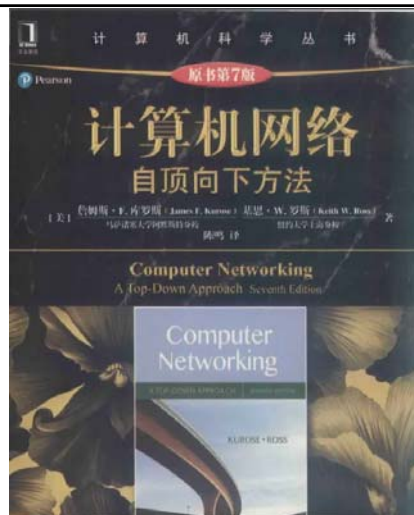


## Chapter 2 应用层 Application Layer



## 应用层 The Application Layer: Overview

- 网络应用的原理
- Web和HTTP
- E-mail, SMTP, IMAP
- 域名系统DNS
- P2P应用
- 视频流和内容分发网络
- 使用UDP和TCP进行套接字编程



Application Layer: 2-2

## 应用层：简介 The Application Layer

### 我们的目标:

- 应用层协议的概念和实现
- 传输层服务模型
- 网络应用程序体系结构
  - 客户-服务器体系结构: client-server architecture
  - P2P（对等方到对等方）体系结构: peer-to-peer architecture
- 通过学习流行的应用程序层协议来了解协议
  - HTTP
  - SMTP, IMAP
  - DNS
- 网络应用编程
  - socket API

Application Layer: 2-3

## 网络应用是计算机网络存在的理由

- 社交网络
- Web
- 短信
- e-mail
- 多用户网络游戏
- 流媒体存储视频(YouTube, Hulu, Netflix奈飞)
- P2P文件共享
- IP语音(e.g., Skype)
- 实时视频会议
- 互联网搜索
- 远程登录
- ...

**Q:** 你喜欢的网络应用是什么?

Application Layer: 2-4

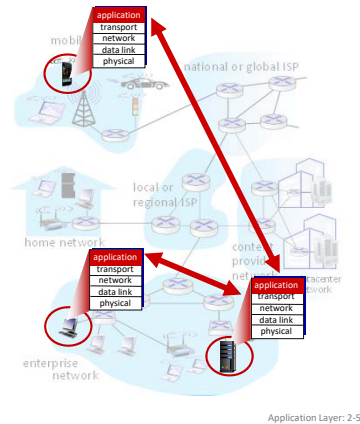
## 创建网络应用程序/应用

### 编写程序:

- 在(不同的)端系统上运行
- 通过网络进行通信
- 例如, Web服务器软件与浏览器软件通信

### 不需要为网络核心设备编写软件

- 网络核心设备不运行用户应用程序, 主要是他们并不在应用层上起作用
- 端系统允许快速的网络应用程序的开发
- 将应用软件限制在端系统上, 促进了大量的网络应用程序的迅速开发和部署



Application Layer: 2-5

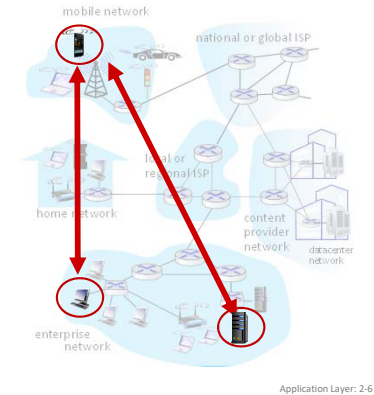
## 网络应用程序体系结构: 客户-服务器体系结构 Client-server architecture

### 服务器 server:

- 不间断运行的主机
- 永久的IP地址
- 通常是在数据中心, 易于扩展

### 客户clients:

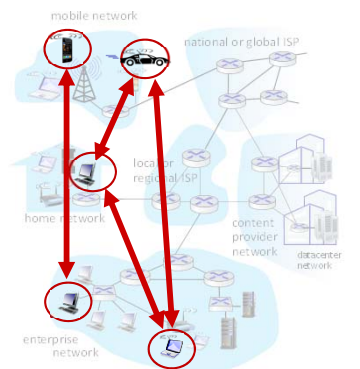
- 与服务器连接、通信
- 可能是断续连接
- 可能有动态IP地址
- 客户与客户之间不直接通信
- 例如: HTTP, IMAP, FTP



Application Layer: 2-6

## 网络应用程序体系结构: 对等方到对等方/P2P 体系结构 Peer-peer architecture

- 没有不间断运行的服务器
- 任意端系统直接通信
- Peers (对等方) 请求的服务来自其他对等方, 并提供服务给其他对等方
  - self scalability (自扩展性) - 新的对等方带来新的业务容量, 以及新的业务需求
- 对等方连接闪断, 会改变IP地址
  - 复杂的管理
- 非集中式结构, 面临安全性、性能和可靠性等挑战
- 示例: P2P文件共享



Application Layer: 2-7

## 进程通信 Processes Communicating

一个Process (进程): 运行在端系统中的程序

- 在同一端系统内, 两个进程使用进程间通信机制(由OS定义)相互通信。
- 不同端系统中的进程通过跨越计算机网络交换报文相互通信

客户clients, 服务器 servers

client process

(客户进程/客户):

发起通信的进程

server process

(服务器进程/服务器):

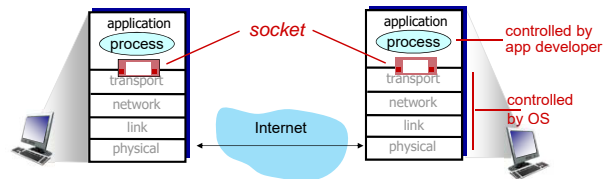
等待联系的进程

- 注意: 使用P2P架构的应用程序也有客户进程和服务器进程

Application Layer: 2-8

## 进程与计算机网络之间的接口-套接字 Sockets

- 进程向它的**套接字**发送/接收消息
- 进程类比房子，进程的套接字类比于房子的门
  - 发送进程将报文推出门（套接字）
  - 发送进程假定该门到另外一侧之间有传输基础设施，该设施将报文传送到目的进程的门口，报文通过目的进程的门口（套接字）传递
  - 涉及两个套接字：每侧一个
- 也称为应用程序编程接口API：建立网络应用程序的可编程接口



## 进程寻址 Addressing Processes

- 标识 接收消息,过程必须有**标识 (identifier)**
- 主机设备有唯一的32位IP地址
- Q:**进程所在主机的IP地址是否足以识别进程?
- A:**不足，许多进程可以在同一台主机上运行
- 标识包括IP地址和端口号**  
标识包括主机上与进程相关联的**IP地址 (IP address)** 和 **端口号 (port number)**。
  - 例如端口号:
    - HTTP server: 80
    - mail server: 25
  - 发送HTTP消息到 gaia.cs.umass.edu web服务器:
    - IP address:** 128.119.245.12
    - port number:** 80
  - 更多...

Application Layer: 2-10

## 应用层协议定义什么?

- 交换的报文类型 types of messages exchanged,**
  - 例如,请求、响应
- 报文的语法 message syntax:**
  - 消息中的哪些字段&如何描述字段
- 报文的语义 message semantics**
  - 字段中信息的含义
- 规则 rules**
  - 何时以及如何发送和响应消息的流程
- 公共/开放协议 open protocols:**
  - 在RFCs中定义，每个人都可以读取和获得协议的定义
  - 允许互操作性
  - 例如：HTTP, SMTP
- 专用协议 proprietary protocols:**
  - 例如：Skype

Application Layer: 2-11

## 应用程序需要怎样的传输服务?

- 可靠数据传输 reliable data transportation**
  - 一些应用程序(例如，文件传输、电子邮件、金融应用)需要100%可靠的数据传输
  - 其他应用程序(如音频)可以承受一些损失-容忍丢失的应用
  - 提供确保的数据交付服务的传输协议
- 定时 timing**
  - 一些实时的应用程序(如网络电话、多方游戏)需要低延迟才能“有效”
- 吞吐量 throughput**
  - 可用吞吐量**  
发送进程能够向接收进程交付比特的速率
  - 带宽敏感的应用需要传输协议确保可用吞吐量总是至少为r比特/秒**  
如网络电话需要最少的吞吐量才能“有效”
  - 弹性应用利用可供使用的吞吐量**  
如电子邮件、文件传输等(“弹性应用程序”)利用可供使用的任何吞吐量
- 安全 security**
  - 加密、数据完整性、端点鉴别 ...

Application Layer: 2-12

## 常见应用对传输服务的要求

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

Application Layer: 2-13

## 互联网/因特网提供的传输服务

### TCP 服务:

- **可靠传输**: 在发送和接收进程之间
- **面向连接**: 在客户和服务器进程之间需要建立连接 (setup)
- **流量控制**: 发送数据的速度决不超过接收的速度
- **拥塞控制**: 当网络出现拥塞时, 抑制发送进程, 减缓发送速度
- **不提供**: 实时性、最小带宽承诺

### UDP 服务:

- **“不可靠的”数据传输**: 在客户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

Q: why bother? Why is there a UDP?

Application Layer: 2-14

## 流行的网络应用及其应用层协议和传输层协议

应用	应用层协议	所依赖的传输协议
文件传输	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web	HTTP 1.1 [RFC 7320]	TCP
网络电话	SIP [RFC 3261], RTP [RFC 3550], or proprietary HTTP	TCP or UDP
流媒体	[RFC 7320], DASH	TCP
网络交互游戏	魔兽世界, FPS (proprietary)	UDP or TCP

Application Layer: 2-15

## TCP的安全性服务加强

### TCP & UDP sockets:

- 没有加密
- 发送到套接字的明文密码以明文在Internet传播!
- **安全套接字层/传输层安全 (SSL (前身) /TLS)**
- SSL: Secure Socket Layer
- TLS: Transport Layer Security
- 使用加密确保机密性
- 使用数字签名确保数据完整性
- 端点鉴别

### 应用层实现SSL/TLS

- 应用程序使用TLS类/库, 而TLS库使用TCP
- 对发送到套接字的明文加密后在互联网上传输
- 参见第八章

Application Layer: 2-16

## 应用层

- 网络应用原理
- Web 和 HTTP
- E-mail, SMTP, IMAP
- 域名系统DNS
- P2P应用程序
- 视频流媒体和内容分发网络
- 编程与UDP和TCP



Application Layer: 2-17

## Web和HTTP

- Web的应用层协议是HTTP
- web页面(也叫文档)由**对象(objects)**组成，每一个对象可存储在不同的web服务器上
- 1个对象可以是1个HTML文件，1个JPEG图像，1个Java小程序，一个音频文件，...
- 1个web页面通常包含1个**HTML基本文件(base HTML file)**，几个**引用对象(referenced objects)**，
- HTML基本文件通过引用对象的**URL**地址引用它们寻址

www.someschool.edu/someDept/pic.gif  
主机名 路径名

Application Layer: 2-18

## HTTP

HTTP:超文本传输协议 hypertext transfer protocol

- Web的应用层协议
- 客户/服务器模式:
  - **client**: 浏览器 (browser) 请求、接收 (using HTTP protocol) 展示Web对象
  - **server**: Web服务器 (Web server) 发送 (using HTTP protocol) 对象，对请求进行响应



Application Layer: 2-19

## HTTP

**HTTP使用TCP:**

- 客户发起到服务器的TCP连接(创建套接字)，默认端口80
- 服务器接收来自客户的TCP连接
- 浏览器(HTTP客户端)和Web服务器(HTTP服务器)之间交换HTTP报文(应用层协议报文)
- TCP连接关闭

**HTTP是“无状态的”**  
服务器不维护关于过去客户请求的信息

*aside*  
**维护“状态”的协议非常复杂!**

- 过去的历史(状态)必须保留
- 如果服务器/客户崩溃，它们的“状态”视图可能不一致，必须协调

Application Layer: 2-20

## 两种类型的HTTP连接

### 非持续连接

#### Non-persistent connection

1. 打开1个到服务器的TCP连接
2. 通过该TCP连接最多传送1个对象，仅处理1个请求响应事务
3. 关闭该TCP连接

即  
下载多个对象需要多个连接

- 多个连接可以并行进行，如大部分浏览器每次打开5-10个并行连接

Application Layer: 2-21

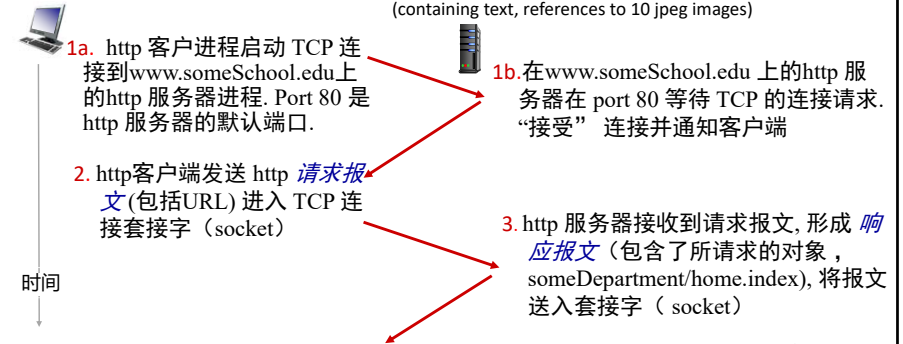
### 持续连接

#### Persistent connection

- 打开1个到服务器的TCP连接
- 通过该连接传送多个对象，即同一台服务器上一个完整的页面用单个持续TCP连接传送
- 关闭该TCP连接
- 甚至，位于同一台服务器的多个页面也可以用单个持续连接发送给同一用户

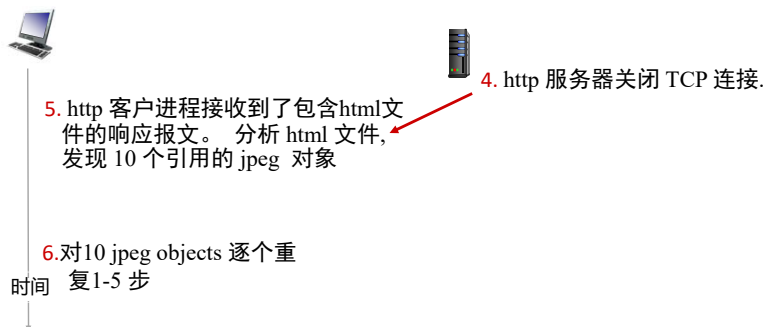
## 非持续连接的HTTP: 例子

假设用户键入了一个 URL :  
`www.someSchool.edu/someDepartment/home.index`  
(containing text, references to 10 jpeg images)



Application Layer: 2-22

## 非持续连接的HTTP: 例子



Application Layer: 2-23

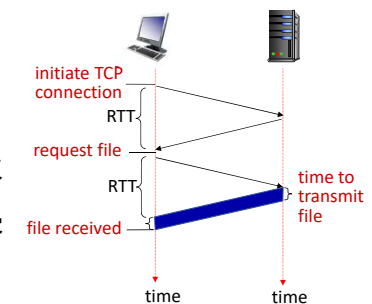
## 非持续连接的HTTP: 响应时间

RTT (Round Trip Time 往返时间): 一个短分组从客户到服务器、然后再返回客户所花费的时间

HTTP response time (per object):

- 一个RTT用于创建TCP连接: 三次握手的前两次所占用的时间
- 一个RTT用于TCP三次握手后的客户向服务器的确认返回+HTTP请求报文, +返回的HTTP响应
- 一个对象/HTML文件的传输时间

非持续连接的HTTP响应时间= 2RTT+ HTML文件的传输时间



Application Layer: 2-24



## 持续连接的 HTTP (HTTP 1.1)

### 非持续连接的 HTTP 的问题:

- 取每个对象需要 2RTTs
- 每个TCP连接的客户端和服务器的操作系统开销
  - 分配TCP缓冲区
  - 保持TCP变量
- 大多数浏览器一般都是同时打开多个并行的连接

### 持续连接的 HTTP (HTTP 1.1):

- 服务器在发送响应后保持连接打开
- 同一客户机/服务器之间的后续HTTP消息通过该打开的连接发送
- 客户机一旦遇到被引用的对象就发送请求
- 所有引用对象只需一个RTT(将响应时间缩短一半)

Application Layer: 2-25

## HTTP请求报文 HTTP Request Message

- 两种类型的HTTP报文: 请求报文 (request), 响应报文 (response)

### HTTP 请求报文:

- ASCII文本 (有一定计算机知识的人都能够阅读它)

请求行(GET, POST, HEAD命令) → GET /index.html HTTP/1.1\r\n

首部行 header lines → 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

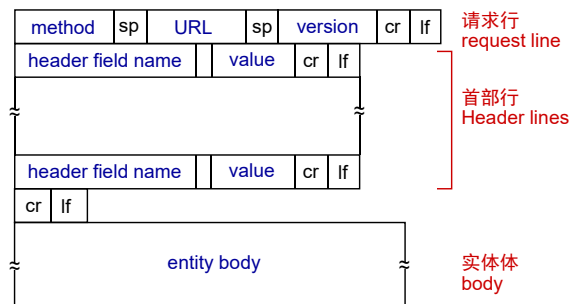
回车、换行表示 报文结束 → \r\n

回车符号  
换行字符

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

Application Layer: 2-26

## HTTP请求报文: 通用格式



Application Layer: 2-27

## HTTP请求报文的类型

### POST method:

- 网页通常包括表单输入
- 用户向服务器发送的数据包含在HTTP POST请求报文的实体体中

### HEAD method:

- 仅用于请求头
- 在HTTP GET请求报文中指定仅请求头, 则将仅返回请求头部分。

### PUT method:

- 将新文件 (对象) 上载到服务器
- 用POST HTTP请求的报文实体体中的内容完全替换指定URL处存在的文件

### GET method (用于向服务器发送数据):

- 在HTTP GET请求消息的URL字段中包含用户数据 (后跟“?”):

[www.somesite.com/animalsearch?monkeys&banana](http://www.somesite.com/animalsearch?monkeys&banana)

Application Layer: 2-28

## HTTP 响应报文

状态行  
(协议状态码状态短语)

首部行

实体体, 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 ...
```

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

Application Layer: 2-29

## HTTP 响应状态码和状态码对应的短语

- 位于（服务器→客户端）响应报文的第一行。
- 样例:
  - 200 OK**
    - 请求成功, 信息在返回的响应报文中
  - 301 Moved Permanently**
    - 被请求的对象被永久转移了, 新的URL定义在响应报文的Location: 客户软件将自动获取新的URL
  - 400 Bad Request**
    - 该请求不能被服务器理解
  - 404 Not Found**
    - 被请求的文档不在服务器上
  - 505 HTTP Version Not Supported**
    - 服务器不支持请求报文使用的HTTP协议版本

Application Layer: 2-30

## HTTP 实操

### 1. 用Telnet 连接测试用的服务器

`telnet gaia.cs.umass.edu 80`

- 打开 TCP 连接到 port 80
- 后续键入的内容将发送到gaia.cs.umass.edu 80 号端口

### 2. 键入一条 http请求报文:

`GET /kurose_ross/interactive/index.php HTTP/1.1`  
`Host: gaia.cs.umass.edu`

将该指令键入后 (按两次回车键), 就将此最短的 (却完整的)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!

(或使用Wireshark查看捕获的HTTP请求/响应)

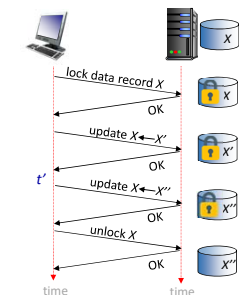
Application Layer: 2-31

## 如何维护用户/服务器状态? - cookies

HTTP请求和响应的交互是  
**无状态的**

- HTTP协议是不保留状态的  
没有多步HTTP报文交换来完成“事务”
  - 不需要客户/服务器跟踪多步交换的“状态”
  - 所有的HTTP请求都是相互独立的
  - 客户/服务器不需要从一个“仅部分完成却未全部完成”的事务中“恢复”

有状态协议: 客户对服务器上存储的X做了两次更改, 若是中间出了问题则取消所有的更改



Q: 如果网络连接或客户端崩溃怎么办?

Application Layer: 2-32



## 如何维护用户/服务器状态? - cookies

Web站点和客户机浏览器使用 *cookie* 来维护事务之间的某些状态

*cookies* 技术有4个组件

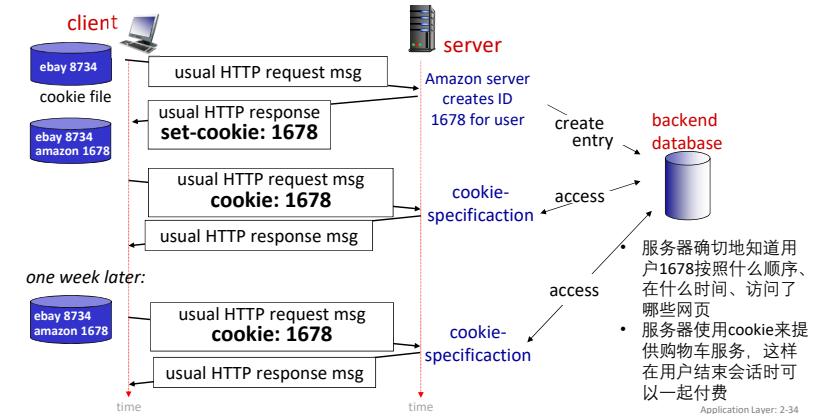
- 1) HTTP响应报文中的一个cookie首部行
- 2) HTTP请求报文中的一个cookie首部行
- 3) 在用户端系统中保留一个cookie文件, 并由用户的浏览器进行管理
- 4) 位于Web站点的一个后端数据库

例子:

- Susan在笔记本电脑上使用浏览器, 第一次访问特定的电子商务网站
- 当初始的HTTP请求到达站点时, 站点创建:
  - 唯一识别码ID(又名“cookie”)
  - 并以ID为索引, 在后端数据库中产生一个表项
- 服务器用一个包含 set-cookie: 1678(如) 首部的HTTP响应报文对Susan的浏览器进行响应
- 从Susan到这个站点的后续HTTP请求将包含cookie ID值, 允许站点“识别”Susan

Application Layer: 2-33

## 如何维护用户/服务器状态? - cookies



Application Layer: 2-34

## 对使用HTTP Cookies 的争论

*cookies* 可以用来做什么:

- 授权
- 购物车
- Recommendations推荐
- 用户会话状态 (Web电子邮件)

*挑战* 如何保持状态

- 协商端点: 在多个事务上维护发送方/接收方的状态
- cookies: HTTP消息携带状态

*cookies* 和隐私: aside

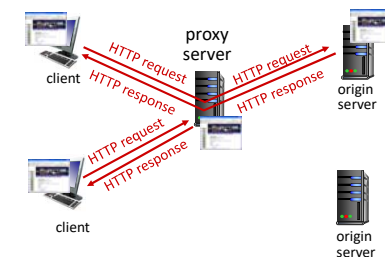
- 允许网站了解更多关于你的信息。
- 第三方持久cookie (跟踪cookie) 允许了对身份信息 (cookie值) 的跨网站跟踪

Application Layer: 2-35

## Web缓存器 (也叫代理服务器) Web caches (proxy servers)

*目标*: 能够代表初始Web服务器来满足HTTP请求的实体

- 用户将浏览器配置为指向Web缓存器
- 浏览器将所有HTTP请求发送到缓存器
  - if 对象在 缓存器中: 缓存器返回对象给浏览器
  - Else 缓存器向初始服务器请求对象, 缓存器接收到的对象, 然后返回对象给客户



Application Layer: 2-36

## Web缓存器(也叫代理服务器) Web caches (proxy servers)

- 缓存器同时充当客户和服务
    - 服务器, 对初始请求客户来说, 它是服务器
    - 客户, 对于初始服务器来说, 它是客户
  - 缓存器通常是由ISP(大学、公司、接入ISP)安装的
- 为什么需要部署Web缓存器?**
- 大大减少客户请求的响应时间
    - 缓存更接近客户
    - 特别是当客户与初始服务器之间的瓶颈带宽远低于客户与Web缓存器之间的瓶颈带宽时
  - 大大减少一个机构的接入链路到因特网的通信量
  - 互联网充满了Web缓存器
    - 使“缺钱”的内容提供商可以更有效地交付内容

Application Layer: 2-37

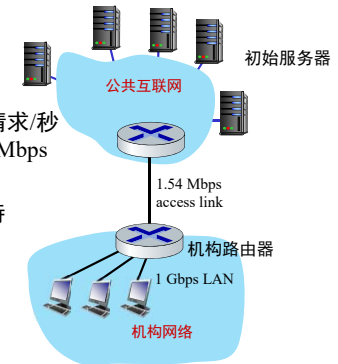
## 缓存的例子

### 假设:

- 接入链路速率: 1.54 Mbps
- 从机构路由器到服务器的RTT: 2秒
- Web对象大小: 100K比特
- 浏览器到初始服务器的平均请求速率: 15个请求/秒
  - 浏览器获得所请求的文件平均速率: 1.50 Mbps

### 性能:

- 局域网的流量强度 = (15个请求/秒 \* 100K比特/请求) / 1G = 0.0015
- 接入链路的流量强度 = (15个请求/秒 \* 100K比特/请求) / 1.54M = 0.974
- 端端延迟 = 互联网时延 + 接入链路时延 (两台路由器之间的时延) + LAN时延  
= 2秒 + 数分钟 + 毫秒



问题: 高利用率的大延迟! Application Layer: 2-38

## 例子的解决方案: 购买更快的接入链接

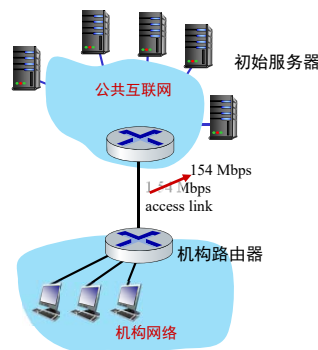
### 假设:

- 接入链接速率: 1.54 Mbps
- 从机构路由器到服务器的RTT: 2秒
- Web对象大小: 100K比特
- 浏览器到原始服务器的平均请求速率: 15个请求/秒
  - 浏览器的平均数据速率: 1.50 Mbps

### 性能:

- 局域网流量强度: 0.0015
- 接入链路流量强度 = 0.97 → .0097
- 平均端到端延迟 = 互联网延迟 + 访问链接延迟 + LAN延迟  
= 2秒 + 分钟 + 毫秒

成本更高: 更快的访问链接 (昂贵!)



Application Layer: 2-39

## 例子的解决方案: 安装Web缓存器

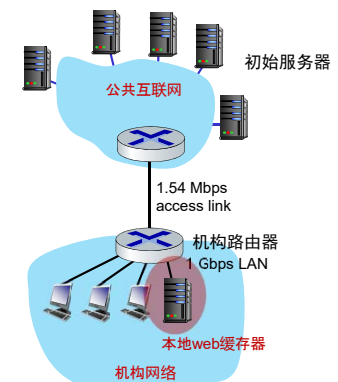
### 假设:

- 接入链接速率: 1.54 Mbps
- 从机构路由器到服务器的RTT: 2秒
- Web对象大小: 100K比特
- 浏览器到起始服务器的平均请求速率: 15个/秒
  - 浏览器的平均数据速率: 1.50 Mbps

### 性能:

- 局域网利用率: ?
  - 访问链接利用率 = ?
  - 平均端到端延迟 = ?
- 如何计算链接利用率, 延迟?

成本低: Web缓存器 (便宜!)

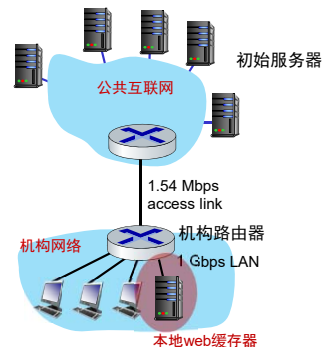


Application Layer: 2-40

## 例子的解决方案：安装Web缓存器

计算访问链接利用率，使用缓存器的端到端延迟：

- 假设缓存命中率为0.4：缓存满足40%的请求，原始满足60%的请求
- 访问链接：60%的请求使用访问链接
- 通过访问链接到浏览器的数据速率  
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
- 利用率 =  $0.9 / 1.54 = .58$
- 平均端到端延迟 =  $0.6 * (\text{来自初始服务器的延迟}) + 0.4 * (\text{缓存满时的延迟})$   
 $= 0.6 (2.01) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$



比使用154 Mbps链路的平均终端延迟低(也更便宜!)

Application Layer: 2-41

## 条件GET方法 Conditional GET

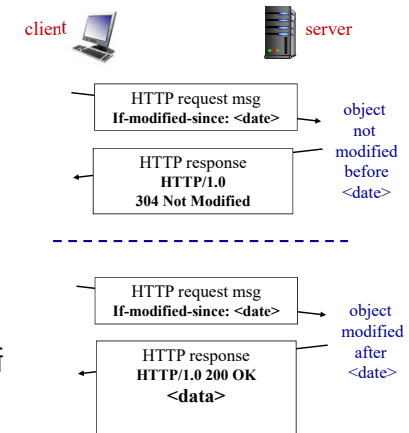
**目标：**若缓存中存储的对象是最新版本，则不发送对象。即允许缓存器证实它存储的对象是最新版本。

- 无对象传输延迟
- 降低链路利用率

■ **缓存：**在HTTP请求中指定缓存副本的日期

If-modified-since: <date>

■ **服务器：**如果缓存的副本是最新的，则响应不包含任何对象：  
HTTP/1.0 304 Not Modified



Application Layer: 2-42

## HTTP/1.1

**目标：**减少多对象HTTP请求中的延迟

**HTTP/1.1:** 通过单个TCP连接完成多个、流水线式GET

- 默认使用持久连接
- 服务器对GET请求进行有序响应 (FCFS: 先到先服务)
- 在服务器到客户的连接中，较小的对象可以在较大的对象之后被阻塞 (head-of-line (HOL) blocking, 队首阻塞)
- 丢失恢复 (重新传输丢失的TCP段) 使对象传输停滞

Application Layer: 2-43

## HTTP/2

**目标：**减少多对象HTTP请求中的延迟

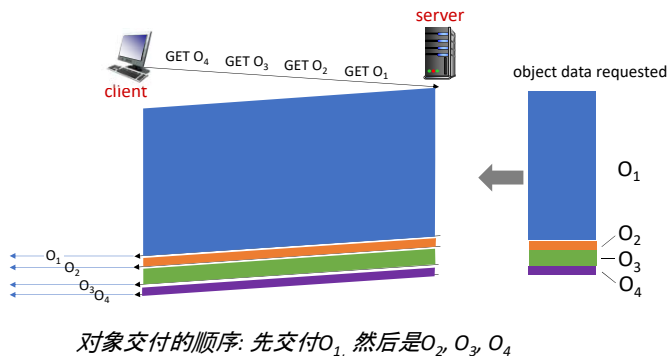
**HTTP/2:** [RFC 7540, 2015]: 服务器在向客户端发送对象时增加了灵活性：

- 根据客户端指定的对象优先级 (不一定是FCFS) 安排请求对象的传输顺序
- 将未请求的对象推送给客户
- 将对象分成帧，安排帧的发送顺序以减轻HOL阻塞
- 方法，状态码和大多数首部行字段与HTTP 1.1相同
- 在报文丢失重传时，暂停对象传输
- HTTP/3:** 错误，拥塞控制，安全性，更多基于UDP的流水线传输

Application Layer: 2-44

## HTTP/2: 缓解队首阻塞问题

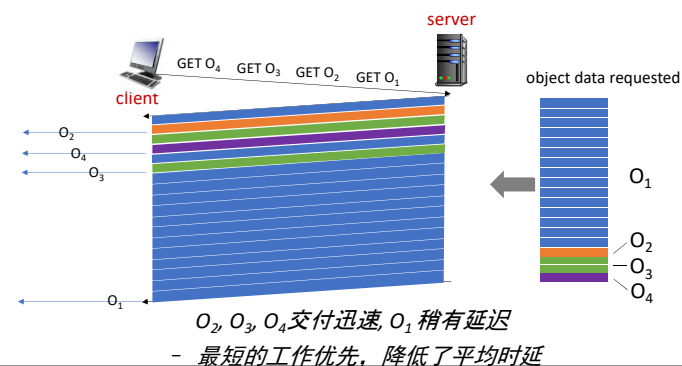
客户请求1个大对象(例如, 1个大视频文件和3个小对象)



Application Layer: 2-45

## HTTP/2: 缓解队首阻塞问题

对象划分成帧, 每个对象的帧交替传输



Application Layer: 2-46

## 应用层

- 网络应用原理
- Web和HTTP
- E-mail, SMTP, IMAP
- 域名系统DNS
- P2P应用程序
- 视频流媒体和内容分发网络
- 编程与UDP和TCP



Application Layer: 2-47

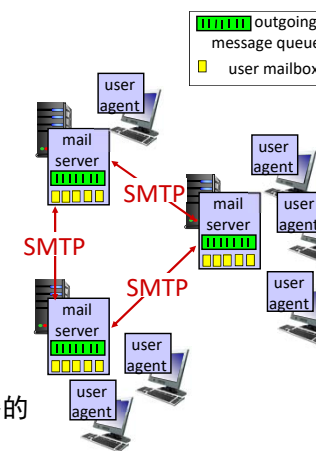
## E-mail

三个主要组成部分:

- 用户代理
- 邮件服务器
- 简单邮件传输协议: SMTP

用户代理

- 又称为“邮件阅读器”
- 撰写, 编辑, 阅读邮件
- 例如Outlook, iPhone邮件客户端
- 向邮件服务器发送邮件、从邮件服务器的邮箱中取得邮件



Application Layer: 2-48

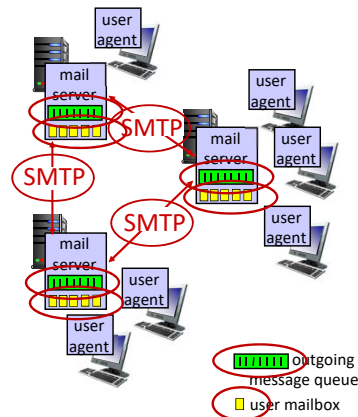
## 邮件服务器 E-mail: mail servers

### 邮件服务器:

- **邮箱** 管理和维护发送给用户的邮件报文
- **报文队列** 外发（待发送）邮件报文的队列

**SMTP协议**: 从发送方的邮件服务器发送邮件报文到接收方的邮件服务器

- **客户端**: 发送邮件服务器
- **服务器**: 接收邮件服务器



Application Layer: 2-49

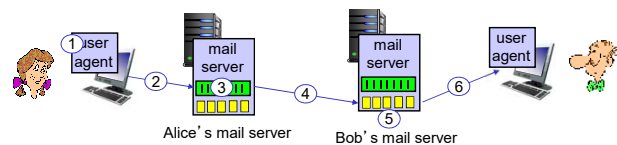
## SMTP: RFC 5321, 初始的RFC可追溯到1982年

- 使用TCP可靠地将电子邮件从客户端（启动连接的邮件服务器）传输到接收方邮件服务器的端口25
- 直接传输：从发送服务器（行为类似于客户）到接收服务器
- 传输的三个阶段
  - 握手（问候）
  - 报文传输
  - 关闭
- 命令/响应的交互（类似HTTP）
  - **命令**: ASCII文本
  - **响应**: 状态码和状态短语
- 规定：包括首部在内，邮件报文的体部分也必须采用7位ASCII

Application Layer: 2-50

## 场景：Alice向Bob发送电子邮件 Scenario: Alice sends e-mail to Bob

- 1) Alice使用她的用户代理编写电子邮件到bob@someschool.edu
- 2) Alice的用户代理将邮件发送到她的邮件服务器；邮件放入邮件队列
- 3) 运行在Alice的邮件服务器上的SMTP客户端打开与Bob邮件服务器的TCP连接
- 4) SMTP客户端通过TCP连接发送Alice的邮件报文
- 5) Bob的邮件服务器将邮件报文放置在Bob的邮箱中
- 6) Bob调用他的用户代理来阅读消息



Application Layer: 2-51

## SMTP交互报文的例子

```

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
  
```

Application Layer: 2-52

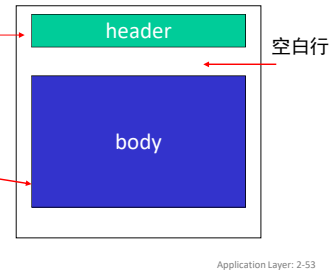
## SMTP协议的报文格式

SMTP: RFC 5321中定义了用于交换电子邮件的协议, RFC5322定义了邮件格式

- 首部行(由RFC5322定义), 例如

- To:
- From:
- Subject:

- 正文: “消息”, 仅ASCII字符



Application Layer: 2-53

## SMTP是一种推协议

与HTTP的比较:

- HTTP: 拉协议
- SMTP: 推协议

相同之处

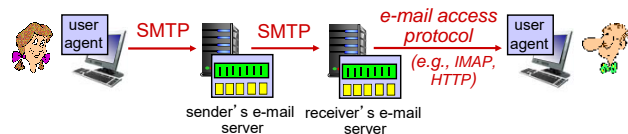
- 两者都具有ASCII命令/响应交互, 都具有状态码
- SMTP 使用持续连接; 持续的HTTP也是用持续连接  
如果发送邮件服务器有几个报文发往同一个接收邮件服务器, SMTP可以通过同一个TCP连接发送所有的报文。

不同之处:

- SMTP: 推  
HTTP: 拉
- SMTP 要求邮件报文 (包括体) 均为7位ASCII  
HTTP不受这种限制
- 对于既包含文本又包含图像的文档  
SMTP将所有报文对象放在一个报文中  
HTTP把每个对象封装在它的自己的HTTP响应报文中

Application Layer: 2-54

## 邮件访问协议



- SMTP:** 将电子邮件传递/存储到收件人的服务器
- 邮件访问协议:** 从服务器取回邮件报文
  - IMAP:** 因特网邮件访问协议[RFC 3501]: 服务器上存储报文的取回、删除、为服务器上存储的邮件创建文件夹
  - POP3:** 第三版的邮局协议[RFC1939]: 服务器上存储报文显示列表、取回、删除
  - HTTP:** Gmail, Hotmail, Yahoo!Mail等在SMTP (发送), IMAP (或POP3-第三版的邮局协议) 之上提供基于Web的界面来检索电子邮件

Application Layer: 2-55

## 应用层

- 网络应用原理
- Web和HTTP
- E-mail, SMTP, IMAP
- 域名系统DNS
- P2P应用程序
- 视频流媒体和内容分发网络
- 编程与UDP和TCP



Application Layer: 2-56



## 域名系统

### DNS: Domain Name System

人: 许多标识符:

- 身份证号码, 姓名

互联网主机, 路由器:

- IP 地址 (32位) -用于寻址数据报
- 主机名, 例如cs.umass.edu, 这些名字便于人记忆和使用

Q: 如何在IP地址和名称之间进行映射?

域名系统:

- 一个由分层的DNS服务器实现的分布式数据库
- 应用层协议: 使得主机能够查询分布式数据库的应用层协议 (地址/名称转换)
  - DNS协议运行在UDP之上, 使用53端口
- DNS服务器通常是运行BIND(Berkeley Internet Name Domain)软件的UNIX机器

注意: 用应用层协议实现的核心Internet功能

Application Layer: 2-57

## DNS: 服务和结构

DNS 服务

- 主机名到IP地址的映射
- 主机别名
  - 一台主机可以拥有1个规范主机名和若干个主机别名
  - 多个主机别名 - 更容易记忆的名字
- 邮件服务器别名
- 负载分配
  - 冗余的Web服务器之间进行负载均衡: 许多IP地址对应一个名字
  - 通常服务器用IP地址集合进行响应, 但在每个回答中循环这些地址次序。

Q: DNS为什么不采用集中式设计?

- 单点故障
- 通信流量
- 远距离集中式数据库
- 维护

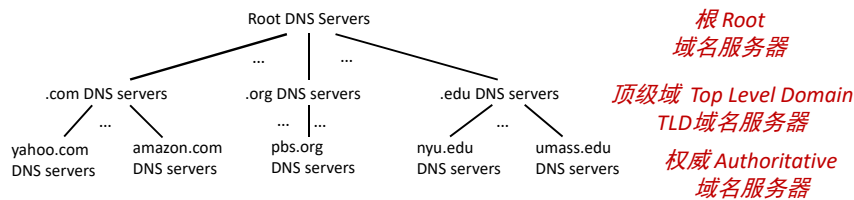
A: 集中式数据库不能扩展!

- 单点故障
- 通信容量600B DNS查询/天
- 远距离集中式数据库
- 维护

Application Layer: 2-58

## DNS:分布式、层次数据库

### DNS: a Distributed, Hierarchical Database



客户端需要www.amazon.com的IP地址:

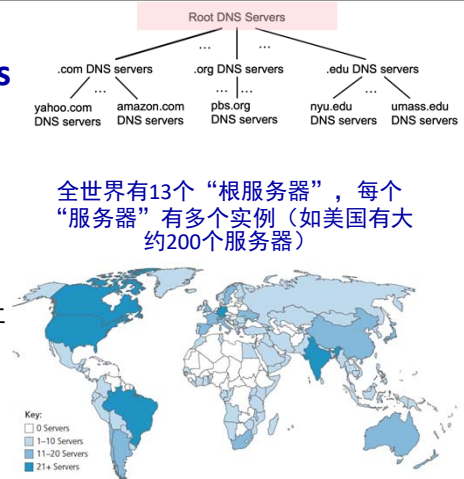
- 客户端查询根域名服务器以查找.com TLD DNS服务器的IP地址
- 客户端查询.com TLD DNS服务器以获取amazon.com权威DNS服务器的IP地址
- 客户端查询amazon.com 权威DNS服务器以获取www.amazon.com的IP地址

Application Layer: 2-59

## DNS: 根域名服务器

### DNS: Root Name Servers

- 官方的, 无法解析名字时应联系的地方
- 非常重要的网络功能
  - 没有它, 互联网将无法运行!
  - DNSSEC -提供安全性 (身份验证和消息完整性)
- ICANN (互联网名称与数字地址分配机构) 管理根DNS域

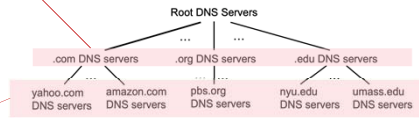


Application Layer: 2-60

## 顶级域名服务器、权威域名服务器 Top-Level Domain, authoritative servers

### 顶级域（TLD）服务器：

- 负责.com, .org, .net, .edu, .aero, .jobs, .museums和所有顶级国家/地区域名，例如：.cn, .uk, .fr, .ca, .jp
- 提供权威DNS服务器的IP地址
- 每个顶级域（如.com）都有TLD服务器或者集群
  - Verisign Global Registry Services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：

- 一个组织自己的DNS服务器，里面的记录将互联网上该组织可公共访问的主机名映射为IP地址
- 也可以付费给某个服务提供商，将这些记录存储在服务商的权威DNS服务器中

Application Layer: 2-61

## 本地域名服务器 Local DNS Name Server

- 不严格属于DNS服务器的层次结构中
- 每个ISP（居民ISP，公司，大学）都有一个本地DNS服务器
  - 也称为“默认名称服务器”
- 主机进行DNS查询时，查询将发送到其本地DNS服务器
  - 本地DNS服务器通常邻近本主机
  - 具有名称到地址转换对（name-to-address translation pairs）的本地缓存（但可能已过期！）
  - 充当代理，将查询转发到DNS服务器的层次结构中

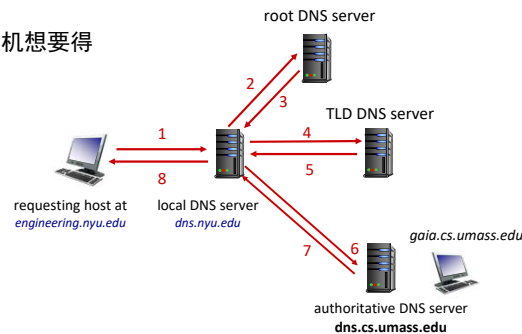
Application Layer: 2-62

## DNS名字解析：迭代查询

示例：engineering.nyu.edu的主机想要得到gaia.cs.umass.edu的IP地址

### 迭代查询：

- 所联系的服务器回复需要联系的服务器名称
- “我不知道这个名字，但是问这个服务器”



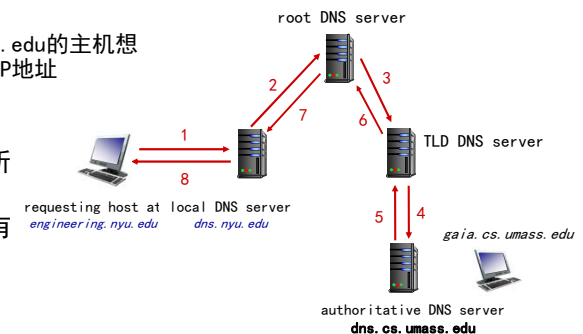
Application Layer: 2-63

## DNS名字解析：递归查询

示例：engineering.nyu.edu的主机想要gaia.cs.umass.edu的IP地址

### 递归查询：

- 给解析的任务交给所联系的域名服务器
- 对高层域名服务器有较重的负载



Application Layer: 2-64

## Caching DNS Information 缓存DNS信息

- 一旦某个DNS服务器接收到一个DNS回答，它将**缓存**该映射
  - 一段时间（TTL，生存时间，通常是2天）后，缓存条目将被丢弃
  - 服务器通常缓存在本地DNS服务器中
    - 因此，根域名服务器并不经常被访问
- 缓存的条目可能已**过期**（**尽力而为的名称到地址的转换！**）
  - 如果主机更改了其IP地址，则在所有TTL都到期之前，可能无法在Internet范围内被知道！
- 更新/通知机制建议的IETF标准
  - RFC 2136

Application Layer: 2-65

## DNS的资源记录

**DNS：存储资源记录（RR）的分布式数据库**

**RR格式：** (name, value, **type**, ttl)

### type=A

- name是主机名
- value是IP地址  
(relay1.bar.foo.com, 145.37.93.126, A)

### type=CNAME

- name是规范主机名的别名
- value是别名为name的主机对应的规范主机名  
(foo.com, relay1.bar.foo.com, CNAME)

### type=NS

- name是域名（例如foo.com）
- value是个知道如何获得该域中主机IP地址的权威DNS服务器的主机名  
(foo.com, dns.foo.com, NS)

### type=MX

- MX允许邮件服务器主机名具有简单的别名；value是与别名为name的SMTP邮件服务器的规范主机名  
(foo.com, mail.bar.foo.com, MX)

Application Layer: 2-66

## DNS 协议报文

**DNS查询和回答报文，具有相同的格式：**

**首部区域：前12个字节**

- 标识符 identification：** #16位 #用于查询，对查询的回答使用相同的 #
- 标志 flags：**
  - 查询或回答标志位，1位
  - 权威域名标志位，1位，当某DNS服务器时所请求名字的权威DNS服务器
  - 希望递归查询标志位，1位
  - 递归可用标志位，1位
  - 4个关于数量的字段，指出首部后的4类数据区域出现的数量

← 2 bytes → ← 2 bytes →	
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

Application Layer: 2-67

## DNS协议报文

**DNS查询和回答消息，有相同的格式：**

**问题区域：** 名字字段：正在被查询的主机的名字  
类型字段：有关该名字的问题类型

**回答区域：** 对所请求名字的资源记录RRs

**权威区域：** 其他权威服务器的记录  
**附加区域：** 其他有帮助的记录，如对于MX请求的回答区域包含了一条资源记录，提供了规范主机名，附加区域提供一个类型A记录，该记录提供了邮件服务器的规范主机名的IP地址。

← 2 bytes → ← 2 bytes →	
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

Application Layer: 2-68

## 如何将某公司的信息放入DNS数据库中

例如：你的创业公司“mycompany.com”

- 在**DNS注册商**处注册名称mycompany.com（DNS注册登记机构，例如Network Solutions）
  - 提供名称，权威名称服务器的IP地址（主要和辅助）
  - 注册商将NS和A，RR插入.com TLD服务器：
    - (mycompany.com, dns1. mycompany.com, NS)
    - (dns1. mycompany.com, 212.212.212.1, A)
- 在本地创建具有IP地址212.212.212.1的权威服务器
  - 输入www. mycompany.com的A记录
  - 为mycompany.com输入MX记录

Application Layer: 2-69

## DNS安全

### DDoS attacks

#### DDoS攻击

- 用流量攻击根服务器
  - 迄今未成功
  - 流量过滤
  - 本地DNS服务器缓存TLD服务器的IP，从而允许根服务器绕过；
    - 但a)缓存被污染 b) 缓存没过期、数据库中的记录却已被删掉了 c) 根服务器的内容被修改了
- 攻击TLD服务器
  - 可能更危险

### 欺骗攻击

- 拦截DNS查询，返回虚假答复
  - 缓存中毒
  - RFC 4033：DNSSEC身份验证服务

Application Layer: 2-70

## 应用层

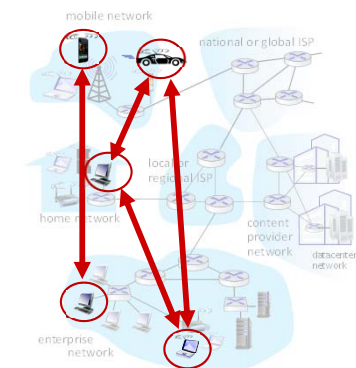
- 网络应用原理
- Web和HTTP
- E-mail, SMTP, IMAP
- 域名系统DNS
- P2P应用程序
- 视频流媒体和内容分发网络
- 编程与UDP和TCP



Application Layer: 2-71

## 对等方到对等方/P2P（Peer-to-peer）体系结构

- 没有永远在线的服务器
- 任意端系统直接通信
- 对等方向其他对等方请求服务，也向其他对等方提供服务
  - 自可扩展性—新的同伴带来新的服务能力和新的服务需求
- 对等方间歇连接并更改IP地址
  - 复杂的管理
- 常见的P2P应用：P2P文件共享（BitTorrent），流式视频（优酷、亚马逊、YouTube），基于IP的语音传输（Skype）

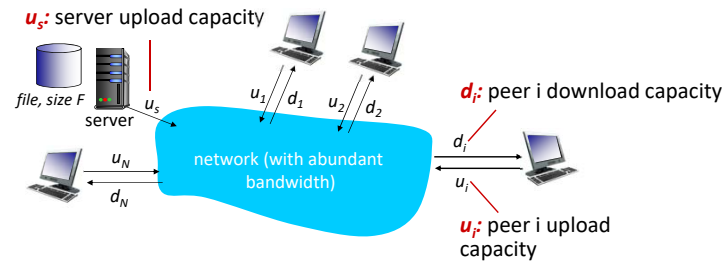


Application Layer: 2-72

## 文件分发：客户端服务器 VS P2P

**Q:** 一台服务器向N个对等方分发文件（大小为F）需要多少时间？

- 对等方上载/下载容量有限



Introduction: 1-73

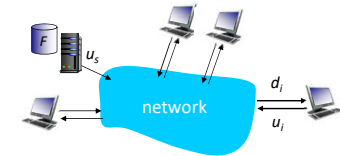
## 文件分发时间：客户-服务器体系结构

■ **服务器传输:** 必须顺序发送（上传）N个文件副本:

- 发送一份副本的时间:  $F/u_s$
- 发送N份副本的时间:  $NF/u_s$

■ **客户端:** 每个客户端必须下载文件副本

- $d_{min}$  = 最小客户端下载速率
- 最小分发时间至少为:  $F/d_{min}$



客户-服务器结构下、分发F到N个客户端的时间

$$D_{C-S} \geq \max\{NF/u_s, F/d_{min}\}$$

N线性增加

Introduction: 1-74

## 文件分发时间：P2P体系结构

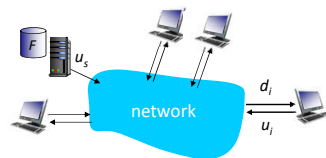
■ **服务器传输:** 必须顺序发送（上传）N个文件副本:

■ 发送一份副本的时间:  $F/u_s$

■ **对等方（客户端）:** 每个对等方必须下载文件副本

- 最小分发时间至少为:  $F/d_{min}$

■ **多个对等方:** 系统必须向N个对等方每个上载F比特，因此总交付NF比特。系统整体的上载能力等于服务器的下载速率加上每个单独的对等方的上载速率  $u_s + \sum u_i$



P2P结构下、分发F到N个对等方的时间

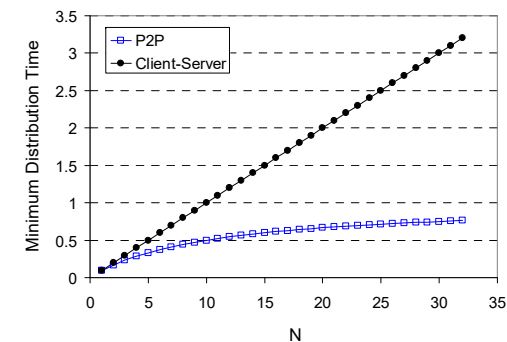
N线性增加

每个对等方都带来了服务能力

75

## 客户-服务器 VS P2P：分发时间比较的例子

客户上传率 =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$ ,  $d_{min} \geq u_s$



Application Layer: 2-76

## P2P文件分发协议：BitTorrent

- 文件划分成等长度的文件块（chunk，.torrent为扩展名，通常为256KB）
- 洪流中的对等方发送/接收文件块

**追踪器 tracker:** 每个洪流的基础设施节点。当一对等方加入时，向追踪器注册自己、并周期性地通知它是否在洪流中。追踪器跟踪参加洪流的对等方

Alice 加入洪流并向追踪器注册自己  
Alice 周期性地从追踪器获得对等方的部分列表；询问她的邻近对等方（建立TCP连接的对等方）他们所具有的块列表，并对她当前还没有的块发出请求；同时向那些请求块的邻居发送块

**洪流 torrent:** 参与一个特定文件分发的所有对等方的集合。  
一个洪流在某个时刻可能有数以千计的对等方

Alice 需要决策2个问题：

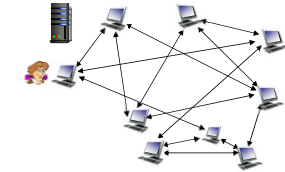
(1) Alice 向她的邻居请求哪些块？

(2) Alice 应向哪些向她请求块的邻居发送块？

Application Layer: 2-77

## P2P文件分发：BitTorrent 的主要机制

- 对等方加入洪流
  - 开始时没有任何块，但随着时间的推移会其他对等方那里累积文件块
  - 向追踪器注册以获取部分对等方的列表，连接到对等方的子集（已经建立TCP连接的称为“邻近对等方” neighbors）
  - 一个对等方的邻近对等方随时而波动
- 在下载时，对等方会将块上传到其他对等方
- 对等方可能会更改与之交换块的对等方
- 流失：**对等方会来来去去
- 一旦对等方拥有了整个文件，它可能（自私地）离开或（以利他的方式）留在洪流中



为什么加入的人越多，  
下载文件的速度越快？

Application Layer: 2-78

## BitTorrent：请求和发送文件块

### 请求（哪些）文件块？

- 在任何给定时间，不同的对等方都有不同的文件块子集
- 定期询问，Alice向每个邻近对等方询问他们拥有的块的列表
- Alice向邻近对等方请求其缺少的数据块，采用**最稀缺优先技术 rarest first**
  - 最稀缺的块**：那些在她的邻居中副本数量最少的块
  - 针对她没有的块、首先请求那些最稀缺的块
  - 最稀缺优先目标是（大致地）均衡每个块在洪流中的副本数目**

### 向谁发送文件块？

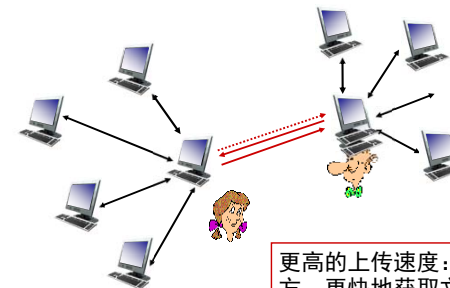
**一报还一报（tit-for-tat）的激励机制**  
或者称“乐观地疏通”邻居

- Alice将数据块发送给当前**以最高速率发送给她数据块的4个邻近对等方**（这4个邻近对等方被称为疏通）
  - 每10秒重新评估前4名的邻居
  - 每30秒：随机选择另一个邻居，开始向其发送块，由于她在发送数据给这个邻居，有可能成为该邻居的前4位上传者；导致这个第五个邻居成为前4位上传者
  - 除了五个对等方（4个+1个试探对等方）之外的其他同伴被Alice choked（阻塞）了（也不能从她那里收到文件块）
- 随机地选择一名新的伴侣并开始与伴侣进行交换，能够趋向于找到彼此协调的速率上传

Application Layer: 2-79

## BitTorrent：一报还一报

- Alice “乐观地疏通” Bob
- Alice成为Bob的前四名提供者之一；Bob回赠
- Bob成为Alice的前四名提供者之一



更高的上传速度：找到更好的对等方，更快地获取文件！

Application Layer: 2-80



## 应用层

- 网络应用原理
- Web和HTTP
- E-mail, SMTP, IMAP
- 域名系统DNS
- P2P应用程序
- 视频流和内容分发网络
- UDP和TCP网络编程



Application Layer: 2-81

## 视频流和内容分发网络CDN：背景

- 流式视频流量：Internet带宽的主要消耗者
  - Netflix, YouTube, 腾讯视频：80%的家庭流量（2020年）
- 挑战：规模 如何支持~1B用户？
  - 单个大视频服务器无法正常工作（为什么？）
- 挑战：异构性 heterogeneity
  - 不同的用户具有不同的功能（例如，有线与移动；带宽较大与带宽较小）
- 解决方案：分布式的、应用层基础架构

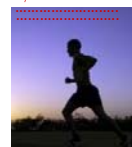


Application Layer: 2-82

## 多媒体：视频

- 视频：以恒定速率显示的图像序列
  - 例如24张图像/秒
- 数字图像：像素阵列
  - 每个像素用位表示
- 编码：在图像内部和图像之间利用冗余以减少用于编码图像所使用的位的数量-对原始的视频进行编码压缩，以去除空间、时间维度的冗余
  - 空间（在图像内）
  - 时间（从一幅图像到下一幅图像）

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），



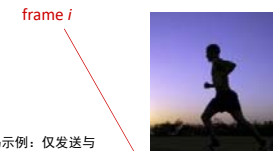
时间编码示例：仅发送与帧的差异，而不是发送i+1的完整帧

Application Layer: 2-83

## 多媒体：视频

- CBR: (固定码率 constant bit rate): 视频编码率固定
- VBR: (可变码率 variable bit rate): 视频编码率随着空间、时间编码量的变化而变化
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps, 700M
  - MPEG2 (DVD) 3-6 Mbps, 4.3G
  - MPEG4 (通常用于Internet, 64Kbps – 12 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），

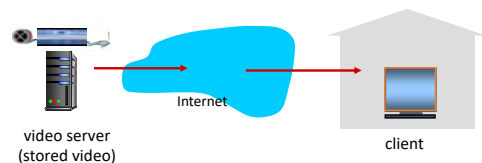


时间编码示例：仅发送与帧的差异，而不是发送i+1的完整帧

Application Layer: 2-84

## 流式视频的主要挑战

一简单场景:

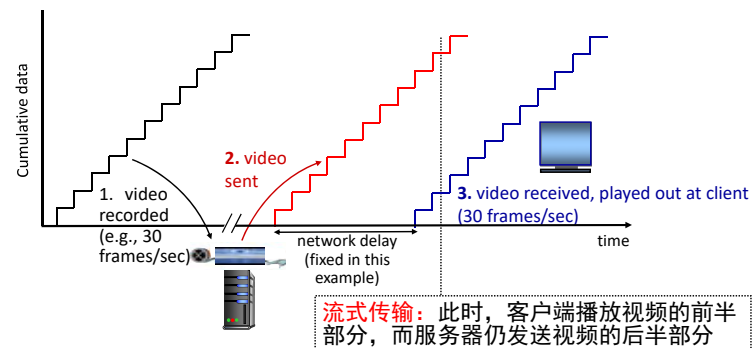


主要挑战:

- 服务器到客户端的带宽将随时间变化。网络拥挤程度不断变化（房间里，接入网，网络核心，视频服务器）
- 拥塞造成的数据包丢失和延迟将影响播放，或导致视频质量下降

Application Layer: 2-85

## 流式视频的传输



Application Layer: 2-86

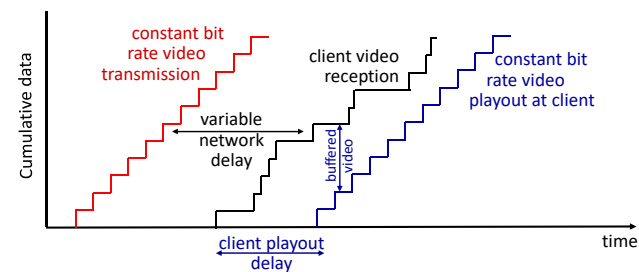
## 流式视频的挑战

- 连续播放的挑战:** 客户端播放开始后, 播放必须与原始时间匹配
  - 但是**网络延迟是可变的** (抖动), 因此需要**客户端缓冲区**来满足播放需求
- 其他挑战:**
  - 客户端交互: 暂停, 快进, 快退, 跳转视频
  - 视频数据包可能会丢失, 如何重新传输



Application Layer: 2-87

## 流式视频的播放: 客户端缓存和播放延迟



- 客户端缓存和播放延迟:** 补偿网络延迟、延迟的抖动

Application Layer: 2-88

## HTTP上的动态自适应流媒体协议 DASH

■ **DASH**: Dynamic Adaptive Streaming over HTTP, HTTP上的动态自适应流媒体协议

### ■ server 服务器:

- 将视频文件分成多个块，长度为几秒
  - 存储的每个块，以不同的速率编码
  - 告示文件 **manifest file**: 提供不同块的 URL
- DASH的manifest文件Media Presentation Description(MPD), 使用XML格式, 对音视频流作了多个维度的划分和相关信息说明
- **minimumUpdatePeriod**: MPD最低限度更新时间
  - **minBufferTime**: 最小缓存时间, 播放器根据@minBufferTime \* @Representation.bandwidth计算出起播的最小缓冲数据

### ■ client: 客户:

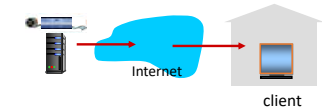
- 定期测量接收带宽, 运行速率决策算法来选择下次请求的块
- 轮询告示文件, 一次请求一个块
  - 在给当前带宽的情况下选择最大比特率的块
  - 可以在不同的时间点选择不同的比特率 (取决于当时的可用带宽)



## HTTP上的动态自适应流媒体协议 DASH

### ■ 客户端的“智慧”：客户确定

- 何时请求块 (这样就不会发生缓冲区不足或溢出)
- 请求哪种编码格式 (在有更多可用带宽时质量更高)
- 在何处请求块 (可以向“接近”客户端或具有高可用带宽的URL服务器请求)



Streaming video = 编码格式 + DASH + 播放缓冲

Application Layer: 2-90

## 内容分发网络 CDNs

### Content Distribution Networks (CDNs)

- 挑战: 如何将内容 (从数百万个视频中选择) 流式传输到成千上万的 *同时* 的用户?
- 选项1: 单个“大型服务器”
  - 单点故障
  - 网络拥塞点
  - 距离客户端远
  - 通过链接发送视频的多个副本

这种方案很简单: 但 **无法扩展**

Application Layer: 2-91

## 内容分发网络CDNs

- 挑战: 如何将内容 (从数百万个视频中选择) 流式传输到成千上万的 *同时* 的用户?
- 选项2: 在多个地理分布的站点 (CDN) 上存储/提供多个视频副本
  - 深入 enter deep: 将CDN服务器推入许多接入网
    - 贴近用户, Akamai公司
      - 1998年9月开始运作, 首创CDN技术, 一套突破性的运算法则, 用于在网络服务器所组成的大型网络中智能安排路由和复制内容, 而且不需要依赖如今站点拥有者所使用的中央服务器-用智力-智能化的内容分发网络CDN结束World Wide Wait(世界一起等待)的尴尬局面
      - 在120多个国家/地区部署了24万台服务器 (2015年), 全世界的90%的互联网用户经过1跳就可以找到一台Akamai服务器
    - 邀请做客 bring home: 原则是在少量 (例如10个) 关键位置建造大集群来邀请到ISP做客, 即将集群放置在网络交换点的。这样产生较低的维护和管理开销, 但对端用户有较高延迟和较低吞吐量
      - Limelight公司: 拥有全球光纤网络的CDN公司

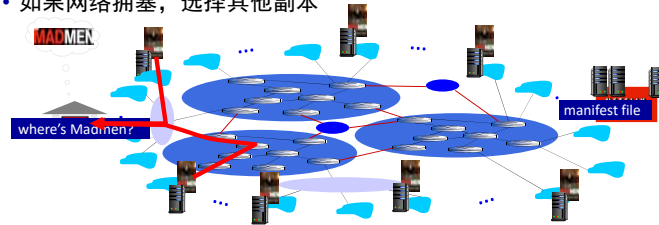


Application Layer: 2-92

## 内容分发网络CDNs

- CDN: 将内容副本存储在CDN节点上
  - 例如 Netflix、美国首屈一指的在线电影和TV节目服务提供商, 存储MadMen的副本
- CDN订阅者从CDN请求内容
  - 直接到附近的副本, 下载内容
  - 如果网络拥塞, 选择其他副本

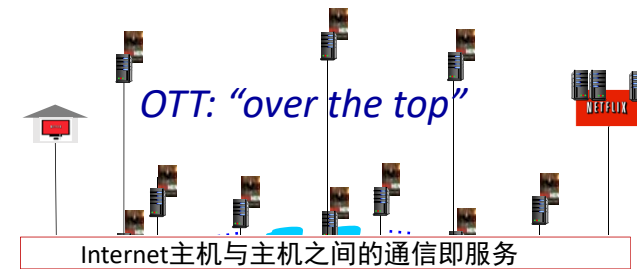
Akamai: 数十万服务器、数百万个集群, 分布在90多个国家800多个城市、1000多个云、2500多个节点上, 每天有万亿+的请求。



Application Layer: 2-93

## 内容分发网络CDNs

peak load: 7 million viewers, 2 T bytes



**OTT挑战:** 如何应对拥挤的互联网

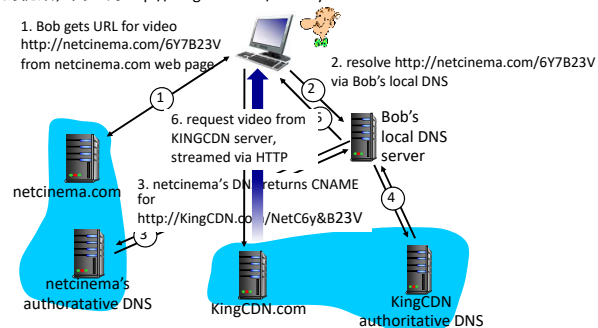
- 从哪个CDN节点检索内容?
- 观众在网络拥塞情况下的行为?
- CDN节点中应放置哪些内容?

Application Layer: 2-94

## CDN的内容访问：近距离策略

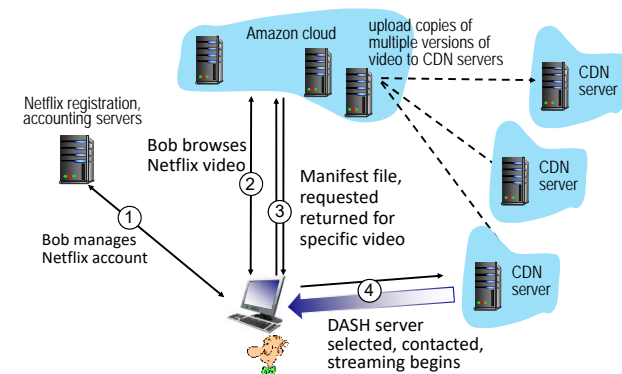
Bob (客户) 请求视频 <http://netcinema.com/6Y7B23V>

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



Application Layer: 2-95

## Case study: Netflix



Application Layer: 2-96

## 应用层

- 网络应用原理
- Web和HTTP
- E-mail, SMTP, IMAP
- 域名系统DNS
- P2P应用程序
- 视频流媒体和内容分发网络
- UDP和TCP网络编程

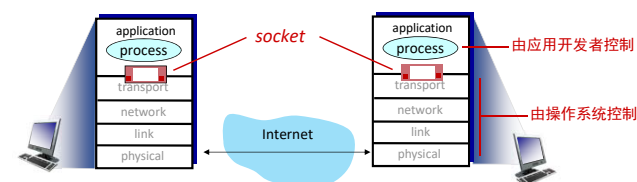


Application Layer: 2-97

## 套接字（Socket）编程

**目标：**学习如何编写使用套接字进行通信的客户端/服务器应用程序

**套接字：**应用程序进程与端到端传输协议之间的门



Application Layer: 2-98

## 套接字编程

两种传输服务的两种套接字类型：

- **UDP:** 不可靠的数据报
- **TCP:** 可靠，面向字节流

客户-服务器应用程序的例子：

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. 客户端接收修改后的数据并在其屏幕上显示行

Application Layer: 2-99

## 使用UDP的套接字编程

**UDP:** 客户端与服务器之间没有“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:** 传输的数据可能会丢失或乱序接收

**使用UDP的应用程序：**

- 在客户端和服务器之间提供不可靠的数据报传输

Application Layer: 2-100

## UDP客户端/服务器套接字交互




server (running on serverIP)

create socket, port= x:  
serverSocket =  
socket(AF\_INET, SOCK\_DGRAM)

从serverSocket  
读取数据报

write reply to  
serverSocket,  
并指定客户地址, 端口号

client 

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

Application Layer: 2-101

## UDP客户端程序

### Python UDPClient

包括Python的套接字库 → from socket import \*

为服务器创建UDP套接字 → serverName = 'hostname'  
serverPort = 12000  
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))

将套接字中的回复字符读入  
到字符串modifiedMessage中 → modifiedMessage, serverAddress =  
clientSocket.recvfrom(2048)

打印出接收到的字符串并关闭套接字 → print modifiedMessage.decode()  
clientSocket.close()

Application Layer: 2-102

## UDP服务器程序

### Python UDPServer

创建UDP套接字 → from socket import \*

将套接字绑定到本地端口号12000 → serverPort = 12000  
serverSocket = socket(AF\_INET, SOCK\_DGRAM)  
serverSocket.bind(('', serverPort))  
print ( "The server is ready to receive")

永远循环 → while True:

从UDP套接字读入消息, 获取客户端的地址  
(客户端IP和端口) → message, clientAddress = serverSocket.recvfrom(2048)  
modifiedMessage = message.decode().upper()

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

Application Layer: 2-103

## 使用TCP的套接字编程

### 客户端必须连接服务器

- 服务器进程必须先运行
- 服务器必须已经创建了允许客户端联系的套接字 (门)

### 客户端通过以下方式联系服务器:

- 创建TCP套接字, 指定服务器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
  - 允许服务器与多个客户端通话
  - 用于区分客户端的源端口号 (第3章会有更详细的讲解)

### 注意

TCP 在客户端和服务端之间提供可靠, 有序的字节流传输 (可以想象为“管道”)

Application Layer: 2-104



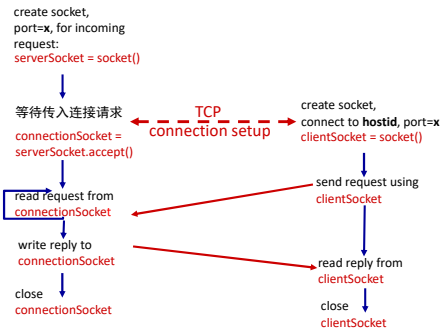
## TCP客户端/服务器套接字交互



server (running on hostid)



client



## TCP客户端程序

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

为服务器创建TCP套接字，远端端口12000

无需附加服务器名称，端口

Application Layer: 2-106

## TCP服务器程序

Q: 分析下TCP和UDP 客户端代码和服务器端代码的不同之处  
UDP P107-108; TCP P111-112

### 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 True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
  
```

创建TCP欢迎套接字-等待连接的套接字. 等待连接

服务器开始侦听传入的TCP请求

永远循环

服务器使用accept()等待传入请求, 并返回新的套接字

从套接字读取字节 (与UDP不同)

关闭与该客户端的套接字 (但不关闭服务器等待连接的套接字)

Application Layer: 2-107

## 第二章：总结

我们对应用层的学习现已完成！

- 网络应用程序的体系结构
  - client-server
  - P2P
- 应用服务要求：
  - 可靠性、带宽、延迟
- 互联网传输服务模式
  - 面向连接、可靠传输：TCP
  - 不可靠、数据报传输：UDP

- 具体协议：
  - HTTP
  - SMTP, IMAP
  - DNS
  - P2P: BitTorrent
- 视频流, CDN
- 套接字编程：
  - TCP、UDP sockets

## 第二章：总结

最重要的是：了解协议！

- 典型的请求/回复报文交换：

- 客户端请求信息或服务
- 服务器响应数据、状态码

- 报文格式：

- *Headers 报文头*：提供有关数据信息的字段
- *Data 数据*：正在传达的信息（有效载荷）

重要主题：

- 集中与分布
- 无状态与有状态
- 可扩展性
- 可靠与不可靠的报文传输
- “网络边缘的复杂性”