

# Chapter 2

## Application Layer

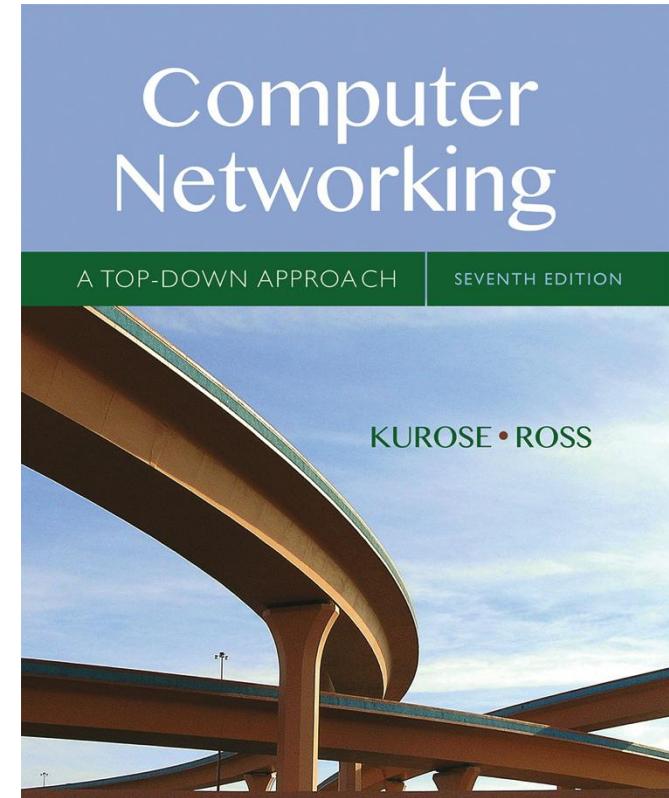
### A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2016  
© J.F Kurose and K.W. Ross, All Rights Reserved



*Computer  
Networking: A Top  
Down Approach*

7<sup>th</sup> edition  
Jim Kurose, Keith Ross  
Pearson/Addison Wesley  
April 2016

# Chapter 2: Application Layer

## Our goals:

- r 学习网络应用协议的原理及实现方面的知识
- r 几个重要的概念：
  - ❖ 客户-服务器模式
  - ❖ 对等模式
  - ❖ 传输层服务
  - ❖ 进程与传输层接口
- r 几个流行的应用层协议：
  - ❖ HTTP
  - ❖ FTP
  - ❖ SMTP / POP3 / IMAP
  - ❖ DNS
- r 开发网络应用程序：
  - ❖ socket API

# Chapter 2: outline

2.1 principles of  
network  
applications

2.2 Web and HTTP  
补充: File Transfer  
and FTP

2.3 electronic mail  
❖ SMTP, POP3,  
IMAP

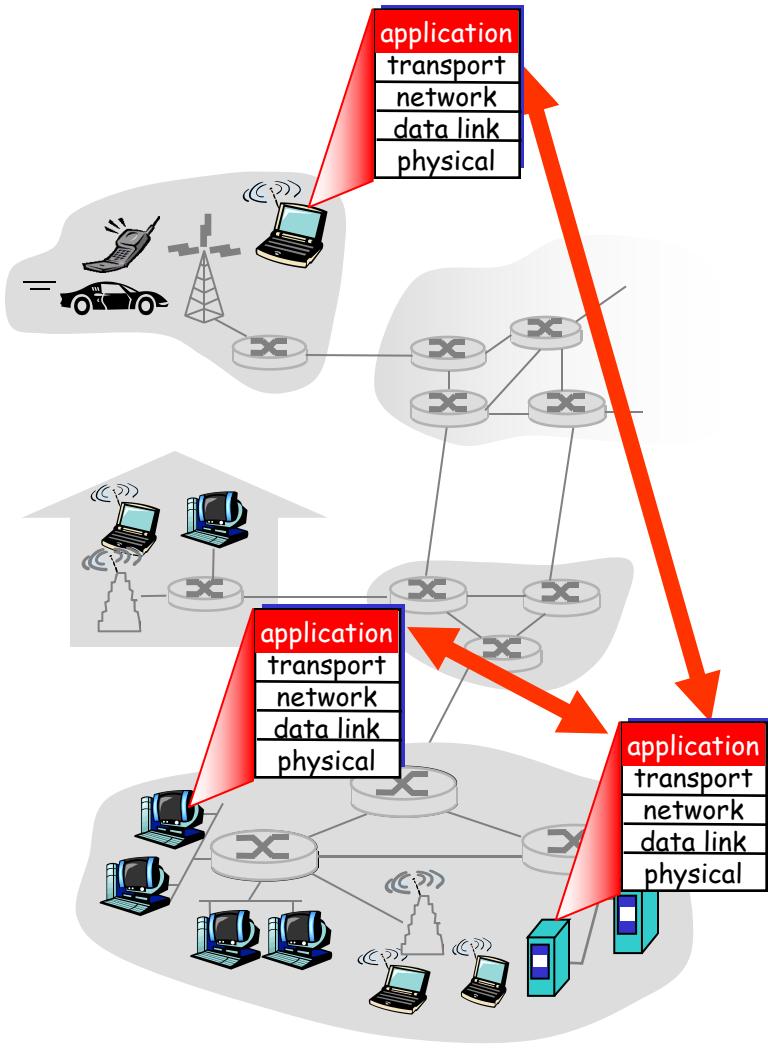
2.4 DNS

2.5 P2P applications  
2.6 video streaming  
and content  
distribution  
networks (CDNs)

2.7 socket  
programming with  
UDP and TCP

# 创建一个网络应用

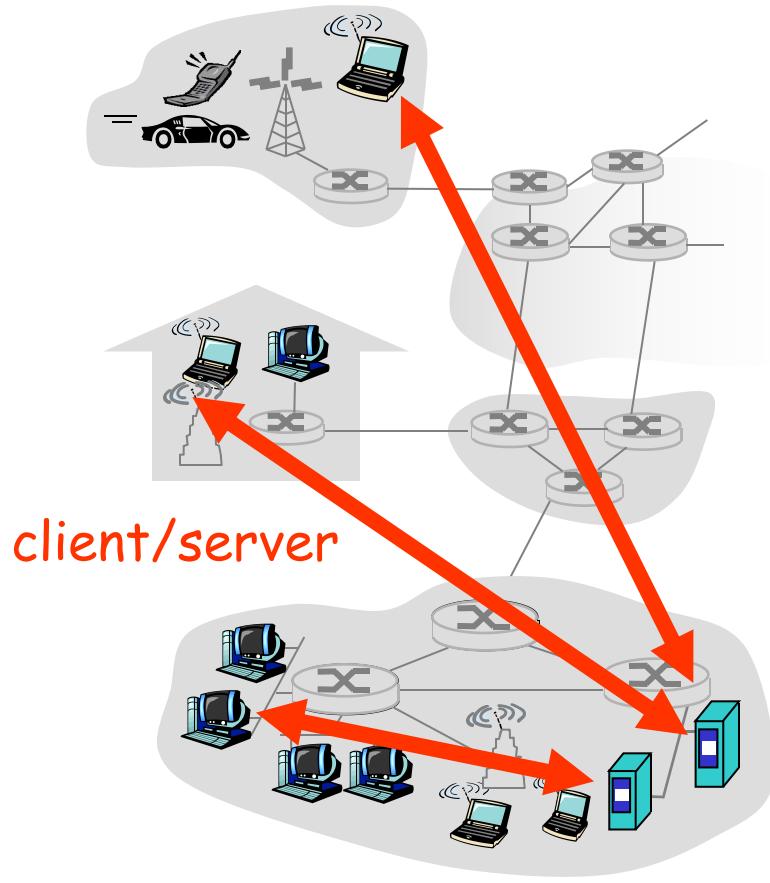
- r 创建一个网络应用的核心，是编写一个分布式程序，使其可以：
  - ❖ 运行在不同的端系统上，并能通过网络相互通信
  - ❖ 例如，**web**服务器软件与浏览器软件
- r 应用程序只运行在终端上：
  - ❖ 应程序员不需要为网络核心设备编写程序
- r “只在端系统上开发应用”是一个非常重要的设计决定：
  - ❖ 极大地方便了网络应用的开发和快速部署



## 2.1.1 网络应用架构

- r 在开发一个应用程序之前，首先要决定采用什么样的网络应用架构
- r 网络应用架构规定了在各个端系统上组织应用程序的方法
- r 目前有两种主流的网络应用架构：
  - ❖ 客户-服务器架构 (**Client-server**)
  - ❖ 对等架构 (**Peer-to-peer , P2P**)

# 客户-服务器架构



服务器 (**server**) :

- ❖ 有一台总是在线的主机，上面运行着服务器程序(**server**)
- ❖ 服务器主机(**server machine**)具有永久的、众所周知的地址

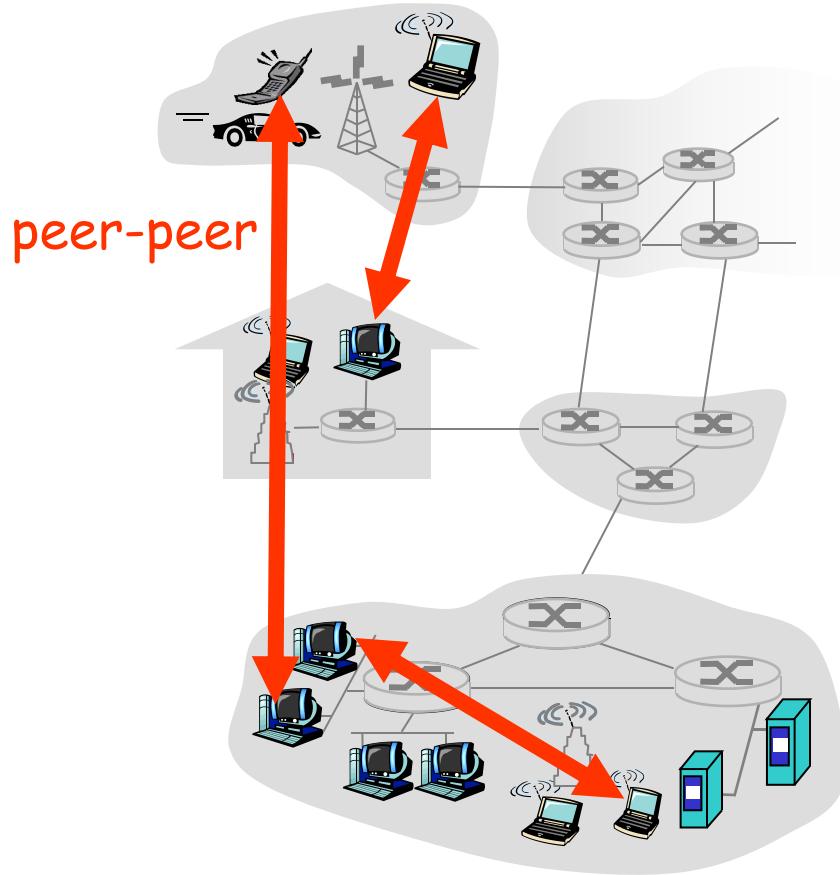
客户 (**client**) :

- ❖ 用户终端上运行一个客户程序(**client**)，需要时与服务器程序通信，请求服务
- ❖ 客户机 (**client machine**) 使用动态地址，通常不会总是在线

客户只与服务器通信，客户之间不通信

# P2P架构

- r 没有总是在线的服务器主机
- r 任意一对端系统（对等方）可以直接通信
- r 对等方多为用户自己的计算机，使用动态地址
- r 每个对等方既可请求服务，也可提供服务
- r 典型的P2P应用：
  - ❖ BT、迅雷
  - ❖ Skype
  - ❖ PPLive



# 两种架构的比较

## 客户-服务器架构:

- r 资源集中:
  - ❖ 资源（服务）只在某些固定的终端上提供
- r 优点:
  - ❖ 资源发现简单
- r 缺点:
  - ❖ 集中式计算带来的问题，如服务端扩容压力、网络流量不均衡、响应延迟长

## P2P架构:

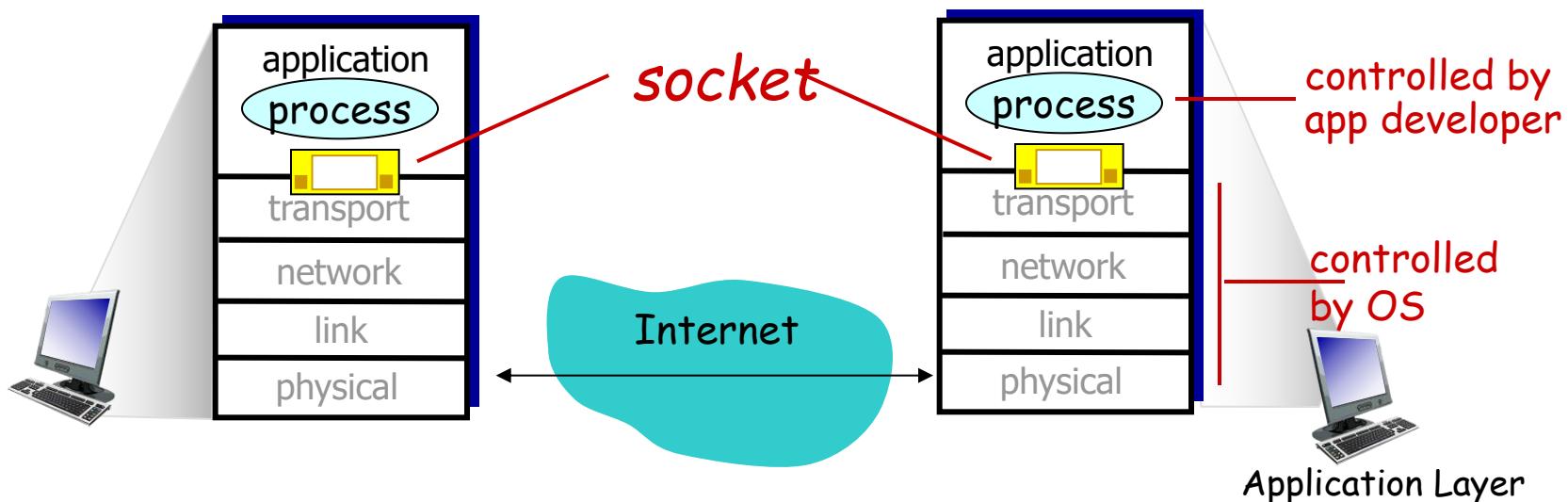
- r 资源分散:
  - ❖ 任何终端都可以提供资源（服务）
- r 优点:
  - ❖ 易于扩容、均衡网络流量
- r 缺点:
  - ❖ 资源发现困难，社会问题（版权、安全性等）

## 2.1.2 不同终端上的进程通信

- r **进程**: 主机上运行的程序
- r 在分布式应用中，不同终端上的进程需要通信
- r 进程通信的方法:
  - ❖ 同一个主机内: 进程间通信机制 (**OS**提供)
  - ❖ 在不同主机上: 通过交换报文进行通信
- r 在一次确定的通信会话中，总能标示一方为客户进程，另一方为服务器进程:
  - ❖ **客户进程**: 主动发起请求的进程
  - ❖ **服务器进程**: 接收请求的进程

# 进程与网络的接口：套接字(socket)

- ✓ 进程如何将报文送入网络，又如何从网络接收报文？
- ✓ 设想在应用程序和网络之间存在一扇门（套接字）：
  - ❖ 发送报文：发送进程将报文推到门外
  - ❖ 门外的运输设施（因特网）将报文送到接收进程的门口
  - ❖ 接收报文：接收进程打开门，即可收到报文
- ✓ 套接字是应用层和传输层的接口，也是应用程序和网络之间的API



# 进程如何标识自己：进程编址

- r 每个进程需要一个**标识**，以便其它进程能够找到它
- r **Q:** 可以用进程所在主机的地址来标识进程吗？
- r **A:** 不能，因为同一个主机上可能运行着许多进程
- r **端口号：**用于区分同一个主机上的不同进程
- r **进程标识包括：**
  - ❖ 主机地址
  - ❖ 与该进程关联的**端口号**
- r 端口号的例子：
  - ❖ 众所周知的端口号分配给服务器，成为服务的标识
  - ❖ 比如，**HTTP server**使用端口**80**，**Mail server**使用端口**25**

## 2.1.3 应用需要什么样的传输服务？

在创建一个应用程序时，开发者需要选定一种传输服务

### Data integrity

- r 有些应用（如音视频）可以容忍一定程度的数据丢失
- r 有些应用（如文件传输）要求完全可靠的数据传输

### Throughput

- r 有些应用（如多媒体）要求保证最低可用带宽
- r 有些应用（称弹性应用）可以适应各种可能的带宽

### Timing

- r 有些应用（如网络电话，交互式网络游戏）要求延迟保证
- r 有些应用（如邮件传输）对延迟不敏感

### Security

- r 加密，数据完整性，.....

## 2.1.4 因特网能够提供的传输服务

因特网提供**2**种传输服务，分别用**TCP**和**UDP**协议实现

### TCP service:

- r 面向连接：保证传输顺序
- r 可靠传输：不出错
- r 流量控制：发送进程不会“压垮”接收进程
- r 拥塞控制：网络超载时抑制发送进程
- r **不提供：**及时性，最低带宽保证，安全性

### UDP service:

- r 通过因特网接收和发送报文
- r **不提供：**顺序保证，可靠传输，流量控制，拥塞控制，及时性，最低带宽保证，安全性

## 2.1.5 应用层协议

应用进程之间交换的报文是什么样的？

应用层协议定义：

- r 报文类型：
  - ❖ e.g., request, response
- r 报文语法：
  - ❖ 报文中的字段及其描述
- r 报文语义
  - ❖ 各字段中信息的含义
- r 发送/响应报文应遵循的规则

公共域协议：

- r 在RFC文档中定义
- r 允许互操作
- r e.g., HTTP, SMTP

专用协议：

- r e.g., Skype

# 小结

- r 为创建一个新的网络应用，需要：
  - ❖ 选择一种网络应用架构：客户-服务器 or P2P
  - ❖ 选择一种网络服务：TCP or UDP
  - ❖ 确定一个端口号
  - ❖ 定义应用层协议
  - ❖ 编写客户程序和服务器程序（调用套接字接口）
- r 网络应用和应用层协议：
  - ❖ 应用层协议只是网络应用的一部分
  - ❖ 网络应用还包括客户程序、服务器程序等
- r 要求掌握的概念：
  - ❖ 客户-服务器模型
  - ❖ 网络应用编程接口

## Chapter 2: outline

2.1 principles of  
network  
applications

**2.2 Web and HTTP**

补充: File Transfer  
and FTP

2.3 electronic mail

- ❖ SMTP, POP3,  
IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming  
and content  
distribution  
networks (CDNs)

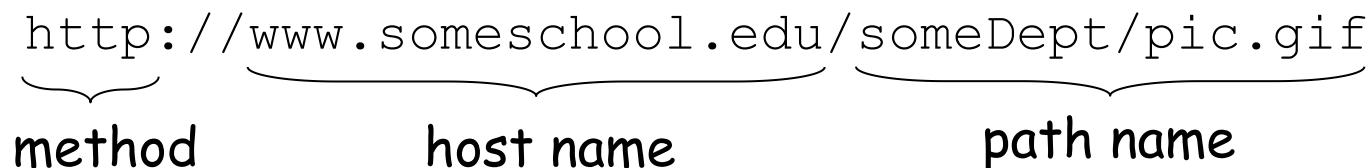
2.7 socket programming  
with UDP and TCP

# Web应用画像

- r 应用层资源：网页（**web pages**）
- r 网络应用架构：客户-服务器架构
  - ❖ 客户：浏览器
  - ❖ 服务器：**web**服务器
- r 要求的网络服务：TCP服务
- r 端口号：80
- r 应用层协议：HTTP（Hyper Text Transfer Protocol）

# 一些术语

- r **Web page**由一些对象（**object**）组成
- r 对象简单来说就是**文件**，可以是**HTML**文件、图像、**Java**小程序、音频文件，...
- r 每个对象通过一个**URL (Uniform Resource Locator)** 获取，例如：

http:///someDept/pic.gif  
method                  host name                  path name

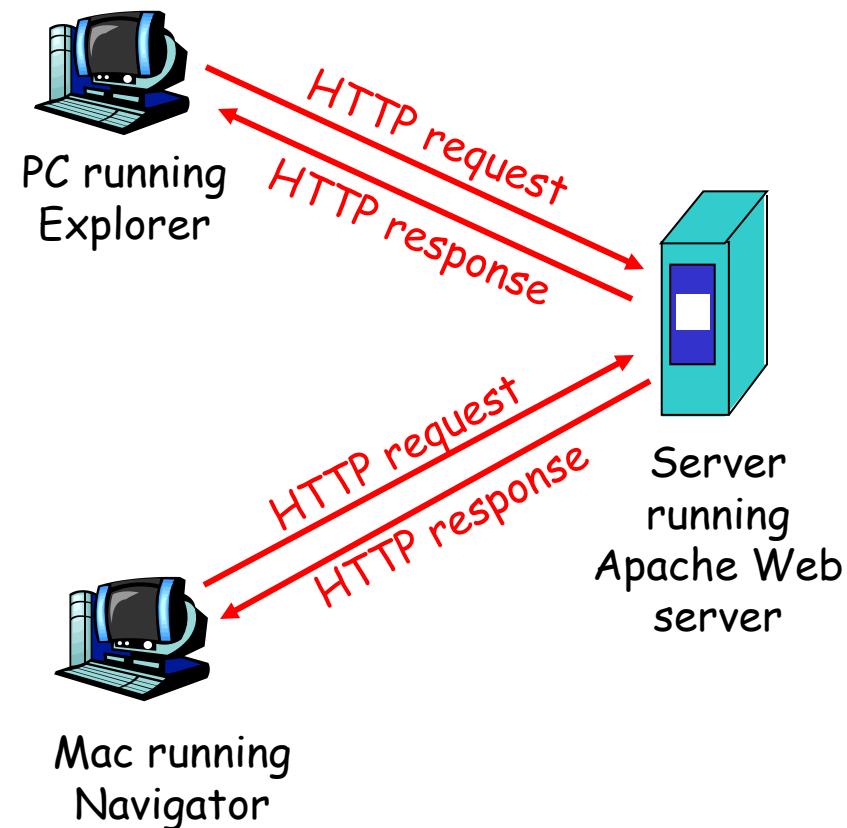
- r **HTML (Hyper Text Markup Language)** 是一种格式语言，用于描述页的内容和显示方式
- r **HTML**文件是用**HTML**语言编写的**超文本文件**，需要通过专门的浏览器软件进行展示
- r **Web**页通常包含一个**HTML基本文件**和若干引用对象，每个引用对象在文档中用其**URL**给出

## 一些术语（续）

- r 超文本文件：
  - ❖ 包含超链接的文件，其中的文字包含有可以链接到文档中其他位置或者其它文档的连结，允许从当前阅读位置直接切换到超文本连结所指向的位置
- r 超文本（Hyper text）：
  - ❖ 一种信息组织方式，通过超链接将各种不同空间的文字信息组织成网状文本

## 2.2.1 超文本传输协议--HTTP 概述

- r Web采用客户-服务器模式
  - ❖ *client*: 浏览器请求、接收和显示web对象
  - ❖ *server*: Web服务器应客户请求发送对象
- r HTTP协议: 定义了浏览器和 web服务器之间的通信规则
- r HTTP 1.0 (RFC 1945) 和 HTTP 1.1 (RFC 2068)



# HTTP 使用 TCP服务

- r 客户发起到服务器 80 端口的 **TCP** 连接（**客户端创建一个套接字**）
- r 服务器接受来自客户的 **TCP**连接（**服务器端创建一个套接字**）
- r 浏览器和服务器交换 **HTTP报文**（**通过各自的套接字**）
- r 关闭**TCP** 连接（**关闭各自的套接字**）

**HTTP是“无状态的”**

- r 服务器不保存有关客户请求的任何信息

*Protocols that maintain  
"state" are complex!*

- aside*
- r past history (state) must be maintained
- r if server/client crashes, their views of "state" may be inconsistent, must be reconciled

## 2.2.2 非持久连接和持久连接

### 非持久 HTTP

- r 在一个**TCP**连接上最多发送一个对象
- r **HTTP/1.0** 使用非持久连接

### 持久 HTTP

- r 在一个**TCP**连接上可以发送多个对象
- r **HTTP/1.1** 缺省使用持久连接

# 非持久 HTTP

假设用户输入以下URL:

http://www.someSchool.edu/someDepartment/home.index

(包含文本及  
10个Jpeg图像)

- 1a. HTTP客户发起到以下HTTP服务器进程的连接  
www.someSchool.edu on port 80  
(建立TCP连接)
- 1b. www.someSchool.edu上的HTTP服务器在端口80等待TCP连接，接受连接，并通知客户。
2. HTTP客户将一个HTTP请求报文（包含URL）写入它的套接字，报文指示希望获取以下对象：  
someDepartment/home.index  
(请求对象)
3. HTTP服务器接收请求报文，构造包含所请求对象的响应报文，将报文写入它的套接字中。

time  
↓

## 非持久HTTP (续)

time  
↓

5. HTTP 客户接收包含HTML文件的响应报文，显示HTML文件，解析文件发现有10个引用的jpeg对象

4. HTTP 服务器关闭TCP连接

6. 对于每个jpeg对象，重复步骤 1-5

# 非持久HTTP的响应时间

## RTT (Round-Trip Time):

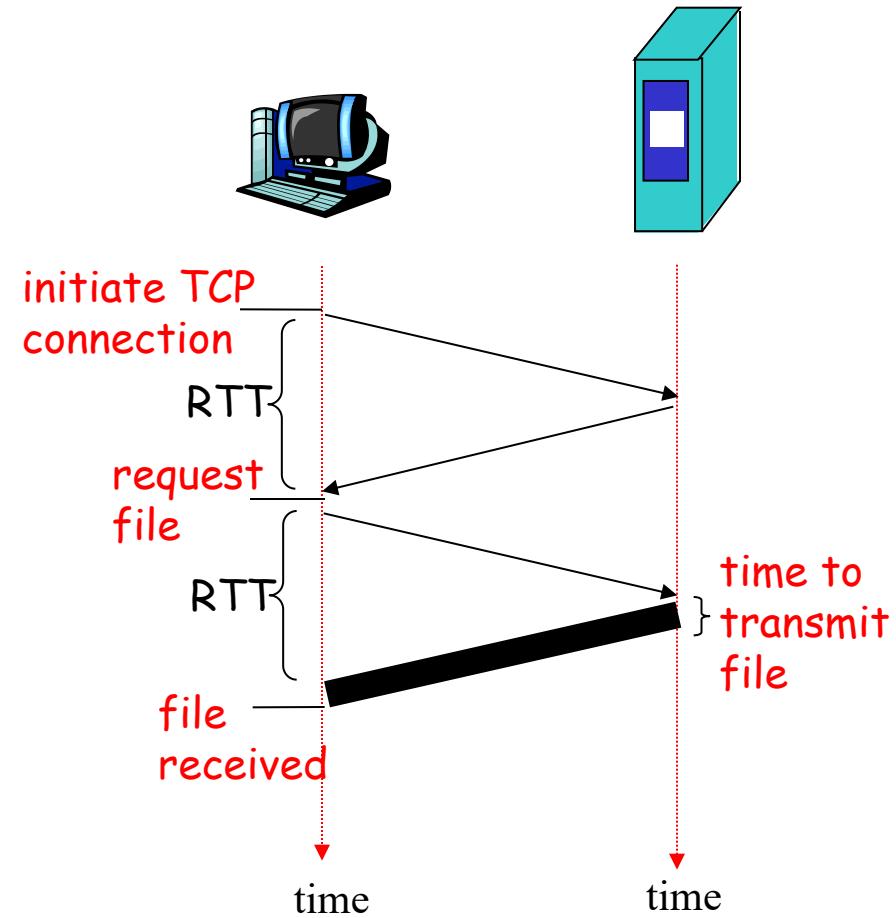
一个小分组从客户发送到服务器，再返回客户的时间

## Response time:

- r 建立TCP连接用时一个RTT
- r 发送HTTP请求至收到响应的前几个字节用时一个RTT
- r 传输文件的时间

## 请求一个网页的时间:

- r 请求一个对象用时 $2\text{RTT}$ （忽略对象传输时间）
- r 请求一个完整的网页用时 $22\text{RTT}$ （11个对象）



# 持久 HTTP

## 非持久HTTP 的问题:

- r 获取每个对象需要**2个RTT**
- r 每个**TCP**连接需要消耗操作系统资源
- r 浏览器需要打开多个**TCP**连接来获取一个网页

## 持久 HTTP

- r 服务器在发送响应后保持连接
- r 同一对客户-服务器之间的后续**HTTP**报文可以在该连接上  
传输

## 无流水线方式:

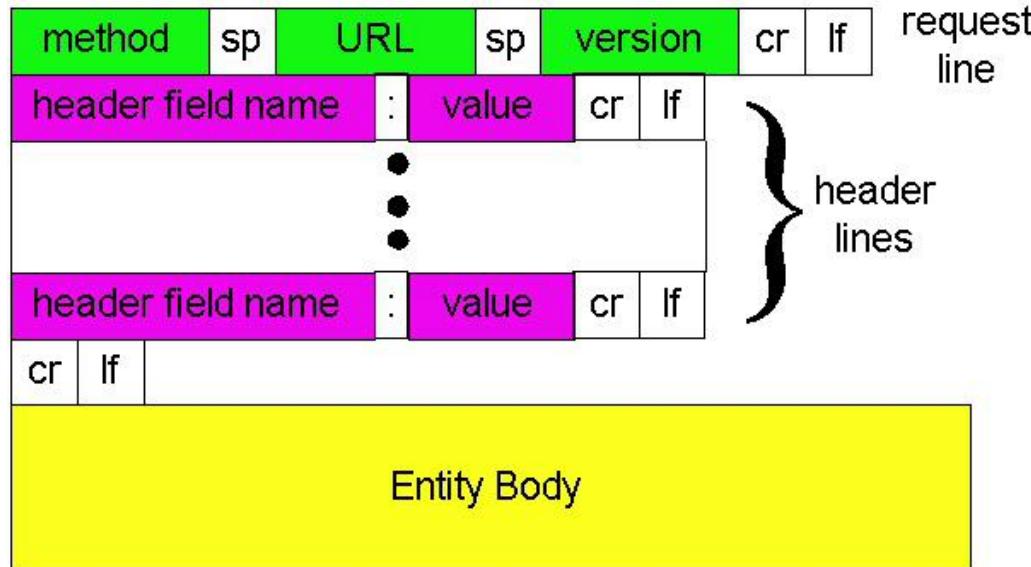
- r 客户仅当收到前一个响应后再发送新的请求
- r 请求每个对象用时**1个RTT**
- r 请求一个网页用时**12RTT**

## 流水线方式:

- r **HTTP/1.1**缺省使用该方式
- r 客户每解析到一个引用对象就可以发送请求
- r 可在一个**RTT**时间内请求所有引用对象
- r 请求一个网页用时约**3RTT**

## 2.2.3 HTTP 报文格式

- r 两类HTTP报文:
  - ❖ 请求报文: 客户发往服务器
  - ❖ 响应报文: 服务器发往客户
- r HTTP报文由**ASCII**文本构成
- r **HTTP请求报文:**



# HTTP请求报文

r 报头包括：

请求行

GET /somedir/page.html HTTP/1.1

Host: www.someschool.edu

User-agent: Mozilla/4.0

Connection: close

Accept-language: fr

首部行

一个额外的回车换  
行表示报头结束

(extra carriage return, line feed)

# 上传表单输入

## Post 方法:

- r Web 页通常包含表单输入
- r 输入的表单内容放在报文体中上传到服务器

## URL 方法:

- r 使用 GET 方法
- r 输入内容放在请求行的 URL 字段中上传, 如:

www.somesite.com/animalsearch?monkeys&banana

# HTTP方法

## HTTP/1.0

- r GET
- r POST
- r HEAD
  - ❖ 要求服务器不返回对象，只用一个报文头响应（实体为空），常用于故障跟踪。

## HTTP/1.1

- r GET, POST, HEAD
- r PUT
  - ❖ 将文件放在报文实体中，传到URL字段指定的路径
- r DELETE
  - ❖ 删除URL字段指示的文件

# HTTP 响应报文

状态行

HTTP/1.1 200 OK

首部行

Connection: close

Date: Thu, 06 Aug 1998 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 1998 .....

Content-Length: 6821

Content-Type: text/html

数据，如请求的  
HTML文件

data data data data data ...

# HTTP 响应状态代码

A few sample codes:

## **200 OK**

- ❖ request succeeded, requested object later in this message

## **301 Moved Permanently**

- ❖ requested object moved, new location specified later in this message (Location:)

## **400 Bad Request**

- ❖ request message not understood by server

## **404 Not Found**

- ❖ requested document not found on this server

## **505 HTTP Version Not Supported**

## 2.2.4 用户与服务器交互: cookie

许多大型的Web网站都使用cookie

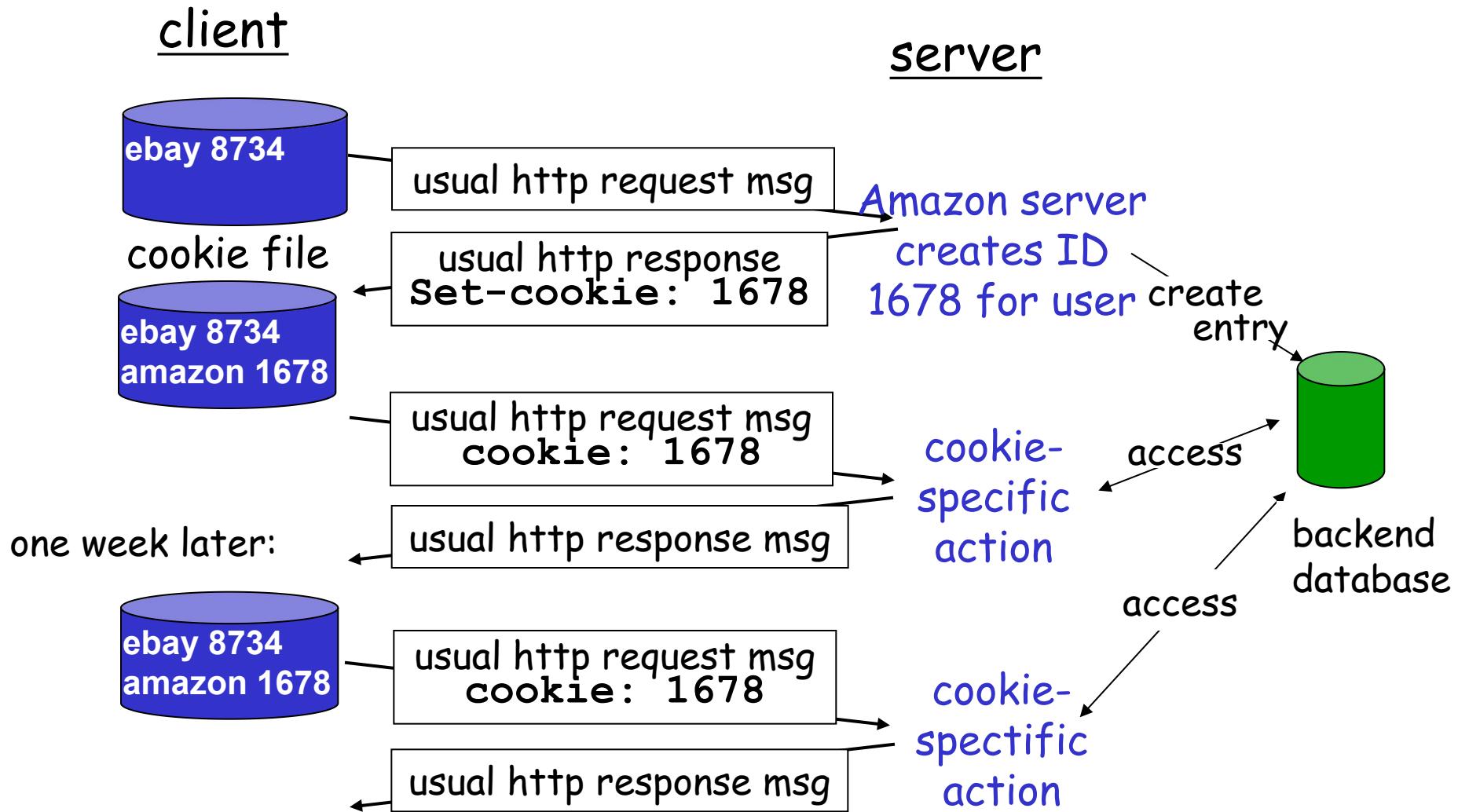
### Four components:

- 1) HTTP响应报文中的cookie首部行
- 2) HTTP请求报文中的cookie首部行
- 3) 保存在用户主机的cookie文件, 由用户的浏览器管理
- 4) Web网站的后端数据库

### Example:

- r Susan 总是从她的PC机上网
- r 她第一次访问某个电子商务网站
- r 当HTTP请求报文到达该网站时, 网站为其创建:
  - ❖ 一个ID
  - ❖ 在后端数据库中为该ID建立一个表项

# Cookies: 保存“状态”



# Cookies (continued)

## What cookies can bring:

- r authorization
- r shopping carts
- r recommendations
- r user session state  
(Web e-mail)

aside

## Cookies 和隐私:

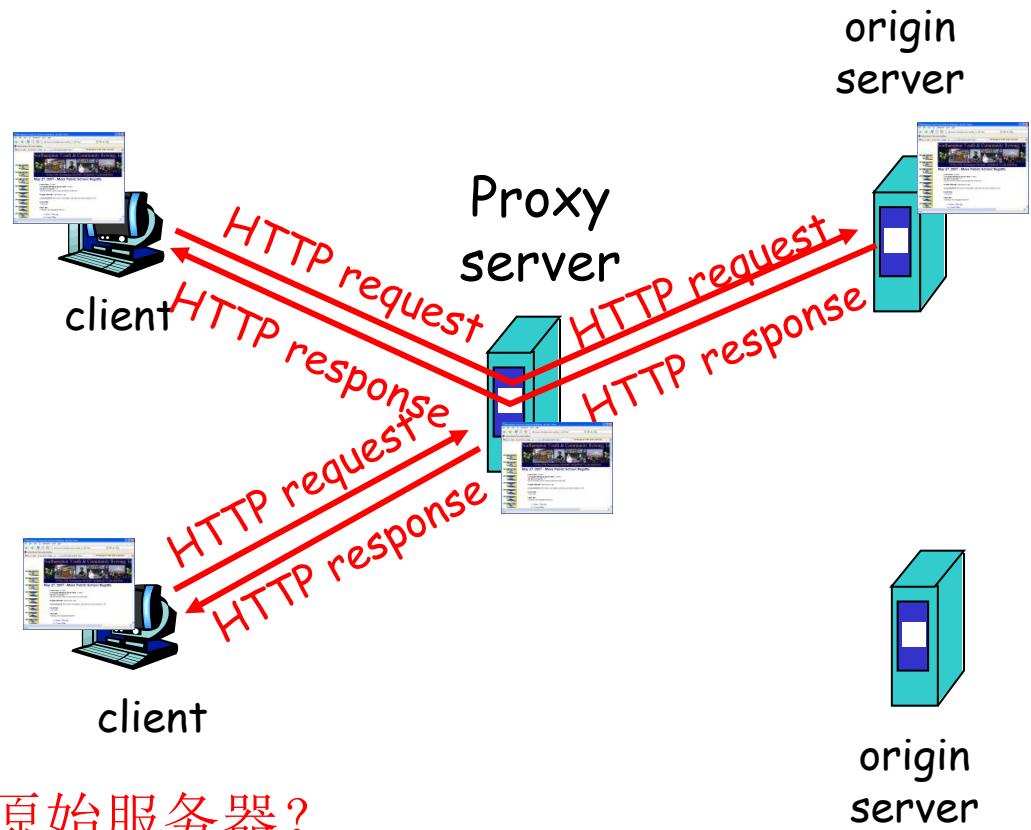
- r Cookies 允许网站收集用户的大量信息
- r you may supply name and e-mail to sites

## Where to keep "state":

- r 服务端：信息保存在服务端的后端数据库，返回ID给客户
- r 客户端：信息发回客户端，保存在cookie文件中，并随请求报文发送给服务器

## 2.2.5 Web缓存（代理服务器）

- r 代理服务器：代表原始服务器满足HTTP请求的网络实体，保存最近请求过的对象的拷贝。
- r 用户设置浏览器：所有HTTP请求首先发往web缓存
- r 浏览器将HTTP请求发送给web缓存：
  - ❖ 对象在web缓存中： web缓存返回对象
  - ❖ 对象不在web缓存中： web缓存从原始服务器获取对象，缓存在本地，然后返回给客户



Q: web缓存如何知道对象的原始服务器？

A: HTTP请求头中的**host**首部行指出了原始服务器

# More about Web caching

- r Web cache既是服务  
器又是客户：
  - ❖ 对于浏览器来说是服  
务器
  - ❖ 对于原始服务器来说  
是客户
- r Web cache通常由ISP  
提供，多级ISP可能形  
成多级web缓存

## Why Web caching?

- r 减少客户请求的响应时间
- r 减少机构接入链路上的流  
量

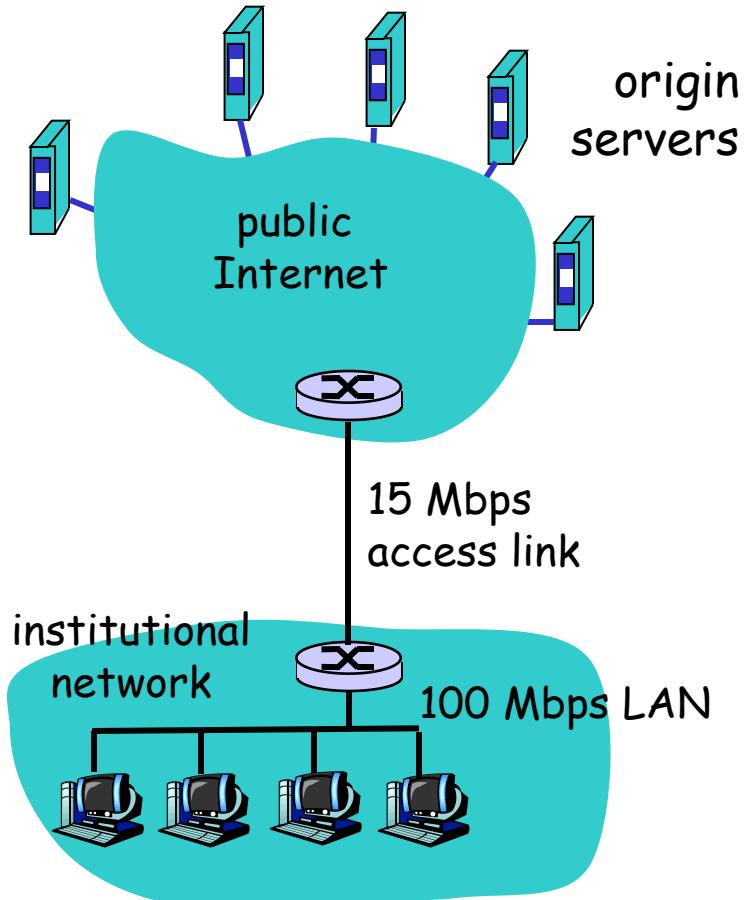
# Caching example

## Assumptions

- r 平均对象长度 = 1Mb
- r 从机构浏览器到原始服务器的平均请求速率 = 15/sec
- r 从接入路由器到原始服务器的来回延迟 (RTT) = 2 sec

## Consequences

- r LAN利用率 = 15%
- r 接入链路的利用率 = 100%
- r  $\text{total delay} = \text{Internet delay} + \text{access delay} + \text{LAN delay}$   
= 2 sec + **minutes** + milliseconds



# Caching example (cont)

## possible solution

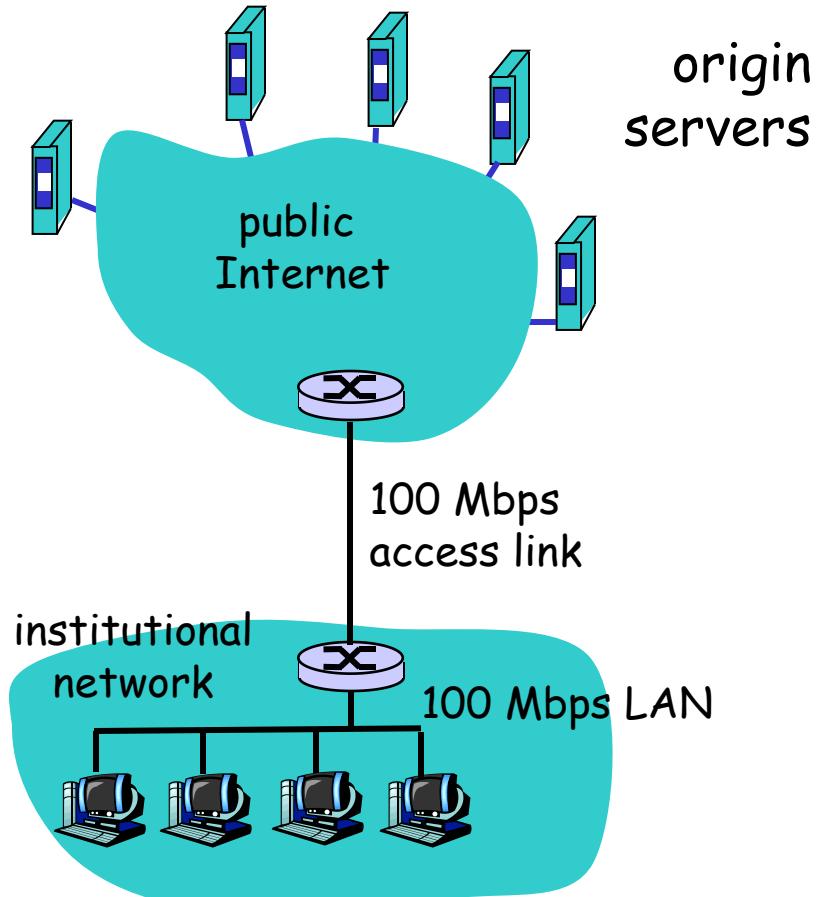
- r 提高接入链路带宽至 100Mbps

## consequence

- r LAN利用率 = 15%
- r 接入链路利用率 = 15%
- r Total delay = Internet delay + access delay + LAN delay

$$= 2 \text{ sec} + \text{msecs} + \text{msecs}$$

- r 链路升级的代价高昂！



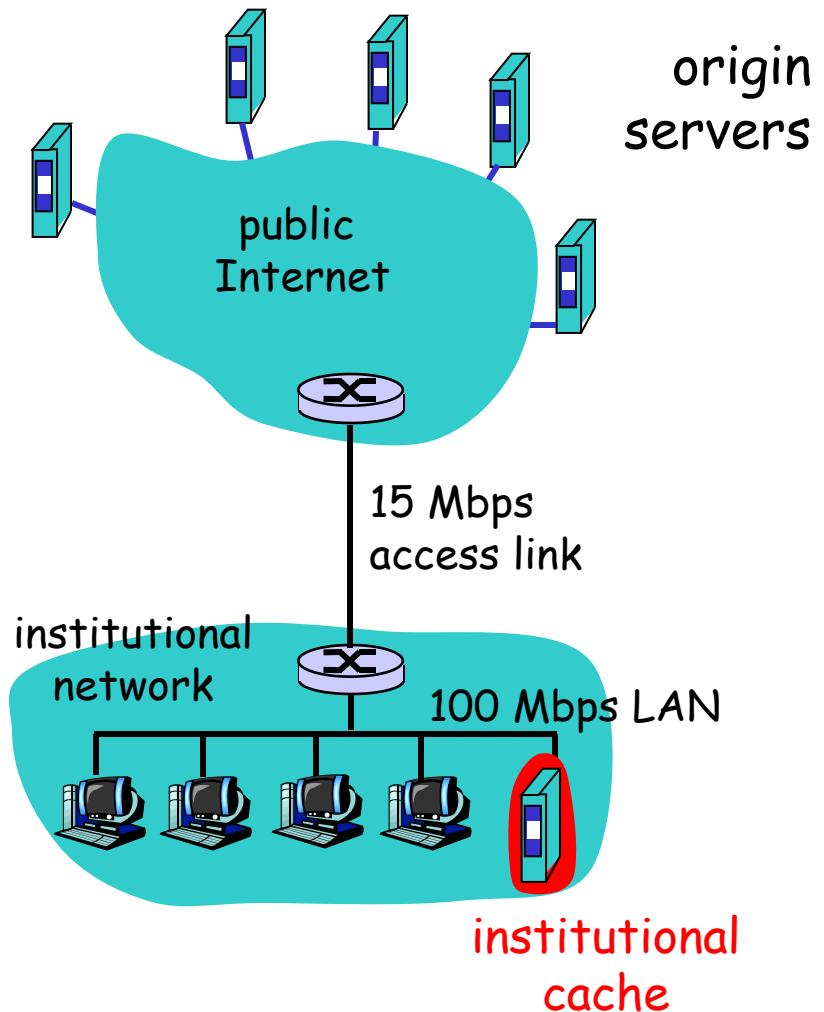
# Caching example (cont)

## possible solution:

- r 安装web cache
- r 假设cache命中率为 0.4

## consequence

- r 40%的请求可被立即满足
- r 60%的请求被原始服务器满足
- r 接入链路利用率为60%，延迟为  
~10msecs
- r total avg delay = Internet  
delay + access delay + LAN  
delay =  $0.6 * (\sim 2.01 \text{ secs}) +$   
 $0.4 * (\sim \text{msecs}) = \sim 1.2 \text{ secs}$
- r 比升级链路的延迟还要低！



## 2.2.6 条件 GET

代理服务器如何发现缓存的对象是不是最新的？

- r web cache 在发送的 GET 报文中包含以下首部行

If-modified-since:  
<date>

- r Server:

❖ 若发现对象无更新，响应报文中不包含对象：

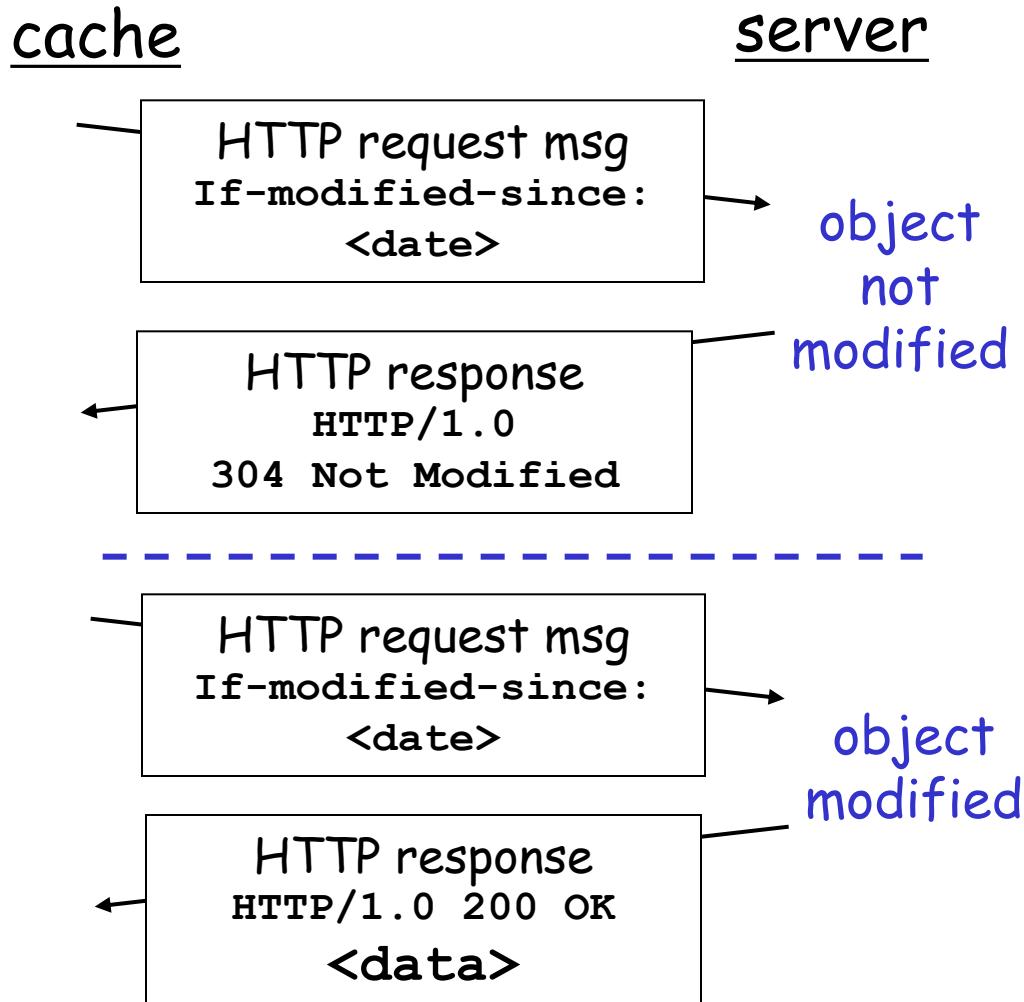
HTTP/1.0 304 Not Modified

❖ 若发现对象有更新，响应报文中包含对象

- r 对象修改时间在哪里提供？

❖ HTTP 响应报文中的 Last-modified 首部行

❖ Web 缓存将该信息拷贝到 if-modified-since 首部行



# 小结

- r 因特网上的每个资源（对象）都可以通过**URL**访问
- r HTTP使用TCP连接
  - ❖ 非持续HTTP
  - ❖ 持续HTTP
- r 客户与服务器交互：
  - ❖ 报文请求/响应
- r HTTP服务器是无状态的，利用**cookie**技术可以保存用户状态
- r 使用**web**缓存提高请求-响应速度：
  - ❖ 使用条件**get**获得最新的对象副本

# 理解http及web技术的发展脉络

- r 为什么HTTP最初设计为是非持久连接、无状态的？
- r 后来为什么引入持久连接？
- r 为什么引入cookie技术，它带来什么问题？
- r 为什么引入web缓存技术，它带来什么问题，如何解决？

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充: File Transfer and  
FTP

2.3 electronic mail

- ❖ SMTP, POP3, IMAP

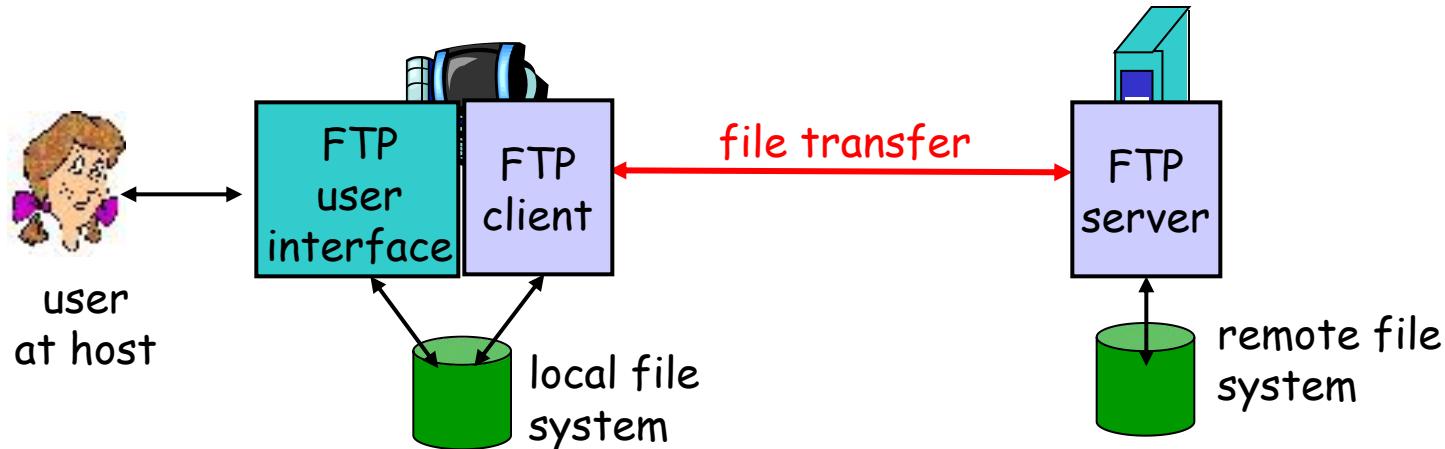
2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# 文件传输应用画像



- r 应用层资源：文件
- r 网络应用架构：客户-服务器
- r 使用传输服务：TCP
- r 端口号：21、20
- r 应用层协议：FTP
  - ❖ 使用命令/响应交互（不是像HTTP那样使用报文交互）
  - ❖ 通常情况下，用户通过FTP用户代理与文件服务器交互

# FTP 用户代理

8uftp - 已连接到 staff.ustc.edu.cn

文件(F) 编辑(E) 传输(T) 视图(V) 队列(Q) 服务器(S) 工具(T) 帮助(H)

地址: staff.ustc.edu.cn 用户名: bhua 密码: \*\*\*\*\* 端口: 21 连接(C)

本地: USTC

名称 /	大小	类型	修改时间
上层目录			
课外拓展		文件夹	2017/9/12 10:10
图片		文件夹	2017/9/12 11:47
~第一章_2017.docx	162	Microsoft Off...	2017/9/12 10:53
2017-01114401.xls	17 KB	Microsoft Off...	2017/9/11 10:47
Chapter1_2017.ppt	5 MB	Microsoft Off...	2017/9/6 9:16
Chapter2_2017.ppt	3 MB	Microsoft Off...	2017/9/12 13:09
Introduction.pptx	1 MB	Microsoft Off...	2017/9/4 10:22
Test_1.docx	16 KB	Microsoft Off...	2017/9/7 14:22
第二章_2017.docx	40 KB	Microsoft Off...	2017/9/12 13:09
第一章_2017.docx	91 KB	Microsoft Off...	2017/8/28 11:29
开场白.docx	20 KB	Microsoft Off...	2017/8/21 16:52

远程: /public\_html/

名称 /	大小	类型	修改时间	权限	所属用户 / 组
上层目录					
advanced_network		文件夹	2012-11-8 13:41	drwxr-xr-x	4470 500
course_reference		文件夹	2013-9-4 13:41	drwxr-xr-x	4470 501
experiments		文件夹	2012-12-6 13:41	drwxr-xr-x	4470 501
image		文件夹	2008-7-25 13:41	drwxr-xr-x	4470 501
index.files		文件夹	2008-7-25 13:41	drwxr-xr-x	4470 501
intro_computer_2017		文件夹	2017-8-30 17:43	drwxr-xr-x	4470 500
Kurose_Labs		文件夹	2014-9-3 13:41	drwxr-xr-x	4470 500
network		文件夹	2011-12-23 17...	drwxr-xr-x	4470 100
network_2017		文件夹	2017-9-7 18:14	drwxr-xr-x	4470 500
network_algorithmic...		文件夹	2017-9-7 0:22	drwxr-xr-x	4470 500
network_foundation		文件夹	2010-9-6 0:22	drwxr-xr-x	4470 501
network_processor		文件夹	2009-12-17 0:22	drwxr-xr-x	4470 501
network_v2		文件夹	2008-7-25 0:22	drwxr-xr-x	4470 501
project_reference		文件夹	2015-2-2 0:22	drwxr-xr-x	4470 500
publications		文件夹	2017-7-21 14:49	drwxr-xr-x	4470 501
software_school		文件夹	2012-5-30 14:49	drwxr-xr-x	4470 500
temp.files		文件夹	2015-8-30 14:49	drwxr-xr-x	4470 501
00000886.gif	1 KB	GIF 文件	2007-4-4 0:1848	-rw-r--r--	4470 100
00000887.gif	989	GIF 文件	2007-4-4 0:1848	-rw-r--r--	4470 100
00000888.gif	2 KB	GIF 文件	2007-4-4 0:1848	-rw-r--r--	4470 100
00000889.gif	47 KB	GIF 文件	2015-8-30 0:18...	-rw-r--r--	4470 500
00000889.jpg	93 KB	JPG 文件	2009-3-26 0:18...	-rw-r--r--	4470 100
00000890.gif	20 KB	GIF 文件	2007-4-4 0:1848	-rw-r--r--	4470 100
index.htm	60 KB	360 se HTML...	2017-9-11 12:42	-rw-r--r--	4470 500
Quiz_4.doc	27 KB	Microsoft Off...	2012-4-11 13:41	-rw-r--r--	4470 500

选定一个 5296128 字节的文件。

17 个文件夹和 8 个文件, 大小: 259804 字节。

本地文件名[0/0 字节]) 大小 方向 远程文件名 主机

状态: 正在取得目录列表...

命令: CWD /public\_html/  
响应: 250 Directory successfully changed.

命令: FWD  
响应: 251 "/public\_html"

命令: PASV  
响应: 227 Entering Passive Mode (202,141,160,11,29,116)

命令: TYPE A  
响应: 200 Switching to ASCII mode.

命令: LIST  
响应: 150 Here comes the directory listing.  
响应: 226 Directory send OK.

状态: 成功取得目录列表

命令: FWD  
响应: 251 "/public\_html"

命令: DELE /public\_html/Chapter1\_2017.ppt  
响应: 250 Delete operation successful.

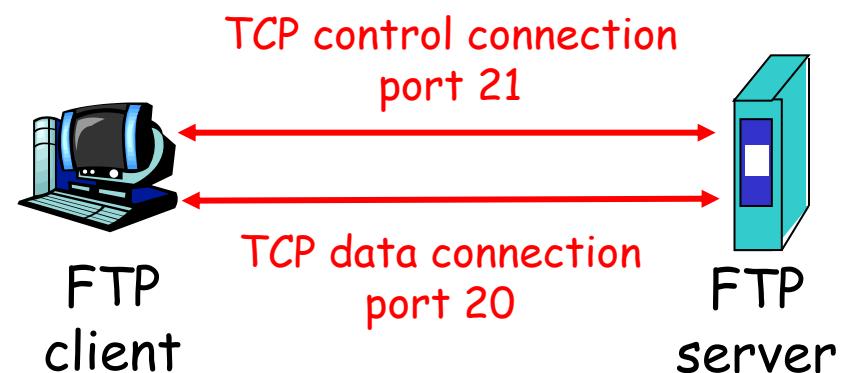
命令: FWD  
响应: 251 "/public\_html"

队列: 0 字节 13:36 2017/9/12

# 使用分离的控制连接和数据连接

## r 控制连接：

- ❖ 使用端口**21**, 传送客户命令和服务器响应
- ❖ 控制连接在整个会话期间一直保持

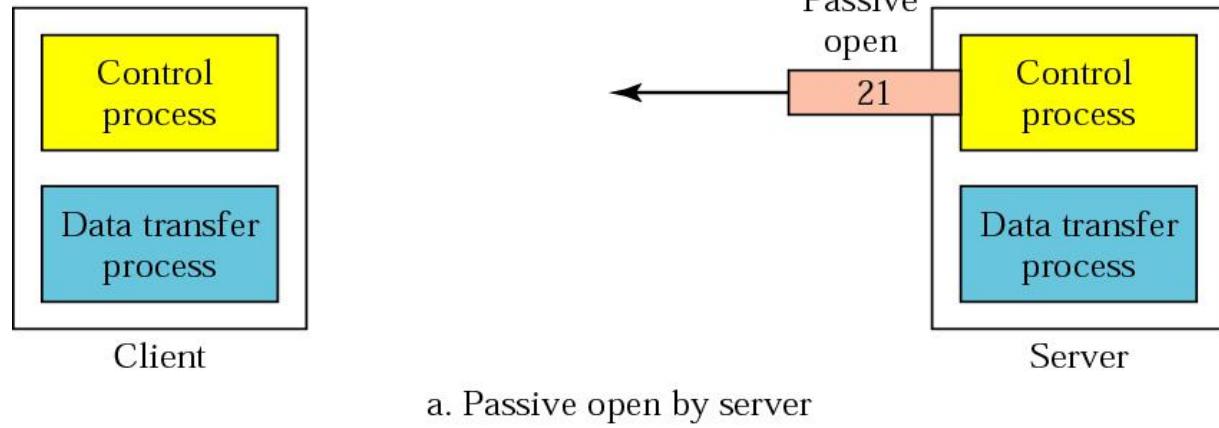


## r 数据连接：

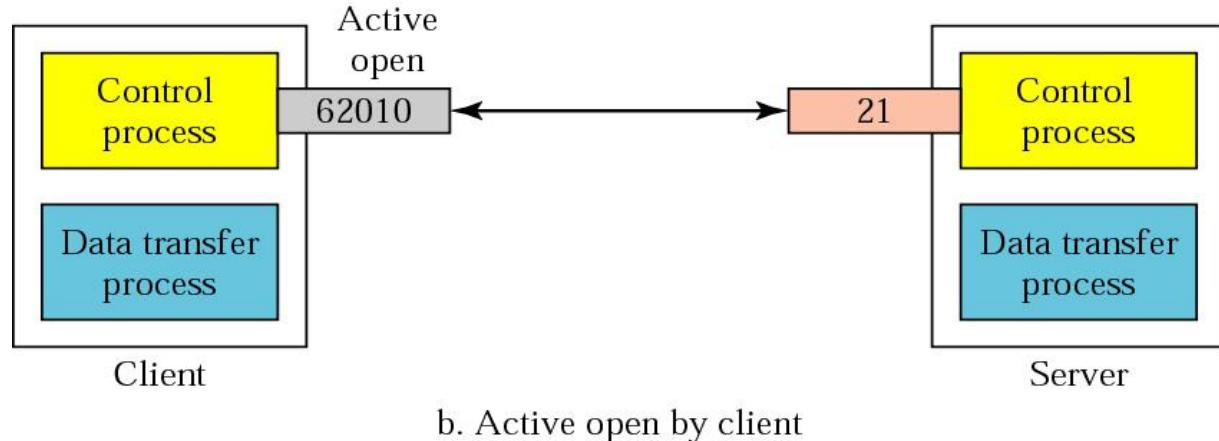
- ❖ 使用端口**20**, 传输文件
- ❖ 每个数据连接只传输一个文件, 发送方用关闭连接表示一个文件传输结束

# 建立控制连接

- 服务器在端口**21**等待客户

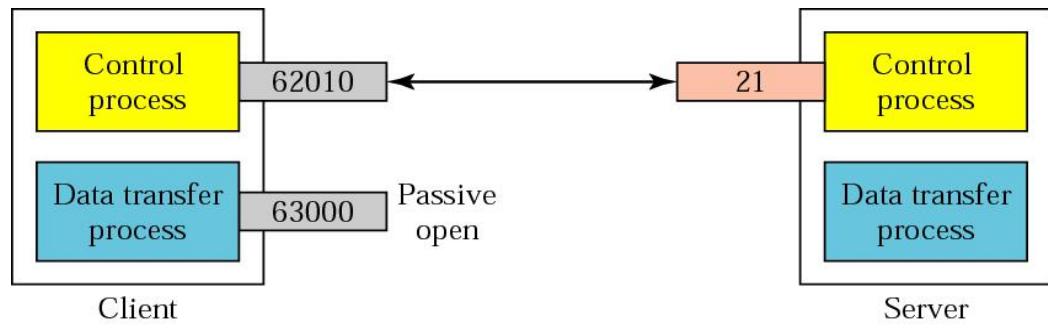


- 客户使用临时端口号建立连接

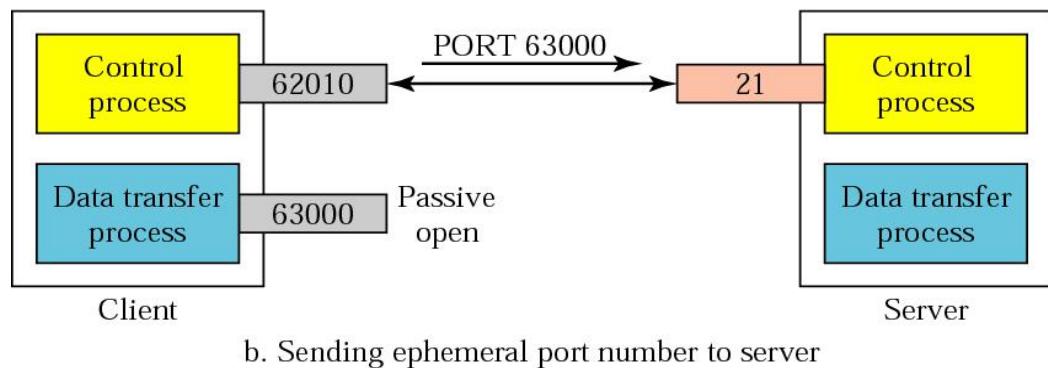


# 建立数据连接（主动模式）

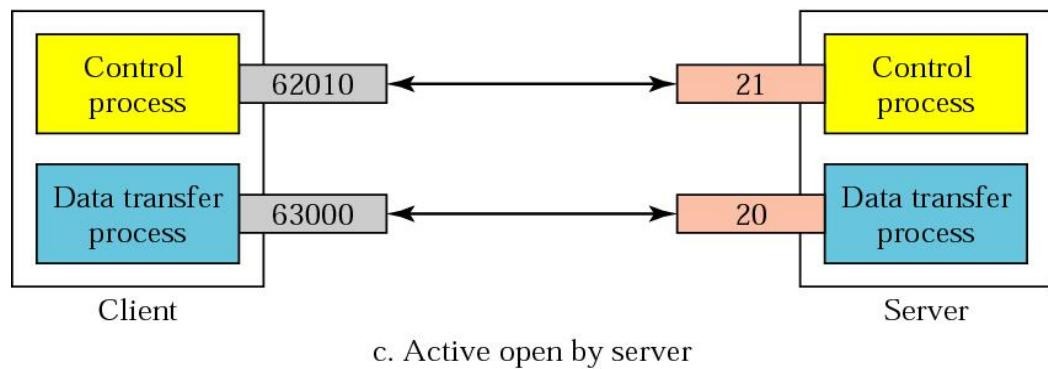
- 客户选择一个临时端口号，在该端口上等待服务器的连接请求
- 客户在控制连接上用**PORT**命令将临时端口号发送给服务器
- 服务器使用端口**20**与客户机给出的端口建立连接



a. Passive open by client



b. Sending ephemeral port number to server



c. Active open by server

# 有关控制连接与数据连接的问题

- r **Q:** 为什么将控制连接与数据连接分开?
- r **A:** 不会混淆数据与命令/响应，简化协议设计和实现；在传输文件的过程中可以继续执行其它的操作；便于控制传输过程（如客户可以随时终止传输）
  
- r **Q:** 为什么用关闭数据连接的方式结束文件传输？
- r **A:** 允许动态创建文件（不需预先告知文件的大小）

# FTP小结

- r 使用**TCP**协议，服务器端口号**21、20**
- r 采用命令/**响应**交互方式
- r 使用两条**TCP**连接：
  - ❖ 控制连接（端口**21**）：会话期间始终保持
  - ❖ 数据连接（端口**20**）：每条连接仅传输一个文件，且客户与服务器角色交换
- r 要求理解：
  - ❖ 为什么使用两条分开的连接，即为什么不在同一条连接上既传输命令/**响应**、又传输数据？
  - ❖ 为什么每条数据连接只传输一个文件？

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充: File Transfer and  
FTP

2.3 electronic mail

- ❖ SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

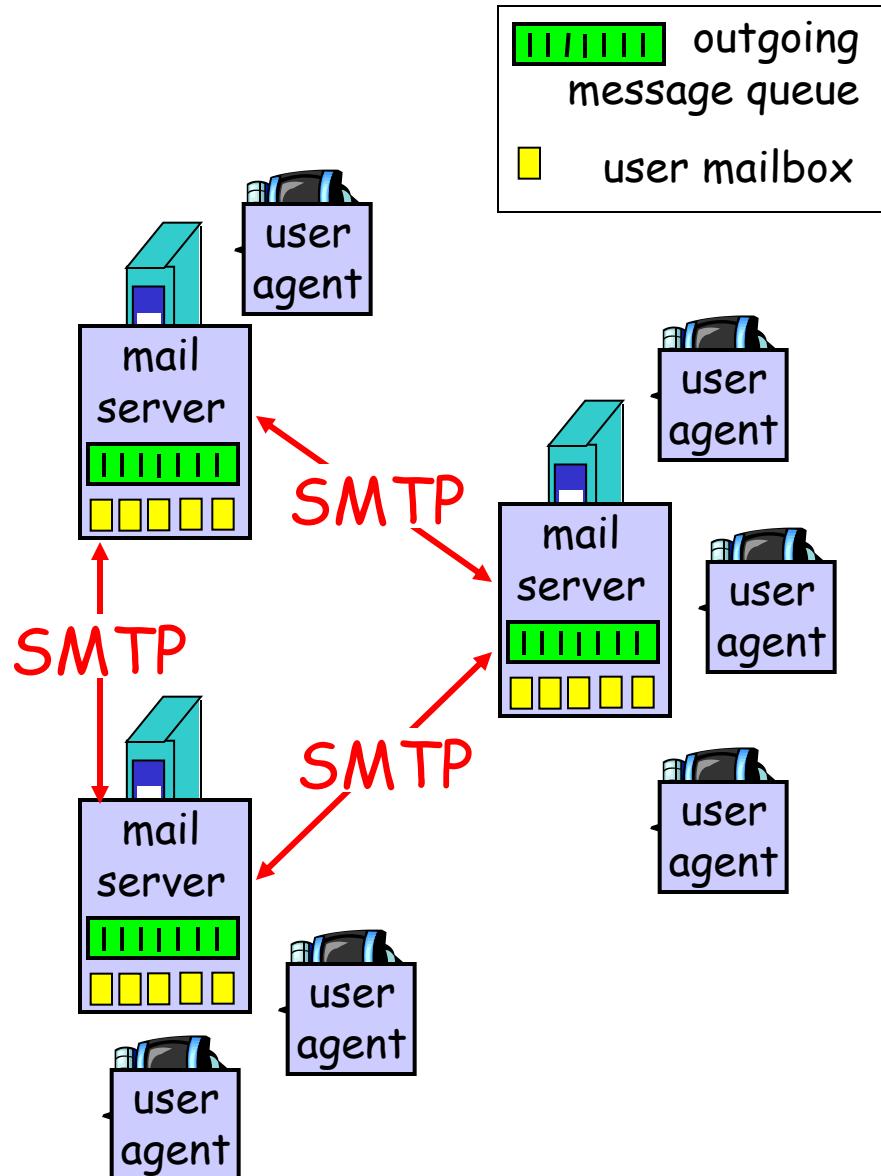
# 电子邮件应用画像

- r 应用层传输对象：邮件
- r 网络应用架构：客户-服务器架构
- r 使用传输服务：TCP
- r 应用层协议：
  - ❖ 邮件传输协议：SMTP（端口号25）
  - ❖ 邮件访问协议：POP3（端口号110），IMAP（端口号143），HTTP（端口号80）
- r SMTP、POP3、IMAP采用命令/响应交互

## 2.3.1 电子邮件系统

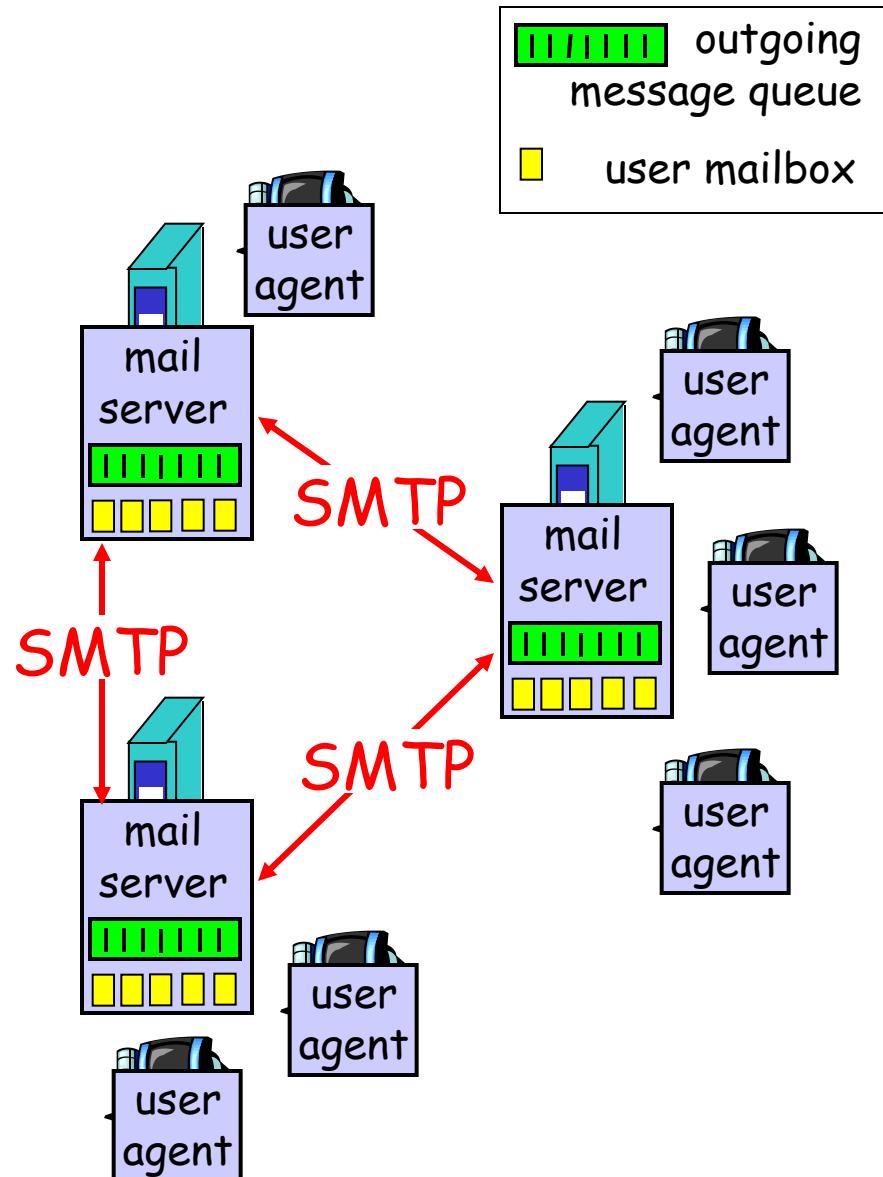
最初由三个部分组成：

- r 用户代理
- r 邮件服务器
- r 简单邮件传输协议



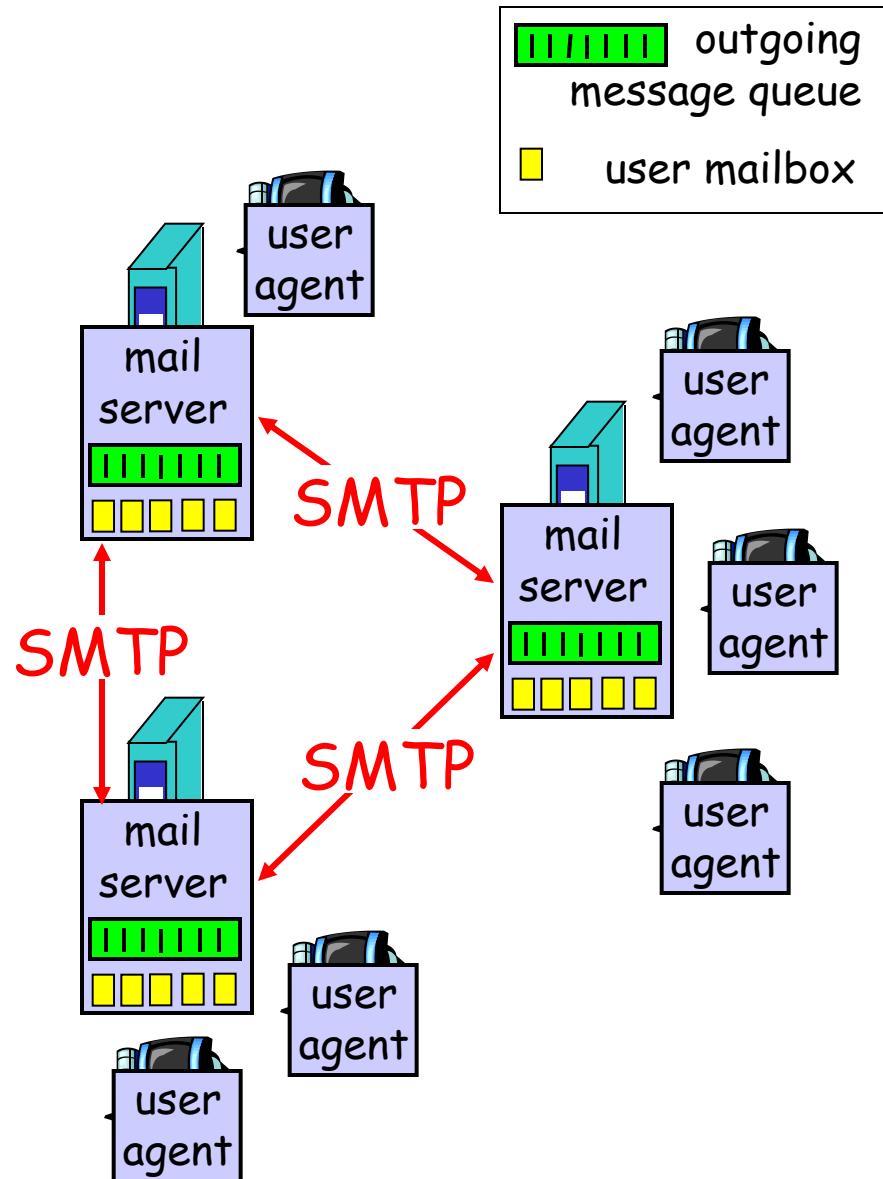
# 用户代理

- r 编辑、阅读、回复邮件等
  - r 将要外发的邮件发送到用户的邮件服务器
  - r 从用户邮箱中取邮件
- 
- r 一些用户代理:
    - ❖ Outlook, elm, Mozilla, Thunderbird



# 邮件服务器

- r 邮件服务器上包含：
  - ❖ **用户信箱**：存放到来的邮件
  - ❖ **发送报文队列**：存放要发送出去的邮件
  - ❖ **报文传输代理MTA**：运行在服务器后台的系统守护进程，负责在邮件服务器之间传输邮件，及将收到的邮件放入用户信箱



# 电子信箱

## r 电子信箱:

- ❖ 由计算机上的一个存储区域（如磁盘上的一个文件）组成
- ❖ 每个信箱均被分配了唯一的电子邮件地址

## r 电子邮件地址:

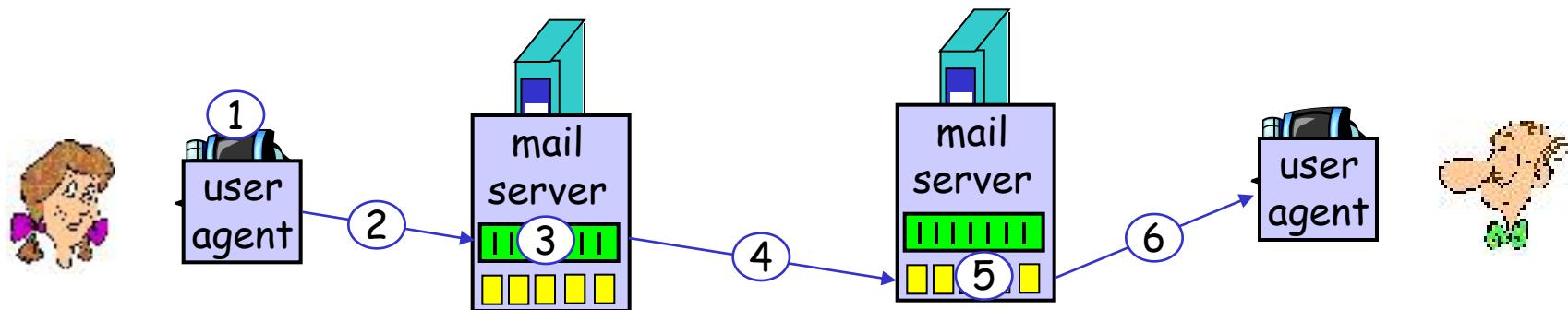
- ❖ 由两个部分组成，形如: `mailbox@computer`
- ❖ 前者为标识用户信箱的字符串，后者为信箱所在的邮件服务器的名字

# 简单邮件传输协议--SMTP [RFC 2821]

- r 邮件服务器之间传输邮件采用客户-服务器模式：
  - ❖ 客户：发送邮件的服务器
  - ❖ 服务器：接收邮件的服务器
- r SMTP使用TCP协议，服务器端口为25
- r 发送服务器和接收服务器之间直接传输邮件
- r SMTP采用命令/响应交互方式：
  - ❖ 命令：ASCII文本
  - ❖ 响应：状态码和短语
- r 报文只能包含简单ASCII文本，即7位ASCII码（最初仅为英文电子邮件而设计）

# Scenario: Alice sends message to Bob

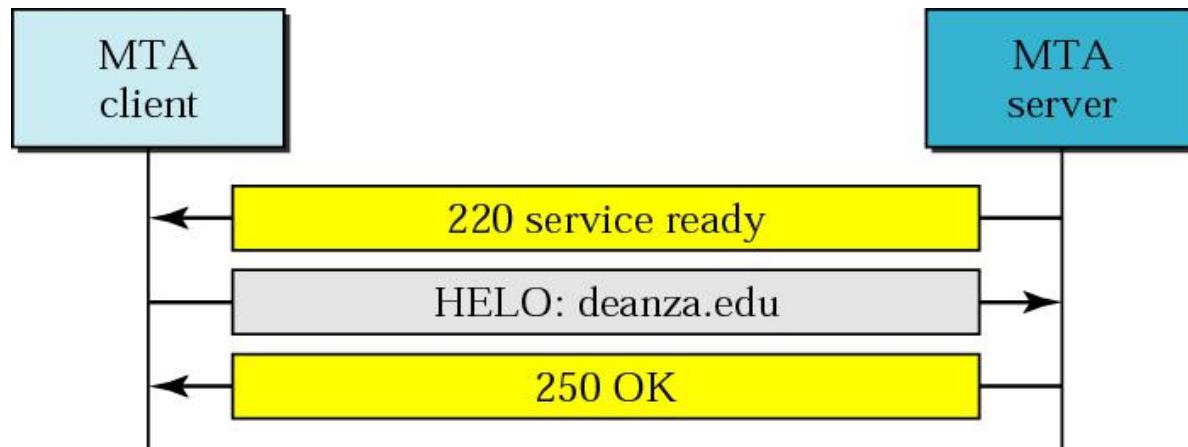
- 1) Alice 使用用户代理编辑邮件，发送给 **bob@someschool.edu**
- 2) Alice 的用户代理将报文发送给其邮件服务器（使用 **SMTP**）
- 3) 邮件被置于邮件服务器的发送报文队列中
- 4) Alice 的邮件服务器与 Bob 的邮件服务器建立 **TCP** 连接，然后发送 Alice 的邮件
- 5) Bob 的邮件服务器将邮件放置在 Bob 的信箱中
- 6) Bob 调用他的用户代理阅读邮件



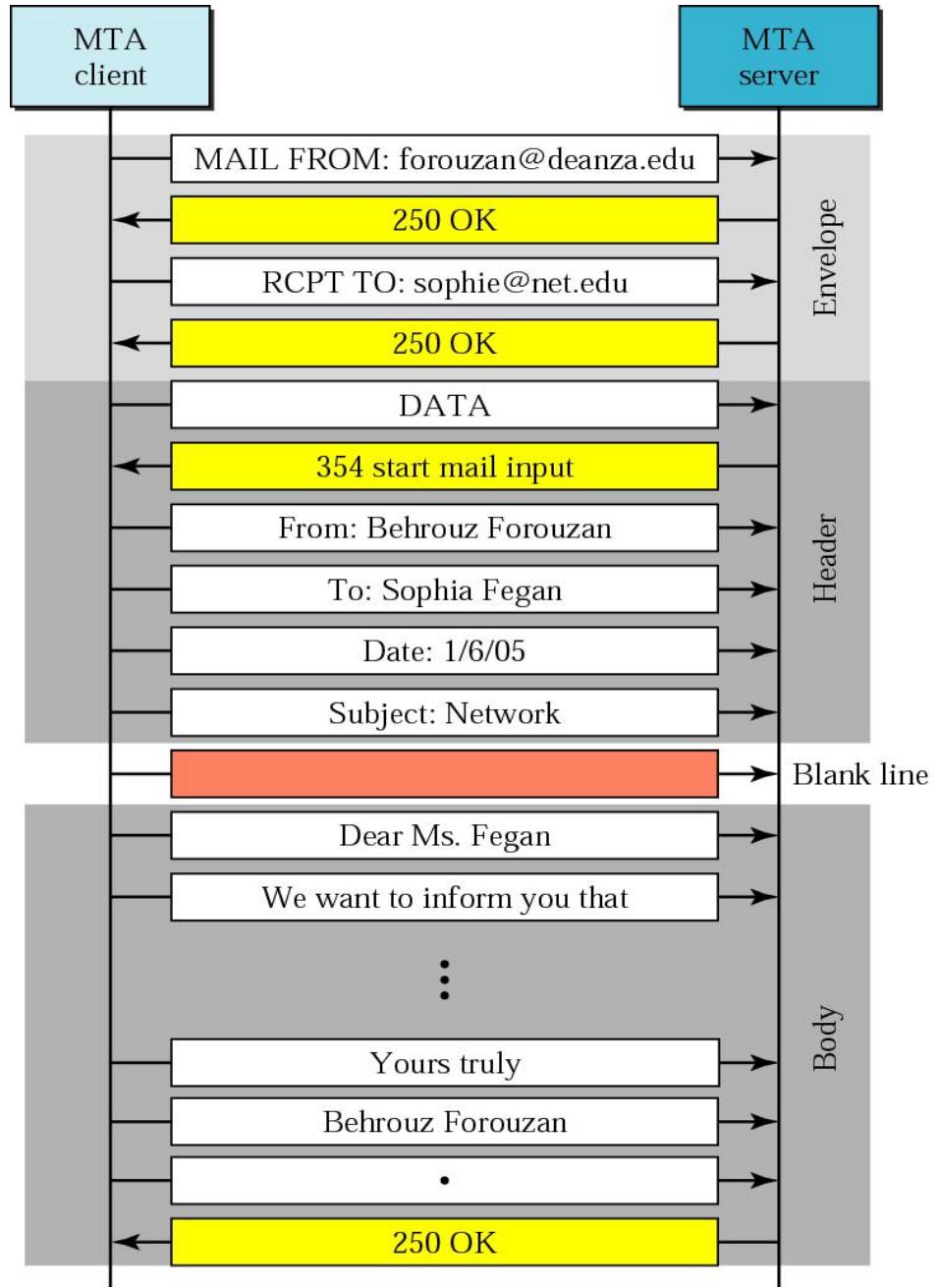
# 邮件传送阶段（1）

## 连接与握手：

- ❖ MTA客户与MTA服务器在端口25建立TCP连接。
- ❖ 服务器发送服务就绪报文
- ❖ 客户发送HELO报文，用域名标识自己
- ❖ 服务器响应



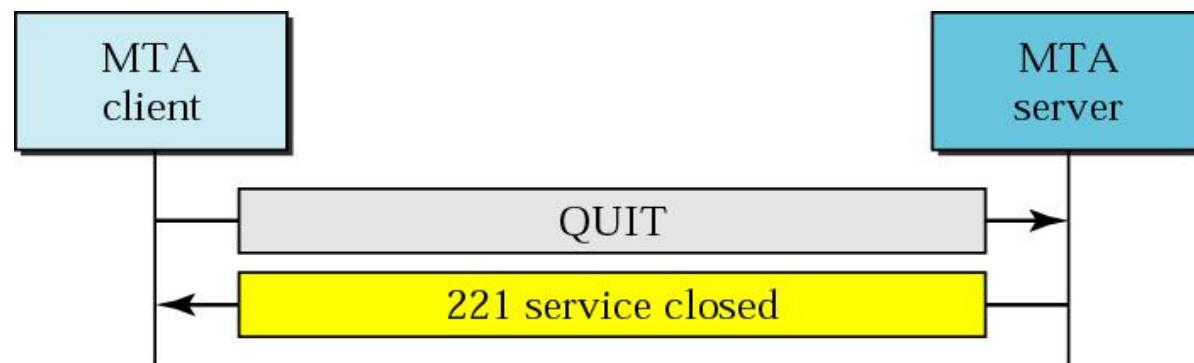
## 邮件传送阶段（2）



# 邮件传送阶段（3）

## 结束传输

- ❖ 客户发送QUIT报文
- ❖ 服务器响应
- ❖ 释放TCP连接

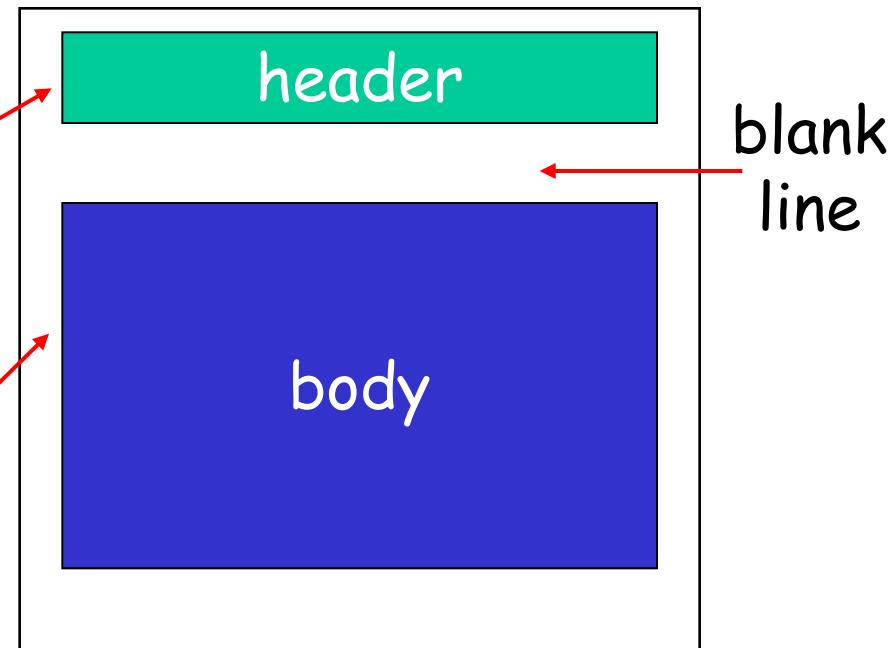


# SMTP: final words

- r SMTP使用持久连接：
  - ❖ 可以在一条TCP连接上传输多个报文（FTP只传输一个文件，HTTP可传输一个或多个对象）
  - ❖ 一个方向的报文传输结束后，可以在另一个方向上传输报文（SMTP特有的）
- r SMTP 服务器使用“.”表示报文结束（FTP使用关闭连接表示文件结束，HTTP使用长度域表示报文结束）
- r SMTP要求报文（报头和实体）只包含简单ASCII文本（HTTP无此要求）

## 2.3.2 邮件报文格式

- r RFC 822规定了邮件报文格式
- r 报头由一些首部行组成，如：
  - ❖ To:
  - ❖ From:
  - ❖ Subject:
- r 实体：只能使用简单 ASCII字符



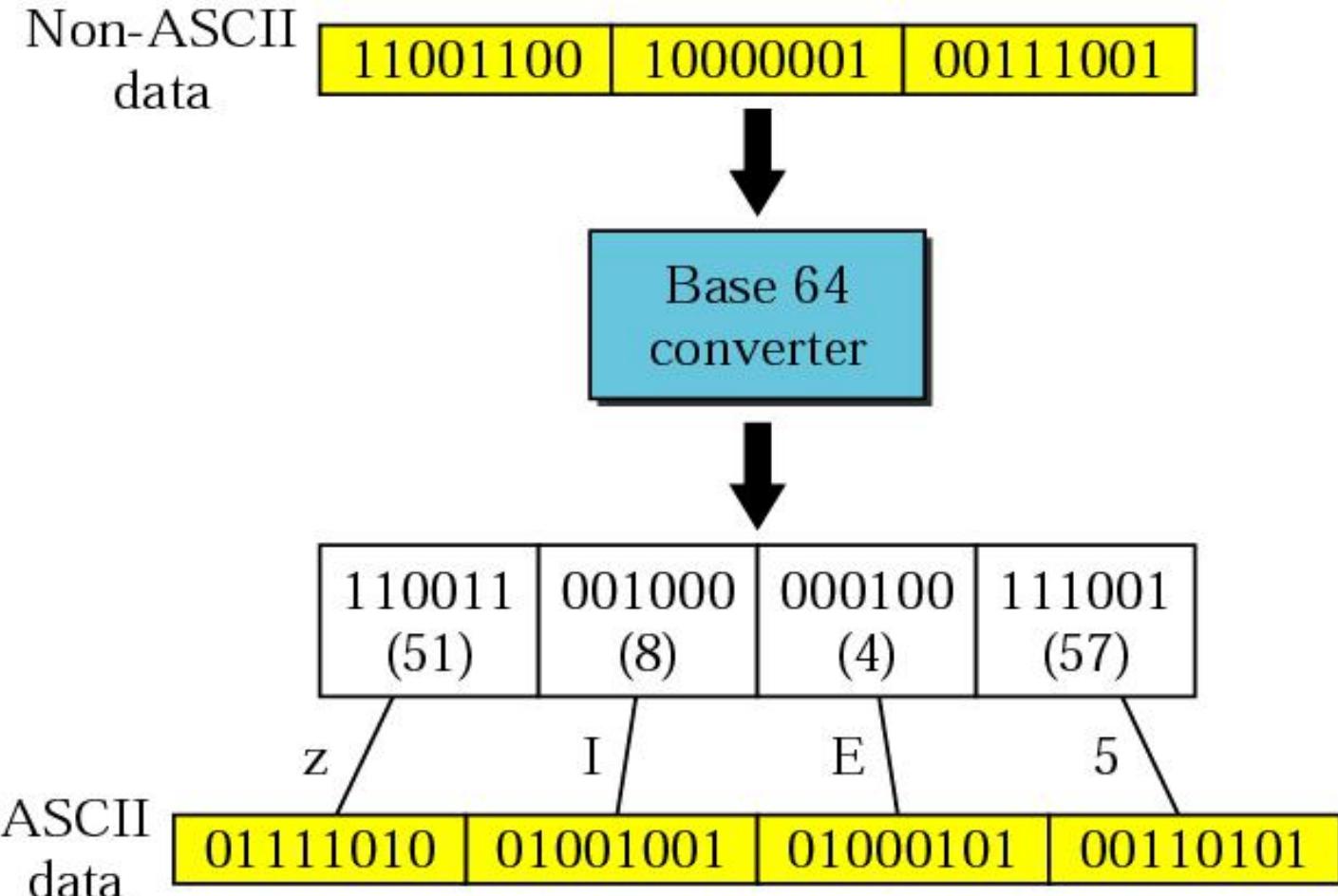
# 如何传输包含非ASCII文本的报文？

- 『 现在的电子邮件要求能传输其它语系的文字，甚至非文本信息（如图片、语音等）
  - ❖ 非ASCII文本形式的数据，在发送前须转换成简单ASCII文本
- 『 非ASCII文本的报文大多具有特殊的数据类型，需要特殊的邮件浏览器（如JPEG图形的解压缩软件）来阅读
  - ❖ 需扩展数据类型

# Base64编码

- r Base64用来将一个二进制字节序列，转换成由ASCII字符序列构成的文本
- r 每24比特数据划分成4个6比特的单元，每个单元编码成一个ASCII字符，其对应关系为：
  - ❖ 0~25编码成 ‘A’ ~ ‘Z’
  - ❖ 26~51编码成 ‘a’ ~ ‘z’
  - ❖ 52~61编码成 ‘0’ ~ ‘9’
  - ❖ 62和63分别编码成 ‘+’ 和 ‘/’
  - ❖ 若最后一组只有8比特或16比特，分别加上 ‘==’ 和 ‘=’ 后缀
  - ❖ 回车和换行忽略，可以插在任何地方

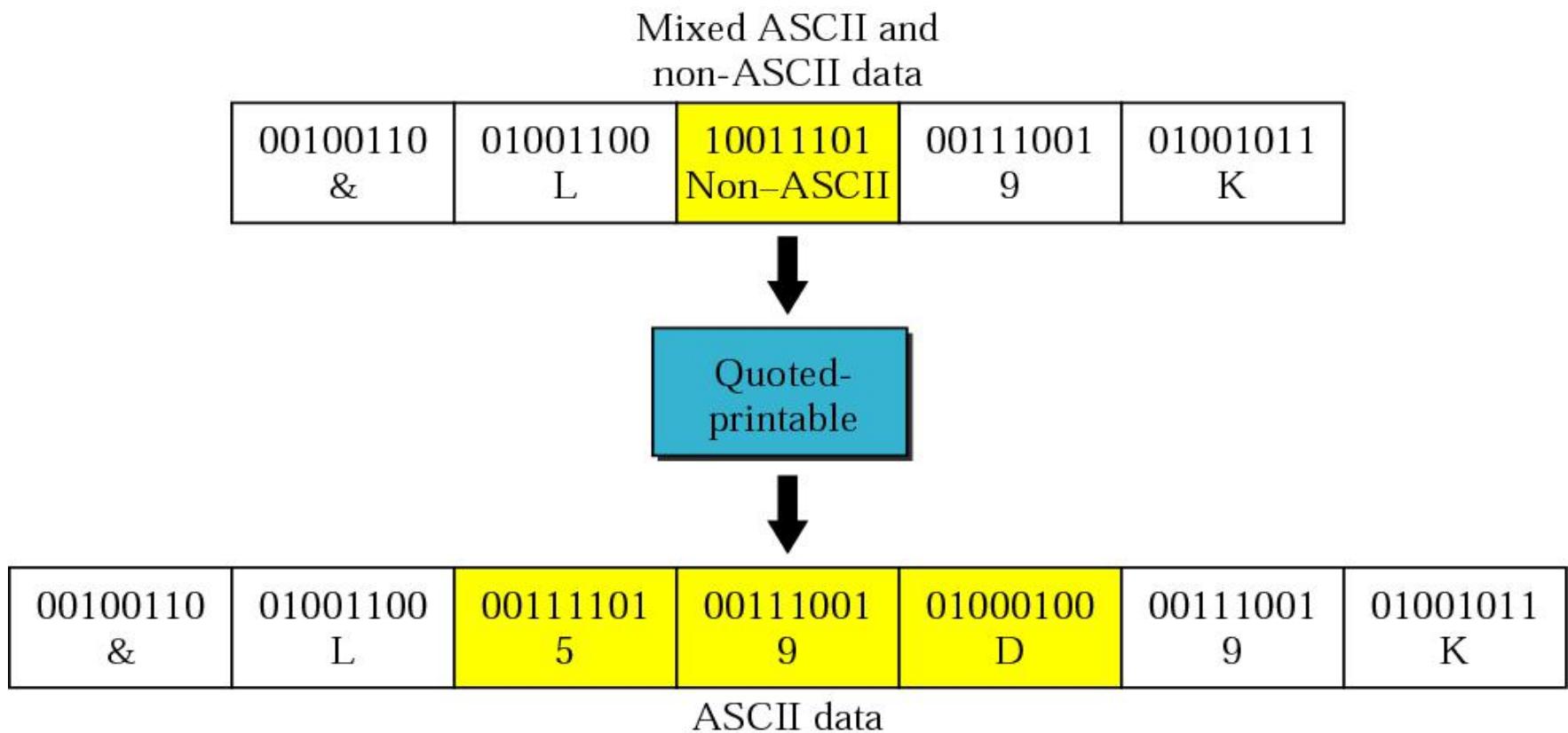
# Base64编码示例



## quoted-printable编码

- r 适用于绝大部分都是ASCII字符的报文实体，其编码方法是：
  - ❖ 每个ASCII字符保持不变
  - ❖ 对于非ASCII字符（大于127的字符），将该字符的十六进制表示用两个ASCII字符标记，前面冠以特殊字符“=”。

# quoted-printable编码示例



# 多用途因特网邮件扩展协议MIME

- 『 扩展了RFC 822，允许实体具有不同的数据类型，并规定了非ASCII文本信息在传输时的统一编码形式
- 『 扩充了一些首部行，最重要的是：
  - ❖ **Content-Transfer-Encoding:** 实体采用的传输编码形式
  - ❖ **Content-Type:** 实体的数据类型及子类型

# MIME报文格式 [RFC 2045, 2056]

MIME version  
method used  
to encode data  
multimedia data  
type, subtype,  
parameter declaration  
encoded data

The diagram illustrates the structure of a MIME message. It consists of two main parts: a header section and a body section. The header is enclosed in a rectangular box and contains the following fields:

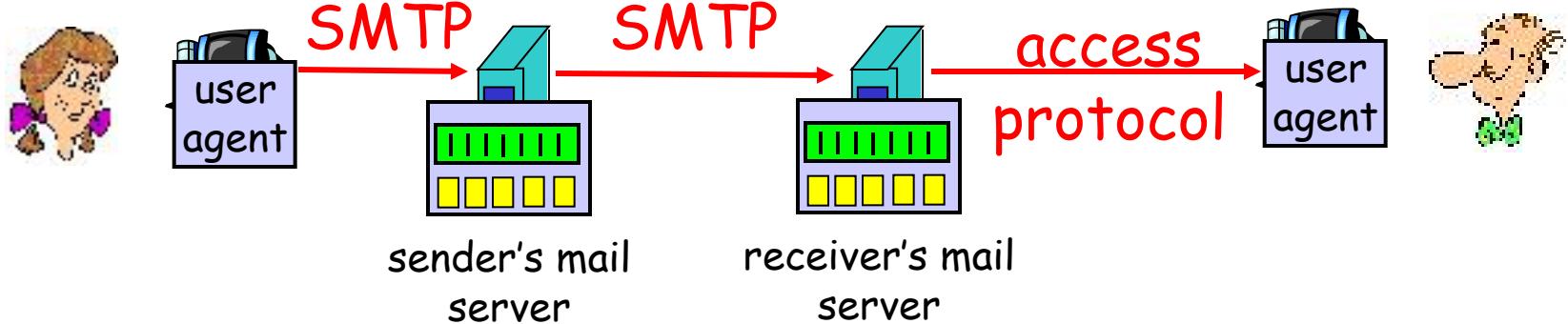
- From:** alice@crepes.fr
- To:** bob@hamburger.edu
- Subject:** Picture of yummy crepe.
- MIME-Version:** 1.0
- Content-Transfer-Encoding:** base64
- Content-Type:** image/jpeg

Below the header, there is a large bracketed area containing the text "base64 encoded data .....". This bracket spans several lines of dots, indicating the continuation of the encoded data. Red arrows from the left margin point to specific parts of the header and the encoded data area, corresponding to the labels "MIME version", "method used to encode data", "multimedia data type, subtype, parameter declaration", and "encoded data".

## 2.3.3 邮件访问

- r 邮件访问方式：
  - ❖ 早期：用户登陆到邮件服务器上，直接在服务器上运行一个邮件阅读程序来阅读邮件
  - ❖ 今天：用户在终端上安装用户代理，获取和阅读邮件
- r Q: 能将用户信箱放在本地终端吗？
- r A: 不能，用户终端不可能一直连在因特网上
- r Q: 用户代理能用**SMTP**从邮件服务器获取邮件吗？
- r A: 不能，**SMTP**是一个“推”协议，只能将邮件从用户代理推送到其邮件服务器
- r 解决方案：设计一个新的协议从服务器获取邮件

# 邮件的两阶段交付



- r 在具有永久因特网连接的计算机上运行一个SMTP服务器，为用户分配一个永久信箱
- r 第一阶段：邮件被投递到收信人的永久信箱
- r 第二阶段：用户从永久信箱中获取邮件
- r 为此，带永久信箱的计算机必须运行两个服务器程序：
  - ❖ SMTP服务器：收发用户邮件，将收到的邮件放入用户信箱
  - ❖ 邮件访问服务器：允许用户从信箱中提取邮件

# 邮件访问协议

可以从服务器获取邮件的协议有：

- r **POP**: Post Office Protocol [RFC 1939]
  - ❖ authorization (agent <-->server) and download
- r **IMAP**: Internet Mail Access Protocol [RFC 1730]
  - ❖ more features (more complex)
  - ❖ manipulation of stored msgs on server
- r **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.

# POP3 protocol

## 认证和授权阶段

- r 客户命令:
  - ❖ **user**: declare username
  - ❖ **pass**: password
- r 服务器响应
  - ❖ +OK
  - ❖ -ERR

## 事务阶段

- r 客户命令:
  - ❖ **list**: list message numbers
  - ❖ **retr**: retrieve message by number
  - ❖ **dele**: delete
  - ❖ **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

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
```

# POP3 (more) and IMAP

## More about POP3

- r 前一个例子使用了“**download and delete**”模式，在另一台终端上无法再获取邮件
- r “**Download-and-keep**”模式可将邮件保留在服务器中

## IMAP

- r 所有邮件保存在服务器上
- r 允许用户将邮件组织在文件夹中
- r 允许用户在文件夹之间移动邮件

# 基于web的邮件访问：HTTP

- r 用户代理为普通浏览器：
  - ❖ 发送邮件：浏览器使用**HTTP协议**将邮件发送到邮件服务器
  - ❖ 获取邮件：浏览器使用**HTTP协议**从信箱取邮件
  - ❖ 传输邮件：邮件服务器之间仍使用**SMTP协议**传输报文
- r 和**IMAP**一样，用户可以在远程服务器上用文件夹来组织他们的邮件

# 现代因特网电子邮件系统的组成

- r 用户代理
- r 邮件服务器
- r 简单邮件传输协议**SMTP**
- r 邮件访问协议（**POP3, IMAP, HTTP**）

# 小结

- 『 电子邮件系统：

- ❖ 4个组成部分

- 『 **SMTP** 协议：

- ❖ 使用 **TCP** 协议，服务器端口号 **25**
  - ❖ “推”协议：将邮件推向用户信箱
  - ❖ 命令 / 响应交互方式
  - ❖ 信体只能包含简单 **ASCII** 文本

- 『 **MIME** 协议：

- ❖ 允许信体包含非 **ASCII** 文本
  - ❖ 规定传输编码类型，扩展数据类型

- 『 两阶段交付过程：

- ❖ 邮件投递：邮件从发信方投递到用户信箱
  - ❖ 邮件访问：收信人从用户信箱获（拉）取邮件

# 理解HTTP、FTP、SMTP设计上的不同

- r HTTP、FTP、SMTP均是在TCP连接上传输文件，但是在设计上有一些不同
- r 使用持久连接或非持久连接：
  - ❖ HTTP可在一条TCP连接上传输多个对象，SMTP可以传输多个邮件，FTP只传输一个文件
- r 文件传输结束的标记：
  - ❖ HTTP使用长度指示报文结束，FTP使用关闭连接表示文件结束，SMTP 使用“.” 表示报文结束
- r 文件内容的要求：
  - ❖ SMTP要求邮件只包含简单ASCII文本，FTP和HTTP无此要求
- r 客户-服务器交互方式：
  - ❖ HTTP采用报文交互，SMTP和FTP采用命令/响应交互

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充: File Transfer and  
FTP

2.3 electronic mail

- ❖ SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# DNS: Domain Name System

- r 因特网主机需要标识:
  - ❖ IP地址（32 bit） - 由机器使用
  - ❖ 名字，如 `www.yahoo.com` - 由人类使用，便于记忆
- r Q: 如何提供主机名到IP地址的映射？
- r 因特网的目录服务**DNS**:
  - ❖ 将主机名映射到IP地址
- r DNS实现为一个应用层服务:
  - ❖ 由其它网络应用使用的服务
  - ❖ 客户-服务器模式
  - ❖ 传输服务:
    - 主要使用**UDP**, 有时使用**TCP**
  - ❖ 端口号**53**
  - ❖ 请求/响应报文交互

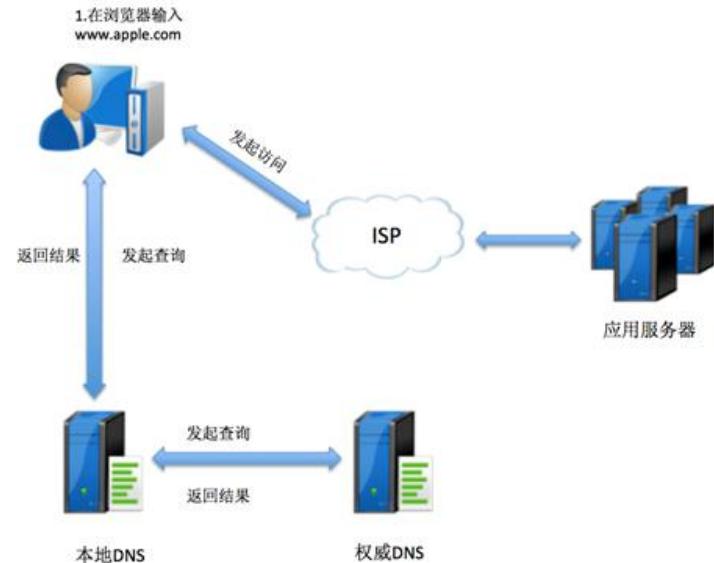
## 2.4.1 DNS提供的服务

- r 主机名-IP地址转换
- r 主机别名：
  - ❖ 允许主机除了规范名外，具有一个或多个别名（易于记忆），如 www.ustc.edu.cn
  - ❖ 提供主机别名到规范名的映射
  - ❖ 迁移服务不需要修改主机名
- r 邮件服务器别名：
  - ❖ 允许使用域名作为邮件服务器的别名
  - ❖ 如：xxx@ustc.edu.cn
- r 负载分配：
  - ❖ 允许一个规范主机名对应一组IP地址
  - ❖ 将服务请求在一群相同功能的服务器之间分配

## 2.4.2 DNS工作机理

将主机名转换成IP地址：

- r 应用程序（如浏览器）调用一个本地例程（**解析器**），主机名作为参数之一传递
- r 解析器向网络中的**DNS**服务器发送查询报文，包含要查询的主机名
- r 解析器收到包含**IP**地址的响应报文
- r 解析器将**IP**地址返回给调用者（如浏览器）



\*\* 对应用程序而言，**DNS**是一个提供直接转换服务的黑盒子

# DNS是一个分布式数据库

**Q:** 为什么不使用集中式的**DNS**?

- r 单点失效
- r 流量集中：单个**DNS**服务器需处理全部查询
- r 响应时间长：远距离的集中式数据库
- r 需要维护庞大的数据库

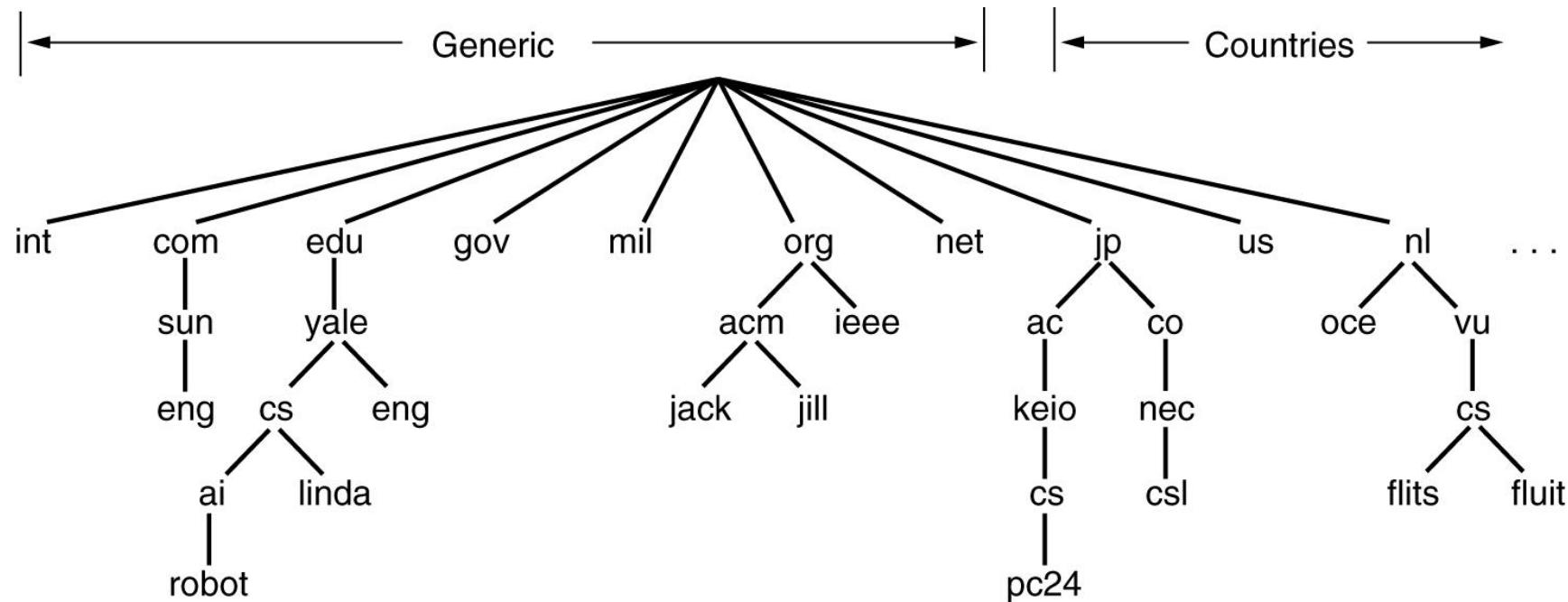
**A:** *Doesn't scale!*

# 分布式环境中的名字空间

如何在分布式环境下避免出现名字冲突？

『 DNS 使用名字空间来规范对主机的命名：』

- ❖ 名字空间是因特网主机名字的集合，它同时给出了命名主机的方法
- ❖ 概念上，因特网被划分成200多个顶级域，每个顶级域可继续划分子域，依次类推
- ❖ 主机名字采用分层的命名方法

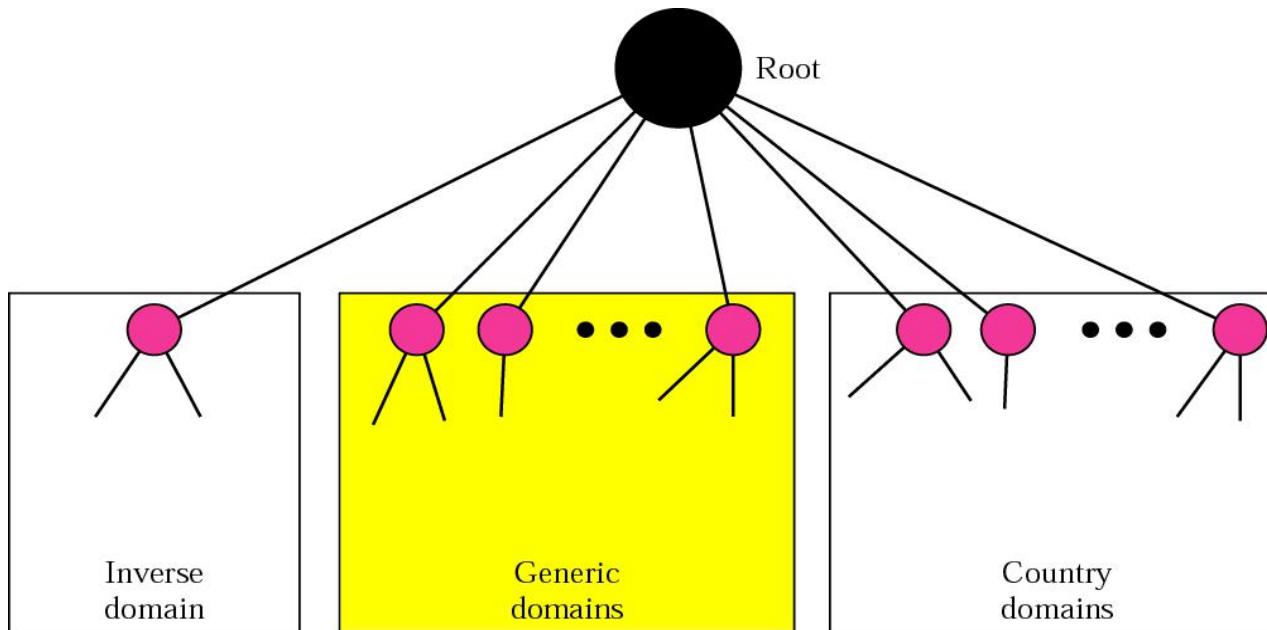


# 域名

- r 域 (domain) :
  - ❖ 名字树中，以任何一个节点为根的子树构成一个域
- r 标记 (label) :
  - ❖ 树上每一个节点都有一个标记（最多63个字符），树根的标记是一个空字符串
- r 域名 (domain name) :
  - ❖ 某个域的名字表示为：从该域开始向上直到树根的标记序列，标记之间用句点隔开（类似国外邮政地址的写法）
- r 域名的任一后缀也是一个域，同一个机构内的主机具有相同的域名后缀
- r **每个节点只需保证其孩子节点的标记不重名**

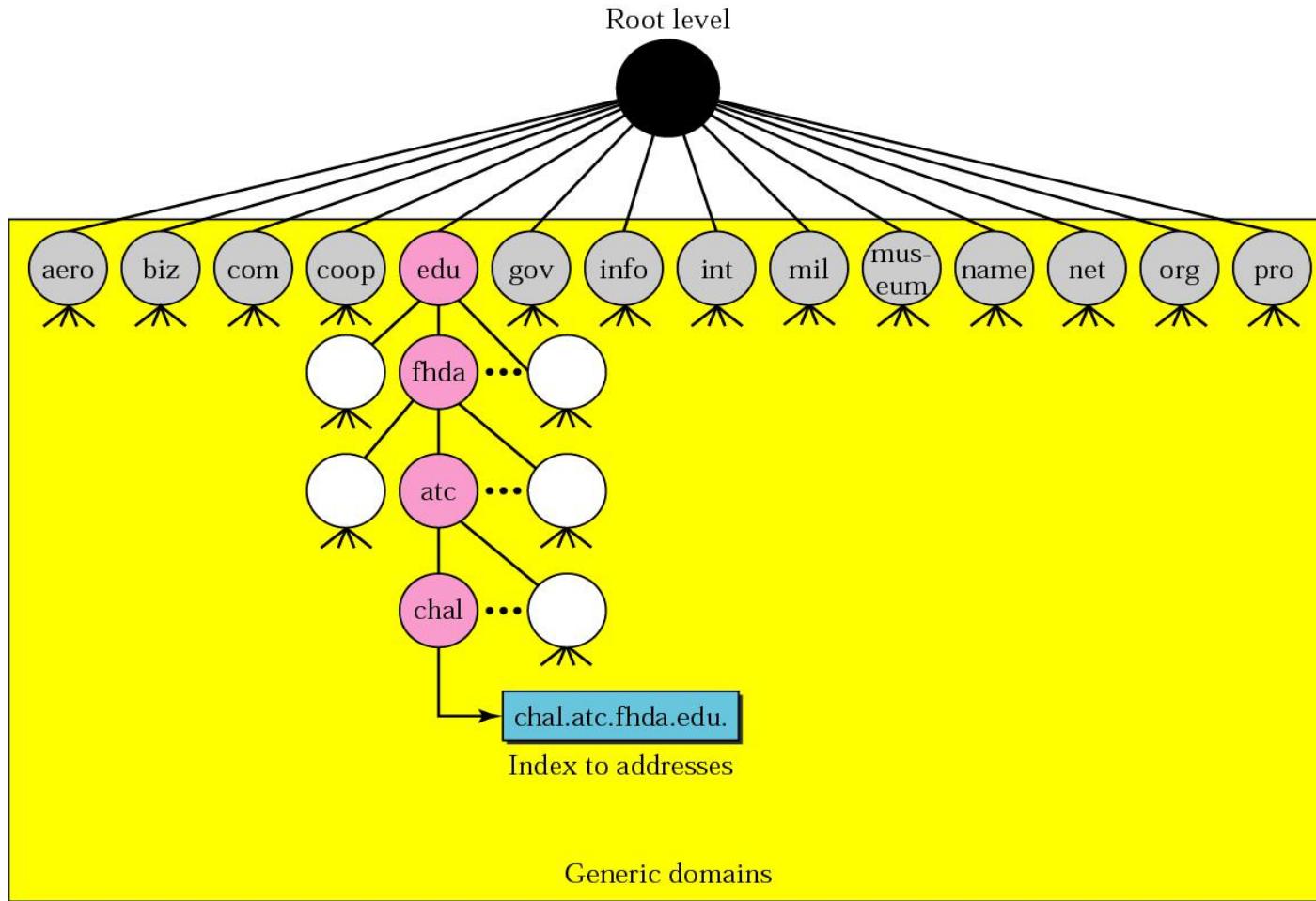
# 顶级域

- 顶级域分为组织域、国家域和反向域三种



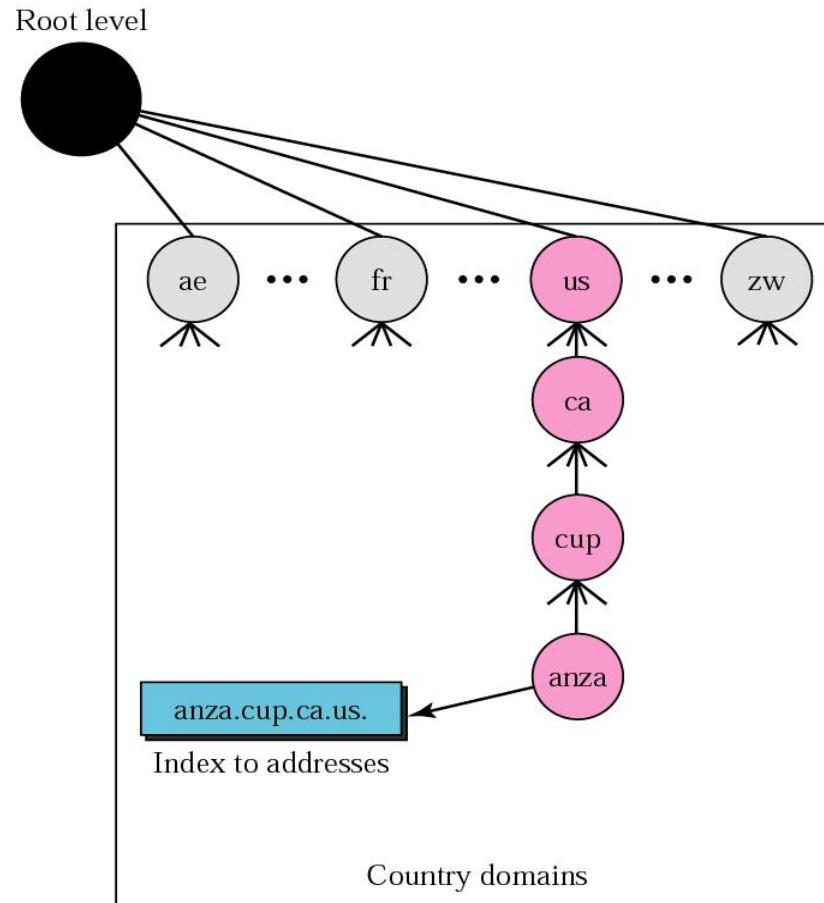
# 组织域

由美国国内及一些国际组织使用



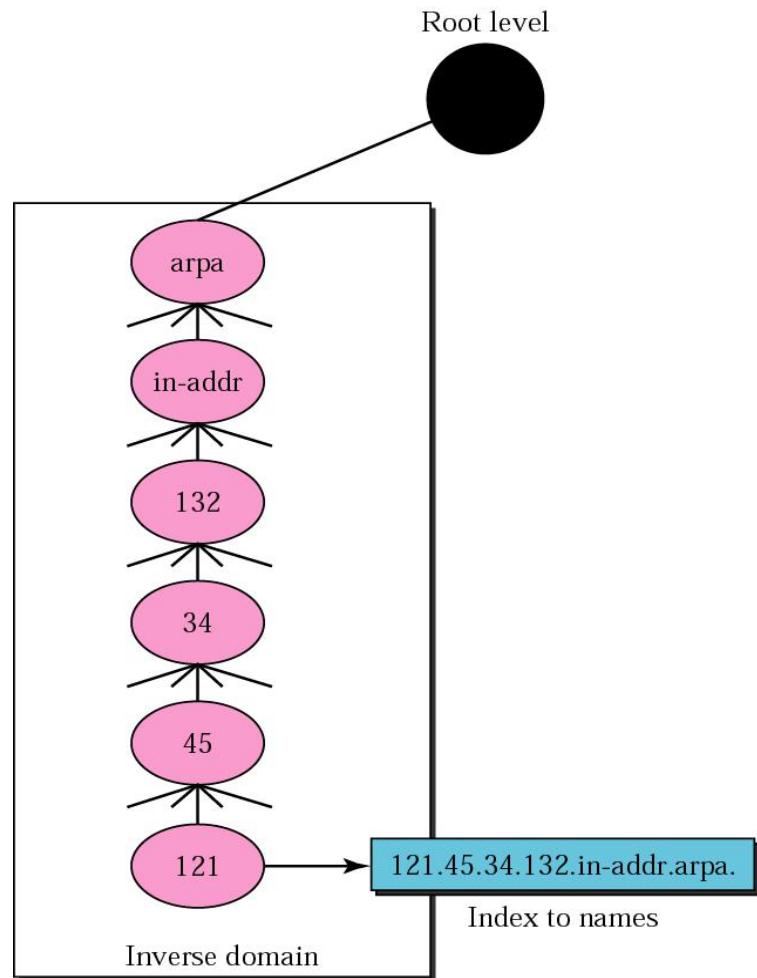
# 国家域

使用二字符的国家代码，每个国家对应一个

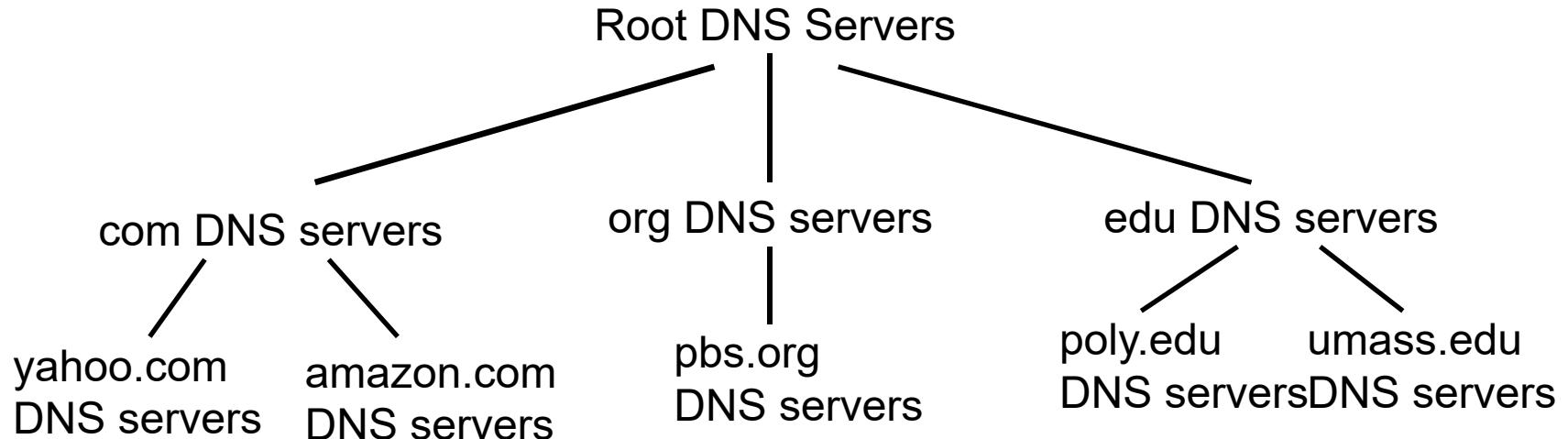


# 反向域

- r 顶级域名为arpa，用来将一个IP地址映射为注册的域名，反向域名解析是为了溯源
- r DNS提供了一个反向解析域in-addr.arpa：
  - r 欲解析的IP地址表示成像域名一样的一个串，例如，IP地址132.34.45.121表示为121.45.34.132.in-addr.arpa
  - r 以这个字符串作为参数调用解析器
  - r 电信运营商使用自己的DNS服务器提供相关IP地址的反向解析服务



# 域名服务器的组织层次

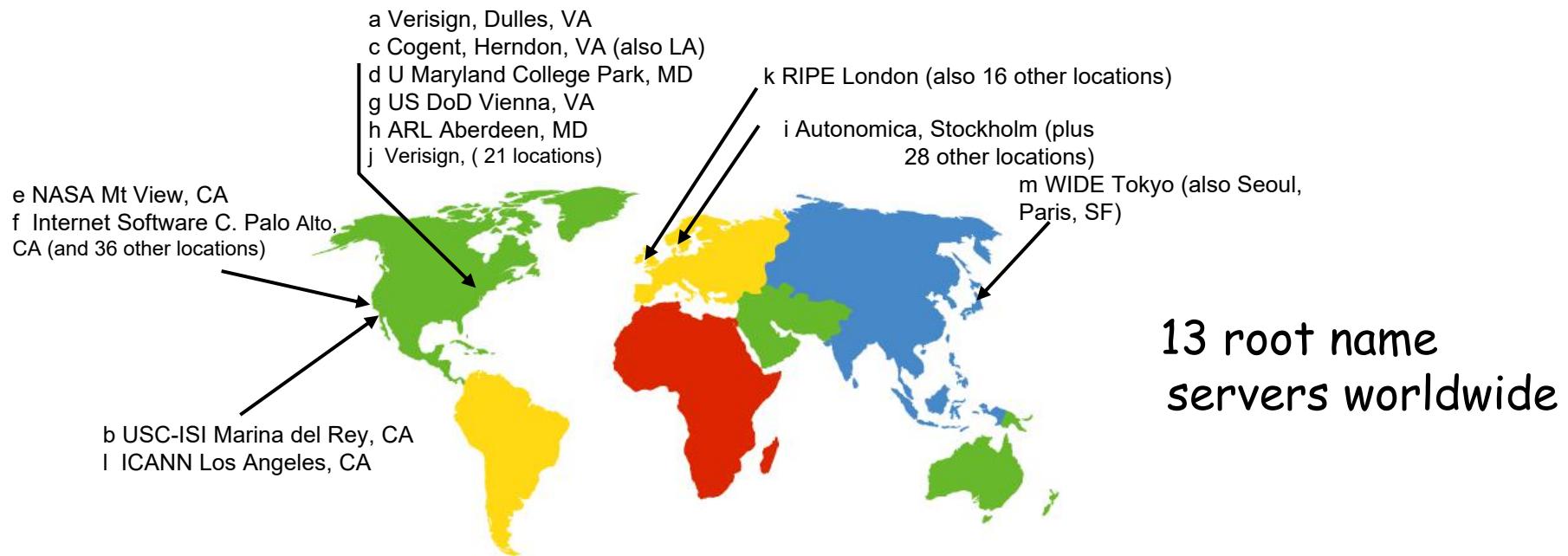


客户想知道www.amazon.com的IP地址：

- r DNS客户查询根服务器，得到com域的DNS服务器地址
- r DNS客户查询com域的DNS服务器，得到amazon.com域的DNS服务器地址
- r DNS客户查询amazon.com域的DNS服务器，得到www.amazon.com的IP地址

# 根域名服务器

- r 全球有13个根服务器（每个根服务器是一个服务器集群）
- r 根服务器知道所有顶级域服务器的IP地址



# 顶级域服务器，权威服务器

- r 顶级域 (**Top Level Domain, TLD**) 服务器：
  - ❖ 每个**TLD**服务器负责一个顶级域
  - ❖ 知道其所有二级子域的域名服务器的地址
  
- r 权威**DNS**服务器：
  - ❖ 机构的**DNS**服务器，提供机构内部服务器的名字映射
  - ❖ 提供一个主域名服务器、一个或多个辅助域名服务器
  - ❖ 可由机构维护，也可委托**ISP**维护

# 本地DNS服务器

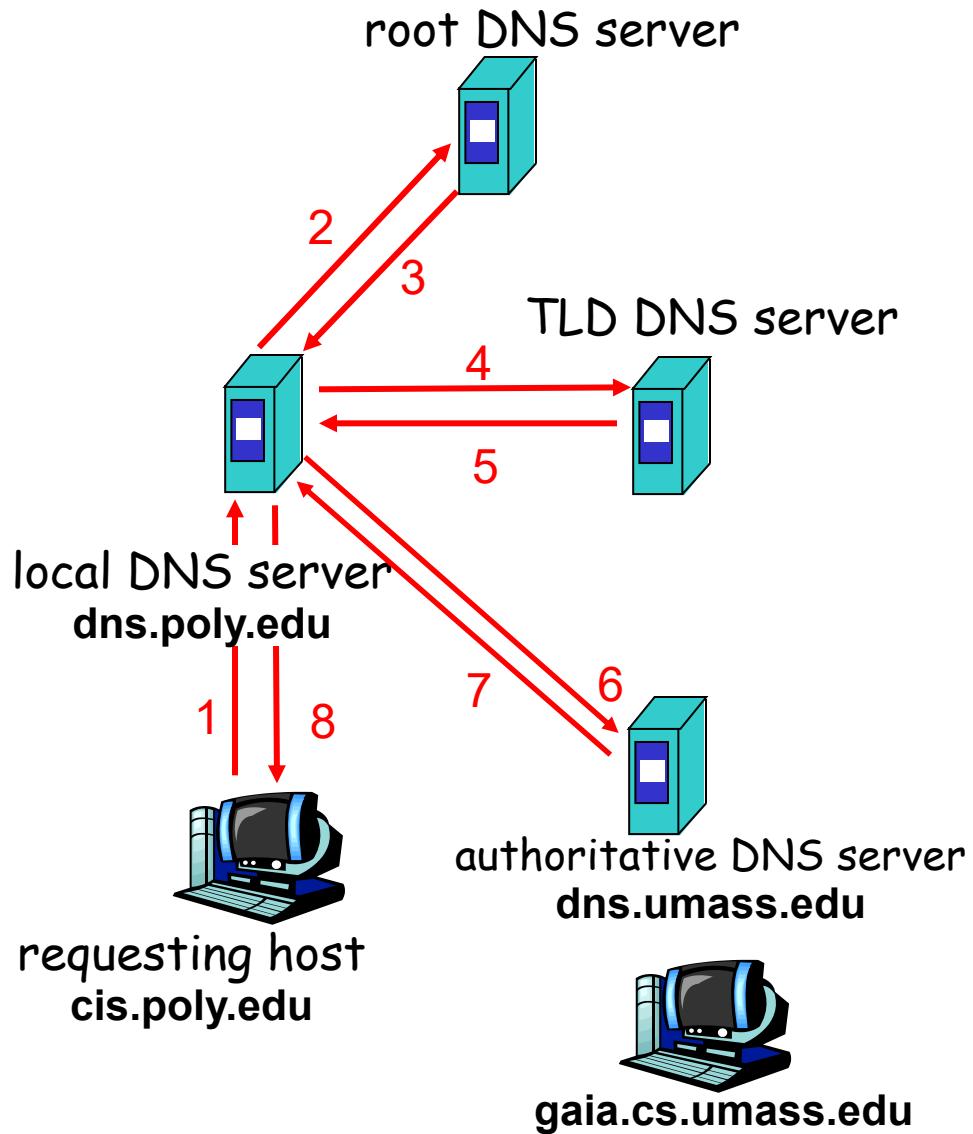
- r 严格来说，本地**DNS**服务器**不属于****DNS**服务器的层次结构
- r 每个**ISP**都有一台本地**DNS**服务器，也称“默认的**DNS**服务器”
- r 解析器的**DNS**查询报文实际上发送给本地**DNS**服务器
- r **本地DNS服务器起着代理的作用**，负责将**DNS**查询报文发送到**DNS**层次结构中，并将查找结果返回给解析器

# 域名解析的例子

- 『 cis.poly.edu 上的一台主机想知道 gaia.cs.umass.edu 的 IP 地址』

## 迭代查询:

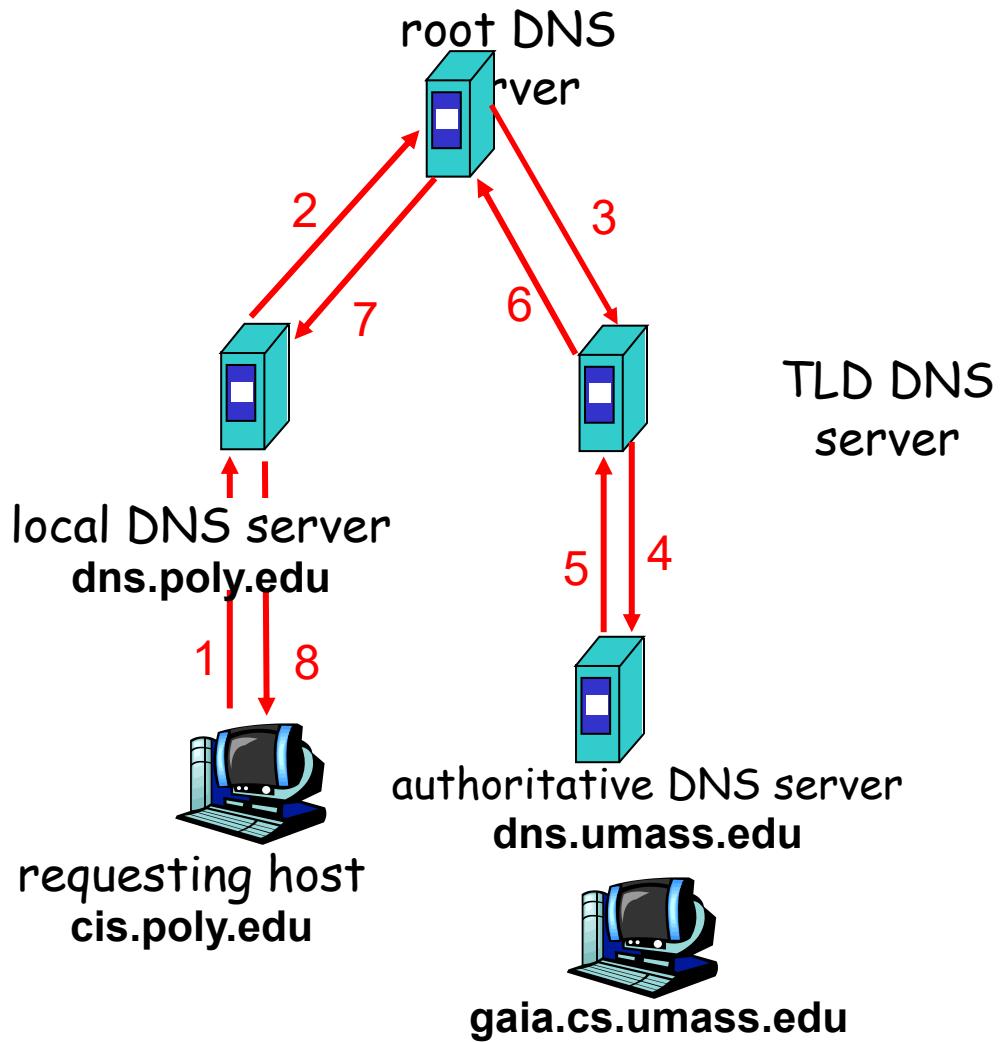
- 『 收到查询报文的服务器将下一个需要查询的服务器地址返回给查询者』
- 『 "I don't know this name, but ask this server" 』



# 域名解析的例子

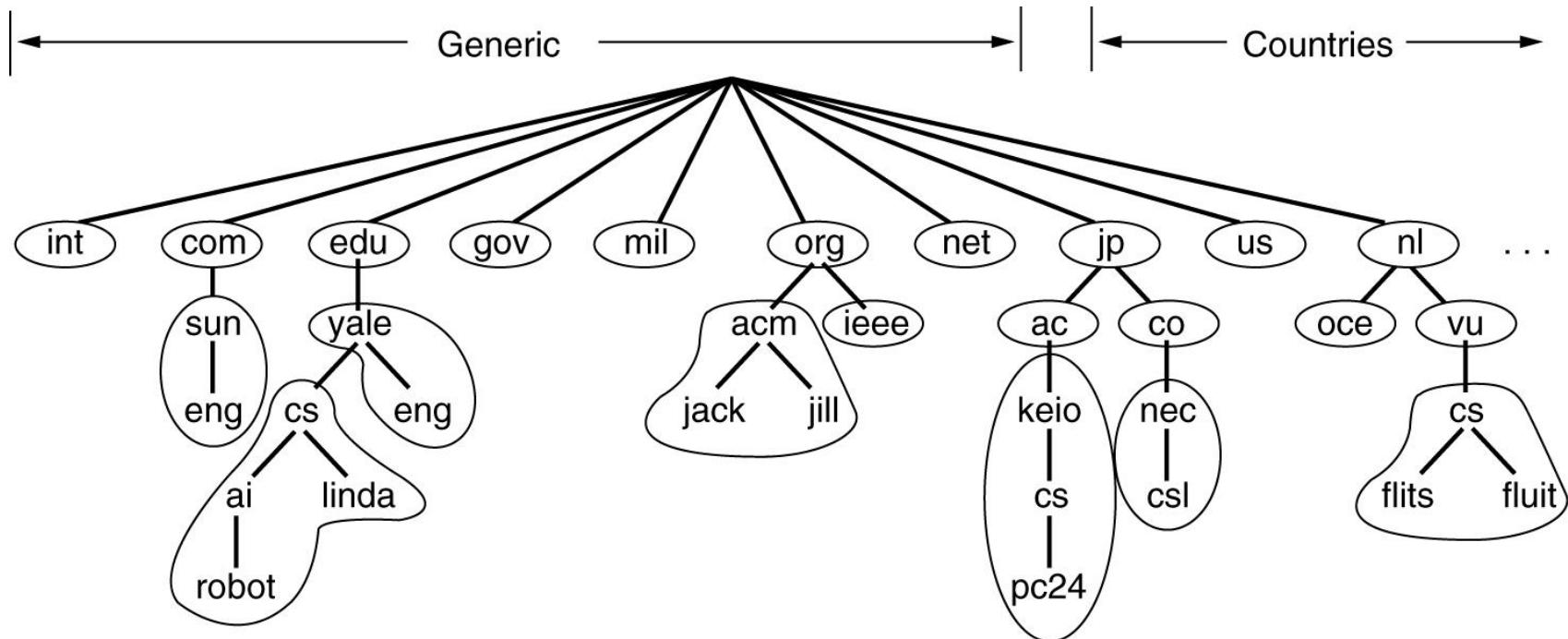
## 递归查询：

- r puts burden of name resolution on contacted name server
- r 实际的域名解析过程：
  - ❖ 递归 + 迭代
  - ❖ 解析器：递归查询
  - ❖ 本地DNS服务器：迭代查询
- r 本地DNS服务器的存在，使得DNS对于解析器来说是一个黑盒子



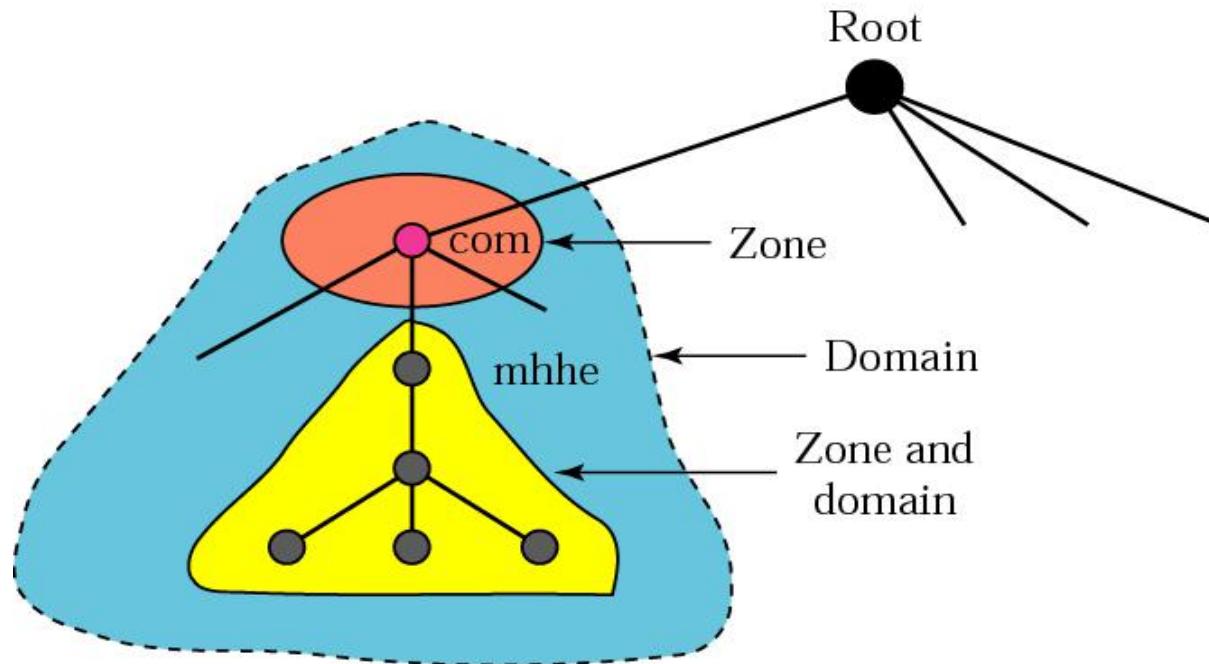
# DNS zone

- 整个DNS名字空间被划分成一些不重叠的区域，称为DNS zone，每个zone包含域名树的一部分。
- 在管理上，每个DNS zone代表一个权威域的边界。



# 物理服务器的层次

- r 一个物理服务器保存的信息可能涉及域名空间的若干层，它也可以把它的域划分成若干子域，把其中的一些子域委托给其它服务器
- r 实际的物理服务器的层次与域名空间的逻辑层次不同



# DNS缓存

- r 每当收到一个响应报文，**DNS**服务器将报文中的映射信息缓存在本地。
- r **DNS**服务器首先使用缓存中的信息响应查询请求
- r **DNS**缓存中的映射在一定时间后被丢弃
- r 特别地，本地**DNS**服务器通常会缓存**TLD**服务器的**IP地址**，因而很少去访问根服务器

## 2.4.3 DNS资源记录

DNS更准确的说法：存储资源记录（RR）的分布式数据库

RR format: (name, type, ttl, value)

- r Type=A
  - ❖ Name: 主机名
  - ❖ Value: IP地址
- r Type=NS
  - ❖ Name: 域 (e.g. foo.com)
  - ❖ value: 该域的权威 DNS服务器的主机名
- r Type=CNAME
  - ❖ Name: 别名
  - ❖ Value: 规范名
- r Type=MX
  - ❖ Name: 域(e.g. foo.com)
  - ❖ Value: 该域的邮件服务器名字

# DNS数据库内容示例

```
; Authoritative data for cs.vu.nl
cs.vu.nl.      86400  IN  SOA   star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.      86400  IN  TXT   "Divisie Wiskunde en Informatica."
cs.vu.nl.      86400  IN  TXT   "Vrije Universiteit Amsterdam."
cs.vu.nl.      86400  IN  MX    1 zephyr.cs.vu.nl.
cs.vu.nl.      86400  IN  MX    2 top.cs.vu.nl.

flits.cs.vu.nl. 86400  IN  HINFO Sun Unix
flits.cs.vu.nl. 86400  IN  A    130.37.16.112
flits.cs.vu.nl. 86400  IN  A    192.31.231.165
flits.cs.vu.nl. 86400  IN  MX   1 flits.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX   2 zephyr.cs.vu.nl.
flits.cs.vu.nl. 86400  IN  MX   3 top.cs.vu.nl.
www.cs.vu.nl.   86400  IN  CNAME star.cs.vu.nl
ftp.cs.vu.nl.   86400  IN  CNAME zephyr.cs.vu.nl

rowboat          IN  A    130.37.56.201
                  IN  MX   1 rowboat
                  IN  MX   2 zephyr
                  IN  HINFO Sun Unix

little-sister    IN  A    130.37.62.23
                  IN  HINFO Mac MacOS

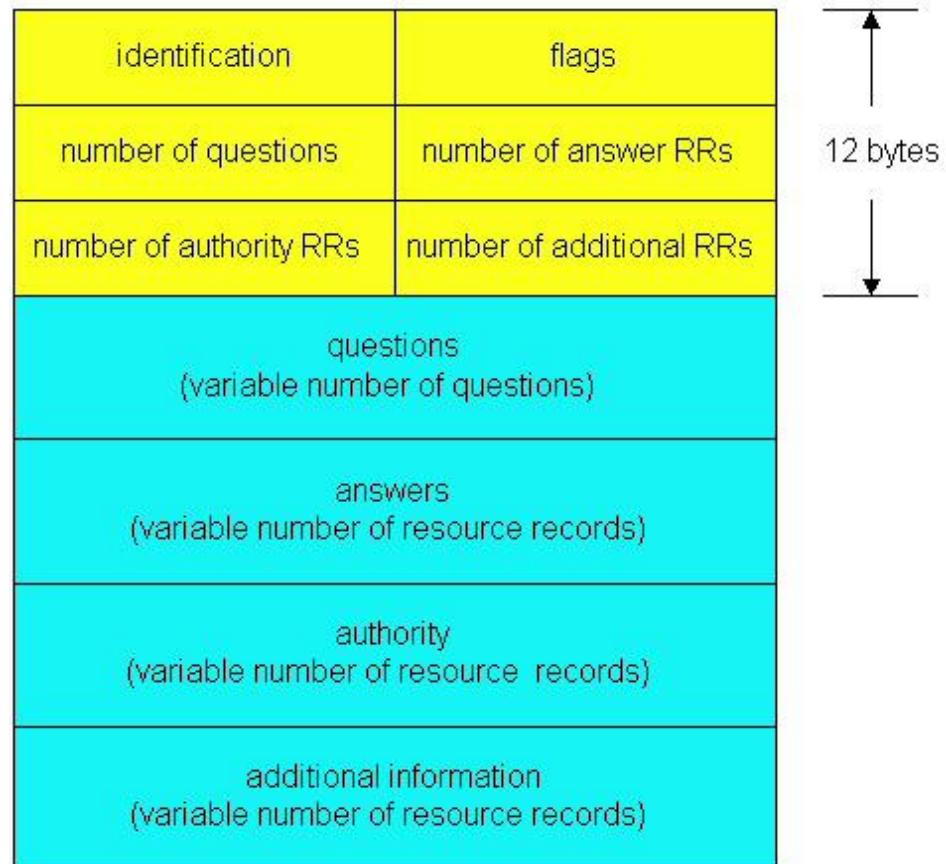
laserjet         IN  A    192.31.231.216
                  IN  HINFO "HP Laserjet IISi" Proprietary
```

## 2.4.4 DNS协议，报文

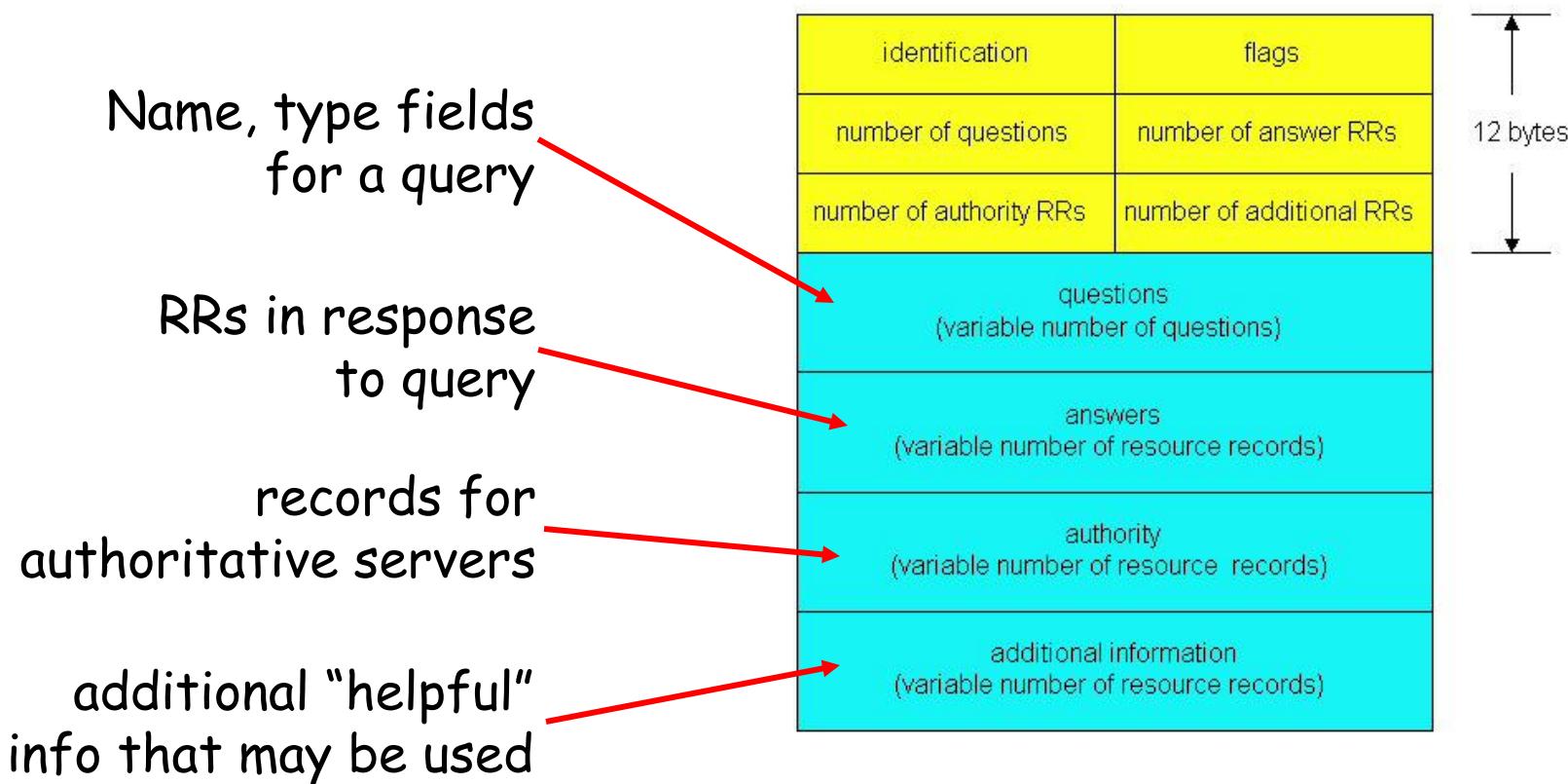
**DNS协议:** 定义了查询和响应两种报文，查询和响应使用相同的报文格式

报头

- r **identification:** 16 bit #  
for query, reply to query  
uses same #
- r **flags:**
  - ❖ query or reply
  - ❖ recursion desired (q)
  - ❖ recursion available (r)
  - ❖ reply is authoritative



# DNS协议，报文



# DNS报文的封装

- r DNS主要使用UDP，有时使用TCP，服务器的熟知端口都是53：
  - ❖ 当响应报文的长度小于512字节时，使用UDP
  - ❖ 当响应报文的长度超过512字节时，使用TCP
  - ❖ 当解析器事先不知道响应报文的长度时，先使用UDP；若响应报文的长度超过512字节，服务器截断这个报文，置DNS报文首部的TC标志为1；解析程序打开TCP连接，并重复这个请求，以便得到完整的响应
- r 思考：为什么DNS主要使用UDP？

# 往DNS中插入资源记录

- r example: new startup “Network Utopia”
- r 向DNS注册机构注册域名“**networkutopia.com**”
  - ❖ 提供权威**DNS**服务器（主域名服务器，辅助域名服务器）的名字和**IP地址**
  - ❖ 对每个权威域名服务器，注册机构往 **com TLD** 服务器中插入两条资源记录，例如：  
**(networkutopia.com, dns1.networkutopia.com, NS)**  
**(dns1.networkutopia.com, 212.212.212.1, A)**
- r 建立权威**DNS**服务器，特别是：
  - ❖ 建立**www.networkutopia.com**的**Type A**记录
  - ❖ 建立**networkutopia.com**的**Type MX**记录，以及相应邮件服务器的**A**记录

# DNS的安全问题

- r 历史上重大的断网事件，主要都是由根服务器受到攻击引起的：
  - ❖ 2016年10月22日，美国最主要DNS服务商Dyn遭遇史上最严重DDoS攻击，大半个美国的用户集体断网
- r 谁控制了根服务器，谁就控制了互联网：
  - ❖ 我国没有根服务器，因此中国互联网的命脉掌握在别人手里
- r 其它攻击手段：DNS缓存投毒，DNS劫持等
- r 僵尸网络通常利用DNS实现僵尸主机与C&C服务器通信

# 小结

## r DNS

- ❖ 提供了一种按层次结构**命名主机的方法**
- ❖ 实现了一个由**DNS**服务器构成的**分布式数据库**
- ❖ 提供了查询域名数据库的**应用层协议**

## r 域名服务器的类型和层次（逻辑层次，物理层次）

## r DNS服务的调用方法：

- ❖ 向本地**DNS**代理的一个**RPC**调用
- ❖ 递归+迭代的查询方式

## r DNS协议：

- ❖ 主要使用**UDP**，也可以使用**TCP**，端口号均为**53**
- ❖ 报文请求/响应交互

## r DNS缓存

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充: File Transfer and  
FTP

2.3 electronic mail

- ❖ SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# P2P文件共享

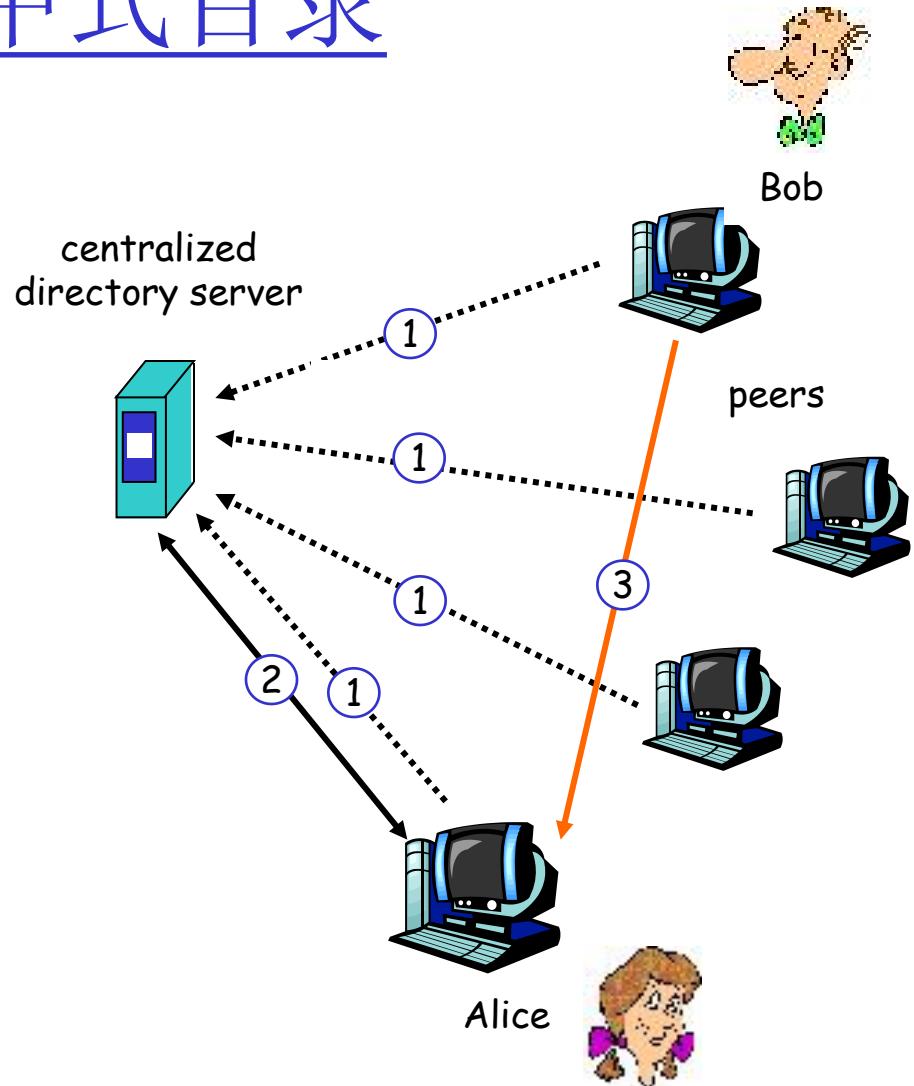
## 一个典型的应用例子

- r Alice在她的笔记本电脑上运行P2P客户应用
- r 通过ISP连接到因特网上
- r 请求歌曲“Hey Jude”
- r P2P客户应用显示拥有该歌曲拷贝的对等方列表
- r Alice选择其中的一个对等方，比如Bob
- r 文件从Bob的PC机下载到Alice的笔记本电脑
- r 当Alice从Bob的PC机下载时，其他用户可能从Alice的笔记本电脑下载
- r Alice的P2P应用程序既是一个Web客户，又是一个临时的Web服务器

# P2P应用架构: 集中式目录

第一个P2P文件共享服务  
**Napster**的设计：

- 1) 每个对等方与一个集中式的目录服务器连接，告知：
  - ❖ 自己的**IP**地址
  - ❖ 可以共享的内容
- 2) Alice查询歌曲“Hey Jude”
- 3) Alice从Bob下载音乐文件



# P2P: 集中式目录的问题

- r 单点失效
- r 目录服务器成为性能瓶颈
- r 由于版权问题，易成为诉讼的对象

file transfer is decentralized, but locating content is highly centralized

# P2P应用架构：查询洪泛

## Gnutella

- r fully distributed
  - ❖ no central server
- r public domain protocol
- r many Gnutella clients implementing protocol

## overlay network: graph

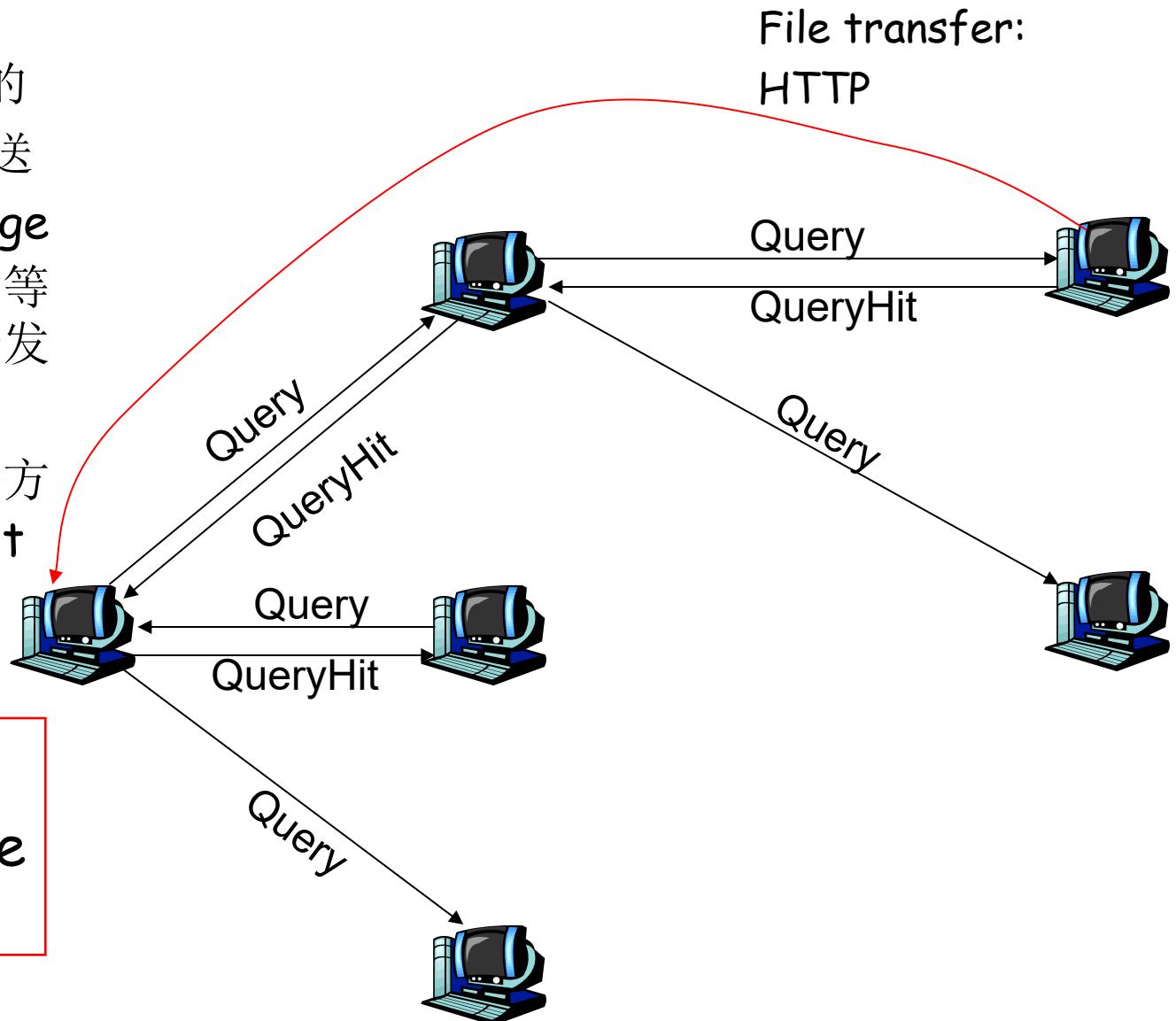
- r 如果对等方X与Y之间存在一条TCP连接，称它们之间存在一条边
- r 所有活跃的对等方以及它们之间的边，形成一个覆盖网络（**overlay net**）
- r 边：虚拟链路
- r 一个对等方通常有 < 10 个的覆盖网邻居

# Gnutella: Peer joining

1. joining peer Alice must find another peer in Gnutella network: use list of candidate peers
2. Alice sequentially attempts TCP connections with candidate peers
3. **Flooding:** Alice sends Ping message to each overlay neighbor; each neighbor forwards Ping message to his overlay neighbors ...
  - peers receiving Ping message respond to Alice with Pong message
4. Alice receives many Pong messages, and can then setup additional TCP connections

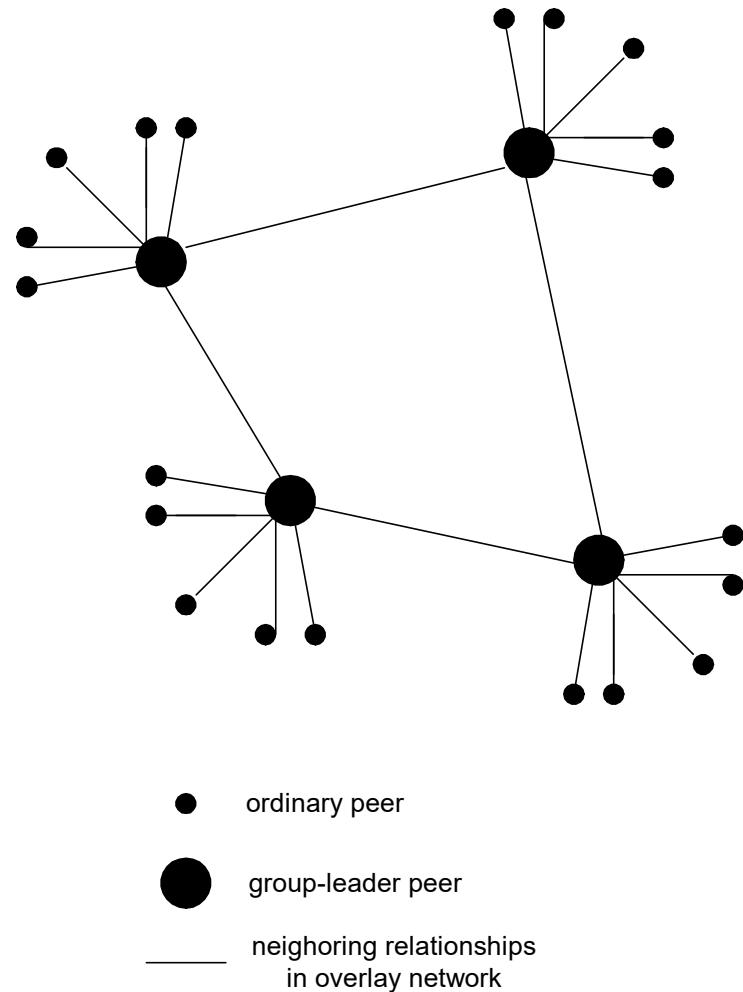
# Gnutella: protocol

- r 查询方在已有的TCP连接上发送**Query message**
- r 收到消息的对等方向其邻居转发查询报文
- r 有内容的对等方返回**QueryHit**



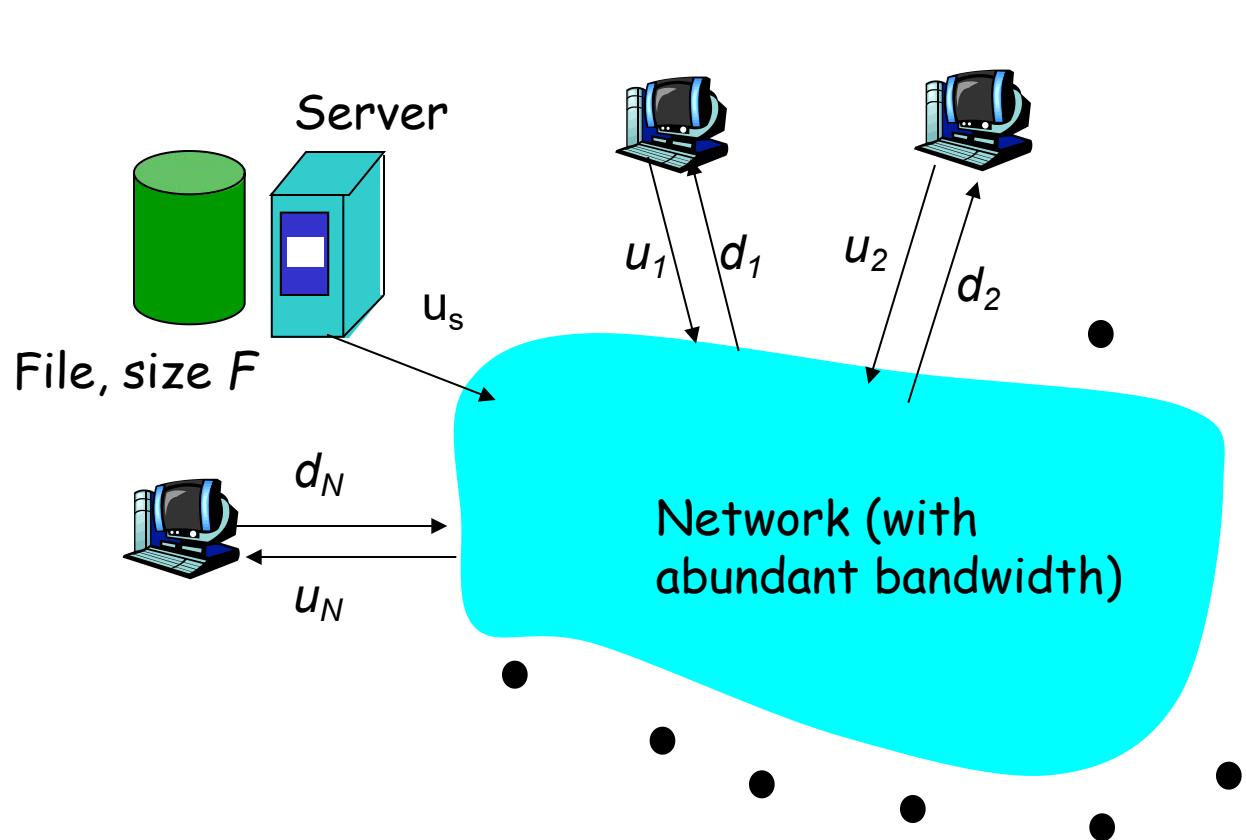
# P2P应用架构：层次结构的覆盖网

- r 介于集中式目录和查询洪泛之间
- r 每个对等方或是一个超级节点 (*group leader*)，或是从属于某个超级节点的普通节点：
  - ❖ 每个普通节点与其超级节点之间建立TCP连接
  - ❖ 某些超级节点之间建立TCP连接
- r 超级节点知道其所有从属节点的共享内容列表



# 比较客户-服务器和P2P架构

**Question**：将文件从一台服务器分发到网络中的 $N$ 个终端，需要多少时间？



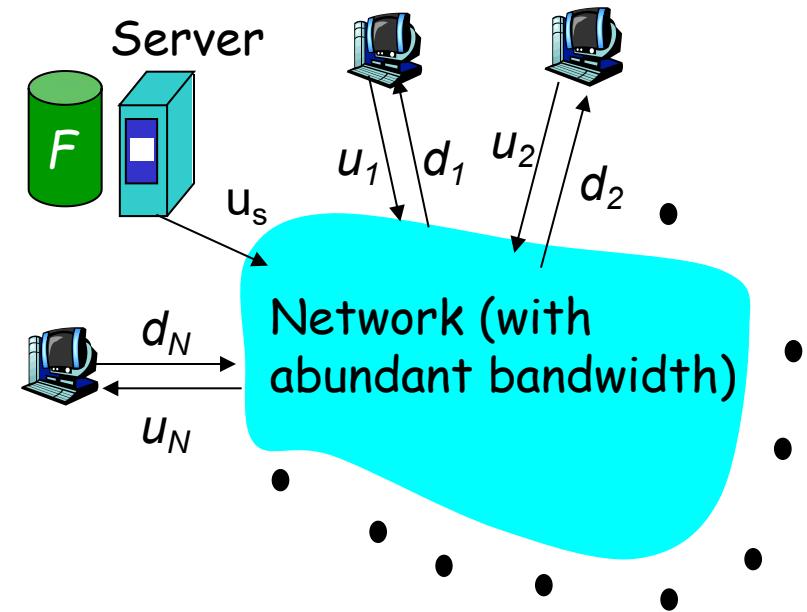
$u_s$ : 服务器上传带宽

$u_i$ : 第  $i$  个终端的上  
传带宽

$d_i$ : 第  $i$  个终端的下  
载带宽

# 客户-服务器架构的文件分发时间

- r 假设：
  - ❖ 所有瓶颈都在接入链路
  - ❖ 服务器和客户的上传和下载带宽全都用于分发文件
- r 服务器顺序地发送  $N$  个文件拷贝，耗时  $NF/u_s$
- r 第  $i$  个客户花费  $F/d_i$  下载文件



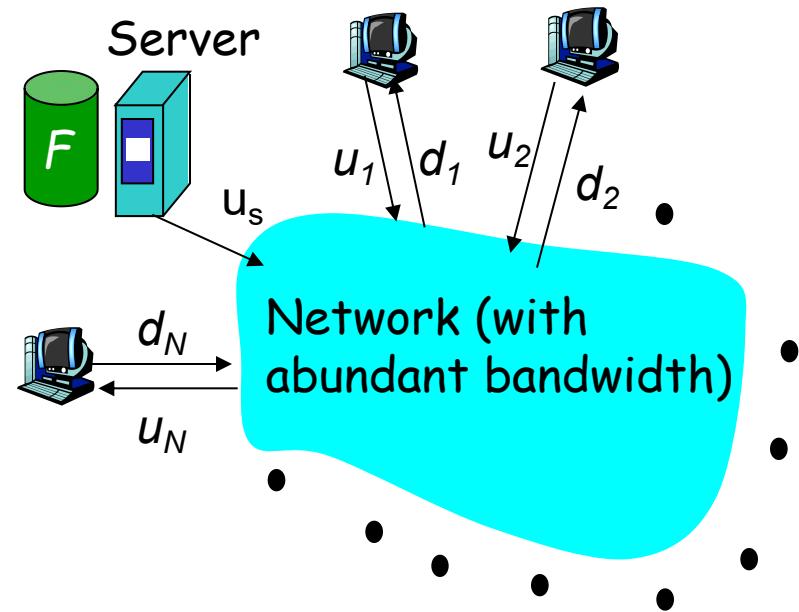
将文件  $F$  分发给  $N$  个客户的耗时

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

increases linearly in  $N$   
(for large  $N$ )

# P2P架构的文件分发时间

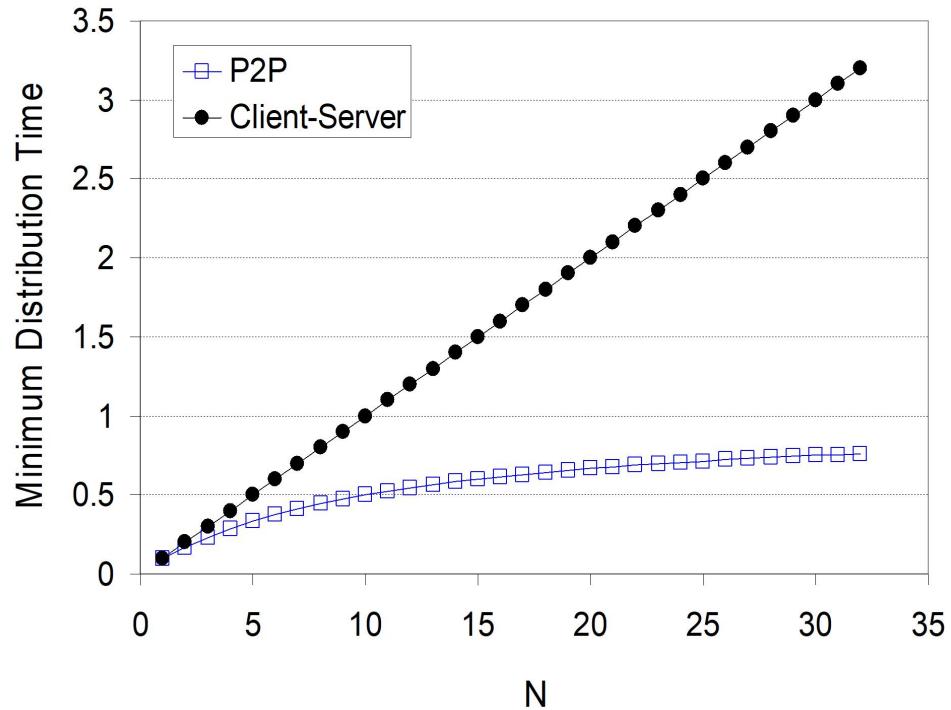
- ✓ 服务器至少必须发送一个文件拷贝，耗时  $F/u_s$
- ✓ 第  $i$  个客户至少耗时  $F/d_i$  下载文件
- ✓ 总共  $NF$  bits 必须在网络中被上传/下载，网络中总的上传带宽为:  $u_s + \sum u_i$



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

# 文件分发时间比较

- r 假设：
  - ❖ 客户的上传带宽均为 $u$ ,  
 $F/u = 1 \text{ hour}$ ,  $us = 10u$ ,  
 $d_{min} > us$  (下载速率不是瓶颈)
- r 客户-服务器架构：
  - ❖ 分发时间随 $N$ 线性增大
- r P2P架构：
  - ❖ 最小分发时间总是小于客户-服务器架构
  - ❖ 对于任意的 $N$ , 最小分发时间总是小于1小时



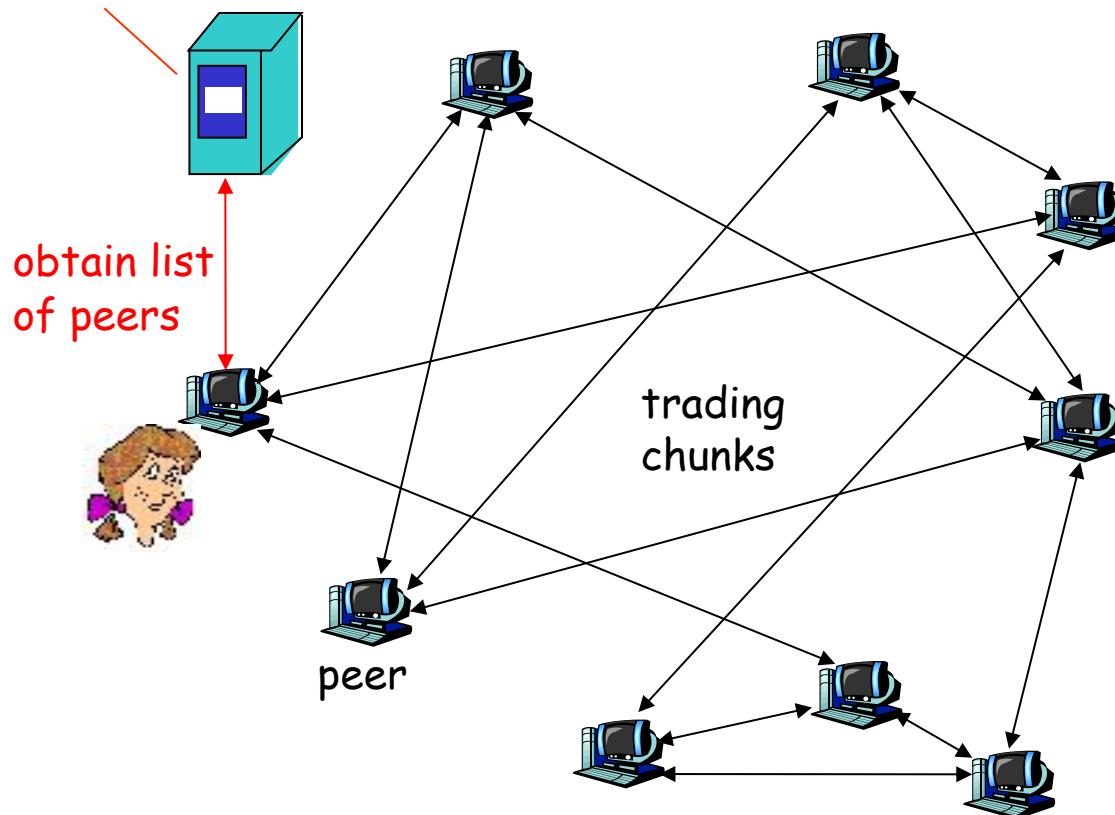
# P2P案例学习: BitTorrent

tracker:

跟踪洪流中的对等方

Torrent (洪流):

参与一个特定文件分发的对等方集合

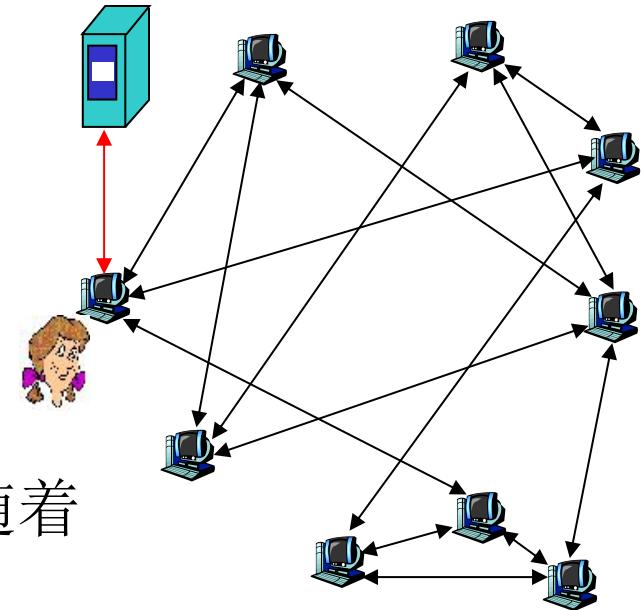


**对等方加入洪流:**

- 向跟踪器注册，获得一个对等方列表
- 尝试向列表中的对等方建立TCP连接

# BitTorrent (概述)

- r 文件被划分成长为**256KB**的块
- r 对等方加入洪流时没有数据块，但随着时间的推移逐步积累
- r 对等方在下载数据块的同时，也向其它对等方上载数据块
- r 对等方可以动态加入或离开系统
- r 一旦对等方获取了整个文件，它可以（自私地）离开，也可以（无私地）留在网络中，为其它对等方上传文件块



# BitTorrent (操作)

## Pulling Chunks

- r 在任何给定时刻，不同的对等方拥有不同的数据块子集
- r 周期性地，对等方（如 **Alice**）询问每个邻居他们拥有的数据块集合
- r **Alice**向邻居请求她缺少的数据块：
  - ❖ **最稀罕优先**：优先请求在邻居中拷贝数量最少的数据块

## Sending Chunks: tit-for-tat

- r **Alice**选择当前向其发送数据最快的**4**个邻居，响应他们的数据块请求
  - r 每隔**10**秒，重新评估向其提供数据最快的**4**个邻居
- r 每隔**30**秒，随机选择另一个对等方（如**Bob**）响应其请求
  - ❖ **Alice**可能成为向**Bob**上载最快的**4**个邻居之一
  - ❖ **Bob**也可能成为向**Alice**上载最快的**4**个邻居之一

# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充: File Transfer and  
FTP

2.3 electronic mail

- ❖ SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# 视频流和内容分发网络

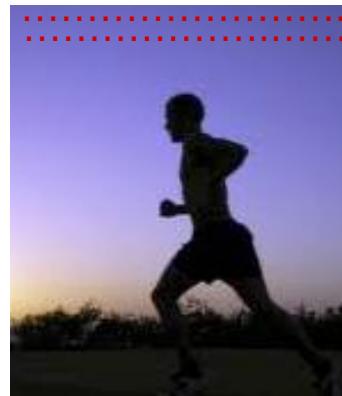
- 视频已成为因特网流量的主体：
  - 2015年，Netflix和YouTube分别消耗了住宅ISP流量的 37% 和 16%
  - 有大约10亿YouTube用户、7500万Netflix用户
- 挑战一：如何能够支持10亿用户？
  - 单台视频服务器不可行
- 挑战二：网络环境异构
  - 不同用户的接收能力不同，例如，使用有线网络或无线网络，高带宽或低带宽
- 解决方案：采用分布式应用层基础设施



# 视频(video)的概念

- ✓ 视频是以恒定速率播放的图像序列：
  - ❖ 比如，**24帧/秒**
- ✓ 数字图像是一个像素阵列
  - ❖ 每个像素用一些比特来表示
- ✓ 图像编码技术利用图像的帧内冗余和帧间冗余来减少需要使用的比特数：
  - ❖ 空间冗余：帧内
  - ❖ 时间冗余：帧间

*spatial coding example:* instead of sending  $N$  values of same color (all purple), send only two values: color value (*purple*) and number of repeated values ( $N$ )



frame  $i$

*temporal coding example:* instead of sending complete frame at  $i+1$ , send only differences from frame  $i$

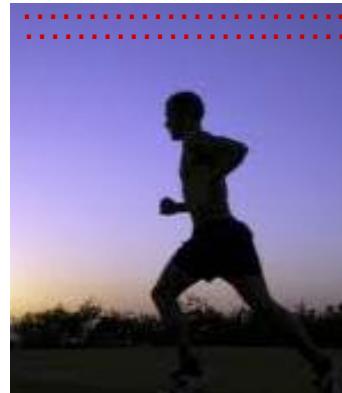


frame  $i+1$

# 视频编码速率

- CBR: (constant bit rate):
  - 编码速率固定
- VBR: (variable bit rate):
  - 编码速率可变
- 视频编码标准:
  - MPEG I (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

*spatial coding example:* instead of sending  $N$  values of same color (all purple), send only two values: color value (purple) and number of repeated values ( $N$ )



frame  $i$

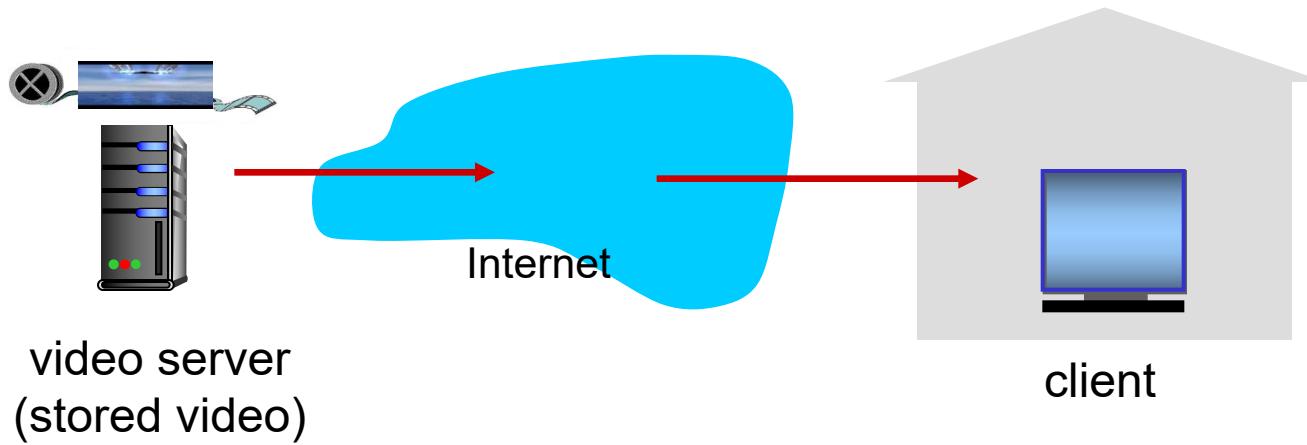
*temporal coding example:* instead of sending complete frame at  $i+1$ , send only differences from frame  $i$



frame  $i+1$

# 流式存储视频：一个简单的实现

一个简单的实现：



- 视频作为一个普通文件，保存在 HTTP服务器中
- 服务器与客户建立TCP连接，发送文件
- 视频应用周期性地从应用缓存中取帧、解码、展示
- 问题：所有客户使用相同的编码速率

# 流式多媒体: DASH

- r **DASH: Dynamic, Adaptive Streaming over HTTP**
- r 服务器:
  - ❖ 将视频文件划分成多个块
  - ❖ 每个块以不同的码率编码和存储
  - ❖ 元文件为不同的块提供URL
- r 客户:
  - ❖ 周期性地测量服务器到客户的网络带宽
  - ❖ 查询元文件，每次请求一个块
    - 选择当前带宽可支持的最大码率
    - 不同时刻可以选择不同码率的块
- r 客户端能够“智能地”确定:
  - ❖ 什么时候请求块（避免缓存不足或溢出）
  - ❖ 请求什么码率的块（获得当前最好的视频质量）
  - ❖ 向谁请求块（离客户最近或具有最高带宽的URL）

# 如何将内容流式传输给同时在线的大量用户？

## 方法1: 使用一个巨型服务器（不可行）

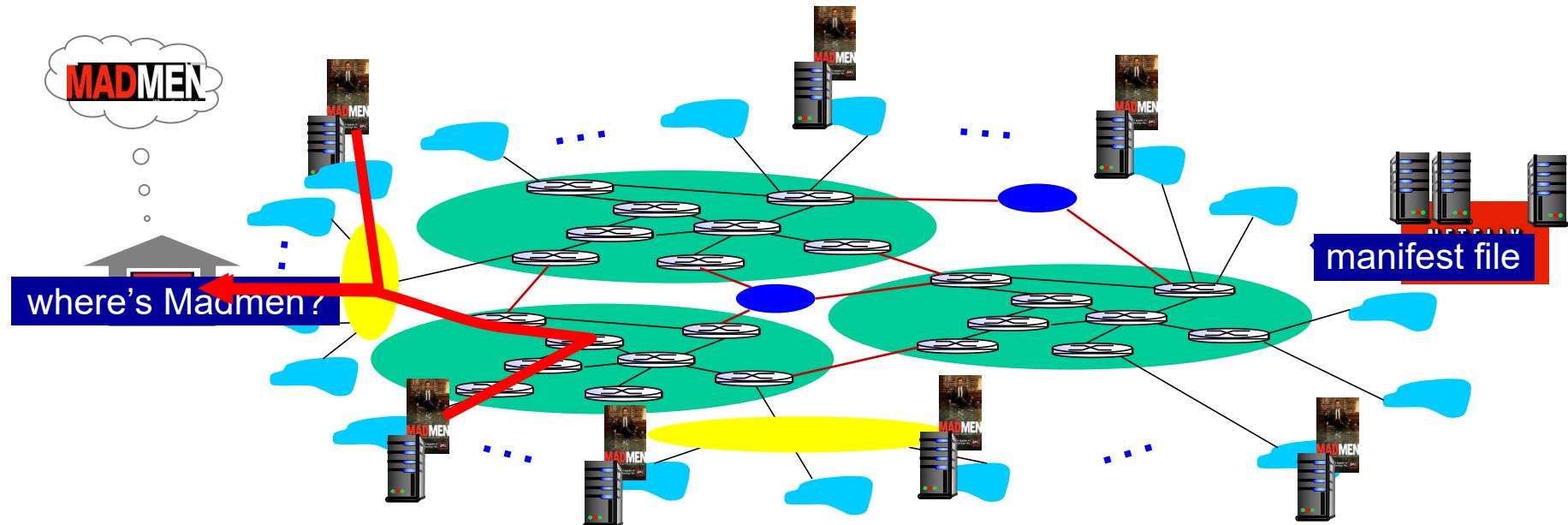
- ❖ 单点故障
- ❖ 网络拥塞点
- ❖ 远端用户传输距离长，跨多个**ISP**，带宽低
- ❖ 同一条链路上传输多个视频拷贝，浪费带宽

## 方法2: 在地理上分布的多个站点存储和提供服务 (**CDN**):

- ❖ *enter deep*: 将**CDN**服务器深入部署到大量接入网中，靠近用户
  - Akamai拥有1700个站点
- ❖ *bring home*: 将少量（几十个）较大的集群部署在靠近接入网的POP中
  - Limelight采用此法

# 内容分发网络(CDN)

- 在多个CDN站点存储内容的拷贝
- 用户从CDN请求内容：
  - 用户请求被定向到附近的站点，获取内容
  - 或网络拥塞，可向不同的站点请求内容拷贝



# 内容分发网络(CDN)



## OTT挑战: 如何应对拥塞的因特网

- ❖ 从哪个CDN站点获取内容?
- ❖ 面对拥塞时观看者的行为?
- ❖ 将内容放置在哪个CDN节点?

more .. in chapter 7

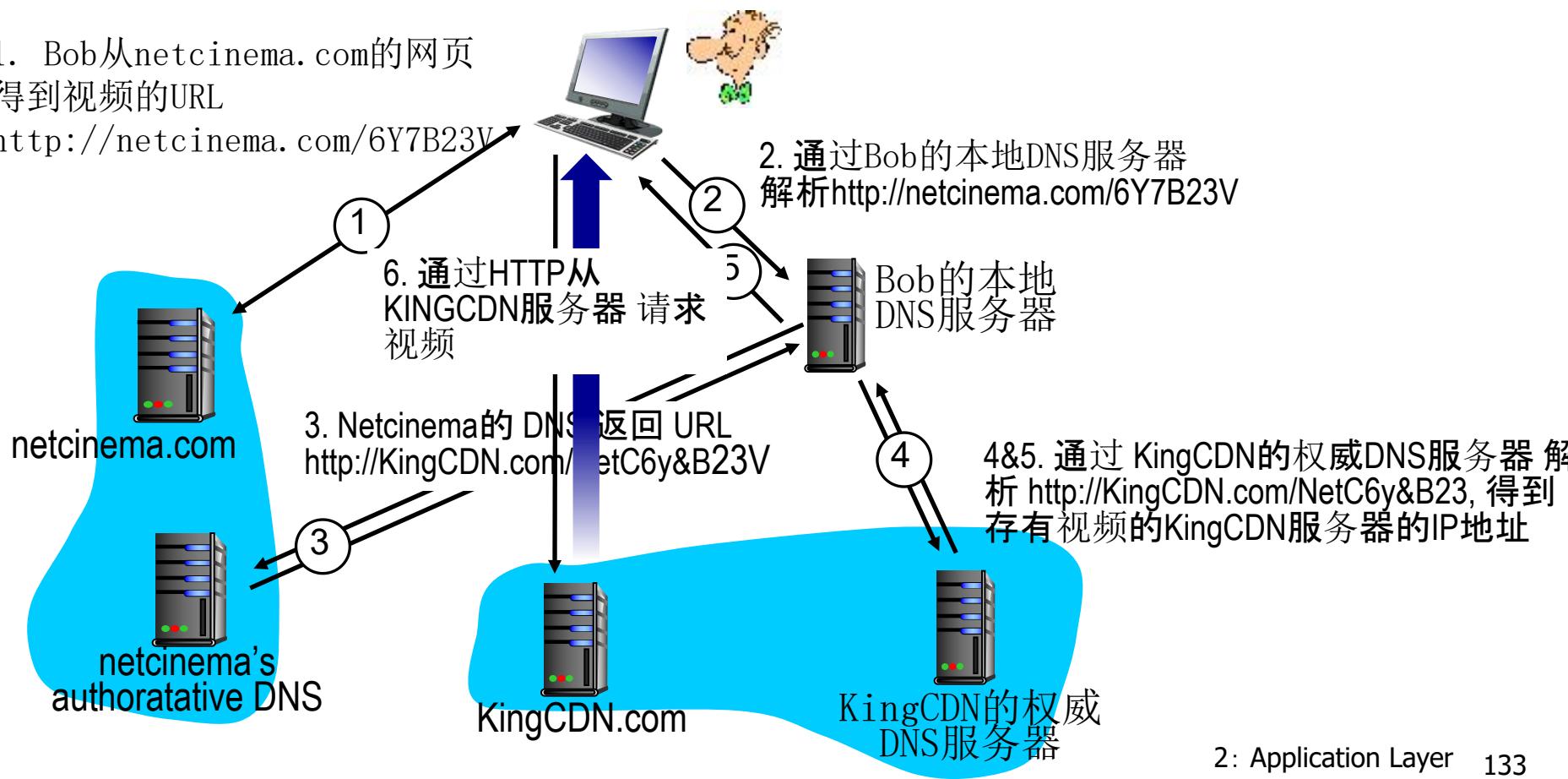
# CDN内容访问: a closer look

Bob (client) 从 <http://netcinema.com/6Y7B23V> 请求视频

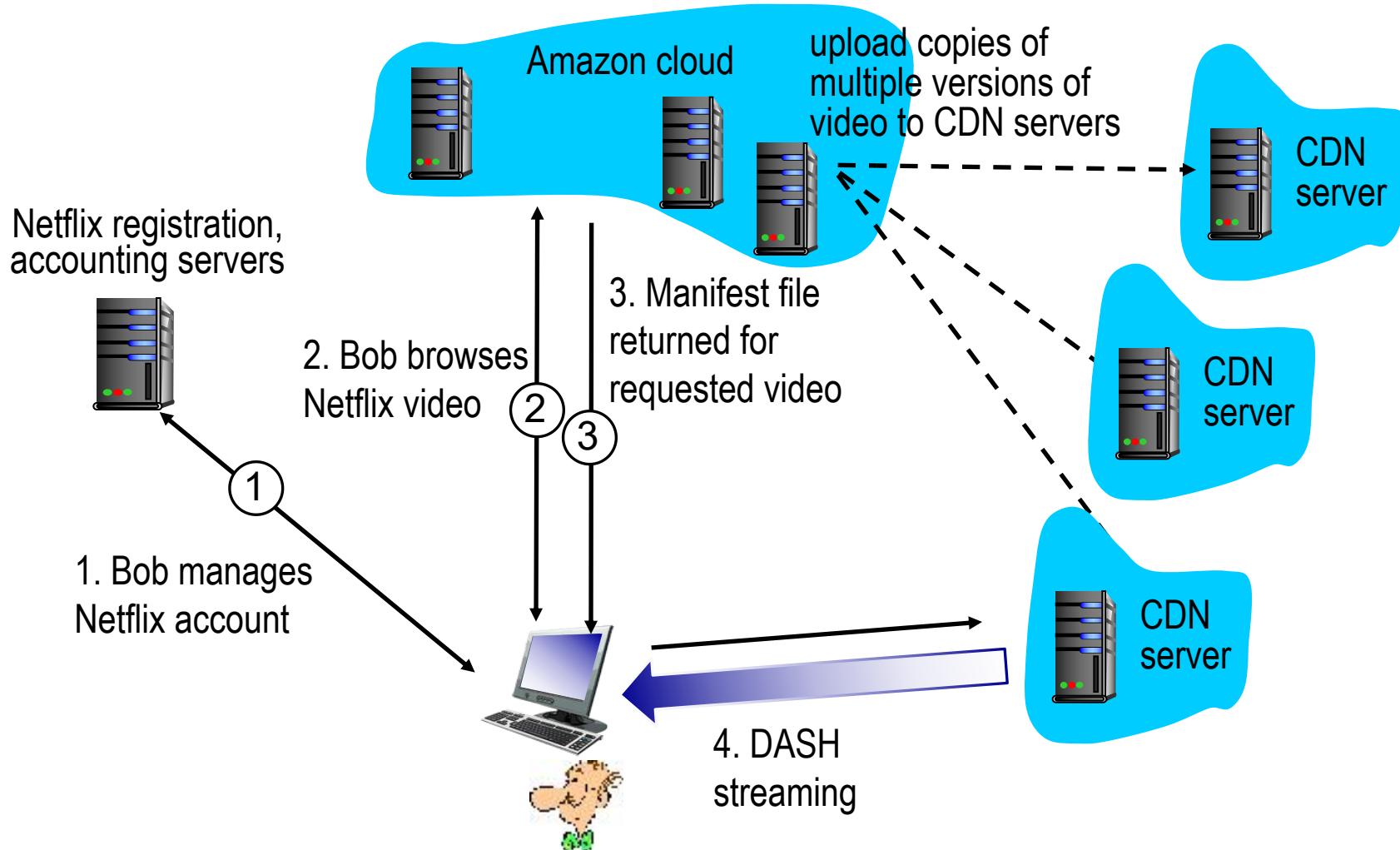
- 视频保存在 <http://KingCDN.com/NetC6y&B23V>

1. Bob从netcinema. com的网页  
得到视频的URL

<http://netcinema. com/6Y7B23V>



# Case study: Netflix



# Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充: File Transfer and  
FTP

2.3 electronic mail

- ❖ SMTP, POP3, IMAP

2.4 DNS

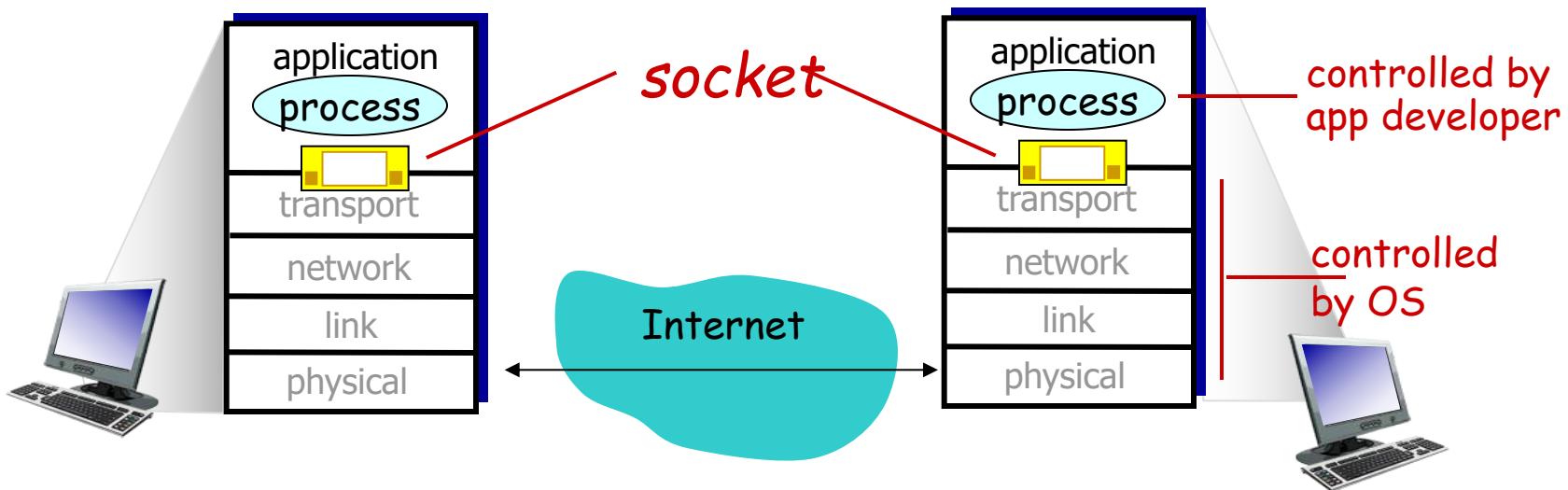
2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# 套接字编程

- r 创建网络应用程序，需要：
  - ❖ 编写一个客户程序、一个服务器程序
  - ❖ 客户和服务器可以利用套接字收、发报文
- r **套接字：应用进程和传输层之间的“门”**



# Socket API

- r 1981年在**BSD UNIX 4.1**中引入，此后成为编写因特网程序的标准接口
- r 应用需显式地创建、使用和释放套接字
- r 采用客户-服务器模式
- r 应用通过**socket API**可以调用两种传输服务：
  - ❖ 不可靠的数据报服务：由**UDP**协议实现
  - ❖ 可靠的字节流服务：由**TCP**协议实现

# 本节使用的一个应用例子

1. 客户程序从键盘读入一行字符（数据），发送给服务器
2. 服务器接收数据，将小写字符转换成大写字符
3. 服务器将修改后的数据发送给客户
4. 客户接收修改后的数据，在屏幕上显示出来

## 2.7.1 UDP套接字编程

### server (running on serverIP)

在端口x上创建套接字

**serverSocket**

从**serverSocket**

读UDP报文

写UDP响应报文到

**serverSocket**

### client

创建套接字

**clientSocket**

用**serverIP** 和端口x创建UDP报文  
交给**clientSocket**

从**clientSocket**

读UDP报文

关闭**clientSocket**

# Example app: UDP server

## Python UDPServer

```
from socket import *
serverPort = 12000
create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port  
number 12000 → serverSocket.bind(('', serverPort))
print "The server is ready to receive"
loop  
forever → while 1:
Read from UDP socket → message, clientAddress = serverSocket.recvfrom(2048)
into message, getting  
client's address (client IP  
and port) → modifiedMessage = message.upper()
send upper case string → serverSocket.sendto(modifiedMessage, clientAddress)
back to this client
```

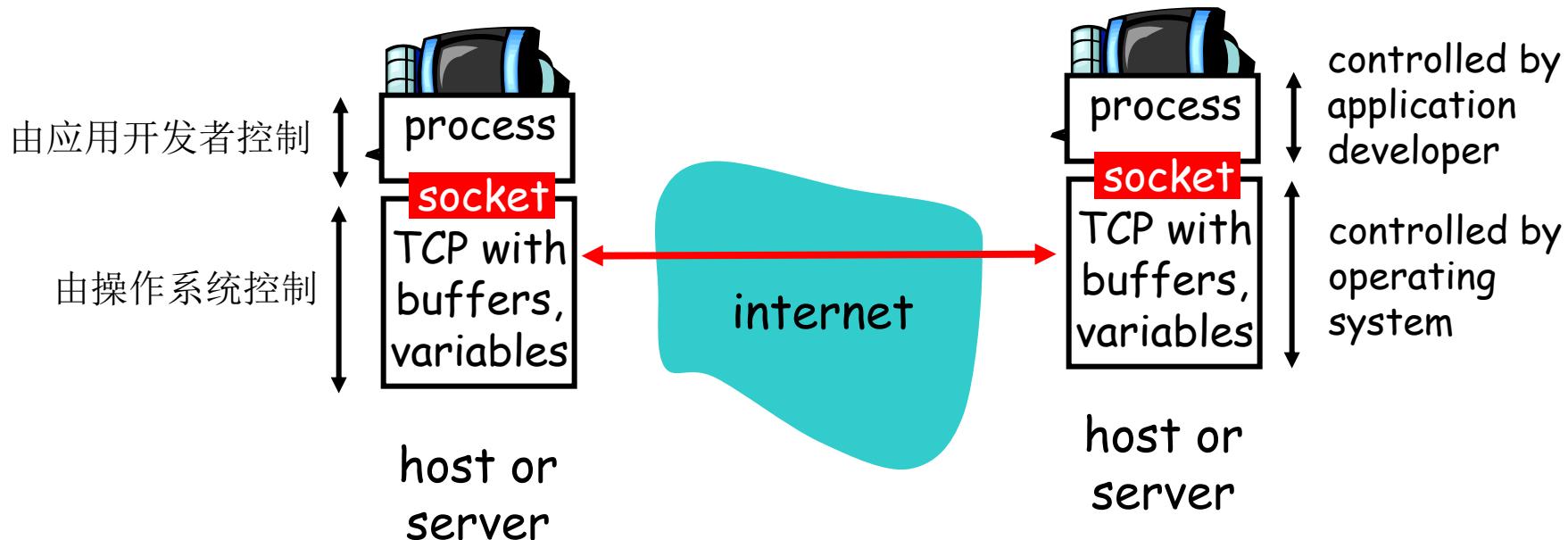
# Example app: UDP client

## Python UDPClient

```
include Python's socket  
library → from socket import *  
  
create UDP socket for  
server → clientSocket = socket(socket.AF_INET,  
                                socket.SOCK_DGRAM)  
  
get user keyboard  
input → message = raw_input('Input lowercase sentence:')  
  
Attach server name, port  
to message; send into  
socket → clientSocket.sendto(message,(serverName, serverPort))  
  
read reply characters  
from  
socket into string → modifiedMessage, serverAddress =  
                                clientSocket.recvfrom(2048)  
  
print out received string  
and close socket → print modifiedMessage  
                                clientSocket.close()
```

## 2.7.2 TCP套接字编程

- r 可将**TCP**连接想像成是一对套接字之间的一条封闭管道：
  - ❖ 发送端**TCP**将要发送的字节序列从管道的一端（套接字）送入
  - ❖ 接收端**TCP**从管道的另一端（套接字）取出字节序列
  - ❖ 在管道中传输的字节不丢失，并保持顺序



# 服务器使用多个套接字服务客户

- r 服务器进程在**欢迎套接字**上等待客户的连接请求
- r 客户进程需要通信时，创建一个客户套接字，与服务器**欢迎套接字**通信：
  - ❖ 在此过程中，客户**TCP**向服务器**TCP**发送连接请求
- r 服务器进程创建一个临时套接字（称**连接套接字**）和一个新的服务器进程，与客户进程通信
- r 服务器进程回到**欢迎套接字**上继续等待
  - ❖ 允许服务器同时服务多个客户
- r 客户服务结束后，服务器销毁进程，关闭连接套接字

# 客户-服务器交互: TCP

## Server (running on **hostid**)

在端口x上创建欢迎套接字:

```
welcomeSocket =  
    ServerSocket()
```

等待连接请求  
**TCP**  
connectionSocket =

```
welcomeSocket.accept()
```

从connectionSocket  
读服务请求

写响应到  
connectionSocket

关闭connectionSocket

## Client

创建本地套接字

```
clientSocket = Socket()
```

连接到 **hostid, port=x**

使用clientSocket  
发送服务请求

从clientSocket读响应

关闭clientSocket

# Example app: TCP server

## Python TCPServer

```
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 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming  
socket → from socket import \*

server begins listening  
for incoming TCP  
requests → serverPort = 12000

loop  
forever → serverSocket = socket(AF\_INET,SOCK\_STREAM)

server waits on accept()  
for incoming requests, new  
socket created on return → serverSocket.bind(("",serverPort))

read bytes from socket  
(but not address as in  
UDP) → serverSocket.listen(1)

close connection to this  
client (but *not* welcoming  
socket) → print 'The server is ready to receive'

→ while 1:

→ connectionSocket, addr = serverSocket.accept()

→ sentence = connectionSocket.recv(1024)

→ capitalizedSentence = sentence.upper()

→ connectionSocket.send(capitalizedSentence)

→ connectionSocket.close()

# Example app: TCP client

## Python TCPClient

```
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)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for  
server, remote port  
12000

No need to attach  
server name, port

# UDP服务与TCP服务

---

## UDP

- r 报文传输服务
- r 由于没有建立管道，应用程序发送每个报文必须给出远程进程地址
- r 服务器使用一个进程和一个套接字为所有客户服务，一次请求-响应完成一次服务

## TCP

- r 字节流传输服务
- r 由于建立了管道，应用程序只需向套接字中写入字节序列，不需指出远程进程地址
- r 服务器为每个客户单独生成一个套接字和一个新进程，允许双方长时间通信

# Chapter 2: Summary

- r 应用架构
  - ❖ client-server
  - ❖ P2P
- r 应用服务需求:
  - ❖ *reliability, bandwidth, delay, security*
- r 因特网传输服务模型
  - ❖ 面向连接, 可靠: TCP
  - ❖ 不可靠, 数据报: UDP
- r 应用典型地采用请求/响应方式进行交互:
  - ❖ 客户请求信息或服务
  - ❖ 服务器用数据或状态码进行响应
- r 报文格式:
  - ❖ 报头: 携带元数据
  - ❖ 实体: 携带数据本身
- r 套接字编程:
  - ❖ TCP
  - ❖ UDP

# 作业

1. 习题: 1, 3, 7, 8, 9
2. HTTP实验
3. DNS实验

- r 作业提交时间:
- ❖ **HTTP实验: 9月24日**
  - ❖ 习题:
  - ❖ DNS实验:

# 课后思考

1. 了解什么是web 1.0和web2.0。
2. 内容分发网络（Content Delivery Network, CDN）为何能加快内容请求的速度？
3. 查看一下本机浏览器中的cookie文件
4. 了解1~2种与DNS相关的网络攻击，它们是怎么做到的？