QQ是一个基于TCP/UDP协议的通讯软件

发送消息的时候是UDP打洞,登陆的时候使用HTTP~因为登陆服务器其实就是一个HTTP服务器,只不过不是常用的那些,那个服务器是腾讯自行开发的！！！

一、登录

QQ客户端在局域网内，当你打开QQ登录到QQ服务器时，通过外网，你的客户端与QQ服务器建立了一个长连接。你可以用netstat -bn  看到此连接的状态是 establish

[QQ通信原理及QQ是怎么穿透内网进行通信的？](http://photo.blog.sina.com.cn/showpic.html#blogid=70334c580100nt9u&url=http://static11.photo.sina.com.cn/orignal/70334c58x9782a3dfb5ea)  
此时，在QQ服务器那面看到的连接的IP是**你们局域网对外的IP**。举个例子：         
QQ服务器      IP：121.115.11.81     服务端口：80         
你的机器在局域网中内部IP：   10.19.9.89         
你局域网出口InternetIP：  61.183.172.149         
你的客户端的请求将通过外网出去，如果防火墙没有禁止访问Internet上80端口服务，那么你的QQ客户端可以正常登录。你看到的连接是 （netstat  -bn）         
10.19.9.89:55579           124.115.11.81：80    establish       

这是一个假象。通过QQ服务器看到的连接是：         
124.115.11.81：80      61.183.172.149：**31234**    establish         
        
这样，防火墙上的**31234**口对应的就是你机器的55579口。（由于你是发起方，这个数是变化的。动态的）         
当有信息给你时，QQ服务器只需要发给防火墙的55579口即可。（这里防火墙作了地址翻译）         
        
不管UDP还是TCP，最终登陆成功之后，QQ都会有一个TCP连接来保持在线状态。这个TCP连接的远程端口一般是80，采用UDP方式登陆的时候，端口是8000。因此，假如你所在的网络开放了80端口（80端口是最常用端口。。就是通常访问Web的端口，禁掉它的话，你的网络对你来说价值已经不大了），但没有屏蔽腾讯的服务器IP，恭喜你，你是可以登陆成功QQ的。

二、聊天消息通信  
    采用UDP协议，通过服务器中转方式。大家都知道，UDP 协议是不可靠协议，它只管发送，不管对方是否收到的，但它的传输很高效。但是，作为聊天软件，怎么可以采用这样的不可靠方式来传输消息呢？于是，腾讯采用了上层协议来保证可靠传输：如果客户端使用UDP协议发出消息后，服务器收到该包，需要使用UDP协议发回一个应答包。如此来保证消息可以无遗漏传输。之所以会发生在客户端明明看到“消息发送失败”但对方又收到了这个消息的情况，就是因为客户端发出的消息服务器已经收到并转发成功，但客户端由于网络原因没有收到服务器的应答包引起的。

**因为用户一般都是在局域网内,地址都为私有IP,腾讯服务器是如何将信息转发到用户的?**

 首先先介绍一些基本概念：  
    **NAT**(Network AddressTranslators)，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.\*.\*.\***，ClientA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UDPSocket,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SymmetricNAT，后一种叫做ConeNAT。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 HolePunching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：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  (192.168.0.20:60000)             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ServerS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，ClientB登录Server S，NAT B给此次Session分配的端口是40000，那么ServerS收到的B的地址是**187.34.1.56**:40000。s  
    此时，Client A与Client B都可以与ServerS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ServerS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ClientB就能收到了呢？答案是不行，因为如果这样发送信息，NATB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）。那该怎么办呢？   首先我们假设Server S是219.237.60.1：7000，当Clinet A（202.187.45.3：60000）向Server S（219.237.60.1：7000）发送数据包，Server S是可以正常接收到数据，因为它是属于外型开放的服务器端口。当Server S收到数据包后可以获知Clinet A（202.187.45.3：60000）对外通信的临时session信息（这个叫临时的端口，假设是60000会过期，具体时间不同，一般是每30S发送一个keep住连接以保证端口维持通信连接不断）Server S此时应将次信息保存起来。而同时，Client B (192.168.0.10:40000)也在时刻向  
Server S发送心跳包，Server S就向Client B (192.168.0.10:40000)发送一个通知，让Client B  (192.168.0.10:4000) 发送探测包（这个数据包最好发几个），Client B  (192.168.0.10:4000)在收到通知后在向Server S发送反馈包，说明以向自己以向Client A  (192.168.0.20:60000)发送了探测包，Server S在收到反馈之后再向Client A  (192.168.0.20:60000)转发反馈包，Client A  (192.168.0.20:60000)在收到数据包之后在向原本要求请求的Client B  (192.168.0.10:4000)发送数据包，此时连接已经打通，实现穿透。Client B  (192.168.0.10:4000)会将数据包转发给  
Client A  (192.168.0.20:60000)从而在转发给内网内网IP:192.168.0.1。

对于Symmetric NAPT的情况，网上有人说可以通过探测端口的方式，不过成功率并不高，我建议可用服务器进行中转。另外，最好在数据包发送前先检测是否进行的是同个NAT的情况，也就是内网发内网，如果是，直接发送即可，而无需通过外网再绕回来。