计算机网络

Computer Networks

课程QQ群

教师/助教信息

- 主讲：华蓓
  - 办公室：科技实验楼西楼617室
  - 电子邮件: bhua@ustc.edu.cn
  - 主页：http://staff.ustc.edu.cn/~bhua
- 助教：
  - 宋骐老师：qisong09@ustc.edu.cn
  - 贺若舟：fward@mail.ustc.edu.cn
  - 应宇峰：nbyyf2002@mail.ustc.edu.cn
  - 梁峻涛：ljt990113@mail.ustc.edu.cn
  - 梁奕涵：liangyh@mail.ustc.edu.cn
  - 孙若培：ruopeisun@mail.ustc.edu.cn

万物互联时代：网络无处不在

工作场所

家庭

路上

车站

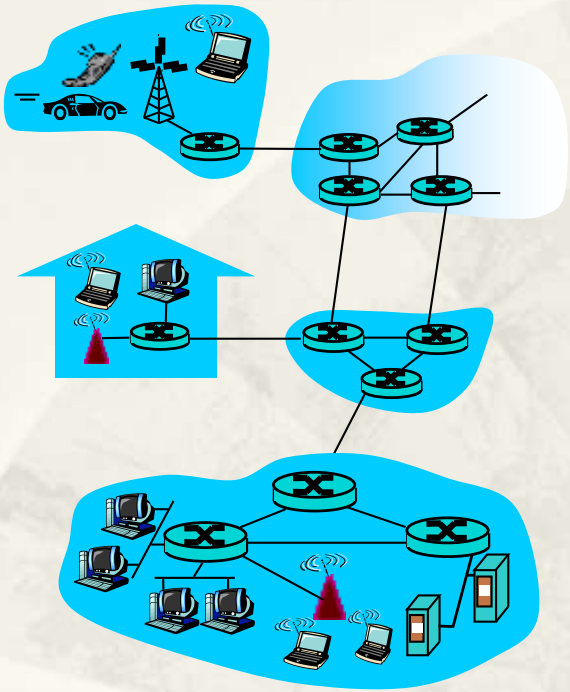
物联网：万物相连的互联网

普通用户看到的网络

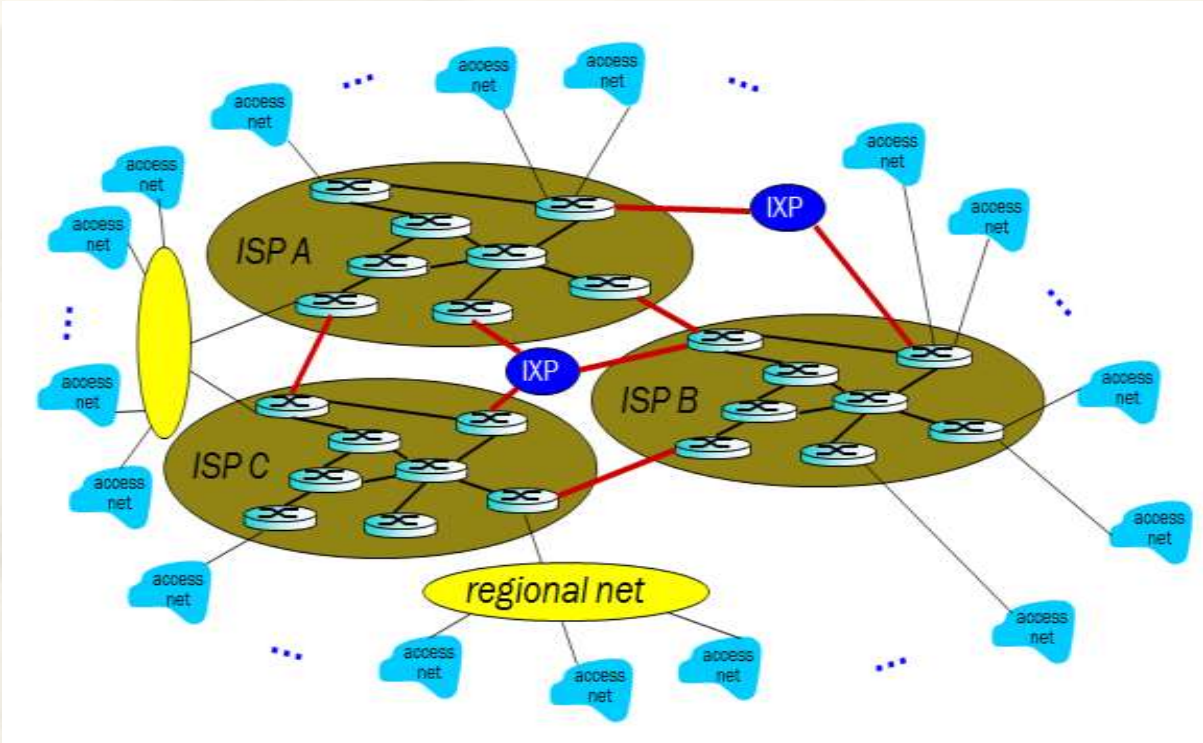
* 网络应用

* 上网设备

本课程要了解的网络



因特网 = 终端+接入网+网络核心



因特网的网络核心

教学目标

- 掌握计算机网络的基本概念、工作原理、体系结构、重要协议（以因特网为实例）
- 了解典型网络设备的工作原理
- 能够运用计算机网络的知识分析和解决一些实际问题

教材与参考书

- 三大经典教材：
  - Kurose & Ross, Computer Networking: A Top-Down Approach (7th Edition). （本课程教材）
  - Tanenbaum, Computer Networks.
  - Peterson & Davie, Computer Networks: A Systems Approach.
- 需要考研的同学，可以参考以下教材：
  - Tanenbaum, Computer Networks
  - 谢希仁，计算机网络

教材的使用

- 复习题：
  - 课后自己复习
- 习题：
  - 选出一些作为作业
  - 其余建议自选练习，提倡相互讨论，也可以问助教
  - 提问较为集中的题目，将安排在习题课上讲
- 编程作业：
  - 不做（下学期有“网络系统实验”课）
- 观察实验：
  - 本课程的实验内容

关于实验

- 8个观察实验：
  - 利用Wireshark从网络中抓包，按照实验指导书的要求观察包结构并回答问题
  - Wireshark是最流行的包分析工具，是进行网络流量分析、入侵检测、网络管理等的必备工具
- 实验目的：
  - 通过观察数据包加深对协议的理解
  - 学会使用Wireshark及一些常用的网络工具，如nslookup、ipconfig、arp、ping、tracert等

课程成绩

- 作业：30%
- 实验：20%
- 期中考试：25%（平行班统一试卷）
- 期末考试：25%（平行班统一试卷）
- 作业和实验报告必须按时交，抄袭不计成绩，迟交成绩减半（有事提前说明）
- 实验缺交1/3以上的处理：
  - 本课程的实验成绩为0，学分为0
  - 在补交实验之前，不能参加补考

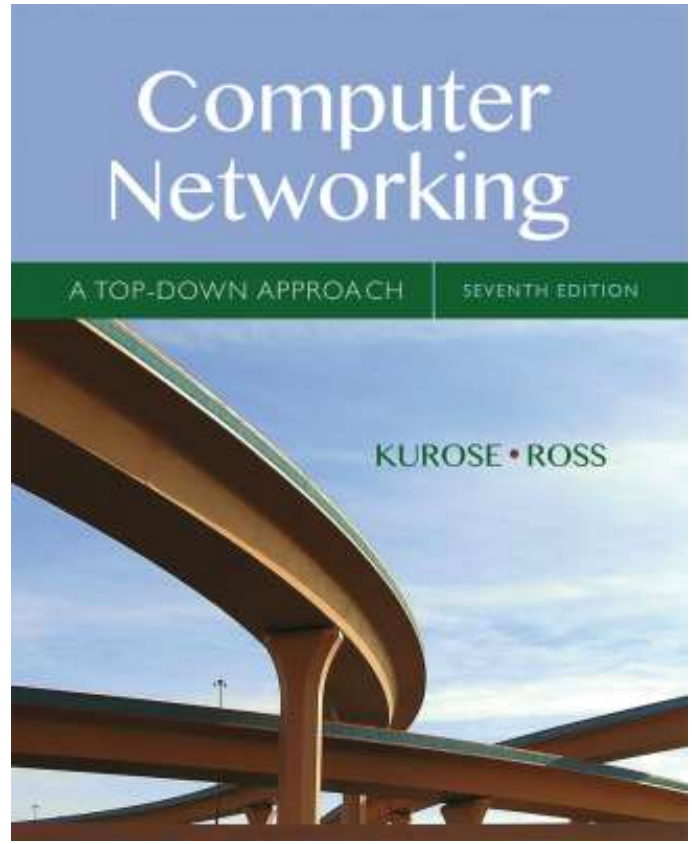
后续相关课程

- 网络系统实验（三下）
  - 使用C++语言编程实现一个真实可用的TCP协议（参照斯坦福大学CS144课程实验进行设计）
- 网络算法学（四上）
  - 关注网络系统的高效实现
- 信息安全导论（三下）
  - 针对信息安全较全面的介绍

几点提醒

- 学习网络和使用网络是两回事：
  - 使用网络是愉快的，学习网络可能是枯燥的
- 要有正确的学习方法：
  - 网络课涉及的知识点很多，要经常梳理知识点，建立知识点之间的联系，变成自己的知识
  - 协议虽然是一系列的规定，但切忌死记硬背，要理解为什么这么设计，并能灵活运用
- 学习知识固然重要，但是学会方法更重要！
- 亲自做实验，实践很重要

Chapter 1 Introduction



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Chapter 1: Introduction

Our goal:

- get "feel" and terminology
- more depth, detail *later* in course
- approach:
  - use Internet as example

Chapter 1: roadmap

1.1 What *is* the Internet?

1.2 Network edge

- end systems, access networks, links

1.3 Network core

- circuit switching, packet switching, network structure

1.4 Delay, loss and throughput in packet-switched networks

1.5 Protocol layers, service models

1.6 Networks under attack: security

1.7 History

1.1 What is the Internet?

- 将从以下两个角度描述因特网是什么:
  - 因特网的具体构成
  - 因特网的功能

1.1.1 因特网的具体构成

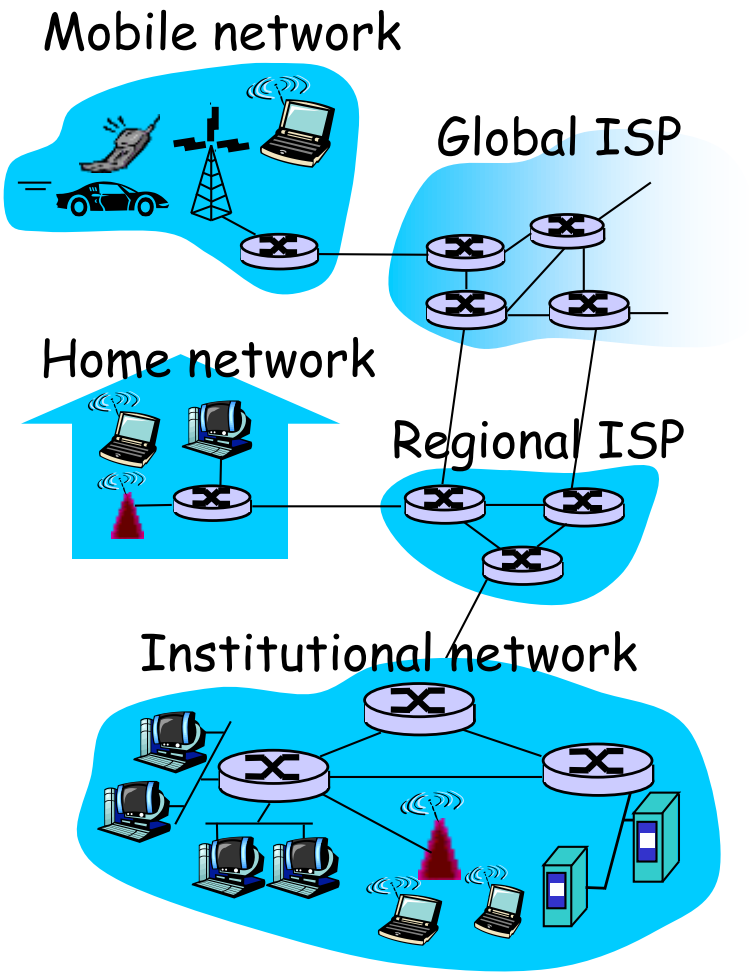
桌面机

服务器

笔记本

手持设备

- 终端：
  - 也称主机（host）或端系统（ end system）
  - 运行应用程序



- 通信链路：
  - 光纤，铜线，电磁波
  - 主要指标为传输速率，也称带宽(bandwidth)

无线链路

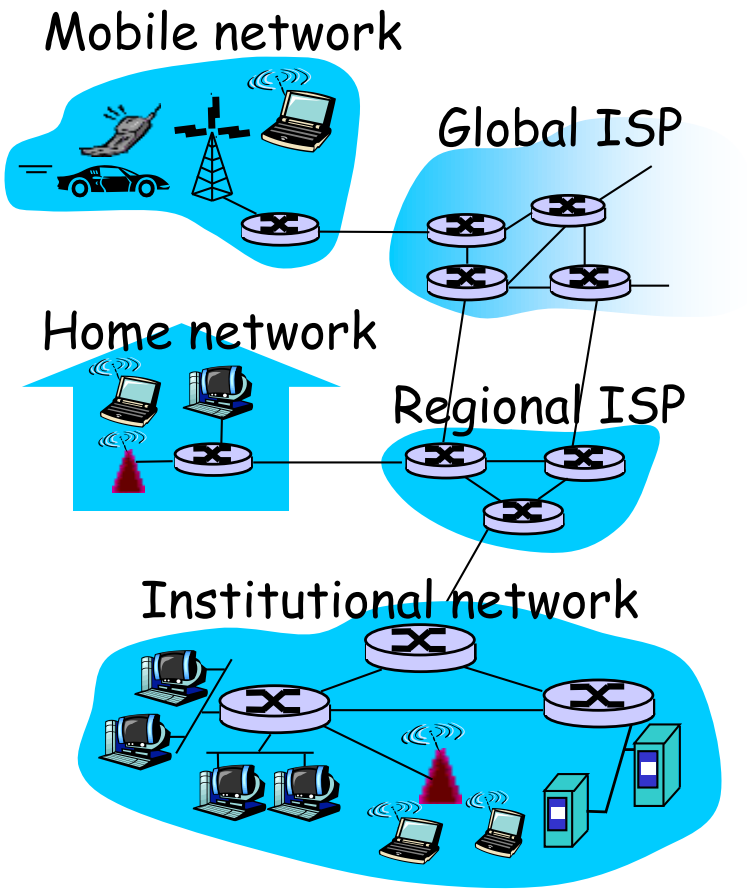
有线链路

- 交换设备：
  - 转发分组（packet）
  - 路由器和交换机

路由器

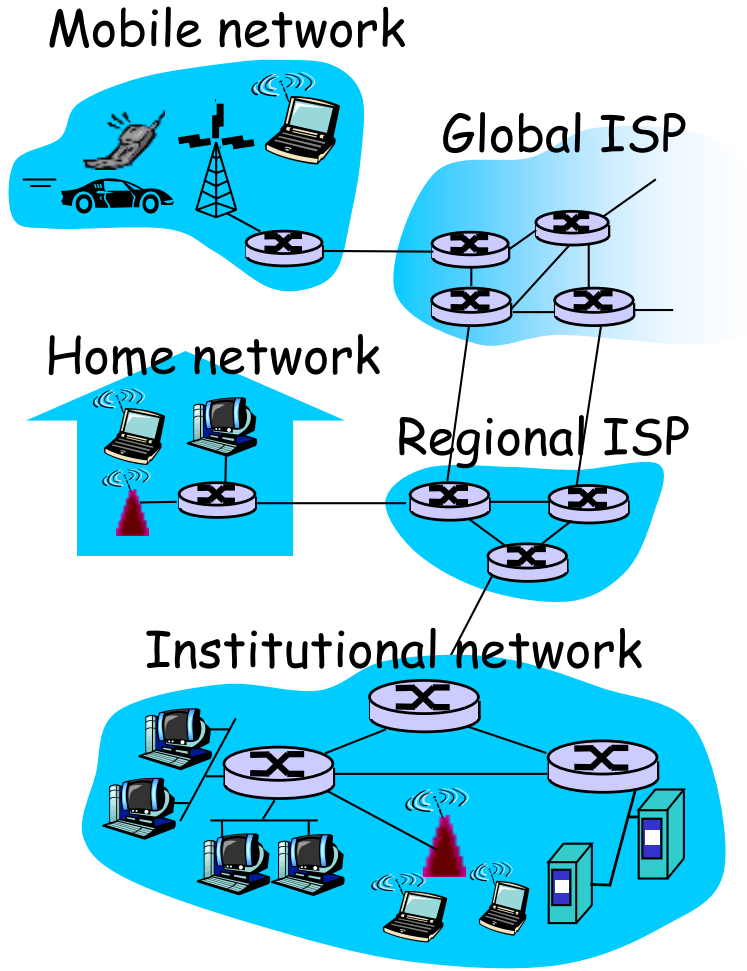
因特网的具体构成（续）

- Internet Service Provider：
  - ISP是由交换设备和通信链路组成的网络，为终端提供因特网接入服务
  - 不同层次的ISP：本地ISP，地区ISP，全球ISP
  - 每个ISP是自治的



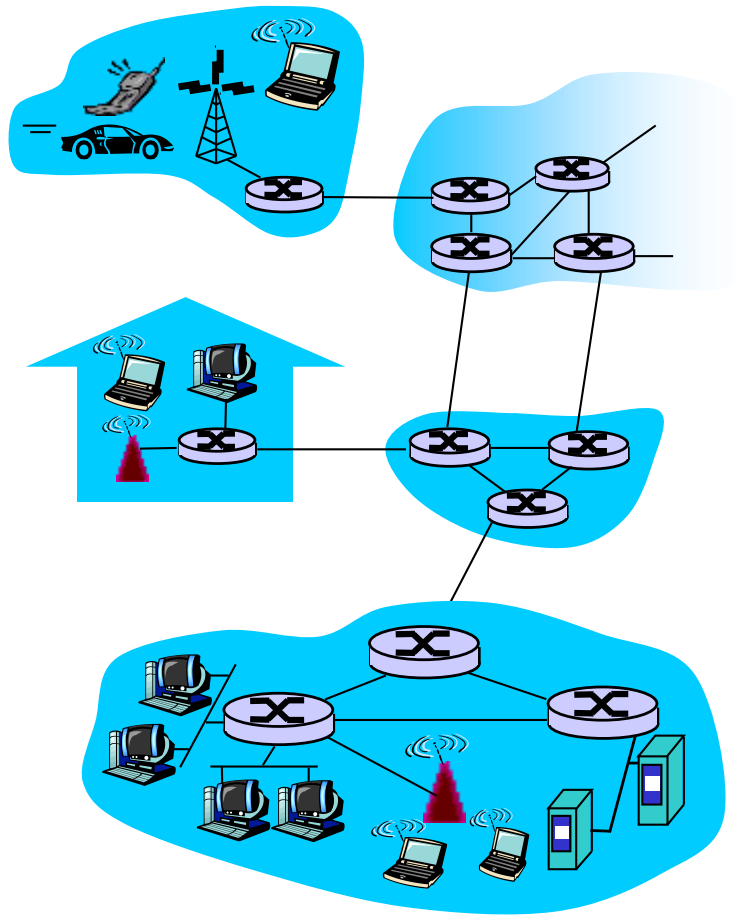
因特网的具体构成（续）

- 协议规定了设备之间通信需要遵循的规则：
  - 终端与终端之间
  - 终端与交换设备之间
  - 交换设备与交换设备之间
- 因特网协议标准：
  - 由IETF组织统一管理，以RFC xxx文档的形式发布
  - 因特网中最核心的两个协议是TCP和IP，因特网协议统称为TCP/IP协议族



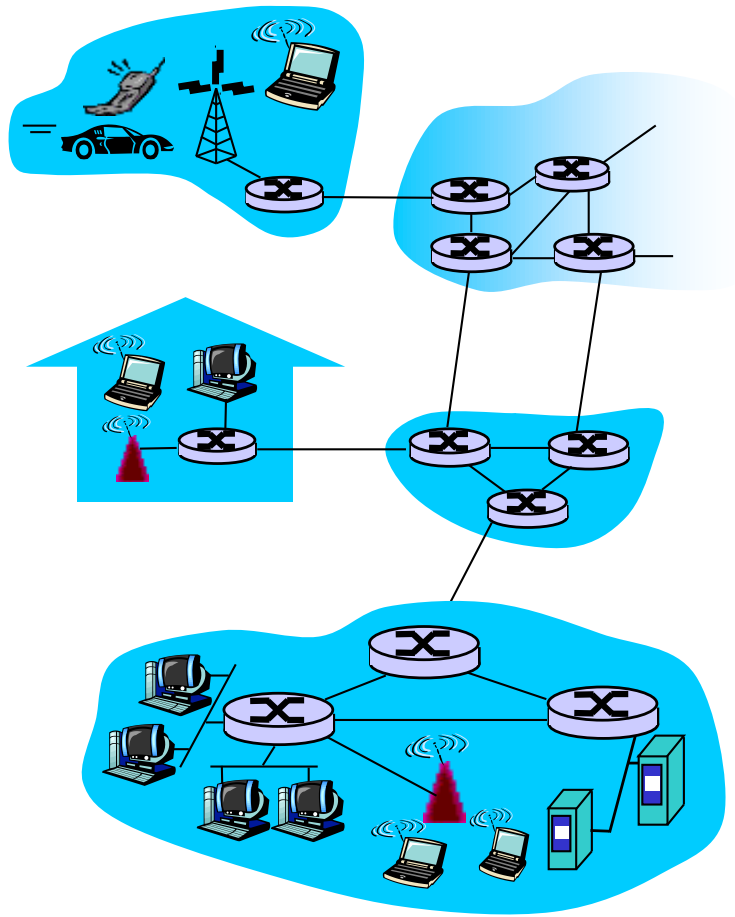
因特网的具体构成（续）

- 因特网定义一：
  - 由一群遵循TCP/IP协议的ISP，按照松散的层次结构组织而成的网络的网络
- 因特网的几个特点：
  - 因特网是“网络的网络”
  - 因特网不存在严格的层次结构
  - 因特网没有统一的管理机构



1.1.2 因特网的功能

- 因特网定义二：
  - 因特网是为分布式应用提供通信服务的基础设施
- 传统通信系统的服务接口：
  - 电话系统：拨号，振铃
  - 邮政系统：邮筒，信箱
- 因特网提供给应用程序的服务接口：
  - 一组用于在因特网上发送和接收数据的应用编程接口API



小结

- 因特网定义一：
  - 由一群遵循TCP/IP协议的ISP，按照松散的层次结构组织而成的网络的网络
- 对于通信功能的实现有指导作用：
  - ISP内部实现
  - ISP之间互联
- 因特网定义二：
  - 为分布式应用提供通信服务的基础设施
- 对于服务接口的定义有指导作用：
  - 有序、可靠的数据交付服务
  - 不可靠的数据交付服务

本课程使用这两种定义，介绍因特网服务接口及端到端通信的实现

Chapter 1: roadmap

1.1 What *is* the Internet?

1.2 Network edge

- end systems, access networks, links

1.3 Network core

- circuit switching, packet switching, network structure

1.4 Delay, loss and throughput in packet-switched networks

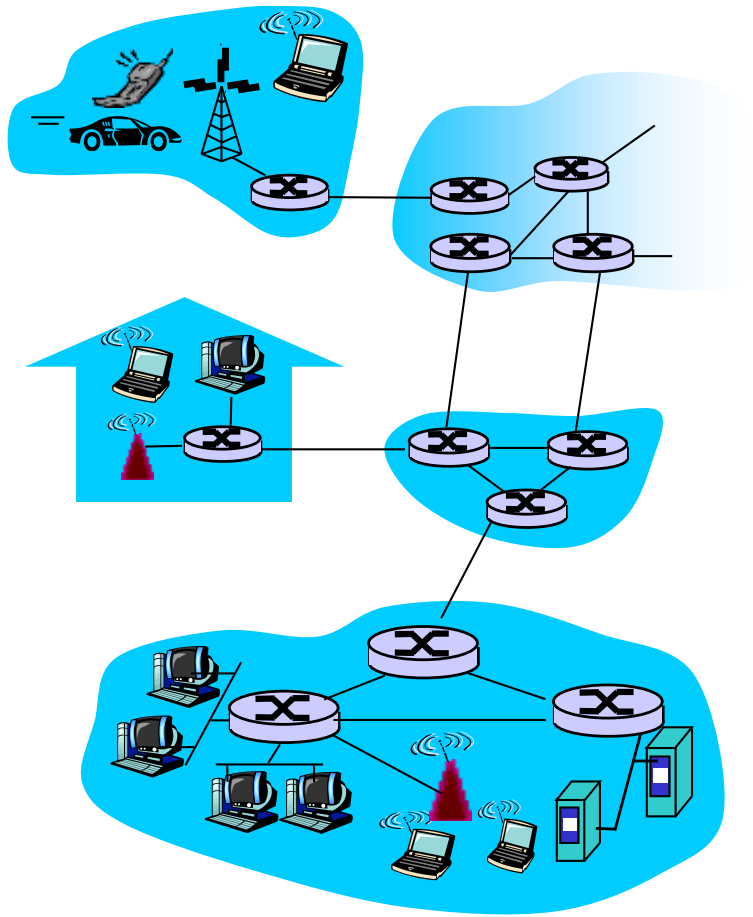
1.5 Protocol layers, service models

1.6 Networks under attack: security

1.7 History

A closer look at network structure:

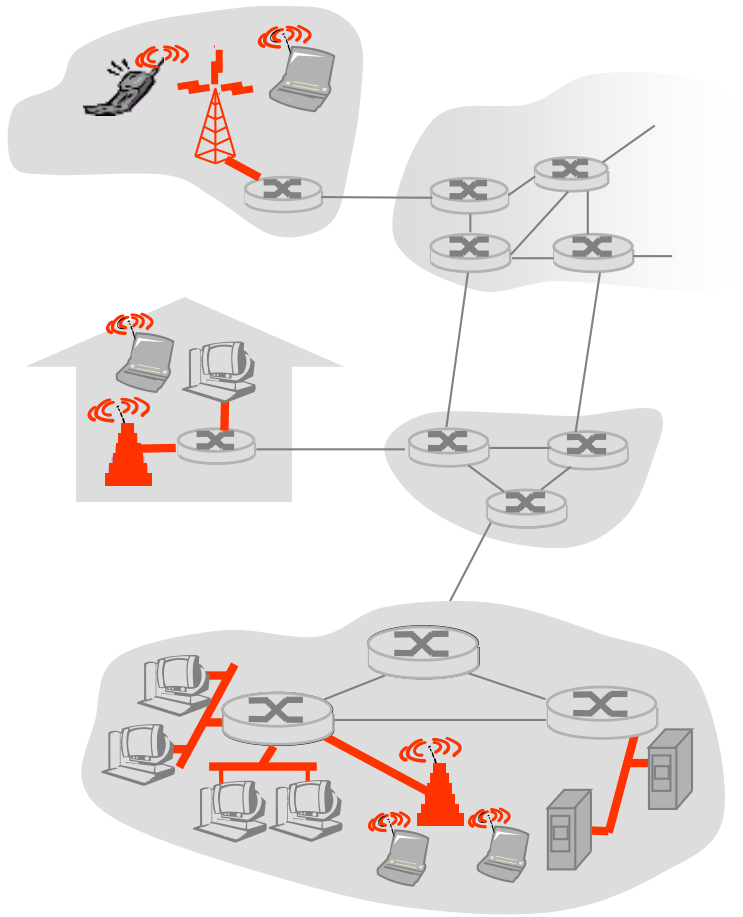
- **end systems:**
  终端
- **access networks:**
  将终端连接到其边缘路由器的物理链路
- **network core:**
  路由器和通信链路组成的网网络



1.2.1 接入网

Q: How to connect end systems to edge router?

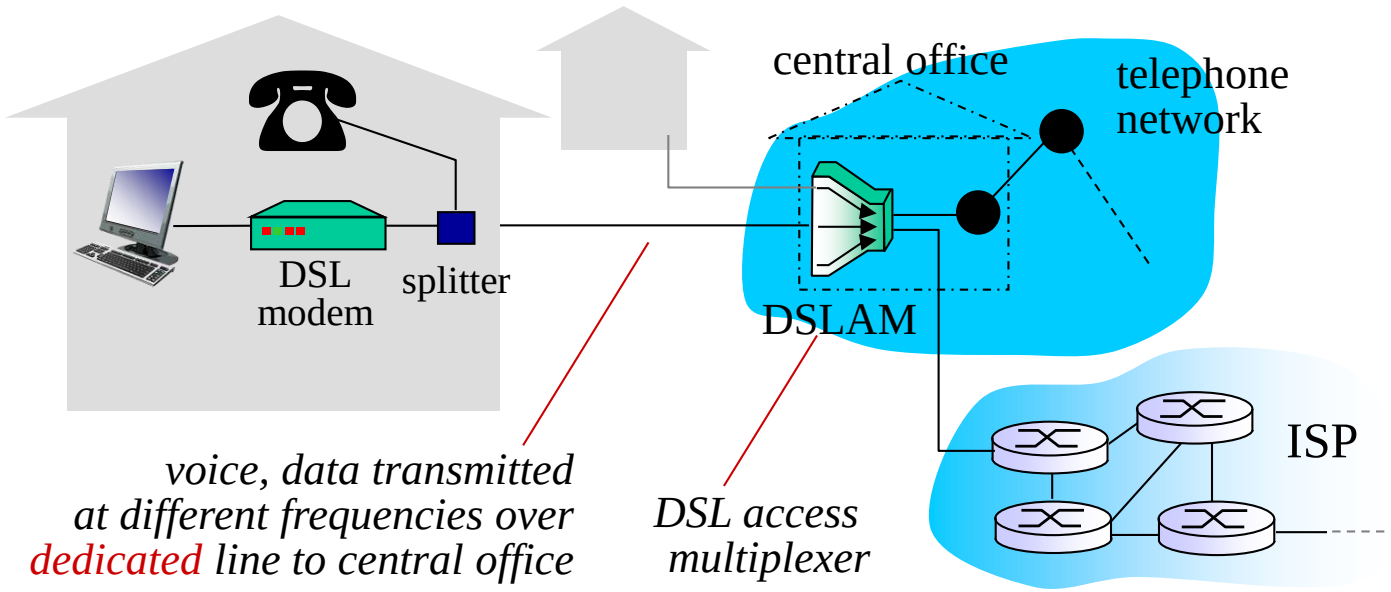
- 住宅接入
- 企业接入（学校，公司）
- 移动接入



Keep in mind:

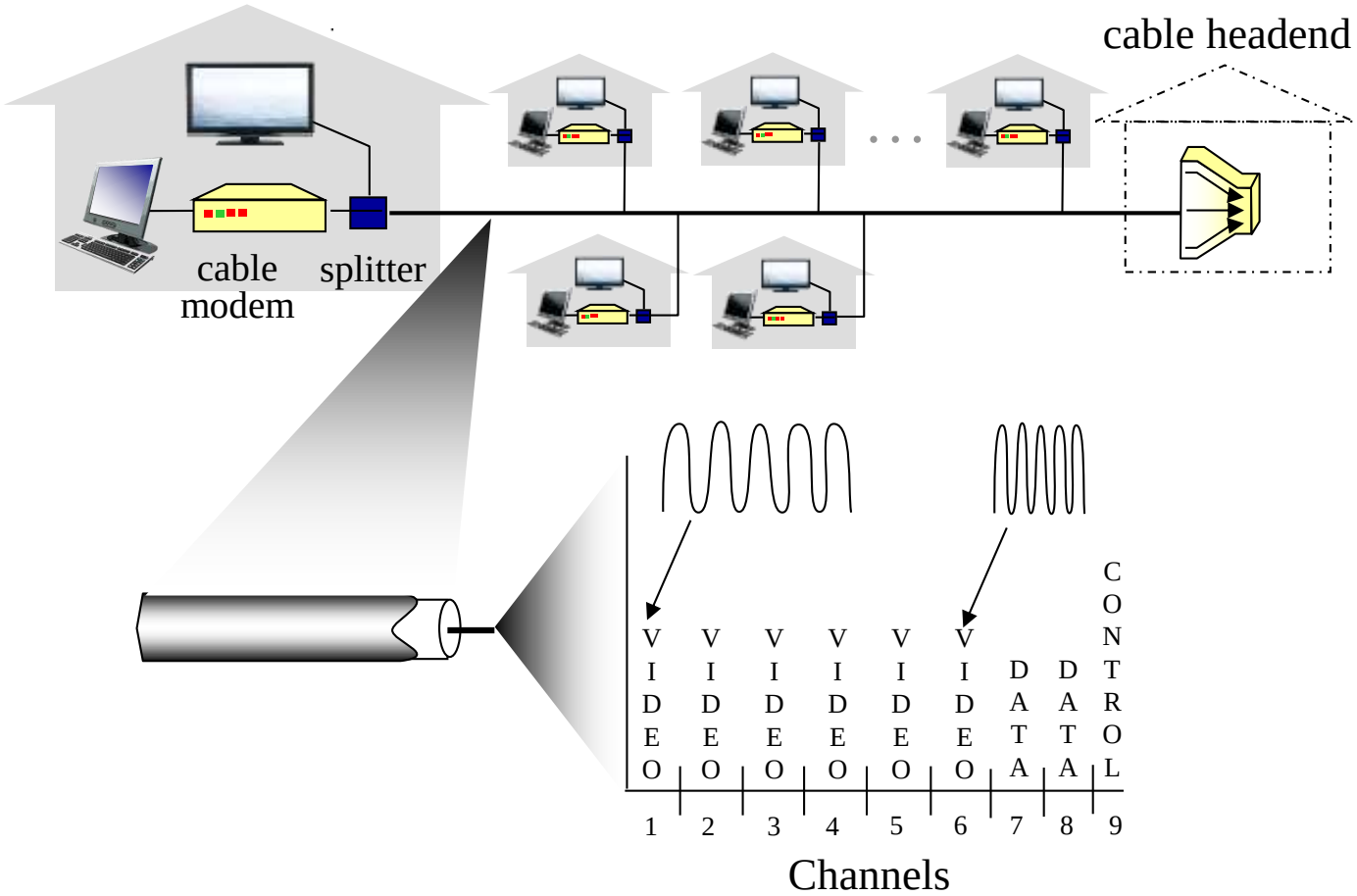
- 接入网的带宽是多少？
- 共享还是专用？

住宅接入：数字用户线（DSL）



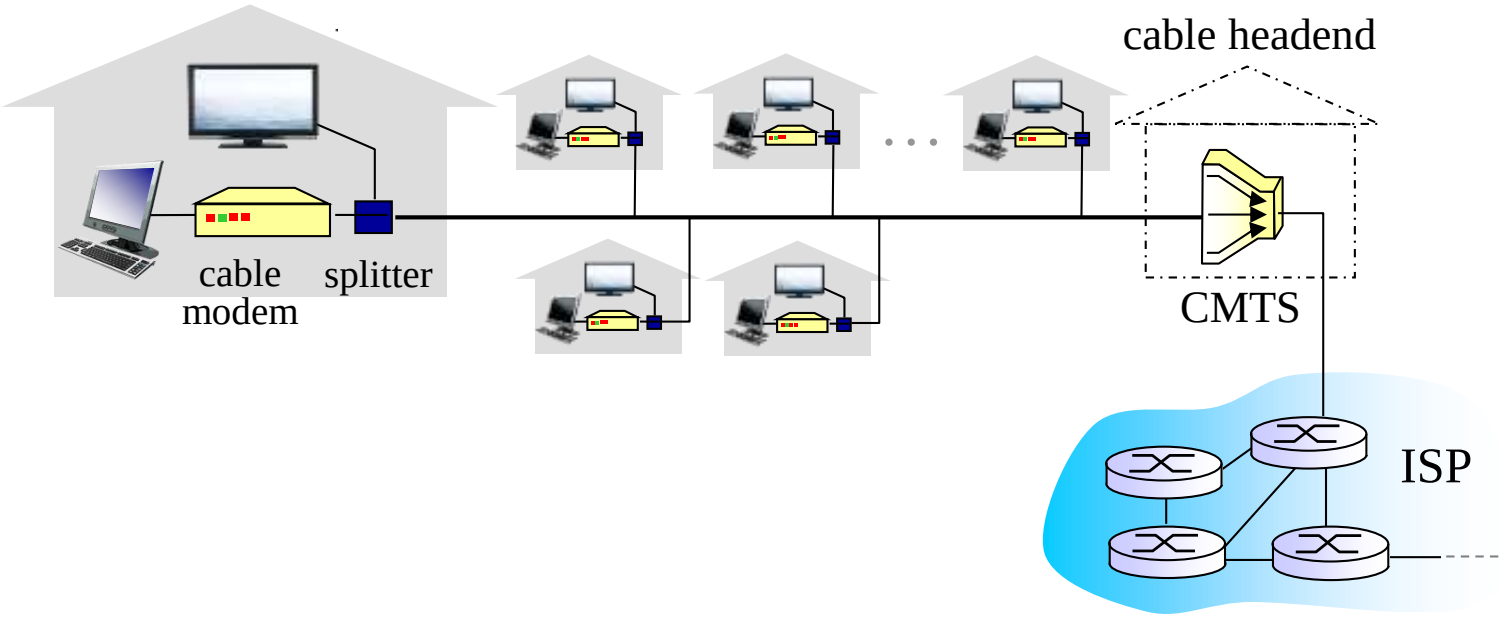
- 由电话公司提供，使用已有的数字电话线（每户一条线）：
  - DSL modem：转换模拟信号和数字信号
  - Splitter：合并/分离话音和数据（用户侧）
  - DSLAM：汇聚/分离多条DSL线路（ISP侧）
- 上行速率< 2.5 Mbps (典型地 < 1 Mbps)
- 下行速率< 24 Mbps (典型地 < 10 Mbps)

住宅接入：电缆网



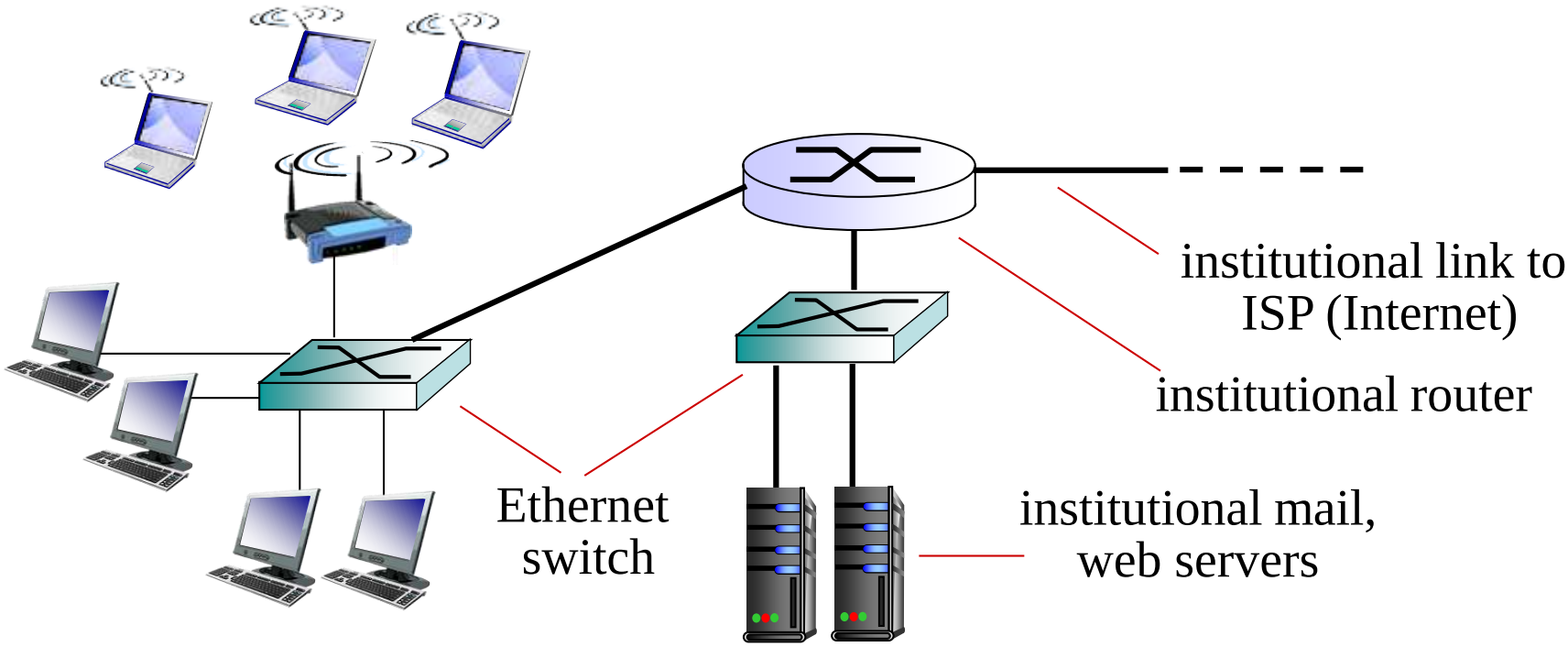
- 有线电视公司提供，使用已有的有线电视基础设施
- 数据和电视信号在同一条电缆中传输

住宅接入：电缆网



- 混合光纤同轴电缆HFC: hybrid fiber coax
  - 有线电视网由光纤网+电缆网组成（光纤网未画出）
  - cable modem、Splitter、电缆、光纤、CMTS构成接入网
  - 下行速率最高 30Mbps，上行速率最高 2 Mbps
  - 几百~几千户家庭共用一条电缆

企业接入网：以太网（Ethernet）

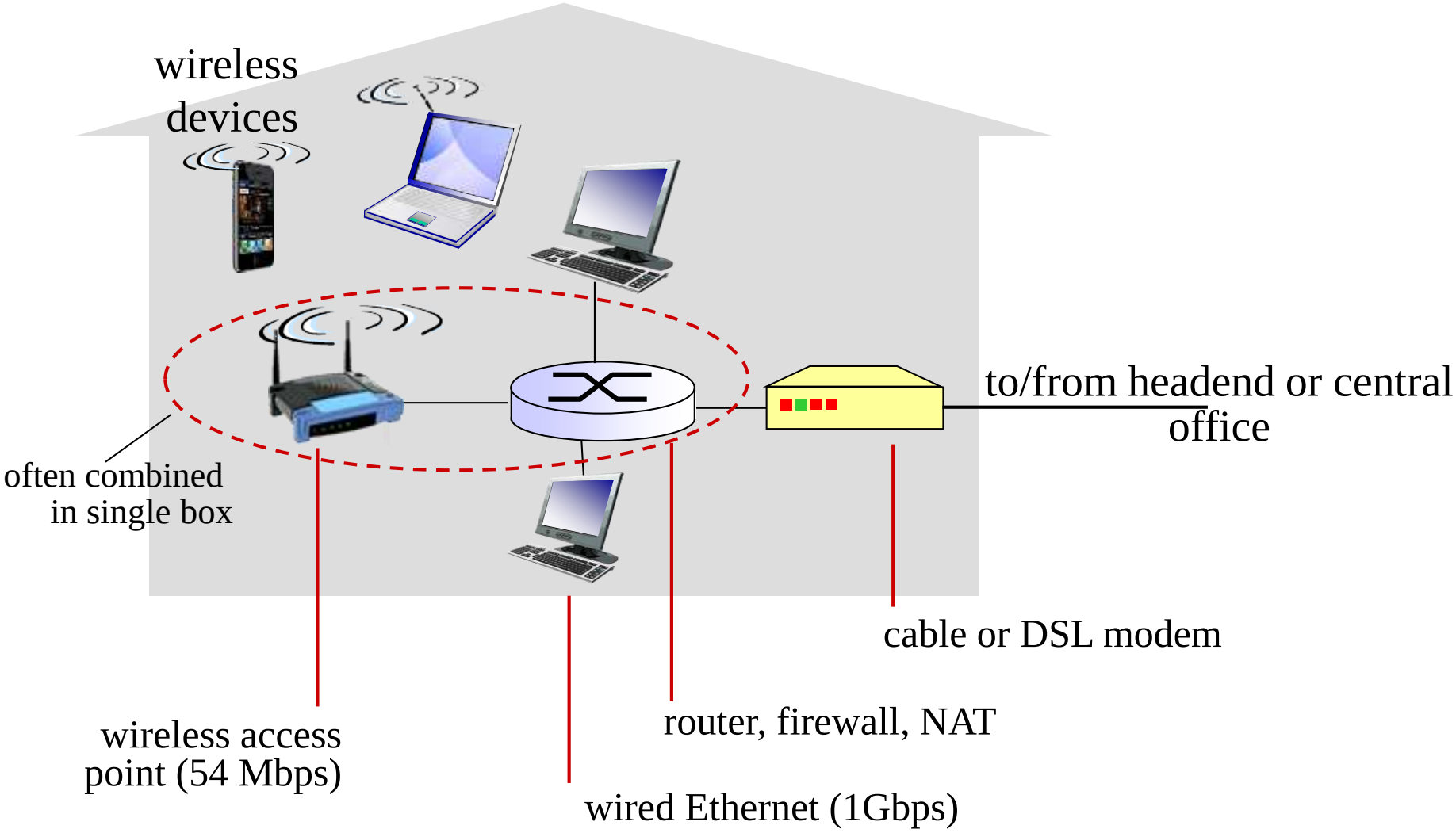


- 用于公司、学校及有较多终端的家庭：
  - 以太网交换机及链路构成接入网
  - 传输速率：10 Mbps, 100Mbps, 1Gbps, 10Gbps

无线接入网：无线局域网（Wifi）

- 公司或个人提供基站（接入点），将移动终端连接到有线网络：
  - 终端与基站相距几十米内
  - 基站通常位于有线网络上
  - 无线传输速率：11 Mbps 、54Mbps、450Mbps
  - 无线局域网是共享的

一个典型的家庭网络


wireless
devices
to/from headend or central
office
often combined
in single box
cable or DSL modem
wireless access
point (54 Mbps)
router, firewall, NAT
wired Ethernet (1Gbps)

无线接入网：广域无线接入

- 由移动通信公司提供，使用现有的蜂窝电话网络
- 基站可为数万米半径内的用户提供无线接入服务
- 传输速率（共享信道）：
  - 3G：最大（静止）2Mbps
  - 4G：下行100Mbps，上行20Mbps

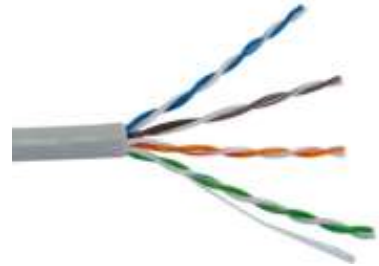
to Internet

1.2.2 物理媒体（传输媒体，传输介质）

- 设备之间通过物理媒体相连，物理媒体两端各需要一对收/发设备
- 在一条路径上，每对设备之间的物理媒体可以不同
- 导引型媒体:
  - 信号沿固体媒体传播，如铜线，光纤
- 非导引型媒体:
  - 信号在空间自由传播，如电磁波

双绞线：

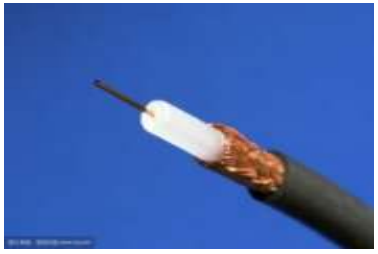
- 两条绝缘的铜导线：
  - 3类线：10 Mbps
  - 5类线：100Mbps~1Gbps
  - 6类线：10Gbps
- 电话线，网线



物理媒体（续）

同轴电缆:

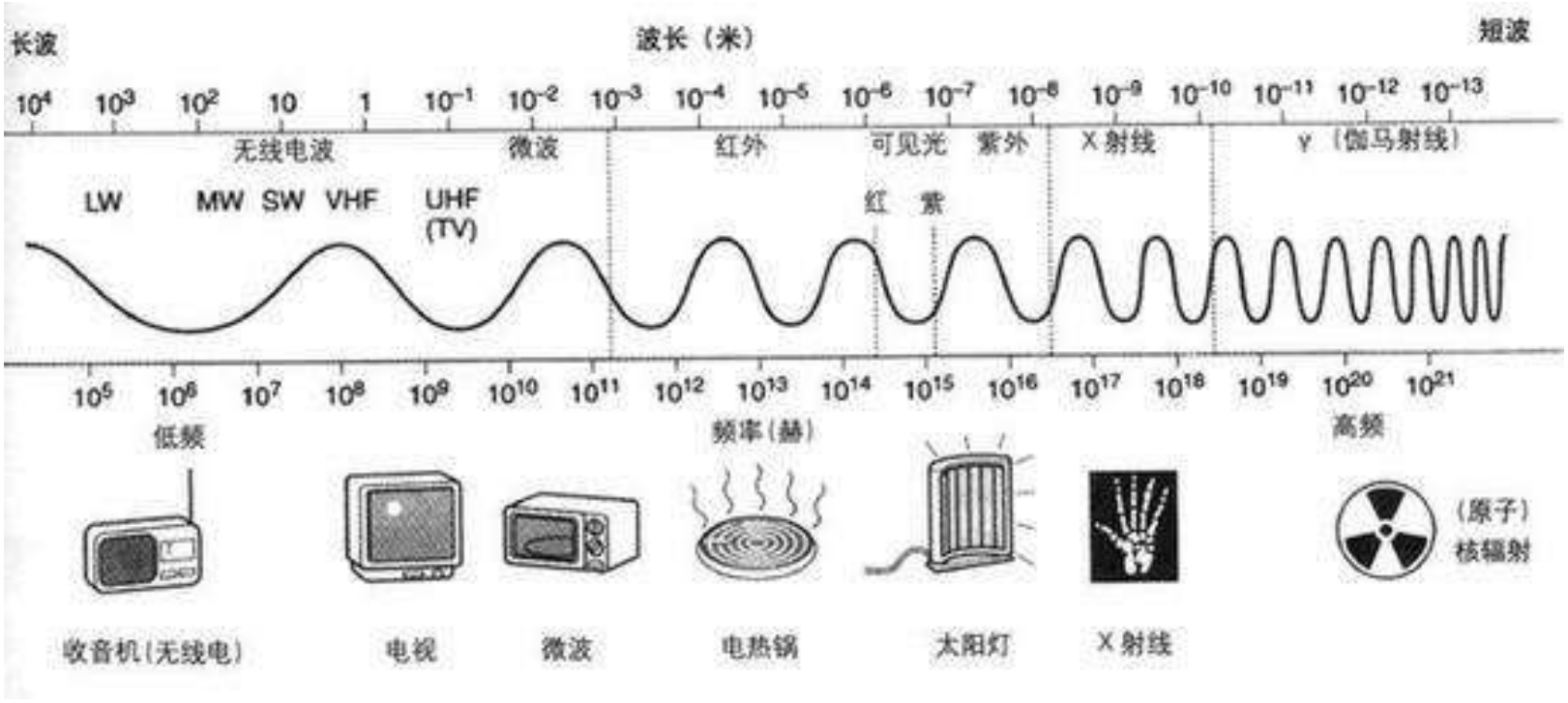
- 铜芯和网状屏蔽层组成一对同心导电体
- 有线电视电缆



光纤:

- 能引导光脉冲的玻璃纤维
- 传输速率:
  - 几十 ~ 几百Gbps
- 低误码率，长距离传输，抗电磁干扰

物理媒体：电磁波



- 蓝牙：2.4GHz，10米左右
- Wifi：2.4GHz，几十米
- 红外：室内短距离
- 陆地微波：2GHz，长距离
- 卫星：2GHz，长距离、大范围
- 可见光：1-2km，50Gbps

Chapter 1: roadmap

1.1 What *is* the Internet?

1.2 Network edge

- end systems, access networks, links

1.3 Network core

- circuit switching, packet switching, network structure

1.4 Delay, loss and throughput in packet-switched networks

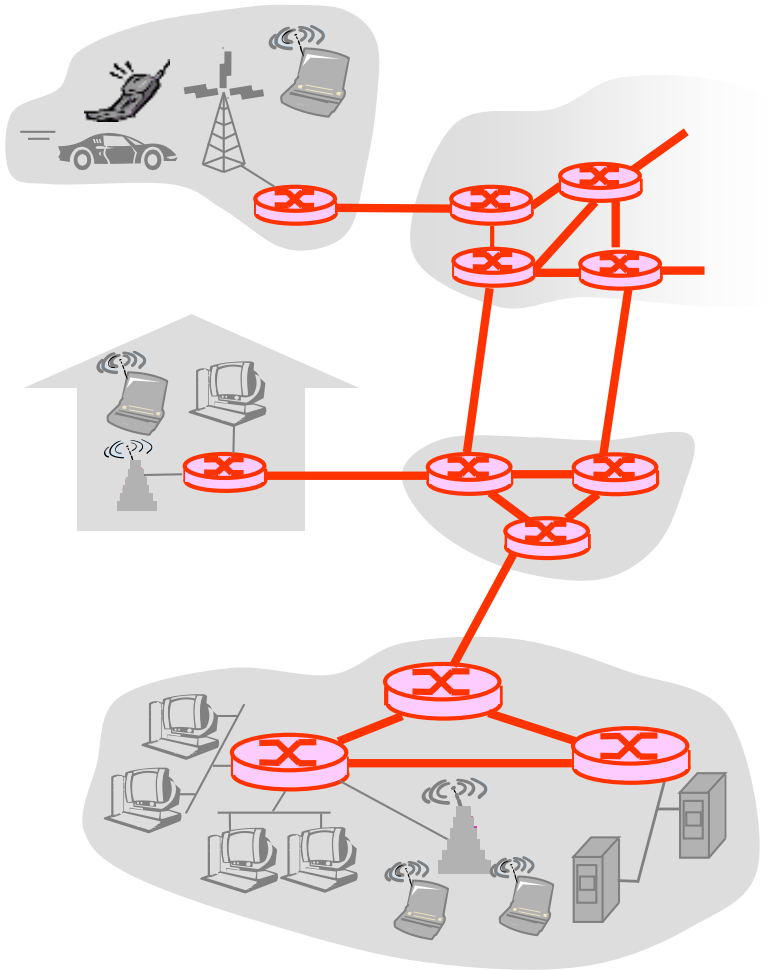
1.5 Protocol layers, service models

1.6 Networks under attack: security

1.7 History

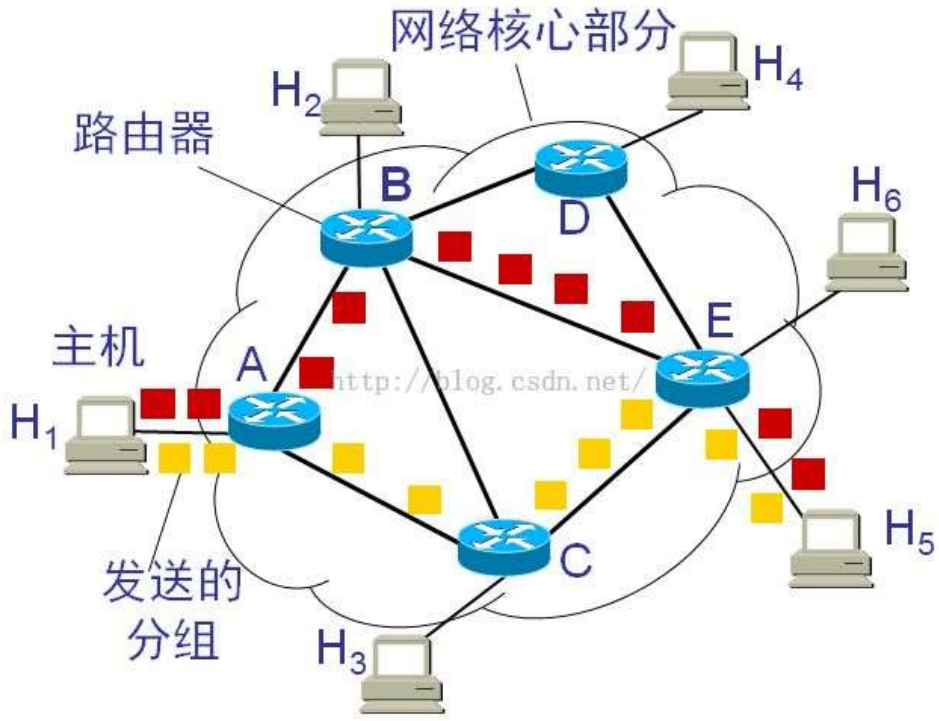
1.3 网络核心

- 网络核心：由路由器和链路形成的网状网络
- 任务：将数据包从发送侧的边缘路由器，传送到接收侧的边缘路由器
- 基本问题：数据包如何在网络核心中高效地传递？
  - 分组传输延迟小
  - 网络吞吐量高
- 通信网络中移动数据的两种基本方法：
  - 电路交换（独占信道）：电话网使用
  - 分组交换（复用信道）：计算机网络使用



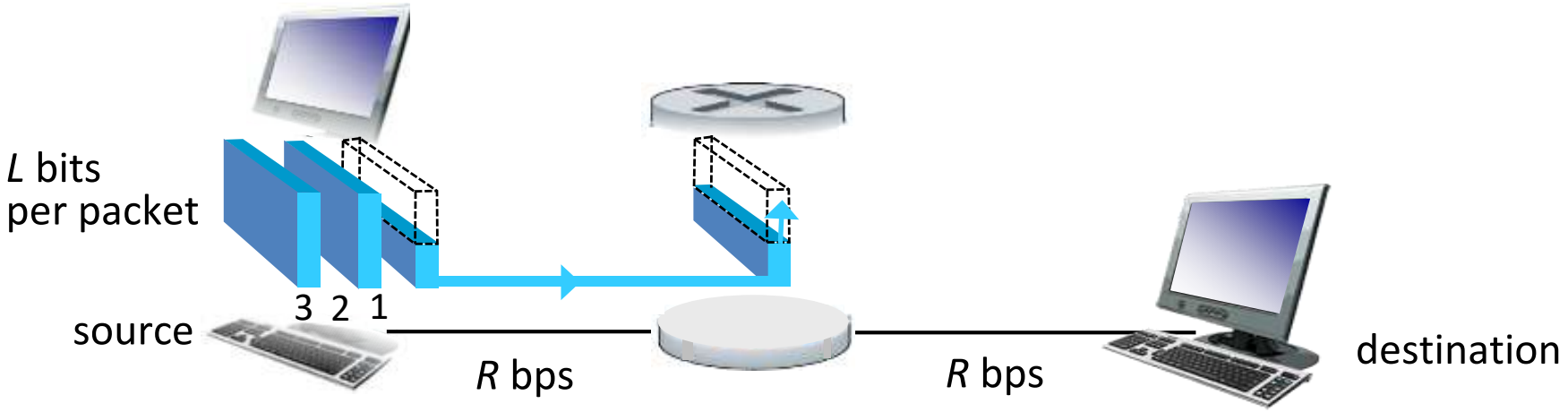
1.3.1 分组交换（packet switching）

- 分组交换的过程：
  - 主机将要传输的数据分段，并组装成一系列分组
  - 交换：在传输路径上，交换设备从一条链路上接收分组，将其发送到另一条链路上
  - 存储转发：交换设备在接收到完整的分组后，才可以开始转发



- 思考：
  - 为什么不是边收边发？

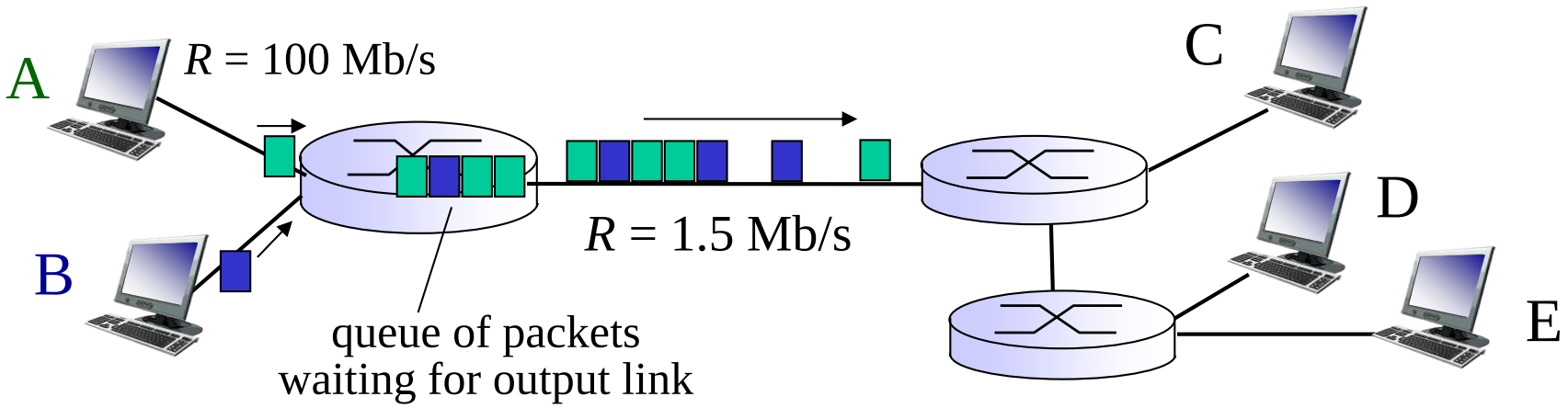
存储转发引入序列化延迟



- 将一个分组全部推送到一条链路上，耗时 L/R 秒
- 将一个分组从源发送到目的，总耗时 = 2 L/R （不考虑信号传播时间）
- 3个分组从源终端发送到目的终端，总耗时=？
  - 4 L/R

- 问题：P个分组经过N条链路的总耗时是多少？
  - （P+N-1）L/R
- **当P远大于N时，存储转发不会引入过多的延迟!**

存储转发引入排队延迟和丢包



- 排队延迟：分组在输出链路的缓存中排队，引入延迟
- 丢包：若输出链路的缓存满，溢出的分组被丢弃
- 当大量分组集中到达时，排队延迟和丢包较严重

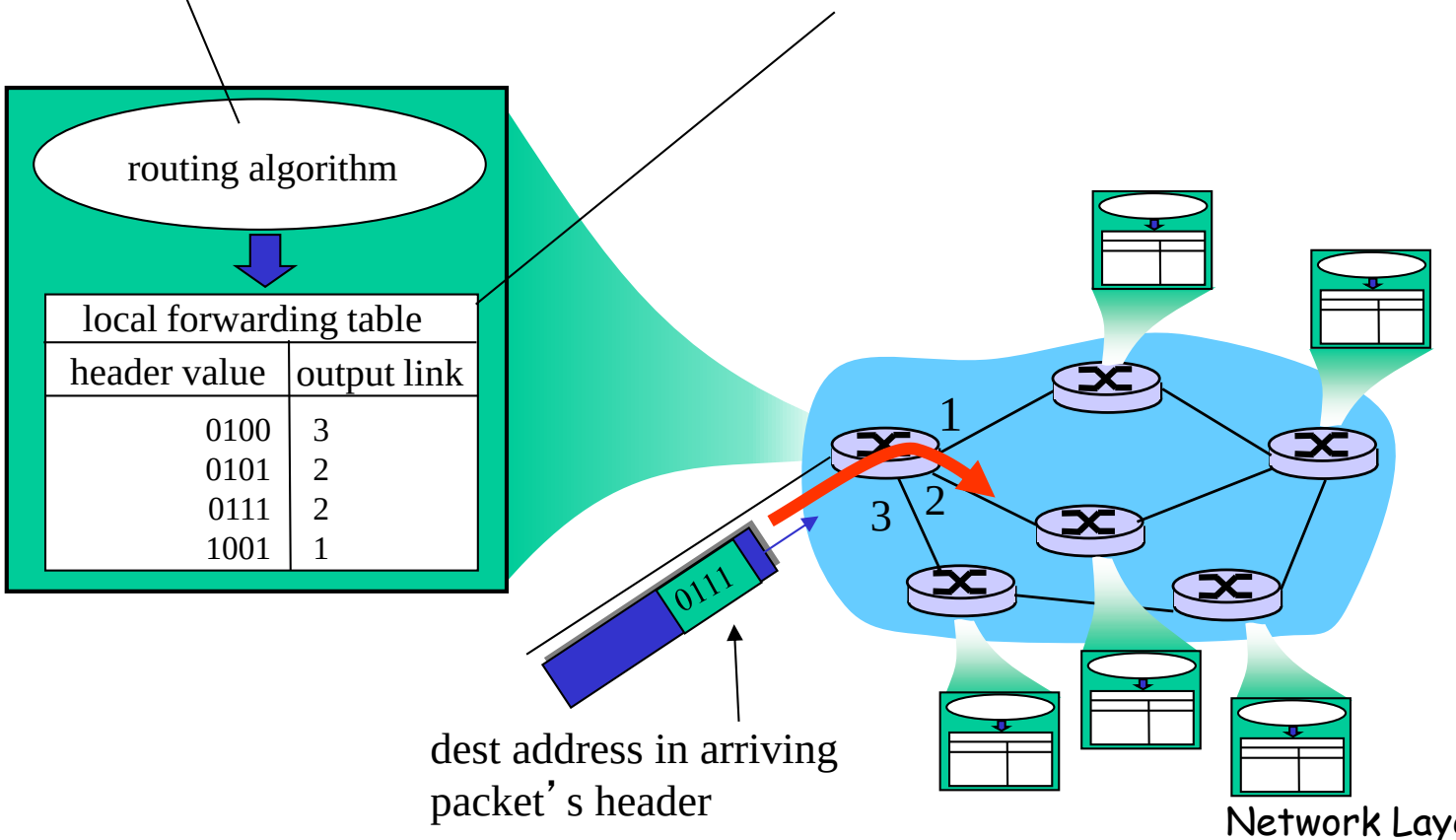
网络核心的两个重要功能

选路（routing）：

交换设备确定不同目的地的转发端口，生成转发表

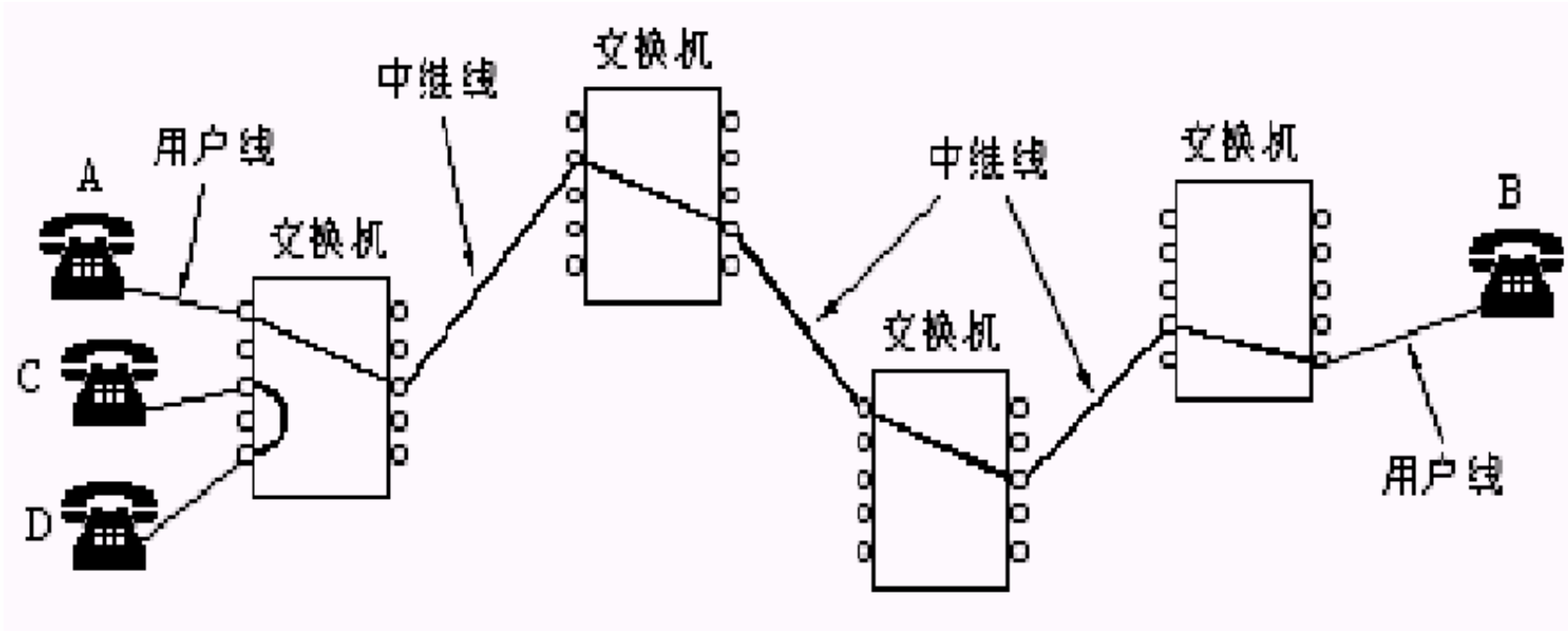
转发（forwarding）：

交换设备按照转发表，将分组移动到相应的输出链路



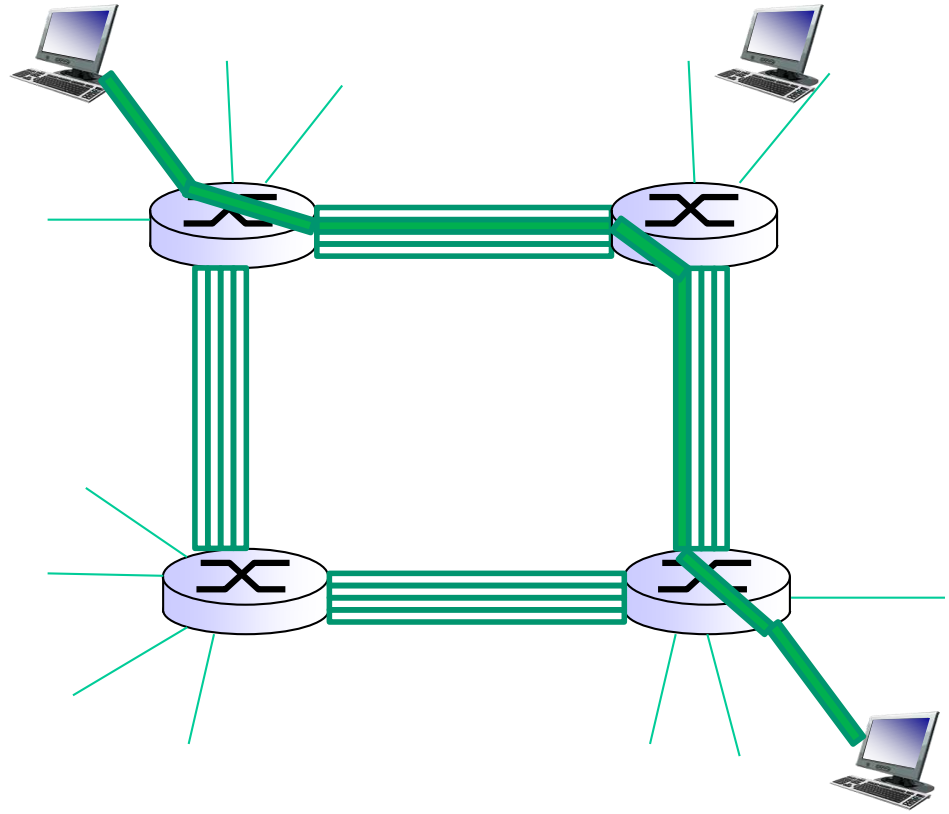
1.3.2 电路交换(circuit switching)

- 电话网采用电路交换：
  - 通话前完成两部电话机之间的电路接续，通话结束后释放整条电路
- 本质是预留资源和独占资源



概念区分：链路和电路

- 链路（link）：
  - 物理媒体，也称信道（channel）
  - 可以通过某种方式划分为若干条独立的子信道
- 电路（circuit）：
  - 物理媒体中的一条子信道

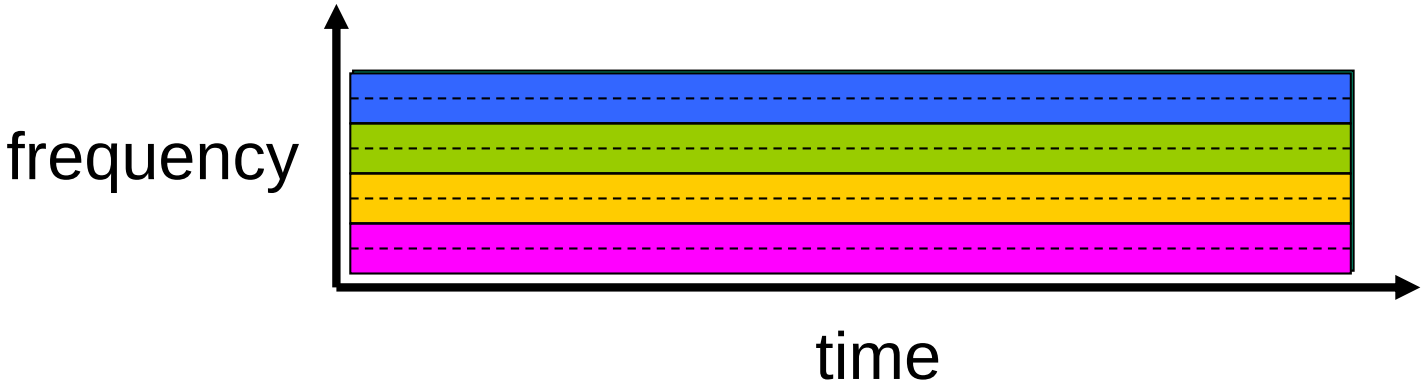


多路复用（multiplex）

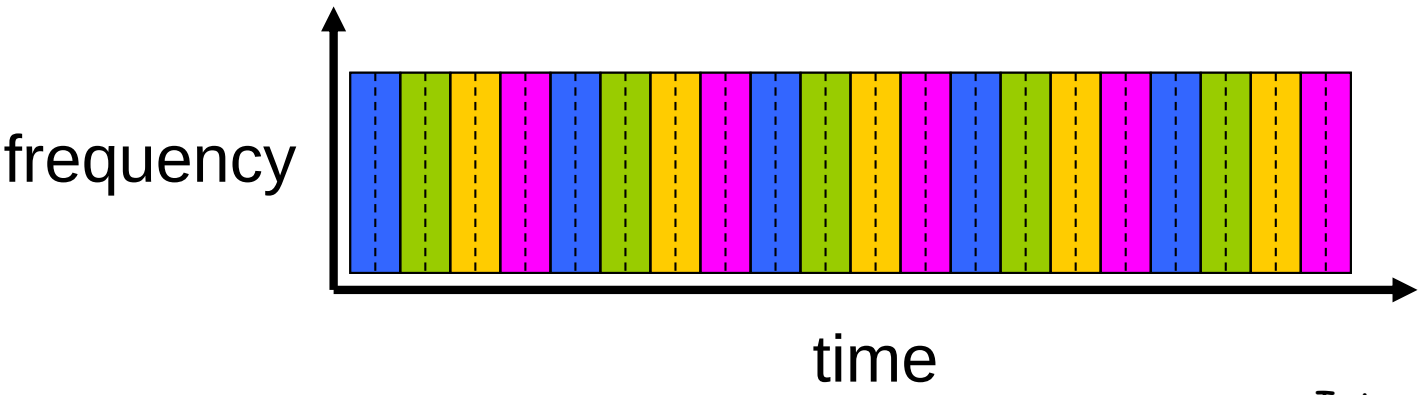
Example:

4 users

频分复用FDM



时分复用TDM



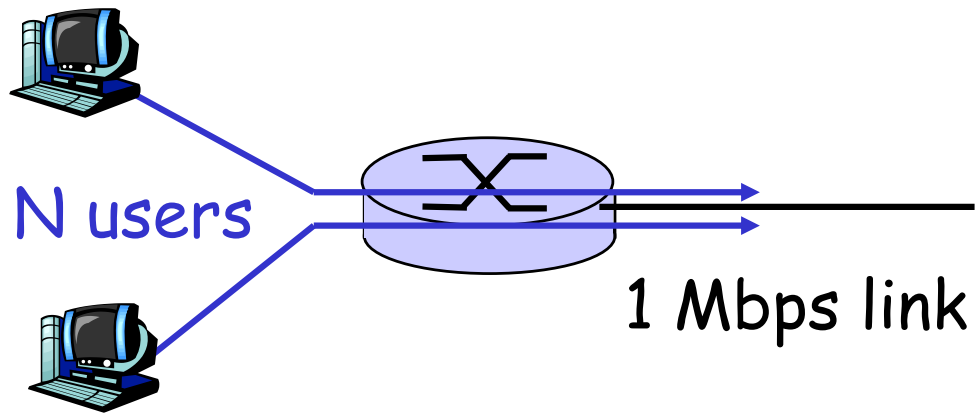
采用电路交换的文件传输时间

- How long does it take to send a file of 640,000 bits from host A to host B over a circuit-switched network?
  - All links are 1.536 Mbps
  - Each link uses TDM with 24 slots/sec
  - 500 msec to establish end-to-end circuit
- Let's work it out!
  - 数据传输速率：1.536Mbps/24 = 64kbps
  - 传输数据的时间：640kbits/64kbps = 10s
  - 总时间：500ms+10s = 10.5s

为什么采用分组交换?

同样的链路容量，分组交换允许支持更多的用户！

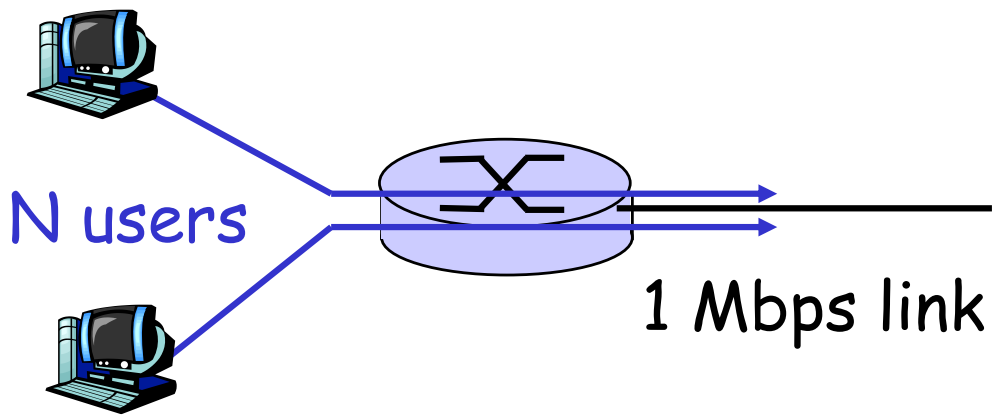
- 1 Mb/s link
- each user:
  - 100 kb/s when "active"
  - active 10% of time
- 电路交换（固定分配）
  - 10 users
- 分组交换（按需分配）
  - with 35 users, probability > 10 active at same time is less than .0004



为什么采用分组交换?

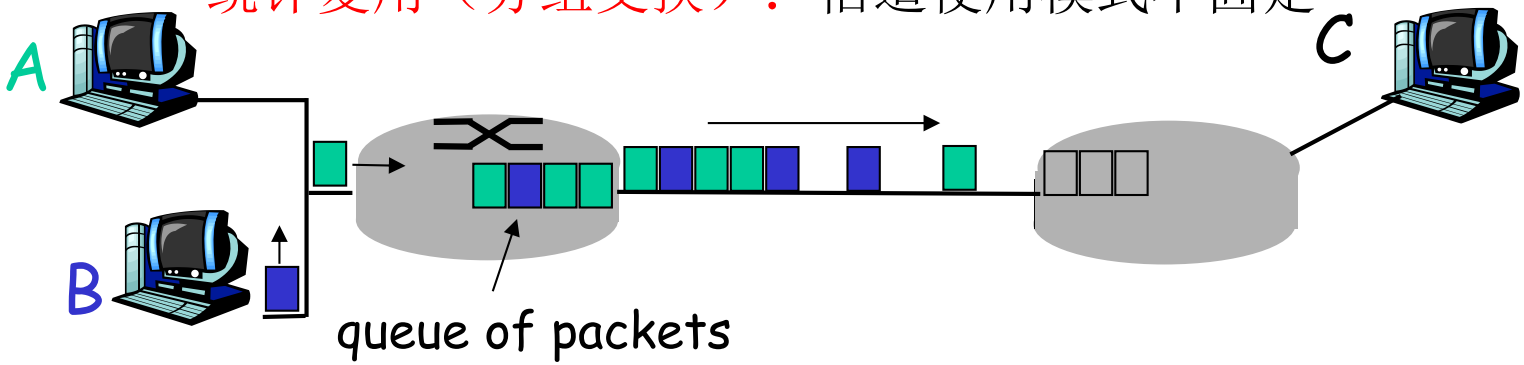
轻负载时，分组交换可以更快地服务用户！

- 1 Mb/s link
- Only one active user:
  - 1000 1kb-packet's
- 电路交换（固定分配）
  - Need 10s
- 分组交换（按需分配）
  - Need 1s

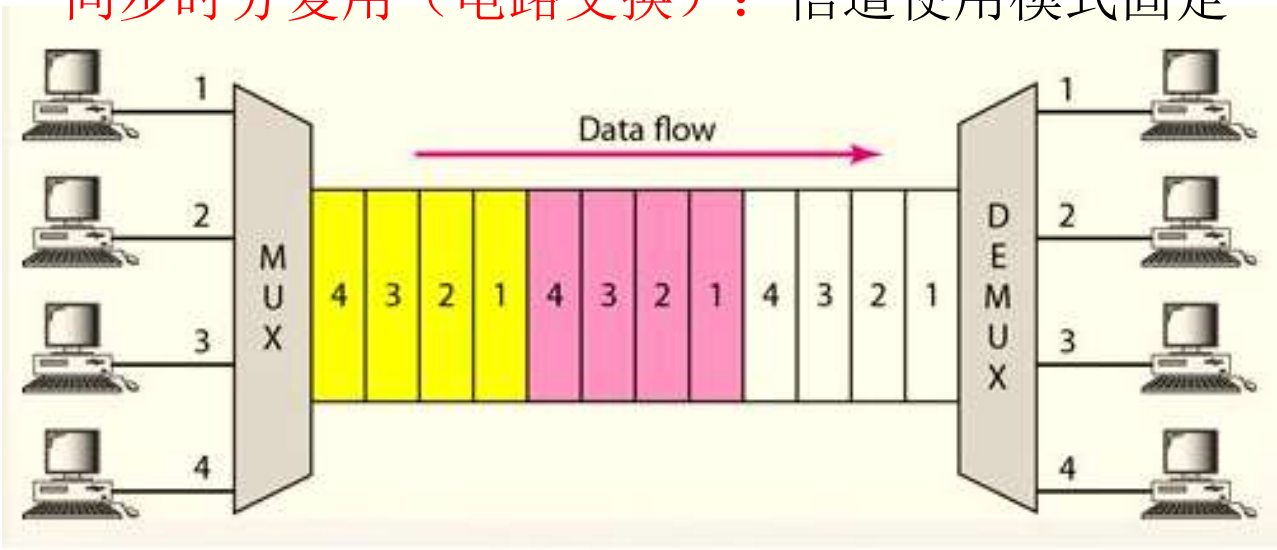


统计复用 vs 同步时分复用

统计复用（分组交换）：信道使用模式不固定



同步时分复用（电路交换）：信道使用模式固定



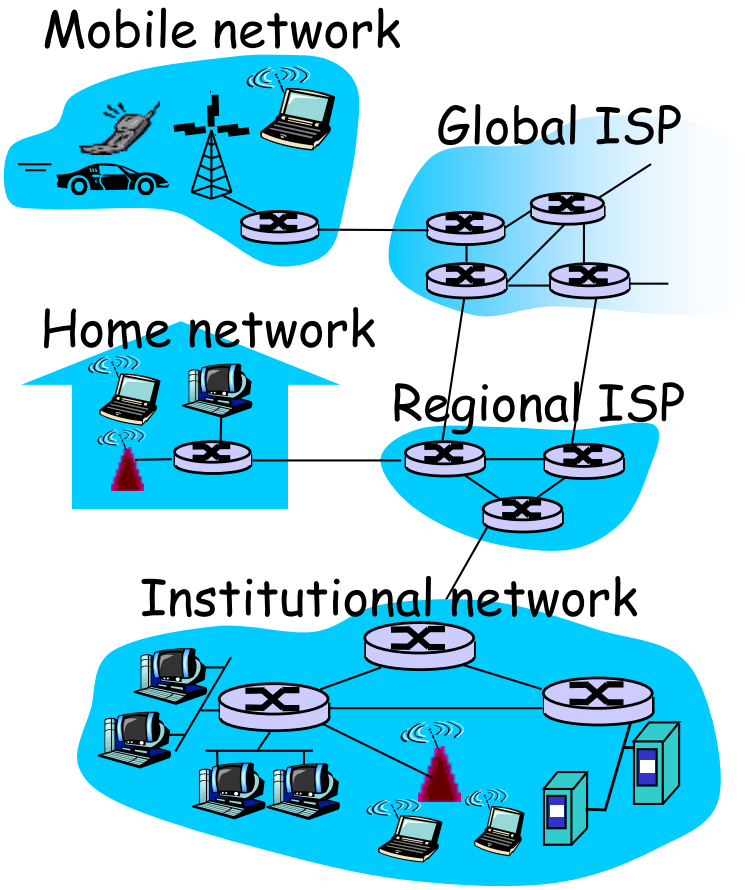
Packet switching versus circuit switching

is packet switching a "slam dunk winner?"

- great for bursty data
  - resource sharing
  - simpler, no call setup
- excessive congestion possible: packet delay and loss
  - protocols needed for reliable data transfer, congestion control
- Q: How to provide circuit-like behavior?
  - bandwidth guarantees needed for audio/video apps
  - still an unsolved problem (chapter 9)

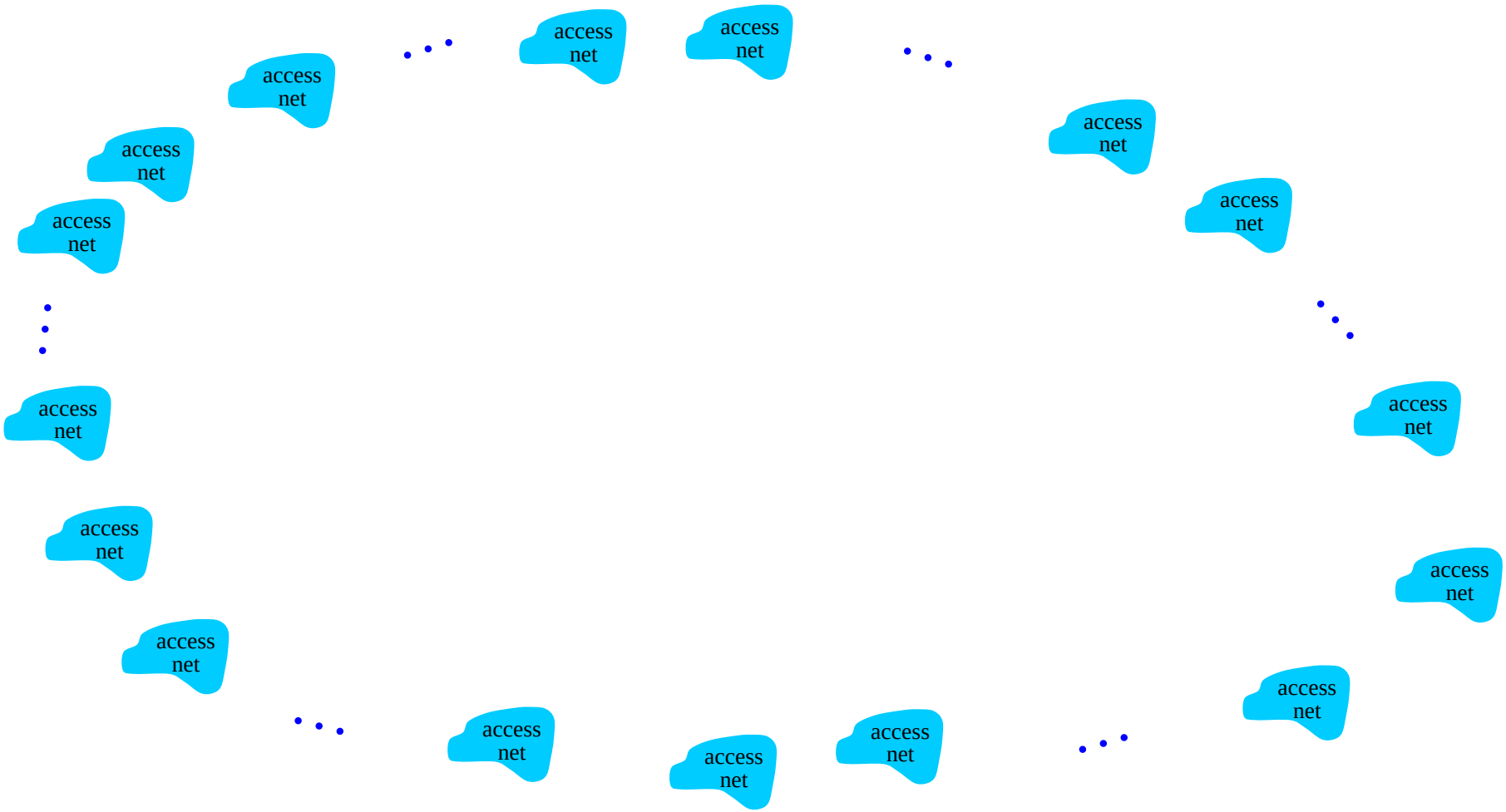
1.3.3 网络的网络

- 因特网是由一群ISP组成的网络的网络
- 网络核心的任务是将全球的本地ISP连接在一起
- 问题：如何连接？



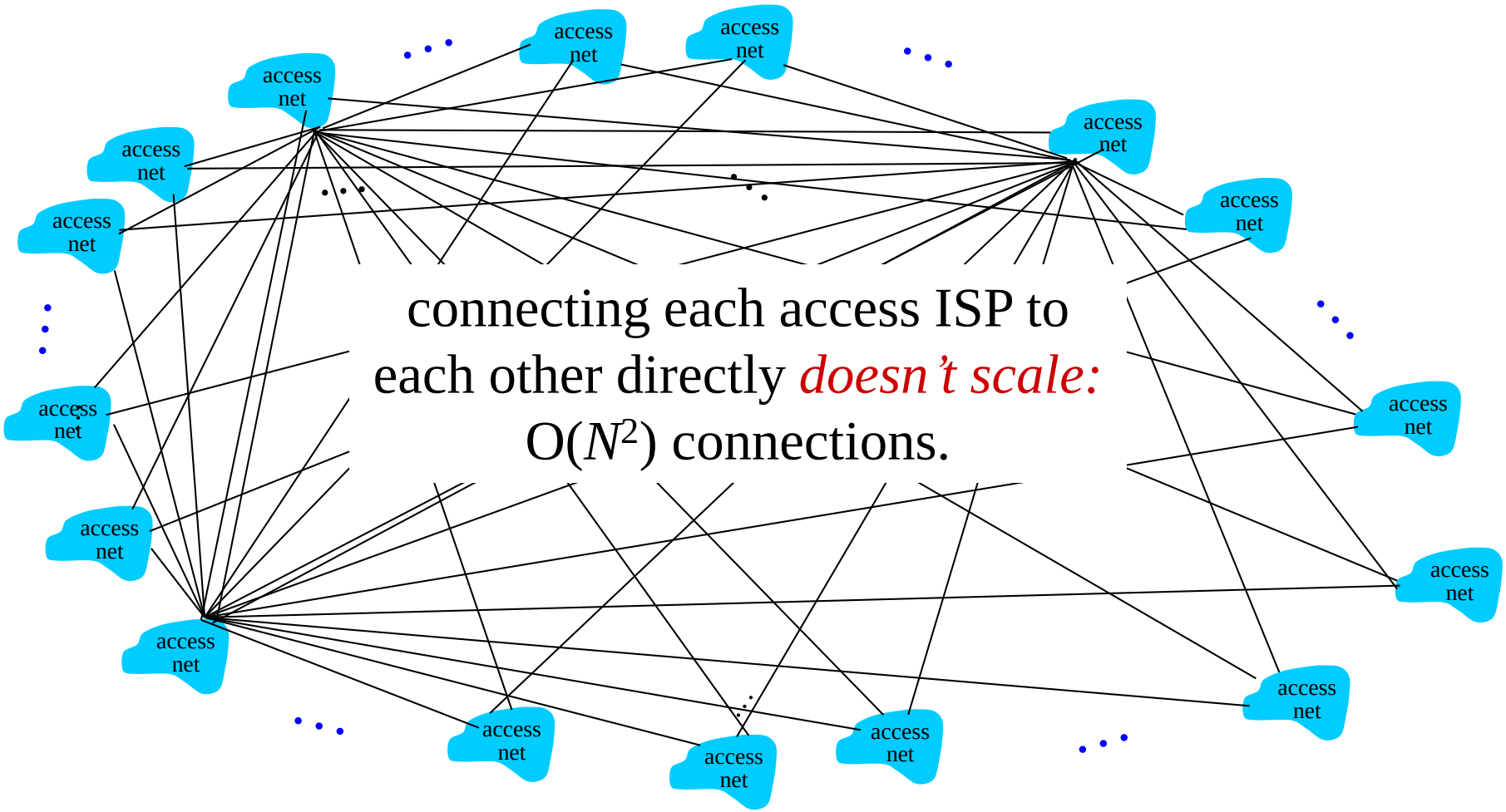
网络的网络

Question: given *millions* of access ISPs, how to connect them together?



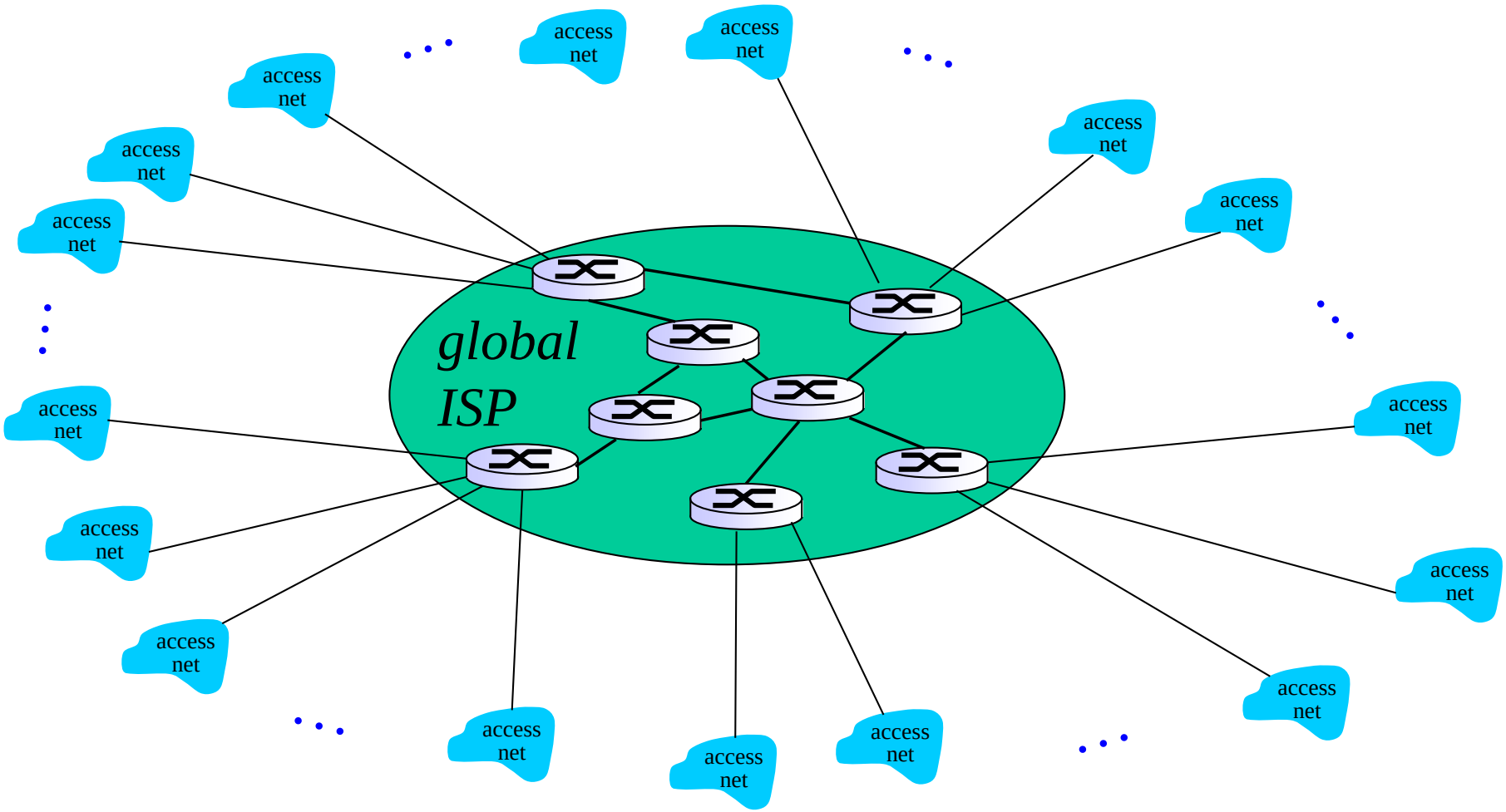
网络的网络：朴素的方法

Option: connect each access ISP to every other access ISP?



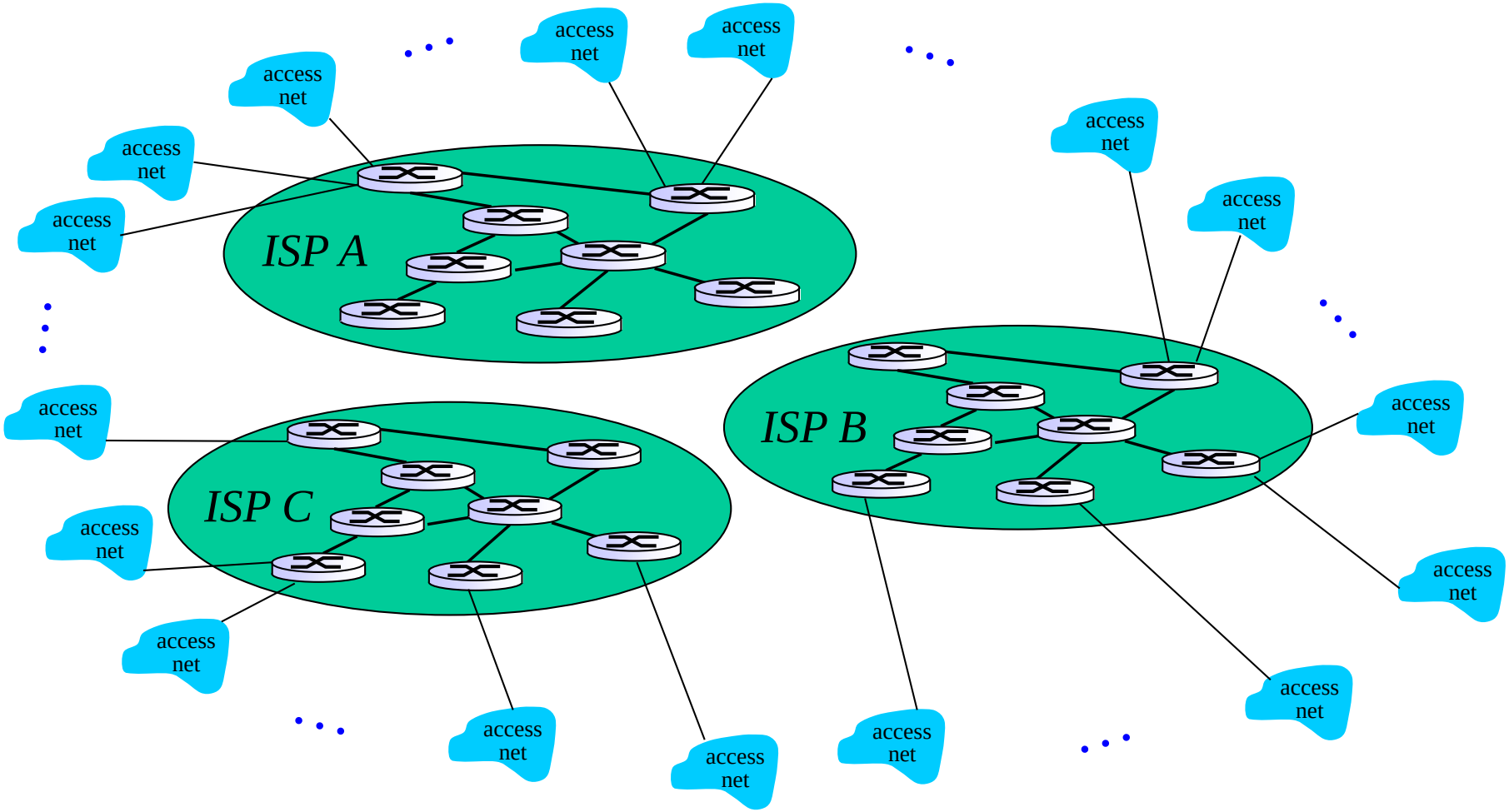
网络的网络：连接到一个全球ISP

Option: connect each access ISP to a global transit ISP? Customer and provider ISPs have economic agreement.



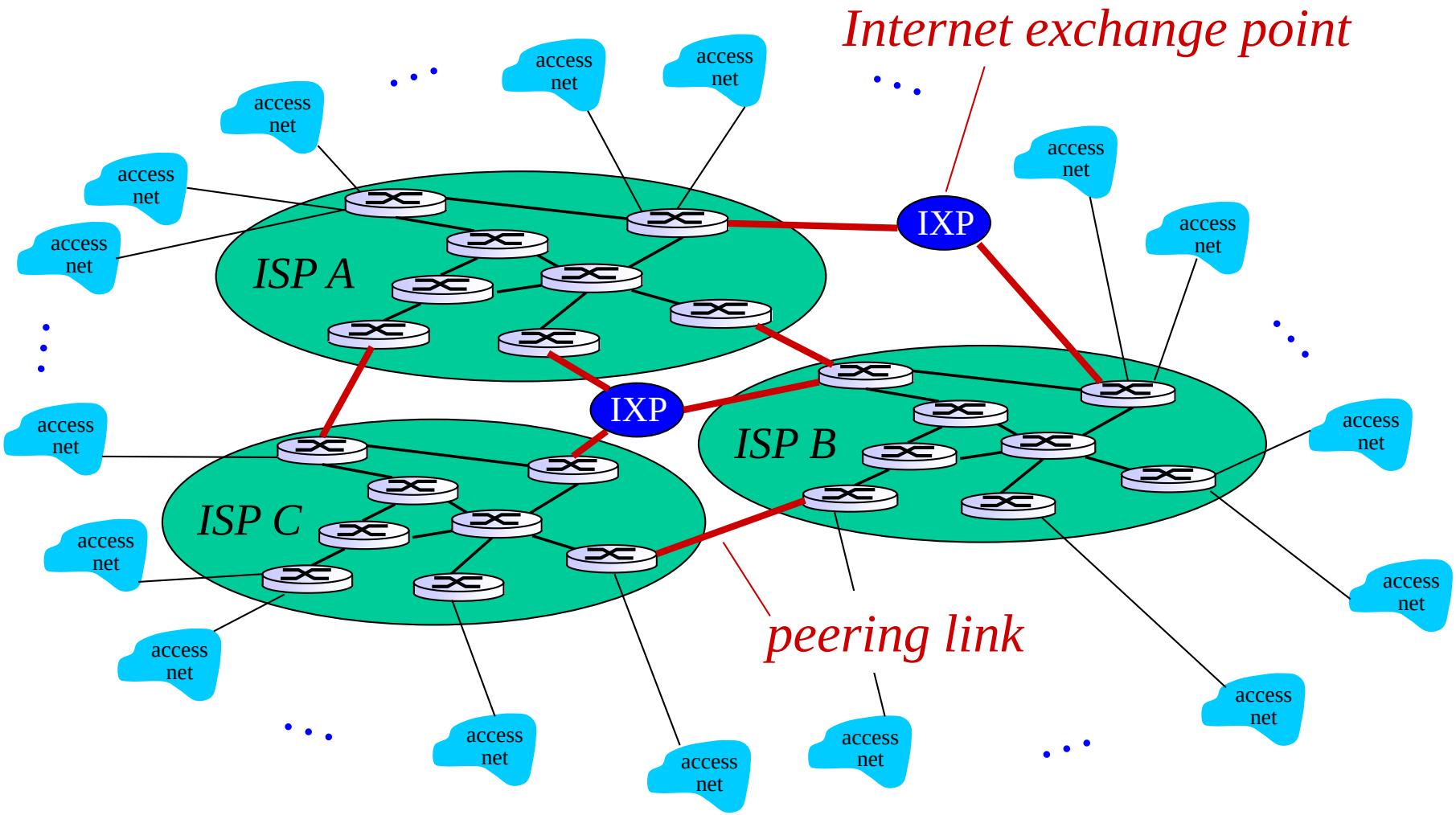
网络的网络：建立多个全球ISP

But if one global ISP is viable business, there will be competitors ….

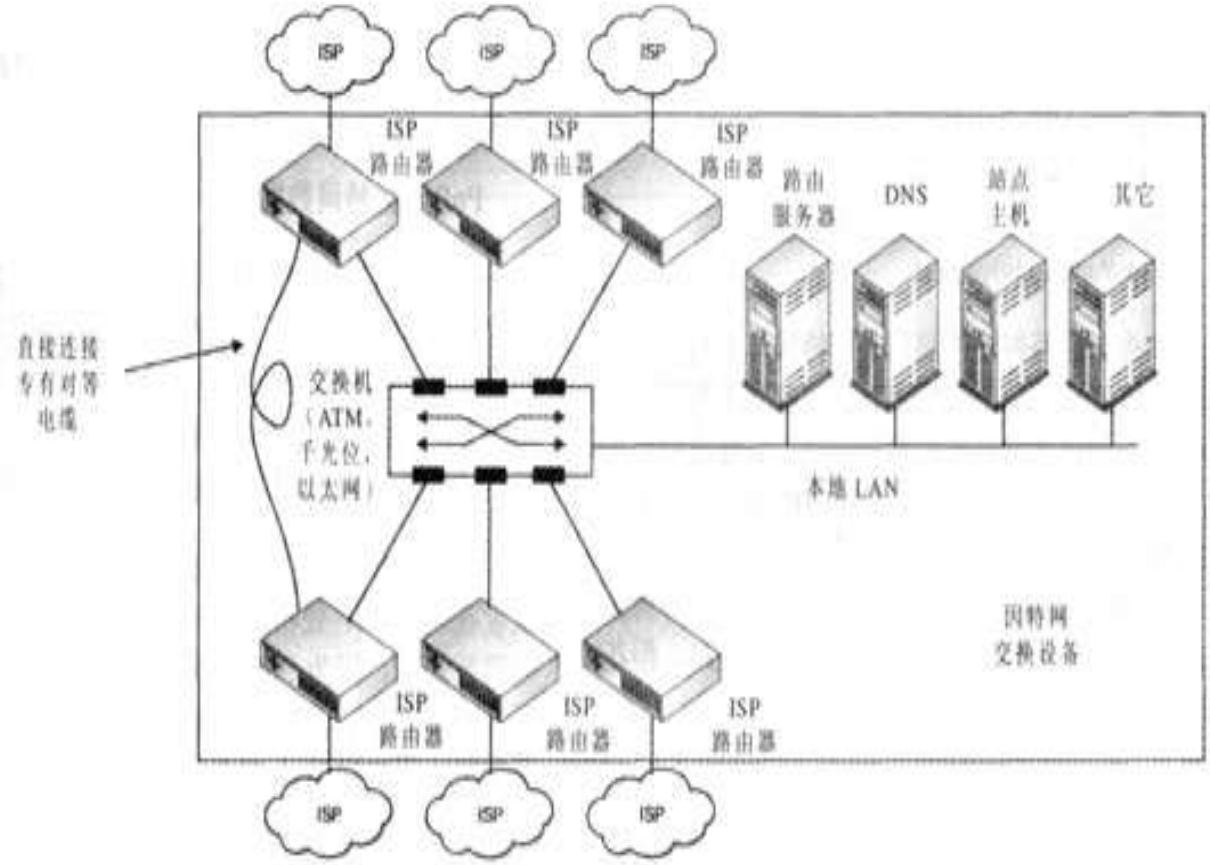


网络的网络：多个全球ISP

But if one global ISP is viable business, there will be competitors …. which must be interconnected

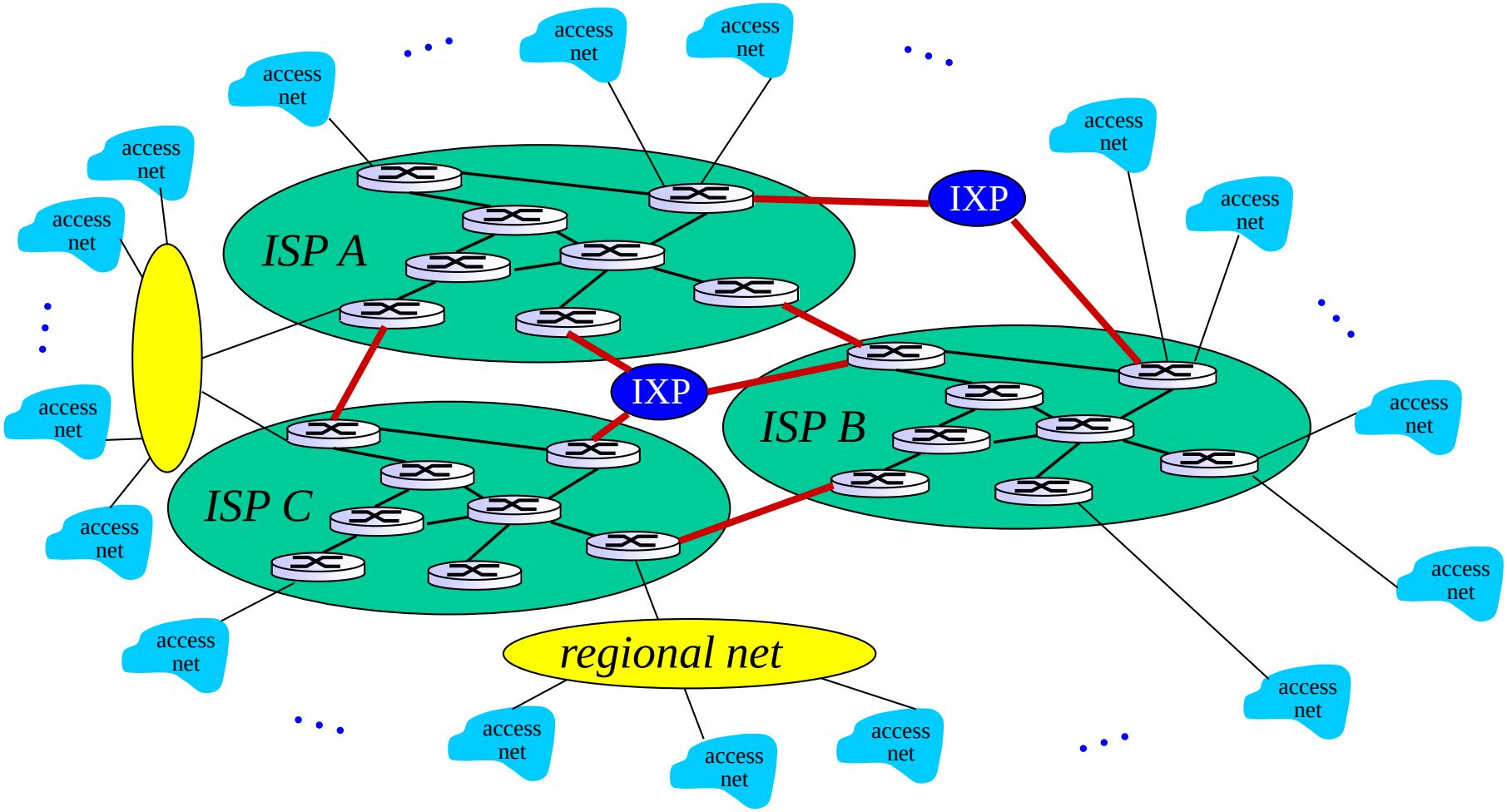


因特网交换点（IXP)


ISP
ISP
ISP
ISP
路由器
ISP
路由器
ISP
路由器
路由
服务器
DNS
站点
主机
其它
直接连接
专有对等
电缆
交换机
（ATM，
千兆位，
以太网）
本地 LAN
因特网
交换设备
ISP
路由器
ISP
路由器
ISP
路由器
ISP
ISP
ISP

网络的网络：多层结构

… and regional networks may arise to connect access nets to ISPS

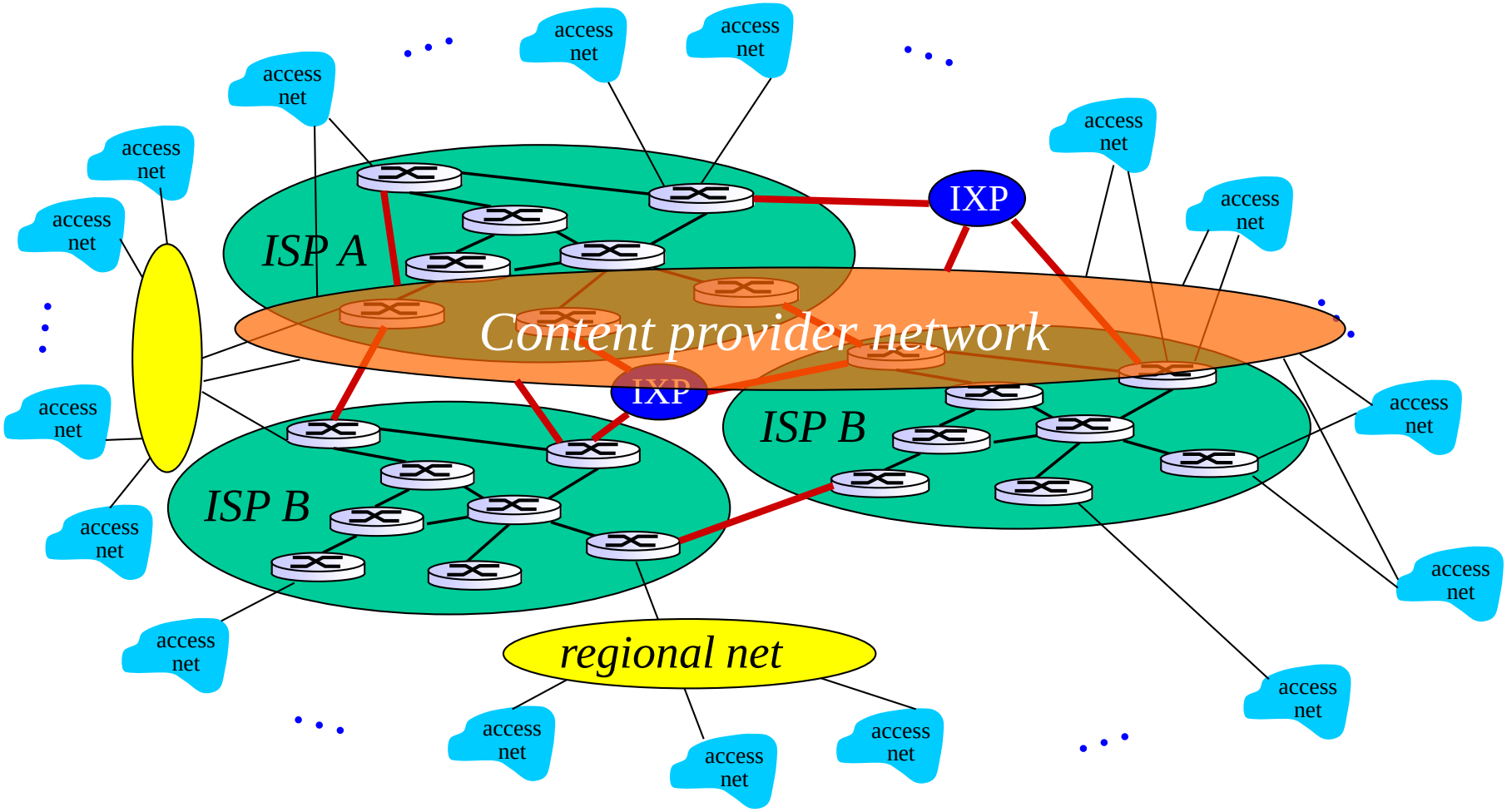


因特网生态系统

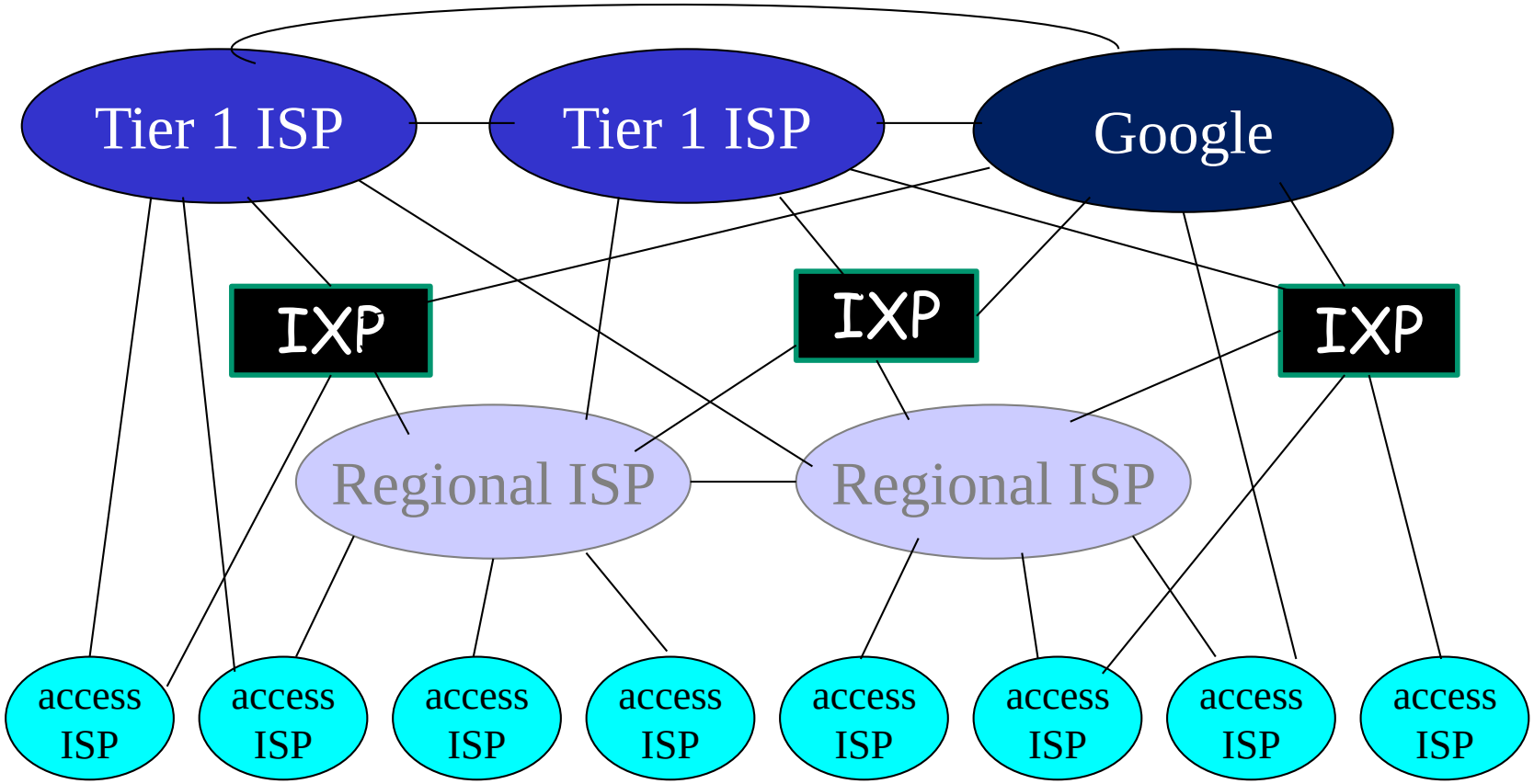
- 接入ISP
- 地区ISP
- 第一层ISP
- 对等链路
- 因特网交换点IXP：多个ISP共同对等的地方
- 存在点PoP（Point of Presence）：低层ISP接入高层ISP的地方
- 多宿（multi-home）：一个低层ISP可以接入多个高层ISP

网络的网络：内容提供商网络

… and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users



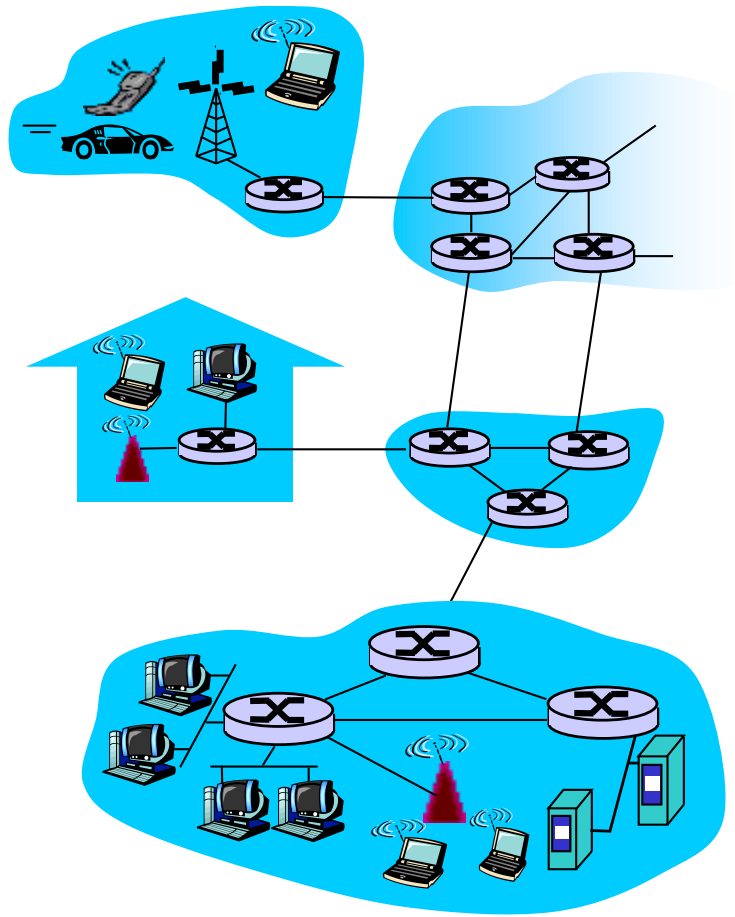
网络的网络：今天的因特网结构



- at center: small # of well-connected large networks
  - "tier-1" commercial ISPs : national & international coverage
  - content provider network : private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs

小结

- 端系统
  - 调用因特网服务接口，实现分布式应用
  - 因特网中的通信过程对其不可见
- 接入网
  - 因特网到用户的“最后一公里”，将各类终端接入因特网
  - 关注物理媒体、信号传输技术
- 网络核心
  - 任务是高效、准确地投递分组到目的地
  - 关注选路、转发、拥塞控制等



Chapter 1: roadmap

1.1 What *is* the Internet?

1.2 Network edge

- end systems, access networks, links

1.3 Network core

- circuit switching, packet switching, network structure

1.4 Delay, loss and throughput in packet-switched networks

1.5 Protocol layers, service models

1.6 Networks under attack: security

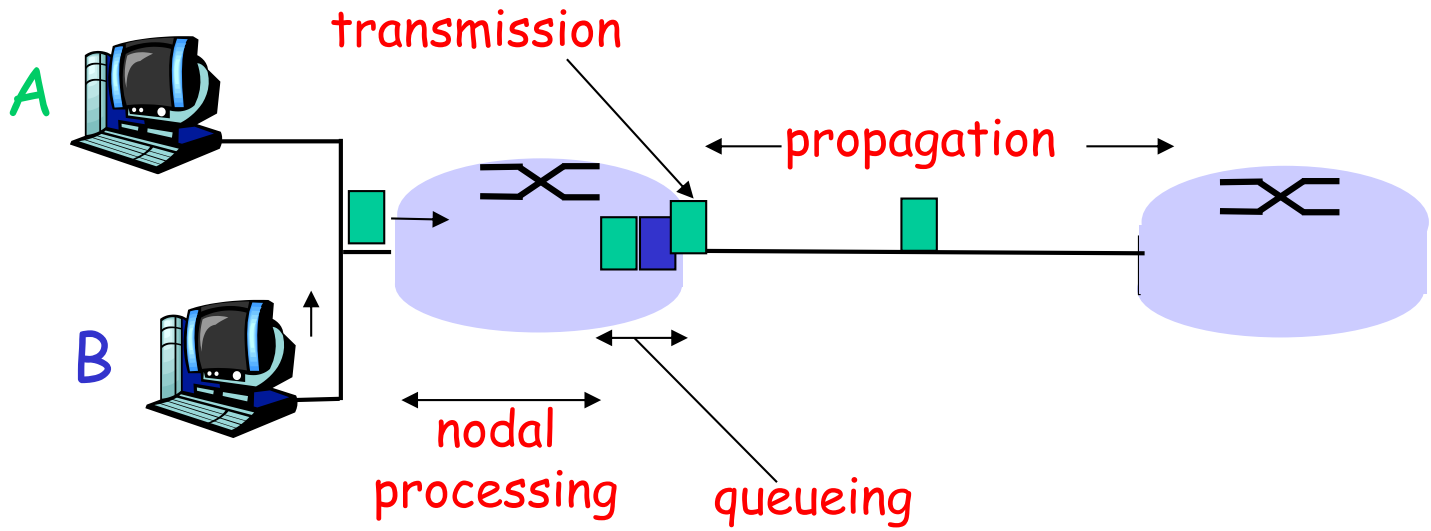
1.7 History

1.4 衡量网络性能的主要指标

- 延迟
  - 分组从源终端到达目的终端的时间
- 丢包率
  - 未成功交付到目的终端的分组比例
- 吞吐量
  - 单位时间内网络成功交付的数据量

分组延迟的来源

- 1. 节点处理:
  - 检查错误
  - 确定输出链路
- 2. 排队
  - 在输出缓存等待传输
  - 时间长短取决于链路负载大小



分组延迟的来源

3. 传输延迟:

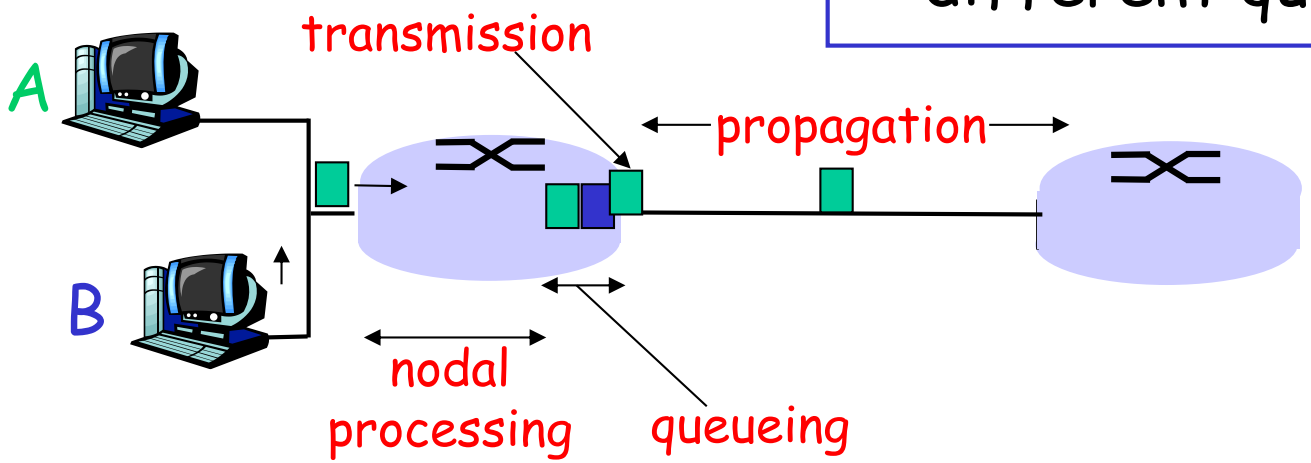
- R=link bandwidth (bps)
- L=packet length (bits)
- 将分组发送到链路上的时间 = L/R

  （分组序列化时间）

4. 传播延迟:

- d = length of physical link
- s = propagation speed in medium (~2×10^8 m/sec)
- propagation delay = d/s

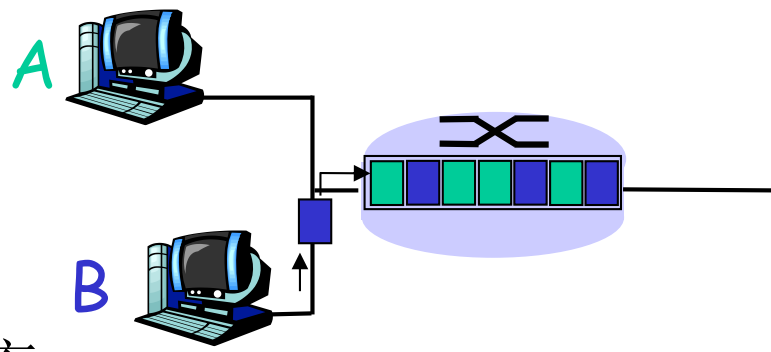
Note: s and R are *very* different quantities!



节点延迟

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

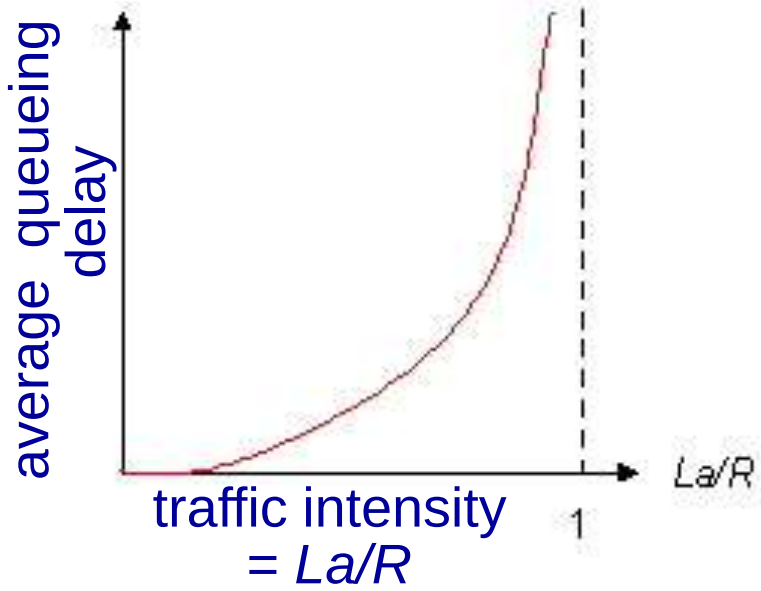
- d_{proc} = 处理延迟
  - 典型地为几个微秒或更低
- d_{queue} = 排队延迟
  - 差异很大，取决于链路负载
- d_{trans} = 传输延迟
  - 微秒～毫秒，主要取决于链路速率
- d_{prop} = 传播延迟
  - 几微秒～几百毫秒，主要取决于链路长度



排队延迟与流量强度

- *R*: link bandwidth (bps)
- *L*: packet length (bits)
- a: average packet arrival rate

traffic intensity = La/R



- *La/R* ~ 0: avg. queueing delay small
- *La/R* -> 1: avg. queueing delay large
- *La/R* > 1: more "work" arriving than can be serviced, average delay infinite!

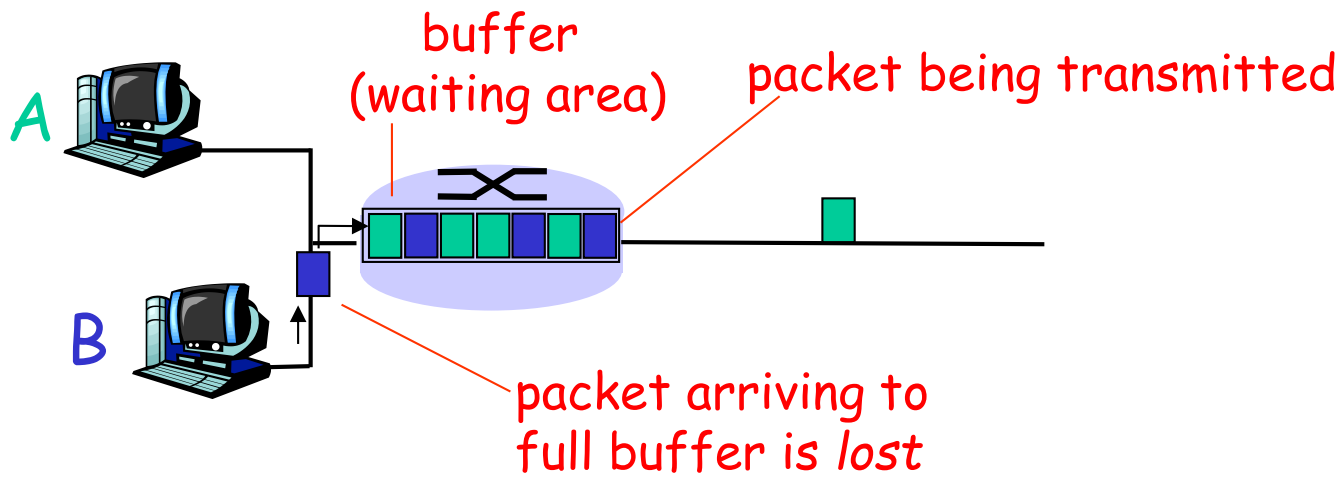
La/R ~ 0

La/R -> 1

* Check online interactive animation on queuing and loss

排队与丢包

- 输出队列的容量是有限的；队列满时，新来的分组被丢弃



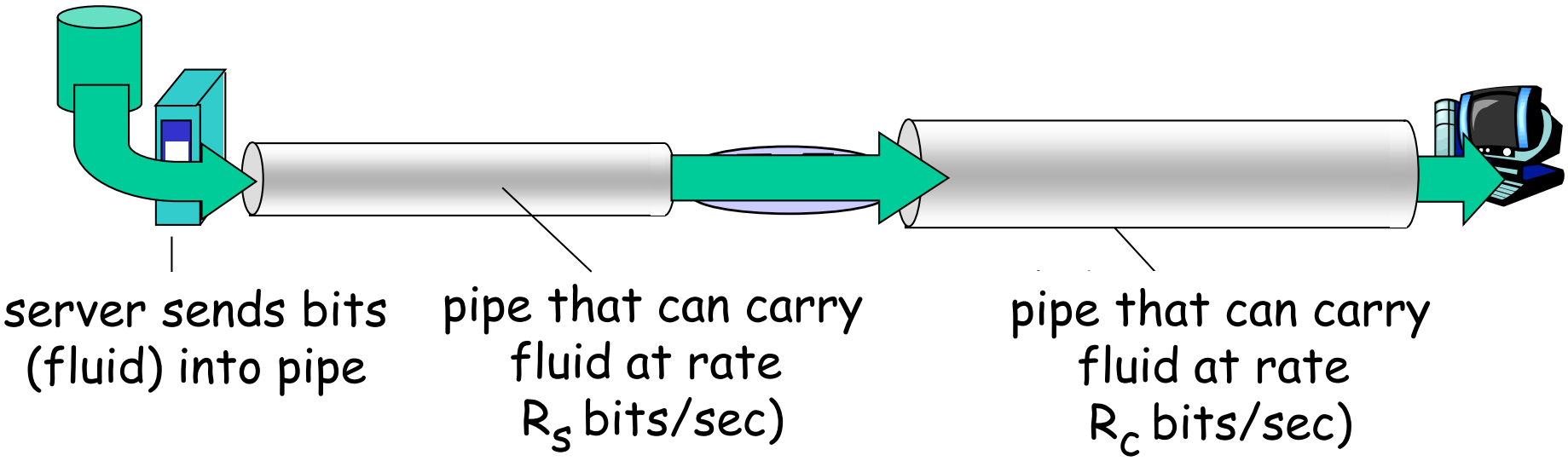
- 队列长度是一个重要的参数：
  - 队列太短：丢包率增大
  - 队列太长：排队延迟增大（也会造成间接丢包！）

端到端延迟

- 端到端延迟：
  - 分组传输路径上所有节点的节点延迟之和
- 对端到端延迟敏感的应用：
  - 高度敏感：实时交互应用，如网络电话、视频会议
  - 中度敏感：在线交互应用，如网页浏览
- 探测端到端延迟：
  - Ping, Traceroute

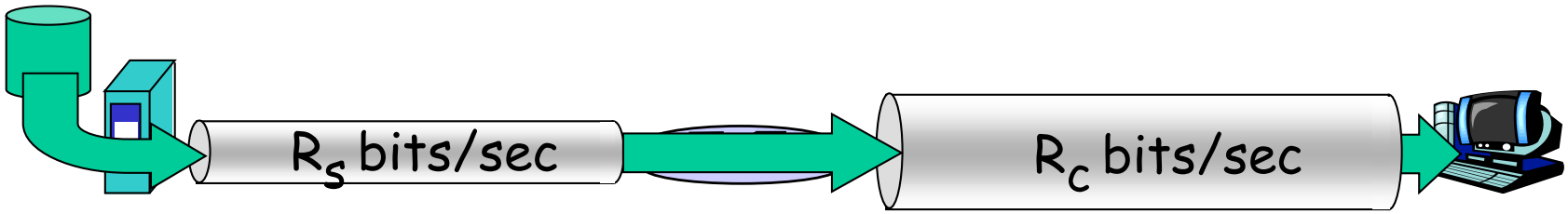
端到端吞吐量

- 单位时间内向接收端成功交付的数据量：
  - 瞬时吞吐量：给定时刻的传输速率
  - 平均吞吐量：较长时间内的传输速率

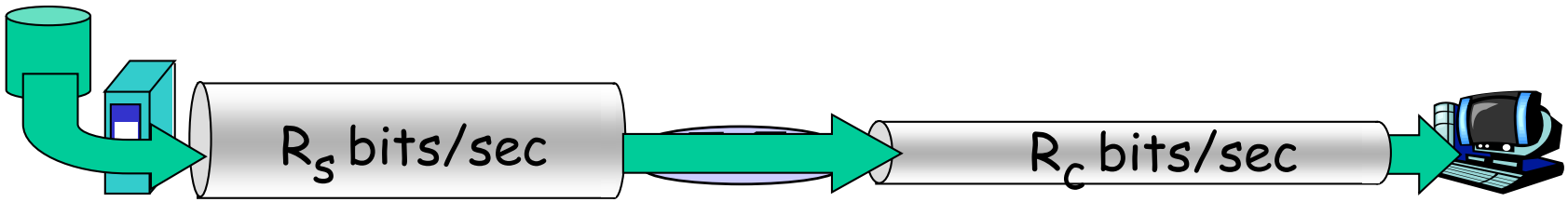


端到端吞吐量 (续)

- $R_s < R_c$ What is average end-end throughput?



- $R_s > R_c$ What is average end-end throughput?

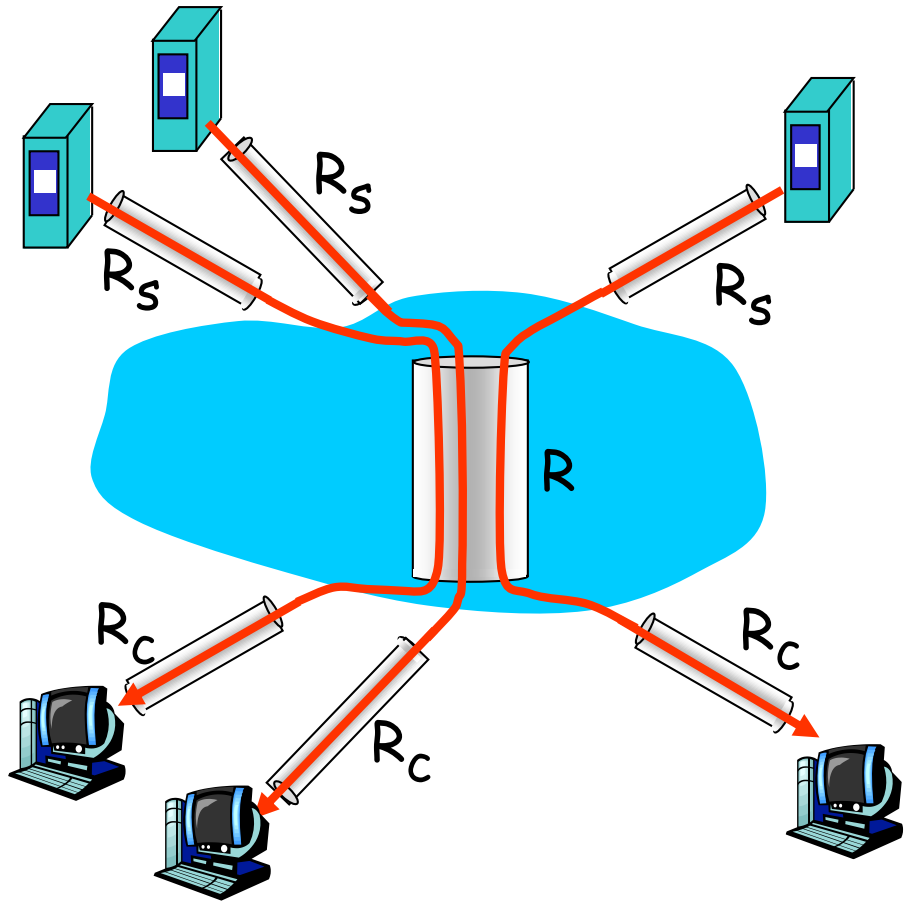


bottleneck link

瓶颈链路的带宽限制了端到端吞吐量。

Throughput: Internet scenario

- 端到端吞吐量: $\min(R_c, R_s, R/10)$
- 端到端吞吐量与瓶颈链路的速率、以及链路上的负载有关



10 connections (fairly) share backbone bottleneck link R bits/sec

小结

- 延迟、丢包率、吞吐量三个指标，均与负载有关
- 如何通过调节负载来获得这些指标的平衡，是因特网的重要研究内容之一

Chapter 1: roadmap

1.1 What *is* the Internet?

1.2 Network edge

- end systems, access networks, links

1.3 Network core

- circuit switching, packet switching, network structure

1.4 Delay, loss and throughput in packet-switched networks

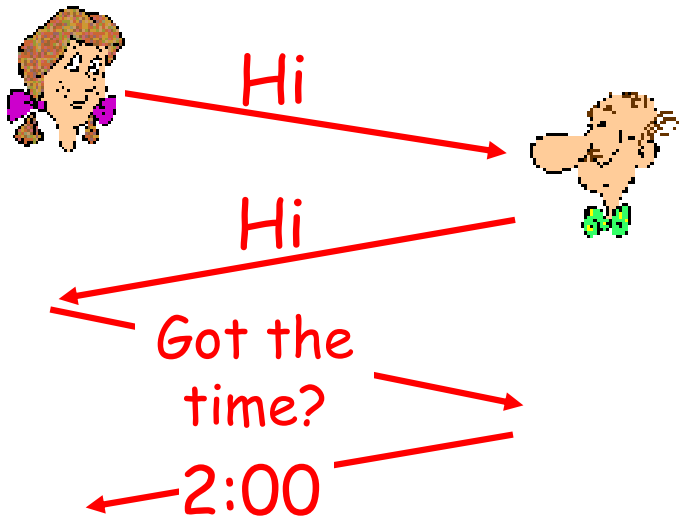
1.5 Protocol layers, service models

1.6 Networks under attack: security

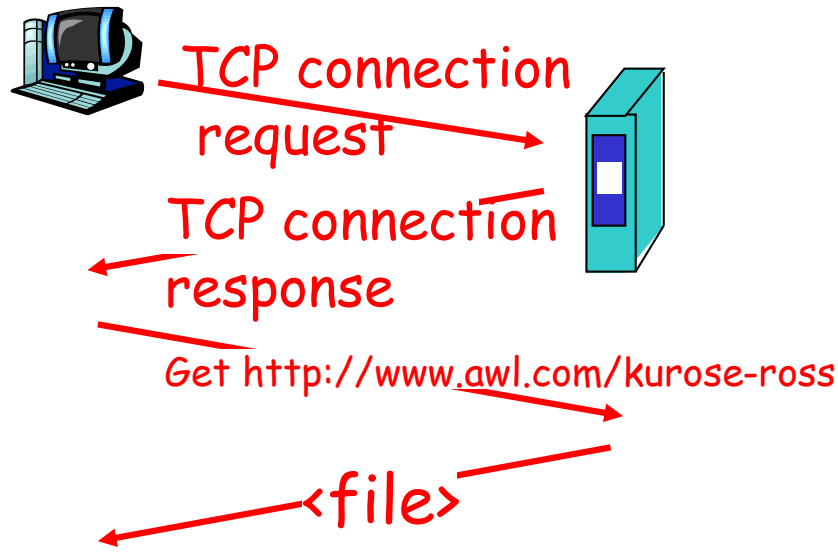
1.7 History

什么是协议？

一个人类协议



一个计算机网络协议



协议的要素

- Human/network protocols:
  - specific mesages sent
  - specific actions taken when messages received, or other events

- 网络协议定义了：
  - 通信实体之间交换的报文的格式和次序
  - 在发送/接收报文、或其它事件后采取的动作

- 掌握计算机网络知识的过程，就是理解网络协议的构成、原理和工作的过程

Networks are complex !

- many "pieces":
  - hosts
  - routers
  - links of various media
  - applications
  - Protocols

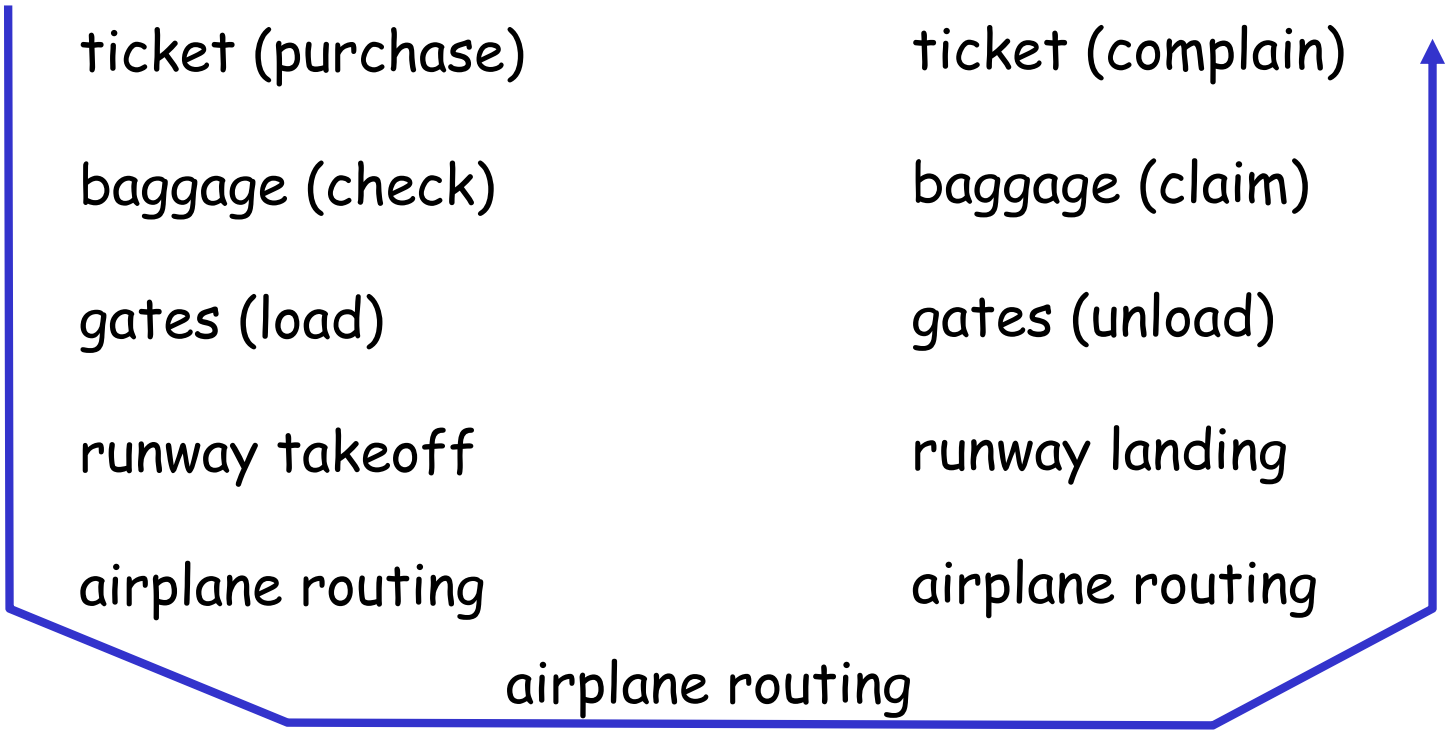
Question:

Is there any hope of *organizing* structure of network?

Or

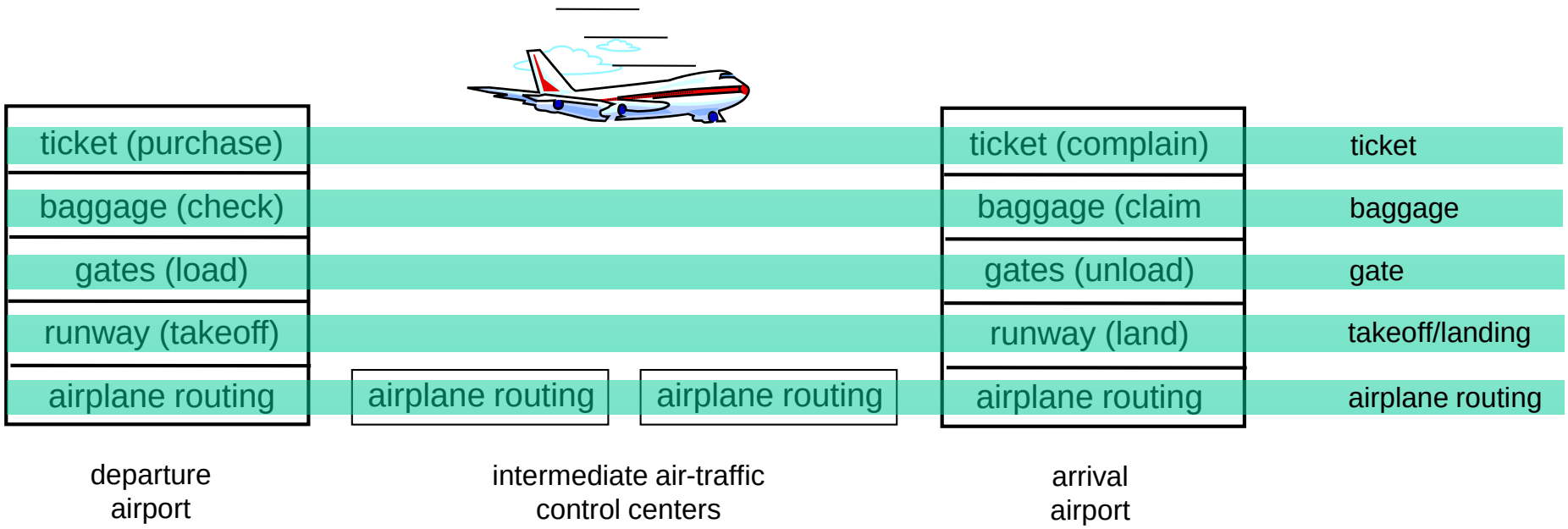
at least our discussion of networks?

Organization of air travel



- a series of steps

Layering of airline functionality



- 系统分层：将系统按功能划分成一系列水平的层次，每一层实现一个功能（服务）
- 层次间关系：每一层的功能实现都要依赖其下各层提供的服务

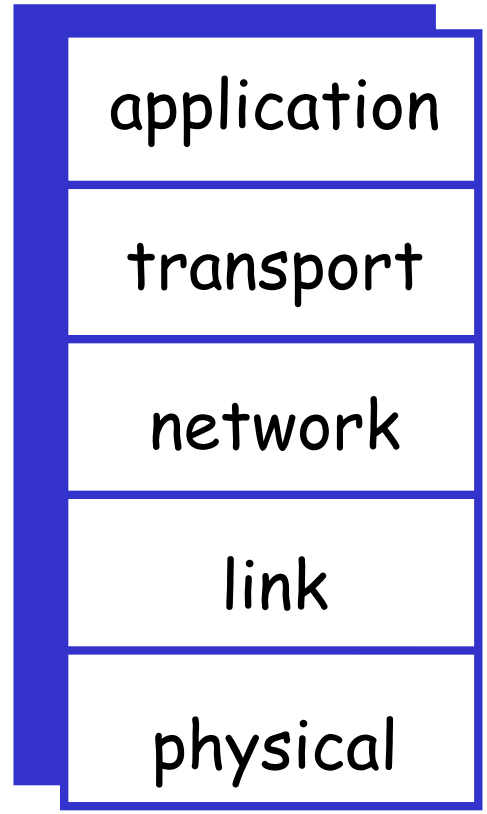
分层的好处

系统分层：易于处理复杂的系统

- 显式的层次结构易于确定系统的各个部分及其相互关系
- 模块化简化了系统的维护和升级
  - 改变某层服务的实现方式，对于其它层次是透明的

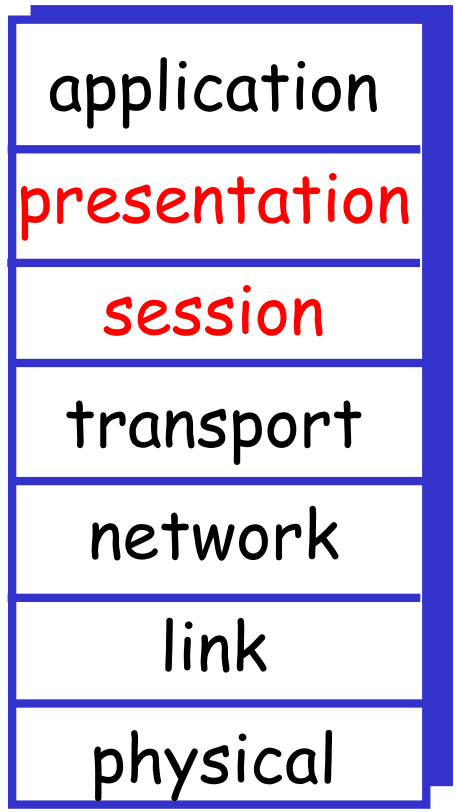
Internet协议栈

- application: 在应用程序之间传输应用特定的报文（message）
  - E.g., FTP, SMTP, HTTP
- transport: 在应用程序与网络的接口间（进程-进程）传输报文段（segment）
  - TCP, UDP
- network: 在源主机和目的主机（终端-终端）之间传输分组（packet）
  - IP, routing protocols
- link: 在相邻设备之间传输帧（frame）
  - E.g., PPP, Ethernet
- physical: 在物理媒体上传输比特（bit）



ISO/OSI reference model

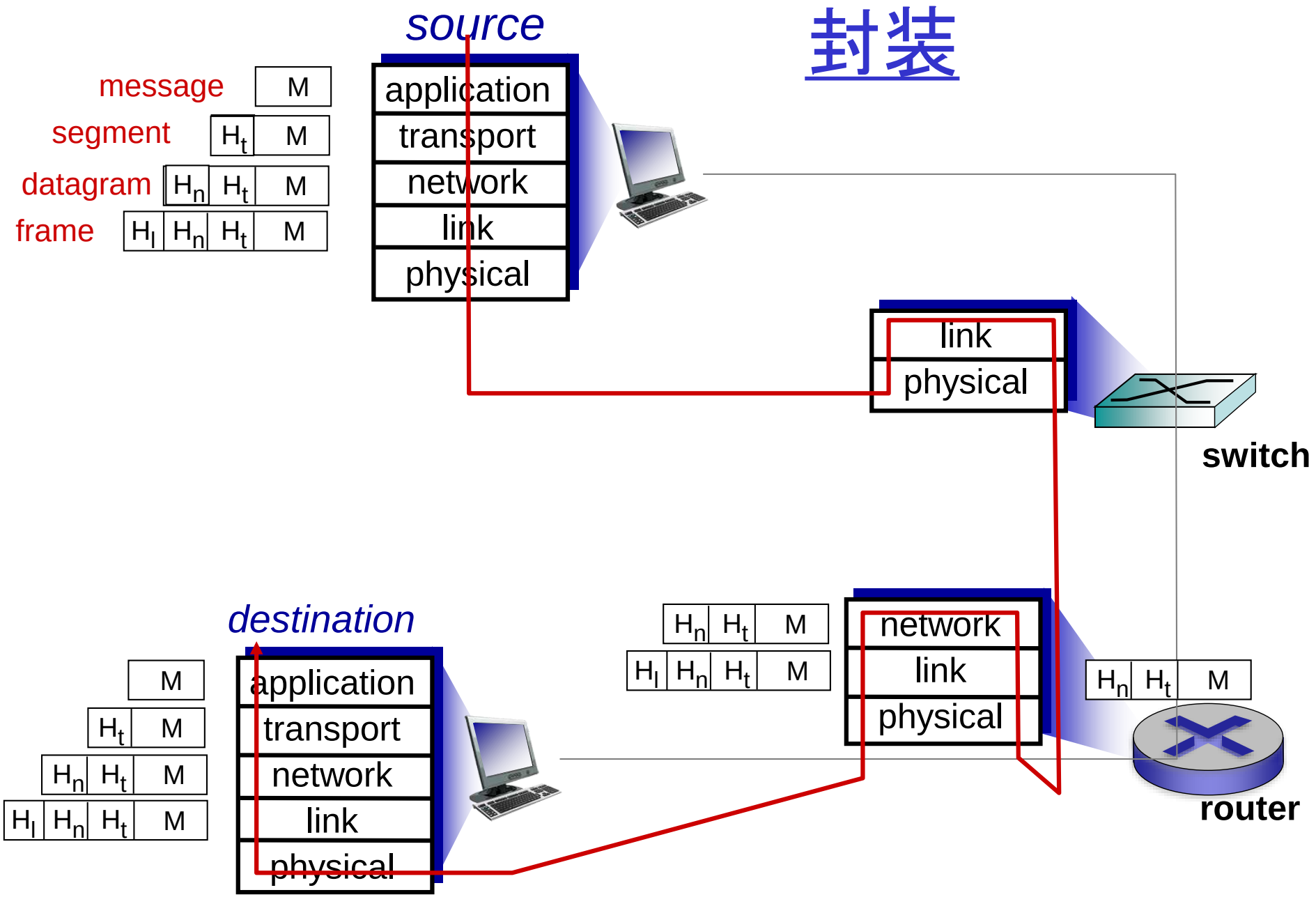
- presentation: allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack "missing" these layers!
  - these services, *if needed*, must be implemented in application
  - needed?



网络功能的分布式实现

- 某一层上的网络功能，需要该层上的实体（分布在不同的节点）协同完成
- 协同计算要求功能实体之间能够交互信息，需要解决以下问题：
  - 信息交互的载体是什么：各层上的报文
  - 信息交互的约定：报文格式及语义规定
  - 报文的传输方式：封装和解封装

封装


source
message
M
segment
H_t
M
datagram
H_n
H_t
M
frame
H_l
H_n
H_t
M
application
transport
network
link
physical
link
physical
switch
destination
M
H_t
M
H_n
H_t
M
H_l
H_n
H_t
M
application
transport
network
link
physical
H_n
H_t
M
H_l
H_n
H_t
M
network
link
physical
H_n
H_t
M
router

小结

- 网络按功能划分层次，每层实现一个功能
- 在不同系统的同一层上：
  - 对等实体执行该层的协议
- 相同系统的上下层：
  - 调用服务/提供服务
  - 封装/解封装分组
- 不同系统的不同层：
  - 不直接通信

Chapter 1: roadmap

1.1 What *is* the Internet?

1.2 Network edge

- end systems, access networks, links

1.3 Network core

- circuit switching, packet switching, network structure

1.4 Delay, loss and throughput in packet-switched networks

1.5 Protocol layers, service models

1.6 Networks under attack: security

1.7 History

Network Security

- Internet最初设计的时候，并没有考虑安全的问题
  - *original vision*: "a group of mutually trusting users attached to a transparent network" ☺

- 今天的Internet：
  - 每天都有各种各样的安全事件发生
  - 网络安全已成为近年来计算机网络领域的中心主题

因特网面临的安全威胁

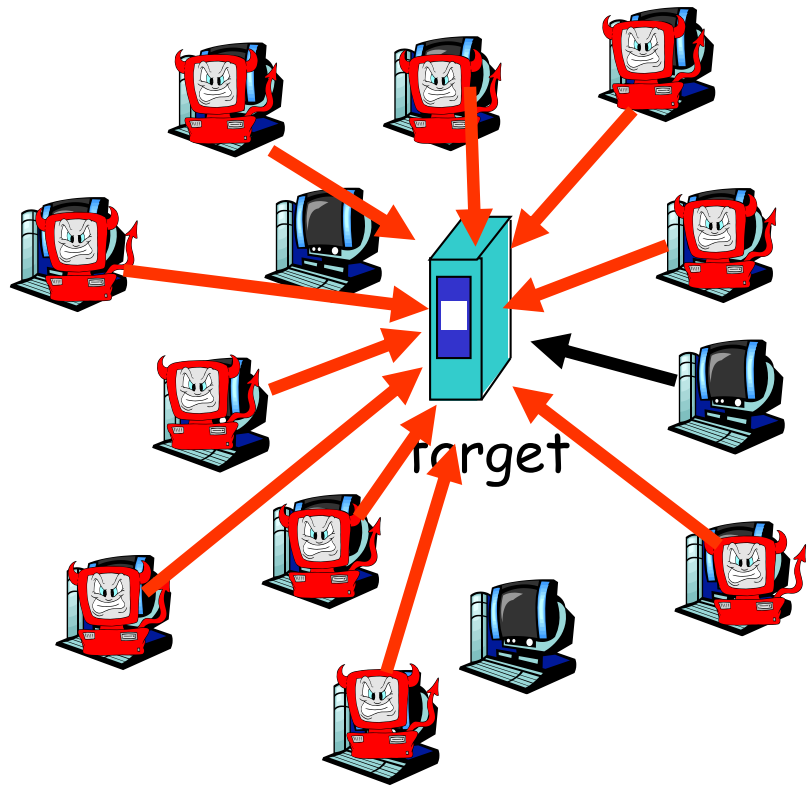
- 针对因特网基础设施的攻击：
  - 恶意软件（如病毒、蠕虫）入侵计算机设备
  - 对主机、网络等实施拒绝服务攻击（Denial of Service），使其中止服务

- 针对因特网中信息的攻击：
  - 窃听网络中传输的数据
  - 在网络中注入虚假的信息欺骗用户

拒绝服务（DoS）攻击

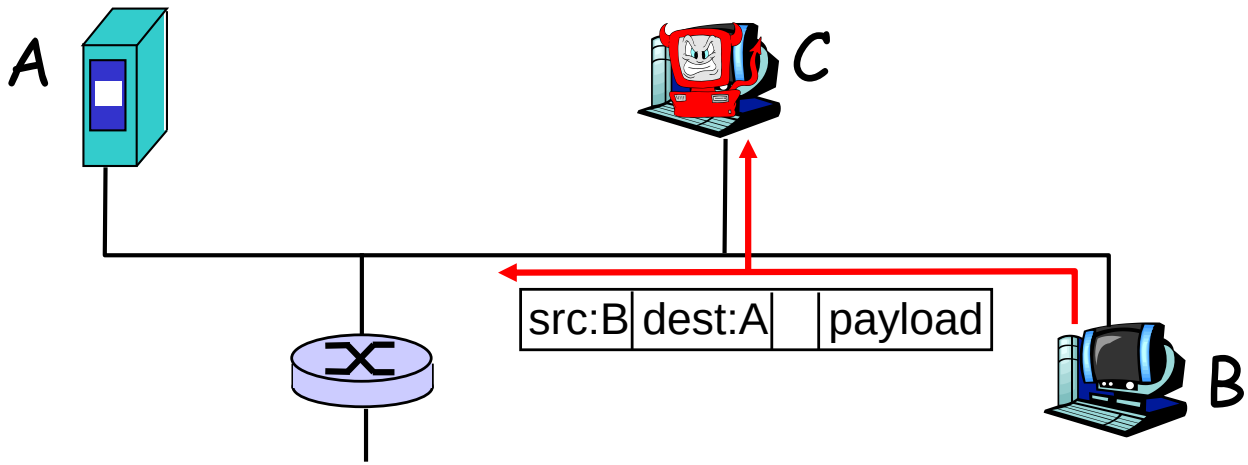
- 攻击者通过耗尽主机或网络带宽资源，使得合法用户得不到所需的服务

1. 选择目标
2. 利用恶意软件攻陷网络中的主机（称肉鸡、僵尸机器）
3. 从僵尸主机向目标发送大量数据包



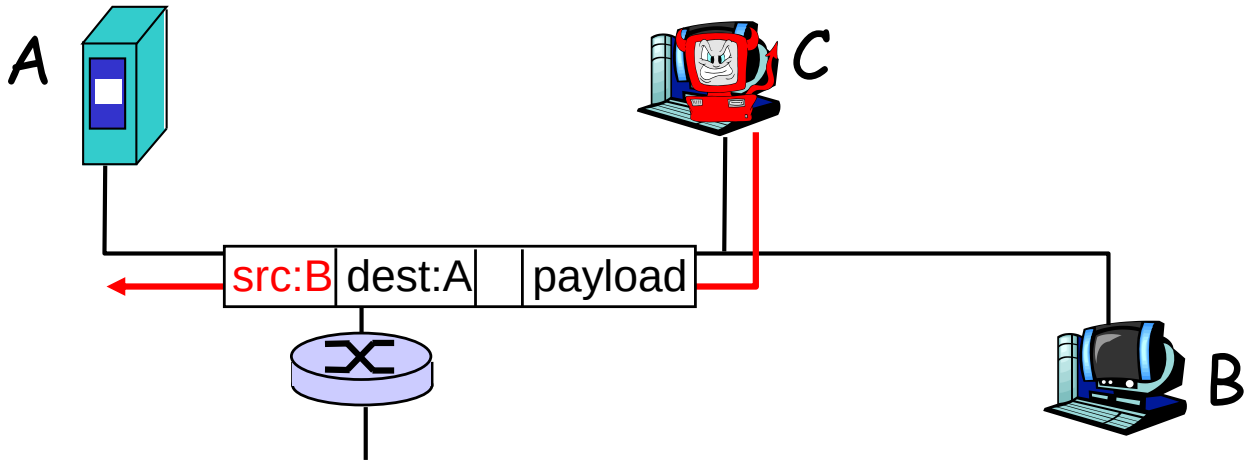
嗅探

- 嗅探:
  - 监听网络中传输的数据包，获取数据包中携带的信息，如密码
- Whireshark就是一款免费的嗅探器



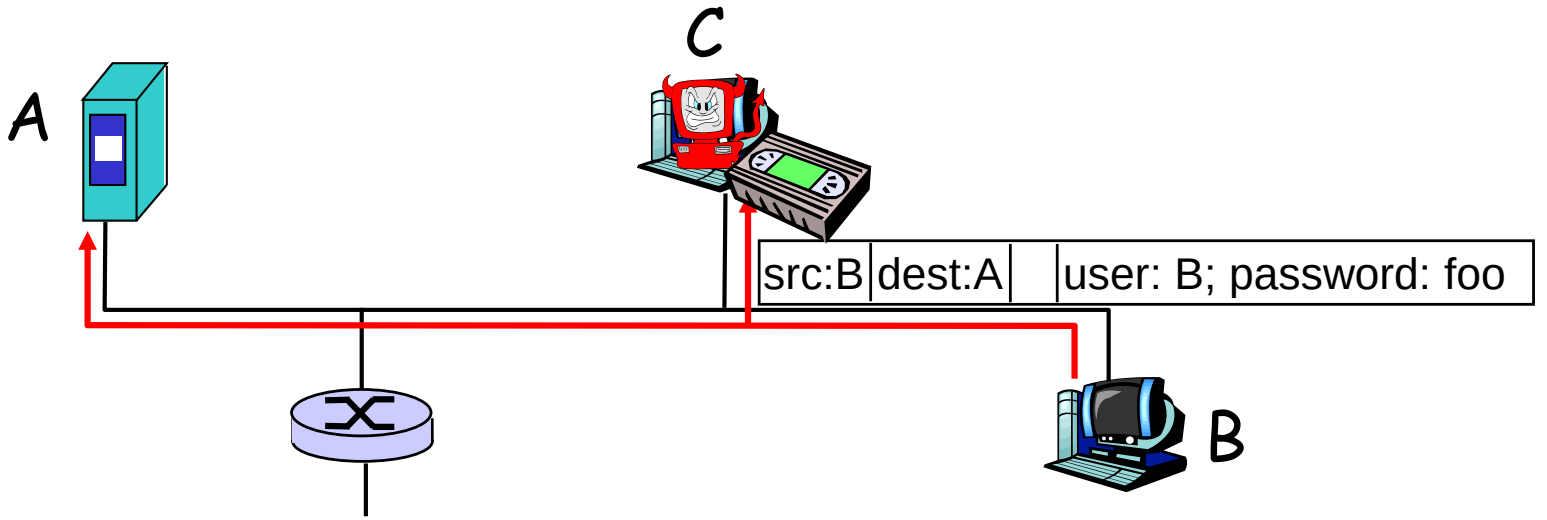
伪装

- IP欺骗：发送虚假地址的数据包

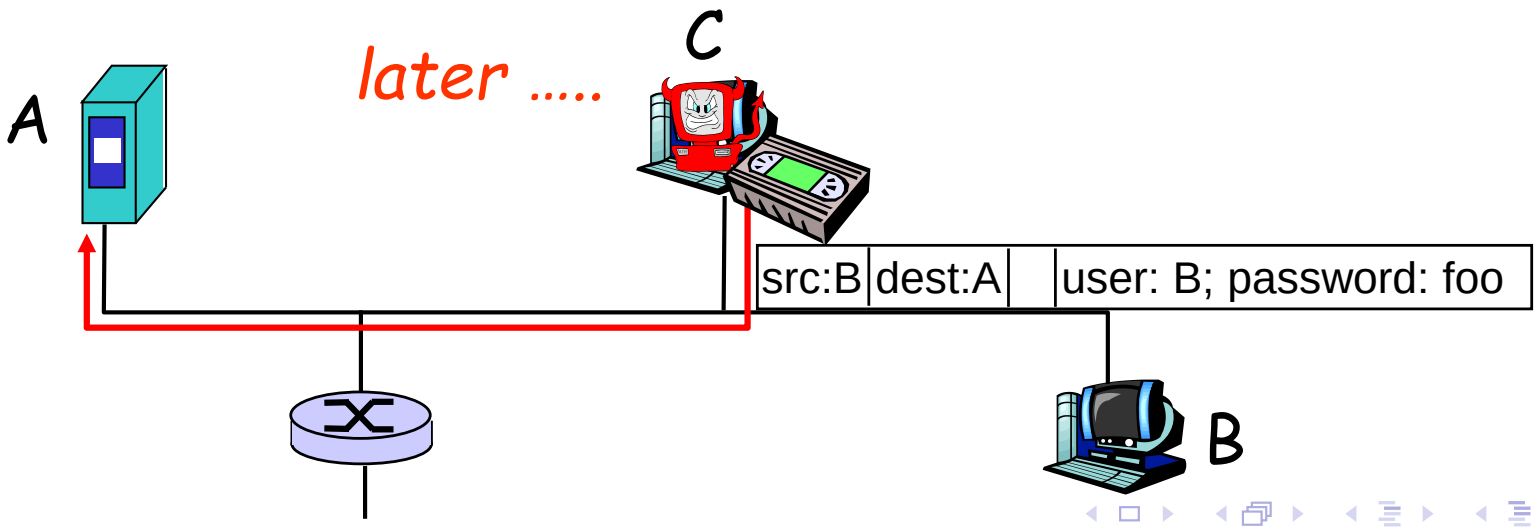


伪装

- 重放攻击: 嗅探敏感信息（比如，某用户的口令），之后重新注入网络（以假冒该用户）



later .....



Chapter 1: roadmap

1.1 What *is* the Internet?

1.2 Network edge

- end systems, access networks, links

1.3 Network core

- circuit switching, packet switching, network structure

1.4 Delay, loss and throughput in packet-switched networks

1.5 Protocol layers, service models

1.6 Networks under attack: security

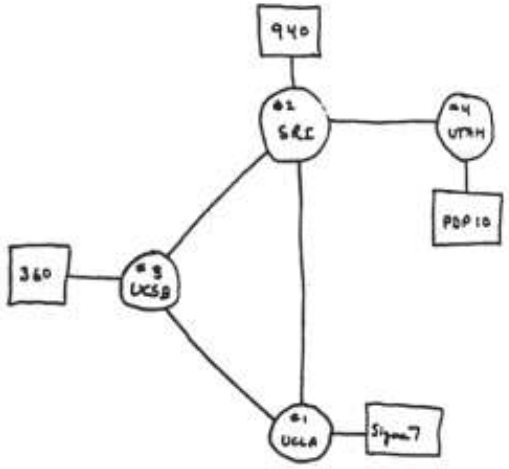
1.7 History

Internet History

1961-1972: Early packet-switching principles

- 1961: Kleinrock - queueing theory shows effectiveness of packet-switching
- 1964: Baran - packet-switching in military nets
- 1967: ARPAnet conceived by Advanced Research Projects Agency
- **1969: first ARPAnet node operational**

- 1972:
  - ARPAnet public demonstration
  - NCP (Network Control Protocol) first host-host protocol
  - first e-mail program
  - ARPAnet has 15 nodes



THE ARPA NETWORK

Internet History

1972-1980: Internetworking, new and proprietary nets

- 1970: ALOHAnet satellite network in Hawaii
- 1974: Cerf and Kahn - architecture for interconnecting networks
- 1976: Ethernet at Xerox PARC
- ate70's: proprietary architectures: DECnet, SNA, XNA
- late 70's: switching fixed length packets (ATM precursor)
- 1979: ARPAnet has 200 nodes

Cerf and Kahn's internetworking principles:

- minimalism, autonomy - no internal changes required to interconnect networks
- best effort service model
- stateless routers
- decentralized control

define today's Internet architecture

Internet History

1980-1990: new protocols, a proliferation of networks

- **1983: deployment of TCP/IP**
- 1982: smtp e-mail protocol defined
- 1983: DNS defined for name-to-IP-address translation
- 1985: ftp protocol defined
- 1988: TCP congestion control
- new national networks: Csnet, BITnet, **NSFnet**, Minitel
- 100,000 hosts connected to confederation of networks

Internet History

1990, 2000's: commercialization, the Web, new apps

- Early 1990's: ARPAnet decommissioned
- **1991: NSF lifts restrictions on commercial use of NSFnet (decommissioned, 1995)**
- **early 1990s: Web**
  - hypertext [Bush 1945, Nelson 1960's]
  - HTML, HTTP: Berners-Lee
  - 1994: Mosaic, later Netscape
  - late 1990's: commercialization of the Web

Late 1990's – 2000's:

- more killer apps: **instant messaging, P2P file sharing**
- network security to forefront
- est. 50 million host, 100 million+ users
- backbone links running at Gbps

Internet History

2001~: Mobile Internet, Internet of Things

- 2008年开启移动互联网时代:
  - 国际电信联盟正式公布第三代移动通信标准（3G）
  - 苹果公司发布iPhone3G、iOS 2.0，推出App Store
- 移动互联网：
  - 通过智能移动终端，从互联网获取业务和服务
- 2009开启物联网时代：
  - 物联网计划在欧洲、美国（智慧地球）、中国（感知中国）启动
- 物联网：
  - 在互联网的基础上，将用户端扩展和延伸到任何物体，允许对任何能够被独立寻址的普通物体进行智能化识别、定位、跟踪、监控和管理

Internet history

- ~5G devices attached to Internet (2016)
  - smartphones and tablets
- aggressive deployment of broadband access
- increasing ubiquity of high-speed wireless access
- emergence of online social networks:
  - Facebook: ~ one billion users
- service providers (Google, Microsoft) create their own networks
  - bypass Internet, providing "instantaneous" access to search, video content, email, etc.
- e-commerce, universities, enterprises running their services in "cloud" (e.g., Amazon EC2)

本章小结

- 计算机网络关注：
  - 功能性问题：可达性（选路、转发），正确性（可靠性控制）
  - 性能问题：吞吐量，延迟，丢包率
  - 安全性问题：基础设施安全，信息安全

- 重点理解：
  - 分组交换：存储转发，设计考虑（利），引入的问题（弊）

  说明：在因特网中使用分组交换而不是电路交换，是一个重大的决定。需要理解这个决定背后的考虑，以及这些决定可能带来的问题，如何解决这些问题则是协议的主要内容
  - 网络分层架构：服务，功能，接口，协议，封装

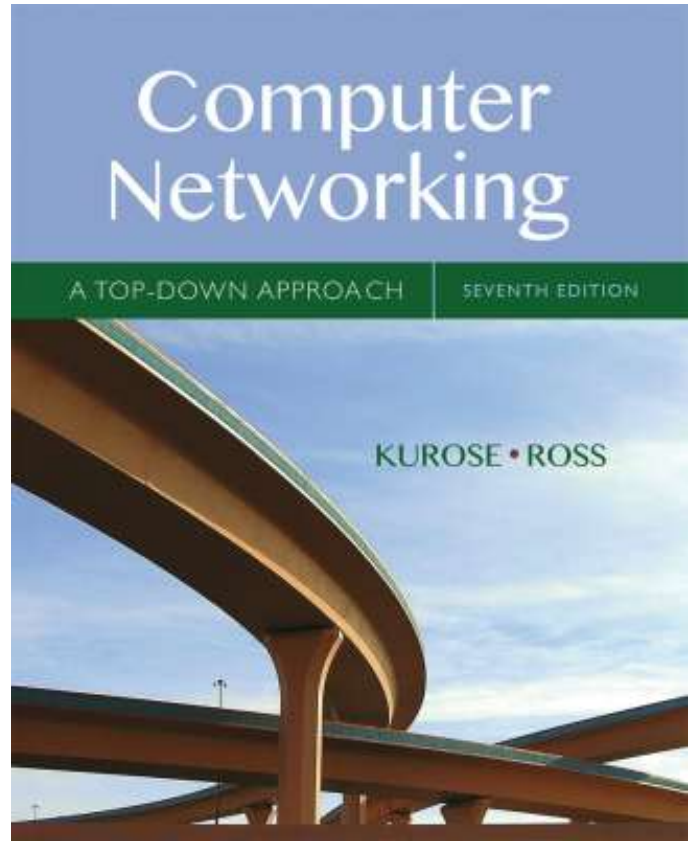
作业

- 习题：
  - 9，10，13，21，22，25，31，33
- 实验：
  - 入门实验
- 提交时间：
  - 习题：9月15日
  - 实验：9月17日

- 思考与讨论：

理论上说，由于因特网采用分组交换，不能保证分组延迟，因此因特网对音视频应用的支持不好。这句话和你的亲身感受相符吗？回想一下，你在网络上看电影、看直播、打电话有什么样的体会。如果不符，举出例子；如果相符，也举出例子。对此你有什么疑问或思考吗？

Chapter 2
Application Layer



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Chapter 2: Application Layer

Our goals:

- 学习网络应用协议的原理及实现方面的知识
- 几个重要的概念：
  - 客户-服务器模式
  - 对等模式
  - 传输层服务
  - 进程与传输层接口
- 几个流行的应用层协议：
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- 开发网络应用程序：
  - socket API

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

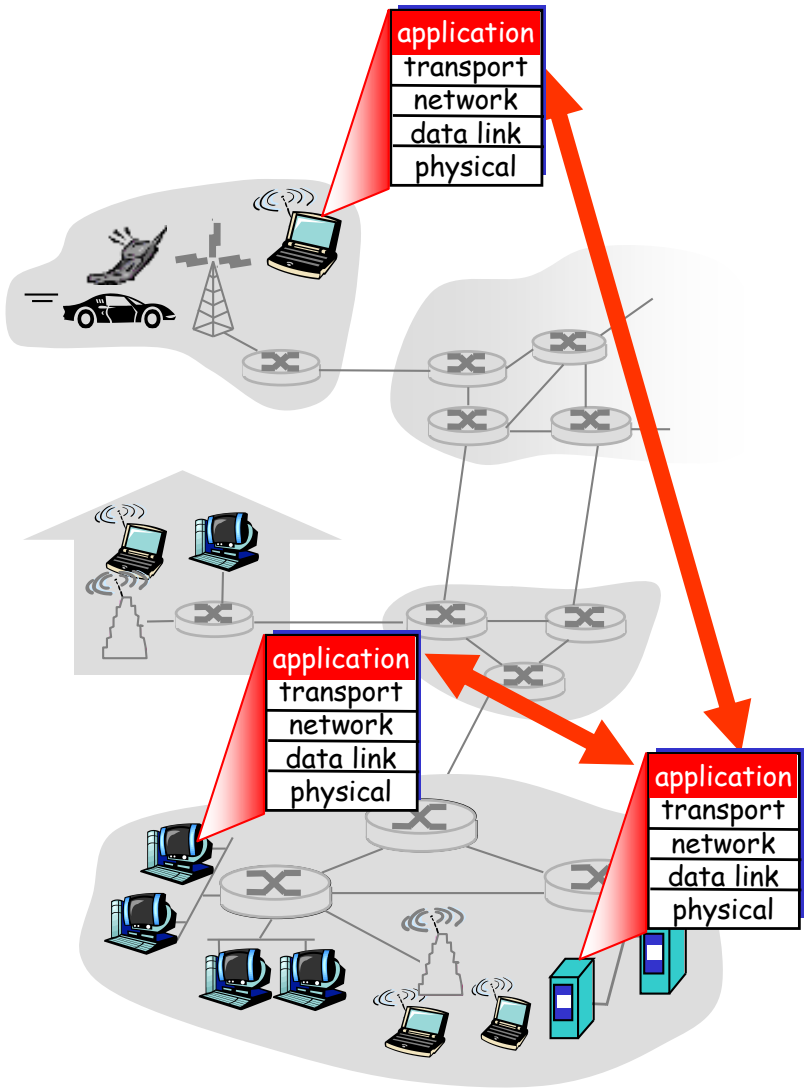
2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

创建一个网络应用

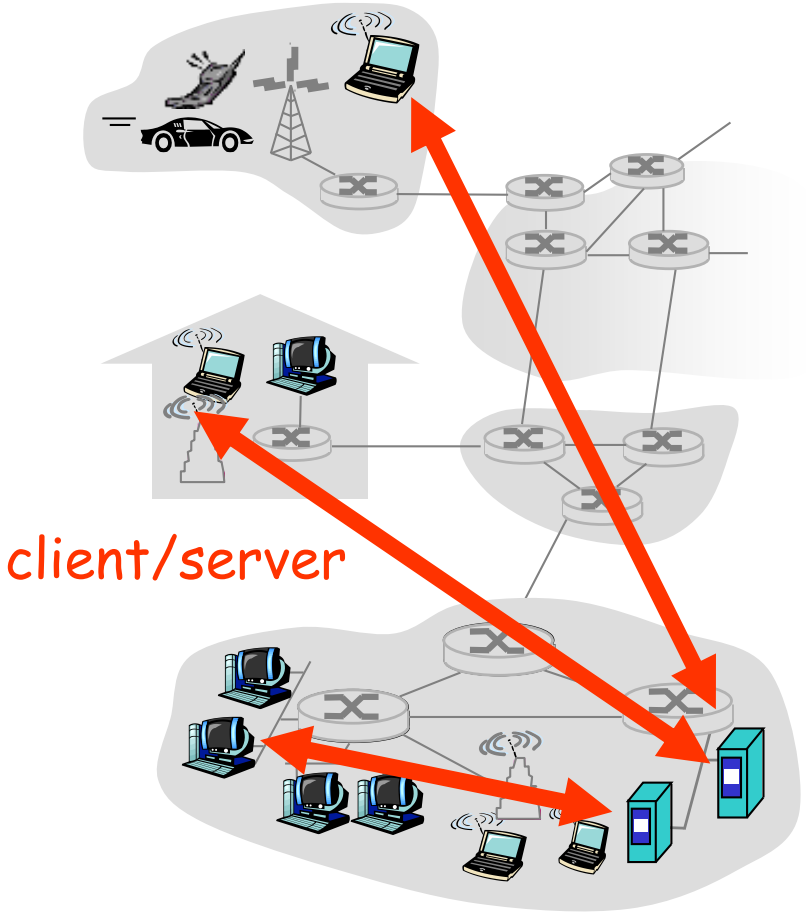
- 创建一个网络应用的核心，是编写一个分布式程序，使其可以：
  - 运行在不同的端系统上，并能通过网络相互通信
  - 例如，web服务器软件与浏览器软件
- 应用程序只运行在终端上：
  - 应用程序员不需要为网络核心设备编写程序
- "只在端系统上开发应用"是一个非常重要的设计决定：
  - 极大地方便了网络应用的开发和快速部署



2.1.1 网络应用架构

- 在开发一个应用程序之前，首先要决定采用什么样的网络应用架构
- 网络应用架构规定了在各个端系统上组织应用程序的方法
- 目前有两种主流的网络应用架构：
  - 客户-服务器架构（Client-server）
  - 对等架构（Peer-to-peer，P2P)

客户-服务器架构



服务器（server）：

- 有一台总是在线的主机，上面运行着服务器程序(server)
- 服务器主机(server machine)具有永久的、众所周知的地址

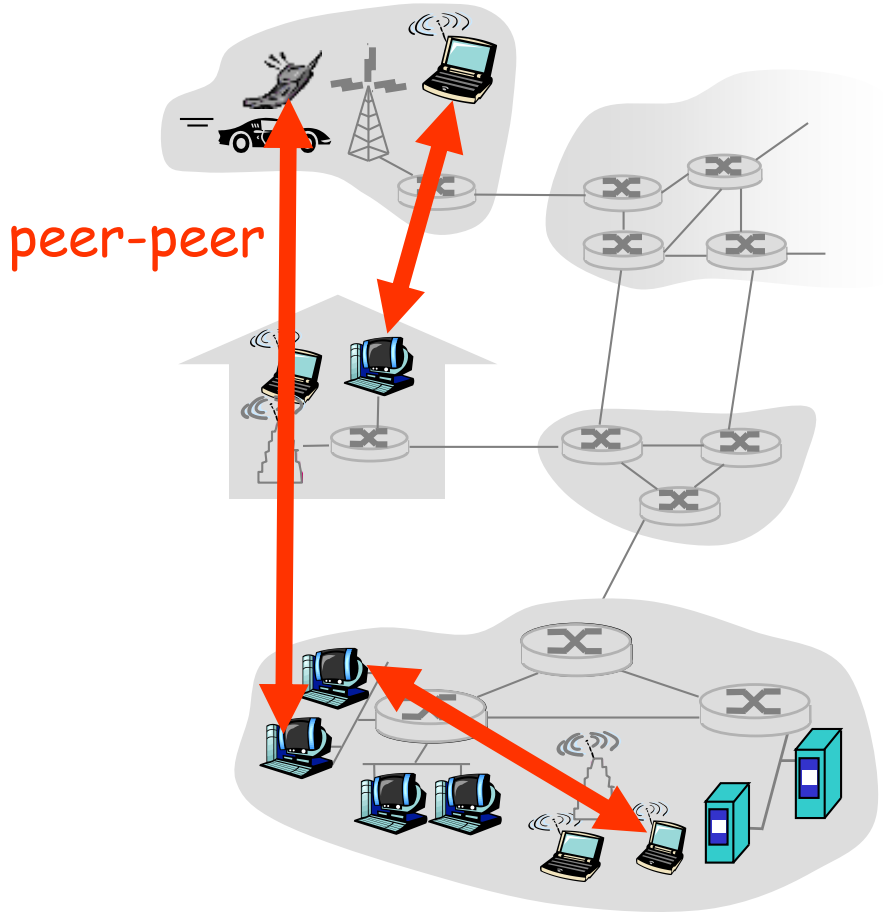
客户（client）：

- 用户终端上运行一个客户程序(client)，需要时与服务器程序通信，请求服务
- 客户机（client machine）使用动态地址，通常不会总是在线

客户只与服务器通信，客户之间不通信

P2P架构

- 没有总是在线的服务器主机
- 任意一对端系统（对等方）可以直接通信
- 对等方多为用户自己的计算机，使用动态地址
- 每个对等方既可请求服务，也可提供服务
- 典型的P2P应用：
  - BT、迅雷
  - Skype
  - PPLive



两种架构的比较

客户-服务器架构：

- 资源集中：
  - 资源（服务）只在某些固定的终端上提供
- 优点：
  - 资源发现简单
- 缺点：
  - 集中式计算带来的问题，如服务端扩容压力、网络流量不均衡、响应延迟长

P2P架构：

- 资源分散：
  - 任何终端都可以提供资源（服务）
- 优点：
  - 易于扩容、均衡网络流量
- 缺点：
  - 资源发现困难，社会问题（版权、安全性等）

2.1.2 不同终端上的进程通信

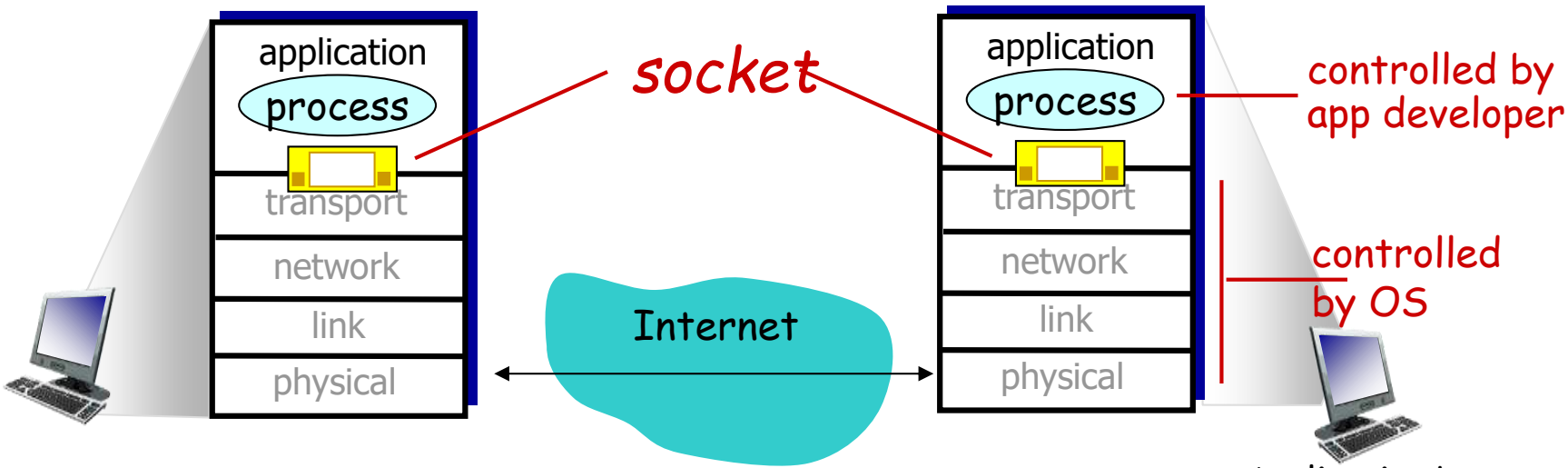
- 进程: 主机上运行的程序
- 在分布式应用中，不同终端上的进程需要通信

- 进程通信的方法：
  - 同一个主机内：进程间通信机制（OS提供）
  - 在不同主机上：通过交换报文进行通信

- 在一次确定的通信会话中，总能标示一方为客户进程，另一方为服务器进程：
  - 客户进程：主动发起请求的进程
  - 服务器进程：接收请求的进程

进程与网络的接口：套接字（socket）

- 进程如何将报文送入网络，又如何从网络接收报文？
- 设想在应用程序和网络之间存在一扇门（套接字）：
  - 发送报文：发送进程将报文推到门外
  - 门外的运输设施（因特网）将报文送到接收进程的门口
  - 接收报文：接收进程打开门，即可收到报文
- 套接字是应用层和传输层的接口，也是应用程序和网络之间的*API*



进程如何标识自己：进程编址

- 每个进程需要一个标识，以便其它进程能够找到它
- Q: 可以用进程所在主机的地址来标识进程吗？
- A: 不能，因为同一个主机上可能运行着许多进程
- 端口号：用于区分同一个主机上的不同进程
- 进程标识包括：
  - 主机地址
  - 与该进程关联的端口号
- 端口号的例子：
  - 众所周知的端口号分配给服务器，成为服务的标识
  - 比如，HTTP server使用端口80，Mail server使用端口25

2.1.3 应用需要什么样的传输服务?

在创建一个应用程序时，开发者需要选定一种传输服务

Data integrity

- 有些应用（如音视频）可以容忍一定程度的数据丢失
- 有些应用（如文件传输）要求完全可靠的数据传输

Throughput

- 有些应用（如多媒体）要求保证最低可用带宽
- 有些应用（称弹性应用）可以适应各种可能的带宽

Timing

- 有些应用（如网络电话，交互式网络游戏）要求延迟保证
- 有些应用（如邮件传输）对延迟不敏感

Security

- 加密，数据完整性，.....

2.1.4 因特网能够提供的传输服务

因特网提供2种传输服务，分别用TCP和UDP协议实现

TCP service:

- 面向连接: 保证传输顺序
- 可靠传输：不出错
- 流量控制: 发送进程不会“压垮”接收进程
- 拥塞控制: 网络超载时抑制发送进程
- **不提供:** 及时性，最低带宽保证，安全性

UDP service:

- 通过因特网接收和发送报文
- **不提供:** 顺序保证，可靠传输，流量控制，拥塞控制，及时性，最低带宽保证，安全性

2.1.5 应用层协议

应用进程之间交换的报文是什么样的？

应用层协议定义：

- 报文类型：
  - e.g., request, response
- 报文语法：
  - 报文中的字段及其描述
- 报文语义
  - 各字段中信息的含义
- 发送/响应报文应遵循的规则

公共域协议：

- 在RFC文档中定义
- 允许互操作
- e.g., HTTP, SMTP

专用协议：

- e.g., Skype

小结

- 为创建一个新的网络应用，需要：
  - 选择一种网络应用架构：客户-服务器 or P2P
  - 选择一种网络服务：TCP or UDP
  - 确定一个端口号
  - 定义应用层协议
  - 编写客户程序和服务器程序（调用套接字接口）
- 网络应用和应用层协议：
  - 应用层协议只是网络应用的一部分
  - 网络应用还包括客户程序、服务器程序等
- 要求掌握的概念：
  - 客户-服务器模型
  - 网络应用编程接口

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

Web应用画像

- 应用层资源：网页（web pages）
- 网络应用架构：客户-服务器架构
  - 客户：浏览器
  - 服务器：web服务器
- 要求的网络服务：TCP服务
- 端口号：80
- 应用层协议：HTTP（Hyper Text Transfer Protocol）

一些术语

- Web page由一些对象（object）组成
- 对象简单来说就是文件，可以是HTML文件、图像、Java小程序、音频文件，...
- 每个对象通过一个URL (Uniform Resource Locator) 获取，例如：

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

method　　host name　　path name

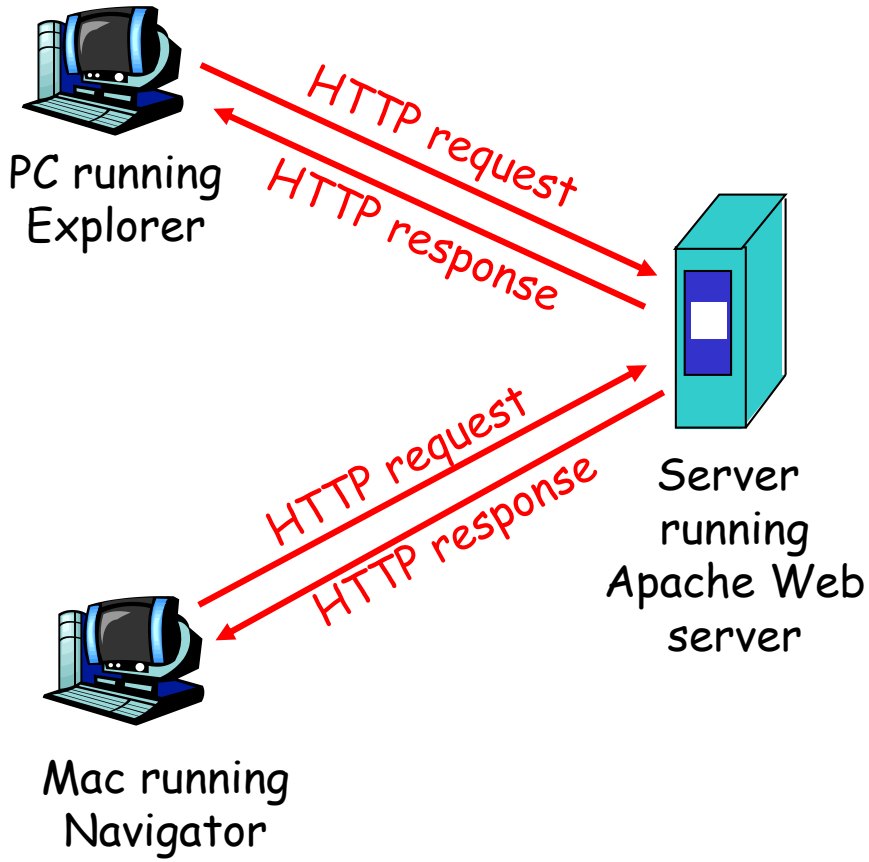
- HTML（Hyper Text Markup Language）是一种格式语言，用于描述页的内容和显示方式
- HTML文件是用HTML语言编写的超文本文件，需要通过专门的浏览器软件进行展示
- Web页通常包含一个HTML基本文件和若干引用对象，每个引用对象在文档中用其URL给出

一些术语（续）

- 超文本文件：
  - 包含超链接的文件，超链接中的文字包含有连接到文档中其他位置或者其它文档的链接，允许从当前阅读位置直接切换到超链接所指向的位置
- 超文本（Hyper text）：
  - 一种信息组织方式，通过超链接将各种不同空间的文字信息组织成网状文本

2.2.1 超文本传输协议--HTTP 概述

- Web采用客户-服务器模式
  - *client:* 浏览器请求、接收和显示web对象
  - *server:* Web服务器应客户请求发送对象
- HTTP协议：定义了浏览器和web服务器之间的通信规则
- HTTP 1.0（RFC 1945）和HTTP 1.1（RFC 2068）



HTTP 使用TCP服务

- 客户发起到服务器 80 端口的 TCP 连接（客户端创建一个套接字）
- 服务器接受来自客户的TCP连接（服务器端创建一个套接字）
- 浏览器和服务器交换HTTP报文 （通过各自的套接字）
- 关闭TCP 连接（关闭各自的套接字）

HTTP是“无状态的”

- 服务器不保存有关客户请求的任何信息

aside

Protocols that maintain “state” are complex!

- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled

2.2.2 非持久连接和持久连接

非持久 HTTP

- 在一个TCP连接上最多发送一个对象
- HTTP/1.0 使用非持久连接

持久 HTTP

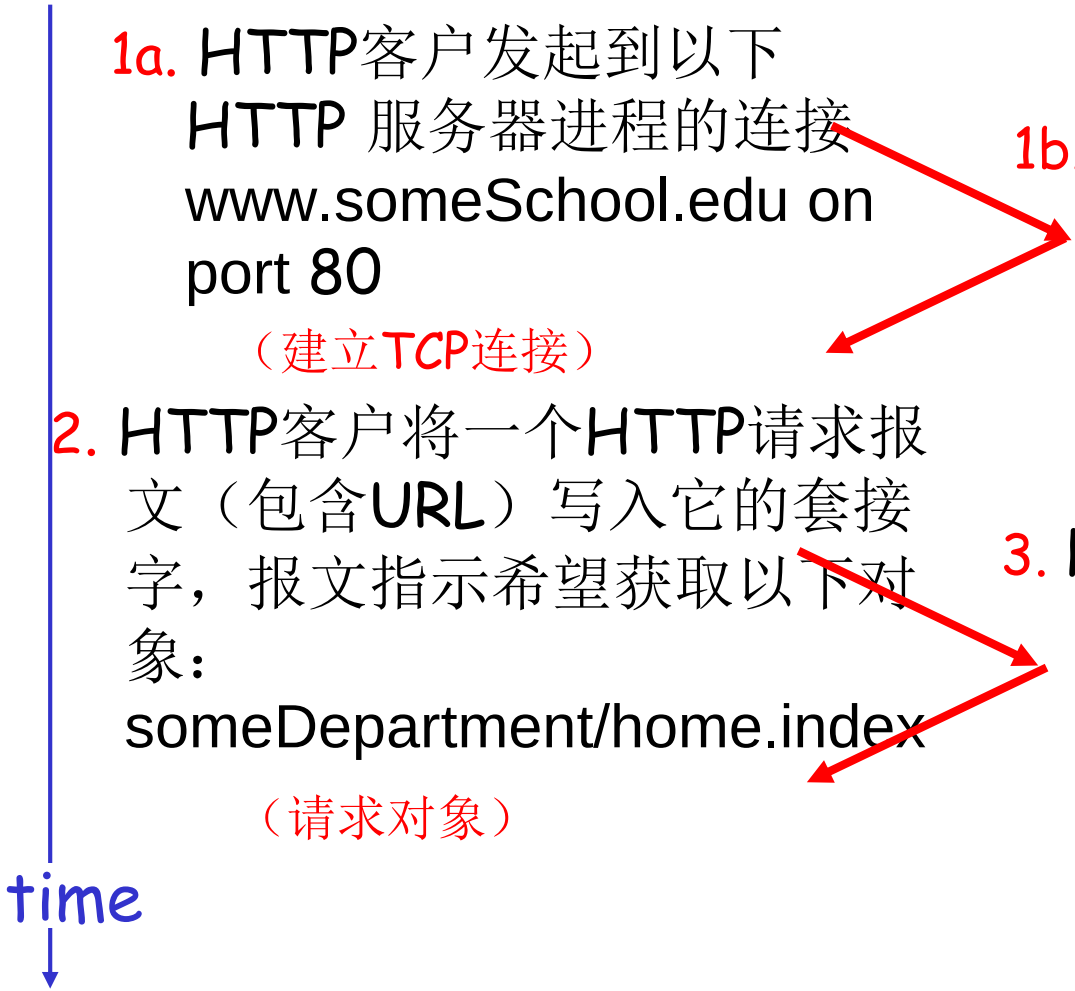
- 在一个TCP连接上可以发送多个对象
- HTTP/1.1 缺省使用持久连接

非持久 HTTP

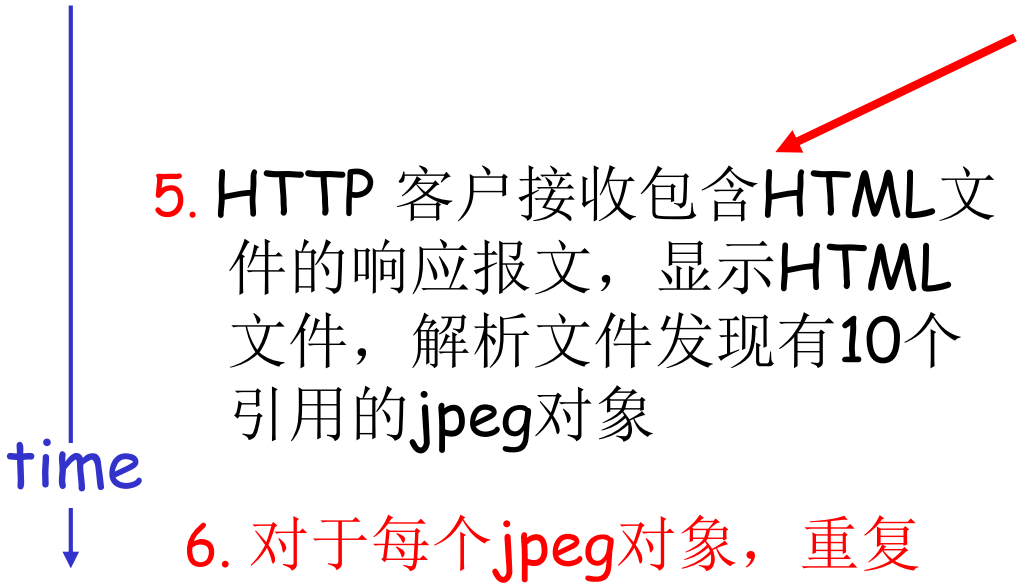
假设用户输入以下URL:

(包含文本及10个Jpeg图像)

http://www.someSchool.edu/someDepartment/home.index



非持久HTTP（续）


4. HTTP 服务器关闭TCP连接
5. HTTP 客户接收包含HTML文件的响应报文，显示HTML文件，解析文件发现有10个引用的jpeg对象
time
6. 对于每个jpeg对象，重复步骤 1-5

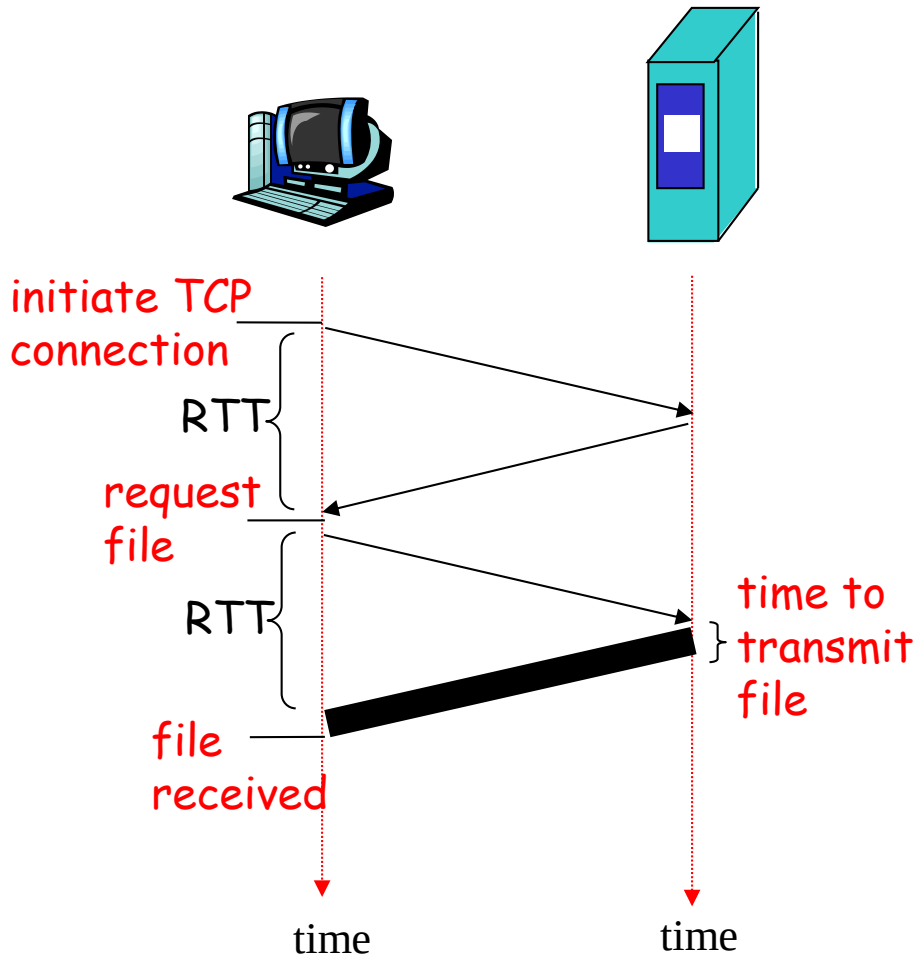
非持久HTTP的响应时间

RTT (Round-Trip Time):

一个小分组从客户发送到服务器，再返回客户的时间

Response time:

- 建立TCP连接用时一个RTT
- 发送HTTP请求至收到响应的前几个字节用时一个RTT
- 传输文件的时间



请求一个网页的时间：

- 请求一个对象用时2RTT（忽略对象传输时间）
- 请求一个完整的网页用时22RTT（11个对象）

持久 HTTP

非持久HTTP 的问题:

- 获取每个对象需要2个RTT
- 每个TCP连接需要消耗操作系统资源
- 浏览器需要打开多个TCP连接来获取一个网页

持久 HTTP

- 服务器在发送响应后保持连接
- 同一对客户-服务器之间的后续HTTP报文可以在该连接上传输

无流水线方式:

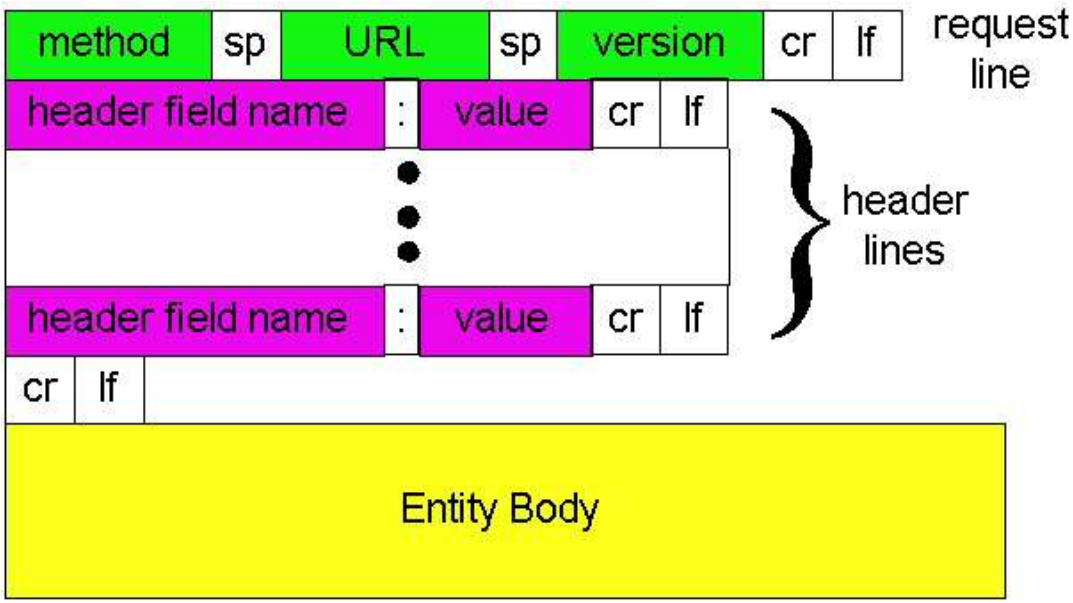
- 客户仅当收到前一个响应后再发送新的请求
- 请求每个对象用时1个RTT
- 请求一个网页用时12RTT

流水线方式:

- HTTP/1.1缺省使用该方式
- 客户每解析到一个引用对象就可以发送请求
- 可在一个RTT时间内请求所有引用对象
- 请求一个网页用时约3RTT

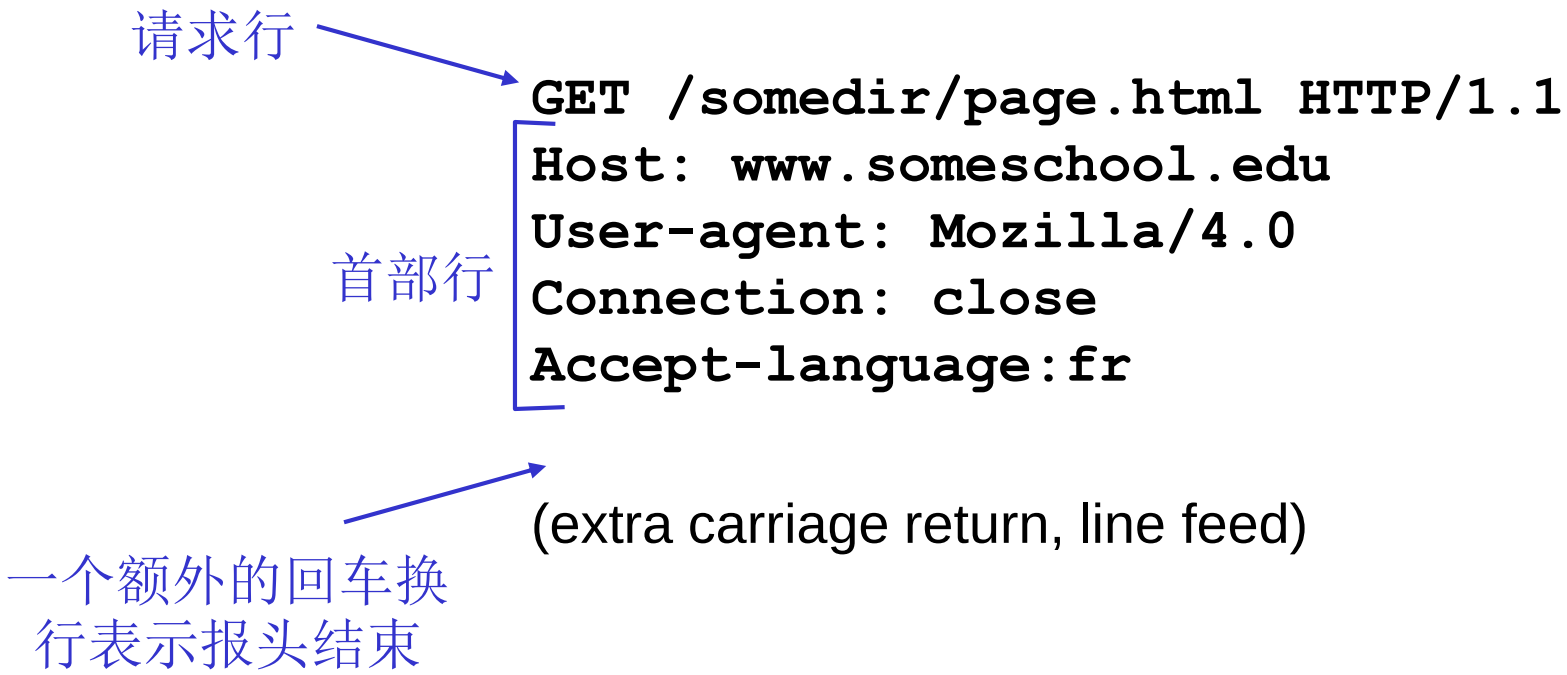
2.2.3 HTTP 报文格式

- 两类HTTP报文:
  - 请求报文：客户发往服务器
  - 响应报文：服务器发往客户
- HTTP报文由*ASCII*文本构成
- HTTP请求报文:



HTTP请求报文

- 报头包括：



上传表单输入

Post 方法:

- Web页通常包含表单输入
- 输入的表单内容放在报文体中上传到服务器

URL 方法:

- 使用 GET 方法
- 输入内容放在请求行的URL字段中上传，如：

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

HTTP方法

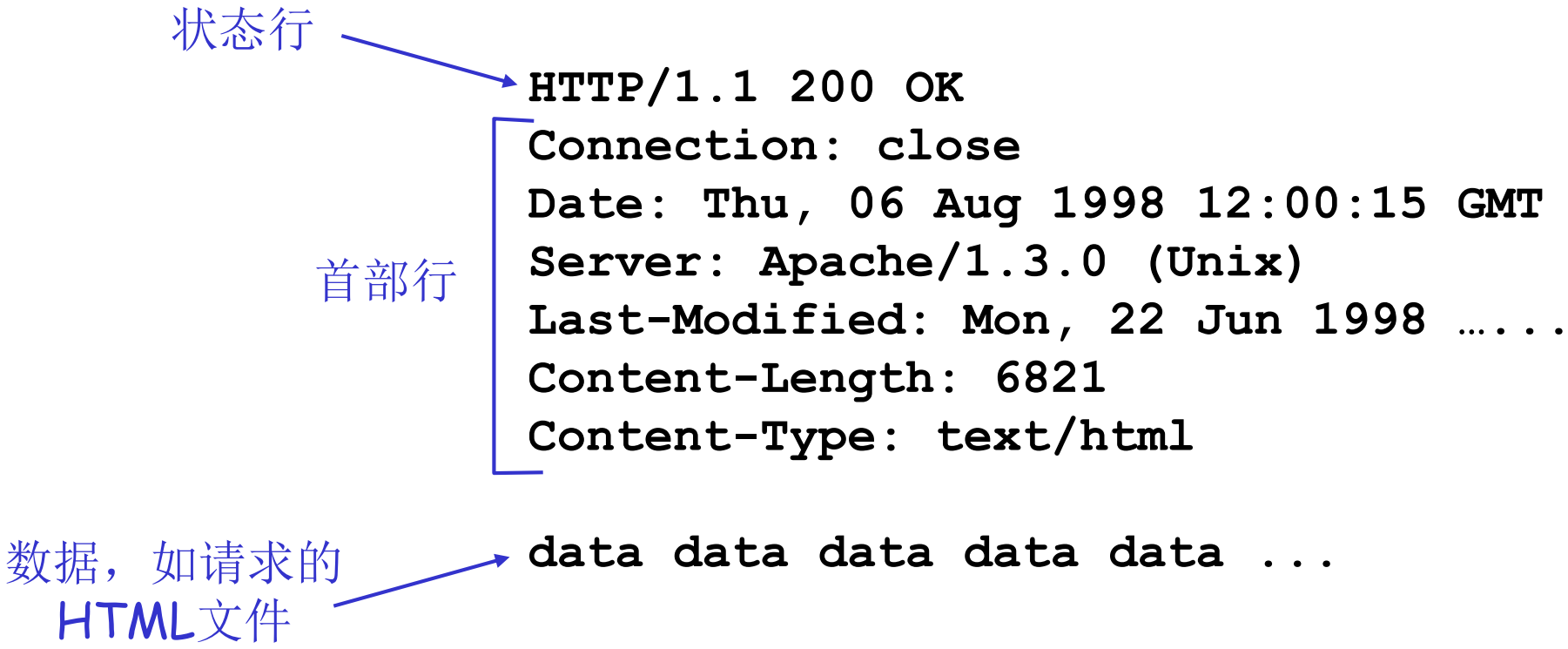
HTTP/1.0

- GET
- POST
- HEAD
  - 要求服务器不返回对象，只用一个报文头响应（实体为空），常用于故障跟踪。

HTTP/1.1

- GET, POST, HEAD
- PUT
  - 将文件放在报文实体中，传到URL字段指定的路径（目录）
- DELETE
  - 删除URL字段指示的文件

HTTP 响应报文


状态行
HTTP/1.1 200 OK
首部行
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html
数据，如请求的
HTML文件
data data data data data ...

HTTP 响应状态代码

A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

2.2.4 用户与服务器交互: cookie

许多大型的Web网站都使用cookie

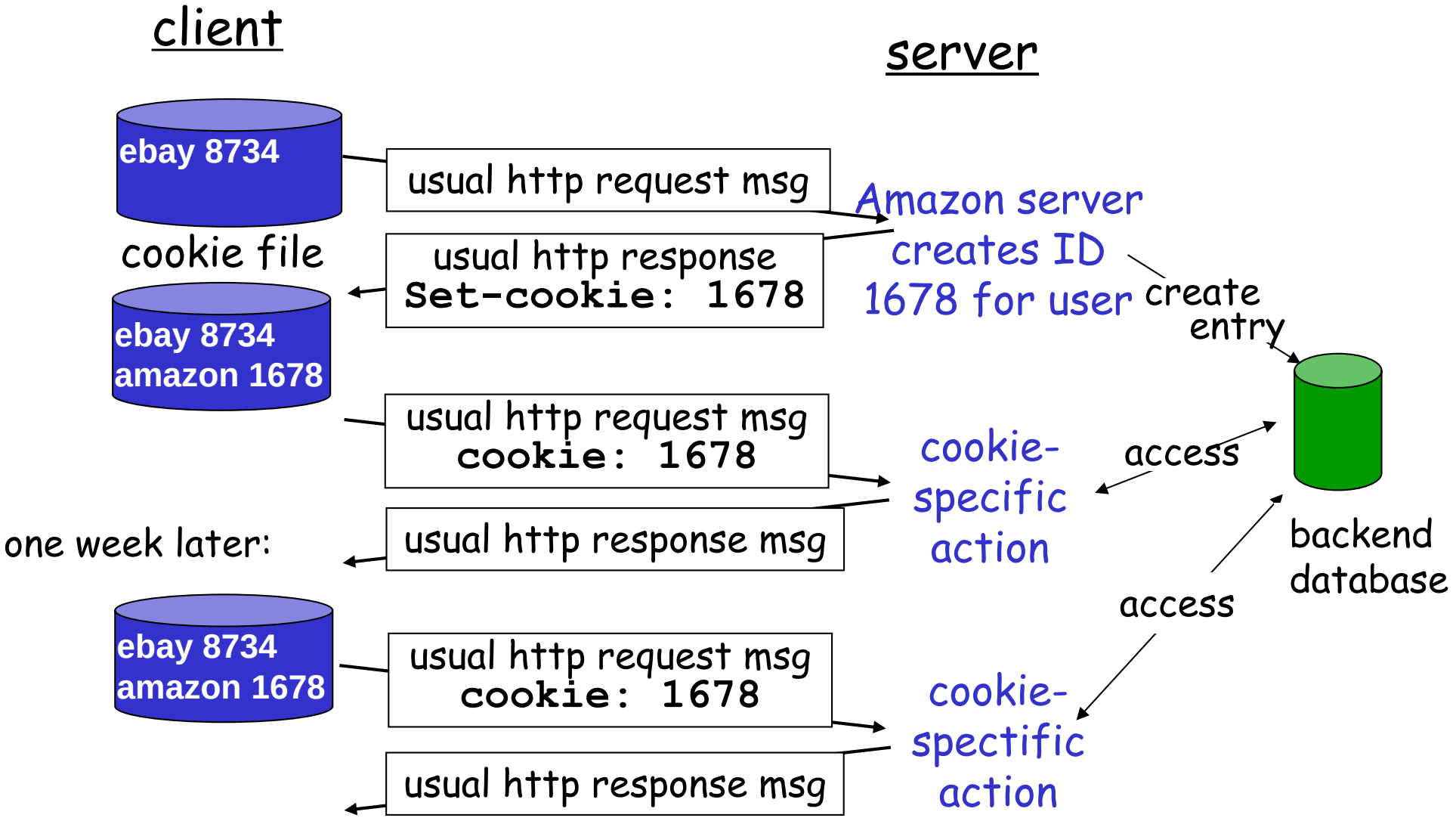
Four components:

1) HTTP响应报文中的cookie首部行
2) HTTP请求报文中的cookie 首部行
3) 保存在用户主机的cookie文件，由用户的浏览器管理
4) Web网站的后端数据库

Example:

- Susan 总是从她的PC机上网
- 她第一次访问某个电子商务网站
- 当HTTP请求报文到达该网站时，网站为其创建:
  - 一个ID
  - 在后端数据库中为该ID建立一个表项

Cookies: 保存“状态”



Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

Cookies 和隐私:

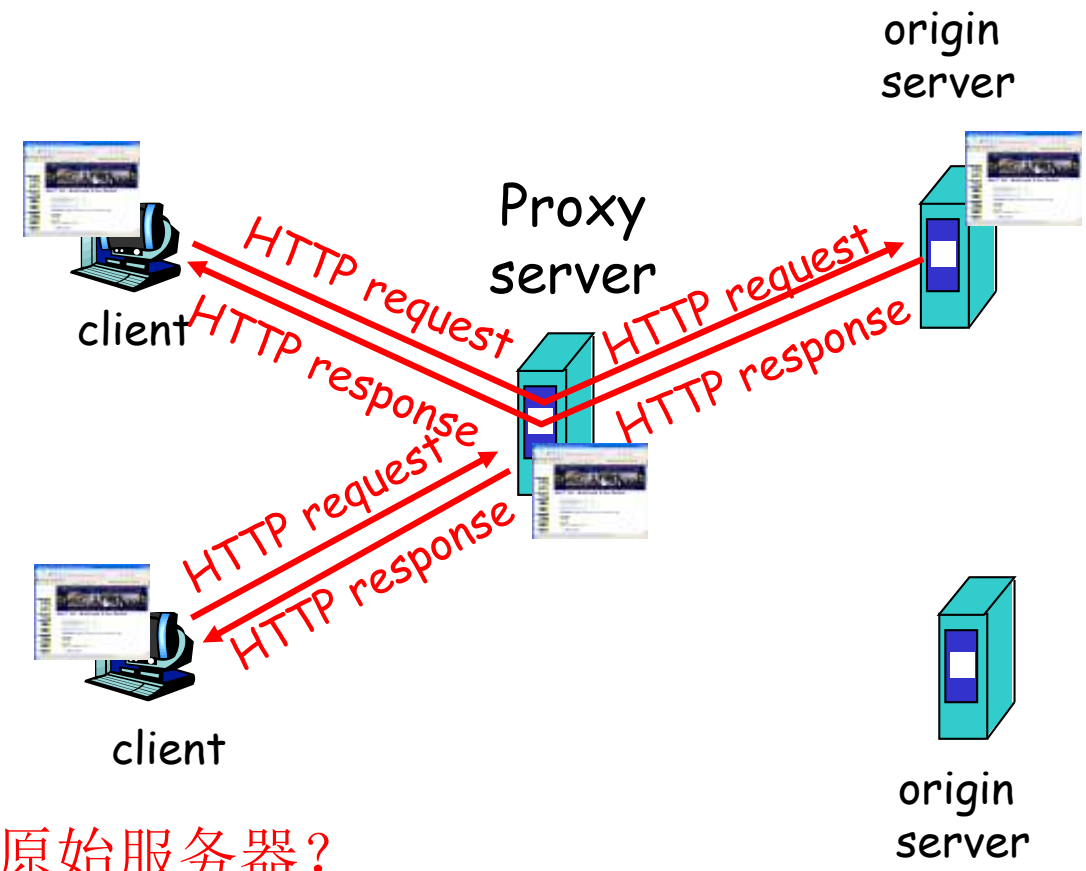
- Cookies允许网站收集用户的大量信息
- you may supply name and e-mail to sites

Where to keep "state":

- 服务端：信息保存在服务端的后端数据库，返回ID给客户
- 客户端：信息发回客户端，保存在cookie文件中，并随请求报文发送给服务器

2.2.5 Web缓存（代理服务器）

- 代理服务器：代表原始服务器满足HTTP请求的网络实体，保存最近请求过的对象的拷贝。
- 用户设置浏览器：所有HTTP请求首先发往web缓存
- 浏览器将HTTP请求发送给web缓存：
  - 对象在web缓存中：web缓存返回对象
  - 对象不在web缓存中：web缓存从原始服务器获取对象，缓存在本地，然后返回给客户



Q: web缓存如何知道对象的原始服务器？

A: HTTP请求头中的host首部行指出了原始服务器

More about Web caching

- Web cache既是服务器又是客户：
  - 对于浏览器来说是服务器
  - 对于原始服务器来说是客户
- Web cache通常由ISP提供，多级ISP可能形成多级web缓存

Why Web caching?

- 减少客户请求的响应时间
- 减少机构接入链路上的流量

Caching example

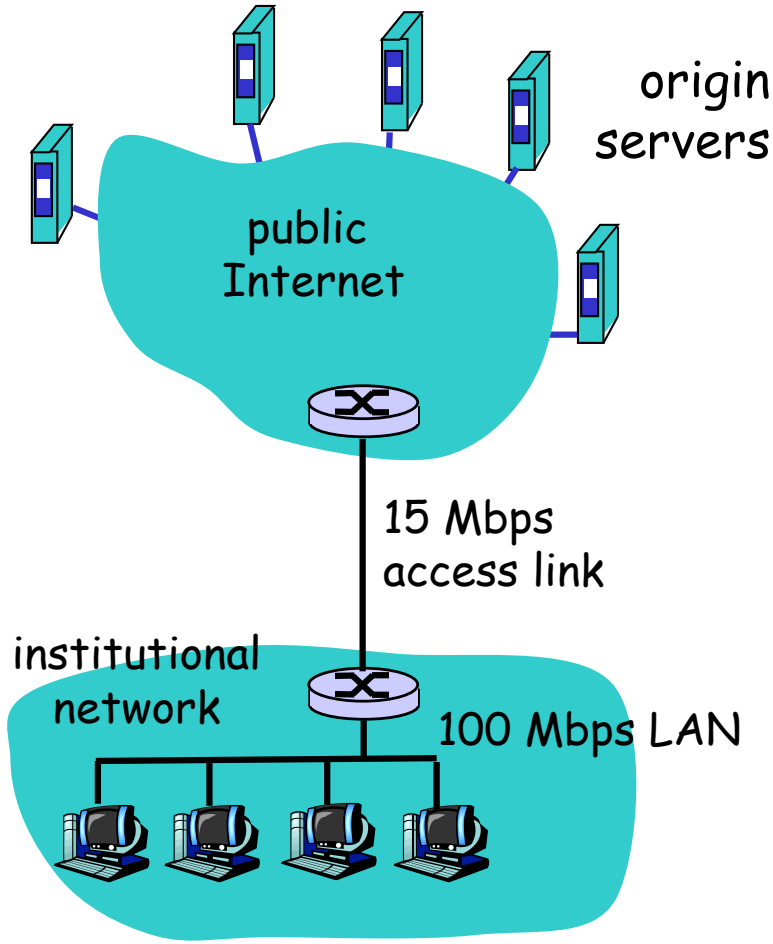
Assumptions

- 平均对象长度 = 1Mb
- 从机构浏览器到原始服务器的平均请求速率 = 15/sec
- 从接入路由器到原始服务器的来回延迟（RTT）= 2 sec

Consequences

- LAN利用率 = 15%
- 接入链路的利用率 = 100%
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + milliseconds



Caching example (cont)

possible solution

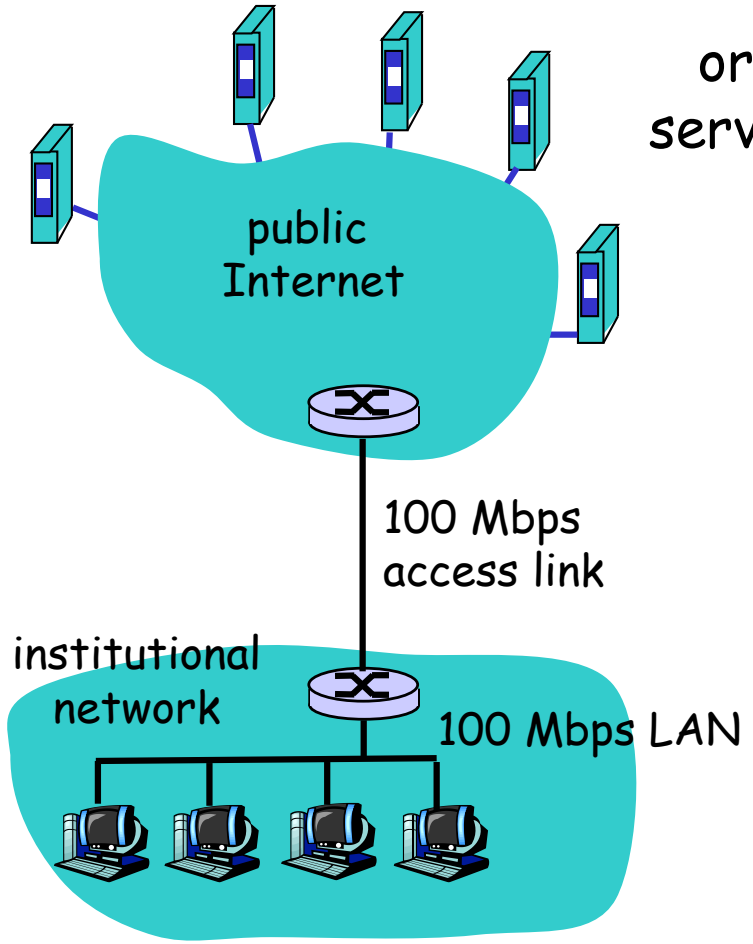
- 提高接入链路带宽至100Mbps

consequence

- LAN利用率 = 15%
- 接入链路利用率 = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs

- 链路升级的代价高昂！



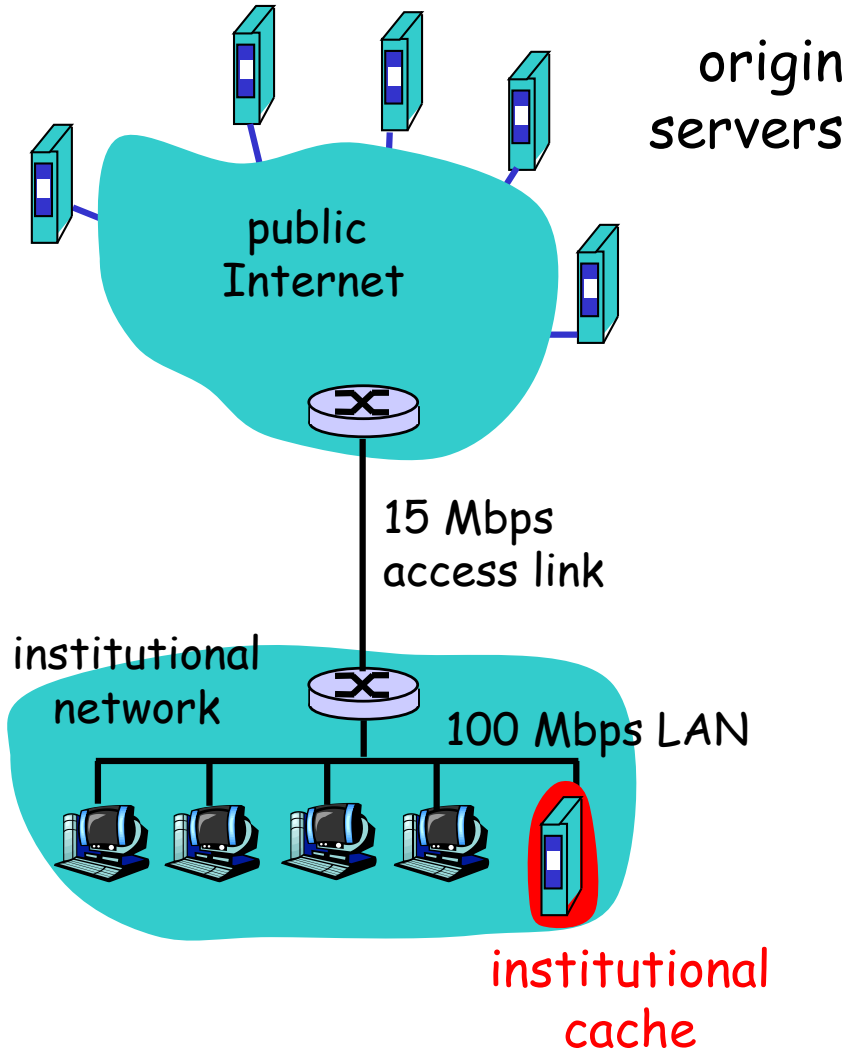
Caching example (cont)

possible solution:

- 安装web cache
- 假设cache命中率为 0.4

consequence

- 40%的请求可被立即满足
- 60%的请求被原始服务器满足
- 接入链路利用率为60%, 延迟为～10msecs
- total avg delay = Internet delay + access delay + LAN delay = 0.6*(~2.01 secs) + 0.4*(~msecs) = ~ 1.2 secs
- 比升级链路的延迟还要低！



2.2.6 条件 GET

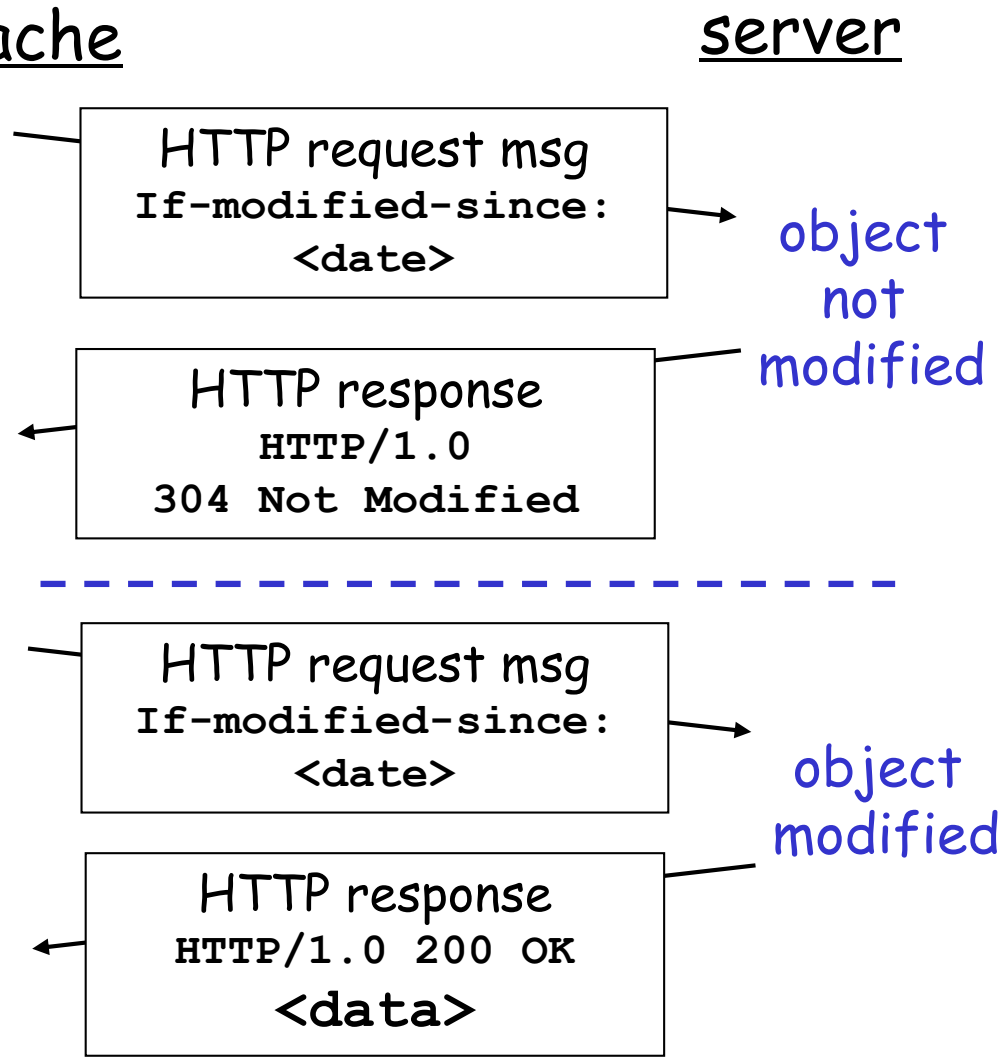
代理服务器如何发现缓存的对象是不是最新的？

- web cache在发送的GET报文中包含以下首部行

  If-modified-since: <date>

- Server:
  - 若发现对象无更新，响应报文中不包含对象:

    HTTP/1.0 304 Not Modified
  - 若发现对象有更新，响应报文中包含对象
- 对象修改时间在哪里提供？
  - HTTP响应报文中的Last-modified首部行
  - Web缓存将该信息拷贝到if-modified-since首部行



小结

- 因特网上的每个资源（对象）都可以通过URL访问
- HTTP使用TCP连接
  - 非持续HTTP
  - 持续HTTP
- 客户与服务器交互：
  - 报文请求/响应
- HTTP服务器是无状态的，利用cookie技术可以保存用户状态
- 使用web缓存提高请求-响应速度：
  - 使用条件get获得最新的对象副本

理解http及web技术的发展脉络

- 为什么HTTP最初设计为是非持久连接、无状态的？
- 后来为什么引入持久连接？
- 为什么引入cookie技术，它带来什么问题？
- 为什么引入web缓存技术，它带来什么问题，如何解决？

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

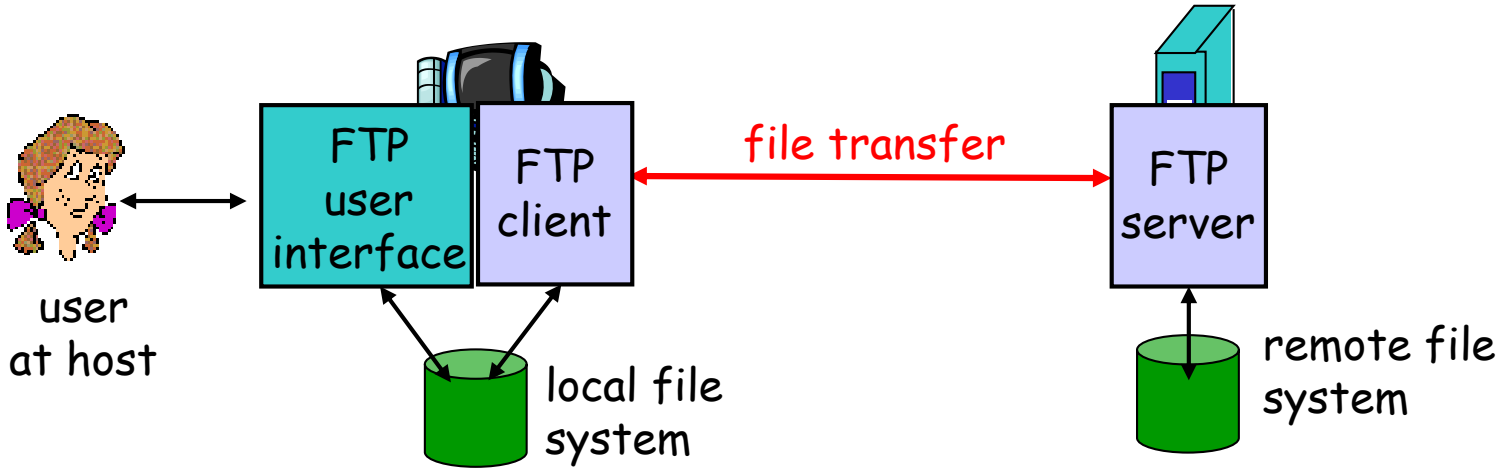
2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

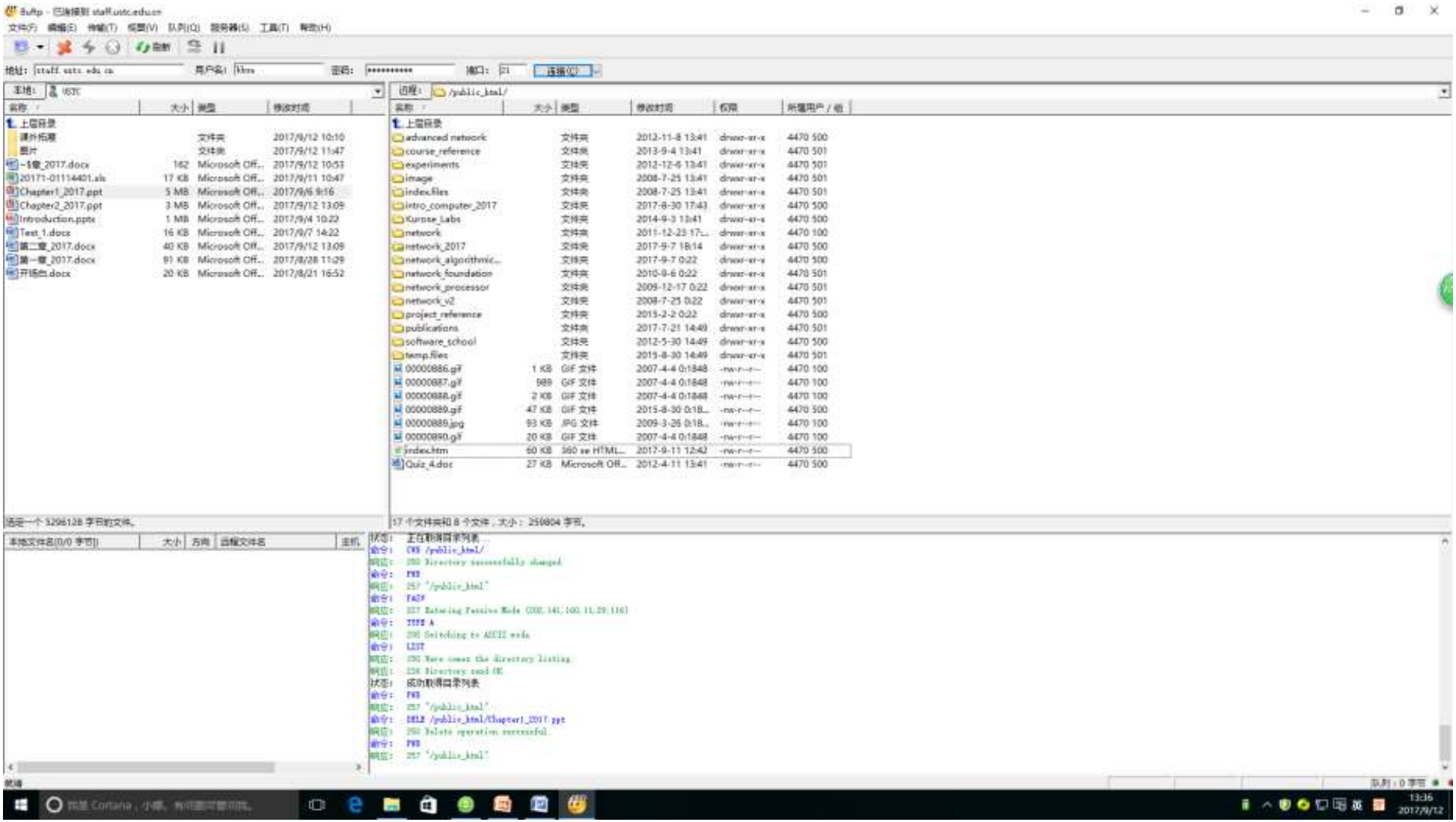
2.7 socket programming with UDP and TCP

文件传输应用画像



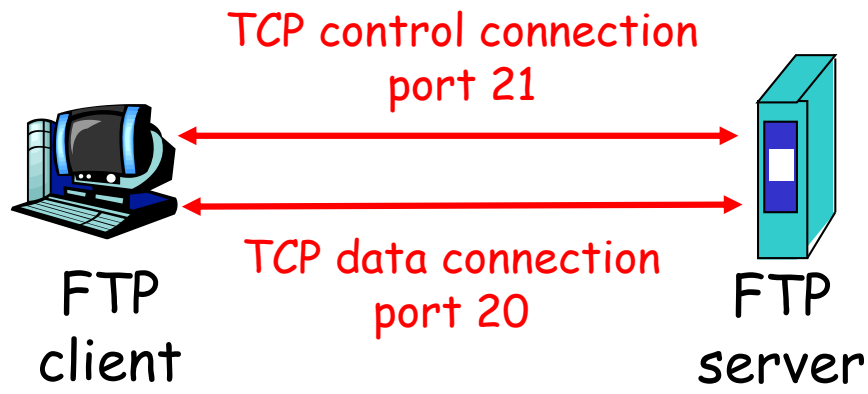
- 应用层资源：文件
- 网络应用架构：客户-服务器
- 使用传输服务：TCP
- 端口号：21、20
- 应用层协议：FTP
  - 使用命令/响应交互（不是像HTTP那样使用报文交互）
  - 通常情况下，用户通过FTP用户代理与文件服务器交互

FTP用户代理



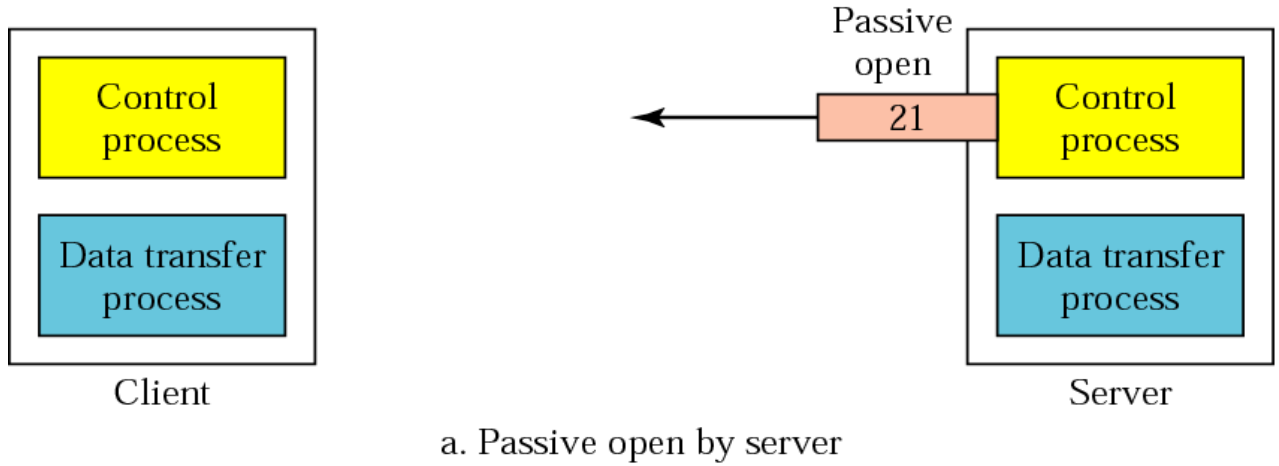
使用分离的控制连接和数据连接

- 控制连接：
  - 使用端口21，传送客户命令和服务器响应
  - 控制连接在整个会话期间一直保持
- 数据连接：
  - 使用端口20，传输文件
  - 每个数据连接只传输一个文件，发送方用关闭连接表示一个文件传输结束



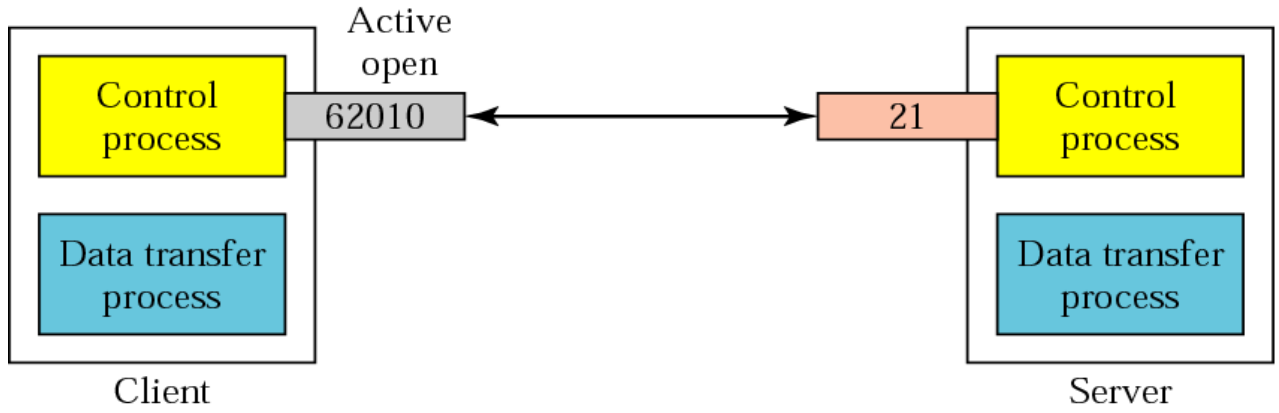
建立控制连接

- 服务器在端口21等待客户



a. Passive open by server

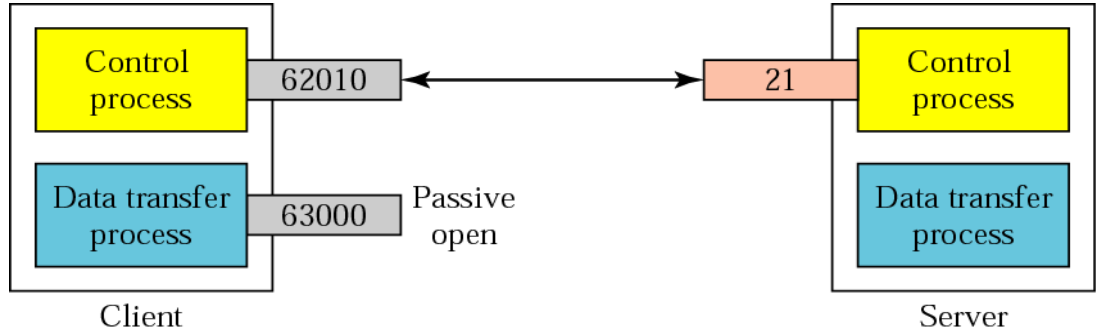
- 客户使用临时端口号建立连接



b. Active open by client

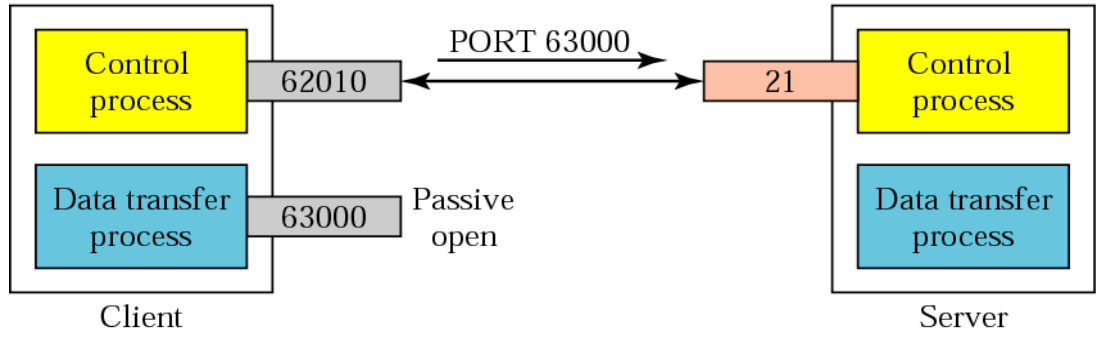
建立数据连接（主动模式）

- 客户选择一个临时端口号，在该端口上等待服务器的连接请求



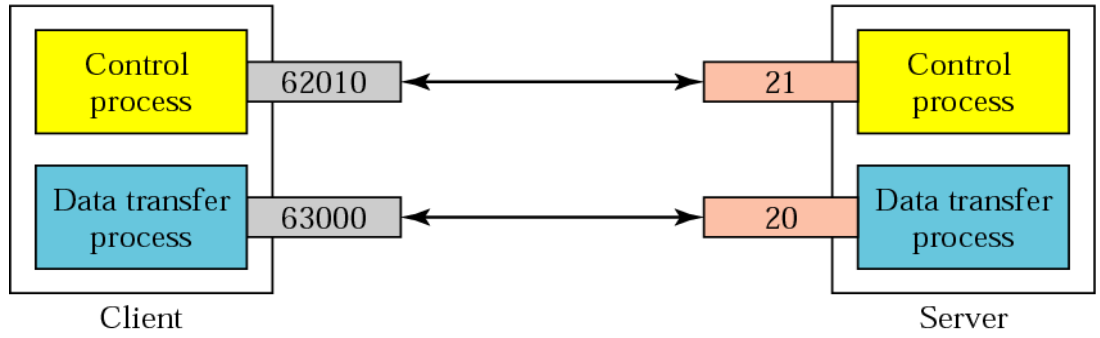
a. Passive open by client

- 客户在控制连接上用PORT命令将临时端口号发送给服务器



b. Sending ephemeral port number to server

- 服务器使用端口20与客户机给出的端口建立连接



c. Active open by server

有关控制连接与数据连接的问题

- Q：为什么将控制连接与数据连接分开？
- A：不会混淆数据与命令/响应，简化协议设计和实现；在传输文件的过程中可以继续执行其它的操作；便于控制传输过程（如客户可以随时终止传输）

- Q：为什么用关闭数据连接的方式结束文件传输？
- A：允许动态创建文件（不需预先告知文件的大小）

FTP小结

- 使用TCP协议，服务器端口号21、20
- 采用命令/响应交互方式
- 使用两条TCP连接：
  - 控制连接（端口21）：会话期间始终保持
  - 数据连接（端口20）：每条连接仅传输一个文件，且客户与服务器角色交换
- 要求理解：
  - 为什么使用两条分开的连接，即为什么不在同一条连接上既传输命令/响应、又传输数据？
  - 为什么每条数据连接只传输一个文件？

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

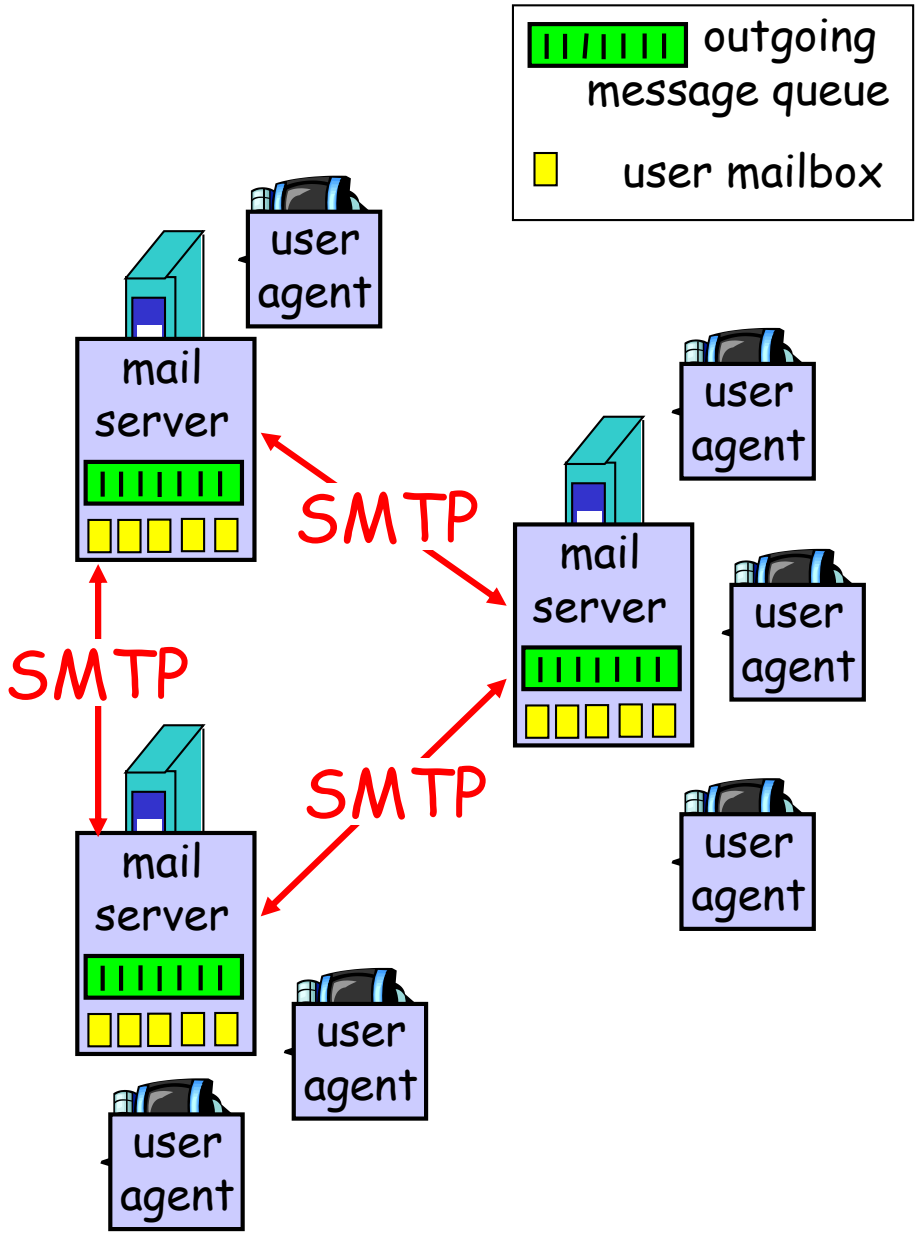
电子邮件应用画像

- 应用层传输对象：邮件
- 网络应用架构：客户-服务器架构
- 使用传输服务：TCP
- 应用层协议：
  - 邮件传输协议：SMTP（端口号25）
  - 邮件访问协议：POP3（端口号110），IMAP（端口号143），HTTP（端口号80）
- SMTP、POP3、IMAP采用命令/响应交互

2.3.1 电子邮件系统

最初由三个部分组成:

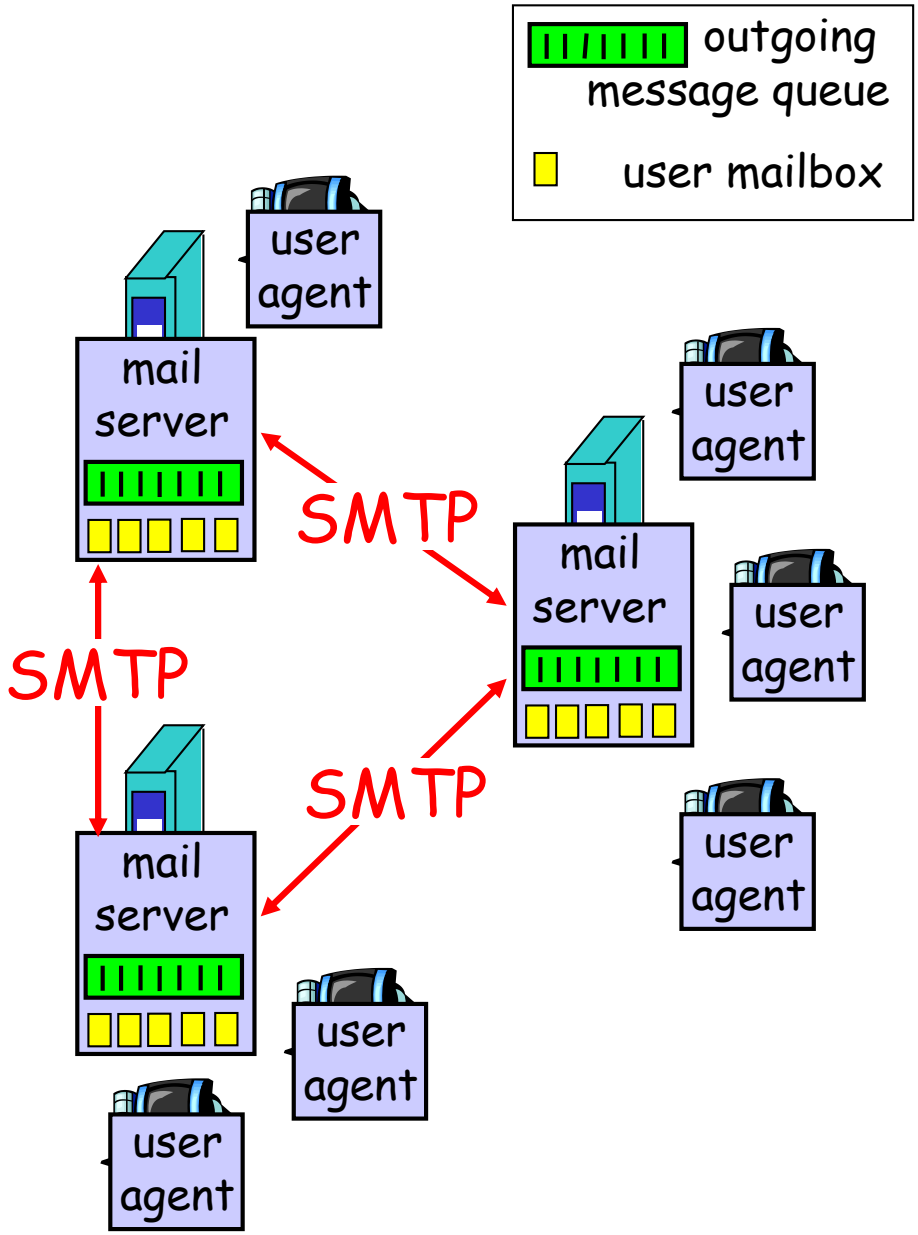
- 用户代理
- 邮件服务器
- 简单邮件传输协议



用户代理

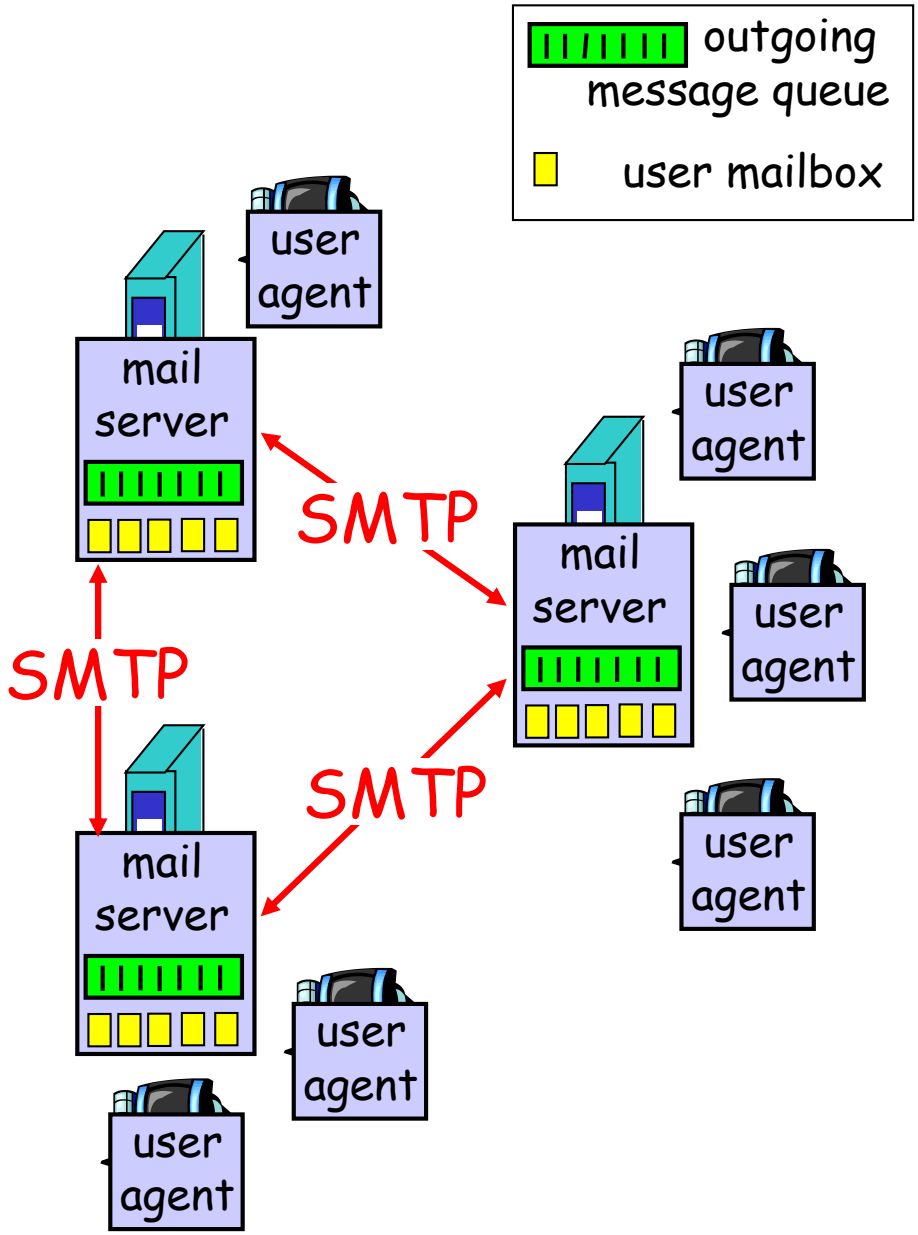
- 编辑、阅读、回复邮件等
- 将要外发的邮件发送到用户的邮件服务器
- 从用户邮箱中取邮件

- 一些用户代理：
  - Outlook, elm, Mozilla, Thunderbird



邮件服务器

- 邮件服务器包含：
  - 用户信箱：存放到来的邮件
  - 发送报文队列：存放要发送出去的邮件
  - 报文传输代理MTA：运行在服务器后台的系统守护进程，负责在邮件服务器之间传输邮件，及将收到的邮件放入用户信箱



电子信箱

- 电子信箱：
  - 由计算机上的一个存储区域（如磁盘上的一个文件）组成
  - 每个信箱均被分配了唯一的电子邮件地址

- 电子邮件地址：
  - 由两个部分组成，形如：mailbox@computer
  - 前者为标识用户信箱的字符串，后者为信箱所在的邮件服务器的名字

简单邮件传输协议--SMTP [RFC 2821]

- 邮件服务器之间传输邮件采用客户-服务器模式：
  - 客户：发送邮件的邮件服务器（报文传输代理）
  - 服务器：接收邮件的邮件服务器（报文传输代理）
- SMTP使用TCP协议，服务器端口为25
- 发送方和接收方的邮件服务器之间直接传输邮件
- SMTP采用命令/响应交互方式：
  - 命令：ASCII文本
  - 响应：状态码和短语
- 报文只能包含简单ASCII文本，即7位ASCII码（最初仅为英文电子邮件而设计）

Scenario: Alice sends message to Bob

1) Alice使用用户代理编辑邮件，发送给 **bob@someschool.edu**

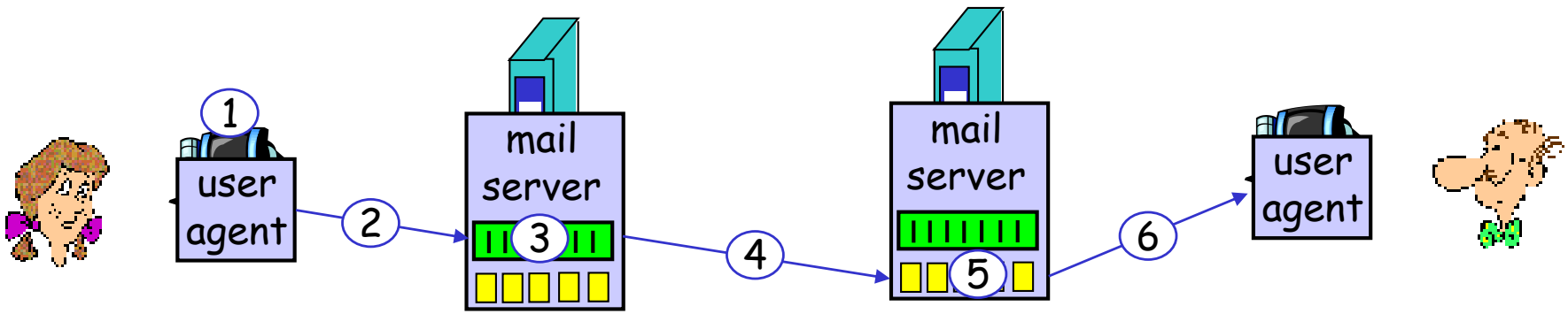
2) Alice的用户代理将报文发送给其邮件服务器（使用SMTP）

3) 邮件被置于邮件服务器的发送报文队列中

4) Alice的邮件服务器与Bob的邮件服务器建立TCP连接，然后发送 Alice的邮件

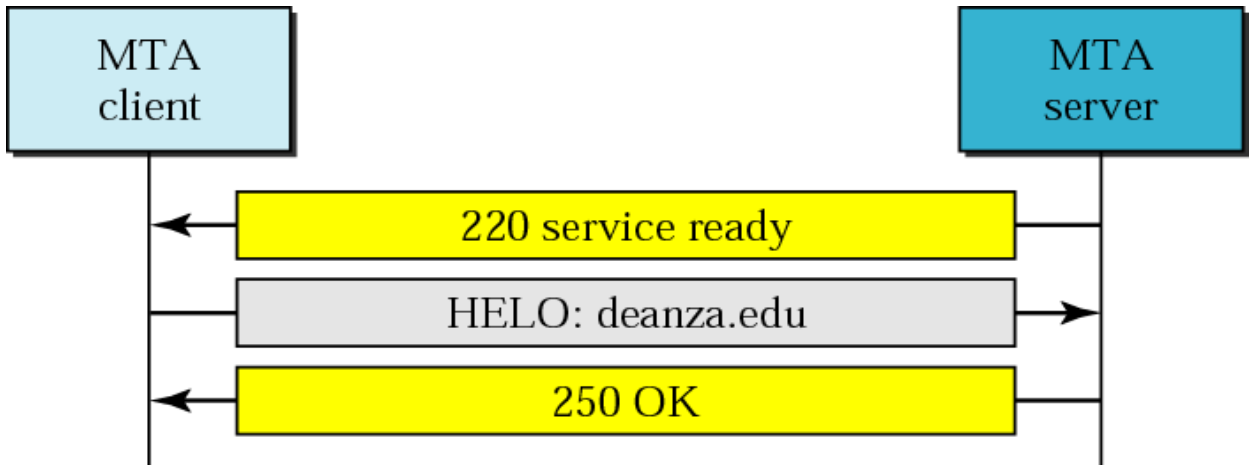
5) Bob的邮件服务器将邮件放置在Bob的信箱中

6) Bob调用他的用户代理阅读邮件

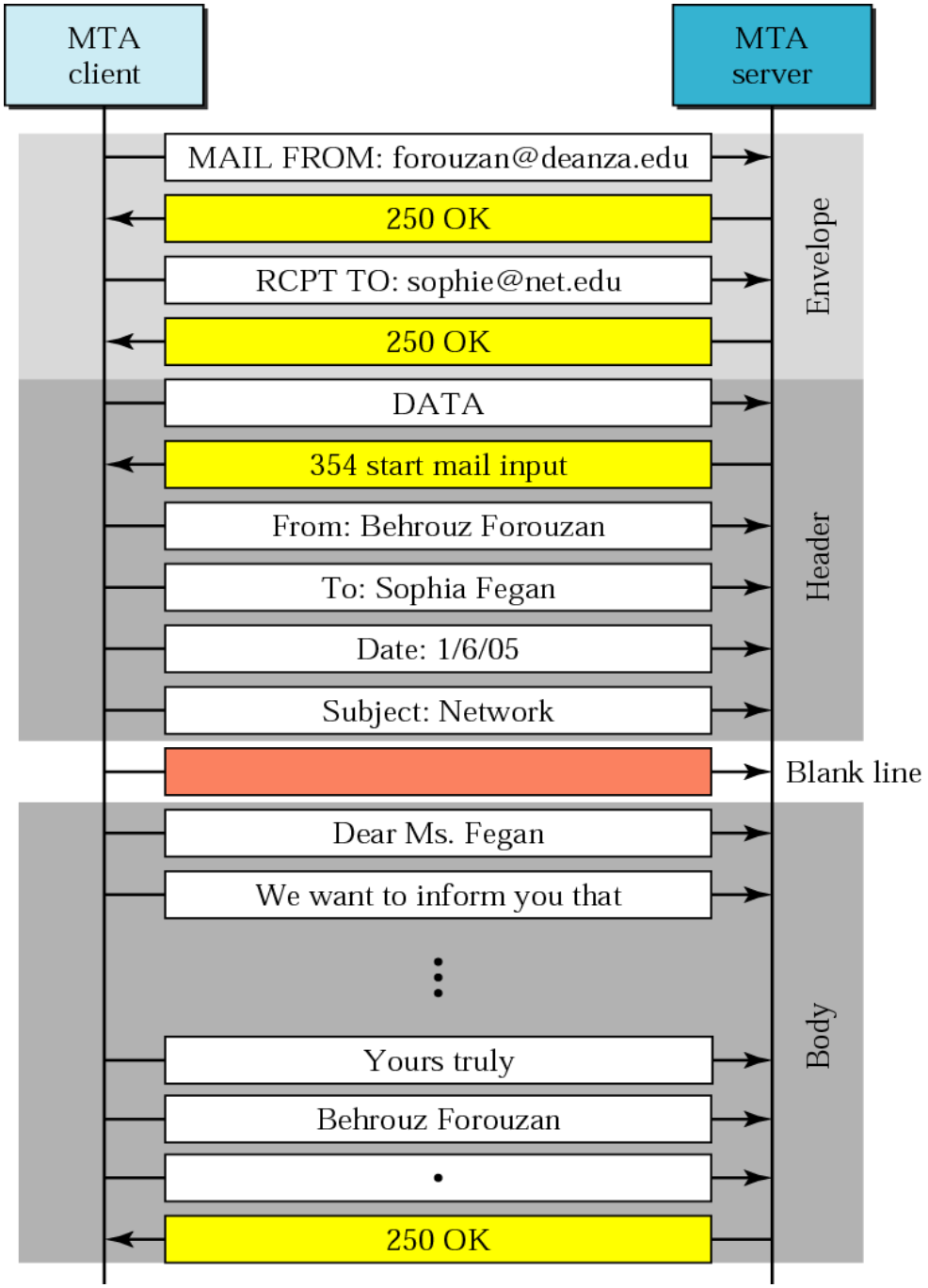


邮件传送阶段（1）

- 连接与握手：
  - MTA客户与MTA服务器在端口25建立TCP连接。
  - 服务器发送服务就绪报文
  - 客户发送HELO报文，用域名标识自己
  - 服务器响应

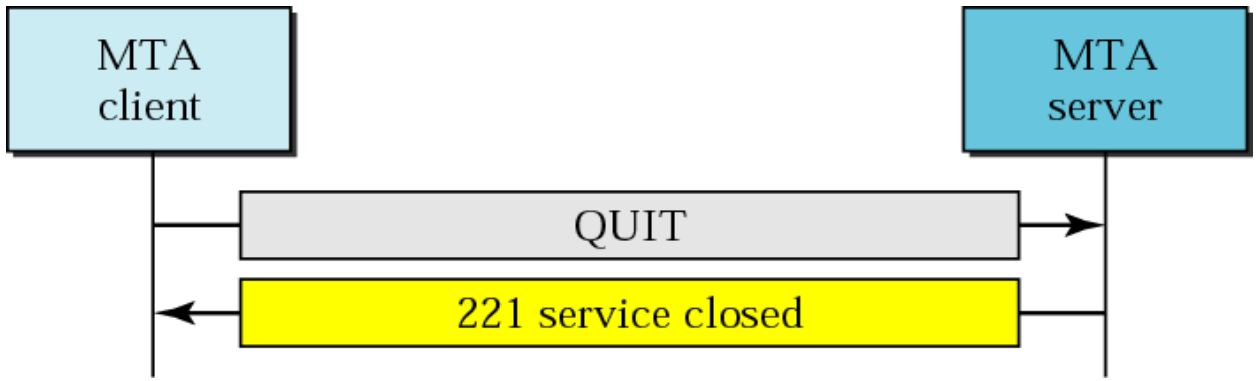


邮件传送阶段（2）


MTA client
MTA server
MAIL FROM: forouzan@deanza.edu
250 OK
RCPT TO: sophie@net.edu
250 OK
Envelope
DATA
354 start mail input
From: Behrouz Forouzan
To: Sophia Fegan
Date: 1/6/05
Subject: Network
Header
Blank line
Dear Ms. Fegan
We want to inform you that
Yours truly
Behrouz Forouzan
250 OK
Body

邮件传送阶段（3）

- 结束传输
  - 客户发送QUIT命令
  - 服务器响应
  - 释放TCP连接

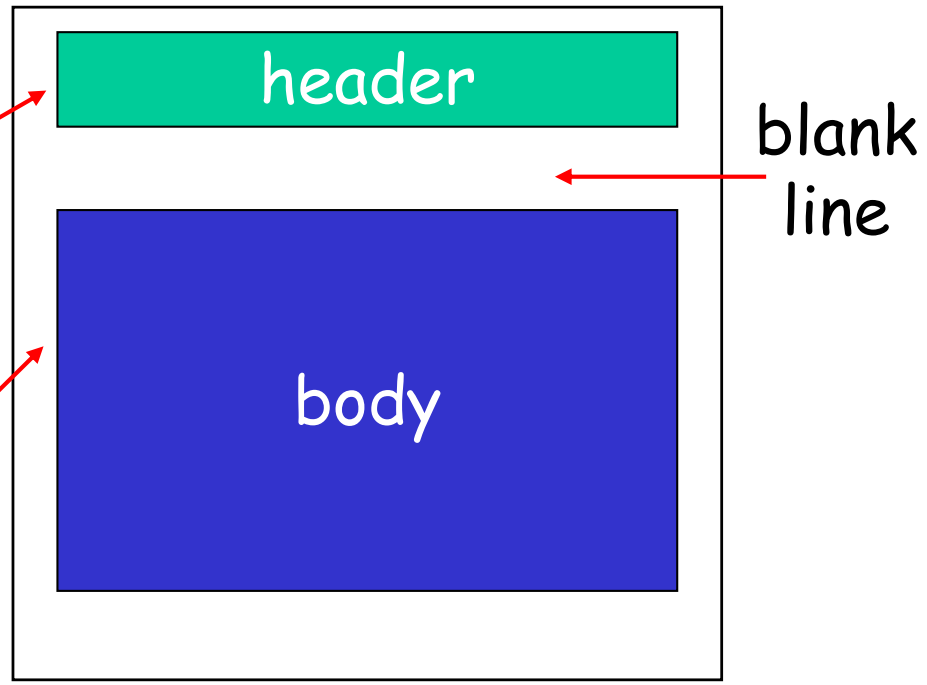


SMTP: final words

- SMTP使用持久连接：
  - 可以在一条TCP连接上传输多个报文（FTP只传输一个文件，HTTP可传输一个或多个对象）
  - 一个方向的报文传输结束后，可以在另一个方向上传输报文（SMTP特有的）
- SMTP 服务器使用“.”表示报文结束（FTP使用关闭连接表示文件结束，HTTP使用长度域表示报文结束）
- SMTP要求报文（报头和实体）只包含简单ASCII文本（HTTP无此要求）

2.3.2 邮件报文格式

- RFC 822规定了邮件报文格式
- 报头由一些首部行组成，如：
  - To:
  - From:
  - Subject:
- 实体: 只能使用简单ASCII字符



如何传输包含非ASCII文本的报文？

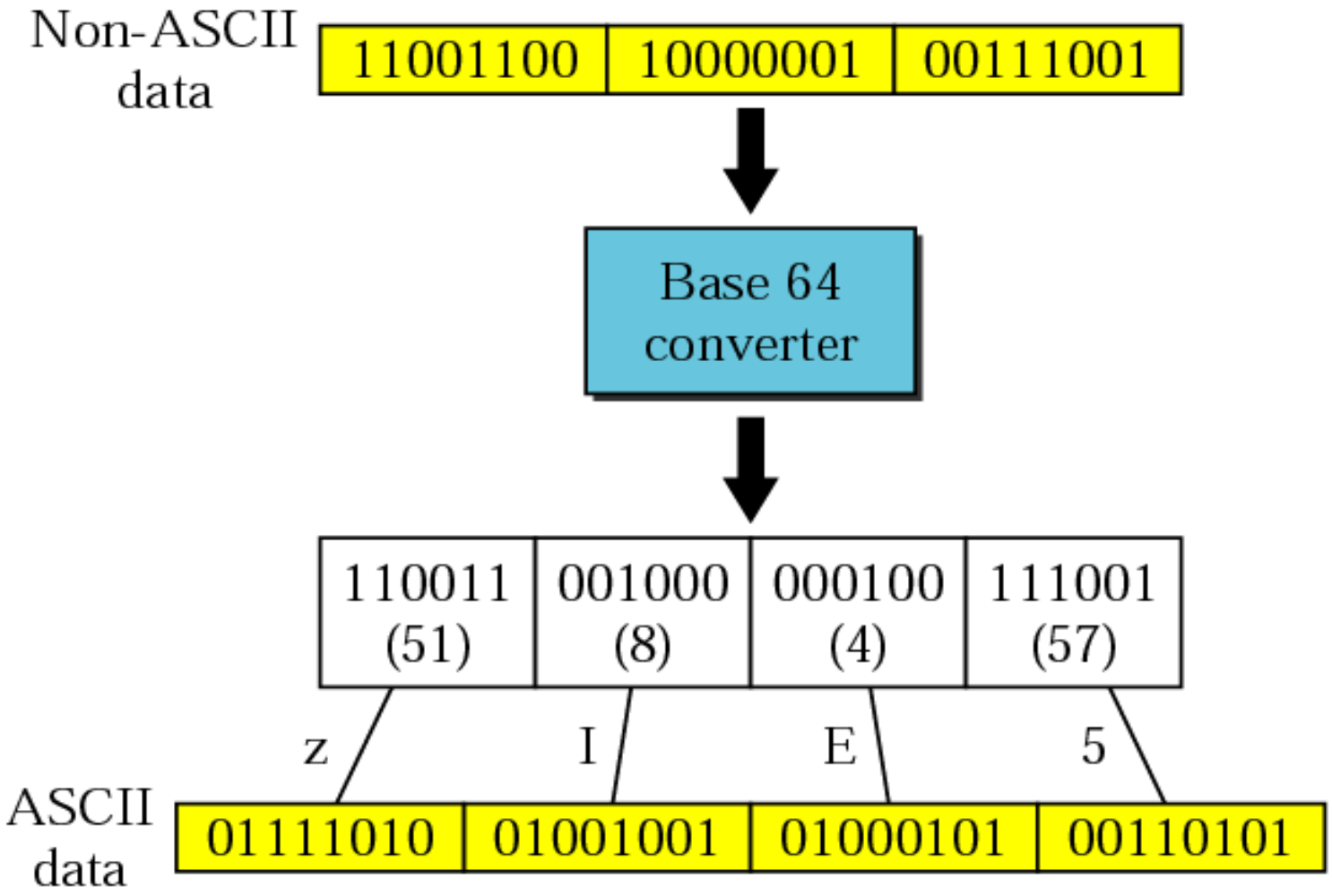
- 现在的电子邮件要求能传输其它语系的文字，甚至非文本信息（如图片）
  - 非ASCII文本形式的数据，在发送前须转换成简单ASCII文本

- 非ASCII文本的报文大多具有特殊的数据类型，需要特殊的邮件浏览器（如JPEG图形的解压缩软件）来阅读
  - 需扩展报文的数据类型

Base64编码

- Base64用来将一个二进制字节序列，转换成由ASCII字符序列构成的文本
- 每24比特数据划分成4个6比特的单元，每个单元编码成一个ASCII字符，其对应关系为：
  - 0～25编码成‘A’～‘Z’
  - 26～51编码成‘a’～‘z’
  - 52～61编码成‘0’～‘9’
  - 62和63分别编码成‘＋’和‘/’
  - 若最后一组只有8比特或16比特，分别加上‘＝＝’和‘＝’后缀
  - 回车和换行忽略，可以插在任何地方

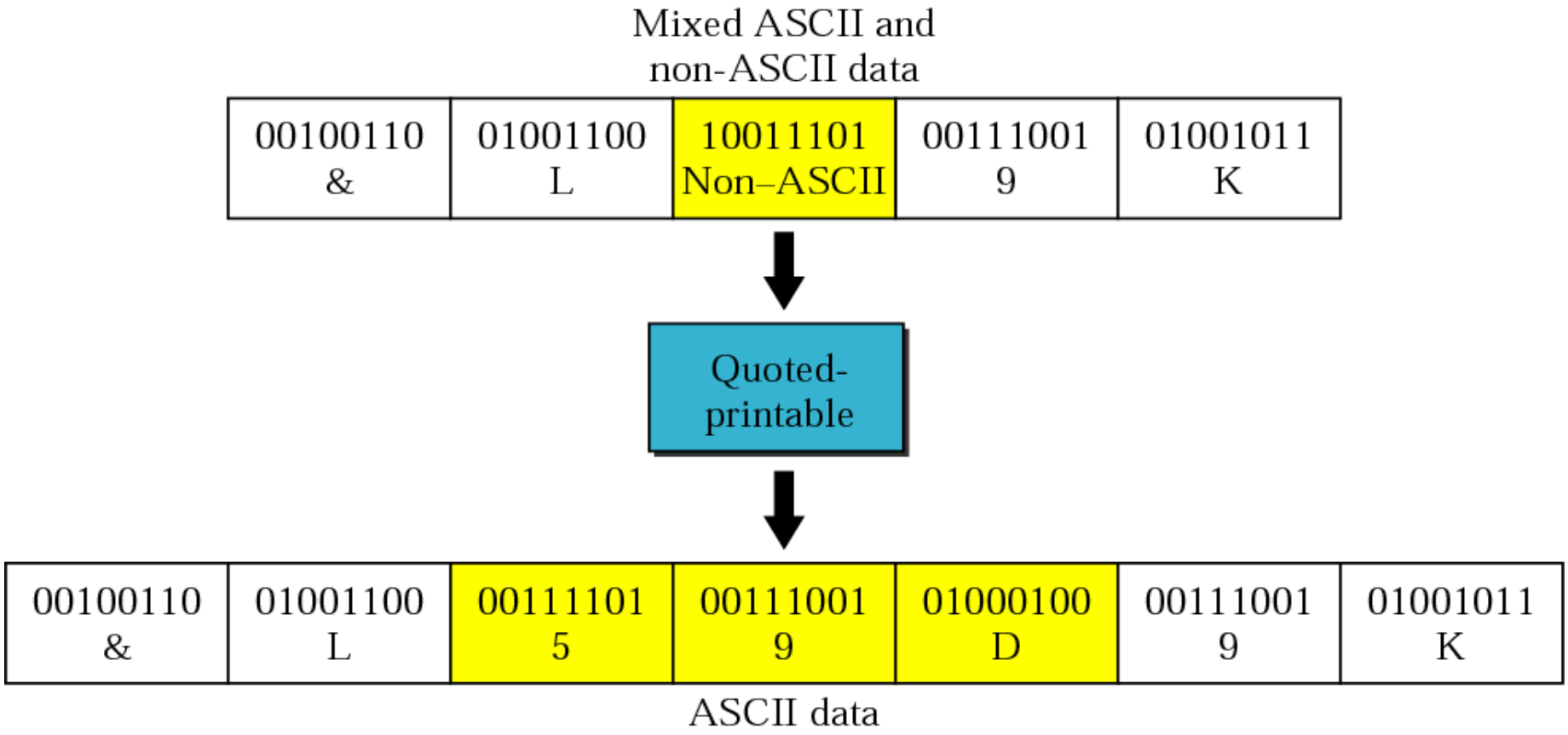
Base64编码示例


Non-ASCII data
11001100
10000001
00111001
Base 64 converter
110011 (51)
001000 (8)
000100 (4)
111001 (57)
z
I
E
5
ASCII data
01111010
01001001
01000101
00110101

quoted-printable编码

- 适用于绝大部分都是ASCII字符的报文实体，其编码方法是：
  - 每个ASCII字符保持不变
  - 对于非ASCII字符（大于127的字符），将该字符的十六进制表示用两个ASCII字符标记，前面冠以特殊字符“=”。

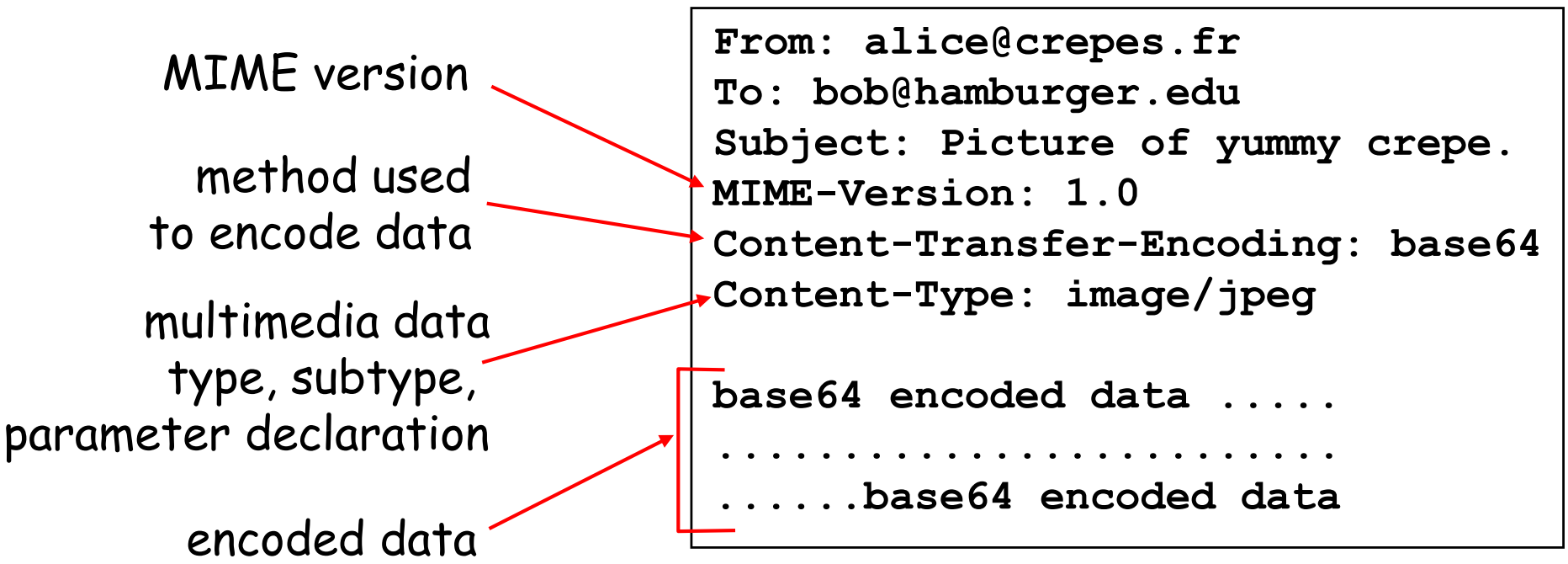
quoted-printable编码示例



多用途因特网邮件扩展协议MIME

- 扩展了RFC 822，允许实体具有不同的数据类型，并规定了非ASCII文本信息在传输时的统一编码形式
- 扩充了一些首部行，最重要的是：
  - **Content-Transfer-Encoding**：实体采用的传输编码形式
  - **Content-Type**：实体的数据类型及子类型

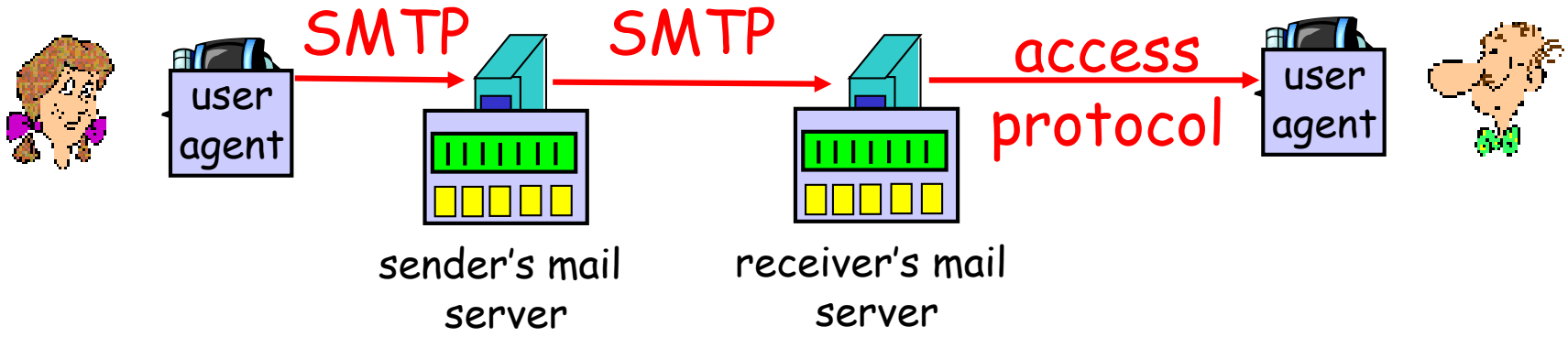
MIME报文格式 [RFC 2045, 2056]



2.3.3 邮件访问

- 邮件访问方式：
  - 早期：用户登陆到邮件服务器上，直接在服务器上运行一个邮件阅读程序来阅读邮件
  - 今天：用户在终端上安装用户代理，获取和阅读邮件
- Q：能将用户信箱放在本地终端吗？
- A：不能，用户终端不可能一直连在因特网上
- Q：用户代理能用SMTP从邮件服务器获取邮件吗？
- A：不能，SMTP是一个"推"协议，只能将邮件从用户代理推送到其邮件服务器，或从发送方邮件服务器推送到收件人邮件服务器
- 解决方案：设计一个新的协议从服务器获取邮件

邮件的两阶段交付



- 在具有永久因特网连接的计算机上运行一个SMTP服务器，为用户分配一个永久信箱
- 第一阶段：邮件被投递到收信人的永久信箱
- 第二阶段：用户从永久信箱中获取邮件
- 为此，带永久信箱的计算机必须运行两个服务器程序：
  - SMTP服务器：收发用户邮件，将收到的邮件放入用户信箱
  - 邮件访问服务器：允许用户从信箱中提取邮件

邮件访问协议

可以从服务器获取邮件的协议有：

- POP: Post Office Protocol [RFC 1939]
  - authorization (agent <-->server) and download
- IMAP: Internet Mail Access Protocol [RFC 1730]
  - more features (more complex)
  - manipulation of stored msgs on server
- HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

认证和授权阶段

- 客户命令:
  - `user`: declare username
  - `pass`: password
- 服务器响应
  - `+OK`
  - `-ERR`

事务阶段

- 客户命令:
  - `list`: list message numbers
  - `retr`: retrieve message by number
  - `dele`: delete
  - `quit`

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

More about POP3

- 前一个例子使用了“download and delete”模式，在另一台终端上无法再获取邮件
- “Download-and-keep”模式可将邮件保留在服务器中

IMAP

- 所有邮件保存在服务器上
- 允许用户将邮件组织在文件夹中
- 允许用户在文件夹之间移动邮件

基于web的邮件访问：HTTP

- 用户代理为普通浏览器:
  - 发送邮件：浏览器使用HTTP协议将邮件发送到邮件服务器
  - 获取邮件：浏览器使用HTTP协议从信箱取邮件
  - 传输邮件：邮件服务器之间仍使用SMTP协议传输报文
- 和IMAP一样，用户可以在远程服务器上用文件夹来组织他们的邮件

现代因特网电子邮件系统的组成

- 用户代理
- 邮件服务器
- 简单邮件传输协议SMTP
- 邮件访问协议（POP3，IMAP，HTTP）

小结

- 电子邮件系统：
  - 4个组成部分
- SMTP协议：
  - 使用TCP协议，服务器端口号25
  - “推”协议：将邮件推向用户信箱
  - 命令/响应交互方式
  - 信体只能包含简单ASCII文本
- MIME协议：
  - 允许信体包含非ASCII文本
  - 规定传输编码类型，扩展数据类型
- 两阶段交付过程：
  - 邮件投递：邮件从发信方投递到用户信箱
  - 邮件访问：收信人从用户信箱获（拉）取邮件

理解HTTP、FTP、SMTP设计上的不同

- HTTP、FTP、SMTP均是在TCP连接上传输文件，但是在设计上有一些不同
- 使用持久连接或非持久连接：
  - HTTP可在一条TCP连接上传输多个对象，SMTP可以传输多个邮件，FTP只传输一个文件
- 文件传输结束的标记：
  - HTTP使用长度指示报文结束，FTP使用关闭连接表示文件结束，SMTP 使用“.”表示报文结束
- 文件内容的要求：
  - SMTP要求邮件只包含简单ASCII文本，FTP和HTTP无此要求
- 客户-服务器交互方式：
  - HTTP采用报文交互，SMTP和FTP采用命令/响应交互

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

DNS: Domain Name System

- 因特网主机需要标识:
  - IP地址（32 bit）- 由机器使用
  - 名字，如 www.yahoo.com - 由人类使用，便于记忆
- Q: 如何提供主机名到IP地址的映射?
- 因特网的目录服务DNS:
  - 将主机名映射到IP地址
- DNS实现为一个应用层服务:
  - 由其它网络应用使用的服务
  - 客户-服务器模式
  - 传输服务:
    - 主要使用UDP，有时使用TCP
  - 端口号53
  - 请求/响应报文交互

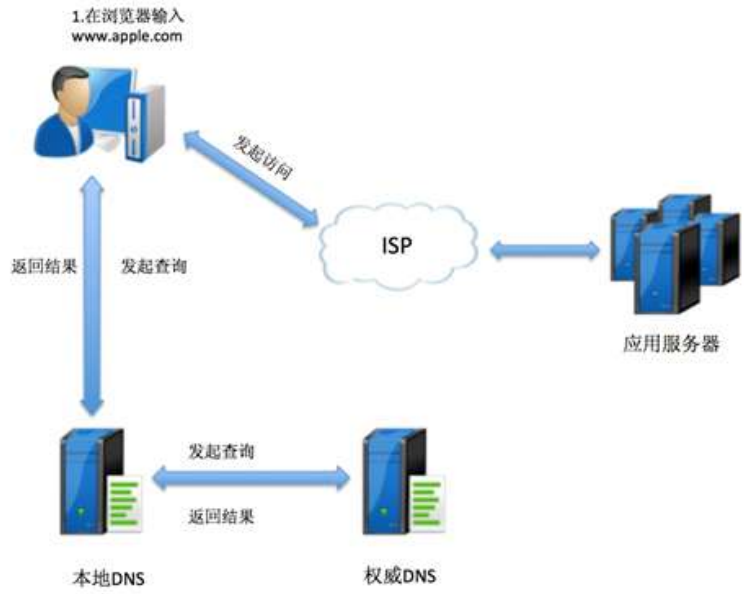
2.4.1 DNS提供的服务

- 主机名-IP地址转换
- 主机别名：
  - 允许主机除了规范名外，具有一个或多个别名（易于记忆），如www.ustc.edu.cn
  - 提供主机别名到规范名的映射
  - 迁移服务不需要修改主机名
- 邮件服务器别名：
  - 允许使用域名作为邮件服务器的别名
  - 如：xxx@ustc.edu.cn
- 负载分配：
  - 允许一个规范主机名对应一组IP地址
  - 将服务请求在一群相同功能的服务器之间分配

2.4.2 DNS工作机理

将主机名转换成IP地址：

- 应用程序（如浏览器）调用一个本地例程（解析器），主机名作为参数之一传递
- 解析器向网络中的DNS服务器发送查询报文，包含要查询的主机名
- 解析器收到包含IP地址的响应报文
- 解析器将IP地址返回给调用者（如浏览器）



** 对应用程序而言，DNS是一个提供直接转换服务的黑盒子

DNS是一个分布式数据库

Q：为什么不使用集中式的DNS？

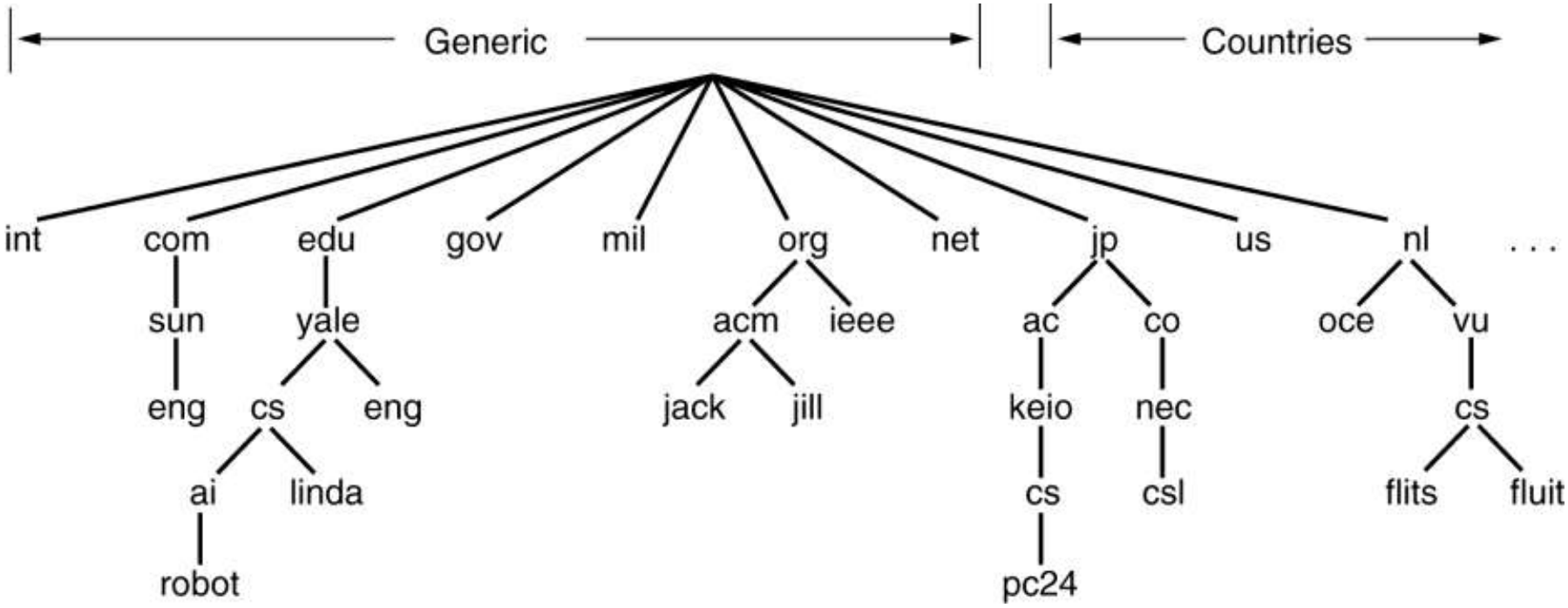
- 单点失效
- 流量集中：单个DNS服务器需处理全部查询
- 响应时间长：远距离的集中式数据库
- 需要维护庞大的数据库

A：Doesn't scale!

分布式环境中的名字空间

如何在分布式环境下避免出现名字冲突？

- DNS使用名字空间来规范对主机的命名：
  - 名字空间是因特网主机名字的集合，它同时给出了命名主机的方法
  - 概念上，因特网被划分成200多个顶级域，每个顶级域可继续划分子域，依次类推
  - 主机名字采用分层的命名方法



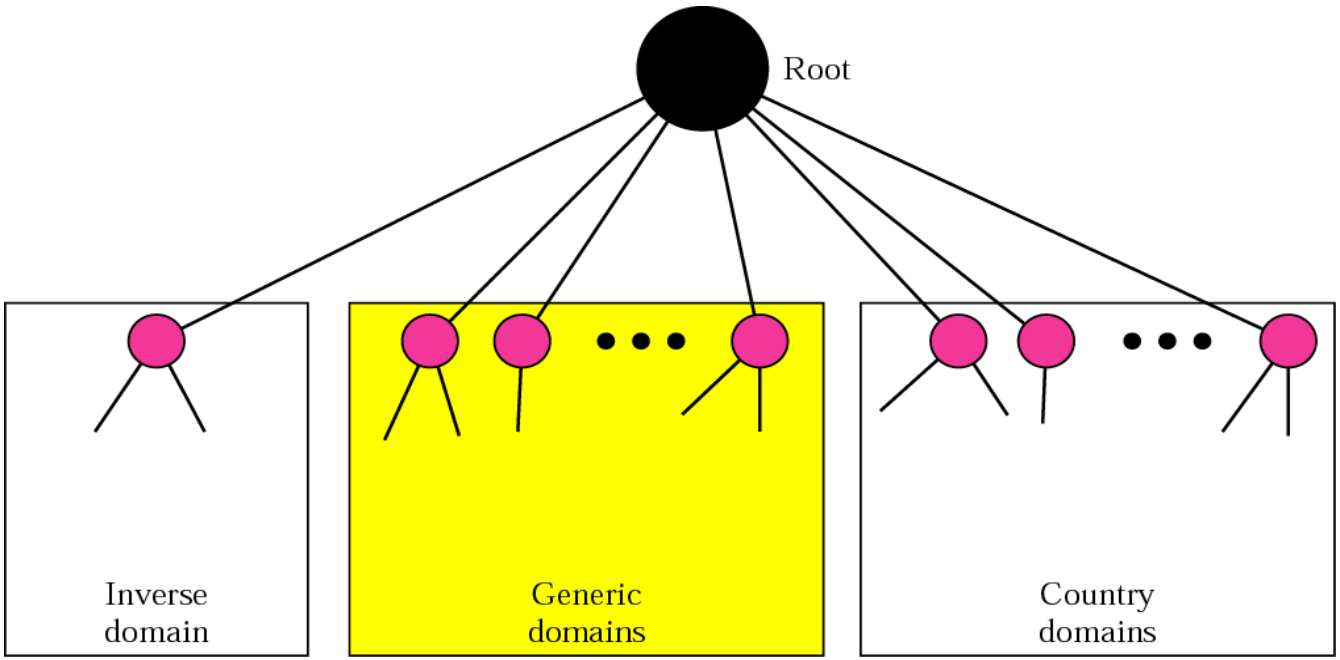
域名

- 域（domain）：
  - 名字树中，以任何一个节点为根的子树构成一个域
- 标记（label）：
  - 树上每一个节点都有一个标记（最多63个字符），树根的标记是一个空字符串
- 域名（domain name）：
  - 某个域的名字表示为：从该域开始向上直到树根的标记序列，标记之间用句点隔开（类似国外邮政地址的写法）

- 域名的任一后缀也是一个域，同一个机构内的主机具有相同的域名后缀
- **每个节点只需保证其孩子节点的标记不重名**

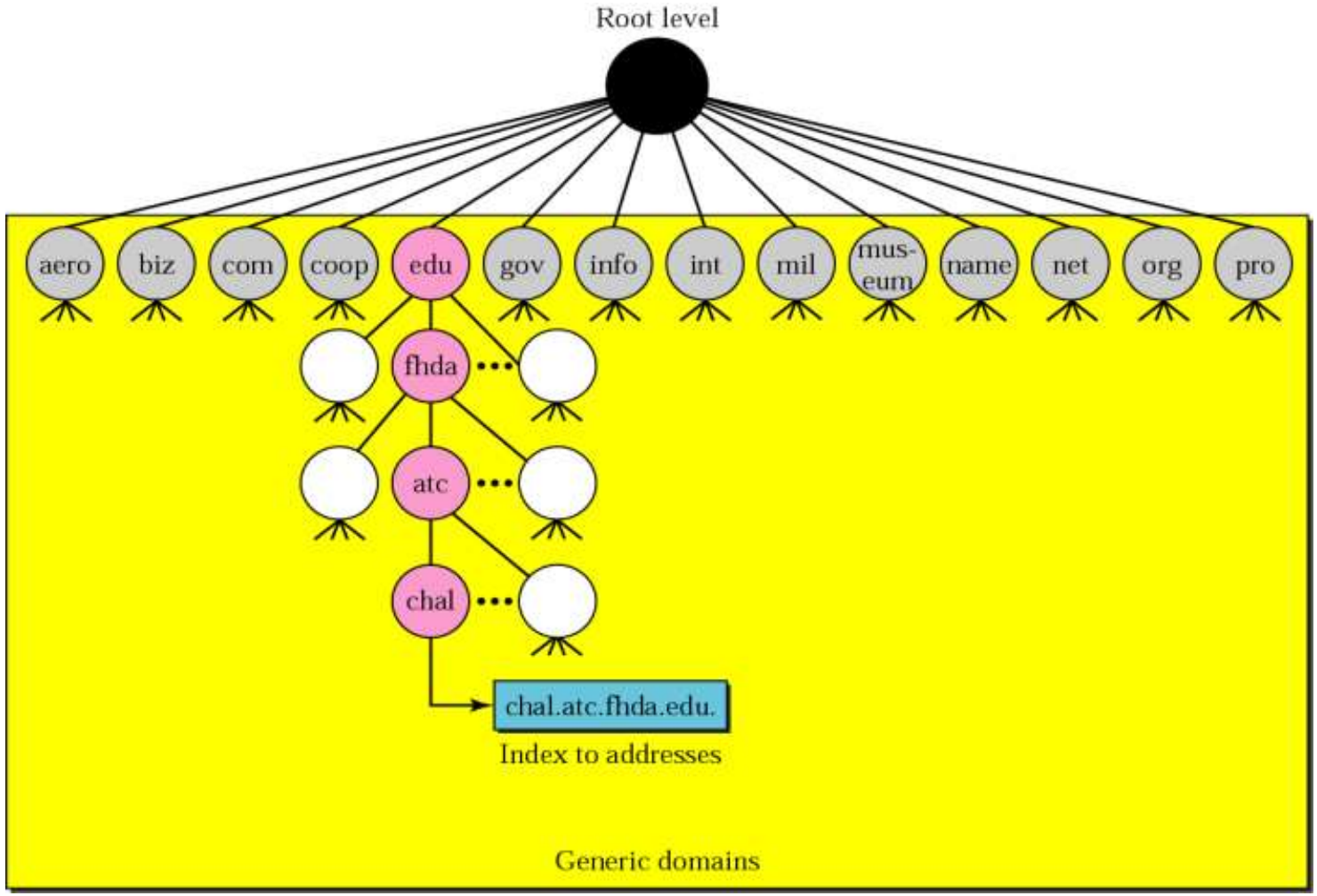
顶级域

- 顶级域分为组织域、国家域和反向域三种



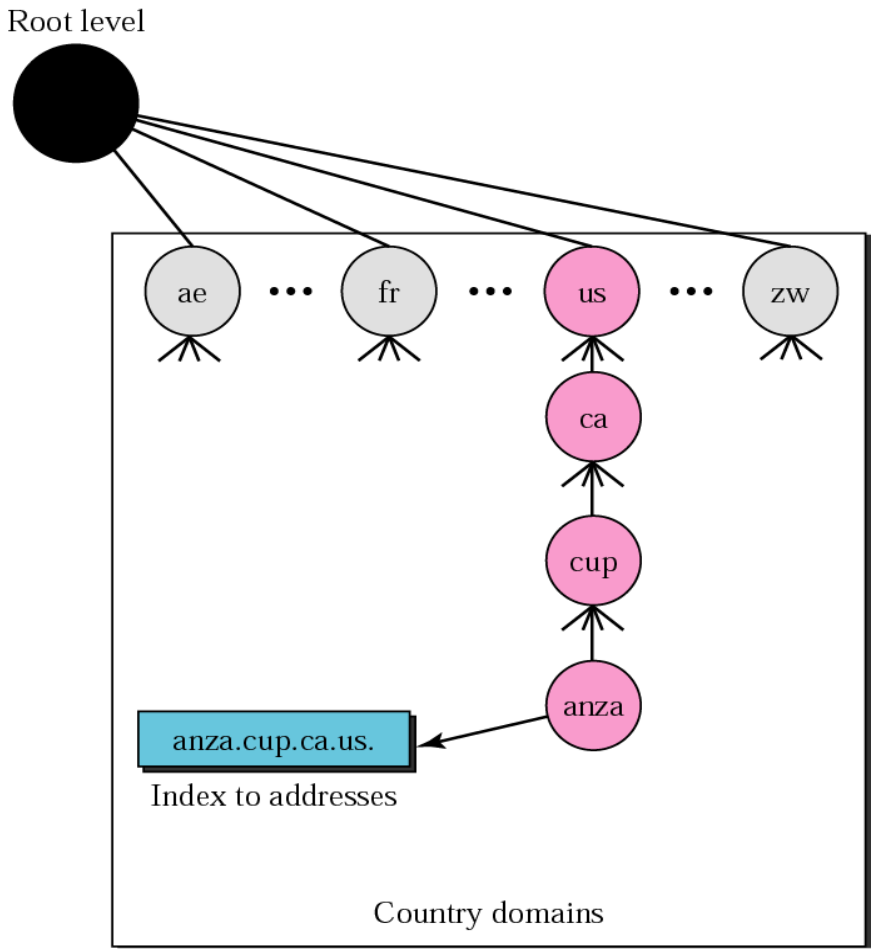
组织域

由美国国内及一些国际组织使用



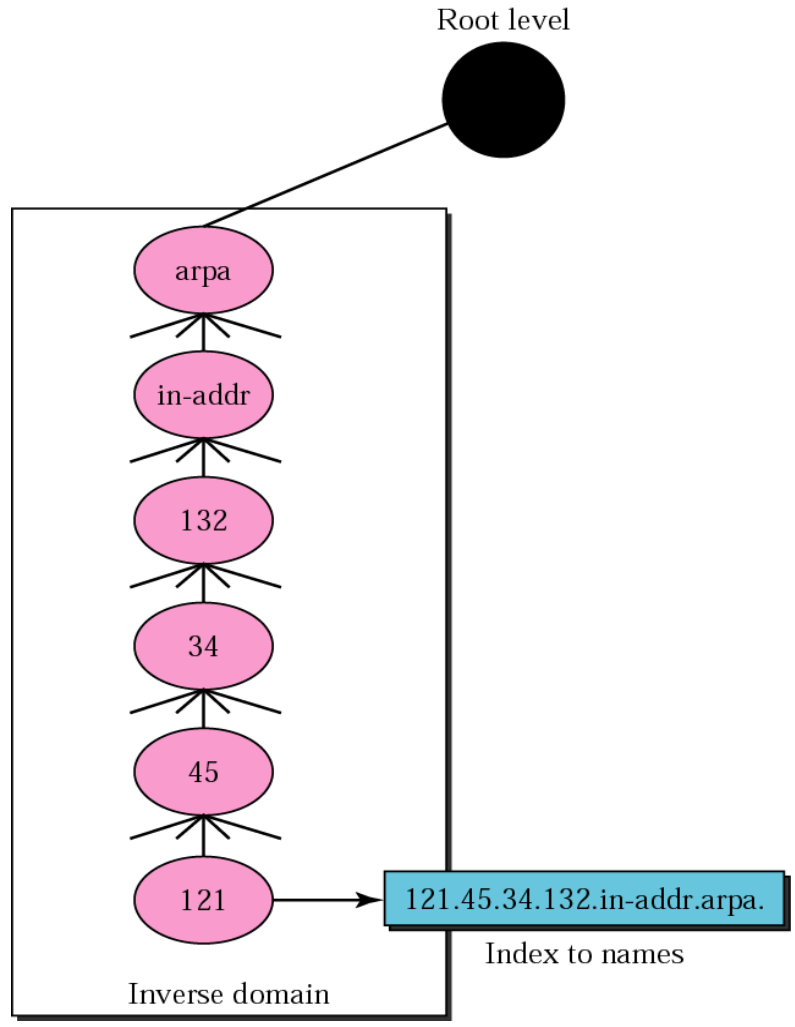
国家域

使用二字符的国家代码，每个国家对应一个

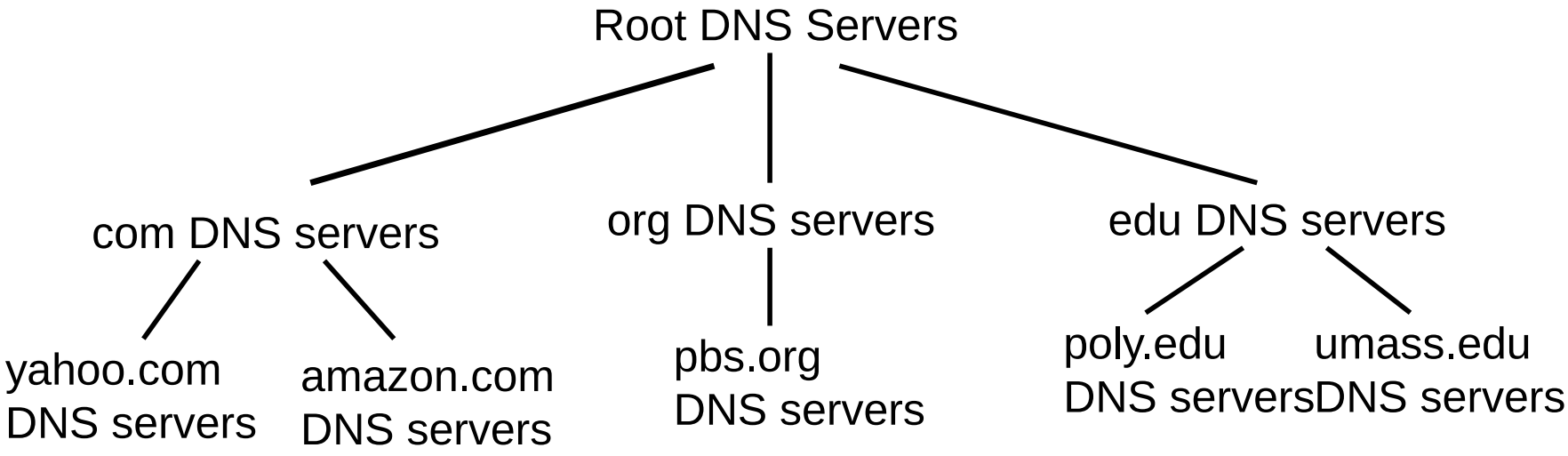


反向域

- 顶级域名为arpa，用来将一个IP地址映射为注册的域名，反向域名解析是为了溯源
- DNS提供了一个反向解析域in-addr.arpa：
  - 欲解析的IP地址表示成像域名一样的一个串，例如，IP地址132.34.45.121表示为 121.45.34.132.in-addr.arpa
  - 以这个字符串作为参数调用解析器
- 电信运营商使用自己的DNS服务器提供相关IP地址的反向解析服务



域名服务器的组织层次

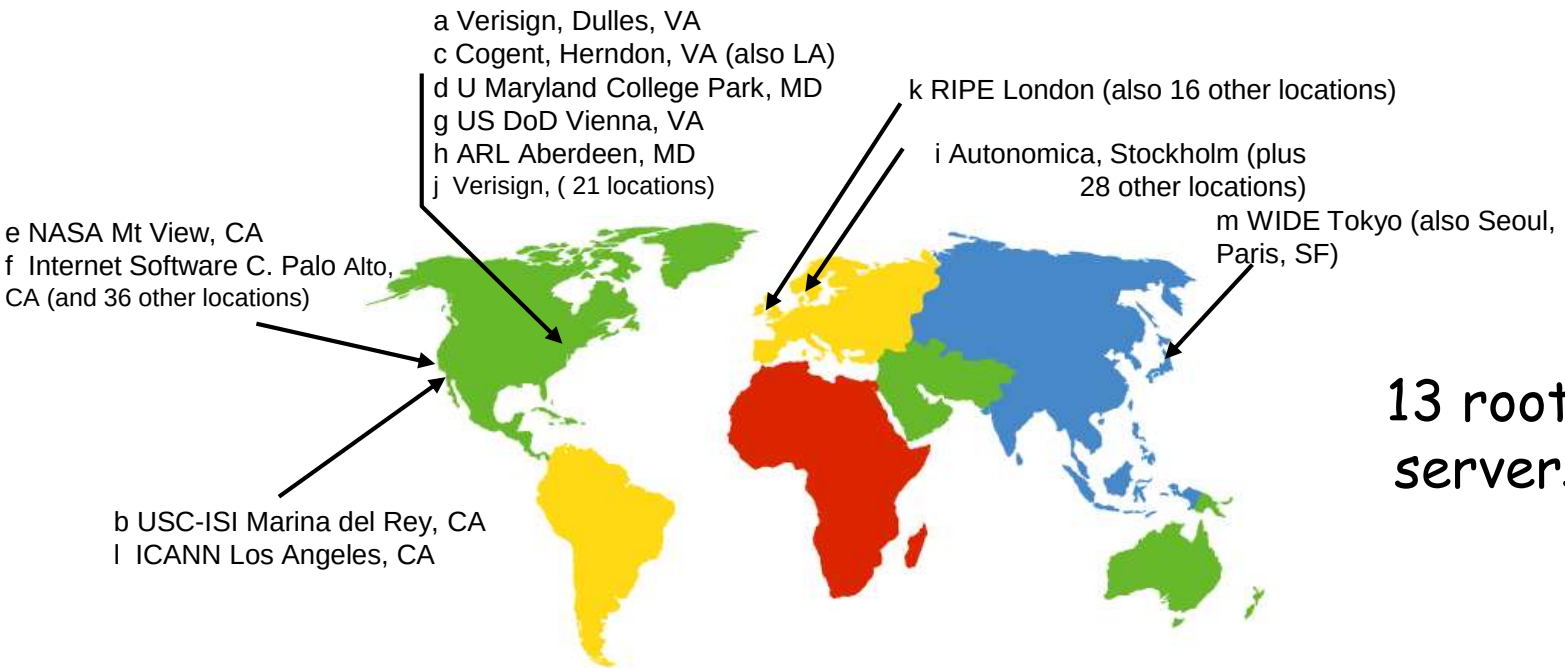


客户想知道www.amazon.com的IP地址:

- DNS客户查询根服务器，得到com域的DNS服务器地址
- DNS客户查询com域的DNS服务器，得到amazon.com域的DNS服务器地址
- DNS客户查询amazon.com域的DNS服务器，得到www.amazon.com的IP地址

根域名服务器

- 全球有13个根服务器（每个根服务器是一个服务器集群）
- 根服务器知道所有顶级域服务器的IP地址



13 root name servers worldwide

顶级域服务器，权威服务器

- 顶级域 (Top Level Domain, TLD) 服务器:
  - 每个TLD服务器负责一个顶级域
  - 知道其所有二级子域的域名服务器的地址

- 权威DNS服务器:
  - 机构的DNS服务器，提供机构内部服务器的名字映射
  - 提供一个主域名服务器、一个或多个辅助域名服务器
  - 可由机构维护，也可委托ISP维护

本地DNS服务器

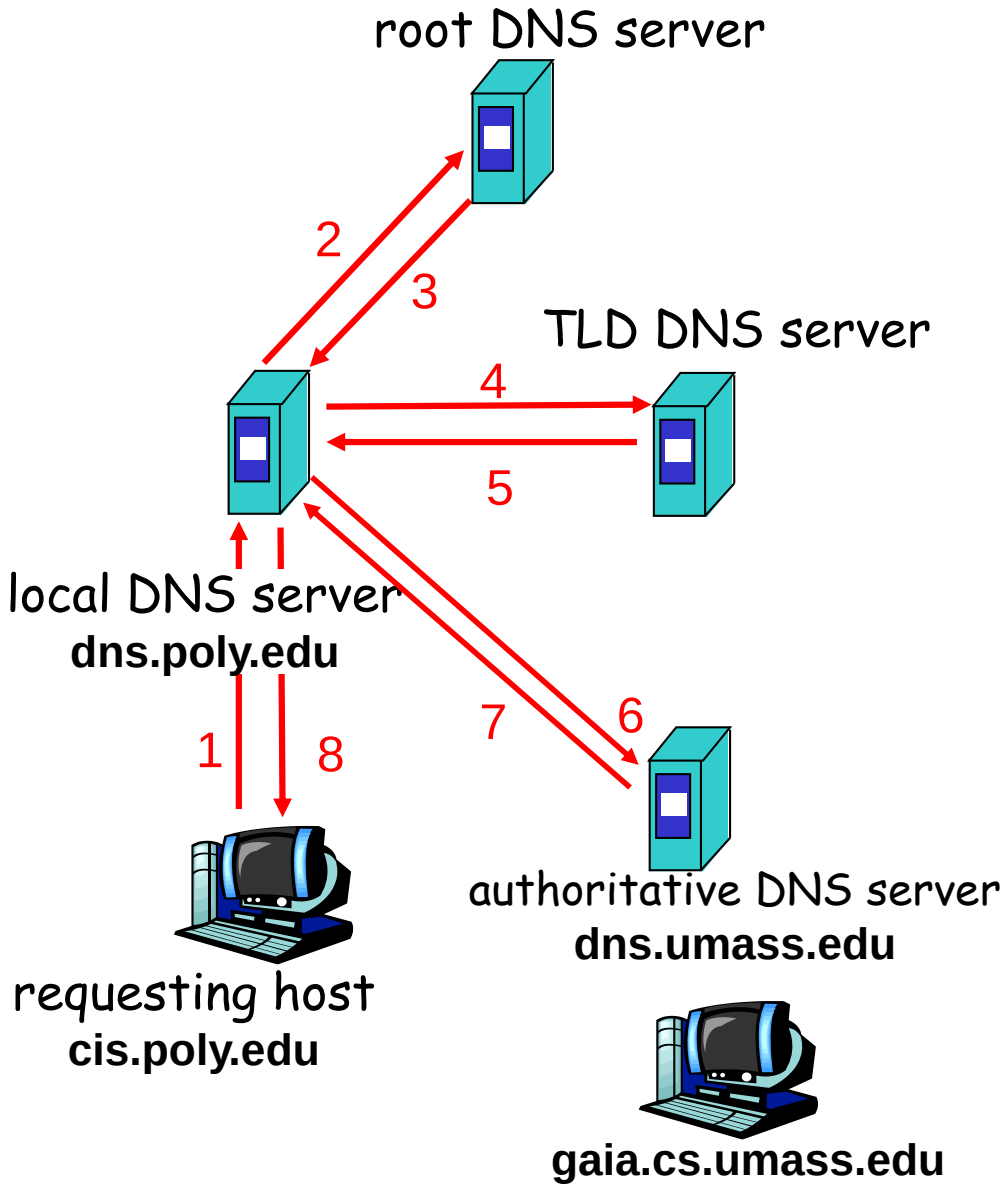
- 严格来说，本地DNS服务器不属于DNS服务器的层次结构
- 每个ISP都有一台本地DNS服务器，也称“默认的DNS服务器”
- 解析器的DNS查询报文实际上发送给本地DNS服务器
- **本地DNS服务器起着代理的作用，**负责将DNS查询报文发送到DNS层次结构中，并将查找结果返回给解析器

域名解析的例子

- cis.poly.edu上的一台主机想知道gaia.cs.umass.edu的IP地址

迭代查询:

- 收到查询报文的服务器将下一个需要查询的服务器地址返回给查询者
- "I don't know this name, but ask this server"



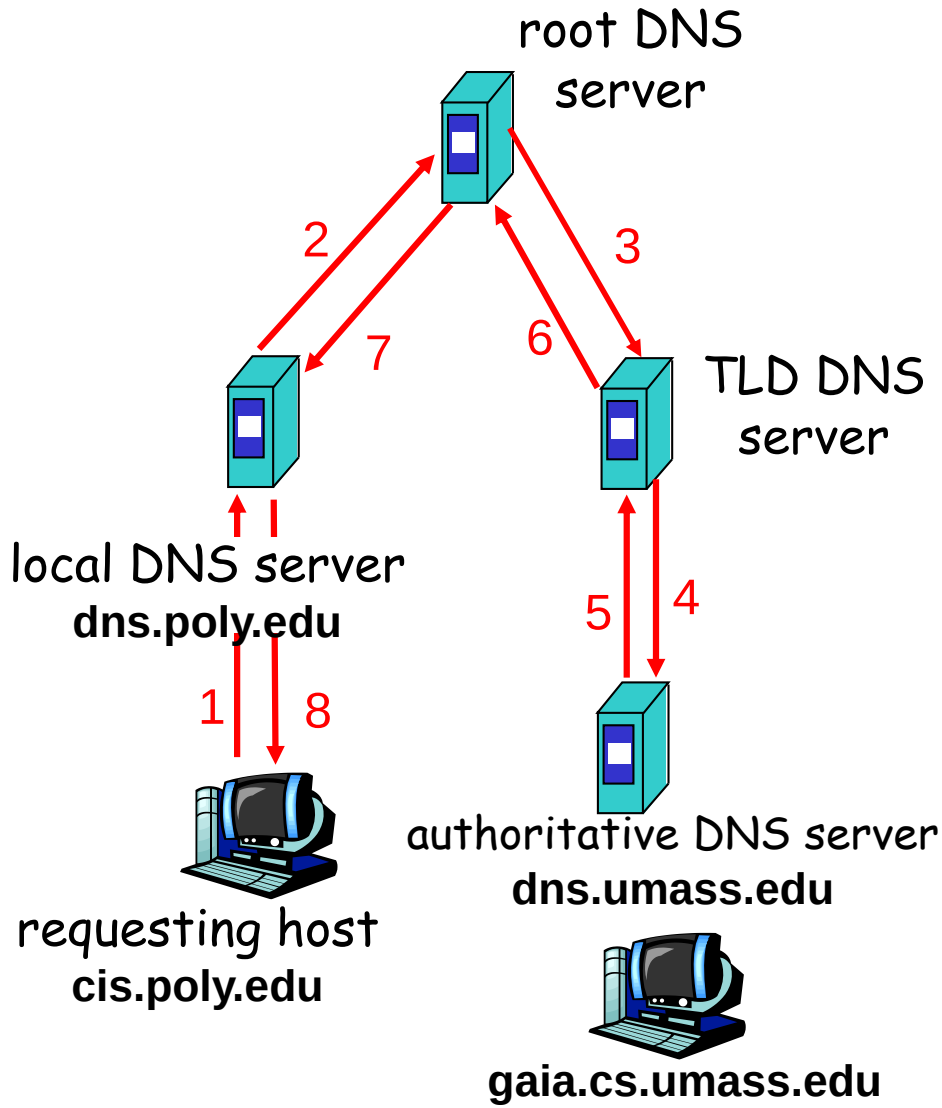
域名解析的例子

递归查询:

- puts burden of name resolution on contacted name server

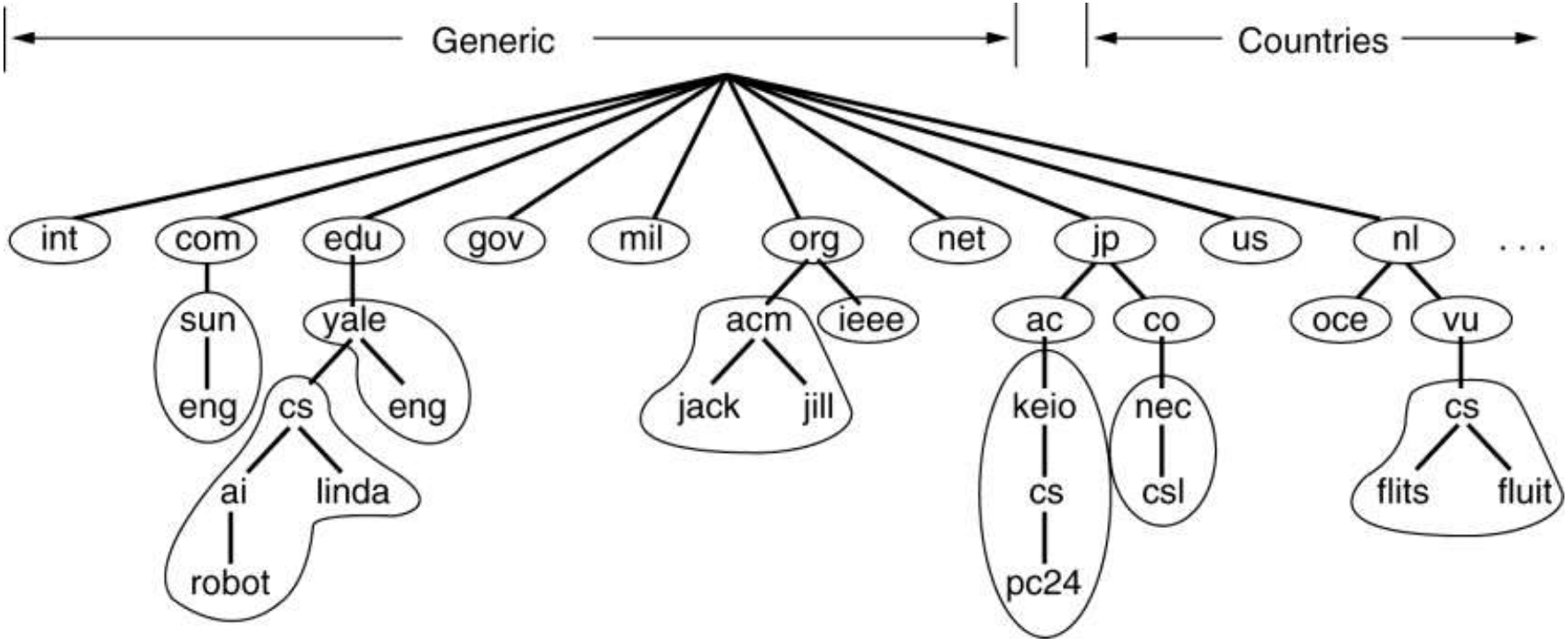
- 实际的域名解析过程:
  - 递归 + 迭代
  - 解析器：递归查询
  - 本地DNS服务器：迭代查询

- 本地DNS服务器的存在，使得DNS对于解析器来说是一个黑盒子



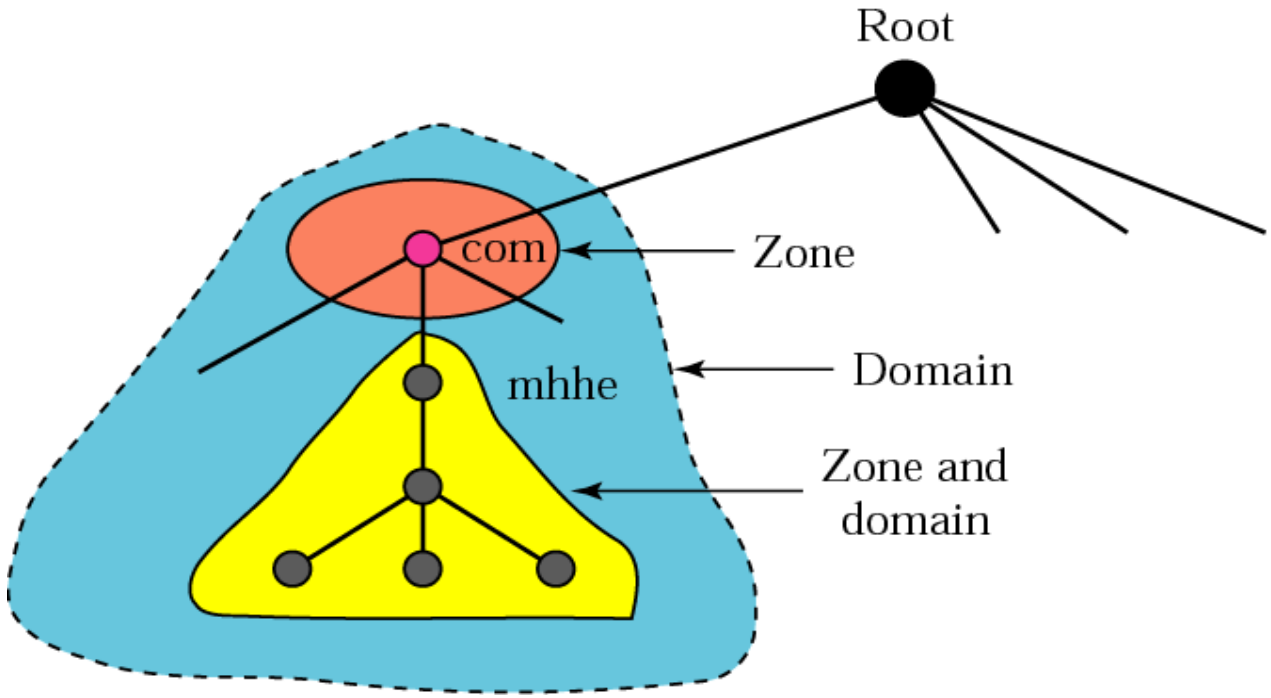
DNS zone

- 整个DNS名字空间被划分成一些不重叠的区域，称为DNS zone，每个zone包含域名树的一部分。
- 在管理上，每个DNS zone代表一个权威域的边界。



物理服务器的层次

- 一个物理服务器保存的信息可能涉及域名空间的若干层，它也可以把它的域划分成若干子域，把其中的一些子域委托给其它服务器
- 实际的物理服务器的层次与域名空间的逻辑层次不同



DNS缓存

- 每当收到一个响应报文，DNS服务器将报文中的映射信息缓存在本地。
- DNS服务器首先使用缓存中的信息响应查询请求
- DNS缓存中的映射在一定时间后被丢弃
- 特别地，本地DNS服务器通常会缓存TLD服务器的IP地址，因而很少去访问根服务器

2.4.3 DNS资源记录

DNS更准确的说法: 存储资源记录（RR）的分布式数据库

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

- Type=A
  - Name：主机名
  - Value：IP地址
- Type=NS
  - Name：域 (e.g. foo.com)
  - value：该域的权威DNS服务器的主机名
- Type=CNAME
  - Name：别名
  - Value：规范名
- Type=MX
  - Name：域(e.g. foo.com)
  - Value：该域的邮件服务器名字

DNS数据库内容示例

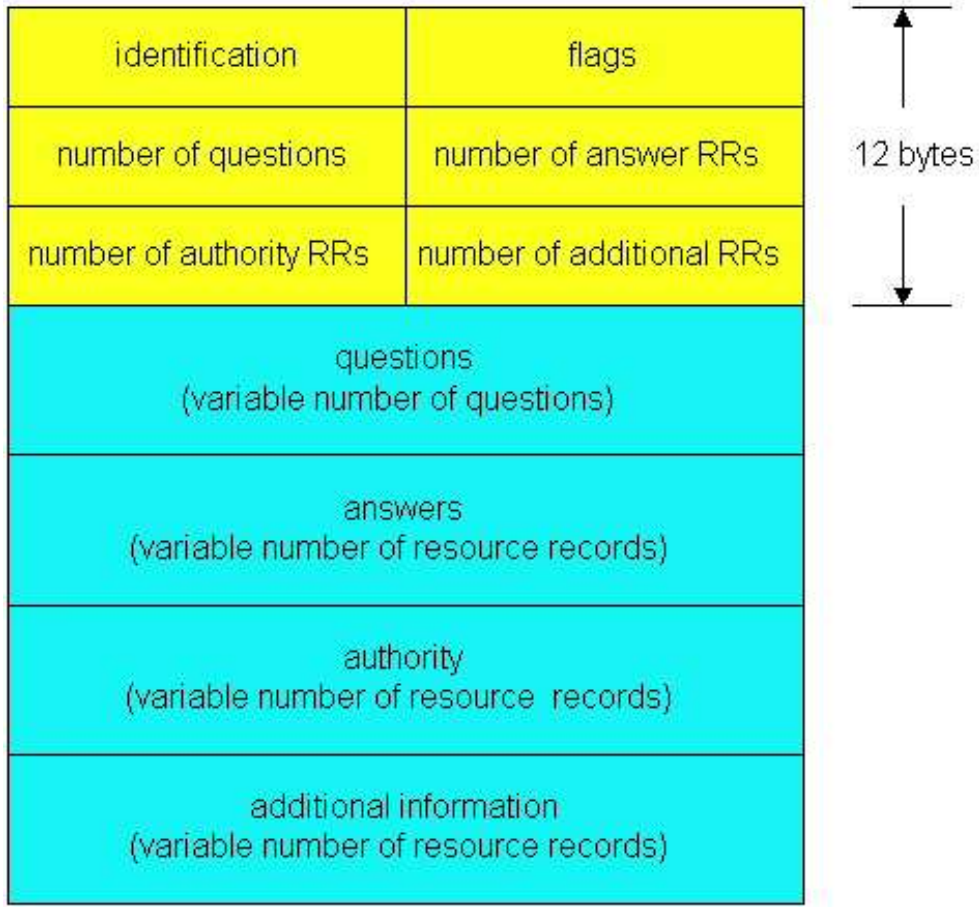
```
; Authoritative data for cs.vu.nl
cs.vu.nl.          86400   IN  SOA     star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.          86400   IN  TXT     "Divisie Wiskunde en Informatica."
cs.vu.nl.          86400   IN  TXT     "Vrije Universiteit Amsterdam."
cs.vu.nl.          86400   IN  MX      1 zephyr.cs.vu.nl.
cs.vu.nl.          86400   IN  MX      2 top.cs.vu.nl.

flits.cs.vu.nl.    86400   IN  HINFO   Sun Unix
flits.cs.vu.nl.    86400   IN  A       130.37.16.112
flits.cs.vu.nl.    86400   IN  A       192.31.231.165
flits.cs.vu.nl.    86400   IN  MX      1 flits.cs.vu.nl.
flits.cs.vu.nl.    86400   IN  MX      2 zephyr.cs.vu.nl.
flits.cs.vu.nl.    86400   IN  MX      3 top.cs.vu.nl.
www.cs.vu.nl.      86400   IN  CNAME   star.cs.vu.nl
ftp.cs.vu.nl.      86400   IN  CNAME   zephyr.cs.vu.nl

rowboat                    IN  A       130.37.56.201
                           IN  MX      1 rowboat
                           IN  MX      2 zephyr
                           IN  HINFO   Sun Unix

little-sister              IN  A       130.37.62.23
                           IN  HINFO   Mac MacOS

laserjet                   IN  A       192.31.231.216
                           IN  HINFO   "HP Laserjet IIISi" Proprietary
```

2.4.4 DNS协议，报文

DNS协议：定义了查询和响应两种报文，查询和响应使用相同的报文格式

报头

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired (q)
  - recursion available (r)
  - reply is authoritative



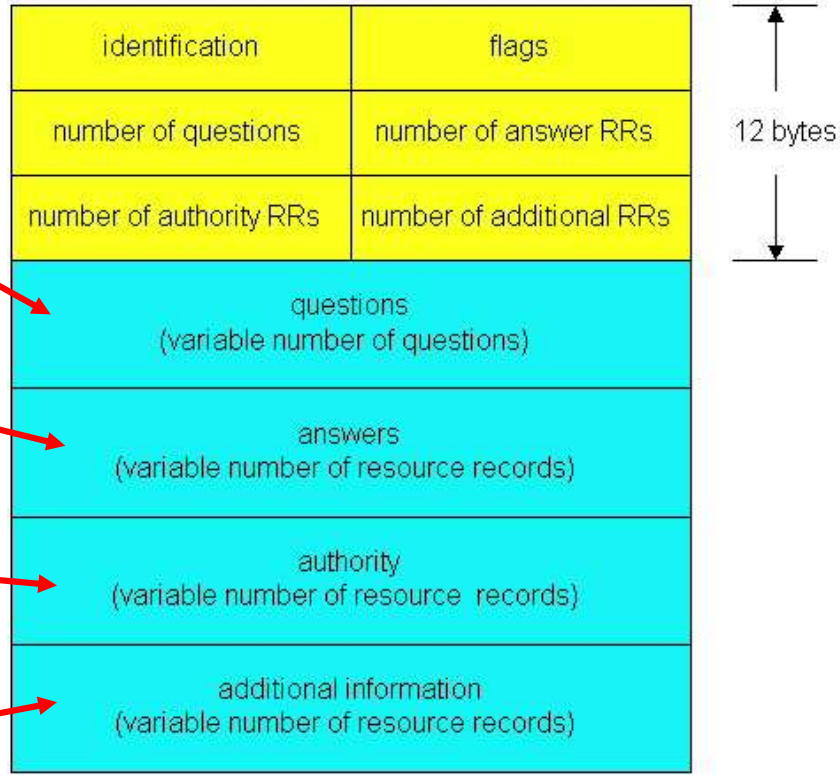
DNS协议，报文

Name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



DNS报文的封装

- DNS主要使用UDP，有时使用TCP，服务器的熟知端口都是53：
  - 当响应报文的长度小于512字节时，使用UDP
  - 当响应报文的长度超过512字节时，使用TCP
  - 当解析器事先不知道响应报文的长度时，先使用UDP；若响应报文的长度超过512字节，服务器截断这个报文，置DNS报文首部的TC标志为1；解析程序打开TCP连接，并重复这个请求，以便得到完整的响应
- 思考：为什么DNS主要使用UDP？

往DNS中插入资源记录

- example: new startup "Network Utopia"
- 向DNS注册机构注册域名"networkutopia.com"
  - 提供权威DNS服务器（主域名服务器，辅助域名服务器）的名字和IP地址
  - 对每个权威域名服务器，注册机构往 com TLD 服务器中插入两条资源记录，例如：

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

- 建立权威DNS服务器，特别是：
  - 建立www.networkuptopia.com的Type A记录
  - 建立networkutopia.com的Type MX记录，以及相应邮件服务器的A记录

DNS的安全问题

- 历史上重大的断网事件，主要都是由根服务器受到攻击引起的：
  - 2016年10月22日，美国最主要DNS服务商Dyn遭遇史上最严重DDoS攻击，大半个美国的用户集体断网
- 谁控制了根服务器，谁就控制了互联网：
  - 我国没有IPv4根服务器，但是建设了IPv6根服务器
- 其它攻击手段：DNS缓存投毒，DNS劫持等
- 僵尸网络通常利用DNS实现僵尸主机与C&C服务器通信

小结

- DNS
  - 提供了一种按层次结构命名主机的方法
  - 实现了一个由DNS服务器构成的分布式数据库
  - 提供了查询域名数据库的应用层协议
- 域名服务器的类型和层次（逻辑层次，物理层次）
- DNS服务的调用方法：
  - 向本地DNS代理的一个RPC调用
  - 递归+迭代的查询方式
- DNS协议：
  - 主要使用UDP，也可以使用TCP，端口号均为53
  - 报文请求/响应交互
- DNS缓存

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

P2P文件共享

一个典型的应用例子

- Alice在她的笔记本电脑上运行P2P客户应用
- 通过ISP连接到因特网上
- 请求歌曲 "Hey Jude"
- P2P客户应用显示拥有该歌曲拷贝的对等方列表
- Alice选择其中的一个对等方，比如Bob
- 文件从Bob的PC机下载到Alice的笔记本电脑
- 当Alice从Bob的PC机下载时，其他用户可能从Alice的笔记本电脑下载
- Alice的P2P应用程序既是一个Web客户，又是一个临时的Web服务器

P2P应用架构: 集中式目录

第一个P2P文件共享服务

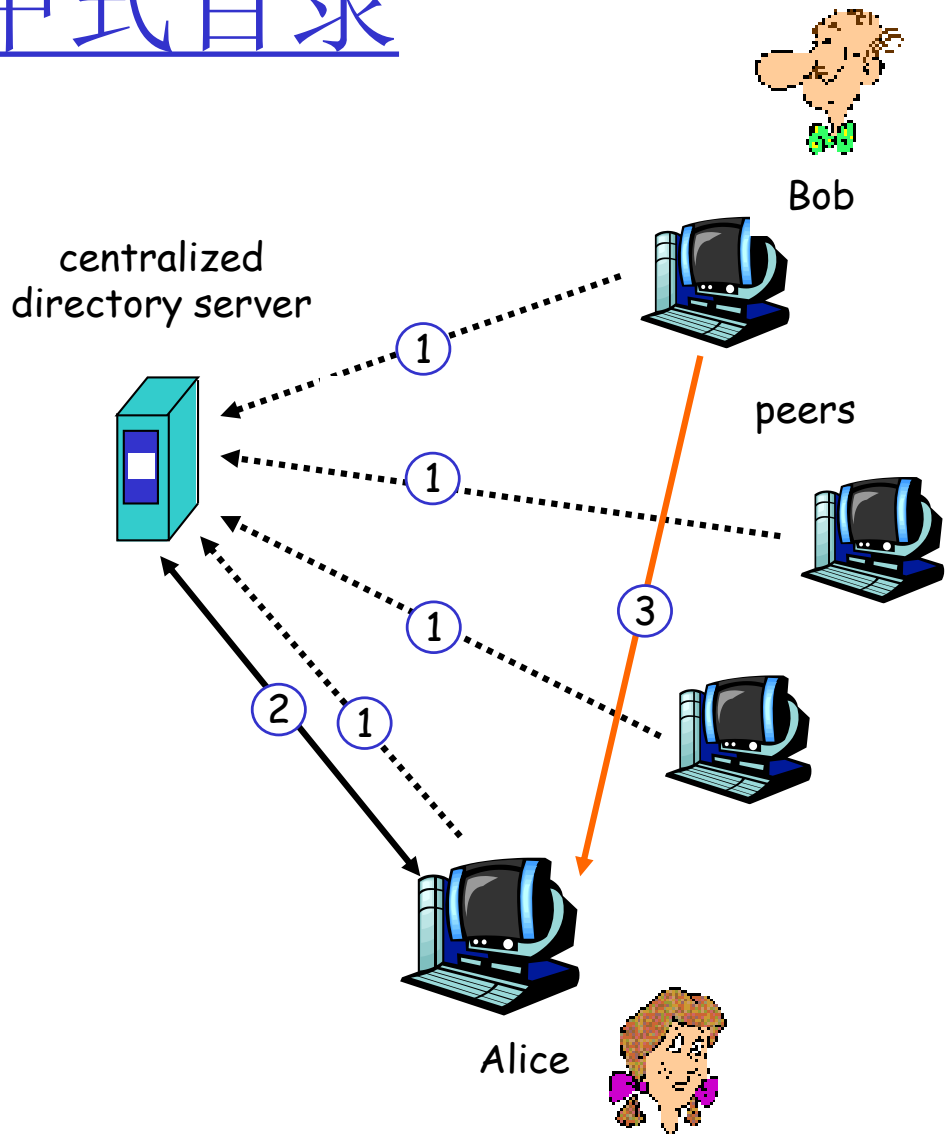
Napster的设计：

1）每个对等方与一个集中式的目录服务器连接，告知：

- 自己的IP地址
- 可以共享的内容

2) Alice查询歌曲 "Hey Jude"

3) Alice从Bob下载音乐文件



P2P: 集中式目录的问题

- 单点失效
- 目录服务器成为性能瓶颈
- 由于版权问题，易成为诉讼的对象

file transfer is decentralized, but locating content is highly centralized

P2P应用架构：查询洪泛

Gnutella

- 完全分布
  - 没有集中式服务器
- 公共域协议
- 有许多Gnutella客户软件

overlay network: graph

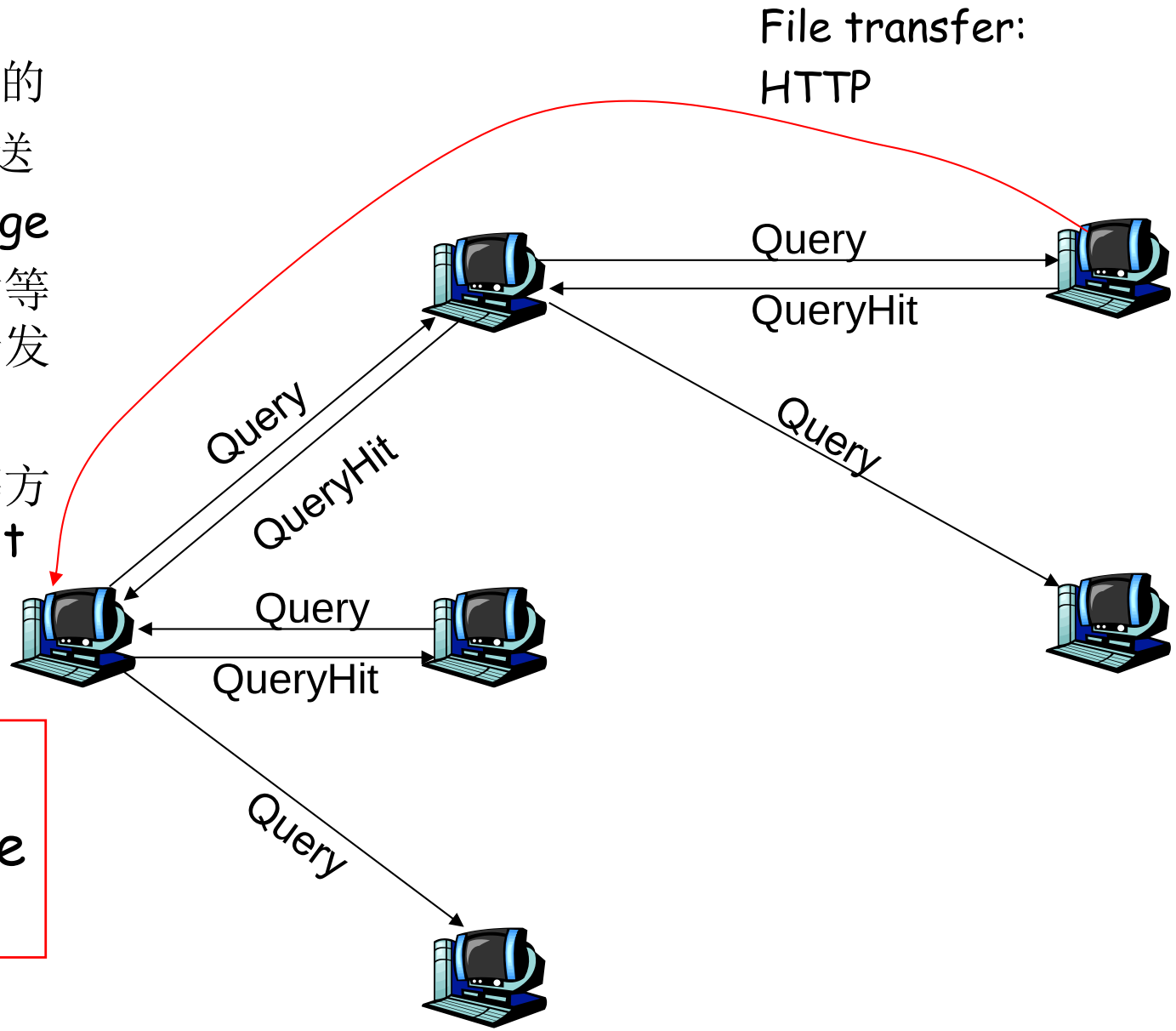
- 如果对等方X与Y之间存在一条TCP连接，称它们之间存在一条边
- 所有活跃的对等方以及它们之间的边，形成一个覆盖网络（overlay net）
- 边：虚拟链路
- 一个对等方通常有 < 10 个的覆盖网邻居

Gnutella: 对等方加入

1. 新加入的对等方Alice必须在Gnutella网上发现其它对等方: 使用一个候选对等方列表（客户软件中自带）
2. Alice依次和候选列表中的对等方尝试建立TCP连接
3. 洪泛: Alice向每个覆盖网邻居发送Ping报文；每个邻居向他的覆盖网邻居转发Ping报文……
   - 收到Ping报文的对等方向Alice发送Pong报文
4. Alice收到许多Pong报文，从而可以建立更多的TCP连接

Gnutella: protocol

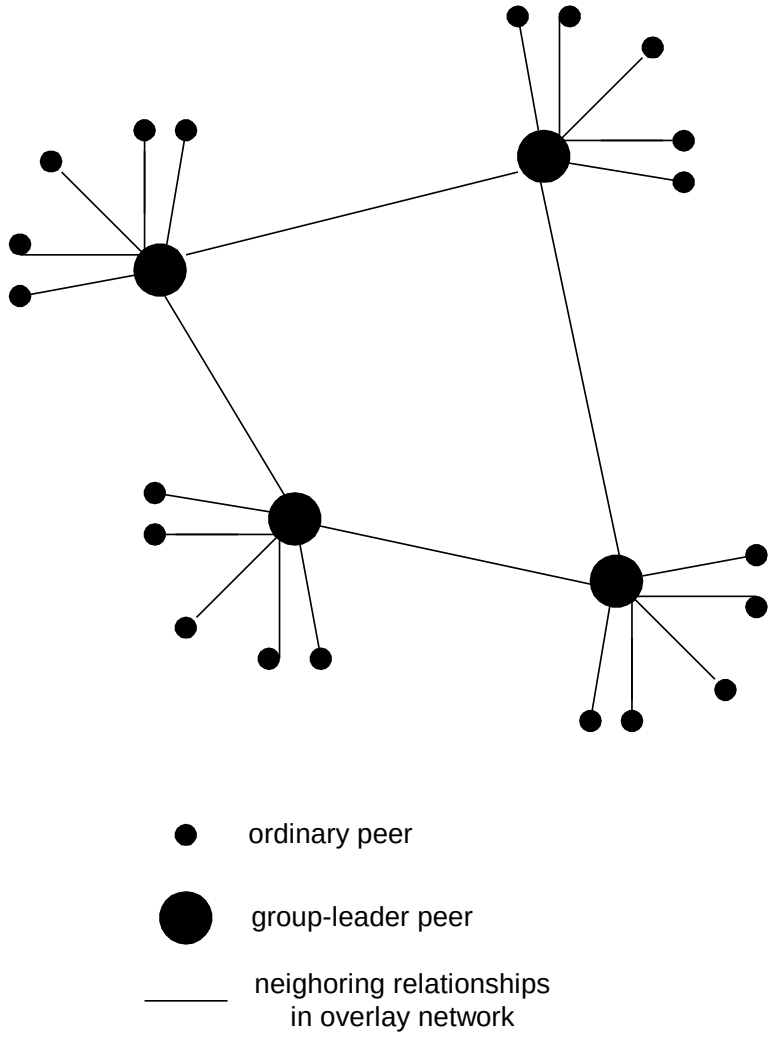
- 查询方在已有的TCP连接上发送Query message
- 收到消息的对等方向其邻居转发查询报文
- 有内容的对等方返回QueryHit



Scalability:
limited scope
flooding

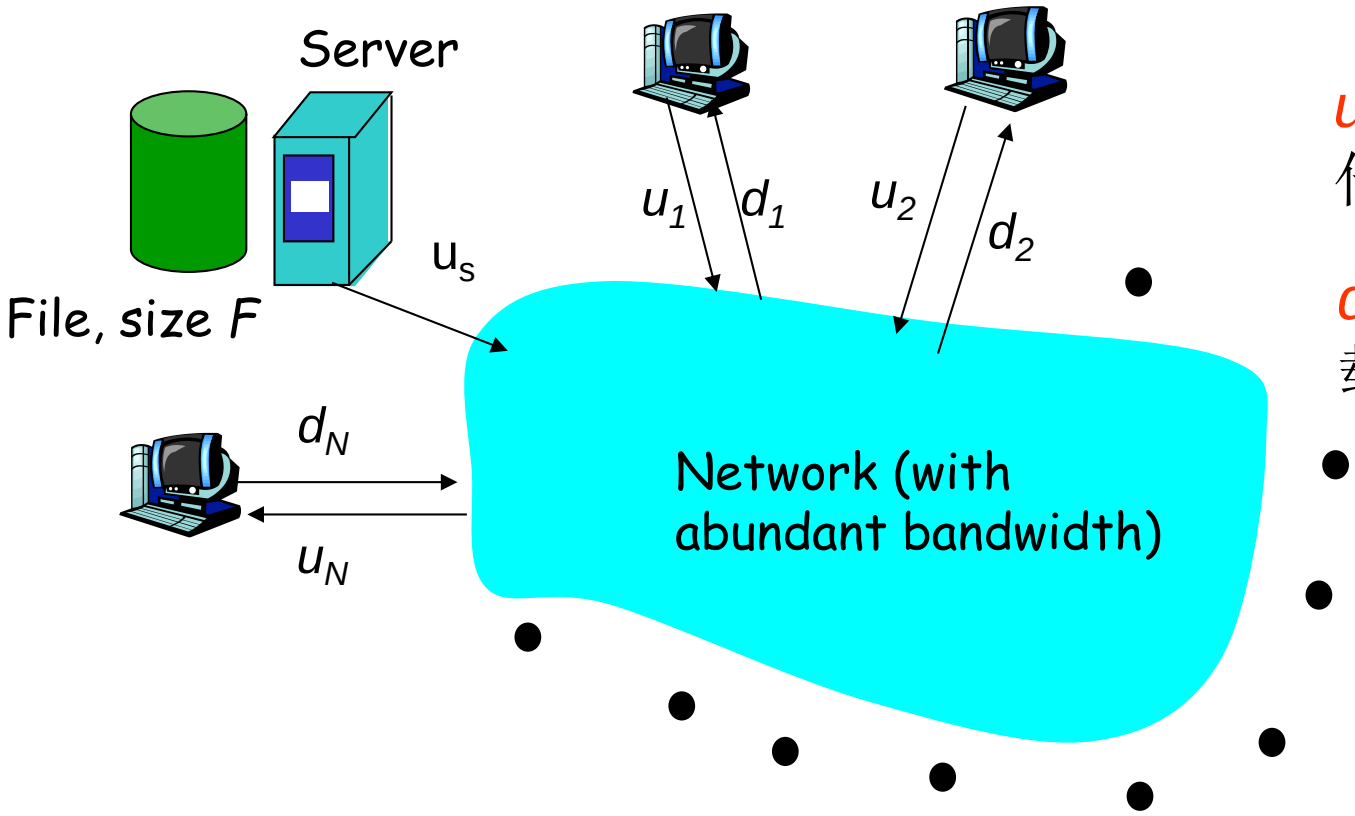
P2P应用架构：层次结构的覆盖网

- 介于集中式目录和查询洪泛之间
- 每个对等方或是一个超级节点（*group leader*），或是从属于某个超级节点的普通节点：
  - 每个普通节点与其超级节点之间建立TCP连接
  - 某些超级节点之间建立TCP 连接
- 超级节点知道其所有从属节点的共享内容列表



比较客户-服务器和P2P架构

Question：将文件从一台服务器分发到网络中的N个终端，需要多少时间？



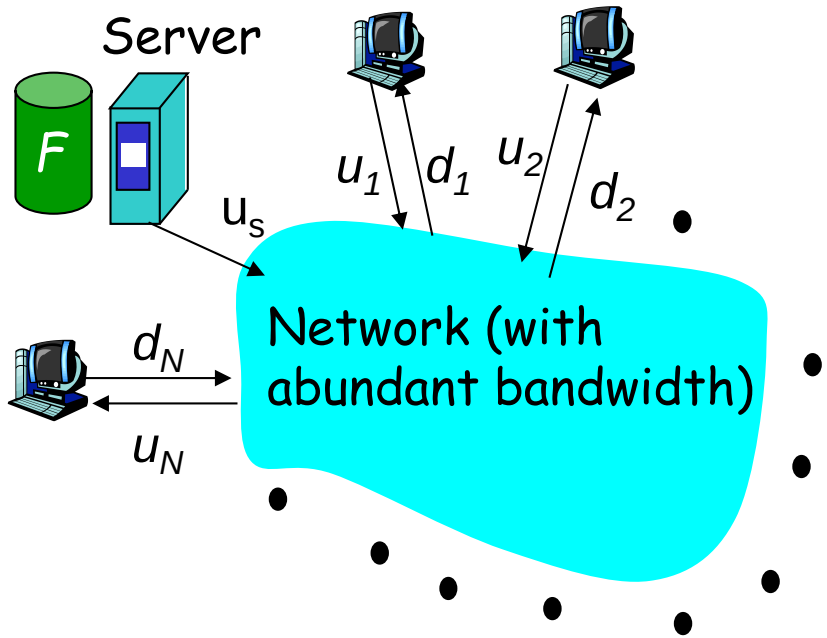
u_s: 服务器上传带宽

u_i: 第 i 个终端的上传带宽

d_i: 第 i 个终端的下载带宽

客户-服务器架构的文件分发时间

- 假设：
  - 所有瓶颈都在接入链路
  - 服务器和客户的上传和下载带宽全都用于分发文件
- 服务器顺序地发送N 个文件拷贝，耗时NF/u_s
- 第 i 个客户花费F/d_i下载文件



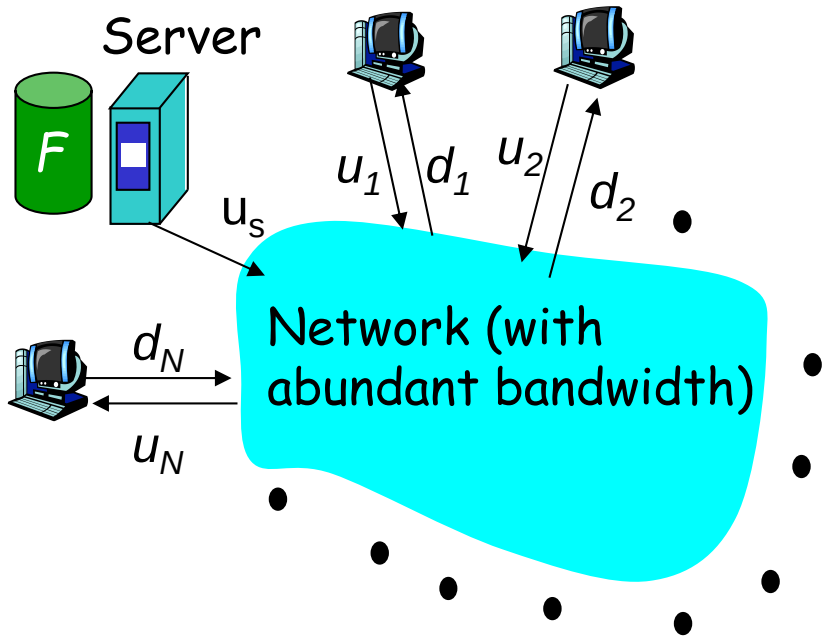
将文件F分发给N 个客户的耗时

$$d_{cs} >= \max\{ NF/u_s, F/\min_i(d_i) \}$$

increases linearly in N (for large N)

P2P架构的文件分发时间

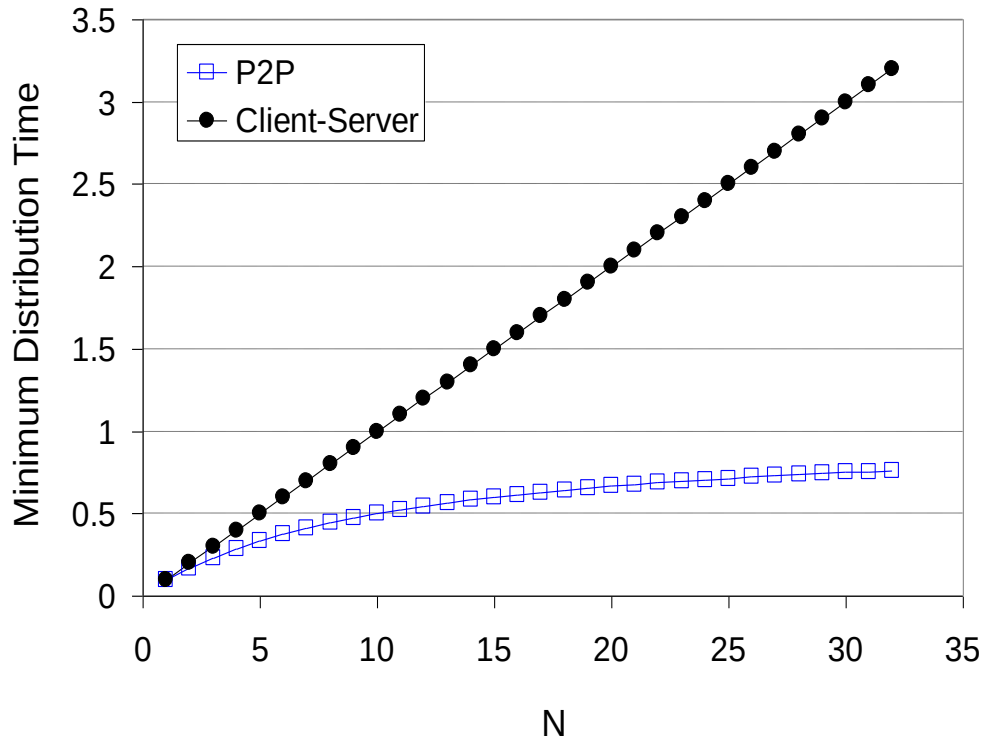
- 服务器至少必须发送一个文件拷贝，耗时F/u_s
- 第 i 个客户至少耗时F/d_i下载文件
- 总共NF bits 必须在网络中被上传，网络中总的上传带宽为: $u_s + \Sigma u_i$



$$d_{P2P} = \max \{ F/u_s, F/\min(d_i)_i, NF/(u_s + \sum_{i=1,N} u_i) \}$$

文件分发时间比较

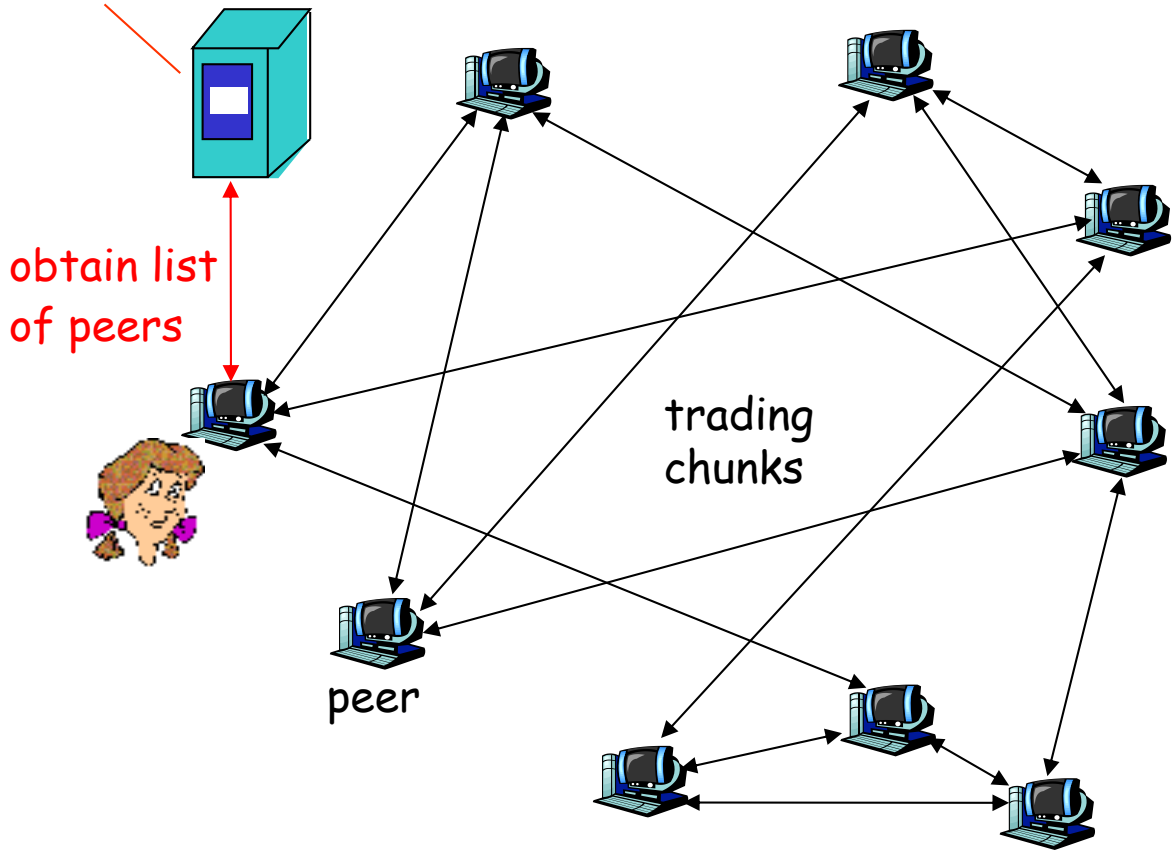
- 假设：
  - 客户的上传带宽均为u，F/u=1 hour，us=10u，dmin>us（下载速率不是瓶颈）
- 客户-服务器架构：
  - 分发时间随N线性增大
- P2P架构：
  - 最小分发时间总是小于客户-服务器架构
  - 对于任意的N，最小分发时间总是小于1小时



P2P案例学习: BitTorrent

tracker:
跟踪洪流中的对等方

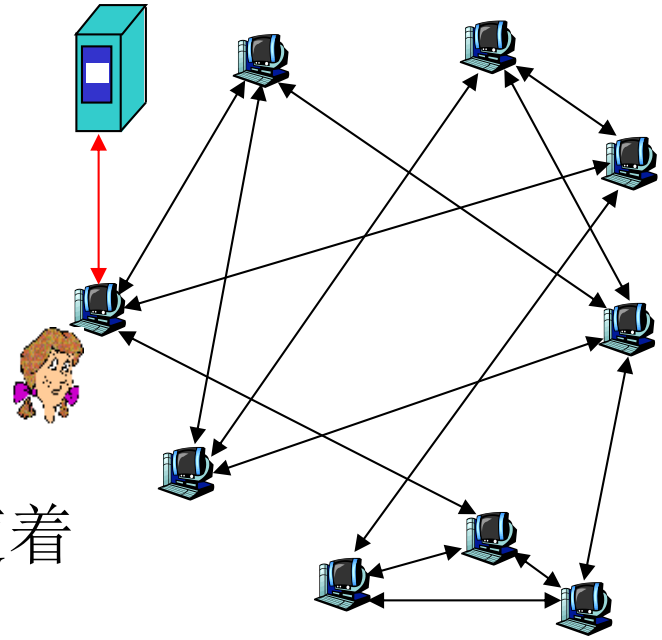
Torrent（洪流）：
参与一个特定文件
分发的对等方集合



对等方加入洪流：

- 向跟踪器注册，获得一个对等方列表
- 尝试向列表中的对等方建立TCP连接

BitTorrent (概述)



- 文件被划分成长为256KB的块
- 对等方加入洪流时没有数据块，但随着时间的推移逐步积累
- 对等方在下载数据块的同时，也向其它对等方上载数据块
- 对等方可以动态加入或离开系统
- 一旦对等方获取了整个文件，它可以（自私地）离开，也可以（无私地）留在网络中，为其它对等方上传文件块

BitTorrent (操作)

Pulling Chunks

- 在任何给定时刻，不同的对等方拥有不同的数据块子集
- 周期性地，对等方（如Alice）询问每个邻居他们拥有的数据块集合
- Alice向邻居请求她缺少的数据块：
  - 最稀罕优先：优先请求在邻居中拷贝数量最少的数据块

Sending Chunks: tit-for-tat

- Alice选择当前向其发送数据最快的4个邻居，响应他们的数据块请求
  - 每隔10秒，重新评估向其提供数据最快的4个邻居
- 每隔30秒，随机选择另一个对等方（如Bob）响应其请求
  - Alice可能成为向Bob上载最快的4个邻居之一
  - Bob也可能成为向Alice上载最快的4个邻居之一

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

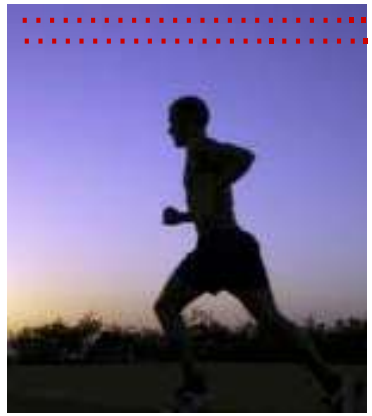
视频流和内容分发网络

- 视频已成为因特网流量的主体：
  - 2015年，Netflix和YouTube分别消耗了住宅ISP流量的 37%和16%
  - 有大约10亿YouTube用户、7500万Netflix用户
- 挑战一：如何能够支持10亿用户?
  - 单台视频服务器不可行
- 挑战二：网络环境异构
  - 不同用户的接收能力不同，例如，使用有线网络或无线网络，高带宽或低带宽
- 解决方案：采用分布式应用层基础设施

视频(video)的概念

- 视频是以恒定速率播放的图像序列：
  - 比如，24帧/秒
- 数字图像是一个像素阵列
  - 每个像素用一些比特来表示
- 图像编码技术利用图像的帧内冗余和帧间冗余来减少需要使用的比特数：
  - 空间冗余：帧内
  - 时间冗余：帧间

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)



frame *i*

frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

视频编码速率

- CBR: (constant bit rate):
  - 编码速率固定
- VBR: (variable bit rate):
  - 编码速率可变
- 视频编码标准:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)

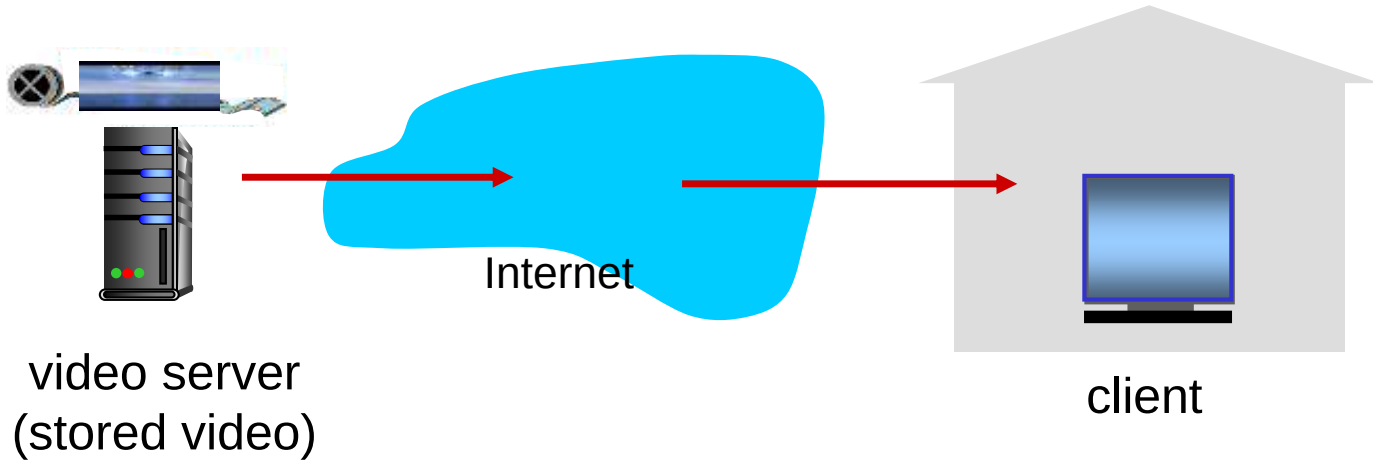
frame *i*

frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

流式存储视频：一个简单的实现

一个简单的实现:



- 视频作为一个普通文件，保存在HTTP服务器中
- 服务器与客户建立TCP连接，发送文件
- 视频应用周期性地从应用缓存中取帧、解码、展示
- 问题：所有客户使用相同的编码速率

流式多媒体: DASH

- DASH: Dynamic, Adaptive Streaming over HTTP
- 服务器:
  - 将视频文件划分成多个块
  - 每个块以不同的码率编码和存储
  - 元文件为不同的块提供URL
- 客户:
  - 周期性地测量到服务器的网络带宽
  - 查询元文件，每次请求一个块
    - 选择当前带宽可支持的最大码率
    - 不同时刻可以选择不同码率的块
- 客户端能够“智能地”确定:
  - 什么时候请求块（避免缓存不足或溢出）
  - 请求什么码率的块（获得当前最好的视频质量）
  - 向谁请求块（离客户最近或具有最高带宽的URL）

如何将内容流式传输给同时在线的大量用户？

方法1: 使用一个巨型服务器（不可行）

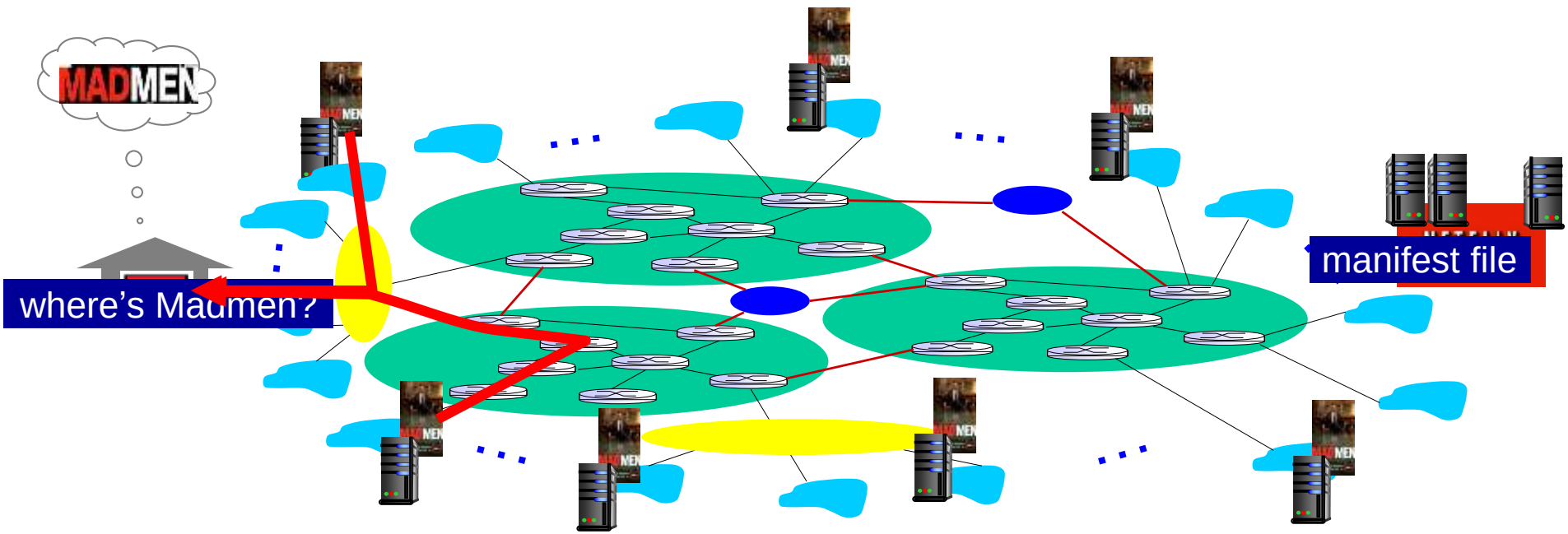
- 单点故障
- 网络拥塞点
- 远端用户传输距离长，跨多个ISP，带宽低
- 同一条链路上传输多个视频拷贝，浪费带宽

方法2: 在地理上分布的多个站点存储和提供服务 *(CDN)*:

- *enter deep:* 将CDN服务器深入部署到大量接入网中，靠近用户
  - Akamai拥有1700个站点
- *bring home:* 将少量（几十个）较大的集群部署在靠近接入网的POP中
  - Limelight采用此法

内容分发网络(CDN)

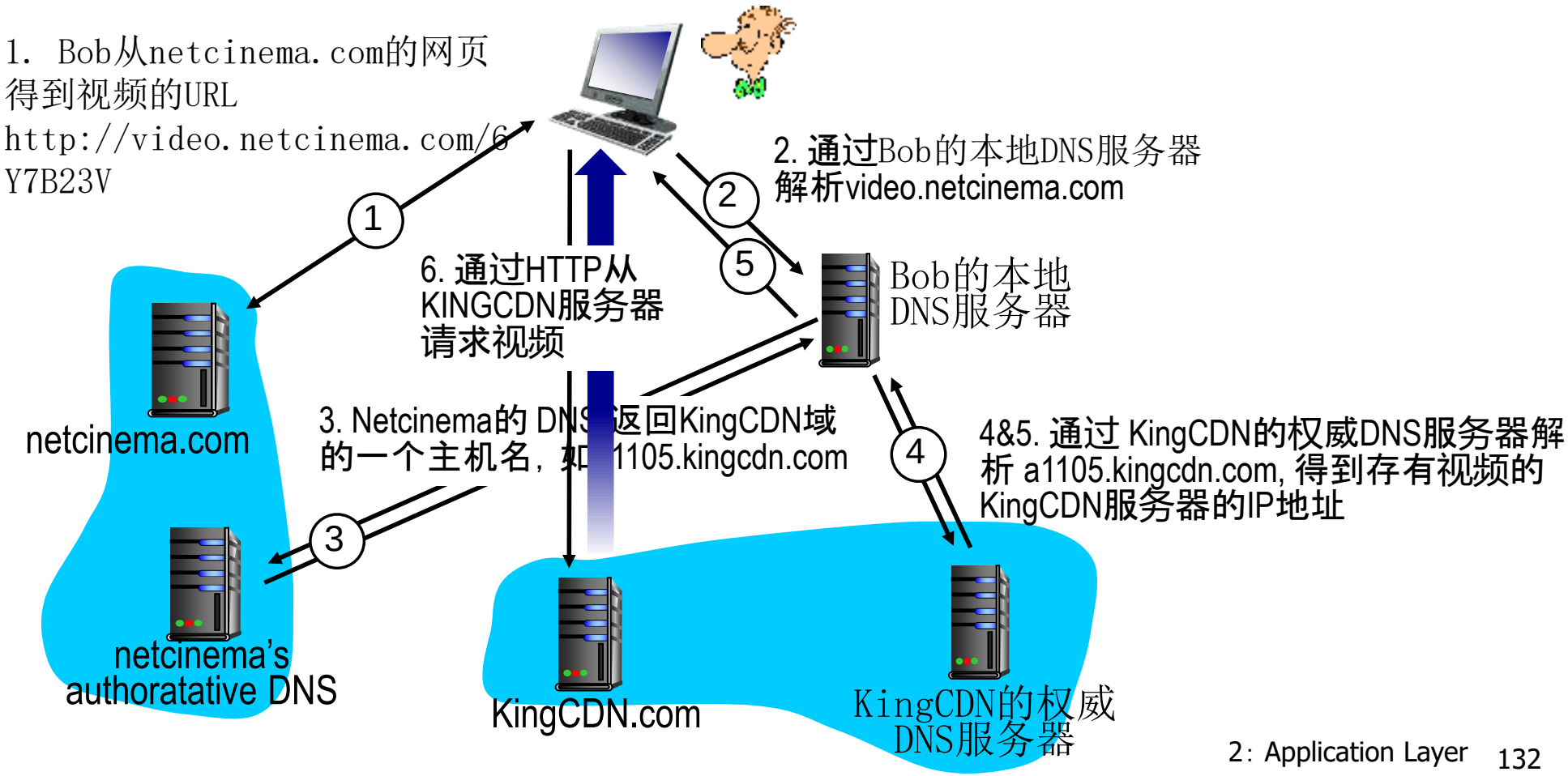
- 在多个CDN站点存储内容的拷贝
- 用户从CDN请求内容:
  - 用户请求被定向到附近的站点，获取内容
  - 或网络拥塞时，可向不同的站点请求内容拷贝



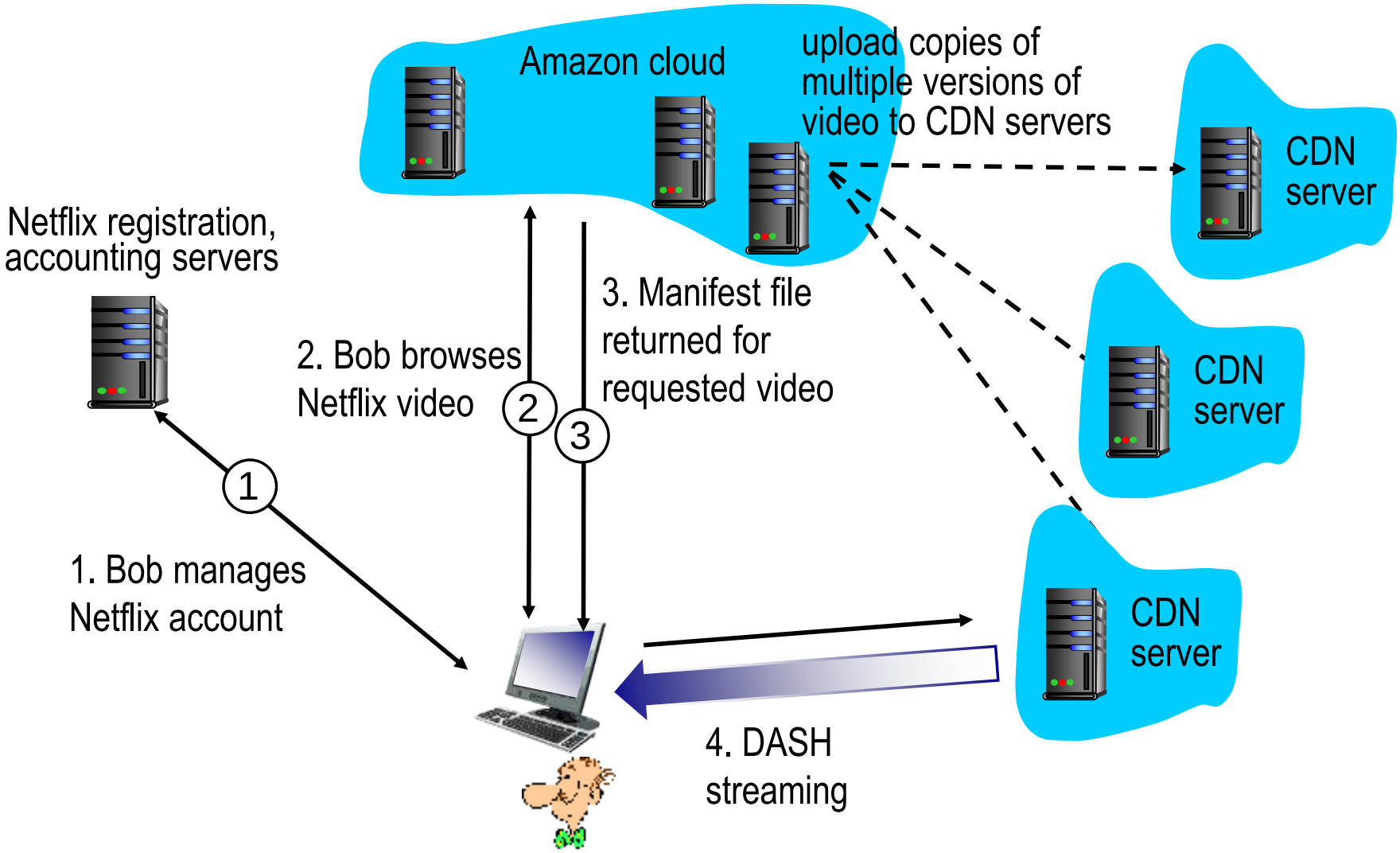
CDN内容访问：a closer look

Bob (client) 从 http://video.netcinema.com/6Y7B23V请求视频

- 视频保存在 KingCDN.com域中的一台服务器上



Case study: Netflix



Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

补充：File Transfer and FTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

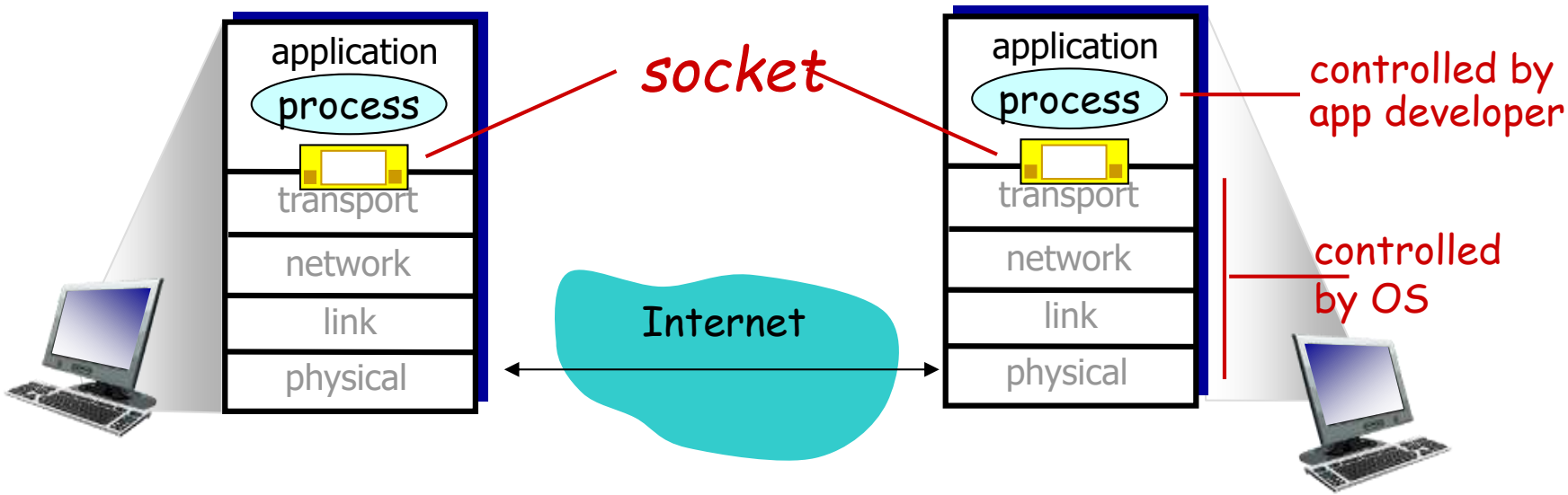
2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

套接字编程

- 创建网络应用程序，需要：
  - 编写一个客户程序、一个服务器程序
  - 客户和服务器可以利用套接字收、发报文
- **套接字:** 应用进程和传输层之间的"门"



Socket API

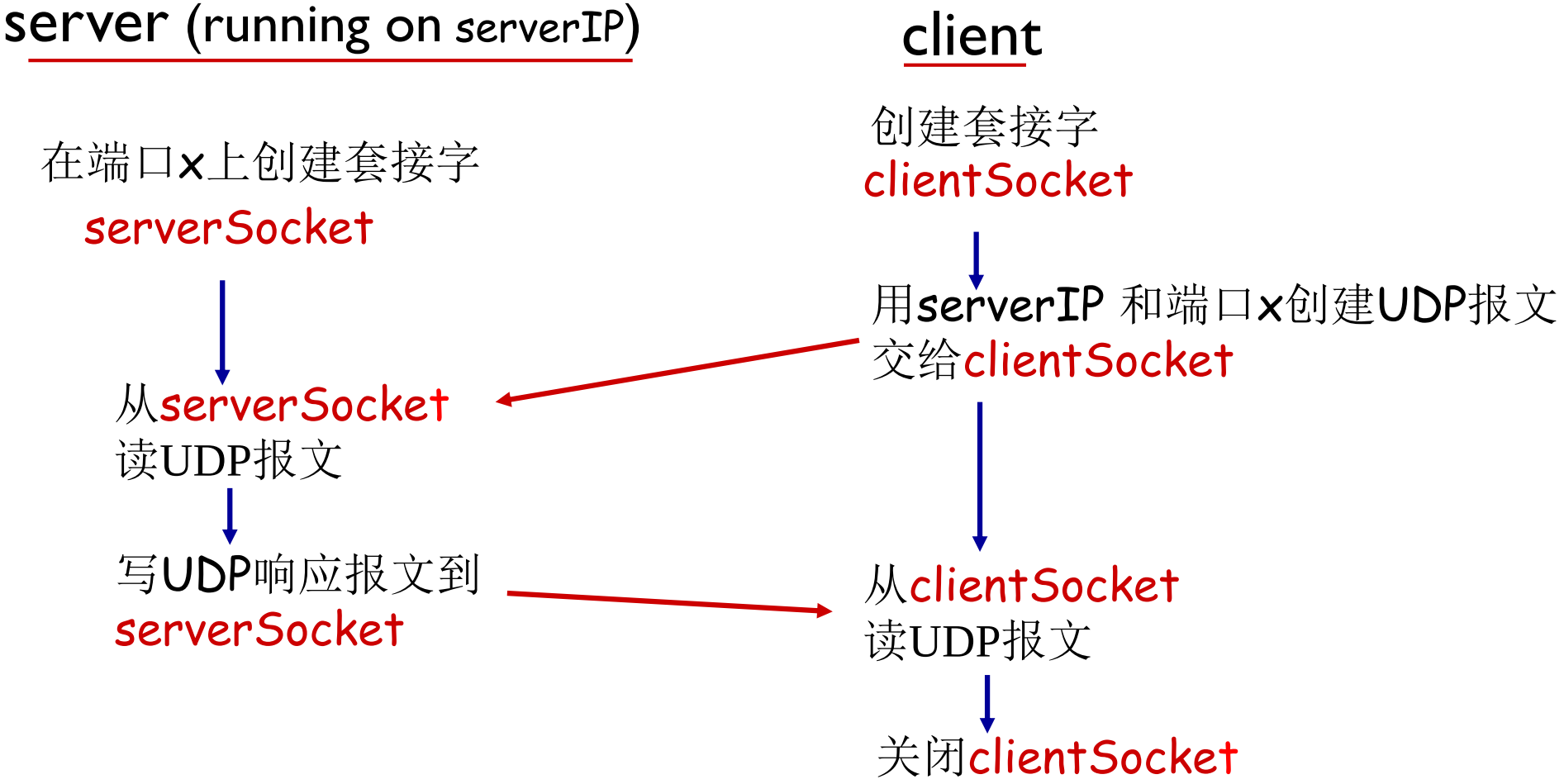
- 1981年在BSD UNIX 4.1中引入，此后成为编写因特网程序的标准接口
- 应用需显式地创建、使用和释放套接字
- 采用客户-服务器模式

- 应用通过socket API可以调用两种传输服务：
  - 不可靠的数据报服务：由UDP协议实现
  - 可靠的字节流服务：由TCP协议实现

本节使用的一个应用例子

1. 客户程序从键盘读入一行字符（数据），发送给服务器
2. 服务器接收数据，将小写字符转换成大写字符
3. 服务器将修改后的数据发送给客户
4. 客户接收修改后的数据，在屏幕上显示出来

2.7.1 UDP套接字编程



Example app: UDP server

Python UDPServer

include Python's socket library

```
from socket import *
serverPort = 12000
```

create UDP socket

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

bind socket to local port number 12000

```
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
```

loop forever

```
while 1:
```

Read from UDP socket into message, getting client's address (client IP and port)

```
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
```

send upper case string back to this client

```
    serverSocket.sendto(modifiedMessage, clientAddress)
```

Example app: UDP client

Python UDPClient

include Python's socket library

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

create UDP socket for server

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

get user keyboard input

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

Attach server name, port to message; send into socket

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

read reply characters from socket into string

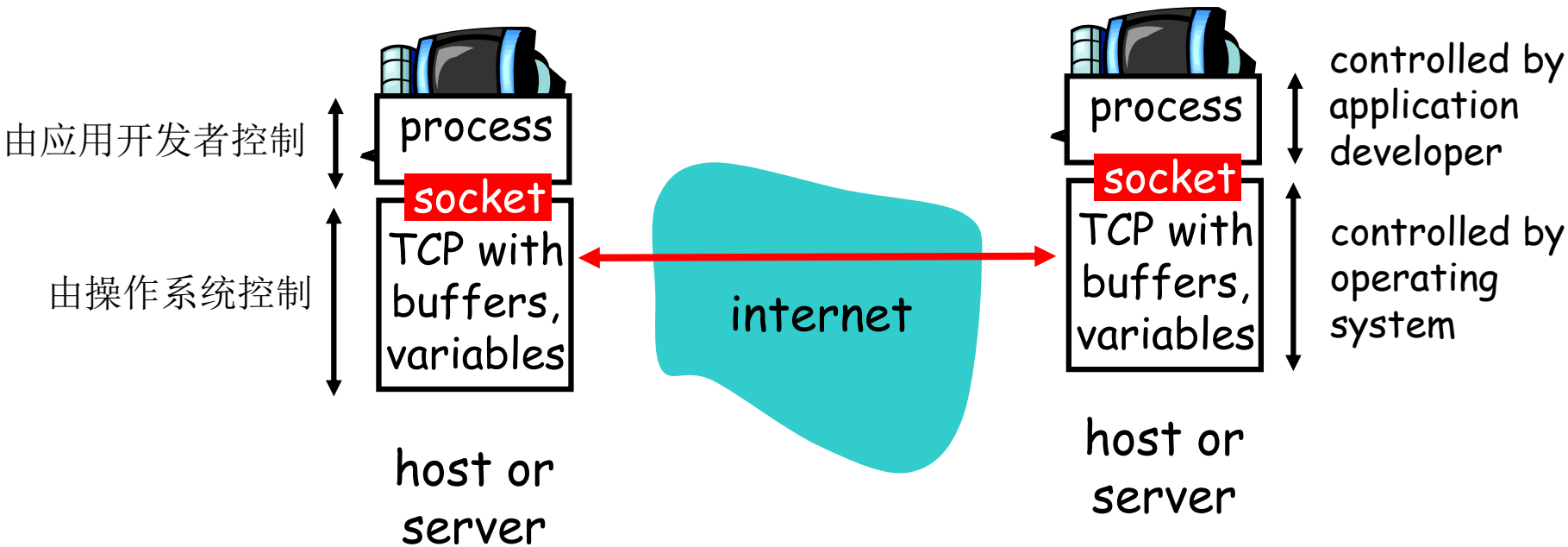
```
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```

print out received string and close socket

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

2.7.2 TCP套接字编程

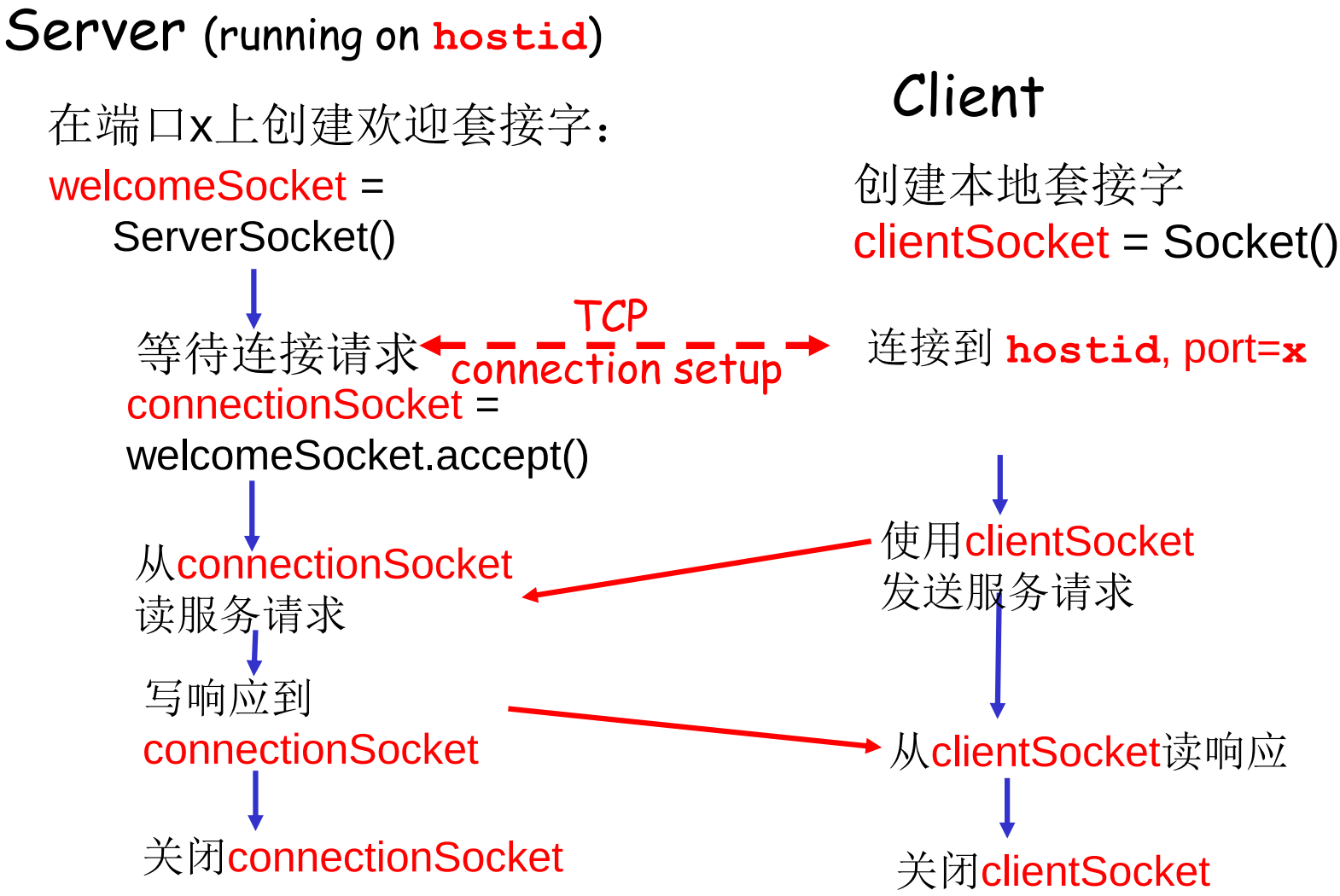
- 可将TCP连接想像成是一对套接字之间的一条封闭管道：
  - 发送端TCP将要发送的字节序列从管道的一端（套接字）送入
  - 接收端TCP从管道的另一端（套接字）取出字节序列
  - 在管道中传输的字节不丢失，并保持顺序



服务器使用多个套接字服务客户

- 服务器进程在欢迎套接字上等待客户的连接请求
- 客户进程需要通信时，创建一个客户套接字，与服务器欢迎套接字通信：
  - 在此过程中，客户TCP向服务器TCP发送连接请求
- 服务器进程创建一个临时套接字（称连接套接字）和一个新的服务器进程，与客户进程通信
- 服务器进程回到欢迎套接字上继续等待
  - 允许服务器同时服务多个客户
- 客户服务结束后，服务器销毁进程，关闭连接套接字

客户-服务器交互：TCP


Server (running on hostid)
在端口x上创建欢迎套接字：
welcomeSocket =
ServerSocket()
等待连接请求
connectionSocket =
welcomeSocket.accept()
从connectionSocket
读服务请求
写响应到
connectionSocket
关闭connectionSocket
TCP
connection setup
Client
创建本地套接字
clientSocket = Socket()
连接到 hostid, port=x
使用clientSocket
发送服务请求
从clientSocket读响应
关闭clientSocket

Example app: TCP server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET,SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

UDP服务与TCP服务

UDP

- 报文传输服务
- 由于没有建立管道，应用程序发送每个报文必须给出远程进程地址
- 服务器使用一个进程和一个套接字为所有客户服务，一次请求-响应完成一次服务

TCP

- 字节流传输服务
- 由于建立了管道，应用程序只需向套接字中写入字节序列，不需指出远程进程地址
- 服务器为每个客户单独生成一个套接字和一个新进程，允许双方长时间通信

Chapter 2: Summary

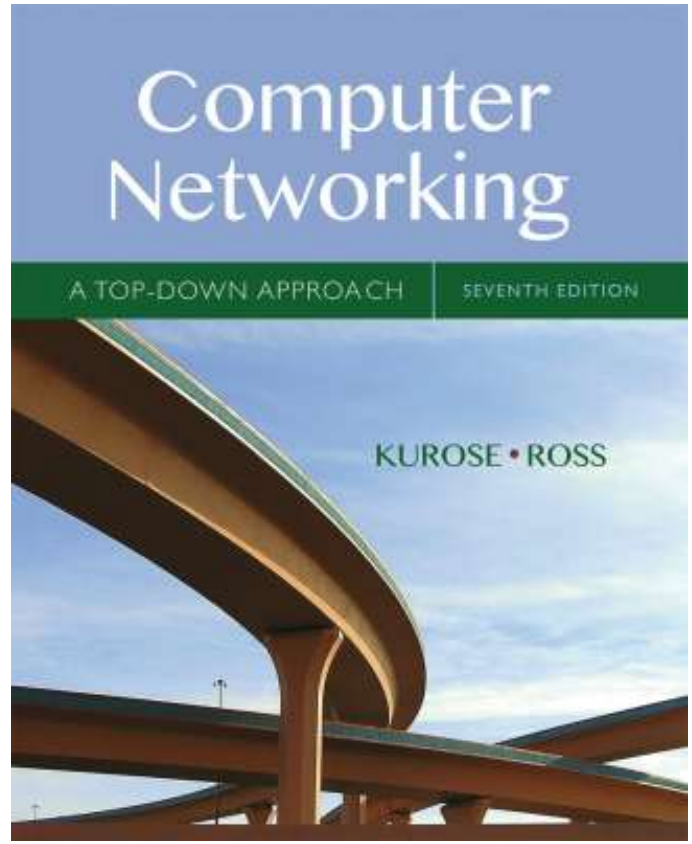
- 应用架构
  - client-server
  - P2P
- 应用服务需求:
  - reliability, bandwidth, delay, security
- 因特网传输服务模型
  - 面向连接，可靠: TCP
  - 不可靠，数据报: UDP
- 应用典型地采用请求/响应方式进行交互:
  - 客户请求信息或服务
  - 服务器用数据或状态码进行响应
- 报文格式:
  - 报头: 携带元数据
  - 实体: 携带数据本身
- 套接字编程:
  - TCP
  - UDP

作业

1. 习题：1，3，7，8，9
2. HTTP实验
3. DNS实验

- 作业提交时间：
  - HTTP实验：9月21日
  - 习题：9月26日
  - DNS实验：10月7日

Chapter 3
Transport Layer



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Chapter 3: 传输层

Our goals:

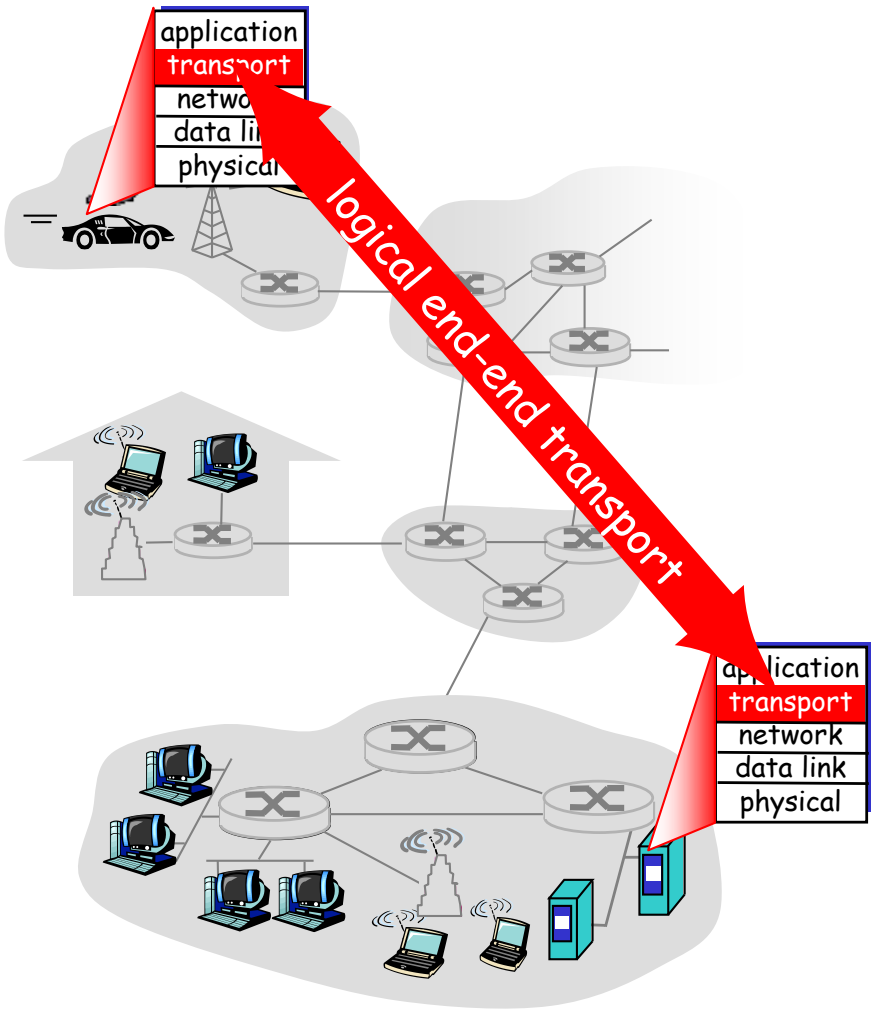
- 理解传输层服务原理：
  - 多路复用与多路分解
  - 可靠数据传输
  - 流量控制
  - 拥塞控制
- 学习因特网的传输层协议：
  - UDP: 无连接传输
  - TCP: 面向连接的传输
  - TCP拥塞控制

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

3.1 传输服务

- 在应用程序看来，
  - 源进程向本地套接字写入报文或数据，目的进程在本地套接字即可收到报文或数据
  - 源进程和目的进程仿佛直接连接在一起
  - 传输层提供了进程间的逻辑通信
- 在传输层看来，
  - 发送方传输层将报文交给本地网络层接口，接收方传输层从本地网络层接口即可收到报文
  - 网络层提供了终端间的逻辑通信



传输层和网络层的关系

- 网络层:
  - 提供主机之间的逻辑通信
- 传输层:
  - 提供进程之间的逻辑通信

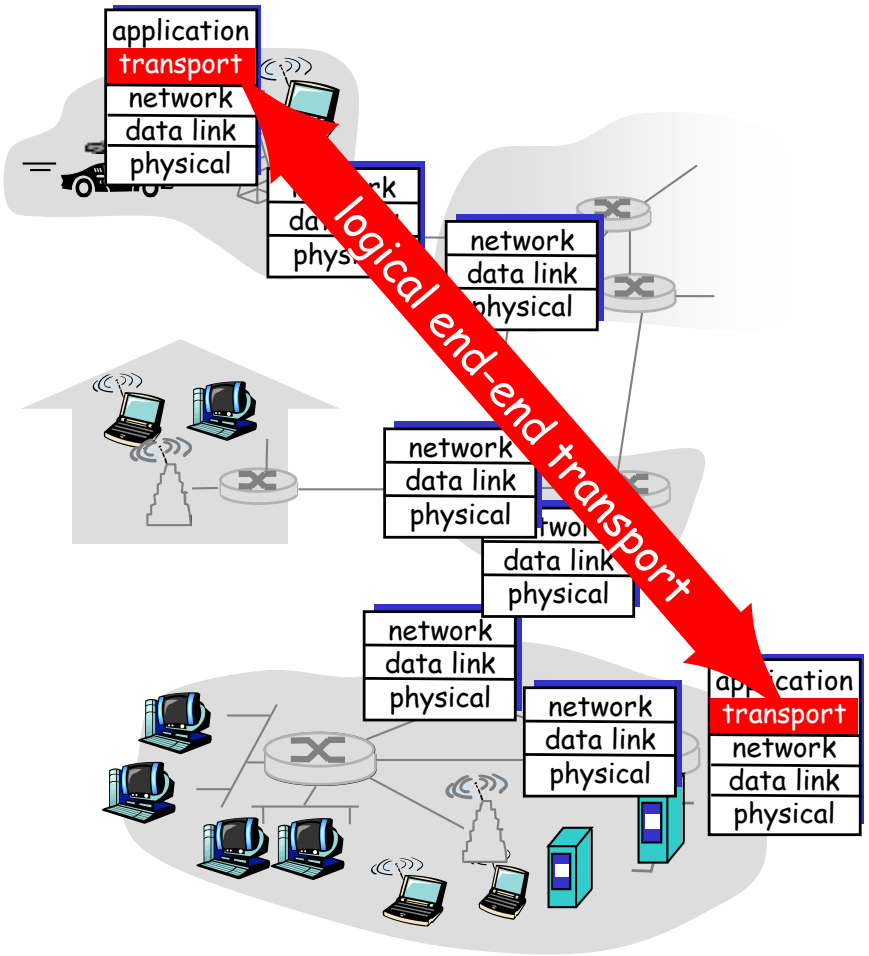
- **传输层依赖并增强网络层服务**

Household analogy:

- 一个家庭的12个孩子和另一个家庭的12个孩子通信，分别推选Ann和Bill负责收集和邮寄信件，以及查看信箱和分发信件
- 进程 = 孩子
- 应用报文 = 信
- 传输层 = Ann 和 Bill（提供人到人的服务）
- 主机 = 住宅
- 网络层 = 邮政系统（提供门到门的服务）

传输服务和网络服务

- 网络层提供尽力而为的服务：
  - 网络层尽最大努力在主机间交付分组，但不提供任何承诺
  - 具体来说，不保证交付，不保证按序交付，不保证数据完整，不保证延迟，不保证带宽，……
- 传输层不能提供的服务：
  - 延迟保证
  - 带宽保证
- 传输层可以提供的服务：
  - 保证可靠、按序的交付：TCP
  - 不保证可靠、按序的交付：UDP



Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

3.2 多路复用与解复用

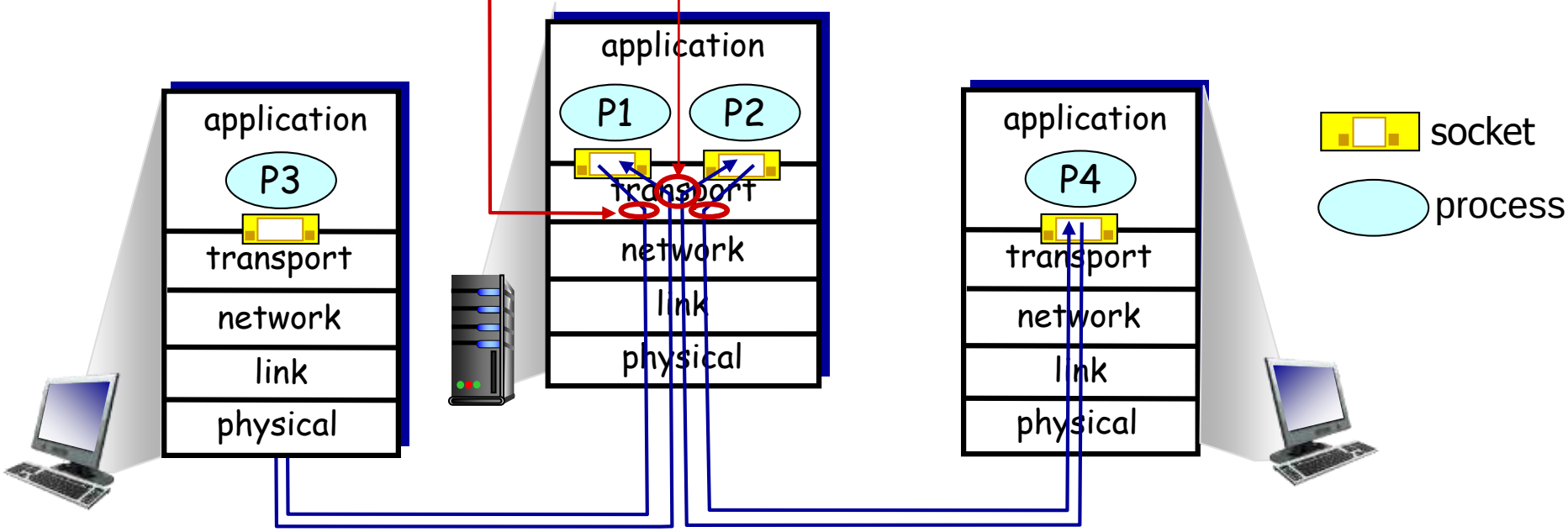
传输层基本服务：将主机间交付扩展到进程间交付

（发送端）多路复用：

传输层从多个套接字收集数据，交给网络层发送

（接收端）解复用：

传输层将收到的数据交付到正确的套接字



如何进行多路复用和解复用？

为将邮件交付给收信人：

- 每个收信人应有一个信箱，写有收信人地址和姓名（唯一标识）
- 信封上有收信人地址和名字

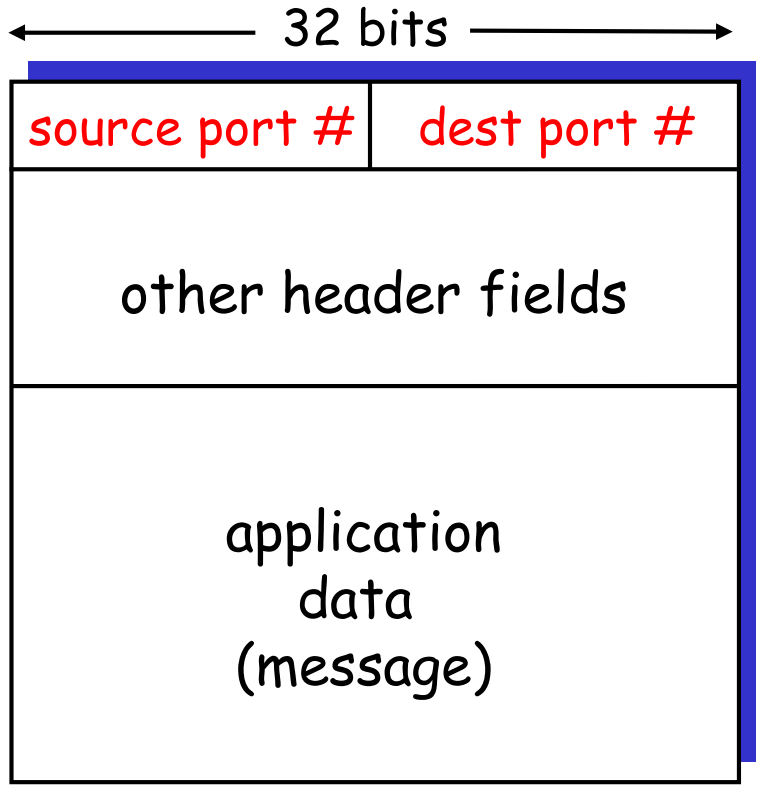
为将报文段交付给套接字：

- 主机中每个套接字应分配一个唯一的标识
- 报文段中包含接收套接字的标识

- 多路复用：
  - 发送方传输层将源/目的套接字标识置于报文段中，交给网络层
- 解复用：
  - 接收方传输层根据报文段中的目的套接字标识，将报文段交付到正确的套接字

套接字与端口号

- 端口号是套接字标识的一部分
  - 每个套接字在本地关联一个端口号
- 端口号：
  - 一个16比特的数
  - 0～1023由公共域协议使用，称众所周知的端口号
- 报文段中有两个字段携带端口号
  - 源端口号：与发送进程关联的本地端口号
  - 目的端口号：与接收进程关联的本地端口号



TCP/UDP报文段格式

如何分配UDP套接字的端口号？

- 自动分配：
  - 例如，new Datagramsocket()，不指定端口号
  - 通常由操作系统从1024~65535中分配
  - 客户端通常使用这种方法
- 使用指定端口号创建套接字：
  - 例如，new Datagramsocket(53)
  - 实现公共域协议的服务器应分配众所周知的端口号
  - 服务器通常采用这种方法

- **UDP套接字标识为<IP地址，端口号>二元组**

UDP解复用

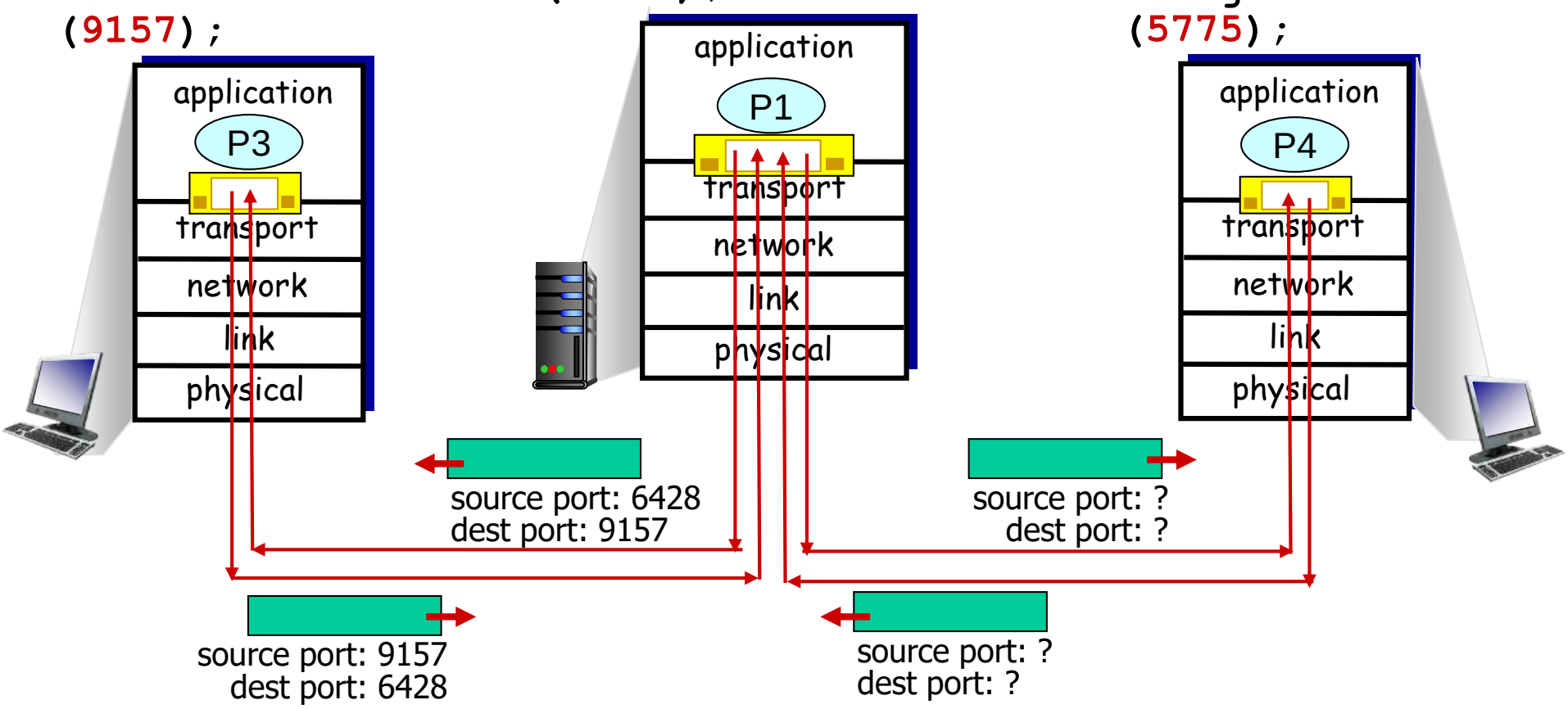
- 接收方传输层收到一个UDP报文后:
  - 检查报文中的目的端口号，将UDP报文交付到具有该端口号的套接字
  - <目的IP地址，目的端口号> 相同的UDP报文被交付给同一个套接字，与 <源IP地址，源端口号> 无关
  - 报文中的 <源IP地址，源端口号> 被接收进程用来发送响应报文

UDP解复用：举例

```
DatagramSocket
 mySocket2 = new
 DatagramSocket
 (9157);
```

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

```
DatagramSocket
 mySocket1 = new
 DatagramSocket
 (5775);
```



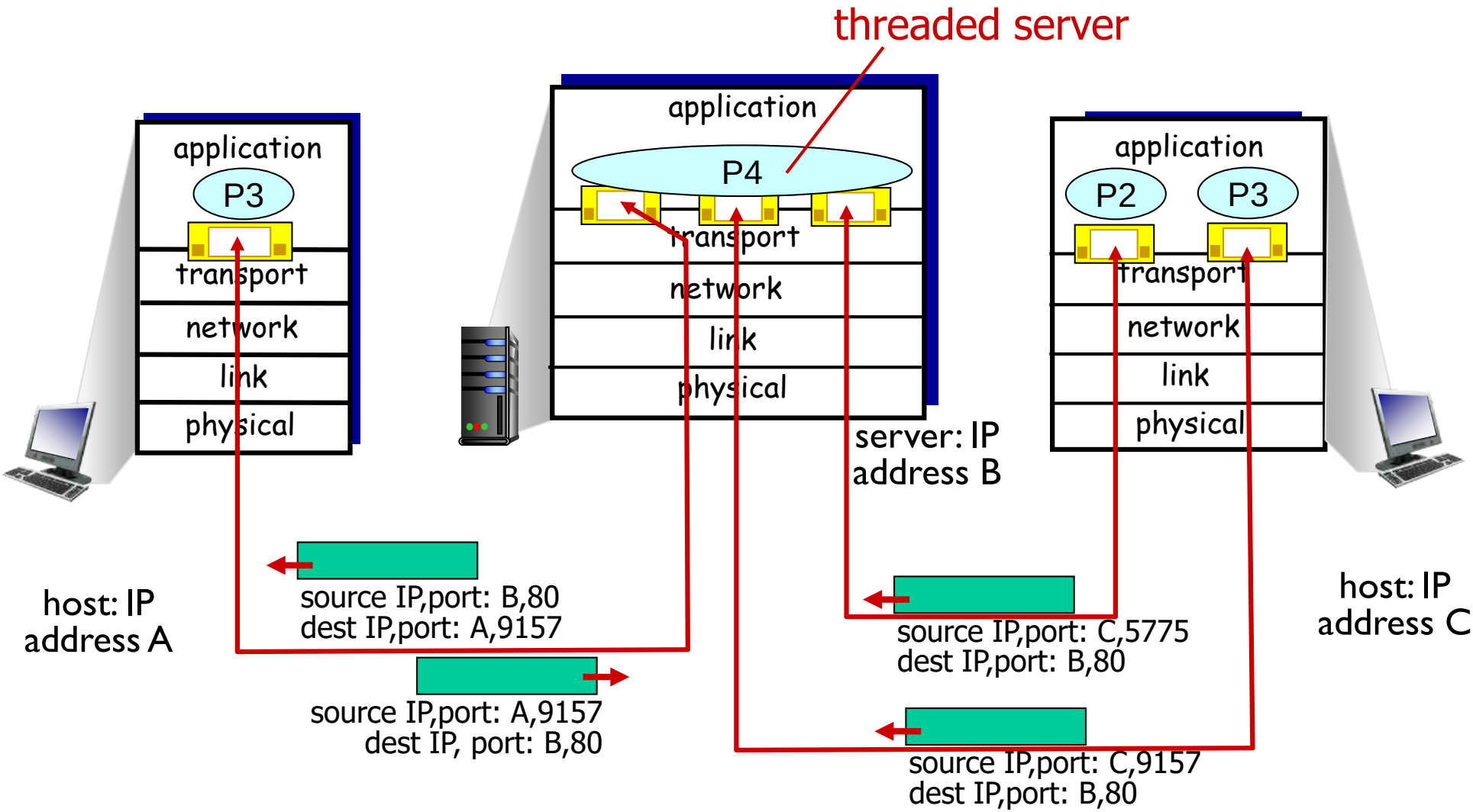
创建TCP套接字

- 服务器在port=x创建一个欢迎套接字：
  - welcomeSocket = new ServerSocket(x)
- 客户A创建一个与欢迎套接字通信的客户套接字（假设自动分配端口号y）：
  - clientSocket = new Socket("hostname", x)
- 服务器在收到客户A的连接请求后创建一个连接套接字：
  - connectionSocket = welcomeSocket.accept()
  - 该连接套接字只与客户A的套接字通信，即只接收具有以下四元组的报文段：
    - 源IP地址 = 客户A的IP地址
    - 源端口号 = y
    - 目的IP地址 = 服务器的IP地址
    - 目的端口号= x
- 不同的客户进程与服务器上不同的连接套接字对应

TCP解复用

- 服务器主机可能有多个连接套接字
- 每个连接套接字与一个进程相联系，并由 <源IP地址，目的IP地址，源端口号，目的端口号> 四元组进行标识
- 服务器使用该四元组将报文段交付到正确的连接套接字

TCP解复用：举例


threaded server
application
P3
transport
network
link
physical
application
P4
transport
network
link
physical
server: IP address B
application
P2
P3
transport
network
link
physical
host: IP address A
source IP,port: B,80
dest IP,port: A,9157
source IP,port: A,9157
dest IP, port: B,80
source IP,port: C,5775
dest IP,port: B,80
source IP,port: C,9157
dest IP,port: B,80
host: IP address C

小结

UDP套接字

- 使用<IP地址，端口号>二元组标识套接字
- 服务器使用一个套接字服务所有客户

TCP套接字

- 使用<源IP地址，目的IP地址，源端口号，目的端口号>四元组标识连接套接字
- 服务器使用多个连接套接字，每个连接套接字服务一个客户

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

学习网络协议的要点

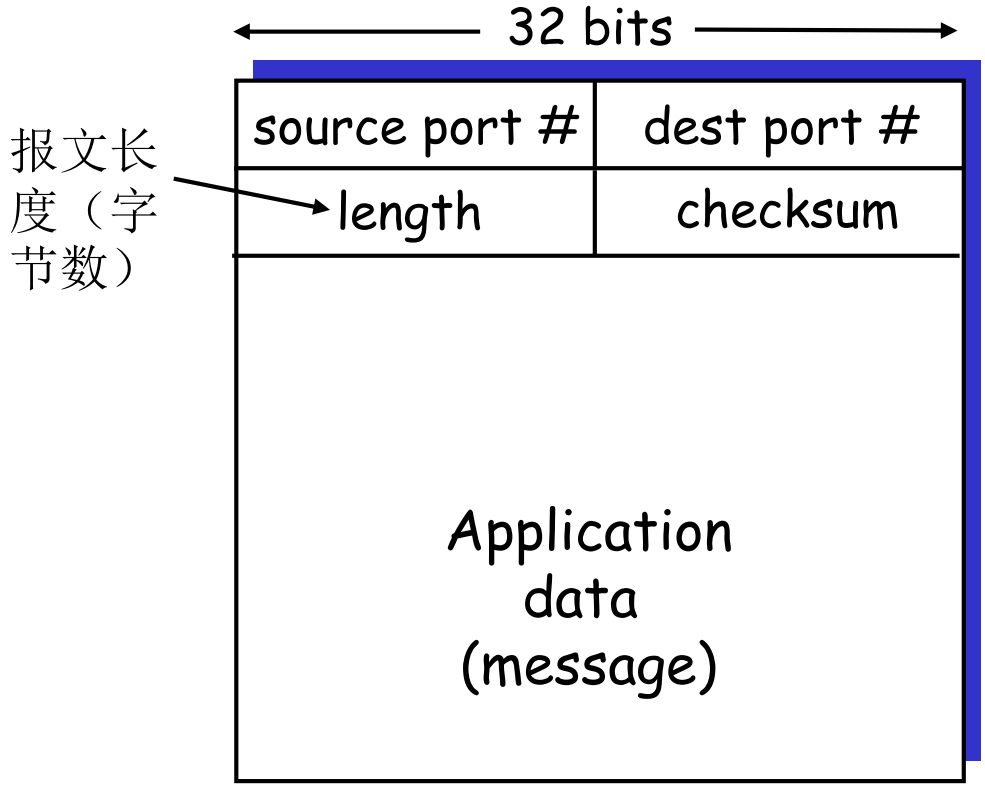
- 当学习一个网络协议时，可从以下几个方面去理解和梳理知识点：
  - 这个协议提供什么服务
  - 相应地，需要实现哪些功能
  - 这些功能的实现如何体现在报头的设计中
  - 这些功能的实现如何体现在协议机制的设计中

UDP: User Datagram Protocol [RFC 768]

- 网络层提供的服务（best-effort service）：
  - 尽最大努力将数据包交付到目的主机
  - 不保证投递的可靠性和顺序
  - 不保证带宽及延迟要求
- UDP提供的服务：
  - 进程到进程之间的报文交付
  - 报文完整性检查（可选）：检测并丢弃出错的报文
- UDP需要实现的功能：
  - 多路复用与解复用
  - 报文检错

3.3.1 UDP报文结构

- UDP报文：
  - 报头：携带协议处理需要的信息
  - 载荷（payload）：携带上层数据
- 用于多路复用/解复用的字段：
  - 源端口号，目的端口号
- 用于检测报文错误的字段：
  - 报文总长度
  - 检查和（checksum）



UDP报文格式

3.3.2 UDP检查和（checksum）

作用：对传输的报文进行检错

发送方：

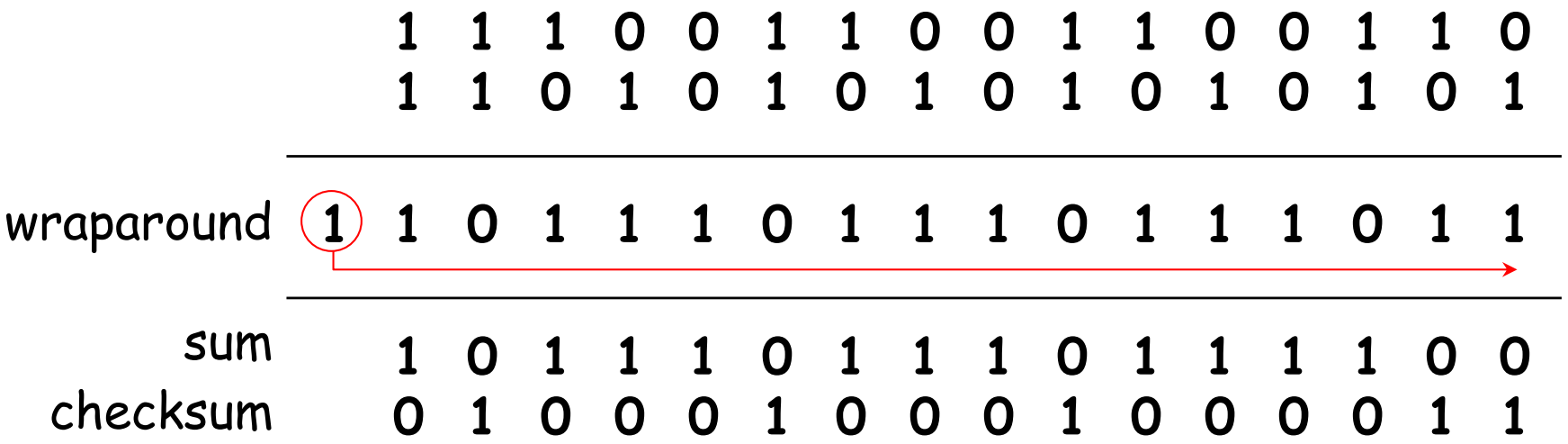
- 将报文看成是由16比特整数组成的序列
- 对这些整数序列计算检查和
- 将检查和放到UDP报文的checksum字段

接收方：

- 对收到的报文进行相同的计算
- 与报文中的checksum字段进行比较：
  - 不相等：报文有错误
  - 相等：**认为没有错误**

Internet checksum: example

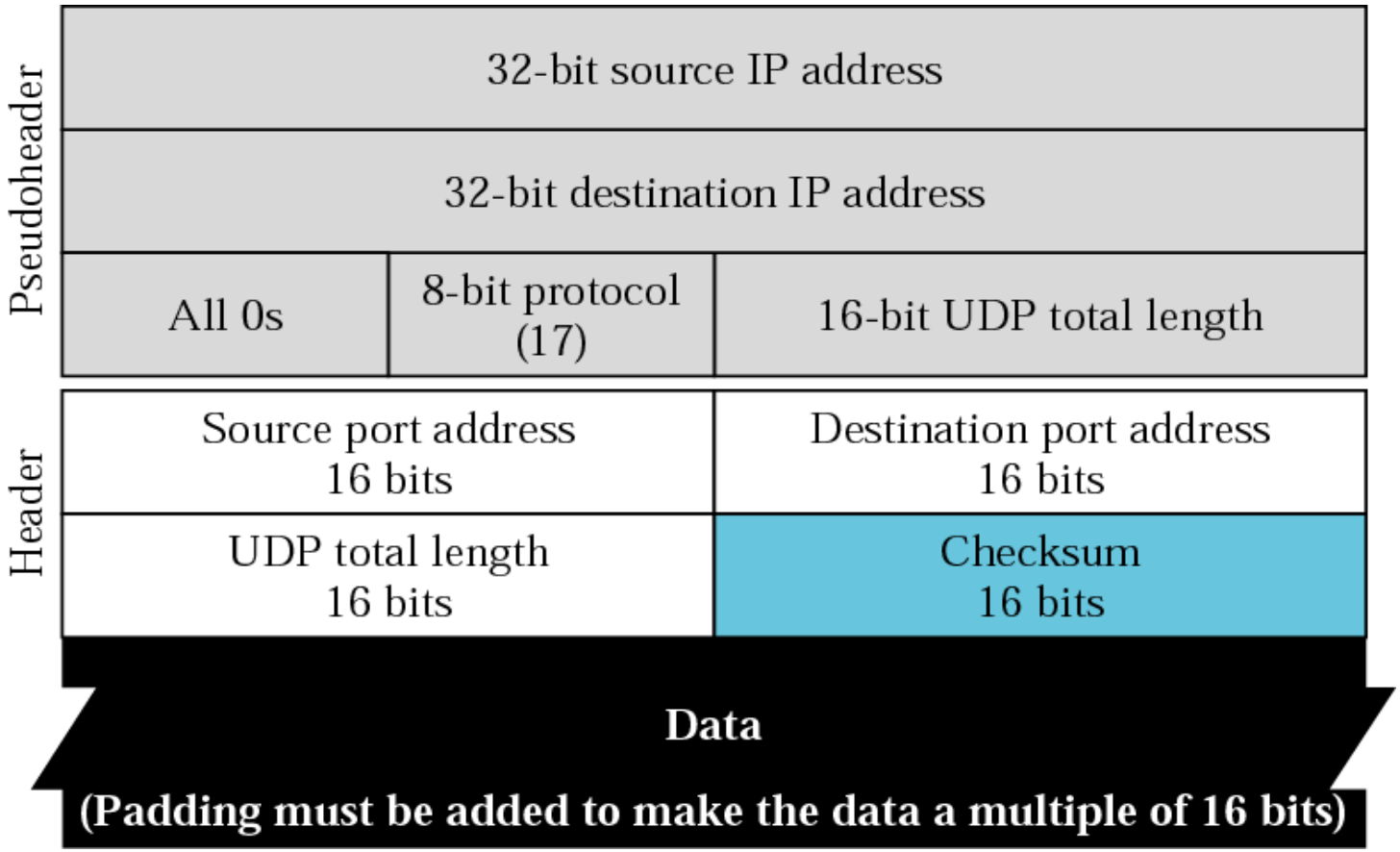
example: add two 16-bit integers



Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

UDP检查和计算

- 计算UDP检查和包括伪头、UDP头和数据三个部分。
- 检查和的使用是可选的，若不计算检查和，该字段填入0。



检查和计算举例

153.18.8.105			
171.2.14.10			
All 0s	17	15	
1087		13	
15		All 0s	
T	E	S	T
I	N	G	**All 0s**

10011001	00010010	153.18
00001000	01101001	8.105
10101011	00000010	171.2
00001110	00001010	14.10
00000000	00010001	0 and 17
00000000	00001111	15
00000100	00111111	1087
00000000	00001101	13
00000000	00001111	15
00000000	00000000	0 (checksum)
01010100	01000101	T and E
01010011	01010100	S and T
01001001	01001110	I and N
01000111	00000000	G and 0 (padding)
10010110	**11101011**	Sum
01101001	**00010100**	Checksum

- 计算检查和时，checksum字段填0
- 接收方对UDP报文（包括检查和）及伪头求和，若结果为0xFFFF，认为没有错误

为什么需要UDP？

why is there a UDP?

- 应用可以尽可能快地发送报文：
  - 无建立连接的延迟
  - 不限制发送速率（不进行拥塞控制和流量控制）
- 报头开销小
- 协议处理简单

- UDP适合的应用：
  - 容忍丢包但对延迟敏感的应用：如流媒体
  - 以单次请求/响应为主的应用：如DNS
- 若应用要求基于UDP进行可靠传输：
  - 由应用层实现可靠性

Chapter 3 outline

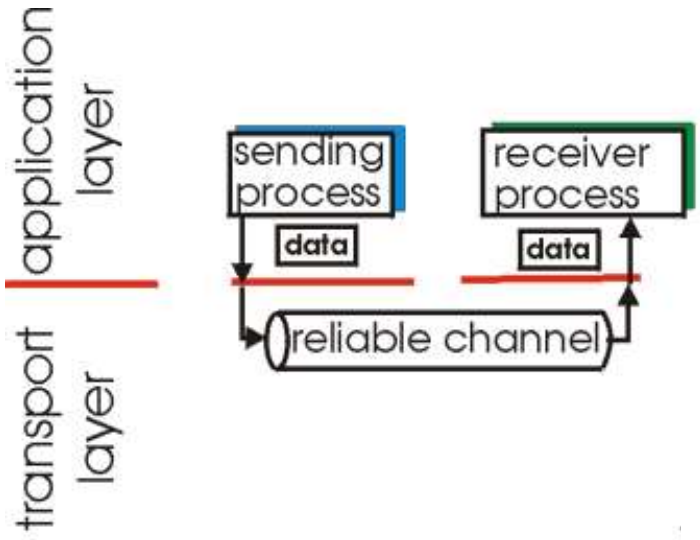
- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

可靠传输是因特网的重要设计问题

- 可靠数据传输是因特网中最为重要的问题：
  - 对于有些应用（如文件传输），1比特的错误都可能导致传输的数据无效
  - 在因特网这样的大规模网络中，数据传输极易出错
  - 可靠传输在链路层、传输层、甚至应用层都需要
- 可靠数据传输也是本课程的重点和难点之一
- 可靠性传输的一般性原理适用于所有层次

可靠数据传输原理

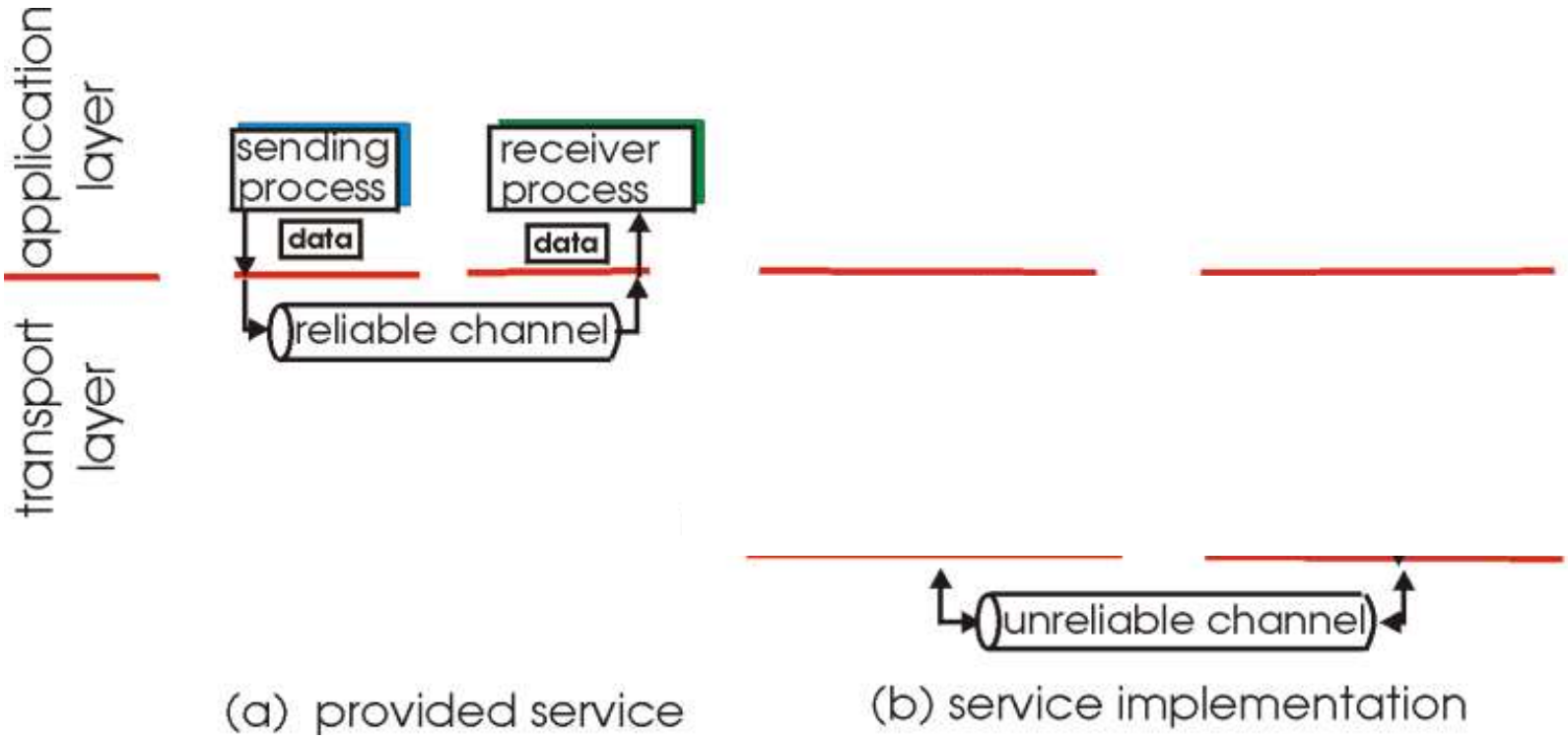
- 可靠传输的服务抽象：数据通过一条理想信道传输，数据不会有比特损坏或丢失，并按照发送的顺序被接收



(a) provided service

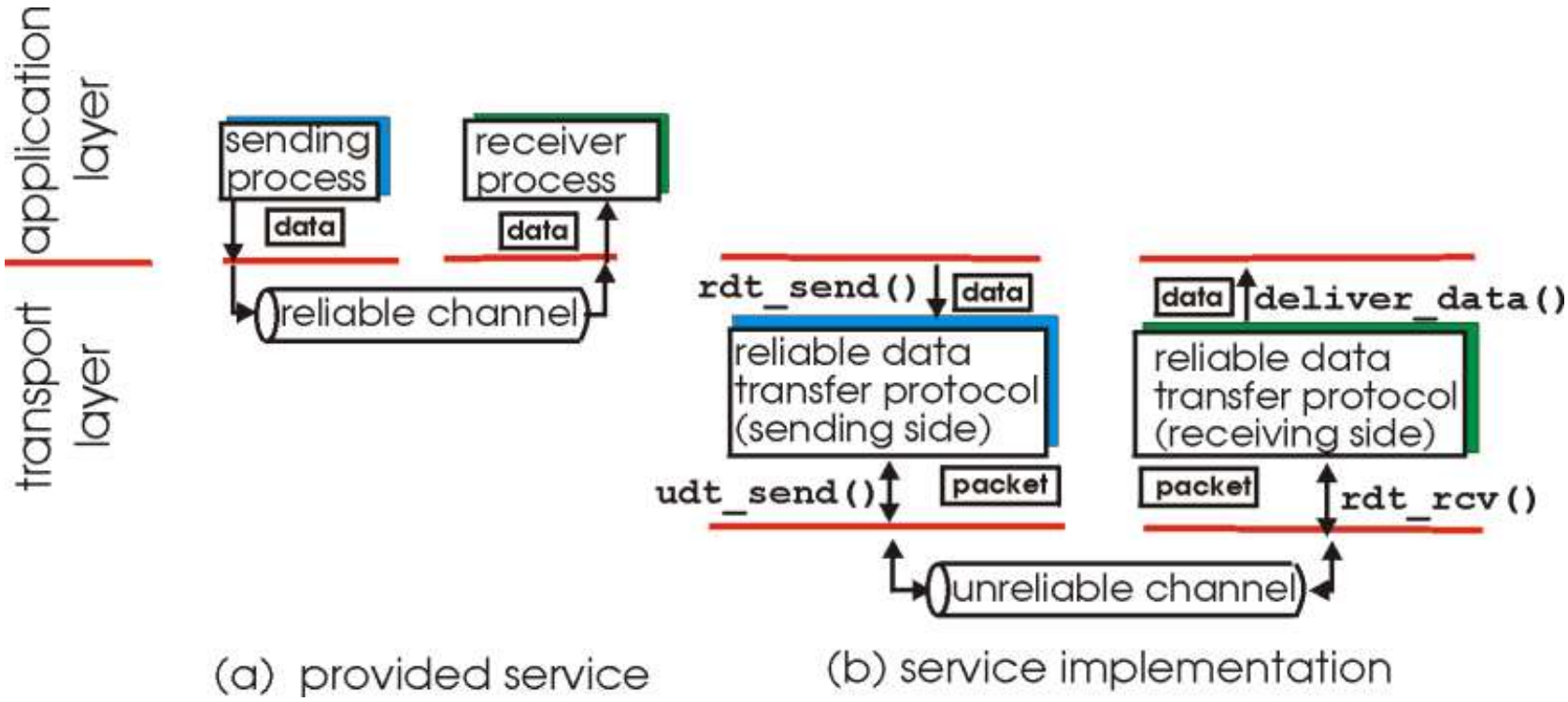
可靠数据传输原理

- 可靠传输的服务抽象：数据通过一条理想信道传输，数据不会有比特损坏或丢失，并按照发送的顺序被接收



可靠数据传输原理

- 可靠传输的服务抽象：数据通过一条理想信道传输，数据不会有比特损坏或丢失，并按照发送的顺序被接收

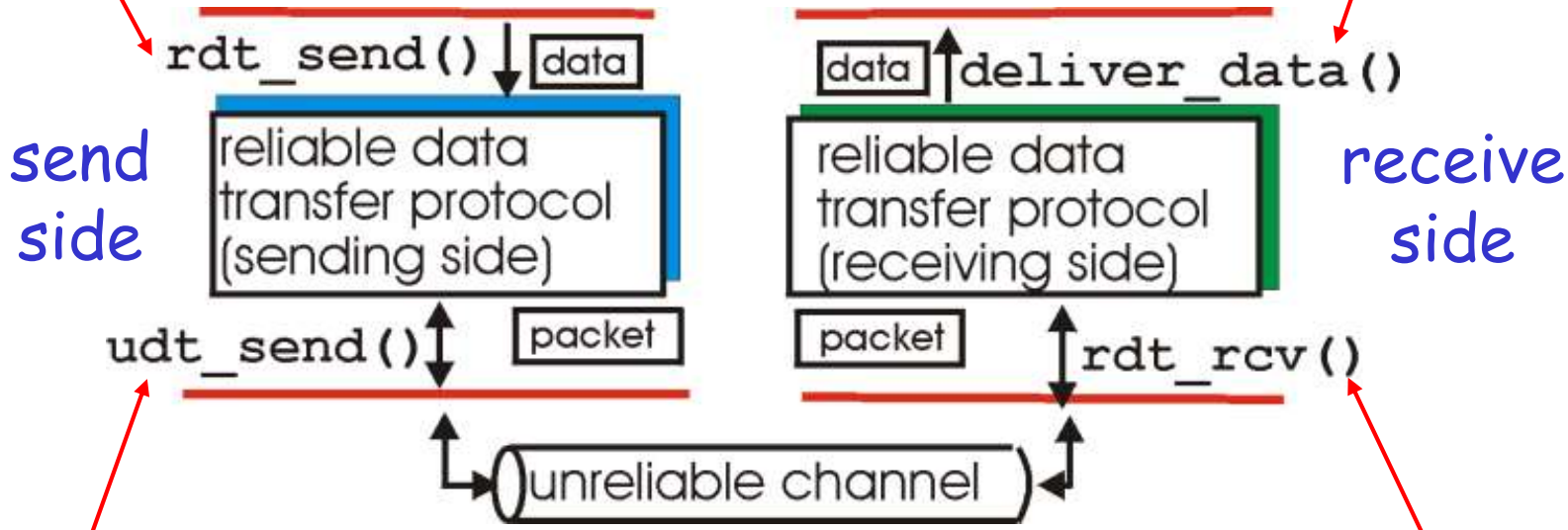


- 不可靠信道的特性决定了可靠数据传输协议的复杂性

3.4.1 构造可靠数据传输协议

rdt_send()：由上层实体调用，将要发送的数据传给rdt

deliver_data()：由rdt调用，将数据交付给上层实体



udt_send()：由rdt调用，将要发送的分组交给下层协议实体

rdt_rcv()：当分组到达接收端时，由下层协议实体调用

getting started

我们将：

- 使用有限状态机（FSM）来描述发送方和接收方
- 从理想的信道模型开始，增量地开发可靠数据传输协议（rdt）的发送方和接收方
- 只考虑单向数据传输，但控制信息可以双向传输

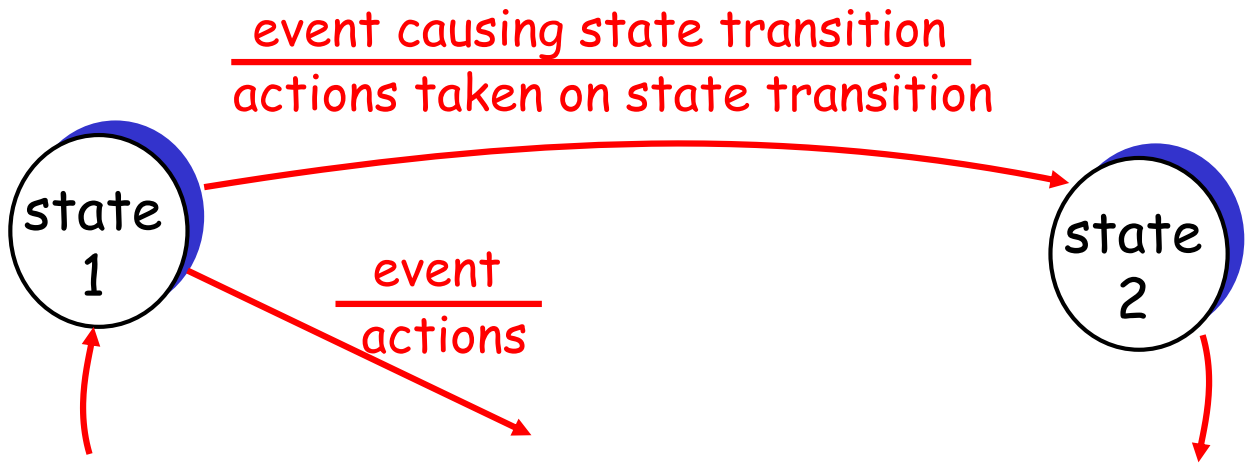
将考虑以下几种信道：

- 理想信道：完全不会出错，有序传输
- 可能有比特错误，但不会丢包
- 可能有比特错误，也可能丢包

重点理解：

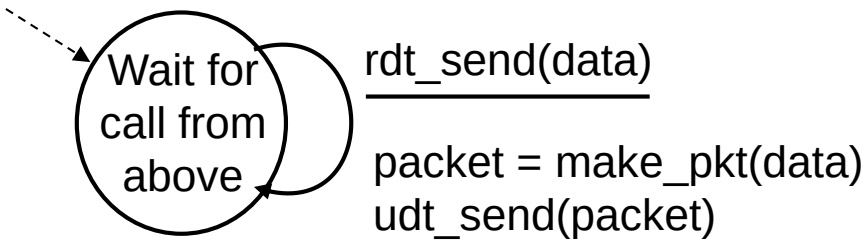
- 每一种措施因为什么而提出？
- 引入这个措施会产生什么问题？

state: when in this "state" next state uniquely determined by next event

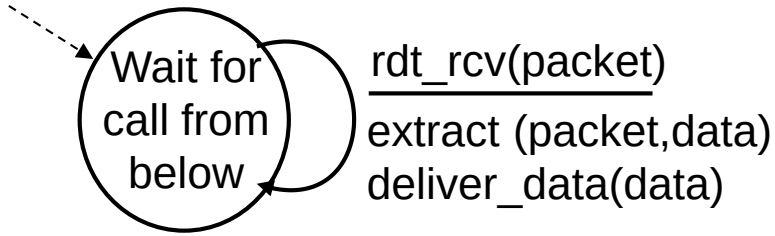


Rdt1.0: 理想信道上的可靠传输

- 下层信道是完全可靠的（理想情况）
  - 没有比特错误，没有分组丢失
- 假设发送能力 ≤ 接收能力（不会出现来不及接收的现象）
- 发送方和接收方的FSM均只有一个状态:
  - 发送方：从上层接收数据，封装成分组送入下层信道
  - 接收方：从下层信道接收分组，取出数据交给上层



sender

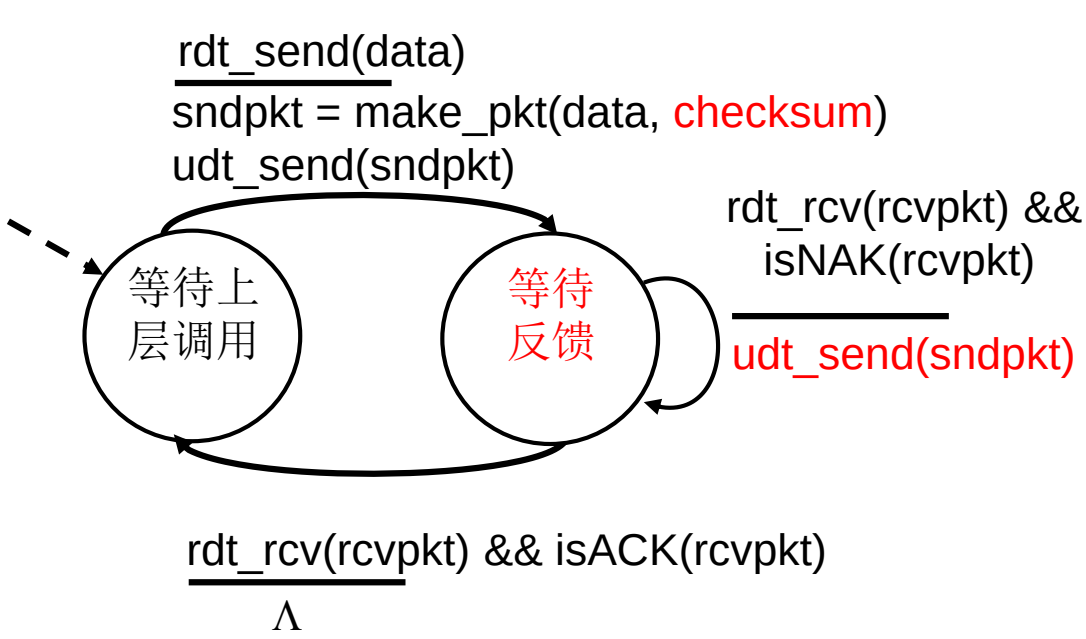


receiver

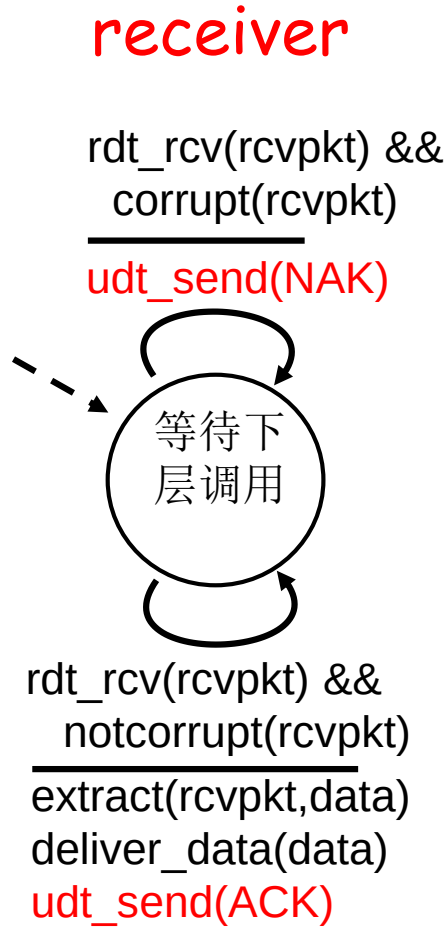
Rdt2.0: 可能产生比特错误的下层信道

- 下层信道可能使分组中的比特产生错误（比特翻转）
  - 可以通过某种检错码（如checksum）检测比特错误
- **问题：如何从错误中恢复？**
  - 肯定确认(ACK): 接收方显式地告诉发送方，收到的分组正确
  - 否定确认(NAK): 接收方显式地告诉发送方，收到的分组有错
  - 重传：发送方收到NAK后，重传出错的分组
- **与rdt1.0相比，rdt2.0中需要三种新的机制:**
  - 接收方检错
  - 接收方反馈
  - 发送方重传分组

rdt2.0: FSM specification



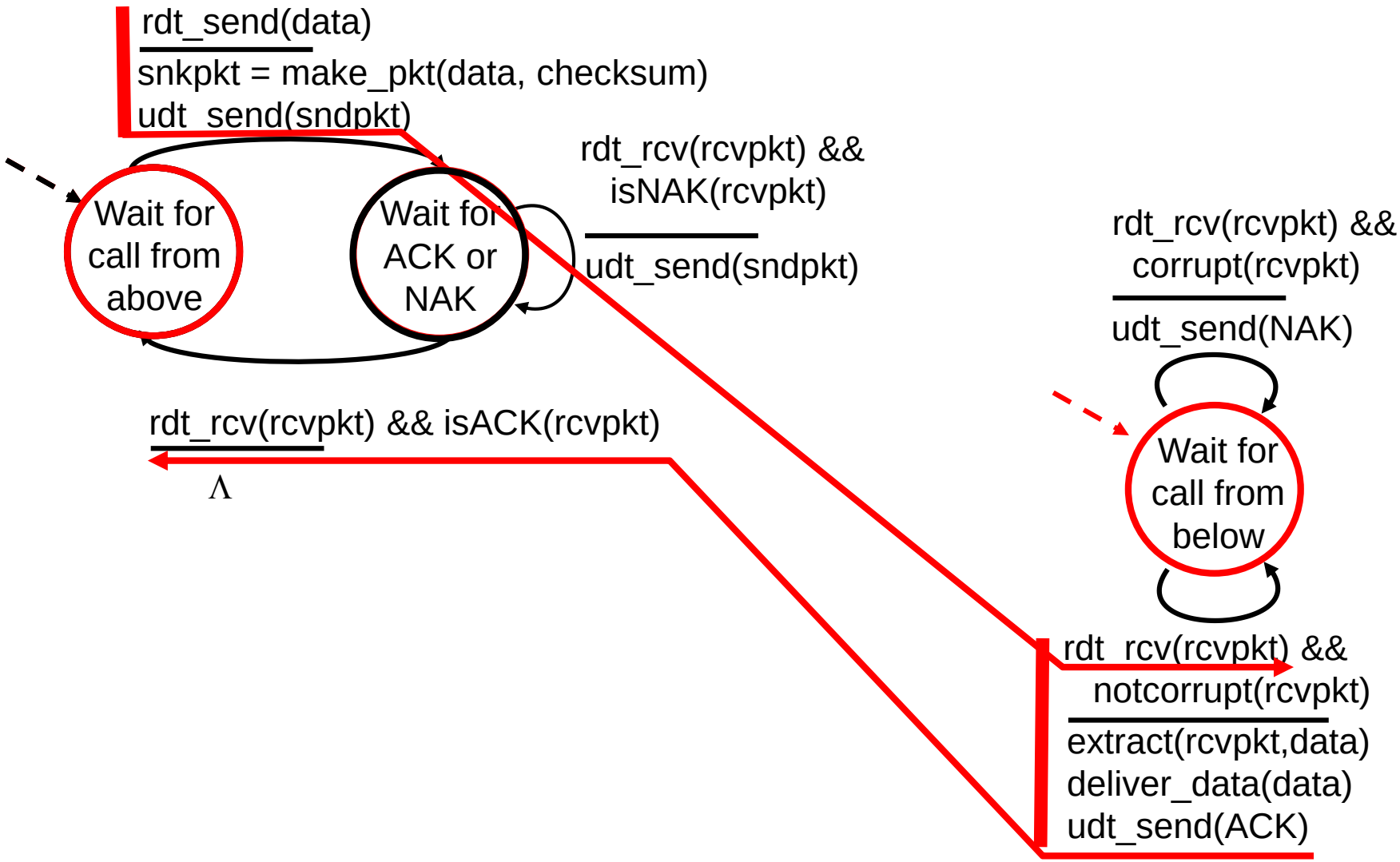
sender



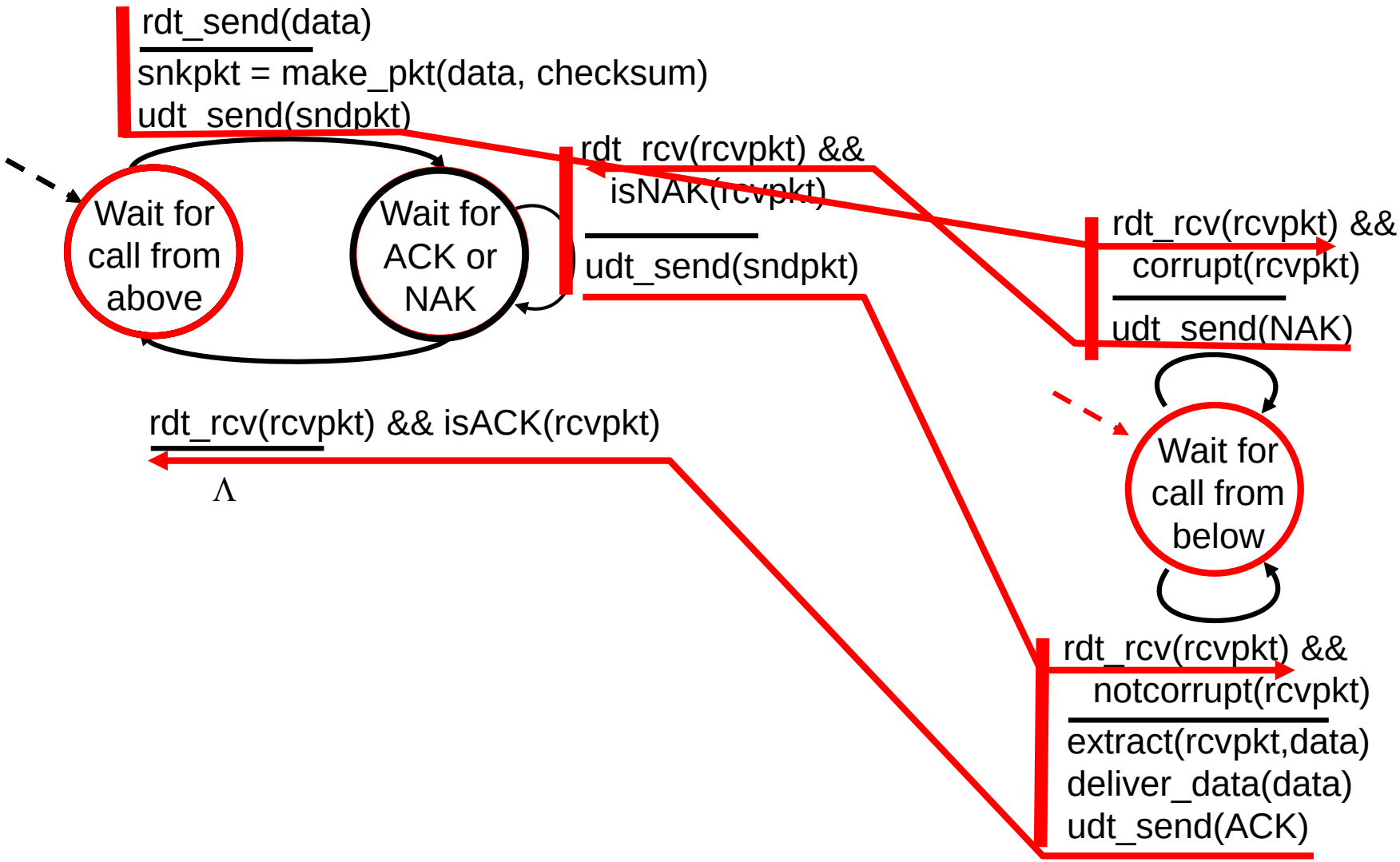
- 发送方FSM中增加“等待反馈”状态
- 构造分组时增加checksum
- 收到NAK后重发分组

- 接收方根据检错结果发送ACK或NAK

rdt2.0: operation with no errors



rdt2.0: error scenario



rdt2.0 has a fatal flaw!

ACK或NAK出错会怎样？

- 发送方不清楚接收方发生了什么！
- 常见的处理方法：发送方在收到出错的确认后，重传该分组（假设分组也出错了）
- 问题：可能在接收端产生冗余分组

如何处理冗余问题？

- 发送方给每个分组添加一个序号
- 接收方根据序号检测冗余的分组，并丢弃（不会交付给上层）

分组序号的长度？

- 如果前一个分组发送成功了再发送下一个分组，只需1个比特

发送方FSM：rdt2.0 vs Rdt2.1

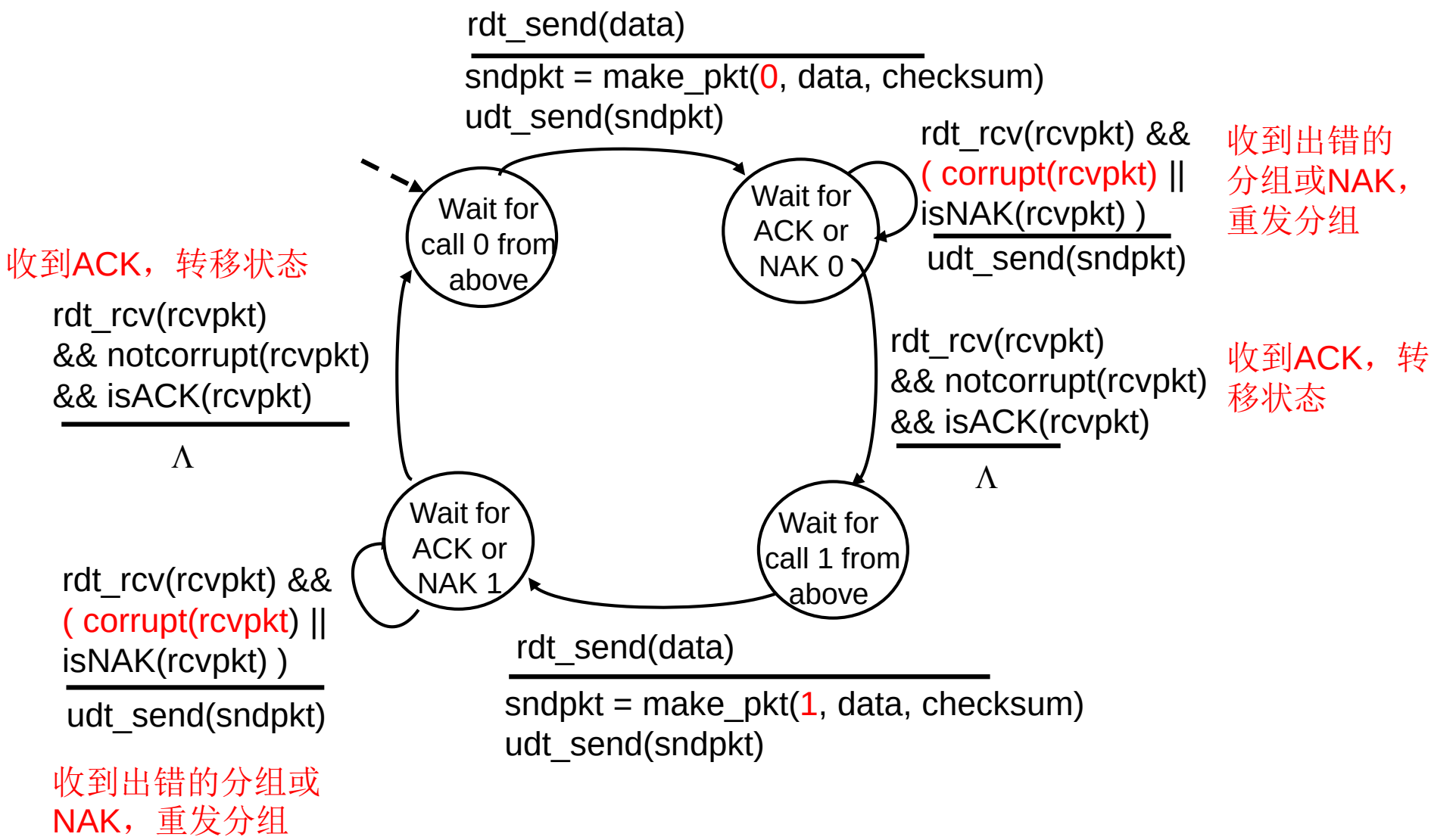
Rdt2.0

- 构造分组：未加序号
- 等待反馈：
  - 收到ACK，转移到下一个状态
  - 收到NAK，重发分组

Rdt 2.1

- 构造分组：加入序号
- 等待反馈：
  - 收到ACK，转移状态
  - 收到NAK，重发分组
  - 收到出错的反馈，重发分组

rdt2.1: 发送方，处理出错的ACK/NAK



接收方FSM：rdt2.0 vs rdt2.1

Rdt2.0:

- 收到出错的分组：发送NAK
- 收到正确的分组：交付数据，发送ACK

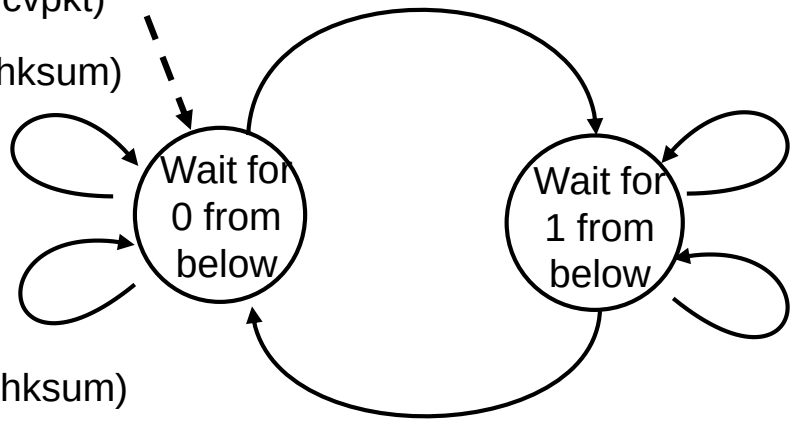
Rdt2.1:

- 收到出错的分组：发送NAK
- 收到冗余的分组：发送ACK，不交付数据
- 收到正确的新分组：交付数据，发送ACK

增加一个状态：

- 区分新的分组和冗余的分组

rdt2.1: 接收方，处理出错的ACK/NAK



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)

extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

正确收到序号为0的分组，
发送ACK，转移状态

收到出错的分组，发送NAK

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)

收到出错的分组，发送NAK

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq1(rcvpkt)

sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

正确收到序号为1的分组
（冗余），发送ACK

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)

extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

正确收到序号为1的分组，发送ACK
转移状态

rdt_rcv(rcvpkt) &&
not corrupt(rcvpkt) &&
has_seq0(rcvpkt)

sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

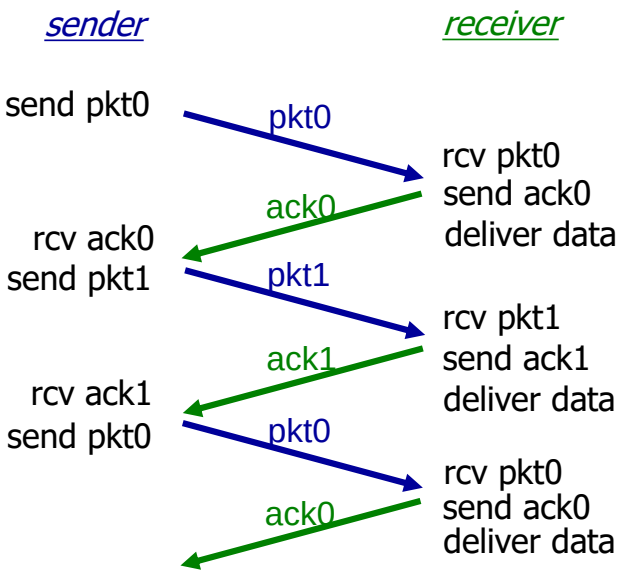
正确收到序号为0的分组
（冗余），发送ACK

rdt2.2: 不使用NAK的协议

rdt2.1也可以不用NAK，只用ACK

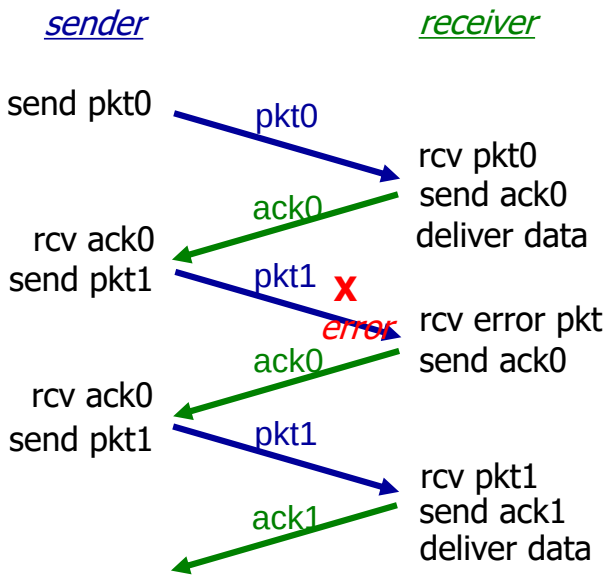
- 让ACK携带所确认分组的序号
- 接收方只对正确接收的分组发送ACK
- 对于出错的分组，重发最近一次的ACK

Rdt2.2 in action



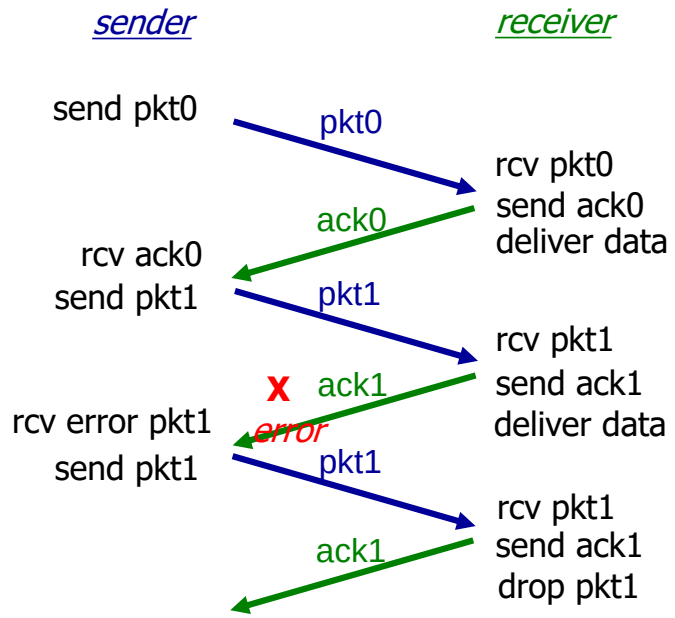
(a) no error

- 发送方收到期待序号的ACK后，发送下一个分组
- 接收方收到期待序号(#n)的分组后，发送ACK n，交付数据



(b) packet error

- 发送方收到非期待序号的ACK后，重发当前分组
- 接收方收到出错的分组后，重发ACK



(c) ACK error

- 发送方收到出错的ACK后，重发当前分组
- 接收方收到非期待序号的分组，发送ACK，丢弃分组

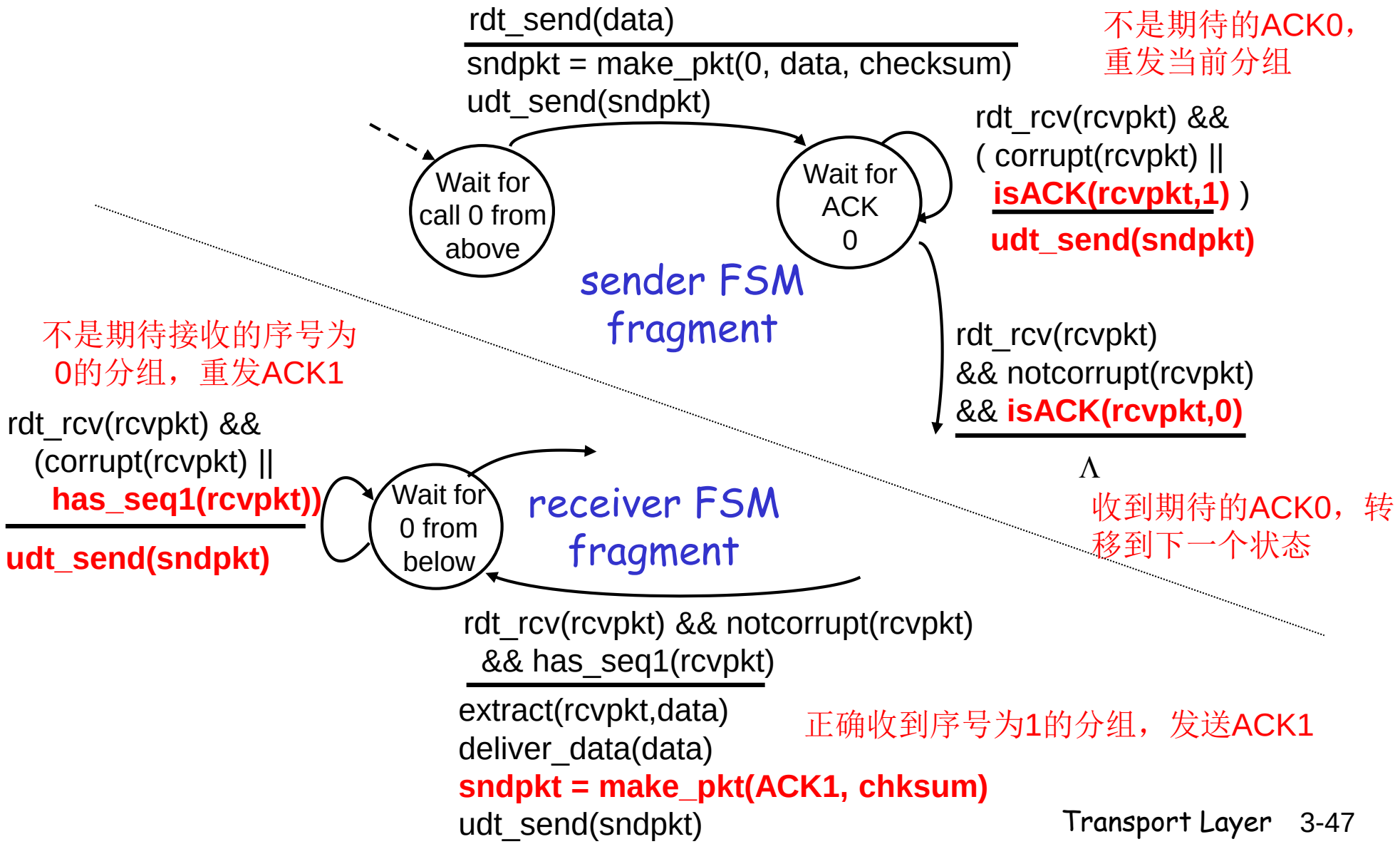
rdt2.2: 不使用NAK的协议

rdt2.1也可以不用NAK，只用ACK

- 让ACK携带所确认分组的序号
- 接收方只对正确接收的分组发送ACK
- 对于出错的分组，重发最近一次的ACK

- 发送方：
  - 收到期待序号的ACK：发送下一个分组
  - 其它情况（收到出错的ACK、非期待序号的ACK）：重发当前分组
- 接收方：
  - 收到期待序号(#n)的分组：发送ACK n，交付数据
  - 其它情况（收到出错的分组、非期待序号的分组）：重发最近一次的ACK

rdt2.2: sender, receiver fragments



Rdt 2.x小结

- Rdt 2.0
  - 数据可能出错
  - 引入检错、反馈（ACK 和 NAK）和重传机制
- Rdt 2.1
  - 反馈也可能出错
  - 若反馈出错，重传分组
  - 引入分组序号，用于区分重复分组
- Rdt 2.2
  - 取消NAK
  - ACK中引入分组序号
- 与理想信道上的Rdt 1.0相比，在有比特错误的信道上运行的rdt 2.2增加了以下机制：
  - 分组检错
  - 使用1比特分组序号
  - 使用带序号的ACK进行反馈
  - 在收到出错分组或非期待序号的分组时，重传分组（发送方和接收方一样）

rdt3.0: 可能产生比特错误和丢包的下层信道

增加新的假设:

下层信道可能丢包（包括数据及ACK）

需要两个新机制：

- 检测丢包
- 从丢包中恢复

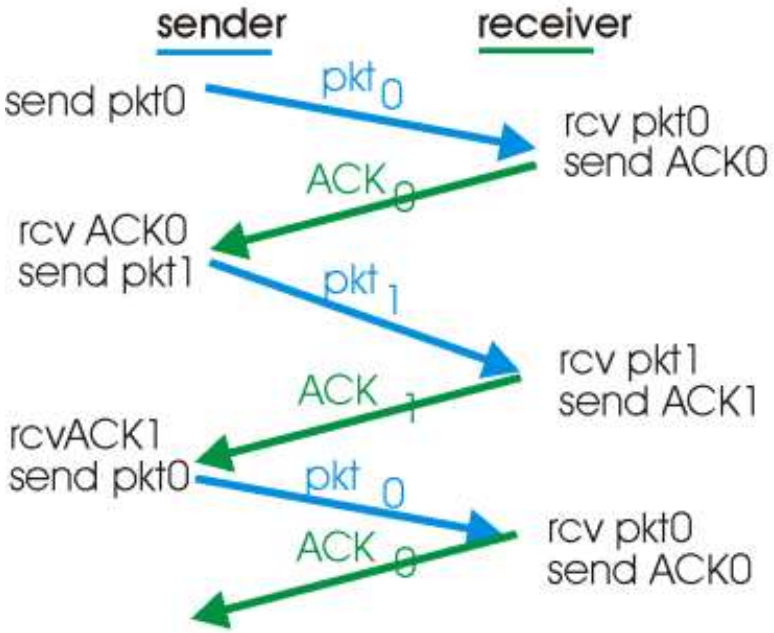
方法:

- 检测丢包：
  - 若发送方在“合理的”时间内未收到ACK，认为丢包（需要定时器）
- 从丢包中恢复：
  - 发送方重发当前分组

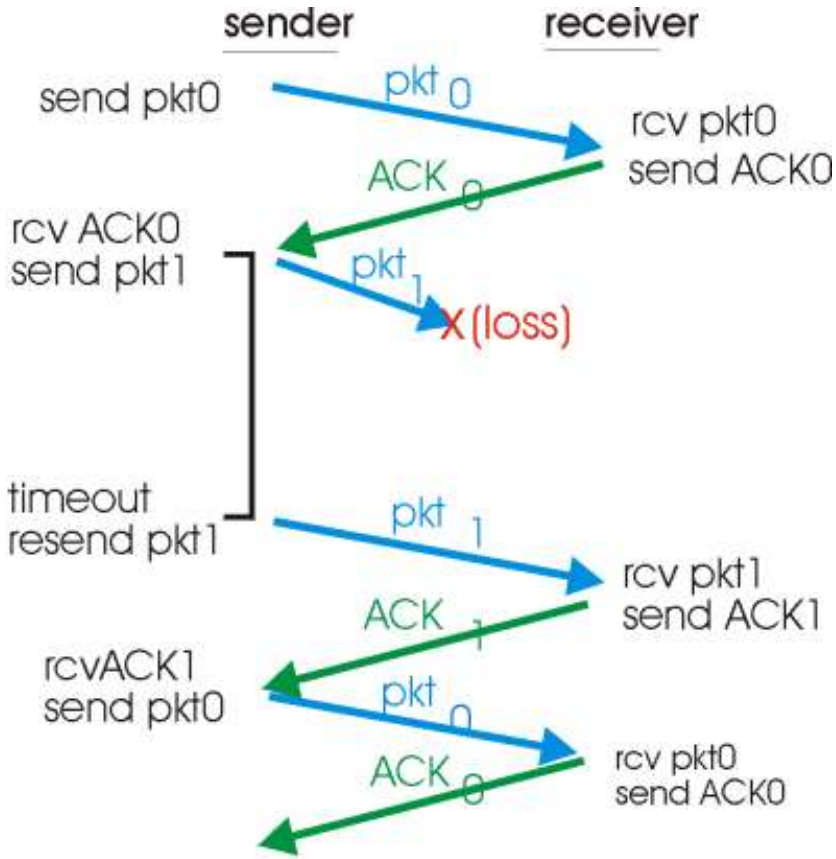
问题：

- ACK丢失或超时设置过短导致的重发，会在接收端产生冗余分组
- 用分组序号解决

rdt3.0 in action

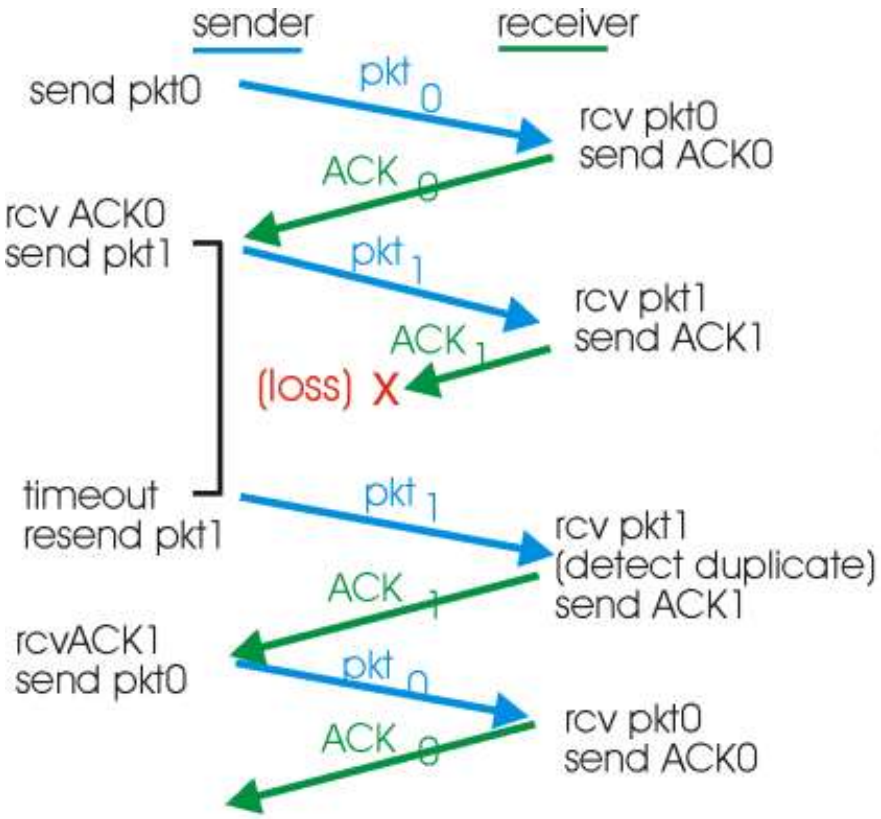


(a) operation with no loss

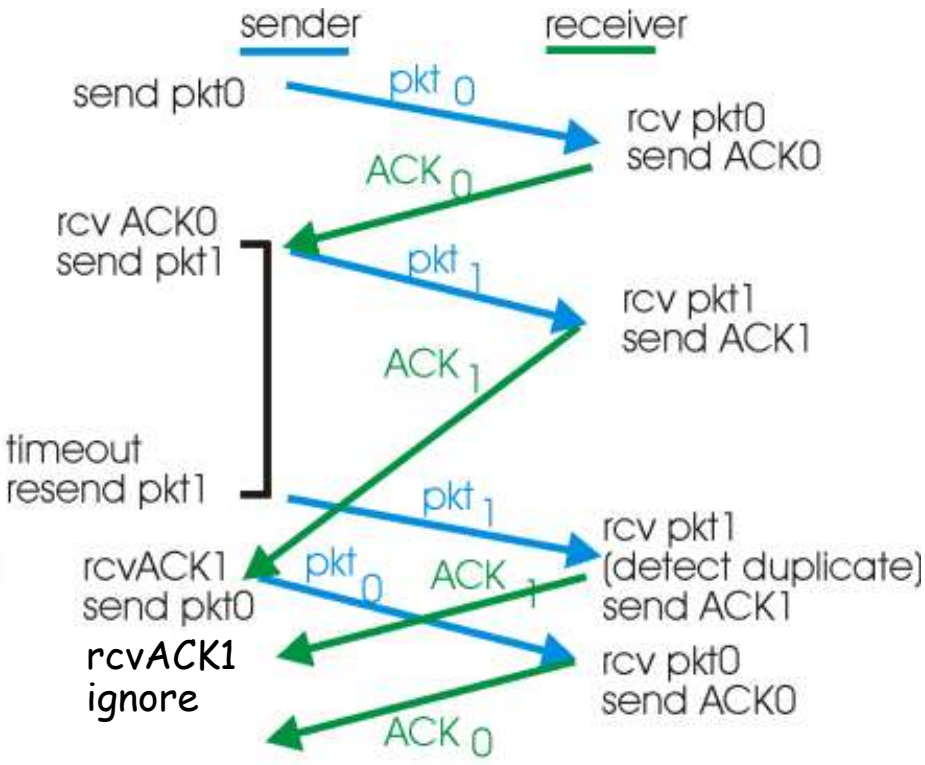


(b) lost packet

rdt3.0 in action



(c) lost ACK



(d) premature timeout

发送方FSM：Rdt2.2 vs Rdt3.0

Rdt2.2：

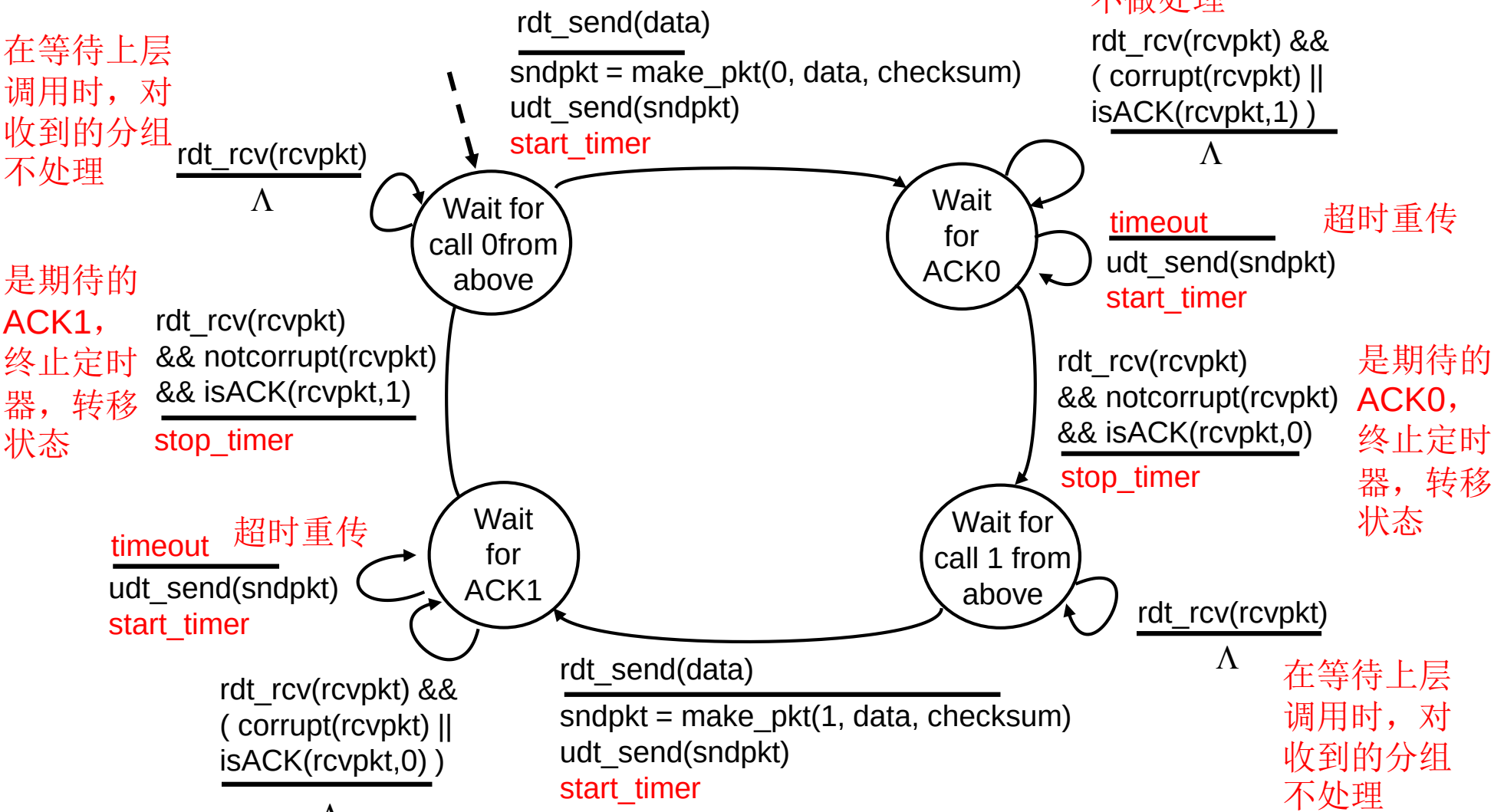
- 发送分组，等待确认
- 收到期待序号的ACK，转移状态
- 不是期待序号的ACK，重发分组（说明该分组丢了）

Rdt3.0：

- 发送分组后启动一个定时器
- 收到期待序号的ACK，终止定时器，转移状态
- 收到非期待序号的ACK，不做处理（为什么？）
- 定时器超时后，重发分组

- 与rdt2.2相比，rdt3.0增加了定时器的处理，并且重传仅由超时触发，对于不是当前等待的事件（期待序号的ACK分组、上层调用）均不做处理。

rdt3.0 sender



接收方FSM：Rdt2.2 vs Rdt3.0

Rdt3.0新增的事件

- 数据分组丢失：
  - 接收方感知不到丢包，该事件对接收方FSM无影响
- ACK丢失、过早超时：
  - （发送方重发）接收方收到重复的数据分组
  - 接收方利用序号检测重复分组，重发前一次ACK

Rdt2.2

- ACK损坏：
  - （发送方重发）接收方收到重复的数据分组
  - 接收方利用序号检测重复分组，重发前一次ACK

Rdt2.2的接收方FSM不加修改就可以用于rdt3.0的接收方

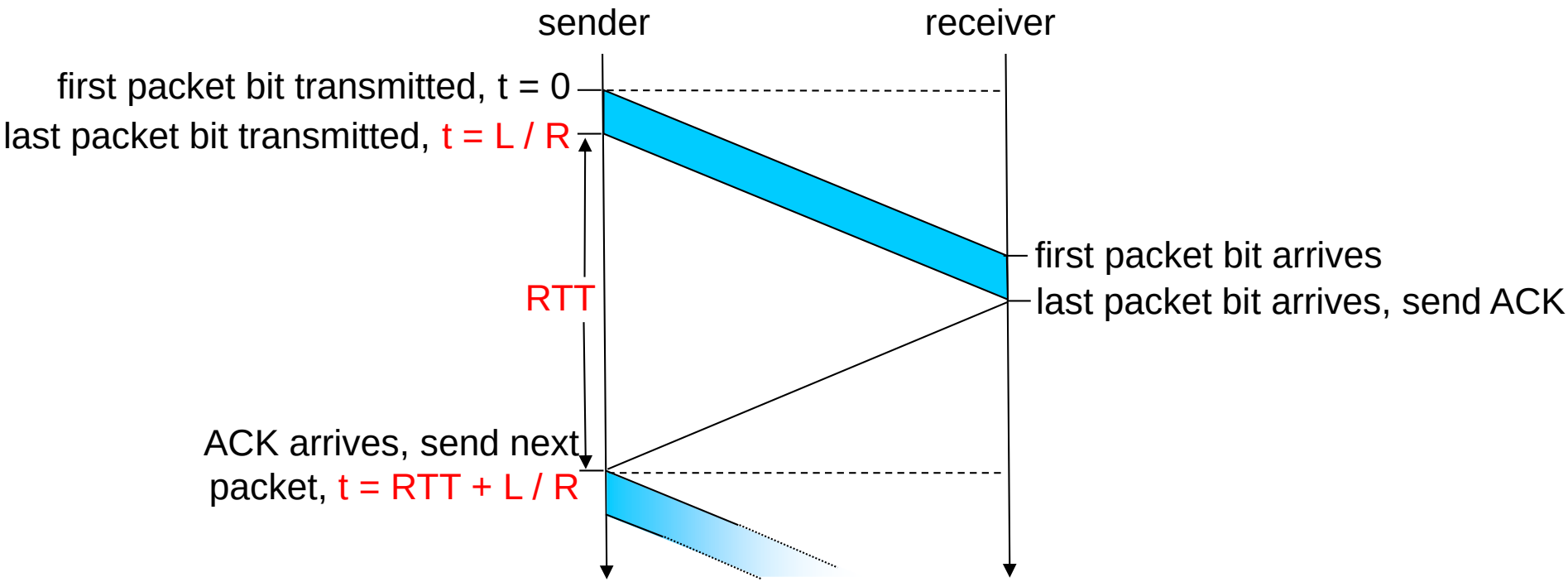
Rdt协议小结

- Rdt 1.0（理想信道）
  - 发送方只管发送，接收方只管接收
- Rdt 2.2（有比特错误的信道）
  - 分组检错，使用分组序号，使用带序号的ACK
  - 发送方/接收方未收到期待序号的ACK/分组时，重传当前分组/ACK
- Rdt 3.0（有比特错误和丢包的信道）
  - 分组检错，使用分组序号，使用带序号的ACK
  - 发送方定时器超时后重发分组
  - 接收方未收到期待序号的分组时，重发ACK

梳理可靠传输协议的设计过程

1. 不可靠信道有哪些出错类型：比特错误，丢包
2. 如何发现错误：接收方通过检错码发现出错包，利用定时器发现丢失包
3. 采取什么恢复措施：重传出错或丢失的包
4. 恢复措施引入的问题：ACK出错、ACK丢失或超时设置过短，导致接收端出现冗余分组
5. 如何解决冗余分组的问题：给分组加上序号，序号多长
6. 将以上措施汇总，画出各种情形下的时间线图（正常，分组出错，ACK出错，分组丢失，ACK丢失，过早超时）
7. 归纳发送方、接收方的事件类型及采取的动作，画出FSM

rdt3.0是一种停-等（stop-and-wait）协议



$$U_{sender} = \frac{L / R}{RTT + L / R}$$

rdt3.0的性能

rdt3.0是一个正确的协议，但性能不佳！

- example: 1 Gbps link, 30 ms RTT, 1KB packet:

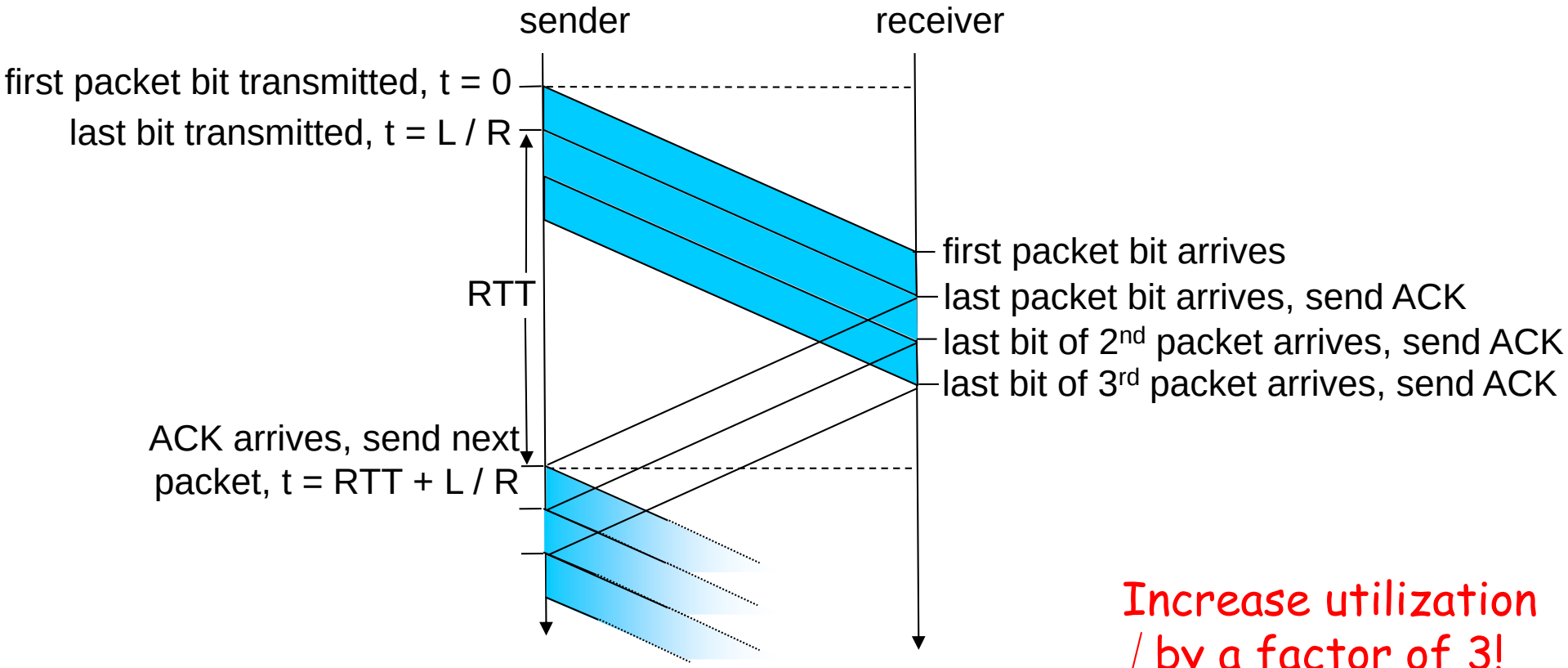
$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10**9\ b/sec} = 8 \text{ microsec}$$

  - U_{sender}: utilization – fraction of time sender busy sending

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

  - 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - network protocol limits use of physical resources!

流水线：提高链路利用率



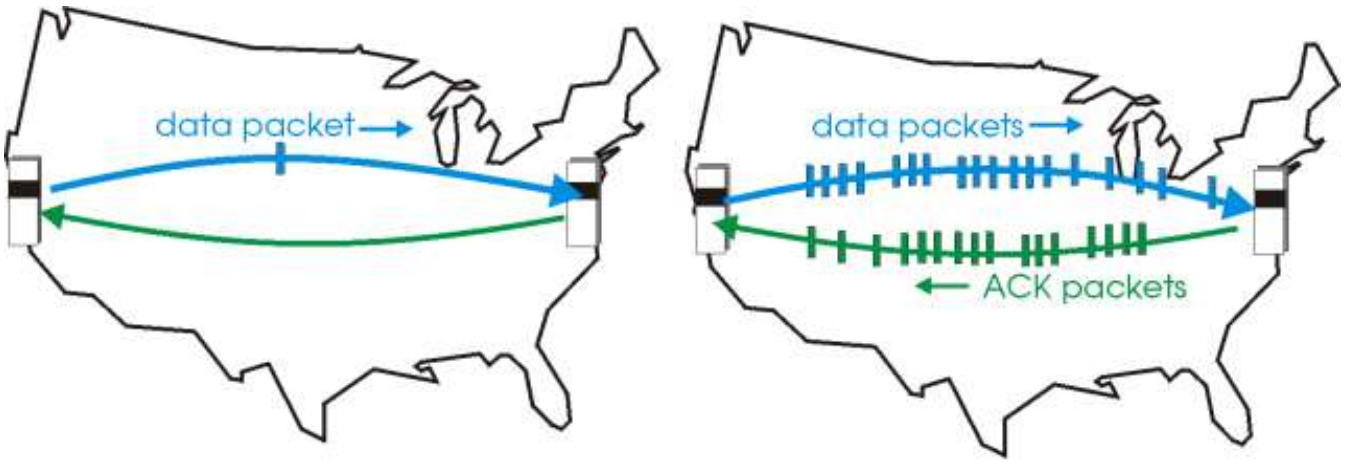
Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

3.4.2 流水线可靠数据传输协议

流水线协议： 允许发送方有多个已发送、未确认的分组

- 分组的序号范围应扩大（停等协议只使用1比特序号）
- 发送端和接收端可能需要缓存多个分组（停等协议中，发送端缓存一个分组，接收端不缓存）



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

(b) a pipelined protocol in operation

- 两种基本的流水线协议: *go-Back-N, selective repeat*

Go-Back-N的收发规则

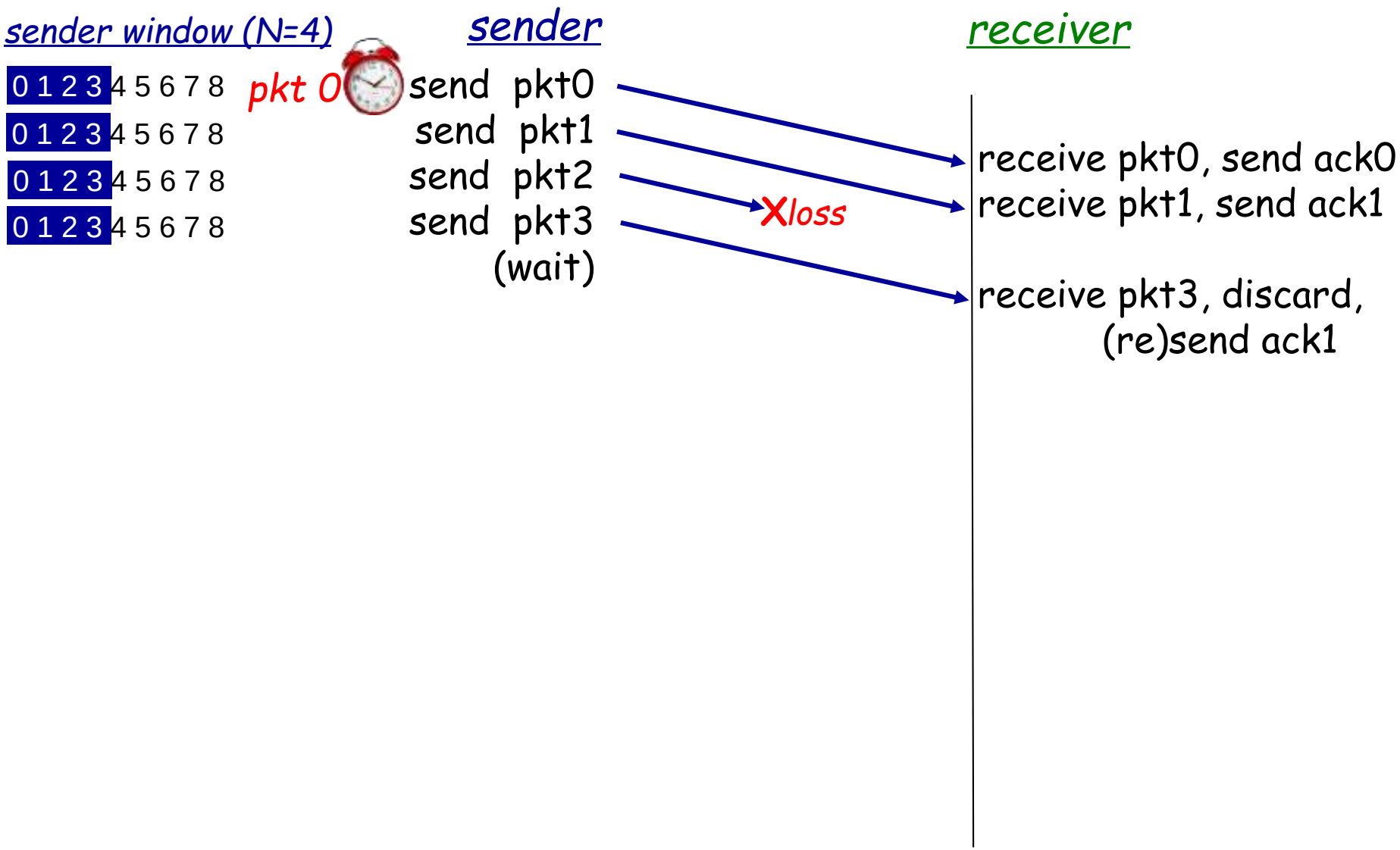
Sender:

- 最多允许N个已发送、未确认的分组
- 对于最早的已发送、未确认的分组使用一个定时器
- 若定时器超时，**从最早已发送、未确认的分组开始，顺序发送其后的分组**

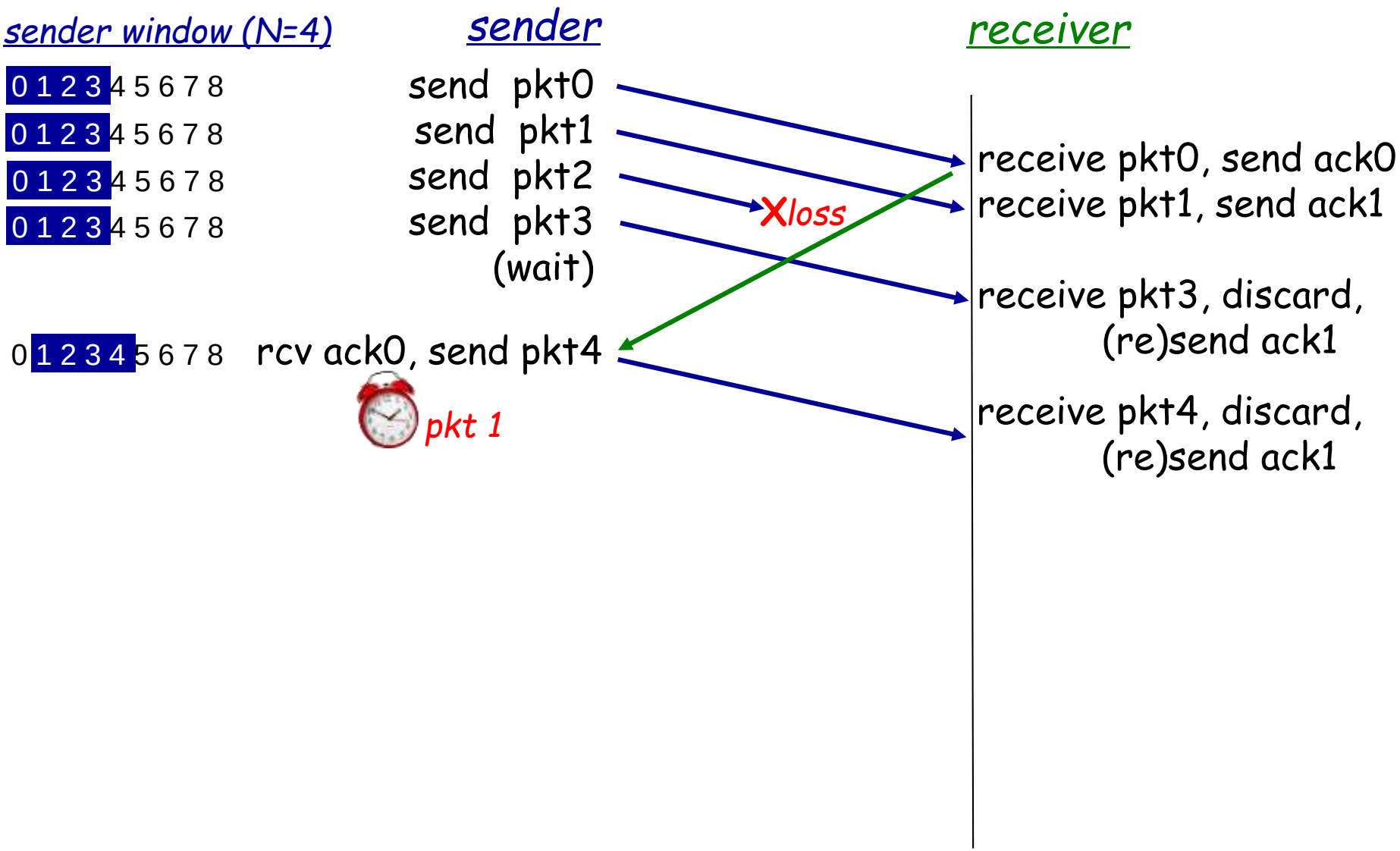
Receiver:

- 每收到一个分组，都要发送一个带序号的ACK
- **对于未按顺序到来的分组，丢弃，**重发最近一次的ACK
- 使用**累积确认**：若ACK包含序号q，表明“序号至q的分组均正确收到”

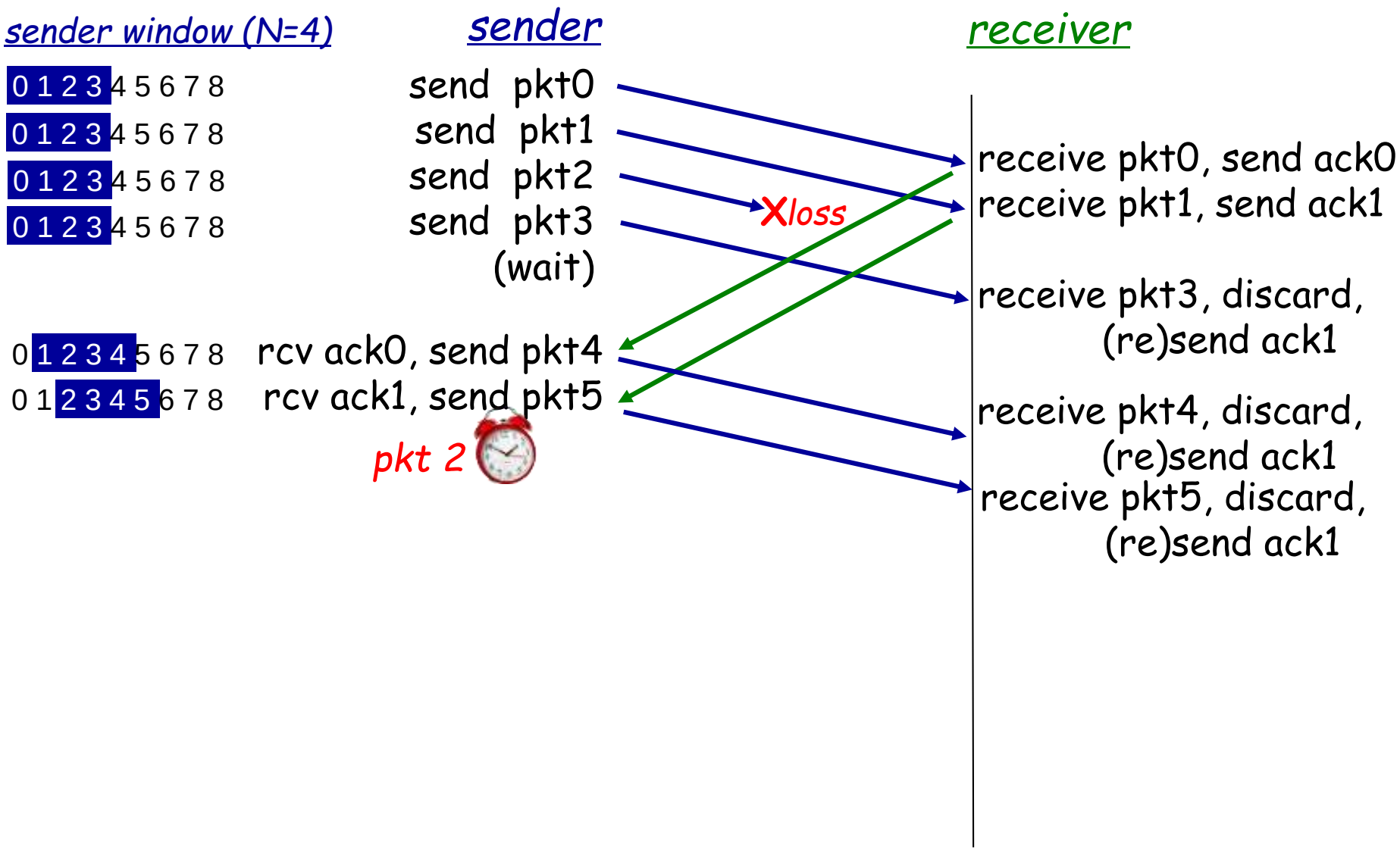
GBN in action



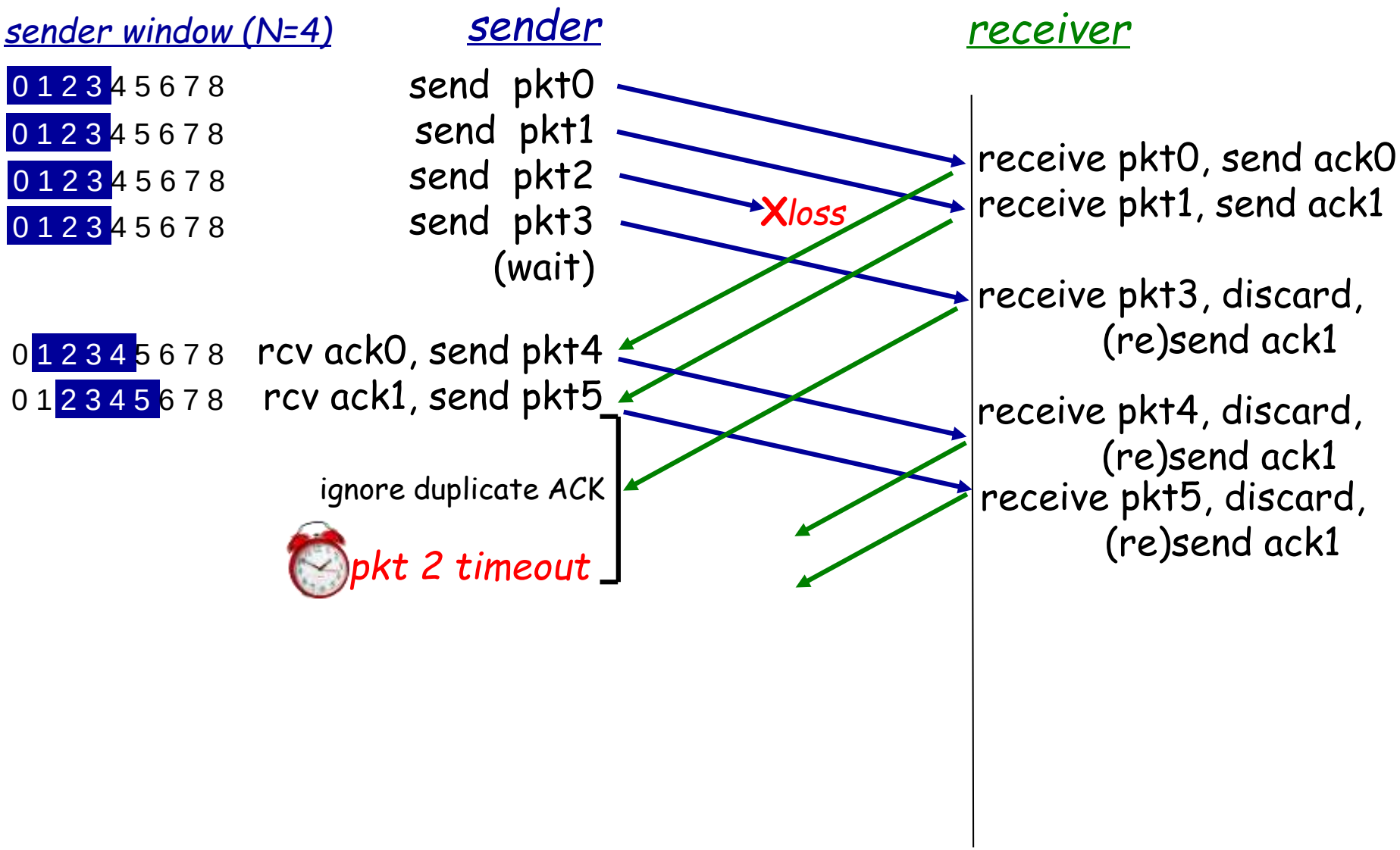
GBN in action



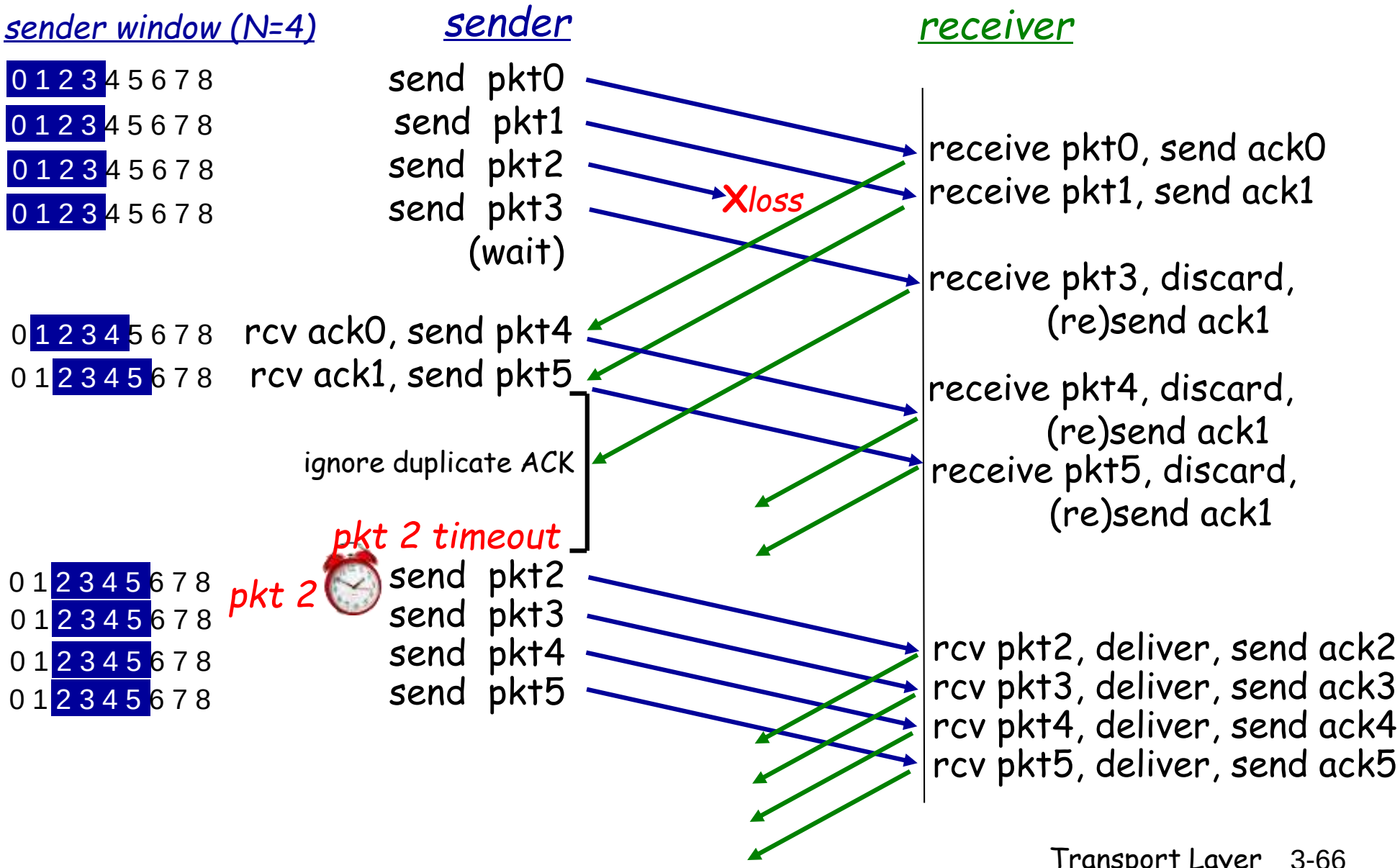
GBN in action



GBN in action



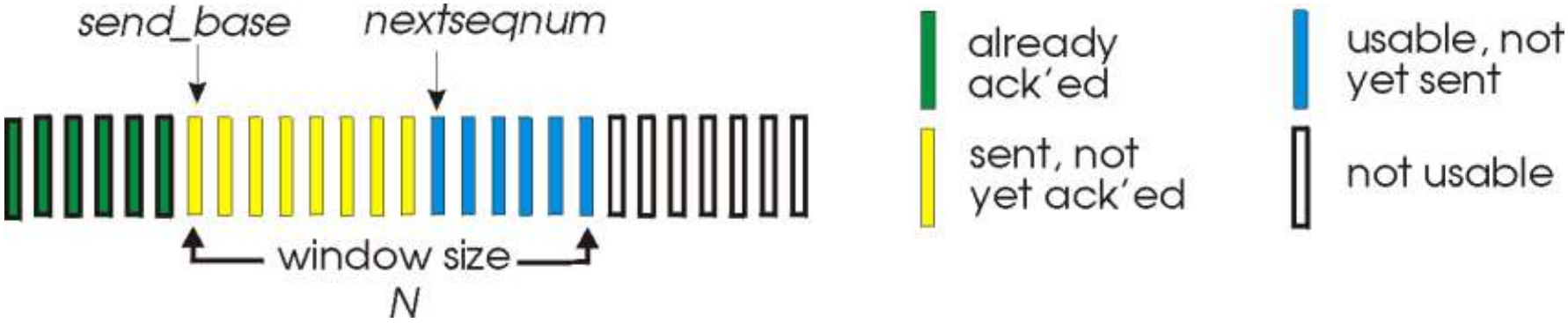
GBN in action



Go-Back-N的概念和术语

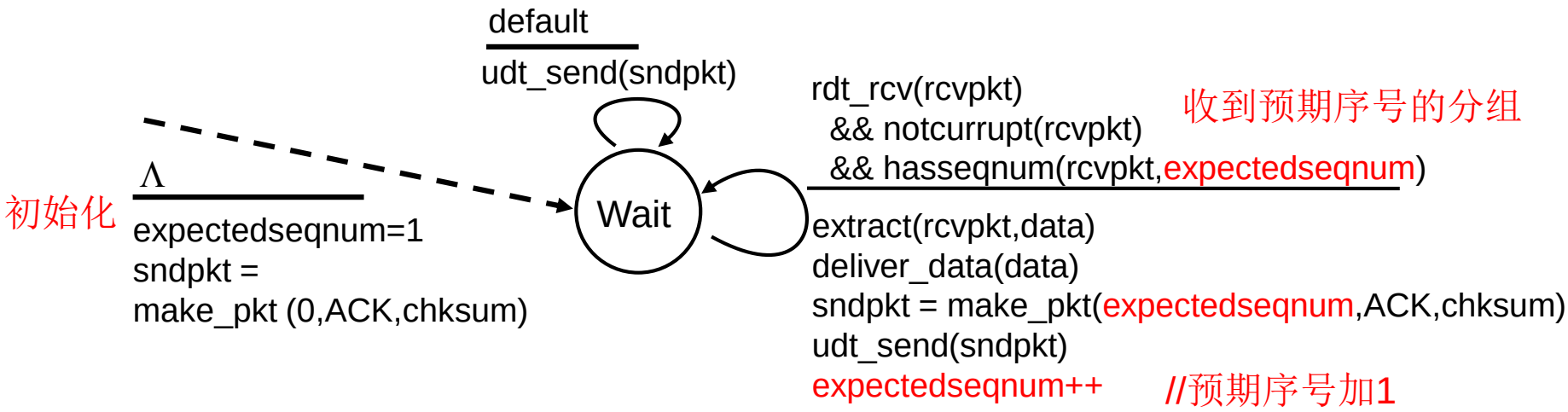
Sender:

- 假设最多允许N个已发送、未确认的分组
- 发送端看到的序号空间（由序号长度决定）划分为以下4个区域：



- 已发送未确认序号 + 未发送可用序号 = 发送窗口（包含N个序号）
- Ack分组携带所确认分组的序号
- 使用累积确认: 若ACK包含序号q，表明“序号至q的分组均正确收到”
- 若发送方收到ACK q，则更新基序号为 q+1，整体滑动发送窗口
- 发送方只对基序号分组使用一个定时器
- 超时：发送方重传发送窗口中从基序号开始的所有分组

GBN: 接收方FSM

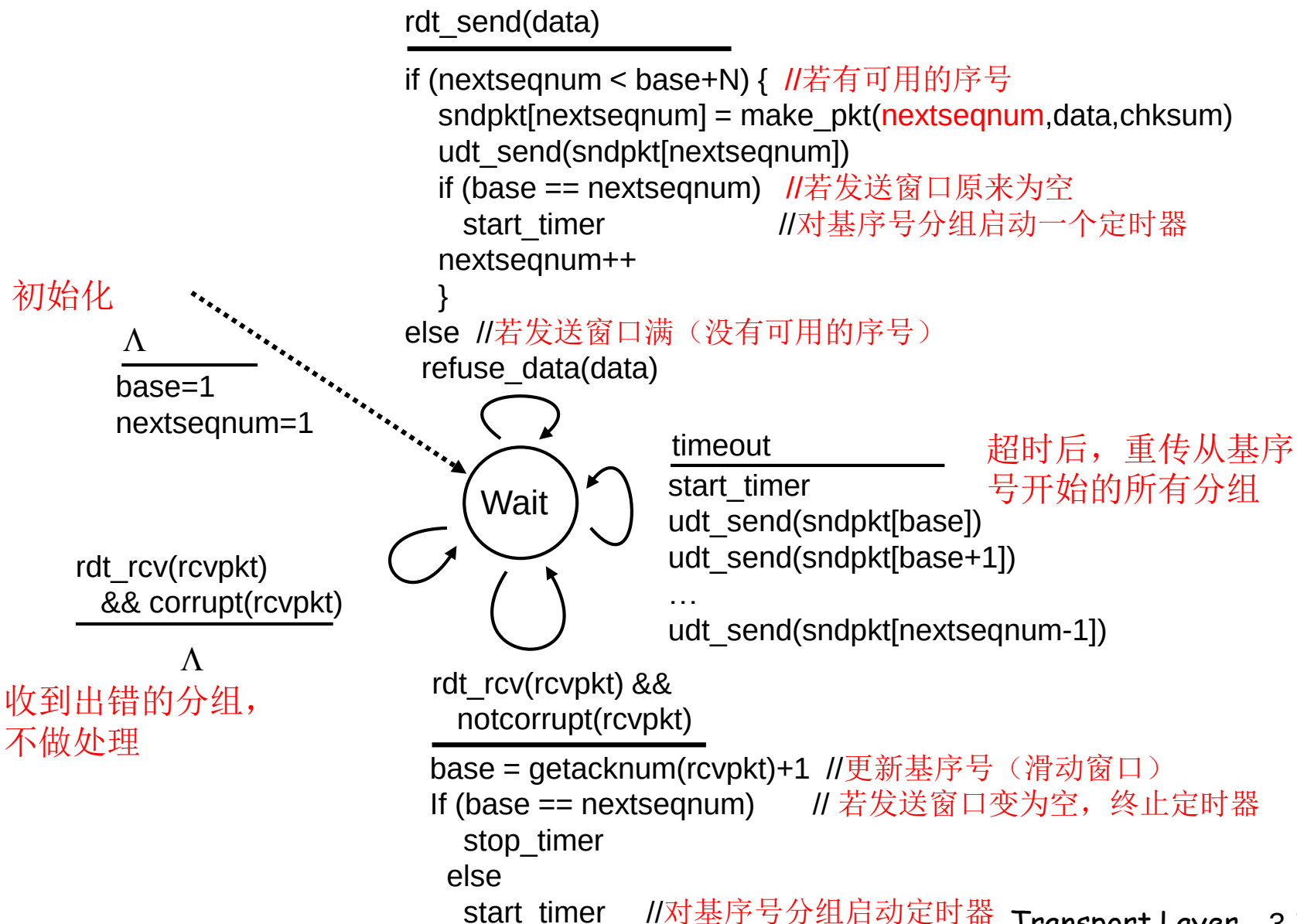


- 接收方只接收预期序号的分组：
  - 失序的分组一律丢弃，并重发前一次的ack，这意味着接收方只能按顺序接收分组
- 接收方使用累积确认：
  - 只对正确收到且序号连续的一系列分组中的最高序号进行确认

GBN的发送方

- 收到上层的发送请求：
  - 若发送窗口满：拒绝请求
  - 若发送窗口不满：
    - 构造分组并发送（从下一个可用序号开始设置）
    - 若原来发送窗口为空：对基序号启动一个定时器
- 收到*ACK q*：
  - 更新基序号为*q+1*
  - 若发送窗口变为空：终止定时器
  - 若发送窗口不空：对基序号启动一个定时器
- 收到出错的*ACK*：
  - 不做处理，为什么？
  1. 若ack p出错，可由后续收到的ack q（q>p）进行累积确认
  2. 若ack p出错，且后续未收到ack，则超时后重传 pkt p
- 定时器超时：
  - 启动定时器，从已发送未确认的分组开始，发送位于当前发送窗口内的所有分组

GBN: 发送方FSM


rdt_send(data)
if (nextseqnum < base+N) { //若有可用的序号
sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
udt_send(sndpkt[nextseqnum])
if (base == nextseqnum) //若发送窗口原来为空
start_timer //对基序号分组启动一个定时器
nextseqnum++
}
else //若发送窗口满（没有可用的序号）
refuse_data(data)
初始化
Λ
base=1
nextseqnum=1
Wait
timeout
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
…
udt_send(sndpkt[nextseqnum-1])
超时后，重传从基序号开始的所有分组
rdt_rcv(rcvpkt)
&& corrupt(rcvpkt)
Λ
收到出错的分组，
不做处理
rdt_rcv(rcvpkt) &&
notcorrupt(rcvpkt)
base = getacknum(rcvpkt)+1 //更新基序号（滑动窗口）
If (base == nextseqnum) // 若发送窗口变为空，终止定时器
stop_timer
else
start_timer //对基序号分组启动定时器

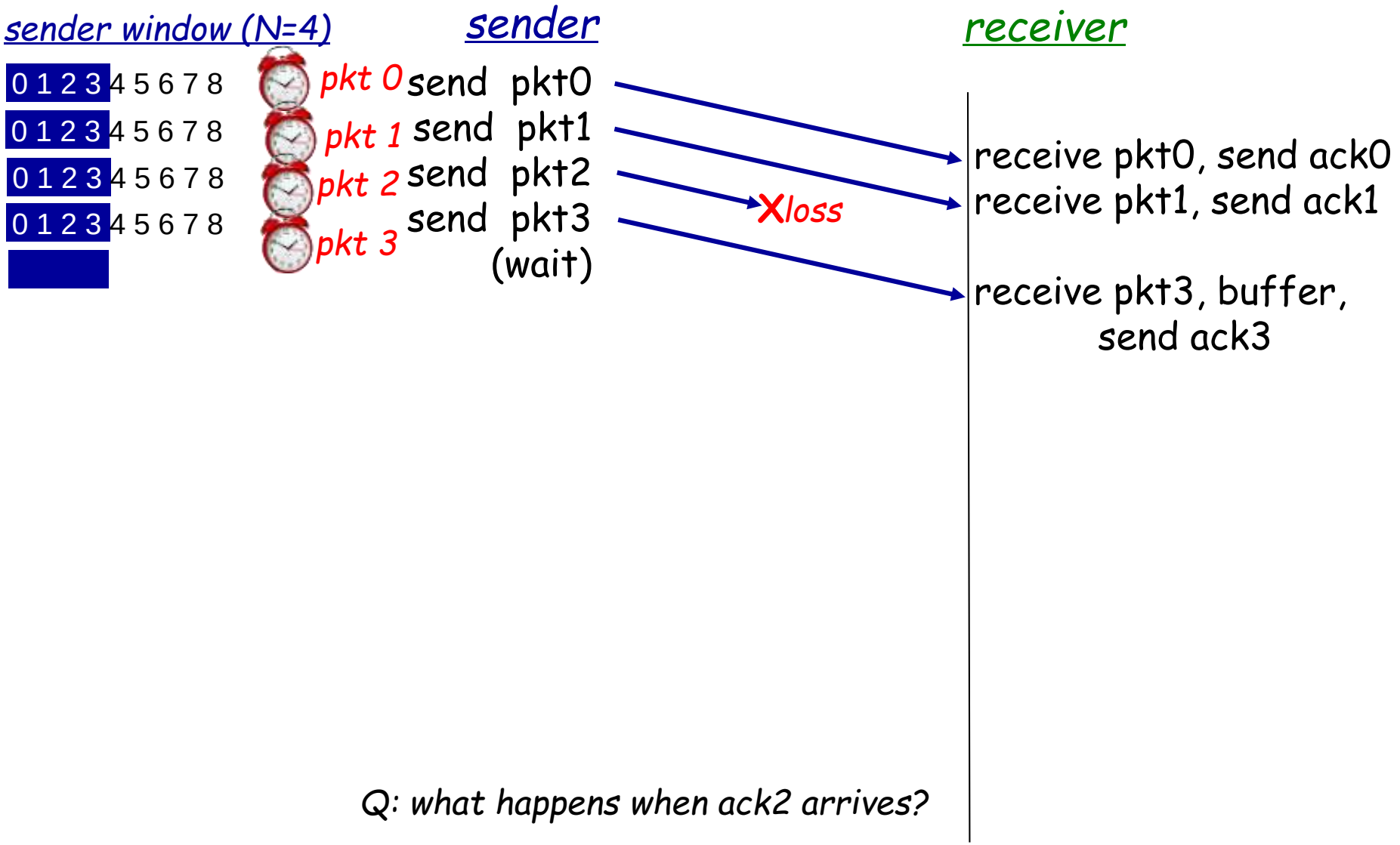
GBN小结

- 接收端：
  - 按顺序接收分组，不缓存失序的分组
  - 按照累积确认的要求发送ACK
- 发送端：
  - 仅当发送窗口不满时，才能发送新的分组
  - 收到确认后，更新发送窗口
  - 当发送窗口发生变化（空<->非空，基序号更新）时，需要重新设置定时器
  - 超时后，重发发送窗口中的全部分组
- 特点：
  - 接收端简单，发送端复杂，出错后需要较多的重传

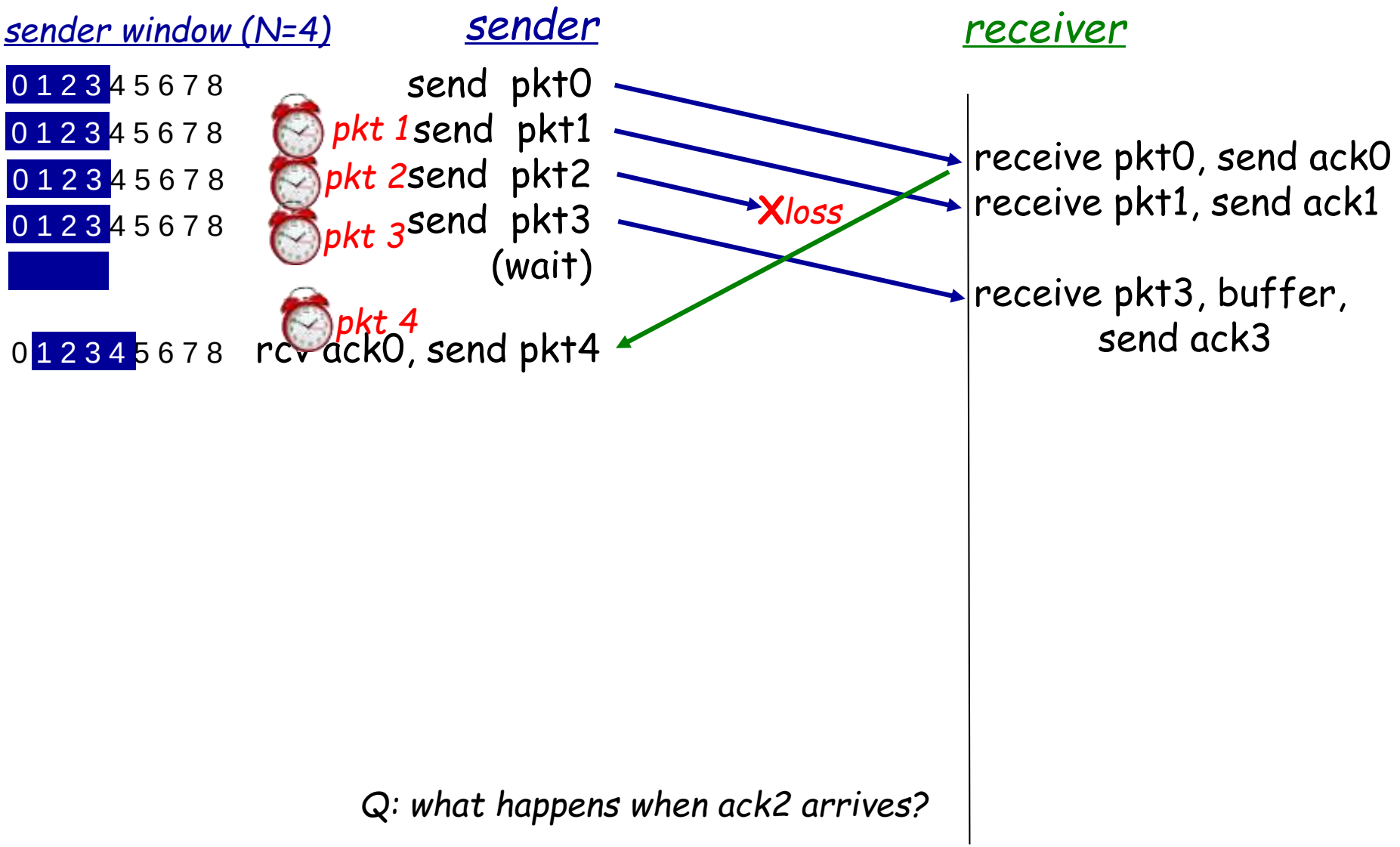
选择重传（Selective Repeat）

- 要点：
  - 发送方仅重传它认为出错（未收到ACK）的分组，以避免不必要的重传
- 为此：
  - 接收端需缓存失序的分组
  - 接收端需对每个正确收到的分组单独确认（选择确认） **（注意：选择重传不使用累积确认）**
  - 发送的每个分组需要一个定时器，以便被单独重发

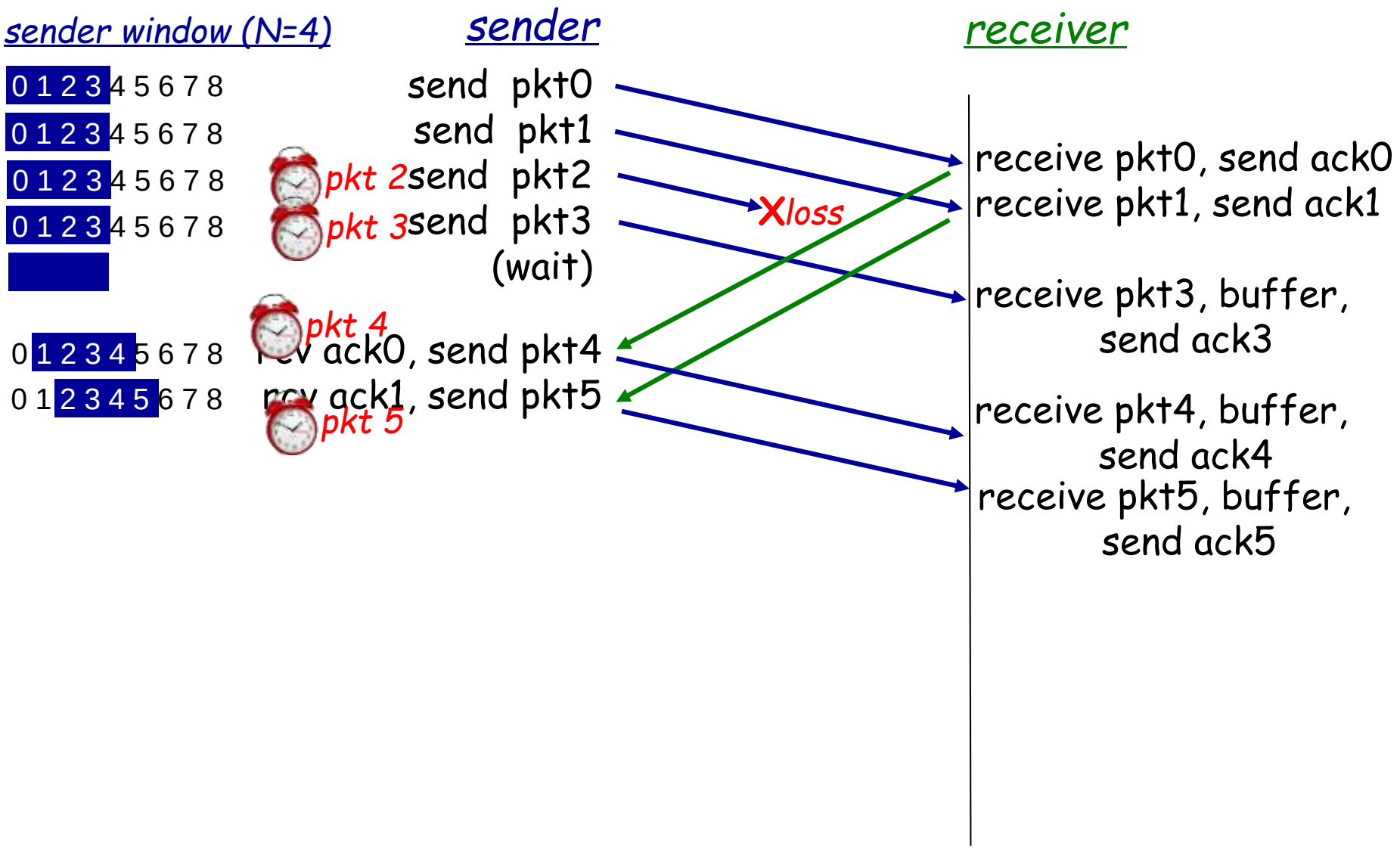
Selective repeat in action



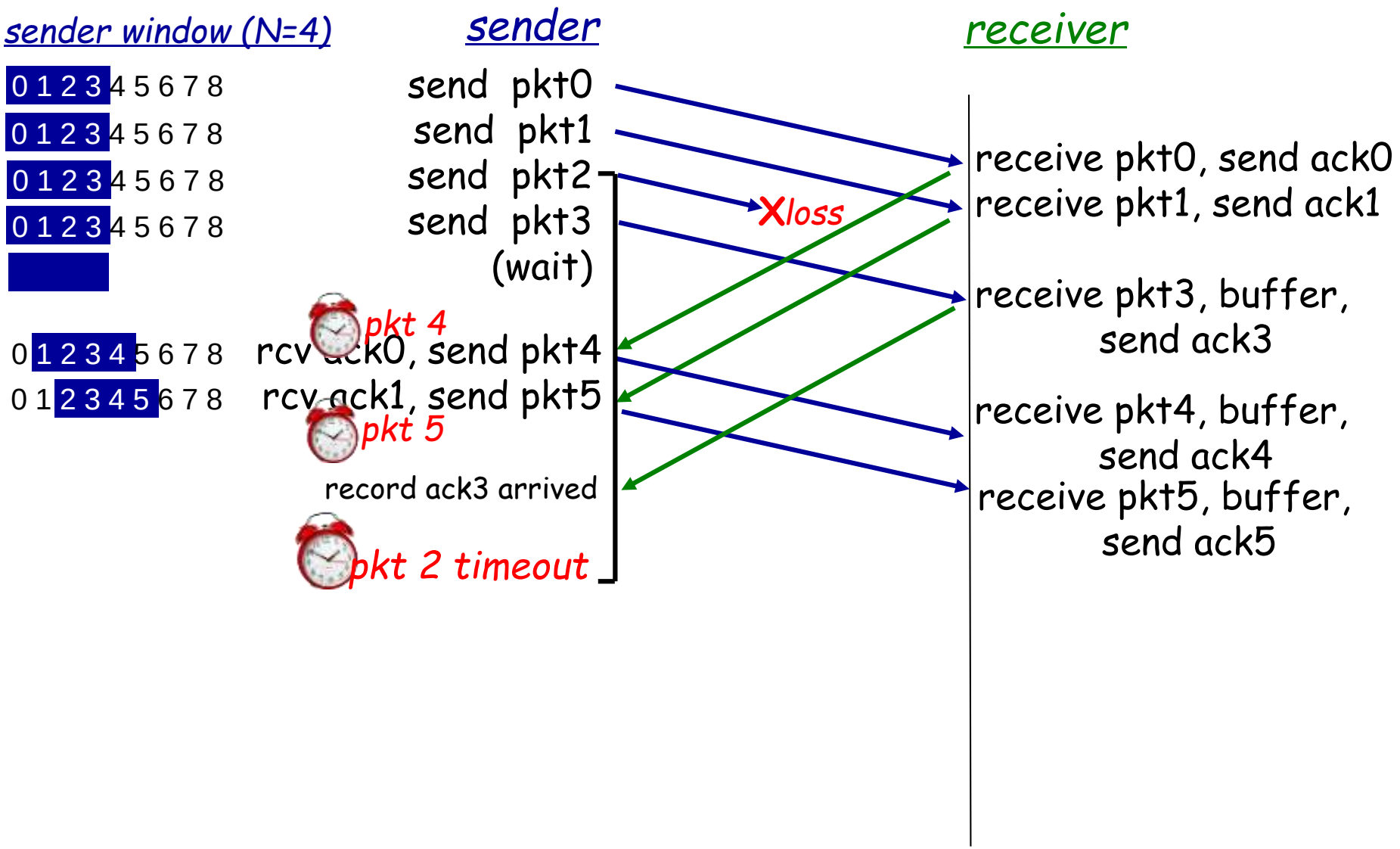
Selective repeat in action



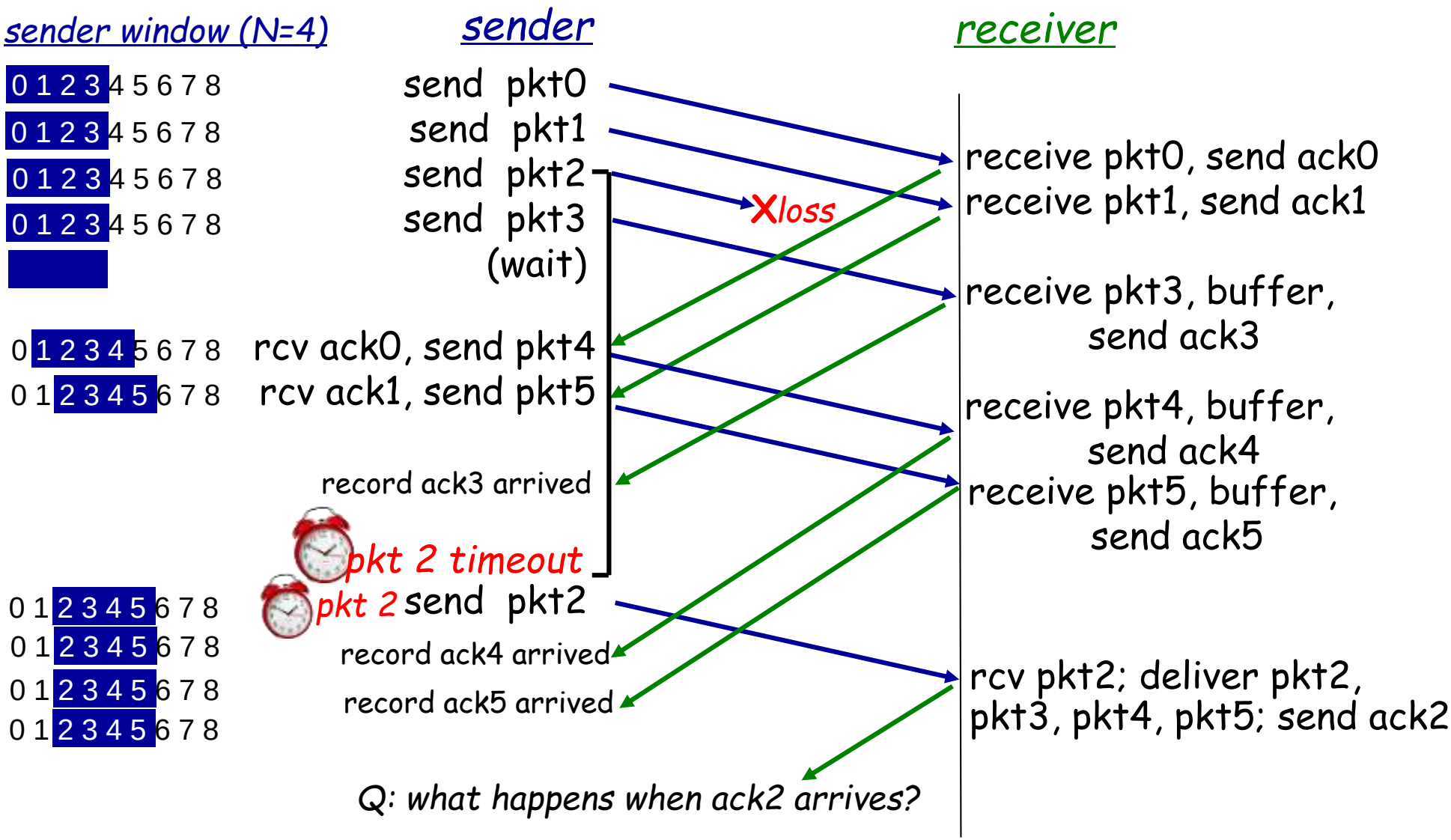
Selective repeat in action



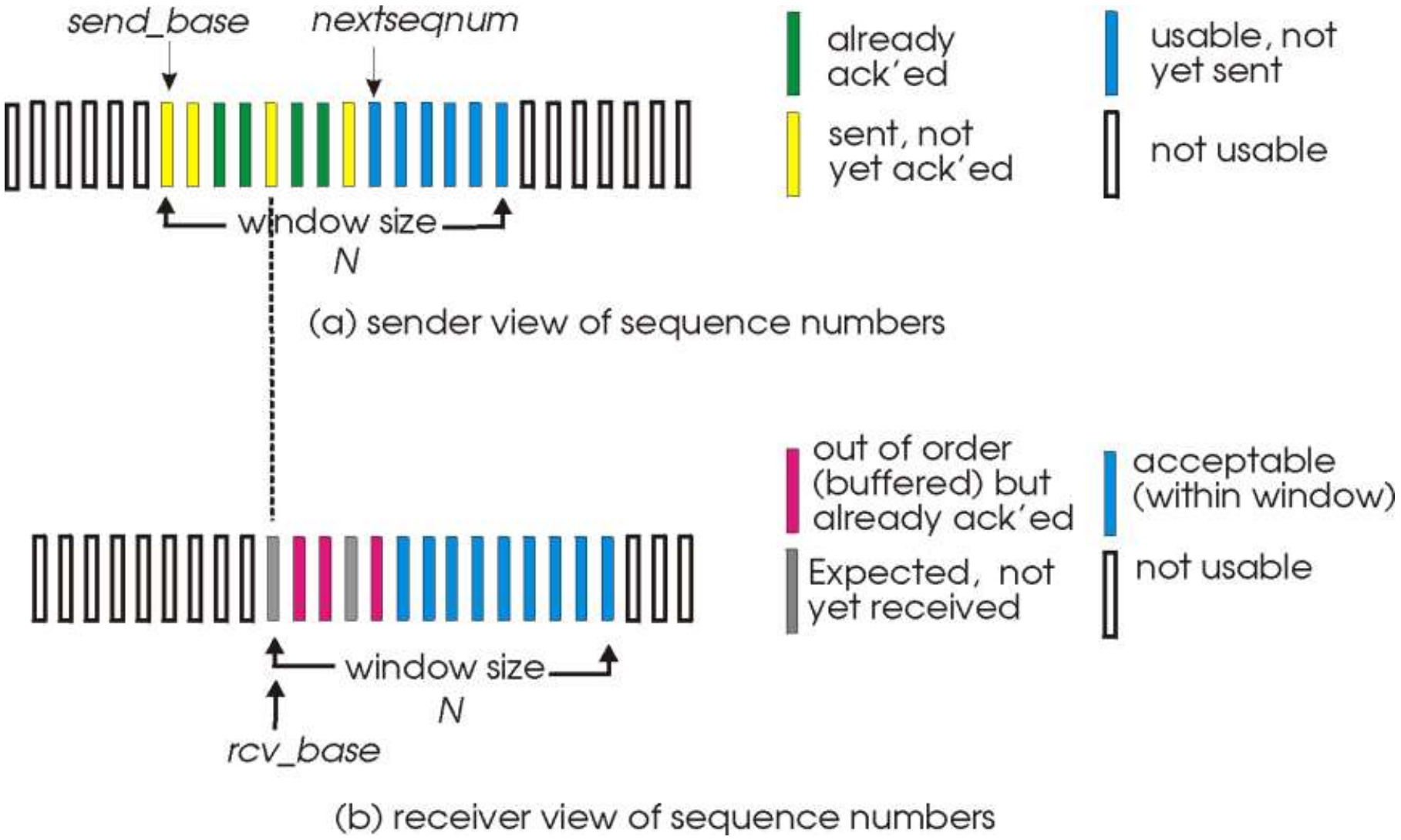
Selective repeat in action



Selective repeat in action



发送方和接收方看到的序号空间



SR的发送窗口和接收窗口

- 发送窗口：
  - 包含"已发送未确认"和"未发送可用"的序号
  - 可能有"已发送已确认"的序号交织其中
  - 基序号是"已发送未确认"或"未发送可用"的序号
  - 收到基序号的ACK时，滑动发送窗口
- 接收窗口：
  - 包含"期待但未收到"和"允许接收"的序号
  - 可能有"已确认已缓存"的序号交织其中
  - 基序号为"期待但未收到"或"允许接收"的序号
  - 收到基序号分组时，按顺序交付分组，滑动接收窗口

选择重传的工作过程

sender

从上层接收数据:

- 若发送窗口未满，发送分组，启动定时器

定时器 n 超时:

- 重传分组n, 重启定时器

收到发送窗口内的ACK(n):

- 标记分组n为已确认
- 若n=基序号，滑动发送窗口，使基序号=最小未确认的序号或下一个序号

其余情形:

- 忽略

receiver

收到接收窗口内的分组n:

- 发送ACK(n)
- 若为失序分组：缓存该分组
- 若n=基序号：交付从n开始的若干连续分组；滑动接收窗口，使基序号=下一个期待接收的序号

收到[rcvbase-N,rcvbase-1]内的分组n

- 发送ACK(n)

其余情形:

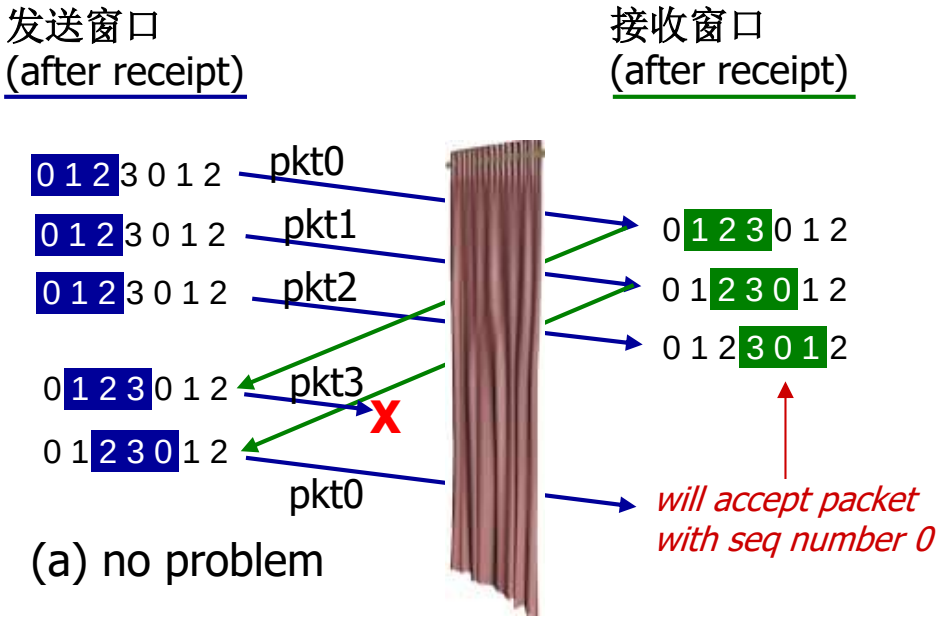
- 忽略

Selective repeat: dilemma

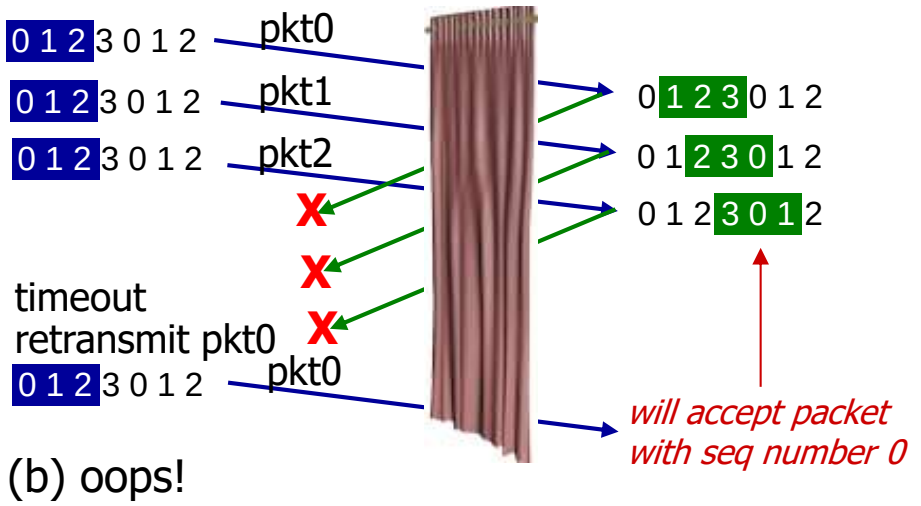
example:

- 序号空间：[0, 3]
- 窗口大小=3
- 第二种情形：重复分组被当成新分组接收下来

Q：窗口大小和序号空间大小应当满足什么条件，才能避免出现错误？



receiver can't see sender side.
receiver sees no difference in two cases!
something's (very) wrong!



选择重传：窗口大小和序号空间的关系

- 选择重传：通常发送窗口大小 = 接收窗口大小
- 考虑以下情形：发送方发送了一个满窗口（[0,N-1]）的分组；接收方全都接收正确，发送了*ACK*，并滑动接收窗口至[N,2N-1]。但N个*ACK*全都没有正确接收，发送端超时后逐个重发这N个分组
- 为使发送端能够移动发送窗口，接收端必须对[0,N-1]中的分组进行确认，这回答了接收方为什么需要对[rcvbase-N, rcvbase-1]（前一个满窗口）中的分组进行确认
- 为使接收端不会将重发的分组当成新的分组，窗口[0,N-1]和窗口[N,2N-1]不能重叠。所以，N不能大于序号空间的一半

SR小结

- 接收端：
  - 落在接收窗口中的分组都要接收，每个分组单独确认
  - 尽管允许乱序接收，但仍须按顺序交付数据
- 发送端：
  - 发送窗口不满时可以发送新的分组
  - 每个已发送的分组都需要一个定时器
  - 若定时器n超时，只重发序号为n的分组
- 特点：
  - 出错后重传代价小，发送端使用较多定时器，接收端需要较大缓存，实现复杂

梳理流水线机制的设计

- 一次允许发送多个包会带来什么新的问题：
  - 序号长度k，窗口大小N和k有什么关系，发现分组出错或丢失怎么处理
- GBN：一旦发现出错后，接收方不再接收新的包，双方从出错的包开始重新发送和接收
- SR：发现出错后，接收方可以继续接收落在接收窗口内的包，过后仅发送/接收出错的包
- 画出GBN和SR的时间线图，据此画出FSM

要求掌握

- 表3.1归纳了可靠数据传输协议中引入的机制及用途，理解这些机制是为了解决什么问题而引入的
- 学会画各种场景的时间线图，根据时间线图总结发送方和接收方要处理的事件及应当采取的动作，进而给出有限状态机描述
- 掌握流水线传输机制，理解GBN和SR的协议机制及不同，理解窗口大小和序号长度的关系

下层信道模型的一个遗留假设

- 可能产生比特错误、丢包和分组重排序的下层信道：
  - 后发的分组可能早于先发的分组到达
  - 当连接两端的信道是一个网络时，可能发生重排序
- 新的问题：
  - 一个很老的分组可能到达接收端，并且其序号刚好落在当前接收窗口内（序号重用引起的问题）
- 解决方法：
  - 假设分组在网络中的“存活”时间不超过某个固定值
  - 使用很长的序号，确保序号回绕时间超过该固定值

Chapter 3 outline

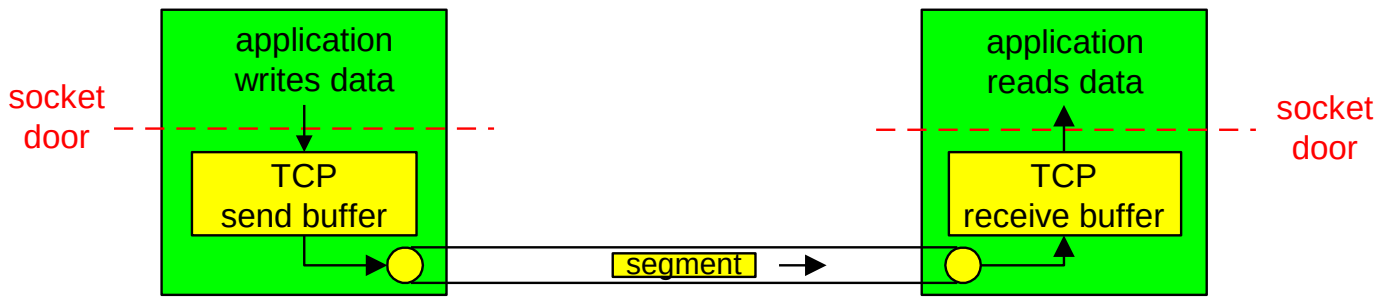
- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

3.5.1 TCP: Overview

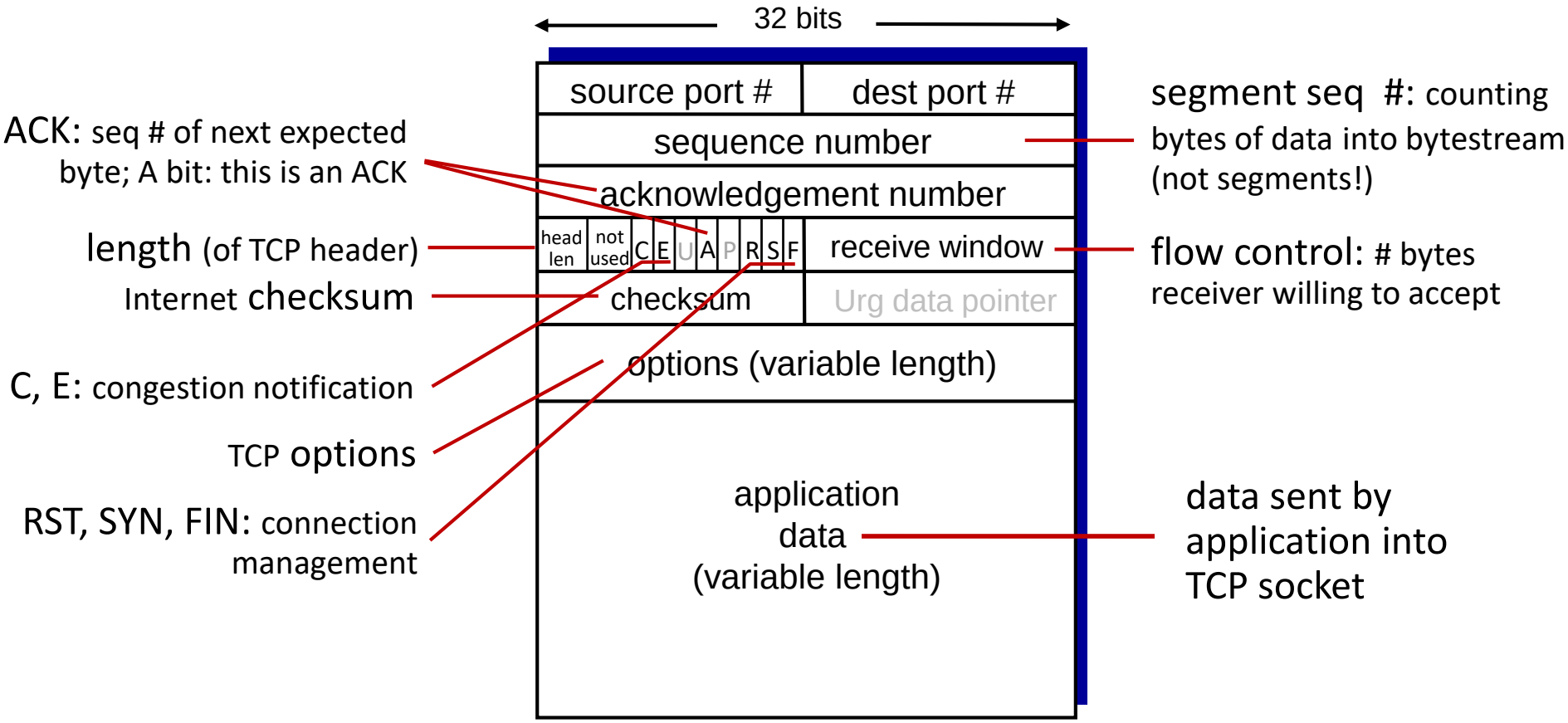
- TCP服务模型：
  - 连接一对通信进程的、理想的字节流管道
- 点到点通信：
  - 涉及一对通信进程
- 全双工：
  - 可以同时双向传输数据
- 可靠、有序的字节流：
  - 不保留报文边界

需要的机制

- 建立连接：
  - 通信双方为本次通信建立数据传输所需的状态（套接字、缓存、变量等）
- 流水式发送报文段：
  - 可靠数据传输
- 流量控制：
  - 通过调整发送速率，不使接收方缓存溢出



3.5.2 TCP报文段结构



重要的TCP选项

- 最大段长度（MSS）：
  - TCP段中可以携带的最大数据字节数
  - 建立连接时，每个主机可声明自己能够接受的MSS，缺省为536字节
- 窗口比例因子（window scale）：
  - 建立连接时，双方可以协商一个窗口比例因子
  - 实际接收窗口大小 = window size * 2^window scale
- 选择确认（SACK）：
  - 最初的TCP协议只使用累积确认
  - 改进的TCP协议引入选择确认，允许接收端指出缺失的数据字节

序号和确认序号

序号:

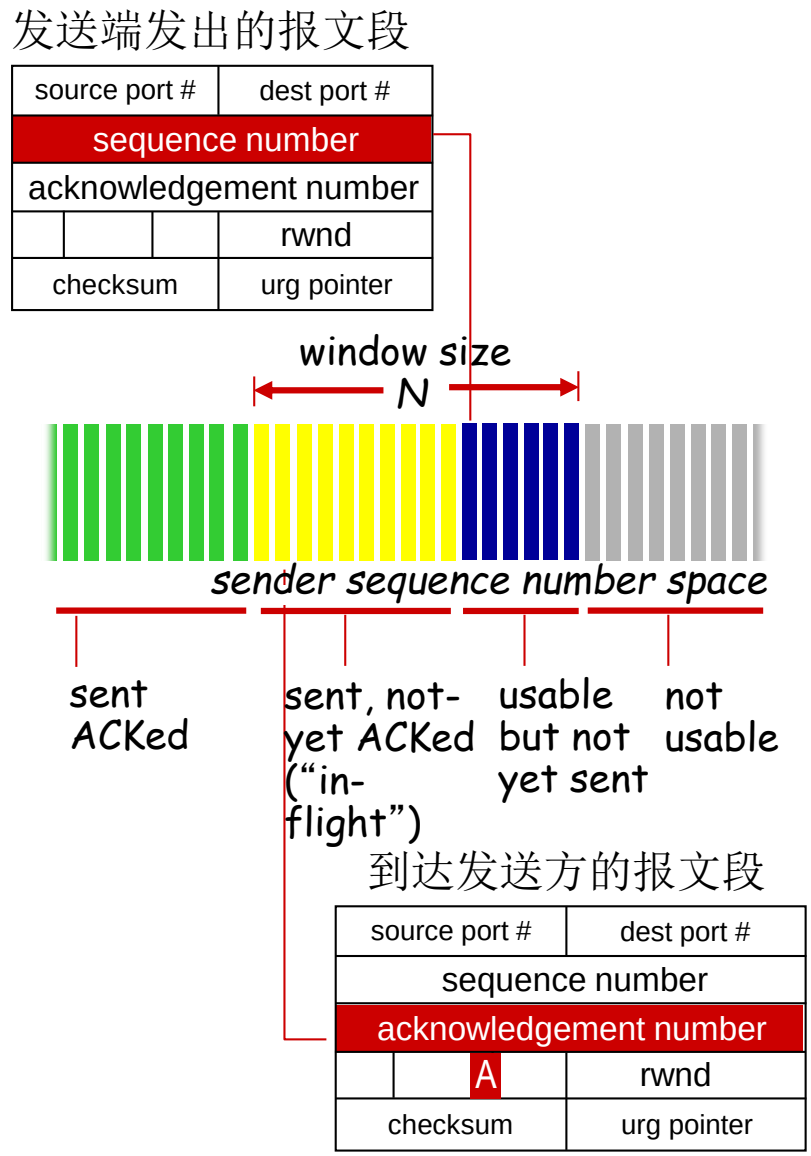
- 报文段中第一个数据字节的序号

确认序号:

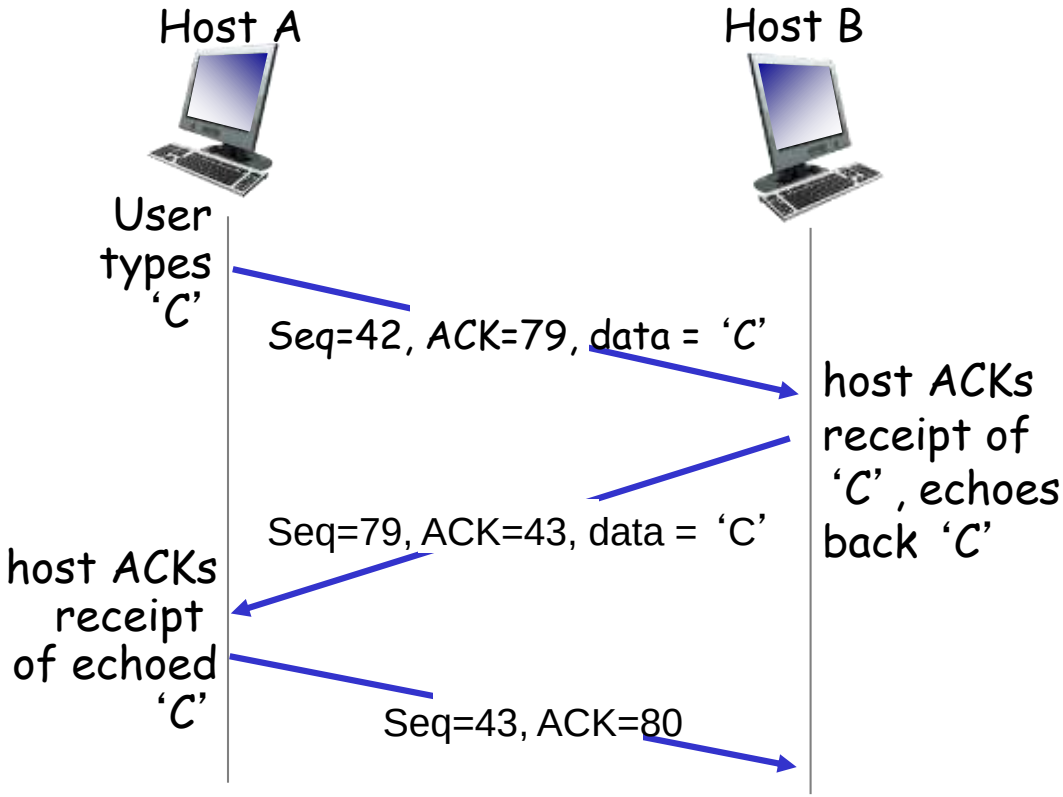
- 使用累积确认，指出期望从对方接收的下一个字节的序号

Q: 接收方如何处理失序的报文段？

A: TCP规范未涉及，留给实现者考虑



序号和确认序号：举例



simple telnet scenario

TCP超时设置

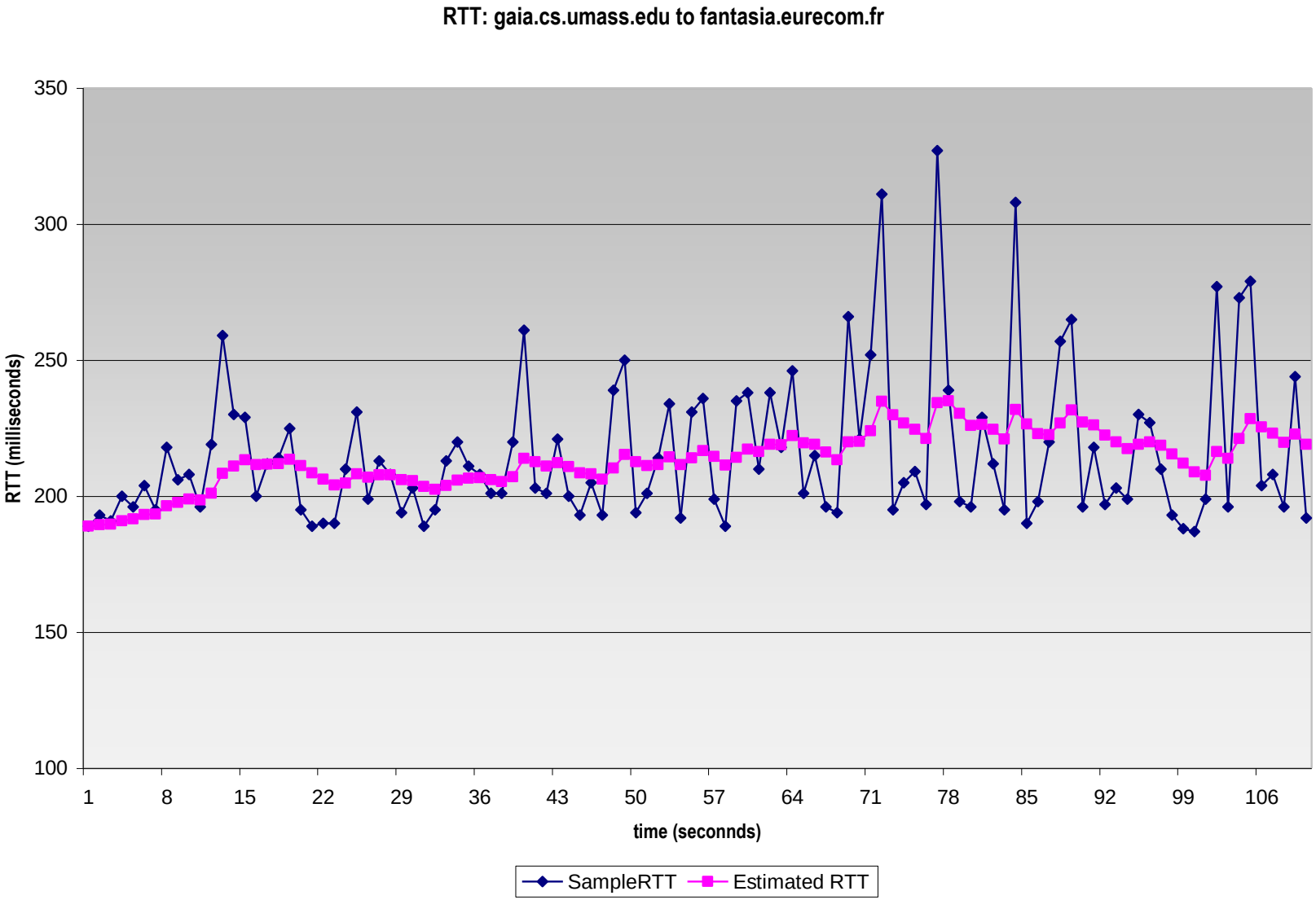
Q: 如何设置TCP超时值?

- 超时值太小:
  - 产生不必要的重传
- 超时值太大:
  - 丢失恢复的时间太长
- 应大于RTT:
  - RTT是变化的

Q: 如何估计RTT?

- SampleRTT:
  - 测量从发出某个报文段到收到其确认报文段之间经过的时间
- SampleRTT是变化的，更有意义的是平均RTT

Example RTT estimation:



估算平均RTT

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

- 指数加权移动平均
- 典型值: α = 0.125

- 瞬时RTT和平均RTT有很大的偏差
- 应当在EstimtedRTT 加上一个“安全距离”，作为超时值
- 安全距离的大小与RTT的波动幅度有关

设置超时值

- SampleRTT 与 EstimatedRTT的偏差估计:

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

(typically, β = 0.25)

设置超时值:

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

3.5.3 可靠数据传输

TCP 在不可靠的IP服务上建立可靠的数据传输

IP层信道：IP包可能出错、丢失、重排序

TCP采用的数据传输机制：

- 发送端采用流水式发送报文段
- 接收端采用累积确认进行响应
- 发送端采用重传来恢复丢失的报文段

一个高度简化的TCP协议

仅考虑可靠传输机制，且数据仅在一个方向上传输

- 接收方：
  - 仅在正确、按序收到报文段后，更新确认序号；其余情况，重复前一次的确认序号（与GBN类似，使用累积确认）
  - 缓存失序的报文段（与SR类似）
- 发送方：
  - 流水式发送报文段
  - 仅对最早未确认的报文段使用一个重传定时器（与GBN类似）
  - 仅在超时后重发最早未确认的报文段（与SR类似，接收端缓存了失序的报文段）

TCP发送方处理的事件

收到应用数据:

- 创建并发送TCP报文段
- 若当前没有定时器在运行（没有已发送、未确认的报文段），启动定时器

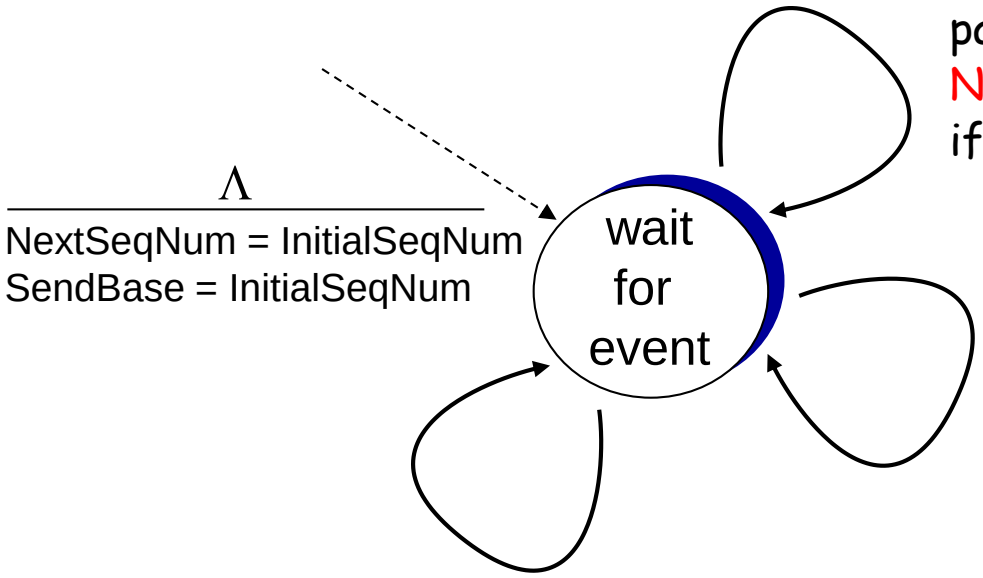
超时:

- 重传包含最小序号的、未确认的报文段
- 重启定时器

收到ACK:

- 如果确认序号大于基序号：
  - 推进发送窗口（更新基序号）
  - 如果还有未确认的报文段，启动定时器，否则终止定时器

TCP sender (simplified)



data received from application above

create segment, seq. #: NextSeqNum
pass segment to IP (i.e., "send")
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
start timer

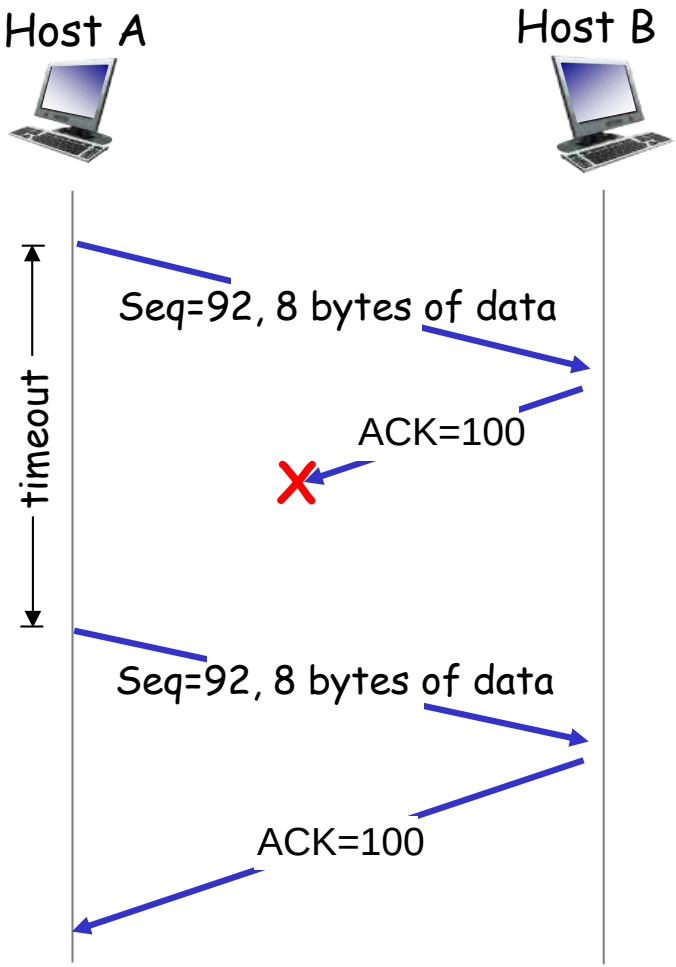
timeout

retransmit not-yet-acked
segment with smallest seq. #
start timer

ACK received, with ACK field value y

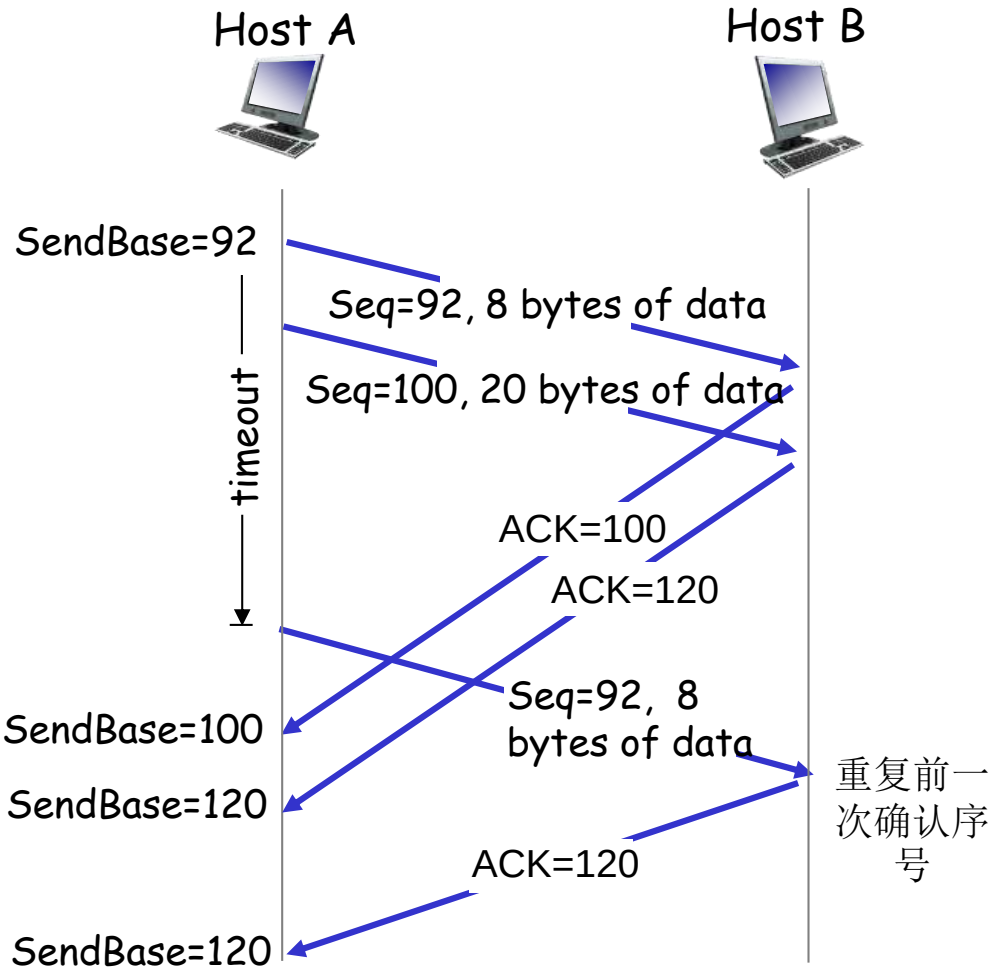
```
if (y > SendBase) {
    SendBase = y
    /* SendBase–1: last cumulatively ACKed byte */
    if (there are currently not-yet-acked segments)
        start timer
      else stop timer
    }
```

TCP: 重传场景



ACK丢失

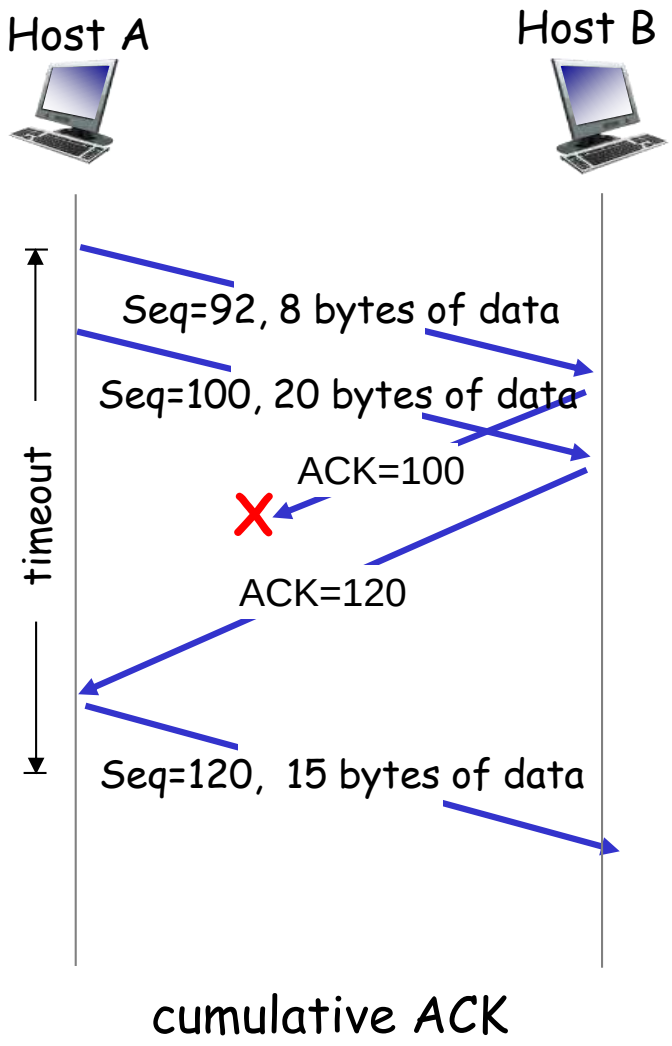
报文段丢失与此类似



过早超时

只重发了第一个报文段

TCP: 重传场景



cumulative ACK

第一个报文段没有重发

思考：

- 对于情形二，如果TCP像SR一样，每个报文段使用一个定时器，会怎么样？
- 对于情形三，采用流水式发送和累积确认，可以避免重发哪些报文段？

TCP减少重传的机制：

- 只使用一个定时器，避免超时设置过小时重发大量报文段
- 利用流水式发送和累积确认，避免重发某些丢失了ACK的报文段

TCP确认的二义性

- 估计RTT的问题：
  - TCP是对接收到的数据而不是对携带数据的报文段进行确认，因此TCP的确认是有二义性的
  - 对重传报文段的RTT估计不准确
- 解决方法：
  - 忽略有二义性的确认，只对一次发送成功的报文段测量SampleRTT，并据此更新**EstimtedRTT**
  - 当TCP重传一个段时，停止测量SampleRTT

定时器补偿

- 简单忽略重传报文段的问题：
  - 重传意味着超时设置可能偏小了，需加大
  - 若简单忽略重传报文段（不估计、不更新RTT），则超时设置也不会更新，超时设置过小的问题没有解决
- 解决方法：
  - 采用定时器补偿策略，发送方每重传一个报文段，超时值就增大一倍
  - 若连续发生超时事件，超时值呈指数增长（至一个设定的上限值）

Karn算法

- Karn算法结合使用RTT估计值和定时器补偿策略确定超时值:
  - 使用EstimatedRTT估计初始的超时值
  - 若发生超时，每次重传时对定时器进行补偿，直到成功传输一个报文段为止
  - 若收到上层应用数据、或某个报文段没有重传就被确认了，用最近的EstimatedRTT估计超时值

TCP的接收端

- 理论上说，接收端只需区分两种情况：
  - 收到期待序号的报文段：发送更新的确认序号
  - 其它情况：重复当前确认序号
- 为减小通信量，TCP允许接收端推迟确认：
  - 接收端可以在收到若干个报文段后，发送一个累积确认的报文段
- 推迟确认的缺点：
  - 若延迟太大，会导致不必要的重传
  - 推迟确认造成RTT估计不准确
- TCP协议规定：
  - 推迟确认的时间最多为500ms
  - 接收方至少每隔一个报文段使用正常方式进行确认

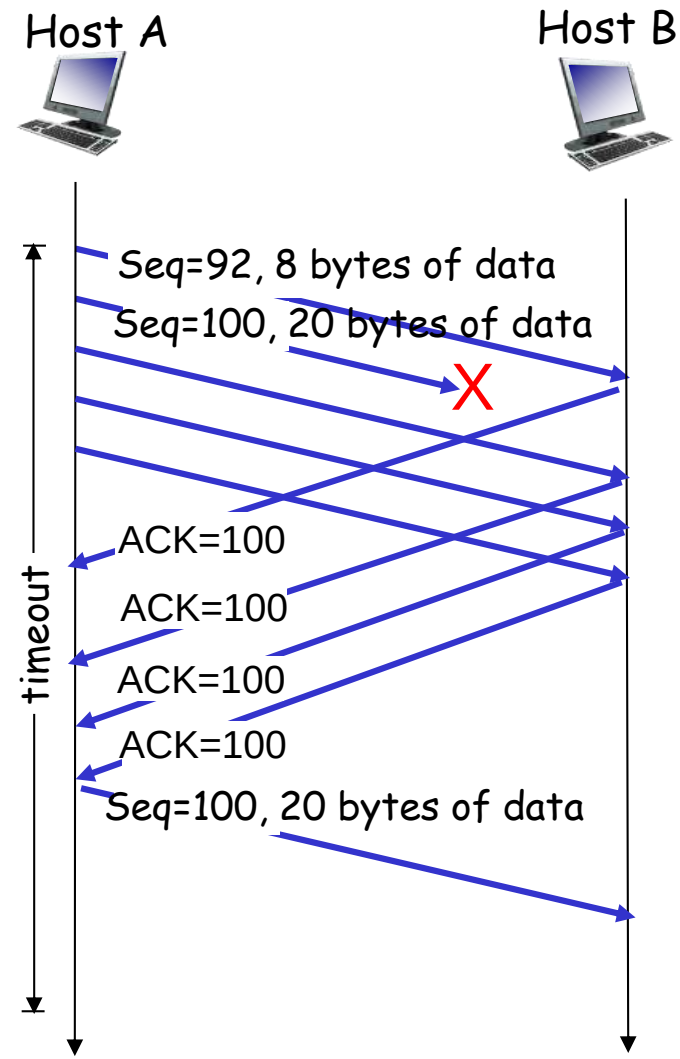
TCP接收端的处理

接收端事件	接收端动作
收到一个期待的报文段，且之前的报文段均已发过确认	推迟发送确认. 在500ms时间内若无下一个报文段到来，发送确认
收到一个期待的报文段，且前一个报文段被推迟确认	立即发送确认（为准确估计RTT）
收到一个失序的报文段（序号大于期待的序号），检测到序号间隙	立即发送确认（快速重传的需要），重复当前的确认序号
收到部分或全部填充间隙的报文段	若报文段始于间隙的低端，立即发送确认（以免发送端超时），更新确认序号

快速重传

- 仅靠超时重发丢失的报文段，恢复太慢！
- 利用重复ACK检测报文段丢失：
  - 发送方通常连续发送许多报文段
  - 若报文段丢失，会有许多重复ACK发生
  - 多数情况下IP按序交付分组，重复ACK极有可能因丢包产生
- 协议规定：当发送方收到对同一序号的3次重复确认时，立即重发包含该序号的报文段
- 快速重传：在定时器到期前重发丢失的报文段

快速重传:举例



fast retransmit after sender receipt of triple duplicate ACK

快速重传算法

```
event: ACK received, with ACK field value of y
            if (y > SendBase) { //收到更新的确认号
                SendBase = y
                if (there are currently not-yet-acknowledged segments)
                    start timer
              }
            else { //收到重复的ACK
                increment count of dup ACKs received for y
                if (count of dup ACKs received for y = 3) {
                    resend segment with sequence number y
                }
```

a duplicate ACK for already ACKed segment

fast retransmit

小结

- 可靠传输的设计：
  - 流水式发送报文段
  - 采用累积确认
  - 只对最早未确认的报文段使用一个重传定时器
  - 超时后只重传包含最小序号的、未确认的报文段

以上措施可大量减少因ACK丢失、定时器过早超时引起的重传

- 超时值的确定：
  - 基于RTT估计超时值+定时器补偿策略
- 测量RTT：
  - 不对重传的报文段测量RTT
  - 不连续使用推迟确认
- 快速重传：
  - 收到3次重复确认，重发报文段

TCP使用GBN还是SR？

Go-Back-N

- 接收方：
  - 使用累积确认
  - 不缓存失序的分组
  - 对失序分组发送重复ACK
- 发送方：
  - 超时后重传从基序号开始的所有分组

TCP

- 接收方：
  - 使用累积确认
  - 缓存失序的报文段
  - 对失序报文段发送重复ACK
- 发送方：
  - 超时后仅重传最早未确认的报文段
  - 增加了快速重传

TCP使用GBN还是SR？

SR

- 接收方：
  - 缓存失序的分组
  - 每个分组使用一个定时器
  - 单独确认每个正确收到的分组
- 发送方：
  - 超时后仅重传未被确认的分组

修改的TCP [RFC2018]

- 接收方：
  - 缓存失序的报文段
  - 只对最早未确认的报文段使用一个定时器
  - SACK选项头中给出收到的非连续数据块的上下边界
- 发送方：
  - 仅重传接收方缺失的数据
  - 增加了快速重传

TCP结合了GBN和SR的优点

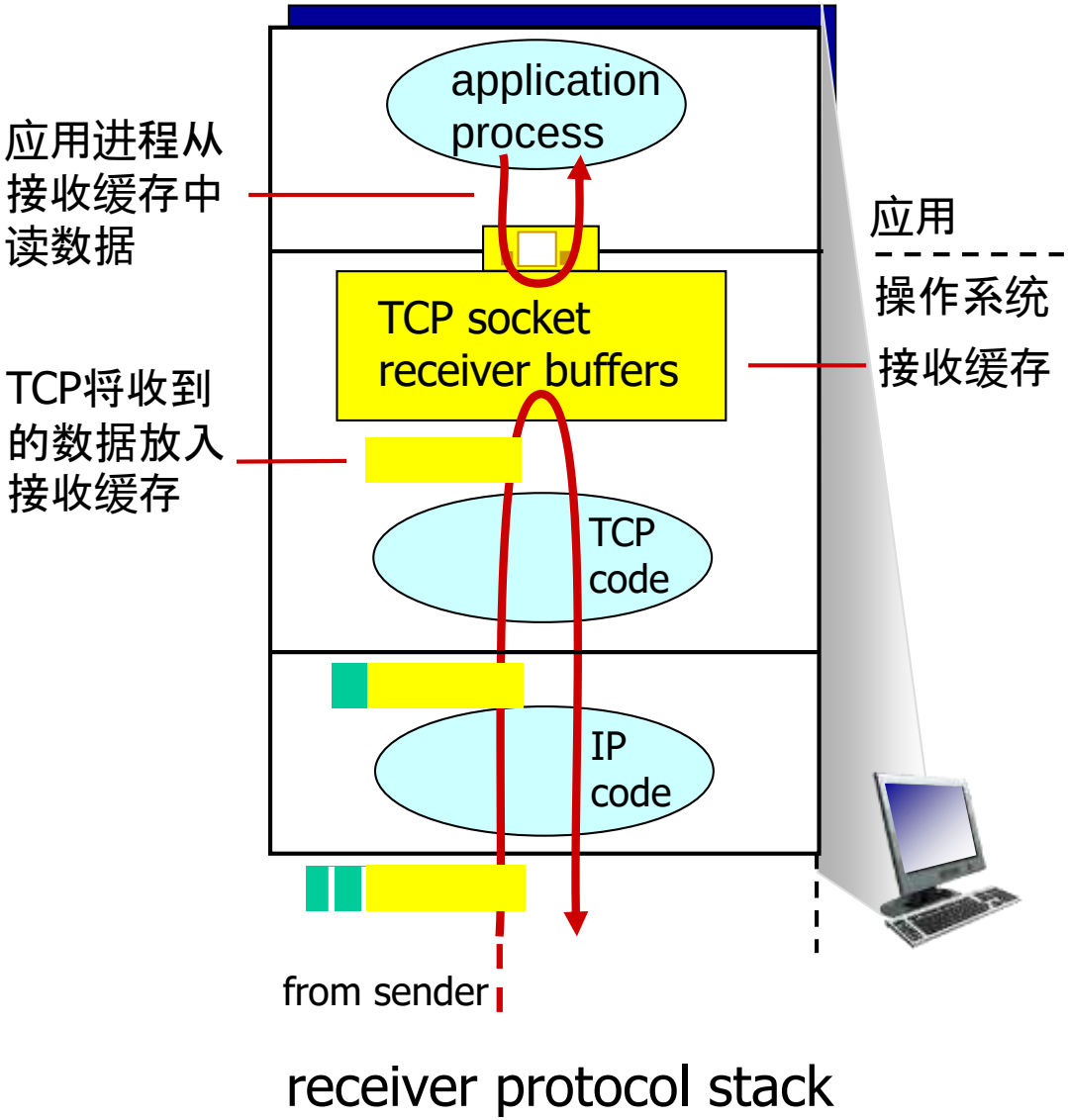
- TCP的可靠传输机制可以看成是GBN和SR的混合体：
  - 定时器的使用：与GBN类似，只对最早未确认的报文段使用一个定时器
  - 超时重传：与SR类似，只重传缺失的数据

- TCP在减小定时器开销和重传开销方面要优于GBN 和 SR！

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

3.5.5 TCP流量控制



- 问题：
  - 进入接收缓存的数据不一定被立即取走、取完
  - 若应用消费数据的速度较慢，接收缓存可能溢出
- 流量控制：
  - 发送端TCP调节发送速率，不使接收端缓存溢出

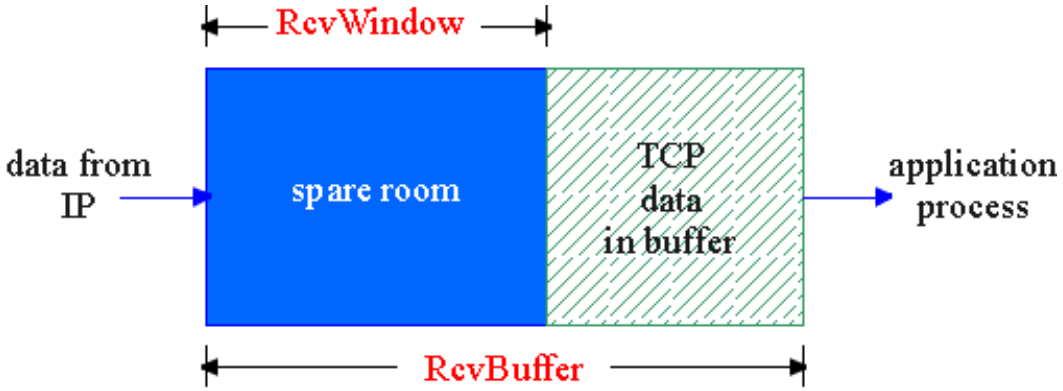
为什么GBN或SR不考虑流量控制？

- GBN和SR均假设：
  - 正确、按序到达的分组被立即交付给上层
  - 其占用的缓冲区被立即释放
- 从而，GBN和SR不会出现接收端缓存溢出的问题

为什么UDP没有流量控制？

- UDP不保证交付：
  - 接收端UDP将收到的报文载荷放入接收缓存
  - 应用进程每次从接收缓存中读取一个完整的报文载荷
  - 当应用进程消费数据不够快时，接收缓存溢出，报文数据丢失，并不违反UDP的服务承诺

TCP Flow control: how it works



- 接收缓存中的可用空间称为接收窗口（RcvWindow）

  RcvWindow = RcvBuffer-[LastByteRcvd - LastByteRead]

- 接收方将RcvWindow放在报头中，向发送方通告接收缓存的可用空间

- 发送方限制未确认的字节数不超过接收窗口的大小，即：

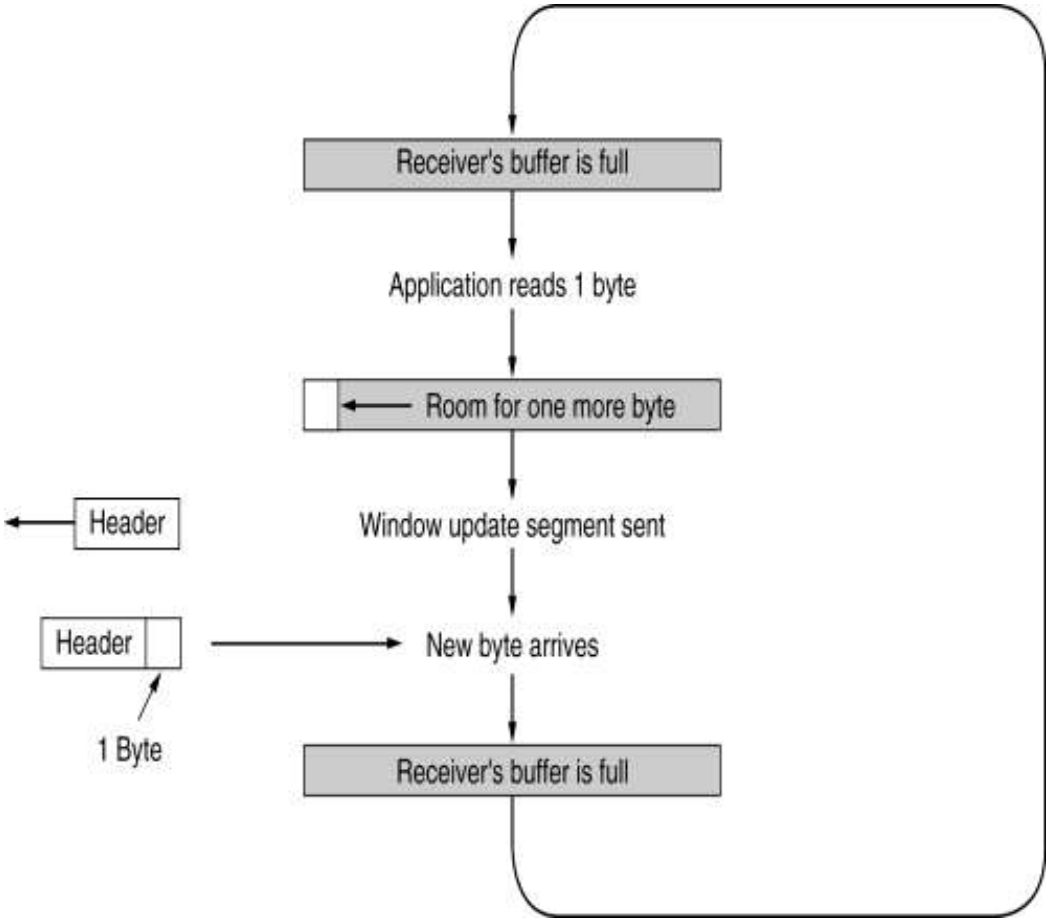
  LastByteSent-LastByteAcked ≦ RcvWindow

零窗口通告和零窗口探测

- 特别是，发送端收到RcvWindow = 0的报文段（零窗口通告）时，必须停止发送，然后：
  - 发送端启动一个定时器
  - 定时器超时后，发送端发送一个零窗口探测报文段（序号为上一个段中最后一个字节的序号）
  - 接收端在响应的报文段中通告当前的接收窗口
  - 若发送端仍收到零窗口通告，重新启动定时器

- 说明：零窗口探测用于触发接收端发送一个响应

糊涂窗口综合症（silly window syndrome）



- 当数据发送很快、而消费很慢时，零窗口探测的简单实现带来以下问题：
  - 接收方不断发送微小窗口通告
  - 发送方不断发送很小的数据分组
  - 大量带宽被浪费

接收方启发式策略

- 接收端避免糊涂窗口综合症的策略：
  - 通告零窗口之后，仅当窗口大小显著增加之后才发送更新的窗口通告
  - 什么是显著增加：窗口大小达到缓存空间的一半或者一个MSS，取两者的较小值
- TCP执行该策略的做法：
  - 当窗口大小不满足以上策略时，推迟发送确认（但最多推迟500ms，且至少每隔一个报文段使用正常方式进行确认），寄希望于推迟间隔内有更多数据被消费
  - 仅当窗口大小满足以上策略时，再通告新的窗口大小

发送方启发式策略

- 发送方避免糊涂窗口综合症的策略：
  - 发送方应积聚足够多的数据再发送，以防止发送太短的报文段
- 问题：发送方应等待多长时间？
  - 如等待时间不够，报文段会太短
  - 如等待时间过久，应用程序的时延会太长
  - 更重要的是，TCP不知道应用程序会不会在最近的将来生成更多的数据

Nagle算法

- Nagle算法很好地解决了该问题：
  - 在新建连接上，当应用数据到来时，组成一个TCP段发送（那怕只有一个字节）
  - 在收到确认之前，后续到来的数据放在发送缓存中
  - 当数据量达到一个MSS或上一次传输的确认到来（取两者的较小时间），将缓存中的数据发走
- Nagle算法的优点：
  - 适应网络延时、MSS长度及应用速度的各种组合
  - 常规情况下不会降低网络的吞吐量

TCP流量控制的总结

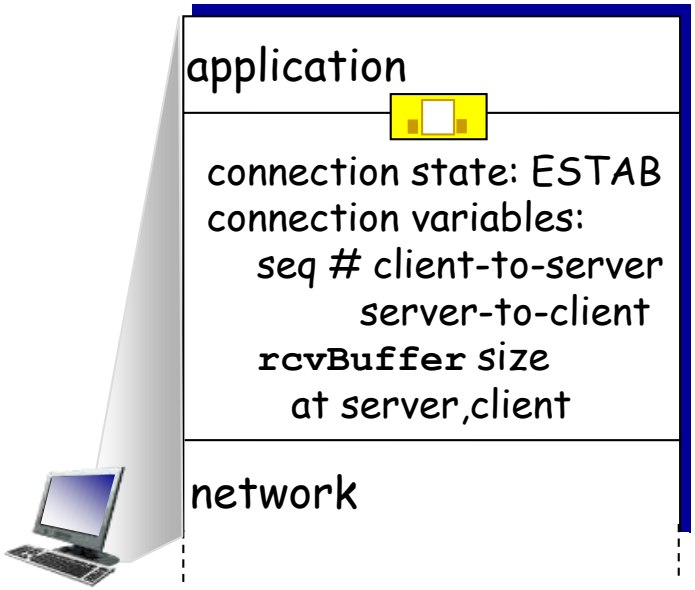
- TCP接收端：
  - 使用显式的窗口通告告知发送方可用的缓存空间大小
  - 在接收窗口较小时推迟发送确认（条件允许的话）
  - 在零窗口通告后，仅当接收窗口显著增加时通告新的窗口大小
- TCP发送端:
  - 使用Nagle算法确定发送时机
  - 使用接收窗口限制发送的数据量（已发送未确认的字节数不超过接收窗口的大小）

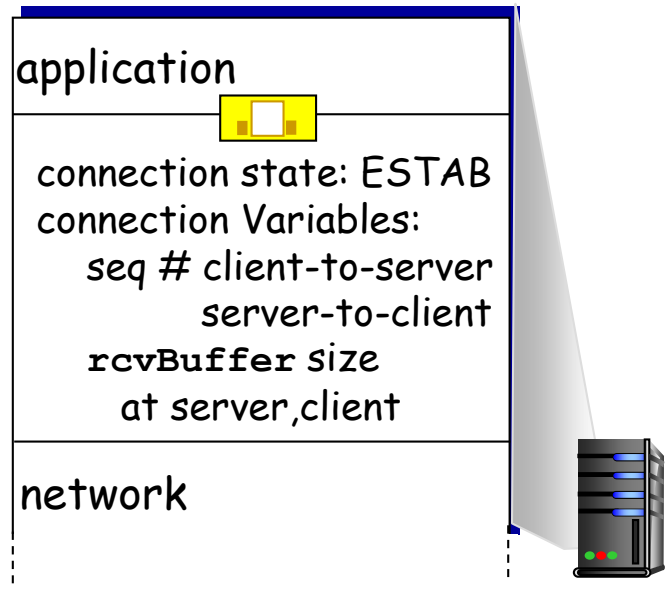
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

3.5.6 连接管理

- 建立连接要确定两件事：
  - 双方都同意建立连接（知晓另一方想建立连接）
  - 初始化连接参数（序号，MSS等）



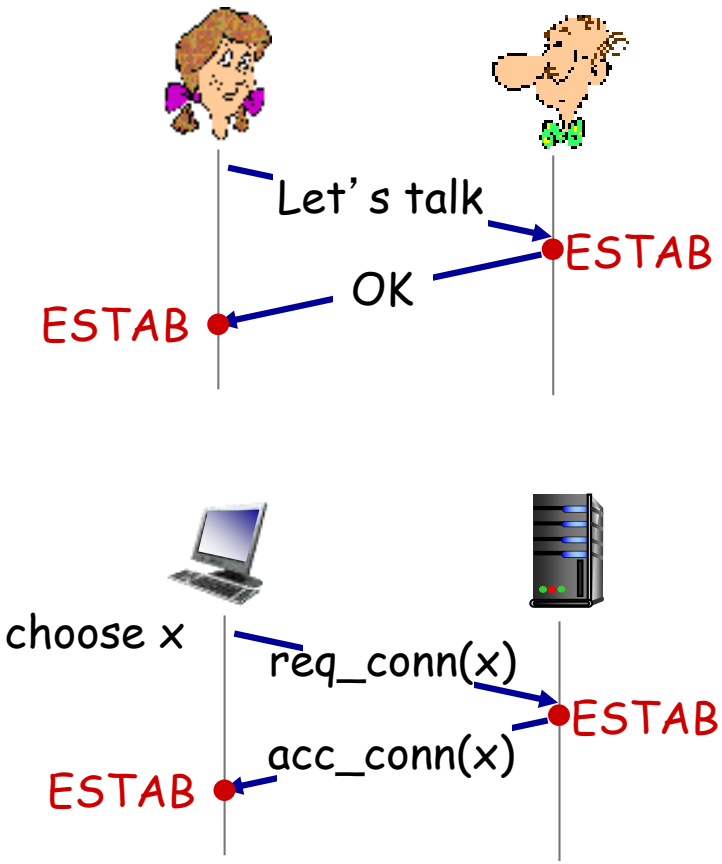



```
Socket clientSocket =
  newSocket("hostname","port
  number");
```

```
Socket connectionSocket =
  welcomeSocket.accept();
```

Agreeing to establish a connection

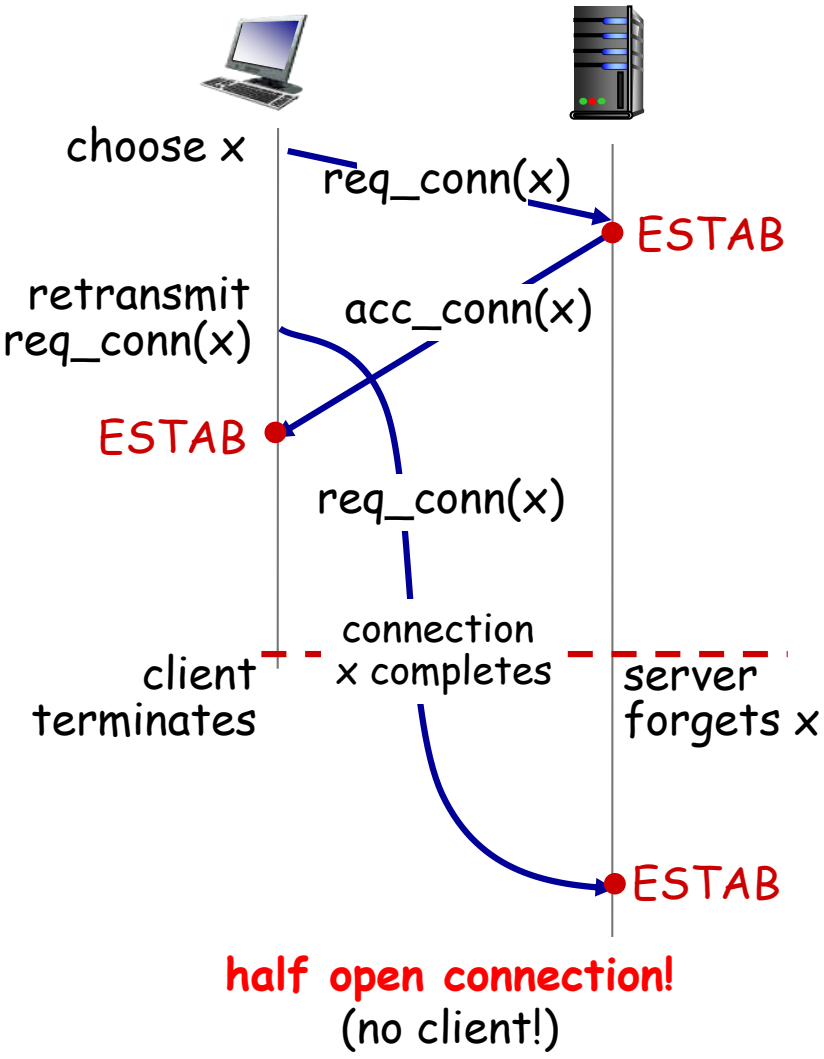
2-way handshake:

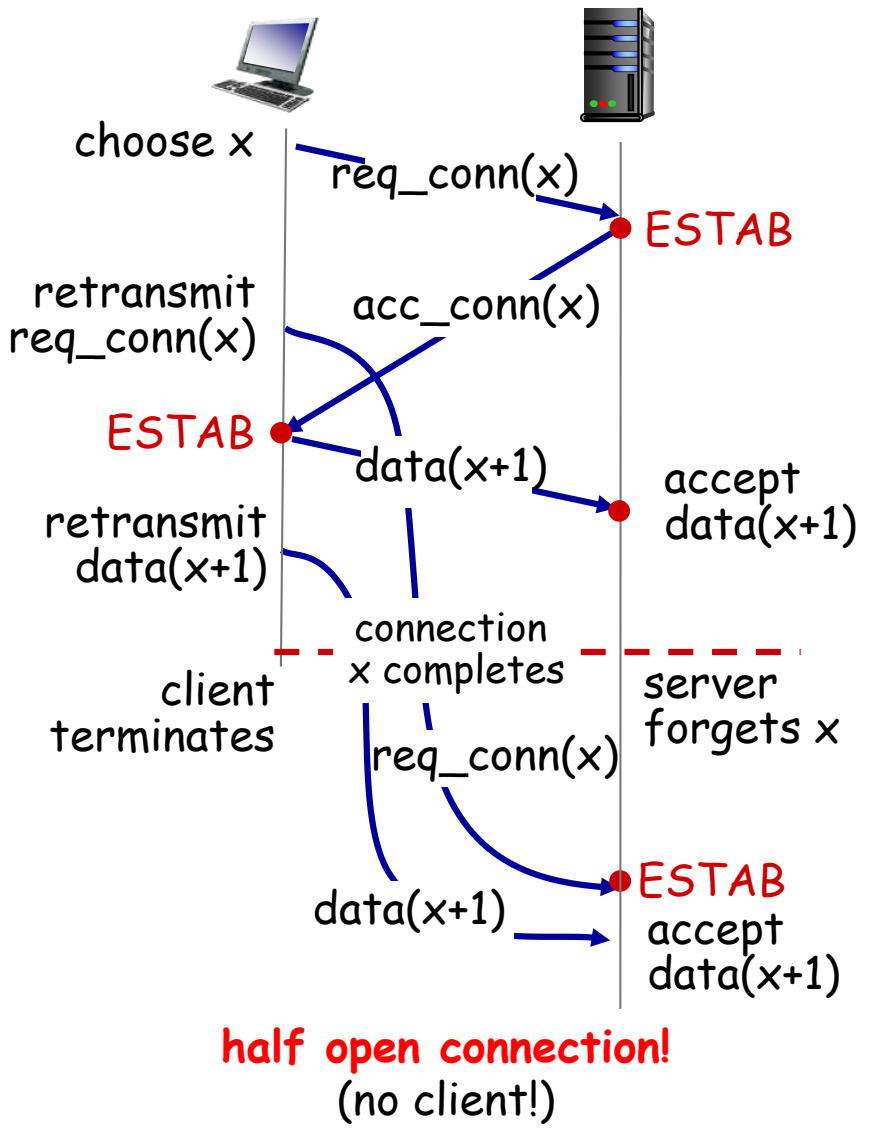


问题：在网络中2次握手总是可行的吗？

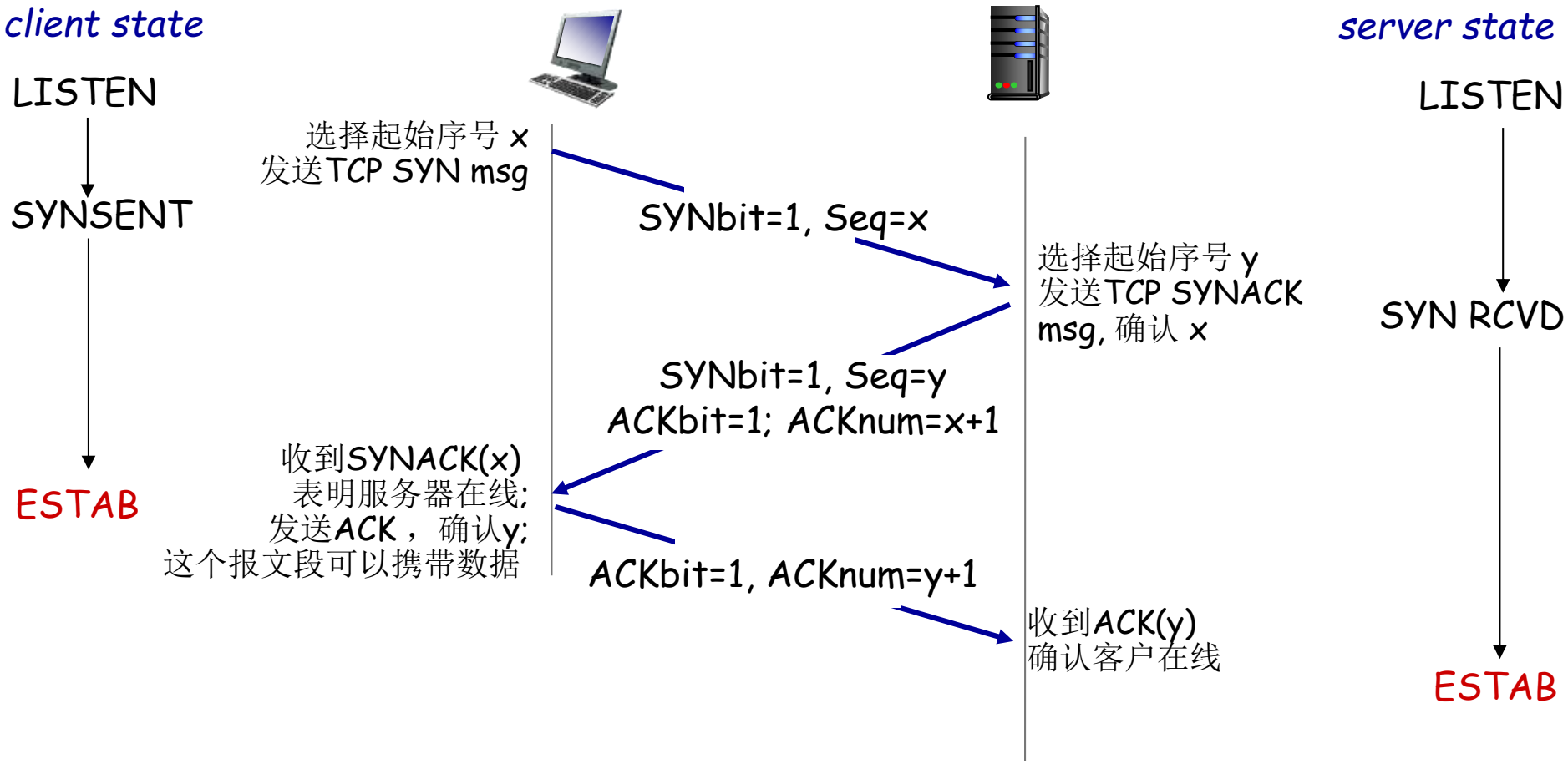
- 在一个不可靠的网络中，会有一些意外发生，比如：
  - 包传输延迟变化很大
  - 存在重传的报文段
  - 存在报文重排序
- 除了收到的包，看不到对方的情况……

2次握手失败的情形


choose x
req_conn(x)
ESTAB
retransmit
req_conn(x)
acc_conn(x)
ESTAB
req_conn(x)
connection
x completes
client
terminates
server
forgets x
ESTAB
half open connection!
(no client!)


choose x
req_conn(x)
ESTAB
retransmit
req_conn(x)
acc_conn(x)
ESTAB
data(x+1)
accept
data(x+1)
retransmit
data(x+1)
connection
x completes
client
terminates
server
forgets x
req_conn(x)
ESTAB
data(x+1)
accept
data(x+1)
half open connection!
(no client!)

TCP三次握手


client state
LISTEN
SYNSENT
ESTAB
选择起始序号 x
发送TCP SYN msg
SYNbit=1, Seq=x
选择起始序号 y
发送TCP SYNACK
msg, 确认 x
SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1
收到SYNACK(x)
表明服务器在线;
发送ACK，确认y;
这个报文段可以携带数据
ACKbit=1, ACKnum=y+1
收到ACK(y)
确认客户在线
server state
LISTEN
SYN RCVD
ESTAB

TCP三次握手建立连接

Step 1: 客户TCP发送SYN 报文段（SYN=1, ACK=0）

- Seq = 客户的起始序号
- Ack：无有效内容
- 不包含数据

Step 2: 服务器TCP发送SYNACK报文段（SYN=ACK=1）

- Seq = 服务器的起始序号
- Ack = 客户起始序号+1
- 不包含数据（服务器端分配缓存和变量）

Step 3: 客户发送ACK报文段（SYN=0，ACK=1）

- Seq =客户起始序号+1
- Ack = 服务器起始序号+1
- 可以包含数据（客户端分配缓存和变量）

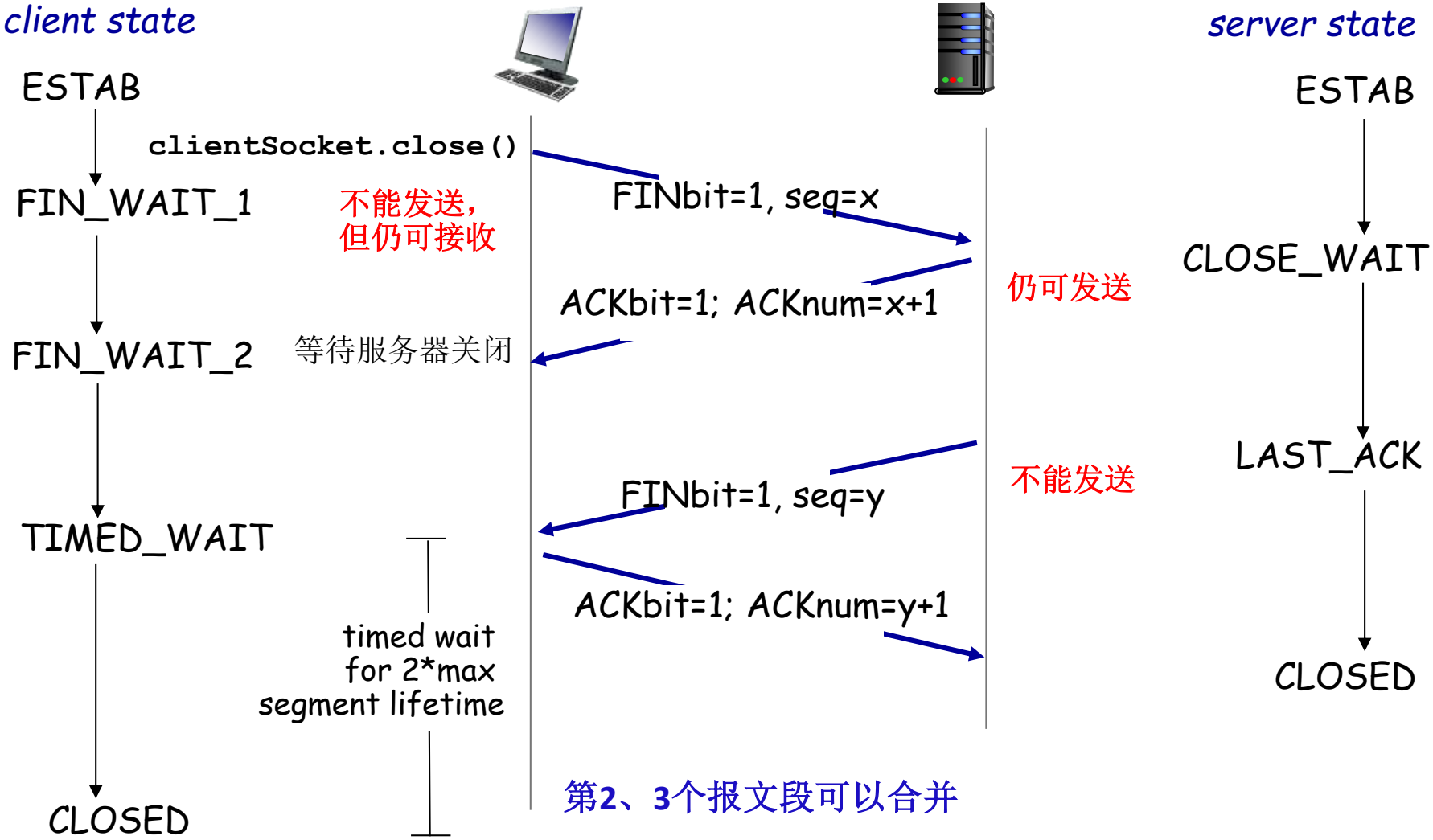
如何选择起始序号？

- 为什么起始序号不从1开始？
  - 若每个新建连接都从序号1开始，那么在不同时间、同一对套接字之间建立的连接，它们的握手报文段都一样，且旧连接上的报文段可能会干扰新连接上的传输（报文段序号有重叠）
- 可以随机选取起始序号吗？
  - 新、旧连接上报文段序号重叠的可能性将大为减小，但不能完全避免
- **必须避免新、旧连接上的序号产生重叠**

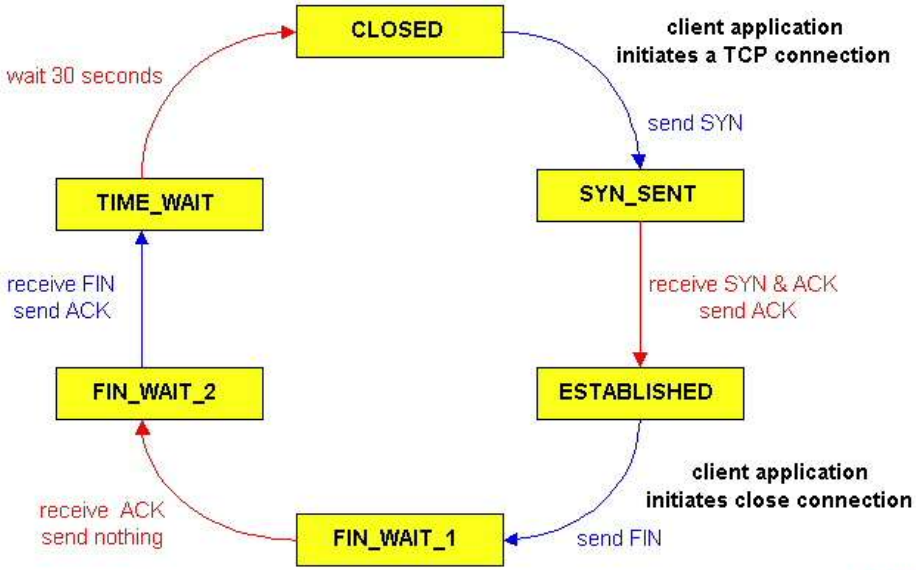
TCP起始序号的选取

- 基于时钟的起始序号选取算法：
  - 每个主机使用一个时钟，以二进制计数器的形式工作，每隔ΔT时间，计数器加1
  - **新建一个连接时，以计数器值的最低32位作为起始序号**
  - 该方法确保**连接的起始序号随时间单调增长**
- **取较小的ΔT**，确保起始序号的增长速度超过TCP连接上序号的增长速度
- **使用较长的序号（32位）**，确保序号回绕的时间远大于分组在网络中的最长寿命

关闭TCP连接

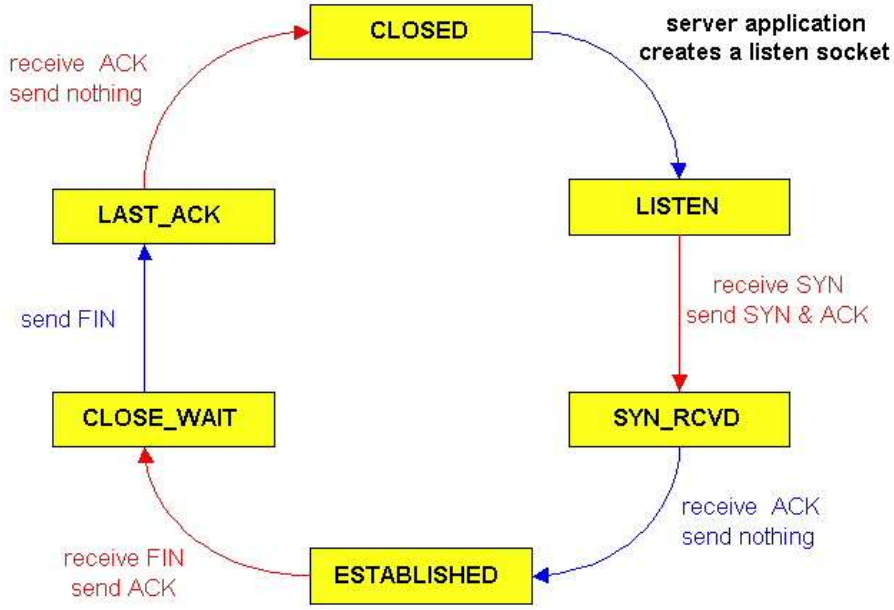

client state
ESTAB
clientSocket.close()
FIN_WAIT_1
不能发送，
但仍可接收
FINbit=1, seq=x
ACKbit=1; ACKnum=x+1
仍可发送
FIN_WAIT_2
等待服务器关闭
FINbit=1, seq=y
不能发送
TIMED_WAIT
ACKbit=1; ACKnum=y+1
timed wait
for 2*max
segment lifetime
CLOSED
第2、3个报文段可以合并
server state
ESTAB
CLOSE_WAIT
LAST_ACK
CLOSED

客户/服务器经历的TCP状态序列



TCP client lifecycle

TCP server lifecycle



SYN洪泛攻击（SYN flood attack）

- TCP实现的问题：
  - step 2：服务器在收到SYN段后，发送SYNACK段，分配资源
  - 若未收到ACK段，服务器超时后重发SYNACK段
  - 服务器等待一段时间（称SYN超时）后丢弃未完成的连接，SYN超时的典型值为30秒~120秒
- SYN洪泛攻击：
  - 攻击者采用伪造的源IP地址，向服务器发送大量的SYN段，并且不发送ACK段
  - 服务器为维护一个巨大的半连接表耗尽资源，导致无法处理正常客户的连接请求

端口扫描

- 扫描程序利用与目标机器建立TCP连接过程中获得的响应消息来收集信息
- 在典型的TCP端口扫描过程中，发送端向目标端口发送SYN报文段，
  - 若收到SYNACK段，表明该目标端口上有服务在运行
  - 若收到RST段，表明该目标端口上没有服务在运行
  - 若什么也没有收到，表明路径上有防火墙（有些防火墙会丢弃来自外网的SYN报文段）

FIN扫描

- FIN扫描过程如下：
  - 发送端向目标端口发送FIN报文段
  - 若收到ACK=1、RST=1的TCP段，表明目标端口上没有服务在监听；若没有响应，表明有服务在监听（RFC 973的规定）
- 缺点：
  - 有些系统的实现不符合RFC 973规定，如在Microsoft的TCP实现中，总是返回ACK=1、RST=1的TCP段

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

拥塞控制原理

交通拥堵（类比的例子）：

- 起因：大量汽车短时间内进入路网，超出路网的承载能力
- 表现：道路通行能力下降，车速变慢，甚至完全停滞
- 措施：减少车辆进入（交通管制）

网络拥塞：

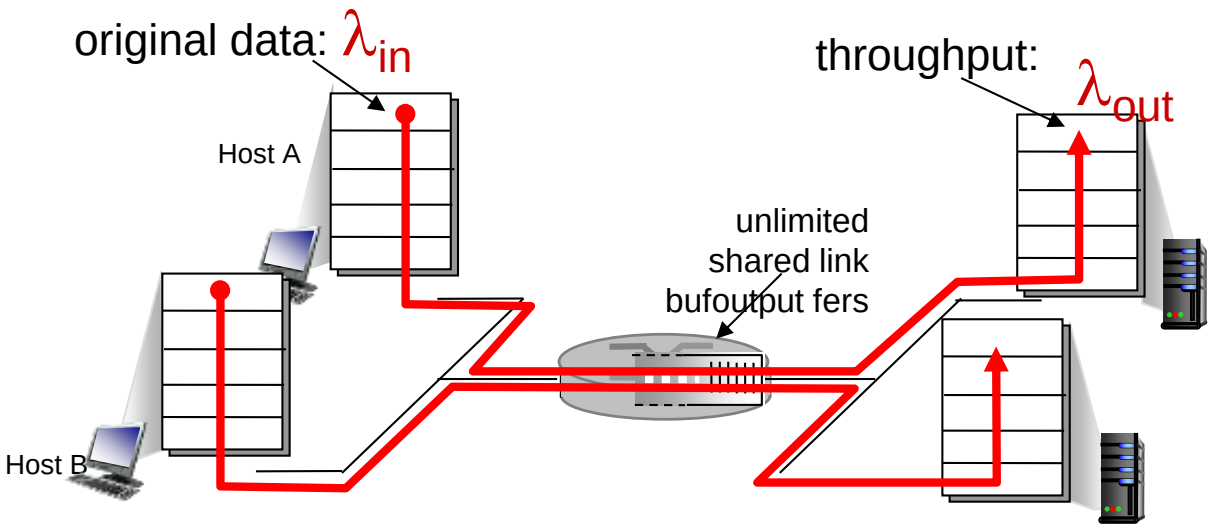
- 起因：大量分组短时间内进入网络，超出网络的处理能力
- 表现：网络吞吐量下降，分组延迟增大
- 措施：减少分组进入网络（拥塞控制）

流量控制与拥塞控制：

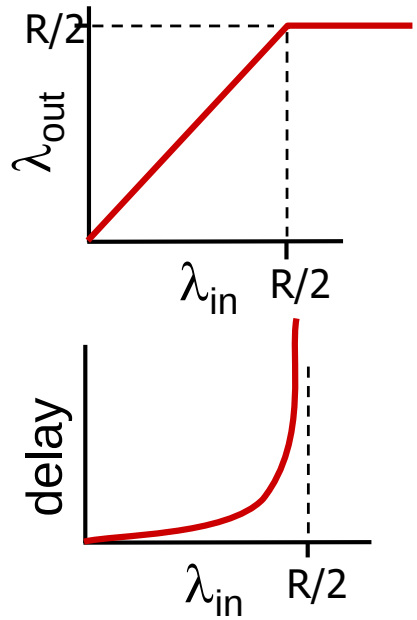
- 流量控制：限制发送速度，使不超过接收端的处理能力
- 拥塞控制：限制发送速度，使不超过网络的处理能力

网络拥塞的代价之一：延迟增大

- 拥塞发生在路由器中：
  - 对于路由器的某条输出链路而言，当进入该链路的数据速率接近或超出链路带宽时，就会发生拥塞
- **代价一：分组延迟增大**
  - 即使不考虑丢包（假设有无限大的缓存），当链路接近满载时，排队延迟急剧增大

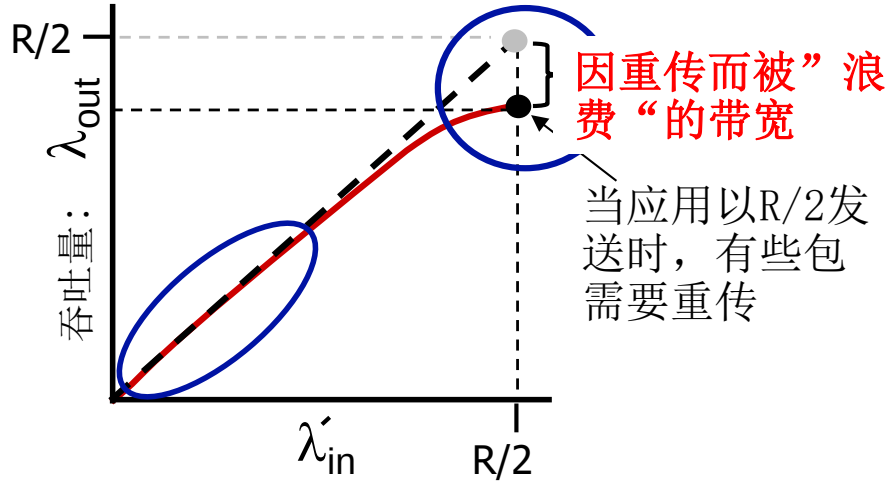


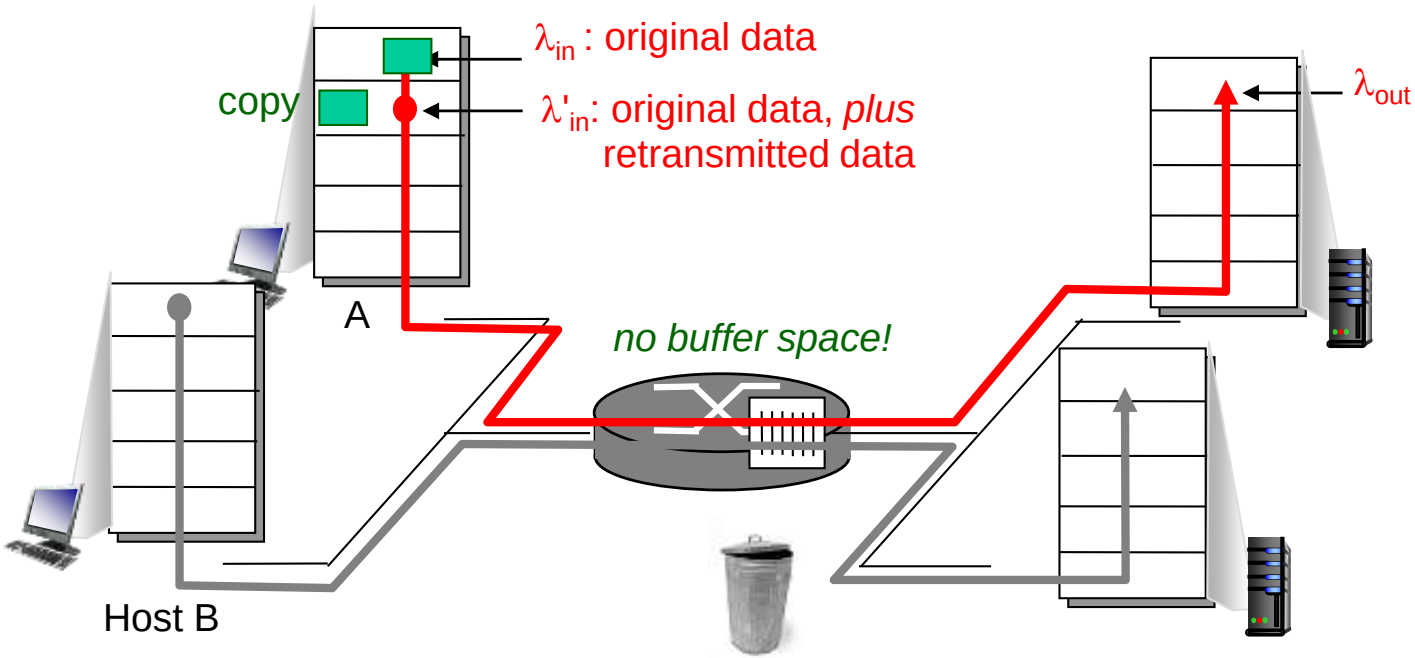
输出链路带宽为R，两条流各使用一半的带宽



网络拥塞的代价之二：吞吐量下降

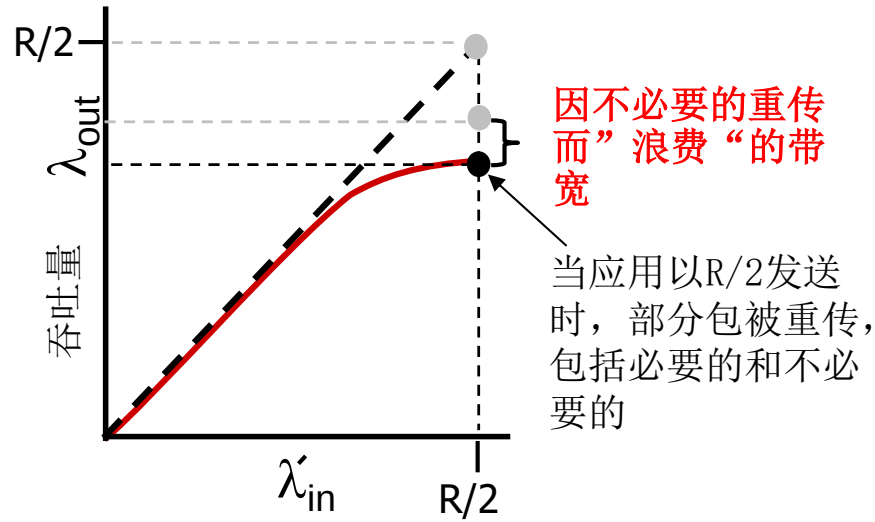
- 吞吐量下降，场景一：
  - 路由器输出缓存溢出，发送端TCP重传被丢弃的包
  - 应用层$\lambda_{in} = \lambda_{out}$
  - 传输层$\lambda'_{in} > \lambda_{in}$

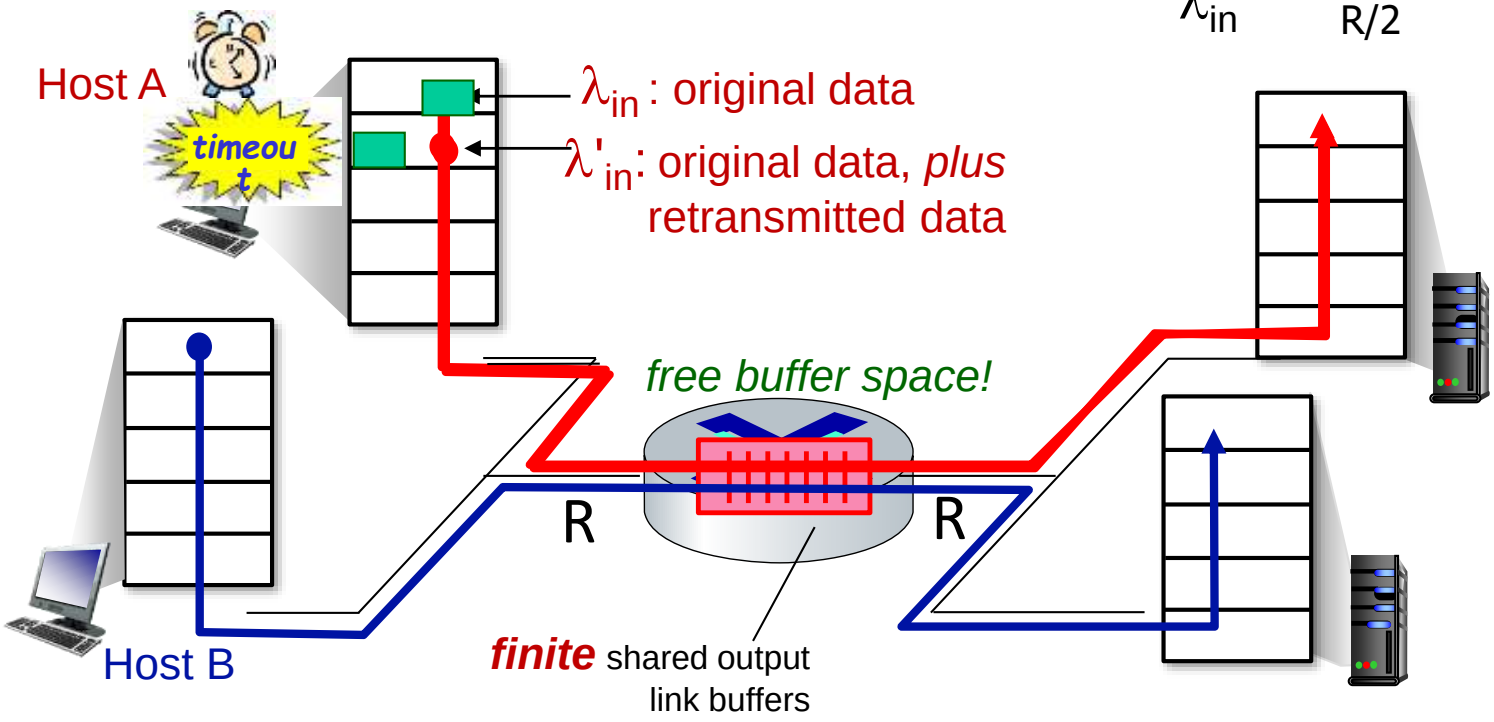




网络拥塞的代价之二：吞吐量下降

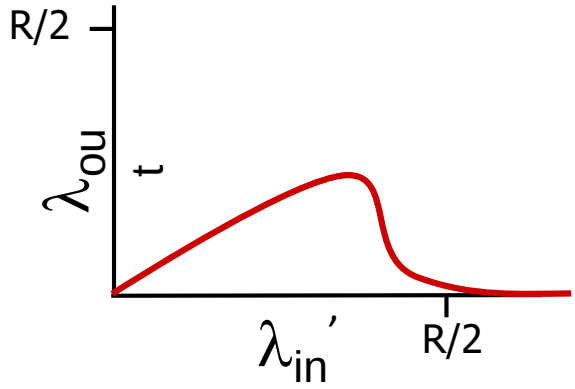
- 吞吐量下降，场景二：
  - 发送端定时器过早超时，产生了不必要的重传
  - 应用层$\lambda_{in} = \lambda_{out}$
  - 传输层$\lambda'_{in} > \lambda_{in}$

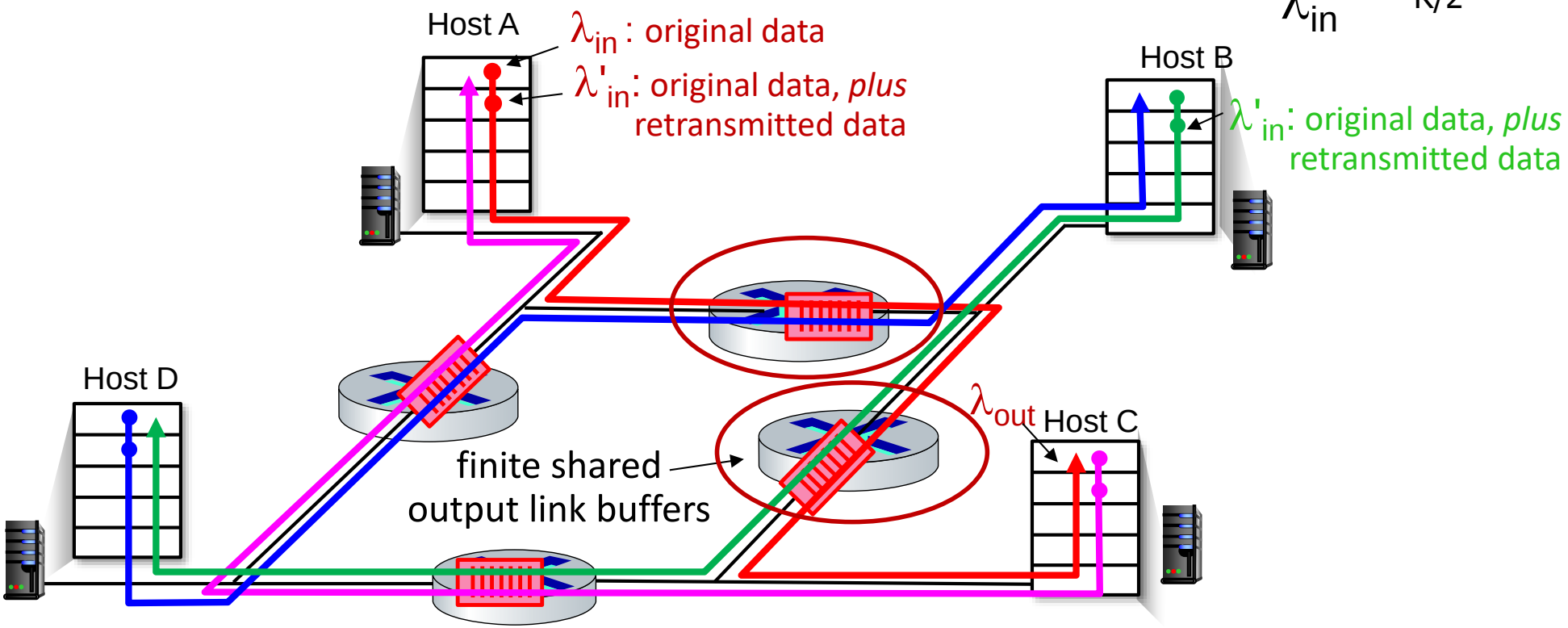




网络拥塞的代价之二：吞吐量下降

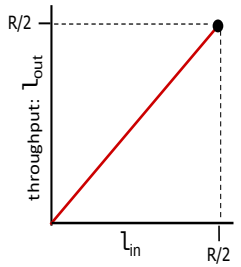
- 吞吐量下降，场景三：
  - 当B->D（绿色）的 λ'_{in} 不断增大时，到达右侧路由器的红包被丢弃，A->C的吞吐量降至0
  - **当一个包被丢弃时，上游用于传输该包的带宽和缓存空间都被浪费了！**

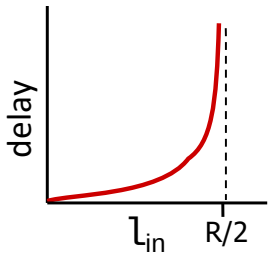


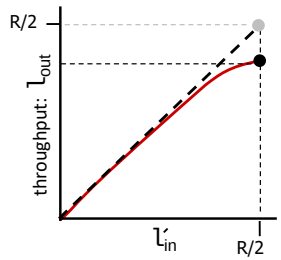


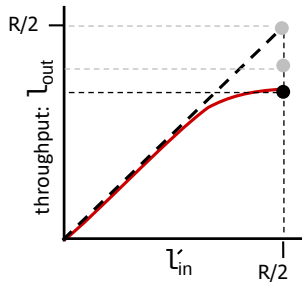
Causes/costs of congestion: insights

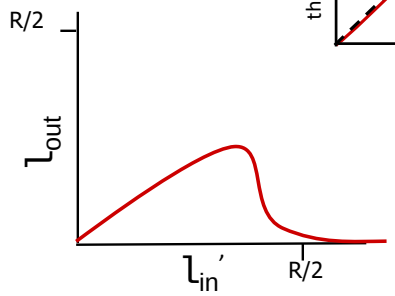
- 吞吐量不可能超过链路容量
- 负载接近链路容量时延迟增大
- 丢包重传导致有效吞吐量下降
- 不必要的重传进一步降低了有效吞吐量
- 下游被丢弃的包浪费了上游的传输容量和缓存空间







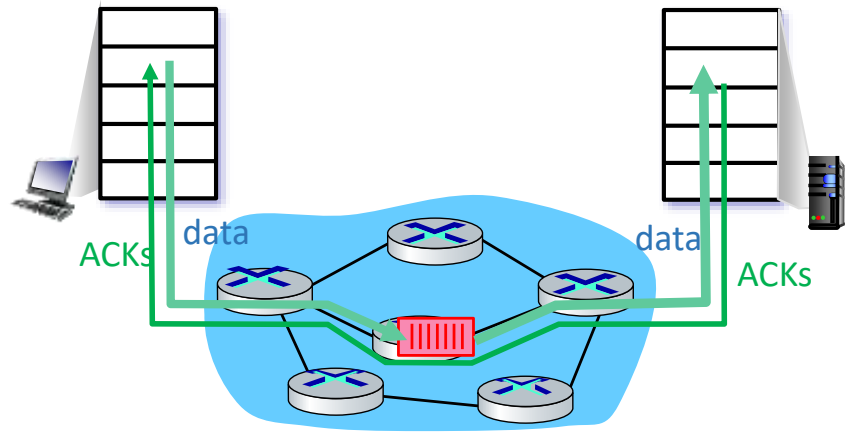




拥塞控制方法

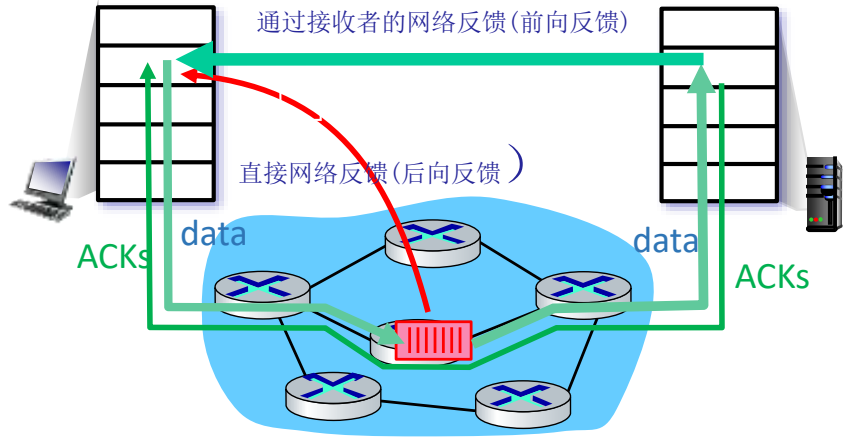
端到端拥塞控制：

- 网络不向端系统提供显式反馈
- 端系统通过观察丢包和延迟来推断拥塞的发生，进而降低发送速率
- TCP采用此类方法



网络辅助的拥塞控制：

- 发生拥塞的路由器向相关的发送端/接收端提供直接反馈，指示拥塞程度或直接给出发送速率
- 发送端相应降低发送速率
- TCP ECN, ATM, DECbit协议采用此方法



Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP拥塞控制

- TCP使用端到端拥塞控制机制：
  - 发送方根据自己感知的网络拥塞程度，限制其发送速率
- 三个问题：
  - 发送方如何感知网络拥塞？
  - 发送方采用什么机制来限制发送速率？
  - 发送方感知到网络拥塞后，采取什么速率调节策略？

TCP拥塞控制

1. 发送方如何感知拥塞？

- 利用丢包事件感知拥塞
  - 丢包或分组延迟过大，对于发送端来说都是丢包了
- 丢包事件包括：
  - 超时
  - 3次重复的ACK

3. 发送方调节拥塞窗口的策略

- 加性增、乘性减
- 慢启动

2. 发送方使用拥塞窗口CongWin限制已发送未确认的数据量：

$$\text{LastByteSent} - \text{LastByteAcked} \le \text{CongWin}$$

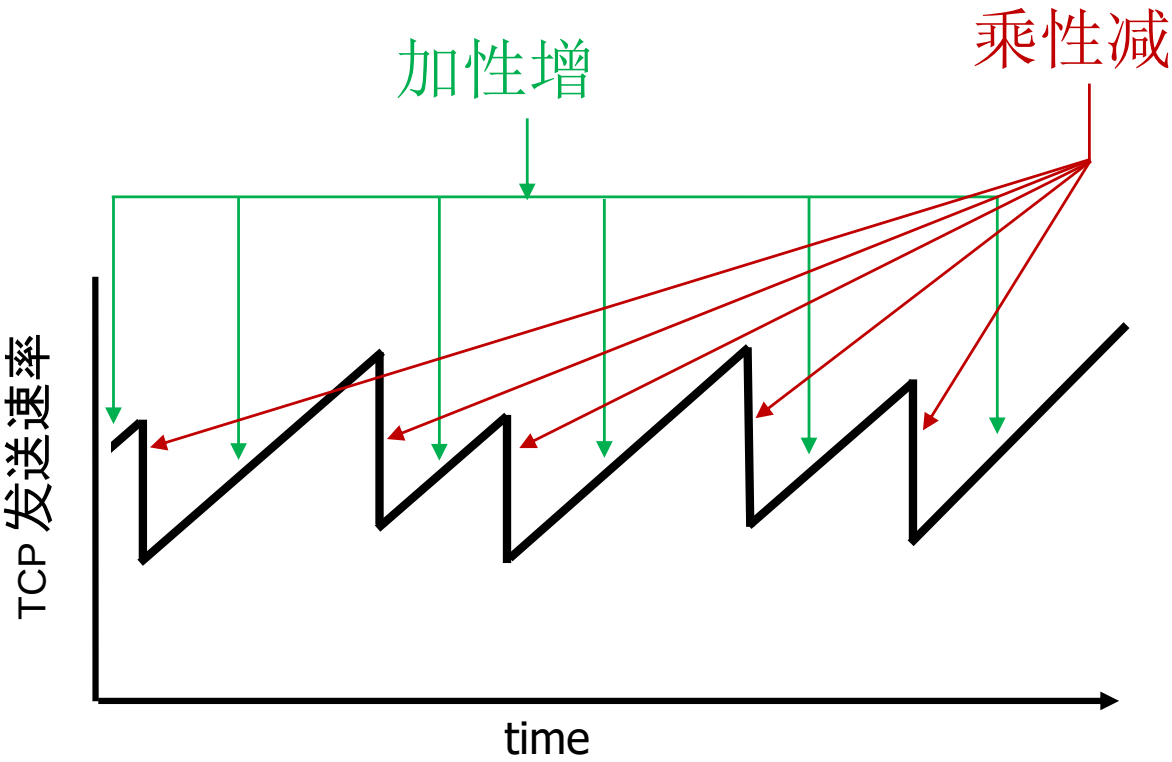
- Roughly,

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

- CongWin随所感知的网络拥塞程度而变化

加性增、乘性减（AIMD）

- **乘性减：** 每检测到一个丢包事件，CongWin减半（迅速减小），但不能小于一个MSS
- **加性增：** 若没有丢包，每经过一个RTT，CongWin增大一个MSS（缓慢增大），直到检测到丢包

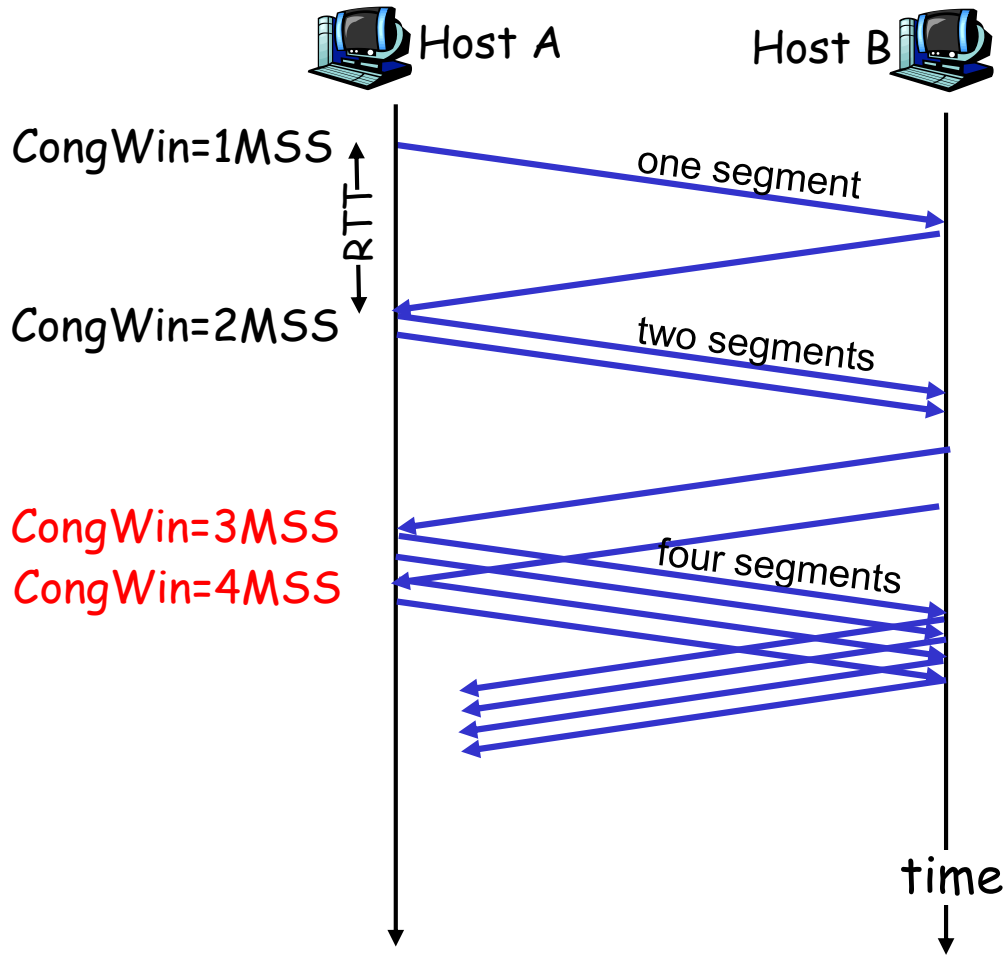


TCP慢启动

- 连接刚建立时
  - CongWin = 1 MSS
- 按照加性增来增大CongWin：
  - 每隔一个RTT才能增大一个MSS，速度太慢！
- 希望迅速增大CongWin至可用的发送速度
- 基本思想：
  - 在新建连接上指数增大CongWin，直至检测到丢包（此时终止慢启动过程）

慢启动的实施

- 慢启动的策略：
  - 每经过一个RTT，CongWin加倍
- 具体实施方法：
  - 每收到一个ACK段，CongWin增加一个MSS
- 特点：
  - 以一个很低的速率开始，按指数增大发送速率

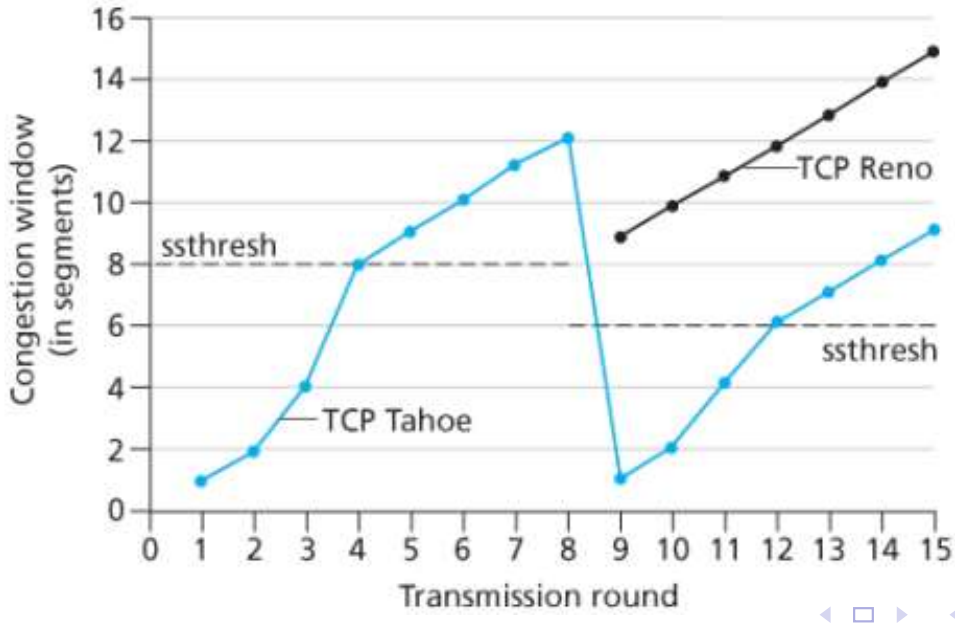


区分不同的丢包事件

- 丢包事件：
  - 收到3个冗余ACK：说明网络仍有一定的交付能力
  - 超时：说明网络交付能力很差
- 目前的TCP实现区分不同的丢包事件
- 收到3个冗余ACK：
  - CongWin降为一半
  - 采用AIMD调节
- 超时：
  - CongWin = 1MSS
  - 使用慢启动增大CongWin，至超时发生时CongWin的一半
  - 使用AIMD调节

实现

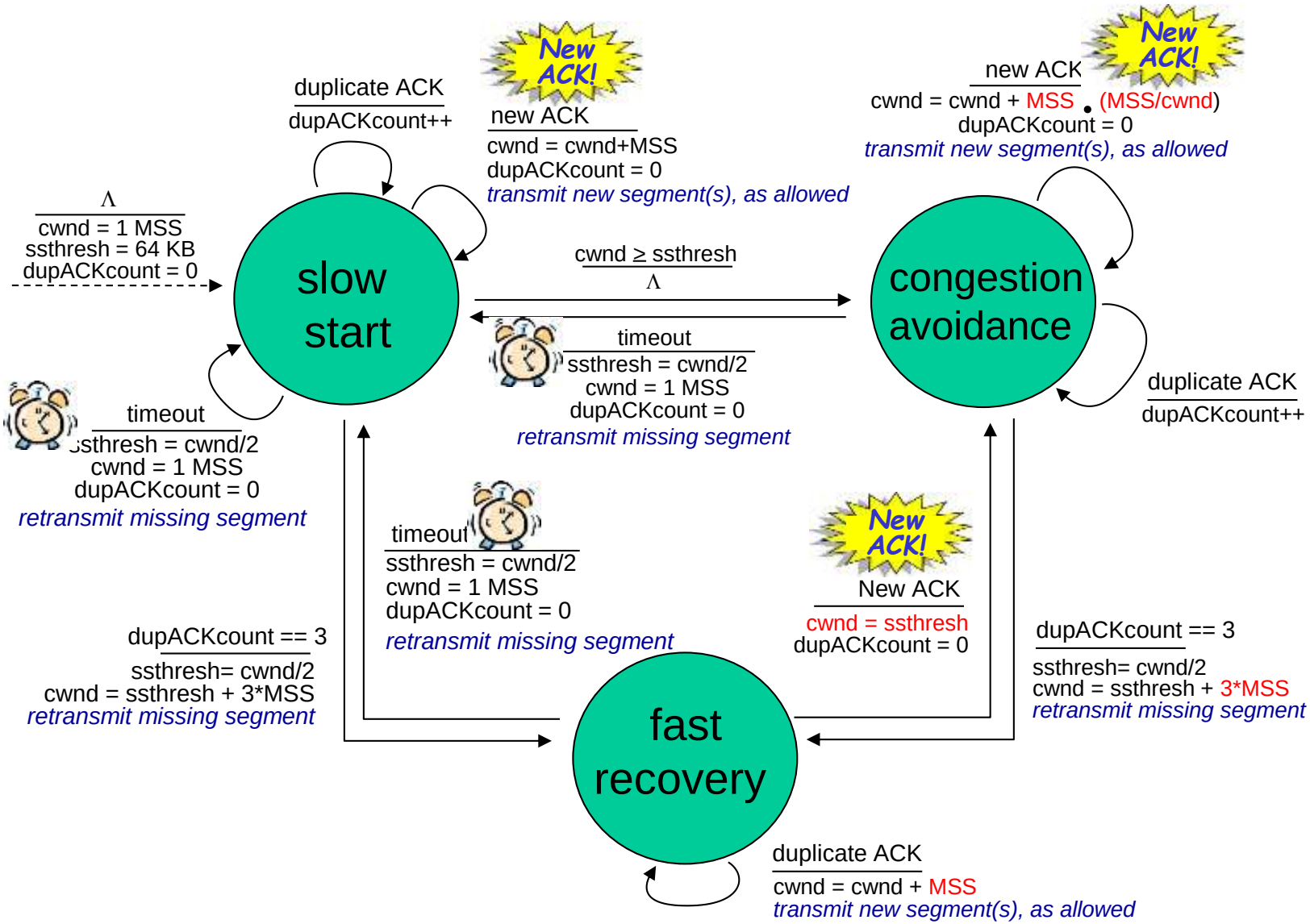
- 发送方维护一个变量Threshold
- 发生丢包时，Threshold设置为当前CongWin的一半：
  - 若收到3次重复的ACK：令CongWin = Threshold+3*MSS，执行AIMD（已有3个报文段到达接收端，不再占用网络资源，扩大CongWin以允许再发送3个报文段）
  - 若发生的是超时：CongWin = 1*MSS，执行慢启动
- Threshold是从慢启动转为拥塞避免的分水岭：
  - CongWin低于门限：执行慢启动
  - CongWin大于等于门限：执行AIMD（也称拥塞避免）



小结

- 当**CongWin < Threshold**时，发送方处于慢启动阶段，拥塞窗口指数增长
- 当**CongWin >= Threshold**时，发送方处于拥塞避免阶段， 拥塞窗口线性增长
- 当收到3个冗余*ACK* 时，**Threshold = CongWin/2，CongWin = Threshold + 3MSS**
- 当超时发生时， **Threshold = CongWin/2 ，CongWin =1 MSS**

Summary: TCP congestion control



TCP吞吐量

一个长期存活的TCP连接的平均吞吐量是多少？（忽略慢启动阶段）

- 令W=发生丢包时的CongWin，此时

  throughput = W/RTT
- 发生丢包后调整 CongWin=W/2，此时 throughput=W/2RTT
- 假设在TCP连接的生命期内，RTT 和 W几乎不变，则：Average throughout=0.75 W/RTT

TCP Futures: TCP over "long, fat pipes"

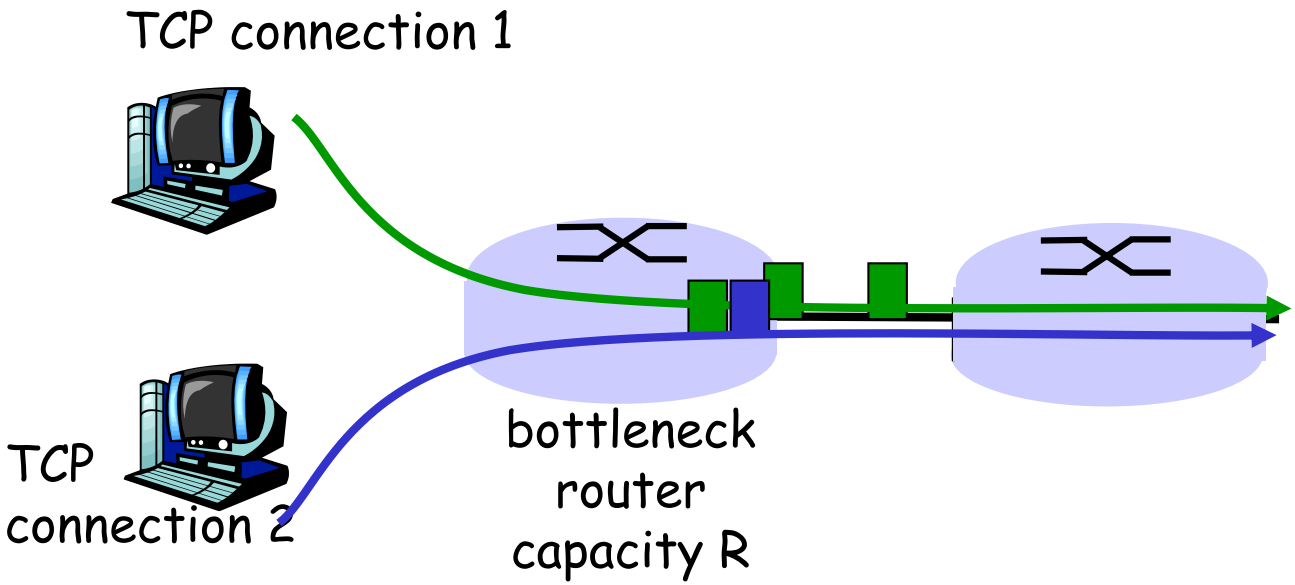
- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- 根据平均吞吐量公式，平均拥塞窗口（0.75W）= 83,333 报文段
- 一条TCP连接的平均吞吐量与丢包率L的关系（课后习题）：

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ L = $2 \cdot 10^{-10}$ *Wow*
- 针对高速网络需要新的TCP拥塞控制算法

TCP的公平性

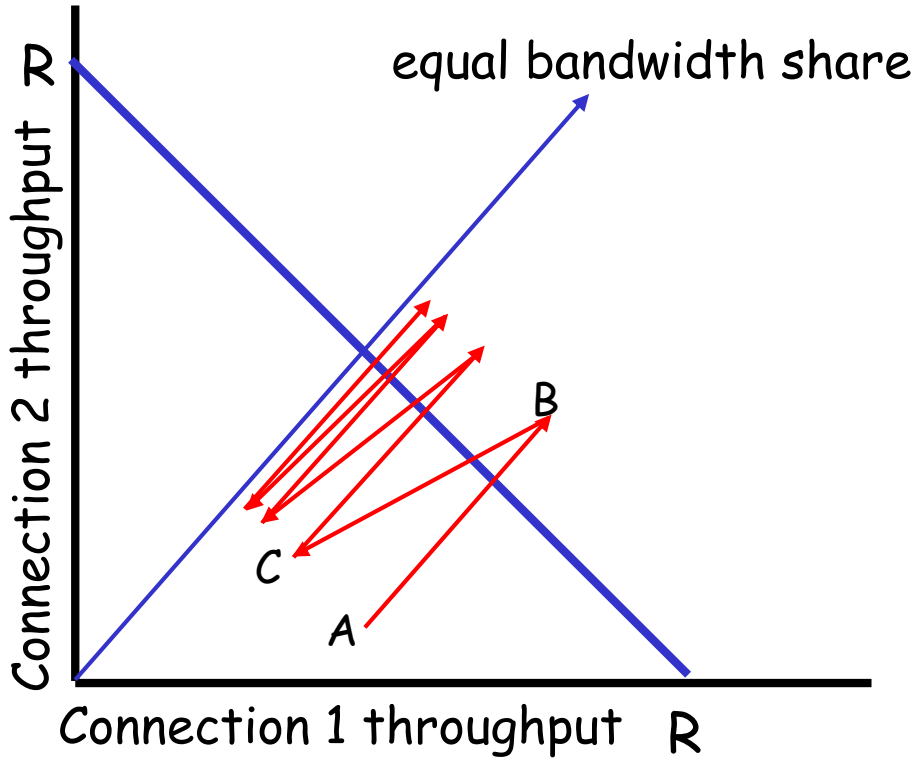
公平性目标：如果K条TCP连接共享某条带宽为R的瓶颈链路，每条连接具有平均速度R/K。



为什么TCP是公平的?

两条竞争的连接（各种参数相同）共享带宽为R的链路:

- addative increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



TCP公平性更复杂的情形

- 若相互竞争的TCP连接具有不同的参数（RTT、MSS等），不能保证公平性；
- 若应用（如web）可以建立多条并行TCP连接，不能保证带宽在应用之间公平分配；比如，一条速率为R的链路上有9条连接：
  - 若新应用建立一条TCP连接，获得速率 R/10
  - 若新应用建立11条TCP，可以获得速率 R/2！

公平性和UDP

- 多媒体应用通常不使用TCP：
  - 不希望拥塞控制限制其发送速率
- 多媒体应用宁愿使用UDP：
  - 可以以恒定速率发送音视频流，哪怕承受少量丢包
- UDP缺少拥塞控制机制的问题：
  - 拥塞发生时感知不到拥塞，也不降低发送速率，影响应用及网络的整体性能
  - 对TCP协议极不友好，损害网络公平性（TCP让出的带宽都被UDP占了）
- 给UDP加上拥塞控制：DCCP = UDP + 拥塞控制

关于TCP和UDP的思考

- 能否说TCP服务优于UDP服务？
- 多媒体应用希望的传输层服务是：带宽有保证，延迟有保证，顺序有保证，但能忍受一些丢包。TCP或UDP能够满足多媒体应用的需求吗？
- 现实中的多媒体应用既有使用TCP的、也有使用UDP的，它们分别出于什么考虑选择采用TCP或UDP？

Chapter 3: Summary

- 传输服务原理:
  - 多路复用、解复用
  - 可靠数据传输
  - 流量控制
  - 拥塞控制
- Internet中的传输服务:
  - UDP
  - TCP

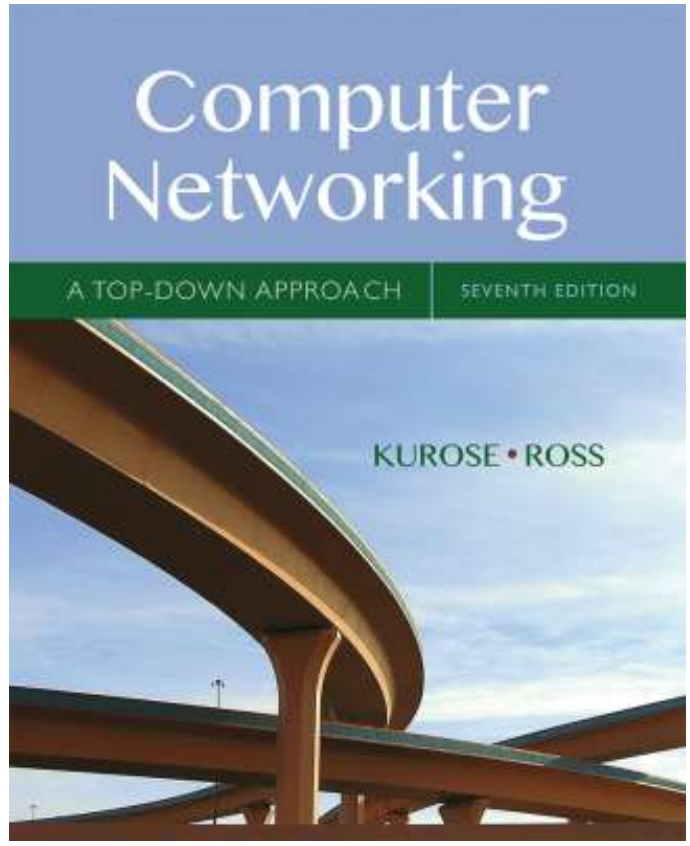
Next:

- leaving the network "edge" (application, transport layers)
- into the network "core"

作业

- 习题（10月20日）
  - 15，22，23
- 实验（10月15日）
  - UDP
- 习题（10月27日）
  - 25，27，37，40，44，45，46
- 实验（10月29日）
  - TCP
- 思考题：
  - 24，41，42，43

Chapter 4
Network Layer: The Data Plane



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Chapter 4: Network Layer

Chapter goals:

- 理解网络层服务原理，主要关注数据面
  - 网络层服务模型
  - 网络层上的重要功能：转发和选路
  - 路由器工作原理
  - 编址
  - 因特网架构
- 因特网的网络层（数据面）
  - IP协议
  - NAT，中间件

Chapter 4: Network Layer

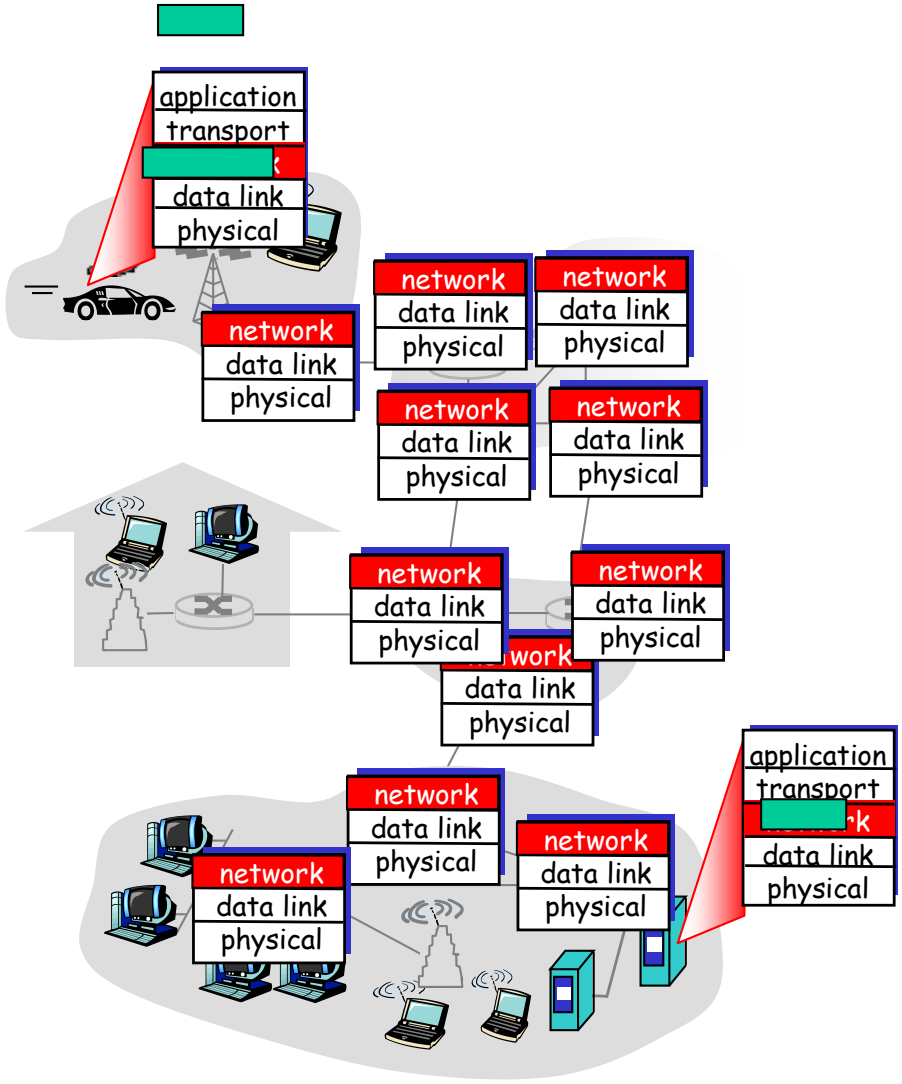
- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

网络层服务

- 网络层为传输层提供主机到主机的通信服务
- 每一台主机和路由器都运行网络层协议
- 发送终端：将传输层报文段封装到网络层分组中，发送给边缘路由器
- 路由器：将分组从输入链路转发到输出链路
- 接收终端：从边缘路由器接收分组，取出报文段交付给传输层



网络层的两个主要功能

网络层的功能

- 选路: 确定去往目的路由器的路由
- 转发: 路由器根据选定的路由，将分组从输入端口转移到输出端口

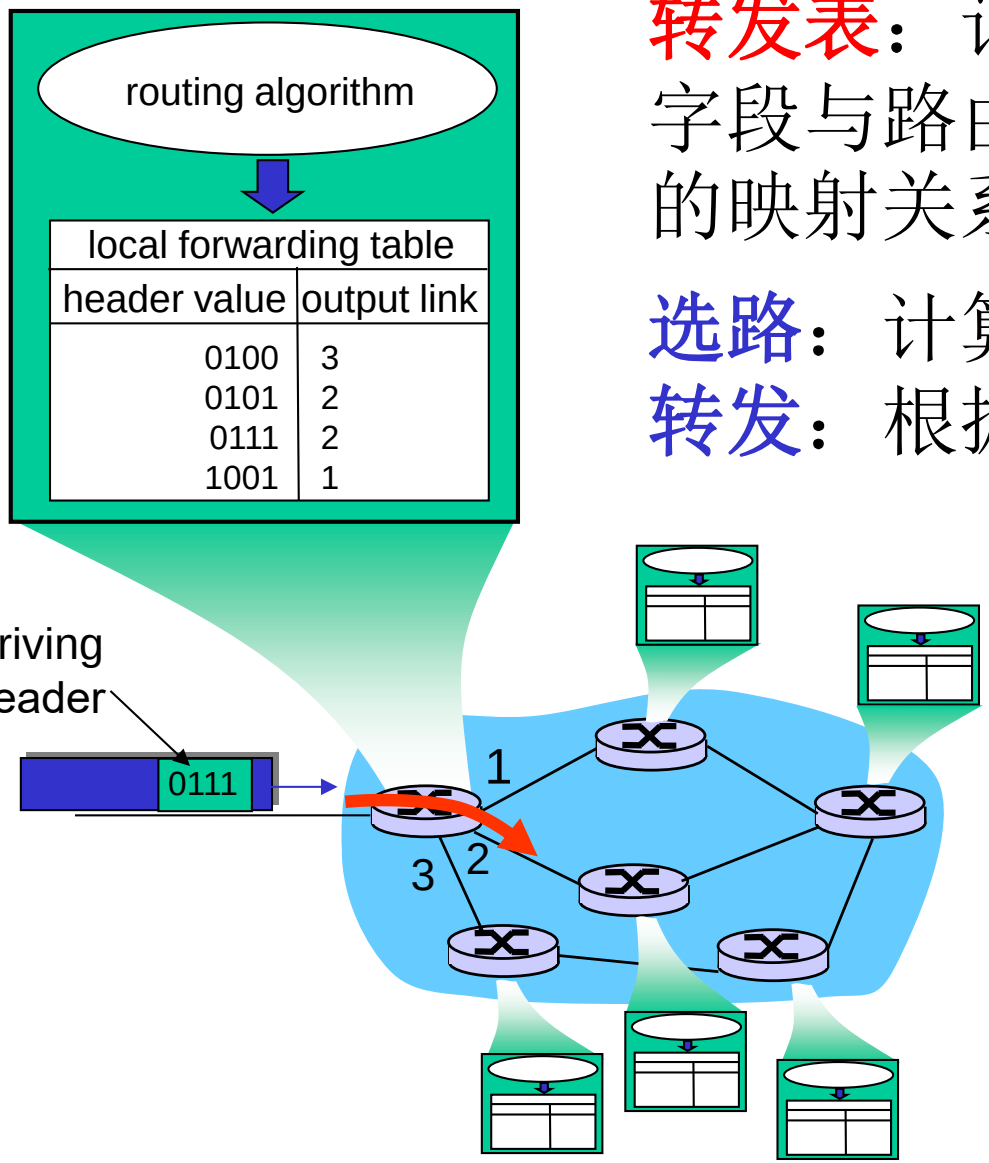
交通出行的类比

- 选路：规划到目的地的路线
- 转发：在到达路口时，根据选好的路线转移到下一个路段

forwarding

routing

选路和转发的关系



转发表：记录分组头中某个字段与路由器输出端口之间的映射关系

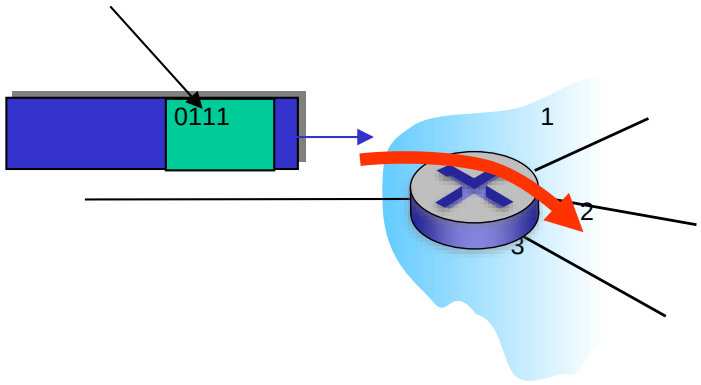
选路：计算转发表

转发：根据转发表转运分组

网络层：数据面和控制面

数据面（Data plane）

- 执行数据传输的功能属于数据面
- 转发是数据面功能，在路由器内部实施分组转运
- 是路由器本地功能



控制面（Control plane）

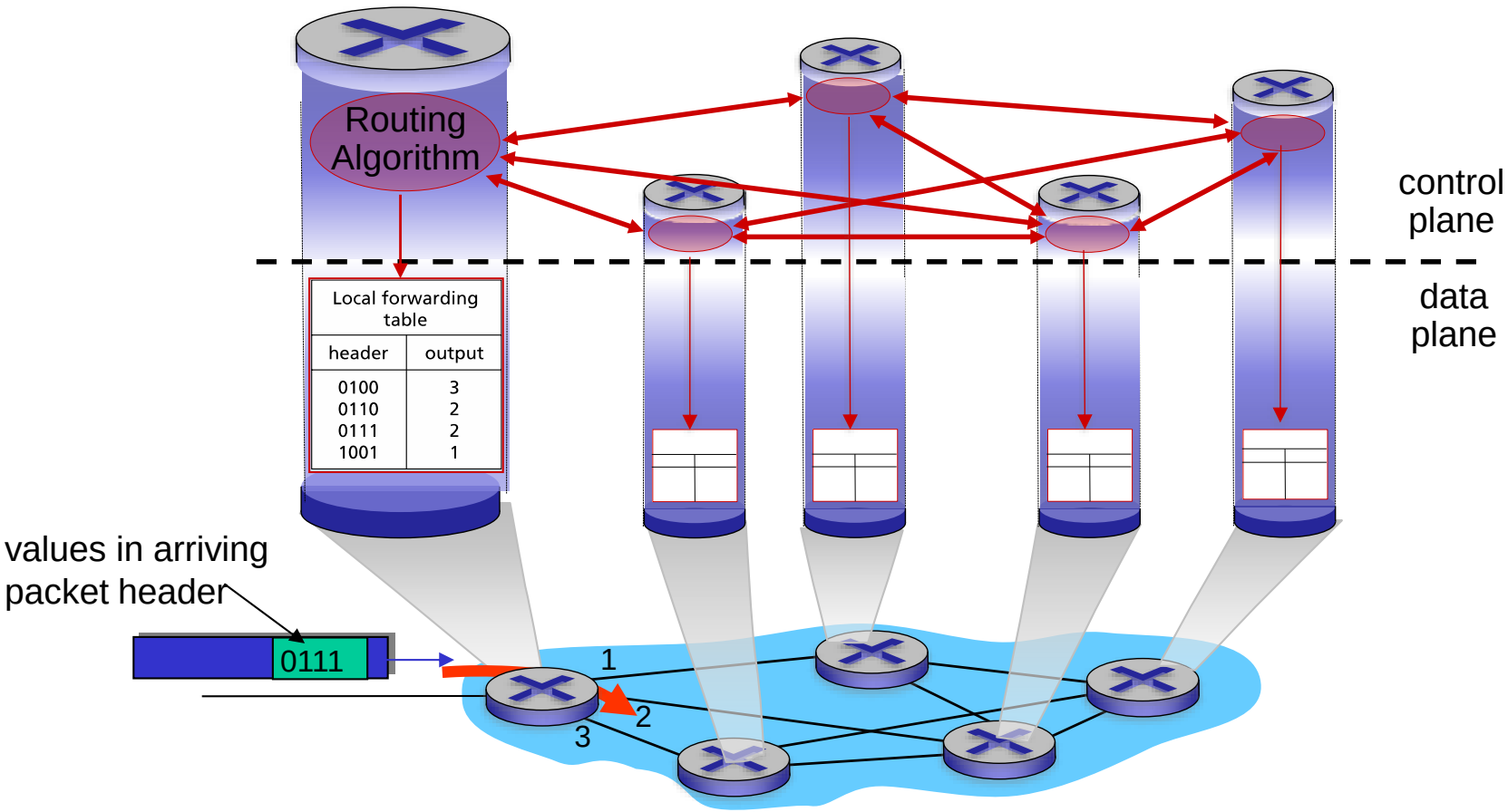
- 控制数据传输的功能属于控制面
- 选路是控制面功能，确定分组如何去往目的节点
- 是网络范围的功能

两种控制面实现方法

- 传统寻路算法：在路由器中实现
- 软件定义网络：在服务器中实现

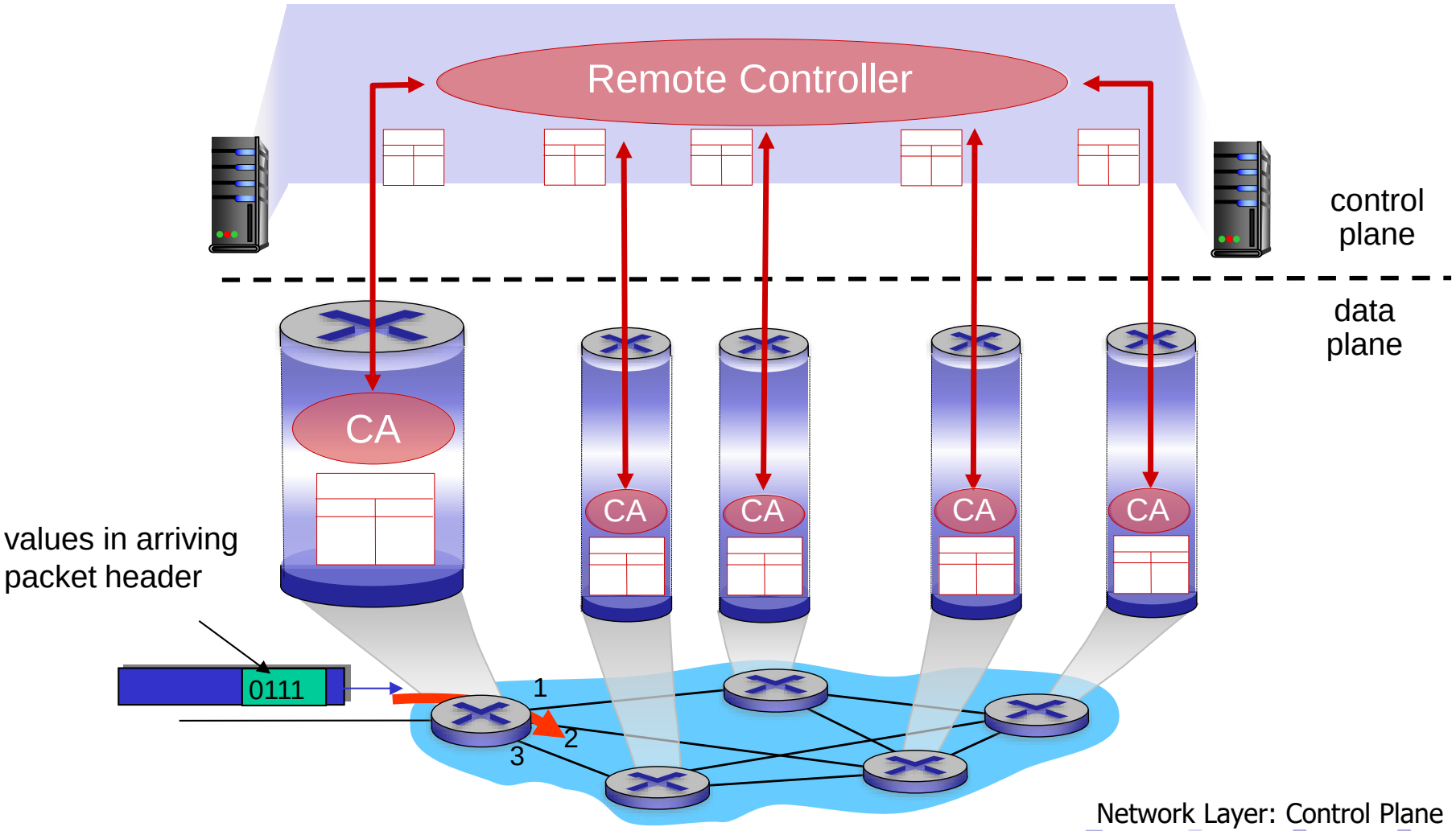
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs)



网络服务模型

网络服务模型：

- 定义了分组在发送终端与接收终端之间的传输特性

可能的网络服务：

- 保证交付
- 具有时延上界的保证交付
- 有序分组交付
- 保证最小带宽
- 安全性

网络层服务模型举例

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	no	no	no	no	no (inferred via loss)
ATM	CBR	恒定速率	yes	yes	yes	无拥塞发生
ATM	ABR	最小速率	no	yes	no	yes

- 不同架构的网络提供的网络层服务可能不同
- 同一个网络也可以提供不同的网络层服务

Chapter 4: Network Layer

- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

面向连接的服务，无连接服务

两种基本的网络类型：

数据报网络：提供网络层无连接服务

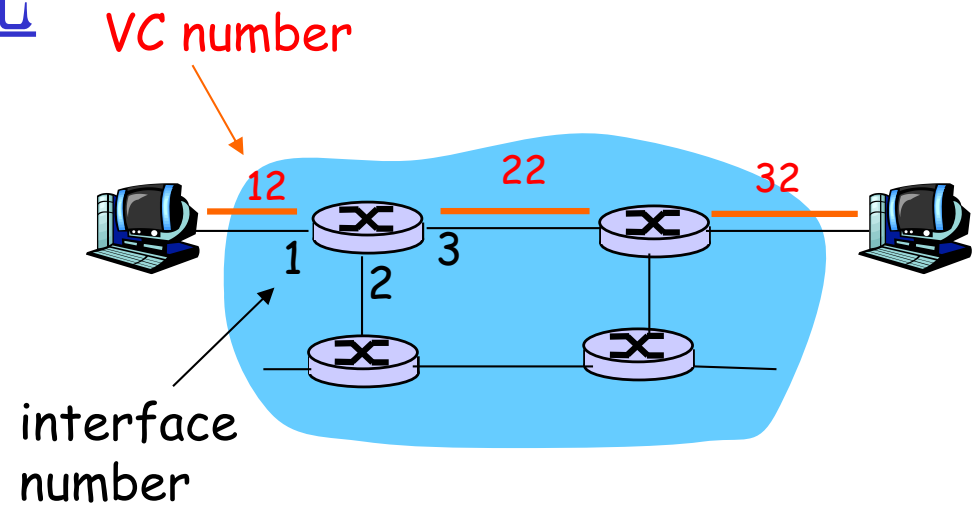
虚电路网络：提供网络层面向连接服务

- 网络层服务
  - 主机-主机
  - 一个网络不能同时提供无连接服务和面向连接的服务
  - 在网络核心实现
- 传输层服务
  - 进程-进程
  - 可同时提供无连接服务和面向连接的服务
  - 在网络边缘实现

虚电路（Virtual circuits）

- 虚电路网络在网络层上使用连接，这些网络层连接称为虚电路
- 虚电路是一条端到端路径：
  - 传输分组前需建立虚电路，传输结束后应拆除虚电路
  - 所有分组在这条虚电路上传输（分组传输是保序的）
  - 如果将路由器资源（带宽、缓存等）预先分配给虚电路，则虚电路可以提供可预期的网络服务
- 建立虚电路的本质：
  - 预先选好从源主机到目的主机的路径，此后分组仅沿选好的路径传输，是否分配资源是可选的

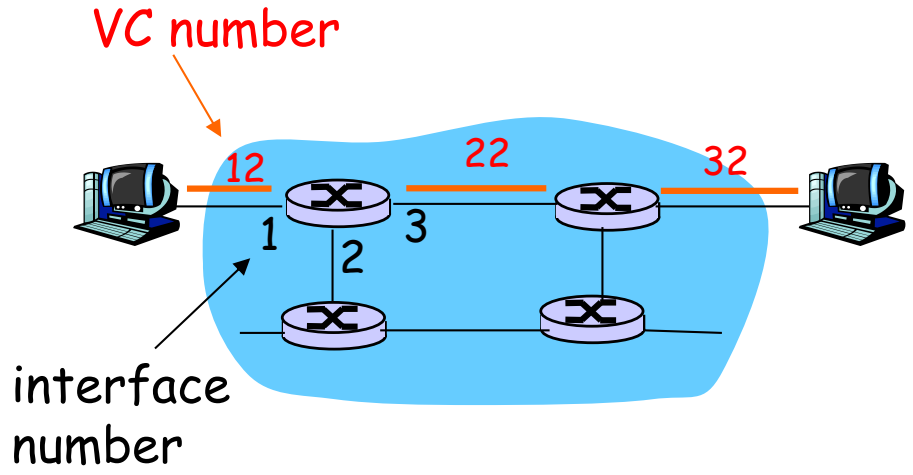
虚电路（VC）实现



一条虚电路由以下几部分组成:

1. 从源主机到目的主机的端到端路径
2. 途经每条链路时的虚电路号（用于区分经过该链路的不同虚电路，仅有本地意义）
3. 沿途每个路由器中的转发表项：<进入端口，进入VC号> --> <输出端口，输出VC号>

虚电路转发



左上角路由器中的转发表

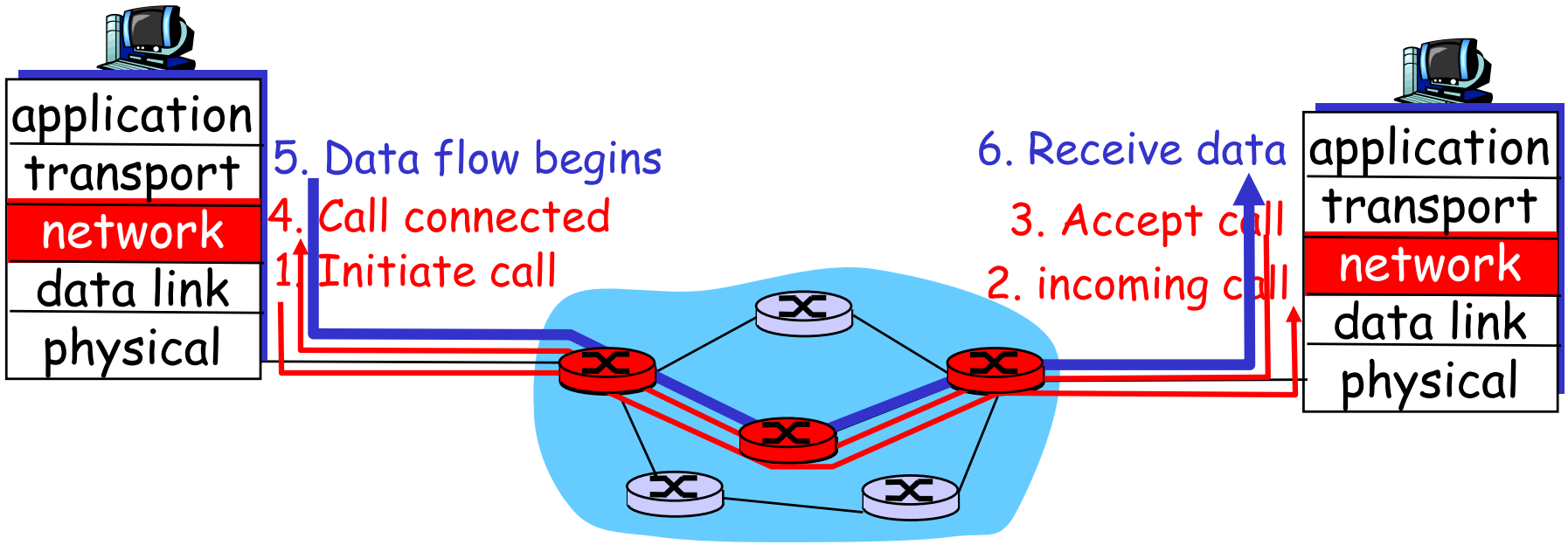
Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
…	…	…	…

- 分组携带VC号，路由器利用输入端口和VC号查找转发表
- 转发前，路由器使用输出链路上的VC号替换分组中的VC号

注意：在虚电路上传输分组时，分组只需携带VC号，不需携带目的地址

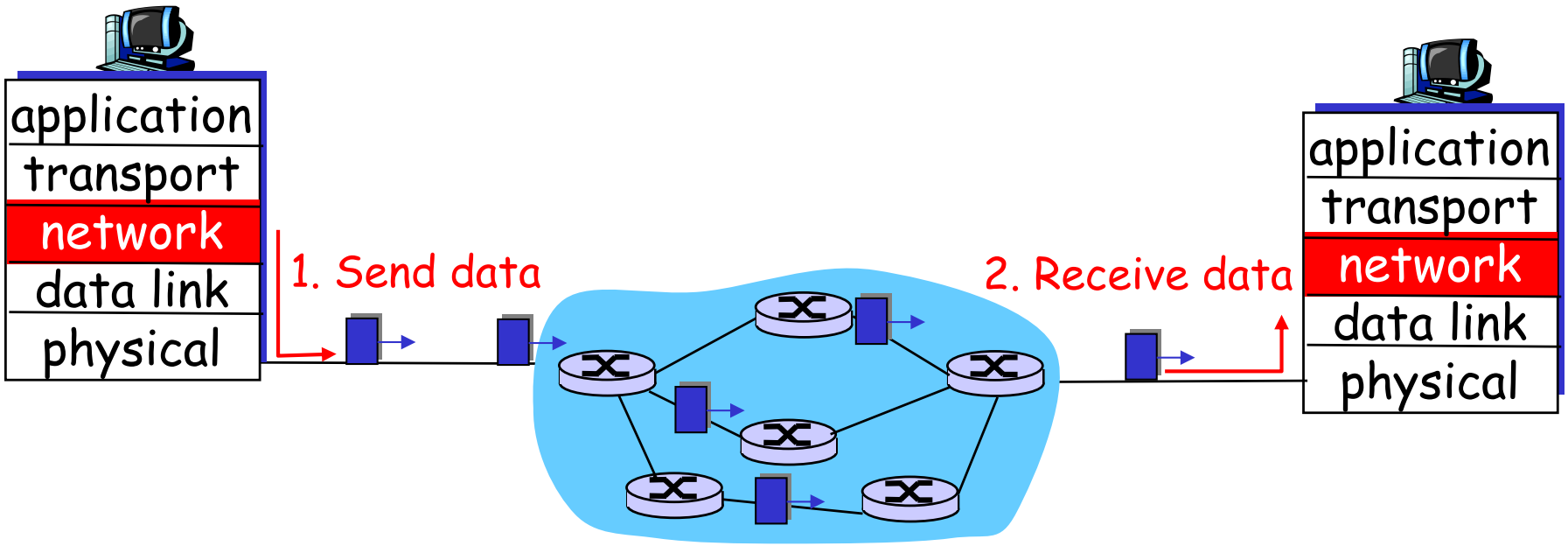
虚电路: 信令（signaling）协议

- 信令报文：专门用于建立、维护、拆除虚电路的控制报文
- 信令协议：交换信令报文的协议



数据报网络

- 分组携带目的主机地址，路由器按目的地址转发分组
- 路由器中的转发表记录目的地址→输出端口的映射
- 转发表被选路模块修改，约1～5分钟更新一次
- 同一对主机之间传输的分组可能走不同的路径，从而可能重排序



Datagram or VC network: why?

Internet（数据报网络）

- 为计算机通信而设计：
  - 早期的网络应用对网络服务没有严格要求（弹性应用）
  - 用户免费使用网络
- 终端（计算机）具有智能
  - 可将复杂的工作（如差错控制）推到网络边缘，以保持网络核心简单

ATM（虚电路网络）

- 由电信网发展而来
  - 注重用户体验，追求高质量服务（用户付费了）
- 终端无智能或很少智能
  - 复杂工作由网络完成，以保持终端简单

Datagram or VC network: why?

- 数据报网络提供最小服务的好处：
  - 可运行在各种链路上
  - 为传输层服务的设计增加了灵活性，可简、可繁
- 符合当初因特网设计的原则：
  - 网络只提供尽力而为的服务
  - 复杂的工作推到网络边缘（增加新服务只涉及终端）
- 这两个好处带来了因特网今天的发展：
  - IP over everything（IP可以运行在任何物理网络上）
  - Everything over IP（任何应用可以运行在IP上）

Chapter 4: Network Layer

- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

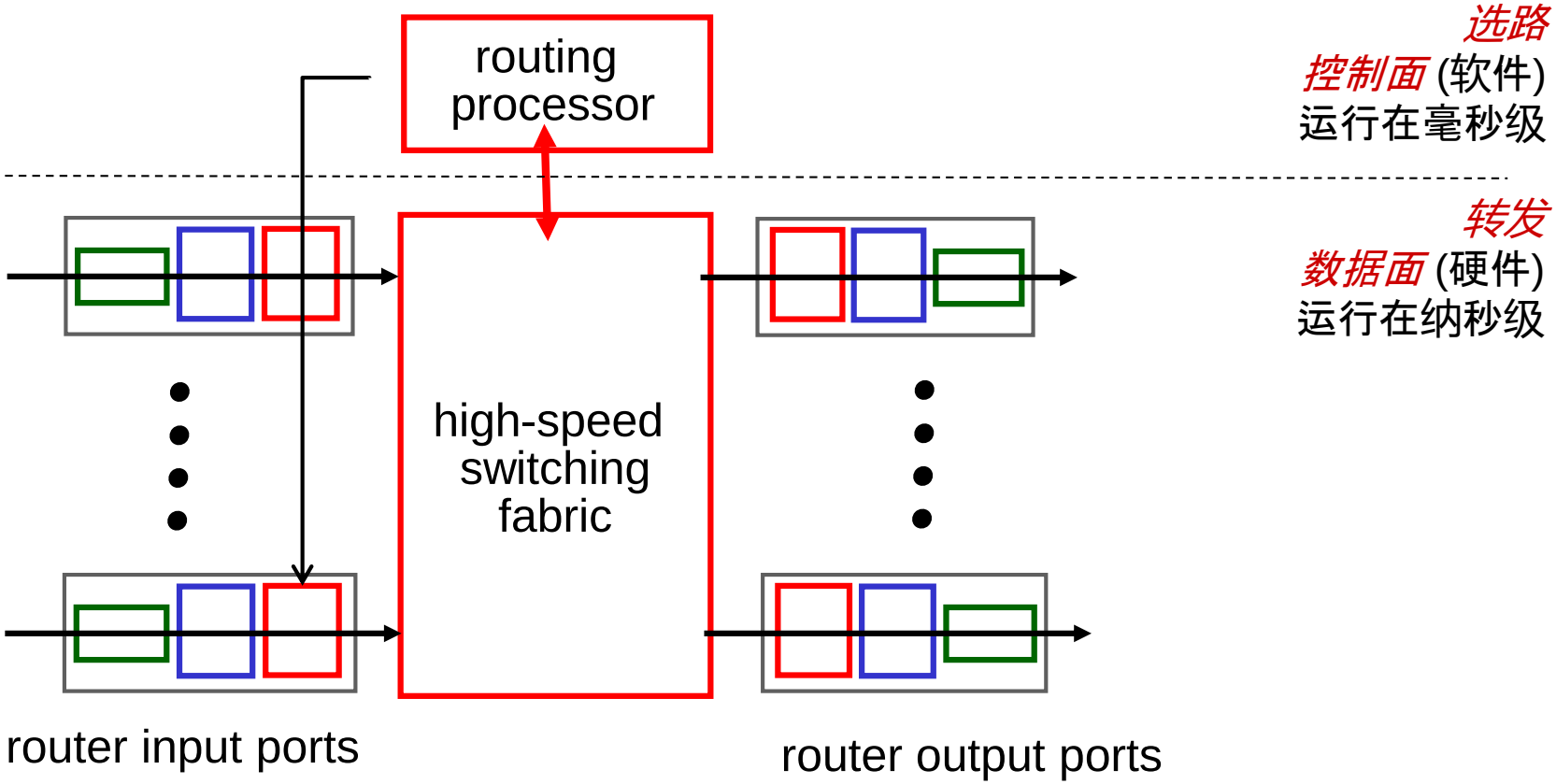
4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

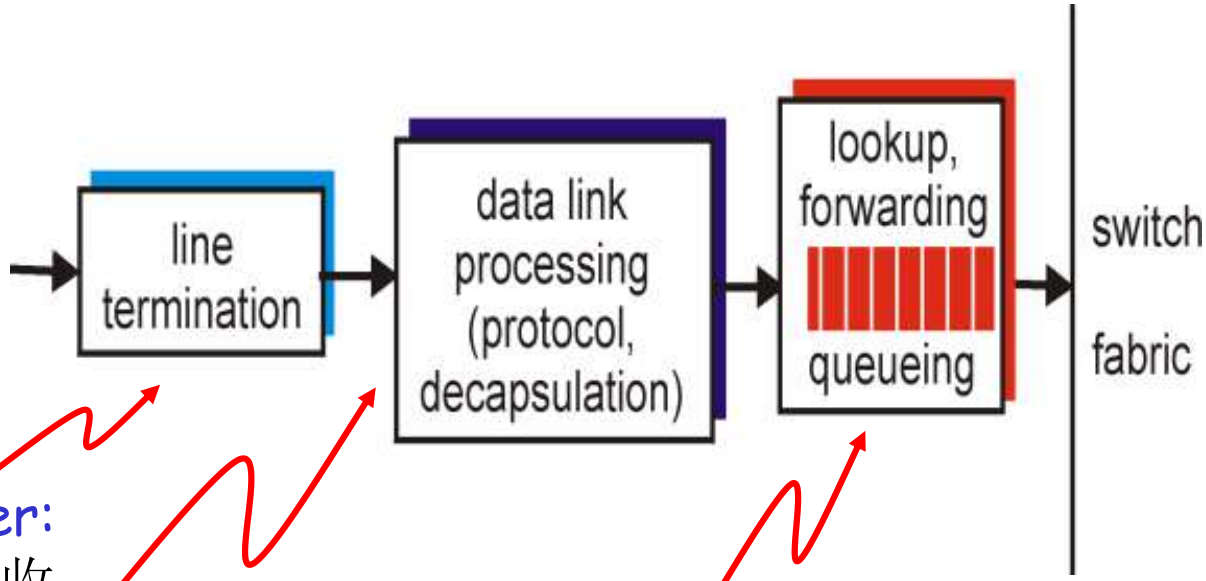
路由器架构概述

路由器的两个主要功能:

- 选路：运行选路协议，计算转发表
- 转发：依据转发表，从输入链路到输出链路转发数据报



输入端口功能



Physical layer:
比特流接收

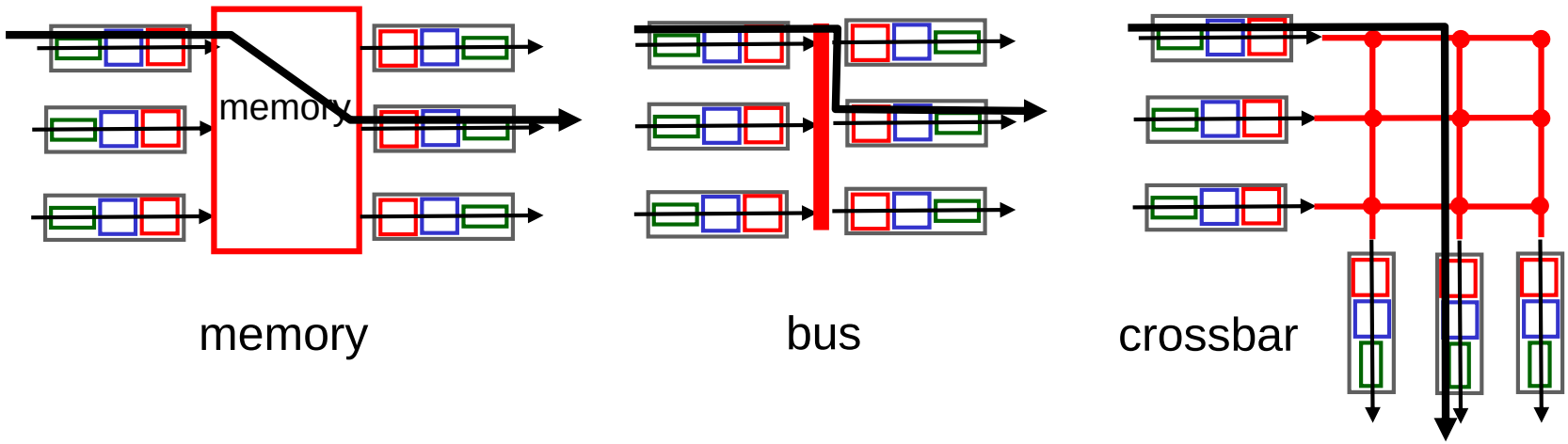
Data link layer:
- 从比特流中提取帧
- 处理帧
- 提取IP数据报

Decentralized switching:
- 查表：每块线卡（line card）上都有转发表的一个镜像，查表仅在本地进行
- 排队：当交换结构阻塞时，分组需在此排队
- 转发：通过交换结构将分组发送到输出端口（这个过程也称为交换，switch）

性能要求：以“线速”完成输入端口处理

交换结构

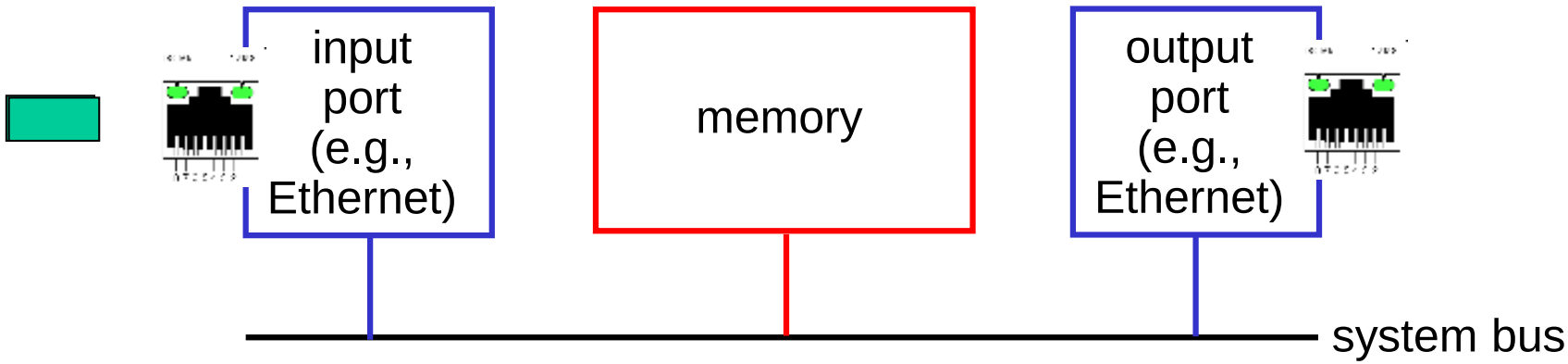
- 路由器中的互联网络，用于在输入端口、输出端口和选路处理器之间转运分组
- 交换速率：
  - 通常是输入/输出链路速率的若干倍
- 三种类型的交换结构



通过内存交换

第一代路由器:

- 即传统计算机，在CPU的直接控制下完成交换
- 数据包拷贝到系统内存中进行交换
- 交换速率受限于内存带宽：每个数据包穿过系统总线2次

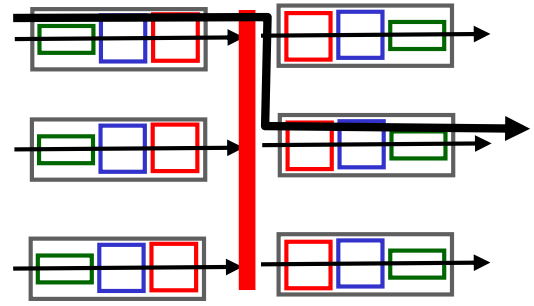


现代路由器的做法:

- 每个端口使用一个内存接口硬件连接到存储系统
- 一个控制器硬件在端口之间传输控制消息（不需要CPU参与）
- 输入端口将一个包放入内存后，接口硬件通过控制器向输出端口发送一个消息，输出端口从内存指定位置读取包，发回响应消息

通过总线交换

- 数据包通过一条共享总线，从输入端口缓存转移到输出端口缓存
- 每个输入和输出端口通过一个接口硬件连接到总线上，每个端口被分配一个内部标签
- **总线竞争：**
  - 总线协议防止多个端口同时传输，比如，采用时分多路复用的方法
  - 各个输入端口在总线上轮流广播分组，每个输出端口根据分组携带的内部标签接收发给本端口的分组
- 交换速率受限于总线带宽

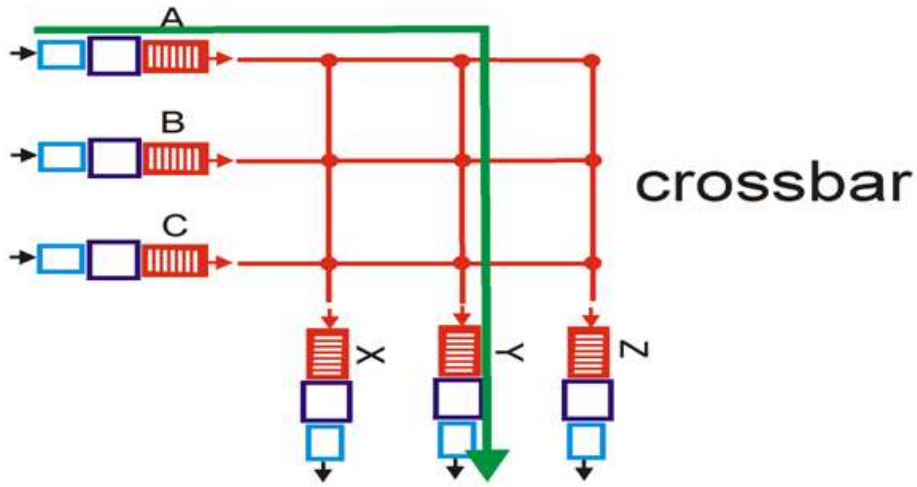


bus

Cisco 5600采用32Gbps总线，满足接入路由器和企业路由器的交换要求

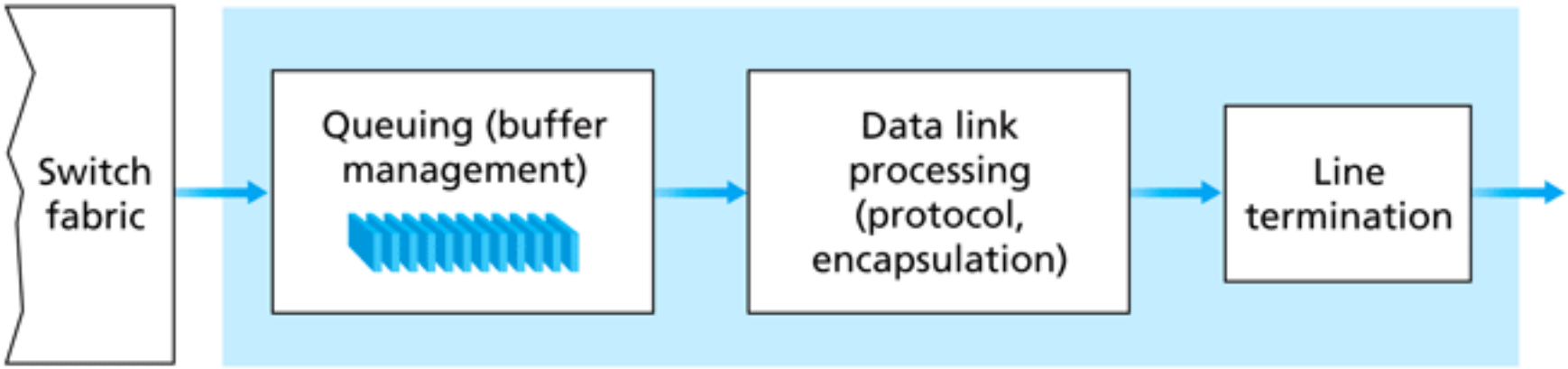
通过互联网络交换

- 交换结构控制器通过控制交叉点的开、闭，在输入端口与输出端口间建立内部专用电路
- 多对端口间可以并行传输
- 分阻塞型与非阻塞型，阻塞型互联网络会产生阻塞
- Cisco 12000通过互联网络获得60 Gbps的交换速度



- 先进设计：
  - 将输入端口和输出端口连接到N个并行运行的交换结构
  - 将分组划分成若干个固定长度的信元（cell），同时送入交换结构
  - 信元离开交换结构后再组装成分组

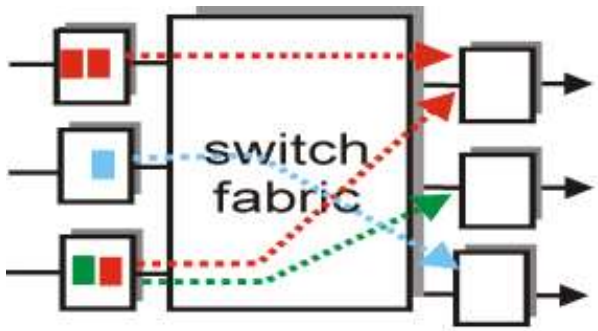
输出端口功能



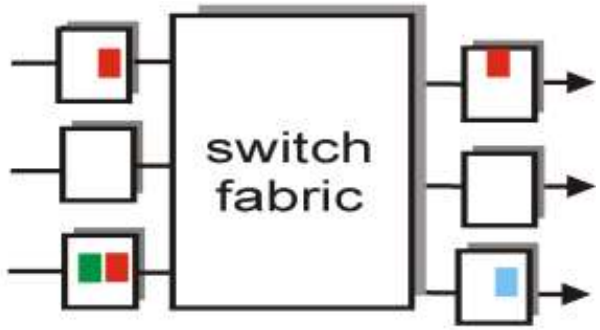
- 网络层处理：
  - 组装：若需要，将交换结构输出的信元组装成分组
  - 排队：若输出端口来不及发送，分组在此排队
  - 调度：输出端口每次选择一个分组发送
- 链路层处理：执行链路层协议，封装
- 物理层处理：将比特流转换成物理信号

输入端口排队与丢包

- 当交换结构不能及时将输入端口的分组转移到输出端口时，输入端口处形成排队
- 排队带来的问题：
  - 队头阻塞：队头分组阻塞其后分组的转发
  - 丢包：当输入队列溢出时，发生丢包
- 当交换结构速率至少为端口速率的n倍时（n为输入端口数），可以消除输入端口的排队，但路由器成本提高了

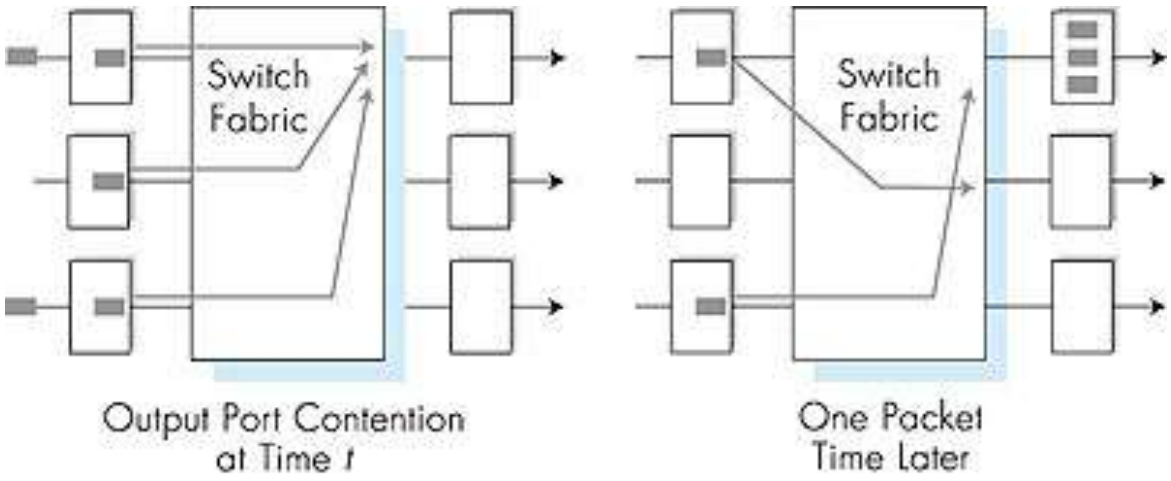


output port contention
at time t - only one red
packet can be transferred



green packet
experiences HOL blocking

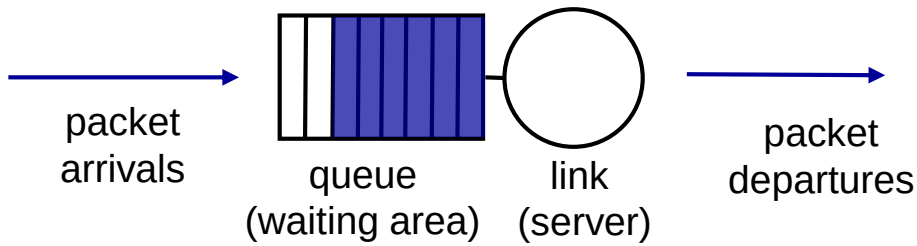
输出端口排队与丢包



- 多个输入端口同时向一个输出端口发送时，在输出端口形成排队
- 当输出队列满时，发生丢包
- **输出端口排队是不可避免的**，设置多大的输出队列是一个问题：
  - 增大输出队列：可以减少丢包的发生，但会增加内存消耗，并增大分组延迟（延迟太大的分组最终被重传，浪费资源）
  - **输出队列并不是越长越好！**

调度策略

- 调度：输出端口选择下一个要发送的分组
- 先来先服务：
  - 按照分组到达输出队列的次序发送
- 分组丢弃策略，当输出队列满时丢弃哪个分组？
  - 弃尾：丢弃到来的分组
  - 按照优先级丢弃：低优先级分组
  - 随机丢弃：随机选择一个分组丢弃，如Ramdom Early Detection （RED）



随机早检测（RED）

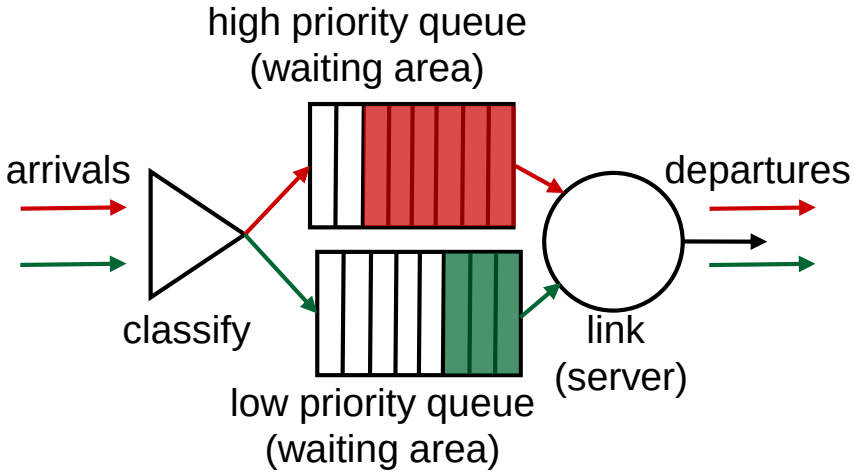
- 在队列满之前就开始丢弃分组，设计为和TCP拥塞控制机制一起使用
- 丢弃策略：
  - 路由器在每个端口上记录输出队列的平均长度：

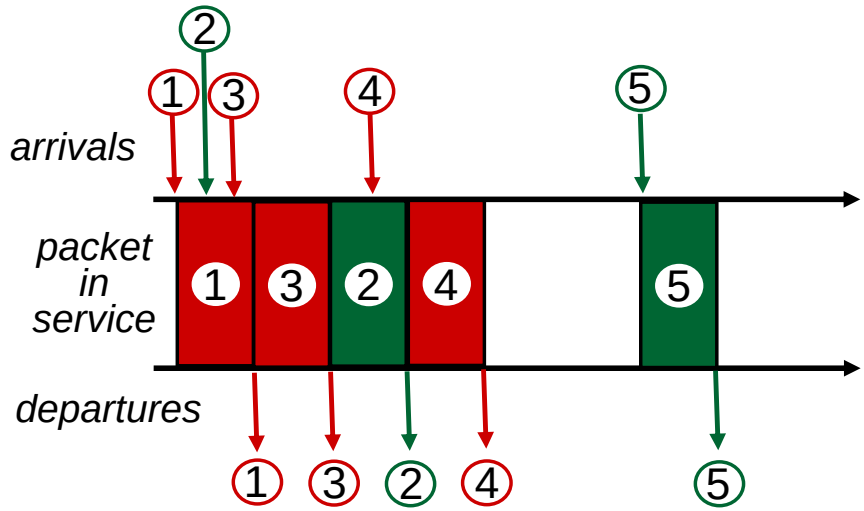
    $$AvgLen = (1- Weight)\times AvgLen + Weight\times SampleLen$$

  - 当平均队列长度达到第一个阈值min_{th}时，按照丢弃概率 p 丢弃到来的分组
  - 当平均队列长度达到第二个阈值max_{th}时，丢弃每一个到达的分组
  - 概率 p 是平均队列长度和上一次丢弃距当前时间的函数，分组队列长度越大，丢弃间隔越大，p越大

调度策略

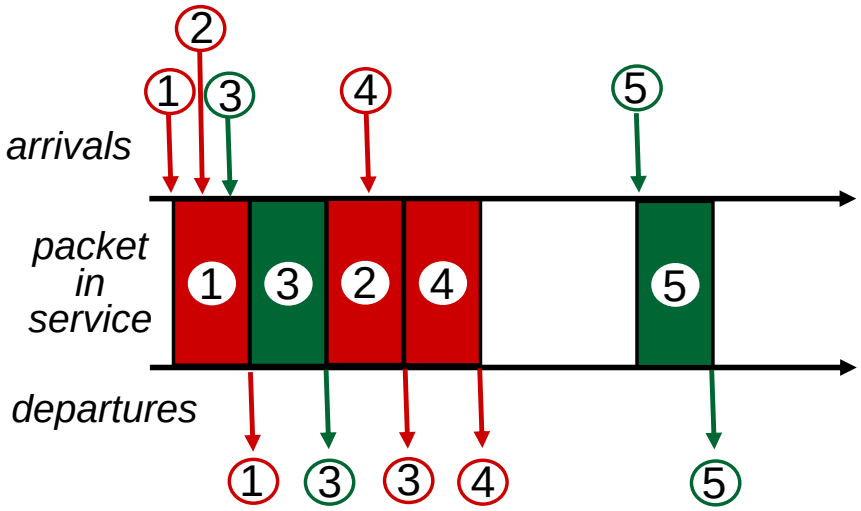
- 优先级调度：
  - 严格按优先级发送分组
  - 同一优先级的分组按先来无服务的顺序发送
- 设置多个优先级类，每个分组被标记一个优先级：
  - 可以根据分组头中的某些域，如IP地址、端口号等设置优先级
- 非抢占式优先级排队：
  - 分组一旦开始传输，就不能被中断



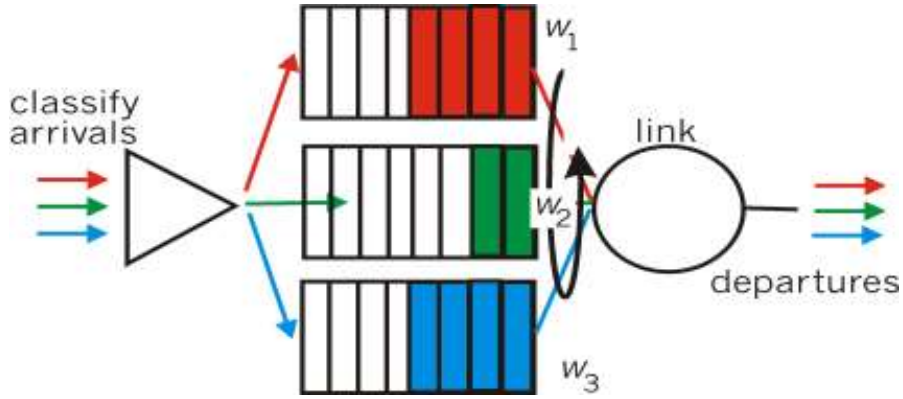


调度策略

- 轮询调度：
  - 在几个队列之间轮流提供服务，每次选择一个队头分组发送



- 加权公平排队：
  - 每个队列分配一个带宽权重（比例）
  - 按照每个队列的带宽权重，选择一定数量的分组发送



Chapter 4: Network Layer

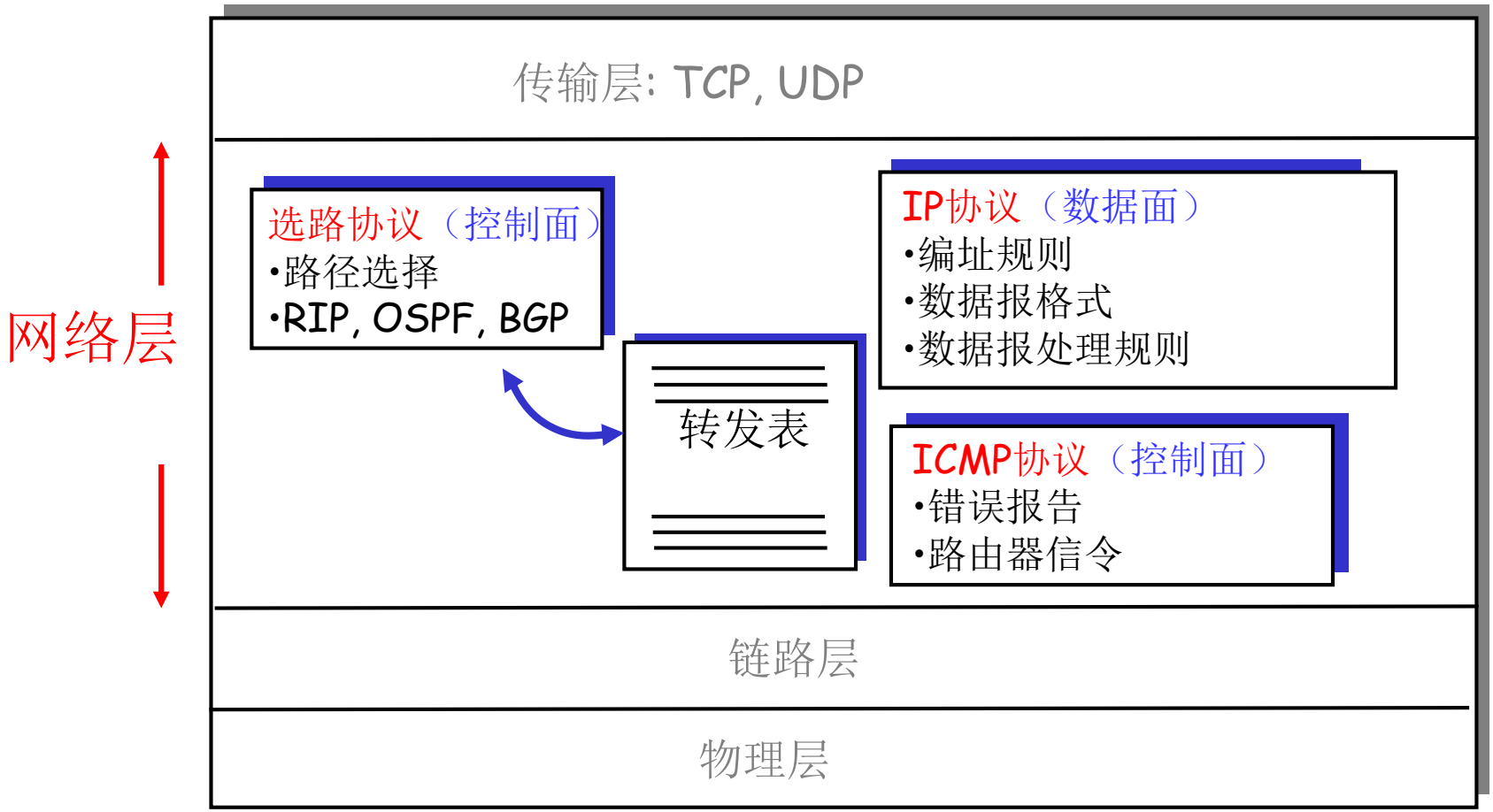
- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

因特网的网络层

主机和路由器的网络层功能包括：



Chapter 4: Network Layer

- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

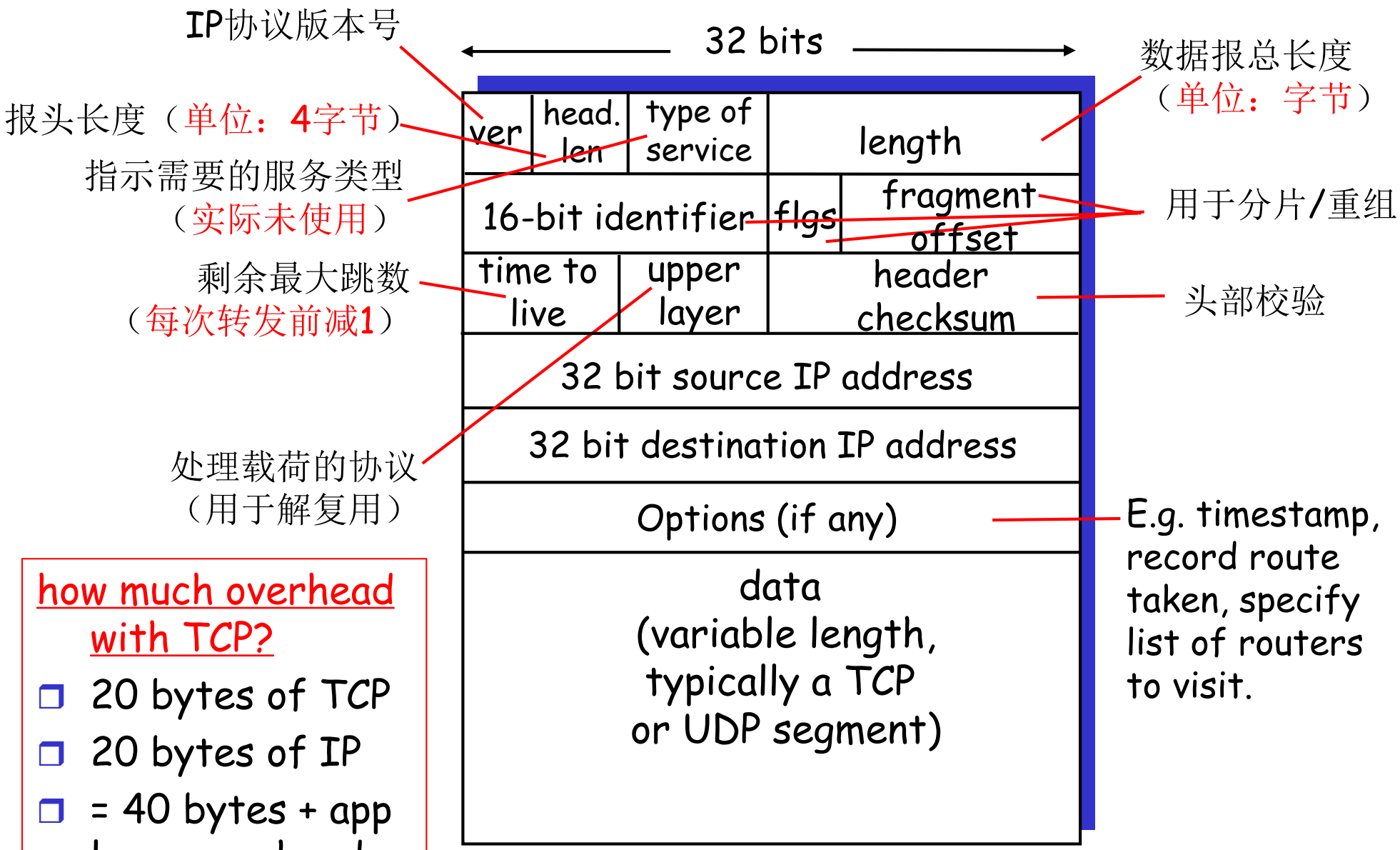
4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

IP只提供最小的传输服务

- IP提供的服务：
  - 将数据报交付到目的地址和目的协议
- 尽管IP不提供任何服务承诺，但仍尽最大努力解决分组在网络中传输时可能遇到的一些问题：
  - 数据报过大无法通过某个中间网络（措施：对数据报进行分片）
  - 数据报可能因寻路错误在网络中循环（措施：设定数据报的最大转发次数）
  - 报头出错可能导致误投递（措施：对报头检错）

IP数据报格式



how much overhead with TCP?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

Chapter 4: Network Layer

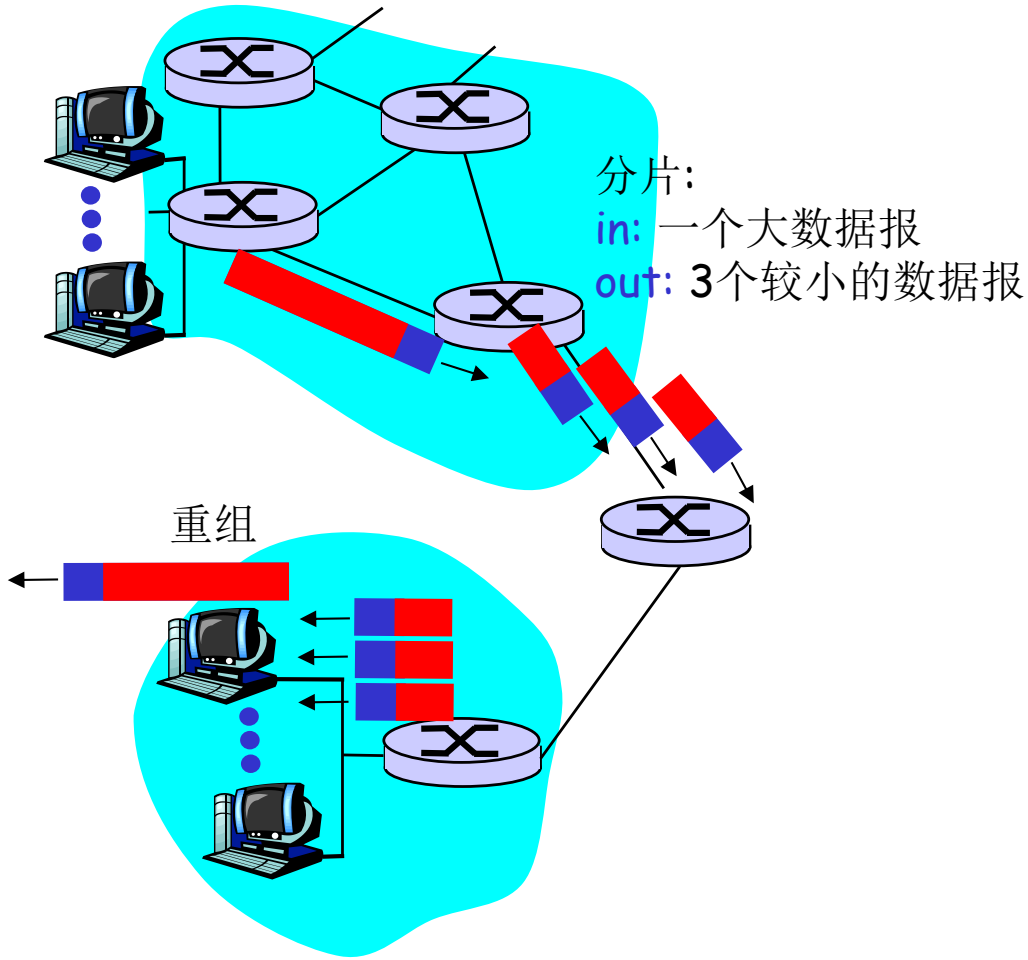
- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

IP分片与重组

- 链路层帧能承载的最大数据字节数称为MTU （Max. Transmission Unit）
  - 不同类型的链路可能具有不同的MTU
- 传输过程中，较大的IP数据报可以被分片：
  - 将数据报载荷划分为若干较小的数据块，每个数据块封装成一个独立的数据报传输
  - 数据报在传输的过程中可以被多次分片，但仅在目的主机上重组



分片的报头

- 分片的报头取自原始数据报
- 与分片有关的字段：
  - 标识：每个分片必须携带与原始数据报相同的标识
  - 偏移量：指示分片中的数据在原始数据报载荷中的位置
  - 标志位：
    - MF（more fragments）：最后一个分片的MF=0，其余分片的MF=1
    - DF（don’t fragment）：DF=1表示不允许对数据报分片
- 分片报头中的以下字段需要修改：
  - 总长度，偏移量，MF，TTL，头部检查和

分片的数据长度

- 假设原始数据报的报头长度为H，分片的数据长度 N，应满足：H+N ≦ MTU
- 由于偏移量只有13比特，除最后一个分片外，其余分片的数据长度应为8字节的整倍数
- 考虑到分组传输效率，除最后一个分片外，分片的数据长度 N 应为满足以上两个条件的最大整数

数据报分片的处理过程

- 根据报头长度H和输出线路的MTU，确定分片的最大数据长度N
- 将数据报的载荷划分成长度为N的若干数据块（最后一个数据块可能不足N字节）
- 将原始报头加到每一个数据块的前面，修改报头中的以下字段：
  - 总长度 = H + 数据块长度
  - 最后一个报头的MF位置0，其余报头的MF位置1
  - 偏移量 = 数据块在原始数据报载荷中的字节序号/8
  - TTL=TTL-1
  - 计算头部检查和

分片的例子

- 例：要将一个总长度=4000字节的IP包发送到MTU=1500字节的链路上，IP报头长度H=20字节
- 数据块最大长度 N = 1480字节
- 原始数据报的载荷（3980字节）被分成三个数据块，长度分别为1480字节、1480字节和1020字节

分片序号	**总长度**	MF	**偏移量**
1	1500（=1480+20）	1	0
2	1500（=1480+20）	1	185（=1480/8）
3	1040（=1020+20）	0	370（=185+185）

重组

- 将收到的分片重新组装成原始数据报的过程称为重组，重组在目的主机中进行：
  - 收集分片：目的主机使用 <源IP地址，标识> 确定属于同一个数据报的分片
  - 利用最后一个分片计算原始数据报长度：

    原始数据报长度=偏移量×8 + 分片总长度

    原始数据报载荷=偏移量×8 + 分片总长度 – 报头长度
  - 组装：将各分片中的数据块按照其在原始数据报载荷中的偏移量重组

分片的问题

- 分片的开销：
  - 降低了路由器的吞吐量
  - 消耗了目的主机的资源：每个重组的数据报需要一个重组缓冲区和一个重组定时器
- 针对分片的DoS攻击：
  - 攻击者发送一系列奇怪的分片，消耗目的主机的资源
- IPv6取消了路由器分片的功能：
  - 源主机发送探测报文，确定路径上的最小MTU
  - 源主机构造的数据报大小不超过最小MTU
  - 路由器丢弃超大的数据报，并发送错误报告

重要知识点

- 分片的方法：
  - 确定分片的大小
  - 修改报头域（尤其是偏移量）
  - 根据最后一个分片计算原始数据报的长度
- 数据报在传输过程中可以被多次分片，但仅在目的主机上组装，为什么？
- 注意报头中几个长度的单位：
  - head length：4个字节
  - Length：1个字节
  - fragment offset：8个字节

Chapter 4: Network Layer

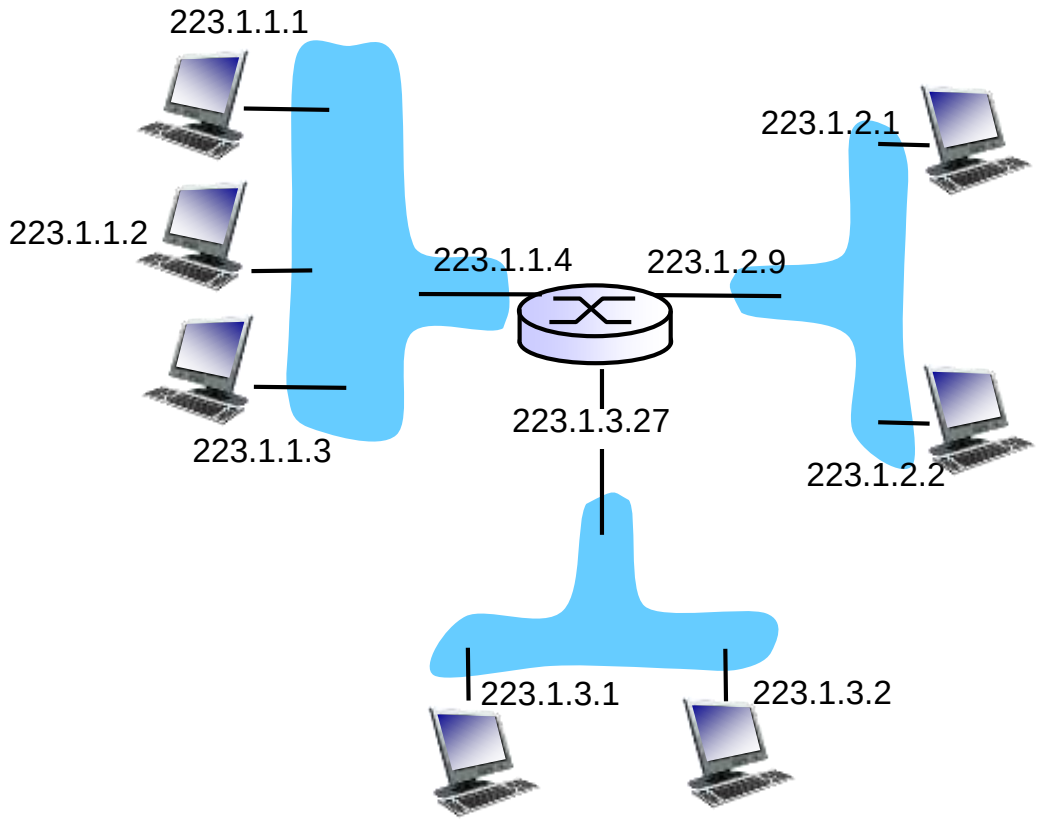
- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

1. IP编址

- 接口（interface）：
  - 主机/路由器与物理链路的边界
  - 路由器有多个接口
  - 主机典型地有一个或两个接口（比如以太网接口、Wifi接口）
- IP address:
  - 每个网络接口对应一个IP地址
  - IP地址是一个32位的二进制数，通常用点分十进制数表示



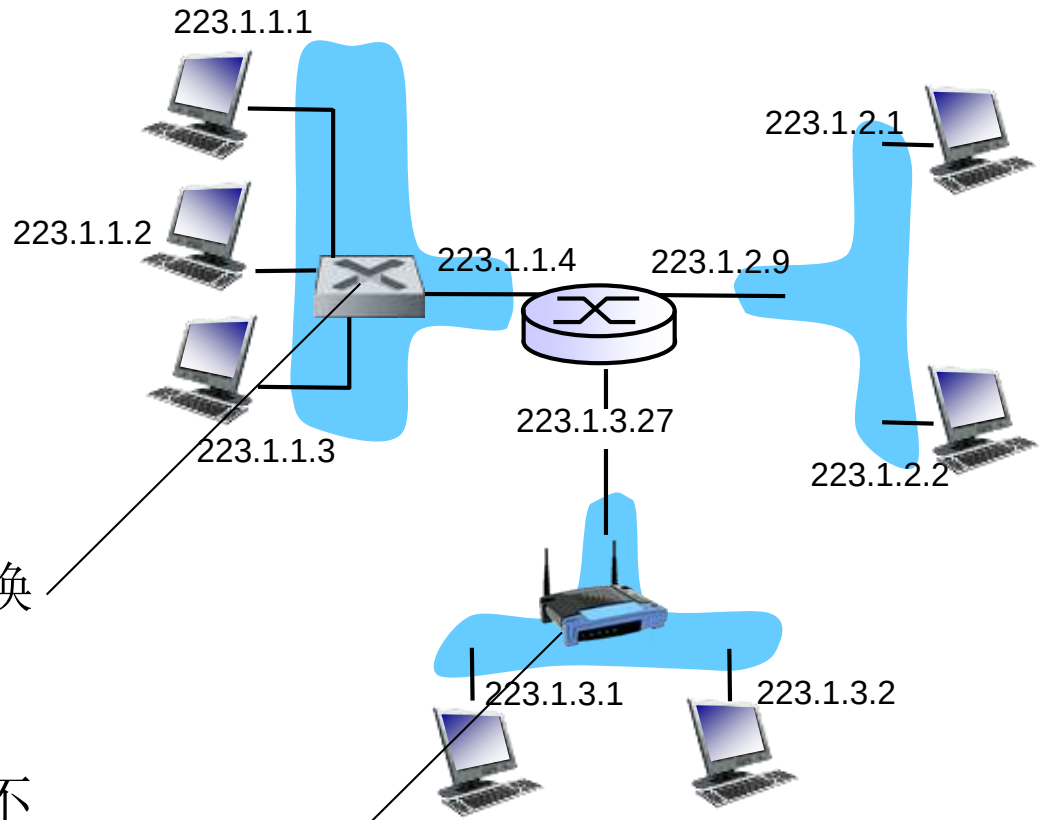
223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

IP编址

Q: 接口之间是怎么连接的？

A: 将在第6、7章介绍

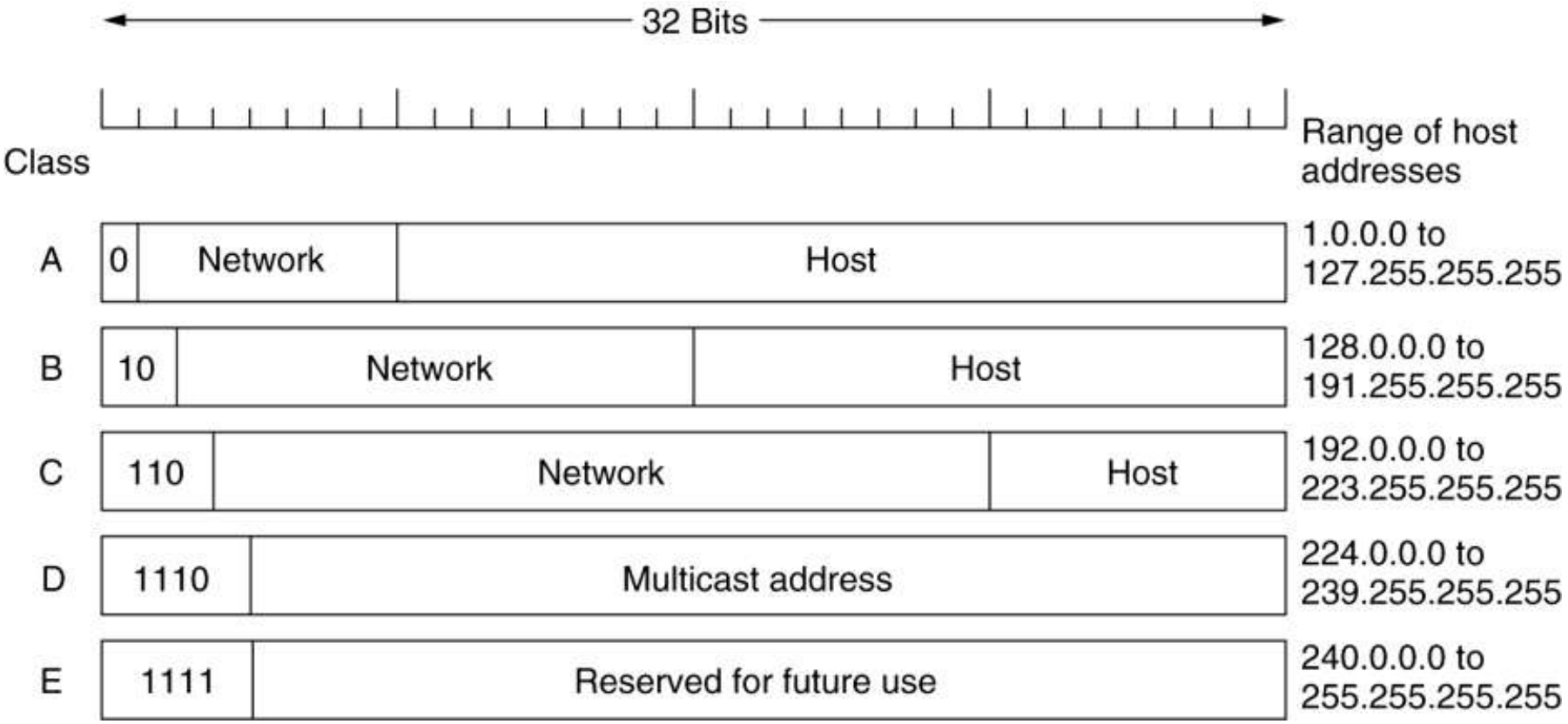


A: 以太网接口通过以太网交换机连接

目前: 暂不考虑一个接口如何不经由路由器连接到另一个接口

A: 无线WiFi接口通过WiFi基站连接

基于类的编址（早期）


32 Bits
Class
Range of host addresses
A
0
Network
Host
1.0.0.0 to 127.255.255.255
B
10
Network
Host
128.0.0.0 to 191.255.255.255
C
110
Network
Host
192.0.0.0 to 223.255.255.255
D
1110
Multicast address
224.0.0.0 to 239.255.255.255
E
1111
Reserved for future use
240.0.0.0 to 255.255.255.255

单播地址结构

- 单播地址除类别标识外，其余比特被划分成网络号和主机号两部分：
  - 网络号：标识一个物理网络
  - 主机号：标识该物理网络上的一个网络接口
- 根据该编址方法可知，同一个物理网络上的网络接口，它们的IP地址具有相同的网络号

地址分配

- 因特网中的每个接口必须具有唯一的IP地址
- 为在因特网范围内保证IP地址的全局唯一性：
  - 网络号由ICANN统一分配
  - 主机号由网络管理员统一分配
- 建立私有网络的组织可以自己选择网络号，但同样必须保证每个网络号在私有网络内的唯一性

特殊的地址

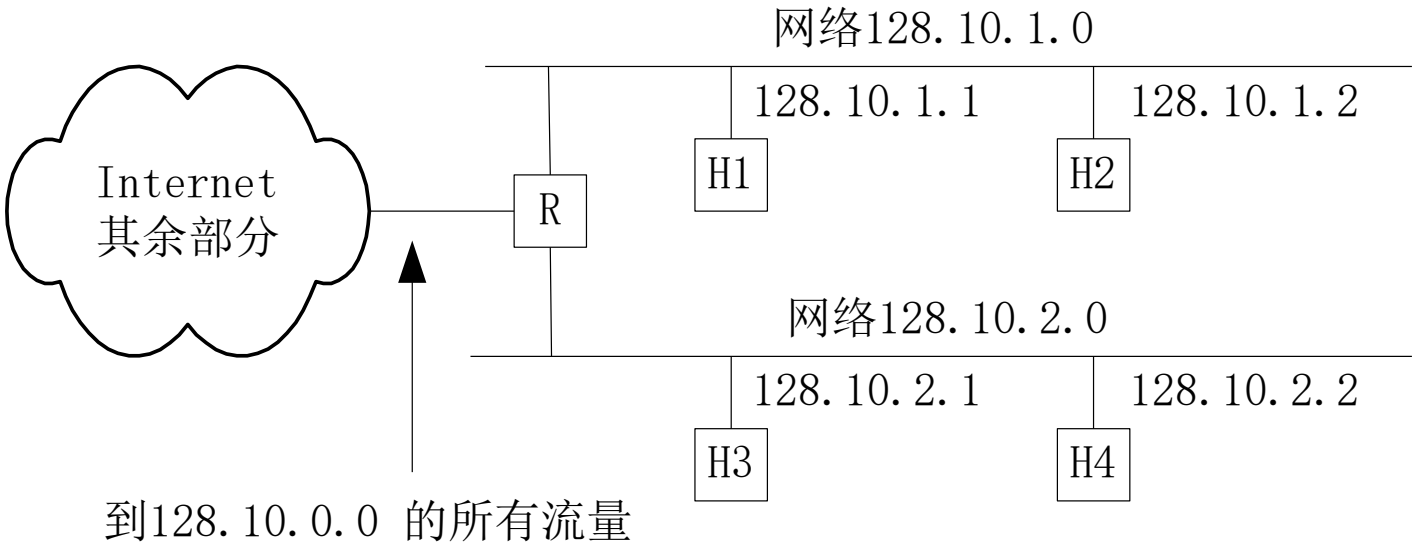
- 全0或全1的网络号及主机号是特殊地址，从不分配给特定的网络接口：
  - 网络号有效、主机号全为0的地址：保留给网络本身。
  - 网络号有效、主机号全为1的地址：保留作为定向广播，即在网络号指定的网络中广播（仅用作目的地址）
  - 32位全1的地址：本地广播地址，表示仅在发送节点所在的网络中广播（仅用作目的地址）
  - 32位全0的地址：指示本机（仅用作源地址）
  - 网络号为0、主机号有效的地址：指代本网中的主机
  - 形如127.xx.yy.xx的地址：保留作为回路测试，发送到这个地址的分组不输出到线路上，而是送回内部的接收端。

网络数量与地址数量

- A、B、C类地址对应不同规模的网络
- 一个A类、B类及C类地址可提供的接口地址数：
  - A类地址：$2^{24}-2 = 16777214$
  - B类地址：$2^{16}-2 = 65534$
  - C类地址：$2^{8}-2 = 254$
- A类、B类、C类地址的个数：
  - A类地址：$2^{7}-2 = 126$
  - B类地址：$2^{14}-2 = 16382$
  - C类地址：$2^{21}-2 = 2097152$

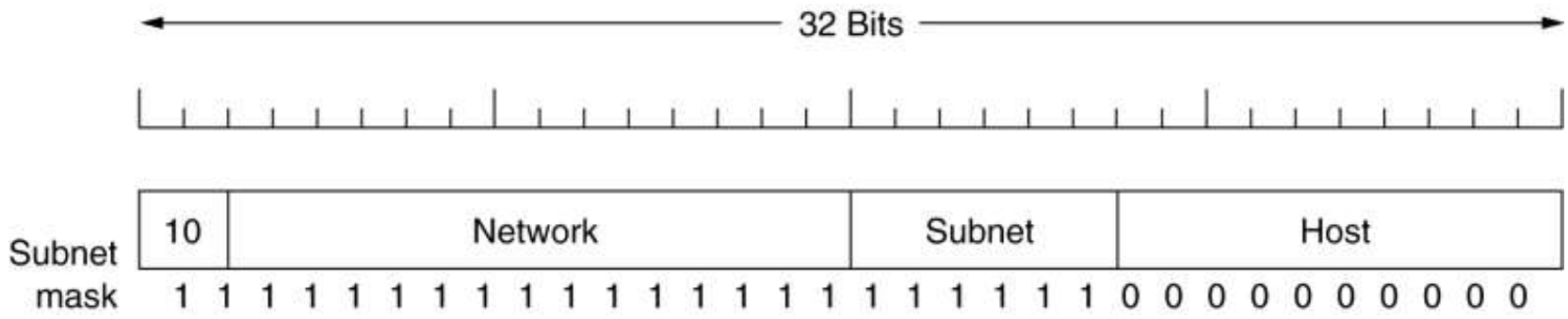
2. 子网（subnet）

- 管理员通常利用路由器，将一个较大的网络划分成若干较小的网络（子网），每个网络使用一部分地址空间



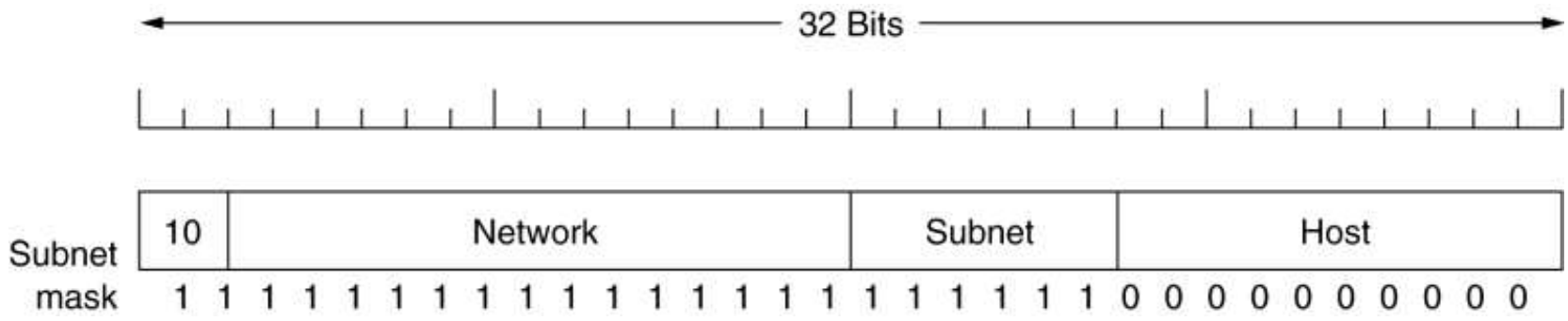
子网编址

- 从概念上说，引入子网仅略微改变了IP地址的解释：
  - 主机号被进一步划分成子网号和主机号两部分
  - 子网号标识网络内的一个子网，主机号标识子网中的一个网络接口
- 管理员可以根据需要，如子网的数量和规模，选择合适的子网号长度



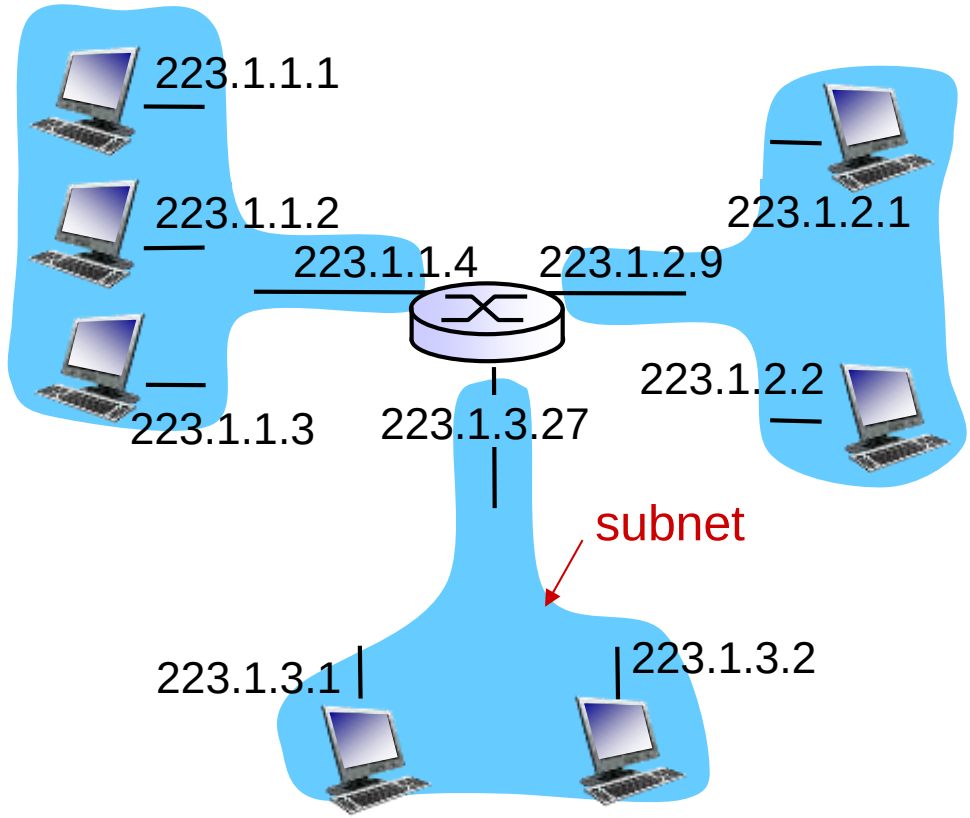
子网掩码（subnet mask）

- 子网掩码用于指示IP地址中子网号与主机号的边界：
  - 子网掩码是一个32比特的数，其中对应主机号的比特为0，其余比特为1
  - 子网掩码也采用点分十进制表示，如255.255.252.0
- 如何从IP地址中获取子网地址？
  - 将IP地址与子网掩码做“与”运算
  - 例如：128.10.1.1 AND 255.255.255.0 = 128.10.1.0
  - 注意：子网地址 ≠ 子网号，子网地址包括主机号之前的所有比特



子网：更确切的含义

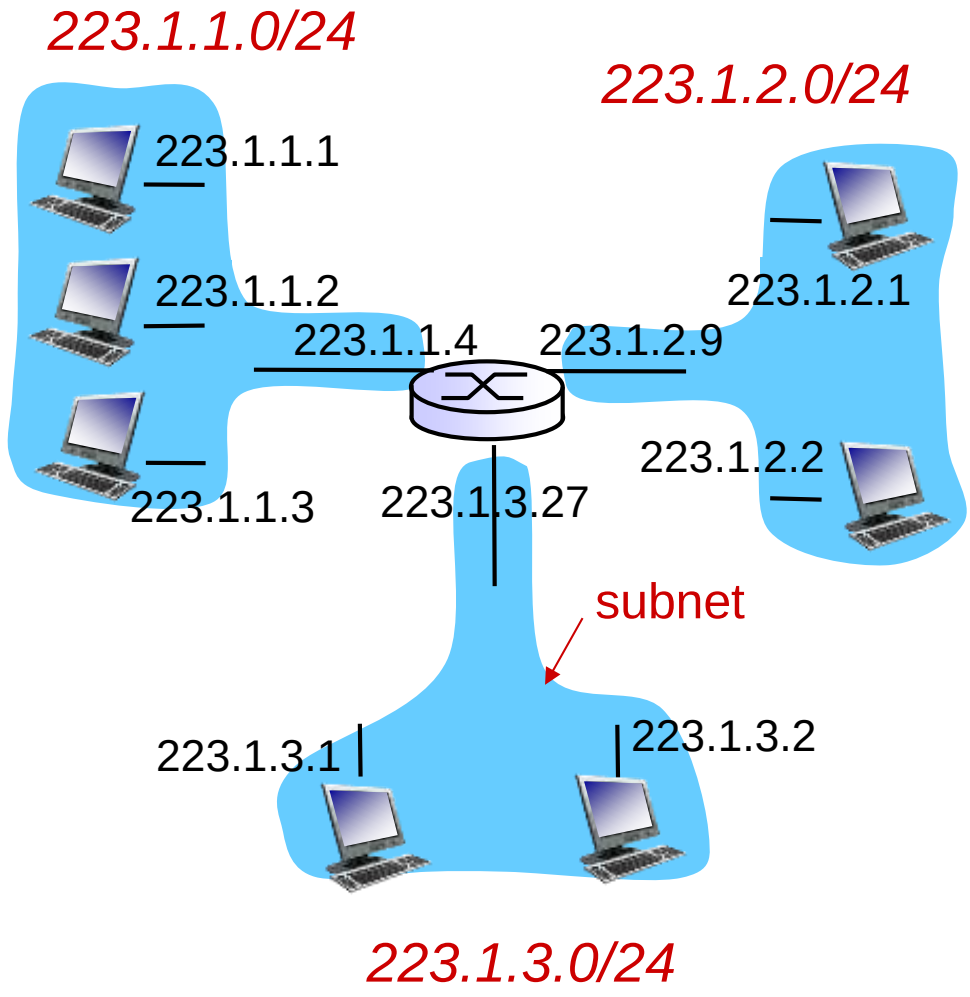
- 子网掩码将IP地址划分为两部分：
  - 子网地址：对应子网掩码中“1”的部分
  - 主机地址：对应子网掩码中“0”的部分
- 子网是什么？
  - 具有相同子网地址、且不需要通过路由器就可以相互到达的网络接口构成一个子网



network consisting of 3 subnets

如何确定子网？

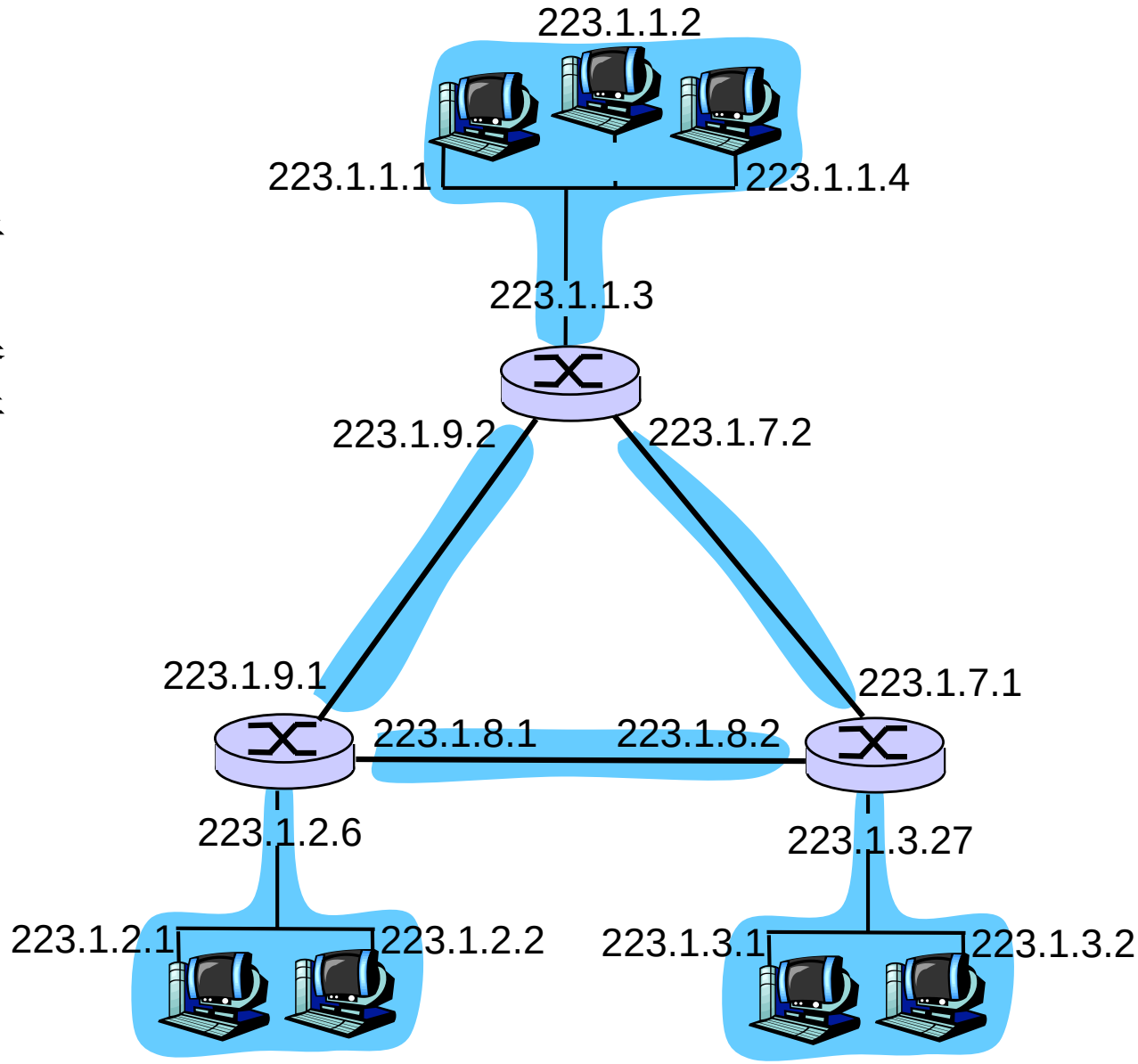
- 将网络接口与主机/路由器分开，形成一些分离的网络岛
- 每个网络岛就是一个子网



subnet mask: /24

举例

- 图示系统有6个子网：
  - 每个子网具有相同的子网地址
  - 子网内部不包含路由器
  - 子网之间被路由器隔开
- **重要的观察：**
  - **路由器的每个端口连接一个子网，不同的端口连接不同的子网**
  - 路由器是在子网之间转发数据包的设备
  - 子网内部通信不需要通过路由器，子网之间通信必须通过路由器



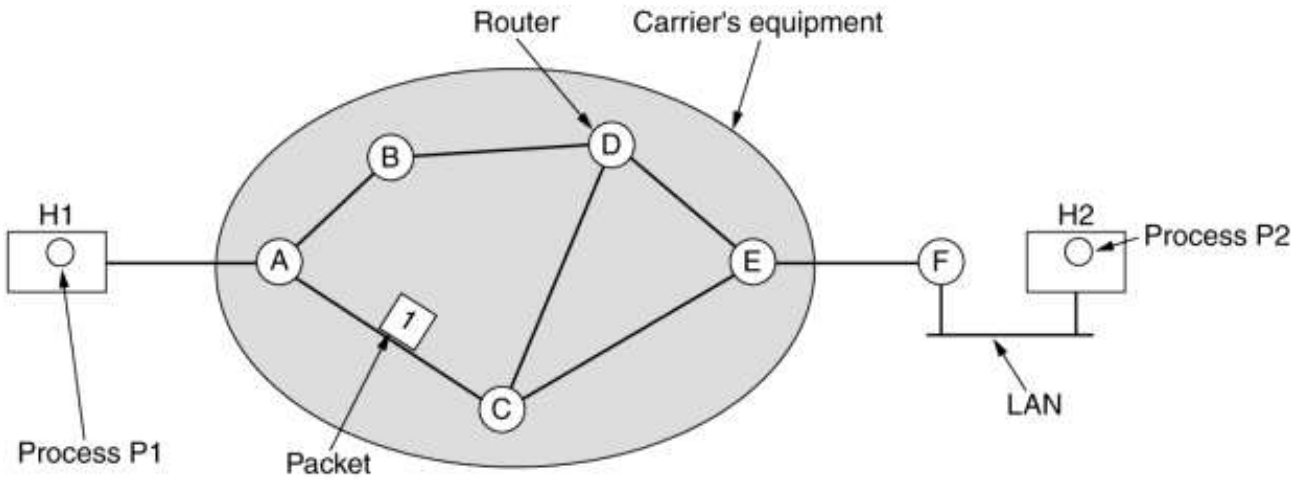
要求理解的概念

- 子网：
  - 子网内部的接口具有相同的子网地址
  - 子网内部的通信不需要经过路由器
  - 子网之间通信一定要经过路由器

- 路由器：
  - 在子网之间转发分组的设备，负责将分组从一个子网转发至另一个子网

3. IP数据报转发

- 网络层转发数据报的两种情形：
  - 直接交付（direct delivery）：节点将数据包直接发送给目的主机（不需要其它路由器转发）
  - 间接交付（indirect delivery）：节点将数据包转发给一个路由器去处理



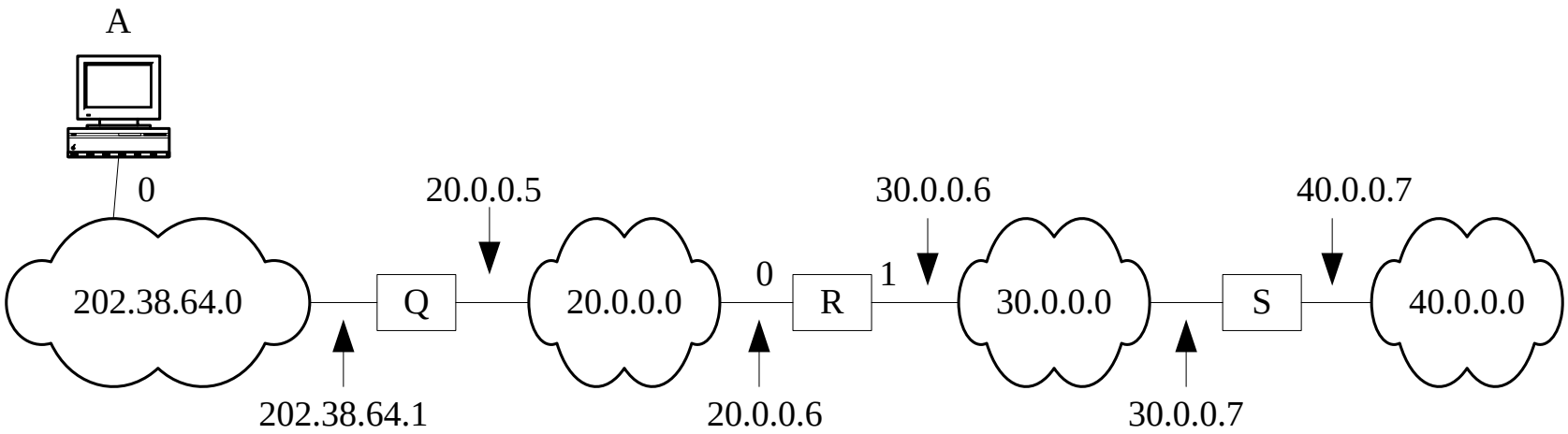
直接交付和间接交付

- 如何判断使用直接交付还是间接交付？
  - 直接交付：数据包的目的地址与本节点的某一端口在同一个子网中
  - 间接交付：数据包的目的地址不与本节点的任何一个端口在同一个子网中
- 直接交付的实现：
  - Chapter 6
- 间接交付的实现：
  - 节点查找转发表，将数据包发送给下一个路由器

转发表

- 转发表记录目的地址到输出端口的映射
- 取决于目的地址类型的不同，有三类转发表项：
  - 目的地址是一个子网地址：地址前缀表项
  - 目的地址是一个特定的网络接口地址：特定主机表项
  - 缺省项：不匹配所有其它表项的地址都被映射到一个默认的路由器端口
- IP采用逐跳选路：
  - 每个转发表项只记录去往目的地址的下一跳信息（下一个要到达的路由器端口），而不是一条完整的端到端路由
- 每个转发表项包括：
  - 目的地址/掩码、下一跳地址、输出端口等
  - 下一跳地址必须与输出端口在同一个子网中（不需要通过其它路由器就可以直接到达）

转发表的例子



R的路由表

目的地址	掩码	下一跳	端口
20.0.0.0	255.0.0.0	直接交付	0
30.0.0.0	255.0.0.0	直接交付	1
202.38.64.0	255.255.255.0	20.0.0.5	0
40.0.0.0	255.0.0.0	30.0.0.7	1

A的路由表

目的地址	掩码	下一跳	端口
202.38.64.0	255.255.255.0	直接交付	0
default	0.0.0.0	202.38.64.1	0

IP数据报的转发过程（主机/路由器）

从数据报中提取目的IP地址D，根据地址类别得到网络地址N；

if D与自己的任何一个IP地址匹配　　//本节点是数据报的目的节点

　　then 将数据包交给protocol域指定的协议实体处理

else if N与自己的任何一个直连网络的地址匹配　//直接交付

　　then 通过该直连网络把数据包直接交付到目的节点D

else if 表中包含到D的特定主机表项　//间接交付

　　then 把数据包发送到表中指定的下一跳

else if 表中包含到N的一个地址前缀表项　//间接交付

　　then 把数据包发送到表中指定的下一跳

else if 表中包含一个缺省项　　//间接交付

　　then 把数据包发送到表中指定的默认路由器端口

else 宣告选路出错，向数据包的源地址发送一条错误报告消息（ICMP）

要求掌握的概念

- 直接交付：
  - 不经过路由器就能到达（在同一个子网内）
- 间接交付：
  - 需要路由器转发才能到达（不在同一个子网内）
- 逐跳转发：
  - 每次只转发到下一跳，下一跳必须是直接可达的（在同一个子网内）
- 如何依据转发表进行转发决策

4. CIDR: Classless InterDomain Routing

- 分类编址的缺点：
  - 只能按照三种固定的大小分配地址空间，地址浪费严重（尤其是A类、B类地址）
  - 转发表必须记录每个已分配的网络，转发表规模爆炸式增长（C类地址网络非常多）
- CIDR：
  - 按照实际需要的地址数量分配地址空间，提高地址使用效率
  - 允许将若干条转发表项进行聚合，减小转发表规模

按照实际需要分配地址

- 举例：
  - 若一个网络需要2000个地址，可为其分配一个具有2048个连续地址的地址块；这些地址的前21位必须相同，从而可将其看成是一个具有21位子网地址的网络
- CIDR地址分配的原则：
  - 地址块的长度 L 必须是 2 的幂次
  - 所有地址的前（$32-\log_2 L$）位必须相同
- 网络地址的表示方法：
  - 用掩码指示网络地址的长度，如194.24.0.0 ， 255.255.248.0
  - 用“/长度”指示子网地址的长度，如194.24.0.0/21

机构如何获得网络地址？

- 机构通常从ISP的地址空间中分配地址
- 假设：ISP有一块地址200.23.16.0/20（共4096个地址），8个机构向该ISP申请地址，每个申请512个地址

ISP's block	11001000 00010111 00010000	00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000	00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010	00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100	00000000	200.23.20.0/23
...	.....	....	....
Organization 7	11001000 00010111 00011110	00000000	200.23.30.0/23

CIDR地址分配的另一个例子

- ISP有一块从194.24.0.0/16 开始的地址块
  - 剑桥大学申请2048个地址
  - 牛津大学申请4096个地址
  - 爱丁堡大学申请1024个地址

University	First address	Last address	How many	Written as
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Available)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

地址分配过程

剑桥大学申请2048个地址

ISP's block	11000000 00011000	00000000 00000000	194.24.0.0/16
Cambridge	11000000 00011000	00000000 00000000	194.24.0.0/21
(Available)	11000000 00011000	00001000 00000000	194.24.8.0/21

牛津大学申请4096个地址

ISP's block	11000000 00011000	00000000 00000000	194.24.0.0/16
Cambridge	11000000 00011000	00000000 00000000	194.24.0.0/21
(Available)	11000000 00011000	00001000 00000000	194.24.8.0/21
Oxford	11000000 00011000	00010000 00000000	194.24.16.0/20

爱丁堡大学申请1024个地址

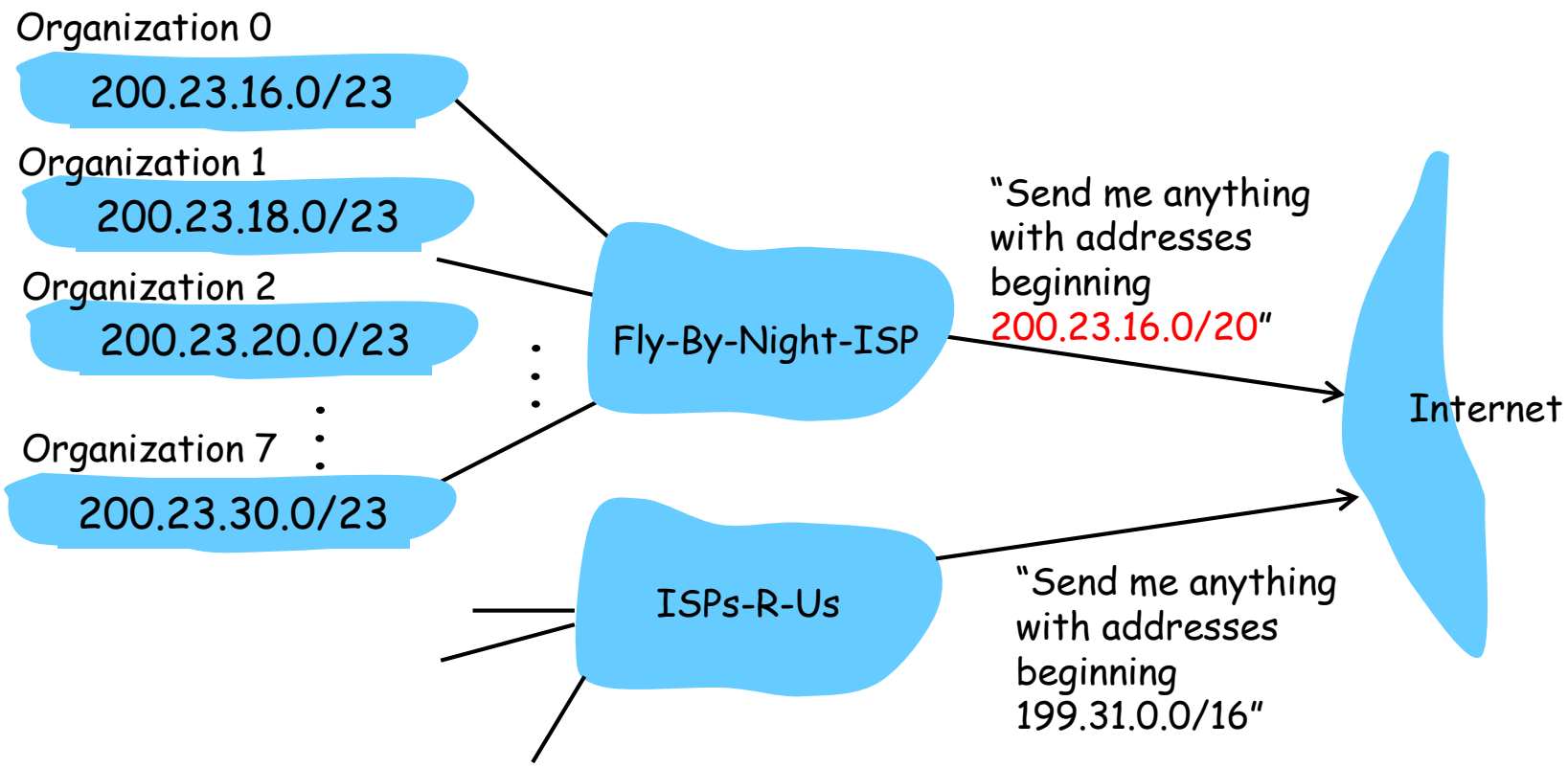
ISP's block	11000000 00011000	00000000 00000000	194.24.0.0/16
Cambridge	11000000 00011000	00000000 00000000	194.24.0.0/21
Edinburgh	11000000 00011000	00001000 00000000	194.24.8.0/22
(Available)	11000000 00011000	00001100 00000000	194.24.12.0/22
Oxford	11000000 00011000	00010000 00000000	194.24.16.0/20

更新转发表

- ISP在其转发表中添加三个表项：
  - 194.24.0.0/21 （掩码：255.255.248.0） // Cambridge
  - 194.24.8.0/22 （掩码：255.255.252.0） // Edinburgh
  - 194.24.16.0/20 （掩码：255.255.240.0）// Oxford
- 若路由器收到目的地址为194.24.17.4的数据包，其查表过程为：
  - 194.24.17.4 AND 255.255.248.0 = 194.24.16.0，与194.24.0.0不匹配
  - 194.24.17.4 AND 255.255.252.0 = 194.24.16.0，与194.24.8.0不匹配
  - 194.24.17.4 AND 255.255.240.0 = 194.24.16.0，与194.24.16.0匹配
    选择该转发表项

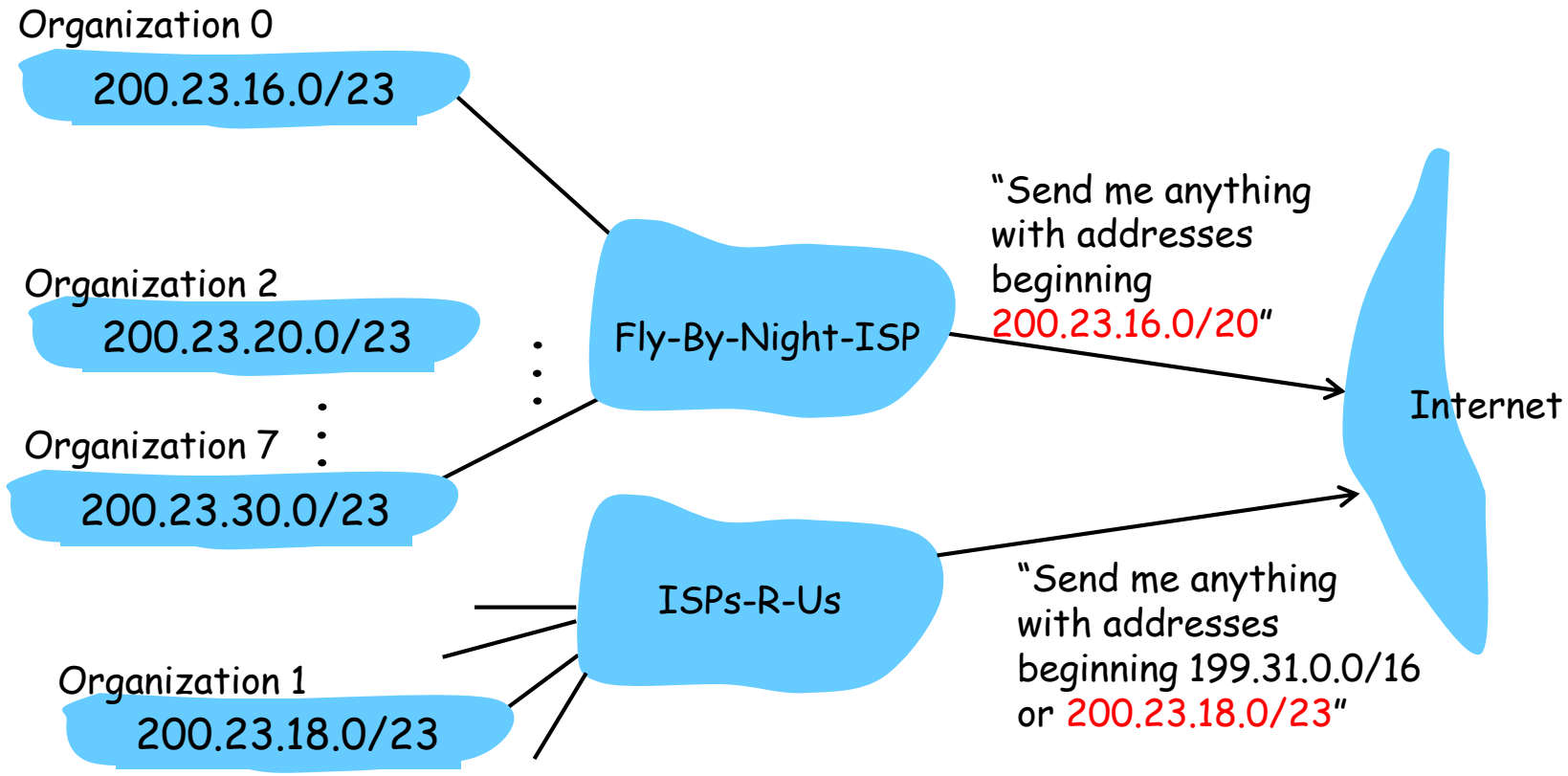
地址聚合

- 转发表中符合以下条件的若干个表项可以合并成一个表项：
  - 这些表项的目的地址可以聚合成一个前缀更短的地址
  - 这些表项使用相同的下一跳
- 地址聚合的过程可以递归进行



不能被聚合的转发表项

- 若个别表项不满足路由聚合的条件：
  - 仍然可以在转发表中给出一条聚合表项： 200.23.16.0/20
  - 同时给出不能被聚合的表项： 200.23.18.0/23
- **最长前缀匹配：**
  - 在所有匹配的路由表项中，选择前缀最长的表项



查找转发表：使用分类地址

- 转发表分为A、B、C三张表，分别记录A、B、C三类地址的转发表项，用哈希表组织

- 路由器收到数据报后：
  - 根据目的地址的类型确定要查找的转发表
  - 根据目的地址的类型提取网络地址
  - 用网络地址在相应的转发表中进行哈希查找（精确匹配）

采用CIDR后出现的问题

- 为与某个转发表项 Dest_addr/prefix_len进行匹配运算：
  - 路由器需要先从表项中读出地址掩码（或prefix-len值）
  - 计算包的目的地址前缀（用地址掩码和包的目的地址相与）
  - 与Dest_addr的地址前缀（Dest_addr与地址掩码相与）进行比较
- 引入的问题：
  - 地址前缀的长度prefix_len可以是任意值
  - Prefix_len无法从地址本身得到，只能从转发表项中得到
  - 必须从所有匹配的表项中选择前缀最长的表项
- 在大规模转发表中进行快速查找是一个难题（已经解决）

IP编址的小结

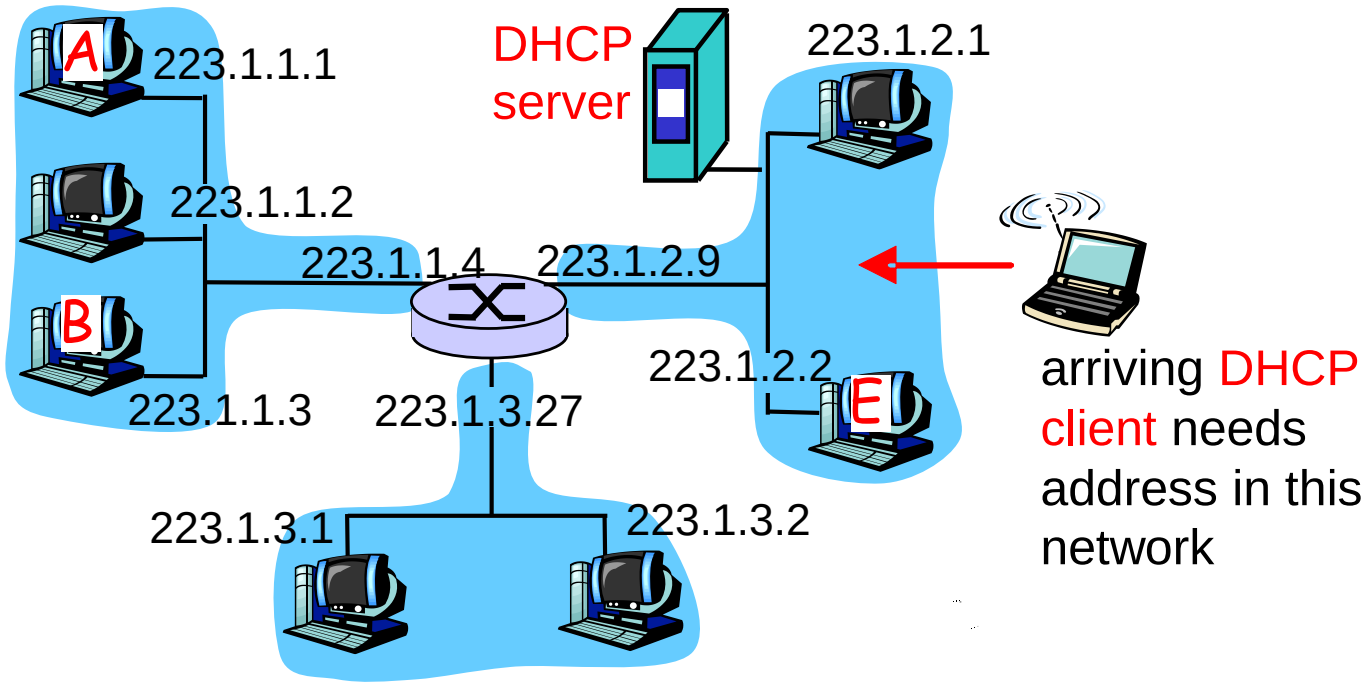
- 基于类的编址存在两个问题：
  - 地址空间浪费
  - 转发表空间爆炸
- CIDR解决了这两个问题：
  - 按需分配解决了地址空间浪费的问题
  - 地址聚合解决了转发表空间爆炸的问题
- CIDR引入的问题：
  - 地址聚合要求采用最长前缀匹配的原则查找转发表
  - 最长前缀匹配给转发表的快速查找带来了困难

5. 主机/路由器如何获得IP地址？

- 路由器：
  - 管理员手工配置路由器各个接口的IP地址
- 主机：
  - 管理员手工配置主机IP地址，服务器通常采用这种方法
  - 使用动态主机配置协议DHCP（Dynamic Host Configuration Protocol）获取IP地址、子网掩码、缺省路由器、本地DNS服务器等配置信息，个人终端通常采用这种方法
- 使用DHCP的好处：
  - 免去手工配置的麻烦（即插即用）
  - 可用少量的IP地址服务较多的客户（地址重用）

DHCP

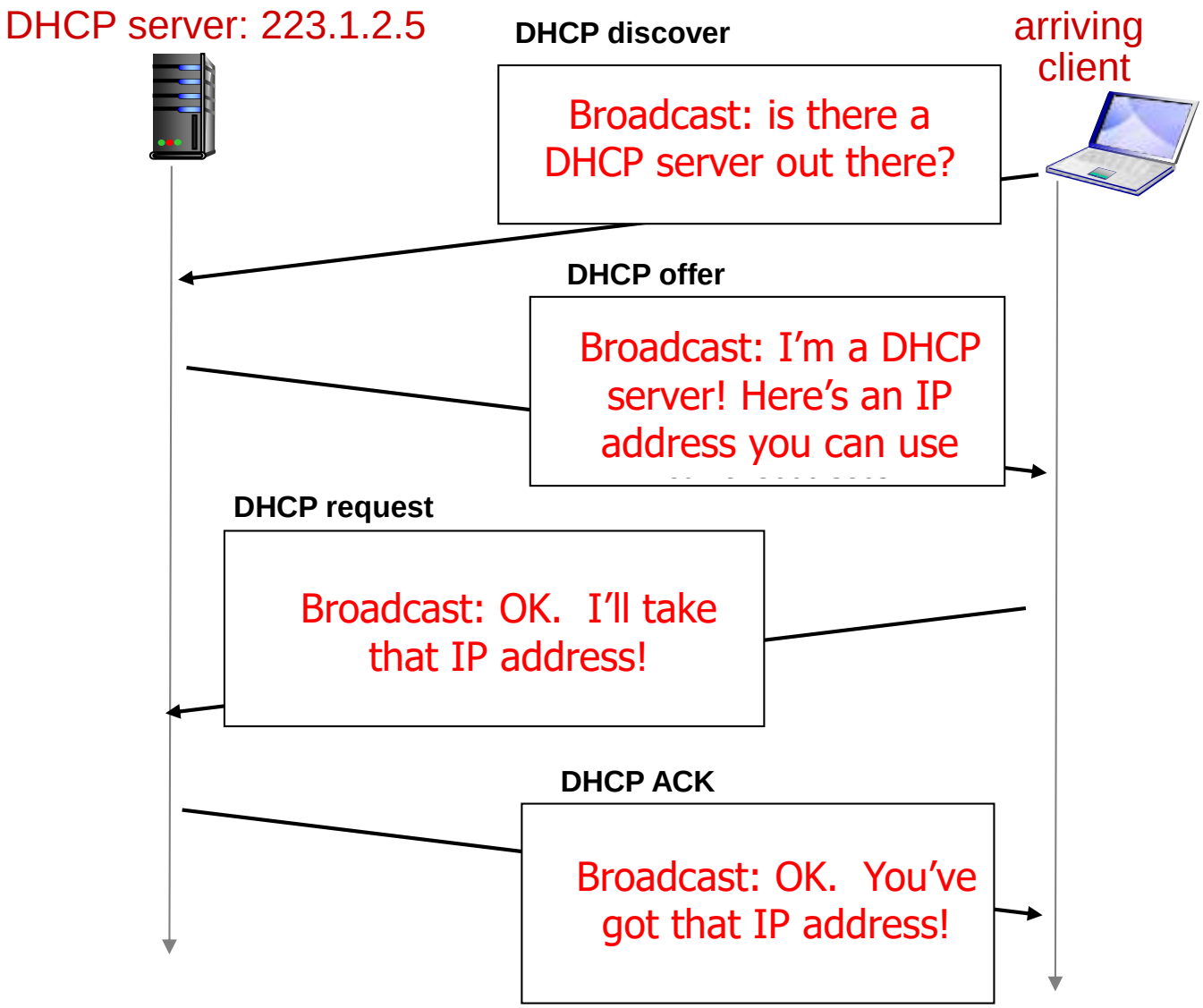
- 目标: 允许主机加入网络时自动获取配置信息
- DHCP是一个客户-服务器模式的应用协议
  - 每个子网中须有一个DHCP服务器，或者一个DHCP代理
  - 新到达的主机上运行DHCP client



DHCP概述

- 主机广播 "DHCP discover" 报文
  - 寻找子网中的DHCP服务器
- DHCP服务器用 "DHCP offer" 报文进行响应
  - 给出推荐的IP地址及租期、其它配置信息
- 主机用 "DHCP request" 报文请求IP地址
  - 主机选择一个DHCP服务器，向其请求IP地址
- DHCP服务器用"DHCP ack" 报文发送IP地址
  - 服务器响应客户的请求，确认所要求的参数
- DHCP服务器使用UDP端口67，客户使用UDP端口68

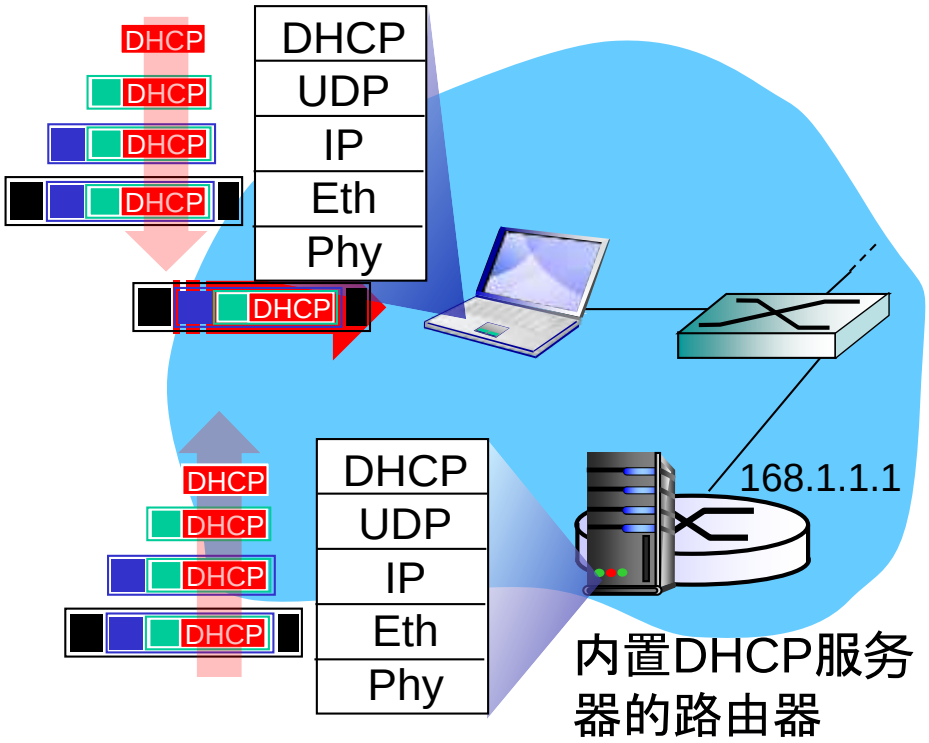
DHCP客户-服务器交互



DHCP: more than IP addresses

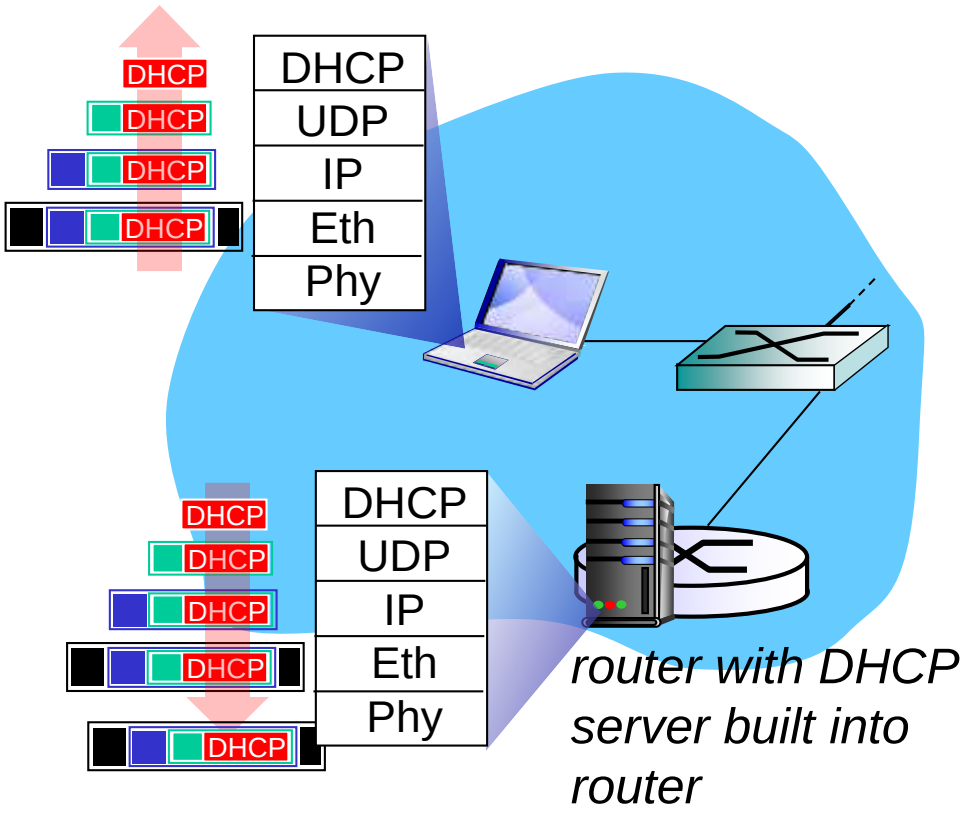
- DHCP不只返回分配的IP地址，还包括:
  - 第一跳路由器的IP地址（缺省网关、边缘路由器）
  - 本地DNS服务器的IP地址
  - 地址掩码

DHCP报文传输



- 入网的笔记本电脑需要本机地址、第一跳路由器地址、本地DNS服务器地址
- DHCP请求被封装到UDP/IP/以太帧中
- 以太帧在局域网中广播(dest: FFFFFFFFFFFF)，被运行了DHCP服务器的路由器收到
- DHCP请求被提取出来，送给DHCP服务器

DHCP报文传输



- DHCP服务器构造DHCP响应，包含客户的IP地址、客户第一跳路由器的IP地址、本地DNS服务器的IP地址

- DHCP响应经过封装、传输、解封装，到达DHCP客户

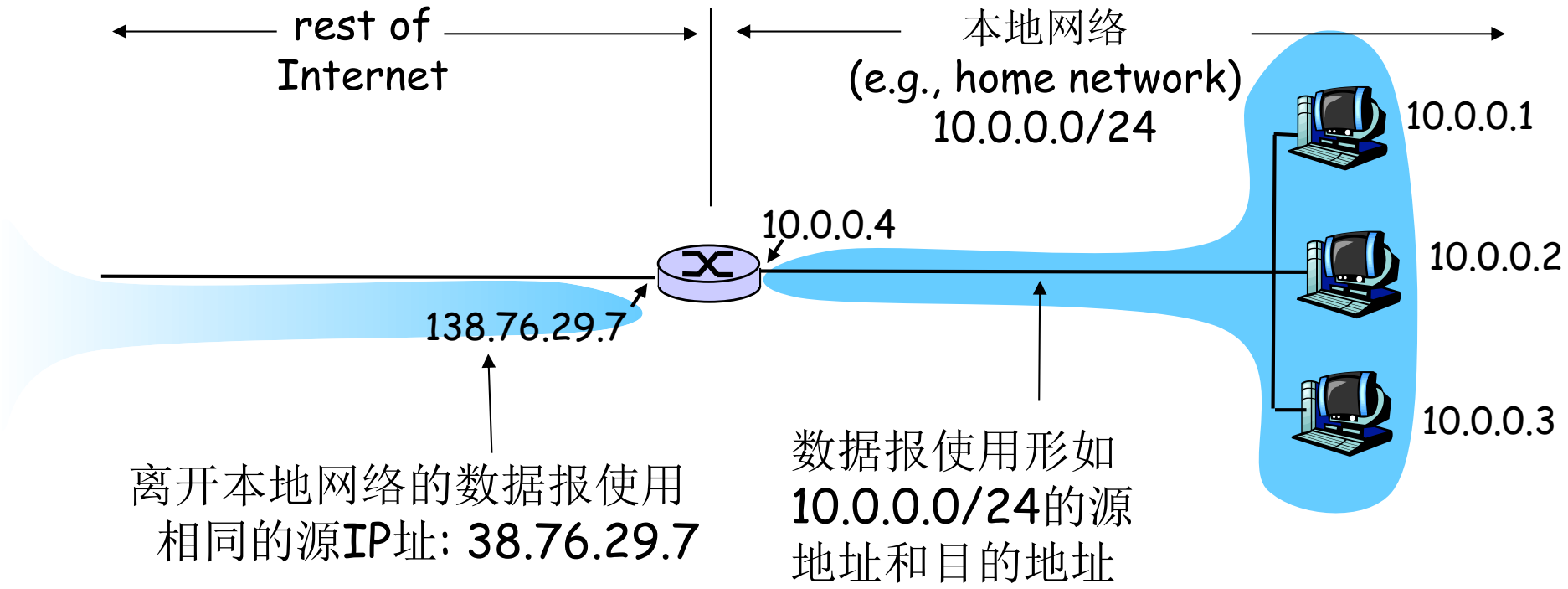
Chapter 4: Network Layer

- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

4.4 Generalized Forward and SDN

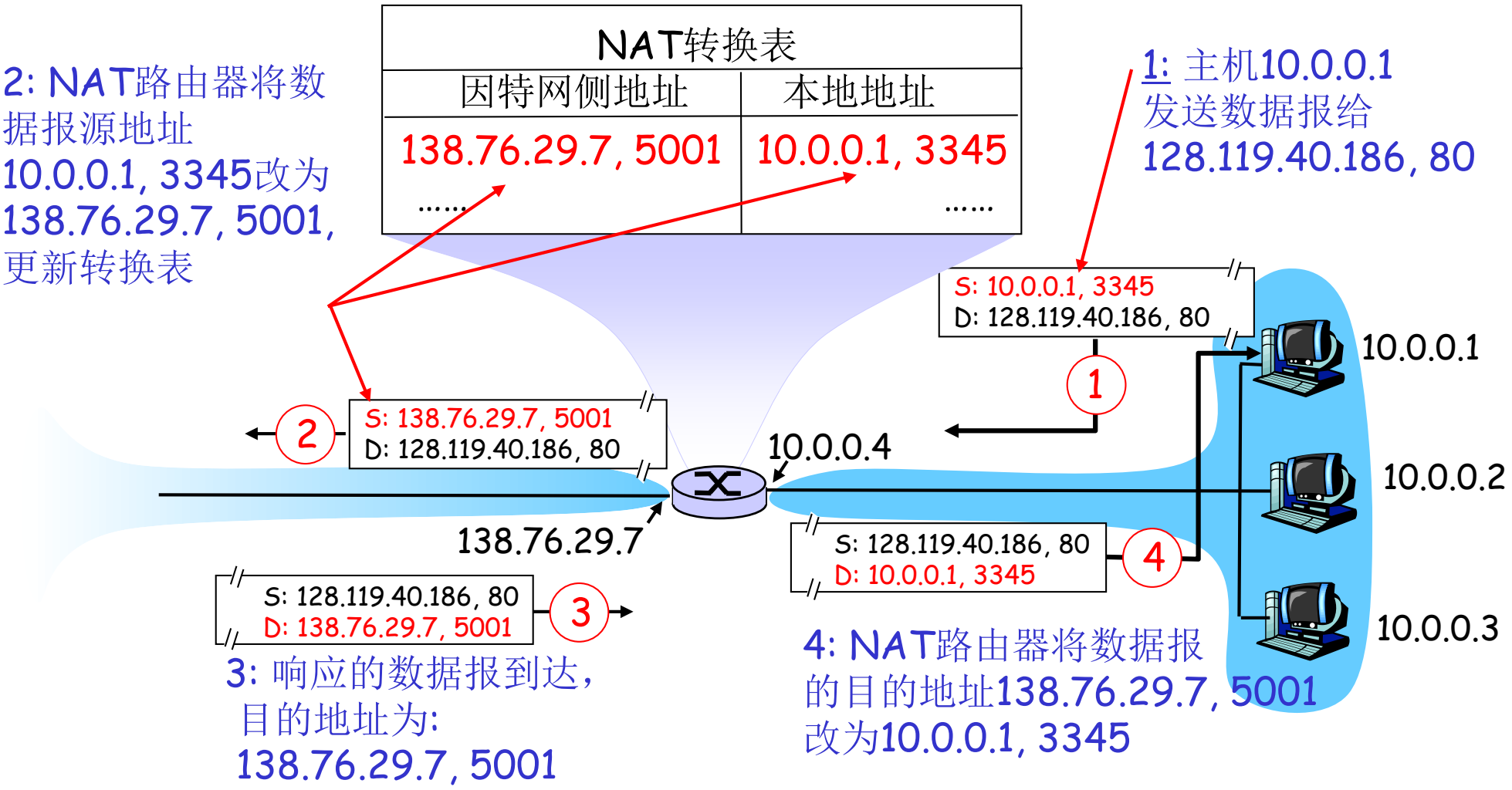
- match
- action
- OpenFlow examples of match-plus-action in action

网络地址转换（NAT）



- 动机：
  - 使用一个公用IP地址支持许多用户同时上网
  - 仅为公共可访问的节点分配公用IP地址（减少需要的公用IP地址数）
  - 网络内部节点对外是不可见的（安全考虑）

NAT举例


NAT转换表
因特网侧地址
本地地址
138.76.29.7, 5001
10.0.0.1, 3345
.....
.....
2: NAT路由器将数据报源地址10.0.0.1, 3345改为138.76.29.7, 5001, 更新转换表
1: 主机10.0.0.1发送数据报给128.119.40.186, 80
S: 10.0.0.1, 3345
D: 128.119.40.186, 80
1
10.0.0.1
10.0.0.2
10.0.0.3
10.0.0.4
S: 138.76.29.7, 5001
D: 128.119.40.186, 80
2
138.76.29.7
S: 128.119.40.186, 80
D: 10.0.0.1, 3345
4
S: 128.119.40.186, 80
D: 138.76.29.7, 5001
3
3: 响应的数据报到达，目的地址为: 138.76.29.7, 5001
4: NAT路由器将数据报的目的地址138.76.29.7, 5001改为10.0.0.1, 3345

NAT实现

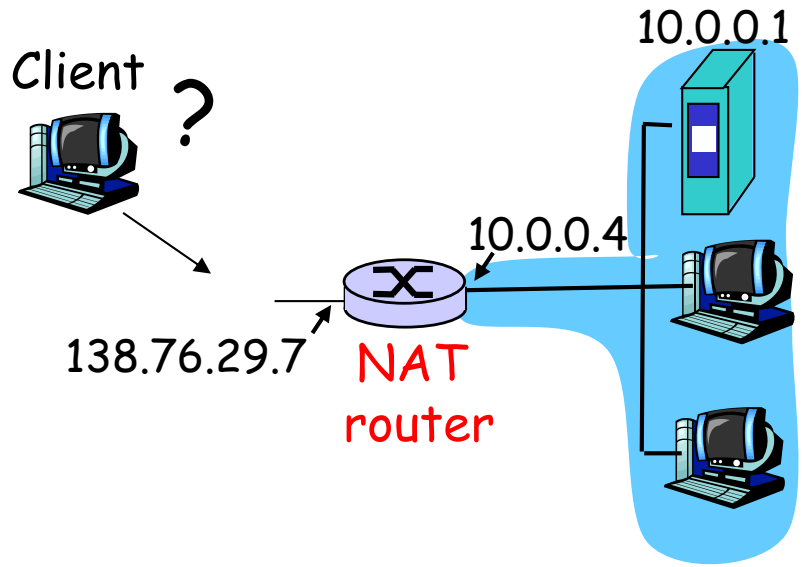
- 外出的数据报:
  - 将数据报中的（源IP地址，源端口号）替换为（NAT IP地址，NAT端口号）
- NAT 转换表:
  - 记录每个（源IP地址，源端口号）与（NAT IP地址，NAT端口号）的转换关系
- 进入的数据报:
  - 取出数据报中的（目的IP地址，目的端口号）查找NAT转换表，然后用转换表中对应的（IP地址，端口号）进行替换

NAT: Network Address Translation

- 16比特端口号:
  - 允许一个NAT IP地址同时支持65535个对外连接
- NAT的使用有争议:
  - 路由器应当只处理三层以下的包头（端口号在传输层）
  - 违反端到端原则（节点介入修改IP地址和端口号）
- NAT妨碍P2P应用:
  - NAT只允许内部主动发起的通信（位于NAT后面的主机对外是不可见的）
  - 但P2P应用要求任何对等方可以向任何其它（参与的）对等方发起通信

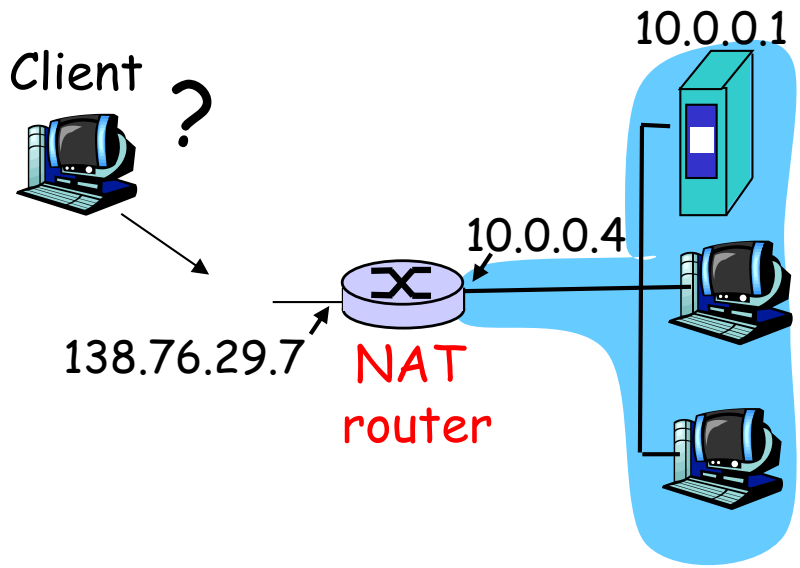
NAT 穿越问题

- Client希望连接服务器10.0.0.1，但是
  - 地址 10.0.0.1 为内部地址，对于client 不可见
  - Client 只能看到NAT路由器的公共IP地址138.76.29.7



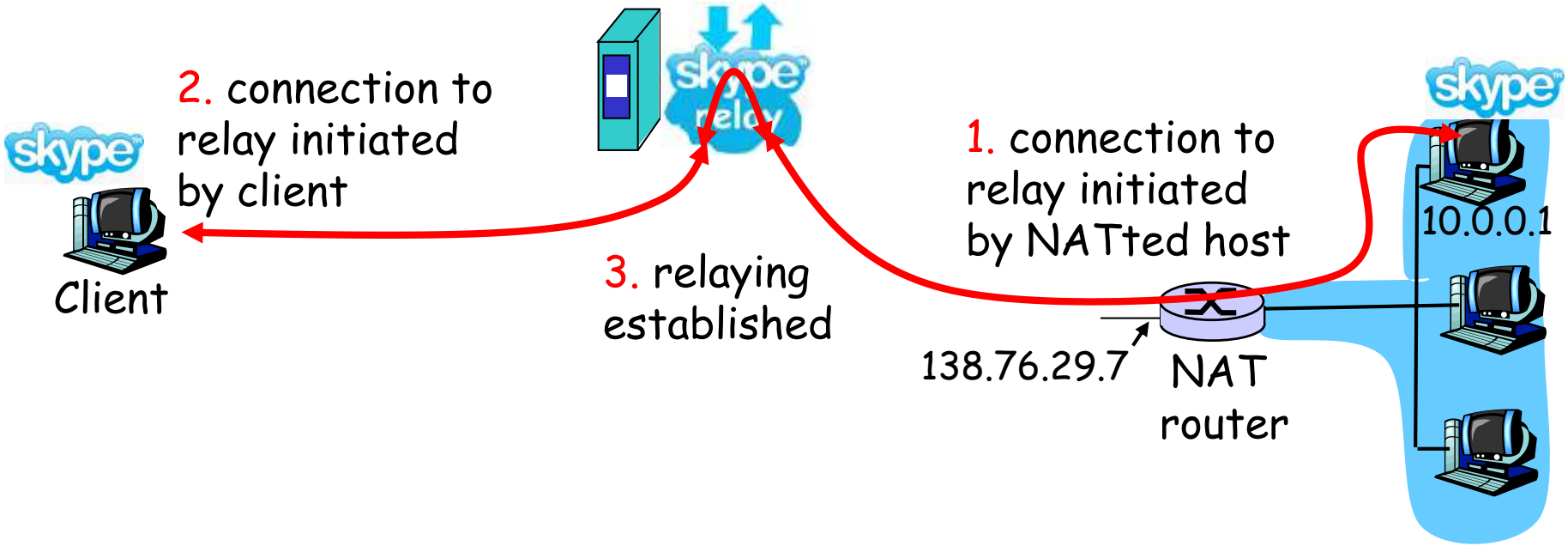
使用UPnP实现NAT穿越

- 假设主机10.0.0.1在端口3345上运行一个BT程序：
  - BT程序请求NAT产生一个“洞”，将 <10.0.0.1, 3345>映射到 <138.76.29.7, 5001>上
  - BT程序向追踪器通告它在 <138.76.29.7, 5001>上可用
  - 其它主机通过追踪器可以看到该主机，并能向 <138.76.29.7, 5001>发起TCP连接
  - NAT将< 138.76.29.7, 5001>上收到的SYN包转发给主机10.0.0.1



使用中继服务器实现NAT穿越

- 在 Skype 中使用：
  - NAT 后面的服务器与中继器建立连接
  - 外部客户与中继器建立连接
  - 中继器在两个连接之间转发分组



重要知识点

- 子网、路由器与数据报转发：
  - 子网内部通信不需要经过路由器，子网之间通信必须经过路由器
  - 子网内部可直接交付，跨子网需间接交付
  - 逐跳转发：下一跳必须直接可达
- CIDR：
  - 地址分配，地址聚合，最长前缀匹配
- NAT的工作原理
- CIDR、DHCP、NAT均可以部分解决IP地址不足的问题，但角度不同

Chapter 4: Network Layer

- 4. 1 Introduction
- ** Virtual circuit and datagram networks
- 4.2 What's inside a router
- 4.3 IP: Internet Protocol
  - Datagram format
  - fragmentation
  - IPv4 addressing
  - Network address translation
  - IPv6

4.4 Generalized Forward and SDN

- match
- action
- OpenFlow examples of match-plus-action in action

IPv6

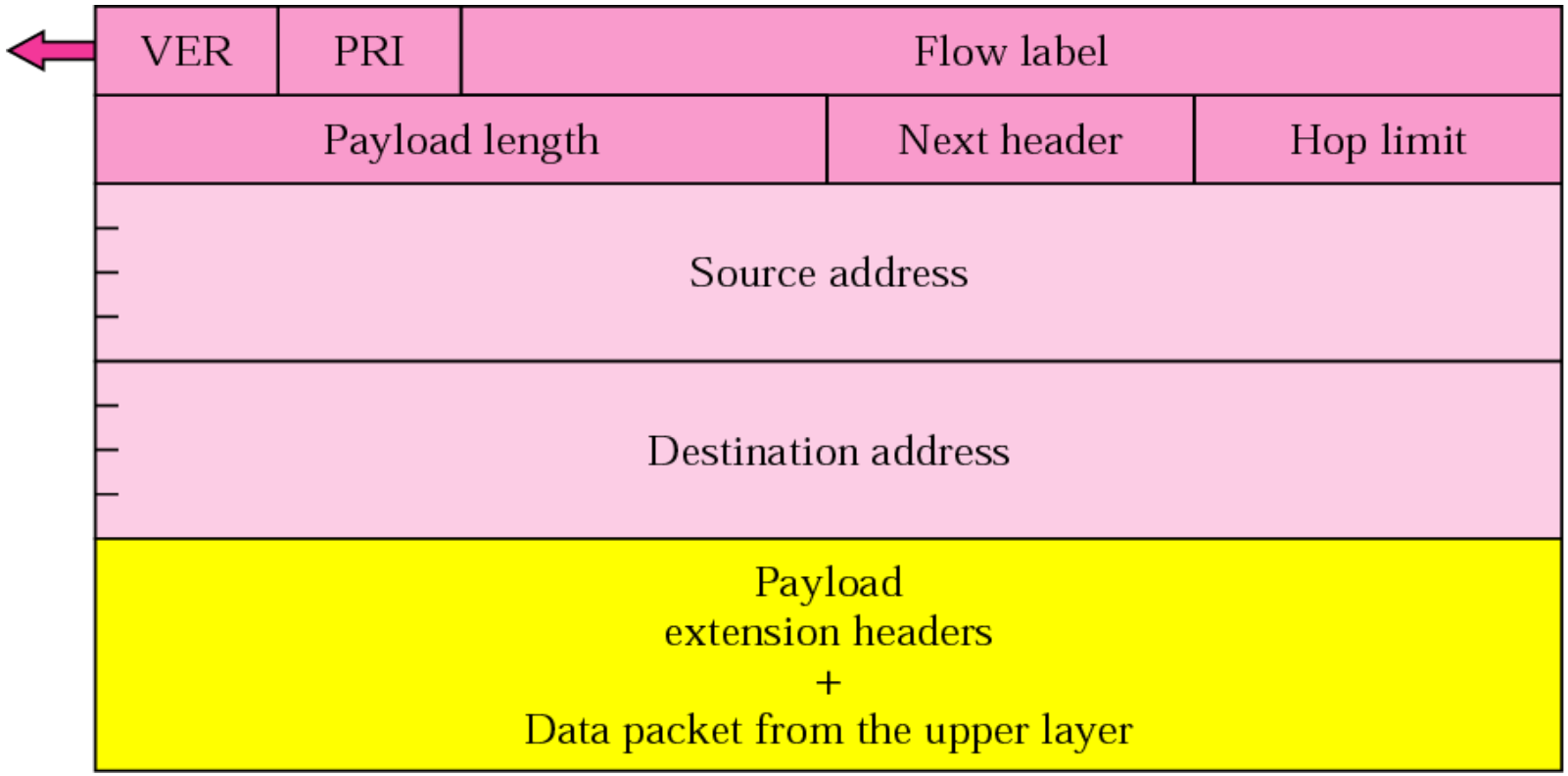
- 最初的动机：IPv4地址将很快耗尽
- 进一步的动机是借机改进IPv4的不足之处：
  - 简化头部格式，加快数据报处理和转发
  - 支持服务质量
  - 支持多播
  - 支持移动性
  - 增强安全性
  - ……
- IPv6与IPv4不兼容，但与其它所有因特网协议都兼容

IPv6地址

- 128位，使用冒号十六进制表示，每16位以十六进制的形式写成一组，组之间用冒号分隔，如 8000:0:0:0:0123:4567:89AB:CDEF
- 地址表示的零压缩技术：可将连续的多组0压缩为一对冒号，如以上地址可表示为：8000::0123:4567:89AB:CDEF
- IPv6定义了三种地址类型：
  - 单播地址：一个特定的网络接口
  - 多播地址：一组网络接口
  - 任播地址（anycast）：一组网络接口中的任意一个（通常是最近的一个）

IPv6数据报格式

- IPv6数据报以一个40字节的基本头开始，后面跟零个或多个扩展头，然后是数据



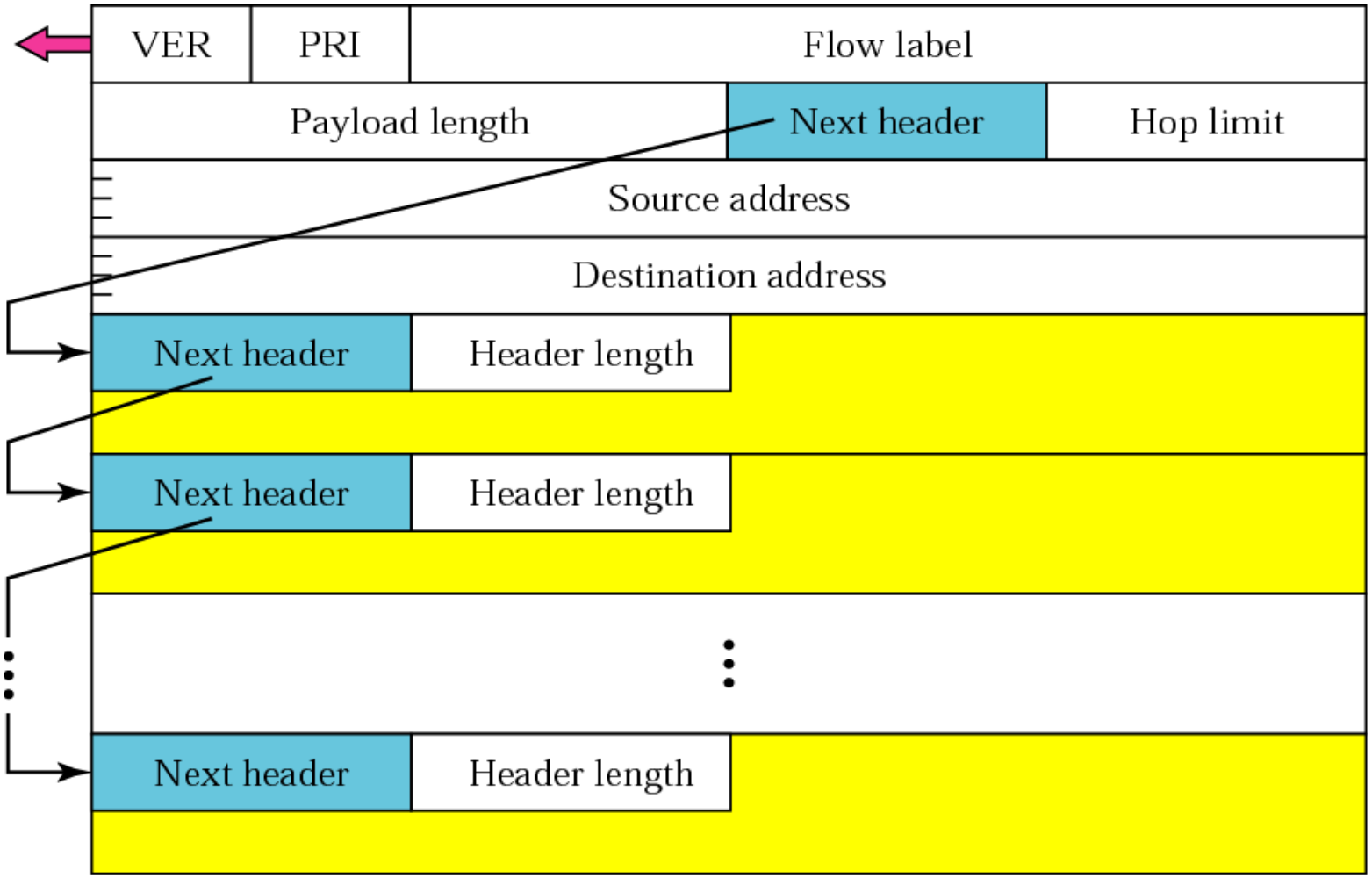
PRI（或traffic class）

- 作用：
  - 发送方在该域定义数据报的优先级
  - 路由器发现网络拥塞时，按优先级从低到高的顺序丢弃包
- IPv6将网络流量划分为两大类：
  - 受拥塞控制的流：
    - 非实时流属于这一类，优先级0～7，按照重要性及用户体验设定
  - 不受拥塞控制的流：
    - 实时多媒体流属于这一类，优先级8～15，尚无标准，可以按照用户要求的服务质量等级定义

流标签（flow label）

- 流标签：用于标明属于同一个流（flow）的数据报，但流的概念并没有明确地定义
- 一般来说，流是具有相同传输特性（如源/目的、优先级、选项等）、并要求相同处理（如使用相同的路径和资源、具有相同的服务质量要求等）的一系列数据包
- 流标签由发送方分配，<源地址，流标签>唯一标识一个流：
  - 路由器维护一张流表（flow table），记录每一个流需要的处理；收到数据包后，根据源地址和流标签查找流表，进行相应的处理
- **流标签的引入使得IPv6具备了对数据包进行区分处理的能力：**
  - 流标签的使用极大地降低了数据包分类的复杂度（多元包分类->二元包分类）

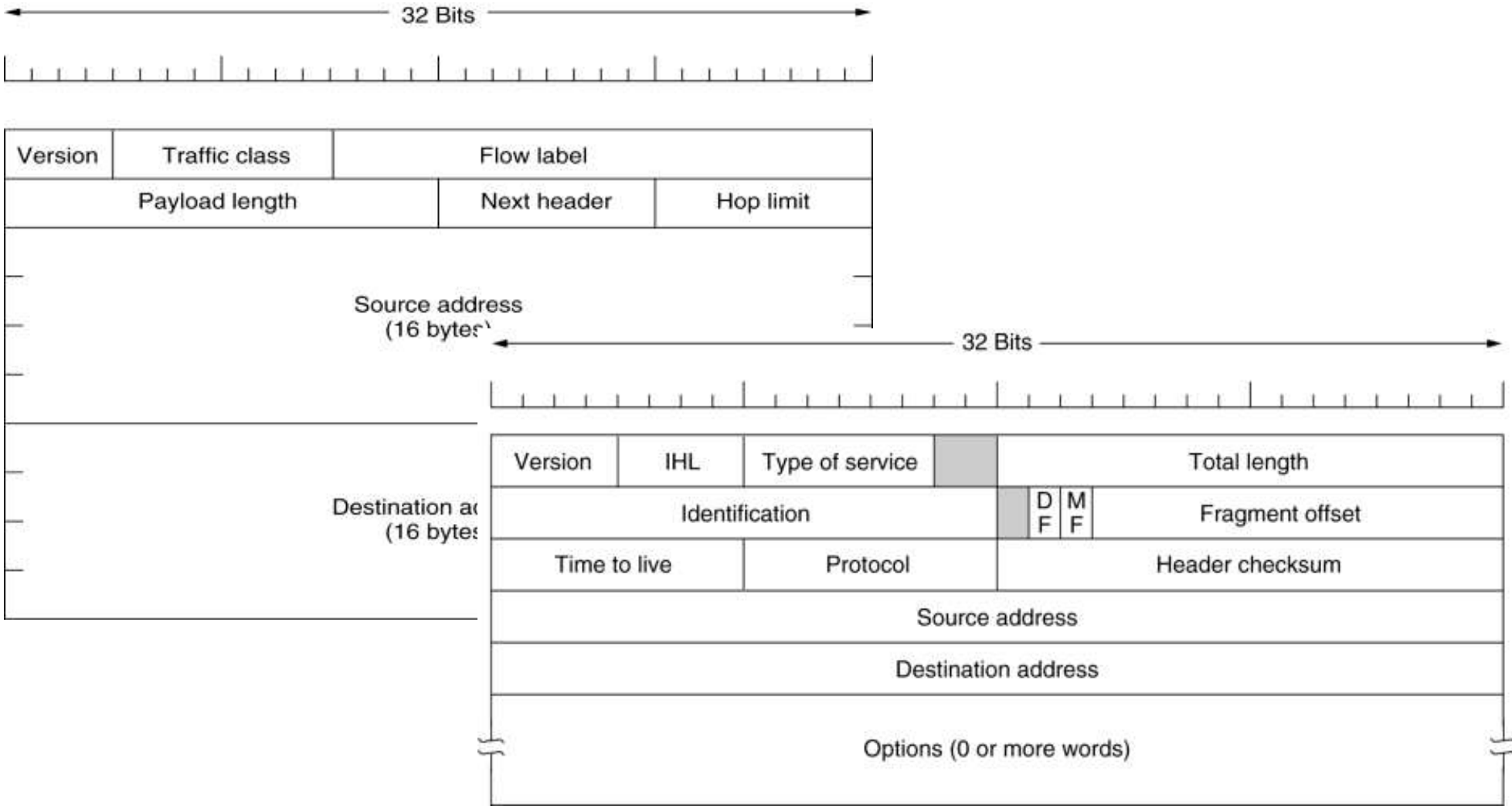
扩展头部


VER
PRI
Flow label
Payload length
Next header
Hop limit
Source address
Destination address
Next header
Header length
Next header
Header length
Next header
Header length

IPv6包格式

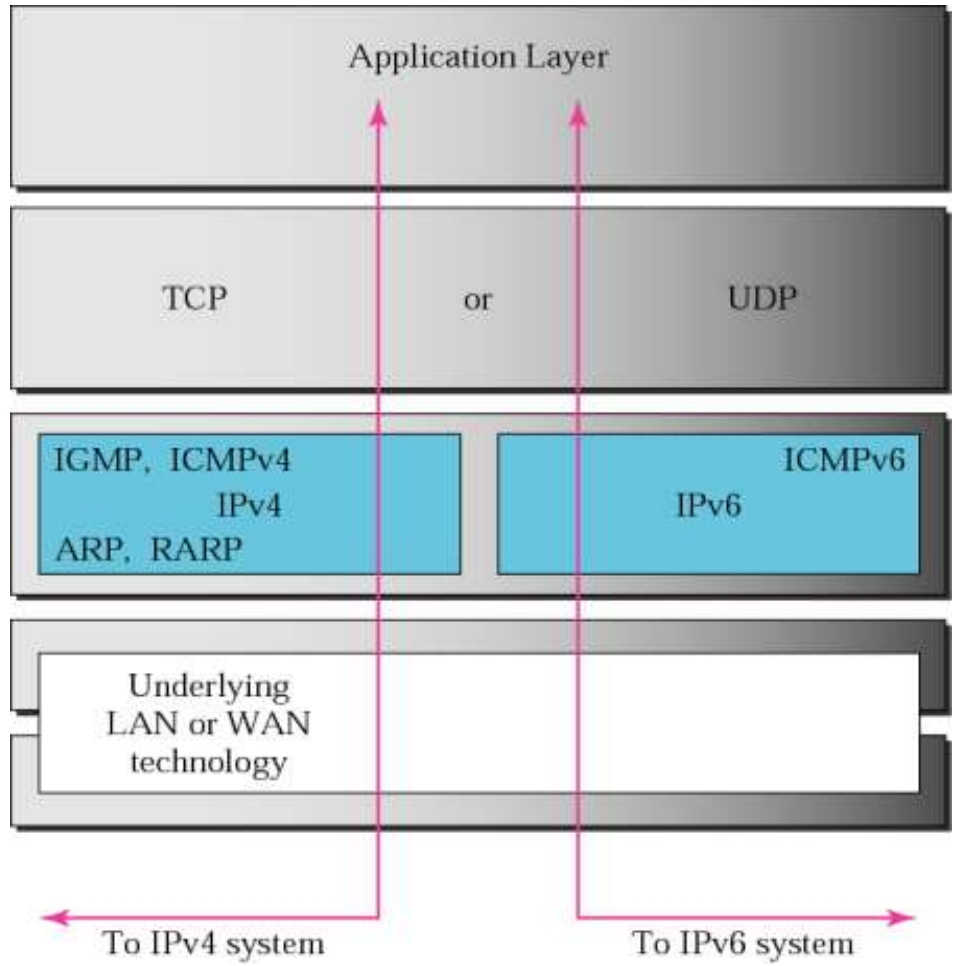
- 与IPv4固定头相比，IPv6的基本头中去掉了以下字段：
  - IHL：IPv6的基本头总是40字节长
  - 与分片相关的字段：IPv6路由器不负责分片
  - 头校验：计算校验和太花时间，现在的网络非常可靠，且链路层和传输层上往往都有差错检测
- IPv6基本头中增加了以下字段：
  - 流标签：支持对数据包区分处理
- IPv6基本头改变了以下字段的作用：
  - Type of Service：代之以Traffic Class
  - 总长度：代之以载荷长度
  - Protocol：代之以Next header，允许任意扩展选项

IPv6的基本头与IPv4的固定头



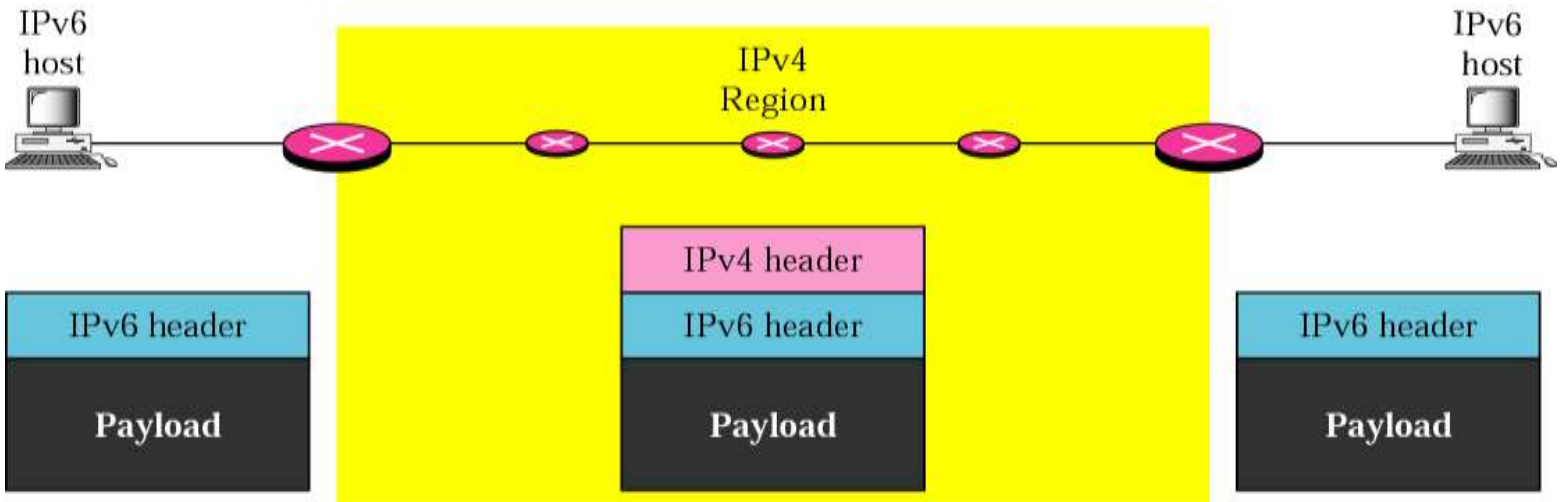
从IPv4过渡到IPv6

- 因特网不可能一夜之间升级到IPv6，IPv4与IPv6如何共存？
- 采用IPv4/IPv6双协议栈：
  - 支持IPv6的主机和路由器同时运行IPv4和IPv6
  - 运行双栈的源节点先对目的节点查询DNS：若DNS返回IPv4地址，发送IPv4分组；若返回IPv6地址，发送IPv6分组
  - 双栈节点同时拥有IPv4和IPv6地址

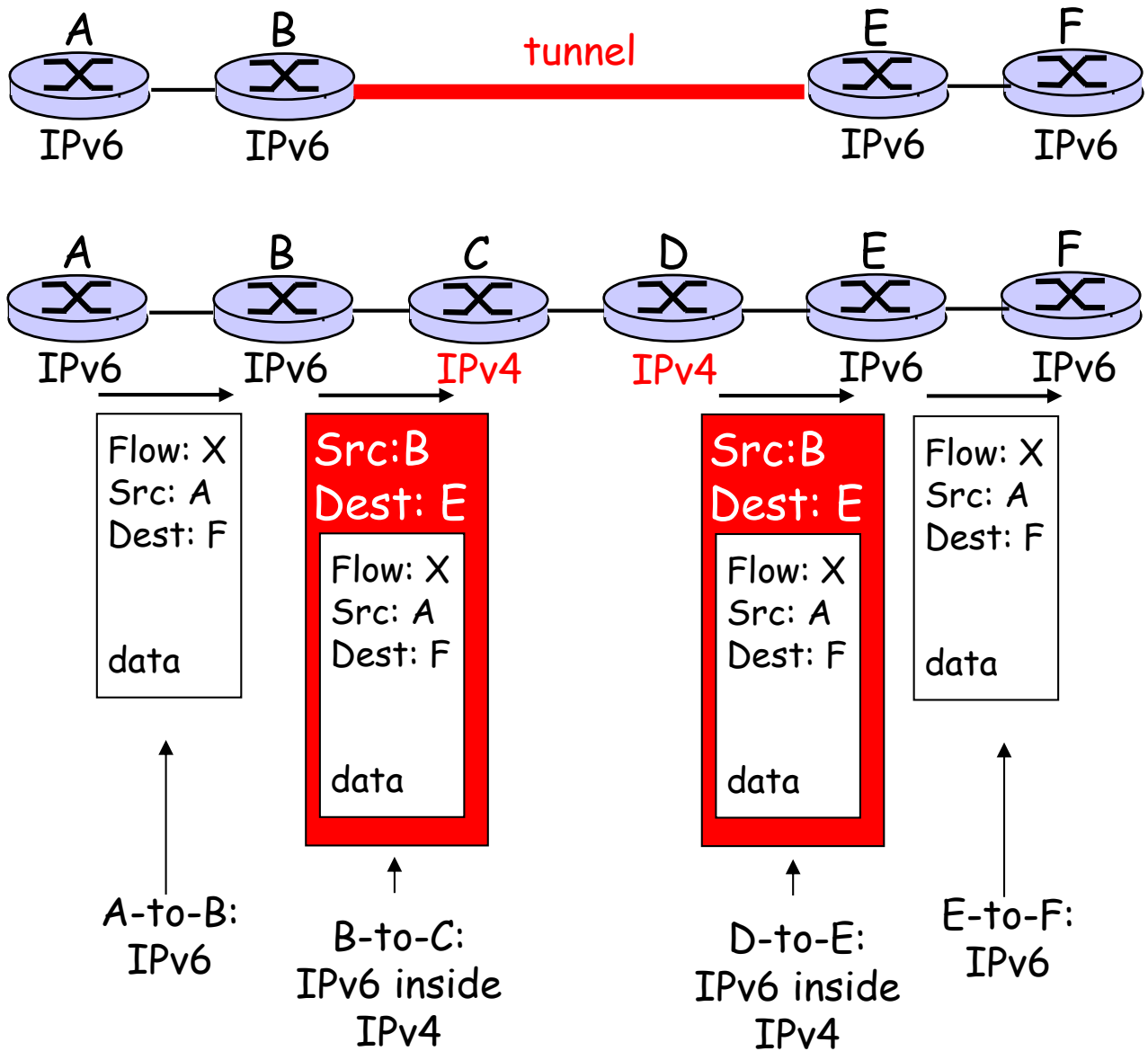


IPv6数据包如何穿越IPv4网络?

- 两台支持IPv6的主机中间隔着一个运行IPv4的网络，IPv6数据报如何穿越IPv4网络？
- 一种有效的方法是建立隧道（tunneling）：
  - IPv6/IPv4边界路由器将IPv6包封装到一个IPv4包中，送入IPv4网络，目的边界路由器取出IPv6包继续传输



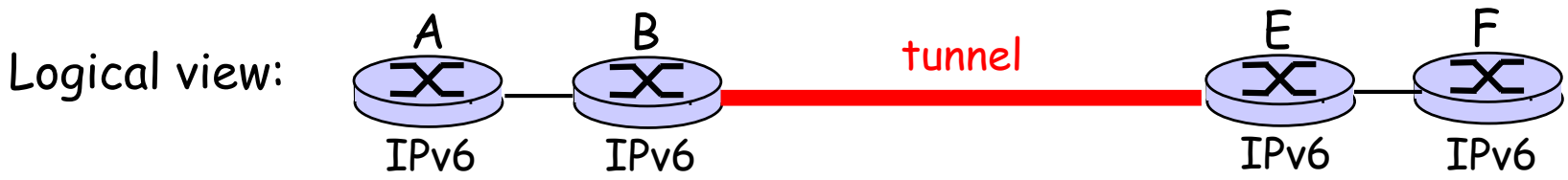
建立隧道的方法：封装数据包



外层IPv4数据报：

- Src IP =
  B的IPv4地址
- Dest IP =
  E的IPv4地址

隧道技术的要点



- 隧道技术的本质是封装数据包：
  - 当分组需要穿过一个不同协议的网络时，将分组封装在中间网络的分组中传输
  - 执行封装和解封装的路由器端口称为隧道的端点
- 为什么称为隧道？
  - 一个类比：IPv4路由器是检查站，每个经过的数据报都会被检查，但在隧道中传输的IPv6包逃避了检查
  - IPv6包在IPv4网络中传输时，没有被IPv4路由器检查和处理，特别是，报头中的TTL未减1
  - 在路由器E的IPv6看来，IPv6包是从路由器B直接到来的

重要知识点

- IPv4与IPv6不兼容
- 隧道技术的要点和应用
- 如何解决IPv4与IPv6共存：
  - 双协议栈 + 隧道技术

我国IPv6的发展现状

- IPv6网络高速公路已经建成：
  - 13个骨干网直联点全部实现IPv6互联互通
  - IPv6的国际出入口带宽（90Gbps）也已经开通
- 已建成规模最大的IPv6网络：
  - 网络基础设施全部支持IPv6，规模全球领先
  - 已申请的IPv6地址资源位居全球第一
  - 2020年覆盖50%的互联网用户
- 已建成业务形态最全的IPv6网络
  - 2018年底全面应用，从商业网站到政府公网全面支持IPv6
  - 数据中心、云产品、内容分发等应用基础设施，已初步具备全国全网IPv6的支持服务能力
- 2021-2025年为向以IPv6为核心的网络演进的阶段，下一代互联网将全面建成，IPv6流量在网络占主体

我国IPv6的发展现状

- 2015年6月开始实施“雪人计划”，面向全球招募了25个根服务器运营志愿单位，正式运营IPv6根服务器和域名解析系统
- 至2017年，已在全球部署了25台IPv6根服务器：
  - 3台主根服务器：分别位于中国、美国和日本
  - 22台辅根服务器：其中3台在中国
- 我国已初步形成端到端的下一代互联网体系，终结了美国主导30多年的互联网霸权
- 当今世界，传统互联网正进入IPv6时代，移动互联网正步入5G时代，我国在技术上完成了逆转，在规模上实现了超越，总体优势明显

Chapter 4: *done!*

4.1 Overview of Network layer: data plane and control plane

**Virtual circuit and datagram networks

4.2 What's inside a router

4.3 IP: Internet Protocol

- datagram format
- fragmentation
- IPv4 addressing
- NAT
- IPv6

4.4 Generalized Forward and SDN

- match plus action
- OpenFlow example

Question: 转发表是如何得到的？

Answer: 通过控制面（下一章）

作业

- 习题：11月19日
  - 5，7，8，10，12，14
- 实验：11月24日
  - IP

第二部分 端节点算法学

端节点算法学

- 端节点算法学：
  - 建立高速服务器的一组系统性技术，是网络算法学在端节点（尤其是服务器）上的运用
  - 节点间通信已成为大数据分析、深度学习等应用的主要瓶颈之一
  - 随着网络功能虚拟化的提出，将来数据中心绝大部分的网络设备都会在通用服务器上实现
- 端节点算法学研究如何减少以下开销：
  - 数据拷贝（chapter 5）
  - 控制转移（chapter 6）
  - 定时器（chapter 7）
  - 解复用（chapter 8）
  - 其它一般性协议处理任务 （chapter 9）

第五章 拷贝数据

消除不必要的拷贝（P1）

- 网络报文在收发和处理的过程中，通常会被拷贝多次
- 计算机中的数据拷贝消耗两个宝贵的资源：
  - 内存带宽：如果处理一个报文涉及k次拷贝，系统吞吐量可能降至1/k
  - 内存：如果一个报文在内存中被保存k份，有效内存容量降至1/k
- 本章关注如何消除不必要的拷贝：
  - 一个拷贝如果不是由硬件要求的，该拷贝是不必要的
- 本章还将讨论其它影响内存使用效率的操作

5.1 为什么要拷贝数据

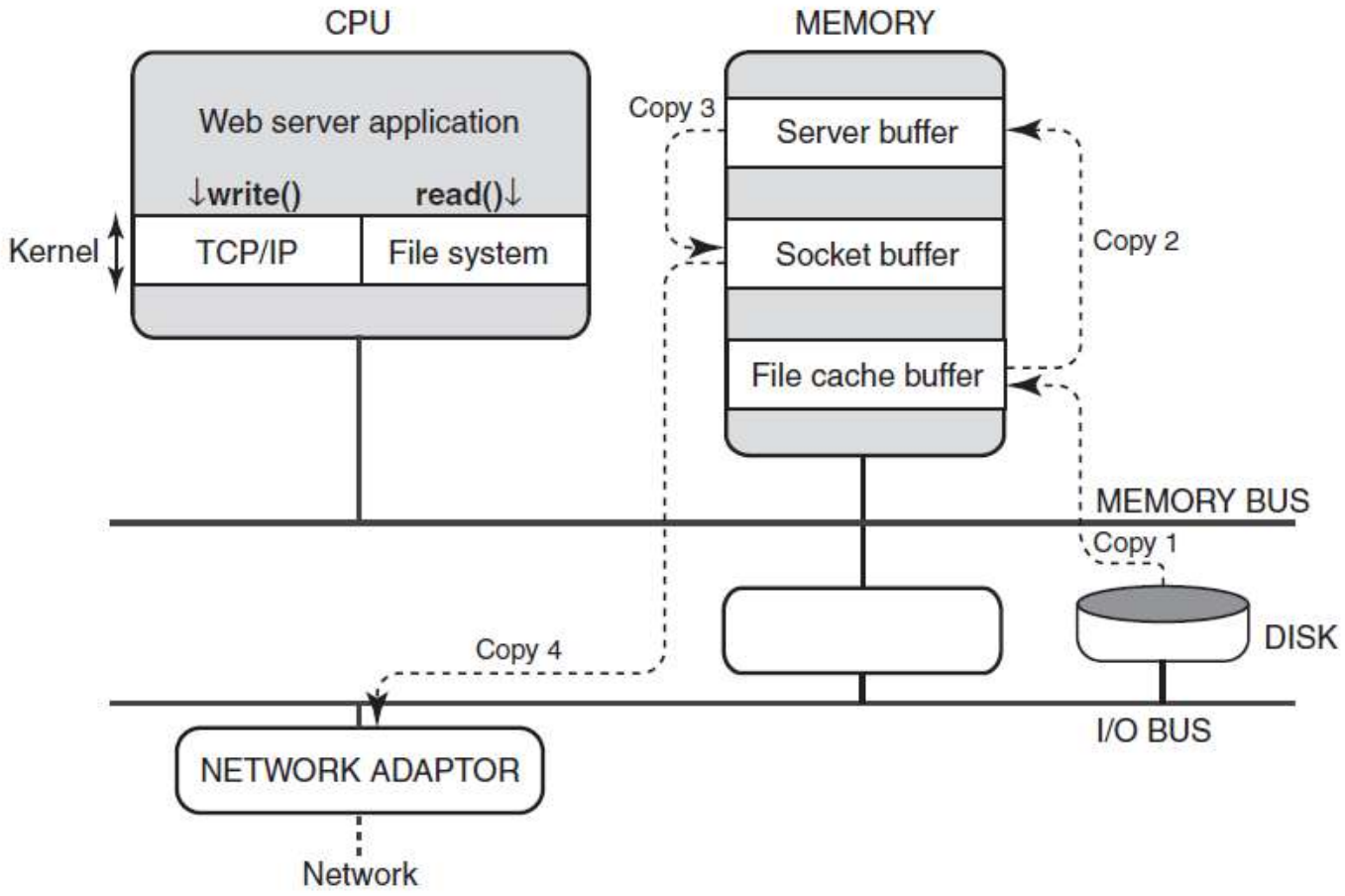


FIGURE 5.2 Redundant copies involved in handling a GET request at a server.

应用场景：

* 用户向web服务器请求一个静态文件
* 服务器从磁盘读出文件，发送到网络上

两个内核子系统：

* 文件子系统
* 网络子系统

一个简单的故事

- 直观上，这是一个简单的故事：
  - web应用程序通过一个系统调用（读文件），将文件从磁盘读入到它的缓冲区中
  - 构造一个HTTP响应头，通过一个系统调用（写套接字），将响应头和缓冲区内容交给网络子系统
  - 网络子系统将数据划分成适当大小的块，加上各层协议头，交给网络驱动程序

一个真实的故事

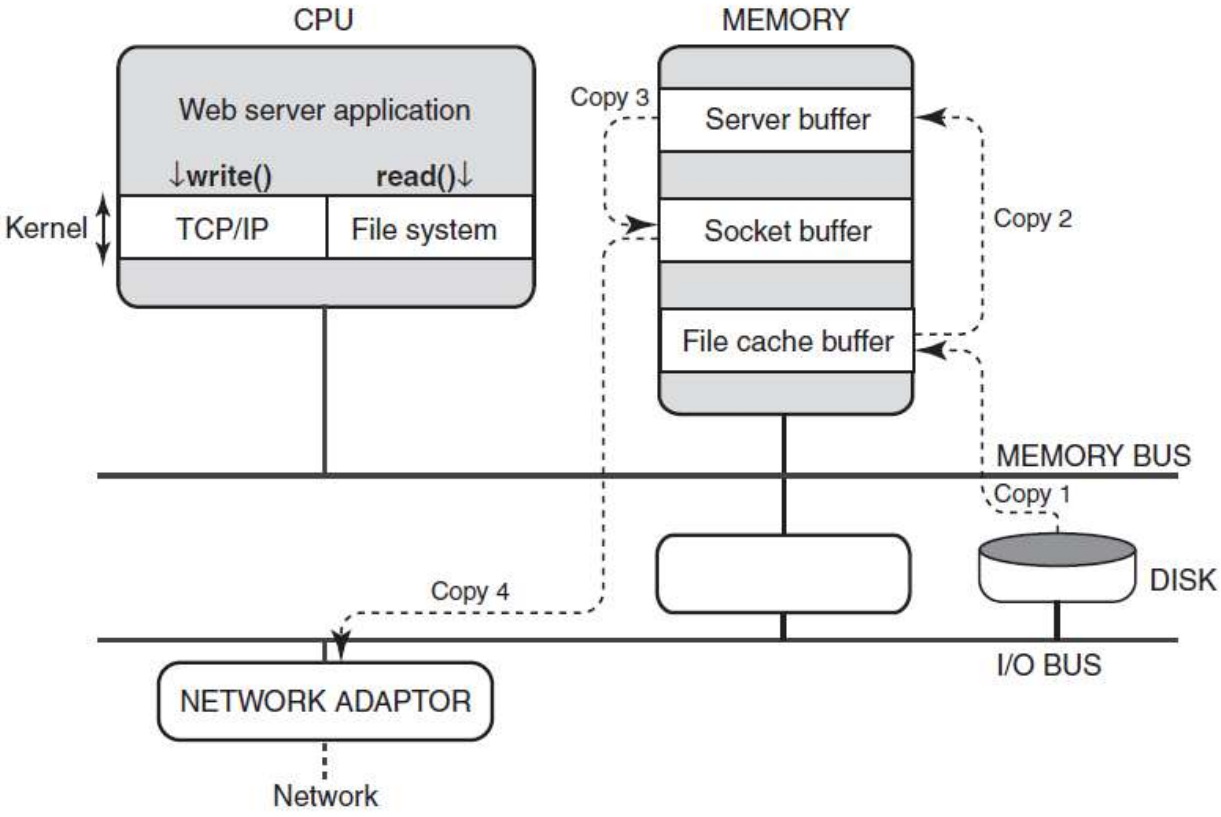


FIGURE 5.2 Redundant copies involved in handling a GET request at a server.

* Copy 1：
  * 硬盘→文件缓冲区（内核空间）
* Copy 2：
  * 文件缓冲区→应用缓冲区 （用户空间）
* Copy 3：
  * 应用缓冲区→套接字缓冲区（内核空间）
* Copy 4：
  * 套接字缓冲区→网卡
* TCP程序还需要扫描一遍数据，计算TCP检查和

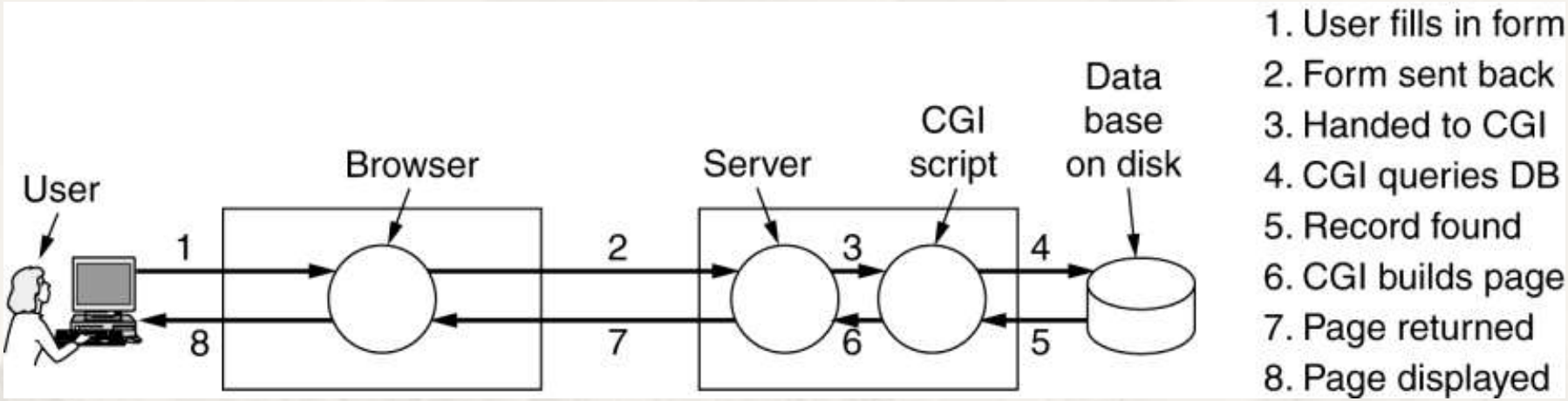
资源消耗情况

- 拷贝和TCP检查和计算：每个字需要穿过内存总线7~9次！
- 不同内存区域之间的拷贝（copy 2，copy 3）：
  - 每个字都要通过内存总线读一次和写一次
- 计算TCP检查和：每个字都要通过内存总线读一次
- 涉及外设的拷贝（copy 1，copy 4）：
  - 如果由CPU做拷贝（PIO）：每个字都要通过内存总线读一次和写一次
  - 如果由设备做拷贝（DMA）：每个字只需通过内存总线读一次或写一次
  - 涉及外设的拷贝都需要消耗I/O总线带宽

对服务器吞吐量的影响

- 在上面的例子中：
  - Web服务器吞吐量不超过T/7，T为内存速度和内存总线速度中的较小值
  - 有效的文件缓冲区大小仅为总容量的1/3
- 多余的拷贝在两个方面损害了服务器的性能：
  - 由于使用了过多的总线和内存带宽，服务器的运行速度远远低于总线速度
  - 由于使用了过多的内存，服务器不得不更多地从磁盘而不是主存读文件
- 如果请求动态内容，还要增加一次拷贝（CGI程序→web服务器）

请求动态内容



Step 6：CGI 程序将构造好的网页文件，通过进程间通信机制传给web服务器程序，涉及一次拷贝

5.2 消除copy 4

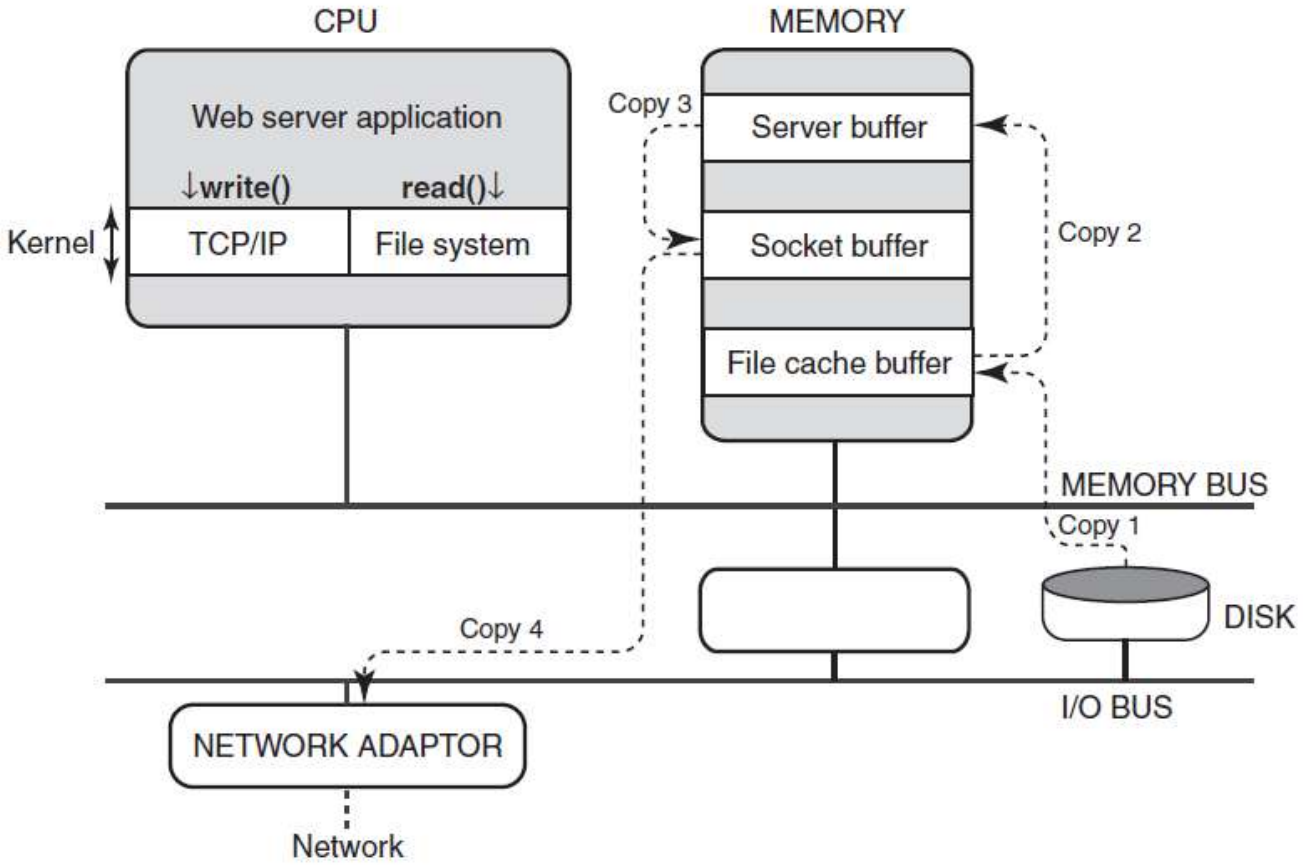


FIGURE 5.2 Redundant copies involved in handling a GET request at a server.

* 为什么需要copy 4?
* 简单的解释：
  * 适配器内存和内核存储空间不在同一个硬件上
* 但是，这个理由并不充分！

消除 Copy 4

- 在一个内存映射的体系结构中，
  - 设备寄存器被映射到一块内存区域，CPU通过读写这块内存区域与设备通信
  - 理论上，内存可以位于总线上的任何地方，包括在适配器中
- 消除copy 4的解决方案：
  - 利用网络适配器中已有的存储空间（P4，利用系统组件），以及内核存储空间放置的自由度（P13，利用自由度），将套接字缓冲区放在网络适配器中
  - 应用缓冲区的内容直接拷贝到网络适配器的内存中

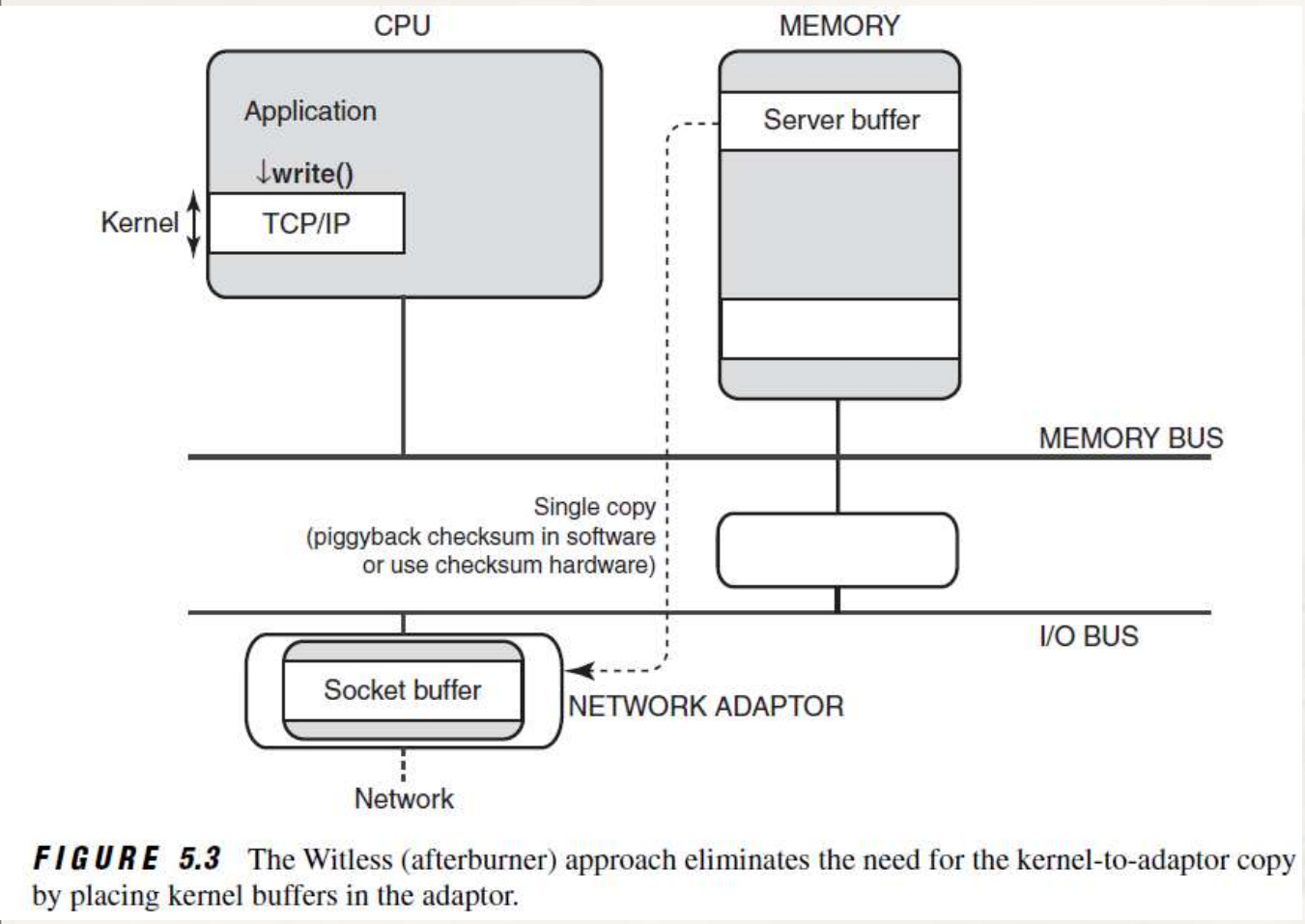


FIGURE 5.3 The Witless (afterburner) approach eliminates the need for the kernel-to-adaptor copy by placing kernel buffers in the adaptor.

* 如何计算TCP检查和？

如何计算检查和?

- Witless方法（P2c，共享开销）：
  - CPU执行拷贝，当读入每个字时，捎带计算检查和
  - 致命的问题：接收的时候，当发现检查和出错时数据已被写入应用缓冲区，与TCP语义不符（所以该方法从未被实施）
- Afterburner适配器（TCP offloading engine）：
  - 数据传输由网卡通过DMA完成，检查和也由网卡计算
  - TCP连接的管理（建立、关闭等）仍由主CPU完成，仅将建立好的TCP连接移交给网络适配器
  - 问题：网络适配器需要很大的内存空间和较强的处理器来支持大量的TCP连接，网卡成本可能较高

5.3 消除 Copy 3

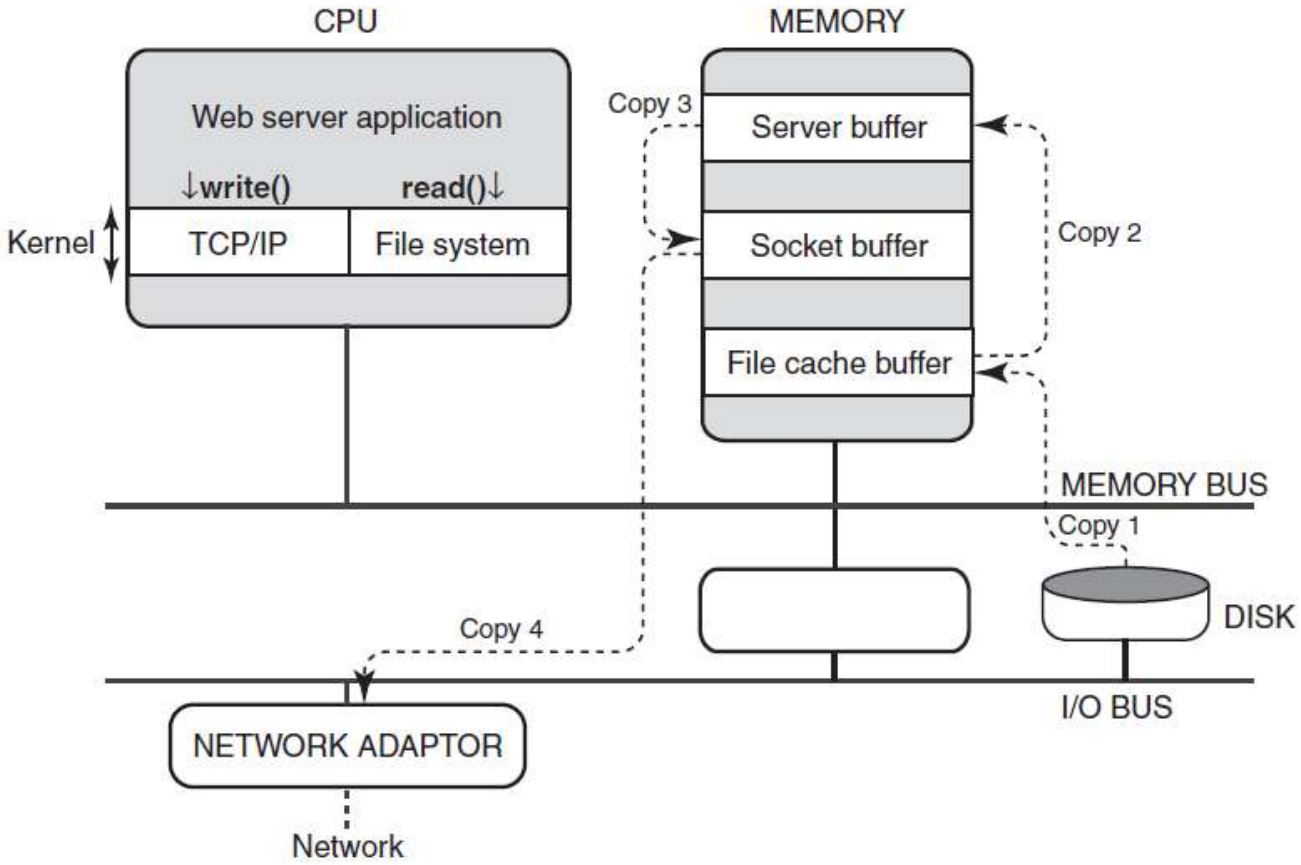


FIGURE 5.2 Redundant copies involved in handling a GET request at a server.

为什么需要copy 3？

* 应用和内核使用不同的虚拟地址空间（不是必要的）
* Socket API使用拷贝语义，应用和内核之间需通过拷贝解除耦合（必要的）

如果拷贝不能避免，那么能否减小拷贝的开销呢？

写时拷贝（copy-on-write）

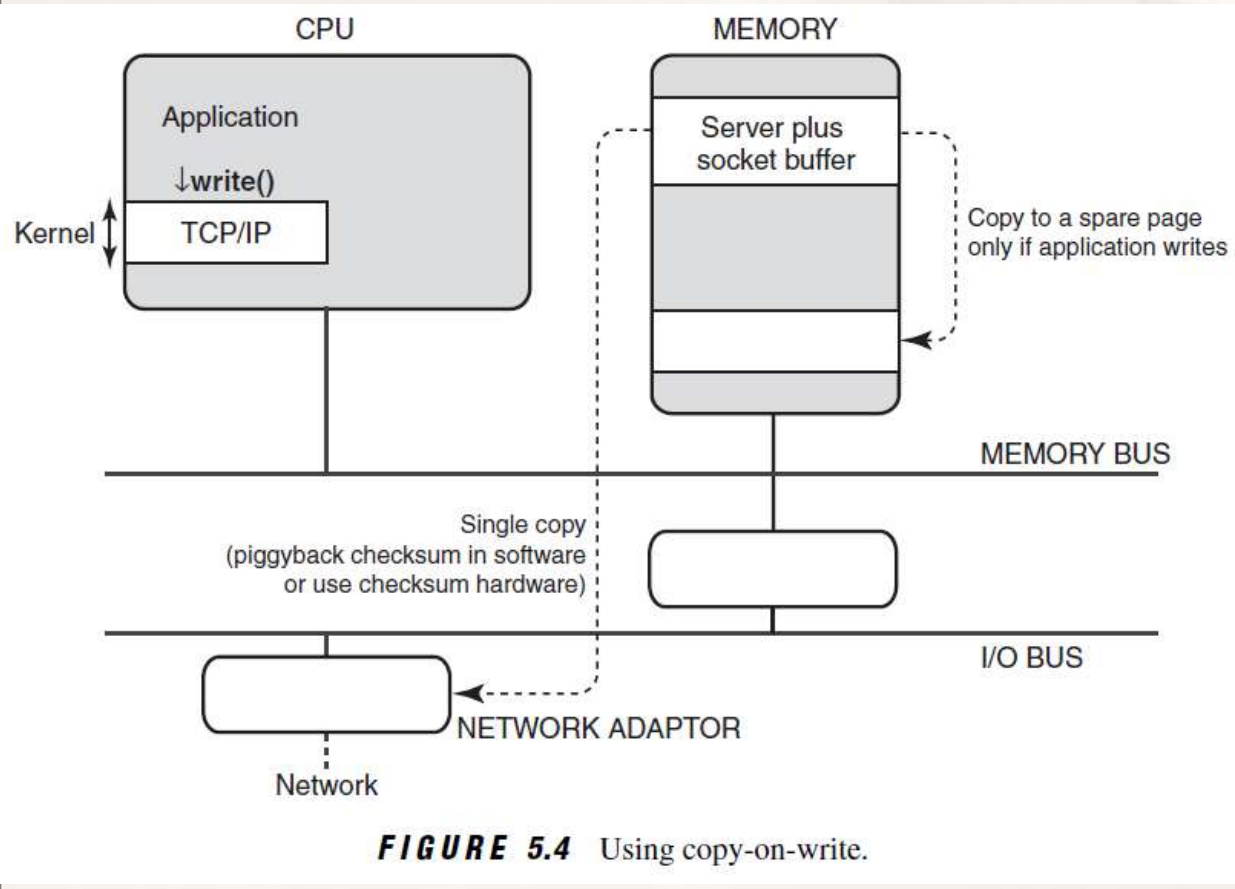


FIGURE 5.4 Using copy-on-write.

* 当应用程序对内核执行一个写时拷贝时，OS将内核缓冲区映射到应用缓冲区的物理内存页上
* 当应用程序试图修改其缓冲区时，内核进行真正的拷贝
* 有些操作系统提供写时拷贝，很多情况下可以避免真正的拷贝

写时拷贝的实现

- 举例：
  - 假定进程 P1 的虚拟页 X 映射到物理页 L上，需要复制X的内容到进程 P2 的虚拟页 Y
- 当P1对 X 进行写时拷贝时：
  - 内核修改P2的页表，令 Y 指向物理页 L
  - 将 X 表项的COW保护位置位
- 当P1试图写页X时：
  - 硬件读X的COW位，发现置位，产生一个异常
  - 操作系统将物理页 L拷贝到物理页 L’，清除 X 的COW位，令 X 指向 L’，Y 继续指向 L

写时拷贝的实现（续）

- 对于不提供写时拷贝功能的操作系统（如UNIX和Windows），也可以基于虚拟内存实现类似的功能：
  - 可以通过修改页表避免物理拷贝
  - 需要找到一种替代COW位的保护机制

5.4 优化页面重映射

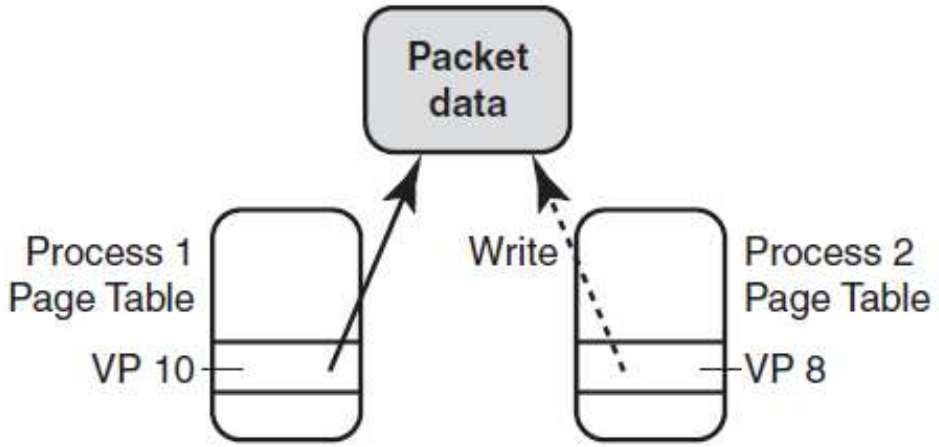


FIGURE 5.5 Basic operations involved in making a copy of a page using virtual memory.

* 对页面重映射过于简单的看法：
  * 只需修改P2的页表（一次写操作），令VP 8指向存放包的物理页，所有工作就结束了 （X）

页面重映射的开销

- 修改多级页表：
  - 实际映射可能要求修改多级页表，当页表不在内存中时要调入，并修改目录页
- 要求锁操作：
  - 修改页表前要请求锁，修改后要释放锁
- 刷新TLB：
  - 新的地址映射写入页表时，相关TLB表项要清除或修正
- 在目标域中分配虚拟内存：
  - 系统要在目标进程中找到一个空闲的页表表项
- 锁住物理页：
  - 为防止页被换出，必须锁住物理页
- 以上开销在多处理器系统中会被放大
- 页面重映射虽然只需常数时间，但这个常数因子非常大

结论：如果只是简单地使用页表重映射来避免拷贝，结果可能不像预期的那么好

Fbufs（fast buffers）

- 基本观察：
  - 如果一个应用正在发送大量的数据包，那么一个包缓冲区可能会被重用多次
- 方法一：
  - 提前分配好需要的包缓冲区，并计算好所有的页面映射信息（P2a），发送时重复使用这些包缓冲区
- 方法二：
  - 数据传输开始时分配包缓冲区并计算页面映射，然后将其缓存起来（P11a），消除后续包的页面映射开销
- 基本思想：映射一次，重复使用

为应用分配一组固定的物理页

- 为避免内核空间和用户空间之间的拷贝，将一组物理页P1、P2、.....、Pk 同时映射给内核和应用来使用
- 数据包经过的一系列处理程序构成一个有序的安全域序列，定义为一条路径
- 为隔离不同的应用，为每一条路径预留固定的一组物理页，数据包到达时立即确定其所属的路径（提前解复用）

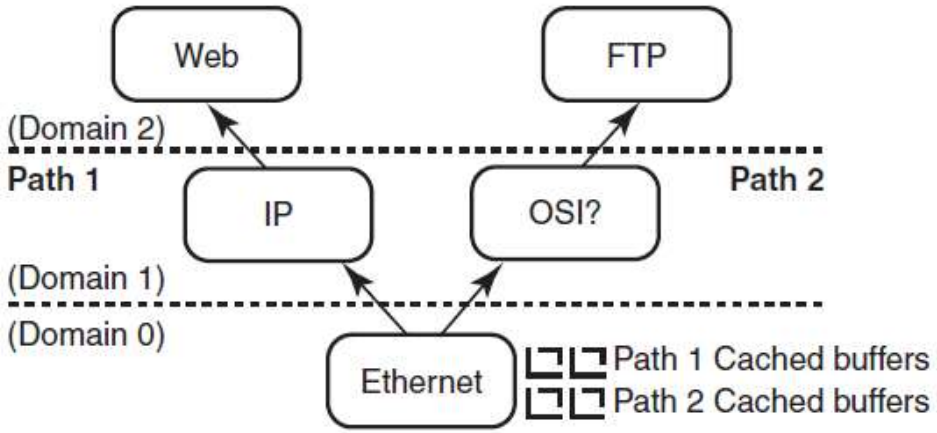


FIGURE 5.6 Premapping or lazily establishing buffer pages into the page tables of each domain in a path avoids the expense of page remapping in the real-time path, after the initial setup.

在路径上传递包缓冲区描述符

- 对于每条路径，适配器有一个空闲缓冲区链表：
  - 适配器把数据包写入一个空闲缓冲区，将缓冲区描述符传给接收路径上的下一个进程
  - 最后一个进程将用完的缓冲区交还给第一个进程，缓冲区重新回到空闲缓冲区链表

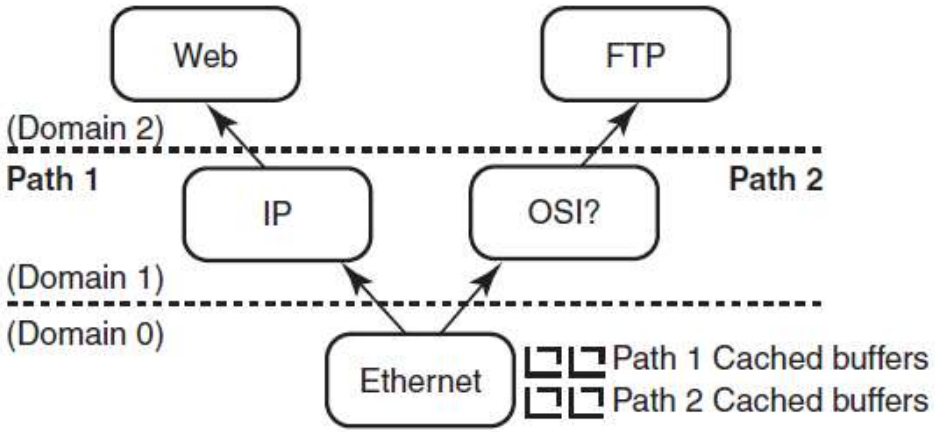


FIGURE 5.6 Premapping or lazily establishing buffer pages into the page tables of each domain in a path avoids the expense of page remapping in the real-time path, after the initial setup.

实现单向路径

* 有序的安全域序列是一条单向路径：
  * 规定第一个进程是writer，其余进程是reader（为了提供一定的保护级别）
  * 给第一个进程的页表表项设置写允许位，给其它进程的页表表项设置只读位

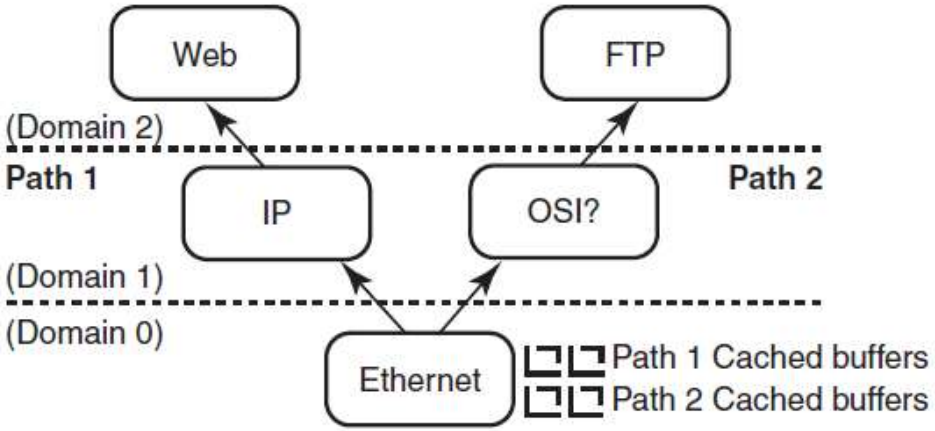
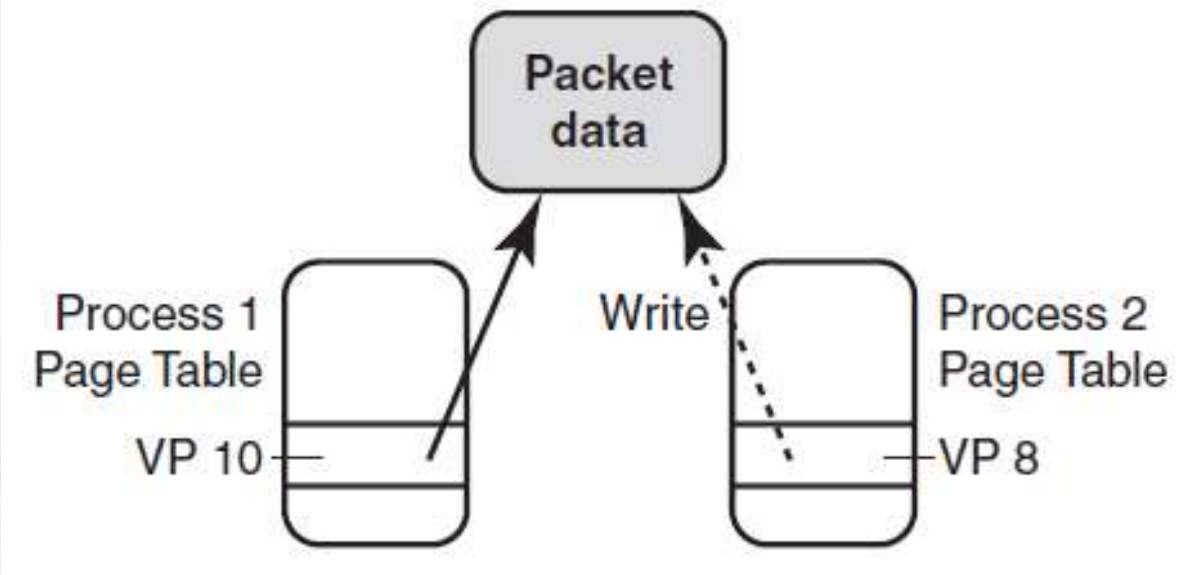


FIGURE 5.6 Premapping or lazily establishing buffer pages into the page tables of each domain in a path avoids the expense of page remapping in the real-time path, after the initial setup.

映射到同一个物理页的虚拟页号应相同



- 在进程间传递缓冲区描述符的问题：
  - 理论上，各个进程映射到同一个物理页上的虚拟页号可能不同
- 解决方法：
  - 规定：映射到同一个物理页的虚拟页号必须相同
  - 实现：所有进程的虚拟内存中一定数量的起始页预留为 fbuf 页

收包处理过程

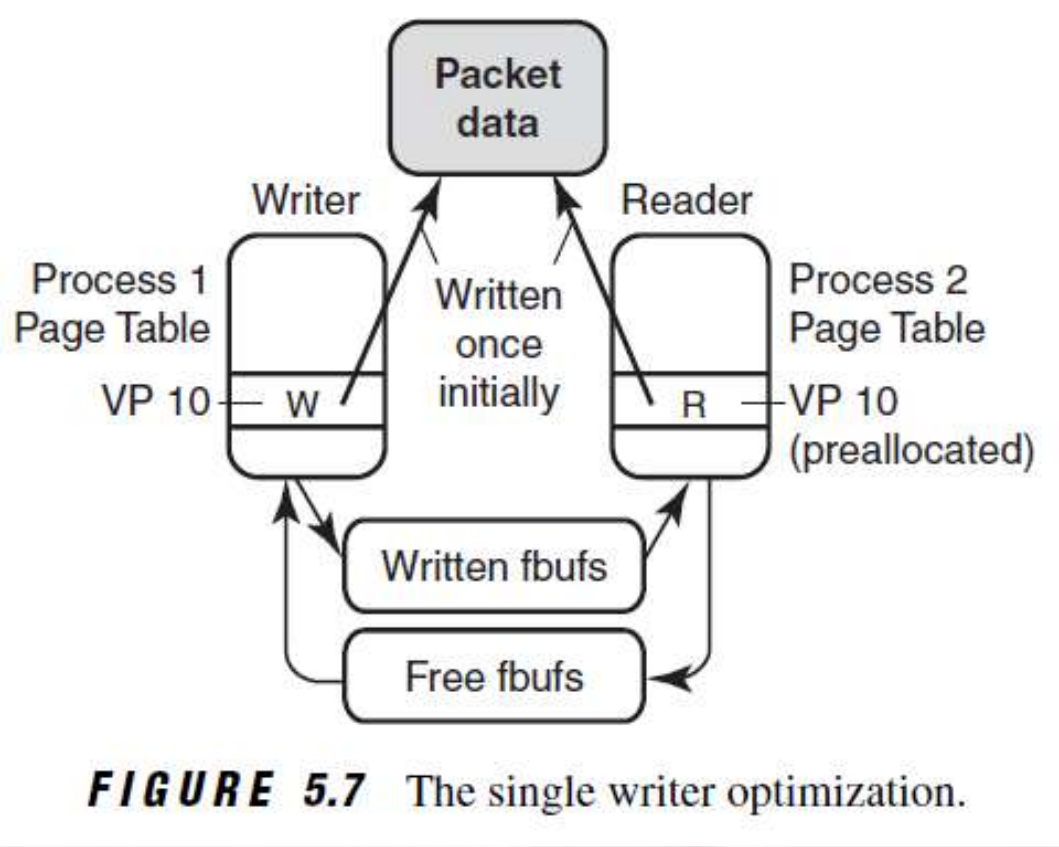


FIGURE 5.7 The single writer optimization.

P1进程：

* 从free fbufs队列取一个缓冲区描述符
* 将数据包写入缓冲区
* 将缓冲区描述符写入written fbufs队列

P2进程

* 从written fbufs队列取缓冲区描述符
* 从相应缓冲区读数据
* 将缓冲区描述符写回free fbufs队列

如何添加包头?

- 在发送路径上，每一个安全域都要给数据包加上一个包头
- 然而，为了实现保护，每条路径只允许一个writer，其余为reader
- 问题：怎么允许其它安全域添加包头呢？

定义数据包为聚合数据结构

- 将数据包定义为一个带有指针的聚合数据结构，每个指针指向一个fbuf
- 给数据包添加包头，就是将一个fbuf添加到聚合数据结构中

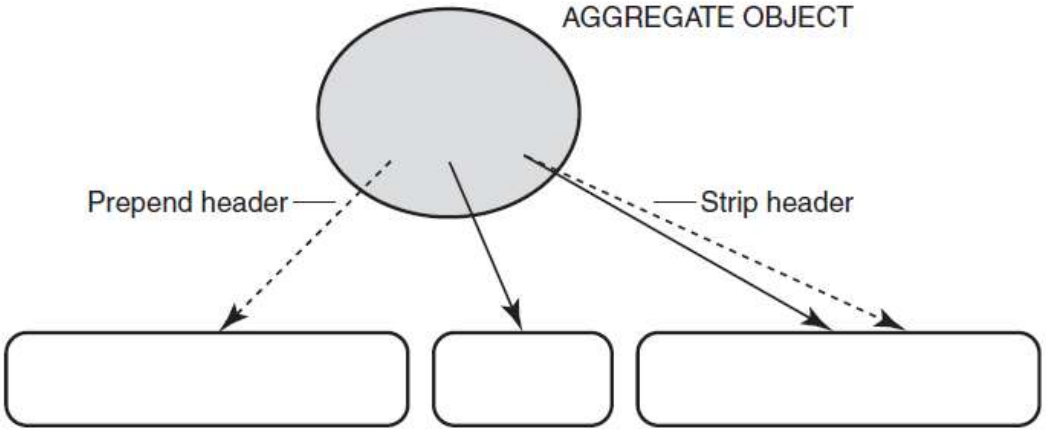


FIGURE 5.8 Using aggregate objects to allow adding layers to add headers while allowing only a single writer.

Fbufs总结

- Fbufs运用了虚拟内存映射的思想，通过在大量数据包之间分摊页面映射开销而做得更高效：
  - 包缓冲区映射一次，重复使用很多次
  - 消除了一般情形中的页表更新
- 有人扩展了Fbufs思想，并实现在Sun Solaris操作系统中
- Intel DPDK也运用了“一次映射，重复使用”的思想

应用如何使用Fbufs?

- 大量已有的应用软件是根据拷贝语义的socket API写的:
  - 应用执行了write()系统调用后，就可以重用包缓冲区，甚至释放包缓冲区了
- 采用fbufs后：
  - 在包缓冲区被其它进程使用完之前，应用不允许写或释放包缓冲区

解决方案：修改应用API

- 解决方法：
  - API 不再保持拷贝语义
  - 应用在写缓冲区之前必须进行判断
- 安全的实现方法：
  - 当一个fbuf从应用传递到内核后，内核翻转一个写允许比特，归还fbuf时再重新设置该位
  - 若应用在不允许写的情况下做写操作，会产生一个异常，提示出错，但不影响其它进程

已有的网络应用软件必须重写吗?

- 方法一:
  - 给已有的 API 增加新的系统调用，要求高性能的应用使用新的系统调用进行重写
- 方法二:
  - 用新的扩展实现一个公共的I/O库，链接到该库的应用不需要修改，就可以得到性能提升

- 实践表明，将应用移植到类fbuf的API，对应用所做的修改不大，且是局部的，因此fbufs方案是可行的

5.5 使用RDMA避免拷贝

- 在web服务器的例子中：
  - Web服务器接收请求，将文件传输到网络上
  - 从web服务器作为发送端的角度考虑消除拷贝
  - Web服务器作为接收端并不需要保存请求消息
- 现考虑在两个计算机之间传输一个大文件，接收端需要保存收到的数据
- 为减少拷贝，接收端采用以下方式之一收包：
  - 采用fbufs
  - 采用TOE网卡

方法一：采用fbufs收包

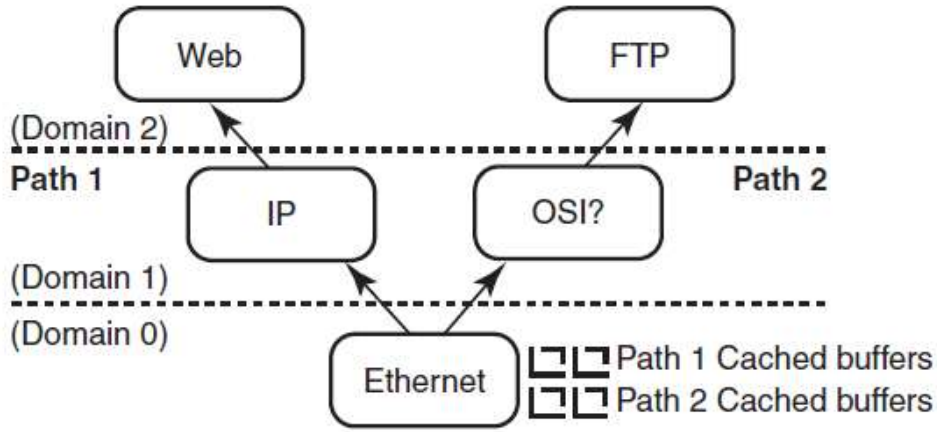


FIGURE 5.6 Premapping or lazily establishing buffer pages into the page tables of each domain in a path avoids the expense of page remapping in the real-time path, after the initial setup.

* 包到达网卡后，被拷贝到一个包缓冲区中
* 包缓冲区描述符在路径上传递，各安全域处理包
* 应用程序将数据拷贝到应用缓冲区，释放包缓冲区（这里需要一次拷贝）

方法二：采用TOE网卡收包

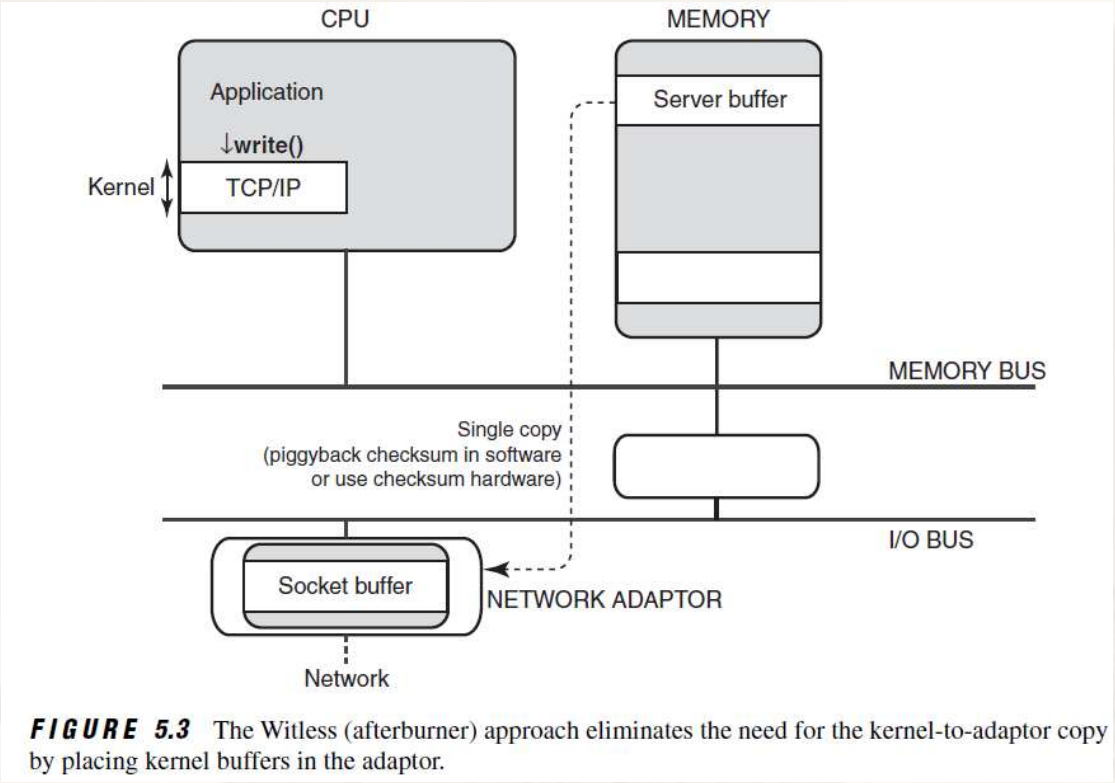


FIGURE 5.3 The Witless (afterburner) approach eliminates the need for the kernel-to-adaptor copy by placing kernel buffers in the adaptor.

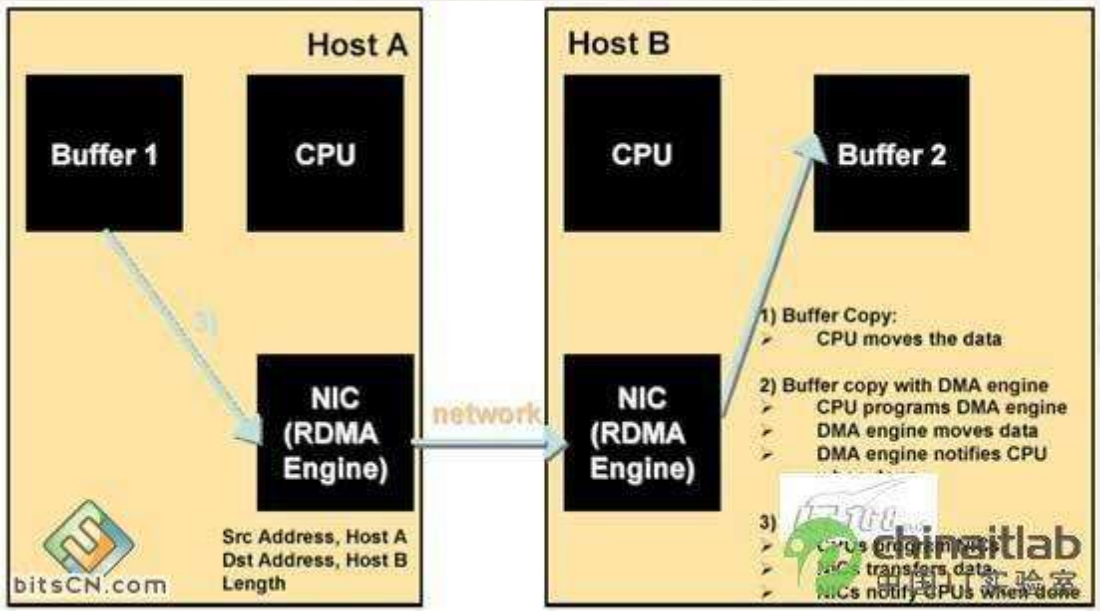
* 包到达网卡后，被送入套接字缓冲区进行协议处理和重组
* DMA控制器将数据送入应用缓冲区，向CPU发出中断
* 驱动程序通知应用接收数据
* 应用拷贝数据到文件缓冲区，将应用缓冲区归还给网卡（这里需要一次拷贝）

直接内存访问（DMA）

- 在上述两种方法中，CPU要参与数据传输，且数据到达目的计算机的内存后还要拷贝一次
- 我们知道，使用DMA在外设和内存之间传输数据，不需要CPU的参与：
  - CPU设置DMA（给出数据的存放地址、长度等）
  - DMA控制器完成数据传输
  - DMA控制器通过中断通知CPU传输完成
- 受DMA的启发，能否在两台计算机的内存之间直接传输数据，而不需要CPU参与？

远程直接内存访问（RDMA）

- RDMA的愿景：
  - 数据在两台计算机的主存之间直接传输，不需要CPU参与到数据传输的过程中
  - 两个网络适配器协作地从一个主存读数据，然后写入另一个主存



RDMA需要解决的问题

- 除了需要网卡执行协议处理外，RDMA还需解决两个问题：
  - 接收端适配器如何知道应将数据放在哪儿？（不能求助CPU）
  - 如何保证安全？（发送进程不能随意写目标终端的内存）

VAX集群的RDMA

- RDMA在VAX集群中已经被使用：
  - VAX系统的核心是一个140Mb/s的网络（称为Computer Interconnect，CL），使用一个以太网风格的协议
  - 用户可以将许多VAX计算机和网络硬盘连接到CL
- RDMA的需求背景：
  - 在远程硬盘和VAX机的内存之间有效传输大量数据
  - 要求包含文件数据的包在进入目的适配器之后，直接到达它的存放位置

接收端适配器应将数据放在哪儿?

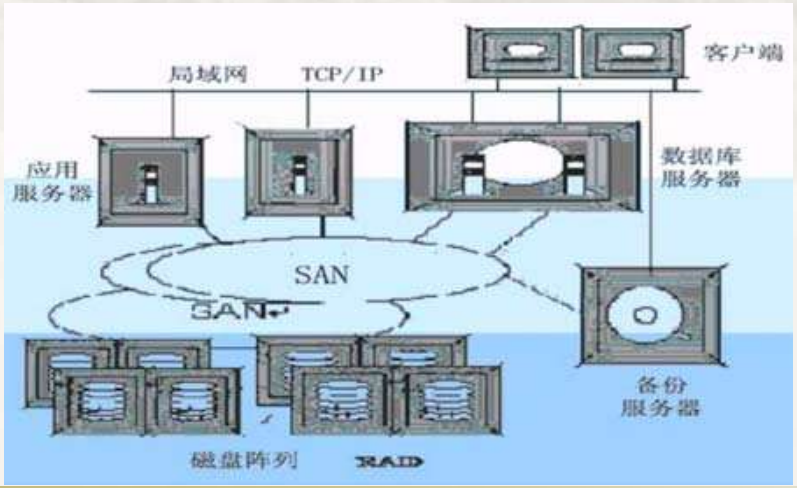
- 接收端应用锁住一些物理页，用作文件传输的目标存储区域（其呈现出来的逻辑视图是由地址连续的虚拟页组成的一个缓冲区），缓冲区ID被发送给发送端应用
- 发送端应用将缓冲区ID及包存放的偏移量，随同数据包一起发送到接收端（P10，传递线索）
- 接收端适配器根据缓冲区ID和偏移量，将数据包内容存放到指定的位置（一步到位）

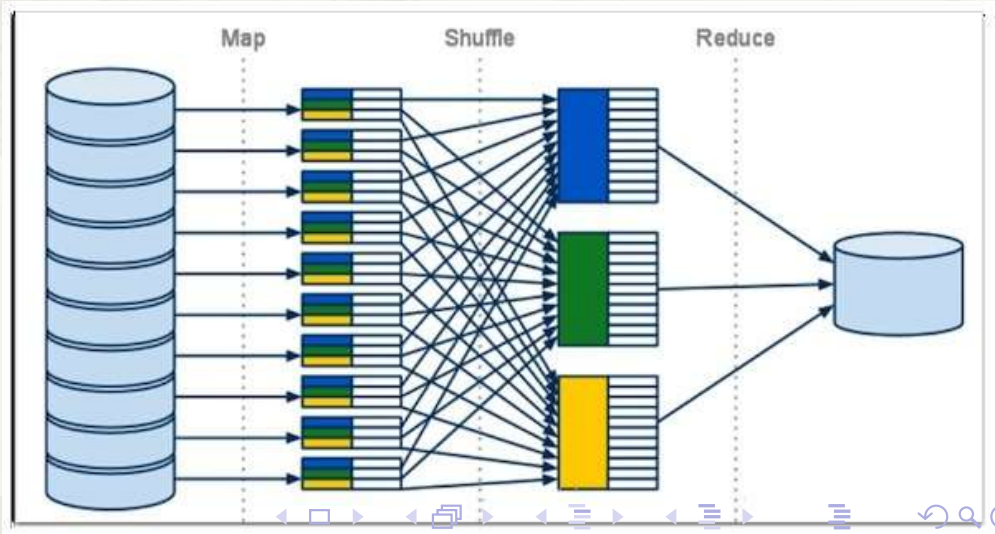
如何保证目标存储区域的安全?

- 允许将一个携带缓冲区ID的网络包直接写入内存，是一个明显的安全隐患
- 为降低安全风险，
  - 缓冲区ID中包含一个难以猜测的随机串（防止伪造）
  - VAX集群只在本集群内部可信的计算机之间使用RDMA传递数据

RDMA的应用

- 存储区域网（Storage Area Network，SAN）：
  - 一种后端网络，将大量计算机和网络硬盘连接在一起
  - 目前有好几种这样的技术，都使用了RDMA的思想，如Fiber Channel（FC）、iSCSI、Infiniband等
- 数据中心支持高性能分布式计算：
  - 大数据分析（MapReduce框架）
  - 深度学习（TensorFlow、Caffe等）





5.6 把避免拷贝技术扩展到文件系统

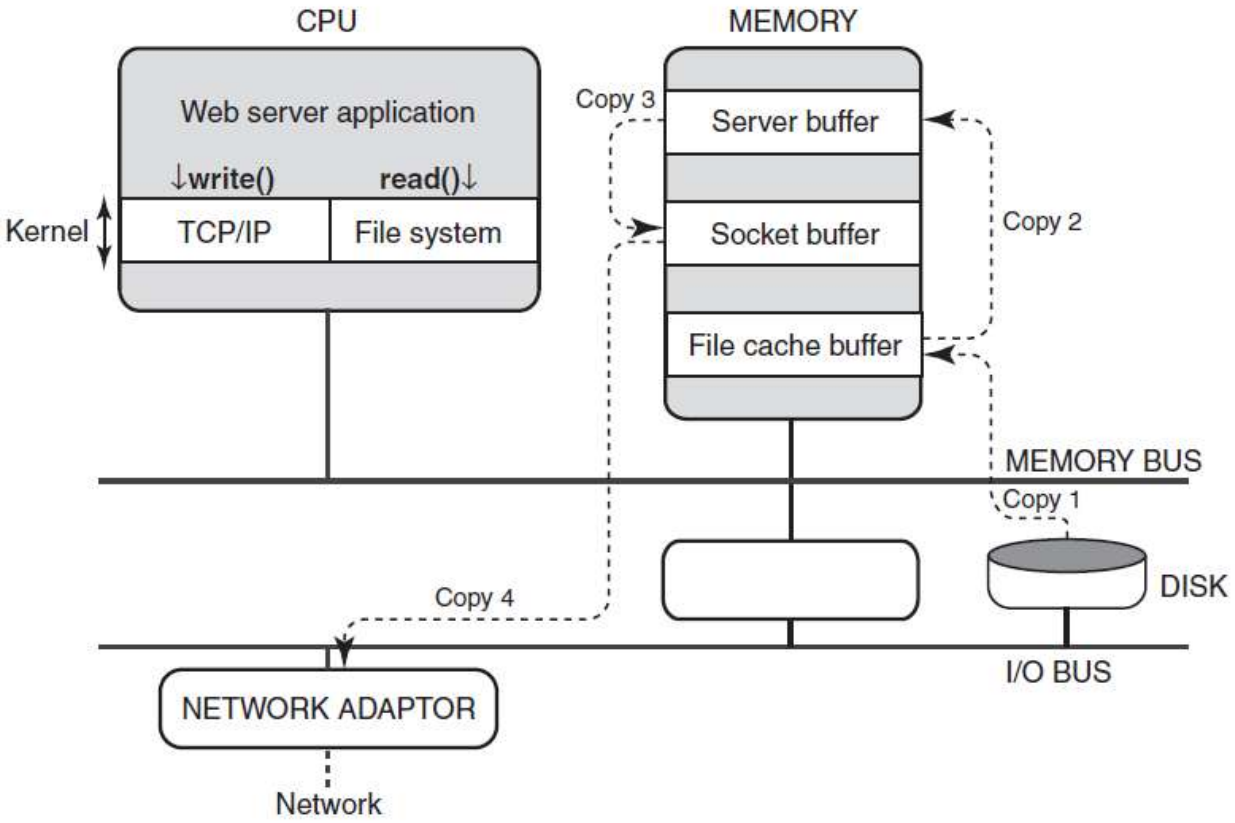


FIGURE 5.2 Redundant copies involved in handling a GET request at a server.

* 为提高响应速度，Copy 1是必要的
* 考虑消除copy 2

5.6.1 共享内存方法

- 类UNIX操作系统提供一个系统调用mmap()，允许应用（如web服务器）将一个文件映射到自己的虚拟地址空间。
- 概念上，当一个文件被映射到一个应用的地址空间，这个应用就好像在自己的内存中缓存了这份文件。当然，这个缓存的文件只是一组映射。
- 如果Web程序将文件映射到自己的地址空间，则它和文件cache访问的是同一组物理页（免除了拷贝）。

举例：Flash Web服务器

- Web应用程序将经常用到的文件映射到自己的内存空间
- 受到可分配给文件页的物理页数量及页表映射的限制，Flash Web服务器只能缓存和映射最近常用的文件
- 事实上，Flash Web服务器只是缓存了一些文件分片（通常是文件的头几个分片），并使用LRU策略将最近一段时间未用的文件unmap

Flash Web尚未解决的问题

- Flash Web 不能避免web服务器与CGI进程之间的拷贝
  - 文件缓存只能缓存静态内容，动态网页要由CGI程序生成
  - CGI程序生成的动态内容通过UNIX管道传给web服务器；典型地，管道要在两个地址空间之间拷贝内容
- 到目前为止，我们的方案都没有涉及TCP检查和
  - 一个被访问多次的文件，文件分片都相同，但TCP检查和未被缓存

由 fbufs 和 mmap() 想到的问题

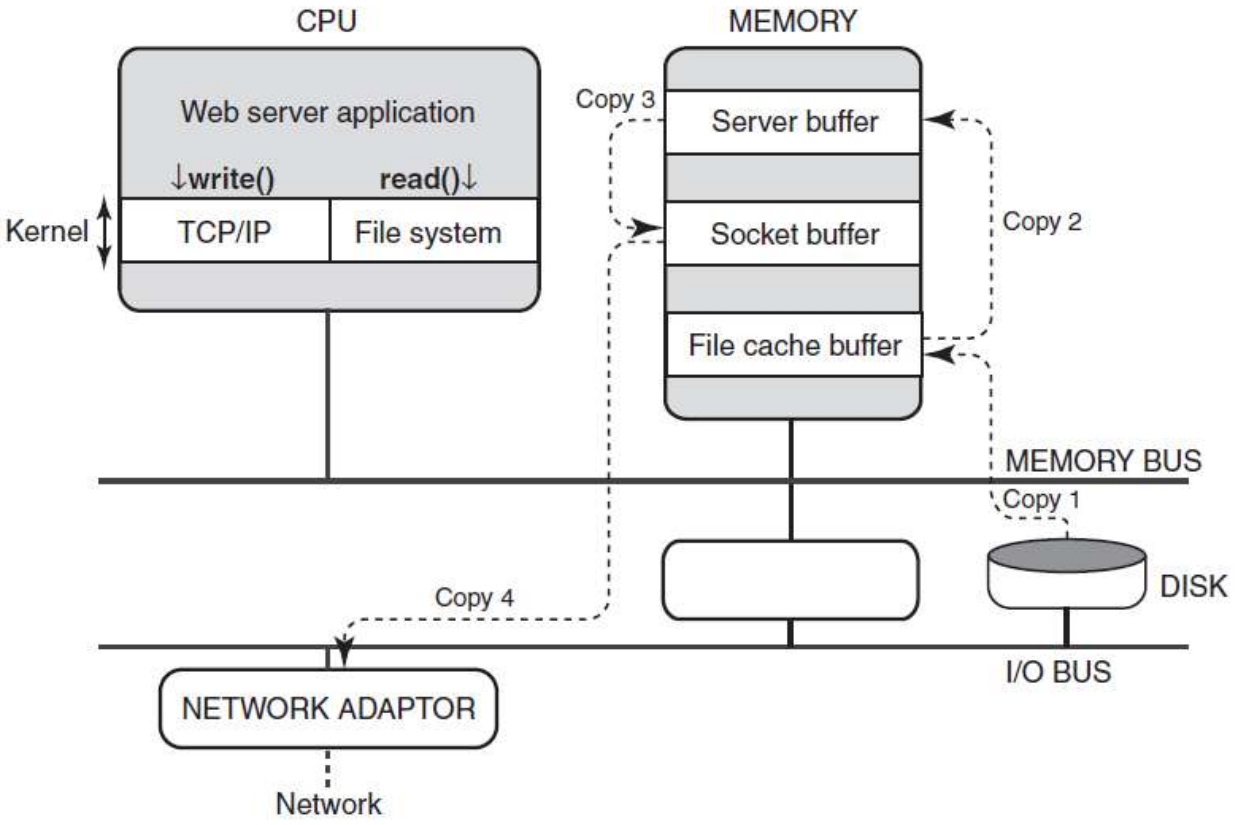


FIGURE 5.2 Redundant copies involved in handling a GET request at a server.

- fbufs可以消除 copy 3
- mmap()可以消除 copy 2
- Q：能否将 fbufs 和 mmap() 结合起来使用，同时消除 copy2 和 copy3 ？

可以结合 fbufs 和 mmap 吗?

- 如果采用fbufs：
  - 所有进程的虚拟内存中一定数量的起始页预留为fbuf页
  - 应用进程的应用缓冲区不能使用这些页
- 如果应用将文件映射到其虚拟地址空间的一个缓冲区：
  - 这个缓冲区不能用fbuf发送，必须要有一次物理拷贝！
- 当用mmap消除copy 2时，copy 3不能避免！

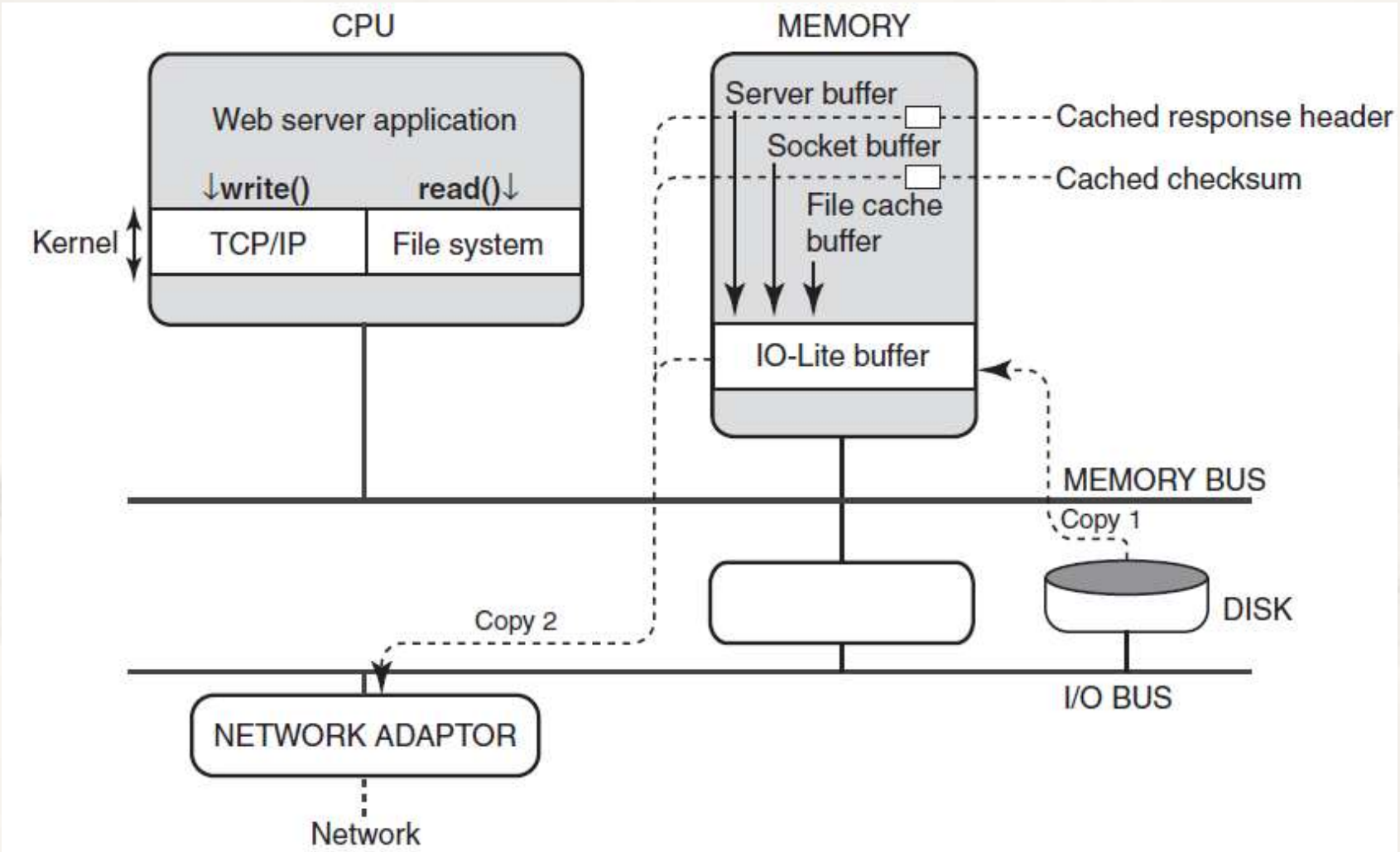
5.6.2 IO-Lite

- IO-Lite 将 fbufs 推广至包含文件系统，从而不必使用 mmap
- IO-Lite可以一揽子解决前面所有的问题：
  - 同时消除 copy 2 和 copy 3
  - 消除CGI程序和web服务器之间的拷贝
  - 缓存传送过的数据块的检查和

IO-Lite 的主要思想

- IO-Lite 借用了 fbufs 的主要思想：
  - 为同一条路径上的进程映射相同的物理页，实现只读共享
  - 推迟创建路径的缓冲区
  - 使用缓冲区聚合以允许添加包头

IO-Lite 响应 Get 请求



IO-Lite 响应 Get 请求的步骤

- 当文件第一次从磁盘读入文件系统的高速缓存时，文件页被保存为IO-Lite buffer
- 当应用通过一个系统调用读文件时，创建一个缓冲区聚合体，指针指向IO-Lite buffer
- 当应用发送文件给TCP时，网络子系统得到一个指向相同IO-Lite页的指针
- 应用将常用文件的HTTP响应头维护在一个高速缓存中
- IO-Lite给每个缓冲区分配一个编号，TCP模块维护一个以缓冲区编号为索引的检查和高速缓存<缓冲区ID，检查和>

实现零拷贝的管道

- IO-Lite也可以用来实现一个消除了拷贝的改良型管道程序（传递IO-Lite buffer的指针而不是拷贝）
- 将改良后的管道应用到CGI程序和web服务器之间，可以消除冗余的拷贝
- IO-Lite已经在UNIX中实现了

5.6.3 使用I/O拼接避免文件系统拷贝

- I/O拼接的基本思想：
  - 引入一个新的系统调用sendfile()，允许内核将读文件的调用和向网络发送消息的调用合并
- 文件到socket传输的传统方法需两次系统调用：

```
read (file, tem_buf, len);
write (socket, tmp_buf, len);
```

- 使用sendfile()传输文件到socket：

```
sendfile (socket, file, len);
```

内核2.1版本的sendfile实现

- 调用sendfile()时：
  - 文件数据先被拷贝到内核中的文件缓冲区（copy 1）
  - 然后从文件缓冲区拷贝到内核中的socket缓冲区（合并copy 2和copy 3）
  - 最后从socket缓冲区拷贝到适配器（copy 4）
- 与read/write方式相比，减少了一次拷贝

内核版本2.4之后的sendfile实现

- 调用sendfile()时：
  - 文件数据先被拷贝到内核中的文件缓冲区（copy 1）
  - 将记录数据位置和长度的信息保存到socket缓冲区
  - 数据通过DMA通道直接发送到适配器（copy 4）
  - 消除了copy 2 和 copy 3
- 基于sendfile的机制不能推广到与CGI程序通信
- Sendfile() 已用于Apache、Nginx、Lighttpd等web服务器中

5.8 扩展到数据操作之外

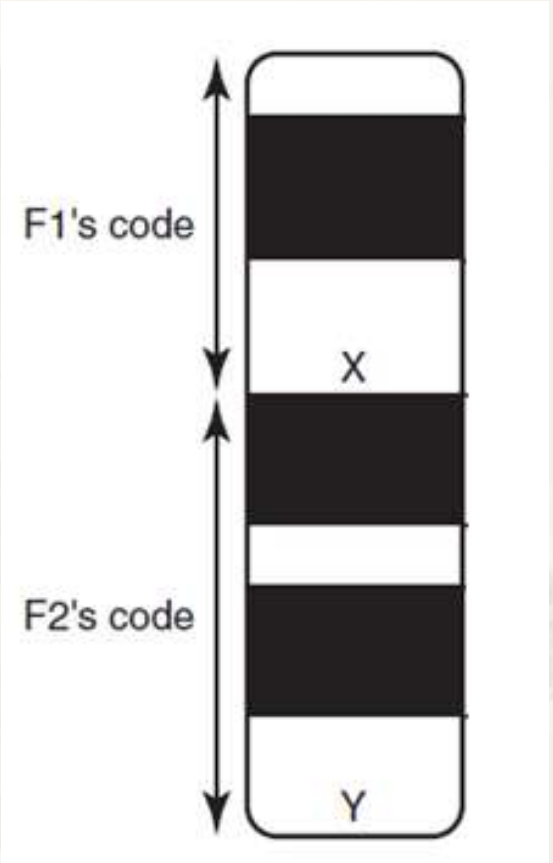
- 消除数据拷贝是为了避免冗余的数据读/写操作，以减少对内存和内存总线的压力
- 除拷贝外，对内存总线使用效率影响较大的因素还有**Cache**

5.8.1 有效使用 I-cache

- 处理器包含一个或多个数据cache（d-cache），以及一个或多个指令cache（I-cache）：
  - 一般而言，包数据几乎不能从 d-cache 获得好处
  - 处理数据包需要的状态可以从d-cache 获益
  - 处理数据包的程序代码可以从 I-cache 获益
- 数据、状态、代码都可能竞争内存带宽，相比而言，代码对内存带宽的竞争更严重：
  - 以太网上最大的数据包为1.5KB
  - 处理一个包需要的状态一般较小，比如一个连接表项
  - 1995年NetBSD TCP协议栈代码34KB（不包括应用协议代码）
- I-cache容量很小，数据包处理代码不可能都在I-cache中！

I-Cache的实现特点（1）

- 大多数处理器使用直接映射的I-cache：
  - 内存地址的低位比特用来检索I-cache条目
  - 如果高位比特匹配，直接从I-cache返回内容
  - 若不匹配，进行一个内存访问，用新的内容替换原来的条目
  - 对于32KB的I-cache，内存地址最低15位相同的指令被映射到cache的同一位置
- 问题一：
  - 被映射到I-cache同一位置的代码会被轮流替换出去，即使它们都是经常使用的代码。



I-Cache的实现特点（2）

- 每一个I-cache条目包含多条指令，可以看成是一个代码块：
  - 当取一条指令时，同一个代码块中的全部指令都会被读入（基于空间局部性假设做的优化）
- 问题二：
  - 不常用的代码会被读入I-cache，如果它与常用代码在一个块中。

举例

- 许多网络代码包含错误检查，比如：

  if condition T do X, else do Z

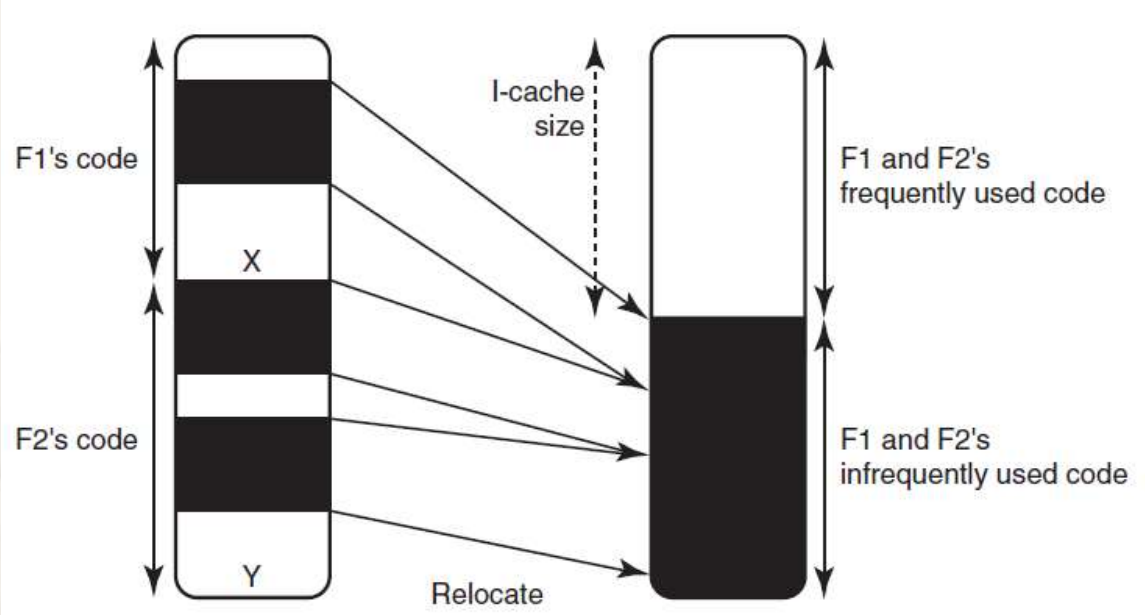
- 虽然 Z 几乎从不被执行，但是编译器通常会将 Z 的代码紧跟在 X 的后面

- 如果 X 和 Z 位于同一个指令块中，取经常使用的代码 X，会把不经常使用的代码 Z 也取进来，浪费了内存带宽和cache空间

问题与解决方案

- 以上两个结果和我们对于cache的一般预期不同：
  - 经常使用的代码不一定在cache中：由一个不完美的映射函数引起
  - 不常使用的代码可能被经常调入cache：由cache对空间局部性的优化引起
- 如何解决以上问题？
  - 重新组织代码，将经常使用的代码连续放置

重新组织代码



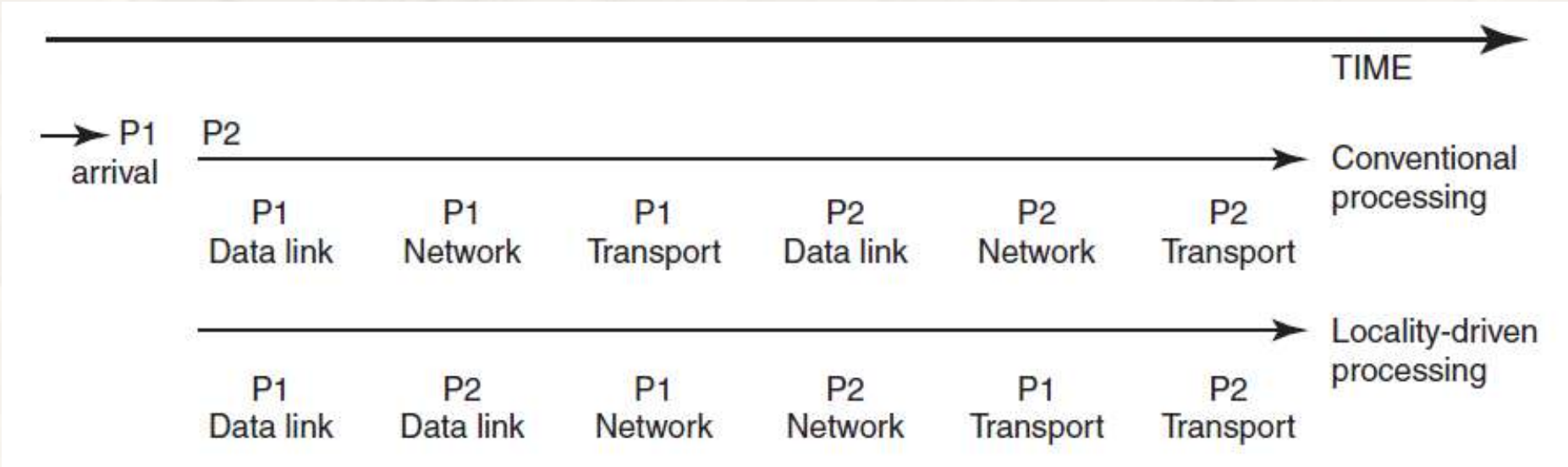
* 如果工作集超过了I-cache的大小，第一个问题仍会出现，但会减少，而第二个问题能够得到很大程度的缓解
* 代码布局的基本思想：通过优化代码在内存中的位置，减少代码的换入换出

新的问题

- 处理包的协议代码肯定无法全部装入指令cache：
  - I-cache的容量非常有限
  - 计算机上的所有程序都要竞争I-cache
- 问题：
  - 如果每处理一个包，就要将全部的协议代码装入I-cache一次，效率太低！！
  - 拷贝数据包的开销和将代码装入I-cache的开销比起来，简直就是微不足道！

局部性驱动的协议层处理

* 基本思想（若每一层协议代码可以装入I-cache）
  * 每个协议层一次处理多个包，分摊装载 I-cache的开销
  * 每一批处理的包数量越多，I-cache的使用越高效
  * 具体实现时，应能动态调整批处理的大小



软件工程方面的考虑

- 代码重新组织可以让编译器来做：
  - 程序员对不常使用的代码分支进行标注，由编译器为I-cache重新组织代码。
- 局部性驱动的协议层处理需修改层间通信方法：
  - 如果协议代码使用一个过程调用将数据包传递给上（下）一层，则代码修改为将数据包加入上（下）一层的一个包队列中
  - 当一个协议层被调用时，从自己的读队列中取数据包处理，直至队列取空
  - 当包缓冲区可以在不同层重用时，该策略工作很好

5.9 小结

- 本章以web应用为例介绍了优化内存和总线带宽
- 的技术，主要包括：
  - 使用适配器内存消除 copy 4
  - 使用fbufs消除copy 3
  - 使用mmap消除copy 2
  - 使用IO-Lite、sendfile消除 copy2 和copy 3
  - 通过代码重新布局、局部性驱动的协议层处理优化I-cache的使用
- 这些技术大部分需要修改内核和应用，但是这些修改都是局部的，并且基本上保留了模块化特性

本章使用的技术以及主要的原则

Number	Principle	Used In
P13	Memory location (on adaptor) as degree of freedom	Afterburner
P2b	Lazy copying using copy-on-write	Mach
P11a P7	Cache VM mappings per path Uniform fbuf space across processes	Solaris fbufs
P10	Pass buffer name and offset in packet	RDMA systems
P4	VM mapping to avoid copies in cache and application	Flash
P11a	Cache VM mappings per path Buffer sequence numbers enable checksum caching	Flash-lite
P6	New system call that splices I/O	Sendfile()
P1	Avoid repeated memory access across manipulations	ILP
P13	Layout code to minimize i-cache misses	x-kernel
P13	Layer processing order as degree of freedom	LDRP

P1原则的运用

- 本章反复运用原则P1（避免显而易见的浪费）来消除不必要的读、写操作
- 运用这个原则的困难在于，如果不把视野尽可能放宽到整个系统，浪费并不是显而易见的
- 运用P1原则要求对整个系统有一个概要的了解，使用简单的模型（硬件、体系结构、操作系统、协议）就可以做到这一点
- 系统技术的复杂性并不在于深度，而在于广度（涉及多领域的知识）

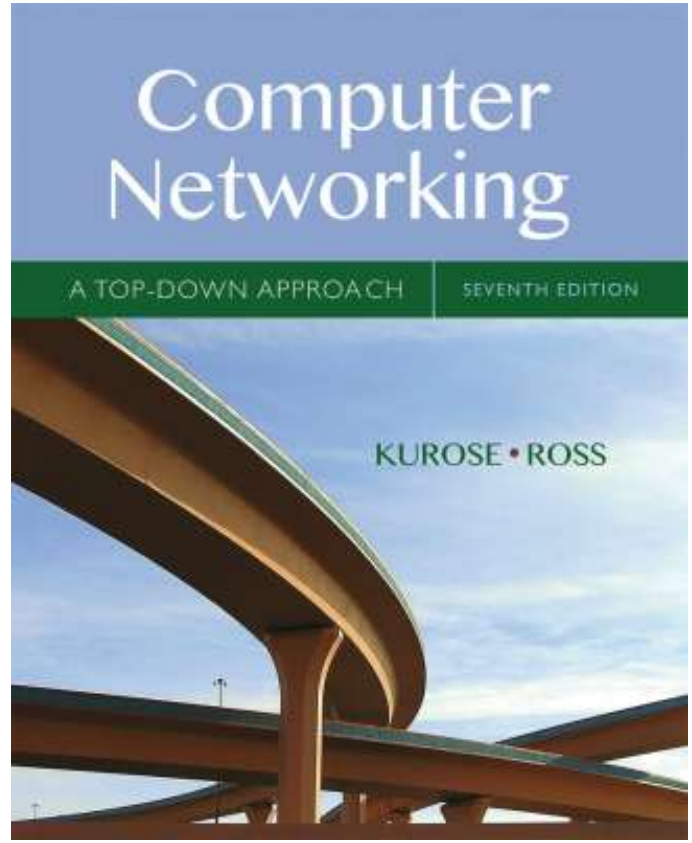
文献阅读

[1] Network Stack Specialization for Performance, Sigcomm 2014

- 要求：选择Sandstorm或者Namestorm，说明它使用了哪些提高吞吐量的技术

- 作业提交：11月18日

Chapter 6
The Link Layer and LANs



A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- *If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)*
- *If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.*

Thanks and enjoy! JFK/KWR

Computer Networking:
A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

Chapter 6: The Data Link Layer

Our goals:

- 理解数据链路层服务原理:
  - 差错检测和纠正
  - 共享广播信道: 链路接入
  - 链路层编址
- 链路层实现
  - 以太网
  - 虚拟局域网

Link layer, LANs: outline

6.1 introduction, services

6.2 error detection, correction

6.3 multiple access protocols

6.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS

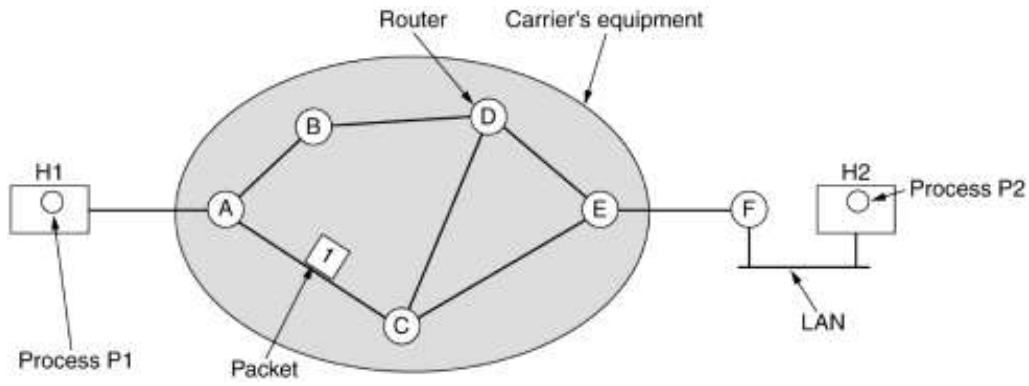
6.6 data center networking

6.7 a day in the life of a web request

网络层、链路层和物理层

网络层：

- 选路：路由器确定去往目的节点的下一跳
- 转发：在路由器内部将数据报从输入端口转移到输出端口



链路层：

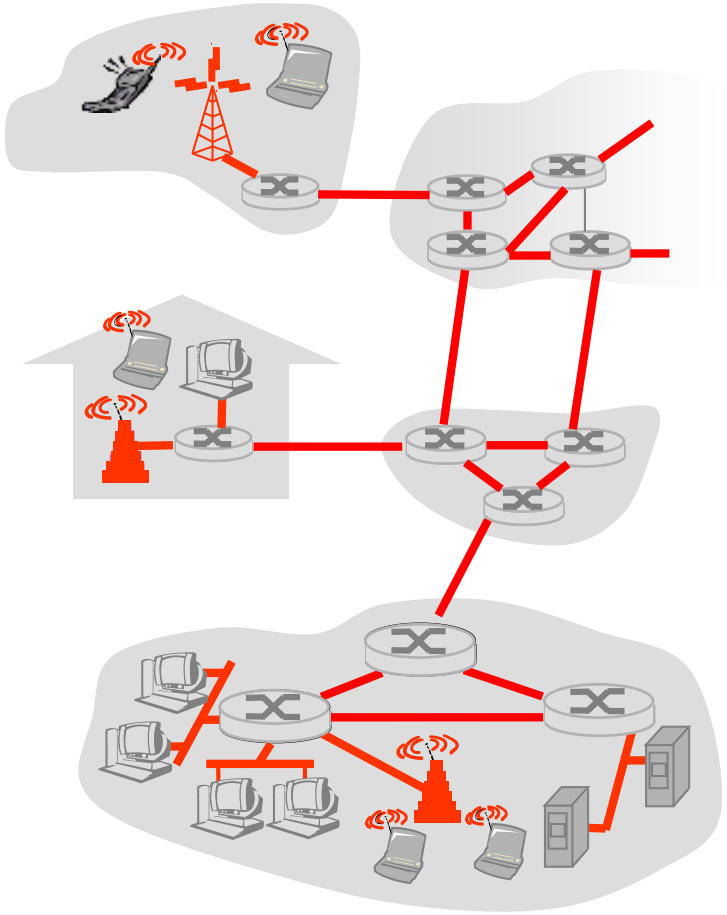
- 将数据报从一个节点传输到相邻的下一个节点

物理层：

- 多种类型的传输媒体
- 传输原始比特流（无结构）
- 容易产生传输错误

一些术语

- 节点：主机和路由器统称为节点
- 链路：连接相邻节点的通信信道
  - 有线链路
  - 无线链路
  - 局域网
- 帧：链路层分组称为帧



链路层服务

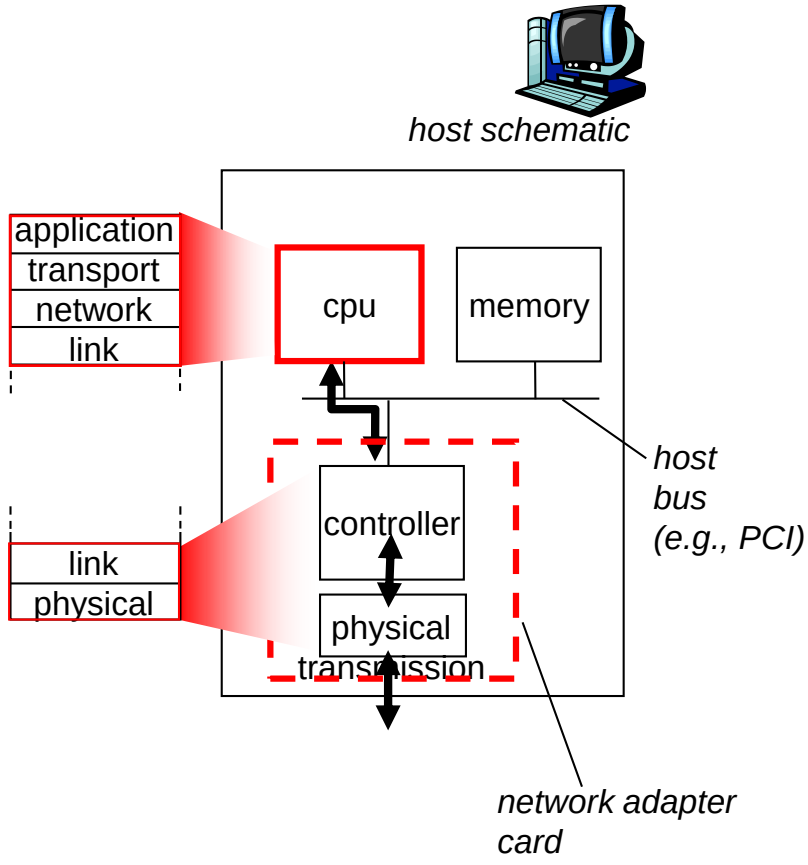
- 组帧（基本服务）
  - 发送：将数据报封装到帧中（加上帧头和帧尾）
  - 接收：从原始比特流中提取出完整的帧
- 链路接入（广播链路需要）
  - 在广播信道上协调各个节点的发送行为
- 差错检测（基本服务）
  - 检测传输错误
- 差错纠正（有些提供）
  - 检测并纠正传输错误（不使用重传）

链路层服务（续）

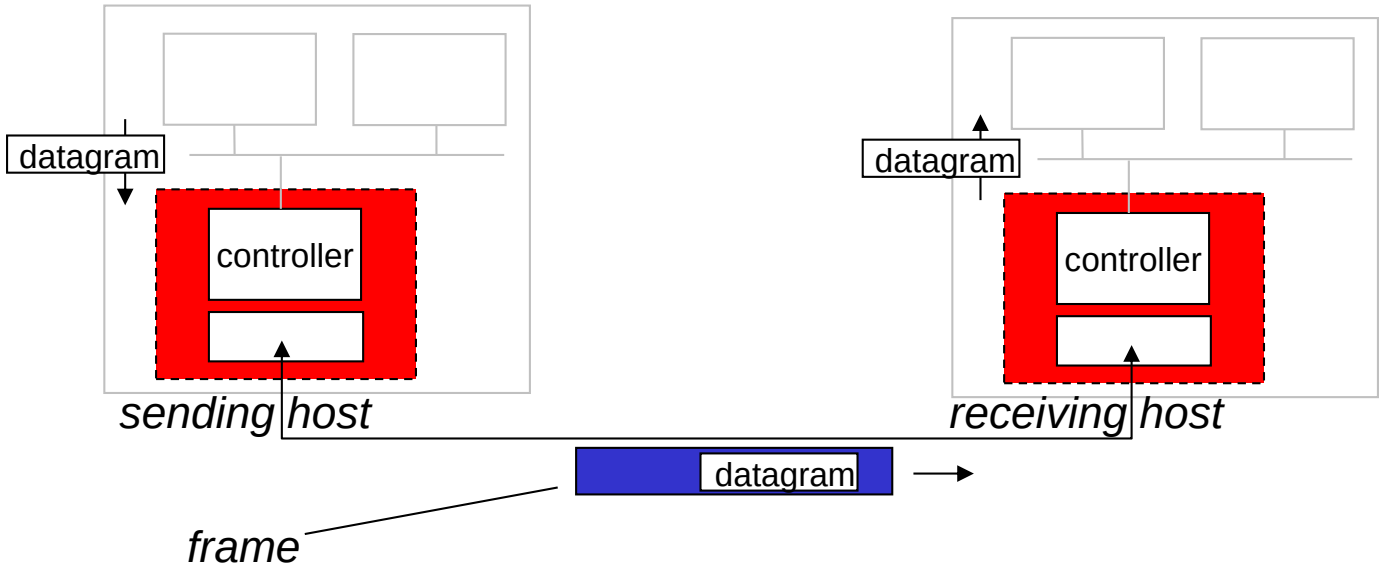
- 可靠交付（部分协议提供）
  - 通过确认、重传等机制确保接收节点正确收到每一个帧（停-等、GBN、SR）
  - 低误码率链路（如光纤、某些双绞线）上很少使用，高误码率链路（如无线链路）应当使用
- 流量控制:
  - 调节发送速度，避免接收节点缓存溢出
  - 提供可靠交付的链路层协议，不需要专门的流量控制
  - 不提供可靠交付的链路层协议，需要流量控制机制
- 半双工和全双工：
  - 半双工通信时，提供收/发转换

链路层在哪儿实现？

- 路由器：链路层在线卡（line card）中实现
- 主机：链路层主体部分在网络适配器（网卡）中实现
- 线卡/网络适配器连接物理媒体，还实现物理层的功能
- 链路层由硬件和软件实现：
  - 网卡中的控制器芯片：组帧、链路接入、检错、可靠交付、流量控制等
  - 主机上的链路层软件：与网络层接口，激活控制器硬件、响应控制器中断等



网络适配器之间的通信



- 发送侧:
  - 将数据报封装到帧中
  - 生成校验比特
  - （可选）执行可靠传输和流量控制
- 接收侧：
  - 提取帧，检测传输错误
  - （可选）执行可靠传输和流量控制
  - 解封装数据报，交给上层协议

Link layer, LANs: outline

6.1 introduction, services

6.2 error detection, correction

6.3 multiple access protocols

6.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS

6.6 data center networking

6.7 a day in the life of a web request

检错和纠错

- 传输出错的类型
  - 单个错：由随机的信道热噪声引起，一次只影响1位
  - 突发错：由瞬间的脉冲噪声引起，一次影响许多位，使用突发长度表示突发错影响的最大数据位数
- 差错控制编码的类型
  - 检错码：只能检测出传输错误的编码，不能确定出错位置，通常与反馈重传机制结合进行差错恢复
  - 纠错码：能够确定错误位置并自行纠正的编码

如何检测与纠正错误？

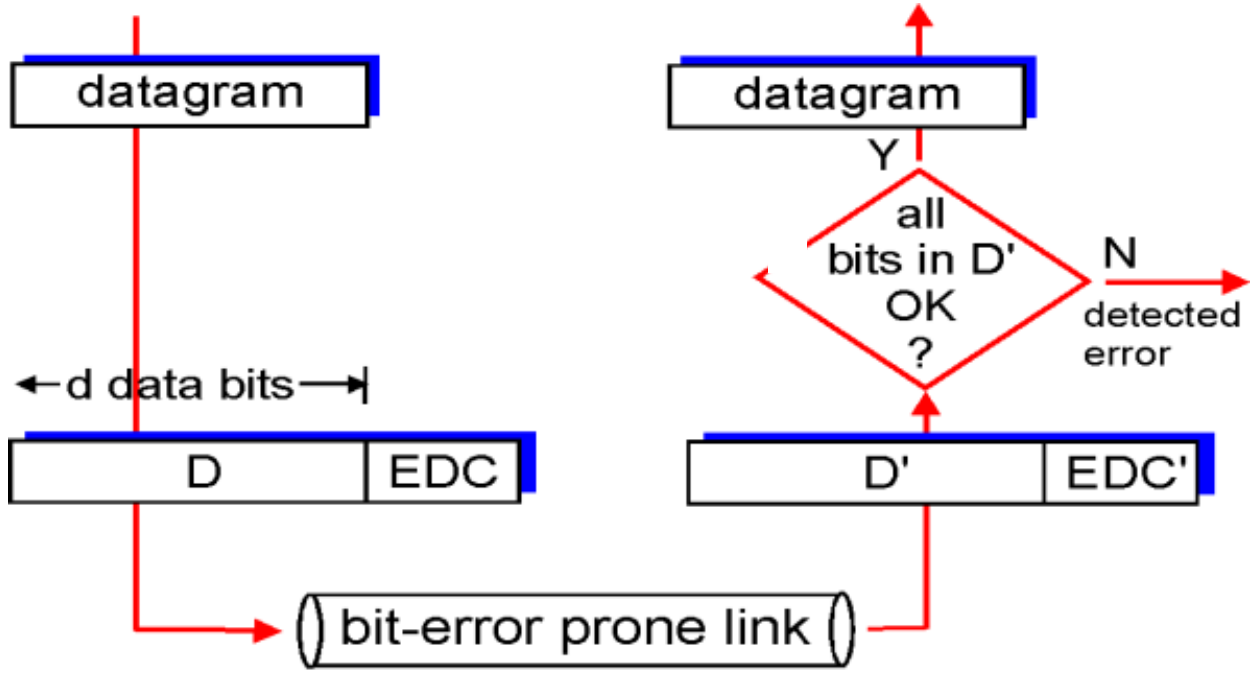
- 码字（codeword）：由 m 比特的数据加上 r 比特的冗余位（校验位）构成
- 有效编码集：由 2^m 个符合编码规则的码字组成
- 检错：若收到的码字为无效码字，判定出现传输错误
- 海明距离（Hamming Distance）：两个码字的对应位取值不同的位数（比如，100和101的海明距离为1）
- 纠错：将收到的无效码字纠正到距其最近的有效码字
- 检错码与纠错码的能力都是有限的！

编码集的检错与纠错能力

- 编码集的海明距离：编码集中任意两个有效码字的海明距离的最小值
- 检错能力：为检测出所有d比特错误，编码集的海明距离至少应为d+1
- 纠错能力：为纠正所有d比特错误，编码集的海明距离至少应为2d+1

差错检测的实施

- 发送端对要保护的数据D（包括帧头字段）生成校验位EDC，添加在帧头（尾）中
- 接收端对收到的数据D'计算校验位，与收到的校验位EDC'比较，不同则判定有错



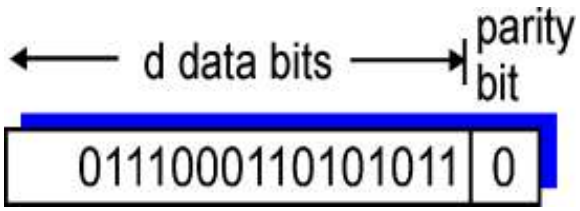
奇偶校验

单比特奇偶校验:

可检测奇数个比特错误

检错率为50%

编码集海明距离为2

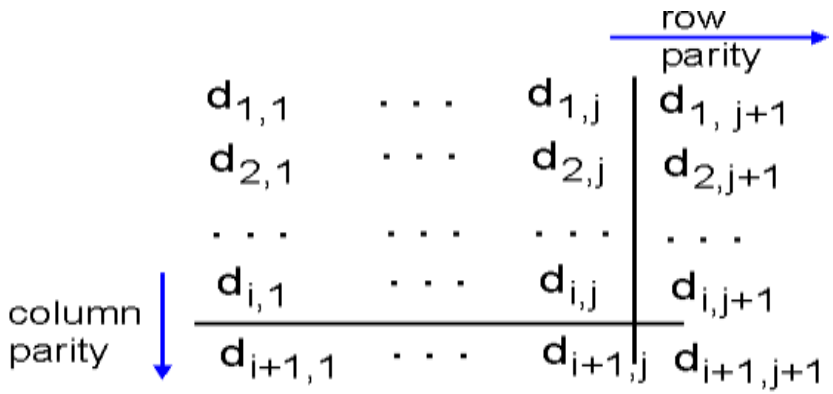


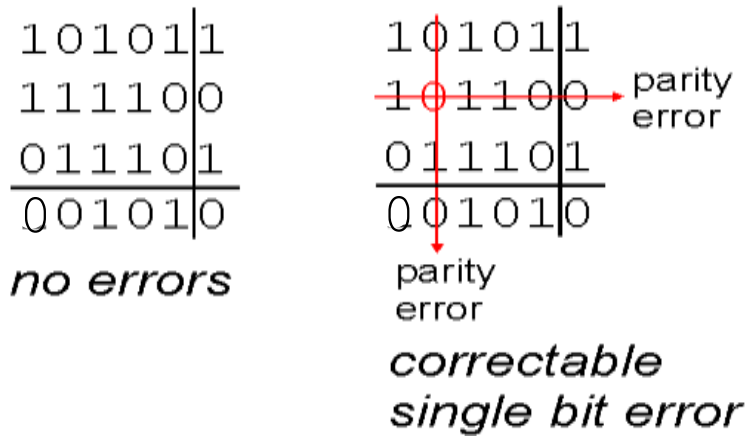
二维奇偶校验:

可检测2比特错和纠正单比特错

编码集海明距离为3

有利于检测突发错误





循环冗余校验（CRC）

- CRC是一种多项式编码，它将一个位串看成是某个一元多项式的系数，如1011看成是一元多项式$X^3 + X + 1$的系数
- 信息多项式M(x)：由m个信息比特为系数构成的多项式
- 冗余多项式R(x)：由r个冗余比特为系数构成的多项式
- 码多项式T(x)：在m个信息比特后加上r个冗余比特构成的码字所对应的多项式，表达式为$T(x) = x^r \cdot M(x) + R(x)$
- 生成多项式G(x)：双方确定用来计算R(x)的一个多项式
- 编码方法：$R(x) = x^r \cdot M(x) \div G(x)$ 的余式（减法运算定义为异或操作）
- 检验方法：若$T(x) \div G(x)$的余式为0，判定传输正确
- CRC码检错能力极强，可用硬件实现，是链路层上应用最广泛的检错码

CRC举例

例1：取$G(X) = X^3 + 1$，对信息比特101110计算CRC码。

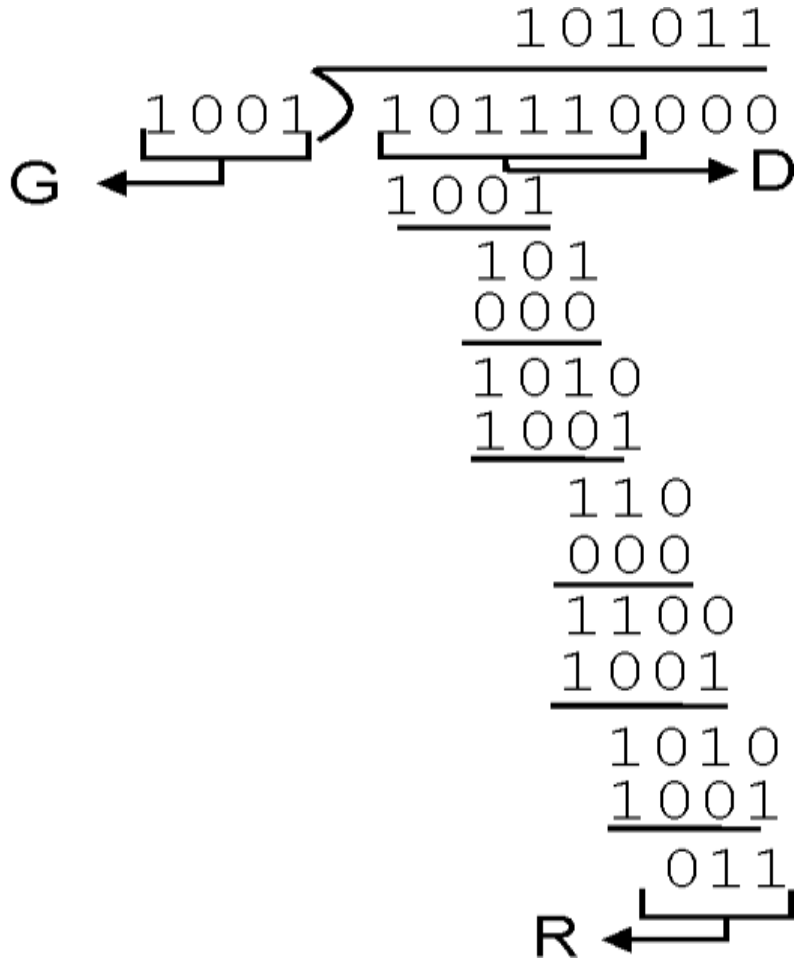
解答：

- 101110000 ÷1001的余式为R=011 (CRC code)
- 码字：101110011

例2：取$G(X) = X^3 + 1$，接收端收到比特串1001001，问是否有错?

解答：

- 1001001÷1001的余式为001（不为0），有传输错误



要求理解的知识点

- 检错和纠错的一般性原理
- 二维奇偶校验、循环冗余码
- 为什么链路层使用CRC，而其上各层使用checksum？（思考）

Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

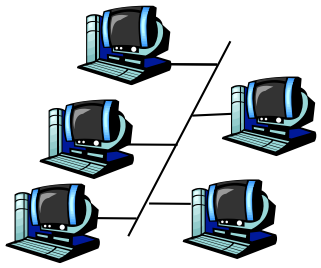
5.5 link virtualization

5.6 data center networking

5.7 a day in the life of a web request

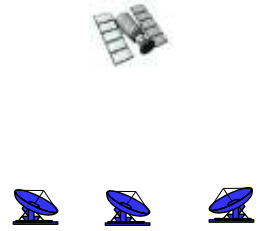
链路的两种类型

- 点到点链路：
  - 仅连接了一个发送方和一个接收方的链路
  - 一条全双工链路可以看成是由两条单工链路组成
- 广播链路：
  - 连接了许多节点的单一共享链路，任何一个节点发送的数据可被链路上的其它节点接收到



共享的电缆
（如早期以太网）

共享的无线射频
（如802.11 WiFi）



共享的无线射频
（如卫星）

humans at a
cocktail party
(shared air, acoustical)

多址接入（Multiple Access）

- 冲突（collision）
  - 在广播链路上，若两个或多个节点同时发送，发送的信号会发生干扰，导致接收失败
- 多址接入协议
  - 规定节点共享信道（谁可以发送）的方法
  - 多址接入协议也称媒体接入控制（Medium Access Control，MAC）协议

理想的多址接入协议

在速率为R bps的广播信道上

1. 当只有一个节点发送时，它应能以速率R发送（信道利用率高）
2. 当有M个节点发送时，每个节点应能以 R/M的平均速率发送（公平性好、信道利用率高）
3. 协议是无中心的（分散式）：
   - 不需要一个特殊的节点来协调发送（健壮性好）
   - 不需要时钟或时隙同步（不需要额外的机制）
4. 简单（实现和运行开销小）

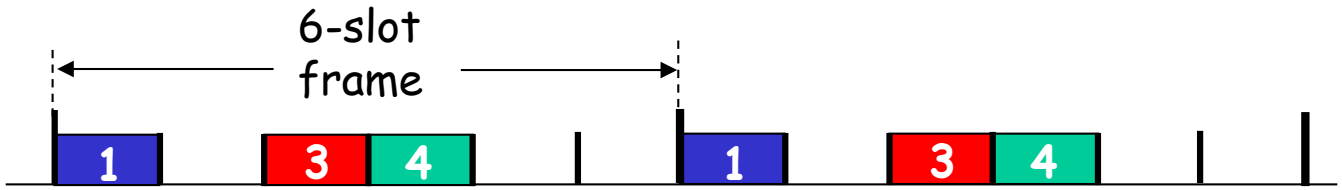
MAC协议的分类

- 信道划分
  - 将信道划分为若干子信道，每个节点固定分配一个子信道，不会发生冲突
  - 关注公平性，轻负载时信道利用率不高
- 随机接入（竞争）
  - 不划分信道，每个节点自行决定何时发送，出现冲突后设法解决
  - 轻负载时信道利用率高，重负载时冲突严重
- 轮流使用信道
  - 不划分信道，有数据的节点轮流发送，不会出现冲突
  - 信道利用率高，公平性好，但需引入额外机制

（1）信道划分的MAC协议

TDMA: 时分多址

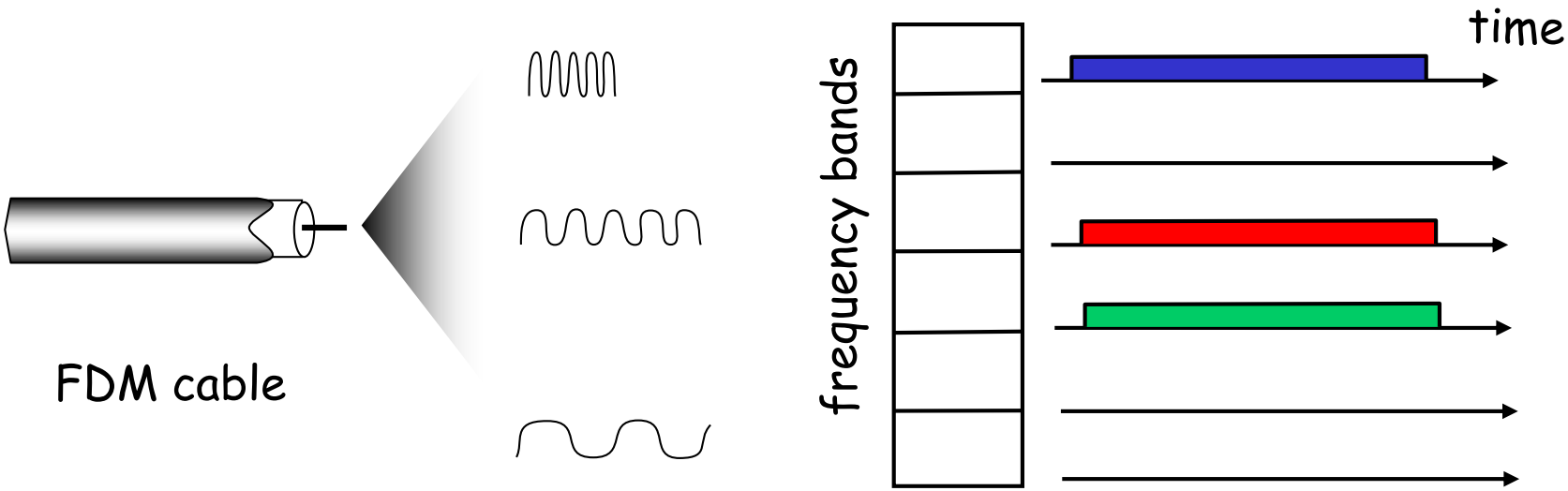
- 将信道的使用时间划分成帧，每个节点在帧中被分配一个固定长度的时间片，每个时间片可以发送一个分组
- 节点只能在分配给自己的时间片内发送
- 若节点不发送，其时间片轮空



信道划分的MAC协议

FDMA: 频分多址

- 将信道频谱划分为若干子频带
- 每个节点被分配一个固定的子频带
- 若节点不发送，其子频带空闲



信道划分的MAC协议

CDMA: 码分多址

- 将每个比特时间进一步划分为m个微时隙（称chip）
- 每个节点被分配一个惟一的m比特码序列（称chip code）
- 发送方编码：发送“1”=发送chip code；发送“0”=发送chip code的反码
- 信号叠加：多个节点发送的信号在信道中线性相加
- 接收方解码：用发送方的chip code与信道中收到的混合信号计算内积，恢复出原数据
- 前提条件：任意两个chip code必须是相互正交的

- **CDMA允许所有节点同时使用整个信道！**

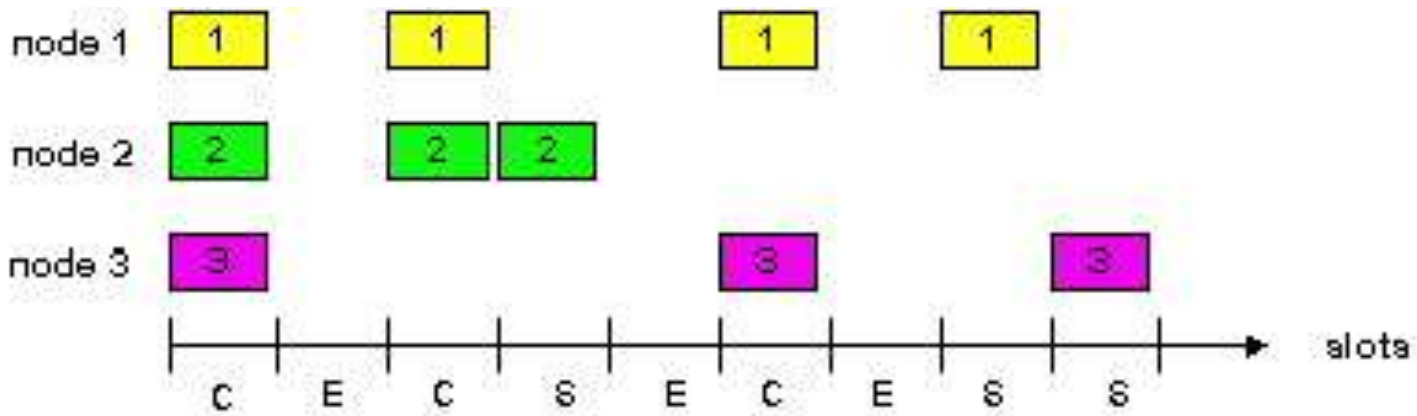
（2）随机接入的MAC协议

- 随机接入的基本思想：
  - 当节点有数据要发送时，以信道速率R发送，发送前不需要协调
  - 随机接入MAC协议规定如何检测冲突，以及如何从冲突中恢复
- 随机接入MAC协议的例子：
  - 发送前不监听信道：ALOHA家族
  - 发送前监听信道：CSMA家族

时分（Slotted） ALOHA

假设:

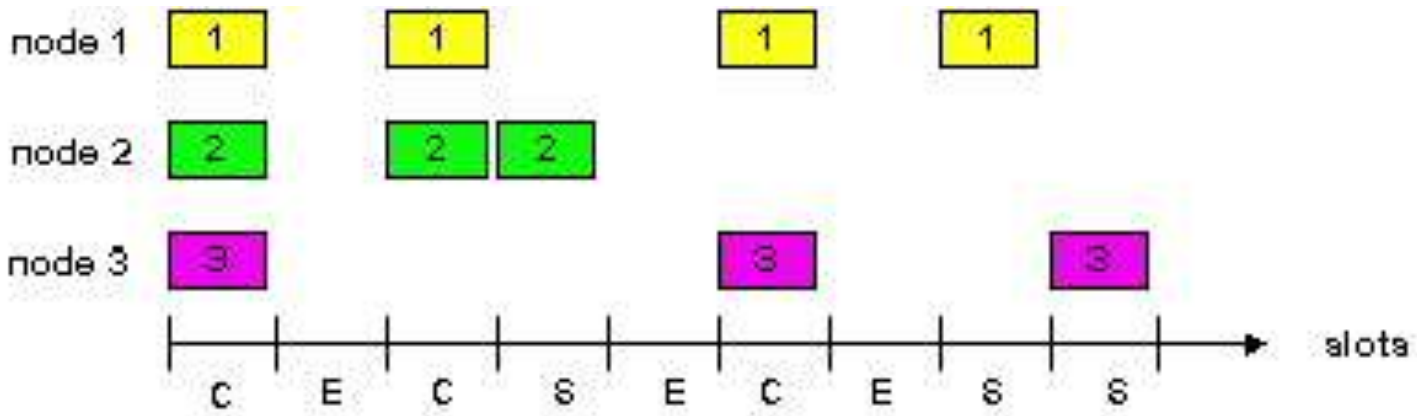
- 所有帧长度相同
- 时间被划分为等长的时隙，每个时隙传一帧
- 节点只能在时隙开始时发送
- 节点是时钟同步的（知道时隙何时开始）
- 所有节点可在时隙结束前检测到是否有冲突发生



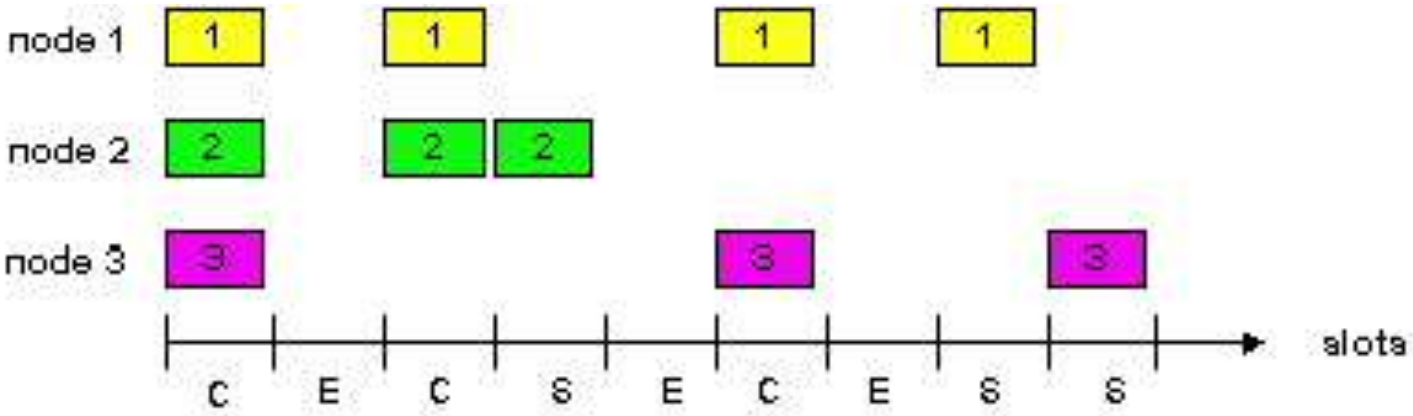
时分ALOHA

操作:

- 节点从上层收到数据后，在下一个时隙发送
- 若时隙结束前未检测到冲突，节点可在下一个时隙发送新的帧
- 若检测到冲突，节点在随后的每一个时隙中以概率P重传，直至发送成功



时分ALOHA



优点

- 单个活跃节点可以信道速率连续发送
- 分散式：节点自行决定什么时候发送
- 简单

缺点

- 发生冲突的时隙被浪费了
- 由于概率重传，有些时隙被闲置
- 需要时钟同步

时分Aloha的效率

效率：当网络中存在大量活跃节点时，长期运行过程中成功时隙所占的比例

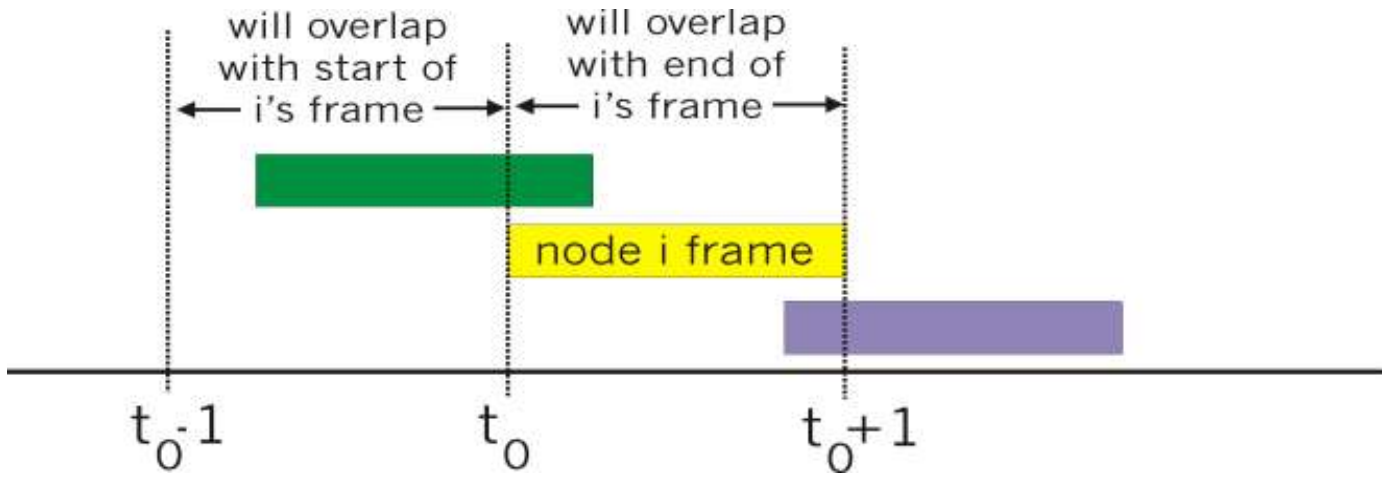
- 假设: 有N个活跃节点，每个节点在每个时隙开始时以概率P发送
- 给定节点在一个时隙中发送成功的概率 = $p(1-p)^{N-1}$
- 给定时隙中有节点发送成功的概率 = $Np(1-p)^{N-1}$

- 最大效率:
  - 找到令$Np(1-p)^{N-1}$最大的概率p^*
  - 代入$Np^*(1-p^*)^{N-1}$，并令N趋向于无穷，得到:
  - 最大效率 = $1/e = 0.37$

最佳情况: 信道用于有效传输的时间仅为37%!

纯ALOHA

- 基本思想：
  - 不需时钟同步，任何节点有数据发送就可以立即发送
  - 节点通过监听信道判断本次传输是否成功
  - 若不成功，立即以概率P重传，以概率（1-P）等待一个帧时后再决定。（帧时：发送一帧的时间，假设帧长度相同）
- 发生冲突的情形：
  - 在时刻t_0发送的帧与在 (t_0-1,t_0+1) 时段内发送的其它帧冲突



纯Aloha的效率

P(给定节点发送成功) = P(节点发送) ·

P(无其它节点在$(t_0-1,t_0]$内发送) ·

P(无其它节点在$[t_0, t_{0+1})$内发送)

$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$

$= p \cdot (1-p)^{2(N-1)}$

求出令节点发送成功概率$Np \cdot (1-p)^{2(N-1)}$最大的p^*，并令N -> infty:

最大效率 = 1/(2e) = 0.18

载波侦听多址接入（CSMA）

- 发送前监听信道（carrier sensing）：
  - 信道空闲：发送整个帧
  - 信道忙：推迟发送
- 冲突仍可能发生：
  - 由于存在传输延迟，节点可能没有监听到其它节点正在发送
  - 即使忽略传输延迟，当两个（或多个）节点同时发现信道由忙变为空闲、并都决定立即发送时，仍会发生冲突

CSMA/CD (Collision Detection)

- 若在发送的过程中检测到冲突，怎么办？
  - 继续发送余下的部分（浪费带宽）
  - 停止发送余下的部分
- CSMA/CD的基本思想：
  - 在发送的过程中检测冲突（发生冲突时信号较强）
  - 检测到冲突后，立即停止发送剩余的部分
  - 立即启动冲突解决的过程

以太网MAC协议

- 早期以太网采用CSMA/CD协议：
  1. 网卡从网络层接收数据报，构造以太帧
  2. 若网卡监听到信道空闲，立即发送帧；若信道忙，坚持监听直至发现信道空闲，然后发送帧
  3. 若网卡发送完整个帧而没有检测到冲突，认为发送成功！
  4. 若网卡在传输过程中检测到冲突，立即停止发送帧，并发送一个阻塞信号（加强冲突）

以太网MA协议（续）

5. 网卡进入指数回退阶段，选择一个等待时间：
   - 第一次冲突后：从{0,1}中选择K，延迟K· 512比特时间
   - 第二次冲突后：从{0,1,2,3}中选择K，……
   - 第三次冲突后：从{0,1,2,3,4,5,6,7}中选择K，……
   - ……
   - 第10次冲突后，从{0,1,2,3,4,…,1023}中选择K，……
6. 返回Step 2

   注：512比特是一个最小以太帧的长度

- 指数回退的目的是根据网络负载调整重传时间：
  - 负载越重（冲突次数越多），等待时间的选择范围越大，再次发生冲突的可能性越小

CSMA/CD的效率

- T_{prop} = 以太网中任意两个节点之间传播延迟的最大值
- t_{trans} = 最长帧的传输时间

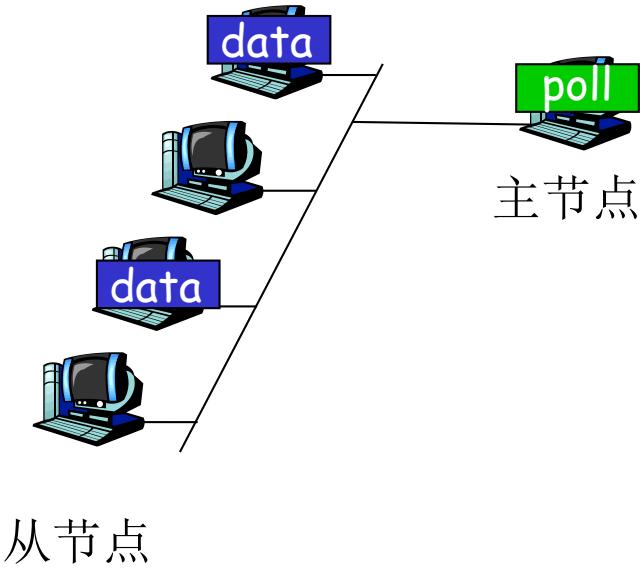
$$efficiency = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

- 在以下情况下，以太网的效率趋近于1：
  - t_{prop} 趋近于 0，或
  - t_{trans} 趋向于无穷
- 结论：应控制以太网的规模

（3）轮流MAC协议

轮询

- 主节点轮流“邀请”从节点发送，邀请到的从节点允许发送
- 缺点:
  - 引入轮询延迟
  - 单点失效（主节点）

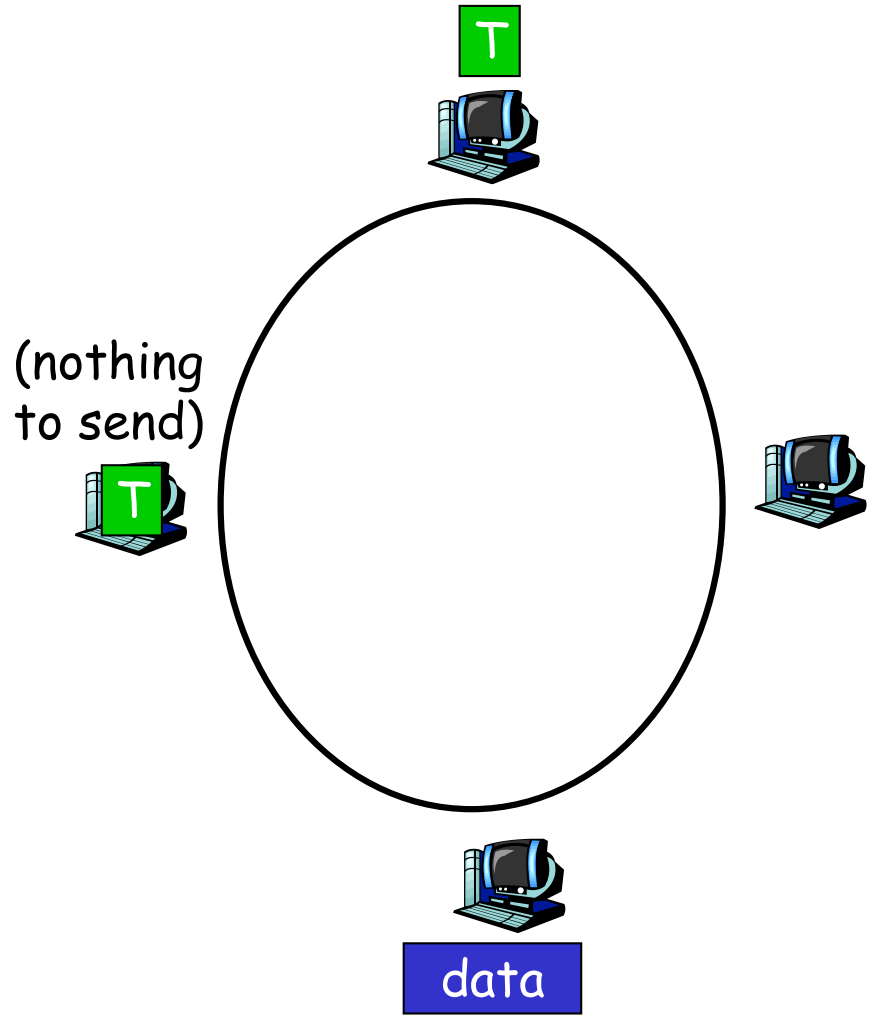


轮流MAC协议

令牌传递

- 网络中有一个令牌，按预定顺序在节点间传递
- 获得令牌的节点可以发送
- 发送完数据后释放令牌

- 缺点：
  - 令牌传递延迟
  - 单点失效（令牌）



MAC协议小结

- 按照时间、频率、编码划分信道：
  - 时分多址、频分多址、码分多址
- 随机接入：
  - 纯ALOHA, S-ALOHA（ALOHA网络）
  - CSMA/CD（早期以太网）
  - CSMA/CA（802.11）（第7章）
- 轮流访问：
  - 中心节点轮询（蓝牙）
  - 令牌传递（FDDI、IBM令牌环、令牌总线）

MAC协议比较

信道划分MAC协议：

- 重负载下高效：没有冲突，节点公平使用信道
- 轻负载下低效：即使只有一个活跃节点也只能使用1/N的带宽

随机接入MAC协议：

- 轻负载时高效：单个活跃节点可以使用整个信道
- 重负载时低效：频繁发生冲突，信道使用效率低

轮流协议（试图权衡以上两者）：

- 按需使用信道（避免轻负载下固定分配信道的低效）
- 消除竞争（避免重负载下的发送冲突）

Link layer, LANs: outline

6.1 introduction, services

6.2 error detection, correction

6.3 multiple access protocols

6.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization: MPLS

6.6 data center networking

6.7 a day in the life of a web request

局域网、城域网和广域网

- 局域网LAN（Local Area Network）
  - 将小范围内的计算机及外设连接起来的网络，范围在几公里以内，通常为个人或机构所有
- 城域网MAN（Metropolitan Area Network ）
  - 通常覆盖一个城市的范围（几十公里），要能支持数据、音频和视频在内的综合业务，服务质量好，支持用户数量多
- 广域网WAN（Wide Area Network）
  - 通常覆盖一个国家或一个洲（一百公里以上），规模和容量可任意扩大

Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization

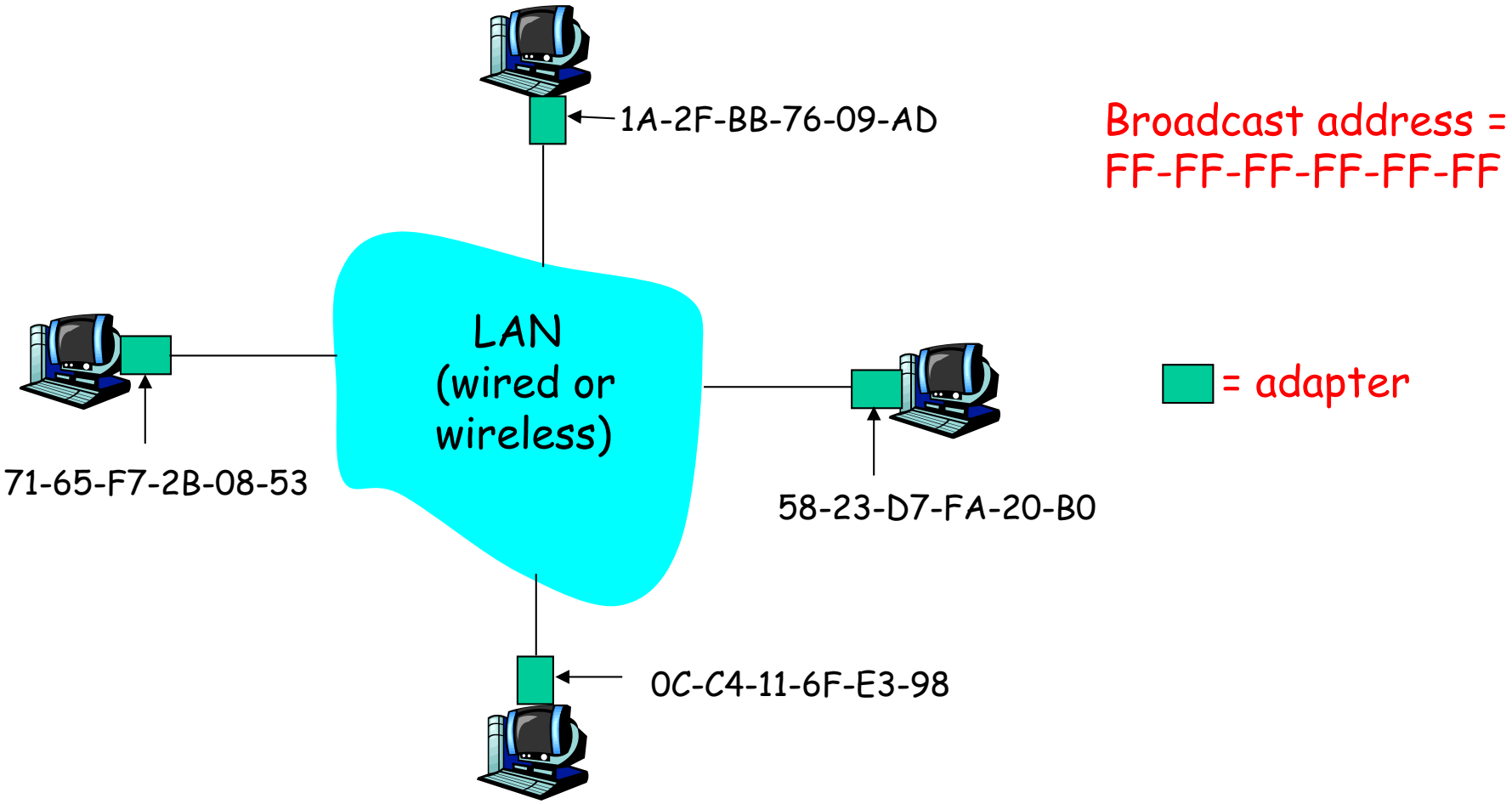
5.6 data center networking

5.7 a day in the life of a web request

链路层编址

- 早期的局域网多采用广播信道，节点如何判断收到的帧是给自己的?
- 每一块网络适配器（网卡）固定分配一个地址，称为物理地址、硬件地址、链路层地址、MAC地址等
- MAC地址长6个字节，一般用由“:”或“-”分隔的6个十六进制数表示
- MAC地址由IEEE负责分配，每块适配器的地址是全球唯一的:
  - 网卡生产商向IEEE购买一块MAC地址空间（前3字节）
  - 生产商确保生产的每一块网卡有不同的MAC地址
  - MAC地址固化在网卡的ROM中
  - 现在用软件改变网卡的MAC地址也是可能的

每个适配器有一个MAC地址



MAC地址类型

- 帧的目的MAC地址有三种类型：
  - 单播地址：适配器的MAC地址，地址最高比特为0
  - 多播地址：标识一个多播组的逻辑地址，地址最高比特为1
  - 广播地址：ff:ff:ff:ff:ff:ff
- 网络适配器仅将发送给本节点的帧交给主机：
  - 目的地址为适配器MAC地址的单播帧
  - 所有广播帧
  - 指定接收的多播帧
- 若将适配器设置成混收模式，适配器将收到的所有帧交给主机

MAC地址和IP地址

- 世界上先有MAC地址，后有IP地址
- 在TCP/IP（互联网）出现之前，只使用MAC地址在单个的物理网络中寻址
- 为什么有了MAC地址，还需要IP地址？
  - MAC地址是扁平结构的，无法在因特网范围内快速确定接口的位置
  - IP地址是有结构的，可以在因特网范围内快速确定网络接口的位置
- IP地址与MAC地址没有固定的关联关系：
  - MAC地址与网卡绑定，与节点在哪个子网无关
  - IP地址与所在子网有关，与网卡没有关系

如何将数据报发送到下一跳?

- 当发送节点A、接收节点B位于同一个物理网络上时，数据报可从A直接交付给B：
  - A的网络层将数据报、B的MAC地址交给数据链路层
  - A的数据链路层将数据报封装在一个链路层帧中，帧的目的地址=B的MAC地址
  - B的适配器收到帧，根据目的MAC地址判断是发给本机的，取出数据报交给网络层
- 问题：
  - A的网络层如何得知B的MAC地址?

地址解析（Address Resolution）

- 问题：已知IP地址，如何得到对应的MAC地址？
- 静态映射IP地址-MAC地址的缺点：
  - 主机每次使用的IP地址可能不同（DHCP）
  - 主机可能更换网卡
- 地址解析协议（ARP）用于动态获得IP地址-MAC地址映射，其基本思想是：
  - 若节点A希望获得节点B的MAC地址，节点A广播B的IP地址（地址解析请求）
  - 节点B用自己的MAC地址进行响应

ARP报文格式

0	8	16	24	32
硬件类型			协议类型	
硬件地址长度	协议地址长度		操作	
发送方硬件地址（字节0-3）				
发送方硬件地址（字节4-5）			发送方IP地址（字节0-1）	
发送方IP地址（字节2-3）			目标硬件地址（字节0-1）	
目标硬件地址（字节2-5）				
目标IP地址（字节0-3）				

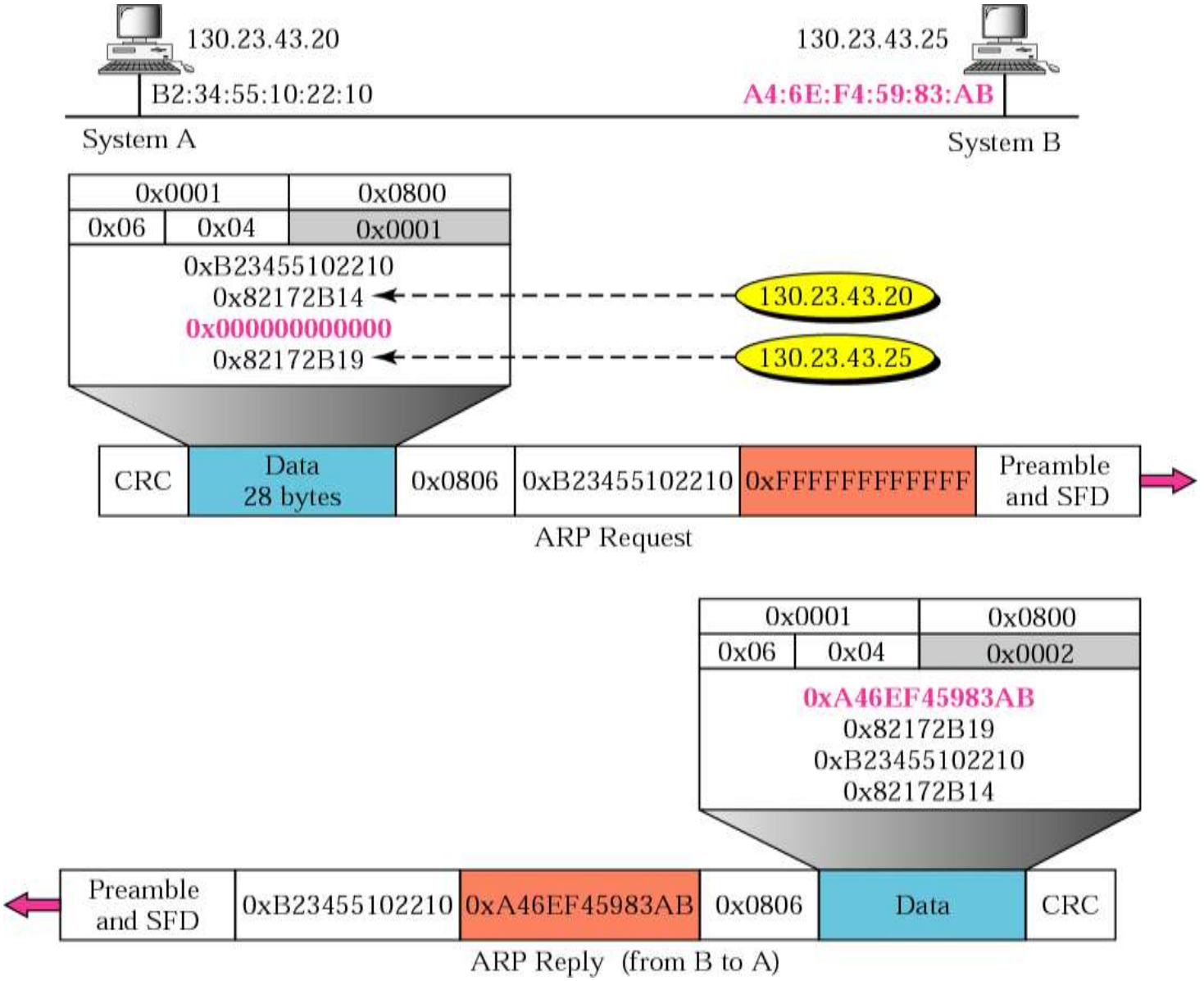
- 硬件类型：硬件接口类型。对于以太网，该值为“1”。
- 协议类型：高层协议地址类型。对于IP地址，该值为0800_{16}。
- 操作：ARP请求为1，ARP响应为2
- 在以太网上，ARP报文封装在以太帧中传输

地址解析的过程

A想知道B的MAC地址：

- A构造一个ARP请求，在发送方字段填入自己的MAC地址和IP地址，在目标字段填入B的IP地址
- A将ARP请求封装在广播帧中发送
- 每个收到ARP请求的节点用目标IP地址与自己的IP地址比较，地址相符的节点进行响应（B响应）
- B构造一个ARP响应，交换发送方与目标字段内容，在发送方硬件地址字段填入自己的MAC地址，修改操作字段为2
- B将ARP响应封装在单播帧（目的地址为A的MAC地址）中发送

IP地址为130.23.43.20、物理地址为oxB23455102210的主机，要获得IP地址为130.23.43.25的主机的MAC地址



改进ARP的措施：ARP缓存

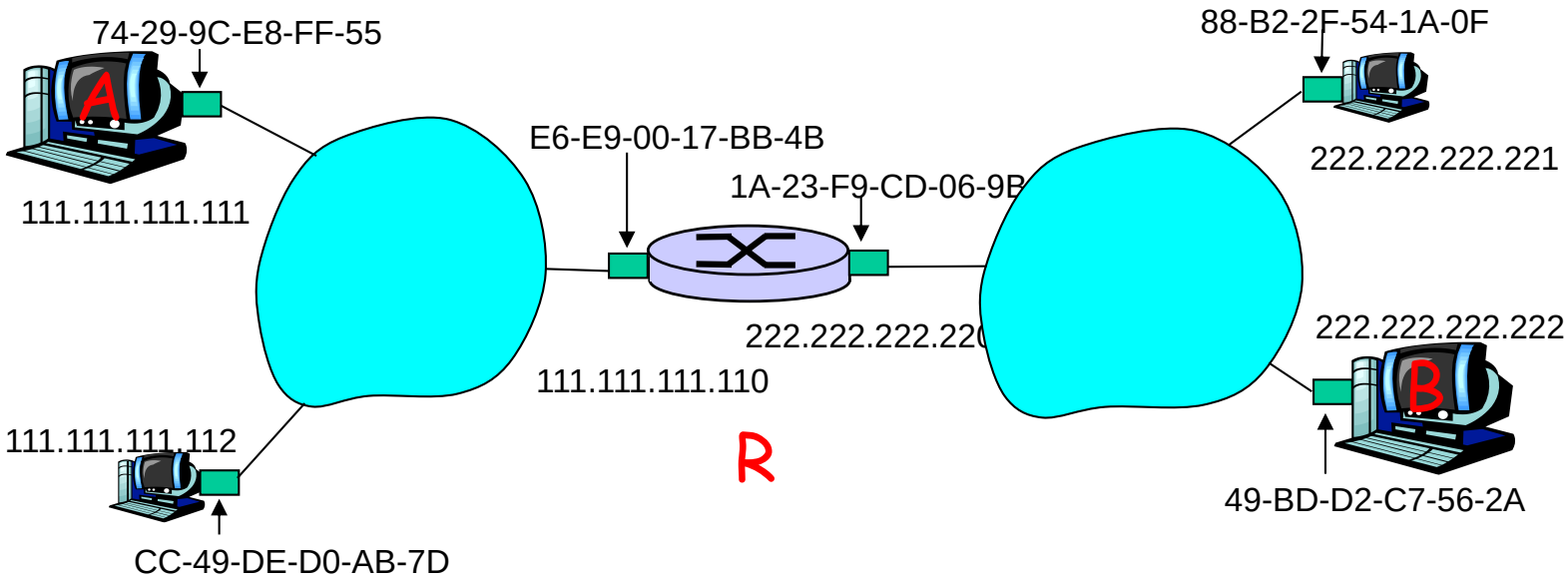
- 每个节点在内存中维护一个地址映射（绑定）表，称ARP缓存
- 每次发送数据报前先查询ARP缓存，若找不到则发送ARP请求，并在收到ARP响应后将地址映射缓存起来
- ARP缓存中的信息，在超时（一般为15～20分钟）后删除

改进ARP的措施：主动学习

- 从ARP请求中获取地址绑定信息：
  - 每个节点可以收到全部的ARP请求报文，可将发送节点的地址映射缓存到自己的ARP表中
- 节点在启动时自动广播自己的地址映射：
  - 节点A在启动时主动广播一个ARP请求，在目标字段内填入自己的IP地址
  - 收到ARP请求的节点将A的地址映射缓存起来
  - 若A收到ARP响应，报告IP地址重复错误

数据报如何从源主机到达目的主机

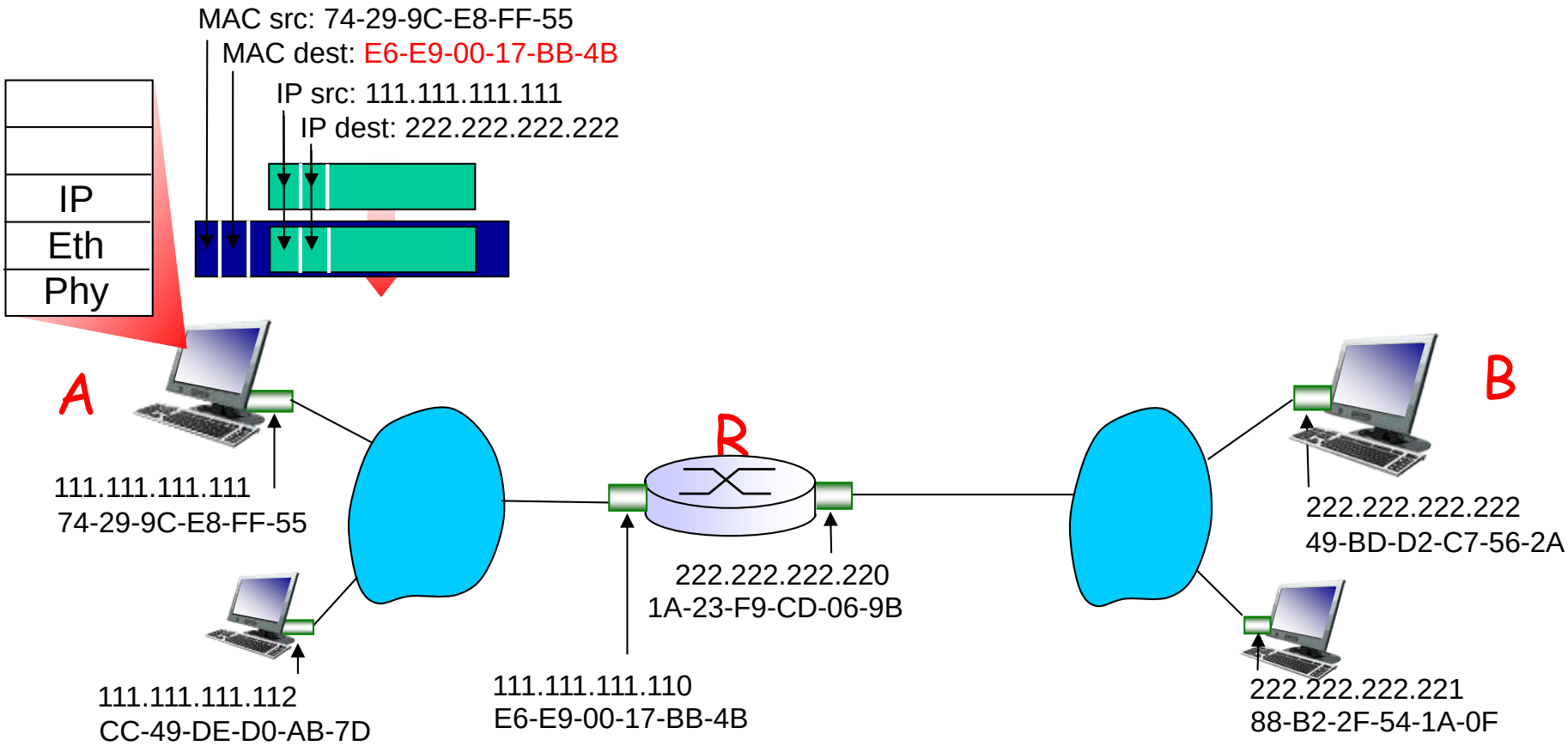
数据报从A经过R到达B：



- 根据转发表，A知道下一跳为111.111.111.110（R-1）
- 根据转发表，R知道B从其端口R-2直接可达

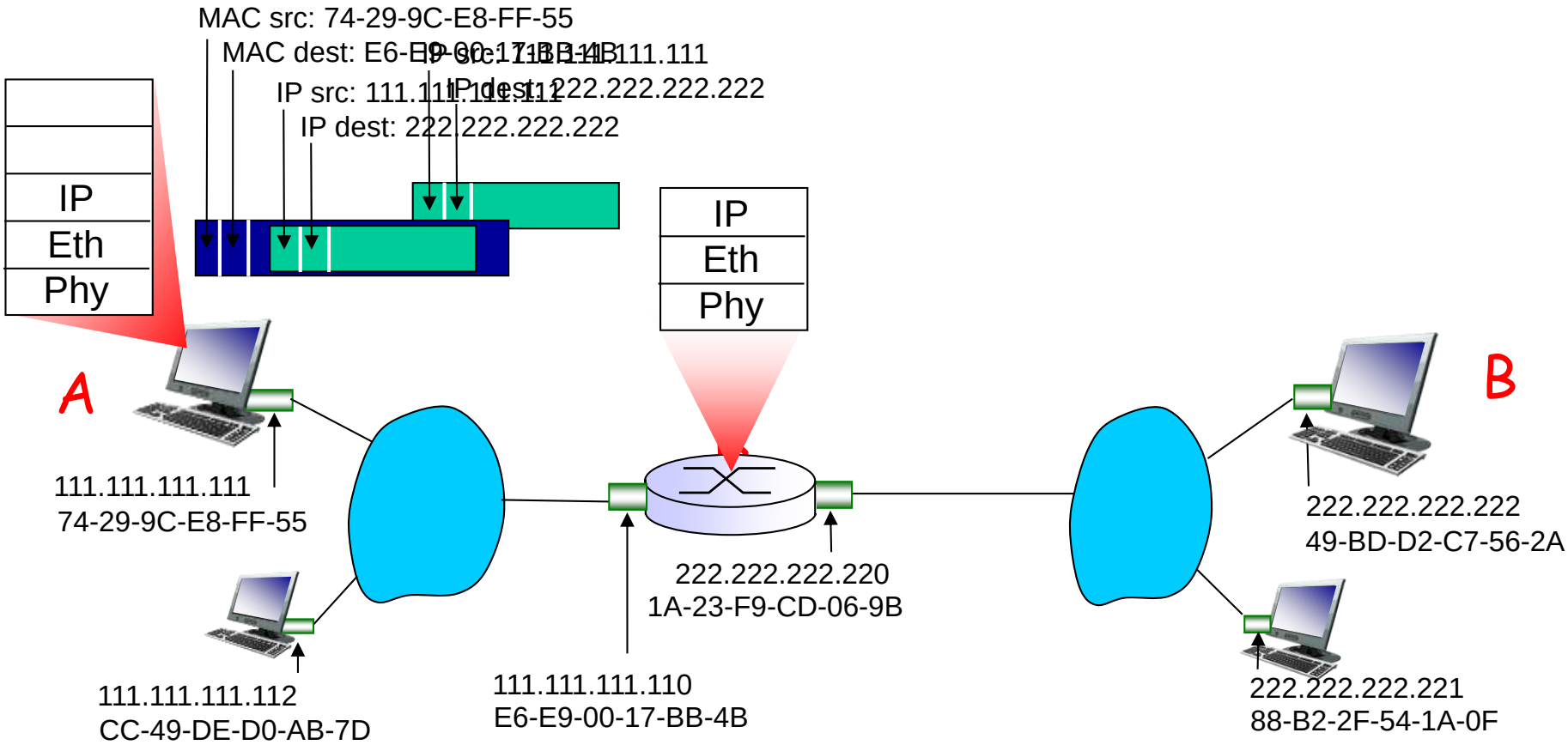
数据报如何从源主机到达目的主机

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram



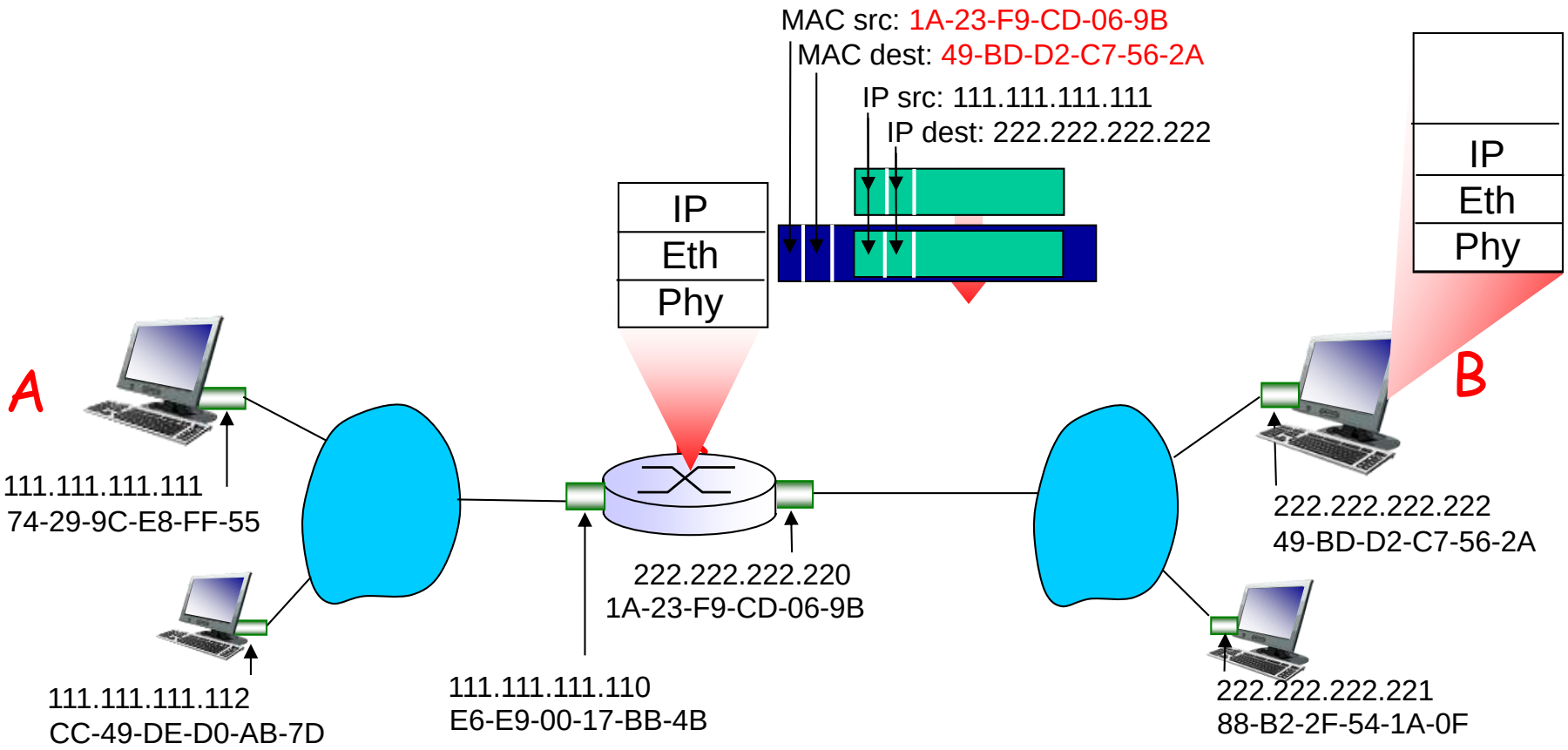
数据报如何从源主机到达目的主机

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



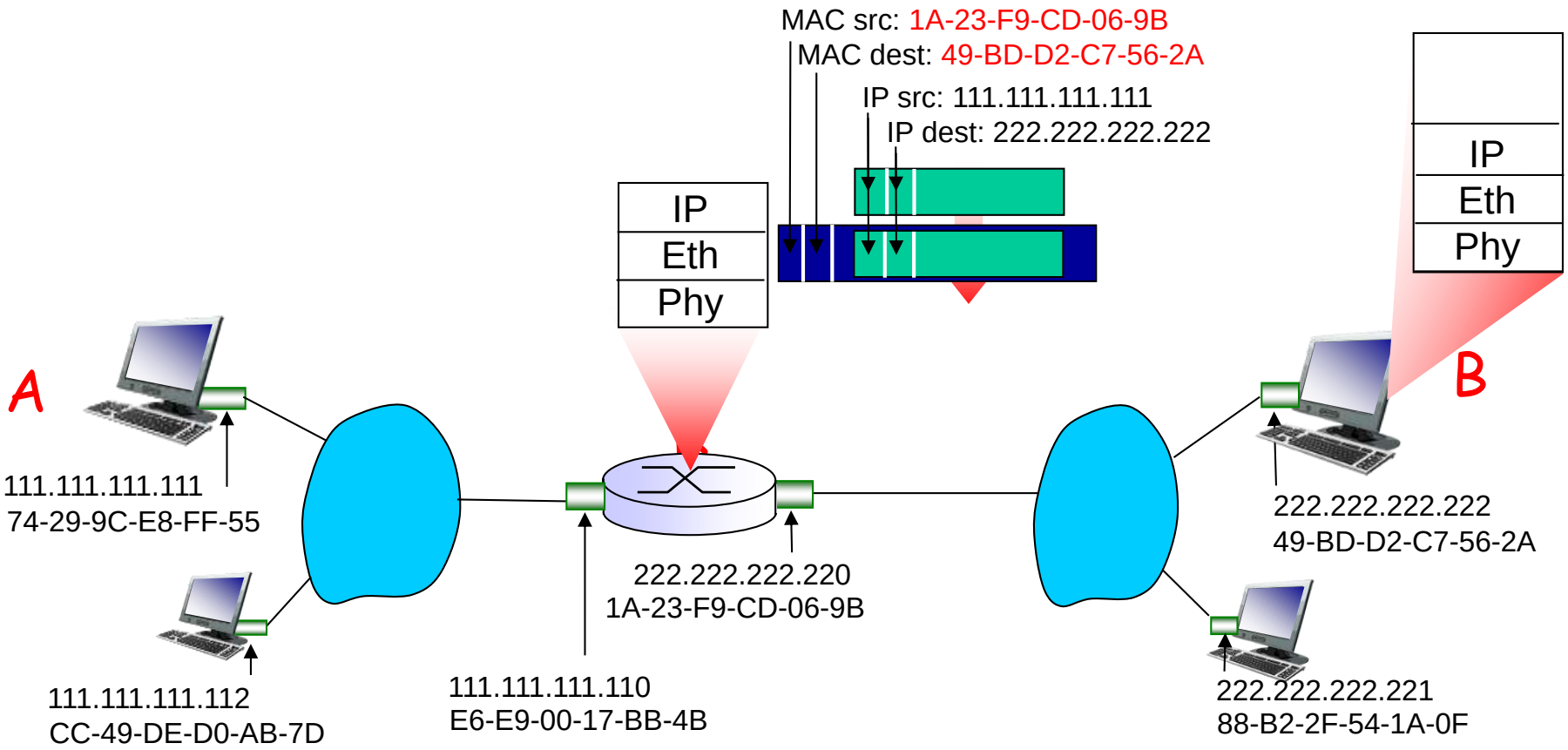
数据报如何从源主机到达目的主机

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



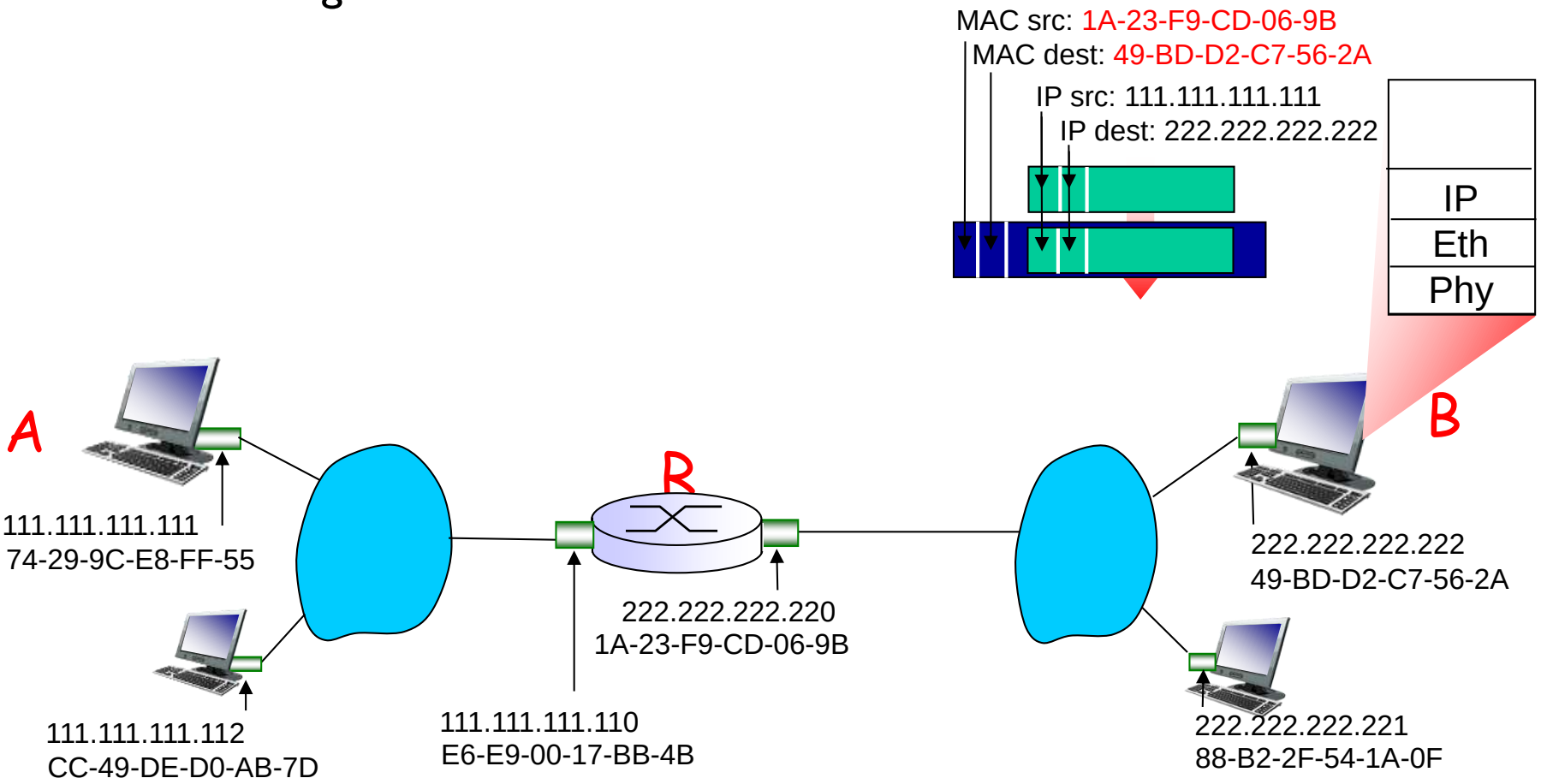
数据报如何从源主机到达目的主机

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram



数据报如何从源主机到达目的主机

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



** Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/*

重要的知识点

- 为什么有了MAC地址，还需要IP地址？
- 地址解析：
  - ARP过程，ARP缓存
- 分组逐跳转发的过程：
  - 仔细梳理源主机、路由器、目的主机上分别进行了什么操作，分组是如何逐跳地从源主机经路由器到达目的主机的

Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization

5.6 data center networking

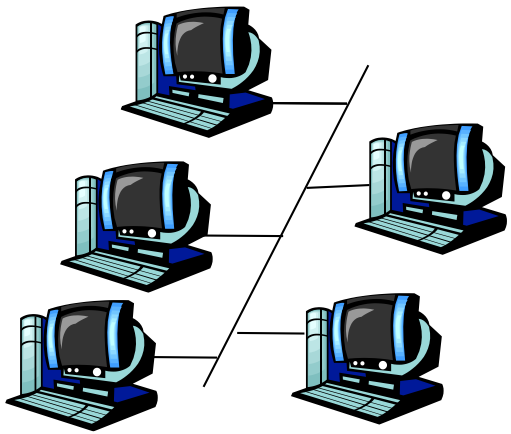
5.7 a day in the life of a web request

以太网

- 第一个广泛应用的局域网技术，也是目前占主导地位的有线局域网技术
- 与其它的局域网技术相比，技术简单、成本低
- 为提高速率，以太网技术不断演化和发展
- 速率持续提高：10 Mbps -> 100Mbps -> 1Gbps -> 10 Gbps -> 40Gbps -> 100Gbps -> ...

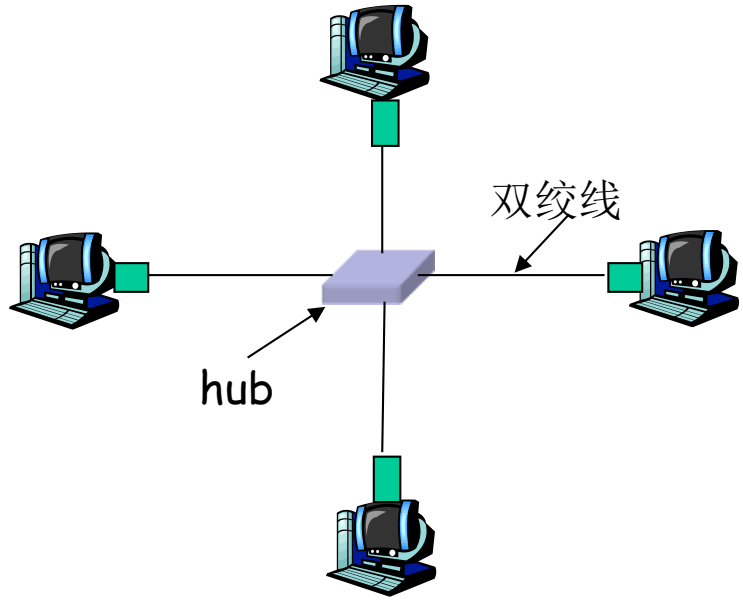
总线拓扑：共享式以太网

- 总线（1970s中期）：
  - 以同轴电缆作为共享传输媒体（总线）
  - 所有节点通过特殊接口连接到这条总线上



总线: 同轴电缆

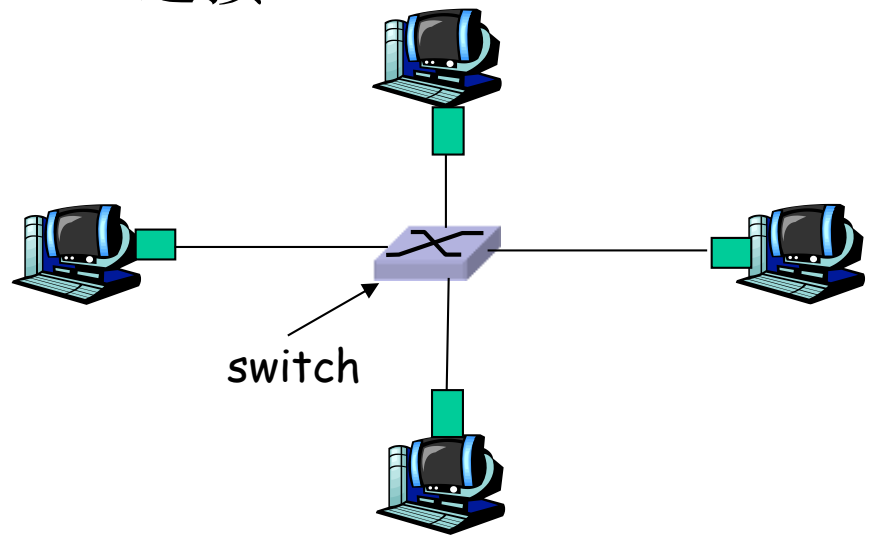
- 集线器（1990s后期）：
  - 一个物理层中继器，从一个端口进入的物理信号（光，电），放大后立即从其它端口输出
  - 集线器相当于共享电缆



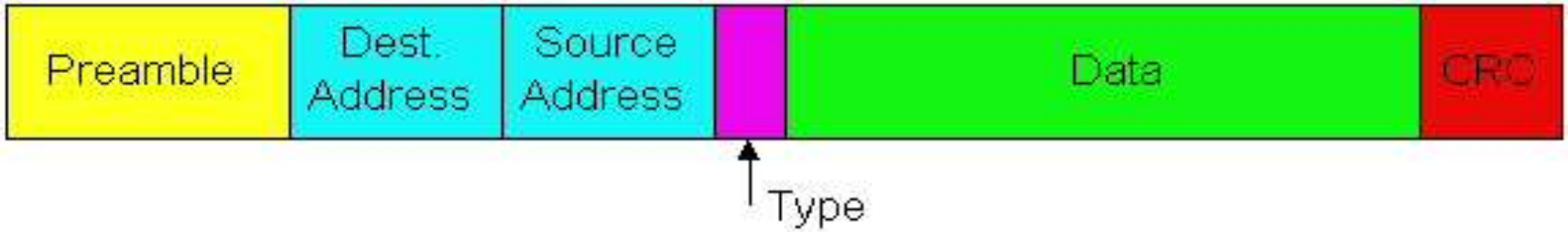
星型拓扑：交换式以太网

- 交换机（21世纪早期）：
  - 主机通过双绞线或光纤连接到交换机
  - 主机与交换机之间为全双工链路
  - 交换机在端口之间存储转发帧（链路层设备）
- **交换式以太网不会产生冲突，不需使用CSMA/CD协议！**

- 星型拓扑：
  - 各节点仅与中心节点直接通信，各节点之间不直接通信
  - 不同于基于hub的星型连接



以太帧结构



- Preamble（前导码）：
  - 7个10101010字节，后跟一个10101011字节，用于在发送方和接收方之间建立时钟同步
  - 一般不计入以太帧的长度
- Dest.Address/Src Address：目的/源MAC地址
- Type（2字节）：指出Data所属的高层协议（如IP、ARP等），每个协议有一个编号
- Data：46～1500字节，不足46字节填充至46字节
- CRC（4字节）：对dest addr.、src addr.、type和data四个字段计算得到的CRC码

无连接、不可靠的数据传输

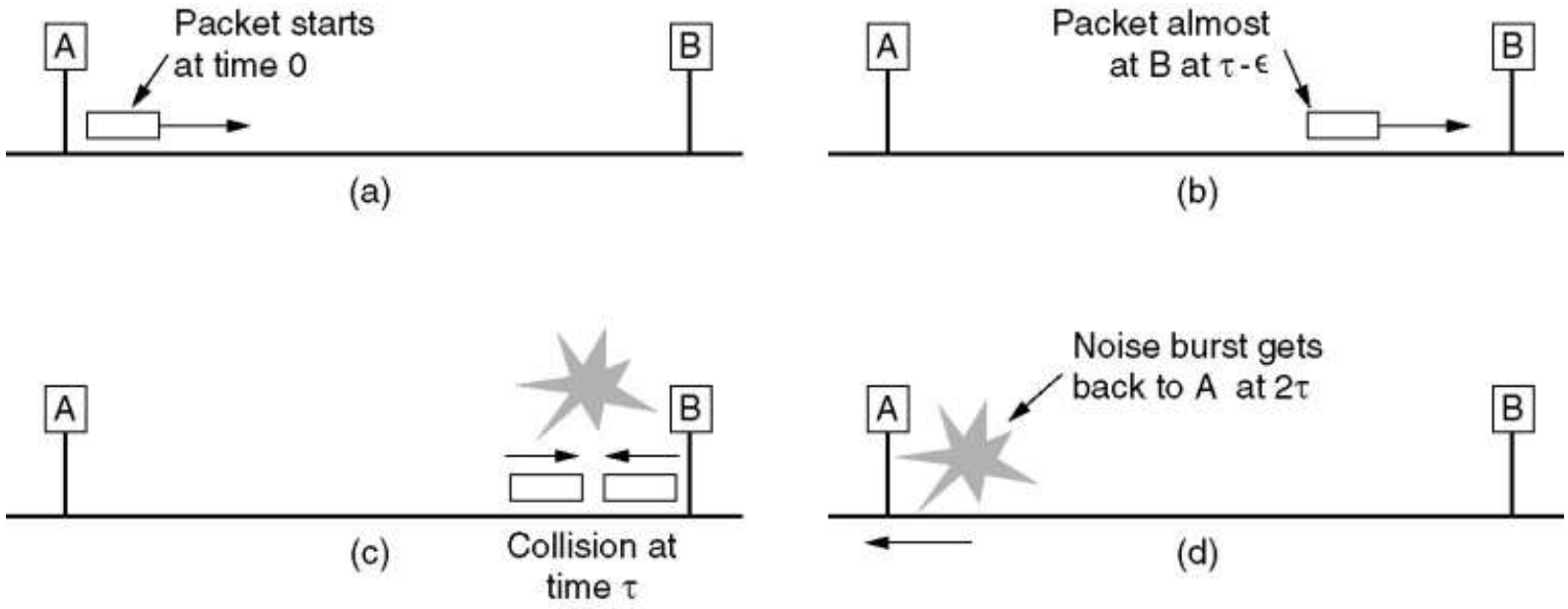
- 无连接:
  - 发送方网卡与接收方网卡之间没有握手
- 不可靠:
  - 接收方网卡不发送确认
  - 接收方网卡丢弃CRC错误的帧
  - 依靠上层协议（TCP或应用）进行错误恢复

为什么有最小帧长的要求?

- CSMA/CD协议规定，发送方仅在发送的过程中检测冲突；为保证在发送结束前检测到冲突，帧的发送时间必须足够长：
  - 节点检测冲突需要时间
  - 假设信号在相距最远的两个适配器之间的往返延迟为2τ，则帧的发送时间不应小于2τ，即帧的最小长度≧链路速率$\times 2\tau$

- 为什么最小帧长为64字节（不包括前导码）：
  - 根据早期以太网的最大直径（2500米）和数据速率（10Mbps）计算得到

检测冲突需要时间



Collision detection can take as long as 2τ.

802.3以太网标准：链路层 & 物理层

- 历史上出现过许多不同的以太网技术：
  - 链路层相同：MAC协议，帧格式，帧处理
  - 物理层不同：
    - 传输媒体：光纤，同轴电缆，双绞线
    - 数据速率：如10Mbps, 100 Mbps, 1Gbps, …
    - 物理层编码方式
- 所有这些以太网技术由IEEE 802.3工作组标准化，形成IEEE 802.3标准族

10Mbps以太网（早期以太网）

- 10Base-5：
  - 基带同轴电缆（粗），每段电缆最大长度500米
- 10Base-2：
  - 基带同轴电缆（细），每段电缆最大长度约200米
- 10Base-T
  - 3类双绞线和集线器，双绞线最大长度100米
- 10Base-F
  - 多模光纤和集线器，光纤最大长度2000米

100Mbps以太网（快速以太网）

仅能使用光纤/双绞线，以及集线器/交换机

- 100Base-TX（可使用集线器或交换机）：
  - 5类双绞线（2对），不超过100米
- 100Base-T4（可使用集线器或交换机）：
  - 3类双绞线（4对），不超过100米
- 100Base-FX（只能使用交换机）：
  - 多模光纤（2条），不超过2000米

千兆、万兆以太网

使用交换机，并增加了对流量控制的支持

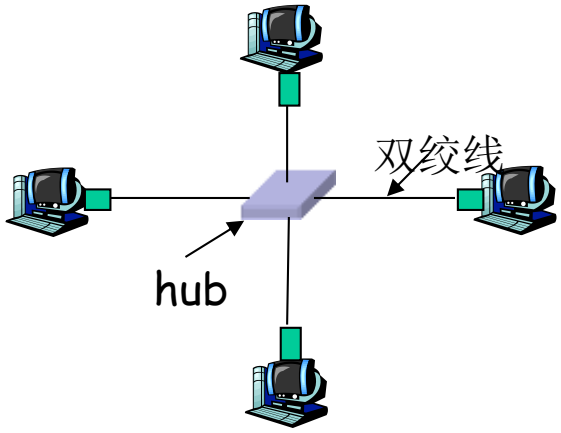
- 1000Base-SX：
  - 多模光纤，不超过550米
- 1000Base-LX：
  - 单模或多模光纤，不超过5000米
- 1000Base-CX（很少用）：
  - 2对屏蔽双绞线，不超过25米
- 1000Base-T：
  - 4对5类双绞线，不超过100米
- 10GBase-T：
  - 只使用光纤，长距离用单模光纤，短距离用多模光纤

DIX以太帧与802.3帧

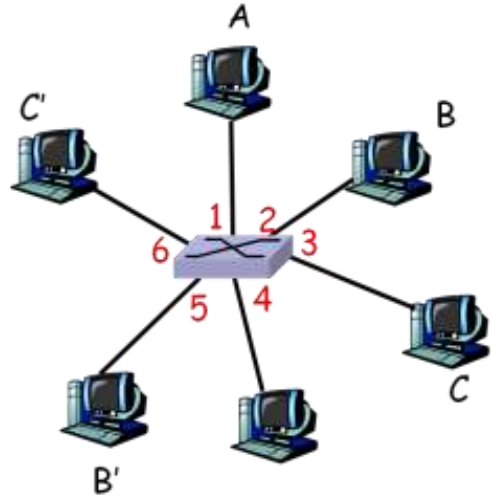
- 最早提出的以太帧称为DIX（DEC-Intel-Xerox）以太帧：
  - type：指出处理data域的协议实体
- 符合IEEE 802.3标准的帧（802.3帧）：
  - length：替代DIX帧中的type域，指出data的长度
- 这两种格式都可使用，当type/length的值大于1500时解释为type，否则解释为length

讨论：共享式以太网和交换式以太网

- 共享式以太网：
  - 集线器的所有端口位于同一个冲突域
  - 任一时刻最多只允许一个主机发送
  - 网络规模（节点数量）与网络性能的矛盾无法解决



- 交换式以太网：
  - 交换机的每个端口为一个冲突域
  - 多对端口可以同时通信
  - 网络的集合带宽=各个端口的带宽之和
  - 从根本上解决了网络规模与网络性能的矛盾



交换式以太网的最小帧长及规模

- 交换式以太网不再使用CSMA/CD协议，理论上不再需要限制帧的最小长度。但为了向后兼容，帧格式及最小帧长度的限制仍然保持不变
- 由于交换式以太网不再使用CSMA/CD协议，网络直径不再受到信号最大往返时间的限制
- 交换式以太网的MAC层除了帧格式保持不变外，其它都和共享式以太网不同了

Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

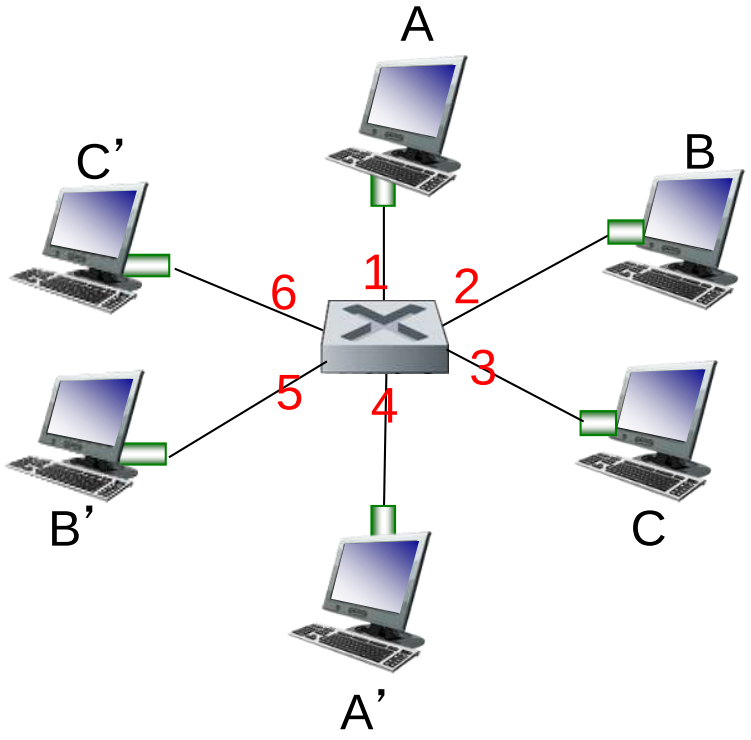
5.5 link virtualization

5.6 data center networking

5.7 a day in the life of a web request

以太网交换机

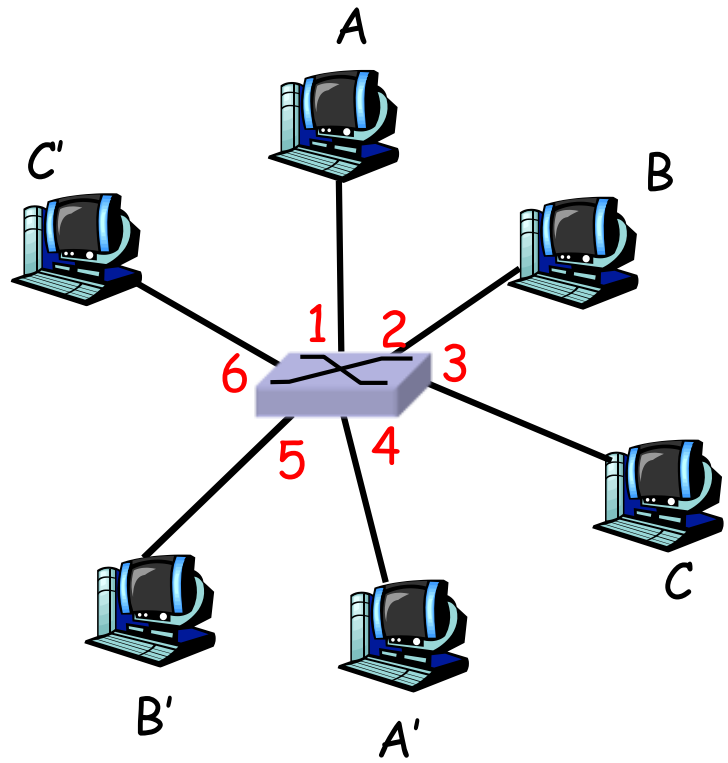
- 链路层设备：
  - 存储-转发帧：检查输入帧的MAC地址，有选择地将帧转发到一条或多条输出链路
- 对主机透明
  - 交换机没有MAC地址，主机不需要了解有关交换机的任何信息，感觉不到交换机的存在
- 即插即用，自主学习
  - 交换机不需要配置



switch with six interfaces (1,2,3,4,5,6)

交换机如何转发？

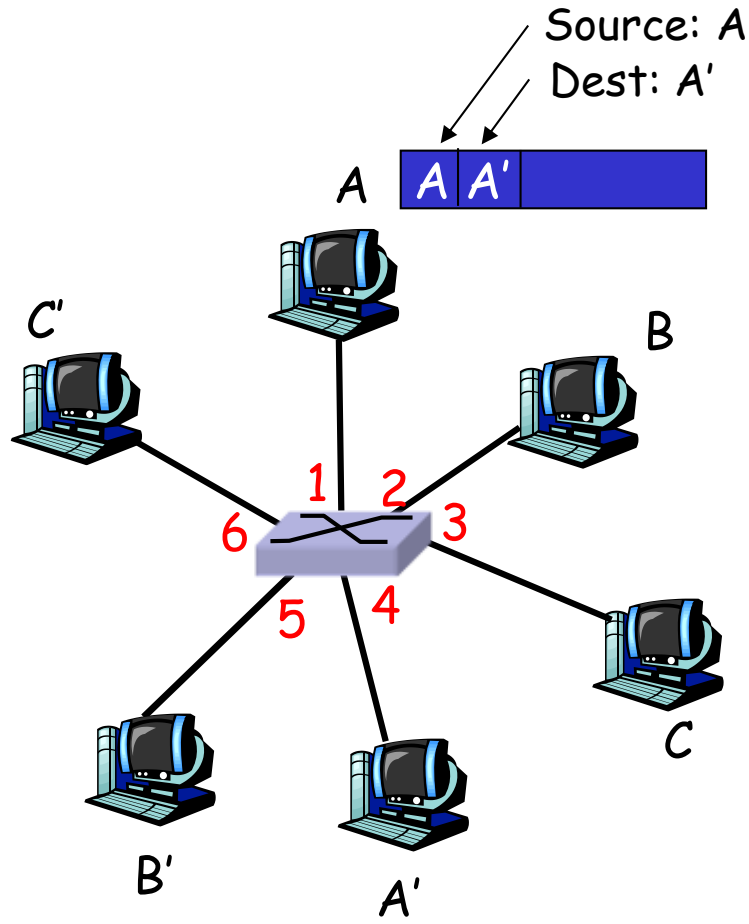
- Q: 交换机如何知道A'通过端口4可达，而B'通过端口5可达？
- A: 每个交换机内部有一张转发表，每个表项记录以下信息:
  - MAC地址，去往该MAC地址的端口
- Q: 转发表是如何建立和维护的？



switch with six interfaces (1,2,3,4,5,6)

自主学习

- 交换机自主学习“哪个主机通过哪个端口可达”：
  - 当一个帧到达时，交换机从源MAC地址了解到发送节点，从帧到来的端口了解到发送节点的位置（从该端口可达）
  - 在转发表中记录发送节点和可达端口



MAC addr	interface	TTL
A	1	60

转发表
（初始为空）

帧的过滤和转发

当帧到来时：

1. 记录帧的到来端口（自学习）
2. 用帧的目的MAC地址查找转发表
3. if 找到目的MAC地址　//已知节点
   then {
     if 目的地址所在端口=帧的到来端口
       then 丢弃帧　//过滤不需要转发的帧
     else 转发帧到表项指定的端口　//按转发表转发帧
   }
   else 扩散帧　//未知节点，采用扩散法转发

向输入端口以外的所有端口转发

交换机收到帧的处理过程

- 用帧的目的地址查找转发表（转发决策）：
  - 若目的地址所在端口 = 帧的进入端口，丢弃帧
  - 若目的地址所在端口 ≠ 帧的进入端口，转发帧
  - 若目的地址不在转发表中，扩散帧
- 用帧的源地址查找转发表（更新转发表）：
  - 若找到地址，更新相应表项
  - 若没有找到该地址，添加源地址和进入端口到转发表，设置表项的生存期为最大值

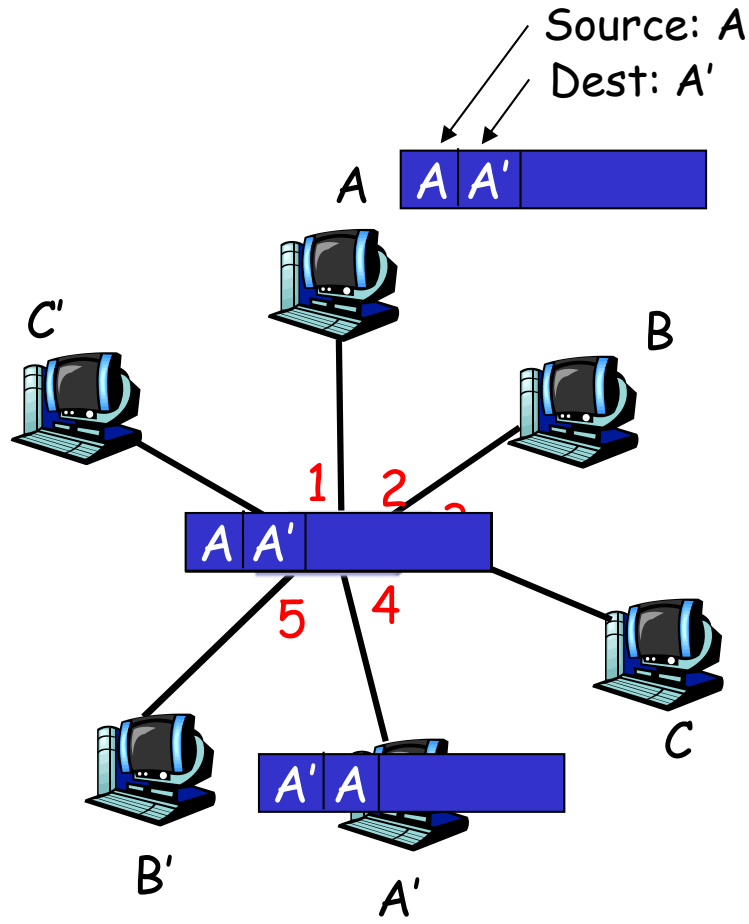
举例

- 目的地址未知：

  扩散

- 目的地址A已知：

  按照转发表转发

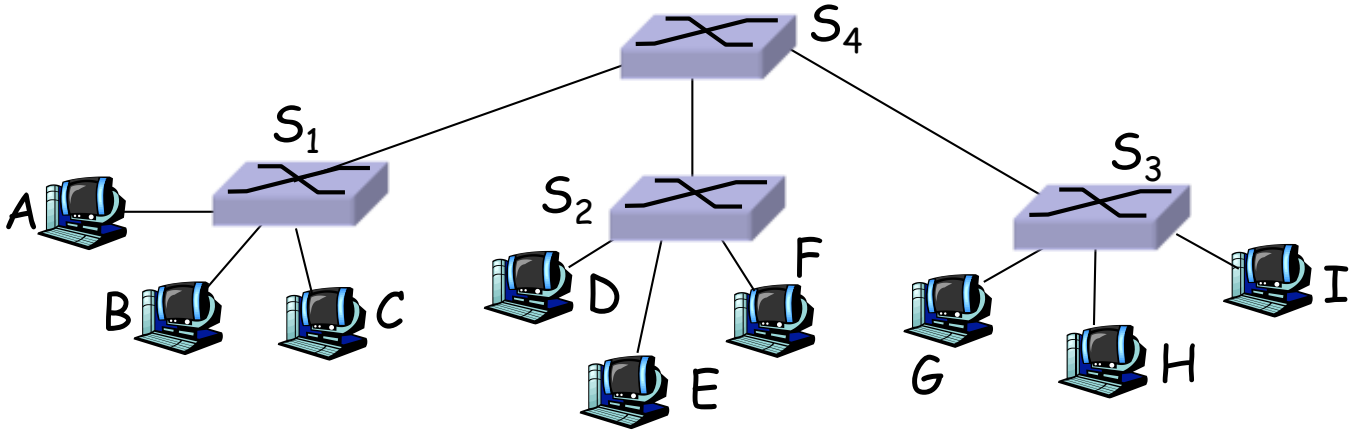


MAC addr	interface	TTL
A	1	60
A'	4	60

转发表
（初始为空）

级联交换机

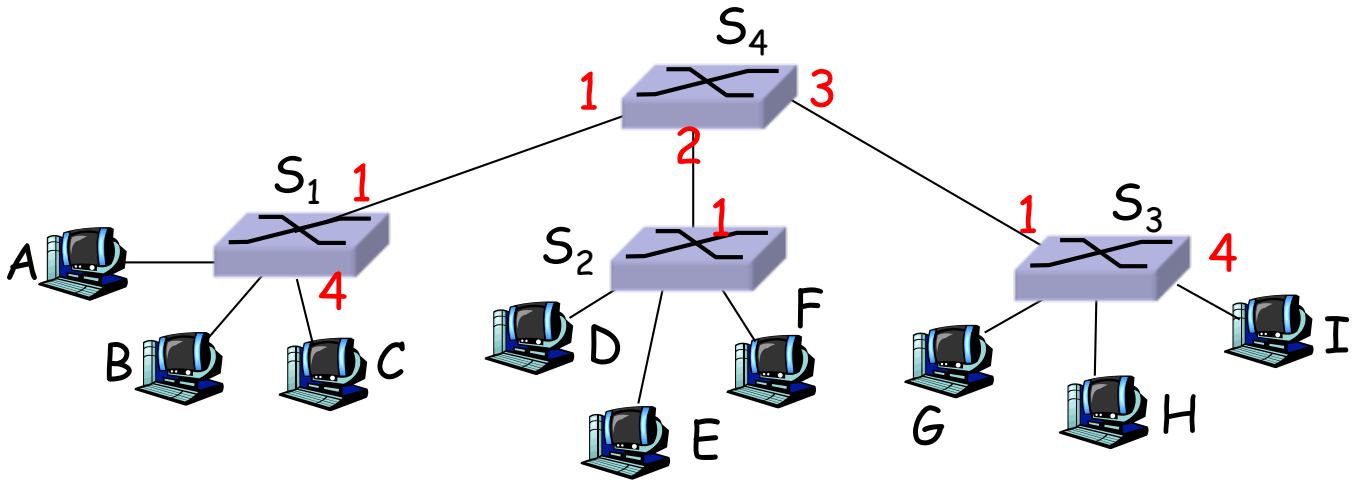
- 多个交换机也可以级联在一起，将更多或更大范围内的节点连接到一个网段中



- Q: 数据包要从A发往F，交换机S_1如何知道应转发给S_4，而S_4如何知道应转发给S_2？
- A: 通过自主学习！（与单交换机情形相同）

举例

假设C发送一个帧给I，I响应C：

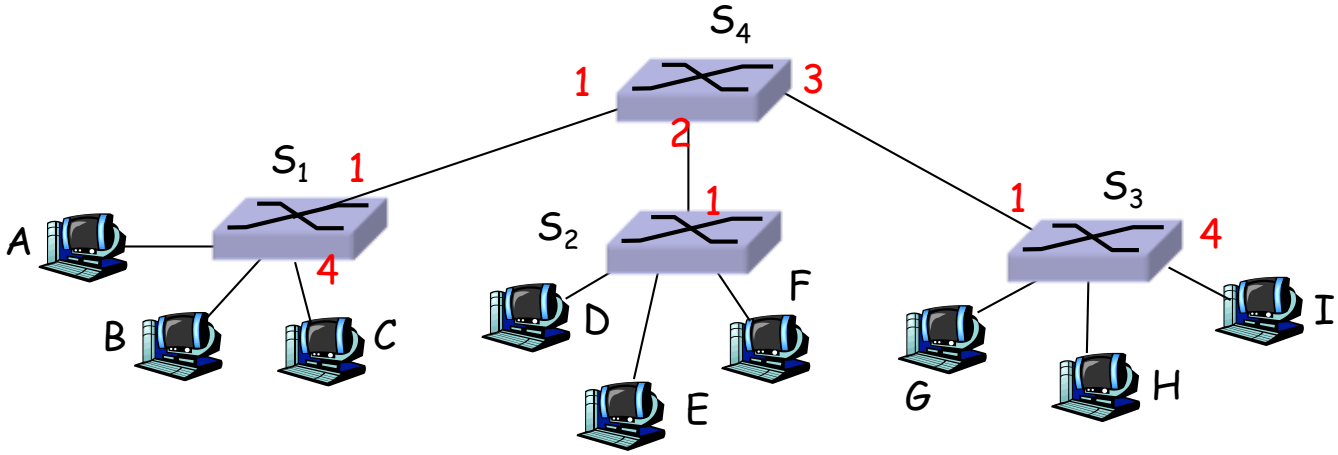


- Q: 给出 S_1, S_2, S_3, S_4 中的转发表和包转发决策

解答

1、C发送一个帧给I

2、I发送一个帧给C



S1

MAC addr	interface	action
C	4	扩散
I	1	转发

S2

MAC addr	interface	action
C	1	扩散

S3

MAC addr	interface	action
C	1	扩散
I	4	转发

S4

MAC addr	interface	action
C	1	扩散
I	3	转发

有环网络和生成树算法

- 实际的网络是有环网络：
  - 实际网络不采用树状结构（可靠性不高），而是存在冗余链路，即网络中存在环
  - 在有环的网络中扩散帧会造成冗余传输，且不能终止
- 解决方法（构造网络的生成树）：
  - 在有环的物理网络上建立一个无环的网络拓扑（生成树），正常情况下只使用无环拓扑转发帧
- IEEE 802.1D标准化了构造生成树的分布式算法：
  - 首先选举具有最小序列号的交换机作为生成树的根
  - 按照根到各个交换机的最短路径构造生成树
  - 只有位于生成树中的交换机可以在属于生成树的边上发送
  - 当有节点或链路发生故障时，重新计算一个无环拓扑

交换机 vs. 路由器

- 交换机工作于链路层，根据MAC地址存储转发帧
- 路由器工作于网络层，根据IP地址存储转发数据报

- 交换机不能连接异构链路（即MAC协议不同的网络），因为交换机只是按原样转发帧
- 路由器可以连接异构链路，因为路由器需重新封装链路层帧

交换机 vs. 路由器

- 交换机不能阻断广播帧的传播：
  - 交换机只能学习到单播MAC地址，所有广播帧都会扩散发送
  - 通过交换机连接的所有主机在同一个广播域中
- 路由器可以阻断广播帧的传播：
  - 路由器根据IP地址转发包（看不到MAC地址）
  - 每个路由器端口是一个独立的广播域

- 冲突域：
  - 共享同一条广播链路的主机集合
  - 任何一个主机发送的帧（各种帧），可被冲突域中的其它主机接收到
- 广播域：
  - 广播帧能够到达的主机集合
  - 广播风暴：广播帧在网络中大量传播，消耗大量资源

三层交换机和路由器

- 路由器可分隔二层网络，但转发速度慢、成本高
- 三层交换机：
  - 具有部分路由功能、又有二层转发速度的交换机
  - 专为加快大型局域网内部的数据交换而设计
  - 但在安全、协议支持等方面不如专业路由器
- 机构网络中三层交换机和路由器的使用：
  - 三层交换机：通常用在机构网络的核心层，连接不同的子网或虚拟局域网（每个虚拟局域网是一个独立的子网）
  - 专业路由器：连接机构网络与外网

三层交换机为什么快？

- 路由器转发IP包的过程：
  1. 用目的IP地址查找转发表，获得下一跳IP地址及端口
  2. 利用ARP获得下一跳MAC地址
  3. 用下一跳MAC地址构造链路层帧，发送

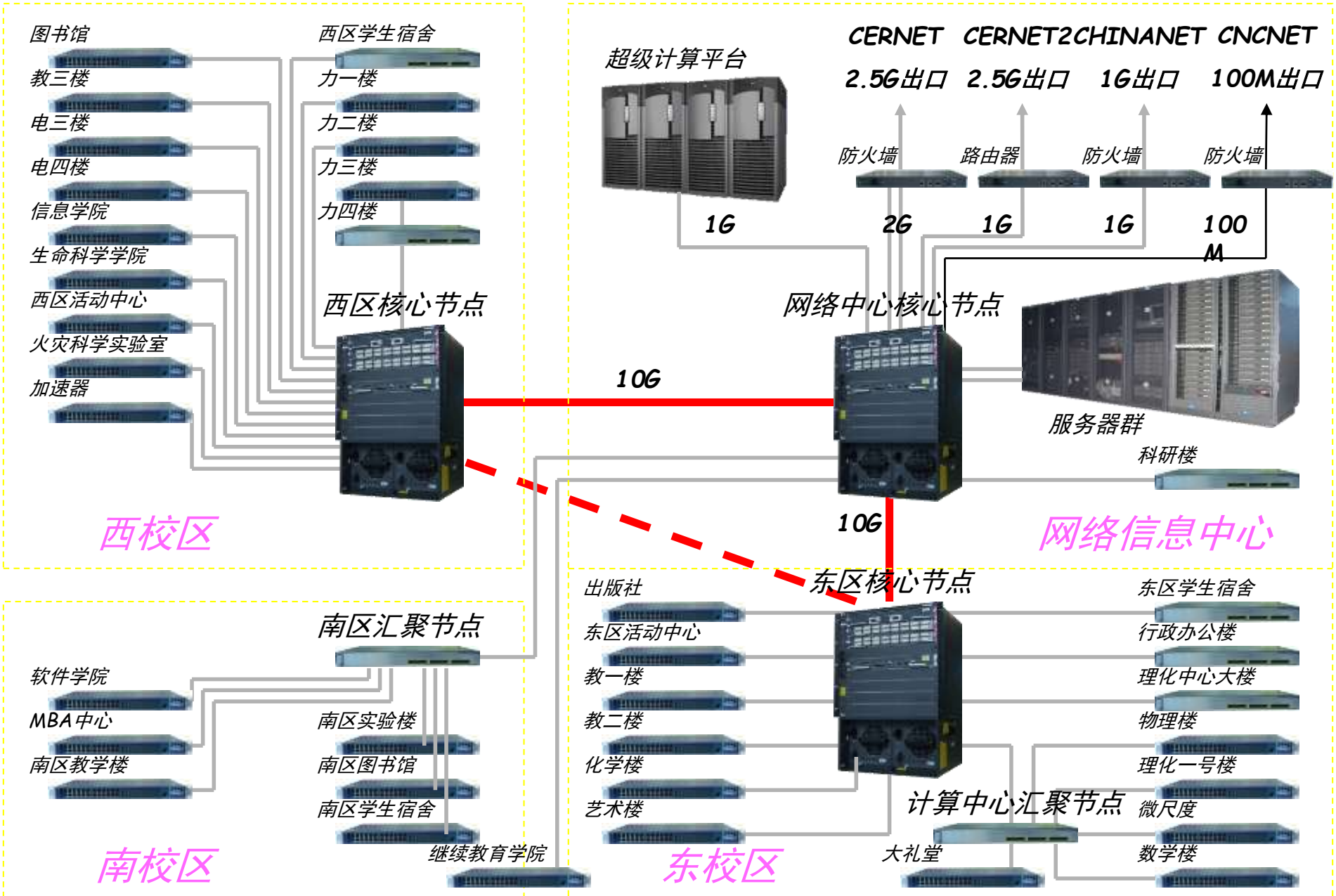
- 三层交换机转发IP包的过程：
  1. 将以上第1、第2步的结果缓存起来（以目的IP地址为索引，哈希表）
  2. 用目的IP地址查找缓存（一次精确匹配）：

     1）若命中，直接用下一跳MAC地址构造链路层帧，发送

     2）若未命中，执行以上第1、2、3步

- 三层交换机转发速度快的原因：
  - **一次选路，多次转发**

中国科学技术大学校园网络示意图（非最新）



Link layer, LANs: outline

5.1 introduction, services

5.2 error detection, correction

5.3 multiple access protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

5.5 link virtualization

5.6 data center networking

5.7 a day in the life of a web request

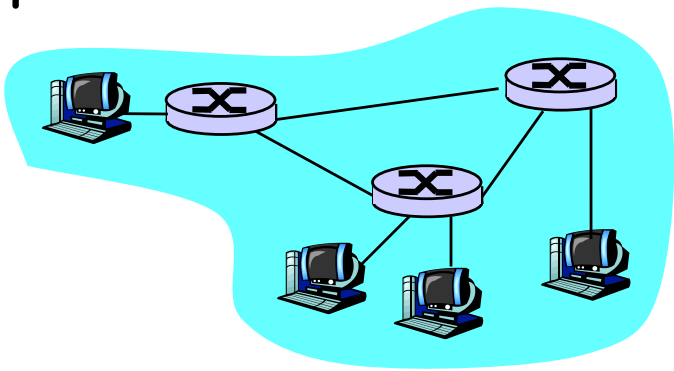
The Internet: virtualizing networks

1974: 多个不连通的网络

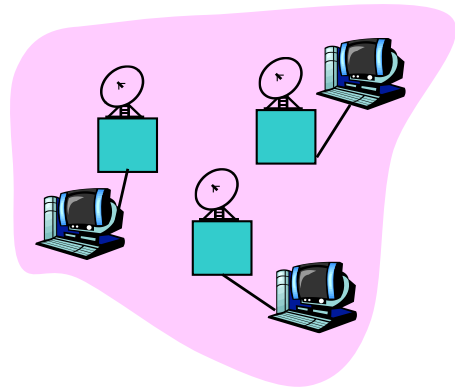
- ARPAnet
- data-over-cable networks
- packet satellite network (Aloha)
- packet radio network

它们在以下方面不同:

- 编址方法
- 包格式
- 差错恢复
- 选路



ARPAnet



satellite net

"A Protocol for Packet Network Intercommunication", V. Cerf, R. Kahn, IEEE Transactions on Communications, May, 1974, pp. 637-648.

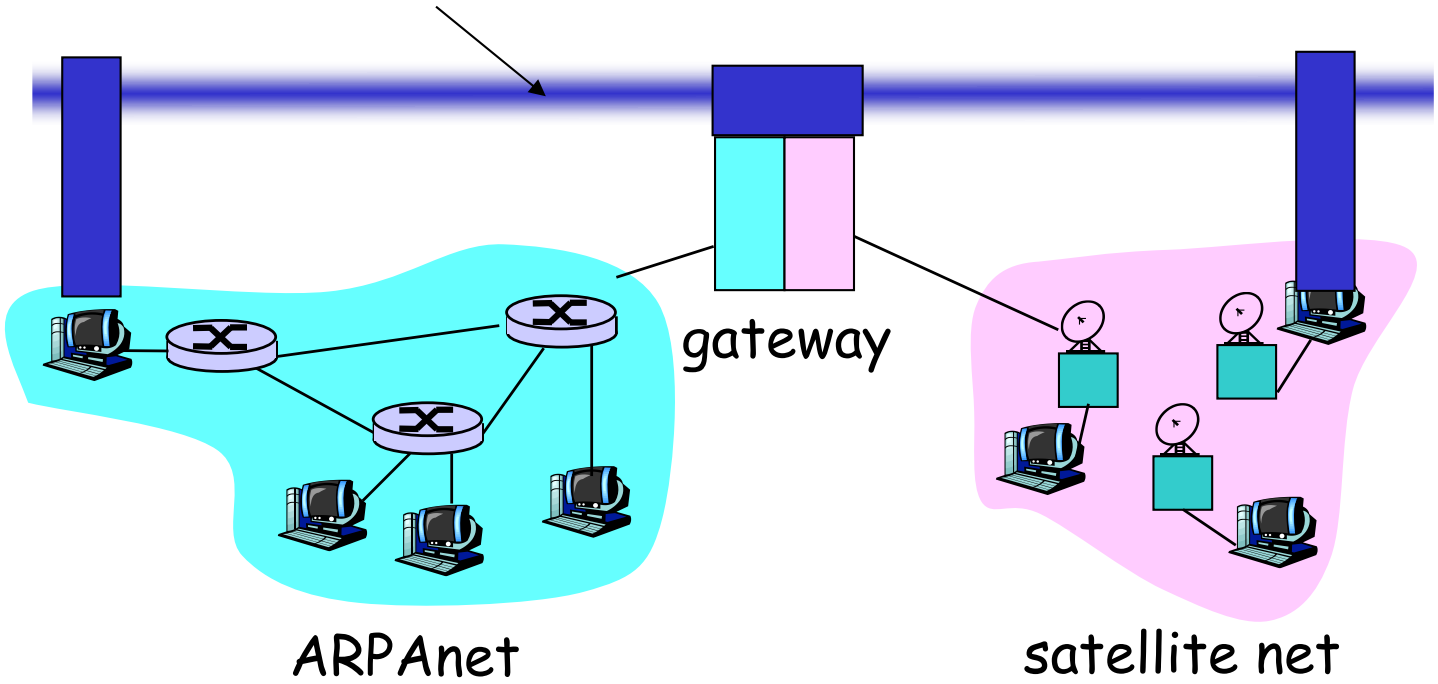
The Internet: virtualizing networks

在物理网络上增加一个逻辑层次（IP层）

- 在逻辑层上统一编址、统一包格式
- 互联在一起的网络看起来像一个网络
- network of networks

用网关连接不同的物理网络:

- 在逻辑层上选路到下一个网关
- 将IP包封装在本地网络帧中，发送到下一个网关



Cerf & Kahn's Internetwork Architecture

- 两级编址: IP网络，物理网络
- IP层提供统一的网络视图：地址，包格式
- 底层可以是任意的物理网络：
  - cable
  - satellite
  - 56K telephone modem
  - today: ATM, MPLS
  - ...
- 物理网络对于IP层是不可见的，对于IP来说物理网络只是一条虚拟链路而已！

Link layer, LANs: outline

6.1 introduction, services

6.2 error detection, correction

6.3 multiple access protocols

64 LANs

- addressing, ARP
- Ethernet
- switches
- VLANS

6.5 link virtualization

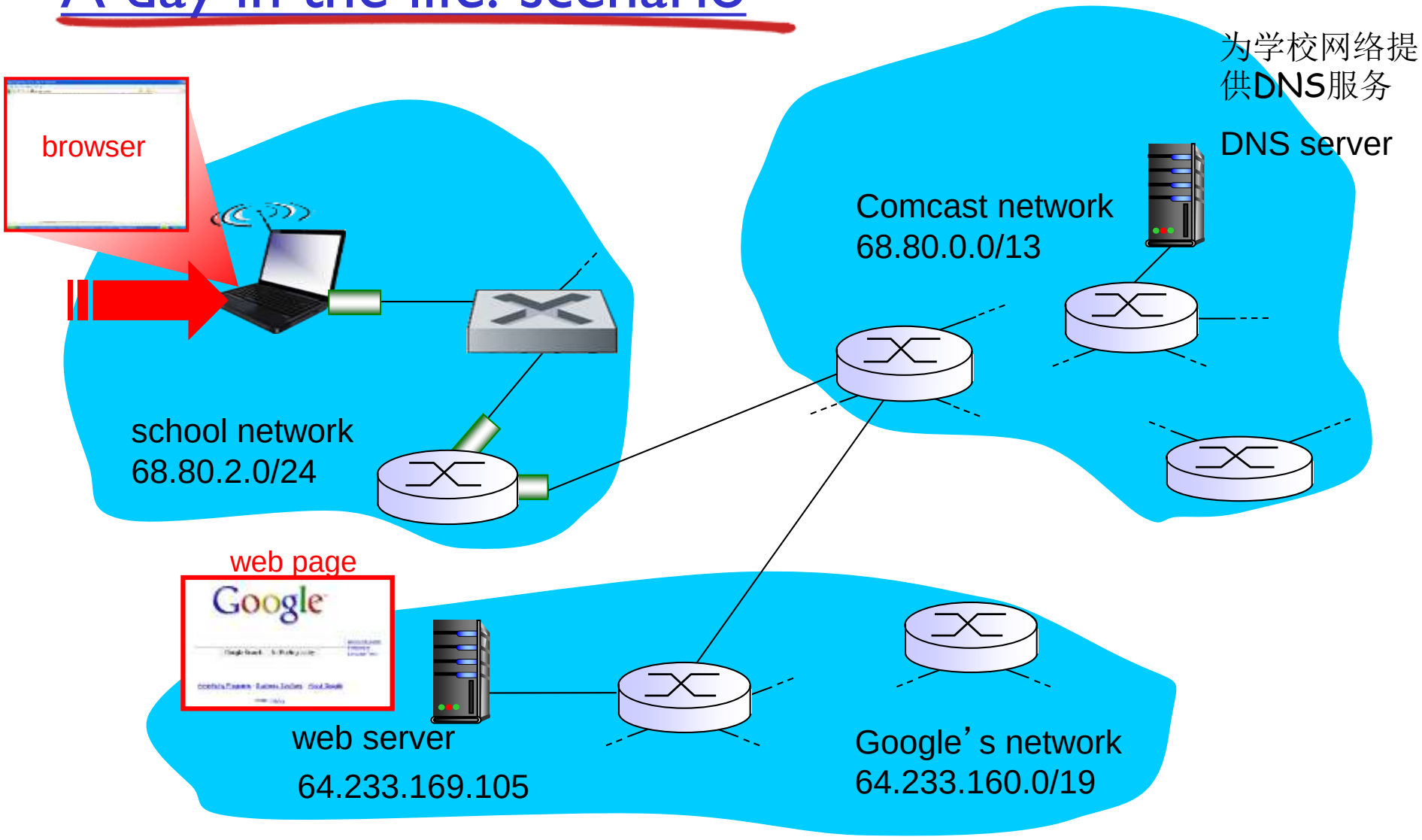
6.6 data center networking

6.7 a day in the life of a web request

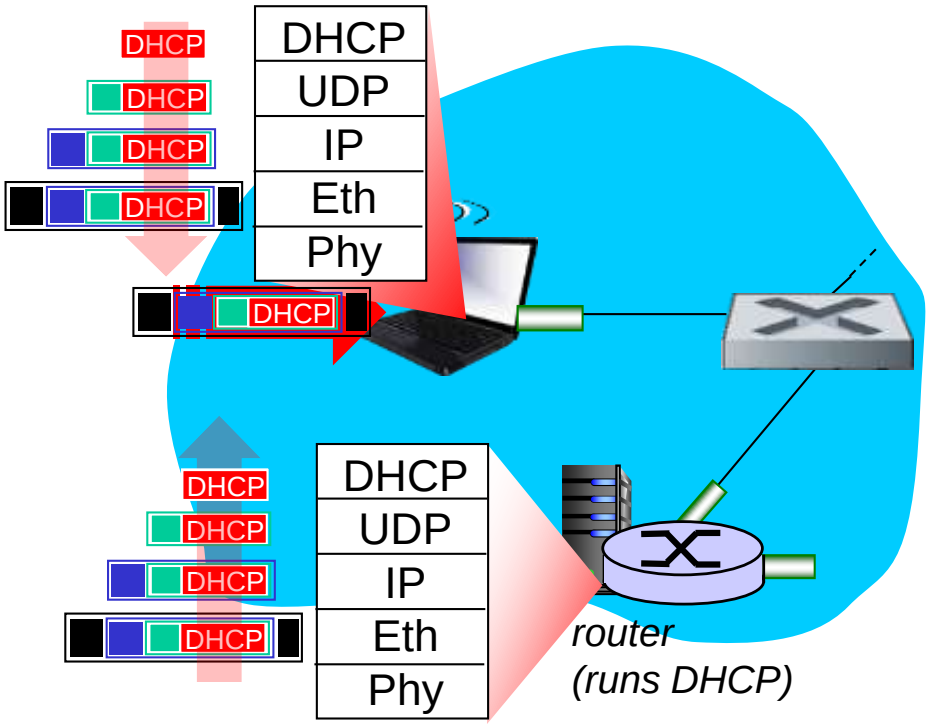
Synthesis: a day in the life of a web request

- 我们已经从上到下走过了整个协议栈：
  - 应用层，传输层，网络层，数据链路层
- 现在将所有内容综合起来：
  - 目的：使用一个简单的场景来复习和理解相关的协议
  - 简单场景：将一台笔记本电脑连入校园网，请求网页www.google.com

A day in the life: scenario

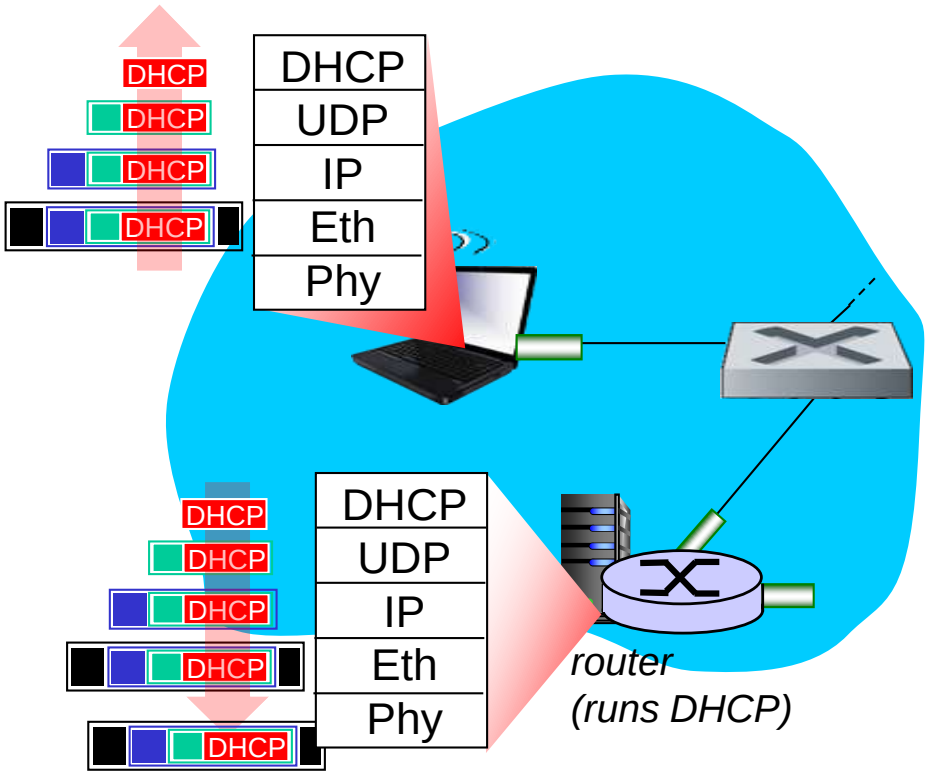


A day in the life… connecting to the Internet



- 笔记本电脑需要获取IP地址、第一跳路由器的IP地址、DNS服务器的IP地址：使用DHCP
  - DHCP请求被封装在UDP/IP/Ethernet中
  - 以太帧在局域网中广播(dest: FFFFFFFFFFFF)，被运行了DHCP服务器的路由器收到
  - 经过层层解封装，DHCP请求到达DHCP服务器

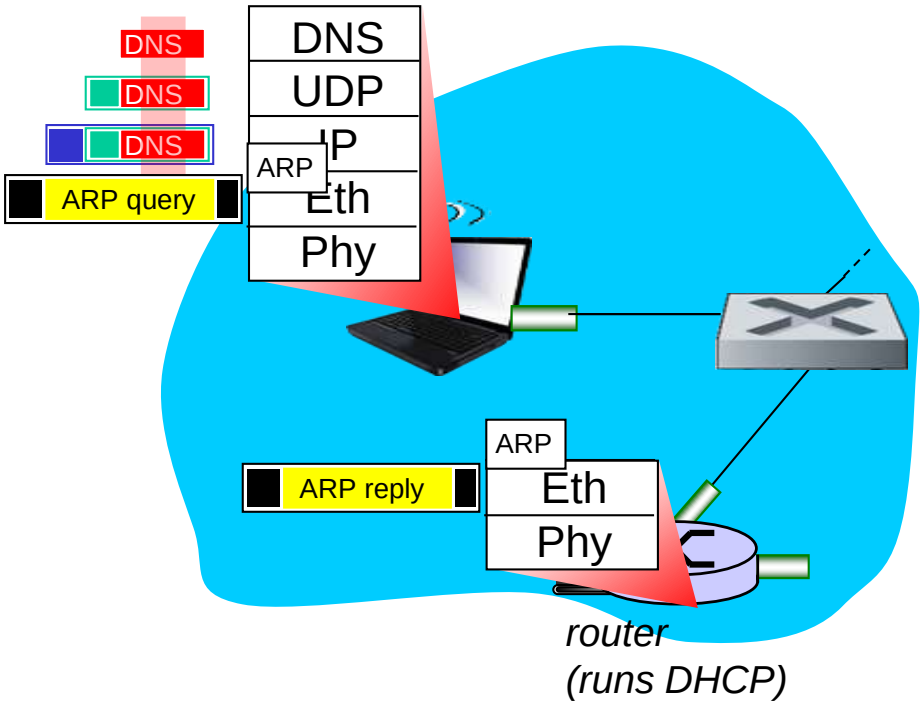
A day in the life… connecting to the Internet



- DHCP服务器构造DHCP ACK，包含客户的IP地址、第一跳路由器的IP地址、DNS服务器名字和IP地址
- DHCP ACK被封装，通过LAN转发 (**交换机自学习**)，到达笔记本
- 经过层层解封装，客户收到DHCP ACK

客户现在知道了IP地址、第一跳路由器的IP地址、DNS服务器的IP地址

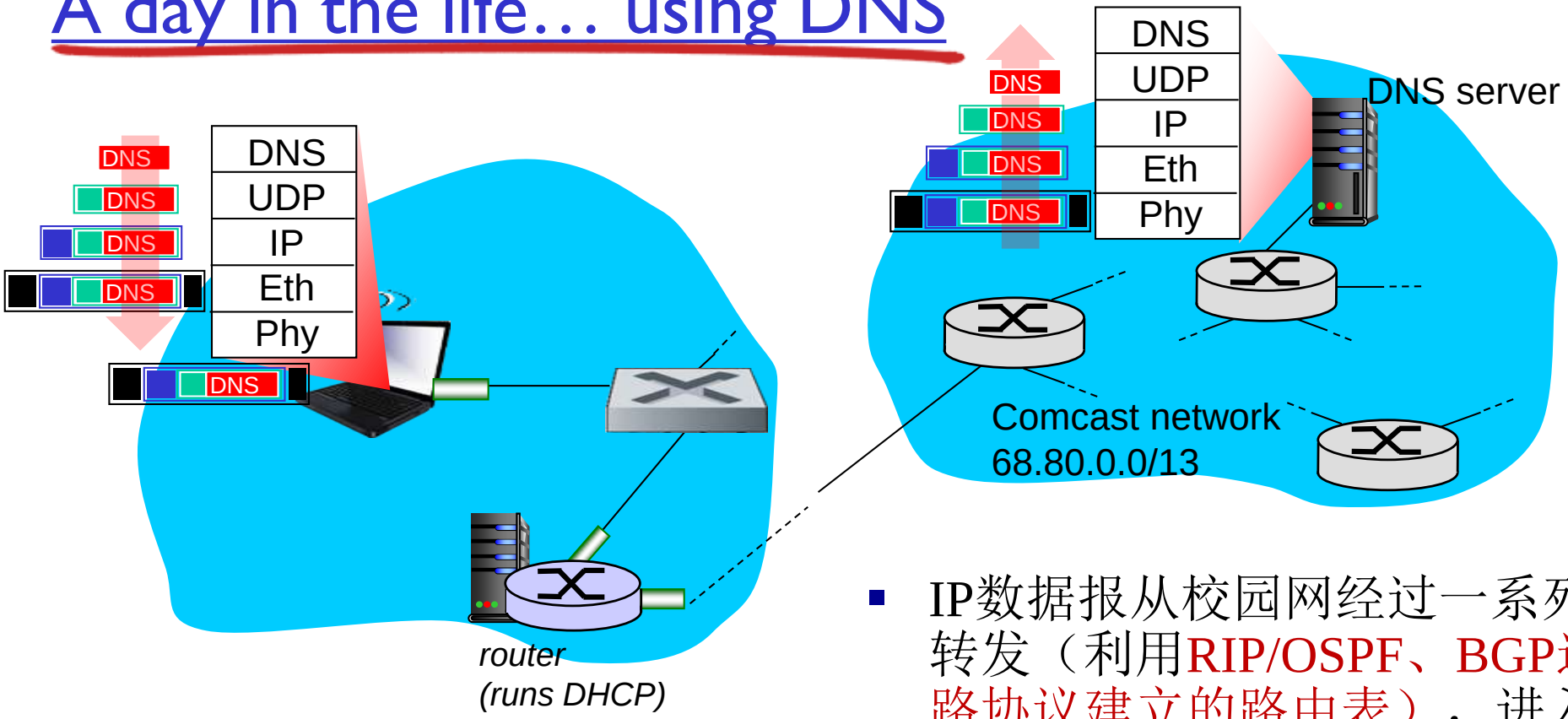
A day in the life… ARP (before DNS, before HTTP)



- 浏览器发送HTTP请求前，需要知道www.google.com对应的IP地址：使用DNS
- 解析器构造DNS查询报文，封装到UDP/IP/Eth中；为发送帧，需要第一跳路由器接口的MAC地址：使用ARP
- ARP查询报文被广播，到达路由器；路由器构造ARP响应报文，通过LAN转发，到达笔记本

客户现在知道了第一跳路由器对应接口的MAC地址

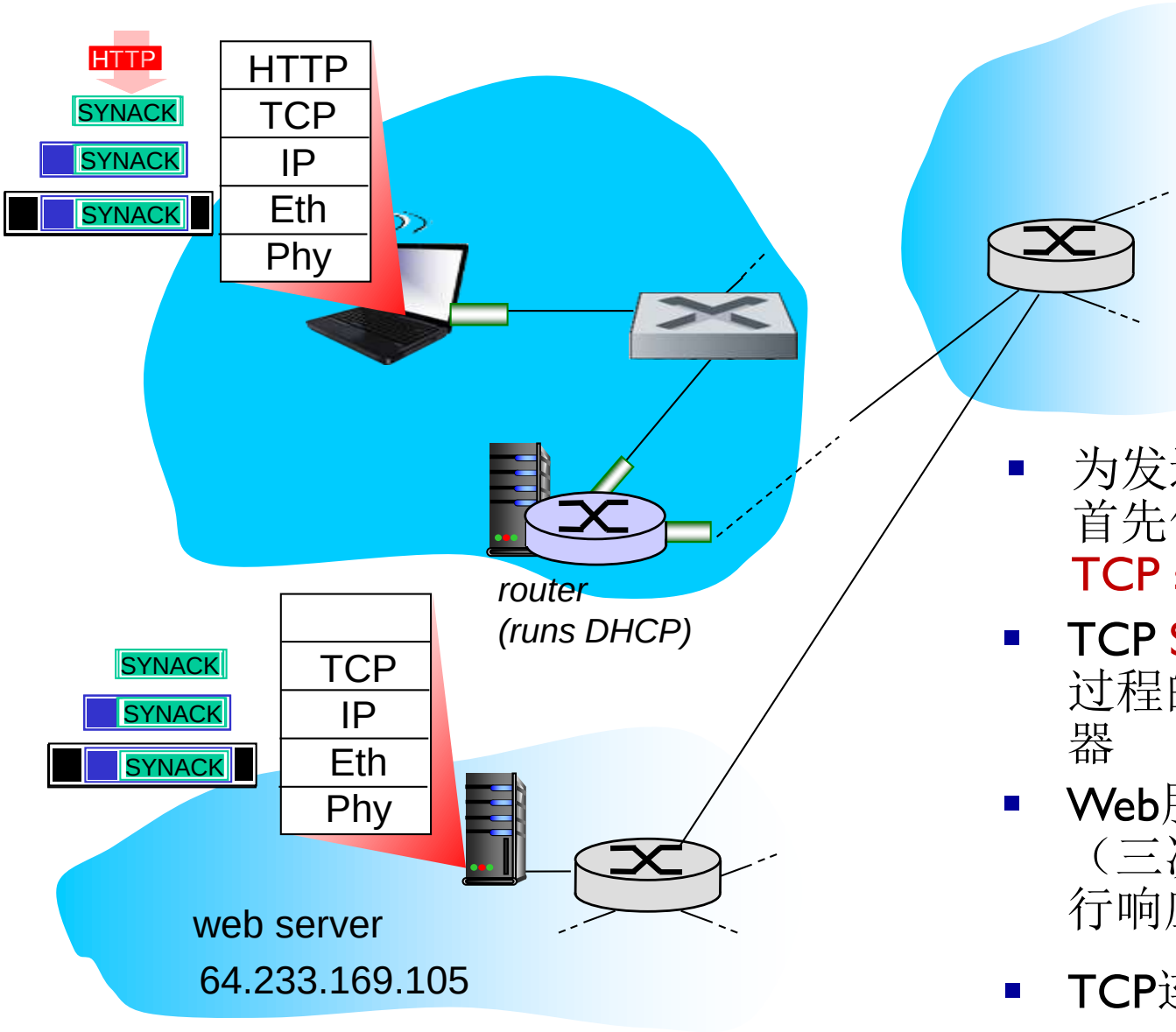
A day in the life… using DNS



- 携带了DNS查询报文的IP包经交换机转发，到达第一跳路由器

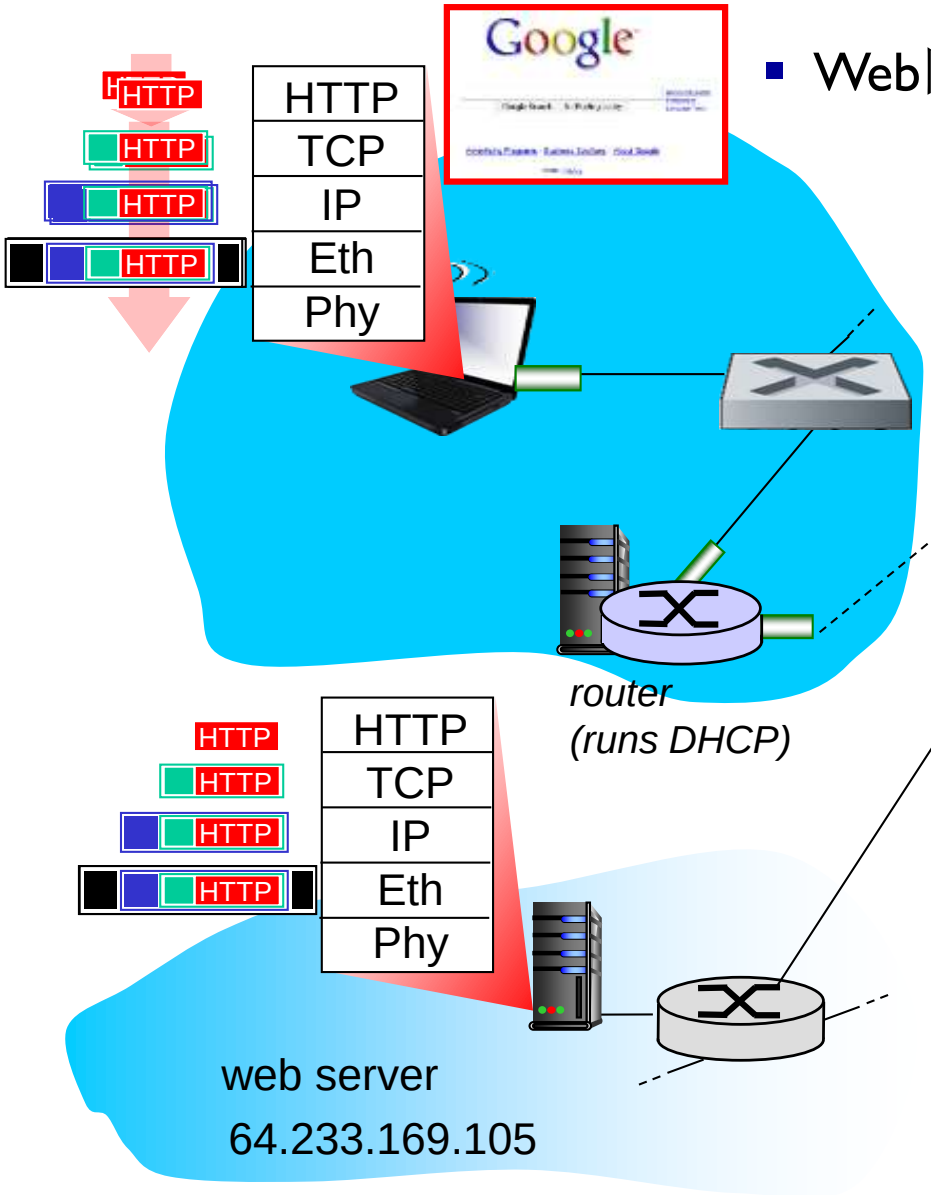
- IP数据报从校园网经过一系列转发（利用RIP/OSPF、BGP选路协议建立的路由表），进入Comcast网络, 到达 DNS服务器
- DNS查询报文经层层解封装到达DNS服务器程序
- DNS服务器用（缓存的）www.google.com对应的IP地址进行响应

A day in the life…TCP connection carrying HTTP



- 为发送HTTP请求，浏览器首先创建到web服务器的TCP socket
- TCP SYN报文段（三次握手过程的第一步）到达web服务器
- Web服务器用TCP SYNACK（三次握手过程的第2步）进行响应
- TCP连接建立

A day in the life… HTTP request/reply



- Web网页终于(!!!)在浏览器中显示

- HTTP request送入TCP套接字
- 携带HTTP请求的IP数据报被路由到www.google.com
- Web服务器用包含网页的HTTP reply 进行响应
- 包含HTTP reply的IP数据报被路由到浏览器

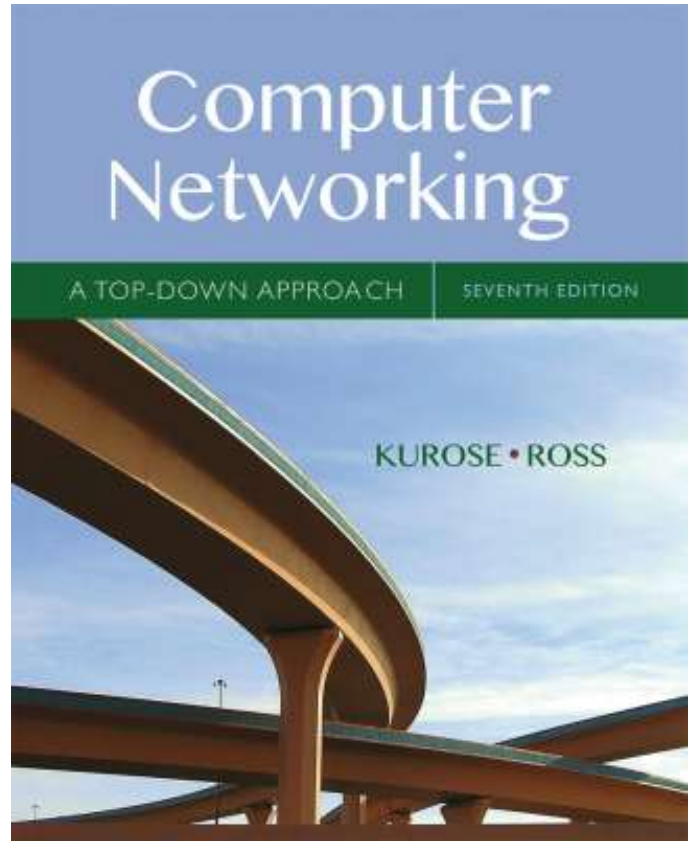
Chapter 6: Summary

- 数据链路层服务原理：
  - 差错检测与纠正
  - 共享广播信道：多址技术
  - 链路层编址，ARP
- 链路层技术实例：
  - 以太网
- 综合: a day in the life of a web request

作业

- 习题（11月28日）
  - 5，8，11，23，24，25，26
- 实验（12月4日）
  - Ethernet-ARP

Chapter 7
Wireless and Mobile Networks



A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

Chapter 7: 无线和移动网络

Background:

- 手机用户的数量已经超过了固话用户的数量（5：1)
- 无线上网的设备已经与采用有线方式上网的设备数量相等
- 与有线固定网络相比，无线移动上网增加了以下两方面的问题：
  - 无线（wireless）：使用无线链路通信，给物理层和数据链路层带来很多问题
  - 移动（mobility）：终端改变网络接入点，给网络层带来很大问题

Chapter 7 outline

7.1 Introduction

Wireless

7.2 Wireless links, characteristics

- CDMA

7.3 IEEE 802.11 wireless LANs ("Wi-Fi")

7.4 Cellular Internet Access

- architecture
- standards (e.g., 3G, LTE)

Mobility

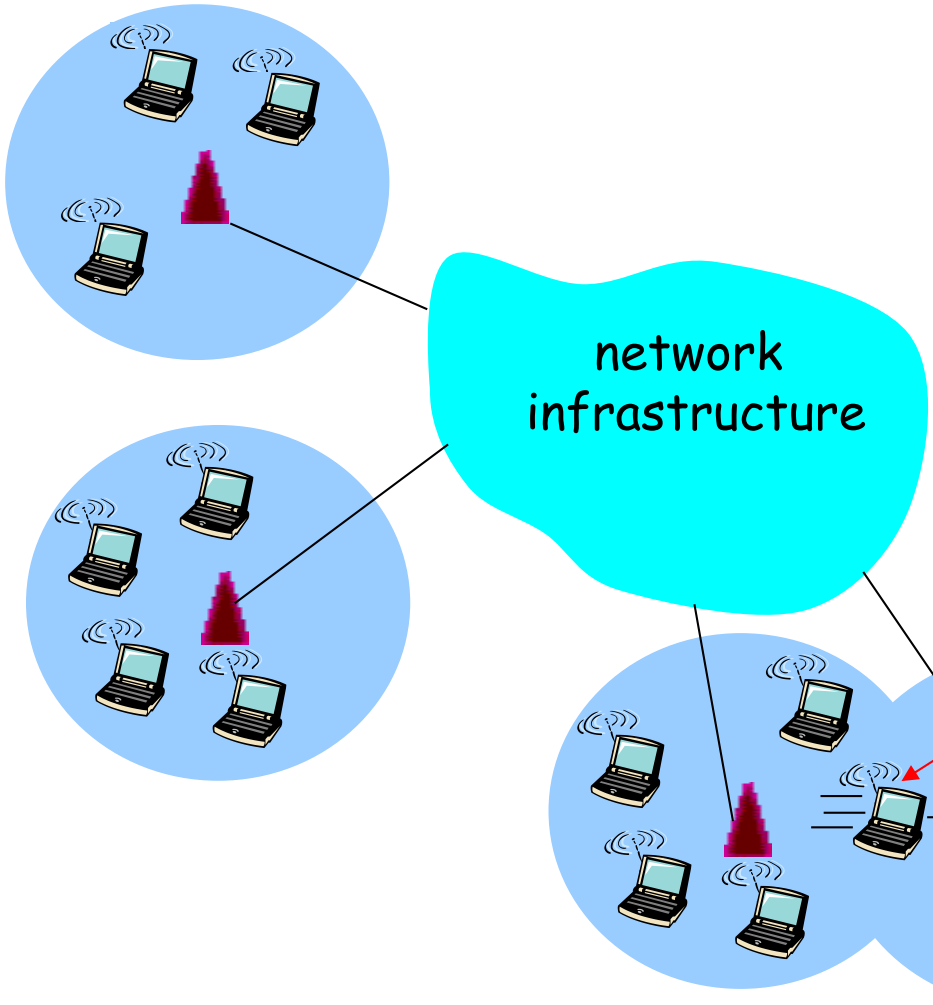
7.5 Principles: addressing and routing to mobile users

7.6 Mobile IP

7.7 Handling mobility in cellular networks

7.8 Mobility and higher-layer protocols

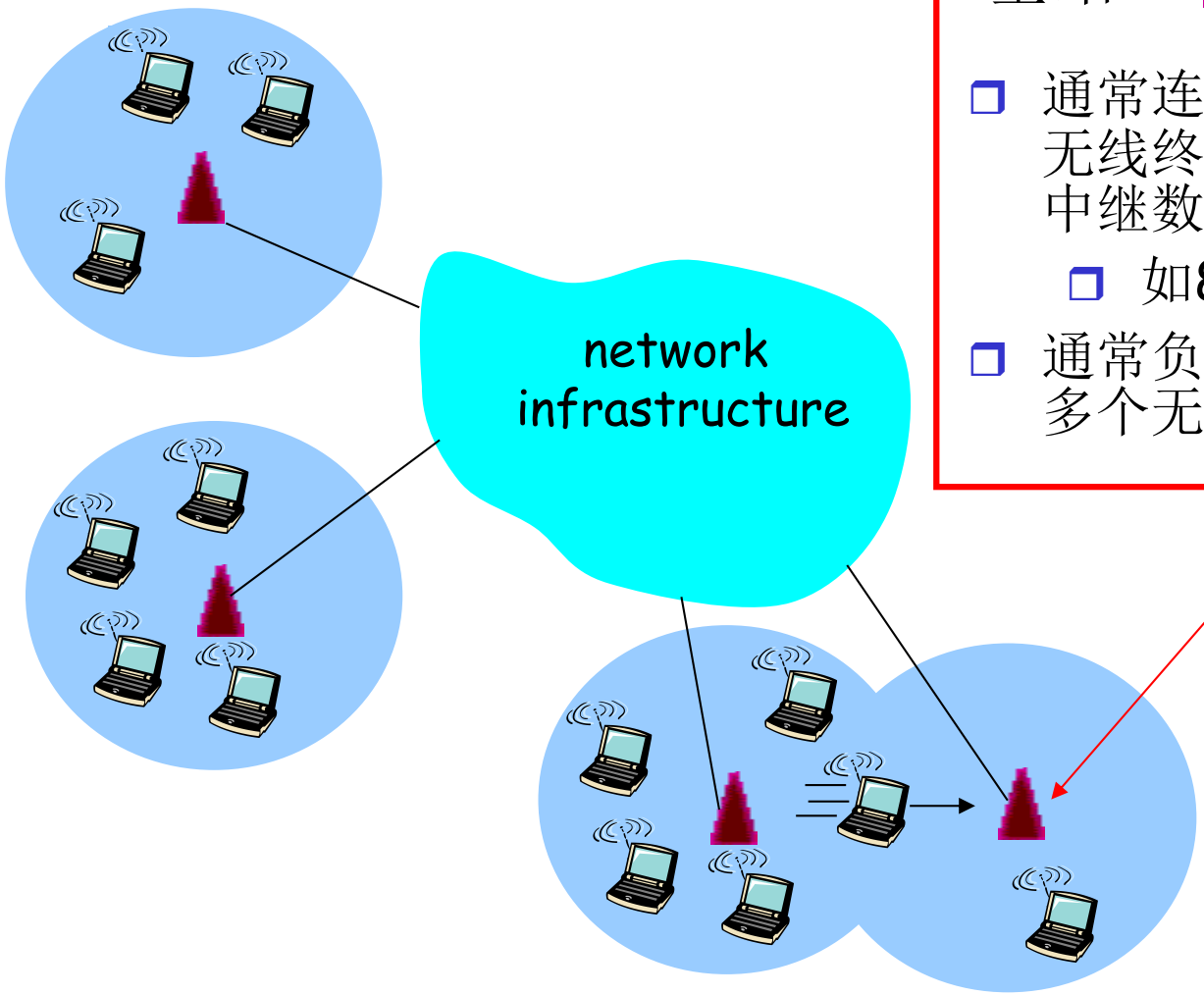
无线网络的组成



无线终端

- laptop, PDA, 智能手机
- 运行网络应用
- 可能静止或移动
  - 无线并不一定意味着移动

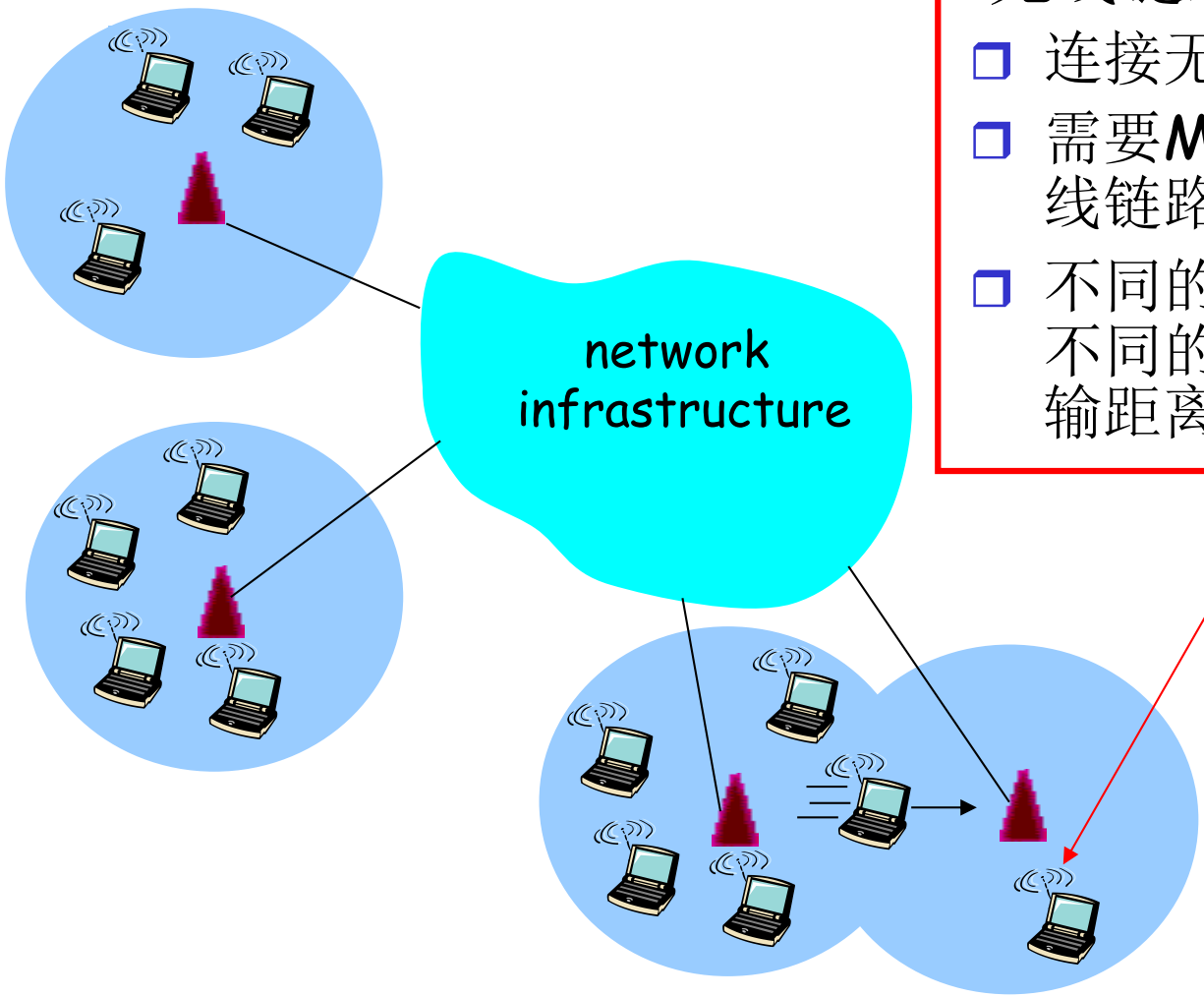
无线网络的组成



基站

- 通常连接到固定网络，在无线终端和固定网络之间中继数据包
  - 如802.11AP，蜂窝塔
- 通常负责协调与之关联的多个无线主机的传输

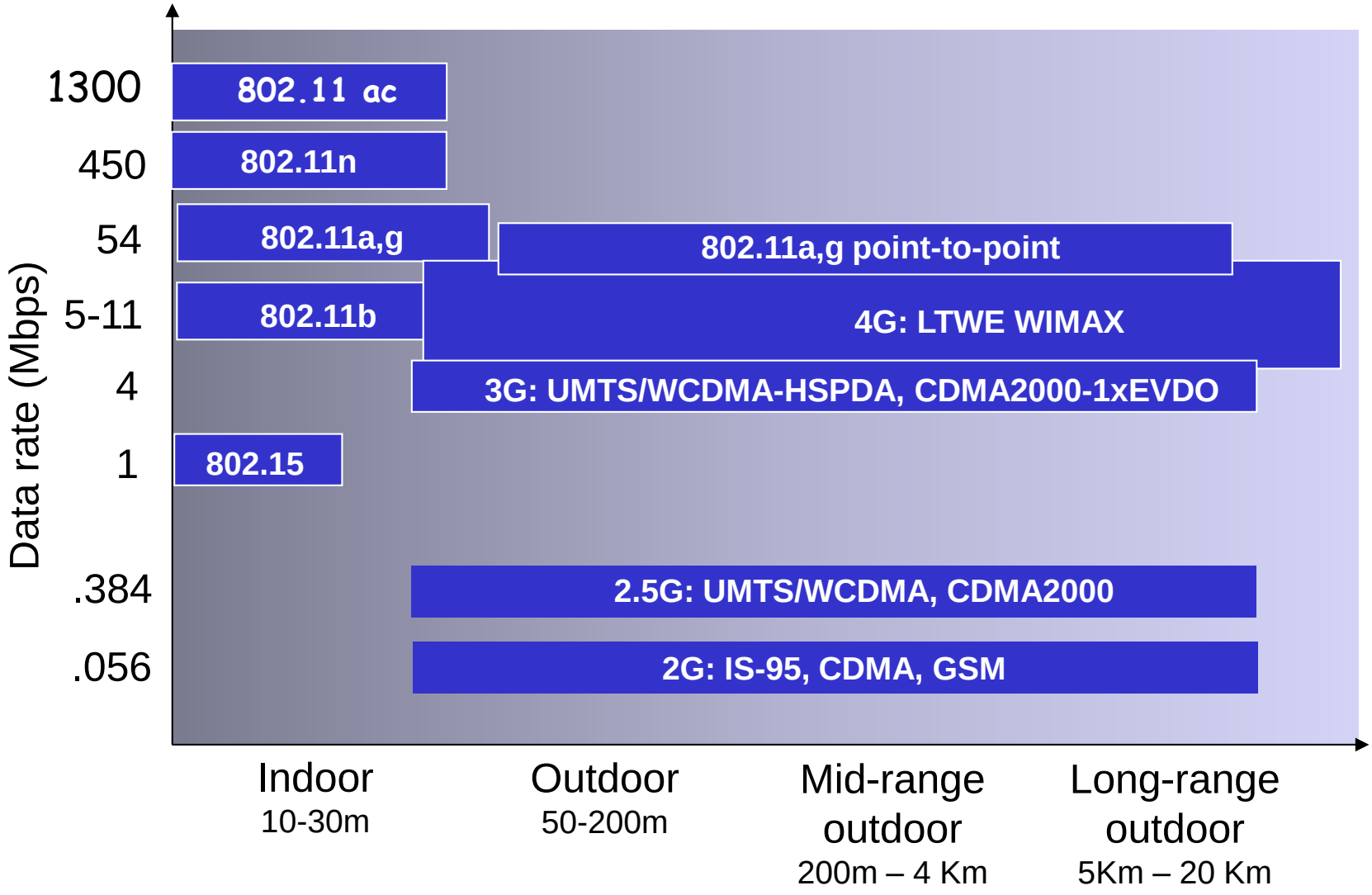
无线网络的组成



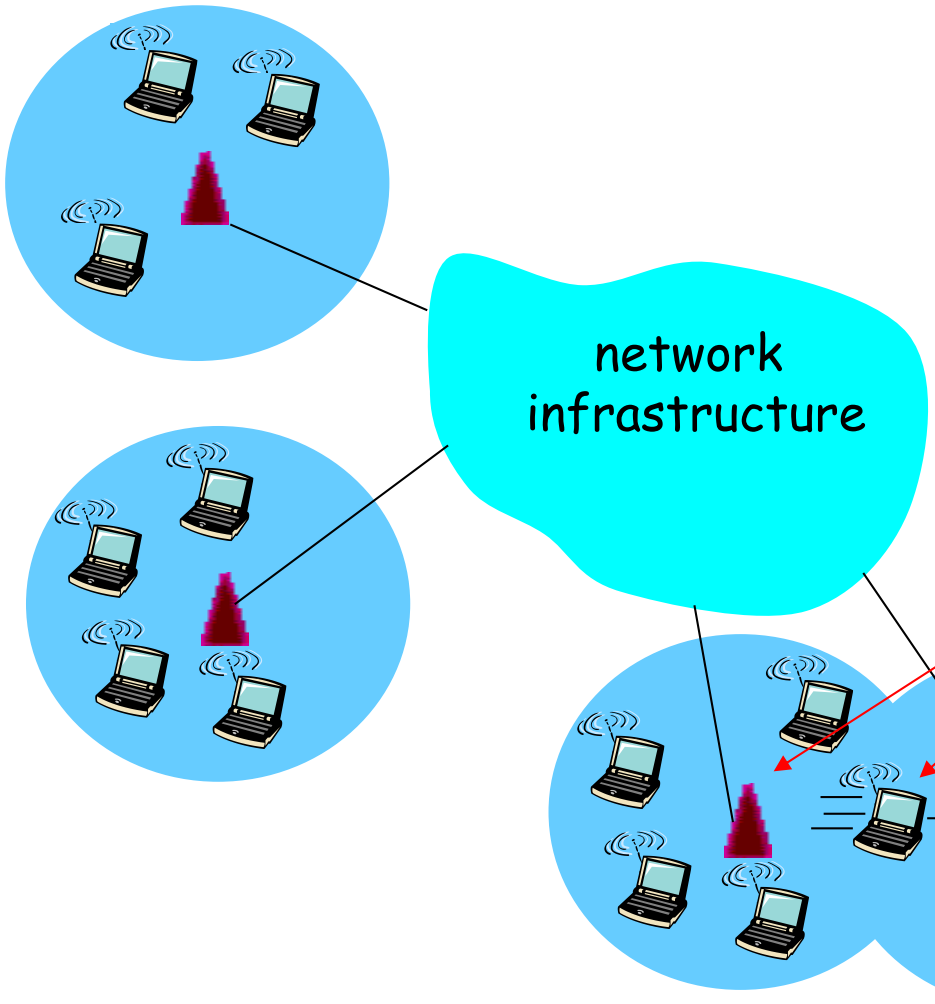
无线链路

- 连接无线终端和基站
- 需要MAC协议协调无线链路的使用
- 不同的无线链路具有不同的数据速率和传输距离

一些无线链路的特性


Data rate (Mbps)
1300
450
54
5-11
4
1
.384
.056
802.11 ac
802.11n
802.11a,g
802.11a,g point-to-point
802.11b
4G: LTWE WIMAX
3G: UMTS/WCDMA-HSPDA, CDMA2000-1xEVDO
802.15
2.5G: UMTS/WCDMA, CDMA2000
2G: IS-95, CDMA, GSM
Indoor
10-30m
Outdoor
50-200m
Mid-range outdoor
200m – 4 Km
Long-range outdoor
5Km – 20 Km

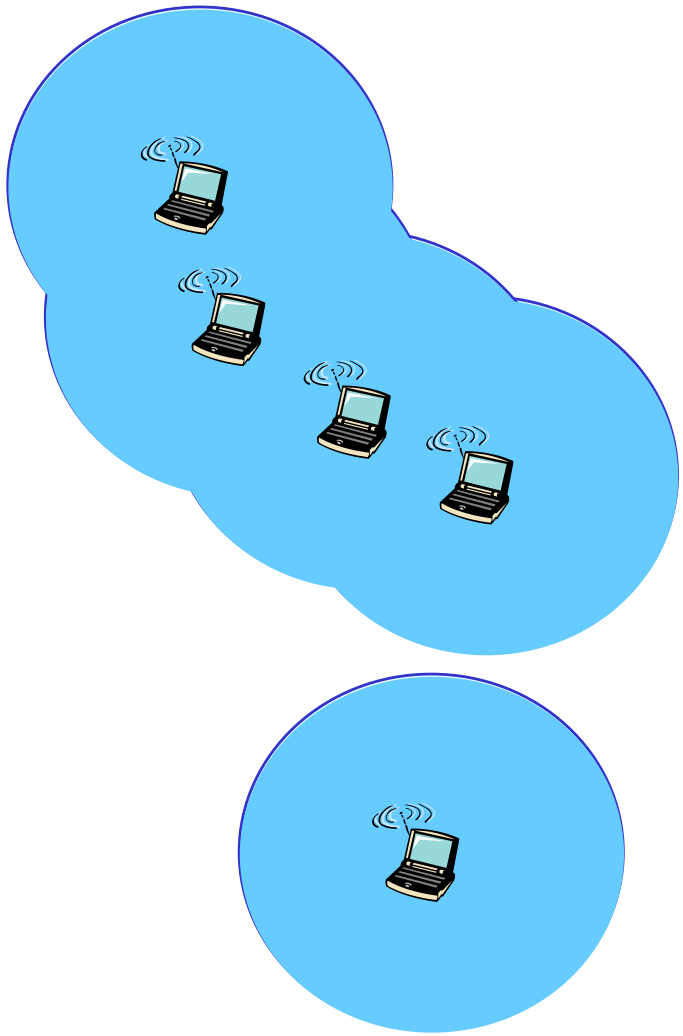
无线网络的运行模式



基础设施模式

- 无线终端通过基站连接到固定网络（网络基础设施），所有传统的网络服务由固定网络提供
- 切换：无线终端接入到不同基站的过程

无线网络的运行模式



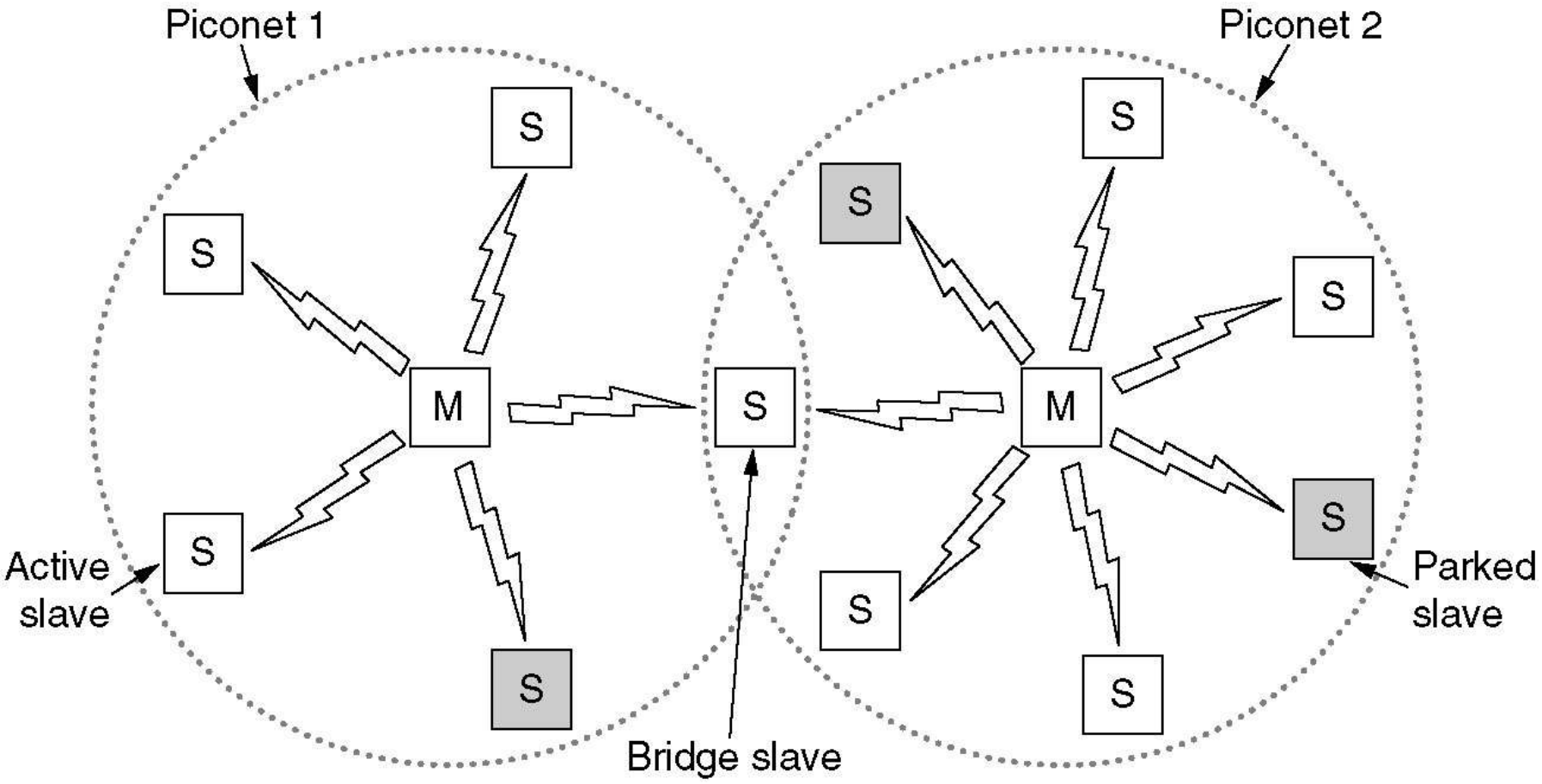
自组织模式

- 网络中没有基站
- 节点只能与其通信范围内的节点通信
- 节点相互帮助转发分组，每个节点既是终端又是路由器

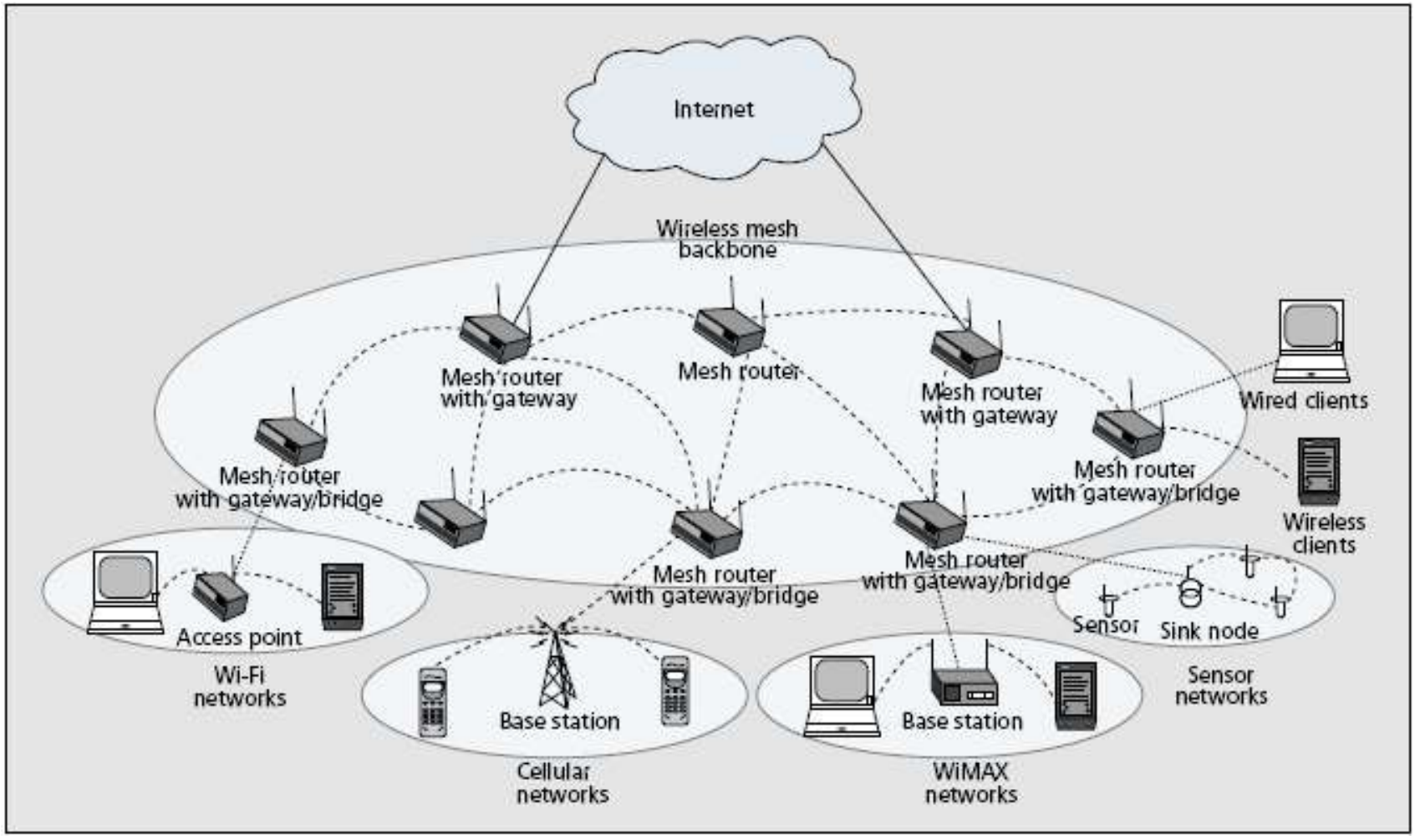
无线网络的分类

	单跳	多跳
有基础设施	主机连接到基站，基站连接到固定网络(如WiFi，cellular)	主机通过多个无线节点的转发才能到达固定网络（如无线网状网络）
无基础设施	无基站，不连接到固定网络，节点间通信不需要中继（如蓝牙网络）	无基站，不连接到固定网络，节点间通信需要通过其它节点转发（如自组网，车载网）

蓝牙网络


Piconet 1
Piconet 2
S
S
S
M
S
S
S
S
M
S
S
S
S
S
Active slave
Parked slave
Bridge slave

无线网状网


Internet
Wireless mesh backbone
Mesh router with gateway
Mesh router
Mesh router with gateway
Wired clients
Mesh router with gateway/bridge
Mesh router with gateway/bridge
Wireless clients
Mesh router with gateway/bridge
Mesh router with gateway/bridge
Access point
Wi-Fi networks
Sensor
Sink node
Sensor networks
Base station
Cellular networks
Base station
WiMAX networks

Chapter 7 outline

7.1 Introduction

Wireless

7.2 Wireless links, characteristics

- CDMA

7.3 IEEE 802.11 wireless LANs ("Wi-Fi")

7.4 Cellular Internet Access

- architecture
- standards (e.g., 3G, LTE)

Mobility

7.5 Principles: addressing and routing to mobile users

7.6 Mobile IP

7.7 Handling mobility in cellular networks

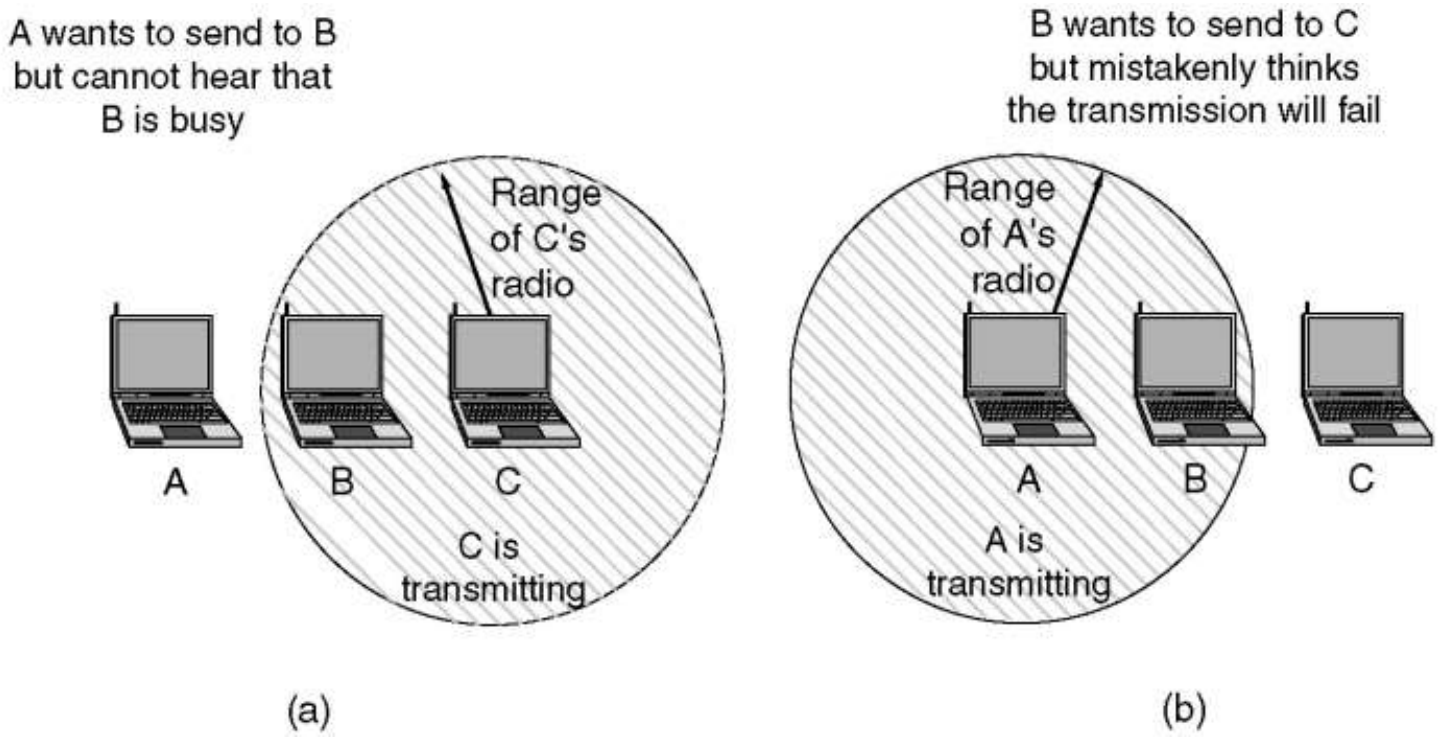
7.8 Mobility and higher-layer protocols

无线链路的特性

- 信号衰减: 信号在传播过程中能量逐渐减少（路径损耗）
- 干扰: 受到其它信号源的干扰
- 多径传播: 由于地面或物体的反射作用，信号沿多条不同长度的路径到达接收端

- 以上特性导致无线链路的传输距离受限、误码率很高

无线网络的特性



(a) 隐藏节点问题:

- C正在向B发送
- A监听到信道空闲，A向B发送
- A和C的信号在B冲突

(b) 暴露节点问题:

- B准备向C发送
- B监听到信道忙（A在发送）
- B不发送，但其实B可以发送（A和B的信号不会在C冲突）

CSMA在多跳无线网络中作用受限

- 通过载波侦听，发送节点只能知道其周围是否有节点在发送；但真正影响此次通信的是接收节点周围是否有节点在发送
- 隐藏节点：不在发送节点的通信范围内、但在接收节点通信范围内的活跃节点（发送节点听不到，但影响接收）
- 暴露节点：在发送节点的通信范围内、但不在接收节点通信范围内的活跃节点（发送节点能听到，但不影响接收）

Chapter 7 outline

7.1 Introduction

Wireless

7.2 Wireless links, characteristics

- CDMA

7.3 IEEE 802.11 wireless LANs ("Wi-Fi")

7.4 Cellular Internet Access

- architecture
- standards (e.g., 3G, LTE)

Mobility

7.5 Principles: addressing and routing to mobile users

7.6 Mobile IP

7.7 Handling mobility in cellular networks

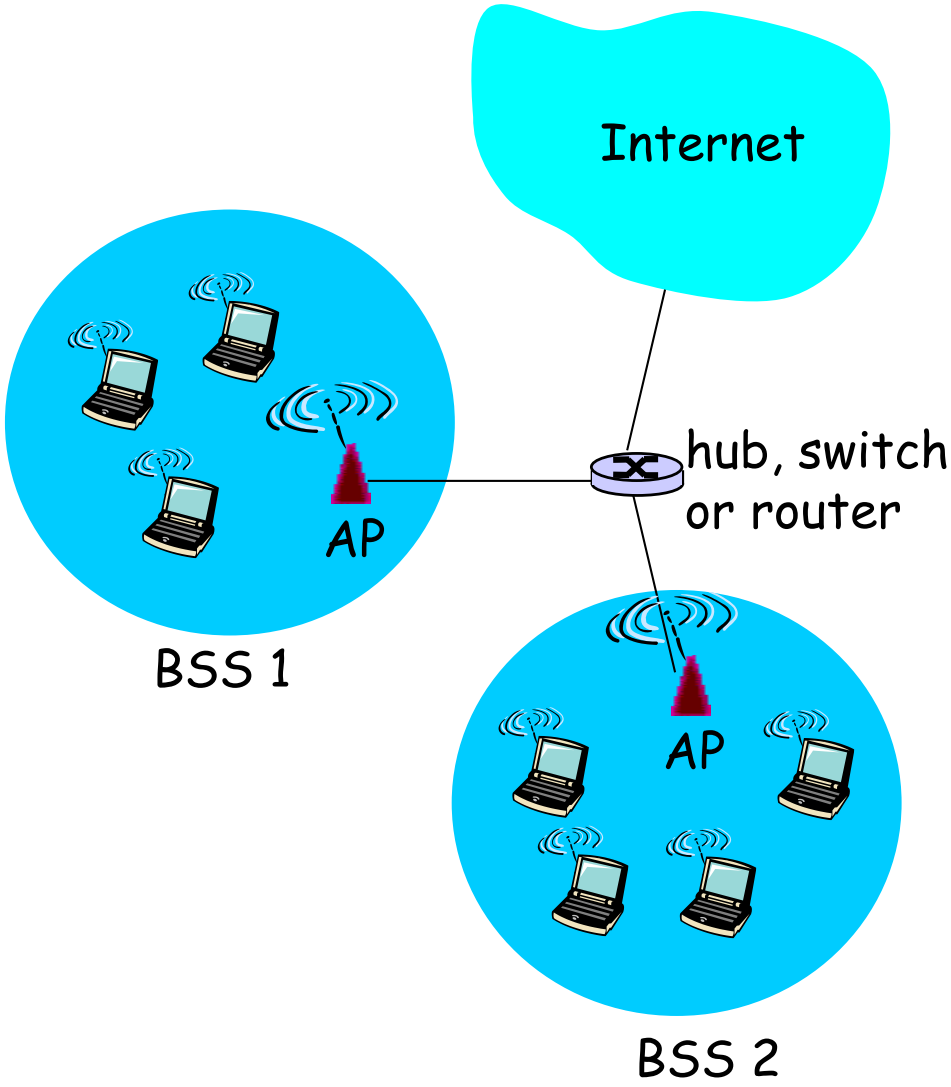
7.8 Mobility and higher-layer protocols

IEEE 802.11无线局域网

- 802.11b
  - 2.4-5 GHz range
  - up to 11 Mbps
- 802.11a
  - 5-6 GHz range
  - up to 54 Mbps
- 802.11g
  - 2.4-5 GHz range
  - up to 54 Mbps
- 802.11n: 多天线
  - 2.4-5 GHz range
  - up to 200 Mbps

- 均使用CSMA/CA 作为MAC协议
- 都支持基站模式和自组织模式
- 物理层不同

（1）802.11无线局域网架构

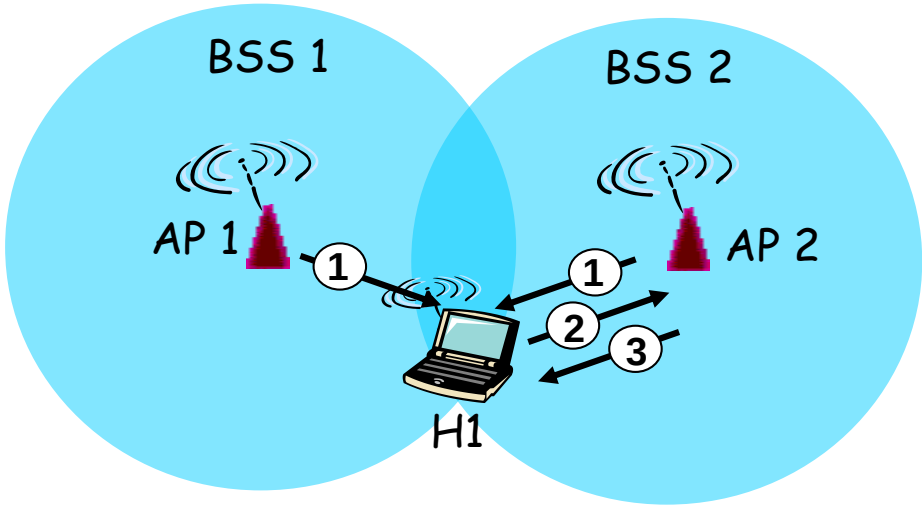


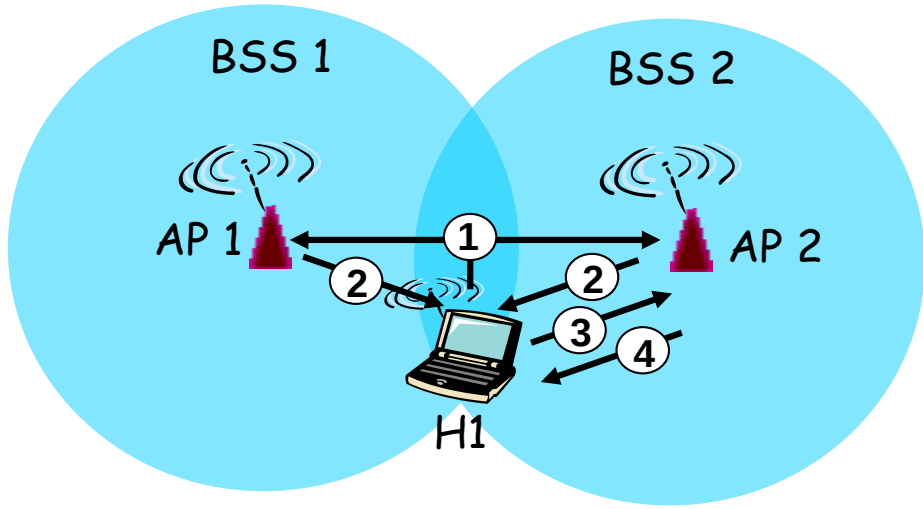
- 802.11无线LAN的基本组成单元是基本服务集（BSS）
- 一个BSS 包括:
  - 若干无线终端
  - 一个无线接入点AP
- 每个无线接口（终端及AP）均有一个全局唯一的MAC地址
- 注意：AP与路由器相连的有线端口没有MAC地址（AP对于路由器是透明的）

802.11: 信道与关联

- 802.11将通信频段划分成若干信道，每个BSS分配一个信道：
  - 管理员安装AP时，为AP分配一个服务集标识符（SSID），并选择AP使用的信道
  - 相邻AP使用的信道可能相互干扰
- 主机必须与一个AP关联：
  - 扫描信道，监听各个AP发送的信标帧（包含AP的SSID和MAC地址）
  - 选择一个AP进行关联（可能需要身份鉴别）
  - 使用DHCP获得AP所在子网中的一个IP地址

802.11: 主动/被动扫描





被动扫描:

(1)主机监听AP发送的信标帧

(2)主机选择一个AP发送关联请求帧

(3)AP向主机发送关联响应帧

主动扫描:

(1)主机广播探测请求帧

(2)AP发送探测响应帧

(3)主机从收到的探测响应中选择一个AP发送关联请求

(4)AP发送关联响应帧

（2）IEEE 802.11：MAC协议

- 采用CSMA：
  - 发送前监听信道，不与当前正在进行的发送冲突
- 发送过程中不检测冲突：
  - 发送过程中检测冲突很困难（接收信号的强度远小于发送信号的强度）
  - 不能检测出所有的冲突（隐藏节点）
- 使用链路层确认：
  - 接收方收到帧后需发送确认帧，以便让发送方知道是否发送成功
- 目标：避免冲突 CSMA/C(ollision)A(voidance)

802.11的操作模式

- PCF（Point Coordination Function）模式（可选）
  - 该模式只能用于有基础设施（基站）的无线网络，基站控制单元内的所有通信活动
  - 轮询：基站依次询问单元中的节点，被询问到的节点可以发送它们的帧，不会有冲突发生
- DCF（Distributed Coordination Function）模式：
  - 可用于有基础设施和无基础设施的无线网络，所有节点（AP和无线终端）使用CSMA/CA协议竞争信道
  - 支持两种机制：信道预约机制（可选），无信道预约的机制（所有实现必须支持）

使用信道预约机制的CSMA/CA

假设A欲向AP发送一个数据帧：

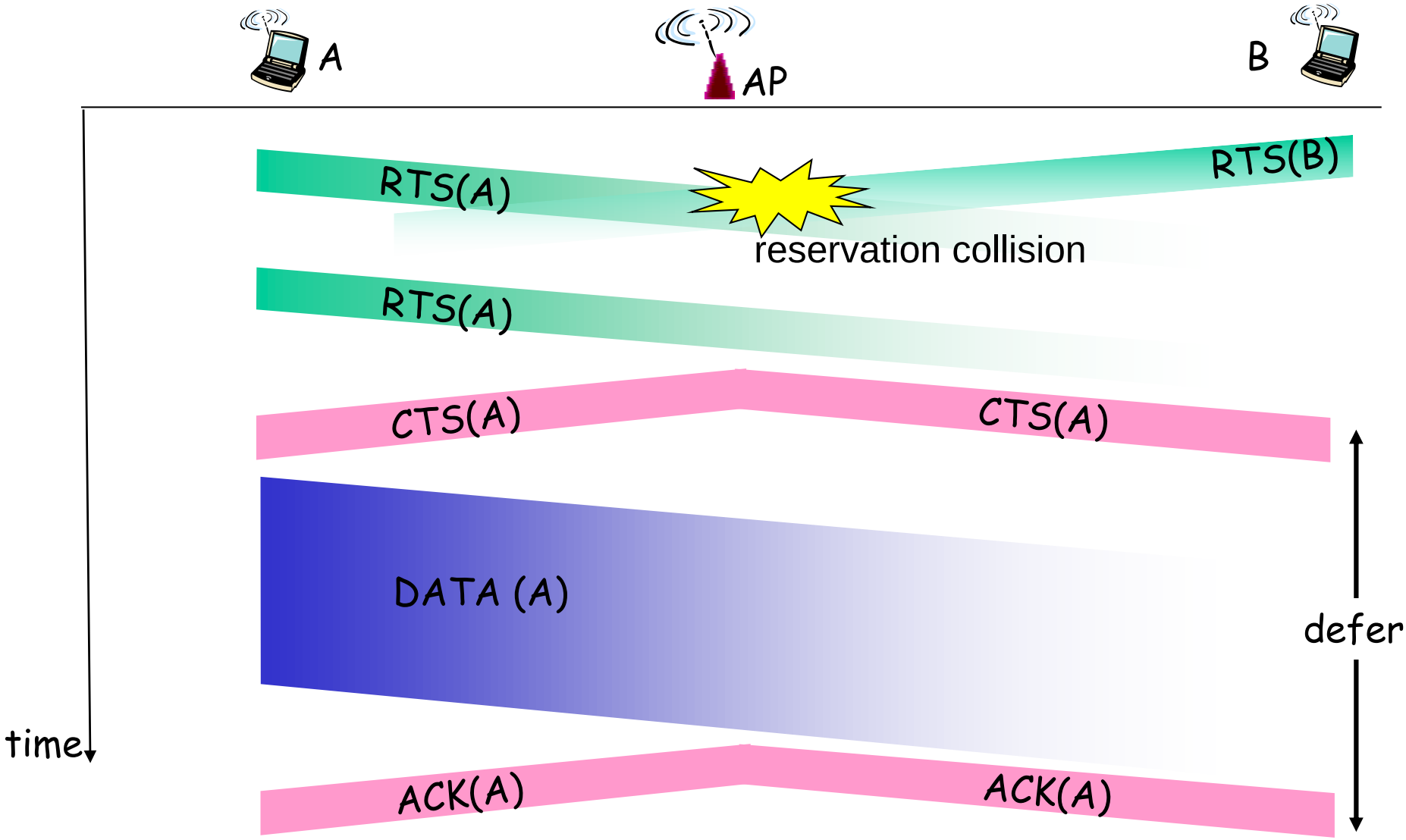
- A向AP发送一个RTS帧：帧中给出随后要发送的数据帧及ACK帧需要的总时间
- AP收到后回复一个CTS帧：帧中给出同样的时间
- A收到CTS帧后：发送数据帧
- AP收到帧后：发送一个ACK帧进行确认

- 收到RTS帧（A附近）及CTS帧（AP附近）的节点均沉默指定的时间，让出信道让A和AP完成发送

- 若A和B同时发送RTS帧，产生冲突，不成功的发送方随机等待一段时间后重试

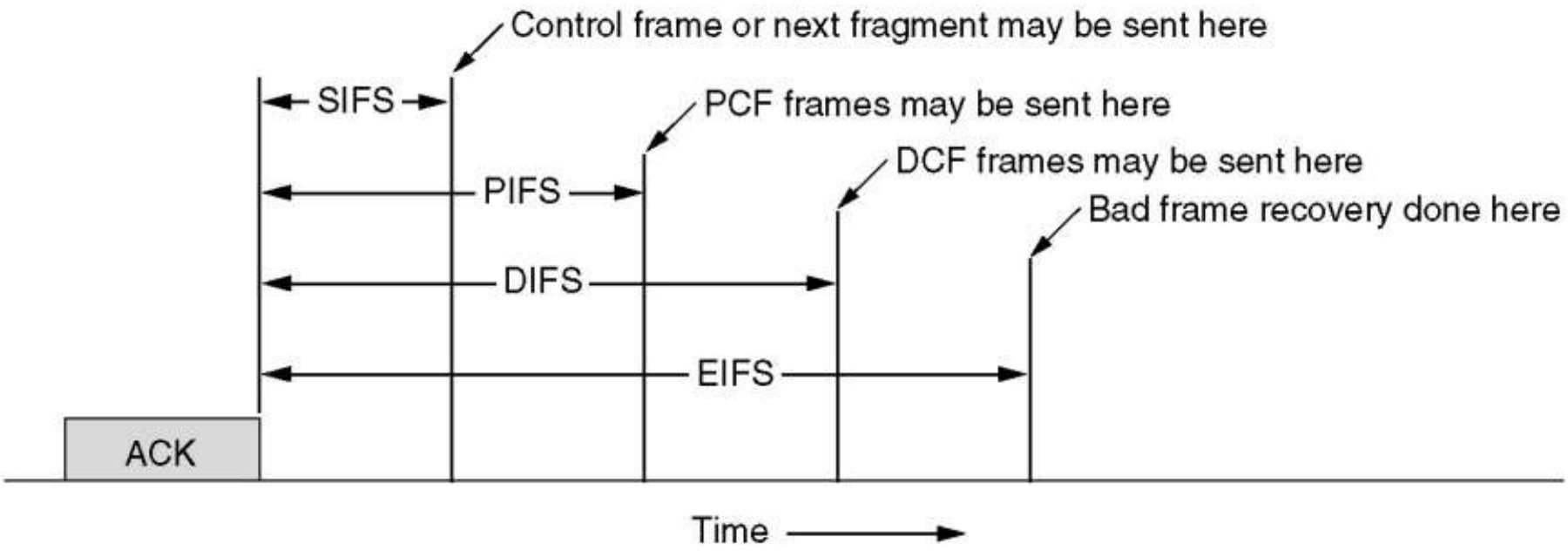
- 当要发送的数据较少时，一般不使用信道预约机制

信道预约: RTS-CTS交换



帧间距机制

- 802.11允许DCF和PCF在一个单元内共存，这是通过帧间距机制实现的。



帧间距机制（续）

- SIFS：允许正处于会话中的节点优先发送，如收到RTS的节点发送一个CTS，收到数据帧的节点允许发送一个ACK帧
- PIFS：如果在SIFS后没有节点发送，PCF模式的基站可以发送一个信标帧或一个轮询帧
- DIFS：如果PIFS后没有基站发送，任何节点可以竞争信道
- EIFS：如果以上间隔都没有发送，收到坏帧或未知帧的节点可以发送一个错误报告帧

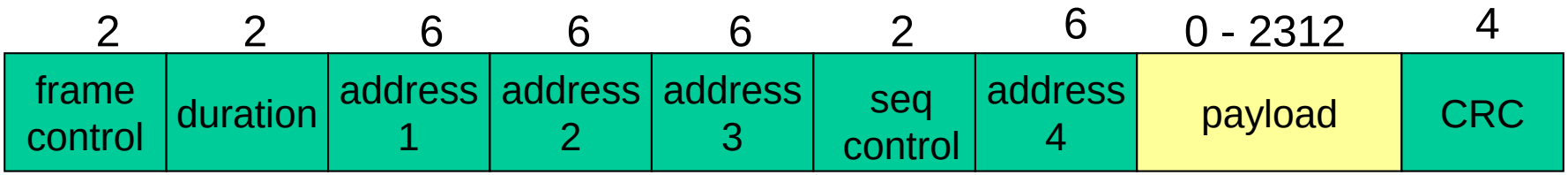
不使用信道预约机制的CSMA/CA

- 当节点有帧要发送时，侦听信道：
  1）若一开始就侦听到信道空闲，等待DIFS时间后发送帧
  2）否则，选取一个随机回退值，在侦听到信道空闲时递减该值；在此过程中若侦听到信道忙，冻结计数值
  3）当计数值减为0时，发送整个帧并等待确认
  4）若收到确认帧，表明帧发送成功，若还有新的帧要发送，从第2步开始CSMA/CA；若未收到确认，重新进入第2步中的回退阶段，并从一个更大的范围内选取随机回退值
- 可见，如果有k个节点等待发送，它们随机选取的回退值确定了它们的发送顺序

CSMA/CA与CSMA/CD的不同

- CSMA/CA与CSMA/CD最主要的不同：
  - CSMA/CD在发送过程中检测冲突，不使用确认机制
  - CSMA/CA在发送过程中不检测冲突，使用确认机制
- 由此带来的协议处理方面的不同：
  - 在CSMA/CD中，节点侦听到信道空闲时立即发送（不怕冲突，冲突后立即停发，损失不大）
  - 在CSMA/CA中，节点侦听到信道空闲后随机回退（冲突对无线网络损害很大，要尽量避免冲突）

（3）802.11帧格式



Address 1: 帧的目的MAC 地址

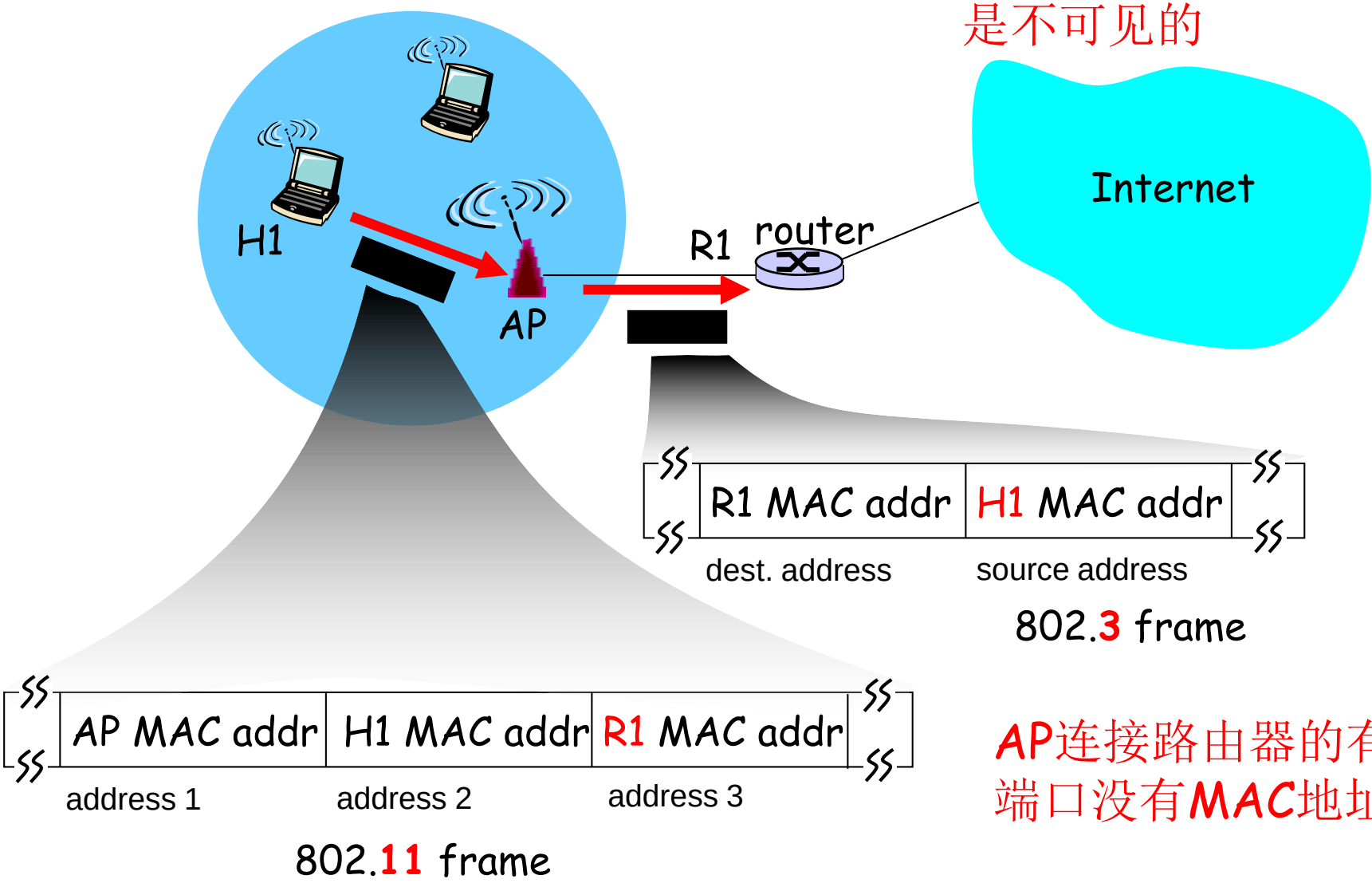
Address 2: 帧的源MAC地址

Address 3: 连接AP的路由器接口的MAC 地址

Address 4: 只在自组织模式中使用

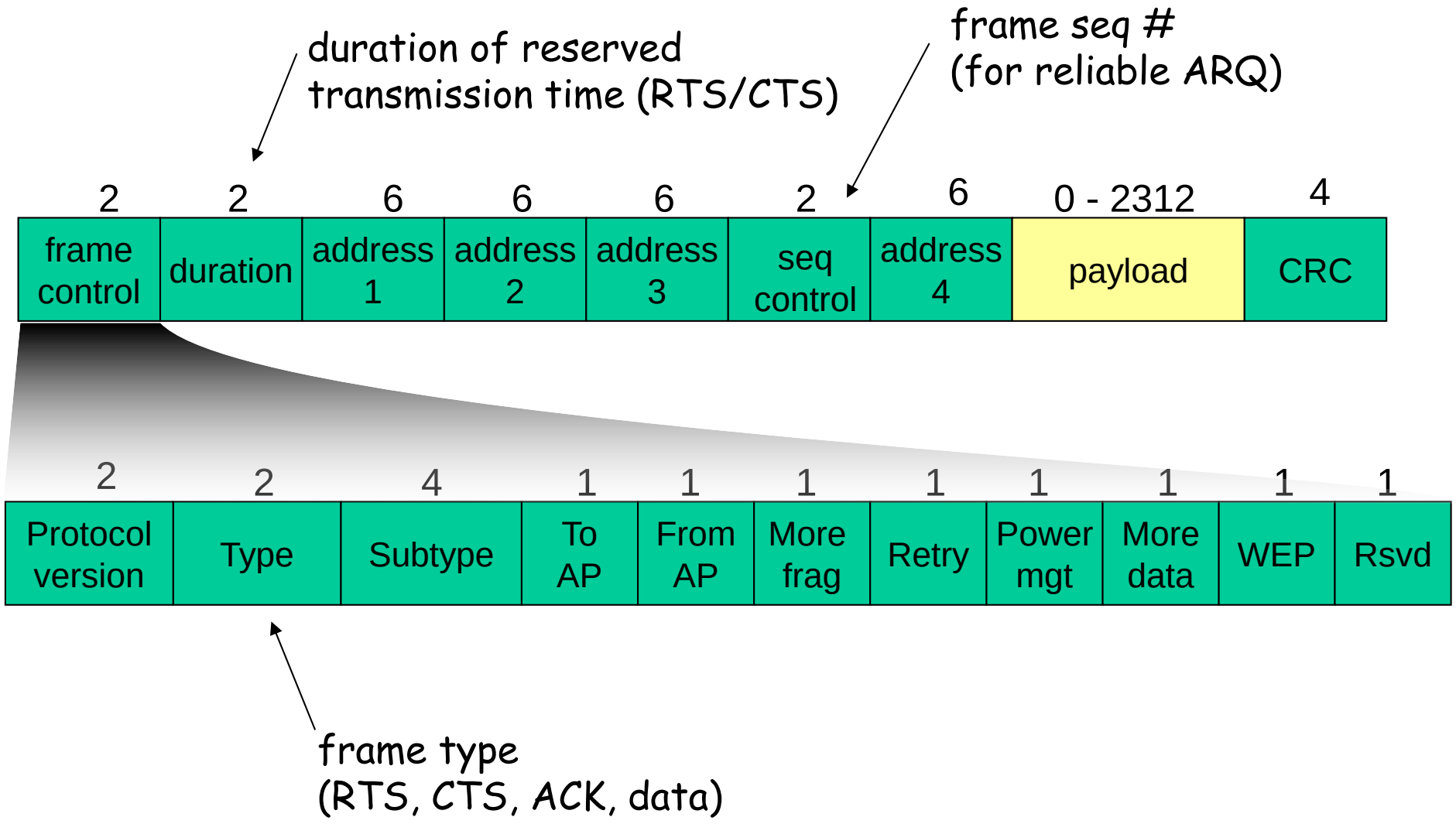
802.11帧：寻址举例

AP仅对无线终端可见，对于固定网络上的设备是不可见的



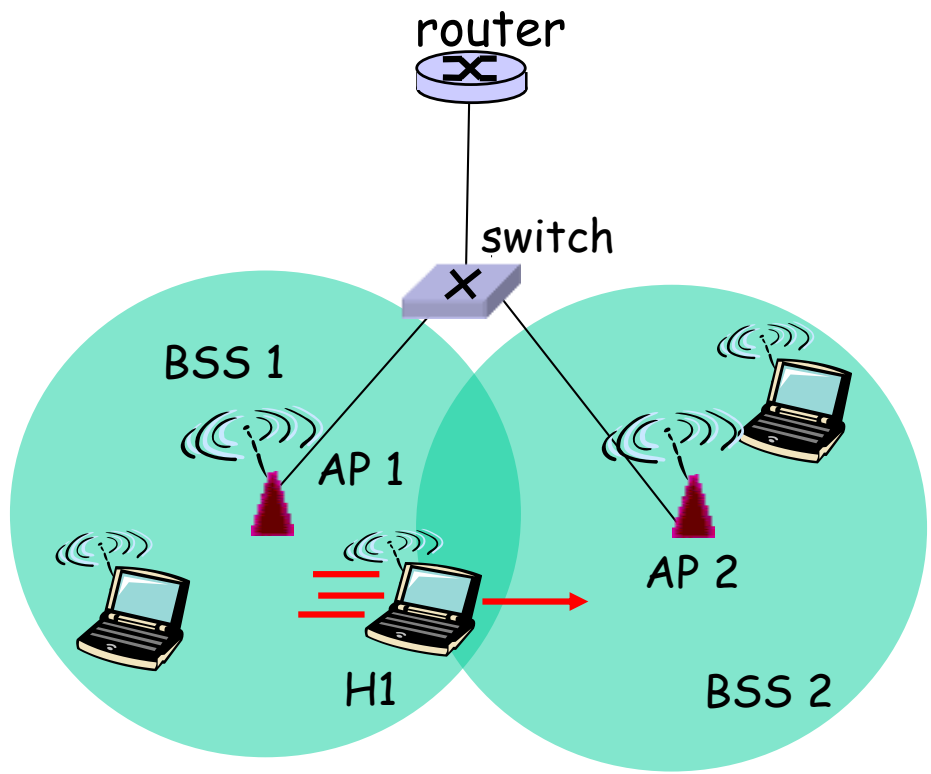
AP连接路由器的有线端口没有MAC地址！

802.11帧（续）


duration of reserved transmission time (RTS/CTS)
frame seq # (for reliable ARQ)
2
2
6
6
6
2
6
0 - 2312
4
frame control
duration
address 1
address 2
address 3
seq control
address 4
payload
CRC
2
2
4
1
1
1
1
1
1
1
1
Protocol version
Type
Subtype
To AP
From AP
More frag
Retry
Power mgt
More data
WEP
Rsvd
frame type (RTS, CTS, ACK, data)

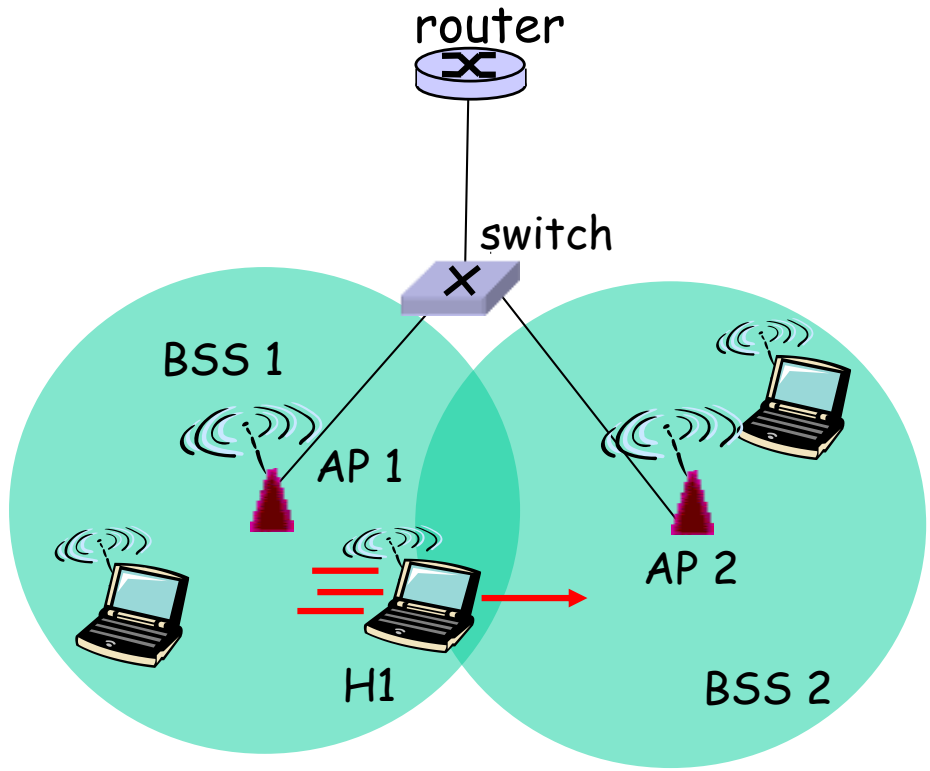
（4）802.11：终端在子网内移动

- 切换：
  - 终端从一个BSS移动到另一个BSS
- 发生切换时，终端要关联到新的AP上：
  - 当H1检测到来自AP1的信号逐渐减弱时，开始扫描新的信标帧
  - 当H1收到来自AP2、信号更强的信标帧时，先解除与AP1的关联，然后关联到AP2



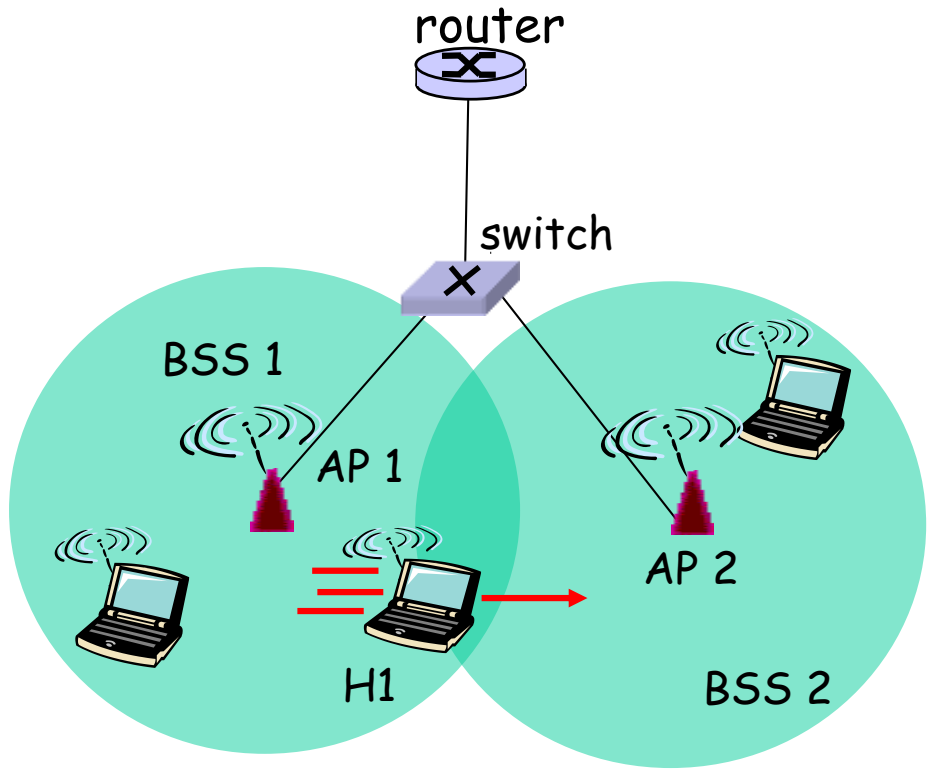
802.11：终端在子网内移动

- 发生切换时，交换机中的转发表也需要更新
- 交换机通过自主学习更新转发表：
  - 交换机收到H1发送的帧时，更新H1所在的端口
  - 若转发表未及时更新，可能产生丢包
- 802.11f规定了AP间漫游的方法



802.11：终端在子网内移动

- 若主机停留在同一个IP子网中，IP 地址保持不变
- 切换过程中，终端上的应用正常运行：
  - 由于IP地址没变，网络层及以上层次感觉不到这个移动
  - 切换过程中产生的延迟及丢包，在上层协议看来是正常的



（5）802.11: 先进功能

速率适应

- 当主机移动或信噪比变化时，基站和主机动态改变传输速率（物理层调制技术）

功率管理

- 节点设置功率管理比特，告知AP它将进入休眠状态：
  - 节点进入休眠，并在下一个信标帧之前醒来
  - 在节点休眠期间，AP缓存发往该节点的帧
  - AP在发送的信标帧中包含一个移动节点列表，这些节点有帧缓存在AP中
  - 列表中的节点向AP请求帧，其余节点重新进入休眠

需要掌握的知识点

- 无线网络的特性：
  - 隐藏节点，暴露节点
- CSMA/CA协议：
  - 信道预约机制
  - 无信道预约机制
- 终端在子网内移动：
  - 切换AP
  - 交换机更新转发表

Chapter 7 outline

7.1 Introduction

Wireless

7.2 Wireless links, characteristics

- CDMA

7.3 IEEE 802.11 wireless LANs (“Wi-Fi”)

7.4 Cellular Internet Access

- architecture
- standards (e.g., 3G, LTE)

Mobility

7.5 Principles: addressing and routing to mobile users

7.6 Mobile IP

7.7 Handling mobility in cellular networks

7.8 Mobility and higher-layer protocols

终端在IP子网间移动

- 终端进入到一个新的子网后，必须分配该子网上的一个地址（DHCP），并使用新的地址通信：
  - 终端在进入新的子网后，不能保留其IP地址
- 然而，当终端改变IP地址后，终端上正在运行的应用将中断：
  - 通信的对方不知道终端的新地址，无法与其通信
  - 即使对方获知了终端的新地址，应用必须重新建立连接，因为通信的端点（套接字）变了
- 能否在移动的过程中维持正在进行的连接？

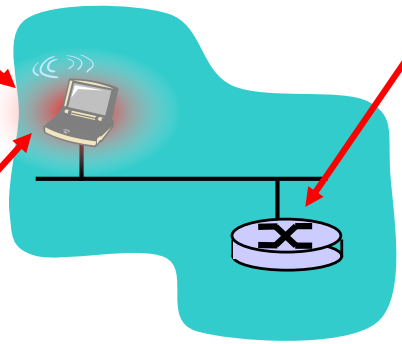
在移动中通信：人类类比的例子

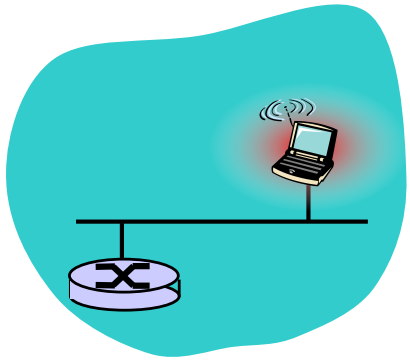
- 一个旅行爱好者经常外出旅行，朋友如何联系到他？
- 旅行者每到一地，立即将当前住址告知家里
- 旅行者的家庭地址是固定不变的
- 朋友可将信件寄到旅行者的家中，再由其家人转寄给旅行者；朋友也可用获得的新地址直接给旅行者寄信
- 旅行者使用了两个地址：
  - 家庭地址：固定不变
  - 当前地址：经常改变
- 家里总是知道旅行者的当前地址
- 朋友总是可以通过旅行者的家庭地址联系到他（直接或间接）
- 移动终端好比是旅行者
- 与移动终端通信的另一方好比是旅行者的朋友

一些术语

归属网络：移动节点的永久“居所”
(e.g., 128.119.40/24)

归属代理：当移动节点在外地时，为移动节点执行移动管理功能的实体





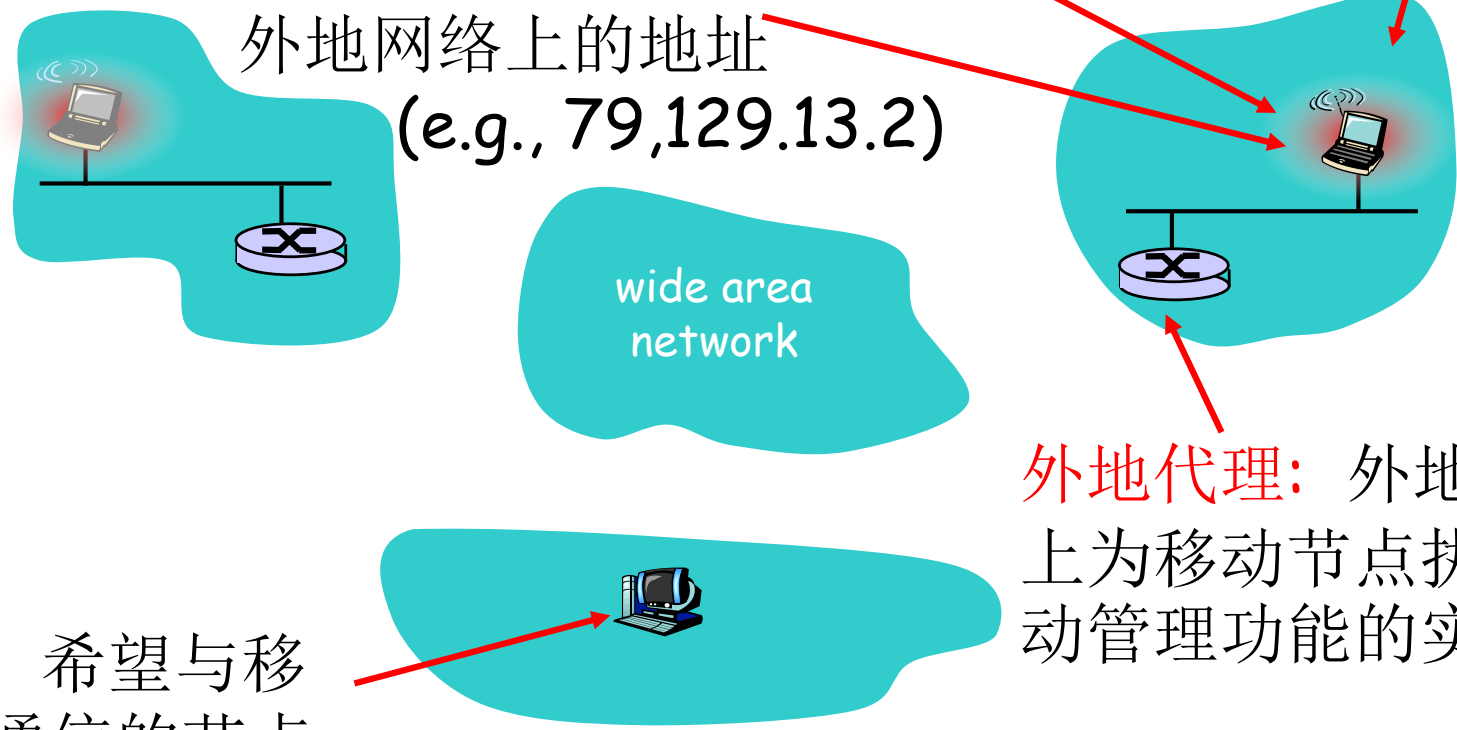
永久地址：移动节点在归属网络中的地址，总是可以使用这个地址与移动节点通信
e.g., 128.119.40.186

一些术语（续）

永久地址：保持不变
(e.g., 128.119.40.186)

外地网络：移动节点当前所在的网络
(e.g., 79.129.13/24)

转交地址：移动节点在外地网络上的地址
(e.g., 79,129.13.2)

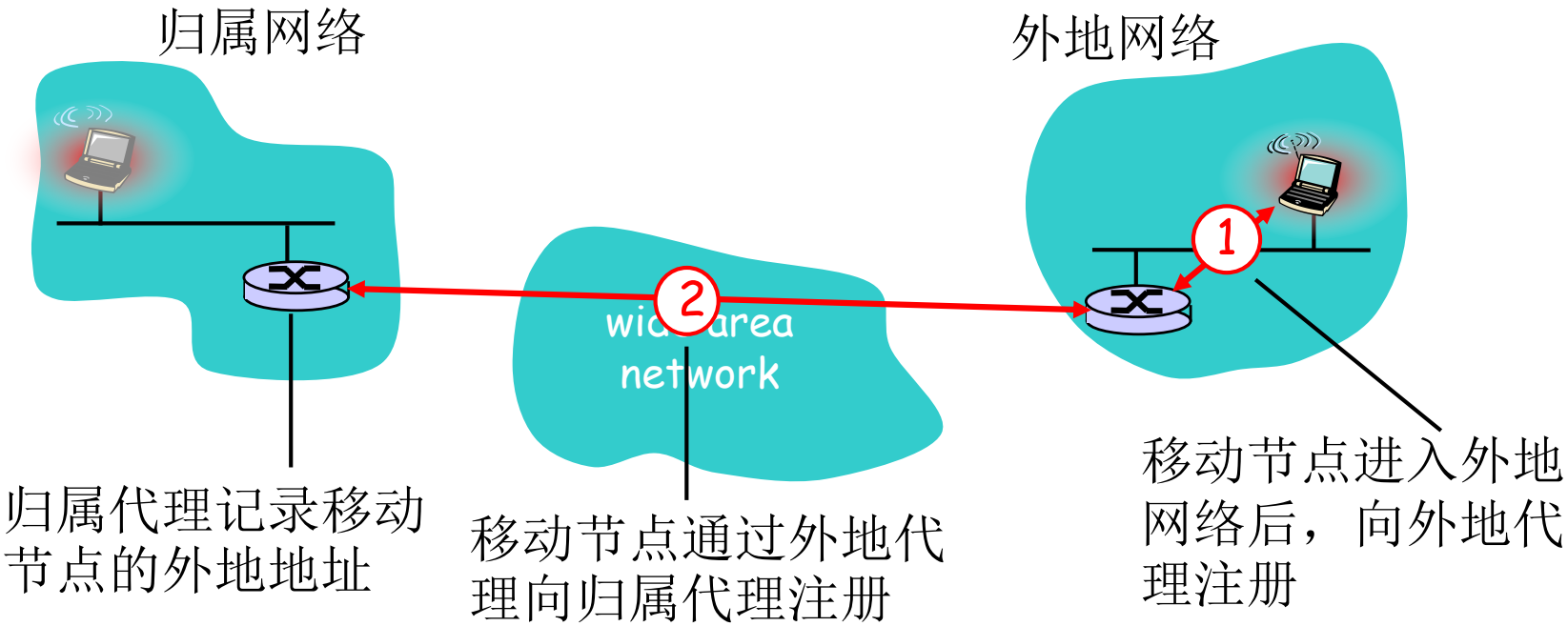


外地代理：外地网络上为移动节点执行移动管理功能的实体

通信者：希望与移动节点通信的节点

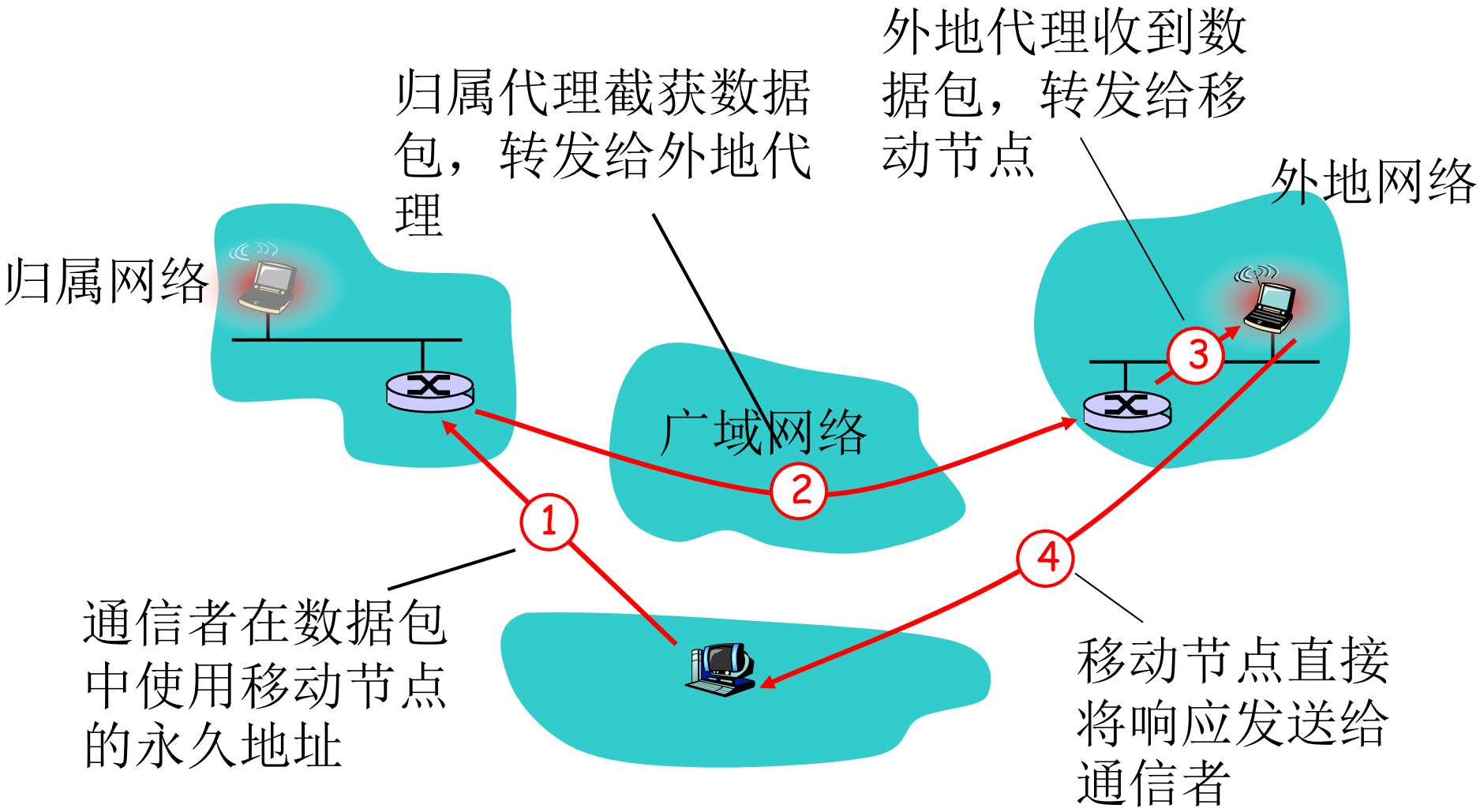
移动节点注册

- 注册：
  - 移动终端向归属代理通报其在外地网络的新地址



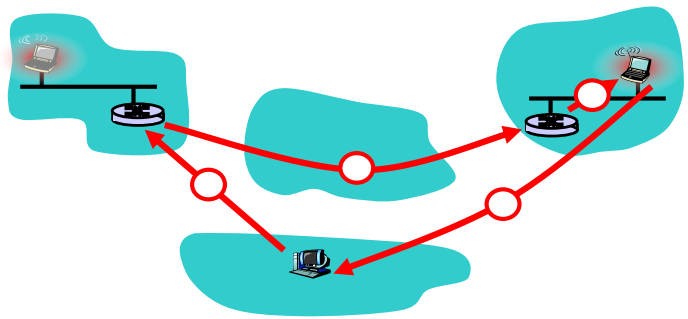
- 注册完成后：
  - 外地代理知道移动节点在本地网络上
  - 归属代理知道移动节点的转交地址，记录到地址绑定表中

间接选路到移动节点


外地代理收到数据包，转发给移动节点
归属代理截获数据包，转发给外地代理
外地网络
归属网络
广域网络
1
2
3
4
通信者在数据包中使用移动节点的永久地址
移动节点直接将响应发送给通信者

间接选路：三角选路问题

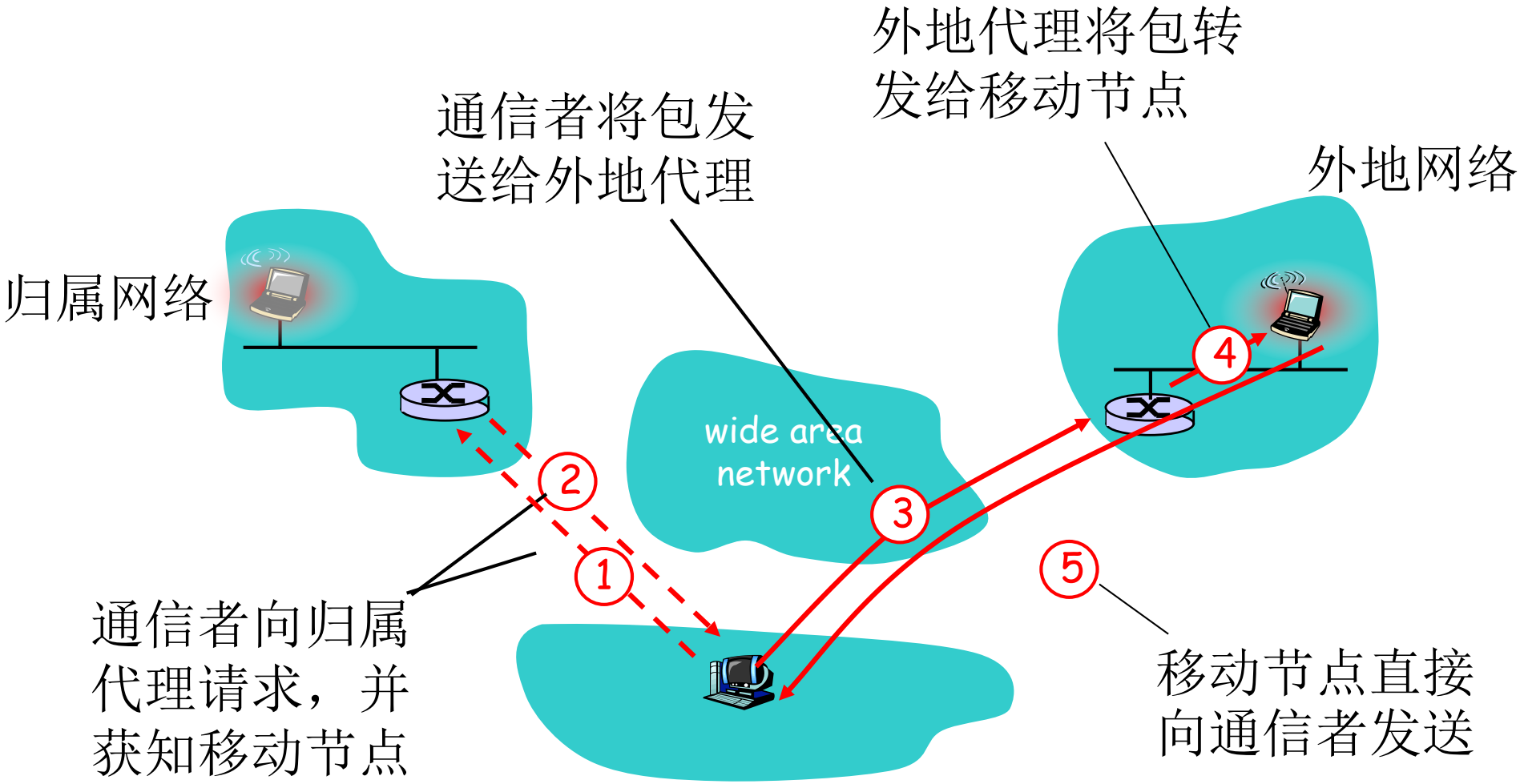
- 移动节点使用两个地址：
  - 永久地址：通信者用来向移动节点发送数据报（从而移动节点的位置对于通信者是透明的）
  - 转交地址：归属代理用来向移动节点转发数据报
- 三角选路：通信者-归属网络-移动节点
  - 当通信者和移动节点在同一个网络中时很低效



间接选路：终端在外地网络间移动

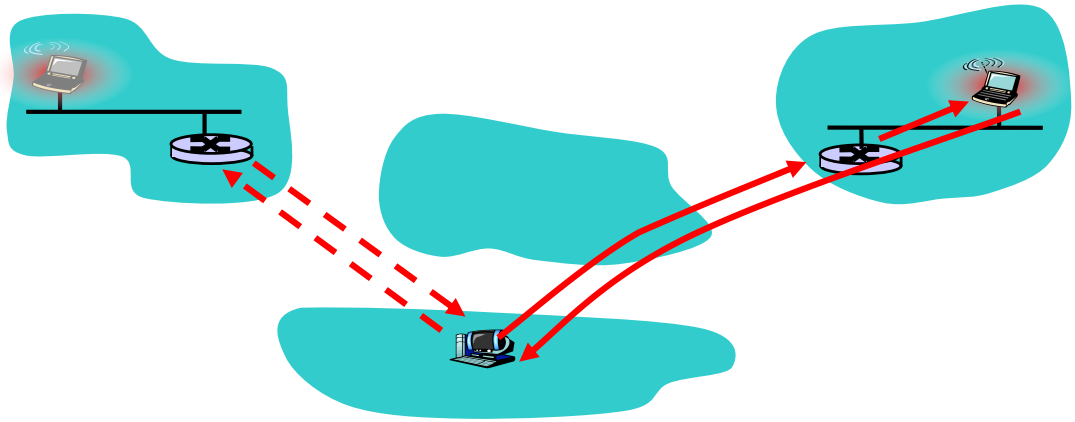
- 假设节点移动到另一个网络：
  - 向新的外地代理注册
  - 新的外地代理向归属代理注册
  - 归属代理更新移动节点的转交地址
  - 归属代理使用新的转交地址向移动节点转发包
- 节点移动及变换外地网络等对通信者都是透明的，移动节点的通信端点也没有改变，正在进行的通信可以保持！

直接选路到移动节点


外地代理将包转
发给移动节点
通信者将包发
送给外地代理
外地网络
归属网络
wide area
network
2
1
3
4
5
通信者向归属
代理请求，并
获知移动节点
的转交地址
移动节点直接
向通信者发送

直接选路: comments

- 克服了三角选路的问题
- 对通信者不透明:
  - 通信者需要知道移动节点的转交地址
  - 通信者（包括固定节点）需要增加对移动通信的支持



Chapter 7 outline

7.1 Introduction

Wireless

7.2 Wireless links, characteristics

- CDMA

7.3 IEEE 802.11 wireless LANs (“Wi-Fi”)

7.4 Cellular Internet Access

- architecture
- standards (e.g., 3G, LTE)

Mobility

7.5 Principles: addressing and routing to mobile users

7.6 Mobile IP

7.7 Handling mobility in cellular networks

7.8 Mobility and higher-layer protocols

Mobile IP

- 移动IP：支持移动性的因特网体系结构与协议
- 具有许多我们已经看到的特性，如：
  - 归属代理，外地代理，永久地址，转交地址，移动节点注册
- 标准化了三个部分：
  - 代理发现
  - 移动节点注册
  - 数据报间接选路

（1）代理发现

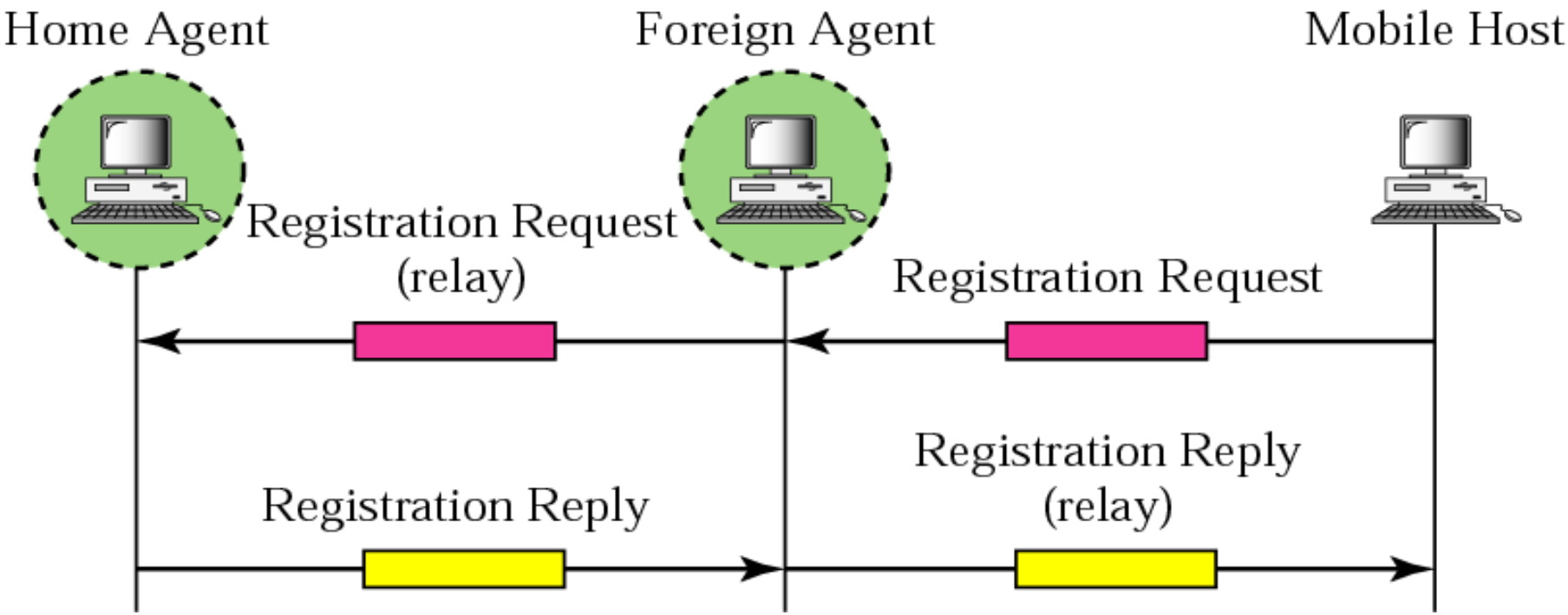
- 愿意充当归属代理或外地代理的路由器定期在网络上发送代理通告，宣布自己的存在及IP地址
- 愿意充当外地代理的路由器在代理通告中会提供一个或多个转交地址（通常使用自己的IP地址作为转交地址）
- 移动节点通过接收和分析代理通告，判断自己是否处于外地网络以及是否切换了网络
- 如果发现在外地网络上，移动节点从外地代理提供的转交地址中选择一个作为自己的转交地址

（2）移动主机注册

- 移动节点向外地代理发送一个注册请求，给出自己的永久地址、转交地址、归属代理地址以及认证信息等
- 外地代理记录相关信息，向归属代理转发注册请求
- 归属代理处理注册请求，若认证通过，将移动节点的永久地址及转交地址保存在绑定表中，发回一个注册响应
- 外地代理收到有效的注册响应后，将移动节点记录在自己的转发表中，向移动节点转发注册响应

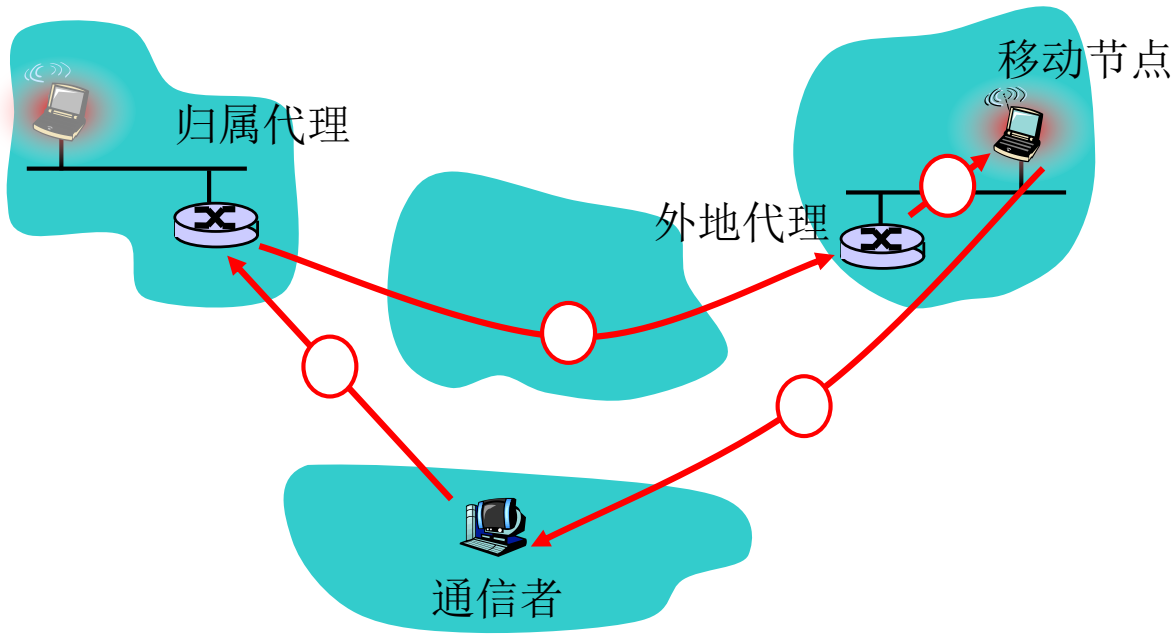
- 当移动节点回到归属网络时，要向归属代理注销

注册请求和响应

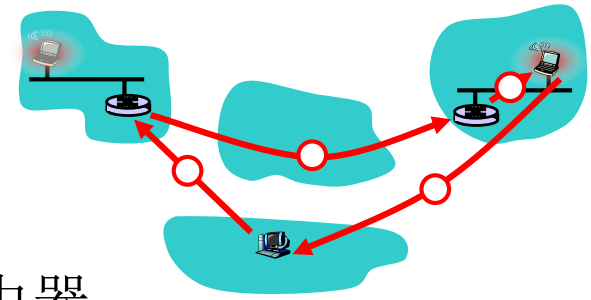

Home Agent
Foreign Agent
Mobile Host
Registration Request
(relay)
Registration Request
Registration Reply
Registration Reply
(relay)

（3）数据报间接选路

- 数据包首先被归属代理得到
- 归属代理查找地址绑定表，获得移动节点当前的转交地址
- 归属代理将数据包发送到转交地址（外地代理）
- 外地代理将数据包转发给移动节点



归属代理如何得到数据报？

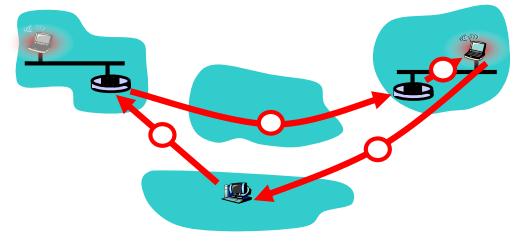


- 若通信者不在归属网络上：
  - 数据包首先到达移动节点归属网络上的路由器
  - 路由器查表得知可以直接交付，于是查找ARP缓存或者发送ARP请求，得到与移动节点永久地址对应的MAC地址
  - 利用得到的MAC地址，将数据报封装到链路层帧中发送
- 若通信者在归属网络上：
  - 通信者查表得知移动节点直接可达，于是查找ARP缓存或者发送ARP请求，得到与移动节点永久地址对应的MAC地址，封装数据报发送
- **数据报如何能被归属代理得到？**
  - 帧的目的地址＝归属代理的MAC地址
  - 也就是说，**移动节点的永久地址** 应当映射到**归属代理的MAC地址。怎么做到？**

归属代理如何得到数据报？

- 归属代理作为移动主机的ARP代理：
  - 归属代理为位于外地网络的移动主机发送ARP响应，用自己的MAC地址进行响应
  - 也就是说，**将移动主机的永久地址映射到归属代理的MAC地址**
- 归属代理发送免费ARP：
  - 当接收到移动主机的注册请求后，**归属代理主动发送ARP请求（将移动主机的永久地址关联到归属代理的MAC地址）**，刷新其它节点的ARP缓存

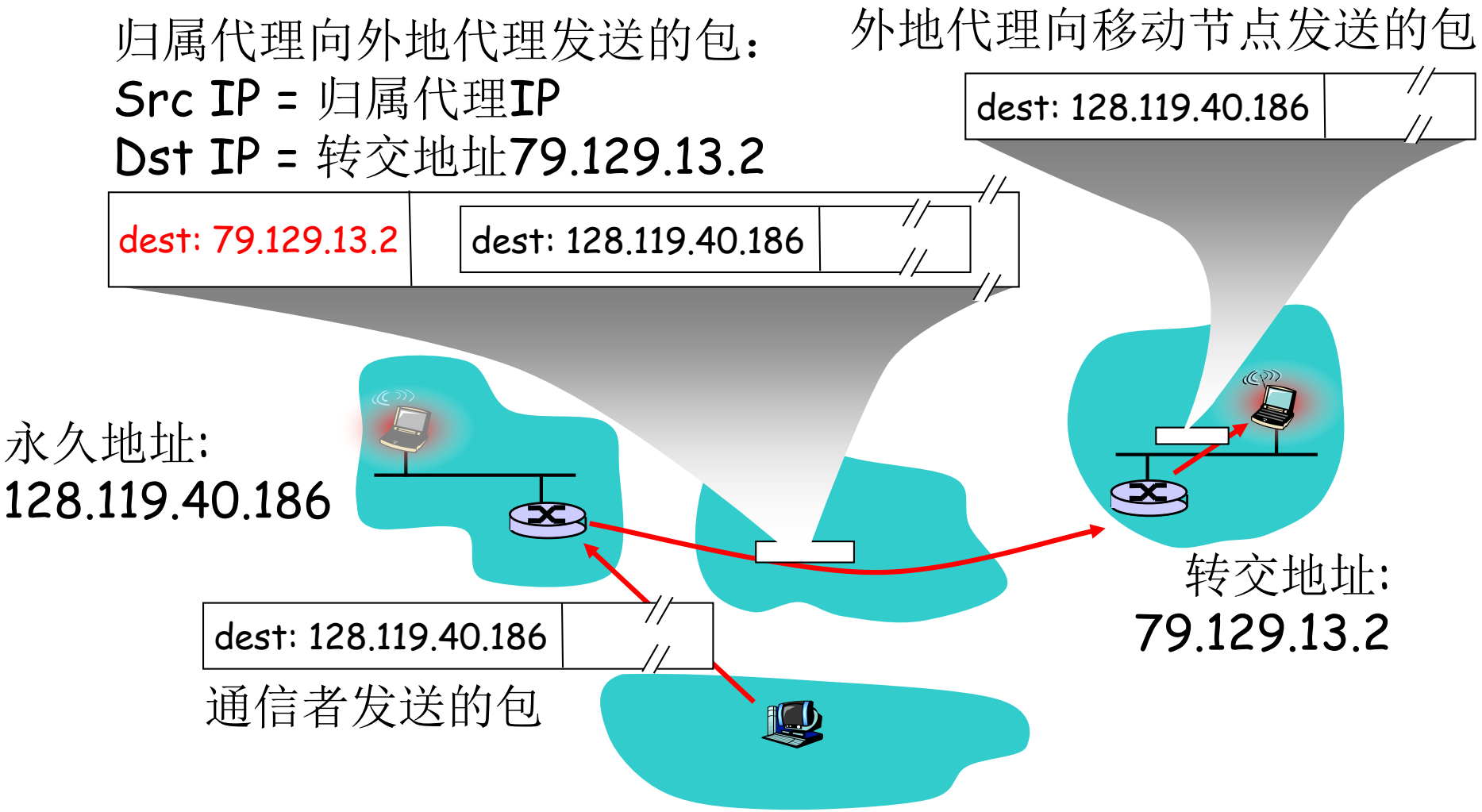
数据报如何到达转交地址？



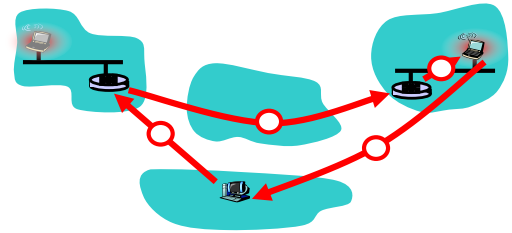
- 归属代理如何将数据报发送到转交地址？
  - 归属代理收到的数据报，目的地址为移动节点的永久地址，而移动节点的转交地址位于外地网络
  - 如何将目的地址在本地的数据报送达外地网络？

- 方法：
  - 修改目的地址=转交地址 （×）（转交地址是外地代理的IP地址，而数据报的最终目的地应是移动节点）
  - 使用隧道（√）（隧道技术的又一个应用例子）

归属代理通过隧道转发数据包

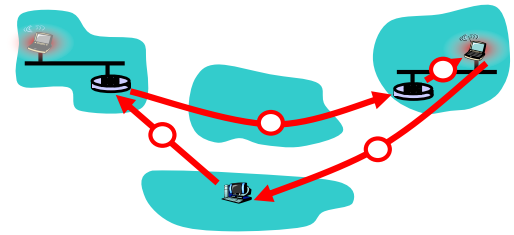


外地代理如何转发数据包到移动节点？



- 外地代理解封装收到的数据包，得到原始数据报
- 外地代理如何获得移动节点的MAC地址？
  - 在移动节点注册阶段，外地代理获知了移动节点的永久地址和MAC地址，记录在其转发表中
  - 外地代理利用目的IP地址查找转发表，得到移动节点的MAC地址
- 外地代理利用移动节点的MAC地址，将数据报封装到链路层帧中，发送给移动节点

移动节点如何发送数据包？



- 移动节点将数据包发送给外地代理（缺省路由器）：
  - SrcIP=移动节点永久地址，DestIP=通信者IP地址
  - SrcMAC=移动节点MAC，DestMAC=外地代理MAC
- 外地代理按照正常方式转发数据包

- 移动节点如何得知外地代理的MAC地址？
  - 代理通告报文的源MAC是外地代理的地址

需要掌握的知识点

- 移动通信相关的概念：
  - 归属网络，外地网络，归属代理，外地代理，永久地址，转交地址
- 移动节点注册
- 间接选路和直接选路
- 间接选路方式下数据报如何传输：
  - 归属代理如何得到数据报
  - 数据报如何到达转交地址
  - 数据报如何到达移动节点

Chapter 7 outline

7.1 Introduction

Wireless

7.2 Wireless links, characteristics

- CDMA

7.3 IEEE 802.11 wireless LANs (“Wi-Fi”)

7.4 Cellular Internet Access

- architecture
- standards (e.g., 3G, LTE)

Mobility

7.5 Principles: addressing and routing to mobile users

7.6 Mobile IP

7.7 Handling mobility in cellular networks

7.8 Mobility and higher-layer protocols

无线和移动对上层协议的影响

- 无线链路带来的问题：
  - 误码率高：传输距离、环境干扰、多径传输造成信噪比降低
  - 丢包率高：由于误码、发送冲突造成传输失败
  - 延迟增大：由于冲突重传造成延迟增大

- 节点移动带来的问题：
  - 丢包增加：切换（AP切换、交换机更新转发表、移动IP注册）造成丢包
  - 延迟增大：切换及间接选路带来延迟增大

无线和移动对上层协议的影响

- 逻辑上，没什么影响：
  - 为上层协议提供的仍然是尽力而为的服务，因此TCP和UDP也可以运行在无线网络上
- 性能上，有很大影响：
  - 丢包率升高，传输延迟增大
  - TCP将丢包（长延迟也当作丢包）解释为拥塞，不必要地减小拥塞窗口，导致应用吞吐率很低
  - 无线链路、有线/无线混合链路上的TCP拥塞控制是一个研究问题

Chapter 7 Summary

7.1 Introduction

Wireless

7.2 Wireless links, characteristics

- CDMA

7.3 IEEE 802.11 wireless LANs ("Wi-Fi")

7.4 Cellular Internet Access

- architecture
- standards (e.g., 3G, LTE)

Mobility

7.5 Principles: addressing and routing to mobile users

7.6 Mobile IP

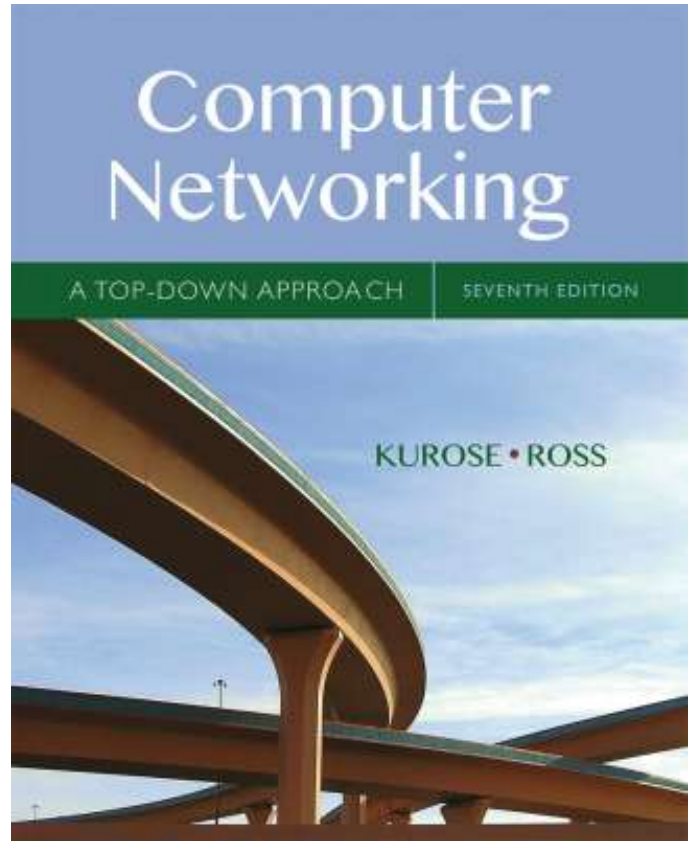
7.7 Handling mobility in cellular networks

7.8 Mobility and higher-layer protocols

作业

- 习题：R7，R11，P5，P6
- 提交时间：12月5日

Chapter 8
Security



Computer Networking: A Top Down Approach

7th edition

Jim Kurose, Keith Ross

Pearson/Addison Wesley

April 2016

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Chapter 8: Network Security

Chapter goals:

- 理解网络安全原理:
  - 加密
  - 身份鉴别
  - 报文完整性
- 网络安全应用：
  - 通信安全：应用层、传输层、网络层和链路层
  - 运行安全：防火墙和入侵检测系统

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and Digital Signature

8.4 End point authentication

8.5 Securing e-mail

8.6 Securing TCP connections: SSL

8.7 Network layer security: IPsec

8.8 Securing wireless LANs

8.9 Operational security: firewalls and IDS

什么是网络安全？

网络安全的通用定义：

- 网络安全是指网络系统的硬件、软件及其系统中的数据受到保护，不受偶然的或者恶意的原因而遭到破坏、更改、泄露，系统连续可靠地运行，网络服务不中断。

网络中的通信安全

- 机密性:
  - 报文内容的机密性：仅发送方和希望的接收方能够理解报文的内容
  - 通信活动的机密性：通信活动或其特征不被外界察觉
- 端点鉴别:
  - 发送方和接收方都能够证实通信过程中涉及的另一方
- 报文完整性:
  - 报文来自真实的来源，且传输过程中未被修改
- 运行安全性:
  - 网络系统正常运行，网络服务可用

安全攻击的类型：被动攻击

- 试图从系统中获取信息，但不对系统产生影响
- 两种类型：
  - 偷听：监听并记录网络中传输的内容
  - 流量分析：从通信频度、报文长度等流量模式推断通信的性质

安全攻击的类型：主动攻击

- 试图改变系统资源或影响系统的操作
- 四种类型：
  - 伪装：一个实体假冒另一个实体
  - 重放：从网络中被动地获取一个数据单元，经过一段时间后重新发送到网络中
  - 报文修改：修改、插入、删除报文或报文部分内容
  - 拒绝服务：阻止通信设施的正常使用或管理

常见的安全机制

- 加密：使用数学算法对数据进行变换，使其不易理解
- 鉴别：通过报文交换证实一个实体的身份，以防假冒
- 报文完整性：验证一个报文是否可信，包括来源和内容
- 数字签名：附加在一个数据单元后面的数据，用来证明数据单元的来源及完整性，以防伪造及抵赖
- 流量填充：在数据流间隙中插入比特，以挫败流量分析的企图
- 访问控制：通过授权机制限制用户对资源的访问，防止越权行为

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and Digital Signature

8.4 End point authentication

8.5 Securing e-mail

8.6 Securing TCP connections: SSL

8.7 Network layer security: IPsec

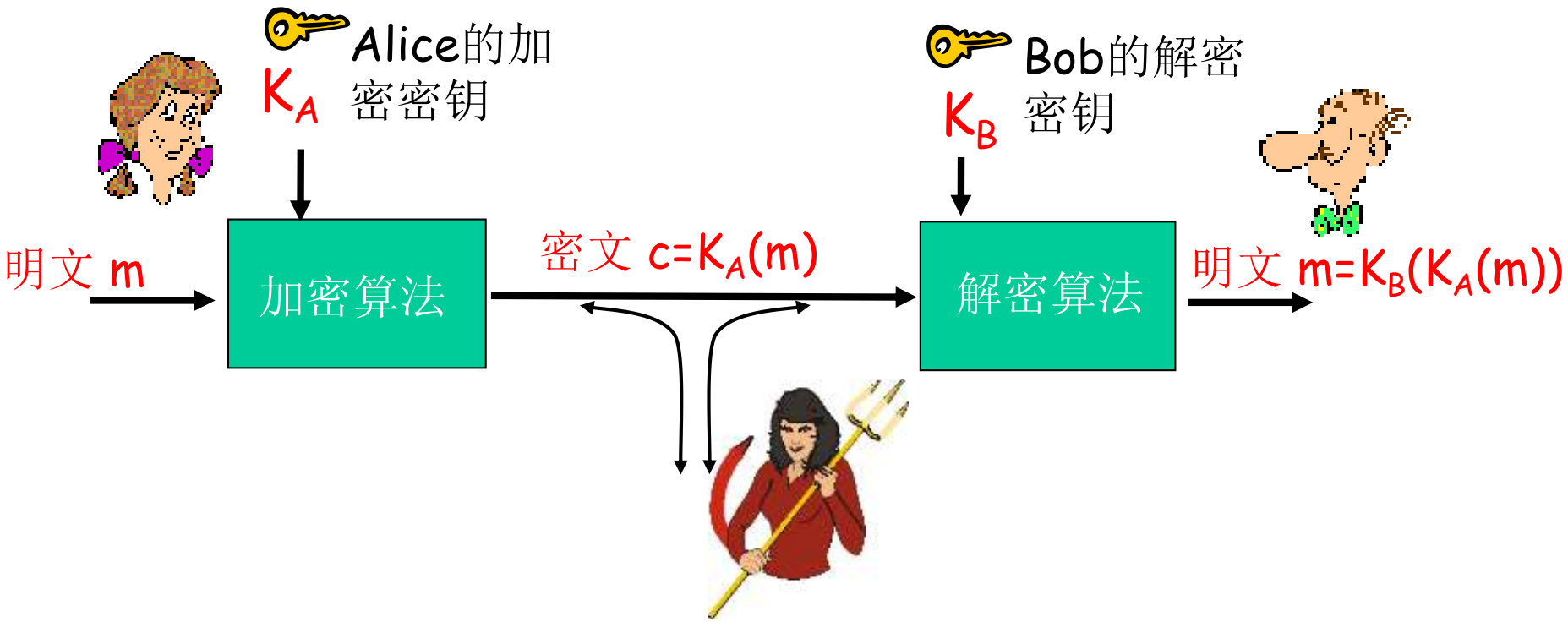
8.8 Securing wireless LANs

8.9 Operational security: firewalls and IDS

密码学术语

- 明文（plaintext）：欲加密的原始数据
- 密文（ciphertext）：明文经加密算法作用后的输出
- 密钥（key）：加密和解密时使用的参数
- 密码分析（cryptanalysis）：破译密文
- 密码学（cryptology）：设计密码和破译密码的技术统称为密码学

密码学术语（图示）



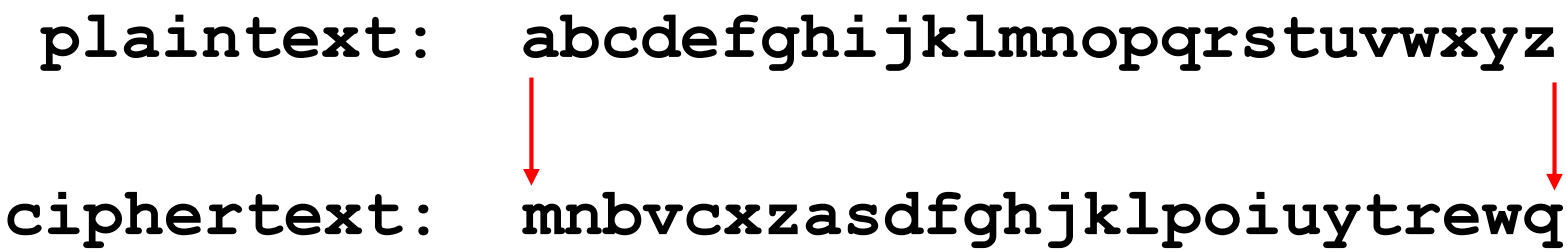
加密算法的分类

- 按照加密密钥与解密密钥是否相同，加密算法分为：
  - 对称加密算法：加密密钥与解密密钥相同
  - 非对称加密算法：加密密钥与解密密钥不同
- 按照明文被处理的方式，加密算法分为：
  - 块密码（分组密码）：每次处理一个明文块，生成一个密文块
  - 流密码：处理连续输入的明文流，并生成连续输出的密文流

传统加密方法：替换

- 替换密码：
  - 用密文字母替换明文字母，但字母位置不变
  - 例子：凯撒密码，单字母表替换，多字母表替换

单字母表替换的例子：



密钥：字母映射表

传统加密方法：换位

- 换位密码：
  - 保留明文字母不变，但改变字母的位置
  - 例子：列换位密码

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Plaintext

pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Ciphertext

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

密码的安全性

- 传统加密方法的安全性建立在算法保密的基础上
- 现代加密方法也使用替换和换位两种基本手段，但现代密码学的基本原则是：
  - **加密与解密的算法是公开的，只有密钥是需要隐藏的**
- 一个加密算法被称为是**计算安全**的，如果由该算法产生的密文满足以下两个条件之一：
  - 破译密文的代价超过信息本身的价值
  - 破译密文所需的时间超过信息的有效生命期
- 现代密码学中，**密码的安全性是通过算法的复杂性和密钥的长度来保证的**

针对加密系统的密码分析攻击

- 惟密文攻击：
  - 密码分析者仅能根据截获的密文进行分析，以得到明文或密钥（对密码分析者最不利的情况）
- 已知明文攻击：
  - 密码分析者除了有截获的密文外，还有一些已知的"明文-密文对"来帮助破译密码，以得出密钥
- 选择明文攻击：
  - 密码分析者可以任意选择一定数量的明文，用被攻击的加密算法加密，得到相应的密文，以利于将来更有效地破解由同样加密算法及相关密钥加密的信息
- **一个安全的加密系统必须能抵御选择明文攻击**

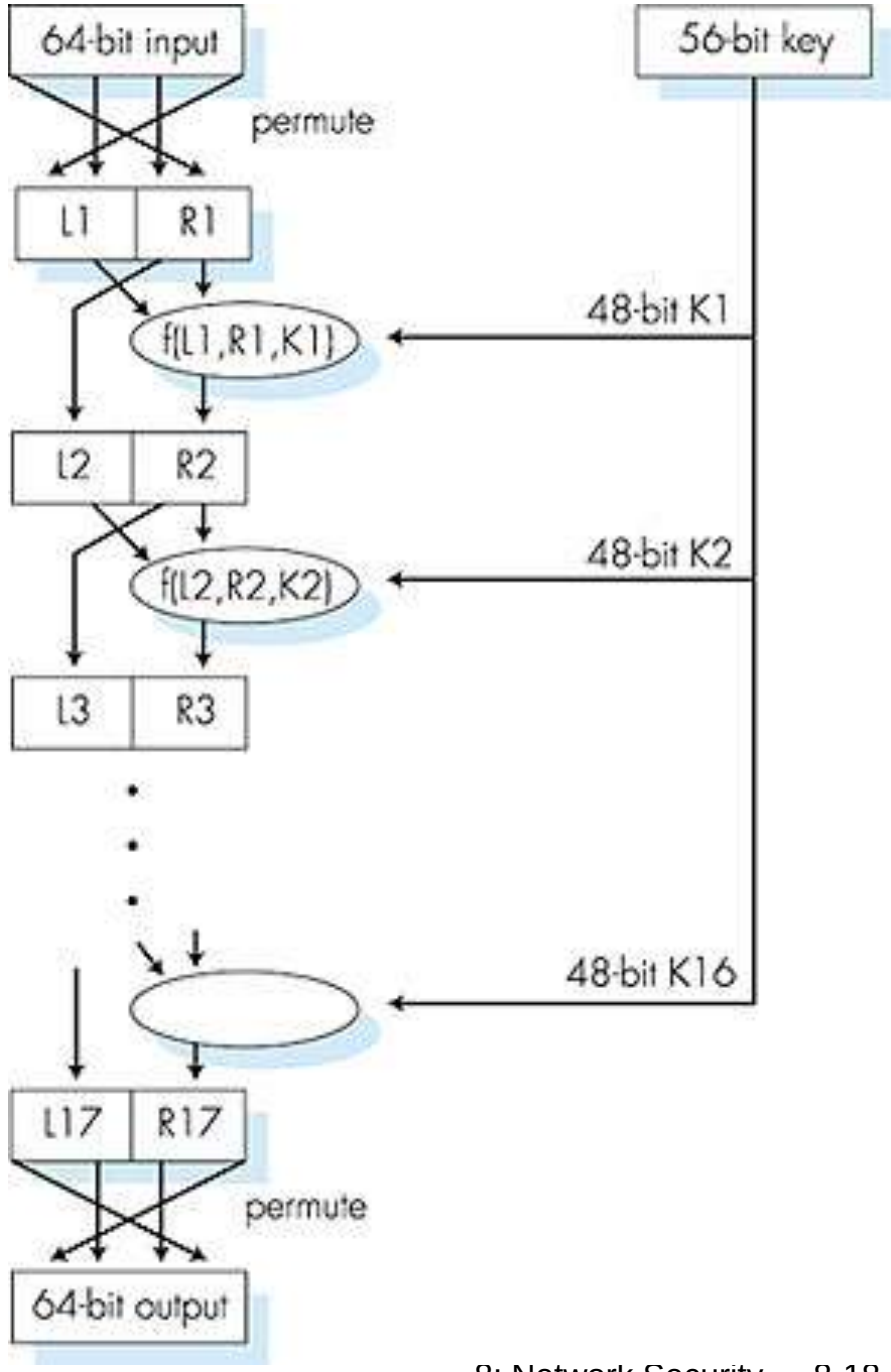
现代密码学：对称密钥算法

DES: Data Encryption Standard

- 1997年成为美国的数据加密标准
- DES是一种块加密算法，每次以64比特的明文块作为输入，输出64比特的密文块
- DES是基于迭代的算法，每一轮迭代执行相同的替换和换位操作，但使用不同的密钥
- DES使用一个56比特的主密钥，每一轮迭代使用的子密钥（48比特）由主密钥产生
- **DES是对称加密算法，加密和解密使用相同的函数，两者的不同只是子密钥的次序刚好相反**
- 缺点：密钥长度不够长，迭代次数不够多

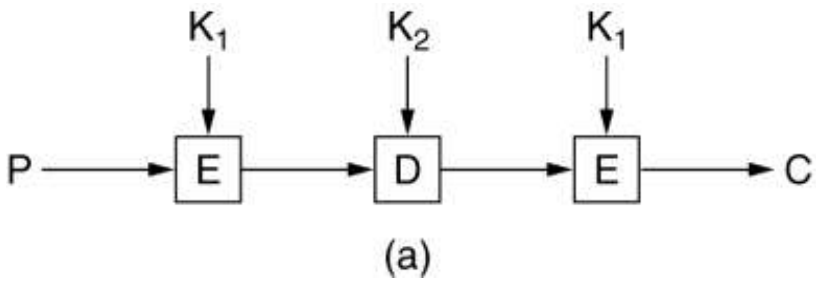
DES的计算过程

- 首先进行一次初始换位（permute）
- 然后进行16轮相同的迭代，每轮迭代使用一个不同的子密钥（由主密钥生成）
- 最后再进行一次换位（与初始换位相反）

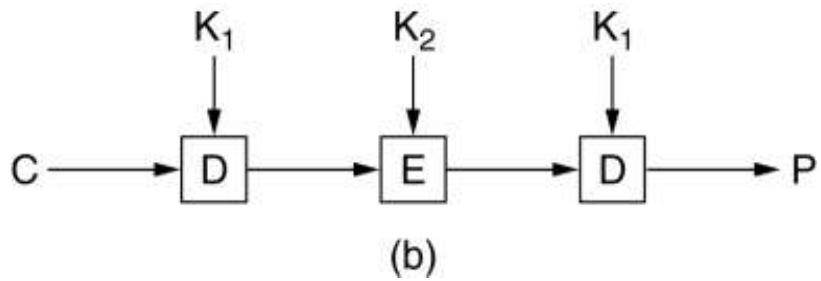


Triple DES（3DES）

- 3DES使用两个密钥进行三轮DES计算：
  - 第一轮令DES设备工作于加密模式，使用密钥K_1对明文进行变换
  - 第二轮令DES设备工作于解码模式，使用密钥K_2对第一轮的输出进行变换
  - 第三轮令DES设备工作于加密模式，用密钥K_1对第二轮的输出进行变换，输出密文



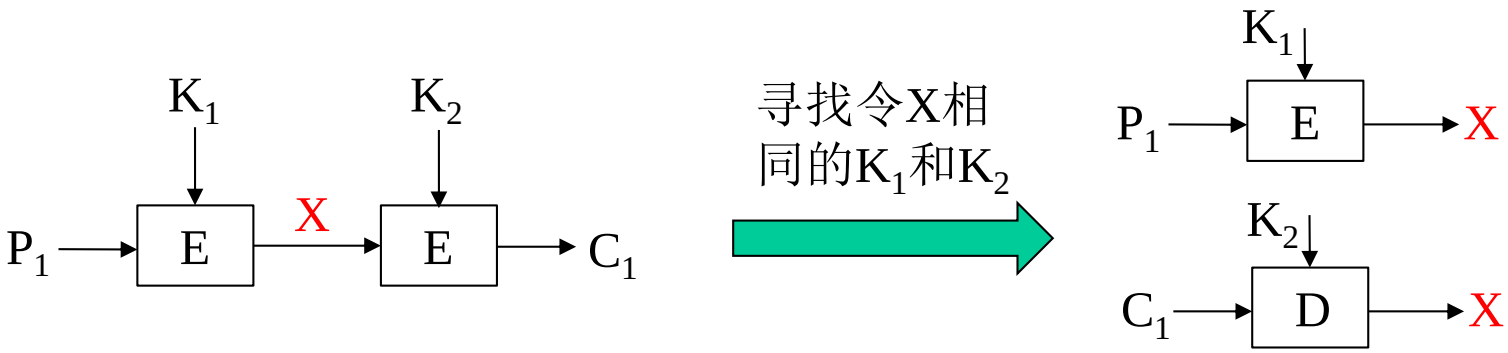
(a)



(b)

有关3DES的三个问题

- 为什么使用两个密钥而不是三个密钥？
  - 112比特的密钥已经足够长
- 为什么不使用两重DES（EE模式）而是三重DES？
  - 考虑采用EE模式的两重DES，且攻击者已经拥有了一个匹配的明文--密文对（P_1, C_1），即有$C_1 = E_{K2}(E_{K1}(P_1))$
  - 令$X = E_{K1}(P_1) = D_{K2}(C_1)$。攻击者分别计算$E_{K1}(P_1)$和$D_{K2}(C_1)$，并寻找使它们相等的$K_1$和$K_2$，则穷尽整个密钥空间只需$2^{56}$的攻击量而不是$2^{112}$。（中途攻击）



有关3DES的三个问题

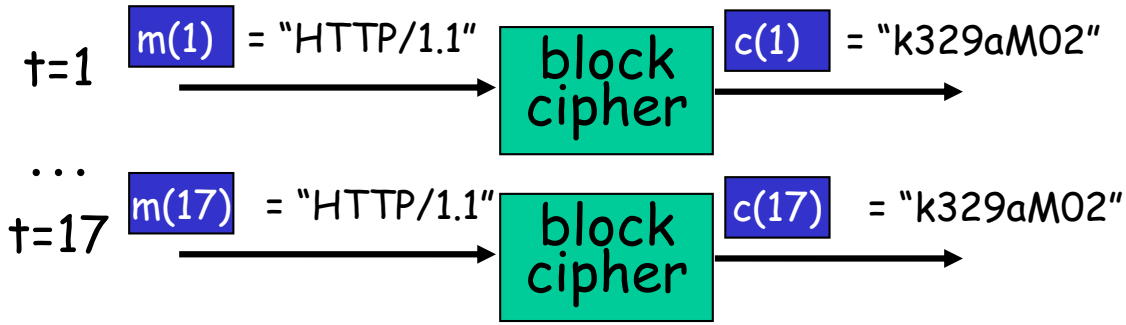
- 为什么使用两个密钥而不是三个密钥？
  - 112比特的密钥已经足够长
- 为什么不使用两重DES（EE模式）而是三重DES？
  - 考虑采用EE模式的两重DES，且攻击者已经拥有了一个匹配的明文--密文对（P_1, C_1），即有$C_1 = E_{K2}(E_{K1}(P_1))$
  - 令$X = E_{K1}(P_1) = D_{K2}(C_1)$。攻击者分别计算$E_{K1}(P_1)$和$D_{K2}(C_1)$，并寻找使它们相等的$K_1$和$K_2$，则穷尽整个密钥空间只需$2^{56}$的攻击量而不是$2^{112}$。（中途攻击）
- 为什么是EDE而不是EEE？
  - 为了与单次DES兼容。3DES用户解密单次DES用户加密的数据，只需令$K_1 = K_2$就行了。

AES: Advanced Encryption Standard

- 2001年11月成为新的对称加密标准，代替DES
- 每次处理128比特明文块，输出128比特密文块
- 密钥长度可以是128、192或256比特
- 如果使用强力方法破解，假设破解DES需要1秒，则破解AES（128比特密钥）需要149万亿年！

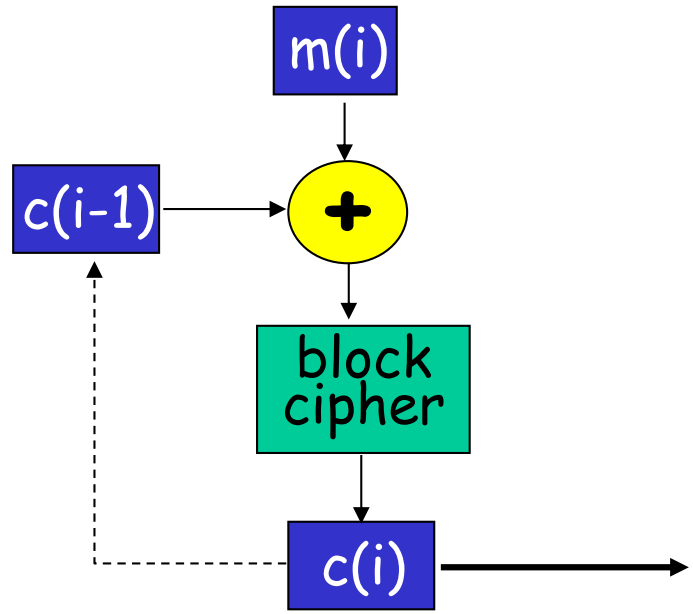
密码块链接（Cipher Block Chaining）

- 若每个明文块被独立加密，相同的明文块生成相同的密文块，容易被重放攻击利用。



密码块链接（CBC）：

- 发送方生成一个随机的初始向量c(0)，用明文发送给接收者
- 每一个明文块加密前，先与前一个密文块进行异或，然后再加密：
  - 第一个明文块与c(0)异或
- 相同的明文块几乎不可能得到相同的密文块



现代密码学：非对称加密

对称加密算法：

- 要求发送者和接收者使用同一个密钥

存在密钥传递问题：

- 发送方选择了一个密钥后，如何将密钥安全地传递给接收方？

非对称加密算法：

- 发送者和接收者不共享密钥
  - 发送者使用加密密钥
  - 接收者使用解密密钥

不存在密钥传递问题：

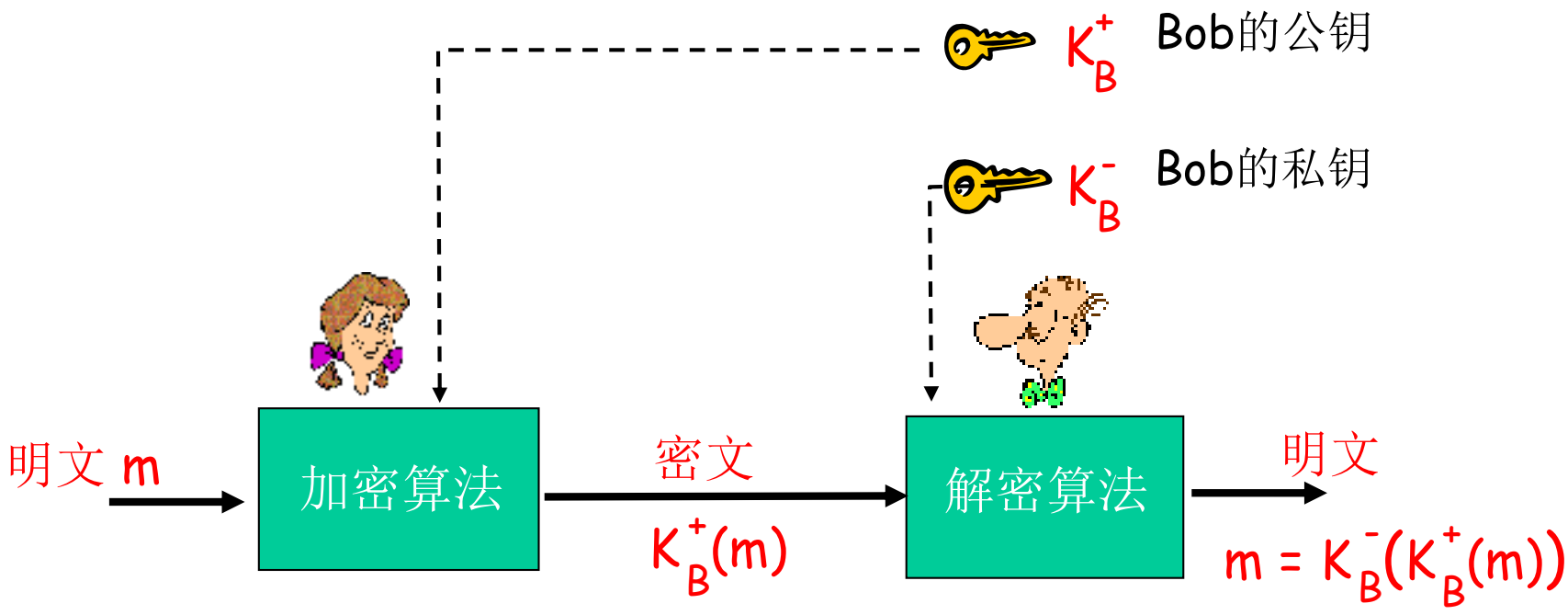
- 加密密钥是公开的
- 解密密钥是私有的

公开密钥算法的使用

- 每个用户生成一对加密密钥和解密密钥：
  - 加密密钥放在一个公开的文件中，解密密钥妥善保管
- 当Alice希望向Bob发送一个加密信息时：
  - Alice从公开的文件中查到Bob的加密密钥，用Bob的加密密钥加密信息，发送给Bob
- Bob用自己的解密密钥解密信息

- 公开密钥和私有密钥：
  - 公开密钥：加密密钥，由发送者使用
  - 私有密钥：解密密钥，由接收者使用

公开密钥算法的使用示例



要求: ① $K_B^-(K_B^+(m)) = m$

② 给定公钥K_B^+，不可能计算出私钥K_B^-

公开密钥算法应满足的条件

- 从计算上说，
  - 生成一对加密密钥和解密密钥是容易的
  - 已知加密密钥，从明文计算出密文是容易的
  - 已知解密密钥，从密文计算出明文是容易的
  - 从加密密钥推出解密密钥是不可能的
  - 从加密密钥和密文计算出原始明文是不可能的

RSA算法：生成密钥

- 选择两个大素数 p 和 q （典型值为大于 10^{100}）
- 计算 $n=p\times q$ 和 $z=(p-1)\times(q-1)$
- 选择一个与 z 互质的数，令其为 d
- 找到一个 e 使满足 $e\times d=1 \pmod z$
- 公开密钥为 (e, n)，私有密钥为 (d, n)

RSA算法：加密和解密

- 加密方法：
  - 将明文看成是一个比特串，将其划分成一个个数据块M，且有$0 \leq M < n$
  - 对每个数据块M，计算$C = M^e \pmod n$，C即为M的密文
- 解密方法：
  - 对每个密文块C，计算$M = C^d \pmod n$，M即为要求的明文

RSA算法举例

- 密钥计算：
  - 取p＝3，q＝11
  - 则有n＝33，z＝20
  - 7和20没有公因子，可取d＝7
  - 解方程$7\times e=1 \pmod{20}$，得到e＝3
  - 公钥为（3, 33），私钥为（7, 33）
- 加密：
  - 若明文M＝4，则密文$C=M^e \pmod{n}=4^3 \pmod{33}=31$
- 解密：
  - 计算$M=C^d \pmod{n}=31^7 \pmod{33}=4$，恢复出原文

RSA: 另一个重要的特性

这个特性在数字签名中将会很有用:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{先用公钥 再用私钥}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{先用私钥 再用公钥}}$$

结果相同!

RSA的特点

- 优点：
  - 安全性好：RSA的安全性建立在难以对大数提取因子的基础上，这是目前数学家尚未解决的难题
  - 使用方便：免除了传递密钥的麻烦
- 缺点：
  - 计算开销大，速度慢
- RSA的应用：
  - RSA一般用来加密少量数据，如用于鉴别、数字签名或发送一次性会话密钥等

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and Digital Signature

8.4 End point authentication

8.5 Securing e-mail

8.6 Securing TCP connections: SSL

8.7 Network layer security: IPsec

8.8 Securing wireless LANs

8.9 Operational security: firewalls and IDS

报文完整性（报文鉴别）

- 报文完整性（又称报文鉴别）：
  - 用于验证一个报文是否可信的技术
- 一个报文是可信的，如果它来自声称的源并且没有被修改
- 报文鉴别涉及两个方面：
  - 来源鉴别：报文是否来自声称的源
  - 内容完整性检查：报文是否被修改过

朴素的想法：对整个报文加密

- 如果发送方和接收方有一个共享的密钥，可以通过加密报文来提供报文鉴别：
  - 发送方用共享的密钥加密整个报文，发送给接收方
  - 如果接收方能够正确解密收到的报文，则报文必是可信的
- 这种方法的缺点：
  - 混淆了机密性和报文鉴别两个概念，有时我们只想知道报文是否可信，而报文本身并不需要保密
  - 加密整个报文会带来不必要的计算开销

将报文鉴别与数据机密性分开

- 设想：
  - 发送者用明文发送报文，并在报文后附上一个标签，允许接收者利用这个标签来鉴别报文的真伪
- 用于鉴别报文的标签必须满足两个条件：
  - 能够验证报文内容的完整性（是否被修改）
  - 不能被伪造
- 问题：
  - 如何验证报文内容的完整性？
  - 如何保证鉴别报文的标签不被伪造?

内容完整性和报文摘要

- 报文摘要（数字指纹）：
  - 将一个散列函数作用到一个任意长的报文m上，生成一个固定长度的散列值H(m)，这个散列值称为该报文的报文摘要（message digest），也称数字指纹。
- 使用报文摘要验证报文内容的完整性：
  - 发送者对发送的报文计算一个报文摘要，作为标签和报文一起发给接收者
  - 接收者对收到的报文也计算一个报文摘要，和收到的标签（发送方计算的报文摘要）进行比较

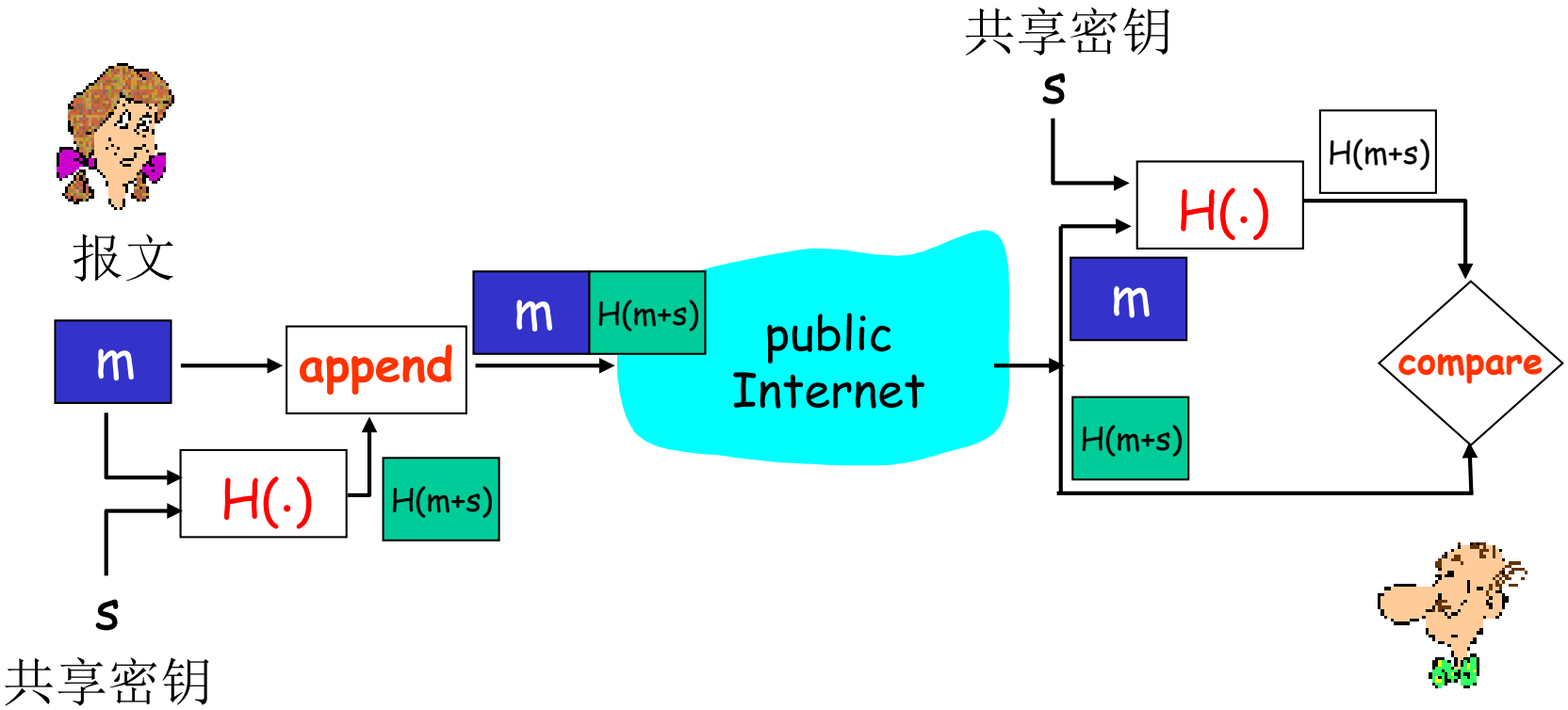
如何保证报文摘要不被伪造？

- 基于加密算法的报文鉴别（方法一）：
  - 发送方计算报文摘要，用与接收方共享的密钥加密报文摘要，形成报文鉴别标签（也称报文鉴别码）
  - 接收方用共享的密钥解密报文鉴别码，得到发送方计算的报文摘要，与自己计算的报文摘要进行比较
- 缺点：
  - 需要使用加密算法

基于哈希运算的报文鉴别

- 为什么要开发一个不需要加密算法的报文鉴别技术？
  - 加密软件通常运行得很慢，即使只加密少量的数据
  - 加密硬件的代价是不能忽略的
  - 加密算法可能受专利保护（如RSA），因而使用代价很高
  - 加密算法可能受到出口控制（如DES），因此有些组织可能无法得到加密算法
- 基于哈希运算的报文鉴别（方法二）：
  - 使用散列函数计算报文摘要时需要包含一个密钥，但它并不用来做加密运算
  - 发送方用双方共享的一个秘密密钥K_S添加到报文m之前，然后计算报文摘要$H(K_S \| m)$形成报文鉴别码

基于哈希运算的报文鉴别码



密码散列函数应满足的特性

- 不同于普通的散列函数，信息安全领域使用的散列函数H（称密码散列函数）必须具有以下一些特性
- 对于任意给定的数据块x，H(x)很容易计算
- 对于任意给定的值h，要找到一个 x 满足H(x)=h，在计算上是不可能的（单向性）：
  - 该特性对于基于哈希运算的报文鉴别很重要
  - 如果根据 $H(K_S||m)=h$ 可以找到一个 x，使得H(x)=h，那么根据 x 和 m 可以推出K_S

密码散列函数应满足的特性（续）

- 对于任意给定的数据块x，要找到一个 y≠x 并满足H(y)=H(x)，在计算上是不可能的：
  - 该特性对于使用加密算法的报文鉴别很重要
  - 如果能找到一个不同于x的数据块y，使得H(y)=H(x)，那么就可以用y替换x而不被接收方察觉
- 要找到一对（x, y）满足H(y) = H(x)，在计算上是不可能的。（抵抗生日攻击）
- 满足前三个特性的散列函数称为弱散列函数，满足所有四个特性的散列函数称为强散列函数。

密码散列函数标准

- 目前使用最多的两种密码散列函数：
  - MD5 [RFC 1321]：散列码长度为128比特
  - SHA-1：美国联邦政府的标准，散列码长度为160比特
- 目前获得最多支持的报文鉴别方案为HMAC，可与MD5和SHA-1一起使用

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and Digital Signature

8.4 End point authentication

8.5 Securing e-mail

8.6 Securing TCP connections: SSL

8.7 Network layer security: IPsec

8.8 Securing wireless LANs

8.9 Operational security: firewalls and IDS

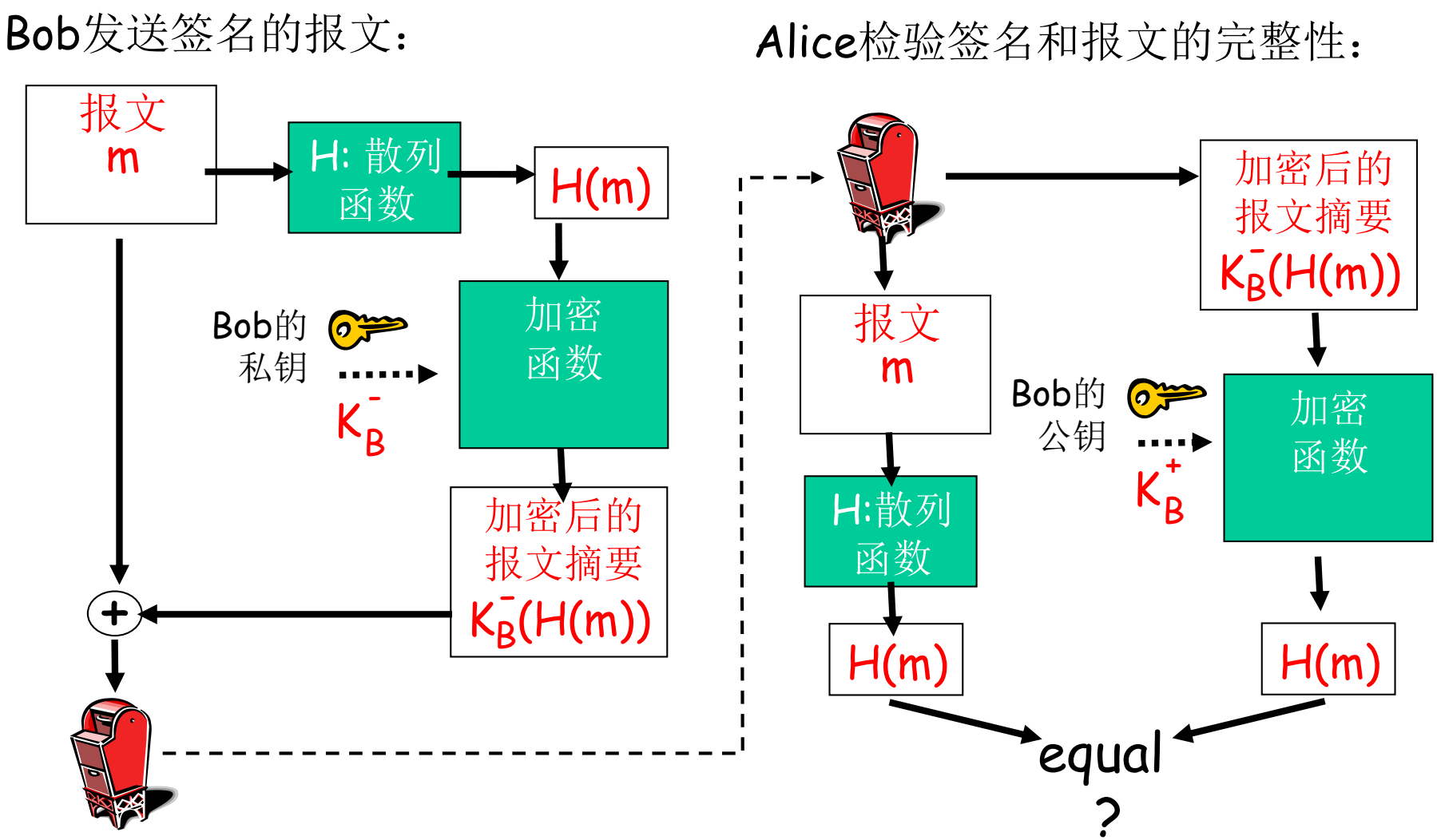
数字签名

- 一个可以替代手写签名的数字签名必须满足以下三个条件：
  - 接收方通过文档中的数字签名能够鉴别发送方的身份（起源鉴别）
  - 发送方过后不能否认发送过签名的文档（防抵赖）
  - 接收方不可能伪造被签名文档的内容

数字签名：用私钥加密报文摘要

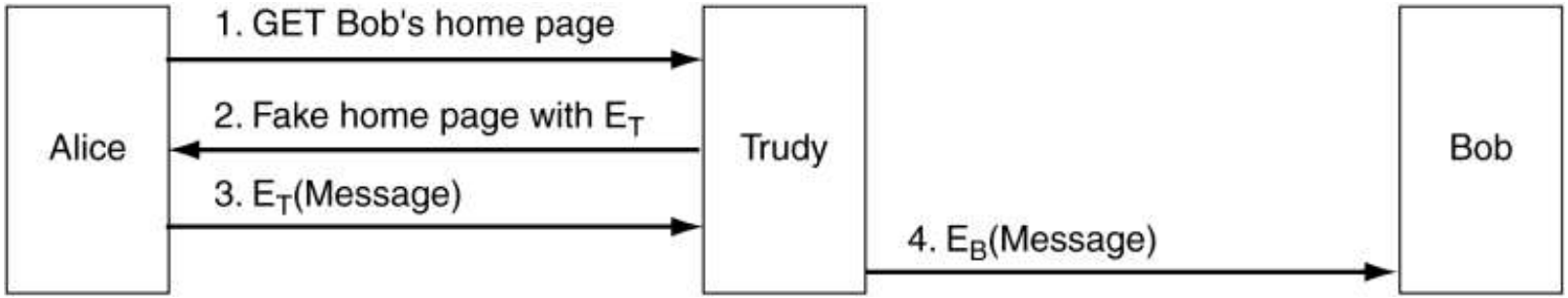
- 发送方先计算报文摘要，然后用自己的私钥加密报文摘要形成数字签名，数字签名附加在报文后面一起发送。
- 接收方拷贝一份数字签名，妥善保存，以备将来需要时使用
- 接收方用发送方的公钥得到原始的报文摘要，对收到的报文计算摘要，如果两者相符，表明报文是真实的。

Digital signature = signed MAC



如何可靠地获取公钥？

- 考虑下面的例子：
  - Bob将公钥E_B发布在自己的主页上
  - Alice获取Bob主页的请求被Trudy截获，Trudy将假冒的Bob主页发送给Alice，主页中的公钥是Trudy的公钥E_T
  - Alice使用Trudy的公钥加密会话密钥，发送给Bob
  - Trudy截获会话密钥，用Bob的公钥加密后再发送给Bob
  - Alice和Bob之间通信的报文都被Trudy破译



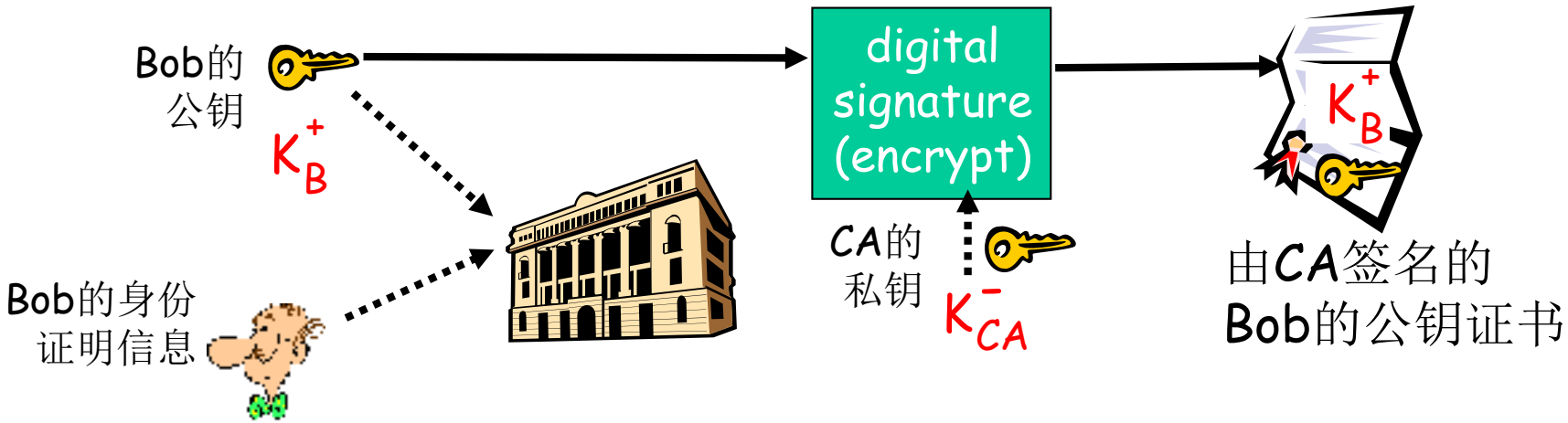
- 问题：
  - 当Alice从公开的途径得到Bob的公钥后，Alice如何确认她得到的就是Bob的公钥，而不是其他人的公钥？

公钥证书

- 为使公钥密码体系有实际应用，每个实体必须能够确认它得到的公钥确实来自声称的实体。
- 解决方案是引入证书机制：
  - 使用证书（certificate）来证明某个主体（principal）拥有某个公钥
  - 证书由一个可信任的第三方机构颁发，该机构称为认证权威CA（certification authority）
  - 证书包含主体的公钥和CA的签名，任何人无法伪造或篡改证书的内容
  - 当一个主体获得其公钥证书后，可将证书放在任何一个可公开访问的地方

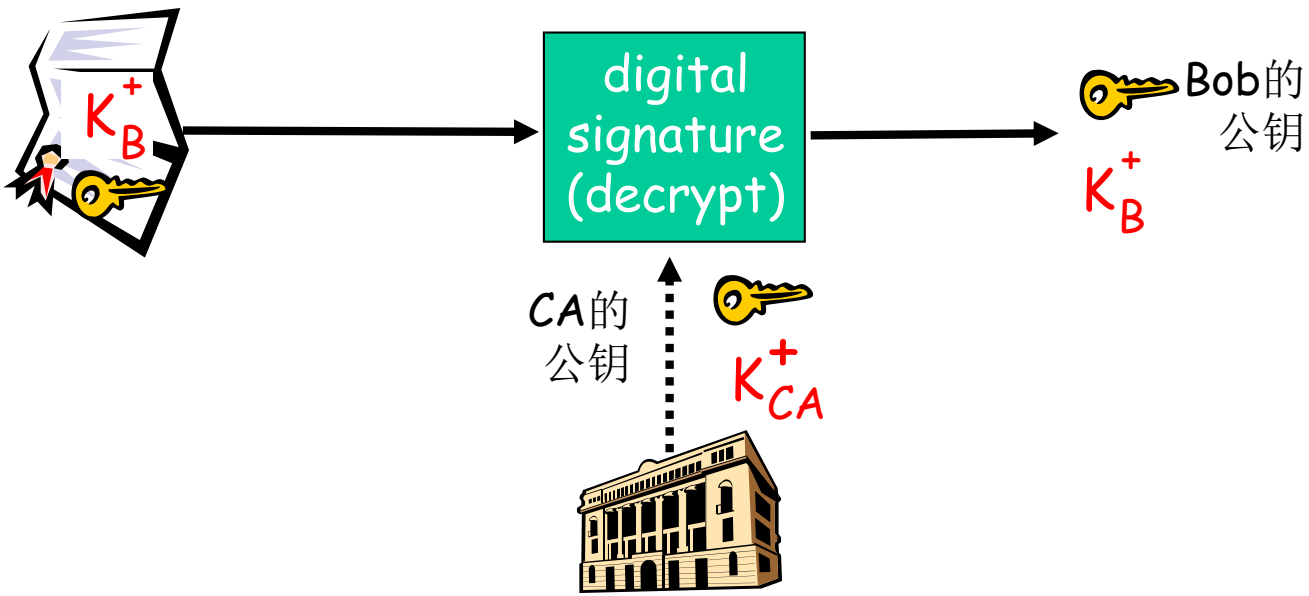
证书的获取

- Bob向CA注册其公钥：
  - Bob向CA提供身份证明
  - CA验证了Bob的身份后创建证书，绑定Bob及其公钥
  - 证书包含Bob的公钥，并有CA的签名



证书的验证

- 当Alice需要Bob的公钥时:
  - 获取Bob的证书
  - 使用CA的公钥验证Bob的证书，得到Bob的公钥



X.509证书

- 目前最常用的证书标准是X.509
- X.509建立在公钥算法和数字签名的基础上：
  - CA对证书内容先进行SHA-1散列，然后用CA的私钥对报文摘要加密，形成数字签名。
- 为验证公钥证书的真实性：
  - 验证方用CA的公钥解开证书的签名，得到证书内容的报文摘要
  - 对收到的证书内容计算报文摘要，并与解密得到的报文摘要进行比较，两者相同表明这是合法的公钥证书

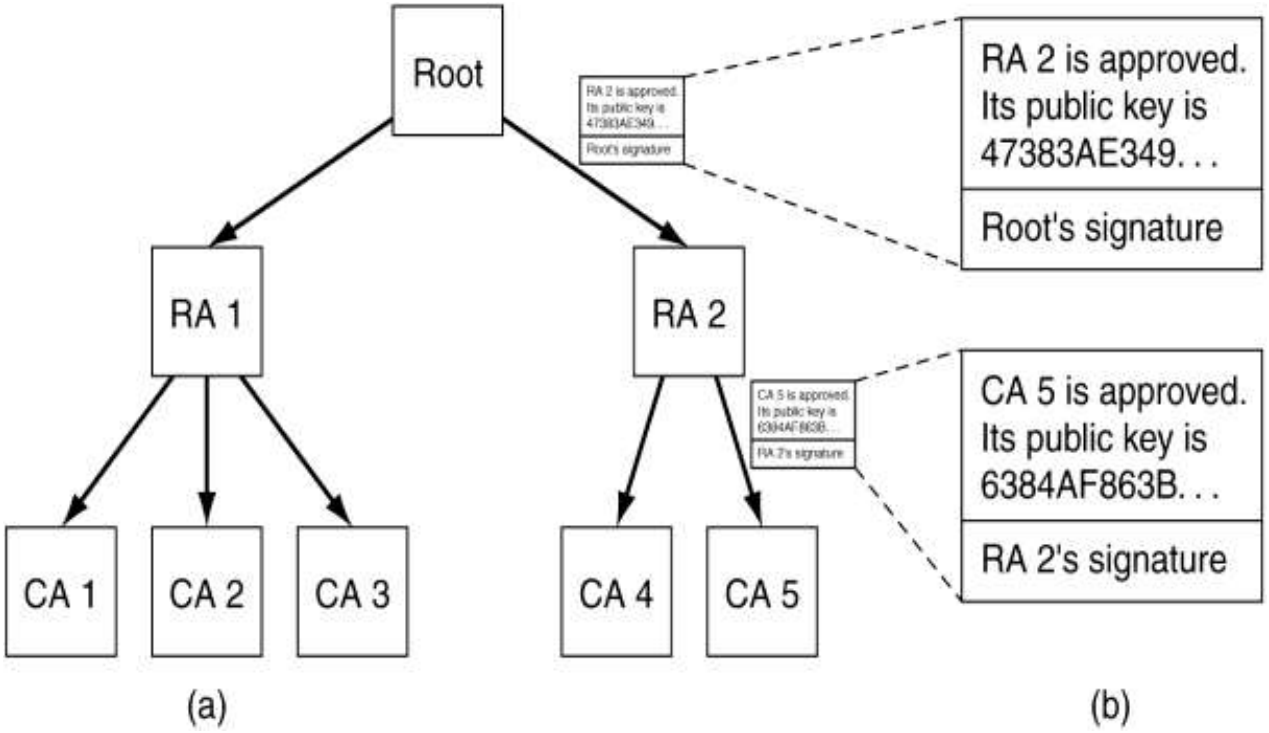
X.509证书格式

字段	含义
Version	X.509版本号
Serial number	证书序列号，与CA名字一起唯一标识一个证书
Signature algorithm	签名该证书使用的算法及相关参数
Issuer	签发该证书的CA的名字
Valid period	证书有效期的起止时间
Subject name	主体名字，证书被颁给的对象
Public key	与主体绑定的公钥、使用该公钥的算法名字及相关参数
Issuer ID	唯一标识CA的一个可选的ID
Subject ID	唯一标识主体的一个可选的ID
Extensions	一个或多个扩展字段（X.509v3才有）
Signature	签发该证书使用的算法、相关参数及证书签名。证书签名覆盖证书的所有内容

如何管理公钥和证书？

- 问题：谁可以运行*CA*？世界上有几个*CA*？
- 使用一个CA签发全世界所有的证书?
  - 流量压力，单点失效
- 由一个组织运行多个CA?
  - 密钥泄露，信任问题
- 分布式公钥基础设施（Public Key Infrastructure，PKI）
  - 提供公钥加密和数字签名服务的系统或平台
  - 包含不同组织运行的CA，每个CA拥有自己的私钥，负责为一部分用户签发证书
  - 用户自己决定使用哪一个CA

CA的一种组织结构



RA：证书注册审批机构（审核信息、发放和管理证书）

CA：证书签发机构（生成证书）

- 系统中的所有实体都有根CA的公钥（通过安全的物理途径获取）
- 所有根CA间可以进行交叉认证
- 许多根CA的公钥被预装在浏览器上，由浏览器厂商认证并嵌入到软件中，随软件一起发布，用于验证服务器

证书的撤销

- 每个证书都有有效期，过期后证书自动失效
- CA也可以显式地撤销证书，这要求CA定期地发布证书撤销列表（Certificate Revocation List，CRL），表中给出已经撤销的证书序列号
- 每个用户在使用一个证书前都要去获取CRL，检查该证书是否在CRL中

证书目录

- 证书存放在哪里？
  - 使用DNS作为证书目录，该方案的标准为DNSSEC
  - 使用专门的目录服务器存放证书，该方案的标准为LDAP
- 证书撤销列表通常与证书存放在一起，CA定期地将CRL推进目录服务器，由目录服务器负责将CRL中列出的证书清除掉

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and Digital Signature

8.4 End point authentication

8.5 Securing e-mail

8.6 Securing TCP connections: SSL

8.7 Network layer security: IPsec

8.8 Securing wireless LANs

8.9 Operational security: firewalls and IDS

鉴别（Authentication）

Goal: Bob希望Alice“证明”她的身份

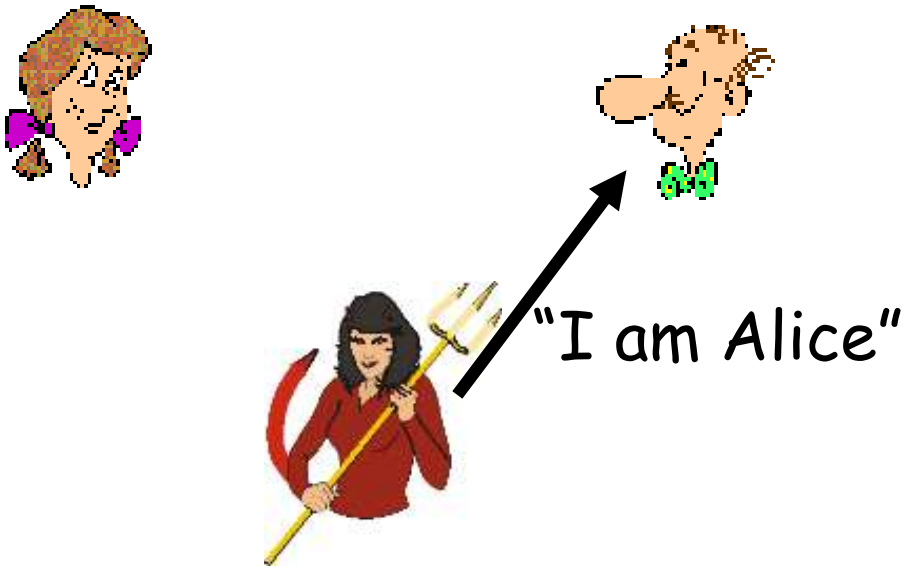
Protocol ap1.0: Alice says “I am Alice”

Failure scenario??

鉴别（Authentication）

Goal: Bob希望Alice"证明"她的身份

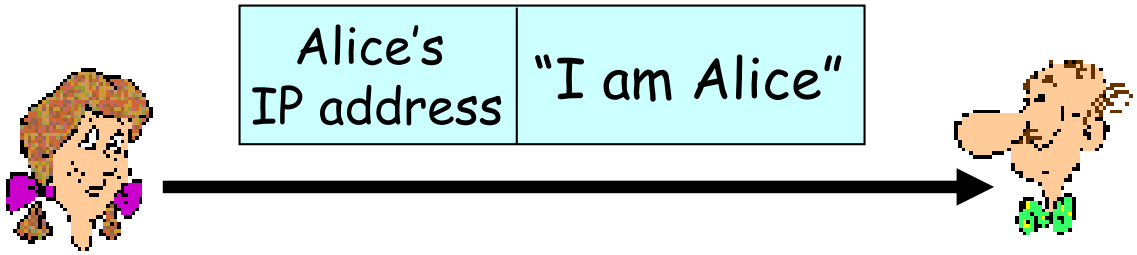
Protocol ap1.0: Alice says "I am Alice"



在网络中，Bob"看"不到Alice, 因此Trudy可以声称她就是Alice

Authentication: another try

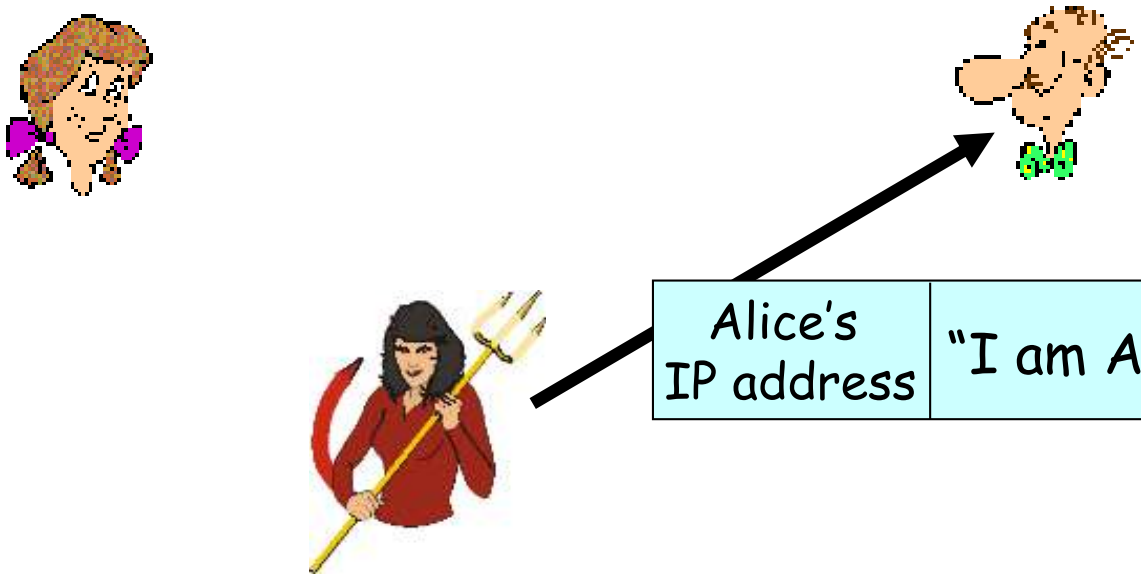
Protocol ap2.0: Alice用自己的IP地址进行证明



Failure scenario??

Authentication: another try

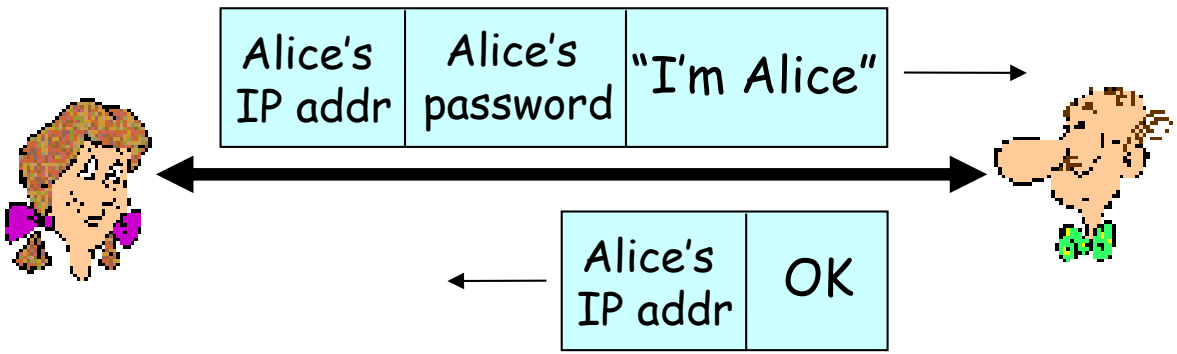
Protocol ap2.0: Alice用自己的IP地址进行证明



Trudy用Alice的IP地址创建一个数据包（IP地址欺骗）

Authentication: another try

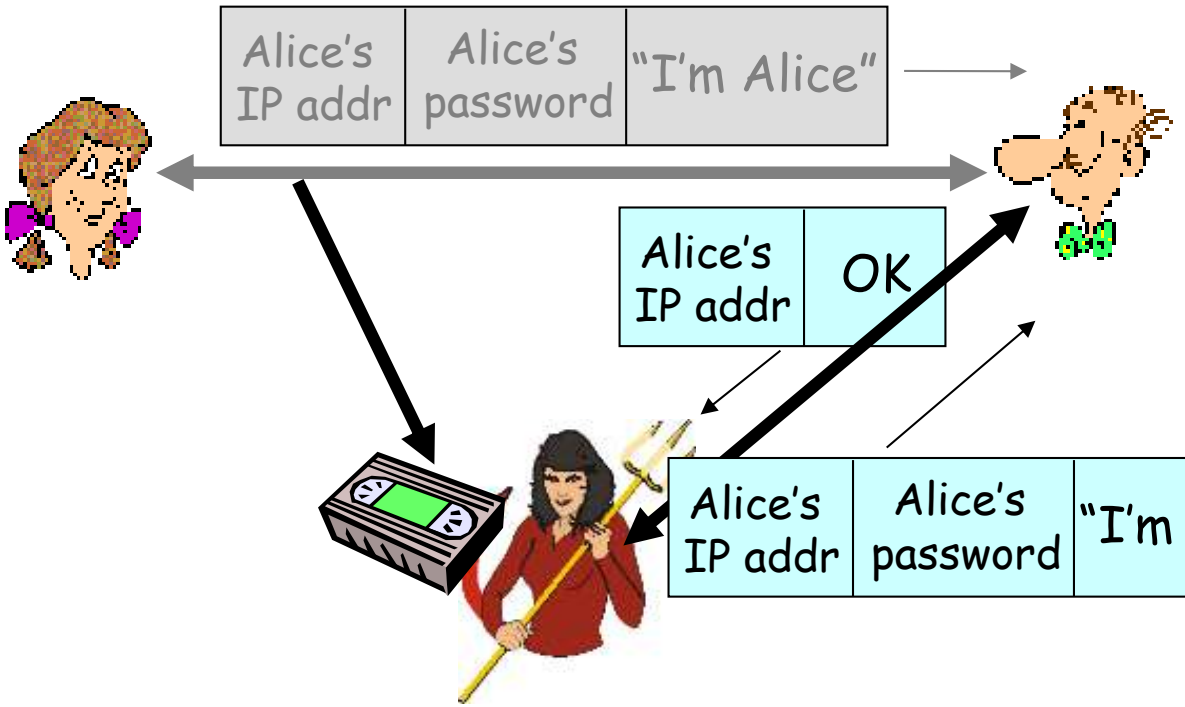
Protocol ap3.0: Alice向Bob发送口令证明自己



Failure scenario??

Authentication: another try

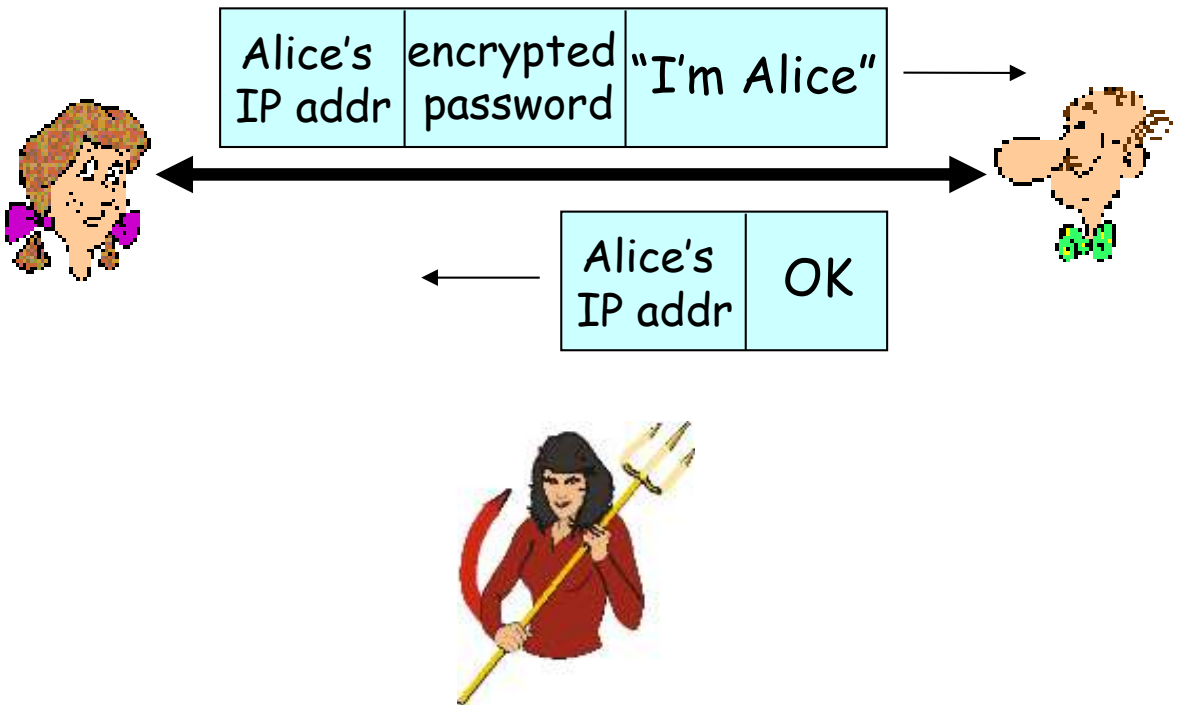
Protocol ap3.0: Alice向Bob发送口令证明自己



Trudy监听到Alice发送的明文口令，过后发送给Bob

Authentication: yet another try

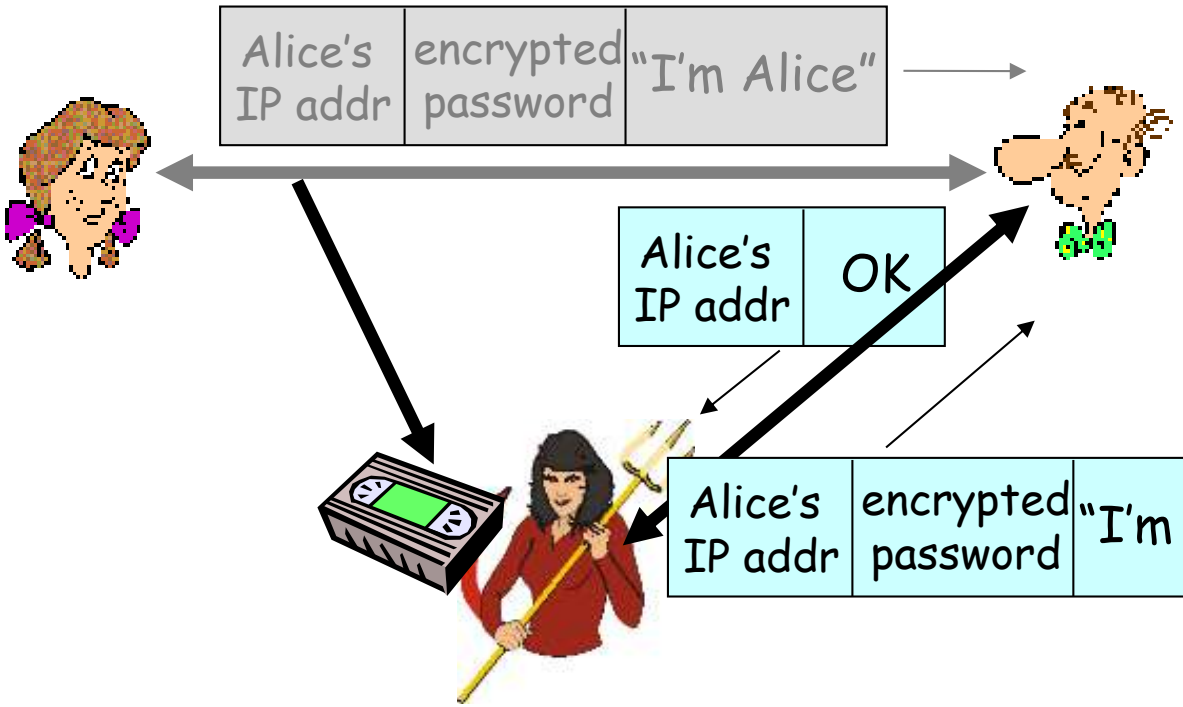
Protocol ap3.1: Alice将口令加密，发送给Bob



Failure scenario??

Authentication: another try

Protocol ap3.1: Alice将口令加密，发送给Bob



Trudy截获数据包，过后发送给Bob

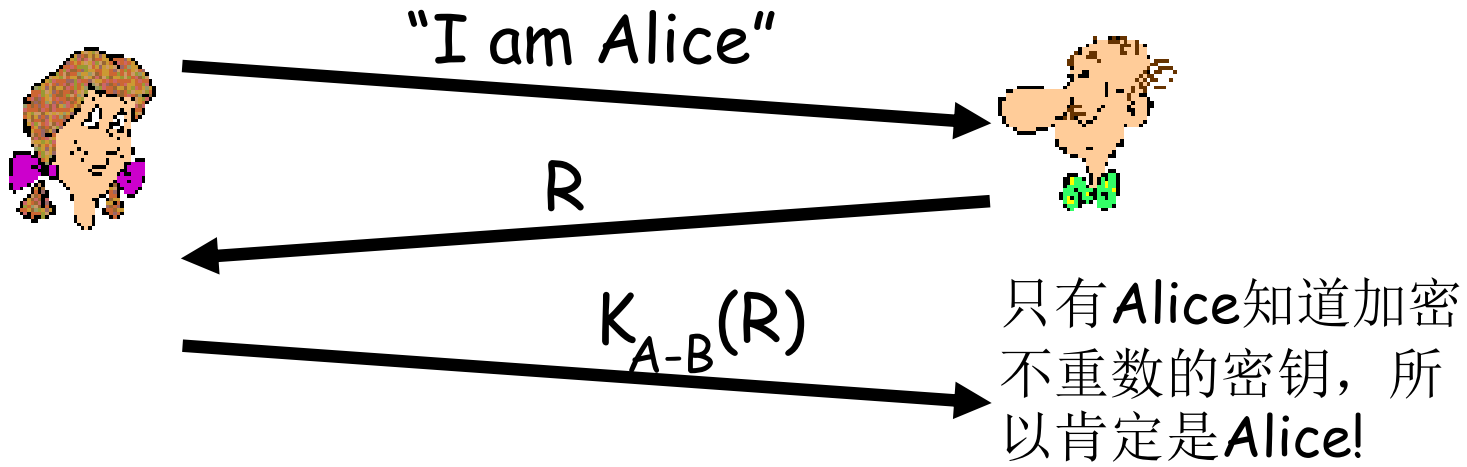
（重放攻击）

Authentication: yet another try

Goal: 避免重放攻击

Nonce: 只用一次的数（不重数）

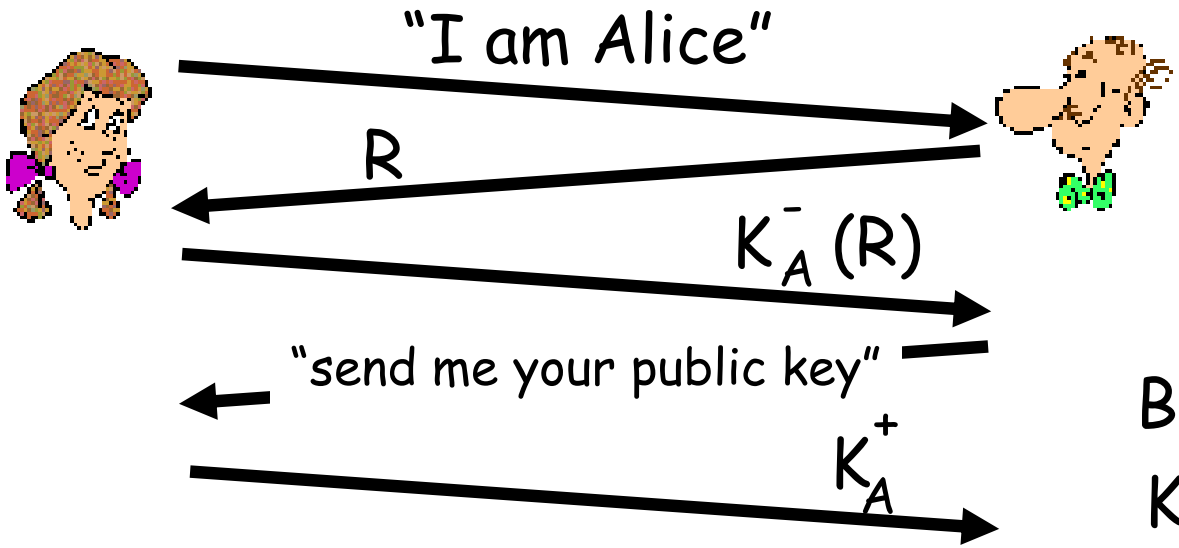
ap4.0: Bob向Alice发送不重数R，Alice用共享密钥加密R，回送给Bob。



缺点：需要一个共享的对称密钥

Authentication: ap5.0

ap5.0: 采用公开密钥算法加密不重数



Bob计算:

$K_A^+(K_A^-(R)) = R$

只有Alice拥有这个私钥，因而一定是Alice!

X.509提供的单向鉴别服务

单向鉴别：涉及发送方到接收方的一次报文传输

$A \rightarrow B: t_A \| r_A \| ID_B \| Data \| K_b^+(K_{a-b}) \| signature_A$

- 说明：
  - **t_A**：时间戳，由报文的产生时间和到期时间组成
  - **r_A**：A随机选择的一个不重数，供接收者检测重放攻击
  - ID_B：B的标识，指示报文的接收者
  - Data：报文中包含的数据信息
  - **K_{a-b}**：若Data需要保密，则K_{a-b}为A加密Data使用的对称密钥
  - **K_b^+**：B的公开密钥，用于加密对称密钥K_{a-b}
  - $signature_A$：A的数字签名，对t_A、r_A、ID_B和Data的明文生成
- 接收方鉴别：
  - B用自己的私钥解出K_{a-b}，用K_{a-b}解密Data，计算前面4个部分的报文摘要
  - B用A的公钥从签名中得到原始的报文摘要，进行比较

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and Digital Signature

8.4 End point authentication

8.5 Securing e-mail

8.6 Securing TCP connections: SSL

8.7 Network layer security: IPsec

8.8 Securing wireless LANs

8.9 Operational security: firewalls and IDS

电子邮件安全

- 电子邮件安全包括真实性（起源、内容）和机密性两个方面
- 目前最流行的两个安全电子邮件协议：
  - PGP：一个开放源码的安全电子邮件软件包，提供对邮件的保密、鉴别和压缩服务。PGP较多地用于个人电子邮件安全。（因特网安全电子邮件的事实标准）
  - S/MIME：基于公钥加密技术对MIME所做的安全扩展。S/MIME较可能作为一种工业标准，被商业组织或一些机构使用

Pretty Good Privacy（PGP）

- PGP提供五种服务：
  - 鉴别，机密性，压缩，兼容电子邮件，分段
- 鉴别：
  - PGP使用基于公开密钥算法的数字签名提供鉴别服务
  - 生成可供鉴别的电子邮件：
    - 发送方创建电子邮件（报文）
    - 用SHA-1计算邮件的报文摘要，然后用发送者的私钥加密报文摘要，形成数字签名
    - 将数字签名附在报文的前面，与报文一起发送：

      Sgn || Data

PGP的机密性服务

- PGP使用对称密钥算法保护邮件的机密性：
  - 发送方将选择的一次性会话密钥用接收方的公钥加密，与报文一起发送给接收方
- 仅使用机密性服务的过程：
  - 发送方（A）生成一个报文和一个随机的128比特数（一次性会话密钥）
  - 先用会话密钥加密报文，再用接收方（B）的公钥加密会话密钥
  - 将加密后的会话密钥放在报文前面，与报文一起发送：

$$K_B^+(K_{A\text{-}B}) \,||\, K_{A\text{-}B}(\text{Data})$$

PGP的压缩服务

- 缺省地，PGP在完成签名之后、在加密报文之前对报文进行压缩，压缩算法采用ZIP：

  $K_B^+(K_{A\text{-}B}) \| K_{A\text{-}B}(Zip(Sgn \| Data))$

- 在加密报文之前进行压缩，一方面可以减少要加密的数据量，另一方面压缩后的消息冗余很少，增加密码分析的困难。

PGP的兼容电子邮件服务

- PGP使用Base64编码将二进制数据流转换成简单ASCII文本，以解决邮件的传输问题。
- PGP可被配置为仅对报文中的某些部分（如签名部分）进行Base64编码转换。

鉴别 + 机密性 + 压缩 + 兼容性

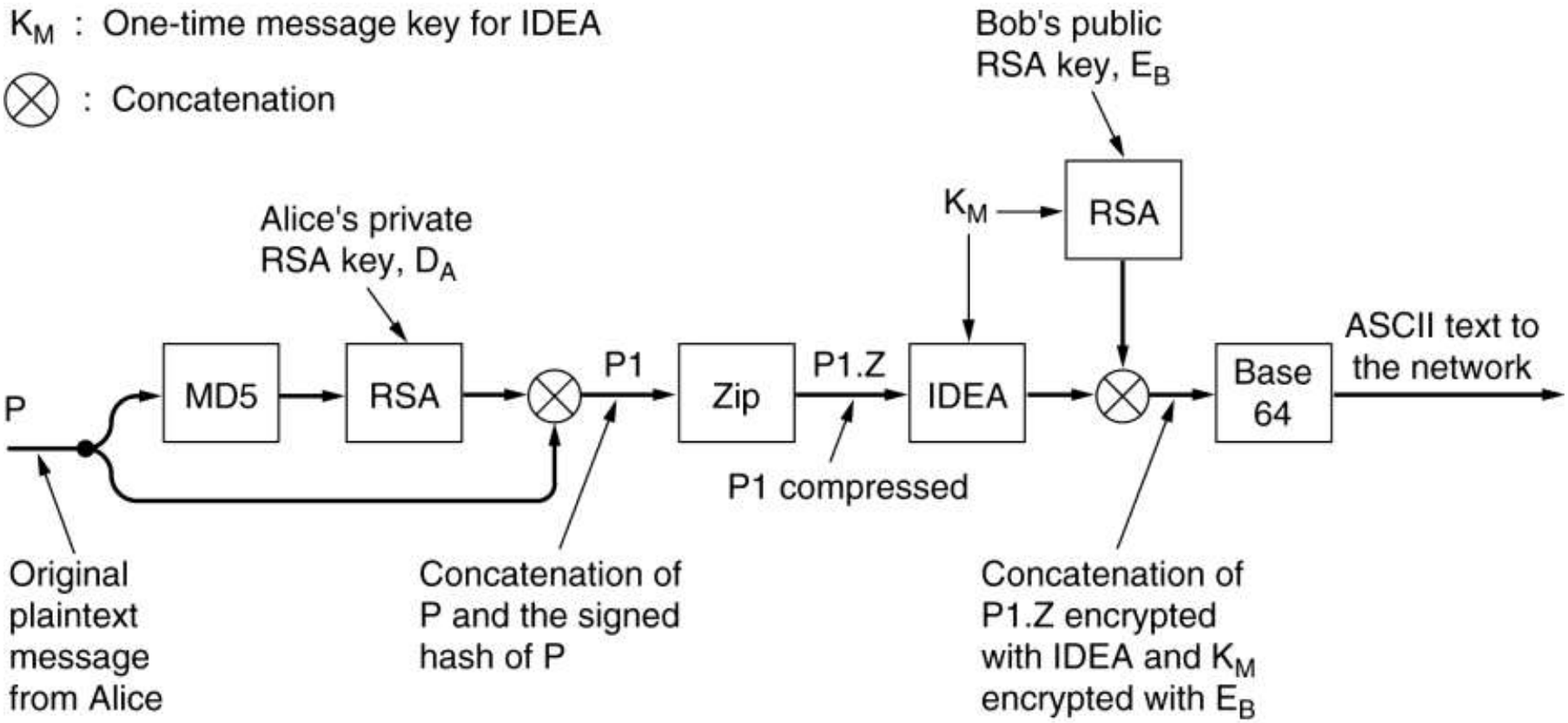
- 同时使用以上四种服务的过程：
  - 发送方先对明文报文计算签名，将签名放在报文前面
  - 签名与明文一起被压缩
  - 用会话密钥对压缩后的数据块进行加密
  - 用接收方的公钥加密会话密钥，放在报文的前面
  - 将整个数据块转换成Base64编码格式：

$$Encode_{Base64}(K_B^{+}(K_{A-B}) \| K_{A-B}(Zip(Sgn \| Data)))$$

PGP的邮件分段服务

- 许多电子邮件系统能够接收的最大报文长度不超过50,000字节
- PGP在完成对报文的全部处理后，自动将超过长度的报文分成小块传输，会话密钥和签名只在第一个片段中出现
- 接收端去掉每个片段的头部，然后将所有的片段重新组装成一个数据块

使用PGP发送一个邮件

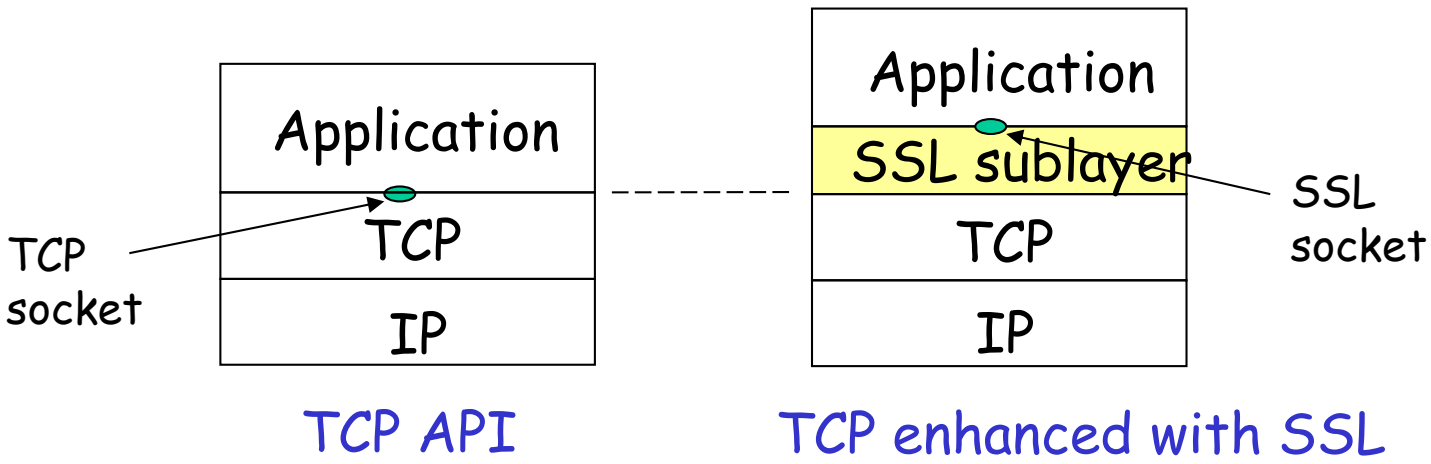


Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and Digital Signature
8.4 End point authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: IPsec
8.8 Securing wireless LANs
8.9 Operational security: firewalls and IDS

Secure sockets layer (SSL)

- SSL向基于TCP的网络应用提供安全的传输层服务：
  - 如支持Web浏览器和服务器之间的安全通信（https）
- 安全服务：
  - 机密性、数据完整性、服务器鉴别、客户鉴别（可选）



SSL/TLS（Transport Layer Security）

- SSL建立在TCP之上，依靠TCP提供可靠的端到端连接
- SSL是涉及到两个层次的一组协议：
  - SSL记录协议：为各种高层协议（如HTTP）提供基本的安全服务（机密性、完整性）
  - 其它三个高层协议：用于SSL交换管理

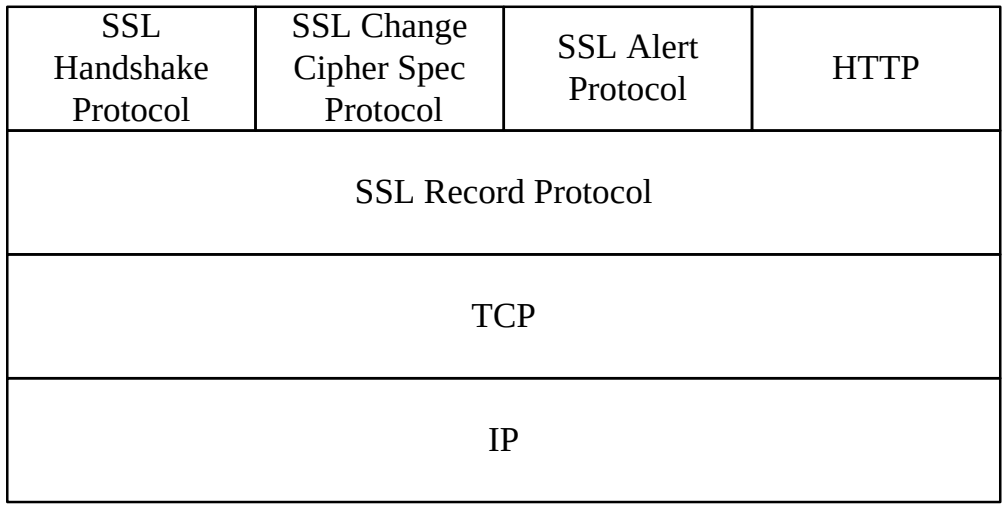


图8-16 SSL协议栈

SSL握手协议

- 允许服务器和客户之间相互鉴别，并协商加密算法、MAC算法及密钥等
- 握手协议由客户和服务器之间的一系列报文交换组成：

能力协商

- 浏览器向服务器发送建立SSL会话的请求报文，说明可支持的SSL协议最高版本、支持的加密算法（按优先级从高到低排列）和压缩方法等，以及浏览器选择的一个随机数Rc
- 服务器从浏览器给出的选择中确定合适的SSL版本号、加密算法和压缩方法，与服务器选择的一个随机数Rs一起发送给浏览器

服务器鉴别

- 服务器向浏览器发送它的公钥证书（和必要的证书链）以及其它信息。
- 浏览器检查签发证书的CA是否在浏览器的可信CA列表中，若不在向用户警告该问题； 如果在则使用该CA的公钥验证证书，得到服务器的公钥。

SSL握手协议（续）

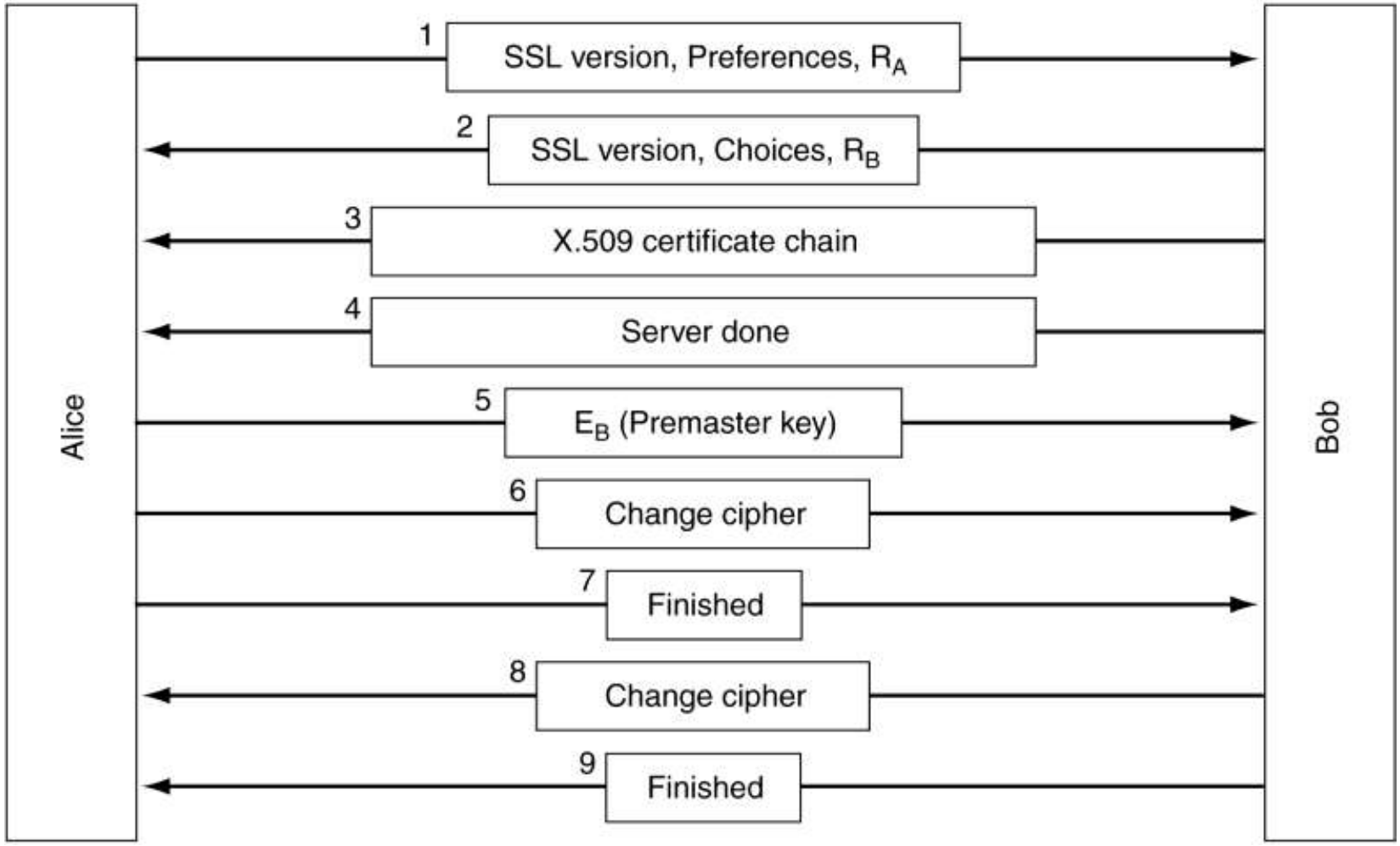
- 如果客户也需要被鉴别（收到服务器的证书请求），则浏览器向服务器发送它的公钥证书

生成密钥

- 浏览器生成一个48字节的随机数，称预密钥，用服务器的公钥加密后发送给服务器
- 客户和服务器各自从预密钥、Rc和Rs中计算加密数据需要的会话密钥，以及计算MAC需要的密钥

- 浏览器向服务器发送所有握手报文（级联）的MAC
- 服务器向浏览器发送所有握手报文（级联）的MAC

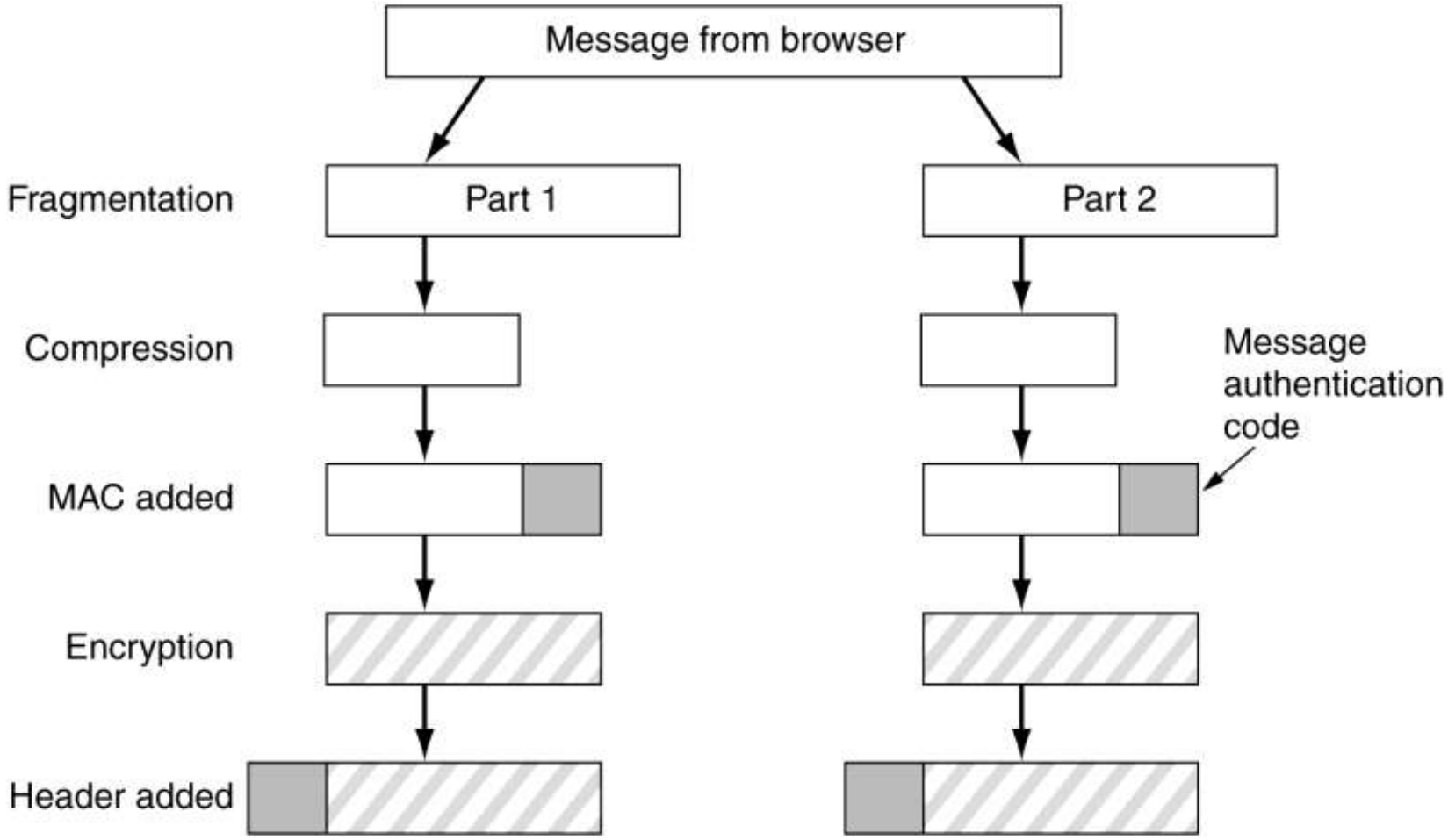
SSL会话建立示例


Alice
Bob
1
SSL version, Preferences, R_A
2
SSL version, Choices, R_B
3
X.509 certificate chain
4
Server done
5
E_B (Premaster key)
6
Change cipher
7
Finished
8
Change cipher
9
Finished

SSL记录协议

- SSL记录协议为SSL连接提供两种服务：
  - 机密性：通过加密SSL载荷实现
  - 完整性：通过报文鉴别码保护
- SSL记录协议的操作过程：
  - 从上层接收一个要传输的应用报文，将报文划分成长度不超过2^{14}字节的数据块
  - （可选）对数据块进行压缩
  - 对数据块生成基于哈希运算的报文鉴别码
  - 使用对称密钥算法对（压缩的）数据块及报文鉴别码进行加密，加密算法可以是DES、3DES、IDEA、RC等
  - 在处理完的数据块前加上SSL头，包括内容类型、SSL版本号、压缩数据块的长度等

使用SSL传输数据


Message from browser
Fragmentation
Part 1
Part 2
Compression
Message
authentication
code
MAC added
Encryption
Header added

Chapter 8 roadmap

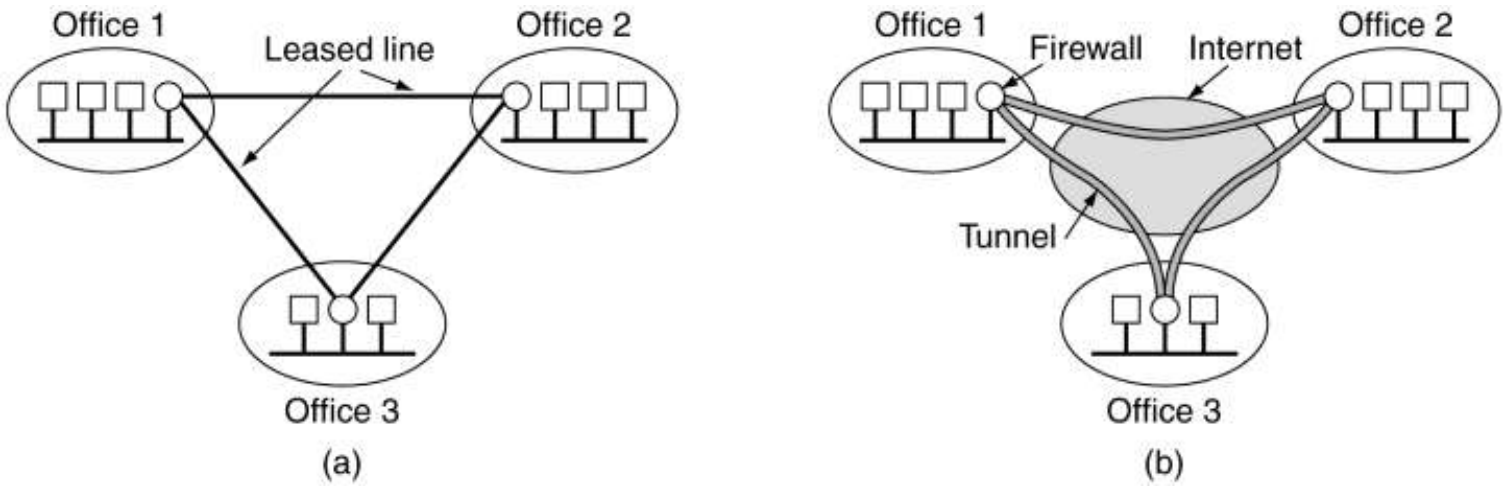
8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and Digital Signature
8.4 End point authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: IPsec
8.8 Securing wireless LANs
8.9 Operational security: firewalls and IDS

IP安全协议（IPSec）

- IPv4在设计时没有考虑安全性：
  - 缺少对通信双方身份的鉴别，容易遭受地址欺骗攻击
  - 缺少对网络中数据的完整性和机密性的保护，数据很容易被窃听、修改甚至劫持
- IP Security（IPSec）：
  - IETF以RFC形式公布的一组安全协议集
  - 目标是把安全特征集成到IP层，以便对因特网中的安全业务提供低层的支持。

专用网和虚拟专用网

- 专用网：
  - 通过电信专线将分散在各地的计算机（网络）连接而成的网络
  - 安全性好，但代价高
- 虚拟专用网（Virtual Private Network）：
  - 建立在公用网上的一个覆盖网络（overlay），在逻辑上与其它流量隔离，数据在发送到公用网之前进行加密

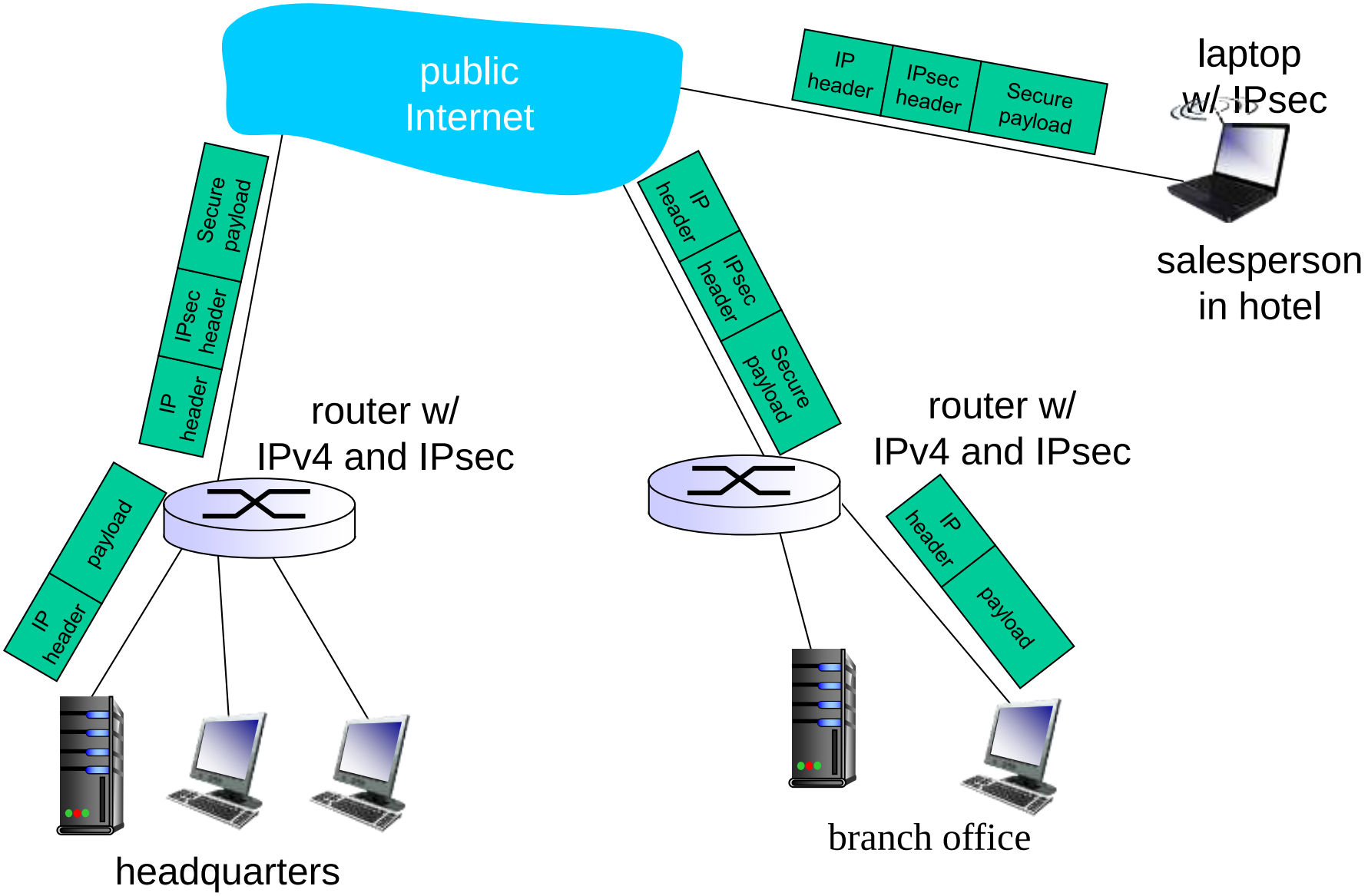


(a) A leased-line private network. (b) A virtual private network.

VPN的实现

- VPN的典型结构：
  - 在每个局域网上设置一个安全网关，在每一对安全网关间创建一条穿过因特网的隧道，在隧道中使用IPSec
- VPN的优点：
  - 可以在一对局域网间提供完整性控制及机密性服务，甚至对流量分析也有相当的抵御能力
  - 对因特网中的路由器及用户软件是透明的，只要系统管理员设置好安全网关就可以了

VPN的实现


public
Internet
IP header
IPsec header
Secure payload
laptop w/ IPsec
salesperson in hotel
router w/ IPv4 and IPsec
router w/ IPv4 and IPsec
IP header
payload
headquarters
branch office

IPSec提供了一个安全体系框架

- IPSec提供了一个用于集成多种安全服务、加密算法及安全控制粒度的安全体系框架
  - IPSec提供的安全服务包括：访问控制、无连接完整性、数据起源认证、抗重放攻击、机密性等
  - IPSec的安全机制独立于算法，因此在选择和改变算法时不会影响其它部分的实现
  - IPSec提供多种安全控制粒度，包括：一条TCP连接上的通信，一对主机间的通信，一对安全网关之间的所有通信
- 用户可以为数据通信选择合适的安全服务、算法、协议和控制粒度

IPSec的组成

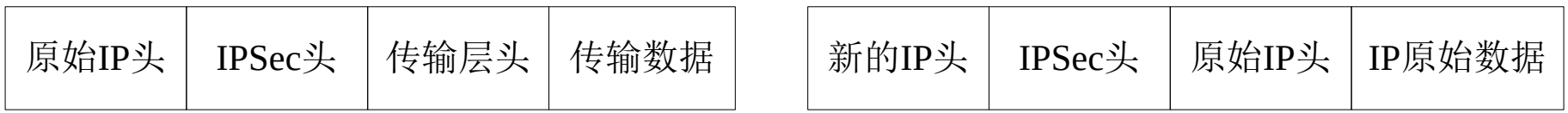
- 从技术上说，IPSec主要包括两个部分：
  - IPSec安全协议：包括AH和ESP两个安全协议，定义了用于安全通信的IP扩展头和字段，以提供机密性、完整性和源鉴别服务。
  - 密钥管理协议：定义了通信实体间进行身份鉴别、协商加密算法以及生成共享会话密钥的方法。
- 将以上两部分绑定在一起的是称为安全关联（SA）的抽象。

Security Association（SA）

- SA是通信对等实体之间对某些要素的协定，如使用的安全协议、协议的操作模式、使用的密码算法、密钥及密钥的生存期等。
- SA是两个通信端点间的一个单工连接，由一个安全参数索引（SPI）唯一标识，如果在两个方向上都需要安全通信，则需要建立两个SA。
- SPI携带在数据包中，由数据包的处理进程用来查找密钥及相关信息。
- SA可以建立在一对主机之间、一台主机与一个安全网关之间、或一对安全网关之间。

IPSec的使用模式

- 传输模式：IPSec头被插入到原始IP头和传输层头之间，路由器根据原始IP头转发数据包。
- 隧道模式：原始数据包被封装在一个新的IP包中，IPSec头被放在新的IP头和原始IP头之间，路由器根据外层IP头的信息转发数据包。隧道的端点（外层IP头中的地址）通常是一个支持IPSec的安全网关。



（a）传输模式　　（b）隧道模式

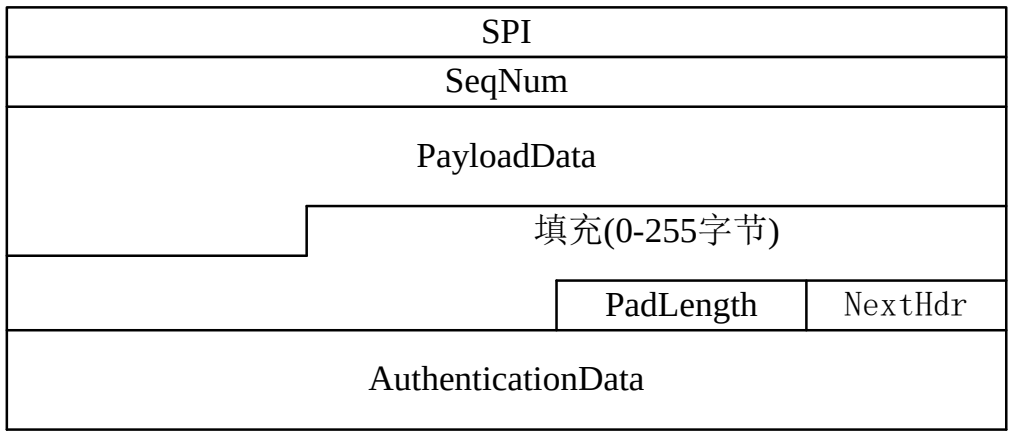
图8-6　IPSec的使用模式

两种模式的比较

- 传输模式比隧道模式占用较少的带宽
- 隧道模式更安全：
  - 隐藏内部网络的细节（原始IP头不可见）
  - 隧道模式可以将一对端点间的通信聚合成一个加密流，从而有效地防止入侵者进行流量分析

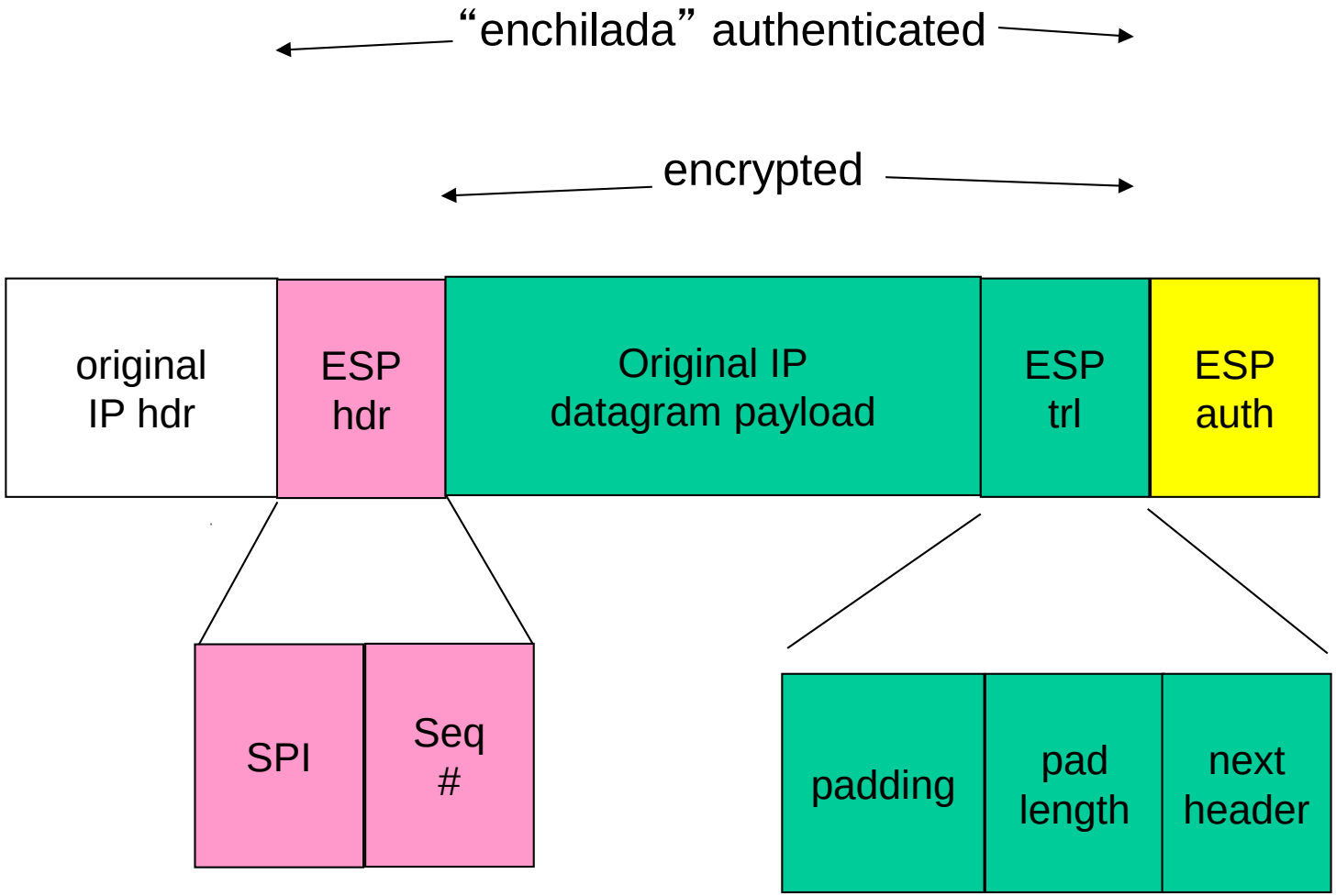
封装安全载荷（Encapsulating Security Payload）

- ESP数据包分为以下几个部分：
  - ESP头：包含SPI和SeqNum
  - 载荷：原始数据包中被加密部分的密文
  - ESP尾：包括填充（需要的话）、填充长度和下一个头，ESP尾也要被加密
  - 鉴别数据：覆盖ESP头、载荷和ESP尾的报文鉴别码

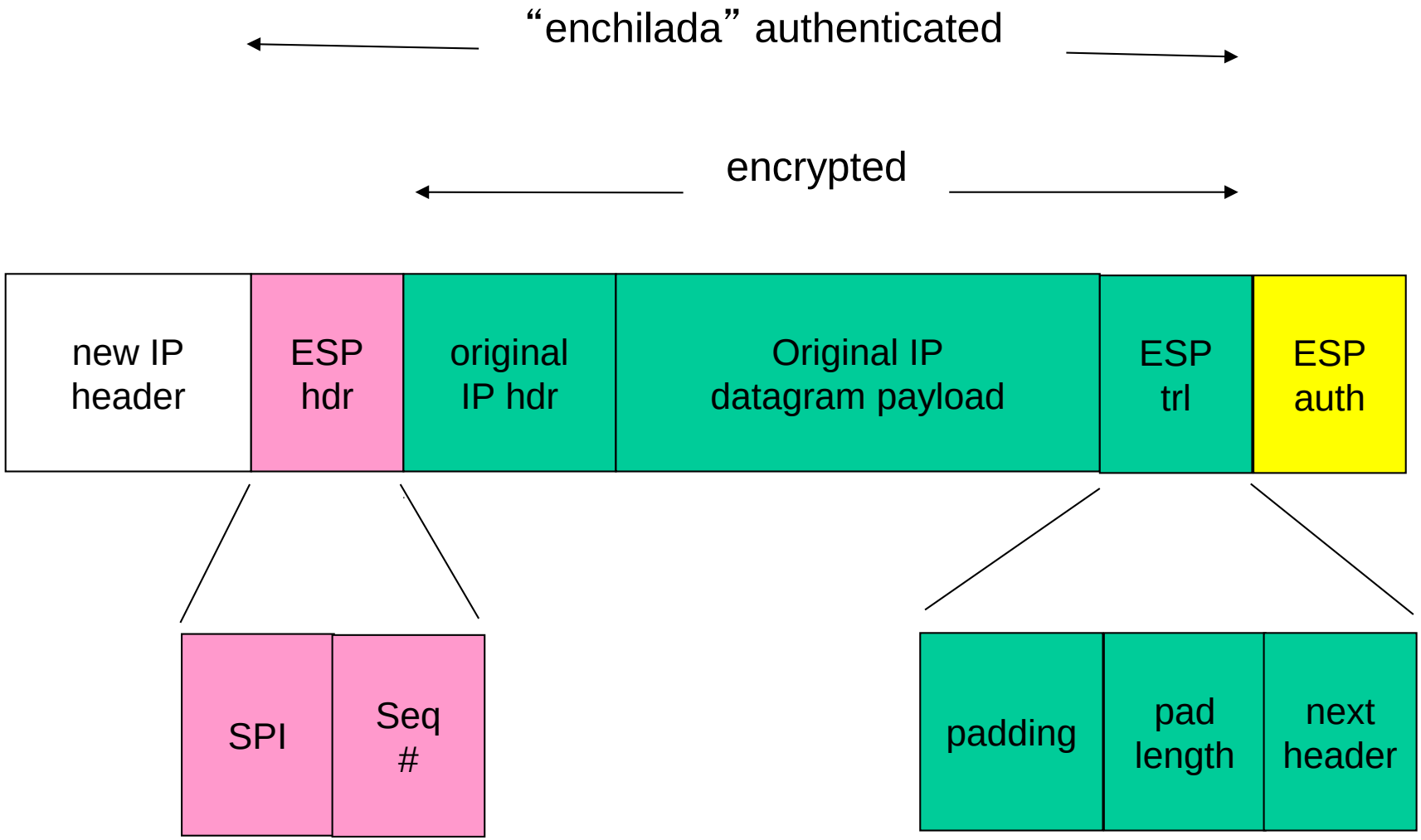


- SPI：32比特的数，和目的IP地址、安全协议结合起来，唯一标识数据报的SA。
- SeqNum：对SA上发送的数据包进行编号，供接收端检测重放攻击。一个SA上的序号不能重用，因此在传输的数据包数量达到2^{32}之前，必须协商一个新的SA和新的密钥。
- Authentication Data：包含报文鉴别码的可变长度域

传输模式下的ESP封装形式



隧道模式下的ESP封装形式



ESP协议提供的安全服务

- ESP协议提供数据机密性、无连接完整性、抗重放攻击、数据起源鉴别和有限的数据流机密性服务：
  - 原始数据包的载荷部分被加密，因而可提供数据机密性
  - HMAC覆盖数据包载荷部分，可提供无连接完整性服务
  - ESP头中有序号，且被HMAC覆盖，可抵抗重放攻击
  - ESP隧道模式中，原始IP头也被HMAC覆盖，因而ESP隧道模式可提供数据起源鉴别
  - ESP隧道模式中，原始IP头也被加密，路由器只能看到外层IP头，因而ESP隧道模式可提供数据流机密性服务

Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and Digital Signature
8.4 End point authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: IPsec
8.8 Securing wireless LANs
8.9 Operational security: firewalls and IDS

IEEE 802.11 security

- 802.11 WEP（Wired Equivalent Privacy）：
  - 最初的802.11规范使用的安全协议
  - 在主机和基站之间提供较弱的加密及鉴别服务
  - 没有密钥分发机制
- 802.11i
  - 具有更强安全机制的802.11版本
  - 提供较强的加密机制及鉴别机制
  - 提供密钥分发机制

WEP：主机鉴别

- 主机鉴别过程：
  - 无线主机向AP请求鉴别
  - AP向主机发送一个128比特的不重数
  - 主机使用与AP共享的对称密钥加密不重数，发送给AP
  - AP解密不重数，若与AP发送给主机的不重数相同，完成主机鉴别
- 利用主机与基站共享密钥这个事实鉴别主机，主机与AP共享一个半永久的对称密钥

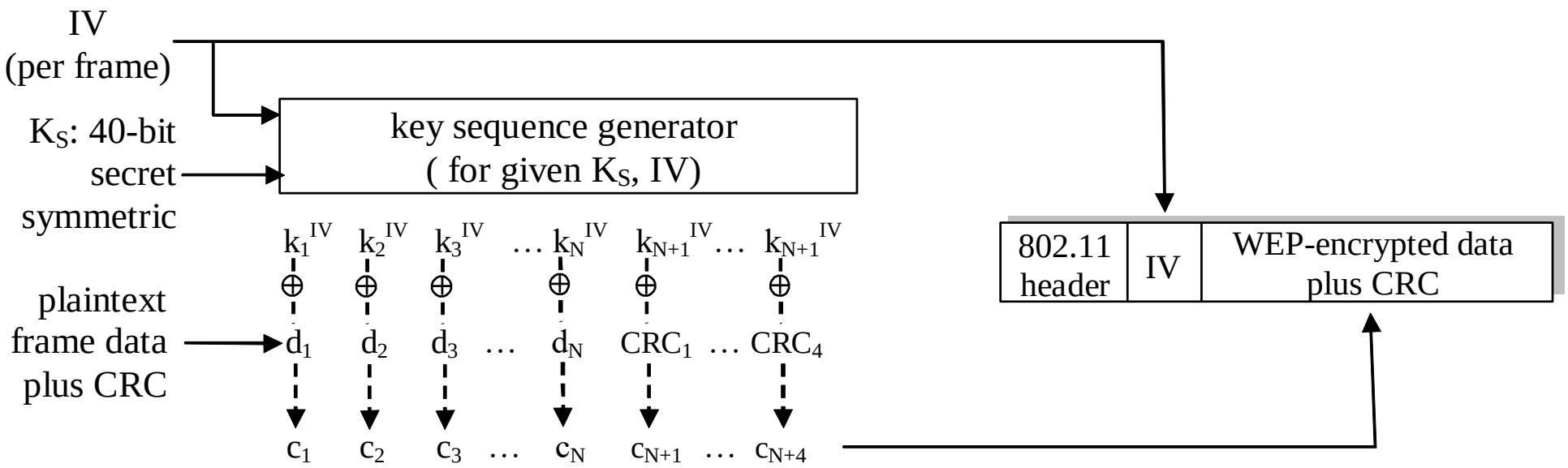
WEP：数据加密

- 主机与AP共享一个40比特的对称密钥K_S（半永久）
- 对于每个帧，发送方生成一个24比特的初始向量IV，添加到K_S后面，形成一个64比特的密钥 (K_S, IV)
- (K_S, IV) 用于生成一个密钥流 $\{k_i^{IV} | i=1,2,...\}$
- 第 i 个密钥k_i^{IV}用来加密帧中的第 i 个字节d_i：

$$c_i = d_i \text{ XOR } k_i^{IV}$$

- IV和加密后的字节c_i放在帧中传输
- 接收方使用相同的 (K_S, IV) 生成相同的密钥流，执行解密运算：$d_i = c_i \text{ XOR } k_i^{IV}$

802.11 WEP加密（图示）



发送方WEP加密

802.11 WEP加密的安全漏洞

安全漏洞:

- 每个K_S只有2^{24}个(K_S, IV) 可用：IV会被重复使用
- IV用明文传输：攻击者可以观察到IV的重用

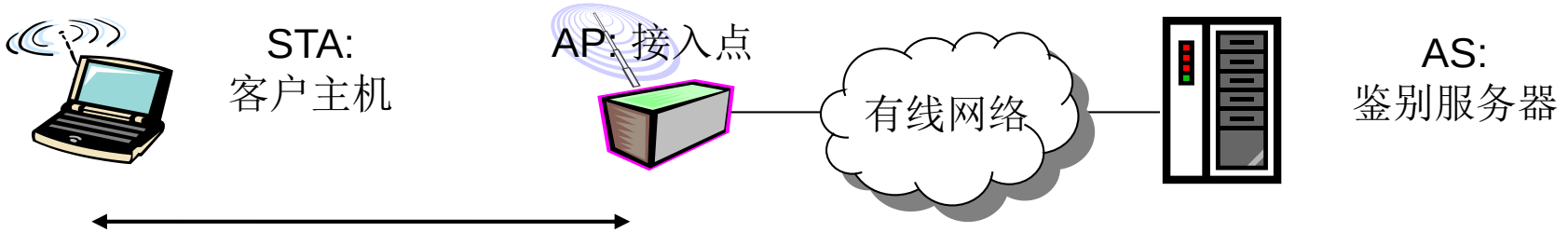
攻击:

- Trudy（可能通过欺骗方式）让Alice加密他选择的明文 (d_1 d_2 d_3 d_4 …)
- Trudy能够获得Alice加密的密文: $c_i = d_i$ XOR k_i^{IV}
- Trudy知道c_i和d_i, 就可以计算出k_i^{IV}： d_i XOR $c_i = k_i^{IV}$
- Trudy得到了加密所用的密钥流 k_1^{IV} k_2^{IV} k_3^{IV} …
- 当过后观察到IV被重用时，Trudy就可以破解密文了！

802.11i: 增强的安全性

- 可以使用各种（较强的）加密算法
- 提供了密钥分发机制
- 使用专门的鉴别服务器（而不是AP）来提供鉴别服务

802.11i的操作



1 安全能力发现

2 STA和AS相互鉴别, 生成主密钥（MK)，AP起一个中继的作用

3 STA导出成对主密钥(PMK)

3 AS导出相同的PMK，发送给AP

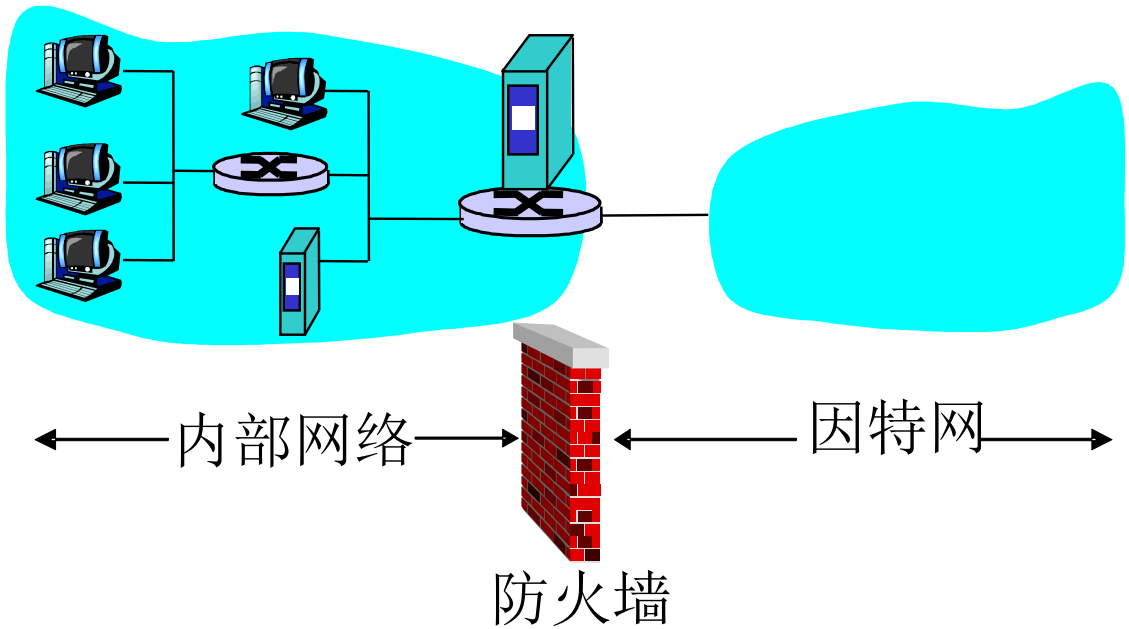
4 STA和AP使用PMK导出用于通信所需的临时密钥

Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity and Digital Signature
8.4 End point authentication
8.5 Securing e-mail
8.6 Securing TCP connections: SSL
8.7 Network layer security: IPsec
8.8 Securing wireless LANs
8.9 Operational security: firewalls and IDS

防火墙

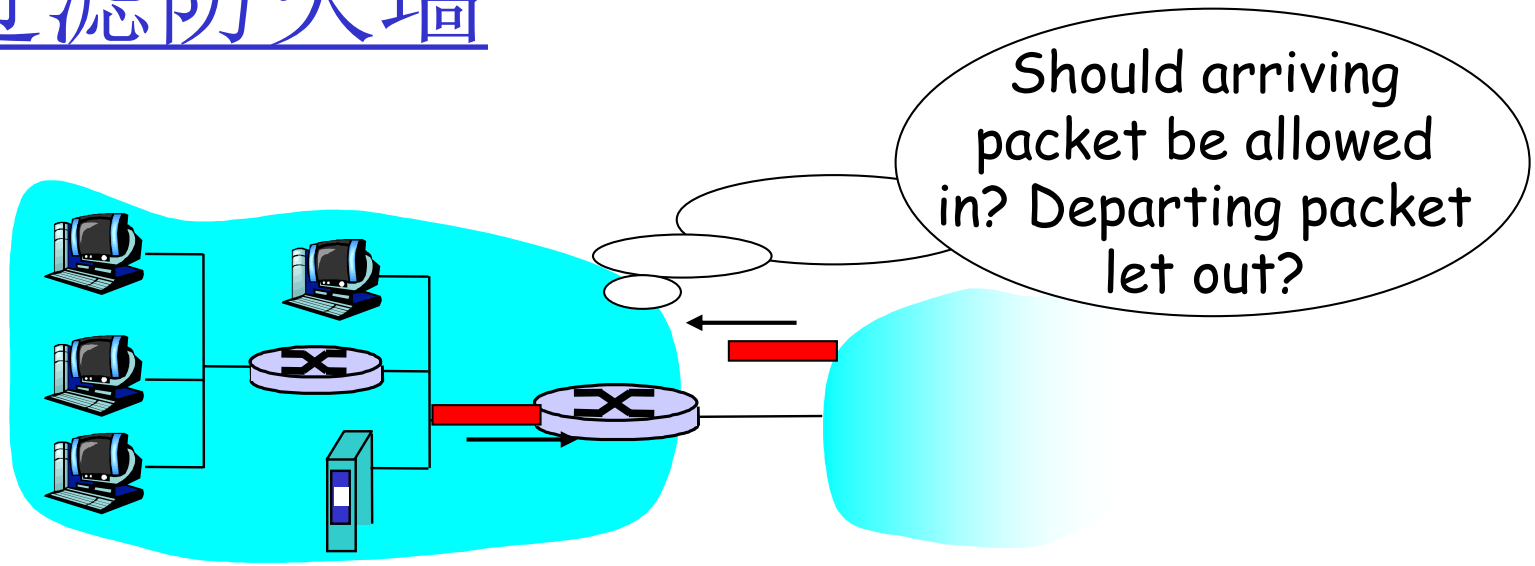
- 在可信的内部网络与不可信的外部网络之间执行访问控制策略的硬件或软件系统
- 目的是保护内部网络免受来自外部网络的攻击。



防火墙的类型

- 包过滤防火墙
- 状态检测防火墙
- 应用网关

包过滤防火墙



- 内部网络通过有包过滤功能的路由器连接到因特网上
- 路由器对数据包进行逐包过滤，基于以下字段决定转发包还是丢弃包:
  - 源IP地址，目的IP地址
  - TCP/UDP源端口号、目的端口号
  - ICMP 报文类型
  - TCP SYN标志和 ACK标志

包过滤策略的例子

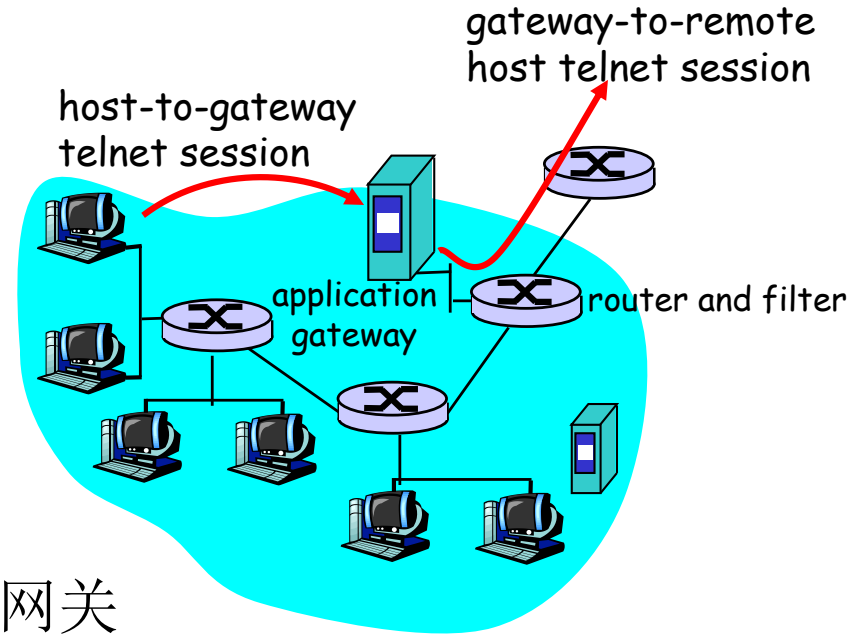
安全策略	过滤规则
不允许访问外部Web网站	丢弃所有外出的、目的端口为80的包
不允许外部发起的TCP连接，除非访问的是内网的公共web服务器	丢弃进入的TCP SYN包，除非去往130.207.244.203的端口80
防止因特网广播吞噬网络带宽	除DNS包和路由器广播包，丢弃其它进入的UDP包
防止网络拓扑被探测（traceroute）	丢弃所有外出的ICMP TTL expired包

状态检测防火墙

- 包过滤防火墙孤立地过滤每个包，仍会允许一些异常的包进入：
  - 例如，允许dest port = 80, ACK=1的包进入，哪怕并没有相应的连接存在
- 状态检测防火墙可以跟踪TCP连接的状态：
  - 跟踪连接的建立（SYN）和关闭（FIN）等状态，判断收到的包是否有意义

应用网关

- 应用网关除了检查网络层及传输层协议头，还检查应用层数据
- 例如: 允许特定的内部用户使用telnet登录外部主机



1. 所有telnet用户必须连接到应用网关
2. 对于授权用户，应用网关建立与目的主机的telnet会话，并在2个连接之间中继数据
3. 包过滤防火墙阻塞所有不源自应用网关的telnet连接

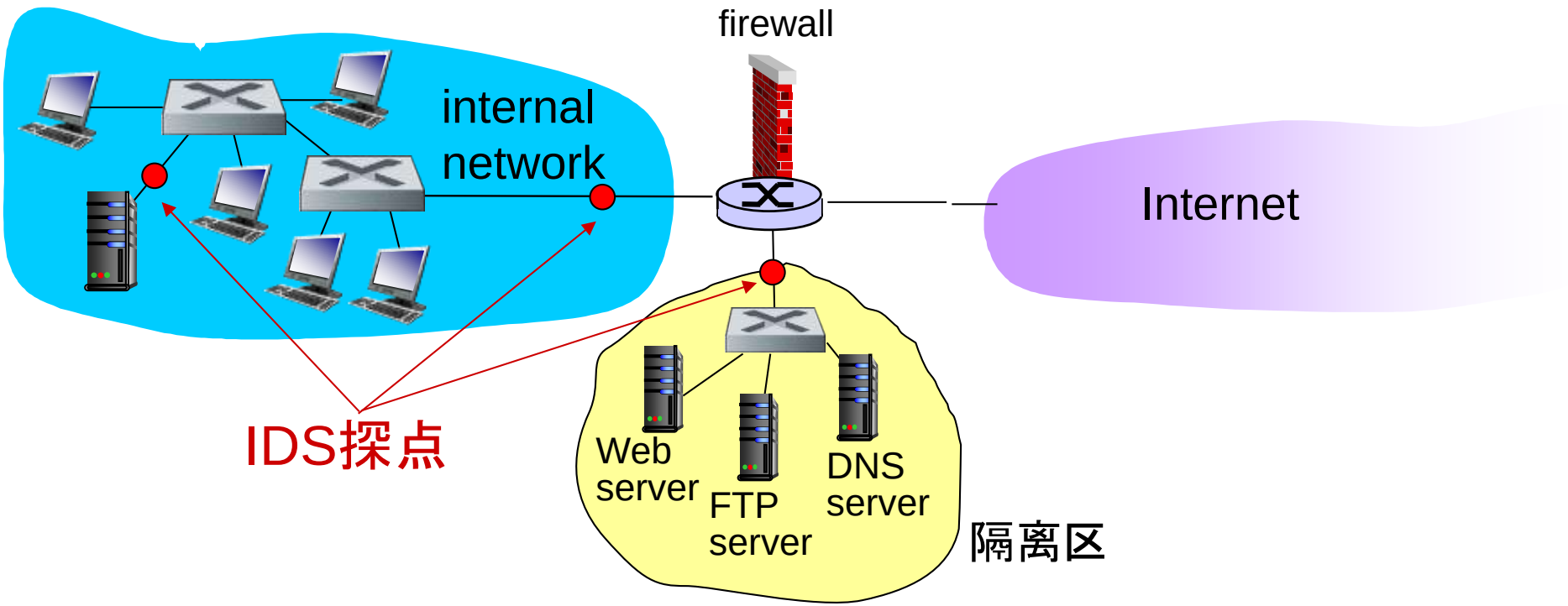
- 应用网关的局限性：
  - 应用网关处理开销大，速度慢
  - 每个被代理的应用都需要一个应用网关

Intrusion detection systems

- 防火墙:
  - 包过滤防火墙仅检查传输层和网络层协议头
  - 应用网关仅检查特定应用的数据包
  - 不检查数据包的内容及数据包之间的关联
- *IDS: intrusion detection system*
  - 深度数据包检查: 查看包内容（如检查包中是否包含已知的病毒特征、攻击特征等）
  - 检查多个包之间的关联性:
    - 端口扫描
    - DoS攻击

Intrusion detection systems

- 网络中可以设置多个IDS: 在不同位置进行不同类型的检查



作业

- 作业：
  - 8，12，18
- 提交时间：
  - 12月12日