# P2P业务说明

## P2P简介

P2P即点对点通信，这个点的实际表现是一个可以从网络上定位到的唯一地址——一般包括公网地址（ipv4）+端口，内网地址+端口，[2级内网地址+端口]，[...]。点对点通讯，也称为对等联网，与传统的服务器客户端模式有着明显的区别，传统的服务器客户端模型如图2所示。P2P这一术语在不同的上下文环境里可能有不同的内涵，它可以指一种通信模式、一种逻辑网络模型、一种技术、甚至一种理念。在P2P网络中如图1所示，所有通信节点的地位都是对等的，每个节点都扮演着客户机和服务器双重角色，节点之间通过直接通信实现文件信息、处理器运算能力、存储空间等资源的共享。P2P网络具有非中心化、可扩展性、健壮性等特点，这使得P2P技术在信息共享、实时通信、协同工作、分布式计算、网络存储等领域都有广阔的应用。

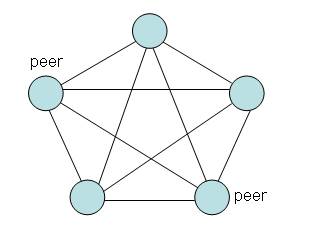


图1 P2P结构模型

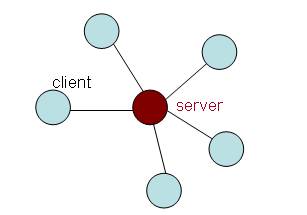


图2 CS模式

* 非中心化：网络中的资源和服务分散在所有结点上，信息的传输和服务的实现都直接在结点之间进行，可以无需中间环节和服务器的介入，避免了可能的瓶颈。P2P的非中心化基本特点，带来了其在可扩展性、健壮性等方面的优势。
* 可扩展性：在P2P网络中，随着用户的加入，不仅服务的需求增加了，系统整体的资源和服务能力也在同步地扩充，始终能比较容易地满足用户的需要。理论上其可扩展性几乎可以认为是无限的。例如：在传统的通过FTP的文件下载方式中，当下载用户增加之后，下载速度会变得越来越慢，然而P2P网络正好相反，加入的用户越多，P2P网络中提供的资源就越多，下载的速度反而越快。
* 健壮性：P2P架构天生具有耐攻击、高容错的优点。由于服务是分散在各个结点之间进行的，部分结点或网络遭到破坏对其它部分的影响很小。P2P网络一般在部分结点失效时能够自动调整整体拓扑，保持其它结点的连通性。P2P网络通常都是以自组织的方式建立起来的，并允许结点自由地加入和离开。
* 高性价比：性能优势是P2P被广泛关注的一个重要原因。随着硬件技术的发展，个人计算机的计算和存储能力以及网络带宽等性能依照摩尔定理高速增长。采用P2P架构可以有效地利用互联网中散布的大量普通结点，将计算任务或存储资料分布到所有结点上。利用其中闲置的计算能力或存储空间，达到高性能计算和海量存储的目的。目前，P2P在这方面的应用多在学术研究方面，一旦技术成熟，能够在工业领域推广，则可以为许多企业节省购买大型服务器的成本。
* 隐私保护: 在P2P网络中，由于信息的传输分散在各节点之间进行而无需经过某个集中环节，用户的隐私信息被窃听和泄漏的可能性大大缩小。此外，目前解决Internet隐私问题主要采用中继转发的技术方法，从而将通信的参与者隐藏在众多的网络实体之中。在传统的一些匿名通信系统中，实现这一机制依赖于某些中继服务器节点。而在P2P中，所有参与者都可以提供中继转发的功能，因而大大提高了匿名通讯的灵活性和可靠性，能够为用户提供更好的隐私保护。
* 负载均衡: P2P 网络环境下由于每个节点既是服务器又是客户机，减少了对传统C/S结构服务器计算能力、存储能力的要求，同时因为资源分布在多个节点，更好的实现了整个网络的负载均衡。

### 1.1 P2P网络拓扑结构

随着p2p网络与实用的不断结合，p2p现有网络拓扑结构有大致四种：

1.**中心化拓扑**（Centralized Topology）——有一个中心服务器负责记录共享信息(对点的定位索引信息)并应答对这些信息的查询。然后点与点之间再建立连接。

2.**分布式非结构化拓扑**（Decentralized Unstructured Topology）——采用了随机图的组织方式来形成松散的网络，没有中心服务器，采用洪泛式搜索(Flooding)和随机转发机制(TTL转发机制)，每个节点功能相同，兼作服务器和客户机。

3.**分布式结构化拓扑**（Decentralized Structured Topology，也称作DHT网络）——结构化是对网络解决的管理方式，是一种逻辑上可以结构化查询，而不是物理连接的变动，结构化是为了搜索算法的快捷，一般相当于折半查找。

4.**混合式(半分布式)拓扑**（Partially Decentralized Topology）——在分布式模式基础上，将用户节点按能力进行分类，使某些节点担任特殊的任务。用户节点：可以从索引节点处得到相临的搜索节点地址。搜索节点：处理搜索请求，要有128k以上的速度，从子节点中搜索文件列表。索引节点：速度快、内存大的节点，保存可以利用的搜索节点信息、搜集状态信息，并维护网络结构。

### 1.2 不同拓扑结构的优缺点

**中心化拓扑**最大的优点是维护简单，资源发现效率高。由于资源的发现依赖中心化的目录系统，发现算法灵活高效并能够实现复杂查询。最大的问题与传统客户机/服务器结构类似，容易造成单点故障，访问的“热点”现象和版权纠纷等相关问题，这是第一代P2P网络采用的结构模式，经典案例就是著名的MP3共享软件Napster.

Napster是最早出现的P2P系统之一，并在短期内迅速成长起来。它实质上并非是纯粹的P2P系统，而是通过一个中央索引服务器保存所有Napster用户上传的音乐文件索引和存放位置的信息。它的工作原理如图1所示。当某个用户需要某个音乐文件时，首先连接到Napster中央索引服务器，在服务器上进行检索，服务器返回存有该文件的用户信息，再由请求者直接连到文件的所有者传输文件。Napster首先实现了文件查询与文件传输的分离，有效地节省了中央服务器的带宽消耗，减少了系统的文件传输延时。

然而，这种对等网络模型存在以下这些问题：

* 中央索引服务器的瘫痪容易导致整个网络的崩溃，因此可靠性和安全性较低。
* 随着网络规模的扩大，对中央索引服务器进行维护和更新的费用将急剧增加，所需成本较高。
* 中央索引服务器的存在常引起版权问题上的纠纷，服务提供商容易被追究法律责任。

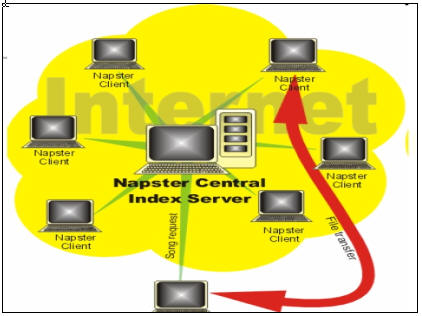


图1 Napster的拓扑结构

【目前，王波提供的即是中心化的这种结构】

**分布式非结构化拓扑**的是在重叠网络（Overlay Network）采用了随机图的组织方式，结点度数服从Power-law规律（幂次法则），从而能够较快发现目的结点，面对网络的动态变化体现了较好的容错能力，因此具有较好的可用性。同时可以支持复杂查询，如带有规则表达式的多关键词查询，模糊查询等。

它与中心化拓扑结构的最大区别在于它是更纯粹的p2p系统，因为它没有中央索引服务器，每台机器在Gnutella网络中是真正的对等关系，既是客户机同时又是服务器，所以被称为对等机(Servent，Server+Client的组合)。在文件检索方面，它与Napster也不相同。在Gnutella网络的发展初期，它主要采用基于完全随机图的Flooding搜索算法。图2 显示了Flooding的工作流程：当一台计算机要下载一个文件，它首先以文件名或者关键字生成一个查询，并把这个查询发送给与它相连的所有计算机，这些计算机如果存在这个文件，则与查询的机器建立连接，如果不存在这个文件，则继续在自己相邻的计算机之间转发这个查询，直到找到文件为止。为了控制搜索消息不至于永远这样传递下去，一般通过TTL (Time To Live)的减值来控制查询的深度。

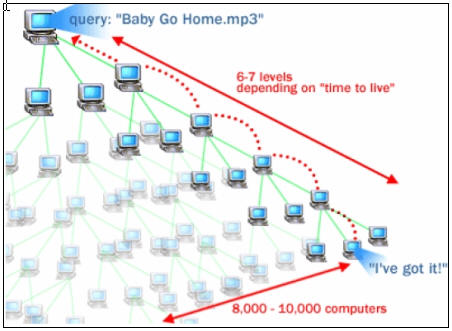


图2Gnutella的拓扑结构和文件检索方法

**分布式结构化拓扑**的P2P网络主要是采用分布式散列表（Distributed Hash Table, DHT）技术来组织网络中的结点。DHT是一个由广域范围大量结点共同维护的巨大散列表。散列表被分割成不连续的块，每个结点被分配给一个属于自己的散列块，并成为这个散列块的管理者。通过加密散列函数，一个对象的名字或关键词被映射为128位或160位的散列值。分布式散列表起源于SDDS（Scalable Distribute Data Structures）研究，Gribble等实现了一个高度可扩展，容错的SDDS集群。DHT类结构能够自适应结点的动态加入/退出，有着良好的可扩展性、鲁棒性、结点ID分配的均匀性和自组织能力。由于重叠网络采用了确定性拓扑结构，DHT可以提供精确的发现。只要目的结点存在于网络中DHT总能发现它，发现的准确性得到了保证。DHT这类结构最大的问题是DHT的维护机制较为复杂，尤其是结点频繁加入退出造成的网络波动（Churn）会极大增加DHT的维护代价。DHT所面临的另外一个问题是DHT仅支持精确关键词匹配查询，无法支持内容/语义等复杂查询。

**半分布式拓扑结构**（也叫混杂模式，Hybrid Structure）吸取了中心化结构和全分布式非结构化拓扑的优点，选择性能较高（处理、存储、带宽等方面性能）的结点作为超级结点（英文表达为SuperNodes或者Hubs），在各个超级结点上存储了系统中其他部分结点的信息，发现算法仅在超级结点之间转发，超级结点再将查询请求转发给适当的叶子结点。半分布式结构也是一个层次式结构，超级结点之间构成一个高速转发层，超级结点和所负责的普通结点构成若干层次。

半分布式结构的优点是性能、可扩展性较好，较容易管理，但对超级点依赖性大，易于受到攻击，容错性也受到影响。

## P2P穿越问题

目前P2P通信在穿透上至少存在着两个问题：防火墙穿透和NAT穿透，两者对于网络访问的限制是处于不同角度而实现的。

1）防火墙是基于网络数据传输安全上的考虑，其行为主要表现为对网络协议和访问端口的限制，实际上每种限制都包含了两个方向：进和出。

2）NAT则是基于网络地址转换的实现对内网主机进行的保护，目前来说NAT的存在至少存在以下两方面的意义：解决IPV4地址匮乏的问题和保护网内主机的目的，所以即使将来IPV6解决了IP地址数量上的问题，但出于对内网主机的保护，NAT仍然有其存在的必要。

综上所述，要实现一个完善的P2P程序必须至少突破以上两个方面的限制，当然，实际情况会存在一些无法突破的情况（比如双方都是对称型NAT或对称型与端口限制型NAT的通信，对于此类问题在实际开发时可使用服务器转发或代理服务来处理）。文中所提到的P2P通信意为：使用服务器来转发并处理控制信令，客户端结点间直接通信。

## 基础知识

### 3.1 NAT简介

目前，IPv4地址资源的紧缺使得NAT技术获得了广泛的应用。NAT技术是一种把内部网络（简称为内网）私有IP地址转换为外部网络（简称为外网）公共IP地址的技术，它使得一定范围内的多台主机只利用一个公共IP地址连接到外网，可以在很大程度上缓解了公网IP地址紧缺的问题。

NAT技术虽然在一定程度上解决了IPv4地址短缺的问题，在构建防火墙、保证网络安全方面都发挥了一定的作用，却破坏了端到端的网络通信。NAT阻碍主机进行P2P通信，主要表现：

1）内网是私有的，外网主机看不到，无法主动发起连接；

2）即使有内网ip及port，但NAT会丢弃没有在映射表中的数据包；

3）内网主机可以作为客户端访问外网，但不能作为服务器提供服务；

4）当两个主机都位于各自的NAT后，要实现P2P连接，就不仅仅是谁主动的问题了，而是解决在两个NAT上同时有对方的映射表项的问题。

要在NAT网络环境中进行有效的P2P通信，就必须寻找相应的解决方案。

### 3.2 NAT知识

#### 3.2.1常见NAT类型

NAT共分为两大类：Cone NAT和Symmetric NAT。Cone NAT指的是只要源IP端口不变，无论发往的目的IP是否相同，在NAT上都映射为同一个端口，形象的看来就像锥子一样，而Symmetric NAT对于发往不同目的IP的会话在NAT上将映射为不同的端口，也就是不同的会话。 其中Cone NAT又可细分为3类，分别是Full Cone型、Restricted Cone型和Restricted Port Cone。限制的严格程度和对局域网内主机的保护由松到紧依次为：**Full Cone、Restricted Cone、Restricted Port Cone、Symmetric NAT**

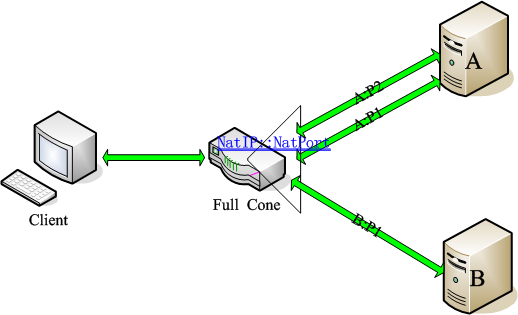
这里“限制”指的是NAT对由外到内的数据包进行审查、过滤，看看数据包的源地址和他发送到的“洞”是否有关系，如果没有那么就将其丢弃

“由松到紧”指后者不仅继承了前者在限制上的特性，而且自己还添油加醋的干了些坏事，以至于对“由外到内”的数据包比前者有着更严格的限制。例如Restricted Cone限制了外部进入内部的IP，使得只有被打洞IP发出的数据包才允许进入NAT，而Restricted Port Cone不但限制了IP，还限制了端口，使得只有被打洞的IP:PORT才能往这个洞里发送数据，其他任何来自不同于被打洞地址（IP:PORT）的数据包都不能使用这个洞将数据发送到NAT之后。

**1）全锥形NAT**

**IP、端口都不受限**。只要客户端由内到外打通一个洞之后（NatIP:NatPort -> A:P1），其他IP的主机(B)或端口(A:P2)都可以使用这个洞发送数据到客户端。

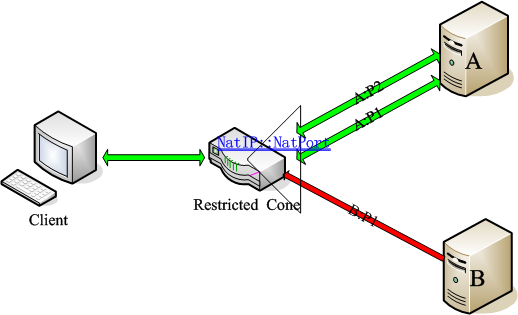
映射关系为：Client->NatIP:NatPort->Any，即任何外部主机都可通过NatIP:NatPort发送数据到Clietn上。



**2）受限锥形NAT**

**IP受限，端口不受限**。当客户端由内到外打通一个洞之后(NatIP:NatPort -> A:P1)，A机器可以使用他的其他端口（P2）主动连接客户端，但B机器则不被允许。

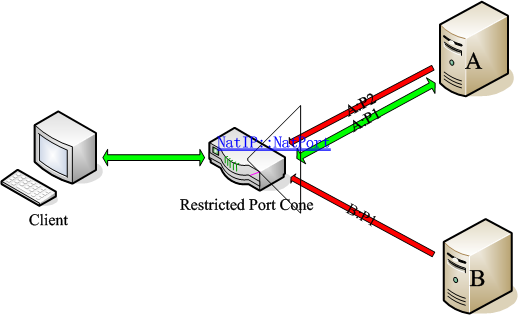
映射关系为：Client-> NatIP:NatPort->A，即只有来自A的数据包才能通过NatIP:NatPort发送到Client上



**3）端口受限锥型**

**IP、端口都受限。**返回的数据只接受曾经打洞成功的对象（A:P1），由A:P2、B:P1发起的数据将不被NatIP:NatPort接收。

映射关系为：Client->NatIP:NatPort->A:P1，即只有来自A:P1的数据才可通过NatIP:NatPort发送到Client上。



#### 3.2.2 NAT类型的探测

**前提条件**：有一个公网的Server并且绑定了两个公网IP(IP-1,IP-2)。这个Server做UDP监听(IP-1,Port-1)，(IP-2,Port-2)并根据客户端的要求进行应答。

**第一步：检测客户端是否有能力进行UDP通信以及客户端是否位于NAT后？**

客户端建立UDP socket然后用这个socket向服务器的(IP-1,Port-1)发送数据包要求服务器返回客户端的IP和Port，客户端发送请求后立即开始接受数据包，要设定socket Timeout（300ms），防止无限堵塞。重复这个过程若干次。如果每次都超时，无法接受到服务器的回应，则说明客户端无法进行UDP通信，可能是防火墙或NAT阻止UDP通信，这样的客户端也就不能P2P了（检测停止）。

当客户端能够接收到服务器的回应时，需要把服务器返回的客户端（IP,Port）和这个客户端socket的 （LocalIP，LocalPort）比较。如果完全相同则客户端不在NAT后，这样的客户端具有公网IP可以直接监听UDP端口接收数据进行通信（检测停止）。否则客户端在NAT后要做进一步的NAT类型检测(继续)。

**第二步：检测客户端NAT是否是Full Cone NAT？**

客户端建立UDP socket然后用这个socket向服务器的(IP-1,Port-1)发送数据包要求服务器用另一对(IP-2,Port-2)响应客户端的请求往回发一个数据包，客户端发送请求后立即开始接受数据包，要设定socket Timeout（300ms），防止无限堵塞. 重复这个过程若干次。如果每次都超时，无法接受到服务器的回应，则说明客户端的NAT不是一个Full Cone NAT，具体类型有待下一步检测(继续)。如果能够接受到服务器从(IP-2,Port-2)返回的应答UDP包，则说明客户端是一个Full Cone NAT，这样的客户端能够进行UDP-P2P通信（检测停止）。

**第三步 检测客户端NAT是否是Symmetric NAT？**

客户端建立UDP socket然后用这个socket向服务器的(IP-1,Port-1)发送数据包要求服务器返回客户端的IP和Port, 客户端发送请求后立即开始接受数据包，要设定socket Timeout（300ms），防止无限堵塞. 重复这个过程直到收到回应（一定能够收到，因为第一步保证了这个客户端可以进行UDP通信）。

用同样的方法用一个socket向服务器的(IP-2,Port-2)发送数据包要求服务器返回客户端的IP和Port。比较上面两个过程从服务器返回的客户端(IP,Port)，如果两个过程返回的(IP,Port)有一对不同则说明客户端为Symmetric NAT，这样的客户端无法进行UDP-P2P通信（检测停止）。否则是Restricted Cone NAT，是否为Port Restricted Cone NAT有待检测(继续)。

**第四步：检测客户端NAT是否是Restricted Cone NAT还是Port Restricted Cone NAT？**

客户端建立UDP socket然后用这个socket向服务器的(IP-1,Port-1)发送数据包要求服务器用IP-1和一个不同于Port-1的端口发送一个UDP 数据包响应客户端，客户端发送请求后立即开始接受数据包，要设定socket Timeout（300ms），防止无限堵塞。重复这个过程若干次。如果每次都超时，无法接受到服务器的回应，则说明客户端是一个Port Restricted Cone NAT，如果能够收到服务器的响应则说明客户端是一个Restricted Cone NAT。以上两种NAT都可以进行UDP-P2P通信。

注：以上检测过程中只说明了可否进行UDP-P2P的打洞通信。另外对于Symmetric NAT不是说完全不能进行UDP-P2P达洞通信，可以进行端口预测打洞，不过不能保证成功。

### 3.3 防火墙知识

防火墙主要基于两个方面的限制：

1）协议限制：可通过隧道技术把受限协议封装成不受限协议而骗过FW，接收后再把封装的数据包解开。比如对于屏蔽了UDP而开放HTTP的防火墙在发送端可使用HTTP格式来包装UDP封包而骗过FW；

2）端口限制的解决方式：使用防火墙开放的端口，大多数防火墙都会开放一些常用端口，比如我们可以使用FW开放的80端口再加上http隧道或使用80端口的代理服务器来绕过防火墙。

## P2P穿越NAT的几种方案

### 4.1反向链接技术

当通信的双方中只有一方位于NAT之后时，它们可以利用反向链接技术来进行P2P通信。图3中Client A（拥有内网IP地址10.0.0.1）位于NAT之后，它通过TCP端口1234连接到服务器（拥有外网IP地址）的TCP端口1235上，NAT设备（拥有外网IP地址155.99.25.11）为这个连接重新分配了TCP端口62000。Client B（拥有外网IP地址138.76.29.7）也通过TCP端口1234连接到服务器端口1235上。Client A和Client B从服务器处获知的对方的外网地址二元组{IP地址:端口号}分别为{138.76.29.7:1234}和{155.99.25.11:62000}，它们在各自的本地端口上进行侦听。

由于Client B 拥有外网IP地址，所以Client A要发起与Client B的通信，那么它可以直接通过TCP连接到Client B。但如果Client B尝试通过TCP连接到Client A进行P2P通信，则会失败，原因是Client A位于NAT设备后，虽然Client B发出的TCP SYN请求能够到达NAT设备的端口62000，但NAT设备会拒绝这个连接请求。要想与Client A通信，Client B要通过服务器给Client A转发一个连接请求，反过来请求Client A连接到Client B（即进行反向链接），从而建立起它们之间的TCP连接。

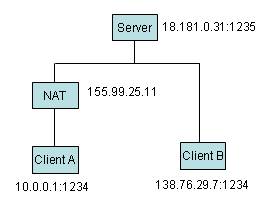


图3 反向链接示意图

### 4.2UDP打洞技术

如果两个P2P客户端都位于NAT设备后面，想要进行P2P通信，那又该如何解决呢？UDP打洞技术就是为解决这个问题而应运而生的，它能够通过中间服务器实现P2P客户端互连。该技术目前在多种在线游戏协议中已经得到了应用，下面来重点介绍下。

#### 4.2.1集中服务器

打洞技术假定**客户端A和客户端B都可以与公网内的已知集中服务器建立UDP连接**，一个客户端在集中服务器上登陆的时候，服务器记录下该客户端的两对地址**二元组信息{IP地址:UDP端口**}，一对是该客户端与集中服务器进行通信的自身的IP地址和端口号，另一对是集中服务器记录下的由服务器“观察”到的该客户端实际与自己通信所使用的IP地址和端口号。我们可以把前一对地址二元组看作是**客户端的内网IP地址和端口号**，把后一对地址二元组看作是**客户端的内网IP地址和端口号经过NAT转换后的外网IP地址和端口号**。集中服务器可以从**客户端的登陆消息中得到该客户端的内网相关信息**，**还可以通过登陆消息的IP头和UDP头得到该客户端的外网相关信息**。如果该客户端不是位于NAT设备后面，那么采用上述方法得到的两对地址二元组信息是完全相同的。

#### 4.2.2建立P2P的Session

假定客户端A要发起对客户端B的直接连接，具体的“打洞”过程如下：

（1）客户端A最初不知道如何向客户端B发起连接，于是客户端A向集中服务器发送消息，请求集中服务器帮助建立与客户端B的UDP连接。

（2）集中服务器将含有客户端B的外网和内网的地址二元组发给客户端A，同时，集中服务器将包含有客户端A的外网和内网的地址二元组信息的消息也发给客户端B。这样一来，客户端A与客户端B就都知道对方外网和内网的地址二元组信息了。

（3）当客户端A收到由集中服务器发来的包含客户端B的外网和内网的地址二元组信息后，客户端A开始向客户端B的地址二元组发送UDP数据包，并且客户端A会自动锁定第一个给出响应的客户端B的地址二元组。同理，当客户端B收到由集中服务器发来的客户端A的外网和内网地址二元组信息后，也会开始向客户端A的外网和内网的地址二元组发送UDP数据包，并且自动锁定第一个得到客户端A回应的地址二元组。由于客户端A与客户端B互相向对方发送UDP数据包的操作是异步的，所以客户端A和客户端B发送数据包的时间先后并没有时序要求。

下面来看下这三者之间是如何进行UDP打洞的。在这我们分三种具体情景来讨论：

第一种是最简单的一种情景，两个客户端都位于同一个NAT设备后面，即位于同一内网中；

第二种是最普遍的一种情景，两个客户端分别位于不同的NAT设备后面，分属不同的内网；

第三种是客户端位于两层NAT设备之后，通常最上层的NAT是由网络提供商提供的，第二层NAT是家用的NAT路由器之类的设备提供的。

#### 4.2.3P2P的两个客户端位于同一个NAT设备后面

首先假设两个客户端位于同一个NAT设备后面，并且位于内网，如图4所示。客户端A与集中服务器建立了UDP连接，经过NAT转换后，A的公网端口被映射为62000。客户端B同样与集中服务器建立了UDP连接，公网端口映射为62005。

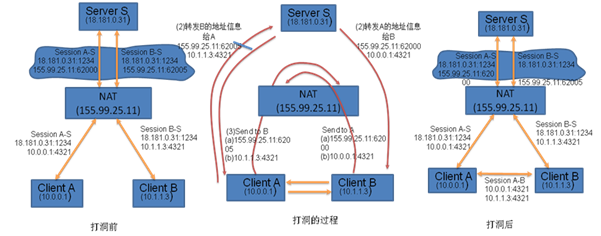


图4 位于同一个NAT设备后的UDP打洞过程

假设客户端A想通过集中服务器，发起对客户端B的连接。客户端A向集中服务器发出消息请求与客户端B进行连接，集中服务器将客户端B的外网地址二元组以及内网地址二元组发给客户端A，同时把客户端A的外网以及内网的地址二元组信息发给客户端B。客户端A和客户端B发往对方公网地址二元组信息的UDP数据包不一定会被对方收到，这取决于当前的NAT设备是否支持**不同端口之间的UDP数据包能否到达**即Hairpin转换特性，无论如何客户端A与客户端B发往对方内网的地址二元组信息的UDP数据包是一定可以到达的，内网数据包不需要路由，且速度更快。客户端A与客户端B推荐采用内网的地址二元组信息进行常规的P2P通信。

假定NAT设备支持Hairpin转换，具体的Hairpin转换见4.2.5章节，应用程序也应忽略与内网地址二元组的连接，如果客户端A、客户端B采用外网的地址二元组做为P2P通信的连接，这势必会造成数据包无谓地经过NAT设备，这是一种对资源的浪费。就目前的网络情况而言，**应用程序在“打洞”的时候，最好还是把外网和内网的地址二元组都尝试**一下。如果都能成功，**优先以内网地址进行连接**。

#### 4.2.4P2P客户端位于不同的NAT设备后面

假定客户端A与客户端B在不同的NAT设备后面，分属不同的内网，如图5所示。客户端A与客户端B都经由各自的NAT设备与集中服务器建立了UDP连接，客户端A与客户端B的本地端口号均为4321，集中服务器的公网端 口号为1234。在向外的会话中，客户端A的外网IP被映射为155.99.25.11，外网端口为62000，客户端B的外网IP被映射为138.76.29.7，外网端口为31000。

如下所示：

客户端A——>本地IP:10.0.0.1，本地端口:4321，外网IP:155.99.25.11，外网端口:62000

客户端B——>本地IP:10.1.1.3，本地端口:4321，外网IP:138.76.29.7，外网端口:31000

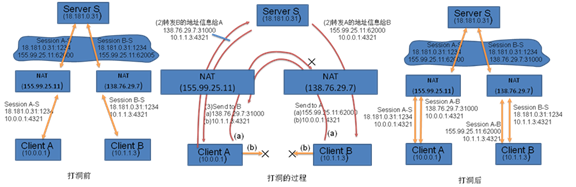


图5 位于不同NAT设备后的UDP打洞过程

在客户端A向服务器发送的登陆消息中，包含有客户端A的内网地址二元组信息，即10.0.0.1:4321；服务器会记录下客户端A的内网地址二元组信息，同时会把自己观察到的客户端A的外网地址二元组信息记录下来，即155.99.25.11:62000。同理，服务器也会记录下客户端B的内网地址二元组信息为10.1.1.3:4321和由服务器观察到的客户端B的外网地址二元组信息，138.76.29.7:31000。无论A与B二者中的任何一方向服务器发送P2P连接请求，服务器都会将其记录下来的上述的外网和内网地址二元组发送给A或B。

A、B分属不同的内网，它们彼此的内网地址在外网中是没有路由的，所以发往各自内网地址的UDP数据包会发送到错误的主机或者根本不存在的主机上。现在假定A的第一个消息将发往B的外网地址，如图5所示。该消息途经A的NAT设备，并在该设备上生成一个会话表项，该会话的源地址二元组信息是{10.0.0.1:4321}，该地址二元组信息和客户端A与服务器建立连接的时候NAT生成的源地址二元组信息一样，但它的目的地址不同。如果A的NAT设备给出的响应是OK的，那么A的NAT设备将保留A的内网地址二元组信息，并且所有来自A的源地址二元组信息为{10.0.0.1:4321}的数据包都沿用A与集中服务器事先建立起来的会话，外网地址二元组信息均为{155.99.25.11:62000}。

A向B的外网地址发送消息的过程就是“打洞”的过程，从A的内网的角度来看应为从{10.0.0.1:4321}发往{138.76.29.7:31000}，从A在其NAT设备上建立的会话来看，是从{155.99.25.11:62000}发到{138.76.29.7:31000}。

如果A发给B的外网地址二元组的消息包在B向A发送消息包之前到达B的NAT设备，B的NAT设备会认为A发过来的消息是未经授权的外网消息，会丢弃掉该数据包。B发往A的消息包与上述的过程一样，会在B的NAT设备上建立一个{10.1.1.3:4321，155.99.25.11:62000}的会话（通常也会沿用B与集中服务器连接时建立的会话，只是该会话现在不仅接受由服务器发给B的消息，还可以接受从A的NAT设备155.99.25.11:6200发来的消息），一旦A与B都向对方的NAT设备在外网上的地址二元组发送了数据包，就打开了A与B之间的“洞”，A与B向对方的外网地址发送数据，等效为向对方的客户端直接发送UDP数据包了。一旦应用程序确认已经可以通过往对方的外网地址发送数据包的方式让数据包到达NAT后面的目的应用程序，程序会自动停止继续发送用于“打洞”的数据包，转而开始真正的P2P数据传输。

#### 4.2.5 P2P客户端位于多层NAT设备后面

有的网络拓扑结构包含了多个NAT设备，如果没有掌握该拓扑结构的详细信息，两个客户端之间是无法建立“最优化”的P2P路由的。现在我们来讨论最后一种情况，如图6所示。假定NAT C是由ISP(Internet Service Provider)提供的NAT设备，NAT C提供将多个用户节点映射到有限的几个公网IP的服务，NAT A和NAT B作为NAT C的内网节点将把用户的家庭网络或内部网络接入NAT C的内网，然后用户的内部网络就可以经由NAT C访问公网了。从这种拓扑结构上来看，只有服务器与NAT C是真正拥有公网可路由IP地址的设备，而NAT A和NAT B所使用的公网IP地址，实际上是由ISP服务提供商设定的（相对于NAT C而言）内网地址（本文的后续部分把这个由ISP提供的内网地址相对于NAT C称之为“伪”公网地址），同理隶属于NAT A与NAT B的客户端，相对与NAT A，NAT B而言，它们处于NAT A，NAT B的内网，以此类推，客户端可以放到到多层NAT设备后面。客户端A和客户端B发起对服务器S的连接的时候，就会依次在NAT A和NAT B上建立向外的Session，而NAT A、NAT B要联入公网的时候，会在NAT C上再建立向外的Session。

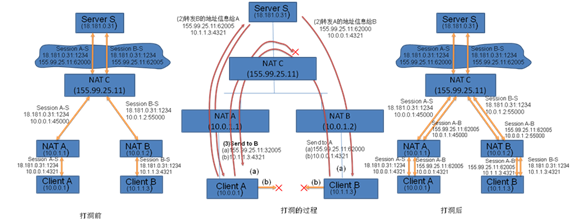


图6 多层NAT下的打洞过程

现在假定客户端A和B希望通过UDP“打洞”完成两个客户端的P2P直连。最优化的路由策略是客户端A向客户端B的“伪公网”IP上发送数据包，即ISP服务提供商指定的内网IP，NAT B的“伪”公网地址二元组，{10.0.1.2:55000}。由于从服务器的角度只能观察到真正的公网地址，也就是NAT A，NAT B在NAT C建立session的真正的公网地址{155.99.25.11:62000}以及{155.99.25.11:62005}，非常不幸的是客户端A与客户端B是无法通过服务器知道这些

“伪”公网的地址，而且即使客户端A和B通过某种手段可以得到NAT A和NAT B的“伪”公网地址，我们仍然不建议采用上述的“最优化”的打洞方式，这是因为这些地址是由ISP服务提供商提供的或许会存在与客户端本身所在的内网地址重复的可能性（例如:NAT A的内网的IP地址域恰好与NAT A在NAT C的“伪”公网IP地址域重复，这样就会导致打洞数据包无法发出的问题）。

因此客户端别无选择，只能使用由公网服务器观察到的A，B的公网地址二元组进行“打洞”操作，用于“打洞”的数据包将由NAT C进行转发。

当客户端A向客户端B的公网地址二元组{155.99.25.11:62005}发送UDP数据包的时候，NAT A首先把数据包的源地址二元组由A的内网地址二元组{10.0.0.1:4321}转换为“伪”公网地址二元组{10.0.1.1:45000}，现在数据包到了NAT C，NAT C应该可以识别出来该数据包是要发往自身转换过的公网地址二元组，如果NAT C可以给出“合理”响应的话，NAT C将把该数据包的源地址二元组改为{155.99.25.11:62000}，目的地址二元组改为{10.0.1.2:55000}，即NAT B的“伪”公网地址二元组，NAT B最后会将收到的数据包发往客户端B。同样，由B发往A的数据包也会经过类似的过程。目前也有很多NAT设备不支持类似这样的“Hairpin转换”，但是已经有越来越多的NAT设备商开始加入对该转换的支持中来。

#### 4.2.6UDP在空闲状态下的超时问题

由于UDP转换协议提供的“洞”不是绝对可靠的，多数NAT设备内部都有一个UDP转换的空闲状态计时器，如果在一段时间内没有UDP数据通信，NAT设备会关掉由“打洞”操作打出来的“洞”，作为应用程序来讲如果想要做到与设备无关，就最好在穿越NAT以后设定一个穿越的有效期。

很遗憾目前没有标准有效期，这个有效期与NAT设备内部的配置有关，某些设备上最短的只有20秒左右。在这个有效期内，即使没有P2P数据包需要传输，应用程序为了维持该“洞”可以正常工作，也必须向对方发送“打洞”心跳包。这个心跳包是需要双方应用程序都发送的，只有一方发送不会维持另一方的Session正常工作。除了频繁发送“打洞”心跳包以外，还有一个方法就是在当前的“洞”超时之前，P2P客户端双方重新“打洞”，丢弃原有的“洞”，这也不失为一个有效的方法。

### 4.3关于TCP打洞技术

建立穿越NAT设备的P2P的TCP连接只比UDP复杂一点点，TCP协议的”“打洞”从协议层来看是与UDP的“打洞”过程非常相似的。尽管如此，基于TCP协议的打洞至今为止还没有被很好的理解，这也造成了的对其提供支持的NAT设备不是很多。在NAT设备支持的前提下，基于TCP的“打洞”技术实际上与基于UDP的“打洞”技术一样快捷、可靠。实际上，只要NAT设备支持的话，基于TCP的P2P技术的健壮性将比基于UDP技术的更强一些，因为**TCP协议的状态机给出了一种标准的方法来精确的获取某个TCP session的生命期**，而UDP协议则无法做到这一点。

#### 4.3.1套接字和TCP端口的重用

实现基于TCP协议的P2P打洞过程中，最主要的问题不是来自于TCP协议，而是来自于应用程序的API接口。这是由于标准的伯克利(Berkeley)套接字的API是围绕着构建客户端/服务器程序而设计的，API允许TCP流套接字通过调用connect()函数来建立向外的连接，或者通过listen()和accept函数接受来自外部的连接，但是，API不提供类似UDP那样的，同一个端口既可以向外连接，又能够接受来自外部的连接。而且更糟的是，TCP的套接字通常仅允许建立1对1的响应，即**应用程序在将一个套接字绑定到本地的一个端口以后，任何试图将第二个套接字绑定到该端口的操作都会失败**。  
      为了让TCP“打洞”能够顺利工作，我们需要使用一个本地的TCP端口来监听来自外部的TCP连接，同时建立多个向外的TCP连接。幸运的是，所有的主流操作系统都能够支持特殊的TCP套接字参数，通常叫做“**SO\_REUSEADDR**”，该参数**允许应用程序将多个套接字绑定到本地的一个地址二元组**（只要所有要绑定的套接字都设置了SO\_REUSEADDR参数即可）。BSD系统引入了SO\_REUSEPORT参数，该参数用于区分端口重用还是地址重用，在这样的系统里面，上述所有的参数必须都设置才行。

#### 4.3.2打开P2P的TCP流

假定客户端A希望建立与B的TCP连接。我们像通常一样假定A和B已经与公网上的已知服务器建立了TCP连接。服务器记录下来每个接入的客户端的公网和内网的地址二元组，如同为UDP服务的时候一样。从协议层来看，TCP“打洞”与UDP“打洞”是几乎完全相同的过程。

l  客户端A使用其与服务器的连接向服务器发送请求，要求服务器协助其连接客户端B；

l  服务器将B的公网和内网的TCP地址的二元组信息返回给A，同时，服务器将A的公网和内网的地址二元组也发送给B；

l  客户端A和B使用连接服务器的端口异步地发起向对方的公网、内网地址二元组的TCP连接，同时监听各自的本地TCP端口是否有外部的连接联入；

l  A和B开始等待向外的连接是否成功，检查是否有新连接联入。如果向外的连接由于某种网络错误而失败，如：“连接被重置”或者“节点无法访问”，客户端只需要延迟一小段时间（例如延迟一秒钟），然后重新发起连接即可，延迟的时间和重复连接的次数可以由应用程序编写者来确定；

l  TCP连接建立起来以后，客户端之间应该开始鉴权操作，确保目前联入的连接就是所希望的连接。如果鉴权失败，客户端将关闭连接，并且继续等待新的连接联入。客户端通常采用“先入为主”的策略，只接受第一个通过鉴权操作的客户端，然后将进入P2P通信过程不再继续等待是否有新的连接联入。

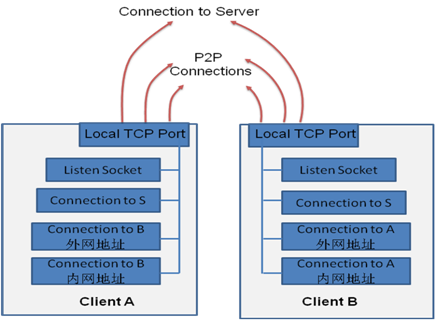


图7 TCP打洞

与UDP不同的是，因为使用UDP协议的每个客户端只需要一个套接字即可完成与服务器的通信，而TCP客户端必须处理多个套接字绑定到同一个本地TCP端口的问题，如图7所示。现在来看实际中常见的一种情景，A与B分别位于不同的NAT设备后面，如图5所示，并且假定图中的端口号是TCP协议的端口号，而不是UDP的端口号。图中向外的连接代表A和B向对方的内网地址二元组发起的连接，这些连接或许会失败或者无法连接到对方。如同使用UDP协议进行“打洞”操作遇到的问题一样，TCP的“打洞”操作也会遇到内网的IP与“伪”公网IP重复造成连接失败或者错误连接之类的问题。  
    客户端向彼此公网地址二元组发起连接的操作，会使得各自的NAT设备打开新的“洞”允许A与B的TCP数据通过。如果NAT设备支持TCP“打洞”操作的话，一个在客户端之间的基于TCP协议的流通道就会自动建立起来。如果A向B发送的第一个SYN包发到了B的NAT设备，而B在此前没有向A发送SYN包，B的NAT设备会丢弃这个包，这会引起A的“连接失败”或“无法连接”问题。而此时，由于A已经向B发送过SYN包，B发往A的SYN包将被看作是由A发往B的包的回应的一部分，所以B发往A的SYN包会顺利地通过A的NAT设备，到达A，从而建立起A与B的P2P连接。

#### 4.3.3从应用程序的角度来看TCP“打洞”

从应用程序的角度来看，在进行TCP“打洞”的时候都发生了什么呢？假定A首先向B发出SYN包，该包发往B的公网地址二元组，并且被B的NAT设备丢弃，但是B发往A的公网地址二元组的SYN包则通过A的NAT到达了A，然后，会发生以下的两种结果中的一种，具体是哪一种取决于操作系统对TCP协议的实现：

（1）A的TCP实现会发现收到的SYN包就是其发起连接并希望联入的B的SYN包，通俗一点来说就是“说曹操，曹操到”的意思，本来A要去找B，结果B自己找上门来了。A的TCP协议栈因此会把B作为A向B发起连接connect的一部分，并认为连接已经成功。程序A调用的异步connect()函数将成功返回，A的listen()等待从外部联入的函数将没有任何反映。此时，B联入A的操作在A程序的内部被理解为A联入B连接成功，并且A开始使用这个连接与B开始P2P通信。

由于收到的SYN包中不包含A需要的ACK数据，因此，A的TCP将用SYN-ACK包回应B的公网地址二元组，并且将使用先前A发向B的SYN包一样的序列号。一旦B的TCP收到由A发来的SYN-ACK包，则把自己的ACK包发给A，然后两端建立起TCP连接。简单的说，第一种，就是即使A发往B的SYN包被B的NAT丢弃了，但是由于B发往A的包到达了A。结果是，A认为自己连接成功了，B也认为自己连接成功了，不管是谁成功了，总之连接是已经建立起来了。

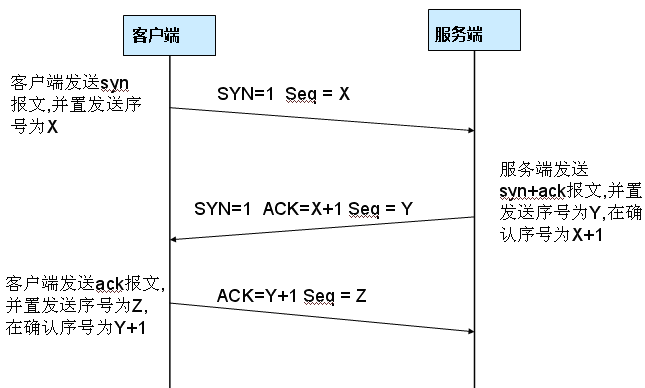
（2）另外一种结果是，A的TCP实现没有像（1）中所讲的那么“智能”，它没有发现现在联入的B就是自己希望联入的。就好比在机场接人，明明遇到了自己想要接的人却不认识，误认为是其他的人，安排别人给接走了，后来才知道是自己错过了机会，但是无论如何，人已经接到了任务已经完成了。然后，A通过常规的listen()函数和accept()函数得到与B的连接，而由A发起的向B的公网地址二元组的连接会以失败告终。尽管A向B的连接失败，A仍然得到了B发起的向A的连接，等效于A与B之间已经联通，不管中间过程如何，A与B已经连接起来了，结果是A和B的基于TCP协议的P2P连接已经建立起来了。

第一种结果适用

于基于BSD的操作系统对于TCP的实现，而第二种结果更加普遍一些，多数Linux和Windows系统都会按照第二种结果来处理。

**TCP建立连接知识补充**：

（1）客户端发送一个带SYN标志的TCP报文到服务器。这是三次握手过程中的报文1。  
（2） 服务器端回应客户端的，这是三次握手中的第2个报文，这个报文同时带ACK标志和SYN标志。因此它表示对刚才客户端SYN报文的回应；同时又标志SYN给客户端，询问客户端是否准备好进行数据通 讯。  
（3） 客户必须再次回应服务段一个ACK报文，这是报文段3



## 5 防火墙穿越的几种方案

### 5.1隧道

采用FW允许的隧道协议包装应用程序的自定义协议，主要用于突破防火墙的协议封锁，例如：Http隧道、SOCKS隧道、SSL隧道等等。

不可幻想使用原始套接字自行封装传输层及其以上协议来解决此问题，此法在技术和理论上虽然可行，但那样带来的后续问题和工作量将会远远超过“使用TCP打洞”方案，除非你想发明一种更好的穿墙技术甚至一种传输层协议 。

所以，如果非要在“不使用代理”而又要“穿越防火墙”的情况下进行P2P的话，目前而言最佳的选择只有“使用TCP打洞”，在TCP打洞成功后即可使用HttpTunnel在终端间进行传输，达到既穿墙又直接传输数据的目的

HTTPTunnel，Tunnel这个英文单词的意思是隧道，通常HTTPTunnel被称之为HTTP暗道，它的原理就是将数据伪装成 HTTP的数据形式来穿过防火墙，实际上是在HTTP请求中创建了一个双向的虚拟数据连接来穿透防火墙。说得简单点，就是说在防火墙两边都设立一个转换程序，将原来需要发送或接受的数据包封装成HTTP请求的格式骗过防火墙，所以它不需要别的代理服务器而直接穿透防火墙。HTTPTunnel包括两个程序，htc和hts，其中htc是客户端，而hts是服务器端，我们现在来看看我是如何用它们的。

比如开了FTP的机器的IP是 192.168.1.231，我本地的机器的IP是192.168.1.226，现在我本地因为防火墙的原因无法连接到 FTP上，现在用HTTPTunnel的过程如下：

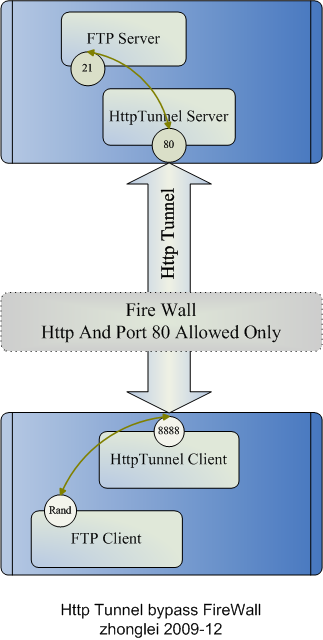
**第一步：在我的机器上（192.168.1.226）启动HTTPTunnel客户端**。启动MS-DOS的命令行方式，然后执行htc -F 8888 192.168.1.231:80命令，其中htc是客户端程序，-F参数表示将来自192.168.1.231:80的数据全部转发到本机的8888端口，进入8888的数据通过http封包后转发到192.168.1.231:80，这个端口可以随便选，只要本机没有占用就可以。然后我们用Netstat看一下本机现在开放的端口，发现8888端口已在侦听。

**第二步：在对方机器上启动HTTPTunnel的服务器端，并执行命令“hts -F localhost:21 80”**，这个命令的意思是说把本机21端口发送的数据都转到80端口后再发出去，并且开放80端口作为侦听端口，80端口进来的数据都转到21上，再用Neststat看一下他的机器，就会发现80端口现在也在侦听状态。

第三步：在我的机器上用FTP连接本机的8888端口，现在已经连上对方的机器了，快点去下载吧！

可是，人家看到的怎么是127.0.0.1而不是192.168.1.231的地址？因为我现在是连接本机的8888端口，防火墙肯定不会有反应，因为我没往外发包，当然局域网的防火墙不知道了。现在连接上本机的8888端口以后，FTP的数据包不管是控制信息还是数据信息，都被htc伪装成HTTP数据包然后发过去，在防火墙看来，这都是正常数据，相当于欺骗了防火墙。

需要说明的是，这一招的使用需要其他机器的配合，就是说要在他的机器上启动一个hts，把他所提供的服务，如FTP等重定向到防火墙所允许的80 端口上，这样才可以成功绕过防火墙！



### 5.2代理转发

可绕过防火墙的IP封锁、端口封锁、协议封锁，这是比较万能而且通用的突破方法，缺点就是需要部署代理服务器，并且需要在终端之间转发数据包，从而降低端到端的通信速率，不能实现端到端的直接通信，对于一些实时性要求不高和通信量不大的应用来说倒是一个比较好的选择，例如一些木马程序（当然这时候对代理服务器的选择需要做一些 手脚，否则别人一抓包就知道了你的代理服务器地址）和远程控制软件。

我们经常使用的代理服务器按协议类型大致包含：SSL代理、FTP代理、Http代理、SOCKS代理、Gopher代理，也就是说这些代理服务分别支持了不同的网络协议，以针对不同协议的代理客户端，某种协议的代理服务器和其客户端自然就是使用那种指定的协议进行通信的。

**代理服务器的原理：**

代理服务器是部署在局域网和外部Internet之间的中间代理机构，负责局域网内主机和外部Internet主机之间的数据转发，其中，网内主机通过指定协议（具体得看代理服务器支持什么样的协议）与代理服务器通信(这里发给代理服务器的数据包是经过代理协议封包之后的，其中包有客户端期望与远端服务器通信的原始协议数据包)，当代理服务器收到网内主机发过来的包之后对其进行解释，然后剥离出原有的原始协议数据，再将其原样发送到真实目的主机，收到远端主机回复后将其进行协议封包(具体得看代理服务器使用什么协议与代理客户端通信)，网内主机收到，然后对其进行代理协议解释，最终剥离出真实协议数据。

代理协议封包的目的是为了代理服务器和代理客户端之间进行通信，当代理服务器与外界进行通信时将剥离出原有封包内的真实协议数据，也就是说我们可以使用代理协议来骗过防火墙，让他认为我们只是在使用他允许的代理协议进行通信，而不对这些数据包进行协议过滤，前提是防火墙必须开放这个代理协议的限制，使其畅通无阻。

## 6 P2P传输建议

（1）首先作为P2P的营运商，可以多设几台P2P种子服务器，分布在不同的网段中。比如：北方网通设一台(组)，南方电信设一台(组)，种子的内容是一样的。种子服务器多了，可以降低优化算法的难度。

（2）种子服务器和普通节点的优先级：种子服务器的优先级总数低于普通节点的，如果普通节点的速度快了，就减少从种子服务器获取的数据量。

（3） 全球IP地址表。P2P节点仲裁服务器中，应该有一个全球IP地址表，分中国大陆、香港、台湾、北美、欧洲、澳洲、其它。中国大陆先按照营运商分：电信、网通、铁通、联通、教育网等，再按照省份分类。(网上有下载，可以整理)

（4） 高速网段表。在P2P访问中，节点动态地将速度快的其它节点IP地址传回服务器，服务器根据全球IP地址表算出网段，以网段-网段的方式记录在数据库中。

（5）当一个新用户连入节点时，在全球IP地址表中找到最近的节点，按照比例依次分配最快网段的节点;最近的节点;差一个级别的稍近的节点;随机节点以及种子服务器。

（6）P2P在数据传送中，可以将30秒数据文件作为1块数据包;数据包中按照每16KB作为一个数据块。每个时间段(如2秒)，本节点向其它节点交换一下数据块的传送情况，然后计算一下数据包中每个数据块的拥有率，优先传送拥有率低的数据块。在拥有率相当的情况下，随机选择。

（7）在数据交换中，对于传送慢的节点，定期剔除，然后问节点仲裁服务器要新的节点。

（8）如果数据包中小于10%的数据块没有传送完毕，在时间充足的情况下，对于余下的数据块，可以同一个数据块向多个节点请求。

（9）节点仲裁服务器也会将新的P2P节点强行加载到另一个节点上，但不能超过节点最大连接数。

（10）普通节点定期向调试服务器上传本节点的信息以文件调度

**P2P通信问题：**

P2P 网络的拓扑结构和 Peer 节点的功能角色划分；

资源和服务如何标识；

进行资源查找时如何进行 Peer 定位；

P2P 网络中 Peer 节点的动态变化的处理；

如何穿越 NAT（Network Address Translation）和防火墙进行 Peer 节点之间的直接通信

## 7面临的问题

我们可能会面对如下问题。

### 7.1知识产权保护

在P2P共享网络中普遍存在着知识产权保护问题。尽管目前Gnutella、Kazaa等P2P共享软件宣传其骨干服务器上并没有存储任何涉及产权保护的内容的备份，而仅仅是保存了各个内容在互联网上的存储索引。但无疑的是，P2P共享软件的繁荣加速了盗版媒体的分发，提高了知识产权保护的难点。美国唱片工业协会RIAA与这些共享软件公司展开了漫长的官司拉锯战，著名的Napster便是这场战争的第一个牺牲者。另一个涉及面很关的战场则是RIAA和使用P2P来交换正版音乐的平民。从2004年1月至今RIAA已提交了1000份有关方面的诉讼。尽管如此，至今每个月仍然有超过150,000,000的歌曲在网络上被自由下载。后Napster时代的P2P共享软件较Napster更具有分散性，也更难加以控制。即使P2P共享软件的运营公司被判违法而关闭，整个网络仍然会存活，至少会正常工作一段时间。  
　　另一方面，Napster以后的P2P共享软件也在迫切寻找一个和媒体发布厂商的共生互利之道。如何更加合法合理的应用这些共享软件，是一个新时代的课题。毕竟P2P除了共享盗版软件，还可以共享相当多的有益的信息。  
　　网络社会与自然社会一样，其自身具有一种自发地在无序和有序之间寻找平衡的趋势。P2P技术为网络信息共享带来了革命性的改进，而这种改进如果想要持续长期地为广大用户带来好处，必须以不损害内容提供商的基本利益为前提。这就要求在不影响现有P2P共享软件性能的前提下，一定程度上实现知识产权保护机制。目前，已经有些P2P厂商和其它公司一起在研究这样的问题。这也许将是下一代P2P共享软件面临的挑战性技术问题之一。

### 7.2网络带宽问题

P2P文件共享和下载给用户带来了很大便利。然而，随着它的大面积流行，P2P下载流量占用带宽接入的大量资源，被许多ISP视为洪水猛兽。据统计，在一些地方，Internet超过了70%以上的流量被P2P相关应用占据，很多公司和学校不得不封杀P2P端口来阻止这种视频、音频文件传输。这样的下载流量在有时会影响了某些用户使用正常的Web, Email以及视频点播等业务。

这在某种程度上就不利于我们p2p的推广。