# UploadBandwidthThrottler工作流程

一、各变量的含义

1、m\_StandardOrder\_list：标准列表，可发送控制包和标准包

2、

二、节流阀工作原理

1、统计一个循环经过的时间timeSinceLastLoop，再统计一个循环发送的字节数bytesToSpend，计算出传输速率datarate=bytesToSpend/timeSinceLastLoop，与允许传输速率比较，快了则休眠一段时间。需要注意的是，设定的速率是以秒为单位的，而循环里的计时则是以毫秒为单位的，字节数也是有以Byte为单位的，有以KB为单位的，需要换算。

假设限速300K/S，则换算成B/ms则为300\*1024/10000=307b/ms。如果前200ms传输了200K，则速率达到了1MB/S，还剩100K还需100ms，休眠时间则为1000-300=700毫秒。当然一个循环需要时间没有这么多，（调试跟踪到底需要多少时间）假设需要20ms，则能发送的字节为6K。如果上一个循环发送了7K字节，则本循环仅需要发送5K，需要时间为5/6\*20=17ms，则本次循环需要休眠的时间为3ms。而如果上一个循环发送了5K字节，则本循环需要发送7K，则本次循环不需要休眠，相反可以提高传输速率到350K/S。

假设限速300K/S，上一个循环到现在经历了20ms，则理论应该发送6K，

2、计时部分

2.1 thisLoopTick 在while()中定义const DWORD thisLoopTick = timeGetTime();代码块常变量，初始化之后没再赋值

2.2 lastLoopTick

2.2.1RunInternal局部变量，初始化DWORD lastLoopTick = timeGetTime();

2.2.2

UINT UploadBandwidthThrottler::RunInternal()

{

DWORD lastLoopTick = timeGetTime();

……

While(doRun)

{

DWORD timeSinceLastLoop = timeGetTime() - lastLoopTick;

……..

const DWORD thisLoopTick = timeGetTime();

if(allowedDataRate != \_UI32\_MAX) //在有速度限制的情况下，

{ …….

else if(\_I64\_MAX/timeSinceLastLoop > allowedDataRate && \_I64\_MAX-allowedDataRate\*timeSinceLastLoop > realBytesToSpend)

{

if(timeSinceLastLoop > sleepTime + 2000) ///snow:超过2秒

{ ……

timeSinceLastLoop = sleepTime + 2000;

lastLoopTick = thisLoopTick - timeSinceLastLoop;

}

…..…

}

………..

}

lastLoopTick = thisLoopTick;

if(bytesToSpend >= 1 || allowedDataRate == 0)

{

发送数据 ，分四批发送，首先发送控制包信息，其次发送m\_StandardOrder\_list中长时间没发送过数据的socket，第三再正常发送m\_StandardOrder\_list中各socket，最后如果带宽没用完，就再次发送m\_StandardOrder\_list中的各socket，尽量用完带宽。如果实在用不完，则允许在下一循环中多发送999字节。

}

….

}

…….

}

2.3 在循环开始之前，记下当前时刻，存入lastLoopTick，作为初始值（其实有必要吗？）；每次循环开始执行，用循环开始时间，减去lastLoopTick，为运行已过时间，存入timeSinceLastLoop。因为第一次循环时，计算timeSinceLastLoop的语句紧挨着lastLoopTick赋值语句，所以timeSinceLastLoop=0。在循环执行中，当准备发送数据 的时候，记下当前时刻，用当前时刻的值去更新lastLoopTick，这样在第二次或以后的循环时，timeSinceLastLoop的值就相当于上次循环时发送数据的时间。当需要限速时(allowedDataRate != \_UI32\_MAX)，计算spentBytes/timeSinceLastLoop就可以获取当前上传速度了。

# UploadSpeedSense

automatically finds the best upload speed for your connection.

The goal is to make eMule work right out of the box, without need for configuration of upload speed. The users should be able to just leave the upload speed limit at default 0 (unlimited) in preferences and relax. If they use other programs that want bandwidth, UploadSpeedSense will automatically lower the upload limit for eMule while the other transfer is going on. When the transfer is done, UploadSpeedSense raises the upload limit back to normal speed. UploadSpeedSense will not work for multihomed hosts.

This version contain only UploadSpeedSense, and no other of the ZZ features. I've created this separate version to make it easier to evaluate this feature and to hopefully get it included in the official version.

UploadSpeedSense is based on the DynUp idea, and in fact uses a few lines of code from DynUp. Thanks!

## How UploadSpeedSense works

* Uses traceroute to find one of your ISP's routers to ping. The tracerouting finds the last common host for 10 randomly chosen ips in your server list and known clients. TTL is increased one step at a time, and each host is pinged until the returned ip differs. This means none of the hosts are actually pinged, only your ISPs infrastructure gets the pings. Last common host is saved, and one ip of the routers one hop furhter than that last common host (this is the ping ip, the ip that pings will be sent to). During this phase the lower right corner of eMule will say "Preparing...". When a good host to ping has been found, the ip of that host is reported in the debug log. There's currently no way to manually set which host should be pinged. This may be changed in a future version.
* UploadSpeedSense now regurarly pings the ping ip, but it sets the TTL one to short, to get the ip from the last common host back. It uses this ping to measure latency for the connection. When the ping gets to high, it lowers the upload limit. When the ping gets low, it raises the upload limit. You can see the current ping in the lower right corner in eMule (statusbar).
* If the responding ip for the set TTL changes (remember, we set TTL one shorter than the number of hops to the ip we ping), then UploadSpeedSense assumes that the topology has changed, and will redo the tracerouting phase to find the new topology. This may happen if the computer has been reconnected to the internet, i.e a modem redial.

Preferences

You might need to experiment with the defaults in the Extended settings page in preferences.

These are the defaults:

Ping Tolerance: 800%  
Up Slowness: 1000  
Down Slowness: 1000  
Max number of pings for average: 1

Higher ping tolerance will make it accept a higher ping before it lowers the speed.

The slowness values controls how fast/slow the speed is changed to match the ping. Higher values makes it change the speed slower. Lower values makes it change the speed faster. The higher values, the smoother graph, and the slower it reacts to ping changes.

"Max number of pings for average" controls how many pings it will average to calculate the ping. Higher value will make it react less, and slower, to ping changes. Too high value here will make the upload graph wave-formed.

## How to tweak

First set the upload limit value to the highest speed you want to allow eMule to upload to. UploadSpeedSense will never raise upload limit to a higher value than this value. If you want UploadSpeedSense to control speed completely, disable upload limit.

**The UploadSpeedSense are available in Preferences->Extended settings.**

**Find best upload limit automatically:** Check this to enable.

**Lowest allowed upload speed:** the lowest speed that UploadSpeedSense is allowed to set.

**Ping tolerance (%)**: This value is only used if "Method for ping tolerance" is set to "Percent (%)". Start at 500% and then try to raise it 100 or lower it 100 and see the difference. The lower this value, the lower upload speed you will get. 100% is the lowest value possible, and will probably make USS lower your upload limit to whatever MIN SPEED is set to.

**Ping tolerance (ms):** If you prefer, you can set the actual ping value you want. This value is only used if "Method for ping tolerance" is set to "Milliseconds (ms)".

**Going up/down slowness:** These values effect how fast the speed is adjusted. You will probably want both of these to both be set to one single value. Start these at 1000. If you get a roller coaster in the upload graph, try to raise them to 2000, then 3000, etc. The HIGHER the value, the SLOWER USS changes the upload limit. Too high value here will more or less prevent USS from changing upload speed at all.

**Number of pings for average:** Should ALWAYS be 1. I've never gotten good results with any other value.

# AICH

本词条缺少**信息栏**，补充相关内容使词条更完整，还能快速升级，赶紧来编辑吧！

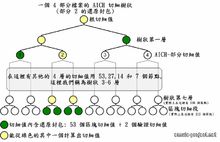
AICH损坏的处理，eMule 使用各种的方式来确保文件在网络共享及下载没有错误. 万一错误发生, 称为损坏, eMule 有进阶功能以最小的额外重新下载资料量来修正这个损坏.文件[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC)和 ICH- 智慧型损坏处理。

## 目录

1. 1 [AICH - 进阶智慧型损坏处理](http://baike.baidu.com/link?url=-8ilqN8gHK573LLNfS6yXMCeYP4t-4x9onWjTuRcfC6Nua9pwpHDz0we6yTaIbShzL9TjfbBuqtGlhu3pIZBHa#1)
2. ▪ [标准的 ICH](http://baike.baidu.com/link?url=-8ilqN8gHK573LLNfS6yXMCeYP4t-4x9onWjTuRcfC6Nua9pwpHDz0we6yTaIbShzL9TjfbBuqtGlhu3pIZBHa#1_1)
3. ▪ [从一个损坏还原](http://baike.baidu.com/link?url=-8ilqN8gHK573LLNfS6yXMCeYP4t-4x9onWjTuRcfC6Nua9pwpHDz0we6yTaIbShzL9TjfbBuqtGlhu3pIZBHa#1_2)
4. ▪ [根哈希值的信任](http://baike.baidu.com/link?url=-8ilqN8gHK573LLNfS6yXMCeYP4t-4x9onWjTuRcfC6Nua9pwpHDz0we6yTaIbShzL9TjfbBuqtGlhu3pIZBHa#1_3)
5. 2 [注意事项](http://baike.baidu.com/link?url=-8ilqN8gHK573LLNfS6yXMCeYP4t-4x9onWjTuRcfC6Nua9pwpHDz0we6yTaIbShzL9TjfbBuqtGlhu3pIZBHa#2)

## AICHAICH - 进阶智慧型损坏处理

### AICH标准的 ICH

标准的 ICH 是相当有效的，虽然它有它的限制，只在整个 9.28 MB 能被验证，并且没有完美的区块. 假如超过[](http://baike.baidu.com/pic/AICH/1879485/0/9f1011b38ff310b4d8335a4f?fr=lemma&ct=single)

一个以上的位置是损坏或是恶意客户端一再的散布损坏的资料或甚至是假的，部分[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC), ICH 再也没有能力去处理.

在这里 AICH 将会考虑完全的完整资料用一个最小重新下载量或者由建立非常完美的[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC)来管理.

根[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC), 区块哈希值 & AICH 片段哈希值

这次我们的起点是在一个文件的 9.28 MB 部分. 每个部分是被分割成 180 KB 的区块, 在每个部分将会产生 53 个区块并且每个区块使用 SHA1 切细运算方式计算出[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC). 那些值称为 区块哈希值 并且根据一个低标准的一个完整 AICH 片段哈希值.

在上面的图片是显示一个完整的[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC)树状图如何建立在一个完整 4 部分文件的区块. 每个部分包含 53 个区块产生出 212 个 区块哈希值 其中建立在一个切细树状的第七层直到 根哈希值 到达时. 这整个树状称为 AICH 片段哈希值.

绿色和黄色点显示小型的 区块[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC) 到 根哈希值 之数学相关性. 这个表示假如我们有一个可信任的根哈希值整个树状能被逆向的来验证它.

eMule 能建立包含根[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC)的链接, 例如

ed2k://|file|name|12043984|6744FC42EDA527B27F0B2F2538728B3E| h=A2NWOTYURUU3P3GCUB6KCNW3FTYYELQB|/

其中 h= 是 根哈希值. 由提供一个可信赖的 根哈希值 并发布它应该能有明显的改善文件的损坏抵抗性. 阅读 根哈希值的信任

### AICH从一个损坏还原

无论何时 eMule 在一个部分侦测到一个损坏它需要用一个完整 AICH [哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC)资料从随便一个客户端中取得一个还原[封包](http://baike.baidu.com/item/%E5%B0%81%E5%8C%85). 这个还原封包包含在切细树状整体损坏部分的全部 53 个 区块哈希值 和一个 验证哈希值 的号码. 上面图片显示一个 4 部分文件的一个还原封包. 验证哈希值 的号码是由文件的分割部分数量来决定 (2^x >= '部分数量', 用 x = 验证哈希值号码).

接收还原[封包](http://baike.baidu.com/item/%E5%B0%81%E5%8C%85)之后 eMule 检查 验证[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC) 逆向确认它的根哈希值. 假如它们相符, eMule 从还原封包的 区块哈希值 逆向检查损坏部分的全部 53 个区块. AICH 能还原全部区块用它们的 区块哈希值 逆向相符来让只有损坏的区块重新下载.

在记录中一个成功的还原看起来会像下面列出的:

09.09.2004 02:43:43: 已下载部分 6 损坏了 :( ([file])

09.09.2004 02:43:46: AICH 成功的还原 8.22 于 9.28 从部分 6 于 [file]

### AICH根哈希值的信任

最佳的方式是从有 根哈希值 的链接来下载. 假定这个链接的来源是可信任的根哈希值而一但受信任将会把这个文件的根哈希值储存在[磁盘](http://baike.baidu.com/item/%E7%A3%81%E7%9B%98).

假如不是由链接提供的根[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC)而是由文件的来源送出的 eMule 也会去信任这根哈希值. 它只会在一个 根哈希值 最少 10 个不同的来源送出相同的值和最少全部 92% 的来源同意这个值才会去相信它是真的. 因为这个 根哈希值 不是那么可靠它只有效于目前工作阶段并且不能储存也不能用 根哈希值 建链接.

一旦 eMule 建立整个 AICH 片段哈希值, 例如:文件已经完成, 它将开始传播 根哈希值 给其他的客户端.

## AICH注意事项

· 新释放或罕见的文件将也许没有足够的完整来源来产生一个可信任的 根哈希值. 建议释放文件时包含这个哈希值.

· 在一般情况下假如在那里没有 根[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC) 或甚至是一个伪造的 eMule 将能够成功下载并且完成这个文件. 而 AICH 特性不能使用在这种情况.

· 如同 AICH 片段[哈希值](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E5%80%BC)能非常大他们不储存在内存但存在 known2.met 并且只能做读取需求.

· AICH 将只能在 eMule [客户端](http://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF) v.44a 及更新版本有效但保留旧客户端的向下相容性.

# Log日志

一、构成：

class CLogFile

{

bool IsOpen() const;

const CString& GetFilePath() const;

bool SetFilePath(LPCTSTR pszFilePath);

void SetMaxFileSize(UINT uMaxFileSize);

bool SetFileFormat(ELogFileFormat eFileFormat);

bool Create(LPCTSTR pszFilePath, UINT uMaxFileSize = 1024\*1024, ELogFileFormat eFileFormat = Unicode);

bool Open();

bool Close();

bool Log(LPCTSTR pszMsg, int iLen = -1); ///snow:写入日志文件

bool Logf(LPCTSTR pszFmt, ...);

void StartNewLogFile();

}

enum EDebugLogPriority{

DLP\_VERYLOW = 0,

DLP\_LOW,

DLP\_DEFAULT,

DLP\_HIGH,

DLP\_VERYHIGH

};

// Log message type enumeration

#define LOG\_INFO 0

#define LOG\_WARNING 1

#define LOG\_ERROR 2

#define LOG\_SUCCESS 3

#define LOGMSGTYPEMASK 0x03

// Log message targets flags

#define LOG\_DEFAULT 0x00

#define LOG\_DEBUG 0x10

#define LOG\_STATUSBAR 0x20

#define LOG\_DONTNOTIFY 0x40

主要函数：Log.cpp

void AddLogTextV(UINT uFlags, EDebugLogPriority dlpPriority, LPCTSTR pszLine, va\_list argp);

Log、DebugLog系列函数，AddLogTextV()的不同参数调用版本

void Log(LPCTSTR pszLine, ...);

void LogError(LPCTSTR pszLine, ...);

void LogWarning(LPCTSTR pszLine, ...);

void Log(UINT uFlags, LPCTSTR pszLine, ...);

void LogError(UINT uFlags, LPCTSTR pszLine, ...);

void LogWarning(UINT uFlags, LPCTSTR pszLine, ...);

void DebugLog(LPCTSTR pszLine, ...);

void DebugLogError(LPCTSTR pszLine, ...);

void DebugLogWarning(LPCTSTR pszLine, ...);

void DebugLog(UINT uFlags, LPCTSTR pszLine, ...);

void DebugLogError(UINT uFlags, LPCTSTR pszLine, ...);

void DebugLogWarning(UINT uFlags, LPCTSTR pszLine, ...);

void LogV(UINT uFlags, LPCTSTR pszFmt, va\_list argp);

void AddLogLine(bool bAddToStatusBar, LPCTSTR pszLine, ...);

void AddDebugLogLine(bool bAddToStatusBar, LPCTSTR pszLine, ...);

void AddDebugLogLine(EDebugLogPriority Priority, bool bAddToStatusBar, LPCTSTR pszLine, ...);

enum ELogFileFormat

{

Unicode = 0,

Utf8

};

void CemuleDlg::AddLogText(UINT uFlags, LPCTSTR pszText)

在emuleDlg!=NULL时由AddLogTextV()调用，代替AddLogTextV()中的日志处理

// Elandal:ThreadSafeLogging -->

// thread safe log calls

CCriticalSection m\_queueLock; ///snow:线程安全，在更新列表时锁定

CTypedPtrList<CPtrList, SLogItem\*> m\_QueueDebugLog;

CTypedPtrList<CPtrList, SLogItem\*> m\_QueueLog;

// Elandal:ThreadSafeLogging <--

// Elandal:ThreadSafeLogging -->

// thread safe log calls

void QueueDebugLogLine(bool bAddToStatusBar, LPCTSTR line,...);

void QueueDebugLogLineEx(UINT uFlags, LPCTSTR line,...);

void HandleDebugLogQueue();

void ClearDebugLogQueue(bool bDebugPendingMsgs = false);

void QueueLogLine(bool bAddToStatusBar, LPCTSTR line,...);

void QueueLogLineEx(UINT uFlags, LPCTSTR line,...);

void HandleLogQueue();

void ClearLogQueue(bool bDebugPendingMsgs = false);

// Elandal:ThreadSafeLogging <--

二、流程

（一）、直接写入日志文件：

调用Log、DebugLog系列函数，比如：

AddDebugLogLine(DLP\_LOW, false, \_T("CEncryptedStreamSocket: Server %s preffered unsupported encryption method (%i)"), DbgGetIPString(), m\_dbgbyEncryptionRequested);

通过调用AddLogTextV或CEmuleDlg::AddLogText()生成日志记录，并根据选项设置决定是否写入日志文件

void AddLogTextV(UINT uFlags, EDebugLogPriority dlpPriority, LPCTSTR pszLine, va\_list argptr)

{

ASSERT(pszLine != NULL);

if ((uFlags & LOG\_DEBUG) && !(thePrefs.GetVerbose() && dlpPriority >= thePrefs.GetVerboseLogPriority()))

return;

TCHAR szLogLine[1000];

\_vsntprintf(szLogLine, \_countof(szLogLine), pszLine, argptr);

szLogLine[\_countof(szLogLine) - 1] = \_T('\0');

if (theApp.emuledlg)

theApp.emuledlg->AddLogText(uFlags, szLogLine);

else

{

TRACE(\_T("App Log: %s\n"), szLogLine);

TCHAR szFullLogLine[1060];

int iLen = \_sntprintf(szFullLogLine, \_countof(szFullLogLine), \_T("%s: %s\r\n"), CTime::GetCurrentTime().Format(thePrefs.GetDateTimeFormat4Log()), szLogLine);

if (iLen > 0)

{

if (!(uFlags & LOG\_DEBUG))

{

if (thePrefs.GetLog2Disk())

theLog.Log(szFullLogLine, iLen);

}

if (thePrefs.GetVerbose() && ((uFlags & LOG\_DEBUG) || thePrefs.GetFullVerbose()))

{

if (thePrefs.GetDebug2Disk())

theVerboseLog.Log(szFullLogLine, iLen);

}

}

}

}

CEmuleDlg::AddLogText()只是增加了窗口更新部分，主体部分是一致的。

（二）、先将日志记录存入队列，再将队列中的记录写入日志文件

1、调用CemuleApp::QueueLogLine系列函数将日志记录存入队列

void CemuleApp::QueueDebugLogLine(bool bAddToStatusbar, LPCTSTR line, ...)

{

if (!thePrefs.GetVerbose())

return;

m\_queueLock.Lock();

TCHAR bufferline[1000];

va\_list argptr;

va\_start(argptr, line);

int iLen = \_vsntprintf(bufferline, \_countof(bufferline), line, argptr);

va\_end(argptr);

if (iLen > 0)

{

SLogItem\* newItem = new SLogItem;

newItem->uFlags = LOG\_DEBUG | (bAddToStatusbar ? LOG\_STATUSBAR : 0);

newItem->line.SetString(bufferline, iLen);

m\_QueueDebugLog.AddTail(newItem);

}

m\_queueLock.Unlock();

}

将日志记录写入m\_QueueDebugLog或m\_QueueLog

2、然后调用HandleDebugLogQueue()系列函数将记录写入日志文件

(1)、在CemuleApp::InitInstance()中初始化了CUploadQueue对象

uploadqueue = new CUploadQueue();

并设定日志文件名

VERIFY( theLog.SetFilePath(thePrefs.GetMuleDirectory(EMULE\_LOGDIR, thePrefs.GetLog2Disk()) + \_T("eMule.log")) ); ///snow:日志文件名

VERIFY( theVerboseLog.SetFilePath(thePrefs.GetMuleDirectory(EMULE\_LOGDIR, false) + \_T("eMule\_Verbose.log")) );

(2)、在CUploadQueue构造函数中设定了定时器

VERIFY( (h\_timer = SetTimer(0,0,100,UploadTimer)) != NULL );

(3)、在定时器CUploadQueue::UploadTimer()中调用了HandleDebugLogQueue()函数，对日志队列进行处理。

theApp.HandleDebugLogQueue();

theApp.HandleLogQueue();

void CemuleApp::HandleDebugLogQueue()

{

m\_queueLock.Lock();

while (!m\_QueueDebugLog.IsEmpty())

{

const SLogItem\* newItem = m\_QueueDebugLog.RemoveHead();

if (thePrefs.GetVerbose())

Log(newItem->uFlags, \_T("%s"), newItem->line);

delete newItem;

}

m\_queueLock.Unlock();

}