

# 计算机网络

## 第2章 应用层

# 目 录

- 应用层协议原理
- WEB应用和HTTP协议
- 文件传输协议：FTP
- 因特网中的电子邮件
- DNS：因特网的目录服务
- P2P应用
- 视频流和内容分发网

## 2.1 应用层协议原理

### ■ 常见的网络应用

- 上网浏览新闻——IE、Maxthon、FireFox.....
- 处理电子邮件——Outlook Express、FoxMail、Outlook.....
- 和熟悉的或者陌生的朋友聊天——ICQ、QQ、MSN Messenger、UC.....
- 网络电话——SkyPe、QQ、Net2Phone.....
- 网络游戏对战——CS、魔兽世界、联众.....
- 资源共享——FTP、BT、电骡.....
- 在线视频——VOD、PPLive.....
- 搜索引擎——Google、百度、Bing.....

## 2.1 应用层协议原理

看了这么多成功的应用，可能你跃跃欲试，很想编写一个类似于Google这样的超级网络应用，期待自己有一天也能一步登天，迈入世界级的IT风云人物之列，甚至试图问鼎一下世界首富.....

*那么现在的你应该做些什么呢？*

## 2.1 应用层协议原理

### ■ 知道什么是网络应用程序

- 可以向网络发送数据
- 可以从网络接收数据
- 可以对数据进行处理
- 也许还能够
  - 将数据展现在界面上，以非常友好的方式让你知道它在做什么，免得你说它怠工
  - 时不时的弹出一个小窗口，提示你不要太辛勤工作了，以表示对你无微不至的关怀

.....

## 2.1 应用层协议原理

### ■ 决定你的网络应用所需采用的体系结构

- 客户机/服务器体系结构 (C/S)

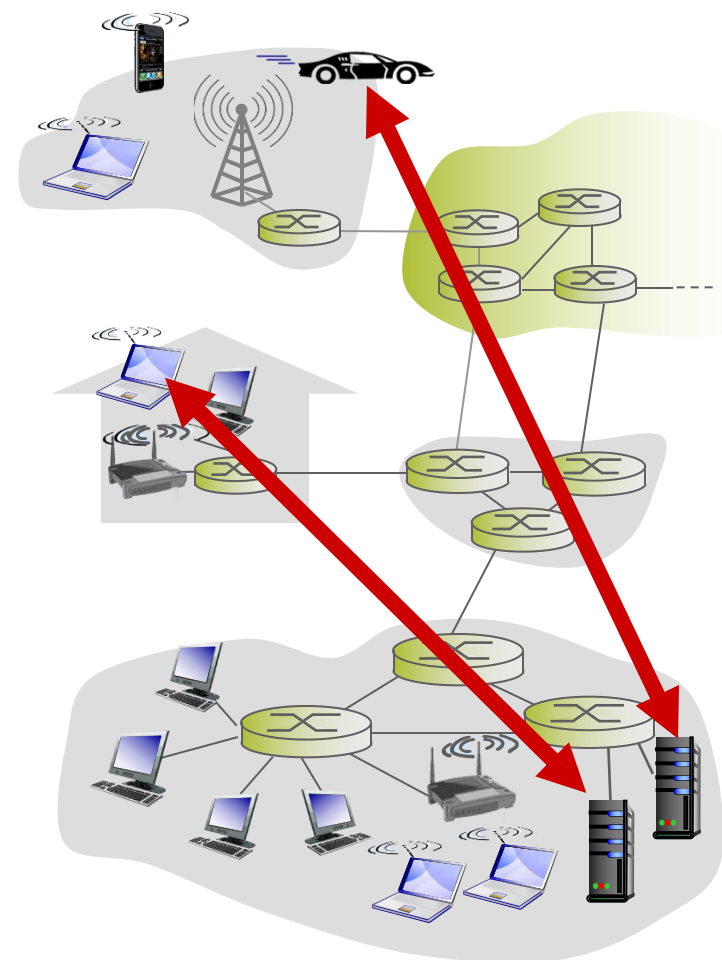
- P2P体系结构

- 混合体系结构

## 2.1 应用层协议原理

### ■ 客户机/服务器体系结构

- 存在一个能够向客户机提供服务的服务器，e.g., WEB服务器
- 存在一个或者多个主动连接服务器，试图从服务器那里获取所需服务的客户机，e.g., IE浏览器
- 特别注意1：客户机之间不能互相通信
- 特别注意2：为提高服务器的处理能力，通常采用服务器群集（Server Farm）



## 2.1 应用层协议原理

### ■ 客户机和服务器

- 不是指硬件服务器和客户主机，而是指软件进程

在给定的的一对进程之间的通信会话中，主动发起通信的进程被标识为**客户机**，被动等待联系的进程是**服务器**

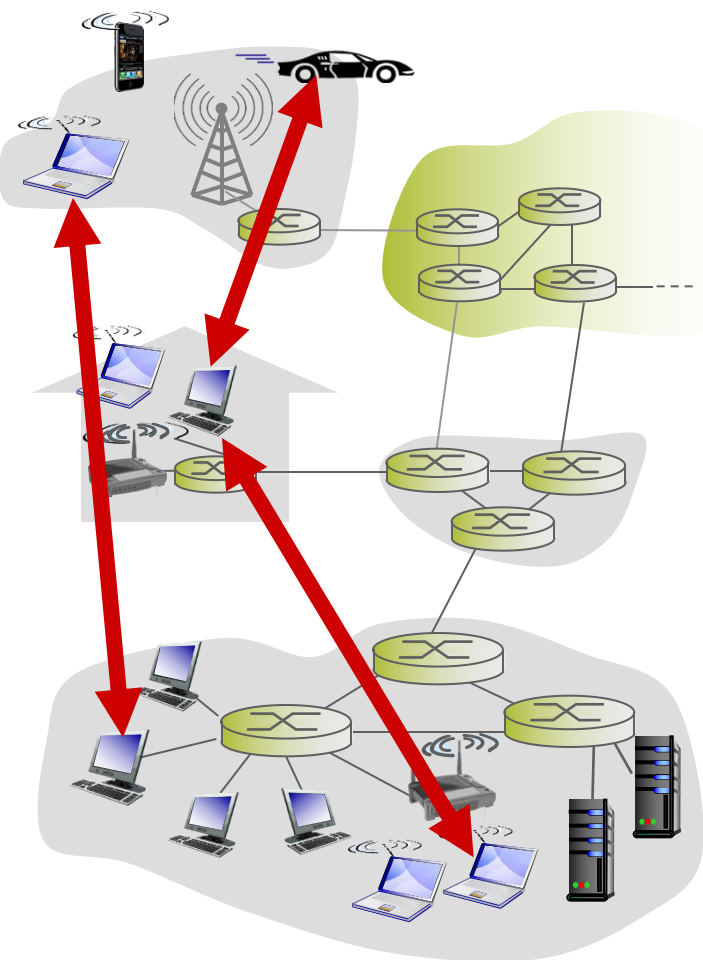
一个浏览器进程访问一个Web服务器，浏览器进程为客户机进程，Web服务器进程是服务器



## 2.1 应用层协议原理

### ■ P2P体系结构

- 任何一方既提供服务又享受服务
- 结点之间可以直接通信
- 结点的地址以及他们之间的连接可能随时发生变化
- 例如: 迅雷、PPLive
- 特别注意: P2P体系结构非常容易扩展, 但也特别难以管理



## 2.1 应用层协议原理

### ■ 混合体系结构

- 那混合体系结构自然而然就是C/S体系结构和P2P体系结构的混合体喽！
- 请大家回想一下第一个P2P应用Napster和即时通信（IM），一切就都明白了！

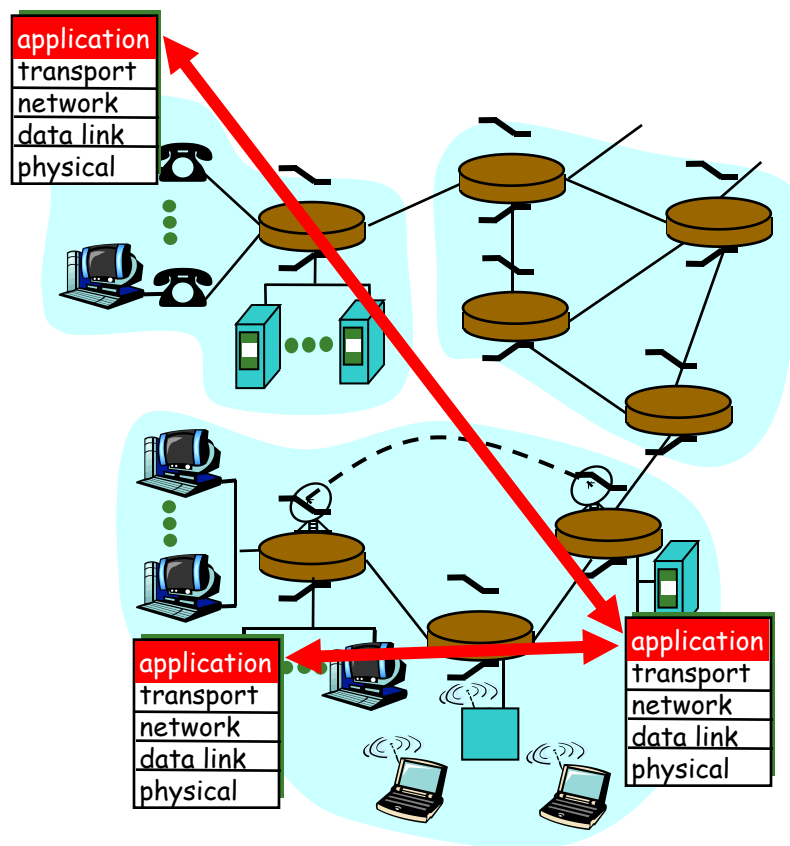
## 2.1 应用层协议原理

### ■ 网络应用会涉及到多个组成部分的交互

- 同一台主机上的进程之间通信的规则，由操作系统制定，和计算机网络无关，本课程就不讨论了。需要了解的，请回头看看《操作系统原理》及相关书籍
- 不同主机上的进程之间通信的规则，当然就和网络相关了，这套规则在计算机网络中，称之为“**应用层协议**”，也是本章重点讨论的内容

## 2.1 应用层协议原理

### ■ 网络进程间的通信涉及到的协议



网络应用程序：**相互通信的分布式的进程**

- 运行在网络主机（端系统）中的 “用户空间”
- 在应用程序间交换报文
- e. g., email, ftp, Web

应用层协议：

- 网络应用程序的一个“组成部分”
- 定义应用程序需交换的报文 和所需采取的动作
- 使用较低层次（运输层）所提供的通信服务（TCP, UDP）

## 2.1 应用层协议原理

- 应用层协议定义了
  - 交换的报文类型
  - 各种报文类型的语法
  - 字段的语义
  - 进程何时、如何发送报文及对报文进行响应

应用层协议  $\neq$  网络应用

## 2.1 应用层协议原理

- 因特网会给网络应用提供很多不同类型的服务，你的网络应用需要哪些服务呢？
  - 数据的可靠传输：你的网络应用是否需要？
  - 带宽的自动控制：你的网络应用是否带宽敏感？
  - 传输和反馈的实时性
  - 安全性

## 2.1 应用层协议原理

### ■ 常见应用程序对传输服务的要求

应用程序	数据丢失	带宽	实时性
文件传输	不丢失	弹性	无
e-mail	不丢失	弹性	无
Web 网页	不丢失	弹性	无
实时音频/视频	允许丢失	音频: 5Kb-1Mb	100's msec
	允许丢失	视频: 10Kb-5Mb	
存储音频/视频	允许丢失	同上	few secs
交互式游戏	允许丢失	几 Kb/s 以上	100's msec
金融应用	不丢失	弹性	yes and no

## 2.1 应用层协议原理

- 因特网运输层将所提供的服务整合成两类，你的网络应用使用哪一类传输服务，你该做出决定了！

### □ TCP

- **面向连接**: 在客户端和服务端进程之间需要建立连接
- **可靠传输**: 在发送和接收进程之间
- **流量控制**: 发送数据的速度决不超过接收的速度
- **拥塞控制**: 当网络超负荷时，束紧发送端口，减缓发送速度
- **不提供**: 实时性, 最小带宽承诺

### □ UDP

- 在客户端和服务端进程之间实现“不可靠的”数据传输
- **不提供**: 连接建立, 可靠性保证, 流量控制, 拥塞控制, 实时性, 最小带宽承诺



## 2.1 应用层协议原理

### □ UDP

- 在客户端和服务端进程之间实现“不可靠的”数据传输
- **不提供**: 连接建立, 可靠性保证, 流量控制, 拥塞控制, 实时性, 最小带宽承诺
- 为什么UDP还有用武之地?
  - 由于没有流量控制和拥塞控制, 因此发送速率不受限制, 特别适合于多媒体网络应用
  - 如果我又想用UDP但又需要可靠性保证怎么办?
  - 自己设计应用层协议, 在应用层提供数据的可靠性保证
  - 应用层协议HTTP、FTP、SMTP都没有可靠性保证, 靠TCP来保证

## 2.1 应用层协议原理

### ■ 因特网常见应用采用的传输协议

<u>应用</u>	<u>应用协议</u>	<u>所依赖的传输协议</u>
<b>e-mail</b>	smtp [RFC 821]	TCP
远程终端访问	telnet [RFC 854]	TCP
<b>Web</b>	http [RFC 2068]	TCP
文件传输	ftp [RFC 959]	TCP
流媒体	专有协议 (e.g. RealNetworks)	TCP or UDP
远程文件服务器	NFS	TCP or UDP
<b>IP电话</b>	专有协议 (e.g., Vocaltec)	typically UDP

## 2.1 应用层协议原理

### ■ 安全性

#### □ TCP/UDP天生不具备安全性

- 如果你敢把密码以明文送给TCP/UDP，TCP/UDP就敢把明文送给网络

#### □ 安全套接字层SSL

- 提供加密的TCP连接
- 数据的完整性检查
- 端点身份鉴别

#### □ SSL位于应用层与TCP之间（详见第7章）

## 2.1 应用层协议原理

- 当你的网络应用程序**Run**起来后，就变成了网络应用进程。可能还会产生如下问题：
  - 当你的网络应用和其它人开发的网络应用共同运行在一台主机上时，如何把不同的网络应用区分开来？
  - 网络核心只负责把数据交付到主机，并不负责把数据交付到应用，主机如何知道数据该交付到哪个网络应用？

## 2.1 应用层协议原理

### ■ 一个例子

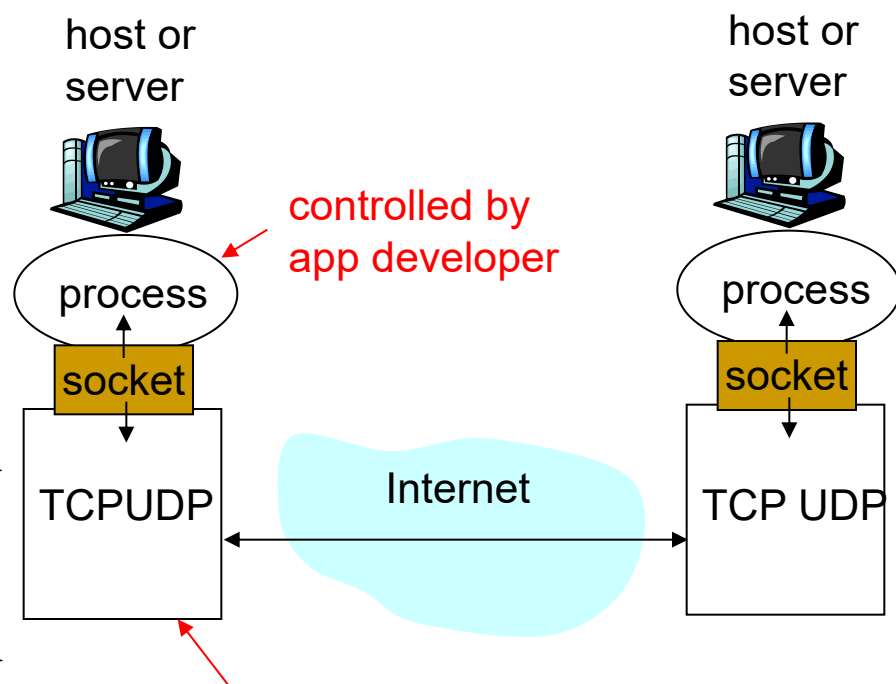
- 你们整栋宿舍有一个信箱，栋长每天都会查看一次信箱，取走新的信件和报纸，当你有信件需要寄送时，直接投递到邮局的邮筒里
- 假设
  - 邮递员仅把邮件送到每栋宿舍唯一的信箱，并不负责投递到个人
  - 栋长取出新的信件和报纸后，负责将信件和报纸投递到具体的房间
- 问题
  - 栋长如何区分哪封信件属于哪个房间呢？

## 2.1 应用层协议原理

- 类比到因特网，提供了类似的解决方法，那就是“**套接字 (Socket)**”
  - 每个网络应用进程都有一个属于自己的套接字，该套接字在整个因特网上独一无二
    - 主机地址：标识该网络应用进程运行在因特网上哪一台主机上，通常使用32位的IP地址进行标识
    - 端口地址：在该主机上标识该网络应用进程，通常使用16位的端口号进行标识
      - e.g., WEB Server: 80; Mail Server: 25;
  - 所以套接字的长度为48位

## 2.1 应用层协议原理

- 进程通过套接字来接收和发送报文
- 套接字相当于一个通道
  - 发送进程将报文交给套接字
  - 套接字将这些报文传输到接收进程的套接字



Socket API屏蔽了从运输层开始的所有底层网络协议细节。因此基于Socket API的网络应用程序只需要关注应用层协议。当利用Socket API创建套接字时，唯一需要指定的参数是：TCP/UDP，对方的IP，对方的Port。其他细节不用关心。

## 2.1 应用层协议原理

- 祝贺你！至此你已经获得了构造属于你自己的网络应用所需要的最基本最基本的知识！
- 但是这还远远不够，你还需要继续学习
  - 协议到底怎样工作
  - 传输层的服务是如何提供的
  - 套接字如何工作
  - IP地址是怎么回事
  - 网卡和网线起了什么样的作用
  - 如何保证网络应用的安全性和性能

.....



## 2.1 应用层协议原理

### ■ 本章重点讨论的网络应用

- WEB

- 文件传输

- 电子邮件

- 目录服务

- P2P

## 2.2 WEB应用和HTTP协议

### ■ 历史的回顾

- ❑ 19世纪70年代，电话的发明，扩展了人类通信的范围，增强了人类通信的实效性
- ❑ 20世纪20年代，广播收音机和电视的发明，极大的丰富了人类可获取信息
- ❑ 20世纪90年代，WEB的发明，极大的提高了人类主动获取信息的能力

### ■ 广播收音机/电视和WEB的异同点

- ❑ 都是广播和按需操作
- ❑ 你不能发布电视节目，但可以发布WEB内容

## 2.2 WEB应用和HTTP协议

### ■ WEB的构成

- WEB服务器：IIS、Apache、TomCat.....
- 浏览器：IE、Maxthon、Firefox
- 协议
  - 信息表达的协议——HTML
  - 信息传输的协议——HTTP

特别说明：WEB属于C/S模式

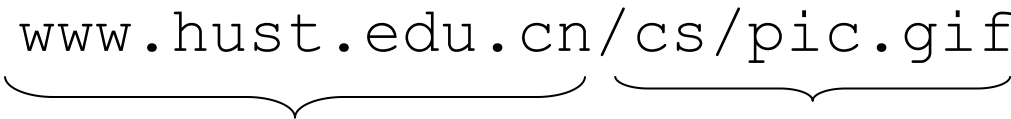
## 2.2 WEB应用和HTTP协议

### ■ WEB内容的表达

- ❑ Web 页面由一些对象组成。
- ❑ 对象可以是HTML文件、JPEG图片、音频文件、Java Applet.....
- ❑ HTML文件是Web页面的基础，它可以包括各种各样的对象，是一个容器对象
- ❑ 任何一个对象都可以用 URL来定位
- ❑ URL的例子: 
$$\underbrace{\text{www.hust.edu.cn}}_{\text{主机名}} / \underbrace{\text{cs/pic.gif}}_{\text{路径名}}$$

## 2.2 WEB应用和HTTP协议

### ■ Web服务器的虚拟路径

- 首先一个网站所有的网页及所需的其他资源如图片，Flash等都按一定的目录结构存放在Web服务器上。
- 任何一个资源对象都用 URL 来定位
- URL的例子：  
  
`www.hust.edu.cn/cs/pic.gif`
- 在这个URL里，路径/cs/pic.gif表示Web服务器  
www.hust.edu.cn根目录下的cs子目录下的pic.gif文件
- 这是一个虚拟的路径，因为我们并不知道www.hust.edu.cn  
根目录(或者叫网站根目录)到底对应到服务器上的哪个物理路径（即真正的物理目录）

## 2.2 WEB应用和HTTP协议

### ■ 虚拟路径的好处

- 首先更安全：隐藏了一个资源对象在Web服务器上的真实的物理位置
- Web服务器管理更方便：只要网站相对目录结构不变，可以随时根据需要改变网站在服务器上的物理路径，只需要重新做个虚拟路径映射，将该虚拟路径重新指向到新的物理文件夹即可，对客户端没有任何影响。
- 屏蔽了不同操作系统对路径表示的差异性：
  - 如Windows : C:\Inetpub\wwwroot
  - Linux: /home/wwwroot

## 2.2 WEB应用和HTTP协议

### ■ 虚拟路径到物理路径的映射

- ❑ 所有的Web服务器，应该提供虚拟路径到物理路径之间的映射管理
- ❑ 假设 网站的根目录在C:\Inetpub\wwwroot\hust下，而该路径被映射到虚拟路径/（二者之间的映射关系由Web服务器负责维护，并提供界面让用户设置）
- ❑ 当用户访问http://www.hust.edu.cn/cs/pic.gif文件时，Web服务器会到目录C:\Inetpub\wwwroot\hust\cs\pic.gif里读出该文件的内容并发送到客户端

## 2.2 WEB应用和HTTP协议

### ■ 虚拟路径到物理路径的映射

- ❑ MS 的Web服务器IIS提供了图形化的界面让用户配置虚拟路径（注意这里的用户是指网站管理员，而不是一般的网站访问者）
- ❑ Apache是通过配置文件进行虚拟路径的管理。
- ❑ 对Apache for Windows，在conf目录下有个配置文件httpd.conf，里面包含了Web服务器的所有配置信息，其中包括了虚拟路径的映射



## 2.2 WEB应用和HTTP协议

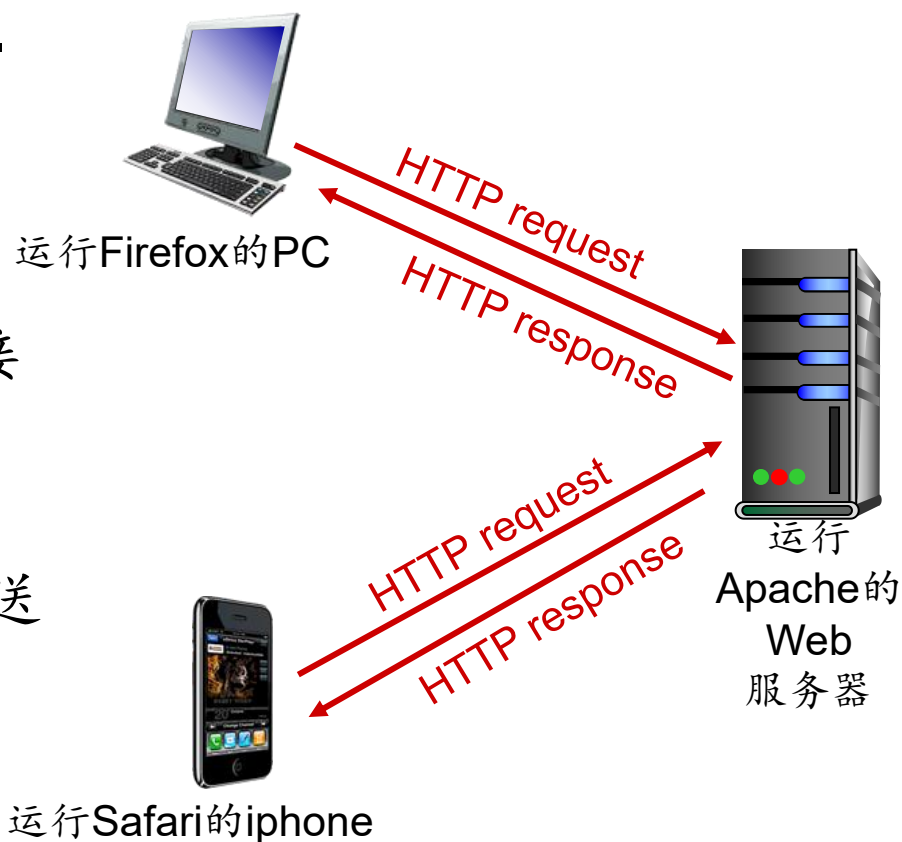
### ■ WEB内容的传输——HTTP协议

#### □ 客户端/服务器模式

- 客户端: 浏览器请求、接收、展示 Web对象 (objects)
- 服务器: Web 服务器发送对象对请求进行响应

□ http1.0: RFC 1945

□ http1.1: RFC 2068



## 2.2 WEB应用和HTTP协议

### http:使用 TCP 传输服务:

- ❑ 客户端启动TCP连接(创建套接字) 到服务器, 端口 80
- ❑ 服务器接受来自客户端的 TCP 连接
- ❑ http 报文(应用层协议报文) 在浏览器 (http client) 和 Web服务器(http server)之间进行交换, 被封装在TCP 报文段里传输
- ❑ 关闭TCP 连接

## 2.2 WEB应用和HTTP协议

http 是 “无状态 (stateless)” 的

- 服务器不保留任何访问过的请求信息

—— 小评论 ——

保留状态的协议很复杂！

- 过去的历史 (状态) 需保留
- 一旦浏览器/服务器崩溃，它们各自的状态视图就会发生分歧，还需要重新进行核对。

## 2.2 WEB应用和HTTP协议

### ■ HTTP1.0的传输模式——非持久性连接

假设用户键入了一个 URL `www.hust.edu.cn/cs/home.index`

(该网页包含文本并引用了10 jpeg 图片)

1a. http 客户端启动 TCP 连接

到`www.hust.edu.cn`上的  
http 服务器 (进程). Port 80  
是 http 服务器的默认端口.

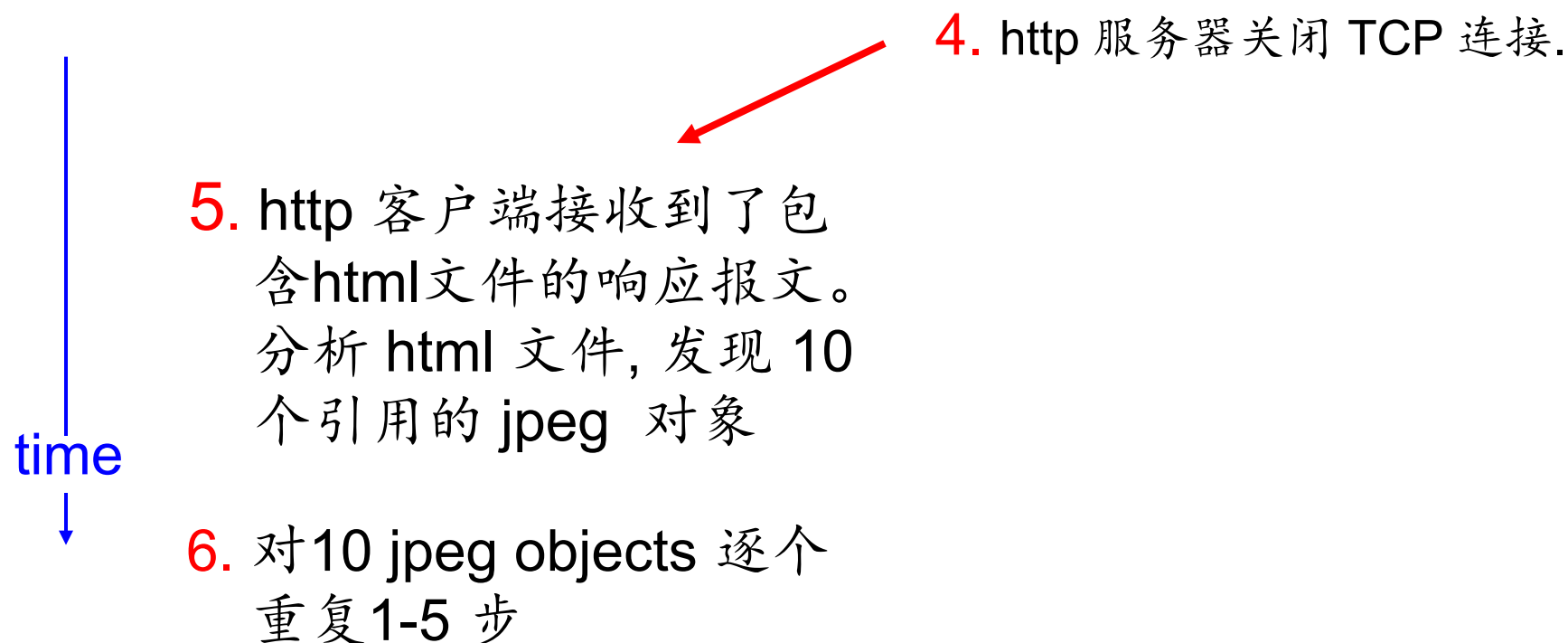
1b. 在`www.hust.edu.cn` 上的  
http 服务器在 port 80 等待  
TCP 的连接请求. “接受”  
连接并通知客户端

2. http客户端发送 http 请  
求报文 (包括URL) 进入  
TCP 连接插口 (socket)

3. http 服务器接收到请求报文,  
形成 响应报文 (包含了所请  
求的对象, `cs/home.index`),  
将报文送入插口 ( socket)

time

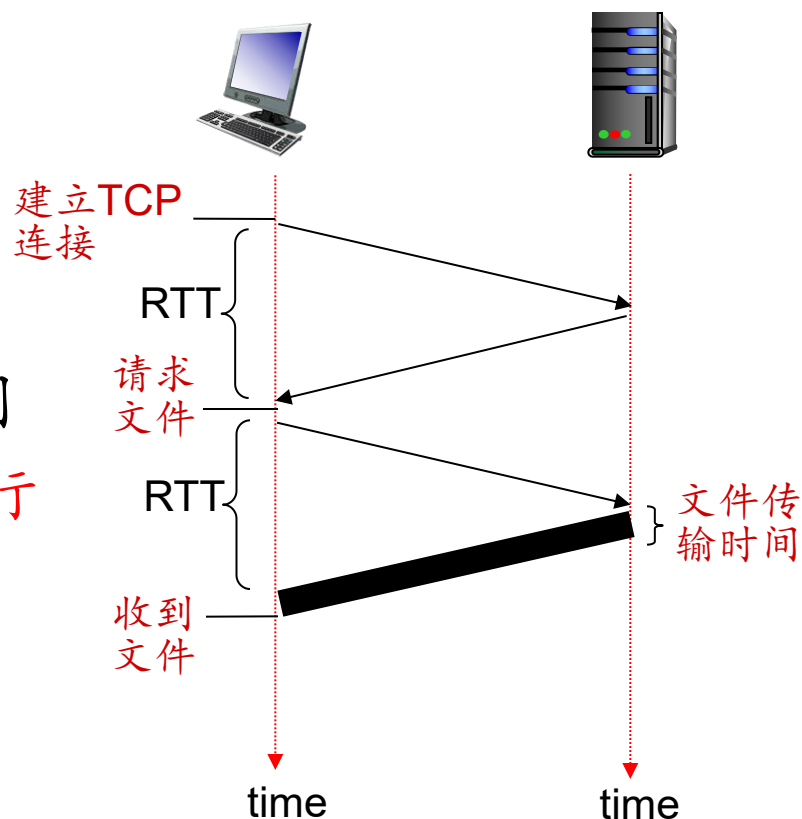
## 2.2 WEB应用和HTTP协议



## 2.2 WEB应用和HTTP协议

### ■ 非持久性连接工作机制

- 取一个对象需要2 RTTs
  - TCP 连接
  - 对象请求/传送
- 总时间=2RTT+文件传输时间
- 许多浏览器同时打开多个**并行**的连接来改善性能



如果忽略文件传输时间，11个对象一共需要22RTT

## 2.2 WEB应用和HTTP协议

### ■ HTTP1.1引入的新传输模式——持久连接

- 服务器在发送响应后，不再断开TCP连接，而是保持该连接，用于后续对象的传送，直至该连接“**休息**”了一个较长的时间后，方断开该连接
- 减少了对服务器端连接数的需要，从而减少了对服务器端套接字资源的占用，提高了服务器的负载能力
- 持久连接又可以分为
  - 非流水线方式：一个对象传输完成方能请求传输下一个
  - 流水线方式：可以一次性发送所有请求，慢慢接收

## 2.2 WEB应用和HTTP协议

### ■ HTTP报文类型

- HTTP请求报文：客户端->服务器
- HTTP响应报文：服务器->客户端



## 2.2 WEB应用和HTTP协议

### ■ HTTP请求报文

#### □ 一段典型的HTTP请求报文（ASCII）

请求行  
(GET, POST,  
HEAD 命令)

首部  
诸行

单独一行回车、换行  
表示报文首部结束

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

回车符  
换行符

应用层协议的报文全部是基于文本  
因此网络应用程序能非常方便地处理

## 2.2 WEB应用和HTTP协议

### □ HTTP请求报文的一般格式



## 2.2 WEB应用和HTTP协议

### ■ 请求行支持的方法

#### □ HTTP1.0 定义的方法

##### ■ GET

- 向服务器请求指定URL的对象

##### ■ POST

- 用于向服务器提交表单数据
- 也可以同时请求一个WEB页面
- 特别注意：可以不使用POST方法，而使用GET方法发送表单数据。

##### ■ HEAD

- 请求服务器返回一个响应报文，但是该报文中并不包含请求的对象。该方法常常用来进行故障跟踪。



## 2.2 WEB应用和HTTP协议

### □ HTTP1.1新定义的方法

#### ■ PUT

- 上传的文件放在实体主体字段中，目标路径由URL字段标明

#### ■ DELETE

- 删除URL字段中指定的文件

### □ 另一种上传表单数据的方式

#### ■ 使用GET方法

#### ■ 将需要上传的表单数据放到URL中

`www.somesite.com/animalsearch?username=aaa&password=123`

# 表单数据提交到服务器的方法

## ■ get方法上传表单数据

- 浏览器把表单中的各个数据值添加到URL后。URL前缀和表单数据以?分隔，表单数据的形式为”变量名=值”，数据之间用&分割。如：

`http://www.aa.com/login.html?username=abc&password=123`

### □ 缺点

- 1: URL长度有限，如IE地址栏URL长度是2083个字符
- 2: URL是明文，不安全。特别是需要提交密码时
- 3: 只能传送字符型的数据

# 表单数据提交到服务器的方法

- post 方法
- 参数会被打包在HTTP请求报文的主体部分中，按照变量和值相对应的形式发送到服务器端
- 优点
  - 1：由于是打包在HTTP报文里，故适合大规模数据传送，如上传文件（包括二进制文件，如图片等）
  - 2：安全性高。因为post方式提交数据时是将表单中的字段与值放置在HTTP请求报文的主体部分中，用户是看不见的。
  - 3：可以传送文本数据和二进制数据

## 2.2 WEB应用和HTTP协议

### ■ HTTP响应报文

状态行

(协议 状态码

状态短语)

HTTP/1.1 200 OK\r\n

Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n

Server: Apache/2.0.52 (CentOS)\r\n

Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n

Content-Length: 2652\r\n

Keep-Alive: timeout=10, max=100\r\n

Connection: Keep-Alive\r\n

Content-Type: text/html; charset=ISO-8859-1\r\n

\r\n

data data data data data ...

首部  
诸行

数据, e.g.,

被请求的html文件

## 2.2 WEB应用和HTTP协议

### □ HTTP响应报文的一般格式





## 2.2 WEB应用和HTTP协议

### ■ 常见的HTTP响应状态码和短语

#### **200 OK**

- 请求成功, 被请求的对象在报文中

#### **301 Moved Permanently**

- 被请求的对象被移动过, 新的位置在报文中说明 (Location:)

#### **400 Bad Request**

- 服务器不懂请求报文

#### **404 Not Found**

- 服务器上找不到请求的对象

#### **505 HTTP Version Not Supported**

- 服务器不支持请求报文使用的HTTP协议版本

## 2.2 WEB应用和HTTP协议

### ■ 了解HTTP报文格式的最好方法就是自行测试

#### 1. 用Telnet 连接测试用的服务器:

```
telnet cis.poly.edu 80
```

打开 TCP 连接到 port 80  
(默认的http 服务器端口) 位于cis.poly.edu后续  
键入的内容将发送到cis.poly.edu的80 号端口

#### 2. 键入一条 http请求报文:

```
GET /~ross/ HTTP/1.1  
Host: cis.poly.edu
```

将该指令键入后 (按两次回车键), 就将此最短  
之 (但是完整的) GET 请求发到了 http 服务器

#### 3. 请注意观察http服务器发回的响应报文!

(或者使用Wireshark观察捕获的HTTP请求/响应报文)

# 2.2 WEB应用和HTTP协议

F12

DOM 资源管理器

控制台

调试程序

网络

性能

内存

仿真

www.baidu.com - F12 开发人员工具

11

?

内容类型

查找(Ctrl+F)

名称 / 路径	协议	方法	结果 / 描述	内容类型	已接收	时间	发起程序 / 类型	0毫秒
a.js?tu=u3514105&op=1&jk=ea0d20dd8bc980c3&... https://edclick.baidu.com/	HTTPS	GET	200 OK	application/x-ja...	0 B	366.53 毫秒	image	
a.js?tu=u4339860&op=1&jk=3e5c9acc1c8a26c8&... https://edclick.baidu.com/	HTTPS	GET	200 OK	application/x-ja...	0 B	231.56 毫秒	image	
a.js?tu=u3432571&op=1&jk=d3f9e0057416d331&... https://edclick.baidu.com/	HTTPS	GET	200 OK	application/x-ja...	0 B	231.21 毫秒	image	
https://www.baidu.com/	HTTPS	GET	200 OK	text/html		1.82 秒	document	
dong1_a1c52951c1f40e1496b46b9ae415c121.gif https://www.baidu.com/img/	HTTPS	GET	200 OK	image/gif	488.21 KB	2.27 秒	www.baidu.com:442 parsedElement	
baidu_jgylogo3.gif https://www.baidu.com/img/	HTTPS	GET	200 OK	image/gif	705 B	1.23 秒	www.baidu.com:442 parsedElement	
baidu_resultlogo@2.png https://www.baidu.com/img/	HTTPS	GET	200 OK	image/png	6.36 KB	368.43 毫秒	www.baidu.com:442 parsedElement	
jquery-1.10.2.min_65682a2.js https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	application/java...	32.39 KB	583.4 毫秒	www.baidu.com:658 parsedElement	
icons_441e82f.png https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	image/png	17.13 KB	429.25 毫秒	image	
zbios_09b6296.png https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	image/png	12.98 KB	417.39 毫秒	image	
all_async_search_67a0787.js https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	application/java...		186.93 毫秒	script	
every_cookie_4644b13.js https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	application/java...	1.31 KB	229.98 毫秒	script	
nu_instant_search_068a951.js https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	application/java...	5.52 KB	57.57 毫秒	script	
swfobject_0178953.js https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	application/java...	3.74 KB	48.92 毫秒	script	
tu_77547af.js https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	application/java...	5.47 KB	52.06 毫秒	script	
search-sug_73a0f48.js https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	application/java...	83 B	27.72 毫秒	script	
quickdelete_33e3eb8.png https://ss1.bdstatic.com/5eN1bjq8AAUyM2zgoY3K/r/www...	HTTP/2	GET	200	image/png	1.07 KB	118.23 毫秒	image	
w.gif?q=inlo&fm=inlo&rsv_psid_page=1&rsv_psid0... https://sp0.baidu.com/5bU_dTmfKgQfm2e88luM_a/	HTTPS	GET	200 OK	image/gif	0 B	416.97 毫秒	image	

0 个错误

22 个请求

588.62 KB 已传输

已耗时 9.55 秒 (DOMContentLoaded: 2.42 秒, 加载: 3.21 秒)

标头

正文

参数

Cookie

计时

请求 URL: https://www.baidu.com/

请求方法: GET

状态码: 200 / OK

请求标头

Accept: text/html, application/xhtml+xml, image/jxr, \*/\*

Accept-Encoding: gzip, deflate

Accept-Language: zh-Hans-CN, zh-Hans; q=0.8, en-US; q=0.5, en; q=0.3

Connection: Keep-Alive

Cookie: BAIDUID=D476A5E545F71EACD07A6B3268A8F728:FG=1

Host: www.baidu.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Ge...

响应标头

Bdpagetype: 1

Bdqid: 0xd6db1c40000af11f

Cache-Control: private

Connection: Keep-Alive

Content-Encoding: gzip

Content-Type: text/html

Cxy\_all: baidu+01ba90de3e95c4d79dc8540cece0b7f1

Date: Tue, 10 Sep 2019 15:44:08 GMT

Expires: Tue, 10 Sep 2019 15:43:29 GMT

Server: BWS/1.1

Set-Cookie: BD\_HOME=0; path=/

Set-Cookie: BDSVRTM=0; path=/

Set-Cookie: BIDUPSID=D476A5E545F71EACD07A6B3268A8F728; expires=Thu, 3...

Set-Cookie: delPer=0; path=/; domain=.baidu.com

Set-Cookie: H\_PS\_PSID=29654\_1444\_21119\_29522\_29521\_29721\_29567\_29221...

Set-Cookie: PSTM=1568130248; expires=Thu, 31-Dec-37 23:55:55 GMT; max-ag...

在这里输入你要搜索的内容

23:46

2019/9/10

## 2.2 WEB应用和HTTP协议

### ■ 用户-服务器交互：Cookie

#### □ WEB站点使用Cookie的目的

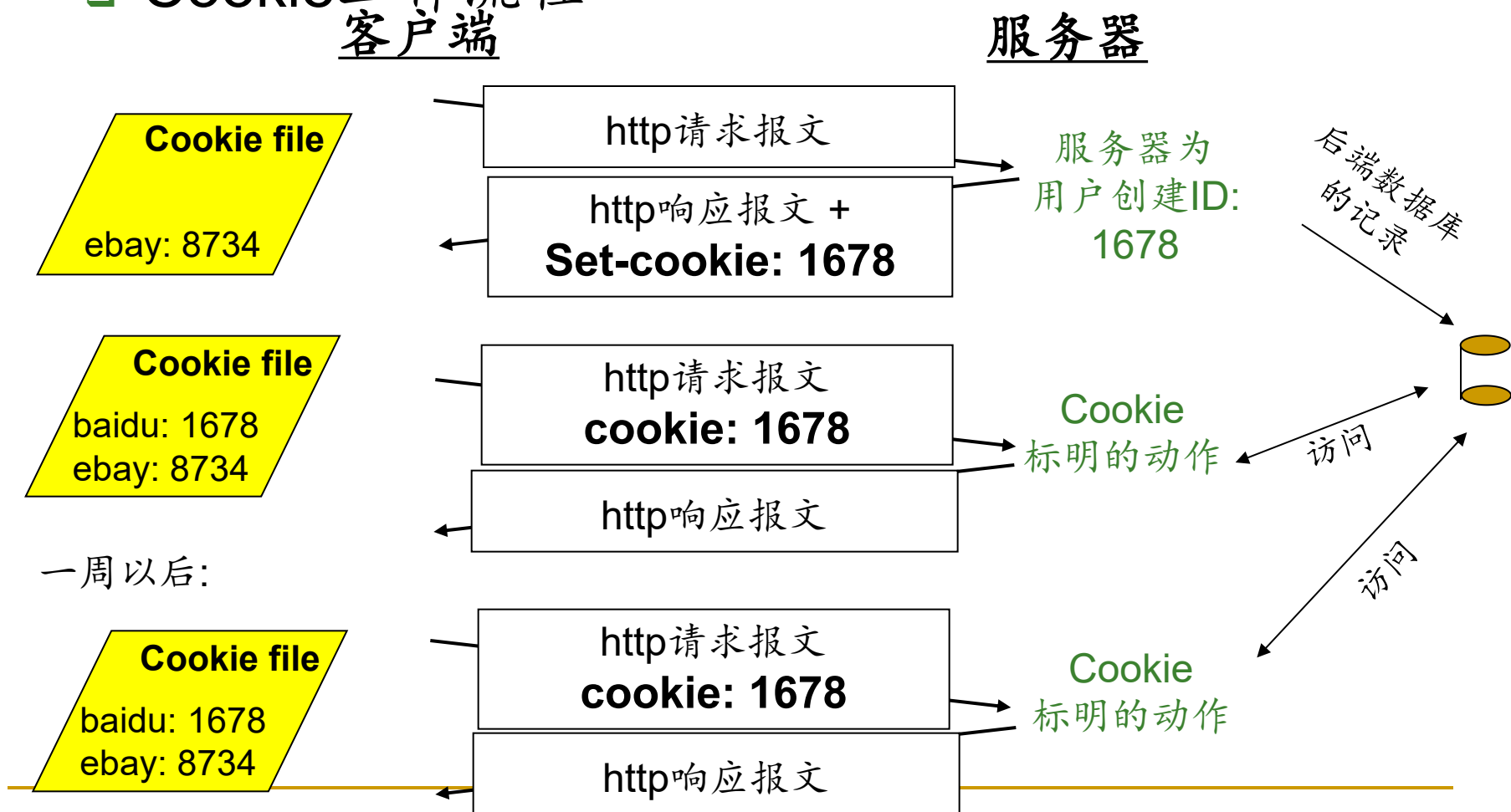
- 限制用户的访问
- 把内容和用户身份关联起来

#### □ Cookie技术的组成部分:

- 在HTTP响应报文中有一个Cookie首部行
- 在HTTP请求报文中也有一个Cookie首部行
- 在用户的端系统中保留了一个Cookie文件，由用户浏览器负责管理
- 在Web站点有一个后端数据库

## 2.2 WEB应用和HTTP协议

### Cookie 工作流程



## 2.2 WEB应用和HTTP协议

□ Cookies能为我们带来什么好处呢？

- 认证
- “购物车”
- “推荐”
- 用户会话状态 (Web e-mail)

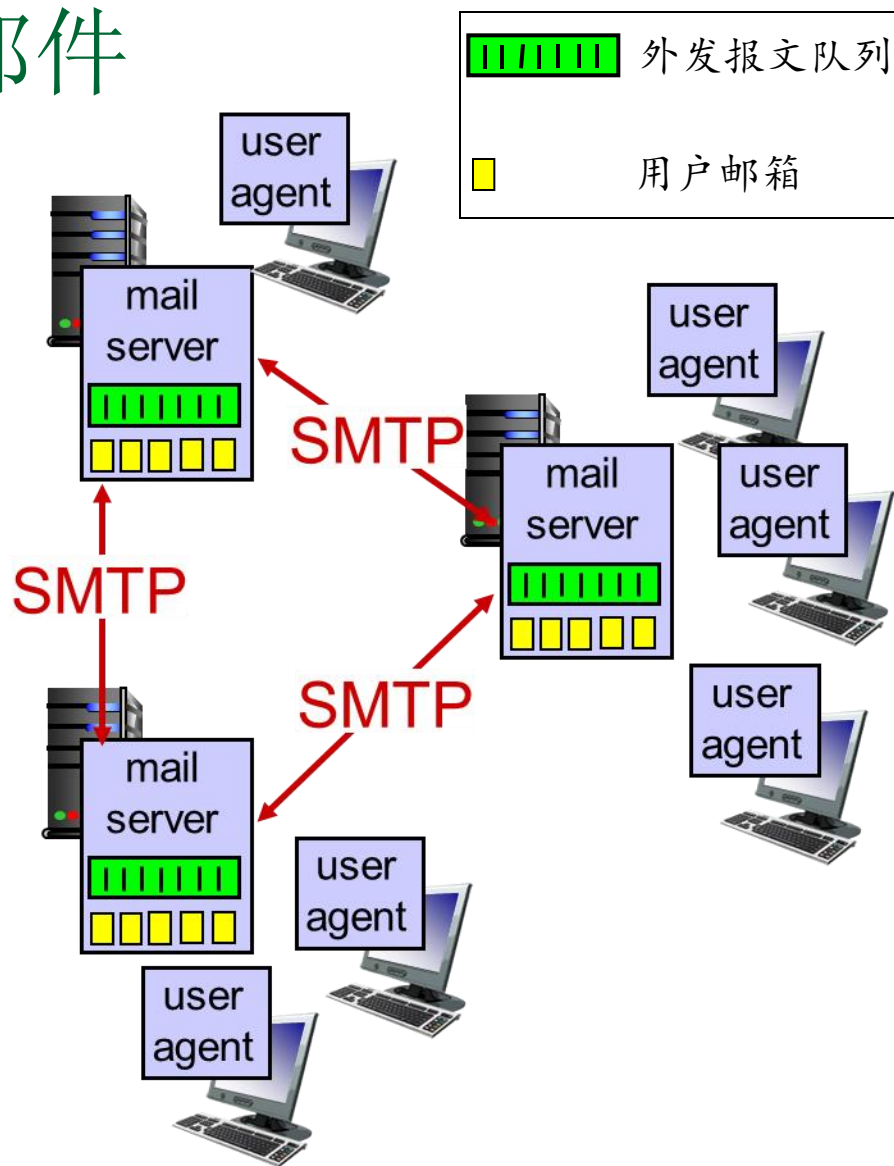
### Cookies和私密性:

- Cookies允许网站获得相当多的用户的信息
- 你可能会向网站提供你的姓名和E-Mail地址
- 搜索引擎也可以使用cookie和重定向技术获得很多的信息
- 广告公司也可以通过用户访问过的网站来获得用户的相关信息

## 2.4 因特网中的电子邮件

### ■ 电子邮件系统的构成

- 用户代理
- 邮件服务器
- 简单邮件传输协议: SMTP





## 2.4 因特网中的电子邮件

### ■ 用户代理

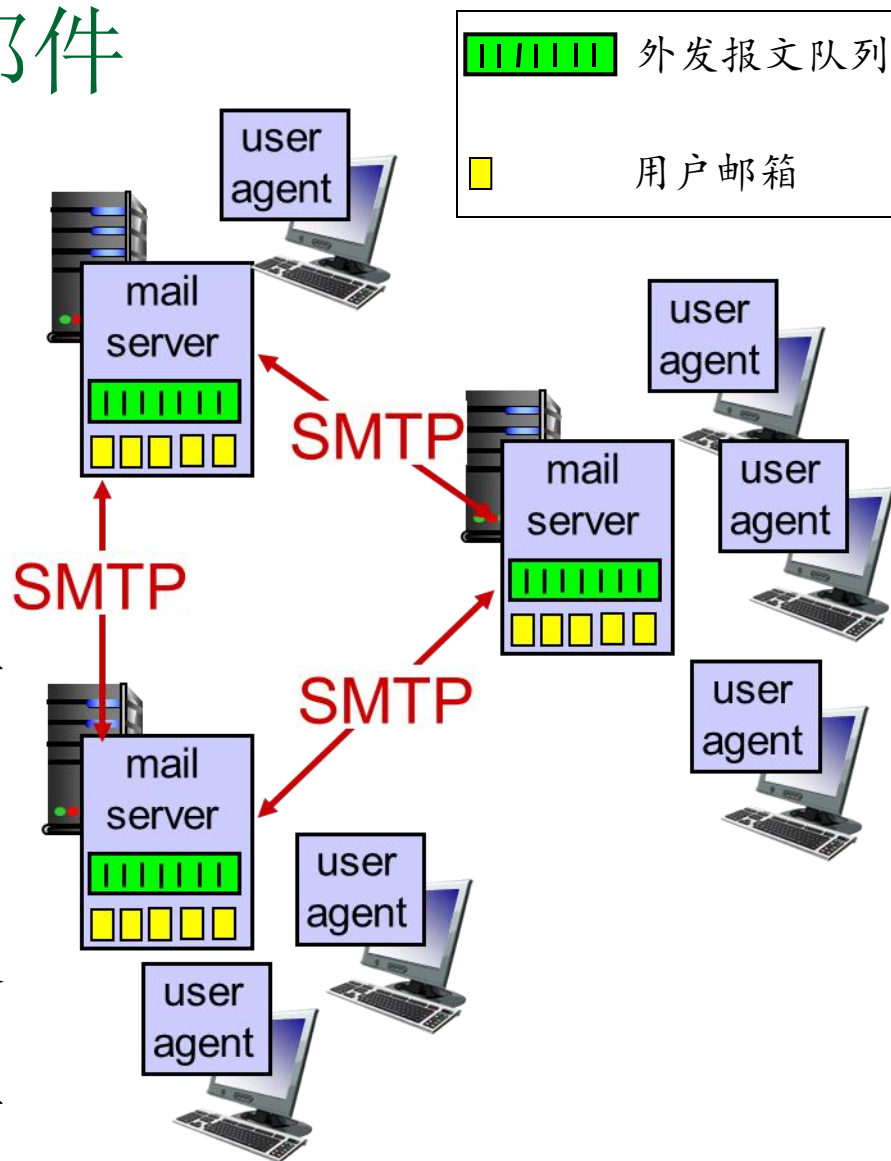
- 写作, 编辑, 阅读邮件报文
- e.g. OE、FoxMail

### ■ 邮件服务器

- **邮箱** 包含了收到的用户邮件 (尚未被阅读)
- **报文** 队列包含了外发的邮件报文

### ■ **SMTP 协议** 用在邮件服务器之间发送邮件

- 客户端: 将邮件发送到邮件服务器
- “服务器”: 接收和转发邮件





## 2.4 因特网中的电子邮件

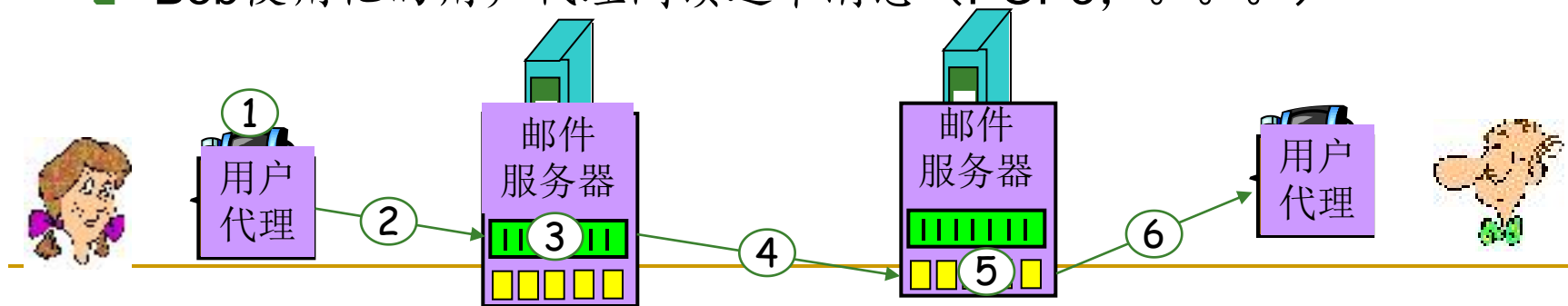
### ■ SMTP协议

- 使用 TCP可靠的传送邮件报文, 端口25
- 直接传输: 发送服务器到接收服务器
- 传输的三个阶段
  - 握手(打招呼)
  - 报文传输
  - 结束
- 命令/响应交互
  - 命令: ASCII文本
  - 响应: 状态码和短语
- 邮件报文必须使用7-bit ASCII表示

## 2.4 因特网中的电子邮件

### ■ 一次邮件传送过程

- Alice使用用户代理发送消息给：bob@someschool.edu
- Alice 的用户代理发送消息给她的邮件服务器；消息被保存在消息队列中
- SMTP的客户端（Alice的邮件服务器）向Bob的邮件服务器建立一个TCP连接
- SMTP的客户端（Alice的邮件服务器）通过这个TCP连接发送Alice的消息到Bob的邮件服务器
- Bob的邮件服务器将这个消息存储到Bob的邮箱中
- Bob使用他的用户代理阅读这个消息（POP3，。。。）



## 2.4 因特网中的电子邮件

telnet mailserver 25

```
S: 220 hamburger.edu ----- (mailserver)
C: HELO crepes.fr ----- (client hostname)
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

## 自测 smtp 交互:

- `$telnet MailServerName 25`
- 见到邮件服务器的 220 响应后
- 键入 HELO, MAIL FROM, RCPT TO, DATA, QUIT 命令

上述过程可以不使用用户代理，就能直接将电子邮件发送出去（因为目前大部分邮件服务器的交互过程趋于复杂，本试验不一定都能进行）。

## 2.4 因特网中的电子邮件

### ■ SMTP评述

- SMTP使用持续连接
- SMTP要求报文(首部 & 信体)全部使用 7-bit ASCII码
- 某些代码组合不允许出现在报文中 (e.g., CRLF.CRLF). 此类数据必须进行编码 (通常使用 base-64 或 quoted printable)
- SMTP服务器用 CRLF.CRLF 表示邮件报文的结束

### ■ SMTP vs HTTP

- 都使用 ASCII 命令/响应来交互并使用状态码
- SMTP要求报文 全部使用 7-bit ASCII码, 而HTTP没有这个限制
- HTTP: pull (拉) vs SMTP: push (推)
  - HTTP: 文件的接受者发起连接
  - SMTP: 文件的发送者发起连接
- HTTP的每个对象分装在各自的响应报文中, 而SMTP的多个对象在~~一个~~“多分部”的报文中传送

## 2.4 因特网中的电子邮件

### ■ 邮件报文格式 (RFC 822)

□ 首部诸行, e.g.,

■ To:

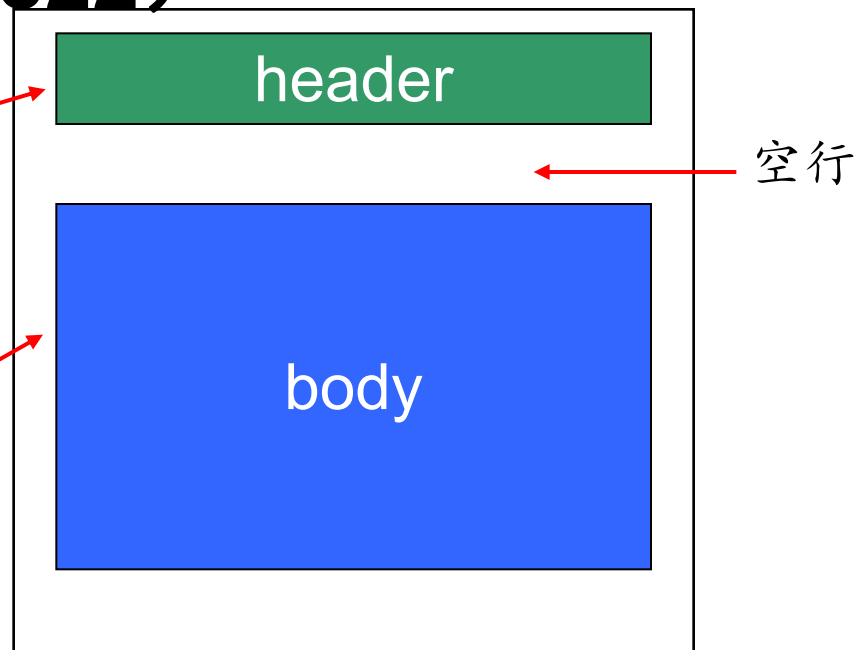
■ From:

■ Subject:

*不同* 于 *smtp* 命令!

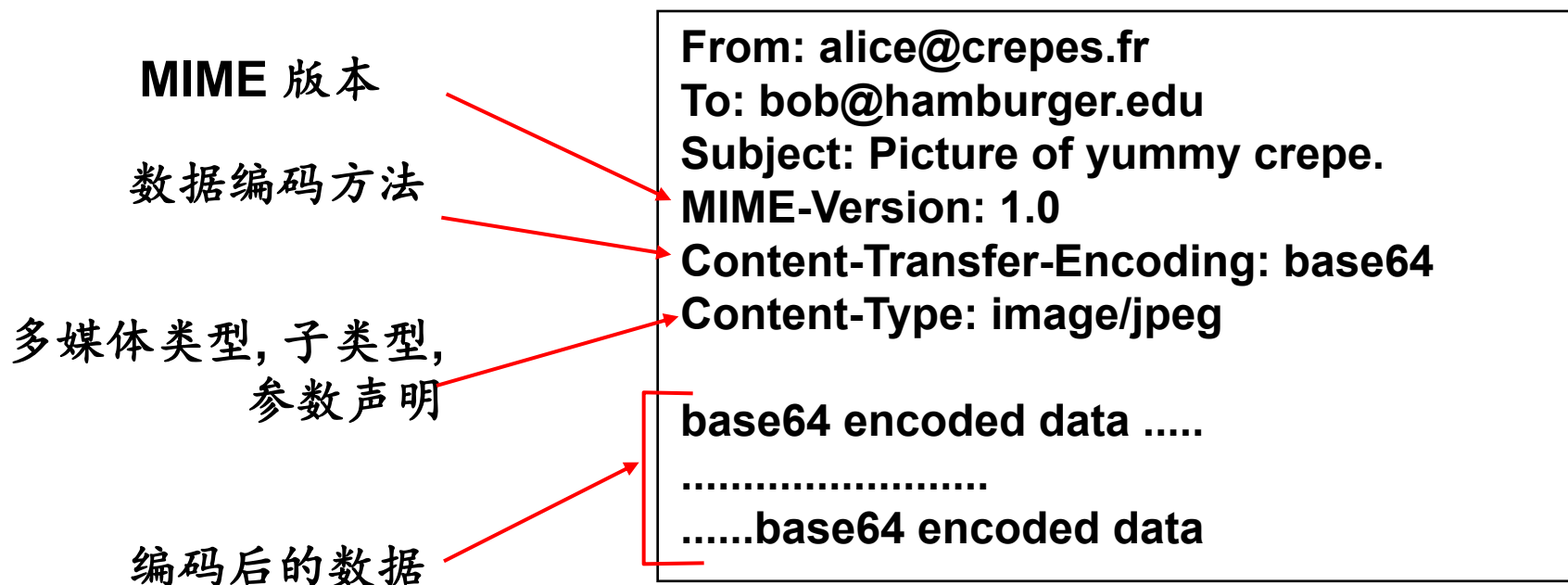
□ 信体

■ 即 “报文”, ASCII characters only



## 2.4 因特网中的电子邮件

### ■ 非ASCII码数据的MIME扩展



## 2.4 因特网中的电子邮件

### ■ 客户机获取邮件的方法

- POP3协议（Post Office Protocol）
- IMAP协议（Internet Mail Access Protocol）
- HTTP



## 2.4 因特网中的电子邮件

### ■ POP3协议的认证阶段

#### □ 客户端命令:

■ user: 用户名

■ pass: 口令

#### □ 服务器响应

**telnet mailserver 110**

■ +OK

■ -ERR

**S: +OK POP3 server ready**

**C: user alice**

**S: +OK**

**C: pass hungry**

**S: +OK user successfully logged on**

## 2.4 因特网中的电子邮件

### ■ POP3协议的交互命令

- list:  
列出报文号码
- retr:  
用报文号码取信
- dele:  
用报文号码删信
- quit

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

## 2.4 因特网中的电子邮件

### ■ POP3评述

- “下载-删除”：用户如果更换客户机无法再次阅读原来的邮件
- “下载-保存”：在邮件服务器上保存邮件的副本
- POP3会话是没有状态的
- 用户使用POP3协议无法在邮件服务器上对自己的邮件进行重组织，只能将邮件下载到本地计算机进行重组织

### ■ IMAP协议

- 将所有的邮件都保存在服务器上
- 允许用户在服务器上组织自己的邮件目录
- IMAP维护了IMAP会话的用户信息：
  - 目录名以及报文ID与目录名之间的映射关系

## 2.4 因特网中的电子邮件

### Web Mail

- 用户代理就是普通的浏览器
- 用户和其远程邮箱的通信（发送和接受）是通过HTTP协议进行
- 和IMAP一样，用户可以在远程服务器上以层次的方式组合他们的报文。

## 2.5 DNS:因特网的目录服务

### ■ 人类社会对人的识别

- 姓名
- 身份证号
- 护照号

.....

### ■ 网络社会对机器的识别

- MAC地址 (48bit)
- IP地址 (32bit)
- 域名 (不定长)

IP 地址和域名之间如何映射(转换) ?



为此人类设计了**DNS**系统，用于**IP**地址和域名之间的转换

计算机通信是通过IP地址和端口号来寻址 (Socket=IP+Port)，但人们不愿意记无意义的数值，愿意记有意义的名字-域名。但真正开始通信前，需要把域名转换成IP地址。

当在浏览器地址栏输入: `http://www.hust.edu.cn` 时，浏览器需要请求建立一个到 `www.hust.edu.cn` 的端口80的一个TCP连接，但建立TCP (Socket) 连接需要服务器的IP和端口，因此需要把域名转换成IP (这一步我们虽然看不到，但的确发生了)

## 2.5 DNS:因特网的目录服务

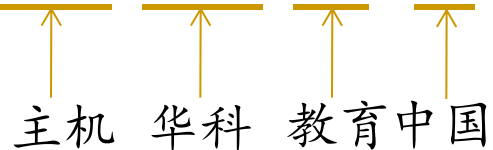
### ■ 人的通信地址有层次性

□ 湖北武汉华中科技大学计算机学院应用系



### ■ 计算机域名也有层次性

□ www. hust. edu. cn



## 2.5 DNS:因特网的目录服务

### ■ DNS简况

- ❑ DNS是一个分布式数据库，由很多台DNS服务器按照层次结构组织起来
- ❑ DNS运行在端到端系统上，且使用UDP协议（53号端口）进行报文传输，因此DNS是应用层协议
- ❑ DNS以C/S的模式工作
- ❑ DNS不直接和用户打交道，而是因特网的核心功能（是特例：应用层协议是因特网的核心功能）

## 2.5 DNS:因特网的目录服务

### ■ 一次最简单的DNS解析过程

#### □ 假设

- Alice通过IE浏览器访问www.hust.edu.cn/index.html
- Alice的主机上存在DNS客户机

#### □ 结果

- IE浏览器从URL中抽取出域名www.hust.edu.cn，将其传送给DNS客户机
- DNS客户机向DNS服务器发出一个包含域名的查询请求报文
- DNS服务器向DNS客户机送回一个包含对应IP地址的响应报文
- DNS客户机将该IP地址传送给IE浏览器
- IE浏览器向该IP地址所在WEB服务器发起TCP连接



## 2.5 DNS:因特网的目录服务

### ■ DNS的实现

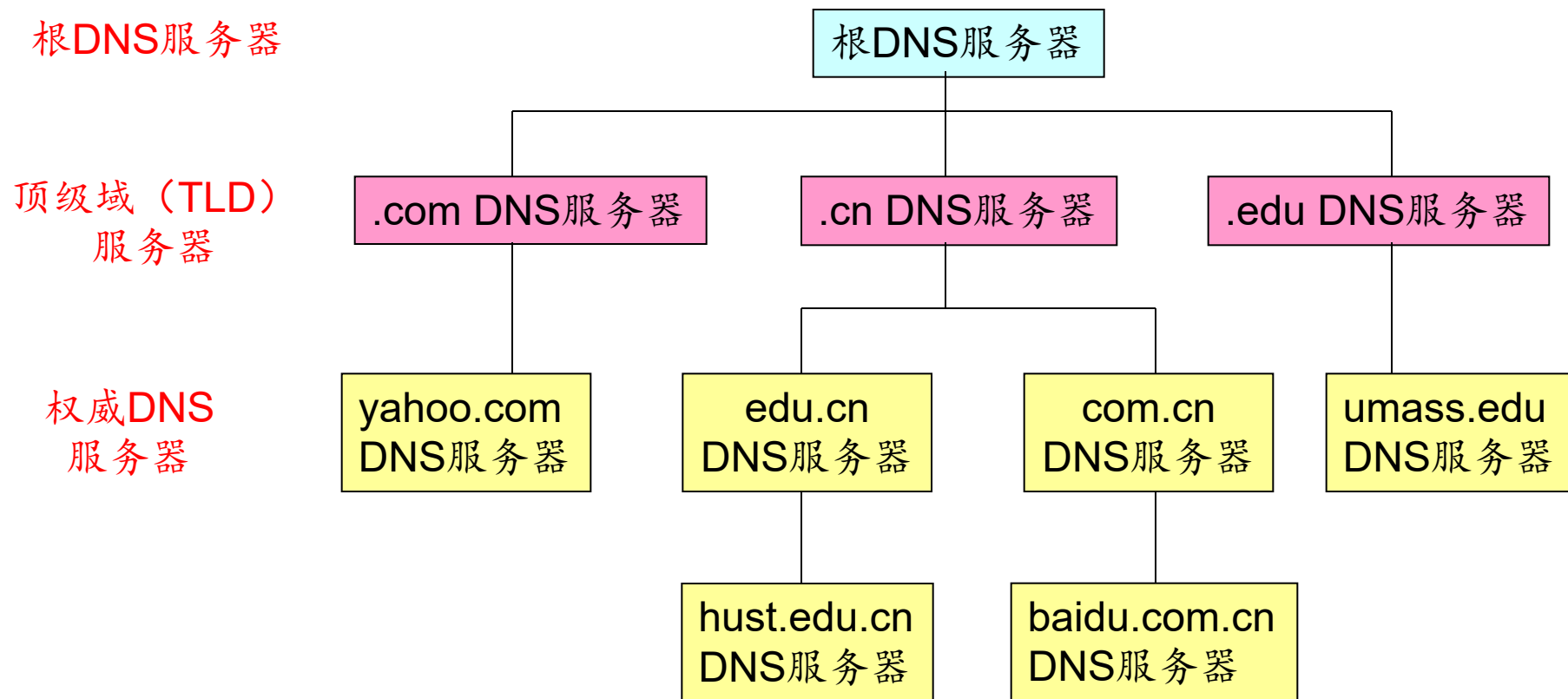
#### □ 最简单的方法——单台DNS服务器

- 单点故障的问题：一旦崩溃，因特网何以堪
- 数据的流通量：使得DNS服务器不堪重负
- 远程的集中式数据库：带来严重的延时
- 维护量巨大：DNS服务器不得不拼命的更新以适应因特网上主机的增加与减少

**显然，这种方法是世界上最笨的方法！！！！**

## 2.5 DNS:因特网的目录服务

### □ 真正的DNS实现：层次结构、分布式



## 2.5 DNS:因特网的目录服务

### ■ 根域名服务器（截止2012年，共13个）



## 2.5 DNS:因特网的目录服务

- **顶级域DNS服务器**：负责顶级域名和所有国家的顶级域名解析工作，例如：**com, org, net, gov, uk, cn, jp**等
  - Network Solution公司负责维护com顶级域DNS服务器
  - Educause公司负责维护edu顶级域DNS服务器
- **权威DNS服务器**：属于某个组织的DNS服务器，为组织的服务器提供一个权威的域名到IP地址的映射服务（例如：**Web** 和 **mail**）
  - 这些DNS服务器一般由所属组织或者服务提供商负责维护

## 2.5 DNS:因特网的目录服务

### ■ 本地DNS服务器

- 严格的讲，本地DNS服务器其并不属于DNS层次结构中的一层
- 每一个网络服务提供商都会提供一个本地DNS服务器
  - 有时候，我们将其称为“默认DNS服务器”
- 当一台主机需要做一个域名查询的时候，查询请求首先被发送到本地域名服务器
  - 本地域名服务器的行为就像一个代理，它会向域名的层次体系中进行进一步的域名查询。

## 2.5 DNS:因特网的目录服务

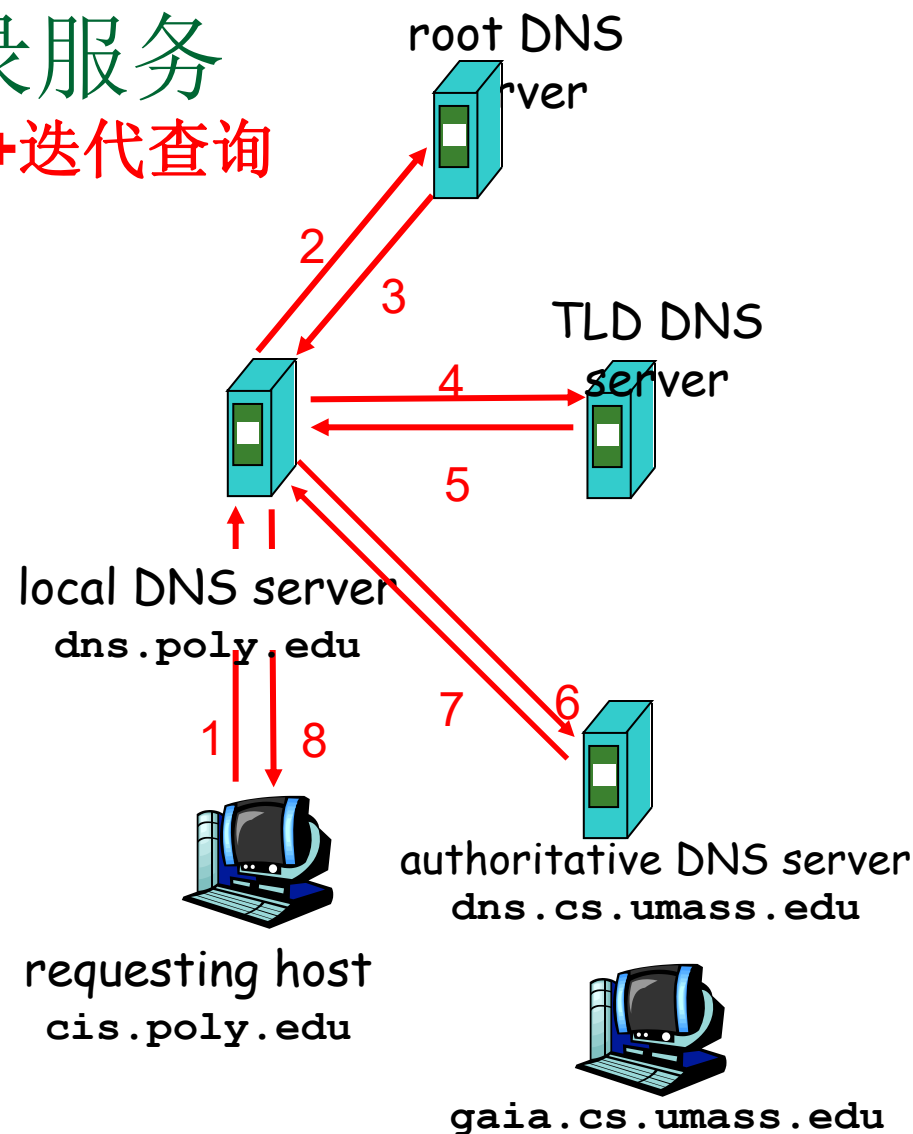
递归+迭代查询

主机 cis.poly.edu 要求

gaia.cs.umass.edu 的IP 地址

1. 联系local DNS server
2. local DNS server 查询root DNS
3. root DNS根据edu后缀返回负责edu的 TLD DNS IP地址
4. local DNS server 查询TLD
5. TLD注意到cs.umass.edu后缀，  
返回权威DNS服务器dns.cs.umass.edu  
的地址。
6. local DNS server 查询权威DNS  
dns.cs.umass.edu
7. 权威DNS dns.cs.umass.edu返回  
gaia.cs.umass.edu的IP给local DNS  
server
8. local DNS server返回结果给  
cis.poly.edu

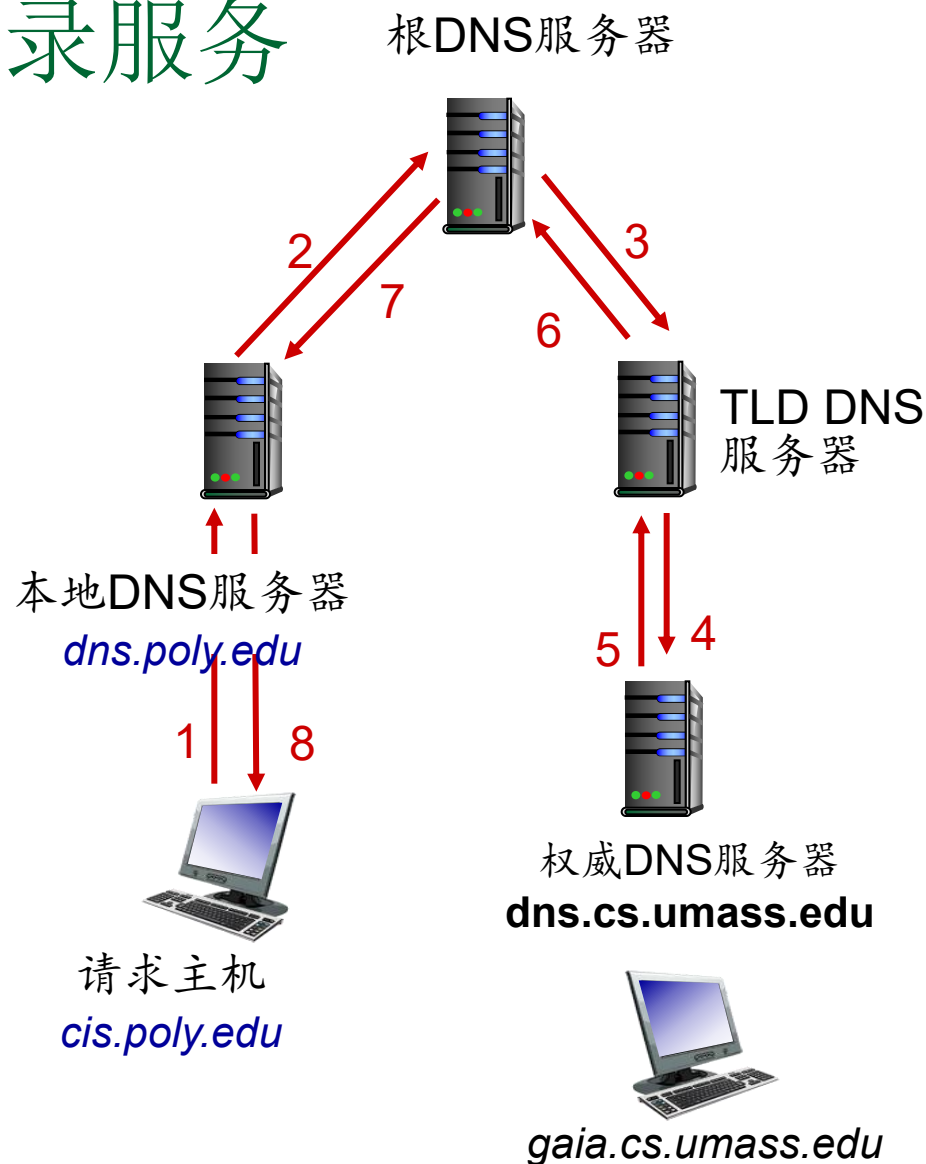
迭代查询：后续每一次查询都是由local  
DNS发起，结果返回给local DNS



## 2.5 DNS:因特网的目录服务

### ■ 另外一种DNS解析流程

#### □ 纯递归查询



## 2.5 DNS:因特网的目录服务

### ■ DNS缓存

- 一旦 (任何) 域名服务器得知了某个映射, 就将其 **缓存**
  - 在一定的时间间隔后缓存的条目将会过期(自动消除)
  - TLD DNS服务器通常被缓存在本地DNS服务器中
    - 这样可以减少根DNS的负载



## 2.5 DNS:因特网的目录服务

### ■ DNS可提供的服务

- 域名到IP地址的转换
  - 主机/邮件服务器别名
    - 为不好记的规范主机/邮件服务器名提供一个易记的别名
- e.g. [www.hotmail.com](http://www.hotmail.com) -> [www.hotmail.aate.nsatc.net](http://www.hotmail.aate.nsatc.net)
- 负载均衡
    - 一个域名对应多个IP
    - DNS服务器在多个IP中进行轮转

## 2.5 DNS:因特网的目录服务

### ■ DNS记录的格式 (RR: 资源记录)

RR 格式: (name, value, type, ttl)

#### ■ Type=A

- name = 主机名
- value = IP 地址

#### ■ Type=NS

- name = 域 (如foo.com)
- value = 该域权威域名服务器的 **主机名**

#### ■ Type=CNAME

- Name= 别名  
www.ibm.com
- value = 真名  
servereast.backup2.ibm.com

#### ■ Type=MX

- value 是别名为name的邮件服务器的规范主机名

## 2.5 DNS:因特网的目录服务

### ■ DNS记录的维护

- 目前基本上都是手工维护
- RFC2136定义了DNS动态更新

## 2.5 DNS:因特网的目录服务

### ■ 在DNS数据库中插入记录

- 例如：要注册一个域名 “network.com”
- 在DNS注册登记机构注册名字 “network.com”
  - 提供权威DNS服务器的名字和IP地址（包括基本的和辅助的）
  - 注册登记机构在com 顶级域名服务器中插入两条记录：  
(network.com, dns1.network.com, NS)  
(dns1.network.com, 212.212.212.1, A)
  - dns1.network.com负责解析network.com域内所有主机的域名
  - 如www.network.com, mail.network.com
- 在权威DNS服务器中为www.network.com创建A记录，为mail.network.com创建MX记录（在本机构内的域名的地址映射记录加在本域权威DNS服务器中）

## 2.5 DNS:因特网的目录服务

### ■ 攻击DNS服务器

#### □ DDoS攻击

- 通过ICMP Ping洪泛攻击根DNS服务器——难以成功
  - 根服务器通常配备分组过滤器
  - 大多数本地DNS服务器缓存了TLD DNS服务器的地址
- 通过DNS请求报文洪泛攻击TLD DNS服务器
  - 难以过滤

#### □ 重定向攻击

- 中间人攻击——攻击者截获来自主机的请求并返回伪造的回答
- DNS毒害攻击——攻击者向一台DNS服务器发送伪造的回答，诱使服务器在它的缓存中接收伪造的记录。

#### □ 利用DNS服务器对目标主机采用DDoS攻击——反射攻击

## 2.6 P2P文件共享

### ■ 一次传输的场景

- Alice在她的笔记本电脑上运行了一个P2P客户端应用
- 她不定期的连接到因特网上，每次都获得一个不同的IP地址
- 她希望获得这样的资源 “Hey Jude”
- 这个应用显示出拥有这个资源的所有其他的计算机
- Alice从中选择了Bob
- 这个资源从Bob的PC拷贝到了Alice的笔记本电脑上
- 当Alice在下载资源的时候，其他的用户也在向Alice的机器上传其他的资源
- Alice的计算机既是一个客户机，也是一个服务器

所有的计算机都是服务器 = 高可扩展性

## 2.6 P2P文件共享

$u_s$ : 服务器上传带宽

$u_i$ : 客户/对等方上传带宽

$d_i$ : 客户/对等方下载带宽

### ■ 文件分发

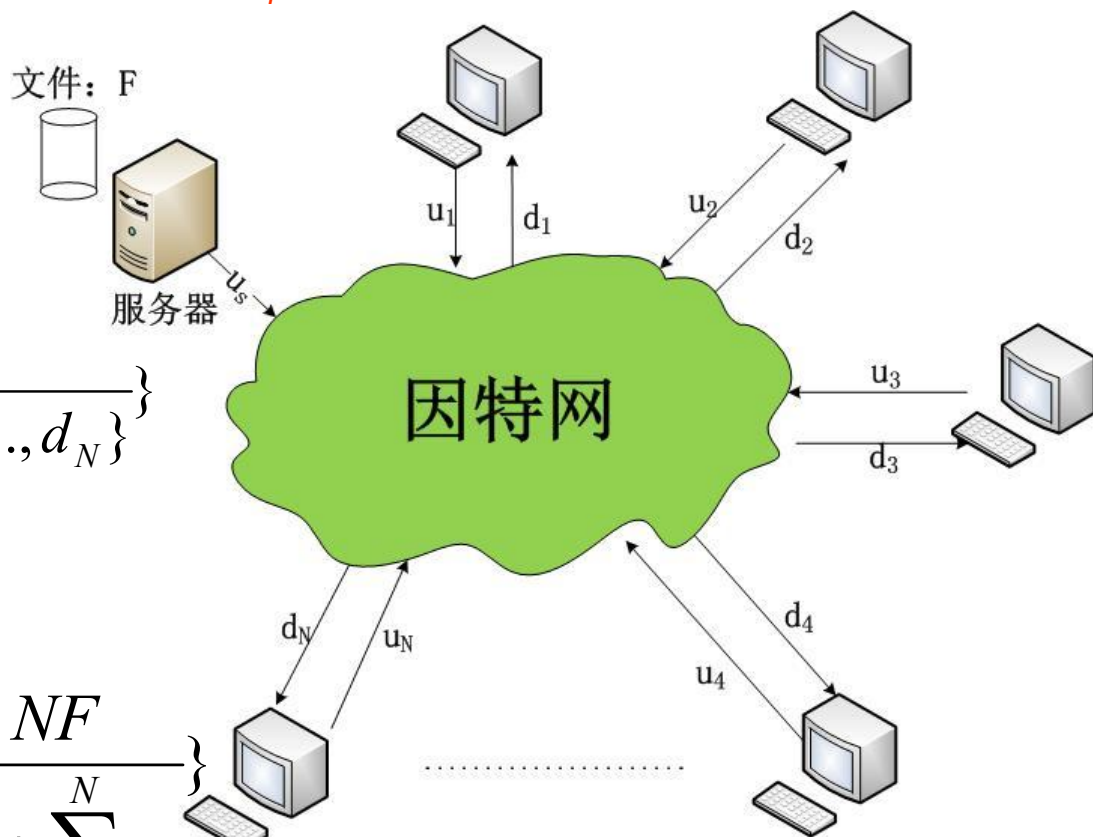
#### □ C/S模式

$$D_{cs} \geq \max \left\{ \frac{NF}{u_s}, \frac{F}{\min \{d_1, d_2, \dots, d_N\}} \right\}$$

#### □ P2P模式

$$D_{P2P} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\}$$

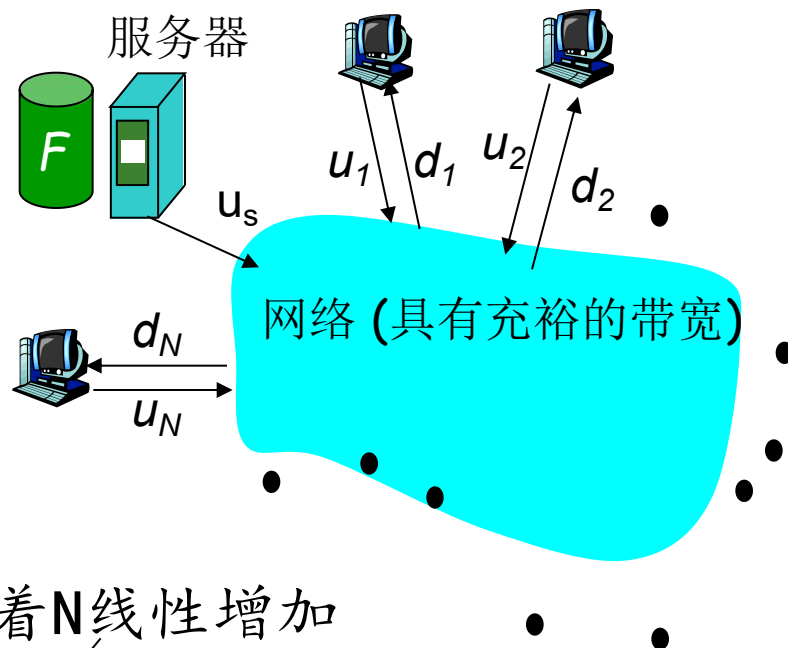
问题：从一个服务器向N个其它计算机传输一个文件需要多少时间？



## 2.6 P2P文件共享

客户-服务器：文件传输时间

- 服务器连续地发送N个拷贝：
  - 至少  $NF/u_s$  的时间
- 客户i花费  $F/d_i$  的时间来下载



随着N线性增加

在客户/服务器方法  
中将文件传输到  
N个客户需要的时间

$$= d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$$

$NF/u_s$ : 服务器将N个文件上传到网络需要的时间

$F/\min_i(d_i)$ : 下载文件需要的最长时间

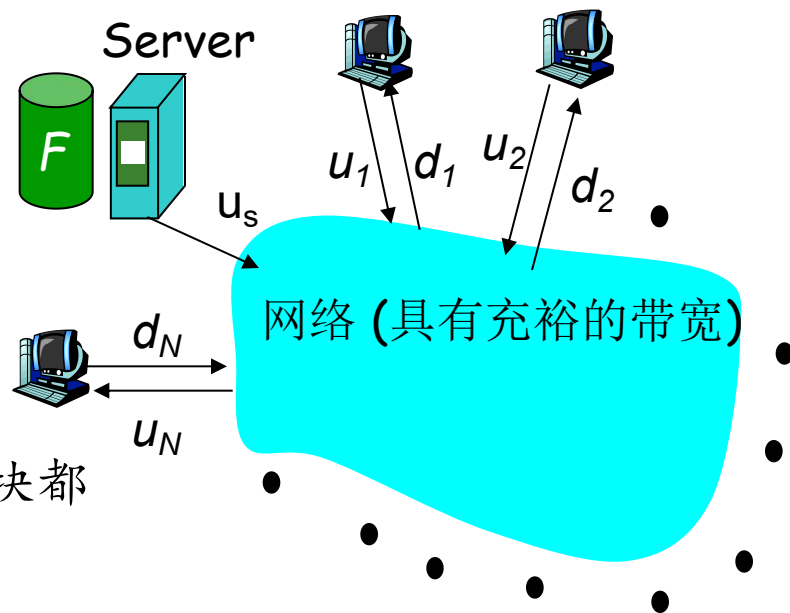
当N很大时,  $NF/u_s$  会很大,  
会决定  $d_{cs}$



## 2.6 P2P文件共享

### P2P: 文件传输时间

- 服务器发送一个文件的时间: 至少  $F/u_s$
- 客户  $i$  至少需要时间  $F/d_i$  下载该文件
- 总共有  $NF$  长度的内容需要下载
- 最高的上载速率 (假设所有节点将文件块都发送到同一个对等方):  $u_s + \sum_{i=1,N} u_i$



服务器只需要发送一个拷贝到网上, 时间  $F/U_s$

一个客户下载文件最长需要  $F/\min(d_i)$

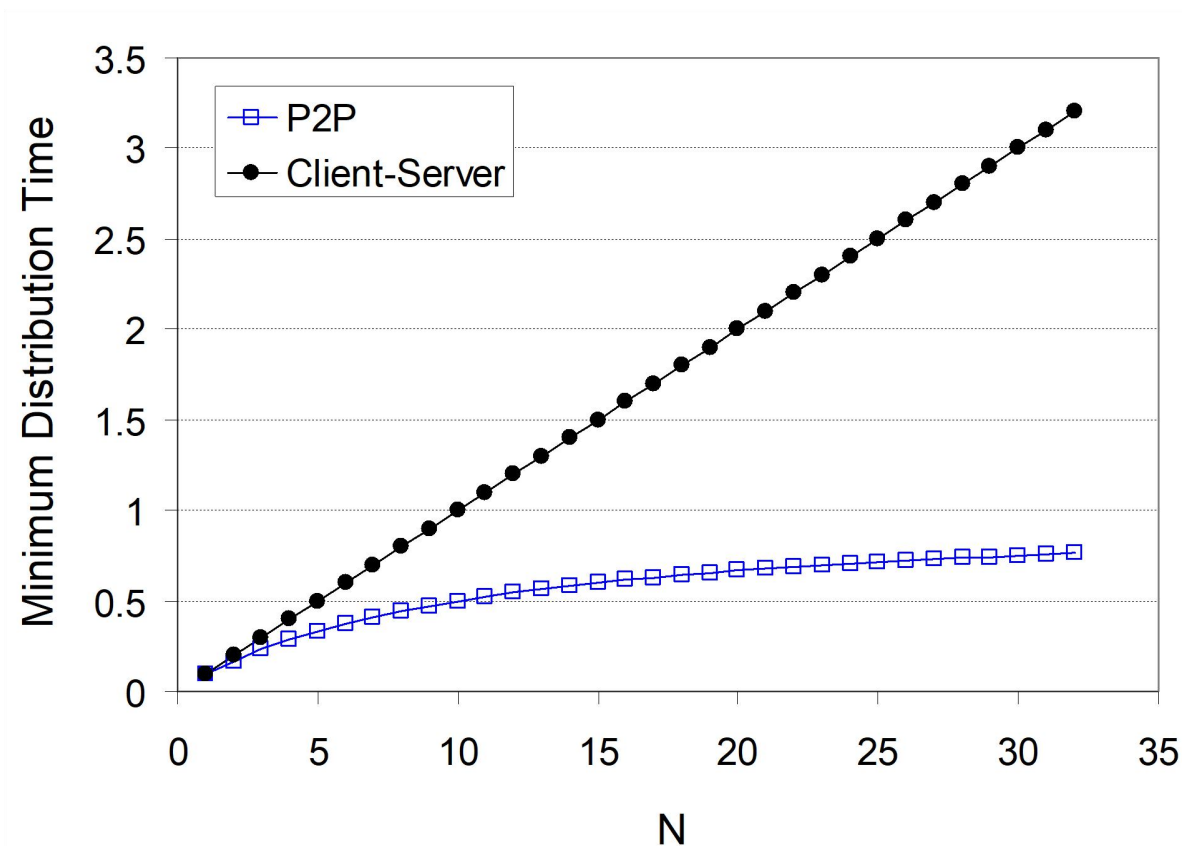
在P2P模式下如果同时共享已下载的文件, 总的上传速率为  $U_s + U_1 + \dots + U_n$ , 总共要上传  $N$  个拷贝, 所以时间为至少  $NF/(U_s + U_1 + \dots + U_n)$ 。这三件事并行, 因此。。。

$$d_{P2P} = \max \left\{ F/u_s, F/\min(d_i)_i, NF/(u_s + \sum_{i=1,N} u_i) \right\}$$

这一项不再随  $N$  线性增加

## 2.6 P2P文件共享

### ■ C/S和P2P体系结构的文件分发时间



## 2.6 P2P文件共享

- BitTorrent(一种P2P协议)的基本概念
  - 洪流(torrent):参与一个特定文件分发的所有对等方的集合
  - 追踪器(tracker):跟踪正参与在洪流中的对等方, 每个peer加入洪流时向tracker注册自己
  - 文件块(chunk): 256KB, 以chunk为单位分发

## 2.6 P2P文件共享

### ■ BitTorrent的基本工作机制

- 每个peer周期性地向邻居询问邻居所拥有的块列表
- 每个peer决定向邻居请求哪些块：最稀罕优先（使得最稀缺的块能迅速分发，最稀缺的块就是在邻居中副本最少的块）
- 每个peer决定要向哪些请求块的邻居发送：优先响应哪些请求——对换算法（4+1）
  - 每10秒重新确定4个最高速率对等方进行发送
  - 同时每30秒随机选择1个新的邻居

## 2.6 P2P文件共享

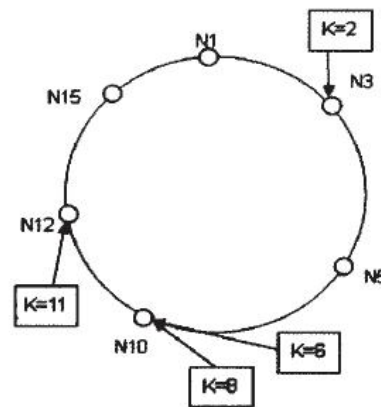
- 如何定位每个对等方所拥有的资源，将问题抽象化，用户的资源可以由键值对来描述，例如（资源标识-IP地址）。使用集中式的数据库存储这样的Hash表？

Key(资源标识)	Value(IP地址)
资源1	IP1
资源2	IP2
资源3	IP3
资源4	IP4
资源5	IP5
.....	.....
资源n	IPn

## 2.6 P2P文件共享

- 但是在P2P环境下，这样一张巨大的Hash表是分布式存储在多个peer上，每个peer存储的是这个表的一个很小的部分，这样分布式存储的表叫DHT（分布式Hash表）
- 讨论为通用键值对设计一个DHT的一般性问题
  - 每个对等方用n位比特来标识，每个对等方的标识符位于 $[0, 2^n-1]$ 之间
  - 要求每个键也是同一范围的整数，因此需要计算每个key的散列值，利用散列函数将每个键也映射为 $[0, 2^n-1]$ 之间的一个整数，后面讨论中key均指散列值
  - 为每个节点和关键字产生一个n位的ID和Key，并按照ID 大小构成环形拓扑。

Original Key	Key	Value
John Washington	8962458	132-54-3570
.....		.....
Lisa Kobayashi	9290124	177-23-0199



## 2.6 P2P文件共享

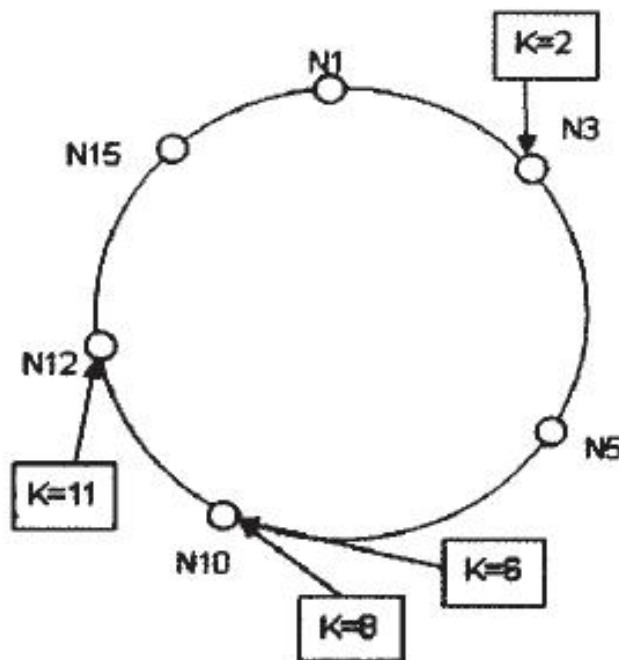
- Key的散列值和对等方的标识在相同的值空间中
- 键值对均匀分布在数以百万计的对等方上
  - 将键值对存放在拥有相同的值的对等方上
  - 即一个 (key, value) 对放在对等方标识=key的那个对等方上

这就是DHT（分布式散列表）

- 问题：如果找不到拥有相同值的对等方怎么办？
- 关键字标识为K的 (K, V) 对存储在这样的节点上，该节点的节点标识等于K或者在环上紧跟在K之后 ( $ID \geq K$ )，这个节点被称为K的后继节点，表示为 **successor (K)**
- 因为标识符采用n位二进制数表示，并且从0到 $2^n - 1$ 顺序排列成一个圆圈，**successor (K)**就是从K开始顺时针方向距离K最近的节点。

## 2.6 P2P文件共享

- 图给出了一个 $n=4$ 的环，环中分布了6个节点，存储了4个关键字，节点标识前加上N而关键字前加上K以示区别。
- 因为 $\text{successor}(2)=3$ ，所以关键字2存储到节点3上。同理，关键字6和8存储到节点10上，而关键字11存储到节点12上。



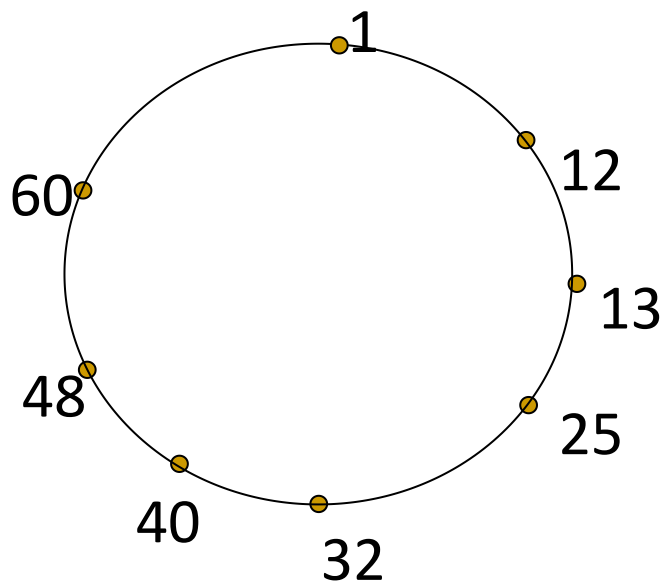


## 2.6 P2P文件共享

- 规则：为具有最接近ID的对等方分配键值对。
- 惯例：“最接近”表示键的**最临近后继**。
- 例如：ID 空间  $\{0, 1, 2, 3, \dots, 63\}$
- 假设有8个对等方：1, 12, 13, 25, 32, 40, 48, 60
  - 如果key = 51，则存储在对等方60上
  - 如果key = 60，则存储在对等方60上
  - 如果key = 61，则存储在对等方1上
- 为保证查询效率，每个对等方需要知道所有的其它对等方，并与其建立联系，但是……

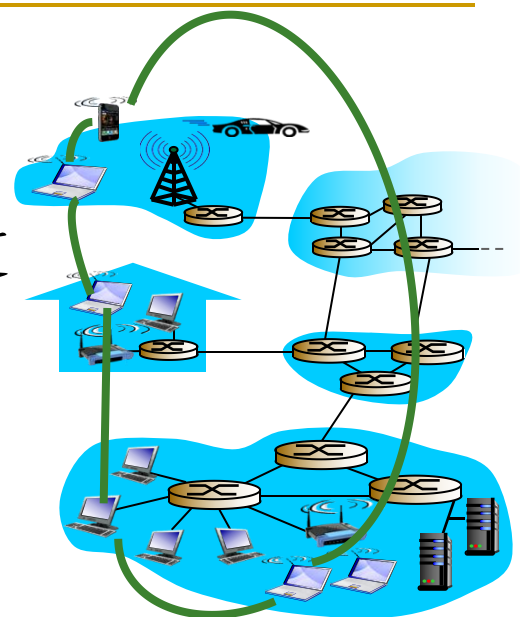
## 2.6 P2P文件共享

### ■ 环形DHT的查询：哪个节点负责11键



覆盖网络 (overlay network)

- 对等方X和Y之间如果维护了一条TCP连接，我们称它们之间有一条边
- 所有活动的对等方和边构成了“覆盖网络”
- 边是一个虚拟（而非物理）链路
- 一个对等方通常只与该覆盖网络中的少量节点连接（少于10个）（限制覆盖网络的规模）

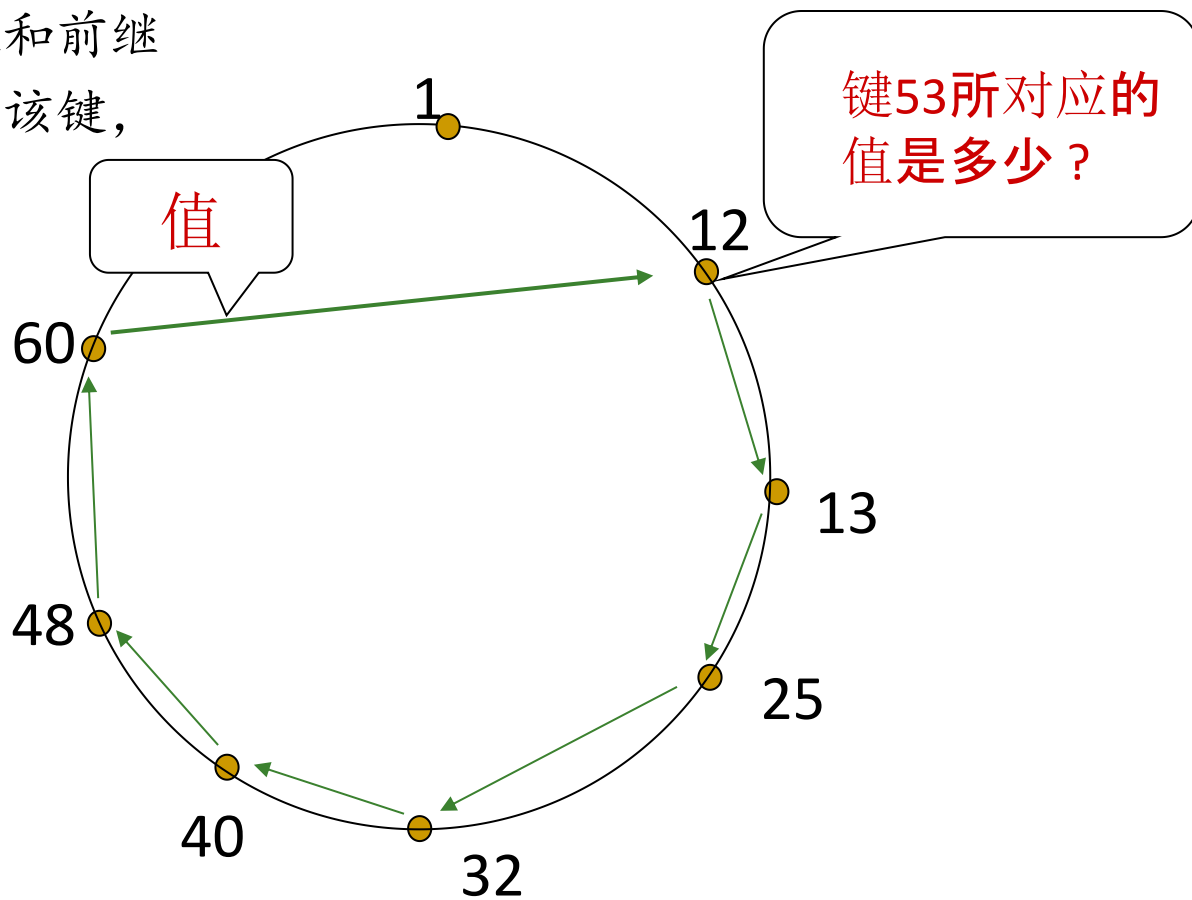


“覆盖网络”

## 2.6 P2P文件共享

### ■ 解析查询

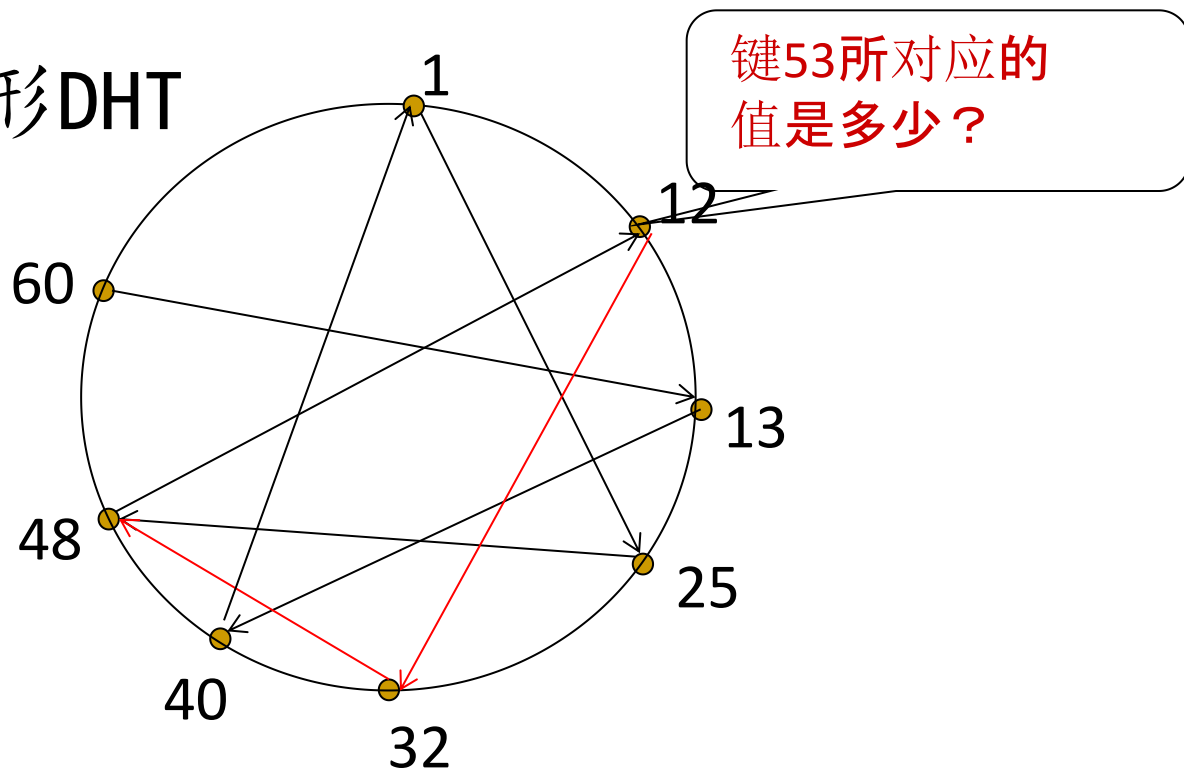
- 每个节点只知道后记和前继
- 如果一个节点不负责该键，  
将报文转发给后继



解决每个查询请求  
的报文数量均为 $O(N)$ ,  
 $N$ 为对等方的数量

## 2.6 P2P文件共享

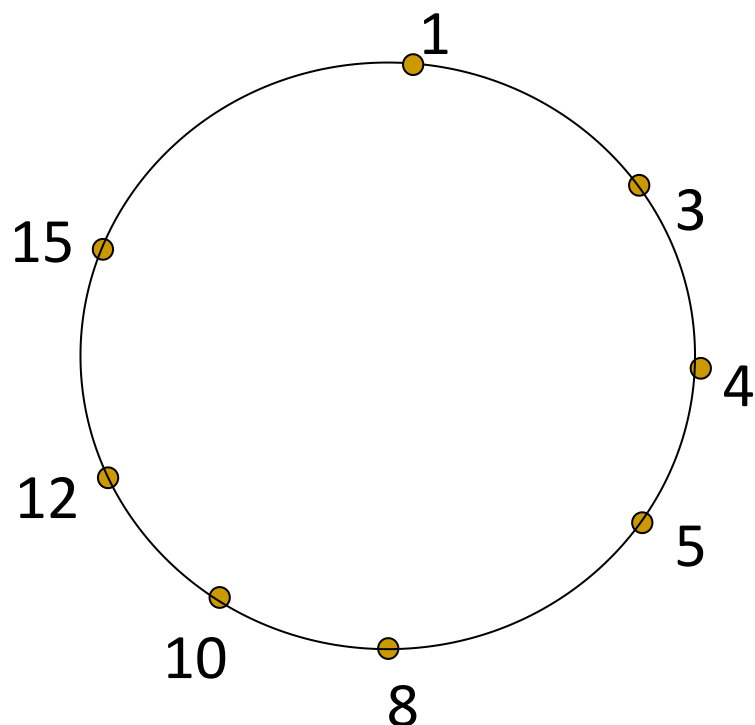
### ■ 具有捷径的环形DHT



- 每个对等方保存前任、后继和捷径邻居的IP地址.
- 从6条报文减少到3条 (12-32-48-60) .
- DHT能设计成每个对等方的邻居数量以及每个请求的报文数均为  $O(\log N)$

## 2.6 P2P文件共享

### ■ 对等方扰动

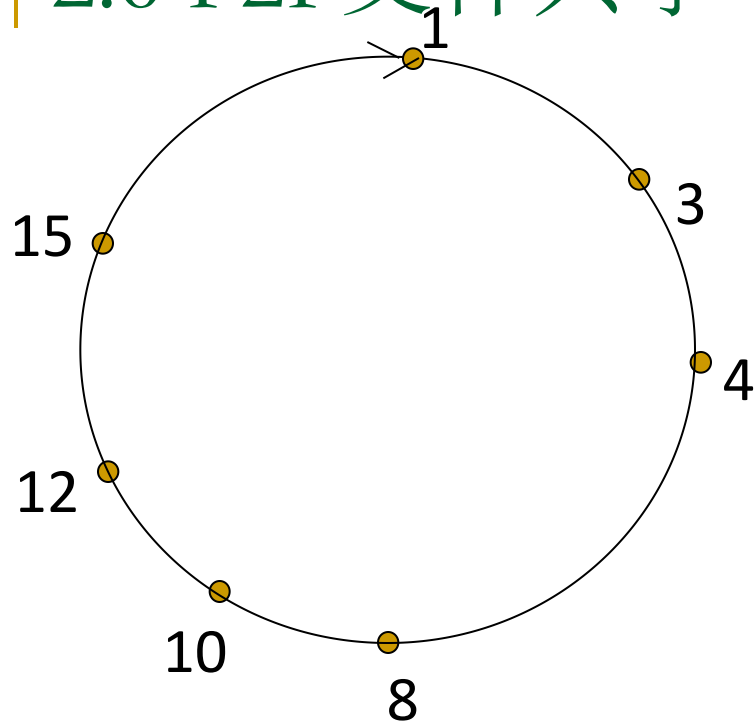


例子：对等方5突然离开

### 处理对等方扰动：

- ❖ 对等方可以随时到来和离去(扰动)
- ❖ 每个对等方知道其第一个和第二个后继的IP地址
- ❖ 每个对等方周期性地向两个后继发送ping报文以检验它们是否存活
- ❖ 如果第一后继离开，则选择第二个后继作为新的第一后继

## 2.6 P2P文件共享



### 处理对等方扰动:

- ❖ 对等方可以随时到来和离去(扰动)
- ❖ 每个对等方知道其第一个和第二个后继的IP地址
- ❖ 每个对等方周期性地向两个后继发送ping报文以检验它们是否存活
- ❖ 如果第一后继离开, 则选择第二个后继作为新的第一后继

### 例子: 对等方5突然离开

- 对等方4检测到对等方5离开; 用其第二个后继(对等方8)来成为其第一个后继
- 对等方4询问对等方8的直接后继是哪个; 使8的直接后继成为4第二个后继.

## 2.6 视频流和内容分发网

- 视频流量：互联网带宽的主要消费者
  - Netflix、YouTube: 37%, 16% 的下游住宅ISP流量
  - 约10亿YouTube用户，约7500万Netflix用户
- 挑战：规模—如何达到~1b用户？
  - 单个视频服务器无法工作（为什么？）
- 挑战：异质性
  - 不同的用户具有不同的功能（例如：有线与无线；宽带与窄带）

**解决方案：分布式应用程序级基础架构**



## 2.6 视频流和内容分发网

- 视频：以恒定速率显示的图像序列

- 例如，24个图像/秒

- 数字图像：像素阵列

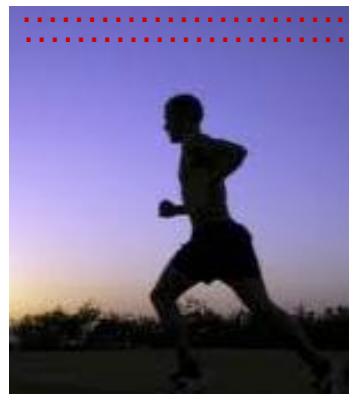
- 以位表示的每个像素

- 编码：使用图像内部和图像之间的冗余来减少用于编码图像的位

- 空间（图像内）

- 时间（从一个图像到下一个图像）

空间编码示例：不发送相同颜色的 $n$ 个值（全部为紫色），只发送两个值：颜色值（紫色）和重复值数（ $N$ ）



帧  $i$

时态编码示例：不在 $i+1$ 发送完整帧，只发送与帧 $i$ 的差异



帧  $i+1$



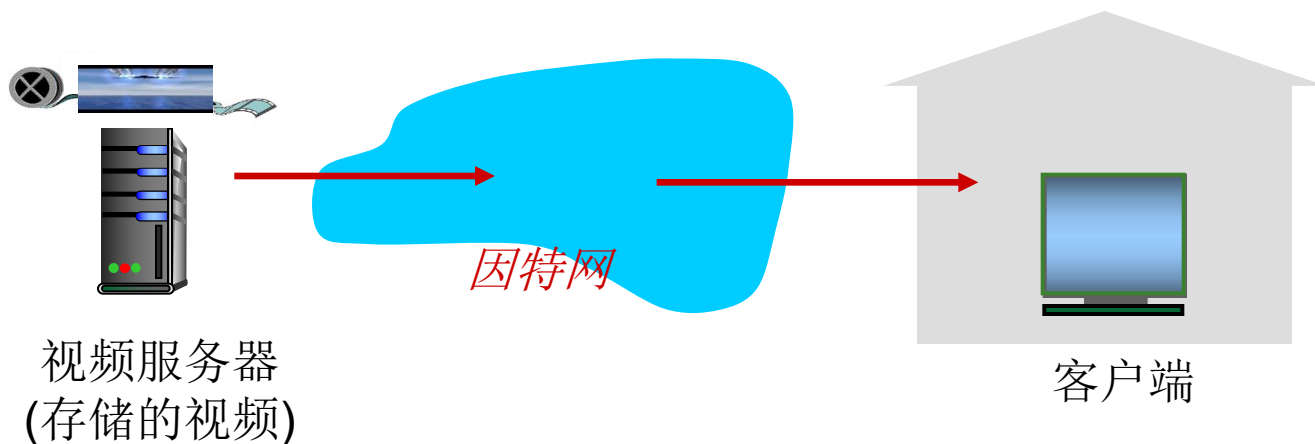
## 2.6 视频流和内容分发网

### ■ 视频编码方式

- CBR: (恒定比特率): 视频编码速率固定
- VBR: (可变比特率): 视频编码速率随空间、时间编码量的变化而变化
- 示例:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (通常用于互联网, < 1 Mbps)

## 2.6 视频流和内容分发网

### ■ 流存储视频



- 不同的客户可能拥有不同的带宽/同一客户不同时间的带宽也是变化的
- 给予的是相同编码的视频

## 2.6 视频流和内容分发网

### ■ 经HTTP的动态适应性流（DASH）

#### □ 服务器：

- 将视频文件分成多个文件块
- 每个块都以不同的速率存储、编码
- 清单文件：为不同的块提供URL

#### □ 客户端：

- 定期测量服务器到客户端的带宽
- 查询清单，一次请求一个块
  - 选择当前带宽可选的最大编码率
  - 可以在不同的时间点选择不同的编码率（取决于当时的可用带宽）

## 2.6 视频流和内容分发网

### ■ 经HTTP的动态适应性流（DASH）

#### □ 客户的“智慧”：客户决定

- 何时请求块（以避免缓冲区不足或溢出）
- 请求的编码速率是多少（当可用带宽更多时，质量更高）
- 请求块的位置（可以从“接近”客户端或具有高可用带宽的URL服务器请求）

## 2.6 视频流和内容分发网

### ■ 内容分发网

- **挑战**：如何将内容（从数百万个视频中选择）同时传输给数十万用户？
- **选项1**：单个，大型“超级服务器”
  - 单点故障
  - 网络拥塞点
  - 通往远方客户的漫长道路
  - 通过外发链接发送的多个视频副本

… 非常简单：此解决方案无法扩展

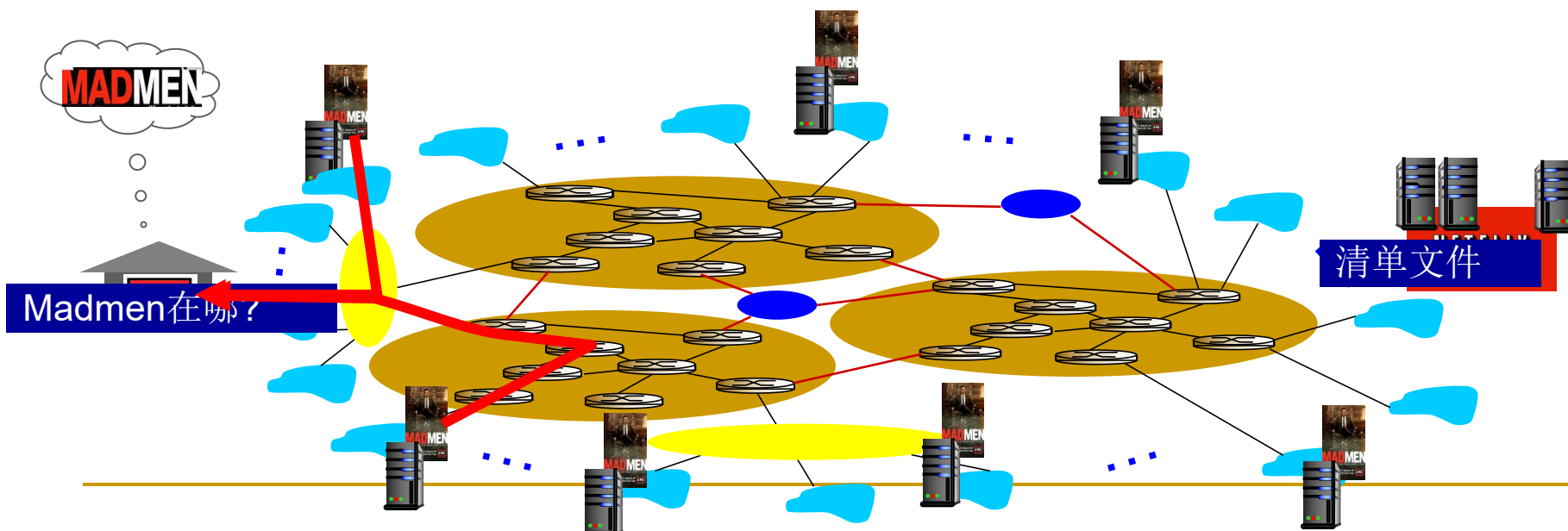
## 2.6 视频流和内容分发网

### ■ 内容分发网

- **挑战**：如何将内容（从数百万个视频中选择）同时传输给数十万用户？
  
- **选项2**：在多个地理位置分散的站点（CDN）存储/提供多个视频副本
  - 专有CDN
  - 第三方CDN

## 2.6 视频流和内容分发网

- **CDN**: 在CDN节点存储内容副本
  - 例如, Netflix存储Madmen的副本
- 订阅服务器从cdn请求内容
  - 定向到附近的副本, 检索内容
  - 如果网络路径拥挤, 可以选择不同的副本



## 2.6 视频流和内容分发网

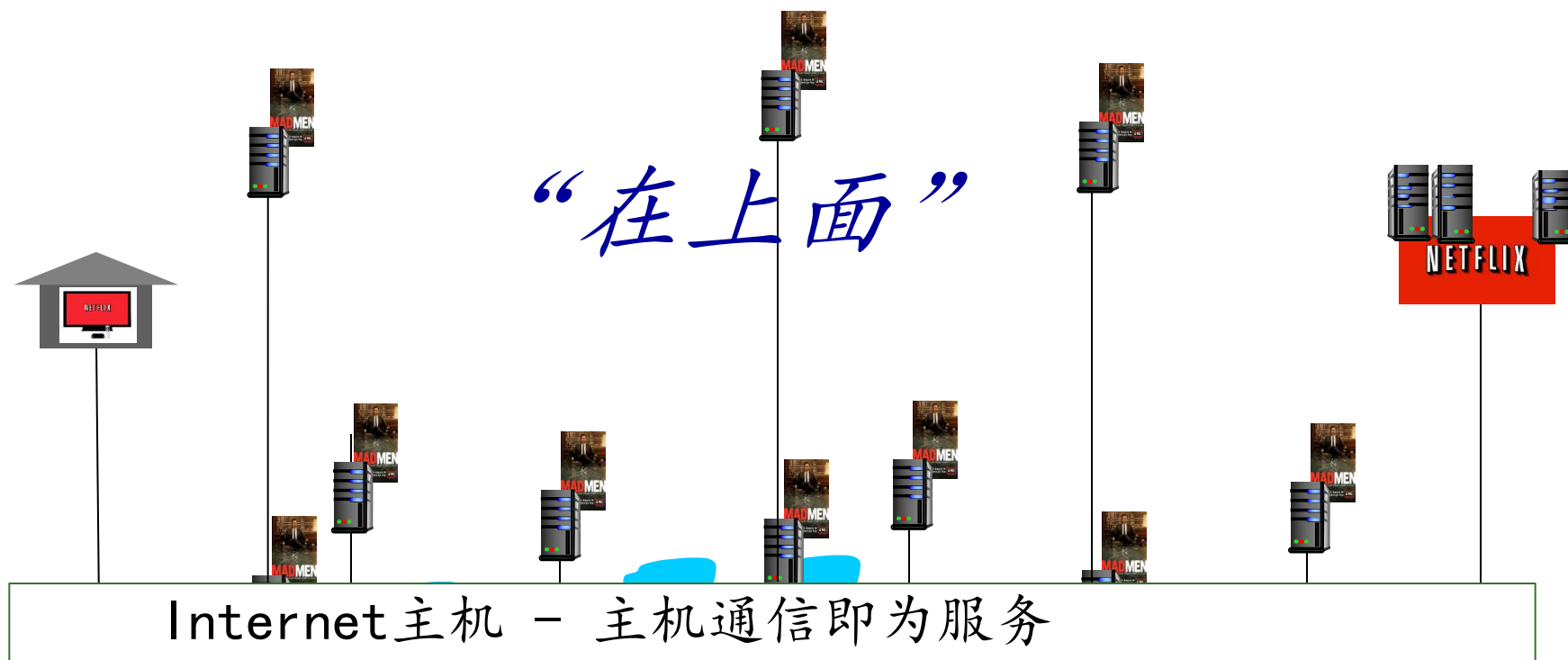
### ■ 内容分发网

#### □ CDN服务器部署原则

- 深入：将CDN服务器深入到许多接入网络中
  - 贴近用户
  - 由Akamai使用，1700个地点
- 邀请做客：将服务器集群放置在因特网交换点（IXP），集群较大，数量较少（10个）
  - 较低的维护和管理开销
  - 较高的时延和较低的吞吐量
  - 由Limelight使用



# 内容分发网络 (CDNs)



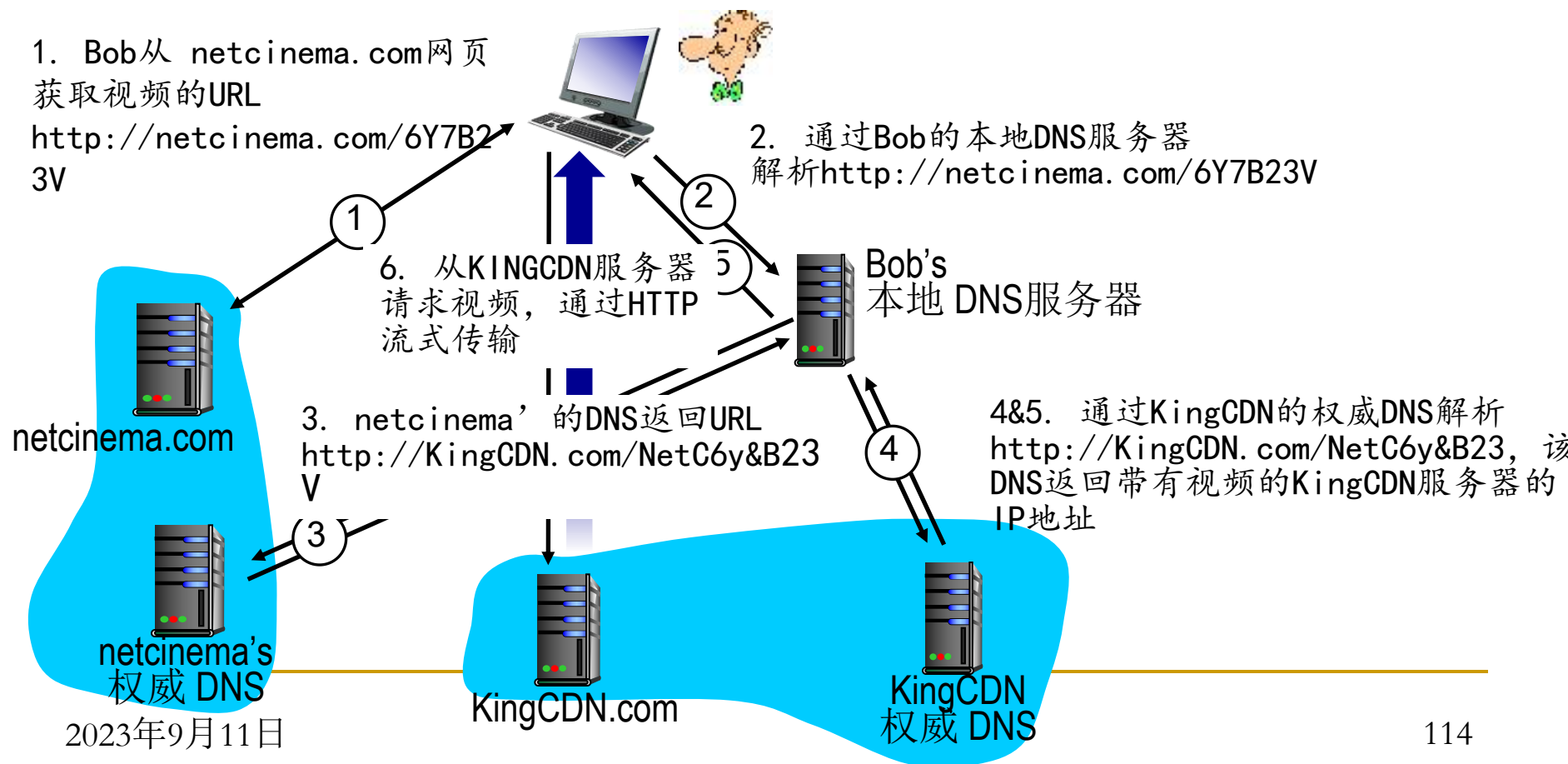
## OTT挑战：应对拥挤的互联网

- ❑ 从哪个CDN节点检索内容？
- ❑ 观众在拥挤时的行为？
- ❑ 在哪个CDN节点中放置什么内容？

## 2.6 视频流和内容分发网

Bob (客户端) 请求视频 `http://netcinema.com/6Y7B23V`

- 视频存储在CDN at `http://KingCDN.com/NetC6y&B23V`



## 2.6 视频流和内容分发网

### ■ 内容分发网

#### □ 集群选择策略

##### ■ 地理上最为邻近策略

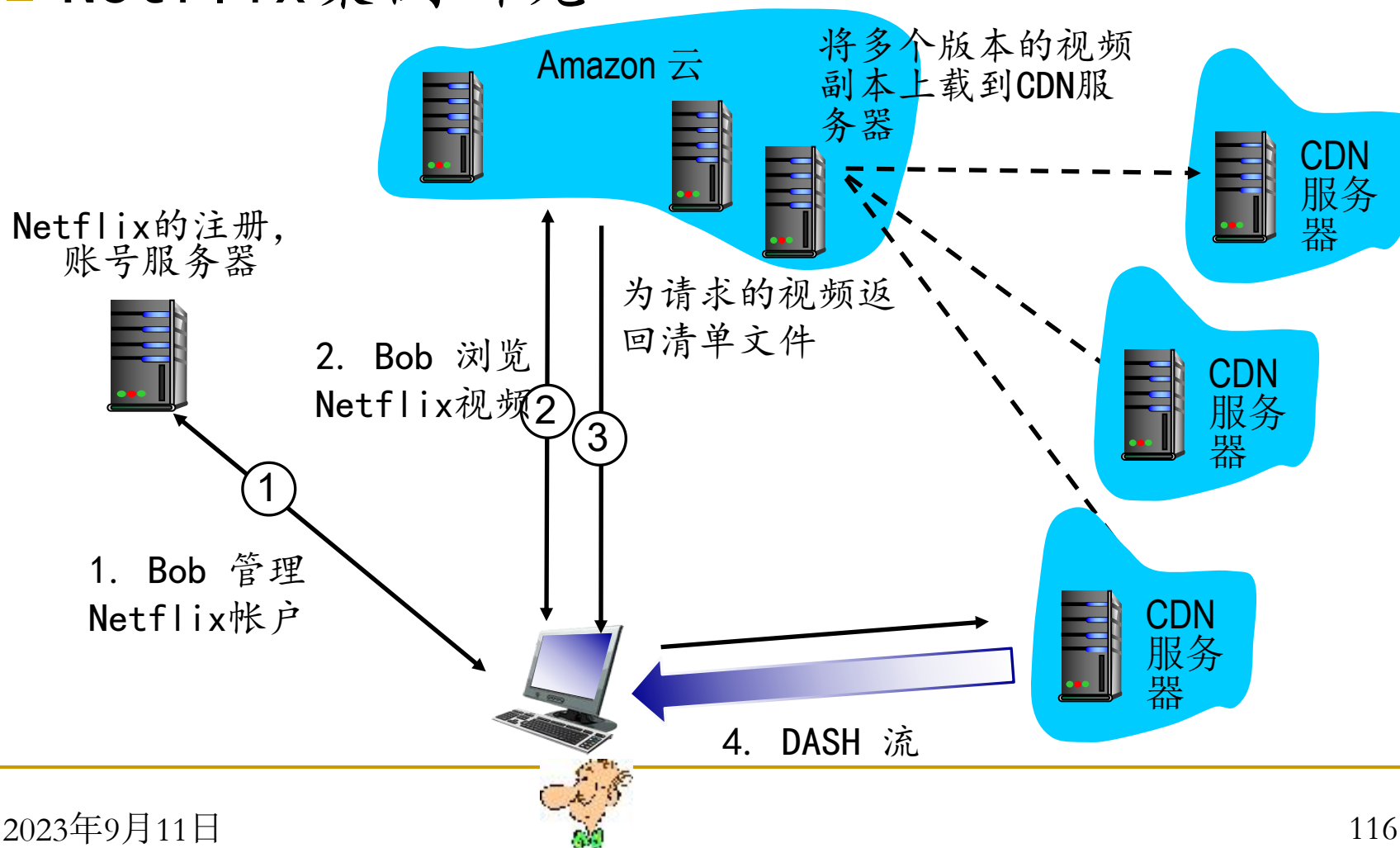
- 根据商用地理位置数据库，选择与客户LDNS最近的集群
- 就网络路径的长度或跳数而言，地理上的邻近并不是网络上的邻近
- 某些用户使用远地LDNS
- 没有考虑时延和带宽随事件的变化

##### ■ 周期性实施测量策略

- 周期性测量CDN各集群到全球各LDNS的时延
- 并不是每一台LDNS允许被测量

## 2.6 视频流和内容分发网

### ■ Netflix 案例研究



## 2.6 视频流和内容分发网

### ■ 迅雷看看代表的CDN-P2P混合模式

#### □ 工作原理

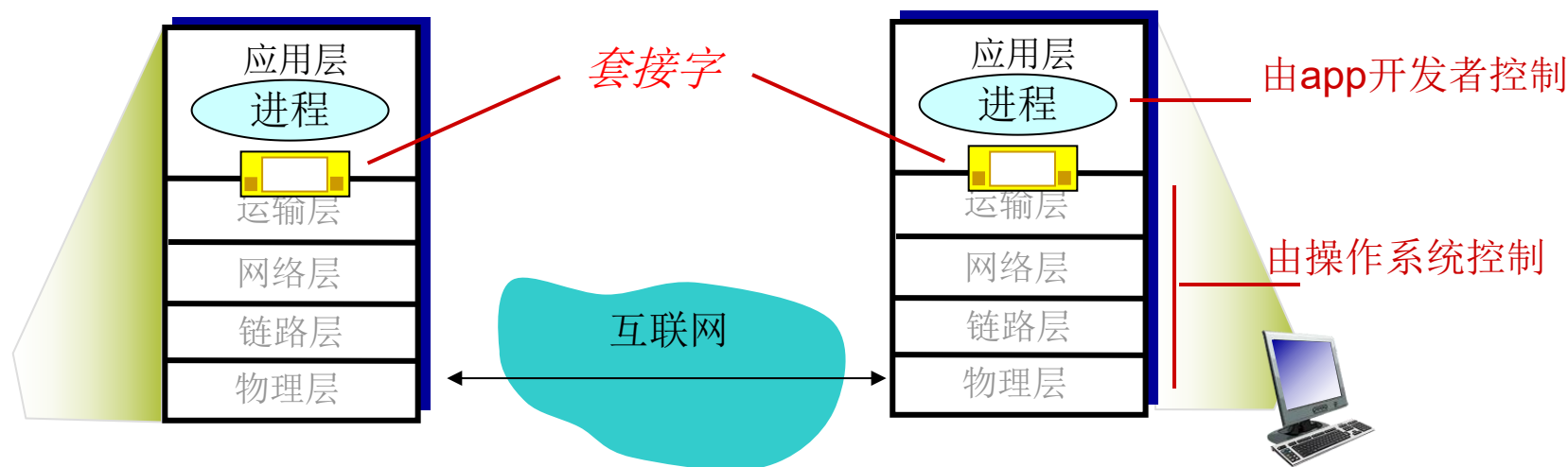
- 通过CDN提供内容的开头部分
- 并行地从对等方请求内容
- 如果P2P流量能够满足视频播放需要，停止从CDN获取内容，仅从P2P获取
- 如果P2P流量不充分，重新启动CDN切换到CDN-P2P混合模式

#### □ 优势

- 极大地减少服务提供商的基础设施和带宽成本

## 2.7 套接字编程：生成网络应用

- **目标：** 了解如何使用套接字构建用于通信的客户机/服务器应用程序
- **套接字：** 应用程序进程和端到端传输协议之间的门



## 2.7 套接字编程：生成网络应用

### ■ 两种传输服务的两种套接字类型：

- UDP：不可靠的数据报
- TCP：可靠，面向字节流

### 应用实例：

1. 客户机从键盘读取一行字符（数据）并将数据向服务器发送
2. 服务器接收数据并将字符转换为大写
3. 服务器将修改的数据发送给客户
4. 客户接收修改的数据并在其监视器上将该行显示出来

## 2.7 套接字编程：生成网络应用

- **UDP：**客户端和服务端之间没有“连接”
  - 发送数据前不要握手
  - 发送方明确地将IP目的地址和端口号附加到每个数据包
  - 接收方从接收的数据包中读取发送方IP地址和端口号
  
- **UDP：**传输的数据可能丢失或无序接收
  
- **应用程序视角：**
  - UDP 在客户端和服务端之间提供不可靠的字节组传输（“数据报”）



## 2.7 套接字编程：生成网络应用

### 服务器（在 server IP 上运行）

创建套接字 port= x:

```
serverSocket =  
socket(AF_INET, SOCK_DGRAM)
```

从serverSocket  
读取UDP报文段

向serverSocket写指定客  
户端地址，端口号的响应  
报文

### 客户

创建套接字:

```
clientSocket =  
socket(AF_INET, SOCK_DGRAM)
```

创建具有server IP 和  
port=x的数据报；经  
clientSocket发送数据报

从clientSocket  
读数据报

关闭  
clientSocket

## 2.7 套接字编程：生成网络应用

### *Python UDPClient*

包括python的socket库

→ from socket import \*

serverName = 'hostname'

serverPort = 12000

为服务器创建UDP套接字

→ clientSocket = socket(AF\_INET,  
SOCK\_DGRAM)

获取用户键盘输入

→ message = raw\_input('Input lowercase sentence:')

将服务器名称、端口附加到  
消息；发送到套接字

→ clientSocket.sendto(message.encode(),  
(serverName, serverPort))

将回复字符从socket读取到  
字符串中

→ modifiedMessage, serverAddress =  
clientSocket.recvfrom(2048)

打印出接收到的字符串并关  
闭套接字

→ print modifiedMessage.decode()  
clientSocket.close()

## 2.7 套接字编程：生成网络应用

### *Python UDPServer*

```

from socket import *
serverPort = 12000

创建 UDP 套接字 → serverSocket = socket(AF_INET, SOCK_DGRAM)
将套接字绑定到本地端口号12000 → serverSocket.bind(("", serverPort))
print ( "The server is ready to receive")

循环 → while True:
从UDP套接字读取消息，获取客户端地址（客户端IP和端口） → message, clientAddress = serverSocket.recvfrom(2048)
modifiedMessage = message.decode().upper()

将大写字符串发送回此客户端 → serverSocket.sendto(modifiedMessage.encode(),
clientAddress)
    
```

## 2.7 套接字编程：生成网络应用

### 使用TCP进行套接字编程

#### ■ 客户端必须与服务器联系

- 服务器进程必须首先运行
- 服务器必须创建欢迎客户联系的套接字（门）

#### ■ 客户端联系服务器通过：

- 创建TCP套接字，指定服务器进程的IP地址，端口号
- 当客户端创建套接字时：客户端TCP建立到服务器TCP的连接

- 当用客户端联系时，服务器TCP为服务器进程创建新的套接字，以便与该特定客户端通信。

- 允许服务器与多个客户端对话
- 用于区分客户端的源端口号（第3章中有详细介绍）

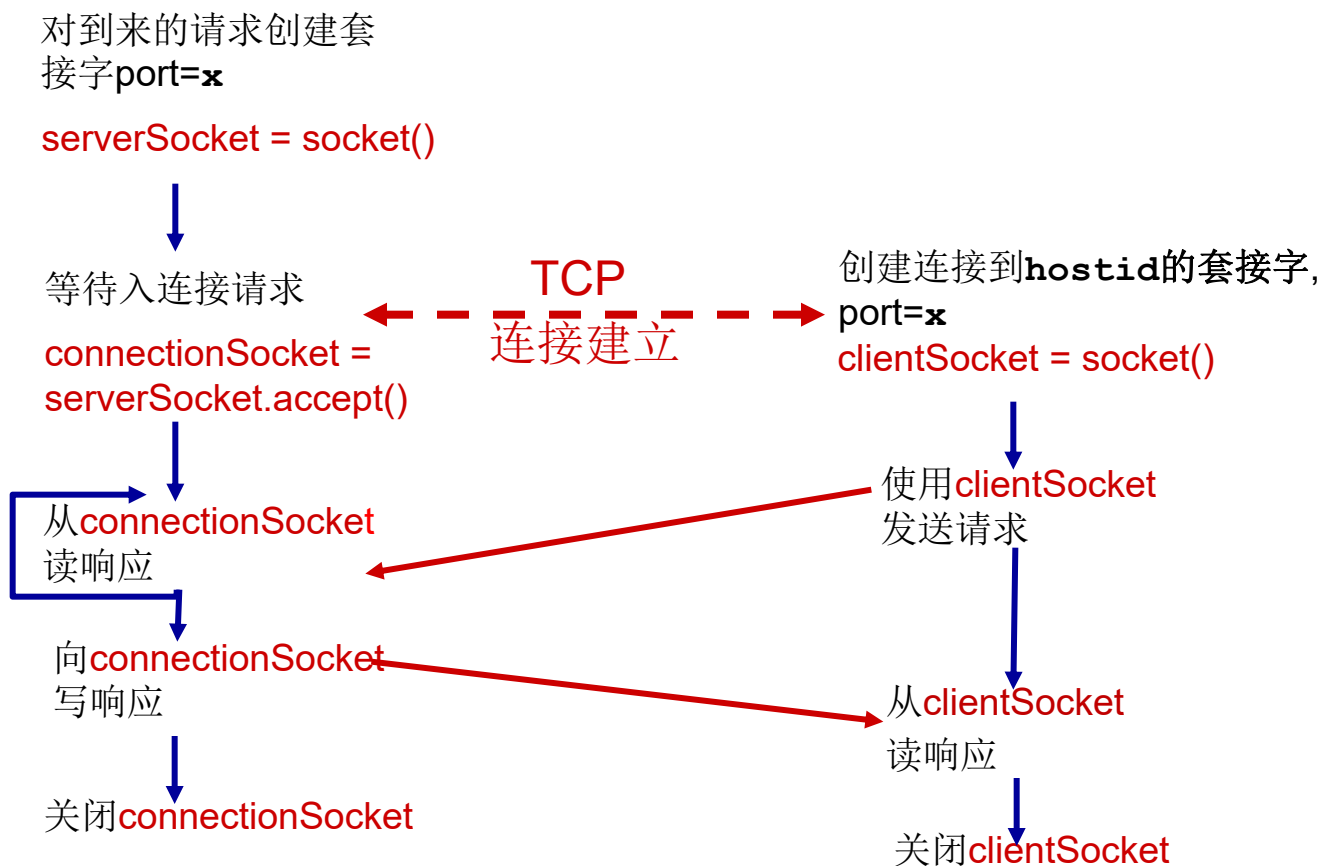
#### 应用程序视角：

TCP在客户端和服务端之间提供可靠的有序字节流传输（“管道”）

## 2.7 套接字编程：生成网络应用

服务器 (在`hostid`运行)

客户



## 2.7 套接字编程：生成网络应用

### *Python TCPCliet*

```

from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
    
```

为服务器创建TCP套接字，远  
程端口12000

→

无需附加服务器名称，端口

→

## 2.7 套接字编程：生成网络应用

### *Python TCPServer*

创建TCP欢迎套接字

服务器开始侦听传入的TCP  
请求

循环

服务器在accept（）上等待传入  
请求，在返回时创建新套接字

从套接字读取字节（但不是  
UDP中的地址）

与此客户端关闭连接（但没  
有欢迎套接字）

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                           encode())
    connectionSocket.close()
```

# 本章小结

*我们对网络应用的学习现已完成！*

- 应用程序架构
  - 客户-服务器
  - P2P
- 应用服务要求：
  - 可靠性、带宽、延迟
- 互联网传输服务模式
  - 面向连接、可靠：TCP
  - 不可靠、数据报：UDP
- 具体协议：
  - HTTP
  - SMTP、POP、IMAP
  - DNS
  - P2P: BitTorrent
- 视频流 CDNs
- 套接字编程：TCP、UDP 套接字



# 本章小结

**最重要的是：了解协议！**

- 典型的请求/回复消息交换：
  - 客户端请求信息或服务
  - 服务器用状态代码，数据响应，
- 消息格式：
  - 首部：提供数据信息的字段
  - 数据：正在通信的信息（有效载荷）

**重要主题：**

- 控制 vs. 消息
  - 带内，带外
- 集中 vs. 分散
- 无状态 vs. 有状态
- 可靠 vs. 不可靠的邮件传输
- “网络边缘的复杂性”

## 课后思考题

- 复习题 2、5、10、11、16、20、21、23
- 习 题 1、4~11、17、22~24、26

# 作 业

## ■ 习题

□ 4、7、9