# 1 P2S协议

Peer to Stun的缩写，是打洞及NAT探测协议协议，STUN，英文全称为" Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs) "，几种常见的nat类型这里不多解析，参见文档“p2p业务说明\_补充.docx”的”3.2 NAT知识“，本P2P系统中nat类型的定义如表所示。

NAT类型

|  |  |
| --- | --- |
| NAT类型 | 类型解释 |
| nat1 | 完全透明NAT(Full |
| nat2 | 受限NAT(Restricted |
| nat3 | 端口受限NAT(Port |
| nat4 | 对称NAT(Symmetric |
| nat5 | 无法接收外部包 |
| nat6 | 末知类型 |

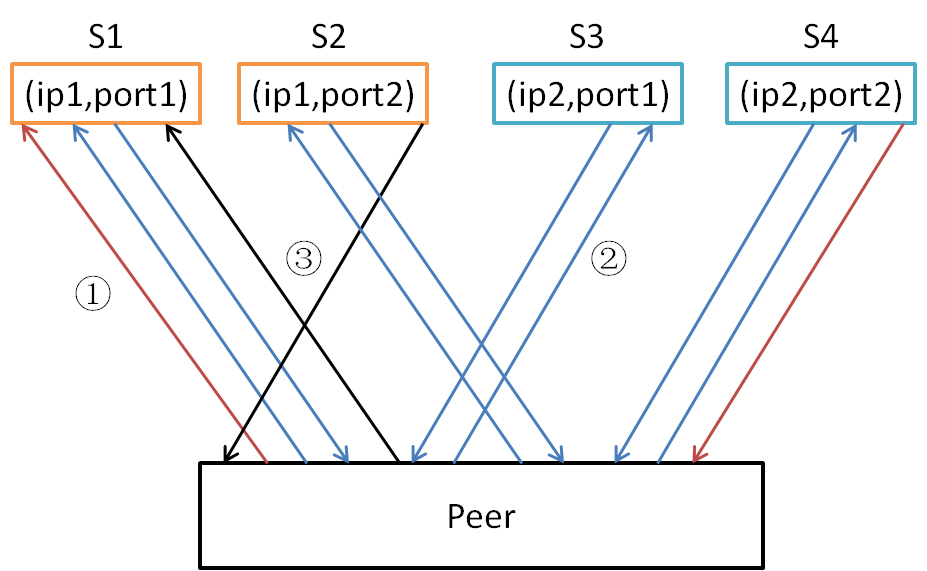
## 1.1 NAT类型探测

### 1.1.1 探测条件

1）stun server由两个不同IP地址的服务器（公网ip）对组成，设A、B服务器。

2）A、B分别绑定两个端口：S1(IP1:Port1)，S2(IP1:Port2)，S3(IP2:Port1)，S4(IP2:Port2)

协议：



探测NAT类型

### 1.1.2探测流程

1）检测客户端是否有能力进行UDP通信及客户端是否位于NAT之后，与S1建立UDP通信，若有数据接收，说明防火墙可能过，确定返回的ip及port与本地的ip及port是否相等，若相等则客户端是公网ip，不在NAT之后，否则下一步；

2）检测是否是NAT1（Full Cone NAT），S1发，使用S4发，若收到数据则为NAT1，否则下一步；

3）检测是否是NAT4（Symmetric　NAT），S1，S2，S3，S4发若收到的ip及port有一对不同则为NAT4；

4）检测是否NAT2，S1发，S2回，若收到则为NAT2，否则为NAT3。

代码中探测及响应的过程如下：

1.req-binding-addr / rsp-binding-addr ：获取本地绑定的外网IP：PORT

2.req-stuns-addr / rsp-stuns-addr ：获取stun server :A、B的信息S1,S4

3.req-full-conn / rsp-full-conn ：探NAT1。 发S1，S4回，收到即是NAT1

4.req-symmetric / rsp-symmetric ：探NAT4。 发S1、S3，S1、S3回IP：PORT，不相同即是NA4

5.req-restricted-conn /rsp-restricted-conn ：探NAT2。发S1、S2回，收到即NA2，不然为NAT3

### 1.1.3 NAT的连通性

NAT类型可连接通性匹配

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | nat1 | nat2 | nat3 | nat4 |
| nat1 | 1 | 1 | 1 | 1 |
| nat2 | 1 | 1 | 1 | 1 |
| nat3 | 1 | 1 | 1 | 0 |
| nat4 | 1 | 1 | 0 | 0 |

表中列和行分别表示请求打洞和被连接者NAT类型，”1“表示可以打洞成功，”0“表示不能打洞成功

DownloadSource中"g\_ConnectionType"定义的连接类型

其中1维为自己nat类型，2维为目标类型，标灰的是外网ip

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Nat1** | **Nat2** | **Nat3** | **Nat4** | **Nat5** | **Nat6** |
|  | **TCP\_CONN** | **TCP\_TURN** | **TCP\_TURN** | **TCP\_TURN** | **TCP\_TURN** | **TCP\_TURN** | **UNUSABLE** |
| **Nat1** | **TCP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UNUSABLE** | **UNUSABLE** |
| **Nat2** | **TCP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UNUSABLE** | **UNUSABLE** |
| **Nat3** | **TCP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** |
| **Nat4** | **TCP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** |
| **Nat5** | **TCP\_CONN** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** |
| **Nat6** | **TCP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UDP\_CONN** | **UNUSABLE** | **UNUSABLE** | **UNUSABLE** |

|  |  |
| --- | --- |
| 连接类型 |  |
| TCP\_CONN | TCP连接 |
| TCP\_TURN | TCP反连接 |
| UDP\_CONN | UDP连接 |
| UNUSABLE | 不可用 |
| UNKNOW | 未知类型 |

SourceService中

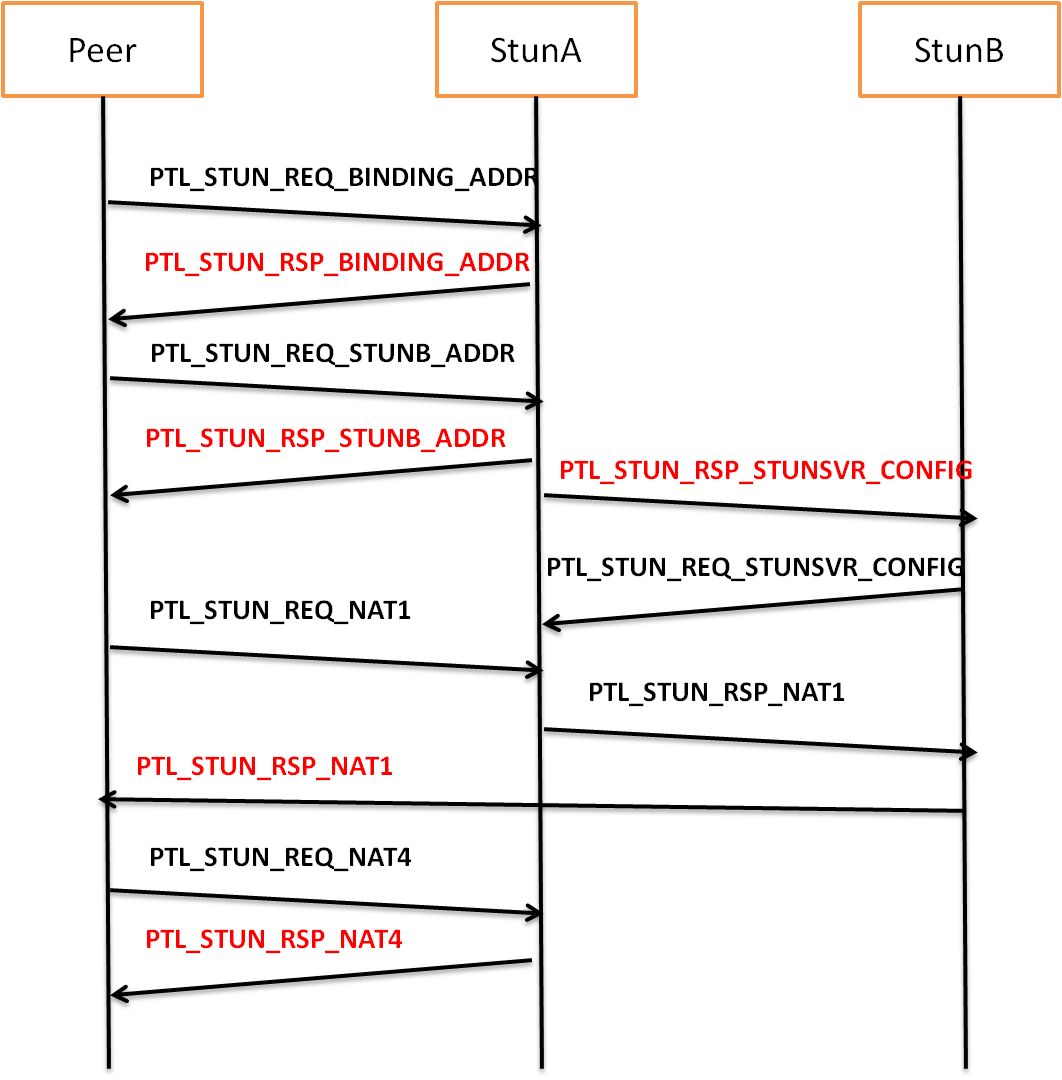
//一维下标:自身的nat类型，二维下标:对方nat类型

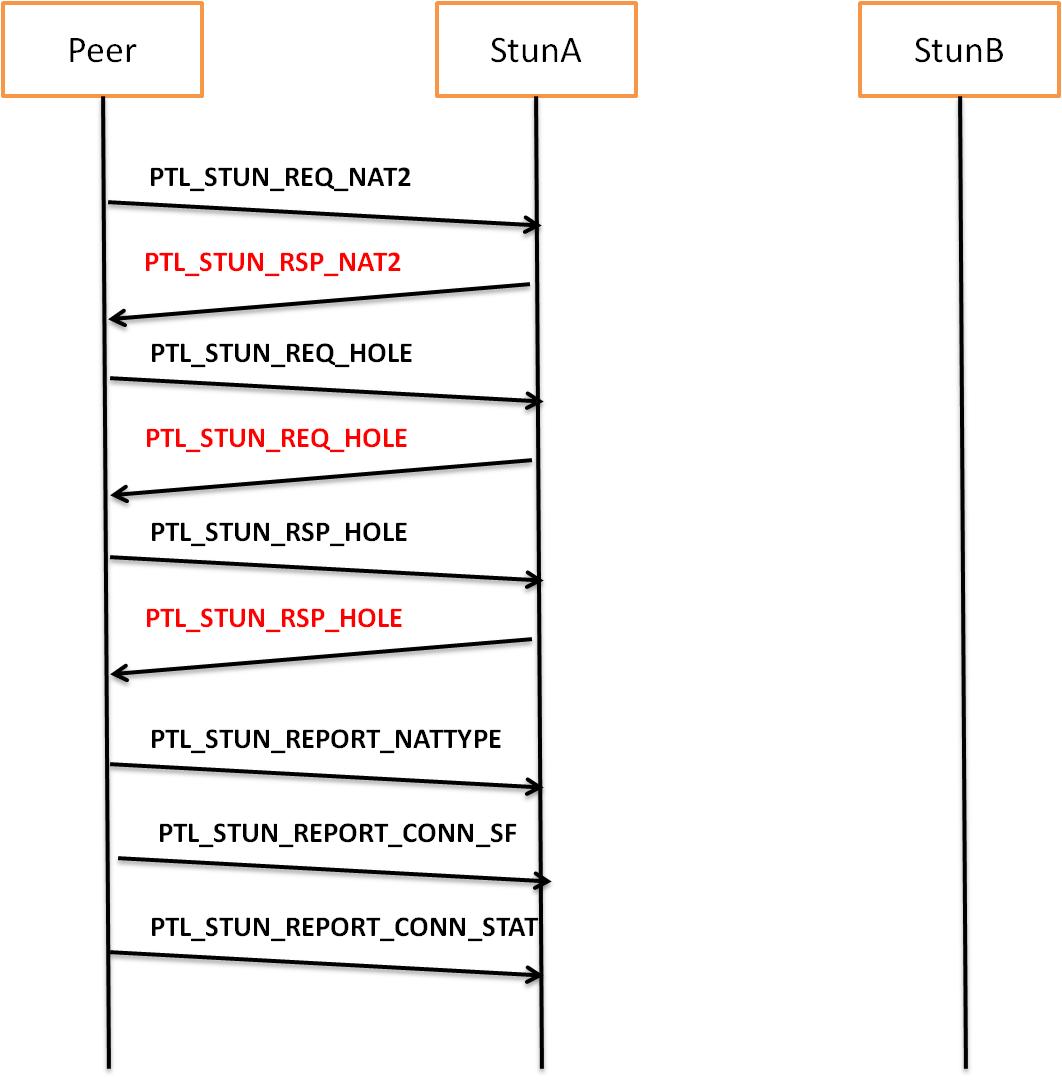
//7表未知类型,自己是6的,当自己是3类型，即只返回0，1，2，3

static char g\_ConnectionType[7][7]，标灰的是外网ip

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Nat1 | Nat2 | Nat3 | Nat4 | Nat5 | Nat6 |
|  | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| Nat1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Nat2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Nat3 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| Nat4 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Nat5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nat6 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

## 1.2 协议命令定义及数据结构





### 1.2.1 打洞命令定义

#define PTL\_STUN\_HEAD\_STX 0x8f

Stun通信判定NAT类型命令：

PTL\_STUN\_REQ\_UID //获取uid,用于后续命令的认证

PTL\_STUN\_RSP\_UID //返回uid,用于后续命令的认证

PTL\_STUN\_REQ\_BINDING\_ADDR //获取绑定IP：PORT

PTL\_STUN\_RSP\_BINDING\_ADDR //

PTL\_STUN\_REQ\_STUNB\_ADDR, //获取S3(SBIP:P1)

PTL\_STUN\_RSP\_STUNB\_ADDR //

PTL\_STUN\_REQ\_STUNSVR\_CONFIG //获取SB的配置

PTL\_STUN\_RSP\_STUNSVR\_CONFIG //

PTL\_STUN\_REQ\_NAT1, //测NAT1

PTL\_STUN\_RSP\_NAT1, //

PTL\_STUN\_REQ\_NAT4 //测NAT4

PTL\_STUN\_RSP\_NAT4 //

PTL\_STUN\_REQ\_NAT2 //测NAT2

PTL\_STUN\_RSP\_NAT2 //

PTL\_STUN\_REQ\_HOLE //请求目标发NAT\_HOLE包,stun 转发给另一端

PTL\_STUN\_RSP\_HOLE //请求目标发NAT\_HOLE包,stun 转发给另一端

//统计命令

PTL\_STUN\_REPORT\_NATTYPE //上报NAT类型

PTL\_STUN\_REPORT\_CONN\_SF //上报连接成功失败

PTL\_STUN\_REPORT\_CONN\_STAT //上报

Stun打洞命令包类型：

UDPS\_CONN\_CMD\_NAT //打洞包

UDPS\_CONN\_CMD\_CONN //请求新连接包

UDPS\_CONN\_CMD\_OK //连接就绪包

UDPS\_CONN\_CMD\_LIVE //通道保活包

UDPS\_CONN\_CMD\_CLOSE //断开通道包

UDPS\_CONN\_CMD\_DATA //通道数据包

UDPS\_CONN\_CMD\_TESTSPEED\_DATA //测速数据包

UDPS\_CONN\_CMD\_TESTSPEED\_ACK //测速回应包

### 1.2.2 打洞定义的数据结构

typedef struct tagPTL\_STUN\_Header

{

uchar stx; //

uchar cmd; //命令值

uint32 uid; //

uint32 uid\_checksum;

tagPTL\_STUN\_Header(void):stx(PTL\_STUN\_HEAD\_STX),cmd(0),uid(0),uid\_checksum(0){}

}PTL\_STUN\_Header\_t;

//BindingAddr

//typedef struct tagPTL\_STUN\_ReqBindingAddr

//{

//}PTL\_STUN\_ReqBindingAddr\_t;

typedef struct tagPTL\_STUN\_RspBindingAddr

{

uint32 eyeIP;

uint16 eyePort;

}PTL\_STUN\_RspBindingAddr\_t;

//StunsAddr

//typedef struct tagPTL\_STUN\_ReqStunBAddr

//{

//}PTL\_STUN\_ReqStunBAddr\_t;

typedef struct tagPTL\_STUN\_RspStunBAddr

{

uint32 stunB\_ip;

uint16 stunB\_port1;

uint32 stunC\_ip;

uint16 stunC\_port1;

}PTL\_STUN\_RspStunBAddr\_t;

//StunsConfig

//typedef struct tagPTL\_STUN\_ReqStunsvrConfig

//{

//}PTL\_STUN\_ReqStunsvrConfig\_t;

typedef struct tagPTL\_STUN\_RspStunsvrConfig

{

uint32 eyeIP;

uint16 eyePort;

uint32 accept\_ip;

uint16 accept\_port1;

uint16 accept\_port2;

uint32 stunB\_ip;

uint16 stunB\_port1;

uint16 stunB\_port2;

uint32 stunC\_ip;

uint16 stunC\_port1;

}PTL\_STUN\_RspStunsvrConfig\_t;

//Nat1

//typedef struct tagPTL\_STUN\_ReqNat1

//{

//}PTL\_STUN\_ReqNat1\_t;

typedef struct tagPTL\_STUN\_RspNat1

{

uint32 des\_nip; //stun server端用于转发，client端忽略

uint16 des\_nport; //

}PTL\_STUN\_RspNat1\_t;

////Nat4

//typedef struct tagPTL\_STUN\_ReqNat4

//{

//}PTL\_STUN\_ReqNat4\_t;

typedef struct tagPTL\_STUN\_RspNat4

{

uint32 eyeIP;

uint32 eyePort;

}PTL\_STUN\_RspNat4\_t;

//Nat2

//typedef struct tagPTL\_STUN\_ReqNat2

//{

//}PTL\_STUN\_ReqNat2\_t;

typedef struct tagPTL\_STUN\_RspNat2

{

uint32 des\_nip; //stun server端用于转发，client端忽略

uint16 des\_nport; //

}PTL\_STUN\_RspNat2\_t;

//Hole

typedef struct tagPTL\_STUN\_ReqHole

{

uint32 connid; //标识连接号，回复同样标识以确定是哪个连接发出的包

uint32 des\_nip; //呼叫des\_nip:des\_nport 发NAT\_TEST包给src\_nip:src\_nport

uint16 des\_nport; //

uint32 src\_nip; //代表发起连接方

uint16 src\_nport;

}PTL\_STUN\_ReqHole\_t;

typedef struct tagPTL\_STUN\_RspHole

{

uint32 connid; //标识连接号，回复同样标识以确定是哪个连接发出的包

uint32 des\_nip; //回复，

uint16 des\_nport; //

uint32 src\_nip; //代表发起连接的另一方

uint16 src\_nport;

}PTL\_STUN\_RspHole\_t;

//report nattype

typedef struct tagPTL\_STUN\_ReportNattype

{

uchar nattype;

}PTL\_STUN\_ReportNattype\_t;

//conn sf信息只有发起方上报

//report conn sf

typedef struct tagPTL\_STUN\_ReportConnSF

{

//ip:port由stunsvr自己获取

uchar bsucceed;

uchar src\_nattype;

uint32 des\_nip;

uint16 des\_nport;

uchar des\_nattype;

}PTL\_STUN\_ReportConnSF\_t;

//conn 信息只有发送数据比接收数据多时写记录

//report conn stat

typedef struct tagPTL\_STUN\_ReportConnStat

{

//ip:port由stunsvr自己获取

uchar src\_nattype;

uint32 des\_nip; //

uint16 des\_nport; //

uchar des\_nattype;

uint32 sizeKB; //有效传输总数据量KB

uint32 sec; //传输总秒数

uint32 ttlMS; //只取最后的记录

uint32 send\_num; //总共发出多少个有效包

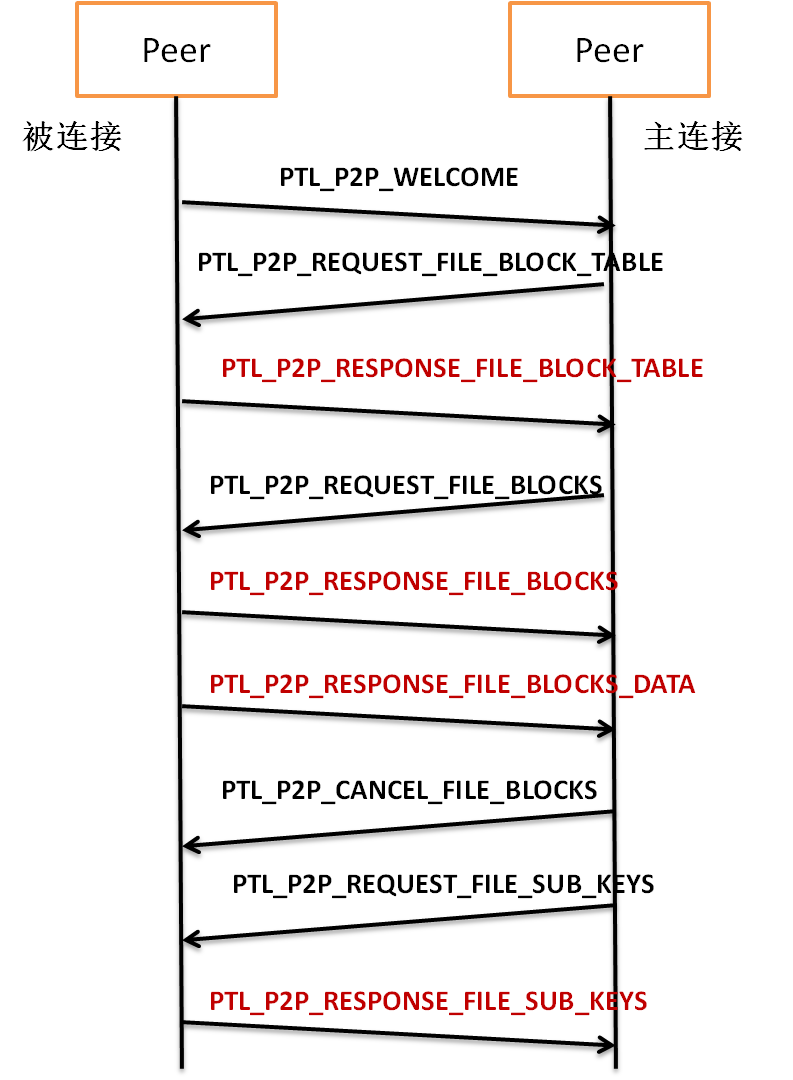
uint32 resend\_num; //总共重复发了多少个包

uint32 resend\_timeo\_num; //总共超时重发出多少个包

uint32 other\_rerecv\_num; //对方重复收了多少个包

}PTL\_STUN\_ReportConnStat\_t;

# 2 P2P协议



### 2.1协议命令定义

PTL\_P2P\_WELCOME

PTL\_P2P\_RELEASE

PTL\_P2P\_REQUEST\_FILE\_BLOCK\_TABLE

PTL\_P2P\_RESPONSE\_FILE\_BLOCK\_TABLE

PTL\_P2P\_REQUEST\_FILE\_BLOCKS

PTL\_P2P\_RESPONSE\_FILE\_BLOCKS

PTL\_P2P\_RESPONSE\_FILE\_BLOCKS\_DATA

PTL\_P2P\_CANCEL\_FILE\_BLOCKS

PTL\_P2P\_REQUEST\_FILE\_SUB\_KEYS

PTL\_P2P\_RESPONSE\_FILE\_SUB\_KEYS

### 2.2协议命令定义的数据结构

**PTL\_P2P\_WELCOME**

typedef struct tagPTL\_P2P\_Welcome

{

uint32 ver; //版本

uint32 sessionID; //反连接时确定对方，uid也可以

char utype; //user type,用户类型,TCP,UDP,HTTP

char ntype; //统计时使用

char turn; //识别是否为返连接，1为返连接

}PTL\_P2P\_Welcome

**PTL\_P2P\_REQUEST\_FILE\_BLOCK\_TABLE**

typedef struct tagPTL\_P2P\_RequestFileBlockTable

{

fhash\_t fhash; //请求url的hash值

uint32 blockSize; //允许请求不同块尺寸的共享

uint32 startBufI; //表示table表中的第几个字节起中的数据,即每8块1字节,startBufI\*8表示实际的绝对索引号

uint32 maxnum; //字节数，不是块数，一次不能要超过1024块

}PTL\_P2P\_RequestFileBlockTable;

**PTL\_P2P\_RESPONSE\_FILE\_BLOCK\_TABLE**

typedef struct tagPTL\_P2P\_ResponseFileBlockTable

{

uint32 result; //-1:没有文件，0：下载中，有块表，1：已经完成

fhash\_t fhash;

uint64 fsize;

uint32 blockSize;

uint32 startBufI; //表示table表中的第几个字节起中的数据

uint32 num;

char tableBuf[1024]; //32000\*100KB=3.200,000KB ; 即100K每块的话，可以表示3.2G大小文件了

}PTL\_P2P\_ResponseFileBlockTable;

**PTL\_P2P\_REQUEST\_FILE\_BLOCKS**

typedef struct tagPTL\_P2P\_RequestFileBlocks

{

fhash\_t fhash;

uint32 blockSize;

uint32 num; //后面2个数组的有效大小

uint32 indexs[32]; //块号

uint32 offsets[32]; //块内偏移

}PTL\_P2P\_RequestFileBlocks;

**PTL\_P2P\_RESPONSE\_FILE\_BLOCKS**

typedef struct tagPTL\_P2P\_ResponseFileBlocks

{

fhash\_t fhash;

uint32 blockSize;

uint32 num; //后面2个数组的有效大小

uint32 indexs[32]; //块号

uint32 blockState[32]; //0:缺少，1：存在

}PTL\_P2P\_ResponseFileBlocks;

**PTL\_P2P\_RESPONSE\_FILE\_BLOCKS\_DATA**

typedef struct tagPTL\_P2P\_ResponseFileBlocksData

{

fhash\_t fhash;

uint32 blockSize;

uint32 blockIndex;

uint32 offset;

uint32 size;

char\* data;

}PTL\_P2P\_ResponseFileBlocksData;

**PTL\_P2P\_CANCEL\_FILE\_BLOCKS**

typedef struct tagPTL\_P2P\_CancelFileBlocks

{

fhash\_t fhash;

uint32 blockSize;

uint32 num; //后面2个数组的有效大小

uint32 indexs[32]; //块号

}PTL\_P2P\_CancelFileBlocks;

**PTL\_P2P\_REQUEST\_FILE\_SUB\_KEYS**

typedef struct tagPTL\_P2P\_RequestFileSubKeys

{

fhash\_t fhash;

uint32 blockSize; //这里的分块大小与下载block的大小不一定一样，但最好是下载block的整倍数

uint32 num;

uint32 indexs[1024];

}PTL\_P2P\_RequestFileSubKeys;

**PTL\_P2P\_RESPONSE\_FILE\_SUB\_KEYS**

typedef struct tagPTL\_P2P\_ResponseFileSubKeys

{

uint32 result;

fhash\_t fhash;

uint32 blockSize; //这里的分块大小与下载block的大小不一定一样，但最好是下载block的整倍数

uint32 num;

uint32 indexs[1024];

uint32 keys[1024]; //子块值为一个4位整数

}PTL\_P2P\_ResponseFileSubKeys;

# 3.P2T协议



### 3.1协议命令定义

PTL\_P2T\_REQUEST\_LOGIN

PTL\_P2T\_RESPONSE\_LOGIN

PTL\_P2T\_REPORT\_LOGOUT

PTL\_P2T\_REPORT\_NAT

PTL\_P2T\_REPORT\_SHARE\_FILE

PTL\_P2T\_REPORT\_REMOVE\_FILE

PTL\_P2T\_REPORT\_START\_DOWNLOAD\_FILE

PTL\_P2T\_REPORT\_STOP\_DOWNLOAD\_FILE

PTL\_P2T\_REQUEST\_FILE\_SOURCE

PTL\_P2T\_RESPONSE\_FILE\_SOURCE

PTL\_P2T\_REPORT\_DOWNLOAD\_WRONG

PTL\_P2T\_REPORT\_DOWNLOAD\_FILE\_SPEED

PTL\_P2T\_REPORT\_STAT

PTL\_P2T\_REPORT\_ERROR

PTL\_P2T\_REQUEST\_CONN\_TURN

PTL\_P2T\_RESPONSE\_CONN\_TURN

PTL\_P2T\_REQUEST\_KEEPLIVE

PTL\_P2T\_RESPONSE\_KEEPLIVE

PTL\_P2T\_REQUEST\_SERVER\_LIST

PTL\_P2T\_RESPONSE\_SERVER\_LIST

PTL\_P2T\_REPORT\_DOWNLOAD\_FILE\_INFO

PTL\_P2T\_REPORT\_START\_DOWNLOAD\_LIST

PTL\_P2T\_REQUEST\_START\_DOWNLOAD\_LIST

PTL\_P2T\_REQUEST\_STOP\_DOWNLOAD\_LIST

PTL\_P2T\_REPORT\_DOWNLOADLIST\_MAXNUM

### 3.2协议命令定义的数据结构

//size=56B

typedef struct tagPTL\_P2T\_FileInfo

{

fhash\_t fhash;

uint64 fsize; //无穷大为-1，用于直播

}PTL\_P2T\_FileInfo;

//size=38B

typedef struct tagPTL\_P2T\_PeerInfo

{

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;

}PTL\_P2T\_PeerInfo;

//size=38B+PUIDLEN = 70B

typedef struct tagPTL\_P2T\_ServerInfo

{

puid\_t uid;

uint32 ver;

char ntype;

char utype; // user type

uint32 menu;

uint32 sessionID;

uint32 beginTime;

uint32 sourceNum;

uint32 tcpRealIP;

uint32 udpRealIP;

uint16 tcpRealPort;

uint16 udpRealPort;

char reserve[4];

}PTL\_P2T\_ServerInfo;

////peer to tracker

#define MENU\_VIP 0x8

typedef struct tagPTL\_P2T\_RequestLogin

{

puid\_t uid; //用户ID

char utype; //用户类型

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,低3位表示是否为VIP

uint32 ver;

uint32 sysver;

uchar mac[6];

}PTL\_P2T\_RequestLogin;

typedef struct tagPTL\_P2T\_ResponseLogin

{

sint32 result;

puid\_t uid; //用户ID

uint32 trackID;

uint32 trackVer;

uint32 sessionID;

uint32 eyeIP;

char reserve[16];

}PTL\_P2T\_ResponseLogin;

//空包

//typedef struct tagPTL\_P2T\_ReportLogout

//{

//}PTL\_P2T\_ReportLogout;

typedef struct tagPTL\_P2T\_ReportNat

{

char ntype; //nat类型

uint32 tcpLocalIP;

uint32 tcpRealIP; //网络IP

uint16 tcpLocalPort;

uint16 tcpRealPort;

uint32 udpLocalIP;

uint32 udpRealIP; //网络IP

uint16 udpLocalPort;

uint16 udpRealPort;

}PTL\_P2T\_ReportNat;

typedef struct tagPTL\_P2T\_ReportShareFile

{

uint32 num;

PTL\_P2T\_FileInfo files[20];

}PTL\_P2T\_ReportShareFile;

typedef struct tagPTL\_P2T\_ReportRemoveFile

{

uint32 num;

PTL\_P2T\_FileInfo files[20];

}PTL\_P2T\_ReportRemoveFile;

typedef struct tagPTL\_P2T\_ReportStartDownloadFile

{

fhash\_t fhash;

}PTL\_P2T\_ReportStartDownloadFile;

typedef struct tagPTL\_P2T\_ReportStopDownloadFile

{

fhash\_t fhash;

}PTL\_P2T\_ReportStopDownloadFile;

typedef struct tagPTL\_P2T\_RequestFileSource

{

fhash\_t fhash;

uint32 maxnum;

}PTL\_P2T\_RequestFileSource;

typedef struct tagPTL\_P2T\_ResponseFileSource

{

uint32 result;

fhash\_t fhash;

uint64 fsize;

uint32 urlflag; //使用http url标志

uint32 num;

PTL\_P2T\_PeerInfo peers[20];

}PTL\_P2T\_ResponseFileSource;

typedef struct tagPTL\_P2T\_ReportDownloadWrong

{

fhash\_t fhash;

fhash\_t fhashnew;

}PTL\_P2T\_ReportDownloadWrong;

typedef struct tagPTL\_P2T\_ReportDownloadFileSpeed

{

fhash\_t fhash;

uint64 size;

uint32 speed; // KB/s

uint16 cacheTimes; //缓冲数

uint16 dragTimes; //拖动次数

uint32 downSeconds; //下载时长(秒)

uint32 cacheSenconds; //缓冲总时长

char reserve[16];

}PTL\_P2T\_ReportDownloadFileSpeed;

typedef struct tagPTL\_P2T\_ReportStat

{

uint32 connSucceedPerNetT[5]; //nat0~nat4各数据

uint32 connFailedPerNetT[5];

uint32 downBytesPerIPT\_KB[3]; //iptype\_tcp/iptype\_udp

uint32 shareBytesPerIPT\_KB[3]; //iptype\_tcp/iptype\_udp

uint32 downBytesPerUserT\_KB[6]; //预留可能增加多的类型

uint16 shareSeconds; //统计总共享时间

uint16 downSeconds;

char reserve[16];

}PTL\_P2T\_ReportStat;

typedef struct tagPTL\_P2T\_ReportError

{

uint16 dumpTimes;

uint16 downWrongBlocks; //下载错误block数

char reserve[32];

}PTL\_P2T\_ReportError;

typedef struct tagPTL\_P2T\_RequestConnTurn

{

uint32 desTrackID;

uint32 desSessionID;

}PTL\_P2T\_RequestConnTurn;

typedef struct tagPTL\_P2T\_ResponseConnTurn

{

PTL\_P2T\_PeerInfo desPeerInfo;

char reserve[4];

}PTL\_P2T\_ResponseConnTurn;

////keeplive 没有实体数据

//typedef struct tagPTL\_P2T\_RequestKeeplive

//{

//}PTL\_P2T\_RequestKeeplive;

////keeplive 没有实体数据

//typedef struct tagPTL\_P2T\_ResponseKeeplive

//{

//}PTL\_P2T\_ResponseKeeplive;

typedef struct tagPTL\_P2T\_RequestServerList

{

int maxnum;

}PTL\_P2T\_RequestServerList;

typedef struct tagPTL\_P2T\_ResponseServerList

{

uint32 trackID;

uint32 trackVer;

uint32 userNum;

uint32 beginTime;

uint32 allNum; //可能数量太多，分多包传送

uint32 startNum; //如果不为0，则为子包

uint32 num; //

PTL\_P2T\_ServerInfo servers[14]; //一次传14个左右,1K以内

}PTL\_P2T\_ResponseServerList;

typedef struct tagPTL\_P2T\_ReportDownloadFileInfo

{

char flag; //0表示下载过程中的，1表示下载停止时的记录

fhash\_t fhash;

uint64 size;

uint32 speed\_KB; // KB/s

uint32 downSeconds; //下载时长(秒)

uint16 cacheTimes; //缓冲数

uint16 dragTimes; //拖动次数

uint32 cacheSenconds; //缓冲总时长

uint16 connSucceedPerNetT[5]; //nat0~nat4各数据

uint16 connFailedPerNetT[5];

uint32 shareBytesPerIPT\_KB[2]; //分享了多少数据

uint32 downBytesPerIPT\_KB[3]; //iptype\_tcp/iptype\_udp/iptype\_http

uint32 downBytesPerUserT\_KB[6]; //client,server,http,center,super

}PTL\_P2T\_ReportDownloadFileInfo;

typedef struct tagPTL\_P2T\_ReportStartDownloadList

{

fhash\_t fhash;

char url[1024]; //VIP启动时用于通知服务器加速

}PTL\_P2T\_ReportStartDownloadList;

typedef struct tagPTL\_P2T\_RequestStartDownloadList

{

fhash\_t fhash;

char url[1024];

}PTL\_P2T\_RequestStartDownloadList;

typedef struct tagPTL\_P2T\_RequestStopDownloadList

{

fhash\_t fhash;

}PTL\_P2T\_RequestStopDownloadList;

//用于super分配VIP加速节目

typedef struct tagPTL\_P2T\_ReportDownloadListMaxnum

{

uint32 downloadlist\_maxnum;

}PTL\_P2T\_ReportDownloadListMaxnum;