

第2章 应用层

网络应用是计算机网络的重要功能之一。

- ✓ 20世纪80年代：基于文本的电子邮件、文件传输、文本聊天等等。
- ✓ 20世纪90年代：Web应用、IP电话、视频会议等。
- ✓ 20世纪末：即时讯息、P2P对等文件共享。
- ✓ 2000年以后，应用爆炸式发展：VoIP、Skype、自媒体（YouTube、优酷）、点播（Netflix）、在线游戏

学习目标和主要内容

学习目标

网络应用的原理和实现方面的知识。

主要内容

- ✓ **应用层概念**：应用层协议、客户机与服务器、进程、套接字和运输层接口。
- ✓ **应用程序**：Web、文件传输、电子邮件、域名系统DNS及P2P对等文件共享。
- ✓ **开发网络应用程序的方法**：套接字API编程。



本章内容

2.1 应用层协议原理

2.2 Web和HTTP

2.3 FTP

2.4 电子邮件

2.5 DNS

2.6 P2P文件共享

2.7 TCP的套接字 编程

2.8 UDP的套接字 编程

2.9 小结

学生能够自己构建一个 Web 服务器

本章要求

掌握：

- (1)应用层协议的原理
- (2)应用层协议的实现机制
- (3)Web应用和HTTP协议
- (4)FTP协议的实现机制
- (5)DNS的功能和实现方法
- (6)电子邮件系统的构成、
传输机制和协议
- (7)TCP和UDP套接字编程
- (8)P2P文件共享原理

理解：

- (1)HTTP协议的报文格式
- (2)FTP协议的报文格式
- (3)DNS记录和报文格式

了解：。

- (1)构造一个简单的Web服务器
 - (2)P2P文件共享协议实现过程
- 自学：
其它应用层协议

作业要求

- ❑ 详细设计文档
- ❑ 源码
- ❑ 可执行代码
- ❑ 打包提交，文件名为：
学号-姓名-第一次作业
- 作业提交地址：
173435880@qq.com

服务器基本流程：

- ❑ 创建套接字
- ❑ 获取HTTP请求，并解析
HTTP请求报文
- ❑ 显示请求报文各字段的字段
名和值，并进行说明
- ❑ 根据HTTP请求报文获得对象
文件路径名
- ❑ 根据路径名打开本地文件
- ❑ 封装本地文件到HTTP响应报
文
- ❑ 使用套接字发送HTTP相应报
文

流行的网络应用程序

- ❑ E-mail
- ❑ Web
- ❑ 即时讯息
- ❑ 远程注册
- ❑ P2P文件共享
- ❑ 多用户网络游戏
- ❑ 流式存储视频片段

- ❑ 因特网电话
- ❑ 实时视频会议
- ❑ 大规模并行计算

虚拟现实VR
大数据分析



编制应用程序

微软的虚拟全息可视头戴设备HoloLens



http://www.iqiyi.com/v_19rrntlh80.html

2.1 应用层协议原理

网络应用程序的研发要点

- ❑ 写出能够分别在不同端系统运行，并通过网络相互通信的程序。

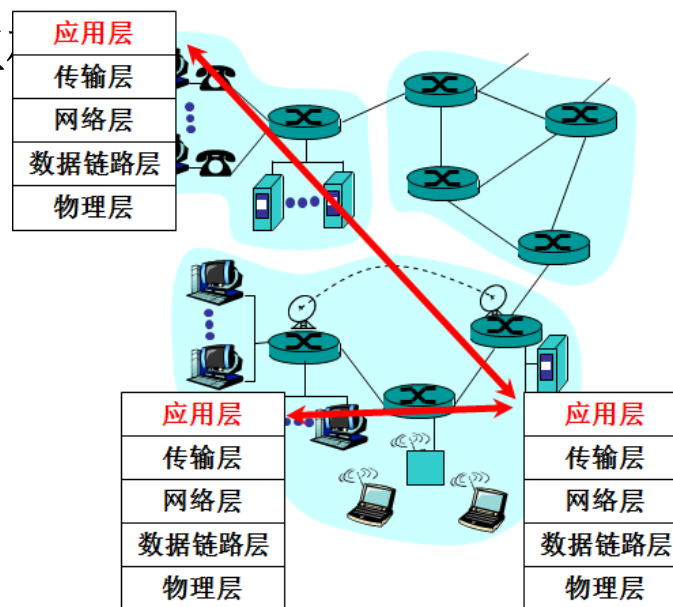
如Web应用程序，由两个可以相互通信的程序组成

- ✓ 浏览器程序：运行在用户主机上；
- ✓ Web服务器程序：运行在Web服务器主

- ❑ 应用程序软件只在端系统运行，不需在网络核心设备上运行。

- 网络核心设备无应用层，只有较低层。

如图2-1。



本节内容

2.1.1 网络应用程序体系结构

2.1.2 进程通信

2.1.3 可供应用程序使用的运输服务

2.1.4 因特网提供的运输服务

2.1.5 应用层协议

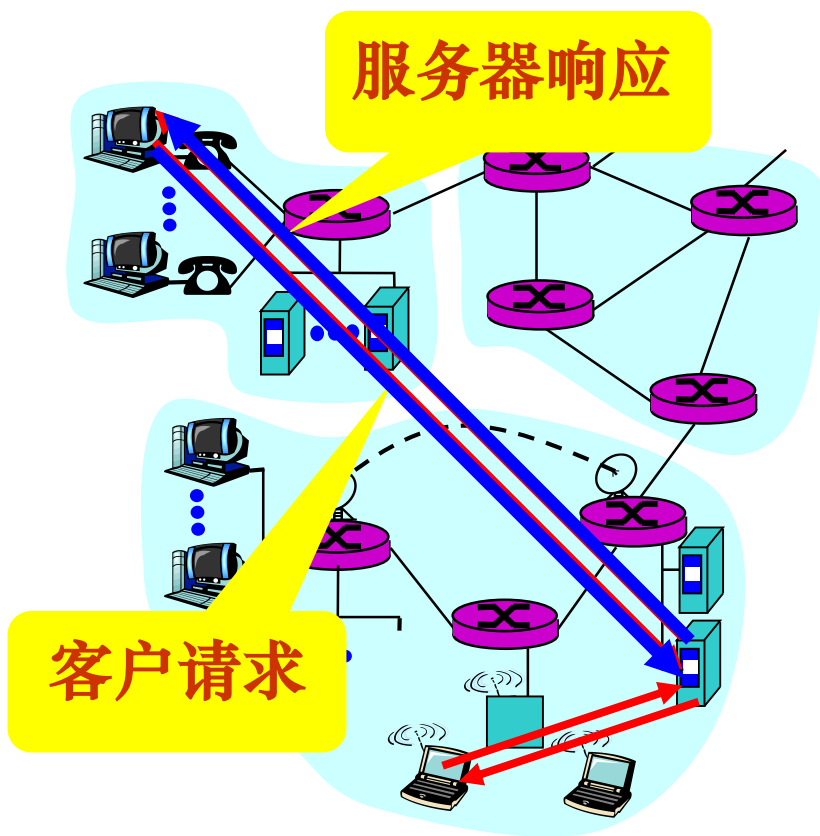
2.1.6 本书涉及的网络应用

Web、文件传输、电子邮件、目录服务、对等文件共享等五个。

2.1.1 网络应用程序体系结构

- ❑ 应用程序体系结构：规定如何在各种端系统上组织应用程序，由研发者设计。
- ❑ 三种类型：
 - ✓ 客户机/服务器
 - ✓ 对等 (P2P)
 - ✓ 客户机/服务器与P2P的混合

1、客户机/服务器体系结构



服务器：

- 总是打开
- 为多个客户机请求提供服务
- 永久的IP地址
- 可扩展为服务器场（主机群集）

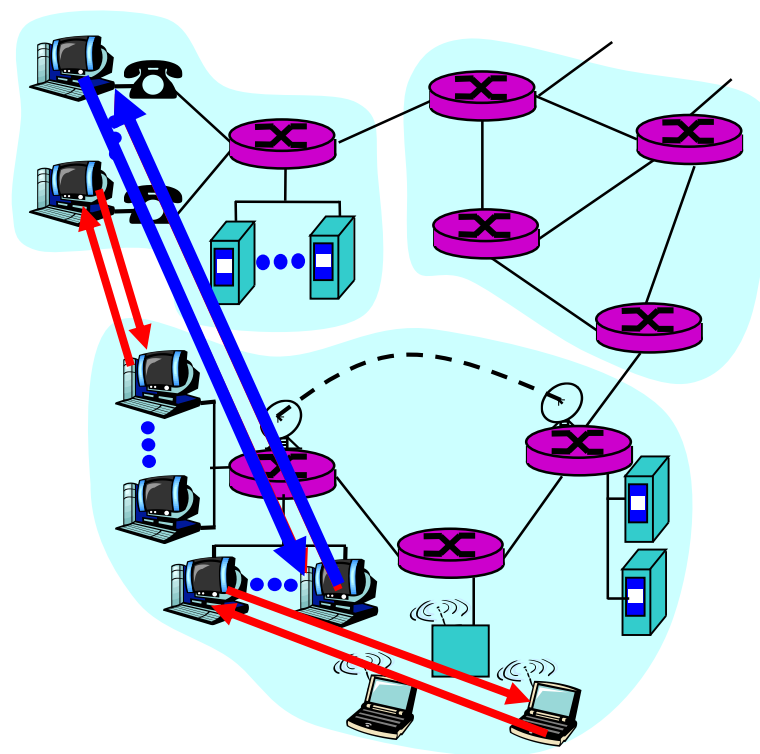
客户机：

- 总是打开或间歇打开
- 向服务器发出请求
- 具有动态的IP地址
- 彼此之间不直接通信

如Web应用程序：总是打开的Web服务器为运行在客户机主机上的浏览器的请求提供服务（接收客户机请求，并发送响应结果）。

2、纯P2P体系结构（peer-to-peer）

- 无（最少）打开的服务器
- 任意端系统（对等方）可以直接通信
- 对等方间歇地连接，IP地址不固定
- 例：文件分发、因特网电话等。



可扩展度高、难以管理
P2P一个重要的特点就是具有自扩展性

3、客户机/服务器与P2P的混合

□ Napster: MP3文件共享应用程序。

- P2P: 对等方直接交换MP3文件

- 服务器注册/定位:

- 对等方在中心服务器上注册内容
- 对等方查询相同的中心服务器以定位内容

□ 即时讯息:

- P2P: 两个用户直接聊天

- 服务器检测/定位:

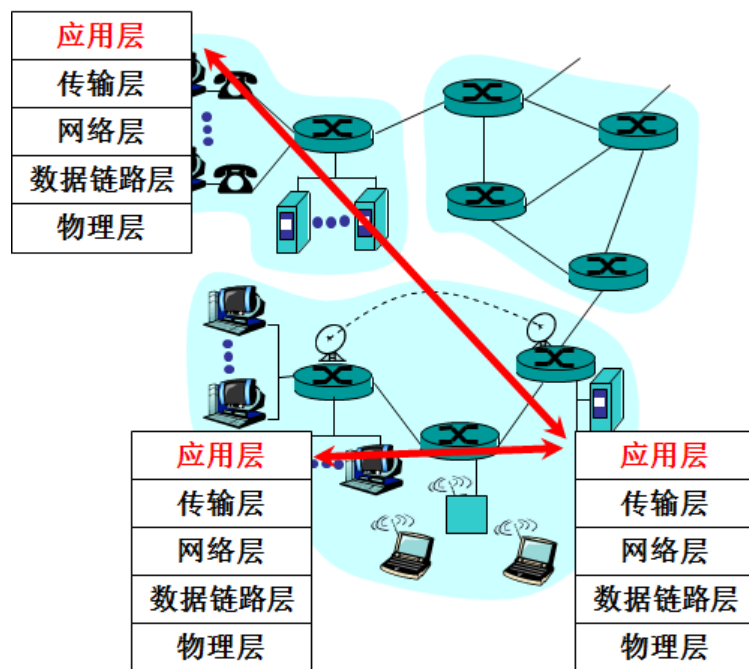
- 用户在线时, 向中心服务器注册其IP地址
- 用户联系中心服务器以找到聊天伙伴的IP地址



2.1.2 进程通信

- 进程 (process)：在主机上运行的程序。
- 进程通信：
 - ✓ 同一主机中两个进程间的通信：由操作系统控制；
 - ✓ 不同主机中进程间的通信：通过网络交换报文进行。
 - ◆ 发送进程：
产生报文并向网络发送；
 - ◆ 接收进程：
接收报文，并回送报文。

如图2-1。



1、客户机和服务器进程

- ❑ 网络应用程序由**成对的进程组成**，并通过网络相互发送报文。[如图2-1](#)
- ❑ 在给定的一对进程的之间的通讯会话场景中，根据功能分别标示为客户机和服务器
- ✓ **客户机进程**：发起通信的进程。
- ✓ **服务器进程**：等待其他进程联系的进程。

如Web应用程序中，一个客户机浏览器进程向某个Web服务器进程发起联系，交换报文。

说明:

P2P结构中的对等的双方，是否区分客户机进程和服务进程？

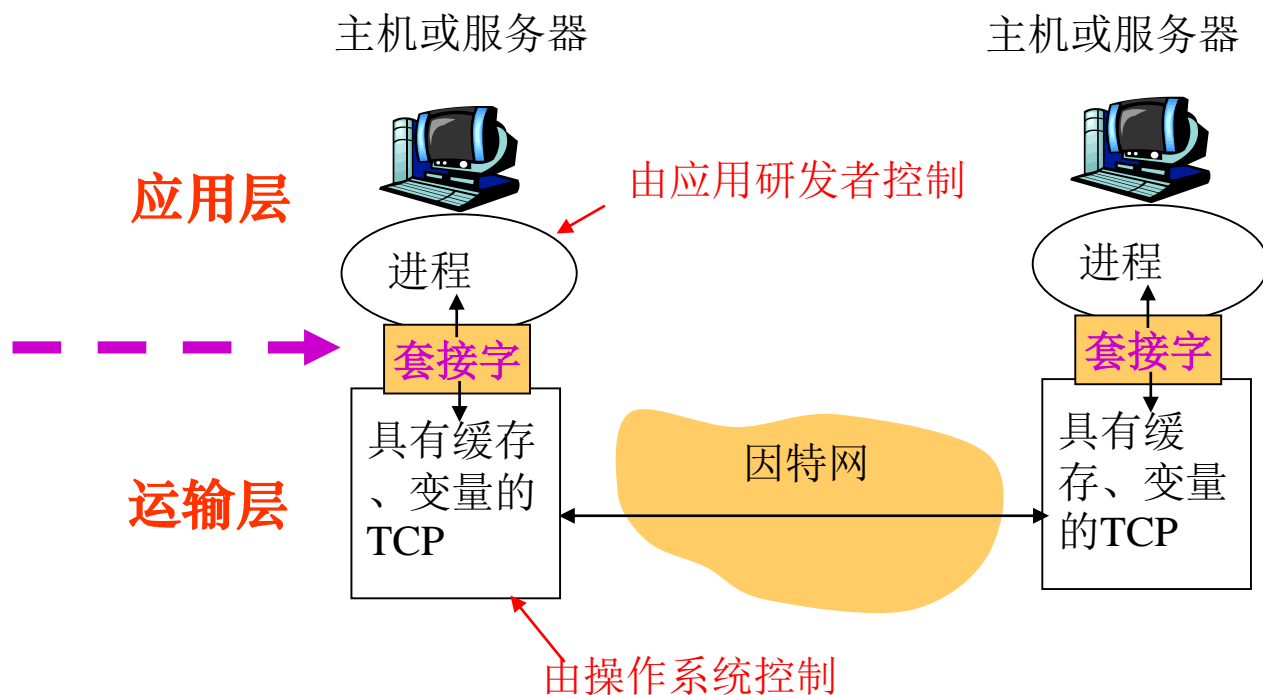
P2P结构的应用程序也可分别看成是客户机进程或服务进程。

如，对等方A（**客户机**）请求对等方B（**服务器**）发送某个文件。

2、进程与计算机网络之间的接口

✓ **套接字**：同一台主机内**应用层与运输层**之间的接口。

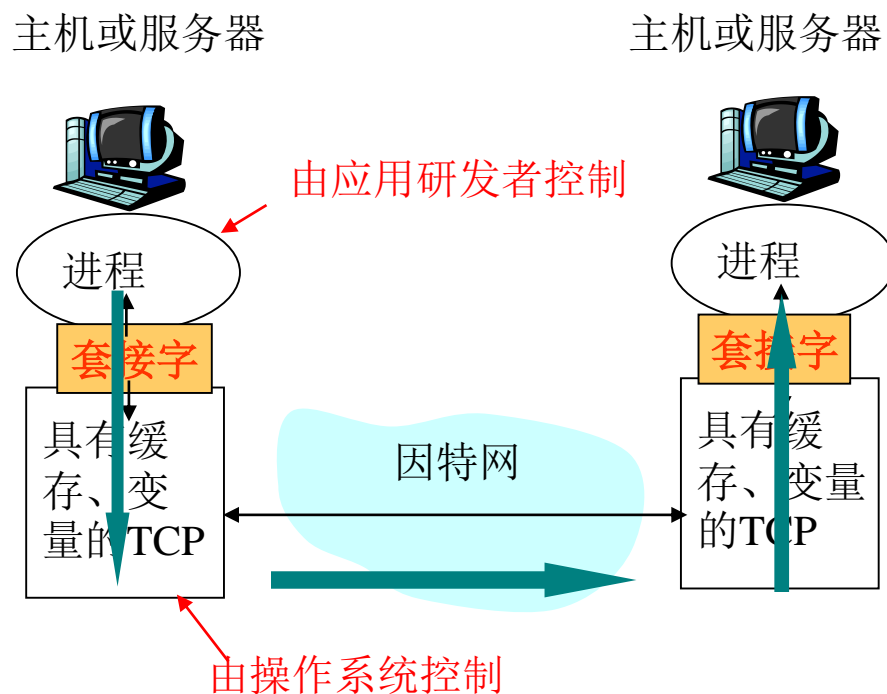
也叫应用程序和网络之间的**应用程序接口API**，是在网络上建立网络应用程序的**可编程接口**。



进程与套接字关系

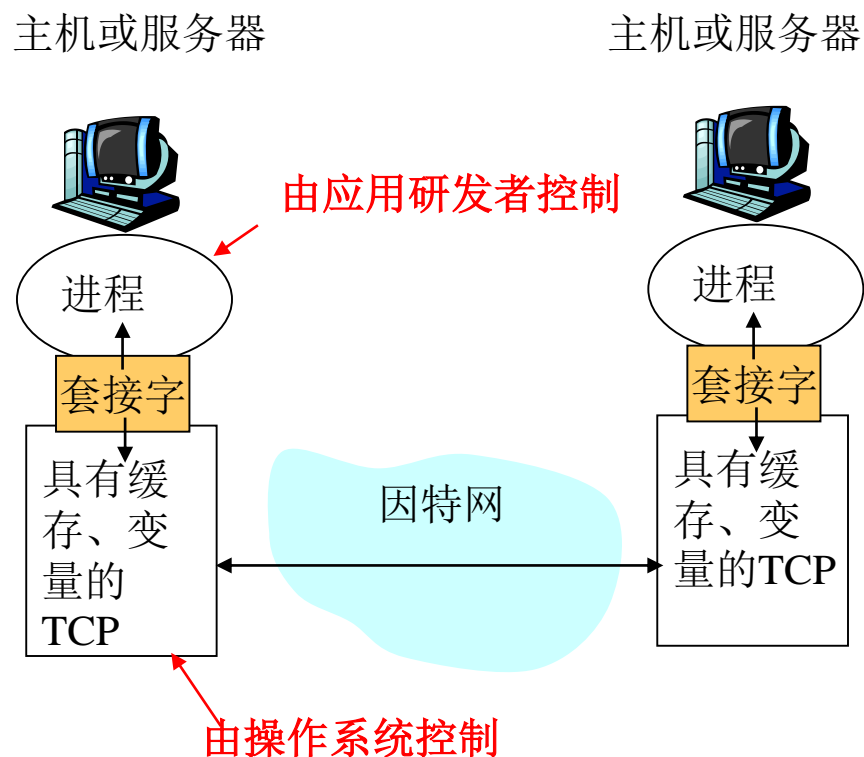
- ✓ 进程类似**房子**，套接字是进程的**门**。
- ✓ 进程通过**套接字**在网络上发送和接收报文。

- ✓ **发送进程**：把报文推出门（套接字）。
- ✓ **传送报文**：通过下面网络把报文传送到目的进程门口。
- ✓ **接收进程**：通过其门（套接字）接收报文



说明

- ✓ 应用程序开发者**可以控制套接字应用层端**的全部；
- ✓ 对套接字的**运输层端几乎不能控制**（只能选择运输层协议、设定几个运输层参数等）。
- ✓ 应用程序开发者选择了一个运输层协议，则应用程序就建立在由该协议提供的运输层服务之上。如TCP协议。



3、进程寻址

如何识别
进程？

- ✓ 主机上的进程可以有多个。
- ✓ 网络中有多个主机，每个主机上有多个进程。

□ 进程识别信息：表示哪台主机上的哪一个进程。

源主机上的进程向目的主机上的进程发送报文时，应带有接收进程的识别信息（标识）。

□ 进程寻址：根据进程识别信息找到相应进程。

确定主机 → 确定进程

进程识别信息（两部分）

- ❑ 主机名称或地址：网络中的哪一个主机。

因特网中，用IP地址标识（32位，全球惟一）。

- ❑ 进程的标识：主机中的哪一个进程。

因特网中，采用端口号标识(port number)。

- ✓ 常用的应用程序被指派固定的端口号（周知端口）。

如，Web服务进程(HTTP协议)：80

邮件服务进程(SMTP协议)：25

- ✓ 创建一个新的网络应用程序时，必须分配一个新的端口号。不重复。

4、用户代理（user agent）

是用户与网络应用之间的接口。

如:

✓ Web应用的用户代理: 是一些浏览器软件。

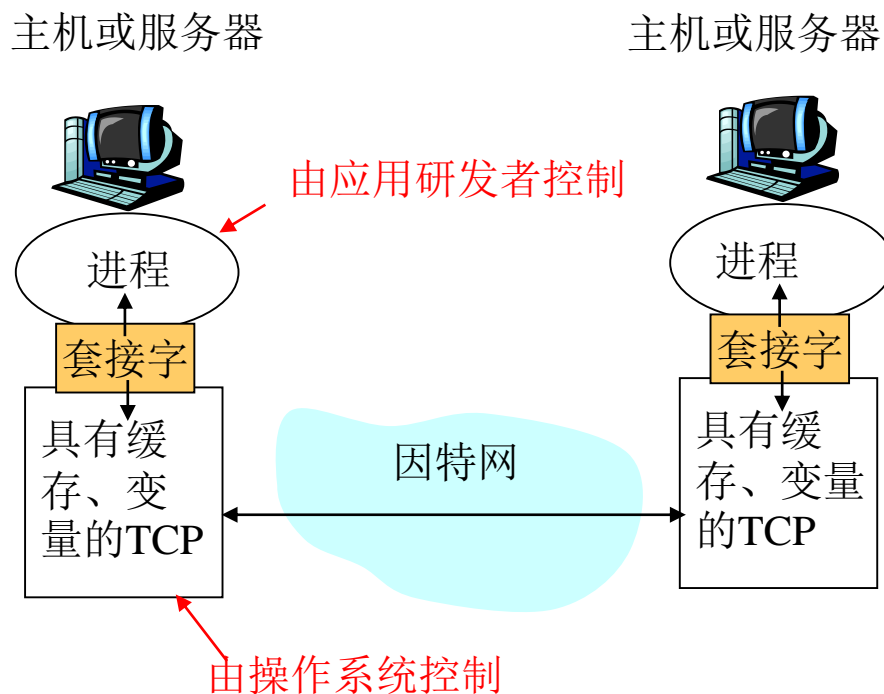
一个通过套接字收发报文, 并提供用户接口的进程。

✓ 电子邮件应用的用户代理: 是“邮件阅读器”。

允许用户进行邮件的撰写和阅读。

2.1.3 可供应用层使用的运输服务

- 应用程序间通信：由**运输协议**跨越网络将发送进程的报文传输到接收进程的门户。
- ✓ 需要使用**运输协议**所提供的服务。
- ✓ 运输协议有多种，提供的服务不同。



应用程序需要什么样的运输服务？

❑ 可靠的数据传输（无数据丢失）

- ✓ 数据不能丢失的应用：如文件传输、金融事务等。
- ✓ 能容忍数据丢失的应用：如多媒体应用。

❑ 吞吐量（可用吞吐量）：带宽（数据传输率）

- ✓ 带宽敏感的应用：需要特定的带宽才能正常工作。

如，因特网电话、其他多媒体应用。

- ✓ 弹性应用：使用的带宽多或少影响不大。

如电子邮件、文件传输以及Web传输。

应用程序需要什么样的运输服务？

□ 定时（数据传输的时间限制）

✓ **交互式实时应用：**对时间敏感，要求时延小。如，因特网电话、视频会议以及多方游戏等。

✓ **非实时应用：**时延无限制，低更好。

□ 安全性：为应用程序提供一种或多种安全性服务。

机密性服务、数据完整性服务和端点鉴别服务

2.1.4 因特网提供的运输服务

两个运输层协议：

- 用户数据报协议UDP

- 传输控制协议TCP

- ✓ 每个协议为调用它们的应用程序提供不同的服务模型。

- ✓ 在创建一个新的因特网应用时，要选择其中一个。

典型应用的运输服务要求

应用程序	数据丢失	带宽	时间敏感
文件传输	不能丢失	弹性	不
电子邮件	不能丢失	弹性	不
Web 文档	不能丢失	弹性	不
实时音频/视频 (因特网电话/视频会议)	容忍丢失	音频: 5kbps-1Mbps 视频: 10kbps-5Mbps	是, 100 ms
存储音频/视频	容忍丢失	同上	是, 几秒
交互式游戏	容忍丢失	几kbps以上	是, 100 ms
即时讯息	不能丢失	弹性	是和不是

1、TCP服务

- ❑ 两个方面:
- ✓ 面向连接的服务:
- ✓ 可靠的传输服务:

面向连接的服务

□ 划分三阶段

✓ 建立连接（握手过程）：

客户机程序和服务器程序之间互相交换控制信息，在两个进程的套接字之间建立一个TCP连接。

✓ 传输报文：

连接是全双工的，即连接双方的进程可以在此连接上同时进行报文收发。

✓ 拆除连接：

应用程序报文发送结束。

可靠的传输服务

- ✓ 通信进程可以**无差错、按适当顺序**交付发送的数据。
- ✓ 没有数据丢失和重复。

拥塞控制

当发送方和接收方之间的网络出现拥塞时，**会抑制发送进程速率。**

对整个网络有益。

安全TCP

TCP和UDP

没有加密

网络明文传输，如用户名和口令信息等

SSL在应用层

应用使用SSL库调用TCP服务接口

SSL提供套接字API

SSL (Security Socket Layer)

提供加密的TCP连接

保证数据完整性

端点认证

2、UDP服务

提供**最小服务模式**运行。

- ✓ **无连接**：两个进程通信前没有握手过程；
- ✓ **不可靠数据传输**：不保证报文能够被接收，或收到的报文是乱序到达。
- ✓ **没有拥塞控制机制**：发送进程可以任何速率发送数据
- ✓ **不提供时延保证**：
- 适于实时应用。

未提供的服务

- ✓ **不确保最小传输速率**：发送进程受拥塞控制机制制约；
- ✓ **不提供时延保证**：数据传输的时间不确定。

TCP协议能保证交付所有的数据，但并不保证这些数据
数据传输的速率以及期待的传输时延。

TCP协议**不适合实时应用**。

因特网应用、应用协议与运输协议

应用	应用层协议	传输协议
电子邮件	SMTP	TCP
远程终端访问	Telnet	TCP
Web	HTTP	TCP
文件传输	FTP	TCP
远程文件服务器	NFS	UDP或TCP
流媒体	HTTP、RTP	UDP或TCP
因特网电话	SIP、RTP	典型用UDP

2.1.5 应用层协议

定义了运行在不同端系统上的应用程序进程间传递报文的格式和方式。

□ 具体内容：

- ✓ 交换的报文类型：如请求报文和响应报文；
- ✓ 各种报文类型的语法：报文中的各个字段及描述；
- ✓ 字段的语义：字段包含信息的含义；
- ✓ 进程何时、如何发送报文及对报文进行响应的规则。

说明

❑ **公共领域协议**：由标准文档RFC定义，如HTTP。

专用层协议：如Skype使用的协议。

❑ **应用层协议是网络应用的一部分。**

如Web应用，客户机从Web服务器获得“文档”。

✓ **组成**：HTML、Web浏览器、Web服务器程序，以及一个应用层协议**HTTP(超文本传输协议)**等。

✓ **HTTP**定义了 in 浏览器程序和Web服务器程序间传输的报文格式和序列。

✓ **其他协议**：电子邮件协议SMTP等等

2.2 Web应用和HTTP协议

产生于20世纪90年代初期。

- ✓ 改变了人们与工作环境内外的交流方式；
- ✓ 提升因特网地位；
- ✓ 生活和工作发生变化；
- ✓ 方便、快捷得到所需要的信息（**按需操作**）；
- ✓ 任何人在Web上发布信息；
- ✓ 超链接和搜索引擎帮助人们浏览Web站点。

本节内容

2.2.1 HTTP概况

2.2.2 非持续连接和持续连接

2.2.3 HTTP报文格式

2.2.4 用户与服务器交互：Cookie

2.2.5 Web缓存

2.2.7 条件GET方法

2.2.1 HTTP概况

□ HTTP（超文本传输协议hypertext transfer protocol）：

应用层协议，Web的核心。

□ 包括两部分：

✓ 客户机程序：浏览器browser请求, 接收Web对象

✓ 服务器程序：Web服务器响应请求, 发送 Web对象

分别运行在不同的端系统中，通过交换HTTP报文进行会话。

□ HTTP协议定义了报文的格式以及客户机和服务器交换报文的格式和方式。

HTTP 1.0: RFC 1945 HTTP 1.1: RFC 2616



Web常用术语

□ Web页(文档): 由若干对象组成。

对象: 是文件。通过一个URL地址来寻址。如HTML文件、JPEG图形文件、Java小程序等。

✓ Web页含有一个基本的HTML文件及几个引用对象。

例如, 一个Web页包含HTML文本和5个JPEG图形文件 (即有6个对象) 。

✓ 在基本的HTML文件中, 每个对象可由URL来寻址。

超文本标记语言

统一资源定位符



Web常用术语

✓ **URL**: Uniform Resource Locator。统一资源定位符。标识万维网WWW上的各种文档，全网范围唯一。

✓ **URL地址组成**:

存放对象的服务器主机名和对象的路径名。

如 `http:// www.someSchool.edu / somedepartment / picture.gif`

↑
协议 主机名 路径名

URL 的一般形式是:

<协议>://<主机>[:<端口>]/<路径>

ftp —— 文件传送协议 FTP

http —— 超文本传送协议 HTTP

News —— USENET 新闻



Web常用术语

- ❑ 浏览器（客户机）：是Web应用的**用户代理**，用于显示所请求的Web页，提供导航功能和配置属性。

实现了HTTP协议的客户端。

- ❑ Web服务器：用于存贮Web对象（由URL寻址）。

实现HTTP协议的服务器端。

HTTP协议定义了Web客户机 (浏览器)如何向Web站点请求Web页，以及服务器如何将Web页传送给客户机。



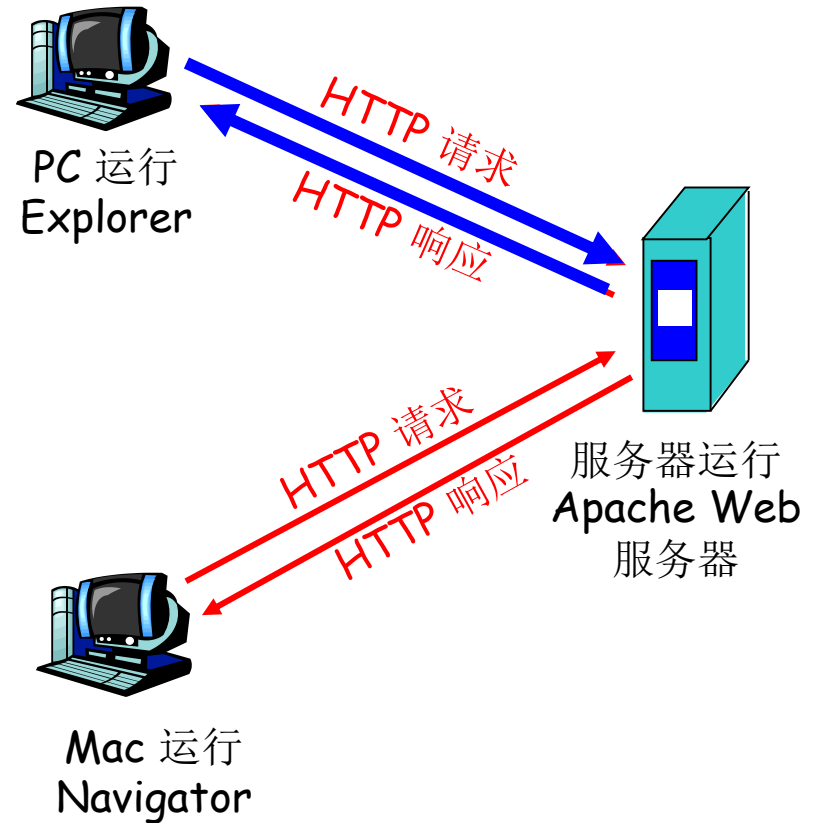
客户机和服务器之间交互过程

□ 客户机：

用户请求一个Web页
(如点击一个超链接)，浏览器向服务器发出对该页所含对象的“HTTP请求报文”。

□ 服务器：

接受请求，回发包含请求对象的“HTTP响应报文”



说明

□ HTTP协议使用的底层运输协议是TCP。

□ 工作过程：

创建TCP连接→交换报文→关闭TCP连接

- ✓ 客户机先与服务器建立TCP连接，然后，浏览器和服务进程通过套接字访问TCP。
- ✓ 客户机：从其套接字接口发送“HTTP请求报文”和接收“响应报文”；
- ✓ 服务器：从其套接字口接收“HTTP请求报文”和发送“响应报文”。



说明

- ❑ TCP提供可靠的数据传输服务：客户机进程和服务进程发出的每个HTTP报文能完整地到达对方。
- ❑ HTTP是无状态协议：服务器不保存关于客户机的任何信息。

服务器如果为每个客户机存储状态信息，千万级的信息存储，会严重限制应用的扩展性

Web使用客户机/服务器结构，Web服务器总是打开，有一个固定IP地址，为多个浏览器服务。



2.2.2 非持续(久)连接和持续(久)连接

非持续HTTP连接

- ✓ 每个TCP连接上只传送一个Web对象
- ✓ 只传送一个请求/响应对
- ✓ HTTP/1.0使用非持续HTTP连接

持续HTTP连接

- ✓ 一个TCP连接上可以传送多个Web对象
- ✓ 传送多个请求/响应对
- ✓ HTTP/1.1默认使用持续HTTP连接

1、非持续连接

例，客户机向服务器请求传送一个Web页：

- ✓ 含有一个基本HTML文件和10个JPEG图形，11个对象位于同一个服务器上。
- ✓ HTML文件的URL为：

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



工作过程

TCP 连接

1a. HTTP客户初始化一个与服务
器主机www.someSchool.edu中
HTTP服务器进程的TCP连接

1b. www.someSchool.edu服务器
主机中的HTTP服务器在80端
口监听来自HTTP客户的TCP
连接请求，收到连接请求，接
受，建立连接，通知客户。

2. HTTP客户发送一个HTTP请
求报文（包含URL）到TCP
连接套接字，报文指明客户
需要的Web对象—
someDepartment/home.index

3. HTTP服务器接收请求报文，
产生一个响应报文（包含被
请求对象），并发送到其TCP
连接套接字

报文传输

time

4. HTTP服务器关闭TCP 连接

5. HTTP客户机接收包含HTML
文件的响应报文，显示并解
析HTML文件，发现10个引
用的 jpeg对象

6. 对10个jpeg对象重复步骤1~5

time



说明：

- ❑ 每个TCP连接在服务器返回对象后关闭（非持久）。
- ❑ 每个TCP连接只传输一个请求报文和一个响应报文；

上例中，要建立11个TCP连接。

- ❑ 浏览器可同时打开多个连接：
 - ✓ 并行的TCP连接：并行数大于1。默认打开5~10个。
 - ✓ 串行的TCP连接：最大并行数为1。



请求一个HTML文件所需时间

即从客户机请求连接开始，到用户收到整个文件为止所花时间。

□ 往返时延RTT:

一个小分组从客户机到服务器，再回到客户机所花时间。

包括传播时延、排队时延以及处理时延。



请求一个HTML文件所需时间

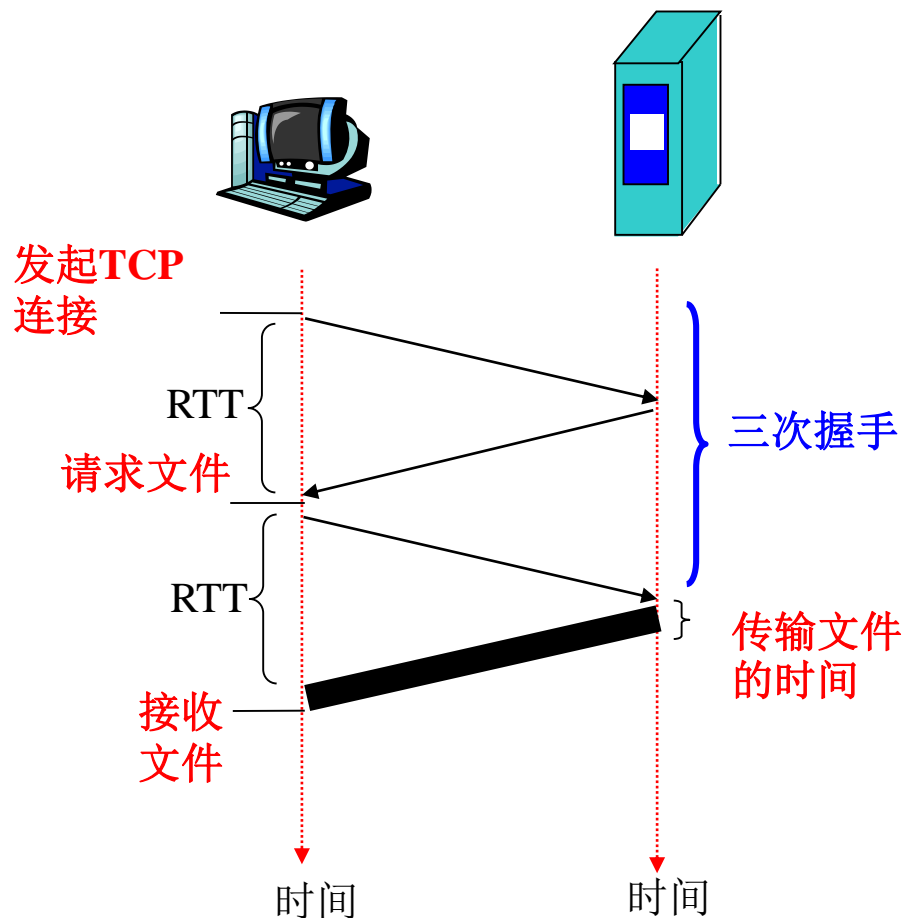
建立TCP连接 → 交换报文

□ TCP连接的“三次握手”过程

- ✓ 客户机发送一个TCP连接请求报文段
- ✓ 服务器回送一个TCP确认响应报文段
- ✓ 客户机向服务器发送一个包含“HTTP请求”与“TCP确认”的报文

- ## □ 总响应时间：两个RTT时延加上服务器发送文件的时间

总计 = $2RTT + \text{文件传输时间}$



非持续连接缺点

- ❑ 服务器负担重：每一个请求对象建立和维护一个新的连接。
- ❑ 每一个对象的传输时延长：包含两个RTT时延，一个用于TCP建立，一个用于请求和接收对象。



2、持续连接

- ❑ 服务器在发送响应后保持该TCP连接：
- ✓ 相同客户机与服务器之间的后续请求和响应报文通过相同的连接进行传送。

如，一个Web页的所有对象可以通过一个持续TCP连接传送。

或同一服务器上的多个Web页也可以通过一个持续TCP连接传送给同一个客户机。

- ❑ 连接经一定时间间隔(超时间隔)未被使用，服务器就关闭该连接。



持续连接两种方式

- 非流水线方式：客户机只能在前一个响应接收到之后才能发出新的请求。
- ✓ 客户机为每一个引用对象的请求和接收都使用一个RTT时延。
- ✓ 会浪费一些服务器资源：服务器在发送完一个对象，等待下一个请求时，会出现空闲状态。



持续连接两种方式

□ 流水线方式：

- ✓ 客户机可一个接一个**连续产生请求**（只要有引用就产生），即在前一个请求接收到响应之前可以产生新的请求。
- ✓ 服务器一个接一个**连续发送相应对象**。

□ 特点：

- ✓ 节省**RTT时延**，可能所有引用对象**只花费一个RTT**。
- ✓ TCP连接空闲时间很短。

□ 默认方式：流水线方式的持久连接。



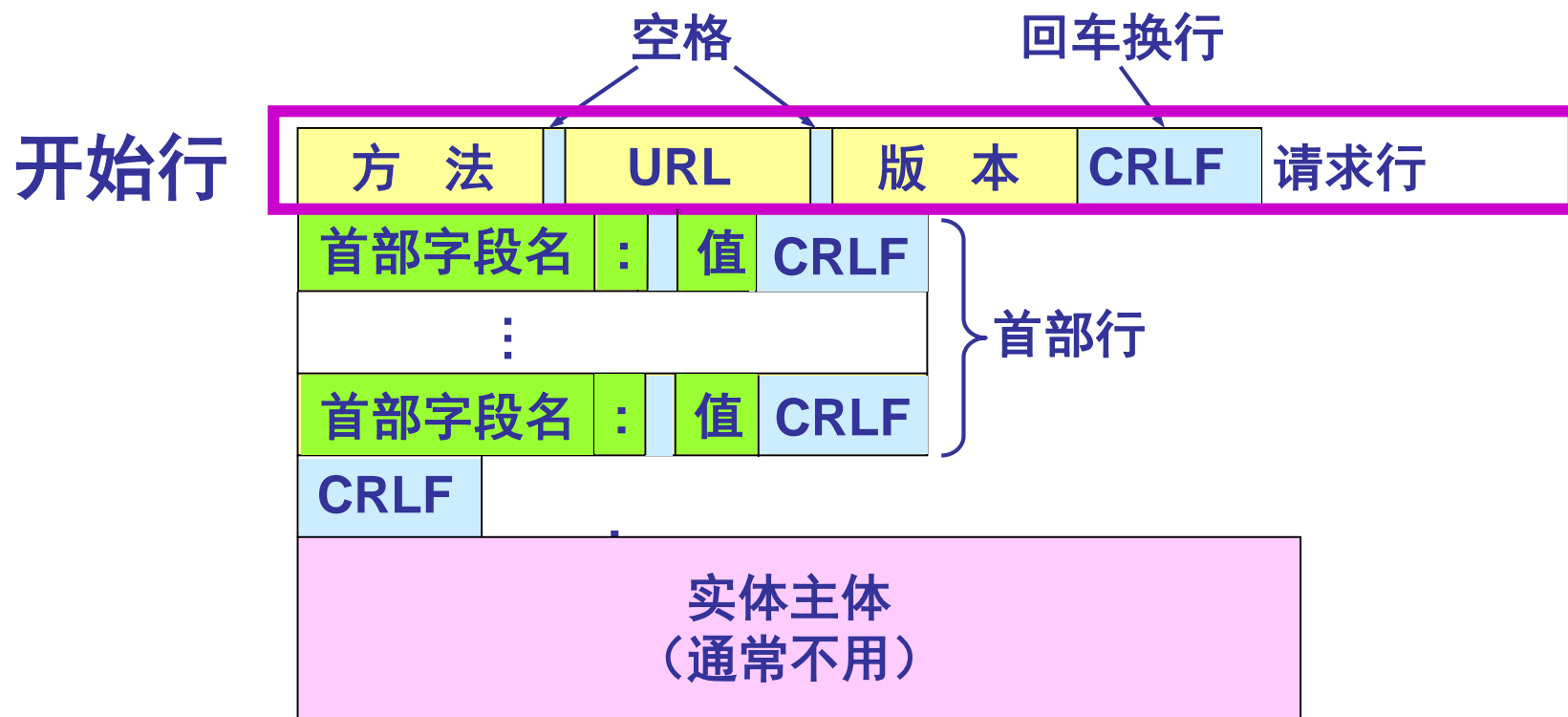
2.2.3 HTTP报文格式

请求报文格式

应答报文格式

1、HTTP请求报文

HTTP请求报文通用格式



报文由三个部分组成，即开始行、首部行和实体主体。
在请求报文中，开始行就是请求行。



1. HTTP请求报文

方法（命令）——

- ✓ GET：请求一个对象。
- ✓ POST：提交表单（添加信息）。
- ✓ HEAD：请求返回对象响应报文首部

文本形式，易读。

对象URL路径名

版本

请求行
(GET, POST,
HEAD命令)

首部行

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

对象主机

浏览器类型

非持久

返回对象的语言

回车，换行指示
报文的结束

(另外的回车，换行)



方法类型

HTTP/1.0

□ GET

□ POST

□ HEAD

- 服务器收到请求时，用HTTP报文进行响应，但不返回请求对象

HTTP/1.1

□ GET, POST, HEAD

□ PUT

- 文件在实体主体中被上传到URL字段指定的路径

□ DELETE

- 删除URL字段指定的文件

上载表单(各字段)输入值

Post方法:

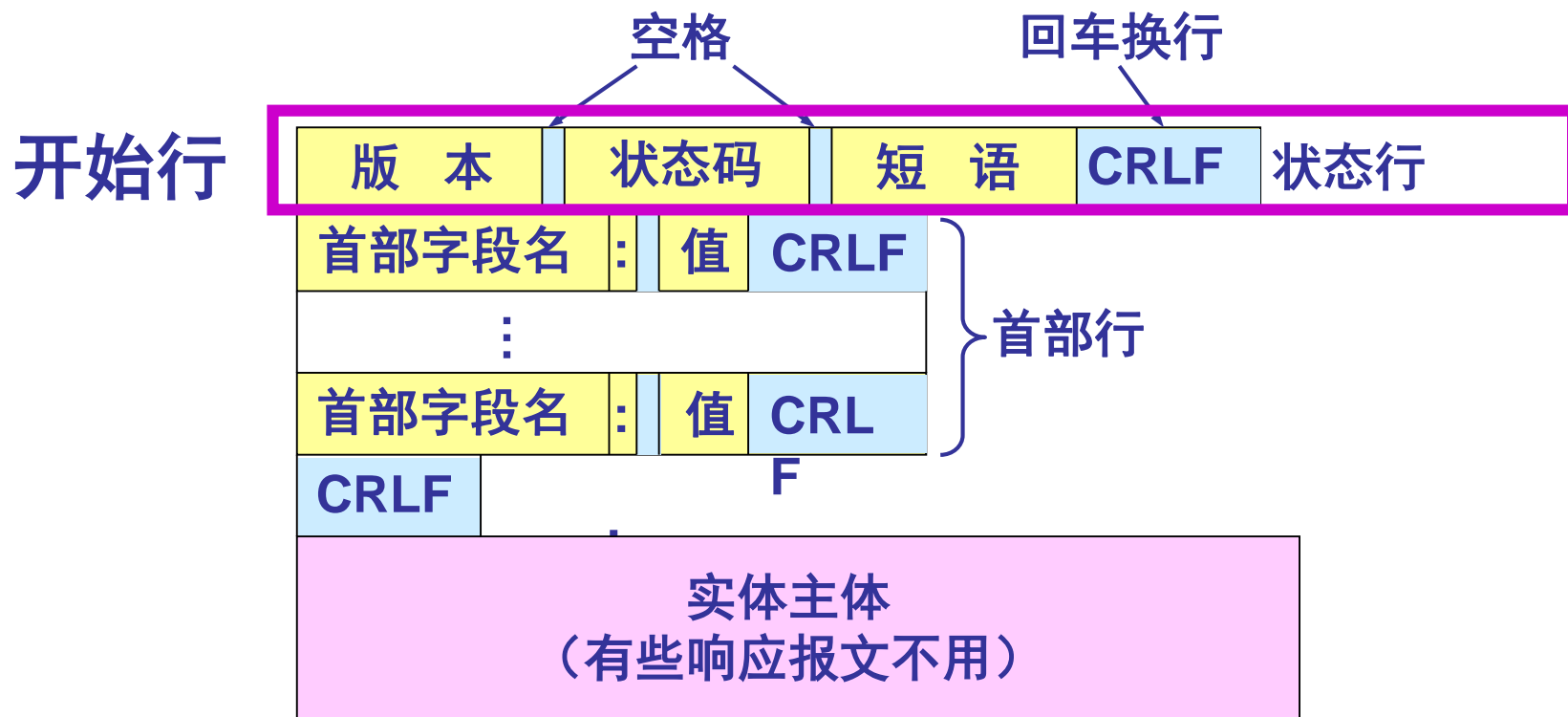
- ❑ 网页时常包含表单输入
- ❑ 输入值在请求报文的实体主体中被上载到服务器

URL方法:

- ❑ 使用GET方法
- ❑ 表单(各字段)输入值被上载,以URL请求行的字段:
`www.somesite.com/animalsearch?monkeys&banana`

2、HTTP 响应报文

HTTP响应报文通用格式



响应报文的开始行是**状态行**。
状态行包括三项内容，即 **HTTP** 的版本，**状态码**，
以及解释状态码的**简单短语**。

2、HTTP 响应报文

服务器到客户机的回答

状态行
(版本、状态码、状态短语)

请求成功

发送日期

关闭连接

首部行

对象创建或修
改日期

对象长度

```
HTTP/1.1 200 OK
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
```

```
data data data data data ...
```

实体：数据，如请
求的HTML文件



HTTP响应状态码

□ 在服务器到客户机响应报文中的首行。

200 OK

- 请求成功，请求的对象在这个报文后面

301 Moved Permanently

- 请求的对象已转移，新的URL在响应报文的Location:首部行中指定

400 Bad Request

- 请求报文不为服务器理解

404 Not Found

- 请求的文档没有在该服务器上发现

505 HTTP Version Not Supported

- 服务器不支持请求报文使用的HTTP版本

自行试验HTTP (客户机侧)

1. Telnet 到某个Web服务器上:

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

打开到位于**cis.poly.edu**的端口**80**(默认的**HTTP**服务器端口). 键入的任何东西将发送到位于**cis.poly.edu**的**80**端口

2. 键入一个GET HTTP请求:

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

向**HTTP**服务器发送最小的**GET**请求

3. 得到由HTTP服务器发送的响应报文!

```
Telnet www.is.uestc.edu.cn
```

```
HEAD / HTTP/1.1
```

```
HOST: WWW. IS. UESTC. EDU. CN
```

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
```

```
Set-Cookie: JSESSIONID=0A07073B9F26B3B78E5C6CD7DA34F512; Path=/; HttpOnly
```

```
Content-Type: text/html; charset=utf-8
```

```
Transfer-Encoding: chunked
```

```
Date: Sun, 22 Jan 2017 08:53:21 GMT
```

2.2.4 用户与服务器交互：Cookie

HTTP服务器是**无状态**的，不保存客户信息。

- **Cookie**: 允许Web站点跟踪、识别用户；服务器可以限制用户访问，或把内容与用户身份关联。

许多重要的Web站点使用cookies。

- **包括四个部分**

- 1)在HTTP响应报文中有一个cookie 首部行
- 2)在HTTP请求报文中有一个cookie 首部行
- 3)用户主机中保留有一个 cookie 文件并由浏览器管理
- 4) Web站点的后端数据库保存cookie

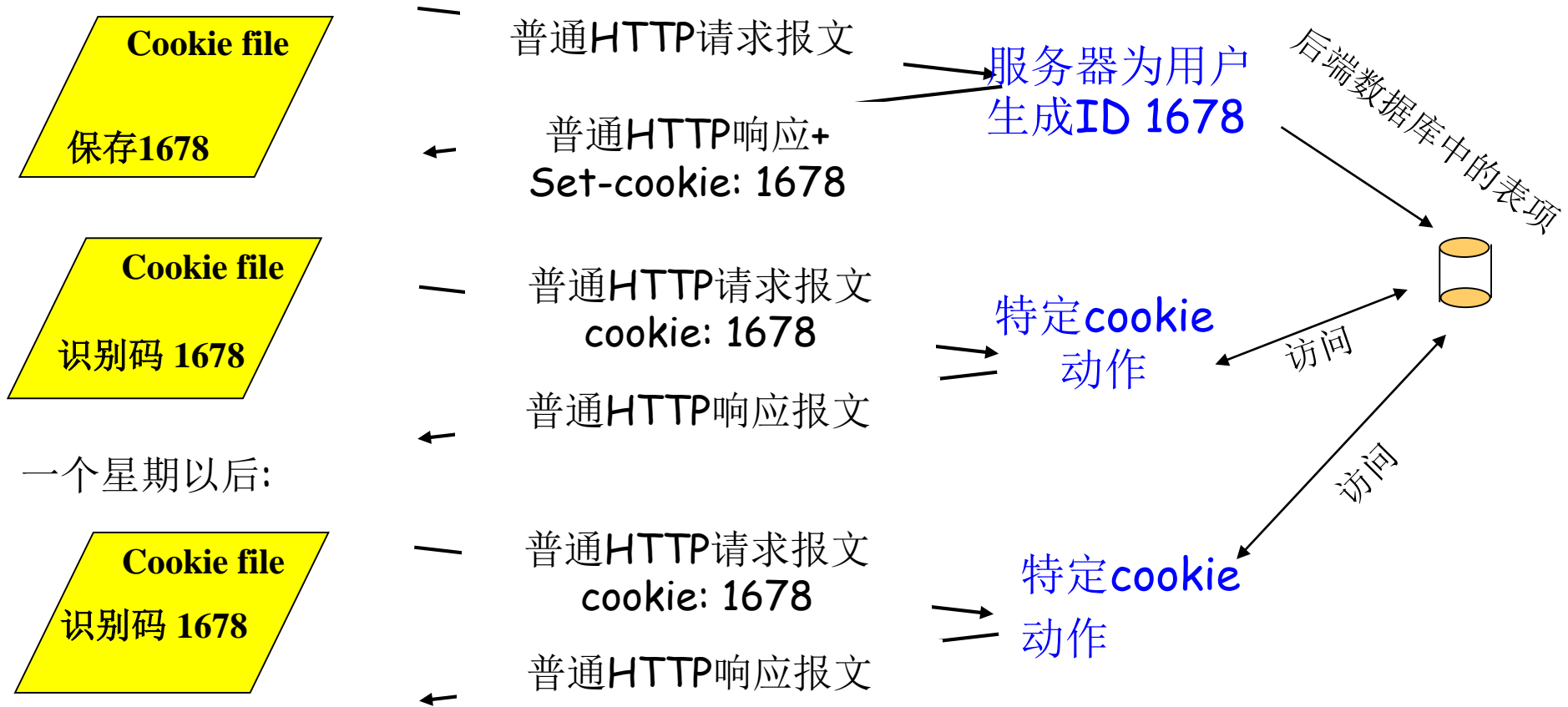
例

- Susan总是从相同的PC访问因特网
- 她首次访问一个特定的电子商务站点
- 当起始HTTP请求到达站点时，站点产生一个独特的ID，并以此ID为索引，在后端数据库中生成一个表项。

工作过程

客户机

服务器



Cookie用途和缺陷

□ Cookie用途

- ✓ 身份认证
- ✓ 虚拟购物车(跟踪用户购买的物品)
- ✓ 推荐广告
- ✓ 用户会话状态 (Web e-mail)

□ Cookie缺陷

- ✓ 站点可以知道用户许多信息
- ✓ 不利用户隐私保护

2.2.5 Web缓存

□ Web缓存器(Web cache): 也叫代理服务器。

能够代表起始服务器来满足HTTP请求的网络实体。

✓ 保存最近请求过的对象的副本。

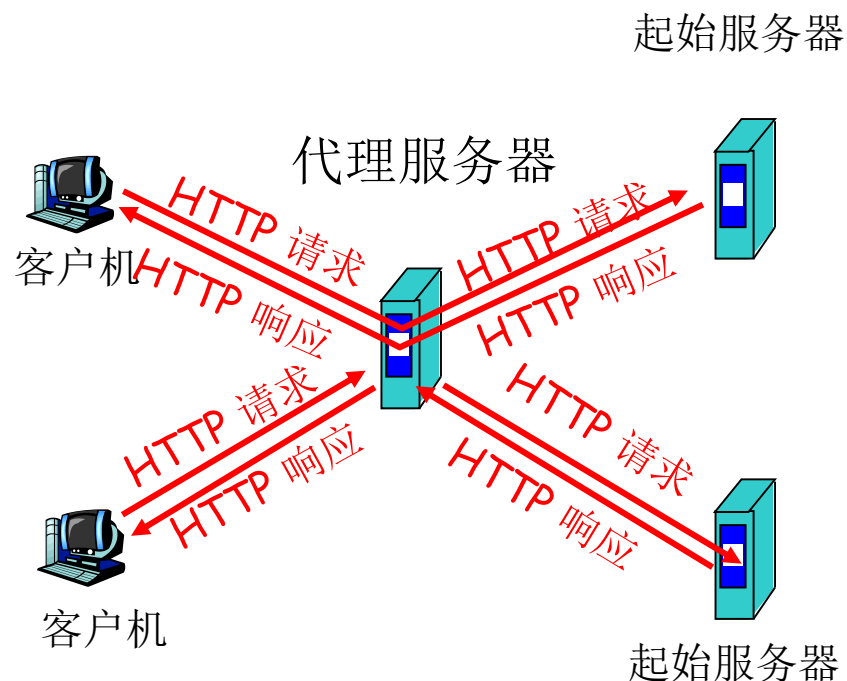
□ 起始（原始）服务器(origin server): 对象最初存放并始终保持其拷贝的服务器。

目标: 代替原始服务器满足HTTP请求。

使用Web缓存器

客户机通过Web缓存器请求对象。

- 用户配置浏览器: 所有Web 访问经由缓存
- 浏览器向缓存发送所有HTTP请求
 - 对象在缓存中: 缓存器返回对象
 - 不在: 缓存向原始服务器发出请求, 接收对象后转发给客户机

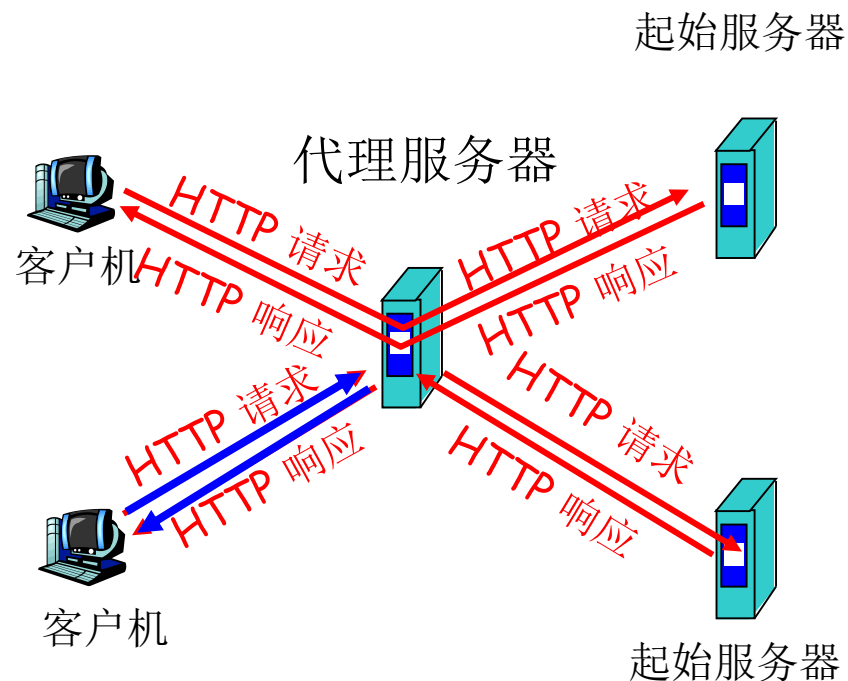


具体操作过程

例：假设浏览器请求对象

`http://www.someschool.edu/campus.gif`

- **浏览器**：建立一个到web缓存的TCP连接，并向缓存发送一个对该对象HTTP请求
- **Web缓存器**：检查本地是否有该对象的拷贝。
- ✓ **有**：就用HTTP响应报文向浏览器转发该对象

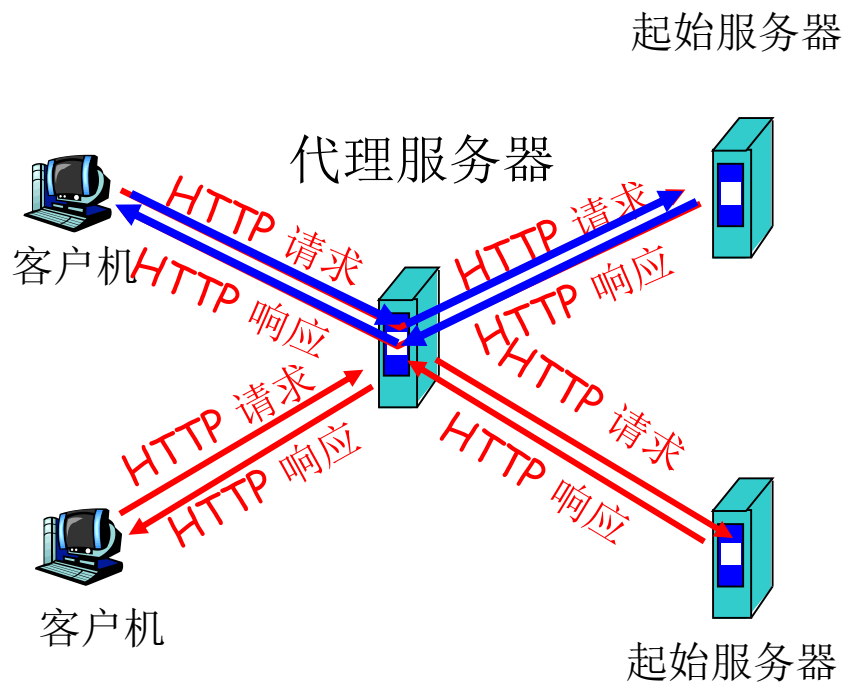


具体操作过程

✓ **没有：** 与该对象的起始服务器打开一个TCP连接。

客户机 \longleftrightarrow Web缓存器 \longleftrightarrow 起始服务器

- ◆ 缓存在TCP连接上发送获取该对象的请求。
- ◆ 起始服务器收到请求，向缓存发送该对象的HTTP响应
- ◆ 缓存接收该对象，存储一份在本地中，并通过HTTP响应报文向浏览器转发该对象（通过已经建立的TCP连接）。



说明

- Web缓存器既可以是服务器也可以是客户机：
 - ✓ 当它接收浏览器请求并发回响应时，是服务器；
 - ✓ 当它向起始服务器发出请求并接收响应时，是客户机

Web缓存优点

□ 减少客户机请求的响应时间:

客户机 \longleftrightarrow Web缓存器 \longleftrightarrow 起始服务器

高速链路

□ 减少机构内部网络与因特网连接链路上的通信量:

降低开销, 改善各种应用的性能。

使得英特网的总体流量减少, 改善了英特网的整体性能。

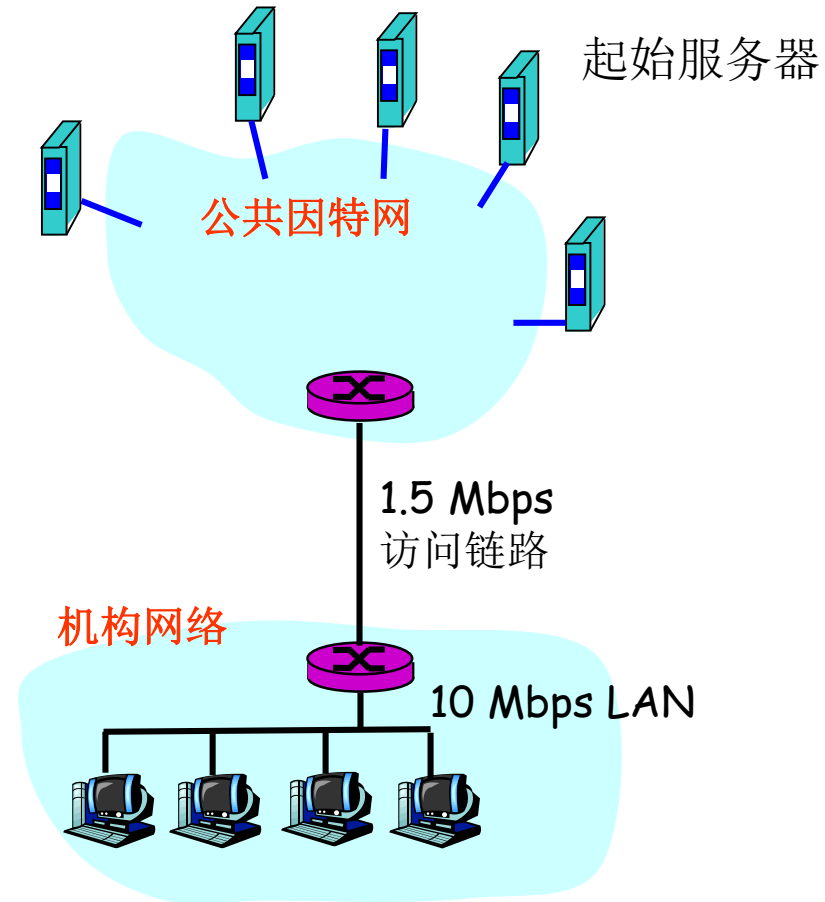
例1，无Web缓存

□ 包括两个网络：**机构的内部网络和因特网。**

✓ **机构内部网络：**是一个高速的局域网。

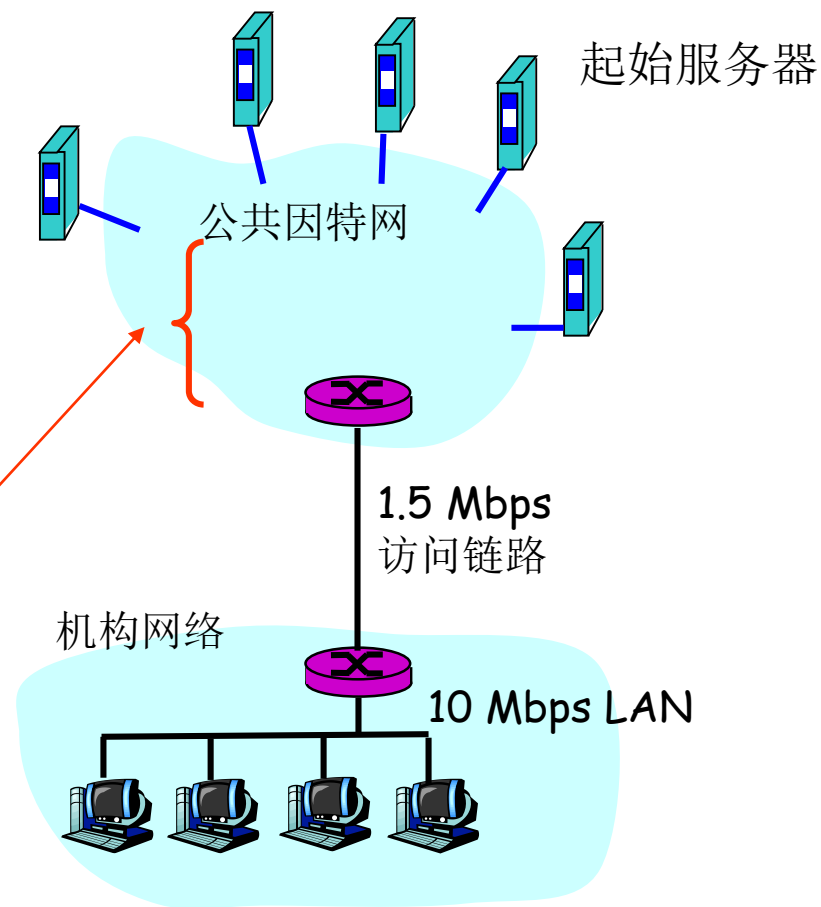
其路由器与因特网上的路由器通过一条1.5Mbps的链路连接。

✓ **起始服务器：**与因特网相连，遍布全球。



假设

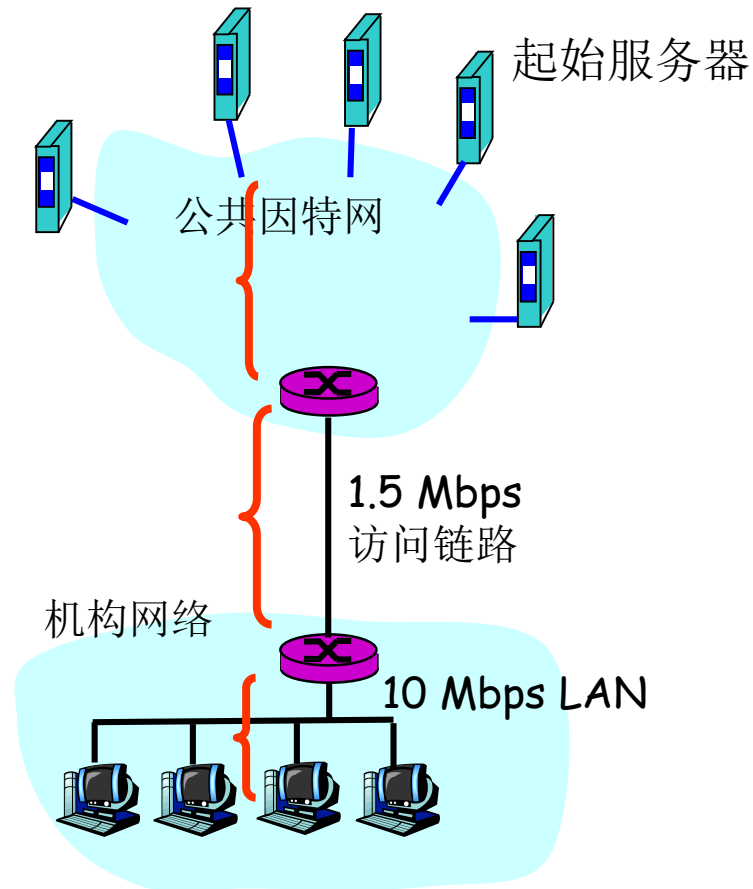
- ✓ 平均对象长度为 **100kb**
- ✓ 机构内浏览器对原始服务器上对象的平均请求率 = **15个请求/sec**
- ✓ **机构内**的HTTP报文小，**忽略**
- ✓ 从因特网路由器转发HTTP请求报文，到收到其响应报文的时间平均 = **2s (因特网时延)**



总的响应时间

浏览器从请求一个对象到接收到的时间：三部分和

- ✓ 局域网时延
- ✓ 接入链路时延(两个路由器间)
- ✓ 因特网时延



时延：与流量强度有关（**比特到达率/推出率**）

✓ **局域网时延**：

$$(15 \text{ 请求/s}) \times (100 \text{ kb/请求}) / (10 \text{ Mbit/s}) = 0.15$$

强度为0.15的通信量最多数十毫秒的时延，可**忽略**

✓ **接入链路时延**：接入**流量强度**（路由器之间）有关

$$(15 \text{ 请求/s}) \times (100 \text{ kb/请求}) / (1.5 \text{ Mbit/s}) = 1$$

强度接近1，链路**时延非常大**或无限增长。

✓ **总响应时间** = 接入链路时延 + 因特网时延

$$= (\text{分钟} + 2 \text{ sec})$$

请求时间长，用户难接受。

改进方法一

- ✓ 增加接入链路的速率：

如从1.5Mbps增加到10Mbps。

使链路上的流量强度减少到0.15，链路时延也可以忽略了。

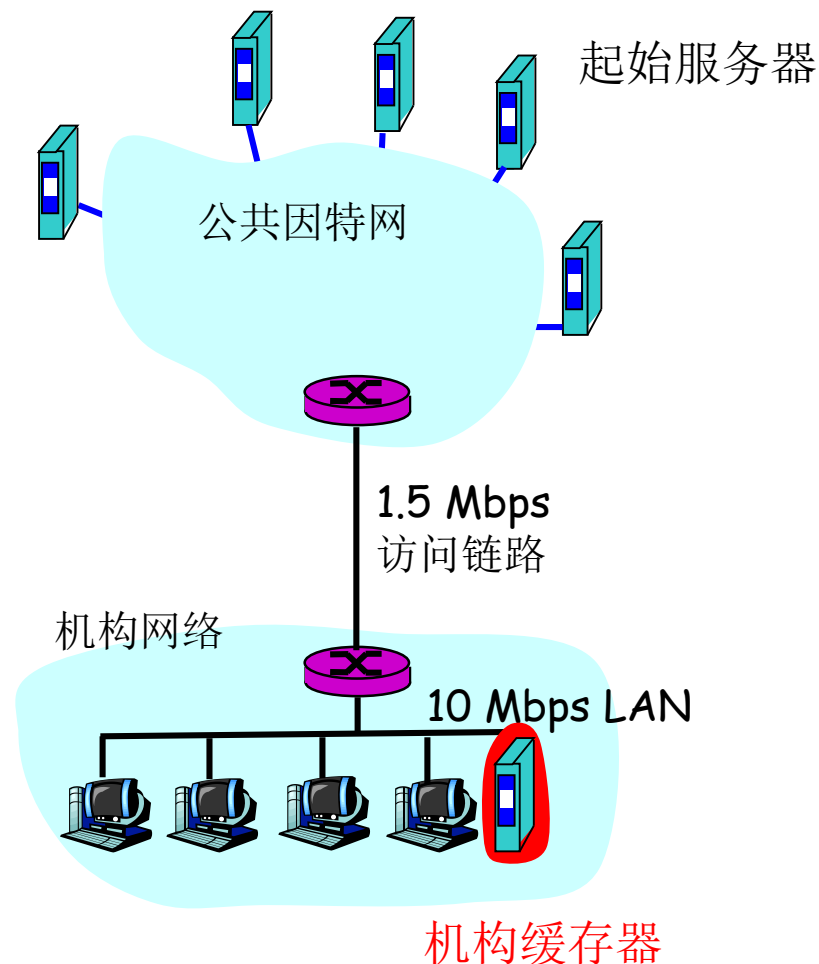
总响应时间=因特网时延=2秒钟

- ✓ 投资较大，成本昂贵。

改进方法二

在机构网络中安装一个Web缓存器。

- ✓ Web缓存器的命中率：缓存器满足请求的比率（0.2~0.7）。
- ✓ 设命中率为0.4。

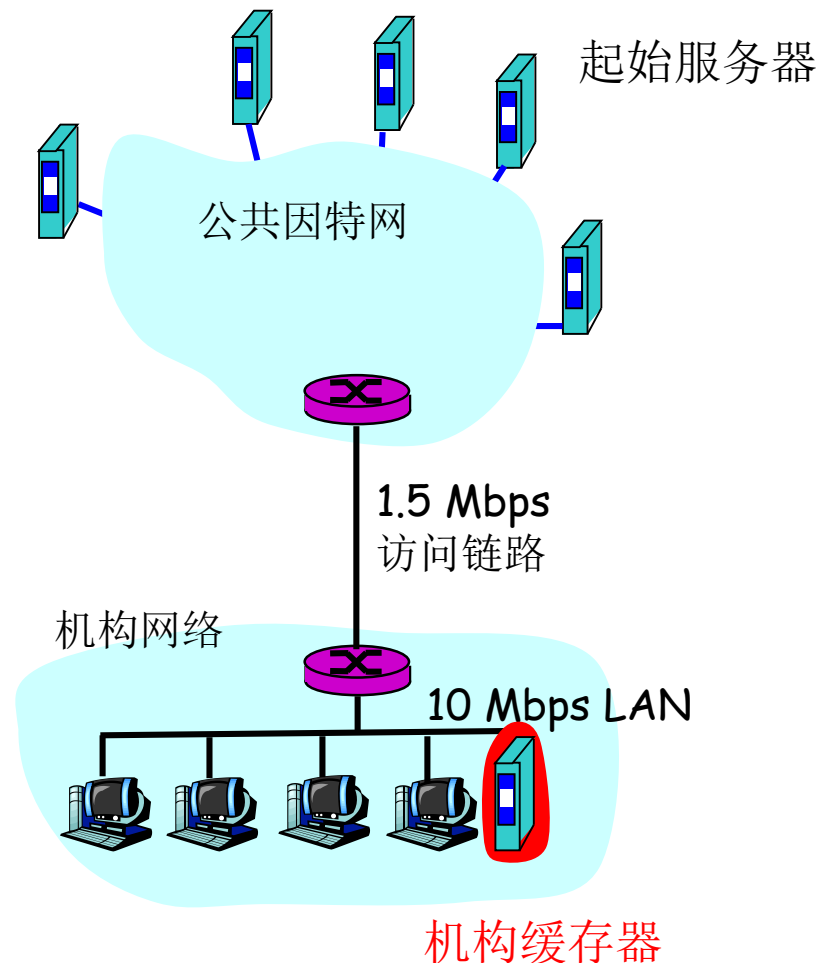


改进方法二

100kb/10Mbps

- ✓ 局域网时延：客户机缓存
寄存器位于同一局域网
40%的请求几乎会立即得
到响应，时延约10ms。

剩下的60%请求需要
通过访问起始服务器才能
满足。



改进方法二

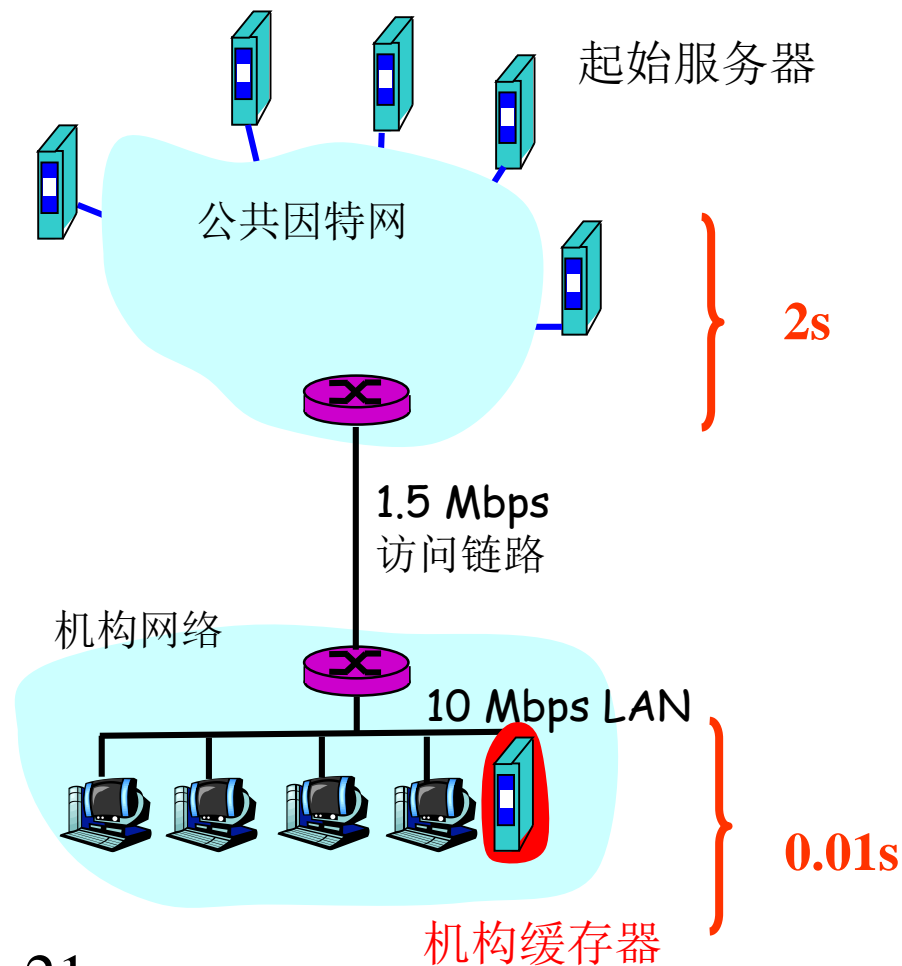
✓ 接入链路时延:

只有60%的请求对象通过接入链路传送, 流量强度从1.0减小到0.6。

通常, 在1.5Mbps链路上, 当流量强度小于0.8时, 时延很小, 可忽略。

✓ 平均时延为:

$$0.4 \times (0.01s) + 0.6 \times (0.01s + 2s) = 1.21s$$



Web缓存器减少响应时延, 降低成本

2.2.6 条件GET方法

□ 高速缓存：

- ✓ 减少响应时间；
- ✓ 存放在缓存中的对象拷贝可能是旧的。即保存在起始Web服务器中的对象可能已经被修改。

□ 条件GET方法：

- ✓ 使缓存器能够证实其保存的对象是否为最新。
- ✓ 如果缓存中是最新对象版本，可继续使用，起始Web服务器就不需重新发送该对象。
- ✓ 否则Web服务器就重新发送该对象。

条件GET方法使用

- ❑ Web服务器回发响应报文：包括对象的最后修改时间
Last-modified: date1
- ❑ 缓存检查Web服务器中的该对象是否已被修改，发送一个条件GET请求报文：
If-modified-since: date1
- ✓ 告诉服务器，仅当自指定日期之后该对象被修改过，才发送该对象。
- ✓ 若Web服务器中的该对象未被修改，则响应报文含有304 Not Modified，并且实体为空。

例

缓存器

缓存器没有，它将向初始服务器发送报文请求该对象

GET /fruit/kiwi.gif HTTP/1.1
Host:www.exotiquecuisine.com

服务器

缓存器将对象转发到浏览器，并保存对象到本地（包括对象的最后修改时间）。

HTTP/1.1 200 OK
Last-modified: date1
<data>

一周后，用户再次请求该对象（仍保留在缓存中），缓存发送一个条件GET，检查该对象是否已被修改

GET /fruit/kiwi.gif HTTP/1.1
Host:www.exotiquecuisine.com
If-modified-since: date1

对象未修改

HTTP/1.1 304 Not Modified
实体为空

对象未修改，缓存可以继续使用对象的拷贝，并转发给用户浏览器。

本次课小结

本次课介绍了第一个互联网应用层协议**HTTP**

(1) **web**常用术语：服务器是无状态的

(2) 非持续连接和持续连接

(3) **HTTP**报文格式：请求和应答

(4) **Cookie**

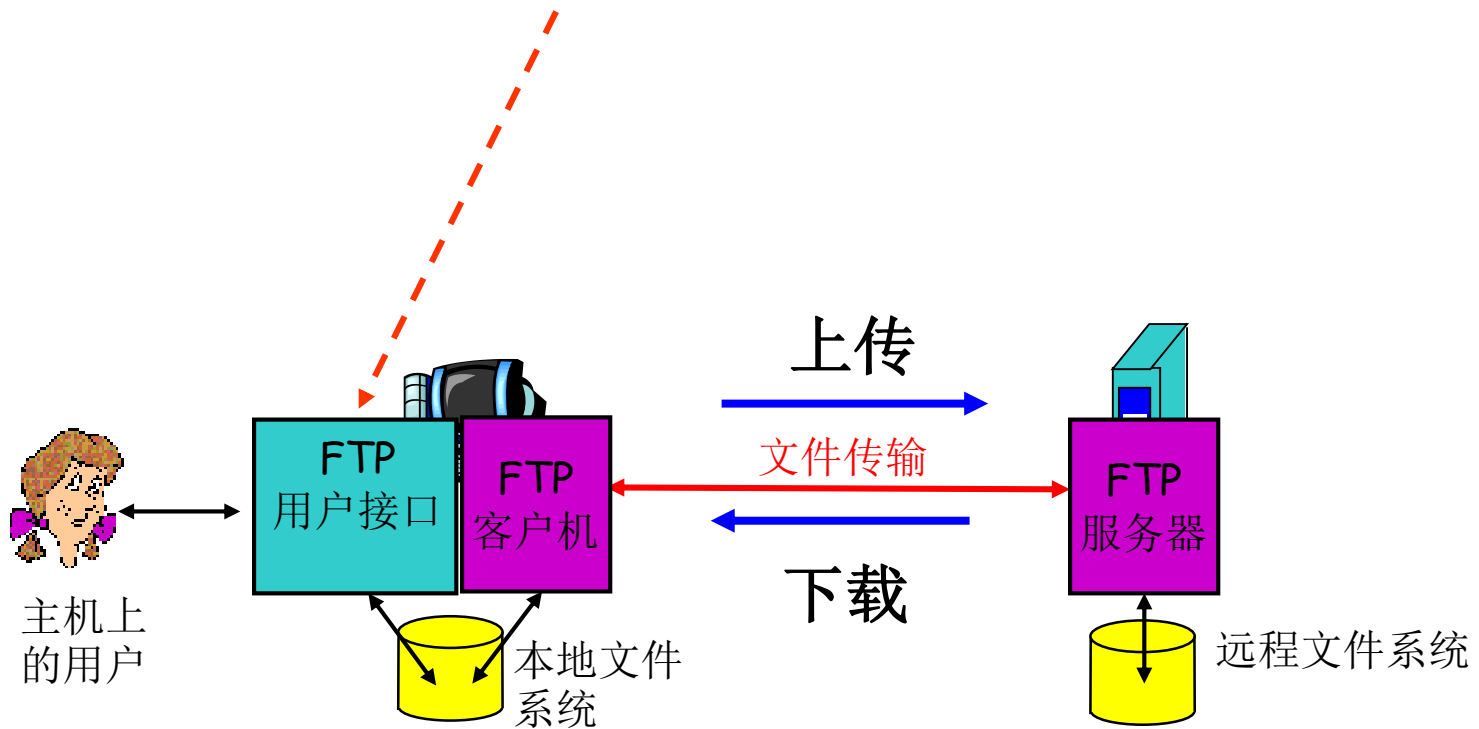
(5) **web**缓存技术

(6) 条件**GET**技术

2.3 文件传输协议：FTP

本地主机上的用户，向远程主机上传或者下载文件。

- ✓ 用户通过一个FTP用户代理与FTP服务器交互。



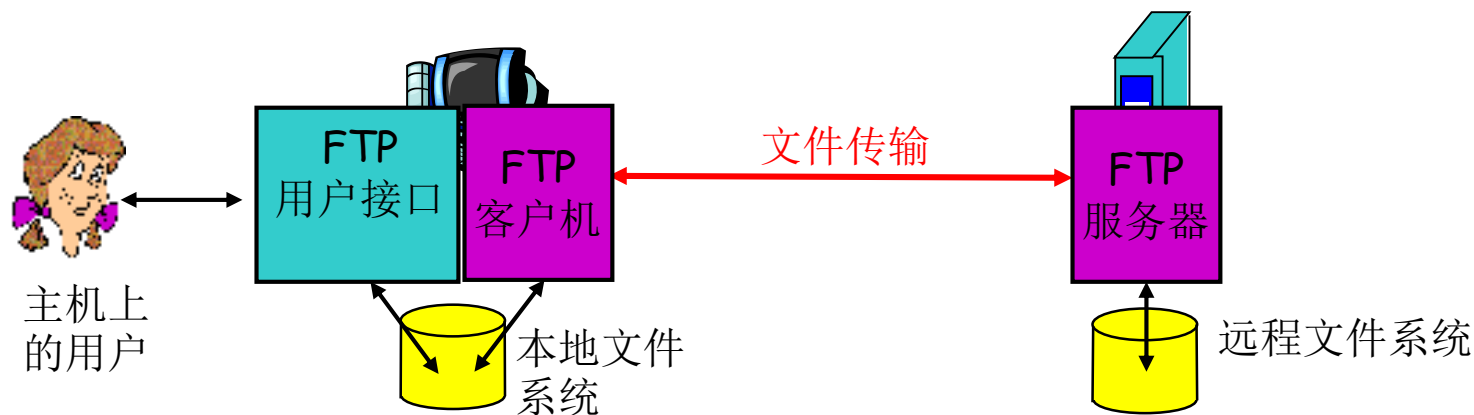
FTP概述

- **文件传送协议** FTP (File Transfer Protocol) 是因特网上使用得最广泛的文件传送协议。
- FTP 提供交互式的访问，允许客户指明文件的类型与格式，并允许文件具有存取权限。
- FTP 屏蔽了各计算机系统的细节，因而适合于在异构网络中任意计算机之间传送文件。
- RFC 959 很早就成为了因特网的正式标准。

文件传输过程

- ✓ 用户提供远程主机的主机名：在本地主机的FTP客户机进程与远程主机FTP服务器进程之间**建立TCP连接**；
- ✓ 提供用户标识和口令：在该TCP连接上向服务器传送。
- ✓ 服务器验证通过后，进行文件传送（双向）：

将本地文件系统中的文件传送到远程文件系统（**上传**）
或从远程文件系统中得到文件（**下载**）



FTP与HTTP比较

都是文件传输协议，并运行在TCP上。

✓ *FTP使用了两个并行的TCP连接:*

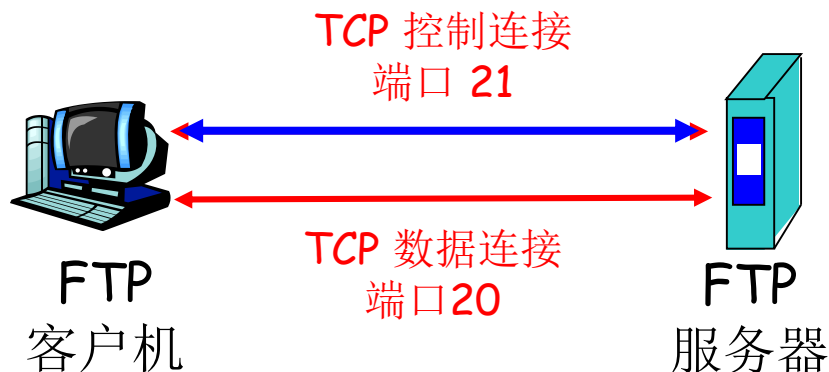
- ◆ 控制连接:
- ◆ 数据连接:



控制连接

用于在两主机间**传输控制信息**（如用户标识、口令等）

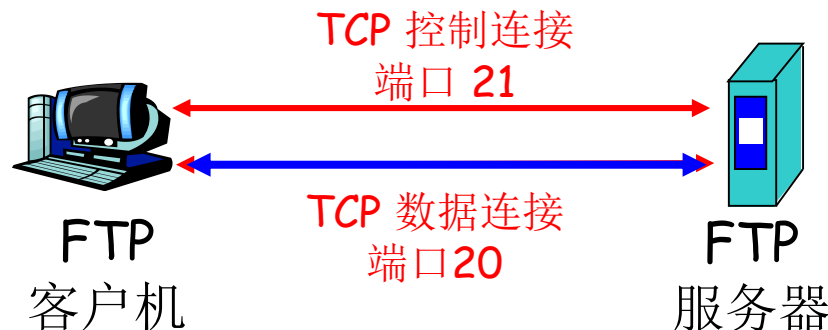
- ✓ FTP会话开始前，FTP的客户机与服务器在**21号端口**上建立一个用于控制的连接。
- ✓ FTP的客户机通过该连接发送用户标识和口令，或改变远程目录的命令。



数据连接

用于准确传输文件。

- ✓ 当服务器收到一个文件传输的命令后(从远程主机上读或写)，在20端口发起一个到客户机的数据连接。
- ✓ 在该数据连接上传送一个文件并关闭连接。
- 控制连接是持久的：在整个用户会话期间一直保持；
- 数据连接是非持久的：会话中每进行一次文件传输，都需要建立一个新的数据连接。



FTP与HTTP比较

- ✓ *FTP* 的控制信息是带外传送 (out-of-band) :

使用分离的控制连接;

HTTP 的控制信息是带内传输(in-band):

请求和响应都是在传输文件的TCP连接中发送。

- ✓ *FTP* 协议是有状态的:

FTP服务器对每个活动用户会话的状态进行追踪, 并保留; 限制同时会话的总数。

HTTP 协议是无状态的: 不对用户状态进行追踪。

FTP命令, 响应

命令示例:

- ❑ 经控制信道以ASCII 文本发送
- ✓ **USER *username***
- ✓ **PASS *password***
- ✓ **LIST**返回当前目录中的文件列表
- ✓ **RETR *filename***获取 (get) 文件
- ✓ **STOR *filename*** 存储 (puts)文件到远程主机

返回码示例:

- ❑ 状态码和短语(如在HTTP中的那样)
- ✓ **331 Username OK, password required**
- ✓ **125 data connection already open; transfer starting**
- ✓ **425 Can't open data connection**
- ✓ **452 Error writing file**

2.4 因特网中的电子邮件

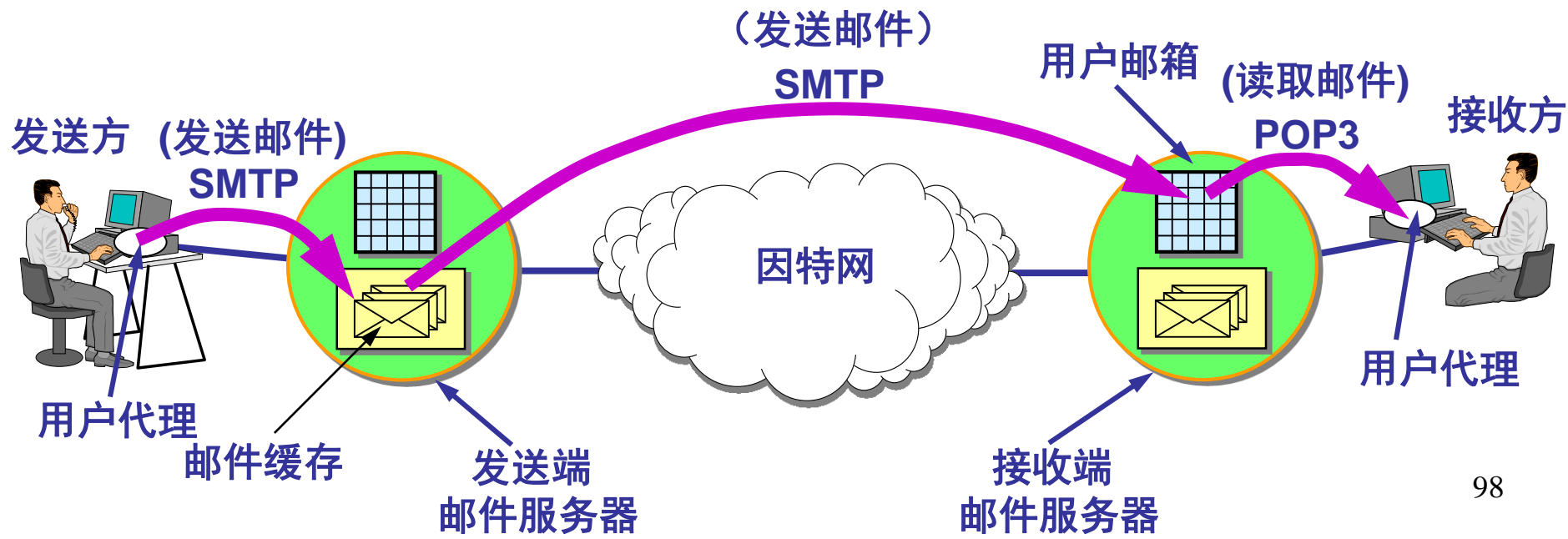
电子邮件快速、多方接收，包含附件、超链接、图像、声音、视频等等。

本节讨论电子邮件的核心，即应用层协议。

因特网电子邮件系统的总体结构

三部分：

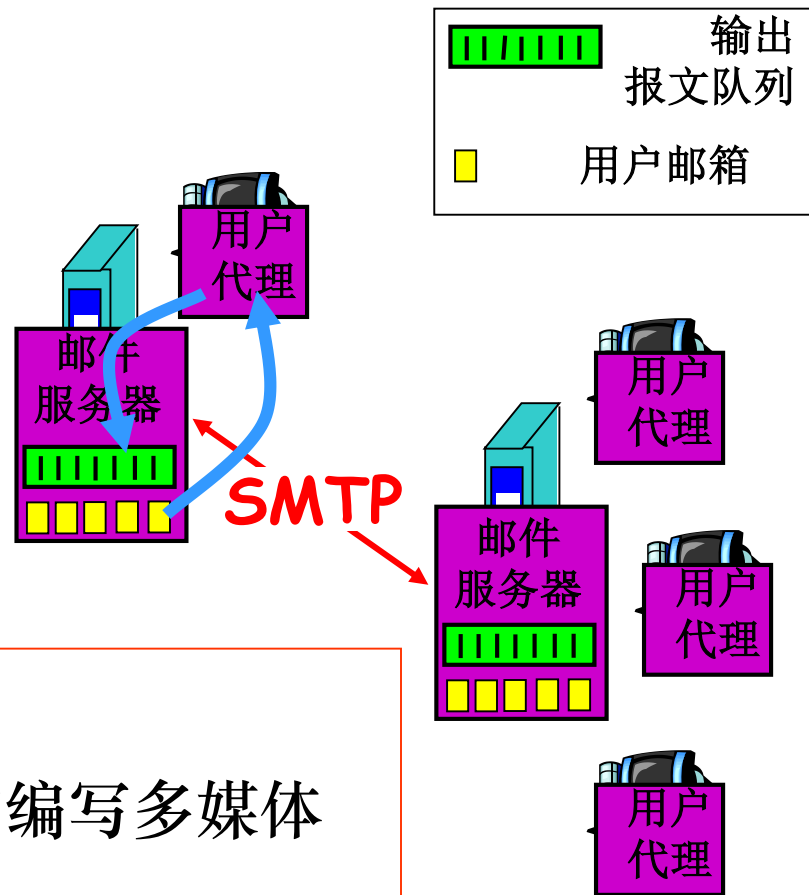
- ✓ 用户代理
- ✓ 邮件服务器
- ✓ 简单邮件传输协议



1、用户代理(user agent)

邮件阅读器。允许用户阅读、回复、发送、保存和撰写报文。

- ✓ 当用户完成邮件撰写时，**邮件代理向其邮件服务器发送邮件**，并存放在发送队列中。
- ✓ 当用户想读取一封邮件时，**邮件代理从其邮件服务器的邮箱中获取该邮件。**



□ 种类：

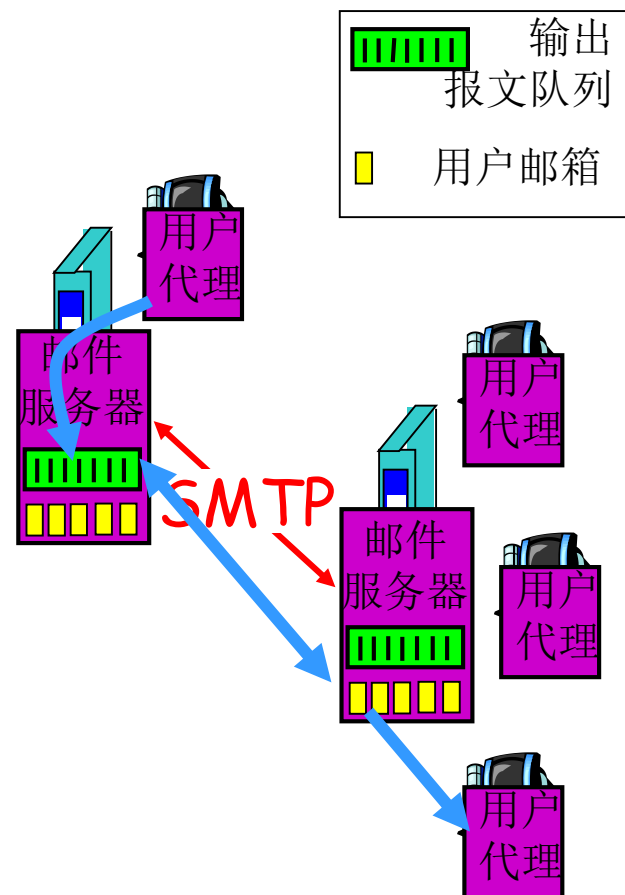
- ✓ GUI（图形用户接口）：阅读和编写多媒体邮件。如Outlook、Foxmail等。

2、邮件服务器(mail server)

- ✓ **邮箱**：发送给用户的邮件报文。
- ✓ **报文队列**：用户要发出的邮件报文。

□ 邮件发送主要过程：

- ✓ 邮件**保存**到发送方报文队列
- ✓ 通过**SMTP协议转发**到接收方邮件服务器，保存到相应邮箱中
- 若投递失败，发送方将其保存在一个报文队列中，以后每30分钟发送一次，若几天后仍未成功，将该报文删除，并通知发送方。
- 用户访问自己邮箱时，邮件服务器对其身份进行验证(用户名和口令)。



3、简单邮件传送协议SMTP

从发送方的邮件服务器向接收方的邮件服务器发送邮件。

- ✓ 应用层协议。
- ✓ 使用TCP可靠数据传输服务。
- 包括两部分：
 - ✓ 客户端：在发送方邮件服务器上运行；
 - ✓ 服务器端：在接收方邮件服务器上运行。

每个邮件服务器上都有SMTP的客户端和服务端。

本节内容

2.4.1 SMTP

2.4.2 SMTP与HTTP比较

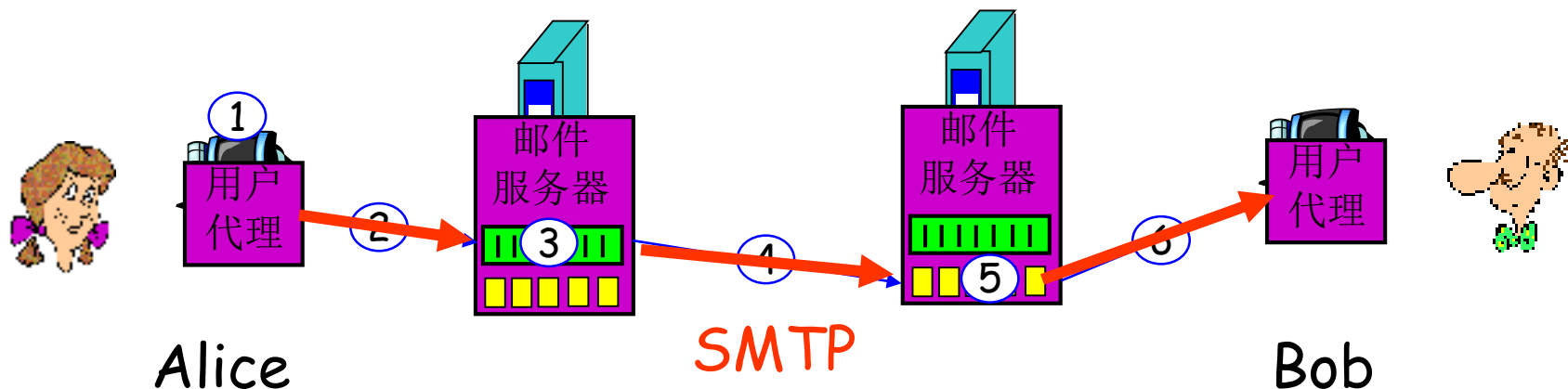
2.4.3 邮件报文格式和MIME

2.4.4 邮件访问协议

2.4.1 SMTP

把一封邮件从发送邮件服务器传送到接收邮件服务器的过程：如Alice 向 Bob发送报文

- 1) Alice启动邮件代理，提供接收方的邮件地址，撰写邮件
- 2) 用户代理把报文发给其邮件服务器，放在发送队列中
- 3) SMTP的客户机侧创建与Bob的邮件服务器的TCP连接
- 4) SMTP通过TCP连接发送报文
- 5) Bob的邮件服务器接收并将该报文放入Bob的邮箱
- 6) Bob调用其用户代理来读报文



说明

- 客户使用TCP来可靠传输邮件报文到服务器端口号25。
 - ✓ 建立TCP连接：
 - ✓ 握手： 指明收发双方的邮件地址
 - ✓ 邮件报文的传输
 - ✓ 结束： 关闭TCP连接

说明

- SMTP不使用中间邮件服务器发送邮件，即TCP 连接是从发送方到接收方的直接相连。

如果接收方的邮件服务器没有开机，该邮件仍保留在发送方邮件服务器上，并在以后进行再次传送。
邮件不会在某个中间邮件服务器停留。

SMTP客户和服务器的命令交互

```
S: 220 hamburger.edu
C: HELO crepes.fr
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
```

2.4.2 与HTTP的对比

共同点

- ✓ 都用于从一台主机向另一台主机传送文件
 - ◆ **HTTP**用于从Web服务器向Web客户机(浏览器)传送文件(对象);
 - ◆ **SMTP**用于从一个邮件服务器向另一个邮件服务器传送文件(电子邮件报文)。
- ✓ 持久HTTP和SMTP都使用持久连接。

区别

□ HTTP是拉协议：用户使用HTTP从服务器拉取信息。
其TCP连接是由想获取文件的机器发起。

SMTP是推协议：发送邮件服务器把文件推向接收邮件服务器，其TCP连接是由要发送文件的机器发起。

区别

□ SMTP使用7位ASCII码格式:

对一些包含了非7位ASCII字符的报文或二进制数据(如图片、声音), 需要按照7位ASCII码进行编码, 再传送。

在接收方需要解码还原为原有报文。

HTTP数据没有该限制。

□ 对含有文本和图形 (或其他媒体类型)的文档:

- ✓ HTTP把每个对象封装在它各自的HTTP响应报文中发送
- ✓ 电子邮件则把所有报文对象放在一个报文中。

2.4.3 邮件报文格式和MIME

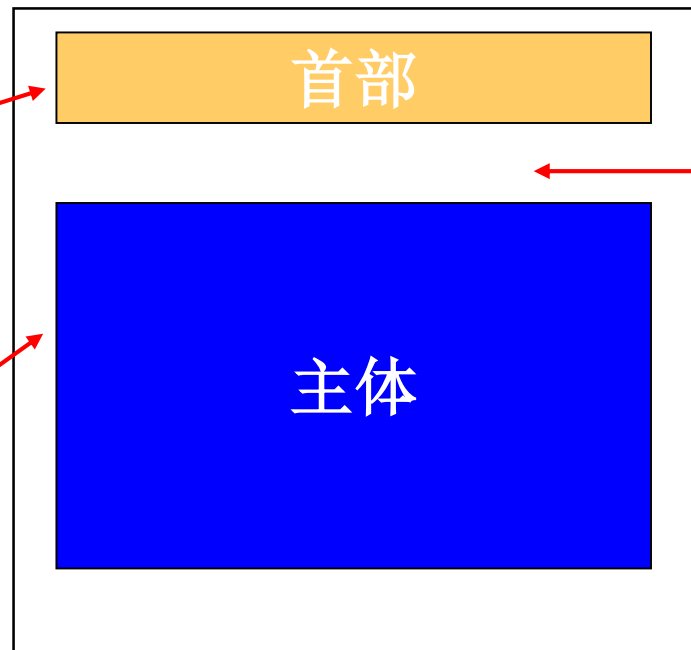
邮件报文格式

□ 首部行（收发人、主题）

- To:
- From:
- Subject:

□ 主体

- “报文”，均为ASCII字符



由RFC 822 规定

MIME（多用途因特网邮件扩展）

- ✓ SMTP只传送7位的ASCII码。
- ✓ SMTP不能传送可执行文件或其他的二进制对象。
- ✓ SMTP 限于传送 7 位的 ASCII 码。许多其他非英语国家的文字（如中文、俄文，甚至带重音符号的法文或德文）就无法传送。

MIME：用于非ASCII数据传输。将非ASCII数据编码后传输，接收方再解码还原。

- ✓ 增加新的MIME邮件首部
- ✓ 采用某种编码

例：传输一个jpeg图形

jpeg格式的静止图像

base64编码：用于
二进制文件

MIME 版本

使用数据编码的方法

多媒体数据
类型/子类型

编码数据

```
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

base64 encoded data .....
.....
.....base64 encoded data
```


MIME 的特点

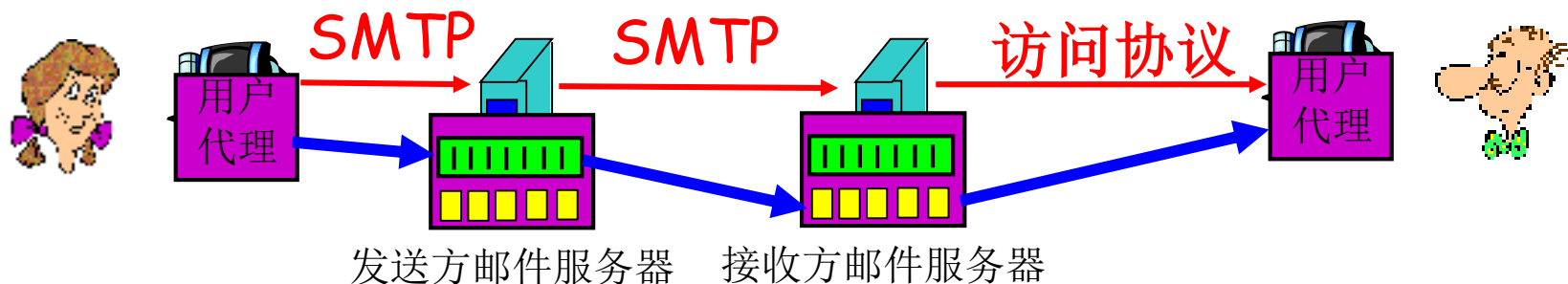
- MIME 并没有改动 SMTP 或取代它。
- MIME 的意图是继续使用目前的[RFC 822]格式，但增加了邮件主体的结构，并定义了传送非 ASCII 码的编码规则。

MIME 和 SMTP 的关系



2.4.4 邮件访问协议

- 发送方：用户代理用SMTP将邮件推入其邮件服务器
→ 邮件服务器再用SMTP将邮件转发到接收方的邮件服务器
 - 接收方：通过其用户代理使用一个邮件访问协议（不是SMTP），从其邮件服务器上取回邮件。
- 取邮件是一个拉操作，而SMTP协议是一个推协议。



□ 邮件访问协议：从服务器获取邮件。

□ 种类：

POP3(第三版的邮局协议)

IMAP(因特网邮件访问协议)

HTTP

1、POP3

POP3简单、功能有限。

在用户代理打开了一个到邮件服务器(服务器)端口110上的TCP连接后，开始工作。

工作步骤（三阶段）

- ✓ **特许阶段：** 用户代理发送用户名和口令获得下载邮件的特许。（身份认证）
- ✓ **事务处理阶段：** 用户代理取回报文，可对邮件进行某些操作。如做删除标记、取消删除标记、获取统计信息等。
- ✓ **更新阶段：** 邮件服务器删除带有删除标记的报文，结束POP会话。

工作步骤（三阶段）：

身份认证阶段

authorization phase

❑ 客户命令:

- user username
- pass password


❑ 服务器响应

- +OK
- -ERR


事物处理阶段

transaction phase, client:

- ❑ list: 列出邮件编号
- ❑ retr: 按编号取邮件
- ❑ dele: 删除
- ❑ 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
```

2、IMAP

- ❑ POP3缺陷：功能简单。POP3会话是无状态的
- ❑ IMAP：功能强
 - ✓ 在用户的PC机上运行IMAP客户程序，然后与ISP的邮件服务器上的IMAP服务器程序建立TCP连接。
 - ✓ 用户在自己的PC机上就可以操纵邮件服务器的邮箱，就像在本地操纵一样，是一个联机协议。
 - ✓ IMAP服务器把每个报文与一个文件夹联系起来。
 - ✓ IMAP 还允许收件人只读取邮件中的某一个部分
 - ✓ IMAP服务器维护用户的会话状态。
 - ✓ 未发出删除命令前，一直保存在邮件服务器
 - ✓ 实现起来复杂。

3、基于web的电子邮件

1995年12月Hotmail 引入该技术。用户使用浏览器收发电子邮件。

□ 用户代理是普通的浏览器，用户和其远程邮箱之间的通信通过HTTP进行：

✓ 发件人使用HTTP 将电子邮件报文从其浏览器发送到其邮件服务器上；

✓ 收件人使用HTTP从其邮箱中取一个报文到浏览器；

□ 邮件服务器之间发送和接收邮件时，使用SMTP。

□ 用户可以在远程服务器上以层次目录方式组织报文。

如，Hotmail、Yahoo、Mail等。

2.5 DNS: 因特网的目录服务

□ 标识主机的两种方式:

- ✓ **主机名**: 由不定长的字母和数字组成。便于记忆。用于人记忆识别

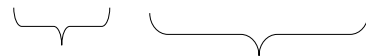
如 www.baidu.com

路由器处理困难。

- ✓ **IP地址**: 由4个字节组成，有着严格的层次结构。

路由器容易处理。用于分组寻址

如IP地址（点分十进制）： 121.7.106.83



网络号 主机号

2.5.1 DNS提供的服务

报文在网络中传输，使用IP地址。

- 域名系统*DNS* (*Domain Name System*): 进行主机名到IP地址的转换。
- ✓ 一个由分层的DNS服务器实现的分布式数据库
- ✓ 允许主机查询分布式数据库的应用层协议;

说明

- ✓ 使用客户-服务器模式运行在通信端系统之间；
- ✓ 再通信的端系统之间通过下面的端到端运输协议来传递DNS报文。
- ✓ DNS协议运行在UDP之上，使用53号端口。
- ✓ DNS通常直接由其他的应用层协议（包括HTTP、SMTP和FTP)使用，以将用户提供的主机名解析为IP地址。用户只是间接使用。

❑ 例，某个用户主机上的一个浏览器访问某个Web页，
www.someschool.edu/index.html

用户主机要**将一个HTTP请求报文发送到Web服务器**
www.someschool.edu，需先得到相应的IP地址。

❑ 过程如下：

- ✓ 用户主机上运行 DNS应用的客户机端。
- ✓ 浏览器从URL中解析出主机地址，传给DNS客户机端。
- ✓ DNS客户机向DNS服务器发送一个包含主机名的请求；
- ✓ DNS客户机收到含有对应主机名的IP地址的回答报文；
- ✓ 浏览器向该IP地址指定的HTTP服务器发起一个TCP连接。

增加一定时延。

DNS提供的服务

❑ 主机名到IP地址的转换

❑ 主机别名

- 规范名和别名：通过DNS可以得到主机别名对应的规范主机名及IP地址。

因特网采用了层次树状结构的命名方法。

… . 三级域名 . 二级域名 . 顶级域名

比如www.google.com 的规范名是

www.l.google.com

❑ 邮件服务器别名

❑ 负载分配

2.5.2 DNS工作原理概述

Domain Name System:

域名系统

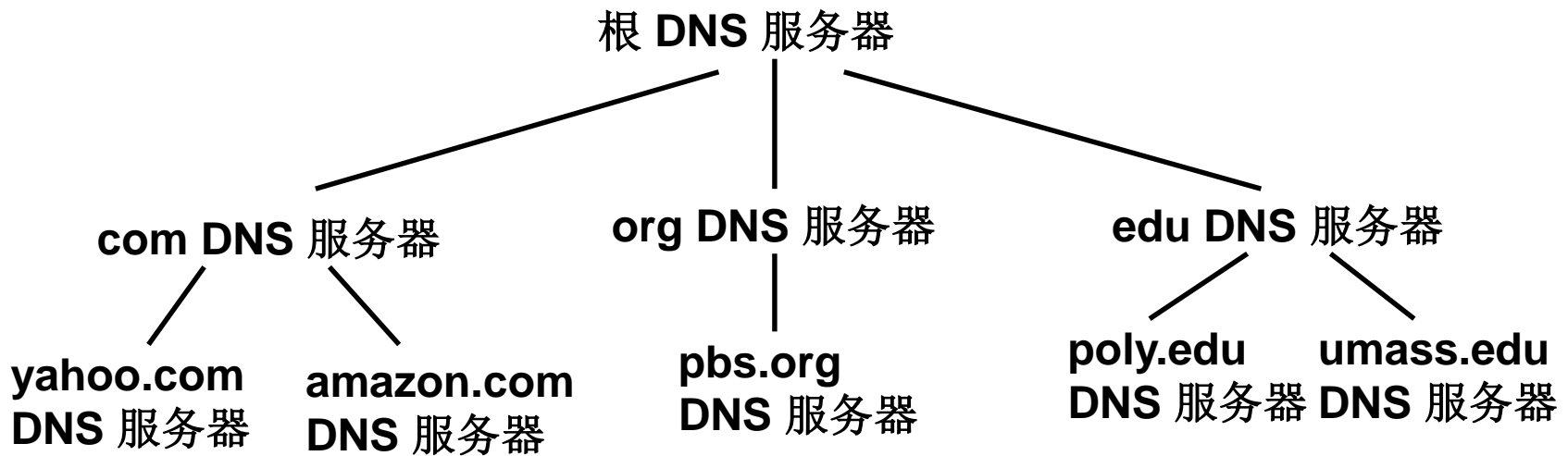
- ❑ **分布式数据库**: 一个由分层DNS服务器实现的分布式数据库
- ❑ **应用层协议**: DNS服务器实现域名转换 (域名/地址转换)

为什么不集中式DNS?

- ❑ 单点故障
- ❑ 巨大访问量
- ❑ 远距离集中式数据库
- ❑ 维护

不可扩展!

分布式、层次数据库



客户机怎样决定主机名www.amazon.com的IP地址？

- ❑ 客户机查询根服务器得到com DNS服务器
- ❑ 客户机查询com DNS服务器得到amazon.comDNS服务器
- ❑ 客户机查询amazon.comDNS服务器得到www.amazon.com的IP地址

DNS: 根名字服务器Root name servers

<http://www.root-servers.org/>

❑ 根名字服务器负责记录顶级域名服务器的信息

13 root name servers

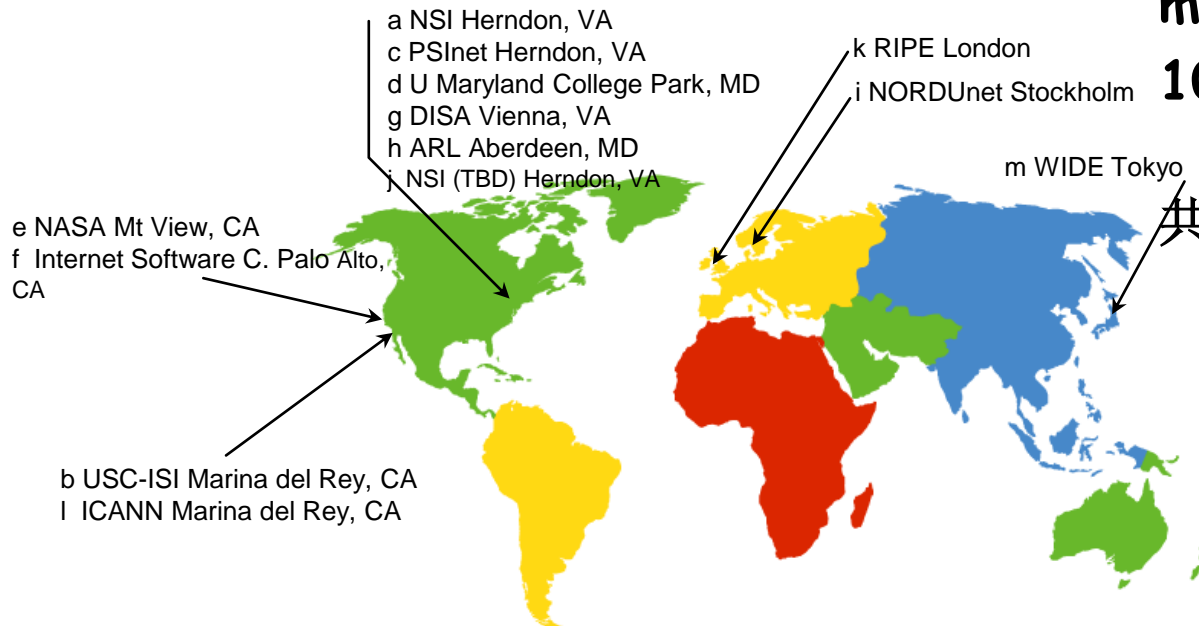
a.root-servers.net

...

m.root-servers.net

**10个美国，1个英国，1
个瑞典，1个日本**

共386台服务器



根域名服务器，分布在全球各地。这样做的目的是为了更方便用户，使世界上大部分 DNS 域名服务器都能就近找到一个根域名服务器。真实的根服务器在**2014年1月25日**的数据为**386**台，分布于全球各大洲。

举例：根域名服务器 f 的地点分布图



- 根域名服务器并不直接把域名直接转换成 IP 地址。
- 在使用迭代查询时，根域名服务器把下一步应当找的顶级域名服务器的 IP 地址告诉本地域名服务器。

顶级域服务器

- **顶级域服务器：** 负责顶级域名 com, org, net, edu, etc, 和所有国家的顶级域名 cn,uk, fr, ca, jp.
 - Network solutions 公司维护com顶级域的TLD服务器
 - Educause 公司维护edu顶级域的 TLD服务器

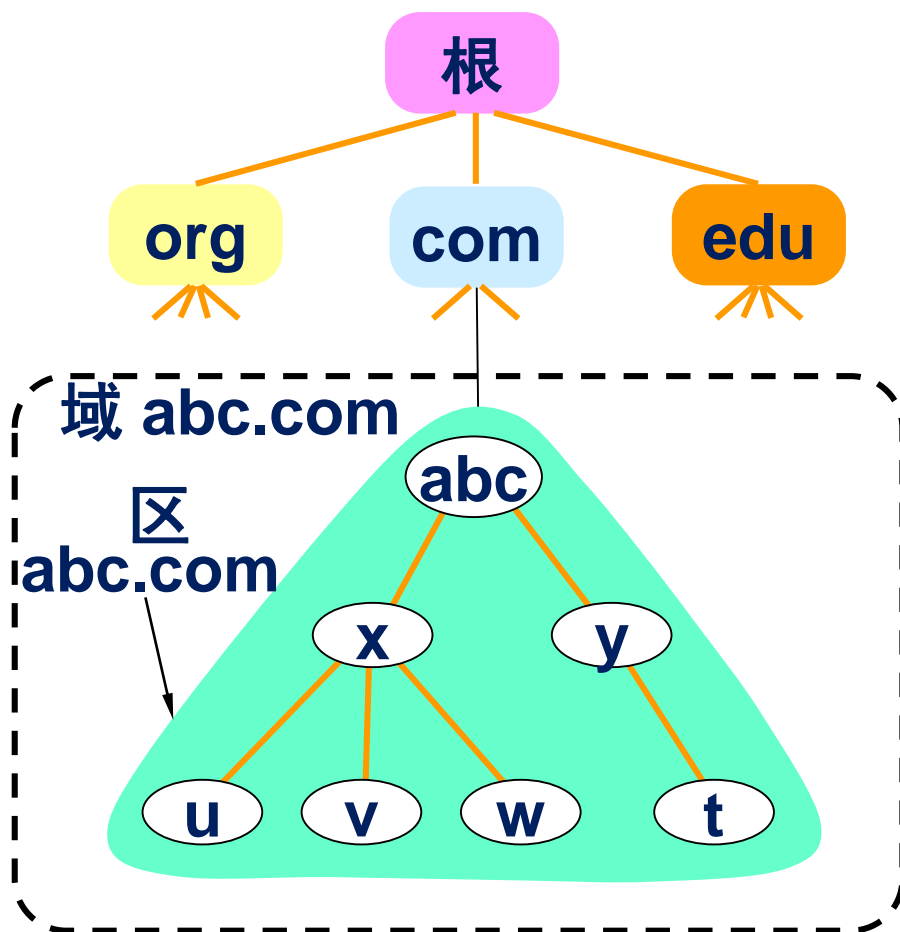
权威DNS服务器

- **权威DNS服务器：**在因特网上具有公共可访问主机（如Web服务器和邮件服务器）的每个组织机构必须提供公共可访问的DNS记录，这些记录将这些主机的名字映射为IP地址。组织机构的权威DNS服务器负责保存这些DNS记录。
 - 多数大学和公司维护它们的基本权威DNS服务器

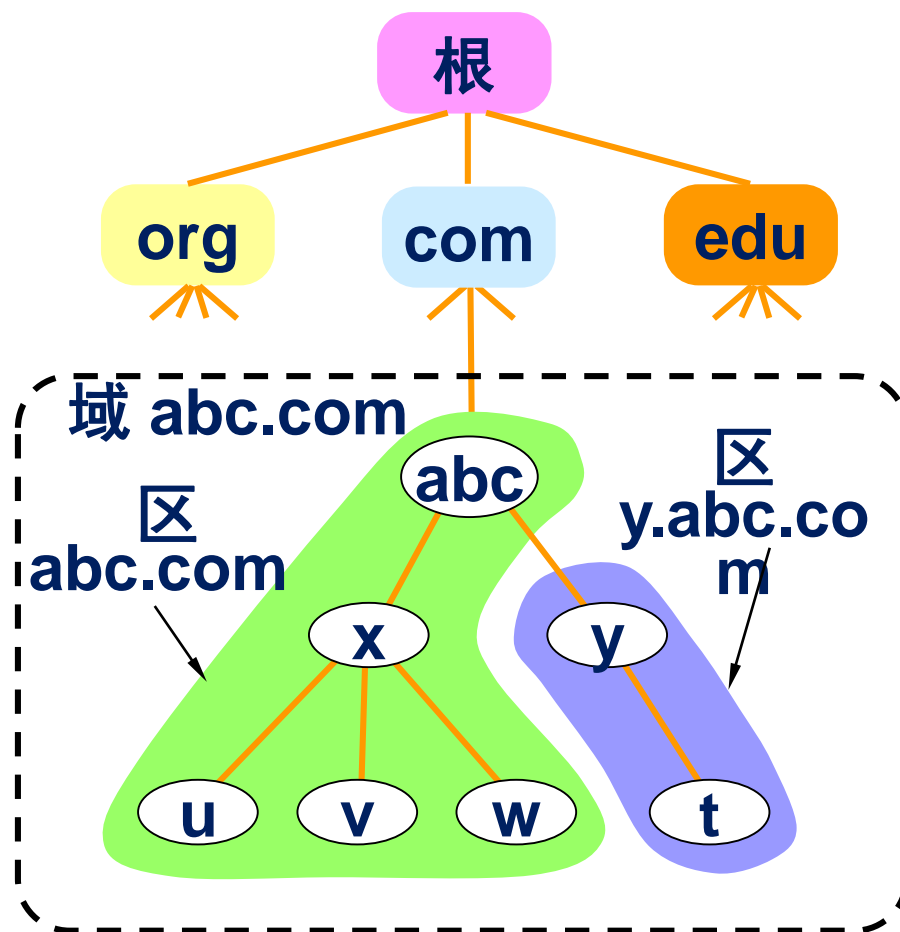
权威DNS服务器

- 一个服务器所负责管辖的（或有权限的）范围叫做区(zone)。
- 各单位根据具体情况来划分自己管辖范围的区。但在一个区中的所有节点必须是能够连通的。
- 每一个区设置相应的权威域名服务器，用来保存该区中的所有主机的域名到IP地址的映射。
- DNS 服务器的管辖范围不是以“域”为单位，而是以“区”为单位。

区的不同划分方法举例

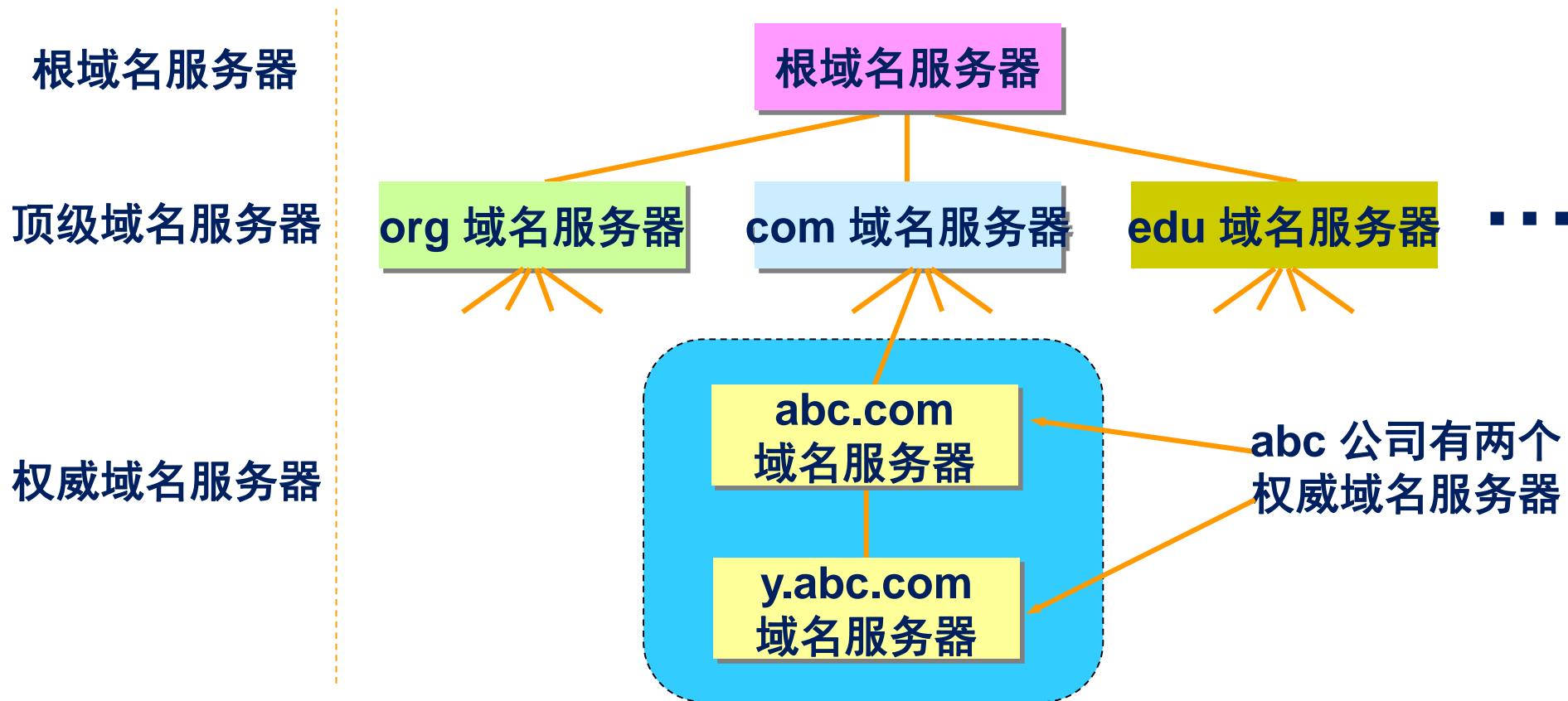


(a) 区 = 域



(b) 区 < 域

树状结构的 DNS 域名服务器



本地DNS服务器

- ❑ 严格来说不属于DNS的层次结构
- ❑ 每个ISP（如居民区ISP、公司、大学）都有一个本地DNS
 - 也叫默认服务器

查看网络状态窗口的信息



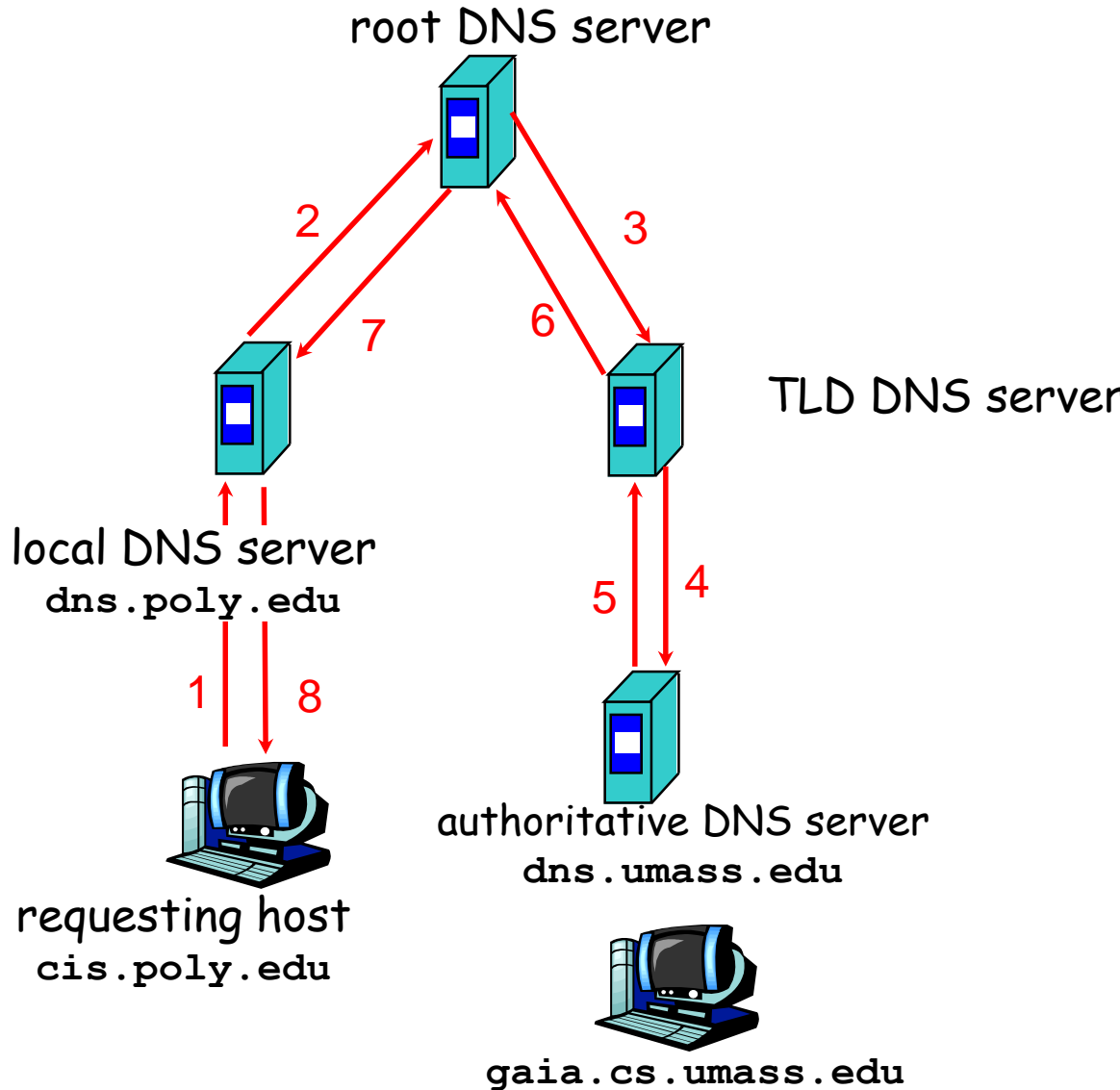
本地DNS服务器

- ❑ 严格来说不属于该服务器的层次结构
- ❑ 每个ISP（如居民区ISP、公司、大学）都有一个本地DNS
 - 也叫默认服务器
- ❑ 当主机发出DNS请求时，该请求被发往本地DNS服务器。
 - 起着代理的作用，转发请求到层次结构中。

DNS查询方法一

递归查询:

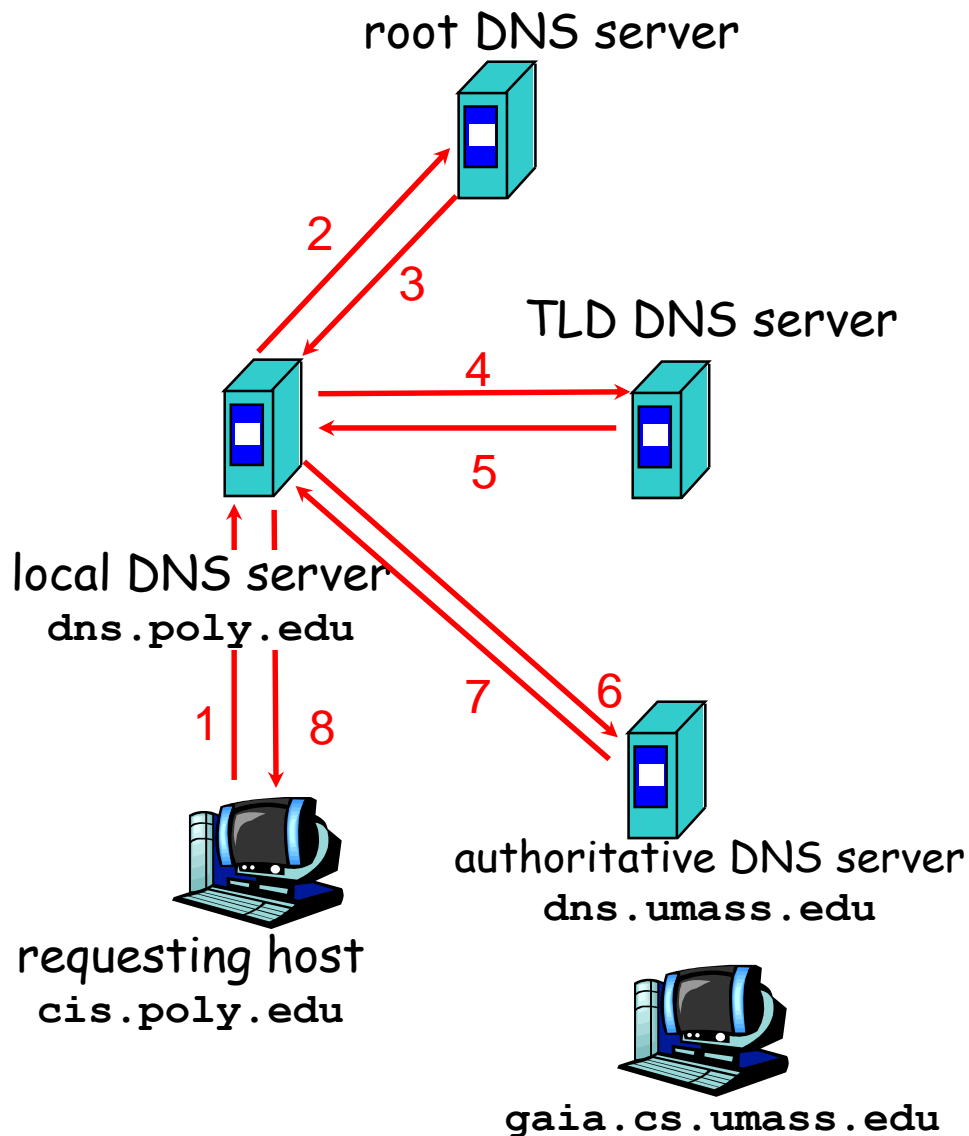
- 名字解析的负担交给被查询的名字服务器
- 被查询的名字服务器负载重?



DNS查询方法二

迭代查询:

- ❑ 被查询的名字服务器 回复可以被查询的名字服务器的IP地址
- ❑ “我不知道它的名字，但是可以问服务器”



DNS缓存和权威DNS记录更新

- ❑ 一旦名字服务器获得DNS映射, 它将缓存该映射到局部内存
 - 服务器在一定时间后将丢弃缓存的信息
 - 本地DNS服务器可以缓存TLD服务器的IP地址
 - 因此根DNS服务器不会被经常访问
- ❑ 权威DNS服务器记录更新: IETF动态更新/通报机制
 - RFC 2136

DNS记录

DNS: 存储资源记录(RR, Resource Records)提供主机名到IP映射

RR 格式: (name, value, type,ttl)

□ Type=A (Address)

- name = 主机名
- value = IP地址

□ Type=NS

(name server)

- name = 域名 (如 [foo.com](#))
- value = 该域权威名字服务器的主机名

□ Type=CNAME (canonical)

- name = 主机别名
[www.ibm.com](#)的真名为
servereast.backup2.ibm.com

- value = 真实的规范主机名

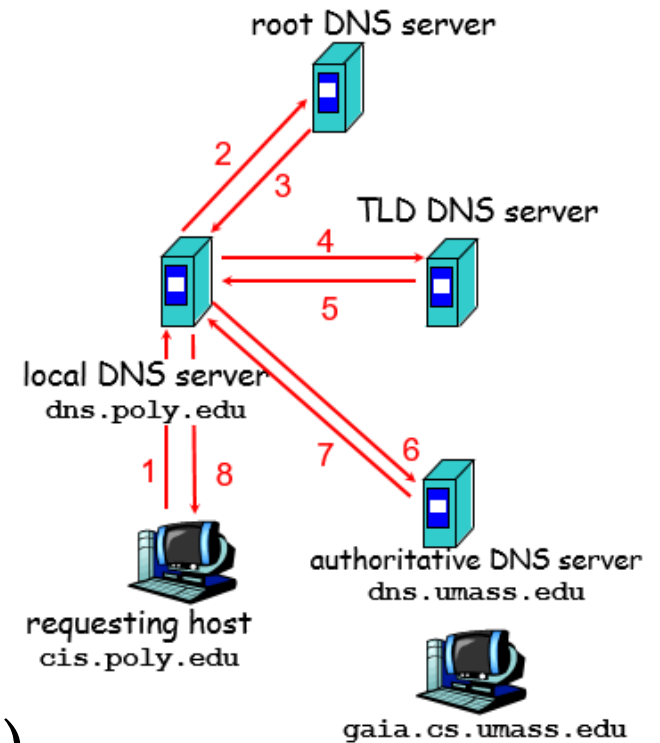
□ Type=MX (mail exchange)

- name = 邮件服务器的主机别名
- value = 邮件服务器的真实规范主机名

- 如果一台DNS服务是用于某特定主机名的权威DNS服务器，那么该服务器会有一条包含该主机名的类型A记录
(gaia.cs.umass.edu, 128.119.40.12, A)

- 查询链中的DNS服务器如果不是某主机名的权威服务器，它将包含一条NS记录，该记录对应于包含主机名的域；它还将包括一条类型A的记录，记录对应权威服务器的IP地址。

(Umass.edn, dns. Umass.edn, NS)
(dns. Umass.edn,128.119.40.111, A)



DNS协议, 消息

DNS协议：查询报文与应答报文，但具有同样的报文格式

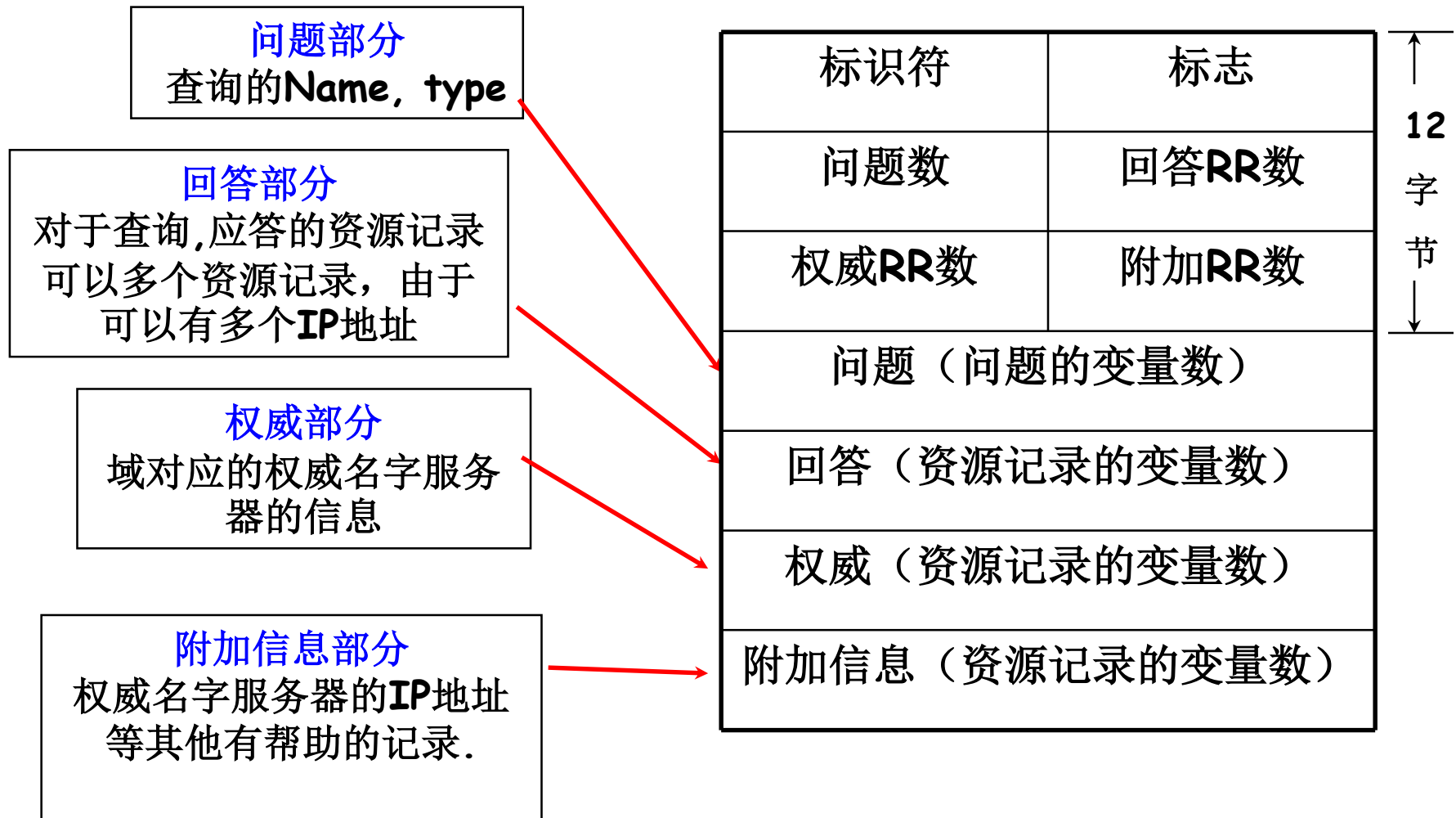
报文头部

- ❑ 标识符：16位，查询和应答报文使用相同的标识符
- ❑ 标志：有若干个标志构成，分别标识不同的功能
 - 查询/应答—0/ 1
 - 查询希望是/非递归查询—1/0
 - 应答可/否获得(支持)递归查询—1/0
 - 应答是/否来自权威名字服务器—1/ 0

标识符	标志
问题数	回答RR数
权威RR数	附加RR数
问题（问题的变量数）	
回答（资源记录的变量数）	
权威（资源记录的变量数）	
附加信息（资源记录的变量数）	

↑
12
字
节
↓

DNS协议, 消息



在DNS数据库中插入记录

例子：刚刚创建一个“网络乌托邦”公司

- ❑ 如果你想在**注册登记机构**注册你的域名network.com，则
 - 需要提供你自己的基本权威DNS服务器和辅助权威DNS服务器的名字和IP地址
 - 该注册登记机构将下列两条资源记录插入注册机构的DNS系统中：
(network.com, dns1.network.com, NS)
(dns1.network.com, 212.212.212.1, A)
- ❑ 如果你想建立一个网站，则可以将网址www.network.com以类型A的方式记录到你的权威DNS服务器dns1.network.com中。
- ❑ 如果你想建一个邮件服务器，则可以将mail.network.com以类型MX的方式记录到你的权威DNS服务器dns1.network.com中。
- ❑ **人们怎样得到你的Web站点的IP地址呢？**

DNS攻击

- ❑ DDoS攻击：对根域名服务器或顶级域名服务器发起拒绝服务攻击
- ❑ 重定向攻击：中间人攻击、DNS中毒攻击（发送欺骗的域名解析结果给DNS服务器）
- ❑ 利用DNS实现DDoS攻击：DNS反弹式拒绝服务攻击(DNS reflector attacks, 又称DNS amplification attacks)。伪造客户地址向大量的dns服务器发出请求，导致客户无法访问dns服务器进行域名解析。

2.6 P2P 文件共享

位于网络边缘的PC机(对等方peer)互相之间可以直接获取对象。

例，用户Alice使用P2P文件共享应用程序下载MP3。

说明：每个参与的对等方既是内容的消费者也是内容的发布者（**下载同时也向其他用户上传**）。

过程

- ❑ Alice在PC机上运行一个P2P文件共享应用软件(对等方);
- ❑ 通过ADSL间歇地接入因特网（无固定IP地址）
- ❑ 使用该应用程序搜索一首MP3歌曲；
- ❑ 显示一张有该首歌曲的对等方列表：
 - ✓ 所有在线对等方，并愿意共享该首音乐的MP3拷贝
 - ✓ 对等方都是普通的PC，是因特网用户。
 - ✓ 列表中提供一些附加信息，如接入带宽和下载时间。
- ❑ 选择一个对等方（Bob）并向其请求该MP3文件；
 - ✓ 两个用户之间建立一个直接的TCP连接；
 - ✓ MP3文件从Bob的PC机向Alice的PC发送；
 - ✓ 下载期间若偶然断开，可从其他对等方继续下载。

P2P文件共享特点

- ❑ **直接在对等方间传输：**所有内容不经过第三方服务器
- ❑ **高度的可扩展能力：**利用众多对等方集合中的资源去分发内容
- ❑ 在文件传输时，请求的对等方是客户机，被选中的对等方是服务器。
- ✓ 服务器对等方使用文件传输协议向客户机对等方传送。
 - ◆ 比如，可以通过传送“HTTP请求”和“HTTP响应”报文进行。
- ✓ 所有的对等方必须既能运行文件传输协议的客户机端程序，又能运行服务器端程序。
 - ◆ 对等方既是一个客户机，又是一个瞬时服务器。

内容定位体系结构

一个对等方如何确定？ 哪个对等方有所需要的内容？

(1) 集中式目录

(2) 查询洪泛

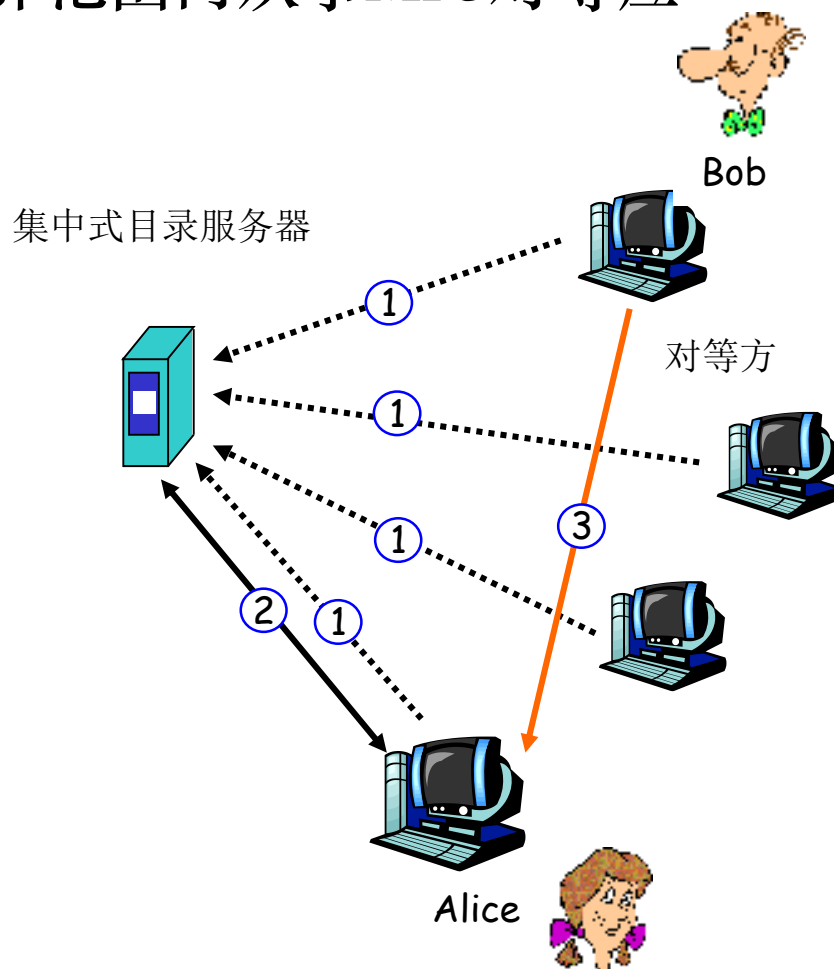
(3) 利用不均匀性 (KaZaA)

(1) 集中式目录

源于Napster公司（第一家世界范围内从事MP3对等应用程序）设计。

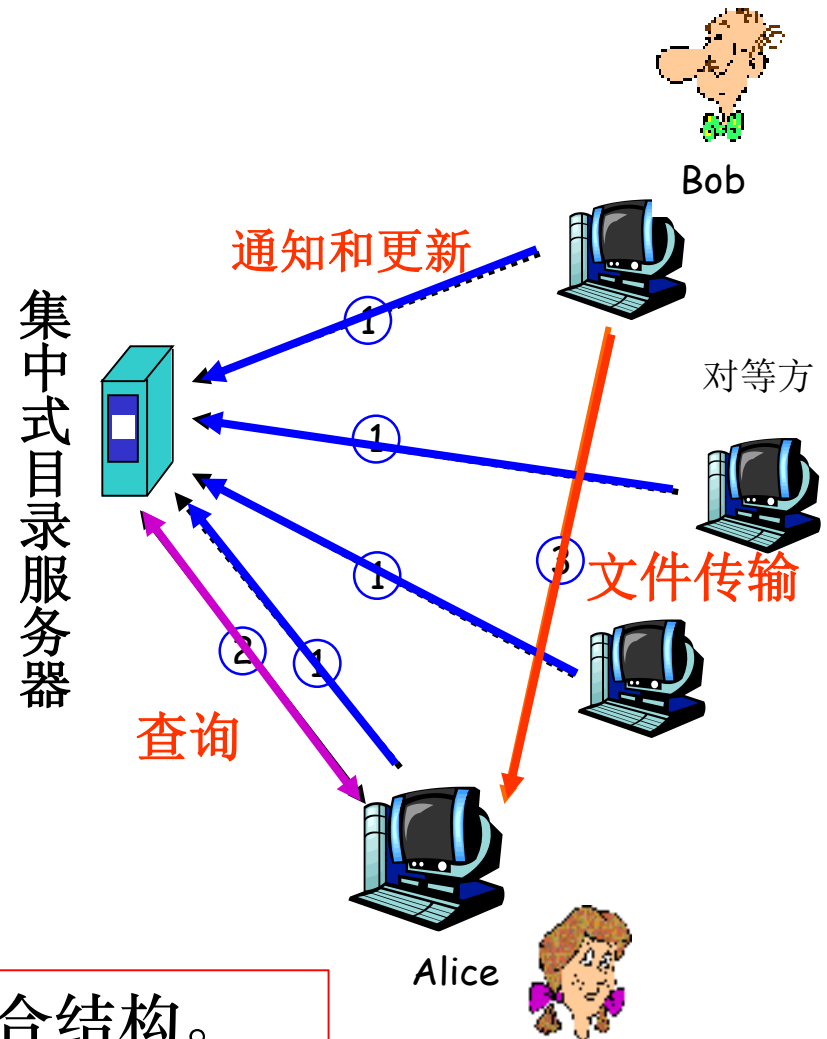
□ **目录服务器**(大型服务器或服务场)：提供目录服务。

收集可共享的对象，建立集中式的动态数据库（对象**名称到IP地址**的映射）。



主要工作

- **通知**: 对等方启动时, 将其IP地址及可共享内容通知目录服务器;
- **查询内容**: 用户查询需要共享的对象。
如, Alice 查询某首歌。
- **获取内容**: 来自Bob。
- **更新**: 当对等方获得新对象或删除对象时, 通知目录服务器更新。



属于客户机/服务器与P2P的混合结构。

集中式目录存在的问题

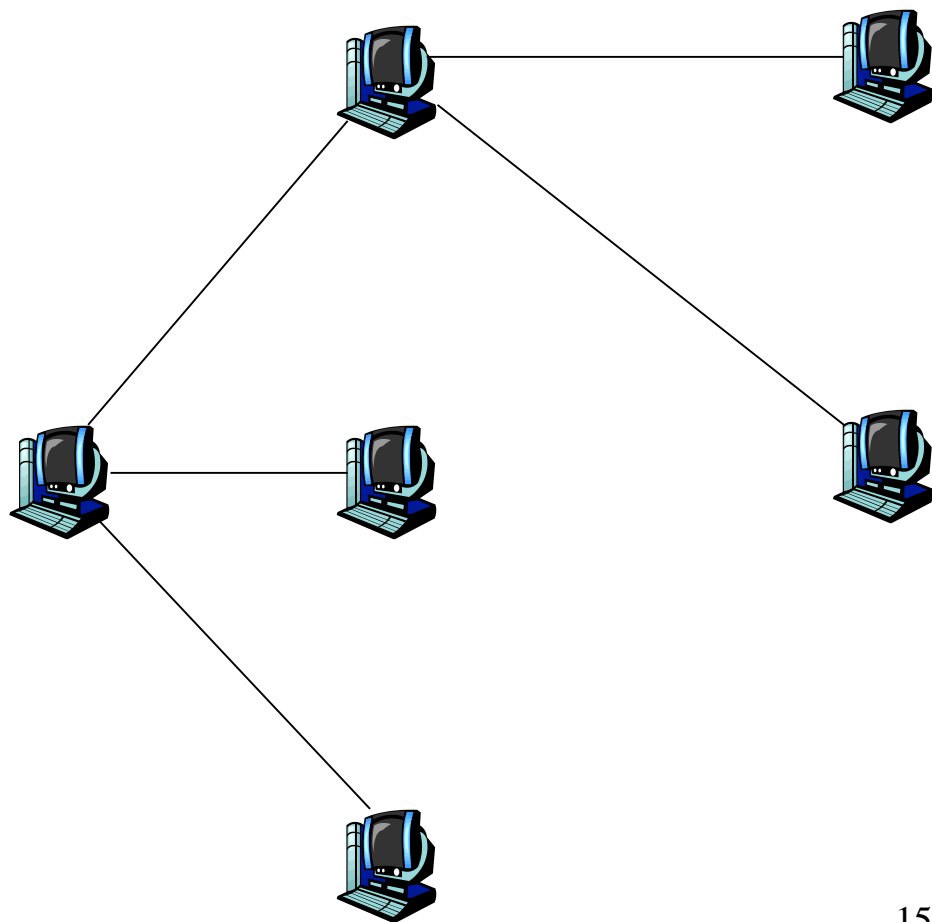
- ❑ 单点故障
- ❑ 性能瓶颈
- ❑ 侵犯版权
- ❑ 可靠性：文件传输是分散的，但定位内容的过程是高度集中的。

(2) 查询洪泛: Gnutella

- ✓ 是一个公共域文件共享应用程序。
- ✓ 全分布方式: 无中心服务器
- ✓ 许多Gnutella客户机实现协议: 运行在对等方。
- 实现:
 - ✓ 对等方先形成一个抽象的逻辑网络 (覆盖网络):
 - ✓ 通过洪泛查询, 找到所需对象:
如, 某个对等方Alice想得到一个对象。

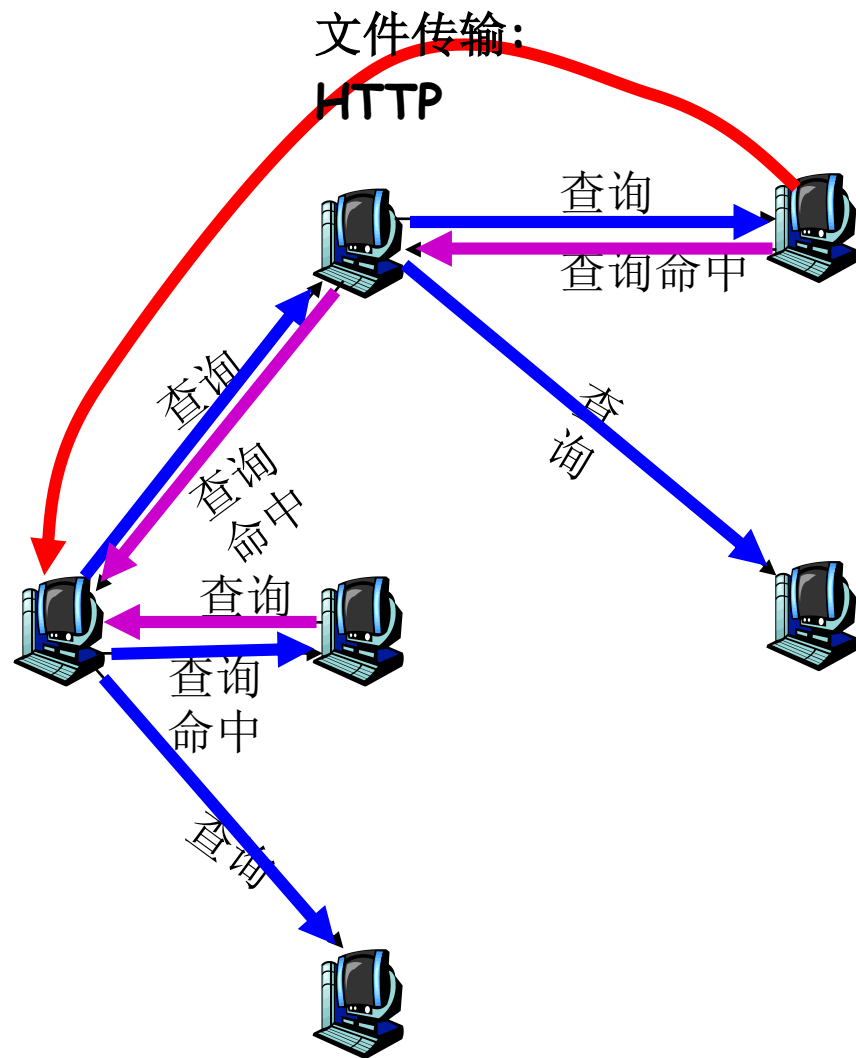
覆盖网络

- ✓ 如果对等方X和Y间**有一条TCP连接**，则存在一条边
- ✓ 所有活动对等方和边形成覆盖网络
- ✓ 边不是物理链路
- ✓ 一个对等方所连接的节点少于10个。



洪泛查询

- ✓向覆盖网络中的每个邻居发送“查询报文”；
- ✓每个邻居再向邻居转发，使覆盖网络上的每个对等方都能收到该查询；
- ✓如果收到查询的对等方中有被请求对象，沿反向路径回发“查询命中”报文；
- 可能收到多个对等方发回的“查询命中”报文
- 选择一个对等方，双方建立一条直接的TCP连接，并通过HTTP报文得到内容。



Gnutella：对等方X加入

- ✓ 对等方X维持一张对等方列表（IP地址）
- ✓ X试图与列表上的对等方建立TCP，直到与Y建立连接
- ✓ X向Y发送Ping报文；Y向邻居转发Ping报文
- ✓ 所有收到Ping报文的对等方Z用Pong报文响应（IP地址）
- ✓ X收到多个Pong报文，建立多个TCP连接，即多个边

Gnutella: 对等方离开

- ❑ 主动离开:离开接点的所有对等方都会刷新自身的**激活对等方列表**,并开始与列表中的新的对等方建立连接
- ❑ 断网:发送信息的时候对等方没有响应,则表明对等方离开,节点刷新自身的**激活对等方列表**,并开始与列表中的新的对等方建立连接

特点

- ✓ 设计简单。
- ✓ “查询报文” 在网络中产生很大的流量。
- ✓ 扩展性差。
- 实际使用中使用限范围的洪泛查询：
 - ✓ 在“查询报文”中设置一个计数字段，并给定一个特定值。
 - ✓ 一定程度上控制查询代价，减少流量。
 - ✓ 不能保证检索到资源

KaZaA

与Gnutella类似，无专用服务器，但对等方地位不平等。

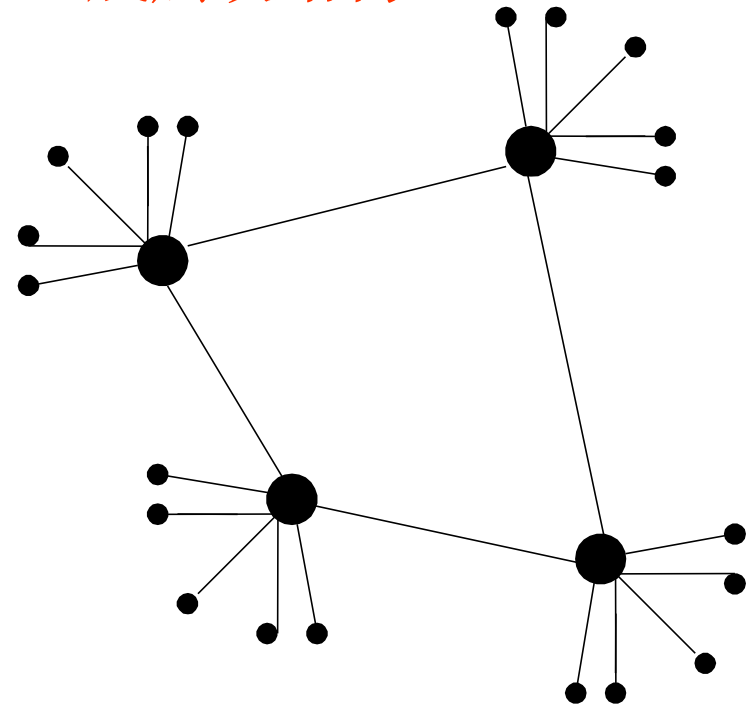
□ 实现：

- ✓ 对等方先形成一个层次的覆盖网络
- ✓ 通过查询，找到所需对象。

层次覆盖网络

对等方根据通信关系划分若干组，组成层次结构。

- 每组包括若干个组员，一个组长。
 - 组员与其组长有一个TCP连接，将共享内容告诉组长。
 - 组长维护一个数据库，该组的共享内容及相关对等方的IP地址。
 - 相关组长之间建立TCP连接
- 组长追踪其所有子节点上的内容



● 普通对等方

● 组长对等方

—— 在覆盖网络中的邻居关系

KaZaA: 查询

对等方确定到某个特定对象的方法：

- ✓ 向组长发出查询，组长用本组中具有该对象的对等方列表响应；
- ✓ 或与其他组长联系，请它们向该对等方发送具有该对象的对等方列表。

KaZaA特点

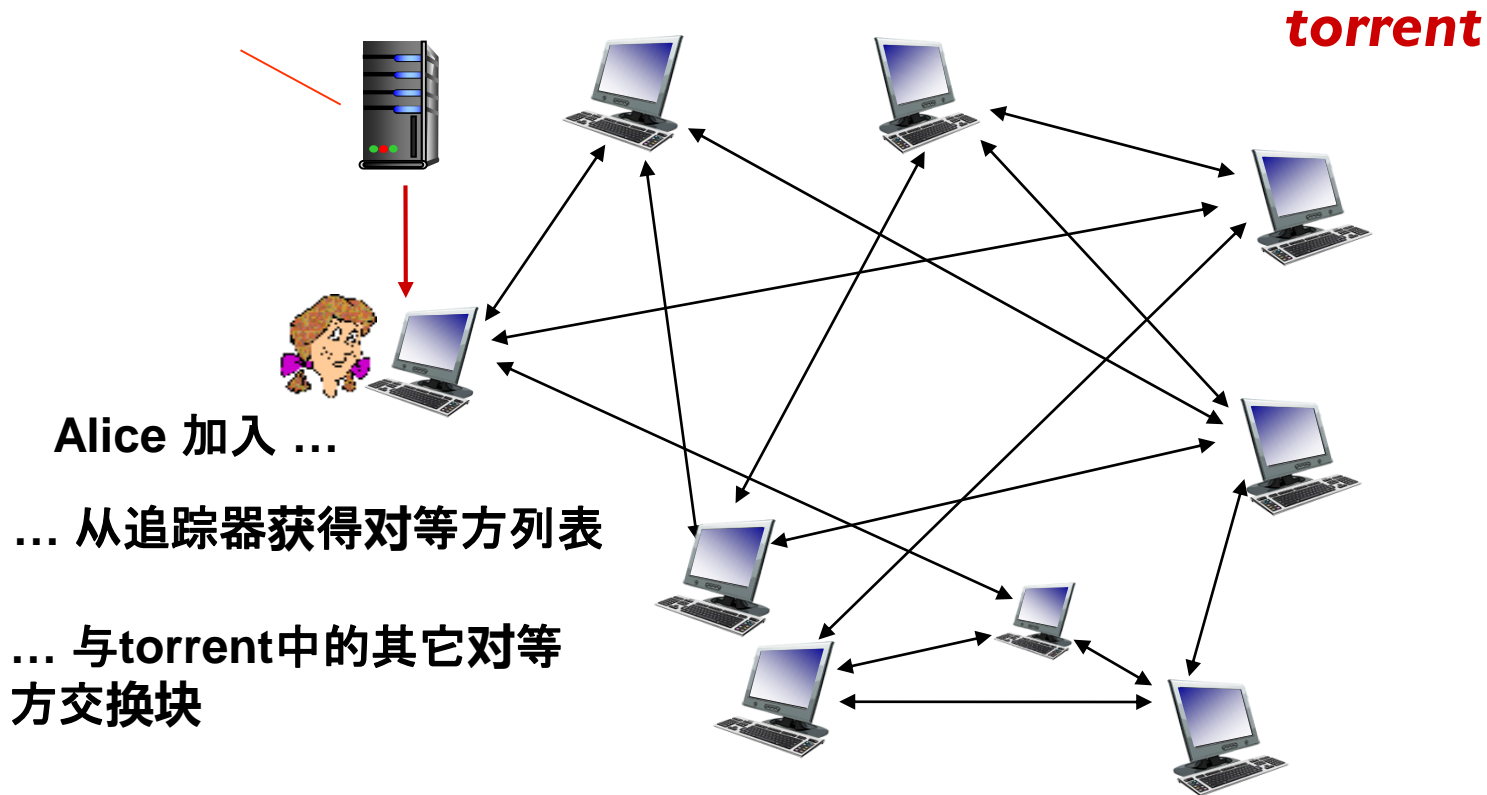
- 层次覆盖网络
- 查询流量不大
- 设计技巧
- ✓ 请求排队：限制并行上载数量
- ✓ 激励优先权：上载文件比下载文件多的用户优先。
- ✓ 并行下载：从多个对等方请求并下载同一个文件的不同部分。

P2P文件分发-BitTorrent

- ❑ BitTorrent是一种用于文件分发的流行P2P协议。
- ❑ 参与一个特定文件分发的所有对等方的集合被称为一个洪流（torrent）。
- ❑ 一个洪流中的对等方彼此下载等长度的文件块（chunk），典型块长度为256KB。

P2P文件分发-BitTorrent

追踪器tracker: 跟踪参与洪流的所有对等方



P2P文件分发-BitTorrent

peer joining torrent:

- has no chunks, but will accumulate them over time from other peers
- registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- ❖ while downloading, peer uploads chunks to other peers
- ❖ peer may change peers with whom it exchanges chunks
- ❖ churn: peers may come and go
- ❖ once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: 请求、发送

请求 文件块(chunks):

- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first(最稀缺优先)

□ 发送文件块: tit-for-tat (一报还一报)

- ❖ Alice sends chunks to those four peers currently sending her chunks *at highest rate*
 - other peers are choked by Alice (do not receive chunks from her)
 - re-evaluate top 4 every 10 secs
- ❖ every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 - newly chosen peer may join top 4

DHT(Distributed Hash Table)

- ❖ DHT: a **distributed P2P database**
- ❖ database has **(key, value)** pairs; examples:
 - key: ss number; value: human name
 - key: movie title; value: IP address
- ❖ Distribute the (key, value) pairs over the (millions of peers)
- ❖ a peer **queries** DHT with key
 - DHT returns values that match the key
- ❖ peers can also **insert** (key, value) pairs

Q: how to assign keys to peers?

- central issue:

- assigning (key, value) pairs to peers.

- basic idea:

- convert each key to an integer
 - Assign integer to each peer
 - put (key,value) pair in the peer that is **closest** to the key

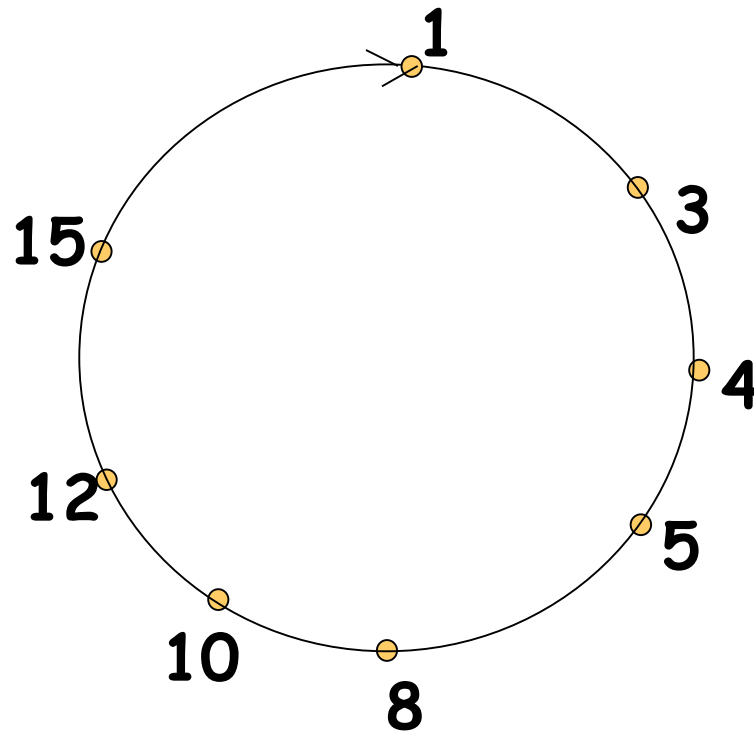
DHT identifiers

- ❑ assign integer identifier to each peer in range $[0, 2^n - 1]$ for some n .
 - each identifier represented by n bits.
- ❑ require each key to be an integer in same range
- ❑ to get integer key, hash original key
 - e.g., key = **hash**("Led Zeppelin IV")
 - this is why its is referred to as a *distributed "hash" table*

Assign keys to peers

- rule: assign key to the peer that has the *closest* ID.
- convention in lecture: closest is the *immediate successor* of the key.
- e.g., $n=4$; peers: 1,3,4,5,8,10,12,14;
 - key = 13, then successor peer = 14
 - key = 15, then successor peer = 1

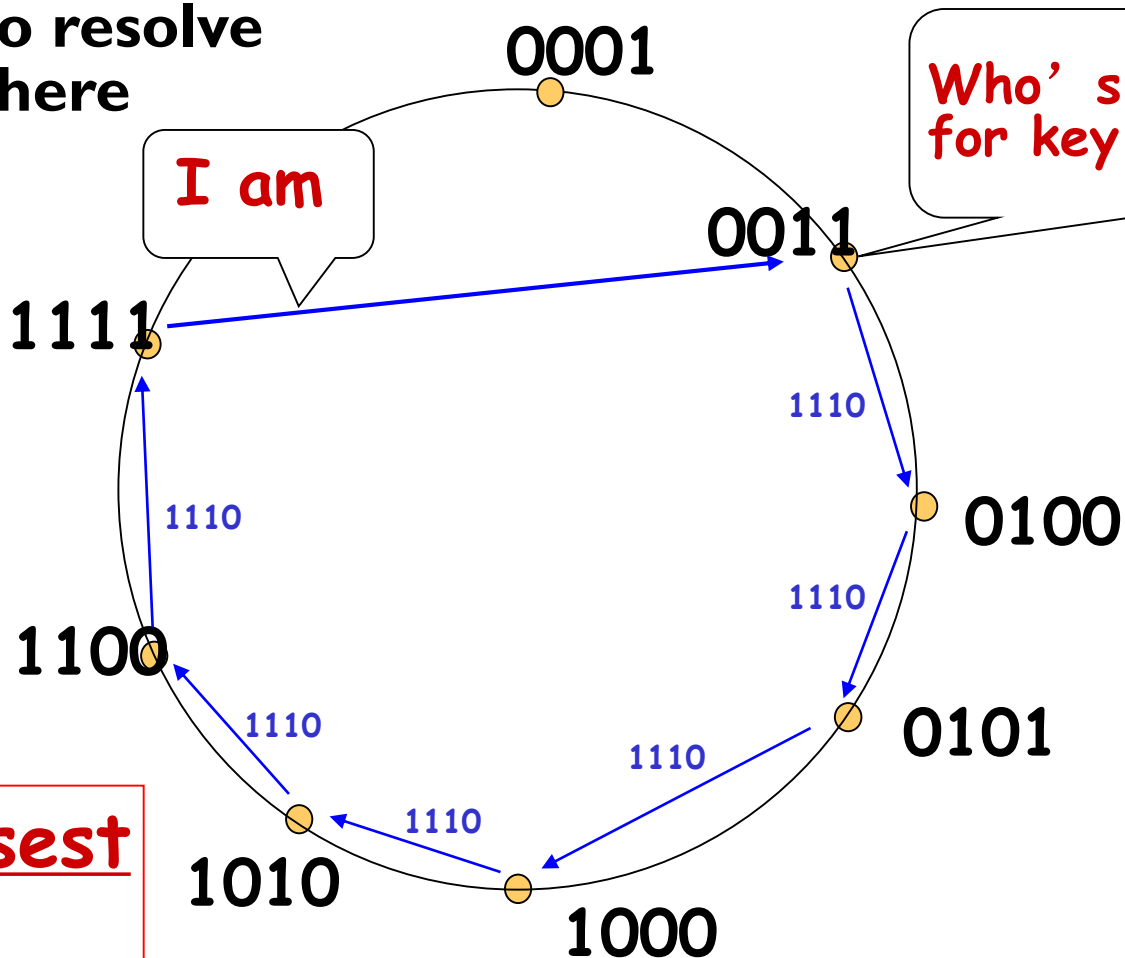
Circular DHT (I)



- ❑ each peer *only* aware of immediate successor and predecessor.
- ❑ “overlay network”

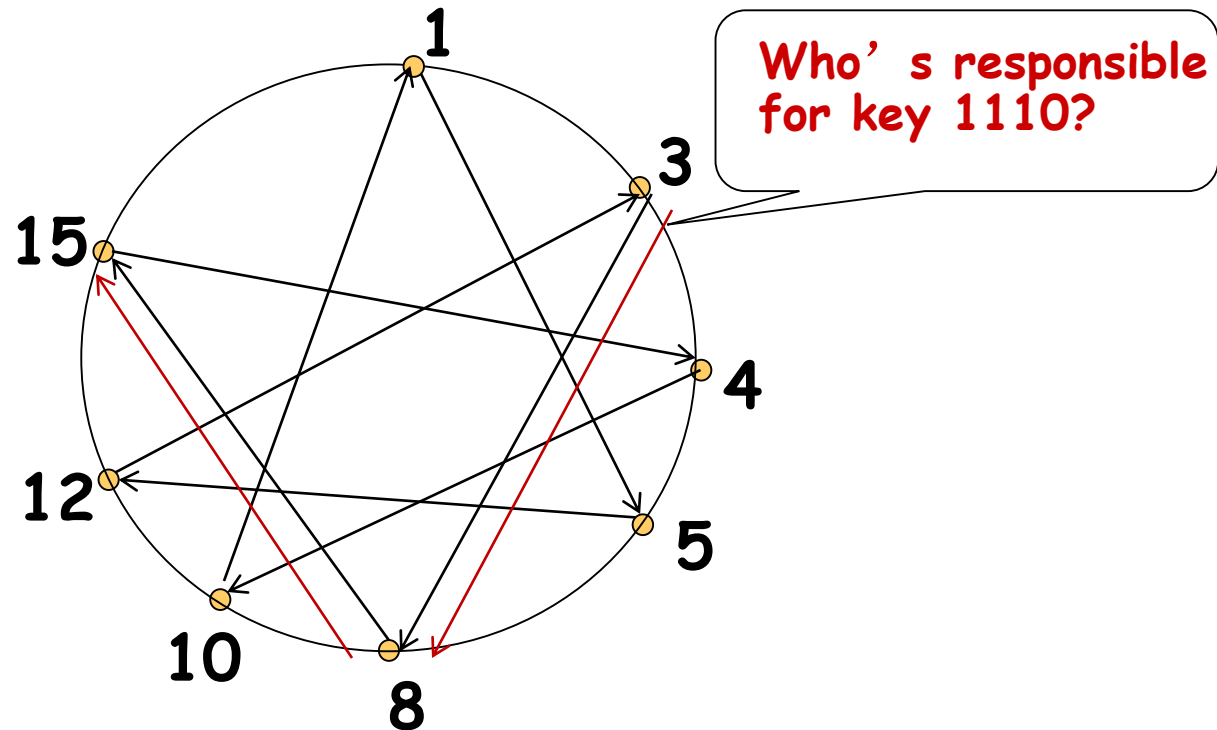
Circular DHT (I)

$O(N)$ messages
on average to resolve
query, when there
are N peers



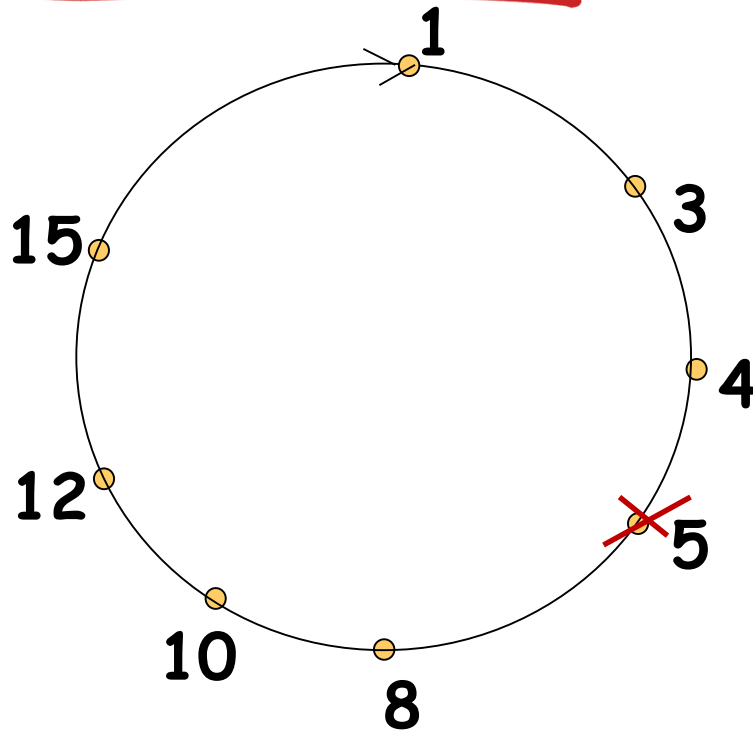
Define closest
as closest
successor

Circular DHT with shortcuts



- ❑ each peer keeps track of IP addresses of predecessor, successor, short cuts.
- ❑ reduced from 6 to 2 messages.
- ❑ possible to design shortcuts so $O(\log N)$ neighbors, $O(\log N)$ messages in query

Peer churn



example: peer 5 ly leaves

□ peer 4 detects peer *abrupt* 5 departure; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

□ what if peer 13 wants to join?

handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

2.7 TCP套接字编程

- ❑ 网络应用程序的核心：客户机程序和服务器程序。

运行时，分别创建一个客户机进程和一个服务器进程，相互之间通过套接字读写数据进行通信。

- ❑ 网络应用程序类型：

- ✓ 通用应用程序：通过RFC文档所定义的标准协议来实现程序必须满足该RFC所规定的规则；使用与协议相关的端口号。如Web应用

- ✓ 专用的应用程序：

程序不必符合RFC规则；开发者根据实际应用设计；不能使用RFC中定义的周知端口号。

Socket API

- ❑ 1981提出于BSD4.1 UNIX操作系统定义了一种 API, 它又称为套接字接口(socket interface)。微软公司在其操作系统中采用了套接字接口 API, 形成了一个稍有不同的 API, 并称之为 Windows Socket。
- ❑ 网络应用程序明确的创建,使用及释放套接字
- ❑ client/server模式
- ❑ 通过Socket API,提供传输层的2类传输服务:
 - 不可靠的数据报传输
 - 可靠的字节流传输

说明

研发初期，先选择运输层协议：

✓ *TCP*:

面向连接的，为两个端系统之间的数据流动提供可靠的字节流通道。

✓ *UDP*:

无连接的，从一个端系统向另一个端系统发送独立的数据分组，不对交付提供任何保证。

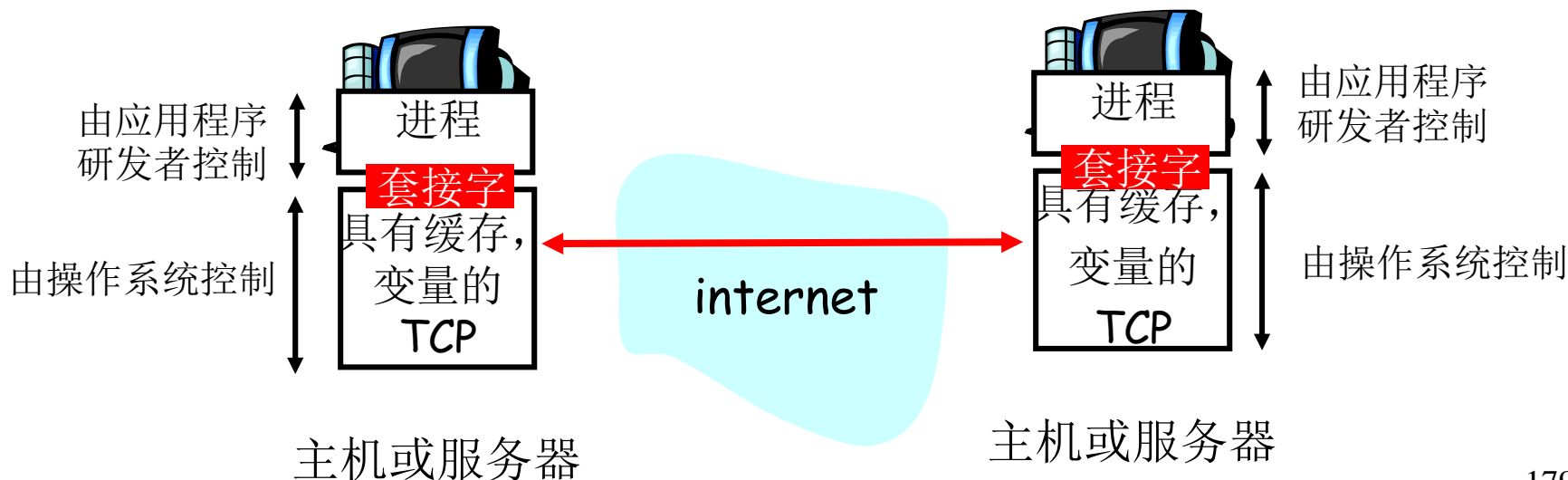
2.7.1 TCP套接字编程

运行在不同机器上的进程彼此通过套接字传递报文来进行通信。

- **进程/套接字**：房子/门户，即套接字是应用进程和TCP之间的门户。

程序开发者可以控制应用层端所有东西；不能控制运输层端。

- **TCP服务**：从一个进程到另一个进程的可靠字节传输



客户机和服务器程序之间的交互

先建立TCP连接，再进行数据传输。

- ✓ 客户机程序是连接的发起方；
- ✓ 服务器必须先准备好，对客户机程序发起的连接做出响应：
 - 服务器程序事先已经在系统中运行；
 - 服务器程序的一个套接字（欢迎套接字）已经打开，准备接收客户机程序发起的连接（敲门）。

具体过程：

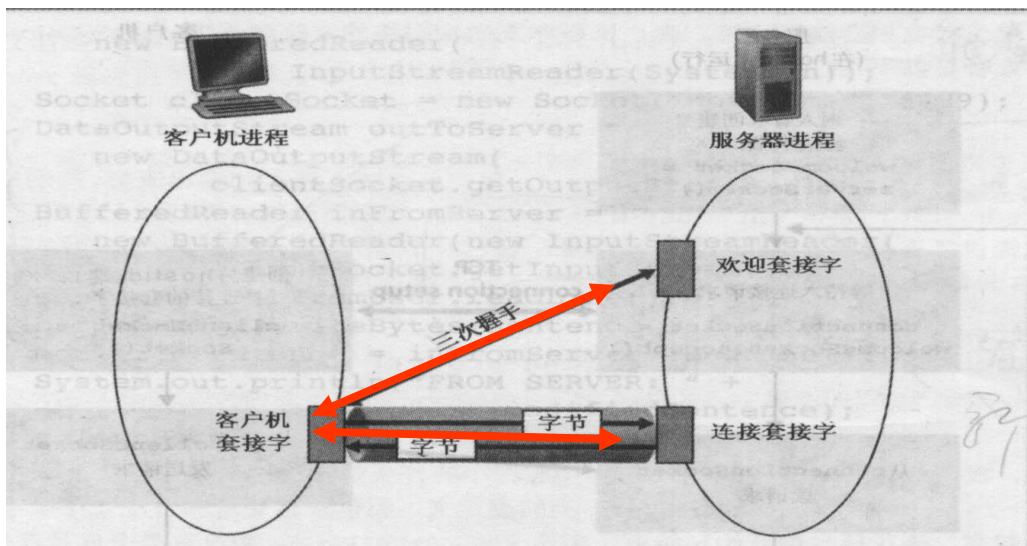
①建立TCP连接

✓ 客户机进程向服务器发起一个TCP连接:

创建一个本地套接字，指定相应服务器进程的地址（IP地址和端口号），发起和服务器的连接请求。开始在“三次握手”的连接过程。

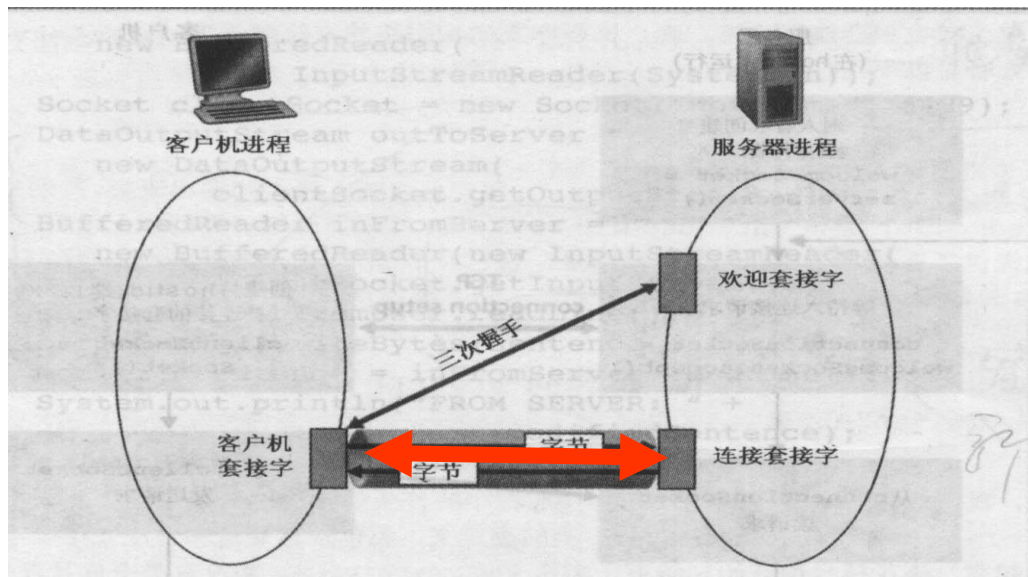
✓ 建立一个TCP连接:

当服务器听到客户机的连接请求（敲门）时，在“三次握手”期间，服务器创建一个新套接字，客户机套接字和服务器连接套接字之间建立一个TCP连接（直接的虚拟管道）。



②传送数据

- ❑ TCP连接为客户机和服务器了一个直接的传输管道。
- ❑ 可靠的, 顺序的, 字节流的传输



术语

- ✓ **流**：流入或流出某进程的一串字符序列。
- ✓ **输入流**：来自某个输入源（如键盘）、或某个套接字（因特网的数据从套接字流入）。
- ✓ **输出流**：到某个输出源（如显示器）、或某个套接字（数据通过套接字流向因特网）。

2.7.2 Java应用程序示例

客户机和服务器经TCP连接进行通信。

- ✓ 客户机从**键盘读**一行字符，通过套接字向服务器发送
- ✓ **服务器**从套接字读取数据；
- ✓ **将该行字符转换成大写**；
- ✓ 将修改的行通过其连接套接字再回发给客户机。
- ✓ **客户机**从其套接字中读取修改的行，并将该行在**显示器上显示**。

客户机/服务器程序交互

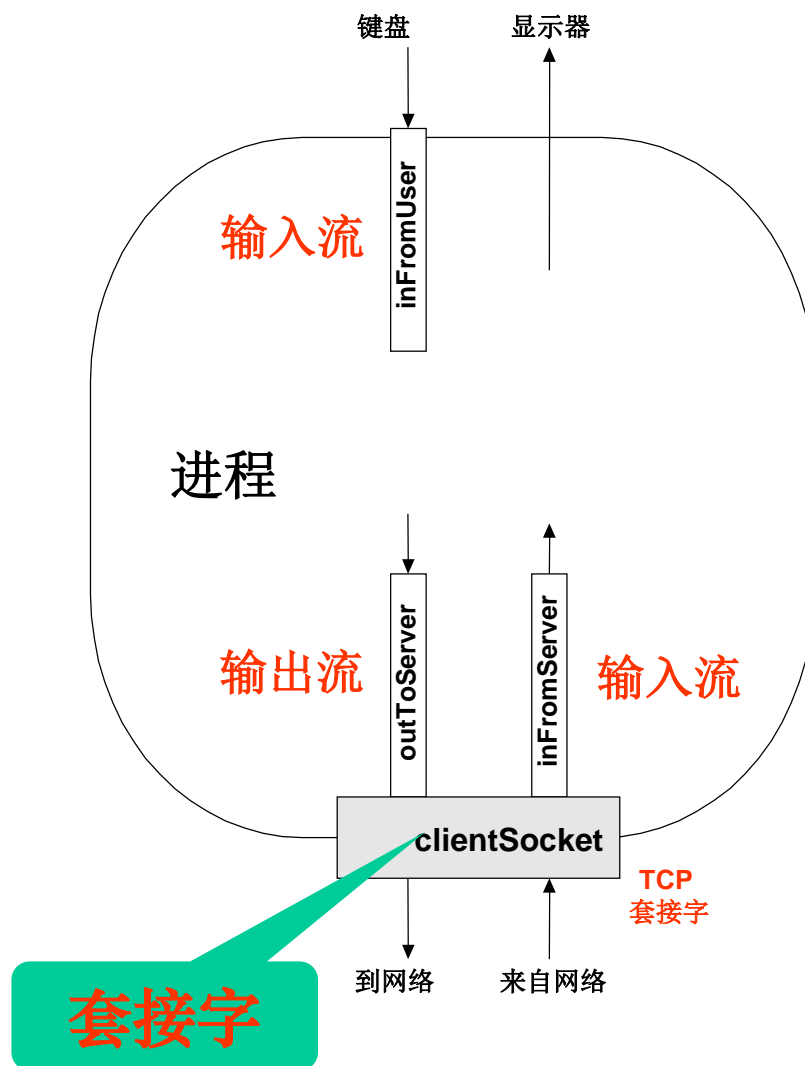
服务器 (运行在 `hostid` 上)

客户机



客户端

- ❑ 创建了三个流和一个套接字，如图所示。
- ✓ **套接字**: `clientSocket` ;
- ✓ ***InFromUser* 输入流**: 连接到键盘;
- ✓ ***InFromServers* 输入流**: 与套接字连接。接收从网络来的字符
- ✓ ***outToServers* 输出流**: 与套接字连接，客户机发送到网络的字符。



客户机 : (TCPClient.java)

产生输入流、输出流、套接字→输入→发送→接收→显示

```
import java.io.*;  
import java.net.*;  
class TCP Client {
```

```
    public static void main(String argv[ ]) throws Exception  
    {
```

```
        String sentence;  
        String modifiedSentence;
```

产生输入流

```
        BufferedReader inFromUser =  
            new BufferedReader(new InputStreamReader(System.in));
```

产生客户机套接字,
与服务器连接

```
        Socket ClientSocket = new Socket("hostname", 6789);
```

生成输出流与
套接字联系

```
        DataOutputStream outToServer =  
            new DataOutputStream(ClientSocket.getOutputStream());
```

产生与套接字
联系的输入流

```
BufferedReader inFromServer =  
new BufferedReader(new  
InputStreamReader(ClientSocket.getInputStream()));
```

```
sentence = inFromUser.readLine();
```

键盘输入

向服务器发送行

```
outToServer.writeBytes(sentence + '\n');
```

从服务器读行

```
modifiedSentence = inFromServer.readLine();
```

```
System.out.println("FROM Server: " + modifiedSentence);
```

```
ClientSocket.close();
```

显示内容

```
}  
}
```

服务器: (TCPServer.java)

产生输入流、输出流、套接字→接收→发送

```
import java.io.*;
import java.net.*;
class TCP Server {
    public static void main(String argv[]) throws Exception
    {
        String ClientSentence;
        String capitalizedSentence;
```

在端口**6789** 生成**欢迎套接字**，监听

→ ServerSocket welcomeSocket = new ServerSocket(6789);

```
while(true) {
```

创建一个连接套接字

→ Socket connectionSocket = welcomeSocket.accept();

生成输入流，与套接字联系

```
BufferedReader inFromClient =
    new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

生成输出流，
与套接字联系

```
DataOutputStream outToClient =  
    new DataOutputStream(connectionSocket.getOutputStream());
```

从套接字读入客
户机来的数据

```
ClientSentence = inFromClient.readLine();
```

```
capitalizedSentence = ClientSentence.toUpperCase() + '\n';
```

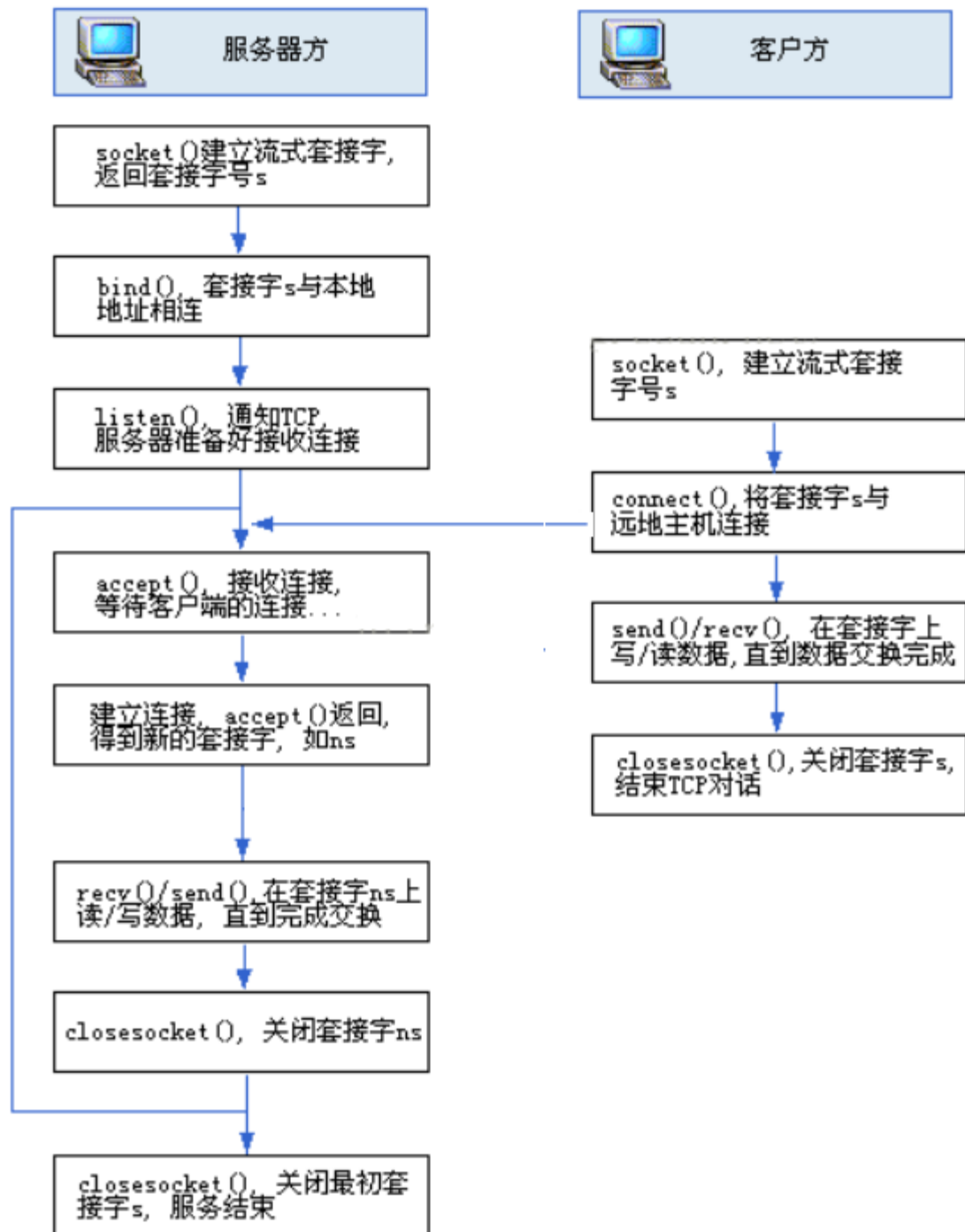
向套接字输出
数据到客户机

```
outToClient.writeBytes(capitalizedSentence);  
connectionSocket.close();
```

```
}  
}  
}
```

循环结束，返回并等待另
一个客户机连接

其他方式



客户机 : (TCPClient.Python)

```
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



服务器：(TCPServer.Python)

Python TCPServer

create TCP welcoming socket	→	<code>from socket import *</code>
		<code>serverPort = 12000</code>
		<code>serverSocket = socket(AF_INET,SOCK_STREAM)</code>
		<code>serverSocket.bind(('',serverPort))</code>
server begins listening for incoming TCP requests	→	<code>serverSocket.listen(1)</code>
loop forever	→	<code>print 'The server is ready to receive'</code>
		<code>while 1:</code>
server waits on accept() for incoming requests, new socket created on return	→	<code>connectionSocket, addr = serverSocket.accept()</code>
read bytes from socket (but not address as in UDP)	→	<code>sentence = connectionSocket.recv(1024)</code>
		<code>capitalizedSentence = sentence.upper()</code>
		<code>connectionSocket.send(capitalizedSentence)</code>
close connection to this client (but <i>not</i> welcoming socket)	→	<code>connectionSocket.close()</code>

2.8 UDP套接字编程

- ✓ UDP是一种无连接的服务，即在两个进程之间没有创建管道时所需的初始握手阶段。
- ✓ 进程之间的数据传递以分组为单位进行。
- ✓ 分组中含目的进程地址（主机IP地址和端口号）。
- ✓ 提供不可靠的传输服务。

编程说明：

- ✓ 通信进程之间没有初始握手，不需要欢迎套接字；
- ✓ 没有流与套接字相联系；
- ✓ 发送主机将信息字节封装生成分组，再发送；
- ✓ 接收进程解封收到的分组，获得信息字节。

客户机/服务器程序交互

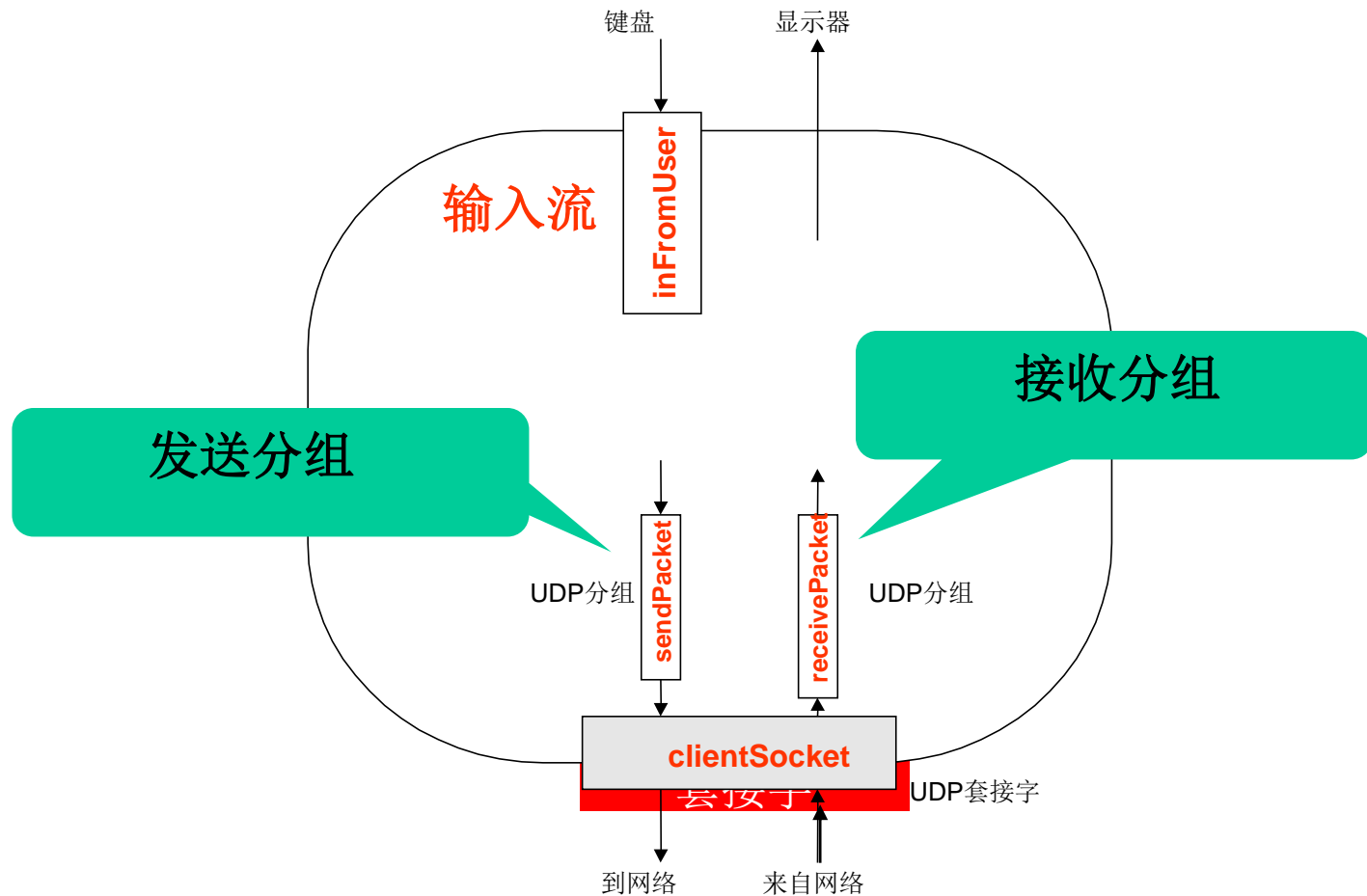
服务器 (运行在hostid上)

客户机



客户机：

- ✓ 一个套接字ClientSocket：发送和接收分组。
- ✓ 一个输入流InfromUser：与标准输入联系（键盘）。



客户机 (UDPClient.java)

产生输入流、套接字→输入→封装→发送

```
import java.io.*;
import java.net.*;
class UDPSocket {
    public static void main(String args[]) throws Exception
```

生成输入流 → `BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));`

生成客户机套接字 → `DatagramSocket ClientSocket = new DatagramSocket();`

使用DNS将主机名转换为IP地址 → `InetAddress IPAddress = InetAddress.getByName("hostname");`

输入

```
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
```

```
String sentence = inFromUser.readLine();
sendData = sentence.getBytes();
```

使用要发送的数据、
长度、**IP**地址、端口
生成数据报

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
                        IPAddress, 9876);
```

向服务器
发送数据报

```
ClientSocket.send(sendPacket);
```

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);
```

从服务器读数据报

```
ClientSocket.receive(receivePacket);
```

```
String modifiedSentence =  
    new String(receivePacket.getData());
```

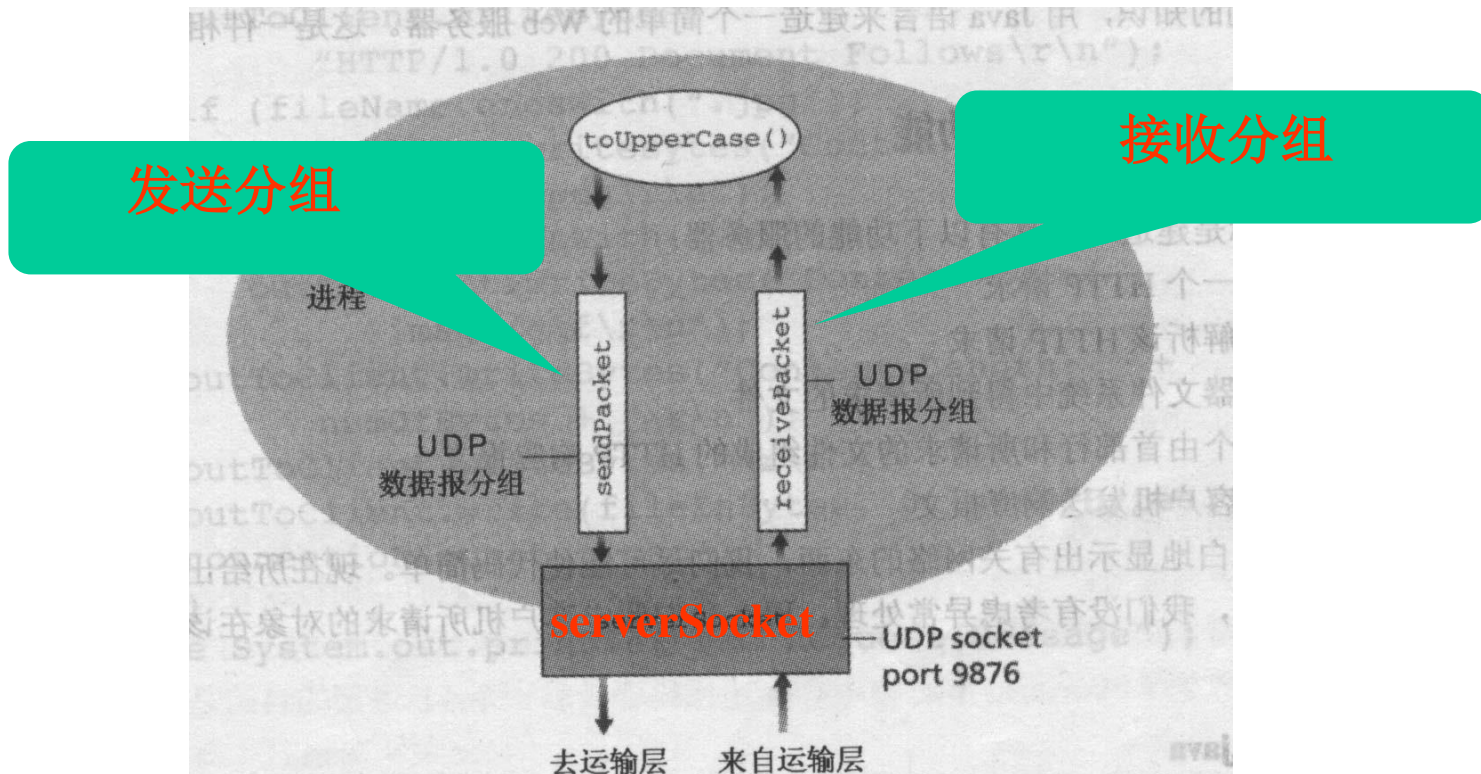
显示

```
System.out.println("FROM Server:" + modifiedSentence);  
ServerSocket.close();  
}
```

```
}
```

服务器：

- 一个套接字serverSocket：发送和接收分组。



服务器(UDPServer.java)

产生套接字→接收→解封→发送

```
import java.io.*;  
import java.net.*;
```

```
class UDPServer {  
    public static void main(String args[]) throws Exception
```

在端口**9876**生成
数据报套接字

```
{  
    DatagramSocket ServerSocket = new DatagramSocket(9876);
```

```
    byte[] receiveData = new byte[1024];  
    byte[] sendData = new byte[1024];
```

```
    while(true)
```

为接收的数据报
生成空间

```
    {  
        DatagramPacket receivePacket =  
            new DatagramPacket(receiveData, receiveData.length);
```

接收数据报

```
        ServerSocket.receive(receivePacket);
```

```
String sentence = new String(receivePacket.getData());
```

获得发送方的 IP
地址，端口号

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();
```

产生数据报发
送给客户机

```
sendData = capitalizedSentence.getBytes();
```

向套接字写
出数据报

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
        IPAddress, port);
```

```
ServerSocket.send(sendPacket);
```

```
}  
}  
}
```

循环结束，返回等待另
一个数据报

客户机 (UDPClient.Python)

include Python's socket library

from socket import *
serverName = 'hostname'
serverPort = 12000

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))
modifiedMessage, serverAddress =

read reply characters from socket into string

clientSocket.recvfrom(2048)

print modifiedMessage

print out received string and close socket

clientSocket.close()

服务器 (UDPServer.Python)

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
into message, getting
client's address (client IP
and port) →

```
message, clientAddress = serverSocket.recvfrom(2048)
```



```
modifiedMessage = message.upper()
```

send upper case string
back to this client →

```
serverSocket.sendto(modifiedMessage, clientAddress)
```

思考题：

1. 简述应用程序体系结构三种类型的特点。
2. 什么是套接字、用户代理和Web缓存？简述用户进程和套接字的关系。
3. 什么是P2P文件共享？简述其内容定位三种方式的特点。

作业要求：设计一个WEB服务器

- ❑ 详细设计文档
- ❑ 源码
- ❑ 可执行代码(包括测试用的简单网页，解压到指定目录(D盘)后，就可以直接运行。在浏览器用<http://127.0.0.1:6699>/index.html就可以直接访问测试)
- ❑ 打包提交，文件名为：
学号-姓名-第一次作业
作业提交地址：
1203035937@qq.com

服务器基本流程：

- ❑ 创建套接字(使用端口号：6699)
- ❑ 获取HTTP请求，并解析HTTP请求报文
- ❑ 显示请求报文各字段的字段名和值，并进行说明
- ❑ 根据HTTP请求报文获得对象文件路径名
- ❑ 根据路径名打开本地文件
- ❑ 封装本地文件到HTTP响应报文
- ❑ 使用套接字发送HTTP相应报文

作业文档要求

提交的**详细设计文档**要求如下(按模板提交):

- ❑ 开发工具和开发平台的说明：选择那些开发工具，具体的版本，包括，如果使用其他的测试工具等，都一起说明
- ❑ 运行环境的说明。说明程序需要运行什么操作系统等环境下。
- ❑ 要包含主要的流程设计。
- ❑ 部分关键的模块的流程设计。
- ❑ 模块划分说明：模块的用途，执行的功能等。
- ❑ 部分关键变量、类型的说明。
- ❑ 部分关键模块的详细设计说明。
- ❑ 可执行文件的安装配置说明，比如需要安装到哪个目录下，需要配置哪些参数才能正确运行等。
- ❑ 只把正个程序拷贝到说明文档里，是不合格的。

小结

- 应用程序体系结构
 - 客户机/服务器
 - P2P
 - 混合
- 应用服务器需要的服务
 - 可靠，带宽，延时
- 网络运输层协议
 - 面向连接，可靠: TCP
 - 不可靠，数据报: UDP

特殊协议:

○ HTTP

○ FTP

○ SMTP, POP, IMAP

○ DNS

套接字编程

小结

重点: 掌握应用层协议

□ 典型的请求/应答消息交换:

- 客户请求信息或服务
- 服务器以数据或状态码应答

□ 消息格式:

- 头部
- 数据

控制报文 **vs.** 数据报文

○ 带内, 带外

集中 **vs.** 分散

可靠 **vs.** 不可靠报文传输
“网络边缘的复杂性”

第二章：复习大纲

- ❑ 网络应用程序体系结构
- ❑ Web应用和HTTP协议
 - 基本术语（网页、URL等）
 - HTTP的特性及其区别(无状态、非持久和持久等)
 - 请求和响应报文
 - COOKIE技术
 - Web缓存
- ❑ 文件传输协议FTP：两种连接
- ❑ 电子邮件：组成及其使用的协议
- ❑ DNS的功能和实现
- ❑ P2P文件共享原理和实现技术
- ❑ TCP、UDP套接字编程的C/S基本流程