[P2P的原理和常见的实现方式(为libjingle开路)](http://www.cppblog.com/peakflys/archive/2013/01/25/197562.html)

   为了项目的后期IM应用，最近在研究libjingle，中间看了也收集了很多资料，感慨网上很多资料要么太过于纠结协议（如STUN、ICE等）实现细节，要么中间有很多纰漏。最后去伪存真，归纳总结了一下，希望对以后的同行有些许帮助。  
   如果有什么需要讨论或者指正的，欢迎留言或者邮件peakflys@gmail.com                              
P2P实现的原理

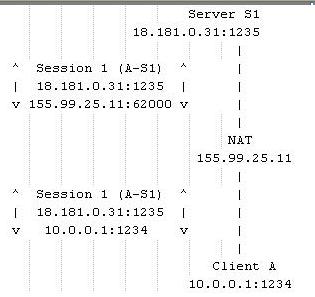
首先先介绍一些基本概念：

   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从历史发展上分为两大类，基本的NAT和NAPT(Network Address/Port Translator)。

   最先提出的是基本的NAT(peakflys注：刚开始其实只是路由器上的一个功能模块)，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（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

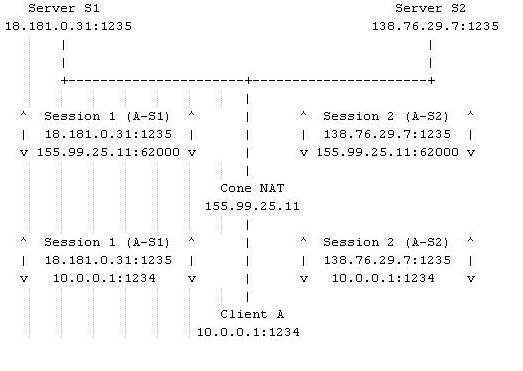
另外一种NAT叫做NAPT，从名称上我们也可以看得出，NAPT不但会改变经过这个NAT设备的IP数据报的IP地址，还会改变IP数据报的TCP/UDP端口。基本NAT的设备可能我们见的不多（基本已经淘汰了），NAPT才是我们真正需要关注的。看下图：  
  
有一个私有网络10.\*.\*.\*，Client A是其中的一台计算机，这个网络的网关（一个NAT设备）的外网IP是155.99.25.11(应该还有一个内网的IP地址，比如10.0.0.10)。如果Client A中的某个进程（这个进程创建了一个UDP Socket,这个Socket绑定1234端口）想访问外网主机18.181.0.31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。

上面的是一些基础知识，下面的才是关键的部分了。

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。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）

peakflys注：Cone NAT具体又分为3种：

(1)全圆锥( Full Cone) : NAT把所有来自相同内部IP地址和端口的请求映射到相同的外部IP地址和端口。任何一个外部主机均可通过该映射发送IP包到该内部主机。

(2)限制性圆锥(Restricted Cone) : NAT把所有来自相同内部IP地址和端口的请求映射到相同的外部IP地址和端口。但是,只有当内部主机先给IP地址为X的外部主机发送IP包,该外部主机才能向该内部主机发送IP包。

(3)端口限制性圆锥( Port Restricted Cone) :端口限制性圆锥与限制性圆锥类似,只是多了端口号的限制,即只有内部主机先向IP地址为X,端口号为P的外部主机发送1个IP包,该外部主机才能够把源端口号为P的IP包发送给该内部主机。

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）。

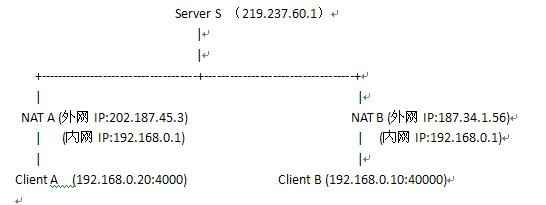
那么我们如果想从外部发送一个数据报给内网的计算机有什么办法呢？首先，我们必须在内网的NAT上打上一个“洞”（也就是前面我们说的在NAT上建立一个Session），这个洞不能由外部来打，只能由内网内的主机来打。而且这个洞是有方向的，比如从内部某台主机（比如：192.168.0.10）向外部的某个IP(比如：219.237.60.1)发送一个UDP包，那么就在这个内网的NAT设备上打了一个方向为219.237.60.1的“洞”，（这就是称为UDP Hole Punching的技术）以后219.237.60.1就可以通过这个洞与内网的192.168.0.10联系了。（但是其他的IP不能利用这个洞）。

P2P的常用实现

一、普通的直连式P2P实现

通过上面的理论，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？自然是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。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，这种情况下一般放弃P2P  ---peakflys）。

二、STUN方式的P2P实现

STUN是RFC3489规定的一种NAT穿透方式，它采用辅助的方法探测NAT的IP和端口。毫无疑问的，它对穿越早期的NAT起了巨大的作用，并且还将继续在NAT穿透中占有一席之地。

STUN的探测过程需要有一个公网IP的STUN server，在NAT后面的UAC必须和此server配合，互相之间发送若干个UDP数据包。UDP包中包含有UAC需要了解的信息，比如NAT外网IP，PORT等等。UAC通过是否得到这个UDP包和包中的数据判断自己的NAT类型。

假设有如下UAC（B），NAT（A），SERVER（C），UAC的IP为IPB，NAT的IP为 IPA ，SERVER的 IP为IPC1 、IPC2。请注意，服务器C有两个IP，后面你会理解为什么需要两个IP。

(1)NAT的探测过程

STEP1：B向C的IPC1的port1端口发送一个UDP包。C收到这个包后，会把它收到包的源IP和port写到UDP包中，然后把此包通过IP1C和port1发还给B。这个IP和port也就是NAT的外网IP和port，也就是说你在STEP1中就得到了NAT的外网IP。

熟悉NAT工作原理的应该都知道，C返回给B的这个UDP包B一定收到。如果在你的应用中，向一个STUN服务器发送数据包后，你没有收到STUN的任何回应包，那只有两种可能：1、STUN服务器不存在，或者你弄错了port。2、你的NAT设备拒绝一切UDP包从外部向内部通过，如果排除防火墙限制规则，那么这样的NAT设备如果存在，那肯定是坏了„„

当B收到此UDP后，把此UDP中的IP和自己的IP做比较，如果是一样的，就说明自己是在公网，下步NAT将去探测防火墙类型，就不多说了(下面有图)。如果不一样，说明有NAT的存在，系统进行STEP2的操作。

STEP2：B向C的IPC1发送一个UDP包，请求C通过另外一个IPC2和PORT（不同与SETP1的IP1）向B返回一个UDP数据包（现在知道为什么C要有两个IP了吧，为了检测cone NAT的类型）。

我们来分析一下，如果B收到了这个数据包，那说明什么？说明NAT来着不拒，不对数据包进行任何过滤，这也就是STUN标准中的full cone NAT。遗憾的是，full cone nat太少了，这也意味着你能收到这个数据包的可能性不大。如果没收到，那么系统进行STEP3的操作。

STEP3：B向C的IPC2的port2发送一个数据包，C收到数据包后，把它收到包的源IP和port写到UDP包中，然后通过自己的IPC2和port2把此包发还给B。

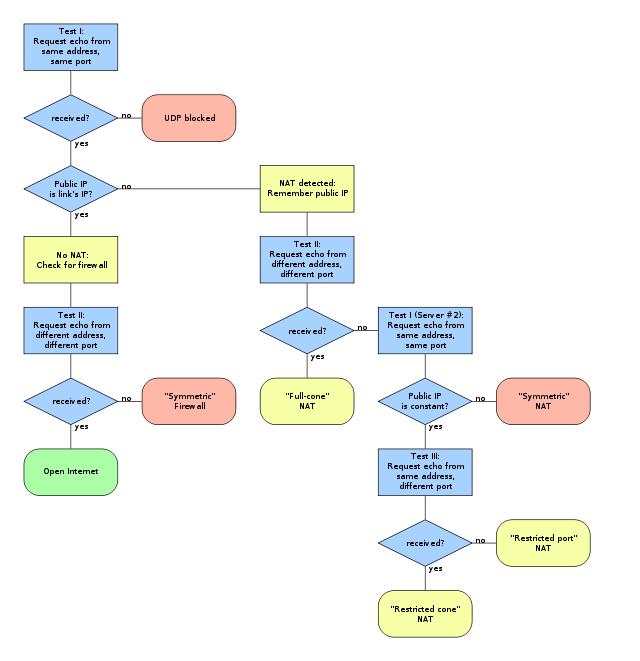
和step1一样，B肯定能收到这个回应UDP包。此包中的port是我们最关心的数据，下面我们来分析：

如果这个port和step1中的port一样，那么可以肯定这个NAT是个CONE NAT，否则是对称NAT。道理很简单：根据对称NAT的规则，当目的地址的IP和port有任何一个改变，那么NAT都会重新分配一个port使用，而在step3中，和step1对应，我们改变了IP和port。因此，如果是对称NAT,那这两个port肯定是不同的。

如果在你的应用中，到此步的时候PORT是不同的，那就只能放弃P2P了，原因同上面实现中的一样。如果不同，那么只剩下了restrict cone 和port restrict cone。系统用step4探测是是那一种。

STEP4：B向C的IP2的一个端口PD发送一个数据请求包，要求C用IP2和不同于PD的port返回一个数据包给B。

我们来分析结果：如果B收到了，那也就意味着只要IP相同，即使port不同，NAT也允许UDP包通过。显然这是restrict cone NAT。如果没收到，没别的好说，port restrict NAT.

协议实现的算法运行图如下：  


一旦路经到达红色节点时，UDP的沟通是没有可能性的(peakflys注：准备来说除了包被防火墙blocked之外，其他情况也是有可能建立P2P的，只是代价太大，一般放弃)。一旦通过黄色或是绿色的节点，就有连接的可能。

最终通过STUN服务器得到自己的NAT类型和公网IP、Port，以后建立P2P时就非常容易了

peakflys注：Libjingle正是通过ICE&STUN方式，建立的P2P连接。关于libjingle的介绍，待续……  
  
参考资料：  
1、维基百科之STUN  
2、http://midcom-p2p.sourceforge.net/draft-ford-midcom-p2p-01.txt（shootingstars）

posted on 2013-01-25 15:43 [peakflys](http://www.cppblog.com/peakflys/) 阅读(26755) [评论(12)](http://www.cppblog.com/peakflys/archive/2013/01/25/197562.html#Post)  [编辑](http://www.cppblog.com/peakflys/admin/EditPosts.aspx?postid=197562) [收藏](http://www.cppblog.com/peakflys/AddToFavorite.aspx?id=197562) [引用](http://www.cppblog.com/peakflys/services/trackbacks/197562.aspx) 所属分类: [P2P](http://www.cppblog.com/peakflys/category/20361.html)