

[首页](#) [资讯](#) [精华](#) [论坛](#) [问答](#) [博客](#) [专栏](#) [群组](#) [更多 ▼](#)

[您还未登录！](#) [登录](#) [注册](#)

淡泊明志，宁静致远

- [博客](#)
- [微博](#)
- [相册](#)
- [收藏](#)
- [留言](#)
- [关于我](#)

P2P之UDP穿透NAT的原理与实现

- 博客分类：
- [P2P](#)

[Socket网络协议Windows互联网软件测试](#)

P2P 之 UDP穿透NAT的原理与实现

原创：shootingstars

参考：<http://midcom-p2p.sourceforge.net/draft-ford-midcom-p2p-01.txt>

工程下载地址：[upload/2004_05/04052509317298.rar](#)

论坛上经常有对P2P原理的讨论，但是讨论归讨论，很少有实质的东西产生（源代码）。呵呵，在这里我就用自己实现的一个源代码来说明UDP穿越NAT的原理。

首先介绍一些基本概念：

NAT(Network Address Translators)，网络地址转换：网络地址转换是在IP地址日益缺乏的情况下产生的，它的主要目的就是为了解决地址重用。NAT分为两大类，基本的NAT和NAPT(Network Address/Port Translator)。

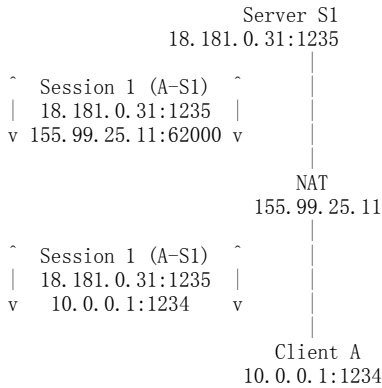
最开始NAT是运行在路由器上的一个功能模块。

最先提出的是基本的NAT，它的产生基于如下事实：一个私有网络（域）中的节点中只有很少的节点需要与外网连接（呵呵，这是在上世纪90年代中期提出的）。那么这个子网中其实只有少数的节点需要全球唯一的IP地址，其他的节点的IP地址应该是可以重用的。

因此，基本的NAT实现的功能很简单，在子网内使用一个保留的IP子网段，这些IP对外是不可见的。子网内只有少数一些IP地址可以对应到真正全球唯一的IP地址。如果这些节点需要访问外部网络，那么基本NAT就负责将这个节点的子网内IP转化为一个全球唯一的IP然后发送出去。（基本的NAT会改变IP包中的原IP地址，但是不会改变IP包中的端口）

关于基本的NAT可以参看RFC 1631

另外一种NAT叫做NAPT，从名称上我们也可以看得出，NAPT不但会改变经过这个NAT设备的IP数据包的IP地址，还会改变IP数据包的TCP/UDP端口。基本NAT的设备可能我们见的不多（呵呵，我没有见到过），NAPT才是我们真正讨论的主角。看下图：



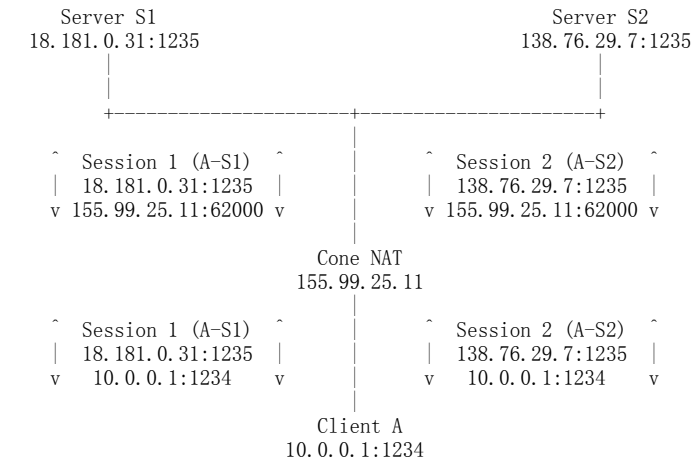
有一个私有网络10.*.*.*，Client A是其中的一台计算机，这个网络的网关（一个NAT设备）的外网IP是155.99.25.11(应该还有一个内网的IP地址，比如10.0.0.10)。如果Client A中的某个进程（这个进程创建了一个UDP Socket，这个Socket绑定1234端口）想访问外网主机18.181.0.31的1235端口，那么当数据包通过NAT时会发生什么事情呢？

首先NAT会改变这个数据包的原IP地址，改为155.99.25.11。接着NAT会为这个传输创建一个Session（Session是一个抽象的概念，如果是TCP，也许Session是由一个SYN包开始，以一个FIN包结束。而UDP呢，以这个IP的这个端口的第一个UDP开始，结束呢，呵呵，也许是几分钟，也许是几小时，这要看具体的实现了）并且给这个Session分配一个端口，比如62000，然后改变这个数据包的源端口为62000。所以本来是（10.0.0.1:1234→18.181.0.31:1235）的数据包到了互联网上变为了（155.99.25.11:62000→18.181.0.31:1235）。

一旦NAT创建了一个Session后，NAT会记住62000端口对应的是10.0.0.1的1234端口，以后从18.181.0.31发送到62000端口的数据会被NAT自动的转发到10.0.0.1上。（注意：这里说18.181.0.31发送到62000端口的数据会被转发，其他的IP发送到这个端口的数据将被NAT抛弃）这样Client A就与Server S1建立了一个连接。

呵呵，上面的基础知识可能很多人都知道了，那么下面是关键的部分了。

看看下面的情况：



接上面的例子，如果Client A的原来那个Socket(绑定了1234端口的那个UDP Socket)又接着向另外一个Server S2发送了一个UDP包，那么这个UDP包在通过NAT时会怎么样呢？

这时可能会有两种情况发生，一种是NAT再次创建一个Session，并且再次为这个Session分配一个端口号（比如：62001）。另外一种NAT再次创建一个Session，但是不会新分配一个端口号，而是用原来分配的端口号62000。前一种NAT叫做Symmetric NAT，后一种叫做Cone NAT。我们期望我们的NAT是第二种，呵呵，如果你的NAT刚好是第一种，那么很可能会有很多P2P软件失灵。（可以庆幸的是，现在绝大多数的NAT属于后者，即Cone NAT）

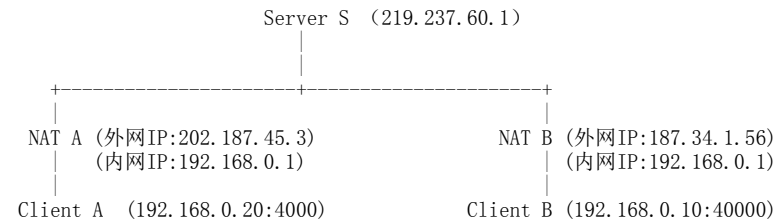
好了，我们看到，通过NAT, 子网内的计算机向外连结是很容易的（NAT相当于透明的，子网内的和外网的计算机不用知道NAT的情况）。

但是如果外部的计算机想访问子网内的计算机就比较困难了（而这正是P2P所需要的）。

那么我们如果想从外部发送一个数据报给内网的计算机有什么办法呢？首先，我们必须在内网的NAT上打上一个“洞”（也就是前面我们说的在NAT上建立一个Session），这个洞不能由外部来打，只能由内网内的主机来打。而且这个洞是有方向的，比如从内部某台主机（比如：192.168.0.10）向外部的某个IP(比如：219.237.60.1)发送一个UDP包，那么就在这个内网的NAT设备上打了一个方向为219.237.60.1的“洞”，（这就是称为UDP Hole Punching的技术）以后219.237.60.1就可以通过这个洞与内网的192.168.0.10联系了。（但是其他的IP不能利用这个洞）。

呵呵，现在该轮到我们的正题P2P了。有了上面的理论，实现两个内网的主机通讯就差最后一步了：那就是鸡生蛋还是蛋生鸡的问题了，两边都无法主动发出连接请求，谁也不知道谁的公网地址，那我们如何来打这个洞呢？我们需要一个中间人来联系这两个内网主机。

现在我们来看看一个P2P软件的流程，以下图为例：



首先，Client A登录服务器，NAT A为这次的Session分配了一个端口60000，那么Server S收到的Client A的地址是202.187.45.3:60000，这就是Client A的外网地址了。同样，Client B登录Server S，NAT B给此次Session分配的端口是40000，那么Server S收到的B的地址是187.34.1.56:40000。

此时，Client A与Client B都可以与Server S通信了。如果Client A此时想直接发送信息给Client B，那么他可以从Server S那儿获得B的公网地址187.34.1.56:40000，是不是Client A向这个地址发送信息Client B就能收到了呢？答案是不行，因为如果这样发送信息，NAT B会将这个信息丢弃（因为这样的信息是不请自来的，为了安全，大多数NAT都会执行丢弃动作）。现在我们需要的是在NAT B上打一个方向为202.187.45.3（即Client A的外网地址）的洞，那么Client A发送到187.34.1.56:40000的信息，Client B就能收到了。这个打洞命令由谁来发呢，呵呵，当然是Server S。

总结一下这个过程：如果Client A想向Client B发送信息，那么Client A发送命令给Server S，请求Server S命令Client B向Client A方向打洞。呵呵，是不是很绕口，不过没关系，想一想就很清楚了，何况还有源代码呢（侯老师说过：在源代码面前没有秘密 8）），然后Client A就可以通过Client B的外网地址与Client B通信了。

注意：以上过程只适合于Cone NAT的情况，如果是Symmetric NAT，那么当Client B向Client A打洞的端口已经重新分配了，Client B将无法知道这个端口（如果Symmetric NAT的端口是顺序分配的，那么我们或许可以猜测这个端口号，可是由于可能导致失败的因素太多，我们不推荐这种猜测端口的方法）。

下面是一个模拟P2P聊天的过程的源代码，过程很简单，P2PServer运行在一个拥有公网IP的计算机上，P2PClient运行在两个不同的NAT后（注意，如果两个客户端运行在一个NAT后，本程序很可能不能运行正常，这取决于你的NAT是否支持loopback translation，详见<http://midcom-p2p.sourceforge.net/draft-ford-midcom-p2p-01.txt>，当然，此问题可以通过双方先尝试连接对方的内网IP来解决，但是这个代码只是为了验证原理，并没有处理这些问题），后登录的计算机可以获得先登录计算机的用户名，后登录的计算机通过send username message的格式来发送消息。如果发送成功，说明你已取得了直接与对方连接的成功。

```
程序现在支持三个命令： send , getu , exit

send格式： send username message

功能：发送信息给username

getu格式： getu

功能：获得当前服务器用户列表
```

exit格式: exit

功能: 注销与服务器的连接(服务器不会自动监测客户是否吊线)

代码很短, 相信很容易懂, 如果有什么问题, 可以给我发邮件zhouhuis22@sina.com 或者在CSDN上发送短消息。同时, 欢迎转发此文, 但希望保留作者版权8-)。

最后感谢CSDN网友 PiggyXP 和 Seilfer的测试帮助

Cpp代码 ☆

```
1. //P2PServer.c
2. /* P2P 程序服务端
3. *
4. * 文件名: P2PServer.c
5. *
6. * 日期: 2004-5-21
7. *
8. * 作者: shootingstars(zhouhuis22@sina.com)
9. *
10. */
11. #pragma comment(lib, "ws2_32.lib")
12.
13. #include "windows.h"
14. #include "..\proto.h"
15. #include "..\Exception.h"
16.
17. UserList ClientList;
18.
19. void InitWinSock()
20. {
21.     WSADATA wsaData;
22.
23.     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
24.     {
25.         printf("Windows sockets 2.2 startup");
26.         throw Exception("");
27.     }
28.     else{
29.         printf("Using %s (Status: %s)\n",
30.             wsaData.szDescription, wsaData.szSystemStatus);
31.         printf("with API versions %d.%d to %d.%d\n\n",
32.             LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion),
33.             LOBYTE(wsaData.wHighVersion), HIBYTE(wsaData.wHighVersion));
34.     }
35. }
36.
37.
38. SOCKET mksock(int type)
39. {
40.     SOCKET sock = socket(AF_INET, type, 0);
41.     if (sock < 0)
42.     {
43.         printf("create socket error");
44.         throw Exception("");
45.     }
46.     return sock;
47. }
48.
49. stUserListNode GetUser(char *username)
50. {
51.     for(UserList::iterator UserIterator=ClientList.begin();
52.         UserIterator!=ClientList.end();
53.         ++UserIterator)
54.     {
55.         if( strcmp( ((*UserIterator)->userName), username) == 0 )
56.             return *((*UserIterator));
57.     }
58.     throw Exception("not find this user");
59. }
60.
61. int main(int argc, char* argv[])
62. {
63.     try{
64.         InitWinSock();
65.
66.         SOCKET PrimaryUDP;
67.         PrimaryUDP = mksock(SOCK_DGRAM);
68.
69.         sockaddr_in local;
70.         local.sin_family=AF_INET;
71.         local.sin_port= htons(SERVER_PORT);
```

```

72.     local.sin_addr.s_addr = htonl(INADDR_ANY);
73.     int nResult=bind(PrimaryUDP, (sockaddr*)&local, sizeof(sockaddr));
74.     if(nResult==SOCKET_ERROR)
75.         throw Exception("bind error");
76.
77.     sockaddr_in sender;
78.     stMessage recvbuf;
79.     memset(&recvbuf, 0, sizeof(stMessage));
80.
81.     // 开始主循环.
82.     // 主循环负责下面几件事情:
83.     // 一:读取客户端登陆和登出消息, 记录客户列表
84.     // 二:转发客户p2p请求
85.     for(;;)
86.     {
87.         int dwSender = sizeof(sender);
88.         int ret = recvfrom(PrimaryUDP, (char*)&recvbuf, sizeof(stMessage), 0, (sockaddr*)&sender, &dwSender);
89.         if(ret <= 0)
90.         {
91.             printf("recv error");
92.             continue;
93.         }
94.         else
95.         {
96.             int messageType = recvbuf.iMessageType;
97.             switch(messageType) {
98. case LOGIN:
99.             {
100.                 // 将这个用户的信息记录到用户列表中
101.                 printf("has a user login : %s\n", recvbuf.message.loginmember.userName);
102.                 stUserListNode *currentuser = new stUserListNode();
103.                 strcpy(currentuser->userName, recvbuf.message.loginmember.userName);
104.                 currentuser->ip = ntohl(sender.sin_addr.S_un.S_addr);
105.                 currentuser->port = ntohs(sender.sin_port);
106.
107.                 ClientList.push_back(currentuser);
108.
109.                 // 发送已经登陆的客户信息
110.                 int nodecount = (int)ClientList.size();
111.                 sendto(PrimaryUDP, (const char*)&nodecount, sizeof(int), 0, (const sockaddr*)&sender, sizeof(sender));
112.                 for(UserList::iterator UserIterator=ClientList.begin();
113.                    UserIterator!=ClientList.end();
114.                    ++UserIterator)
115.                 {
116.                     sendto(PrimaryUDP, (const char*)
117. (*UserIterator), sizeof(stUserListNode), 0, (const sockaddr*)&sender, sizeof(sender));
118.                 }
119.                 break;
120.             }
121. case LOGOUT:
122.             {
123.                 // 将此客户信息删除
124.                 printf("has a user logout : %s\n", recvbuf.message.logoutmember.userName);
125.                 UserList::iterator removeiterator = NULL;
126.                 for(UserList::iterator UserIterator=ClientList.begin();
127.                    UserIterator!=ClientList.end();
128.                    ++UserIterator)
129.                 {
130.                     if( strcmp( ((*UserIterator)->userName), recvbuf.message.logoutmember.userName) == 0 )
131.                     {
132.                         removeiterator = UserIterator;
133.                         break;
134.                     }
135.                 }
136.                 if(removeiterator != NULL)
137.                     ClientList.remove(*removeiterator);
138.                 break;
139.             }
140. case P2PTRANS:
141.             {
142.                 // 某个客户希望服务端向另外一个客户发送一个打洞消息
143.                 printf("%s wants to p2p %s\n", inet_ntoa(sender.sin_addr), recvbuf.message.translatemessage.userName);
144.                 stUserListNode node = GetUser(recvbuf.message.translatemessage.userName);
145.                 sockaddr_in remote;
146.                 remote.sin_family=AF_INET;
147.                 remote.sin_port= htons(node.port);
148.                 remote.sin_addr.s_addr = htonl(node.ip);
149.
150.                 in_addr tmp;
151.                 tmp.S_un.S_addr = htonl(node.ip);
152.                 printf("the address is %s, and port is %d\n", inet_ntoa(tmp), node.port);
153.
154.                 stP2PMessage transMessage;
155.                 transMessage.iMessageType = P2PSOMEONEWANTTOCALLYOU;
156.                 transMessage.iStringLen = ntohl(sender.sin_addr.S_un.S_addr);
157.                 transMessage.Port = ntohs(sender.sin_port);

```

```

158.
159.         sendto(PrimaryUDP,
(const char*)&transMessage, sizeof(transMessage), 0, (const sockaddr *)&remote, sizeof(remote));
160.
161.         break;
162.     }
163.
164.     case GETALLUSER:
165.     {
166.         int command = GETALLUSER;
167.         sendto(PrimaryUDP, (const char*)&command, sizeof(int), 0, (const sockaddr*)&sender, sizeof(sender));
168.
169.         int nodecount = (int)ClientList.size();
170.         sendto(PrimaryUDP, (const char*)&nodecount, sizeof(int), 0, (const sockaddr*)&sender, sizeof(sender));
171.
172.         for(UserList::iterator UserIterator=ClientList.begin();
173.             UserIterator!=ClientList.end();
174.             ++UserIterator)
175.         {
176.             sendto(PrimaryUDP, (const char*)
(*UserIterator), sizeof(stUserListNode), 0, (const sockaddr*)&sender, sizeof(sender));
177.         }
178.         break;
179.     }
180. }
181. }
182. }
183.
184. }
185. catch(Exception &e)
186. {
187.     printf(e.GetMessage());
188.     return 1;
189. }
190.
191.     return 0;
192. }

```

Cpp代码

```

1.  /* P2P 程序客户端
2.  *
3.  * 文件名: P2PClient.c
4.  *
5.  * 日期: 2004-5-21
6.  *
7.  * 作者: shootingstars(zhouhuis22@sina.com)
8.  *
9.  */
10.
11. #pragma comment(lib, "ws2_32.lib")
12.
13. #include "windows.h"
14. #include "..\proto.h"
15. #include "..\Exception.h"
16. #include <iostream>
17. using namespace std;
18.
19. UserList ClientList;
20.
21. #define COMMANDMAXC 256
22. #define MAXRETRY 5
23.
24. SOCKET PrimaryUDP;
25. char UserName[10];
26. char ServerIP[20];
27.
28. bool RecvedACK;
29.
30. void InitWinSock()
31. {
32.     WSADATA wsaData;
33.
34.     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
35.     {
36.         printf("Windows sockets 2.2 startup");
37.         throw Exception("");
38.     }
39.     else{
40.         printf("Using %s (Status: %s)\n",
41.             wsaData.szDescription, wsaData.szSystemStatus);
42.         printf("with API versions %d.%d to %d.%d\n\n",
43.             LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion),

```

```

44.         LOBYTE(wsaData.wHighVersion), HIBYTE(wsaData.wHighVersion));
45.     }
46. }
47.
48. SOCKET mksock(int type)
49. {
50.     SOCKET sock = socket(AF_INET, type, 0);
51.     if (sock < 0)
52.     {
53.         printf("create socket error");
54.         throw Exception("");
55.     }
56.     return sock;
57. }
58.
59. stUserListNode GetUser(char *username)
60. {
61.     for(UserList::iterator UserIterator=ClientList.begin();
62.         UserIterator!=ClientList.end();
63.         ++UserIterator)
64.     {
65.         if( strcmp( ((*UserIterator)->userName), username) == 0 )
66.             return *((*UserIterator));
67.     }
68.     throw Exception("not find this user");
69. }
70.
71. void BindSock(SOCKET sock)
72. {
73.     sockaddr_in sin;
74.     sin.sin_addr.S_un.S_addr = INADDR_ANY;
75.     sin.sin_family = AF_INET;
76.     sin.sin_port = 0;
77.
78.     if (bind(sock, (struct sockaddr*)&sin, sizeof(sin)) < 0)
79.         throw Exception("bind error");
80. }
81.
82. void ConnectToServer(SOCKET sock, char *username, char *serverip)
83. {
84.     sockaddr_in remote;
85.     remote.sin_addr.S_un.S_addr = inet_addr(serverip);
86.     remote.sin_family = AF_INET;
87.     remote.sin_port = htons(SERVER_PORT);
88.
89.     stMessage sendbuf;
90.     sendbuf.iMessageType = LOGIN;
91.     strncpy(sendbuf.message.loginmember.userName, username, 10);
92.
93.     sendto(sock, (const char*)&sendbuf, sizeof(sendbuf), 0, (const sockaddr*)&remote, sizeof(remote));
94.
95.     int usercount;
96.     int fromlen = sizeof(remote);
97.     int iread = recvfrom(sock, (char *)&usercount, sizeof(int), 0, (sockaddr *)&remote, &fromlen);
98.     if(iread<=0)
99.     {
100.         throw Exception("Login error\n");
101.     }
102.
103.     // 登录到服务端后, 接收服务端发来的已经登录的用户的信息
104.     cout<<"Have "<<usercount<<" users logged server:"<<endl;
105.     for(int i = 0;i<usercount;i++)
106.     {
107.         stUserListNode *node = new stUserListNode;
108.         recvfrom(sock, (char*)node, sizeof(stUserListNode), 0, (sockaddr *)&remote, &fromlen);
109.         ClientList.push_back(node);
110.         cout<<"Username:"<<node->userName<<endl;
111.         in_addr tmp;
112.         tmp.S_un.S_addr = htonl(node->ip);
113.         cout<<"UserIP:"<<inet_ntoa(tmp)<<endl;
114.         cout<<"UserPort:"<<node->port<<endl;
115.         cout<<" "<<endl;
116.     }
117. }
118.
119. void OutputUsage()
120. {
121.     cout<<"You can input you command:\n"
122.         <<"Command Type:\\"send\\",\\"exit\\",\\"getu\\"\\n"
123.         <<"Example : send Username Message\\n"
124.         <<"          exit\\n"
125.         <<"          getu\\n"
126.         <<endl;
127. }
128.
129. /* 这是主要的函数: 发送一个消息给某个用户(C)
130. *流程: 直接向某个用户的外网IP发送消息, 如果此前没有联系过

```

```

131. *      那么此消息将无法发送, 发送端等待超时。
132. *      超时后, 发送端将发送一个请求信息到服务端,
133. *      要求服务端发送给客户C一个请求, 请求C给本机发送打洞消息
134. *      以上流程将重复MAXRETRY次
135. */
136. bool SendMessageTo(char *UserName, char *Message)
137. {
138.     char realmessage[256];
139.     unsigned int UserIP;
140.     unsigned short UserPort;
141.     bool FindUser = false;
142.     for(UserList::iterator UserIterator=ClientList.begin();
143.         UserIterator!=ClientList.end();
144.         ++UserIterator)
145.     {
146.         if( strcmp( ((*UserIterator)->userName), UserName) == 0 )
147.         {
148.             UserIP = (*UserIterator)->ip;
149.             UserPort = (*UserIterator)->port;
150.             FindUser = true;
151.         }
152.     }
153.
154.     if(!FindUser)
155.         return false;
156.
157.     strcpy(realmessage, Message);
158.     for(int i=0;i<MAXRETRY;i++)
159.     {
160.         RecvedACK = false;
161.
162.         sockaddr_in remote;
163.         remote.sin_addr.S_un.S_addr = htonl(UserIP);
164.         remote.sin_family = AF_INET;
165.         remote.sin_port = htons(UserPort);
166.         stP2PMessage MessageHead;
167.         MessageHead.iMessageType = P2PMESSAGE;
168.         MessageHead.iStringLen = (int)strlen(realmessage)+1;
169.         int isend = sendto(PrimaryUDP, (const char *)&MessageHead, sizeof(MessageHead), 0, (const sockaddr*)&remote, sizeof(remote));
170.         isend = sendto(PrimaryUDP, (const char *)&realmessage, MessageHead.iStringLen, 0, (const sockaddr*)&remote, sizeof(remote));
171.
172.         // 等待接收线程将此标记修改
173.         for(int j=0;j<10;j++)
174.         {
175.             if(RecvedACK)
176.                 return true;
177.             else
178.                 Sleep(300);
179.         }
180.
181.         // 没有接收到目标主机的回应, 认为目标主机的端口映射没有
182.         // 打开, 那么发送请求信息给服务器, 要服务器告诉目标主机
183.         // 打开映射端口 (UDP打洞)
184.         sockaddr_in server;
185.         server.sin_addr.S_un.S_addr = inet_addr(ServerIP);
186.         server.sin_family = AF_INET;
187.         server.sin_port = htons(SERVER_PORT);
188.
189.         stMessage transMessage;
190.         transMessage.iMessageType = P2PTRANS;
191.         strcpy(transMessage.message.translatemessage.userName, UserName);
192.
193.         sendto(PrimaryUDP, (const char *)&transMessage, sizeof(transMessage), 0, (const sockaddr*)&server, sizeof(server));
194.         Sleep(100); // 等待对方先发送信息。
195.     }
196.     return false;
197. }
198.
199. // 解析命令, 暂时只有exit和send命令
200. // 新增getu命令, 获取当前服务器的所有用户
201. void ParseCommand(char * CommandLine)
202. {
203.     if(strlen(CommandLine)<4)
204.         return;
205.     char Command[10];
206.     strncpy(Command, CommandLine, 4);
207.     Command[4]='\0';
208.
209.     if(strcmp(Command,"exit")==0)
210.     {
211.         stMessage sendbuf;
212.         sendbuf.iMessageType = LOGOUT;
213.         strncpy(sendbuf.message.logoutmember.userName, UserName, 10);
214.         sockaddr_in server;
215.         server.sin_addr.S_un.S_addr = inet_addr(ServerIP);
216.         server.sin_family = AF_INET;
217.         server.sin_port = htons(SERVER_PORT);

```

```

218.
219.     sendto(PrimaryUDP, (const char*)&sendbuf, sizeof(sendbuf), 0, (const sockaddr *)&server, sizeof(server));
220.     shutdown(PrimaryUDP, 2);
221.     closesocket(PrimaryUDP);
222.     exit(0);
223. }
224. else if(strcmp(Command, "send")==0)
225. {
226.     char sendname[20];
227.     char message[COMMANDMAXC];
228.     int i;
229.     for(i=5;;i++)
230.     {
231.         if(CommandLine[i]!=' ')
232.             sendname[i-5]=CommandLine[i];
233.         else
234.         {
235.             sendname[i-5]='\0';
236.             break;
237.         }
238.     }
239.     strcpy(message, &(CommandLine[i+1]));
240.     if(SendMessageTo(sendname, message))
241.         printf("Send OK!\n");
242.     else
243.         printf("Send Failure!\n");
244. }
245. else if(strcmp(Command, "getu")==0)
246. {
247.     int command = GETALLUSER;
248.     sockaddr_in server;
249.     server.sin_addr.S_un.S_addr = inet_addr(ServerIP);
250.     server.sin_family = AF_INET;
251.     server.sin_port = htons(SERVER_PORT);
252.
253.     sendto(PrimaryUDP, (const char*)&command, sizeof(command), 0, (const sockaddr *)&server, sizeof(server));
254. }
255. }
256.
257. // 接受消息线程
258. DWORD WINAPI RecvThreadProc(LPVOID lpParameter)
259. {
260.     sockaddr_in remote;
261.     int sinlen = sizeof(remote);
262.     stP2PMessage recvbuf;
263.     for(;;)
264.     {
265.         int iread = recvfrom(PrimaryUDP, (char *)&recvbuf, sizeof(recvbuf), 0, (sockaddr *)&remote, &sinlen);
266.         if(iread<=0)
267.         {
268.             printf("recv error\n");
269.             continue;
270.         }
271.         switch(recvbuf.iMessageType)
272.         {
273.             case P2PMESSAGE:
274.             {
275.                 // 接收到P2P的消息
276.                 char *comemessage= new char[recvbuf.iStringLen];
277.                 int iread1 = recvfrom(PrimaryUDP, comemessage, 256, 0, (sockaddr *)&remote, &sinlen);
278.                 comemessage[iread1-1] = '\0';
279.                 if(iread1<=0)
280.                     throw Exception("Recv Message Error\n");
281.                 else
282.                 {
283.                     printf("Recv a Message:%s\n", comemessage);
284.
285.                     stP2PMessage sendbuf;
286.                     sendbuf.iMessageType = P2PMESSAGEACK;
287.                     sendto(PrimaryUDP, (const char*)&sendbuf, sizeof(sendbuf), 0, (const sockaddr *)&remote, sizeof(remote));
288.                 }
289.
290.                 delete []comemessage;
291.                 break;
292.             }
293.             case P2PSOMEONEWANTTOCALLYOU:
294.             {
295.                 // 接收到打洞命令, 向指定的IP地址打洞
296.                 printf("Recv p2someonewanttocallyou data\n");
297.                 sockaddr_in remote;
298.                 remote.sin_addr.S_un.S_addr = htonl(recvbuf.iStringLen);
299.                 remote.sin_family = AF_INET;
300.                 remote.sin_port = htons(recvbuf.Port);
301.
302.                 // UDP hole punching
303.                 stP2PMessage message;
304.

```



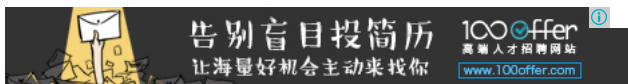
```



305.         message.iMessageType = P2PTRASH;
306.         sendto(PrimaryUDP, (const char *)&message, sizeof(message), 0, (const sockaddr*)&remote, sizeof(remote));
307.
308.         break;
309.     }
310. case P2PMESSAGEACK:
311.     {
312.         // 发送消息的应答
313.         RecvedACK = true;
314.         break;
315.     }
316. case P2PTRASH:
317.     {
318.         // 对方发送的打洞消息, 忽略掉。
319.         //do nothing ...
320.         printf("Recv p2ptrash data\n");
321.         break;
322.     }
323. case GETALLUSER:
324.     {
325.         int usercount;
326.         int fromlen = sizeof(remote);
327.         int iread = recvfrom(PrimaryUDP, (char *)&usercount, sizeof(int), 0, (sockaddr *)&remote, &fromlen);
328.         if(iread<=0)
329.         {
330.             throw Exception("Login error\n");
331.         }
332.
333.         ClientList.clear();
334.
335.         cout<<"Have "<<usercount<<" users logined server:"<<endl;
336.         for(int i = 0;i<usercount;i++)
337.         {
338.             stUserListNode *node = new stUserListNode;
339.             recvfrom(PrimaryUDP, (char*)node, sizeof(stUserListNode), 0, (sockaddr *)&remote, &fromlen);
340.             ClientList.push_back(node);
341.             cout<<"Username:"<<node->userName<<endl;
342.             in_addr tmp;
343.             tmp.S_un.S_addr = htonl(node->ip);
344.             cout<<"UserIP:"<<inet_ntoa(tmp)<<endl;
345.             cout<<"UserPort:"<<node->port<<endl;
346.             cout<<" "<<endl;
347.         }
348.         break;
349.     }
350. }
351. }
352. }
353.
354.
355. int main(int argc, char* argv[])
356. {
357.     try
358.     {
359.         InitWinSock();
360.
361.         PrimaryUDP = mksock(SOCK_DGRAM);
362.         BindSock(PrimaryUDP);
363.
364.         cout<<"Please input server ip:";
365.         cin>>ServerIP;
366.
367.         cout<<"Please input your name:";
368.         cin>>UserName;
369.
370.         ConnectToServer(PrimaryUDP, UserName, ServerIP);
371.
372.         HANDLE threadhandle = CreateThread(NULL, 0, RecvThreadProc, NULL, NULL, NULL);
373.         CloseHandle(threadhandle);
374.         OutputUsage();
375.
376.         for(;;)
377.         {
378.             char Command[COMMANDMAXC];
379.             gets(Command);
380.             ParseCommand(Command);
381.         }
382.     }
383.     catch(Exception &e)
384.     {
385.         printf(e.GetMessage());
386.         return 1;
387.     }
388.     return 0;
389. }

```

```
1.  /* 异常类
2.  *
3.  * 文件名: Exception.h
4.  *
5.  * 日期: 2004.5.5
6.  *
7.  * 作者: shootingstars(zhouhuis22@sina.com)
8.  */
9.
10. #ifndef __HZH_Exception__
11. #define __HZH_Exception__
12.
13. #define EXCEPTION_MESSAGE_MAXLEN 256
14. #include "string.h"
15.
16. class Exception
17. {
18. private:
19.     char m_ExceptionMessage[EXCEPTION_MESSAGE_MAXLEN];
20. public:
21.     Exception(char *msg)
22.     {
23.         strncpy(m_ExceptionMessage, msg, EXCEPTION_MESSAGE_MAXLEN);
24.     }
25.
26.     char *GetMessage()
27.     {
28.         return m_ExceptionMessage;
29.     }
30. };
31.
32. #endif
33.
34. /* P2P 程序传输协议
35. *
36. * 日期: 2004-5-21
37. *
38. * 作者: shootingstars(zhouhuis22@sina.com)
39. *
40. */
41.
42. #pragma once
43. #include <list>
44.
45. // 定义iMessageType的值
46. #define LOGIN 1
47. #define LOGOUT 2
48. #define P2PTRANS 3
49. #define GETALLUSER 4
50.
51. // 服务器端口
52. #define SERVER_PORT 2280
53.
54. // Client登录时向服务器发送的消息
55. struct stLoginMessage
56. {
57.     char userName[10];
58.     char password[10];
59. };
60.
61. // Client注销时发送的消息
62. struct stLogoutMessage
63. {
64.     char userName[10];
65. };
66.
67. // Client向服务器请求另外一个Client(userName)向自己方向发送UDP打洞消息
68. struct stP2PTranslate
69. {
70.     char userName[10];
71. };
72.
73. // Client向服务器发送的消息格式
74. struct stMessage
75. {
76.     int iMessageType;
77.     union _message
78.     {
79.         stLoginMessage loginmember;
80.         stLogoutMessage logoutmember;
81.         stP2PTranslate translatemessage;
82.     }message;
83. };
84.
85. // 客户节点信息
86. struct stUserListNode
87. {
```

```
88.     char userName[10];
89.     unsigned int ip;
90.     unsigned short port;
91. };
92.
93. // Server向Client发送的消息
94. struct stServerToClient
95. {
96.     int iMessageType;
97.     union _message
98.     {
99.         stUserListNode user;
100.     }message;
101. };
102. };
103.
104. //=====
105. // 下面的协议用于客户端之间的通信
106. //=====
107. #define P2PMESSAGE 100 // 发送消息
108. #define P2PMESSAGEACK 101 // 收到消息的应答
109. #define P2PSOMEONEWANTTOCALLYOU 102 // 服务器向客户端发送的消息
110. // 希望此客户端发送一个UDP打洞包
111. #define P2PTRASH 103 // 客户端发送的打洞包, 接收端应该忽略此消息
112.
113. // 客户端之间发送消息格式
114. struct stP2PMessage
115. {
116.     int iMessageType;
117.     int iStringLen; // or IP address
118.     unsigned short Port;
119. };
120.
121. using namespace std;
122. typedef list<stUserListNode *> UserList;
```



分享到:  

[P2P原理及UDP穿透简单说明](#) | [Napster](#)

- 2009-08-11 14:01
- 浏览 6530
- [评论\(1\)](#)
- [相关推荐](#)

评论

1 楼 [gmwlu](#) 2013-11-19
有问题, 我测试了很多无法穿透



发表评论



[您还没有登录, 请您登录后再发表评论](#)



andylin02

- 浏览: 1047191 次
- 性别: 
- 来自: 北京
-  我现在离线

最近访客 [更多访客>>](#)



[u010815305](#)



[ho012tpiu](#)

[小帅1127](#)[hoorav520](#)

文章分类

- [全部博客 \(657\)](#)
- [ACE \(35\)](#)
- [BAT \(9\)](#)
- [C/C++ \(116\)](#)
- [fast-cgi \(14\)](#)
- [COM \(27\)](#)
- [python \(59\)](#)
- [CGI \(4\)](#)
- [C# \(2\)](#)
- [VC \(84\)](#)
- [DataBase \(29\)](#)
- [Linux \(96\)](#)
- [P2P \(6\)](#)
- [PHP \(15\)](#)
- [Web \(6\)](#)
- [Memcached \(7\)](#)
- [IME输入法 \(11\)](#)
- [设计模式 \(2\)](#)
- [搜索引擎 \(1\)](#)
- [个人情感 \(4\)](#)
- [笔试/面试 \(3\)](#)
- [一亩三分地 \(33\)](#)
- [历史 \(2\)](#)
- [地理 \(1\)](#)
- [人物 \(3\)](#)
- [经济 \(0\)](#)
- [不仅仅是笑哦 \(43\)](#)
- [小故事大道理 \(2\)](#)
- [http://www.bidsmyvsik120.com/ \(0\)](#)
- [http://www.bidsmyv120.com/ \(0\)](#)
- [它山之石可以攻玉 \(15\)](#)
- [大学生你关注些什么 \(28\)](#)
- [数据恢复 \(1\)](#)

社区版块

- [我的资讯 \(0\)](#)
- [我的论坛 \(0\)](#)
- [我的问答 \(0\)](#)

存档分类

- [2015-05 \(3\)](#)
- [2014-04 \(1\)](#)
- [2012-08 \(4\)](#)
- [更多存档...](#)

最新评论

- [effort fan](#): 好文章! 学习了, 谢谢分享!
[com技术简介](#)
- [vcell](#): 有错误os.walk(strPath)返回的已经是全部的文件和 ...
[通过python获取目录的大小](#)
- [feifeigd](#): feifeigd 写道注意: 文章中的CPP示例第二行 #inc ...
[ATL入门: 利用ATL编写简单的COM组件](#)
- [feifeigd](#): 注意: 文章中的CPP示例第二行 #include " ...
[ATL入门: 利用ATL编写简单的COM组件](#)
- [stecdeng](#): 有验证么 百度不是获取到的这个IP
[gethostbyname\(\) -- 用域名或主机名获取IP地址](#)

声明: ITeye文章版权属于作者, 受法律保护。没有作者书面许可不得转载。若作者同意转载, 必须以超链接形式标明文章原始出处和作者。
© 2003-2016 ITeye.com. All rights reserved. [京ICP证110151号 京公网安备110105010620]