

一种改进的 Chord 路由算法

姜守旭, 韩希先, 李建中

(哈尔滨工业大学 计算机科学与技术学院, 黑龙江 哈尔滨 150001)

(xxhan1981@163.com)

摘 要:如何有效地确定存储给定数据项的节点在 P2P 中非常重要。Chord 是一种比较成功的 P2P 路由算法,但是 Chord 的路由表存在严重的信息冗余。提出了一种对 Chord 的改进算法,继承了 Chord 算法简单、高效、可靠、负载平衡及开销少的优点,对 Chord 的路由表提出了改造,增加了路由表中的有效信息,提高了查询效率。

关键词:Peer-to-Peer; 分布哈希; 路由; Chord

中图分类号: TP393 **文献标识码:** A

Improvement of Chord routing algorithm

JIANG Shou-xu, HAN Xi-xian, LI Jian-zhong

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin Heilongjiang 150001)

Abstract: Routing is essential in P2P applications, and it is very important to locate the nodes that contain given data items. Chord is a successful routing algorithm, but the routing table in Chord has a terrible problem of information redundancy. To address this problem, an improved Chord algorithm was proposed which inherited the advantages of Chord, such as simplicity, efficiency, reliability, load balance and low cost. The routing table was improved to gain more useful information and get higher efficiency.

Key words: Peer-to-Peer; distributed Hash table; routing; Chord

0 引言

对等网络(P2P)是没有中央控制和分级机构的分布式网络系统,由若干个互联协作的节点构成,每个节点都同时扮演服务器与客户端的角色,它们运行相同的协议,能够意识到彼此间的存在,找到想要的资源,构成一个虚拟或实际的对等网络群体。这一特色使 P2P 系统在文件共享、分布式计算、数据发布和其他领域的应用取得了很大的成功。P2P 结构在可扩展性、实时性、可靠性和负载平衡方面具有天生优势。

P2P 系统性能的主要决定因素是系统的网络拓扑和路由协议。与 Gnutella^[1]等系统的无结构随机拓扑的网络相对应,产生了新的基于分布式哈希表(DHC)的结构化网络拓扑。

随着网络规模的增大,特别是大规模 Internet 网络的应用,对等网络的可扩展性已成为急需解决的问题。各国研究人员在支持大规模对等网络的分布式查找服务方面进行了大量的工作,提出了各种具有良好可扩展性的分布式查找服务,例如:Chord^[2]、Pastry^[3]、CAN^[4]和 Tapestry^[5]等。

MIT 提出的 Chord,想法简洁而清晰,但是该系统还存在着不少影响查询效率的问题。假设一个有 N 个节点的网络,Chord 的平均查询路径长度是 $\frac{1}{2}(\log_2 N)$,有很多的方法来对 Chord 的思想进行改进。文献[6]等提出的双向路由 Chord 是其中最具有代表性的一种改进,它的基本思想是从当前节点的顺时针和逆时针方向同时构建路由表,双向路由 Chord 的平均查询路径长度可以达到 $\frac{1}{3}(\log_2 N)$,但它存在的问题是

路由表空间是原始 Chord 的两倍。本文提出了一种对于 Chord 的改进,在不增加路由表空间的情况下,减少路由表中的冗余信息,增加相同数量的有价值信息,提高查询效率。

1 Chord 分布式查找服务

Chord 分布式查找服务提出了一种哈希函数的快速分布式计算:将文件哈希得到的键值映射到相应的节点来负责管理。如果系统中每一个数据都指定了关键字,那么数据分布式查找问题就容易解决了。

1.1 相容散列

Chord 采用了相容散列的一种变体计算方法为节点分配关键字,改善了相容散列的可扩展性。相容散列函数为每个节点和关键字分配 m 位字节的标识符,把节点和键值映射到一个大小为 2^m 环形空间上。此标识符可以用 SHA21 等散列函数产生。节点的标识符可以通过散列节点的 IP 地址产生,而关键字的标识符可以直接散列此关键字。标识符长度 m 必须足够长,这样才能保证两个节点或者关键字散列到同一个标识符上的概率小到可以忽略不计。

在相容散列中,每个关键字都保存在它的后继(successor)节点中,后继节点是节点标识符大于或等于关键字 k 标识符的第一个节点,可表示为 $\text{successor}(k)$ 。如果标识符采用 m 位二进制数表示,并且将从 0 到 $2^m - 1$ 的数排列成一个圆圈,那么 $\text{successor}(k)$ 就是从 k 开始顺时针方向距离最近的节点。

1.2 Chord 路由和文件定位

Chord 中的每个节点维护一个后继节点表和一张路由

表。后继节点表保存标识符空间中顺时针紧跟在其后的节点,用来保证查询及指针表的正确性和健壮性。路由表保存标识符空间中顺时针距当前点的距离为 2^{i-1} ($1 \leq i \leq m$) 的标识符及该标识符的后继节点。标识符用 $finger[i].start$ 表示, $finger[i].start = myid + 2^{i-1} \pmod{2^m}$ ($myid$ 是当前节点的标识符),其后继点为 $finger[i].successor$, $finger[i].interval$ 表示区间 $[finger[i].start, finger[i+1].start]$ 。

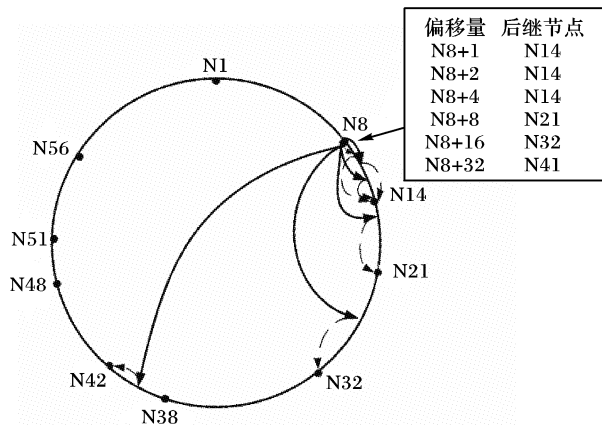


图1 Chord的数据实例

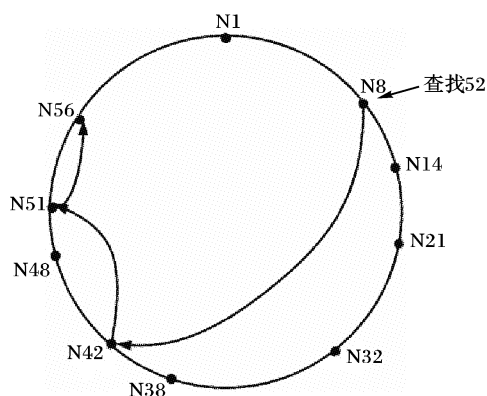


图2 Chord 查找方式

查询时,当目标 id 落于某个 $finger[i].interval$ 时,则将 $finger[i].successor$ 作为下一查询跳点(后继节点)。可见利用该表每一步至少覆盖剩余距离的一半,所以可以得到路由长度为 $O(\log_2 N)$ 的性能。

以图1为例,节点8指针表只有6个表项,第一表项标识符 $(8 + 2^0) \pmod{26} = 9$,其后继节点指向14,最后一表项标识符 $(8 + 2^5) \pmod{2^6} = 40$,后继节点指向42。该方式有两个重要的特性:1) 每个节点只需要知道一部分节点信息,而且离它越近的节点,它就知道越多的信息;2) 每个节点的指针表并不包含所有关键字的位置。例如,图1中的节点8就不知道关键字52的位置,因为节点8的后继节点信息并不直接指向关键字54的后继节点56。当节点 n 不知道关键字 k 的后继节点时,如果 n 能够找到一个节点,这个节点的标识符更接近 k ,那么这个节点将会知道该关键字的更多信息。根据这一特性, n 将查找它的指针表,找到节点标识符大于 k 的第一个节点 j ,并询问节点 j ,看 j 是否知道哪个节点更接近 k 。通过重复这个过程, n 最终将会知道 k 的后继节点。如图1所示,节点8需要查找关键字52的后继节点。由于节点8的最大指针为节点42,因此节点8将要求42去寻找关键字52的后继节点。依此类推,节点56将查找它的指针表并发现关键字52的后继节点是它本身,于是节点56将告诉节点8,节点56是它要找的节点。

2 改进的 Chord 路由算法

2.1 Chord 的不足

对于标识符为 $myid$ 的当前节点 P ,在 P 的路由表中,随着 i 的增大, $finger[i].interval$ 呈2倍增长。而相对 2^m 的标识符空间来说,节点是非常稀少的,这些点又基本上是均匀分布的(根据哈希散列函数),所以当 i 较小时, $interval$ 也较小,中间的节点数就很少,甚至没有,即前后相邻的两个或多个指针完全相同,造成不必要的重复存储,从而减少了路由表有价值的信息量的存储;相反,随着 i 的增大, $interval$ 越来越大,落于其中的节点数会越来越多。例如, $finger[m].interval$ 包含 2^{m-1} 个逻辑点。也就是说,落于标识符空间上一半 $[myid + 2^{m-1}, myid + 2^m]$ 的目标点的查询都只能交给节点 $start[m].successor$ 来处理,因为本节点没有保存位于这一半空间上的任何其他节点。

见图1Chord的数据实例,节点 P 的路由表共有6项,我们可以看到,从节点 $N8$ 开始,以 2^i 为跨度来寻找路由表中的后继节点,分别以 $2^0, 2^1, 2^2$ 为跨度,找到的后继节点都是 $N14$,也就是有接近一半的信息是冗余信息,这大大减少了节点路由表的有效信息,降低了 Chord 的查询效率。

2.2 改进的 Chord

首先分析 Chord 系统中节点路由表的冗余信息占总的路由表的比例。假设采用的是 2^m ($m = 32$) 的环形空间,网络中共有节点 10 000 个,系统采用的是相容散列来为节点分配关键字,可以近似认为节点在环上是均匀分布的。每一个节点之间的间隔是 $2^{32}/10\,000$,差不多间隔是 429 496,相当于 2^{18} ,每一个节点的路由表中共有 $m = 32$ 项的路由信息,其中有 $18/32 = 57\%$ 的信息是冗余的,如果对路由表来进行改进,去掉冗余信息,增加有效的路由信息,将大大提高系统的性能。

在 Chord 的长度为 m 的初始路由表构建完毕以后,对表中的元素进行再次操作。从路由表开始进行扫描,查找路由表中的重复信息,以图1的 Chord 系统为例,从节点 $N8$ 开始,找到了 $(t+1)$ 项的后继节点都是指向 $N14$ 的,只保留其中的第一个项,而删除其余的 t 项,然后把路由表中再增加 t 项信息,这 t 项信息取的是 $(myid + 2^{m-1}, myid + 2^m)$ 之间的信息,这样就增加了以前 Chord 中所没有的信息,加快了查询的速度,提高了查询的效率。

路由表重构方法:

- 1) 扫描路由表中的信息,标记出冗余的信息,假设共有 b 条冗余信息;
- 2) 删除路由表中的冗余信息;
- 3) 从当前节点开始的 $\frac{1}{2}$ 环开始,顺时针到当前节点的 $\frac{1}{2}$ 环上,均匀的取 b 条信息,增加到路由表中。

本文把增加的 b 项信息均匀的指向区间 $(myid + 2^{m-1}, myid + 2^m)$ 中,使得 Chord 的最大步幅由原来的可以一步跳到 Chord 环的一半的距离,到现在的可以一步跳到整个 Chord 环,这就提高了查询的速度。

现在说明从一个节点如何找到指定 ID 的文件。假设 n 是当前要开始查询的节点,它要查询一个标识符为 id 的文件 F ,存放这个文件的节点的节点标识符 $ID \geq id$ 的第一个节点,称为 F 的后继节点。 n 要查找文件 F ,就要查找 F 的后继节点,算法如下:

//节点 n 查找 id 的后继节点


```

n. find_successor(id)
n' = find_predecessor(id);
return n'. successor;
//节点 n 查找 id 的前导节点
n. find_predecessor(id)
n' = n;
while(id (n', n'. successor))
n' = n'. closest_preceding_finger(id);
return n';
//返回路由表中 id 的最近前导节点
n. closest_preceding_finger(id)
for i = m downto 1
if(finger[i]. node (n, id))
return finger[i]. node;
return n;

```

根据这一算法可以看到,去掉路由表中的重复节点并不会影响 Chord 的查询。

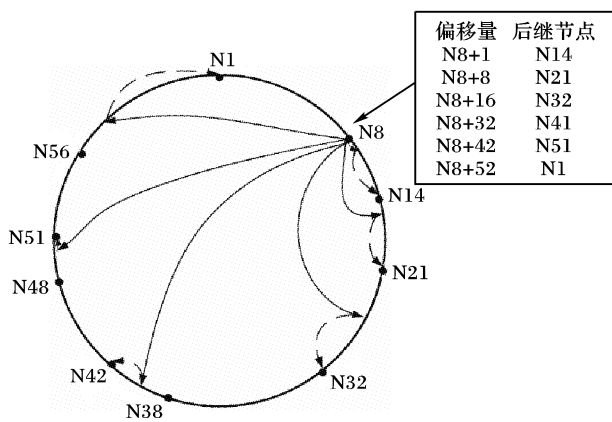


图3 改进的 Chord 的数据实例

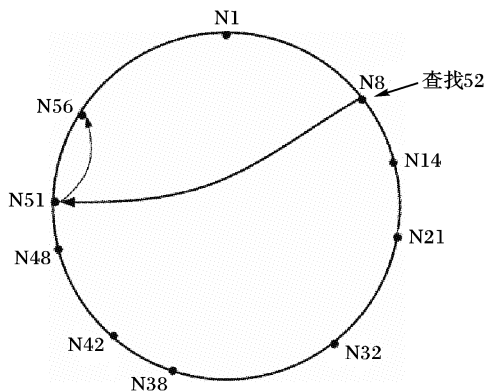


图4 改进的 Chord 的查找方式

如果当前节点要查询的文件在从当前节点开始顺时针旋转的前半个 Chord 环上,那么改进后的 Chord 查询的效率和原始 Chord 一致;如果当前节点要查询的文件从当前节点开始顺时针旋转的后半个 Chord 环上,那么改进后的 Chord 系统就可以比初始的 Chord 环有更好的性能,原本的第一步最远跳到 Chord 环的 $\frac{1}{2}$ 处,现在则可以跳得超过 $\frac{1}{2}$,并且跳的尽可能远得接近目标节点。以图 3 为例,节点 8 指针表只有 8 个表项。每个节点只需要知道一部分节点信息,而且离它越近的节点,它就知道越多的信息。如图 4 所示,节点 8 需要查找关键字 52 的后继节点。由于节点 8 现在通过利用改进的 Chord 的路由表的信息,一步就可以把查询提交到 N51,让 N51 来继续查询,下一步就找到了所要查询的节点,这比初始的 Chord 协议更快的找到了目标节点,提高了 Chord 的查询

效率。

2.3 性能分析

已知初始 Chord 的平均查询路径长度是 $\frac{1}{2}(\log_2 N) = 6$ 。

下面对改进的 Chord 的查询效率来进行分析。

假设一个大小为 2^m 环形网络,网络中共有 2^n 个节点,节点的 ID 依次是 $0, 1, 2, \dots, 2^n - 1$, 为了分析过程不至于太复杂,不失一般性,假设 $m = 32, n = 12$ 。Chord 采用相容散列的一种变体计算方法为节点分配关键字,各个节点基本上是均匀地散布在环上。两个节点之间的平均间隔是 $\frac{2^m}{2^n}$, 每个节点的路由表中的冗余信息数量 $b = \log_2\left(\frac{2^m}{2^n}\right) = 20$, 这是路由表中冗余的信息,将冗余信息删除,根据以上的方法增加 b 项有效信息。

用期望来求平均查询路径长度,对于目标节点在 Chord 环上的各种可能情况来进行综合分析, N . ID 表示 $\frac{2^m}{2^n}$ Chord 上的第 N 个节点的 ID 值。

$$length_{average} = \frac{1}{2} * \sum_{i=0}^{2^{n-1}-1} length_i$$

$length_i$ 表示目标节点在区间 $[i.ID, (i+1).ID]$ 内,从初始节点开始到达目标节点的路径长度。根据重新优化路由表的方法,当目标节点位于从初始节点开始顺时针的前半个 Chord 环上,改进的 Chord 和初始 Chord 具有相同的性能。

不失一般性,假设初始查询节点的 ID 是 0,如果目的节点的 ID 在 $(0, 2^{n-1})$ 上,查询性能和初始 Chord 一致,目的节点落在 $(0, 2^{n-1})$ 上的概率是 $\frac{1}{2}$,初始 Chord 的平均查询路径

长度是 $\frac{1}{2}(\log_2 N)$ 。我们现在在半个 Chord 环来考虑,也就是在这半个 Chord 环上的平均查询路径长度是 $\frac{1}{2}(\log_2 \frac{N}{2})$

$= \frac{1}{2}(\log_2 \frac{2^n}{2}) = 5.5$, 目标节点落在这前半个 Chord 环上的概率是 $\frac{1}{2}$, 所以 $length_{average} = \frac{1}{2} * (\sum_{i=0}^{2^{n/2}-1} length_i + \sum_{i=2^{n/2}}^{2^n-1} length_i)$, $\sum_{i=0}^{2^{n/2}-1} length_i = \frac{1}{2} * \frac{1}{2}(\log_2 \frac{N}{2}) = 2.75$ 。

下面主要是求出如果目标节点落在后半 Chord 环上的时候,它的平均查询路径长度。

在区间 $[2^{n-1}, 2^n]$ 的后半个 Chord 环上,共有 2^{n-1} 个节点,这些节点基本上都是均匀分布,彼此之间都有着基本相同的间隔。在区间 $[2^{n-1}, 2^n]$ 上,原 Chord 系统本来是没有指针指向这个区间的节点的,改进的 Chord 把原 Chord 中各个节点路由表中 b 条冗余信息去掉,增加了到区间 $[2^{n-1}, 2^n]$ 上的指针,这样,如果目标节点在这个区间内的时候,就可以减少查询跳数。

给定大小为 2^m 的环形网络,网络中共有 2^n 个节点,初始查询节点是节点 0,那么区间 $[2^{n-1}, 2^n]$ 上共有 2^{n-1} 个节点,依次是节点 $2^{n-1}, 2^{n-1} + 1, \dots, 2^n - 1$ 。增加 $b = 20$ 条有价值的路由信息后,对落在区间 $[2^{n-1}, 2^n]$ 内的目标节点进行查询,平均查询路径长度是 $\frac{1}{2} * (\sum_{i=2^{n/2}}^{2^n-1} length_i)$, 对于目标节点落在

每一个小区间上,求出从初始查询节点到目标节点的查询路径长度,然后再根据概率来求出对于整个系统的平均查询路径长度的贡献值,求和就得到了整个系统的平均路径长度。

根据以上叙述,计算目标节点在区间 $[2^{m-1}, 2^m]$ 上的一个个的小区间内的查询路径长度,得到:

$$\frac{1}{2^n} * \left(\sum_{i=2^{n/2}}^{2^n-1} length_i \right) = 2.08$$

根据:

$$length_{average} = \frac{1}{2^n} * \left(\sum_{i=0}^{2^{n/2}-1} length_i + \sum_{i=2^{n/2}}^{2^n-1} length_i \right)$$

$$\sum_{i=0}^{2^{n/2}-1} length_i = \frac{1}{2} * \frac{1}{2} (\log_2 \frac{N}{2}) = 2.75$$

则改进的 Chord 的 $length_{average} = 4.82$, 初始 Chord 的平均查询路径长度是 $length_{average} = \frac{1}{2} (\log_2 N) = 6$, 所以改进的 Chord 比初始的 Chord 有更好的查询性能。

根据以上的分析,当 m 值一定时,在节点数 2^n ($n = 8, 9, 10, 11, 12, 13$) 的情况下,计算理论上改进 Chord 的性能和初始 Chord 的性能进行比较。

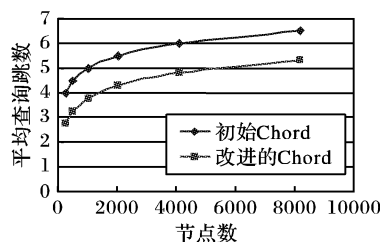


图5 理论平均路径长度比较图

根据图5 可以看到,改进的 Chord 在性能比初始 Chord 有了很大的提高。

当 n 值一定, m 值变化的时候, 因为 Chord 的环的空间变大, 节点之间的间隔变大, 使得节点路由表中的冗余信息占总路由表信息量的比例更大, 经过改进, 系统的性能将会有更大的提高。

3 仿真和试验结果

3.1 协议仿真

Chord 协议可以通过迭代和递归两种方式来实现。迭代方式, 也就是发起查询的节点从自身的路由表中得到一系列节点的信息, 每一次移动使得更接近于目标节点; 递归方式, 也就是每一个中间节点把查询提交到下一个节点上, 直到到达目标节点。仿真中采用迭代的方式来实现。

3.2 试验环境

首先认为相容散列可以把文件几乎平均的分配到各个节点上。在一个有 N 个节点和 K 个文件的网络中, 近似认为分配到一个节点上的文件数接近 K/N 。

首先考虑节点数一定、文件数增加的情况下, 改进 Chord 系统的查询效率和初始 Chord 的查询效率的对比情况。

设计一个包括 4000 个节点的网络, 其中每一个节点分别含有 10, 20, 40, 60, 80, 100 个文件, 在这个网络上进行 Chord 模拟, 得到查询效率对比图。

对每一个值独立实验 20 次, 以得到更准确的实验结果。根据图 6 可以看出, 改进的 Chord 的查询效率优于初始的

Chord 系统, 每次查询跳数比初始的 Chord 的查询跳数减少了 1, 也就是改进的 Chord 的查询时间复杂性是 $O(\log(N/2))$ 。

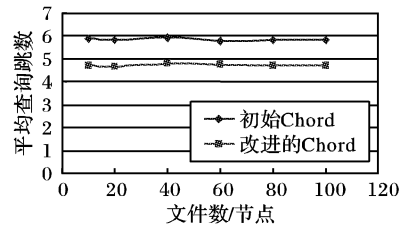


图6 查询效率对比1

然后考虑分别包括 500、1 000、2 000、3 000、4 000、5 000、6 000、8 000 个节点的网络, 每个节点平均有 100 个文件, 分别对这些网络进行测试, 得到各自网络的平均查询跳数, 将改进的 Chord 和初始 Chord 的性能比较与理论结果进行对比。对每一个值重复试验 20 次, 以得到更准确的仿真结果。

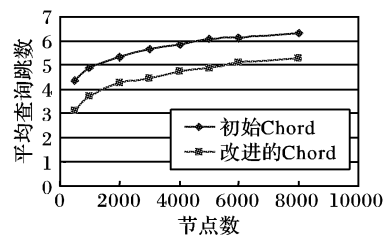


图7 查询效率对比2

根据图7 可以看出, 改进后的 Chord 比初始的 Chord 在查询性能方面有了很大的改进, 同理论分析的结果相符合。

4 结语

实验表明, 改进后的 Chord 提高了路由表中有价值信息的含量, 在一定程度上解决了原 Chord 未考虑路由表冗余信息的缺点, 提高了查询性能。另外, 还可以通过保存定位指针或采用多哈希函数来进一步提高 Chord 的性能。

当节点 C 发布文件 O 时, 在路由过程中遇到的各节点上保存 C 的 IP 地址及 O 的标识符, 即形如 $\langle O, C \rangle$ 的定位指针, 以便在查询 O 时, 一旦遇到保存有 $\langle O, C \rangle$ 的节点, 则能立刻转向 C 获得 O 。

采用多哈希函数或关键字加索引的方式将每个文件映射到 k 个节点, 可以增强文件的可用性和查找效率。节点 n 在查找某文件时, 从 k 个节点中选择离自己最近 (逻辑空间上) 的节点作为后继节点, 当发现该节点失效后再选择第二近的节点进行路由。路由表中的新增节点, 在 Chord 环上重新寻找后继节点的策略可能还可以进行修改, 以取得更好的效果。

Chord 及其他路由算法都没有对具有虚拟地址的节点加入 P2P 的特殊路由情况加以考虑, 今后还将考虑解决这方面的问题。

参考文献:

- [1] Gnutella[EB/OL]. <http://gnutella.wego.com/>, 2002.
- [2] STOICA I, MORRIS R, KARGER D, et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications [A]. ACM SIGCOMM'01[C]. San Diego, California, USA, 2001.
- [3] ROWSTRON A, DRUSCHEL P. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems[A]. Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms[C]. 2001.

(下转第 925 页)

当 A 连接 S 的时候,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为:

$$X' + \Delta z, \Delta z = f(X', z)$$

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征的分布特性的 Δz 值。因此,可以通过其分布特性来预测 Δz 所属范围,并对该范围端口进行试探的来实施穿透。

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。

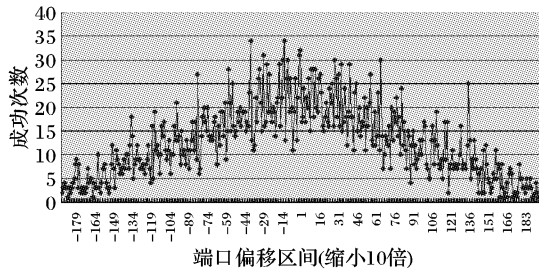


图5 穿透成功次数/端口偏移区间(缩小10倍)

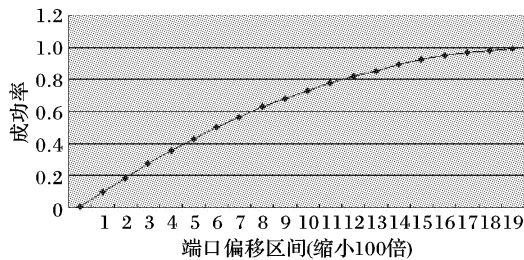


图6 成功率/端口偏移区间(缩小100倍)

3.3 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。在具体应用中,采用在探测直接连接的过程中同时进行服务器中转数据的方法,可以减少探测对应用的影响。

4 结语

在实际的网络中,常常都部署有防火墙。对于网络中即使不是端口限制 NAT,但是因为防火墙的原因,也使得内部的普通 Cone NAT 的行为变成了端口限制型 NAT。而且现在很多 NAT 与防火墙合并的软硬件设备都采用了对称 NAT。因此 P2P 应用中,常常无法用已有的稳定方法(例如 STUN)进行直接的 P2P 连接。根据对称 NAT 的特征和 NAT 具体实现时候采用的不同技术和策略,本文试着采用一些经验方法和试探方法,提出了穿透方案,并在实际网络中的测试,得到了预期的效果。此方案虽然不具有稳定的穿透性能,但能够在较多的场合尽可能的建立 P2P 连接。此方案已经在某项目中得到应用,提高了该项目平台中音视频流穿透 NAT 的能力,取得了良好的效果。

参考文献:

- [1] SRISURENTH P, NETWORKS J, EGEVANG K. Traditional IP Network Address Translator (Traditional NAT), RFC 3022 [S]. IETF, 2001.
- [2] ROSENBERG J, WEINBERGER J. STUN Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs), RFC3489 [S]. IETF, 2003.
- [3] STUKAS M, SICKER DC. An Evaluation of VoIP Traversal of Firewalls and NATs within an Enterprise Environment [J], Information Systems Frontiers, 2004, 6(3): 219 - 228.
- [4] 何宝宏. 浅析 NAT 的类型 [J]. 电信网技术, 2004, (8).
- [5] 吴煜卓, 童恒庆, 刘喜雨. 基于 UDP 协议穿透 NAT 代理的研究与设计 [J]. 微机发展, 2005, 15(1): 122 - 124.
- [6] 柯金水, 王芙蓉, 戴彬. 基于软交换的 NAT/防火墙穿透技术研究 [J]. 天津通信技术, 2004, (2): 34 - 39.
- [7] FU XD, SHI WS, AKKERMAN A. CANS: Composible, Adaptive Network Services Infrastructure [A]. Proceedings of 3rd USENIX Symposium Internet Technologies and Systems [C]. 2001.
- [8] ZHAO BY, KUBIATOWICZ JD, JOSEPH AD. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing [R]. UC Berkeley Technical Report UCB//CSD- 01-1141, 2000.
- [9] GANESAN P, MANKU GS. Optimal Routing in Chord [D]. Stanford University, SODA 2004.
- [10] DABEK F, KAASHOEK MF, KARGER D, et al. Wide-area cooperative storage with CFS [A]. ACM SOSP01 [C]. Banff, Canada, 2001.
- [11] DABEK F. A Cooperative File System [D]. Master's Thesis, MIT, 2001.
- [12] STANDARD SH. National Institute of Standards and Technology, FIPS PUB 180-1 [S]. 1995.
- [13] CANNY J. Secure Hash Algorithms [EB/OL]. www.cs.berkeley.edu/~jfc/cs174/lects/lec22/lec22.pdf, 2005.
- [14] JOVANOVIĆ MA, ANNEXSTEIN FS, BERMAN KA. Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella Network [EB/OL]. http://www.eecs.uc.edu/~mjovanov/Research/paper.html, 2001.
- [15] Kazaa [EB/OL]. http://www.kazaa.com/, 2002.
- [16] YANG B, GARCIA - MOLINA H. Efficient Search in Peer-to-Peer Networks [R]. Technical Report of Stanford Database Group, In ICDCS, 2002.
- [17] CRESPO A, GARCIA-MOLINA H. Routing indices for peer-to-peer systems [R]. Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002) [C]. 2002.
- [18] KUBIATOWICZ J, BINDEL D, CHEN Y. OceanStore: An Extremely Wide-Area Storage System [R]. U. C. Berkeley Technical Report UCB//CSD- 00-1102, 1999.
- [19] ZHAO BY, DUAN Y, HUANG L. Brocade: landmark routing on overlay networks [A]. First International Workshop on Peer-to-Peer Systems (IPTPS) [C]. Cambridge, MA, 2002.
- [20] LIBEN-NOWELL D, BALAKRISHNAN H, KARGER D. Observations on the Dynamic Evolution of Peer-to-Peer Networks [A]. Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02) [C]. Cambridge, MA, 2002.
- [21] CASTRO M, DRUSCHEL P, HU YC. Exploiting network proximity in distributed hash tables [A]. FuDiCo 2002: International Workshop on Future Directions in Distributed Computing. University of Bologna Residential Center Bertinoro (Forlì) [C]. Italy, 2002.

(上接第 921 页)