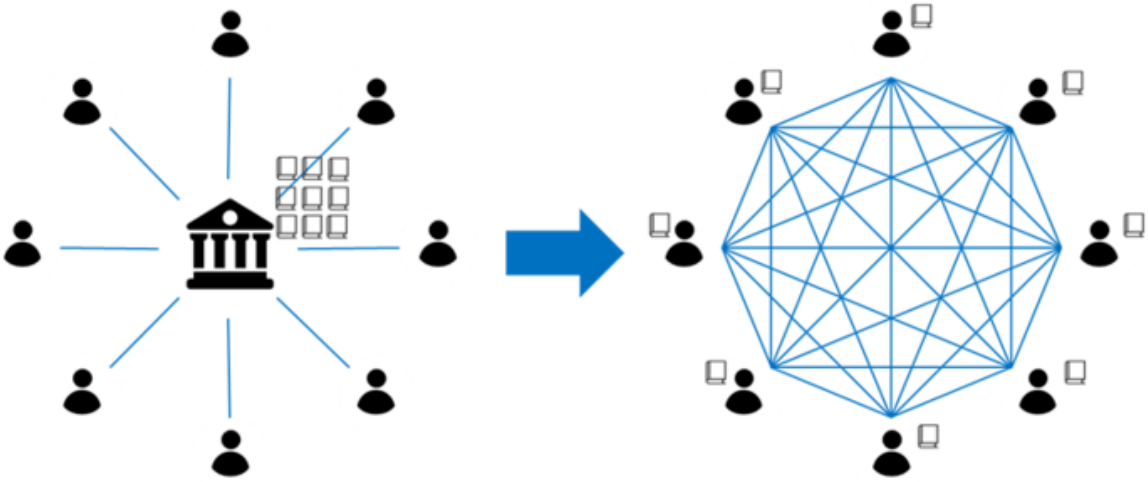
Kademlia算法

近年来，区块链技术（部分人更愿意称之为分布式账本技术）的走红将分布式技术的概念带入大众的视野。区块链技术之所以备受追捧，一方面是其展现了一种在计算机的辅助下，人类可以以无中心、无权威、无层级的方式来进行社会协作的美妙前景；另一方面，从物理上可论证，分布式的简单协议，比中心化的复杂协议更为高效。分布式技术似乎能够在带来公平的同时，还带来效率。
要理解分布式技术并不困难，因为分布式技术并不高深，但其设计上往往巧妙得令人拍手称赞。
本文介绍一种常见而巧妙的分布式技术，Kademlia算法。它也是众享互联的分布式网络安全通讯解决方案的一种技术组分。

Kademlia算法是一种分布式存储及路由的算法。什么是分布式存储？试想一下，一所1000人的学校，现在学校突然决定拆掉图书馆（不设立中心化的服务器），将图书馆里所有的书都分发到每位学生手上（所有的文件分散存储在各个节点上）。即是所有的学生，共同组成了一个分布式的图书馆。



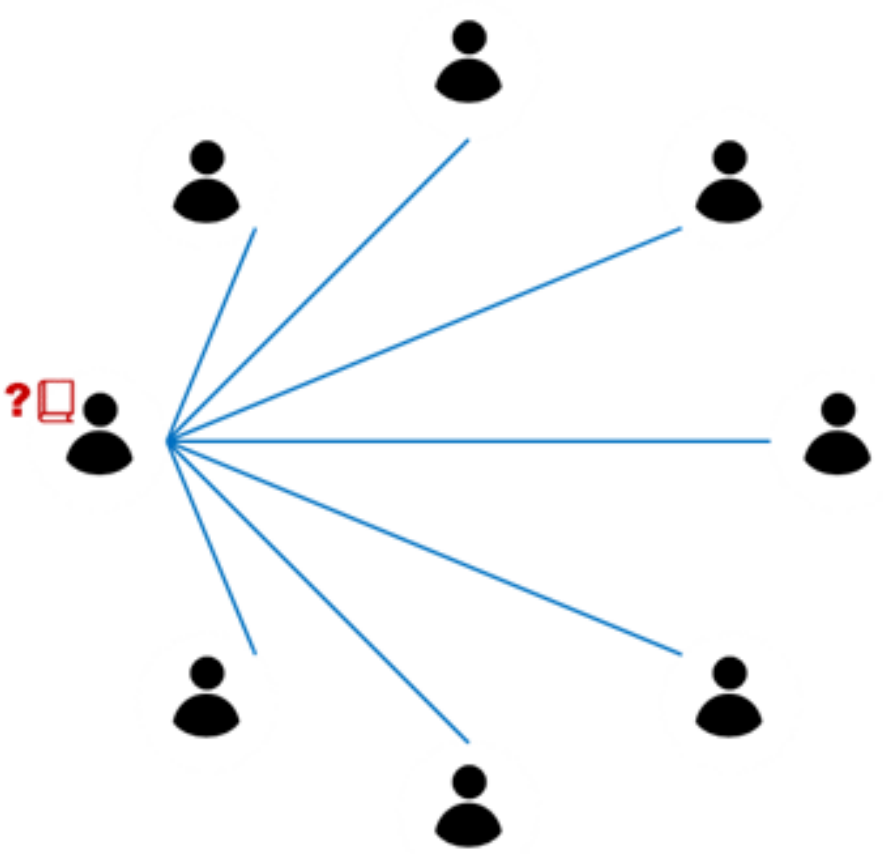
由中心图书馆到分布式图书馆

在这种场景下，有几个关键的问题需要回答。

1）关键问题

1 每个同学手上都分配哪些书。即如何分配存储内容到各个节点，新增/删除内容如何处理。

2 当你需要找到一本书，譬如《分布式算法》的时候，如何知道哪位同学手上有《分布式算法》（对1000个人挨个问一遍，“你有没有《分布式算法》？”，显然是个不经济的做法），又如何联系上这位同学。即一个节点如果想获取某个特定的文件，如何找到存储文件的节点/地址/路径。



如何寻找需要的书籍?

接下来，让我们来看看Kademlia算法如何巧妙地解决这些问题。

2）节点的要素

首先我们来看看每个同学（节点）都有哪些属性：

- 学号（Node ID，2进制，160位）
- 手机号码（节点的IP地址及端口）

每个同学会维护以下内容：

- 从图书馆分发下来的书本（被分配到需要存储的内容），每本书当然都有书名和书本内容（内容以<key, value>对的形式存储，可以理解为文件名和文件内容）；
- 一个通讯录，包含一小部分其他同学的学号和手机号，通讯录按学号分层（一个路由表，称为“k-bucket”，按Node ID分层，记录有限个数的其他节点的ID和IP地址及端口）。

根据上面那个类比，可以看看这个表格：

协议概念	类比概念
Node ID	学生的学号
IP Address, port	学生的手机号
Key	书名的hash
Value	书
路由表，K-bucket	每人手上维护的一份通讯录，通讯录里面记录着部分同学的<学号，手机号>

概念对比

（*Hash*的概念解释，可参见百度百科-哈希算法）

关于为什么不是每个同学都有全量通讯录（每个节点都维护全量路由信息）：其一，分布式系统中节点的进入和退出是相当频繁的，每次有变动时都全网广播通讯录更新，通讯量会很大；其二，一旦任意一个同学被坏人绑架了（节点被黑客攻破），则坏人马上就拥有了所有人的手机号码，这并不安全。

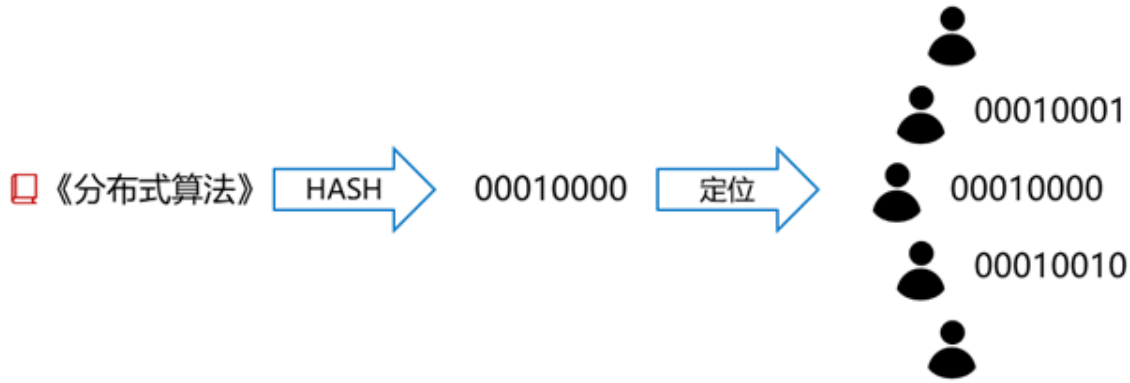
3）文件的存储及查找

原来收藏在图书馆里，按索引号码得整整齐齐的书，以一种什么样的方式分发到同学们手里呢？大致的原则，包括：1）书本能够比较均衡地分布在同学们的手里，不会出现部分同学手里书特别多、而大部分同学连一本书都没有的情况；2）同学想找一本特定的书的时候，能够一种相对简单的索引方式找到这本书。

Kademlia作了下面这种安排：

假设《分布式算法》这本书的书名的hash值是 *00010000*，那么这本书就会被要求存在学号为*00010000*的同学手上。（这要求hash算法的值域与node ID的值域一致。Kademlia的Node ID是

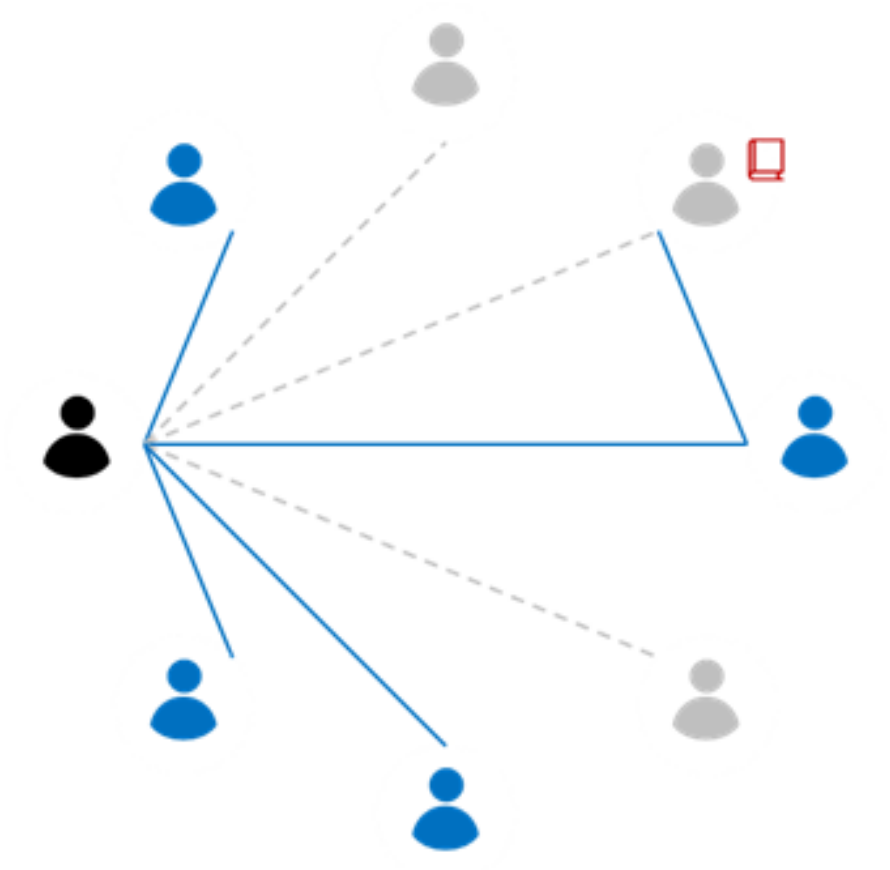
160位2进制。这里的示例对Node ID进行了简略）
但还得考虑到会有同学缺勤。万一*00010000*今天没来上学（节点没有上线或彻底退出网络），那《分布式算法》这本书岂不是谁都拿不到了？那算法要求这本书不能只存在一个同学手上，而是被要求同时存储在学号最接近*00010000*的k位同学手上，即*00010001*、*00010010*、*00010011*…等同学手上都会有这本书。
同样地，当你需要找《分布式算法》这本书时，将书名hash一下，得到 *00010000*，这个便是索书号，你就知道该找哪（几）位同学了。剩下的问题，就是找到这（几）位同学的手机号。



书籍搜索定位

4）节点的异或距离

由于你手上只有一部分同学的通讯录，你很可能并没有*00010000*的手机号（IP地址）。那如何联系上目标同学呢?



通讯录上并没有目标同学的情况

一个可行的思路就是在你的通讯录里找到一位拥有目标同学的联系方式的同学。前面提到，每位同学手上的通讯录都是按距离分层的。算法的设计是，如果一个同学离你越近，你手上的通讯录里存有ta的手机号码的概率越大。而算法的核心的思路就可以是：当你知道目标同学Z与你之间的距离，你可以在你的通讯录上先找到一个你认为与同学Z最相近的同学B，请同学B再进一步去查找同学Z的手机号。

上文提到的距离，是学号（Node ID）之间的异或距离(XOR distance）。异或是针对yes/no或者二进制的运算.

异或的运算法则为：0⊕0=0，1⊕0=1，0⊕1=1，1⊕1=0（同为0，异为1）

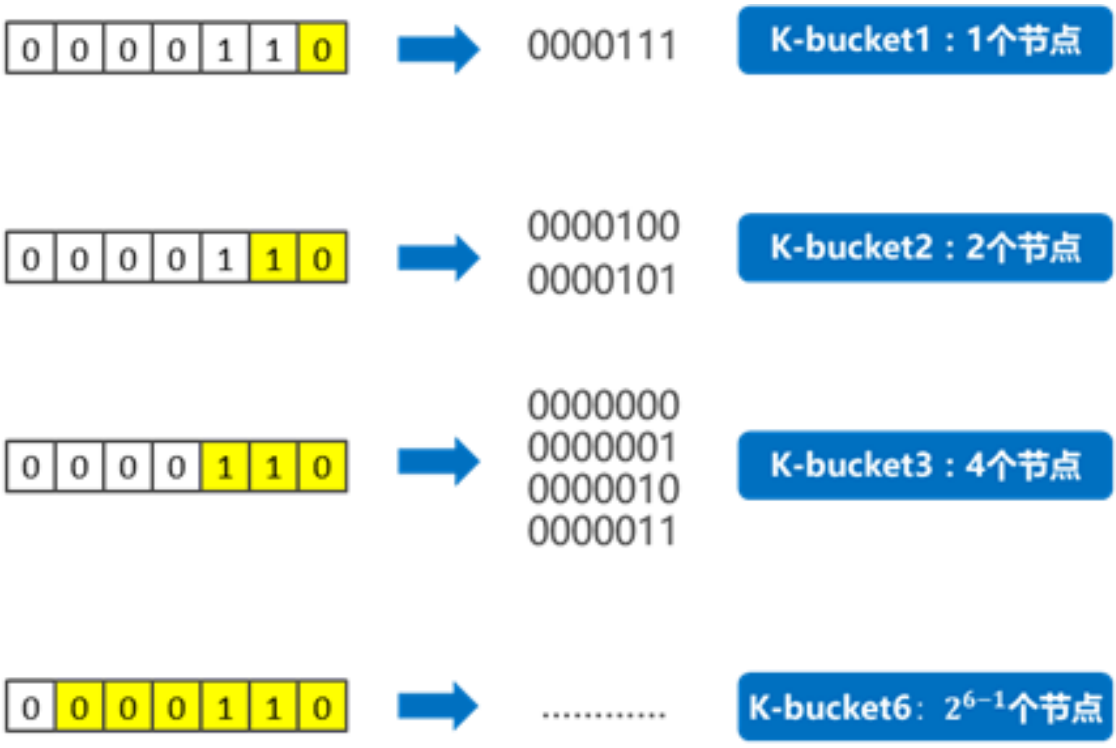
百度百科-异或

举2个例子：

*01010000*与*01010010*距离（即是2个ID的异或值）为*00000010*（换算为十进制即为2）；

*01000000*与*00000001*距离为*01000001*（换算为十进制即为

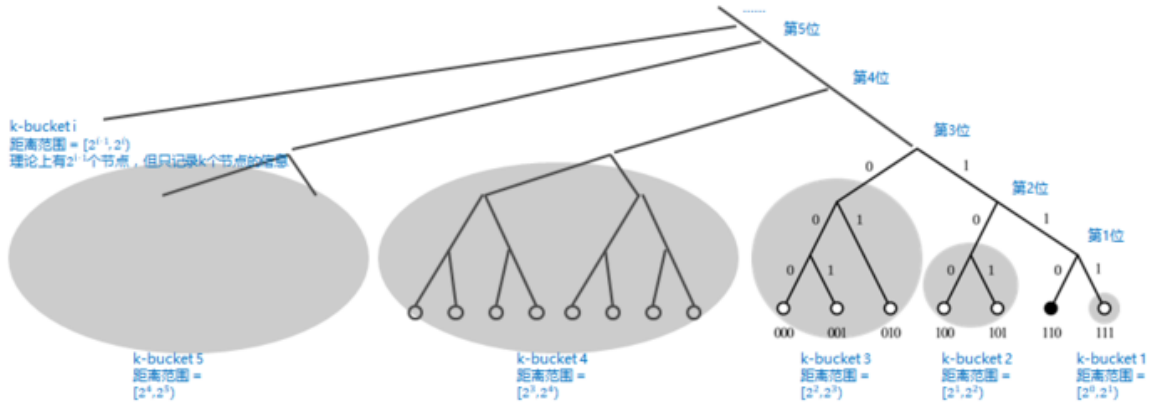
2^6+1，即65）；
如此类推。
那通讯录是如何按距离分层呢？下面的示例会告诉你，按异或距离分层，基本上可以理解为按位数分层。设想以下情景：
以*0000110*为基础节点，如果一个节点的ID，前面所有位数都与它相同，只有最后1位不同，这样的节点只有1个——*0000111*，与基础节点的异或值为*0000001*，即距离为1；对于*0000110*而言，这样的节点归为**“k-bucket 1”**；
如果一个节点的ID，前面所有位数相同，从倒数第2位开始不同，这样的节点只有2个：*0000101*、*0000100*，与基础节点的异或值为*0000011*和*0000010*，即距离范围为3和2；对于*0000110*而言，这样的节点归为**“k-bucket 2”**；
……
如果一个节点的ID，前面所有位数相同，从倒数第n位开始不同，这样的节点只有$2^{(i-1)}$个，与基础节点的距离范围为[$2^{(i-1)}$，2^i）；对于*0000110*而言，这样的节点归为**“k-bucket i”**；



按位数区分k-bucket

对上面描述的另一种理解方式：如果将整个网络的节点梳理为一

个按节点ID排列的二叉树，树最末端的每个叶子便是一个节点，则下图就比较直观的展现出，节点之间的距离的关系。



k-bucket示意图：右下角的黑色实心圆，为基础节点（按wiki百科的配图修改）

回到我们的类比。每个同学只维护一部分的通讯录，这个通讯录按照距离分层（可以理解为按学号与自己的学号从第几位开始不同而分层），即k-bucket1, k-bucket 2, k-bucket 3…虽然每个k-bucket中实际存在的同学人数逐渐增多，但每个同学在它自己的每个k-bucket中只记录k位同学的手机号（k个节点的地址与端口，这里的k是一个可调节的常量参数）。

由于学号（节点的ID）有160位，所以每个同学的通讯录中共分160层（节点共有160个k-bucket）。整个网络最多可以容纳2^160个同学（节点），但是每个同学（节点）最多只维护160 * k 行通讯录（其他节点的地址与端口）。

5）节点定位

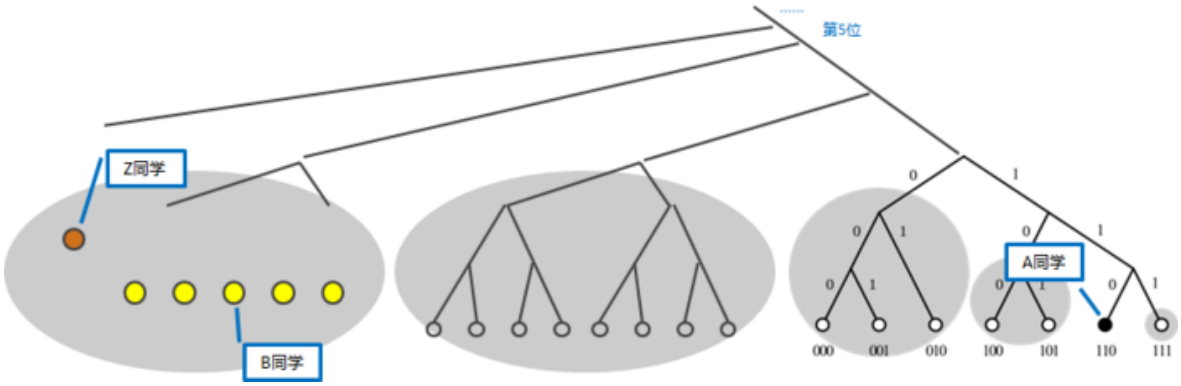
我们现在来阐述一个完整的索书流程。

A同学（学号*00000110*）想找《分布式算法》，A首先需要计算书名的哈希值，hash(《分布式算法》) = *00010000*。那么A就知道ta需要找到*00010000*号同学（命名为Z同学）或学号与Z邻近的同学。

Z的学号*00010000*与自己的异或距离为 *00010110*，距离范围在$[2^4, 2^5)$，所以这个Z同学可能在k-bucket 5中（或者说，Z同学的学号与A同学的学号从第5位开始不同，所以Z同学可能在k-

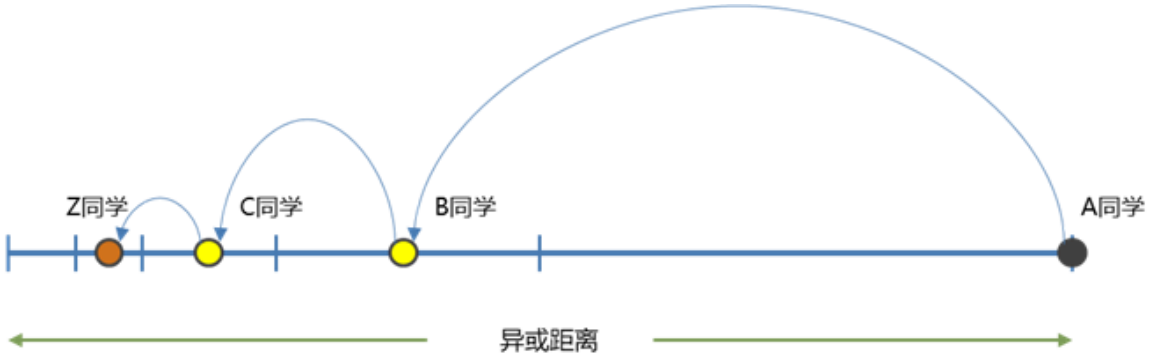
bucket 5中）。
然后A同学看看自己的k-bucket 5有没有Z同学：

- 如果有，那就直接联系Z同学要书；
- 如果没有，在k-bucket 5里随便找一个B同学（注意任意B同学，它的学号第5位肯定与Z相同，即它与Z同学的距离会小于2^4，相当于比Z、A之间的距离缩短了一半以上），请求B同学在它自己的通讯录里按同样的查找方式找一下Z同学：
- -- 如果B知道Z同学，那就把Z同学的手机号（IP Address）告诉A；
- -- 如果B也不知道Z同学，那B按同样的搜索方法，可以在自己的通讯录里找到一个离Z更近的C同学（Z、C之间距离小于2^3），把C同学推荐给A；A同学请求C同学进行下一步查找。



查询方式示意

Kademlia的这种查询机制，有点像是将一张纸不断地对折来收缩搜索范围，保证对于任意n个学生，最多只需要查询$\log_2(n)$次，即可找到获得目标同学的联系方式（即在对于任意一个有$[2^{(n-1)}, 2^n)$个节点的网络，最多只需要n步搜索即可找到目标节点）。



每次搜索都将距离至少收缩一半

以上便是Kademlia算法的基本原理。以下再简要介绍协议中的技术细节。

6）算法的三个参数：keyspace，k和α

- keyspace
- -- 即ID有多少位
- -- 决定每个节点的通讯录有几层
- k
- -- 每个一层k-bucket里装k个node的信息，即<node ID, IP Adress, port>
- -- 每次查找node时，返回k个node的信息
- -- 对于某个特定的data，离其key最近的k个节点被会要求存储这个data
- α
- -- 每次向其他node请求查找某个node时，会向α个node发出请求

7）节点的指令

Kademlia算法中，每个节点只有4个指令

- PING
- -- 测试一个节点是否在线
- STORE
- -- 要求一个节点存储一份数据
- FIND_NODE
- -- 根据节点ID查找一个节点
- FIND_VALUE
- -- 根据KEY查找一个数据，实则上跟FIND_NODE非常类似

8)k-bucket的维护及更新机制

- 每个bucket里的节点都按最后一次接触的时间倒序排列
- 每次执行四个指令中的任意一个都会触发更新
- 当一个节点与自己接触时，检查它是否在K-bucket中
- -- 如果在，那么将它挪到k-bucket列表的最底（最新）
- -- 如果不在，PING一下列表最上面（最旧）的一个节点
- -- a) 如果PING通了，将旧节点挪到列表最底，并丢弃新节点
- -- b) 如果PING不通，删除旧节点，并将新节点加入列表

该机制保证了任意节点加入和离开都不影响整体网络。

9）总结

Kademlia是分布式哈希表（Distributed Hash Table, DHT）的一种。而DHT是一类去中心化的分布式系统。在这类系统中，每个节点（node）分别维护一部分的存储内容以及其他节点的路由/地址，使得网络中任何参与者（即节点）发生变更（进入/退出）时，对整个网络造成的影响最小。DHT可以用于构建更复杂的应用，包括分布式文件系统、点对点技术文件分享系统、合作的网页高速缓存、域名系统以及实时通信等。

Kademlia算法在2002年由Petar Maymounkov 和 David Mazières 所设计，以异或距离来对哈希表进行分层是其特点。Kademlia后来被eMule、BitTorrent等P2P软件采用作为底层算法。Kademlia可以作为信息安全技术的奠基之一。

Kademlia的优点在于：

- 对于任意一个有[$2^{(n-1)}$,2^n)个节点的网络，最多只需要n步搜索即可找到目标节点；
- K-bucket的更新机制一定程度上保持了网络的活性和安全性。

Kademlia作为一种高效的分布式路由算法，还可以应用在分布式网络中的安全通讯。众享互联的分布式网络安全解决方案，将单点对单点的信息传输方式改为在多节点的分布式网络中多路径传输。采用Kademlia这种DHT算法，使得物联网设备的通讯信息在

分布式网络中能够更快更准确地到达目标设备，并并大大提高了网络攻击者彻底阻断通讯路径的难度，令物联网设备通讯的安全性极大地提高。