第二章 应用层

目录

网络应用程序

- 电子邮件
- Web浏览器
- 远程登录（ssh、远程桌面）
- P2P文件共享
- 多人在线网游
- 视频（优酷、腾讯视频）
- 网络电话（微信电话）
- 视频会议（腾讯会议、网课）
- 社交（QQ、微信）
- 搜索引擎
- …

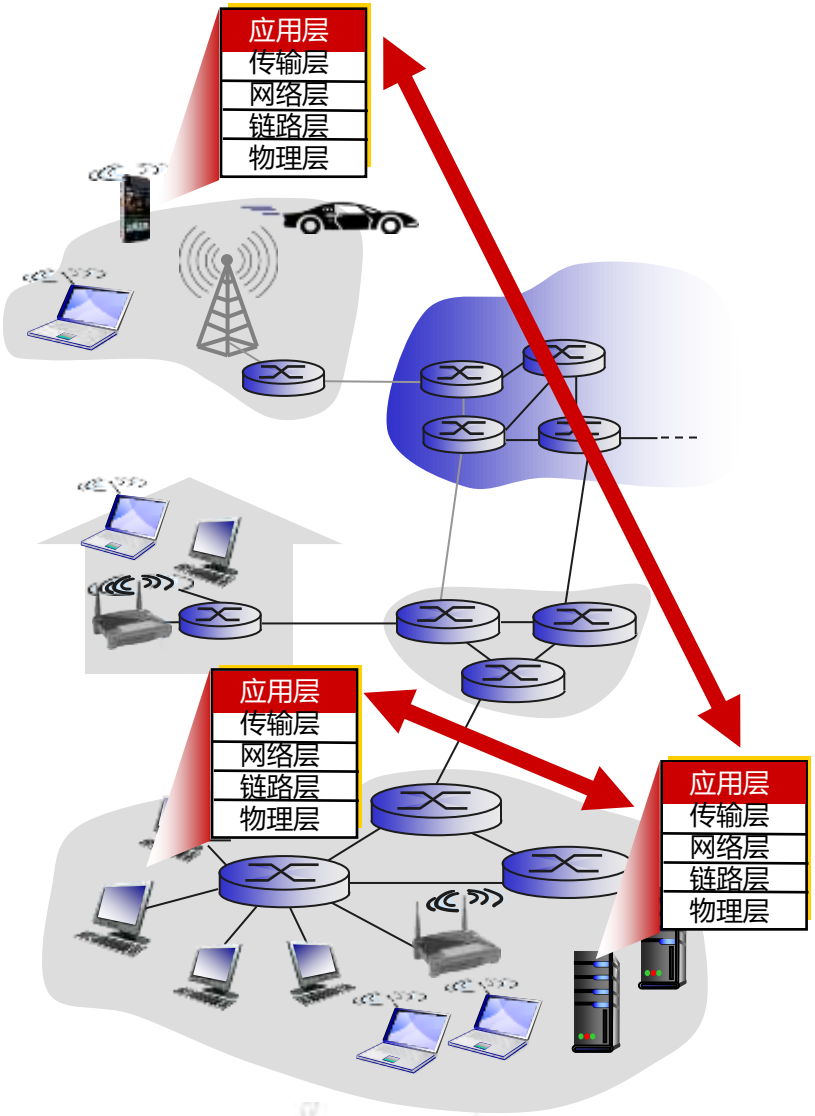
如何编写网络应用

编写程序:

- 运行在（不同的）终端系统上
- 通过网络进行通信
- 例如：web服务器和浏览器软件通过网络通信

无需对网络核心部分的设备编程

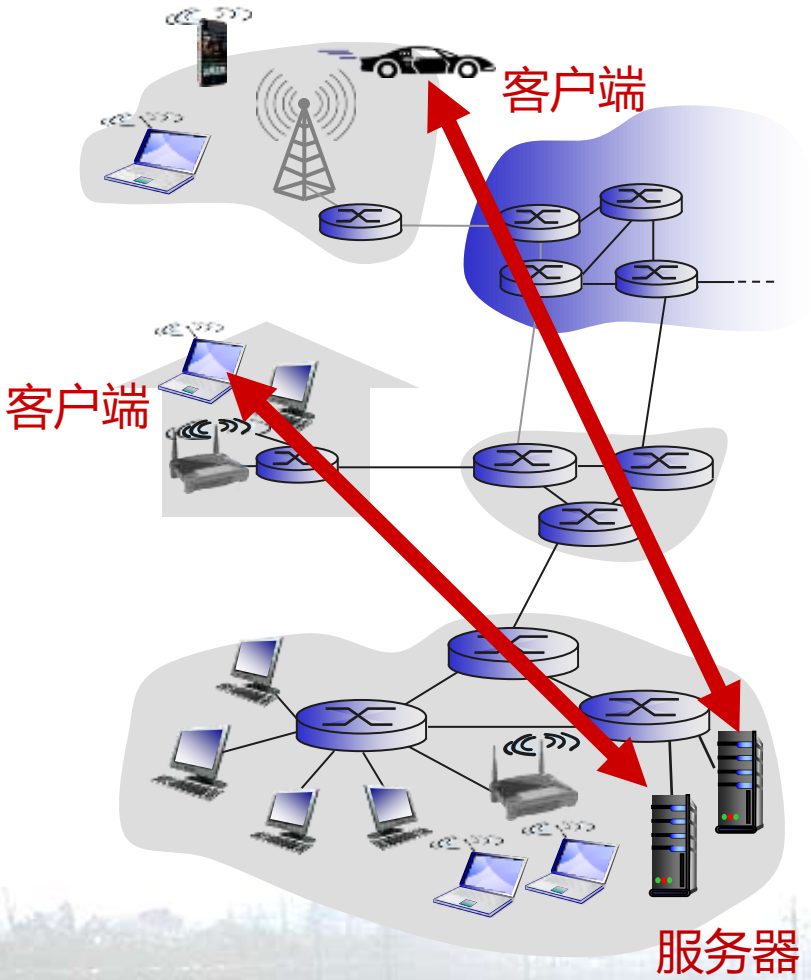
- 网络核心部分的设备（交换机、路由器）不运行用户应用程序
- 终端系统上的应用程序便于快速开发和传播



网络应用的体系结构

- 两类
  - 客户端-服务器（client-server）结构
  - 对等网络（peer-to-peer，P2P）结构

客户端-服务器结构



服务器：

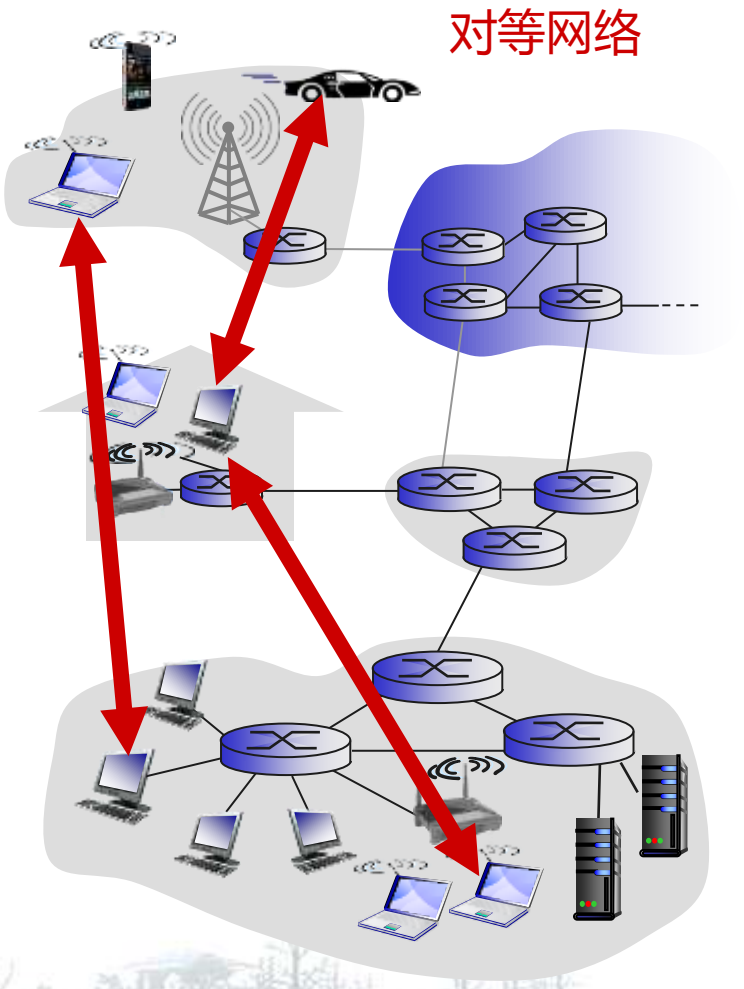
- 运行在永远在线的主机上
- 永久IP地址
- 部署在数据中心，提供大规模服务

客户端：

- 与服务器通信
- 断续地连接网络
- 可以使用动态地址
- 彼此间不直接通信

对等网络结构

- 没有永远在线的服务器程序
- 任何终端系统直接通信
- Peer向其它peer请求服务，并且也服务其它peer
  - 可扩展性—新的peer带来新的服务能力和服务需求
- Peer之间断续连接，可以改变IP地址
  - 需要复杂的管理机制



进程通信

进程：主机上运行中的程序

- 通信的两个进程在同一个主机上，进程间通信，操作系统的范畴
- 通信的两个进程在不同主机上，通过相互交换消息报文实现通信，网络范畴

clients, servers

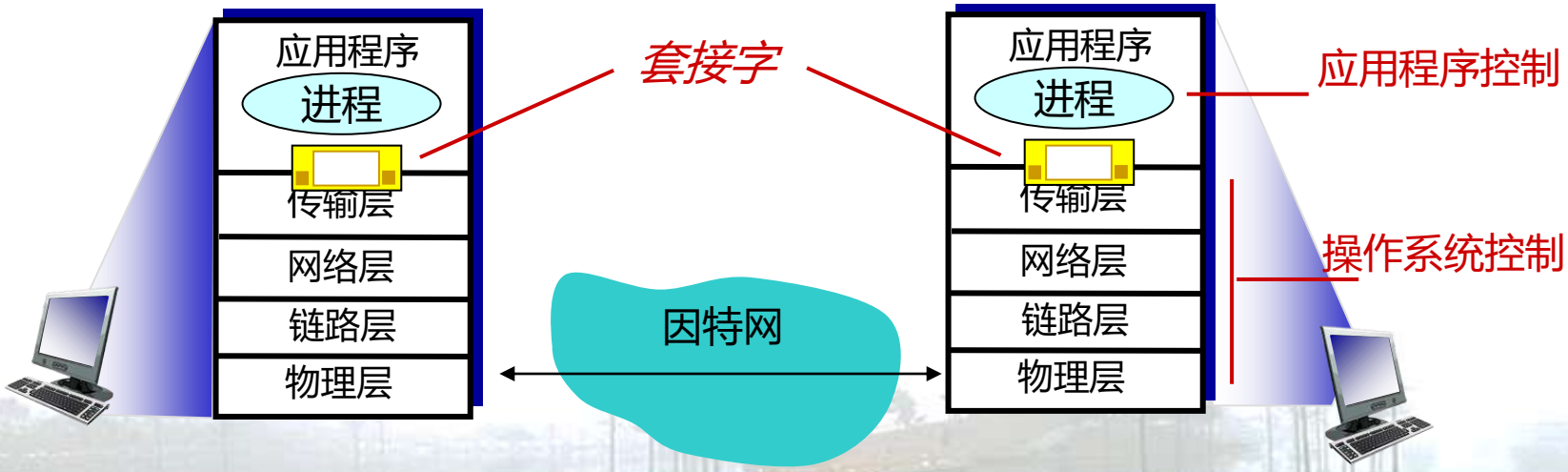
客户端进程：发起通信的一方

服务器进程：等待被连接的一方

- 对等网络应用同时具有服务器进程和客户端进程

套接字

- 进程通过其套接字（socket）收发报文消息
- 套接字类似门
  - 发送端进程将消息推出门
  - 依赖网络基础设施和对方进程的门将消息送达



标识进程

- 为接收报文消息，进程必须有一个标识ID
- 主机设备有一个32-bit的IP地址
- 问: IP地址是否可以用来标识进程?

- 进程标识包括主机的IP地址和进程在主机上分配的端口号.
- 端口号举例:
  - HTTP服务器进程端口号: 80
  - 邮件服务器进程端口号: 25
- 向mail.ustc.edu.cn服务器进程发送HTTP报文消息，
  - 对方IP地址: 202.38.64.8
  - 对方端口号: 80
- more shortly…

应用层协议定义如下内容

- 消息类型
  - 例如，请求、响应
- 消息词法
  - 包含哪些字段和字段的顺序
- 消息语义
  - 字段信息的含义
- 处理、发送、回复消息的规则

开放协议：

- RFC定义
- 协议间具备互操作性
- 例如, HTTP, SMTP

私有协议：

- 例如，微信

应用程序需要的传输层服务

数据完整性

- 某些应用(例如，文件、web)要求100%可靠数据传输
- 另一些应用(例如,音视频流)可以容忍一定的数据丢失

时延管理

- 某些应用(例如，网络电话,交互性网游) 要求时延低于特定值

吞吐率

- 某些应用(例如， 多媒体)要求一定的吞吐率
- 另一些应用(“弹性应用”) 有多少带宽用多少

安全

- 加密、数据完整性、…

应用程序需要的传输层服务

网络应用	数据丢失	吞吐率	延迟敏感
文件传输	不能容忍	弹性	不敏感
电子邮件	不能容忍	弹性	不敏感
Web	不能容忍	弹性	不敏感
实时音视频	可容忍	音频: 5kbps-1Mbps 视频:10kbps-5Mbps	敏感, 几百毫秒
非实时音视频	可容忍	与上面相同	敏感, 几秒
交互性网游	可容忍	几个 kbps	敏感, 几百毫秒
即时消息	不能容忍	弹性	不确定

因特网的传输层服务

TCP服务:

- 在收发进程间的可靠数据传输
- 流控制: 发送端的数据发送速率不会超过接收端能力
- 拥塞控制：网络过载时限制发送端的发送速率
- 不提供: 时延保障、最低吞吐率保障、安全
- 面向连接: 收发进程需要预先建立连接

UDP服务:

- 在收发进程间不可靠的数据传输
- 不提供: 可靠性、流控制、拥塞控制、时延保障、吞吐率保障、安全、连接建立

问: 有TCP，为何需要UDP?

因特网应用程序的传输层协议

网络应用	应用层协议	下层的传输层协议
电子邮件	SMTP [RFC 2821]	TCP
远程登录	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
文件传输	FTP [RFC 959]	TCP
流媒体	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP 或UDP
网络电话	SIP, RTP, 私有协议 (e.g., 微信)	TCP或UDP

增强TCP安全

TCP & UDP

- 不加密
- 通过socket在因特网上明文传输密码等信息

SSL

- 提供加密的TCP连接
- 保证数据完整性
- 终端认证

SSL位于应用层

- 应用程序使用SSL库与TCP协议交互

SSL套接字编程接口

- 明文密码加密后在因特网上传输
- 详见第八章

目录

Web和HTTP

- Web页面由对象构成
- 对象可以是HTML文件、JPEG图片、Java小程序、音视频文件、…
- Web页面包含一个基础的HTML文件，文件中包含对多个对象的引用
- 引用每个对象可用URL寻址, 例如,

`www.someschool.edu/someDept/pic.gif`

主机名　　路径名

HTTP概览

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

- Web服务的应用层协议
- 客户端/服务器模式
  - 客户端: 使用HTTP协议请求、接收、展示web对象的浏览器软件
  - 服务器: 使用HTTP协议回应请求并发送web对象的Web server软件

HTTP概览

HTTP如何使用TCP:

- 客户端进程创建套接字，向服务器进程的80端口发起连接请求
- 服务器进程接收来自客户端的TCP连接请求
- 客户端（浏览器进程）和服务器（web server进程）交换HTTP消息(应用层协议报文)
- 关闭TCP连接

HTTP是“无状态的”协议

- 服务器不记录客户端以前的请求

aside

有状态的协议较为复杂

- 需要维护历史信息（状态）
- 如果客户端或服务器进程意外崩溃，它们的状态可能不一致，需要重新协商

HTTP连接

非持久HTTP

- 一个TCP连接最多传输一个对象
  - 传输完成，关闭TCP连接
- 下载多个对象时，需要建立多个TCP连接

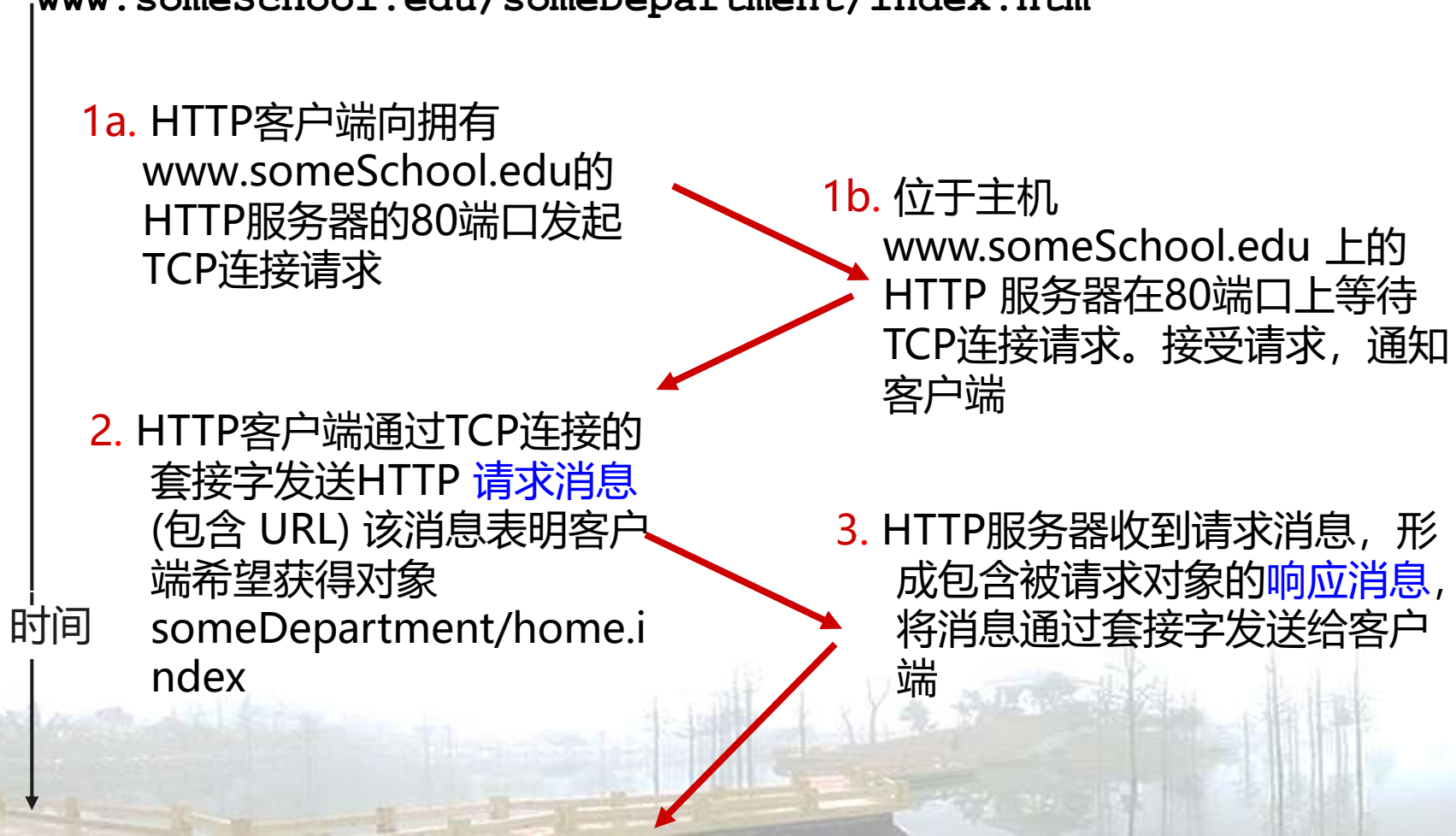
持久HTTP

- 多个对象可通过客户端和服务器之间的一个TCP连接传输

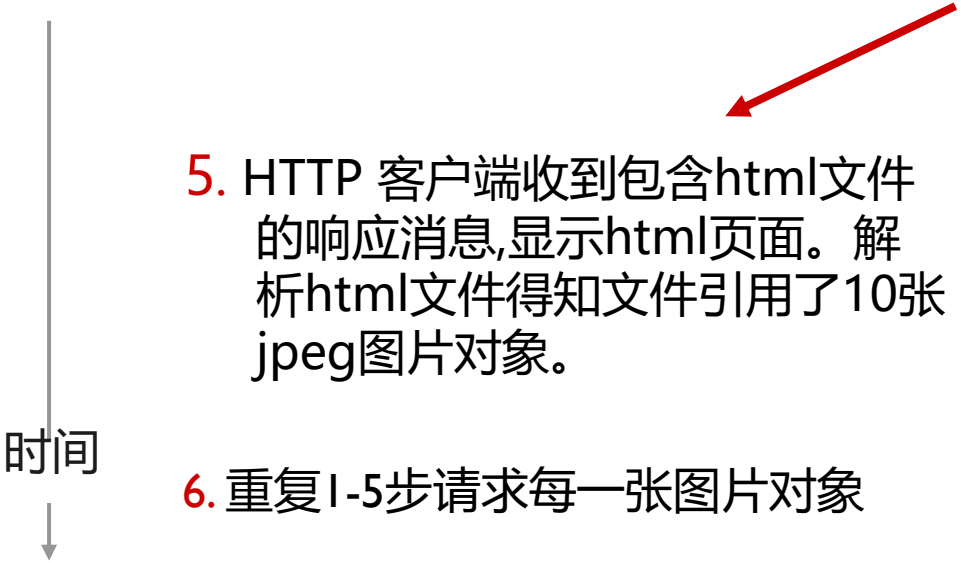
非持久HTTP

假设用户在浏览框输入以下URL: (包含文本和10张图片)

www.someSchool.edu/someDepartment/index.htm



非持久HTTP

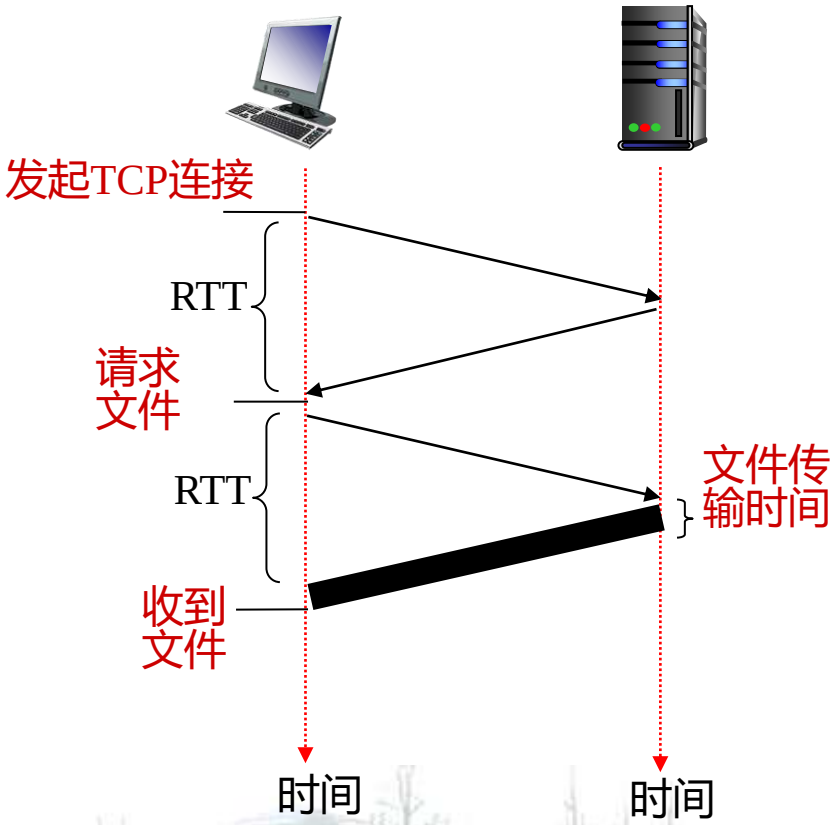

4. HTTP 服务器关闭TCP连接.
5. HTTP 客户端收到包含html文件的响应消息,显示html页面。解析html文件得知文件引用了10张jpeg图片对象。
时间
6. 重复1-5步请求每一张图片对象

非持久HTTP：响应时间

往返时延RTT (定义): 一个小数据包从客户端到服务器再返回客户端所需时间

HTTP响应时间:

- 一个RTT用于建立TCP连接
- 一个RTT用于发送HTTP请求并开始收到HTTP响应
- 文件传输时间
- 非持久HTTP响应时间 =2RTT+文件传输时间



持久HTTP

非持久HTTP存在的问题:

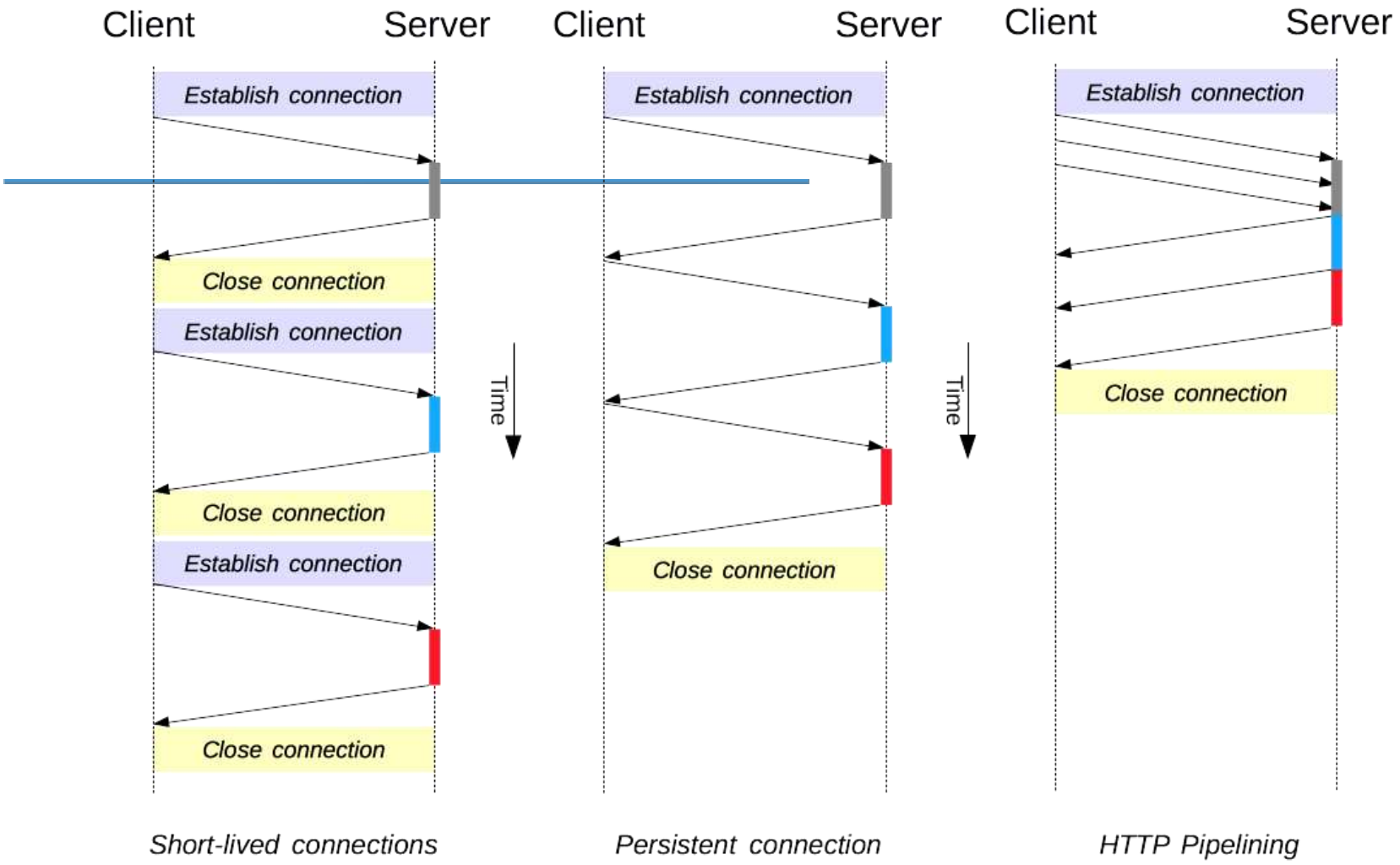
- 每获取一个对象需要2个RTT
- 每个TCP连接消耗操作系统资源
- 浏览器通常建立多个并发TCP连接，获取引用的对象
  - 浏览器可设置最大并发连接数
  - 设置为1，则实际上串行请求对象

持久HTTP:

- 服务器发送响应消息后不关闭TCP连接
- 后续客户端和服务器之间的HTTP消息可以使用这个TCP连接传输
- 客户端发现引用对象后可以立刻发出请求消息

持久HTTP

- 非流水线：通过一个TCP连接逐个请求对象
  - 一个RTT建立TCP连接
  - 获取每个对象，请求和响应消息往返消耗一个RTT
  - N个对象：RTT+N×RTT
- 流水线：通过一个TCP连接并发请求多个对象
  - 一个RTT建立TCP连接
  - 一个RTT用于获取所有对象的请求和响应消息往返
  - N个对象：RTT+RTT



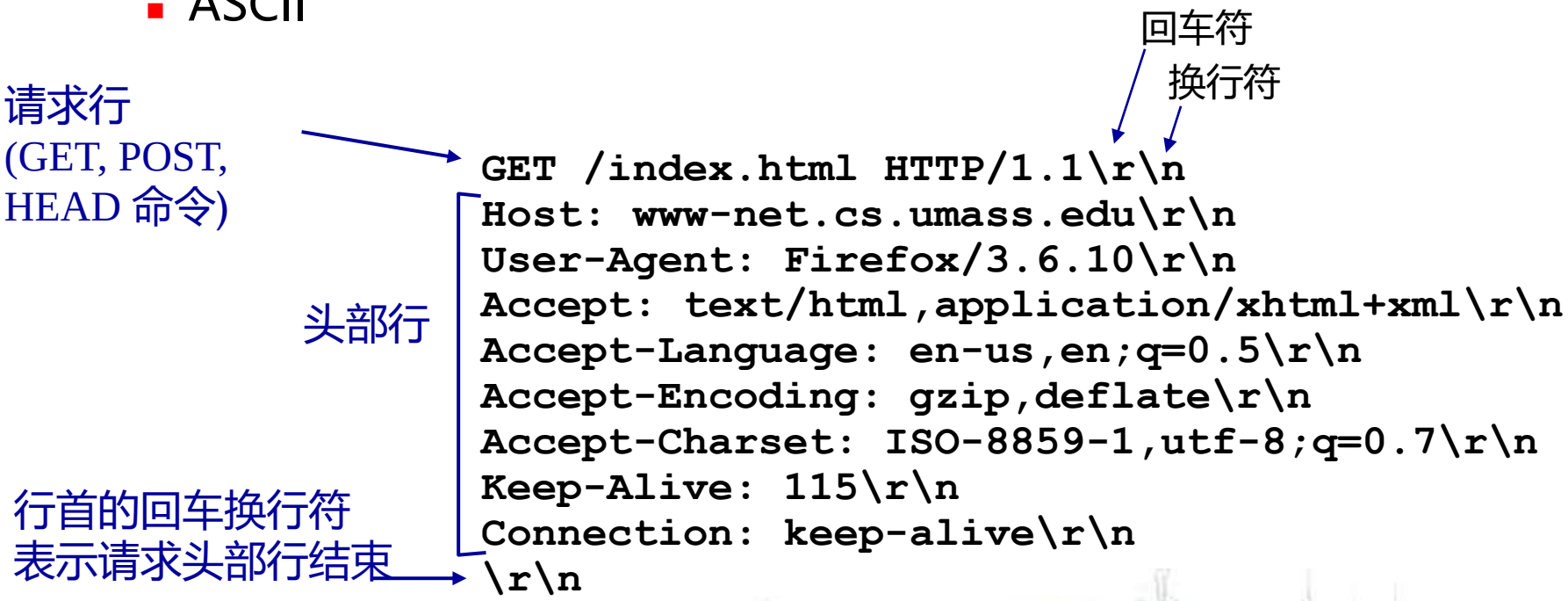
HTTP/1.0
2 RTT per object

HTTP/1.1
1 RTT for connection
1 RTT per object

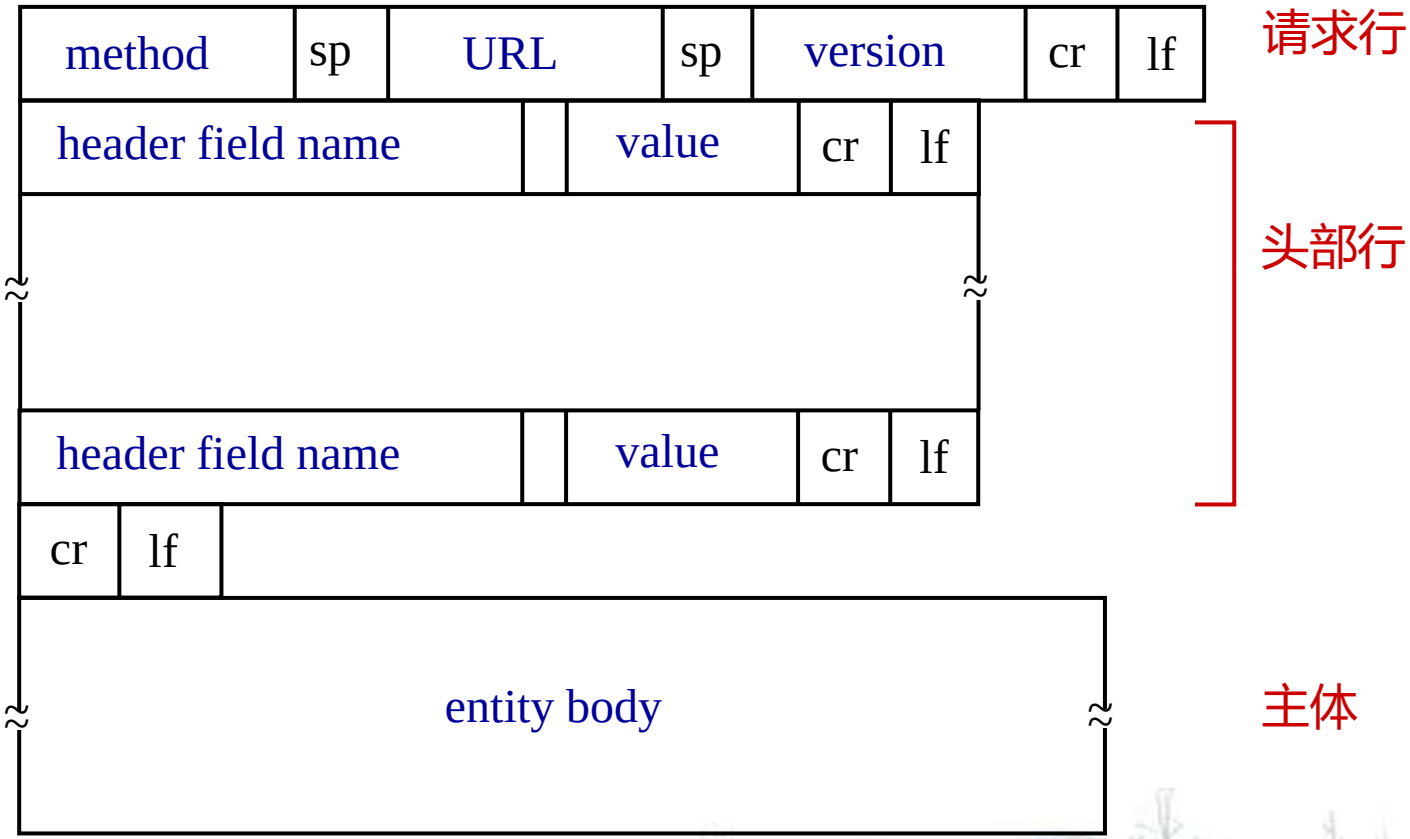
HTTP/1.1
1 RTT for connection
1 RTT for all objects

HTTP请求消息

- 两种HTTP消息类型:请求、响应
- 请求消息:
  - ASCII



HTTP请求消息格式


method
sp
URL
sp
version
cr
lf
请求行
header field name
value
cr
lf
header field name
value
cr
lf
头部行
cr
lf
entity body
主体

上传输入

POST 方法:

- 网页包含输入框
- 输入被放在实体中上传

包含输入框的网页

URL方法:

- 使用GET命令
- 输入框输入被放在请求行的URL字段上传:

```
www.somesite.com/animalsearch?monkeys&banana
```

方法类型

HTTP/1.0:

- GET
- POST
- HEAD
  - 类似GET，但是服务器端仅返回消息头部，不返回实体
  - 用于测试URL的有效性

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - 将实体部分中的文件上传到URL指定的路径
- DELETE
  - 删除URL指定路径的文件

HTTP请求消息：头部行

- Host: 存放对象的主机
- Connection:
  - keep-alive: 指定时间内维持TCP连接
  - close: 关闭持久连接
- User-agent: 浏览器名称和版本
- Accept-language:

```
GET /index.html HTTP/1.1\r\n
Host: www.ustc.edu.cn\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
```

HTTP响应消息

状态行
(状态代码
和短语)

头部行

数据，例如
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 ...
```

HTTP响应状态代码

- 出现在服务器发向客户端的响应消息的第一行
- 一些状态代码和短语:

200 OK

- 请求成功，被请求对象在此消息的数据部分

301 Moved Permanently

- 被请求的对象移走了，它的新位置（URL）在消息的Location头部行

400 Bad Request

- 请求消息不被服务器理解

404 Not Found

- 在服务器上没有找到所请求的对象

505 HTTP Version Not Supported

动手尝试客户端HTTP

1. 使用telnet登录Web server:

```
telnet gaia.cs.umass.edu 80
```

在服务器80端口建立TCP连接
(80是缺省的HTTP服务器端口) .
之后任何输入字符都会被送到
gaia.cs.umass.edu的80端口

2. 输入GET HTTP请求消息:

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

向HTTP服务器发送一个最小但完整的
HTTP请求消息
两次回车换行结束消息

3. 观察HTTP服务器返回的响应消息!
(或者用wireshark抓包观察请求和响应)

用户在服务器的状态：Cookie

许多网站使用Cookie

四部分构成:

1) HTTP响应消息中包含cookie头部行
2) 下一个HTTP请求消息中包含cookie头部行
3) 用户在主机的浏览器中上持有一个cookie文件
4) 网站的后端数据库存储用户的cookie信息

举例:

- 张三一直用他的PC上网
- 张三用PC第一次访问淘宝
- 当张三的浏览器发出第一条HTTP请求信息时，淘宝网站意识到这是一个新的用户或用户设备，并为张三的这台PC创建:
  - 独一无二的用户ID
  - 在后台数据库添加ID条目

用户在服务器的状态：Cookie

Taobao 8734

Cookie文件

普通的HTTP请求消息

京东服务器
为用户创建
ID 1678

添加数据
库条目

后端数据库

普通的HTTP响应消息
set-cookie: 1678

Taobao 8734
JD 1678

普通的HTTP请求
cookie: 1678

关联cookie
的操作

访问

普通的HTTP响应消息

一周以后:

访问

Taobao 8734
JD 1678

普通的HTTP请求消息
cookie: 1678

关联cookie
的操作

Cookie

Cookie有什么用:

- 认证
- 购物车
- 推荐
- 保持用户会话状态（例如：写了一半的email）

隐私问题:

- Cookie使网站能够持续了解用户
- 可以将网络浏览、购物等行为和真实的人关联起来

Web缓存（代理服务器）

目的： 在不访问源服务器的情况下满足客户端请求

- 用户设置浏览器通过代理缓存访问web
- 浏览器将所有HTTP请求发给缓存
  - 对象在缓存中：返回给客户端
  - 否则向源服务器请求对象，并返回给客户端

Web缓存

- 缓存同时扮演客户端和服务器
  - 对于发起请求的客户端，缓存是服务器
  - 对于源服务器，缓存是客户端
- 缓存通常由因特网服务提供商(ISP)设置(大学、企业、运营商)

为何需要缓存?

- 缩短客户端请求的响应时间
- 降低机构（大学、企业）接入链路的流量
- 缓存大量部署在因特网上：有助于用户访问那些无法部署大量源服务器的内容提供者发布的内容(P2P文件共享有类似效果)

缓存示例

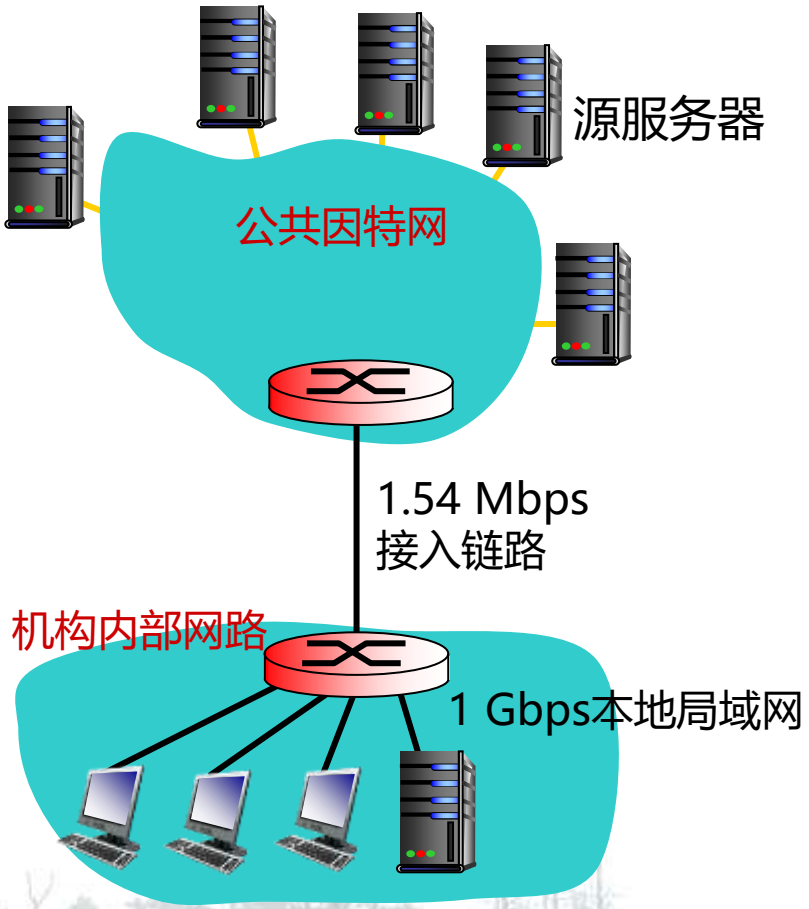
假设:

- 对象的平均大小: 100K bits
- 浏览器向源服务器发起请求的平均速率:15个/秒
- 浏览器平均接收数据速率: 1.50 Mbps
- 机构网关路由器到源服务器的RTT: 2秒
- 机构接入链路的带宽: 1.54 Mbps

没有缓存:

- 本地局域网带宽利用率: 0.15%
- 机构接入链路的带宽利用率 = 99%
- 时延 = 因特网时延 + 接入链路时延 + 本地局域网时延
  = 2秒 + 若干分钟 + 若干微秒

拥塞!



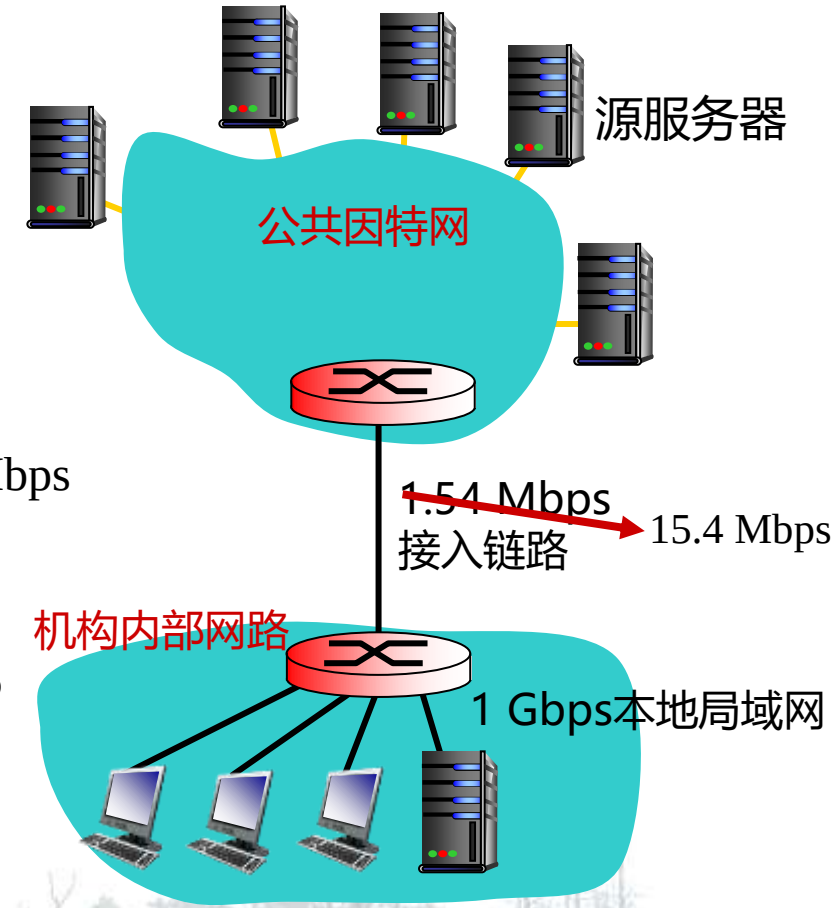
缓存示例：增加接入带宽

假设:

- 对象的平均大小: 100K bits
- 浏览器向源服务器发起请求的平均速率:15个/秒
- 浏览器平均接收数据速率: 1.50 Mbps
- 机构网关路由器到源服务器的RTT: 2秒
- 机构接入链路的带宽: ~~1.54 Mbps~~ 15.4 Mbps

没有缓存:

- 本地局域网带宽利用率: 0.15%
- 机构接入链路的带宽利用率 = ~~99%~~ 9.9%
- 时延 = 因特网时延 + 接入链路时延 + 本地局域网时延
  = 2秒 + ~~若干分钟~~ + 若干微秒
  若干毫秒



代价：接入网带宽并不便宜

缓存示例：安装本地缓存

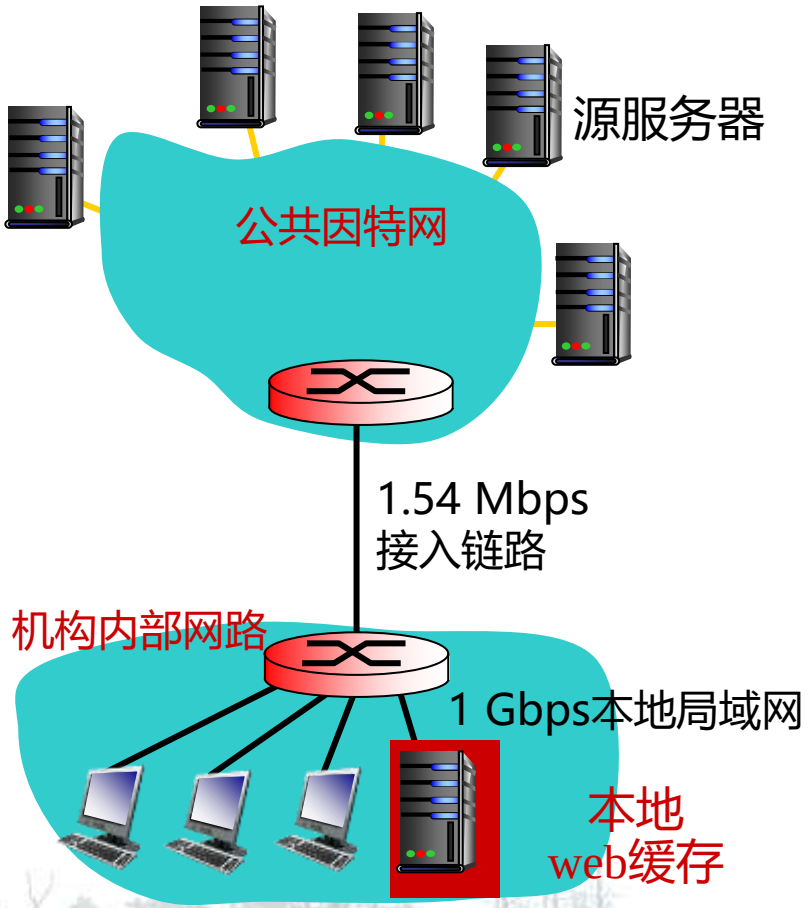
假设:

- 对象的平均大小: 100K bits
- 浏览器向源服务器发起请求的平均速率:15个/秒
- 浏览器平均接收数据速率: 1.50 Mbps
- 机构网关路由器到源服务器的RTT: 2秒
- 机构接入链路的带宽: 1.54 Mbps

没有缓存:

- 本地局域网带宽利用率: 0.15%
- 机构接入链路的带宽利用率 = ？？
- 时延 = ？？

如何计算?

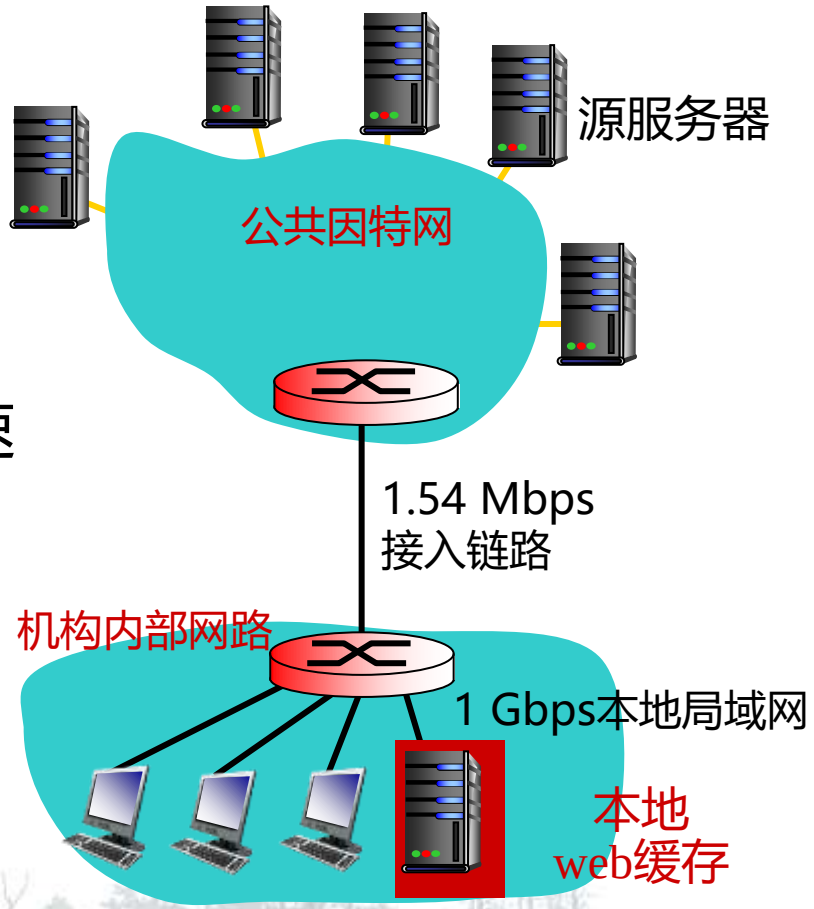


代价：缓存服务器，便宜

缓存示例：安装本地缓存

安装缓存后计算带宽利用率和时延:

- 假设缓存命中率是0.4
  - 40%的请求被缓存满足, 60%的请求被源服务器满足
- 接入链路的带宽利用率:
  - 60% 的请求对象通过接入链路传输
- 所请求对象在接入链路上的数据速率 = 0.6*1.50 Mbps = 0.9 Mbps
  - 利用率= 0.9/1.54 = 0.58
- 总的时延
  - = 0.6 * (浏览器到源服务器时延) +0.4 * (浏览器到缓存的时延)
  - = 0.6 (2.01秒) + 0.4 (若干毫秒) = 约1.2秒
  - 比使用15.4 Mbps接入链路的时延更短，而且更便宜

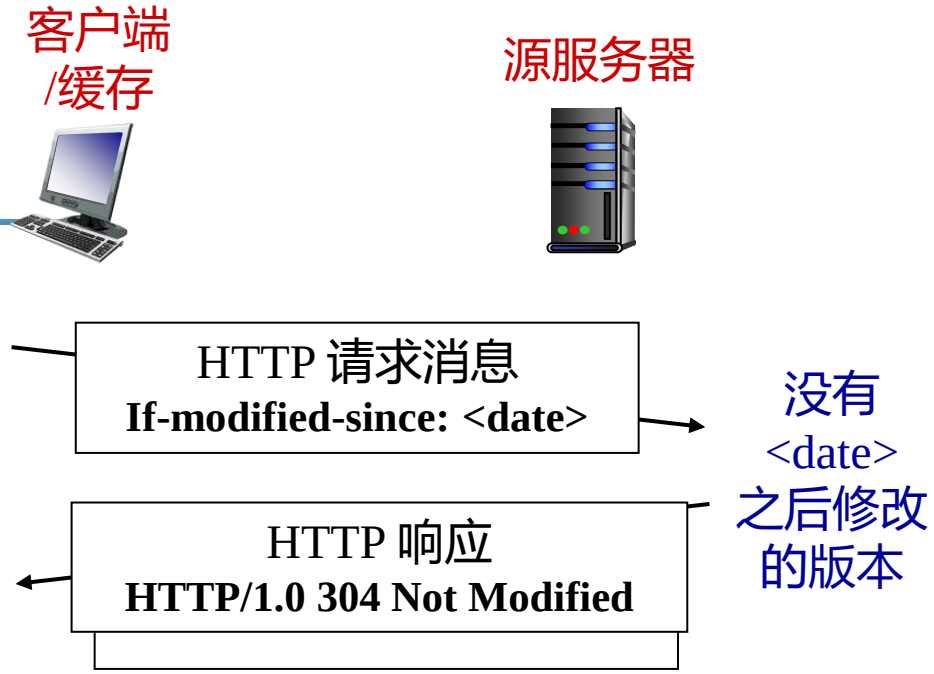


条件GET

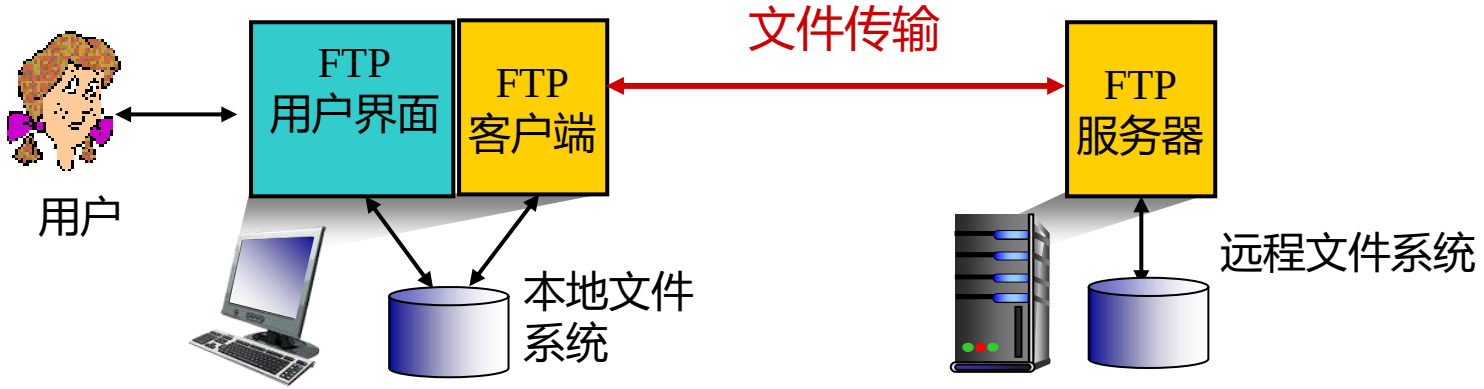
- 目标: 如果缓存中的对象备份未过时，源服务器不传输对象
  - 没有传输造成的时延
  - 不消耗链路带宽
- 缓存：在HTTP请求中明确本地备份的时间戳

  If-modified-since: <date>
- 源服务器: 如果源服务器上没有更新的版本，HTTP响应中不包含对象:

  HTTP/1.0 304 Not Modified



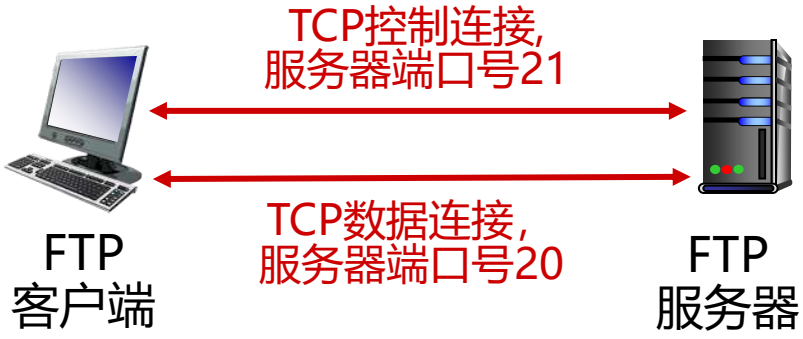
FTP：文件传输协议



- 本地和远程主机之间传输文件：上传、下载
- 客户端-服务器模式
  - 客户端: 发起文件传输的一方，可以是上传/下载文件
  - 服务器: 远程主机
- ftp协议: RFC 959
- ftp服务器工作在TCP 21端口上

FTP：单独的控制和数据连接

- FTP客户端在21端口与FTP服务器建立TCP连接，该连接是控制连接
- 客户端通过控制连接认证
- 客户端通过控制连接发送命令，浏览远程文件目录
- 当服务器收到文件传输命令，建立第二条TCP连接用于文件传输，该连接是数据连接，服务器端口号是20
- 当完成一个文件的传输，服务器关闭数据连接



- 服务器在端口20建立新的数据连接传输新的文件.
- 控制连接：“带外控制”
  - HTTP，“带内控制”，一条TCP连接同时是控制和数据
- FTP服务器维护状态: 当前的目录、用户认证状态等

FTP的命令和响应

命令示例:

- 以ASCII文本在控制连接中传输
- **USER 用户名**
- **PASS 密码**
- **LIST** 返回当前目录下文件
- **RETR 文件名** 获取文件
- **STOR 文件名** 在远端主机上存放文件

响应码示例

- 状态码和短语
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

目录

- 2.1 应用层的原则
- 2.2 Web和HTTP，FTP
- 2.3 电子邮件
  - SMTP、POP3、IMAP
- 2.4 DNS
- ~~2.5 对等网络应用~~
- ~~2.6 视频流和内容分发网络~~
- 2.7 UDP和TCP套接字编程

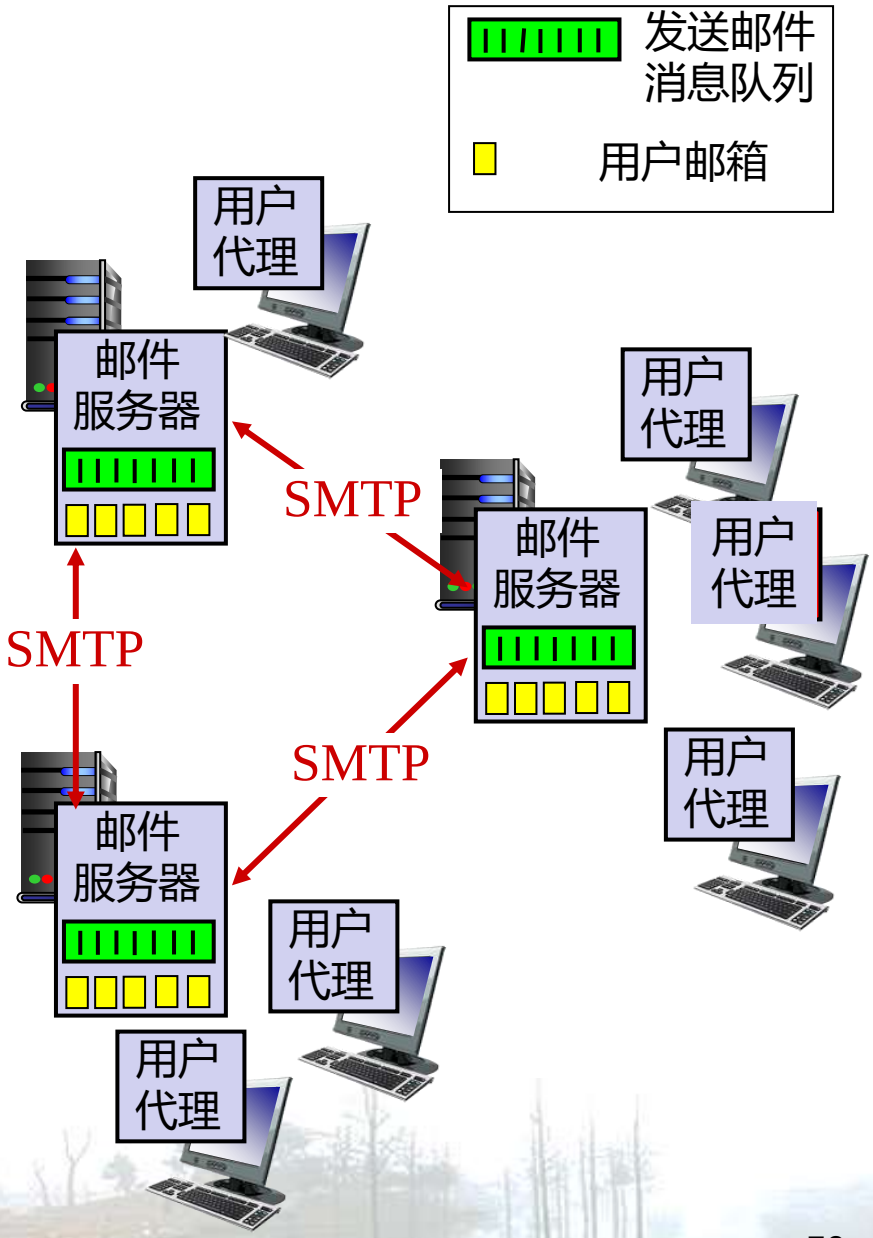
电子邮件

三部分构成:

- 用户代理软件
- 邮件服务器
- 简单邮件传输协议: SMTP

用户代理

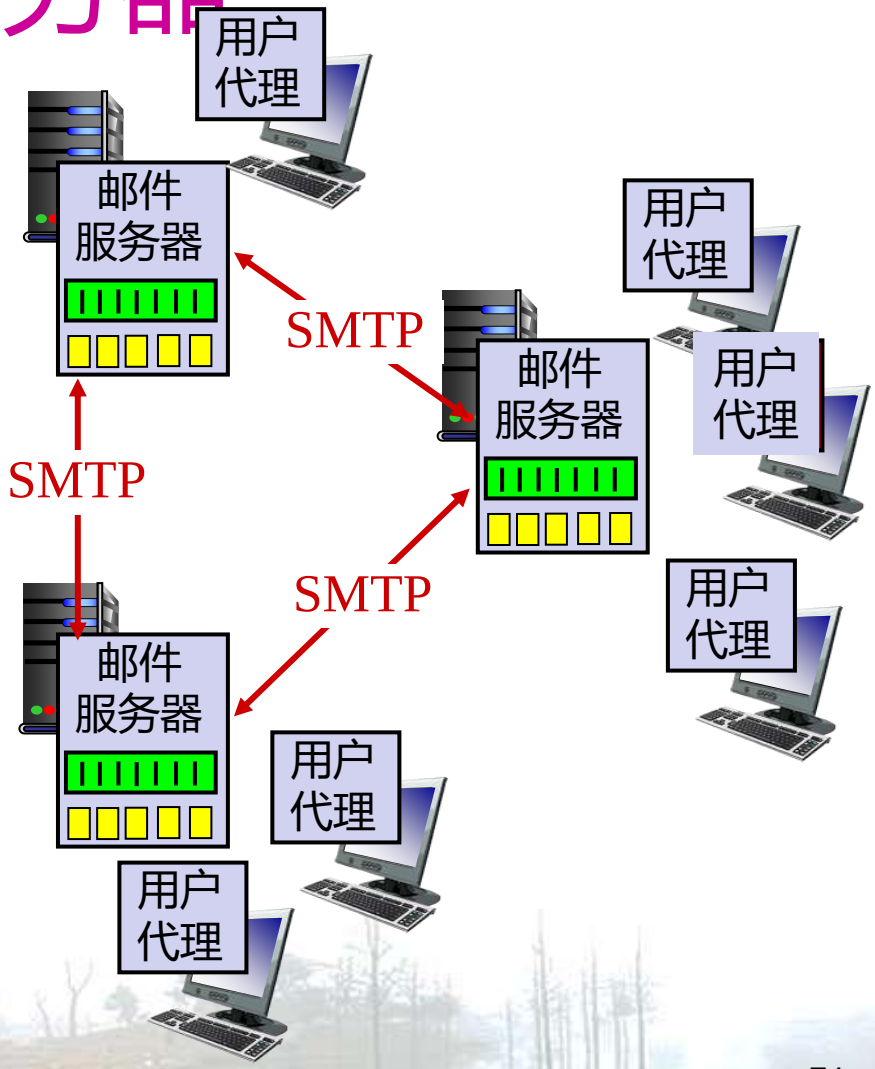
- 用于书写、编辑、阅读邮件消息
- 例如，Outlook, Thunderbird, 各种手机上的邮件app
- 发出和接收的邮件存储在邮件服务器上



电子邮件：邮件服务器

邮件服务器:

- 邮箱存放收到的用户邮件消息
- 消息队列待发出的邮件消息
- 邮件服务器之间使用SMTP 协议传输邮件消息
  - 客户端：发送邮件的服务器
  - 服务器端：接收邮件的服务器

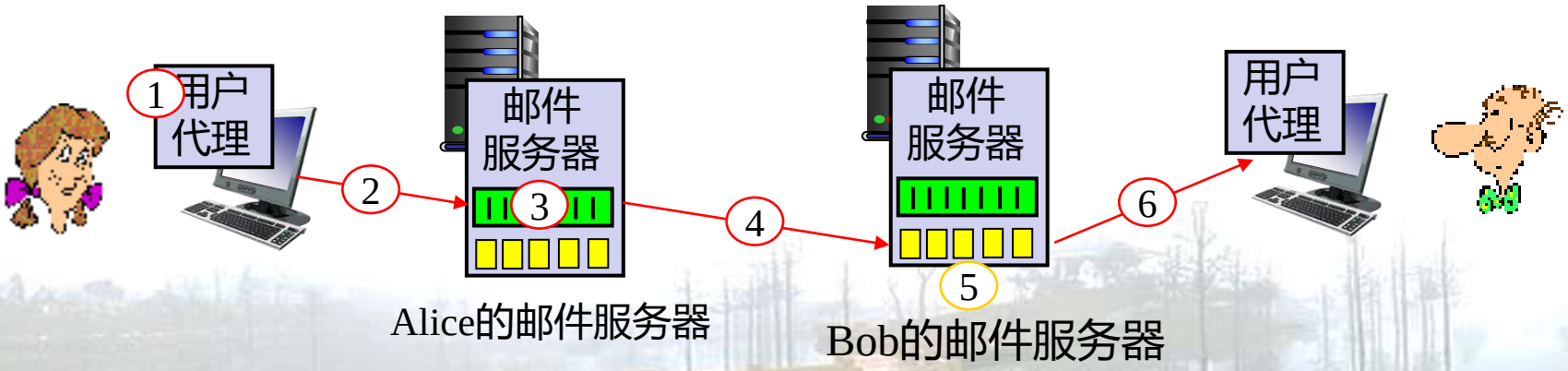


电子邮件：SMTP（RFC 2821）

- 使用TCP在25端口号上从客户端到服务器端可靠地传输邮件消息
- 服务器到服务器直接传输
- 传输邮件的三个步骤
  - 握手
  - 传输邮件消息数据
  - 关闭
- 通过命令/响应 交互 (类似HTTP)
  - 命令: ASCII文本
  - 响应: 状态码和短语
- 邮件消息（包括标题、正文、附件）使用7-bit ASCII编码

举例：Alice发一封邮件给Bob

1) Alice使用用户代理编写了一封邮件给 bob@someschool.edu
2) Alice的用户代理将邮件消息发给她的邮件服务器; 邮件消息在消息队列中排队准备发送
3) SMTP客户端的邮件服务器（Alice的邮件服务器）向SMTP服务器端的邮件服务器（Bob的邮件服务器）建立TCP连接
4) SMTP客户端通过TCP连接发送Alice的邮件消息
5) Bob的邮件服务器将收到的消息放到Bob的邮箱中
6) Bob使用他的用户代理，从邮件服务器的邮箱读取邮件消息



STMP协议交互示例

```
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
```

动手尝试STMP协议交互

- telnet 邮件服务器 25
- 收到邮件服务器的220响应
- 输入HELO、MAIL FROM、RCPT TO、DATA、QUIT命令

纯手工发送邮件

SMTP

- SMTP 使用持久TCP连接
- SMTP 要求邮件消息的所有数据（标题、内容、附件）都使用7-bit ASCII编码
- SMTP 服务器使用CRLF.CRLF表示消息结束

与HTTP对比

- HTTP: 拉（客户端请求web对象）
- SMTP: 推（客户端主动发送邮件）
- 都是用ASCII编码的命令和响应消息，都有状态码
- HTTP: 每个对象封装在一条响应消息里发送
- SMTP: 多个对象在一条包含多个部分的消息中发送

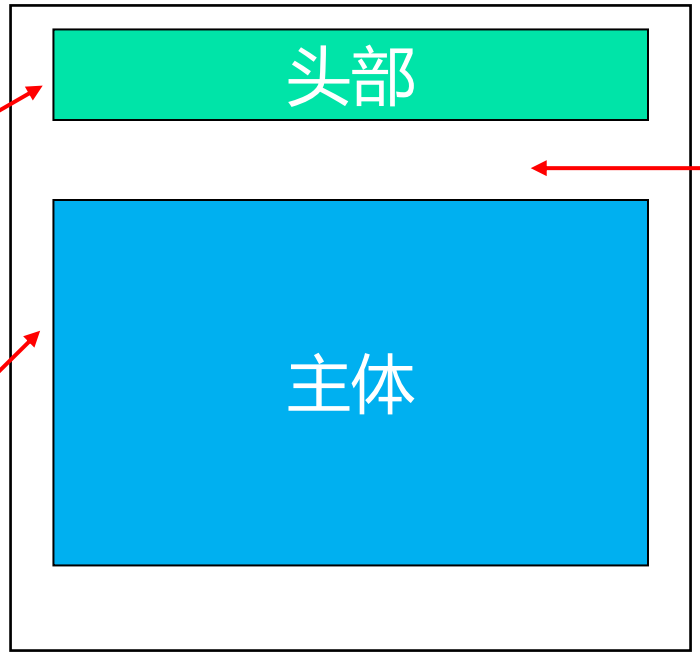
邮件消息格式

SMTP: 交换邮件消息的协议

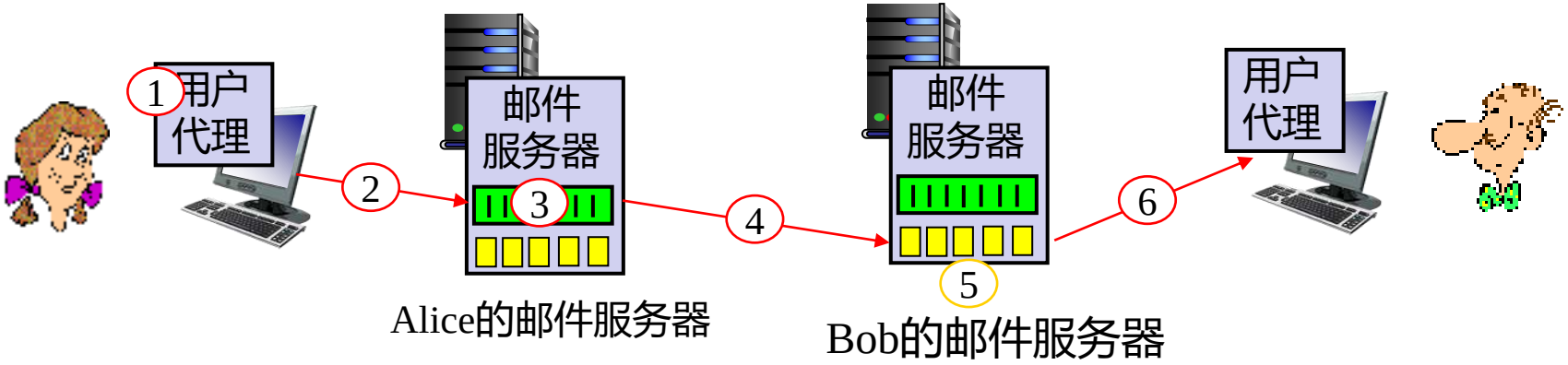
RFC 822: 邮件消息的文本格式:

- 头部行, 例如,
  - To:
  - From:
  - Subject:

  不同于SMTP协议的 MAIL FROM, RCPT TO: 等命令!
- 主体：邮件内容
  - 只用ASCII编码



访问邮件的协议



- SMTP: 将邮件传输存放到接收这封邮件服务器
- 邮件访问协议: 从邮件服务器读取用户的邮件
  - POP: 邮局协议[RFC 1939]: 认证、下载，默认TCP端口号110
  - IMAP: 因特网邮件访问协议[RFC 1730]:更多的功能，包括直接上邮件服务器上对邮件操作，默认TCP端口号143
  - HTTP: 基于web的邮箱

POP3协议

认证阶段

- 客户端命令:
  - **user:** 用户名
  - **pass:** 密码
- 服务器响应
  - **+OK**
  - **-ERR**

事务阶段, 客户端:

- **list:** 列出邮件消息编号
- **retr:** 按编号下载邮件消息
- **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和IMAP

POP3

- 之前例子使用“下载并删除模式”
  - 用户改变客户端后无法再看到邮件
- POP3的“下载并保持”模式：不同客户端都可以持有邮件副本
- POP3在不同会话之间是无状态的
  - 在一个客户端对邮件副本的操作并不会影响其它客户端上的副本

IMAP

- 所有邮件消息存放在邮件服务器
- 允许用户使用文件夹在服务器上组织邮件
- 在多个会话间保持状态：
  - 文件夹的名称和邮件与文件夹的映射关系在多个客户端上保持一致

目录

DNS：域名系统domain name system

人有许多ID:

- 名字、身份证号、 学号等等

因特网上的主机和路由器:

- IP地址(32 bit) - 用于数据包路由寻址
- “名字”,例如, www.baidu.com – 用于人类记忆

问: 如何将名字映射为IP地址?或者反向映射?

域名系统:

- 一个有众多域名服务器层次化组织起来的分布式数据库
- 应用层协议: 主机和域名服务器通过协议实现名字和IP地址之间的翻译（称为解析）
  - 因特网的核心功能，但是在应用层上实现
  - 名字服务器部署在网络边缘

DNS：结构和服务

DNS提供的服务

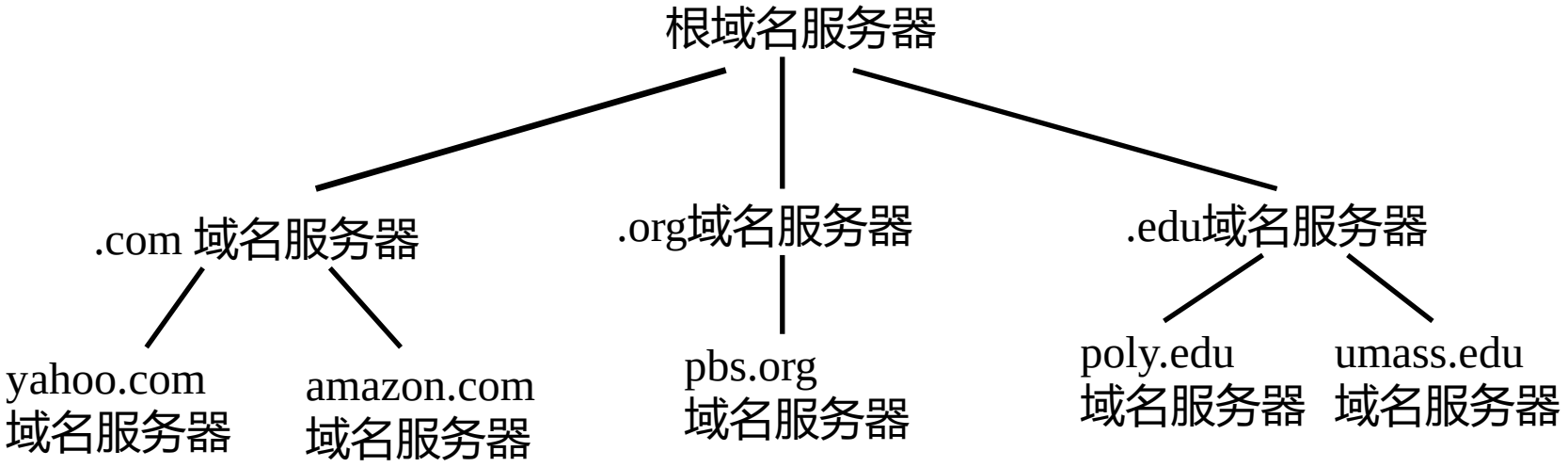
- 将主机名翻译为IP地址
- 映射主机别名
  - 规范名、别名
- 邮件服务器别名
- 负载均衡
  - 一个名字对应多个IP地址和副本服务器
  - www.baidu.com

集中式DNS?

- 单点故障
- 域名解析产生大量的流量
- 距离远，解析时延长
- 难以维护

结论：不具有可扩展性

DNS：分布式分层数据库

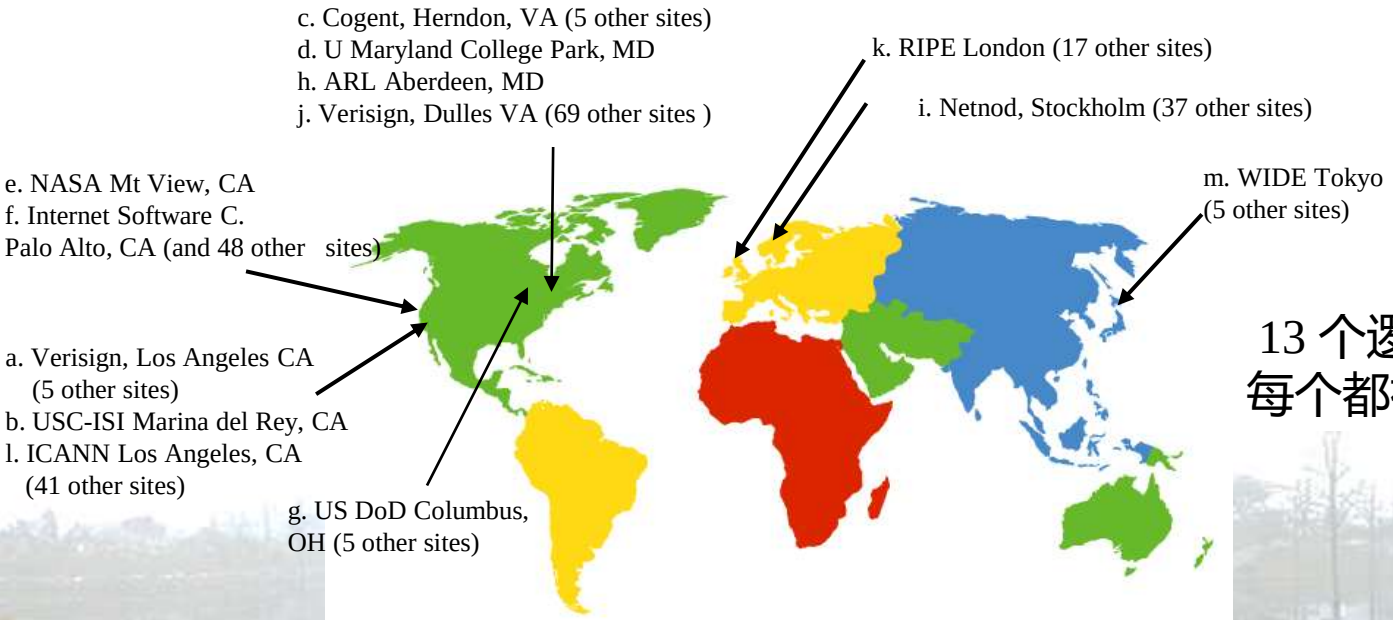


客户端希望获得www.amazon.com的IP地址：

- 客户端向根域名服务器查询，获得.com 域名服务器地址
- 客户端向.com的顶级域名服务器查询，获得amazon.com域名服务器地址
- 客户端向amazon.com的权威域名服务器查询，获得www.amazon.com的IP地址

DNS：根域名服务器

- 当本地域名服务器不能解析域名，联系根域名服务器



13 个逻辑根域名服务器，每个都被大量备份

顶级域名服务器和权威域名服务器

顶级域名 (TLD) 服务器:

- 负责com、org、net、edu、aero、jobs、museums，以及所有国家顶级域名的解析
  - 国家顶级域名: cn、uk、fr、ca、jp
- 公共网络服务维护.com顶级域名服务器
- 公共教育部门维护.edu顶级域名服务器

权威域名服务器:

- 机构自己的域名服务器，提供主机名到IP地址的权威映射
- 由机构或者网络服务提供商维护

本地域名服务器

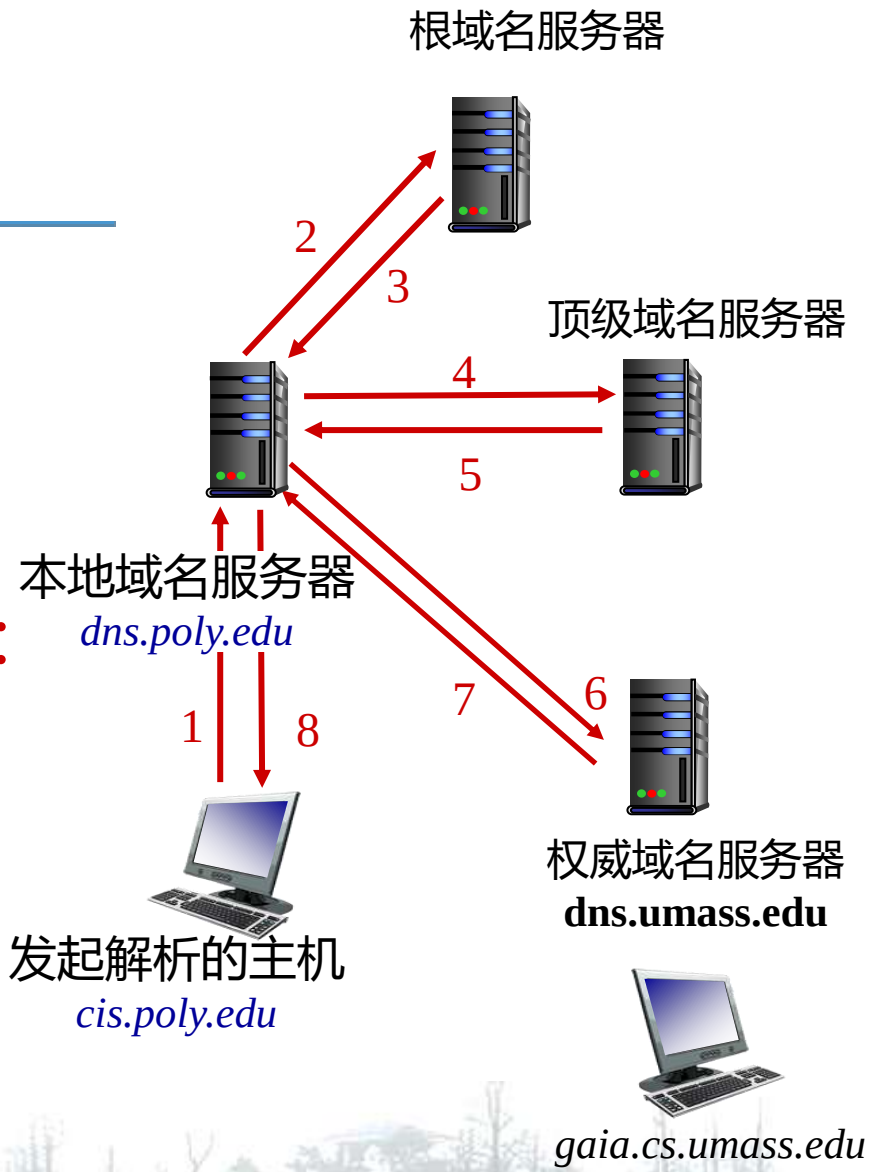
- 并不属于上述层次结构
- 每个ISP (本地电信商、企业、大学)都有一个
  - 有时被称为缺省域名服务器，配置到主机操作系统
- 当主机发起DNS查询，查询消息送到本地域名服务器
  - 本地域名服务器缓存最近解析的名字-地址映射（缓存的映射可能过期）
  - 作为代理，向层级结构的DNS体系转发域名查询

域名解析示例

- 在cis.poly.edu的主机希望获得gaia.cs.umass.edu的IP地址

迭代查询（本地域名服务器）：

- 被询问的服务器返回下一步需要询问的服务器名
- “我不知道这个名字对应的IP地址，不过你可以问这个服务器”



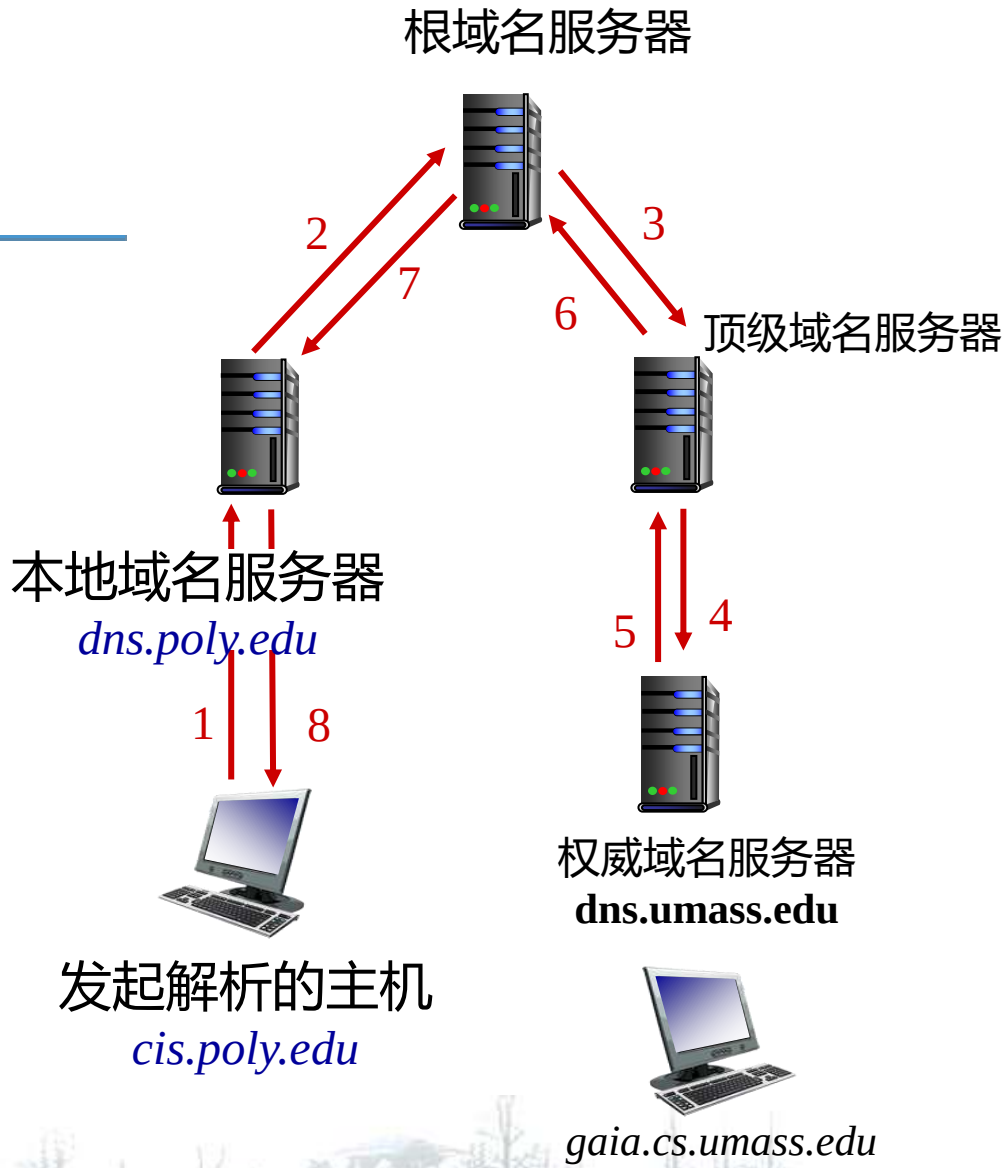
域名解析示例

递归解析：

- 解析的通信负载在域名服务器上
- 层次越高，负载越大

实际解析过程：

- 主机到本地域名服务器是递归的
- 剩余过程是迭代的
- 如上页图所示



DNS：缓存和更新记录

- 域名服务器缓存所有获取的名字映射记录
  - 缓存条目在一定时间（TTL：time to live）后过期删除
  - 本地域名服务器通常缓存顶级域名服务器的映射记录
    - 因此无需访问根域名服务器
- 缓存的“名字-地址”映射可能过期(尽力而为的服务)
  - 如果主机改变了IP地址，这一改变需要等待所有相关缓存过期，该主机才能被整个因特网访问
- IETF标准定义怎样更新缓存
  - RFC 2136

DNS映射记录

DNS: 分布式数据库存储资源记录 (RR)

RR 格式: `(name, value, type, ttl)`

type=A

- **name** 是主机名
- **value** 是IP地址

type=NS

- **name** 是域名(例如,alibaba.com)
- **value** 是这个域的权威域名服务器的主机名

type=CNAME

- **name**是主机的别名
- **value**是主机的规范名
- **www.ibm.com** 的规范名是 **servereast.backup2.ibm.com**

type=MX

- **value**是**name**表示的邮件服务器的规范名

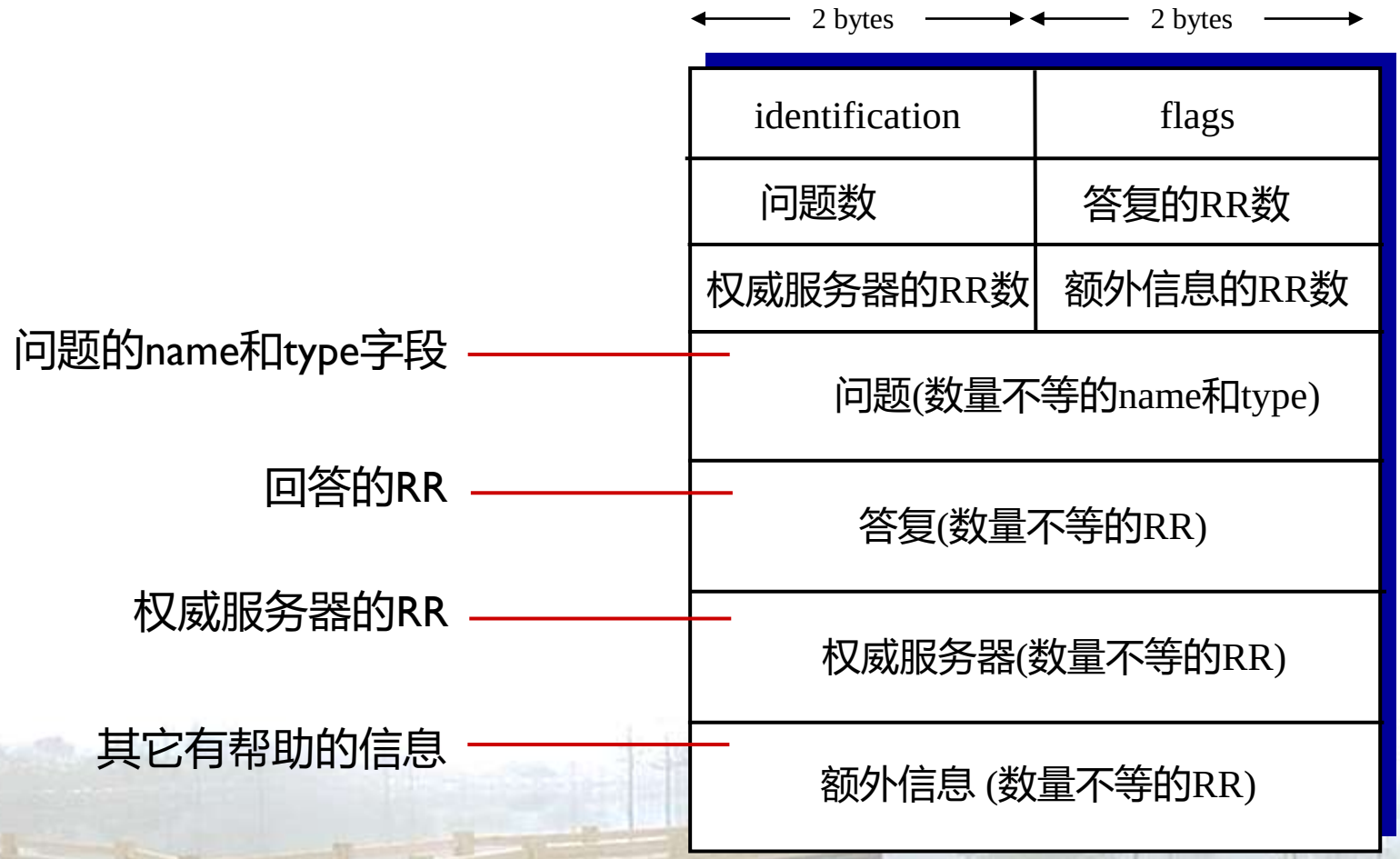
DNS协议和消息

- 两类消息：查询（query）和回复（reply），格式相同

消息头部

- identification: 16 bit数组 用于匹配query和reply
- flags:
  - query 或reply
  - 是否要求递归
  - 是否支持递归
  - Reply是否来自权威域名服务器（还是缓存）

DNS协议和消息


2 bytes
2 bytes
identification
flags
问题数
答复的RR数
权威服务器的RR数
额外信息的RR数
问题的name和type字段
问题(数量不等的name和type)
回答的RR
答复(数量不等的RR)
权威服务器的RR
权威服务器(数量不等的RR)
其它有帮助的信息
额外信息 (数量不等的RR)

将RR添加到DNS

- 例如: 初创企业“Network Utopia”
- 在注册服务商注册域名networkuptopia.com
  - 注册服务商提供权威服务器的名字和IP地址（权威服务器可以有主备）
  - 注册服务商向.com顶级域名服务器添加两条RR:
    (networkutopia.com, dns1.networkutopia.com, NS)
    (dns1.networkutopia.com, 212.212.212.1, A)
- 在权威域名服务器上创建www.networkuptopia.com的类型A的RR; 创建networkutopia.com的类型MX的RR
  - **(www.networkutopia.com, 212.212.212.2, A)**
    (mail.networkutopia.com, smtp1.networkutopia.com, MX)

对DNS的攻击

DDoS 攻击

- 对根服务器流量轰炸
  - 没有成功先例
  - 根服务器流量过滤
  - 本地域名服务器缓存顶级域名服务器的IP地址，从而绕过了根服务器
- 流量轰炸顶级域名服务器
  - 更具威胁

重定向攻击

- 中间人攻击
  - 截获域名解析请求
- DNS下毒
  - 发送虚假的响应，污染本地域名服务器的缓存

利用DNS发起DDoS攻击

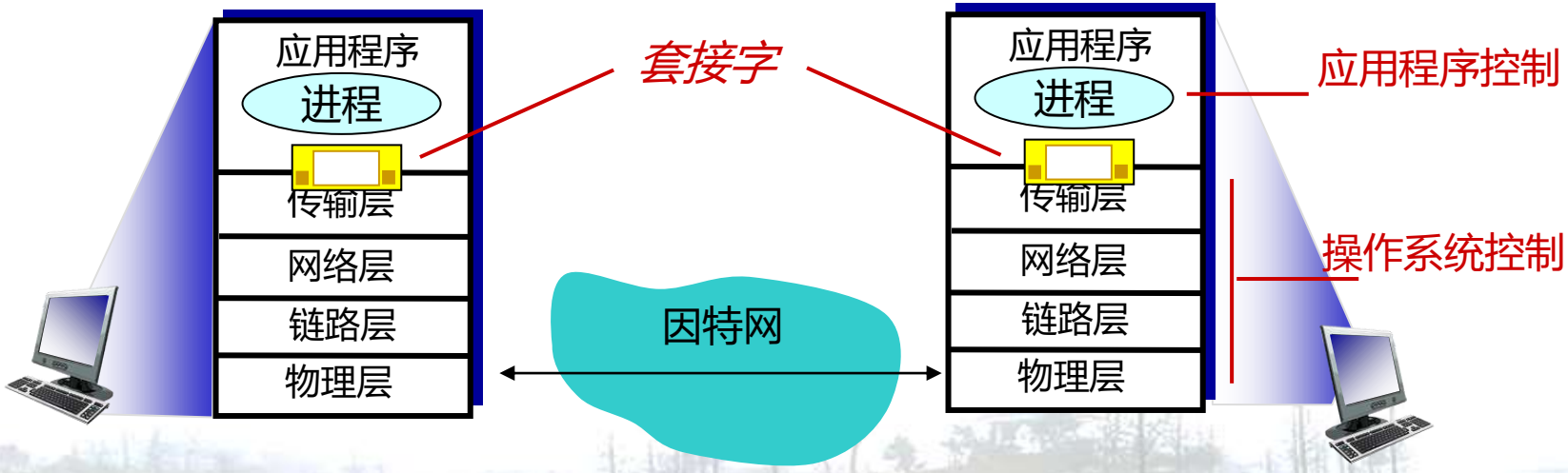
- 使用攻击目标的IP地址为源地址伪造解析请求
- 目标地址收到大量响应

目录

套接字编程

目的：客户端和服务器如何通过套接字通信

套接字（socket）：应用程序进程之间端到端通信的“门”



套接字编程

两类套接字对应两类传输层服务:

- UDP: 不可靠的数据报
- TCP: 可靠的byte流

示例程序:

1. 客户端程序读取键盘输入的字符串，向服务器程序发送
2. 服务器程序接收字符串，转换成大写
3. 服务器程序将修改后的字符串发给客户端程序
4. 客户端程序将收到的字符串显示在屏幕上

UDP套接字编程

UDP:客户端和服务器之间无连接

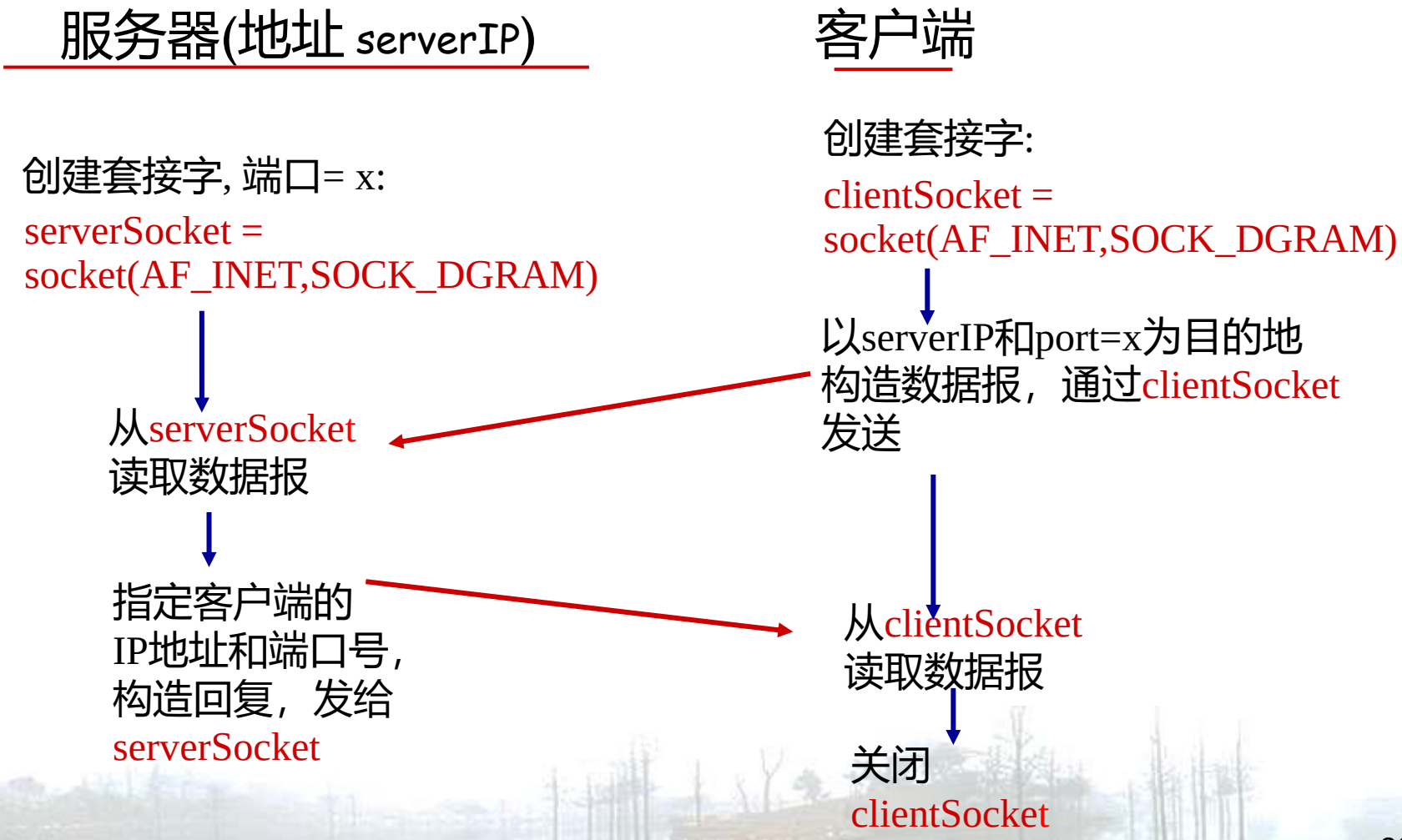
- 数据传输前无需握手
- 发送每个数据包时明确指定目的地IP地址和端口号
- 接收端抽取UDP数据包中发送端的IP地址和端口号，以便回复

UDP: 传输中的数据包可能丢失或者乱序到达

从应用程序角度:

- UDP 提供了一种不可靠的客户端和服务器之间的数据传输方式（以一个个数据报datagram的形式）

客户端/服务器之间的socket交互：UDP



应用例程：UDP客户端

Python UDP客户端

包含套接字库

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

创建与服务器通信的UDP套接字

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

获取键盘输入

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

指定服务器名和端口号，通过套接字发送数据

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

从套接字读取服务器修改的字符串

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

打印至屏幕并关闭套接字

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

应用例程：UDP服务器端

Python UDP服务器端

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ( "The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

创建UDP 套接字

将套接字绑定到本地12000端口

死循环，服务器端永远在线

从套接字读取消息，获取客户端的IP地址和端口号

将改写后的字符串发给客户端

TCP套接字编程

客户端主动连接服务器端

- 服务器端进程先运行
- 服务器端必须先创建套接字，以便客户端连接

客户端通过以下方式连接服务器端

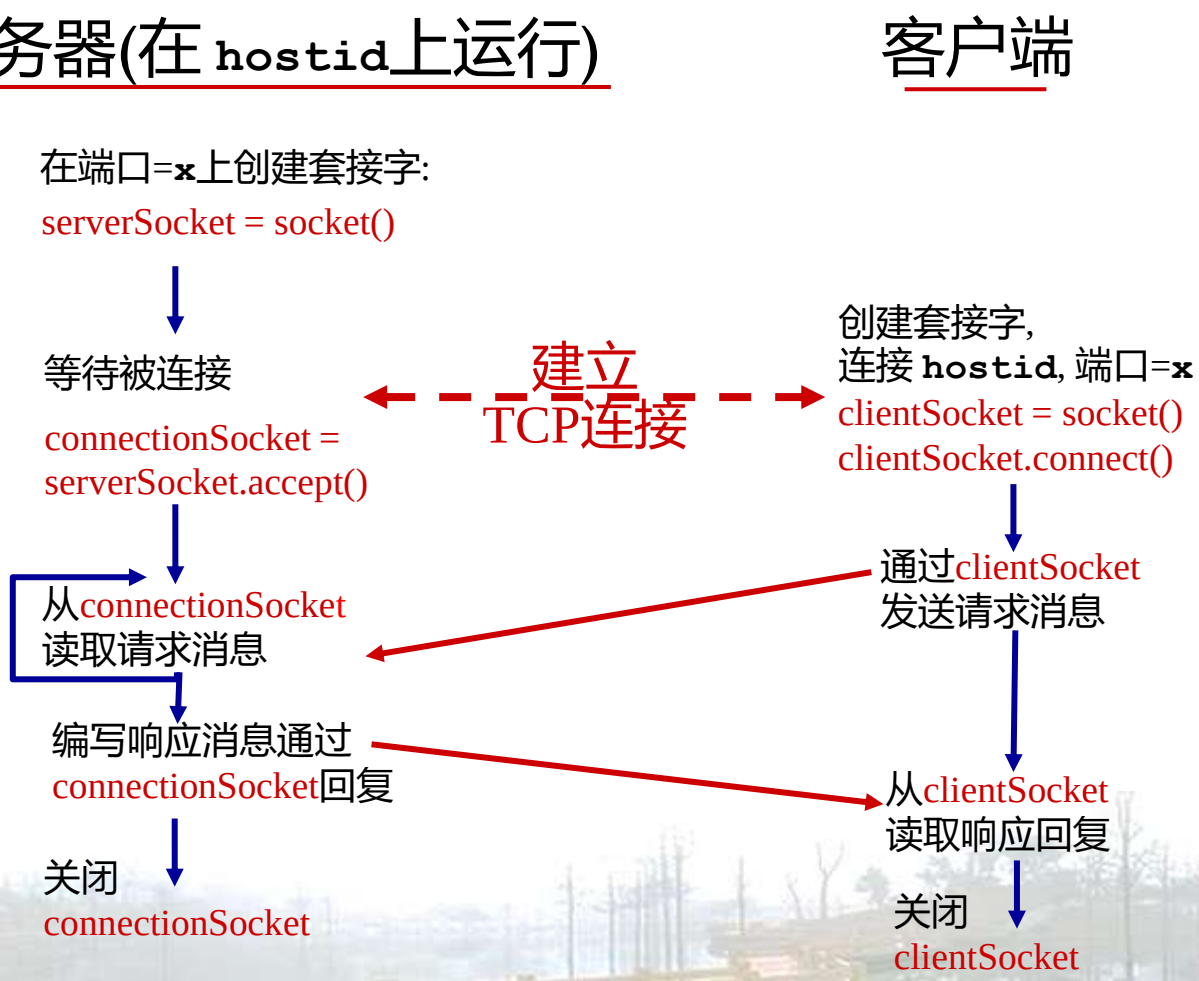
- 创建TCP套接字，指定服务器端的IP地址和端口号
- 客户端创建套接字: 与服务器端建立TCP连接

- 当被客户端连接，服务器端创建新的套接字，用于和该客户端通信
  - 服务器端可以和多个客户端同时通信
  - 使用地址+端口号区分不同的客户端

从应用程序角度:

TCP提供了客户端和服务器端之间一种可靠、顺序的字节流管道

客户端/服务器之间的socket交互：TCP



应用例程：TCP客户端

TCP客户端

```
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()
```

创建连接服务器端的TCP套接字，端口号 12000

不再需要指定服务器地址和端口号

应用例程：TCP服务器端

Python TCP服务器端

```
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

服务器端开始在套接字上监听连接请求

死循环，永远在线

服务器端使用accept()等待并接受连接请求，同时创建新的套接字用于客户端通信

读取字符串（但此时无需提取客户端地址和端口）

关闭与客户端通信的套接字(不是欢迎套接字)