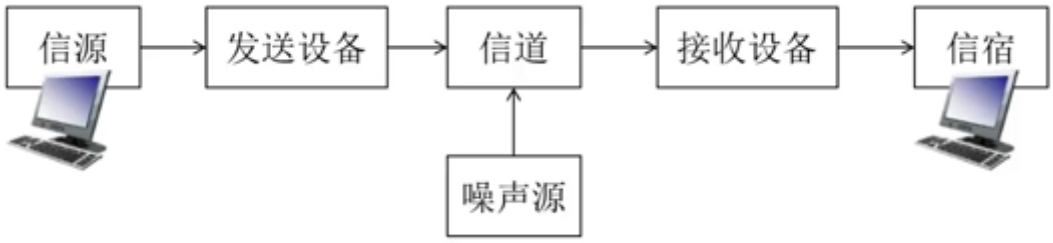
整理自哈工大计算机网络MOOC课件

1. 计算机网络概述

1.1 计算机网络基本概念

计算机网络 = 通信技术 + 计算机技术

- 计算机网络是**通信技术**与**计算机技术**结合的产物。
- 通信系统模型



- 计算机网络就是一种通信网络

什么是计算机网络?

- 计算机网络是**互联的**、**自治的**计算机集合。
  - 自治-无主从关系
  - 互联-互联互通
- 通过**交换网络**互联主机。
  - 交换结点-路由器或交换机

什么是Internet? -组成细节角度

- 全球最大的互联网络
  - ISP网络互联的网络之网络
- 数以百万计的互联的**计算设备**集合
  - 主机(host) = 端系统(end systems)
  - 运行各种网络应用
- 通信链路
  - 光纤、铜缆、无线电、卫星.....
- 分组交换：转发分组(数据包)
  - 路由器(routers)和交换机(switches)

什么是Internet? -服务角度

- 为网络应用提供通信服务的通信基础设施
  - Web，VoIP，email，网络游戏.....
- 为网络应用提供应用编程接口(API)
  - 支持应用程序"连接"Internet，发送/接收数据
  - 提供类似于邮政系统的数据传输服务

> 仅有硬件连接，Internet不能保持顺畅运行，无法保证应用数据有序交付，还需**网络协议**。

协议是计算机网络有序运行的重要保证

- **硬件**(主机、路由器、通信链路等)是计算机网络的**基础**。
- 计算机网络的数据交换必须遵守事先约定好的**规则**。

什么是网络协议?

- 网络协议简称**协议**，是为进行网络中的数据交换而建立的规则、标准或规定。
- 协议规定了通信实体之间所交换的**信息**、**意义**、**顺序**以及针对收到信息或发生的事件所采取的的“动作”。
    - 协议的三要素
        - 语法
            - 数据与控制信息的结构或格式
            - 信号电平
        - 语义
            - 需要发出何种控制信息
            - 完成何种动作以及做出何种响应
            - 差错控制
        - 时序
            - 时间顺序
            - 速度匹配

协议有什么作用?

- 协议规范了网络中所有信息发送和接收的过程
- 学习网络的重要内容之一
- 网络创新的表现形式之一

1.2 计算机网络的结构

- 网络边缘
    - 主机
    - 网络应用
- 接入网络，物理介质
    - 有线或无线通信链路
- 网络核心
    - 互联的路由器(或分组转发设备)
    - 网络之网络

网络边缘

- 主机(端系统)
    - 位于网络边缘
    - 运行网络应用程序
- 通讯方式
    - 客户/服务器(C/S)应用模型
        - 客户发送请求，接收服务器响应
    - 对等(P2P)应用模型
        - 无(或不仅依赖)专用服务器

        - 通信在对等实体之间进行

接入网络

如何将网络边缘接入核心网(边缘路由器)?

接入网络

- 住宅(家庭)接入网络
- 机构接入网络(学校、企业等)
- 移动接入网络

用户关心的是

- 带宽(bps)

网络核心

- 互联网的路由器网络
- 网络核心的关键功能：**路由+转发**
- 网络核心解决的基本问题
    - 利用数据交换实现数据从源主机通过网络核心送达目的主机

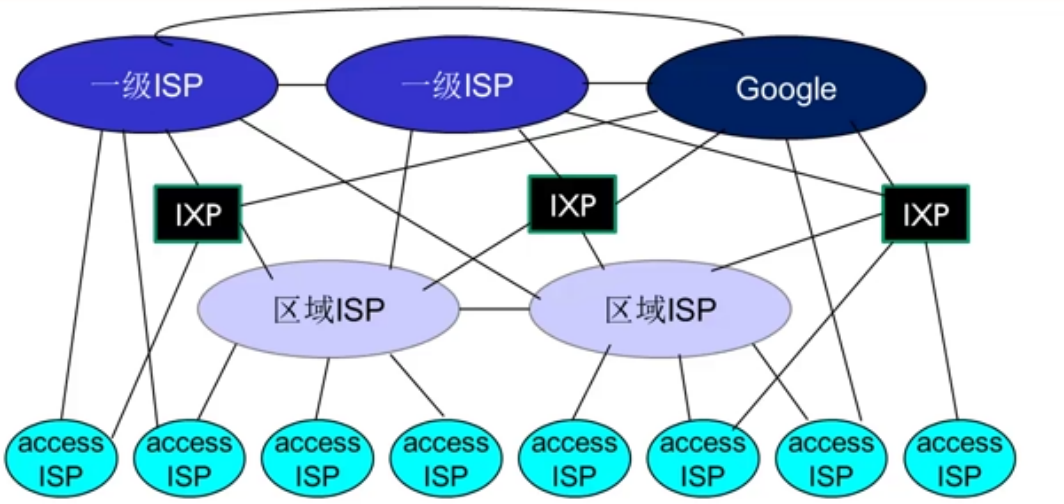
Internet结构：网络之网络

- 端系统通过接入ISP连接到Internet
    - 家庭、公司和大学ISPs
- 接入ISP必须进一步互连
    - 这样任意两个主机才能互相发送分组
- 构成复杂的网络互连的网络
    - 经济和国家政策是网络演进的主要驱动力

数以百万计的接入ISP时如何连接在一起的?

答：可选方案：将每个接入ISP连接到一个国家或全球ISP。

然而，从商业角度来看，必定有竞争者，利用IXP进行互连，可能会出现区域网络连接接入ISP和运营商ISP，内容提供商也可能运行其自己的网络，并就近为端用户提供服务、内容。



- 在网络中心: 少数互连的大型网络
  - “一级”(tier-1)商业ISPs (如：网通、电信、Sprint、 AT&T)，提供国家或国际范围的覆盖
  - 内容提供商网络（content provider network， 如：Google)：私有网络，连接其数据中心与Internet，通常绕过一级ISP和区域ISPs

1.3 网络核心

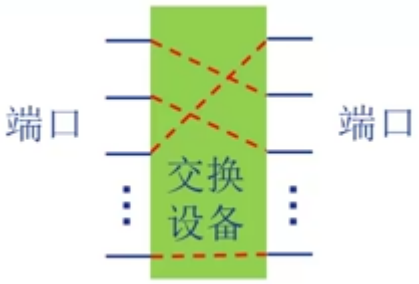
利用**数据交换**实现数据从源主机通过网络核心送达目的主机

为什么需要数据交换?

答：N^2链路问题、连通性、网络规模

什么是交换?

- 动态转接



- 动态分配传输资源

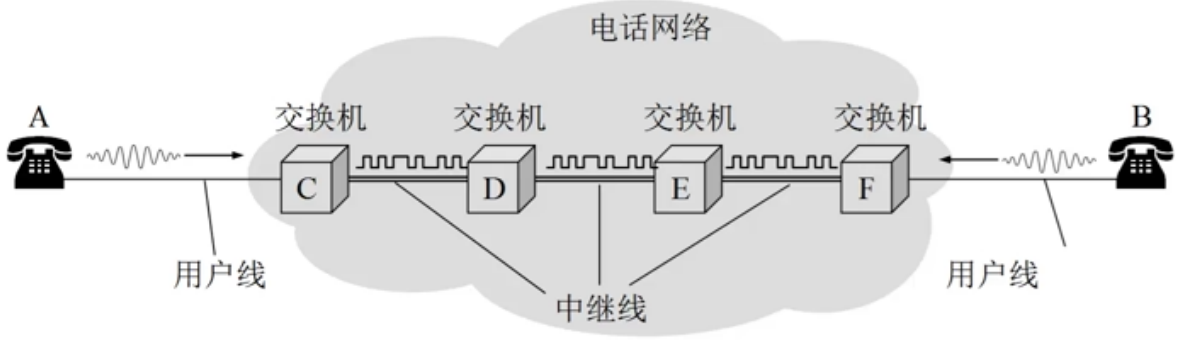
数据交换的类型?

- 电路交换
- 报文交换
- 分组交换

电路交换

电路交换的特点

最典型电路交换网络：电话网络



电路交换的三个阶段

- 建立连接(呼叫/电路建立)
- 通信
- 释放连接(拆除电路)

独占资源

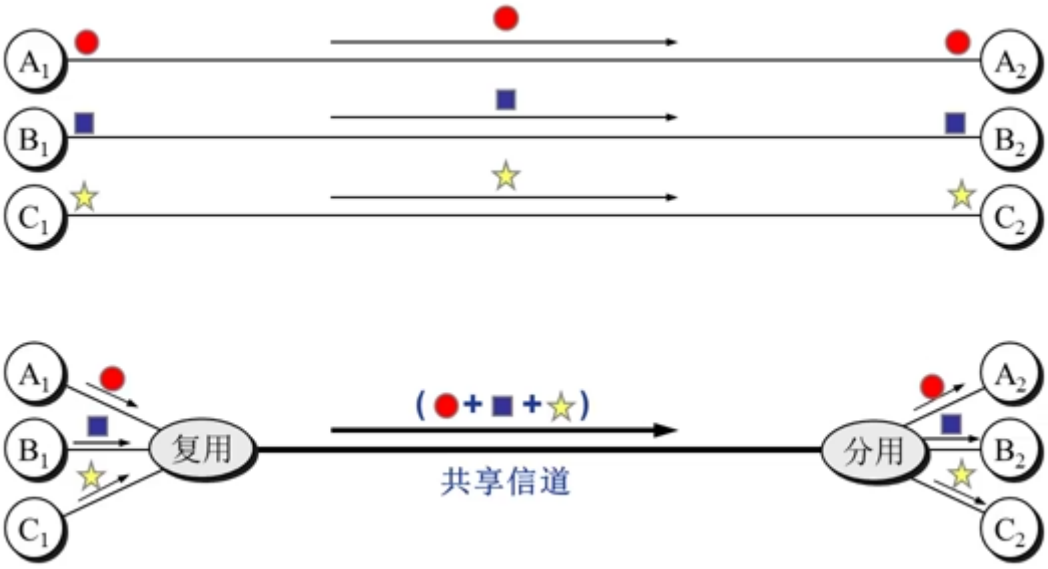
通话过程中电路占用的电路资源不能被第三方使用(但不能说电路交换网络中，每条电路独占其经过的物理链路，因为一条物理链路要多路复用)。

电路交换网络的链路共享?

电路交换网络如何共享中继线？—多路复用。

多路复用

多路复用，简称**复用**，是通信技术中的基本概念。

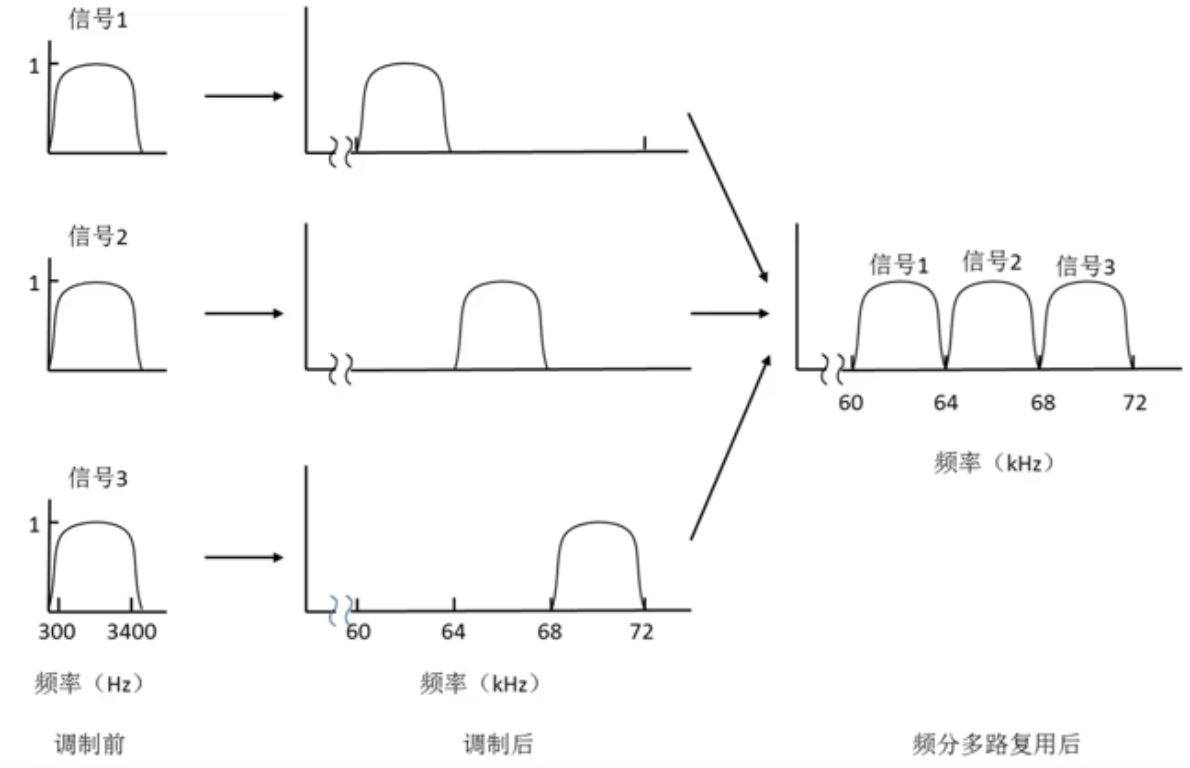


- 链路/网络资源(如带宽)划分为“资源片”
- 将资源片分配给各路“呼叫”
- 每路呼叫**独占**分配到的资源片进行通信
- 资源片可能“闲置”(无共享)

典型多路复用方法

- 频分多路复用(FDM)
- 时分多路复用(TDM)
- 波分多路复用(WDM)
- 码分多路复用(CDM)

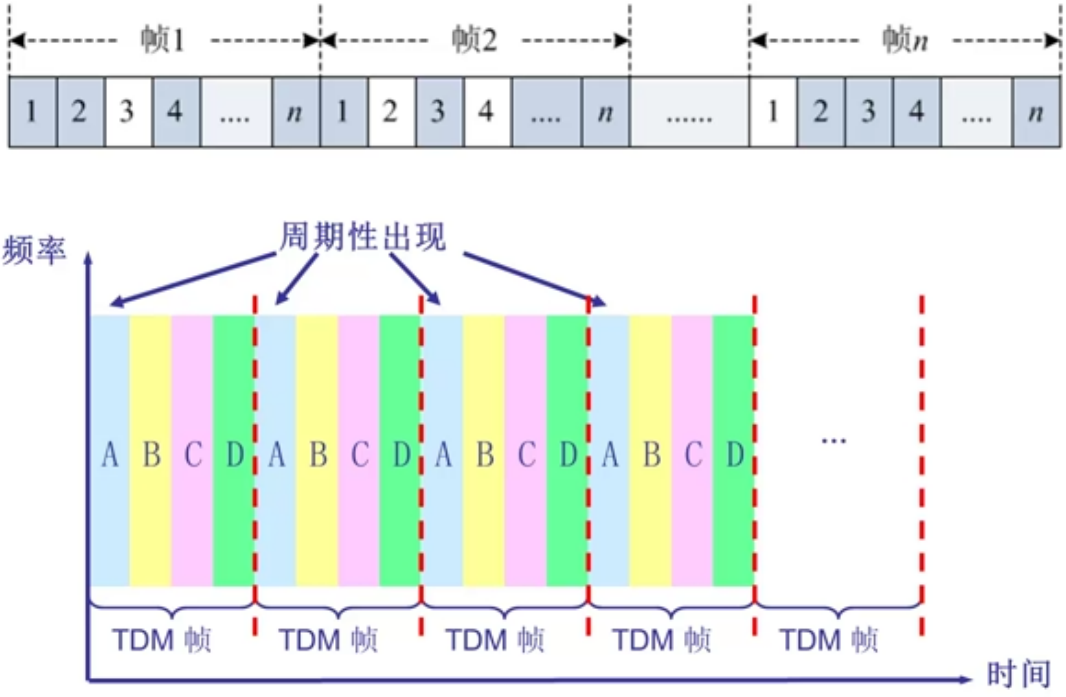
频分多路复用FDM



- **频分多路复用**的各用户占用不同的带宽资源(这里的带宽是频率带宽Hz,不是数据发送速率)
- 用户在分配到一定频带后，在通信过程中自始至终都占用这个频带

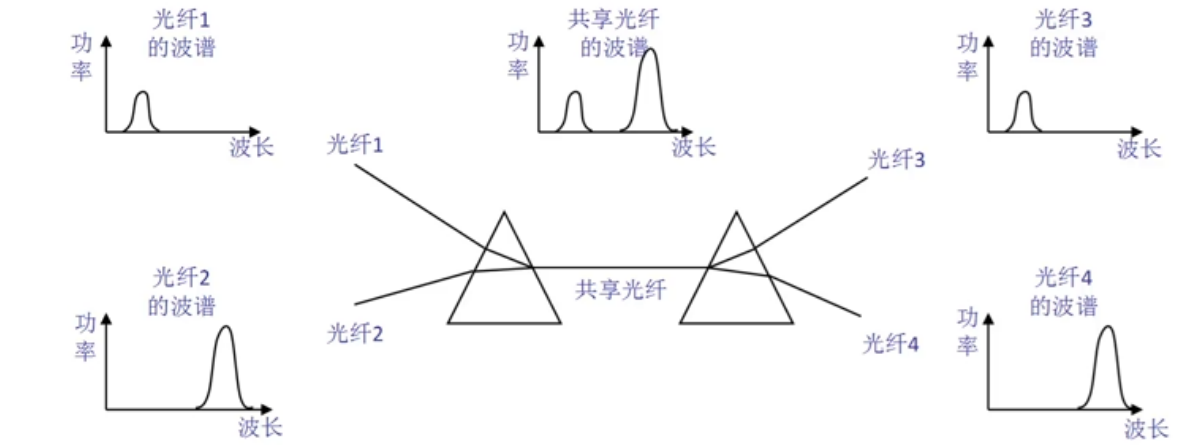
时分多路复用TDM

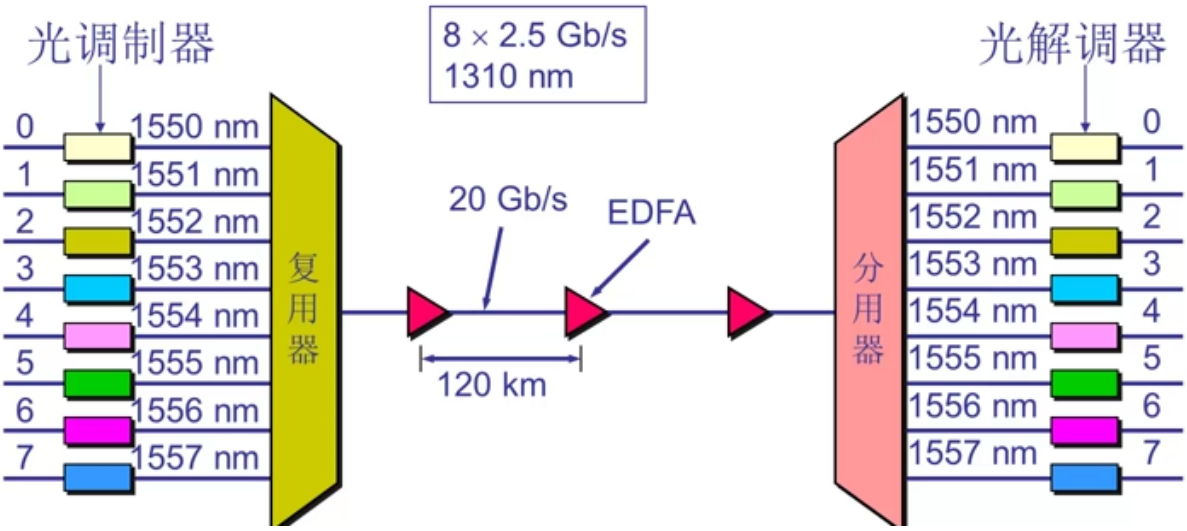
- **时分复用**是将时间划分为一段段等长的**时分复用帧**(TDM帧)，每个用户在每个TDM帧中占用固定序号的时隙。
- 每个用户所占用的时隙**周期性出现**(其周期就是TDM帧的长度)



波分多路复用WDM

- 波分复用就是光的频分复用





码分多路复用CDM

- 广泛应用于无线链路共享(如蜂窝网，卫星通信等)
- 每个用户分配一个唯一的m bit**码片序列**，其中“0”用“-1”表示，“1”用"+1"表示，
  例如：
  S站的码片序列：(-1,-1,-1,+1,+1,-1,+1,+1)
- 各用户使用**相同频率**载波，利用各自码片序列编码数据
- 编码信号=(原始数据)×(码片序列)
  - 如发送比特1(+1)，则发送自己的m bit码片序列
  - 如发送比特0(-1)，则发送该码片序列的m bit码片序列的反码

- 各用户码片序列相互正交

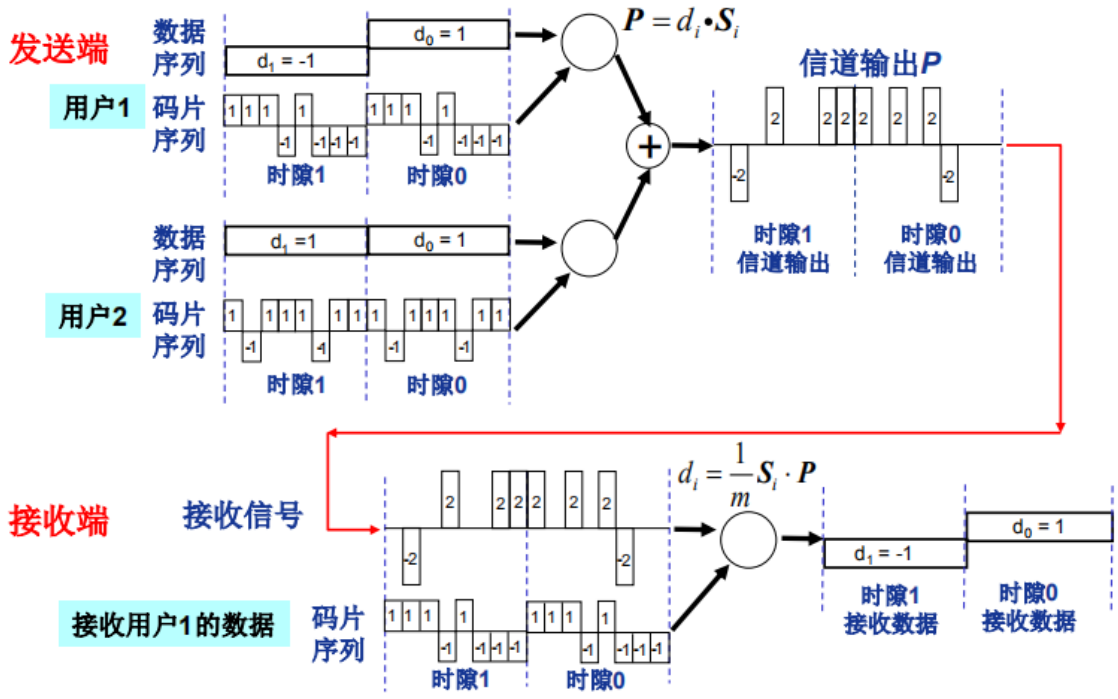
$$\frac{1}{m}\boldsymbol{S}_i \cdot \boldsymbol{S}_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \qquad \frac{1}{m}\boldsymbol{S}_i \cdot \bar{\boldsymbol{S}}_j = \begin{cases} -1, & i = j \\ 0, & i \neq j \end{cases}$$

令$\{d_i\}$为原始数据序列，各用户的叠加向量为

$$\boldsymbol{P} = \sum_{i=1}^{N} d_i \bullet \boldsymbol{S}_i = \sum_{i=1}^{N} \overset{(-)}{\boldsymbol{S}}_i$$

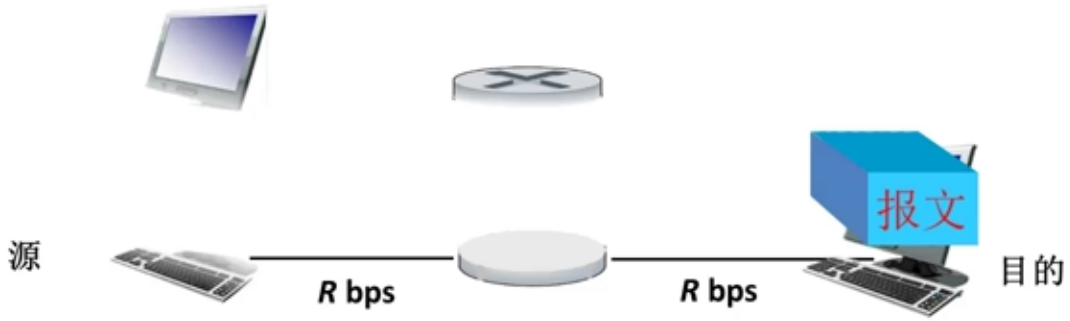
解码: 码片序列与编码信号的内积

$$\frac{1}{m}\boldsymbol{S}_i \cdot \boldsymbol{P} = \begin{cases} 1 & \boldsymbol{S}_i \in \boldsymbol{P} \\ -1 & \bar{\boldsymbol{S}}_i \in \boldsymbol{P} \\ 0 & \boldsymbol{S}_i, \bar{\boldsymbol{S}}_i \notin \boldsymbol{P} \end{cases}$$



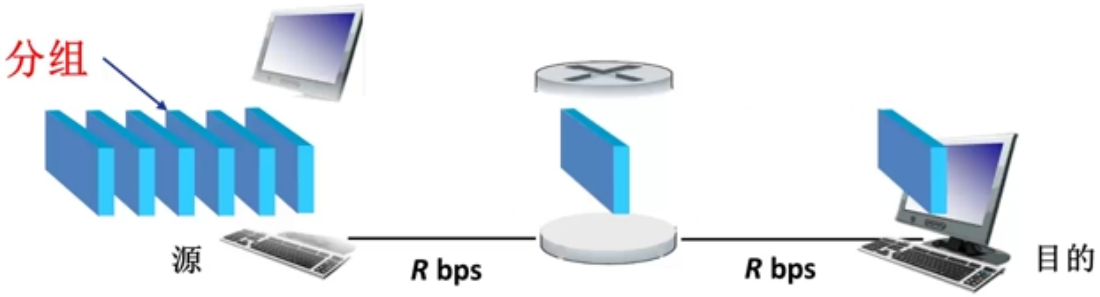
报文交换

- 报文：源(应用)发送信息整体

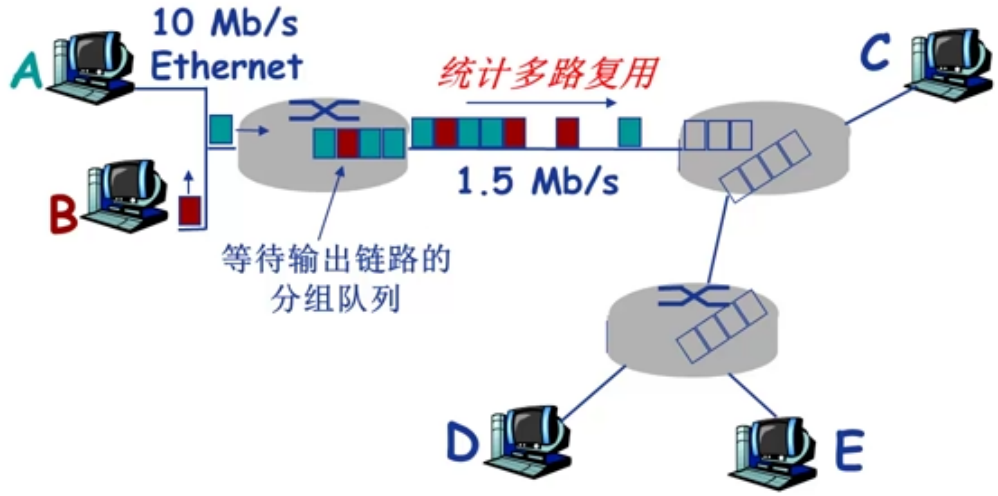


分组交换

- 分组：报文分拆出来的一系列相对较小的数据包(头部+数据)
- 分组交换需要报文的**拆分**与**重组**
- 产生**额外开销**



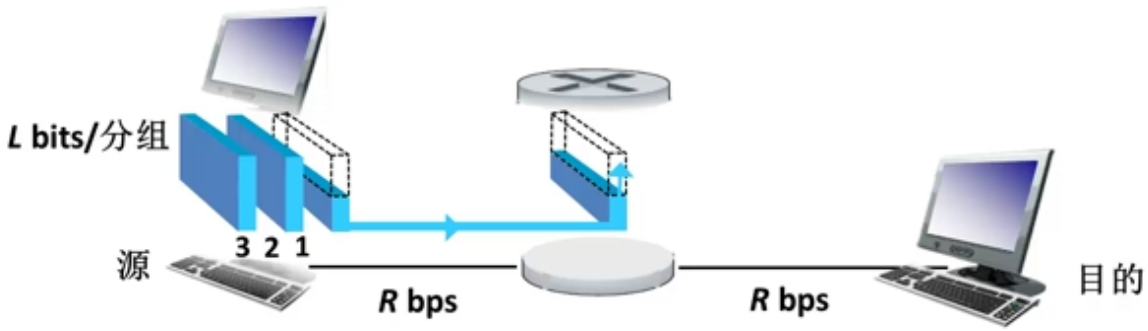
- 分组交换：统计多路复用



A & B分组序列不确定，按需共享链路

➔ ***statistical multiplexing***.

存储-转发



- **报文交换**与**分组交换**均采用**存储-转发**交换方式
- 区别
  - 报文交换以**完整报文**进行"存储-转发"
  - 分组交换以较小的**分组**进行"存储-转发"

传输延迟

- 发送主机:
  - 接收应用报文(消息)
  - 拆分为较小长度为L bits的分组(packets)
  - 在传输速率为R的链路上传输分组

$$\text{分组传输延迟（时延）（delay）} = \frac{L\ (\text{bits})}{R\ (\text{bits/sec})}$$

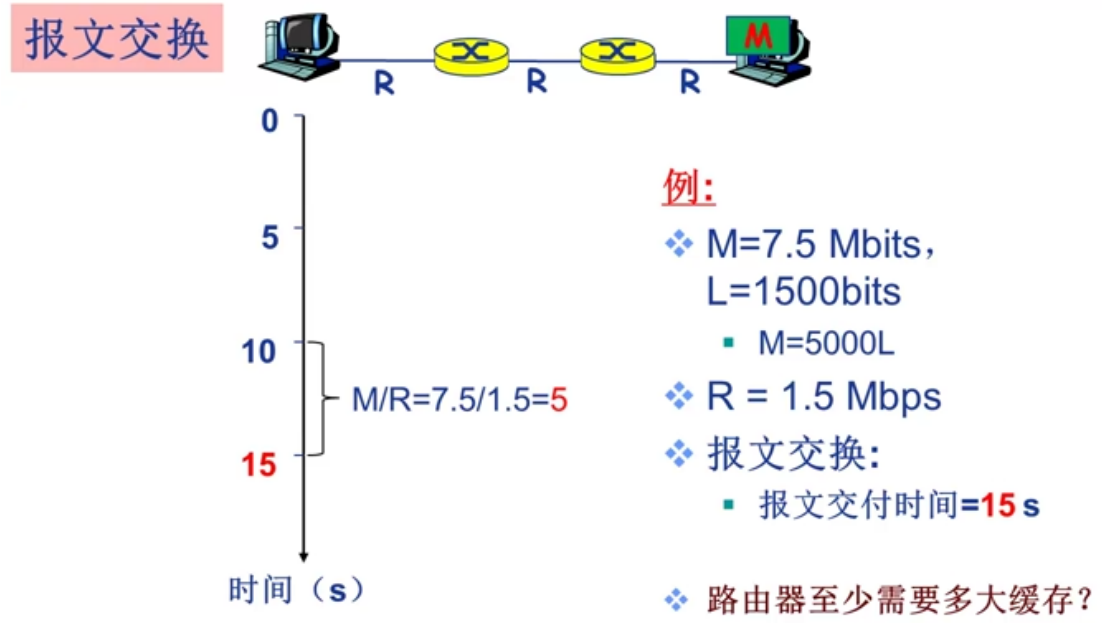
报文交换 vs 分组交换?（只有结果，不懂看视频）

- 报文交换:
  - 报文长度为M bits
  - 链路带宽为R bps
  - 每次传输报文需要M/R秒
- 分组交换:
  - 报文被拆分为多个分组
  - 分组长度为L bits
  - 每个分组传输时延为L/R秒

例:

- M=7.5 Mbits，L=1500bits
  - M=5000L
- R = 1.5 Mbps

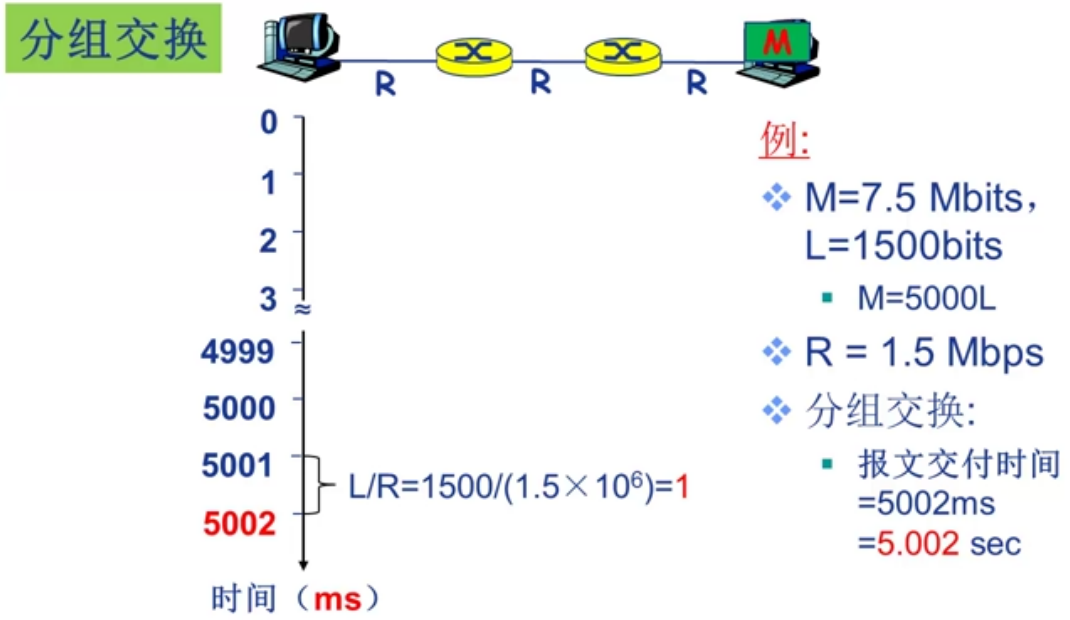
- 报文交换



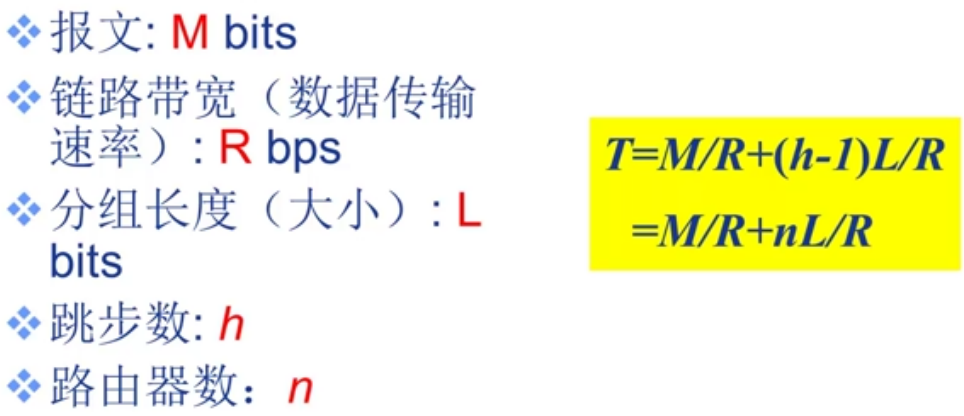
例:

- M=7.5 Mbits，L=1500bits
  - M=5000L
- R = 1.5 Mbps
- 报文交换:
  - 报文交付时间=15 s
- 路由器至少需要多大缓存?

- 分组交换



  - 分组交换的报文交付时间(忽略传播时延跟处理时延用)



    - 一般计算：第一个分组到达目的需要的总时间+其余分组从起点发出的时间

分组交换 vs 电路交换?

分组交换绝对优于电路交换吗?

- 分组交换适用于突发数据传输网络
  - 资源充分共享
  - 简单、无需呼叫建立
  - 但可能产生**拥塞**，在成分组延迟和丢失
    - 需要协议处理可靠数据传输和拥塞控制

1.4 计算机网络性能

速率

- 速率即**数据率**或称**数据传输速率**或**比特率**
  - 单位时间(秒)传输信息(比特)量
  - 计算机网络中最重要的一个性能指标
  - 单位：b/s(bps)、kb/s、Mb/s、Gb/s
  - k=10^3、M=10^6、G=10^9
- 速率往往是指**额定速率**或**标称速率**

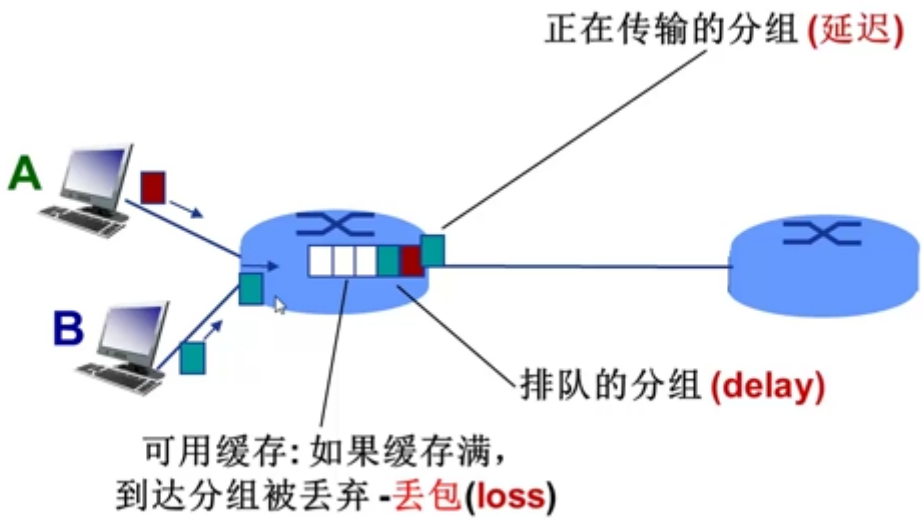
带宽

- “带宽”原本指信号具有的频带宽度，即最高频率与最低频率之差，单位是赫兹(Hz)
- 网络的“带宽”通常是数字信道所能传送的**“最高数据率”**，单位b/s (bps)
- 常用的带宽单位：
  - kb/s(10^3 b/s)
  - Mb/s(10^6 b/s)
  - Gb/s(10^9 b/s)
  - Tb/s(10^{12} b/s)

延迟/时延

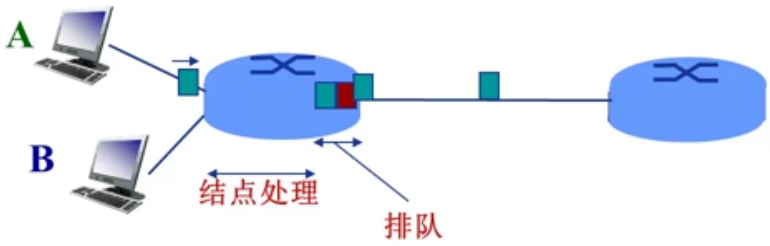
分组交换为什么会产生丢包和时延?

答：分组在路由器缓存中排队



- 四种分组延迟

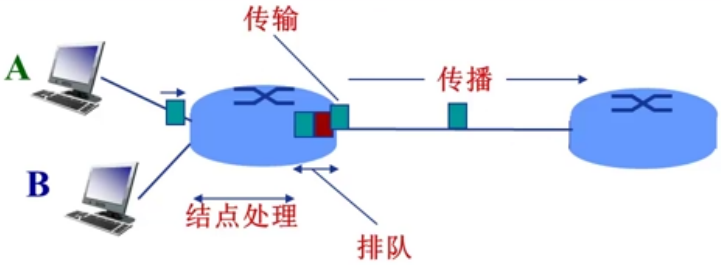
- 结点处理延迟和排队延迟



d_{proc}: 结点处理延迟
（nodal processing delay）
- 差错检测
- 确定输出链路
- 通常< msec

d_{queue}: 排队延迟
（queueing delay）
- 等待输出链路可用
- 取决于路由器拥塞程度

- 传输延迟和传播延迟



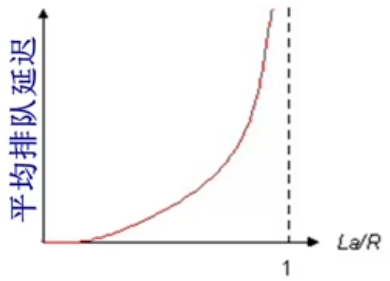
d_{trans}: 传输延迟
（transmission delay）
- L: 分组长度(bits)
- R: 链路带宽 (bps)
- $d_{trans} = L/R$

d_{prop}: 传播延迟（propagation delay）
- d: 物理链路长度
- s: 信号传播速度 (~2×10^8 m/sec)
- $d_{prop} = d/s$

- $d_{nodal}=d_{proc}+d_{queue}+d_{trans}+d_{prop}$
- 排队延迟

- R: 链路带宽(bps)
- L: 分组长度 (bits)
- a: 平均分组到达速率

流量强度（traffic intensity）
= La/R



- $La/R \sim 0$: 平均排队延迟很小
- $La/R \to 1$: 平均排队延迟很大
- $La/R > 1$: 超出服务能力，平均排队延迟无限大！

La/R ~ 0

La/R -> 1

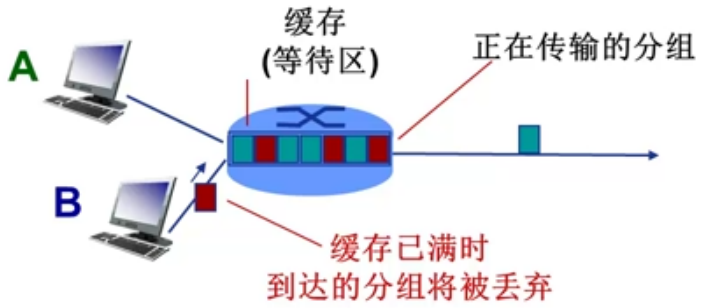
时延带宽积

- **时延带宽积=传播时延×带宽=d_{prop}×R（bits）**

❖链路的时延带宽积又称为以比特为单位的链路长度

分组丢失(丢包)

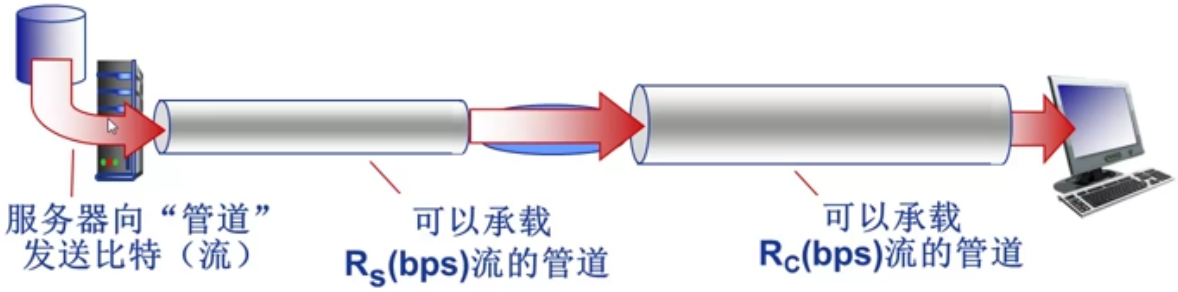
- 队列缓存容量有限
- 分组到达已满队列将被丢弃 (即丢包)
- 丢弃分组可能由前序结点或源重发（也可能不重发）



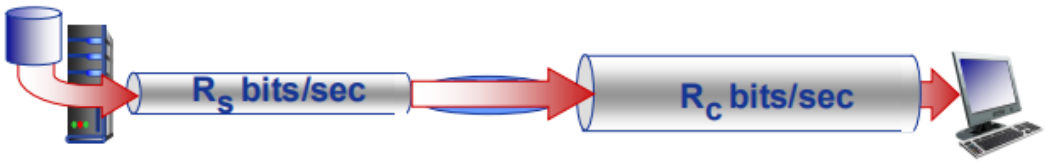
$$丢包率=\frac{丢包数}{已发分组总数}$$

吞吐量/率

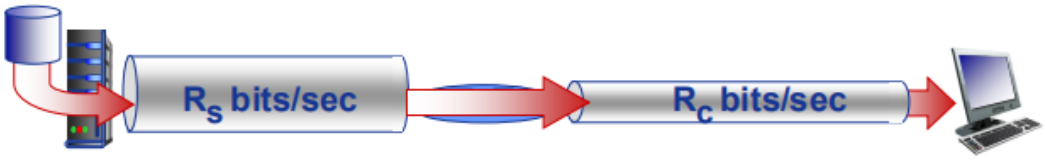
- 吞吐量表示在发送端与接收端之间传送数据速率(b/s)
  - 即时吞吐量：给定时刻的速率
  - 平均吞吐量：一段时间的平均速率



- 若$R_s < R_c$，则端到端的吞吐量是多少？



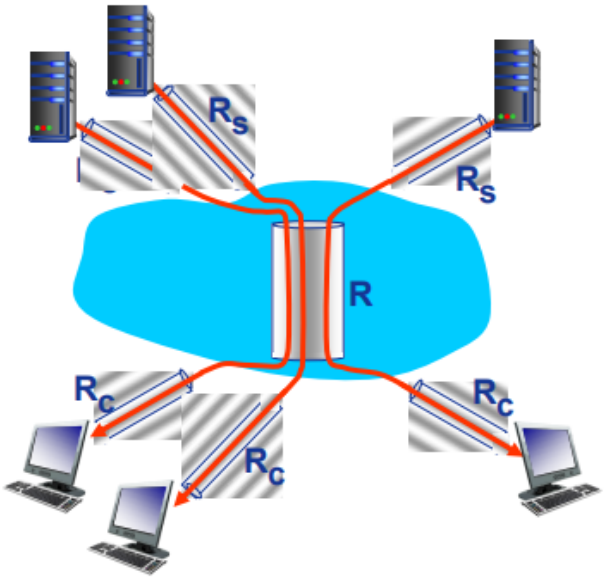
- 若$R_s > R_c$，则端到端的吞吐量是多少？



瓶颈链路（bottleneck link）

端到端路径上，限制端到端吞吐量的链路。

- 每条“连接”的端到端吞吐量: $\min(R_c,R_s,R/10)$
- 实际网络: R_c 或 R_s 通常是瓶颈



10 条“连接”共享
主干网瓶颈链路R bits/sec

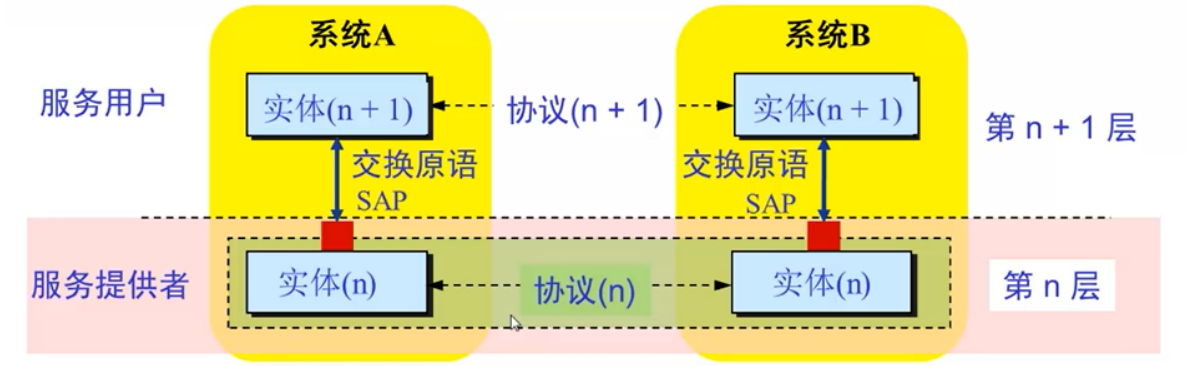
1.5 计算机网络的体系结构

为方便讨论，引入体系结构，从功能上来描述的计算机网络。计算机网络体系结构简称网络体系结构，是一种抽象的分层结构，每层遵循某些网络协议完成本层功能

采用分层结构的原因?

- 结构清晰，有利于识别复杂系统的部件及其关系
- 模块化的分层易于系统的维护、更新
  - 任何一层服务实现的改变对系统其它层都是**透明**的
- 有利于标准化

分层网络体系结构基本概念

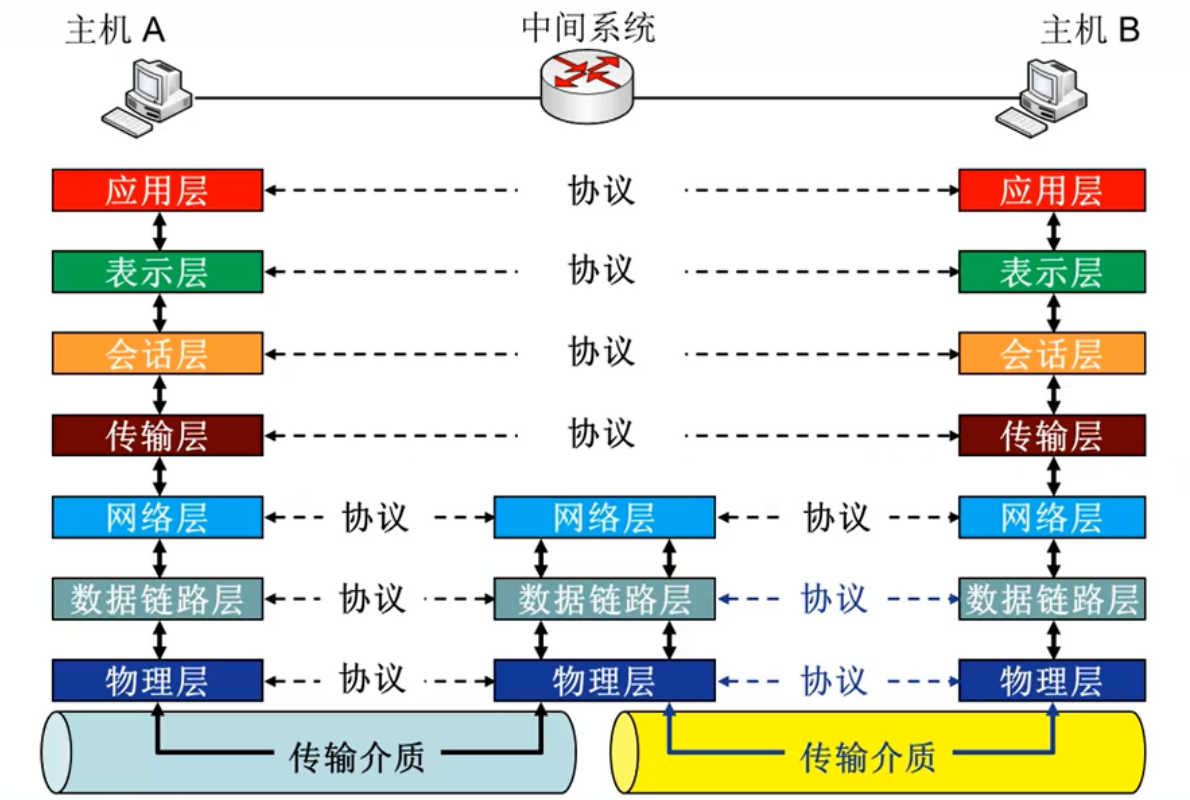


- **实体**表示任何可发送或接收信息的硬件或软件进程
- 协议是控制两个**对等实体**进行通信的规则的集合，协议是**水平的**
- 任一层实体需要使用**下层**服务，遵循**本层**协议，实现**本层功能**，向**上层**提供服务，服务是**垂直的**
- 下层协议的实现对上层的服务用户是**透明**的
- 同系统的相邻层实体间通过**接口**进行交互，通过服务访问点SAP，交换原语，指定请求的特定服务

1.5.1 OSI参考模型

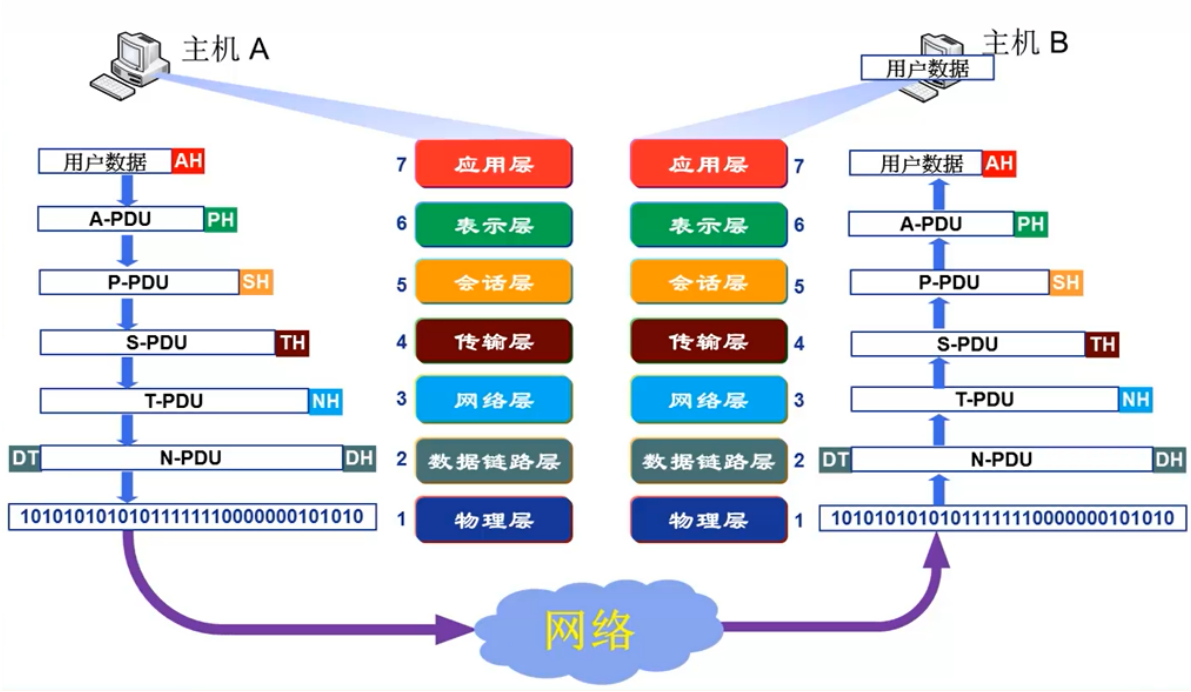
- OSI参考模型是由ISO在1984年提出的分层结构
- 目的是支持异构网络系统的互联互通，是异构网络互联系统的国际标准
- 是理解网络通信的最佳学习工具（理论成功，市场失败）
- 具有7层，每个层次完成不同的功能

OSI参考模型解释的通信过程



Tips:**传输层及之上为端到端通信**

OSI参考模型数据封装与通信过程



为什么要数据封装?

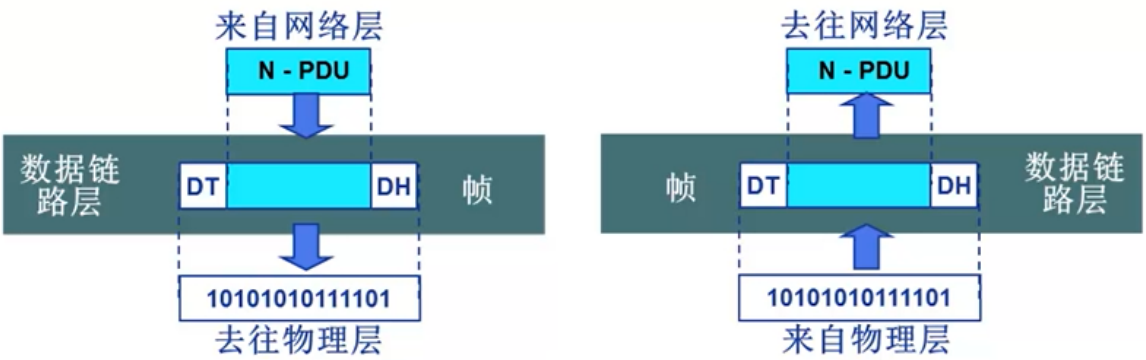
- 增加控制信息
  - 构造协议数据单元PDU

  控制信息主要包括:
  - 地址：标识发送端/接收端
  - 差错检测编码：用于差错检测或纠正
  - 协议控制：实现协议功能的附加信息，如：优先级、服务质量和安全控制等

OSI参考模型-物理层功能

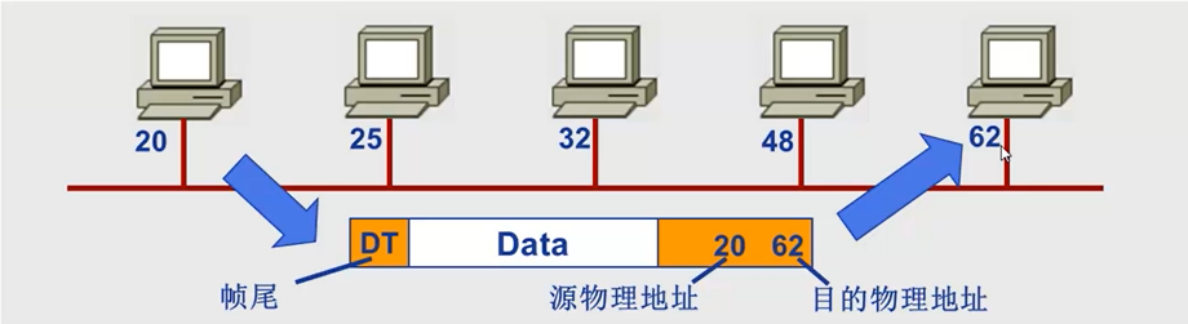
- 接口特性
  - 机械特性、电气特性、功能特性、规程特性
- 比特编码
- 数据率
- 比特同步
  - 时钟同步
- 传输模式
  - 单工通信
  - 半双工通信
  - 全双工通信

OSI参考模型-数据链路层功能



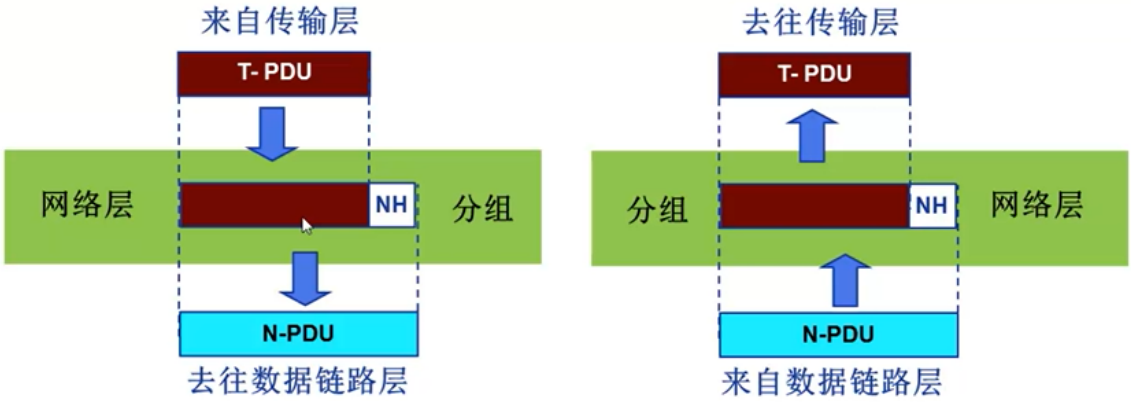
- 负责结点-结点的数据传输
- 组帧
- 物理寻址
  - 在帧头中增加发送端和/或接收端的物理地址标识数据帧的发送端和/或接收端
- 流量控制
  - 避免淹没接收端
- 差错控制
  - 检错并重传损坏或丢失帧，并避免重复帧
- 访问/接入控制
  - 在任一给定时刻决定哪个设备拥有链路（物理介质）控制使用权

为什么要有物理寻址的功能?

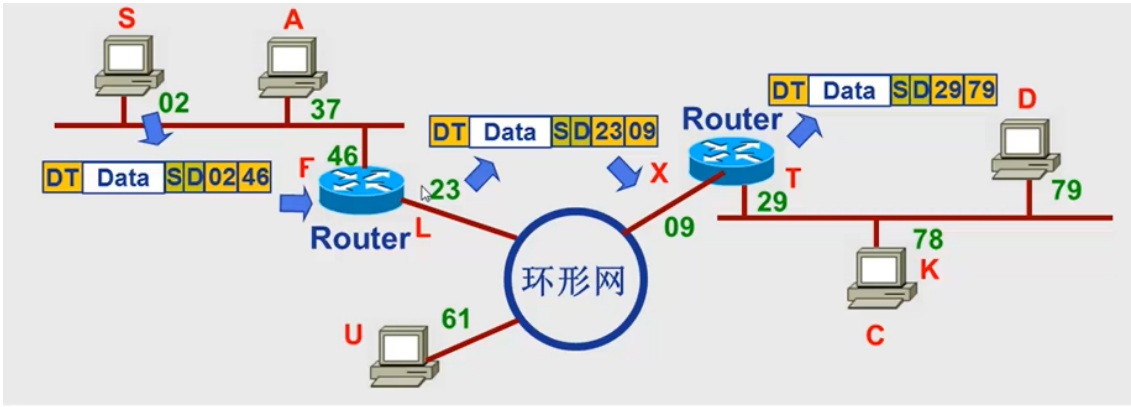


若不能物理寻址，数据发出后没有办法保证**被正确接收**！

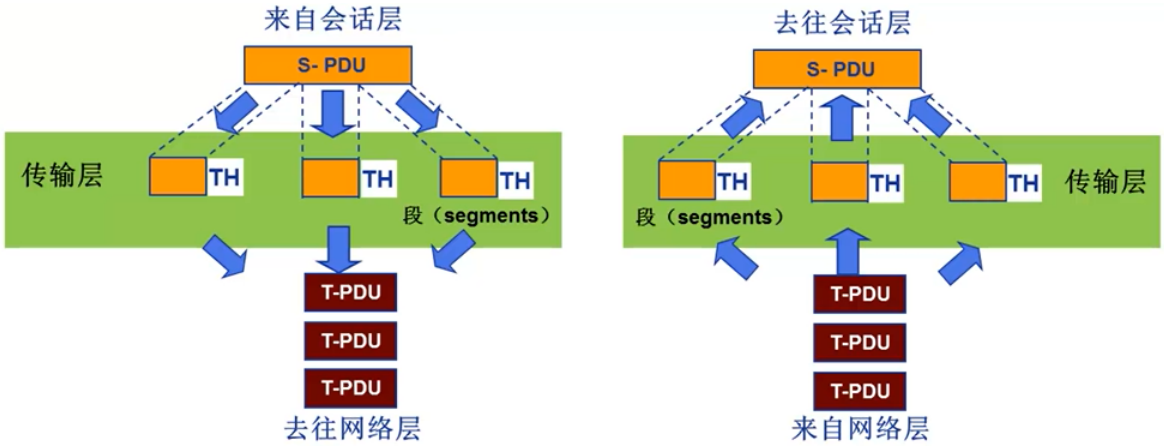
OSI参考模型-网络层功能



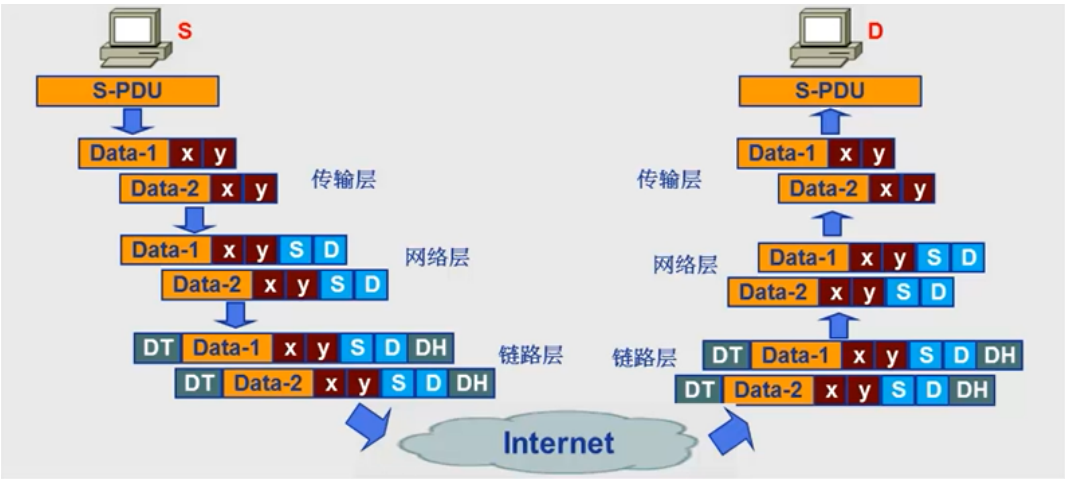
- 负责源主机到目的主机数据分组（packet）交付
  - 可能穿越多个网络
- 逻辑寻址
  - 全局唯一逻辑地址，确保数据分组被送达目的主机，如：IP地址
- 路由（routing）
  - 路由器（或网关）互连网络，并路由分组至最终目的主机
  - 路径选择
- 分组转发



OSI参考模型-传输层功能

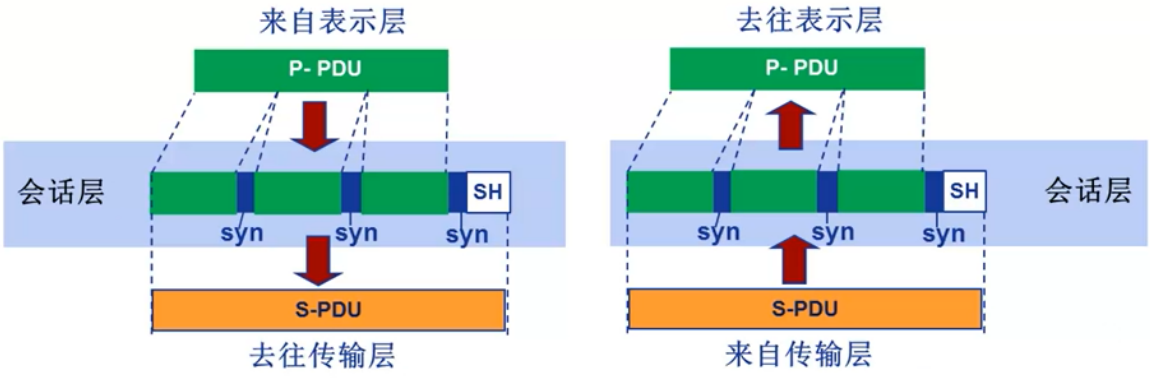


- 负责源-目的（端-端）（进程间）完整报文传输
- 分段与重组
- SAP寻址
    - 确保将完整报文提交给正确进程，如端口号



- 连接控制
- 流量控制
- 差错控制

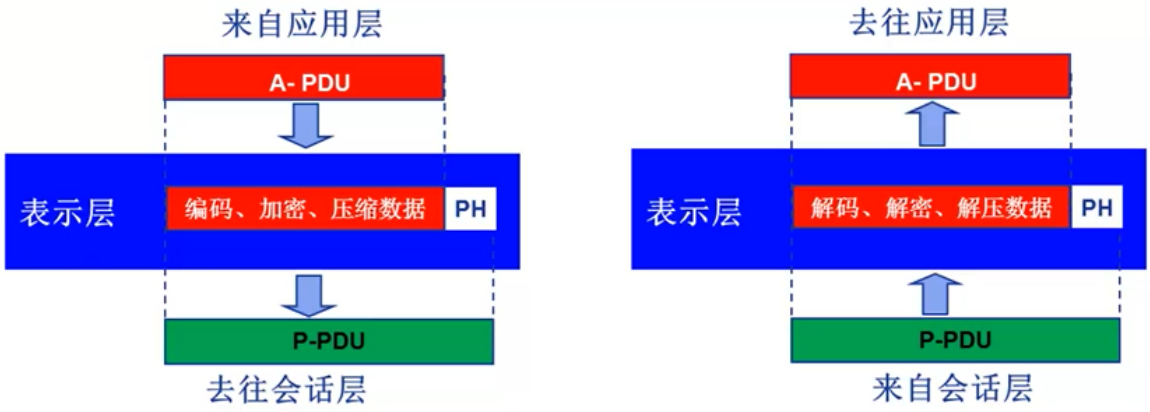
OSI参考模型-会话层功能



- 对话控制
    - 建立、维护
- 同步
    - 在数据流中插入“同步点”

功能少，最薄的一层
PS：**在实际的网络中，不存在会话层这一层**

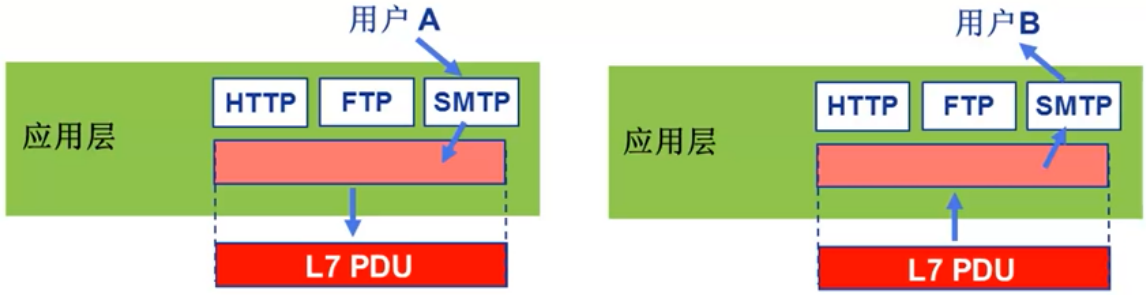
OSI参考模型-表示层功能



表示层处理两个系统间交换信息的语法与语义问题

- 数据表示转化
  - 转换为主机独立的编码
- 加密/解密
- 压缩/解压缩

OSI参考模型-应用层功能

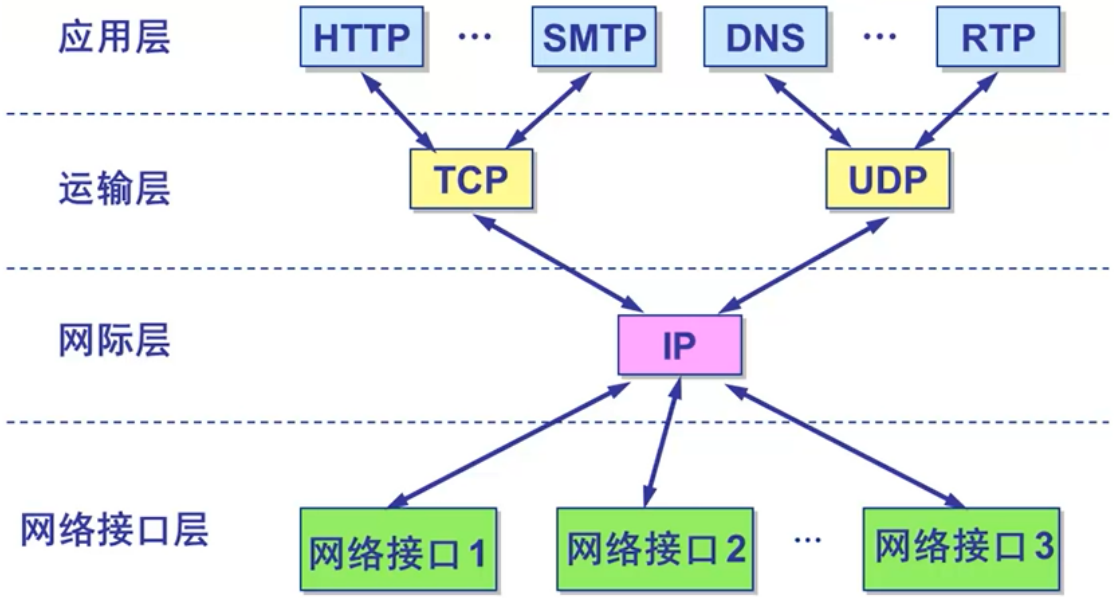


支持用户通过用户代理（如浏览器）或网络接口使用网络（服务）

典型应用层服务：

- 文件传输（FTP）
- 电子邮件（SMTP）
- Web（HTTP）等

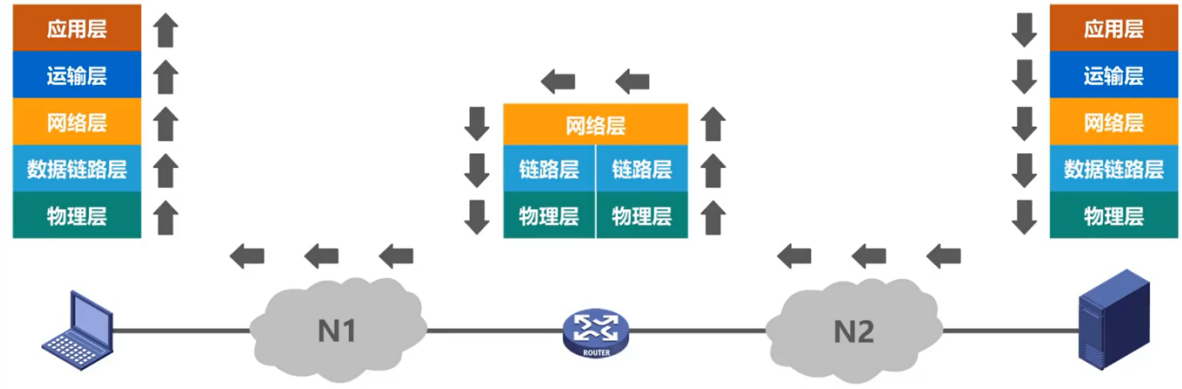
1.5.2 TCP/IP参考模型



1.5.3 五层参考模型

- 综合 OSI 和 TCP/IP 的优点
- *应用层*: 支持各种网络应用
  - FTP, SMTP, HTTP
- *传输层*: 进程-进程的数据传输
  - TCP, UDP
- *网络层*: 源主机到目的主机的数据分组路由与转发
  - IP协议、路由协议等
- *链路层*: 相邻网络元素（主机、交换机、路由器等）的数据传输
  - 以太网（Ethernet）、802.11 (WiFi)、PPP
- *物理层*:比特传输

五层参考模型的数据封装



1.6 计算机网络与Internet发展历史

一看就不考

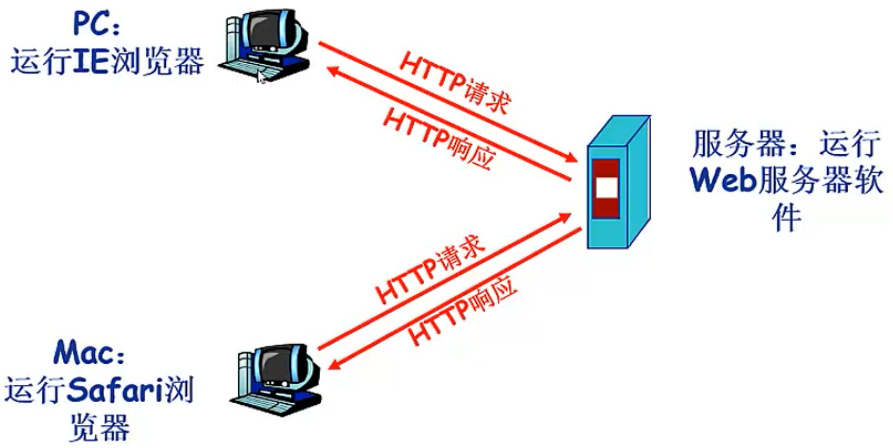
整理自哈工大计算机网络MOOC课件

2.网络应用

2.1 网络应用的体系结构

2.1.1 客户机/服务器（C/S）结构

❖例子：Web



服务器需要满足什么条件?

- 7*24小时提供服务
- 永久性访问地址/域名
- 利用大量服务器实现可拓展性

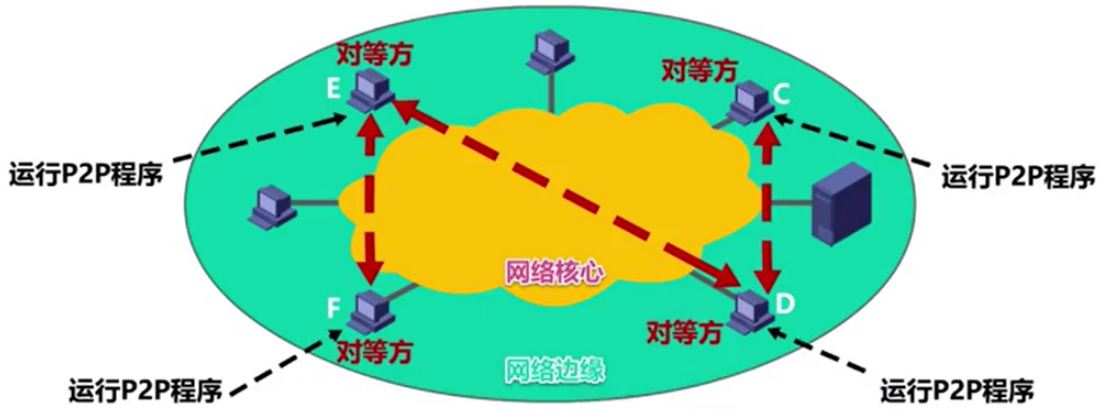
客户机需要满足什么条件?

- 与服务机通信，使用服务器提供的服务
- 间接性接入网络
- 可能使用动态IP
- 不会与其他客户机直接通信

优点：交互性强、具有安全的存取模式、响应速度快、利于处理大量数据

缺点：单点故障问题，运维难度大等

2.1.2 纯P2P结构



- **没有永远在线**的服务器
- 任意端系统/节点之间可以**直接通信**
- 节点间歇性接入网络
- 节点可能改变IP地址

优点：高度可伸缩

缺点：难于管理

2.1.3 混合结构

混用两者优点，规避两者缺点。

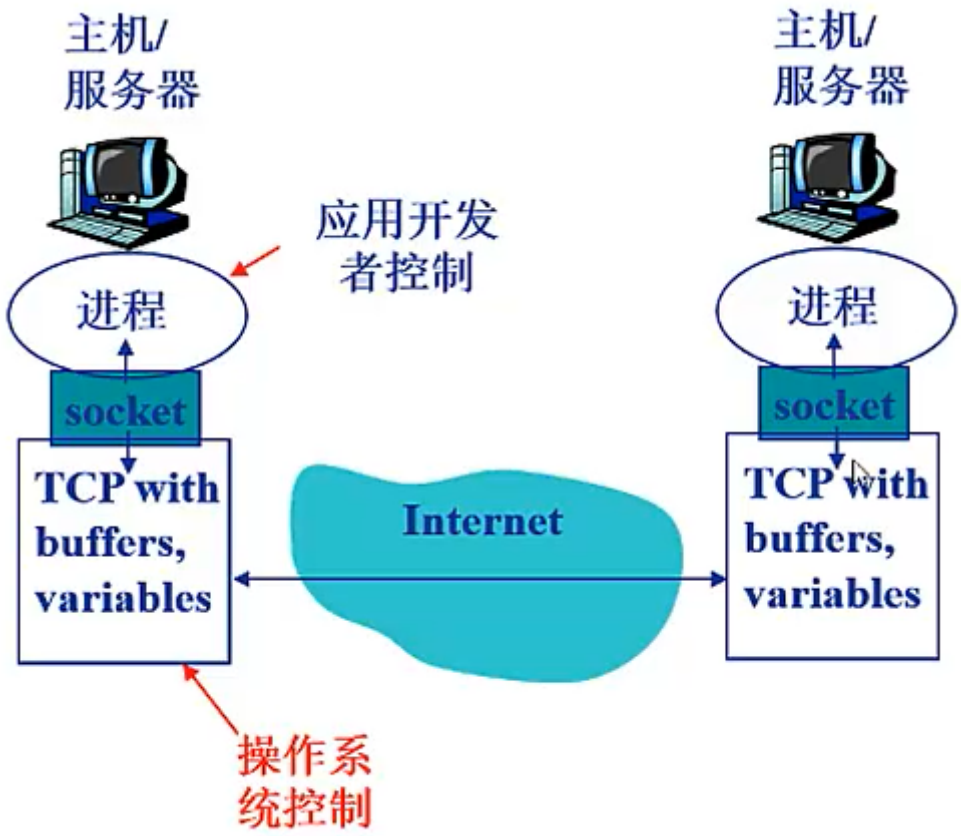
如：Napster

- 文件传输使用P2P
- 文件搜索采用C/S结构-集中式
    - 每个节点向自己的服务器登记自己的内容
    - 每个节点向中央服务器提交查询请求，查找感兴趣的内容

2.2 网络应用进程通信

- 进程
    - 主机上运行的程序
- 同一主机上运行的进程之间如何通信?
    - 进程间通信机制
    - 操作系统提供
- 不同主机上运行的进程间如何通信?
    - 信息交换
- 客户机&服务器进程
    - 发起通信的进程为客户机进程
    - 等待通信请求的进程为服务器进程

2.2.1 套接字：Socket



- 进程间利用套接字来发送/接收消息
- 类似于寄信
  - 发送方将消息发送到门外邮箱
  - 发送方依赖（门外的）传输基础设施将消息传到接收方所在主机，并送到接收方的门外
  - 接收方从门外获取消息
- 传输基础设施向进程提供API
  - 传输协议的选择
  - 参数的设置

2.2.2 如何寻址进程

- 不同主机上的进程间通信，每个主机进程必须有标识符
- 如何寻址主机？---**IP地址**
  - IP地址只能定位主机，但因为主机上可能有多个进程要通信，需要额外机制来识别进程---**端口号**
- 端口号/Port number
  - 为主机上每个需要通信的进程分配一个端口号
  - HTTP Server：80
  - Mail Server：25
- 进程的标识符
  - **IP地址+端口号**

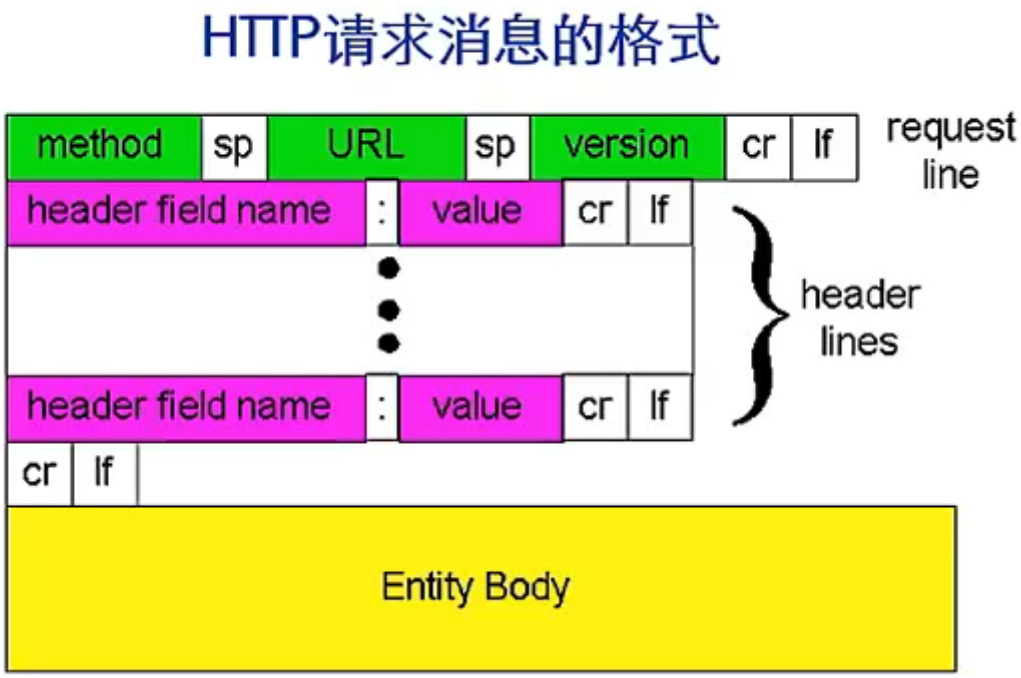
协议	本机IP地址: 端口号	外部IP地址: 端口号	状态
TCP	192.168.0.100:49225	202.108.23.105:5287	ESTABLISHED
TCP	192.168.0.100:49241	sinwns1011813:https	ESTABLISHED

2.2.3 应用层协议

- 网络应用需遵循应用层协议
- 公开协议
  - 有RFC定义
  - 互相允许操作
  - 如：HTTP、SMTP....
- 私有协议
  - 多数P2P文件共享应用

2.2.3.1 应用层协议的内容

例子：



- 消息的类型
  - 请求消息
  - 响应消息
- 消息的语法/格式
  - 消息中有哪些字段?
  - 每个字段如何描述
- 字段的语义
  - 字段中信息的含义
- 规则
  - 进程何时发送/响应消息?
  - 进程如何发送/响应消息?

2.2.3.2 网络应用对传输服务的要求

> 典型网络应用对传输服务的需求

Application	Data loss	Bandwidth	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

- 数据丢失/可靠性
  - 某些网络应用能容忍一定程度的数据丢失，如网络电话
  - 某些网络应用要求100%可靠的数据传输，如文件传输、telent
- 时间/延迟
  - 有些应用只有在延迟足够低时才"有效"
  - 网络游戏/网络电话
- 带宽
  - 某些应用只有在带宽打到最低要求时才"有效"，如网络视频
  - 某些应用能够适应任何带宽-----弹性应用：Email
- 安全性....

2.2.3.3 Internet提供的传输服务

❖TCP服务

- 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服务

- 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

2.3 Web应用

2.3.1 Web

- World Wide Web
  - 网页
  - 网页间互相连接
- 网页包含多个对象
  - 对象：HTML文件、图片、视频动态脚本等
  - 基本HTML文件：包含对其他对象引用的链接
- 对象的寻址

- URL-统一资源定位器
  - URL的一般语法格式为(带方括号[]的为可选项):

    protocol :// hostname[:port] / path / [;parameters][?query]#fragment

    如：(http://)www.someschool.edu/someDept/pic.gif

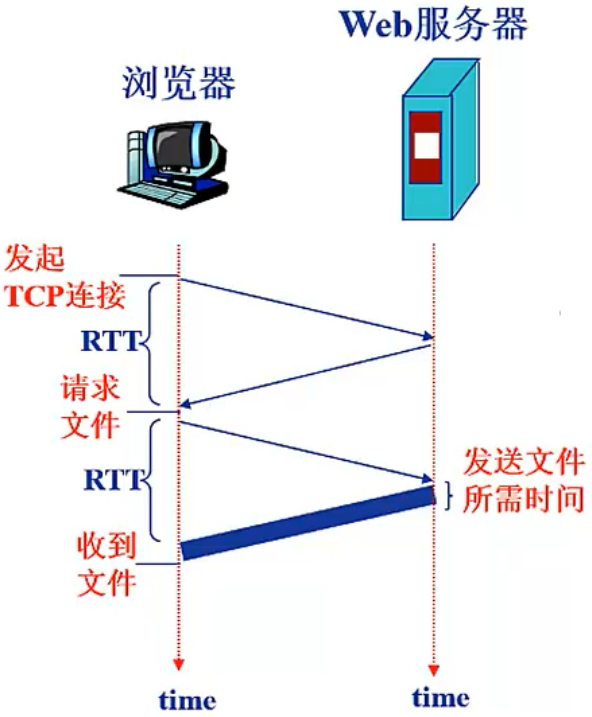
2.3.2 HTTP概述

- 万维网遵循什么协议?
  - HTTP(HyperText Transfer Protocol) 超文本传输协议
- 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
  - HTTP/1.0：RFC1945 非持续连接
  - HTTP/1.1：RFC 2068 持续连接
- HTTP使用TCP传输服务
  1. 服务器在80端口等待客户的请求
  2. 浏览器发起到服务器的TCP连接（创建套接字Socket）
  3. 服务器接受来自浏览器的TCP连接
  4. 浏览器（HTTP客户端）与Web服务器（HTTP服务器端）交换HTTP信息
  5. 关闭TCP连接
- HTTP是无状态的
  - 服务器不维护任何有关客户端过去所发请求的信息
    - 解决办法：Cookie

2.3.2.1 HTTP连接的类型

- 非持久性连接
  - 每个TCP连接最多允许传输一个对象
  - HTTP/1.0
- 持久性连接
  - 每个TCP连接允许传输多个对象
  - HTTP/1.1

通信过程



Tip：**图为传送一个文件的示意图，使用非持续连接时，若请求多个文件，每一个文件都要重复这个过程，使用持续连接时，只需要一次TCP连接建立，每个文件占用一个RTT+文件发送时间；此外，持续连接还支持流水线的发送方式，时间会更短，最少只需要3RTT+文件发送时间**

- RTT：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
    1. 发起、建立TCP连接：1个RTT
    2. 发送HTTP请求信息到HTTP响应信息的前几个字节到达：1RTT
    3. 响应信息中所含的文件/对象传输时间
    4. 总时间：2RTT+文件发送时间

2.3.2.2 HTTP消息格式

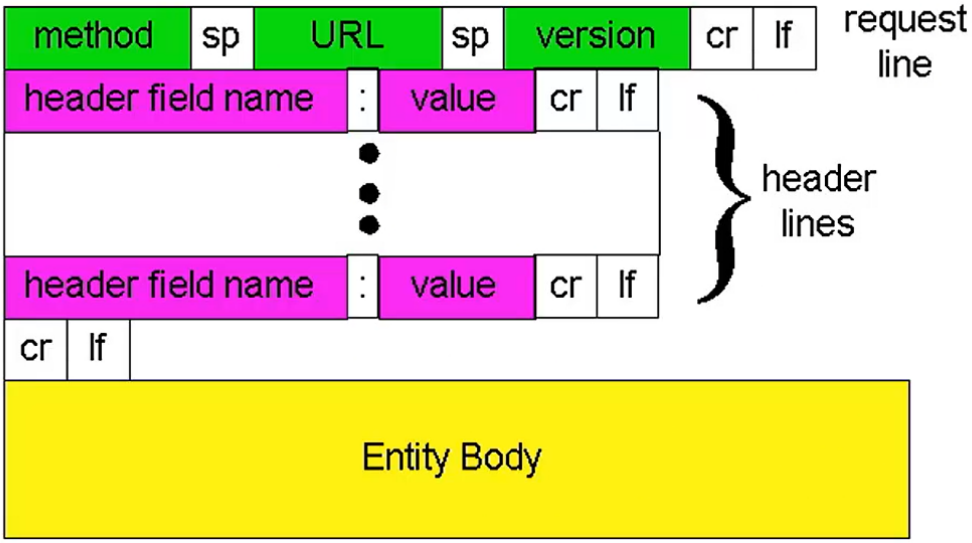
HTTP协议有两类消息

- 请求消息
- 响应消息

1. HTTP请求消息

- ASCII：人直接可读

> HTTP请求报文格式：



请求报文怎么上传输入数据?

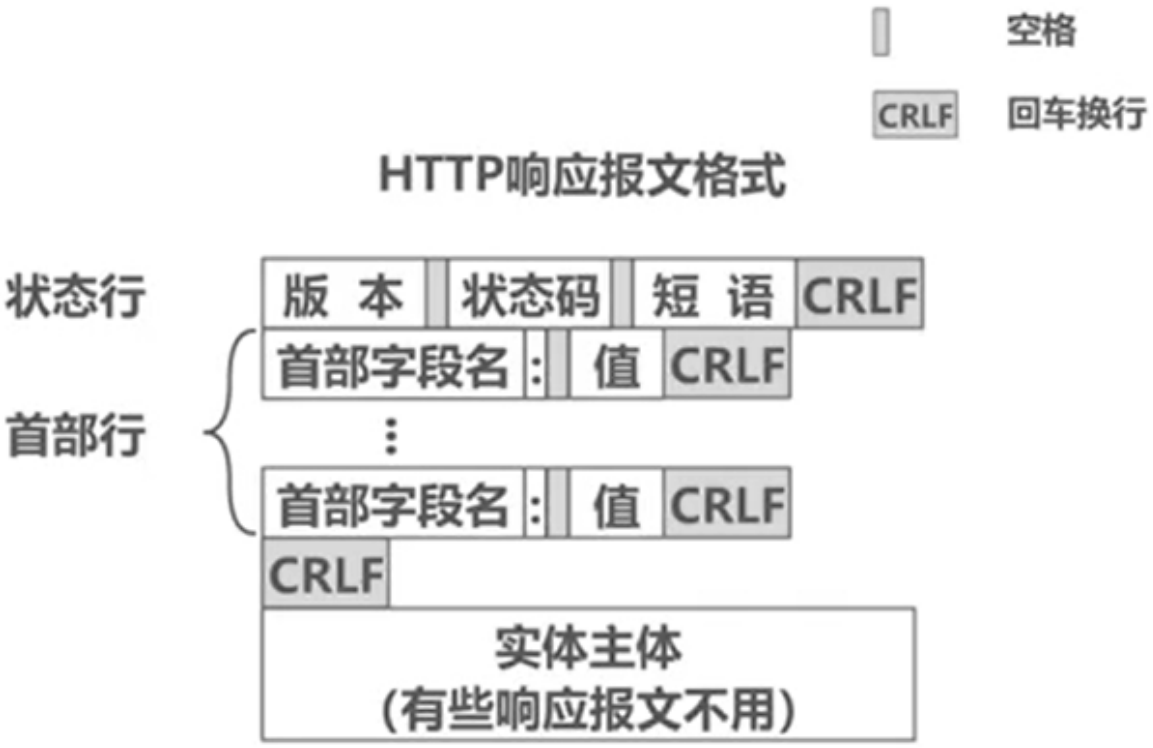
- POST方法
  - 网页经常需要填写表格
  - 在请求消息的消息体（Entity Body）中上传客户端的输入
- URL方法
  - 使用GET方法
  - （信息较少时）输入信息通过request行的URL字段上传

具体协议中的方法类型

- HTTP/1.0
  - GET
  - POST
  - HEAD
    - 请Server不要将所请求的对象放入响应消息中（测试用）
- HTTP/1.1
  - GET，POST，HEAD
  - PUT
    - 将消息体中的文件上传到URL字段指定的路径
  - DELETE
    - 删除URL字段所指定的文件

2. HTTP响应消息

- ASCII：人直接可读



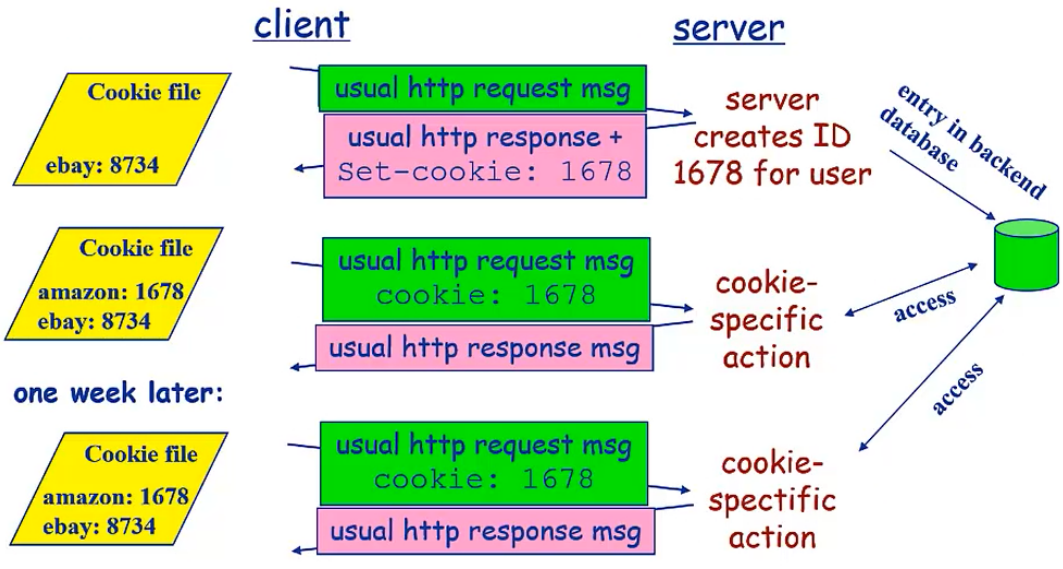
- 状态码来返回网页状态

2.3.3 Cookie

因为HTTP是无状态连接，可以使用Cookie来辨别用户身份，进行session跟踪，Cookie储存在用户本地终端

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原理
>
> 
>

Cookie的作用

- 身份认证
- 购物车

- 推荐
- Web E-Mail...

Cookie带来了隐私问题

2.3.4 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

发明Web缓存的原因?

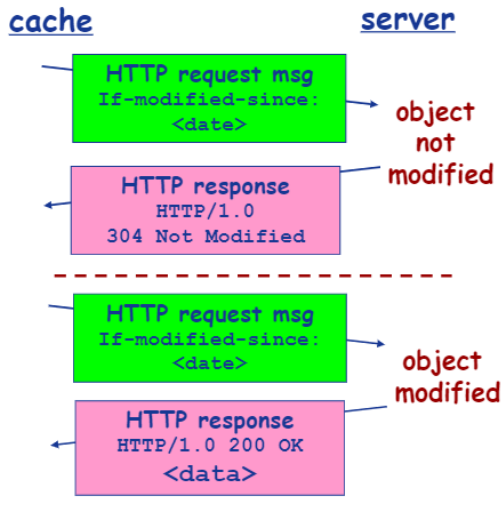
- 缩短客户请求的响应时间
- 减少机构/组织的流量
- 在大范围内(Internet)实现有效的内容分发

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
  - 如果所请求的对象在缓存中，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架设

条件性GET方法-防止缓存服务器中的文件与远端服务器的文件版本不同

- 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



2.4 Email应用

Email应用的构成组件

- 邮件客户端
  - 读、写Email消息
  - 与服务器交互，收、发Email消息
  - Web 客户端
- 邮件服务器
  - 为每一个用户分配一个邮箱：存储到发送到该用户的Email
  - 提供消息队列：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
- 服务器端：接收消息的服务器

SMTP协议 RFC:2821

- 使用TCP协议进行Email消息的可靠传输
- 持久性连接
- 端口25
- 传输过程的三个阶段
  - 握手
  - 消息的传输
  - 关闭
- 采用命令/响应交互模式
  - 命令：ASCII文本
  - 响应：状态代码和语句
- Email消息只能包含7位ASCII码
- SMTP服务器利用CRLF.CRLF（回车换行）确定消息的结束

SMTP交互实例

```
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
```

SMTP与HTTP对比

HTTP：推

SMTP：拉

它们都使用命令/响应交互模式

命令和状态代码都是ASCII码

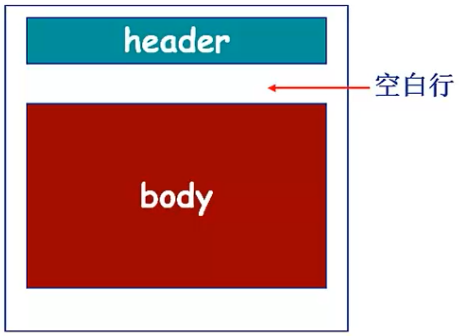
HTTP：每个对象都封装在独立的响应消息中

SMTP：多个对象在由多个部分构成的消息中发送

Email消息格式

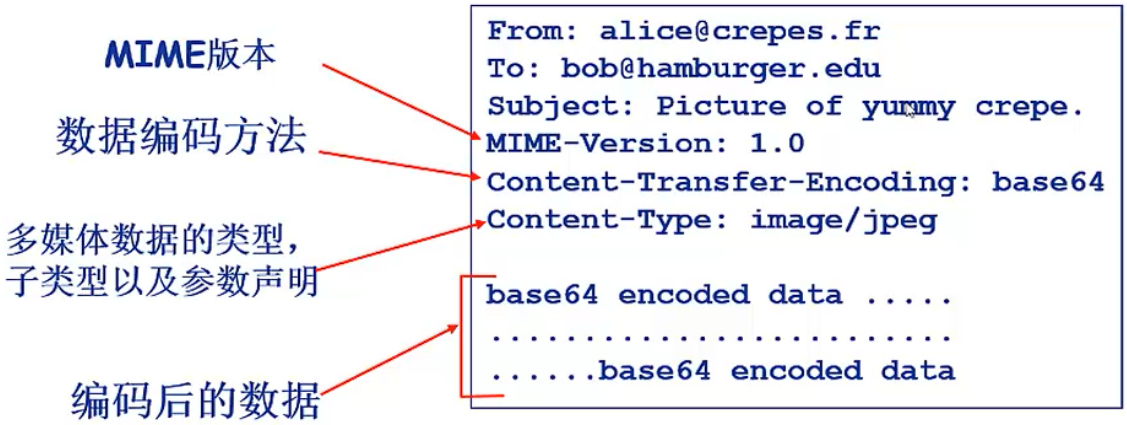
- SMTP：email消息的传输/交换协议
- RFC 822：文本消息格式标准
  - 头部行(header)
    - To
    - From
    - Subject

  与SMTP命令不同
  - 消息体(body)
    - 消息本身
    - 只能是ASCII字符



因为只能传送ASCII字符，故RFC 2045,2056引入了**MIME**多媒体邮件扩展

- 通过在邮件头部增加额外的行以声明MIME的内容类型

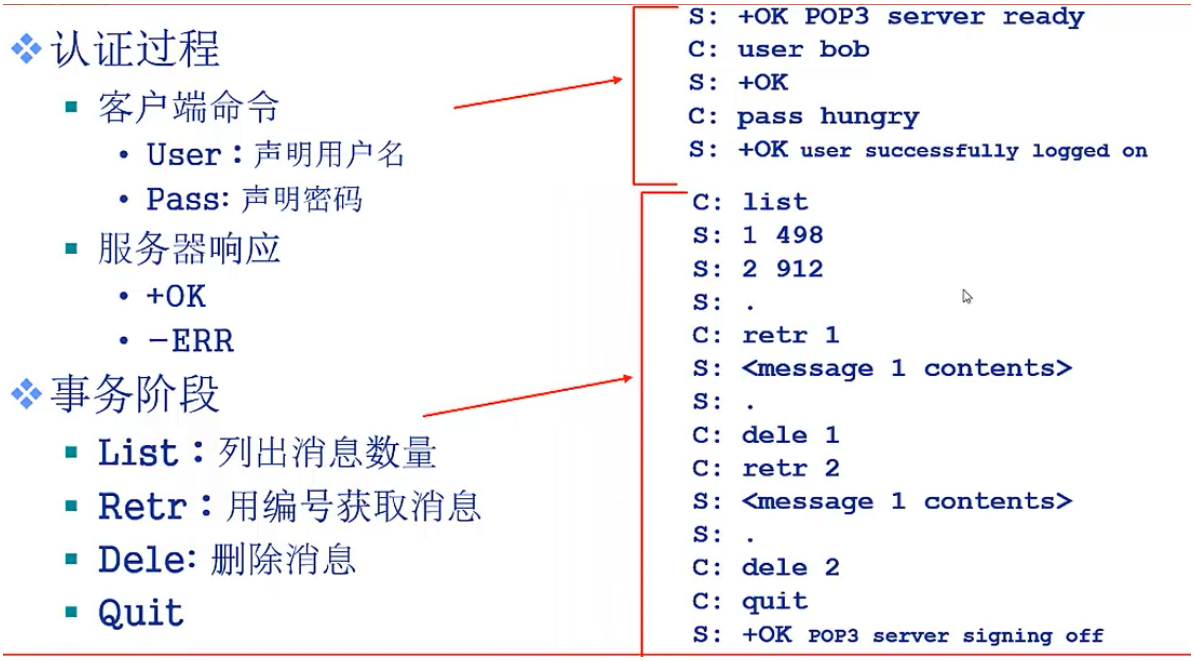


邮件访问协议

从服务器获取邮件

1. POP： Post Office Protocol [RFC 1939]
   - 认证/授权（客户端⇔服务器）和下载
2. 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信息
3. HTTP协议：163、QQmail等

POP3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
  - 不同客户端都可保留消息的拷贝
- POP3是无状态的协议

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支持跨会话的用户状态
  - 文件夹的名字
  - 文件夹与ID之间的映射等

2.4 DNS应用

- Internet上主机/路由器的识别问题
  - IP地址
  - 域名

域名和IP地址之间映射？→ 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

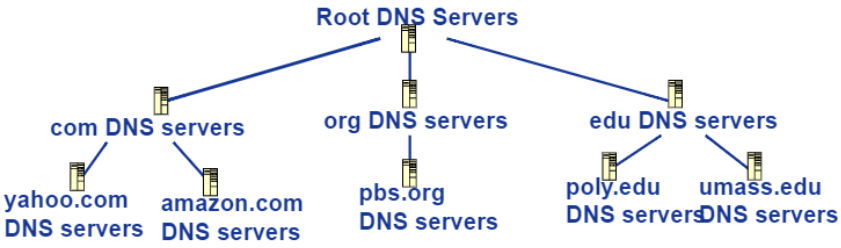
DNS服务

- 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（当进行翻译时，提供多个映射，轮流提供服务）

为什么不采用集中式DNS?

1. 单点失效问题
2. 流量问题
3. 距离问题
4. 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

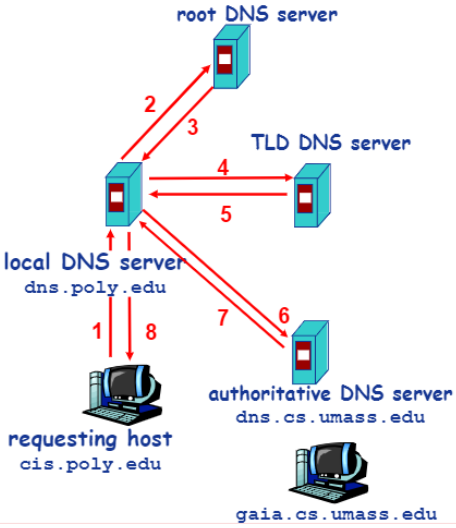
1. DNS根域名服务器
   - 本地域名服务器无法解析域名时，访问根域名服务器
     - 根域名服务器如果不知道映射，访问权威域名服务器
     - 获得映射
     - 向本地域名服务器返回映射
2. DNS顶级域名服务器
   - 负责.com、org、edu、net等顶级域名或国家顶级域名如cn、uk、fr等
3. 权威域名服务器
   - 组织的域名解析服务器，提供组织内部服务器的解析服务
     - 组织负责维护
     - 服务提供商负责维护
4. 本地域名解析服务器
   - 不严格属于层次体系
   - 每个ISP有一个本地域名服务器
     - 默认域名解析服务器
   - 当主机进行DNS服务查询时，查询被发送到本地域名服务器
     - 本地应服务器作为代理，将查询转发给（层级式）域名解析服务系统

DNS查询

1. 迭代查询

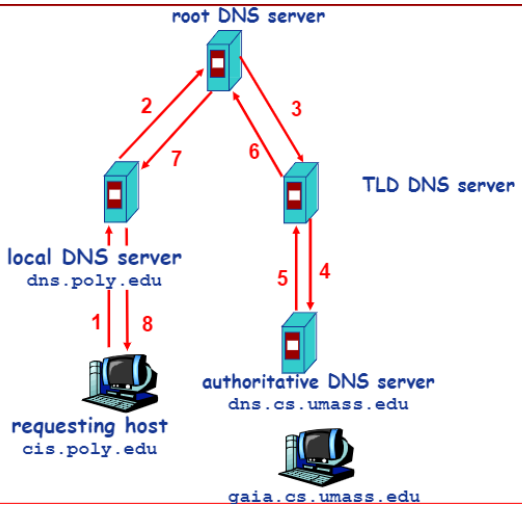
- Cis.poly.edu的主机想获得gaia.cs.umass.edu的IP地址

- 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”



2. 递归查询

- 递归查询
  - 将域名解析的任务交给所联系的服务器



DNS缓存和更新

只要域名解析服务器获得域名-IP映射，就缓存这个映射

- 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

DNS记录

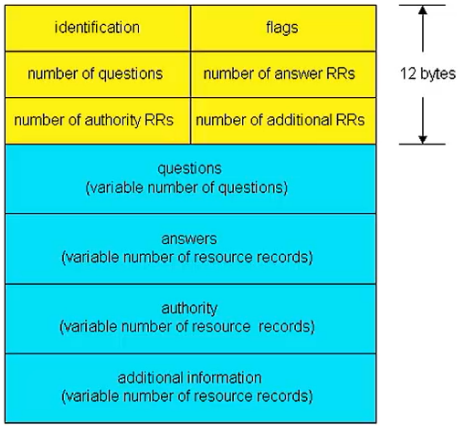
- 资源记录(RR, resource records)

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

- 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
- 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

- 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



怎样注册域名?

- 例子：你刚刚创建了一个公司 “Network Utopia”
- 在域名管理机构(如Network Solutions)注册域名networkutopia.com
  - 向域名管理机构提供你的权威域名解析服务器的名字和IP地址
  - 域名管理机构向com顶级域名解析服务器中插入两条记录

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

- 在权威域名解析服务器中为www.networkuptopia.com加入Type A记录，为networkutopia.com加入Type MX记录

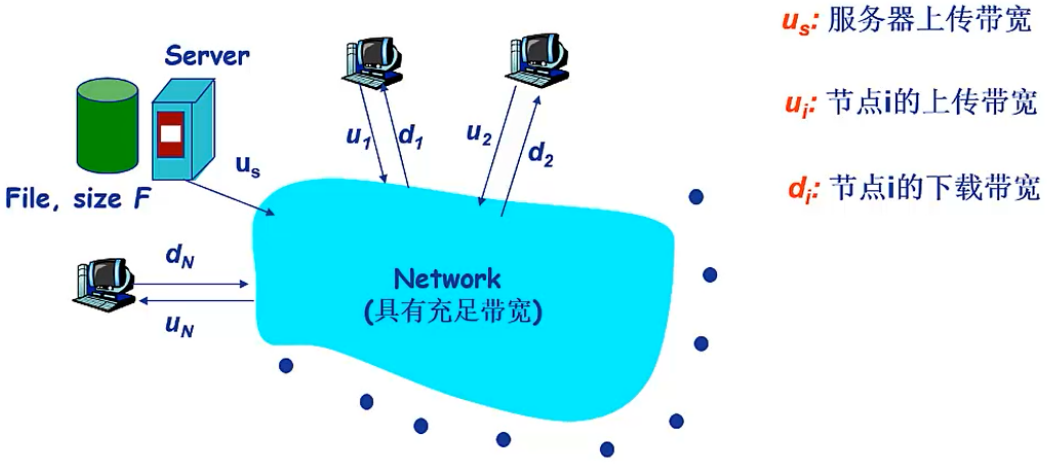
2.5 P2P应用

回顾:

- P2P没有服务器
- 任意端系统之间直接通信
- 节点阶段性接入Internet
- 节点可能更换IP地址

2.5.1 文件分发：C/S vs P2P

问题：从一个服务器向N个节点分发一个文件需要多长时间?



C/S:

- 服务器需要串行发送n个副本
  - 时间：nF/u_s
- 客户机i需要F/d_i时间下载

分发需要的时间为$d_{c/s} = max\{nF/u_s, F/min\{d_i\}\}$

P2P:

- 服务器必须发送一个副本
  - 时间：F/u_s
- 客户机i需要F/d_i时间下载
- 总共需要下载nF比特
- 最快可能的上传速度为：$u_s + \sum u_i$

分发需要的时间为$d_{p2p} = max\{F/u_s, F/min\{d_i\}, nF/(u_s + \sum u_i)\}$

[2.5.2 P2P应用具体例子没有打]

2.5.3 P2P应用：索引技术

- P2P系统的索引：信息到节点位置（IP地址+端口号）的映射
  - 文件共享（如电驴）
    - 利用索引动态跟踪节点所共享的文件的位置
    - 节点需要告诉索引它拥有哪些文件
    - 节点搜索索引，从而得知能获得哪些文件
  - 即时消息（如QQ）
    - 索引负责将用户名映射到位置
    - 当用户开启IM应用时，需要通知索引它的位置
    - 节点检索索引，确定用户的IP地址
- 索引的类别
  - 集中式索引 - **内容和文件传输是分布式的，但内容定位是高度集中式的**
    - 查询步骤

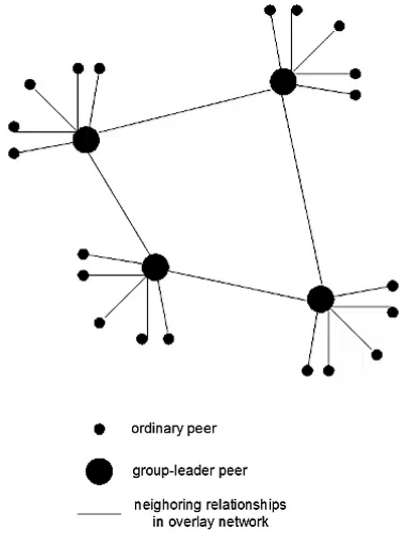
    1. 节点加入时，通知中央服务器它的IP地址、内容
    2. 用户要查找文件时，向中央服务器查询索引，中央服务器返回该文件持有者
    3. 用户与持有者之间互相传输数据，不需要经过中央服务器

    > 缺点
    >
    > - 单点失效
    > - 性能瓶颈
    > - 版权问题

  - 洪泛式查询
    - 完全分布式的架构
    - 每个节点对它共享的文件进行索引，且只对它共享的文件进行索引
    - 覆盖网络
      - 节点X与Y若有TCP连接，则构成一个边
      - 所有的活动结点和边构成覆盖网络
      - 边：虚拟链路；节点一般邻居数少于10个
    - 查询步骤

1. 节点将查询消息通过已有的TCP连接发送
2. 节点收到查询消息后，继续向周围有连接节点转发这个查询消息
3. 如果查询命中，则利用反向路径发回查询节点

- 层次式覆盖网络



  - 介于集中式索引和洪泛式查询之间的方法
  - 每个节点要么是一个超级节点，要么被分配给一个超级节点
    - 节点和超级节点之间维持TCP连接
    - 某些超级节点对之间维持TCP连接
  - 超级节点负责跟踪子节点的内容

整理自哈工大计算机网络MOOC课件

3. 传输层

本章内容

- 理解传输层服务的基本理论和基本机制
  - 多路复用/分用
  - 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
  - TCP：面向连接的传输服务
  - TCP拥塞控制

3.1 传输层服务

3.1.1 传输层服务和协议

- 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segment，并向下传给网络层
  - **接收方**：将接受到的segment组装成消息，并上交给应用层
- 传输层可以为上层提供多种协议
  - TCP、UDP

3.1.2 传输层 vs 网络层

- 网络层：提供**主机**之间的逻辑通信
- 传输层：提供**应用进程**之间的逻辑通信
  - 位于网络层之上
  - 依赖于网络层的服务
  - 对网络层服务进行（可能的）增强
- Internet传输层协议
  - 可靠、按需交付服务（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”的网络层，没有做（可靠性方面的）扩展

3.2 复用和分用

- 为什么要进行复用分用?
  - 如果某层的协议对应直接上层的多个协议/实体，则需要复用/分用
-

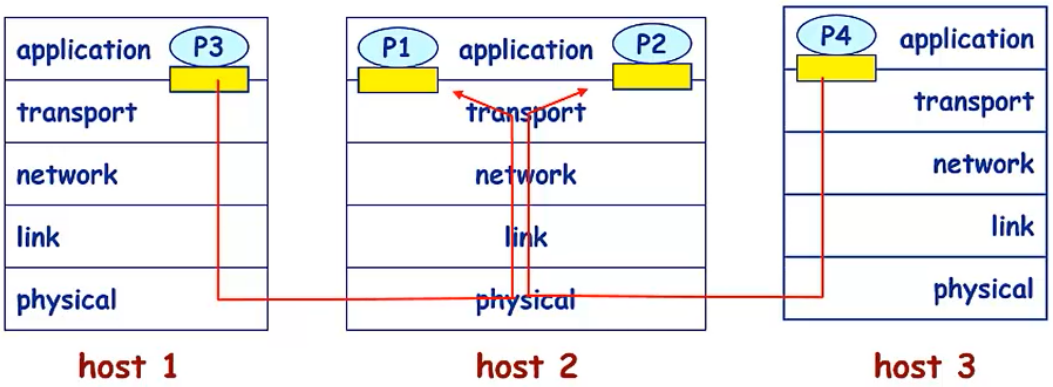
接收端进行多路分用：

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

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

= socket　= process



- **分用如何工作?**
  - 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

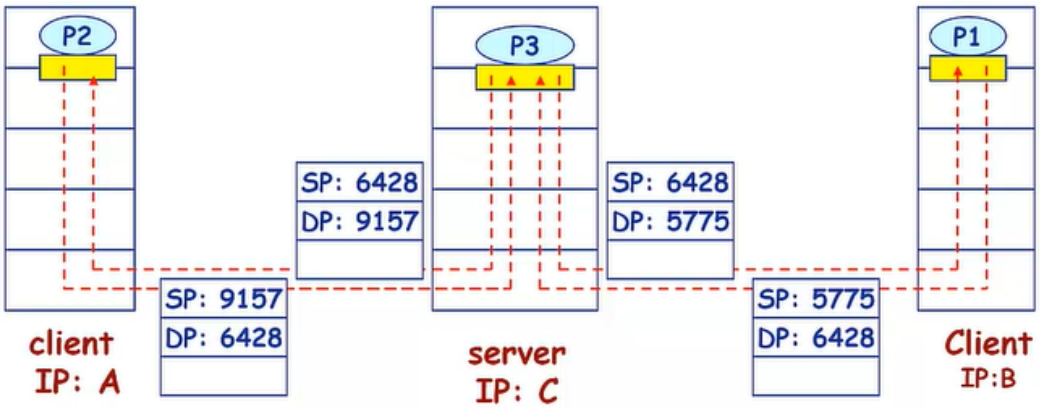
TCP/UDP 段格式

- **无连接分用**
  - 利用端口号创建Socket

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

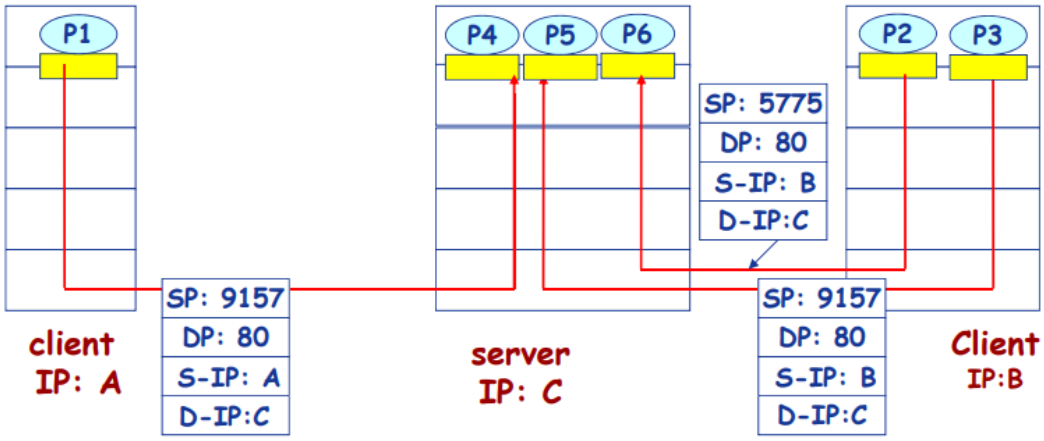
  - UDP的Socket用二元组标识
    - (目的IP地址，目的端口号)
  - 主机收到UDP段后
    - 检查段中的目的端口号
    - 将UDP段导向绑定在该端口号的Socket
  - 来自不同源IP地址和/或源端口号的IP数据包被导向同一个Socket

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

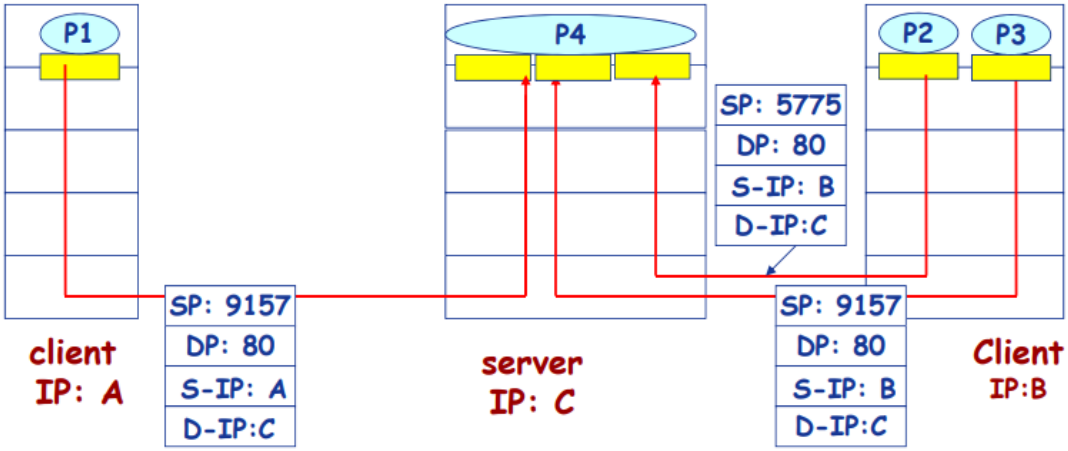


SP 提供“返回地址”

- **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

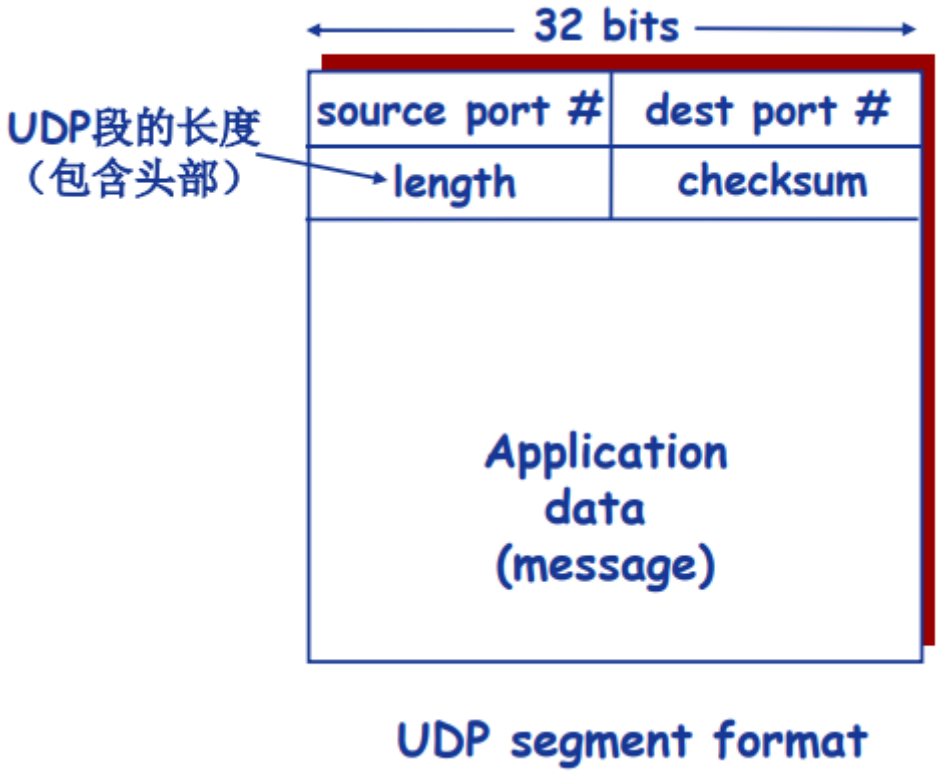


多线程Web服务器:



3.3 无连接传输协议-UDP

- 基于Internet IP协议
  - 复用/分用
  - 简单的错误校验
- 尽力而为的服务
  - UDP段可能丢失
  - UDP段可能非按序到达
- 无连接
  - UDP的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
- UDP为何存在?
  - 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：应用可以更好地控制发送时间和速率
- 常用于流媒体应用
  - 容忍丢失
  - 速率敏感
- UDP还应用于
  - DNS
  - SNMP
- 在UDP上怎样实现可靠传输?
  - 在应用层增加可靠性机制
  - 应用特定的错误恢复机制
- UDP报文段格式



UDP segment format

- UDP校验和

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

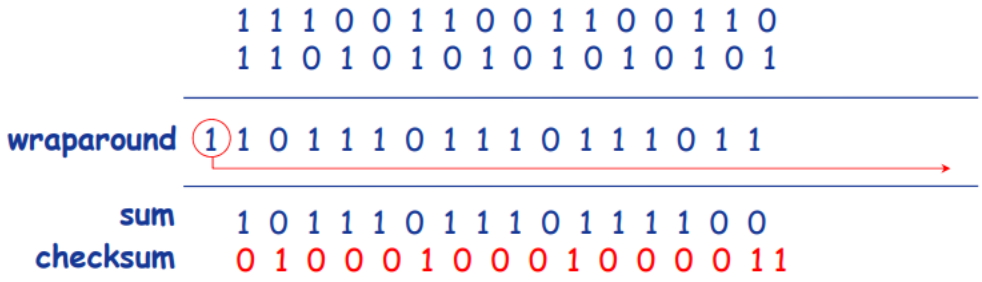
❖接收方

- 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）

❖注意：

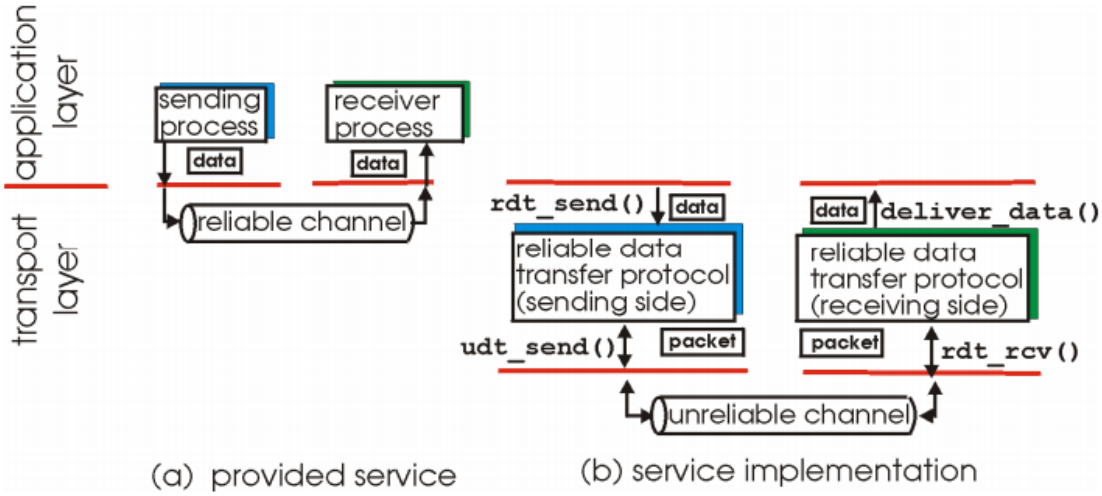
- 最高位进位必须被加进去

❖示例：

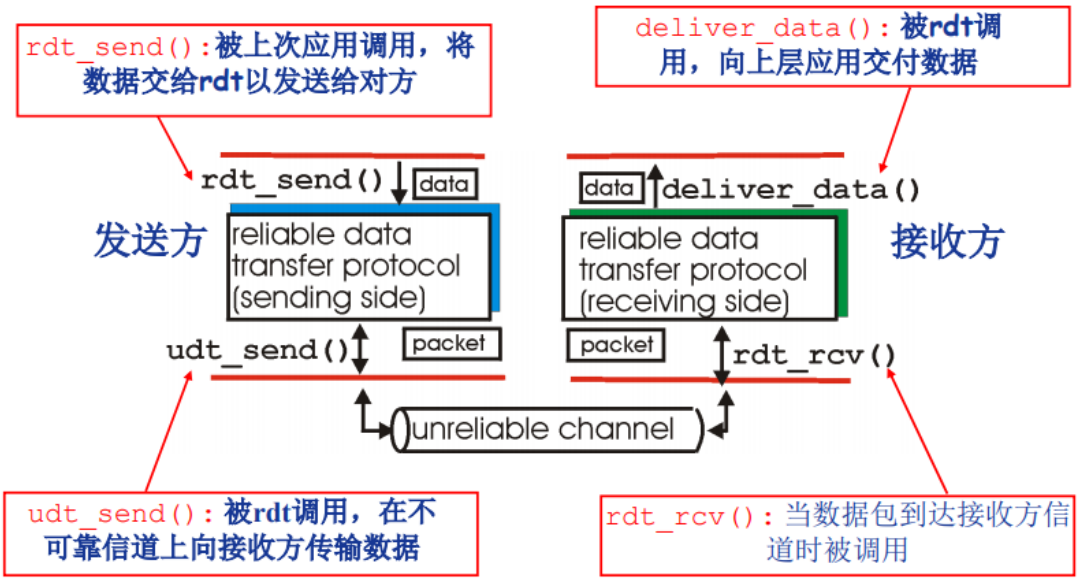


3.4 可靠数据传输的基本原理

- 什么是可靠?
  - 不错、不丢、不乱
- 可靠数据传输协议
  - 对应用层、传输层、链路层都很有用，是网络TOP-10问题
  - 信道的不可靠特性决定了可靠数据传输协议（Rdt）的复杂性



- 可靠数据传输协议基本结构：接口



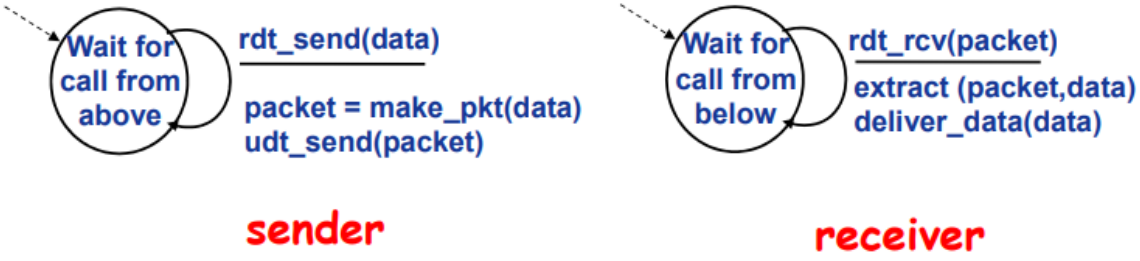
3.4.1 Rdt协议的发展

- 渐进地设计可靠数据传输协议的发送方和接收方
- 只考虑单向数据传输
  - 但控制信息双向流动
- 利用状态机(Finite State Machine, FSM)刻画传输协议

- 圆圈代表当前的状态
- 箭头代表状态之间的转换
- 横线上方代表引起状态变迁事件
- 横线下方代表当进行状态转换的过程中采取的行动
- 有限状态自动机必须准确无误地定义，因此每一个状态之间的变迁都必须准确，不可能出现一个事件同时触发两种变迁

3.4.1.1 Rdt1.0：可靠信道上的可靠数据传输

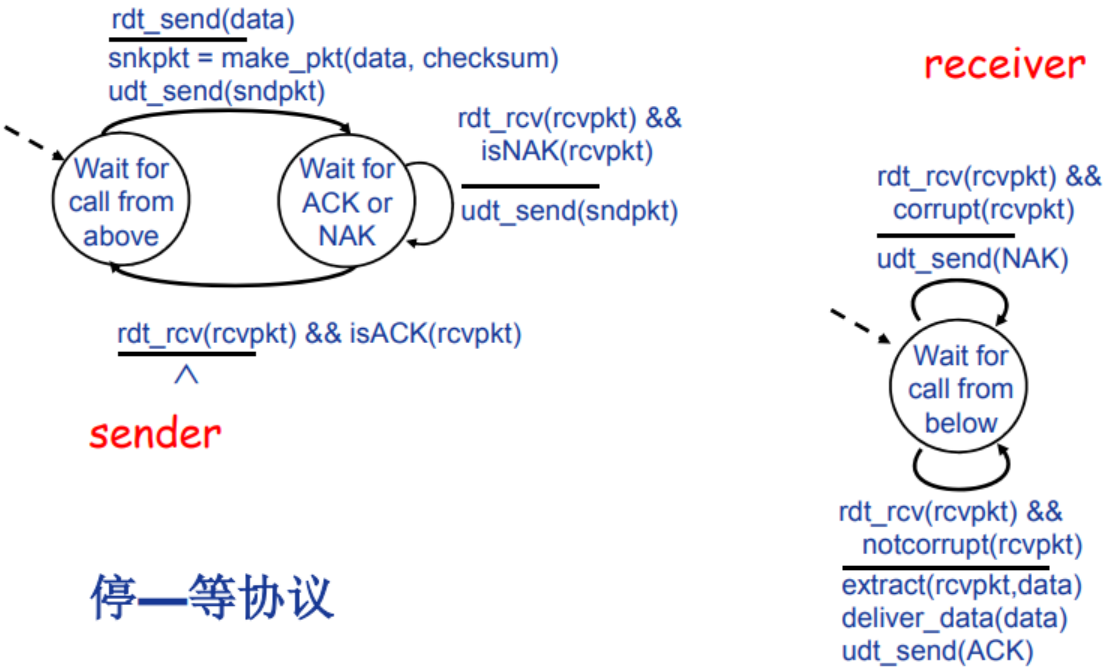
- 底层信道完全可靠
  - 不会发生错误(bit error)
  - 不会丢弃分组
- 发送方和接收方的FSM独立



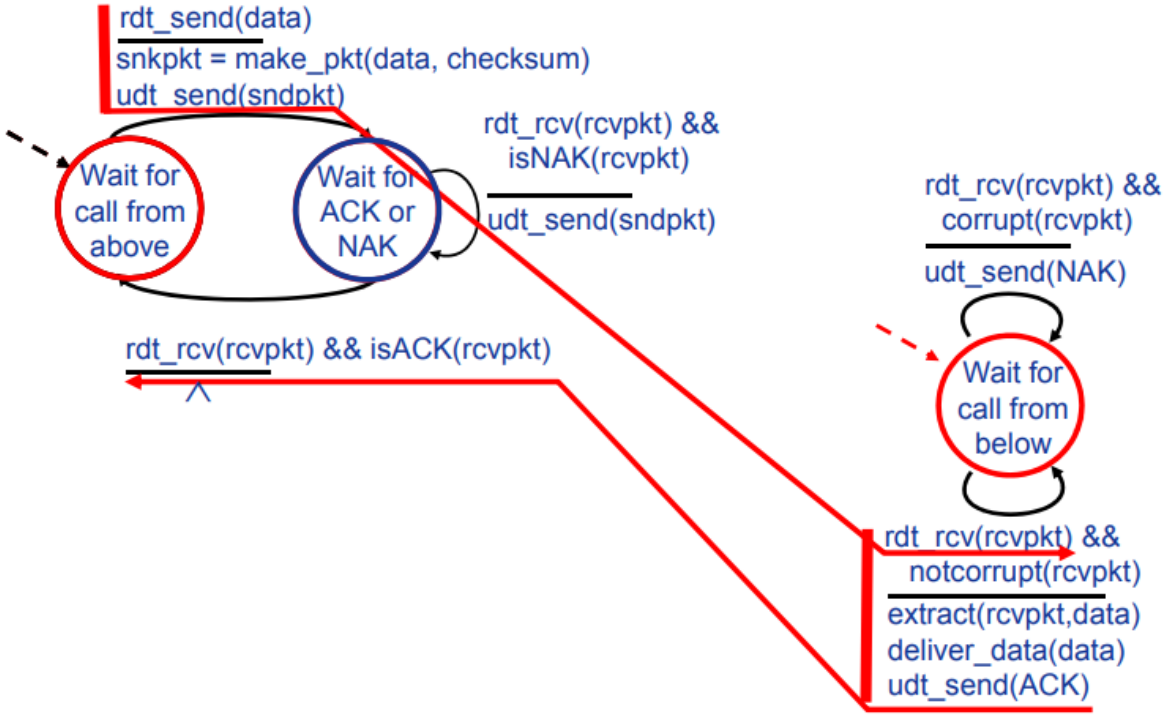
3.4.1.2 Rdt2.0：产生位错误的信道

- 底层信道可能翻转分组中的位(bit)
  - 利用校验和检测位错误
- 如何从错误中恢复?
  - 确认机制(ACK)：接收方显式地告知发送方分组已正确接受
  - NAK：接收方显式地告知发送方分组有错误
  - 发送方收到NAK，重传分组
- 基于这种重传机制的Rdt协议称为ARQ(Automatic Repeat Request)协议
- Rdt2.0引入的新机制
  - 差错检测
  - 接收方反馈控制消息：ACK/NAK
  - 重传

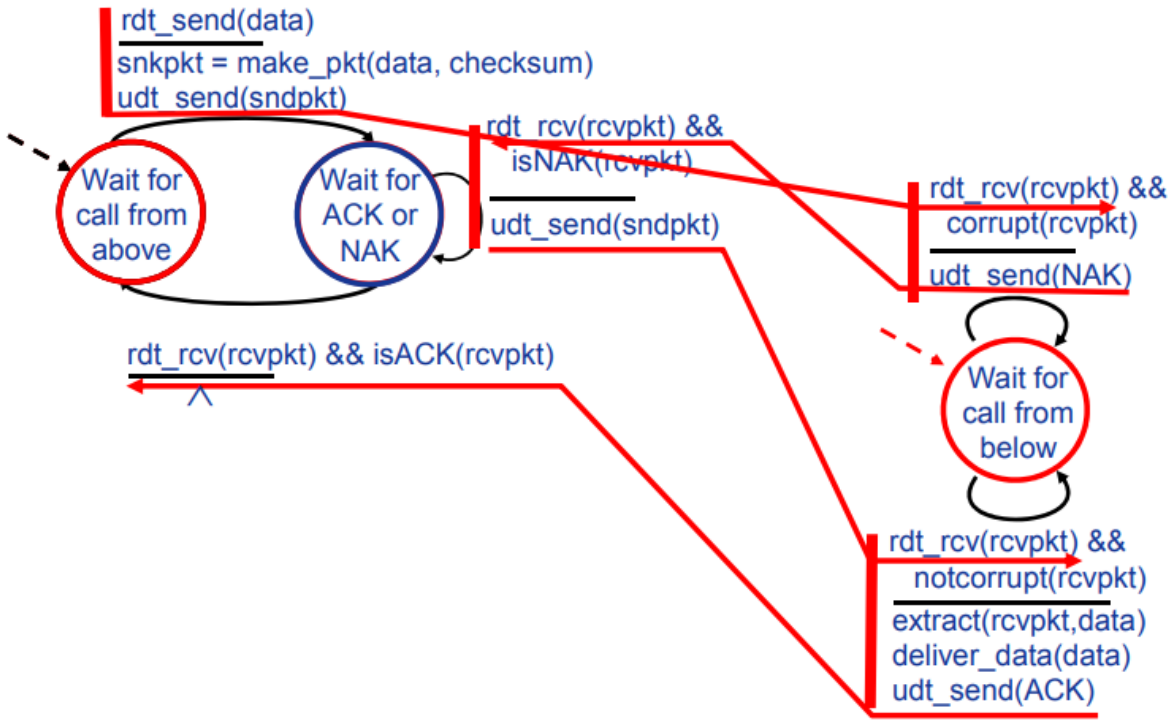
Rdt2.0：FSM规约



Rdt2.0：无错误场景



Rdt2.0：有错误场景



3.4.1.3 Rdt2.1和2.2（仍采用停-等）

Rdt 2.0有何缺陷?

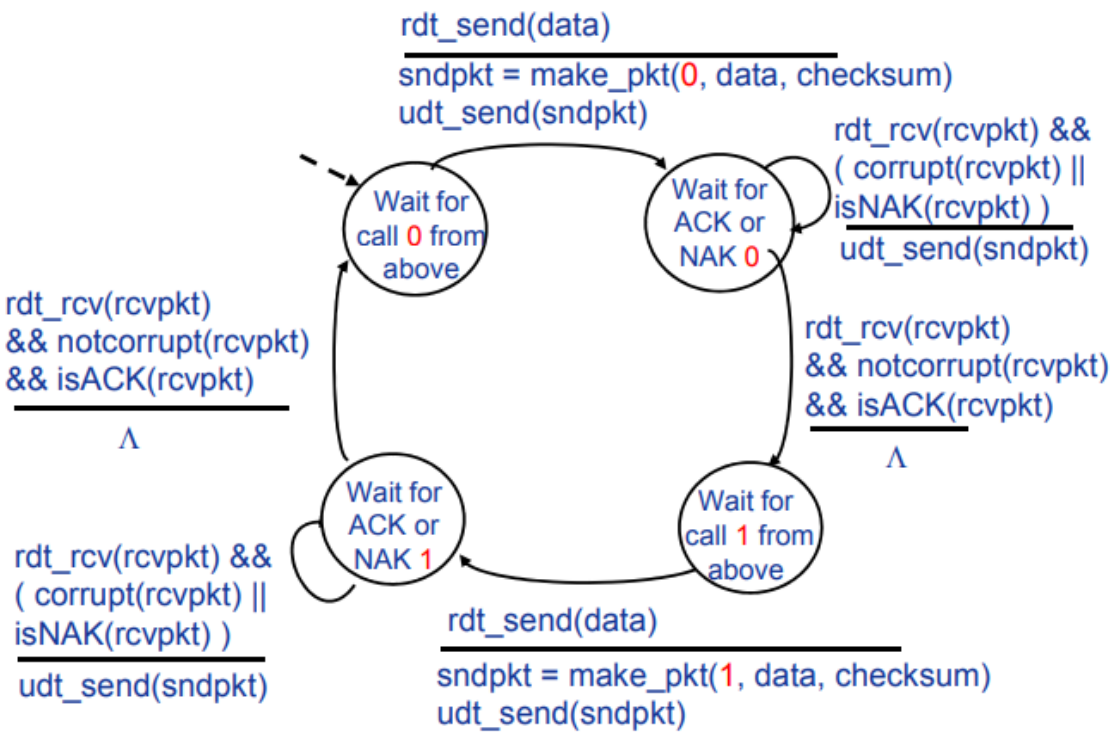
- 没有考虑ACK/NAK消息发生错误或被破坏的情况

解决方案?

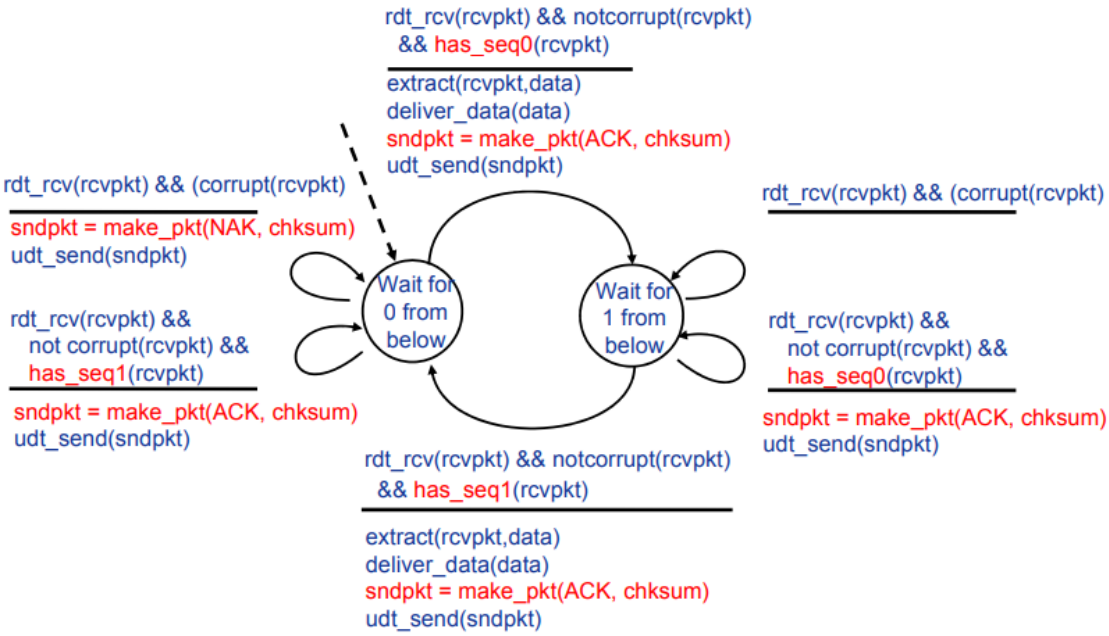
- 为ACK/NAK增加校验和，检错并纠错
- 发送方收到被破坏ACK/NAK时不知道接收方发生了什么，添加额外的控制消息
- 如果ACK/NAK坏掉，发送方重传（也是常用的方法）
    - 不能简单重传：产生分组重复问题

- 如何解决?
  - 序列号：发送方给每个分组增加序列号
  - 接收方丢弃重复分组

Rdt2.1 发送方



Rdt2.1 接收方



Rdt2.1 vs Rdt2.0

❖发送方：

- 为每个分组增加了序列号
- 两个序列号(0, 1)就够用，为什么？ 因为采用停等协议
- 需校验ACK/NAK消息是否发生错误
- 状态数量翻倍
  - 状态必须“记住”“当前”的分组序列号

❖接收方

- 需判断分组是否是重复
  - 当前所处状态提供了期望收到分组的序列号
- 注意：接收方无法知道ACK/NAK是否被发送方正确收到

Rdt 2.2：无NAK消息协议

不需要一定采用ACK+NAK这种模式

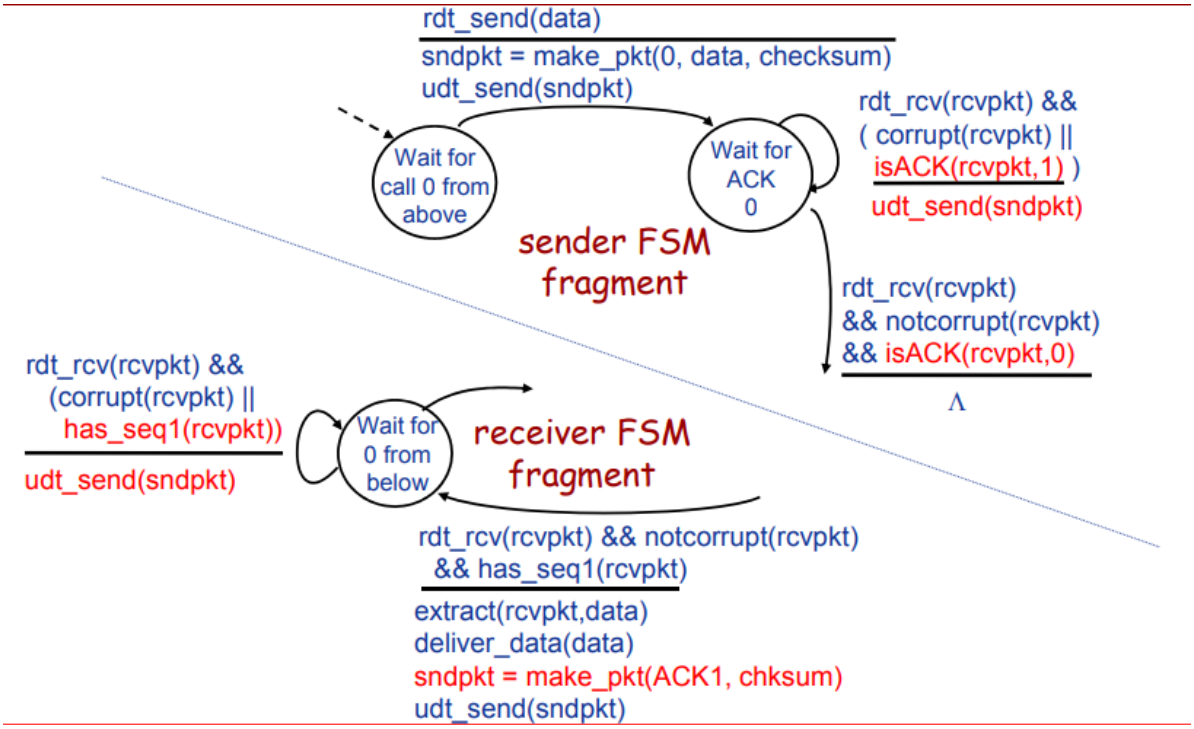
Rdt2.2与Rdt2.1功能相同，但没有NAK消息，只采用ACK，那它是如何实现呢？

- 接收方通过ACK告知最后一个被正确接收的分组
- 在ACK消息中显式地加入**被确认分组的序列号**

发送方收到重复的AKC之后，采取与收到NAK消息相同的动作

- 重传当前分组

Rdt2.2 FSM片段



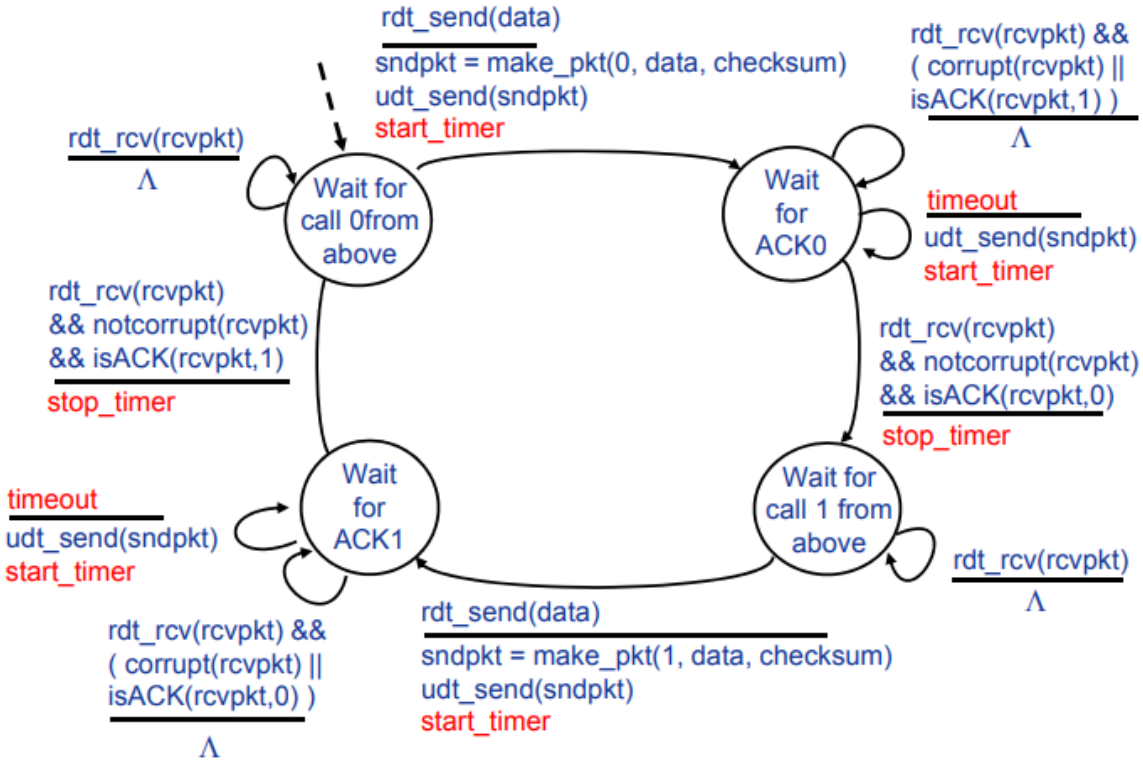
3.4.1.4 Rdt3.0（仍采用停-等）

- 若信道即可能发生错误，也可能丢失分组，“校验和+序列号+ACK+重传”不够用
- 解决办法
  - 方法：发送方等待“合理”时间
    - 如果没收到ACK，重传
    - 如果分组或ACK只是延迟而不是丢了

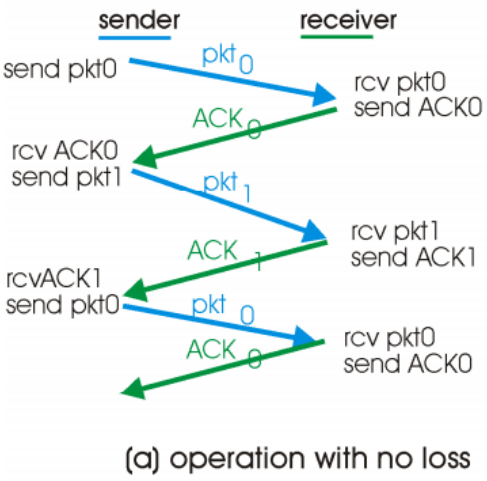
- 重传会产生重复，序列号机制能够处理
- 接收方需在ACK中显式告知所确认的分组

- 需要定时器

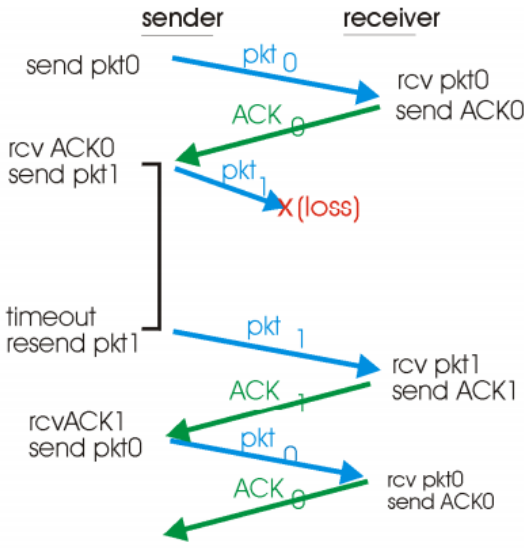
Rdt3.0 发送方FSM



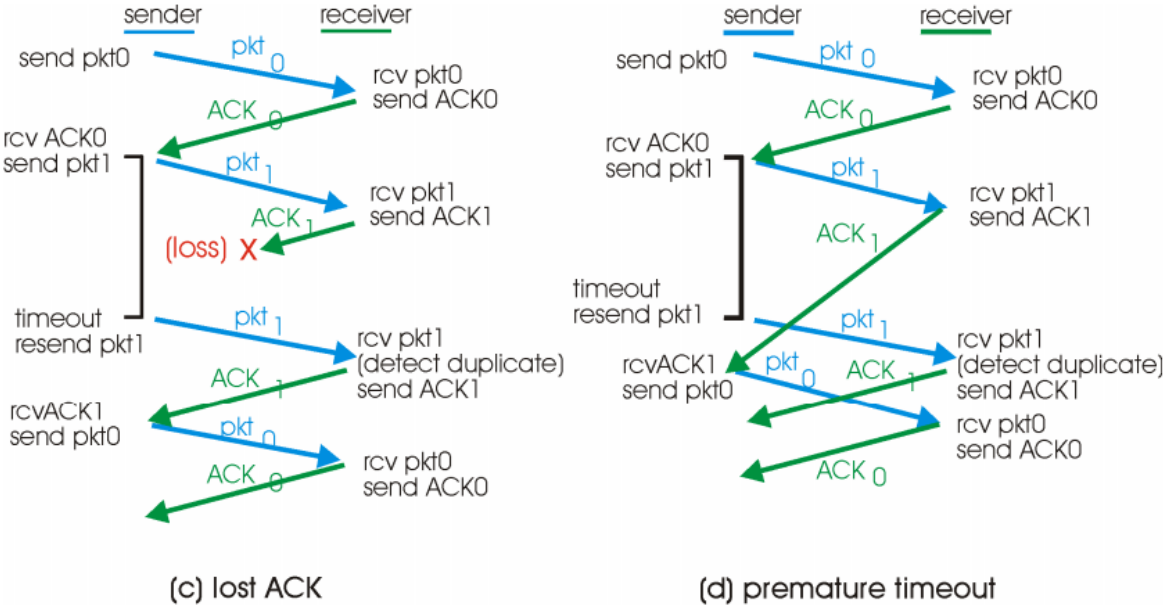
Rdt3.0 示例



(a) operation with no loss



(b) lost packet



Rdt3.0 性能分析

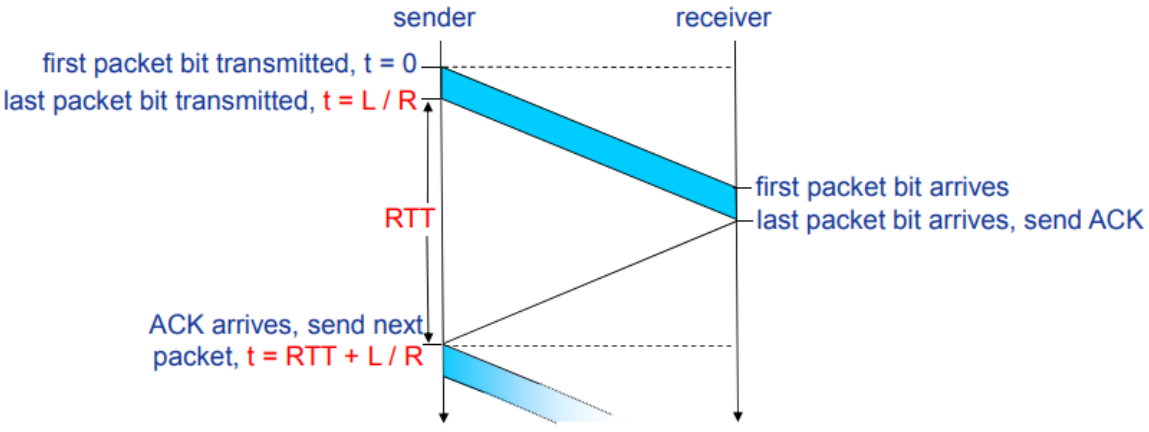
- Rdt 3.0能够正确工作，但性能很差
- 示例：1Gbps链路，15ms端到端传播延迟，1KB分组

$$T_{transmit} = \frac{L\ (\text{packet length in bits})}{R\ (\text{transmission rate, bps})} = \frac{8\text{kb/pkt}}{10^9\ \text{b/sec}} = 8\ \text{microsec}$$

- 发送方利用率：发送方发送时间百分比

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

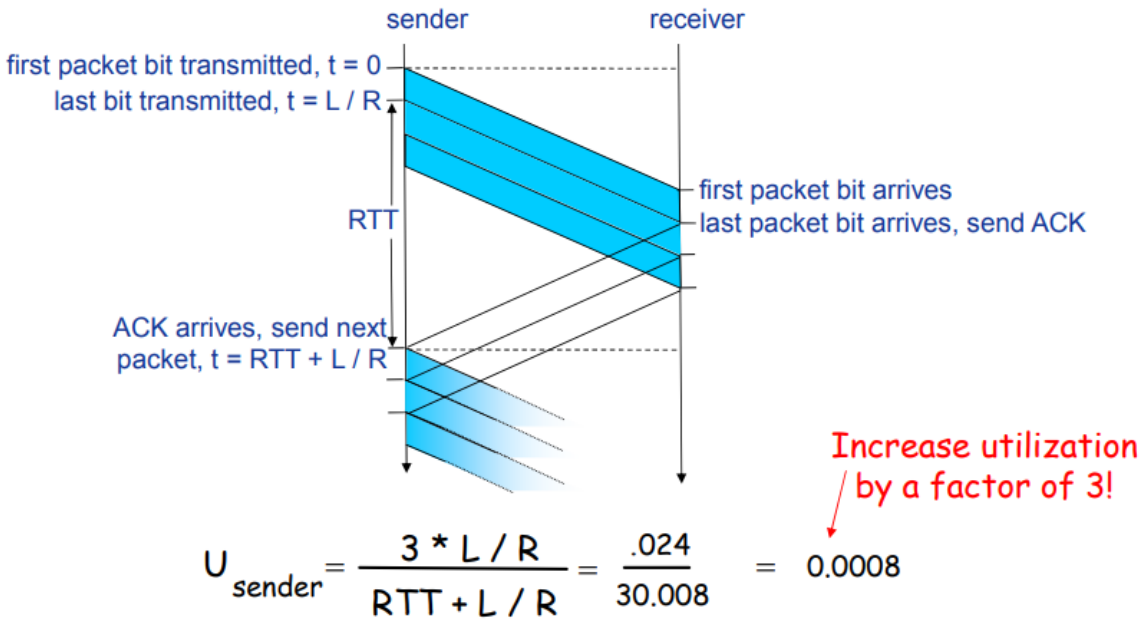
- 在1Gbps链路上每30毫秒才发送一个分组➔33KB/sec
- 网络协议限制了物理资源的利用



$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

3.5 滑动窗口协议

3.5.1 流水线机制与滑动窗口协议

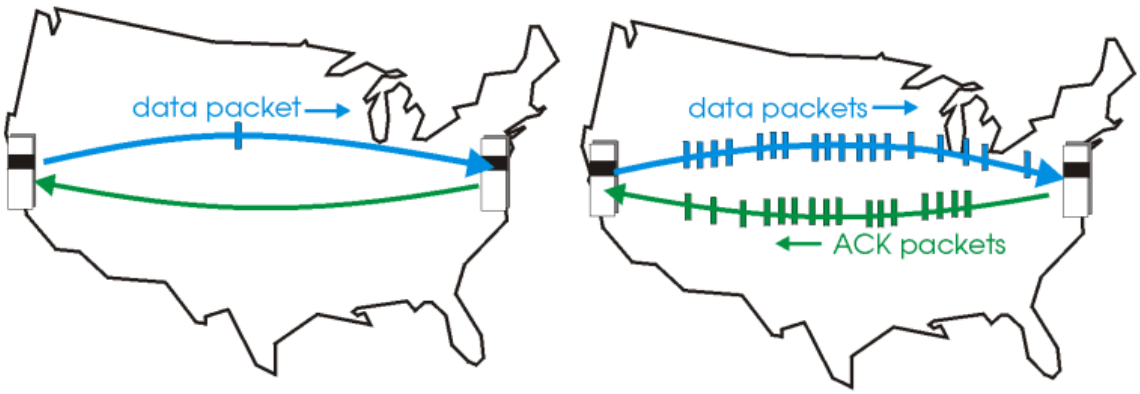


采用流水线后，与上节最后一张图对比，效率提高了3倍

流水线协议

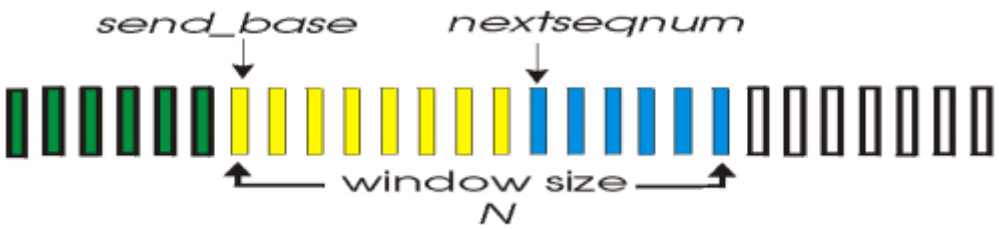
允许发送方在收到ACK之前连续发送多个分组

- 更大的序列号范围
- 发送方/接收方需要更大的存储空间以缓存分组



(a) a stop-and-wait protocol in operation　　(b) a pipelined protocol in operation

滑动窗口协议



滑动窗口协议：Sliding-window protocol

- 窗口
  - 允许使用的序列号范围
  - 窗口尺寸为N：最多有N个等待确认的消息

- 滑动窗口：
  - 随着协议的运行，窗口在序列号空间内向前滑动
- 滑动窗口协议
  - GBN、SR

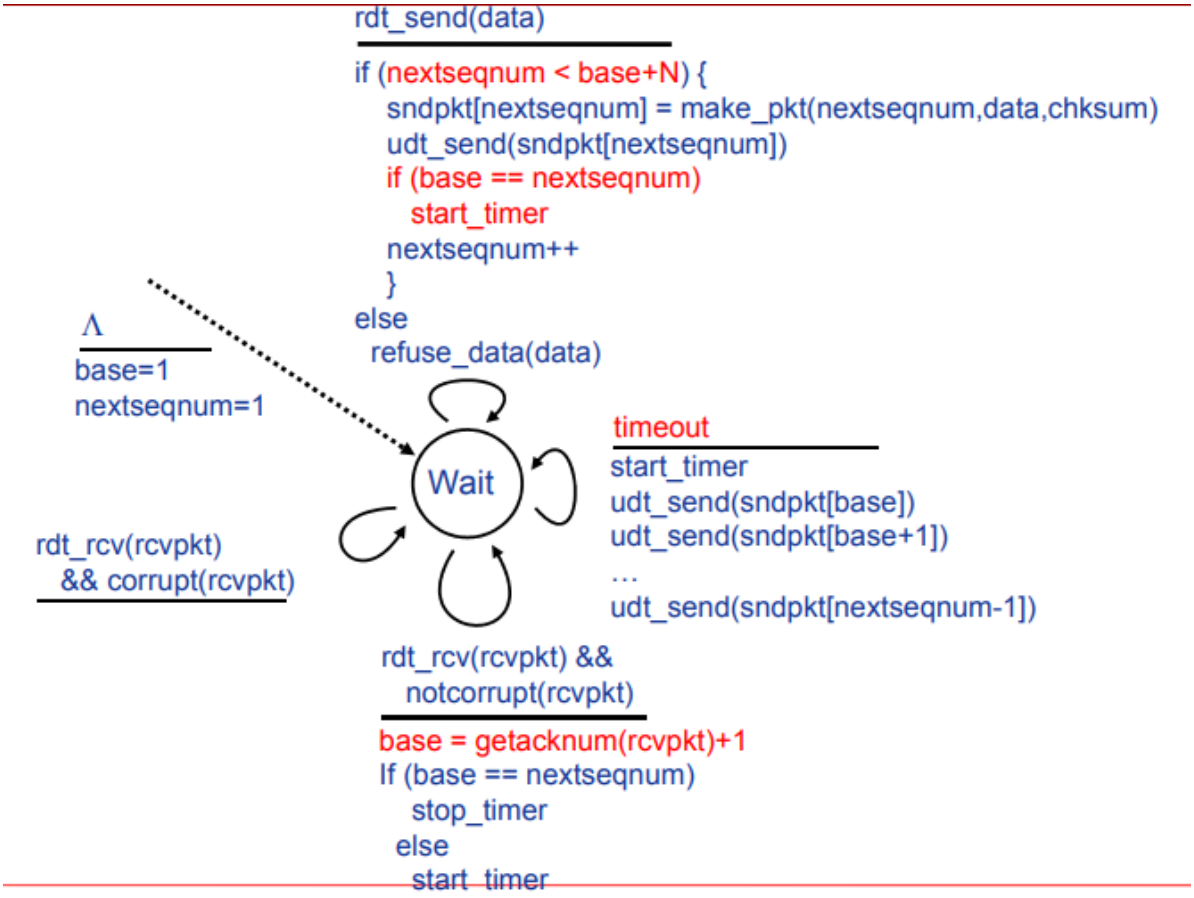
3.5.2 GBN协议（Go-Back-N）

GBN-发送方

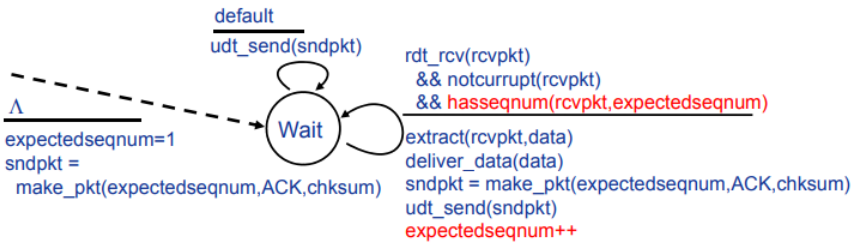
- 分组头部包含k-bit序列号
- 窗口尺寸为N，最多允许N个分组未确认

- ACK(n): 确认到序列号n(包含n)的分组均已被正确接收
  - 可能收到重复ACK
- 为空中的分组设置计时器(timer)
- 超时Timeout(n)事件: 重传序列号大于等于n，还未收到ACK的所有分组

GBN-发送方拓展FSM

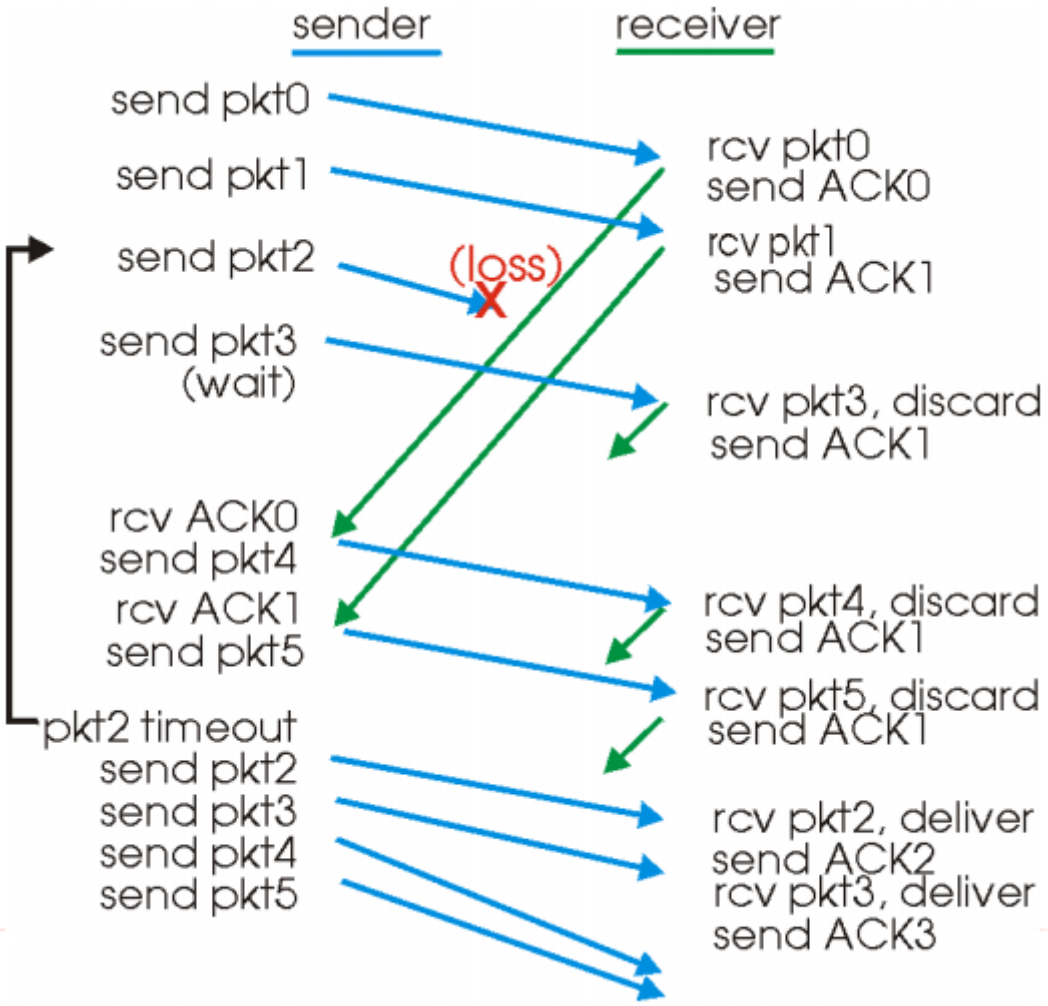


GBN-接收方拓展FSM



- ACK机制: 发送拥有最高序列号的、已被正确接收的分组的ACK
  - 可能产生重复ACK
  - 只需要记住唯一的**expectedseqnum**
- 乱序到达的分组：
  - 直接丢弃➔接收方没有缓存
  - 重新确认序列号最大的、按序到达的分组

GBN-示例



3.5.3 SR协议（Selective Repeat协议）

GBN的缺陷

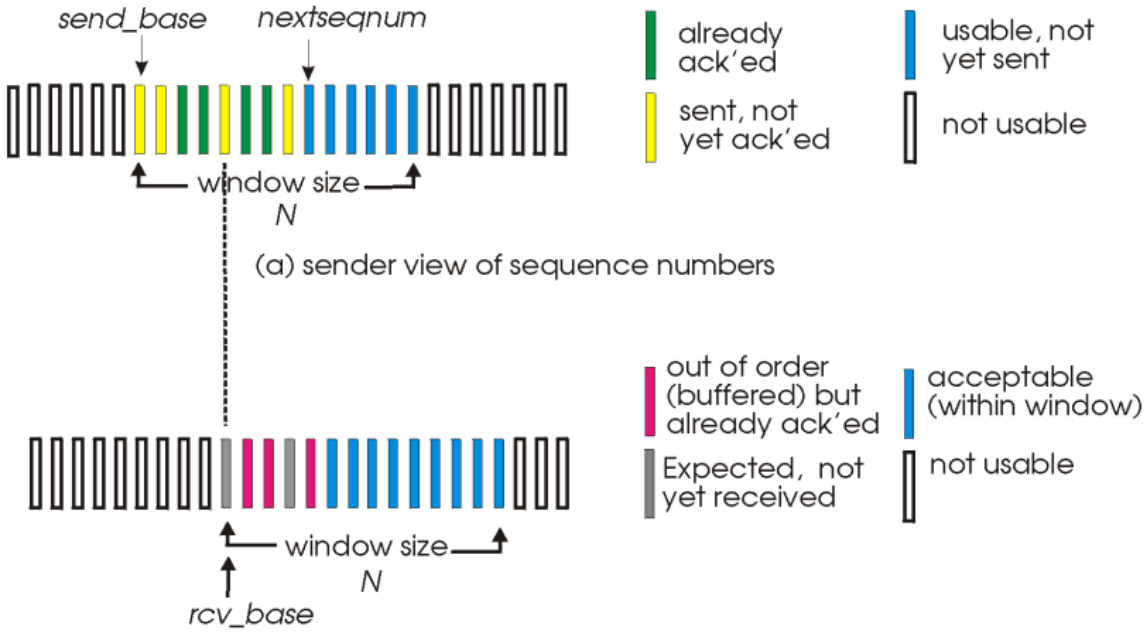
重传时会重传很多分组，有时是不必要的，会导致网络中出现很多重复分组，影响性能

SR协议

- 接收方对每个分组单独进行确认
  - 设置缓存机制，缓存乱序到达的分组
- 发送方只重传那些没收到ACK的分组
  - 为每个分组设置定时器
- 发送方窗口
  - N个连续的序列号
  - 限制已发送且未确认的分组

3.5.3.1 SR发送方、接收方

窗口：



sender

data from above :

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

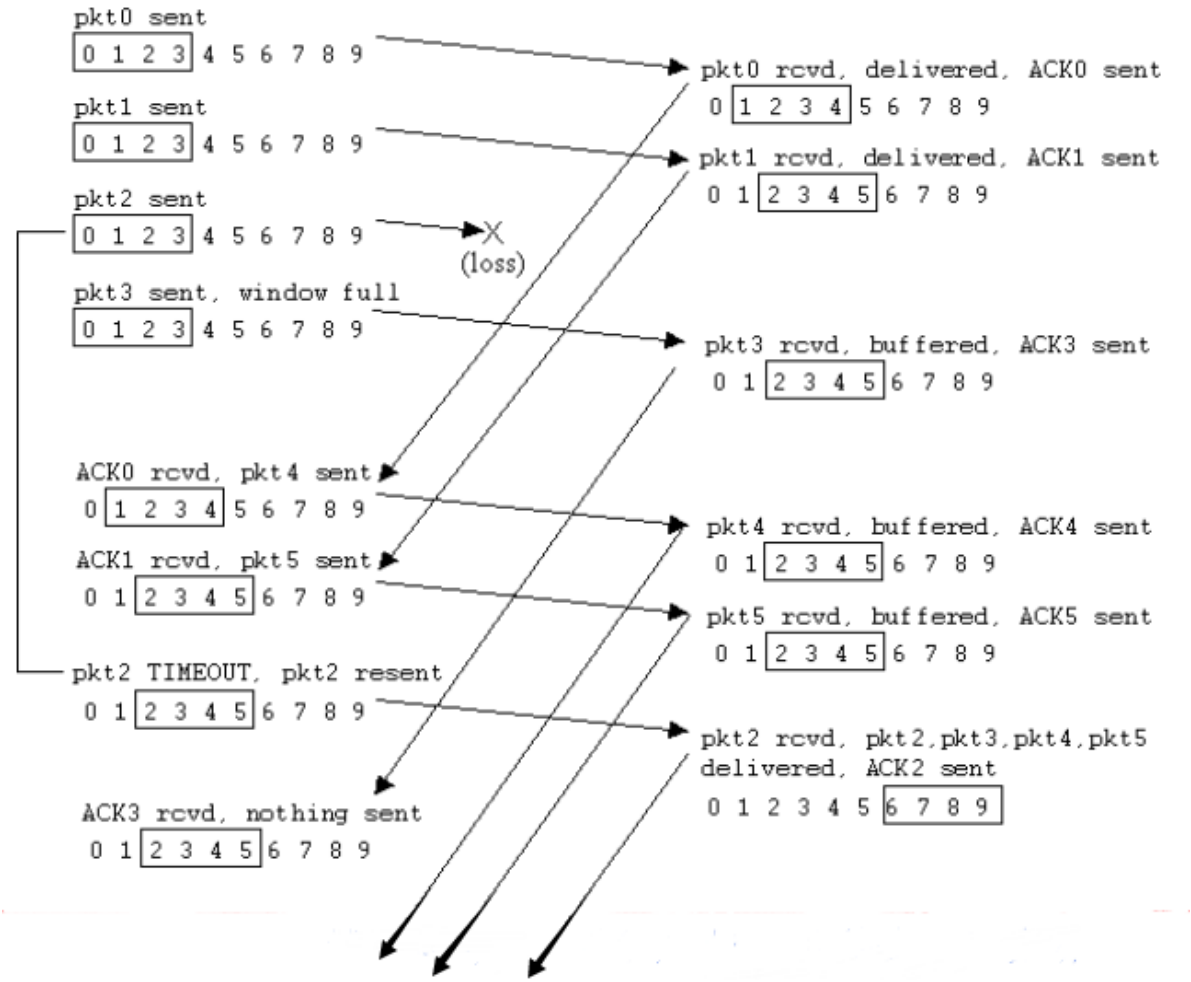
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

3.5.3.2 SR示例



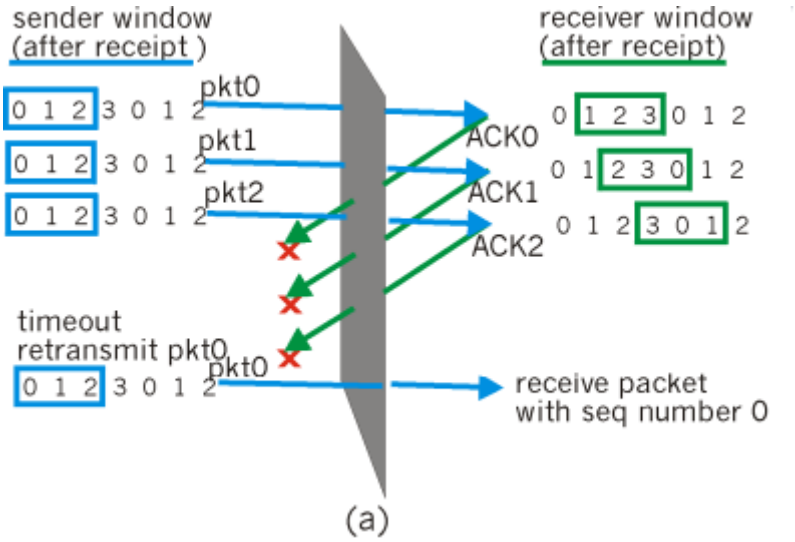
3.5.3.3 SR协议的困境

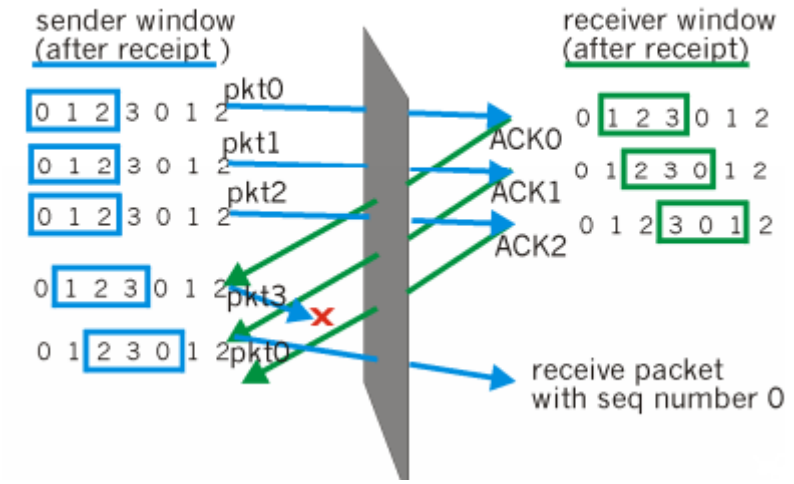
考虑如下情况：

序列号0,1,2,3

窗口尺寸为3

那么，接收方无法分辨以下两种情况：





序列号空间大小与窗口尺寸需要满足什么关系?

$N_S + N_R \leq 2^k$

3.6 面向连接的传输协议-TCP

3.6.1 TCP概述

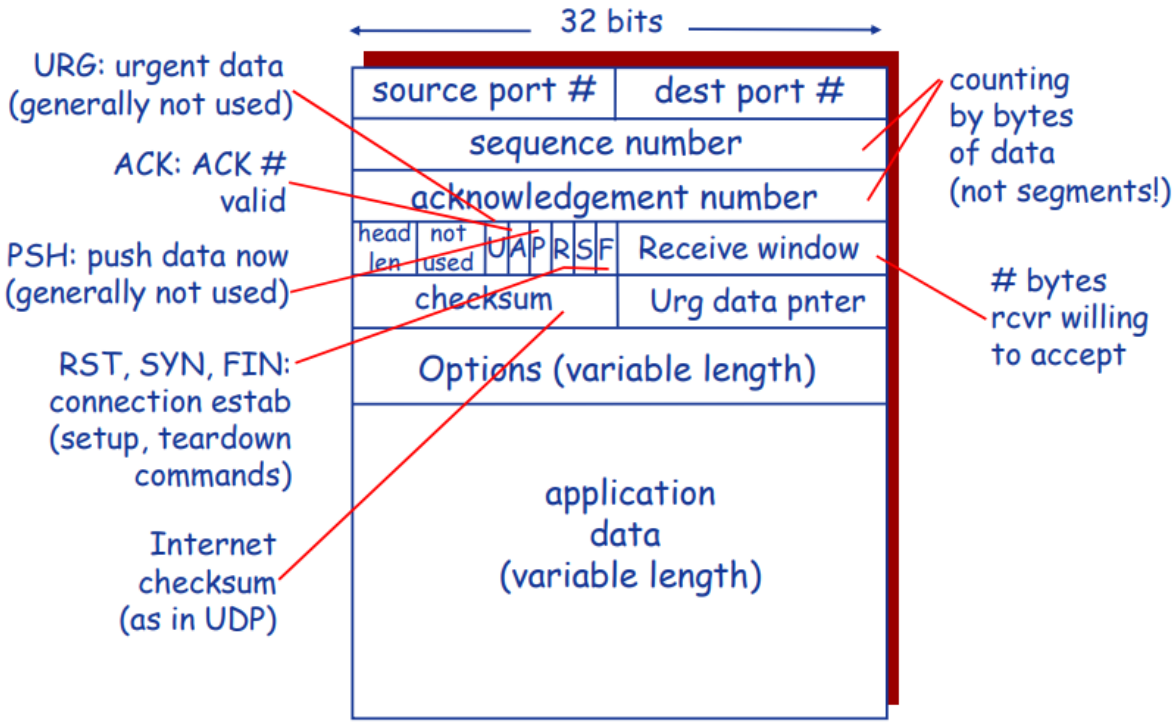
- 点对点
  - 一个发送方，一个接收方
- 可靠的、按序的字节流
- 流水线机制
  - TCP拥塞控制机制和流量控制机制设置窗口
- 发送方/接受方缓存

- 全双工
  - 同一连接中能传输双向数据流
- 面向连接

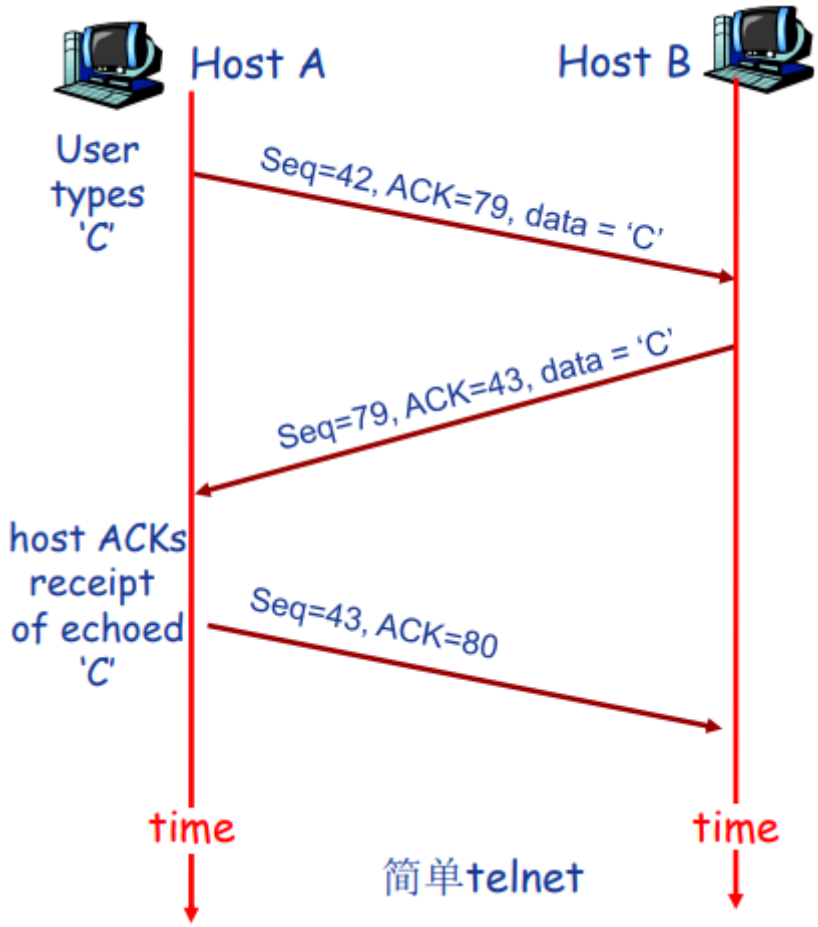
- 通信双方在发送数据之前必须建立连接
- 连接状态只在连接的两端中维护，在沿途结点不维护
- TCP连接包括：两台主机上的缓存、连接状态变量、socket等

- 流量控制机制

3.6.2 TCP段结构



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受
- 接收方如何处置未按序到达的Segment?
    - TCP规范中未明确规定，由TCP的实现者做出决策

3.6.3 TCP可靠数据传输

概述

- 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+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

RTT和超时

- 如何设置定时器的超时时间?
    - 大于RTT，但RTT是不断变化的，若设置的过短，会引起不必要的重传，若设置的过长，会导致 对段丢失时间反应慢
- 怎样估计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
    - $EstimatedRTT = (1-\alpha) * EstimatedRTT + \alpha * sampleRTT$
        - 指数加权移动平均，α的典型值为0.125
- 怎样设置定时器?
    - 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
        - $DevRTT = (1-\beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT|$
            - β的典型值为0.25
    - 定时器超时时间的设置
        - $TimeoutInterval = EstimatedRTT + 4 * DevRTT$

TCP发送方事件

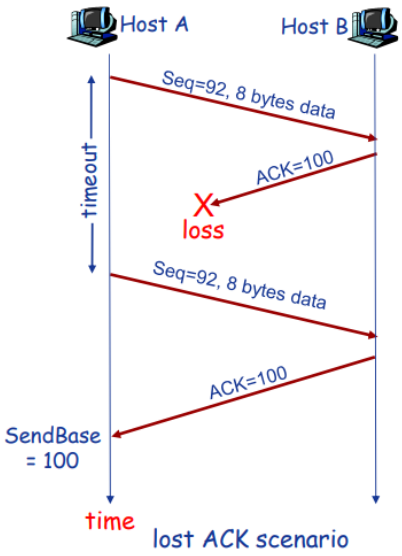
- 从应用层收到数据
  - 创建Segment
  - 序列号是Segment第一个字节的编号
  - 开启计时器
  - 设置超时时间：TimeOutInterval
- 超时
  - 重传引起超时的Segment
  - 重启定时器
- 收到ACK
  - 如果确认此前未确认的Segment
    - 更新SendBase
    - 如果窗口中还有未被确认的分组，重新启动定时器

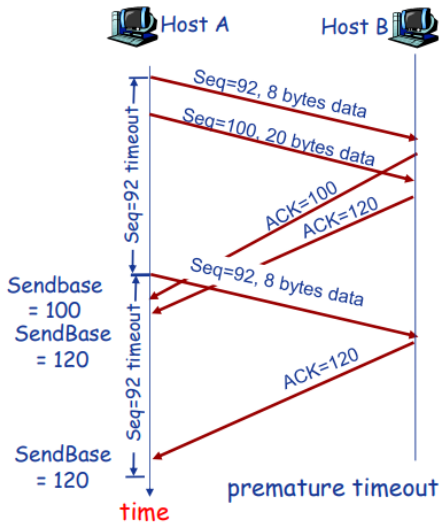
TCP发送端程序：

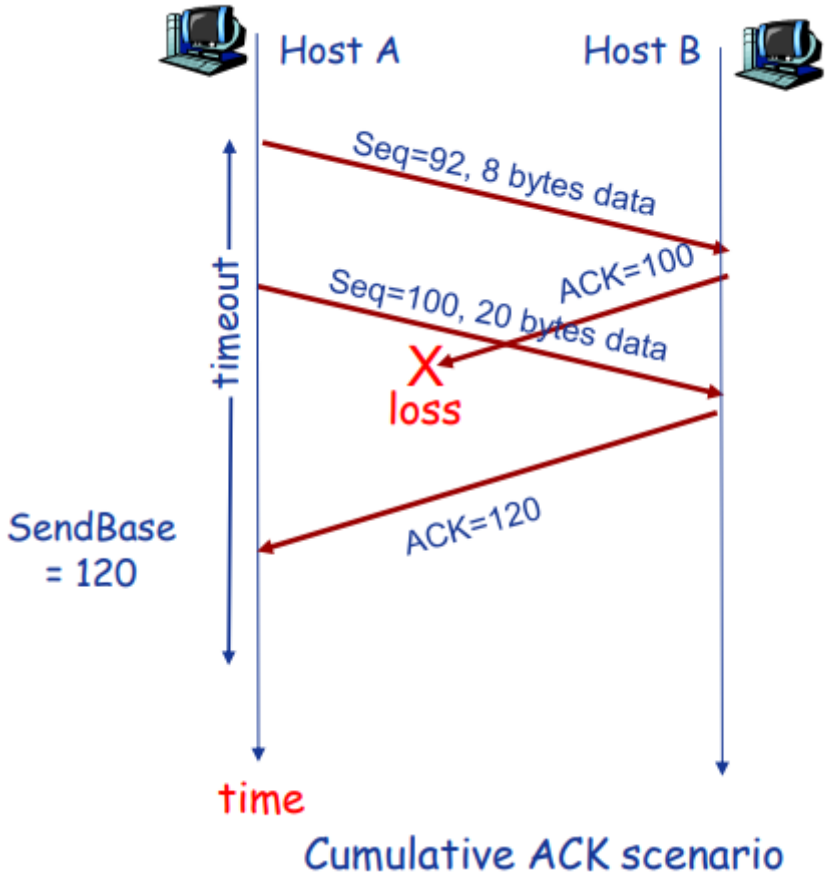
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
loop (forever) {
    switch(event)

    event: data received from application above
        create TCP segment with sequence number NextSeqNum
        if (timer currently not running)
            start timer
        pass segment to IP
        NextSeqNum = NextSeqNum + length(data)
    event: timer timeout
        retransmit not-yet-acknowledged segment with
            smallest sequence number
        start timer
    event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase = y
            if (there are currently not-yet-acknowledged segments)
                start timer
        }
} /* end of loop forever */
```

TCP重传示例







TCP ACK生成: RFC 1122, RFC 2581

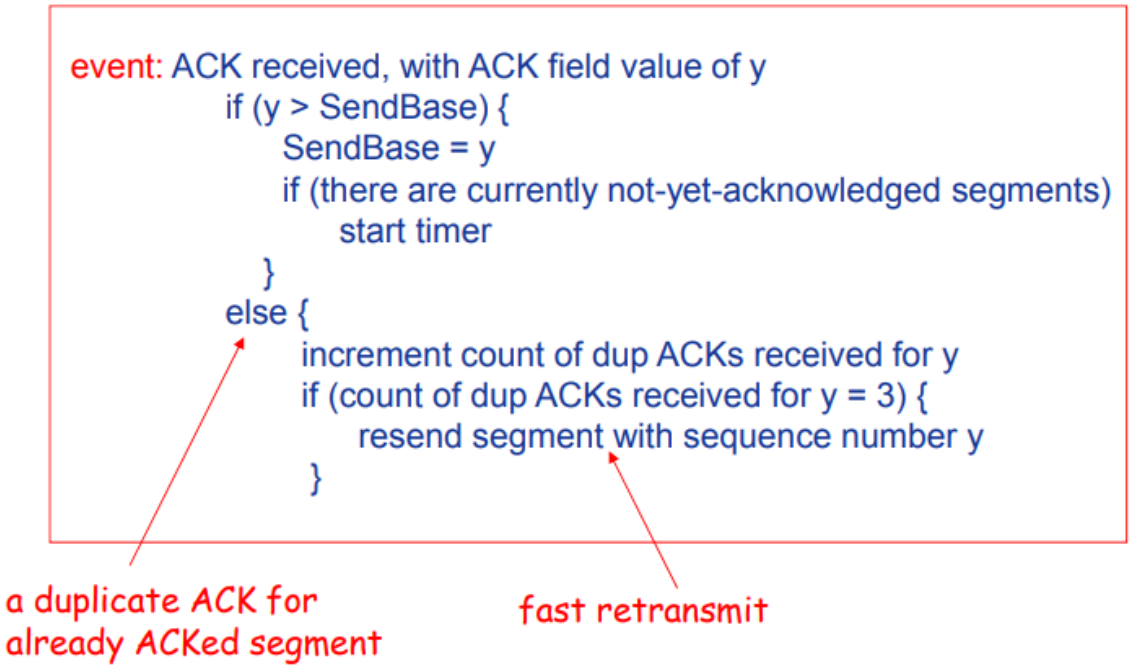
Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately 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

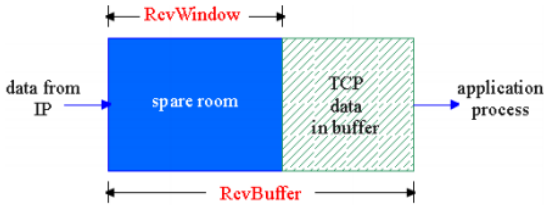
- 通过重复ACK检测分组丢失
  - Sender会背靠背地发送多个分组
  - 如果某个分组丢失，可能会引发多个重复的ACK
  - 如果sender收到对同一数据的3个ACK，则假定该数据之后的段已经丢失
    - 快速重传：在定时器超时之前即进行重传

快速重传算法

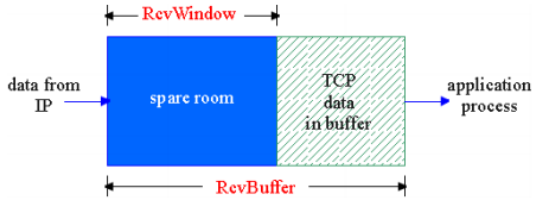


3.6.4 TCP流量控制

- 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



(假定TCP receiver丢弃乱序的segments)

- Buffer中的可用空间(spare room)

```
= RcvWindow
= RcvBuffer-[LastByteRcvd -
  LastByteRead]
```

flow control

发送方不会传输的太多、太快以至于淹没接收方

（buffer溢出）

- 速度匹配机制

- 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Sender **RcvWindow=0，会出现什么情况？**

为避免死锁，发送方会设置定时器，定期发送零窗口探测报文，直到接收方有空间为止

3.6.5 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();
```

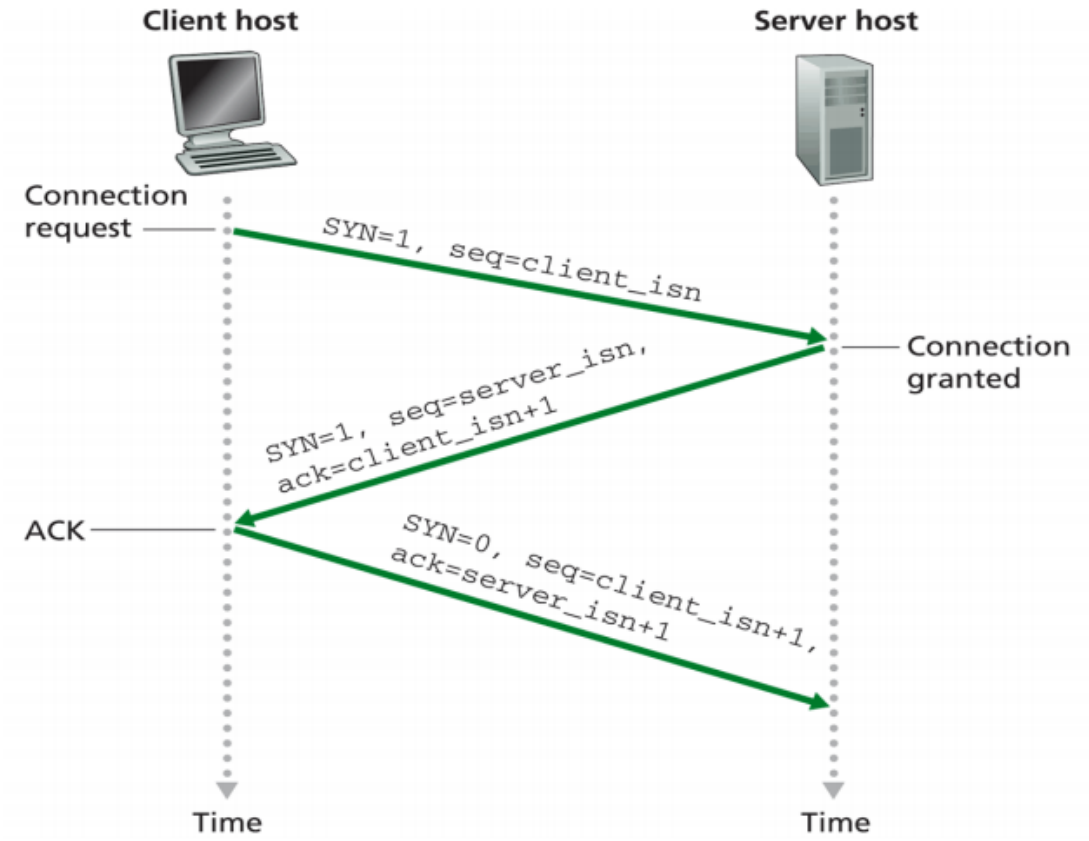
Three way handshake:

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

连接建立



为什么要三次握手?

为了防止已经失效的连接请求报文段突然又传到服务端，因而产生错误”，这种情况是：

一端(client)A发出去的第一个连接请求报文并没有丢失，而是因为某些未知的原因在某个网络节点上发生滞留，导致延迟到连接释放以后的某个时间才到达另一端(server)B。本来这是一个早已失效的报文段，但是B收到此失效的报文之后，会误认为是A再次发出的一个新的连接请求，于是B端就向A又发出确认报文，表示同意建立连接。如果不采用“三次握手”，那么只要B端发出确认报文就会认为新的连接已经建立了，但是A端并没有发出建立连接的请求，因此不会去向B端发送数据，B端没有收到数据就会一直等待，这样B端就会白白浪费掉很多资源。如果采用“三次握手”的话就不会出现这种情况，B端收到一个过时失效的报文段之后，向A端发出确认，此时A并没有要求建立连接，所以就不会向B端发送确认，这个时候B端也能够知道连接没有建立。

连接关闭

Closing a connection:

client closes socket: `clientSocket.close();`

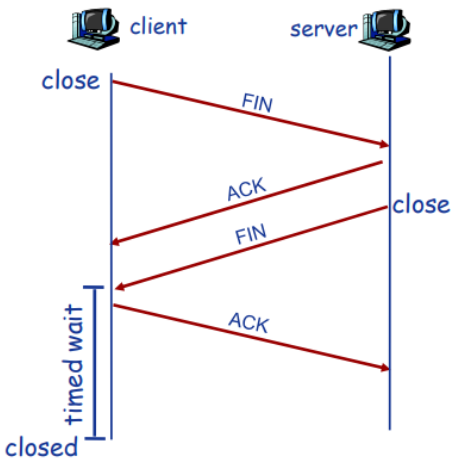
Step 1: client向server发送TCP FIN 控制segment

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.

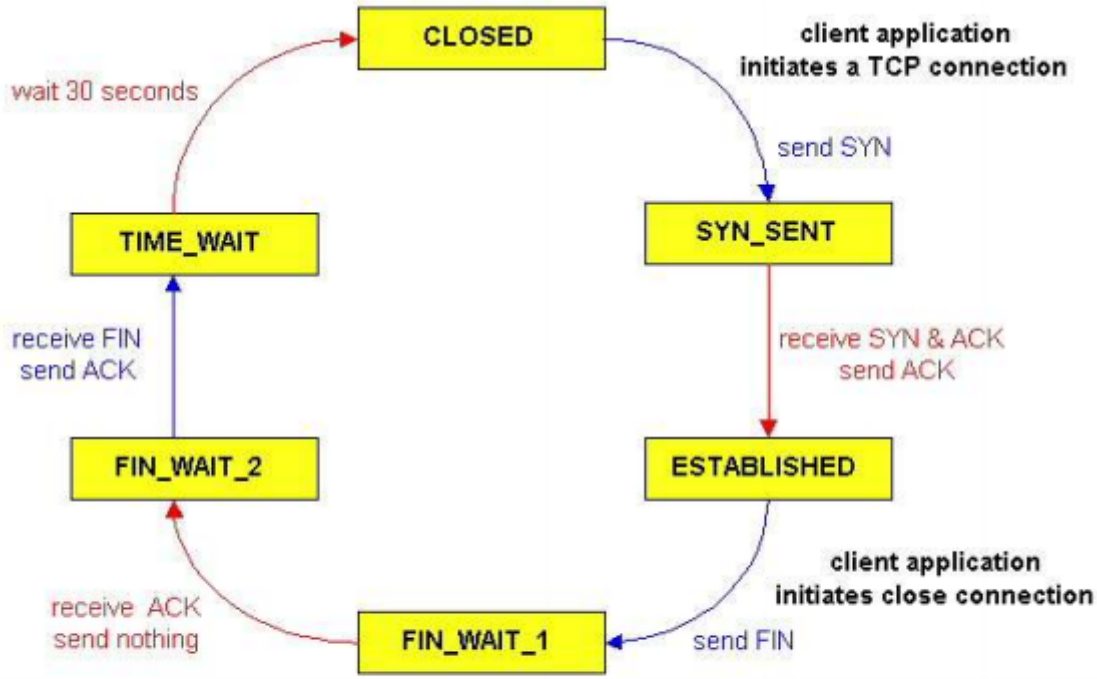
- 进入“等待” –如果收到FIN，会重新发送ACK

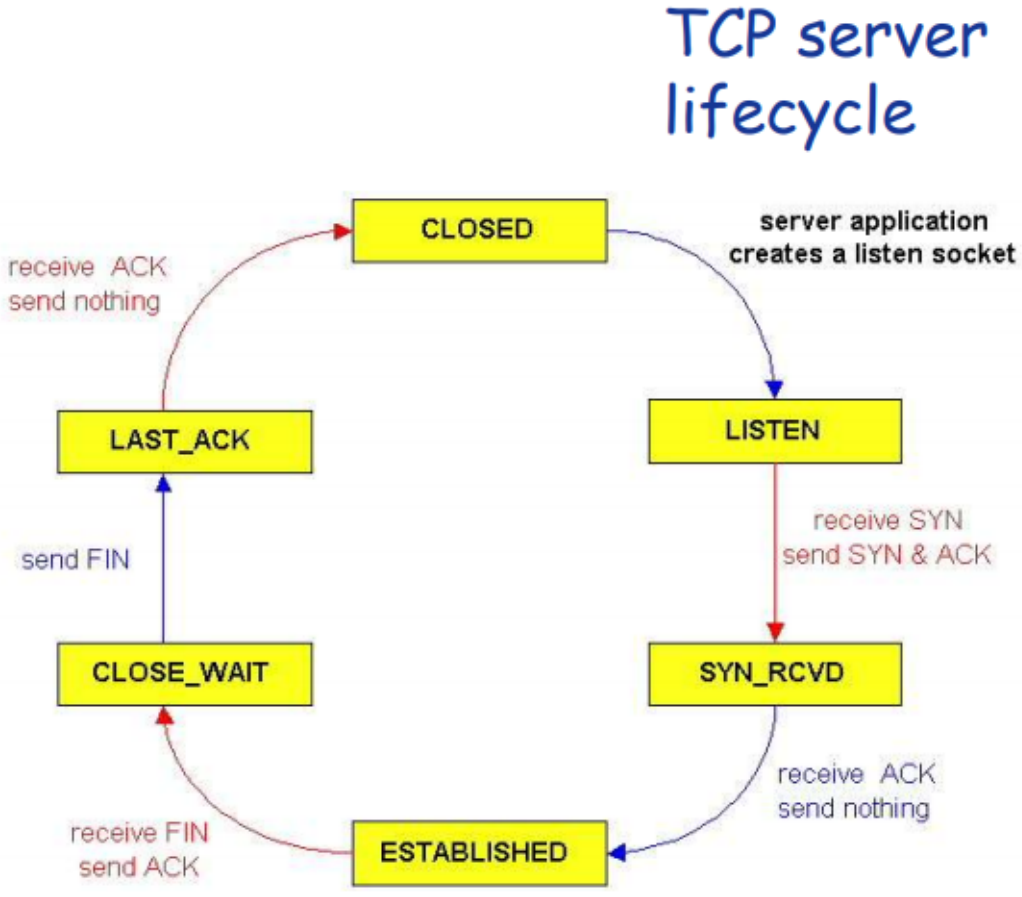
Step 4: server收到ACK. 连接关闭.



四次挥手可以简化为三次挥手，即服务器回复ACK时捎带发送FIN

TCP客户端、服务器的生命周期





3.6.6 拥塞控制原理

拥塞

非正式定义：“太多发送主机发送了太多数据或者发送速度太快，以至于网络无法处理”

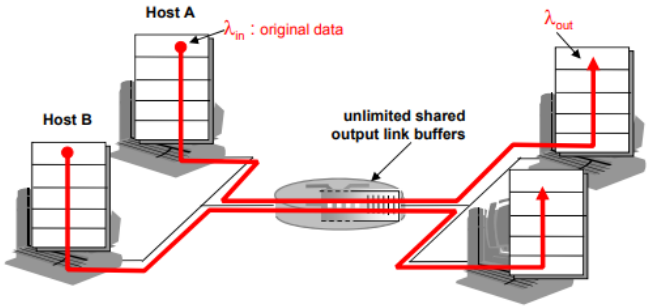
表现

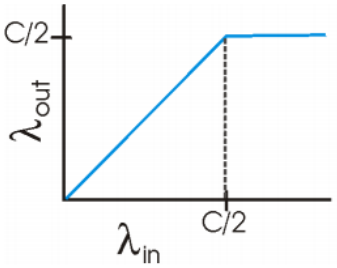
- 分组丢失（路由器缓存溢出）
- 分组延迟过大（在路由器缓存中排队）

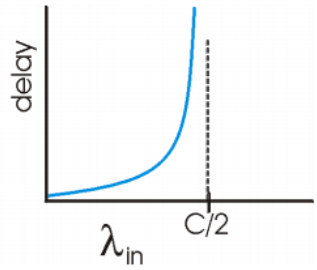
拥塞的成因和代价

场景1

- 两个senders,两个receivers
- 一个路由器, 无限缓存
- 没有重传



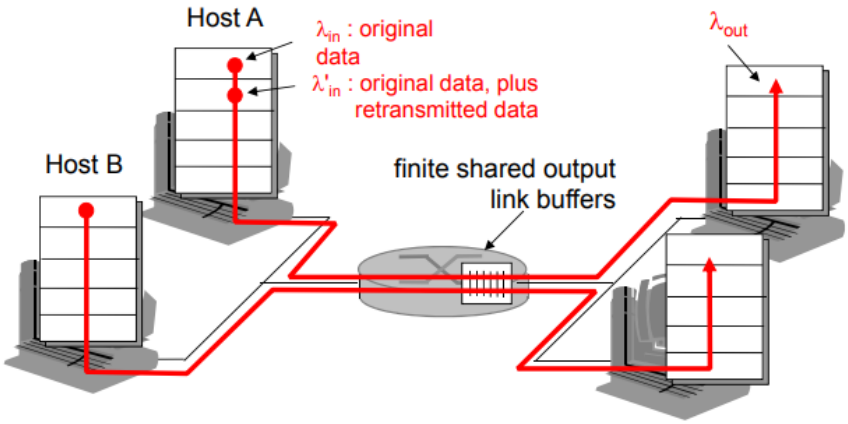




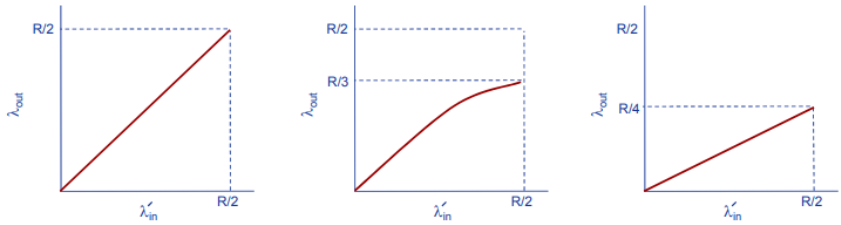
- 拥塞时分组延迟太大
- 达到最大throughput

场景2

- 一个路由器, 有限buffers
- Sender重传分组



- 情况a: Sender能够通过某种机制获知路由器buffer信息，有空闲才发 $\lambda_{in} = \lambda_{out}$ (goodput)
- 情况b: 丢失后才重发: $\lambda'_{in} > \lambda_{out}$
- 情况c：分组丢失和定时器超时后都重发，λ'_{in} 变得更大



拥塞的代价:

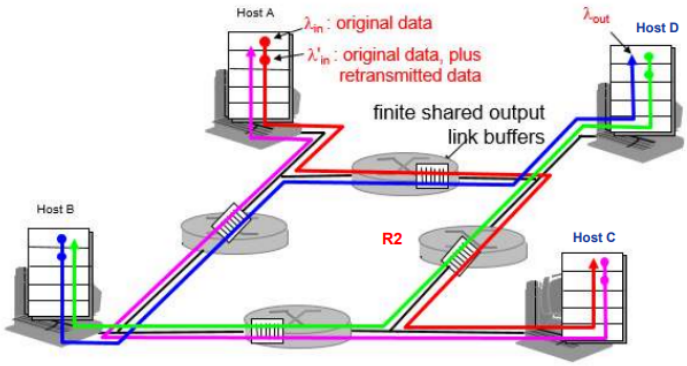
- 对给定的"goodput"，要做更多的工作 (重传)
- 造成资源的浪费

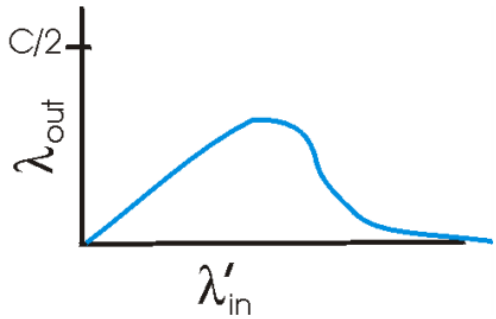
场景3

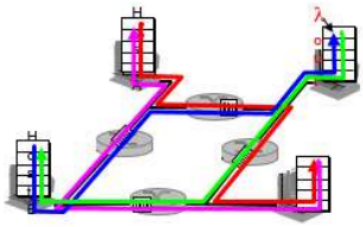
- 四个发送方
- 多跳
- 超时/重传

Q: 随着 λ_{in} 和 λ'_{in} 不断增加，会怎么样？

由于竞争会变得很低，分组不断消失







拥塞的另一个代价:

- 当分组被drop时，任何用于该分组的“上游”传输能力全都被浪费掉

拥塞控制的方法

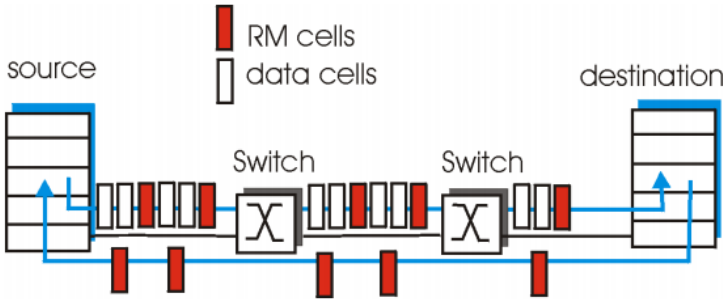
- 端到端拥塞控制：
  - 网络层不需要显式的提供支持
  - 端系统通过观察loss，delay等网络行为判断是否发生拥塞
  - TCP采取这种方法

- 网络辅助的拥塞控制：
  - 路由器向发送方显式地反馈网络拥塞信息
  - 简单的拥塞指示(1bit)：SNA, DECbit, TCP/IP ECN, ATM)
  - 指示发送方应该采取何种速率

实例：ATM ABR拥塞控制

- ABR：available bit rate
  - “弹性服务”
  - 如果发送方路径“underloaded”
    - 使用可用带宽
  - 如果发送方路径拥塞
    - 将发送速率降到最低保障速率

- RM(resource management) cells
  - 发送方发送
  - 交换机设置RM cell位(网络辅助)
    - NI bit: rate不许增长
    - CI bit: 拥塞指示
  - RM cell由接收方返回给发送方



- 在RM cell中有显式的速率(ER)字段：两个字节
  - 拥塞的交换机可以将ER置为更低的值
  - 发送方获知路径所能支持的最小速率
- 数据cell中的EFCI位: 拥塞的交换机将其设为1
  - 如果RM cell前面的data cell的EFCI位被设为1，那么发送方在返回的RM cell中置CI位

3.6.7 TCP拥塞控制

基本原理

- Sender限制发送速率

```
LastByteSent-LastByteAcked
                <= CongWin
```

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

- 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

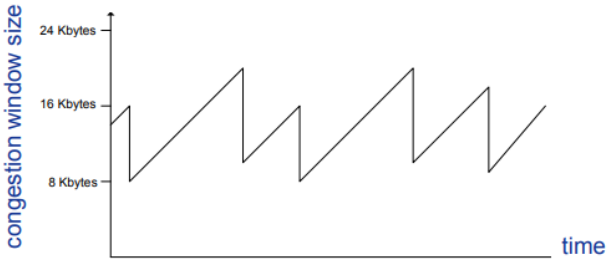
如何合理地调整发送速率？

- 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

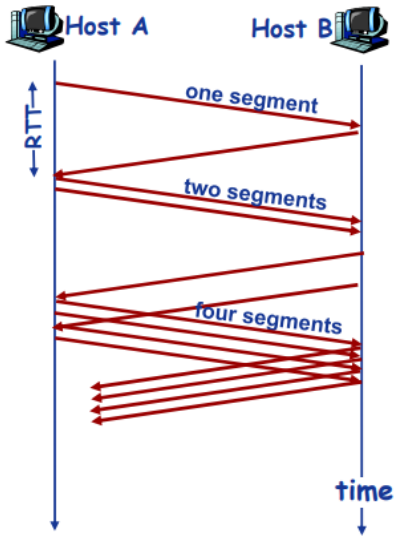
- 原理：
  - 当连接开始时，指数性增长

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

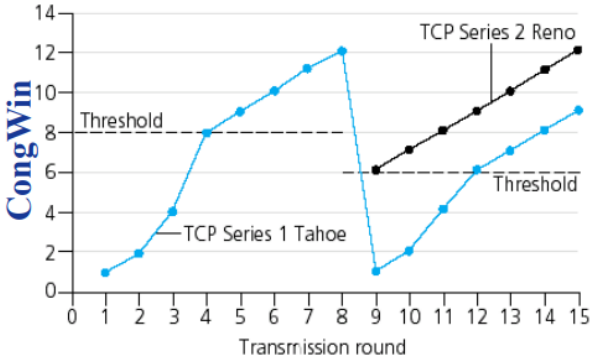
- 初始速率很慢，但是快速攀升



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:

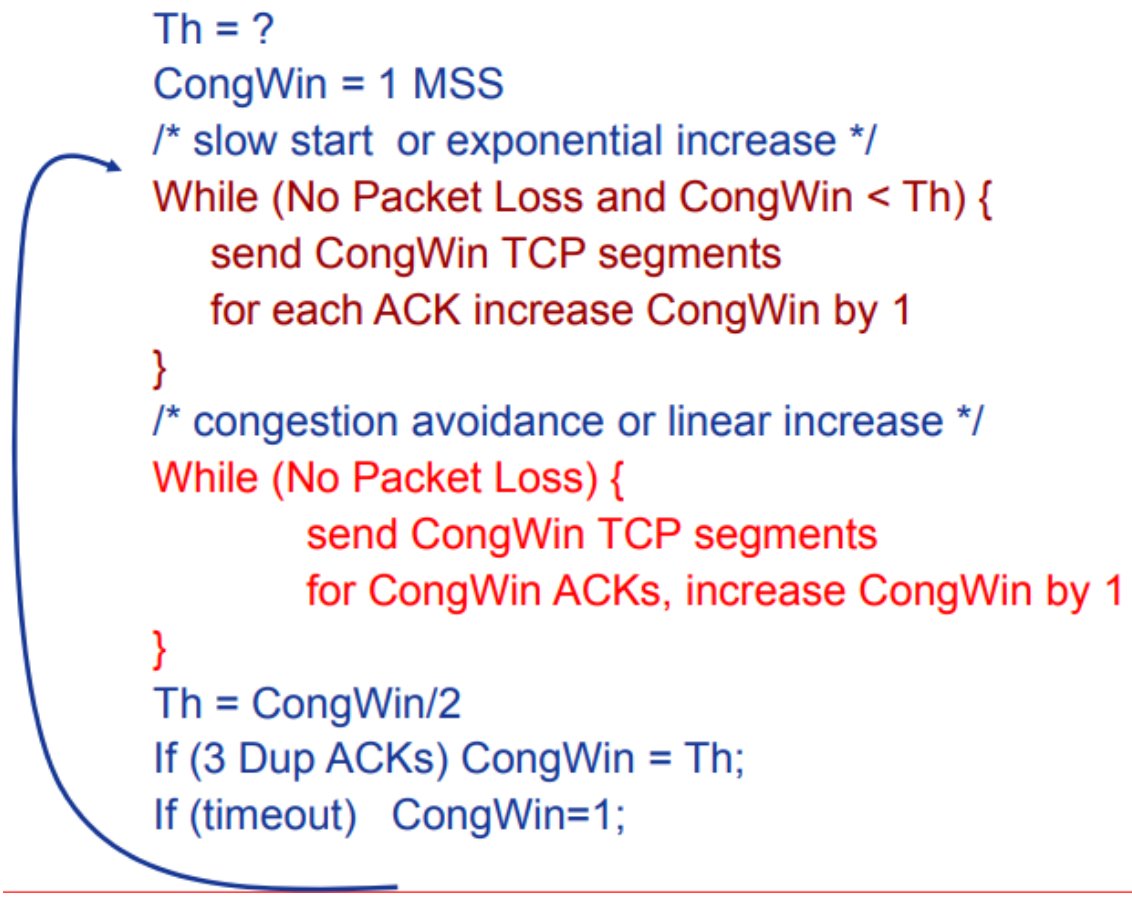
- 3个重复ACKs表示网络还能够传输一些 segments
- timeout事件表明拥塞更为严重

TCP拥塞控制：总结

- 拥塞窗口(CongWin)大小**低于**门限值(Threshold)时，发送方处于**慢启动阶段**，拥塞窗口**指数型增长**
- 拥塞窗口大小**高于**门限值时，发送方处于**拥塞避免阶段**，拥塞窗口**线性增长**
- 发送方收到**三个重复ACK时**，门限值和拥塞窗口**都减为原来的一半**
- 发送方发生**超时事件**时，**门限值减为原来的一般，拥塞窗口直接减为1**

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



3.6.8 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

$W/2+(W/2+1)+(W/2+2)+\ldots+W = 3W^2/8+3W/4$

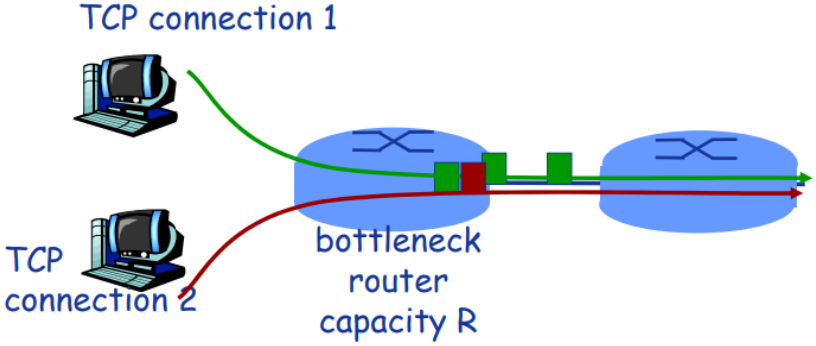
  - W很大时，$3W^2/8>>3W/4$，因此$L \approx 8/(3W^2)$

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

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

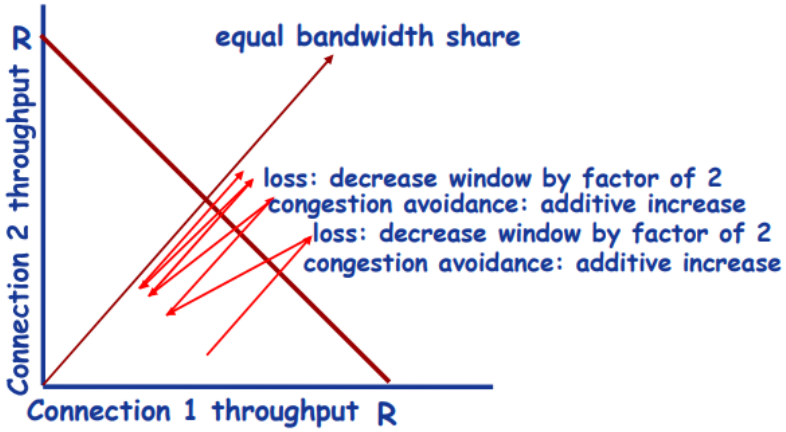
TCP的公平性

- 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



TCP具有公平性吗?

- 是的



TCP公平性的讨论

- 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