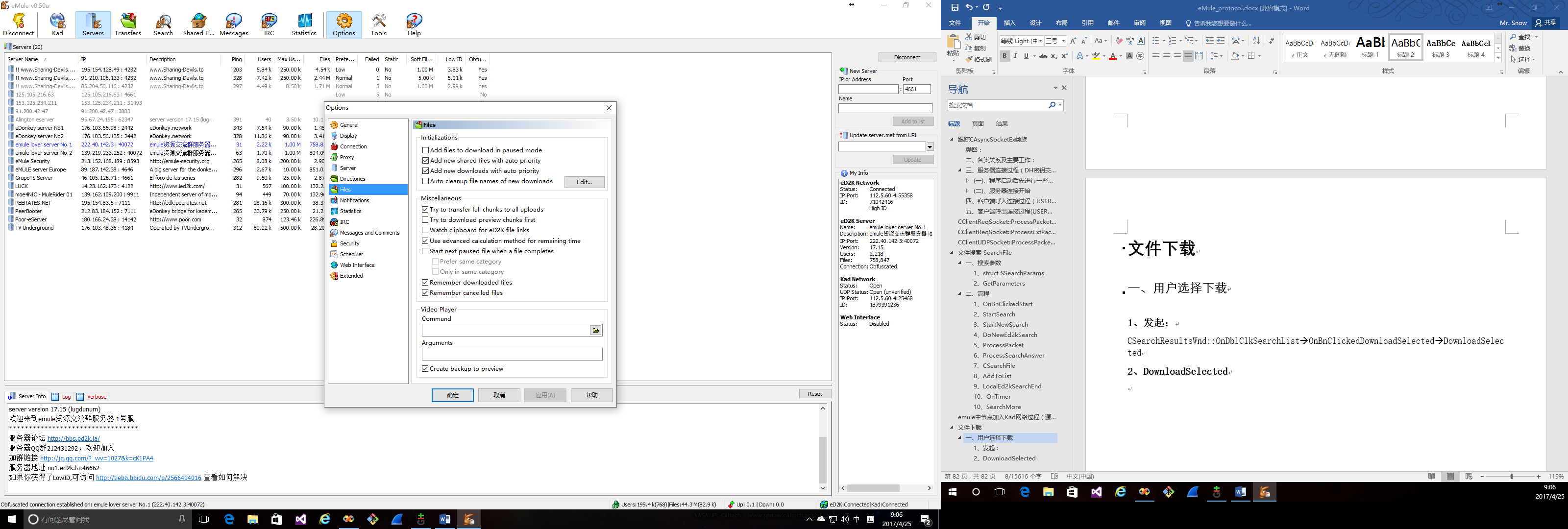
## 一、用户选择下载

### 1、发起：

CSearchResultsWnd::OnDblClkSearchList🡪OnBnClickedDownloadSelected🡪DownloadSelected

#### 1.1、DownloadSelected

///snow:下载选择的文件:参数bPaused为thePrefs.AddNewFilesPaused()，在选项-->Files-->Add new file to download in pause mode中选定，默认为false



void CSearchResultsWnd::DownloadSelected(bool bPaused)

{

CWaitCursor curWait;

POSITION pos = searchlistctrl.GetFirstSelectedItemPosition();

while (pos != NULL)

{

int iIndex = searchlistctrl.GetNextSelectedItem(pos);

if (iIndex >= 0)

{

// get selected listview item (may be a child item from an expanded search result)

const CSearchFile\* sel\_file = (CSearchFile\*)searchlistctrl.GetItemData(iIndex);

// get parent

const CSearchFile\* parent;

if (sel\_file->GetListParent() != NULL)

parent = sel\_file->GetListParent();

else

parent = sel\_file;

if (parent->IsComplete() == 0 && parent->GetSourceCount() >= 50)

{

CString strMsg;

strMsg.Format(GetResString(IDS\_ASKDLINCOMPLETE), sel\_file->GetFileName());

int iAnswer = AfxMessageBox(strMsg, MB\_ICONQUESTION | MB\_YESNO | MB\_DEFBUTTON2);

if (iAnswer != IDYES)

continue;

}

// create new DL-queue entry with all properties of parent (e.g. already received sources!)

// but with the filename of the selected listview item.

CSearchFile tempFile(parent);

tempFile.SetFileName(sel\_file->GetFileName());

tempFile.SetStrTagValue(FT\_FILENAME, sel\_file->GetFileName());

theApp.downloadqueue->AddSearchToDownload(&tempFile, bPaused, GetSelectedCat());

// update parent and all childs

searchlistctrl.UpdateSources(parent);

}

}

}

#### 1.2、AddSearchToDownload

void CDownloadQueue::AddSearchToDownload(CSearchFile\* toadd, uint8 paused, int cat)

{

if (toadd->GetFileSize()== (uint64)0 || IsFileExisting(toadd->GetFileHash()))

return;

if (toadd->GetFileSize() > OLD\_MAX\_EMULE\_FILE\_SIZE && !thePrefs.CanFSHandleLargeFiles(cat)){

LogError(LOG\_STATUSBAR, GetResString(IDS\_ERR\_FSCANTHANDLEFILE));

return;

}

CPartFile\* newfile = new CPartFile(toadd,cat);

if (newfile->GetStatus() == PS\_ERROR){

delete newfile;

return;

}

if (paused == 2)

paused = (uint8)thePrefs.AddNewFilesPaused();

AddDownload(newfile, (paused==1));

// If the search result is from OP\_GLOBSEARCHRES there may also be a source

if (toadd->GetClientID() && toadd->GetClientPort()){

CSafeMemFile sources(1+4+2);

try{

sources.WriteUInt8(1);

sources.WriteUInt32(toadd->GetClientID());

sources.WriteUInt16(toadd->GetClientPort());

sources.SeekToBegin();

newfile->AddSources(&sources, toadd->GetClientServerIP(), toadd->GetClientServerPort(), false);

}

catch(CFileException\* error){

ASSERT(0);

error->Delete();

}

}

// Add more sources which were found via global UDP search

const CSimpleArray<CSearchFile::SClient>& aClients = toadd->GetClients();

for (int i = 0; i < aClients.GetSize(); i++){

CSafeMemFile sources(1+4+2);

try{

sources.WriteUInt8(1);

sources.WriteUInt32(aClients[i].m\_nIP);

sources.WriteUInt16(aClients[i].m\_nPort);

sources.SeekToBegin();

newfile->AddSources(&sources,aClients[i].m\_nServerIP, aClients[i].m\_nServerPort, false);

}

catch(CFileException\* error){

ASSERT(0);

error->Delete();

break;

}

}

}

#### 1.3、建立PartFIle

##### 1.3.1、CPartFile

CPartFile::CPartFile(CSearchFile\* searchresult, UINT cat)

{

Init();

const CTypedPtrList<CPtrList, Kademlia::CEntry\*>& list = searchresult->getNotes();

for(POSITION pos = list.GetHeadPosition(); pos != NULL; )

{

Kademlia::CEntry\* entry = list.GetNext(pos);

m\_kadNotes.AddTail(entry->Copy());

}

UpdateFileRatingCommentAvail();

m\_FileIdentifier.SetMD4Hash(searchresult->GetFileHash());

if (searchresult->GetFileIdentifierC().HasAICHHash())

{

m\_FileIdentifier.SetAICHHash(searchresult->GetFileIdentifierC().GetAICHHash());

m\_pAICHRecoveryHashSet->SetMasterHash(searchresult->GetFileIdentifierC().GetAICHHash(), AICH\_VERIFIED);

}

for (int i = 0; i < searchresult->taglist.GetCount();i++){

const CTag\* pTag = searchresult->taglist[i];

switch (pTag->GetNameID()){

case FT\_FILENAME:{

ASSERT( pTag->IsStr() );

if (pTag->IsStr()){

if (GetFileName().IsEmpty())

SetFileName(pTag->GetStr(), true, true);

}

break;

}

case FT\_FILESIZE:{

ASSERT( pTag->IsInt64(true) );

if (pTag->IsInt64(true))

SetFileSize(pTag->GetInt64());

break;

}

default:{

bool bTagAdded = false;

if (pTag->GetNameID() != 0 && pTag->GetName() == NULL && (pTag->IsStr() || pTag->IsInt()))

{

static const struct

{

uint8 nName;

uint8 nType;

} \_aMetaTags[] =

{

{ FT\_MEDIA\_ARTIST, 2 },

{ FT\_MEDIA\_ALBUM, 2 },

{ FT\_MEDIA\_TITLE, 2 },

{ FT\_MEDIA\_LENGTH, 3 },

{ FT\_MEDIA\_BITRATE, 3 },

{ FT\_MEDIA\_CODEC, 2 },

{ FT\_FILETYPE, 2 },

{ FT\_FILEFORMAT, 2 }

};

for (int t = 0; t < ARRSIZE(\_aMetaTags); t++)

{

if (pTag->GetType() == \_aMetaTags[t].nType && pTag->GetNameID() == \_aMetaTags[t].nName)

{

// skip string tags with empty string values

if (pTag->IsStr() && pTag->GetStr().IsEmpty())

break;

// skip integer tags with '0' values

if (pTag->IsInt() && pTag->GetInt() == 0)

break;

TRACE(\_T("CPartFile::CPartFile(CSearchFile\*): added tag %s\n"), pTag->GetFullInfo(DbgGetFileMetaTagName));

CTag\* newtag = new CTag(\*pTag);

taglist.Add(newtag);

bTagAdded = true;

break;

}

}

}

if (!bTagAdded)

TRACE(\_T("CPartFile::CPartFile(CSearchFile\*): ignored tag %s\n"), pTag->GetFullInfo(DbgGetFileMetaTagName));

}

}

}

CreatePartFile(cat);

m\_category=cat;

}

##### 1.3.2、CreatePartFile

void CPartFile::CreatePartFile(UINT cat)

{

if (m\_nFileSize > (uint64)MAX\_EMULE\_FILE\_SIZE){

LogError(LOG\_STATUSBAR, GetResString(IDS\_ERR\_CREATEPARTFILE));

SetStatus(PS\_ERROR);

return;

}

// decide which tempfolder to use

CString tempdirtouse=theApp.downloadqueue->GetOptimalTempDir(cat,GetFileSize());

// use lowest free partfilenumber for free file (InterCeptor)

int i = 0;

CString filename;

do{

i++;

filename.Format(\_T("%s\\%03i.part"), tempdirtouse, i);

}

while (PathFileExists(filename));

m\_partmetfilename.Format(\_T("%03i.part.met"), i);

SetPath(tempdirtouse);

m\_fullname.Format(\_T("%s\\%s"), tempdirtouse, m\_partmetfilename);

CTag\* partnametag = new CTag(FT\_PARTFILENAME,RemoveFileExtension(m\_partmetfilename));

taglist.Add(partnametag);

Gap\_Struct\* gap = new Gap\_Struct;

gap->start = 0;

gap->end = m\_nFileSize - (uint64)1;

gaplist.AddTail(gap);

CString partfull(RemoveFileExtension(m\_fullname));

SetFilePath(partfull);

if (!m\_hpartfile.Open(partfull,CFile::modeCreate|CFile::modeReadWrite|CFile::shareDenyWrite|CFile::osSequentialScan)){

LogError(LOG\_STATUSBAR, GetResString(IDS\_ERR\_CREATEPARTFILE));

SetStatus(PS\_ERROR);

}

else{

if (thePrefs.GetSparsePartFiles()){

DWORD dwReturnedBytes = 0;

if (!DeviceIoControl(m\_hpartfile.m\_hFile, FSCTL\_SET\_SPARSE, NULL, 0, NULL, 0, &dwReturnedBytes, NULL))

{

// Errors:

// ERROR\_INVALID\_FUNCTION returned by WinXP when attempting to create a sparse file on a FAT32 partition

DWORD dwError = GetLastError();

if (dwError != ERROR\_INVALID\_FUNCTION && thePrefs.GetVerboseLogPriority() <= DLP\_VERYLOW)

DebugLogError(\_T("Failed to apply NTFS sparse file attribute to file \"%s\" - %s"), partfull, GetErrorMessage(dwError, 1));

}

}

struct \_stat fileinfo;

if (\_tstat(partfull, &fileinfo) == 0){

m\_tLastModified = fileinfo.st\_mtime;

m\_tCreated = fileinfo.st\_ctime;

}

else

AddDebugLogLine(false, \_T("Failed to get file date for \"%s\" - %s"), partfull, \_tcserror(errno));

}

m\_dwFileAttributes = GetFileAttributes(partfull);

if (m\_dwFileAttributes == INVALID\_FILE\_ATTRIBUTES)

m\_dwFileAttributes = 0;

if (m\_FileIdentifier.GetTheoreticalMD4PartHashCount() == 0)

m\_bMD4HashsetNeeded = false;

if (m\_FileIdentifier.GetTheoreticalAICHPartHashCount() == 0)

m\_bAICHPartHashsetNeeded = false;

m\_SrcpartFrequency.SetSize(GetPartCount());

for (UINT i = 0; i < GetPartCount();i++)

m\_SrcpartFrequency[i] = 0;

paused = false;

if (thePrefs.AutoFilenameCleanup())

SetFileName(CleanupFilename(GetFileName()));

SavePartFile();

SetActive(theApp.IsConnected());

}

##### 1.3.3、SavePartFile

bool CPartFile::SavePartFile(bool bDontOverrideBak)

{ ……//省略

file.WriteUInt8( IsLargeFile()? PARTFILE\_VERSION\_LARGEFILE : PARTFILE\_VERSION);

//date

file.WriteUInt32(m\_tUtcLastModified);

//hash

m\_FileIdentifier.WriteMD4HashsetToFile(&file);

UINT uTagCount = 0;

ULONG uTagCountFilePos = (ULONG)file.GetPosition();

file.WriteUInt32(uTagCount);

CTag nametag(FT\_FILENAME, GetFileName());

nametag.WriteTagToFile(&file, utf8strOptBOM);

uTagCount++;

CTag sizetag(FT\_FILESIZE, m\_nFileSize, IsLargeFile());

sizetag.WriteTagToFile(&file);

transtag.WriteTagToFile(&file);

…///省略：各Tag

if (m\_FileIdentifier.HasAICHHash()){

CTag aichtag(FT\_AICH\_HASH, m\_FileIdentifier.GetAICHHash().GetString() );

aichtag.WriteTagToFile(&file);

}

#### 1.4、AddDownload

void CDownloadQueue::AddDownload(CPartFile\* newfile,bool paused) {

// Barry - Add in paused mode if required

if (paused)

newfile->PauseFile();

SetAutoCat(newfile);// HoaX\_69 / Slugfiller: AutoCat

filelist.AddTail(newfile);

SortByPriority();

CheckDiskspace();

theApp.emuledlg->transferwnd->GetDownloadList()->AddFile(newfile);

AddLogLine(true, GetResString(IDS\_NEWDOWNLOAD), newfile->GetFileName());

CString msgTemp;

msgTemp.Format(GetResString(IDS\_NEWDOWNLOAD) + \_T("\n"), newfile->GetFileName());

theApp.emuledlg->ShowNotifier(msgTemp, TBN\_DOWNLOADADDED);

ExportPartMetFilesOverview();

}

### 2、轮询下载队列：

CUploadQueue::UploadTimer()🡪CDownloadQueue::Process() 🡪CPartFile::Process()🡪 CUpDownClient::AskForDownload()

#### 2.1、CDownloadQueue::Process

///snow:CUploadQueue::UploadTimer()中调用

void CDownloadQueue::Process(){

ProcessLocalRequests(); // send src requests to local server

///snow:选择下载速度，-200之间

uint32 downspeed = 0;

uint64 maxDownload = thePrefs.GetMaxDownloadInBytesPerSec(true);

if (maxDownload != UNLIMITED\*1024 && datarate > 1500){

downspeed = (UINT)((maxDownload\*100)/(datarate+1));

if (downspeed < 50)

downspeed = 50;

else if (downspeed > 200)

downspeed = 200;

}

///snow:avarage\_dr\_list起什么作用呢？一个CList<TransferredData>，包含两个成员变量：datalen和timestamp，计算平均下载速度

while(avarage\_dr\_list.GetCount()>0 && (GetTickCount() - avarage\_dr\_list.GetHead().timestamp > 10\*1000) ) ///snow:自开始计时超过秒了，把秒前的统计数据去除

m\_datarateMS-=avarage\_dr\_list.RemoveHead().datalen;

if (avarage\_dr\_list.GetCount()>1){

datarate = (UINT)(m\_datarateMS / avarage\_dr\_list.GetCount());

} else {

datarate = 0;

}

uint32 datarateX=0;

udcounter++; ///snow:它又起什么作用呢？定时器每ms调用Process()一次，udcounter++，udcounter应该是计时用，每.5秒进行服务器列表统计，每一秒发送一个UDPPakcet

theStats.m\_fGlobalDone = 0;

theStats.m\_fGlobalSize = 0;

theStats.m\_dwOverallStatus=0;

//filelist is already sorted by prio, therefore I removed all the extra loops..

///snow:遍历filelist，调用CPartFile::Process()

for (POSITION pos = filelist.GetHeadPosition();pos != 0;){

CPartFile\* cur\_file = filelist.GetNext(pos);

// maintain global download stats

theStats.m\_fGlobalDone += (uint64)cur\_file->GetCompletedSize();

theStats.m\_fGlobalSize += (uint64)cur\_file->GetFileSize();

if (cur\_file->GetTransferringSrcCount()>0) ///snow:m\_anStates[DS\_DOWNLOADING]

theStats.m\_dwOverallStatus |= STATE\_DOWNLOADING;

if (cur\_file->GetStatus()==PS\_ERROR)

theStats.m\_dwOverallStatus |= STATE\_ERROROUS;

if (cur\_file->GetStatus() == PS\_READY || cur\_file->GetStatus() == PS\_EMPTY){ ///snow:什么时候赋值呢？CPartFile::Init()中置为PS\_EMPTY

datarateX += cur\_file->Process(downspeed, udcounter);

}

else{

//This will make sure we don't keep old sources to paused and stoped files..

cur\_file->StopPausedFile();

}

}

TransferredData newitem = {datarateX, ::GetTickCount()};

avarage\_dr\_list.AddTail(newitem);

m\_datarateMS+=datarateX;

if (udcounter == 5){ ///snow:定时器调用Process()5次了

if (theApp.serverconnect->IsUDPSocketAvailable()){

if((!lastudpstattime) || (::GetTickCount() - lastudpstattime) > UDPSERVERSTATTIME){

lastudpstattime = ::GetTickCount();

theApp.serverlist->ServerStats();

}

}

}

if (udcounter == 10){ ///snow:定时器调用Process()10次了，重新开始计时

udcounter = 0;

if (theApp.serverconnect->IsUDPSocketAvailable()){

if ((!lastudpsearchtime) || (::GetTickCount() - lastudpsearchtime) > UDPSERVERREASKTIME)

SendNextUDPPacket();

}

}

CheckDiskspaceTimed();

// ZZ:DownloadManager -->

if((!m\_dwLastA4AFtime) || (::GetTickCount() - m\_dwLastA4AFtime) > MIN2MS(8)) {

theApp.clientlist->ProcessA4AFClients();

m\_dwLastA4AFtime = ::GetTickCount();

}

// <-- ZZ:DownloadManager

}

#### 2.2、CPartFile::Process

///snow:CDownloadQueue::Process()中调用，参数reducedownload：下载速度控制，参数icounter:0-10之间，表示是计时开始后第几次调用，每ms调用一次

uint32 CPartFile::Process(uint32 reducedownload, UINT icounter/\*in percent\*/)

{

if (thePrefs.m\_iDbgHeap >= 2)

ASSERT\_VALID(this);

UINT nOldTransSourceCount = GetSrcStatisticsValue(DS\_DOWNLOADING); ///snow:nOldTransSourceCount=m\_anStates[0]

DWORD dwCurTick = ::GetTickCount();

if (dwCurTick < m\_nLastBufferFlushTime)

{

ASSERT( false );

m\_nLastBufferFlushTime = dwCurTick;

}

// If buffer size exceeds limit, or if not written within time limit, flush data

if ((m\_nTotalBufferData > thePrefs.GetFileBufferSize()) || (dwCurTick > (m\_nLastBufferFlushTime + thePrefs.GetFileBufferTimeLimit())))

{

// Avoid flushing while copying preview file

if (!m\_bPreviewing)

FlushBuffer(); ///snow:SetStatus(PS\_READY)

}

datarate = 0;

// calculate datarate, set limit etc.

if(icounter < 10)

{

uint32 cur\_datarate;

for(POSITION pos = m\_downloadingSourceList.GetHeadPosition();pos!=0;) ///snow:AddDownloadingSource()中添加到队尾

{

CUpDownClient\* cur\_src = m\_downloadingSourceList.GetNext(pos);

if (thePrefs.m\_iDbgHeap >= 2)

ASSERT\_VALID( cur\_src );

if(cur\_src && cur\_src->GetDownloadState() == DS\_DOWNLOADING) ///snow:正从该客户端处下载

{

ASSERT( cur\_src->socket );

if (cur\_src->socket)

{

cur\_src->CheckDownloadTimeout();

cur\_datarate = cur\_src->CalculateDownloadRate(); ///snow:计算从该客户端下载的速度

datarate+=cur\_datarate;

if(reducedownload) ///snow:下载速度，控制下载速度

{

uint32 limit = reducedownload\*cur\_datarate/1000;

if(limit<1000 && reducedownload == 200)

limit +=1000;

else if(limit<200 && cur\_datarate == 0 && reducedownload >= 100)

limit = 200;

else if(limit<60 && cur\_datarate < 600 && reducedownload >= 97)

limit = 60;

else if(limit<20 && cur\_datarate < 200 && reducedownload >= 93)

limit = 20;

else if(limit<1)

limit = 1;

cur\_src->socket->SetDownloadLimit(limit);

if (cur\_src->IsDownloadingFromPeerCache() && cur\_src->m\_pPCDownSocket && cur\_src->m\_pPCDownSocket->IsConnected())

cur\_src->m\_pPCDownSocket->SetDownloadLimit(limit);

}

}

}

}

}

else ///snow:icounter=10

{

bool downloadingbefore=m\_anStates[DS\_DOWNLOADING]>0;

// -khaos--+++> Moved this here, otherwise we were setting our permanent variables to 0 every tenth of a second...

memset(m\_anStates,0,sizeof(m\_anStates));

memset(src\_stats,0,sizeof(src\_stats));

memset(net\_stats,0,sizeof(net\_stats));

UINT nCountForState;

for (POSITION pos = srclist.GetHeadPosition(); pos != NULL;) ///snow:AddSource()->CDownloadQueue::CheckAndAddSource()

{

CUpDownClient\* cur\_src = srclist.GetNext(pos);

if (thePrefs.m\_iDbgHeap >= 2)

ASSERT\_VALID( cur\_src );

// BEGIN -rewritten- refreshing statistics (no need for temp vars since it is not multithreaded)

nCountForState = cur\_src->GetDownloadState();

//special case which is not yet set as downloadstate

if (nCountForState == DS\_ONQUEUE)

{

if( cur\_src->IsRemoteQueueFull() )

nCountForState = DS\_REMOTEQUEUEFULL;

}

// this is a performance killer -> avoid calling 'IsBanned' for gathering stats

//if (cur\_src->IsBanned())

// nCountForState = DS\_BANNED;

if (cur\_src->GetUploadState() == US\_BANNED) // not as accurate as 'IsBanned', but way faster and good enough for stats.

nCountForState = DS\_BANNED;

if (cur\_src->GetSourceFrom() >= SF\_SERVER && cur\_src->GetSourceFrom() <= SF\_PASSIVE)

++src\_stats[cur\_src->GetSourceFrom()];

if (cur\_src->GetServerIP() && cur\_src->GetServerPort())

{

net\_stats[0]++;

if(cur\_src->GetKadPort())

net\_stats[2]++; ///snow:双网都连通

}

if (cur\_src->GetKadPort())

net\_stats[1]++; ///snow:Kad连通

ASSERT( nCountForState < sizeof(m\_anStates)/sizeof(m\_anStates[0]) );

m\_anStates[nCountForState]++; ///snow:总共15种状态

switch (cur\_src->GetDownloadState())

{

case DS\_DOWNLOADING:{

ASSERT( cur\_src->socket );

if (cur\_src->socket)

{

cur\_src->CheckDownloadTimeout();

uint32 cur\_datarate = cur\_src->CalculateDownloadRate();

datarate += cur\_datarate;

if (reducedownload && cur\_src->GetDownloadState() == DS\_DOWNLOADING)

{

uint32 limit = reducedownload\*cur\_datarate/1000; //(uint32)(((float)reducedownload/100)\*cur\_datarate)/10;

if (limit < 1000 && reducedownload == 200)

limit += 1000;

else if(limit<200 && cur\_datarate == 0 && reducedownload >= 100)

limit = 200;

else if(limit<60 && cur\_datarate < 600 && reducedownload >= 97)

limit = 60;

else if(limit<20 && cur\_datarate < 200 && reducedownload >= 93)

limit = 20;

else if (limit < 1)

limit = 1;

cur\_src->socket->SetDownloadLimit(limit);

if (cur\_src->IsDownloadingFromPeerCache() && cur\_src->m\_pPCDownSocket && cur\_src->m\_pPCDownSocket->IsConnected())

cur\_src->m\_pPCDownSocket->SetDownloadLimit(limit);

}

else{

cur\_src->socket->DisableDownloadLimit();

if (cur\_src->IsDownloadingFromPeerCache() && cur\_src->m\_pPCDownSocket && cur\_src->m\_pPCDownSocket->IsConnected())

cur\_src->m\_pPCDownSocket->DisableDownloadLimit();

}

}

break;

}

// Do nothing with this client..

case DS\_BANNED:

break;

// Check if something has changed with our or their ID state..

case DS\_LOWTOLOWIP:

{

// To Mods, please stop instantly removing these sources..

// This causes sources to pop in and out creating extra overhead!

//Make sure this source is still a LowID Client..

if( cur\_src->HasLowID() )

{

//Make sure we still cannot callback to this Client..

if( !theApp.CanDoCallback( cur\_src ) )

{

//If we are almost maxed on sources, slowly remove these client to see if we can find a better source.

if( ((dwCurTick - lastpurgetime) > SEC2MS(30)) && (this->GetSourceCount() >= (GetMaxSources()\*.8 )) )

{

theApp.downloadqueue->RemoveSource( cur\_src );

lastpurgetime = dwCurTick;

}

break;

}

}

// This should no longer be a LOWTOLOWIP..

cur\_src->SetDownloadState(DS\_ONQUEUE);

break;

}

case DS\_NONEEDEDPARTS:

{

// To Mods, please stop instantly removing these sources..

// This causes sources to pop in and out creating extra overhead!

if( (dwCurTick - lastpurgetime) > SEC2MS(40) ){

lastpurgetime = dwCurTick;

// we only delete them if reaching the limit

if (GetSourceCount() >= (GetMaxSources()\*.8 )){

theApp.downloadqueue->RemoveSource( cur\_src );

break;

}

}

// doubled reasktime for no needed parts - save connections and traffic

if (cur\_src->GetTimeUntilReask() > 0)

break;

cur\_src->SwapToAnotherFile(\_T("A4AF for NNP file. CPartFile::Process()"), true, false, false, NULL, true, true); // ZZ:DownloadManager

// Recheck this client to see if still NNP.. Set to DS\_NONE so that we force a TCP reask next time..

cur\_src->SetDownloadState(DS\_NONE);

break;

}

case DS\_ONQUEUE:

{

// To Mods, please stop instantly removing these sources..

// This causes sources to pop in and out creating extra overhead!

if( cur\_src->IsRemoteQueueFull() )

{

if( ((dwCurTick - lastpurgetime) > MIN2MS(1)) && (GetSourceCount() >= (GetMaxSources()\*.8 )) )

{

theApp.downloadqueue->RemoveSource( cur\_src );

lastpurgetime = dwCurTick;

break;

}

}

//Give up to 1 min for UDP to respond.. If we are within one min of TCP reask, do not try..

if (theApp.IsConnected() && cur\_src->GetTimeUntilReask() < MIN2MS(2) && cur\_src->GetTimeUntilReask() > SEC2MS(1) && ::GetTickCount()-cur\_src->getLastTriedToConnectTime() > 20\*60\*1000) // ZZ:DownloadManager (one resk timestamp for each file)

cur\_src->UDPReaskForDownload();

}

case DS\_CONNECTING:

case DS\_TOOMANYCONNS:

case DS\_TOOMANYCONNSKAD:

case DS\_NONE:

case DS\_WAITCALLBACK:

case DS\_WAITCALLBACKKAD:

{

if (theApp.IsConnected() && cur\_src->GetTimeUntilReask() == 0 && ::GetTickCount()-cur\_src->getLastTriedToConnectTime() > 20\*60\*1000) // ZZ:DownloadManager (one resk timestamp for each file)

{

if(!cur\_src->AskForDownload()) // NOTE: This may \*delete\* the client!!

break; //I left this break here just as a reminder just in case re rearange things..

}

break;

}

}

}

if (downloadingbefore!=(m\_anStates[DS\_DOWNLOADING]>0))

NotifyStatusChange();

if( GetMaxSourcePerFileUDP() > GetSourceCount()){

if (theApp.downloadqueue->DoKademliaFileRequest() && (Kademlia::CKademlia::GetTotalFile() < KADEMLIATOTALFILE) && (dwCurTick > m\_LastSearchTimeKad) && Kademlia::CKademlia::IsConnected() && theApp.IsConnected() && !stopped){ //Once we can handle lowID users in Kad, we remove the second IsConnected

//Kademlia

theApp.downloadqueue->SetLastKademliaFileRequest();

if (!GetKadFileSearchID())

{

Kademlia::CSearch\* pSearch = Kademlia::CSearchManager::PrepareLookup(Kademlia::CSearch::FILE, true, Kademlia::CUInt128(GetFileHash()));

if (pSearch)

{

if(m\_TotalSearchesKad < 7)

m\_TotalSearchesKad++;

m\_LastSearchTimeKad = dwCurTick + (KADEMLIAREASKTIME\*m\_TotalSearchesKad);

pSearch->SetGUIName(GetFileName());

SetKadFileSearchID(pSearch->GetSearchID());

}

else

SetKadFileSearchID(0);

}

}

}

else{

if(GetKadFileSearchID())

{

Kademlia::CSearchManager::StopSearch(GetKadFileSearchID(), true);

}

}

// check if we want new sources from server

if ( !m\_bLocalSrcReqQueued && ((!m\_LastSearchTime) || (dwCurTick - m\_LastSearchTime) > SERVERREASKTIME) && theApp.serverconnect->IsConnected()

&& GetMaxSourcePerFileSoft() > GetSourceCount() && !stopped

&& (!IsLargeFile() || (theApp.serverconnect->GetCurrentServer() != NULL && theApp.serverconnect->GetCurrentServer()->SupportsLargeFilesTCP())))

{

m\_bLocalSrcReqQueued = true;

theApp.downloadqueue->SendLocalSrcRequest(this);

}

count++;

if (count == 3){

count = 0;

UpdateAutoDownPriority();

UpdateDisplayedInfo();

UpdateCompletedInfos();

}

}

if ( GetSrcStatisticsValue(DS\_DOWNLOADING) != nOldTransSourceCount ){

if (theApp.emuledlg->transferwnd->GetDownloadList()->curTab == 0)

theApp.emuledlg->transferwnd->GetDownloadList()->ChangeCategory(0);

else

UpdateDisplayedInfo(true);

if (thePrefs.ShowCatTabInfos() )

theApp.emuledlg->transferwnd->UpdateCatTabTitles();

}

return datarate;

}

#### 2.3、AskForDownload

bool CUpDownClient::AskForDownload()

{

if (m\_bUDPPending)

{

m\_nFailedUDPPackets++;

theApp.downloadqueue->AddFailedUDPFileReasks();

}

m\_bUDPPending = false;

if (!(socket && socket->IsConnected())) // already connected, skip all the special checks

{

if (theApp.listensocket->TooManySockets())

{

if (GetDownloadState() != DS\_TOOMANYCONNS)

SetDownloadState(DS\_TOOMANYCONNS);

return true;

}

m\_dwLastTriedToConnect = ::GetTickCount();

// if its a lowid client which is on our queue we may delay the reask up to 20 min, to give the lowid the chance to

// connect to us for its own reask

if (HasLowID() && GetUploadState() == US\_ONUPLOADQUEUE && !m\_bReaskPending && GetLastAskedTime() > 0){

SetDownloadState(DS\_ONQUEUE);

m\_bReaskPending = true;

return true;

}

// if we are lowid <-> lowid but contacted the source before already, keep it in the hope that we might turn highid again

if (HasLowID() && !theApp.CanDoCallback(this) && GetLastAskedTime() > 0){

if (GetDownloadState() != DS\_LOWTOLOWIP)

SetDownloadState(DS\_LOWTOLOWIP);

m\_bReaskPending = true;

return true;

}

}

m\_dwLastTriedToConnect = ::GetTickCount();

SwapToAnotherFile(\_T("A4AF check before tcp file reask. CUpDownClient::AskForDownload()"), true, false, false, NULL, true, true);

SetDownloadState(DS\_CONNECTING);

return TryToConnect();

}

### 3、连接：

发现到各客户端的连接，参见客户端呼出连接过程

### 4、发送请求

下载连接建立之后

CUpDownClient::ConnectionEstablished()🡪SetDownloadState(DS\_CONNECTED); SendFileRequest();发出文件名请求OP\_REQUESTFILENAME--> CClientReqSocket::ProcessPacket()🡪OP\_REQFILENAMEANSWER --> CClientReqSocket::ProcessPacket()🡪 CUpDownClient::ProcessFileInfo() --> SendHashSetRequest()\SendStartupLoadReq--> OP\_HASHSETREQUEST\OP\_STARTUPLOADREQ--> CClientReqSocket::ProcessPacket()🡪 CUploadQueue::AddClientToQueue()->

1、一种情况是如果对方客户端经在下载了，则OP\_ACCEPTUPLOADREQ--> CClientReqSocket::ProcessPacket()🡪CUpDownClient:: ProcessAcceptUpload()--> StartDownload() -->SetDownloadState(DS\_DOWNLOADING), SendBlockRequests()🡪 OP\_REQUESTPARTS\_I64/ OP\_REQUESTPARTS🡪 CClientReqSocket::ProcessPacket()🡪CUpDownClient::AddReqBlock()🡪m\_BlockRequests\_queue.AddTail(reqblock);

到这里告一段落，下面将是对方排队发送过程

2、对方客户端还没开始下载：AddUpNextClient()🡪SendPacket(OP\_ACCEPTUPLOADREQ,0)，后续操作如上

### 5、下载

等待对方上传，见文件上传部分

CClientReqSocket::ProcessPacket() case OP\_SENDINGPART🡪

CClientReqSocket::ProcessPacket()

case OP\_COMPRESSEDPART:

case OP\_SENDINGPART\_I64:

case OP\_COMPRESSEDPART\_I64: 🡪

CUpDownClient::ProcessBlockPacket()🡪CPartFile::WriteToBuffer(),CUpDownClient::SendBlockRequests()

??OP\_END\_OF\_DOWNLOAD 没看见哪里发出这个OPCode

### 6、检查Hash

# 文件上传

1、请求下载文件的客户发起连接后，已将下载请求存入队列，当准备上传的客户端准备上传时发现连接已断开：AddClientToQueue🡪 CUploadQueue::AddUpNextClient🡪 SetUploadState(US\_CONNECTING);🡪TryToConnect(true)🡪

CUpDownClient::ConnectionEstablished() case US\_CONNECTING: 🡪SetUploadState(US\_UPLOADING)🡪SendPacket(OP\_ACCEPTUPLOADREQ,0)

AddClientToQueue():在添加Client时区分四种情况：一种是LowID客户端再次连接成功的时候直接调用AddUpNextClient，添加到uploadinglist，一种是.emulecollection文件及长度不足50时，直接调用AddUpNextClient，添加到uploadinglist，一种是waitinglist为空时，直接调用AddUpNextClient，添加到uploadinglist，前面这三种情况添加的客户端均不经过排队，直接到上传队列，发送信息包，等待上传；除了这三种情况外，客户端将被添加到waiting list，由Process()轮询时按客户端的信誉等级从waitinglist移动到uploadinglist。

2、连接已建立，尚未断开：

CClientReqSocket::ProcessPacket() case OP\_HELLO:🡪 CUpDownClient::ConnectionEstablished()🡪SendFileRequest();发出文件名请求OP\_REQUESTFILENAME--> CClientReqSocket::ProcessPacket()🡪OP\_REQFILENAMEANSWER --> CClientReqSocket::ProcessPacket()🡪 CUpDownClient::ProcessFileInfo() --> SendHashSetRequest()\SendStartupLoadReq--> OP\_HASHSETREQUEST\OP\_STARTUPLOADREQ--> CClientReqSocket::ProcessPacket()🡪 CUploadQueue::AddClientToQueue()->AddUpNextClient()🡪SendPacket(OP\_ACCEPTUPLOADREQ,0)

二、对方接收信息包，开始处理：CClientReqSocket::ProcessPacket()🡪CUpDownClient:: ProcessAcceptUpload()--> StartDownload() -->SetDownloadState(DS\_DOWNLOADING), SendBlockRequests()🡪 OP\_REQUESTPARTS\_I64/ OP\_REQUESTPARTS🡪 CClientReqSocket::ProcessPacket()🡪CUpDownClient::AddReqBlock()🡪m\_BlockRequests\_queue.AddTail(reqblock);

两种途径开始发送数据

1、CUploadQueue::Process()🡪CUpDownClient::SendBlockData()🡪

2、CUploadQueue::HSUploadTimer()🡪

CUpDownClient::CreateNextBlockPackage() Requested\_Block\_Struct\* currentblock = m\_BlockRequests\_queue.GetHead();🡪CreateStandartPackets()/ CreatePackedPackets()

🡪 OP\_SENDINGPART/ OP\_SENDINGPART\_I64/ OP\_COMPRESSEDPART\_I64/ OP\_COMPRESSEDPART

Process()

1、CUpDownClient::ConnectionEstablished()

switch(GetUploadState())

{

case US\_CONNECTING:

if (theApp.uploadqueue->IsDownloading(this))

{

SetUploadState(US\_UPLOADING);

if (thePrefs.GetDebugClientTCPLevel() > 0)

DebugSend("OP\_\_AcceptUploadReq", this);

Packet\* packet = new Packet(OP\_ACCEPTUPLOADREQ,0);

theStats.AddUpDataOverheadFileRequest(packet->size);

SendPacket(packet,true);

}

}

2、CUploadQueue::AddUpNextClient

newclient->SetUploadState(US\_CONNECTING);🡪TryToConnect(true)

3、

# DownloadQueue类

一、CheckAndAddKnownSource（A）与CheckAndAddSource(B)

1、A中source为已知Source，所以不delete，不检查是否是本机IP

2、A中检查是否是LANIP，A与B都未进行IP filter

3、B执行了theApp.clientlist->AttachToAlreadyKnown(&source,0)，而A执行

sender->srclist.AddTail(source);

source->SetSourceFrom(SF\_PASSIVE);

二、filelist中存储着partfile，每个Partfile有个sourcelist，sourcelist存放着client

三、对filelist进行优先级堆排序

四、检查磁盘

[eMue片选择算法](http://hustlg.bokee.com/)

作者： [hustlg](http://hustlg.bokee.com/)2008-03-24 13:07分类：[默认分类](http://hustlg.bokee.com/category/0/1)标签：

由于从事eMule协议的相关开发已经有一段时间了，最近经常收到一些网友的邮件，探讨p2p网络中片选择的一些问题。  比如，在p2p假如一个文件被分为很多块，当有很多个client请求时，谁向谁请求哪些文件块，因为client和文件的提供者都是不断变化的啊。不知道emule是怎样处理这个问题的。就某一个时刻而言，client和文件的提供者是固定的，以什么样的规则请求文件块呢？

对一个下载者来说，在选择下一个被下载的片断时，通常选择的是它的peers们所拥有的最少的那个片断，也就是所谓的"最少优先"。确保了每个下载者都拥有它的peers们最希望得到的那些片断，从而一旦有需要，上载就可以开始。这也确保了那些越普通的片断越放在最后下载，从而减少了这样一种可能性，即某个peer当前正提供上载，而随后却没有任何的被别人感兴趣的片断了。也就说，每个peer都优先选择整个系统中最少的那些片断去下载，而那些在系统中相对较多的片断，放在后面下载，这样，整个系统就趋向于一种更优的状态。如果不用这种算法，大家都去下载最多的那些片断，那么这些片断就会在系统中分布的越来越多，而那些在系统中相对较少的片断仍然很少，最后，某些 peer 就不再拥有其它 peer 感兴趣的片断了，那么系统的参与者越来越少，整个系统的性能就下降。通常在下载的过程分为几个阶段，第一片选择，最后阶段模式，**片选择要遵循的一个基本规则：一旦请求了某个片断的子片断，那么该片断剩下的子片断优先被请求。这样，可以尽可能快的获得一个完整的片断。**

**具体对于emule来说，eMule仔细挑选选块的下载顺序。下面是emule网络中片选择的规则，具体的实现可以参见源码的CpartFile.cpp文件**

每个文件被分成9.28M的块，每部分分成180KB的片。  
块下载的顺序是由发送请求文件块消息（6.4.4节）的下载客户端决定。下载客户端可以在任何给定时刻从各个源中下载一个单独的文件块，所有从相同源中请求的片都在同一个块中。下面的原理（以这个顺序）应用于下载块等级：  
1.（可获得的）大片的频率，尽可能快的下载非常稀少的大片来形成一个新的源。  
2.用来预览的块（最初+最后的大片），预览或检查文件（比如，电影、mp3）  
3.请求状态（过程中下载），尝试向每个源询问其它的大片。在所有源之间扩散请求。  
4.完成（未到某种程度的完成），在开始下载另一个时应该完成获得部分的大片  
   
**频率标准定义了三个区域：非常稀少、稀少和一般。在每个区域里，标准有特定的权重，用来计算块等级。较低等级的块先下载。下面的列表根据上面的原理指定文件等级范围：  
l         0-9999 - 不请求和请求非常稀少的块  
l         10000-19999 - 不请求稀少和预览块  
l         20000-29999 - 不请求大部分完成的一般的块  
l         30000-39999 - 请求的稀少和预览的块  
l         40000-49999 - 请求的没有完成的一般的块**   
这个算法通常选择第一个最稀少的块。然而，部分完成的块，接近完成的，也可能被选中。对于一般的块，在不同的源之间扩散下载。理论上是可以统计出所有文件块的拥有者的，但是在实际情况下只能达到一个局部最优的效果，也就是小世界理论所说的。一个Peer通过服务器或者KAD网络获得对方的Peer信息，然后交换Peer的片信息，在他的邻居范围内就可以确定一个片的请求频率的。在emule中具体的片选择策略的实现是在：（emule0.48a官方版本的算法，至于其他的修改版本不再次讨论之列，不过也大通小异）