

4.3 无结构P2P网络（第二代）

◆ 过程：洪泛请求模式

- 每个Peer的请求直接广播到连接的Peers
- 各Peers又广播到各自连接的Peers
- 直到收到应答或达到最大洪泛步数(典型5-9步)

◆ 特点

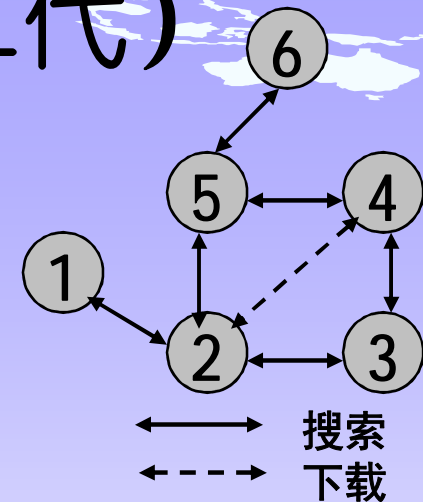
- 可智能发现节点，完全分布式
- 大量请求占用网络带宽，可扩展性并不一定最好，

◆ 协议

- Gnutella/KaZaA/eDonkey/Freenet使用该模式
- 消息协议：用于节点间相互发现和搜索资源
- 下载协议：用于两节点间传输文件（使用标准的HTTP协议: GET）

◆ 改进

- Kazaa 设立Super-Peer客户软件，以集中大量请求
- Cache最近请求



◆ 何为无结构或结构化P2P

- 根本区别在于每个peer所维护的邻居是否能够按照某种全局方式组织起来以利于快速查找。

◆ 无结构网络优点

- 将重叠网络看着一个完全随机图，结点之间的链路不遵循某些预先定义的拓扑来构建。
- 容错性好，支持复杂查询
- 结点频繁加入和退出，但对系统的影响小。

◆ 无结构网络的发展阶段

- 首先，文件交换服务以Napster和BT独领风骚，其技术是建立一个大型的集中化：但七大唱片公司把Napster公司推上法庭，三年后法院最终判定Napster侵权。
- 其次，分散式服务以Kazaa和国内迅速崛起的POCO为代表。随机选出品质较优的用户来作为节点服务器，从其上获得Peerlist，下载方法也越来越进步。

4.3.1 Gnutella

◆ 背景

- 2000.3诞生于Nullsoft公司，由MP3播放软件WinAmp的设计者Justin Frankel和Tom Peper发明
- 3.14日Napster版权案出现后，其母公司AOL（America On Line）在该软件于Gnutella网站上公布仅1.5小时后就关闭了该网站
- 但就在这1.5小时几千用户MP3迷下载了Gnutella，并将其公开、改造和克隆，保留下来。

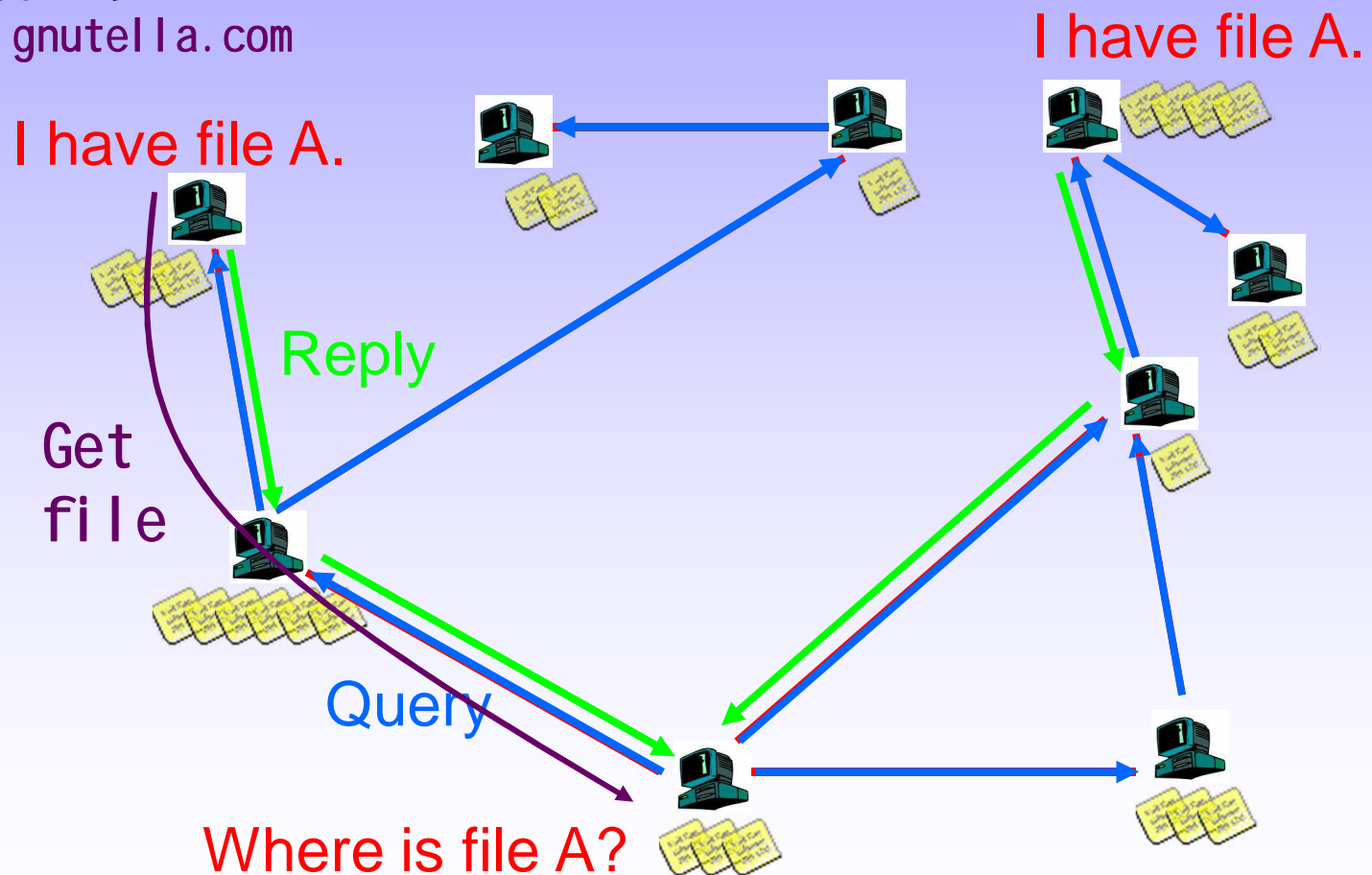
◆ 现状：Gnutella更多指无结构P2P网络协议

- gnutella.wego.con: 改造或克隆软件
- www.gnutellaworld.net: 交换各种相关信息
- Rfc-gnutella.sourceforge.net: 协议文档0.4，0.6建议使用UltraPeer
- www.Gnutella.com: 商业网站，各种应用软件集合和联盟

认识Gnutella



- Gnutella是**开源码**
- 衍生出Windows/Linux平台下诸多客户端
- 最新Windows平台下的Phex
- 下载地址: www.gnutella.com



Gnutella原理

◆ 纯分布式对等

- 每个Peer即使服务器也是客户机
- 每个Peer监视网络局部的状态信息，相互协作

◆ 工作过程

- 原始加入：必须首先连接到一个“众所周知”几乎总是在线的【或称中介、**自举**、入口】节点（功能同一般Peer），进入Gnutella网
- 查询和应答消息采用**广播或回播**（Back-propagate）机制
 - ☞ 每条消息被一个全局唯一**16字节随机数编码为GUID（128 bit）**
 - ☞ 每个节点**缓存最近路由的消息**，以支持回播或阻止重广播
 - ☞ 每条消息都有一个**TTL数**，每跳一次减少一次

Gnutella典型消息

◆ 组成员消息

- Ping: 新节点**加入网络**时用=I am here !; 探测其它节点=Are you there?
- Pong: 收到ping后决定是否回播**pong**消息, 然后将**ping**广播给邻居节点 (含IP地址、端口号、共享文件数量和大小) =Yes, I am here

◆ 查询消息

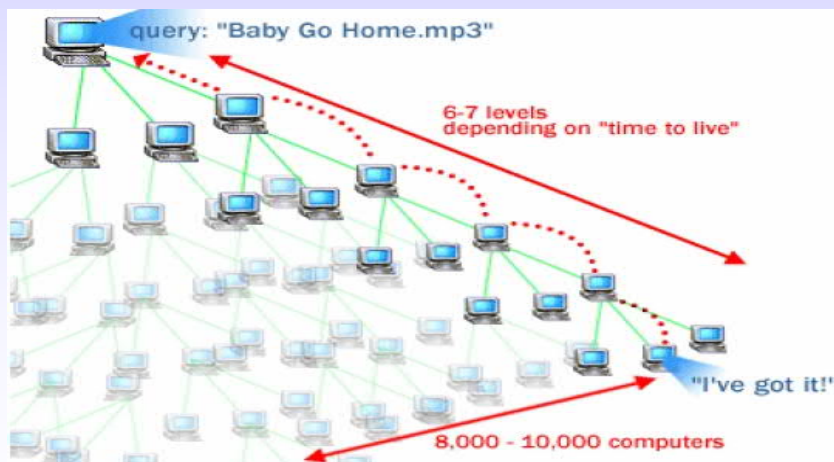
- Query: 指定查询**文件和响应速度**等信息, 每个查询有唯一ID= I am looking for...
- Query Response: **回应包含IP地址、端口号、位置和带宽**等信息, 以及命中节点的NodeID, 本消息沿来路回播=Your wanted file is here...

◆ 文件传输消息

- Get: **获取文件** = I'll get...from you
- Push: 请求Firewall后的文件拥有节点, **主动**建立连接把文件**上传**给自己 = I can't get...from you ,so you push it to me

Gnutella的问题与改进

- ◆ 洪泛广播加重网络带宽**负担**，受TTL限制，消息只能达到**一定范围**，这又导致有些文件**不能查询到**？
- ◆ 基本呈**幂率分布**：有**连接数L**的节点数占网络总节点数的**%比**，正比于 L^{-a} （显然L越大，占比越低） Gnutella的 $a=2.3 < 3$ ，节点随机失效的容错性比较高 → 但是不能防止某些节点失效而使Gnutella分裂。
- ◆ 改进：**分层P2P**=增加**超级节点**负责查询消息的路由，构成**P2P骨干网**，叶节点只是通过**超级节点代理**接入



最初的Gnutella采用的Flooding搜索算法示意图

BearShare

Hosts | Search | Downloads | Uploads | Monitor | Statistics

Host	Origin	Status	Time	Shared	% In	Out	Dup	Bandwidth	Version
enigma...	Australia	Auto	1m		65	34	6	57 Kbps	
adsl-20...	BellSouth	Auto	1m	242/983 MB	54	45	4	20 Kbps	BearSh..
u37n21...	Canada	Auto	1m	2/20 MB	78	21	3	31 Kbps	
006man...	Charter Communications	Auto	1m		27	72	9	8.8 Kbps	
bhc2.res...	Cornell University	Auto	2m	54/208 MB	73	26	2	144 Kbps	
ceres.as...	Croatia/Hrvatska	Auto	2m		47	52	5	24 Kbps	
us-6132...	Florida State University	Auto	1m	93/346 MB	49	50	5	66 Kbps	
nas1-83...	France	Auto	1m	23/1.0 MB	47	52	5	4.8 Kbps	
pc09.m...	Germany	Auto	1m		77	22	2	104 Kbps	
hive.tel...	Latvia	Auto	1m		75	24	4	77 Kbps	
fcalabro...	Princeton University	Auto	2m		59	40	3	30 Kbps	
cs16292...	RoadRunner	Auto	1m		71	28	2	33 Kbps	
h24-64-8...	Shaw Cable	Auto	2m		52	47	3	29 Kbps	
36-BAR...	Spain	Auto	2m		22	77	1	5.5 Kbps	
camp05...	Sweden	Auto	1m		60	39	7	47 Kbps	
dt195-92...	University of Oregon	Auto	1m		75	24	2	36 Kbps	

To connect manually, enter the IP address or host name here

Auto connect to: 20 hosts

Accept incoming hosts: ☒

Hosts: 22

Bandwidth: 845 Kbps

About... Community... Setup... Exit Help

采用第二代Gnutella协议最经典的软件-Bearshare

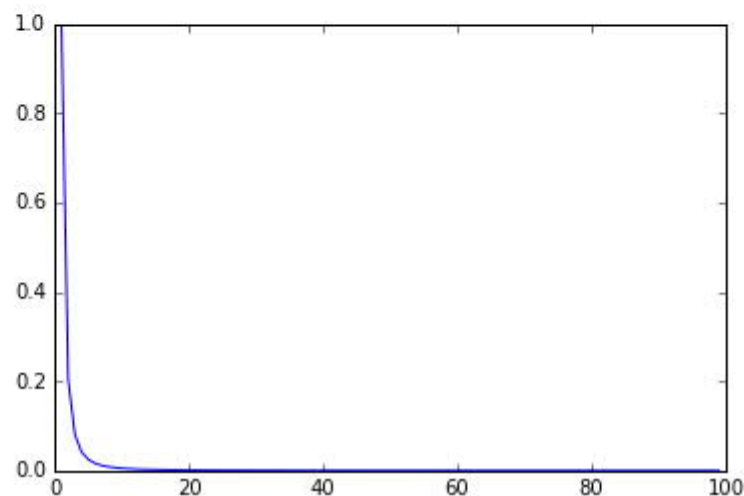
Gnutella的问题与改进

- ◆ 基本呈**幂率分布**：有**连接数L**的节点数占网络总节点数的**%比**，正比于 L^{-a} （显然L越大，占比越低） Gnutella的 $a=2.3 < 3$

```
In [7]: %pylab inline
L = range(100)[1:]
print L
plot(L, [i**(-2.3) for i in L])
```

Populating the interactive namespace from numpy and matplotlib
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]

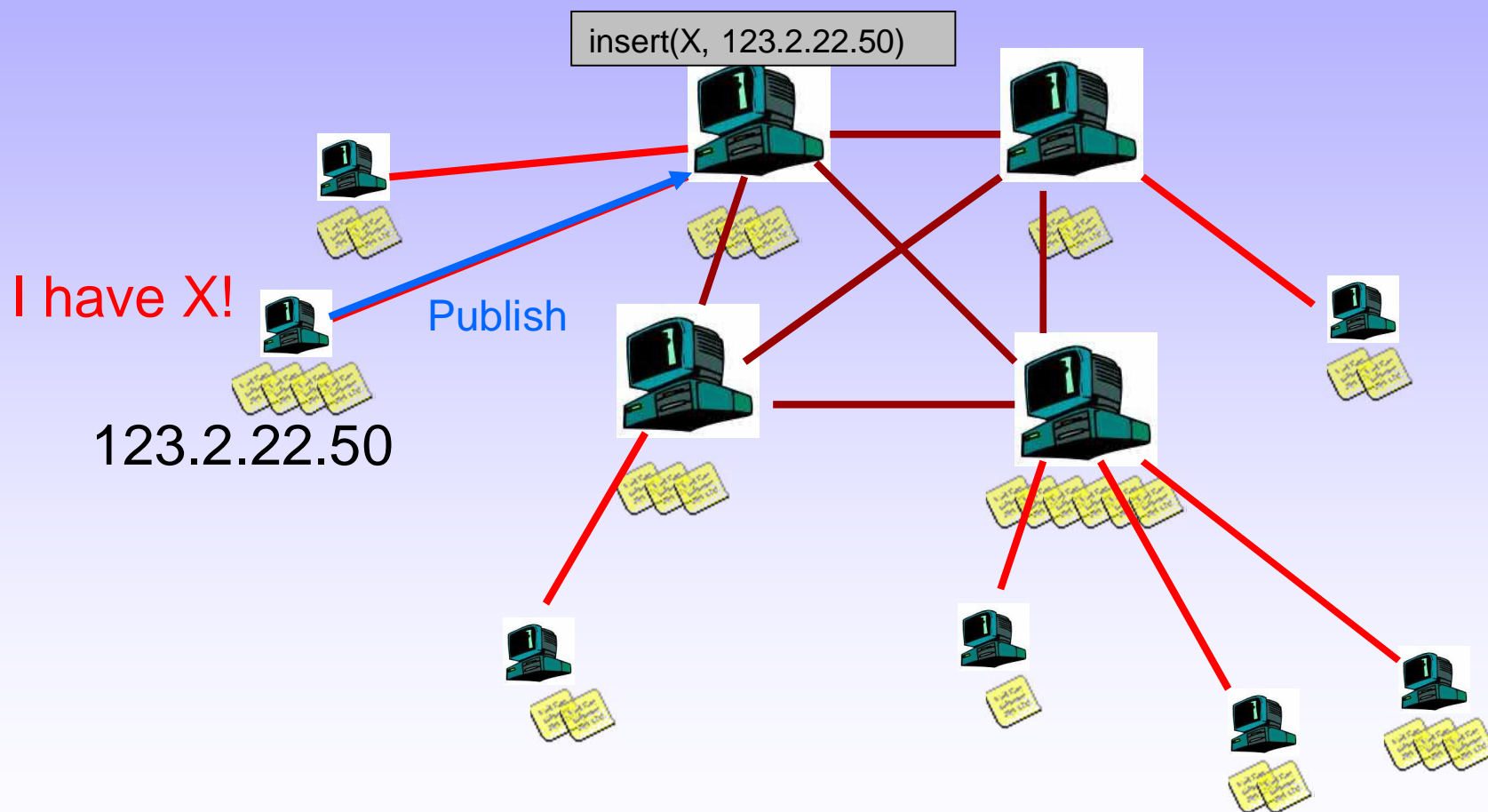
Out[7]: [<matplotlib.lines.Line2D at 0x7844c50>]



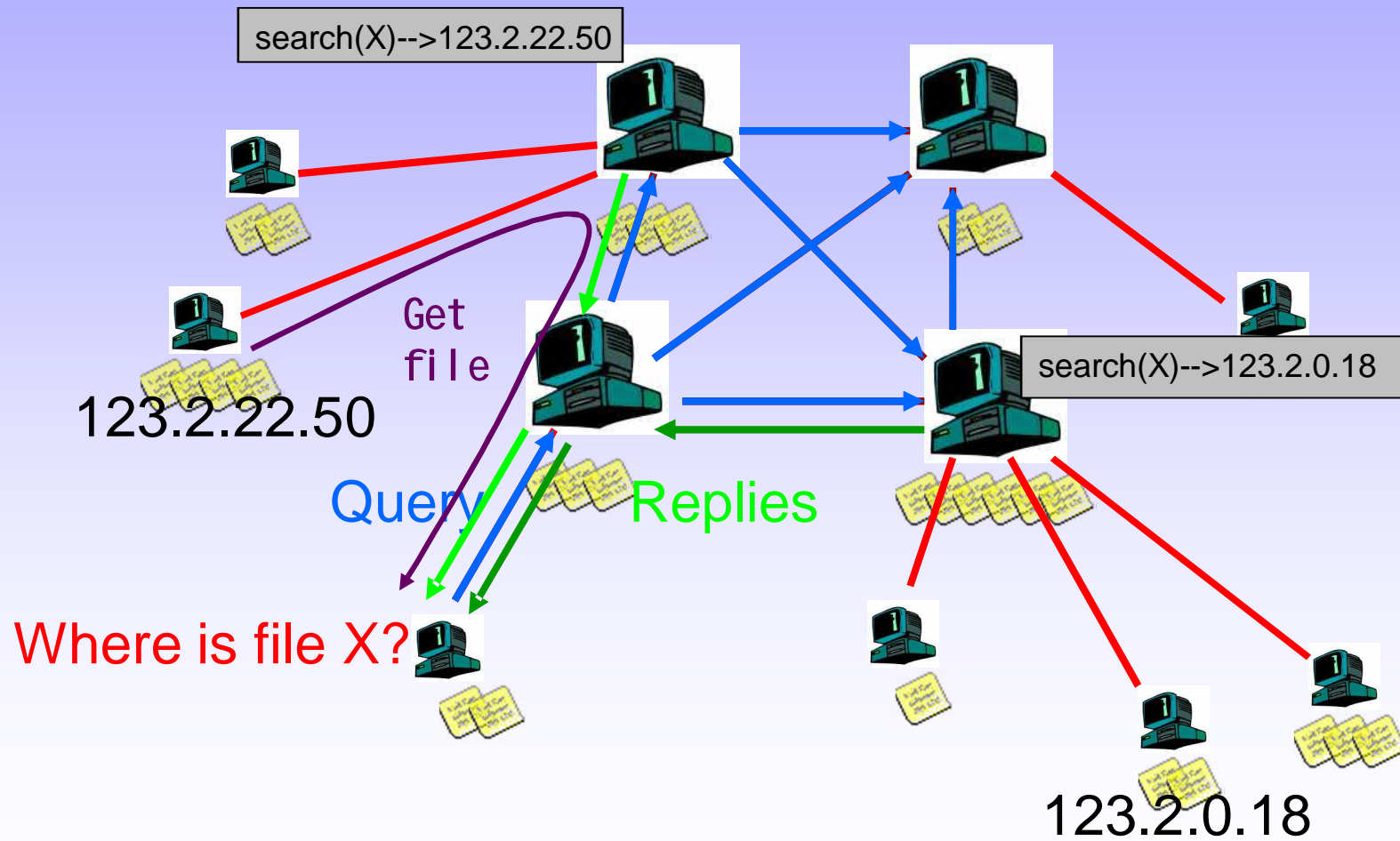
4.3.2 基于超节点的KaZaA

- ◆ 2000.7, 斯堪的纳维亚的Ni kl as和丹麦的Fri is开发
 - Ni kl as是著名P2P企业家, 在KaZaA之后, 创办了
 - Jol tid公司: 推广P2P解决方案和P2P流量优化技术
 - Al tnet公司: 第一个安全应用P2P网络, 发行数字版权管理许可证
 - Skype公司: 全球第一家实时语音通信公司
- ◆ 基于FastTrack协议
 - 比Gnutel l a早引入SuperNode
 - KaZaA是专有协议, 对消息加密, 存在超级和普通两类节点
 - 超级: 高带宽、高处理能力、大存储容量、不受NAT限制
 - 普通: 低带宽、低处理能力、小存储容量、受NAT限制
- ◆ 加入、上载与查询
 - 普通节点选择一超级节点作为父节点加入, 并维持半永久TCP连接
 - 将自己贡献的文件元数据、描述符上传给它, 并生成Hash值
 - 父超节点根据文件描述符关键字查询, 返回文件所在IP地址+元数据

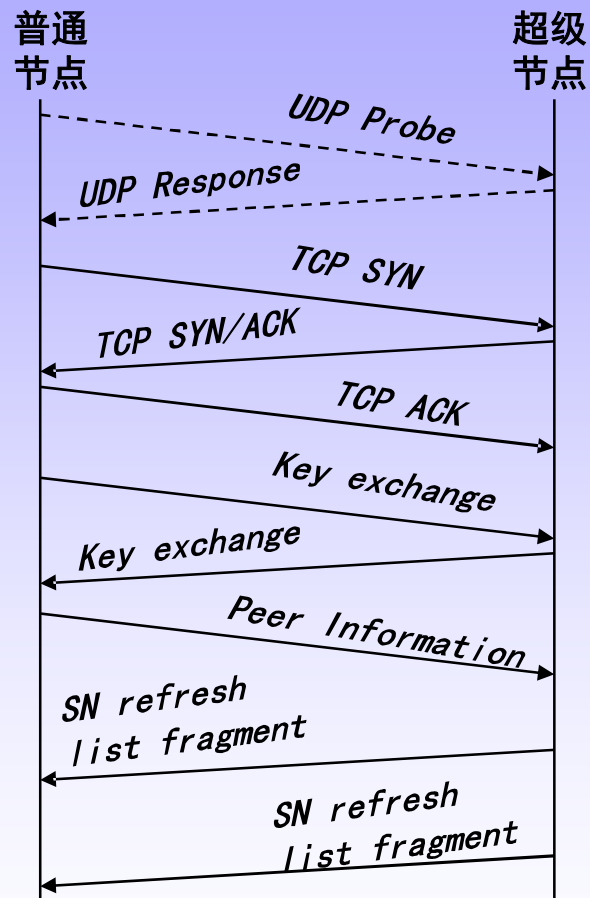
KaZaA共享文件过程



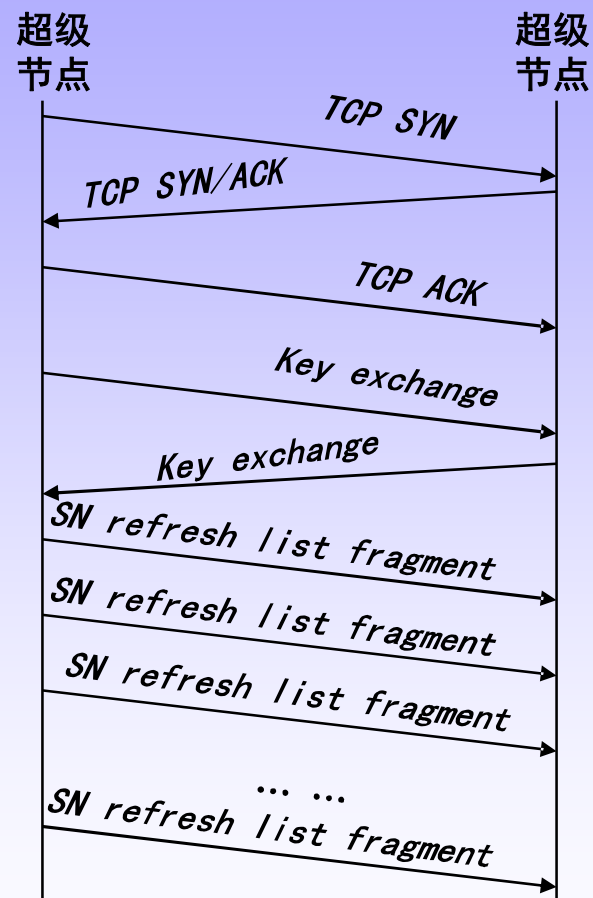
KaZaA原理



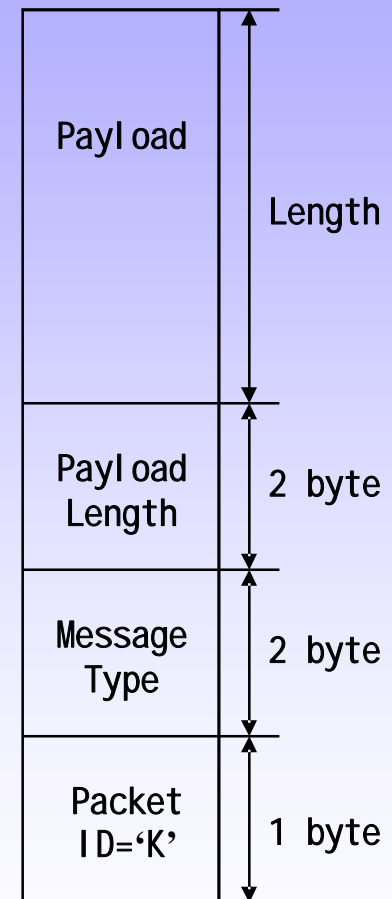
KaZaA连接的建立和消息格式



普通到超级连接的建立



超级节点间连接的建立



消息格式

4.3.3 eDonKey/eMule/Overnet

◆ 背景

- eDonKey, 2000年, Jed McCaleb创立
 - ☞ 与BT类似, 文件分块下载; 内容Hash作完整性验证, 服务器为核心
 - ☞ BT是基于文件、由 Tracker服务器来查询、搜索和跟踪用户; 但 eDonKey 是基于用户的类似KaZaA的超级节点。
- eMule, 2002.5.13, Merkey因不满eDonKey客户端
 - ☞ 在eDonKey加入新功能、优化图形界面
- Overnet是一个独立的分布式搜索应用
 - ☞ 被eDonKey整合到自己的体系中

◆ 特点

- 分块下载的双层无结构P2P网络

eDonKey 结构与过程

◆ 结构

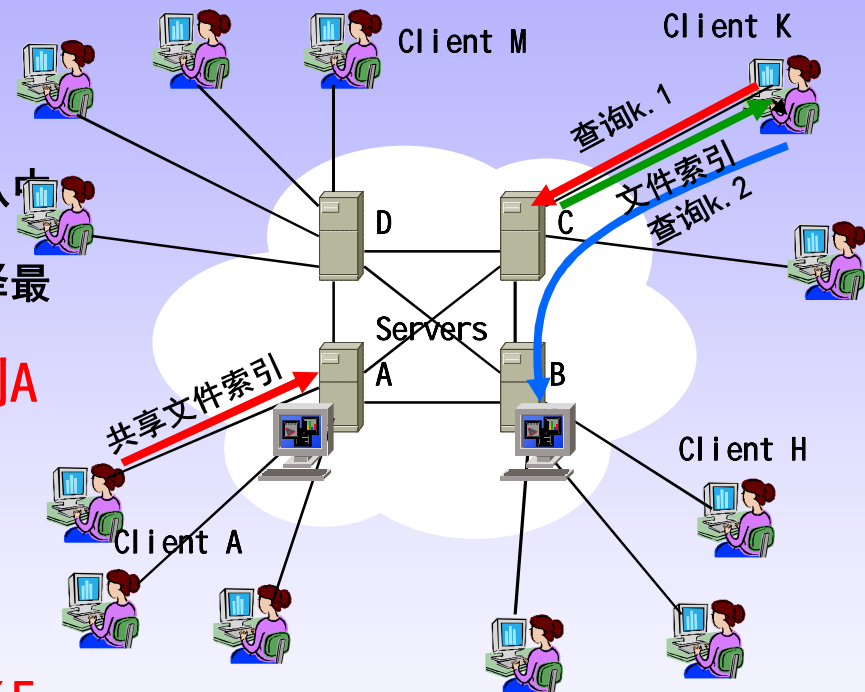
- 服务器层S或超节点+客户端层C
- S间交换文件索引和服务器列表
- 每个C连接到一个S进行文件查询和S列表更新

◆ 加入与查询

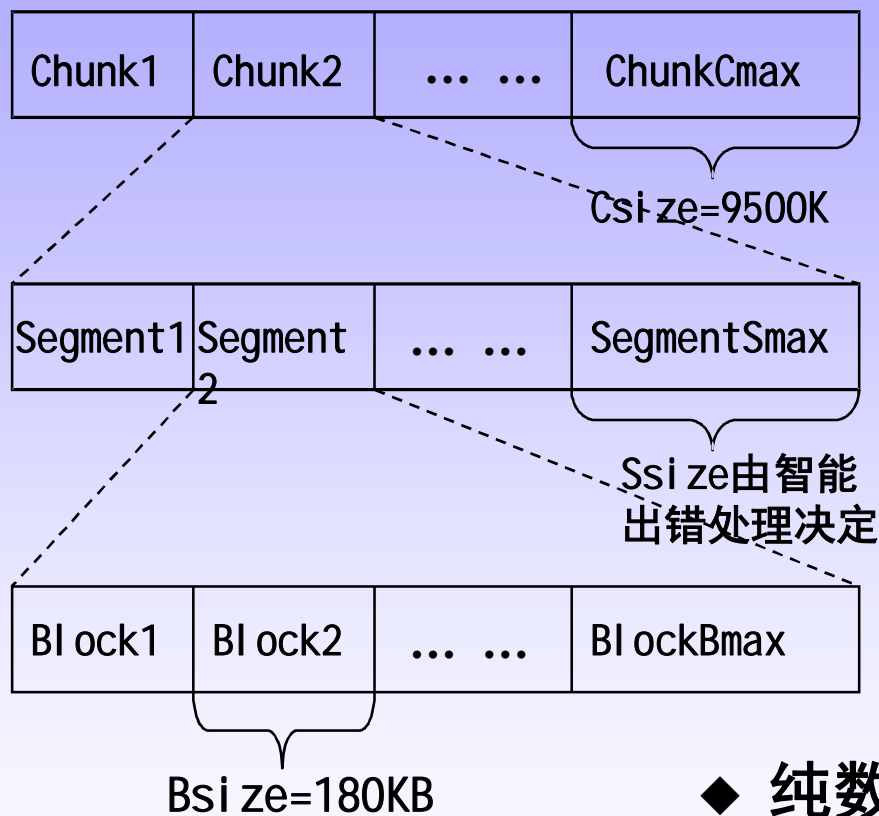
- 连接最适合S
 - 与过去“入口S”列表中的S建立连接，从中选择离自己延时最小的那个
 - 通过入口S获得普通S列表，从此表中选择最适合S连接，并断开原入口S
- 上载共享文件信息：客户A上载索引到A，A和B、C、D交换文件信息
- 查询：客户k向C/D发出查询
- 回答：返回索引=文件名+大小+位置

◆ 连接方式

- C \leftrightarrow S间TCP/4661；深层查询UDP/4665
- C \leftrightarrow C间TCP/4662
- 动态自适应：下载者每40s向上传者重发下载请求，否则关闭连接
- S \leftrightarrow S间周期性交换服务器、文件列表



eDonKey分块及性能测量



eDonkey文件分块细节

C \leftrightarrow C间TCP/4662上的网络参数测量

‘4662’端口所有连接	343.1743万
主机数	25万
所有流的传输总量	295 G Byte
下载连接的传输总量	208 G Byte (70.5%)
下载连接数	7.7111万 (2.24%)

- ◆ 纯数据下载比70.5%并不高
- ◆ 仅2.24% 的数据连接却承担了70.5%的通信量
- ◆ 总体不如BT

4.3.4 无结构网络总结

◆ 覆盖网络的拓扑特性

- 用户自发形成的、随机松散、任意形状的普通拓扑
- 但也符合内在某些规律
 - ☞ 小世界模型：5-6跳找到（人、信息）
 - ☞ 幂率模型：互联网中有连接数 L 的节点数占网络总节点数的份额正比于 L^{-a} ， a 是网络本身的常数因子

◆ 路由和定位方法

- 洪泛法
 - ☞ 预先不知道数据在何处？路由存在很大随机性
 - ☞ TTL 控制洪泛半径，大于半径的可能存查不到
- 扩展环：试探性洪泛，不断增加TTL
- 随机走：随机选择一个邻居行走，直到TTL耗尽
- 超节点路由

◆ 容错性与自适应

- 幂率特性对随机节点失效有高容错性
- 自适应：检测邻居在线否
- 超级节点列表定期更新

◆ 可扩展性

- 改造洪泛提高可扩展性

◆ 安全性与匿名性

- 无结构不易追踪

◆ 增强机制—复制

- 查询分布：均匀、Zipf
- 复制份数：均匀、依查询概率比例、方根复制

◆ 优势和缺陷

- 高容错性和良好自适应性，较高安全性和匿名性
- 路由效率低/可扩展差/准确定位差

无结构网络的缺点

◆ 无确定拓扑结构的支持

- 无法保证资源发现的效率。即使需要查找的目的节点存在发现也有可能失败。
- 由于采用TTL (Time-to-Live)、洪泛 (Flooding)、随机漫步或有选择转发算法，因此直径不可控，可扩展性较差。

◆ 面临两个重要问题

- 发现的准确性
- 可扩展性的

◆ 目前研究主要集中在：提高发现的准确率和性能

- 改进发现算法
- 复制策略

4.4 结构化P2P网络（第三代）

◆ P2P网络拓扑演进背景

- 1999，混合式
- 2000，无结构
- 2001，结构化

◆ 2001年后学术界开始关注

- IEEE成立P2P专业协会
- ACM成立SIGCOMM
- 国际会议/刊物发表论文

◆ 提出第三代模型

- Chord/CAN/Tapestry
- Pastry/CFS/PAST

◆ 成立专门研究机构

- MIT的Chord和CFS
- UC Berkeley的Tapestry和OceanStore
- 微软和Rice大学的Pastry和PAST
- Stanford的Peers研究组

◆ 商业领域大发展

- 基于异或度的Kademlia网络被BT/eDonkey/eMule所使用
- Azureus/eXeem等
- eMule、BitTorrent是典型结构化P2P网络
- eMule0.42开始使用Kademlia协议

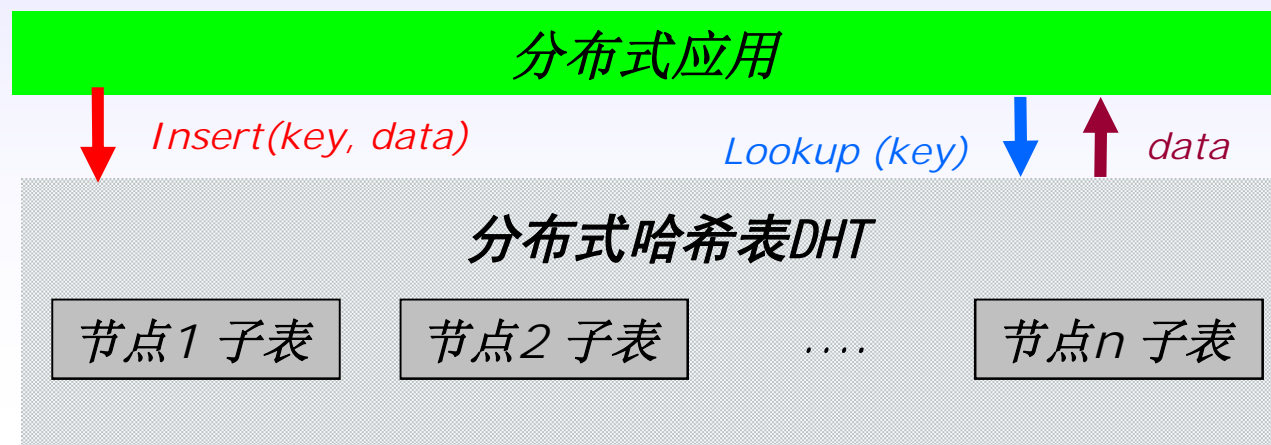
4.4.1 分布式哈希表结构

◆ Distributed Hash Table

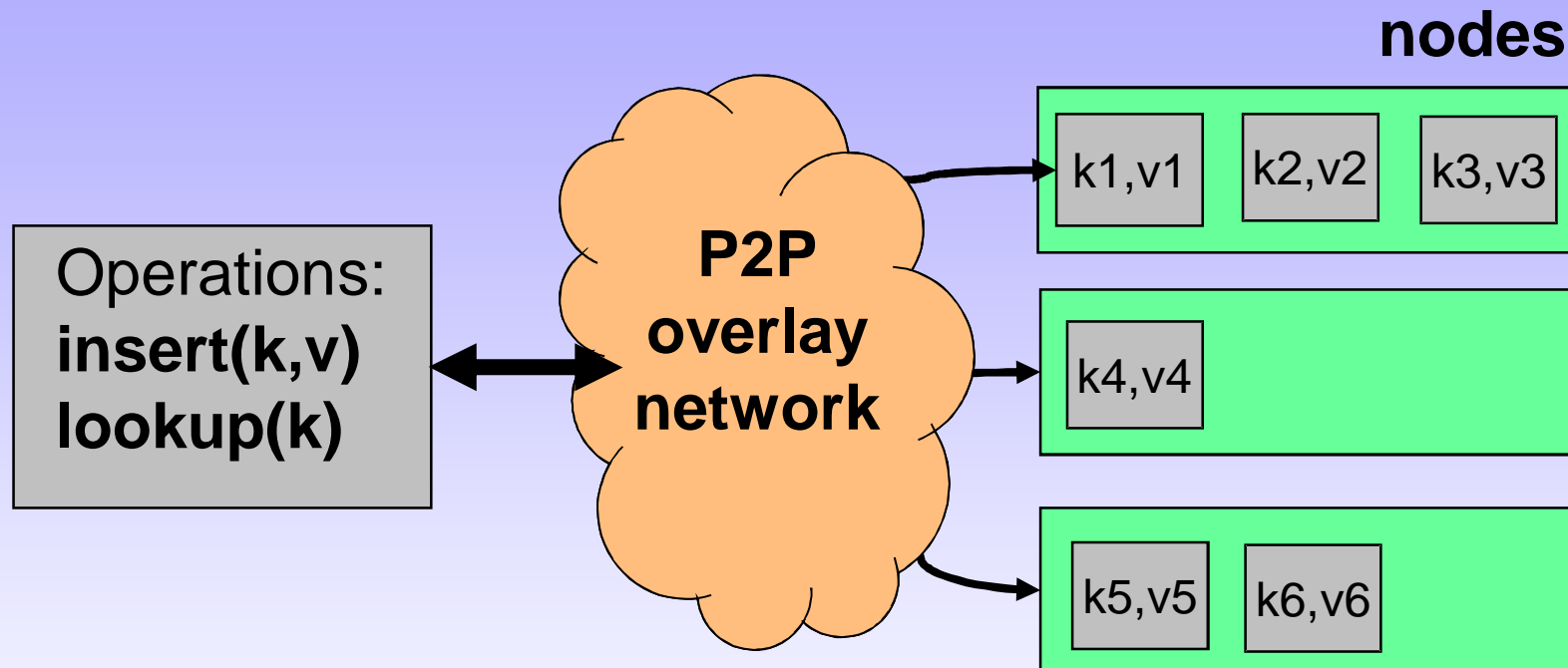
- 把涉及全系统的数据表，分布分段存储在各节点中，并通过 Hash 方法进行插入和查询
- 分布式数据结构可以是环/树/超立方体/跳表/蝶形网, CFS, OceanStore, PAST, Chord CAN ... 采用

◆ DHT的特点

- 能自适应结点的动态加入/退出
- 有着良好的可扩展性、鲁棒性
- 结点ID分配的均匀性和自组织能力。
- 确定性拓扑结构可精确发现



Distributed Hash Tables (DHT)



- p2p overlay maps keys to nodes
- completely decentralized and self-organizing
- robust, scalable

Distributed Hash Table 历史

◆DHT: 2000-2001年，理论研究者加入

◆动机

- 这些不成熟的P2P应用竟然如此流行，“我们可以做得更好”
- 保证系统中的文件能够被找到
- 保证搜索时间在可证实的范围
- 保证数百万节点的可伸缩性

◆成为研究热点

◆ 设计目标

- 非中心化
- 原子自组织

◆ DHT是什么？

- 一个由广域范围大量结点共同维护的巨大散列表
- 它被分割成不连续的块，每个结点被分配给一个属于自己的散列块，并成为这个散列块的管理者。
- DHT的结点是动态的，数量巨大
- 通过加密散列函数，一个对象的名字或关键词被映射为128位或160位的散列值。
- DHT结点被映射到一个空间

◆ DHT的起源

- 起源于SDDS (Scalable Distribute Data Structures) 研究，Gribble等实现了一个高度可扩展，容错的SDDS集群。



◆ 结构化重叠路由

- 加入：开始时，联系一个“bootstrap”节点，加入分布式数据结构，获得一个节点id
- 发布：向数据结构中最近的节点发布文件id的路由信息
- 搜索：向路由表中最近的节点查询文件id，数据结构保证查询会找到发布节点
- 获取：两个选项
 - ☞ 查询到的节点保存有文件，则从查询结束的节点获取
 - ☞ 查询到节点返回结果：节点x有文件，则从节点x获取

DHT技术

◆ 节点与资源的编址

- NodeID: 结点按照一定的方式分配一个唯一结点标识符
- ObjectID: 资源对象通过Hash运算产生一个唯一的资源标识符

◆ 定位

- 资源以文件形式存在
- 把文件名或关键字等文件属性信息抽象表示为Key
- 把文件或存储该文件节点的IP 地址抽象表示为Value
- 每个文件有一一对应的二元组<Key, Value>

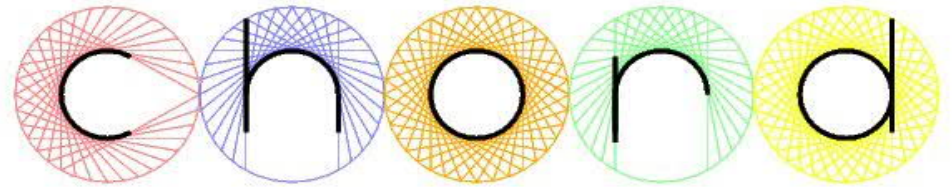
◆ 资源上传

- 资源Object ID将存储在与结点ID之相等或者相近的结点上

◆ 资源下载

- 通过<Key, Value>定位到存储该资源的结点，并直接下载

4.4.2 Chord/CFS



◆ 环形P2P网络

- Chord: 弦/带环弦。最简单、优美而精确的P2P网络
- 在N个节点的网络，每个节点保存 $O(\log N)$ 个其它节点的信息
- 在 $O(\log N)$ 跳内可找到存储数据对象的节点
- 节点离开或加入网络时，保持Chord自适应所需消息数 $O(\log^2 N) = O((\log N)^2)$
- 可提供数据对象的存储、查询、复制和缓冲
- 在其上构架有协同文件系统CFS (cooperative file system)

◆ MIT提出

Chord基础工作原理

- ◆ ID的分配：通过安全hash函数(如SHA-1) (Secure Hash Standard)
 - $\text{nodeID} = H(\text{node属性}) = H(\text{IP地址/端口号/公钥/随机数/或其组合})$
 - $\text{objectID} = H(\text{object属性}) = H(\text{数据名称/内容/大小/发布者/或其组合})$
 - SHA-1的长度值 $\geq 160 \text{ Bits} = m$ ，从而保证其唯一性和几乎不重复性，故nodeID/objectID均可在 $[0 \dots 2^m]$ 中选取
- ◆ 索引的分配
 - nodeID从小到大、顺时针排列于1个环上
 - 对象k的后继Successor(k)节点: 数据对象k(即objectID=k)也按环上顺时针方向，分配到节点k或第一个比k大 ($\text{mod } 2^m$) 的节点上 (K不在)
 - 形式化表示 $\text{Successor}(\text{object}_k) = \text{node}_k \text{ 或 } \text{node}_x \text{ mod } 2^m$ ，x是现存网上顺时针第一个大于k的节点
- ◆ 查询索引
 - 按上述后继关系，Chord显然可以正确工作
 - 但效率低下，找到紧邻后的节点显然要顺藤摸瓜 $O(N)$ 个节点

把内容key分配到其后继节点

$H(\text{天堂1.MP3}) = \text{ObjectID} = 1 = \text{key1}$

$\text{Successor}(\text{key1}) = 1 = N1$



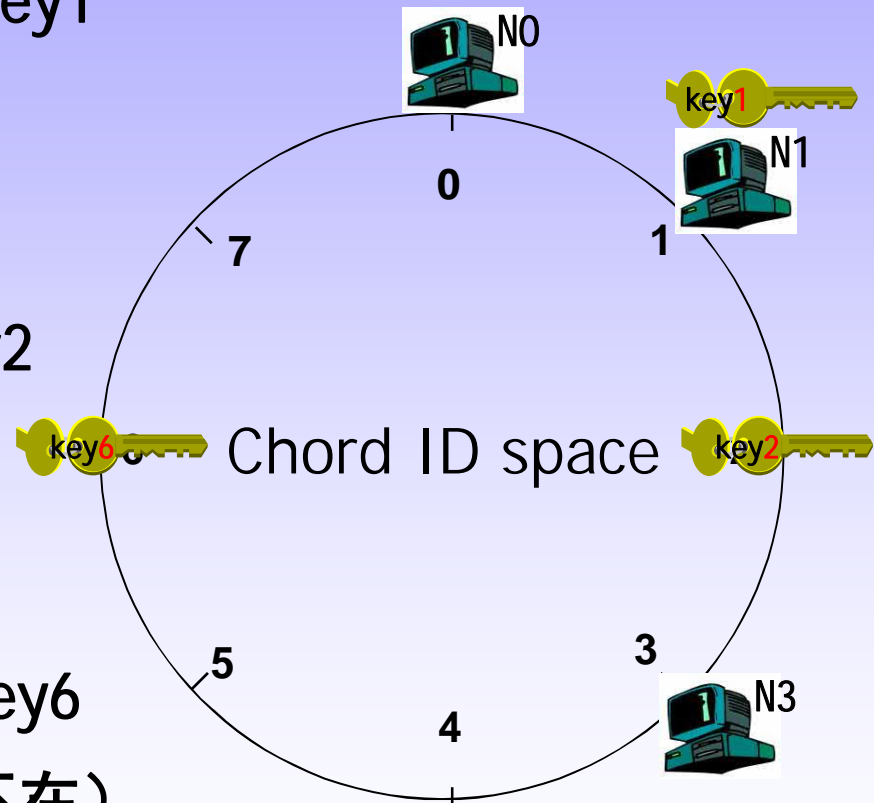
$H(\text{天堂2.MP3}) = \text{ObjectID} = 2 = \text{key2}$

$\text{Successor}(\text{key2}) = 3 = N3$ (2不在)



$H(\text{天堂6.MP3}) = \text{ObjectID} = 6 = \text{key6}$

$\text{Successor}(\text{key6}) = 0 = N0$ (6, 7不在)

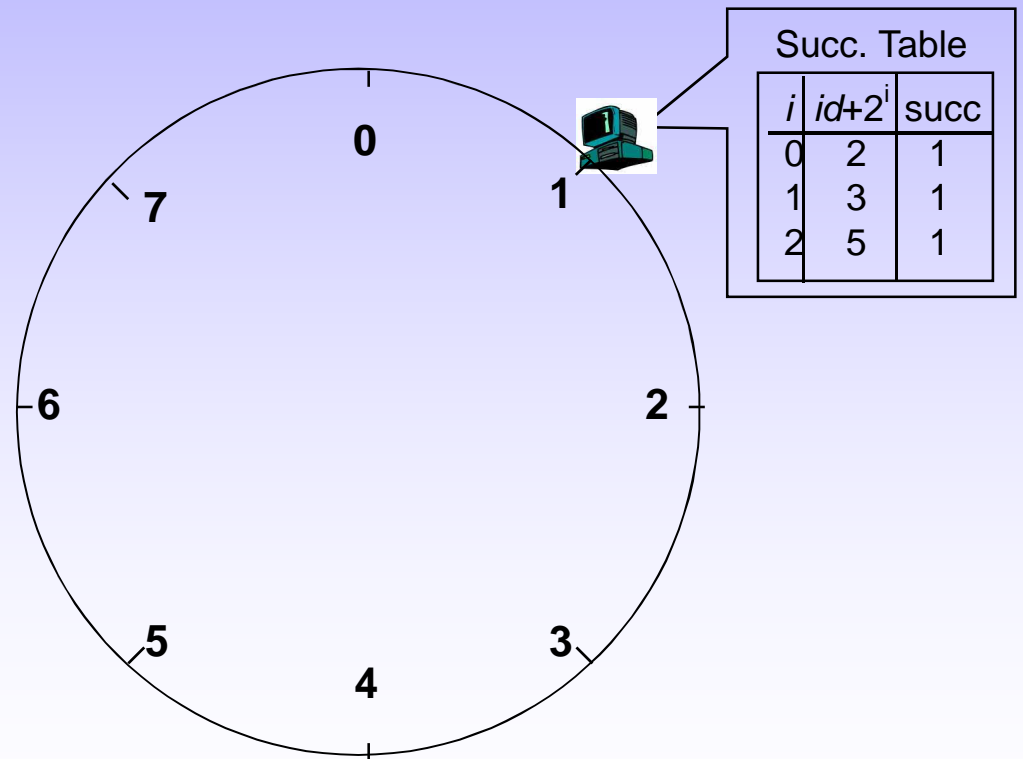


节点路由表的分配

- ◆ 用路由表：使每步更快接近目的节点
- ◆ 指向表：finger table
 - 每节点存储m项大小的路由表，以减少路由跳数
 - 每个节点表存m个路由项；节点n的路由表中，第i项指向节点 $s = \text{successor}(n+2^i)$, $0 \leq i \leq m-1$
 - 故s是在顺时针方向到节点n的距离至少为 2^{i-1} 的第一个节点：记作 $n.\text{finger}[i].\text{node}$ ；也就是 $n.\text{successor}$
 - 一个路由表项还包括相关节点的nodeID和IP地址、端口号。
- ◆ 特点
 - 每节点只保存很少其它节点信息，且对离它越远的节点所知甚少
 - 节点不能从自己的路由表中直接看出对象k的后继节点
- ◆ 前驱节点：Predecessor(k)
 - 在节点k之前，不等于k且离k最近的节点Predecessor(k)
 - 节点n在自己的路由表中寻找在k之前且离k最近的节点j，让j去找离k进一步最近的节点，如此递归下去，j将离k越来越近，最终找到k的前驱节点
 - n的前驱节点为 $n.\text{predecessor}$

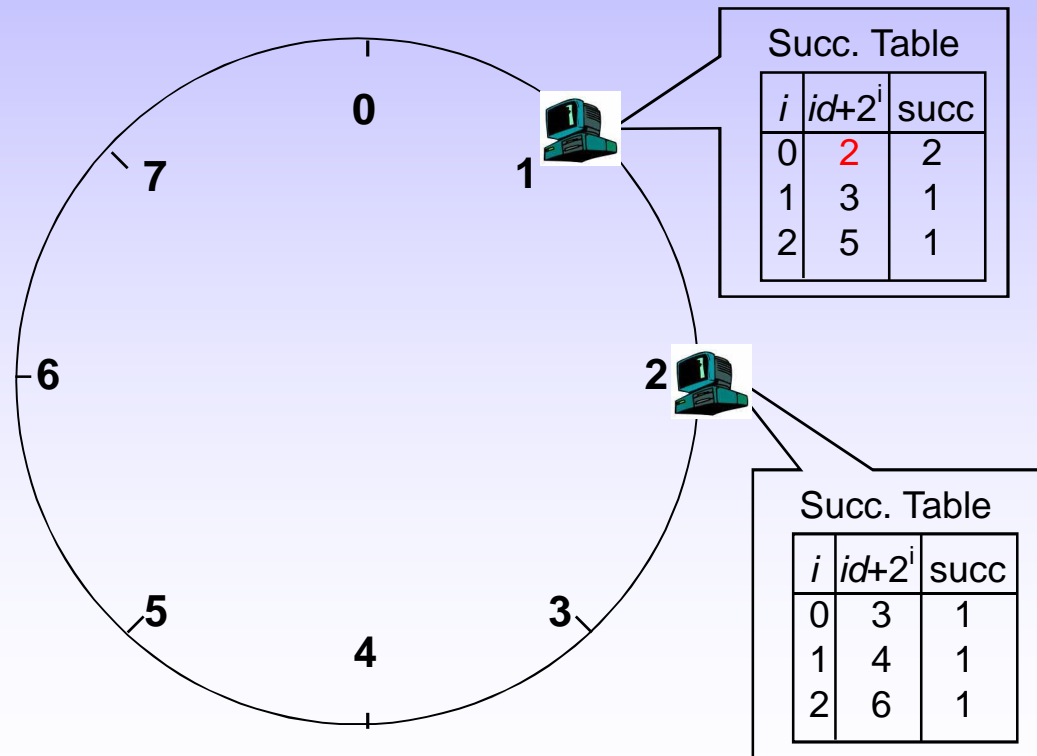
Chord的自组织与路由表

- ◆ 假设id空间是 $[0..2^m-1]$
 - $m=3$, $[0..7]$; $0 \leq i \leq 2$
- ◆ 节点1加入
 - 即id=1之节点加入



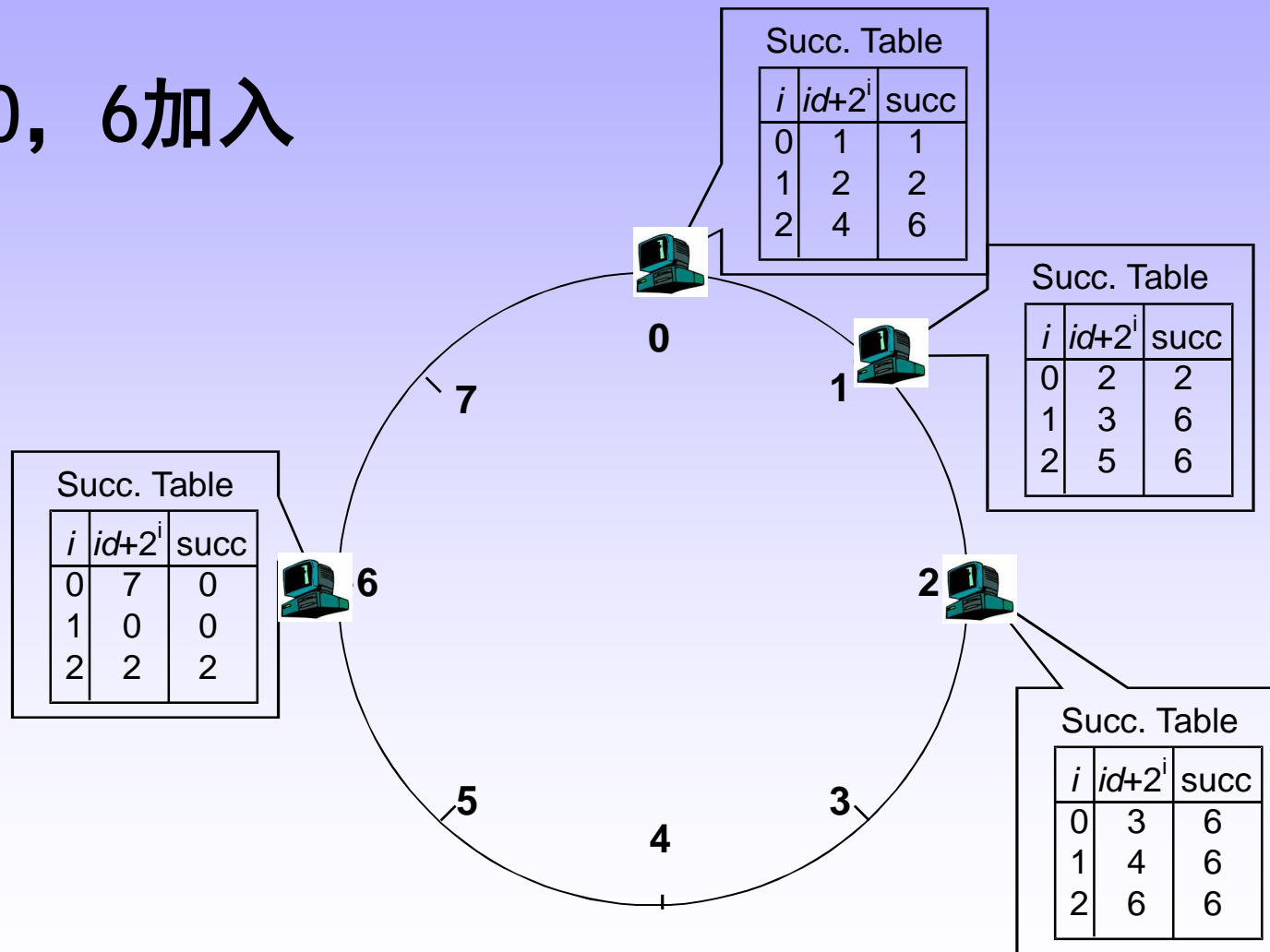
Chord自组织与路由表

◆节点2加入



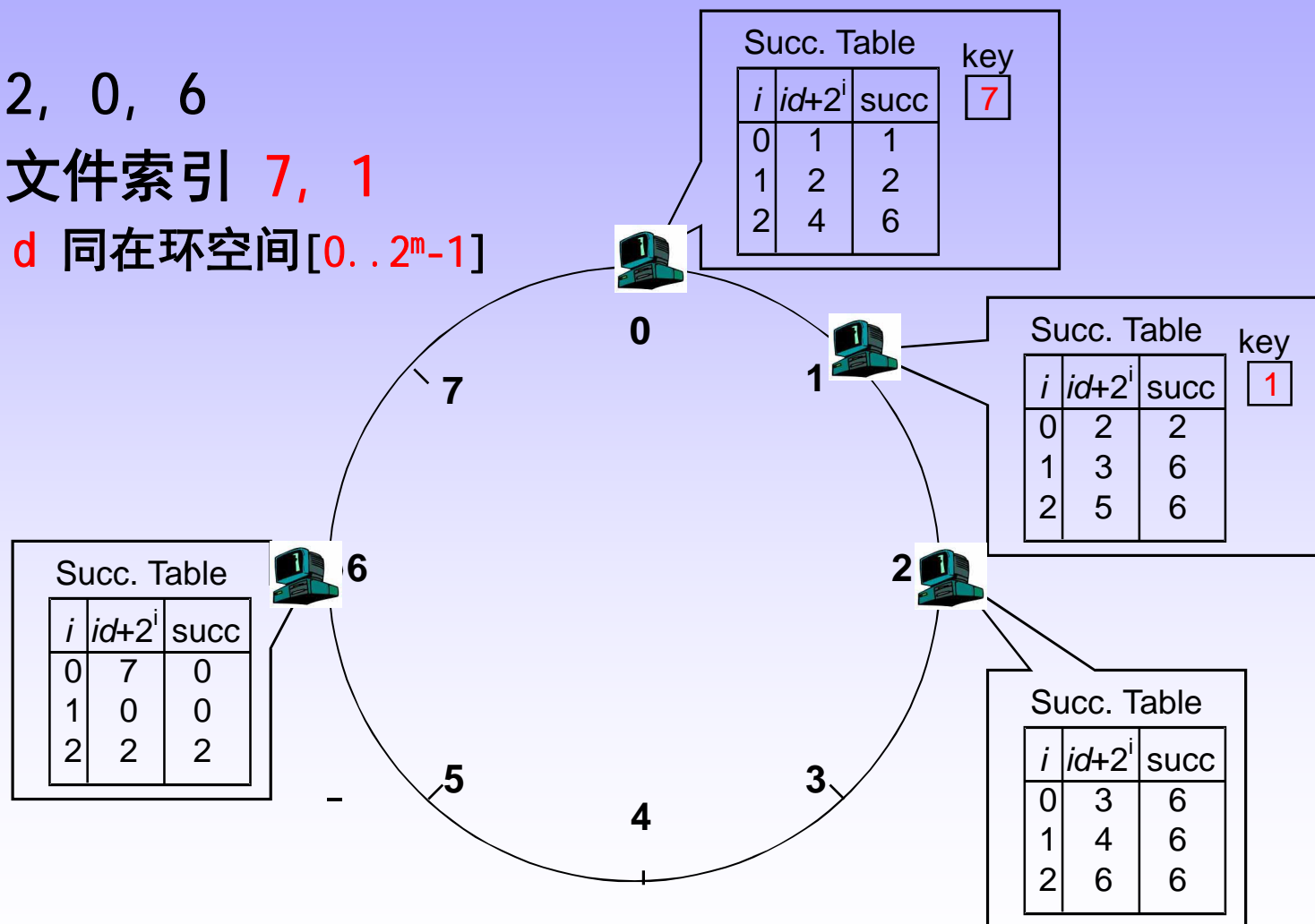
Chord路由表的变化

◆节点0, 6加入



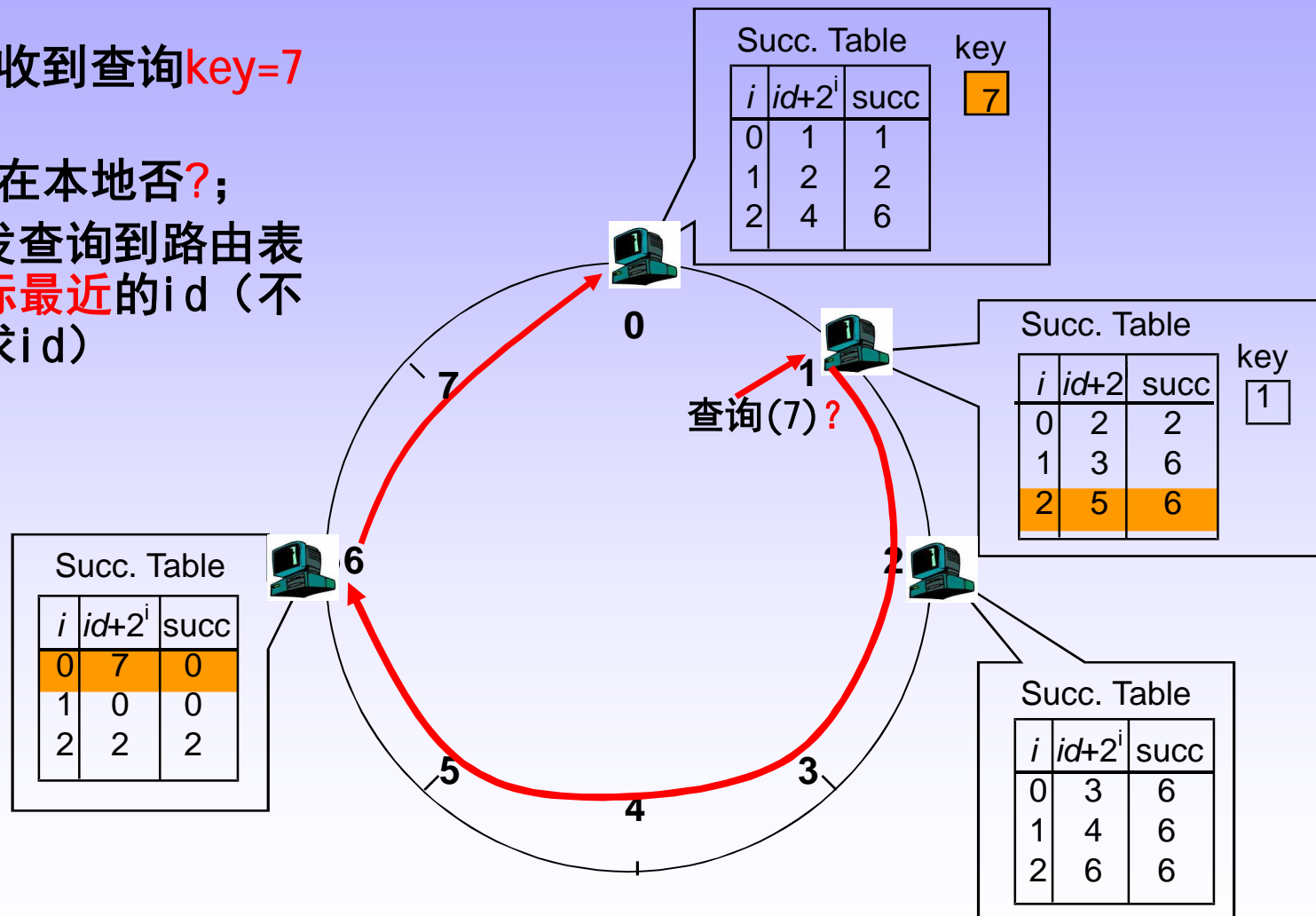
Chord路由表的变化

- ◆ 节点: 1, 2, 0, 6
- ◆ 文件上载: 文件索引 **7, 1**
 - 文件 **keyid** 同在环空间 $[0..2^m-1]$



Chord查询

- ◆ 当节点1收到查询key=7请求后:
- ◆ key=7存在本地否;
- ◆ 否, 转发查询到路由表中与目标最近的id (不超过请求id)



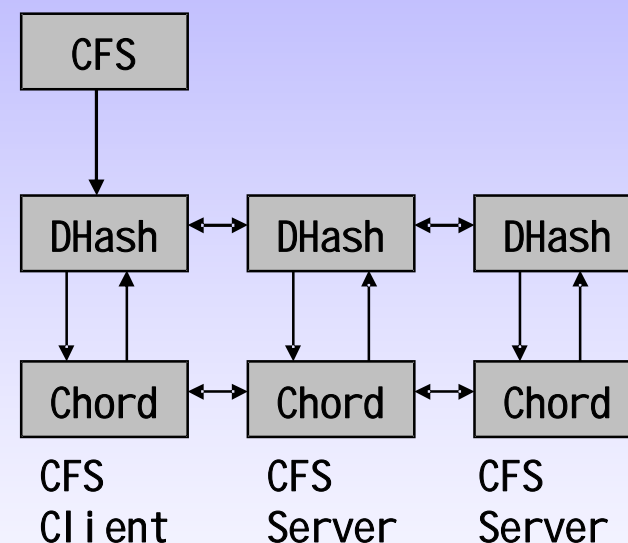
协同文件系统CFS简介

◆ CFS是以Chord作为基础的P2P只读文件存储系统

- 依靠Chord作为其分布式HASH表，提供高效、容错和负载均衡的文件存储和获取
- 系统由文件系统FS、分布式散列表DHash和底层定位散列表Chord三层构成
- 每个节点既是服务器又是客户机

◆ 功能说明

- FS: 高层，从DHash层获取数据块，将这些块转换成文件，给更高层文件系统接口
- DHash: 中间层，分布和缓存数据块以负载均衡，复制数据块以容错，通过服务器选择来减少延迟，用Chord定位数据块
- Chord: 底层，维护路由表，定位数据块所在的服务器



CFS的系统结构

Chord: 总结

◆ 简单、精确，但要求

- 每个节点的后继节点始终准确
- 每个对象k的后继节点始终承担k的索引

◆ 每个节点维护DHT的一小部分表，大小= $\text{Log}N = m$

- 其中第i项保存距自身的间隔至少为 2^{i-1} 的第一个节点的位置信息

◆ 查询与维护代价

- 查询代价：m项的平均间隔 $\text{delta} = [2^0 + 2^1 + \dots + 2^{m-1}] / m$; 设经J跳命中，应总跳数应 $\leq 2^m$; 故 $J * \text{delta} \leq 2^m$; 故有 $J \leq m = O(\log M)$; 一次查询的路由平均跳数为 $O(\log M)$
- 维护代价：节点的进/出，自适应到最新状态需 $O(\log^2 M)$

◆ 缺点：

- 没有实际使用（只有一个文件共享应用）
- 不支持非精确查找

Discussion

- ◆ Query can be implemented
 - Iteratively
 - Recursively
- ◆ Self-organization
 - Join and leave
 - Gracefully and abruptly
 - Distributed or locally
- ◆ Performance: routing in the overlay network can be more expensive than in the underlying network
 - Because usually there is **no** correlation between node ids and their locality; a query can repeatedly jump from Europe to North America, though both the initiator and the node that store the item are in Europe!
 - Solutions: Tapestry takes care of this implicitly; CAN and Chord maintain multiple copies for each entry in their routing tables and choose the closest in terms of network distance

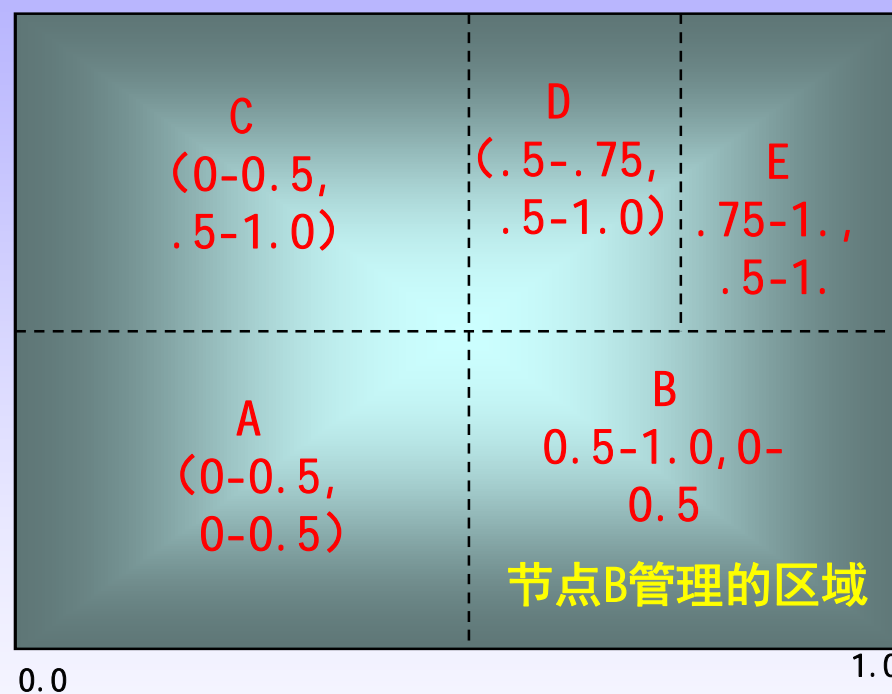
4.4.3 内容可寻址网络CAN

◆ Content Address Network背景

- Ratnasamy等2001年ACM会发表, 与Chord相提并论, 美加州伯克利提出

◆ 基本思想

- 采用多维空间拓扑结构, 严格是多维环面结构 (Torus)
- 空间动态分配给网络各节点, 每个节点有一个属于自己的方块并负责方块中的所有“点”
- 每个数据对象通常被唯一映射到多维空间中的一个“点”, 由负责该点的节点来存储该数据对象的索引
- 每点维护一个路由表, 记录它在多维空间上的邻居信息
- 定位也是逐步完成, 每跳选中当前节点路由表中离目的节点最“近”的邻居作为下一跳



5个节点的二维CAN

CAN网络的构建

◆ 新节点加入

➤ Join1 自举:

- ☞ 新节点n找到一个**已在CAN节点**，如通过**CAN的DNS**获得**举荐者的入口IP地址**

➤ Join2 寻找区域:

- ☞ n**随机选择**CAN空间一点P，通过**举荐者向P发加入请求**，并**达到P区节点n'**
- ☞ n'**把区一分为二**，**先横后纵**，**分给n**，相应**数据对象也转给n**

➤ Join3 加入路由表:

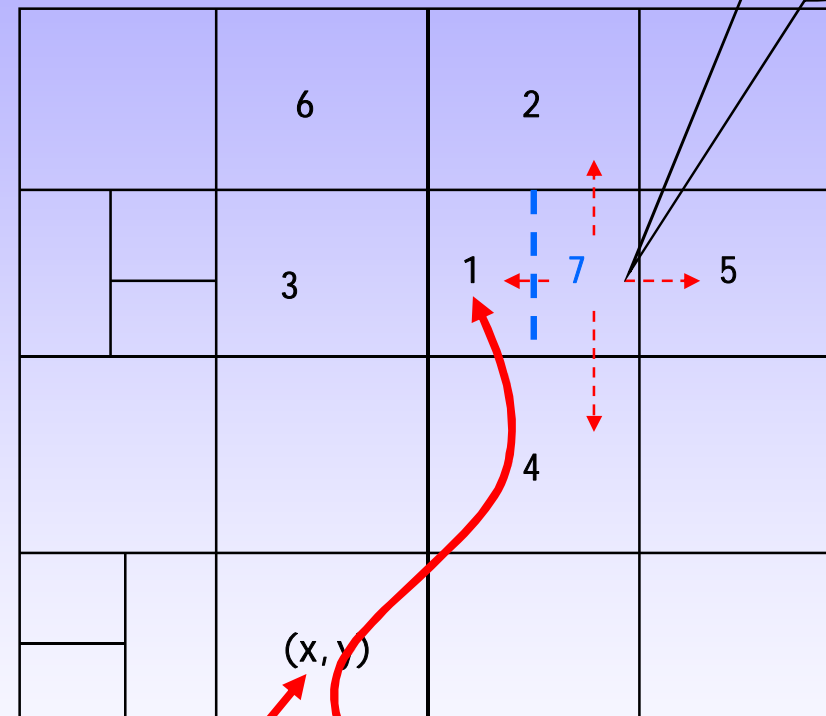
- ☞ n从n'获得**邻居信息**
- ☞ **通知邻居改变路由表**以反应n加入的变化

◆ 自适应

- 当节点**插入、离开或周期更新**
- 自动向邻居**发已区路由表信息**
- 0(2d)邻居，开销是0(2d)

1的**新路由**={2, 4, 3, 7}

7的**新路由**={2, 4, 1, 5}



7号节点自举
通过 (x, y)

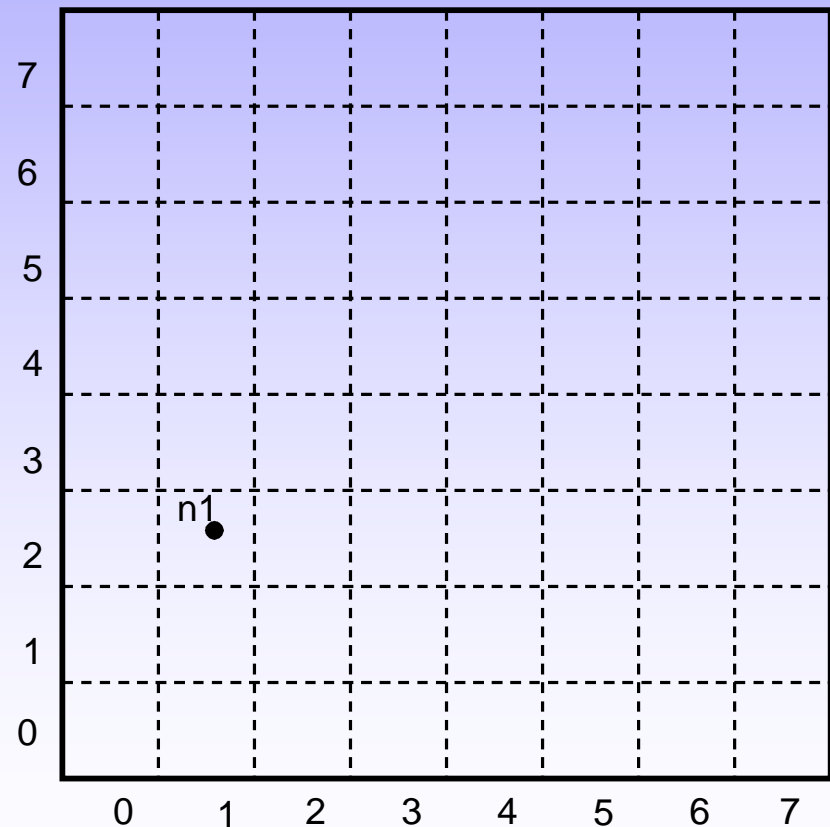
7选择1区的点

1的**原路由**={2, 4, 3, 5}

7的**原路由**={ ϕ , ϕ , ϕ , ϕ }

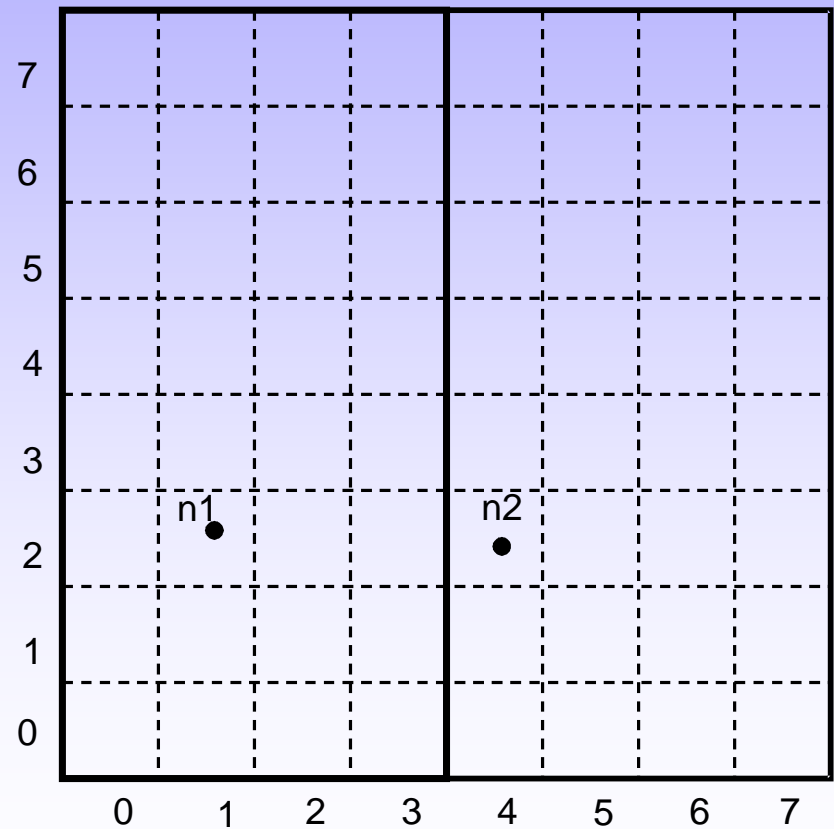
CAN Example

- ◆ Space divided between nodes
- ◆ All nodes cover the entire space
- ◆ Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- ◆ Example:
 - Assume space size (8 x 8)
 - Node n1: (1, 2) first node that joins → cover the entire space



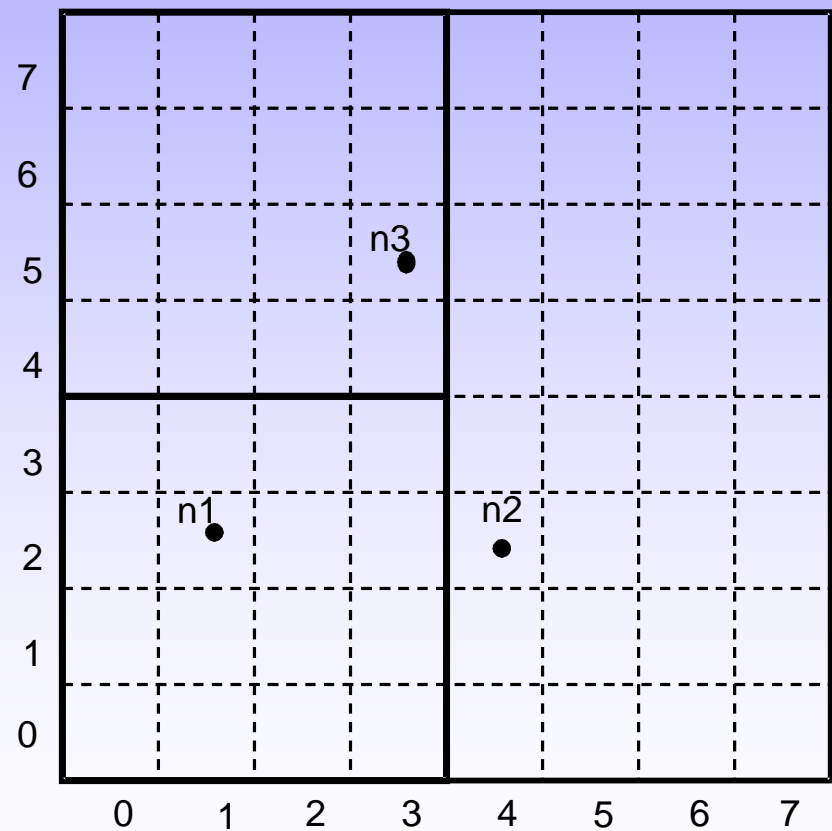
CAN Example: join

- ◆ Node n_2 : (4, 2) joins
→ space is divided
between n_1 and n_2



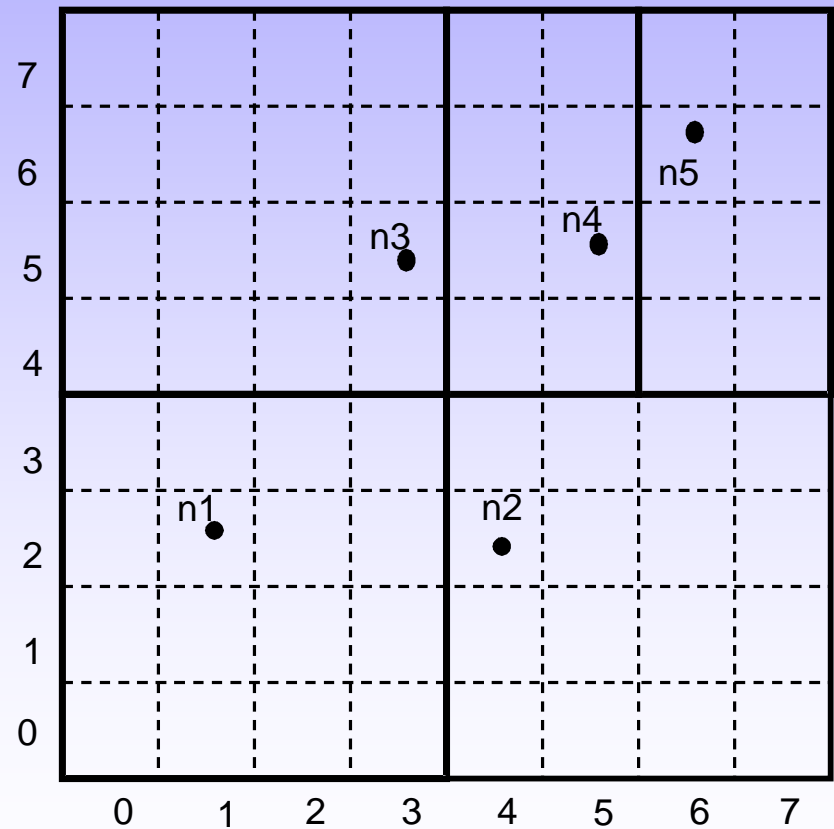
CAN Example: join

- ◆ Node n_3 : (3, 5)
joins \rightarrow space is
divided between
 n_1 and n_3



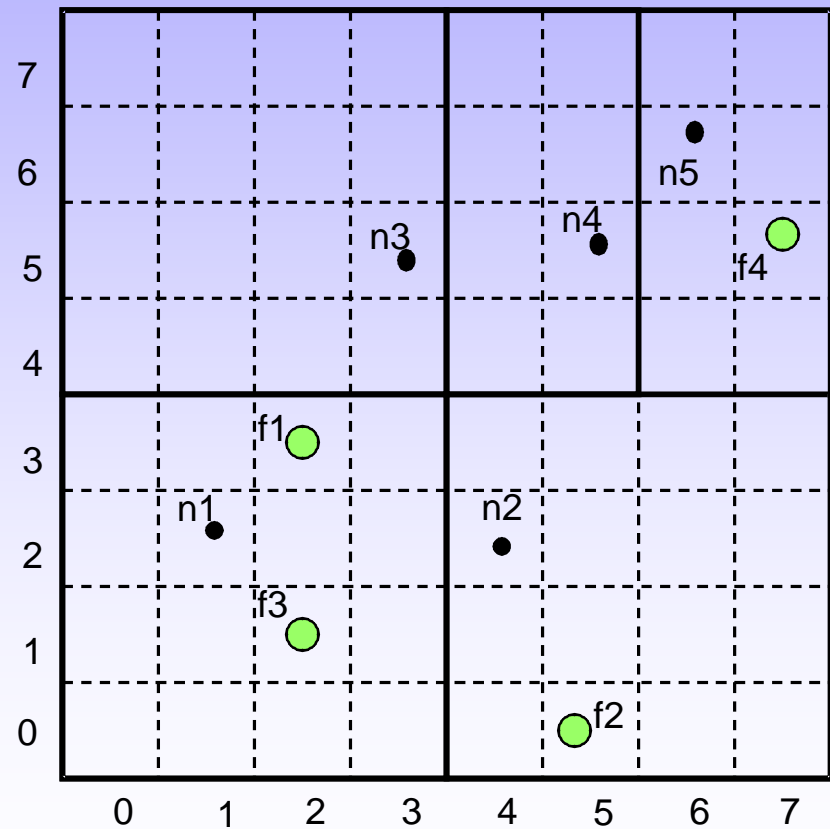
CAN Example: join

- ◆ Nodes $n_4: (5, 5)$ and $n_5: (6, 6)$ join



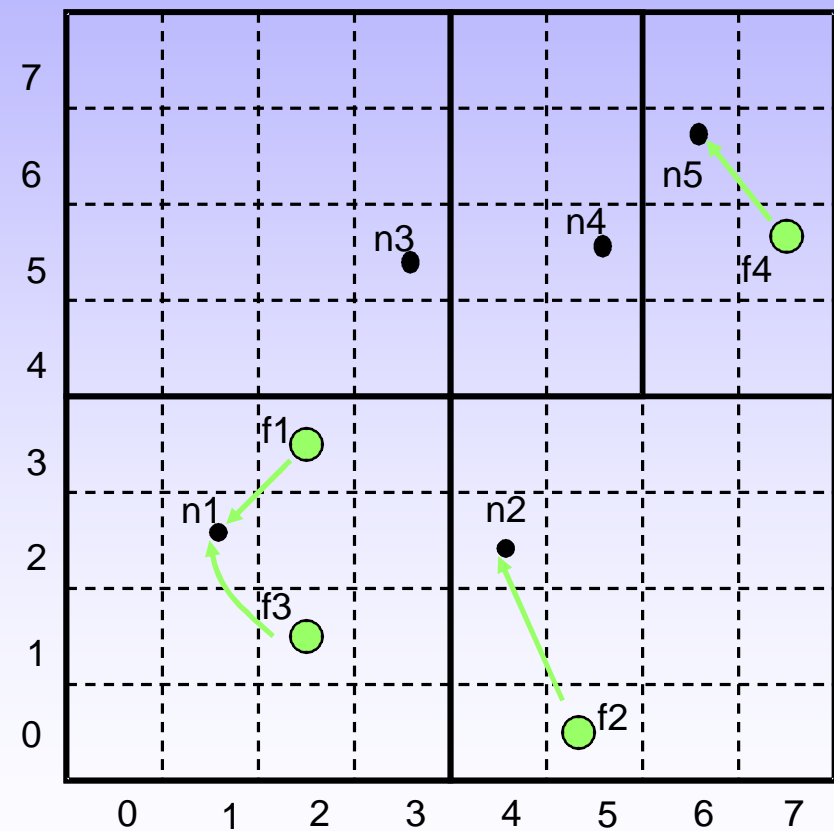
CAN Example: join

- ◆ Nodes: $n1: (1, 2);$
 $n2: (4, 2);$ $n3: (3, 5);$
 $n4: (5, 5);$ $n5: (6, 6)$
- ◆ Items: $f1: (2, 3);$
 $f2: (5, 0);$ $f3: (2, 1);$
 $f4: (7, 5);$



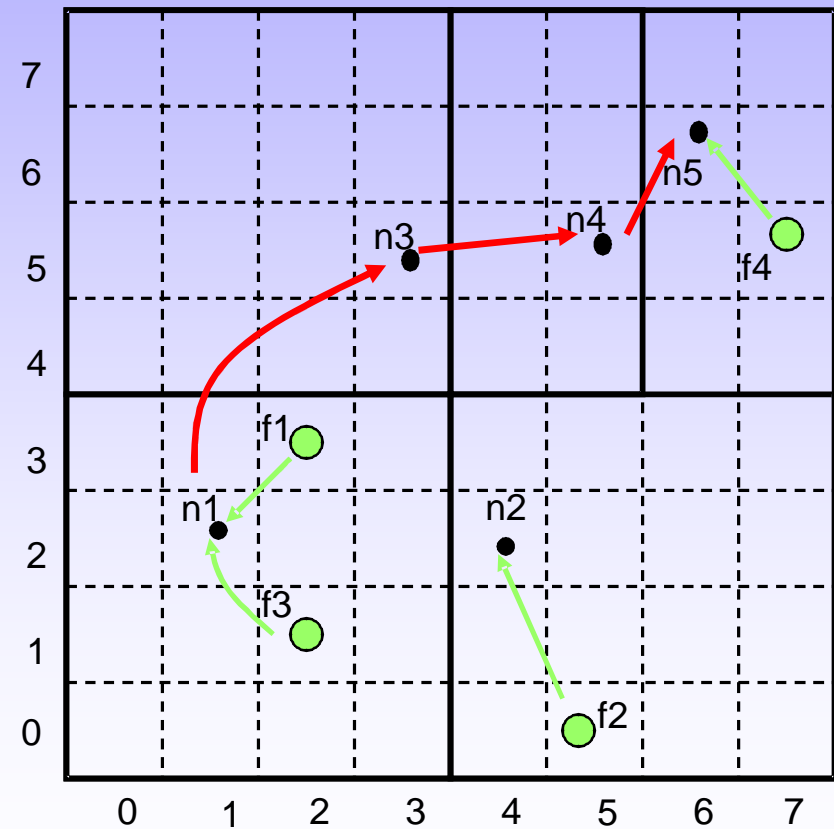
CAN Example: band 1 e

- ◆ Each item is stored by the node who owns its mapping in the space



CAN Example : Query

- ◆ Each node knows its neighbors in the d -space
- ◆ Forward query to the neighbor that is closest to the query id
- ◆ Example: assume $n1$ queries $f4$



节点的离开与破碎

◆ CAN节点的离开

➤ 正常离开

- ☞ 应把其负责区转交给一邻居，邻居可合并区域；
- ☞ 如不能，则转交给区域最小的邻居，并不合并

➤ 动态突发、不辞而别？

- ☞ CAN周期检测在线节点，一旦发现，邻居开始接管
- ☞ 接管后向邻居发 TakeOver；同时接管的多个邻居比较区域大小，最大者取消接管

◆ 支离破碎问题

- 随着频繁加入离开，区域划分会破碎
- 由一个节点负责的区越来越多，超出节点能力
- 背景区域重分配变规整，区域负责均匀化

CAN增强与总结

◆ CAN增强机制

- 多维d: 维护路由表 2^d 项, 定位效率 (路径长) = $O[d \times (N)^{1/d}]$; d 增加时, 路由表不会增加多少, 但路径减小很快, 容错也好
- 多空间: CAN空间只是实际网络的一个逻辑映射, 可用多空间来实现, 如多份拷贝, 提高可用性和效率, 可选择最近节点
- 多散列: 多Hash可把同一数据对象分配到多个区域中, 多Hash通常在同一数据对象上加入不同的Salt (盐值), 映射出不同效果

◆ 区域超载与复制缓存

- 区域后维护一个区域超载表, 通过减少跳数、延时和复制解决
- 把数据显式复制到邻居区域中

◆ 总结

- 简单直观, 每个节点维护 2^d 邻居信息, 定位跳数 $O[d \times (N)^{1/d}]$
- 高容错

- ◆ Virtual Cartesian coordinate space: d-dimensional torus
- ◆ Associate to each node and item a unique id in an d-dimensional space
- ◆ Entire space is partitioned amongst all the nodes
 - every node “owns” a zone in the overall space
- ◆ Properties
 - Routing table size $O(d)$
 - Guarantees that a file is found in at most $O(d \times \sqrt[d]{N})$ steps
 - where N is the total number of nodes

4.4.4 Tapestry

◆ 背景

- 2000.3, UC Berkeley's Ben Y. Zhao等开始开发, 2003.6发布 Tapestry2.0, JAVA编写/Linux; 应用广泛, 著名例子UC Berkeley's OceanStore广域存储系统
- 基于Plaxton Mesh覆盖网络结构的非严格“超立方体结构”

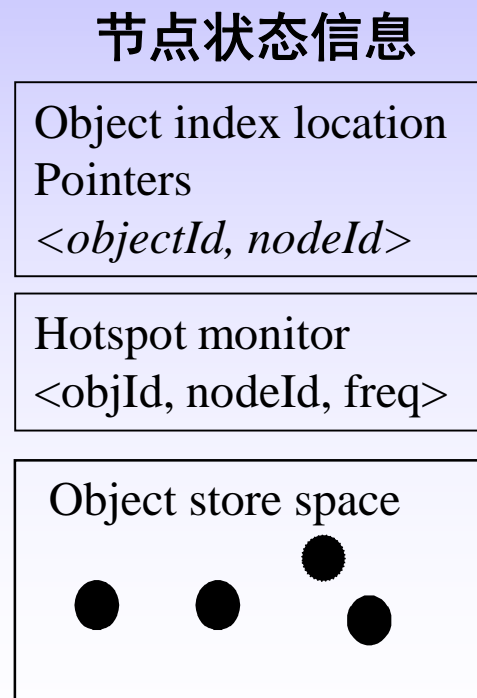
◆ 定位与路由: 任何网络须考虑的三大问题

- locating(名→址), routing, topology
- 名: 每个节点/数据对象有全局唯一nodeID, 和唯一objectID
- 绑定: 每数据对象分配一负责它的节点→成为该对象的“根” root
 - ☞ If 恰好某节点的nodeID==objectID; then root(objectID)= nodeID ;
 - ☞ Else root(objectID)= 最接近objectID的nodeID
- 路由: 后(前)缀匹配: XXX8→XX98→X598→4598(4598的匹配)
 - ☞ 每节点维护一层次化路由表(邻居表), 每层代表与自身nodeID匹配一定位数后缀的节点
 - ☞ 基于后缀的路由

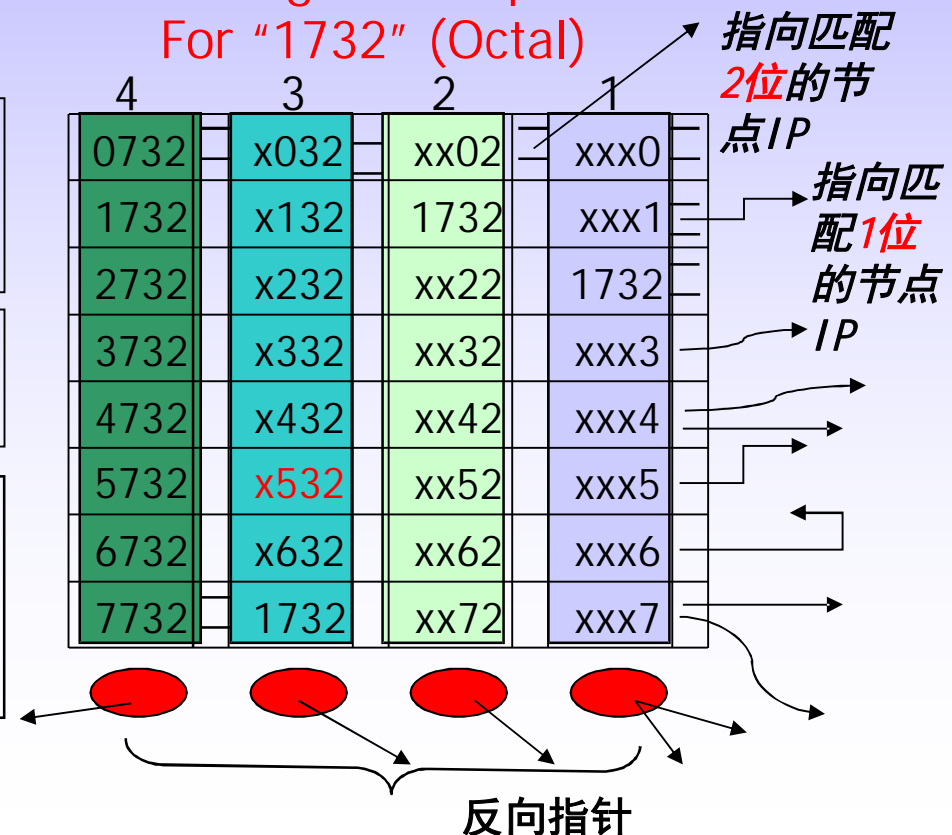
单个Tapestry节点的路由表

◆ 8进制4位路由表（1732节点的）

- 每列称为一层，从右往左第1/2/3/4，表示分别匹配第0/1/2/3位
- 第i层第j项（0—7共8项）表示当前nodeId后缀匹配为i-1位，且以j开头的节点，如ij (3, 5)=x532表示与当前节点1732匹配后2位32，并以5开头的节点



Neighbor Map
For "1732" (Octal)



Tapestry 路由过程: 0325⇒4598

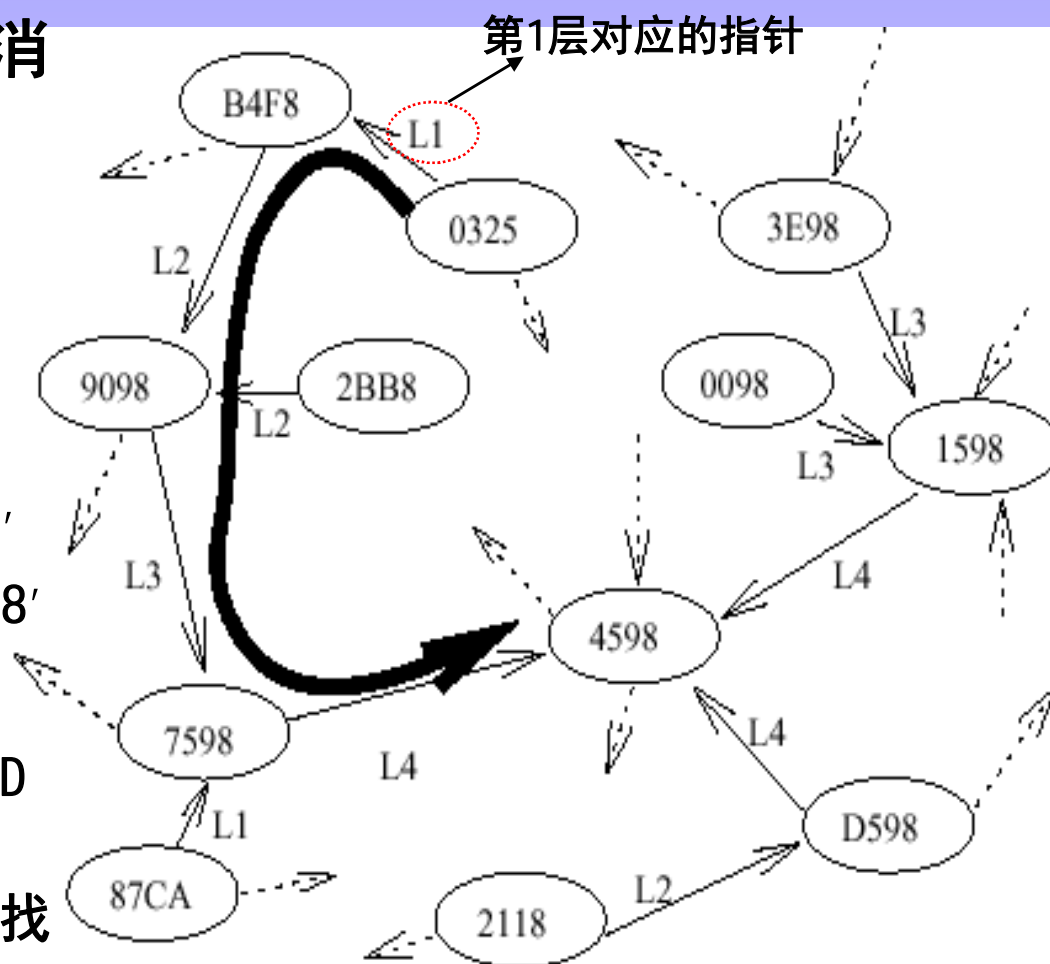
- ◆ 节点0325要发送一条消息给目的节点
nodeID=4598

- ◆ 路由过程

- 第1跳匹配第0位' 8'
- 第2跳匹配第1位' 98'
- 第3跳匹配第2位' 598'
- 第4跳匹配第3位' 4598'

- ◆ 路由算法

- 若当前节点ID与目标ID
后缀匹配刚好n位,
- 则在路由表第n+1层中找
匹配更多位的项;
- 该项指针指下一跳



Suffix-based routing

每个节点指派一个根节点

- ◆ 根节点: 对object O 有最长后缀的唯一节点

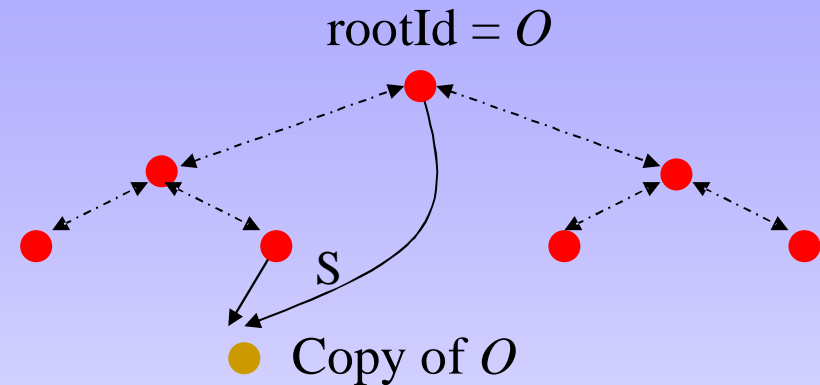
➤ 根知道 O 在何处

- ◆ 发布: 节点 S 发送
 $\text{msg} \langle \text{objectID}(O), \text{nodeID}(S) \rangle$
到对象 O 的根

- ◆ 如果对象 O 有多分拷贝, 就存储在最近的位置

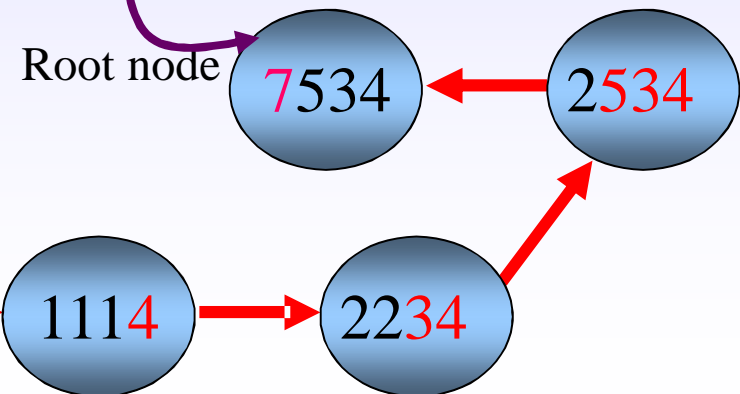
- ◆ 定位查询: 针对 O 的消息朝着 O 's 根路由

- ◆ 怎样找到对象的根节点?



.....

6534	empty
7534	



Data object

Tapestry动态节点算法

◆ 反向指针 (Back Pointer)

- 路由表中有反向指针项，指向那些把自己作为路由表项的点
- 利用BP周期性发送心跳 (Heartbeat) 消息给反向节点，宣告自己的存在，若一段时间内收不到则认为该路由表项失效

◆ 路由表中每项保存一个主项、和两个次项

- 当主项节点失效时，就使用次项节点
- 两次检测到失效后才替换路由表项，以防止“闪断”现象

◆ 节点的加入

- 初始化路由表：新节点N通过现存节点G发送以N自己为目址的消息
- 设第j步达到节点Hj，初始时H0=G；依后缀匹配算法，N和Hj共享长度j的后缀，故N从Hj那里获得路由表的第j+1层是适合的
- 路由表优化：对每项，若次要节点对N来说比主项节点更近，则次项变主项；找新主项节点路由表相关项，比较N到每项的距离...收敛



◆ 更新其它路由节点表

- N通过G经 $\log N$ 跳内最终达到N的根节点R
- R收到消息后，计算出与N匹配的后缀位数p，
- 通过反向指针告诉那些与R也匹配p位后缀的反向节点：
 - ☞ 如果N对自己是适合的，则在自己的路由表中添加N或用N替代原来的项
 - ☞ 将对象索引中应该由N负责的那部分索引移交给N

◆ 正常离开

- 告诉自己的反向节点在路由表中删除节点N
- 节点N把自己负责的对象索引移交给它们的新根节点

◆ 非正常离开

- 则通过周期性检测来发现失效和删除

Tapestry的体系结构

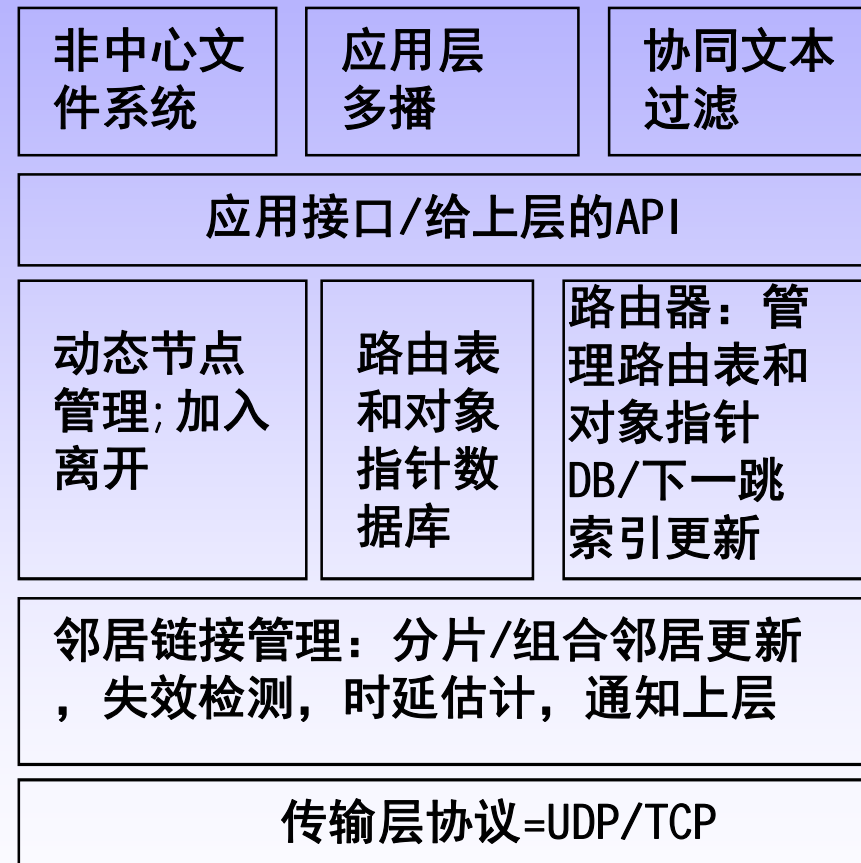
◆ 3个重要的接口函数

- Deliver(ID, AID, Msg): 接收到要发往ID的消息Msg, 当前节点是路由的终点
- Forward(ID, AID, Msg): 将以ID为目的址的消息Msg往前传, 它通过Route(...)完成
- Route(ID, AID, Msg, NextHopNode): 将以ID为目的址的消息Msg路由到下一跳节点

◆ 说明

- Id指nodeID或objectID
- AID代表某种应用类型
- 应用并不限于右图3类

单Tapestry节点组件结构



Tapestry 总结

- ◆ 是一个分布存取、容错的超立方体结构P2P模型
- ◆ 每个节点、数据对象和应用都有全局唯一ID，每个数据对象有一个负责它的节点（根），根是nodeID与objectID最匹配的节点
- ◆ 在N个节点的网络中，任何一次定位都能在 $\log_B(n)$ 跳内完成
 $n \leq N$ 是实际物理网的节点数。B为ID编码的进制，如B=8
- ◆ 共有 $\log_B(n)$ 层，每层有B项，故总表有 $B \log_B(N)$
- ◆ 反向指针可用于节点加入、离开、失效检测和修复
- ◆ 5层体系结构，支持很多应用，典型有
 - 文件分布式系统：OceanStore
 - 应用层多播：Bayeux
 - 协同文本过滤：SpamWatch

4.4.5 Pastry/PAST

◆ 背景

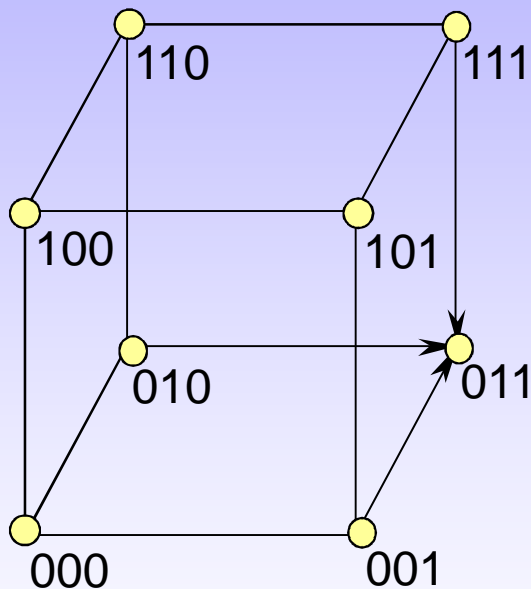
- 2000年, Microsoft Research (微软研究院), Rice University's Antony Rowstron and Peter Druschel 设计开发, 现有两个版本
 - ☞ Rice University's开发的FreePastry
 - ☞ Microsoft Research开发的SimPastry/VisPastry
- 著名应用是PAST归档存储系统

◆ 概况

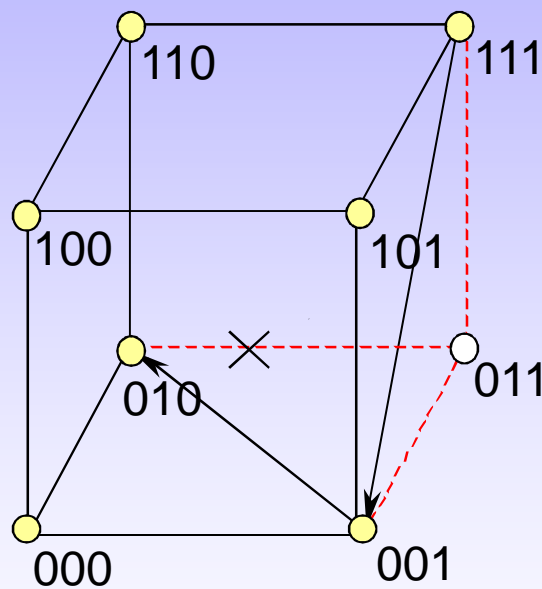
- 容错、高效、可扩展的混合式结构P2P网络
- 结合环形结构和超立方体 (Plaxton Mesh) 结构的优点, 提供高效查询路由、确定性的对象定位和独立于应用的负载均衡
- 提供复制、缓存和错误复原等增强机制

Pastry: hypercube connection

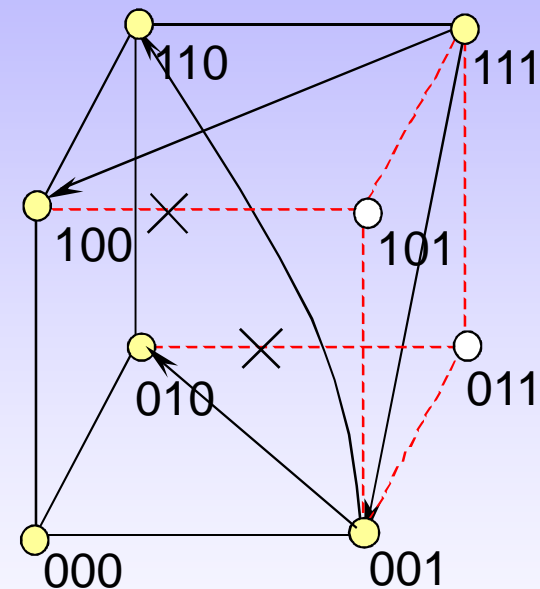
- ◆所有活动节点维持3个邻居节点，冗余备份；
- ◆当节点失效或负载很高时，可负载迁移或阻隔；



(a)



(b)



(c)

(a)传统超立方体 (b)节点011失效后的可重构 (c)节点101再失效后的可重构

Pastry的路由

◆ Hash(IP) 128位/均匀/唯一ID

- 采用前缀匹配，每节点维护一个路由表、叶集、邻居集
- 叶集→后继节点
- 邻居集→物理层邻近节点
- nodeID独立于节点属性，以标识网络覆盖
- objectID, 对象索引由与objectID最近的nodeID节点负责

◆ 路由表R

- 4进制8层路由表
- 从上至下第0, 1..7行，分别代表与当前nodeID匹配第0, 1..7位前缀的节点
- 每行4列从左至右以0, 1, 2, 3开头
- 与当前nodeID在该位恰等的列标为阴影（10233102）
- nodeID以X-Y-Z形式便于观看
 - ☞ X: 匹配的前缀，第0行无X
 - ☞ Y: 第一个不匹配位
 - ☞ Z: ID的后几位

Nodeid 10233102			
Leaf set			
	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

◆ 每节点维护一叶集L

- 包含 $|L|$ 个与当前nodeID最邻近的“叶节点”
- 其中 $|L|/2$ 个比当前节点ID小的叶节点, $|L|/2$ 大的节点
- 作用在于保证路由正确性, 类似Chord的后继节点表

◆ 每节点维护一邻居集M

- 包含 $|M|$ 个网络物理层与当前节点邻近的节点
- 以增强工作的局部性, 通常不使用

◆ 节点状态

- 状态数 = $L+R+M$
- 表项数 = $|L| + B \log_B N + |M|$;

◆ 路由算法

- 每步至少比前一步多匹配1位前缀, 直到无法再匹配更多位
- 即寻找当前节点的叶节点集中与目的ID的最接近的节点
- 符号说明

- ☞ A: 当前节点ID, D: 目的节点ID
- ☞ R_j^i : R中第j行的第i项
- ☞ L_i : 叶集L中离当前节点ID第i近的节点
- ☞ D_j : D的第j位
- ☞ $\text{shl}(A, B)$: A、B匹配的前缀长度

If D 位于叶集范围内

forward to L_i ; $|D-L_i|$ 最小者

Else利用R {

$j = \text{shl}(D, A)$; 求匹配前缀长度

if($R_j^{D_j} \neq \text{null}$) forward to $R_j^{D_j}$

else forward to T ∈ LURUM

$\text{shl}(T, D) \geq j$; 极少出现

$|T-D| < |A-D|$; 扩大路由

Routing 算法

```
(1) if ( $L_{\lfloor |L|/2 \rfloor} \leq D \leq L_{\lceil |L|/2 \rceil}$ ) {  
(2)   //  $D$  is within range of our leaf set  
(3)   forward to  $L_i$ , s.th.  $|D - L_i|$  is minimal;  
(4) } else {  
(5)   // use the routing table  
(6)   Let  $l = shl(D, A)$ ;  
(7)   if ( $R_l^{D_l} \neq null$ ) {  
(8)     forward to  $R_l^{D_l}$ ;  
(9)   }  
(10)  else {  
(11)    // rare case  
(12)    forward to  $T \in L \cup R \cup M$ , s.th.  
(13)       $shl(T, D) \geq l$ ,  
(14)       $|T - D| < |A - D|$   
(15)  }  
(16) }
```

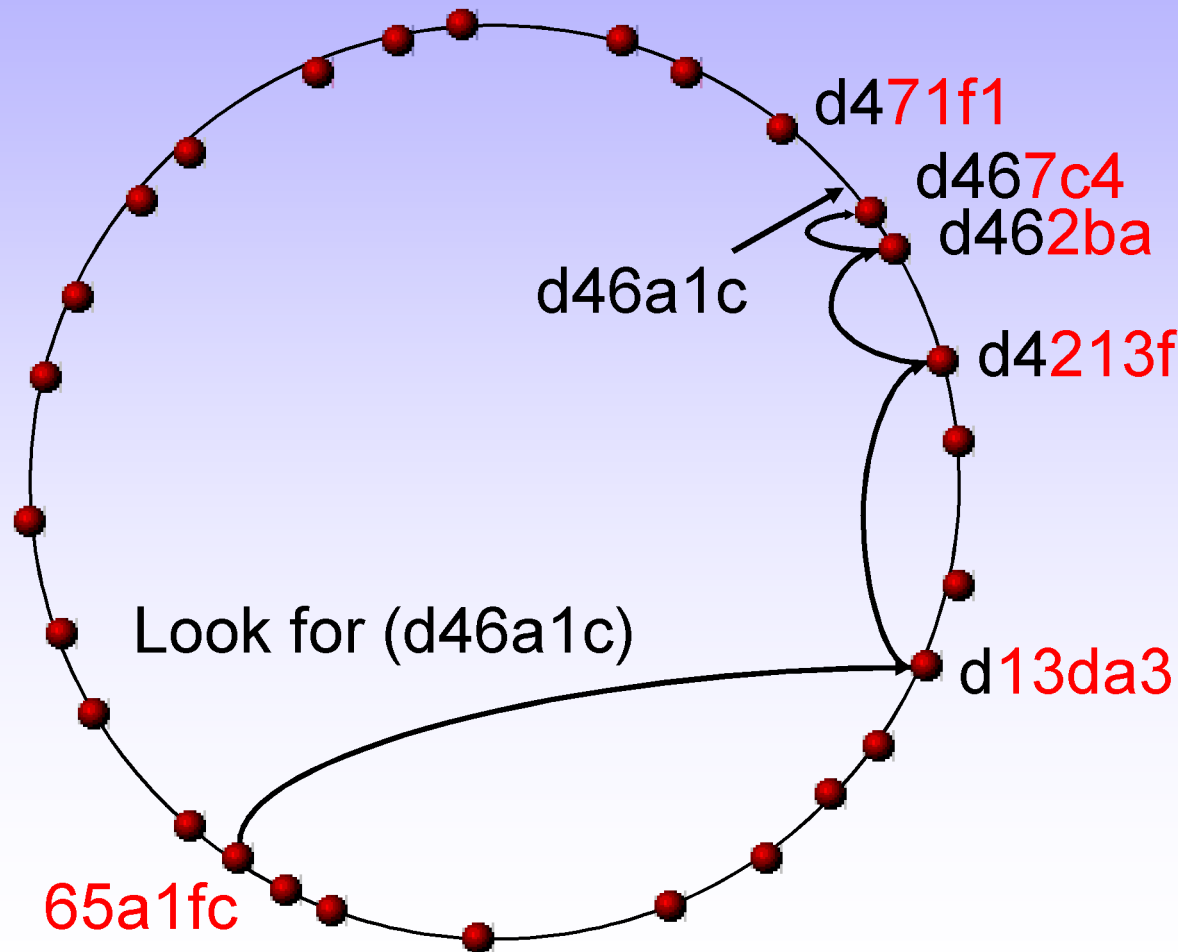
(1) Single hop

(2) Towards better prefix-match

(3) Towards numerically closer *NodeId*

D : Message Key
 L_i : i^{th} closest *NodeId* in leaf set
 $shl(A, B)$: Length of prefix shared by nodes A and B
 R_j^i : $(j, i)^{\text{th}}$ entry of routing table

Pastry路由



Properties

- $\log_2^b N$ steps
- $O(\log N)$ state

Pastry自组织和自适应

◆ Join Step1: 初始化R/L/M三表

- 新节点X加入，先通过某种方式联系到现存节点A
- X通过A发出一条以X为目的地的消息
- 最终达到离X最近的节点Z

◆ Join Step2: 通知其它节点自己的到来

- X把自己节点信息发给自的R/L/M中节点即可
- 剩下工作由收到更新消息的节点自己去做

◆ Join Step3: 新节点获取需要负责的数据

- 从ID最接近（如叶节点）的节点获取数据

◆ Fail Step1: 修正叶集

- 若某叶节点失效，通过未失效最远节点获取新叶集

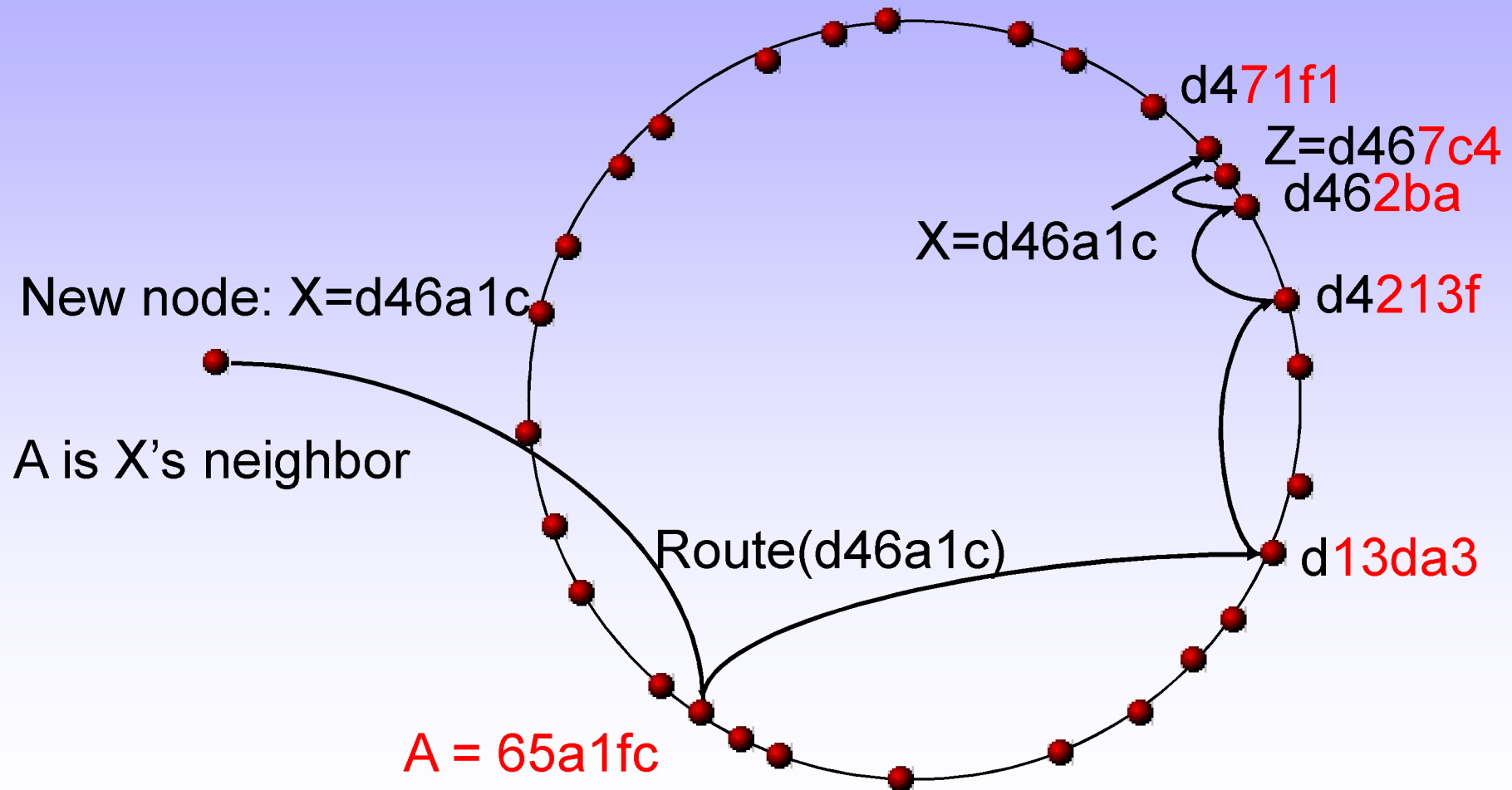
◆ Fail Step2: 修正路由表

- 若 R_{ld} 失效，发消息给第l行未失效第i项节点 ($i \neq d$)
- 找第l行第d项作为代替项

◆ Fail Step3: 修正邻居集

- 由于不常用，修正宽松
- 每个一个月修正
- 若存在失效邻居，则发消息给未失效邻居获取新的集

Pastry: Node addition



基于Pastry的归档系统PAST

- ◆ 广域P2P归档存储系统，提供安全高可用、持久性的数据存储服务
- ◆ 向用户提供需要证书合法的插入、查询、回收基本操作，且插入、回收还需“收据”确认
- ◆ 每个PAST用户有一权威机构发行的“智能卡”，是安全核心部件
- ◆ 存储管理2个目的
 - 不断平衡网络节点剩余的空闲存储空间
 - 保持每个数据对象有k个副本存放在nodeID相近但不同的k个节点上
 - 为此，采用“副本转移”、“文件转移”、“路径缓存”等方法

Pastry总结

- ◆ Generic p2p overlay network
- ◆ Scalable, fault resilient, self-organizing, secure, IDs are in base 2^b , randomly assigned from $\{0, \dots, 2^{128}-1\}$
- ◆ $O(\log_B N)$ routing steps (expected); $B=2^b$
- ◆ $O(\log_B N)$ routing table size
 - **b defines the tradeoff: $(\log_{2^b} N) \times (2^b - 1)$ entries Vs. $\log_{2^b} N$ routing hops**
- ◆ Network locality properties

4.4.6 新型P2P网络Kademlia

Chord的问题（1）：路由维护有延迟

- ◆ **节点加入：**Chord需要环形拓扑中的任意一个节点来协助完成，且加入过程包括新节点本身的Join操作和被其他节点发现两个阶段；
- ◆ **节点失效的处理：**Chord需要周期性对节点的前继节点和后继节点进行探测，并按照节点加入时的算法重建Finger表；
- ◆ **对于节点退出的处理：**Chord采取了将节点的退出当作为失效来处理的方式。

Chord的问题（2）：finger table有冗余。

- ◆ 以 2^i 为跨度来寻找路由表中的后继节点，这会在finger 中产生一定的冗余量，导致Chord的查询效率不高。

4.4.6 新型P2P网络Kademlia

I) 基本概念

◆ 常数度P2P网--理论上特殊的新型结构化

- 路由、定位、自组织方式与前4种区别不大
- 每个节点的“度”（连接数）是固定的，与规模无关
 - ☞ 维护固定路由表项，仍能达到 $O(\log N)$ 跳的指数定位效率
 - ☞ 路由表固定导致网络自适应开销减少

◆ 更容错实用的结构化网络Kademlia

- 路由方法类似Chord
- 但采用基于XOR的距离度量
- 将结构配置信息融合到每条消息，从而构建高容错和自适应的P2P新年息系统
- 加入、离开节点更加简单

KADEMLIA简介

- ◆ Kademlia协议是美国纽约大学的 Petar P. Maymounkov和David Mazi eres 在2002年发布的一项研究结果
《Kademlia: A peer-to-peer information system based on the XOR metric》。
- ◆ Kademlia 是一种分布式哈希表（DHT）技术，Kademlia通过独特的以异或算法（XOR）为距离度量基础，建立了一种全新的DHT 拓扑结构，相比于其他算法，大大提高了路由查询速度。

Kademlia

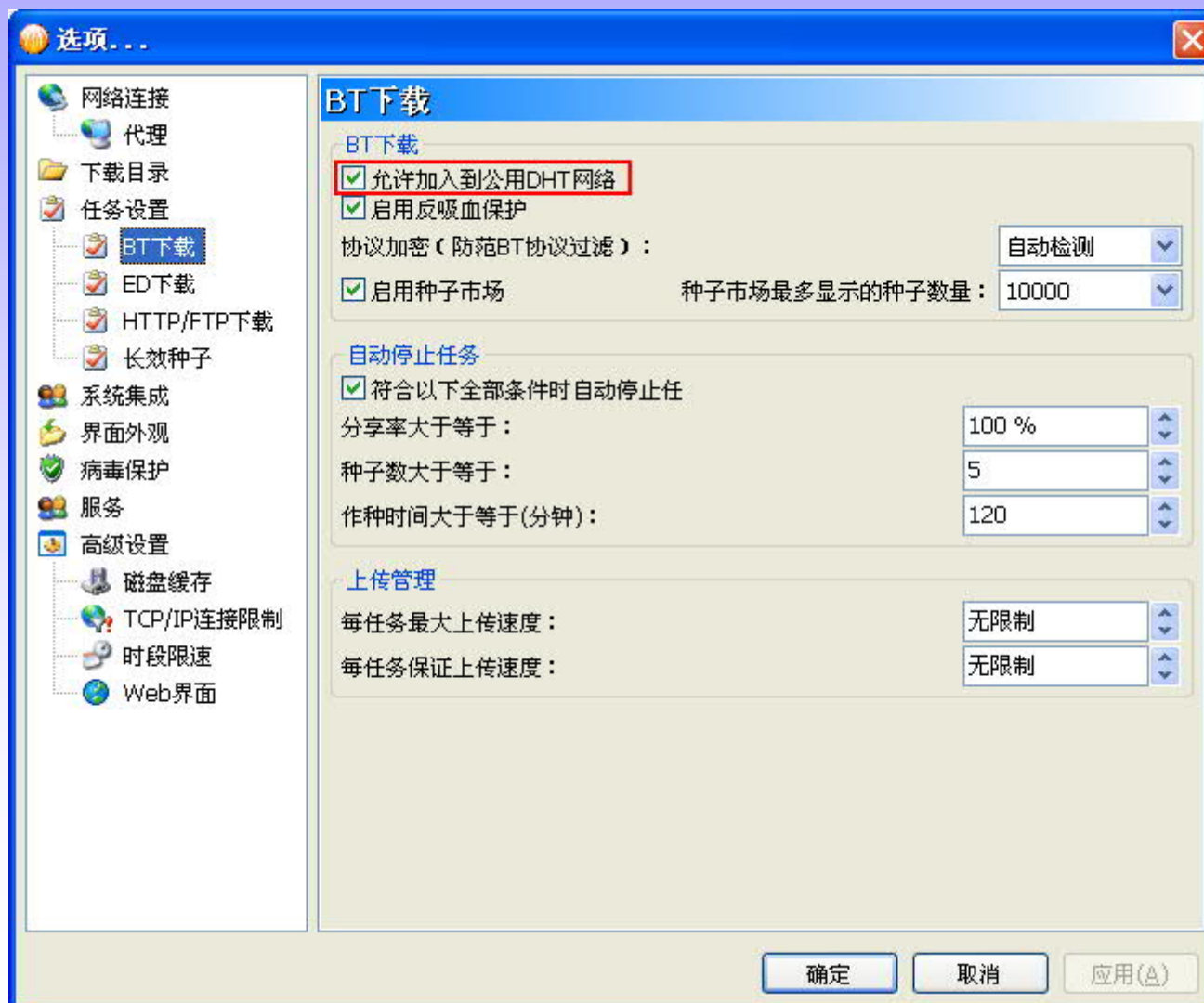
◆ Kademlia

- XOR度量不影响可**扩展性和工作效率**
- 提供**容错性**和**灵活性**
- XOR是对称的，从路由消息中可获得有用的网络配置信息
- “携带更新” 降低开销提高了自适应性
- 节点可发消息给其路由表中任意一段（interval）中的每个节点，让它们基于**时延返回路由下一跳**，还可发并行的异步消息

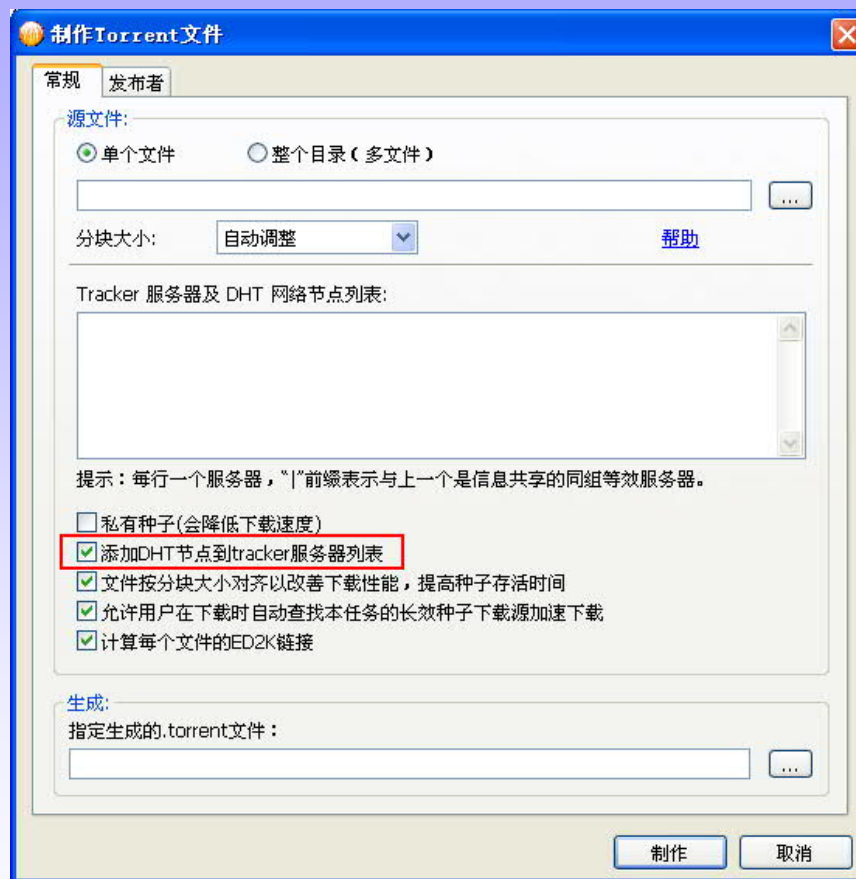
II) 当前应用现状

- ◆ 在2005 年5 月著名的Bi tTorrent 在4.1.0 版实现基于Kademl i a 协议的DHT 技术后，很快国内的Bi tComet 和Bi tSpi ri t 也实现了和Bi tTorrent 兼容的DHT 技术，实现trackerless 下载方式。
- ◆ 另外，emule 中也很早就实现了基于Kademl i a 类似的技术（BT中叫DHT，emule 中也叫Kad，），和BT 软件使用的Kad 技术的区别在于key、value 和node ID 的计算方法不同。

II) 当前应用现状 (Bi tComet)



II) 当前应用现状 (BitComet)



0 项选中/当前类别共有 6 项

test4444 已登录

DHT已连接节点:1435

外网:

✓ 允许加入到公用DHT网络

II) 当前应用现状 (迅雷)

迅雷链接:

1. 在原地址前面加"AA", 后面加"ZZ" (不包括引号), 地址变为
AAhttp://im.baidu.com/install/BaiduHi.exeZZ
2. 此地址base64编码为
QUFodHRwOi8vaW0uYmFpZHUuY29tL2Iuc3RhbGwvQmFpZHVl aS5l eGVaWg==
3. 迅雷专链即在上地址前加thunder://, 即
Thunder://QUFodHRwOi8vaW0uYmFpZHUuY29tL2Iuc3RhbGwvQmFpZHVl aS5l eGVaWg==

II) 当前应用现状（迅雷）

- 迅雷能够同时在Server和P2P上下载，所以号称是P2SP，即同时支持多种下载协议。
- 迅雷P2P可以直接支持bit torrent协议，也可以在用户使用迅雷打开一个URL下载地址时，迅雷会将该URL来源上传给迅雷服务器并储存，将目标文件数据发送至迅雷服务器进行比对检索，并在短时间内返回大量拥有该文件的目标来源。

II) 当前应用现状（迅雷）

迅雷支持的磁力链接

- 随着BT服务器的大量关闭，导致很多种子文件无法获取。既然资源查询只需要hash即可，那么种子文件便可以简化成一个链接，这便是磁力链接。磁力链接是以“magnet:?xt=urn:btih:”开头的字符串，后面则是资源文件的infohashes值。
- magnet:?xt=urn:btih:D3F7987EDF395104240AA16B30564B7372E55FFA该链接为变形金刚4的磁力链接，是从迅雷下载中直接复制得到。
- 由于HASH是不可逆的，所以从磁力链接上不能倒退出种子文件的信息，需要借助第三方种子库完成该转换。迅雷种子库hxxp://bt.box.n0808.com提供该服务

Kademlia协议的应用

所在网络及分野		具体应用说明
Overnet 网络	Overnet	已被整合到eDonkey2000中
	eDonkeyHybrid	混合式eDonkey软件
	ml Donkey	运行于多平台，多网络的eDonkey扩展版软件，支持多协议，如BitTorrent, eDonkey, DirectConnet, FastTrack, Gnutella2, FTP/HTTP, Kad, Cvernet, OpenNap, Soul Seek
Kad 网络	eMule	开始与0.40版
	ml Donkey	开始于2.5—28版
	aMule	即all-platform eMule，是eMule的扩展版，支持很多OS，开始于2.10
RevConnect		基于DirectConnet协议的P2P文件共享系统，以Kademlia为分布式Hash，能多源下载，部分文件共享，安全用户认证，自动资源回收、增强的自动搜索等功能，开始于.404版
KadC		用以在Overnet网络中发布、获取信息的C语言库
Azureus		开始于2.3.0.0版，使用Kademlia网络作为BitTorrent Trackers失效是的替代定位方法
BitTorrent		其Beta 4.1.0版拥有一个Kademlia网络，为无Trackers的Torrents服务
BitSpittit		基于BitTorrent协议的一个客户端，开始于3.0版
eXeem		基于BitTorrent网络的一个P2P文件共享软件，目的是取代BitTorrent中原有的Trackers，开始于Beta 0.25版

III) 节点间的异或距离

◆ 定义两个节点 x, y (ID值表示) 的“异或”距离 (非欧距离)

- 节点与数据对象都用SHA-1分配160 Bits ID,
- 对象索引由与objectID最接近的nodeID负责, “最接近”由XOR距离度量
- $d(x, y) = x \oplus y$, 如 $1011 \oplus 0010 = 1001 = 9 > 1011 \oplus 1010 = 0001 = 1$

◆ 异或距离的性质:

- 合理性: $d(x, x) = 0$
- 非负性: $d(x, y) \geq 0$
- 对称性: $d(x, y) = d(y, x)$, 对任何 x, y , 且 $x \neq y$. Chord不具备
- 三角不等式: $d(x, y) + d(y, z) \geq d(x, z)$
 - ☞ 因为 $d(x, z) = d(x, y) \oplus d(y, z)$
 - ☞ 对任意 $a \geq 0, b \geq 0, a + b \geq a \oplus b$
- 单向性: 任意节点 x 和距离 d , 系统中仅有唯一节点 y 满足 $d(x, y) = d$
 - ☞ 单向性保证相同数据对象的定位最终将会汇聚于相同的路径
 - ☞ 越往后走汇聚可能性越高, 于是路径缓存可提高定位效率
- 传递性: 显然 if $d1 \geq d2$ and $d2 \geq d3$, then $d1 \geq d3$ 成立

◆ 异或距离结构性好处

- 按当前节点ID与所有其它节点ID间XOR距离大小排队，知道目标结点ID后，就很容易计算出目标节点在这条长队中的位置；
- 如果给定一个异或距离，你也能很容易从这条长队里找出与该距离最接近的那些结点
- 节点0011实现了与自身前缀相似度越高的结点离自己越近（异或之后高位为0）

◆ 异或求距离

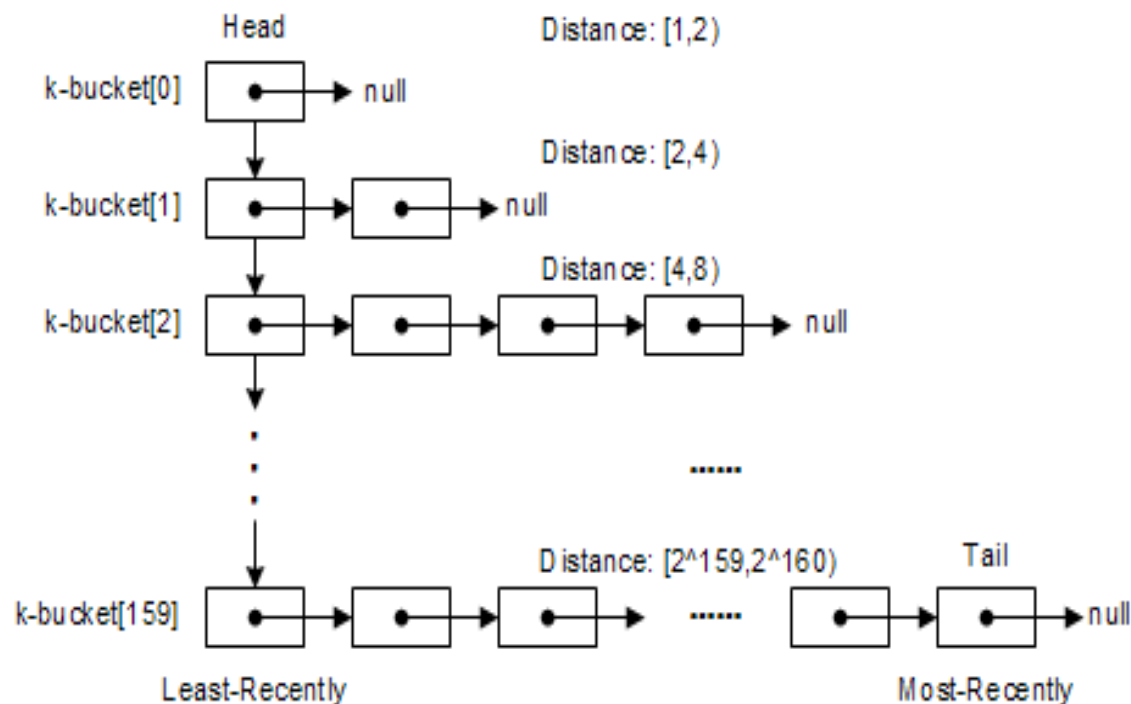
```
      010101
XOR  110001
-----
```

100100 = $32 + 4 = 36$ 。显然，高位更容易影响距离

IV) K-桶路由表

- ◆ K-buckets: 每节点维护一个路由表，它由 $\log N$ 个k桶构成
 - 实质是一个链表集，每节点 $0 \leq i < 160$ 个桶，每个桶的大小分理论
上依此为: 2^i ($0 \leq i < 160$); 每桶内装Max个 (10或20) 记录= (序
号i, 到自己的异或距离 $2^i - 2^{i+1}$, 节点信息)
 - 节点信息 = <拥有者IP, 下载端口, 拥有者节点ID >三元组
 - 表项以访问时间排序, 最近 (least-recently) 看到的放在头部, 最远 (most-recently) 看到的放在尾部
 - Kad网络是靠UDP协议交换信息

I	距离	邻居
0	$[2^0, 2^1)$	(IP address, UDP port, Node ID) _{0,1} (IP address, UDP port, Node ID) _{0,k}
1	$[2^1, 2^2)$	(IP address, UDP port, Node ID) _{1,1} (IP address, UDP port, Node ID) _{1,k}
2	$[2^2, 2^3)$	(IP address, UDP port, Node ID) _{2,1} (IP address, UDP port, Node ID) _{2,k}
.....		
i	$[2^i, 2^{i+1})$	(IP address, UDP port, Node ID) _{i,1} (IP address, UDP port, Node ID) _{i,k}
.....		



◆ Interval：间隔大小

- 对很小*i*，是与自己很近的节点
- 对很大*i*，是与自己很远的节点
- 限制表项增长, 上限为max，一般偶数，emule的max=10，OverNet的max=20，BT的max=8

◆ Kad路由表实际是一颗二叉树

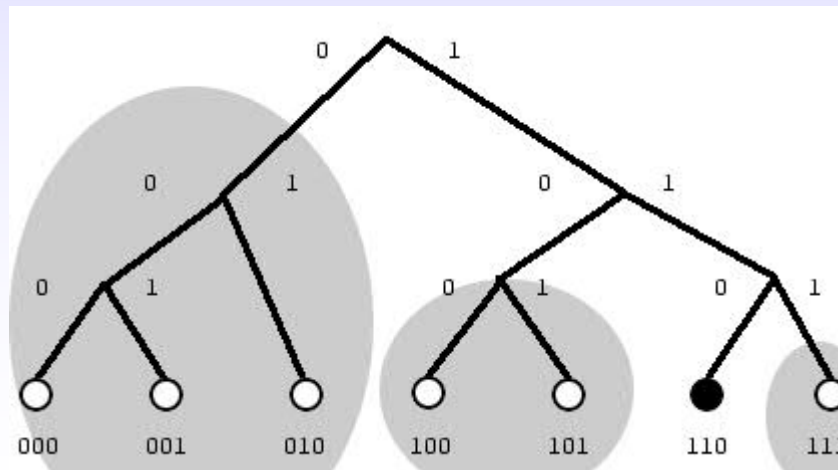
- 叶节点为K桶，存放有相同ID 前缀的节点信息
- 这个前缀就是该K桶在二叉树中的位置
- 每个K桶都覆盖了ID 空间的一部分，全部K桶的信息加起来就覆盖了整个160bit的ID空间，且不重叠

◆ K桶覆盖范围呈指数增长

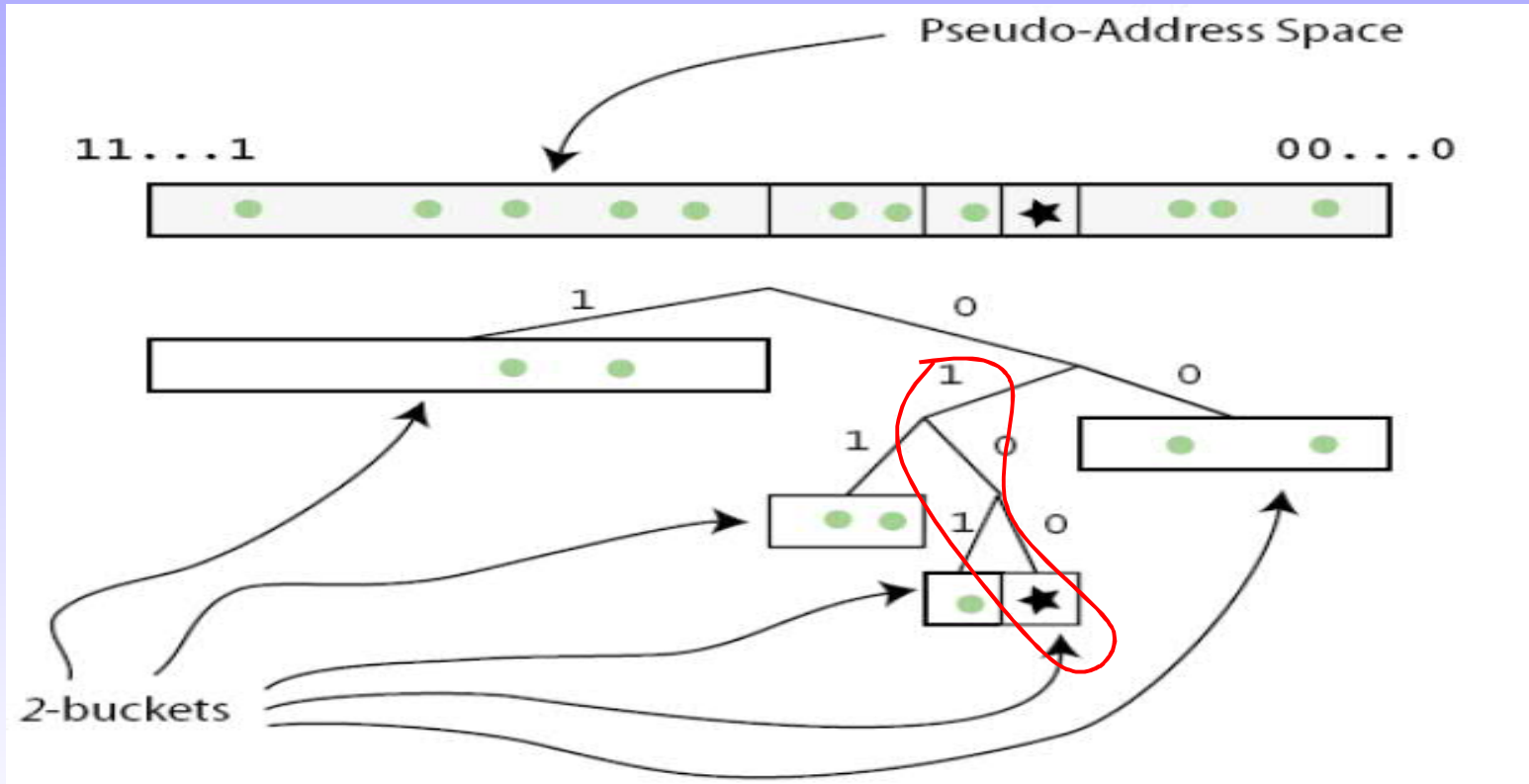
- 这就形成了离自己近的节点的信息多，离自己远的节点的信息少，从而可以保证路由查询过程是收敛。（每个桶包含M个节点，*i* 越大距离越远覆盖范围越大，）
- 指数划分区间，对有N个节点的Kad网络，最多只需要经过 $\log N$ 步查询，就可以准确定位到目标节点。

K桶可以看作一颗二叉树

- ◆ 所有**节点**都被当作一棵二叉树的**叶子**，且每个节点的位置都由其**ID值确定，相同前缀合并到相同子树**。
- ◆ 对于一个节点，都可以把这棵二叉树分解为一系列连续的，不包含自己的子树：如下图
 - 在最高层子树，左边是不包含自己的子树（000，001，010）
 - 下一层子树由剩下部分**不包含自己的一半**组成（100，101）
 - 依此类推，直到分割完整颗树（111）



节点0100的K-BUCKET分裂过程

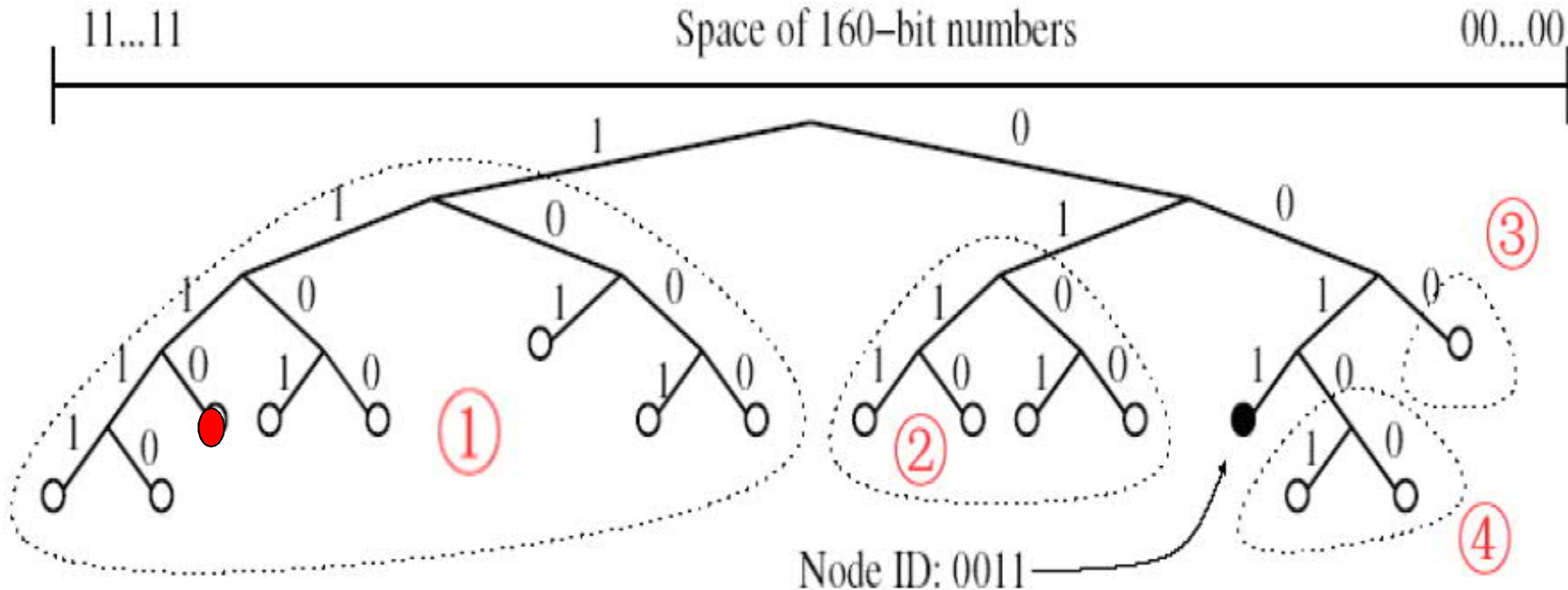


有了二叉树的概念，对于一个有N个节点的Kad网络，最多只需要经过 $\log N$ 步查询，就可以准确定位到目标节点。这个特性和Chord 网络上节点的finger table 划分距离空间的原理类似。

节点0011子树的划分

◆ 虚线包含部分都是0011看到的子树

- 由上到下各层的前缀分别为1, 01, 000, 0010
- Kademlia确保每个节点**知道其各子树的至少一个节点**, 只要这些子树非空
- 因此, 每节点都可以通过ID值来找到任何一个节点。这个路由的过程是通过XOR距离得到



V) 节点的行为（初始化）

节点初始化

◆ Kademlia读取本地配置文件

- ☞ src_index.dat, key_index.dat load_index.dat (索引)
- ☞ nodes.dat (上次程序启动时连接上的节点 RoutingZone.cpp)
- ☞ preferencesKad.dat (上次程序启动时本地节点IP、ID、Port)

◆ 生成ID

- ◆ 根据特定信息Hash或随机生成160位ID

◆ 构造本地结点二叉树

- 所有节点都被当作**一棵二叉树的叶子**，位置都由其ID值的最短唯一前缀确定
- 每个节点都在本地维护一个二叉树，来标示网络中节点与自己的距离远近
本地结点二叉树生成规则：
 - ☞ 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。

结点的行为（加入）

- ◆ 新节点u必需首先获得一个已经在网络中的结点w的 contact 信息；
- ◆ u首先将 w 插入到其对应的 k-bucket 中；
- ◆ u执行一个对自己id的 FIND_NODE 操作；根据收到的信息更新自己的K桶内容
- ◆ u对自己邻近节点由近及远的逐步查询刷新所有的 k-bucket；（刷新算法）
- ◆ 同时也把自己的信息发布到其他节点的K桶中（其他节点收到FIND_NODE也会更新K桶）

VI) 节点间的交互行为

◆ Kademlia协议包括四种远程RPC操作

- PING: 探测一节点, 判断其是否仍然在线。并在回应中携带网络地址
- STORE: 指示一节点存储一个<key , value>对, 以便查找
 - ☞ key 是对象的hash值, 即objectID
 - ☞ Value是真正的数据对象或其索引
- FIND_NODE: 以160bit ID 作为参数。本操作的接受者返回它所知离目标ID最近的a个节点的(IP address , UDP port , Node ID)三元组信息。(如果a=1, 类似于chord)
- FIND_VALUE: 以key为参数寻找key对应的value.
 - ☞ 同FIND_NODE过程
 - ☞ 若接受者已收到同一key的STORE消息, 则直接返回存储的value 值。(cache: 加速)
 - ☞ 一旦查询成功, 发起请求的节点会把 <key , value> 对存储在它所观察到的距离 key 最近但是没有返回 value 的节点上。(cache: 查找成功范围扩大)

VI) 节点间的交互行为

◆ Pi ggy back确认捎带

- PING 消息可以在其他RPC响应里使用捎带确认机制来获得发送者网络地址，避免多余的PONG

◆ 防止ID伪造

- 所有RPC都带一160 bi t随机 RPC ID，由发送者产生，
- 接收者返回消息里面必需拷贝此 RPC ID。目的是防止地址伪造。

更新K-BUCKET

- ◆ $x \leftarrow y$ 消息，则更新与 $d(x, y) = x \oplus y$ 对应的K桶
 - 若 y 已在于该K 桶中，把对应项移到该K 桶的尾部
 - 若 y 不在该K 桶中
 - ☞ 若该K桶不满k，把y的三元组插入队列尾部
 - ☞ 若该K桶满k，向头节点 z （最老）发RPC_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。（对应特点2）
 - 若某bucket过去1小时内没有任何的node lookup操作
 - ☞ 则这个node就随机搜索一个在该bucket覆盖范围内的id
- ◆ 特点
 1. 离自己越近的节点越容易放在K桶中，越远越欠了解
 2. 在线时间长的节点具有较高的可性继续保留在K桶中
 3. 可以保持系统稳定和减少节点进出的路由维护代价
 4. 若某节点长时间在线，则网络中有很多节点连接到该节点，其负载会随着在线时间的增大而增大，导致负载极不平衡

节点查找原理简述

◆ K桶中节点的选择

- 离自己异或距离越近的结点越容易被选到联系人列表里
 - ☞ 了解自己越近的节点，越远越欠了解，类似交朋友

◆ 联系人列表更改

- 不在线的联系人将被删除，被新找到的联系人所代替
- 要查找某个结点y时，如果y不在我的联系人列表里，我就从联系人列表里找到与y距离最近的结点z，
 - 然后向z查询y。如果z认识y，就把y的信息告诉我；
 - 如果z也不认识y，z就从自己的联系人列表里找到与y距离更近的w，
- 然后我向w查询y。这个递归过程不断重复，直到找到y为止

◆ 结点查找是进行信息存储和查询的基础。

节点查找（找目的节点的三元组）

◆ node lookup:

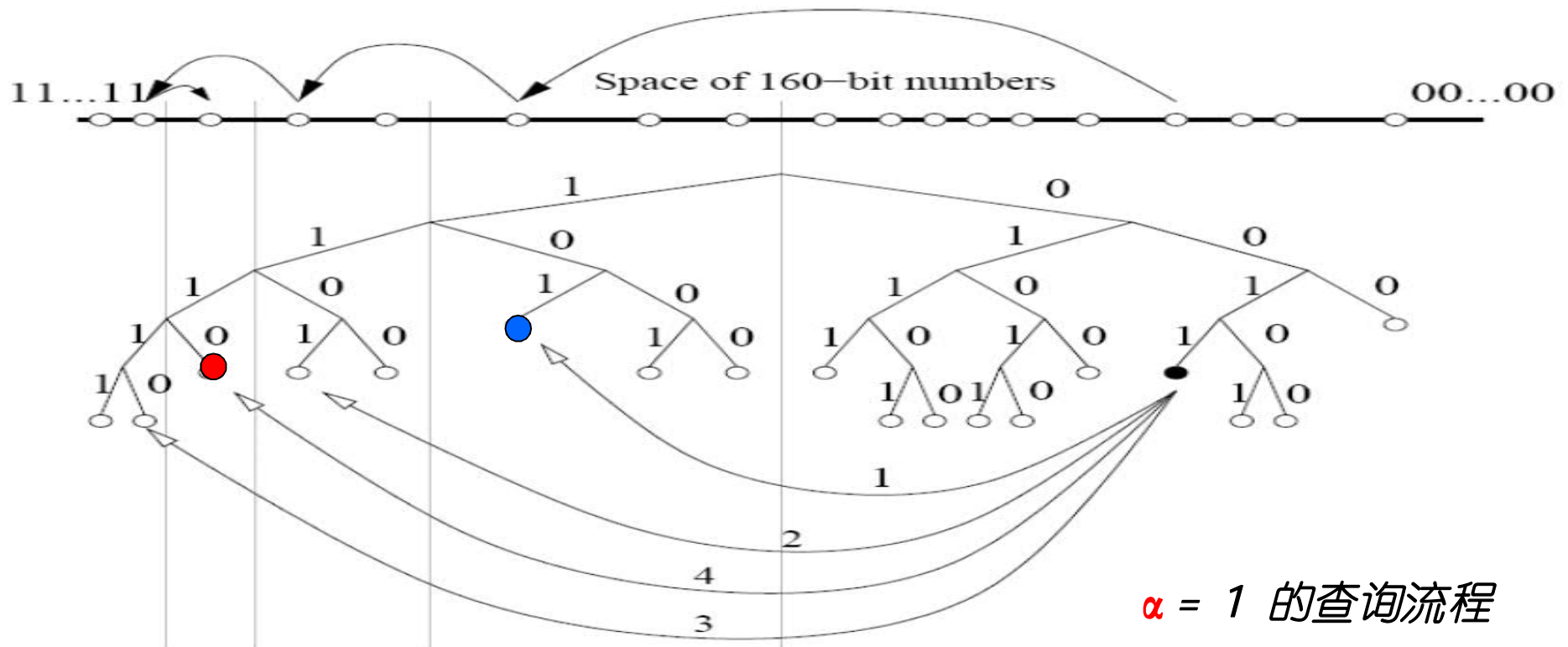
- 找到给定距离的 ID 最近的 a 个 node, a: 并发参数, 比如3
- Step1: 从最近的 k-bucket 里面取出 a 个最近的 node（不一定在同一个桶内），然后向这 a 个 node 并行发送异步的 FIND_NODE RPC
- Step2: 发送 FIND_NODE RPC 给前一步返回的 node, 不必等 a 个 PRC 都返回后才开始
 - ☞ 从发送者已知的最接近目标的 nodes 当中, 取出 a 个还没有查询的 node, 向这 a 个 node 发送 FIND_NODE RPC
 - 没有迅速响应的 node 将考虑排除, 直到其响应。
 - ☞ 若一轮 FIND_NODE RPC 没有返回一个比已知的所有 node 更接近目标ID, 发送者将再向a个最接近目标的、还没有查询的 node 发送 FIND_NODE RPC。
- Step3: 查找结束的条件: 发送者已经向a个最近的 node 发送了查询, 并且也得到了响应。

节点查找具体步骤

- ◆ **x**要查找ID值为**t**的节点，Kad递归操作步骤：
 - 计算到**t**的距离： $d = d(x, t) = x \oplus t$
 - 从**x**的第 $\lceil \log d \rceil$ 个K桶中取出 **a** 个节点信息，执行FIND_NODE
 - ☞ 若该**K桶**表项少于 **a** 个，则从附近多个桶中选择**最接近d** 的总共 **a** 个节点
 - 对受到查询的节点
 - ☞ 若自己是**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 **a** 个**节点**的信息给**x**
 - **x**对新接收到的每个节点
 - ☞ 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。
 - ☞ 没有迅速响应的节点将被迅速排除出候选列表，直到其响应。
 - 通过上述查找操作，**x** 得到了**a**个最接近**t** 的节点信息
 - Kad一般取 **a** = 20

◆ 节点0011查询节点1110（注意图中长度不够的ID后面补0）

- 计算距离 $d = 0011 \oplus 1110 = 1101 = 13$
- → 自举节点101（蓝色，一启动就存在的邻居节点），它计算 $101x \oplus 1110 = 0100 = 4$ ，查对应4的k桶内容有→1101节点；
- 1101节点计算 $1101 \oplus 1110 = 0011 = 3$ ，查对应3的k桶内容有→11110
- 11110节点计算 $11110 \oplus 1110x = 0010 = 2$ ，查对应2的k桶命中→1110
- 所以过程是收敛的



数据的存储

◆ 存储<Key, Value>的步骤:

- 用node lookup算法, 找到距离 Key 最近的 k 个 nodes
- 向这 k 个 nodes 发送 STORE RPC
- 每隔一定周期 (一个小时), 每个node重新发布(re-publish)<Key, Value>

◆ 更新要求:

- <Key, Value>的原始发布者每隔24小时重新发布一次
- 否则<Key, Value>将在发布之后的24小时 (某些应用更长) 之后过期

◆ 发布与搜寻的一致性

- 当节点w 发现新节点u比w 上更接近, 则w把自己的<key, value>对数据复制到u上, 但是并不会从w 上删除

节点数据的有效性保障

◆ 节点有效性

- 利用流经自己的节点查询操作，持续更新对应的K桶信息
- 对过去一个小时内还没收到任何节点查询操作的某个桶（bucket）执行刷新操作---BT 协议实现规定为15 分钟。
- 刷新操作：从该桶（bucket）中随机选择一节点执行一次 FIND_NODE 操作

◆ 数据有效性

- 特点：节点离开网络不发布任何信息（弹性网络特点或目标）
- 要求：每个Kad 节点必须周期性的发布（一个小时）本节点存放的全部<key , value>数据对，并把这些数据缓存在自己的k 个最近邻居处，
- 使失效节点上数据会被很快更新到其他新节点上。

4.4.7 结构化网络总结

◆ Chord/CAN/Tapestry/Pastry

◆ 目标相同

- 减少路由到指定文件的P2P跳数
- 减少每个Peer必须保持的路由状态

◆ 算法异同

- 节点与对象Hash映射到**同一空间**，走“**最接近**”路由
- 都保证算法的跳数与Peer群组的大小相关
- 方法上的差别很小

P2P 算法的比较

P2P系统	不同P2P定位算法的比较标准					
	模式	参数	跳数	路由状态	进出 Peers	可靠性
Napster	集中元数据索引/查S直接下载	无	常数	常数	常数	中心服务器返回多个下载位置, 客户端可在下载
Gnutella	广播请求直接下载	无	无保证	常数 (约3-7)	常数	接收多个可用数据的响应, 可重试
Chord	单维/循环ID空间	N: 群组的Peers数	$\log N$	$\log N$	$(\log N)^2$	所个顺序Peers重复数据, 应用可重试
CAN	多维ID空间	d: 维数	$d \cdot N^{1/d}$	$2d$	$2d$	多Peers响应, 应用可重试
Tapestry	全局Mesh	b: 所选标识符的基	$\log_b N$	$\log_b N$	$\log N$	重复数据在多个Peers上, 保持多踪迹
Pastry	全局Mesh	b: 所选标识符的基	$\log_b N$	$b \cdot \log_b N + b$	$\log N$	重复数据在多个Peers上, 保持多踪迹 多通路



◆ CAN和Chord

- CAN平面几何划分；Chord幂相邻关系
- 缺点：P2P网络没有考虑物理网络距离
- 信息共享

◆ Pastry

- 直接定位对象
- 缺点：研究不彻底

◆ Tapstry

- Root冗余（优）、动态构建、对象复制
- 缺点：softstate，路由失败，动态构建不彻底

优化性能的增强技术

◆ 复制(Replication)

- 把对象/文件的拷贝放在请求Peers附近, 最小化连接距离
- 改变数据时必须保持数据拷贝的一致性
- OceanStore基于冲突解的更新传播模式支持一致性语义

◆ 高缓(Cache)

- 减少获取文件/对象路径的长度, 进而Peers间交换消息数
- 这一减少很有意义-Peers间通信时延是严重的性能瓶颈
- Freenet: 命中文件传播到请求者途中所有节点高缓它
- 目标是最小化时延, 最大化请求吞吐率, 很少高缓大数据

◆ 智能路由和网络组织

- 社交“**小世界**”现象局部搜索策略, 代价与网络规模成子-线性增加
- OceanStore/Pastry网络上积极移动数据提高性能

DHT存在的问题和研究重点

- ◆ 解决P2P系统中固有的Churn高的现象，每个节点上下线都要 $O(\log N)$ 的修复操作，研究方向：**增大邻接表，增加发布及搜索冗余。**
- ◆ 模糊查询的支持
 - OverNet的查询存在的问题：Key不能太多，切词准确性
 - 让DHT支持复杂查询，不按照NodeID组织DHT，而按照关键字来组织（这方面MSRA那边在作些尝试）
 - 其他办法2：用DHT维护拓扑，而资源的定位仍然采用广播方式查询；分级的索引
- ◆ 文件的本地存储特性的消失
 - 目录的结构信息（本地相同目录下两个相关文件可能被Hash到不同的节点）
- ◆ DHT这种架构更容易受到攻击（研究重点）
 - 伪装节点
 - 拒绝转发消息等
- ◆ 不能吻合系统中节点的异质性（可能弱节点被分配到热门关键字）：
a) 对热门关键字作Cache b) 采用分层DHT结构如Keli ps，包括以物理位置为依据的分层，以兴趣为依据的分层等

P2P网络路由方式总结

- ◆ 服务器路由(Napster);
- ◆ 洪泛路由(Gnutella);
- ◆ 双层路由(KaZaA);
- ◆ 数值邻近路由(Chord);
- ◆ XOR邻近路由(Chord);
- ◆ 逐位匹配路由(Tapestry);
- ◆ 位置邻近路由(CAN), ;
- ◆ 层次路由(Viceroy);
- ◆ 混合式路由(Pastry)...

结论

- ◆ The key challenge of building wide area P2P systems is a scalable and robust location service
- ◆ Solutions covered in this course
 - Naptser: centralized location service
 - Gnutella: broadcast-based decentralized location service
 - Freenet: intelligent-routing decentralized solution (but correctness not guaranteed; queries for existing items may fail)
 - CAN, Chord, Tapestry, Pastry, Kademlia: intelligent-routing decentralized solution
 - ☞ Guarantee correctness
 - ☞ Tapestry (Pastry ?) provide efficient routing, but more complex

P2P技术新进展：Bi tCoin





P2P技术新进展：Bi tCoin

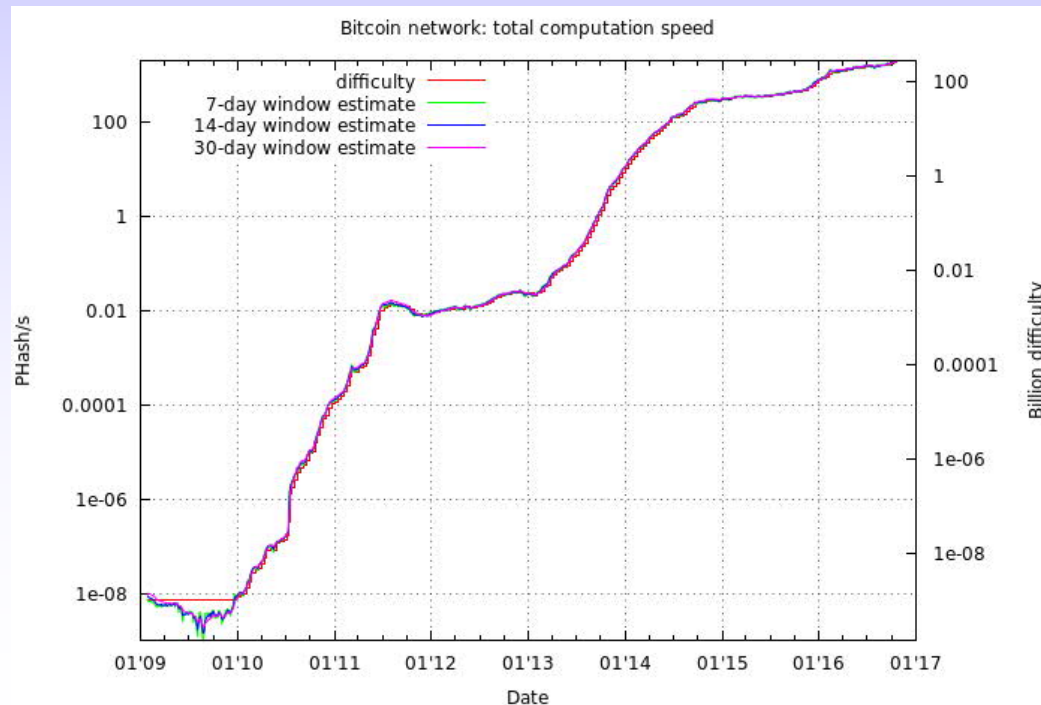
- 中本聪发明的一种P2P形式的虚拟货币。点对点的传输意味着一个去中心化的发行系统。
- Bitcoin: A Peer-to-Peer Electronic Cash System
- “We propose a solution to the double-spending problem using a peer-to-peer network”
- 比特币不依靠特定货币机构发行，它通过特定算法的大量计算产生，比特币经济使用整个P2P网络中众多节点构成的分布式数据库来确认并记录所有的交易行为。这个数据库称为区块链。
- 比特币与其他虚拟货币最大的不同，总数量将被永久限制在2100万个之内。

P2P技术新进展：Bi tCoin

- 区块链是所有比特币的交易的一份按照时间排序的公开记录。区块链在所有比特币用户中共享。它被用来验证比特币地址的有效额度和防止双重支付。
- 区块是区块链中间的一个记录，它确认了很多待确认的交易。平均大约每过10分钟，一个新的包含交易的区块会通过挖矿被追加到区块链里。
- 比特币挖矿就是搜集10分钟内产生的交易，然后按照指定的hash计算方法，搜索可以对交易结果hash到特定模式的随机数。
- Hash计算能力越强，产生正确hash模式的速度越快。

P2P技术新进展：Bi tCoin

- 伪造区块是一个消耗巨大计算力的过程，意味着攻击者需要掌握超过Bi tCoin整个P2P网络55%计算能力才能成功。
- 2016年4月统计：比特币全网的计算能力相当于全球最快的500台超级计算机计算能力加总之和的两倍。



主要参考文献

- ◆ Peer-to-Peer Computing Dejan S. Milojevic, Vana Kalogeraki, Rajan Lu Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, Zhi Chen Xu, HP Laboratories Palo Alto HPL-2002-57 March 8th, 2002
- ◆ 对等网络：结构、应用与设计；陈贵海等，清华大学出版社，2007.9
- ◆ Kademlia protocol
- ◆ Bitcoin: A Peer-to-Peer Electronic Cash System

参考书

