第一章

* 1. **P2P网络的定义**

P2P网络是一种在IP网络之上的应用层的分布式网络，网络的参与者即对等节点(Peer)共享他们所拥有的一部分硬件资源(如处理能力、存储能力、网络连接能力等)。P2P网络中的这些共享资源提供的服务和内容能被P2P网络中的节点访问，而且访问过程不需要经过P2P网络以外的其他中间实体。在P2P网络中的对等节点既是资源提供者，又是资源获取者。

人们常常将P2P狭义的理解为P2P文件共享和内容分发，但实际上P2P还可以用于实时通信、网络游戏、协同工作等应用。P2P网络节点也不一定是用户PC机，也有可能是由服务器或者是其他特殊终端(比如各类传感器)组成的。

P2P与分布式计算是一种从属关系，分布式计算是一个更大的范畴。分布式计算最初是指在一台计算机多个处理器之间的协同计算，但随着计算机网络的发展，分布式计算一般是指多台计算机之间协同完成任务。P2P属于分布式计算，但是分布式计算不一定是P2P，因为分布式计算中的各台计算机之间的角色可能不对等。

* 1. **P2P的特点**

传统的C/S模式中，客户端之间进行交互都需要以来中心化的服务器，服务器就容易成为网络的瓶颈，当网络规模越大时，服务器负载就越重，一但服务器崩溃整个网络就会瘫痪。而且由于服务器能力有限，往往不能及时地对每一个客户端的请求作出响应，因此造成资源利用率低下。P2P网络中没有中心化的服务器，不存在系统瓶颈，每个节点既当客户端又当服务器，因此具有很高的资源利用率。

归结来说，P2P网络的特点主要体现在以下几个方面上：

* 可扩展性

对于用户节点组成的P2P网络，随着用户的加入，虽然服务的需求增加了，但相应的系统整体的资源和服务能力也在同步的扩充，始终能够比较容易地满足用户的需要，理论上其可扩展性是无限的。由于P2P自组织、自配置、自动负载均衡的特性，系统扩容就变得非常容易了。

* 健壮性

P2P架构本身具有耐攻击、高容错的优点。P2P网络一般在部分节点失效时能够自动调整整体拓扑，保持其他节点的连通性。P2P网络通常都是以自组织的方式建立起来的，并允许节点自由地加入和离开。P2P网络还能够根据网络带宽、节点性、负载等变化不断地做自适应式的调整。

* 高性能/价格比

P2P架构可以有效地利用互联网中散布的大量普通用户节点的空闲资源，不需要部署服务器或需要的服务器很少。P2P网络可以将计算任务或数据分布到所有用户节点上，利用其中闲置的带宽、计算能力或存储空间；另一方面，可以使用一群高性价比的普通服务器来取代价格极高的超级服务器。

* 私密性

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一个集中环节，用户的隐私信息被窃听和泄露的可能性大大缩小。此外，采用中继转发的技术方法，可以将通信的参与者隐藏在众多的网络实体中。传统的匿名通信通常使用中继服务器来实现匿名通信的目的；而在P2P中，所有的参与者都可以提供中继转发的功能，大大提高了匿名通信的灵活性和可靠性，能够为用户提供更好的隐私保护。

* 流量均衡

P2P网络环境下，硬件资源和数据内容分布在多个节点，而P2P节点可以分布在网络各个角落，可以很好地实现整个网络的流量均衡。

* 自组织、低部署维护成本

P2P网络采用了自动计算技术，以实现P2P网络的自组织、自配置、自愈等特性，从而大大降低人为的干预的需要。

* 1. **P2P的功能和对应技术**

P2P网络主要的功能可以分为3种：基于P2P的数据分发和传输、基于P2P的数据存储和检索、基于P2P的分布式数据处理。

* 基于P2P的数据分发和传输

基于P2P的数据分发和传输是指P2P网络中点到点的具体数据传输过程，传输方式有一对一、一对多、多对多三种模式。一对一模式即一个源端到一个目的端的传输，主要用于即时通信和VOIP等应用的用户间直接通信；一对多模式即一个源端到多个目的端的传输，主要用于群组通信和消息广播，还常用于P2P网络的维护；多对多模式即多个源端到多个目的端的传输，主要见于内容分发应用(文件分发和流媒体分发)。

* 基于P2P数据存储和检索

包括基于结构化P2P网络的方法和基于非结构化P2P网络的方法两类。

* 基于P2P的分布式数据处理

利用P2P对等网络，也能够进行分布式数据处理技术，但这一部分知识基于的是分布式计算技术，涉及大量分布式计算与并行计算的知识。

第三章 P2P网络模型概述

1. **集中目录式P2P网络模型**
   1. **原理介绍**

集中目录式P2P网络模型是最早出现的P2P应用模式。由于它采用中央目录服务器管理P2P网络中的各个节点，仍然具有中心化的特点，因而也被称为非纯粹的P2P结构。

传统的C/S模式采用的是一种垄断的手段，所有资料存放在服务器上，客户端只能被动地从服务器上读取数据，而且客户端之间没有交互能力。而在集中目录式P2P结构中，中央目录服务器只保留索引信息，由对等节点负责保存各自提供服务的全部资料；此外，服务器与对等节点以及对等节点之间都具有交互能力。

集中目录式P2P网络模型采用星型拓扑结构，群组中所有对等节点都与中心目录服务器相连，并向其发布共享的文件列表。查询节点可向中心目录服务器发起文件检索请求，得到回复后，查询节点则根据网络流量和延迟等信息选择合适的节点建立直接连接，而不必经过中央服务器，文件的交换可以直接在两对等节点之间进行。在该过程中，中央目录服务器会负责记录群组中所有参加者的信息，以此来进行适当的管理。

* 1. **优缺点介绍**

**优点：**资源的发现依赖于中心化的目录结构，发现算法灵活高效并能够实现复杂查询；这种结构提高了网络的可管理性，使得对共享资源的查找和更新非常方便。

**缺点：**

* 可靠性和安全性较低，因为与传统的C/S结构类似，容易造成单点故障。中央目录服务器的失效意味着该服务器下的对等节点全部失效。
* 维护成本高。随着网络规模的增大，对中央服务器进行维护和更新的费用将急剧上升。
* 存在法律版权和资料浪费

总结：基于以上优缺点，集中目录式网络结构只适用于小型网络，不适合于大型网络应用。

* 1. **典型应用：****BitTorrent**

|  |  |
| --- | --- |
| 名词 | 意义 |
| Tracker | 收集下载者信息的服务器，并将此信息提供给其他下载者，使下载者们相互连接起来，传输数据 |
| 种子 | 当一个下载任务中所有的文件都被某一个下载者完整地下载后，此时的下载者成为一个种子。发布者本身所发布的文件就是原始种子。 |
| 做种 | 发布者提供下载任务的全部内容的行为以及下载者下载完成后继续提供给他人下载的行为 |
| .torrent文件 | 由文件发布者生成，包含了要下载文件的信息：文件名、大小、文件的散列信息以及指向Tracker服务器的URL。 |

Tracker服务器相当于中央目录服务器，起资源定位作用，Tracker服务器负责收集种子和下载者的信息，当收到下载请求时将相关信息提供给下载者，使下载者可以相互连接起来传输数据。种子和下载者相当于对等节点，他们在Tracker服务器中进行注册，当下载者需要下载文件时，通过解析事先下载好的.torrent文件获得服务器的地址，并向服务器发出请求，再根据服务器返回的种子列表进行数据传输(这个过程在种子和下载者之间完成，不经过服务器)。

**BitTorrent工作模式详解**

BitTorrent协议是架构于TCP/IP之上的一个P2P文件传输协议，处于TCP/IP结构的应用层。

BitTorrent完成一次下载过程，至少需要一个Tracker服务器和一个“种子”(拥有完整被下载文件的节点)。下载时，下载者需要先从Web服务器上下载一个扩展名为.torrent的文件（这个文件本质上是一个文本文件，包含Tracker信息和文件信息两部分。Tracker信息主要是BT下载中需要用到的Tracker服务器的地址和针对于Tracker服务器的设置；文本信息是根据对目标文件的计算生成的），通过解析获得Tracker服务器的地址，然后连接至该服务器。Tracker服务器回应下载者的请求，提供给该下载者其他下载者的IP(包括种子和发布者的IP)。该下载者再连接到其他下载者，根据.torrent文件的文件信息，两者分别告知对方自己已有的块，然后交换对方没有的数据(BitTorrent协议中，目标下载文件会被分为若干256KB的块，块在整个文件中是顺序排列的，每个块都有自己的编号。一个块还可以被分成多个16KB的子块，并把每个块的索引信息和Hash验证码写入.torrent文件中，所以.torrent文件就是被下载文件的“索引”。当下载者每下载完成一个块就需要用摘要验证其正确性，即算出这个下载块的Hash验证码与.torrent文件中的进行对比，只有验证正确的数据才能提供给其他节点下载，否则要重新下载这个块。为了尽快共享整块数据，下载时总是会优先下载同一块内的子块)，这样的方式大大减轻了服务器的负担。

此外，在下载过程中，下载者需要周期性地向Tracker服务器登记，使得Tracker服务器能够了解它们的进度。这是由于在传输开始时，只有种子发布者拥有全部文件，在传输进行过程中，有部分下载者会获得部分文件，随着传输的继续，文件会全部公布在系统中。所以通过周期性地向Tracker服务器登记下载进度，即使种子发布者退出网络也不会影响该文件的传播。

1. **纯P2P网络模型**

纯P2P网络模型中，每个节点既是客户端也是服务器，节点之间的通信(包括发送请求、接收响应以及下载文件等)是完全对等的。每个节点都维护着一个邻居列表，节点通过和他的邻居进行交互来完成特定的行为，整个过程不再需要服务器。

纯P2P可以从网络拓扑上进一步分为非结构覆盖网络(Unstructured Overlay Network)和结构覆盖网络(Structured Overlay Network)两种，两者之间的差异很大。

* 1. **纯P2P非结构化网络模型**

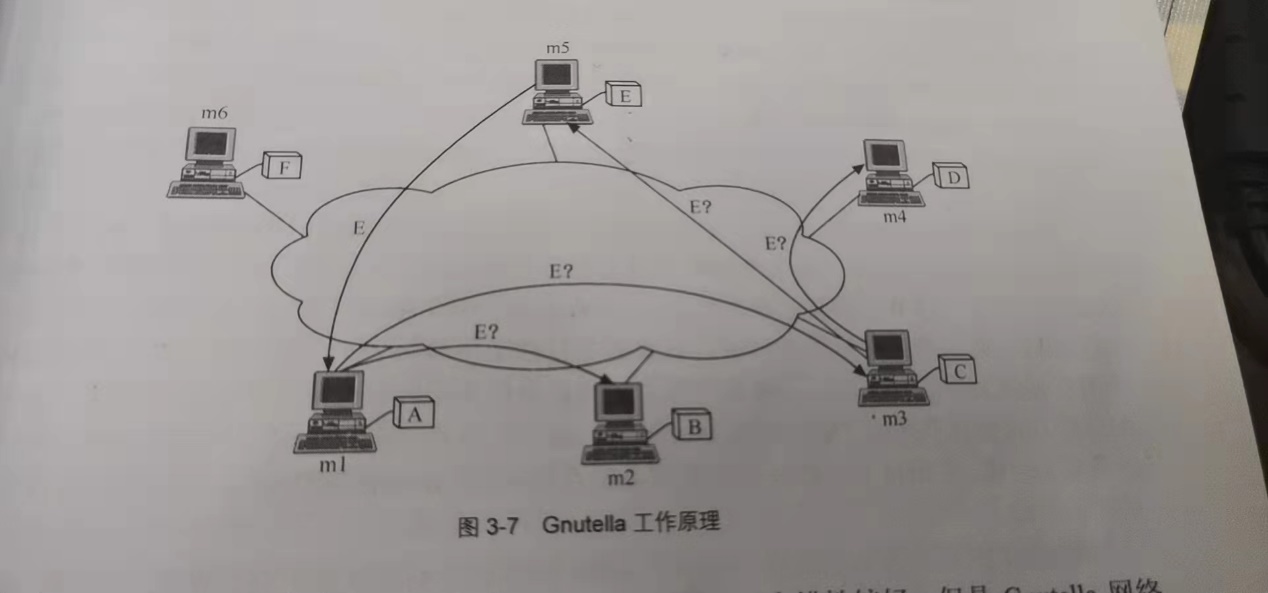
纯P2P非结构化网络模型也被称为广播式的P2P模型。由于没有专门的目录服务器，对等节点之间的内容查询和内容共享都是直接相邻节点广播接力完成的。

每个用户随机接入网络，并与自己相邻的一组邻居节点通过端到端连接构成一个逻辑覆盖的网络。每个节点的邻居都是随意、无规则选择的，信息资源在P2P网络中的存放位置也和网络本身的拓扑结构无关。没有一个节点可以知道整个网络的结构或者每个对等节点的身份，对等节点必须使用他们所在的网络来定位其他对等点。

当希望知道网络中另一个对等节点的位置时，该查询节点就会发出一个查询请求并直接广播到所连接的邻居节点，这些邻居节点会试图满足这个请求。如果不能满足这个请求，就会以相同的方式广播到他们的邻居节点(邻居的邻居)，以此类推。为防止搜索环路的出现，每个节点还会记录自己的搜索轨迹，直到收到应答或者达到最大步数(为避免系统无限循环而定义的检索级数，通常设置为5~9)，在收到应答之后，发起原始查询的终端即可直接向对等节点获取内容。

* 1. **纯P2P非结构化网络模型—****Gnutella模型**

Gnutella模型采用了基于完全随机图的泛洪(Flooding)发现和随机转发机制，通过IP多播技术让对等点定期发布资源和传播查询。不需要向中央服务器报告共享的信息，而是将请求泛洪到直接相连的邻居节点，直到收到响应，或者达到了最大泛洪步数。为了控制搜索消息的传输，通过TTL的减值来实现。



上述图说明了Gnutella的工作原理。假定m1已经通过发送ping/pong消息得知邻居是m2和m3，m3的邻居是m4和m5。当对等体m1希望查找资源E时，他首先向其邻居m2和m3发送请求，然后，若m2和m3没有资源E，则分别向m4和m5转发关于E的请求，如果m5发现符合此请求的资源E，则根据数据包内的信息将资源E直接发送给m1。

* 1. **纯P2P非结构化网络模型优缺点**

优点：①完全的分布式特性使之具有最大的容错性，不会出现单点崩溃现象；②能潜在的获得最多的查询结果

缺点：①整个网络的可扩展性较差，随着对等节点数量的增加，网络可能因为过多的查询消息而发生拥塞；②模型中没有中央目录服务器对用户进行管理，因此缺乏较好的集中控制和策略；③查询的有效性和正确性不能保证；④能力有限的对等节点容易成为系统瓶颈；⑤网络中对等节点的查询和定位比较复杂，效率低下。

* 1. **纯P2P结构化网络模型**

所谓的结构化和非结构化模型的根本区别在于每个节点所维护的邻居是否能够按照某种全局方式组织起来以便于快速查找。结构化P2P网络模型中，每个节点维护的邻居是有规律的，网络的拓扑结构也是受到严格控制的，信息资源将有规律地组织存放到合适的节点上。查询将会以较少的跳数，路由到负责所查询信息资源的节点上。

目前结构化P2P的主流方式是采用分布式散列表(DHT)技术，DHT在非结构化的P2P系统中加入了人为控制策略，把整个系统的工作重点放在如何有效查找信息上，通过分布式散列函数，将输入的关键字唯一映射到某一个节点上，然后通过路由算法同该节点建立连接。

DHT的主要思想是：首先，每条文件索引被表示成一个(key,value)对键值对，key称为关键字，可以是文件名(或文件的其他描述信息)的散列值，value是实际存储文件的节点的IP地址(或节点的其他描述信息)。所有的文件索引条目(即所有的键值对)组成一张巨大的文件索引散列表，只要输入目标文件的key值，就可以从这张表中查询出该文件的节点地址。网络中每个节点根据性能不同被分配维护部分散列表(具体分配方式根据不同的DHT算法有所不同)。这样，节点查询文件时，只要把文件关键字通过散列函数变为key值，再根据key查询到对应信息的存储位置，从而快速定位资源的位置。

* 1. **纯P2P结构化网络模型优缺点**

优点：①由于DHT各节点不需要维护整个网络的信息，而只是存储其相邻的后继节点信息，因此较少的路由信息就可以有效地实现到达目标节点；②取消了泛洪算法，利用分布式散列表进行定位查询，有效地减少了节点信息的发送数量，增强了P2P网络的扩展性；③大部分DHT总是在节点的虚拟标识与关键字最为近似的节点上复制备份冗余信息，也避免了单一节点失效的问题；④该模式下，使用者匿名，数据传输加密。

缺点：①维护机制复杂，节点频繁加入、退出造成的网络波动会极大地增加DHT的维护成本，若由于某种不可预知的原因导致节点出现故障，维护的成本更加昂贵。因此结构化P2P网络模型不适合于高动态性的网络环境；②DHT仅支持精确关键词匹配查找，无法支持内容/语义等复杂查询；③结构化P2P网络模型受到自身算法限制，不适合于超大型的P2P系统。

1. **分层式P2P网络模型**

集中目录式网络模型有利于网络资源的快速查找，但是中心化的模式容易遭到直接的攻击；纯P2P网络模型解决可抗攻击的问题，但是又缺乏快速搜索和可扩展性。所以，出现了分层式P2P网络模型，它吸取了集中目录式网络模型和纯P2P网络模型的优点，在设计和处理能力上都进行了优化，按节点能力的不同(计算能力、内存大小、连接带宽、网络滞留时间等)区分为超级节点和普通节点两类。在资源共享方面，所有节点的地位相同，区别在于，超级节点上存储着系统中其他部分节点的信息，发现算法也仅仅在超级节点之间转发，超级节点负责将查询请求转发给适当的普通节点。这样，在超级节点之间就构成了一个普通转发层，超级节点和普通节点构成了若干层次。

另外，我们还可以根据需要在各个超级节点之间再选取性能最优的节点，或者另外引入新的性能最优的节点作为更高一层超级节点，来保存整个网络中可以利用的超级节点的信息，并且负责维护整个网络的结构。

性能最好的机器作为超级节点，这些节点存储着离他最近的普通节点的文件信息，承载着部分服务器的任务(如管理部分普通节点、负责搜索消息的转发等)。这些超级节点再联通起来形成一个上层的overlay。由于超级节点的索引功能，使得搜索效率大大提高。

* 1. **分层式P2P网络模型——KaZaa**

KaZaa协议中每一个节点上线后都会寻找一个超级节点挂靠，并和原先挂靠在该超级节点下的其他普通节点随机相连，组成一个小的无结构网络。每个普通节点的共享文件索引汇报给所挂靠的超级节点。因此，KaZaa网络大体上可以看做是两层的无结构网络，上层是超级节点组成的无结构网络；下层是普通节点组成的多个无结构网络，按照所挂靠的超级节点分成多个簇。

当普通节点发起文件搜索请求时，将请求消息发给所挂靠的超级节点，超级节点从自己存储的共享文件索引信息中查找区域内符合条件的文件，同时将搜索请求转发给若干个其他超级节点，由他们返回区域内搜索结果。如果需要，这个转发过程可以执行多步以获得更大范围的搜索结果。

* 1. **查询机制**

一个或多个超级节点与其临近的若干普通节点之间构成一个自治的簇，簇内节点可以自治地进行消息查询；整个P2P网络中各个不同的簇之间，通过纯P2P的模式将超级节点相连起来进行消息查询。

根据簇的规模不同，簇内采用不同的查询机制：

* 簇内只有少量节点(比如几十个)。每个节点可以维护一个本地路由表，通过一致散列函数来分配和定位(key,value)对，使得查询簇内的其他任意一个节点的跳数为O(1);
* 簇内有比较多的节点(几百个)。利用超级节点进行查询。只要查询节点向簇内的超级节点发送查询请求，就可以在O(1)内查找到目的节点
* 簇内有大量节点(上千个)。在簇内使用Chord、CAN、Pastry或Tapestry这类算法。查询跳数将是O(logN)，N为簇内节点数。

假设簇g内有一个节点x要在网络中查询节点y，则查询步骤如下：

1. 节点x通过簇内协议在簇g中查找，若节点y在簇g内，则查找结束；若节点y不在簇g内，则由超级节点组成的上层overlay利用簇间查询协议进行查找；
2. 利用超级节点组成的上层overlay找到距离节点y最近的簇g’，由g’继续进一步查询；
3. 簇g’根据其内部协议查询到节点y，并把查询结果返回给节点x。
   1. **簇管理**

簇如何划分？超级节点如何维护？  
 若节点x要加入一个分层式P2P网络，假定x应被分配到ID值为k的超级节点下。首先，x用k与网络中已经存在的另一节点y取得联系，y将定位并返回对应于k的簇中超级节点的ID。如果整个返回的超级节点ID值恰好为k，那么x就作为普通节点加入该超级节点所在的簇，并将自己的CPU、带宽等信息告知该超级节点；如果返回的超级节点ID不是k，那么就会生成一个新的簇，这个簇只包含x一个节点。

在一个网络中，若每个簇有m个超级节点，那么最先加入簇的m个节点将成为超级节点。这些超级节点将监视着新加入簇的节点。同时维护着一个“超级节点候选名单”，一个节点在网络中持续时间越长，它所拥有的资源越多，那么就更容易被选为超级节点。名单会定期发送给簇内的普通节点。当一个超级接点失效或退出时，名单中的第一候选节点将成为超级节点，并加入上层overlay，并通知簇内所有节点和其他簇内的超级节点。这种机制使得上层overlay中的节点十分稳定，并且能够快速修复超级节点偶尔的失效或离开。

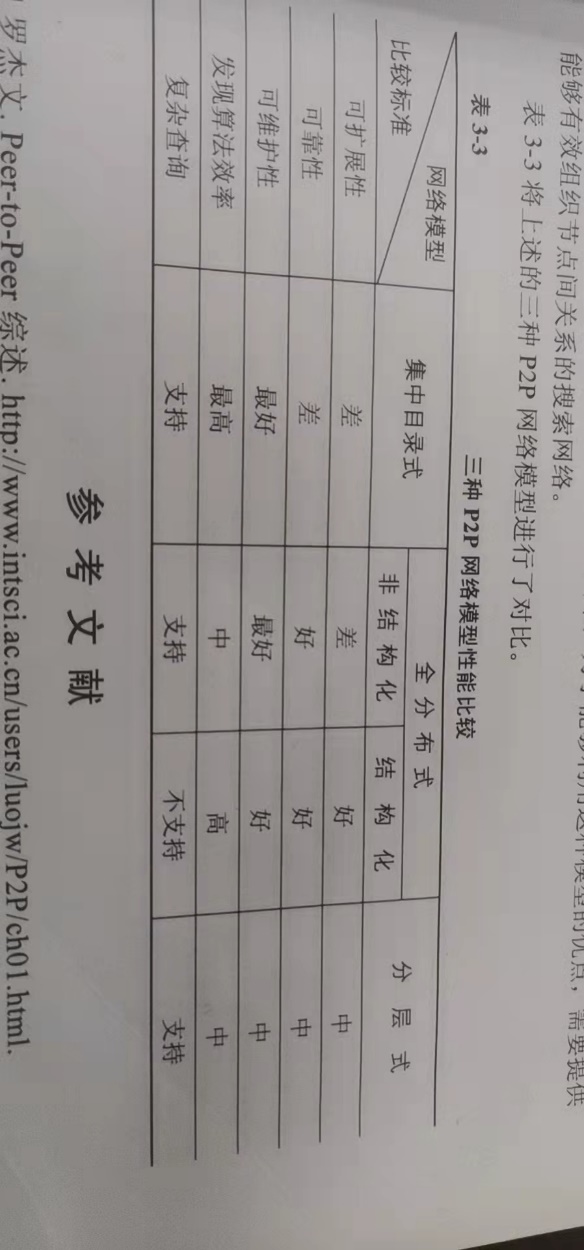
* 1. **分层式P2P网络模型优缺点**

**优点：**

1. 按照性能对节点进行分类。根据节点的能力合理分担负载，只有计算能力强、网络带宽高的节点能够成为超级节点，并承担簇内查询任务
2. 各簇相互独立。如果一个簇改变了其内部查询机制，这种改变对于其他簇和上层查询机制是独立的。同样的，一个节点的失效或加入，只对其归属的簇的路由表产生影响，而不会对其他簇造成影响
3. 提高了查询速度。
4. 减少了查询消息传播的数量。由于上层overlay的性能稳定，减少了重发消息数量

**缺点：**实现上比较困难，为了有效利用这种模型的优点，需要提供能够有效组织节点间关系的搜索网络。

1. **三种P2P网络模型对比**

****

第四章 P2P网络中的资源定位方法

1. **绪论**

P2P网络包括结构化的P2P网络和非结构化的P2P网络，这两类网络在资源定位上的差异很大。

* 非结构化的P2P网络中。资源的组织和管理相当松散，拓扑构造随意性大，资源对象的放置通常放置在本地，与P2P系统的拓扑结构无关。在进行资源定位时，资源定位请求被转发到各个节点上(一般采用泛洪或优化后的泛洪方法进行)，定位条件的具体匹配操作通常是在本地节点上进行的。由于是在本地进行，所以可支持多种资源查询方法，包括关键字和语义查询，高级的数据库查询等。
* 结构化的P2P网络(DHT网络)中，资源的放置于网络的拓扑结构紧密相关，资源和资源的索引按照自己的资源ID存放在网络中的特定节点上，在进行资源定位时，一般采用精确定位方式，即是采用资源ID利用DHT网络提供的算法进行资源的定位。当然通过对DHT算法进行扩展也可以支持一些其他的资源查询方法，包括关键字和语义查询等。

资源的定位可以进一步分为“资源搜索”和“资源查询”

资源搜索：特指查询请求的传播操作，非结构化的P2P网络中的资源定位。

资源查询：包括查询请求的传播以及获得该资源或是其索引的整个过程，结构化的P2P网络中的资源定位。

1. **资源搜索算法**

大致可以分为盲目搜索算法和启发式智能搜索算法。盲目式搜索算法是通过网络传播查询信息并把这些信息不断扩散到网络中的各个节点，采用泛洪或随机选择节点进行转发查询请求的方式在整个网络中搜索想要的资源。而启发式智能搜索在搜索过程中利用一些已有的信息来辅助资源的查找。

* 1. **盲目搜索算法**
     1. **泛洪搜索算法**

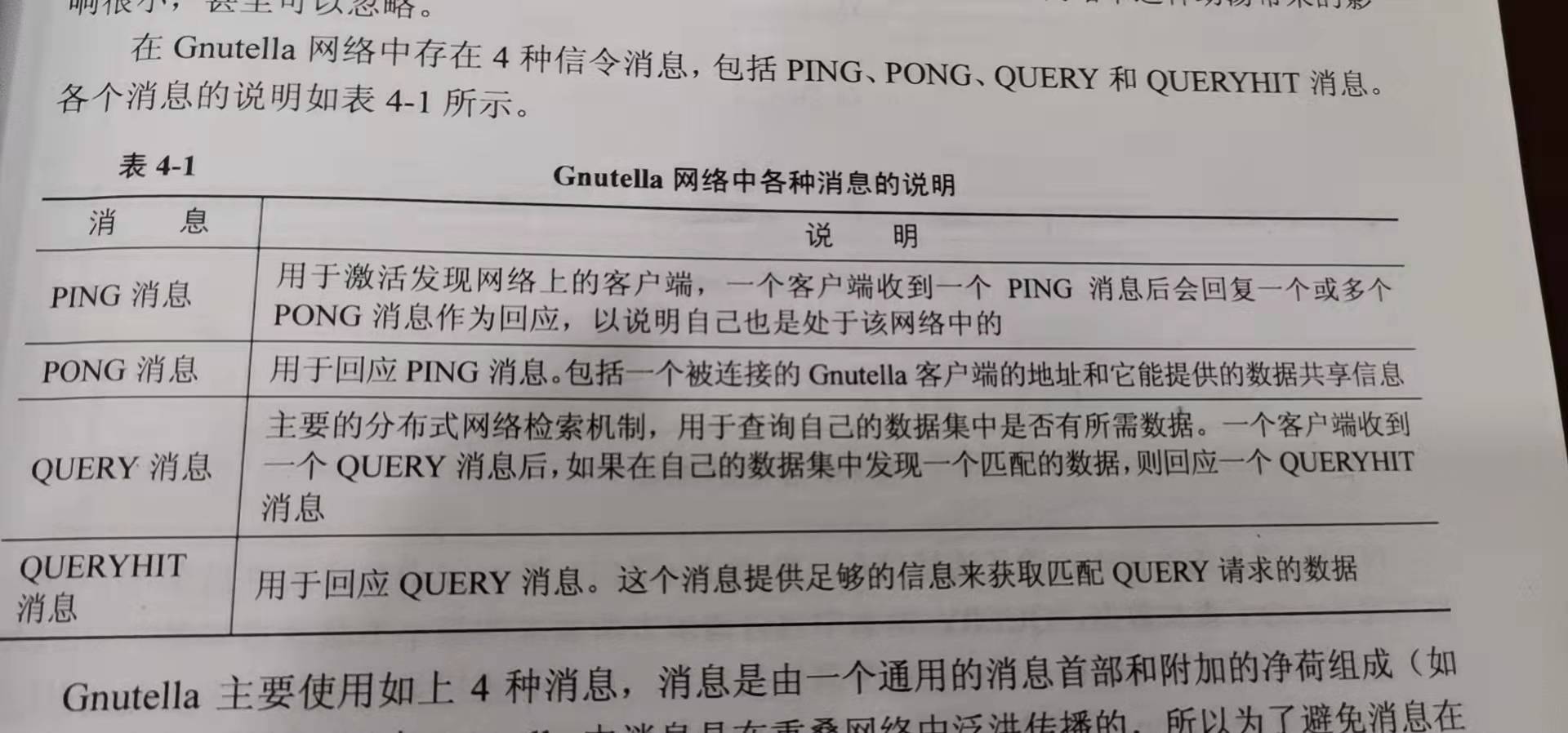
在网络中，每个节点都不知道其他节点的资源。当节点需要在网络中搜索某个资源时，该节点向他所有的邻居节点广播查询消息，接到查询请求的邻居节点包含所需资源时，就返回一个查询命中信息给发出查询请求的节点，否则邻居节点再向自己的邻居节点广播该查询请求，这个过程将不断持续下去。为了限制搜索的范围，发起查询请求的节点往往在消息中设置一个初始的TTL值，消息每经过一个节点，TTL值减1。当TTL=0时，对应的查询请求搜索过程终止。

以上的资源搜索方式使查询消息像洪水一样在网络中的各个节点流动，所以称为泛洪搜索。由于这种搜索时先遍历自己的临近点，然后再由其相邻点继续传播，所以又称为广度优先搜索方法(BFS)。

一条查询消息到达它的生命终点有两种可能：一是被判定为冗余消息(即该节点如果在短时间之前曾收到并转发过该查询消息，则判断该消息为冗余消息)；二是它的TTL值归零。在实际的泛洪搜索算法中采用上述两种方式均可，第一种可以避免循环，但需要额外的对一段时间的查询请求的缓存机制；第二种方式的实现比较简单，易于实现。

在Guntella 0.4中最早使用泛洪算法作为P2P网络中的消息分发算法。Guntella 0.4网络是一种纯P2P非结构化网络，由大量分布于世界各地的节点组成，没有任何中心单元。在整个网络中，节点在启动时会首先连接到一个中心实体，从而获得活跃的对等节点的信息，然后和这些活跃的节点建立连接从而加入到网络中，另外它会通过向已连接的节点发送PING/PONG消息来进一步获得更广泛的网络中的其他节点的信息(PING/PONG消息采用泛洪的方式进行)，在启动阶段完成后节点会保存已获得的路由信息，但不会对这些路由信息进行定期的维护。对于节点动态加入或者退出网络时会带来网络的动荡，由于网络中其他节点并不会因此而大范围的更新他们的路由信息，所以在Gnutella网络中这种动荡带来的影响很小，甚至可以忽略。

在Gnutella网络中存在4种信令消息，包括PING、PONG、QUERY、QUERYHIT消息。各个消息的说明如下表所示：



Gnutella主要使用上述4种消息，消息是由一个通用的消息首部和附加的载荷组成的。由于在Gnutella中消息是在重叠网络中泛洪转发的，所以为了避免消息在网络中被重复转发或陷入循环，Gnutella的消息结构中定义了描述符ID（Descriptor ID）来标识一个消息，当节点两次收到相同的消息时，将不会再进行转发，这样可以避免循环。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0-15 | 16 | 17 | 18 | 19-22 | 23-N+22 |
| 描述符ID  Descriptor ID | 净荷描述符（Payload Descriptor） | TTL | 跳（Hops） | 净荷长度（Payload Length） | 净荷(Payload)  N个字节 |

图 基本Gnutella消息结构

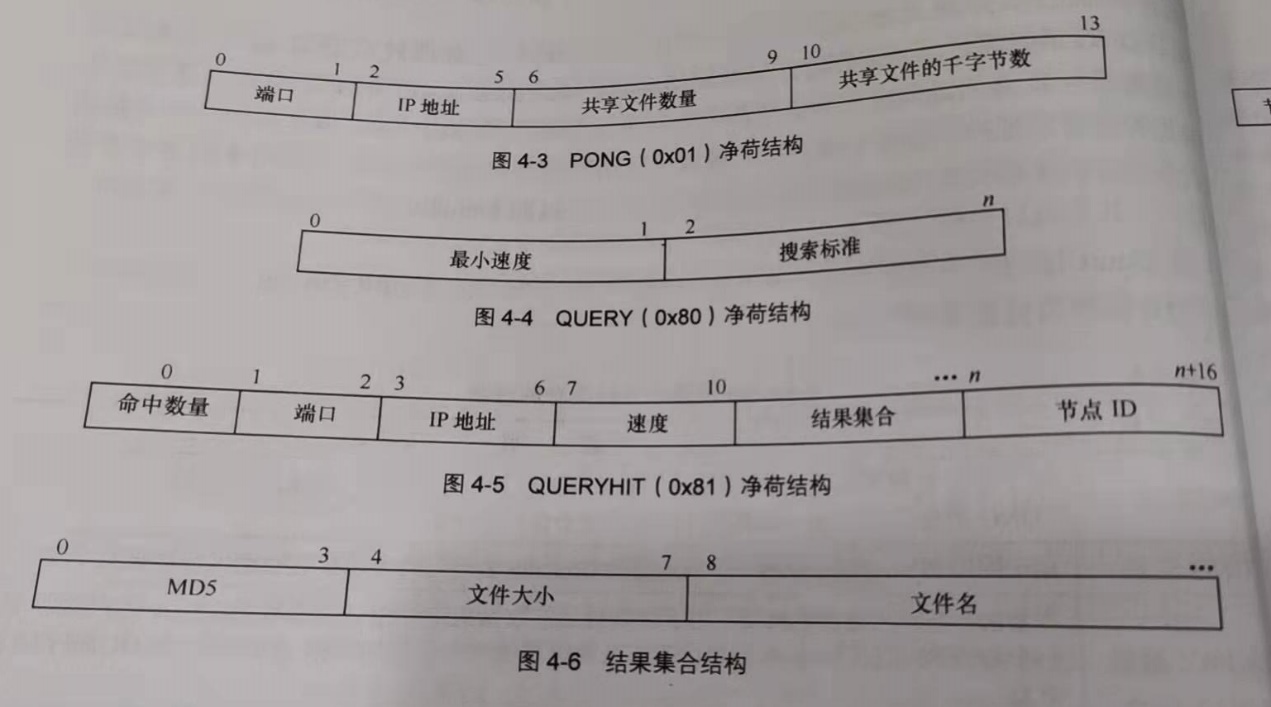
在每个节点都必须存储这个ID和IP地址(向该节点发出请求的节点的IP地址)并保持一段时间，这样就可以实现响应消息的返回路由。

<TTL>值确定了一条消息在重叠网中被转发的跳数，接收到这条消息的每个节点在转发消息之前将这个值减1.当TTL值到达0时，消息不再转发，这是为了避免在网络中无休止的泛洪同一条消息。通常情况下，TTL的值为7被认为是完全能达到参与重叠网中绝大部分节点的。

<Hops>值用于声明一条消息已经转发的跳数，并在每个转发的对等端加1.他能够用于确保一个对等端在对消息初始化时不会使用大于7的值，即TTL(0) = TTL(i)+Hops(i)≤7。

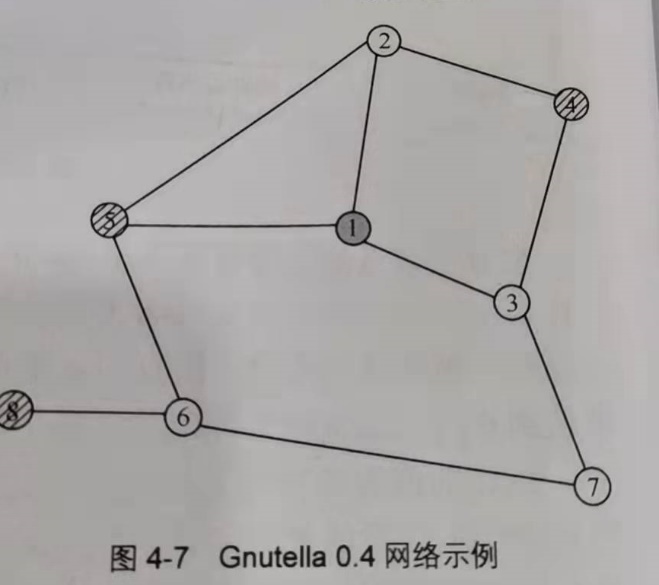
<Payload length>（净荷长度）声明消息的尺寸，以使该消息能够被清楚地识别。

<Payload Descriptor >字段是消息结构中最重要的字段，具体的：0x00表示PING,0x01表示PONG,0X80表示QUERY,0X81表示QUERYHIT。其中每种消息的净荷部分的结构是不同的，这和消息的目的有关。其中PING消息没有任何净荷，其他消息的净荷结构如下所示：



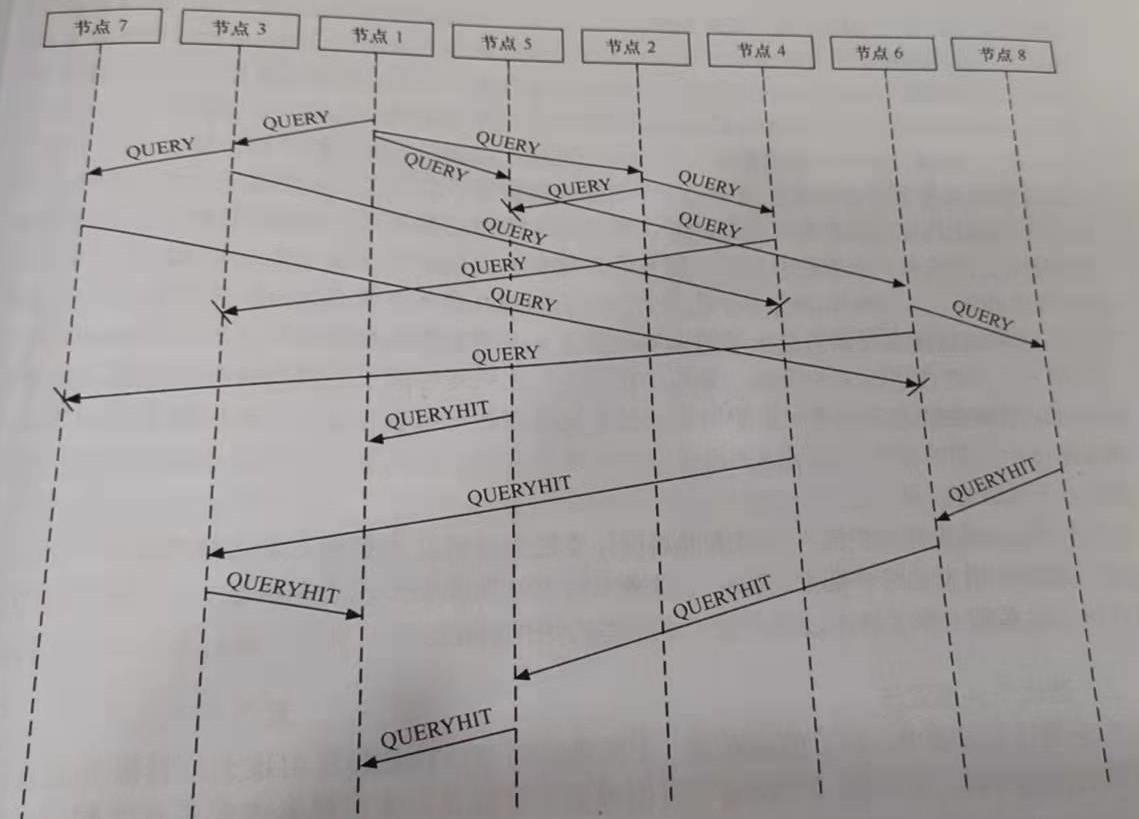
PONG消息的载荷中，除了连接信息(IP地址+端口)外，关于共享文件总量的信息被声明了。

QUERY消息中包含请求方所要求的最小下载速度参数，还包含了一个空结尾的查找字符串和一个空格分隔的关键字，它用于描述被请求的对象。这个消息的平均长度是78.4Byte。QUERY消息也是通过泛洪在重叠网络中传播。一旦一个节点收到一条QUERY消息，那么它会将查找关键字和描述本地共享内容的关键字进行比较。在至少命中一次的情况下，节点发回一条QUERYHIT消息，该QUERYHIT消息的结果集合中可以包含一个以上的文件，QUERYHIT消息较大，平均长度为748.8Byte。



在上图中的网络中，共有8个节点，其中只有部分节点建立起网络连接，现在以节点1为例来说明资源查询的过程。

假设节点需要查询资源X，在该网络中的8个节点中，只有节点4、节点5.节点8含有该资源。查询消息QUERY在节点上终止转发的条件是在该节点之前转发过此消息，即此消息为冗余消息(网络较小，不合适通过TTL来控制消息的生存时间的方式)。节点1首先向节点5、2、3发送一条QUERY消息进行对资源X的查询。节点5收到QUERY消息后，就会提取出其中的关键字信息，然后将其与描述本地共享内容的关键字进行比较，由于节点5存有该资源，所以它将向节点1发送QUERYHIT消息作为回应。然后节点5把查询请求转发给在网络中与它相连的节点2、6，其他节点也会进行相同的操作。如果有某一个节点已经收到并转发过此QUERY消息，那么它将直接舍弃此消息。



以上是Gnutella网络中资源搜索的过程。可以看出，当网络规模扩大时，这种泛洪查找带来的网络开销是很大的。仅在这样一个小网络中，一次搜索产生了12条QUERY消息(理想情况下)，假定网络中节点4、5、8回答了这条QUERY，这将产生6条QUERYHIT消息，这种消息的传输方式显然大于中心服务器方式的消息流量。

**泛洪搜索算法优缺点**

**优点：**回应时间短，覆盖范围广，高可靠性。在Gnutella网络中，采用泛洪方式，能够搜索到95%的节点(TTL=7),这是因为在泛洪算法中，随着跳数的增加，更多的节点会参与进来对查询请求进行转发，参与节点的个数呈现指数级增加。其次，在采用泛洪算法的P2P系统中，节点的加入、离开、失效等行为对于整个系统的消息转发和信息交换能力带来的负面影响很小，因为所有节点都担任着相同的角色，这也说明了泛洪搜索算法的健壮性。

**缺点：**带宽消耗严重。可扩展性差。随着联网节点的不断增加、网络规模的扩大，通过泛洪定位对等点的方式将造成网络流量急剧增加，从而导致网络中部分低宽带节点因为网络资源过载而失效，查询结果可能不完全，查询速度较慢，由此带来可扩展性差等问题。泛洪方式产生的消息大量为冗余消息，当同一查询消息的多份拷贝通过某节点的邻节点传达给该节点时，除了最先到达的消息是可用的，后续到达的皆为冗余消息。

* + 1. **迭代泛洪搜索算法**

在泛洪搜索算法中，TTL的设置是一个两难的问题：TTL设置过大，则很可能造成大量无所谓的搜索开销；TTL设置过小，搜索不易成功，使得搜索需要重复发起。迭代泛洪搜索法便是为了改善这一问题被提出的。

在迭代泛洪(Iterative Flooding)中，进行多次泛洪搜索，每次的深度限制(TTL值)是递增的。当查询结果满足要求或者已经到达最大的深度限制时，过程结束。

迭代泛洪搜索算法过程为：首先需要决定一个策略，说明每一次的迭代的深度。比如说：我们想要进行三次迭代，第一次的深度为a，第二次的深度为b，第三次的深度为c，我们的策略就是P={a，b，c}。因为迭代泛洪算法需要和深度为D的直接泛洪算法具有相同的搜索范围，所以最后一次迭代的深度值必须为D。同时还需注意，必须给出迭代之间的时间间隔W。

在迭代泛洪方法P={a，b，c}中，查询源节点S首先运行一个深度为a的广度优先搜索算发进行查询；发送TTL=a的请求消息给它所有的邻居节点。深度为a的节点接收到查询消息并进行处理时，这个节点将暂时存储查询消息。因此查询将在所有距离源节点有a跳的节点那里“冻结”。与此同时，源节点S对应答信息进行处理，如果已经找到查询结果，那么源节点S就什么也不做了，否则源节点S将开始进行下一次的迭代深入，下一次的深度为b。

在下一次广度优先搜索开始时，源节点S将发送一个TTL为a的Resend消息，收到Resend消息的节点仅仅把请求消息向前传递。如果一个节点处于深度为a的位置上，它将终止Resend消息继续向前传递，同时“解冻”相应的查询消息(Gnutella中每一条消息都有自己的ID，因此节点知道对哪一条消息进行解冻)，并发送TTL=(b-a)的Query消息给它的邻居节点。在深度为b处，查询仍然像深度为a的情况下运行。而且因为c是三次迭代的最大深度，所以查询不会在深度c上的节点处发生冻结保留。即使没有找到查询目标，源节点S也不会再启动另一个迭代了。

在好的情况下，迭代泛洪算法确实能减小进行查询操作的节点个数，从而减少搜索的资源开销。因为迭代不一定要以D为深度泛洪，只要以比D小的深度泛洪就可能得到满意的查询结果。随着深度的增加，节点个数是按指数增加的。迭代泛洪算法避免了以最大深度进行泛洪，因此大量减少了查询的节点个数。但是在坏的情况下，迭代泛洪算法可能需要进行最后一次迭代。这时，迭代泛洪算法就比直接泛洪算法更糟糕。因为多次迭代中网络上发生了大量的Resend消息，占用了网络带宽，也给节点增加了处理负担。因此只在实际泛洪深度小于制定的泛洪深度时，迭代泛洪要优于直接泛洪。

* + 1. **随机漫步搜索法**

由于泛洪算法带来大量消息开销，一些研究提出随机搜索技术来进行资源定位。标准的随机搜索是指各节点将搜索消息转发给随机选择的某一个邻居。

* K路随机转发方法

请求者发出K个查询请求给随机挑选的K个相邻节点。然后每个查询信息在系统中独立地进行随机转发，并在以后的搜索过程中周期性地与原始请求者保持联系，询问是否还要继续下一步。如果请求者继续漫步搜索，则又开始随机选择下一步漫步搜索的节点，否则终止搜索。

* 随机BFS方法

与K路随机转发方法不同的是，每个接收到搜索请求的节点都将搜索请求转发到多个随机选择的邻居节点上，直到搜索请求被满足或是TTL值为0。

* 总结

随机BFS方法的搜索延迟比K路随机转发要小，但会产生更多的消息开销。Christos等人采用随机过程等数学工具对随机搜索方法进行量化建模，理论分析指出随机搜索方法在至少以下两种情况下比泛洪搜索方法更加优越：

* 节点对于同一资源对象发出多次搜索请求，而两次搜索期间P2P系统的拓扑结构变化不大（即具有较强相关性）。
* P2P系统中存在节点聚集现象，即属于同一社区(物理上临近的区域)的节点间连接数较多，而处于不同社区的节点之间连接数较少。

一般来说，与泛洪搜索方法相比，各种随机搜索方法产生的消息开销更少，但搜索延迟较大。

* + 1. **区域索引法**

区域索引法也是对泛洪算法的一种改进算法。每个节点N上保存有一个数据索引表，该数据索引表记录了距离此节点为r跳以内所有节点上保存的数据的索引（r是系统设置的变量，表示索引半径）。当r=0时，就退化为广度优先方法，每个节点只有自己的的数据索引。当一个节点收到请求信息时，它能够在r跳以内每一个节点的索引中进行查询。用这种方法，只需要处理少量的节点就能够得到查询结果。所以此方法既能保障查询结果的数量，又能保持较低的开销。

在区域索引法中由系统规定了每个查询将被处理的深度，所以没有在这个深度列表上的节点只继续向他的邻居节点发送请求信息，而不对请求进行处理。例如，区域索引法有深度列表P={1,3,5}。查询源节点S将发送请求消息给深度为1的所有邻居节点，所有这些节点需要处理请求信息，同时把查询请求发送给所有深度为2的节点。深度为2的节点不会对这些查询请求进行处理，而是只将这些Query消息发送给深度为3的节点。深度为3的节点将处理这些Query消息，同时将其转发给深度为4的节点。深度为4的节点只负责将这些Query消息转发给深度为5的节点。深度为5的节点会负责处理这些Query消息，因为5是在深度列表P中的最后一个值，所以这些节点将在完成消息处理后结束这个查询。

区域索引法与迭代泛洪方法的不同：

* 迭代泛洪方法中，深度表示迭代将结束的深度，所有结束深度以内的节点都必须对查询消息进行处理
* 区域索引法需要创建和维护每个节点的索引，当有节点加入、离开或者更新它的数据时，需要采用额外的措施来更新索引信息：当节点x加入网络时，它发送一个TTL=r的加入信息，并且包含可共享的内容的路由表信息。当一个节点从节点x收到加入信息，它将发送包含自身共享数据的信息给节点x。这两个节点都将把接收到的共享内容信息加入到自己的路由表中。当一个节点离开网络，拥有此节点信息的其他节点将在一段时间后在路由表中删除此节点索引的路由表信息。
  + 1. **最大聚集度优先查询方法**

此种方法是利用P2P网络拓扑具有“小群体”特性进行的。每个节点包含3类列表，列表1记录本节点的共享信息文件目录表，称为“本地共享信息文件路由表”；列表2记录与本节点相邻的对等节点的邻居连接关系信息，称为“相邻节点聚集度列表”；列表3记录与本节点相邻的对等节点的共享信息文件路由表，称为“相邻共享信息文件路由表”。后两种信息列表可以通过在相邻节点间周期性地交换消息包来刷新，或者是当一个活动对等节点有新节点加入时发送这种消息包来刷新。因此，网络在启动一段时间后，在一群相邻的对等节点中，聚集度最高的(连接节点数量最多)的对等节点就会聚集了与自己相邻的对等节点具有的“本地共享信息文件路由表”列表信息。从而实现了高聚集度节点聚集有较多的共享信息文件路由表。同时，每个节点都知道与其相邻的节点的节点聚集度，这就意味着，每个节点都知道谁是它邻居节点中聚集度最高的节点。

Gnutella网络节点拓扑具有“幂规律”，因此其节点总有少数节点具有较高的“聚集度”，而多数节点的“聚集度”较低。如果其节点不具备此规律，则其网络要么是连通度很高、很规则的“规则网络”，要么是连通度较低、节点分布随机的普通“随机网络”，再就是处于两者之间的既不规则也不很随机的一般网络，这三种网络在使用“最大聚集优先”算法时不会提高路由查找效率。而对于“幂规律”的随机网络，网络中仅有少量的节点有较高的“聚集度”，多数节点的“聚集度”较低，“聚集度”较高的节点同其他节点的联系比较多，因此通过它找到待查信息的概率较高。因此，采用“最大聚集度优先”的路由查找算法，可以再有限步转发查找包后，即可到达高聚集度节点，从而遍历较大范围的节点共享信息。

利用此算法进行查找时，当一个节点需要进行信息查询时，它将首先查找自己的“本地共享信息文件路由表”，若命中就直接响应查询命中包(QueryHit)。若未命中，则查询自己的“相邻共享信息文件路由表”，若命中就直接与对应的相邻节点建立连接，进行所需信息的下载。如果上述两种都未命中，查自己的“相邻节点聚集度列表”，找到其相邻节点中聚集度最高的节点并向其转发请求消息。

另外，在利用消息包进行查询时，包中还包含一个“被转发的途径节点标识列表”域，每经过一个节点的转发，就将该节点的IP地址添加到该列表域中。当下一个节点收到该查询包时，首先检查“被转发的途径节点标识列表”域中是否有自己的IP地址，若有说明自己已经被访问过，则将查自己的“相邻节点聚集度列表”，找到相邻节点中聚集度最高的节点，将该查询包转发给它。若没有，则采用相同的方法查询“本地共享信息文件路由表”和“相邻共享信息文件路由表”，若命中，则相应一个查询命中包(QueryHit)；否则查自己的“相邻节点聚集度列表”，找到相邻节点中聚集度最高的节点，将该查询包转发给它。重复以上过程，直到查询包的TTL域值减为0，或找到了具有待查共享信息的节点。

* + 1. **基于超级节点的查询方法**

Gnutella 0.6结构的一个主要目标是降低Gnutella 0.4网络中的高的消息负担，这些消息负担也是影响Gnutella 0.4网络可扩展性的重要因素。降低这种带宽消耗的有效方式是引入层次的概念。在Gnutella 0.6中引入了超级节点（SuperPeers）和叶子节点（LeafNodes）的概念。

每一个超级节点不应该具有超过100个叶子节点，这依赖于超级节点的处理能力和连接速率。超级节点用于存储离它最近的叶子节点的文件信息，这些超级节点再联通起来形成一个P2P网络。当叶子节点需要查询文件时，首先从它所连接的超级节点的索引中查询，如果找到了文件就直接根据文件所在的IP地址与对应主机节点建立连接；如果没有找到，则SuperPeers把这个查询请求发给它所连接的其他超级节点，直到得到想要的资源。

在Gnutella 0.6中保留了Gnutella 0.4中定义的所以消息类型，即PING/PONG/QUERY/QUERYHIT消息。同时为了让叶子节点将其共享的内容向其连接的超级节点宣告，采用了一种特殊的消息ROUTE\_TABLE\_UPDATA(0X30)，消息格式如下：



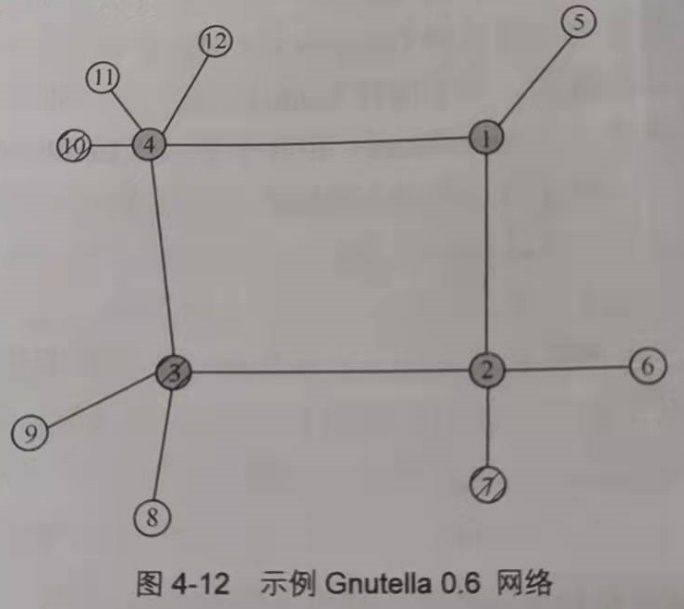
其中，<Variant>参数用于标识一条ROUTE\_TABLE\_UPDATA消息究竟是用于复位（Reset）还是用于更新（Update）。

Reset消息用于清除接收端的路由表（此处的接收端指的是超级节点），因此还需要额外声明要请求删除的表长度。参数<Infinity>暂未使用。

Update消息用于上传并在超级节点处设定一个新的路由表。为了避免传送一个很巨大的表，可以将一个路由表最多分为255块，具体的块数可以通过（Seq No）设定，目前已使用的块的最大数量由<Seq Size>声明。参数<Compressor>表示压缩方法，用此压缩方法对路由表进行压缩。参数<DATA>包含一系列32位长的关键字散列值，它是由叶子节点共享的对象的关键字进行散列得到的。这些散列值相互拼接在一起，并作为一个数据串进行传输，或者是被分成更小的块进行传输。一条ROUTE\_TABLE\_UPDATA消息的平均长度是269字节。

通过叶子节点传来的路由表，超级节点可以确定哪条QUERY需要被转发到其旗下的哪个叶子节点上。仅当QUERY消息中的关键字和路由表中的一项相匹配时，才会把这条QUERY消息转到给叶子节点。

只有各个超级节点之间会保持联系，查询消息时只要在超级节点之间进行，必要时会转发给合适的叶子节点，这样相当于大大减少了直接参与到网络中的节点数目，网络规模可以认为是之前的几十分之几。整个网络的流量也呈现指数级下降，这样就明显增加了系统的可扩展性，也增加了资源搜索的成功率，减小了搜索的时延。



上图中，节点1~4位超级节点，节点5~12位叶子节点，每个叶子节点和一个超级节点相连，而超级节点根据自己的能力不同可以和多个叶子节点保持联系。

在此网络中，假定叶子节点5发起查询请求，需要查询某一资源，而在这个网络中节点3、节点7、节点10对此资源进行了共享，整个查询过程如下：节点5先向他连接的超级节点1发送查询请求，然后节点1把查询请求转发给节点2和节点4。节点4在查询自己的路由表时，发现叶子节点10包含此资源，因此他把查询请求发给节点10，然后进一步把查询请求转发给节点3。节点3在本地进行查询，发现自己有此资源，它会向节点4返回QUERYHIT消息。节点2的操作也以此类推。具体的消息传播过程如下图所示：



由于超级节点的引入，使得一条查询请求所产生的的转发消息在网络中的数目大量减少。此网络中12个节点的规模，一条查询请求只产生了7条QUERY消息，相比之前8个节点的Gnutella 0.4网络中则要产生12条QUERY消息。

* 1. **启发式智能搜索算法**
     1. **启发式泛洪**

启发式泛洪中，节点不是向所有的邻居节点都发送查询请求，而是选择其中一部分过去表现优秀的邻居节点。这是基于这样一个假设：过去表现优秀的节点将来也会更加优秀。节点为了选择邻居，需要对每一个邻居节点的信息进行统计。比如说统计邻居节点在过去的查询中返回的结果个数，或者统计邻居节点在过去的查询中响应的时间等。基于这些统计结果，我们就能选择其中表现最为优秀的节点，比如：

1. 选择在以前的查询中返回结果最多的邻居节点
2. 选择在以往的查询中响应时间最快的邻居节点
3. 选择在线时间最长的邻居节点

启发式泛洪只是在转发节点的选择上相比于普通的泛洪进行了优化，使其更具有针对性。但是这样只选择部分节点进行转发损失了消息查询的全面性，可能对于某些稀缺资源的查询会出现查不到的情况，但毫无疑问，它对于带宽的减少是明显的。还需要注意的是，采用启发式泛洪也会产生额外的开销，即节点需要定期和邻居节点进行交互，以便获得和更新自己所保存的邻居节点的数据，这一行为增加了节点的处理和存储开销，也增加了带宽的消耗。

启发式泛洪能够大大减少参与查询的节点个数。另外，启发式泛洪选择的是以往表现优秀的节点，因此查询的结果数量、查询的响应时间等在很大程度上都能得到保障。但是启发式泛洪假设的是过去表现优秀的节点将来也会表现的优秀，显然这个假设并不是永远成立的。如果资源恰好存放在过去表现不好的节点上，采用启发式泛洪就可能搜索不到资源。

* + 1. **移动代理式**

所谓移动代理就是一个智能代理(Intellgent Agent)的软件程序，它可以从网络中的某一主机迁移到另一主机，还可以与其他的代理相互通信，并与网络中的分布式系统做沟通和交互。移动代理可以被派遣至远程的服务器上观察并且回报某一事件的发生情况：可以到远程的服务器上要求服务器做搜索并取得某些资料或者直接到远程的服务器上执行所需的服务，也可以与其他的移动代理做互动与会谈，已达到分工合作的目的。

和一般的网络应用程序相比，由于移动代理是将客户端与服务器端之间的交互工作和计算端的计算功能(并非资源)包装起来直接送到远程主机执行的，使得客户端不需要持续地与服务器保持联机，使得网络流量大大减少，处理效率提高，因此特别适合用来处理大量资料的应用和分布式计算。

移动代理和P2P相结合，可以通过在P2P软件中嵌入代理的运行环境。当有节点需要搜索时，它就会发送一个代理给它的相邻节点，移动代理记录着它的一些搜索信息。当这个代理到达一台新的机器上时，就会在这个机器上进行资源搜索任务，如果这台机器上没有所需的资源，该主机就会把这些搜索的信息传给它的邻居节点；如果找到资源，则返回给请求的机器。

利用移动代理来进行节点发现的大致过程如下(在节点上具体的资源查询方法由代理采用的主算法而定)：

1. 在某节点产生一个移动代理，该节点即为代理的创建点，代理将创建点的地址作为主地址，并通过创建节点的相关信息更新自身。同时，确定代理漫游时间参数a（往往需要根据代理返回创建点前所能访问的最多的节点数来确定）以及在任一节点该代理能够被复制的次数参数b。这两个参数用来控制网络搜索的深度和广度以及最大的代理访问节点数。
2. 首先给予移动代理一些网络中的其他节点地址，要求提供的节点的地址数应该小于b
3. 代理自我复制足够多次，使得代理能被派遣至步骤②中所确定节点
4. 到达每一节点后，移动代理的漫游时间参数a开始计时，根据代理本身携带的创建点信息更新这一新到达的节点，并根据当前节点更新移动代理。如果两个来自于同一创建点的代理相继到达该节点，则第二个代理自动销毁。这样可以使得在保持信息更新的同时避免来自于同一创建点的代理对节点的重复处理，减少网络负担。
5. 如果参数a到期，则移动代理自动返回创建节点，根据在漫游中得到的所有节点信息来更新创建节点的信息。如果参数a还未到期，则代理继续自我复制足够多次并将其派遣至当前节点的所有邻居节点。通过排除先前已知的节点，访问的范围逐渐减少，直到最后完成资源的发现任务。
6. 重复步骤④

基于移动代理的P2P资源发现方法和目前一般的P2P定位方法相比，可以大大减少对带宽的要求。在一般的P2P网络中，每个节点通过分布式协议在相互之间建立通信通道后，所执行的交互往往导致网络拥塞。而基于移动代理的资源发现方法则可以使这些交互，包括所有需要的数据统统包装起来直接送到远程主机上执行。所以当一个移动代理到达某一节点时，它本身已经携带了所有执行必须的数据，而不再需要与其他节点通信了。

移动代理特别适用于资源处理的应用，在传统的P2P系统中需要先发现数据，然后再将资源数据传回源节点进行处理，而移动代理可以直接在某节点处理好原始数据，再直接将结果返回给创建节点，这样可以避免资源传递带来的开销。移动代理可以迅速地复制并派遣到网络中的任意节点。在传递过程中即使一些代理被销毁，其他的代理也可以继续处理，保证了资源发现任务可以被最快速的完成。

移动代理还具有异步性，当一个移动代理被派遣至远程主机执行资源搜索任务时，创建节点对应的主机可以继续执行其他的操作而不必等待，甚至不用与之保持联机状态。移动代理具有自治性，这是一个非常适合网络资源发现的性质。移动代理可以具备学习能力，随着不断派遣处理过程而不断学习。比如，移动代理可以访问其开始时并不知道的节点并将其加入历史记录，在以后的漫游中进而根据历史记录与当前节点情况做出自主判断，大大提高资源发现的效率。

* + 1. **高速缓存法**

在高速缓存法中不管是超级节点还是一般节点都存有资源查询的路径信息（即之前资源搜索结果信息），这就是高速缓存器的思路。新的搜索并不需要直接到达资源的存储地，只要在搜索的路径中找到以前的搜索记录，就可以把请求消息返回。这样可以大大减少搜索的消息，提高效率。

但是该算法需要在节点上保存资源查询的路径信息，对于网络中有大量资源的情况，缓存空间的大小和机器的能力限制都限制了该方法的使用。如果缓存空间设置的过小，则缓存的命中率会很小；若缓存空间设置的过大，又会增加节点的处理负载和存储负载。

因此高速缓存方法可以再比较稳定的集中部署的节点上使用，这样在一定程度上可以提高搜索的效率，同时不增加其他节点的负载，减小搜索带来的网络流量。

1. **资源查询算法**

资源查询主要是对某个节点根据资源搜索中包含的信息对该资源进行定位的操作，查询分为两类：精确匹配查询和语义查询。精确匹配查询是指根据资源的key值进行快速的匹配查询；语义查询即根据文档的关键字等语义信息对该资源进行定位。

在非结构化的P2P网络中，由于它一般采用泛洪或是优化后的泛洪方法，所以这种P2P网络天然可以支持多种资源查询方法，包括关键字和语义查询，以及更加高级的数据库查询等。因此对于非结构化P2P网络的查询方法不再重点介绍。

在结构化的P2P网络(一般为DHT网络)中，一般采用精确匹配查询的方法，即根据资源的key值进行快速的匹配查询。但缺点就是不能将信息和该资源的关键字、数据库的键值等相关联，应用比较有限。但现在某些DHT网络也可以采用语义查询，例如把语义信息当做资源在网络中进行存储。这样可以快速地得到一类资源。

关于精确匹配的查找可以参考DHT算法部分，下面着重介绍结构化P2P系统中采用的语义查询。

* 1. **关键字查询**
  2. **向量空间模型的检索方案**
  3. **隐含语义的检索方案**

第五章 结构化P2P系统的算法

1. **分散式散列表(DHT)**

DHT(distributed hash table,分布式散列表)，也称分布式哈希表，是一种分布式存储方法。在不需要服务器的情况下，每个客户端负责一个小范围的路由器，并负责存储一小部分数据，从而实现整个DHT网络的寻址和存储。

**什么是哈希（Hash）？**

Hash，即哈希，也称散列，就是把任意长的输入通过散列算法（MD5、SHA-1等），变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，并且是不可逆的。通过这种方法可以把不同空间的输入散列到同一个空间中，方便查询和管理。

**什么是散列表（Hash table）？**

“散列表”是用来存储“键值对”的一种容器。“键值对”即“key/value pairs”，简称“K/V”。散列表的作用就是：如果有了“散列表”，就可以很方便快速地通过 key 来获得 value。

　　举个例子：

　　手机通讯簿可以通俗理解成一个“散列表”。里面的每一条记录都包含“姓名”和“电话号码”。“姓名”相当于“键值对”中的 key，电话号码相当于 value。你可以通过姓名方便地查找出电话号码。

**什么是分布式散列表（Distributed Hash table）？**

“分布式散列表”在概念上类似与传统的“散列表”，差异在于——“传统的散列表” 主要是用于单机上的某个软件中；“分布式散列表”主要是用于分布式系统。

在DHT网络中客户端主机成为节点，存储的数据项称为对象。ID空间是系统的一个标识域，在这个ID空间中所有的标识符都是独一无二的。标识符用来标识节点和对象，每个节点和对象都有唯一的标识符。在DHT系统中，节点标识符可以通过对一个节点的名字(一般是节点的IP地址)进行散列获得；对象标识符可以通过对资源对象的名(文件名)进行散列来获得。

在DHT网络中，每个节点存储的是资源对象的<key,value>对。每个资源对象都有自己的关键字(一般长度不一，格式不一),通过对这些关键字进行散列之后可以得到资源对象的关键字key，而value值就是资源对象真实的存储位置(该资源的在主机中的存储位置或目录)，key与value的关系是(key = f(value))。需要注意：每个key对于的value与节点标识符不是一个东西。Value是资源对象具体的存储位置(深入到主机下的文件夹、文件)，节点标识符仅仅是对主机节点名字(IP地址)的散列（后者是散列值，前者不是散列）。

这样子，整个DHT网络就形成了一张巨大的<key,value>的文件索引散列表。每个节点都按照某种规则只维护一部分散列表（这里的规则依DHT算法的不同而不同）。用户在查询资源对象时，就是将资源对象关键字再次进行散列获取其资源关键字key，通过DHT算法路由获得到该key在整个DHT网络中的真实存储位置value(匹配查询)。

每个节点在加入网络的时候，会对存储在本节点上的资源进行索引并建立索引表，以满足本地资源检索的目的。然后该节点会按照某种预定的规则选择一些节点作为自己的邻居节点，加入到P2P网络中。

当网络中某节点发起查询请求q，它会将q发送给自己的邻居节点。它的邻居节点收到这个查询请求q之后，它会检查本身是否有查询的信息，如果不存在，继续转发该查询请求q，直到返回结果。

当一个节点离开P2P网络时，它会利用DHT算法通知网络内有关的节点，并将自身负责的资源通过算法内部的某种机制让网络中其他节点负责，使得整个网络能够正常地维护。另外，在DHT网络中，还有探测和容错机制，以保证系统的可用性。

1. **DHT功能及实现**

分布式散列表（DHT）的主要功能包括3项：①标识符的生成和管理；②提供重叠网络中的查询定位路由服务；③对提供的服务或文件的信息进行管理。

标识符的生成包括节点标识符ID和资源关键字key的生成。节点加入DHT网络时，将该节点的信息（例如IP）进行散列得到全局唯一的标识符作为节点标识符ID，资源发布到DHT网络时，该资源的信息（例如名称）进行散列得到全局唯一的标识符作为资源关键字key。而标识符的管理指的是节点在动态加入和退出DHT网络时系统的处理方式，具体处理方式根据算法的不同而存在差异。

提供覆盖网络中的查询定位的路由服务是DHT的基本功能，也是现有的DHT架构的热点研究问题。由于整个DHT是按照一定的组织形态存储在不同的网络节点上，某一个节点只负责存储一部分。而当某一个节点需要用到其他节点的资源时就必须在这个由众多DHT节点构成的网络中查询所用到的资源的具体位置。在查询时，资源请求会让节点查询自身所存储的路由表，如果目标节点具体信息在路由表中，则直接完成查询过程。如果没有，则向相应的目标节点(比如邻居节点)发送查询请求，接收到查询消息的节点运行与发起查询的节点相同的查找过程，直到完成目标节点的定位。

1. **DHT的分类**

按照路由复杂度可以将DHT算法分为Muti-hop DHT、O(1)DHT、One-hop DHT三类

* 1. **Muti****-hop DHT**

多跳的DHT算法，主要指在一个N节点的网络中，系统的路由跳数为O(logN)，且每个节点连接到O(logN)个邻居。在该类DHT算法中，节点用于维护节点间信息变化的消耗较低，但在路由查找时时延很大。当系统规模很大时，节点数量可能超过百万，这样的查询效率显然难以满足使用的需要。即使用户能够忍受漫长的时延，查询过程中产生的大量消息也会给网络带来不必要的负荷。而且当节点抖动较强烈时，对网络拓扑的影响也比较大。

此类典型的算法如：Chord、Pastry、CAN、Kademlia

* 1. **O(1)DHT**

平均路由跳数为常数。这类算法的特点是路由复杂度不随网络规模的变化而变化。在该类DHT算法中，为保持一定的路由跳数，需要对数据和节点信息使用相同的机制(如复制机制)，从而导致需要移动资源，系统同时产生了额外的开销，但是相比Multi-hop DHT算法路由的时延减小了很多。

典型算法：Kelips、Beehive等

* 1. **One-hop DHT**

每个节点将存储全局的路由表，从而实现单跳查询。为了确保路由的正确性，每个节点维持的路由表再节点信息变化时都需要及时进行维护。在动态环境中，节点的加入和退出是很频繁的，因此会导致路由维护的代价消耗很大。

此类算法也是O(1)DHT算法的一种，但是是比较特殊的一种。

1. **DHT算法—Chord**

在Chord算法中，通过使用散列函数(Hash)，将每个节点的信息进行散列得到唯一的节点标识符（Node ID）,同时对每一个资源的信息（资源名）进行散列得到唯一的key值。对于每一个资源对象的关键字key，Chord选择将关键字存储到相应的节点上，由这个节点来存储该对象的<key,value>对。而Chord中关键字key与节点的存储映射关系是将关键字key存储在节点标识符（Node ID）等于或大于key值(Node ID ≥ key)的第一个节点上。当需要查找资源时，采用同样的方法，先通过散列运算得到资源的key值，然后根据key与节点的映射关系，可以快速定位到存储资源的节点上。

在Chord系统中，节点并不需要知道网络中其他所有节点的信息，而仅需要保存部分节点的路由信息，通过与其他相邻节点进行通信从而完成路由查询。

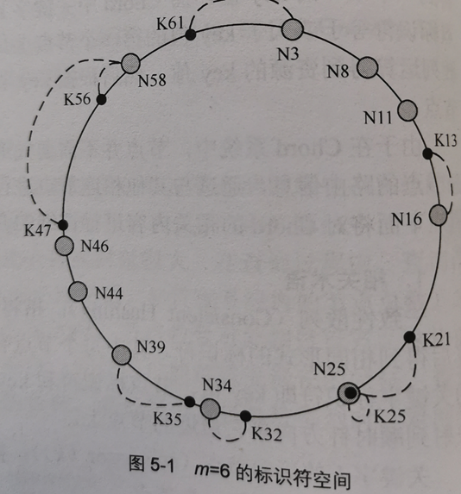
* 1. **相关术语**
* 一致性散列（Consisten Hashing）：是指将节点信息（如IP地址）和资源信息经过散列运算后得到相同形式的标识符（Node ID）。其中每个节点有其唯一的节点标识符，每个资源内容有其唯一的关键字标识符key值(key值由其相应的value散列得到)，节点标识符和key值位于同一值域。在Chord算法中，将关键字key映射到顺时针方向离他最近的节点上。
* 关键字k的后向节点（successor（k））：指的是存储关键字k的节点，该节点的节点标识符（Node ID）是等于或大于k的第一个节点
* 关键字k的前向节点（predecessor（k））：指的是在Chord环中关键字k的逆时针方向上的第一个节点
* 节点的后向节点（successor）：一般指某个节点的后向节点，即Chord环中某节点在顺时针方向上的第一个节点
* 节点的前向节点（predecessor）：Chord环中某节点在逆时针方向上的第一个节点
* 后向节点列表（successor-list）：一般指某个节点的r个后向节点组成的列表，即Chord环中的某个节点在顺时针方向上最近的r个连续的节点组成的列表。
  1. **标识符空间**

每个参与P2P网络的节点信息通过散列函数（典型的如SHA-1）映射成为m bit的唯一的节点标识符。同时，Chord采用同一个散列函数，把每个资源的信息(value)映射为m bit的关键字标识符，即key值。由此得到的节点标识符和关键字标识符在数值上都是[0,2^(m-1)]区间内的整数。比如IP地址为192.168.0.5的节点经过SHA-1散列运算后得到的标识符为123，而“LetItBe”散列运算后的key值为60。将所有的节点和资源按照它们的标识符值从小到大按顺时针方向排列并形成一个环形的标识符域，称为标识符空间，也成为了Chord标识符环。

在散列运算时，散列函数Hash将输入参数（如节点名字、资源名称）进行映射，生成一个高度随机的数值。因此节点标识符和关键字标识符的分布不是连续的，在Chord环上体现为，有些位置是存在节点，而有些位置为空。同时，由于节点和关键字共用一个标识符空间，标识符长度m必须足够长，这样才能保证节点或资源的标识符的唯一性。

Chord系统中，每个关键字都保存在它的后向节点中，也可以说，每个节点只负责保存以他为后向节点的关键字的信息。由此实现了信息的分布式存储。

下图是一个Chord标识符环，假设标识符比特数为m = 6，则标识符空间大小为2^6=64，即标识符的取值范围是[0,63]中的整数。给定的资源有8个，首先对资源进行散列运算得到关键字标识符，用K表示，分别为K13、K21、K25、K32、K35、K47、K56、K61；节点有10个，用N表示对节点信息进行散列运算得到的节点标识符，分别为N3、N11、N16、N25、N34、N39、N44、N46、N58。



标识符按顺时针方向从小到大的顺序组织起来形成一个环。关键字标识符K13在Chord环顺时针方向的下一个节点（即后向节点）为N16，则由N16保存K13的信息；同理，在顺时针方向上的节点标识符数值大于21的第一个节点为N25,由N25保存K21的信息。特殊的，K25的后向节点正是与它的关键字标识符在数值上相等的N25,K25的信息由N25存储；由于Chord的环形结构，K61的后向节点为N3。

* 1. **Chord路由**

在分布式环境中，在最简单的情况下每个节点只需要知道它在Chord标识符环上的后向节点是谁即可进行定位操作。我们可以设想，一个最简单的查询应该是这样一个过程：当在节点n上查找一个资源时，首先将资源关键字进行散列得到key值，然后判断该key值是否在节点n与其后向节点（successor）之间。如果是，返回当前节点的后向节点作为查询结果；如果不是，则将查询信息发送给节点n的后向节点，再在n的后向节点与它的后向节点上进行相同的操作，一直往下找到保存该关键字的节点。

以上的这种方法虽然只要保证后向节点信息正确就可以保证查找的正确性，但是查询速度比较慢，可能要遍历网络中所有的节点才能找到目标。因此，为了缩短查找过程，Chord中的每个节点都需要维护额外的路由信息，每次都将查询请求转发到节点所知的离请求关键字最近的节点上，实现关键字的快速定位。

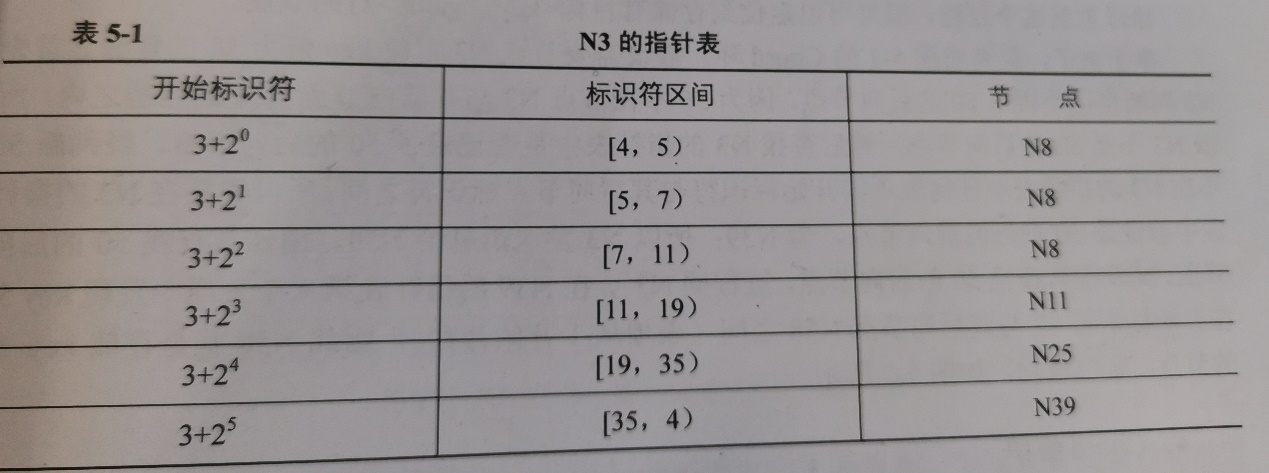
下面详细介绍Chord节点维护的路由信息和资源查找路由过程。

* + 1. **节点维护的信息**

同样的，关键字标识符和节点标识符都用m bit表示。每个节点n需要维护3个信息：前向节点（predecessor）；后向节点列表（successor-list）；一张最多有m条信息的指针表（finger table）。其中前向节点和路由表信息在路由过程中发挥了重要作用，而后向节点列表用于维护Chord环的稳定。

节点n的指针表包括m个表项，每个表项包含3个内容：开始标识符（start）、标识符区间（interval）、节点（node）。节点n的指针表中的第i个表项是这样构造的：开始标识符，为（n+）mod；标识符区间，为第i项与第i+1项的开始标识符之间的区域（包含第i项的开始标识符，但不包括第i+1项的开始标识符）；节点，为该表项的开始标识符在Chord环中顺时针方向上的下一个节点，性质类似于后向节点，表示为s=successor（n+），s称为节点n的第i个指针。这里1≤i≤m，且所有项都必须对取模运算。

仍以图5-1为例，给出的节点N3的典型指针表如下表所示。整个指针表共有6个表项。第一个表项的开始标识符为3+=4；标识符区间为[4,5） ;节点是以4为标识符的后向节点，为N8。



指针表的这种结构有两个重要特征。第一，每个节点都只需要存储少量其他节点的信息，且对在Chord环上离自己较近的节点的信息了解得比离它远的节点更少。第二，单个节点的指针表没有存储足够的信息来直接决定任意一个标识符的后向节点。例如，节点N3无法依靠自己存储的信息来直接得到标识符为13的后向节点，因为其后向节点N16没有出现在节点N3的指针表里。

当一个节点n并不知道关键字标识符k的后向节点该怎么办？如果节点n能够在自己的指针表中找到比自己的节点标识符更加靠近k的节点，那么这个节点会比自己更加了解k的信息。根据这一特性，n将查找它的指针表，找到节点标识符大于关键字标识符的第一个节点j，然后向j询问key k的后向节点是谁，通过重复这样的操作，最终能够找到k的后向节点。

* + 1. **资源查找过程**

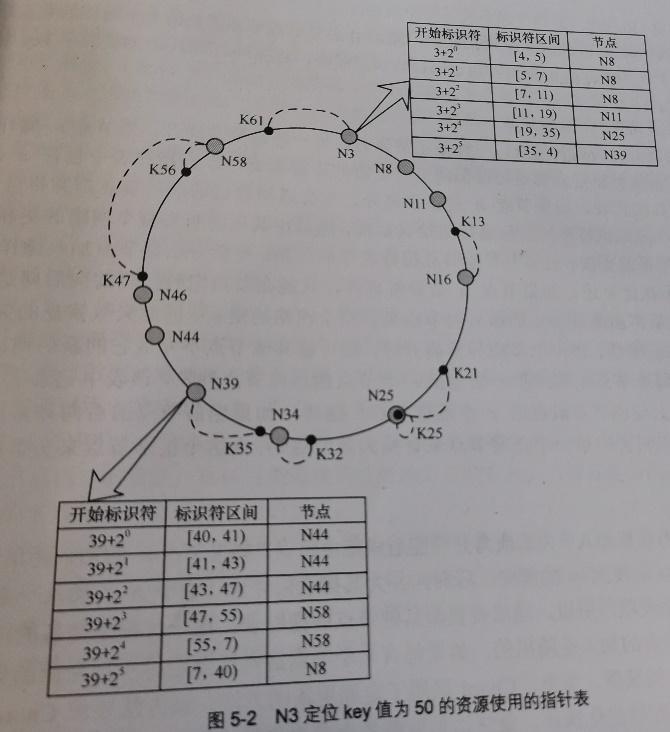
当一个节点n收到查询某个资源的请求时，首先对这个资源的信息进行散列运算，得到关键字标识符key，假设为k，然后在P2P网络上查找k的后向节点，找到后向节点就相当于找到了资源。由于每个节点都维护了其前向节点、路由表信息，节点在查找k的后向节点时，分为以下几个步骤：

1. 判断这个k值是否落在n与n的前向节点之间，如果是，则节点n就是该资源的后向节点，否则转②
2. 查找节点n的指针表，在指针表中找到k值所属的标识符区间，在判断这个key值是否在该表项的开始标识符和其后向节点之间，如果在，那么这个表项中的后向节点就是存储目标<key,value>对的节点，查找结束，否则转③
3. 在指针表中查找k的前向节点，即逆时针方向上离k最近的节点。首先在节点n的指针表的节点一列中查询节点标识符最大但是不超过k的节点，该节点是根据节点n的指针表确定的k的前向节点，并发送查询请求消息给这个前向节点，通知其查找k值的后向节点，转①。

通过重复这个过程，最终可以定位到存储有目标<key,value>对的节点。

举例，仍考虑图5-1的Chord环。假设需要节点N3定位key值为50的资源。

首先，N3判断自己是否是50的后向节点，因为50不在节点N3与其前向节点N58的范围之内，所以N3不是50的后向节点；然后查找N3的指针表中是否记录了50的后向节点，经判断50不在N3的指针表中任何表项的开始标识符与其后向节点标识符之间；所以需要在N3的指针表中查找距离50最近的前向节点，为N39；所以N3发送消息给N39，通知其查找50的后向节点。N39继续查找50的后向节点，过程同N3，在N39的第4个表项中发现key 50 在开始标识符47与其后向节点N58之间，因此N58就是存储key 50 的节点，定位成功，如下图所示。



* 1. **Chord维护**

在动态网络中，节点可能随时加入或离开，为了保证网络中能够准确地对资源进行定位，Chord需要保证两个不变的量：第一，每个节点都有正确的后向节点；第二，每个key值都需要由正确的后向节点维护。第三，为了实现快速查询，指针表也必须保证正确。因此，Chord维护的主要目标就是保证这三者的正确性。

Chord环中的每个节点都需要维护节点的前向节点信息，一个作用是帮助路由查询，另一个重要作用就是它可以简化节点的加入和离开机制。

* + 1. **节点加入**

在Chord中，新节点n加入时必须完成以下3个步骤：

1. 初始化节点n的前向节点、后向节点列表、指针表信息
2. 更新已经存在的其他相关节点维护的信息
3. 转移关键字信息。其他节点将所有关键字键值应归属于n的关键字key转移到节点n上，这些key对应的数据项应该由n来维护。

具体来说，当一个新节点n加入Chord环时，首先需要确定Chord环上的一个已经存在的节点n’，通过节点n’对n进行初始化并将其加入到自己的网络中，具体过程如下：

（1）初始化节点n的前向节点、后向节点列表、指针表信息。节点n’首先会查询获得节点n的后向节点，记为节点s；获取后向节点s的当前前向节点作为n的前向节点；向s请求它的后向节点列表，将s后向列表的前（r-1）项和s作为节点n的后向节点列表成员；构造n的指针表

（2）节点n加入以后，更新已经存在的其他相关节点维护的信息。相关节点包括n的前向、后向、逆时针方向的前r个节点以及那些在节点n加入以后必须更改路由表的节点。

更新n的前向、后向节点的过程为：修改n的后向节点的前向为n，修改n的前向节点的后向为n。

更新n的前r个节点的后向节点列表的过程为：将原后向节点列表中保存的从s开始的所有节点向后移动一位，移出的一位从后向节点列表中删除，节点n作为后向节点列表中的一项放在原节点s的位置上。

更新当n加入以后必须改变的路由表节点的过程较为复杂。当且仅当以下两个条件都成立时，节点n将成为节点p的第i项中的后向节点：（a）p先于n至少，即p+≤n；（b）p的第i项，即开始标识符为（p+），它的后向节点为n。将满足这两个条件的p的相应项更新为n。

很高的概率下，一个节点加入网络所需要更新的节点数是O(logN)，找到并更新这些节点需耗时O((logN)^2)。

（3）转移关键字信息。节点n加入以后，原本由n的后向节点s维护的key对应的数据项应该由n来维护，因此节点n需要和自己的后向节点联系，将应该属于n的关键字key转移到节点n上。

* + 1. **节点离开**

节点的离开分为两种情况，一种是正常离开，另一种是异常离开。当节点n离开之后，为了保证在系统重新建立稳定之前仍能进行正常的资源查询，那些指针表中包含n的节点需要更新其指针表。

如果节点n是正常离开的，它会通知自己的后向节点，并将自己负责存储的那一段标识符空间的信息拷贝给其后向节点，然后由其后向节点负责整个网络的更新，包括了前向后向关系的更新、网络上其他节点指针表项的更新，更新的方法是节点加入操作时的逆向过程。

如果节点n是异常离开的，这种情况只能在节点定时检查其前后向节点是否存在时发现，然后再由离开节点的原后向节点负责整个网络的更新。因此失效恢复的关键点就在于确保后向的正确性。当一个节点异常离开时，很可能其他节点不知道它的新的后向节点将会是谁，为了避免这种情况，每个节点上都会维护一张包括r个节点的后向节点列表，该表中记录了Chord环顺时针方向上与该节点最近的r个连续节点。这样，如果当前节点的后向节点异常退出，该节点就用后向列表中第一个正常节点来替换失效节点，并由这个正常节点来负责网络的更新（因为它是异常退出节点的原后向节点）。

* + 1. **稳定化**

Chord节点的频繁加入、失效或离开可能造成路由错误。如果节点n的加入是单一的，n可以通过已知节点n’找到n的前向、后向，并为其构造指针表。由于n的加入一定会影响到某些节点的路由表项，所以，随后要更新其他节点的指针表。这个过程是完备的，但是在网络环境中，节点的加入时随机的，如果每有一个节点加入就需要进行一次全面的更新，系统的消耗将很大，影响查找的速度。为此，Chord采用定期更新的方法。该方法要求Chord环中的节点必须周期性进行稳定化操作，最终达到更新各个节点指针表中错误表项的目的。

当已存在的节点n进行稳定化操作时，它首先向Chord网络中任意一个已知节点n’查询获得n的后向节点，然后请求n的后向节点返回它的前向节点p。如果p=n，则表明n和它的后向互相承认作为对方的前向和后向；如果p位于n和n的后向节点之间，则表明p是刚刚加入到网络中的，节点n将其后向节点更新为p，p将以n作为其前向节点。然后采用节点加入时的方法对网络中其他节点维护的信息进行更新。

* 1. **Chord算法特点**

在一个节点数量为N的稳定的Chord网络中，每个节点需要维护O(logN)个其他节点的路由信息，路由复杂度为O(logN)。当节点加入或离开网络时，Chord需要传递O((logN)^2)条消息来更新路由消息。

除了以上路由性能上的特点，Chord具有以下优势：

负载平衡：Chord采用分布式散列函数，所有节点以同等的概率分担系统负荷，从而避免了某些节点负载过大的问题，此算法本身就保证了负载的平衡性。

分布性：Chord是一个全分布式系统，所有节点完全平等。这一特点不仅使Chord系统具有很好的健壮性，而且更适合于松散的P2P应用系统。

可扩展性：Chord系统的路由开销随着系统规模（节点总数为N）的增加以O(logN)的速度增长，因此即使是很大规模的系统也是切实可行的。且获得这种可扩展性不需要进行任何参数的调整。

有效性：Chord能够根据节点的加入或离开自动地更新路由表，即使系统状态在不断变化，仍能够确保节点查询可靠地运行。

灵活的命名：Chord没有对关键字设置任何的限制，也就是说Chord的关键字是扁平的。在应用程序给关键字设置自己所需要的名字时具有很大的灵活性

同时，Chord也存在一些问题：

1. Chord网络的绕路（Detouring）问题。Chord在构建网络时没有考虑节点的实际物理拓扑结构，忽视了覆盖网络与底层网络的差异，可能导致在实际网络中相距很近的两个节点通过散列映射在覆盖网络上相距很远。这就是Chord网络的绕路
2. 未考虑节点的性能差异。Chord网络不加区分地赋予网络中所有节点同样的责任，这可能导致高性能的节点没有充分发挥其能力，而相对性能较差的节点却担负着无法担负的责任。
3. 指针表信息冗余。由于Chord算法构造指针表的方法和Chord环形结构的特点，使得信息冗余在指针表中不可避免。显然，在Chord节点的指针表中，随着i的增大，标识符区间范围成2倍增长。因此当i较小时，标识符区间也较小，其中的节点数可能很少甚至没有，导致节点指针表的前几行的表项内容重复。
4. **DHT算法—Kademila**
   1. **概述**

Kademlia是一种点对点分布式哈希表（DHT），它在容易出错的环境中也具有可证明的一致性和性能。使用一种基于异或指标的拓扑结构来路由查询和定位节点，这简化了算法并有助于证明。该拓扑结构有一个特点：每次消息交换都能够传递或强化有效信息。系统利用这些信息进行并发的异步查询，可以容忍节点故障，并且故障不会导致用户超时。

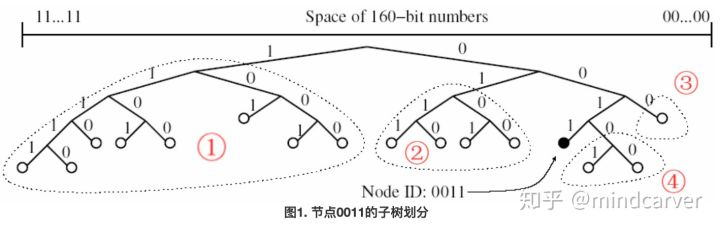
* 1. **节点状态**

节点的基本属性包括如下：

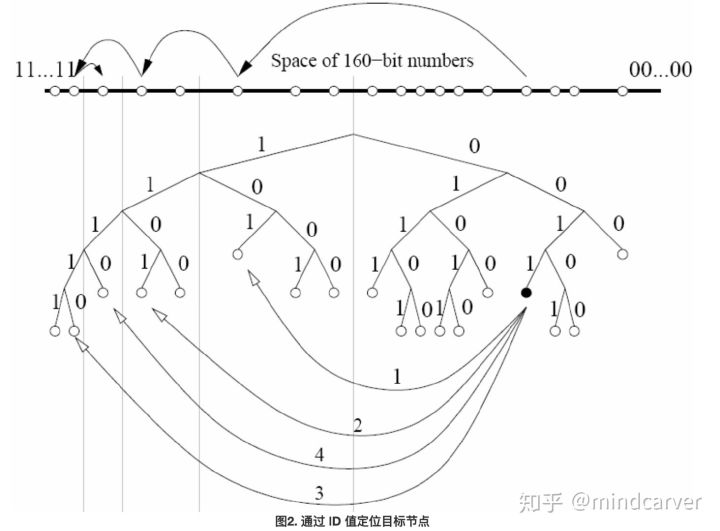
* 节点ID，Node ID
* 节点IP地址与端口号

在 Kad 网络中，所有节点都被当作一颗二叉树的叶子，并且每一个节点的位置都由其 ID 值的最短前缀唯一的确定。

对于任意一个节点，都可以把这颗二叉树分解为一系列连续的，不包含自己的子树。最高层的子树，由整颗树不包含自己的树的另一半组成；下一层子树由剩下部分不包含自己的一半组成；依此类推，直到分割完整颗树。图 1 就展示了节点0011如何进行子树的划分：



虚线包含的部分就是各子树，由上到下各层的前缀分别为0，01，000，0010。Kad 协议确保每个节点知道由其分割的各子树的至少一个节点，只要这些子树非空。在这个前提下，每个节点都可以通过ID值来找到任何一个节点。这个路由的过程是通过所谓的 XOR（异或）距离得到的。图 2 就演示了节点0011如何通过连续查询来找到节点1110的。节点0011通过在逐步底层的子树间不断学习并查询最佳节点，获得了越来越接近的节点，最终收敛到目标节点上。



需要说明的是:只有第一步查询的节点101，是节点0011已经知道的，后面各步查询的节点，都是由上一步查询返回的更接近目标的节点，这是一个递归操作的过程。

* 1. **节点距离**

Kad 网络中每个节点都有一个 160 bit 的 ID 值作为标志符，Key 也是一个 160 bit 的标志符(在系统中使用160bit来对对象进行标识是为了减小标识符或键值相同的可能性)，每一个加入 Kad 网络的计算机都会在 160 bit 的 key 空间被分配一个节点 ID（node ID）值（可以认为 ID 是随机产生的）， <key,value> 对的数据就存放在 ID 值“最”接近 key 值的节点上。

判断两个节点 x,y 的距离远近是基于数学上的异或的二进制运算， d(x,y)=x⊕y ，既对应位相同时结果为0，不同时结果为1。例如：

010101

XOR 110001

----------

100100

则这两个节点的距离为 32+4=36 。显然，高位上数值的差异对结果的影响更大。对于异或操作，有如下一些数学性质：

两个节点间的距离是随机的

节点与自身的距离是0

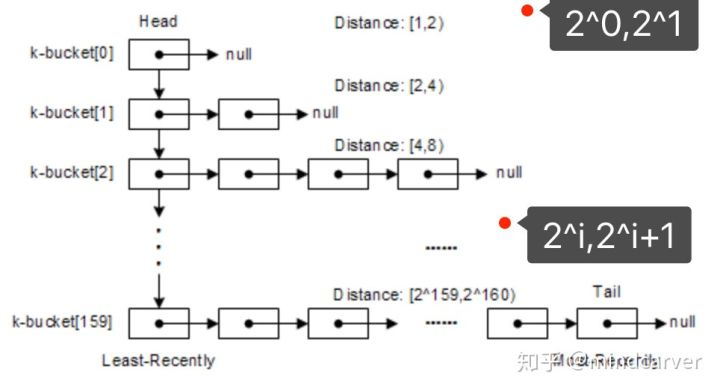
对称性:A 到 B 的距离和 B 到 A 的距离相等

三角不等:distance(A,B)+distance(B,C) <= distance(A,C)

对于任意给定的节点 x 和距离 Δ≥0 ，总会存在一个精确的节点 y ，使得 d(x,y)=Δ 。另外，单向性也确保了对于同一个 key 值的所有查询都会逐步收敛到同一个路径上，而不管查询的起始节点位置如何。

* 1. **K桶**
     1. **K桶的概念**

Kad 的路由表是通过一些称之为 K 桶的表格构造起来的。每一个节点一般包含有160个K桶，第i个K桶存储了距离本节点距离为的节点的属性信息(IP地址，UDP端口号，节点ID)。并且每个 K 桶内部节点的存放位置是根据最后联系时间进行排列的，最近联系的节点在K桶的尾部(Tail)，最早联系的节点在K桶的头部(Head)。一个节点的全部 K 桶列表如下图所示：



Kad规定：每个桶都有不超过 k 个的数据项。当 i 值很小时，K 桶通常是空的（也就是说没有足够多的节点，比如当 i = 0 时，就最多可能只有1项）；而当 i 值很大时，其对应 K 桶的项数又很可能会超过 k 个（当然，覆盖距离范围越广，存在较多节点的可能性也就越大），这里 k 是为平衡系统性能和网络负载而设置的一个常数，但必须是偶数，比如 k = 20。在 BitTorrent 的实现中，取值为 k = 8。

由于每个 K 桶覆盖距离的范围呈指数关系增长，这就形成了离自己近的节点的信息多，离自己远的节点的信息少(因为i值大的K桶最多也就只能存储k条消息，多余的距离比较远的节点可能没地方存储)，从而可以保证路由查询过程是收敛。因为是用指数方式划分区间，经过证明，对于一个有 N 个节点的 Kad 网络，最多只需要经过 logN 步查询，就可以准确定位到目标节点。

* + 1. **K桶的更新机制**

当接收者节点 x 收到一个 PRC 消息时，发送者 y 的 IP 地址就被用来更新对应的 K 桶，具体步骤如下：

1. 计算自己和发送者的距离： d(x,y)=x⊕y ，注意：x 和 y 是 ID 值，不是 IP 地址
2. 通过距离 d 选择对应的 K 桶进行更新操作
3. 如果 y 的 IP 地址已经存在于这个 K 桶中，则把对应项移到该 K 桶的尾部(按联系时间排序)
4. 如果 y 的 IP 地址不存在于该 K 桶中，有以下两种可能性：
5. k桶中节点的数量少于k个(未满)，将节点A添加到k桶的底部。
6. 如果k桶已经满了，向k桶头部节点z发送ping

* 如果 z 没有响应，则从 K 桶中移除 z 的信息，并把 y 的信息插入队列尾部
* 如果 z 有响应，则把 z 的信息移到队列尾部，同时忽略 y 的信息

K 桶的更新机制非常高效的实现了一种把最近看到的节点更新的策略，除非在线节点一直未从 K 桶中移出过。也就是说在线时间长的节点具有较高的可能性继续保留在 K 桶列表中。所以，通过把在线时间长的节点留在 K 桶里，Kad 就明显增加 K 桶中的节点在下一时间段仍然在线的概率，这对应 Kad 网络的稳定性和减少网络维护成本（不需要频繁构建节点的路由表）带来很大好处。

这种机制的另一个好处是能在**一定程度上防御 DOS 攻击**，因为只有当老节点失效后，Kad 才会更新 K 桶的信息，这就避免了通过新节点的加入来泛洪路由信息。

为了防止 K 桶老化，所有在一定时间之内无更新操作的 K 桶，都会分别从自己的 K 桶中随机选择一些节点执行 RPC\_PING 操作。

上述这些 K 桶机制使 Kad 缓和了流量瓶颈（所有节点不会同时进行大量的更新操作），同时也能对节点的失效进行迅速响应。

* 1. **协议消息**

Kademlia 协议包括四种远程 RPC 操作：PING、STORE、FIND\_NODE、FIND\_VALUE。

1. PING 操作的作用是探测一个节点，用以判断其是否仍然在线。

2. STORE 操作的作用是通知一个节点存储一个 <key,value> 对，以便以后查询需要。

3. FIND\_NODE 操作使用一个 160 bit 的 ID 作为参数。本操作的接受者返回它所知道的更接近目标 ID 的 K 个节点的 (IP address, UDP port, Node ID) 信息。这些节点的信息可以是从一个单独的 K 桶获得，也可以从多个 K 桶获得（如果最接近目标 ID 的 K 桶未满）。不管是哪种情况，接受者都将返回 K 个节点的信息给操作发起者。但如果接受者所有 K 桶的节点信息加起来也没有 K 个，则它会返回全部节点的信息给发起者。

4. FIND\_VALUE 操作和 FIND\_NODE 操作类似，不同的是它只需要返回一个节点的 (IP address, UDP port, Node ID) 信息。如果本操作的接受者收到同一个 key 的 STORE 操作，则会直接返回存储的 value 值。

为了防止伪造地址，在所有 RPC 操作中，接受者都需要响应一个随机的 160 bit 的 ID 值。另外，为了确信发送者的网络地址，PING 操作还可以附带在接受者的 RPC 回复信息中（**在上述 4种操作中 接受者回复 发送者时，可以携带上 接受者对 发送者的 PING, 以此校验 发送者是否还健在**）。

* 1. **路由查找**

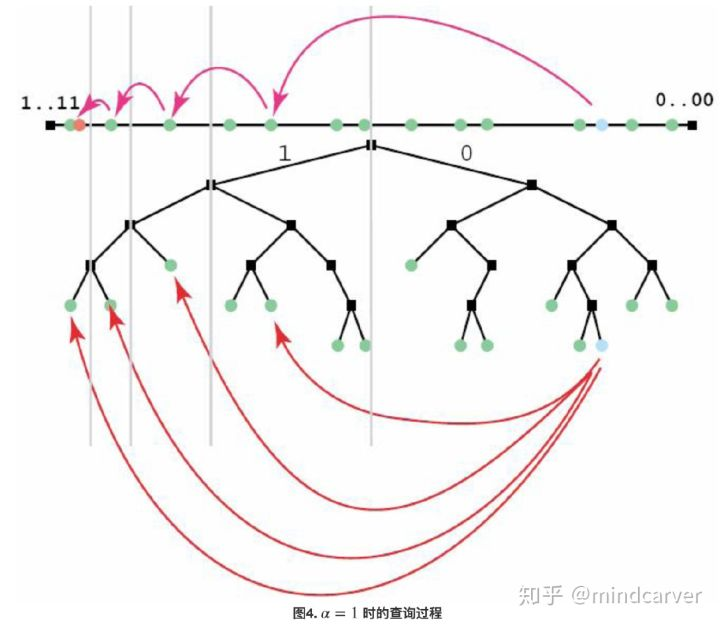
Kad 技术的最大特点之一就是能够提供快速的节点查找机制，并且还可以通过参数进行查找速度的调节。

假如节点 x 要查找 ID 值为 t 的节点，Kad 按照如下递归操作步骤进行路由查找：

1. 计算到 t 的距离： d(x,y)=x⊕y
2. 从 x 的第 [] 个 K 桶中取出 α 个节点的信息（“[ ]”是取整符号），同时进行 FIND\_NODE 操作。如果这个 K 桶中的信息少于 α 个，则从附近多个桶中选择距离最接近 d 的总共 α 个节点。
3. 对接受到查询操作的每个节点，如果发现自己就是 t，则回答自己是最接近 t 的；否则测量自己和 t 的距离，并从自己对应的 K 桶中选择 α 个节点(距离t最近的)的信息给 x。
4. X 对新接受到的每个节点都再次执行 FIND\_NODE 操作，此过程不断重复执行，直到每一个分支都有节点响应自己是最接近 t的(如果t真实存在于网络中，最后的一次结果必然是只有t本身对x进行了回复)。
5. 通过上述查找操作，x 得到了 k 个最接近 t 的节点信息。

注意：这里用“最接近”这个说法，是因为 ID 值为 t 的节点不一定存在网络中，也就是说 t 没有分配给任何一台电脑。

这里 α 也是为系统优化而设立的一个参数，就像 K 一样。在 BitTorrent 实现中，取值为 α=3 。当 α=1 时，查询过程就类似于 Chord 的逐跳查询过程，如图 4。



整个路由查询过程是递归操作的，其过程可用数学公式表示为：

|  |
| --- |
| N0=x (即查询操作的发起者) N1=find ⎯noden0(t) N2=find ⎯noden1(t) ... ... Nl=find ⎯nodenl−1(t) |

这个递归过程一直持续到 Nl=t ，或者 Nl 的路由表中没有任何关于 t 的信息，即查询失败。由于每次查询都能从更接近 t 的 K 桶中获取信息，这样的机制保证了每一次递归操作都能够至少获得距离减半（或距离减少 1 bit）的效果，从而保证整个查询过程的收敛速度为 O(logN) ，这里 N 为网络全部节点的数量。

当节点 x 要查询 <key,value> 对时，和查找节点的操作类似，x 选择 k 个 ID 值最接近 key 值的节点，执行 FIND\_VALUE 操作，并对每一个返回的新节点重复执行 FIND\_VALUE 操作，直到某个节点返回 value 值。

一旦 FIND\_VALUE 操作成功执行，则 <key,value>会被缓存在没有返回 value 值的最接近的节点上。这样下一次查询相同的 key 时就会更加快速的得到结果。通过这样的方式，热门 <key,value>的缓存范围就逐步扩大，使系统具有极佳的响应速度( 缓存的**cache 可以存活24小时**，但是目标节点上的内容会**每1小时**向其他最近节点重新发布使得数据的超时时间得以刷新，而远离目标节点的节点的数据存活时间当然就可能不会被重新发布到，所以也就是数据缓存的超时时间和节点的距离成反比)

* 1. **数据的存储**

存放 <key,value> 对数据的过程为：

1. 发起者首先定位 k 个 ID 值最接近 key 的节点
2. 发起者对这 k 个节点发起 STORE 操作
3. 执行 STORE 操作的 k 个节点每小时重发布自己所有的 <key,value> 对数据
4. 为了限制失效信息，如果在接收不到更新消息的情况下，相应的 <key,value>数据会在初始发布24小时后过期

另外，为了保证数据发布、搜寻的一致性，规定在任何时候，当节点 w 发现新节点 u 比 w 上的某些 <key,value> 对数据更接近，则 w 把这些 <key,value> 对数据复制到 u 上，但是并不会从 w 上删除。

* 1. **节点的加入和离开**

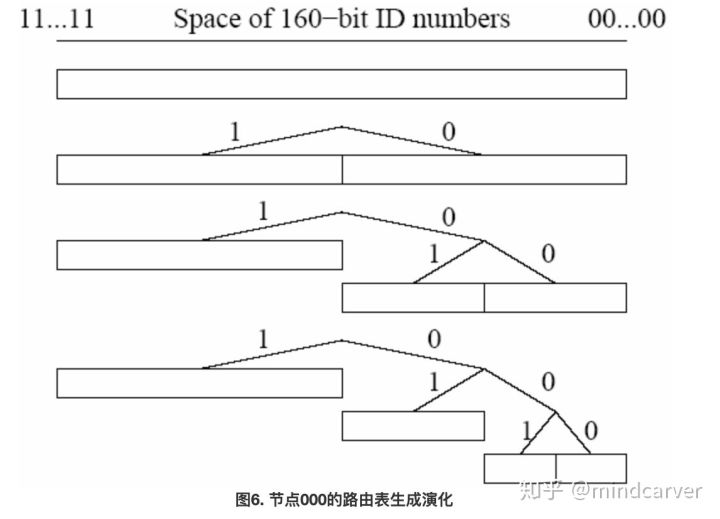
如果节点 u 要想加入 Kad 网络，它必须要和一个已经在 Kad 网络的节点，比如 w，取得联系。u 首先把 w 插入自己适当的 K 桶中，然后对自己的节点 ID 执行一次 FIND\_NODE 操作 (向 w 发布 查找 u 的 FIND\_NODE 请求)，然后根据接收到的信息更新自己的 K 桶内容。通过对自己邻近节点由近及远的逐步查询，u 完成了仍然是空的 K 桶信息的构建，同时也把自己的信息发布到其他节点的 K 桶中。

节点 u 为例，其路由表的生成过程为：

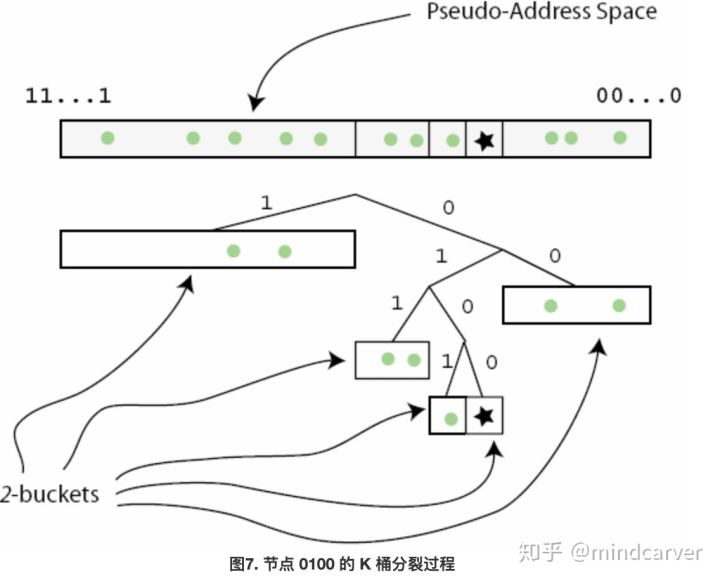
* 1. 最初，u 的路由表为一个单个的 K 桶(此时节点只有一个K桶)，覆盖了整个 160 bit ID 空间，如图 6 最上面的路由表；
  2. 当学习到新的节点信息后，则 u 会尝试把新节点的信息，根据其前缀值插入到对应的 K 桶中。
  3. 如果该 K 桶没有满，则新节点直接插入到这个 K 桶中；
  4. 如果该 K 桶已经满了，分为以下两种情况：

1. 如果该 K 桶覆盖范围包含了节点 u 的 ID的前缀，则把该 K 桶分裂为两个大小相同的新 K 桶，并对原 K 桶内的节点信息按照新的 K 桶前缀值进行重新分配。
2. 如果该 K 桶覆盖范围没有包含节点 u 的 ID的前缀，则直接丢弃该新节点信息

5. 上述过程不断重复，最终会形成表 1 结构的路由表。达到距离近的节点的信息多，距离远的节点的信息少的结果，保证了路由查询过程能快速收敛。



在图 7 中，演示了当覆盖范围包含自己 ID 值(所谓覆盖自己，就是指节点与该K桶的ID值的前几位是完全相同的)的 K 桶是如何逐步分裂的。



当 K 桶 010 (其前三位与节点0100是完全相等的)满了之后，由于其覆盖范围包含了节点 0100 的 ID，故该 K 桶分裂为两个新的 K 桶：0101 和 0100，原 K 桶 010 的信息会根据其其前缀值重新分布到这两个新的 K 桶中。注意，这里并没有使用 160 bit 的 ID 值表示法，只是为了方便原理的演示，实际 Kad 网络中的 ID 值都是 160 bit 的

**节点离开 Kad 网络不需要发布任何信息**，Kademlia 协议的目标之一就是能够弹性工作在任意节点随时失效的情况下。为此，Kad 要求每个节点必须周期性 【一般是： 每小时】 的发布全部自己存放的<key,value>数据，并把这些数据缓存在自己的 k 个最近邻居处，这样存放在失效节点的数据会很快被更新到其他新节点上。所以有节点离开了，那么就离开了，而且节点中的k-桶刷新机制也能保证会把已经不在线的节点信息从自己本地k-桶中移除。

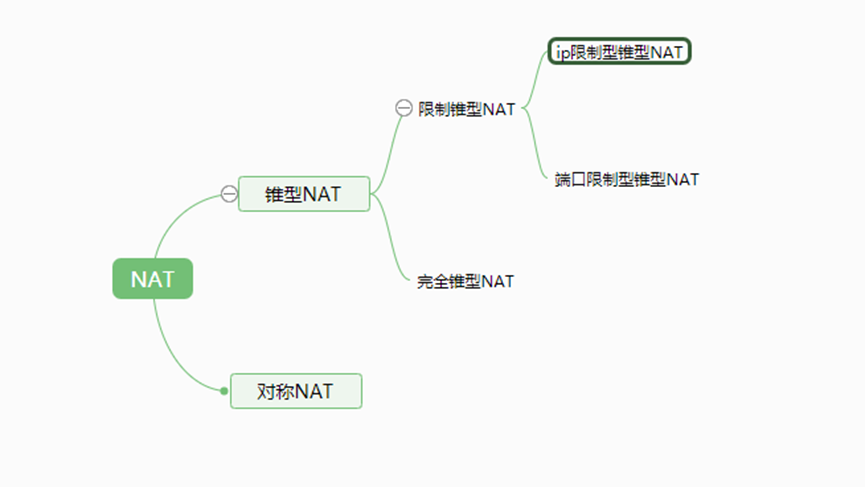
k-桶刷新机制下，节点会对那些在过去一小时之内没有收到任何节点查询操作的k-桶执行刷新操作(BT协议实现规定为15min)。具体实现是从节点的某k-桶中选择一个随机的节点信息，并对该节点ID执行一次FIND\_NODE操作。

第七章 P2P网络中的NAT穿越技术

1. **P2P网络中的NAT穿越原理**
   1. **NAT的原理和分类**

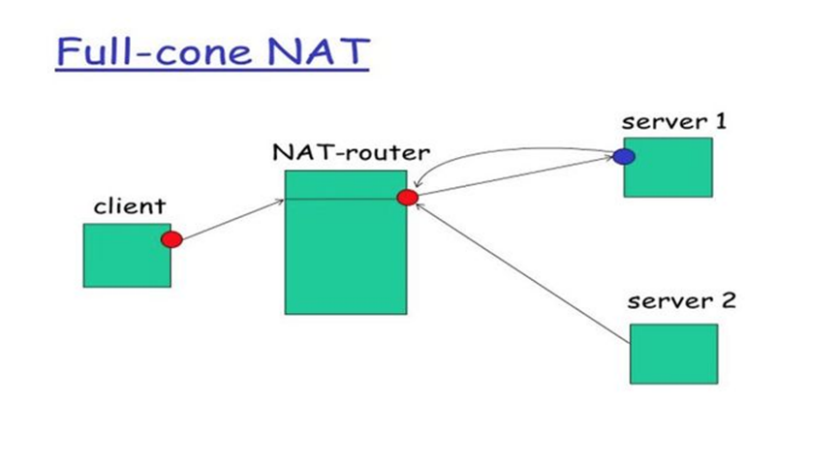
NAT是IETF标准，它通过将局域网内的主机IP映射为Internet上有效的公网IP地址，从而实现网络地址的复用。使用NAT技术，局域网内的多台PC可以共享单个、全局路由的IP地址，减少了所需的IP地址的数量。

NAT主要可以分为两类：基本NAT和NAPT(Network Address/PortTranslation)。基本NAT一般用于NAT拥有多个公网IP的情形下，将公网IP地址与内网主机进行静态绑定。假如NAT只拥有少量的公网IP地址，那么就要使用NAPT技术，NAPT与基本的NAT技术不同，它将内部连接映射到外部网络中的一个单独的IP地址上，同时在该地址上加上一个由NAT设备选定的端口号，即NAPT技术是对基本NAT技术的一个扩展，它实施的是<IP:PORT>对的转换。根据映射方式的不同，NAPT可以进一步分为以下两种类型：圆锥形NAT(Cone NAT)和对称型NAT(Symmetic NAT)。其中圆锥形NAT包括完全圆锥形NAT（Full Cone NAT）、受限圆锥形NAT(Restricted Cone NAT)和端口受限型NAT（Port Restricted Cone NAT）。



* + 1. **完全圆锥形NAT（Full Cone NAT）**

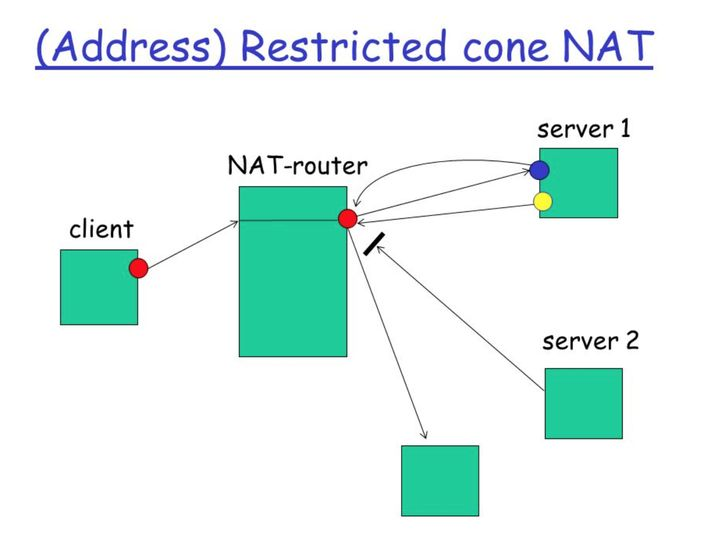
完全圆锥形NAT，将从同一内部IP地址和端口来的所有请求(不管它的目的主机的IP或端口是否不同)，都映射到相同的外部IP地址和端口。而且，任何外部主机通过向映射的外部地址发送报文，都可以实现和内部主机进行通信。这是一种比较宽松的策略，只要建立了内部网络的IP地址和端口与公网IP地址和端口的映射关系，则所有Internet上的主机都可以访问该NAT之后的主机。



NAT会将内网地址{X:y}（X代表内网主机的IP地址，y代表端口）映射成公网地址{A:b}（A代表映射的公网IP地址，b代表映射的端口）并绑定。任何数据分组都可以通过地址{A:b}送到此内网的主机。

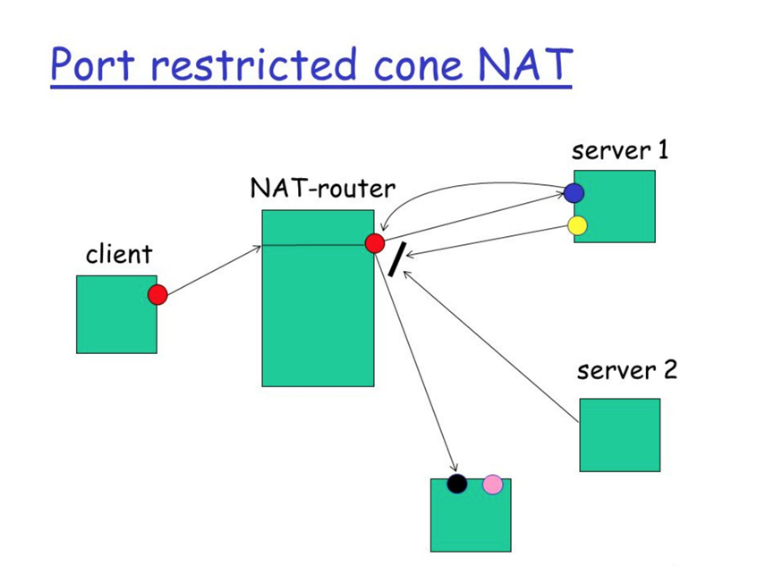
* + 1. **受限圆锥形NAT(Restricted Cone NAT)**

也是将相同的内部IP地址和端口号来的所有请求，映射到相同的公网IP地址和端口。但是与完全圆锥形NAT不同，内部客户端必须首先发送数据包到内网外的对方(IP=X.X.X.X)，然后才能接收来自该方(IP=X.X.X.X)的数据包。即是说：同一个内部地址（IP+PORT）会映射相同的NAT的公网地址(eAddr:ePort)，映射完成后，内部必须先发一个包给目标，然后才能收到目标回发的包，在进行接受目标数据包时，对目标的端口没有限制，而对目标的IP地址是严格限制的，只能是由内部主机在进行发送时的对端IP。



* + 1. **端口受限型NAT（Port Restricted Cone NAT）**

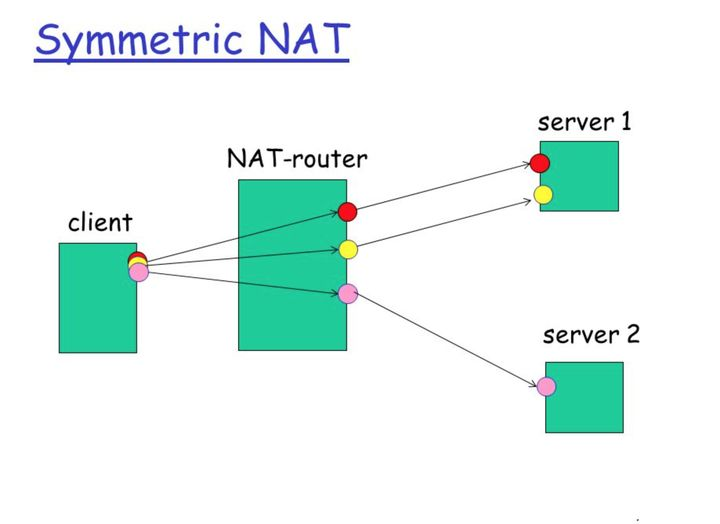
类似于受限圆锥型NAT，但更加严格。端口受限型NAT增加了对端口号的限制。也是同一个内部地址只会映射相同的NAT公网地址(eAddr:ePort)，映射完成后，必须先发一个包给目标(tAddr:tPort)，然后才能收到目标(tAddr:tPort)回发的包，在接受目标回发的包时，要注意，比如说内网的主机之前已经分别和地址为M、端口为n的主机以及地址为P、端口为q的住进进行了通信，所以只有来自这两个公网地址和端口的分组才能经过NAT网管到达内网主机。



* + 1. **对称型NAT(Symmetic NAT)**

对称型NAT网管具有多个公网地址。它把从同一内网地址（IP+PORT）到相同目的地址和端口的所有请求，都映射到同一个公网地址和端口。如果同一个内网主机，用相同的内网地址和端口向另一个目的地址（即使是只有端口不同）发送分组，则会使用不同的映射，而且公网主机也是只有在接收到来自于NAT内网的分组后，才能向与发送分组的内网主机进行通信。可见，对称型NAT是所有NAT类型中限制最为严格的。

对称型NAT会将内网地址{X:y}转换成公网地址{A:b}并绑定为{X:y}|{A:b}<-->{P:q}。这就意味着NAT只允许地址{A:b}接受来自{P:q}的分组，把它转给{X:y}。当该内网客户机请求与一个不同的公网地址{M:n}建立链接时，NAT会新分配一个外部地址{C:d}。



* 1. **常见的NAT穿越解决方案**

NAT技术在缓解IPV4地址紧缺问题、构建防火墙、保证网络安全等方面都发挥了重要的作用。然而，NAT设备的广泛存在却给Internet上的主机，特别是处于不同内网中的主机进行P2P通信带来了障碍，限制了P2P的应用。NAT阻碍主机进行P2P通信的主要原因是NAT不允许公网主机主动访问内网主机，这使得Internet上具有公网IP地址的主机不能主动访问NAT之后的主机，而位于不同NAT之后的主机之间更是无法相互识别因而不能直接交换信息。因此，要在目前的网络环境中进行有效地P2P通信，就必须采用相应的方案来穿越NAT。

目前业界主要的解决方案由：ALG方式、MIDCOM方式、STUN方式、TURN方式、ICE方式、Full Proxy方式等。

* + 1. **应用层网管（ALG）方式**

应用层网管（ALG，Application Layer Gateway）是指能够识别特定协议（如H.323和SIP等）的设备。在网络中增加了ALG设备可以很好地配合NAT完成应用协议消息中的地址字段翻译。

NAT和NAPT只能对IP报文的**头部**地址和TCP/UDP**头部**的端口信息进行转换，对于报文的**数据部分**可能含有包含IP地址或端口信息的特殊协议（如H.323、SIP、MGCP等），则无法实现有效的转换，而许多应用中需要对数据包负载中的数据进行分析转换。例如一个FTP服务器处于内网中，只有内网IP地址，该服务器在和公网主机建立会话的过程中，需要将自己的IP地址发送给对方，而这个地址信息放在IP报文的数据部分，现有的NAT设备是无法对它进行地址转换的，当公网主机接收到这个私有地址并使用它时，是不可能建立连接的。增加应用层网管ALG之后，就可以进一步分析数据包负载内的数据，即应用层的数据。所以，当网络中使用了NAT设备来屏蔽内部IP地址时，应用层网管ALG就可以同时实行业务流对NAT的穿越了。ALG可以是单独地连接与公网和内网之间的设备，也可以置于NAT设备或防火墙设备中。

由于ALG大部分功能集成在NAT设备中，这导致了该方式的可扩展性不强，因为在进行应用升级时，也需要对NAT设备升级，这在实际中很难实现。ALG方式的另一个不足就是ALG必须透彻地理解应用协议，需要根据协议过程协调NAT的地址翻译过程，并同步修改数据包负载中的相关字段，这会加重网关的负担，对于NAPT来说情况更复杂，处理开销过大，可能会影响正常的网络运行，形成网络瓶颈。而且ALG无法识别加密后的报文内容，这就必须保证报文采用明文传送，使得在公网中传送时存在安全隐患。

* + 1. **MIDCOM方式**

MIDCOM代理方式与ALG方式在功能上基本相同，也需要了解相应的NAT的工作机制，负责具体的应用处理，协助NAT一起对数据包进行处理。不同之处在于MIDCOM的框架结构采用可信的第三方（MIDCOM代理）对NAT进行控制。

MIDCOM代理的物理实体可以是代理服务器，所以在升级应用服务器的时候，不需要更改NAT的基本特性，通过对MIDCOM代理服务器的升级就可以支持更多的新业务，这是MIDCOM相对ALG方式的一个很大的优势。但是MIDCOM技术的实施需要对服务器进行MIDCOM协议的升级扩展，因此该方案的实施过程中会遇到投入较大、实现周期较长的问题。而且目前的MIDCOM代理方案还在研究过程中，并没有形成一种成熟的产品，所以现阶段使用MIDCOM实现NAT穿越的实际应用还不是很多。

* + 1. **STUN方式**

STUN，即UDP对NAT的简单穿越方式。这种解决方案的基本思路是，局域网接入用户（内网用户）通过某种机制预先得到其内网地址对应在出口NAT网管上的公网地址，然后在其报文段的地址信息表处（报文段的数据部分）直接填写出口NAT上的公网地址，而非用户的私有IP地址，这样报文段的内容在经过NAT时就无需被修改了，只需要按照普通NAT流程转换报文头部的IP地址即可，负载数据中的IP地址信息和报文头部地址信息是一致的。STUN协议就是基于此思路来解决应用层地址的转换问题的。

在穿越过程中，内网主机向NAT外的SYUN服务器发送STUN请求，STUN服务器接收到请求消息，产生响应信息，响应消息中包含了请求消息的源端口地址，即内网主机在NAT网管上对应的外部端口地址。然后响应消息通过NAT转发给内网主机，内网主机在以后的业务呼叫过程中，会将该公网地址填入呼叫协议的报文负载中，告知对方自己的RTP接收地址和端口号为NAT外的公网地址和端口号。由于通过STUN协议已在NAT上预先建立了媒体流的NAT映射表，所以媒体流可以顺利穿越NAT.

STUN协议最大的优点是无需现有NAT设备做任何改动。目前，网络中已有大量的NAT防火墙，而且这些NAT防火墙不一定支持VoIP应用。如果采用MIDCOM或ALG方案，则需要替换现有的NAT防火墙，实施起来难度较大，且MIDCOM方式无法实现对多级NAT的有效控制。如果采用STUN方式，不仅无需改动NAT防火墙，而且能够很好地适应多个NAT串联的网络环境。

STUN协议的局限性在于需要终端应用服务程序支持STUN客户端的功能;同时STUN不支持TCP连接的穿越；另外STUN方案也不支持业务对防火墙的穿越以及对称型NAT的穿越。

* + 1. **TURN方式**

TURN，即采用中继的NAT穿越。该方式与STUN在技术机制上是相同的，也采用了服务器/客户端模式，通过访问处于公网中的服务器得到内网中主机所分配得到的公共地址，进而进行通信。不同的只是在STUN方式中得到的公共地址是由NAT分配的，而在TRUN模式中内网主机得到的公共地址是由TRUN服务器分配的。TRUN通过分配TRUN Server的地址和端口作为客户端的对外的接收地址和端口，即内网主机发出的报文都需要经过TRUN Server进行中继转发，这种方式除了具有STUN方式的优点之外，还解决了STUN应用无法穿越对称式NAT，同时支持基于TCP的应用。

然而，由于服务器的存在，当数据量变得很大时，也会造成网络瓶颈，而且由于报文都必须经过TURN服务器转发，增大了报文的延迟和丢包的可能性，也会给QoS带来问题，目前网络流行的是将TURN和STUN结合在一起使用。

* + 1. **ICE方式**

ICE (Interactive Connective Establishment)目前已经被公认为在非对称性NAT环境下首选的NAT穿越方案。ICE本身是一种方法，它综合运用STUN、TURN等协议来提供一个通用的解决方案，使之在最合适的情况下工作，以弥补单独使用其中任何一种所带来的固有缺陷。

ICE的优势是支持各种NAT的穿越，但是需要每一个中断都支持ICE穿越方法，在开发上显著地增加了复杂性，而且网络还必须支持STUN、TURN等功能，实现起来较为复杂。

* + 1. **Full** **Proxy方式**

Full Proxy方式的思路是利用转发设备将内网主机的所有数据信息进行处理和转发，通过这种方式实现对NAT的顺利穿越，其关键之处在于同时完成对终端呼叫信令的代理转发和媒体中继。内网主机产生的信令流与业务数据流首先通过Proxy设备，由Proxy设备对其进行解析，并将信息中的私有地址改为Proxy设备的公网IP地址和端口，完成修改后将信息发送到目的地。

在Full Proxy方式下，无需对现有的NAT设备进行任何改动，适应性非常强，同时私有网络中主机的应用程序也无需支持STUN和TURN协议，这是Full Proxy方式的一个优势。但是由于Proxy设备是所有信息的必经之处，需要对数据包进行集中的分析和处理，对设备的要求很高，而且Full Proxy方式会增加包的延时和丢包的可能性，可能会造成网络瓶颈，而且如果该节点不能正常工作，那整个内网都无法与外界进行联系。

下面对上述几种方案进行一个小结，分别是各种方案在性能、可扩展性、对现有设备改造、安全以及QoS方面的比较。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 方式 | 设备负担 | 服务器 | TCP/UDP | 可扩展性 | 组网应用 | 现有设备改造 | 安全性 | QoS |
| ALG | 大 | 无 | 全部 | 升级NAT | 小区/企业网 | 需支持ALG | 比较高 | 无 |
| MIDCOM | 小 | MIDCOM代理 | 全部 | MIDCOM代理进行协议开发 | 小区/企业网/互通网关，视NAT效率而定 | 现有设备和呼叫代理支持MIDCOM协议 | 高 | 无 |
| STUN | 小 | STUN | UDP | 升级网络终端 | 小区/企业网 | 支持STUN Client | 低 | 无 |
| TURN | 小 | TURN | 全部 | 升级网络终端 | 小区/企业网 | 支持TURN Client | 低 | 无 |
| FullProxy | 小 | FullProxy | 全部 | Proxy需要新协议的扩展 | 小区/企业网 | 提供FullProxy设备，其他设备无需改造 | 比较高 | 有 |
| ICE | 小 | STUN和TURN | UDP | 终端和服务器进行协议扩展 | 小区/企业网 | 需支持ICE设备 | 低 | 无 |

* 1. **P2P网络中典型的NAT穿越技术详解**
     1. **P2P网络中UDP穿越NAT的原理和实现**

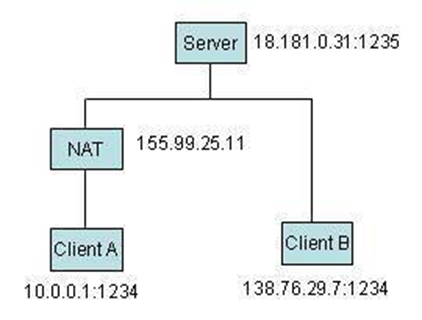
P2P网络中，UDP穿越NAT方式主要有以下几种。

1. **中转方式(Relay)**

中转方式是最简单的，也是比较可靠的的NAT穿越模式，它将P2P通信简单地转换为客户端/服务器通信模式。现假设A和B两个客户端都有一个到公共服务器S的已初始化的UDP连接，客户端位于各自的私有网络中，其各自的NAT阻止任何其它客户端直接向其发起连接。如果不直接连接，两个客户端可以简单地使用服务器S为他们转发消息。例如，为了向客户端B发送消息，客户端A可以沿着它已经建立的连接服务器S发送消息，然后S使用与B现存的连接向B转发该消息，达到A与B通信的目的。这种方式下，只要客户端能够连接到服务器，中转就可以工作，而且是最有效的。但是，其最大的缺点就是消耗了服务器的处理能力和网络带宽，多了中转环节，节点间反应时间也会增加，因而效率也是最低的。

1. **反向连接(Connection Reversal)**

反向连接方式只适合于仅有一个主机在NAT之后，另外的主机都拥有合法的公网IP地址的情况。两个客户端都需要与一个公网服务器S存在连接，这种方式的适用范围有明显的局限性。如果A想要初始化一个到B的连接，直接连接就可以了，因为B没有在NAT之后，A的NAT会认为这是外出的会话，不会阻拦。如果B想要初始化一个到A的连接，直接连接会被A的NAT阻止，因此B需要通过服务器S向A发送连接请求，请求A反向连接到B上。



1. **UDP打洞(UDP Hole Punching)**

UDP打洞技术假定了客户端A和客户端B可以与1上的已知服务器S建立UDP连接(可以互发UDP数据包)

当一个客户端在服务器S上登录的时候，服务器S会记录下该客户端的两个endpoints(包括IP地址和UDP端口号)，一个是该客户端确信自己是通过该IP和端口与服务器S进行通信的，另一个是服务器S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和端口号。我们可以把前一个endpoint看作是客户端的内网IP和端口，把后一个endpoint看作是客户端的内网IP和端口经过NAT转换后的公网IP和端口号。

服务器可以从客户端的登录消息的消息体中得到该客户端的内网endpoint相关信息，可以通过登录消息的IP或UDP头得到该客户端的公网endpoint。如果该客户端不是位于NAT设备后，那么采用上述方法得到的两个endpoint的值应完全相同。

建立点对点会话打洞过程(假设A想要与B直接建立UDP会话)如下所示。

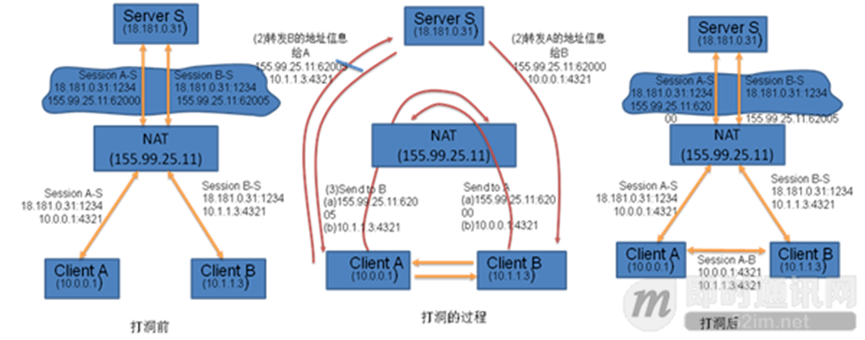
1. A最初不知道如何向B发起连接，于是A想服务器S发送消息，请求S帮助建立与B的UDP通信
2. S将含有B的公网和内网的endpoint发给A,同时，S将含有A的公网和内网的endpoint的用于请求连接的消息也发给B。一旦这些消息顺利到达，A和B就都知道了对方的公网和内网的endpoint。
3. 当A收到由S发来的包含B的公网和内网endpoint的消息，A开始向这些B的endpoint发送UDP数据包，并且A会自动锁定第一个给出相应的B的endpoint。同理，当B收到由S发来的A的公网和内网endpoint以后，也会开始向A的公网和内网的endpoint发送UDP数据包，并且自动锁定第一个得到A的回应的endpoint。由于A与B的互相向对方发送UDP数据包的操作是异步的，所以A和B发送数据包的时间前后并没有严格的时序要求。在这里分为以下3种具体情况来讨论。

第一种：两个客户端位于同一个NAT设备之后，即位于同一个内网中；

第二种：两个客户端分别位于不同的NAT设备后，即分属于不同的内网；

第三种：客户端位于两层NAT设备后，通常最上层的NAT是由ISP网络提供商提供的， 第二层的NAT是家用的NAT路由器之类的设备。

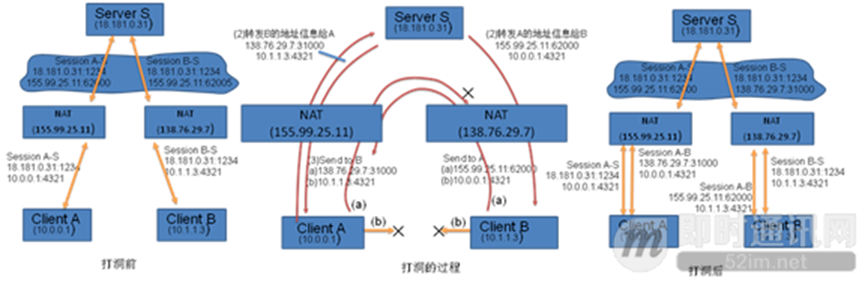
1. 首先假设两个客户端位于同一个NAT设备之后，即位于相同的内网(相同的私有IP地址域)，如下图所示：



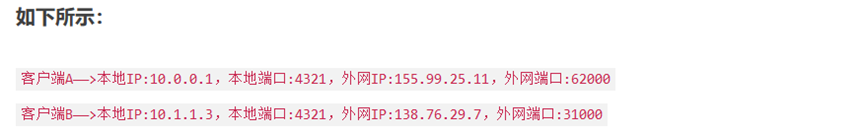
A与S建立了UDP连接，经过NAT转换后，A的公网端口被映射为62000。B同样与S建立了UDP连接，公网端口被映射为62005。假设A和B使用UDP穿孔作为方案，使用服务器S作为引导来建立一个通信通道。那么A与B就可以从服务器S处互相知道对方的公网IP地址和端口号，并且可以互相向公共地址上发送消息进行通信。这两个主机若采用这种方式通信，在NAT看来，是同一IP地址的不同端口之间的通信，因此只有在NAT允许内部的网络主机能够和内部的其他主机进行UDP会话的情况下才可以实现(也就是同一台设备，不同端口之间的UDP数据包能够到达)。即使NAT支持这种解释和转寄的步骤，但是很明显在这种环境下是不必要的，不仅会增加A和B之间的对话反应时间，而且也增加了NAT的负担。因为A和B是在同一个内网中的，因此内网数据包不需要路由就可以直接到达，并且速度更快。

总结来说，A和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数据包能到达（即Hairpin转换特性）。但无论如何A与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。因此A与B有很大可能性采用内网的endpoint直接进行通信。

1. 假定A与B在不同的NAT设备之后，分属于不同的内网，如下图所示：



A与B都经由各自的NAT设备与服务器S建立了UDP连接，映射关系如下所示：



服务器的地址为 IP:18.181.0.31 端口：1234

在A向服务器发送的登陆消息中，包含有A的内网地址二元组信息,即10.0.0.1:4321；服务器S会记录下A的内网地址二元组信息，同时会把自己观察到的A的外网地址二元组信息记录下来。

同理，服务器S也会记录下B的内网地址二元组信息和由服务器观察到的客户端B的外网地址二元组信息。

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和B。

由于A和B分属不同的内网，它们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。因此应用程序对于收到的消息必须要经过授权和过滤，只有通过授权的消息才能被接收。

当A的第一个消息发往B的外网地址，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}，和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，但它的目的地址是B的外网地址。

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所对应的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}。

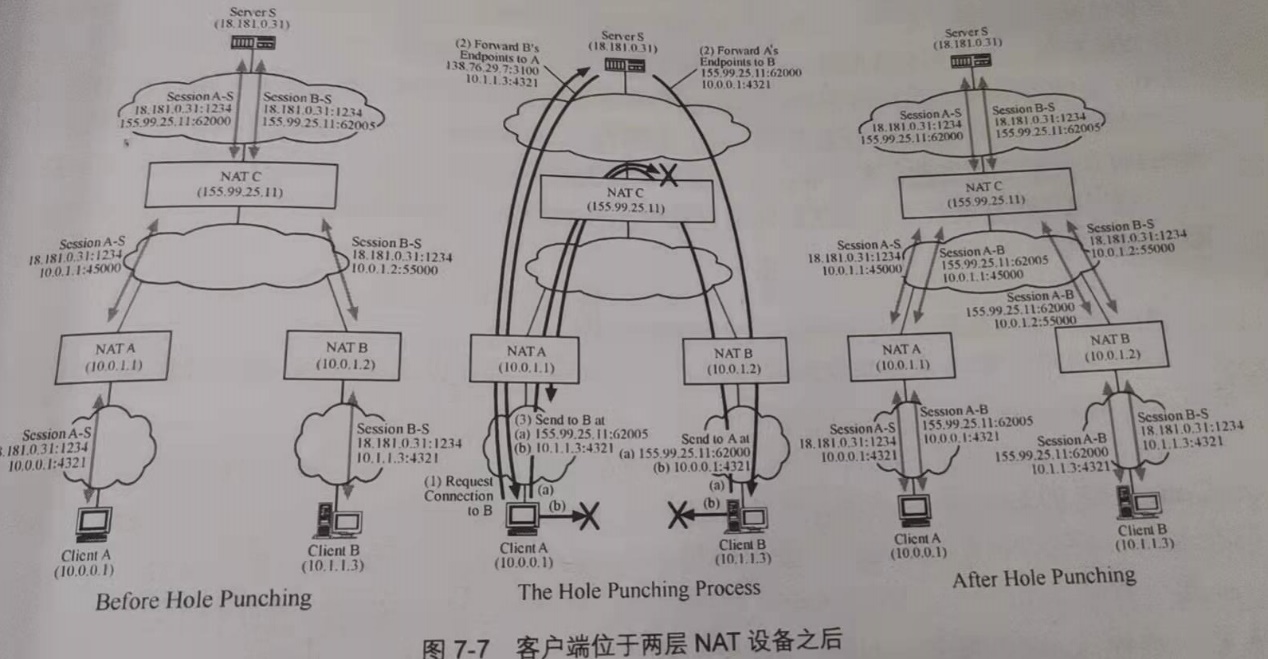
如果A发给B的外网地址二元组的消息包在B向A发送消息包之前到达B的NAT设备，B的NAT设备会认为A发过来的消息是未经授权的外网消息，并丢弃该数据包。但是此时的A的UDP协议会处于一个等待B应答的状态中（虽然该UDP报文被拒绝,但在NAT A上留下了一个入口，这个入口会允许NAT B向其发送的数据，通过查找之前定义的映射表,查找到该报文是发送给哪个内部客户机的，通过将该报文的目标地址替换为内部客户机的IP地址和端口后,转发给内部客户机）。

完成打洞的正确过程为：A在服务器S的要求下向B发送一个请求，若B应答的话就等价于接受了上面的A的UDP请求，双方就正式进入P2P传输模式了。即是说：一旦A或B向对方的NAT设备在外网上的地址二元组发送了数据包，就打开了在自己的NAT网管上的“洞”，这个洞会等待来自于对端的打洞数据包，当双方都建立了“洞”之后。此时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(完成了授权过程)，转而开始真正的P2P数据传输。

但是以上过程有一个限制：对于某些防火墙或者NAT,如果它检测到某个端口收到过外部来的非法数据,当作网络攻击而暂时关闭该端口,直接连接就无法建立。

最后一种情况，即两客户端位于两层(或多层)NAT设备之后，如下图所示：



假定NAT C是由ISP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内部网络接入NAT C的内网，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（我们将这种由ISP提供的内网地址称之为“伪”公网地址）。

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。

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}。因此客户端A与客户端B是无法通过服务器知道“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，仍然不建议采用上述的“最优化”的打洞方式。

因为这些由ISP服务提供商提供的地址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。这里NAT C是否支持内部的网络主机能够和内部的其他主机进行UDP会话的情况非常重要（也就是同一台设备，允许不同端口之间的UDP数据包能够到达）。如果不支持，数据包将无法有NAT C转发给NAT A 和NAT B，进而无法到达客户端A和B。

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}，当数据包到达NAT C，NAT C应该可以识别出来该数据包是要发往自身已经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。

**一个需要考虑的现实问题：UDP在空闲状态下的超时**

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”过程打出来的“洞”。如果P2P应用程序希望“洞”的存活时间不受NAT网关的限制，就最好在穿越NAT以后设定一个穿越的有效期。

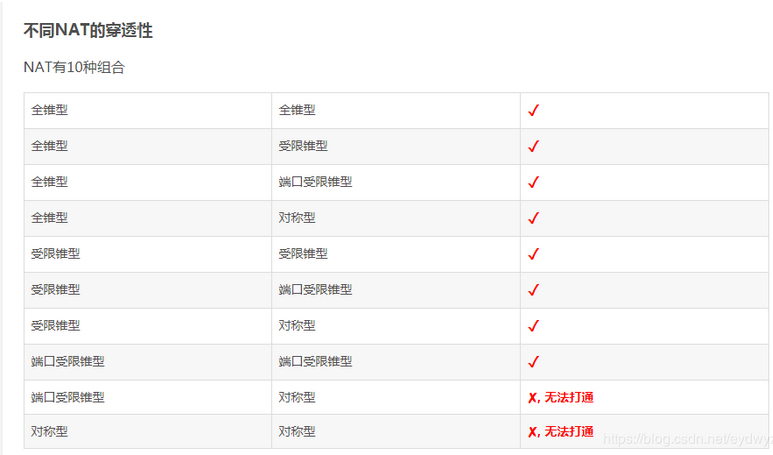
对于有效期目前没有标准值，它与NAT设备内部的配置有关，某些设备上最短的只有20秒左右。在这个有效期内，即使没有P2P数据包需要传输，应用程序为了维持该“洞”可以正常工作，也必须向对方发送“打洞”心跳包。这个心跳包是需要双方应用程序都发送的，只有一方发送不会维持另一方的Session正常工作。

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。

注：使用UDP打洞技术时要注意的是：它仅工作在额NAT都是圆锥形NAT（3种）的情况下。

1. **基于端口预测的UDP穿越****Symmetric NAT**

在 STUN 方式中，把 NAT 分为 4 种类型，分别是 FullCone 型，Restricted Cone 型，Port Restricted 型和 Symmetric型。在这 4 种 NAT 中，STUN 方式可以较好地实现前 3 种类型 NAT 的穿越，但是无法实现对第 4 种类型 NAT 的穿越。原因是在Symmetric 类型的 NAT 中，向不同的 IP 地址和端口发送的数据包，其外部地址将会被映射为不同的端口。这样一来，用户端结点就无法从 STUN Server 的响应消息中得知有效的端口信息。因为目的端结点的 IP 地址和端口号与STUN Server 的不同，这样发送方无法得知向接收方发送数据包时在外部 NAT 所获得的端口号(这里具体的来说，是因为在STUN的UDP打洞过程中，首先发送端会向STUN服务器发送登录请求，此时STUN服务器记录的是发送端的目的地址是STUN服务器地址的二元组信息，而并非目的地址是接收方地址的二元组信息)，从而无法完成对这种类型 NAT 的穿越。



**解决方案一：基于探测对Symmetric** **NAT与端口受限NAT的穿透方案**

此方案解决了一方是Symmetric NAT，而另一方是Cone NAT的UDP穿越问题。

此论文提出了两种Symmetric NAT端口重新分配的前提。

第一种情况是：Symmetric NAT对于从内部网络依次接收到的新连接（即<源IP，源端口，目标IP,目标端口>四元组在NAT映射表中找不到完全匹配的项），分配的输出端口是按照空闲端口依次连续分配。如果NAT为某个内部网络的新连接分配的端口号是X，则其将为第n个新连接分配的端口号为：X+n。

针对这种情况:

1. 假设NAT A为Symmetric NAT,NAT B为端口受限型NAT；且NAT A对每个新连接分配的输出端口每次增加 1。
2. NAT A内部有多个客户同时在利用NAT A访问外部网络的时候,即A发送数据给S和给B的连接之间的产生了 Δz个新 连 接。同 时 考 虑 到 可 能 有 少 量 端 口 (设 为 Δｗ个 )已经在使用中,分配的端口号将顺延。

则可以预测当 Ａ发送数据给B的时候,NAT A为其分配的输出端口号相对于 Ｘ′的偏移量为：Δｘ=Δｚ+Δｗ +1,且 Δｚ≥0,Δｗ≥0。因为在穿透过程中,两次数据的发送间隔不会很长,Δｘ是一个较小范围内的正数。

当服务器S向客户端A发送要求其用端口X向NAT B的端口Y’发送消息的时候，NAT A此时重新分配的端口为X’+x（X’是A与S建立连接时NAT A分配的端口号），此时NAT A留下入口，可以接收B发送至端口X’+x的消息。之后，服务器S向B发出指令，要求B连续发送多个UDP报文到NAT A的X+1端口至X+MAXTEST端口（MAXTEST为穿透测试次数最大值，由服务器S设定），这样当x<MAXTEST时，穿越即可成功。

在Ｂ收到命令并开始执行探测过程 (Δt为每次分配的端口号的偏移值 )：当Δｔ=Δｘ的时候NAT A接收该数据并转发给A, 同时在NAT B上也留下了一个入口。只有当B收到A的回复数据之后, 才知道直接连接已经建立成功。这个时候就可以停止探测了。此时也意味着穿越成功。

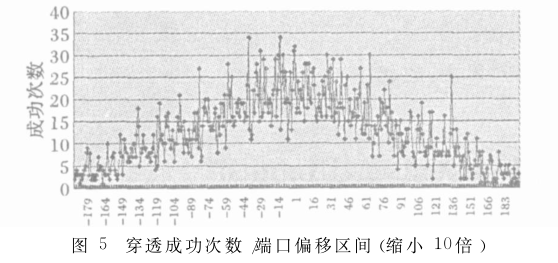
第二种情况是针对NAT从内网依次接收到的新连接，分配的输出端口是在一定端口范围内的随机连接（很多安全性较高的NAT设备对于输出端口的分配是随机的）。如果NAT为某个内部网络的新连接分配的端口号是X’，则其将为后面到来的第N个新连接分配的端口号为：X’+∆x（每个∆x都是由于NAT自行产生的随机数）。

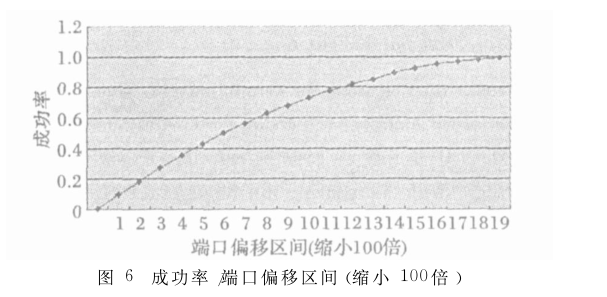
对于这种情况可以合理假设:

1. NAT A对新连接的输出端口分配是每次随机变化,两次之间偏移量为 Δｘ(Δｘ可能为负 )；
2. 在NAT A内部有多个客户同时利用NAT A访问外部网络，即A发送数据给S和B的连接之间产生了Δz个新连接；
3. 虽然每次∆x值的选择是由NAT设备通过某种方式随机选择的，但通常每次分配的端口号之间都会具有一定的函数关系或统计上的相关性。当A连接S时，如果NAT A为其分配输出端口号X’，则A发往NAT B的时候，NAT A为其分配输出端口号：X’+∆x, ∆x=f(X’, ∆x)。

通过在多种网络条件下的测试和分析，也证实了多数情况下存在具有某种特性的分布特性的∆x值。因此，可以通过其分布特性来预测∆x的所属范围，并对该范围端口试探地实施穿透。

测试的结果：在不同时间对部署该类NAT的网络进行了测试,结果都如图5、图6所示。图5中,Δｚ的数值均匀分布在中心点的两边；图6中,当设定最大穿透测试端口偏移量为1200(横坐标的偏移量缩小了 100倍 ),能获得 80%以上的穿透成功率；当设置最大穿透测试端口偏移量接近 2000,能获得接近 100%的穿透成功率。





**穿透方案的实施和改进：**

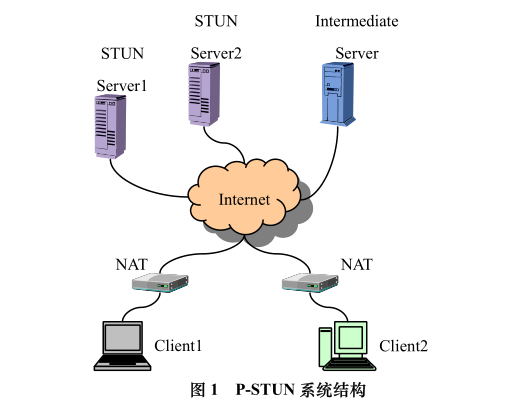
在实际实施中,还采用一些对于特定网络环境的经验办法来提高性能：通常在发送穿透消息前,先随机发送几个假的穿透包 (即不是向目标NAT B的指定端口发送的UDP报文 ),然后再发送穿透消息,这样常常能减少试探次数并提高穿透的成功率。

对于特定网络环境中的 Δｚ,可以通过一些测量方法来进行估计。这些测量方法在实际应用中,可以通过客户端与服务器之间多建立几个连接来进行。同时在应用规模不大的情况下,服务器端也可以保存一些关于某些网络的NAT分配的特征,用来调整 Δｚ的估计范围。

**解决方案二：基于预测方法的 NAT 穿越技术 P-STUN**

该方案针对双方都是Symmetric NAT的UDP穿越问题，提出了一种在公网另外配置两台至数台端口检测服务器进行预测的方法（P-STUN）。这比较适合于Symmetric NAT中端口映射等差变化的情况。

在 P-STUN 方法中，最重要的是要获取足够多的预测信息。为了达到这个目的，必须要知道 NAT 的端口分配策略，因此需要向不同的 IP 地址和端口发送 STUN 请求消息。所以，在 P-STUN 方法中需要两个 STUN Server：STUN Server1和 STUN Server2；同时，为了获取目的端结点的相应信息，还需要一个 Intermediate Server 用于交换这些信息（指的是交换对方的预测端口）。使用P-STUN 方法的系统结构如图 1 所示。

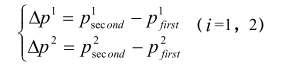


其中，STUN Server1，STUN Server2 和 Intermediate Server 都运行在外部公共网络上，STUN Server1 和 STUN Server2 具有不同的 IP 地址和端口号。P-STUN 方法运行在

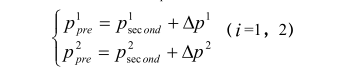
参与通信的两个对等结点 Client1 和 Client2 上。

运用 P-STUN 方法进行 Symmetric NAT 穿越的算法可以用以下步骤表示：

1. 参与通信的 Client i （ i =1，2）分别向 STUN Server1和 STUN Server2 发送 STUN 请求，从得到的两次响应消息中可以得到被分配的外部地址和两次被分配得到的端口号：， （ i =1，2）。
2. Client i （ i =1，2）根据两次被分配的端口号和 计算 Symmetric NAT 的端口分配间距



1. Client i （ i =1，2）根据预测实际建立连接时将要被分配到的端口号：



1. Client i （i =1，2）通过 Intermediate Server 发送各自的，同时获得对方的外部地址预测端口号。
2. Client i （ i =1，2）向目的端对等结点的目的地址及预测端口号发送连接用的数据包，从而建立起连接。在这里需要特别指出的是，为了保证连接的成功建立，理论上建立连接的双方应该确保连接数据包同时到达目的端。因为如果一方的连接数据包已经到达 NAT 而另一方的数据包还没有发送，那么根据 Symmetric NAT 的定义，连接不能成功建立。但是在实际情况中，由于网络情况不同，不可能完全确保两个端口同时打开，因此为了确保连接建立的成功率，Client i 将连续多次向目的端发送连接数据包，直到连接成功建立为止。
3. 连接成功建立，对等结点双方进行后续的通信内容。

**方法性能改善：**

P-STUN 方法是基于预测的 Symmetric NAT 穿越解决方案，因此其性能大部分取决于预测的命中率，即预测的准确程度。而影响预测命中率的因素主要有两个：对目的端端口号的预测准确程度和临界时间（Critical time）的长度。

对目的端端口号的预测准确程度：也就是预测所得到的端口是实际传输中所获得端口的概率。这对于 P-STUN 方法的性能有着很重要的影响，因为如果预测的目的端端口错

误，连接是无法建立的。在该方法中，为了确保对目的端端口预测的准确性，采取了多端口预测策略，即在建立连接的时候，Client i ( i =1，2)不仅向端口发送数据包，同时也向端口+1，+2，⋯，+N（N 可以根据实际情况设为一个大于 1 的自然数，在 P-STUN 里 N 设为 4）发送数据包。这样不仅提高了对目的端端口号的预测准确性，而且也降低了临界时间过长可能导致的连接建立失败的概率。

临界时间（Critical time）的长度：临界时间指的是从Client i (i =1，2)向 STUN Server 发送请求消息开始到向目的端结点发送连接用数据包为止所经历的时间长度。临界时间的长度同样对 P-STUN 方法的成功率有着很大影响。如果临界时间过长，很可能在临界时间内，另一个应用程序发起了新的连接请求，这样 Symmetric NAT 会按照分配规则为它分配一个端口号，这样当临界区结束的时候，预测得到的端口号就不正确了，因为该端口号已经被分配给另一个应用程序了。为了减少这一情况的发生，一方面，需要采取向多个目的端端口发送连接数据包的策略来降低临界时间过长导致的预测端口号失效问题；另一方面，也需要简化预测算法的复杂性，缩短临界时间的长度，因此在 P-STUN的预测算法中，简化了 Symmetric NAT 的类型划分。

**缺点：**

P-STUN方法是基于 STUN 方法的，因此只能完成 UDP 类型的 NAT穿越，而无法完成对 TCP 类型的 NAT 穿越；其次，为了降低预测算法的复杂度，刻意简化了对Symmetric NAT 的类型细分。在算法中，默认 NAT 将按照一定的间距△p 来分配端口号，而实际情况中，NAT 也可以是按照事先设定的分配表来为每个请求分配端口号，这样前后两次分配的端口号不存在线形关系。在这种情况下，预测算法有可能失败。（之所以在 P-STUN 中没有区分这种情况，一方面是因为目前绝大多数Symmetric NAT路由器是按照一定的固定间距来进行端口分配；另一方面也是为了简化预测算法，从而缩短临界时间长度，提高算法的命中率）

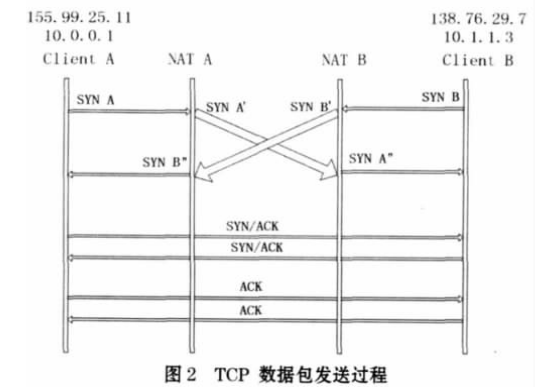
* + 1. **P2P网络中TCP穿越NAT的原理和实现**

目前，UDP穿越NAT的方法已经比较成熟。但是有时P2P应用需要可靠地数据传输，另外防火墙一般仅允许通过实现打开的特定协议和端口，甚至有些防火墙还关闭了UDP端口。因此使用TCP连接P2P通信成为必要的方法。另外，TCP是面向连接的协议，需要握手、协调包的顺序，因此TCP穿越NAT比UDP更加困难。假定两个主机在不同的 NAT 后面， 并且都知道对方的 IP地址。如果这些主机想直接发起 TCP 连接，那肯定失败。因为在直接的 TCP 连接中，必须有一方是发起者（ 发送 SYN 包），另一方必须处于监听状态。在两个主机都在 NAT 后面的情况下，监听者不可能收到 SYN 包，因为 SYN 包在到达 NAT 的时候就被丢弃了。 这是因为 NAT 或者防火墙不允许来自 INTERNET 未经请求的数据包进入他们的网络内部。 所以，为了在不同 NAT 后面的主机之间建立直接的连接， 必须让 NAT 以为这个连接是内部主机发起的。

我们可以让两边的主机都发起一个 TCP 连接，也就是创建一个 SYN 包，这样两边的 NAT 都会以为这个连接是从内部发起的，是经过内部请求的，因此，允许后续数据进入它的网络了。

与UDP打洞技术类似，同样需要借助第三方服务器X来实现TCP穿越NAT。NAT之后的用户A/B首先在这个服务器X上进行注册。服务器X记录用户的私网地址和NAT后的公网地址。如果用户A要与用户B建立TCP连接，A发送一个请求消息给服务器X，服务器X返回B的公网地址（端口）以及私网地址（端口）给用户，同时服务器X也发送消息给B，消息中也包含了A的相关信息。

在得知了对方的地址和端口信息后，建立TCP连接的过程就可以开始了。建立TCP连接的详细过程如下图所示：



第一步： Client A 、Client B 同时发送 TCP SYN 包给对方，如图所示分别命名两个 SYN 包为 SYN A，SYN B 。 他们所包含的信息分别为

SYN A ：

Source Address ： 10.0.0.1 TCP port ： 1234

Destination Address ： 138.76.29.7 TCP port ： 310000

SYN B ：

Source Address ： 10.1.1.3 TCP port ： 1234

Destination Address ： 155.99.25.11 TCP port ： 620000

第二步： SYN A 首先通过 NAT A （ 必须在 SYN B' 到达NAT A 之前）， NAT A 看到 SYN A 包并将其地址信息转换为：

Source Address ：155.99.25.11 TCP port ： 620000

Destination Address ： 138.76.29.7 TCP port ： 310000

我们把这个经过 NAT A 转换的包叫做 SYN A' 。

同理，由连接的对称性可以得到 SYN B' ：

Source Address ： 138.76.29.7 TCP port ： 310000

Destination Address ： 155.99.25.11 TCP port ： 620000

这时， NAT A 和 NAT B 都在自己的 TCP 连接表中存储了上面两个公网的 IP 地址和端口信息，因此，只要看到包含这两个信息的 SYN 包，都会让其通过。

第三步： SYN A' 到达 NAT B ， NAT B 检测 SYN A' ，若它的地址信息和自己 TCP 连接表中的信息相符， 便让 SYN A'通过， 这个包就能够到达内部网中 Client B 上， 并将 SYN A'的地址信息转换为：

Source Address ： 155.99.25.11 TCP port ： 620000

Destination Address ： 10.1.1.3 TCP port ： 1234

我们称这个包为 SYN A'' 。

同理， SYN B' 也能顺利通过 NAT A 顺利到达 Client A 。此时包转化为 SYN B'' ：

Source Address ： 138.76.29.7 TCP port ： 310000

Destination Address ： 10.0.0.1 TCP port ： 1234

这时， Client A 收到了 SYN B'' ， Client B 收到了 SYN A'' ，并返回给对方 ACK ，经过三次握手，这个 P2P 的 TCP 连接就建立了。

上述过程的关键在于要确保接收端的 SYN 包比发送端发送过来的 SYN 包要先一步到达接收端的 NAT 网管（ 即对于 Client A来说 SYN A 比 SYN B' 要先到达 NAT A ， 这个时候 SYN A被转化为了 SYN A' 然后 SYN B' 才到达。 这样当 SYN B'到达的时候， NAT A 的端口地址已经被转换为了外网的地址，所以这时的 SYN B' 就会被认为是内部网络想要的包不会被NAT 丢弃。 ）

为此，一种方法是在接收端适当地设置 NAT 和防火墙，防止那些提前到来的 SYN 包进入内部网络主机。 一些 NAT 和防火墙会发送 TCP 重置包 (RST) 到不请自来的包的源地址。

另一种确保 SYN 包不会过早到达它的目的网络的方法是发送带有 TTL 值低于对方的 NAT 路径长度的 SYN 包。 这包将会在去目的地的路上被抛弃， 并且 TCP RST 包将不会被任何发送者看到， 但是这样做就会产生一个问题， 那就是一个ICMP 超时包将会被 NAT 收到。 因为 ICMP 超时包的出现也会突然地终止一个 TCP 连接。 然而，如果使用者能够设置他们本地的防火墙抛弃 ICMP 包， 或者设置 NAT 不发送 ICMP 消息包到它的内部网络， TCP 连接尝试就不会突然终止。

**注：此方案需要对NAT网管进行设置，不是很适用**

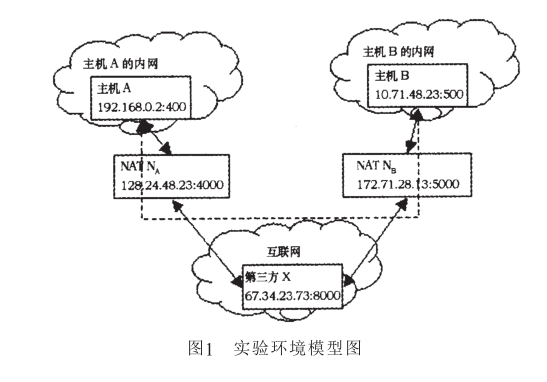
**另一种方案：利用第三方来交换两台内网主机建立TCP连接信息从而完成TCP连接的建立**

* 1. 问题描述及假定

首先假定两台主机都处于不同内部网的 NAT 之后, 且互相都知道对方的 IP 地址（互相知道对方的IP地址可以直接通过公网第三方主机实现，但是端口号是不能轻易互相知道的）。如果两台主机想要直接建立 TCP 连接, 任何一方的 TCP 握手 SYN 报文在抵达对方 NAT 主机就将被丢弃。因此, 要在这样两台主机之间建立 TCP 连接, 任何一方都必须知道另一方的外部端口号。这样的端口号是在内网主机向外网主机发起请求后由 NAT 为其分配的。即使内部的主机也不能知道 NAT 为其分配的端口号。其次再假定我们提供的位于公网第三方主机的 IP 地址和端口号都为两台位于 NAT 后主机所知道。

* 1. 实验研究

实验研究环境模型如图 1 所示。



实验研究的目的就是利用第三方 X 在两台 NAT之后的主机 A 和 B 之间建立直接的 TCP 连接。根据NAT 对端口的分配情况, 一共分为三种情况: 一种是两个主机 NAT 对端口的分配情况是可预测的; 第二种情况是有一个 NAT 端口的分配是可预测的, 一个是随机不可预测的; 最后一种情况是两个 NAT 端口的分配都是随机不可预测的。

1. 对于第一种情况, 两个 NAT 端口的分配都是可预测的

假设 NAT 每收到一个报文后其分配给内网主机的端口号值是连续递增的。要使两台位于 NAT之后的主机 A 和 B 直接建立 TCP 连接, 处理步骤如下：

( 1 ) 主机 A 和 B 都利用第三方 X 作为路由中转发到对方主机的 SYN 报文。

NA:4000 →NB:5000 , SYN: P （NA:4000会被NAT NA网管认为是内部网络的TCP连接的发起主机）

NB:5000 →NA:4000 , SYN: Q （NB:5000会被NAT NB网管认为是内部网络的TCP连接的发起主机）

SYN 表示为 SYN 报文, P、Q 分别表示 IP 报文的流水号。这两个SYN实际上不会被对端主机接受，而会被NAT网管屏蔽。

( 2 ) 第三方主机 X 缓存 A 和 B 双方的 SYN 报文, 并分别转发给对方。

X:1234 →NA:3999 , SYN: Q ~~（NA:3999会被NAT NA网管认为是内部网络的TCP连接的发起主机）~~

X:1235 →NB:4999 , SYN: P （~~NB:4999会被NAT NA网管认为是内部网络的TCP连接的发起主机）~~

~~整个过程对于NAT NA网管来看，就是其端口为4000的应用向端口为3999的应用通过公网服务器发起了一个TCP连接请求，但是它无法注意到SYN包实际上已经被替换。这对于NAT NB网管也是一个道理。~~

( 3 ) 主机 A 和 B 分别利用 X 向对方发送 SYN+ACK报文, 报文的流水号自增一。（在收到 SYN 报文后,TCP/IP 协议栈会自动生成 SYN+ACK 响应报文）

NB:5000 →NA:4000 , SYN+ACK: Q , P+1

NA:4000 →NB:5000 , SYN+ACK: P , Q+1

Q 、P 分别表示 ACK 收到报文的流水号

( 4 ) 主机 A 和 B 收到伪装的 SYN+ACK 报文后分别利用 X 向对方发出 ACK 响应报文。（在收到 SYN+ 报文后, TCP/IP 协议栈会自动生成 ACK 响应报文）

NA:4000 →NB:5000 , ACK: Q+1

NB:5000 →NA:4000 , ACK: P+1

( 5 ) X 在收到主机 A 和 B 发送的 ACK 报文后, 就将其丢弃。因为, 他们都不需要收到 ACK 报文。

这样主机 A 和 B 已经完成了 TCP 连接的建立, 尽管三步握手事实上并未完成, 但双方主机都已受到X 的欺骗, 认为三步握手已经完成。

1. 第二种情况，一个 NAT 端口的分配是可预测的, 而另一个是随机不可预测的

假设可预测的NAT 每收到一个报文后其分配给内网主机的端口号值是连续递增的。要使两台位于 NAT 之后的主机 A和 B 直接建立 TCP 连接, 处理步骤如下：

( 1 ) 假设 N A 的端口分配是随机的, X 无法预测,但可以预测 N B 的端口分配为 5000 。

( 2 ) 假设 N A 分配的端口号为 m 。先由主机 A 通过X 向 B 发送 SYN 报文, X 将其 SYN 报文先缓存, 并获取其分配的端口号。 B 随机不重复地通过 X 向 A 发送SYN 报文 T 次。

NA : m →NB : 5000 , SYN: P

i = 0

While i < T

NB : 5000 →NA : rand , SYN 任意值

i++

End While

T 值作为阈值。

( 3 ) X 缓存 B 的 SYN 报文, 并判断其目的端口是否与 A 的一致。如果一致, X 就转发缓存的 A 和 B 的SYN 报文给对方。

X:1234 →NA: m- 1 , SYN: Q

X:1235 →NB: 4999 , SYN: P

( 4 ) 主机 A 、 B 收到 SYN 报文后, 按照情况一的第三步开始来建立 TCP 连接。

注：第二种情况与第一种情况相比, 只增加了获取随机分配 NAT 端口的过程。

1. 第三种情况，两个 NAT 端口的分配都是随机不可预测的

要使两台位于 NAT 之后的主机 A 和 B直接建立 TCP 连接, 处理步骤如下：

( 1 ) 由于 X 无法预测 N A 和 N B 分配的端口, 以此必须由 A 和 B 先发起连接。

( 2 ) 主机 A 和 B 都连续 T 次通过 X 向对方随机不重复的端口发送 SYN 报文。

i = 0

While i<T

NA : m→NB : 任意值 SYN: 任意值

i++

End While

j = 0

While j<T

NB : n→NA: 任意值 SYN: 任意值

j++

End While

( 3 ) X 缓存 A 和 B 发送的 SYN 报文, 并判断其目的端口是否为对方的对应端口。发现对应报文则转发给双方主机。

X:1234 →NA: m- 1 , SYN: Q

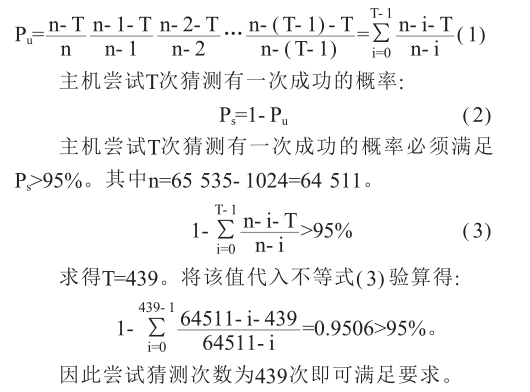
X:1235 →NB: n- 1 , SYN: P

( 4 ) 主机 A 、 B 收到 SYN 报文后, 按照情况一的第三步开始来建立 TCP 连接。

* 1. 尝试猜测次数的取值

尝试猜测次数即是之前提到的 T 值。一台主机猜测 NAT 分配的端口范围是在 1 025 到 65 535 之间。

假设主机尝试 T 次猜测有一次成功的概率为 P s , 主机试 T 次猜测没有一次成功的概率为 P u 。由于每次尝试的端口均不重复, 因此主机尝试 T 次猜测没有一次成功的概率:



1. **存在问题**

尽管使用该方法可以为位于 NAT 之后的两台主机建立 TCP 连接, 但是也存在一定的问题。首先, 需要位于公网的第三方来进行中转, 因此位于 NAT 后主机与第三方的主机之间的认证就变得极其重要。如果第三方主机不能对位于 NAT 后的主机进行有效认证, 黑

客可以很轻易地对第三方主机进行 DoS 攻击。如果位于 NAT 后的主机不能对第三方主机进行有效认证, 黑客可以中途拦截其发出的报文, 从而冒充第三方主机来篡改位于 NAT 后主机的通信消息包, 从而轻易地对位于 NAT 后主机进行攻击。其次, 由于第三方主机需要向位于 NAT 后主机发送猜测端口的 SYN 报文, 如果第三方主机同时中转处理大量位于 NAT 后主机的SYN 报文, 就会造成第三方主机的负载过重, 需要在第三方主机上对中转报文进行流量控制。

* + 1. **P2P网络中NAT的特性**

在P2P网络中，为了保证相关P2P应用可以成功穿越NAT，NAT设备应具有以下特性。

1. 端口映射的一致性

UDP和TCP打洞技术要求NAT设备对端口映射必须一致，即要求从同一内网地址和端口建立的连接映射为相同的公网地址和端口。对于锥形NAT，一般可以满足这一特性；但对于对称型NAT来说，对于内网发起的到不同地址的连接请求，将分配不同的端口号。对于这种情况，需要采用特殊的方式，如基于端口预测的方法。

1. 对于来自公网SYN包的丢弃处理

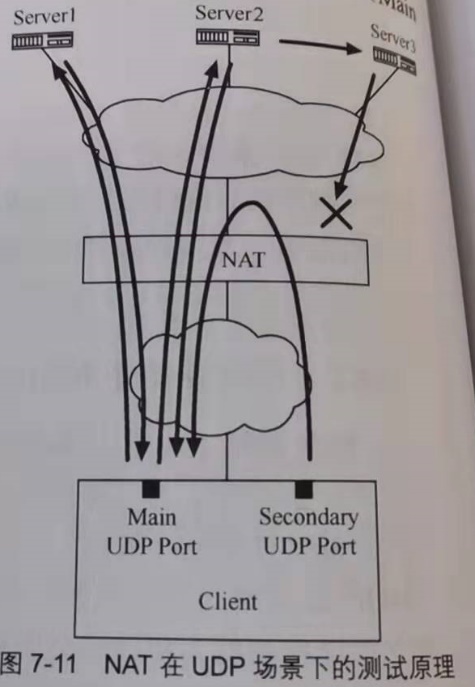
当NAT设备收到来自公网的SYN包时，而其本地的TCP连接表中并没有存储该SYN包的源地址信息，为保证TCP穿越NAT的可能性，此时NAT应直接丢弃该SYN包，而不必做任何处理。若NAT此时向SYN包的源端发送ICMP错误相应消息，源端收到后会直接导致TCP连接的失败。

1. 支持内网不同主机间的通信

在网络中存在多层NAT的情况下，为了实现TCP或UDP打洞，需要NAT设备支持内部的主机能够和内部的其他主机进行会话。对于这种功能的支持在现在大多数NAT设备中还不具备，但也只有在有多层NAT存在的场景下才需要NAT具备这种功能。随着IPV4地址的逐渐枯竭，多层NAT的应用情况越来越多，因此在今后的NAT设备中，对这种功能的支持也越来越重要。

* + 1. **NAT设备测试分析**
  1. 测试方法

对于NAT在UDP场景下的测试原理，如下图所示。



客户端采用相同的UDP端口（Main UDP Port）分别向Server 1和Server 2连续发送Ping请求消息，服务器在返回的消息中分别包含观察到的NAT为其分配的公网地址及端口号信息。如果Server 1和server 2向客户端返回相同的地址和端口信息，则说明NAT具备端口映射的一致性特征。

当Server2收到Client端的消息后，除了直接向Client发送回复消息外，还将Client的消息转发给Server3，Server3得知client的地址信息后，也向Client发送回复消息。如果NAT设备不允许来自公网的消息，则Client处将无法收到来自Server 3的消息。

为了测试NAT是否支持内网不同主机之间的通信，只需要在Client处打开另外一个端口（Secondary UDP Port），并通过此端口向Main UDP Port的公网地址发送消息，如果消息能够被Main UDP Port收到，说明NAT支持内网不同主机之间的通信。

对于NAT在TCP场景下，NAT端口映射是否一致以及是否支持内网不同主机之间的通信的测试原理同UDP类似。不同之处在于，在TCP应用场景下，需要测试NAT设备对于来自公网SYN包的处理方式。为此，当Server2收到来自Client端的请求消息后，不直接发送回复消息，而是先转发至Server3处，Server3根据Client端的公网地址，主动向Client发送SYN数据包，请求建立TCP连接。Server3等待5s。如果5s吗、后，该TCP连接请求仍在处理过程中，则Server3向Server2回复消息，并继续等待TCP连接的建立。如果在5s以内，Server3收到了ICMP错误消息表示TCP连接失败，则向Server2回复消息。收到Server3的回复消息后，Server2才向Client端发送回复消息。此回复消息中应包含Server3的地址信息。之后，Client端才可以与Server3尝试建立TCP连接。

采用这种处理流程，我们可以分析出，加入NAT设备对于来自公网SYN包采用直接丢弃的处理方式，那么Server3等待的时间内，TCP连接过程将不会终止，这样当Client端收到Server2的回复消息后，应可以成功在NAT设备上打洞从而建立与Server3的TCP连接。而如果NAT设备对于该SYN包，向Server3发送ICMP错误响应消息，则Server3将得知TCP连接建立失败，从而导致Client和Server3之间的TCP连接不会建立成功。

* 1. 测试结果

以下选取了多种NAT硬件设备进行了测试，此外，还选取了基于操作系统的NAT实现一同进行了对比。在测试中，若NAT满足端口映射的一致性，又满足对于来自公网SYN包的丢弃处理，则说明其可以实现TCP打洞穿越。对于NAT是否支持内网不同主机之间的通信，则单独测试衡量。

