

计算机网络

第2章 应用层

目 录

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

2.1 应用层协议原理

■ 常见的网络应用

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

2.1 应用层协议原理

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

那么现在的你应该做些什么呢？

2.1 应用层协议原理

■ 知道什么是网络应用程序

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

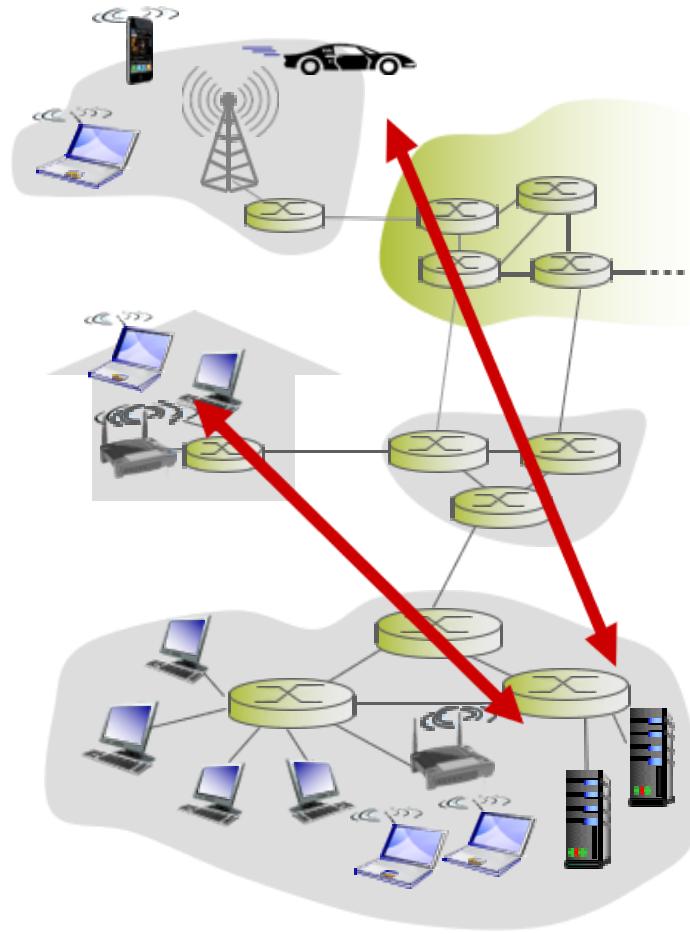
2.1 应用层协议原理

- 决定你的网络应用所需采用的体系结构
 - 客户机/服务器体系结构 (C/S)
 - P2P体系结构
 - 混合体系结构

2.1 应用层协议原理

■ 客户机/服务器体系结构

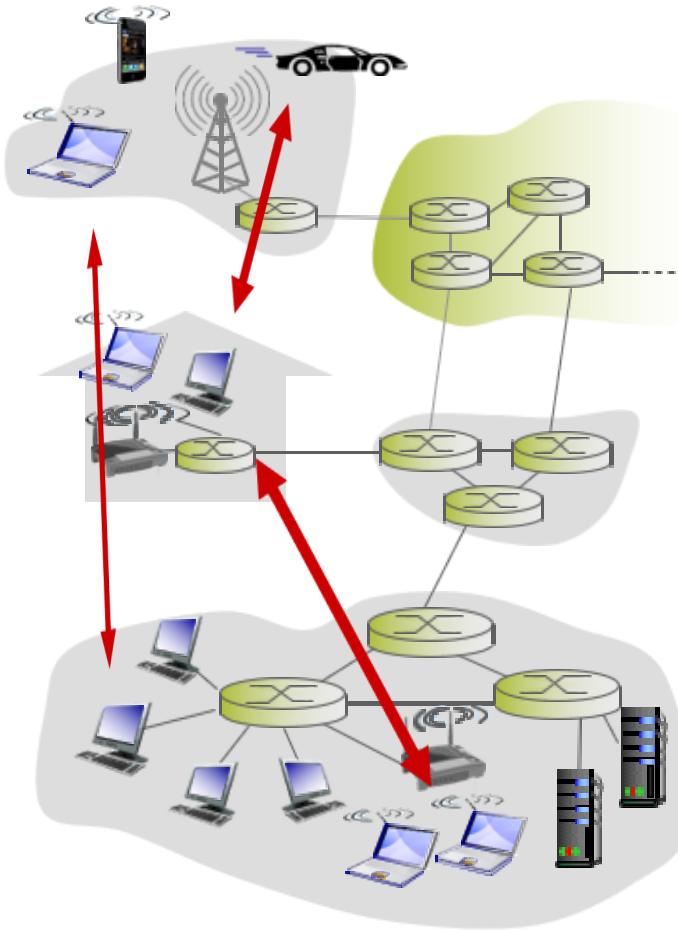
- 存在一个能够向客户机提供服务的服务器, e.g., WEB服务器
- 存在一个或者多个主动连接服务器, 试图从服务器那里获取所需服务的客户机, e.g., IE浏览器
- 特别注意1: 客户机之间不能互相通信
- 特别注意2: 为提高服务器的处理能力, 通常采用服务器群集 (Server Farm)



2.1 应用层协议原理

■ P2P体系结构

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



2.1 应用层协议原理

■ 混合体系结构

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

2.1 应用层协议原理

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

2.1 应用层协议原理

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

应用层协议 ≠ 网络应用

2.1 应用层协议原理

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

2.1 应用层协议原理

■ 常见应用程序对传输服务的要求

<u>应用程序</u>	<u>数据丢失</u>	<u>带宽</u>	<u>实时性</u>
文件传输	不能丢失	弹性	无
e-mail	不能丢失	弹性	无
Web 网页	不能丢失	弹性: 几 kb/s	无
实时音频/视频	允许丢失	音频: 几 k-1Mb/s	100' s msec
	允许丢失	视频: 10k-5Mb/s	
存储音频/视频	允许丢失	同上	几秒
交互式游戏	允许丢失	几 kb/s 以上	100' ms
智能手机讯息	不能丢失	弹性	yes and no

2.1 应用层协议原理

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

□ TCP

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

□ UDP

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

2.1 应用层协议原理

■ 因特网常见应用采用的传输协议

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

2.1 应用层协议原理

■ 安全性

- TCP/UDP天生不具备安全性

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

- 安全套接字层SSL

- 提供加密的TCP连接
 - 数据的完整性检查
 - 端点身份鉴别

- SSL位于应用层与TCP之间（详见第7章）

2.1 应用层协议原理

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

2.1 应用层协议原理

■ 一个例子

□ 你们整栋宿舍有一个信箱，栋长每天都会查看一次信箱，取走新的信件和报纸，当你有信件需要寄送时，直接投递到邮局的邮筒里

□ 假设

- 邮递员仅把邮件送到每栋宿舍唯一的信箱，并不负责投递到个人
- 栋长取出新的信件和报纸后，负责将信件和报纸投递到具体的房间

□ 问题

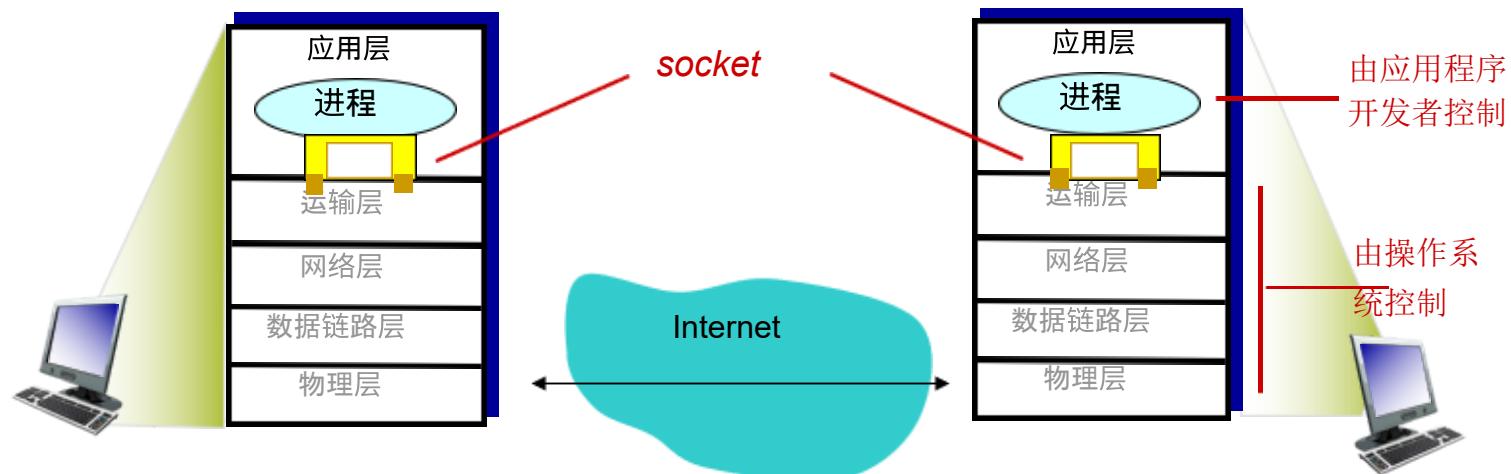
- 栋长如何区分哪封信件属于哪个房间呢？

2.1 应用层协议原理

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

2.1 应用层协议原理

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



2.1 应用层协议原理

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

2.1 应用层协议原理

■ 本章重点讨论的网络应用

- WEB
- 文件传输
- 电子邮件
- 目录服务
- P2P

2.2 WEB应用和HTTP协议

■ 历史的回顾

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

■ 广播收音机/电视和WEB的异同点

- 都是广播和按需操作
- 你不能发布电视节目，但可以发布WEB内容

2.2 WEB应用和HTTP协议

■ WEB的构成

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

特别说明：WEB属于C/S模式

2.2 WEB应用和HTTP协议

■ WEB内容的表达

- Web 页面由一些对象组成。
- 对象可以是HTML文件、JPEG图片、音频文件、Java Applet……
- HTML文件是Web页面的基础，它可以包括各种各样的对象，是一个容器对象
- 任何一个对象都可以用 URL 来定位
- URL的例子：

www.hust.edu.cn/cs/pic.gif

主机名

路径名

2.2 WEB应用和HTTP协议

WEB内容的传输——

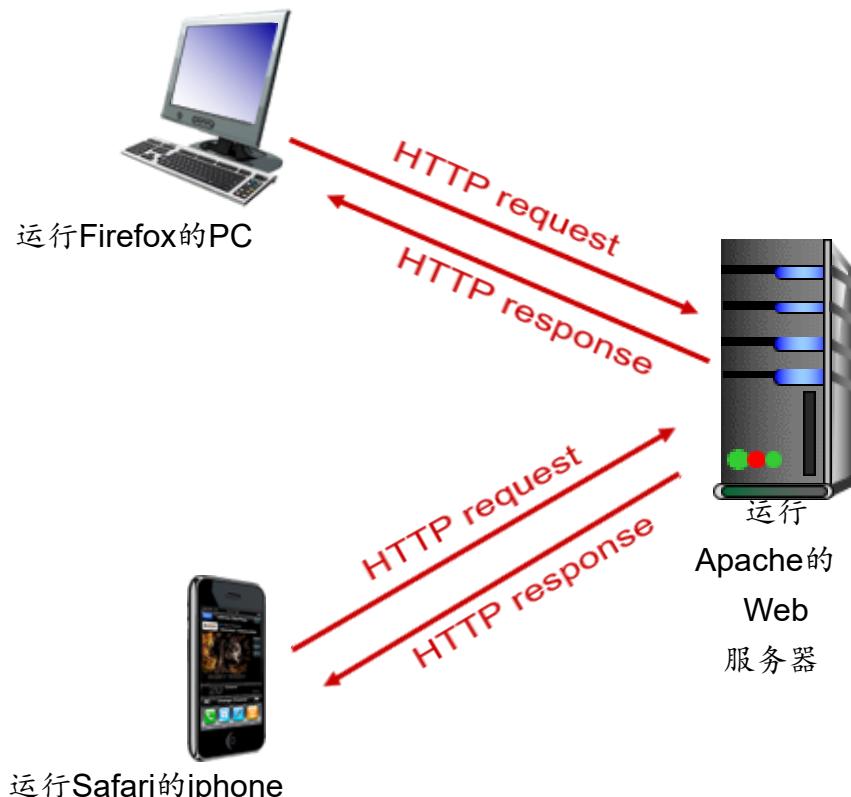
HTTP协议

- 客户端/服务器模式

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

- http1.0: RFC 1945

- http1.1: RFC 2068



2.2 WEB应用和HTTP协议

http: TCP 传输服务:

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

http 是 “无状态”的

- 服务器不保留任何访问过的请求信息

小评论

保留状态的协议很复杂哟!

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

2.2 WEB应用和HTTP协议

■ HTTP1.0的传输模式——非持久性连接

假设用户键入了一个 URL `www.cug.edu.cn/index.htm`

(假设该网页包含文本并引用了10张jpeg图片)

1a. http 客户端启动 TCP 连接

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

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

2. http 客户端发送 http 请求报文 (包括 URL) 进入

TCP 连接插口 (socket)

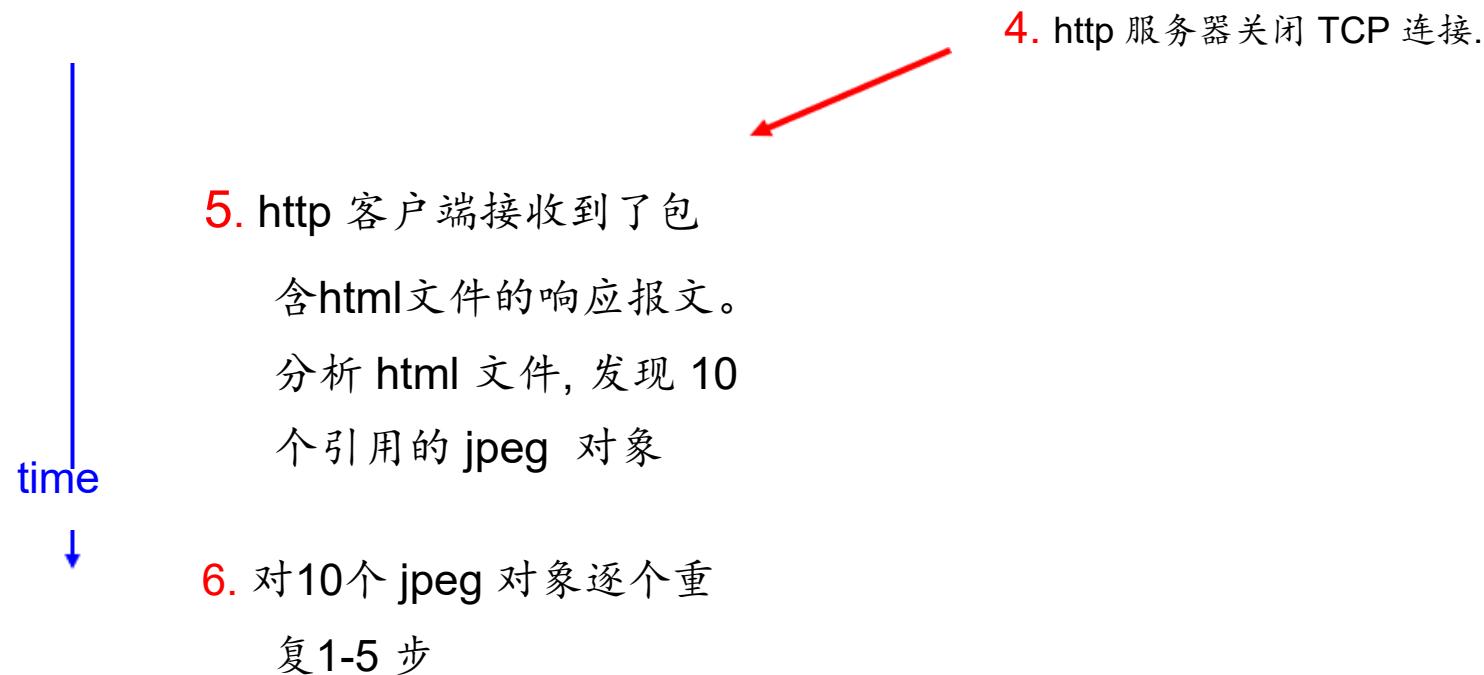
3. http 服务器接收到请求报文, 形成 响应报文 (包含了所请求的对象, `index.htm`), 将报文送入插口 (socket)

time

2000年9月21日

下续
28

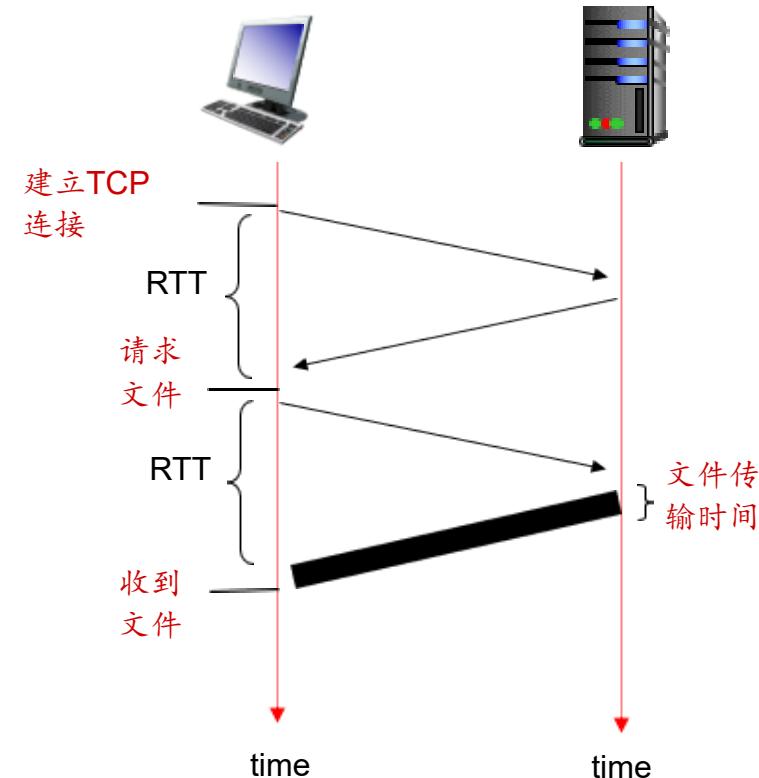
2.2 WEB应用和HTTP协议



2.2 WEB应用和HTTP协议

■ 非持久性连接工作机制

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



2.2 WEB应用和HTTP协议

■ HTTP1.1引入的新传输模式——持久连接

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

2.2 WEB应用和HTTP协议

■ HTTP报文类型

- HTTP请求报文
- HTTP响应报文

2.2 WEB应用和HTTP协议

■ HTTP请求报文

□ 一段典型的HTTP请求报文 (ASCII)

请求行
(GET, POST,
HEAD 命令)

首部
诸行

单独一行回车、换行
表示报文头部结束

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

回车符
换行符

2.2 WEB应用和HTTP协议

□ HTTP请求报文的一般格式



2.2 WEB应用和HTTP协议

■ 请求行支持的方法

□ HTTP1.0 定义的方法

■ GET

- 向服务器请求指定URL的对象

■ POST

- 用于向服务器提交表单数据
- 也可以同时请求一个WEB页面
- 特别注意：可以不使用POST方法，而使用GET方法发送表单数据以获取新的WEB页面。e.g. 搜索引擎

■ HEAD

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

 下续

2.2 WEB应用和HTTP协议

- HTTP1.1新定义的方法

- PUT

- 上传的文件放在实体主体字段中，目标路径由URL字段标明

- DELETE

- 删除URL字段中指定的文件

- 另一种上传数据的方式

- 使用GET方法

- 将需要上传的数据放到URL中

www.somesite.com/animalsearch?monkeys&banana

2.2 WEB应用和HTTP协议

■ HTTP响应报文

状态行

(协议状态码)

状态短语

首部

诸行

数据, e.g.,

被请求的html文件

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
1\r\n
\r\n
data data data data data ...
```

2.2 WEB应用和HTTP协议

□ HTTP响应报文的一般格式



2.2 WEB应用和HTTP协议

■ 常见的HTTP响应状态码和短语

200 OK

- 请求成功，被请求的对象在报文中

301 Moved Permanently

- 被请求的对象被移动过，新的位置在报文中有说明

(Location:)

400 Bad Request

- 服务器不懂请求报文

404 Not Found

- 服务器上找不到请求的对象

505 HTTP Version Not Supported

- 服务器不支持请求报文使用的HTTP协议版本

2.2 WEB应用和HTTP协议

- 了解HTTP报文格式的最好方法就是自行测试

1. 用Telnet 连接测试用的服务器:

telnet cis.poly.edu 80

打开 TCP 连接到 port 80

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

2. 键入一条 http请求报文:

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

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

3. 请注意观察http服务器发回的响应报文!

(或者使用Wireshark观察捕获的HTTP请求/响应报文)

2.2 WEB应用和HTTP协议

■ 用户-服务器交互：Cookie

□ WEB站点使用Cookie的目的

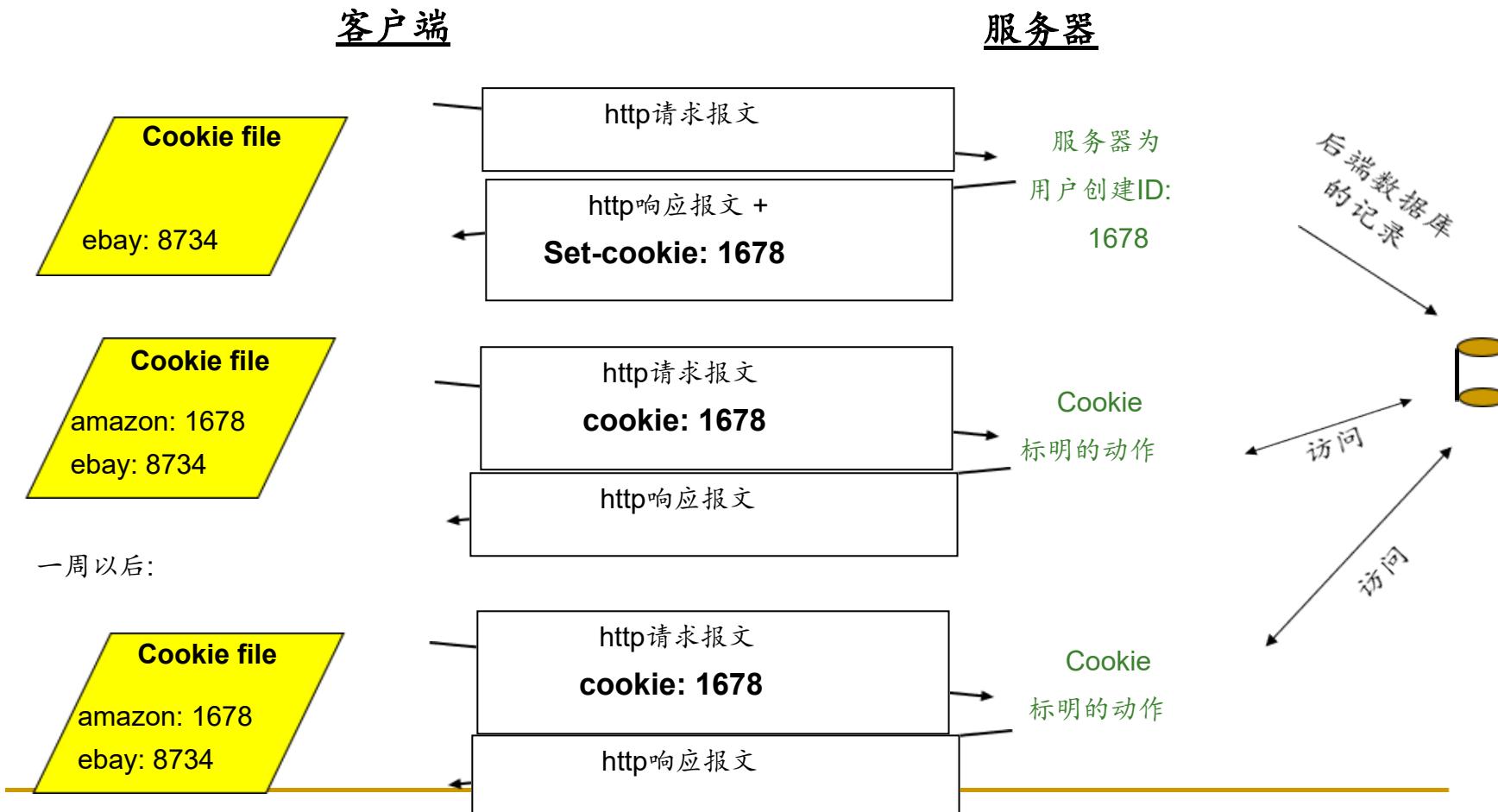
- 限制用户的访问
- 把内容和用户身份关联起来

□ Cookie技术的组成部分：

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

2.2 WEB应用和HTTP协议

□ Cookie工作流程



2020年9月21日

42

2.2 WEB应用和HTTP协议

❑ Cookies能为我们带来什么好处呢？

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

Cookies和私密性：

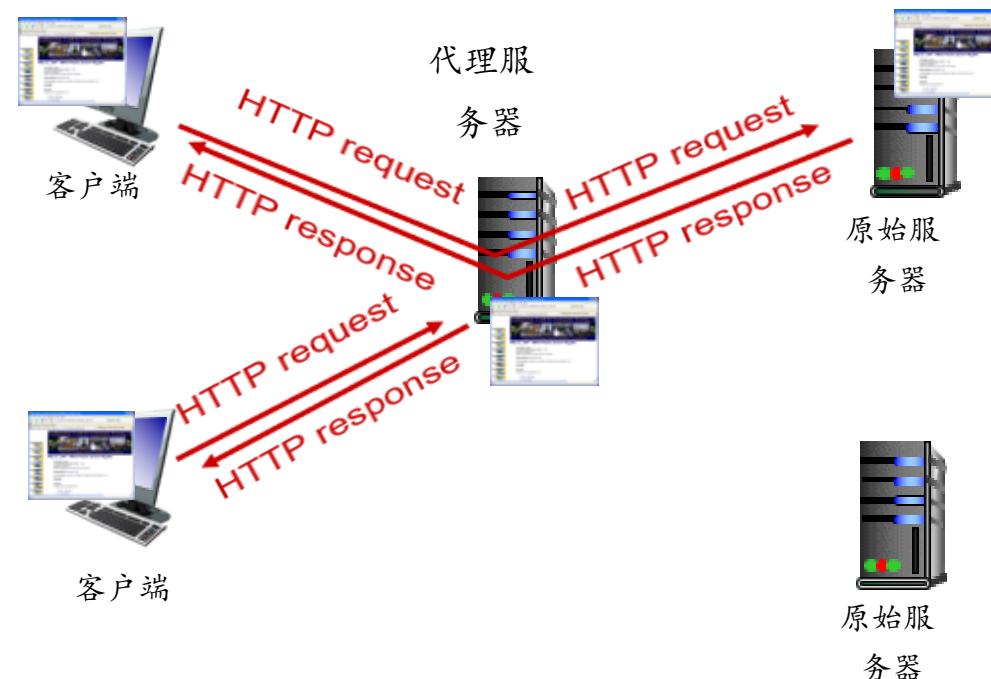
- Cookies允许网站获得相当多的用户的信息
- 你可能会向网站提供你的姓名和E_Mail地址
- 搜索引擎也可以使用cookie和重定向技术获得很多的信息
- 广告公司也可以通过用户访问过的网站来获得用户的相关信息

2.2 WEB应用和HTTP协议

■ WEB缓存

□ 目的

- 加速客户端访问WEB页面的速度，减少时延
- 减少局域网与外部因特网交换的数据量，从而在达到同等服务质量的同时，可以使用户较小的网络带宽，节约费用

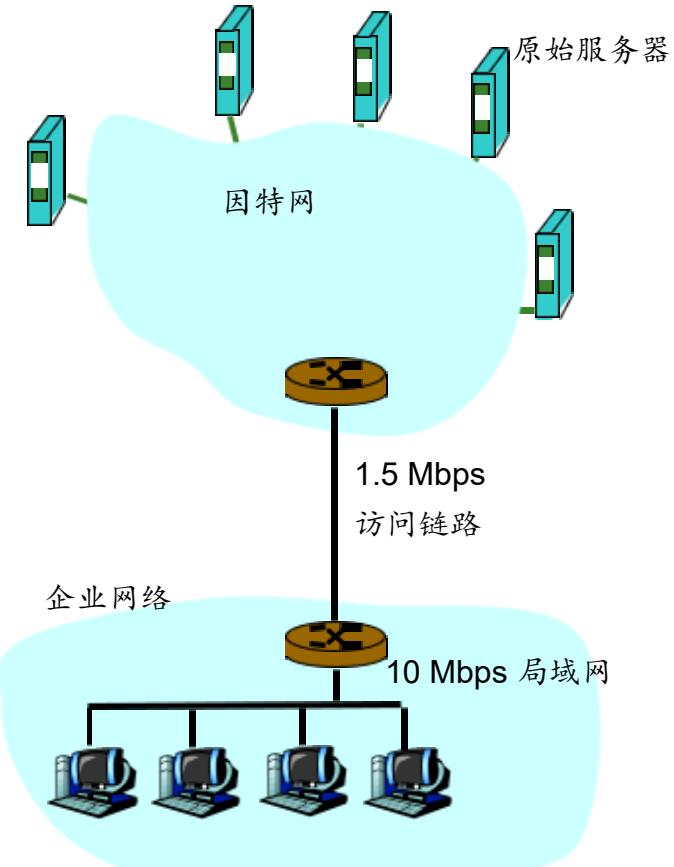


2.2 WEB应用和HTTP协议

■ 缓存举例

□ 假设

- 平均对象的大小 = 100kb
- 浏览器对这些对象的平均访问速率为 15 个/秒
- 从因特网一侧的路由器转发 HTTP 请求到它收到响应报文的平均时间为 2 秒

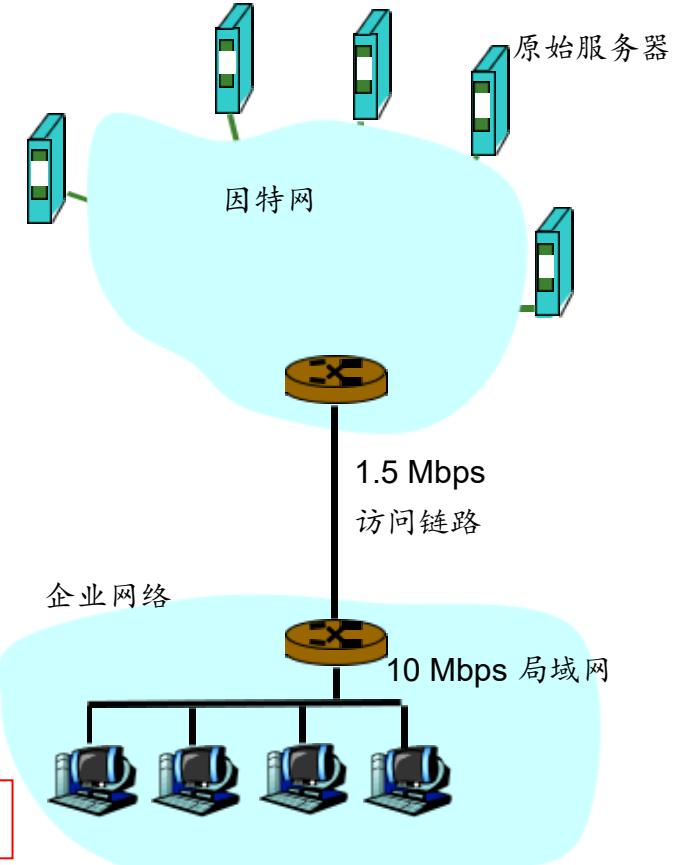


2.2 WEB应用和HTTP协议

□ 结果

- 总延迟 = Internet延迟 +
访问延迟 + 局域网延迟
- 局域网的流量强度 = 0.15
- 接入链路的流量强度 = 1
- 当流量强度为1时，时延可能非常大，从而导致总时延可能得以分钟计了

流量强度 = $L_a/R = 15 \text{个请求/秒} \times 100\text{kb/请求}/R$



2.2 WEB应用和HTTP协议

■ 改进方案1——增加出口带宽

□ 假设

■ 增加到10Mbps

□ 结果

■ 局域网的流量强度 =0.15

■ 接入链路的流量强度 =0.15

■ 总延迟 = Internet延迟 + 访问延迟 + 局域网延迟

$\approx 2 \text{ sec}$

请注意：增加出口带宽的费用通常是非常昂贵的

2.2 WEB应用和HTTP协议

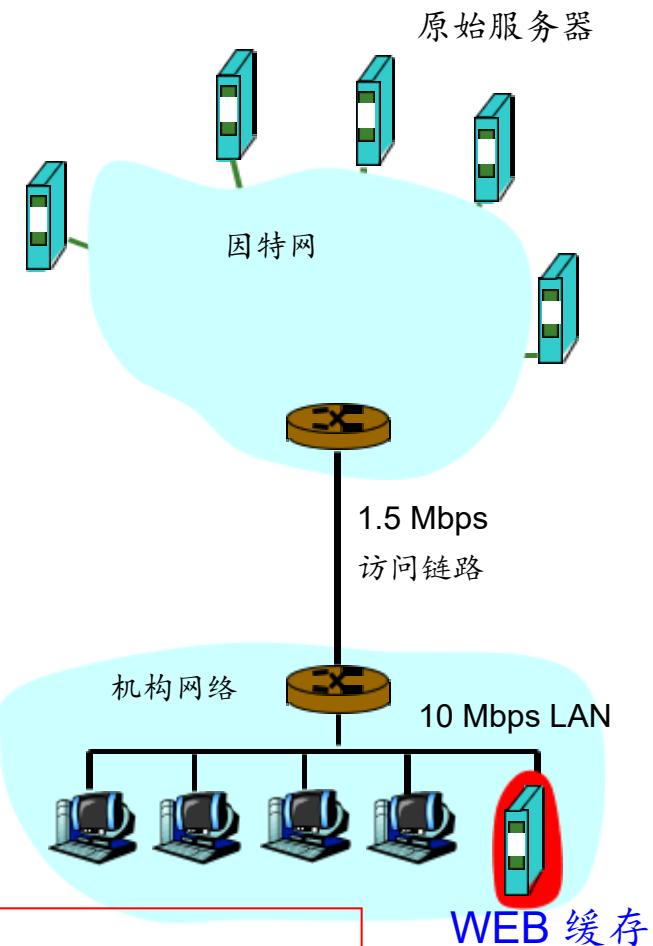
■ 改进方案2 —— 架设WEB缓存

□ 假设命中率为0.4

□ 结果

- 40% 的请求几乎可以马上得到响应
- 60% 的请求必须从服务器上获得响应
- 接入链路的流量强度减少到0.6, 其导致的延迟可以忽略 (例如10msec) 。
- 总的平均延迟 = Internet 延迟 + 访问延迟 + 局域网延迟
 $\approx 0.6 \times (2.01) \text{ 秒} + 0.4 \times (0.01\text{s})$
 略大于1.2 secs, 好于方案1

代价：一台普通PC+一套免费的WEB缓存软件



2.2 WEB应用和HTTP协议

■ 条件GET方法的使用

- 目的：更新WEB缓存中的WEB对象副本
- 举例
 - WEB缓存向WEB服务器发送请求报文

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

2.2 WEB应用和HTTP协议

- WEB服务器向该WEB缓存发送响应报文

HTTP/1.1 200 OK

Date: Mon, 7 Jul 2003 15:39:29

Server: Apache/1.3.0(Unix)

Last-Modified: Wed, 2 Jul 2003 09:23:24

Content - Type: image/gif

(data data data data data data)

2.2 WEB应用和HTTP协议

- 一周后，一个客户端请求该对象，为判断WEB缓存中的该对象副本是否最新的，该WEB缓存向原始服务器发出一个条件GET方法，执行更新检查

```
GET /fruit/kiwi.gif HTTP/1.1  
Host: www.exotiquecuisine.com  
If-Modified-Since: Wed, 2 Jul 2003 09:23:24
```

2.2 WEB应用和HTTP协议

- 如果该对象没有被修改过，WEB缓存上的仍然是最新版本，则WEB服务器发送如下响应报文

HTTP/1.1 304 Not Modified

Date: Mon, 14 Jul 2003 15:39:29

Server: Apache/1.3.0(Unix)

(实体主体为空)

2.2 WEB应用和HTTP协议

- 如果该对象在此之后被修改过，WEB服务器上有最新版本，则WEB服务器发送新版本的对象给WEB缓存

HTTP/1.1 200 OK

Date: Mon, 14 Jul 203 15:39:29

Server: Apache/1.3.0(Unix)

Last-Modified: Sat, 12 Jul 2003 09:23:24

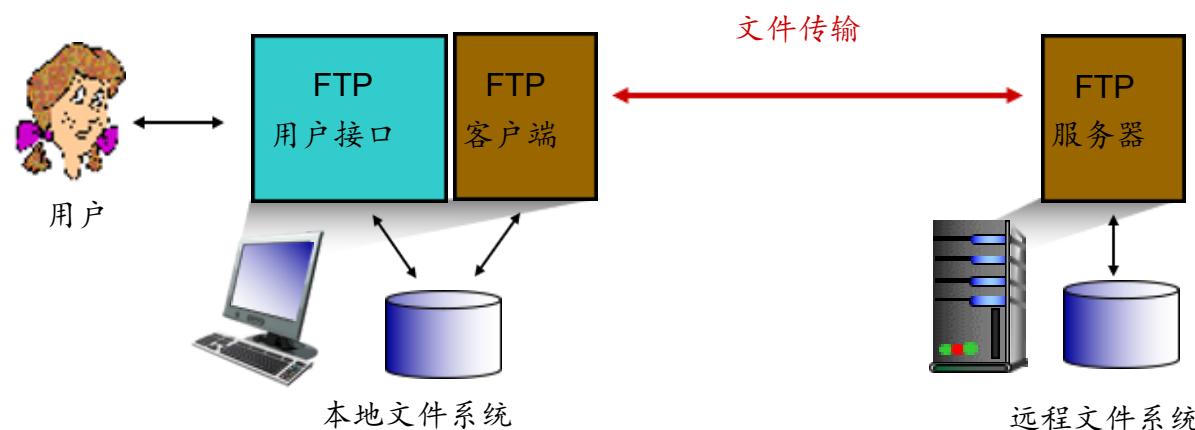
Content - Type: image/gif

(data data data data data data)

2.3 文件传送协议：FTP

■ FTP简介

- 使用TCP协议传输数据（RFC 959）
- C/S模式
- 端口：21/20



2.3 文件传送协议：FTP

■ FTP与HTTP传输文件的共同点

- 均使用TCP协议

■ FTP与HTTP传输文件的不同点

- FTP的控制信息是带外传送的，而HTTP的控制信息则是带内传送的

- FTP存在两个并行的连接

- 控制连接
 - 数据连接



2.3 文件传送协议：FTP

- ❑ FTP连接是有状态的，而HTTP连接则是无状态的
 - FTP服务器会在整个会话期间维护用户的状态信息
 - ❑ 把用户帐户和控制连接联系起来
 - ❑ 追踪用户在远程目录树上的位置
 - ❑ 对活动着的用户会话的状态进行追踪，以限制FTP会话总数

2.3 文件传送协议：FTP

■ 常见的FTP命令

- ❑ USER *username* (登录)
- ❑ PASS *password* (登录)
- ❑ LIST (返回当前目录中的文件列表)
- ❑ RETR *filename* (取 (gets) 文件)
- ❑ STOR *filename* (存(puts) 文件到远程主机)

2.3 文件传送协议：FTP

■ 常见的FTP应答

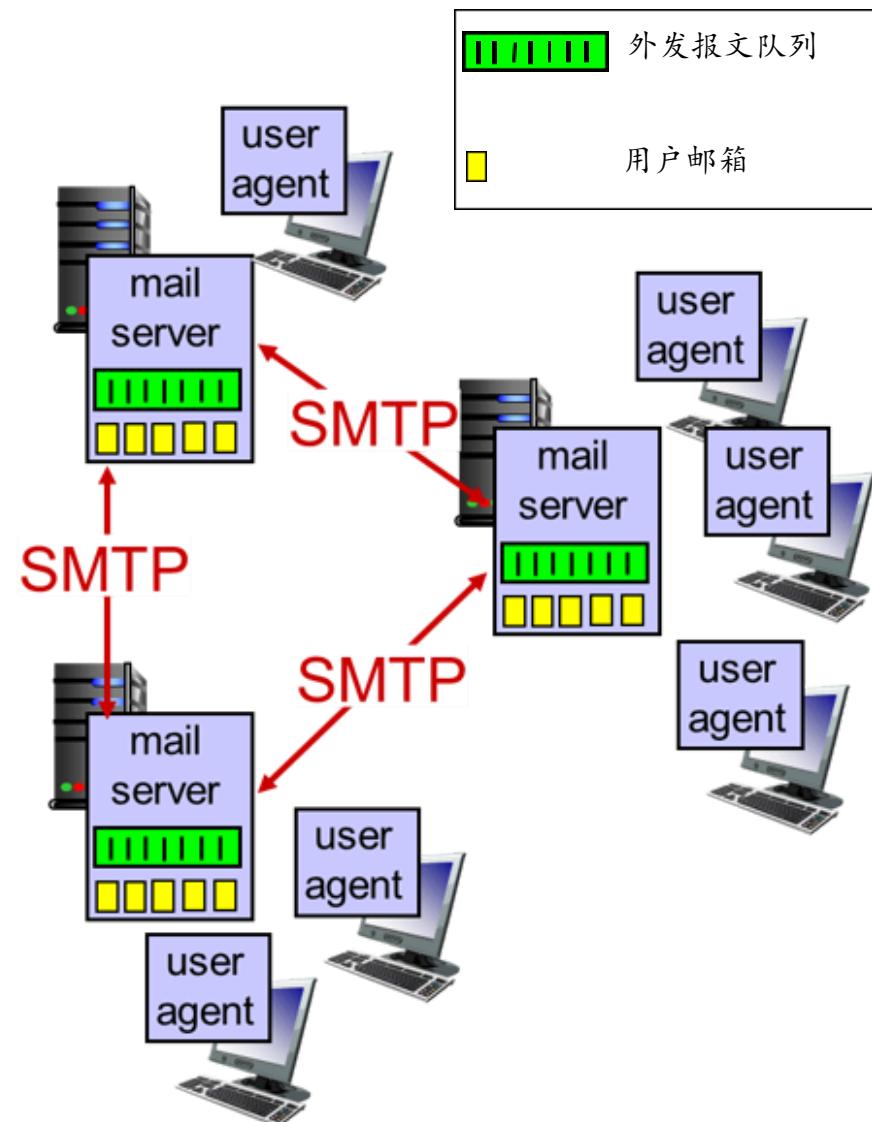
- ❑ 331 Username OK, password required
- ❑ 125 data connection already open; transfer starting
- ❑ 425 Can't open data connection
- ❑ 452 Error writing file

特别注意：FTP的应答和HTTP的应答是否很类似？

2.4 因特网中的电子邮件

电子邮件系统的构成

- 用户代理
- 邮件服务器
- 简单邮件传输协议:
SMTP



2.4 因特网中的电子邮件

■ 用户代理

- 写作, 编辑, 阅读邮件报文
- e.g. OE、FoxMail

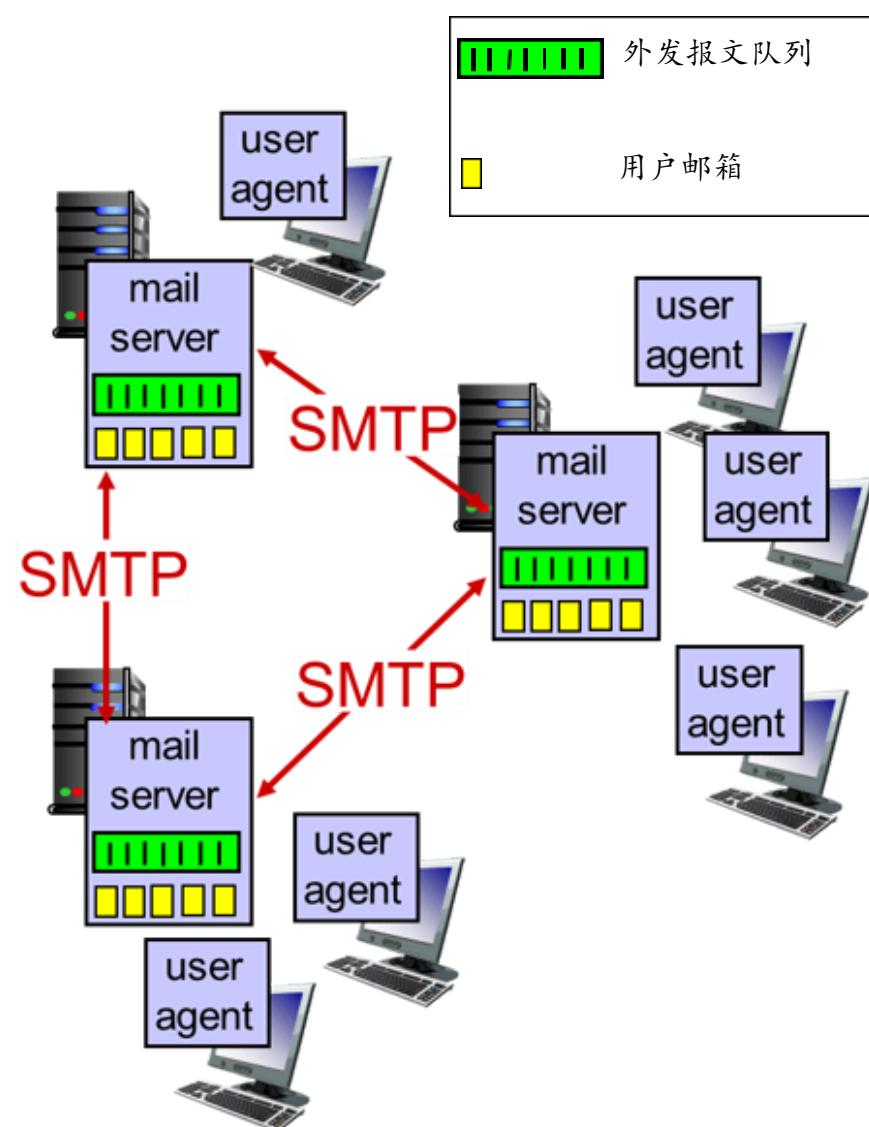
■ 邮件服务器

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

■ SMTP 协议用在邮件服务器

之间发送邮件

- 客户端: 将邮件发送到邮件服
务器
- “服务器”: 接收和转发邮件



2.4 因特网中的电子邮件

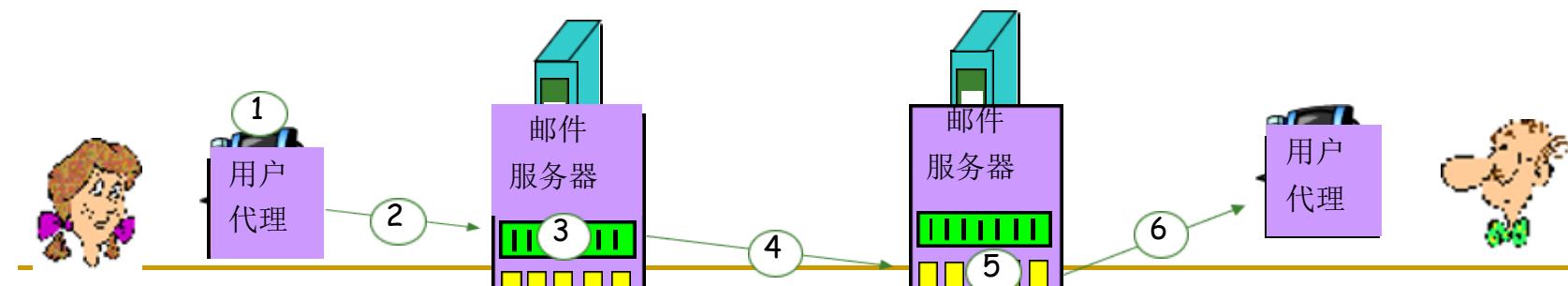
■ SMTP协议

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

2.4 因特网中的电子邮件

■ 一次邮件传送过程

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



2020年9月21日

62

2.4 因特网中的电子邮件

```
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 因特网中的电子邮件

■ SMTP评述

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

■ SMTP vs HTTP

- ❑ 都使用ASCII命令/响应交互,状态码
- ❑ HTTP: pull(拉) and SMTP: push(推)
- ❑ HTTP: 每个对象分装在各自的响应报文中
- ❑ SMTP: 多个对象在一个多分部的报文中传送

2.4 因特网中的电子邮件

■ 邮件报文格式 (RFC 822)

- 首部诸行, e.g.,

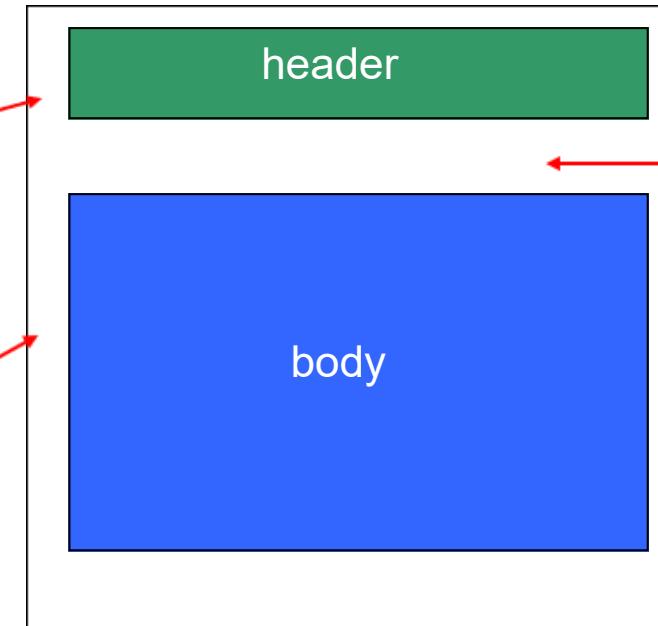
- To:
 - From:
 - Subject:

不同于 smtp 命令!

- 信体

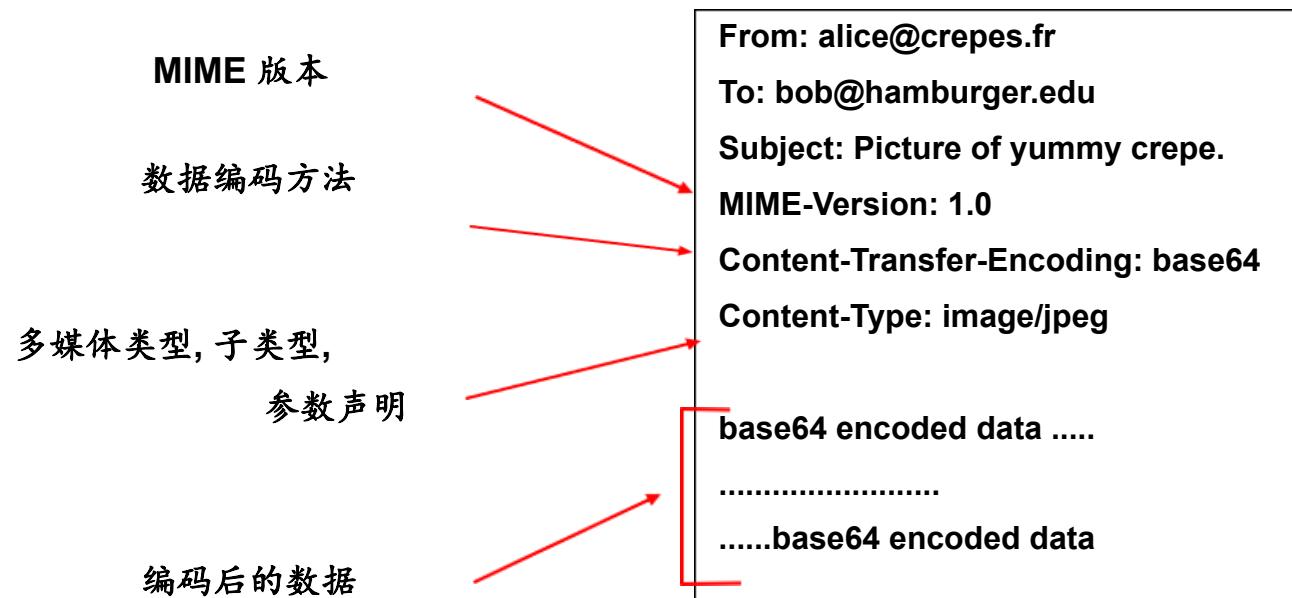
- 即“报文”, ASCII

characters only



2.4 因特网中的电子邮件

■ 非ASCII码数据的MIME扩展



2.4 因特网中的电子邮件

■ 客户机获取邮件的方法

- POP3协议
- IMAP协议
- HTTP

2.4 因特网中的电子邮件

■ POP3协议的认证阶段

□ 客户端命令:

- user: 用户名
- pass: 口令

□ 服务器响应

- +OK
- -ERR

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

2.4 因特网中的电子邮件

■ POP3协议的交互命令

❑ list:

列出报文号码

❑ retr:

用报文号码取信

❑ dele:

用报文号码删信

❑ quit

C: list

S: 1 498

S: 2 912

S: .

C: retr 1

S: <message 1 contents>

S: .

C: dele 1

C: retr 2

S: <message 1 contents>

S: .

C: dele 2

C: quit

S: +OK POP3 server signing off

2.4 因特网中的电子邮件

■ POP3评述

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

■ IMAP协议

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

2.5 DNS:因特网的目录服务

■ 人类社会对人的识别

- 姓名
- 身份证号
- 护照号
-

■ 网络社会对机器的识别

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

IP 地址和域名之间如何映射(转换) ?



为此人类设计了**DNS**系统，用于IP地址和域名之间的转换

2.5 DNS:因特网的目录服务

■ DNS简介

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

2.5 DNS:因特网的目录服务

■ 一次最简单的DNS解析过程

□ 假设

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

□ 结果

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

2.5 DNS:因特网的目录服务

■ DNS的实现

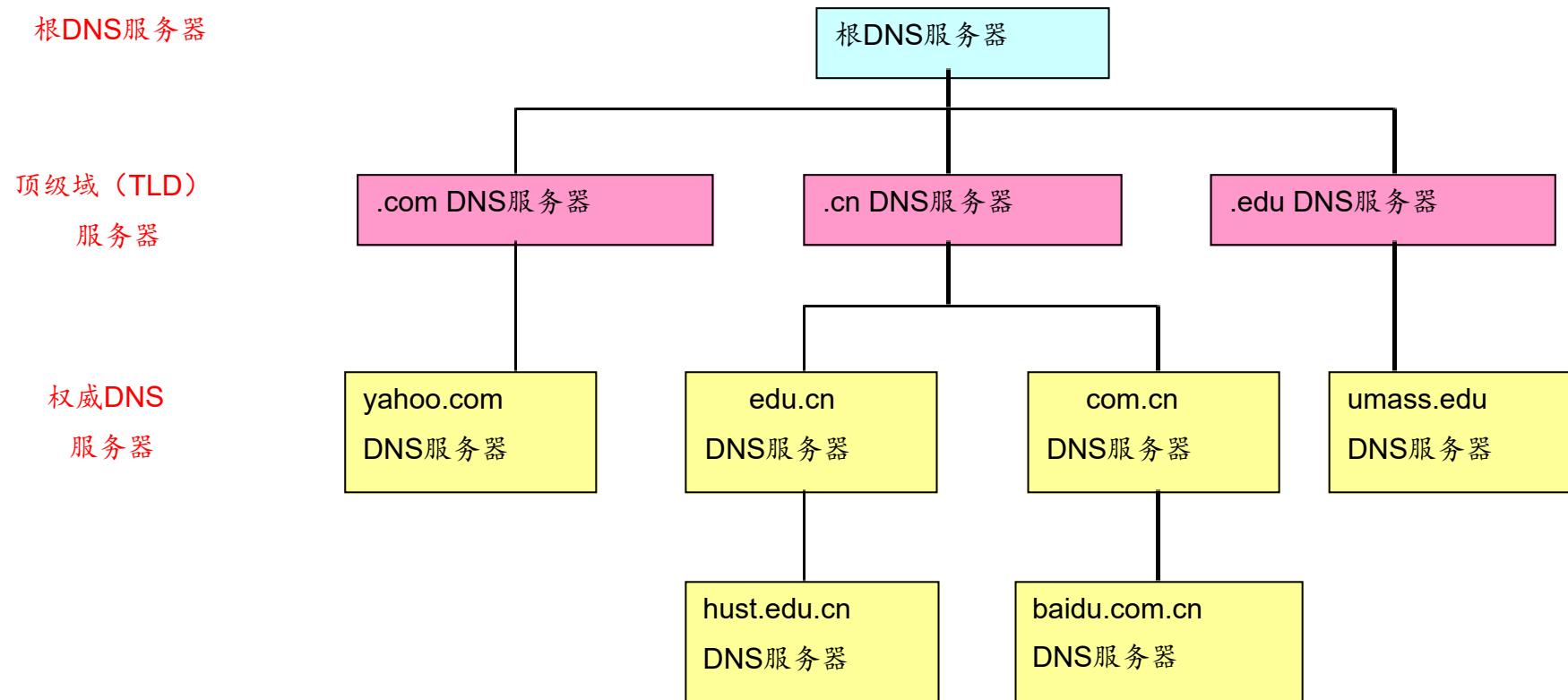
□ 最简单的方法——单台DNS服务器

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

显然，这种方法是世界上最笨的方法！！！

2.5 DNS:因特网的目录服务

□ 真正的DNS实现



2.5 DNS:因特网的目录服务

■ 根域名服务器（截止2012年，共13个）



2.5 DNS:因特网的目录服务

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

2.5 DNS:因特网的目录服务

■ 本地DNS服务器

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

2.5 DNS:因特网的目录服务

一次完整的DNS

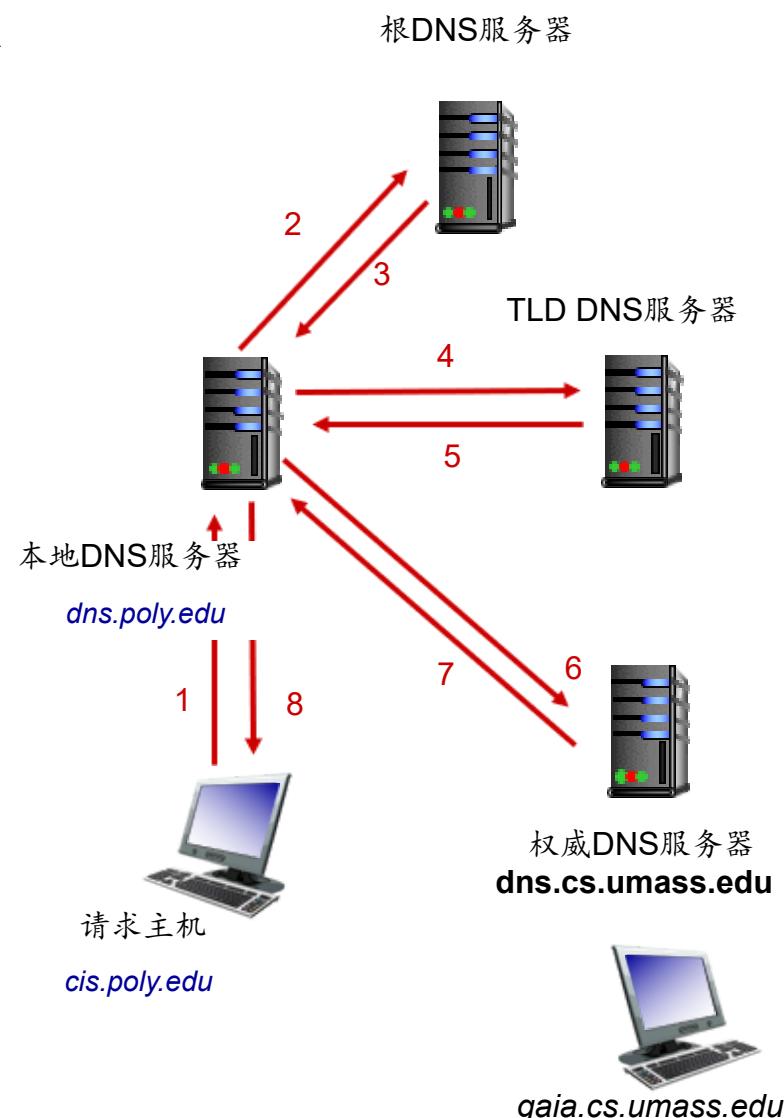
解析过程

□ 发起请求使用递归查询

查询

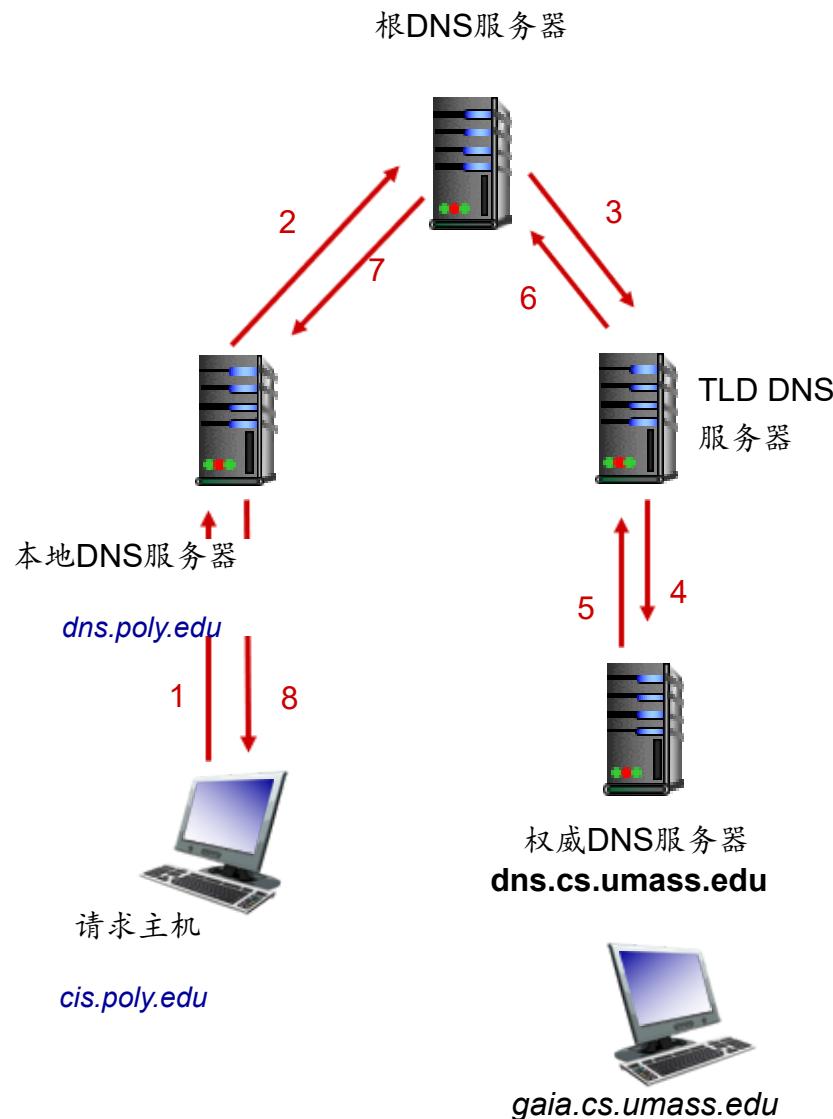
□ 后续解析为迭代查

询



2.5 DNS:因特网的目录服务

- 另外一种DNS解析流程
 - 纯递归查询



2.5 DNS:因特网的目录服务

■ DNS缓存

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

2.5 DNS:因特网的目录服务

■ DNS可提供的服务

- 域名到IP地址的转换
- 主机/邮件服务器别名
 - 为不好记的规范主机/邮件服务器名提供一个易记的别名

e.g. www.hotmail.com -> www.hotmail.ate.nsatc.net
- 负载均衡
 - 一个域名对应多个IP
 - DNS服务器在多个IP中进行轮转

2.5 DNS:因特网的目录服务

■ DNS记录的格式

RR 格式: (name, value, type, ttl)

■ Type=A

- name = 主机名
- value = IP 地址

■ Type=NS

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

■ Type=CNAME

- Name= 别名
- www.ibm.com

■ Type=MX

- value = 与 name相关的邮件服务器域名

2.5 DNS:因特网的目录服务

■ DNS记录的维护

- 目前基本上都是手工维护
- RFC2136定义了DNS动态更新

2.5 DNS:因特网的目录服务

■ DNS报文

- 查询和回答报文的格式是一致的



2.5 DNS:因特网的目录服务

- 向DNS数据库中插入记录
 - 例子：新的企业Network Utopia
 - 在DNS注册机构注册域名networkutopia.com（例如：
Network Solutions）
 - 提供基本和辅助权威DNS服务器的名字和IP地址
 - 向.com的TLD DNS服务器中插入如下两条RR记录
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
 - 将WEB服务器www.networkutopia.com的A记录和邮件
服务器networkutopia.com的MX记录插入权威服务器中

2.5 DNS:因特网的目录服务

■ 攻击DNS服务器

□ DDoS攻击

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

- 通过DNS请求报文洪泛攻击TLD DNS服务器

- 难以过滤

□ 重定向攻击

- 中间人攻击——攻击者截获来自主机的请求并返回伪造的回答
- DNS毒害攻击——攻击者想一台DNS服务器发送伪造的回答，诱使服务器在它的缓存中接收伪造的记录。

□ 利用DNS服务器对目标主机采用DDoS攻击——反射攻击

2.6 P2P文件共享

■ 一次传输的场景

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

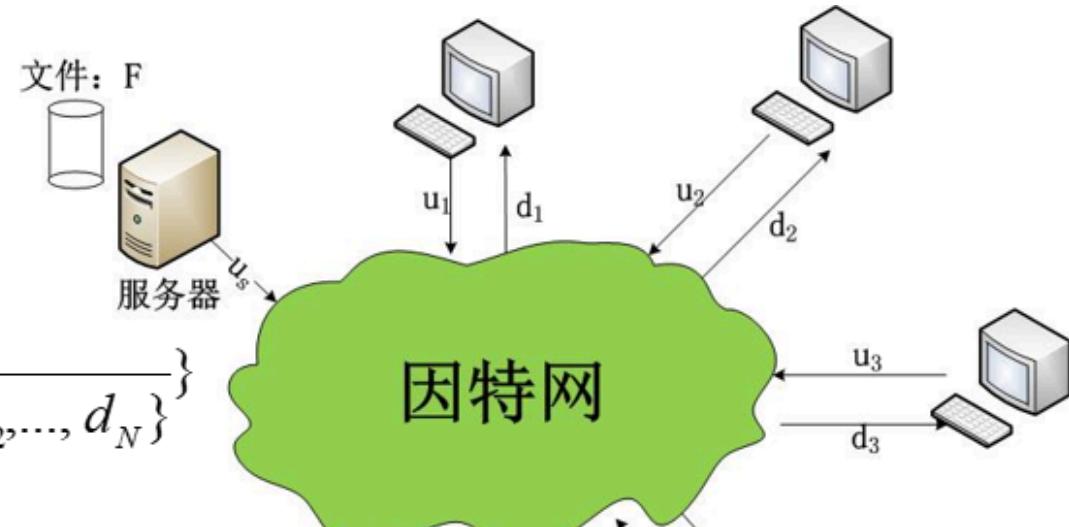
所有的计算机都是服务器 = 高可扩展性

2.6 P2P文件共享

■ 文件分发

□ C/S模式

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



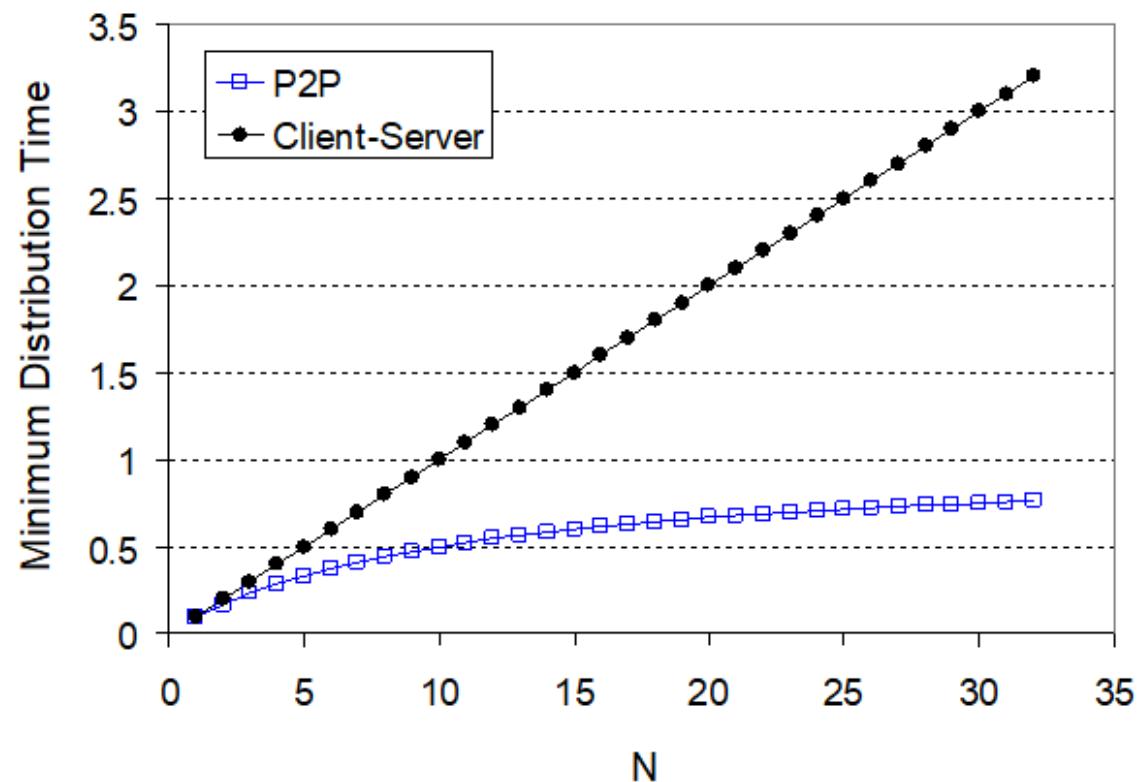
□ P2P模式

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

因特网

2.6 P2P文件共享

- C/S和P2P体系结构的文件分发时间



2.6 P2P文件共享

■ BitTorrent的基本概念

- 洪流(torrent):参与一个特定文件分发的所有对等方的集合
- 追踪器(tracker):跟踪正参与在洪流中的对等方
- 文件块(chunk): 256KB

■ BitTorrent的基本工作机制

- 向邻居请求哪些块——最稀罕优先
- 优先响应哪些请求——对换算法 (4+1)

2.6 P2P文件共享

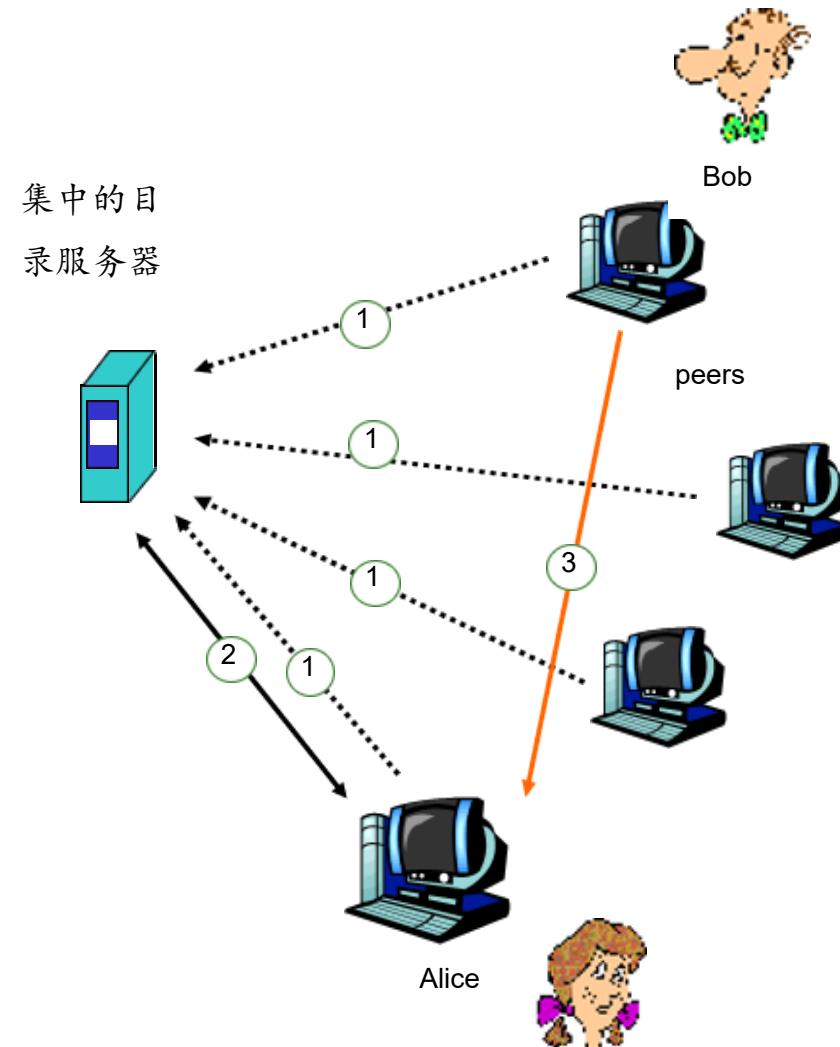
■ P2P文件定位的方法

- 集中式目录——Napster
- 洪泛查询——Gnutella
- 利用不均匀性——KaZaA

2.6 P2P文件共享

集中式目录法

- 当一台计算机上线时，它将下面的信息通知到中央服务器：
 - IP地址
 - 所拥有的资源
- Alice查询“Hey Jude”
- Alice向Bob请求这个文件



2.6 P2P文件共享

■ 集中式目录法存在的问题

- 单点故障
- 性能瓶颈
- 知识产权的侵犯

文件的传输是分布式的，但是资源
的查询定位却是高度集中的

2.6 P2P文件共享

■ 洪泛查询法

□ 特点

- 全分布式
- 无中心服务器

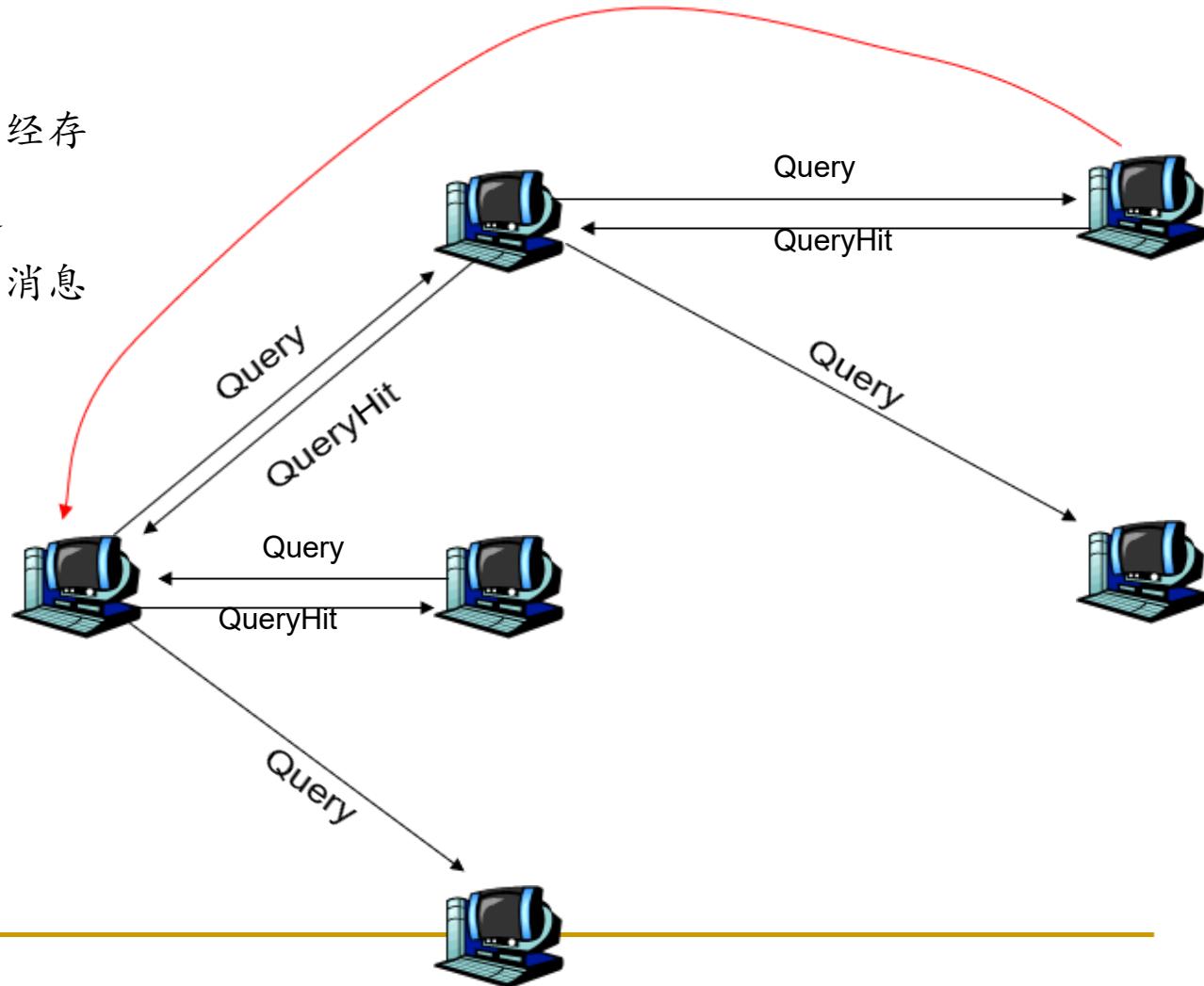
□ 覆盖网络

- 对等方X和Y之间如果维护了一条TCP连接，我们称它们之间有一条边
- 所有活动的对等方和边构成了“覆盖网络”
- 边并非一个物理的连接
- 一个对等方通常所连接的该覆盖网络中的节点要少于10个

2.6 P2P文件共享

□ 文件定位方法

- 查询消息通过已经存在TCP连接传送
- 对等方转发查询消息
- 查询命中的消息通过完全相反路径传送回来



2020年9月21日

96

2.6 P2P文件共享

□ 对等方加入覆盖网络的方法

- 欲加入的对等方X首先必须通过对等方列表发现已经在覆盖网络中的其他的对等方。
- X将试图与列表中的对等方建立TCP连接，直到与某个对等方Y成功建立这样一条连接
- 连接建立成功之后X会向Y发送一个Gnutella的Ping消息，Y受到这个消息之后会向他的所有邻居转发这个Gnutella的Ping消息
- 任何一个对等方受到这个消息之后，会通过覆盖网络向X发送Gnutella的Pong消息
- 当X收到这些Pong消息之后，它不仅知道了Y，而且知道了该覆盖网络中的所有其他的对等方的IP地址，这样X就能同其他的对等方建立TCP连接

2.6 P2P文件共享

■ 利用不均匀性法

□ 思想

- 仍然是全分布的，但所有的对等方并不都是平等的，
存在部分具有特权的对等方，称之为“**组长**”
 - 组长通常具有高带宽连接和高因特网连通性
- 组长之间以洪泛查询的方式连接成覆盖网络
- 每一个普通对等方必须指派到一个组长，从而形成以
组长为中心的小“Napster”

2.6 P2P文件共享

□ 内容定位的基本要点

- 每一个资源都有一个“摘要”和一个描述符
- 客户端向它的“组长”发送一个关键字查询请求
- “组长”返回匹配的内容：
 - 这是一个精确的匹配，包括：摘要，IP地址，描述符等
- 如果“组长”向其他的“组长”转发了这个查询请求，那么它们都会返回匹配的内容。
- 客户端将选择一个匹配的内容进行下载
 - HTTP请求报文使用这个摘要作为被请求资源的标识符向持有该资源的对等方请求下载。

2.6 P2P文件共享

□ 一些改进的技巧

- 请求排队
- 激励优先权
- 并行下载

2.7 视频流和内容分发网络

- 视频流量：互联网带宽的主要消耗者
 - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
 - ~1B YouTube users, ~75M Netflix users
- 挑战：如何达到大约十亿规模的用户
 - 单独的网络视频服务器不能处理(why?)
- 挑战：异构性
 - 不同的用户有不同的需求（例如，有线与移动；带宽的大小）
- *solution:*分布式、应用级的基础架构



多媒体：视频

- 视频：以恒定速率显示的图像序列
 - e.g., 24 images/sec
- 数字图像：像素阵列
 - 每个像素用bit表示
- 编码：在图像内部和图像之间使用冗余来减少用于编码图像的比特数
 - 空间 (within image)
 - 时间 (from one image to next)

空间编码示例：仅发送两个值：
颜色值（紫色）和重复值的数量
(N)，而不是发送N个相同颜色
的值（全紫色），

frame i

时间编码示例：仅发送与
帧*i*的差异，而不是发送*i* +
1的完整帧

frame $i+1$

多媒体：视频

- CBR: (constant bit rate): 固定视频编码率
- VBR: (variable bit rate): 视频编码率随空间, 时间编码量的变化而变化
- 常见协议:
 - MPEG 1 (CD-ROM) 1.5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4 (often used in Internet, < 1 Mbps)

空间编码示例：仅发送两个值：
颜色值（紫色）和重复值的数量
(N)，而不是发送N个相同颜色的值（全紫色），



frame i

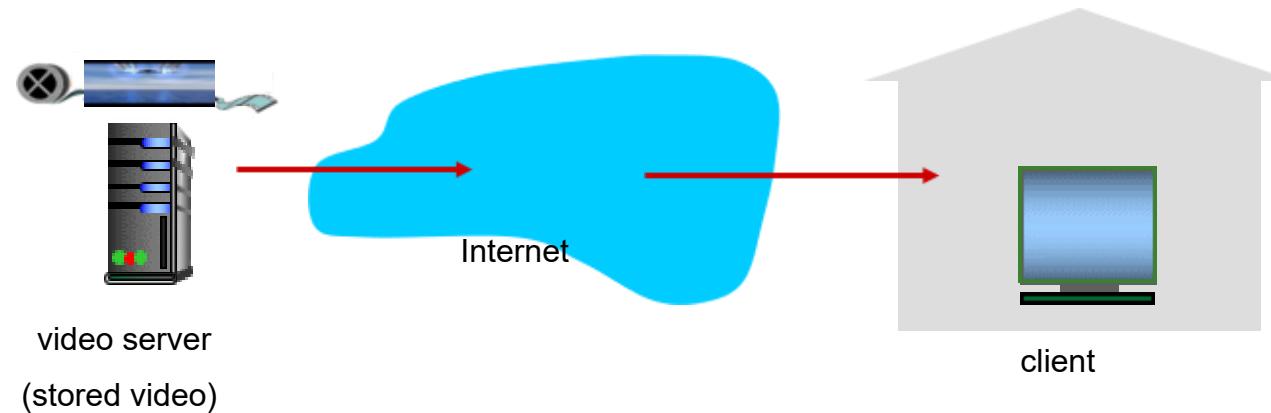
时间编码示例：仅发送与帧*i*的差异，而不是发送*i* + 1的完整帧



frame $i+1$

流存储的视频:

简单场景:



流媒体: DASH

■ **DASH**: 动态的、自适应的HTTP流

■ **server:**

- 将视频文件分成多个块
- 每个块以不同的速率存储和编码
- **清单文件**: 提供不同块的URL

■ **client:**

- 定期测量服务器到客户端的带宽
- 查询清单，每次请求一个数据块
 - 在给定当前带宽的情况下选择最大编码率
 - 可以在不同的时间点选择不同的编码率（取决于当时的可用带宽）

流媒体: DASH

- **DASH:**动态的、自适应的HTTP流
- 客户端的“智能”：客户端确定
 - 何时去请求块（避免发生缓冲区不足或溢出）
 - 请求哪种编码率（在有更多可用带宽时质量更高）
 - 在哪里请求块(可以从离客户端“近”的URL服务器或有高可用带宽的URL服务器请求)

内容分发网络（Content distribution networks, CDN）

■ **挑战:** 如何将内容(从数以百万计的视频中选择)传送给数十万同
步用户?

■ **方案 1:**单个大型“巨型服务器”

单点故障

网络拥塞点

到远端客户端的长路径

通过传出链路发送多个视频副本

...很简单：此解决方案无法扩展

内容分发网络

- 方案 2: 在多个地理位置分散的站点 (CDN) 存储/提供多个视频

副本(CDN)

- *enter deep*: 将CDN服务器推入许多接入网络

- 贴近用户

- Akamai公司所使用，1700个地点

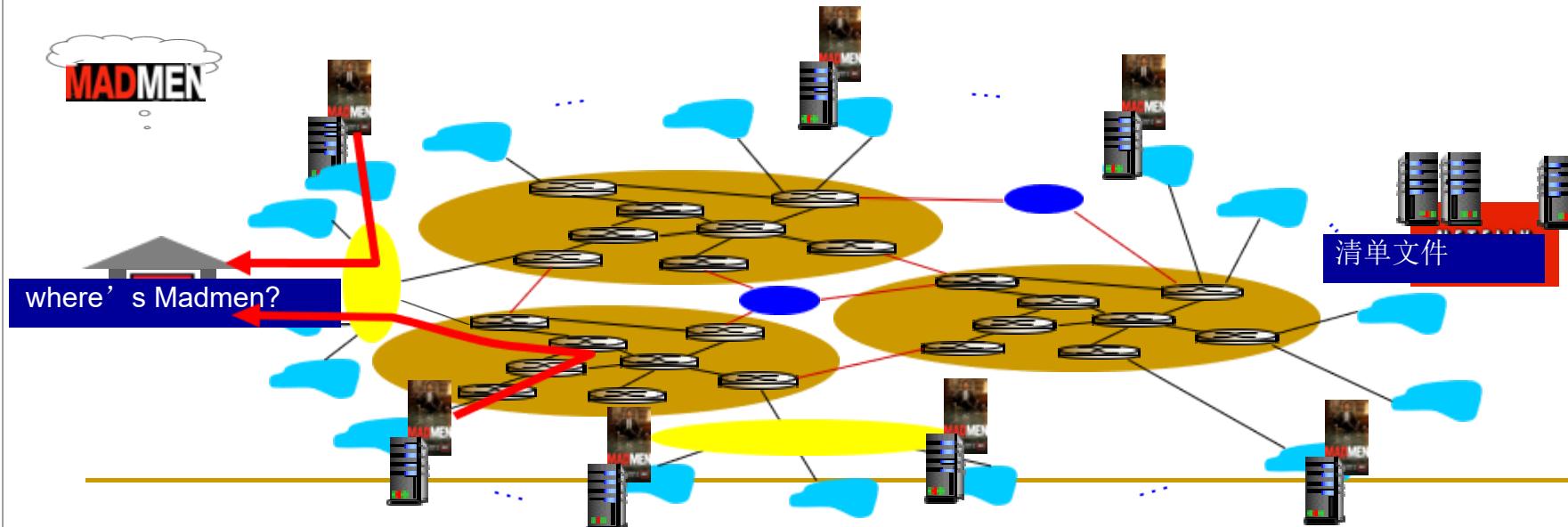
- *bring home*: 靠近访问网络（但不在访问网络内）的POP中数

- 量较少（10个）的较大群集

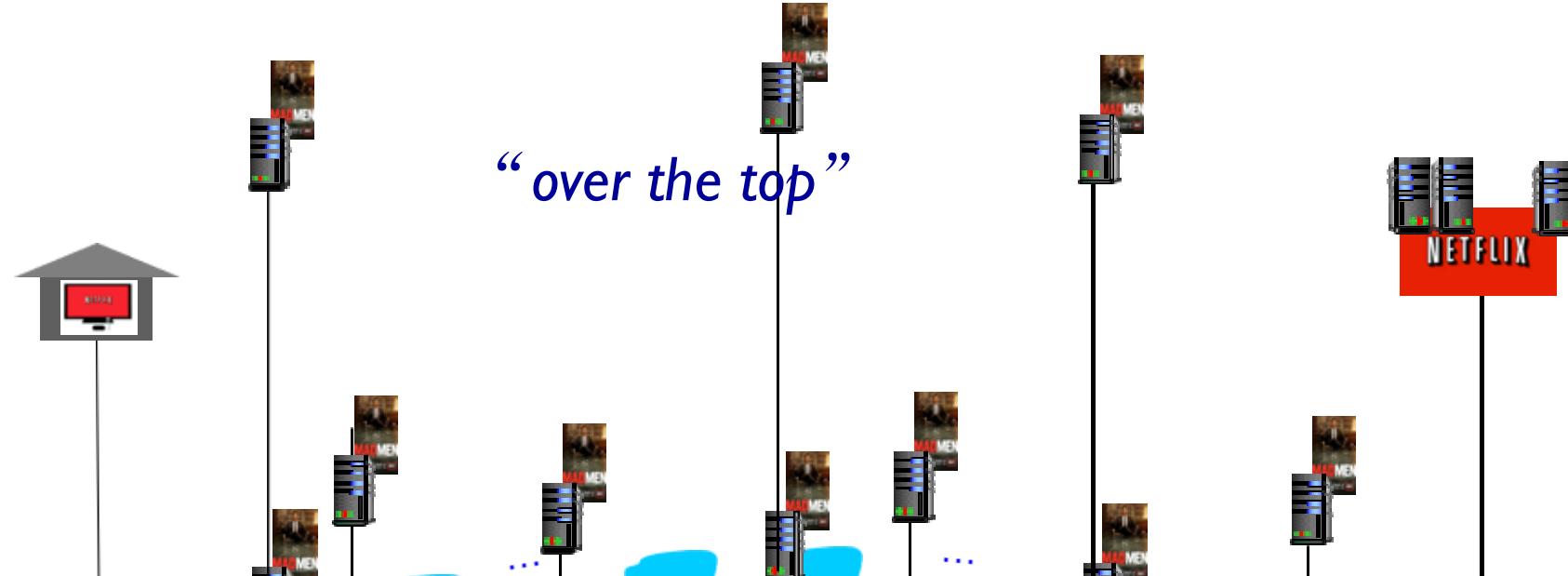
- Limelight公司所使用

内容分发网络

- CDN: 将内容的副本存储在CDN节点上
 - e.g. Netflix存储MadMen的副本
- 订阅用户从CDN请求内容
 - 定向到附近的副本，检索内容
 - 如果网络路径拥塞，可以选择其他副本



内容分发网络



Internet主机-主机通信作为一种服务

OTT challenges:应对拥挤的互联网

从哪个CDN节点检索内容？

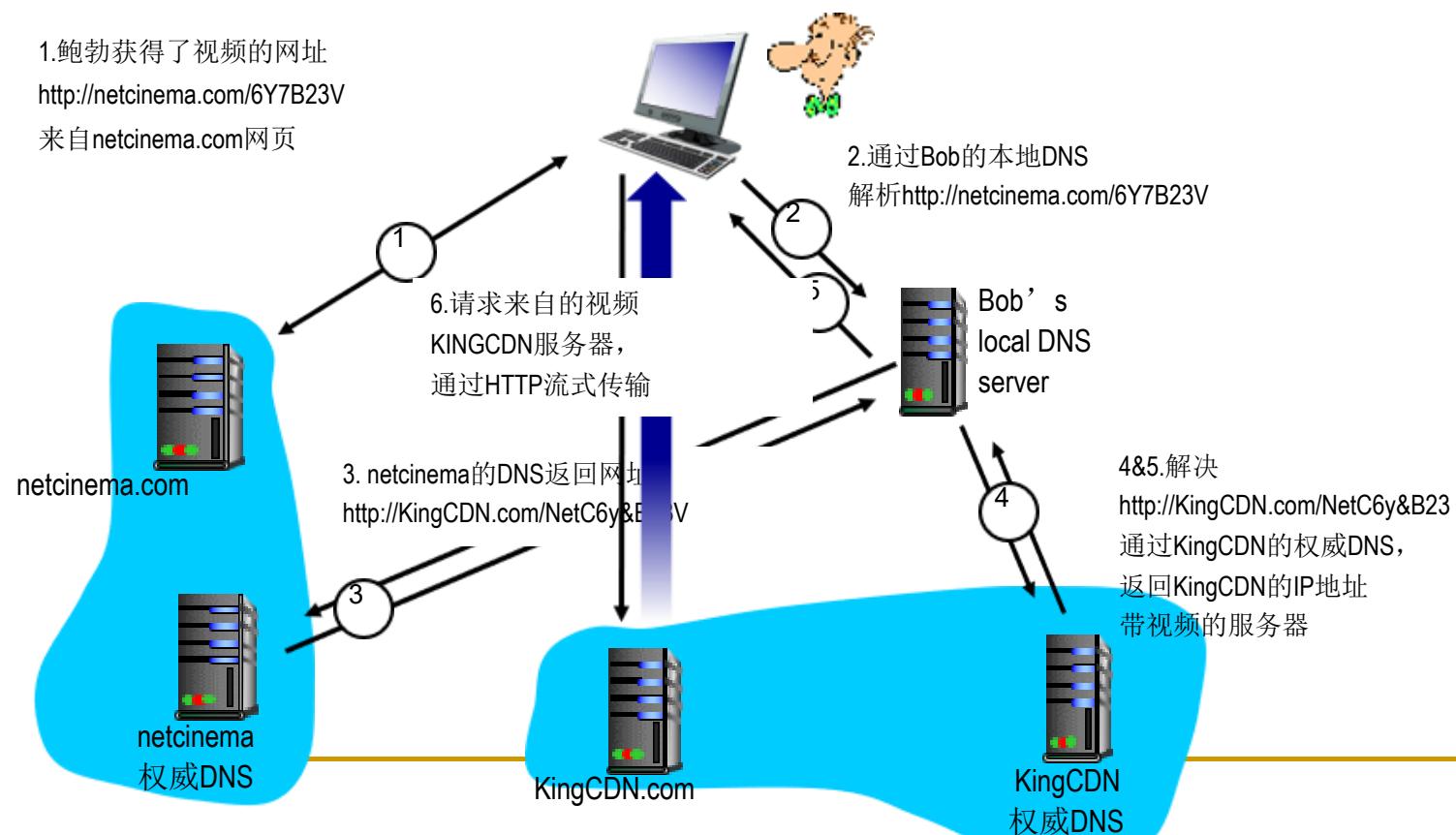
用户在拥挤的情况下下的行为？

CDN节点中应放置哪些内容？

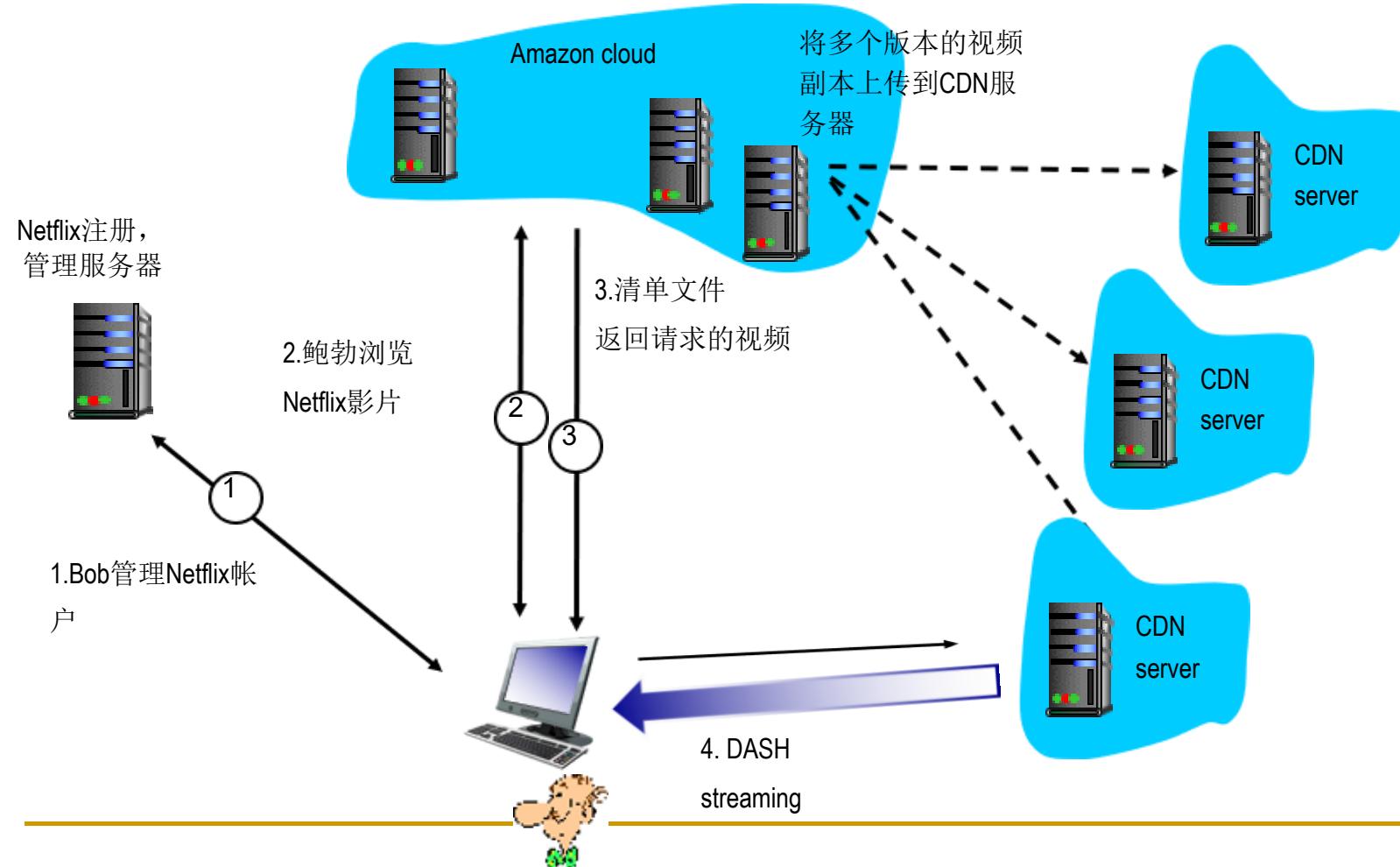
CDN内容访问

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

存储在CDN中的视频, 网址为<http://KingCDN.com/NetC6y&B23V>



案例分析: Netflix

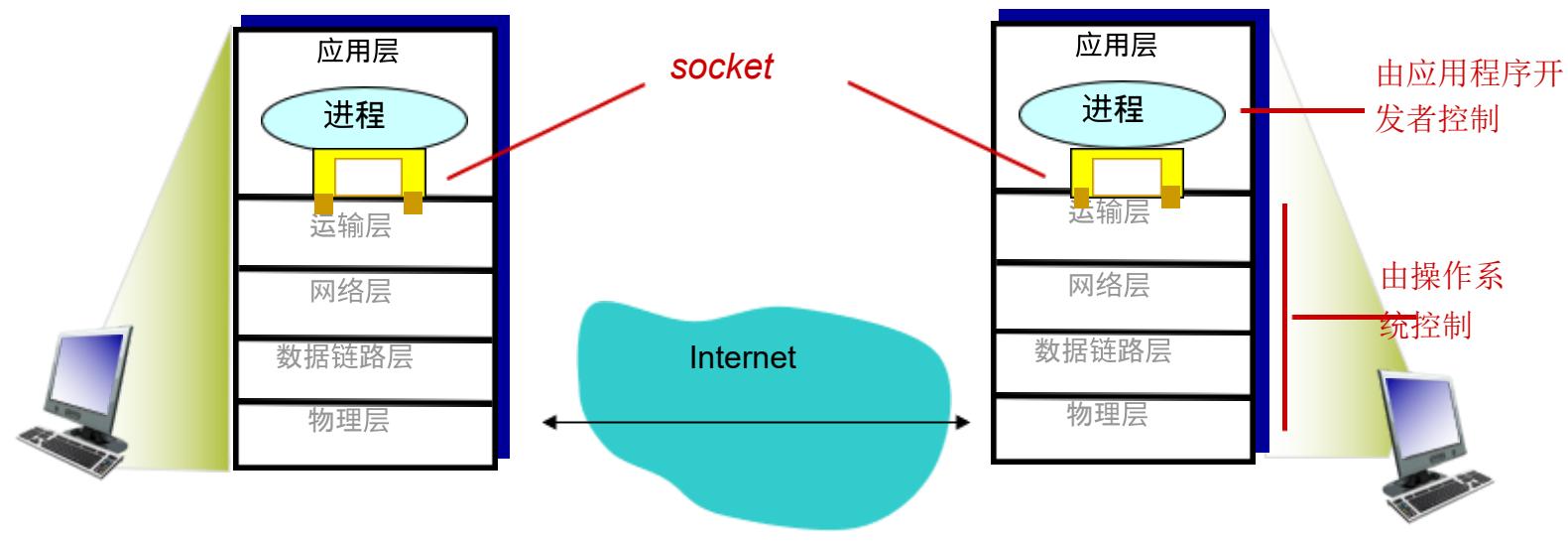


2.8 套接字编程

目标:了解如何构建使用套接字（Socket）进行通信的客户

端/服务器应用程序

套接字:通道之间的应用进程和终端-终端传输协议



套接字编程

两种传输服务的套接字类型:

- **UDP**:不可靠的数据报
- **TCP**:可靠，面向字节流

应用范例:

- 1.客户端从其键盘读取一行字符（数据）并将数据发送到服务器
- 2.服务器接收数据并将字符转换为大写
- 3.服务器将修改后的数据发送到客户端
- 4.客户接收修改的数据并在其屏幕上显示行

UDP套接字编程

UDP:客户端与服务器之间无“连接”

- 发送数据前无握手
- 发送方显式地将IP目标地址和端口号附加到每个数据包
- 接收者从接收到的数据包中提取发送者IP地址和端口号

UDP:传输的数据可能丢失或乱序接收

应用观点:

- UDP在客户端和服务器之间提供不可靠的字节组(“数据报”)传输

UDP的客户端/服务器套接字交互

服务器端(以服务器IP运行)

```
create socket, port= x:  
  
serverSocket =  
socket(AF_INET,SOCK_DGRAM)
```

↓
read datagram from
serverSocket
↓
write reply to
serverSocket
specifying
client address,
port number

客户端

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

↓
Create datagram with server IP and
port=x; send datagram via
clientSocket

↓
read datagram from
clientSocket
↓
close
clientSocket

UDP应用示例：客户端程序

UDPCClient.py (Python编写)

使用包含套接字方法的Python库

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
message = raw_input('Input lowercase sentence: ')
```

为客户创建UDP套接字

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

获取用户键盘输入

```
message = raw_input('Input lowercase sentence: ')
```

将服务器名称，端口附加到消息；发送到套接字

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

将套接字中的应答字符读取为字符串

```
modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)
```

打印出接收到的字符串并关闭套接字

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

UDP应用示例：服务器端程序

UDPServer.py

```
from socket import *
serverPort = 12000
创建 UDP socket      serverSocket = socket(AF_INET, SOCK_DGRAM)
将套接字绑定到      serverSocket.bind(("", serverPort))
本地端口号12000      print ("The server is ready to receive")
无限循环            while True:
从UDP套接字读取到消息中，获取客户端的地址（客户端IP和端口） message, clientAddress = serverSocket.recvfrom(2048)
将大写字符串发送回此客户端 modifiedMessage = message.decode().upper()
                                         serverSocket.sendto(modifiedMessage.encode(),
                                         clientAddress)
```

TCP套接字编程

客户必须连接服务器

- 服务器进程必须先运行
- 服务器必须创建了欢迎客户连接的Socket(通道)

客户通过以下方式连接服务器：

- 创建TCP套接字，指定IP地址，
服务器进程的端口号
- 客户端创建套接字时：客户端
TCP建立与服务器TCP的连接

- 当客户端与服务器联系时，服务器TCP创建新的套接字，以使服务器进程与该特定客户端进行通信

- 1.允许服务器与多个客户端通话
- 2.用于区分客户端的源端口号(第3章中有更多信息)

application viewpoint:

TCP提供可靠，有序的
字节流传输（“管道”）
客户端和服务器之间

TCP的客户端/服务器套接字交互

服务器端 (以服务器IP运行)

客户端

```
create socket,  
port=x, for incoming  
request:  
serverSocket = socket()
```

```
wait for incoming  
connection request  
connectionSocket =  
serverSocket.accept()
```

```
read request from  
connectionSocket
```

```
write reply to  
connectionSocket
```

```
close  
connectionSocket
```

TCP
建立连接

```
create socket,  
connect to hostid, port=x  
clientSocket = socket()
```

```
send request using  
clientSocket
```

```
read reply from  
clientSocket
```

```
close  
clientSocket
```

TCP应用示例：客户端程序

TCPCClient.py

为客户端创建TCP套接字
连接到服务器
serverName, 端口
12000

向服务器端发送文本
无需附加服务器名称，端口

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

sentence = raw_input('Input lowercase sentence: ')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```



TCP应用示例：服务器端程序

TCPServer.py

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

创建TCP套接字 → from socket import *

服务器开始侦听传入的TCP请求 → serverPort = 12000
→ serverSocket = socket(AF_INET,SOCK_STREAM)
→ serverSocket.bind(('',serverPort))
→ serverSocket.listen(1)

无限循环 → print ('The server is ready to receive')
→ while True:

服务器等待accept() → connectionSocket, addr = serverSocket.accept()

对于传入的请求，在返回时创建新的套接字

从套接字读取字节（但不读取地址） → sentence = connectionSocket.recv(1024).decode()

修改（转成大写）后送回客户端 → capitalizedSentence = sentence.upper()
→ connectionSocket.send(capitalizedSentence.encode())

关闭与此客户端的连接（而不是套接字） → connectionSocket.close()

课后思考题

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

实验一 Socket编程

- 目的——掌握**Socket**编程的基本方法，能够使用**Socket**完成基本的网络应用程序编写
- 内容（三选一）
 - 支持多线程访问的WEB服务器（支持HTTP 1.1）
 - FTP客户端
 - 邮件客户端
- 要求

实验一 Socket编程

■ 要求

- 具有图形界面，操作友好
- 基于Windows操作系统
- 不能使用开发平台底层类库封装的FTP类、WWW类和Mail类，可使用Socket类
- 可使用开发平台包括：C++、C#、Java、Python
- 核心功能必须封装为单独的类