# **[P2P之UDP穿透NAT的原理与实现](https://www.cnblogs.com/monjeo/p/9394825.html)**

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能够地址重用。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数据报的IP地址，还会改变IP数据报的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：

                                Server S1                           
                         18.181.0.31:1235                            
                                      |  
          ^  Session 1 (A-S1)  ^      |    
          |  18.181.0.31:1235  |      |     
          v 155.99.25.11:62000 v      |      
                                      |  
                                     NAT  
                                 155.99.25.11  
                                      |  
          ^  Session 1 (A-S1)  ^      |    
          |  18.181.0.31:1235  |      |    
          v   10.0.0.1:1234    v      |    
                                      |  
                                   Client A  
                                10.0.0.1:1234

    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：这里是说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建立以了一个连接。

    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：

    Server S1                                     Server S2  
 18.181.0.31:1235                              138.76.29.7:1235  
        |                                             |  
        |                                             |  
        +----------------------+----------------------+  
                               |  
   ^  Session 1 (A-S1)  ^      |      ^  Session 2 (A-S2)  ^  
   |  18.181.0.31:1235  |      |      |  138.76.29.7:1235  |  
   v 155.99.25.11:62000 v      |      v 155.99.25.11:62000 v  
                               |  
                            Cone NAT  
                          155.99.25.11  
                               |  
   ^  Session 1 (A-S1)  ^      |      ^  Session 2 (A-S2)  ^  
   |  18.181.0.31:1235  |      |      |  138.76.29.7:1235  |  
   v   10.0.0.1:1234    v      |      v   10.0.0.1:1234    v  
                               |  
                            Client A  
                         10.0.0.1:1234

    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）。另外一种是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：

                       Server S （219.237.60.1）  
                          |  
                          |  
   +----------------------+----------------------+  
   |                                             |  
 NAT A (外网IP:202.187.45.3)                 NAT B (外网IP:187.34.1.56)  
   |   (内网IP:192.168.0.1)                      | (内网IP:192.168.0.1)  
   |                                             |  
Client A  (192.168.0.20:4000)             Client B (192.168.0.10:40000)

    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

    天津雍博科技有限责任公司

    欢迎大家访问www.yongbosoft.com ，崇尚技术，为您提供完美产品，优质服务！