Chapter 2
Application Layer

计算机网络:自顶向下方法（第6版）
J.F.Kurose，K.W.Ross著,陈鸣译, 机械工业出版社,2014.

Computer Networking: A Top-Down Approach(Sixth Edition)
J.F.Kurose，K.W.Ross,2012.

本PPT改编自英文版教材附带的PPT。

Chapter 2: Application layer

Chapter 2: Application Layer

Our goals:

- 网络应用协议的概念/实例
  - 传输层服务模型
  - client-server paradigm
  - peer-to-peer paradigm
- 通过常见的应用层协议学习
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- 网络应用编程
  - socket API

一些网络应用

- 电子邮件e-mail
- web
- 即时消息
- 远程登录
- P2P 文件共享
- 多用户网络游戏
- 流媒体：视屏点播
- IP电话
- 流媒体：视屏会议
- 网格计算

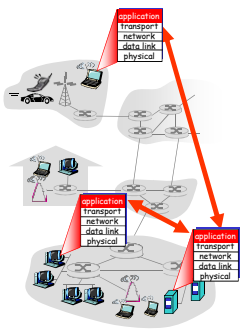
构造一个网络应用

开发一个应用

- 运行于不同 *end systems*
- 通过网络通信
- e.g., web server software communicates with browser software

不需要关心网络核心的设备

- 网络核心设备不运行用户应用
- *end systems* 允许快速的应用开发 、部署、传播

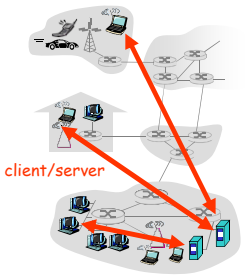


Chapter 2: Application layer

应用架构

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server 架构



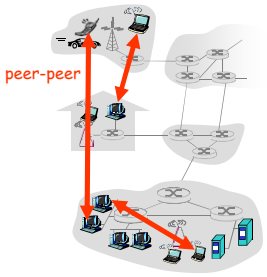
server:
- always-on 主机
- 永久IP address
- 可扩展：服务器集群

clients:
- 与服务器通信
- 可能是间歇性的连接
- 可能是动态的IP地址
- 通常相互之间不直接通信

纯 P2P 架构

- 没有 always-on 服务器
- 任意端节点主机间可相互通信
- 端节点间歇性的连接、相互通信，具有动态的IP地址

Highly scalable but difficult to manage



client-server和 P2P混合架构

Skype
- voice-over-IP P2P 应用
- 中央服务器: finding address of remote party:
- client-client 连接: direct (not through server)

Instant messaging (QQ)
- 用户之间的聊天是P2P
- 中央服务器:
- client presence detection/location
  - User上线后在中央服务器注册IP地址
  - User 联系中央服务器获得好友的IP地址

进程通信

Process: 运行于某个主机上的程序.
- 进程间通信：
- 相同主机上的进程间通信 inter-process communication (OS).

- 消息：
- 不同主机上的进程间通过交换messages通信

Client process:
发起通信的进程
Server process:
等待被访问的进程

- Note:
- P2P 架构的应用程序同时有
- client processes
- & server processes

进程寻址

- 进程接收消息，需要一个标识符
- 主机有32-bit IP 地址

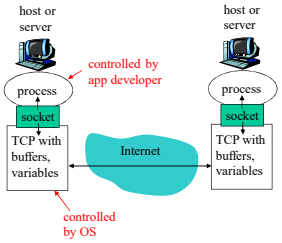
- Q: 有IP 地址后该主机上的进程就可以标识自己，进行通信了吗?

- Answer: No, 同一个主机具有很多进程。

- 标识符包括
- IP address
- 与主机上进程相应的端口号port numbers.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- More on this later

套接字Sockets

- 进程 sends/receives messages 通过socket进行
- socket 类比于一扇门
  - sending process 把message 扔出门口
  - sending process 依赖于传输层结构将message 发送到接收的进程
    - 其实就是函数调用
- API: (1) 可以选择传输层协议; (2) 指定某些参数。



应用层协议的定义

- 交换的消息类型
  eg, request & response messages
- 消息类型的语法
  fields inmessages & how fields are delineated
- 字段的语义
  meaning of information in fields
- 进发送&响应消息的规则
  Rules for when and how processes send & respond to messages

公有协议
Public-domain protocols:
- 在RFCs中定义
- 允许互操作
- e.g., HTTP, SMTP

私有协议
Proprietary protocols:
- e.g., Skype，QQ

应用所需要的传输层服务?

数据的丢失Data loss
- 一些应用(e.g., audio) 能够忍受数据的丢失
- 某些应用(e.g., telnet,file transfer) 要求 100% 可靠的数据传输

时效性Timing
- 一些应用(e.g., Internet telephony, interactive games) 要求低延迟

吞吐率Throughput
- 一些应用(e.g., multimedia) 要求最小的吞吐率带宽
- 某些应用("elastic apps") 随便多少带宽都可以工作

安全性Security
- Encryption, data integrity, …

一些应用对传输服务的要求

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet传输协议服务

TCP service:
- *面向连接connection-oriented:* setup required between client and server processes
- *可靠地reliable transport：* between sending and receiving process
- *流控flow control:* sender won't overwhelm receiver
- *拥塞控制congestion control:* throttle sender when network overloaded
- *不提供:* timing, minimum bandwidth guarantees

UDP service:
- *不可靠的数据传输* unreliable data transfer between sending and receiving process
- *不提供:* connection setup, reliability, flow control, c ongestion control, timing, bandwidth guarantee

Q: why bother? Why is there a UDP?

Internet应用：传输协议

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Chapter 2: Application layer

- 2.1 网络应用的原理
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 电子邮件
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P 应用
- 2.7 TCP Socket 编程
- 2.8 UDP Socket 编程

Web and HTTP

- 一些web的术语
- Web page 由 objects (对象)组成
- Object 可以是HTML 文件, JPEG 图片, Java applet, 声音文件,...
- Web page 一般是基 HTML文件，包含很多索引对象
- 每个object 由URL寻址
- Example URL:

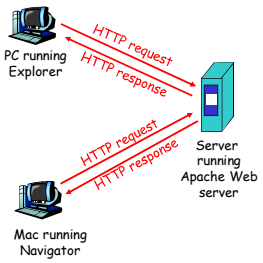
www.someschool.edu/someDept/pic.gif

host name | path name

HTTP概览

HTTP: 超文本传输协议
hypertext transfer protocol

- Web应用层协议
- client/server model
  - *client:* 浏览器
  - 发送请求，接收，显示(渲染) Web objects
  - *server:* Web 服务器
  - 回复针对请求的相应



HTTP 概览 (continued)

使用 TCP:

- Client初始化TCP 连接 (生成socket) 至服务器，端口 80
- server 接受client的TCP 连接请求
- HTTP 消息(应用层协议消息) 在browser (HTTP client) 和 Web server (HTTP server) 之间交换
- TCP 连接关闭

HTTP 是无状态的

- server 不维护过去 client 的requests

aside

一个维护状态的协议很复杂!

- 必须维护过去的历史(state)
- 如果server/client 崩溃, 它们的状态"state" 可能不一致, 就需要恢复

HTTP连接

非持久连接HTTP

- 一次 TCP 连接最多发送一个object.
- HTTP 1.0

持久连接HTTP

- 一次 TCP 连接可以发送多个object.
- HTTP 1.1
- 大部分浏览器默认支持!

非持久连接HTTP

假设用户输入URL

www.someSchool.edu/someDepartment/home.index (包含text, 10 个jpeg 图片)

1a. HTTP client 初始化 TCP 连接至 HTTP server (process) www.someSchool.edu on port 80

1b. 主机www.someSchool.edu HTTP server 在80端口 等待TCP 连接. "accepts" 连接, 通知client

2. HTTP client 发送 HTTP *request message*(包含 URL) 进入 TCP socket连接. Message 表示客户需要object someDepartment/home.index

3. HTTP server 接收request message, 封装*response message* (包含object), 发送该message 进入socket

time

非持久性 HTTP (cont.)

4. HTTP server 关闭 TCP 连接.

5. HTTP client 接收 response message (包含html文件), 显示 html. 解析(Parsing) html 文件, 发现 10 索引的 jpeg objects

6. 为每个 jpeg objects对象重复步骤。10次

time

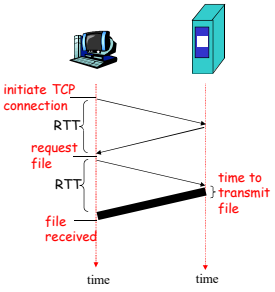
非持久性HTTP: Response time

RTT定义Definition of RTT: client端一个小数据报从发送到server， 至接收到响应的时间.

响应时间Response time:

- 一个 RTT 初始化TCP 连接
- 一个RTT： HTTP request 发送到 HTTP response 返回
- 文件传输时间

total = 2RTT+transmit time



持久性 HTTP

非持久性HTTP 不足:

- 每个 object需要 2 RTTs
- 浏览器只有使用并行的 TCP 连接来获取索引的 web objects
- 每个TCP连接都增加了OS的负担

持久 HTTP

- server 发送response后不是马上关闭连接
- 相同client/server 之间后续 HTTP messages 通过该 open connection交换
- client 在解析到页面中索引的object后马上发送requests
- 一个object一RTT时间

HTTP 请求消息

- 两种类型 HTTP 消息: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

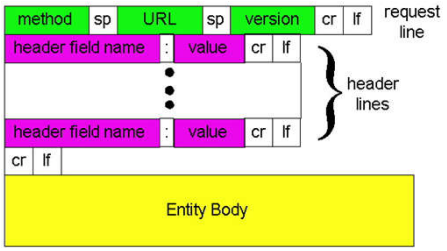
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

HTTP请求消息格式：通常的格式



使用 input上载

Post method:

- Web page 通常包含 form input
- Input 包含在request message中entity body

URL method:

- 使用 GET method
- Input包含在request line中的URL 字段:

www.somesite.com/animalsearch?monkeys&banana

方法类型

HTTP/1.0
- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1
- GET, POST, HEAD
- PUT
  - 在entity body 包含上传文件路径在URL field指定
- DELETE
  - 删除 URL field指定的文件

HTTP 响应消息

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
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 ...
```


HTTP 响应状态码

在server->client response message中的第一行.
A few sample codes:

200 OK
- request succeeded, requested object later in this message

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

400 Bad Request
- request message not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

HTTP客户端测试

1. Telnet 登录到某个Web server:

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

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

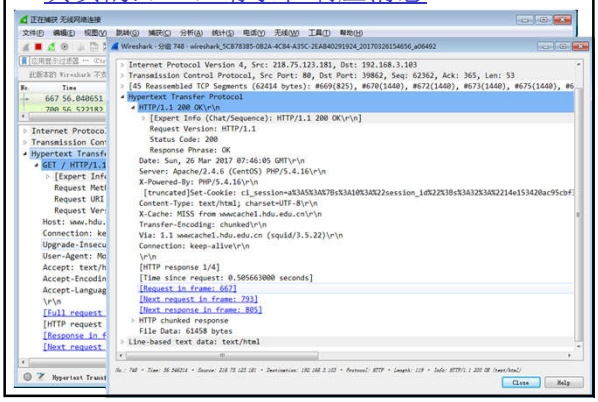
2. 输入GET HTTP request:

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

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. 查看HTTP server的响应消息!

真实的HTTP 请求和响应消息



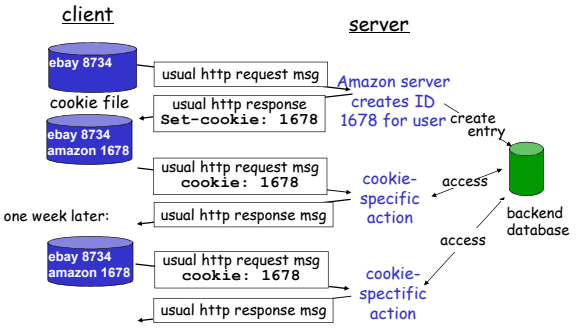
用户端状态: cookies

Cookies使用非常广泛！

四个组成部分:
1) HTTP *响应消息头部*
2) HTTP *请求消息头部*
3) 保留在客户端的cookie文件, 由用户浏览器管理
4) Web 站点后台数据库

Example:
- Susan 通常使用PC上Internet
- 第一次访问某个e-commerce 网站
- 初始化HTTP requests到达该站点后, 站点生成:
  - unique ID
  - 后台数据库与ID相应的entry

Cookies: 保持状态 (cont.)



Cookies (continued)

aside

Cookies 和隐私:

- 站点能够通过cookies 了解很多用户的情况
- 例如name 、 e-mail 等等

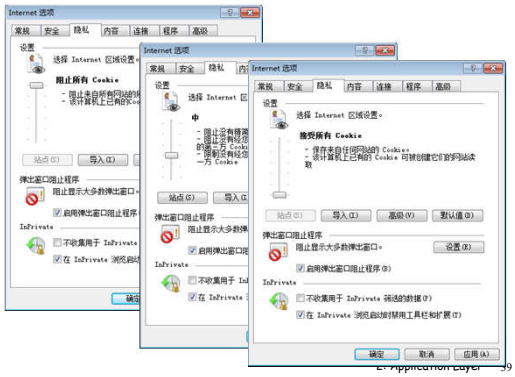
用途:

- 认证(authorization)
- 购物车(shopping carts)
- 推荐(recommendations)
- 用户会话状态(Web e-mail)

保持状态

- protocol endpoints: 在多个事务(transactions)的过程中，在sender/receiver之间维护状态
- cookies: http messages carry state

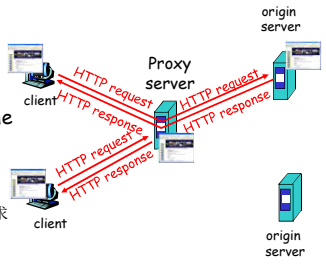
IE隐私设置



Web caches (proxy server代理服务器)

Goal: 不访问origin server 而满足client request

- 用户可以设置browser: Web 通过cache访问
- browser 会发送所有HTTP requests 给cache
  - Cache命中: cache 返回 object
  - 否则: cache向origin server请求object, 返回object，同时缓存



代理缓存

- cache 可以运行在 客户端和服务器端
- 通常cache 由ISP部署 (university, company, residential ISP)

为什么使用Web caching?

- 减少客户端的请求响应时间
- 减少瓶颈线路流量
- Internet 广泛使用caches:

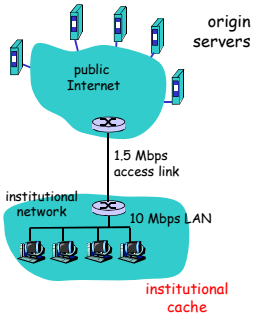
Caching example

Assumptions

- object平均大小 = 100K bits
- 浏览器平均访问率= 15/sec
- institutional router 至origin server 的往返延迟= 2 sec

Consequences

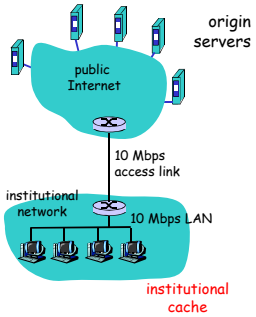
- LAN 利用率= 15%
- access link 利用率= 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds



Caching example (cont)

possible solution
- 增加access link 带宽至10 Mbps

consequence
- LAN 利用率= 15%
- access link 利用率= 15%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + msecs
- often a costly upgrade

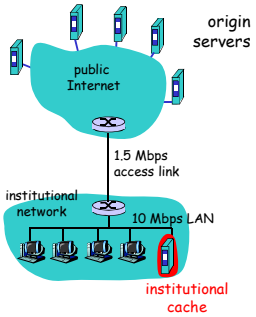


Caching example (cont)

possible solution:
安装cache
- 假设 hit rate 为0.4

consequence
- 40% 访问将在本地得到满足
- 60% 请求由origin server返回
- access link 利用率减至60%, 致使delays 降低(say 10 msec)

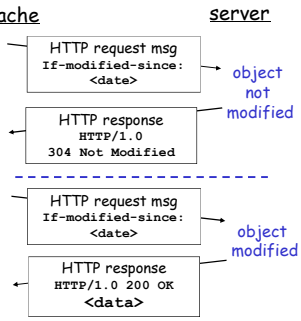
- total avg delay
- = Internet delay + access delay + LAN delay
- = .6*(2.01) secs + 4*milliseconds
- < 1.4 secs



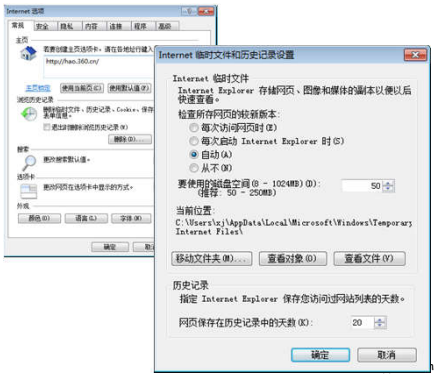
条件取 GET

- Goal: 如果客户端有最新的拷贝就不需要从服务器上取
- cache: 通过在HTTP request `If-modified-since: <date>` 字段指定本地拷贝的生成时间
- server: 如果cache拷贝是最新的(up-to-date)，那么response 不包含任何内容:
- 使用返回状态:

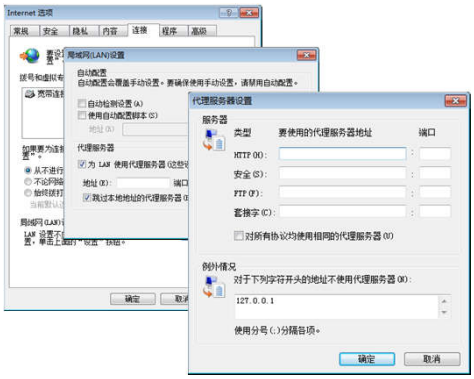
`HTTP/1.0 304 Not Modified`



IE本地临时文件设置



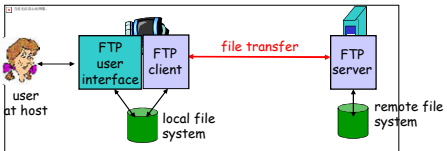
IE代理设置



Chapter 2: Application layer

- 2.1 网络应用的原理
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 电子邮件
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P 应用
- 2.7 TCP Socket 编程
- 2.8 UDP Socket 编程

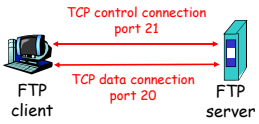
FTP: 文件传输协议



- 客户机与服务器之间传送文件
- client/server model
  - *client:* 初始化传输 (either to/from remote)
  - *server:* 远处主机
- ftp: RFC 959
- ftp server: port 21

FTP: 控制/传输连接

- FTP 客户在 21号端口与服务器连接, 使用TCP
- client 在控制连接被授权
- client 通过控制连接浏览服务器目录.
- server 接到文件传输命令，打开 第2个TCP 连接
- 文件传输完毕后，关闭传输连接.



- FTP server 维护 "state": 当前目录，认证状态
- 控制连接使用: "带外传输"

server 打开单独的TCP 传输文件 .

FTP 命令, 响应

Sample commands:

- 在控制信道发送 ASCII 格式命令
- `USER username`
- `PASS password`
- `LIST` 返回当前目录文件列表
- `RETR filename` 下载指定文件
- `STOR filename` 上传文件至服务器当前目录

Sample return codes

- 状态码和简短说明 (类似HTTP)
- `331 Username OK, password required`
- `125 data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`

FTP 客户端命令

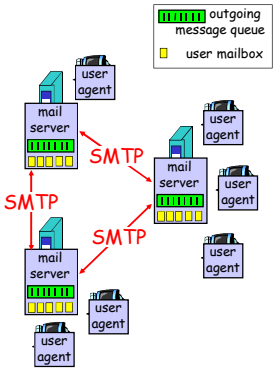
Chapter 2: Application layer

- 2.1 网络应用的原理
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 电子邮件
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P 应用
- 2.7 TCP Socket 编程
- 2.8 UDP Socket 编程

电子邮件

三个组成部分:
- 用户代理
- 邮件服务器
- 简单邮件传输协议: SMTP

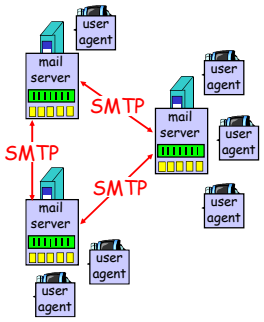
User Agent
- a.k.a. "mail reader"
- 书写, 编辑, 阅读邮件信息
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- 外出及进来邮件都保留在服务器



电子邮件: mail servers

Mail Servers
- 邮箱: mailbox
  保留用户邮件
- 邮件队列: message queue
  等待发送的邮件

SMTP protocol
服务器之间交换邮件的协议
- client: 发送邮件的服务器或客户端
- "server": 接收邮件的服务器

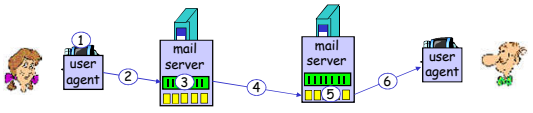


电子邮件: SMTP [RFC 2821]

- 使用 TCP 从客户端可靠的传输邮件信息到服务器,
- 使用端口 25
- 直接传送: 发送服务器直接发送邮件至接收服务器
  - 传输的三个阶段
  - 握手 handshaking (greeting)
  - 传输 transfer of messages
  - 关闭 closure
- command/response 交互模式
  - 命令 commands: ASCII 文本
  - 响应 response: 状态码和简短说明
- 消息是7-bit ASCII字符!

例子: 张三发送消息给李四

1) 张三使用 UA 发送邮件信息到 lisi@someschool.edu
2) 张三的UA 发送邮件到他的邮件服务器;
   消息放在服务器的消息队列中
3) SMTP 客户端打开TCP 连接到李四 的邮件服务器
4) SMTP 客户端通过TCP连接发送张三的邮件给李四的服务器
5) 李四的服务器将邮件放在李四的信箱中
6) 李四通过他的UA阅读信箱中的邮件



SMTP 交互测试:

- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

以上过程让我们不使用客户端或浏览器就能发送邮件!

简单SMTP 交互

```
>telnet smtp.163.com 25
S: 220 163.com
C: HELO localhost
S: 250 OK
C: AUTH LOGIN    //使用身份认证登陆指令
S: 334 dXNlcm5hbWU6
C: cmVkc29zMw== //输入已经base64_encode()的用户名
S: 334 UGFzc3dvcmQ6
C: MbM2MDQ3NQ== //输入已经base64_encode()的密码
S: 235 Authentication successful
C: MAIL FROM: <alice@163.com>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <alice@163.com>
S: 250 alice@163.com ... 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
```


SMTP: 后话

- SMTP 使用持久连接
- SMTP 消息采用 (header & body) 7-bit ASCII编码
- SMTP server 使用 CRLF.CRLF 判断消息的结束

Comparison with HTTP:

- HTTP: 拉 pull
- SMTP: 推 push
- 都使用ASCII command/response 进行交互, status codes
- HTTP:
- 每个 object 都有响应消息
- SMTP:
- 多个objects 在同一个消息中发送

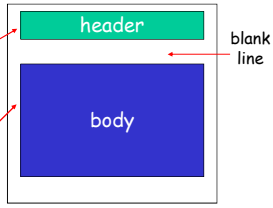
邮件消息格式

SMTP: 交换电子邮件的协议

RFC 822: 消息的标准文本格式:

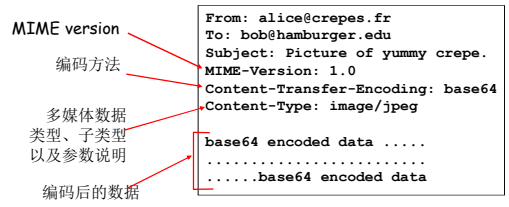
- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP commands!
- body
  - the "message", ASCII characters only

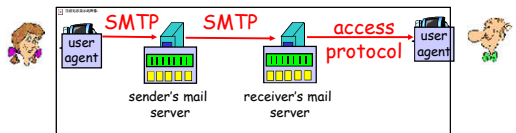


消息格式:多用途电子邮件扩展

- MIME: 多用途电子邮件扩展, RFC 2045, 2056
- 消息头部的附加行说明MIME 内容类型



邮件访问协议



- SMTP: 发送邮件到接收者服务器
- Mail access protocol: 从邮件服务器获取邮件
  - POP: Post Office Protocol [RFC 1939]
    - 认证 (agent <-->server) 和下载
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - 更多的特征
    - 能够在服务器上对邮件做某些操作
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

认证阶段

- client 命令:
  - **user:** username
  - **pass:** password
- server 响应
  - **+OK**
  - **-ERR**

事务阶段

client:

- **list:** 列出消息数量
- **retr:** 根据消息序号检索消息
- **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
```

POP3 (more) and IMAP

More about POP3
- 前例使用 “download and delete” 模式.
- 李四阅读邮件后邮件中别的UA中不能访稳
- “Download-and-keep”: 模式允许不同UA访问邮件
- POP3 是无状态的协议

IMAP
- 所有信息保留在一个地方: server
- 所有用户通过文件夹组织消息
- IMAP 中sessions中保持用户状态：
  - 文件夹的名字
  - 文件夹和消息ID的对应关系

Chapter 2: Application layer

DNS: Domain Name System 域名系统

人类: 身份标识:
- 身份证, 名字, 护照 #

Internet 主机, 路由:
- IP 地址 (32 bit)
- - 用于数据报寻址
- “域名”, e.g., www.yahoo.com
- - 给用户识别

Q: 如何完成IP 地址和域名的映射呢？

Domain Name System:
- *分布式数据库*
- -具层次结构的多个域名服务器
- *应用层协议*
- - 主机, 路由,域名服务器

互相通信完成名字解析
(*resolve*， 地址/域名 映射)

注意:
- 是一个应用层部署的核心协议
- 协议的复杂性位于网络的边缘

DNS

DNS 服务
- 主机名到IP地址的翻译
- 主机别名
  - Canonical, alias names
- 邮件服务器别名
- 负载均衡
  - replicated Web servers: set of IP addresses for one canonical name

为什么不是集中化 DNS?
- 单点错误
- 通迅负载
- 延迟
- 维护

doesn't *scale!*

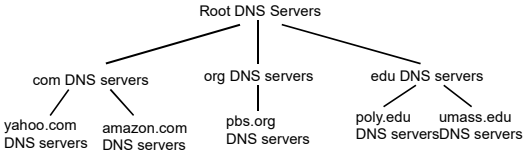
DNS

DNS 服务
- 主机名到IP地址的翻译
- 主机别名
  - 权威, 别名服务
- 邮件服务器别名
- 负载均衡
  - 中继 Web 服务: 一个别名对应多个地址

为什么不是集中化的 DNS?
- 单点错误
- 通迅负载
- 延迟
- 维护

无法扩展!

分布式/层次化的数据库



一个希望获得 www.amazon.comIP的client A.:
- client A查询 root server 获得com DNS server
- client A查询 com DNS server
  获得amazon.com DNS server
- client A查询amazon.com DNS server
  获得www.amazon.com IP 地址

DNS: Root name servers
根域名服务器

- 本地域名服务器(local name server) 与根域名服务器联系
- 根域名服务器:
  - 与权威域名服务器联系完成域名映射
  - 返回给本地域名服务器结果

13 个根域名服务器

TLD and Authoritative Servers
TLD和权威域名服务器

- 顶级域名服务器(Top-level domain TLD)
  - 负责com, org, net, edu, etc,
    以及所有区域顶级域名 uk, fr, ca, jp.
  - Network Solutions 维护com TLD
  - Educause维护edu TLD
- 权威域名服务器:
  - 一个单位的域名服务器,
    为本单位负责服务器与IP地址之间的映射
    (e.g., Web, mail).
  - 由本单位维护或者ISP维护

Local Name Server
本地域名服务器

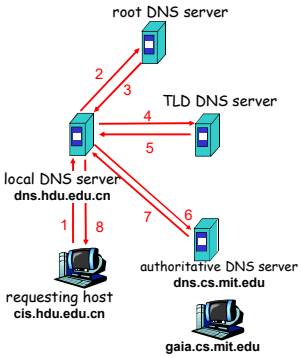
- 不是严格的属于域名服务器的层次结构
- 每个 ISP (区域ISP, 公司, 大学) 都有自己的LNS.
  - 也称"默认域名服务器"
- 当主机进行DNS 查询, 查询转发给LNS
  - 作为代理, 向域名服务器系统转发查询

DNS 解析例子

- Hdu.edu.cn 域
  某主机希望查询主机
  gaia.cs.mit.edu

迭代查询:

- LNS依次查询域名服务器层次结构中的各个服务器，最终获得该主机地址

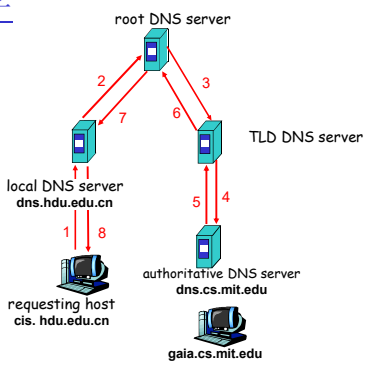


DNS 解析例子

递归查询:

- 将查询的负担完全交给了查询对象

- 负担?



DNS: caching 和记录更新

- 一个 DNS server 了解到域名映射后, 它会*caches* 映射
  - cache 有一定的 timeout 时间
  - TLD servers 通常会在LNS cache保存
    - Thus root name servers not often visited
- update/notify 机制的设计在IETF文档中有说明
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

DNS 记录

DNS: 分布式存储数据库记录(resource records RR)

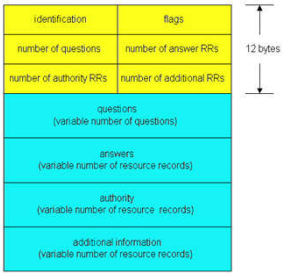
RR format: `(name, value, type, ttl)`

- Type=A
  - **name** 是主机名
  - **value** 是 地址
- Type=NS
  - **name** 域名(e.g. foo.com)
  - **value** 是本域的权威域名服务器
- Type=CNAME
  - **name** 是某台主机的别名 "canonical" (the real) name
    `www.ibm.com` 是对外的名字
  - **value** 是权威的名字
    `servereast.backup2.ibm.com`
- Type=MX
  - **value** 是邮件服务器的名字

DNS 协议, 消息

DNS 协议: *query* 和 *reply* 消息, 具有相同的格式

msg header
- identification: 16 bit # 用于标识query, reply 时检索那一次查询 #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



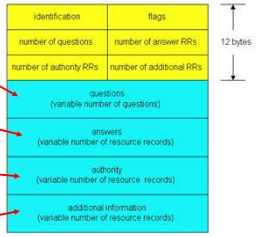
DNS 协议, 消息

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



DNS插入一条记录

- 例如: 一个初始公司"Network Utopia"
- 注册: *在DNS registrar* (e.g., Network Solutions)
  注册名字networkuptopia.com
  - 提供该公司的权威域名服务器的地址和名字 (primary and secondary)
  - registrar 将两条RRs 插入 com TLD 服务器:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- 生成权威服务器类型*A*记录:
- 类型*A* 记录 www.networkuptopia.com;
- 类型MX 记录 networkutopia.com。

DNS安全性

DDoS 攻击
- 流量轰击根服务器
  - Traffic Filtering
  - Local DNS cache
- 流量轰击TLD服务器
  潜在危险性更强

重定向攻击(Redirect attacks)
- Man-in-middle
  - 查询劫持
- DNS 污染
  - 发送虚假DNS响应

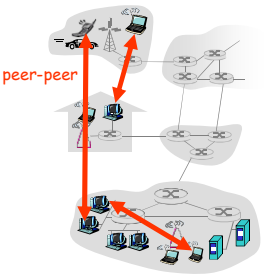
利用DNS 进行DDoS攻击
  - 发送虚假的回复IP地址

Chapter 2: Application layer

- 2.1 网络应用的原理
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 电子邮件
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P 应用
- 2.7 TCP Socket 编程
- 2.8 UDP Socket 编程

纯P2P架构

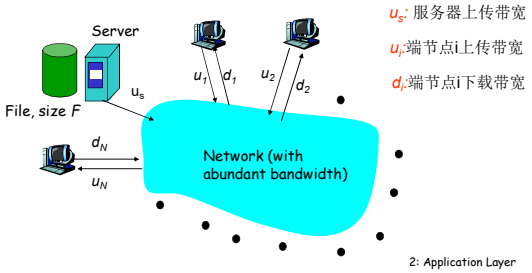
- *没有服务器*
- 端系统之间直接通信
- 端系统：经常改变IP 间歇性地连接



- Two topics:
  - 文件分发
  - DHT

文件分发: Server-Client vs P2P

Question：分发一个大小为*F*的文件给*N* 个端节点要花多少时间?



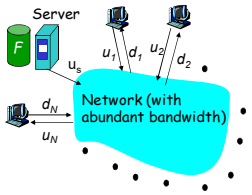
u_s: 服务器上传带宽

u_i:端节点i上传带宽

d_i:端节点i下载带宽

文件分发时间: server-client

- 服务器顺序发送N 个拷贝:
  - NF/u_s time
- 客户机 i 下载时间
  - F/di time

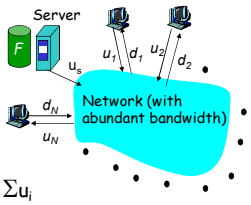


分发文件*F*给*N*个用户的时间 client/server approach $= d_{cs} = \max\{ NF/u_s, F/\min_i(d_i) \}$

随着N线性增加 (for large N)

文件分发时间: P2P

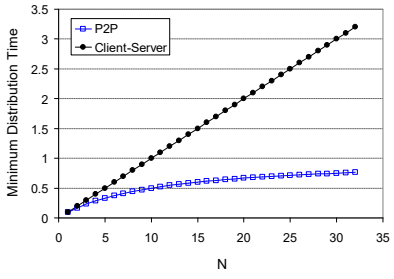
- 服务器顺序发送N 个拷贝:
  - NF/u_s time
- 客户机 i 下载时间
  - F/di time
- 总计NF bits 必须被下载
  - 可能的最快上传速率: $u_s + \Sigma u_i$



$$d_{P2P} = \max\{ F/u_s, F/\min(d_i)_i, NF/(u_s + \Sigma u_i) \}$$

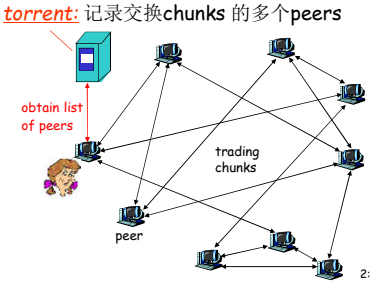
Server-client vs. P2P:

Client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

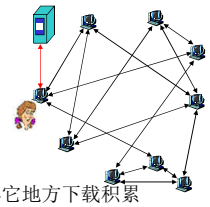


文件分发: BitTorrent

- 文件分成 256KB *chunks*.
- *tracker:* 跟踪参与共享的peer
- *torrent:* 记录交换chunks 的多个peers



BitTorrent (1)



- Peer 加入 torrent:
  - 开始没有 chunks, 但是慢慢从其它地方下载积累
  - 在 tracker 注册获得 peers列表,
  - 与相邻的 peers ("neighbors")取得联系
- Peer downloading, 同时 uploading chunks 给其他 peers.
- Peer 加入离开是动态
- Peer 下载完成, 可能离开 (selfishly) 或继续 (altruistically)

BitTorrent (2)

Pulling Chunks

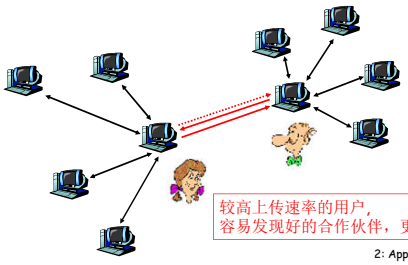
- 不同 peers 具有不同的文件 chunks
- 周期性的,
  peer (Alice) 问相邻peer 获得 chunks 列表.
- Alice 发送缺失的chunks 列表
  - rarest first
    (最稀罕优先)

Sending Chunks: tit-for-tat

- Alice 发送chunks 给四个速度最快的邻居
- every 10 secs
  重估top 4
- every 30 secs:
  随机选择一个节点加入传输
  - 可能成为最快的top 4
  - "optimistically unchoke
    (乐观激活？)"

BitTorrent: Tit-for-tat

(1) Alice随机选择(optimistically unchokes)了Bob;
(2) Alice 成为了Bob's top-four 提供者; Bob 也回报Alice;
(3) Bob 也成为了Alice's top-four 之一。



较高上传速率的用户,
容易发现好的合作伙伴，更快地下载文件！

Distributed Hash Table (DHT)

- Hash 表
- DHT 例子
- 环状 DHT 和叠加网络
- Peer 加入和离开

简单的数据库表

一个简单的数据库表，(key, value) 对:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
.....	.....
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

Hash Table

- 能够方便的存储和搜索数值表示的键值
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
.....		.....
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

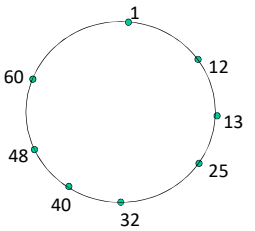
- 在数百万的peer间分配 (key, value) 对
  - 键值对在peers间均匀分布
- 所有的peer 能够query 在数据库中查询关键字
  - 数据库返回 key的value
  - 查询过程中，相关的peers节点间交换少量的消息
- 每个 peer 仅知道几个其它的peer节点
- Peer 能够加入和离开(churn)

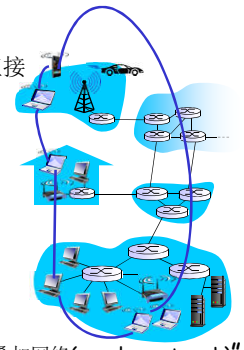
给peers分配key-value对

- 规则:将key-value对分配给具有最近ID (*closest* ID)的peer节点.
- 惯例: 最近的peer节点是key的 *immediate successor*.
- e.g., ID空间{0,1,2,3,...,63}
- 假设8 peers: 1,12,13,25,32,40,48,60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

环状 DHT

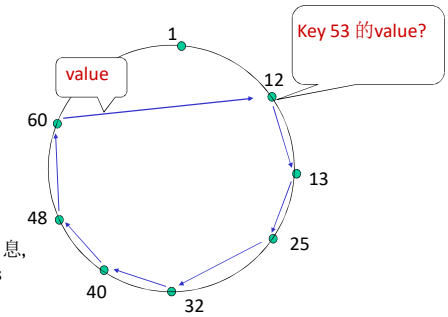
- 每个peer节点 仅知道它的直接 successor 和predecessor.





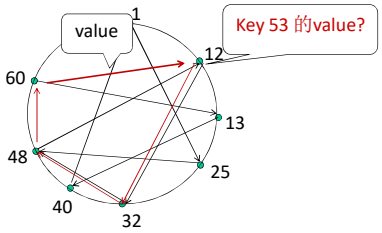
"叠加网络(overlay network)"

执行一次查询



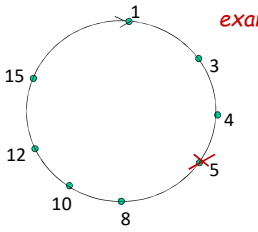
平均*O(N)* 个消息,
假设有*N* peers

带近路的环状DHT



- 每个peer 保留predecessor, successor, short cuts的IP 地址.
- 从6个减少到 3 个消息.
- 一种可能的设计： 每个peer包含*O(log N)* 个邻居,
  每次查询的消息数降为*O(log N)*

Peer加入和离开

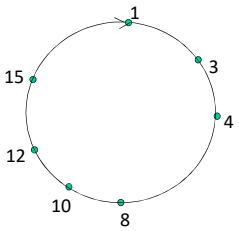


example: peer 5 abruptly leaves

handling peer churn:

- peers 可能加入和离开(churn)
- 每个peer 知道两个后继的地址
- 每个peer 周期性的pings两个后继以检测活性
- 如果直接后继离开,选择下一后继为当前直接后继

Peer加入和离开



handling peer churn:

- peers 可能加入和离开(churn)
- 每个peer 知道两个后继的地址
- 每个peer 周期性的pings两个后继以检测活性
- 如果直接后继离开,选择下一后继为当前直接后继

example: peer 5 abruptly leaves

- peer 4 检测到peer 5的离开； 选择8 它的直接后继
- 4 询问8 谁是它的直接后继； 选择8的直接后继为自己的第二后继.