主要内容

- ◆应用层网络
  - ❖ CDN内容分发网络
  - ❖ 对等网络P2P
- ◆完成小作业5

专题5"应用层网络与网络安全"

1. Stoica, Ion, et al. "Chord: A scalable peer-to-peer lookup service for internet applications." ACM SIGCOMM Computer Communication Review 31.4 (2001): 149-160..
2. Piatek, Michael, et al. "Do incentives build robustness in BitTorrent." Proc. of NSDI. Vol. 7. 2007.
3. Dalton, Michael, et al. "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization." 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). 2018.
4. Goldberg, Sharon. "Why is it taking so long to secure internet routing?." Communications of the ACM 57.10 (2014): 56-63.
5. Kang, Qiao, et al. "Programmable in-network security for context-aware BYOD policies." Proc. USENIX Security. 2020.
6. Konte, Maria, Roberto Perdisci, and Nick Feamster. "Aswatch: An as reputation system to expose bulletproof hosting ases." Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication.

应用层：内容分发

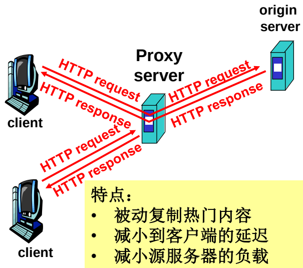
内容分发网络

Content Distribution Networks (CDNs)

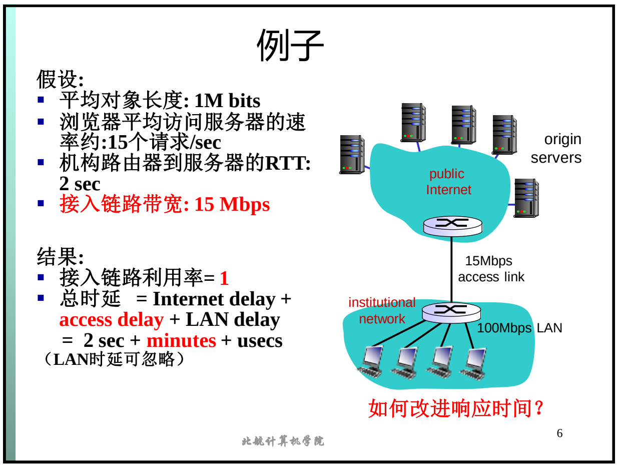
Web Proxy Caches

Web缓存，代理服务器

- ◆ 用户配置浏览器代理: Web accesses via cache
- ◆ 浏览器向代理发送HTTP请求
  - ❖ 若在缓存中: cache returns object
  - ❖ 否则: cache requests object from origin, then returns to client
- ◆ 通常由ISP部署Web缓存
  - ❖ 减少web流量，改善应用性能



特点：
- 被动复制热门内容
- 减小到客户端的延迟
- 减小源服务器的负载
- 减小网络负载、带宽开销
- 维护持久TCP连接



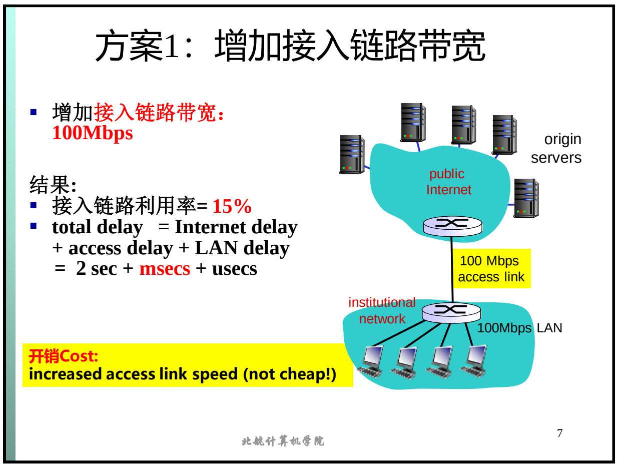
例子

假设:
- 平均对象长度: 1M bits
- 浏览器平均访问服务器的速率约:15个请求/sec
- 机构路由器到服务器的RTT: 2 sec
- 接入链路带宽: 15 Mbps

结果:
- 接入链路利用率= 1
- 总时延 = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs

（LAN时延可忽略）

如何改进响应时间？

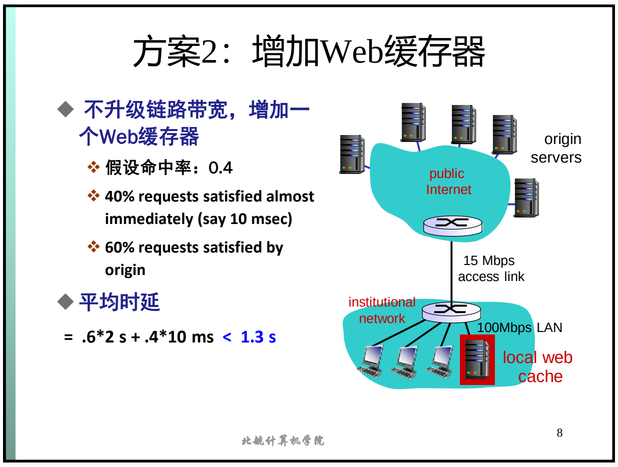


方案1：增加接入链路带宽

- 增加接入链路带宽：100Mbps

结果:
- 接入链路利用率= 15%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + usecs

开销Cost:
increased access link speed (not cheap!)

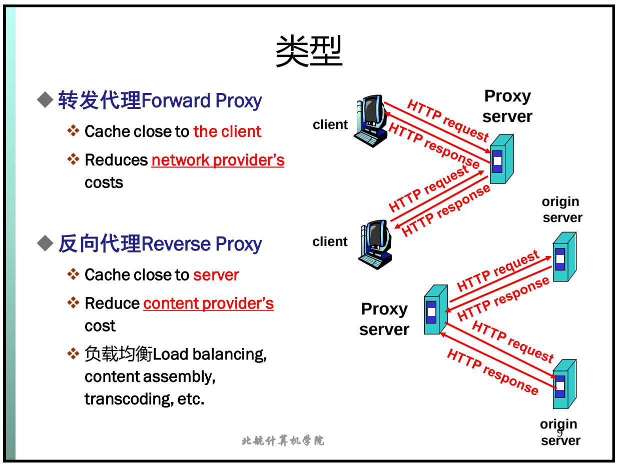


方案2：增加Web缓存器

◆ 不升级链路带宽，增加一个Web缓存器
  - ❖ 假设命中率：0.4
  - ❖ 40% requests satisfied almost immediately (say 10 msec)
  - ❖ 60% requests satisfied by origin

◆ 平均时延

= .6*2 s + .4*10 ms < 1.3 s

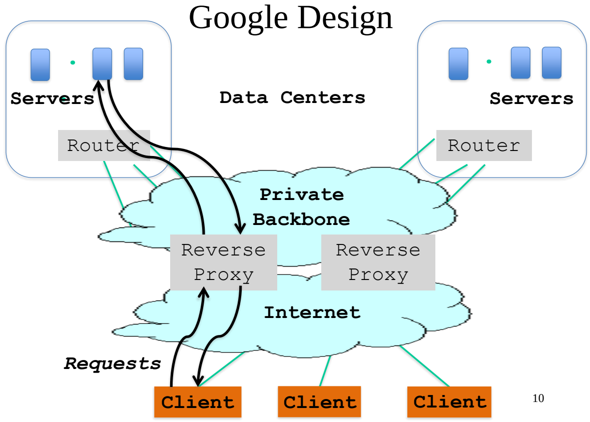


类型

◆ 转发代理Forward Proxy
  - ❖ Cache close to the client
  - ❖ Reduces network provider's costs

◆ 反向代理Reverse Proxy
  - ❖ Cache close to server
  - ❖ Reduce content provider's cost
  - ❖ 负载均衡Load balancing, content assembly, transcoding, etc.

Google Design



视频流和CDN

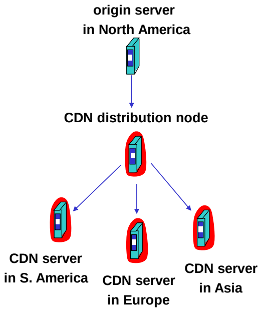
- 视频流量：major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- 挑战
  - 可扩展性：how to reach ~1B users?
  - 异构性heterogeneity：
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor
- *解决方案*: distributed, application-level infrastructure

YouTube NETFLIX hulu Akamai

内容分发网络CDN

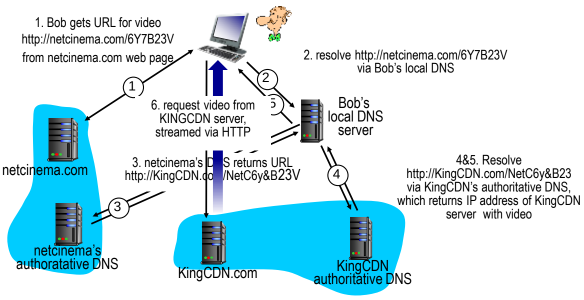
内容分发网络（Content Distribution Network，CDN）：内容提供商在Internet不同位置上建立分布式服务器，给客户提供内容。

- store/serve multiple copies of videos at multiple geographically distributed sites
- 主动内容复制
  - Content provider contracts with a CDN
- CDN 复制内容
  - On many servers spread throughout the Internet
- 更新内容
  - Updates pushed to replicas when the content changes



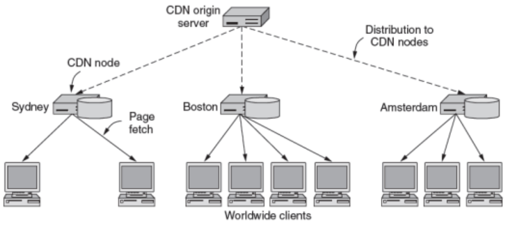
CDN：操作

- 浏览器检索一个特定的视频（由URL标识），CDN截获该请求
  - 确定合适的CDN服务器集群
  - 将客户请求重定向到该集群的某台服务器
- 利用DNS来截获和重定向请求
- 例1
  - Bob (client) requests video http://netcinema.com/6Y7B23V
  - video stored in CDN at http://KingCDN.com/NetC6y&B23V


1. Bob gets URL for video
http://netcinema.com/6Y7B23V
from netcinema.com web page
2. resolve http://netcinema.com/6Y7B23V
via Bob's local DNS
Bob's
local DNS
server
6. request video from
KINGCDN server,
streamed via HTTP
3. netcinema's DNS returns URL
http://KingCDN.com/NetC6y&B23V
4&5. Resolve
http://KingCDN.com/NetC6y&B23
via KingCDN's authoritative DNS,
which returns IP address of KingCDN
server with video
netcinema.com
netcinema's
authoratative DNS
KingCDN.com
KingCDN
authoritative DNS
1
2
3
4
5

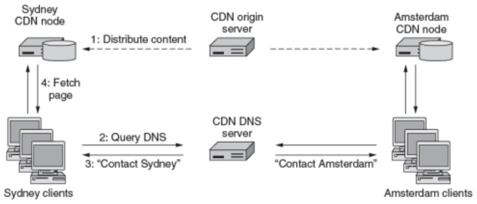
例：分发树

- CDNs对web服务器的扩展：客户端可以使用附近的节点作为服务器
  - 例：分发树
    - 客户端的访问方法：缓存；镜像；DNS重定向



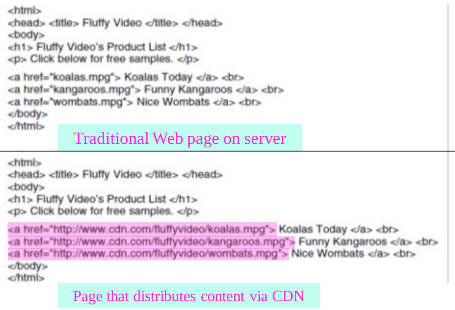
例：CDN使用DNS

- 使用DNS将客户端导向到附近的CDN节点:
  - Client query returns local CDN node as response
  - Local CDN node caches content for nearby clients and reduces load on the origin server



例：支持CDN的网页预处理

- Origin server rewrites pages to serve content via CDN



```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="koalas.mpg"> Koalas Today </a> <br>
<a href="kangaroos.mpg"> Funny Kangaroos </a> <br>
<a href="wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

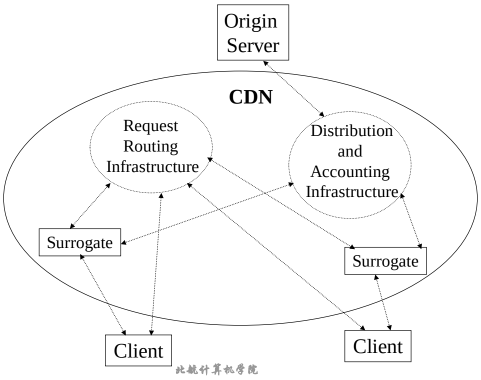
Traditional Web page on server

```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="http://www.cdn.com/fluffyvideo/koalas.mpg"> Koalas Today </a> <br>
<a href="http://www.cdn.com/fluffyvideo/kangaroos.mpg"> Funny Kangaroos </a> <br>
<a href="http://www.cdn.com/fluffyvideo/wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

Page that distributes content via CDN

CDN Architecture


Origin Server
CDN
Request Routing Infrastructure
Distribution and Accounting Infrastructure
Surrogate
Surrogate
Client
Client

CDN 系统组成

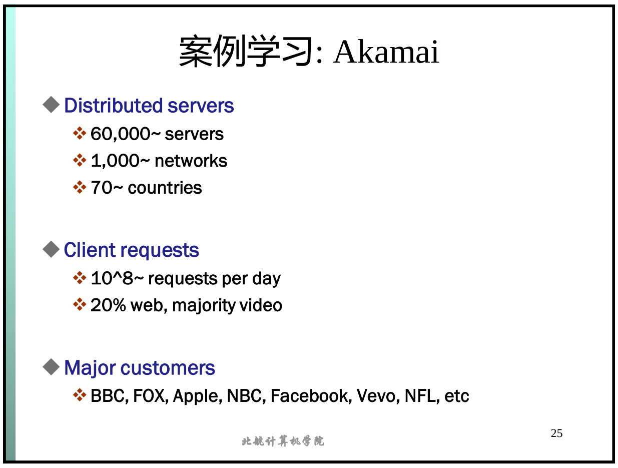
- 内容投递Content Delivery Infrastructure
  - Delivering content to clients from surrogates
- 请求路由Request Routing Infrastructure
  - Steering or directing content request from a client to a suitable surrogate
- 内容分发Distribution Infrastructure
  - Moving or replicating content from content source (origin server, content provider) to surrogates
- 记账 Accounting Infrastructure
  - Logging and reporting of distribution and delivery activities

挑战

- How to replicate content
- Where to replicate content
- How to find replicated content
- How to choose among know replicas
- How to direct clients towards replica

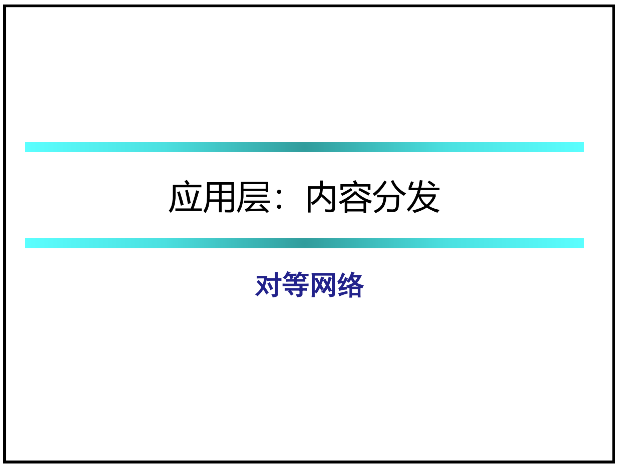
小结

- 内容分发的挑战
  - Many, diverse, changing objects
  - Clients distributed all over the world
  - Reducing latency is king
- 解决方法
  - 被动： Reactive caching
  - 主动：Proactive content distribution networks



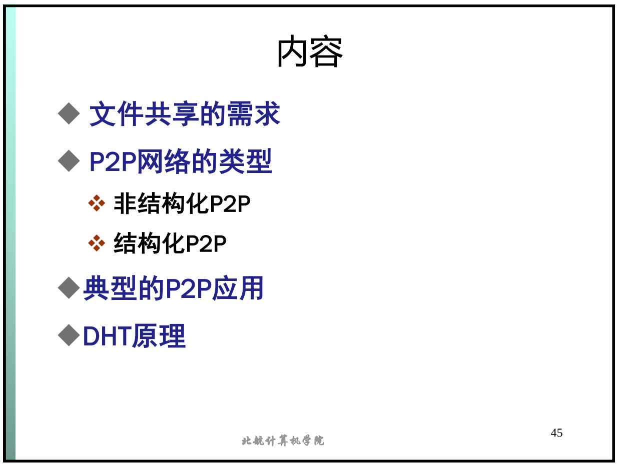
案例学习: Akamai

- Distributed servers
  - 60,000~ servers
  - 1,000~ networks
  - 70~ countries
- Client requests
  - 10^8~ requests per day
  - 20% web, majority video
- Major customers
  - BBC, FOX, Apple, NBC, Facebook, Vevo, NFL, etc



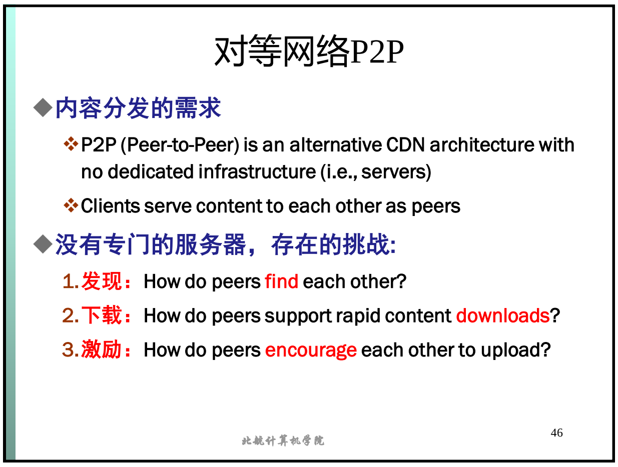
应用层：内容分发

对等网络



内容

- 文件共享的需求
- P2P网络的类型
  - 非结构化P2P
  - 结构化P2P
- 典型的P2P应用
- DHT原理

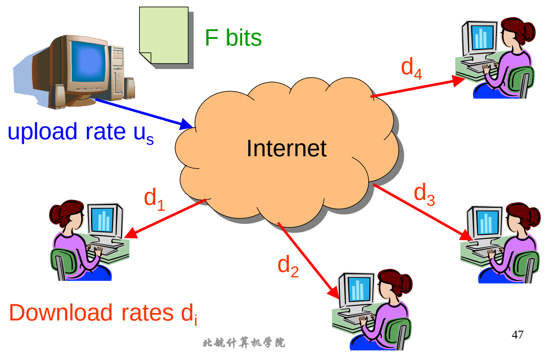


对等网络P2P

- 内容分发的需求
  - P2P (Peer-to-Peer) is an alternative CDN architecture with no dedicated infrastructure (i.e., servers)
  - Clients serve content to each other as peers
- 没有专门的服务器，存在的挑战:

1. 发现：How do peers find each other?
2. 下载：How do peers support rapid content downloads?
3. 激励：How do peers encourage each other to upload?

文件共享：客户 /服务器分发



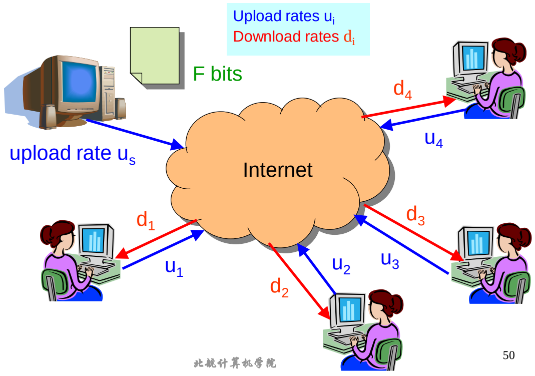
文件共享：客户/服务器分发

- ◆ 服务器向N个接收方发送一个大文件
  - ❖ 文件大小： *F* bits
  - ❖ 服务器上载速率： （upload rate） u_s
  - ❖ 接收方 *i* 的下载速率： （download rate） d_i
- ◆ 服务器发送数据
  - ❖ 服务器传送的数据量： *NF* bits
  - ❖ 花费时间 NF/u_s time
- ◆ 接收方接收数据
  - ❖ 最慢的接收方速率： $d_{min}= min_i\{d_i\}$
  - ❖ 至少需要的时间： F/d_{min}
- ◆ 下载时间（Download time）: $max\{NF/u_s, F/d_{min}\}$
- ◆ *若N足够大，分发时间随N线性增加*

如何加速分发?

- ◆在服务器端增加上载速率
  - ❖在一个服务器端使用更高的带宽
  - ❖多个服务器
  - ❖需要部署更多的基础设施
- ◆另一种解决方案: 借助于接收方
  - ❖接收端得到一个数据的拷贝
  - ❖将数据分发给其他的接收者
  - ❖减少服务器的负担

加速分发：P2P模式



加速分发：P2P模式

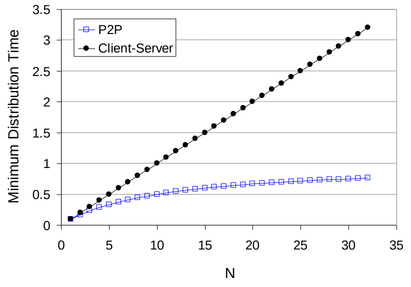
- 初始时，一个大型文件的拷贝
  - 大文件 *F* bits，服务器的上载速率 u_s
  - 对等点 *i* 的下载速率 d_i 和上载速率 u_i
- 分发时延
  - *服务器：发送时间为*F/u_s
  - *最慢的客户端接收时间：* F/d_{min}
- 使用所有上载资源需要的总时间
  - 总的上载bits: *NF*
  - 总的下载带宽: $u_s + \Sigma_i(u_i)$
  - *最小分发时间*$NF/(u_s+\Sigma_i(u_i))$
- Total: $max\{F/u_s, F/d_{min}, NF/(u_s+\Sigma_i(u_i))\}$

两种模型比较

- 下载时间
  - Client-server: $max\{NF/u_s, F/d_{min}\}$
  - Peer-to-peer: $max\{F/u_s, F/d_{min}, NF/(u_s+\Sigma_i(u_i))\}$
- P2P是自扩展的（self-scaling）
  - 对服务器带宽的增长要求很小
  - 分发时间的增长远小于节点的增长

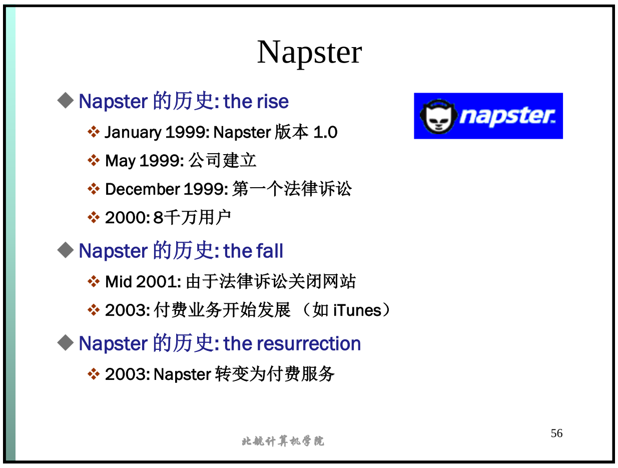
Client-server vs. P2P: 分发时间

client upload rate = *u*, *F/u* = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



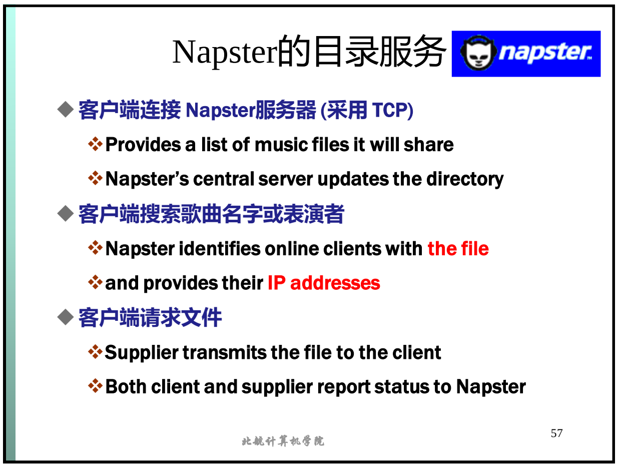
P2P网络的类型

- 无结构 Unstructured Peer-to-Peer Networks
  - Napster: Central directory
  - Gnutella: Query flooding
  - KaZaA, modern Gnutella: Hierarchical overlay
  - BitTorrent，Skype
- 有结构 Structured Networks 和 Distributed Hash Tables (DHT)
  - Chord (MIT/Berkeley)
  - CAN (ICIR/Berkeley)
  - Pastry (Rice)
  - Tapestry (Berkeley)



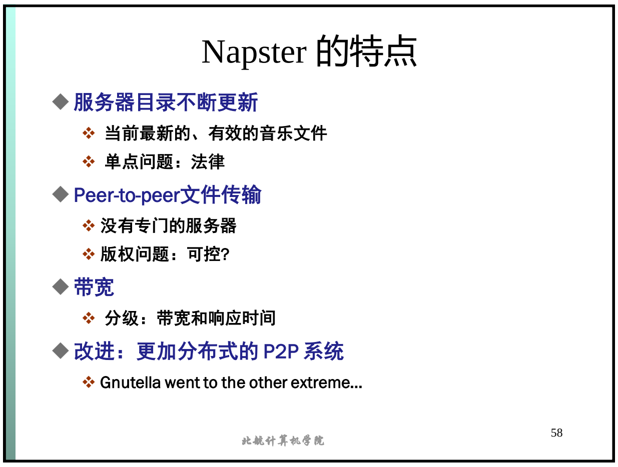
Napster

- Napster 的历史: the rise
  - January 1999: Napster 版本 1.0
  - May 1999: 公司建立
  - December 1999: 第一个法律诉讼
  - 2000: 8千万用户
- Napster 的历史: the fall
  - Mid 2001: 由于法律诉讼关闭网站
  - 2003: 付费业务开始发展 （如 iTunes）
- Napster 的历史: the resurrection
  - 2003: Napster 转变为付费服务



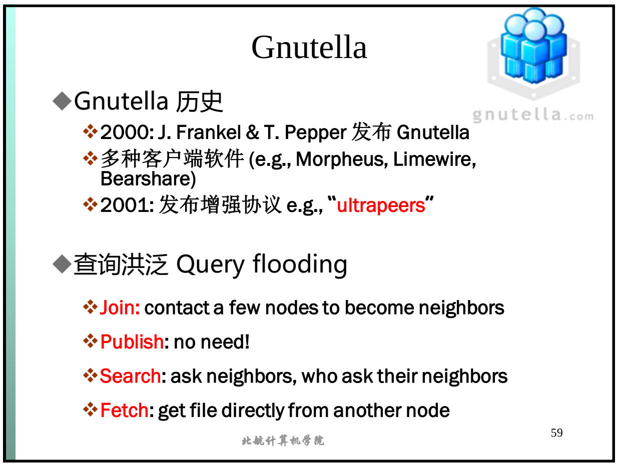
Napster的目录服务

- **客户端连接 Napster服务器 (采用 TCP)**
  - **Provides a list of music files it will share**
  - **Napster's central server updates the directory**
- **客户端搜索歌曲名字或表演者**
  - **Napster identifies online clients with the file**
  - **and provides their IP addresses**
- **客户端请求文件**
  - **Supplier transmits the file to the client**
  - **Both client and supplier report status to Napster**



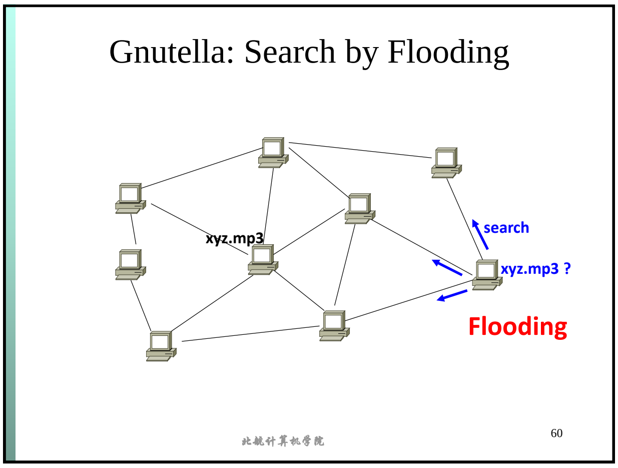
Napster 的特点

- **服务器目录不断更新**
  - **当前最新的、有效的音乐文件**
  - **单点问题：法律**
- **Peer-to-peer文件传输**
  - **没有专门的服务器**
  - **版权问题：可控?**
- **带宽**
  - **分级：带宽和响应时间**
- **改进：更加分布式的 P2P 系统**
  - Gnutella went to the other extreme...

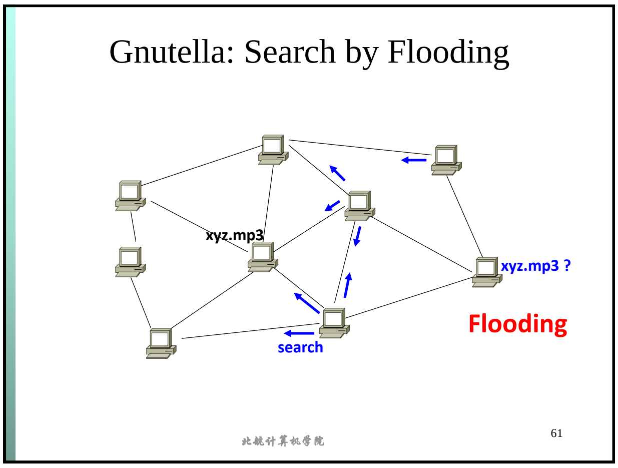


Gnutella

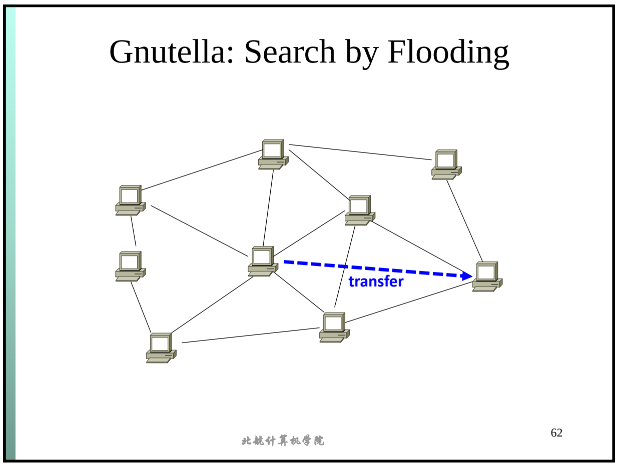
- Gnutella 历史
  - 2000: J. Frankel & T. Pepper 发布 Gnutella
  - 多种客户端软件 (e.g., Morpheus, Limewire, Bearshare)
  - 2001: 发布增强协议 e.g., "ultrapeers"
- 查询洪泛 Query flooding
  - Join: contact a few nodes to become neighbors
  - Publish: no need!
  - Search: ask neighbors, who ask their neighbors
  - Fetch: get file directly from another node



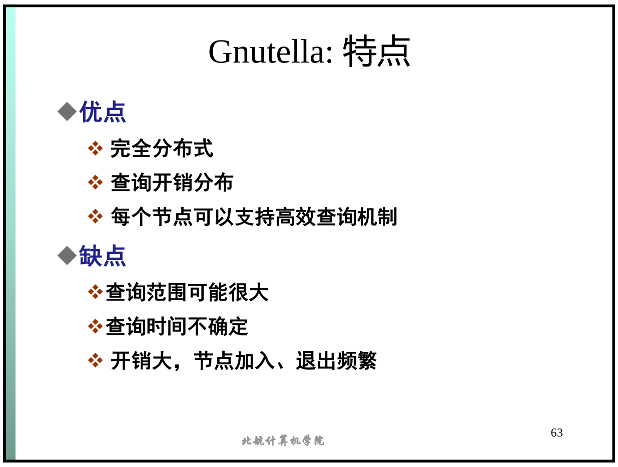
Gnutella: Search by Flooding



Gnutella: Search by Flooding



Gnutella: Search by Flooding



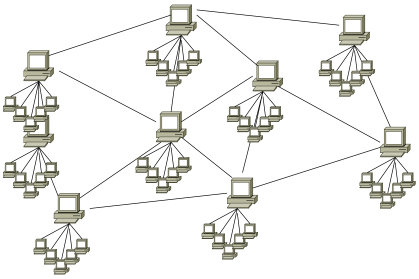
Gnutella: 特点

- 优点
  - 完全分布式
  - 查询开销分布
  - 每个节点可以支持高效查询机制
- 缺点
  - 查询范围可能很大
  - 查询时间不确定
  - 开销大，节点加入、退出频繁

KaAzA

- In 2001, KaZaA created by Dutch company Kazaa BV
- 两级层次结构
  - Gnutella技术和Napster的综合
- 超级节点 Supernodes
  - Join: on start, the client contacts a super-node
  - Publish: client sends list of files to its super-node
  - Search: queries flooded among super-nodes
  - Fetch: get file directly from one or more peers

“Ultra/super peers” in KaZaA and later Gnutella

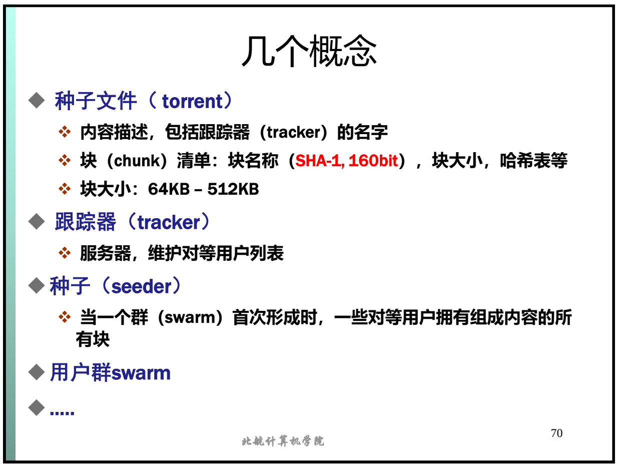


BitTorrent

- 2002: B. Cohen 发布 BitTorrent
- 开放标准
- 动机: 共享流行的内容资源
  - 流行的东西是暂时的
- 关注于有效的下载，而非搜索
  - 参与一个文件分发的所有对等方的集合为一个流（或种子文件）（torrent）
  - 彼此下载相同长度的文件块64KB～512KB（典型 256KB）
  - 基础设施点：追踪器（tracker）
- 避免搭便车 free-loading
  - 激励每个peer做贡献

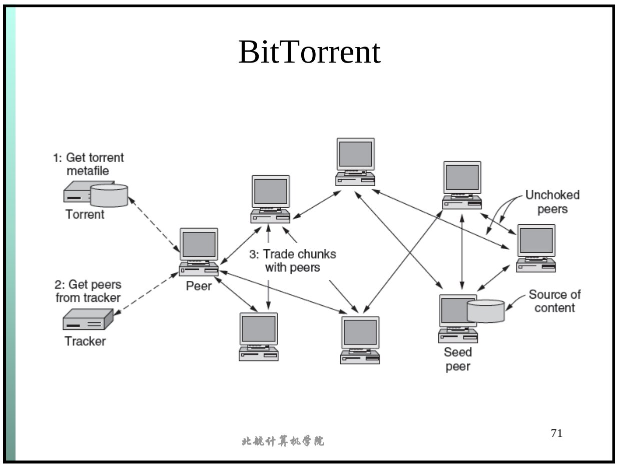
BitTorrent: Overview

- 过程:
  - Join: 连接到中心的 “tracker” 服务器, 获得一组对等点的列表
  - Publish: 运行 tracker 服务器
  - Search: 带外. 例如, 使用 搜索引擎 来寻找文件的 tracker
  - Fetch: 从对等点下载文件块，或上载文件块

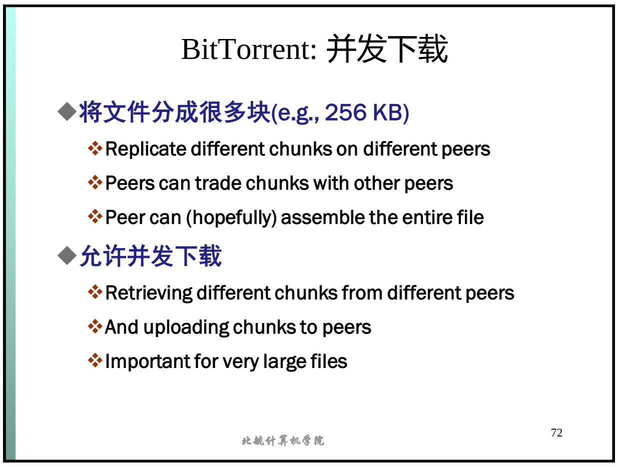


几个概念

- 种子文件（ torrent）
  - 内容描述，包括跟踪器（tracker）的名字
  - 块（chunk）清单：块名称（SHA-1, 160bit），块大小，哈希表等
  - 块大小：64KB – 512KB
- 跟踪器（tracker）
  - 服务器，维护对等用户列表
- 种子（seeder）
  - 当一个群（swarm）首次形成时，一些对等用户拥有组成内容的所有块
- 用户群swarm
- .....

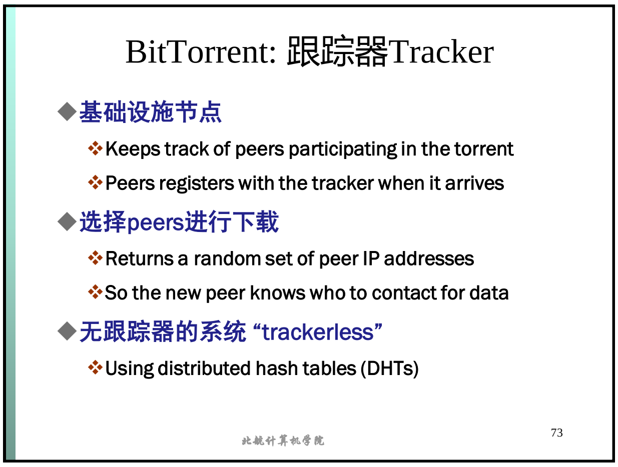


BitTorrent



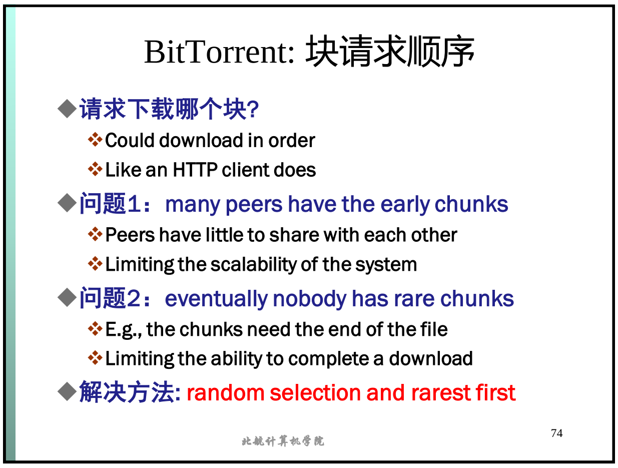
BitTorrent: 并发下载

- 将文件分成很多块(e.g., 256 KB)
  - Replicate different chunks on different peers
  - Peers can trade chunks with other peers
  - Peer can (hopefully) assemble the entire file
- 允许并发下载
  - Retrieving different chunks from different peers
  - And uploading chunks to peers
  - Important for very large files



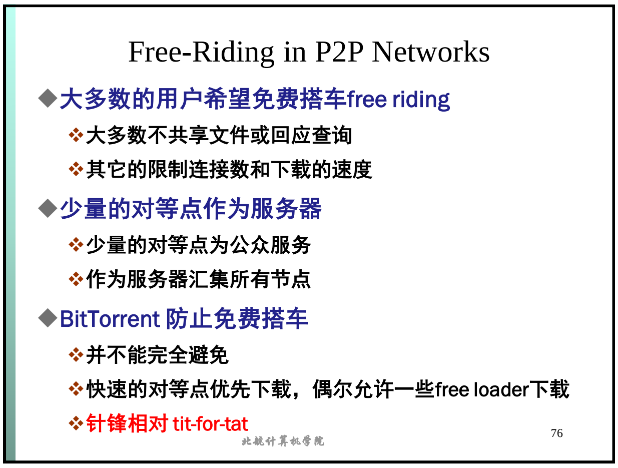
BitTorrent: 跟踪器Tracker

- 基础设施节点
  - Keeps track of peers participating in the torrent
  - Peers registers with the tracker when it arrives
- 选择peers进行下载
  - Returns a random set of peer IP addresses
  - So the new peer knows who to contact for data
- 无跟踪器的系统 “trackerless”
  - Using distributed hash tables (DHTs)



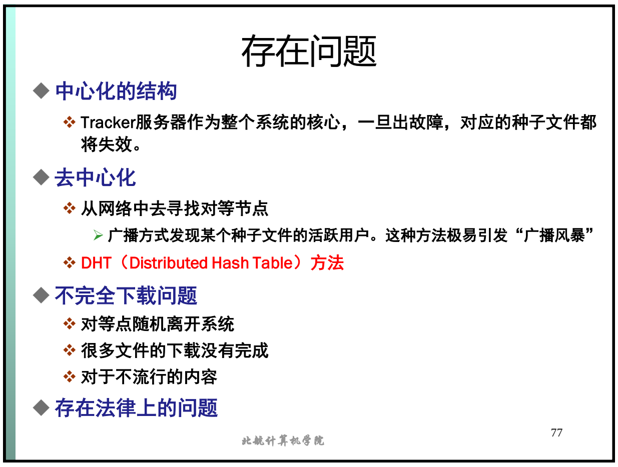
BitTorrent: 块请求顺序

- 请求下载哪个块?
  - Could download in order
  - Like an HTTP client does
- 问题1：many peers have the early chunks
  - Peers have little to share with each other
  - Limiting the scalability of the system
- 问题2：eventually nobody has rare chunks
  - E.g., the chunks need the end of the file
  - Limiting the ability to complete a download
- 解决方法: random selection and rarest first



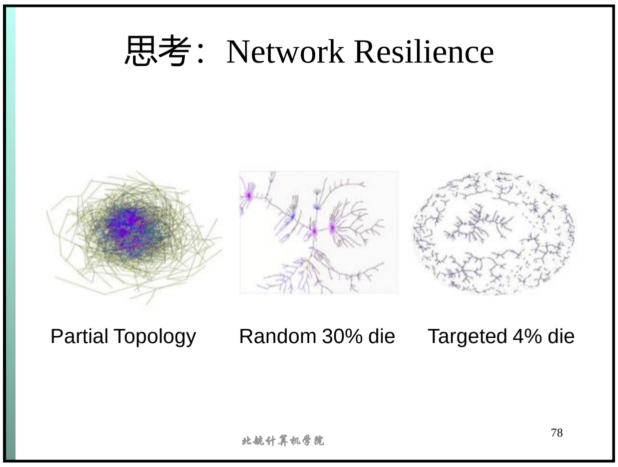
Free-Riding in P2P Networks

- 大多数的用户希望免费搭车free riding
  - 大多数不共享文件或回应查询
  - 其它的限制连接数和下载的速度
- 少量的对等点作为服务器
  - 少量的对等点为公众服务
  - 作为服务器汇集所有节点
- BitTorrent 防止免费搭车
  - 并不能完全避免
  - 快速的对等点优先下载，偶尔允许一些free loader下载
  - 针锋相对 tit-for-tat



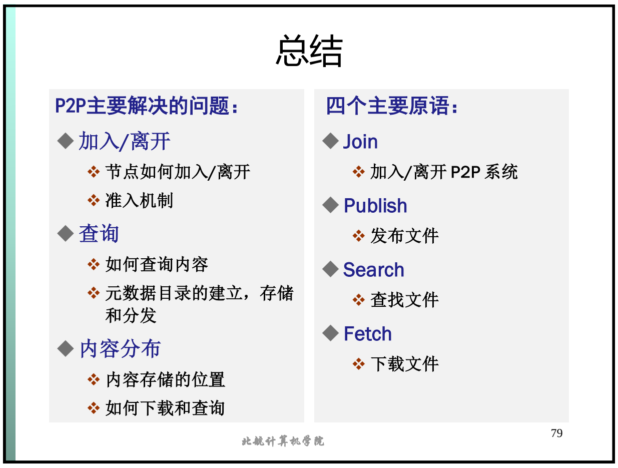
存在问题

- 中心化的结构
  - Tracker服务器作为整个系统的核心，一旦出故障，对应的种子文件都将失效。
- 去中心化
  - 从网络中去寻找对等节点
    - 广播方式发现某个种子文件的活跃用户。这种方法极易引发“广播风暴”
  - DHT（Distributed Hash Table）方法
- 不完全下载问题
  - 对等点随机离开系统
  - 很多文件的下载没有完成
  - 对于不流行的内容
- 存在法律上的问题



思考： Network Resilience

Partial Topology Random 30% die Targeted 4% die



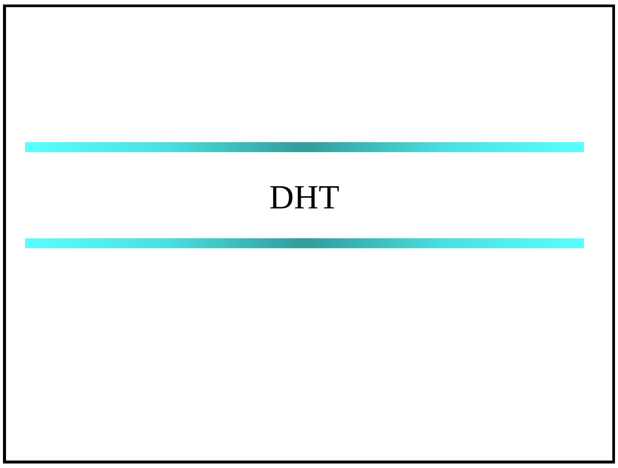
总结

P2P主要解决的问题：

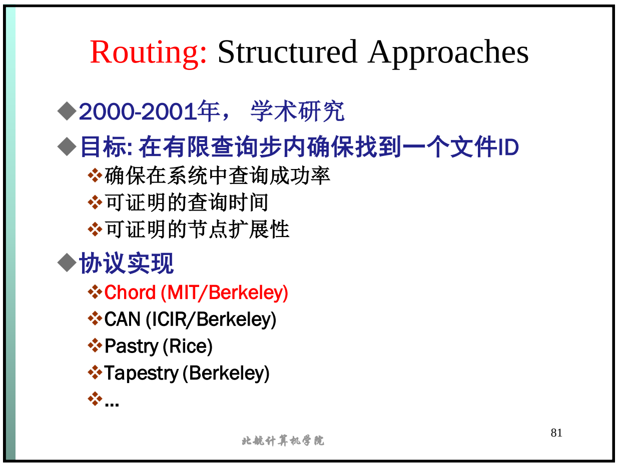
- ◆ 加入/离开
  - ❖ 节点如何加入/离开
  - ❖ 准入机制
- ◆ 查询
  - ❖ 如何查询内容
  - ❖ 元数据目录的建立，存储和分发
- ◆ 内容分布
  - ❖ 内容存储的位置
  - ❖ 如何下载和查询

四个主要原语：

- ◆ Join
  - ❖ 加入/离开 P2P 系统
- ◆ Publish
  - ❖ 发布文件
- ◆ Search
  - ❖ 查找文件
- ◆ Fetch
  - ❖ 下载文件

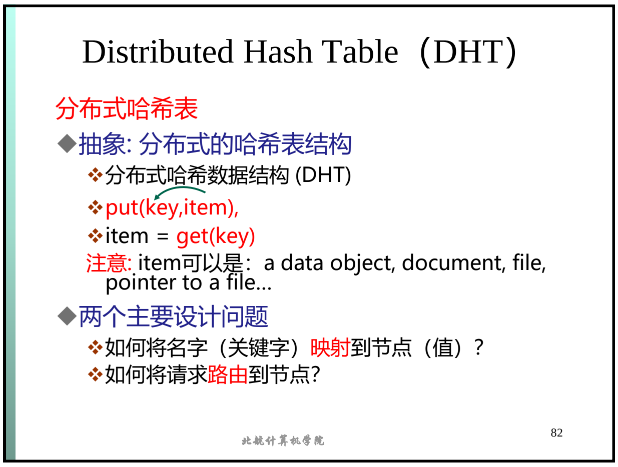


DHT



Routing: Structured Approaches

- ◆ 2000-2001年， 学术研究
- ◆ **目标: 在有限查询步内确保找到一个文件ID**
  - ❖ 确保在系统中查询成功率
  - ❖ 可证明的查询时间
  - ❖ 可证明的节点扩展性
- ◆ **协议实现**
  - ❖ Chord (MIT/Berkeley)
  - ❖ CAN (ICIR/Berkeley)
  - ❖ Pastry (Rice)
  - ❖ Tapestry (Berkeley)
  - ❖ ...



Distributed Hash Table（DHT）

分布式哈希表

- ◆ 抽象: 分布式的哈希表结构
  - ❖ 分布式哈希数据结构 (DHT)
  - ❖ put(key,item),
  - ❖ item = get(key)

  注意: item可以是：a data object, document, file, pointer to a file…
- ◆ 两个主要设计问题
  - ❖ 如何将名字（关键字）映射到节点（值）？
  - ❖ 如何将请求路由到节点?

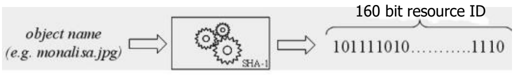
DHT: 概述

◆路由结构:

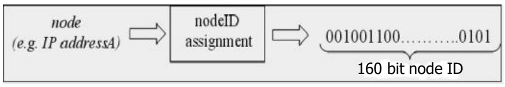
- ❖Join: 初始化时，连接一个 “bootstrap” 节点，并加入分布式数据结构中; 获得一个*node id*
- ❖Publish: 将 *file id* 发布在最近的 *node id* 上
- ❖Search: 路由一个 *file id* 的查询到最近的 *node id* 上，DHT数据结构的组织方式确保查询和发布信息匹配
- ❖Fetch: 两个选项:
  - ➢查询结果为128.2.1.3 has X, 使用 IP 地址128.2.1.3下载文件X
  - ➢查询结果为实际文件 => 下载文件

DHT例子

◆ 将 文件名 (e.g. monalisa.jpg) 转换为一个 160 bit 的标识符（identifier），使用 SHA－1算法



◆ 同样，将节点的 IP address 也转换为一个 160 bit 的标识符（identifier），使用同样的算法



◆节点标识符以降序的方式排成环

◆每个对象由环上最接近（closest）它的ID号的节点维护

DHT 实例： Chord

问题：如何找到一个对等节点，并下载文件?

◆在一个一维的空间中（环），为每一个节点和文件分配一个唯一的ID (Ring)

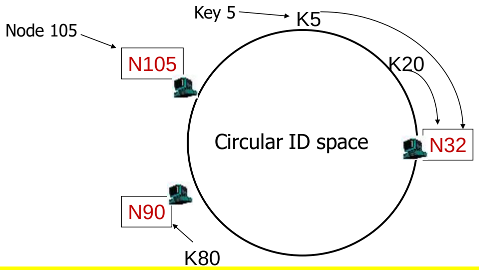
- ❖例如，从整数空间 [0…2^m-1] 中选值（m=160）
- ❖通常对 文件（the file） 或 IP地址（ IP address ）进行哈希取值

◆属性:

- ❖路由表的大小是 O(log *N*) , 这里*N*是节点数
- ❖保证一个文件在 O(log *N*) 跳内找到

DHT: Consistent Hashing

相容哈希（一致性哈希）：



一个 key 被存在它的后继（successor）节点
后继（successor）节点：下一个比较大的ID号的节点

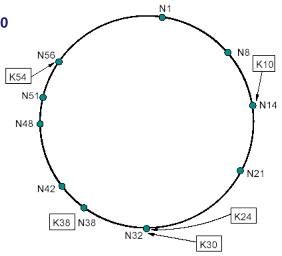
相容哈希的特点

- 平衡Balanced
  - 在不同节点上负载均衡
- 平滑Smoothness
  - 节点加入和退出对节点存储的内容影响小
- 分散Spread
  - 少量节点存储一个对象object
- 负载Load
  - 每个节点上存储的对象较少

Chord

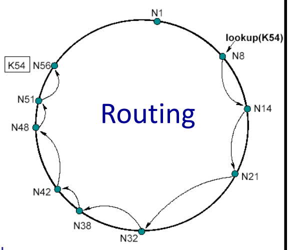
- **Chord 环: ID space mod 2^{160}**
  - ***nodeid = SHA1 (IP address, i)***

    for i=1..v virtual IDs
  - ***keyid = SHA1 (name)***
- **路由方法:**
  - **Each node knows successor后继 节点and predecessor 前趋节点on ring**
- **路由效率:**
  - **Each node knows O(log n) well-distributed neighbors**



Chord中的基本查询

```
lookup (id):
  if ( id > pred.id &&
       id <= my.id )
return my.id;
else
    return succ.lookup(id);
```

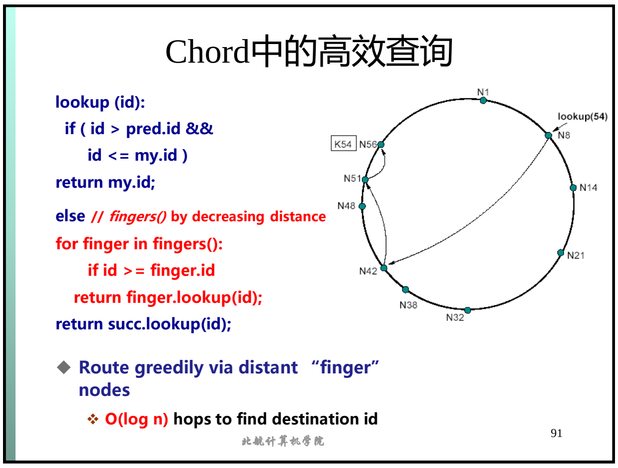


如何加速查找？

- 通过后继结点successors逐跳路由查询
  - O(n) hops to find destination id

查询加速

- 通过维护“路由信息”进行查询加速
- 每个节点维护一个“路由表”，具有(最多) *m* 个表项（N=2m)，也称为 指取表 <u>finger table</u>
  - **<u>start，successor(start)的地址</u>**
- *在节点n上的第 i个表项包括:*
  - *start* [i]= n+ 2^i (mod 2^m)
  - successor(start[i])

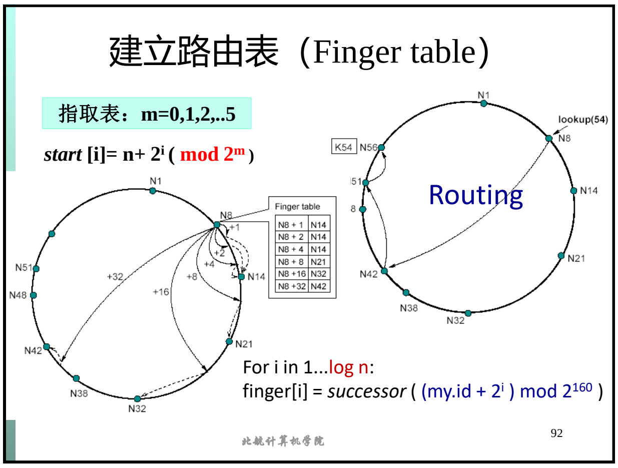


Chord中的高效查询

```
lookup (id):
  if ( id > pred.id &&
       id <= my.id )
return my.id;
else // fingers() by decreasing distance
for finger in fingers():
       if id >= finger.id
    return finger.lookup(id);
return succ.lookup(id);
```

N1
lookup(54)
N8
K54
N56
N51
N48
N14
N21
N42
N38
N32

- Route greedily via distant “finger” nodes
  - O(log n) hops to find destination id



建立路由表（Finger table）

指取表：m=0,1,2,..5

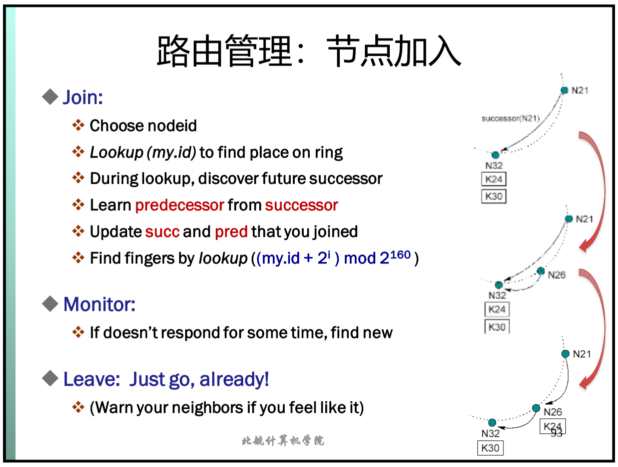
start [i]= n+ 2^i (mod 2^m)

N1
N8
+1
+2
+4
+8
+16
+32
N14
N21
N32
N38
N42
N48
N51
Finger table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42

N1
lookup(54)
N8
K54
N56
51
8
Routing
N14
N21
N42
N38
N32

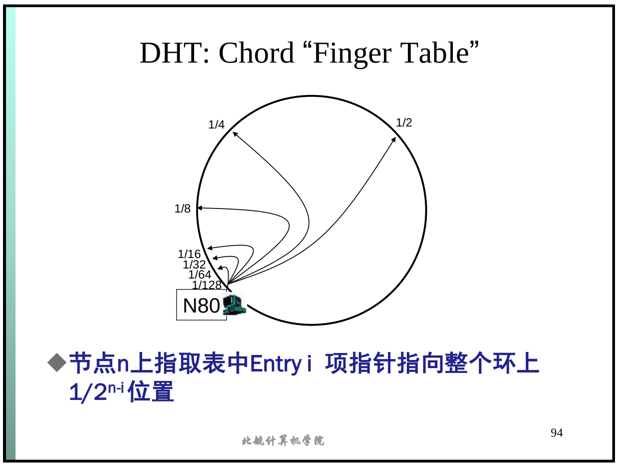
For i in 1...log n:
finger[i] = *successor* ((my.id + 2^i) mod 2^{160})



路由管理：节点加入

- Join:
  - Choose nodeid
  - *Lookup (my.id)* to find place on ring
  - During lookup, discover future successor
  - Learn predecessor from successor
  - Update succ and pred that you joined
  - Find fingers by *lookup* ((my.id + 2^i) mod 2^{160})
- Monitor:
  - If doesn’t respond for some time, find new
- Leave: Just go, already!
  - (Warn your neighbors if you feel like it)

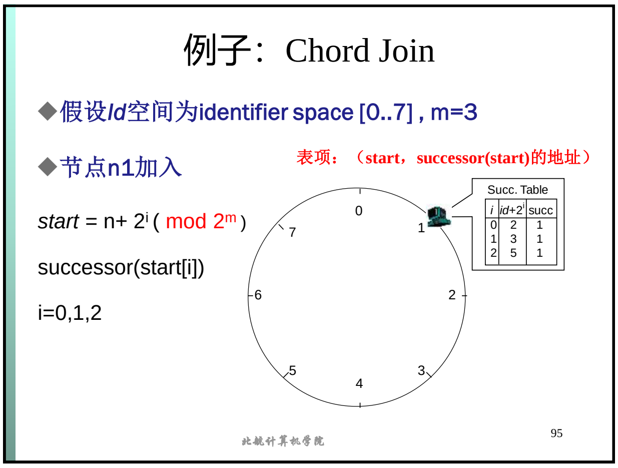
N21
successor(N21)
N32
K24
K30
N21
N26
N32
K24
K30
N21
N26
K24
N32
K30



DHT: Chord “Finger Table”

1/4
1/2
1/8
1/16
1/32
1/64
1/128
N80

- 节点n上指取表中Entry i 项指针指向整个环上$1/2^{n-i}$位置



例子：Chord Join

◆假设*Id*空间为identifier space [0..7] , m=3

◆节点n1加入

表项：（start，successor(start)的地址）

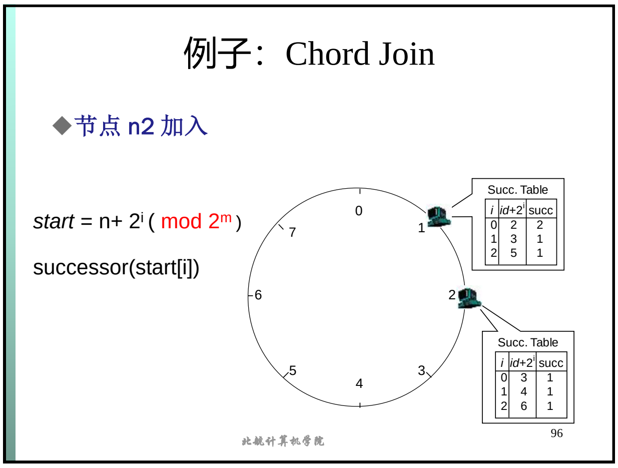
start = n+ 2^i (mod 2^m)

successor(start[i])

i=0,1,2

Succ. Table

i	*id*+2^i	succ
0	2	1
1	3	1
2	5	1



例子：Chord Join

◆节点 n2 加入

start = n+ 2^i (mod 2^m)

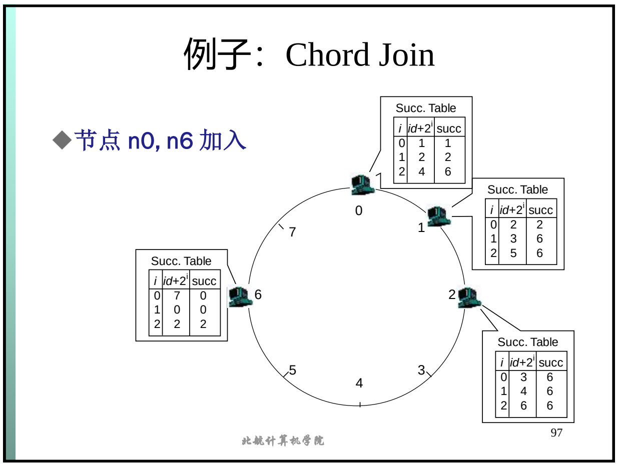
successor(start[i])

Succ. Table

i	*id*+2^i	succ
0	2	2
1	3	1
2	5	1

Succ. Table

i	*id*+2^i	succ
0	3	1
1	4	1
2	6	1



例子：Chord Join

◆节点 n0, n6 加入

Succ. Table

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	6

Succ. Table

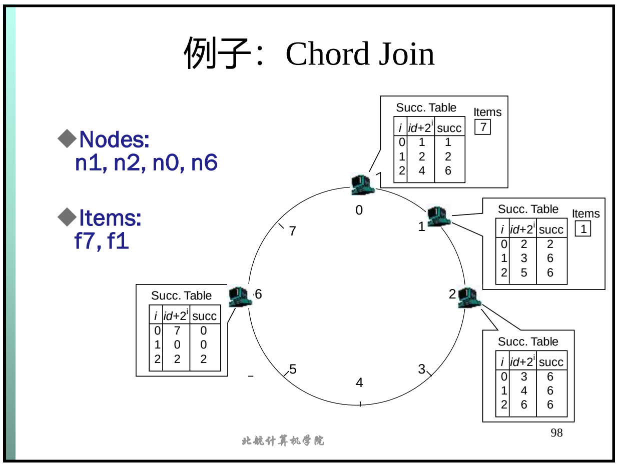
i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Succ. Table

i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

Succ. Table

i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6



例子：Chord Join

◆Nodes:
n1, n2, n0, n6

◆Items:
f7, f1

Succ. Table

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	6

Items
7

Succ. Table

i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Items
1

Succ. Table

i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

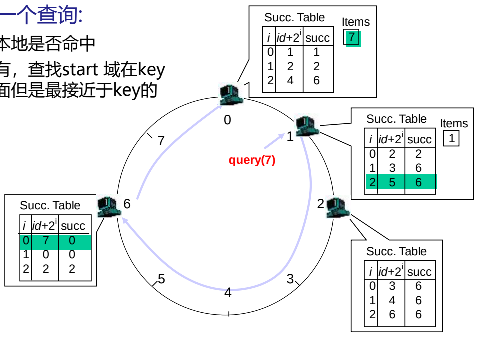
Succ. Table

i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

例子：Chord routing

◆接收到一个查询:

- ❖检查本地是否命中
- ❖若没有，查找start 域在key的前面但是最接近于key的节点



Node 0 — Succ. Table

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	6

Node 1 — Succ. Table

i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Node 2 — Succ. Table

i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Node 6 — Succ. Table

i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

Chord总结

◆路由：Associate to each node and item a unique *id* in an *uni*-dimensional space

◆指取表（Routing table）的大小?

- ❖O(log(N)) fingers，N为节点数

◆查找的时间?

- ❖每一跳的查找是的到达目标节点的距离减少 1/2 => 平均查找次数是O(log *N*)

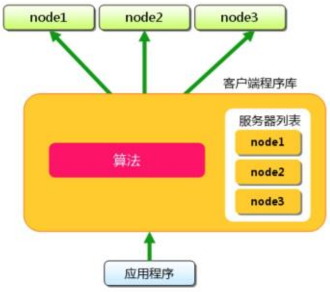
DHT应用：数据分布式存储

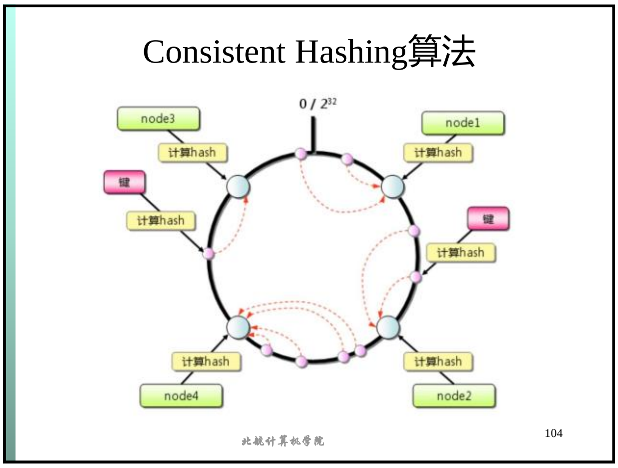
◆**在高并发环境下，数据库有大量的读、写请求，磁盘IO将成为瓶颈，从而导致过高的响应延迟。**

◆**分布式缓存机制**

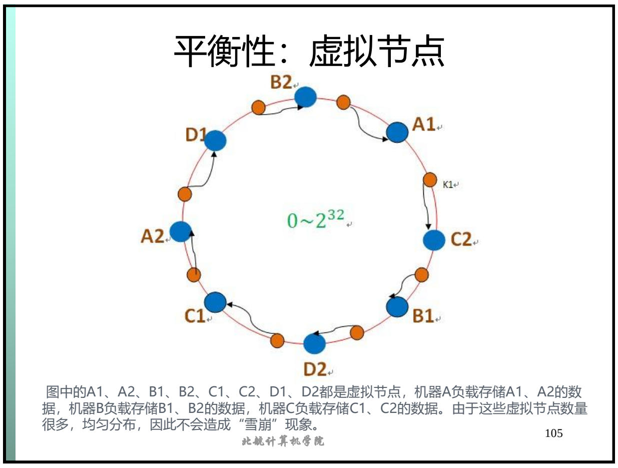
- ❖**Memcached，高性能的分布式内存缓存服务器(In-memory caching subsystem)，一般目的是为了通过缓存数据库的查询命中减少数据库压力、提高应用响应速度、提高可扩展性**
- ❖**Redis，REmote DIctionary Server，由Salvatore Sanfilippo开发的Key-Value存储系统**

memcached分布式原理





Consistent Hashing算法



平衡性：虚拟节点

图中的A1、A2、B1、B2、C1、C2、D1、D2都是虚拟节点，机器A负载存储A1、A2的数据，机器B负载存储B1、B2的数据，机器C负载存储C1、C2的数据。由于这些虚拟节点数量很多，均匀分布，因此不会造成“雪崩”现象。

完成小作业（5）

◆ 专题5“应用层网络与网络安全”

1.任意选择1篇论文进行阅读

2.每人独立完成论文评论（paper review），评论内容要求：

- ➢ 作者主要观点和要解决的问题
- ➢ 研究方法评论（关键技术，优点和局限性）
- ➢ 论文的主要贡献
- ➢ 其他
- ➢ 注意：不是翻译，篇幅不限

3.作业提交（两个文档）

- ➢ .docx文件
- ➢ .pptx文件（约 10 页左右，请勿超过15页，课堂讨论用）