传输层和应用层中厅讲座

2019年1月12日 20:23

传输层

传输层要点

❖理解传输层服务的基本理论和基本机制

- 复用/分用

❖ Why?

❖ 如果某层的一个协议对应直接上层的多个协议/实体，则需要复用/分用

接收端进行多路分用:

传输层依据头部信息将收到的Segment交给正确的Socket，即不同的进程

如何工作

❖主机接收到IP数据报(datagram)

- 每个数据报携带源IP地址、目的IP地址。
- 每个数据报携带一个传输层的段(Segment）。
- 每个段携带源端口号和目的端口号

❖主机收到Segment之后，传输层协议提取IP地址和端口号信息，将Segment导向相应的Socket

- TCP做更多处理

TCP/UDP 段格式

无连接分用

❖利用端口号创建Socket

DatagramSocket mySocket1 = new DatagramSocket(99111);
DatagramSocket mySocket2 = new DatagramSocket(99222);

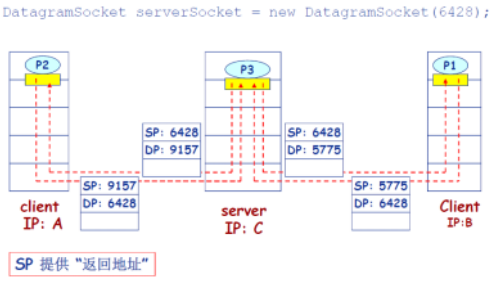
❖UDP的Socket用二元组标识

- (目的IP地址，目的端口号)

❖主机收到UDP段后

- 检查段中的目的端口号
- 将UDP段导向绑定在该端口号的Socket

❖ 来自不同源IP地址和/或源端口号的IP数据包被导向同一个Socket



面向连接的分用

❖TCP的Socket用四元组标识

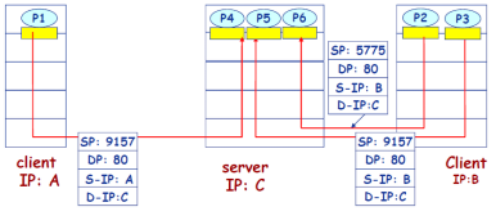
- 源IP地址
- 源端口号
- 目的IP地址
- 目的端口号

❖接收端利用所有的四个值将Segment导向合适的Socket

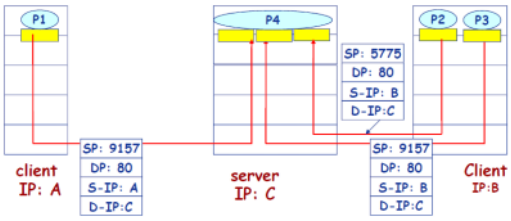
❖服务器可能同时支持多个TCP Socket

- 每个Socket用自己的四元组标识

❖ Web服务器为每个客户端开不同的Socket

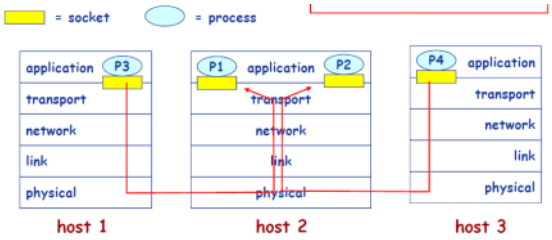


面向连接的分用：多线程Web服务器



发送端进行多路复用:

从多个Socket接收数据，为每块数据封装上头部信息，生成Segment，交给网络层



- 可靠数据传输机制
- 流量控制机制
- 拥塞控制机制

❖掌握Internet的传输层协议

- UDP：无连接传输服务User Datagram Protocol

UDP为什么存在?

❖ 无需建立连接(减少延迟)

❖ 实现简单：无需维护连接状态

❖ 头部开销少

❖ 没有拥塞控制: 应用可更好地控制发送时间和速率

❖ 基于Internet IP协议

- 复用/分用
- 简单的错误校验

把IP服务裸露暴露给应用层

❖Best effort”服务，UDP段可能

- 丢失
- 非按序到达

❖无连接

- UDP发送方和接收方之间不需要握手

- 每个UDP段的处理独立于其他段

❖常用于流媒体应用

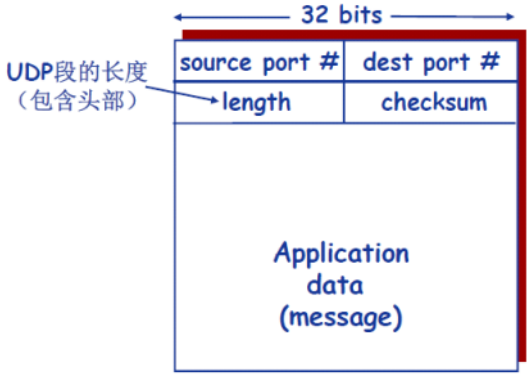
- 容忍丢失
- 速率敏感

❖UDP还用于

- DNS
- SNMP

❖在UDP上实现可靠数据传输?

- 在应用层增加可靠性机制
- 应用特定的错误恢复机制



UDP segment format

UDP校验和(checksum)

目的：检测UDP段在传输中是否发生错误（如位翻转）

❖发送方

- 将段的内容视为16-bit整数
- 校验和计算：计算所有整数的和，进位加在和的后面，将得到的值按位求反，得到校验和
- 发送方将校验和放入校验和字段

❖接收方

- 计算所收到段的校验和
- 将其与校验和字段进行对比
- 不相等：检测出错误
- 相等：没有检测出错误（但可能有错误）

校验和计算示例

❖注意：

- 最高位进位必须被加进去

❖示例：

- TCP：面向连接的传输服务
- TCP拥塞控制

传输层服务概述

❖传输层协议为运行在不同Host上的进程提供了一种**逻辑通信机制(端到端)**

❖端系统运行传输层协议

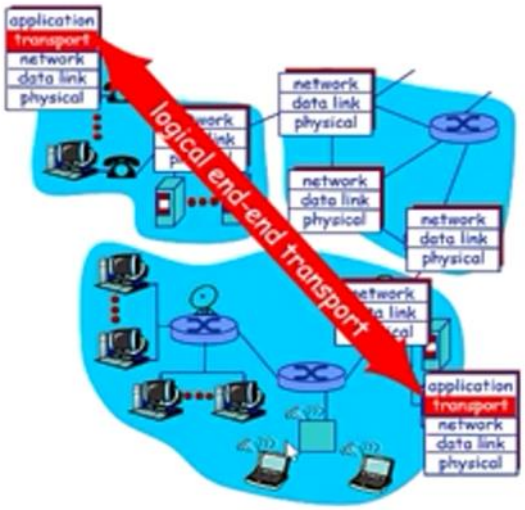
- 发送方：将应用递交的消息分成一个或多个的Segment，并向下传给网络层。
- 接收方：将接收到的segment组装成消息，并向上交给应用层。

❖传输层可以为应用提供多种协议

- Internet上的TCP
- Internet上的UDP

传输层vs. 网络层

- ❖网络层：提供**主机**之间的逻辑通信机制
- ❖传输层：提供**应用进程**之间的逻辑通信机制
  - ▪位于网络层之上
  - ▪依赖于网络层服务
  - ▪对网络层服务进行（可能的）增强

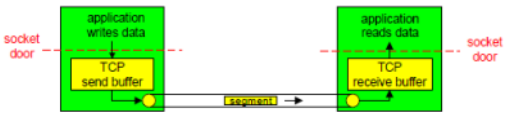


Internet传输层协议

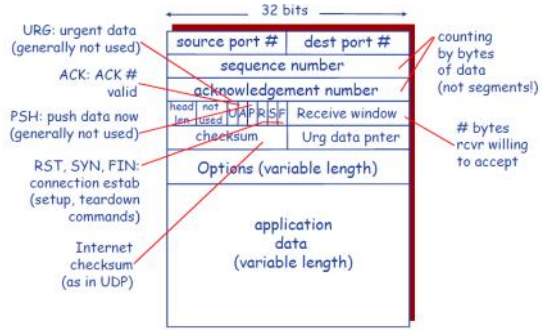
- ❖可靠、按序的交付服务(TCP)
  - ▪拥塞控制
  - ▪流量控制
  - ▪连接建立
- ❖不可靠的交付服务(UDP)
  - ▪基于“**尽力而为(Best-effort)**”的网络层，没有做（可靠性方面的）扩展
- ❖两种服务均不保证
  - ▪延迟
  - ▪带宽

重点TCP

TCP概述



TCP段结构



TCP: 序列号和ACK

序列号:

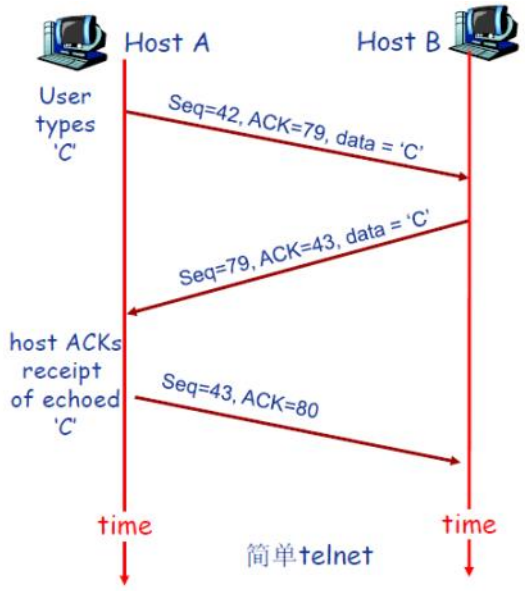
- ▪序列号指的是segment中第一个字节的编号，而不是segment的编号
- ▪建立TCP连接时，双方随机选择序列号

ACKs:

- 希望接收到的下一个字节的序列号
- 累计确认：该序列号之前的所有字节均已被正确接收到

Q: 接收方如何处理乱序到达的Segment?

- A: TCP规范中没有规定，由TCP的实现者做出决策



TCP可靠数据传输

- ❖TCP在IP层提供的不可靠服务基础上实现可靠数据传输服务
- ❖流水线机制
- ❖累积确认
- ❖TCP使用单一重传定时器
- ❖触发重传的事件
  - 超时
  - 收到重复ACK
- ❖渐进式
  - 暂不考虑重复ACK
  - 暂不考虑流量控制
  - 暂不考虑拥塞控制

TCP RTT和超时

- ❖问题：如何设置定时器的超时时间?
- ❖大于RTT
  - 但是RTT是变化的
- ❖过短：
  - 不必要的重传
- ❖过长：
  - 对段丢失时间反应慢

❖ 问题：如何估计RTT?

❖ SampleRTT: 测量从段发出去到收到ACK的时间

- 忽略重传

❖ SampleRTT变化

- 测量多个SampleRTT，求平均值，形成RTT的估计值 EstimatedRTT

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

指数加权移动平均

典型值：0.125

定时器超时时间的设置：

- EstimatedRTT + "安全边界"
- EstimatedRTT变化大➔较大的边界

测量RTT的变化值：SampleRTT与EstimatedRTT的差值

DevRTT = (1- β)*DevRTT + β *|SampleRTT-EstimatedRTT|

(typically, β = 0.25)

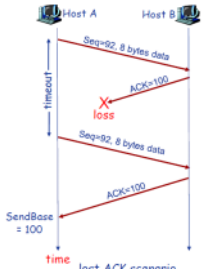
定时器超时时间的设置：

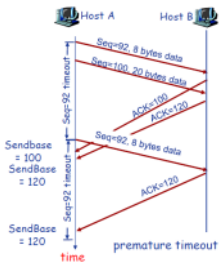
TimeoutInterval = EstimatedRTT + 4*DevRTT

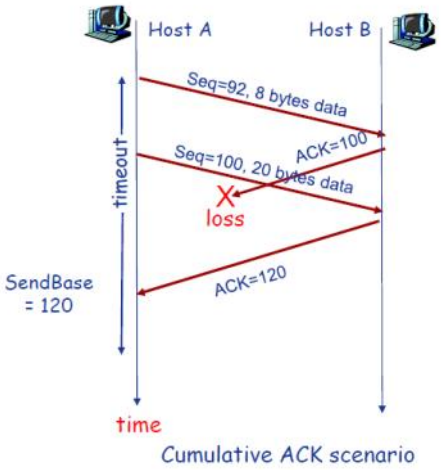
TCP发送方事件

- ❖从应用层收到数据
  - ▪ 创建Segment
  - ▪ 序列号是Segment第一个字节的编号
  - ▪ 开启计时器
  - ▪ 设置超时时间：TimeOutInterval
- ❖超时
  - ▪ 重传引起超时的Segment
  - ▪ 重启定时器
- ❖收到ACK
  - ▪ 如果确认此前未确认的Segment
  - • 更新SendBase
  - • 如果窗口中还有未被确认的分组，重新启动定时器

TCP重传示例







Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send *duplicate ACK*, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment startsat lower end of gap

快速重传机制

❖TCP的实现中，如果发生超时，超时时间间隔将重新设置，即将超时时间间隔加倍，导致其很大

- 重发丢失的分组之前要等待很长时间

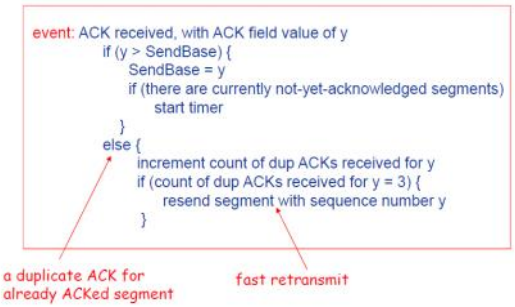
❖通过重复ACK检测分组丢失

- Sender会背靠背地发送多个分组
- 如果某个分组丢失，可能会引发多个重复的ACK

❖如果sender收到对同一数据的3个ACK，则假定该数据之后的段已经丢失

- 快速重传：在定时器超时之前即进行重传

快速重传算法



TCP流量控制

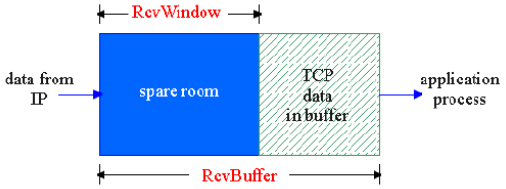
❖接收方为TCP连接分配buffer

上层应用可能处理buffer中数据的速度较慢

Flow control

发送方不会传输的太多、太快以至于淹没接收方

❖速度匹配机制



(假定TCP receiver丢弃乱序的segments)

❖ Buffer中的可用空间(spareroom)

= RcvWindow

= RcvBuffer-[LastByteRcvd -LastByteRead]

❖ Receiver通过在Segment的头部字段将RcvWindow 告诉Sender

❖ Sender限制自己已经发送的但还未收到ACK的数据不超过接收方的空闲RcvWindow尺寸

❖ Receiver告知SenderRcvWindow=0,会出现什么情况?

TCP连接管理

❖TCP sender和receiver在传输数据前需要建立连接

❖初始化TCP变量

- Seq. #
- Buffer和流量控制信息

❖Client：连接发起者

Socket clientSocket = new Socket("hostname","port number");

❖Server: 等待客户连接请求

Socket connectionSocket =welcomeSocket.accept();

Three way handshake:

Step 1: client host sends TCP SYN segment to server

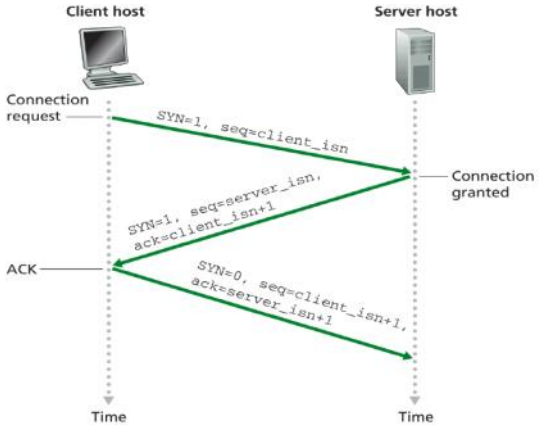
- specifies initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment

- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

TCP连接管理：建立



TCP连接管理：关闭

Closing a connection:

client closes socket: clientSocket.close();

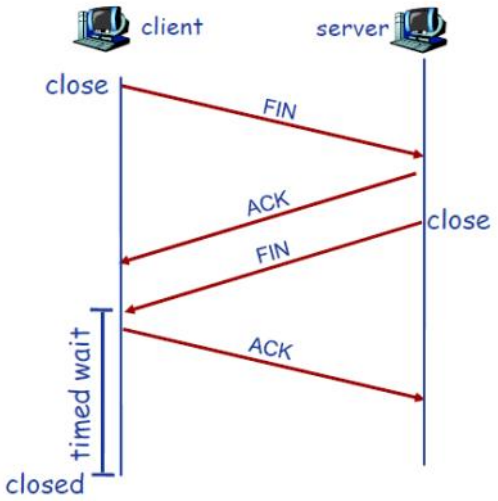
Step 1: client向server发送TCP FIN 控制segment

Step 2: server 收到FIN, 回复ACK. 关闭连接, 发送FIN.

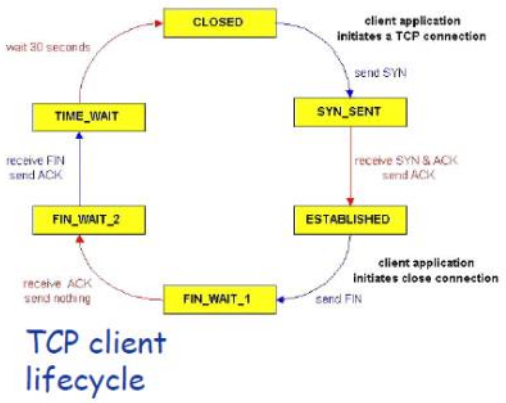
Step 3: client 收到FIN, 回复ACK.

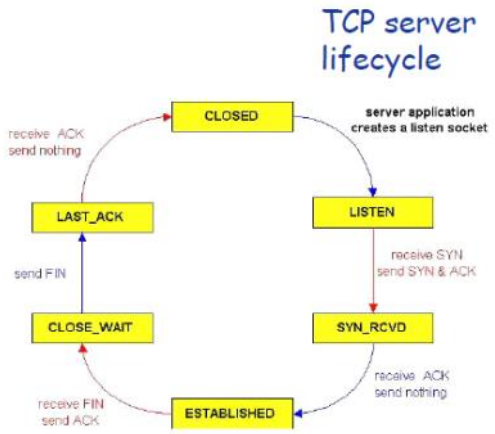
- 进入 “等待” –如果收到FIN，会重新发送ACK

Step 4: server收到ACK. 连接关闭.



TCP连接管理





TCP拥塞控制

基本原理

❖Sender限制发送速率

LastByteSent-LastByteAcked<= CongWin

$$rate \approx \frac{CongWin}{RTT} \text{ Bytes/sec}$$

❖CongWin:

- 动态调整以改变发送速率
- 反映所感知到的网络拥塞

问题：如何感知网络拥塞?

❖Loss事件=timeout或3个重复ACK

❖发生loss事件后，发送方降低速率

如何合理地调整发送速率?

❖加性增—乘性减: AIMD

❖慢启动: SS

加性增—乘性减: AIMD

❖原理：逐渐增加发送速率，谨慎探测可用带宽，直到发生loss

❖方法: AIMD

- Additive Increase: 每个RTT将CongWin增大一个MSS——拥塞避免
- Multiplicative Decrease: 发生loss后将CongWin减半

锯齿行为: 探测可用带宽

TCP慢启动: SS

❖TCP连接建立时，CongWin=1

- 例：MSS=500 byte,RTT=200msec
- 初始速率=20k bps

❖可用带宽可能远远高于初始速率:

- 希望快速增长

❖原理:

- 当连接开始时，指数性增长

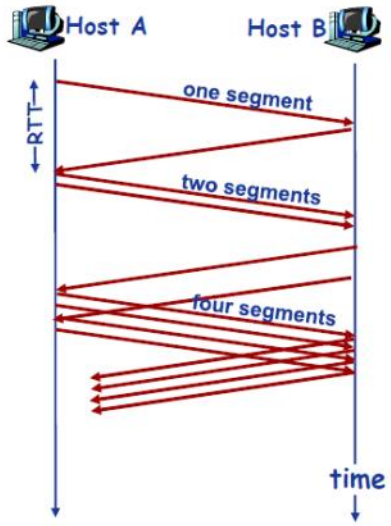
Slowstart algorithm

initialize: Congwin = 1
for (each segment ACKed)
    Congwin++
until (loss event OR
    CongWin > threshold)

❖指数性增长

- 每个RTT将CongWin翻倍
- 收到每个ACK进行操作

❖初始速率很慢，但是快速攀升



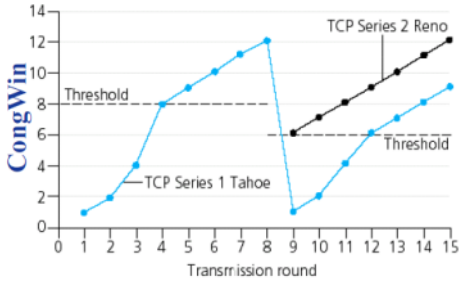
Threshold变量

Q:何时应该指数性增长切换为线性增长(拥塞避免)?

A: 当CongWin达到Loss事件前值的1/2时.

实现方法:

- ❖ 变量Threshold
- ❖ Loss事件发生时, Threshold被设为Loss事件前CongWin值的1/2。



Loss事件的处理

❖ 3个重复ACKs:

- CongWin切到一半
- 然后线性增长

❖ Timeout事件:

- CongWin直接设为1个MSS
- 然后指数增长
- 达到threshold后, 再线性增长

Philosophy:

- 3个重复ACKs表示网络还能够传输一些 segments
- timeout事件表明拥塞更为严重

TCP拥塞控制: 总结

When CongWin is below Threshold, sender in slow-start phase, window grows exponentially.

- When CongWin is above Threshold, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, Threshold set to CongWin/2 and CongWin set to Threshold.
- When timeout occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS.

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP拥塞控制算法

```
Th = ?
CongWin = 1 MSS
/* slow start  or exponential increase */
While (No Packet Loss and CongWin < Th) {
    send CongWin TCP segments
    for each ACK increase CongWin by 1
}
/* congestion avoidance or linear increase */
While (No Packet Loss) {
        send CongWin TCP segments
        for CongWin ACKs, increase CongWin by 1
}
Th = CongWin/2
If (3 Dup ACKs) CongWin = Th;
If (timeout)   CongWin=1;
```

TCP性能分析

TCP throughput: 吞吐率

- 给定拥塞窗口大小和RTT，TCP的平均吞吐率是多少?
  - 忽略掉Slow start
- 假定发生超时时CongWin的大小为W，吞吐率是W/RTT
- 超时后，CongWin=W/2，吞吐率是W/2RTT
- 平均吞吐率为：0.75W/RTT

未来的TCP

- 举例：每个Segment有1500个byte, RTT是100ms，希望获得10Gbps的吞吐率
  - throughput = W*MSS*8/RTT, 则
  - W=throughput*RTT/(MSS*8)
  - throughput=10Gbps, 则W=83,333
- 窗口大小为83,333

- 吞吐率与丢包率(loss rate, L)的关系
  - CongWin从W/2增加至W时出现第一个丢包，那么一共发送的分组数为 $W/2+(W/2+1)+(W/2+2)+\ldots+W = 3W^2/8+3W/4$
  - W很大时，$3W^2/8 >> 3W/4$，因此 $L \approx 8/(3W^2)$

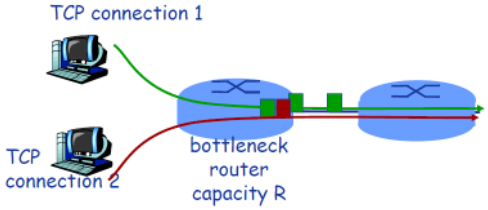
$$W = \sqrt{\frac{8}{3L}} \qquad Throughput = \frac{0.75 \cdot MSS \cdot \sqrt{\frac{8}{3L}}}{RTT} \approx \frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

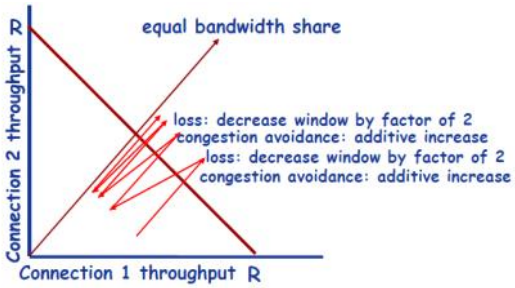
- $L = 2 \cdot 10^{-10}$ Wow!!!
- 高速网络下需要设计新的TCP

TCP的公平性

❖公平性?

▪ 如果K个TCP Session共享相同的瓶颈带宽R，那么每个Session的平均速率为R/K





❖公平性与UDP

▪ 多媒体应用通常不使用TCP，以免被拥塞控制机制限制速率

▪ 使用UDP：以恒定速率发送，能够容忍丢失

▪ 产生了不公平

❖研究：TCP friendly

❖公平性与并发TCP连接

▪ 某些应用会打开多个并发连接

▪ Web浏览器

▪ 产生公平性问题

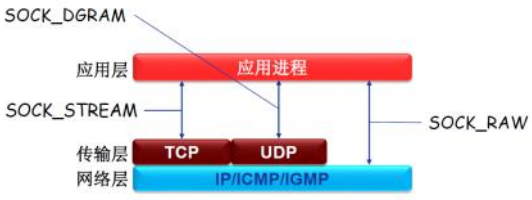
❖例子：链路速率为R，已有9个连接

▪ 新来的应用请求1个TCP，获得R/10的速率

▪ 新来的应用请求11个TCP，获得R/2的速率

应用层

❖Socket编程



❖TCP：可靠、面向连接、字节流传输、点对点

❖UDP：不可靠、无连接、数据报传输

API

❖WSAStartup: 初始化socket库(仅对WinSock)

❖WSACleanup: 清楚/终止socket库的使用 (仅对WinSock)

❖socket: 创建套接字

❖connect:“连接” 远端服务器 (仅用于客户端)

❖closesocket: 释放/关闭套接字

❖bind: 绑定套接字的本地IP地址和端口号（通常客户端不需要）

❖listen: 置服务器端TCP套接字为监听模式，并设置队列大小 (仅用于服务器端TCP套接字)

❖accept: 接受/提取一个连接请求，创建新套接字，通过新套接 (仅用于服务器端的TCP套接字)

❖recv: 接收数据（用于TCP套接字或连接模式的客户端UDP套接字）

❖ recvfrom: 接收数据报（用于非连接模式的UDP套接字）

❖ send: 发送数据（用于TCP套接字或连接模式的客户端UDP套接字）

- ❖ sendto:发送数据报（用于非连接模式的UDP套接字）
- ❖ setsockopt: 设置套接字选项参数
- ❖ getsockopt: 获取套接字选项参数套接字或连接模式的客户端UDP套接字）

❖Internet传输层服务模型

- TCP
  - 面向连接: 客户机/服务器进程间

    需要建立连接
  - 可靠的传输
  - 流量控制: 发送方不会发送速度过快，超过接收方的处理能力
  - 拥塞控制: 当网络负载过重时能够限制发送方的发送速度
  - 不提供时间/延迟保障
  - 不提供最小带宽保障

- UDP
  - 无连接
  - 不可靠的数据传输
  - 不提供:
    - 可靠性保障
    - 流量控制
    - 拥塞控制
    - 延迟保障
    - 带宽保障

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	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage,Dialpad)	typically UDP

❖特定网络应用及协议

- DNS

  概述

  ❖Internet上主机/路由器的识别问题
  - IP地址
  - 域名：www.hit.edu.cn

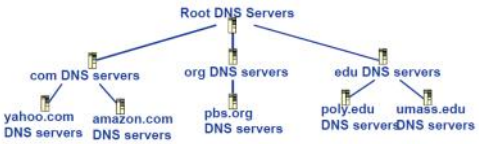
  ❖域名解析系统DNS
  - 多层命名服务器构成的分布式数据库
  - 应用层协议：完成名字的解析
    - Internet核心功能，用应用层协议实现
    - 网络边界复杂

  ❖DNS服务
  - 域名向IP地址的翻译
  - 主机别名
  - 邮件服务器别名
  - 负载均衡：Web服务器

  ❖ 问题：为什么不使用集中式的DNS?

- 单点失败问题
- 流量问题
- 距离问题
- 维护性问题

分布式层次式数据库



❖客户端想要查询www.amazon.com的IP

- 客户端查询根服务器，找到com域名解析服务器
- 客户端查询com域名解析服务器，找到amazon.com域名解析服务器
- 客户端查询amazon.com域名解析服务器，获得www.amazon.com的IP地址

域名服务器

DNS根域名服务器

❖本地域名解析服务器无法解析域名时，访问根域名服务器

❖根域名服务器

- 如果不知道映射，访问权威域名服务器
- 获得映射
- 向本地域名服务器返回映射

顶级和权威域名解析服务器

❖顶级域名服务器(TLD, top-level domain): 负责com, org, net,edu等顶级域名和国家顶级域名，例如cn, uk, fr等

- Network Solutions维护com顶级域名服务器
- Educause维护edu顶级域名服务器

❖ 权威(Authoritative)域名服务器：组织的域名解析服务器，提供组织内部服务器的解析服务

- 组织负责维护
- 服务提供商负责维护

本地域名解析服务器

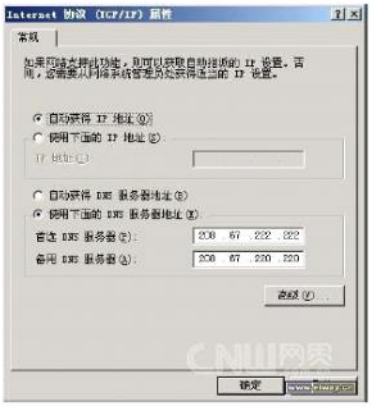
❖ 不严格属于层级体系

❖ 每个ISP有一个本地域名服务器

- 默认域名解析服务器

❖当主机进行DNS查询时，查询被发送到本地域名服务器

- 作为代理(proxy)，将查询转发给（层级式）域名解析服务器系统



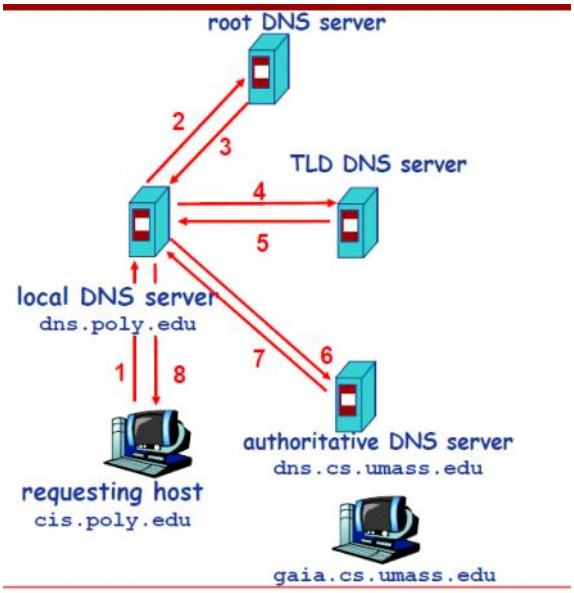
DNS查询示例

❖Cis.poly.edu的主机想获得

gaia.cs.umass.edu的IP地址

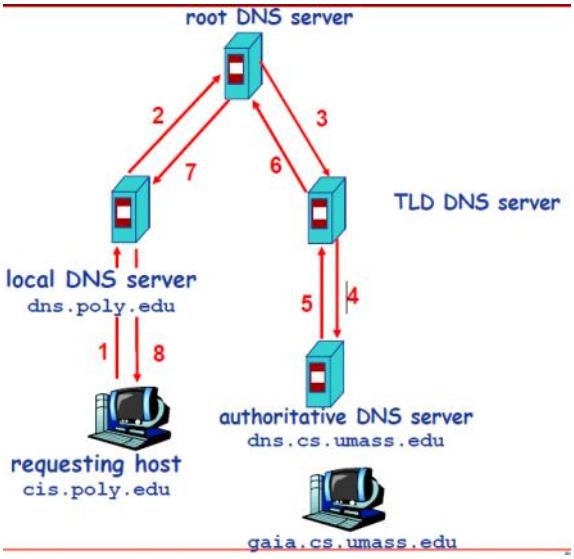
❖迭代查询

- 被查询服务器返回域名解析服务器的名字
- “我不认识这个域名，但是你可以问题这服务器”



❖递归查询

- 将域名解析的任务交给所联系的服务器



DNS记录

DNS记录缓存和更新

❖只要域名解析服务器获得域名—IP映射，即缓存这一映射

- 一段时间过后，缓存条目失效（删除）
- 本地域名服务器一般会缓存顶级域名服务器的映射
- 因此根域名服务器不经常被访问

❖ 记录的更新/通知机制

- RFC 2136
- Dynamic Updates in the Domain Name System (DNS UPDATE)

DNS记录和消息格式

❖资源记录(RR, resourcerecords)

❖Type=A

- Name: 主机域名
- Value: IP地址

❖Type=NS

- Name: 域(edu.cn)
- Value: 该域权威域名解析服务器的主机域名

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

❖Type=CNAME

- Name: 某一真实域名的别名
  - www.ibm.com – servereast.backup2.ibm.com
- Value: 真实域名

❖Type=MX

- Value是与name相对应的邮件服务器

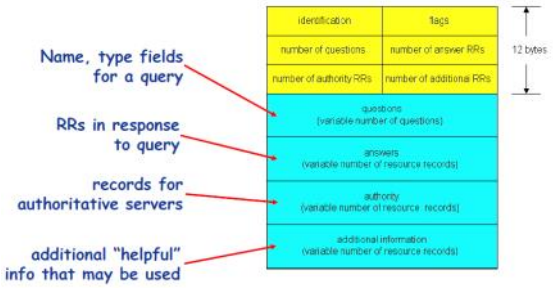
DNS协议与消息

❖DNS协议：

- 查询(query)和回复(reply消息)
- 消息格式相同

❖消息头部

- Identification: 16位查询编号，回复使用相同的编号
- flags
  - 查询或回复
  - 期望递归
  - 递归可用
  - 权威回答



- HTTP

Web与HTTP

World Wide Web: Tim Berners-Lee

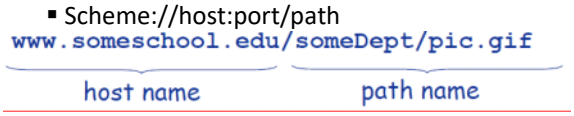
- 网页
- 网页互相链接

❖网页(Web Page)包含多个对象(objects)

- 对象：HTML文件、JPEG图片、视频文件、动态脚本等
- 基本HTML文件：包含对其他对象引用的链接

❖对象的寻址(addressing)

- URL(Uniform Resoure Locator)：统一资源定位器RFC1738
- Scheme://host:port/path



万维网应用遵循什么协议?

❖超文本传输协议

- HyperText Transfer Protocol

❖C/S结构

- 客户—Browser：请求、接收、展示Web

对象

- 服务器—Web Server：响应客户的请求，发送对象

❖HTTP版本：

- 1.0： RFC 1945
- 1.1： RFC 2068

❖使用TCP传输服务

- 服务器在80端口等待客户的请求
- 浏览器发起到服务器的TCP连接(创建套接字Socket)
- 服务器接受来自浏览器的TCP连接
- 浏览器(HTTP客户端)与Web服务器(HTTP服务器)交

换HTTP消息

- 关闭TCP连接

❖无状态(stateless)

- 服务器不维护任何有关客户端过去所发请求的信息

有状态的协议更复杂:

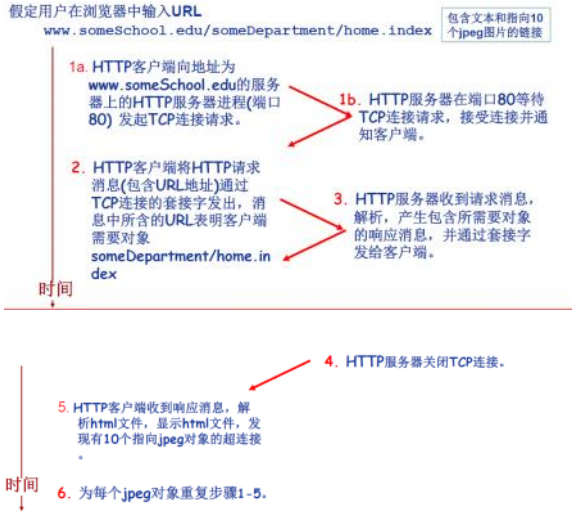
➢ 需维护状态(历史信息)

➢ 如果客户或服务器失效，会产生状态的不一致，解决这种不一致代价高

HTTP连接

❖非持久性连接(NonpersistentHTTP)

- 每个TCP连接最多允许传输一个对象
- HTTP 1.0版本使用非持久性连接



响应时间分析与建模

❖RTT(Round Trip Time)

▪ 从客户端发送一个很小的数据包到服务器并返回所经历的时间

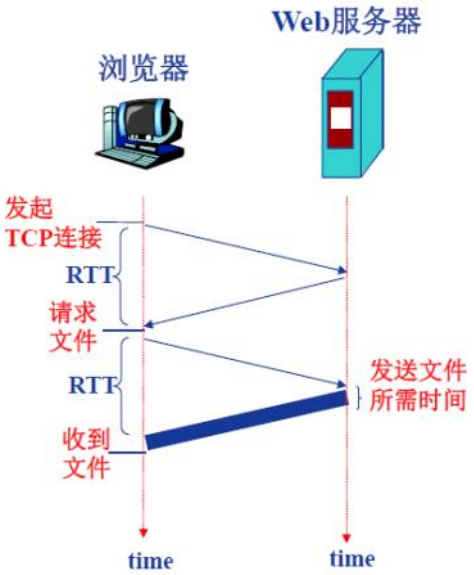
❖响应时间(Response time)

▪ 发起、建立TCP连接：1个RTT

▪ 发送HTTP请求消息到HTTP响应消息的前几个字节到达：1个RTT

▪ 响应消息中所含的文件/对象传输时间

▪ Total=2RTT +文件发送时间



非持久性连接的问题

▪ 每个对象需要2个RTT

▪ 操作系统需要为每个TCP连接开销资源(overhead)

▪ 浏览器会怎么做?

• 打开多个并行的TCP连接以获取网页所需对象

• 给服务器端造成什么影响?

❖持久性连接(Persistent HTTP)

▪ 每个TCP连接允许传输多个对象

▪ HTTP 1.1版本默认使用持久性连接

基本思想

发送响应后，服务器保持TCP连接的打开

▪ 后续的HTTP消息可以通过这个连接发送

❖无流水(pipelining)的持久性连接

▪ 客户端只有收到前一个响应后才发送新的请求

▪ 每个被引用的对象耗时1个RTT

❖带有流水机制的持久性连接

▪ HTTP 1.1的默认选项

▪ 客户端只要遇到一个引用对象就尽快发出请求

- 理想情况下，收到所有的引用对象只需耗时约1个RTT

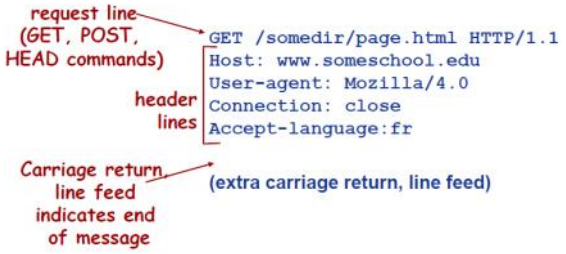
HTTP请求消息

❖HTTP协议有两类消息

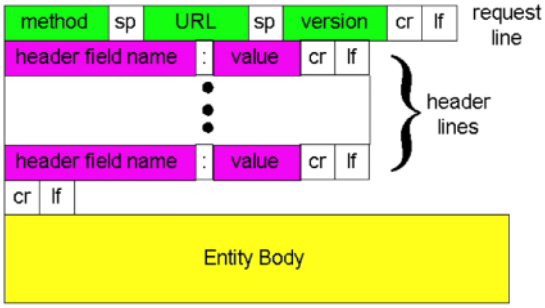
- 请求消息(request)
- 响应消息(response)

❖请求消息

- ASCII：人直接可读



HTTP请求消息的通用格式



上传输入的方法

❖POST方法

- 网页经常需要填写表格(form)
- 在请求消息的消息体(entity body)中上传客户端的输入

❖URL方法

- 使用GET方法
- 输入信息通过request行的URL字段上传

方法的类型

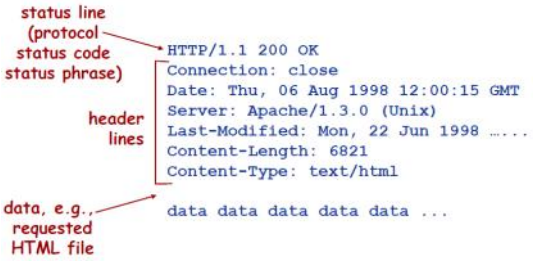
❖HTTP/1.0

- GET
- POST
- HEAD
  - 请Server不要将所请求的对象放入响应消息中

❖HTTP/1.1

- GET, POST, HEAD
- PUT
  - 将消息体中的文件上传到URL字段所指定的路径
- DELETE
  - 删除URL字段所指定的文件

HTTP响应消息



HTTP响应状态代码

❖响应消息的第一行

❖示例

- 200 OK
- 301 Moved Permanently
- 400 Bad Request
- 404 Not Found
- 505 HTTP Version Not Supported

cookie技术

为什么需要Cookie?

HTTP协议无状态

很多应用需要服务器掌握客户端的状态，如网上购物，如何实现?

Cookie技术

- 某些网站为了辨别用户身份、进行session跟踪而储存在用户本地终端上的数据（通常经过加密）。
- RFC6265

❖Cookie的组件

- HTTP响应消息的cookie头部行
- HTTP请求消息的cookie头部行
- 保存在客户端主机上的cookie文件，由浏览器管理
- Web服务器端的后台数据库

❖Cookie能够用于:

- 身份认证
- 购物车
- 推荐
- Web e-mail
- .....

❖隐私问题

Web缓存/代理服务器技术

❖功能

- 在不访问服务器的前提下满足客户端的HTTP请求。

❖为什么要发明这种技术?

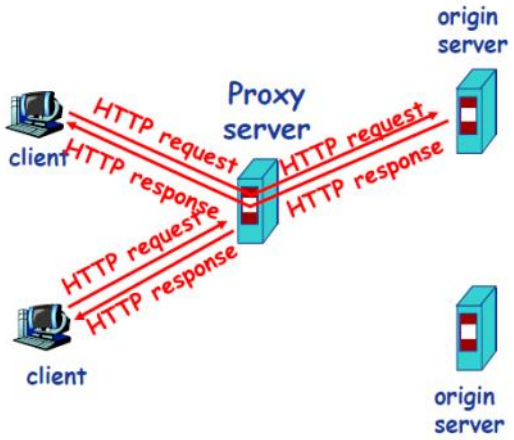
- 缩短客户请求的响应时间
- 减少机构/组织的流量
- 在大范围内(Internet)实现有效的内容分发

❖Web缓存/代理服务器

- 用户设定浏览器通过缓存进行Web访问
- 浏览器向缓存/代理服务器发送所有的HTTP请求
  - 如果所请求对象在缓存中，缓存返回对象
  - 否则，缓存服务器向原始服务器发送HTTP请求，获取对象，然后返回给客户端并保存该对象

❖缓存既充当客户端，也充当服务器

❖一般由ISP(Internet服务提供商)架设

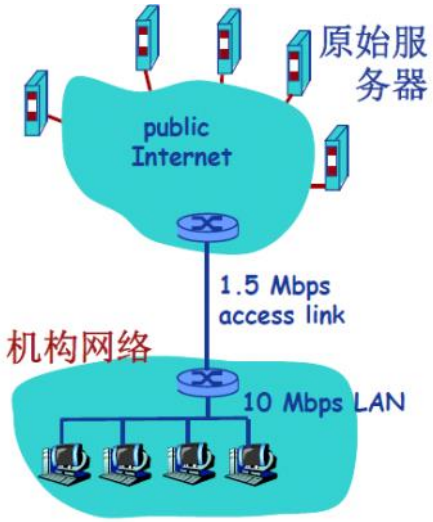


❖假定：

- 对象的平均大小=100,000比特
- 机构网络中的浏览器平均每秒有15个到原始服务器的请求
- 从机构路由器到原始服务器的往返延迟=2秒

❖网络性能分析：

- 局域网(LAN)的利用率=15%
- 接入互联网的链路的利用率=100%
- 总的延迟=互联网上的延迟+访问延迟+局域网延迟=2秒+几分钟+几微秒



❖解决方案1：

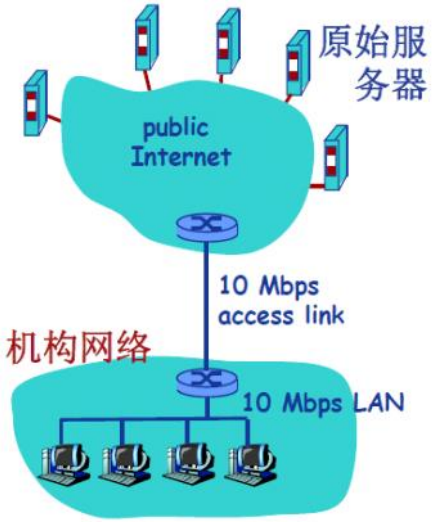
- 提升互联网接入带宽=10Mbps

❖网络性能分析：

- 局域网(LAN)的利用率=15%
- 接入互联网的链路的利用率=15%
- 总的延迟=互联网上的延迟+访问延迟+局域网延迟=2秒+几微秒+几微秒

❖问题：

- 成本太高

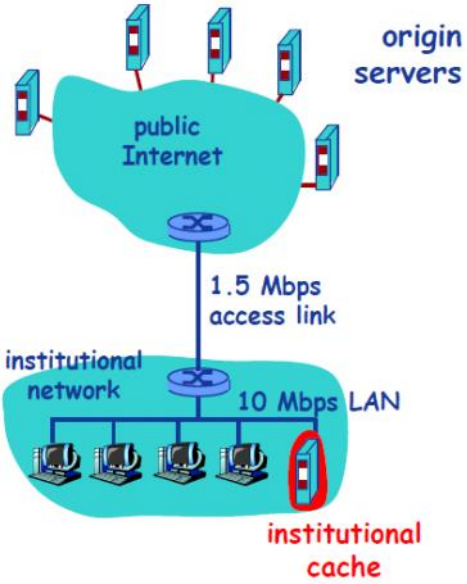


❖解决方案2：

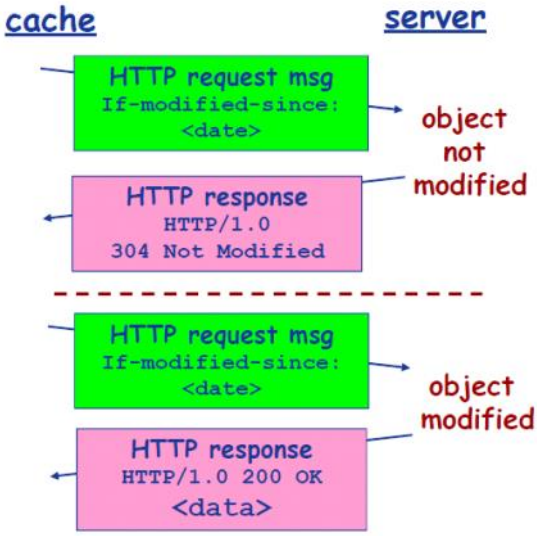
- 安装Web缓存
- 假定缓存命中率是0.4

❖网络性能分析：

- 40%的请求立刻得到满足
- 60%的请求通过原始服务器满足
- 接入互联网的链路的利用率下降到60%，从而其延迟可以忽略不计，例如10微秒
- 总的平均延迟=互联网上的延迟+访问延迟+局域网延迟=0.6×2.01秒+0.4×n微秒<1.4秒



条件性GET方法



- ❖目标：
  - 如果缓存有最新的版本，则不需要发送请求对象
- ❖缓存：
  - 在HTTP请求消息中声明所持有版本的日期
  - If-modified-since: <date>
- ❖服务器：
  - 如果缓存的版本是最新的，则响应消息中不包含对象
  - HTTP/1.0 304 Not Modified

- SMTP, POP, IMAP
  - ❖Email应用的构成组件
    - 邮件客户端(user agent)
    - 邮件服务器
    - SMTP协议(Simple Mail Transfer Protocol)
  - ❖ 邮件客户端
    - 读、写Email消息
    - 与服务器交互，收、发Email消息
    - Outlook, Foxmail, Thunderbird
    - Web客户端
  - ❖邮件服务器(Mail Server)
    - 邮箱：存储发给该用户的Email
    - 消息队列(message queue)：存储等待发送的Email
  - ❖SMTP协议
    - 邮件服务器之间传递消息所使用的协议
    - 客户端：发送消息的服务器
    - 服务器：接收消息的服务器
  - ❖使用TCP进行email消息的可靠传输
  - ❖端口25
  - ❖传输过程的三个阶段
    - 握手
    - 消息的传输
    - 关闭
  - ❖命令/响应交互模式

- 命令(command): ASCII文本
- 响应(response): 状态代码和语句

❖Email消息只能包含7位ASCII码

SMTP协议

❖使用持久性连接

❖要求消息必须由7位ASCII码构成

❖SMTP服务器利用CRLF.CRLF确定消息的结束。

与HTTP对比:

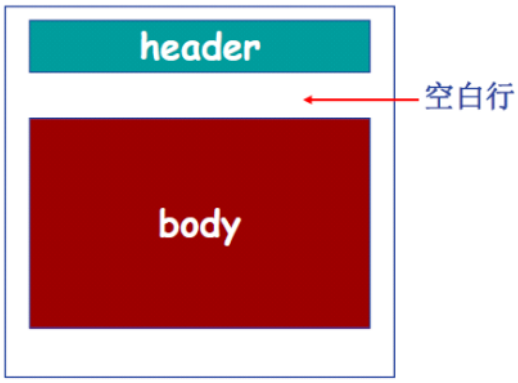
- ❖ HTTP: 拉式(pull)
- ❖ SMTP: 退式(push)
- ❖ 都使用命令/响应交互模式
- ❖ 命令和状态代码都是ASCII码
- ❖ HTTP: 每个对象封装在独立的响应消息中
- ❖ SMTP: 多个对象在由多个部分构成的消息中发送

Email消息格式与POP3协议

❖SMTP： email消息的传输/交换协议

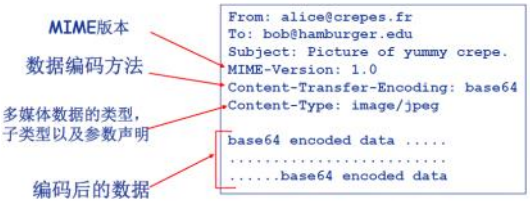
❖RFC 822： 文本消息格式标准

- 头部行(header)
  - To
  - From
  - Subject
- 消息体(body)
  - 消息本身
  - 只能是ASCII字符



❖MIME：多媒体邮件扩展RFC 2045, 2056

- 通过在邮件头部增加额外的行以声明MIME的内容类型

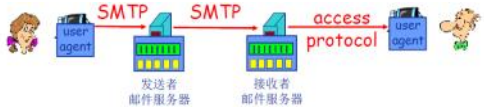


❖邮件访问协议：从服务器获取邮件

- POP: Post Office Protocol [RFC 1939]
  - 认证/授权(客户端←→服务器)和下载
- IMAP: Internet Mail Access Protocol [RFC 1730]
  - 更多功能
  - 更加复杂

• 能够操纵服务器上存储的消息

▪ HTTP：163, QQ Mail等。



POP协议

❖ 下载并删除”模式

▪ 用户如果换了客户端软件，无法重读该邮件

❖ 下载并保持”模式：不同客户端都可以保留消息的拷贝

❖ POP3是无状态的

IMAP协议

❖所有消息统一保存在一个地方：服务器

❖允许用户利用文件夹组织消息

❖IMAP支持跨会话(Session)的用户状态:

▪ 文件夹的名字

▪ 文件夹与消息ID之间的映射等