Kademlia 详解

前两天在网上看到世界知名的电骡服务器 Razorback 2 被查封、4 人被拘禁的消息，深感当前做 eMule / BitTorrent 等 P2P 文件交换软件的不易。以分布式哈希表方式(DHT，Distributed Hash Table)来代替集中索引服务器可以说是目前可以预见到的为数不多的 P2P 软件发展趋势之一，比较典型的方案主要包括：CAN、CHORD、 Tapestry、Pastry、Kademlia 和 Viceroy 等，而 Kademlia 协议则是其中应用最为广泛、原理和实现最为实用、简洁的一种，当前主流的 P2P 软件无一例外地采用了它作为自己的辅助检索协议，如 eMule、Bitcomet、Bitspirit 和 Azureus 等。鉴于 Kademlia 日益增长的强大影响力，今天特地在 blog 里写下这篇小文，算是对其相关知识系统的总结。

1. Kademlia 简述

Kademlia(简称 Kad)属于一种典型的结构化 P2P 覆盖网络(Structured P2P Overlay Network)，以分布式的应用层全网方式来进行信息的存储和检索是其尝试解决的主要问题。在 Kademlia 网络中，所有信息均以的哈希表条目形式加 以存储，这些条目被分散地存储在各个节点上，从而以全网方式构成一张巨大的分布式哈希表。我们可以形象地把这张哈希大表看成是一本字典：只要知道了信息索 引的 key，我们便可以通过 Kademlia 协议来查询其所对应的 value 信息，而不管这个 value 信息究竟是存储在哪一个节点之上。在 eMule、BitTorrent 等 P2P 文件交换系统中，Kademlia 主要充当了文件信息检索协议这一关键角色，但 Kad 网络的应用并不仅限于文件交换。下文的 描述将主要围绕 eMule 中 Kad 网络的设计与实现展开。

2. eMule 的 Kad 网络中究竟存储了哪些信息?

只要是能够表述成为字典条目形式的信息 Kad 网络均能存储，一个 Kad 网络能够同时存储多张分布式哈希表。以 eMule 为例，在任一时刻，其 Kad 网络均存储并维护着两张分布式哈希表，一张我们可以将其命名为关键词字典，而另一张则可以称之为文件索引字典。

a. **关键词字典**：主要用于根据给出的关键词查询其所对应的文件名称及相关文件信息，其中 key 的值等于所给出的关键词字符串的 160 比特 SHA1 散列，而其对应的 value 则为一个列表，在这个列表当中，给出了所有的文件名称当中拥有对应关键词的文件信息，这些信息我们可以简单 地用一个 3 元组条目表示：(文件名，文件长度，文件的 SHA1 校验值)，举个例子，假定存在着一个文件 “warcraft_frozen_throne.iso”，当我们分别以“warcraft”、“frozen”、“throne”这三个关键词来查询 Kad 时，Kad 将有可能分别返回三个不同的文件列

表，这三个列表的共同之处则在于它们均包含着一个文件名为 “warcraft_frozen_throne.iso” 的信息条目，通过该条目，我们可以获得对应 iso 文件的名称、长度及其 160 比特的 SHA1 校 验值。

b. **文件索引字典**：用于根据给出的文件信息来查询文件的拥有者(即该文件的下载服务提供者)，其中 key 的值等于所需下载文件的 SHA1 校验值(这主要是因为，从统计学角度而言，160 比特的 SHA1 文件校验值可以唯一地确定一份特定数据内容的文件)；而对应的 value 也是一个 列表，它给出了当前所有拥有该文件的节点的网络信息，其中的列表条目我们也可以用一个 3 元组表示：(拥有者 IP，下载侦听端口，拥有者节点 ID)，根据这 些信息，eMule 便知道该到哪里去下载具备同一 SHA1 校验值的同一份文件了。

3. 利用 Kad 网络搜索并下载文件的基本流程是怎样的?

基于我们对 eMule 的 Kad 网络中两本字典的理解，利用 Kad 网络搜索并下载某一特定文件的基本过程便很明白了，仍以 “warcraft_frozen_throne.iso”为例，首先我们可以通过 warcraft、frozen、throne 等任一关键词查询关键词 字典，得到该 iso 的 SHA1 校验值，然后再通过该校验值查询 Kad 文件索引字典，从而获得所有提供 “warcraft_frozen_throne.iso” 下载的网络节点，继而以分段下载方式去这些节点下载整个 iso 文件。

在上述过程中，Kad 网络实际上所起的作用就相当于两本字典，但值得再次指出的是，Kad 并不是以集中的索引服务器(如华语 P2P 源动力、 Razorback 2、DonkeyServer 等，骡友们应该很熟悉吧)方式来实现这两本字典的存储和搜索的，因为这两本字典的所有条目均分布式地存储在参与 Kad 网络的各节点中，相关文件信息、下载 位置信息的存储和交换均无需集中索引服务器的参与，这不仅提高了查询效率，而且还提高了整个 P2P 文件交换系统的可靠性，同时具备相当的反拒绝服务攻击能 力；更有意思的是，它能帮助我们有效地抵制 FBI 的追捕，因为俗话说得好：法不治众…看到这里，相信大家都能理解“分布式信息检索”所带来的好处了吧。但 是，这些条目究竟是怎样存储的呢?我们又该如何通过 Kad 网络来找到它们?不着急，慢慢来。

4. 什么叫做节点的 ID 和节点之间的距离?

Kad 网络中的每一个节点均拥有一个专属 ID，该 ID 的具体形式与 SHA1 散列值类似，为一个长达 160bit 的整数，它是由节点自己随机生成的， 两个节点拥有同一 ID 的可能性非常之小，因此可以认为这几乎是不可能的。在 Kad 网络中，两个节点之间距离并不是依靠物理距离、路由器跳数来衡量的，事实 上，Kad 网络将任意两个节点之间的距离 d 定义为其

二者 ID 值的逐比特二进制和数，即，假定两个节点的 ID 分别为 a 与 b，则有：d=a XOR b。在 Kad 中，每一个节点都可以根据这一距离概念来判断其他节点距离自己的“远近”，当 d 值大时，节点间距离较远，而当 d 值小时，则两个节点相距很近。 这里的“远近”和“距离”都只是一种逻辑上的度量描述而已；在 Kad 中，距离这一度量是无方向性的，也就是说 a 到 b 的距离恒等于 b 到 a 的距离，因为 a XOR b==b XOR a

5. 条目是如何存储在 Kad 网络中的?

从上文中我们可以发现节点 ID 与条目中 key 值的相似性：无论是关键词字典的 key，还是文件索引字典的 key，都是 160bit，而节点 ID 恰恰 也是 160bit。这显然是有目的的。事实上，节点的 ID 值也就决定了哪些条目可以存储在该节点之中，因为我们完全可以把某一个条目简单地存放在节点 ID 值恰好等于条目中 key 值的那个节点处，我们可以将满足(ID==key)这一条件的节点命名为目标节点 N。这样的话，一个查找条目的问题便被简单地转化 成为了一个查找 ID 等于 Key 值的节点的问题。

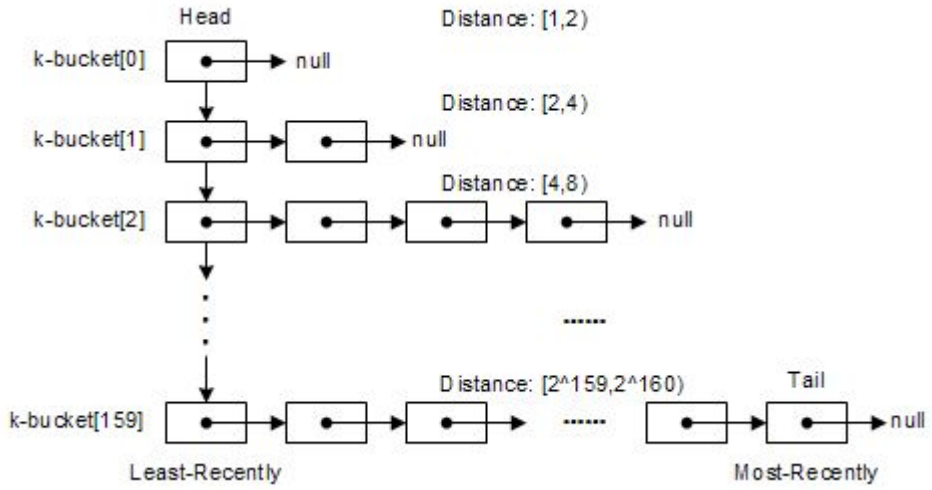
由于在实际的 Kad 网络当中，并不能保证在任一时刻目标节点 N 均一定存在或者在线，因此 Kad 网络规定：任一条目，依据其 key 的具体取值，该条目 将被复制并存放在节点 ID 距离 key 值最近(即当前距离目标节点 N 最近)的 k 个节点当中；之所以要将重复保存 k 份，这完全是考虑到整个 Kad 系统稳定性而 引入的冗余；这个 k 的取值也有讲究，它是一个带有启发性质的估计值，挑选其取值的准则为：“在当前规模的 Kad 网络中任意选择至少 k 个节点，令它们在任意 时刻同时不在线的几率几乎为 0”；目前，k 的典型取值为 20，即，为保证在任何时刻我们均能找到至少一份某条目的拷贝，我们必须事先在 Kad 网络中将该条目复制至少 20 份。

由上述可知，对于某一条目，在 Kad 网络中 ID 越靠近 key 的节点区域，该条目保存的份数就越多，存储得也越集中；事实上，为了实现较短的查询响应 延迟，在条目查询的过程中，任一条目可被 cache 到任意节点之上；同时为了防止过度 cache、保证信息足够新鲜，必须考虑条目在节点上存储的时效性： 越接近目标结点 N，该条目保存的时间将越长，反之，其超时时间就越短；保存在目标节点之上的条目最多能够被保留 24 小时，如果在此期间该条目被其发布源重 新发布的话，其保存时间还可以进一步延长。

6. Kad 网络节点需要维护哪些状态信息?

在 Kad 网络中，每一个节点均维护了 160 个 list，其中的每个 list 均被称之为一个 k-桶(k-bucket)，如下图所示。在第 i 个 list 中，记录了当前节点已知的与自身距离为 2^i~2^(i+1)

的一些其他对端节点的网络信息(Node ID，IP 地址，UDP 端口)，每一个 list(k-桶)中最多存放 k 个对端节点信息，注意，此处的 k 与上文所提到的复制系数 k 含义是一致的；每一个 list 中的对端节点信息均按访问时间排序，最早访问的在 list 头部，而最近新访问的则放在 list 的尾部。



k-桶中节点信息的更新基本遵循 Least-recently Seen Eviction 原则：当 list 容量未满(k-桶中节点个数未满 k 个)，且最新访问的对端节点信息不在当前 list 中时，其信息将直接添入 list 队 尾，如果其信息已经在当前 list 中，则其将被移动至队尾；在 k-桶容量已满的情况下，添加新节点的情况有点特殊，它将首先检查最早访问的队首节点是否仍 有响应，如果有，则队首节点被移至队尾，新访问节点信息被抛弃，如果没有，这才抛弃队首节点，将最新访问的节点信息插入队尾。可以看出，尽可能重用已有节 点信息、并且按时间排序是 k-桶节点更新方式的主要特点。从启发性的角度而言，这种方式具有一定的依据：在线时间长一点的节点更值得我们信任，因为它已经 在线了若干小时，因此，它在下一个小时以内保持在线的可能性将比我们最新访问的节点更大，或者更直观点，我这里再给出一个更加人性化的解释：MP3 文件交 换本身是一种触犯版权法律的行为，某一个节点反正已经犯了若干个小时的法了，因此，它将比其他新加入的节点更不在乎再多犯一个小时的罪……-_-b

由上可见，设计采用这种多 k-bucket 数据结构的初衷主要有二：a. 维护最近-最新见到的节点信息更新；b. 实现快速的节点信息筛选操作，也就是说，只要知道某个需要查找的特定目标节点 N 的 ID，我们便可以从当前节点的 k-buckets 结构中迅速地查出距离 N 最近的若干已知节点。

7. 在 Kad 网络中如何寻找某特定的节点?

已知某节点 ID，查找获得当前 Kad 网络中与之距离最短的 k 个节点所对应的网络信息(Node ID，IP 地址，UDP 端口)的过程，即为 Kad 网络中的一次节点查询过程(Node Lookup)。注意，Kad 之所以没有把节点查询过程严格地定义成为仅仅只查询单个目标节点的过程，这主要是因为 Kad 网络并没有对节点的上线时间作出 任何前提假设，因此在多数情况下我们并不能肯定需要查找的目标节点一定在线或存在。

整个节点查询过程非常直接，其方式类似于 DNS 的迭代查询：
a. 由查询发起者从自己的 k-桶中筛选出若干距离目标 ID 最近的节点，并向这些节点同时发送异步查询请求；
b .被查询节点收到请求之后，将从自己的 k-桶中找出自己所知道的距离查询目标 ID 最近的若干个节点，并返回给发起者；
c. 发起者在收到这些返回信息之后，再次从自己目前所有已知的距离目标较近的节点中挑选出若干没有请求过的，并重复步骤 1；
d. 上述步骤不断重复，直至无法获得比查询者当前已知的 k 个节点更接近目标的活动节点为止。
e. 在查询过程中，没有及时响应的节点将立即被排除；查询者必须保证最终获得的 k 个最近节点都是活动的。

简单总结一下上述过程，实际上它跟我们日常生活中去找某一个人打听某件事是非常相似的，比方说你是个 Agent Smith，想找小李(key)问问他的手机号码(value)，但你事先并不认识他，你首先肯定会去找你所认识的和小李在同一个公司工作的人，比方说小 赵，然后小赵又会告诉你去找与和小李在同一部门的小刘，然后小刘又会进一步告诉你去找和小李在同一个项目组的小张，最后，你找到了小张，哟，正好小李出差 去了(节点下线了)，但小张恰好知道小李的号码，这样你总算找到了所需的信息。在节点查找的过程中，“节点距离的远近”实际上与上面例子中“人际关系的密 切程度”所代表的含义是一样的。

最后说说上述查询过程的局限性：Kad 网络并不适合应用于模糊搜索，如通配符支持、部分查找等场合，但对于文件共享场合来说，基于关键词的精确查找 功能已经基本足够了(值得注意的是，实际上我们只要对上述查找过程稍加改进，并可以令其支持基于关键词匹配的布尔条件查询，但仍不够优化)。这个问题反映 到 eMule 的应用层面来，它直接说明了文件共享时其命名的重要性所在，即，文件名中的关键词定义得越明显，则该文件越容易被找到，从而越有利于其在 P2P 网络中的传播；而另一方面，在 eMule 中，每一个共享文件均可以拥有自己的相关注释，而 Comment 的重要性还没有被大家认识到：实际上，这个 文件注释中的关键词也可以直接被利用来替代文件名关键词，从而指导和方便用户搜索，尤其是当文件名本身并没有体现出关键词的时候。

8. 在 Kad 网络中如何存储和搜索某特定的条目？

从本质上而言，存储、搜索某特定条目的问题实际上就是节点查找的问题。当需要在 Kad 网络中存储一个条目时，可以首先通过节点查找算法找到距离 key 最近的 k 个节点，然后再通知它们保存条目即可。而搜索条目的过程则与节点查询过程也是基本类似，由搜索发起方以迭代方式不断查询距离 key 较近的节 点，一旦查询路径中的任一节点返回了所需查找的 value，整个搜索的过程就结束。为提高效率，当搜索成功之后，发起方可以选择将搜索到的条目存储到查询 路径的多个节点中，作为方便后继查询的 cache；条目 cache 的超时时间与节点-key 之间的距离呈指数反比关系。

9. 一个新节点如何首次加入 Kad 网络?

当一个新节点首次试图加入 Kad 网络时，它必须做三件事，其一，不管通过何种途径，获知一个已经加入 Kad 网络的节点信息(我们可以称之为节点 I)，并将其加入自己的 k-buckets；其二，向该节点发起一次针对自己 ID 的节点查询请求，从而通过节点 I 获取一系列与自己距离邻近的其他节点的信 息；最后，刷新所有的 k-bucket，保证自己所获得的节点信息全部都是新鲜的。

Kademlia

Kademlia 是一种通过分布式哈希表实现的协议算法，他是由 Petar 和 David 为非集中式 P2P 计算机网络而设计的。Kademlia 规定了网络的结构，也规定了通过节点查询进行信息交换的方式。Kademlia 网络节点之间使用 UDP 进行通讯。参与通讯的所有节点形成一张虚拟网（或者叫做覆盖网）。这些节点通过一组数字（或称为节点 ID）来进行身份标识。节点 ID 不仅可以用来做身份标识，还可以用来进行值定位（值通常是文件的散列或者关键词）。其实，节点 ID 与文件散列直接对应，它所表示的那个节点存储着哪儿能够获取文件和资源的相关信息。

当我们在网络中搜索某些值（即通常搜索存储文件散列或关键词的节点）的时候，Kademlia 算法需要知道与这些值相关的键，然后分步在网络中开始搜索。每一步都会找到一些节点，这些节点的 ID 与键更为接近，如果有节点直接返回搜索的值或者再也无法找到与键更为接近的节点 ID 的时候搜素便会停止。这种搜索值的方法是非常高效的：与其他的分布式哈希表的实现类似，在一个包含 n 个节点的系统的值的搜索中，Kademlia 仅访问 O(log(n))个节点。

非集中式网络结构还有更大的优势，那就是它能够显著增强抵御拒绝服务攻击的能力。即使网络中的一整批节点遭受泛洪攻击，也不会对网络的可用性造成很大的影响，通过绕过这些漏洞（被攻击的节点）来重新编织一张网络，网络的可用性就可以得到恢复。

内容

2 学术意义

3 在文件分享网络中的应用

1 系统细节

第一代 P2P 文件分享网络，像 Napster,依赖于中央数据库来协调网络中的查询，第二代 P2P 网络，像 Gnutella,使用泛洪来查询文件，它会搜索网络中的所有节点，第三代 p2p 网络使用分布式哈希表来查询网络中的文件，分布式哈希表在整个网络中储存资源的位置，这些协议追求的主要目标就是快速定位期望的节点。

Kademlia 基于两个节点之间的距离计算，该距离是两个网络节点 ID 号的异或，计算的结果最终作为整形数值返回。关键字和节点 ID 有同样的格式和长度，因此，可以使用同样的方法计算关键字和节点 ID 之间的距离。节点 ID 一般是一个大的随机数，选择该数的时候所追求的一个目标就是它的唯一性（希望在整个网络中该节点 ID 是唯一的）。异或距离跟实际上的地理位置没有任何关系，只与 ID 相关。因此很可能来自德国和澳大利亚的节点由于选择了相似的随机 ID 而成为邻居。

选择异或是因为通过它计算的距离享有几何距离公式的一些特征，尤其体现在以下几点：

节点和它本身之间的异或距离是 0

异或距离是对称的：即从 A 到 B 的异或距离与从 B 到 A 的异或距离是等同的

异或距离符合三角形不等式：给定三个顶点 A B C，假如 AC 之间的异或距离最大,那么 AC 之间的异或距离必小于或等于 AB 异或距离和 BC 异或距离之和.

由于以上的这些属性，在实际的节点距离的度量过程中计算量将大大降低。Kademlia 搜索的每一次迭代将距目标至少更近 1 bit。一个基本的具有 2 的 n 次方个节点的 Kademlia 网络在最坏的情况下只需花 n 步就可找到被搜索的节点或值。

1.1 路由表

为了说明简单，本部分基于单个 bit 构建路由表，如需关于实际路由表的更多信息，请看“查询加速”部分。

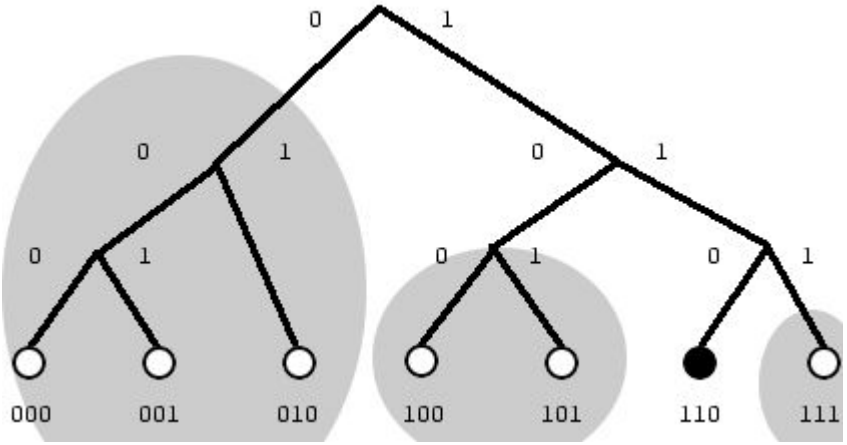
Kademlia 路由表由多个列表组成，每个列表对应节点 ID 的一位（例如:假如节点 ID 共有 128 位，则节点的路由表将包含 128 个列表），包含多个条目，条目中包含定位其他节点所必要的一些数据。列表条目中的这些数据通常是由其他节点的 IP 地址，端口和节点 ID 组成。每个列表对应于与节点相距特定范围距离的一些节点，节点的第 n 个列表中所找到的节点的第 n 位与该节点的第 n 位肯定不同，而前 n-1 位相同，这就意味着很容易使用网络中远离该节点的一半节点来填充第一个列表（第一位不同的节点最多有一半），而用网络中四分之一的节点来填充第二个列表（比第一个列表中的那些节点离该节点更近一位），依次类推。

如果 ID 有 128 个二进制位，则网络中的每个节点按照不同的异或距离把其他所有的节点分成了 128 类，ID 的每一位对应于其中的一类。

随着网络中的节点被某节点发现，它们被逐步加入到该节点的相应的列表中，这个过程中包括向节点列表中存信息和从节点列表中取信息的操作，甚至还包括当时协助其他节点寻找相应键对应值的操作。这个过程中发现的所有节点都将被加入到节点的列表之中，因此节点对整个网络的感知是动态的，这使得网络一直保持着频繁地更新，增强了抵御错误和攻击的能力。

在 Kademlia 相关的文字作品中，列表也称为 K 桶，其中 K 是一个系统变量，如 20，每一个 K 桶是一个最多包含 K 个条目的列表，也就是说，网络中所有节点的一个列表（对应于某一位，与该节点相距一个特定的距离）最多包含 20 个节点。

随着对应的 bit 位变低（即对应的异或距离越来越短），K 桶包含的可能节点数迅速下降（这是由于 K 桶对应的异或距离越近，节点数越少），因此，对应于更低 bit 位的 K 桶显然包含网络中所有相关部分的节点。由于网络中节点的实际数量远远小于可能 ID 号的数量，所以对应那些短距离的某些 K 桶可能一直是空的（如果异或距离只有 1，可能的数量就最大只能为 1，这个异或距离为 1 的节点如果没有发现，则对应于异或距离为 1 的 K 桶则是空的）。



让我们看上边的那个简单网络，该网络最大可有 2^3，即 8 个关键字和节点，目前共有 7 个节点加入，每个节点用一个小圈表示（在树的底部）。我们考虑那个用黑圈标注的节点 6，它共有 3 个 K 桶，节点 0，1 和 2（二进制表示为 000，001 和 010)是第一个 K 桶的候选节点，节点 3 目前（二进制表示为 011）还没有加入网络，节点 4 和节点 5（二进制表示分别为 100 和 101）是第二个 K 桶的候选节点，只有节点 7（二进制表示为 111）是第 3 个 K 桶的候选节点。图中，3 个 K 桶都用灰色圈表示，假如 K 桶的大小（即 K 值）是 2，那么第一个 K 桶只能包含 3 个节点中的 2 个。

众所周知，那些长时间在线连接的节点未来长时间在线的可能性更大，基于这种静态统计分布的规律，Kademlia 选择把那些长时间在线的节点存入 K 桶，这一方法增长了未来某一时刻有效节点的数量，同时也提供了更为稳定的网络。

当某个 K 桶已满，而又发现了相应于该桶的新节点的时候，那么，就首先检查 K 桶中最早访问的节点，假如该节点仍然存活，那么新节点就被安排到一个附属列表中(作为一个替代缓存).只有当 K 桶中的某个节点停止响应的时候，替代 cache 才被使用。换句话说，新发现的节点只有在老的节点消失后才被使用。

1.2 协议消息

Kademlia 协议共有四种消息。

PING 消息—用来测试节点是否仍然在线。

STORE 消息—在某个节点中存储一个键值对。

FIND_NODE 消息—消息请求的接收者将返回自己桶中离请求键值最近的 K 个节点。

FIND_VALUE 消息，与 FIND_NODE 一样，不过当请求的接收者存有请求者所请求的键的时候，它将返回相应键的值。

每一个 RPC 消息中都包含一个发起者加入的随机值，这一点确保响应消息在收到的时候能够与前面发送的请求消息匹配。

1.3 定位节点

节点查询可以异步进行，也可以同时进行，同时查询的数量由α表示，一般是 3。在节点查询的时候，

它先得到它 K 桶中离所查询的键值最近的 K 个节点，然后向这 K 个节点发起 FIND_NODE 消息请求，消息

接收者收到这些请求消息后将在他们的 K 桶中进行查询，如果他们知道离被查键更近的节点，他们就返回

这些节点（最多 K 个）。消息的请求者在收到响应后将使用它所收到的响应结果来更新它的结果列表，这个

结果列表总是保持 K 个响应 FIND_NODE 消息请求的最优节点（即离被搜索键更近的 K 个节点）。 然后消息发

起者将向这 K 个最优节点发起查询，不断地迭代执行上述查询过程。因为每一个节点比其他节点对它周边的节

点有更好的感知能力，因此响应结果将是一次一次离被搜索键值越来越近的某节点。如果本次响应结果中的节

点没有比前次响应结果中的节点离被搜索键值更近了，这个查询迭代也就终止了。 当这个迭代终止的时候，响

应结果集中的 K 个最优节点就是整个网络中离被搜索键值最近的 K 个节点（从以上过程看，这显然是局部的，而

非整个网络）

节点信息中可以增加一个往返时间，或者叫做 RTT 的参数，这个参数可以被用来定义一个针对每个被查

询节点的超时设置，即当向某个节点发起的查询超时的时候，另一个查询才会发起，当然，针对某个节点的查询

在同一时刻从来不超过α个。

1.4 定位资源

通过把资源信息与键进行映射，资源即可进行定位，哈希表是典型的用来映射的手段。由于以前的 STORE

消息，存储节点将会有对应 STORE 所存储的相关资源的信息。定位资源时，如果一个节点存有相应的资源的

值的时候，它就返回该资源，搜索便结束了，除了该点以外，定位资源与定位离键最近的节点的过程相似。

考虑到节点未必都在线的情况，资源的值被存在多个节点上（节点中的 K 个），并且，为了提供冗余，还

有可能在更多的节点上储存值。 储存值的节点将定期搜索网络中与储存值所对应的键接近的 K 个节点并且把

值复制到这些节点上，这些节点可作为那些下线的节点的补充。另外，对于那些普遍流行的内容，可能有更

多的请求需求，通过让那些访问值的节点 把值存储在附件的一些节点上（不在 K 个最近节点的范围之类）来

减少存储值的那些节点的负载，这种新的存储技术就是缓存技术。通过这种技术，依赖于请求的数量，资源

的值被存储在离键越来越远的那些节点上，这使得那些流行的搜索可以更快地找到资源的储存者。由于返回

值的节点的 NODE_ID 远离值所对应的关键字，网络中的“热点”区域存在的可能性也降低了。依据与键的距

离，缓存的那些节点在一段时间以后将会删除所存储的缓存值。 DHT 的某些实现（如 Kad）即不提供冗余

（复制）节点也不提供缓存，这主要是为了能够快速减少系统中的陈旧信息。在这种网络中，提供文件的

那些节点将会周期性地更新网络上的信息（通过 NODE_LOOKUP 消息和 STORE 消息）。当存有某个文件

的所有节点都下线了，关于该文件的相关的值（源和关键字）的更新也就停止了，该文件的相关信息也就

从网络上完全消失了。

1.5 加入网络

想要加入网络的节点首先要经历一个引导过程。 在引导过程中，节点需要知道其他已加入该网络的某个

节点的 IP 地址和端口号（可从用户或者存储的列表中获得）。假如正在引导的那个节点还未加入网络，它

会计算一个目前为止还未分配给其他节点的随机 ID 号， 直到离开网络，该节点会一直使用该 ID 号。

正在加入 Kademlia 网络的节点在它的某个 K 桶中插入引导节点（负责加入节点的初始化工作），然后向

它的唯一邻居（引导节点）发起 NODE_LOOKUP 操作请求来定位自己，这种“自我定位”将使得 Kademlia

的其他节点（收到请求的节点）能够使用新加入节点的 Node Id 填充他们的 K 桶，同时也能够使用那些查询过

程的中间节点(位于新加入节点和引导节点的查询路径上的其他节点)来填充新加入节点的 K 桶。这一自查询过

程使得新加入节点自引导节点所在的那个 K 桶开始，由远及近，逐个得到刷新，这种刷新只需通过位于 K 桶范

围内的一个随机键的定位便可达到。

最初的时候，节点仅有一个 K 桶（覆盖所有的 ID 范围），当有新节点需要插入该 K 桶时，如果 K 桶已满，K 桶

就开始分裂，（参见 A Peer-to-peer Information System 2.4）分裂发生在节点的 K 桶的覆盖范围（表现为二叉

树某部分从左至右的所有值）跨过了该节点本身的 ID 的时候。对于节点内距离节点最近的那个 K 桶，Kademlia

可以放松限制（即可以到达 K 时不发生分裂），因为桶内的所有节点离该节点距离最近，这些节点个数很可能

超过 K 个，而且节点希望知道所有的这些最近的节点。因此，在路由树中，该节点附近很可能出现高度不平衡

的二叉子树。假如K是20，新加入网络的节点ID为“xxx000011001”，则前缀为“xxx0011……”的节点可能有

21 个，甚至更多，新的节点可能包含多个含有 21 个以上节点的 K 桶。（位于节点附近的 k 桶）。 这点保证使得

该节点能够感知网络中附近区域的所有节点。（参见 A Peer-to-peer Information System 2.4）

1.6 查询加速

Kademlia 使用异或来定义距离。两个节点 ID 的异或（或者节点 ID 和关键字的异或）的结果就是两者之间的距

离。对于每一个二进制位来说，如果相同，异或返回 0，否则，异或返回 1。异或距离满足三角形不等式：任何

一边的距离小于（或等于）其它两边距离之和。

异或距离使得 Kademlia 的路由表可以建在多个 bit 之上，即可使用位组（多个位联合）来构建路由表。位组可

以用来表示相应的 K 桶，它有个专业术语叫做前缀，对一个 m 位的前缀来说，可对应 2^m 个 K 桶。(m 位的前缀

本来可以对应 2^m 个 K 桶)另外的那个 K 桶可以进一步扩展为包含该节点本身 ID 的路由树。一个 b 位的前缀可以把

查询的最大次数从 logn 减少到 logn/b.这只是查询次数的最大值，因为自己 K 桶可能比前缀有更多的位与目标键

相同，（这会增加在自己 K 桶中找到节点的机会，假设前缀有 m 位，很可能查询一个节点就能匹配 2m 甚至更多

的位组），所以其实平均的查询次数要少的多。

（参考 Improving Lookup Performance over a Widely-Deployed DHT 第三部分）

节点可以在他们的路由表中使用混合前缀，就像 eMule 中的 Kad 网络。如果以增加查询的复杂性为代价，

Kademlia 网络在路由表的具体实现上甚至可以是有异构的。

2 学术意义

尽管异或标准对于理解 Kademlia 并不是必要，但是对于协议的分析却至关重要。 异或运算形成了允许闭

合分析的循环群，为了能够预见网络的行为和正确性，其他的一些 DHT 协议和算法都要求模拟或复杂的形式

分析，而 Kademlia 并不需要，另外，把位组作为路由信息也简化了 Kademlia 算法。

3 在文件分享网络中的应用

Kademlia 可在文件分享网络中使用，通过制作 Kademlia 关键字搜索，我们能够在文件分享网络中找到我们

需要的文件以供我们下载。由于没有中央服务器存储文件的索引，这部分工 作就被平均地分配到所有的客户

端中去：假如一个节点希望分享某个文件，它先根据文件的内容来处理该文件，通过运算，把文件的内容散列

成一组数字，该数字在文件分享网络中可被用来标识文件。这组散列数字必须和节点 ID 有同样的长度，然后，

该节点便在网络中搜索 ID 值与文件的散列值相近的节点，并把它自己的 IP 地址存储在那些搜索到的节点上，也就

是说，它把自己作为文件的源进行了发布。正在进行文件搜索的客户端将使用 Kademlia 协议来寻找网络上 ID 值

与希望寻找的文件的散列值最近的那个节点，然后取得存储在那个节点上的文件源列表。由于一个键可以对应

很多值，即同一个文件可以有多个源，每一个存储源列表的节点可能有不同的文件的源的信息，这样的话，

源列表可以从与键值相近的 K 个节点获得。

文件的散列值通常可以从其他的一些特别的 Internet 链接的地方获得，或者被包含在从其他某处获得的索引文

件中。文件名的搜索可以使用关键词来实现，文件名可以分割成连续的几个关键词，这些关键词都可以散列并

且可以和相应的文件名和文件散列储存在网络中。搜索者可以使用其中的某个关键词，联系 ID 值与关键词散列

最近的那个节点，取得包含该关键词的文件列表。由于在文件列表中的文件都有相关的散列值，通过该散列值

就可利用上述通常取文件的方法获得要搜索的文件。

Kademlia: 基于异或运算的 P2P 信息系统

摘要

本文我们将描述一个在容易出错的网络环境中拥有可证实的稳定性和高性能的 点对点(P2P)系统。我们的系统使用一个很新颖的基于异或运算的拓扑来发送查询并且定位节点,这简化了算法并且使验证更加容易。 这种拓扑结构具有以下特性，它能通过交换消息传达和加强节点间的有用联系信息。本系统利用这个信息来发送平行的,异步的查询消息来对付节点的失效而不会给 用户带来超时时延。

1．介绍

本 论文描述 Kademlia，一个点对点（P2P）的<键，值>元组存储和查询系统。Kademlia 拥有许多的可喜的特点，这些特点是任何以前的 P2P 系统所无法同时提供的。它减少了节点必须发送的用来相互认识的配置消息的数量。在做键 查询的同时，配置消息将会被自动传播。 节点拥有足够的知识和灵活性来通过低时延路径发送查询请求。 Kademlia 使用平行的,异步的查询请求来避免节点失效所带来的超时时延。通过节点记录相互的存在的算法可以抵抗某些基本的拒绝服务（DoS）攻击。 最后，仅仅使用在分布式运行时间上较弱的假设（通过对现有点对点系统的测量而确认的这些假设），我们可以正式的证实 Kademlia 的许多重要特性。

Kademlia 使 用了许多点对点（P2P）系统的基本方法。 键是一个 160-bit 的隐式数量(例如，对一些大型数据进行 SHA-1 哈希的值)。 每个参与的机器都拥有一个节点 ID, 160 位的键。 <键，值>对将存储在那些 ID 与键很‘接近’的节点上，这里‘接近’当然是按照一个接近度的概念来计算的。最后，一个基于节点 ID 的路由算法使得任何人可以在一个目的键附近定位到一个服务器。

Kademlia 的 许多的优点都是得益于它使用了一个很新颖的方法，那就是用节点间的键作异或运算的结果来作为节点间的距离。异或运算是对称的，允许 Kademlia 的参与者接收来自相同分布的并且包含在其路由表中的节点的查找请求。如果没有这个性质，就像 Chord 一样，系统无法从它们收到的查 询请求中学习到有用的路由信息。更糟的是， 由于 Chord 中的运算是不对称的， Chord 的路由表更加严格。Chord 节点的查找表的每一项都必须存储精确的按 ID 域的间隔递增的节点。在这个间隔内的任何节点都比这个间隔内的某些键大，因此离键很远。相 反，Kademlia 可以在一定的间隔内发送请求给任何节点， 允许基于时延来选择路由，甚至发送平行的，异步的查询。

为 了在特定的 ID 附近定位节点，Kademlia 自始至终使用一个单程的路由算法。相反，其它一些系统使用一种算法来接近目标 ID,然后在最后的几个跳数使 用另外一种算法。在现有系统中，Kademlia 与 pastry 的第一阶段最像，（虽

然作者并没有用这种方式来描述），Kademlia 的异或运算可以使当前节点到目标 ID 的距离粗略的持续减半，以此来寻找节点。在第二阶段，Pastry 不再使用距离运算，而是改为比较 ID 的数字区别。它 使用第二种，数字区别运算作为替代。不幸的是，按第二种运算计算的接近比第一种的远得多，这造成特定节点 ID 值的中断，降低了性能，并且导致在最差行为下 的正式分析的尝试失败。

2．系统描述

每 个 Kademlia 节点有一个 160 位的节点 ID。在 Chord 系统中，ID 是通过某种规则构造出来的，但在这片文章中，为了简化，我们假设每台机器在加 入系统时将选择一个随机的 160 位值。每条节点发送的消息包含它的节点 ID， 同时允许接收者记录下发送者的存在信息，如果有必要的话。

键， 同样也是 160 位的标识符。为了发布和寻找<键，值>对，Kademlia 依赖一个概念，那就是两标识符之间的距离的概念。给定两个标识符， x 和 y，Kademlia 定义两者的位异或（XOR）的结果作为两者的距离，d(x，y)＝x⊕y。我们首先注意到异或运算是一个有意义的运算，虽然不是欧几里得 运算。很明显具有下面的性质： d(x，x)＝0；如果 x≠y，则 d(x, y)>0；任意的 x, y: d(x, y) = d(y, x)。 异或运算还满足三角性质：d(x, y) + d(y, z) ≥ d(x, z)。 这个三角性质之所以成立是基于下面这个事实： d(x, z) = d(x, y) + d(y, z); 并且任意的 a>=0, b≥0: a+b≥a⊕b。

跟 Chord 的顺时针循环运算一样，异或运算也是单向的。对于给定的一个点 x 以及距离Δ，仅有一个点 y，使得 d(x, y) = Δ。 单向性确保所有对于相同的键的查询将汇聚到相同路径中来，而不管是什么起源节点。因此，在查找路径上缓存<键，值>对可以减少‘撞车’的机 会。跟 Pastry 而不是 Chord 一样， 异或运算也是对称的。（对所有的 x 以及 y， d(x,y) = d(y,x）)

2．1．节点状态

Kademlia 节点存储互相的联系信息，以用于路由查询消息。对于任何 0 =< i < 160, 每个节点保存那些到本节点的距离为 2^i 到 2^{i+1} 之 间的节点信息列表，包括<IP 地址，UDP 端口， 节点 ID>。我们把这些列表称为 K-桶。每个 K-桶中的节点按最后联系的时间排序——最久未联系的节点放在头部，最近联系的节点放在尾部。对于比较 小的 i 值，K-桶通常是空的（因为没有合适的节点存在于系统中）。对于比较大的 i 值，列表节点数可以达到 k 的大小，k 是一个系统级别的冗余参数。k 值的选 择必须满足一个条件，那就是任意 k 个节点在一个小时内都失效的可能性很小（例如 k =20）。

图 1：以当前已在线时间的函数的形式显示了节点在接下来的一小时后继续在线的比例。X 轴代表分钟，y 轴代表那些已经在线了 x 分钟的节点中将继续在线 1 小时的比例。

当 一个 Kademlia 节点收到来自另外一个节点的任何消息（请求的或者回复的），它将更新自己的一个 K-桶，即发送节点 ID 对应的那个桶。如果发送节点已 经存在于接收者的 K-桶中，接收者会把它移到列表的尾部。如果这个节点还没有存在于对应的 K-桶中并且这个桶少于 k 个节点，则接收者把发送者插入到列表的 尾部。如果对应的 K-桶已经满了，则发送者将向该 K-桶中的最久未联系节点发送 ping 命令测试是否存在，如果最久未联系节点没有回复，则把它从列表中移 除，并把新的发送者插入到列表尾部。如果它回复了，则新的发送者信息会丢弃掉。

K- 桶 非常高效的实现了剔除最久未联系节点的策略，存活的节点将永远不会从列表中移除。这种偏向保留旧节点的做法是我们对由 Saroiu 等人收集的 Gnutella 协议的跟踪数据进行分析而得出来的。图 1 以当前已存在时间的函数的形式显示了 Gnutella 节点在一小时后继续在线的比例。一个节点存 活的时间越长，则这个节点继续存活一小时的可能性越大。通过保留存活时间最长的那些节点，K-桶中存储的节点继续在线的概率大大提高了。

K- 桶的第二个优点是它提供了对一定的拒绝服务（DoS）的攻击的抵抗。系统中不断涌入新节点并不会造成节点路由状态的更新过快。Kademlia 节点只有在旧节点离开系统时才会向 k-桶中插入新节点。

2．2．Kademlia 协议

Kademlia 协议由 4 个远程过程调用（RPC）组成：PING，STORE，FIND_NODE, FIND_VALUE。 PING RPC 测试节点是否存在。STORE 指示一个节点存储一个<键，值>对以用于以后的检索。

FIND_NODE 把 160 位 ID 作为变量，RPC 的接收者将返回 k 个它所知道的最接近目标 ID 的<IP 地址，UDP 端口，节点 ID>元组。这些元组可以来自于一 个 K-桶，也可以来自于多个 K-桶（当最接近的 K-桶没有满时）。在任何情况下，RPC 接收者都必须返回 k 项（除非这个节点的所有的 K-桶的元组加起来都少于 k 个，这种情况下 RPC 接收者返回所有它知道的节点）

FIND_VALUE 和 FIND_NODE 行为相似——返回<IP 地址，UDP 端口，节点 ID>元组。仅有一点是不同的，如果 RPC 接收者已经收到了这个键的 STORE RPC,则只需要返回这个已存储的值。

在所有 RPC 中，接收者都必须回应一个 160 位的随机 RPC ID,这可以防止地址伪造。PING 中则可以为 RPC 接收者在 RPC 回复中捎回以对发送者的网络地址获得额外的保证。

Kademlia 参 与者必须做的最重要的工作是为一个给定的节点 ID 定位 k 个最接近节点。我们称这个过程为节点查询。Kademlia 使用一种递归算法来做节点查询。查询的 发起者从最接近的非空的 K-桶中取出a个节点（或者，如果这个桶没有a项，则只取出它所知道的最接近的几个节点）。发起者然后向选定的a个节点发送平行 的，异步的 FIND_NODE RPC。a是一个系统级别的并行参数，比如为 3。

在 这个递归的步骤中，发起者重新发送 FIND_NODE 给那些从上次 RPC 中学习到的节点（这个递归可以在之前的所有的a个 RPC 返回之前开始）。在这返回 的与目标最接近的 k 个节点中，发起者将选择a个还没有被询问过的节点并且重新发送 FIND_NODE RPC 给它们。没有立即作出响应的节点将不再予以考虑除非并且直到它们作出响应。如果经过一轮的 FIND_NODE 都没有返回一个比已知最接近的节点更接 近的节点，则发起者将重新向所有 k 个未曾询问的最接近节点发送 FIND_NODE。直到发起者已经询问了 k 个最接近节点并且得到了响应，这个查询才结束。 当a＝1 时，查询算法在消息开支和检测失效节点时的时延上与 Chord 非常相似。 然而，Kademlia 可以做到低时延路由因为它有足够的灵活性来选择 k 个节点中的一个去做查询。

按 照上面的查询过程，大多数的操作都可以实现。要存储一个<键，值>对，参与者定位 k 个与键最接近的节点然后向这些节点发送 STORE RPC。另外，每个节点每个小时都会重新发布它所有的<键，值>对。这可以以高概率的把握确保<键，值>对的持续存在于系统中 （我们将会在验证概略一节中看到）。通常来说，我们还要求<键，值>对的原始发布者每隔 24 小时重新发布一次。否则，所有的<键， 值>对在最原始发布的 24 小时后失效，以尽量减少系统中的陈旧信息。

最后，为了维持<键，值>对在发布－搜索生命周期中的一致性，我们要求任何时候节点 w 拥有一个新节点 u，u 比 w 更接近 w 中的一些<键，值>对。w 将复制这些<键，值>对给 u 并且不从自己的数据库中删除。

为 了查找到一个<键，值>对，节点首先查找 k 个 ID 与键接近的节点。然而，值查询使用 FIND_VALUE 而不是 FIND_NODE RPC。 而且，只要任何节点返回了值，则这个过程立即结束。为了缓存(caching)的缘故，只要一个查询成功了，这个请求节点将会把这个<键， 值>对存储到它拥有的最接近的并且没能返回值的节点上。

由 于这个拓扑的单向性，对相同的键的以后的搜索将很有可能在查询最接近节点前命中已缓存的项。对于一个特定的键，经过多次的查找和传播，系统可能在许多的节 点上都缓存了这个键。为了避免“过度缓存”，我们设计了一个<键，值>对在任何节点的数据库中的存活时间与当前节点和与键 ID 最接近的节点 ID 之间的节点数成指数级的反比例关系。简单的剔除最久未联系节点会导致相似的生存时间分布，没有很自然的方法来选择缓存大小，因为节点不能提前知道系统将会存储多少个值。

一般来说，由于存在于节点之间的查询的通信，桶会保持不停地刷新。为了避免当没有通信时的病态情况，每个节点对在一个小时内没有做过节点查询的桶进行刷新，刷新意味着在桶的范围内选择一个随机 ID 然后为这个 ID 做节点搜索。

为 了加入到这个网络中，节点 u 必须与一个已经加入到网络中的节点 w 联系。u 把 w 加入到合适的桶中，然后 u 为自己的节点 ID 做一次节点查找。最后，节点 u 刷新 所有比最接近的邻居节点更远的 K-桶。在这个刷新过程中，节点 u 进行了两项必需的工作：既填充了自己的 K-桶，又把自己插入到了其它节点的 K-桶中。

3．验证概述

为了验证我们系统中的特有的函数，我们必须证实绝大多数的操作花费 [log n] + c 的时间开销，并且 c 是一个比较小的常数，并且 < 键，值>查找将会以很高的概率返回一个存储在系统中的键。

我们首先做一些定义。对于一个覆盖距离的范围为 [2 i , 2 i +1) 的 K- 桶， 定义这个桶的索引号为 i。定义节点的深度 h 为 160－i，其中 i 是最小的非空的桶的索引号。定义在节点 x 中节点 y 的桶高度为 y 将插入到 x 的桶的索引号减去 x 的最不重要的空桶的索引号。由于节点 ID 是随机选择的，因此高度的不统一分布是不太可能的。因此，在非常高的概率下，任意一个给定节点的高度在 log n 之内，其中 n 是系统中的节点数。而且，对于一个 ID，最接近节点在第 k 接近的节点中的桶高度很有可能是在常数 log k 之内。

下 一步我们将假设一个不变的条件，那就是每个节点的每个 K-桶包含至少一个节点的联系信息，如果这个节点存在于一个合适的范围中。有了这个假设，我们可以发 现节点的查找过程是正确的并且时间开销是指数级的。假设与目标 ID 最接近的节点的深度是 h。如果这个节点的 h 个最有意义的 K-桶都是非空的，查询过程在每 一步都可以查找到一个到目标节点的距离更接近一半的节点（或者说距离更近了一个 bit），因此在 h - log k 步后目标节点将会出现。 如果这个节点的一个 K-桶是空的，可能是这样的一种情况，目标节点恰好在空桶对应的距离范围之内。这种情况下，最后的几步并不能使距离减半。然而，搜索还是能

正确的继续下去就像键中与空桶相关的那个位已经被置反了。因此，查找算法总是能在 h - log k 步后 返回最接近节点。而且，一旦最接近节点已经找到，并行度会从a扩展到 k。寻找到剩下的 k-1 个最接近节点的步数将不会超过最接近节点在第 k 接近节点中的桶高度，即不太可能超过 log k 加上一个常数。

为 了证实前面的不变条件的正确性，首先考虑桶刷新的效果，如果不变条件成立。在被刷新后，一个桶或者包含 k 个有效节点，或者包含在它范围内的所有节点，如果 少于 k 个节点存在的话（这是从节点的查找过程的正确性而得出来的。）新加入的节点也会被插入到任何没有满的桶中去。因此，唯一违反这个不变条件的方法就是 在一个特别的桶的范围内存在 k+1 个活更多的节点，并且桶中的 k 个节点在没有查找或刷新的干涉下全部失效。然而，k 值被精确的选择以保证使所有节点在一小 时内（最大的刷新时间）全都失效的概率足够小。

实际上，失败的概率比 k 个节点在 1 小时内全都离开的概率小得多，因为每个进入或外出的请求消息都会更新节点的桶。这是异或运算的对称性产生的，因为在一次进入或外出的请求中，与一个给定节点通信的对端节点的 ID 在该节点的桶范围之内的分布是非常均匀的。

而 且，即使这个不变条件在单个节点的单个桶中的确失效了，这也只影响到运行时间（在某些查询中添加一个跳数），并不会影响到节点查找的正确性。只有在查找路 径中的 k 个节点都必须在没有查找或刷新的干涉下在相同的桶中丢失 k 个节点，才可能造成一次查找失败。如果不同的节点的桶没有重叠，这种情况发生的概率是 2^{-k*k}。否则，节点出现在多个其它的节点的桶中，这就很可能会有更长的运行时间和更低概率的失败情况。

现 在我们来考虑下<键，值>对的恢复问题。当一个<键，值>对发布时，它将在 k 个与键接近的节点中存储。同时每隔一小时将重新发布 一次。因为即使是新节点（最不可靠的节点）都有 1/2 的概率持续存活一个小时，一个小时后<键，值>对仍然存在于 k 个最接近节点中的一个上的 概率是 $1-2^{-k}$。这 个性质并不会由于有接近键的新节点的插入而改变，因为一旦有这样的节点插入，它们为了填充它们的桶将会与他们的最接近的那些节点交互，从而收到附近的它们 应该存储的<键，值>对。当然，如果这 k 个最接近键的节点都失效了，并且这个<键，值>对没有在其它任何地方缓 存，Kademlia 将会丢失这个<键，值>对。

4．讨论

我 们使用的基于异或拓扑的路由算法与 Pastry [1], Tapestry [2]的路由算法中的第一步和 Plaxton 的分布式搜索算法都非常的相似。然而，所有的这三个算法，当他们选择一次接近目标节点 b 个 bit 的时候都会产生问题（为了加速的目的）。如 果没有异或拓扑，我们还需要一个额外的算法结构来从与目标节点拥

有相同的前缀但是接下来的 b 个 bit 的数字不同的节点找到目标节点。所有的这三个算法在解 决这个问题上采取的方法都是各不相同的，每个都有其不足之处；它们在大小为 O (2 b log 2 bn) 的主表之外 都 另外 需要 一个 大小为 O (2 b) 的 次要路由表，这增加了自举和维护的开支，使协议变的更加复杂了，而且对于 Pastry 和 Tapestry 来说阻止了正确性与一致性的正式分析。Plaxton 虽然可以得到证实，但在像点对点（P2P）网络中的极易失效的环境中不太适应。

相反，Kademlia 则非常容易的以不是 2 的基数被优化。我们可以配置我们的桶表来使每一跳 b 个 bit 的速度来接近目标节点。这就要求满足一个条件，那就是任意的 $0 < j < 2^b$ 和 $0 \le i < 160/b$， 在与我们的距离为$[j2^{160-(i+1)b}, (j+1)2^{160-(i+1)b}]$ 的范围内就要有一个桶，这个有实际的项的总量预计不会超过个桶。目前的实现中我们令 b＝5。

5．总结

使 用了新颖的基于异或运算的拓扑，Kademlia 是第一个结合了可证实的一致性和高性能，最小时延路由，和一个对称，单向的拓扑的点对点(P2P)系统。 此外，Kademlia 引入了一个并发参数，a，这让人们可以通过调整带宽的一个常数参数来进行异步最低时延的跳选择和不产生时延的失效恢复。最 后，Kademlia 是第一个利用了节点失效与它的已运行时间成反比这个事实的点对点（P2P）系统。