# P2P Peer端简述

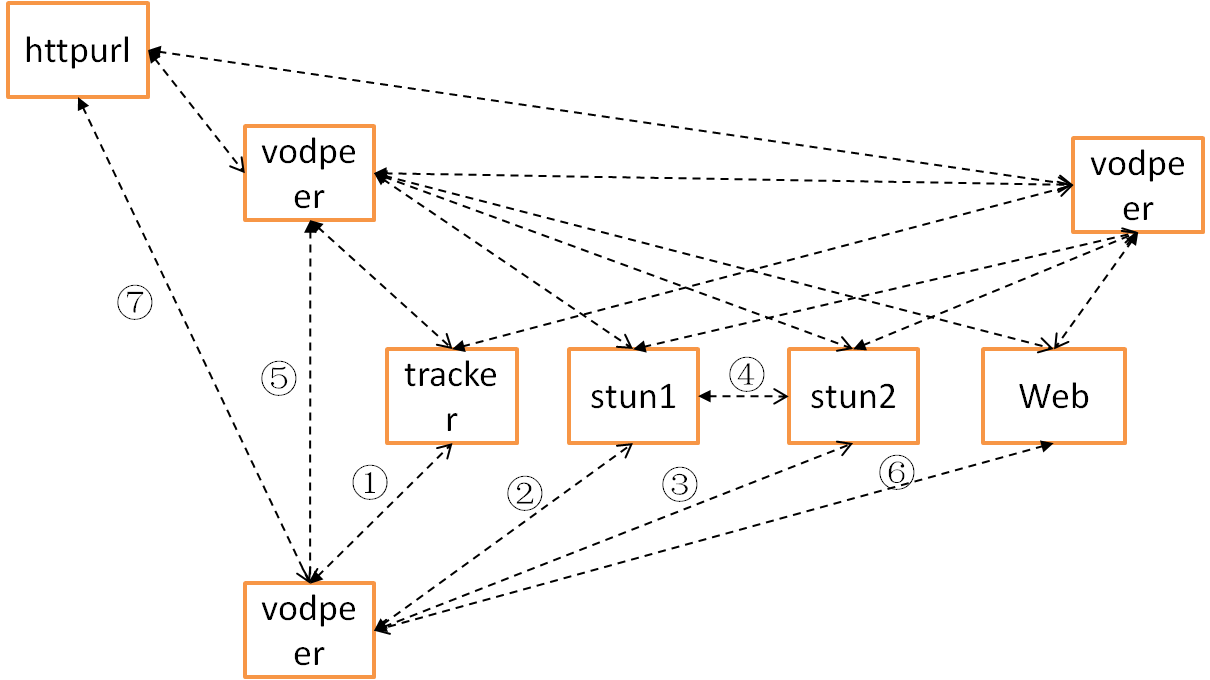


图3 P2P框架图

从P2P框架图看peer端功能主要体现如下：

① peer登陆tracker，实时向tracker上报自己的信息，向tracker请求文件源，tracker根据上报的nat类型分配源；

② peer登陆stun1服务器，探测peer端nat类型；

⑤ peer之间数据分享，通过tracker返回的文件源，通过打洞完成；

⑥ vodpeer从webserver读取配置文件；

⑦ vodpeer向http视频源请求视频，请求的视频片段放在本地，播放器取的视频数据就是已经下载存在本地的；

简单流程：

1.stun client连接stun server，向其nat探测包，进行一系列的通信；

2. Httpsvr收到手机端传入的url，传给downloadlist，获取m3u8列表，每个列表文件创建一个下载download，一个http源（CDN）和若干peer源；

3.tracker自动登陆tracker server，实时上报自己的信息，包括正在插入的视频源；

4.download根据正在播放的视频，向tracker请求视频源，收到视频源后并根据nat类型时行打洞；

5.peer间打洞成功后，互相请求文件块表，文件块，收到对方发的文件表，就知道对方有哪些文件块，去除自己有的块，向对方请求些对方有的（不要太多，保障能顺利播放即可）。

根据框架把P2P分成以下几部分介绍：主流程介绍、tracker通信、数据分享、打洞穿越、web获取配置文件四个部分介绍，另附其它，各部分涉及到的类会的重叠或者只是类名重叠，具体的类关系及序列图见“VodPeer结构图.pdf”，协议部分见“P2P协议.docx”

# 0.主流程介绍

以下以程序中的实现做大概的介绍



1）setting部分，完成读取配置文件（网络或本地）解析；

2）Timer部分，突成定时的任务；

3）IOReactor、SelectorReactor部分，完成TCP server的任务监听，类似select，读、写操作；

4）HashManger；

5）MemBlockPool部分，完成全局内存分配；

6）FileBlockPool部分，管理文件block；

8）FileStorage部分，完成实际文件在磁盘中的读、写操作；

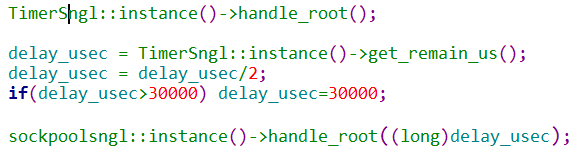
9）FileAutoCache部分，完成本地磁盘中文件的管理，主要是负责定期清理缓冲空间；

10）ShareService部分，完成p2p分享操作；

11）PeerManger部分，所有模块的统筹部分，最核心的是接收底层来自网络的数据，分配给DownloadManger和ShareService模块处理；

12）DownloadManger部分，负责管理下载任务；

13）init\_acceptor部分，主要是完成peer间的通信，包括打洞、连接、数据的发送与接收、nat类型的判断（与stunA和stunB的通信）。并且会在uac\_init中启动一个内部的定时器任务、线程工作和sockpool，线程的工作如下：



14）Tracker部分，与Tracker服务器通信；

15）Nattype部分，完成peer与tccsvr的TCP通信连接确定；

16）DownloadListManger部分，管理下载列表DownloadList；

17）LocalM3u8Mgr部分，本地的m3u8文件管理，在没有请求时会自动向tracker上报，这部分功能暂时未用到，已经被其它部分替代；

18）LocalService部分，功能暂时未用；

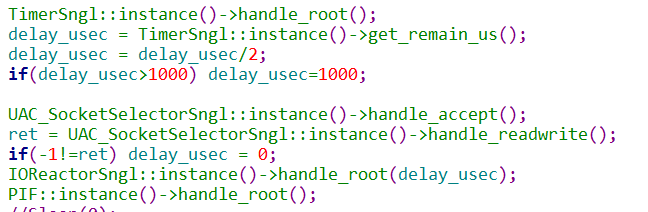
19）Scheduler部分，线程模块，这里主要是主线程的工作：  
 a）这里是完成注册的定时任务；

b）处理被动的peer连接；

c）处理peer的读写数据；

d）处理TCP连接操作；

e）处理来自httpsvr接口的请求；



20）DownloadManger部分，处理处理手动下载任务的操作；

# 1.Tracker通信

主要由tracker类完成，tracker是与Tracker server通信的模块，主要完成向Tracker server上报共享文件修文件、删除共享文件、开始下载文件、停止下载文件、开始下载文件列表、错误下载、下载文件速度、下载文件信息、请求源数据信息以及请求反连接、请求文件源、自动登陆Tracker server等，这样Tracker server就能实时掌握各个peer的各种分享信息，方便及时调度。

### 1.1主要成员变量

Peer\* m\_peer //连接tracker server的peer；

TrackAddr\_t m\_trackAddr //tracker server的地址；

ShareFileList m\_shareFiles //共享文件列表；

PLTNetInfo\_t g\_netLiveInfo //全变量，记录本peer的相关网络信息；

### 1.2主要成员函数

on\_timer：

1）登陆Tracker server成功后，6分钟内没有收到数据重连，每55s保活申请一次，5分钟上报一次状态；若没有成功就自动登陆；

2）上报共享文件；

init：初始始化操作，完成回调注册，这样tracker的peer的相关操作会回调tracker，并调用AutoLogin完成自动登陆。tracker也是peer，TCP连接的；

on\_update\_nattype：当NattypeMgr检测到NAT类型改变后会调用此接口，向tracker server上报nat类型及所有共享文件

PTL\_ReportShareFile：向tracker server上报共享文件；

PTL\_ReportRemoveFile：向tracker server报告删除文件；

PTL\_ReportStartDownloadFile：向tracker server上报开始下载文件；

PTL\_ReportStartDownloadList：向tracker server上报开始下载列表；

PTL\_ReportStopDownloadFile：向tracker server上报停止下载文件；

PTL\_ReportDownloadWrong：向tracker server上报下载错误；

PTL\_ReportDownloadFileSpeed：向tracker server上报下载文件速度；

PTL\_ReportDownloadFileInfo：向tracker server下载文件的信息；

PTL\_RequestFileSource：向tracker server请求文件源；

PTL\_RequestConnTurn：向tracker server请求反连接；

AutoLogin：自动登陆tracker server，调用peer的连接；

OnReadData：收到tracker server返回的数据，处理，见Peer to Tracker的协议；

ON\_PTL\_ResponseLogin：收到向tracker server请求登陆的响应，得到自己的共网ip，上网自己的nat类型，上报所有的共享文件，回调xxxx；

ON\_PTL\_ResponseFileSource：收到向tracker server请求文件源的响应，反把响应传给DownloadManager；

ON\_PTL\_ResponseConnTurn：收到向tracker server请求反连接的响应，若目的地址ip相等则直接使用内网连接，否则创建TCP的peer连接；

ON\_PTL\_RequestStartDownloadList：收到向tracker server请求开始下载列表的响应，会调用DownloadListManager的open\_downloadlist；

### 1.3 重要数据结构

调用及反馈机制



TCPChannel的Connected，Connecting，Disconnected，Readable，Writable回调peer的on；

Peer的Disconnected，Connecting，Connected，Writable，Data回调Tracker的on；

Tracker的Connecting，Connected，Disconnected回调downloadmanger的on；

downloadManger只处理了Connected的回调；

# 2.数据分享

## 2.1文件存储模块

文件存储主要由FileStorage完成，FileStorage文件存储以hash字符串作为文件的索引，存储是FileInfo信息，即每一个文件对应一个hash名和FileInfo，文件存储相关所有的对外接口都在FileStorage中定义：

1）FileInfo管理文件的读写等；

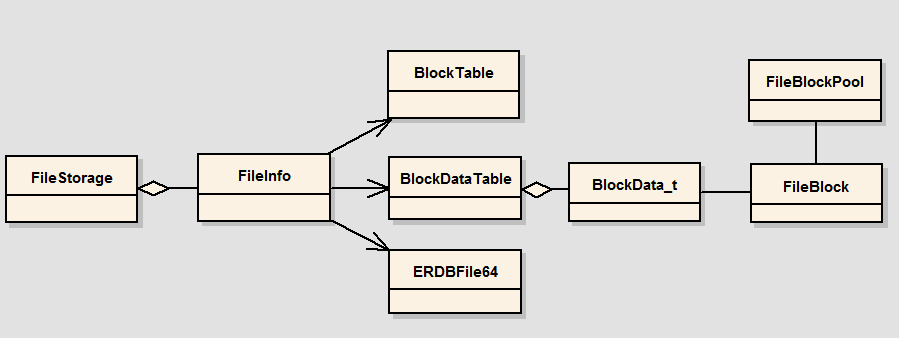
2）ERDBFile64是底层的文件open，close，read，write，seek，flush等；

3）BlockDatable存储的当前已经有文件块的记录；

4）BlockDataTable记录管理文件块的数据；

5）FileBlock实际文件块的数据管理；

6）FileBlockPool管理FileBlock。



文件存储结构图

### 2.1.1主要成员变量

typedef map<hash\_t,FileInfo\*> FileMap

FileMap m\_downinfo //正在下载文件的map集

FileMap m\_readyinfo //已经下载完的文件的map集

### 2.1.2主要成员函数

create\_downinfo：创建下载文件，文件名以“.vkkd!”结尾；

create\_readyinfo：创建完成下载文件，删除正在下载列表m\_downinfo表的项；

get\_downinfo：获取正在下载hash的FileInfo；

get\_fileinfo：返回获取的正在下载hash的FileInfo，否则从已下载列表中查找并返回；

get\_fileblock：获取指定位置的FileBlock；

get\_sharefile：获取正下载或已下载的文件；

write\_filedata：向指定位置写数据，前提是没有写过，不是覆盖式写；

read\_filedata：从指定位置读数据，磁盘没有话从cache中读取，优先从磁盘读；

on\_blockdone：文件块写完，使用BlockDataTable校验块数据的完整性，添加到memcache中，写入硬盘，找出下载缺口；

on\_filedone：本文件写完，将已下载的文件（以“.vkkd!”结尾）重命名，并存到m\_readyinfo中；

### 2.1.3重要数据结构

1）FileInfo

管理文件读写的，采用几个变量共同维护文件数据块的读写，bt\_finished是记录已经下载的数据块，bdt管理记录的已下载的数据块，固通过FileInfo可能知道某个block是否已经下载并能取到数据块。变量成员如下：

string url; //记录一个主URL，这个URL不保存到本地，只是开始下载时产生

int ftype;// FTYPE\_VOD=0,FTYPE\_DOWNLOAD,FTYPE\_SHAREONLY

int ref; //引用计数

int ctime; //创建时间

int mtime; //最后使用时间

int flag; //暂时未使用，保留，例如以后用于恢复原格式文件等标志使用

unsigned int block\_size; //块大小，默认是DEFAULT\_BLOCK\_SIZE 102400

unsigned int block\_offset; //指定从第几块开始下载

unsigned int blocks; //一共需要下载块数

unsigned int block\_gap; //下载缺口,指从block\_offset块起，第个出现未下载的块号

unsigned int down\_blocks; //已经下载完的块数，当其与需要下载块数相等时，下载完成，下载完成，触发HashManager 的check\_filehash，触发downloadManger的on\_file\_done，最终完成downloadlist中的file\_node的状态变化；

BlockTable bt\_finished; //已经下载完成的块的记录

BlockTable bt\_memfinished; //写入cache中的块的记录

BlockDataTable bdt; //文件块的数据存储

typedef map<unsigned int,FileBlock\*> FileBlockMap;

FileBlockMap fbmp\_mem; //存入cache中的数据



文件内容

2）BlockTable

管理文件块，实现[]操作符，记录已经存储的block，变量含义如下：

unsigned char \*m\_buf; //每一bit代表一个块

unsigned int m\_buflen; //数组大小

unsigned int m\_blocks; //blocks，即表中块的个数

unsigned int m\_block\_offset; //表示本table是的起始位置相对文件的起始块号，即table不一定表示完整文件情况.

unsigned int m\_block\_offset\_remainder； //m\_block\_offset/8的余数,主要是补充m\_block\_offset 8位对齐. 保证m\_block\_offset是8的整倍数方便拷贝.

接口如下：

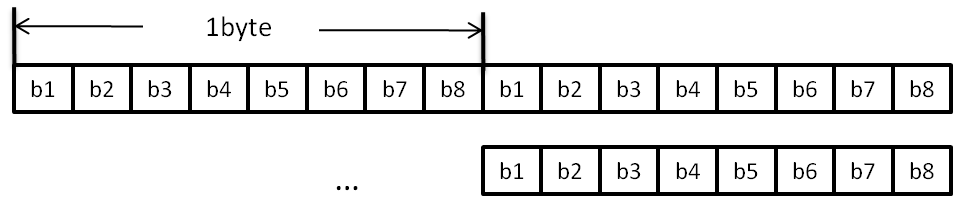
get\_block：判断第i个块是否存在，

使用m\_buf[((i-m\_block\_offset)/8) ]& (1<<((i-m\_block\_offset)%8))来判定；

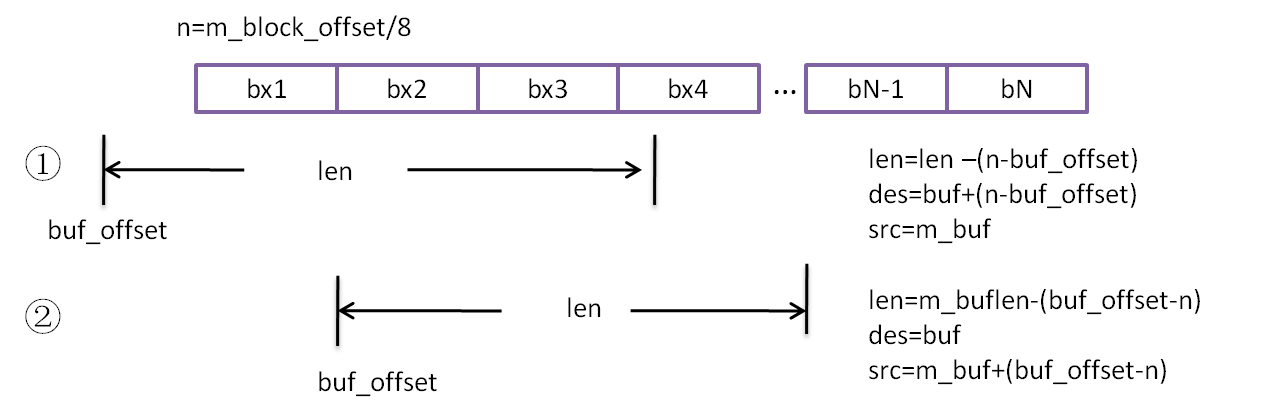
set\_block：把第i个块存在数据中也可清空，写上代表该块已经存在table中了，

m\_buf[(i-m\_block\_offset)/8]|(1<<((i-m\_block\_offset)/8))或者

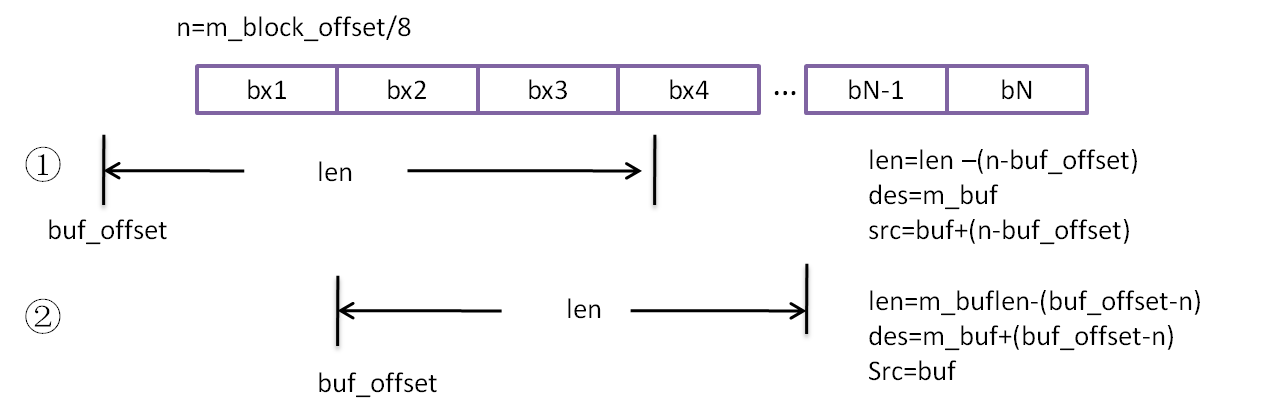
m\_buf[(i-m\_block\_offset)/8]&(1<<((i-m\_block\_offset)/8))；



get\_block\_buf (char\* buf,unsigned int len,unsigned int buf\_offset)：获取block表的情况，buf\_offset是按字节的绝对位置；



set\_block\_buf (const char\* buf,unsigned int len,unsigned int buf\_offset,unsigned int& real\_off,unsigned int& real\_len)：将block表的情况写入；



resize(unsigned int blocks,unsigned int block\_offset,bool copyold)：重新给blocktable定义大小；

3）BlockDataTable

管理数据块，记录已经存储的数据块，成员如下：

BlockData\_t \*\*m\_data;//二维数组，列为为BDT\_SUBNUM，即5000

BlockData\_t \_tmp;

unsigned int m\_blocks; //当前数据块表共有数据块；

unsigned int m\_block\_offset; //当前数据块表第一个数据的起始偏移值；

BlockData\_t的成员如下：

unsigned int size; //已经保存到硬盘的大小

void\* data; //实际是FileBlock的数据

4）FileBlock

文件块的载体，负责文件块的写入，文件块的get与put由FileBlockPool管理：

char\* buf;

unsigned int buflen; //

int ref; //block的引用计数

unsigned int begin; //block的起始位置

unsigned int **end**; //写入block的结束位置

hash\_t key; //文件名与block的索引的hash

5）write\_downinfo\_i的文件格式

flag:.DIF; version 1: hash=HASHLEN ; tth=64; path=256 ; despath=256 ; ftype=4 ,size=8 ;

block\_size=4 ;blocks=4;block\_offset=4;ctime=4;mtime=4; bt\_finished; bdt

bt\_finished (BlockTable)：

flag = m\_buflen + m\_blocks + m\_block\_offset + m\_block\_offset\_remainder + 69;

flag,4

m\_buflen,4

m\_blocks,4

m\_block\_offset,4

m\_block\_offset\_remainder

m\_buf,m\_buflen

bdt(BlockDataTable)：

flag = m\_blocks + m\_block\_offset + size + 69;

flag,4

Fm\_blocks,4

m\_block\_offset,4

size,4

arr,(int)(size\*sizeof(IndexSize\_t))

## 2.2下载管理模块

下载模块主要由downloadmanger、download、downloadlistmanger和downloadlist组成：

downloadmanger：管理下载download；

download：下载的主要处理，涉及到文件存储、下载调度等；

downloadlistmanger：管理下载列表；

downloadlist：下载列表的处理，处理m3u8文件片列表；

### 2.2.1 DownloadManger模块

#### 2.2.1.1主要成员变量

typedef map<hash\_t,Download\*> DownloadMap;

DownloadMap m\_downloads

#### 2.2.1.2主要成员函数

create\_download：创建下载;

share\_file：共享文件，文件下载完，从下载列表m\_downloads中删除，创建share信息，并向tracker上报；

on\_ptl\_packet：收到包的处理函数，在PeerManger中调用；

on\_peer\_ready：peer之间握手成功后调用；

on\_file\_done：文件下载完成，调用DownloadListManager的file\_done，删除当前的下载，调用FileStorage的on\_filedone；

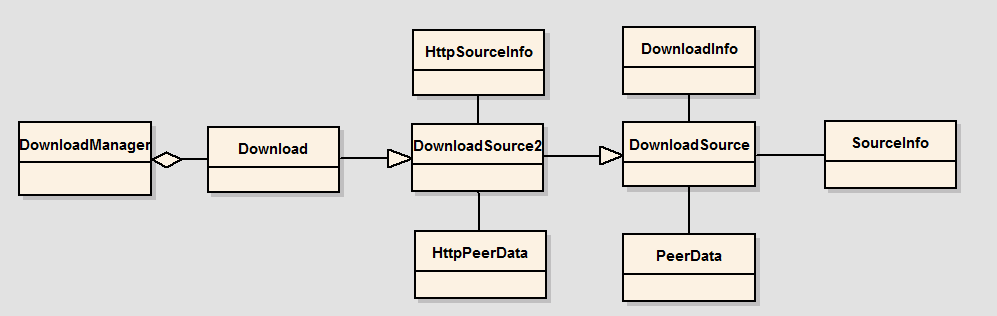
TimerDownloadHeartbeat：心跳操作，每秒会执行的，调用download的on\_second；

create\_turn\_channel：创建反连接通道，tracker申请RequestConnTurn，并保留turn信息；

add\_source\_list：向download下载添加p2p源；

http\_add\_source：向download下载添加http源，添加成功，让download创建http通道；

### 2.2.2 Download



download结构图

类Download继承于DownloadSource2，DownloadSource2继承于DownloadSource。其中DownloadSource2用于处理http的数据结构，DownloadSource用于p2p源的数据结构；

#### 2.2.2.1主要成员函数

create\_new\_download：创建下载任务，先调用FileStorage创建create\_downinfo，取得FileInfo，赋值到m\_downInfo中，调用http\_add\_source创建http source，最后调用start；

create\_old\_download：；

start：若m\_fi->is\_finished即完成当前文件下载则调用on\_file\_done，向tracker上报StartDownloadFile，创建create\_channel，http\_create\_all\_channel，搜索search\_source；

create\_channel\_i（DownloadSource）：根据连接类型的不同TCP\_CONN，TCP\_TURN，UDP\_CONN，LOCAL\_TCP，LOCAL\_UDP创建不同peer并连接connect；

http\_create\_channel\_i：创建HttpPeer并连接connect；

assign\_job：查找对方peer有的，自己没有需要下载的块，并向对方peer请求块，其中考虑到紧急、平缓等；

cancel\_job：取消任务，取消指定peer的指定index的块，若允许且该peer的block列表为空可以再给该peer分配任务；

on\_second：秒处理函数，处理如下：

1）处理http的pending；

2）http与peer源的任务再分配；

3）自动改动demand点，出现跳跃点，寻找第一个不在范围或者在范围但速度最小那个，取消从该peer请求block，调用create\_channel创建新的连接，若连续8块没有完成则同时创建http连接;

4）统计每个peer的下载速度，包括p2p source和http源；

5）统计自己的速度；

6）若自己速度为0且http源为空、p2p源为空，则创建p2p通道，创建http通道（需要添加http源的前提是http源少于1或者速度小于60k）；

7）根据不同的速度设置不同的source搜索周期；

ON\_PTL\_ResponseFileSource：收到tracker的返回的源，添加源，创建channel通道；

ON\_PTL\_ResponseFileBlockTable：接收到对方peer发送的对方已下载的数据块的表，记录信息，并分给该源peer分配任务；

ON\_PTL\_ResponseFileBlocks：收到对方peer发送来看请求块的情况，若缺失则取消该块；

ON\_PTL\_ResponseFileBlocksData：收到对方peer发送来的块数据，将数据存储并看是否需要调度：若我的下载速度比该peer的下载速度大6k以上或者滞后3次以上则不再从该peer下载该块，取消任务；

on\_peer\_ready：对方peer准备好了，可以向对方请求对方的blocktable；

auto\_change\_demand\_point：出现跳跃点时的处理，只要有peer已经下载当前的需求块就直接把当前peer的任务取消掉；

#### 2.2.2.2主要成员变量

成员变量多集中在其子类，直接分析子类的实现。

#### 2.2.2.3重要数据结构

DownloadSource2，是Download直接继承的基类，自己又继承于类DownloadSource，DownloadSource2主要是处理http的下载，DownloadSource主要是处理p2p源的处理以及一些共同的处理；

##### 2.2.2.3.1 DownloadSource2

**主要成员函数**

HttpSourceInfo\* http\_add\_source：添加http源，一个源就是一个httpsourceInfo，源的sessionID也是自己在sourceMap中的索引的key值，拆分server传来的ip、port及cgi值，如果是允许创建，则会调用http\_create\_channel\_i创建http通道；

http\_create\_all\_channel：创建所有的http通道，通道数为http连接最大值-已经有的http通道数；

http\_del\_source：删除http源，擦除掉httpSourceMap中相应的资源并删除资源；

http\_get\_source：根据id索引从httpSourceMap中查找http源，id即sourceInfo中的sessionID；

http\_create\_channel：；创建http通道，寻找连接失败次数少且引用最少的一个httpsource资源建立http通道；

http\_create\_channel\_i：根据传入的资源httpsource创建http通道，创建httppeer，在里面也让资源与httppeer挂沟，解析server的ip，连接server；

http\_deal\_deadpeer：处理不活动的http通道，5s不活动就断开，若最近2s的速度小于setting中最小的速度则也直接断开连接；

http\_deal\_pending：遍历挂起的peer列表，一个个删除；

http\_attach\_peer：将httpsource添加到httppeer上，主要是将httpsource中的一些信息传给httpdata，并把httpsource引用数加1；

http\_detach\_peer：取消该peer的任务；

on(Connected,Peer\* peer)：连接http源成功后的调用，标示该源已经准备好了，开始分http任务；

on(Disconnected,Peer\* peer)：detach该peer，并把peer推入挂起的peer列表中；

http\_assign\_job：给httppeer分配下载任务，若文件大小为0或者httpdata的用户时间为0则直接请求http数据并返回，判断是否需要删除连接。如果本连接上次已经分配下载过任务，看看是否还可继续使用本连接；

http\_check\_reconnect：；

http\_cancel\_job\_p：取消任务，可以取消某一块的数据也可取消该peer所有任务，blockList不为空说明还有任务；

http\_resume\_assign\_job：重新分配http下载任务；

http\_req\_data：通过cgi向http源请求数据，请求http数据时不是按一块请求一次，是把请求列表blockList中所有数据的范围作请求数据的范围；

http\_check\_rerequest：检查是否需再次请求http视频数据，若没有请求到数据则需要；

http\_check\_need\_add\_connection：检查是否需要添加http连接，只有httppeer数为0或者速度小于60k时需要，实际上会控制只能有1个http连接；

http\_check\_need\_del\_connection：检查是否需要删除http连接，只有连接数超过最大http连接数才需要删除；

get\_block\_speed：获取下载某一块的速度，取的是所有下载过该块的用户的速度的最大值；

on(HttpHeader,Peer\* peer,char\* buf,int len)：；

on(Data,Peer\* peer,char\* buf,int len)：；

on(Connected)：连接成功后会开始分配http任务；

on\_attach\_http\_peer：添加httpsource，并调用http\_attach\_peer，最后调用on(Connected())；

**主要成员变量**

typedef map<unsigned int,HttpSourceInfo\*> HttpSourceMap;

HttpSourceMap m\_http\_sources; //httpsource的集合；

HttpPeerMap m\_http\_peers; //

list<HttpPeer\*> m\_http\_pending\_peerls; //

int m\_http\_max\_num; //

**重要数据结构**

**1）HttpPeerData**

int httpstate;//0:ready,1=requesting,2=dataing

bool isReady;

string server; //真实http服务器的地址

string cgi; //请求http视频的请求指令

list<BlockInfo> blockList; //语法块列表

Speedometer<unsigned int> speedometer; //速度统计

int support\_keepalive; //0:not support,1:support,2:unknown,the initial value

int support\_range; //0:not support,1:support,2:unknown,the initial value

sint64 http\_data\_pos; //http数据的位置

sint64 http\_data\_remain;

int lagTimes; //竞争下载时的落后次数

int resume\_state; //0:正常，收到数据后即置0，1：刚resume，2：1状态下分配到下载任务（断开时如果为2状态，即重新创建连接）

int userDataTimes;

**2）HttpSourceInfo**

string url;

string server;

unsigned int ip;

unsigned short port;

string cgi;

unsigned int sessionID; //分配一个来跟peer关联

int support\_keepalive; //0:not support,1:support,2:unknown,the initial value

int support\_range; //0:not support,1:support,2:unknown,the initial value

int connectTimes; //连接次数

int connectFailTimes;

int userDataTimes;

int ref;

unsigned int lastUseTick;

##### 2.2.2.3.2 DownloadSource

下载的基类

**主要成员变量**

DownloadState m\_state; //

//下载的状态，DOWNS\_INIT=0,DOWNS\_STOP,DOWNS\_QUEUE,DOWNS\_START

typedef map<int,list<Peer\*> > BlockRefMap

BlockRefMap m\_blockRefs; //文件块peer列表的map，记录的是某个文件块对应的peer列表，方便计算某个块的速度，索引是块的编号

Speedometer<uint64> m\_speedometer; //测速模块

FileInfo\* m\_fi; //当前下载文件的引用，peer下载和http下载引用的是同一个，都是从FileStorage中引用的；

typedef map<Peer\*,PeerData\*> PeerMap //peerData是记录该peer的下载数据

PeerMap m\_peers; //peer及对应数据的集合；

typedef map<uint32,SourceInfo\*> SourceMap

SourceMap m\_sources; //源列表

DownloadInfo m\_downInfo; //当前下载的信息

**主要成员函数**

on\_peer\_turn：；

add\_source\_list：添加p2p源列表，调用add\_source添加源列表中的源，最初调用的地方在DownloadList创建下载时创建的；

add\_source：添加p2psource源，PTL\_P2T\_PeerInfo格式的，把源存到sourceInfo中，并把sourceInfo存入m\_sources的map中，map的key是peer在tracker端唯一的sessionID，peer与sourceInfo也是通过sessionID联系的，如果源已经在peermap中并且自己是source，则源Source引用次数增加；

get\_p2pconnect\_amount：；

attach\_peer：给peer新建立peerData，给peerData初始化isMySource、ctype、sourceSessionID、bt\_finished，调用PeerManager的add\_dref将peer与downloadSource联系起来，；

detach\_peer：；

create\_channel：从p2psource源列表m\_sources取源，保障只有一个连接且连接失败超过两次就从源表中清除，即当前与peer与对方的peer只有一个连接，不会出现连接超过两次的情况；

create\_channel\_i：根据传入的源sourceInfo创建通道，不同的source类型创建不同的peer，有类型TCP\_CONN，TCP\_TURN，UDP\_CONN，LOCAL\_TCP，LOCAL\_UDP。实际TCP\_CONN应该是连接外网源，可能没有用到，p2p一般使用UDP，如果是局域网使用LOCAL\_TCP。连接成功后，调用attach\_peer，更新此源连接信息，如连接次数及源的引用计数，源的信息是很重要的，连接次数、引用次数。这个创建的peer会连接外网，其实是会触发打洞操作，具体操作参见3.4.3；

get\_block\_speed：统计下载该块的p2p源的下载速度，统计的是最大速度；

add\_block\_ref：把引用该块的peer添加到块引用列表中m\_blockRefs；

del\_block\_ref：与上相反操作；

on\_block\_done：一个文件块下载完成的操作，先调用FileStorage的on\_blockdone，下载20块就开始共享，向tracker上报共享文件信息。若是文件的最后一块，即下载块数与需要下载的块数相同，则触发on\_file\_done；

on\_file\_done：文件下载完的操作，更新下载信息记录m\_downInfo中的信息，主要是更新下载已下载的块数，会触发HashManager去检查check\_filehash；

update\_filesize：更新FileStorage中下载文件的size，同时更新downloadsoure中downloadInfo信息；

**重要数据结构**

1）SourceInfo

记录的是p2p源

PTL\_P2T\_PeerInfo source; //这个源也是tracker返回的p2p源

int ctype; //连接类型如:TCP\_CONN

int connectTimes; //连接次数

int connectFailTimes;

int noDataTimes; //无法获得数据次数

int ref; //源的引用次数

unsigned int lastUseTick;

PTL\_P2T\_PeerInfo的在成员如下，这些也是peer向tracker上报，tracker记录下的：

uint32 trackID; //指定属于哪个tracker,用于支持多tracker级连保留

uint32 sessionID; //tracker使用的sessionID,直接指tracker上面的索引

char utype; //USER 类型 UT\_CLIENT...

char ntype; //nat类型0~6,6为未知

uint32 menu; //菜单功能位:低0位共享支持,低1位TCP支持,低2位UDP支持

uint32 tcpLocalIP;

uint32 tcpRealIP;

uint16 tcpLocalPort;

uint16 tcpRealPort;

uint32 udpLocalIP;

uint32 udpRealIP;

uint16 udpLocalPort;

uint16 udpRealPort;

2）PeerData

下载的peer的数据集合：

PeerState state;

int type;

list<BlockInfo> blockList; //需要下载的块

list<BlockInfo> blockListNew; //分配任务时先分到这些,PTL时再合到blockList

BlockTable bt\_finished; //已经完成的块

TableInfo bt\_info;

int lagTimes; //竞争下载时的落后次数

Speedometer<unsigned int> speedometer;

uint32 sourceSessionID;

bool isReady; //判断是否连接可用

bool isMySource; //由我创建

int lagBlockI; //竞争落后之后下次不再下载此块 no longer download

BlockInfo的结构如下，每一个需要下载的块的信息都很详细：

unsigned int index;

unsigned int pos;

unsigned int blockSize;

TableInfo信息如下：

unsigned int index\_offset;

unsigned int index\_num;

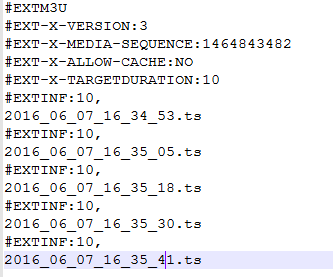
unsigned int last\_update\_tick;

unsigned int state; //0-未完整的，1-已经完整的表

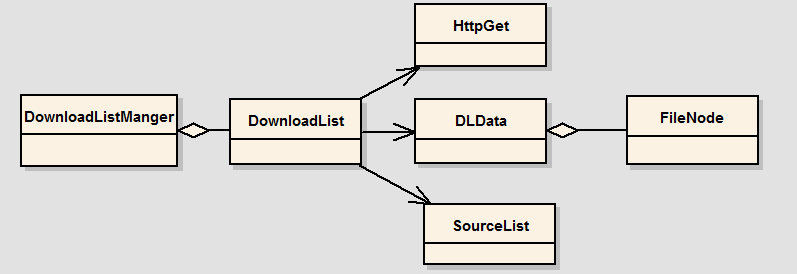
unsigned int assign\_fails; //分配任何失败次数

### 2.2.3 DownloadList

手机端传入的参数类似“http://124.193.180.78:8888/jiangsuws/jiangsuws.m3u8”P2P直播台的url地址，peer请求得到m3u8文件的列表，实时更新并下载分两个方向：一是直接向http源请求数据，另一个是向p2psource请求数据；



直播列表项



downloadlist结构图

1）DownloadListManger管理downloadList；

2）DLData管理下载的文件；

3）HttpGet内聚HttpPeer，向http源请求播放列表，收到数据后回调downloadlist的on函数；

4）SourceList记录本频道的p2p源；

5）FileNode记录视频片段文件。

HT\_URL2类型的download任务，拥有一个主下载的DL令牌概念。一个downloadlist同一时刻只有一个下载获得DL令牌

1拥有DL令牌的下载才允许创建新连接

2拥有DL令牌的任务停止或者完成时令牌回收，并获取它的所有源将源传递给下一个下载，先交接完令牌再赋源，赋源时注意查是否有对应的peer把引用递增

3搜索回来的源只传递给拥有DL令牌的下载，也就是说HT\_URL2没令牌的不会有源

4 HT\_URL2类型的download任务，detach\_peer()时，peer尝试传递下一个任务下载（源不传递）

5 HT\_URL2类型的download任务，任务快结束时注意抢下载策略避免过多的垃圾下载

6主下载的最后块下载过慢目前不考虑从其它下载任务回调peer过来抢下载，或者考虑新创建连接来解决

#### 2.2.3.1主要成员变量

DLData m\_data; //下载数据

HttpGet m\_httpget; //http数据请求

hash\_t m\_token\_hash; //当前主下载文件，即DL令牌

typedef list<SourceInfo\*> SourceList;

SourceList m\_p2p\_sources; //同一频道的其它peer源

#### 2.2.3.2主要成员函数

open：创建downloadlist，客户端传入vodpeer的动力源，传入的是url的hash、url、name等，启动；

start：启动downloadlist，各http源请求获取最新直播列表，并向tracker上报ShareFile及StartDownloadList；

update\_downloadlist：更新下载列表，通过HttpGet向视频源请求m3u8列表；

on\_second：秒函数，每1s检测处理函数，主要处理如下：

1）统计测速；

2）每20分钟上报下载信息；

3）检测到速度为0是中上本下载，开始下一下载，DownloadManager->read\_release\_i；

4）周期更新playlist表，10秒，输出playlist；

5）周期性搜索源；

set\_token：设置主下载文件token，token的时候一定尝试添加剂一次http源，重复会添加失败的。当多HTTP连接并发下载一个任务时，传递HTTP连接会导致某片段的HTTP连接数超量；

create\_download：调用DownloadManager的create\_download，path传的是m3u8列表中的ts名；若已下载完成则check\_create\_next\_download，否则同时添加http和p2p源，并设置token；

find\_next：查找下一个FileNode；

on(HttpGetListener::Response...)：请求m3u8列表成功后触发的函数，重组playlist放成列表，如请求http://124.193.180.78:8888/jiangsuws/jiangsuws.m3u8返回的其中一个是2016\_06\_07\_16\_34\_53.ts，则放入列表http://124.193.180.78:8888/jiangsuws/2016\_06\_07\_16\_34\_53.ts，把新创建的FileNode推入到

m\_data.fn\_list列表中，并更新origi\_downlist列表，修改成本地地址的url形式，最后调用check\_create\_download；

check\_create\_download：计算自己可以下载的数量，查找出需要下载的hash，创建下载任务create\_download；

create\_download：调用DownloadManager创建下载任务，根据返回结果做不同的处理，若处理完，检查创建下一任务；否则创建下载，并调用DownloadManager添加p2p源和http源的接口，把downloadlist更新得到源添加到相应的download中去，并设置token，这也是主下载令牌；

on\_tracker\_connected：连接tracker成功后调用的函数之一，会上报ShareFile，并搜索source\_search；向tracker请求FileSource；

on\_file\_stop：若不用下载则找一个下载传递token；

read\_release：调用DownloadManager的read\_release\_i，若fn状态为FN\_MEMCACHE\_FINISHED则置为FN\_IDLE；

#### 2.2.3.3重要数据结构

1）FileNode

记录下载片段文件的信息

string url; //http真实的片段地址，

//如“http://124.193.180.78:8888/jiangsuws/2016\_06\_07\_16\_34\_53.ts”，

string local\_url; //http://127.0.0.1:7080/playlist/file/hash/name

string local\_path; // add for export playlist，也是下载存放的路径

hash\_t hash; //直播源的hash码，即主url和从url名字的hash，是URL2格式的，UDLDL是下载列表的hash格式；

int i; //排号，即filenode在DLData中的排号；

int state; //FN\_IDLE=0,FN\_DOWNING,FN\_FINISHED,FN\_MEMCACHE\_FINISHED

bool is\_line\_no; //是否是最新更新列表里面的，返回给播放器只返回最新更新到的

char read\_ref;

typedef list<FileNode> FileList; //文件下载列表

2）DLData

管理文件下载

string plpath; //本地保存路径

string pre\_filename;

FileList fn\_list; //文件列表

list<string> origi\_downlist; //原列表

unsigned int file\_i; //排号

unsigned int fini\_num; //一共下载完成多少个

list<int> fini\_i\_ls; //记录已经完成下载的序号

hash\_t cur\_fn;

unsigned int last\_updatepl\_tick;

hash\_t m\_playlist\_first\_hash; //当前更新到最新列表的第1个hash

3）SourceInfo

文件源

PTL\_P2T\_PeerInfo source; //该文件源的信息

int ctype; //连接类型如:TCP\_CONN

int connectTimes; //连接次数

int connectFailTimes; //连接失败次数

int noDataTimes; //无法获得数据次数

int ref;

unsigned int lastUseTick;

4）PeerData

peer的数据记录

PeerState state; //peer的状态PS\_DISCONNECT,PS\_READY,PS\_REQUESTING\_FILE\_TABLE,

PS\_REQUESTING\_FILE\_BLOCK,PS\_DOWNLOADING,PS\_ASSIGN\_JOB\_PAUSE,PS\_ASSIGN\_FAIL\_WAITING,PS\_NOTHING\_TO\_DOWNLOAD，

int ctype;

list<BlockInfo> blockList;

list<BlockInfo> blockListNew; //分配任务时先分到这些,PTL时再合到blockList

BlockTable bt\_finished;

TableInfo bt\_info;

int lagTimes; //竞争下载时的落后次数

Speedometer<unsigned int> speedometer;

uint32 sourceSessionID;

bool isReady; //判断是否连接可用

bool isMySource; //由我创建

int lagBlockI; //竞争落后之后下次不再下载此块 no longer download

5）TableInfo

unsigned int index\_offset;

unsigned int index\_num;

unsigned int last\_update\_tick;

unsigned int state; //0-未完整的，1-已经完整的表

unsigned int assign\_fails; //分配任何失败次数

### 2.2.4 httpsvr模块

libpeer的httpsvrHandler是C代码，引用个全局变量HttpServer g\_httpsvr，HttpServer是封装http操作的类

#### 2.2.4.1主要成员函数

httpsvr\_start：根据传入的ip和port创建httpserver，并传入响应处理函数handle\_https\_req；

handle\_https\_req：主要http处理函数，处理很多，也是播放器请求数据的处理地方；

response\_playlist\_file：当（播放器）请求“/playlist/file/”时的处理函数；

response\_vod\_file：返回请求的视频文件；

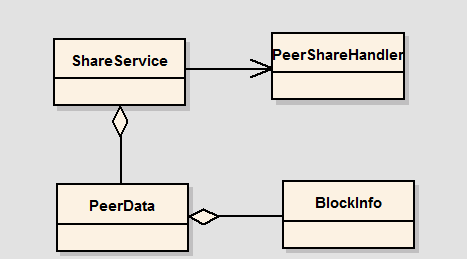
format\_http\_header：；格式化反馈给播放器的处理，有数据范围、内容类型等；

send\_response\_data：发送数据，调用PFile的read函数，若有head先发head再发内容；

response\_playlist\_play：插口请求playlist的处理；

## 2.3 共享模块

ShareService:主要负责响应对方peer请求的数据块表、数据块(PTL\_P2P\_REQUEST\_FILE\_BLOCK\_TABLE，PTL\_P2P\_REQUEST\_FILE\_BLOCKS，PTL\_P2P\_CANCEL\_FILE\_BLOCKS，PTL\_P2P\_REQUEST\_FILE\_SUB\_KEYS)等，要发送的数据放入队列，在定时处理函数中发送；



### 2.3.1主要成员变量

typedef map<Peer\*,PeerData\*> PeerMap;

typedef list<Peer\*> PeerList;

PeerMap m\_peers; //peer和peerData组成的map；

PeerList m\_sendingPeers; //需要发送数据的peer列表；

### 2.3.2主要成员函数

ON\_PTL\_RequestFileBlockTable：对方请求文件块表，在存储中查找，并把信息返回给对方；

ON\_PTL\_RequestFileBlocks：对方请求文件块，判定对方需要块自己是否有（已经存盘或者在内存中），把能分享的数据块发送对方，发送数据块；

SendBlockData：总是保证队列首block为当前回复block，向FileStorage调用get\_fileblock\_trymemcache获取FileInfo信息，Statistician统计发送的数据（按ip类型），DownloadListManager也按ip类型统计；

on\_timer：定时器中发送，m\_sendingPeers发送列表不为空就发送；

on\_writable：发送列表m\_sendingPeers不为空或者统计Statistician允许最大发送为0，则把该peer推入发送列表暂停发送，否则直接发送SendBlockData；

peer\_attach：添加peer到map中，并调用PeerManager的add\_share，调用Statistician的OnShareStart；

### 2.3.3重要数据结构

1）PeerData

记录peer分享数据的情况，分享的列表，当前正在分享的数据块

hash\_t hash;

list<BlockInfo> blockList; //当前peer的要分享的块列表；

unsigned int iCurrIndex; //当前正在提供的块

FileBlock\* fb; //当前正在提供的块

//速度统计

unsigned long iPeerBeginTick;

uint64 iPeerShareBytes;

uchar pause\_send;

2）BlockInfo

数据块的信息记录

hash\_t hash;

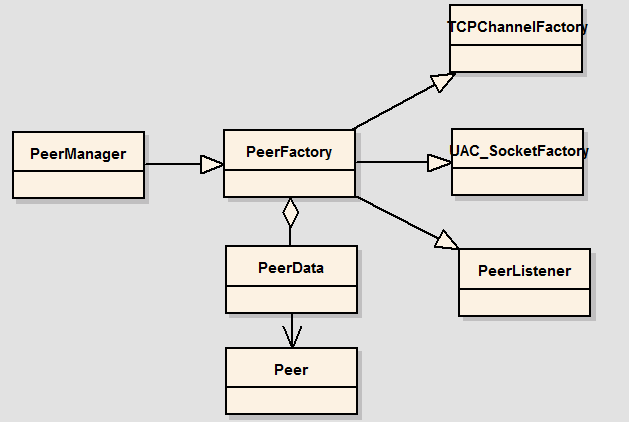
unsigned int index; //下载块号

unsigned int pos; //下载的起始位置

unsigned int blockSize; //本块大小

unsigned int fileBlockSize; //文件分块大小

## 2.4 Peer管理模块



PeerManger：管理peer，包括peer的连接、数据发送接收、数据的分享等，管理下载与分享；

PeerFactory：各种接口的载体，UDP、TCP的连接、断开、数据等；

PeerData：Peer的载体，记录peer的信息；

PeerListener：Peer处理后的回调接口；

UAC\_SockFactory：打洞（UDP）成功后会peer的信息传送到PeerManger的接口；

TCPChannelFactory：TCP连接成功后传送连接信息的接口；

### 2.4.1主要成员变量

PeerDownloadHandler \*m\_pdh; //实际对应的是DownloadManger

PeerShareHandler \*m\_psh; //实际对应的是ShareService

### 2.4.2主要成员函数

PTL\_welcome：向peer发送握手包；

ON\_PTL\_welcome：收到对方peer发送的握手包，设置peer的nat类型及用户类型；

add\_share：将peer的bshare置为true；

add\_dref：传递下载引用；

on(PeerListener::Connected,Peer\* peer)：连接成功后调用PTL\_welcome；

on(PeerListener::Data,Peer\* peer,char\* buf,int len)：接收到peer的数据包，都是Peer to peer的数据；

on(PeerListener::Writable,Peer\* peer)：数据可写，交给ShareService的on\_writable处理；

# 3.打洞穿越

先介绍下主要流程：

1.PeerManger收到tracker返回的同源的可连接的peer，这里发起打洞需求；

2.stun server返回打洞目的地址，请求方向对方发打洞包；

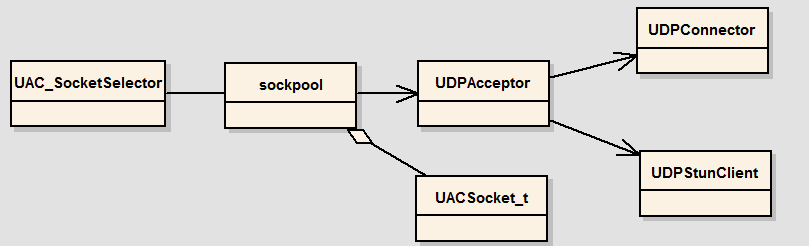
3.打洞成功后再通知PeerManger;

Note：

1.所有UDP数据的发送最终会能过UDPChannel发送出去；

2.UDP的数据的接收通过UDPAcceptor的handle\_input接收，除去nat探测和打洞的包，其它的数据最终会通过UDPConnector的handle\_cmd\_channel\_packet交给UDPChannel，回调sockpool的数据，最终会传向PeerManger

主要涉及到UAC\_SocketSelector、sockpool、UDPAcceptor、UDPConnector、UDPStunClient、Peer等，打洞原理及协议参见“p2p业务说明\_补充.docx”



UAC\_SocketSelector：处理打洞成功后的输入输出；

sockpool：UDP操作的集合；

UDPAcceptor：UDP操作的聚合；

UDPConnector：打洞；

UDPStunClient：探测NAT类型

UACSocket\_t：打洞成功后的数据操作，内有UDPChannel；

## 3.1 UDPAccepter

主要是连接stun server，接收stun server的返回的消息，并交与UDPStunClient或UDPConnector处理

### 3.1.1主要成员变量

UDPConnector \*m\_ctr; //

UDPStunClient \*m\_stun; //

### 3.1.2主要成员函数

open：创建连接stun的udp，创建UDPStunClient和UDPConnector实体，将UDPAcceptor注册到UDPStunClient中，这样UDPStunClient的fire回调会调用UDPAcceptor的on；

handle\_input：直接使用recvfrom接收连接stun的udp的数据，根据数据头格式不同交与不同的模块处理，高四位是UDPS\_CONN\_HEAD\_STX数据包让UDPConnector处理，数据包头是PTL\_STUN\_HEAD\_STX交与UDPStunClient处理；

handle\_select\_read(SockAcceptor)：select可读的，调用handle\_input；

## 3.2 UDPStunClient

探测nat类型，探测类型的请求是在定时函数中根据不同的状态向stun server请求不同数据

### 3.2.1主要成员变量

StunsvrConfig\_t m\_stunconf; //stun server的ip配置信息

unsigned int m\_binding\_ip; //本peer的绑定的ip及port

unsigned short m\_binding\_port; //

### 3.2.2主要成员函数

on\_data：收到stun server返回的数据，处理探测nat的响应，PTL\_STUN\_RSP\_BINDING\_ADDR，PTL\_STUN\_RSP\_STUNB\_ADDR，PTL\_STUN\_RSP\_NAT1，PTL\_STUN\_RSP\_NAT4，PTL\_STUN\_RSP\_NAT2，PTL\_STUN\_REQ\_HOLE，PTL\_STUN\_RSP\_HOLE；

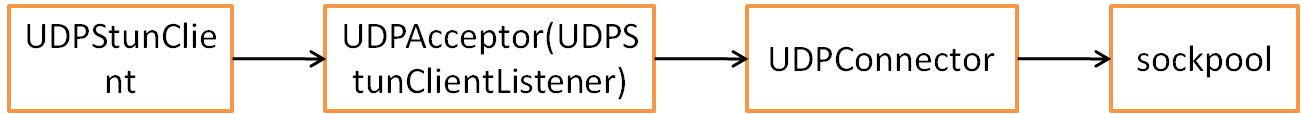
ptl\_request\_hole：向stun server发送请求打洞的消息；

on\_timer：定时获取本peer公网ip的请求(20s)，根据状态的不同向stun server请求不同的nat探测(0.6s)，发起nat探测(120s)；

### 3.2.3 重要数据结构

调用及反馈机制：

peer &stun



UDPStunClient的NatOk，BindingAddrChanged，ReqHole会回调UDPAcceptor的on；

UDPAcceptor会调用UDPConnector的相应处理，UDPConnector连接成功后会attach到sockpool

## 3.3 UDPConnector

### 3.3.1主要成员变量

UDPChannelFactory\* m\_chf;

UDPStunClient \*m\_stun;

UDPSession\_t \*\*m\_udps; //udp 会话记录

typedef list<NatInfo\_t> NatList;

NatList m\_natl; //需要打洞的nat列表

int m\_socknum; //在线连接数

### 3.3.2主要成员函数

on\_rsp\_nat\_hole：对端回应表示对方一定收到了NAT的请求，所以不再需要重发nat\_req到对方去；

register\_channel：声明一个会话在m\_udps，在connect和attach时会调用到；

connect：注册通道，信息保存到m\_udps，状态置成UDP\_CONNECTING，调用UDPStunClient发送打洞请求，并向对方peer发送连接请求；

nat：若已存在不再打洞，把nat请求推入m\_natl列表中，向对方发送打洞命令；

remove\_nat：取消发打洞包,对方可能是nat4,如果是nat4只要ip相同就册，否则ip,port相同才删；

on\_data：收到stun server发送的数据的响应；

handle\_cmd\_nat：我主动连接对方，对方发的打洞包；我是nat4,对方是nat2时，对方端口并不会变；我是nat2,对方是nat4时，对方端口会改变，此时要更新端口。发送连接；

handle\_cmd\_conn：我发打洞包，收对方连接请求,不用管对方是什么类型；

handle\_cmd\_ok：若当前是状态是UDP\_CONNECTING，则调用on\_connecting\_ok，连接和被连接都可能收到OK的消息，这里根据状态判定；

on\_connecting\_ok：继续发送OK，调用UDPChannel的handle\_connected，最终会回调sockpool；

handle\_cmd\_live：对方发来的判定是否打通物处理，若当前状态是被连接则置为UDP\_ACCEPTED，否m\_udps中不存在相对应的则发竤连接close命令；

handle\_cmd\_channel\_packet：收到对方发送的数据，调用UDPChannel的handle\_recv，最终会回调sockpool的函数；

send\_conn：向目标peer发送连接；

send\_conn\_ok：向对方peer发送连接OK包；

timer\_resend：定时完成的任务：

1）创建的"主连接/被连接"(状态为UDP\_ACCEPTING或UDP\_CONNECTING)15秒后不成功断开；

2）本连接超时没有接收数据就断开；

3）规定时间没有发送或接收数据发送保活命令；

4）"被连接"每0.8秒未收到过测速包则重发一次"ok"；

5）主连接每0.8秒发一次"connect"，请求打洞并发送连接请求；

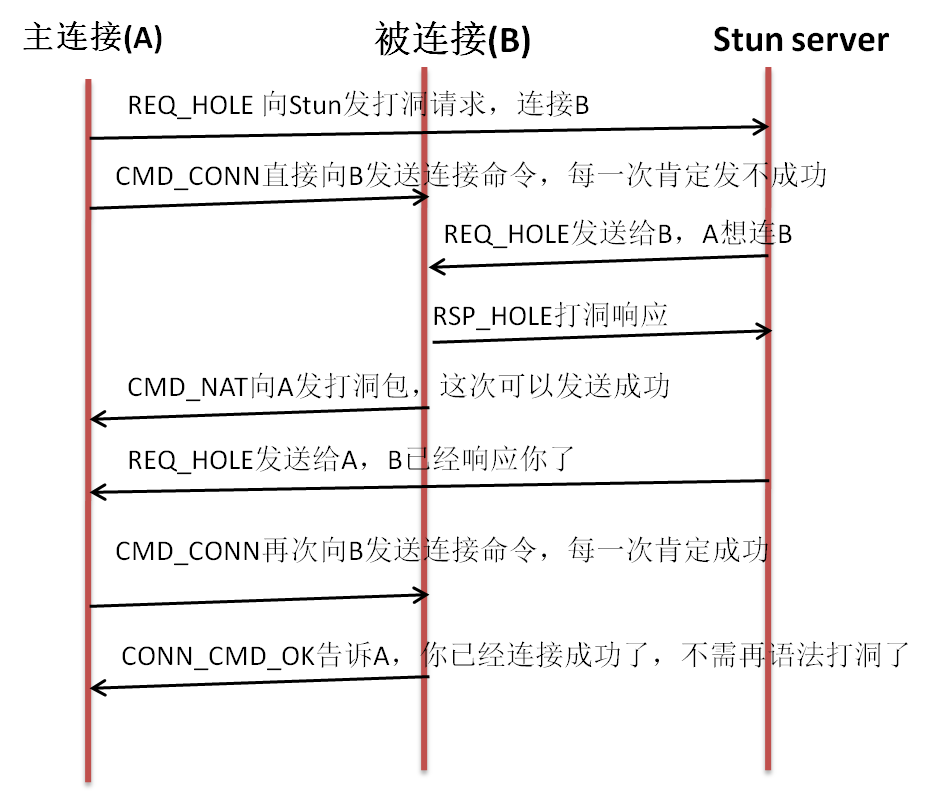
6）测速超过3秒未完成，则当完成；

timer\_channel\_send：针对已连接或被连接成功的会话，发送数据handle\_send

### 3.3.3 重要数据结构

1）打洞流程

先向对方地方发送连接信息，同时请求打洞，这样自己肯定能收到对方发的确认信息包



## 3.4 sockpool

### 3.4.1主要成员变量

UACSocket\_t \*m\_socks;

UDPAcceptor \*m\_acceptor; //

list<UAC\_SOCKET> m\_ls\_accepting; //底层通知上层的消息队列

list<UAC\_SOCKET> m\_ls\_connecting; //上层通知底层的消息队列

list<UAC\_SOCKET> m\_ls\_disconnecting;

list<UAC\_SOCKET> m\_ls\_sending;

### 3.4.2主要成员函数

init：调用UDPAcceptor连接stun server，创建UACSocket\_t数组m\_socks，数组成员变量UDPChannel是新建的UDPChannel，并把sockpool注册到UDPChannel中，这样UDPChannel中能回调sockpool的on；

accept：判定被连接列表，把SOCK\_ACCEPTING置为SOCK\_CONNECTED，并把地址传出；

connect：主动连接，把地址推入m\_ls\_connecting列表，状态置为SOCK\_WAIT\_CONNECTING；

is\_write：判定是否有需要写的，；

is\_read：判定给定fd是否可读，判定其接收列表recvlist是否为空；

select：判定是否可读可写，发送列表有数据或接收列表recvlist不空，告诉上层UAC\_SocketSelector哪个socket可读可写；

send：分包，把发送数据推入sendlist，通知底层发送；

recv：从recvlist获取接收到的数据，是底层接收到存入的，更新接收win的大小；

find\_idel\_socket：查找60s没用的、状为SOCK\_IDLE且连接状态为断开的socket；

handle\_root：处理UDPAcceptor的handle\_select\_read，处理发送列表，连接列表直接连接，处理断开连接；

attach\_udp\_channel：把已经打洞连接成功的地址保存到m\_socks中；

on(ChannelListener::Data,Channel\* ch,memblock \*b)：收到UDPChannel的数据，推入到recvlist列表；

on(ChannelListener::Writable,Channel\* ch)：调用UDPChannel发送数据；

### 3.4.3重要数据结构

1）UACSocket\_t

UDP发送接收数据的集合

int sock\_state; //空闲，正在连接，正在接受，连接完成

char close\_state; //应用层和底层关闭

UAC\_sockaddr addr; //对应peer的ip地址

UDPChannel\* ch; //处理udp的相关操作

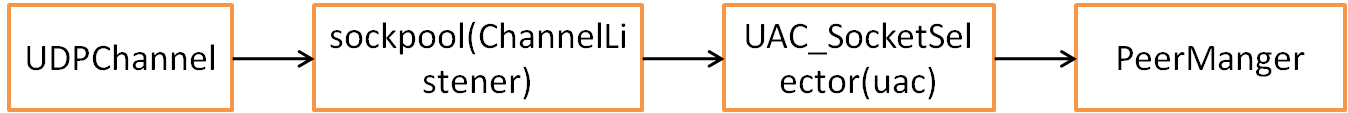
unsigned int max\_sendbufsize;

unsigned int max\_recvbufsize;

list<memblock\*> sendlist; //发送列表

list<memblock\*> recvlist; //接收列表

2）调用及反馈机制：



UDPChannel的Connected，Disconnected，Data，Writable，Connecting会回调sockpool的on；

sockpool会反接收到的信息记录，在UAC\_SocketSelector会处理；

3）对应关系

每个peer对应一个UACChannel，UACChannel的回调也是调用的Peer；

UACChannel的你类UAC\_Socket封装了sockpool的操作，即所有的UACChannel对应一个sockpool，sockpool中的定义了m\_socks的UDPChannel数组，所有的UDPChannel对应一个Sockpool，UDPChannel的回调也是调用的是sockpool的函数；

在调用sockpool的connect或accept时，会从m\_socks寻找一个合适的索引(sock状态是SOCK\_IDLE，UDPchannel的状态是DISCONNECTED)，并把地址m\_socks的地址赋上对方的，并把索引返回给UAC\_Socket中的m\_uac\_fd，连接时通过返回值传递，被连接时通过uac\_attach返回；这样所有UACChannel与sockpool中的m\_socks都有一一的对应关系；

Sockpool的m\_socks索引与UDPConnector的m\_udps索引一致，因m\_socks中的ch即UDPChannel与m\_udps的handle同是UDPChannel

4）发送流程

发送是从上向下的过程，是上层推动下层

a) 在UACChannel中的send中传的是MemBlock，之后就都是buf了；

sockpool中的m\_socks的索引fd，记录在m\_socks中UDPChannel中的idx，idx定义在UDPChannel的父类中，也是在在UDPConnector中的索引中的索引，因为UDPChannel有很多，UDPConnector只有一个，并且UDPConnector中记录UDP会话m\_udps，即UDPConnector中的 b) m\_udps的索引方式与sockpool中的m\_socks索引方式是一样的，都是在sockpool初始中分配的，并记录在UDPChannel中的idx中，并且m\_udps[i]->handle记录的也正是UDPChannel；

c) UDPConnector连接时注册了一个信息在m\_udps中，连接成功后，把连接成功后的信息也存在了m\_udps，并保证sockpool与UDPConnector中的索引的一致同步性；

自己发打洞包，对方连接自己成功后，把记录对方的m\_udps的状态记录成UDP\_ACCEPTING；

d) sockpool中的m\_socks和UDPConnector中的m\_udps都记录的连接者与被连接者



5）接收过程是从下向上，是从底层到上层的过程



5）peer连接过程



peer根据tracker server返回来的源peer进行连接，最终会调用UDPConnector的打洞nat完成打洞的一系统操作，另外会向UAC\_SocketSelector注册可读；

过程可参见UDPConnector中的打洞过程

在sockpool只要连接成功，即变成SOCK\_CONNECTED，即会在select中写文件集中置位，sockpool的select最终被UAC\_SocketSelector的handle\_readwrite调用，调用后会把UAC\_Socket的状态变成UAC\_CONNECTED，注册UAC\_SocketSelector中可读；

连接的时候已经在UAC\_Socket中把UACChannel注册了可写，在连接成功后会在sockpool的select中置可写，这样会直接调用UAC\_Socket的uac\_on\_write，加上在UAC\_Socket中状态还是UAC\_CONNECTING，调用子类的uac\_on\_connected，状态变UAC\_CONNECTED，触发回调函数连接成功，通知Peer，进行通知peerManger，这样就会向对方发送WELCOME消息；

## 3.5 UAC\_SocketSelector

### 3.5.1主要成员变量

Node\_t\* m\_chs //

UAC\_fd\_set m\_rset,m\_wset //

### 3.5.2主要成员函数

register\_socket：UAC\_Socket连接（写）、发送（写）、连接上（读）会向UAC\_SocketSelector注册，这样本层可以直接处理发送、接收、连接等，状态保存到m\_chs中；

handle\_readwrite：根据sockpool的select结果，得到可读可写的文件符集，可读调用UAC\_Socket的uac\_on\_read，可写调用其uac\_on\_write；

handle\_accept：根据sockpool的accept结果，若有则把地址attach到PeerManger；

## 3.6 Peer

记录管理用户连接、发送、接收、断开的载体

### 3.6.1主要成员变量

Channel\* m\_ch; //不同的peer类型，定义的不同，IPT\_TCP对应的是TCPFWChannel，IPT\_UDP对应的UACChannel，IPT\_HTTP对应的是TCPChannel

MemBlock\* m\_block; //数据接收的内存块

### 3.6.2主要成员函数

attach：调用相应Channel的attach；

connect：调用相应Channel的connect；

send：调用相应Channel的发送;

on：各种回调函数，是对应Channel中回调的，on的可读可写还会回调PeerManger中去；

### 3.6.2重要数据结构

1) UACChannel

继承UAC\_Socket，与UAC\_Socket完成UDP通道的数据的发送、连接、接收的传递，总会把UACChannel注册到UACChannel的UAC\_WRITE或UAC\_READ，连接注册的是可写，连接成功后注册的是可读

uac\_connect：会调用sockpool的连接，向UAC\_SocketSelector注册可写；

uac\_send：向UAC\_SocketSelector注册可写，调用sockpool的发送；

uac\_recv：调用uac\_recv的接收；

uac\_on\_read：告诉peer数据可读；

uac\_on\_write：若是连接而调用的，反注册可写，并调用uac\_on\_connected，否则可写；

uac\_on\_connected：连接成功，注册可读；

2）回调机制

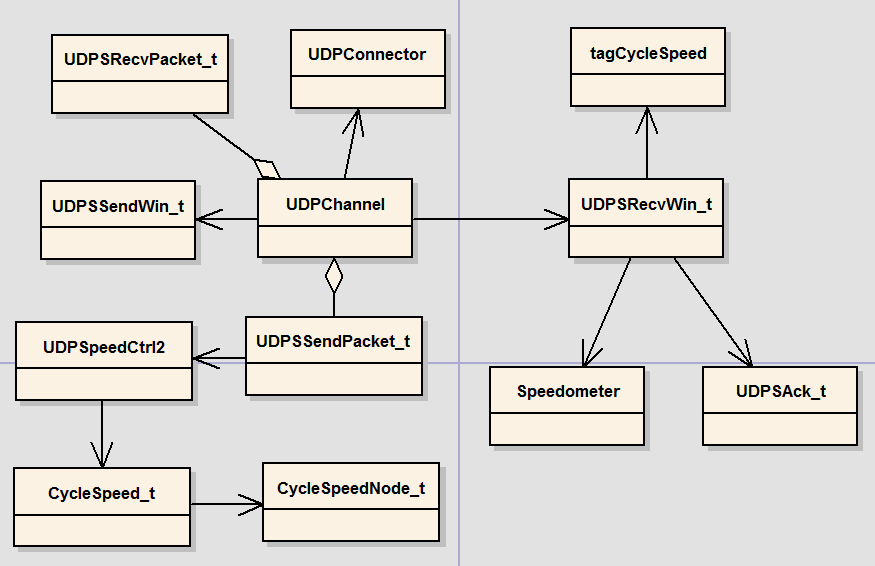


UACChannel中的Readable，Writable，Connected，Connecting，Disconnected回调peer的on；

peer的Connecting，Connected，Disconnected，Writable，Data回调PeerManger中的各种on；

## 3.7 UDPChannel

UDPchannel是实际UDP的发送和接收处理部分，包括发送与响应的处理和一些控制算法，主要是保证UDP的可靠传输及控制，类似http对应https。



### 3.7.1主要成员变量

UDPConnector\* m\_ctr; //连接用的

UDPSSendWin\_t m\_send\_win; //发送窗口

UDPSRecvWin\_t m\_recv\_win; //接收窗口

SendList m\_send\_list; //发送列表

RecvMap m\_recv\_map; //接收map

### 3.7.2主要成员函数

send：发送包括UDPS头，包头命令是UDPS\_CONN\_CMD\_DATA，数据头命令是UDPS\_CHANNEL\_CMD\_DATA，还有一种是UDPS\_CHANNEL\_CMD\_ACK，发送数据包中包含序列号，方便回复响应。最后把包放入发送队列m\_send\_list；

send\_to：如是数据UDPS\_CHANNEL\_CMD\_DATA，打上速度编号，并调用UDPChannelHandler的发送；

handle\_send：处理发送列表m\_send\_list中packet，；

handle\_recv\_data：接收数据的处理，；

send\_ack：发送ack响应，每一次最多回复100个，每个ack回复3次，即最多回复33个ack；

### 3.7.3重要数据结构

#### 3.7.3.1发送控制

1）发送窗口UDPSSendWin\_t

int win\_low\_line;//窗口下界，对方未确认的下界，即对方想要的下一个数据

unsigned int ack\_sequence; //响应的序列号

unsigned int nak\_num;

int send\_num;//下次发送新包使用的序列号

int real\_send\_num;//真实已经发送到的位置，可能上层多发，暂存队列

unsigned int ttl;

unsigned int maxttl; //最近时间内最大TTL

UDPSpeedCtrl2 spctrl;

ArrRuler<unsigned int,10> ttls; //回路TTL

unsigned int resend\_num;//重发了多少个包

unsigned int resend\_timeo\_num;//重发了多少个包

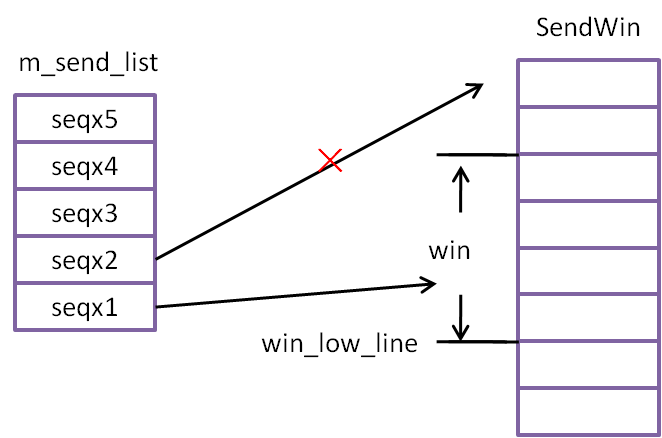
unsigned int other\_rerecv\_num;//对方收到多少个重复包

uint64 send\_sizeB; //一共发了多少数据

unsigned char not\_more\_data\_count; //无持续增加发送数据时，超时重发周期变短

主要接口：

send：发送的数据都要封装成UDPSSendPacket\_t结构体放入m\_send\_list发送列表；



UDPSSendPacket\_t｛

int send\_num;

int send\_count;

int nak\_count; //预计丢失计数

ULONGLONG last\_send\_tick; //用于记录开始发送包时间，当收到ACK时，可以统计该包一个来回用时多少

memblock \*block;//带head｝

handle\_send：处理发送，从发送列表中取发送包处理，限制发送的序列号小于win+win\_low\_line：

若该packet正常发送，即还没有发送过，直接调用send\_to发送；

若发送过一次并且丢包超过一次，就开始转入控制算法，重发；

否则，若丢包了，并且还可能再次丢包（根据时间估计），则重发；

否则则超时重发

发送完就开始调用spctrl.csp.on\_tick统计；

send\_to：给发送数据打上速度编号(m\_send\_win.spctrl.csp.speed\_seq)，一个统计周期使用同一编号，调用spctrl.csp.on\_send，累积size和num，最后调用UDPChannelHandler的发送函数；

2）控制发送窗口UDPSpeedCtrl2

int level;

unsigned int other\_recv\_win\_num;

unsigned int send\_win; //

unsigned int ttlus;//包的往返平均时间，微秒

unsigned int mtu;

CycleSpeed\_t csp; //循环速度

unsigned int last\_change\_win\_tick;

unsigned int last\_send\_tick;

主要接口：

on\_recv\_ack：使用的level分为LEVEL\_WIN和LEVEL\_SPEED，初始阶段采用快速增长速度，应该是收到ack响应就会调用，当send\_win小于other\_recv\_win\_num时，主要是增加send\_win；

get\_max\_send\_packnum：获取最大发送序列号，先计算发送速度，即每秒发送字节数

speed= send\_win\*( mtu+ UDPHEAD\_LENGTH)/ (ttlus/(double)1000000))

num= speed\*t/ mtu + 1，这样就计算出了要发送的序列号；

3）速度控制窗口的CycleSpeed\_t

状态有CSP\_FREE,CSP\_SENDING,CSP\_SENDFINI,CSP\_RECVFINI

unsigned int speedB; //初始化100KB速度

unsigned int const\_speedB; //固定发送速度，由g\_udps\_conf获取

unsigned short max\_lose\_rate; //固定丢码率，由g\_udps\_conf获取

unsigned char speed\_seq; //初始为0不使用

ULONGLONG begin\_tick;

ULONGLONG end\_tick;

unsigned short num;//初始化为0

unsigned int sizeB; //字节数

CycleSpeedNode\_t csn[CSP\_LEN]; //发送速度周期是64

on\_send：发送数据时调用，更新end\_tick，sizeB和num；

on\_ack\_speed：接收到ack时调用，传入参数有序列号i, num, spB，状态置为CSP\_RECVFINI，统计丢码率，csn[i].recv\_num\*100/csn[i].num，若能找到上次有效数据，两次数据一块算，算的是平均，最后调用chang\_speed；

chang\_speed：调整序列i的发送速度，若速度低于最近接收(i)的速度，则调整为1.2倍；

否则若真实发送速度小于参考发送的65%，不调整，

若丢包率大于最大丢包率即速度降为10%；

否则(丢包率不大于xxx)，若速度小于真实速度的1.5倍，则根据码率调整速度，最小速度为30KB

|  |  |
| --- | --- |
| 丢包率 | 提高速度 |
| 0 | Speed+50K |
| 0 ~ 0.33max | Speed+30K |
| 0.33~0.75max | Speed+10K |
| 0.75max~max | Speed |
| >max | 0.9\*Speed |

on\_tick：执行每次发送完成后调用，一个速度控制周期是250ms，

距离上次发送在2s内才将信息统计到csn中，包括num，真实速度和状态（CSP\_SENDFINI）

全部应该是64\*250ms=16全部循环一次，类似速度统计模块；

#### 3.7.3.2接收控制

1）接收窗口UDPSRecvWin\_t

int win\_low\_line;//窗口下界，未确认的下界

int recv\_win\_num; //考虑减掉被未应用层接收的数量

bool brecv\_win\_num\_changed; //相对回复ACK,接收窗更新时为true,回复后变为FALSE

unsigned int max\_recv\_win\_num; //最多可以接收多少个包

int max\_sequence\_num; //收到的最大包序号

int rerecv\_num;//重收了多少个包

//ACK

UDPSAck\_t ack; //

unsigned int ack\_sequence; //响应序列号

unsigned int last\_ack\_tick;

unsigned int unack\_recv\_num; //未回复ACK前收到的包数，用来简单计数，达到一定值会统一进行ack响应；

int last\_ack\_win\_low\_line; //相对回复ACK

int resend\_ack\_win\_low\_line\_count; //重复回复同一个low\_line计数,当变化或者有收到更新的包时置0

Speedometer<unsigned int> speed;

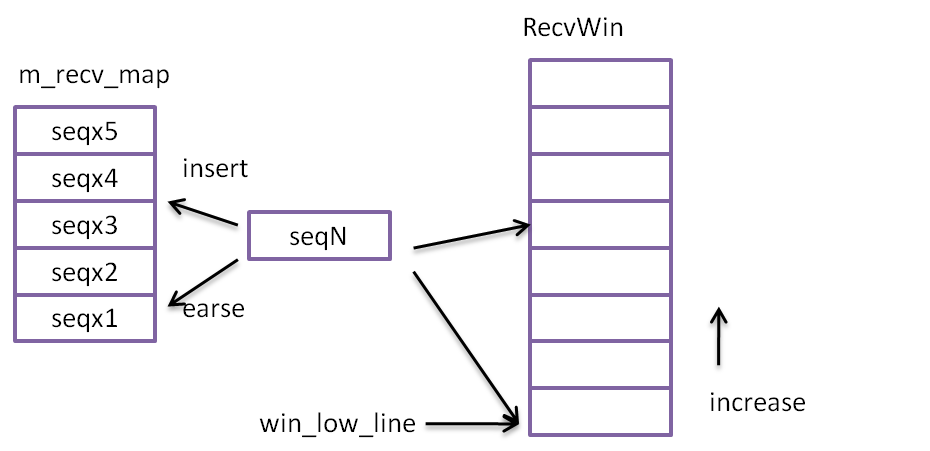
uint64 recv\_sizeB; //一共收了多少数据

handle\_recv\_data：接收数据处理：

若收到的包的序列号与win\_low\_line相同，则更新win\_low\_line，并从m\_recv\_map中移除该序列号所的packet；

若收到包的序列号大于win\_low\_line，则在m\_recv\_map中查找是否有该序列号的包，若有则是重复包，更新接收信息，否则把收到的包存入到m\_recv\_map中；

等到未回复的包积累到一定数，比如10再统一回复，



handle\_recv\_ack：发送方收到接收方反馈的消息：

更新发送方发送窗的ack\_sequence，调用spctrl.csp.on\_ack\_speed计算丢包率；

更新发送窗口的左边界win\_low\_line；

统计发送数据的not ack统计，以及计算ttl时间；

从发送列表send\_list中去除掉过时包。

2）接收win的csp

速度统计周期,相同编号的包统计速度,速度计算丢掉第一个包,数据包含第一个包

unsigned char speed\_seq; //速度序列号

ULONGLONG begin\_tick;

ULONGLONG end\_tick;

unsigned short num;

unsigned int sizeB;

unsigned char last\_speed\_seq;

unsigned short last\_num;

unsigned int last\_speedB;

on\_recv：接收到数据的调用部分，更新end\_tick，num和sizeB，若是新序列号则统计，并把相关信息存储到last\_xxx中；

# 4. web获取配置文件

使用静态网页的形式部署webserver，peer端启动时向webserver获取配置文件，手机端执行程序vkkp2p\_android.e -zone “http://103.244.165.191:4000/p2p/webvodpeer.ini”，这样把地址传过去，在process\_conf会向webserver发请求；

[comm]

version=3000

tracker\_ip=103.244.165.191

tracker\_port=4001

stun\_ip=103.244.165.191

stun\_port=4006

accept\_ip=

accept\_port=7170 //tcp与udp使用同一端口

http\_port=14188

message\_ip=

message\_port=0 //向外发送消息的ip及port，广播信息用

[local]

user\_id=

user\_type=1

user\_ip=0.0.0.0

menu=7 //菜单功能位:低0位共享支持,低1位TCP支持,低2位UDP支持

is\_minor\_version=0 //次版本不写ready，major version写

block\_size=102400

sn\_multinum=1 //默认为1,指同一个源建立多个连接

[cache]

cache\_path=cache/

download\_path=

cache\_sizeMB=1024

disk\_min\_free\_spaceMB=1024 //保持最少磁盘剩余空间

cache\_rdbf=0

cache\_flag=0

cache\_flag\_vod=1

memcache\_winMB=50

memcache\_playlist\_win=18

diskcache\_playlist\_num=18

is\_clear\_url2\_cache=1

urgent\_win=2 //紧急窗，抢动作，随机1

smooth\_win=6 //平稳窗，随机1； 应该为紧急窗的2倍以上为佳

random\_win=5 //随机选分块任务

http\_pause\_win=0 //允许http暂停的窗口大小，0表示不暂停

http\_no\_keep\_alive=0 //=1时表示不使用keep-alive功能

playlist\_http\_pause\_win=3,5 //使http暂停的窗口范围

[limit]

download\_speed\_KB=5000 //总的下载速度

downloadi\_cnns=50 //每个download的最大连接数

downloadhttpi\_cnns=1 //每个downloadhttp的最大连接数

share\_speed\_KB=0

share\_cnns=500

sharefiles\_max\_timer\_sec=120 //最多多少秒内共享完

download\_maxnum=3 //manual\_download 并发数

downloadlist\_maxnum=10 //最多支持多少个downloadlist 并发

downloadlist\_active\_partnum=2 //downloadlist的 片段下载并发数

min\_httpi\_speedKB=0 //如果大于0，侧HTTP连接速度过小时断开重连

[extra]

log\_mode=

preload\_playlist=playlist.inf

daemon\_mode=

playlist\_timeshift=0

playlist\_delaynumber=0

playlist\_size=8

export\_list=1

export\_prefix=./

export\_filename=playlist.m3u8

export\_logfile=export.log

export\_timeshift=0

hosts\_file=

localonly=1

# 5.其它模块

## 5.1 NattypeMgr模块

建立与tcc服务器的TCP连接，主要任务：

1）定时做nat类型检查；

2）与tccsvr完成测试是否在同一局域网的操作；

### 5.1.1主要成员函数

callback\_udp\_ipportchanged：网络ip和port改变时调用的回调；

init：获取本地ip；

start\_check\_tcp\_nattype：检测与tcc的连接，6秒内没断开即强行断开；

on(Connected,Peer\* peer)：连接tcc成功，向tcc发送PTL\_P2S\_REQUEST\_TCP\_CHECK命令；

## 5.2 升级功能

vodpeer可以自动升级的，采用从网页获取，我们布置网页服务器，http://103.244.165.191:4000/p2p/update/vodpeer\_update.ini的内容如下：

ver=vp-20141121

sha1=8003214134500d8408f156d3a5018aba6eb45f26

path=vodpeer.e

"p2p/update"目录下有vodpeer.e，即新版的peer端，sha1是新版peer的sha1码，作下载后的文件校验，ver是新版本的版本号

升级程序"vodpeer.e.updater"，生成程序的目录在“vkkp2p/comm/android/update/libs/armeabi”，这个要在第一次时与peer端代码一起打到jar包，以后只更新peer端代码，升级程序不需要升级。升级的流程如下：

原告执行的"./vodpeer "http://103.244.165.191:4000/p2p/webvodpeer.ini""

1）判定当前目录是否有vodpeer.newversion

若有使用system("./vodpeer.e.updater --reload ./vodpeer.newversion ./vodpeeer.e -zone http://103.244.165.191:4000/p2p/webvodpeer.ini ")，在updater中会把vodpeer.e.newversion修改为vodpeer.e，添加可执行行权限，并再次使用system("./vodpeer http://103.244.165.191:4000/p2p/webvodpeer.ini")；

若没有则调用system("./vodpeer.e.updater --updater vodpeer.bin vp-20141120 http://103.244.165.191:4000/p2p/update/vodpeer\_update.ini & ")，在updater中会比较当前版本与web中版本，若需要更新，则从“http://103.244.165.191:4000/update/vodpeer.e”下载vodpeer.e，即若有更新只是本次下载好，下次启动时会自动替换更新，不影响本次执行；

**Note**：为了防止每次打开都会下载更新，要保证web中vodpeer的程序中的版本号与web网页中的版本号一致。

# 6.策略/算法

## 6.1搜索源策略

1）默认3分钟搜索一次，无源时1分钟搜索1 次，源大于20时6分钟搜索一次；

2）连接tracker成功时搜索一次；

3）请求m3u8文件返回列表时，若无文件列表搜索一次；

## 6.2打洞连接策略

1）p2psource列表连接20次都失败的放弃源；

2）创建的"主连接/被连接"15秒后不成功断开；

3）连接15秒内没有收过数据即超时断开；

4）5秒内没收到过数据，保活一次；

5）"被连接"每0.8秒重发一次"ok"（未收到洞包的）；

6）主连接每0.8秒发一次"connect"；

7）测速超过3秒未完成，则当完成；

8）20s探测绑定端口；

9）探测类型时0.6发送一次探测包，3s内没有收到需要相应的数据进入下一类型的探测；

## 6.3定时更新策略

1）检测速度为0一段时间后，中止本下载开始下一个下载；

2）每20分钟报一次下载信息；

3）10s更新一次播放列表；

4）周期性的搜索源；

## 6.4 httpi策略

配置文件定义http下载窗口playlist\_http\_pause\_win=3,5

1）当已缓存文件数>=cache\_playlist\_http\_win\_up(5)满时执行httpi，更新m\_http\_max值，就样就不会主动请求http源了，让P2P分享更多数据

2）当已缓存文件数<cache\_playlist\_http\_win\_down(3)时执行会重新请求http源；

## 6.5 数据下载及任务分配策略

1）完成1块后尝试继续分任务；

2）我（peer）速度比下载同块的其它peer速度低于6k，不再下载该块；

3）一次最多请求1024个块；