

COMP 2322 Computer Networking

COMP 2322 Computer Networking

00. Introduction:

01. Computer Networks and the Internet

1.1 Internet

"nuts and bolts" view

Service view

1.2 Protocol

1.3 Network structure

Network edge

Access networks & physical media

住宅接入网络 DSL & cable

Access network: Digital Subscriber Line (DSL)

Access network: cable network

企业接入网络 (Ethernet)

无线接入网络 Wireless

Physical media (物理媒体)

Network core

packet switching 分组交换

circuit switching 电路交换

Packet switching vs circuit switching

1.4 Internet structure and ISP

Internet structure

ISP的三个层次和连接

1.5 Performance: loss, delay, throughput

loss and delay

Four sources of packet delay

throughput 吞吐量

1.6 Protocol layers, service models

protocol layers 协议层次

Service models

Internet protocol stack

ISO/OSI reference model

Encapsulation 封装和解封装

02. Application layer 交换报文, 实现网络应用

2.1 principles of network applications

Client-server

peer-to-peer (P2P)

Inter-process communication 进程通信

Addressing processes

What transport service does an app need?

Internet transport layer services

Internet apps: application / transport layer protocols

2.2 Web and HTTP

Web

HTTP

Non-persistent HTTP & Persistent HTTP

- Non-persistent HTTP
- Persistent HTTP
- HTTP message format
 - HTTP request message format
 - HTTP response message format
 - HTTP response status codes
- User-server state: cookies
- Web caching (proxy server)

2.3 Email

- SMTP [RFC 2821]

2.4 DNS

- DNS Name Space
- DNS hierarchy
- Local DNS server
- Domain name resolution
 - recursive query
 - iterated query
- DNS: caching, updating records
- DNS protocol, messages
- Inserting records into DNS

2.7 socket programming with UDP and TCP

- TCP socket programming
 - TCP Server
 - TCP Client
- C/S socket interaction: TCP
- UDP socket programming
 - UDP Server
 - UDP Client
- C/S socket interaction: UDP

03. Transport Layer 进程间的逻辑通信

- 3.1 transport-layer services and protocols
- 3.2 multiplexing and demultiplexing
 - Connectionless demux
 - Connection-oriented demux
- 3.3 connectionless transport: UDP
 - UDP checksum
- 3.4 principles of reliable data transfer
 - RDT 1.0 All reliable
 - RDT 2.0 Bit errors
 - RDT 2.1 ACK/NAK corrupted
 - RDT 2.2: NAK-free protocol
 - RDT 3.0 Errors & loss
 - RDT 3.0 stop-and-wait operation
 - RDT 3.0 Pipelined protocols
 - GBN (Go-Back-N)
 - sending window
 - receiving window
 - SR (Selective Repeat)
 - Pipelined Summary

3.5 connection-oriented transport: TCP

TCP segment structure

TCP sequence number, ACK

TCP序号和确认号之间的关系

TCP round trip time, timeout

EstimatedRTT

DevRTT

TCP reliable data transfer

TCP: retransmission scenarios

TCP fast retransmit

TCP flow control

3.6 TCP congestion control

TCP Tahoe

TCP Reno

04. Network (Data Plane) 端到端

4.1 Overview of network layer

data plane

control plane

4.2 What's inside a router

Input port functions:

Input port queuing

Switching fabrics

Switching via memory

Switching via a bus

Switching via interconnection network

destination-based forwarding (传统转发方式):

Longest prefix matching 最长前缀匹配

Output ports & queueing

4.3 IP: Internet Protocol

IPv4 addressing

Subnets

IP Datagram format

IP 分片和重组(Fragmentation & Reassembly)

Host如何获得一个IP地址?

DHCP: Dynamic Host Configuration Protocol (UDP)

4 important DHCP message overview:

DHCP: 不仅仅可以返回IP addresses

一个ISP如何获得一个地址块?

05. Network (Control Plane)

5.1 Intro: control plane

5.2 routing protocols

Routing algorithm classification

5.2.1 Link state (Dijkstra)

Examples 01

Examples 02

Examples 03

Examples 04

5.2.2 Distance vector (Bellman-Ford equation)

Distance vector: link cost changes

5.2.3 Link State & Distance Vector 算法的比较

5.3 intra-AS routing

Internet approach to scalable routing

5.4 Three intra domain routing protocols

RIP

OSPF (LS)

IGRP

5.4 routing among the ISPs: BGP (DV)

06. Link Layer and LANs 点到点传输层功能

6.1 services

Network interface card 网卡

6.2 error detection, correction

6.2.1 Error detection 错误检测

6.2.2 Parity checking 奇偶校验

6.2.3 Internet checksum 因特网检验和

6.2.4 Cyclic Redundancy Check (CRC) 循环冗余校验

6.3 multiple access protocols

6.3.1 MAC protocols

6.3.2 channel partitioning

Channel partitioning MAC protocols: TDMA

Channel partitioning MAC protocols: FDMA

6.3.3 Random access protocols

Slotted ALOHA

Pure (unslotted) ALOHA 效率比时隙ALOHA更差!

6.3.4 CSMA (carrier sense multiple access) 礼貌的对话人

6.4 LANs

6.5 a day in the life of a web request

00. Introduction:

Midterm Test 15%

- Midterm test (90 minutes) will be given at Week 8
- Close book test (one page reference sheet allowed)

Homework Assignment and Lab Report 25%

- Independent work for completing each homework assignment and lab report
- Assignments and report must be submitted before the due time. Late submission will cause the marks to be deducted 25% per day

Programming Project 15%

- A programming project is required and is due by the end of the semester
- Each student needs to complete a project report and demonstrate project result

Final Examination 45%

- Final examination (2 hours)
- Close book exam (one page reference sheet allowed)

WEEK	DATE/TIME	TOPIC
1	Jan 12 Thu 12:30-15:20PM	Introduction
2	Jan 19 Thu 12:30-15:20PM	Application Layer
3	Feb 2 Thu 12:30-15:20PM	Application Layer
4	Feb 9 Thu 12:30-15:20PM	Transport Layer
5	Feb 16 Thu 12:30-15:20PM	Transport Layer
6	Feb 23 Thu 12:30-15:20PM	Transport Layer
7	Mar 2 Thu 12:30-15:20PM	Network Layer
8	Mar 9 Thu 12:30-15:20PM	Midterm Test Network Layer
9	Mar 16 Thu 12:30-15:20PM	Network Layer
10	Mar 23 Thu 12:30-15:20PM	Network Layer
11	Mar 30 Thu 12:30-15:20PM	Data Link and MAC Sublayer
12	Apr 6 Thu 12:30-15:20PM	Data Link and MAC Sublayer
13	Apr 13 Thu 12:30-15:20PM	Data Link and MAC Sublayer

WEEK	TOPIC	TA
1	/	
2	Lab 1 Wireshark	WANG Chenxu
3	Tutorial 1	WANG Chenxu
4	Lab 2 HTTP	WANG Chenxu
5	Lab 3 DNS	CHEN Xiangchun
6	Tutorial 2	CHEN Xiangchun
7	Tutorial 3	CHEN Xiangchun
8	Lab 4 Socket Programming	MA Yuxiao / WANG Fangxiao
9	Lab 5 HTTP Programming	MA Yuxiao / WANG Fangxiao
10	Tutorial 4	CHU Junchi / WANG Li
11	Lab 6 TCP	MA Yuxiao / WANG Fangxiao
12	Tutorial 5	CHU Junchi / WANG Li
13	Tutorial 6	CHU Junchi / WANG Li

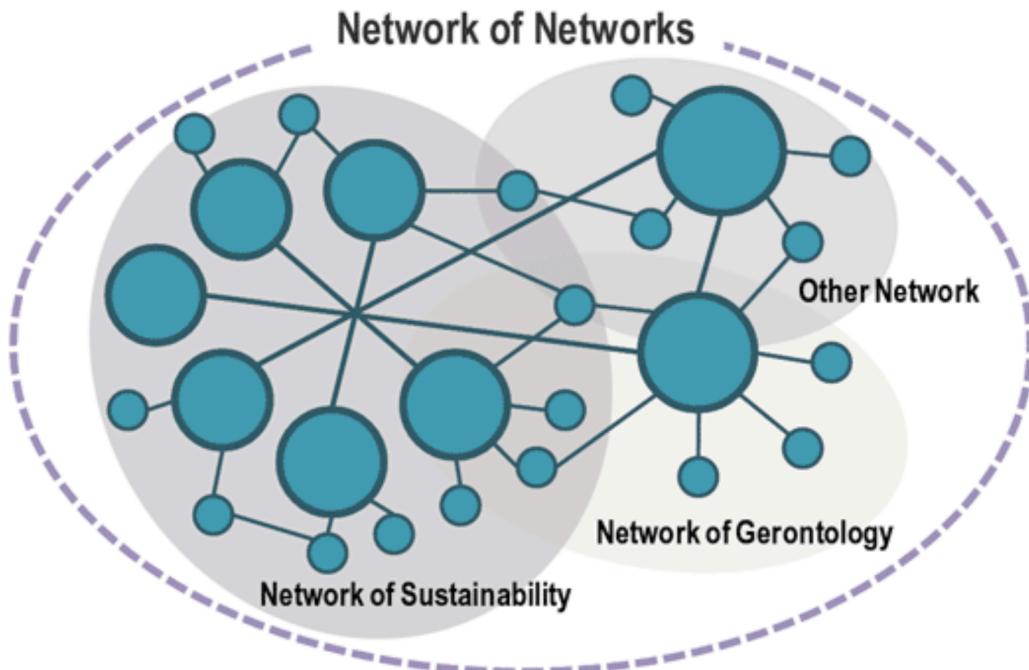
01. Computer Networks and the Internet

1.1 Internet

在本书中，我们使用一种特定的计算机网络，即公共因特网，作为讨论计算机网络及其协议的主要载体。

互联网 (Internet)，由计算机网络互联而成（即“网络的网络”）的庞大的系统，是以 TCP/IP 协议为主的一簇协议支撑工作的网络。

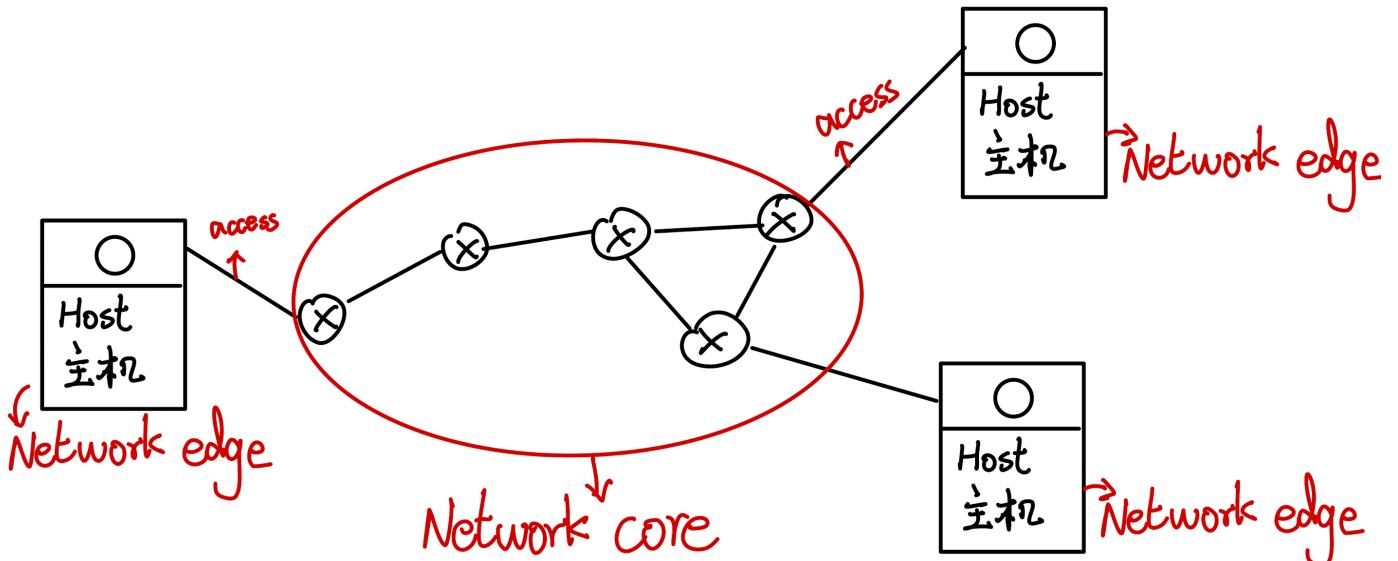
- 构成描述： **Internet = network edge + access nekwork + network core**
- 服务描述： 互联网是为 **分布式的应用进程** 提供通讯服务的 基础设施，包括应用层以下的 **protocol**。互联网通过套接字接口 (API) 为进程提供服务，规定了接入网络核心的端系统数据交付的方式。**分布式应用** (distributed application) 指的是应用程序分布在不同计算机上，通过网络来共同完成一项任务的工作方式。



"nuts and bolts" view

从因特网的具体构成（结构上）来看，Internet 就是一堆的网络，通过(几十亿)网络互联设备连在一起的一个巨型网络。其中有

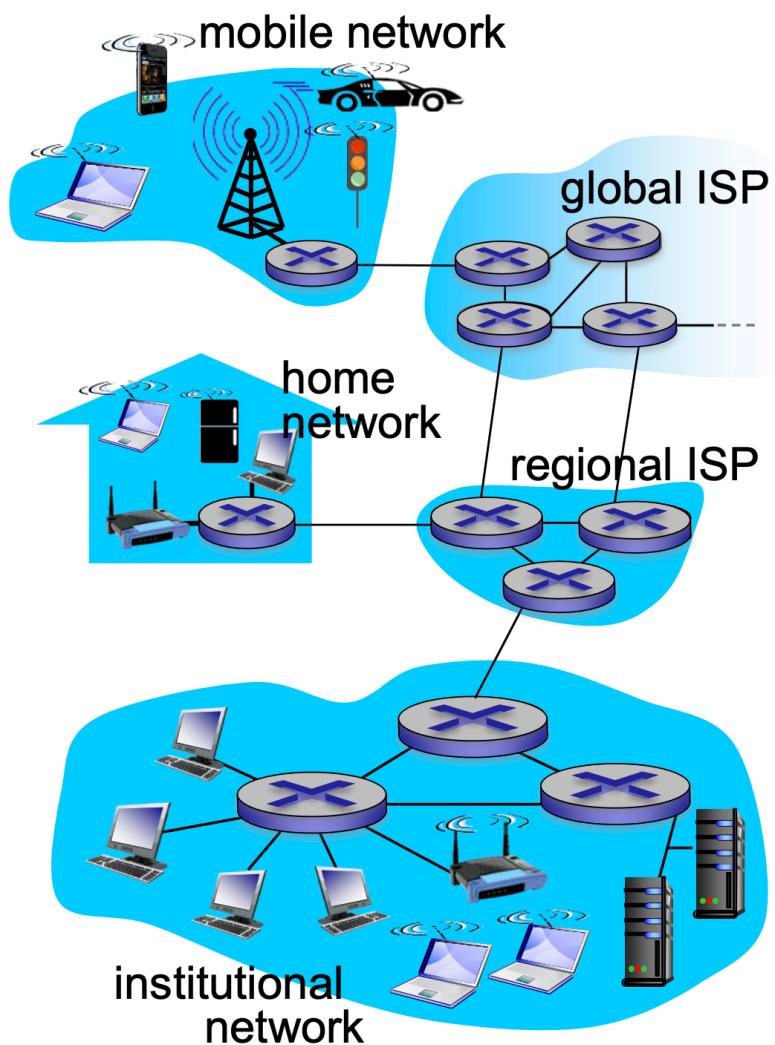
- **Network edge**: 和互联网连接的设备，称为: **hosts 主机 = end systems 端系统** (eg. iPad, iPhone, PC, MacBook) 来运行网络应用程序
- **Access networks (physical media)**: 就是有线或者无线通信链路，可以把边缘接入到网络核心，可以将报文发给另外一个主机。**end systems** 通过 **communication links (通信链路)** 和 **packet switches (分组交换机)** 连接在一起，**link 的 transmission rate = bandwidth 带宽(bit /s or bps)**，发送数据形成的信息包称为 **packet (分组)**
 - **packet switches 分组交换机(forward packets 转发设备)**: 作用是来了一个数据从哪个地方把它转走，不同的网络层工作的层次/支撑它运行的协议不一样，根据不同的协议可以把它分成链路层的交换机、网络层的路由器、防火墙等等。eg. routers 路由器 and switches 交换机
 - **communication links (通信链路)**: eg. fiber 光纤, copper 同轴电缆, radio 无线电, satellite 卫星
 - 当一台端系统要向另一台端系统发送数据时，发送端系统将数据分段，并为每段加上首部字节。由此形成的信息包用计算机网络的术语来说称为 **packet (分组)**。这些分组通过网络发送到目的端系统，在那里被装配成初始数据。
- **network core 属于基础设施**: 就是一堆互相连接的路由器(**mesh of interconnected routers**)，网络核心能够把所有的东西发给我所需要的目标主机，网络核心连接着所有端系统，从而我可以跟所有的端系统通信。作用：数据交换



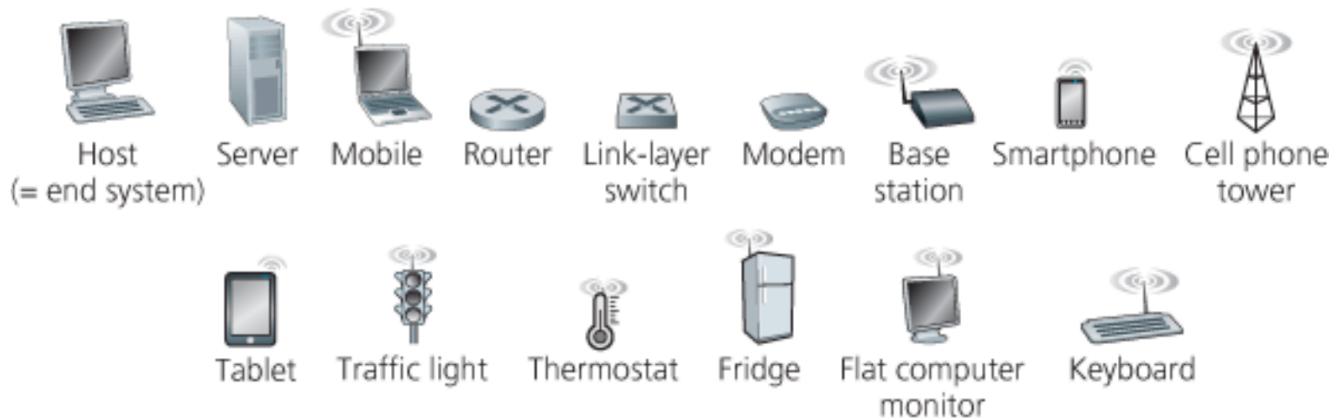
Service view

从服务角度来看：互联网是 分布式的应用 以及 为分布式应用提供通讯的这个基础设施

- Distributed application 是网络存在的理由，为 分布式应用（distributed application） 提供通讯服务的基础设施 (infrastructure, eg. PC)
- 通信基础设施为 apps 提供编程接口 API (通信服务)，使设备之间可以进行连接。eg. 允许发送和接收应用程序 "连接" 到互联网的接口 (API)



ISP: Internet Service Provider

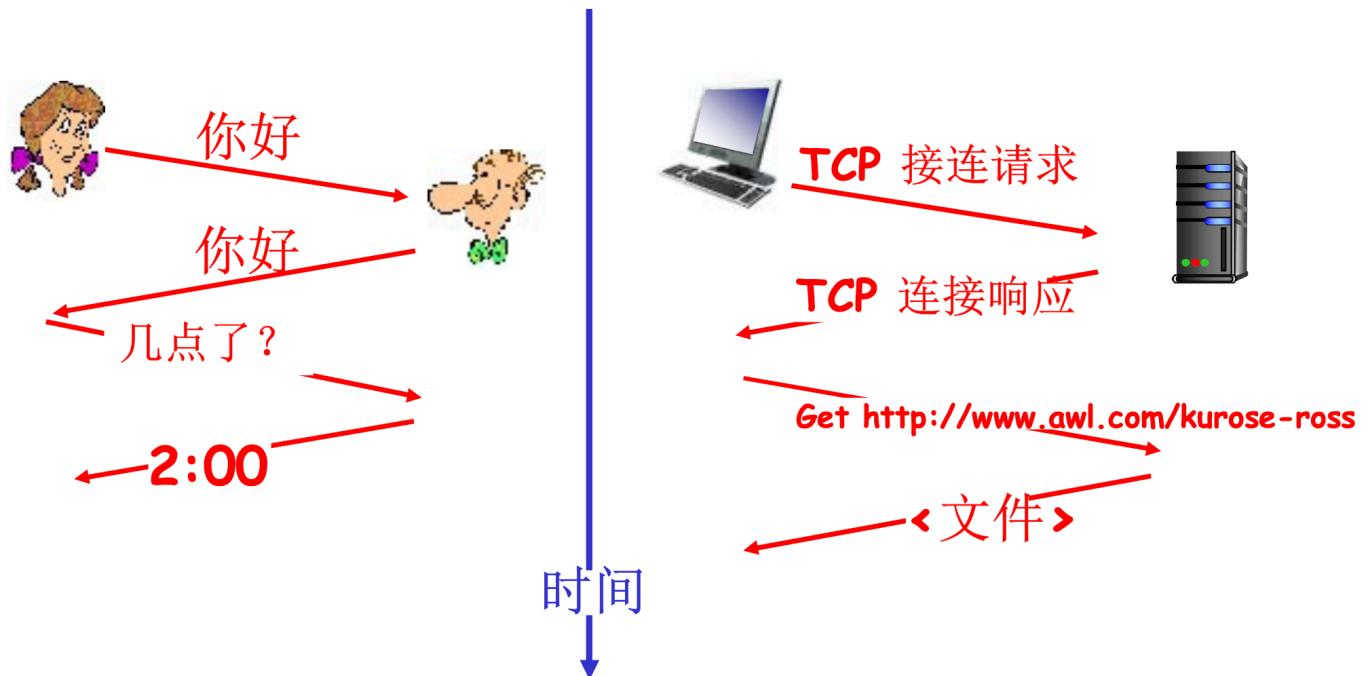


1.2 Protocol

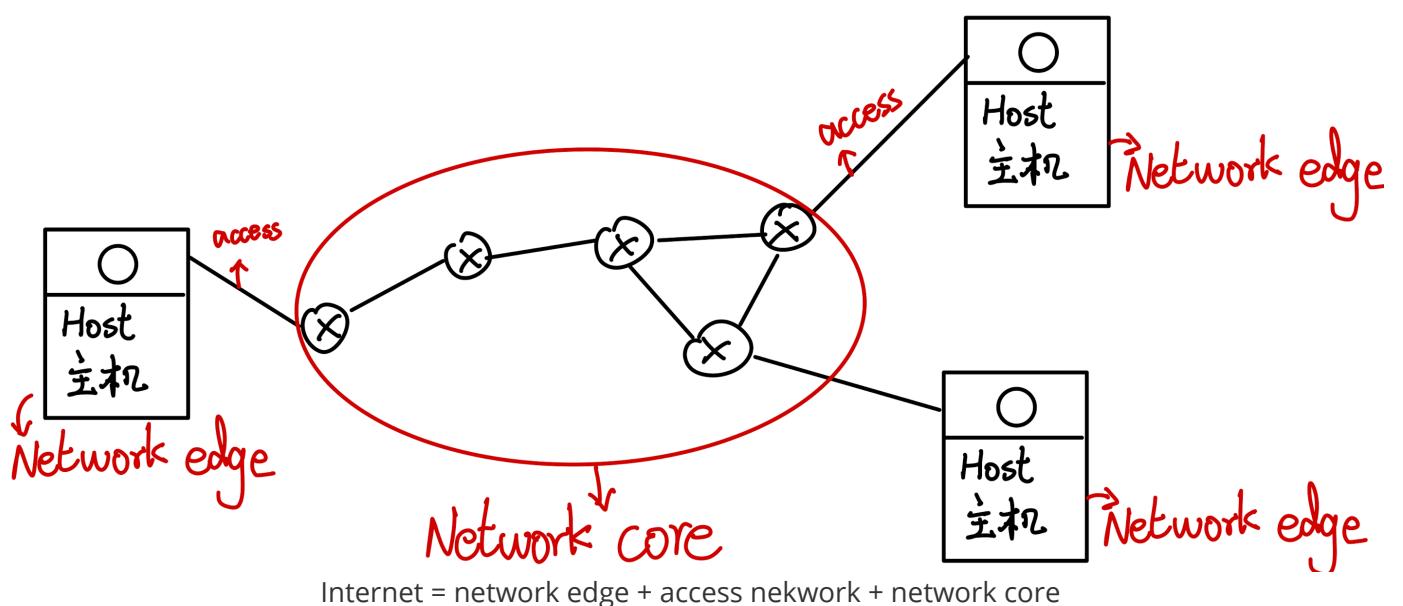
- **protocol 协议(标准):** Internet 中所有的通信行为都受协议制约。对等的实体在通讯过程当中，应该遵守的规则集合，是一个标准和规范。协议定义了在两个或多 个通信实体之间交换的 **format, order of messages sent** (报文格式和次序)，以及 在报文传输和/或接收或 其他事件方面所采取的 **actions** (动作)
- 协议控制发送、接收消息 eg. TCP, IP, HTTP, FTP, PPP, Skype, 802.11

例：一个web的浏览器运行在这个PC机上，他们首先要建立起一个TCP的连接，然后我按照 http 的协议规范，向对方发出请求。发出请求之后，就是把这个报文封装在 http 请求的报文格式当中，按照TCP的规范发给对方。对方建立起TCP的连接之后，就能够收到这个报文，然后根据协议解析这个报文，读取对方请求的信息/动作，捡取消息后把该消息封装成http相应的报文格式发送给对方。

人类协议和计算机网络协议示例



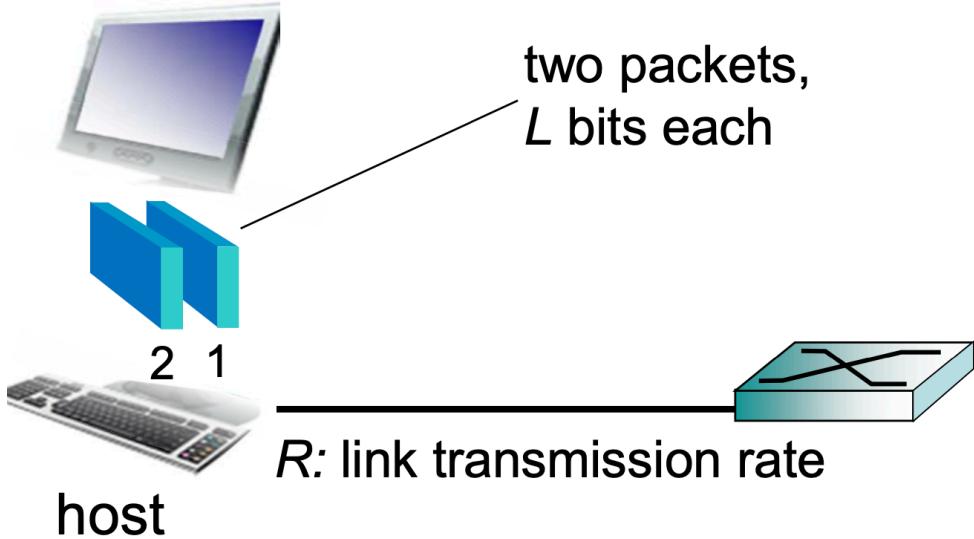
1.3 Network structure



Network edge

network edge 网络边缘, 不属于基础设施: hosts (clients and servers) -> host(客户端, 手机, 电脑)可以跑应用(游戏, 电子商务) -> sends/receives packets of data

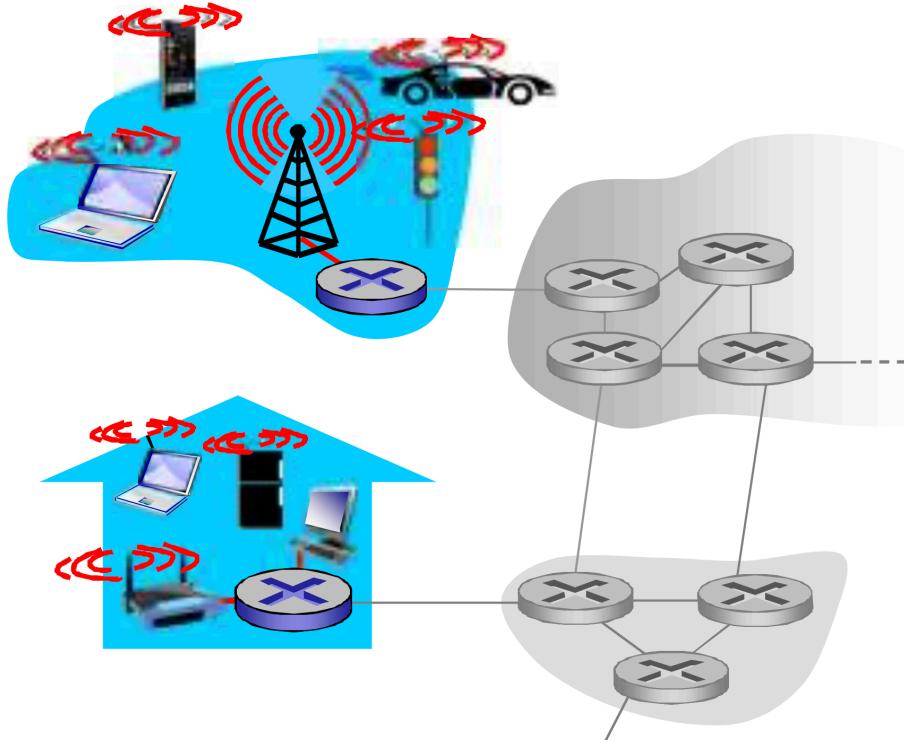
主机发送功能: 获取应用消息, 分成 **smaller chunks** (小的数据块), 称为 **packets**, 长度为 **L bits**。以 **transmission rate R** 将数据包传输到接入网络。**link transmission rate = link bandwidth = link capacity**



$$\text{packet transmission delay} = \frac{\text{time needed to transmit } L\text{-bit packet into link}}{R \text{ (bits/sec)}} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

Access networks & physical media

- **access networks (physical media)** 接入网络和物理媒体, 属于基础设施: wired, wireless communication links. 就是把边缘接入到网络核心, 可以将报文发给另外一个主机。
- 接入有一个非常重要的指标: **bandwidth (data rate) of access network: bps (bits per second), Kbps, Mbps, Gbps**
- 将端系统和边缘路由器连接(**connect end systems to edge router**) 有几种接入方式: 住宅接入网络 (图片左下角)、单位接入网络 (学校、公司)、无线接入网络 (图片左上角)。有独享和共享接入方式。例: 那我们现在通过这个中国电信的这个光纤接入到电信的这个网络交换设备, 是独享。在公司/学校里, 如果使用的是有限接入公司的带宽, 和几百个电脑共享这个带宽, 凌晨访问可能可以有100Mps, 但是平常下午5、6点就可能只有3-4Mps

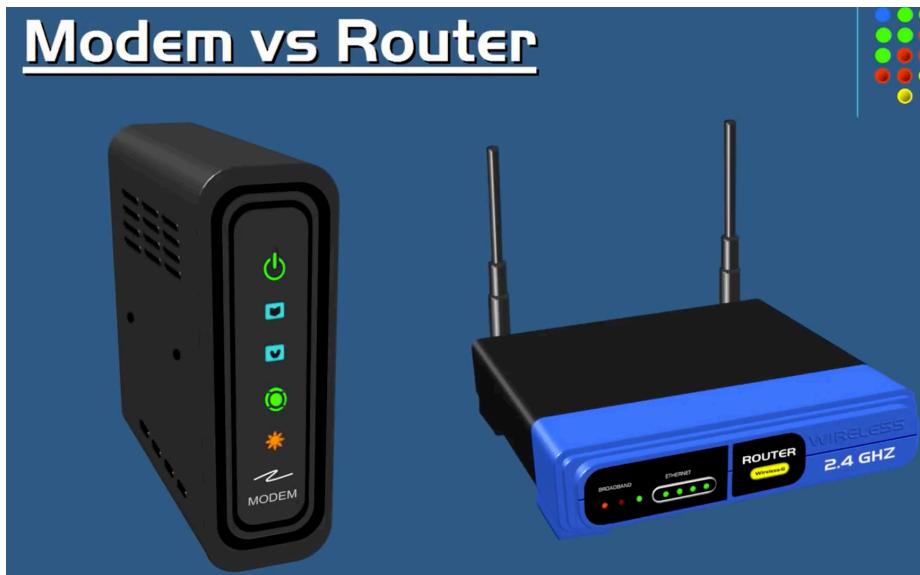


在 Access networks & physical media 中有几种 将端系统和边缘路由器连接(connect end systems to edge router) 的连接方式

住宅接入网络 DSL & cable

宽带住宅接入有两种最流行的类型是：数字用户线 (Digital Subscriber Line, DSL) 和 cable。住户通常从提供本地电话接入的本地电话公司处获得 DSL 因特网接入。因此，当使用DSL时，用户的本地电话公司也是它的ISPC

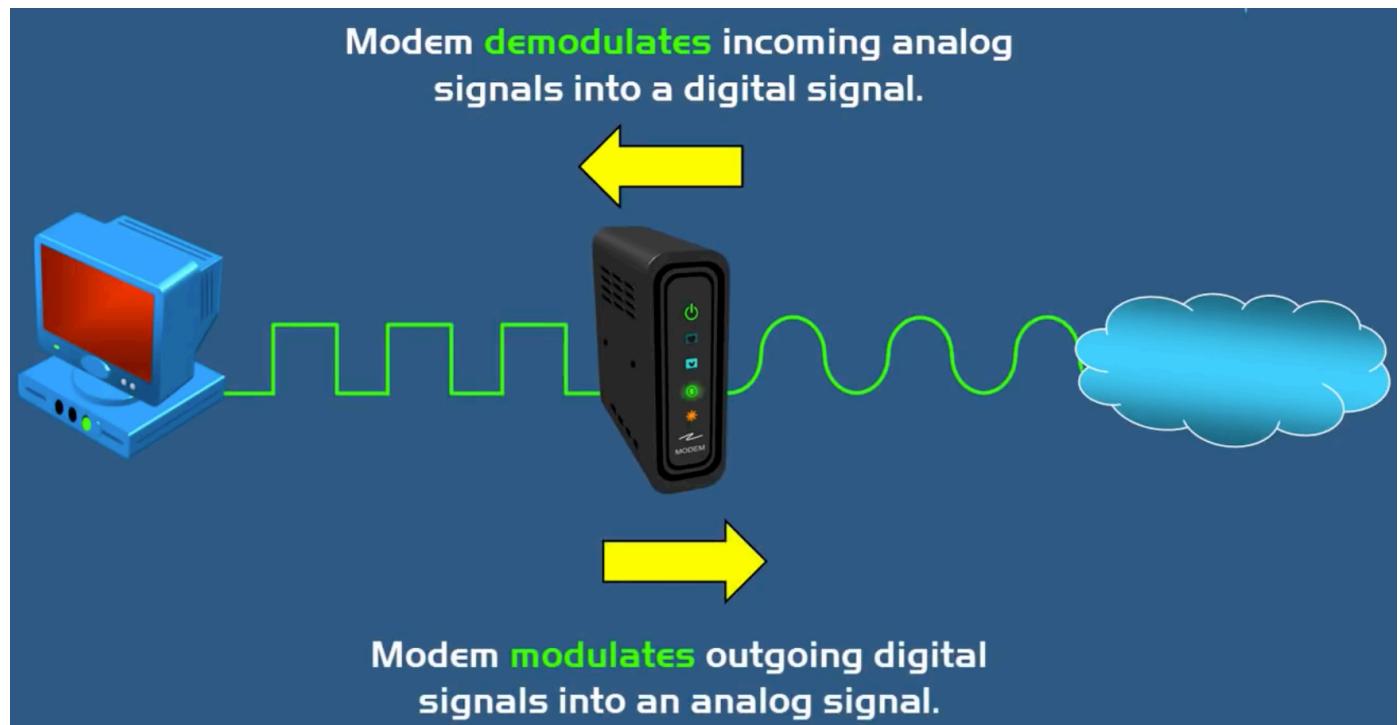
在此之前，我们先了解一下 modem(调制解调器) 和 router(路由器) 区别：



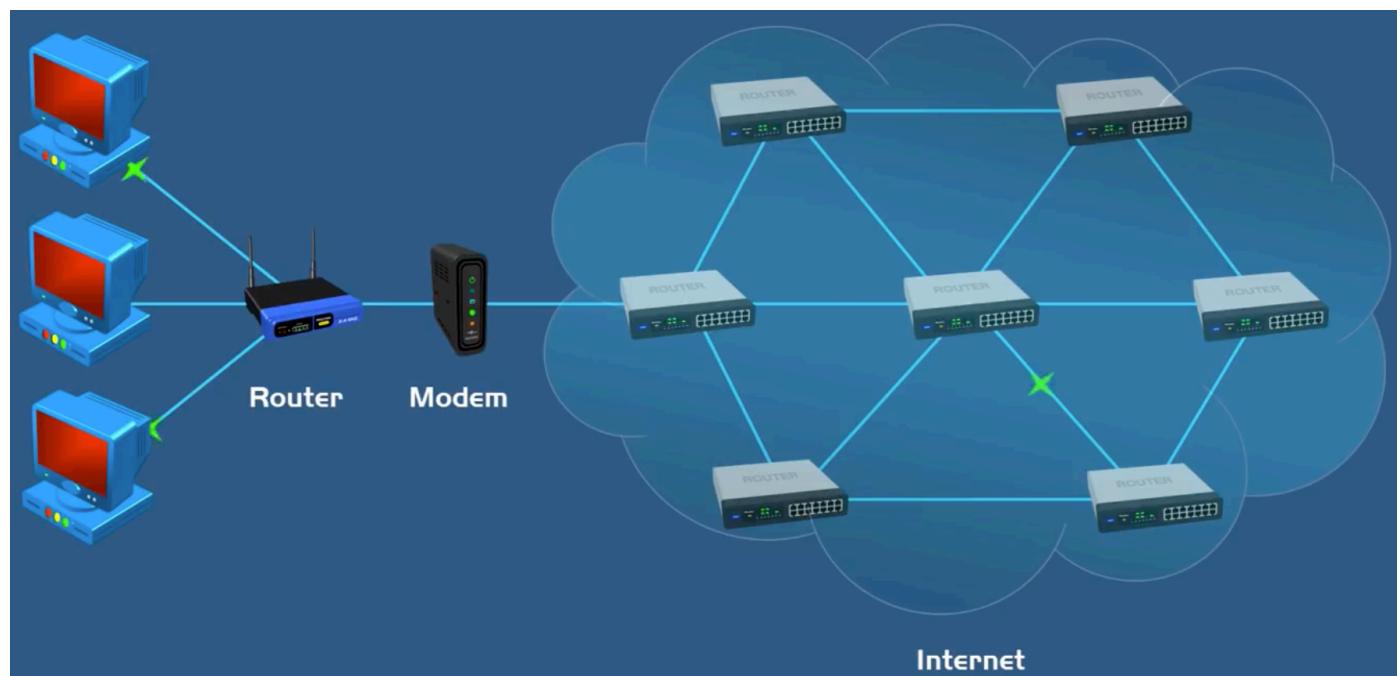
modem 能将互联网带入您的家庭/企业，建立并维护与 ISP 的专用 link，以至于我们才可以访问互联网。

之所以我们需要使用 modem 的原因是因为：计算机和互联网上使用的是两种不同类型的信号（计算机是数字信号且只读取数字信号，互联网上的信号是模拟信号且只读取模拟信号。A computer only reads digital signals. The internet only reads analog signals.)

当模拟信号从互联网上传入我们家时，modem 就会将输入的 analog signals 解调为 digital signals，以便计算机能够理解。反之，modem 也可以将来自计算机的输出的 digital signals 解调为 analog signals。



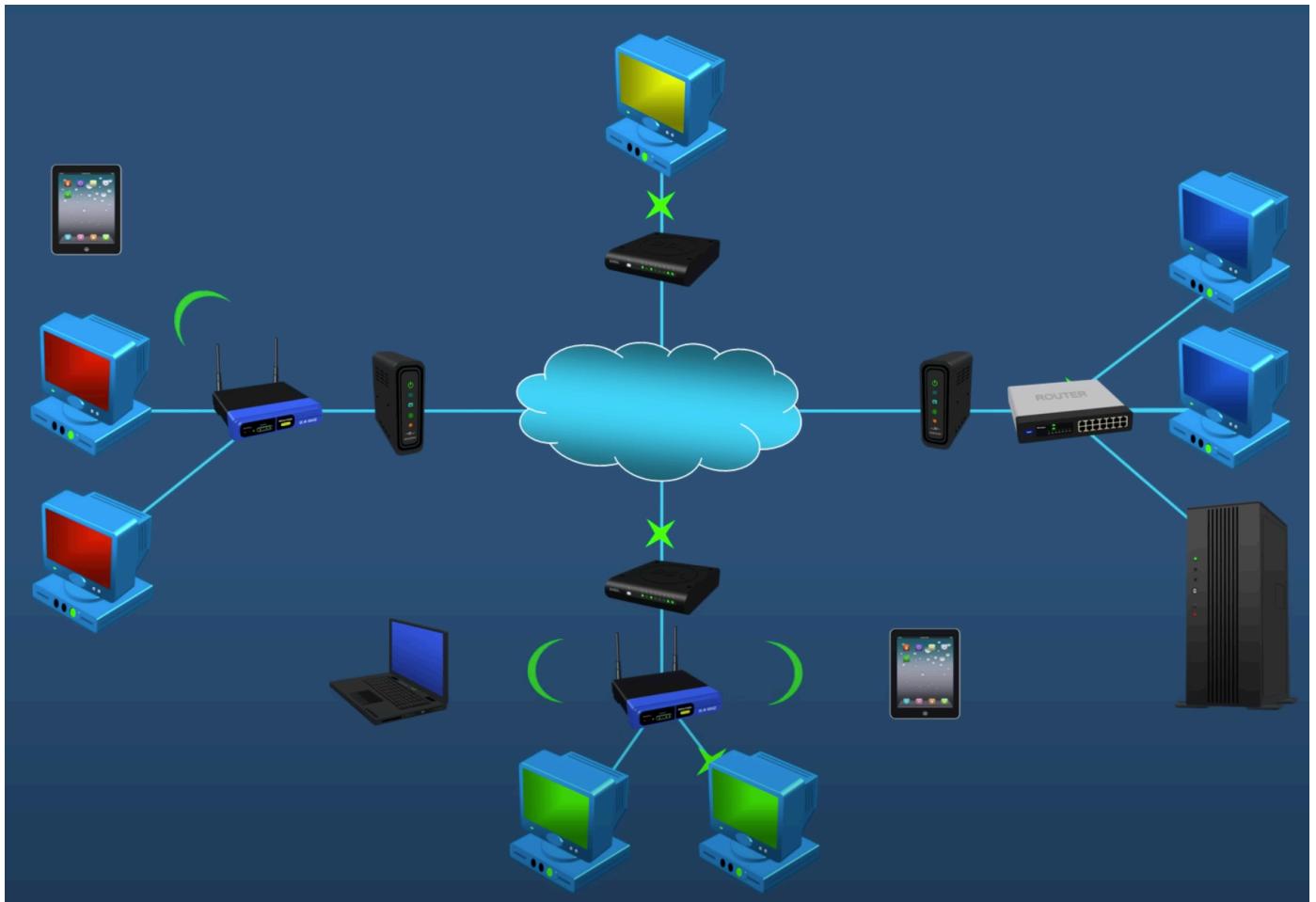
所以从技术上来讲，其实并不需要 router 如果只想要一台设备访问 internet，但是如果有多台机器就需要使用 router 了。



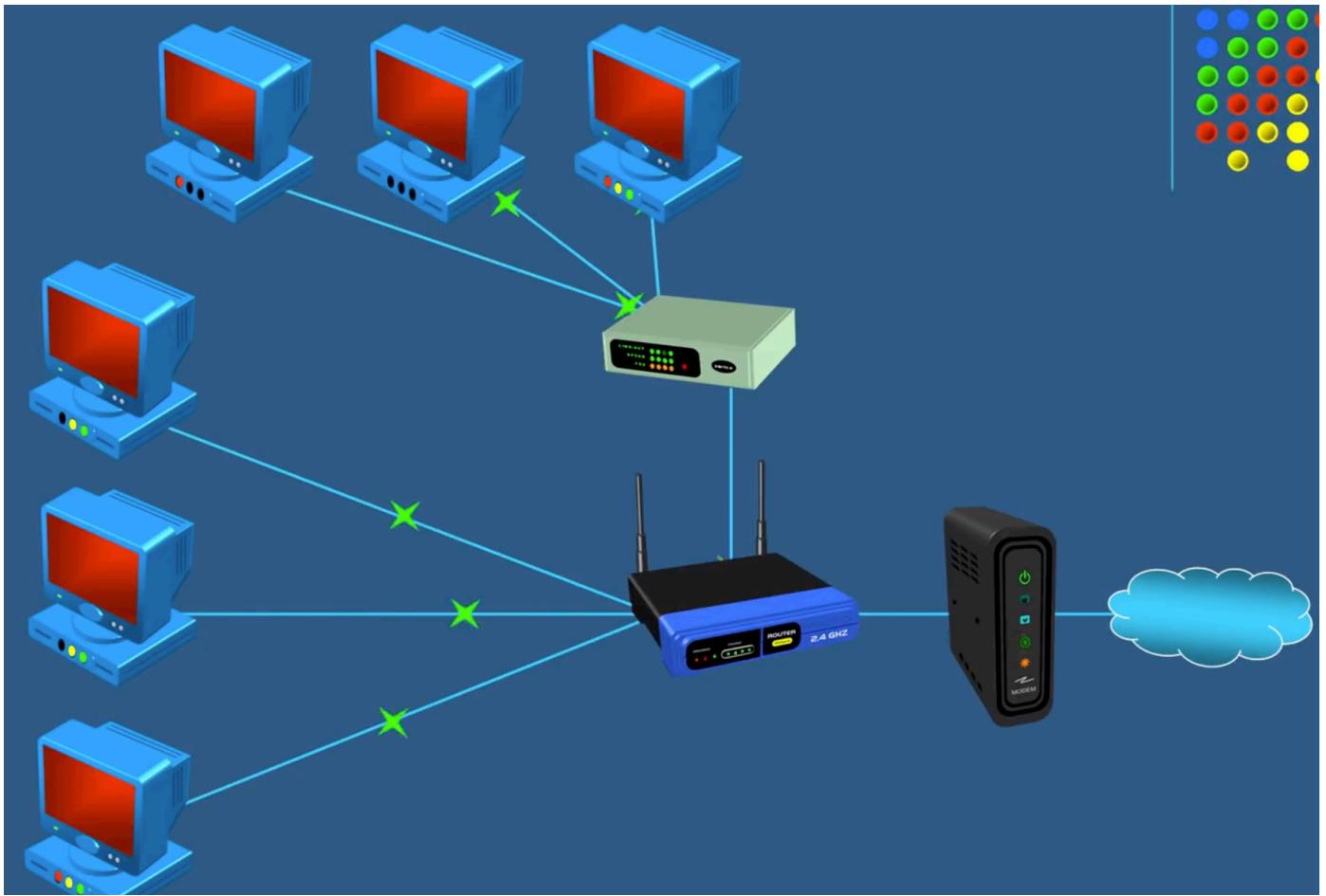
modem 分为 DSL modem 和 cable modem：



router 可以通过网线连接有限设备，通过 Wi-Fi 连接无线信号。下面这幅图演示了，多个不同的网络是如何连接到互联网的。



一般情况下，router 内部都有一个 switch交换机来连接有限设备，假如 switch 有3条接口，那我就可以连接3个有限设备。但是如果我有更多的设备，需要有线连接，那么我就需要额外的 switch：



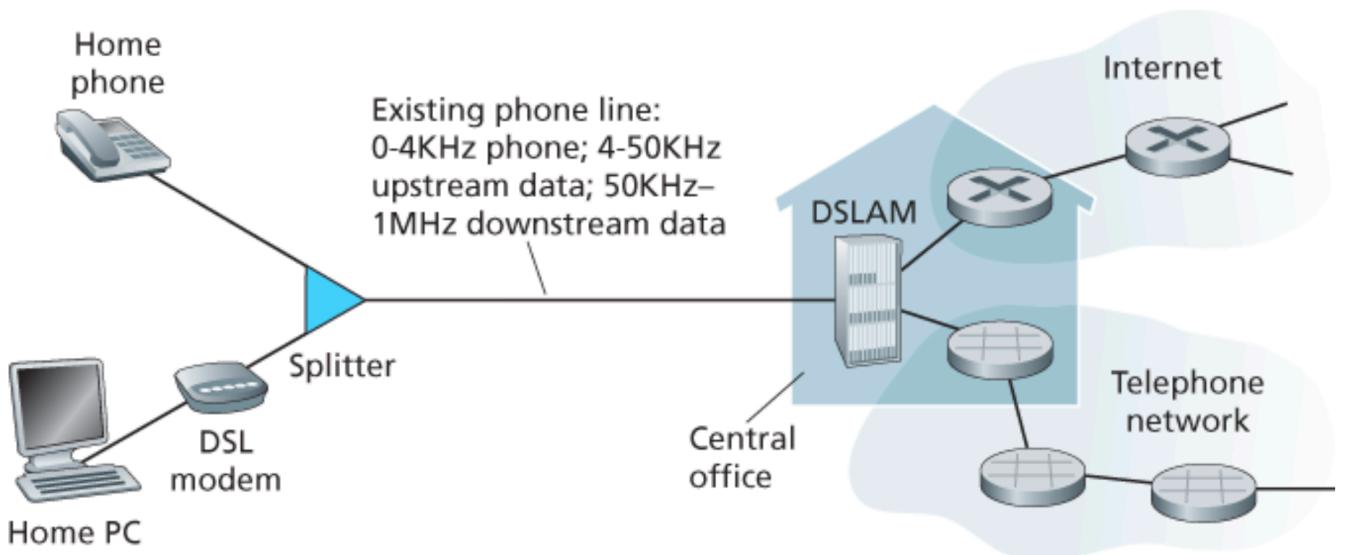
为什么20M宽带，我最大下载速度只能到2.5M?

宽带太小一般用的单位是20Mb，而我们平时一般是用MB作单位的。他们之间的换算关系是：1Byte=8bit。

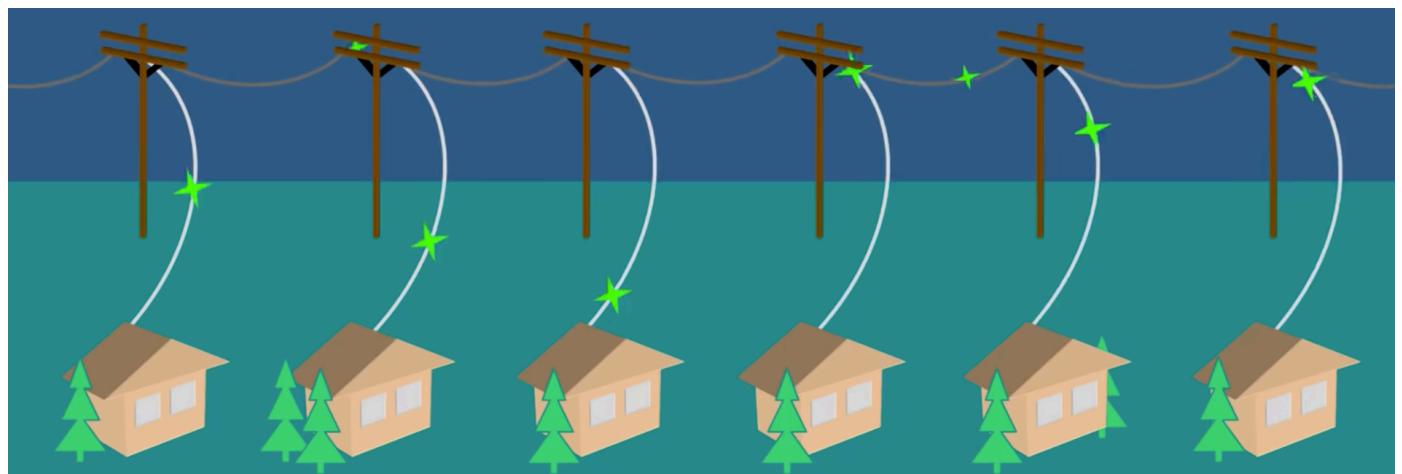
Access network: Digital Subscriber Line (DSL)

- 是使用调制解调方式，直接用电话线和互联网连接，但是带宽不是很大。use **existing telephone line** to central office **DSLAM (DSL接入共享器)**
- DSL线路上的数据被传到互联网，DSL线路上的语音被传到电话网，DSL有专用的连接到中央办公室的通道
- < 2.5 Mbps upstream transmission rate (typically < 1 Mbps) 上行/上传速度
- < 24 Mbps downstream transmission rate (typically < 10 Mbps) 下行/下载速度
- 因为这些上行速率和下行速率是不同的，所以这种接入被称为是不对称的。因为他的下载速度比上传速度快得多
- 数据在专享线路的不同频段传输。现有电话线: 0~4kHz为电话; 4~50kHz 为上行数据; 50kHz~1MHz 为下行数据。

家庭的 DSL modem 得到 数字数据 后将其转换为 高频音，以通过电话线传输给 本地中心局CO(central office); 来自许多家庭的模拟信号在 DSLAM 处被转换回数字形式。

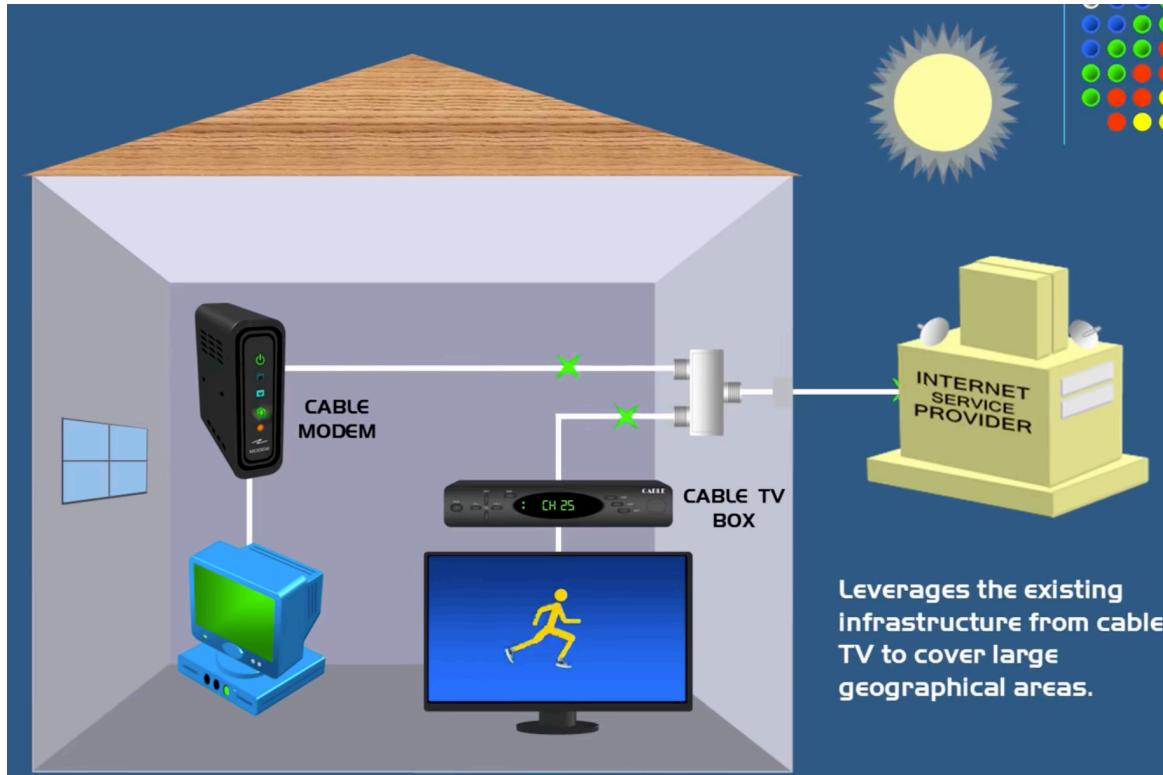


虽然使用 DSL 比cable modem慢，但并不用像有线连接那样与邻居共享宽带。每个使用DS L的用户都有自己的专用连接，而不是共享线路，而且更加便宜，因为它使用的是几乎无处不在的普通电话线。

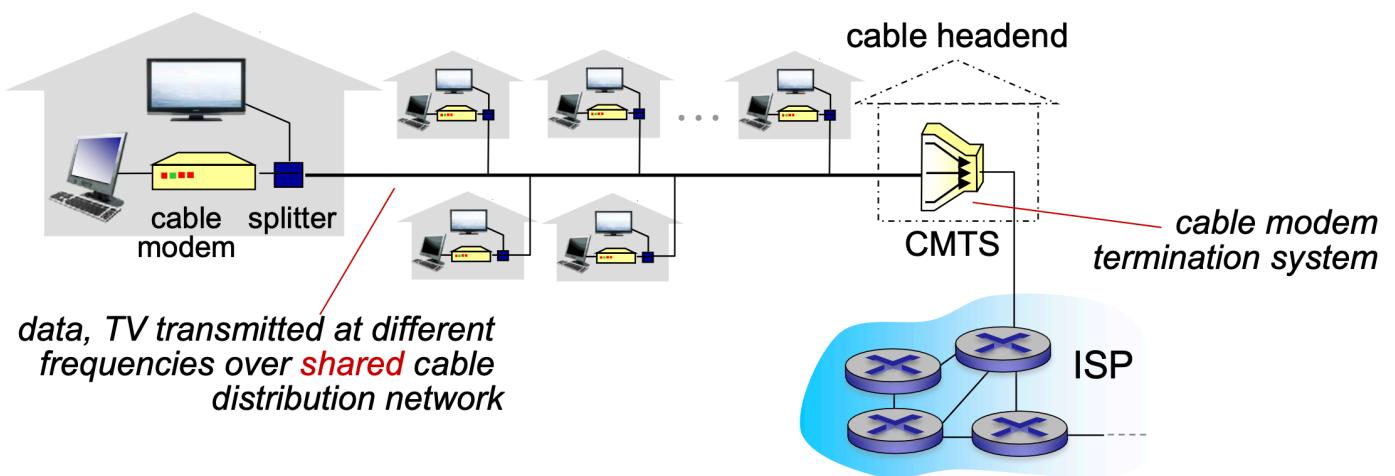


Access network: cable network

- DSL 利用电话公司现有的本地电话基础设施，而电缆因特网接入 (cable Internet access) 通常由向用户提供有线电视的同一提供商提供，也就是说用户可以利用有线电视的 cable 来上网

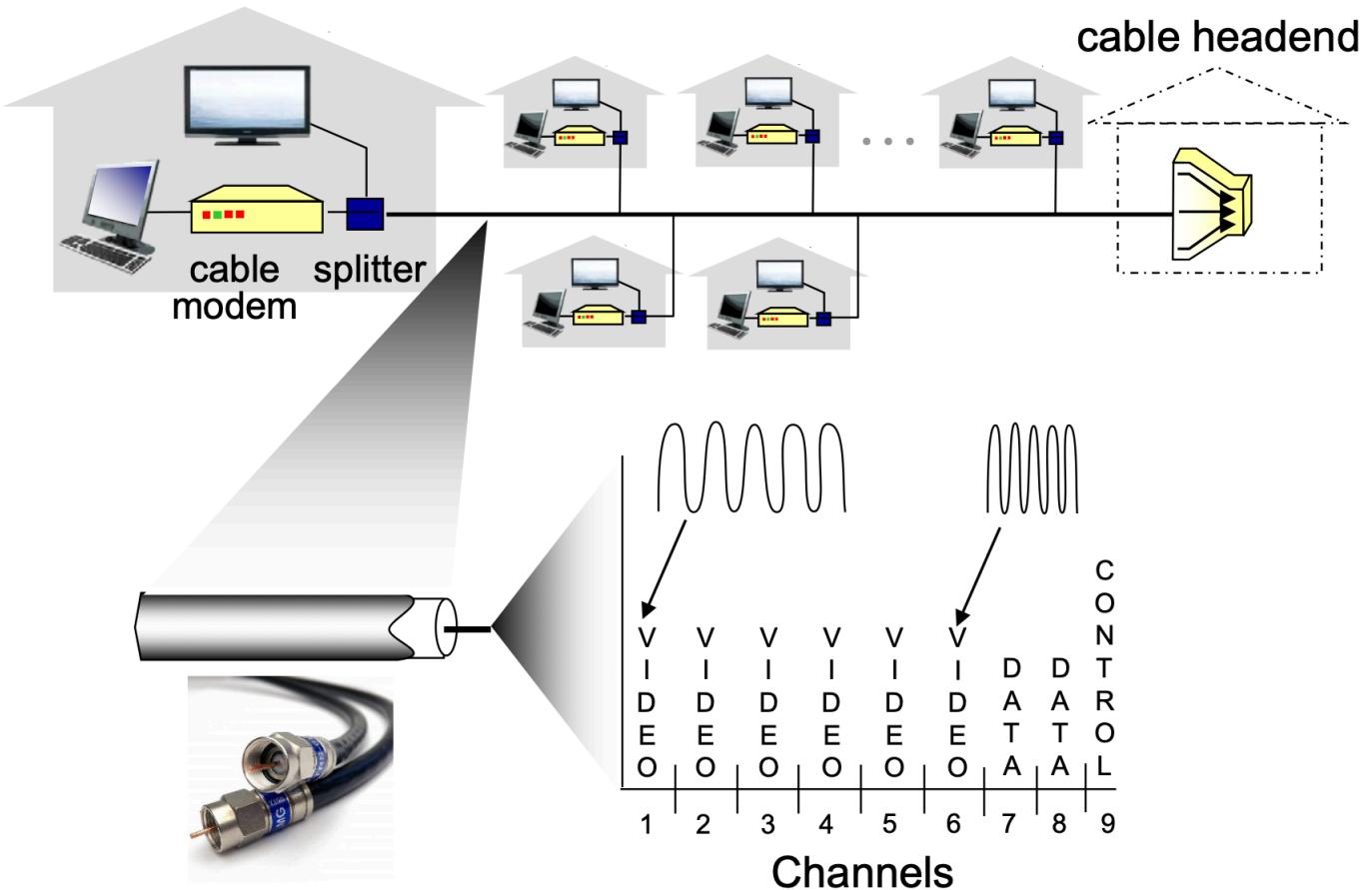


- 但是有一个缺点就是用户需要和他附近的其他家庭共享宽带。住宅从提供有线电视的公司获得了电缆因特网接入。如图所示，光缆将电缆头端连接到地区枢纽，从这里使用传统的同轴电缆到达各家各户和公寓。每个地区枢纽通常支持 500 -5000 个家庭。因为在这个系统中应用了光纤和同轴电缆，所以它经常被称为混合光纤同轴 (Hybrid Fiber Coax, HFC) 系统。所以在高峰期网速就会比较慢。
- Asymmetric (非对称):** 最高 30 Mbps 的下行传输速率 (downstream transmission rate) 和 2 Mbps 上行传输速率 (upstream transmission rate)。因为这些上行速率和下行速率是不同的，所以这种接入被称为是不对称的。
- 线缆和光纤网络将个家庭用户接入到 ISP 路由器，各用户共享到线缆头端的接入网络。与 DSL 不同，DSL 每个用户一个专用线路到 CO(central office)。电缆因特网接入需要特殊的调制解调器，这种调制解调器称为电缆调制解调器 (cable modem)。



有线电视信号线缆双向改造，大家是共享的带宽

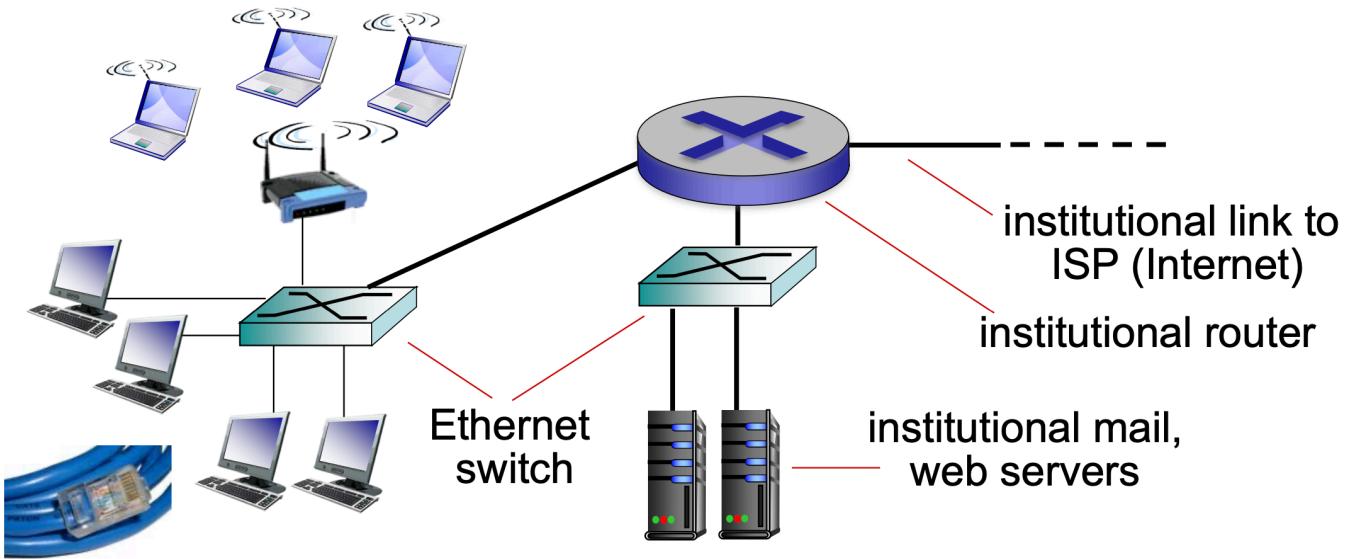
FDM: 在不同频段传输不同信道的数据，数字电视和上网数据(上下行)



接入类型	互联网服务提供商	依托的固有基础设施	使用的技术
数字用户线 (Digital Subscriber Line, DSL)	本地电话公司	由双绞铜线组成的电话线网络	数字用户线复用器 (DSLAM)
电缆因特网接入 (cable Internet access)	有线电视公司	由光线和同轴电缆组成的混合光线同轴系统	电缆调制解调器端接系统 (Cable Modem Termination System, CMTS)
光纤到户 (fiber to the home, FTTH)	本地中心局	建立一条直接入户的光纤路径	光线线路端接器 (Optical Line Terminator, OLT)

企业接入网络 (Ethernet)

Enterprise access networks (Ethernet) 企业接入网(以太网): 经常被企业或者大学等机构采用，公司、大学校园和企业等单位的终端用户使用双绞铜线，通过局域网 (LAN) 的方式接入以太网 (Ethernet) 交换机，连接边缘路由器。10 Mbps, 100Mbps, 1Gbps, 10Gbps传输率。现在，端系统经常直接接到以太网络交换机上。



无线接入网络 Wireless

Wireless access networks (无线接入网络) 有两种:

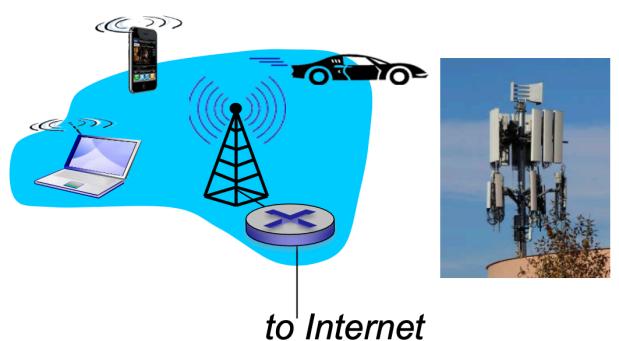
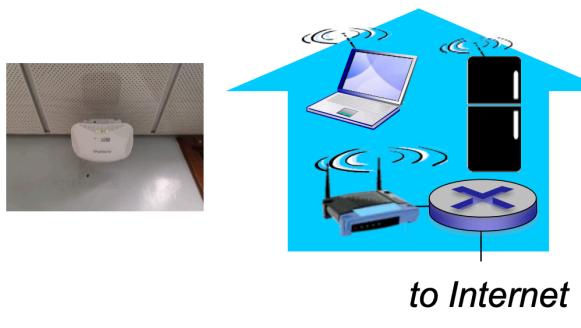
- **wireless LANs (无线局域网):** 各 wireless end systems 共享 wireless router (Wi-Fi), wireless LANs = WLAN = Wi-Fi
- **wide-area wireless access (广域无线接入):** 通过 base station(基站) 或者叫 access point(接入点)

wireless LANs:

- within short range (100's meters)
- 802.11b/g/n/ac (WiFi):
11/54/450/1730 Mbps data rate

wide-area wireless access

- provided by telco (cellular) operator, 10' s km
- 3G, 4G: LTE, 5G: data rates range 0.1Mbps ~ 1Gbps



Physical media (物理媒体)

- **signal:** propagates between transmitter/receiver pairs
- **physical link:** lies between transmitter & receiver
- **guided media:** 信号沿着固体媒介被导引: copper, coax, fiber
- **unguided media:** 开放的空间传输电磁波或者光信号, 在电磁或者光信号中承载数字数据 e.g., radio
- **twisted pair (TP):** 两根绝缘铜导线拧合(two insulated copper wires). Category 5: 100 Mbps, 1 Gbps Ethernet. Category 6: 10Gbps
- **coaxial cable:** two concentric copper conductors, bidirectional. Broadband: multiple channels on

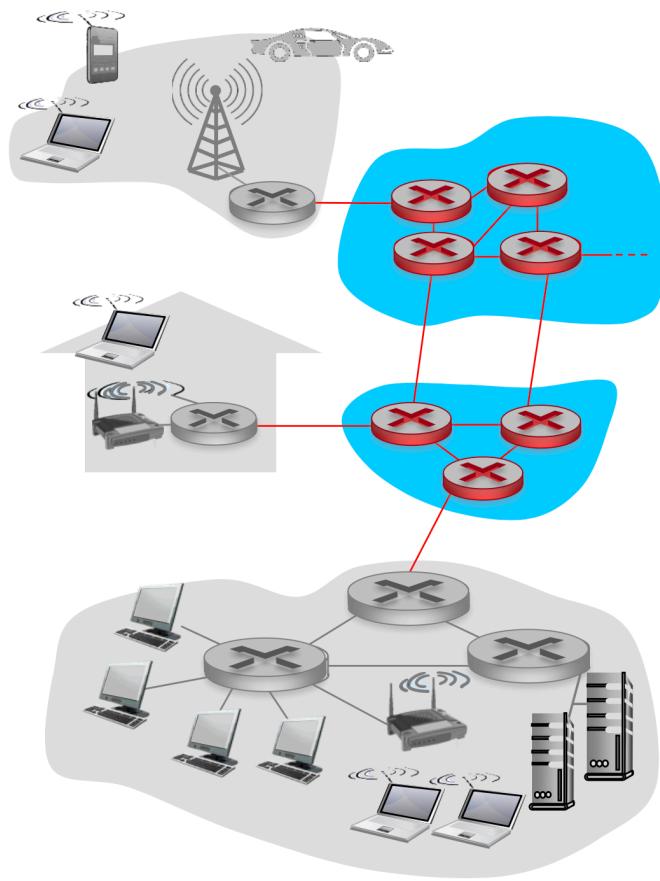
cable, HFC

- **fiber optical cable:** glass fiber carrying light pulses. high-speed operation: high-speed point-to-point transmission (e.g., 10's-100's Gbps transmission rate). low error rate: repeaters spaced far apart, immune to electromagnetic noise

Network core

- 目的/作用：数据交换
- 实现的方式有两种：packet switching 分组交换 & circuit switching 电路交换

network core 属于基础设施：就是一堆互相连接的路由器(**mesh of interconnected routers**)，网络核心能够把所有的东西发给我所需要的目标主机，网络核心连接着所有端系统，从而我可以跟所有的端系统通信。



packet switching 分组交换

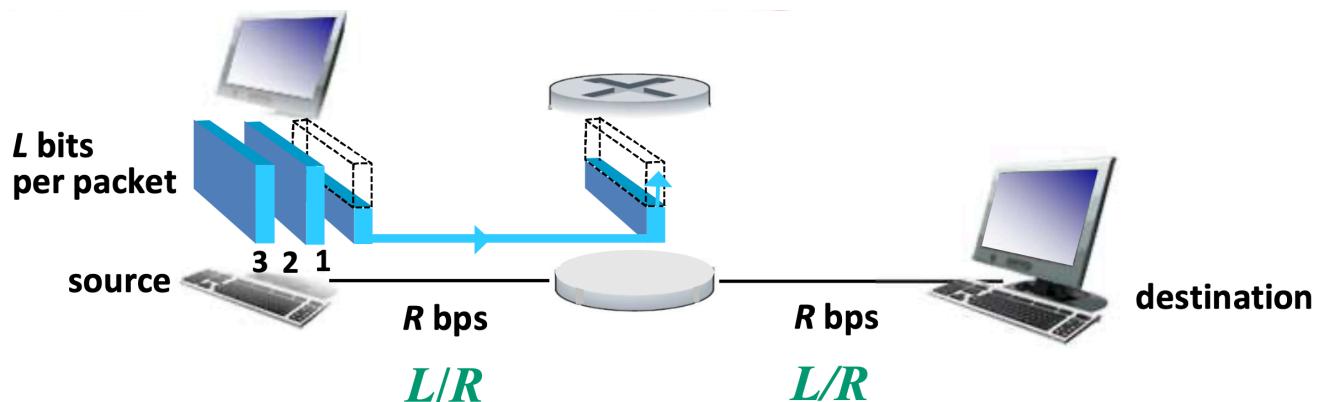
- **store-and-forward (存储-转发):** 网络带宽资源不再分分为一个个片，传输时使用全部带宽
 - 将要传送的 messages/data 分成一个个单位 packet(分组)
 - 将packet从一个路由器传到相邻路由器(hop)，一段段最终从源端传到目标端，期间可能通过许多不同的路由器，就会不断发生存储和转发。在转发之前，节点必须收到整个 packet，延迟比 circuit switching 要大，排队时间长。但是换取了资源共享性
 - each packet 采用链路的最大传输能力(带宽)
 - 在一个邮局通信系统中，邮局收到一份邮件之后，先存储下来，然后把相同目的地的邮件一起转发到下一个目的地，这个过程就是存储转发过程，分组交换也使用了存储转发过程。

在一个速率 R bps 的 link, 一个长度为 L -bit packet 以 R bps 的速率传输到 link 的存储转发延时需要 L/R seconds

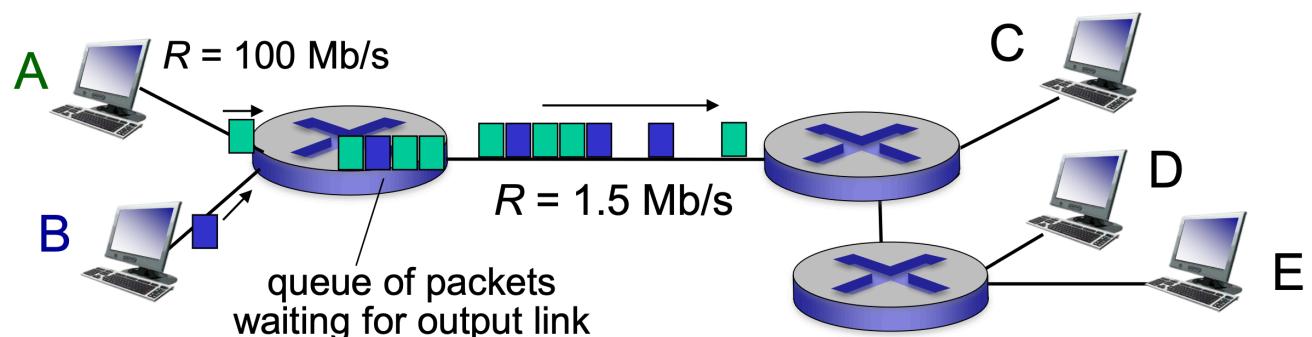
Example: $L = 7.5$ Mbits, $R = 1.5$ Mbps

one-hop transmission delay(每次存储的延迟) = 5 s。注意接收也需要存储的延迟, 图中就应该是有10s。

end-end delay = $2L/R$ (assuming zero propagation delay)



- queueing delay, loss (排队延迟和丢失): 如果 arrival rate (in bits) to link > transmission rate: packets 将在路由器上排队, 等待在link上传输。如果 memory (buffer) 填满/用完了, packets 可能会 被抛弃/丢失 dropped (lost)。可以说这是 packet switching 分组交换 为了获得共享性所必须要付的代价。

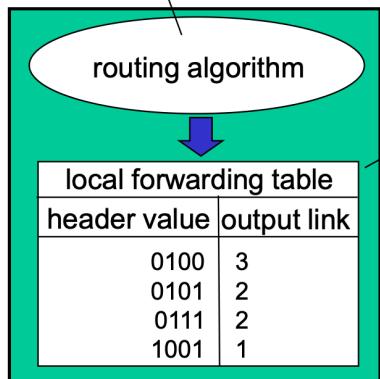


- Two key network-core functions:

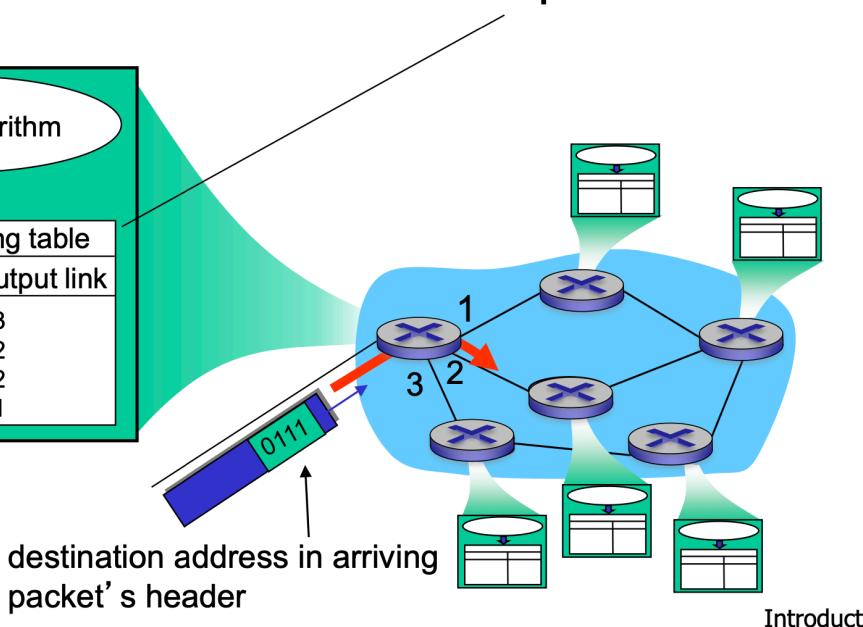
- routing (路由): 决定分组采用的源到目标的路径
- forwarding (转发): 将分组从路由器的输入链路转移到输出链路

routing: determines source-destination route taken by packets

- *routing algorithms*



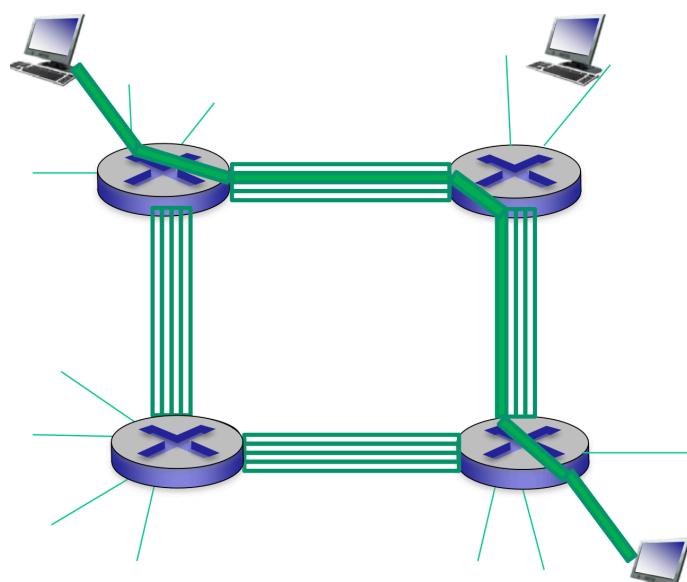
forwarding: move packets from router's input to appropriate router's output



circuit switching 电路交换

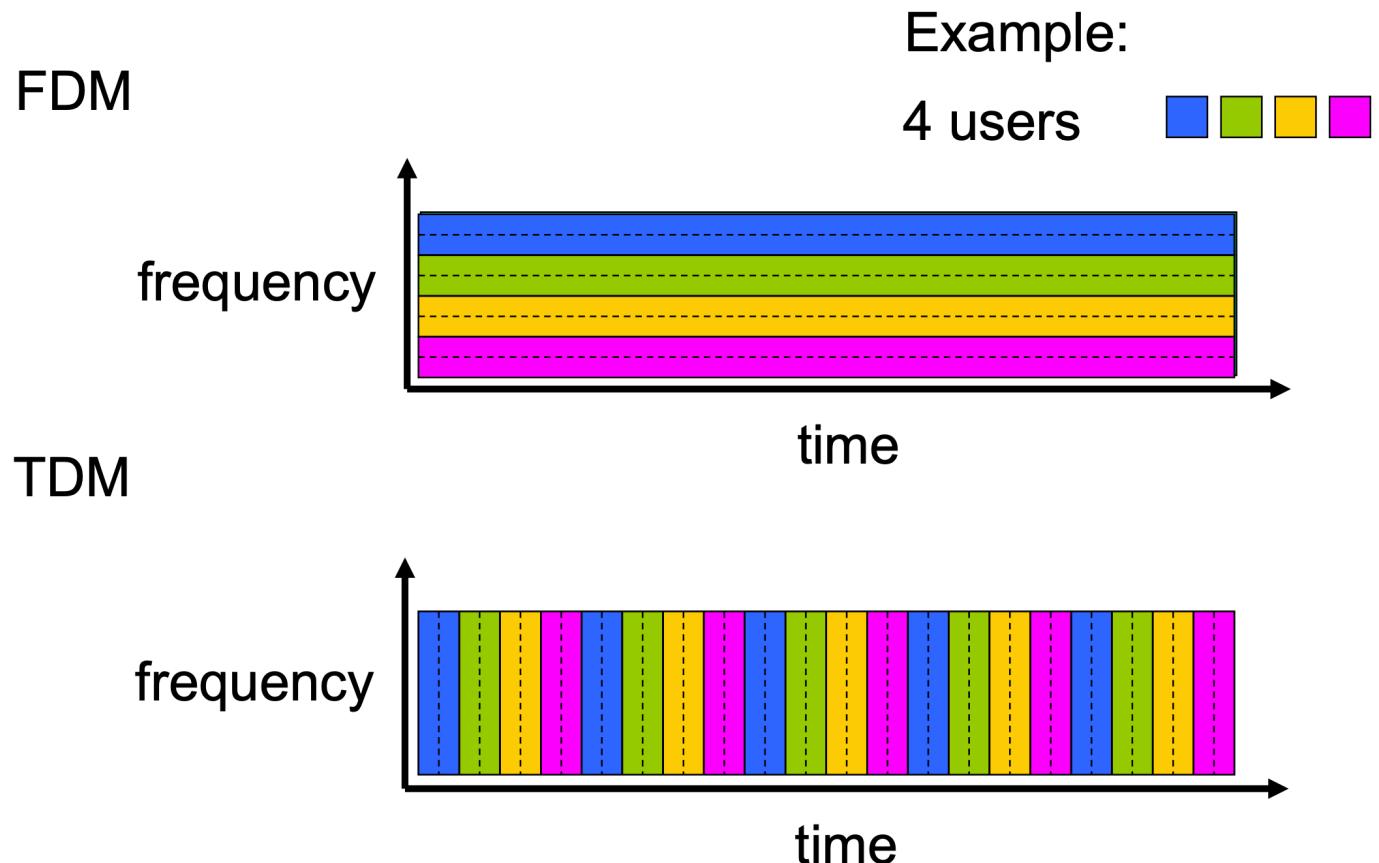
端到端的资源被分配给从源端 到目标端的呼叫 “call”:

- 图中 each link has 4 circuits: call gets 2nd circuit in top link and 1st circuit in right link
- 是 **独享资源(dedicated resources)**: no sharing, 每个呼叫一旦建立起来就能够保证性能
- 如果呼叫没有数据发送, 被分配的资源就会被浪费 (no sharing). 就像两个人打电话, 但是却沉默不语, 电信公司仍然要收钱
- 通常被 traditional telephone networks 采用



In circuit switching, 网络带宽资源会被分为一个个片(piece), 网络资源(如带宽)被分成片的两种方式:

- 频分(Frequency-Division Multiplexing): 交换接电和交换节点之间的链路带宽比较宽，两个host在通讯之前在每一条链路上找到空闲的一片
- 时分(Time-Division Multiplexing): 划分时间片的方式。每个周期的第一片，分给第一个用户使用；每个周期的第二片，分给第二个用户使用...



□ 在一个电路交换网络上，从主机A到主机B发送一个**640,000比特**的文件需要多长时间？

- 所有的链路速率为**1.536 Mbps**
- 每条链路使用时隙数为**24**的TDM
- 建立端-端的电路需**500 ms**

每条链路的速率（一个时间片）： **$1.536 \text{Mbps} / 24 = 64 \text{kps}$**

传输时间： **$640 \text{kb} / 64 \text{kps} = 10 \text{s}$**

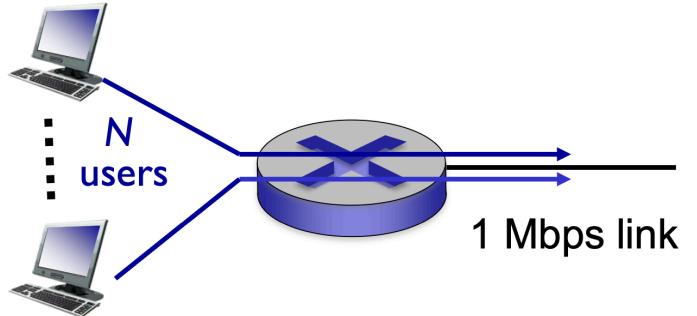
共用时间： 传输时间+建立链路时间= **$10 \text{s} + 500 \text{ms} = 10.5 \text{s}$**

Packet switching vs circuit switching

同样的网络资源，Packet switching 允许更多用户使用网络（共享性，按需分配），避免了 circuit switching 占线却并没有传输数据的浪费资源现象。适合于对突发式数据传输(great for bursty data)。但是过度使用会造成网络拥塞:分组延时和丢失，对可靠地数据传输需要协议来约束: 拥塞控制。

example:

- 1 Mb/s link
- each user:
 - 100 kb/s when “active”
 - active 10% of time



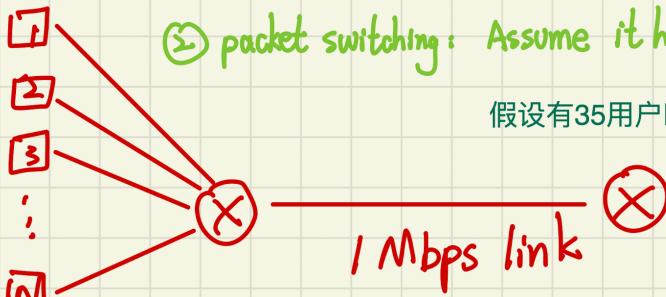
1 Mb/s link, 每个user活跃的概率是10%, 100 kb/s when “active”

问 Packet switching 和 Circuit switching 哪个支持的用户多?

$$\textcircled{1} \text{ Circuit switching: } \frac{1 \text{ Mbps}}{100 \text{ kbps}} = 10 \text{ users}$$

$$\textcircled{2} \text{ packet switching: Assume it has 35 users } P = \sum_{i=11}^{35} \binom{35}{i} 0.1^i \times 0.9^{35-i} = 0.0004$$

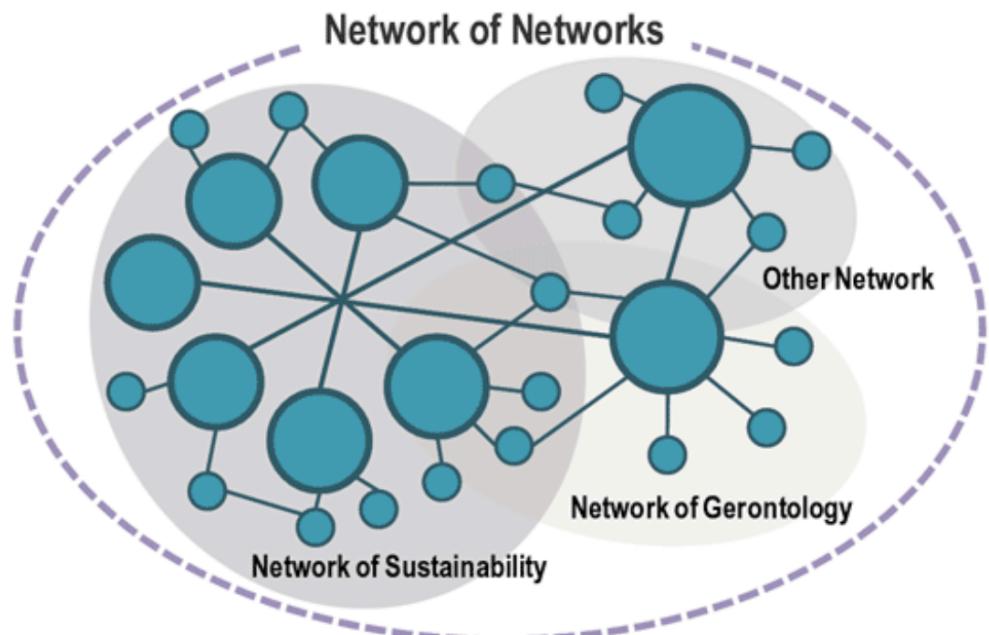
假设有35用户时, >=10个用户活动的概率为 0.0004。



1.4 Internet structure and ISP

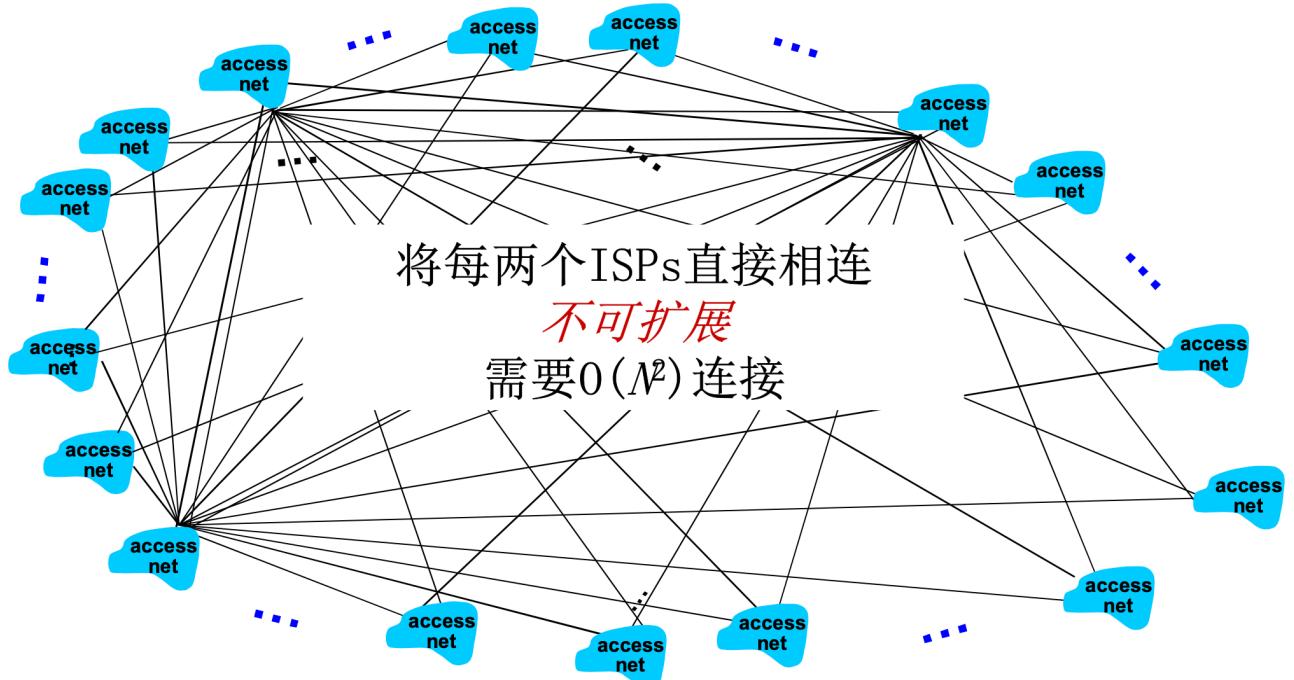
Internet structure

- 互联网结构：网络中的网络 network of networks，网络把主机连接起来，而互连网（internet）是把多种不同的网络连接起来，因此互连网是网络的网络。而互联网（Internet）是全球范围的互连网。
- **ISPs (Internet Service Providers):** 互联网服务提供商，可以从互联网管理机构获得许多 IP 地址，同时拥有通信线路以及路由器等联网设备，个人或机构向 ISP 缴纳一定的费用就可以接入互联网。目前的互联网是一种多层次 ISP 结构，ISP 根据覆盖面积的大小分为第一层 ISP、区域 ISP 和接入 ISP。互联网交换点 IXP 允许两个 ISP 直接相连而不用经过第三个 ISP。
- **End systems (端系统)** 通过接入ISPs 连接到互联网 (eg. 住宅, 公司和大学的ISPs)
- 接入ISPs相应的必须是互联的，因此任何2个端系统可相互发送分组到对方

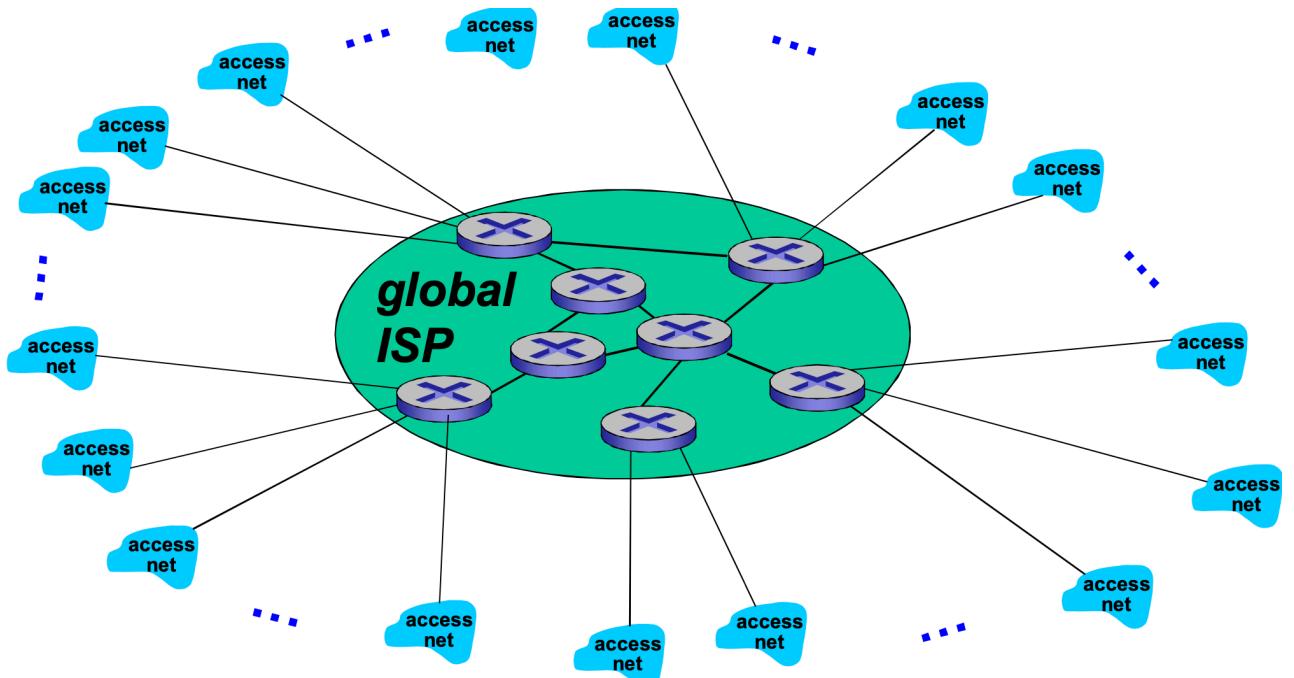


问题: 给定数百万接入ISPs, 如何将它们互联到一起。

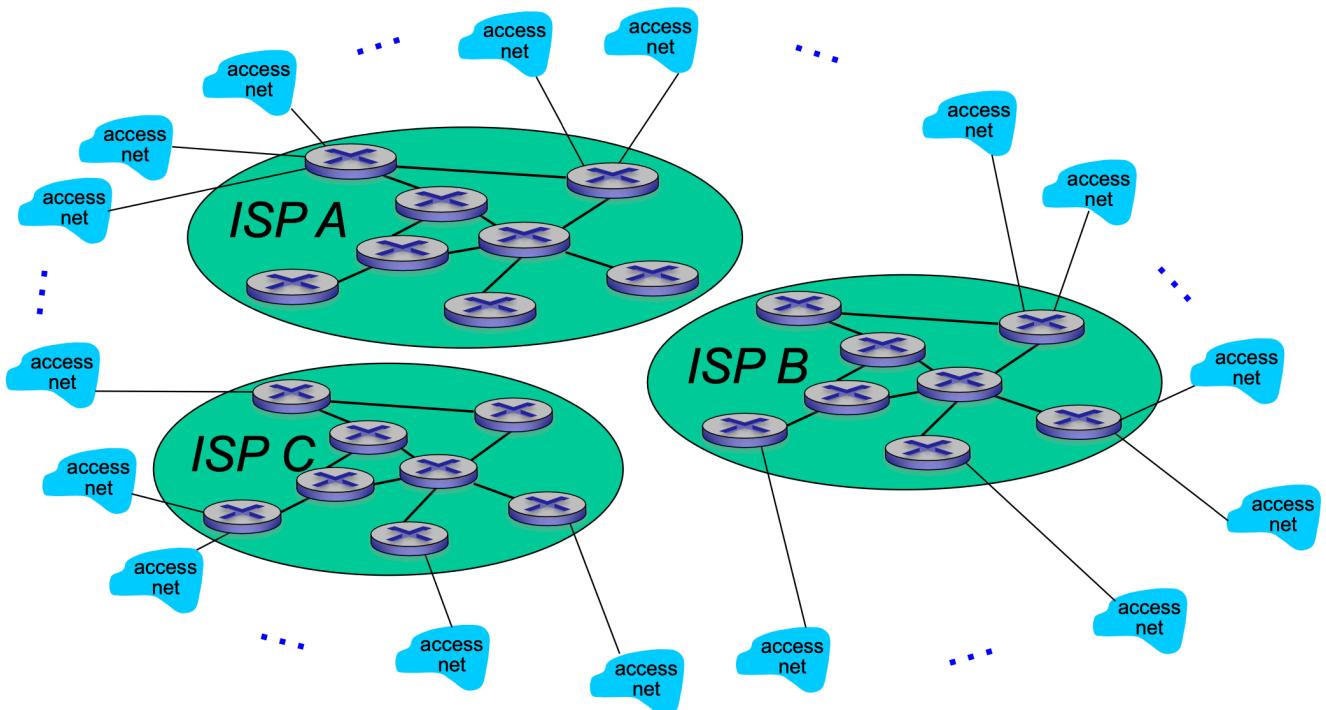
如果将每个接入ISP直接连接到彼此不能扩展(scale), 需要 $O(N^2)$ 连接。



所以我们可以将每个接入ISP都连接到全局ISP(全局范围内覆盖): 客户ISPs和提供者ISPs有经济合约, 这样当然好, 但是如果世界上不可能只有一个ISP网络, 由于有利可图, 肯定会有能做出ISP技术的人出来跟他一起竞争。

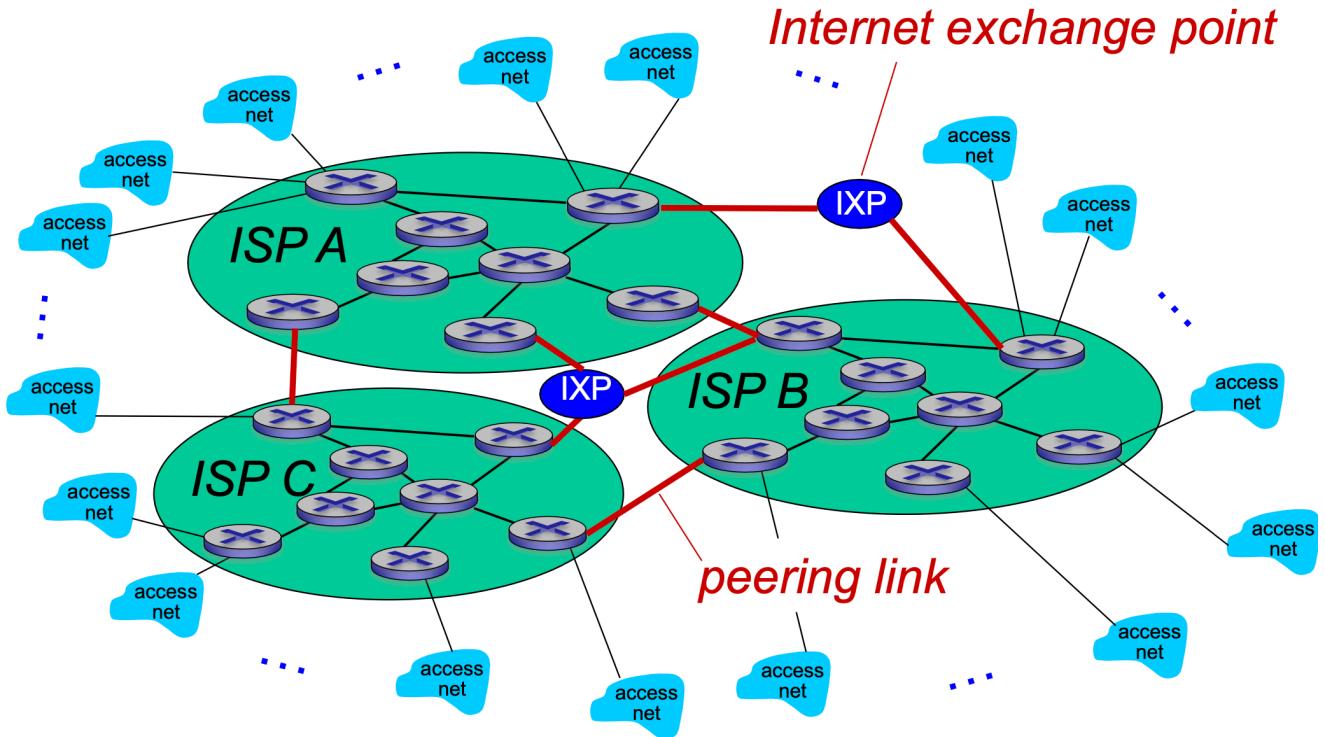


但是，如果全局ISP是可行的业务，那会有竞争者有利可图，一定会有竞争：

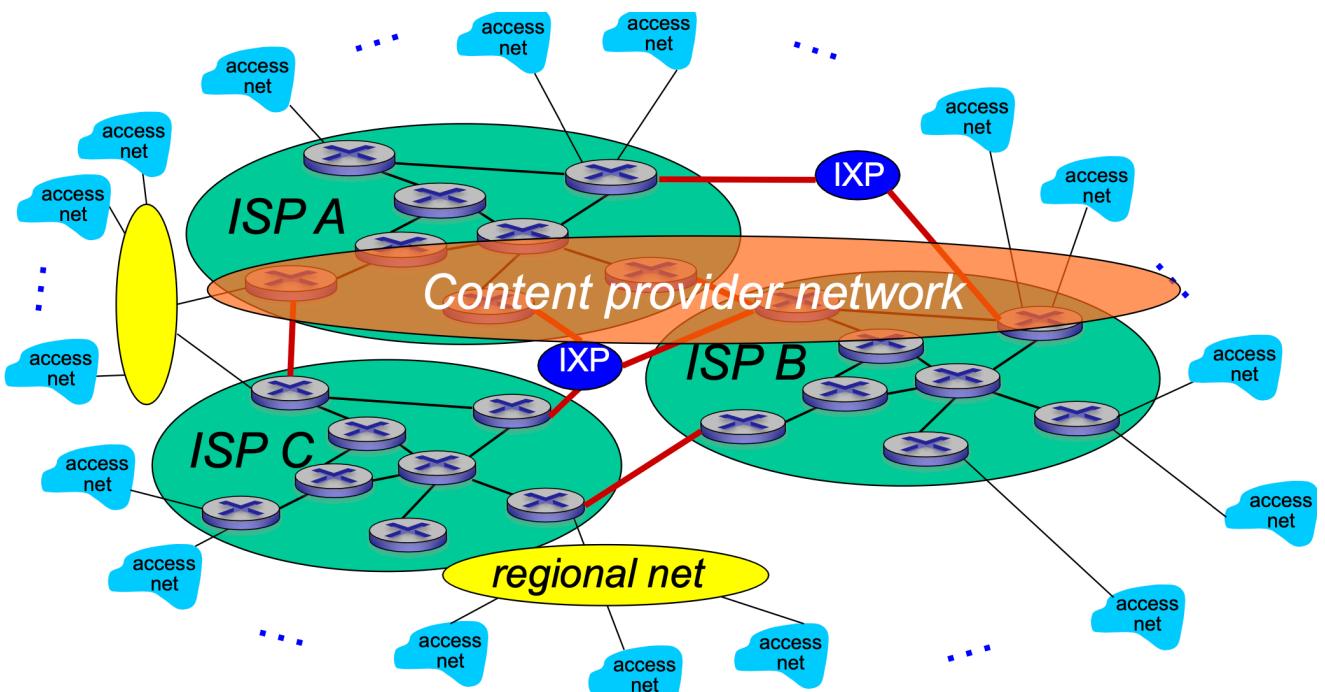


竞争:但如果全局ISP是有利可为的业务，那会有竞争者

合作: 通过ISP之间的合作可以完成业务的扩展，肯定会有互联，对等互联的结算关系。图中ISP A、B、C大家都接入了IXP网络，其中 IXP 就叫 **Internet exchange point**



Internet Content Providers (内容提供商网络): 可能会构建它们自己的网络，将它们的服务、内容更加靠近终端用户，向用户提供更好的服务。而且不用给用其他ISP网络，连接若干local ISP和各级(包括一层)ISP,更加靠近用户。搭建自己的网络就不要给其他ISP网络的运营商交钱，减少运营成本。e.g. Google, Microsoft, Akamai



ISP的三个层次和连接

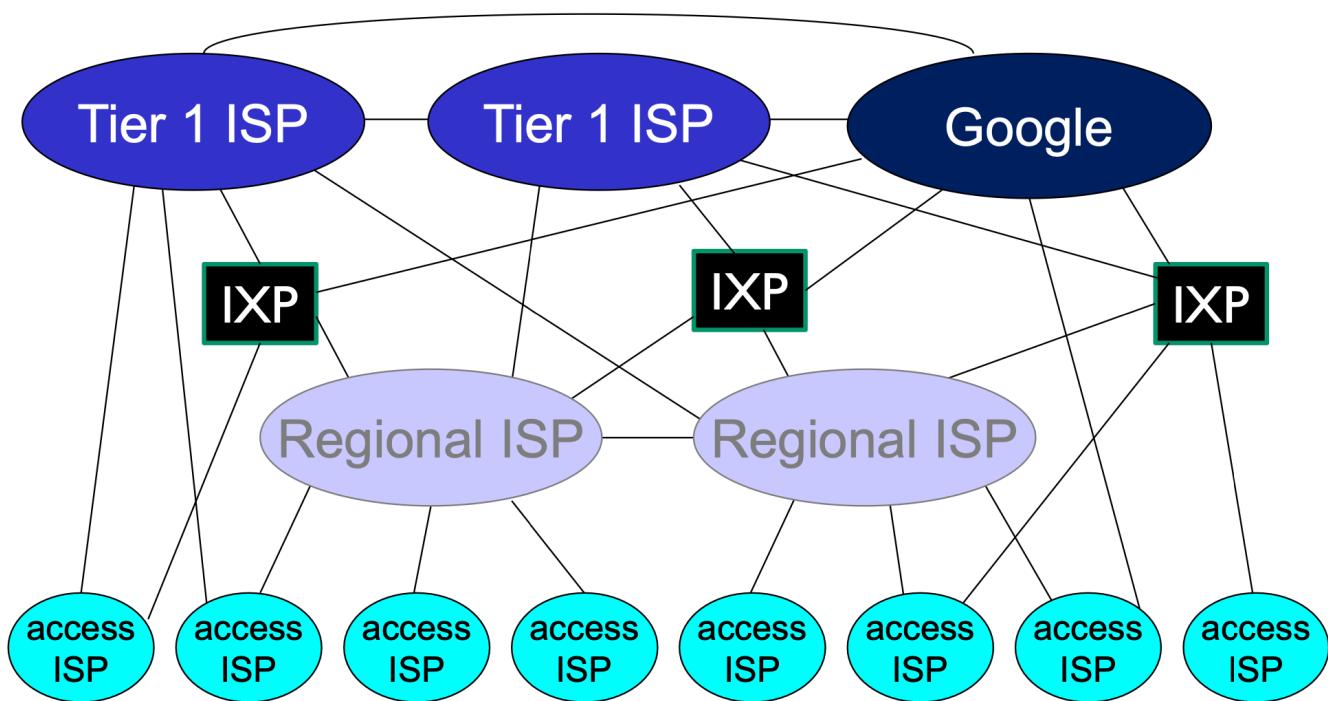
目前的互联网是一种多层次 ISP 结构，ISP 根据覆盖面积的大小分为第一层 ISP、区域 ISP 和接入 ISP。互联网交换点 IXP 允许两个 ISP 直接相连而不用经过第三个 ISP。

- **中心：第一层ISP** (如UUNet, BBN/Genuity, Sprint, AT&T) 国家/国际覆盖，速率极高。直接与其他第一层ISP相连。与大量的第二层ISP和其他客户网络相连
- **第二层ISP:** 更小些的ISP (**regional ISP**)，与一个或多个第一层ISPs，也可能与其他第二层ISP

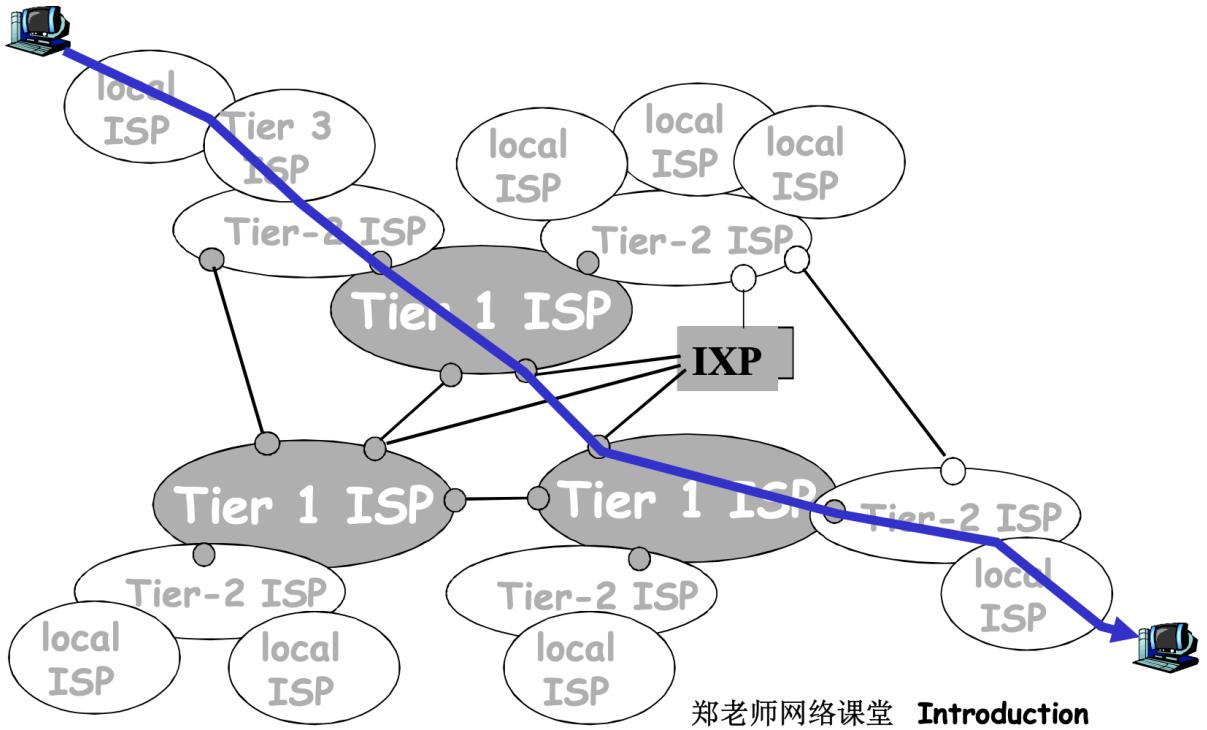
- 第三层ISP与其他本地ISP, local ISP, access net (与端系统最近)
- POP: 高层ISP面向客户网络的接入点, 涉及费用结算。如一个低层ISP接入多个高层ISP, 多宿(multi home)
- 对等接入: 2个ISP对等互接, 不涉及费用结算
- IXP:多个对等ISP互联互通之处, 通常不涉及费用结算。对等接入
- ICP自己部署专用网络, 同时和各级ISP连接

在网络的最中心, 一些为数不多的充分连接的大范围网络(分布广、节点有限、但是之间有着多重连接)

- "tier-1" commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
- content provider network (e.g., Google): 将它们的数据中心接入ISP, 方便周边用户的访问;通常私有网络之间用专网绕过第一层ISP和区域ISPs



□ 一个分组要经过许多网络！



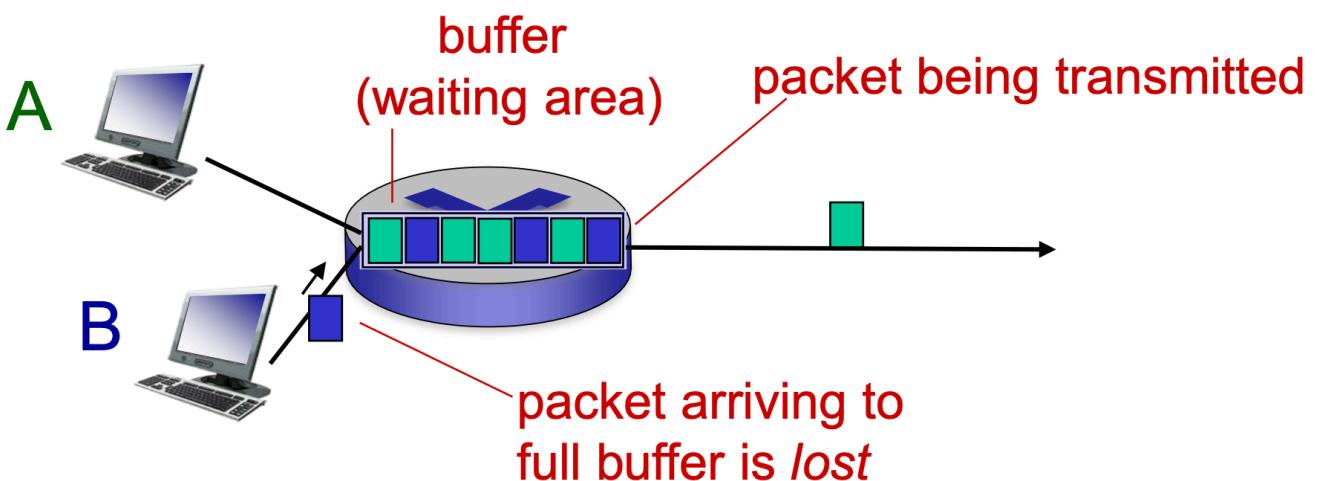
1.5 Performance: loss, delay, throughput

网络核心采用的是 packet switching, 所以可能会产生 loss and 四个delay。

loss and delay

Delay 产生的原因: 在路由器当中, 每条 link 都对应相应的队列, 如果有一个packet需要传输, 他就会先通过查路由表决定通过哪条队列, 通过其 link 往外走。如果这条 link 上, 当前没有 packet 在传输, 就可以直接传。但是如果其他的 packet 在传输, 那就必须排在队列当中。只有排到队头, 才可以被传输。

Loss 产生的原因: 队列是有限的, 如果来了一个 packet 加入这个队列, 但是队列已经满了, 这个 packet 就会被丢弃。丢失的 packet 可能会被前一个节点或源端系统重传, 或根本不重传

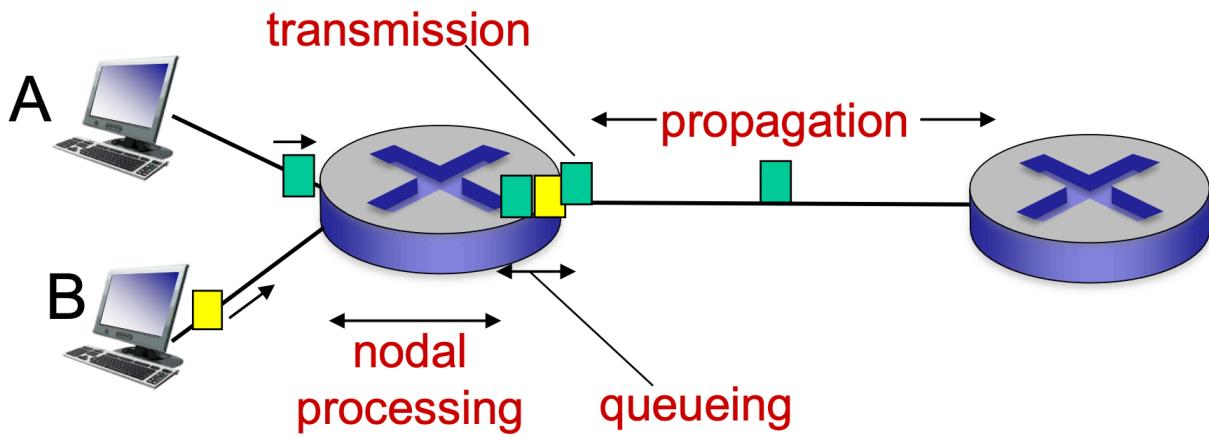


Four sources of packet delay

节点延时 nodal delay 可以分为4个部分：

名称	描述	计算公式
nodal processing (节点处理延时)	数据在路由器中处理需求的时间	NA
queueing delay (排队延时)	数据在路由器前等待前面数据处理的时间	$L: \text{packet length (bits)}$, $R: \text{link bandwidth (bps)}$, $a: \text{average packet arrival rate}$ $\text{traffic intensity (流量强度)} = La/R$ $La/R = 0: \text{平均排队延时很小}$ $La/R > 1: \text{延时变得很大}$ $La/R > 1: \text{平均排队延时将趋向无穷大}$ 所以设计系统时流量强度不能大于1
transmission delay (传输延时)	数据在信道上传播所花费的时间	L/R $L: \text{packet length (bits)}$, $R: \text{link bandwidth (bps)}$
propagation delay (传播延时)	数据从主机到信道上所用的时间	d/s $d = \text{link length}$, $s = \text{在媒体上的传播速度}$

- **nodal processing (节点处理延时):** 检查 bit 级差错，检查 packet(首部) 有没有出错和决定将分组导向何处的处理消耗的时间
- **queueing delay (排队延时):** 在输出链路上等待传输的时间，依赖于路由器的拥塞程度。
- **transmission delay (传输延时):** $R = \text{链路带宽 (bps)}$, $L = \text{分组长度 (bits)}$, 将分组发送到链路上的时间 (传输延时) = L/R , 存储转发延时
- **propagation delay (传播延时):** $d = \text{物理链路的长度}$, $s = \text{在媒体上的传播速度} (\sim 3 \times 10^8 \text{ m/sec})$, 传播延时 = d/s



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

d_{proc} : nodal processing

- check bit errors
- determine output link
- typically < msec

d_{queue} : queueing delay

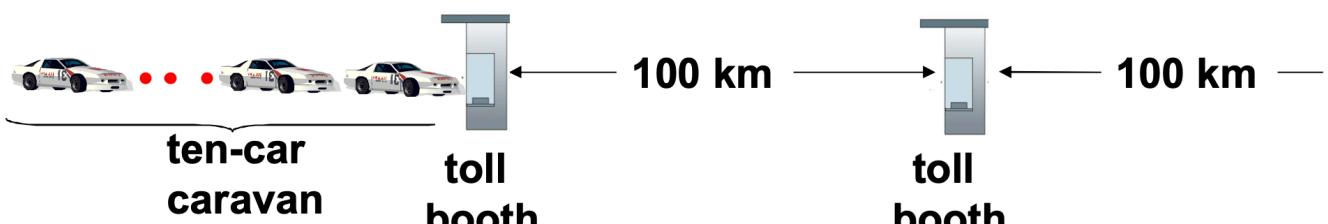
- time waiting at output link for transmission
- depends on congestion level of router

d_{trans} : transmission delay:

- L : packet length (bits)
 - R : link bandwidth (bps)
 - $d_{\text{trans}} = L/R$
- d_{trans} and d_{prop} very different

d_{prop} : propagation delay:

- d : length of physical link
- s : propagation speed ($\sim 3 \times 10^8$ m/sec)
- $d_{\text{prop}} = d/s$



- 汽车以 100 km/hr 的速度传播
- 收费站服务每辆车需 12s (传输时间)
- 汽车~bit; 车队 ~ 分组
- Q: 在车队在第二个收费站排列好之前需要多长时间?
 - 即: 从车队的第一辆车到达第一个收费站开始计时, 到这个车队的最后一辆车离开第二个收费站, 共需要多少时间
- 将车队从收费站输送到公路上的时间 = $12 \times 10 = 120\text{s}$
- 最后一辆车从第一个收费站到第二个收费站的传播时间:
 $100\text{km}/(100\text{km/hr}) = 1 \text{ hr}$
- A: 62 minutes

- 汽车以1000 km/hr 的速度传播汽车
- 收费站服务每辆车需1分钟
- Q: 在所有的汽车被第一个收费站服务之前，汽车会到达第二个收费站吗？

□ Yes! 7分钟后，第一辆汽车到达了第二个收费站，而第一个收费站仍有3辆汽车

□ 在整个分组被第一个路由器传输之前，第一个比特已经到达了第二个路由器！

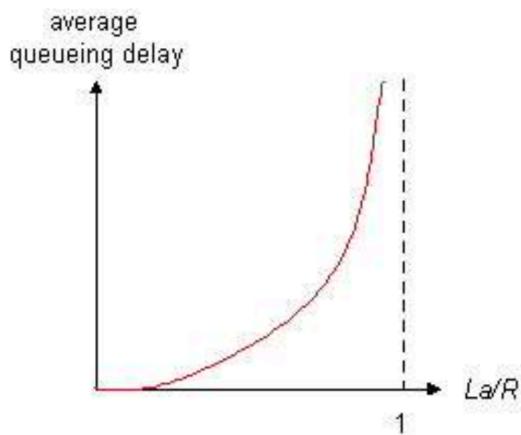
排队延时

- R=链路带宽 (bps)
- L=分组长度 (bits)
- α =分组到达队列的平均速率

流量强度 = La/R

- $La/R \sim 0$: 平均排队延时很小
- $La/R \rightarrow 1$: 延时变得很大
- $La/R > 1$: 比特到达队列的速率超过了从该队列输出的速率，平均排队延时将趋向无穷大！

设计系统时流量强度不能大于1！



$La/R \sim 0$

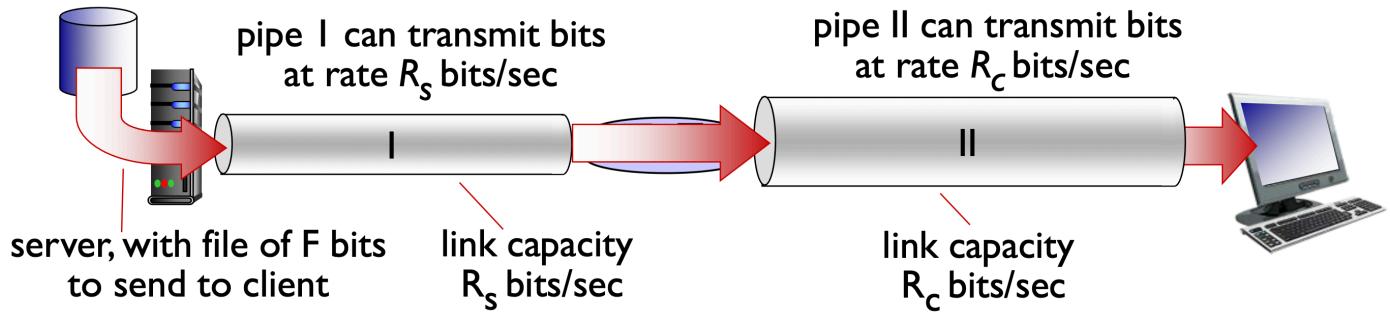


$La/R \rightarrow 1$

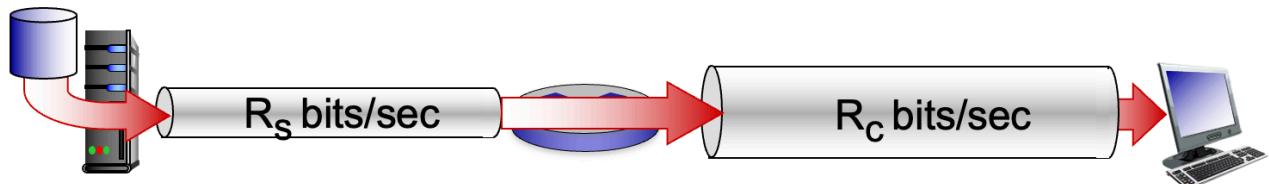
throughput 吞吐量

- 单位时间内，从原主机向目标主机发出去有效的比特的数量，在源端和目标端之间传输的速率 (bits/time unit), rate (bits/time unit) at which bits transferred between sender/receiver (end-to-end)
 - instantaneous (瞬间吞吐量): 在一个时间点的速率
 - average (平均吞吐量): 在一个长时间内平均值

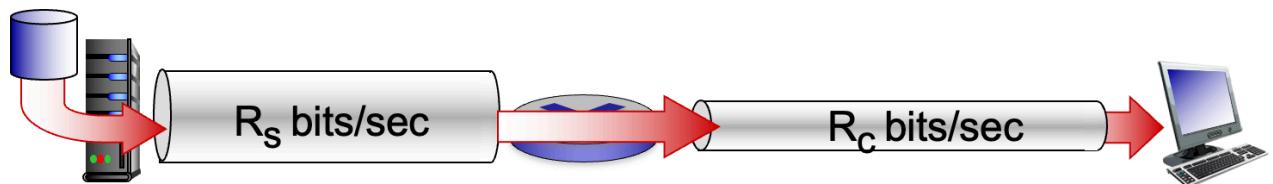
server (服务器) 发送 bits 到管道, 他的有效吞吐量取决于细的管道 (取决于第1个管道)。吞吐量 = $\min\{R_s, R_c\}$



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

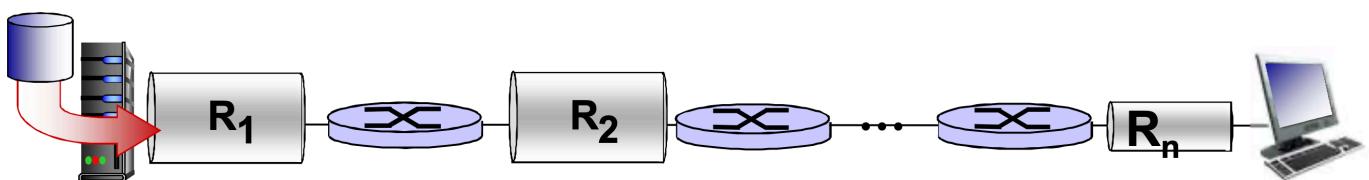


- $R_s > R_c$ What is average end-end throughput?

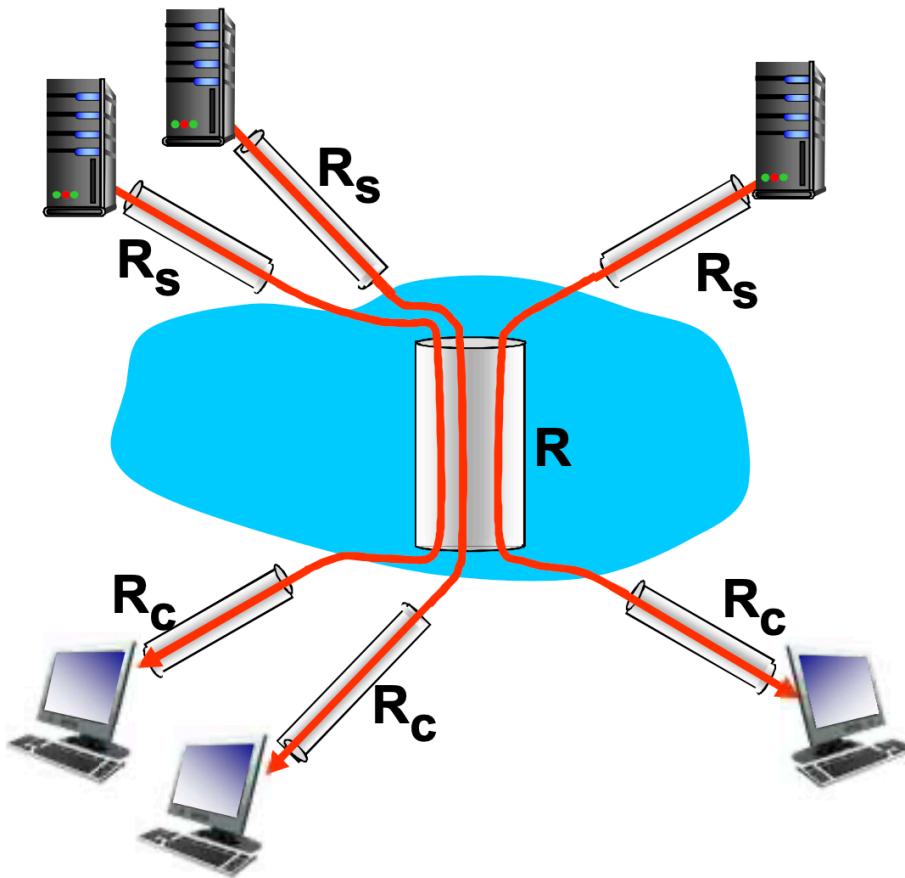


bottleneck link (瓶颈链路): 端到端路径上，限制端到端吞吐的 link (最细的link就是他的 bottleneck link 瓶颈链路)

端到端平均吞吐= $\min\{R_1, R_2, \dots, R_n\}$



在互联网场景中，我们一般都是分组交换，按需使用，这条link是共用的， n 个设备，那么每个设备的带宽将是 $1/n$ ，每个设备所获得的带宽是平均的，取决于 n 个设备。假设有10个设备，那么每个设备的带宽将是 $1/10$



10 个连接(公平地) 共享Rbps的瓶颈
链路

1.6 Protocol layers, service models

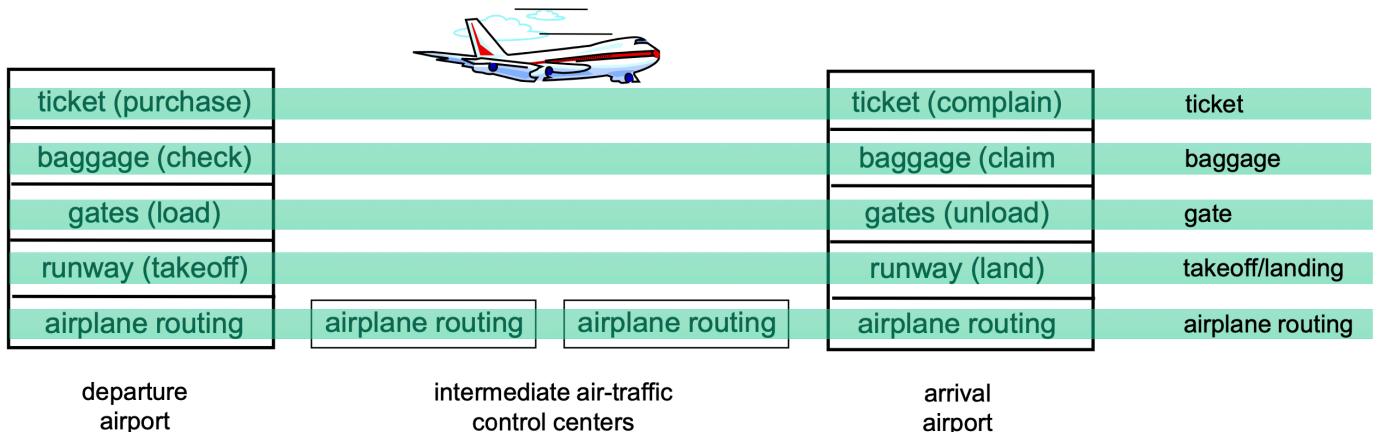
protocol layers 协议层次

- 网络是一个复杂的系统! (Networks are complex, with many "pieces": hosts, routers, links of various media, applications, protocols, hardware, software)
- 问题是：如何设计组织和实现这个复杂的计算机网络功能呢？层次化

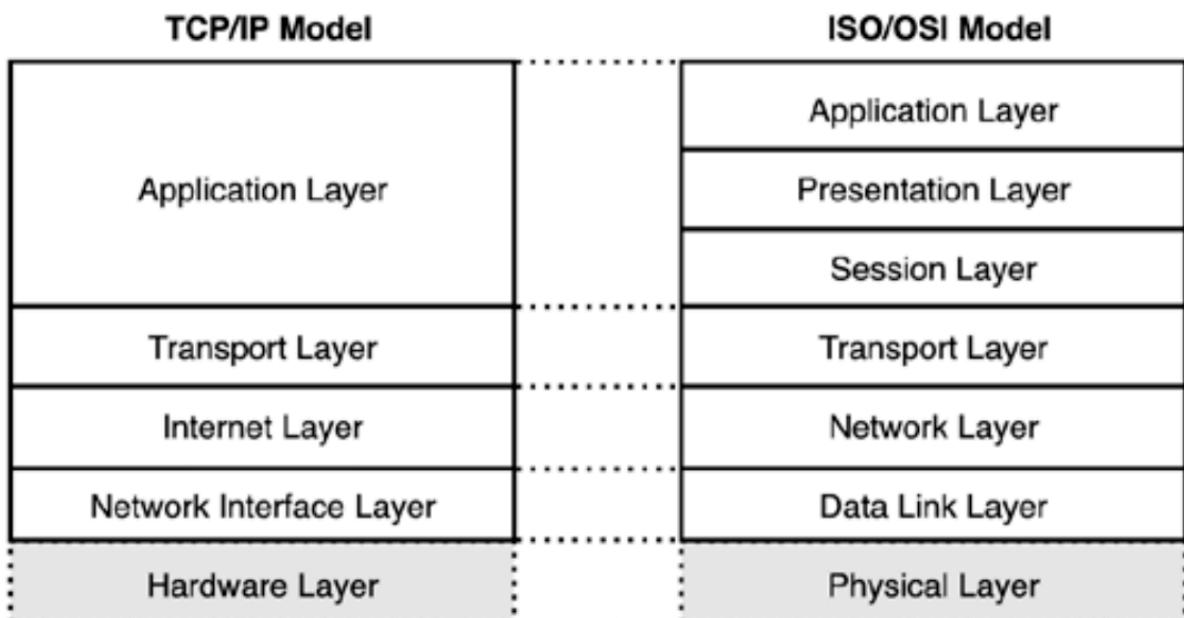
layering (层次化/分层): 层次化方式实现复杂网络功能，每一层实现一个服务(方法)each layer implements a service (function)

- 将网络复杂的功能分层功能明确的层次，每一层实现了其中一个或一组功能，功能中有其上层可以使用的功能: 服务
- 本层协议实体相互交互执行本层的协议动作，目的是实现本层功能，通过接口为上层提供更好的服务
- 在实现本层协议的时候，直接利用了下层所提供的服务
- 本层的服务: 借助下层服务实现的本层协议实体之间交互带来的新功能(上层可以利用的) + 更下层所提供的服务

就像是打仗的时候，司令官(application layer)不会直接给士兵下令，而是给团长下命令一层一层传达，直到最底层士兵层(Physical layer)就不会往下传了，他只能自己指挥自己。整个下面的层次向上提供服务，团长给师长提供的服务，当然包括连长给团长提供的服务以及自己的服务，就是每一层都包括了其底下的服务 + 自己增加到新服务。通过层连接口向上提供服务。



Service models



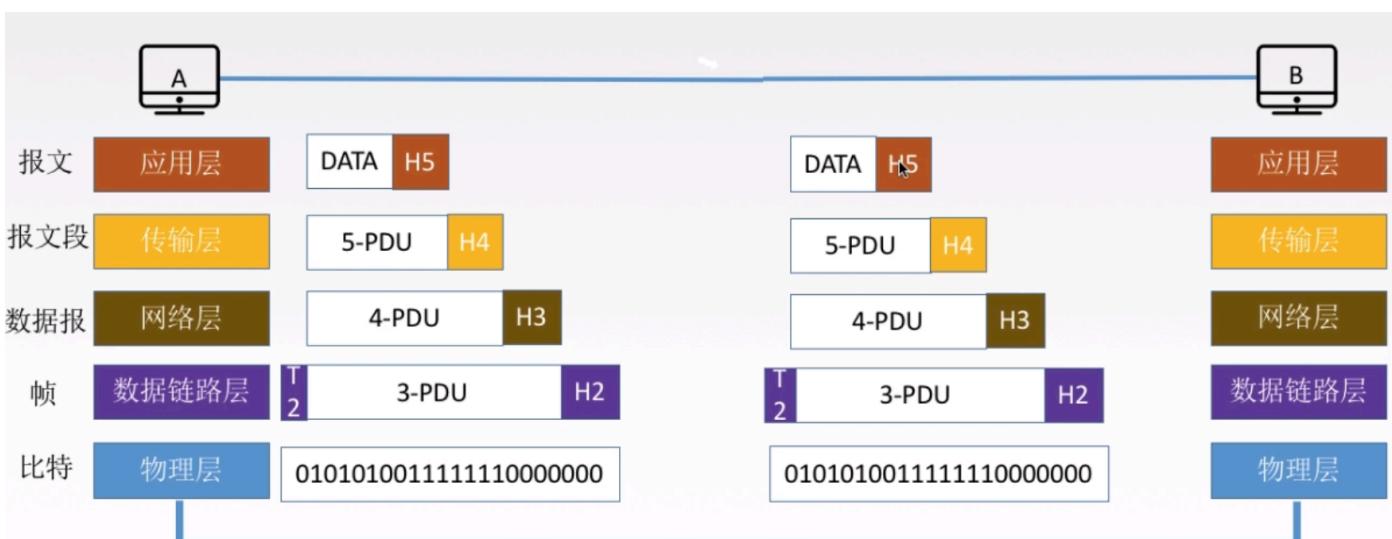
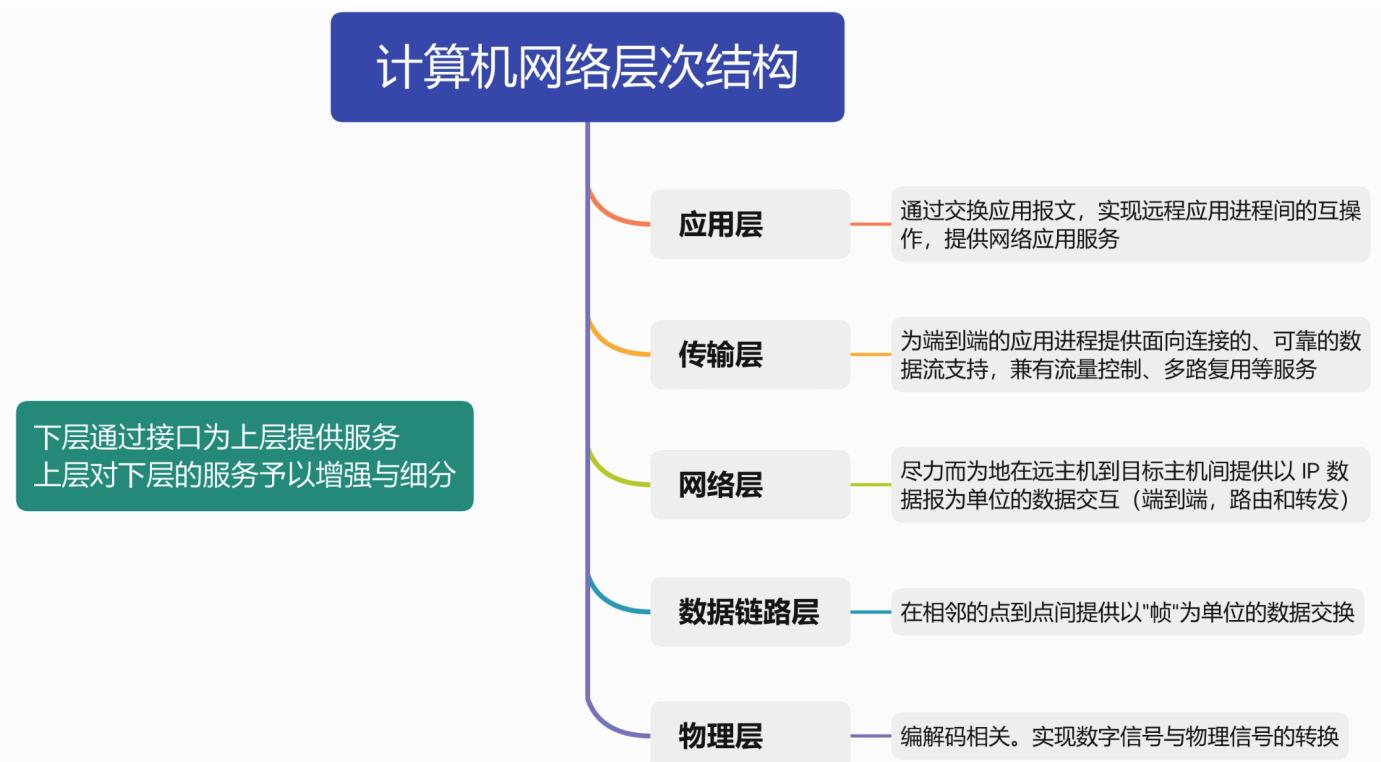
Internet protocol stack

5层参考模型

综合了OSI和TCP/IP的优点

应用层	支持各种网络应用	FTP、SMTP、HTTP
传输层	进程-进程的数据传输	TCP、UDP
网络层	源主机到目的主机的数据分组路由与转发	IP、ICMP、OSPF等
数据链路层	把网络层传下来的数据报组装成帧	Ethernet、PPP
物理层	比特传输	

计算机网络层次结构

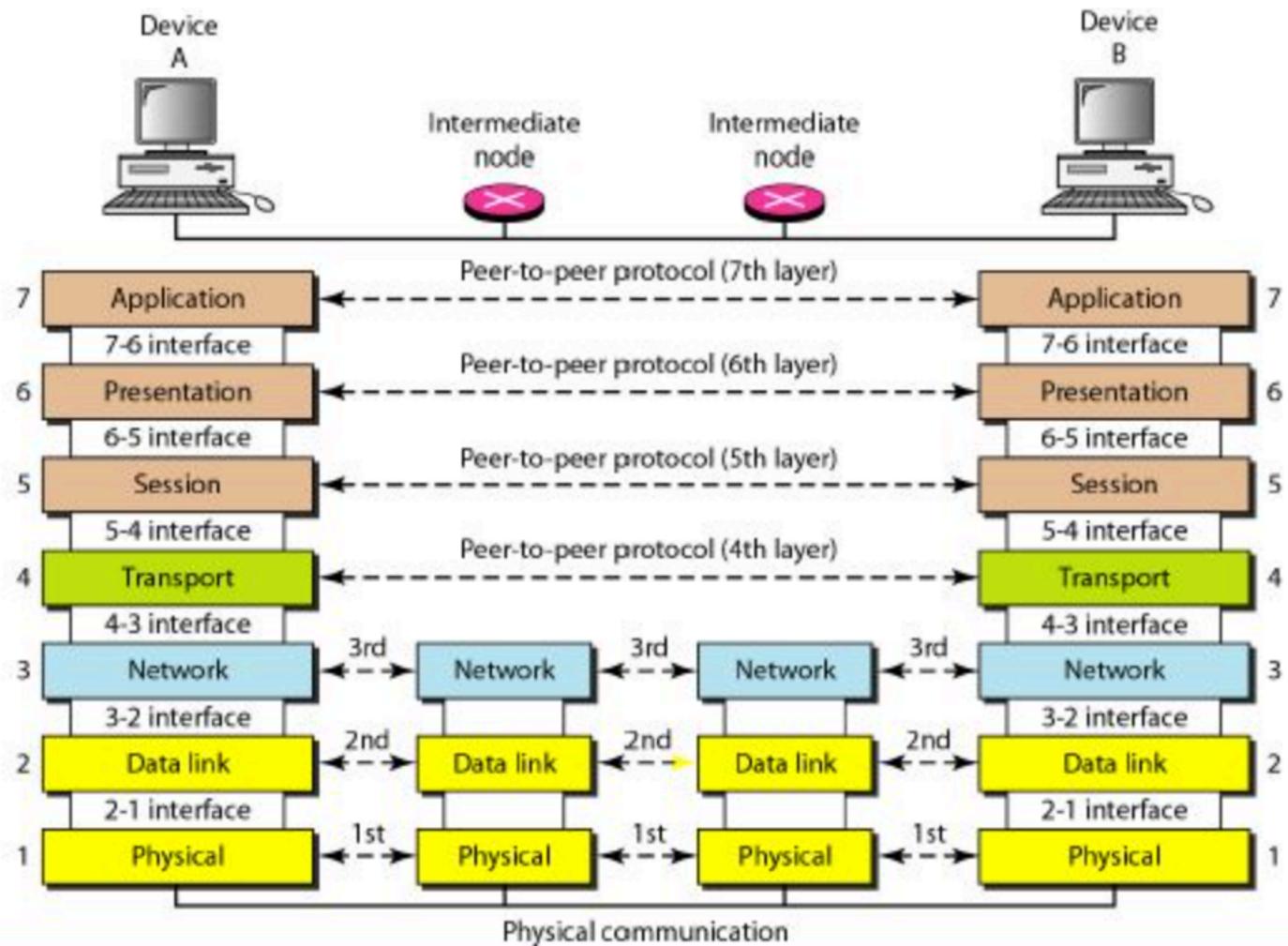


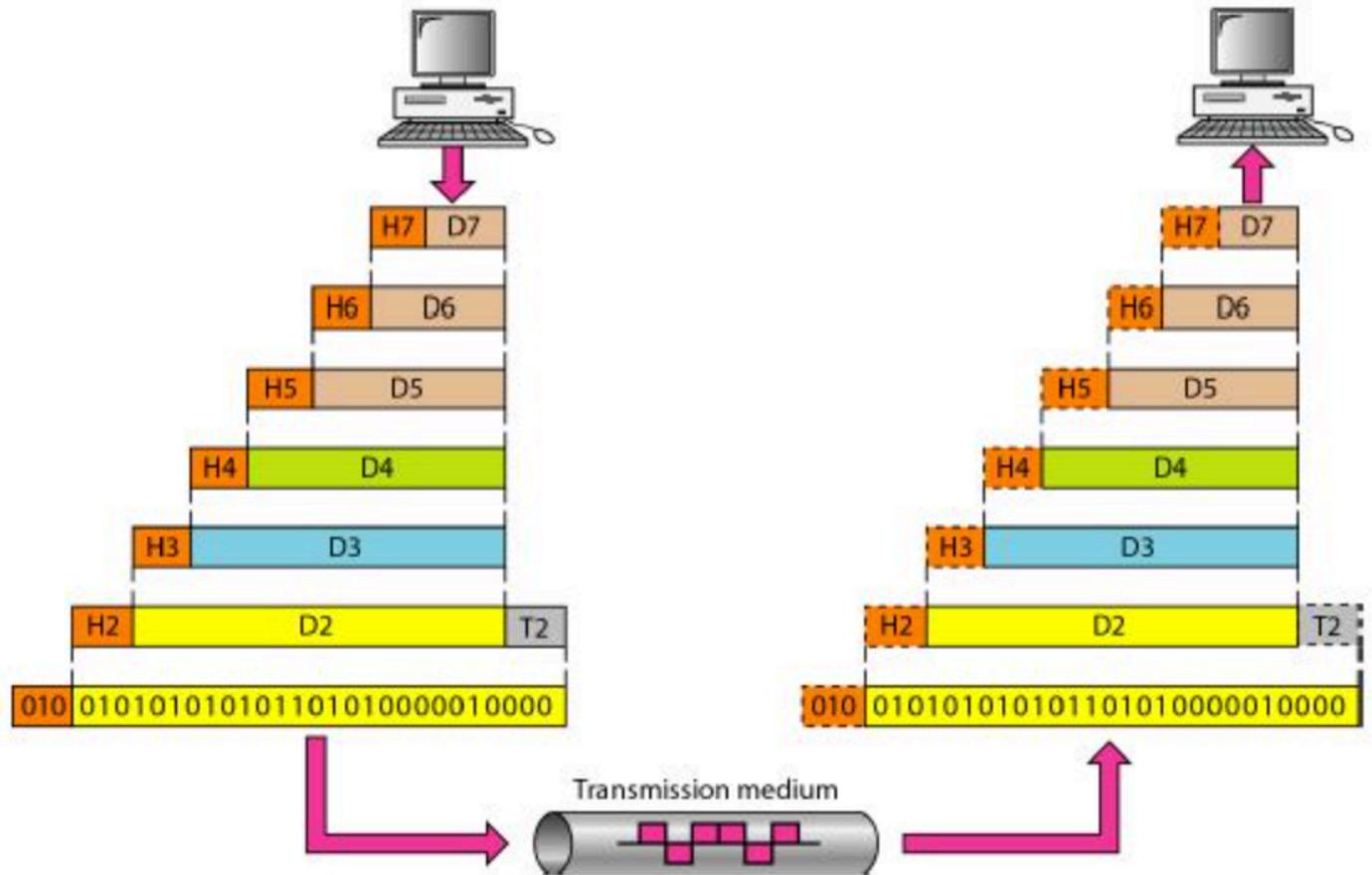
ISO/OSI reference model

- presentation (表示层):** 允许应用解释(interpret) 传输的数据, e.g., 加密, 压缩, 机器相关的表示转换
- session (会话层):** 数据交换的同步, 检查点, 恢复

Internet protocol stack (互联网协议栈) 没有这两层! 这些服务, 如果需要的话, 必须被应用实现吗?

名称	英文	作用
应用层	Application Layer	直接为用户的应用进程 (例如电子邮件、文件传输和终端仿真) 提供服务。如HTTP、SMTP、FTP、DNS等
表示层	Presentation Layer	把数据转换为能与接收者的系统格式兼容并适合传输的格式, 即让两个系统可以交换信息
会话层	Session Layer	负责在数据传输中设置和维护计算机网络中两台计算机之间的通信连接
传输层	Transport Layer	负责端到端通讯, 可靠传输, 不可靠传输, 流量控制, 复用分用
网络层	Network Layer	负责选择路由最佳路径, 规划IP地址(ipv4和ipv6变化只会影响网络层), 拥塞控制
数据链路层	Data Link Layer	帧的开始和结束, 还有透明传输, 差错校验(纠错由传输层解决)
物理层	Physical Layer	定义网络设备接口标准, 电气标准(电压), 如何在物理链路上传输的更快





Encapsulation 封装和解封装

Application (应用层): 报文(message)

Transport (传输层): 报文段(segment): TCP段, UDP数据报

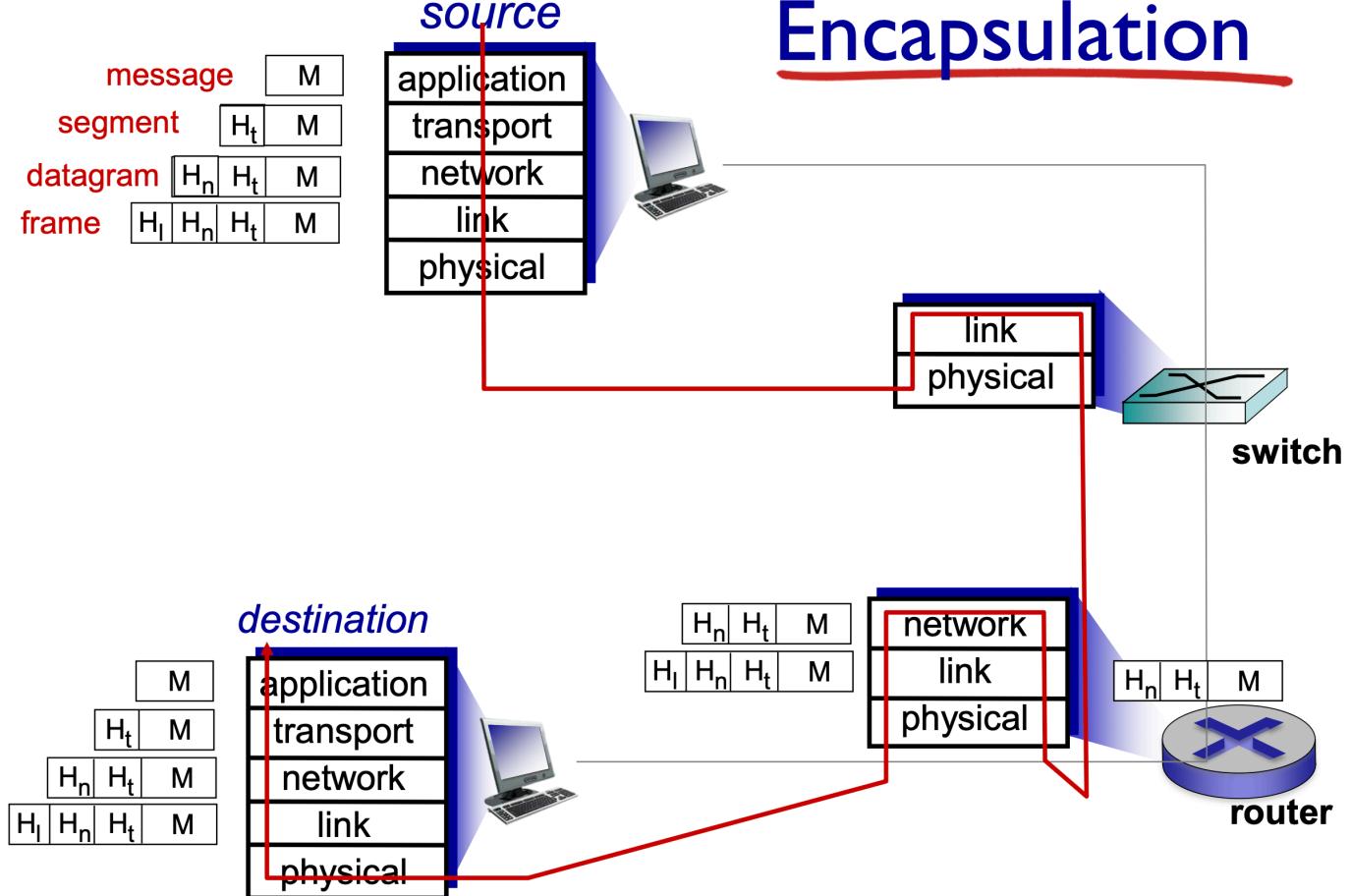
Network (网络层): 分组packet (如果无连接方式: datagram 数据报)

Link (数据链路层): 帧(frame)

Physical (物理层): 位(bit)

在原端做一个大的封装（从Application -> Physical），在目标端做一个大的解封装（从Physical -> Application）

Encapsulation

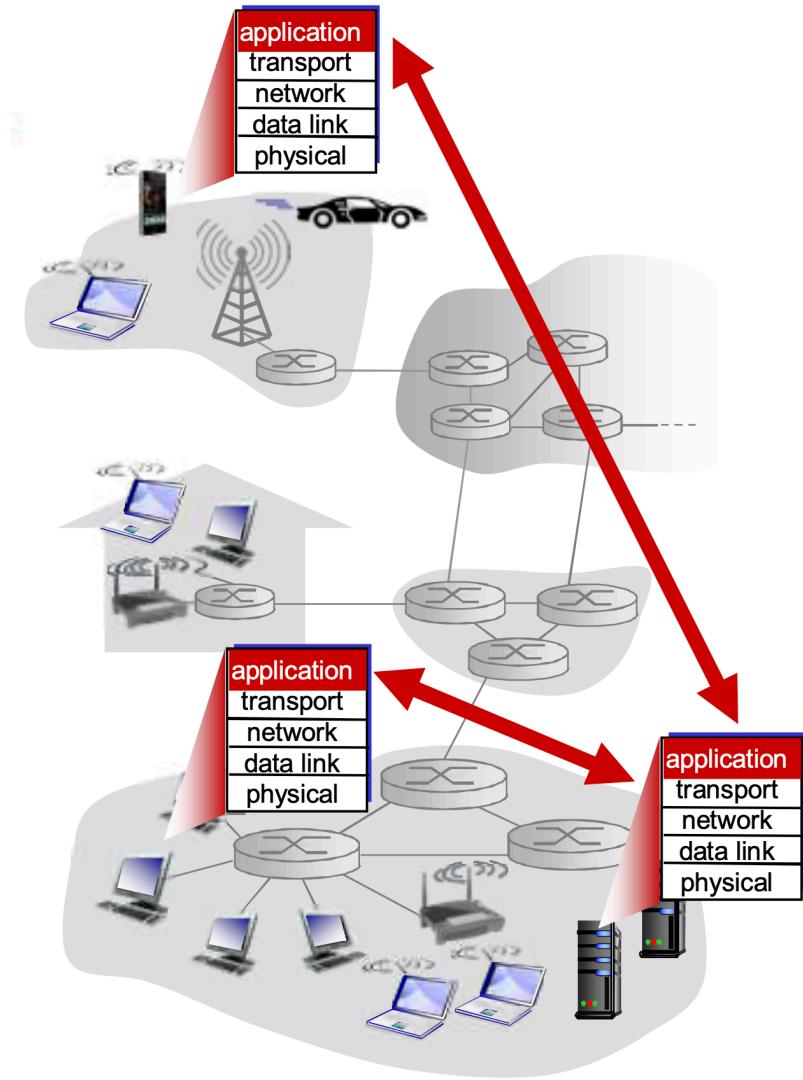


02. Application layer 交换报文，实现网络应用

一些网络应用的例子: E-mail, Web, 文本消息, 远程登录, P2P文件共享, 即时通信, 多用户网络游戏, 流媒体 (YouTube, Hulu, Netflix), 社交网络, Internet 电话, 实时电视会议, 搜索

当我们想创造一个新的网络应用应该怎么做呢?

在端系统上编程通过网络基础设施提供的服务, 应用进程彼此通信, 注意: 网络核心中没有应用层软件, 网络核心没有应用层功能, 网络应用只在端系统上存在, 快速网络应用开发和部署。



2.1 principles of network applications

在本章中，我们学习有关 principles of network applications (应用层协议原理) 和实现方面的知识。我们从定义几个关键的应用层概念开始，其中包括应用程序所需要的网络服务、客户和服务器、进程和运输层接口。我们详细考察几种网络应用程序，包括Web、电子邮件、DNS和对等文件分发。然后我们将涉及开发运行在TCP和UDP上的网络应用程序。

因此，当研发新应用程序时，你需要编写将在多台端系统上运行的软件，例如，该软件能够用C、Java或Python来编写。

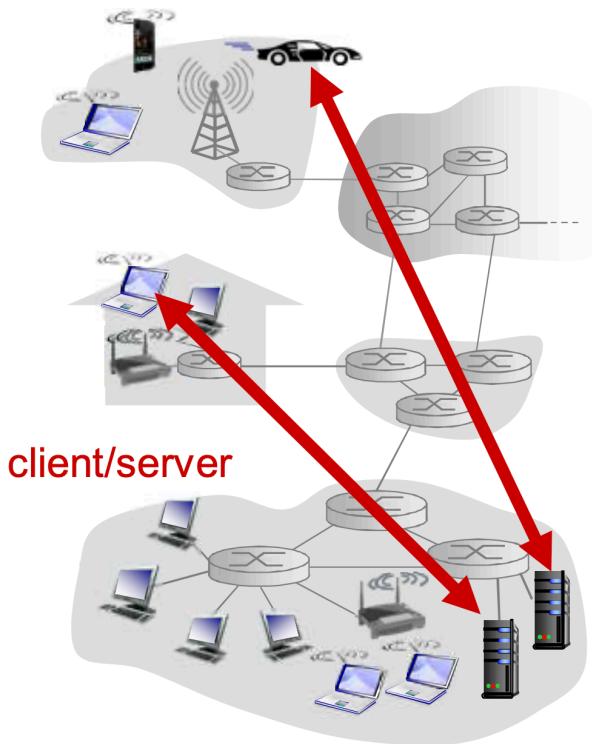
重要的是，你不需要写在网络核心设备如路由器或链路层交换机上运行的软件。换句话来讲，就是我们的软件是基于我们上城里有开发，你底层是怎么用什么协议去通信的我们根本不用管。如果是开发软件我们也只需要知道应用层的一些协议。

应用程序研发者很可能利用现代网络应用程序中所使用的两种主流体系结构之一 (possible structure of applications):

- Client-server
- peer-to-peer (P2P)

Client-server

- 在C/S体系结构中，有一个总是打开的主机称为服务器，它服务于来自许多其他称为客户的主机的请求。
- 一个经典的例子是web应用程序，其中总是打开的Web服务器服务于来自浏览器(运行在客户主机上的)的请求。当Web服务器接收到来自某客户对某对象的请求时，它向该客户发送所请求的对象作为响应。(在谷歌上面输入一个URL链接，谷歌就会返回一个页面作为响应)
- C/S体系结构的服务器具有**固定的、周知的地址**，该地址称为**IP地址** (我们的S端服务器的IP是固定的，谷歌的域名会被解析成一个IP)。具有C/S体系结构的应用程序包括Web、FTP、Telnet和电子邮件等。



□ 服务器:

- 一直运行
- 固定的**IP地址**和**周知的端口号**(约定)
- 扩展性：服务器场
 - ⑩ 数据中心进行扩展
 - ⑩ 扩展性差

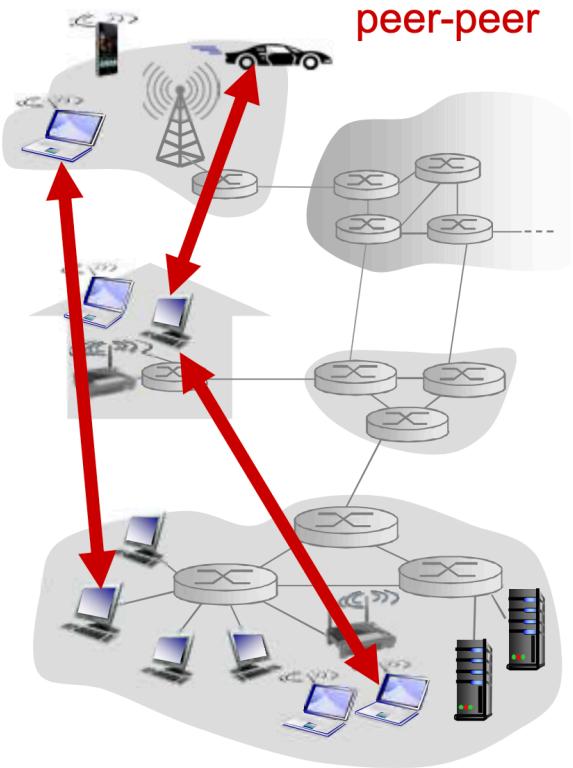
□ 客户端:

- 主动与服务器通信
- 与互联网有间歇性的连接
- 可能是动态**IP地址**
- 不直接与其它客户端通信

peer-to-peer (P2P)

- 在一个P2P体系结构中，对位于数据中心的专用服务器几乎没有依赖。相反，应用程序在间断连接的主机对之间使用直接通信，这些主机对被称为对等方(peer-peer)。P2P体系结构的最引人入胜的特性之一是它们的**自扩展性(self scalability): new peers bring new service capacity, as well as new service demands.** 新的同行带来了新的业务容量，也带来了新的业务需求。
- P2P架构的应用也有客户端进程和服务器进程之分
- 但是他的数据安全性没有CS模式好。

- (几乎) 没有一直运行的服务器
- 任意端系统之间可以进行通信
- 每一个节点既是客户端又是服务器
 - 自扩展性-新peer节点带来新的服务能力，当然也带来新的服务请求
- 参与的主机间歇性连接且可以改变IP地址
 - 难以管理
- 例子: **Gnutella**, 迅雷



Inter-process communication 进程通信

process: a program running in a host

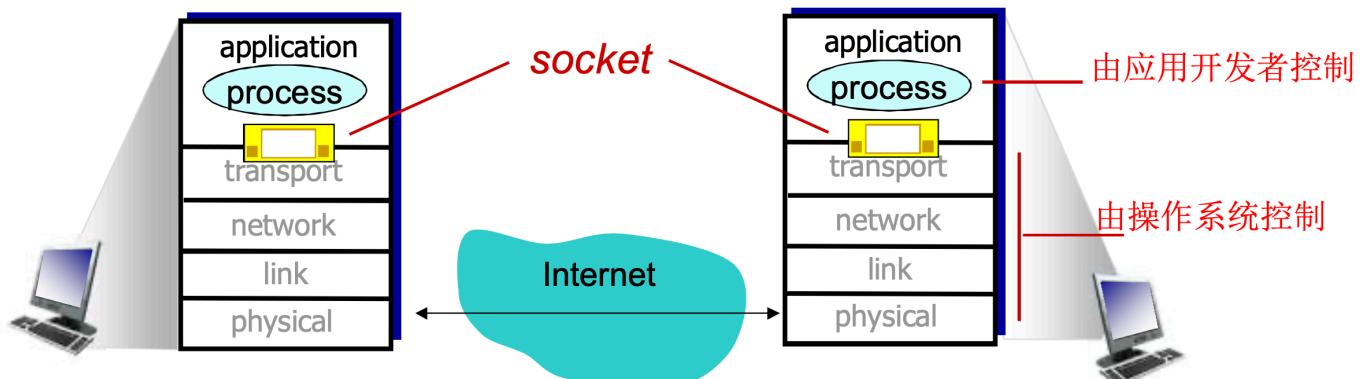
within same host, two processes can communicate using **inter-process communication** (defined by OS)

不同主机，通过交换报文(Message)来通信。对每对**Inter-process communication** (进程通信)，我们通常将这两个进程之一标识为客户端(client)，而另一个进程标识为服务器(server):

- **client process** (客户端进程): 发起通信的进程
- **server process** (服务器进程): 等待连接，然后提供服务的进程

多数应用程序是由通信进程对组成，每对中的两个进程互相发送**message**。从一进程向另一个进程发送的报文必须通过下层的网络。进程通过一个称为 **套接字(socket)** 的软件接口向网络发送报文和从网络接收报文 (process sends/receives messages to/from its socket 进程向 socket 发送报文或从 socket 接收报文)

在这个图，为我们这个程序试运行在我们的应用层。应用程式由应用开发者控制。应用层之下，都是由操作系统控制。我们想要发送数据和接收数据，只能通过socket api来进行。socket 就像一个门一样，我们想要传输信息直接把数据放到这个门里面，然后对方想要接收信息就从这个门里面接收信息



Addressing processes

进程为了接收 message，必须有一个 **identifier(标识)**。那么怎么寻址一个进程呢？

process identifier includes both IP address and port number associated with process on host

- 找主机地址：在因特网中，主机由其 **32 位的 IP address** 且它能够唯一地标识该主机就够了。
- 找目的地端口号 (**port number**)：除了知道报文送往目的地的**主机地址**外，发送进程还必须指定运行在接收主机上的**接收进程(接收socket)**。因为一般而言一台主机能够运行许多网络应用，这些信息是需要的。**目的地端口号 (port number)**用于这个目的。已经给流行的应用分配了特定的端口号 (eg, HTTP server: 80, mail server: 25)

What transport service does an app need?

应用需要传输层提供什么样的服务？如何描述传输层的服务？不同的应用需要不同的传输服务

different apps require different transport services

application	data loss	bandwidth	delay sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's ms
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's ms
text messaging	no loss	elastic	yes and no

- **data loss** 数据丢失率
 - 有些应用则要求100%的可靠数据传输(如文件)
 - 有些应用(如音频)能容忍一定比例以下的数据丢失
- **delay sensitive** 延迟敏感度
 - 有些应用程序(如互动游戏)需要低延迟才能使用
 - 其他一些应用程序(如电子邮件)并不关心
- **bandwidth** 吞吐
 - 一些应用(如多媒体)必须需要最小限度的吞吐，从而使得应用能够有效运转
 - 一些应用能充分利用可供使用的吞吐(弹性应用)

Internet transport layer services

Internet 传输层提供的服务

TCP 服务:

- 可靠的传输服务
- 流量控制：发送方不会淹没接受方
- 拥塞控制：当网络出现拥塞时，能抑制发送方
- 不能提供的服务：时间保证、最小吞吐保证和安全
- 面向连接：要求在客户端进程和服务器进程之间建立连接

UDP 服务:

- 不可靠数据传输
- 不提供的服务：可靠，流量控制、拥塞控制、时间、带宽保证、建立连接

Q: 为什么要有 UDP?

UDP存在的必要性

- 能够区分不同的进程，而IP服务不能，在IP提供的主机到主机端到端功能的基础上，区分了主机的应用进程
- 无需建立连接，省去了建立连接时间，适合事务性的应用
- 不做可靠性的工作，例如检错重发，适合那些对实时性要求比较高而对正确性要求不高的应用。因为为了实现可靠性(准确性、保序等)，必须付出时间代价(检错重发)
- 没有拥塞控制和流量控制，应用能够按照设定的速度发送数据。而在TCP上面的应用，应用发送数据的速度和主机向网络发送的实际速度是不一致的，因为有流量控制和拥塞控制

Internet apps: application / transport layer protocols

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

2.2 Web and HTTP

Web

- Web 是一种应用，http是支持Web应用的协议
- Web page consists of base HTML-file which embeds several referenced objects. object can be HTML file, JPEG image, Java applet, audio file,... Web 页面 (Web page) (也叫文档)是由对象组成的。一个对象 (object) 只是一个文件，诸如一个HTML文件、一个JPEG图形、一个Java小程序或一个视频片段这样的文件，且它们可通过一个URL地址寻址。多数Web页面含有一个HTML基本文件 (baseHTML file) 以及几个引用对象。例如，如果一个Web页面包含HTML文本和5个JPEG图形，那么这个 Web 页面有 6 个对象: 一个 HTML 基本文件加 5 个图形。
- **URL: Uniform Resource Locator 通用资源定位**

一些术语

- **Web页：**由一些**对象**组成
- 对象可以是**HTML**文件、**JPEG**图像、**Java**小程序、声
音剪辑文件等
- **Web**页含有一个**基本的HTML文件**，该基本**HTML**文
件又包含若干对象的引用（链接）
- 通过**URL**对每个对象进行引用
 - 访问协议，用户名，口令字，端口等；
- **URL格式：**

Prot://user:psw@www.someSchool.edu/someDept/pic.gif:port

协议名 用户:口令 主机名 路径名 端口

HTTP

HTTP: hypertext transfer protocol (超文本传输协议) 文本与文本之间任意指向的关系

Web的应用层协议

client/server model (客户/服务器模式):

- **client:** 请求、接收和显示 Web对象的浏览器 (**request**)
- **server:** 对请求进行响应，发送对象的Web服务器 (**response**)

HTTP: hypertext transfer protocol

- Web's application layer protocol
- two types of messages: request, response

use TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is “stateless”

- server maintains no information about past client requests



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

ation Layer

17

Non-persistent HTTP & Persistent HTTP

在许多因特网应用程序中，客户和服务器在一个相当长的时间范围内通信，其中客户发出一系列请求并且服务器对每个请求进行响应。

依据应用程序以及该应用程序的使用方式，这一系列请求可以以规则的 间隔周期性地 或者 间断性地 一个按一个发出。当这种客户-服务器的交互是经TCP进行的，应用程序的研制者就需要做一个重要决定，即每个请求/响应对是经一个单独的TCP连接发送，还是所有的请求及其响应经相同的TCP连接发送呢？

即每个请求/响应对是经一个单独的 TCP 连接发送，还是所有的请求及其响应经相同的 TCP 连接发送？

- 每个请求/响应对是经一个单独的 TCP 连接发送 就是 Non-persistent HTTP
- 所有的请求及其响应经相同的 TCP 连接发送 就是 Persistent HTTP

HTTP既能够使用非持续连接，也能够使用持续连接：

非持久HTTP

- 最多只有一个对象在 TCP连接上发送
- 下载多个对象需要多个TCP连接
- HTTP/1.0使用非持久连接

持久HTTP

- 多个对象可以在一个(在客户端和服务器之间的) TCP连接上传输
- HTTP/1.1 默认使用持久连接

Non-persistent HTTP

往返时间RTT(round-trip time): 一个小的分组从客户端到服务器，在回到客户端的时间(传输时间忽略)

这引起浏览器在它和 Web 服务器之间发起一个 TCP 连接;这涉及一次“三次握手”过程，

即客户向服务器发送一个小 TCP 报文段，服务器用一个小 TCP 报文段做出确认和响应，最后，客户向服务器返回确认。三次握手中前两个部分所耗费的时间占用了一个 RTT。完成了三次握手的前两个部分后，客户结合三次握手的第三部分(确认)向该 TCP 连接发送一个 HTTP 请求报文。一旦该请求报文到达服务器，服务器就在该 TCP 连接上发送 HTML文件。该 HTTP 请求/响应用去了另一个 RTT。因此，粗略地讲，总的响应时间就是两个 RTT 加上服务器传输 HTML 文件的时间。

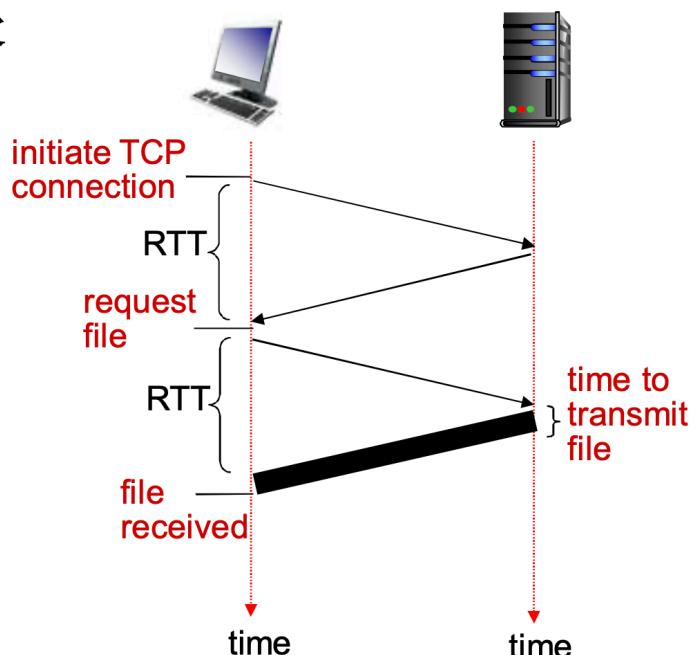
往返时间RTT (round-trip time)

：一个小的分组从客户端到服务器，在回到客户端的时间（传输时间忽略）

响应时间：

- 一个RTT用来发起TCP连接
- 一个 RTT用来HTTP请求并等待HTTP响应
- 文件传输时间

共: 2RTT+传输时间



non-persistent HTTP response time = 2RTT + file transmission time

缺点：

- 每个对象要2个 RTT
- 操作系统必须为每个TCP连接分配资源。（必须为每一个请求的对象建立和维护一个全新的连接。对于每个这样的连接，在客户和服务器中都要分配TCP的缓冲区和保持TCP变量，这给Web服务器带来了严重的负担，因为一台Web服务器可能同时服务于数以百计不同的客户的请求。）

Persistent HTTP

- 服务器在发送响应后保持连接打开。
- 同一客户端/服务器之间的后续HTTP消息通过开放连接发送。
- 客户端在遇到引用对象时立即发送请求。
- 对于所有引用的对象，只需一个RTT。

HTTP message format

HTTP 规范 [RFC 1945; RFC 2616; RFC 7540] 包含了对 HTTP 报文格式的定义。

HTTP 报文有两种：请求报文和响应报文。

Method type

put一般是修改，get查询，post增加/提交，delete删除

HTTP/1.0

- GET
- POST
- HEAD
 - 要求服务器在响应报文中不包含请求对象 → 故障跟踪

HTTP/1.1

- GET, POST, HEAD
- PUT
 - 将实体主体中的文件上传到URL字段规定的路径
- DELETE
 - 删除URL字段规定的文件

HTTP request message format

Get 请求报文一般没有实体行

□ HTTP请求报文:

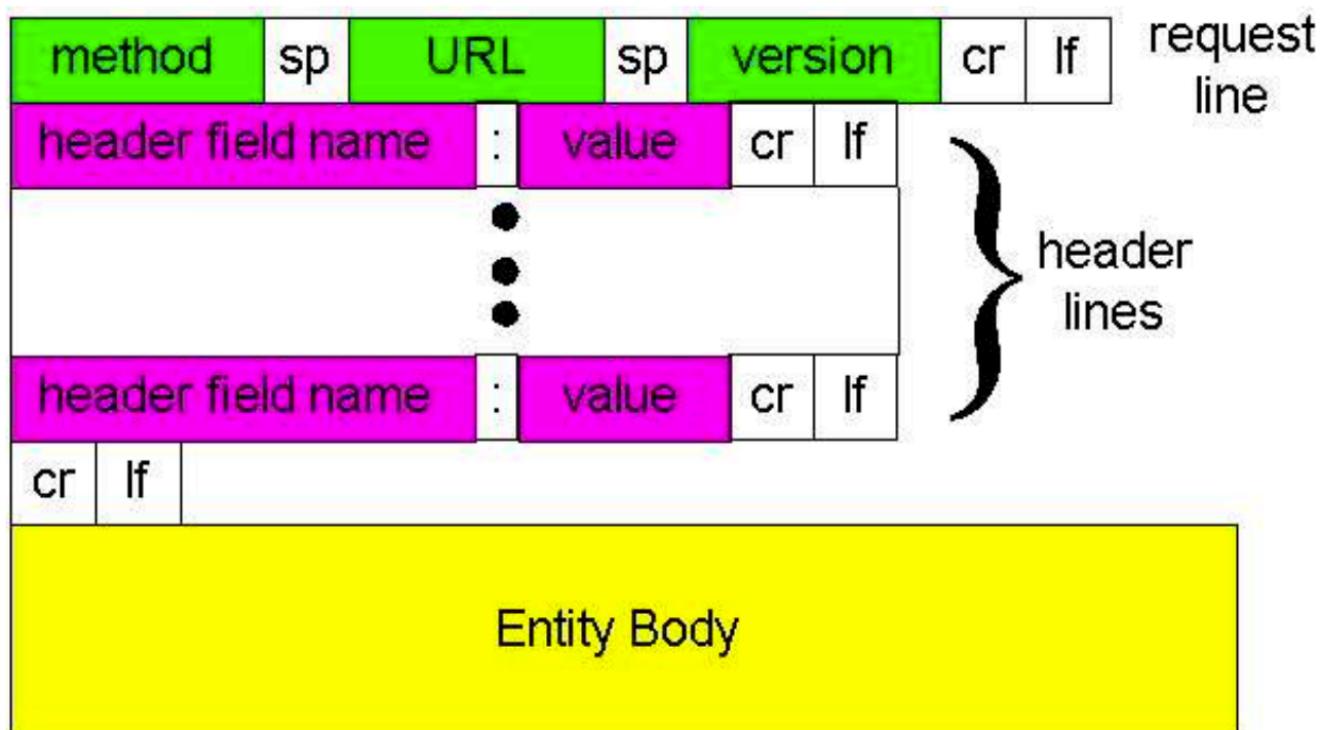
○ ASCII (人能阅读)

请求行 (GET, POST, HEAD命令)

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

换行回车符,
表示报文结束

(一个额外的换行回车符)



Uploading form input:

- **Post:** 使用 POST 方法时使用该实体体。当用户提交表单时, HTTP客户常常使用 POST 方法, 例如当用户向搜索引擎提供搜索关键词时。使用 POST 报文时, 用户仍可以向服务器请求一个web 页面, 但web页面的特定内容依赖于用户在表单字段中输入的内容。如果方法字段的值为 POST 时, 则实体体中包含的就是用户在表单字段中的输入值。
- **Get:** 使用 GET 方法时实体体 (entily body) 为空

Post方式:

- 网页通常包括表单输入
- 包含在实体主体(**entity body**)中的输入被提交到服务器

URL方式:

- 方法: GET
- 输入通过请求行的 URL 字段上载

www.somesite.com/animalsearch?monkeys&banana

<http://www.baidu.com/s?wd=xx+yy+zzz&cl=3>

参数: wd, cl

参数值: XX+YY+zzz, 3

Application Layer 3-42

HTTP response message format

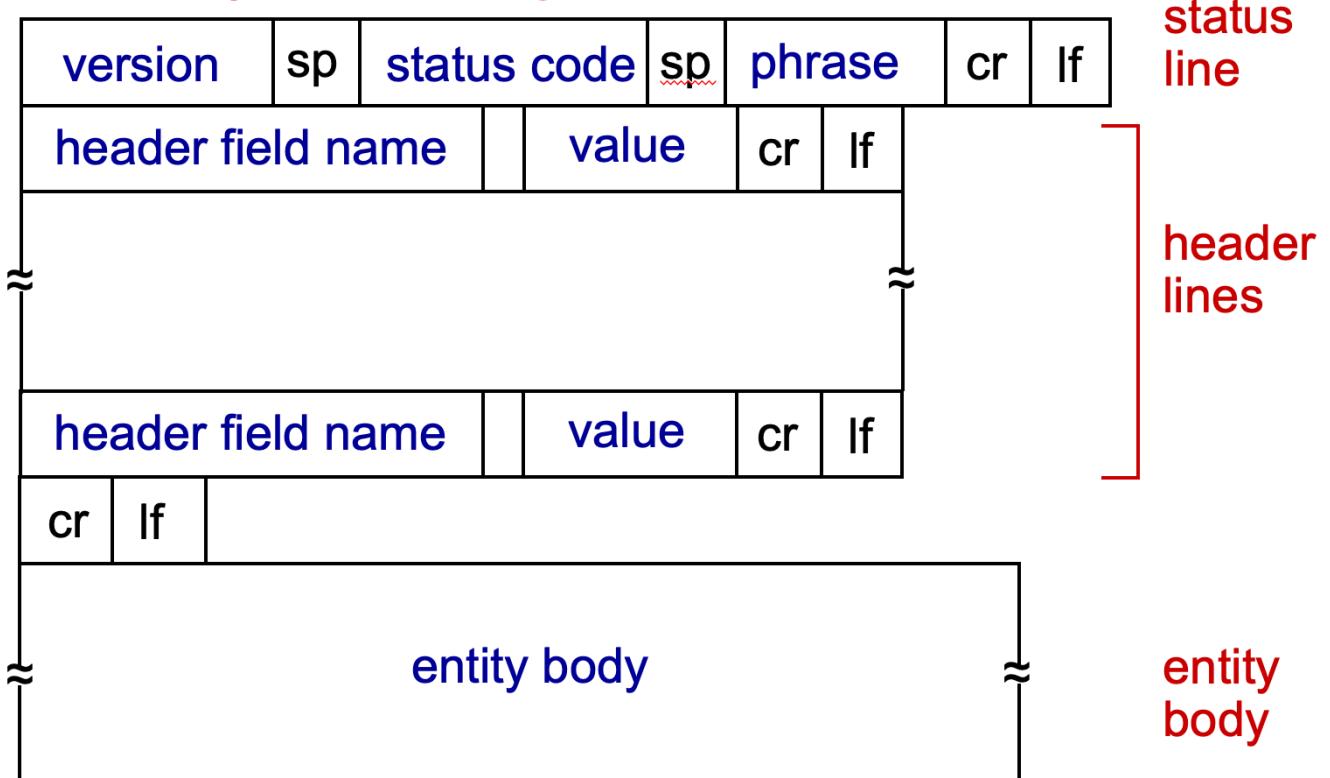
状态行 (协议版本、状态码和相应状态信息)

```
HTTP/1.1 200 OK\r\n
Connection close\r\n
Date: Thu, 06 Aug 1998 12:00:15 GMT\r\n
Server: Apache/1.3.0 (Unix) \r\n
Last-Modified: Mon, 22 Jun 1998 ..... \r\n
Content-Length: 6821\r\n
Content-Type: text/html\r\n
\r\n
\r\n
data data data data data ...
```

首部行

数据, 如请求的HTML文件

■ HTTP response message:



HTTP response status codes

位于服务器 -> 客户端的响应报文中的首行

一些状态码的例子：

200 OK

- 请求成功，请求对象包含在响应报文的后续部分

301 Moved Permanently

- 请求的对象已经被永久转移了；新的URL在响应报文的Location: 首部行中指定
- 客户端软件自动用新的URL去获取对象

400 Bad Request

- 一个通用的差错代码，表示该请求不能被服务器解读

404 Not Found

- 请求的文档在该服务上没有找到

505 HTTP Version Not Supported

User-server state: cookies

header line 首部行

大多数主要的门户网站使
用 **cookies**

4个组成部分：

- 1) 在HTTP响应报文中有
一个**cookie**的首部行
- 2) 在HTTP请求报文含有
一个**cookie**的首部行
- 3) 在用户端系统中保留有
一个**cookie**文件，由用
户的浏览器管理
- 4) 在**Web**站点有一个后
端数据库

- Susan总是用同一个PC使
用Internet Explore上
网
- 她第一次访问了一个使
用了Cookie的电子商务
网站
- 当最初的HTTP请求到达
服务器时，该Web站点
产生一个唯一的ID，并
以此作为索引在它的后
端数据库中产生一个项



Cookies能带来什么:

- 用户验证
- 购物车
- 推荐
- 用户状态 (Web e-mail)

如何维持状态:

- ❖ 协议端节点：在多个事务上，发送端和接收端维持状态
- ❖ cookies：http报文携带状态信息

Cookies与隐私:

- Cookies允许站点知道许多关于用户的信息
- 可能将它知道的东西卖给第三方
- 使用重定向和cookie的搜索引擎还能知道用户更多的信息
 - 如通过某个用户在大量站点上的行为，了解其个人浏览方式的大致模式
- 广告公司从站点获得信息

Web caching (proxy server)

- Web 缓存器 (Web cache) 也叫代理服务器 (proxy server)，它是能够代表初始 Web 服务器来满足 HTTP 请求的网络实体。
- Web 缓存器有自己的磁盘存储空间，并在存储空间中保存最近请求过的对象的副本。
- 可以配置用户的浏览器，使得用户的所有 HTTP 请求首先指向 Web 缓存器。一旦某浏览器被配置，每个对某对象的浏览器请求首先被定向到该Web 缓存器。
- 缓存既是客户端又是服务器，通常缓存是由ISP安装(大学、公司、居民区ISP)

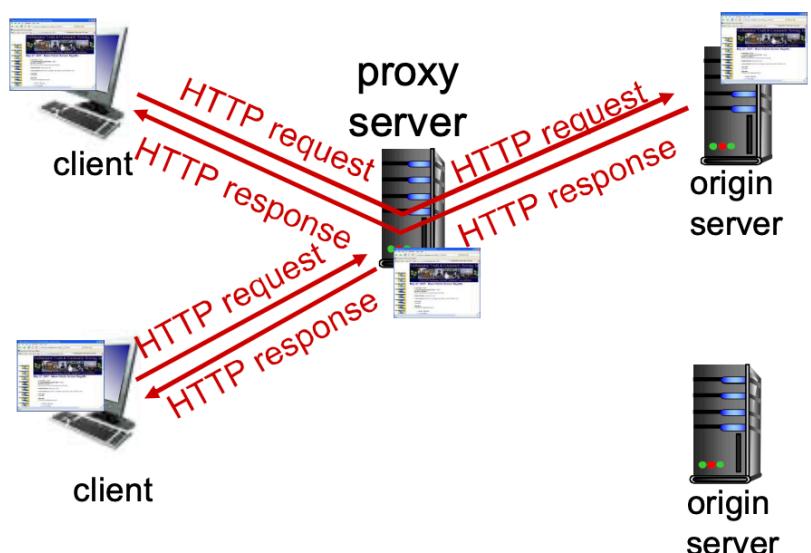
Web缓存 (代理服务器)

目标：不访问原始服务器，就满足客户的请求

□ 用户设置浏览器：通过缓存访问Web

□ 浏览器将所有的HTTP请求发给缓存

- 在缓存中的对象：缓存直接返回对象
- 如对象不在缓存，缓存请求原始服务器，然后将对象返回给客户端



为什么要使用Web缓存？

- 降低客户端的请求响应时间
- 可以大大减少一个机构内部网络与Internet接入链路上的流量
- 互联网大量采用了缓存：可以使较弱的ICP也能够有效提供内容

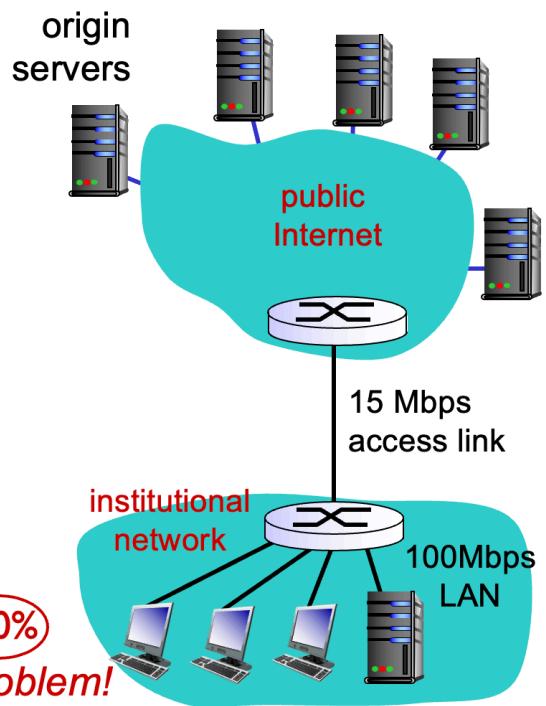
Caching example:

assumptions:

- avg object size: 1M bits = 1000KB
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1Mb*15/s=15Mbps
- RTT from institutional router to any origin server: 2 secs
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15Mbps/100Mbps = 15%
- access link utilization = 15Mbps/15Mbps = **100% problem!**
- total delay
= Internet delay + access delay + LAN delay
= 2 secs + minutes + msec

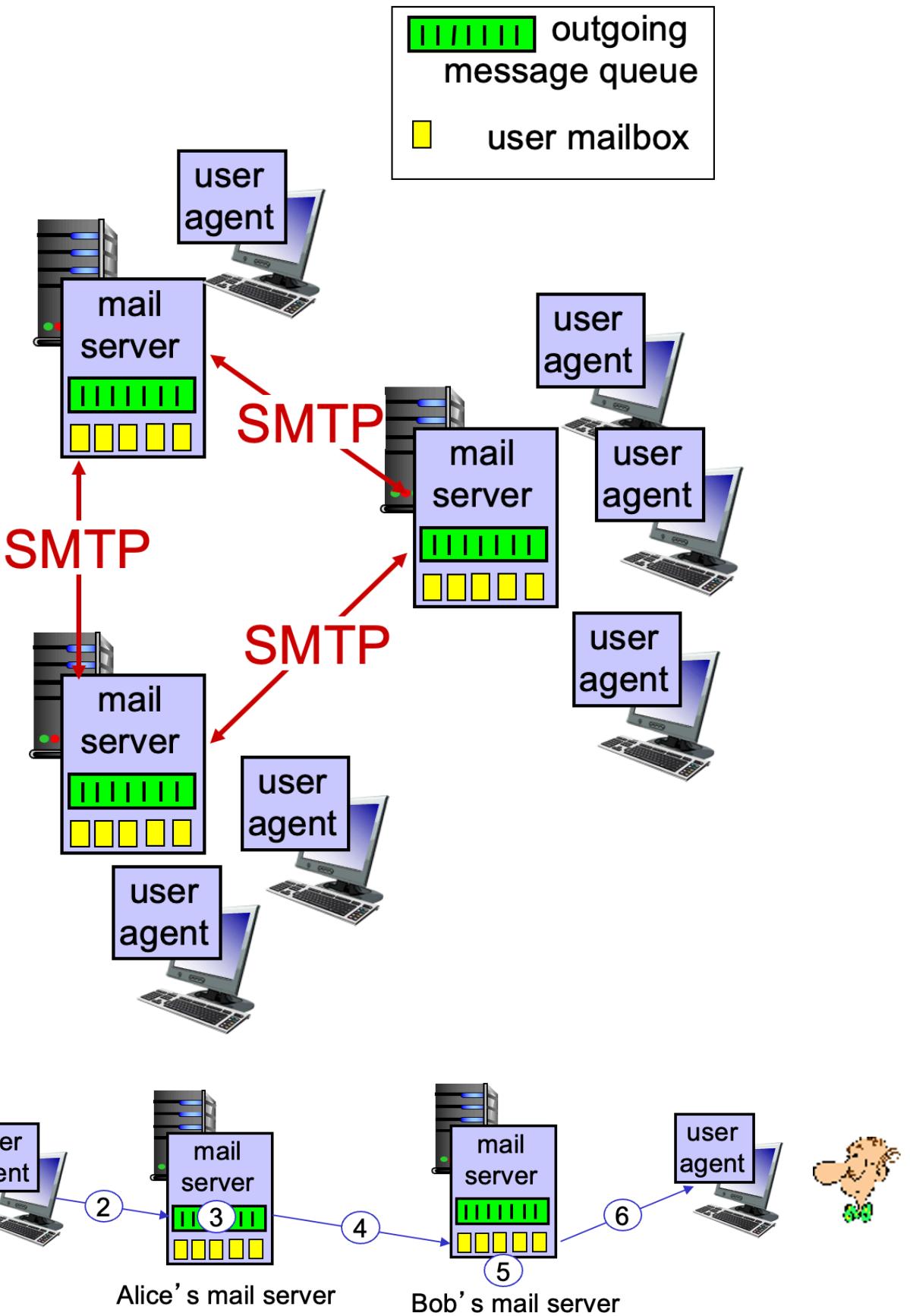


2.3 Email

因特网电子邮件系统的总体情况，有3个主要组成部分：

- user agents (用户代理):** 又名“邮件阅读器”，撰写、编辑和阅读邮件。如Outlook、Foxmail，输出和输入邮件保存在服务器上
- mail servers (邮件服务器):** 邮箱中管理和维护发送给用户的邮件，输出报文队列保持待发送邮件报文
- Simple Mail Transfer Protocol (简单邮件传输协议):** SMTP是因特网电子邮件中主要的应用层协议。
 - client: sending mail server
 - server: receiving mail server

agents = 通过用户代理软件来访问电子邮件这个应用，因此这个软件就是我们这个应用的代理。浏览器是Web应用的应用代理



SMTP [RFC 2821]

- 使用**TCP**在客户端和服务器之间传送报文，端口号为**25**
- 直接传输：从发送方服务器到接收方服务器
- 传输的3个阶段
 - 握手
 - 传输报文
 - 关闭
- 命令/响应交互
 - 命令：**ASCII**文本
 - 响应：状态码和状态信息
- 报文必须为**7位ASCII**码

简单的SMTP交互

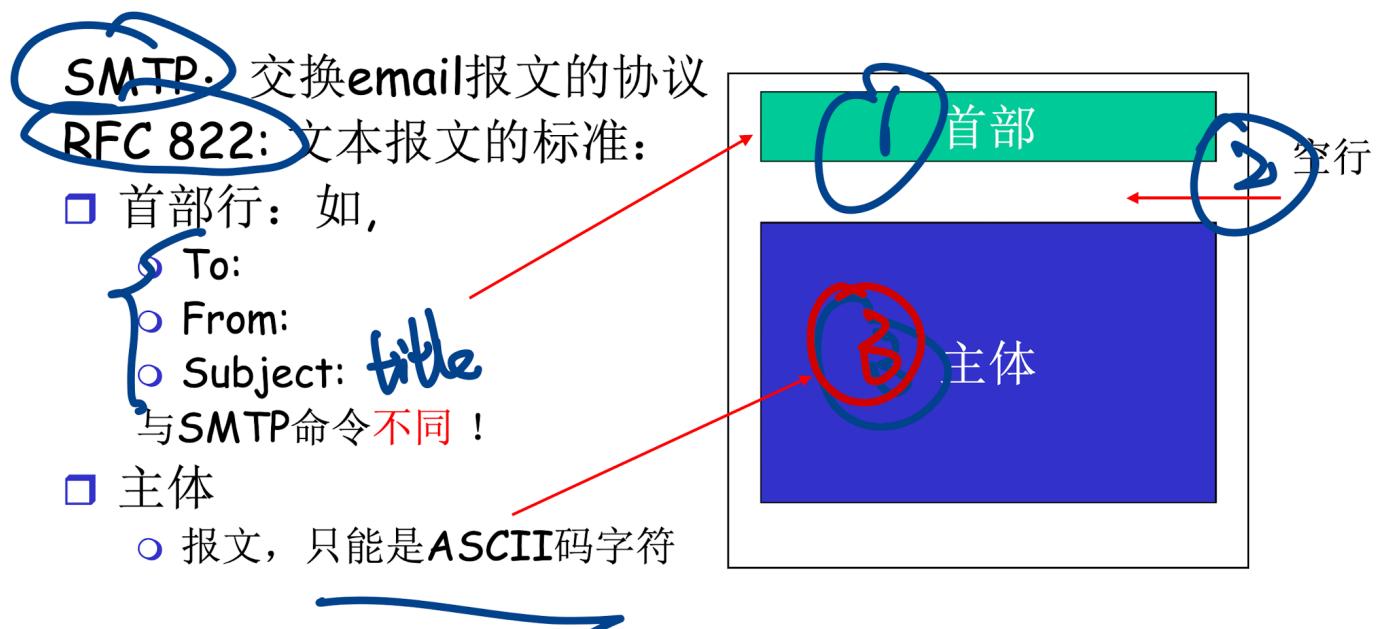
```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

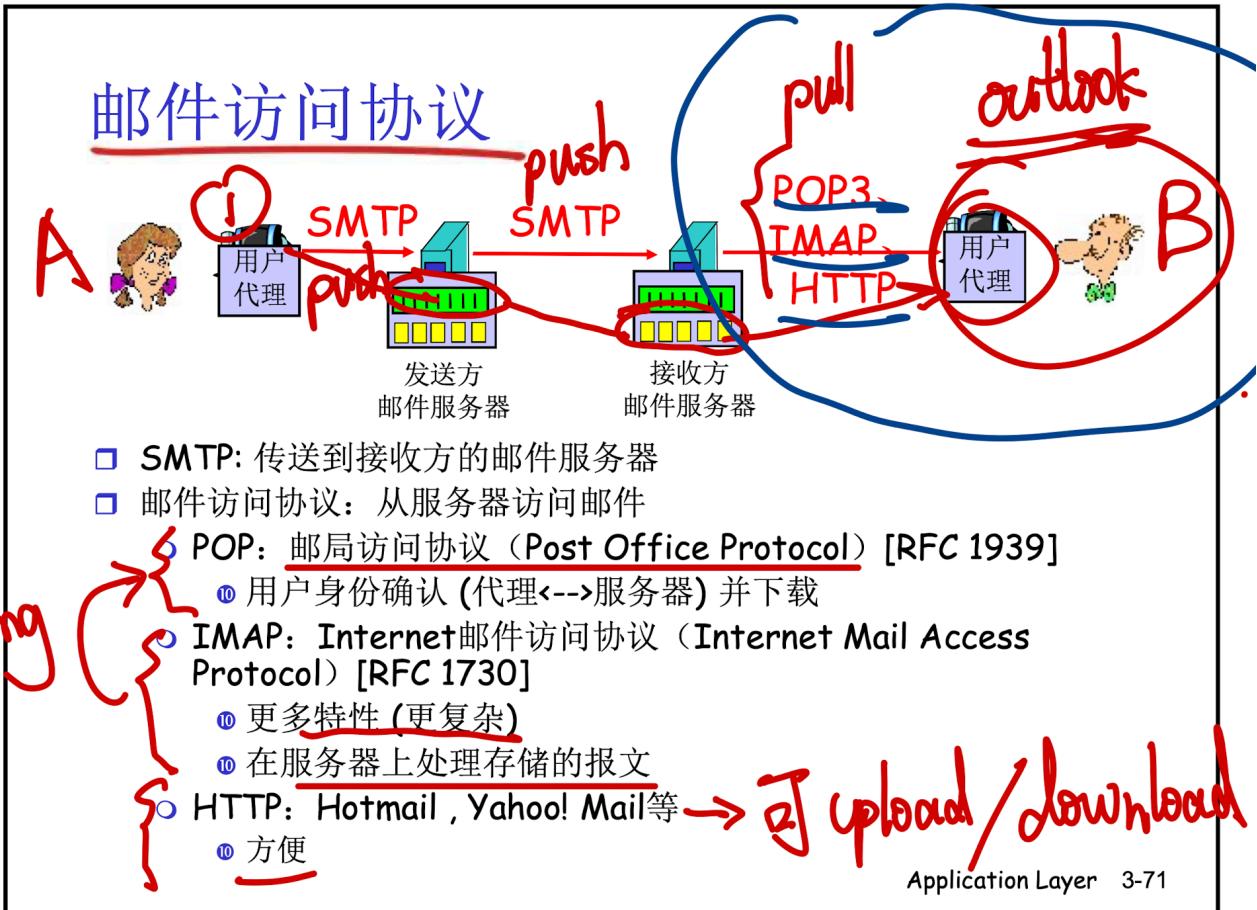
- SMTP使用持久连接
- SMTP要求报文（首部和主体）为7位ASCII编码
- SMTP服务器使用CRLF.CRLF决定报文的尾部

HTTP比较:

- HTTP: 拉 (pull)
- SMTP: 推 (push)
- 二者都是ASCII形式的命令/响应交互、状态码
- HTTP: 每个对象封装在各自的响应报文中
- SMTP: 多个对象包含在一个报文中

邮件报文格式





POP3 (续) 与 IMAP

POP3 (续)

- 先前的例子使用 “下载并删除” 模式。
 - 如果改变客户机, Bob不能阅读邮件
- “下载并保留”：不同客户机上为报文的拷贝
- POP3在会话中是无状态的
— stateless

本地管理文件夹

IMAP

- IMAP服务器将每个报文与一个文件夹联系起来
- 允许用户用目录来组织报文
- 允许用户读取报文组件
- IMAP在会话过程中保留用户状态：
 - 目录名、报文ID与目录名之间映射

state

远程管理文件夹

Application Layer 3-73

2.4 DNS

Domain Name System 运行在UDP上

域名/主机名 => IP地址

在IP协议中，我们与目标主机交互，需要记住对方的IP192.168.0.1，这就像我们要记住好多朋友的电话一样。电话也太难记了，所以我们需要有个电话本，记录小明 => 18316160606的对应关系。好比www.baidu.com对应10.0.21.4。

DNS(Domain Name System)总体思路和目标

□ DNS的主要思路

- 分层的、基于域的命名机制
- 若干分布式的数据库完成名字到IP地址的转换
- 运行在UDP之上端口号为53的应用服务
- 核心的Internet功能，但以应用层协议实现
 - ⑩ 在网络边缘处理复杂性

□ DNS主要目的：

- 实现主机名-IP地址的转换(name/IP translate)
- 其它目的
 - ⑩ 主机别名到规范名字的转换：Host aliasing
 - ⑩ 邮件服务器别名到邮件服务器的正规名字的转换：Mail server aliasing
 - ⑩ 负载均衡：Load Distribution

- 主机别名 (host aliasing): 有着复杂主机名的主机能拥有一个或者多个别名

例如，一台名为 relay1.west-coast.enterprise.com 的主机，可能还有两个别名为 enterprise.com 和 www.enterprise.com

在这种情况下，relay1.west-coast.enterprise.com 也称为 规范主机名 (canonical hostname)。主机别名(当存在时)比主机规范名更 加容易记忆。应用程序可以调用 DNS 来获得主机别名对应的规范主机名以及主机的 IP地址。

- 邮件服务器别名 (mail server aliasing): 显而易见，人们也非常希望电子邮件地址好记忆。

例如，如果 Bob 在雅虎邮件上有一个账户，Bob 的邮件地址就像 bob@yahoo.com 这样简单。然而，雅虎邮件服务器的主机名可能更为复杂，不像 yahoo.com 那样简单好记(例如，规范主机名可能像 relay1.west-coast.hotmail.com 那样)。

电子邮件应用程序可以调用 DNS，对提供的主机名别名进行解析，以获得该主机的规范主机名及其 IP 地址。

事实上，MX 记录(参见后面)允许一个公司的邮件服务器和 Web 服务器使用相同(别名化的)的主机名，例如，一个公司的 Web 服务器和邮件服务器都能叫作 enterprise.com。

- 负载分配 (load distribution)。DNS 也用千在冗余的服务器(如冗余的 Web 服务器等)之间进行负载分配。繁忙的站点(如 cnn.com)被冗余分布在多台服务器上，每台服务器均运行在不同的端系统上，每个都有着不同的 IP 地址。由于这些冗余的 Web 服务器，一个 IP 地址集合因此与同一个规范主机名相联系。DNS 数据库中存储着这些 IP 地址集合。当客户对映射到某地址集合的名字发出一个 DNS 请求时，该服务器用 IP 地址的整个集合进行响应，但在每个回答中循环这些地址次序。因为客户通常总是向 IP 地址排在最前面的服务器发送 HTTP 请求报文，所以 DNS 就在所有这些冗余的 Web 服务器之间循环分配了负载。DNS 的循环同样可以用千邮件服务器，因此，多个邮件服务器可以具有相同的别名。一些内容分发公司如 Akamai 也以

问题1: DNS名字空间(The DNS Name Space)

□ DNS域名结构

- 一个层面命名设备会有很多重名
- DNS采用层次树状结构的 命名方法
- Internet 根被划为几百个顶级域(top level domains)

① 通用的(generic)

.com; .edu ; .gov ; .int ; .mil ; .net ; .org
.firm ; .hsop ; .web ; .arts ; .rec ;

② 国家的(countries)

.cn ; .us ; .nl ; .jp

- 每个(子)域下面可划分为若干子域(subdomains)
- 树叶是主机

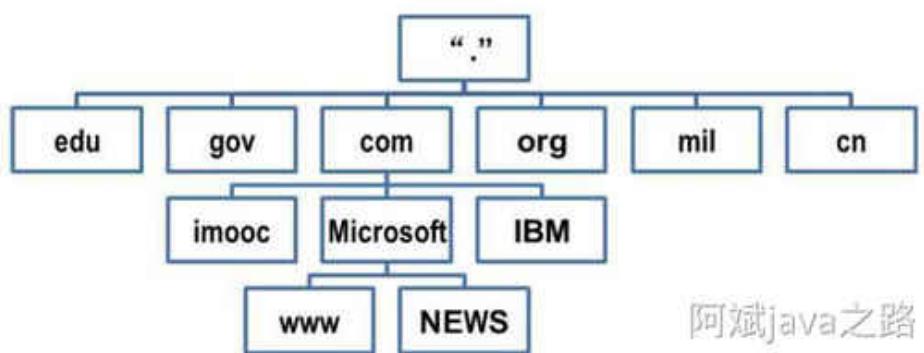
Subdomains

:
:
:

DNS Name Space

域名空间结构

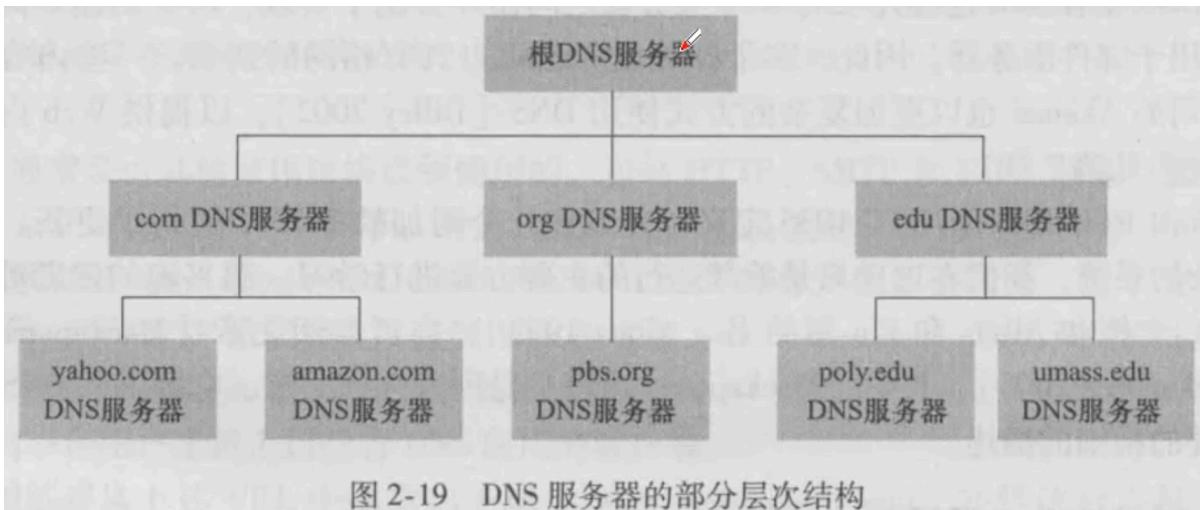
- 根域
- 顶级域
 - 组织域
 - 国家或地区域
- 二级域
- 主机名



阿斌Java之路

上图展示了 DNS 服务器的部分层次结构, 从上到下依次为根域名服务器、顶级域名服务器和权威域名服务器。域名和IP地址的映射关系必须保存在域名服务器中, 供所有其他应用查询。

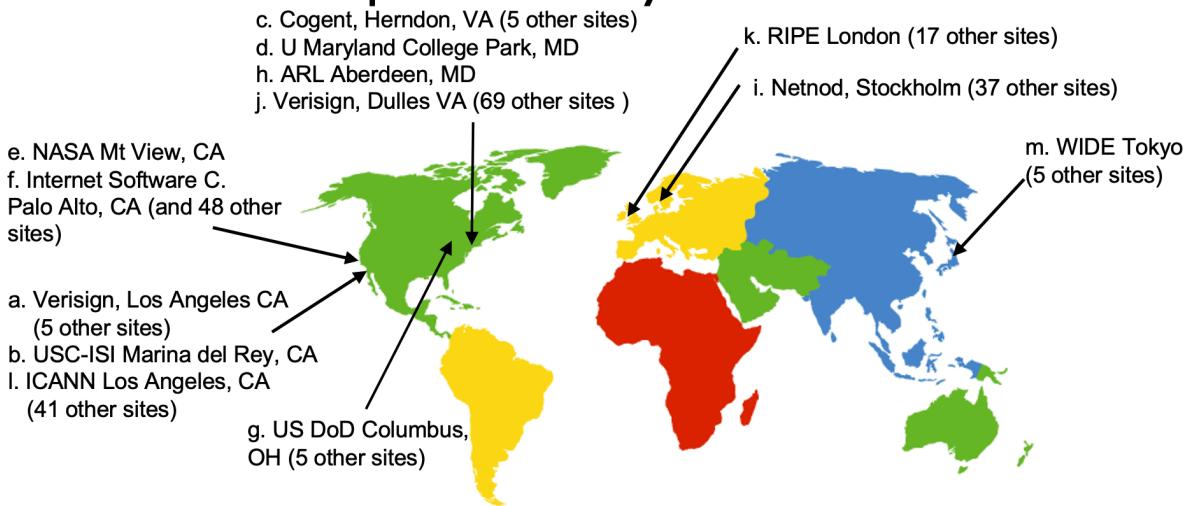
DNS hierarchy



- **根域名服务器 (root name servers):** 根域名服务器是最高层次的域名服务器。每个根域名服务器都知道所有的顶级域名服务器的域名及其IP地址。因特网上共有13个不同IP地址的根域名服务器。当本地域名服务器向根域名服务器发出查询请求时，路由器就把查询请求报文转发到离这个DNS客户最近的一个根域名服务器。这就加快了DNS的查询过程，同时也更合理地利用了因特网的资源。

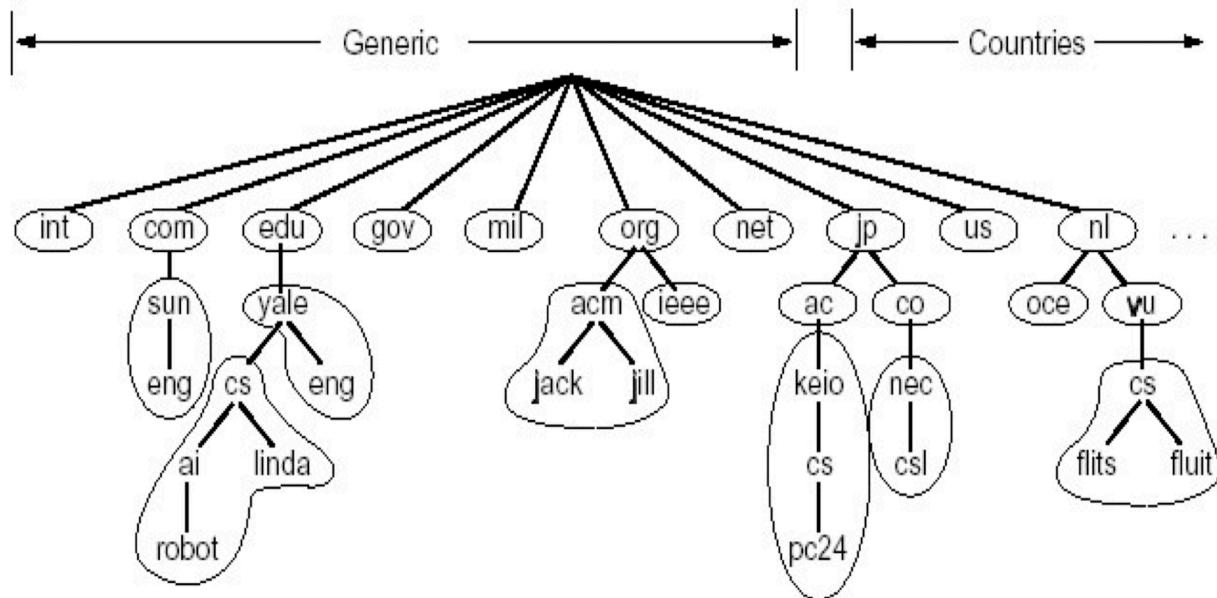
13 logical root name servers worldwide

- each “server” replicated many times



- **顶级域名服务器 (top-level domain (TLD) servers)** 这些域名服务器负责管理在该顶级域名服务器注册的所有二级域名。当收到DNS查询请求时就给出相应的回答（可能是最后的结果，也可能是下一级权限域名服务器的IP地址）。
 - 负责com, org, net, edu, aero, 博物馆和所有顶级国家域名，例如:uk, fr, ca, jp
 - Network Solutions maintains servers for com TLD
 - Educause for edu TLD
- **权威域名服务器 (authoritative DNS servers):** 这些域名服务器负责管理某个区的域名。每一个主机的域名都必须在某个权限域名服务器处注册登记。因此权限域名服务器知道其管辖的域名与IP地址的映射关系。另外，权限域名服务器还知道其下级域名服务器的地址。

名字空间划分为若干区域：Zone

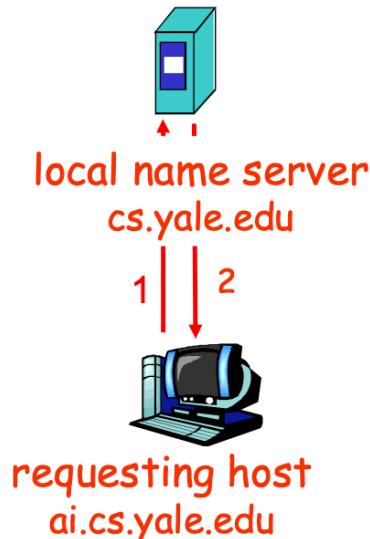


权威DNS服务器：组织机构的DNS服务器， 提供组织机构服务器（如Web和mail）可访问的主机和IP之间的映射
组织机构可以选择实现自己维护或由某个服务提供商来维护

Local DNS server

本地域名服务器不属于上述的域名服务器的等级结构。当一个主机发出DNS请求报文时，这个报文就首先被送往该主机的本地域名服务器。本地域名服务器起着代理的作用，会将该报文转发到上述的域名服务器的等级结构中。本地域名服务器离用户较近，一般不超过几个路由器的距离，也有可能就在同一个局域网中。本地域名服务器的IP地址需要直接配置在需要域名解析的主机中。

- 并不严格属于层次结构
- 每个ISP (居民区的ISP、公司、大学)都有一个本地DNS服务器，也称为“默认名字服务器”
- 当一个主机发起一个DNS查询时，查询被送到其本地DNS服务器。起着代理的作用，将查询转发到层次结构中



如果目标名字在Local Name Server中

Domain name resolution

域名解析分为两种，递归和迭代。

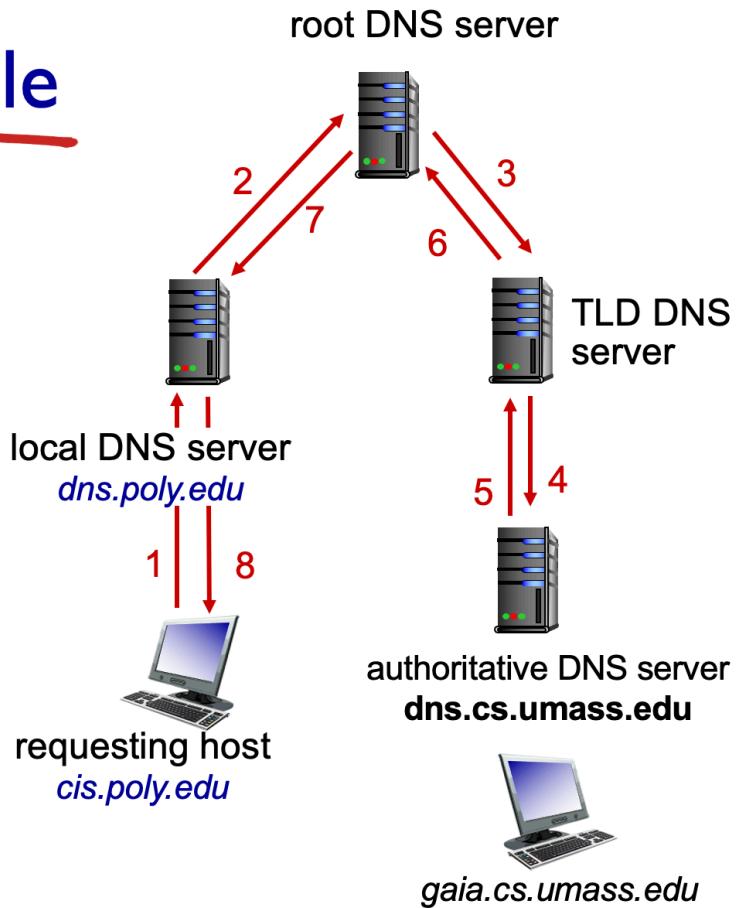
recursive query

- 名字解析负担都放在当前联络的名字服务器上
- 问题：root DNS server 的负担太重，所以我们要学习/使用 iterated queries

DNS name resolution example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



iterated query

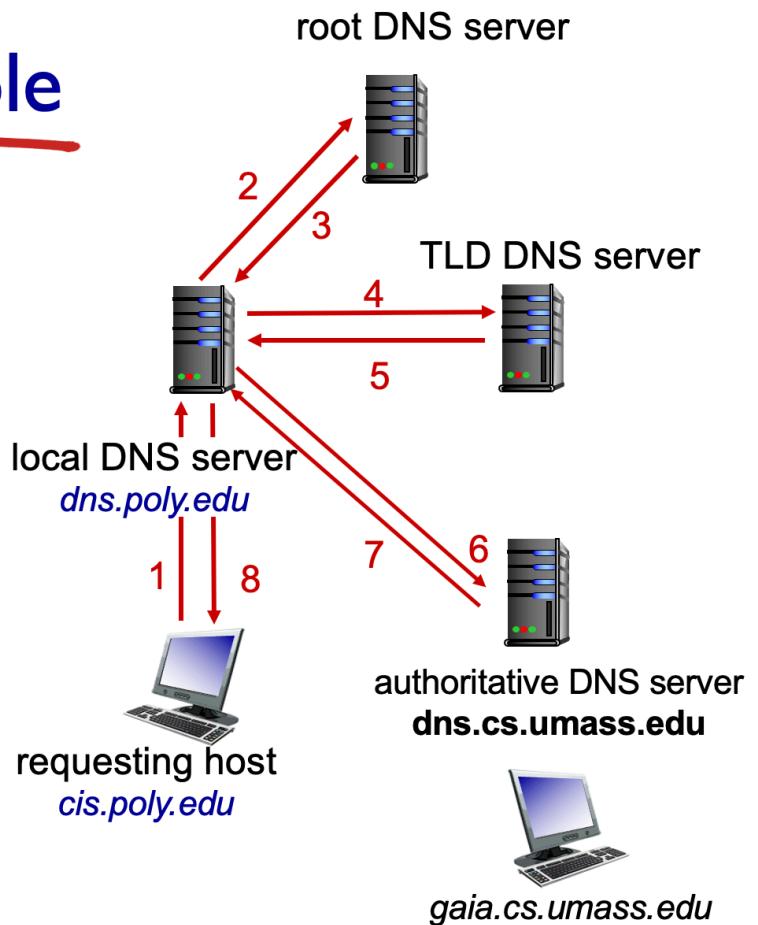
- 迭代的方式，大大节省了根域名服务器的压力。
- 由于域名解析的报文并不长，DNS域名解析的过程采用的UDP的传输协议，大大的增加了传输间的效率。
- DNS解析服务器是分布式的，在主域名解析服务器向其他域名解析服务器直接同步数据时，采用的是TCP的传输协议。

DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

- contacted server replies with name of server for further contact
 - “I don’t know this name, but ask this server”



当你输入一个网址的时候：

- 请求本地缓存，查询host域名配置
- 本地缓存没有，查询本地DNS服务器。什么是本地DNS服务器呢？其实并不是配置在你家里的，而是你的宽带属于哪个服务商，就会使用哪个服务商的DNS。
- 本地DNS服务器没有缓存，查询根域名服务器。
- 根域名服务器会指向顶级域名服务器
- 顶级域名服务器会指向权威域名服务器
- 最终拿到权威域名服务器结果，并缓存在本地DNS服务器

DNS: caching, updating records

- 一旦(任何)名称服务器学会了映射，它就 **caches mapping** (缓存映射)，由于 主机和主机名与 IP 地址间的映射并不是永久的，local DNS 服务器在 一段时间后 (TTL, 通常设置为两天)将丢弃缓存的信息。
- TLD服务器通常缓存在本地名称服务器中，因此根名称服务器不常被访问
- cached entries**(缓存的条目) 可能会 **out-of-date** 过期 (尽最大努力将名称转换为地址!)，如果name host更改了IP地址，可能直到缓存条目的所有TTLs过期才会被全网所知
- 共同实现 DNS 分布式数据库的所有 DNS 服务器存储了 资源记录 (Resource Record , RR), RR 提供了主机名到 IP 地址的映射，每个 DNS 回答报文包含了一条或多条资源记录

资源记录(resource records)

- 作用：维护 域名-IP地址(其它)的映射关系

- 位置： Name Server的分布式数据库中

RR格式: (domain_name, ttl, type, class, Value)

- Domain_name: 域名

- Ttl: time to live : 生存时间(权威, 缓冲记录)

- Class 类别：对于Internet，值为IN

- Value 值：可以是数字，域名或ASCII串

- Type 类别：资源记录的类型—见下页

DNS : 保存资源记录(RR)的分布式数据库
distributed database storing resource records (RR)

RR 格式: (name, value, type, ttl)

Type=A

- Name为主机
- Value为IP地址

Type=NS

- Name域名(如foo.com)
- Value为该域名的权威服务器的域名

Type=CNAME

- Name为规范名字的别名 alias name
www.ibm.com 的规范名字为

servereast.backup2.ibm.com

- value 为规范名字

Type=MX

- Value为name对应的邮件服务器的名字

TTL: 生存时间，决定了资源记录应当从缓存中删除的时间

DNS protocol, messages

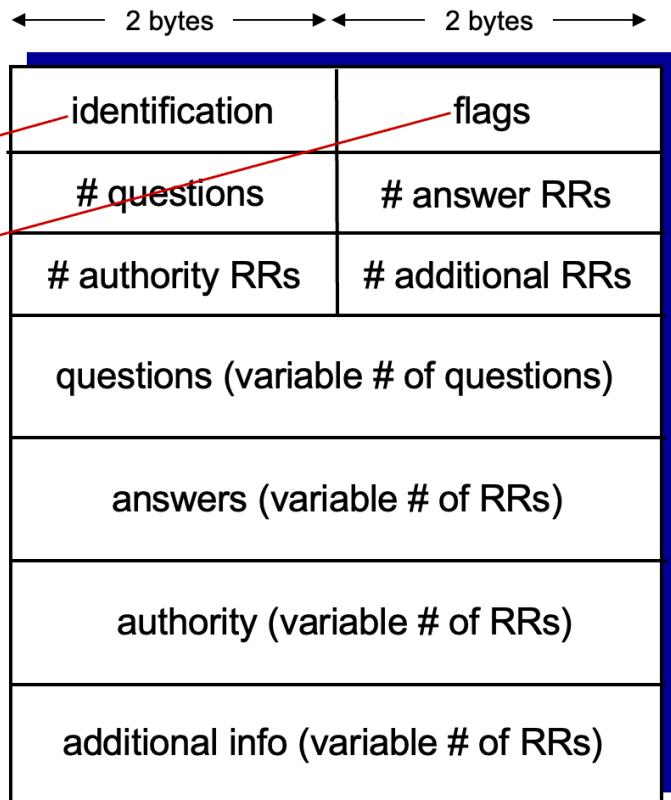
- **query** and **reply** messages, both with same **message format**

message header

- Identification(ID): 16 bit
(query, reply to query uses same)

- flags:

- query or reply
- recursion desired
- recursion available
- reply is authoritative

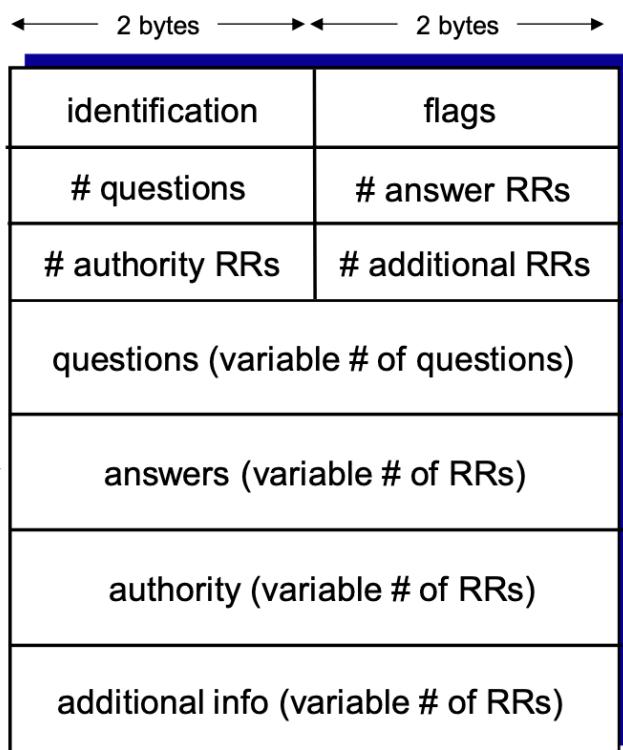


一个查询的
Name, type 字段

对应查询
的RR记录

权威服务
器的记录

附加的
有用信息



Inserting records into DNS

上面的讨论只是关注如何从 DNS 数据库中取数据。你可能想知道这些数据最初是怎么进入数据库中的。

- 在上级域的名字服务器中增加两条记录，指向这个新增的子域的域名 和 域名服务器的地址
- 在新增子域 的名字服务器上运行名字服务器，负责本域的名字解析： 名字->IP地址

例子：在com域中建立一个“**Network Utopia**”

- 到注册登记机构注册域名**networkutopia.com**
 - 需要向该机构提供权威DNS服务器（基本的、和辅助的）的名字和IP地址
 - 登记机构在com TLD服务器中插入两条RR记录：
([networkutopia.com](#), [dns1.networkutopia.com](#), NS)
([dns1.networkutopia.com](#), 212.212.212.1, A)
- 在[networkutopia.com](#)的权威服务器中确保有
 - 用于Web服务器的[www.networkutopia.com](#)的类型为A的记录
 - 用于邮件服务器[mail.networkutopia.com](#)的类型为MX的记录

2.7 socket programming with UDP and TCP

典型的网络应用是由一对程序(即客户程序和服务器程序)组成的，它们位于两个不同的端系统中。当运行这两个程序时，创建了一个客户进程和一个服务器进程，同时它们通过从套接字读出和写入数据在彼此之间进行通信。开发者创建一个网络应用时，其主要任务就是编写 client 程序和 server 程序的代码。

目标: 学习如何构建能借助sockets进行通信的C/S应用程序 (eg. 服务器与客户端的交互：服务器接收数据并将字符转换为大写，服务器将修改后的数据发送到客户端，客户端接收修改后的数据并在其屏幕上显示行)

- socket 其实就是操作系统提供给程序员操作「网络协议栈」的接口，是一套用于不同 host 间通信的API，你能通过socket的接口，来控制协议栈工作，从而用它来实现网络通信，达到跨主机通信。
- 它工作在我们的TCP IP协议栈之上
- 要通过 socket 与不同的组织建立联系，我们只需要指定主机的IP号和端口号，IP地址用于唯一标识你的网络设备。那为什么还要额外指定一个端口好呢？如果没有端口，操作系统就没有办法区分数据到底应该发到哪一个应用之上。因此，端口主要用于区分主机上的不同应用。
- 通过 socket 我们可以建立一条用于不同组织不同应用之间的虚拟数据通道，并且它是应用对应用的，就像是一条数据线，连接在不同应用的插槽上。

socket建立之后，我们的消息收发都通过socket，变得非常方便。Socket是双方会话关系的本地标识。在研发阶段，开发者必须最先做的一个决定是：应用程序是运行在 TCP 上还是运行在 UDP 上。

socket 一般分为 TCP 网络编程和 UDP 网络编程。

- 针对TCP协议 (**reliable, Data Stream, byte stream oriented**), Socket是一个四元组 (本地ip, 本地port, 对方ip, 对方port)。
 - TCP是面向连接的，并且为两个端系统之间的数据流动提供 可靠的字节流通道 (**TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server**)
 - TCP 要求收发数据的双方扮演不同的角色：Server/Client
- 针对UDP协议 (**unreliable datagram**), Socket是一个二元组 (本地ip, 本地port)。
 - UDP 是无连接的，且发送数据前不握手，从一个端系统向另一个端系统发送独立的数据分组，不对交付提供任何保证。
 - sender 将目标的 IP 地址和端口# 显式附加到 each packet
 - receiver 从收到的数据包中提取发送方的IP地址和端口。传输的数据可能会丢失或无序接收

我们通过一个简单的 UDP 应用程序和一个简单的 TCP应用程序来介绍 UDP 和 TCP套接字编程。

TCP socket programming

TCP Server

服务器首先运行，等待连接建立：

```
from socket import*
```

1. **ServerSocket:** 创建 服务器套接字

```
serverSocket = socket(AF_INET, SOCK_STREAM) ## 创建 TCP 服务器 welcoming socket
## 第一个参数指示底层网络使用的是 IPv4。第二个参数指示该 socket 是 SOCK_STREAM 类型，这表明它是一个 TCP socket
```

2. **bind:** 将套接字 绑定 到一个本地地址和端口上

3. **listen:** 将套接字设定为监听模式，准备接受客户端请求

```
serverSocket.bind(('0.0.0.0', 12000)) ## 将我们创建的 socket 关联到我们主机的某一个 network
interface(IP) 和 port 上
serverSocket.listen(1) ## 将socket设定为监听模式，准备接受客户端请求/聆听某个客户敲门，其中参数定义了请求连接的最大数(至少为1)
print ('The server is ready to receive')
```

4. **accept:** 阻塞 等待客户端请求到来。当请求到来后，接受连接请求，返回一个新的对应于此客户端连接的套接字socketClient
5. **IO流操作:** 用返回的套接字socketClient和客户端进行通信
6. **accept:** 返回，等待另一个客户端请求

```

while True: ## loop forever
    connectionSocket, addr = serverSocket.accept()
    ## 当客户敲该门时, 程序为 serverSocket 调用 accept()方法
    ## 这在服务器中创建了一个称为 connectionSocket 的新socket, 由这个特定的客户专用。
    ## 客户和服务器则完成了握手, 在 客户的 clientSocket 和服务器的 serverSocket 之间创建了 一个
    ## TCP 连接。

    sentence = connectionSocket.recv(1024).decode() ## read bytes from socket (but not
address as in UDP)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    ## 借助于创建的 TCP 连接, 客户与服务器现在能够通过该连接相互发送字节。
    ## 使用 TCP, 从一侧发送的所有字节不仅确保到达另一侧, 而且确保按序到达。

```

7. **close:** 关闭套接字

```

connectionSocket.close() ## close connection to this client (but not welcoming
socket)

```

TCP Client

客户端主动和服务器建立连接:

```

from socket import *

```

1. **Socket:** 创建客户端套接字

```

clientSocket = socket(AF_INET, SOCK_STREAM)
## 该行创建了client socket, 称为 clientSocket
## 第一个参数仍指示底层网络使用的是 IPv4. 第二个参数指示该套接字是 SOCK_STREAM 类型. 这表明它是一个
TCP 套接字

```

值得注意的是当我们创建该客户套接字时仍未指定其端口号; 相反, 我们让操作系统为我们做此事 (没有bind)

2. **connect:** 向服务器发出连接请求

```

clientSocket.connect(("127.0.0.1", 12000))
## create TCP socket for server, remote port 12000
## 这行代码执行完后, 执行三次握手, 并在客户和服务器之间创建起一条 TCP 连接

```

3. **IO流操作:** 和服务器进行通信

```

sentence = input('Input lowercase sentence:') ## 从用户获得了一个句子，字符串 sentence 连续收集字符 直到用户键入回车以终止
clientSocket.send(sentence.encode()) ## 发送 sentence 到 TCP，无需附加 server name, port
modifiedSentence = clientSocket.recv(1024) ## 当字符到达服务器时，它们被放置在字符串 modifiedSentence 中
print ('From Server:', modifiedSentence.decode())

```

4. close: 关闭套接字

```

clientSocket.close() ## 关闭套接字，因此关闭了客户和服务器之间的 TCP 连接。它引起客户中的TCP向服务器中的TCP发送一条TCP报文

```

C/S socket interaction: TCP

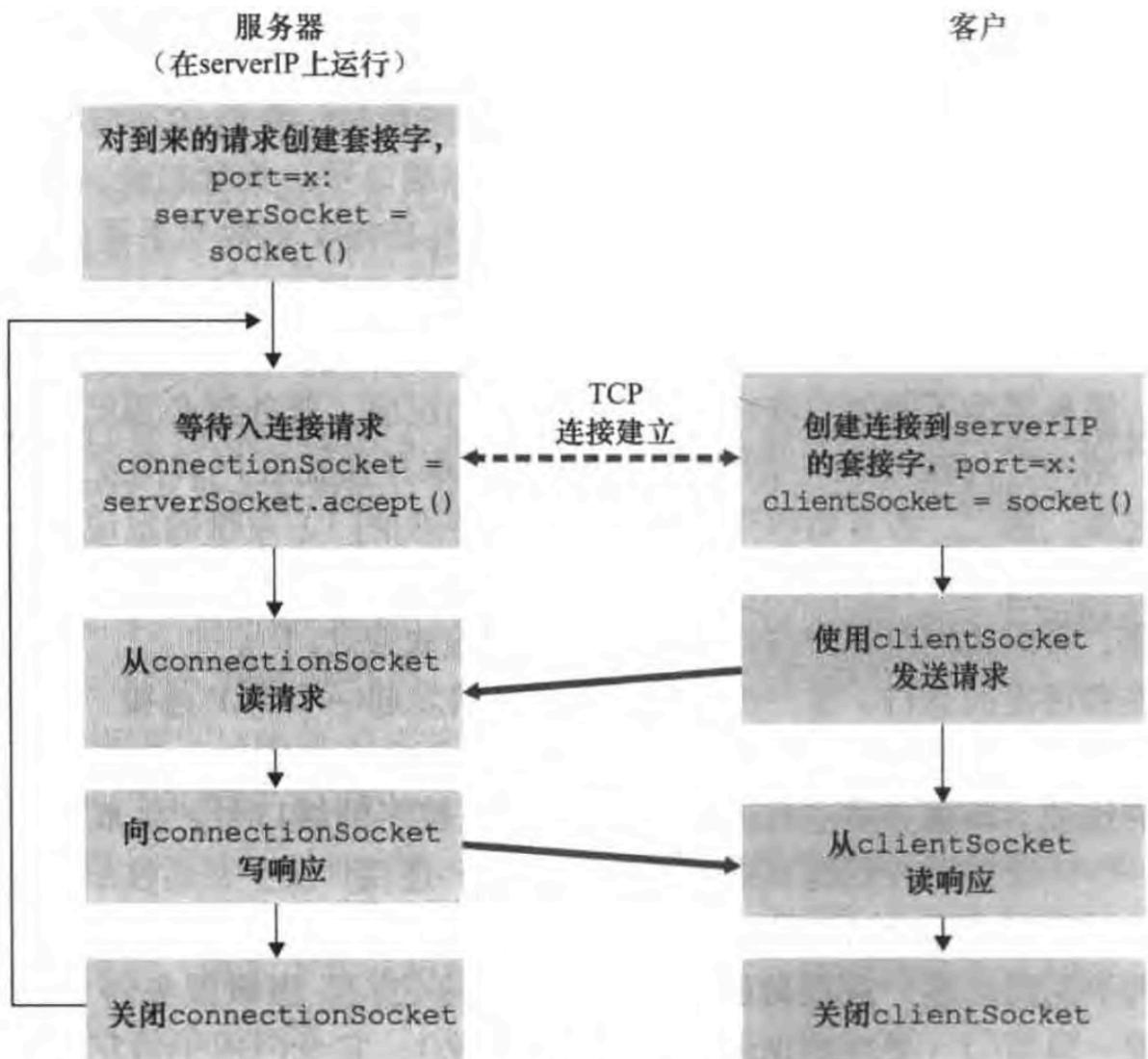


图 2-28 使用 TCP 的客户 - 服务器应用程序

UDP socket programming

- 在客户端和服务器之间没有连接，没有握手
- 发送端在每一个报文中明确地指定目标的IP地址和端口号
- 服务器必须从收到的分组中提取出发送端的IP地址和端口号
- 传送的数据可能乱序，也可能丢失（不可靠的字节组的传送服务）

UDP Server

只有一个socket:

Socket	IP	Port
8888	1.1.1.1	80

```
from socket import *
```

1. **serverSocket**: 创建套接字

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM) ## create UDP socket, SOCK_DGRAM = UDP
```

2. **bind**: 将套接字绑定到一个本地地址和端口上

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

3. **recvfrom**: 阻塞等待接收消息

4. 收到的消息被封装在 modifiedMessage 里，里面有对方的ip和端口

5. **sendto**: 可以根据对方的DatagramPacket再像对方发送消息

```
while True:
    message, clientAddress = serverSocket.recvfrom(2048) ## 从 UDP socket 读取消息，获取
    client IP 和 port
    modifiedMessage = message.decode().upper() ## send upper case string back to this
    client
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

UDP Client

```
from socket import *
```

1. **clientSocket:** 创建客户端套接字

```
serverName = 'hostname'  
serverPort = 12000  
clientSocket = socket(AF_INET, SOCK_DGRAM) ## 创建了客户的套接字, 称为 clientSocket  
## 第一个参数指示了地址簇, AF_INET 指示了底层网络使用了 IPv4  
## 第二个参数指示了该套接字是 SOCK_DGRAM 类型的, 这意味着它是一个 UDP 套接字
```

2. **message:** 创建小写用户输入字符串。

3. **sendto():** 为报文附上目的地址 (serverName , serverPort) 并且向进程的套接字 **clientSocket** 发送结果分组

```
message = input('Input lowercase sentence:') ## get user keyboard input  
clientSocket.sendto(message.encode(),(serverName, serverPort)) ## Attach server name,  
port to message; send into socket
```

4. **modifiedMessage:** 将server修改完的字符串从 socket 放入 modifiedMessage

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048) ## read reply characters  
from socket into string  
print (modifiedMessage.decode()) ## 打印出 modifiedMessage 大写字符串
```

5. **close:** 关闭套接字

```
clientSocket.close()
```

我们可以发现, tcp和udp各自用一套端口, 哪怕都是8000, 但是互不冲突。

C/S socket interaction: UDP

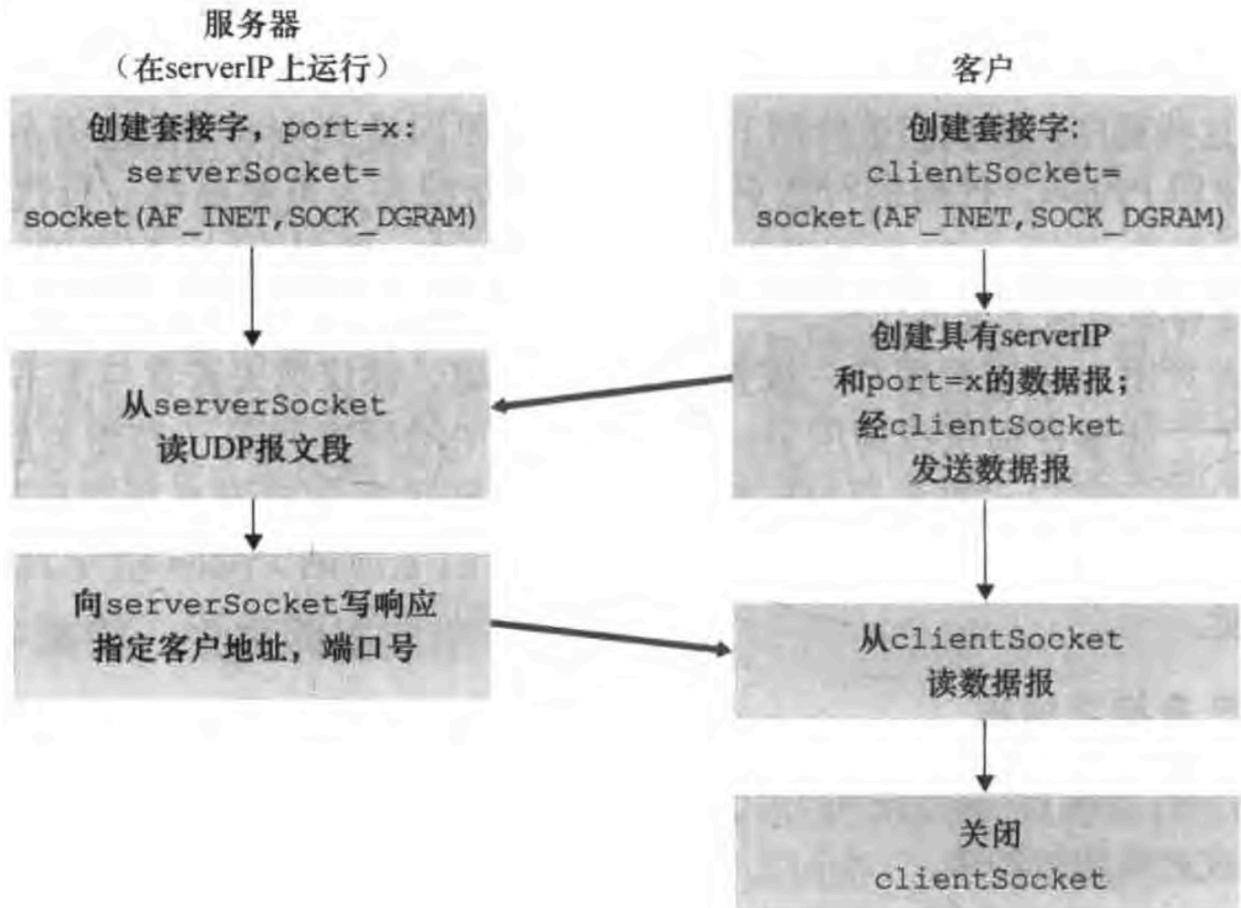


图 2-26 使用 UDP 的客户 – 服务器应用程序

03. Transport Layer 进程间的逻辑通信

运输层协议为运行在不同 主机上的应用进程之间提供了 逻辑通信 (logic communication) 功能

从应用程序的角度看，通过逻辑通信，运行不同进程的主机好像使用socket直接相连一样；

实际上，这些主机也许位于地球的两侧，通过很多路由器及多种不同类型的链路相连。应用进程使用运输层提供的逻辑通信功能彼此发送报文，而无须考虑承载这些 message 的物理基础设施的细节

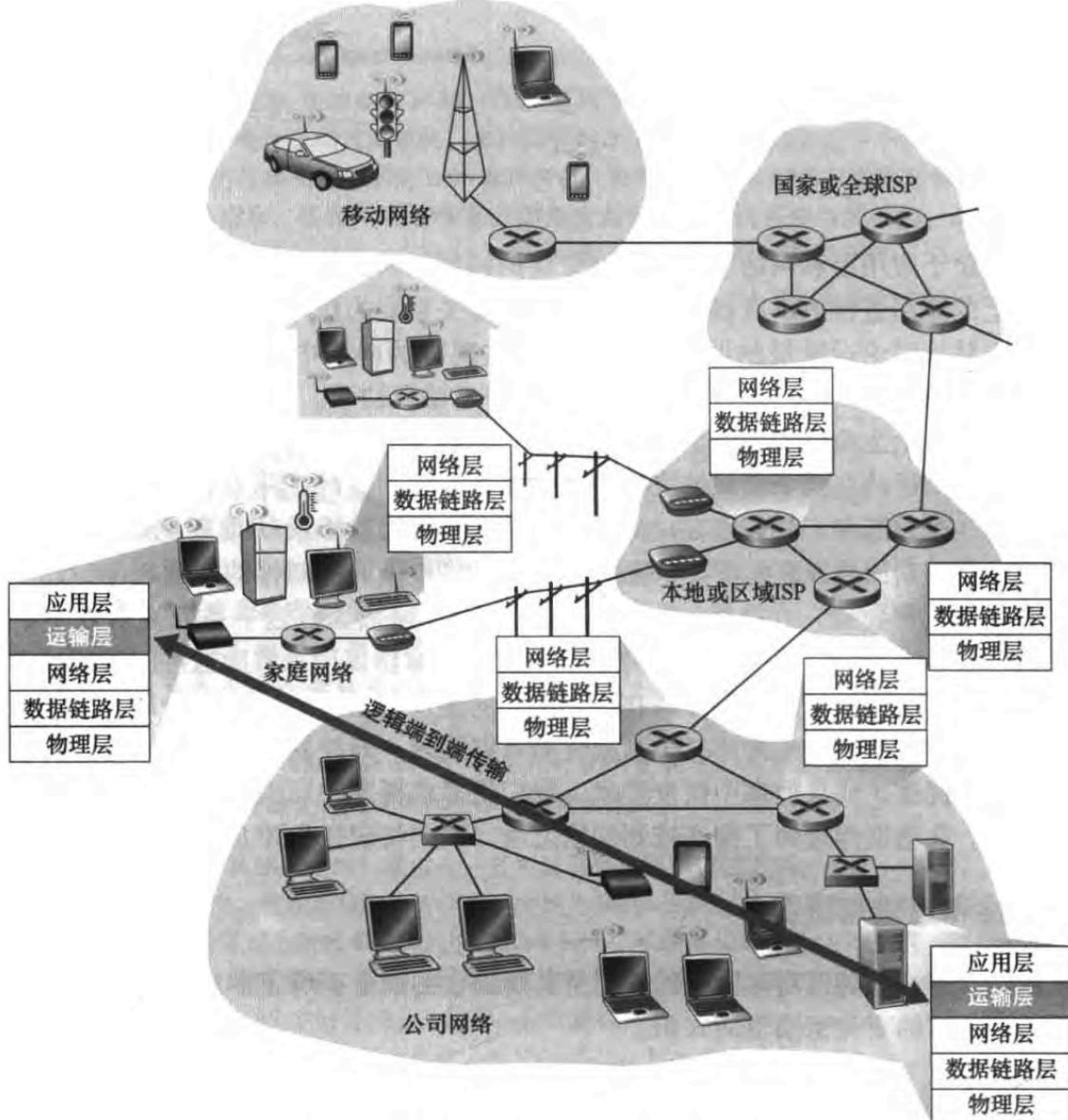
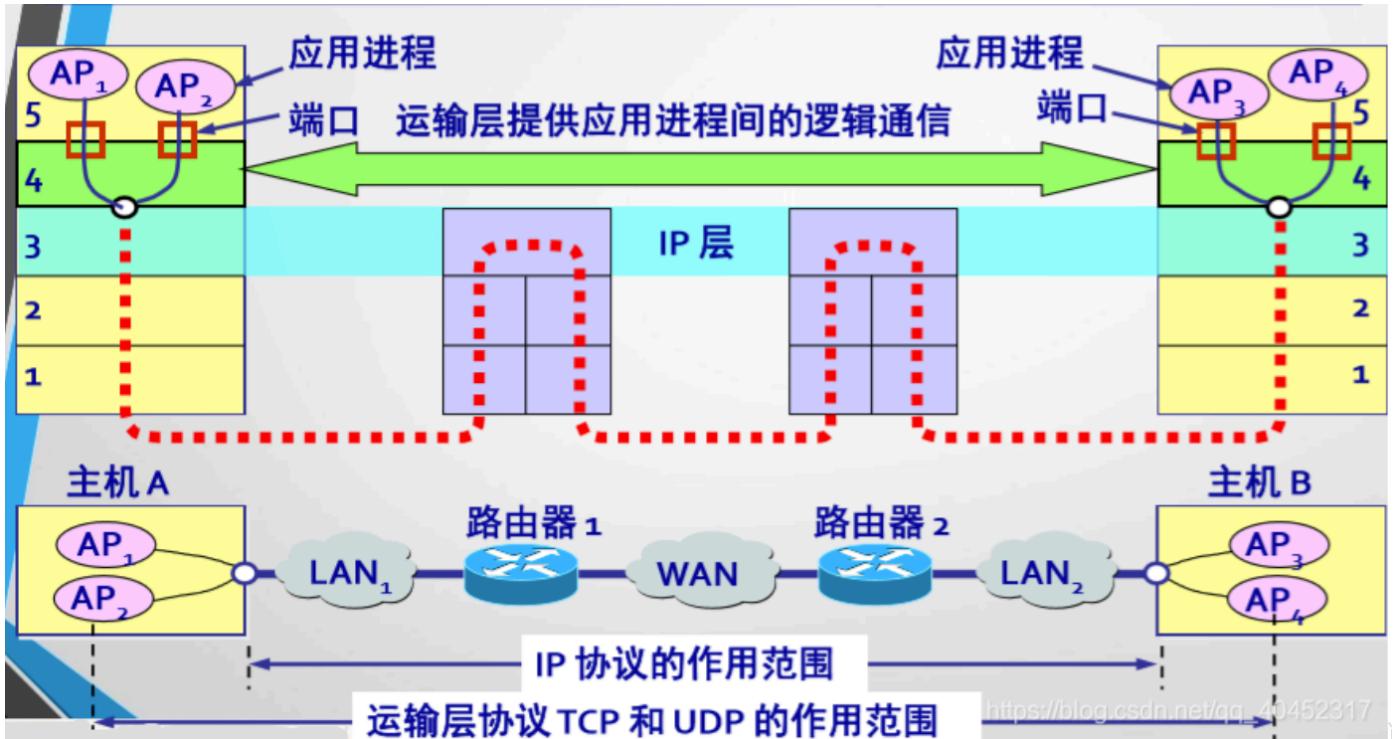


图 3-1 运输层在应用程序进程间提供逻辑的而非物理的通信

3.1 transport-layer services and protocols

- 为运行在不同主机上的应用进程提供 **logical end-to-end connection** (逻辑通信), 逻辑通信的意思是“好像是这样通信, 但事实上并非真的这样通信”。
- Transport layer protocols run in **end systems** (传输协议运行在端系统)
 - 发送方: breaks application messages into segments, passes to network layer (将应用层的报文分成报文段, 传递给网络层)
 - 接收方: reassembles segments into messages, passes to application layer (将报文段重组成报文, 然后传递给应用层)



- 网络应用程序可以使用多种的运输层协议 (Internet传输层协议有两种, 即 TCP 和 UDP). 每种协议都能为调用的应用程序提供一组不同的运输层服务。
- **TCP: reliable, data stream, in-order delivery**
 - 多路复用、解复用
 - congestion control (拥塞控制)
 - flow control (流量控制)
 - connection setup (建立连接)
- **UDP: unreliable, datagram, unordered delivery**
 - 多路复用、解复用
 - extension of “best-effort” IP (从network layer的主机到主机变成了进程到进程, 除此之外, 就没有添加更多的其它额外服务了)

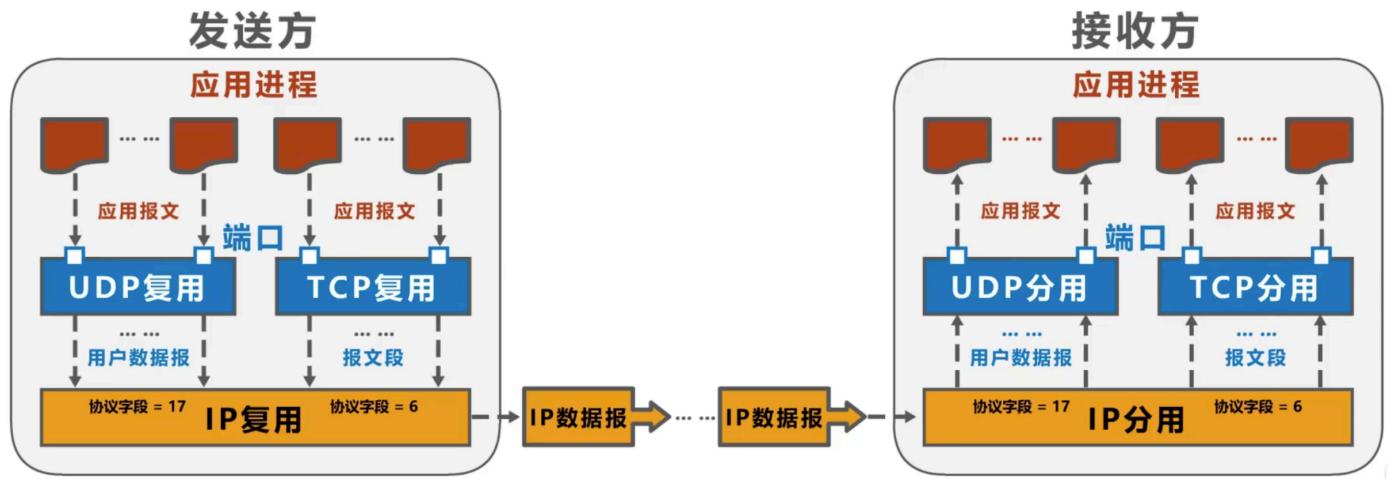
都不提供的服务: delay guarantees(延时保证), bandwidth guarantees(带宽保证)

3.2 multiplexing and demultiplexing

在一台主机中经常有多个应用进程同时分别和另一台主机中的多个应用进程通信, 一个进程(作为网络应用的一部分)有一个或多个 socket。这表明运输层有一个很重要的功能——复用 (multiplexing)和分用(demultiplexing)。

- **复用 (multiplexing):** 从多个 socket 接收来自多个进程的message, 根据 socket 对应的IP地址和端口号等信息对报文段用头部加以封装 (该头部信息用于以后的解复用), 然后将报文段传递到网络层
- **分用/解复用 (demultiplexing):** 根据报文段的头部信息中的IP地址和端口号将接收到的报文段发给正确的 socket (和对应的应用进程)。

对接收到的报文进行差错检测。网络层的IP数据报首部中的检验和字段, 只对首部差错进行检验。根据应用程序的不同需求, 运输层需要有两种不同的运输协议, 即面向连接的TCP和无连接的UDP。



运输层向高层用户屏蔽了下面网络核心的细节（如网络拓扑、所采用的路由选择协议等），它使应用进程看见的就是好像在两个运输层实体之间有一条端到端的逻辑通信信道。

Connectionless demux

无连接(UDP)多路解复用

创建套接字：

服务器端：

```
serverSocket=socket(PF_INET,
    SOCK_DGRAM,0);
```

serverSocket和Sad指定的端口号捆绑

客户端：

UDP和TCP不同

```
ClientSocket=socket(PF_INET,
    SOCK_DGRAM,0);
```

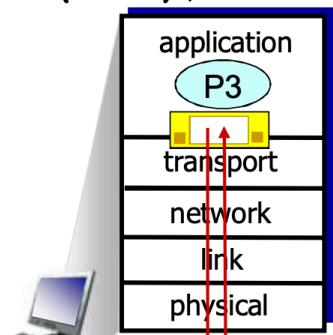
没有Bind, ClientSocket和os为之分

配的某个端口号捆绑（客户端使用什
么端口号无所谓，客户端主动找服务
器）

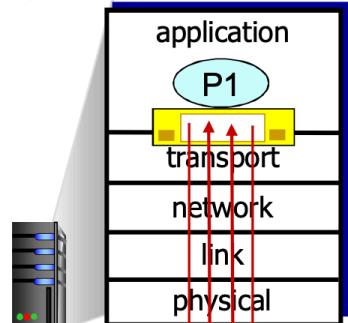
- 在接收端，UDP套接字用二元组标识（目标IP地址、目标端口号）
- 当主机收到UDP报文段：
 - 检查报文段的目标端口号
 - 用该端口号将报文段定位给套接字
- 如果两个不同源IP地址/源端口号的数据报，但是有相同的目标IP地址和端口号，则被定位到相同的套接字

PID	Socket	Des IP	Des Port
12345	77888	202.38.64.1	80
12346	77999	202.38.64.1	11010
.....			

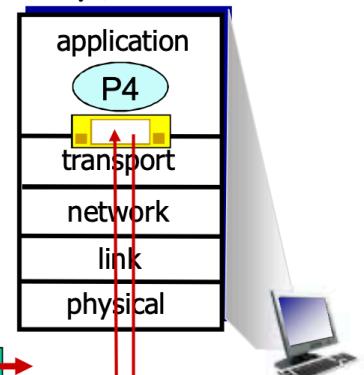
```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```



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



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



source port: 9157
dest port: 6428

source port: 6428
dest port: 9157

source port: ?
dest port: ?

source port: ?
dest port: ?

Connection-oriented demux

面向连接(TCP)的多路复用

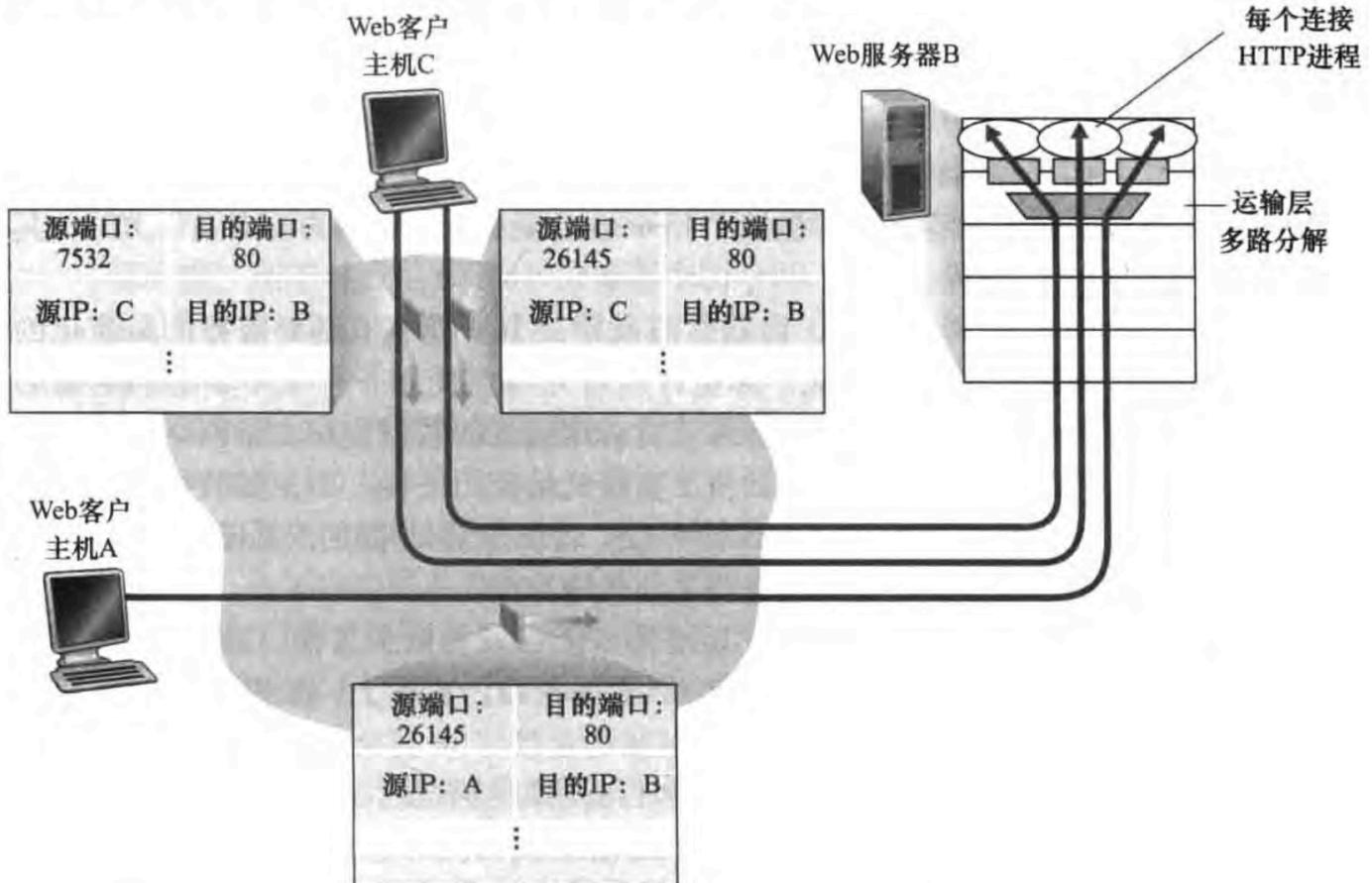


图 3-5 两个客户使用相同的目的端口号（80）与同一个 Web 服务器应用通信

3.3 connectionless transport: UDP

UDP只在IP的数据报服务之上增加了很少一点的功能：

- 复用和分用的功能
- 差错检测的功能，发现错误的就扔掉

虽然UDP用户数据报只能提供不可靠的交付，但UDP在某些方面有其特殊的优点。

UDP的主要特点

- **connectionless:** 发送数据之前不需要建立连接，因此减少了开销和发送数据之前的时延。UDP发送端和接收端之间没有握手，每个UDP报文段都被独立地处理
- **best-effort:** 尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的连接状态表。
- **面向报文:** UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。UDP一次交付一个完整的报文。
- **没有拥塞控制:** 因此网络出现的拥塞不会使源主机的发送速率降低。这对某些实时应用是很重要的。很适合对时延要求较高的多媒体通信的要求。
- **没有流量控制:** 应用->传输的速率= 主机->网络的速率
- **small header size:** 只有8个字节，比TCP的20个字节的首部要短。

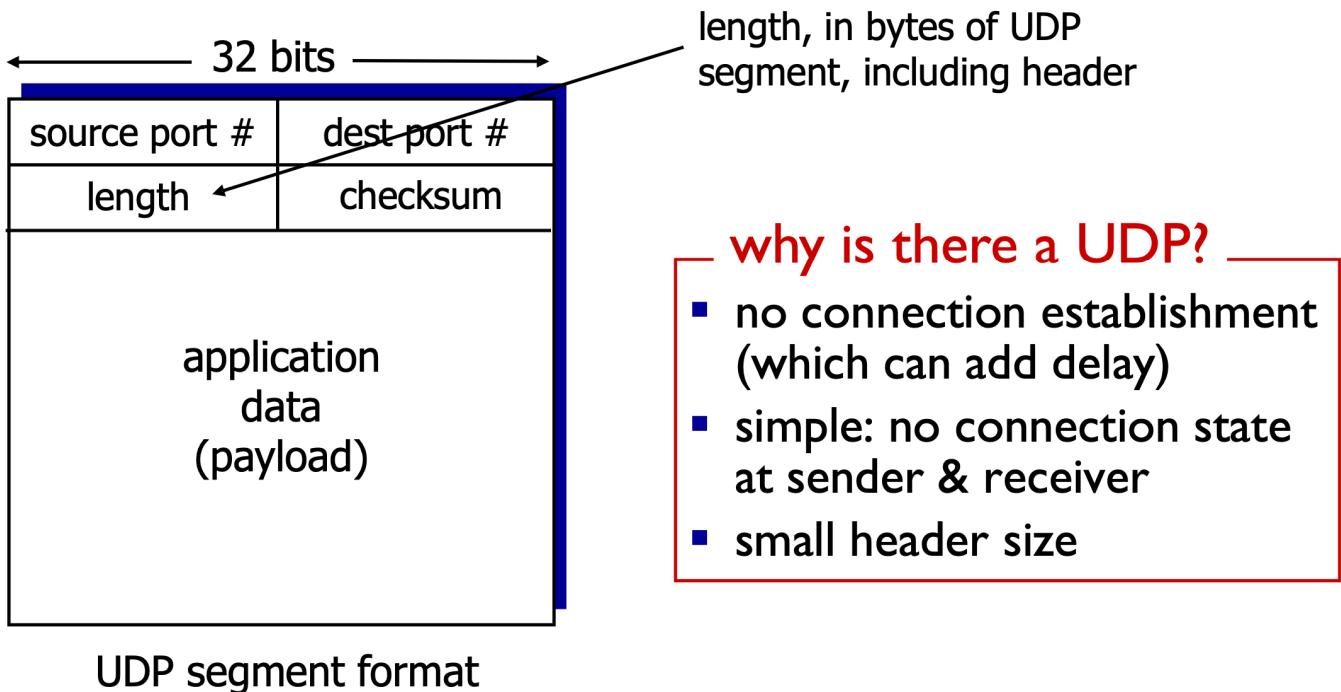
- 支持一对一、一对多、多对一和多对多的交互通信。

发送方 UDP 对应用程序交下来的报文，在添加首部后就向下交付 IP 层。UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。应用层交给 UDP 多长的报文，UDP 就照样发送，即一次发送一个报文。

接收方 UDP 对 IP 层交上来的 UDP 用户数据报，在去除首部后就原封不动地交付上层的应用进程，一次交付一个完整的报文。应用程序必须选择合适大小的报文。

若报文太长，UDP 把它交给 IP 层后，IP 层在传送时可能要进行分片，这会降低 IP 层的效率。若报文太短，UDP 把它交给 IP 层后，会使 IP 数据报的首部的相对长度太大，这也降低了 IP 层的效率。

UDP: segment header



UDP checksum

UDP checksum 作用：确定当 UDP 报文段从源到达目的地移动时，其中的比特是否发生了改变（只是检测错误，检测出来就丢弃这个报文，但是不能恢复这个报文）

发送方:

- 将报文段的内容视为16比特的整数
- 校验和：报文段的加法和（1的补运算）
- 发送方将校验和放在UDP的校验和字段

接收方:

- 计算接收到的报文段的校验和
- 检查计算出的校验和与校验和字段的内容是否相等：
 - 不相等---检测到差错
 - 相等---没有检测到差错，但也许还是有差错
 - 残存错误

Internet checksum: example 01

注意:当数字相加时，在最高位的进位要回卷，再加到结果上（回卷 wraparound：前面溢出来的数，拉到最后再往上加）

目标端:校验范围+校验和=1111111111111111 通过校验，否则没有通过校验

注:求和时，必须将进位回卷到结果上

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
																
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	1

Internet checksum: example 02

假定我们有下面 3 个 16 比特的字：

0110011001100000
0101010101010101
1000111100001100

这些 16 比特字的前两个之和是：

0110011001100000
0101010101010101
—————
1011101110110101

再将上面的和与第三个字相加，得出：

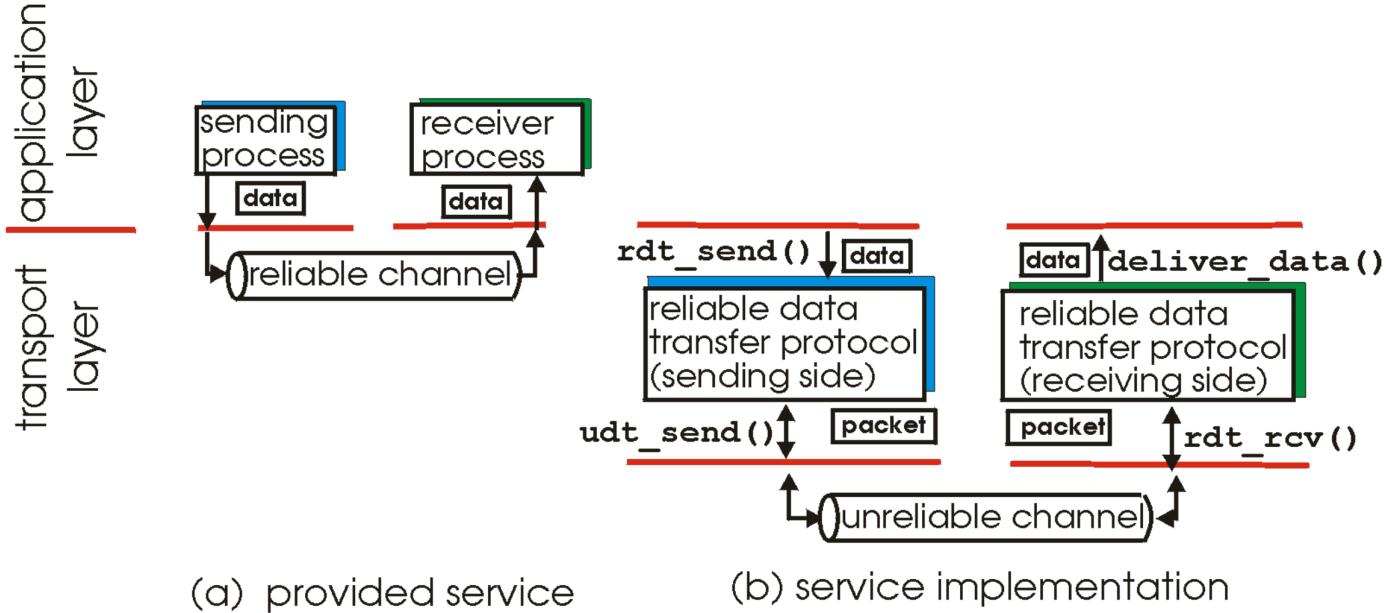
1011101110110101
1000111100001100
—————
0100101011000010

- 注意到最后一次加法有溢出，它要被回卷。
- 该和 0100101011000010 的反码运算结果是 1011010100111101 (反码运算就是将所有的 0 换成 1 所有的 1 转换成 0)，这就变为了检验和。
- 在接收方，全部的4个16比特字(包括检验和) 加在一起，即 1011010100111101 (checksum) + 另外3个16比特字。
- 如果该分组中没有引入差错，则显然在接收方处该和将是1111111111111111。如果这些比特有一个是 0，那么我们就知道该分组中已经出现了差错。

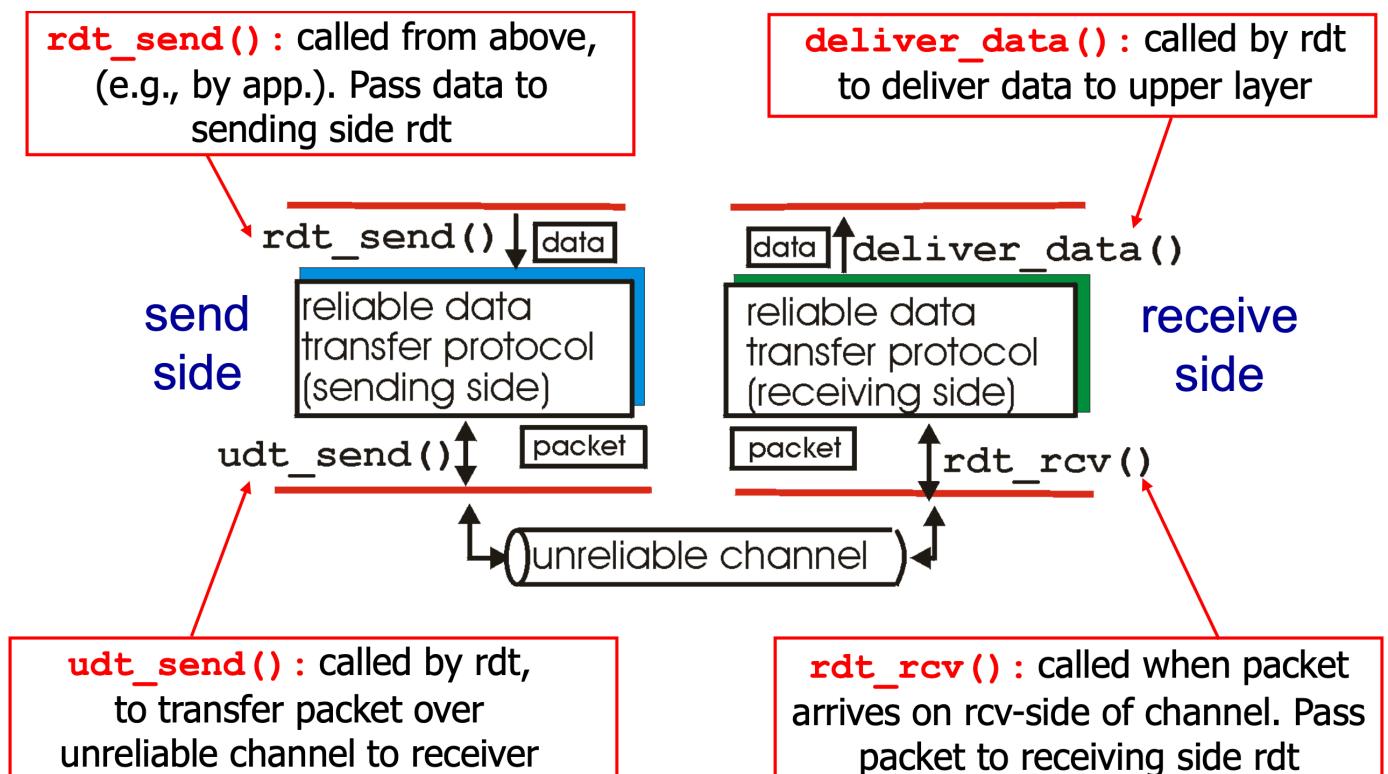
3.4 principles of reliable data transfer

可靠数据传输(rdt)的原理

数据的可靠传输是计算机网络中的通用概念，也是UDP和TCP的基石。计算机五层模型中，上层需要借助下层提供的功能来完成数据的传输，那么如果下层不可靠，我们该如何保证数据的可靠传输？



(b) service implementation



- 通过调用 `rdt_send()` 函数，上层可以调用数据传输协议的发送方。它将要发送的数据可靠交付给位于接收方的较高层。
- 在接收端，当分组从信道的接收端到达时，将调用 `rdt_rcv()`。
- 当 rdt 协议想要向较高层交付数据时，将通过调用 `deliver_data()` 来完成
- 除了交换含有待传送的数据的分组之外，rdt 的发送端和接收端还需往返交换控制分组。rdt 的发送端和接收端都要通过调用 `udt_send()` 发送分组给对方

接下来会一步步假设，一步步的暴露问题，来看看可靠性传输RDT是如何演进的？只考虑 单向数据传输 (unidirectional data transfer)。但控制信息是双向流动的 (But control info will flow on both directions) 因为有反馈机制。

有限状态机 (Finite-State Machine, FSM) 来描述发送方和接收方

RDT 1.0 All reliable

scenario: 下层的信道是完全可靠的 (没有比特差错, 没有分组丢失)

发送方和接收方的FSM: 发送方将数据发送到下层信道, 接收方从下层信道接收数据

所以rdt1.0做的工作就是封装和解封装

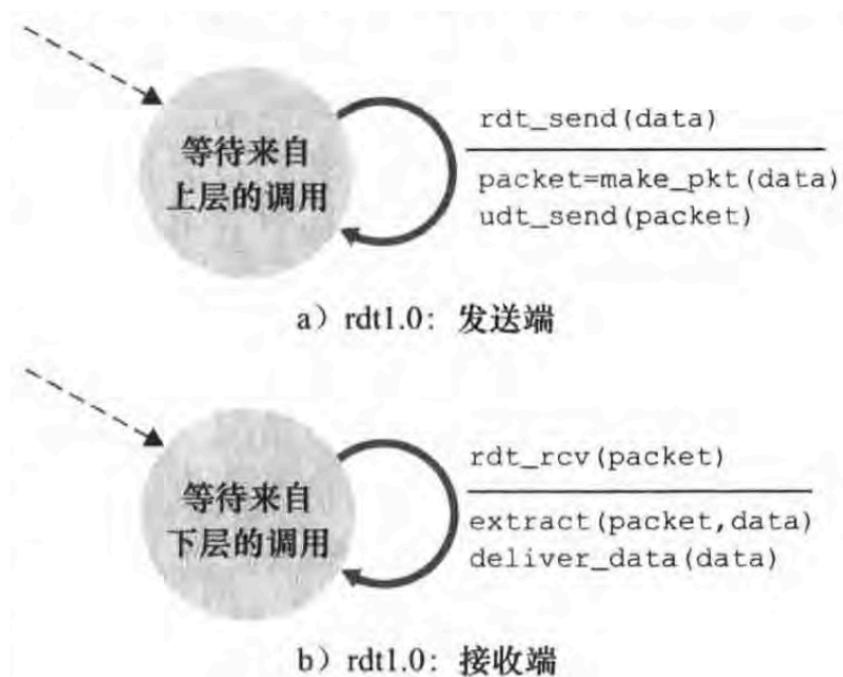


图 3-9 rdt1.0: 用于完全可靠信道的协议

RDT 2.0 Bit errors

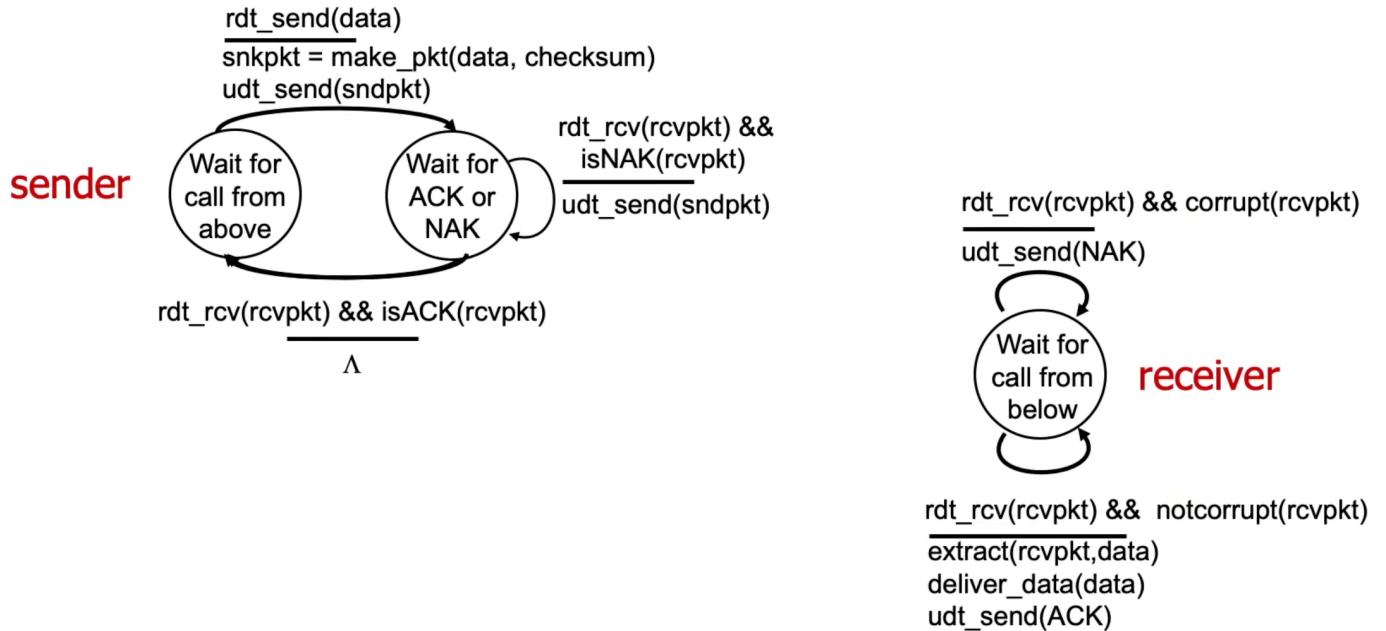
scenario: 下层信道可能出现差错 (underlying channel may flip bits in packet 分组中的比特可能翻转, 0变为1, 1变为0)

应对手段:

- **checksum:** 用校验和的方式进行差错检测 (error detection)
- **feedback:** send control msgs (ACK,NAK) from receiver to sender
 - **acknowledgements (ACKs):** 接收方显式的告诉发送方分组已经被接收 (no error scenario)
 - **negative acknowledgements (NAKs):** 接收方显式地告诉发送方分组出现了差错 (error scenario, 发送方接收到NAK后, 会重传分组)

过程描述: 发送方发送packet到下层信道, 接收方接收, 通过校验和发现数据有差错, 返回NAK,发送方再次发送原packet; 无差错返回ACK, 发送方发送新的pactet。所以Rdt2.0新机制就是采用了差错控制编码进行差错检测

stop and wait (停等协议): 发送方发送一个分组, 然后等待接收方的应答。只有应答之后才会发送下一个分组。



存在问题 (fatal flaw): ACK/NAK分组也可能出错/受损

所以引出了 Rdt 2.1

RDT 2.1 ACK/NAK corrupted

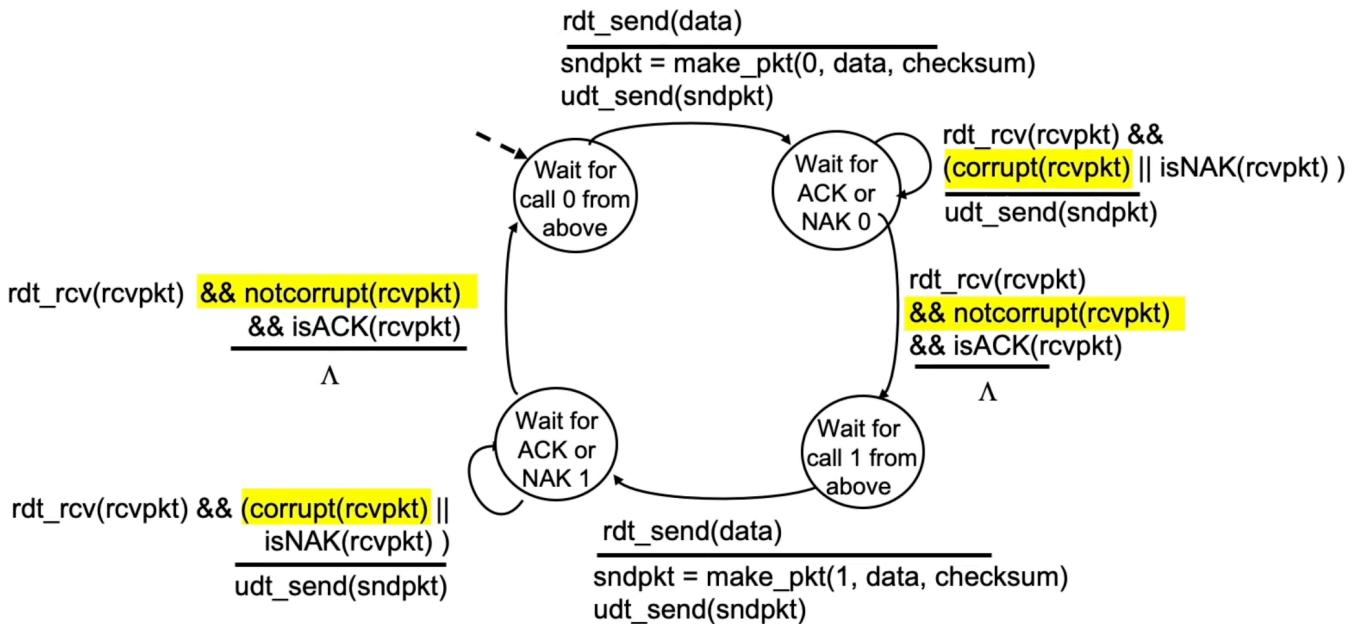
scenario: 2.0 中接收方返回的 ACK/NAK 有可能出错，如果重传数据，还有可能重复。

应对手段：

- **加上序号 (adds sequence number):** 给每个packet加上序号，哪怕客户端接收到重复数据，也能判断到重复。 (比如说， 我们想要传的第一组分组接受方返回了很模糊的信号，我们的发送方管他三七二十一就再次发送这个分组，并且标序号为0证明他是上一个的重复。接收方可以发现这次重复的分组，如果他之前的分组有错的话就直接替换，如果没有错的话就丢弃)
- 停止等待：发送方只发送一个分组，然后等待接收方的应答的方式称为停止等待协议

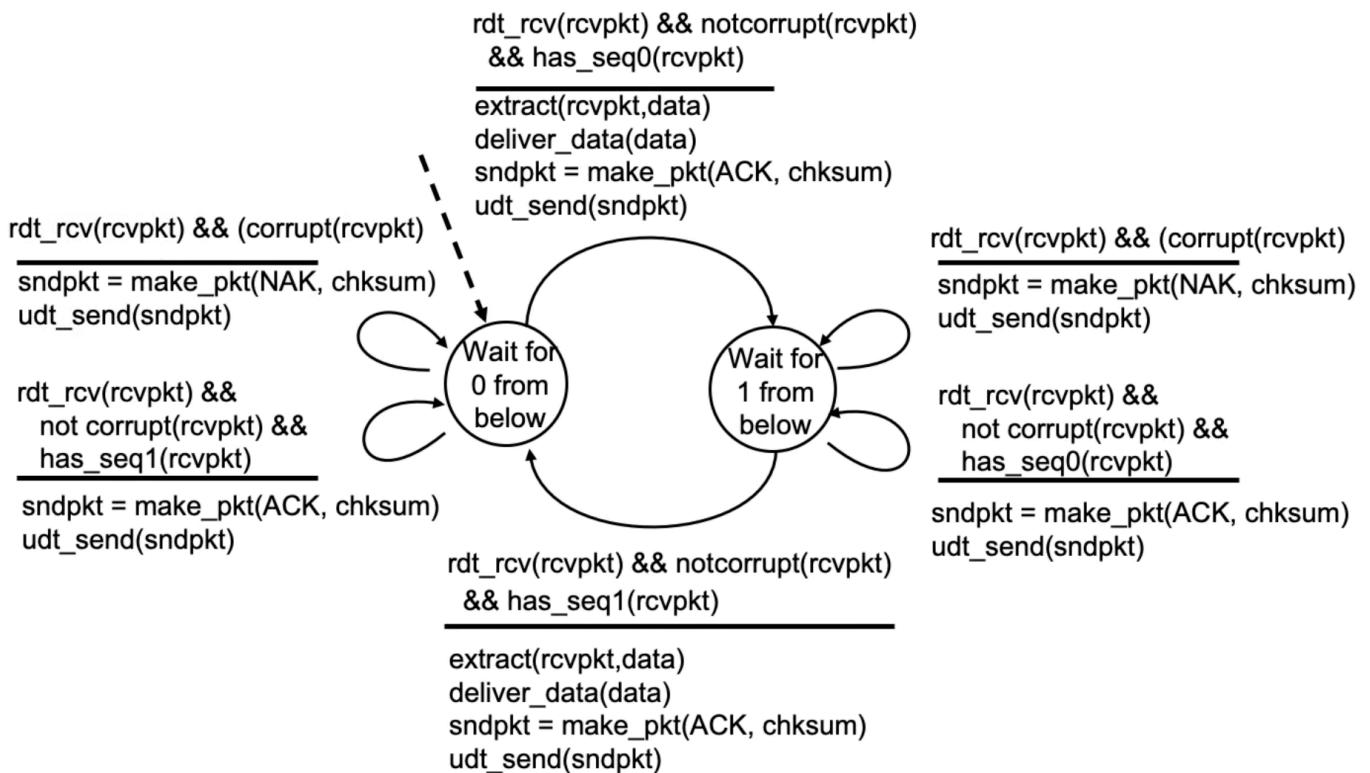
Sender:

- 在分组中加入序列号，两个序列号(0, 1)就足够了，一次只发送一个未经确认的分组
- 必须检测ACK/NAK是否出错(需要EDC)
- 状态数变成了两倍，必须记住当前分组的序列号为0还是1

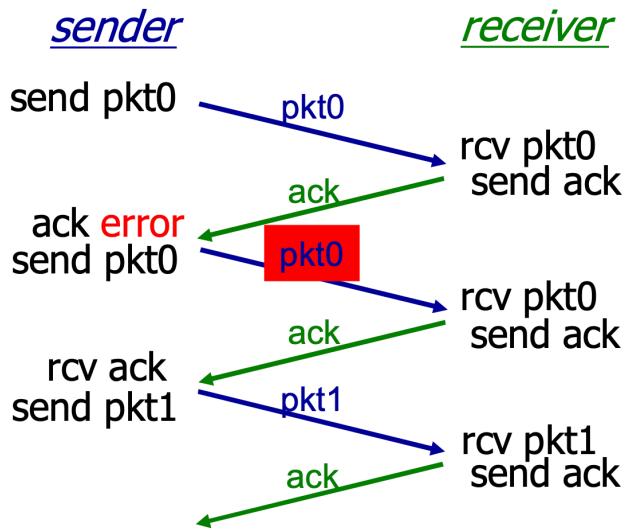


Receiver:

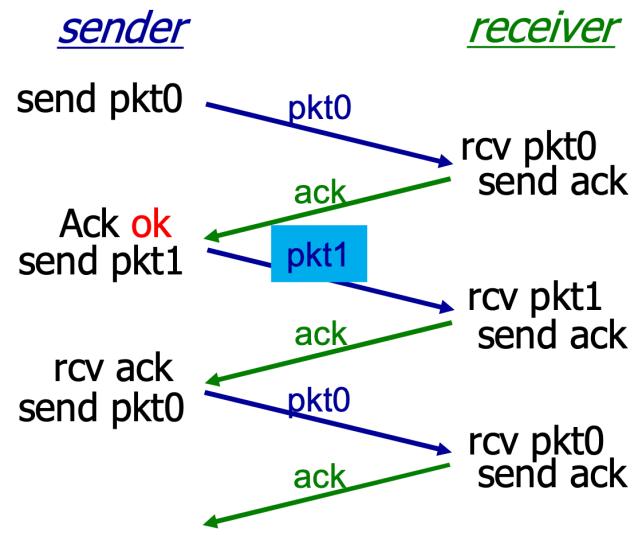
- 必须检测接收到的分组是否是重复的，状态会指示希望接收到的分组的序号为0还是1
- 但是 receiver 并不知道 sender 是否正确收到了其ACK/NAK



- receiver 不知道它最后发送的 ACK/NAK 是否被正确地收到
- sender 不对收到的 ACK/NAK 给确认，没有所谓的确认的确认；
- 比如， receiver 发送 ACK，如果后面 receiver 收到的是：老分组p0，则 ACK 错误。下一个分组P1，ACK 正确



(a) ack error



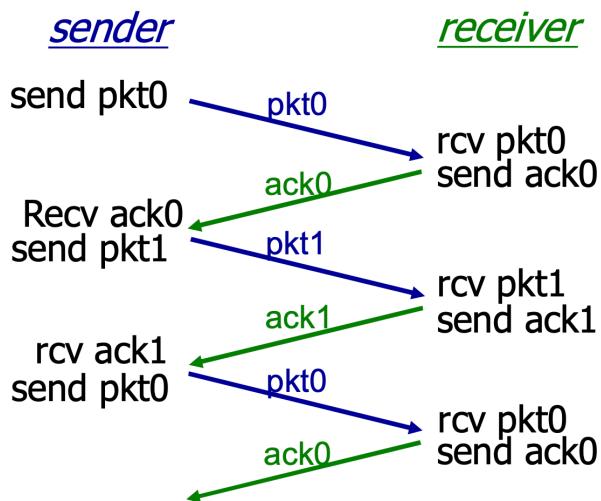
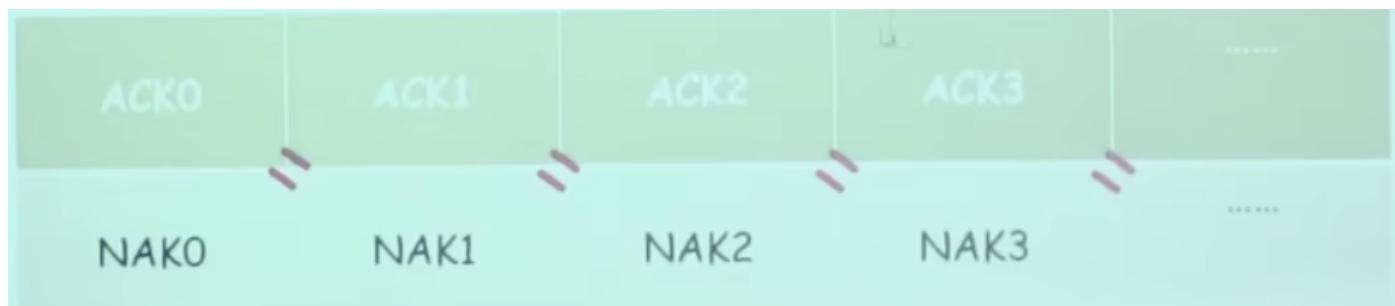
(b) ack right

RDT 2.2: NAK-free protocol

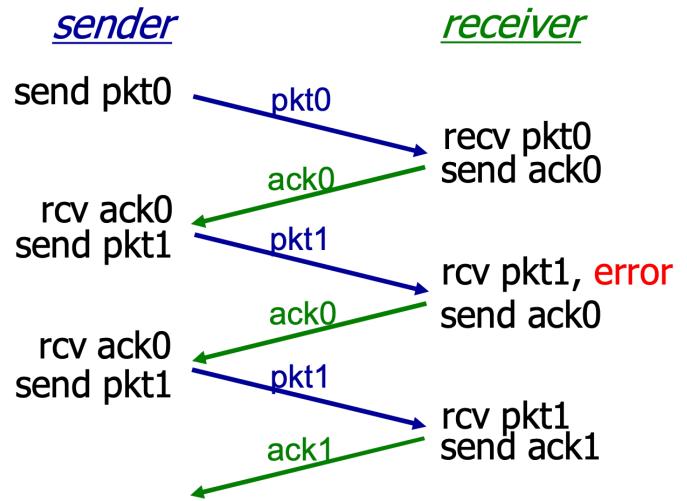
scenario: 发送方需要识别ACK和NACK两套状态管理，太复杂了

应对手段：

- 取消NACK：取消了NACK的方式，那么接收方怎么告诉发送方数据错了呢？可以通过返回ACK+上一个packet的序号的方式返回。
- 重复ACK：当接收方收到重复的ACK时，说明下一个packet发送失败了。接收方会重发当前ACK序号后的报文



(a) No error (packet or ack error)



(b) packet error

存在问题：返回的ACK也有可能丢失，发送方就会一直等待确认。

所以引出了 Rdt3.0

RDT 3.0 Errors & loss

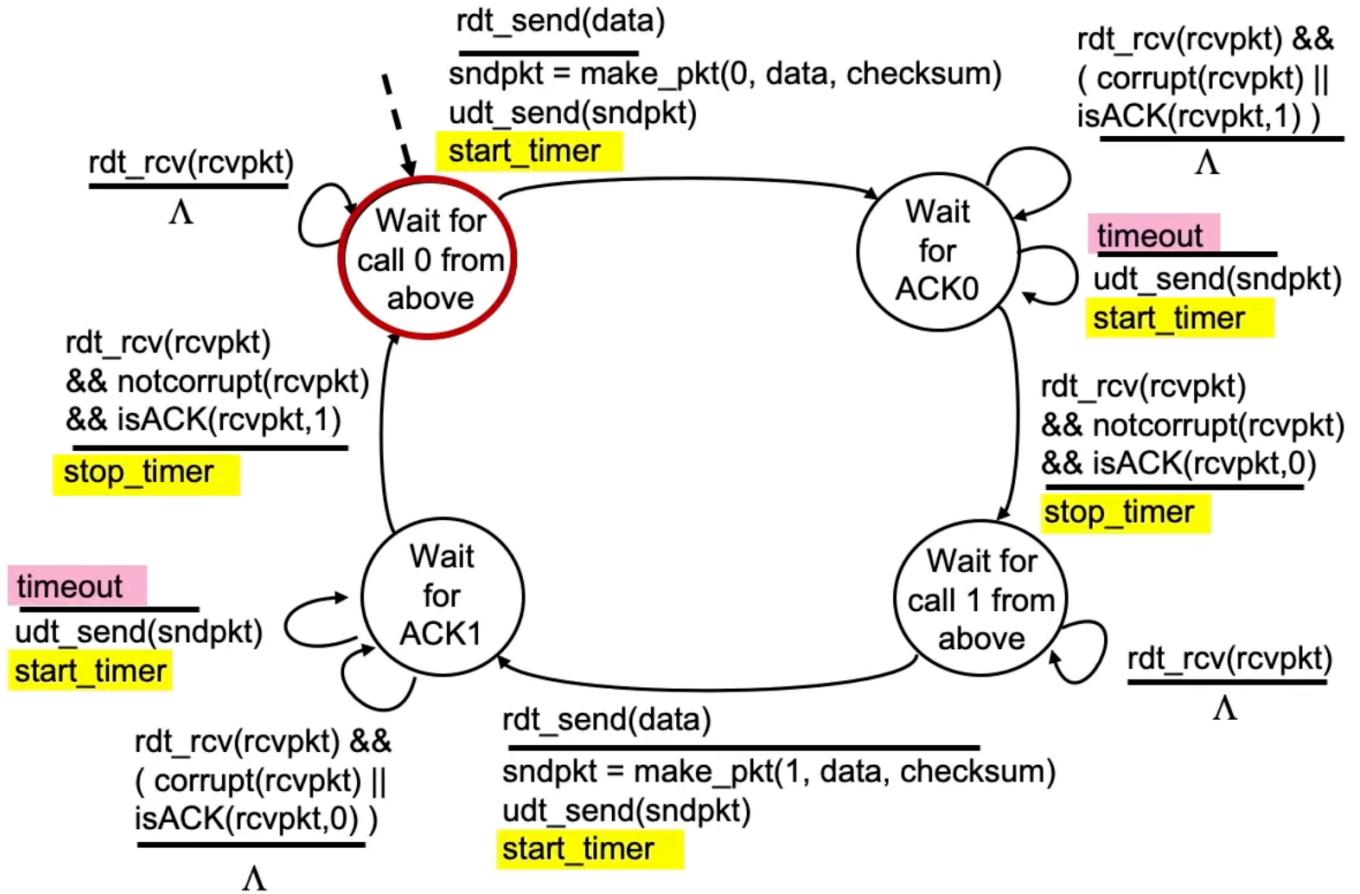
scenario: 发送方会一直等待ACK，但是ACK有可能丢失，除了比特受损外 (Error)，底层信道还会丢包 (Loss)

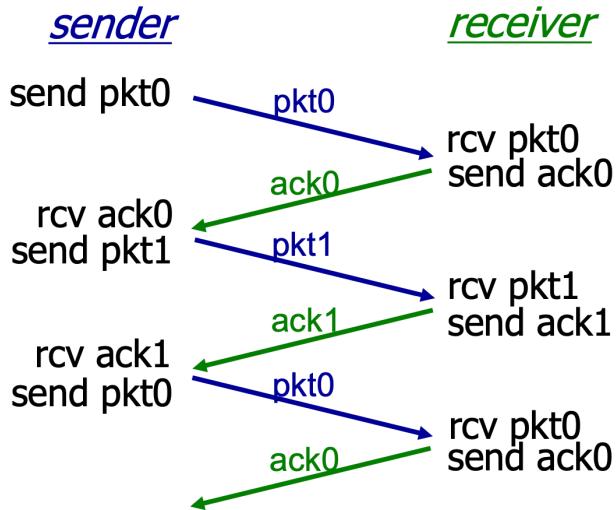
应对手段：

- 倒计数定时器 (countdown timer): 对应时间内没收到ACK就直接触发超时重传机制 (认为可能传输过程中 loss 了). 发送端超时重传:如果到时没有收到ACK->重传 (链路层的timeout时间确定的, 传输层timeout时间是适应式的)

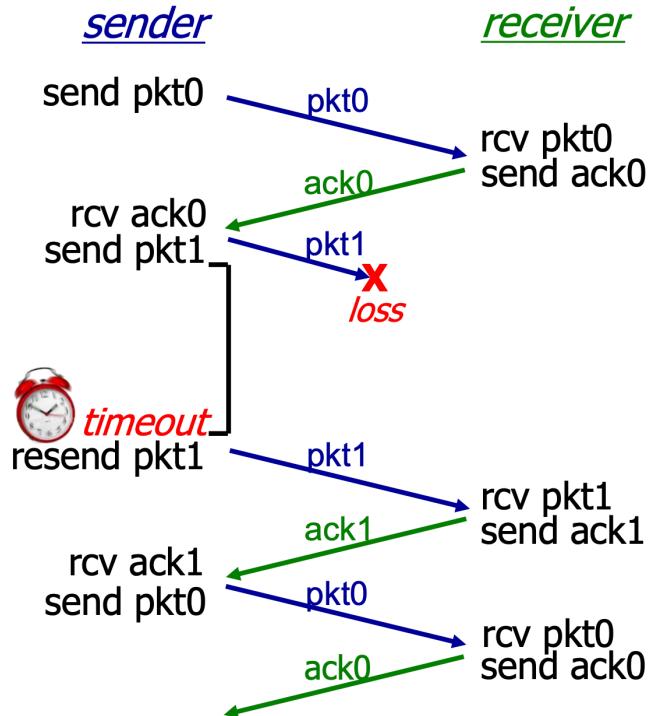
存在问题：RDT3.0以及之前，一直采用停止等待协议，也就是一个包没收到响应就不会发送下一个，信道利用率太低。由此引入了流水线协议。

sender:



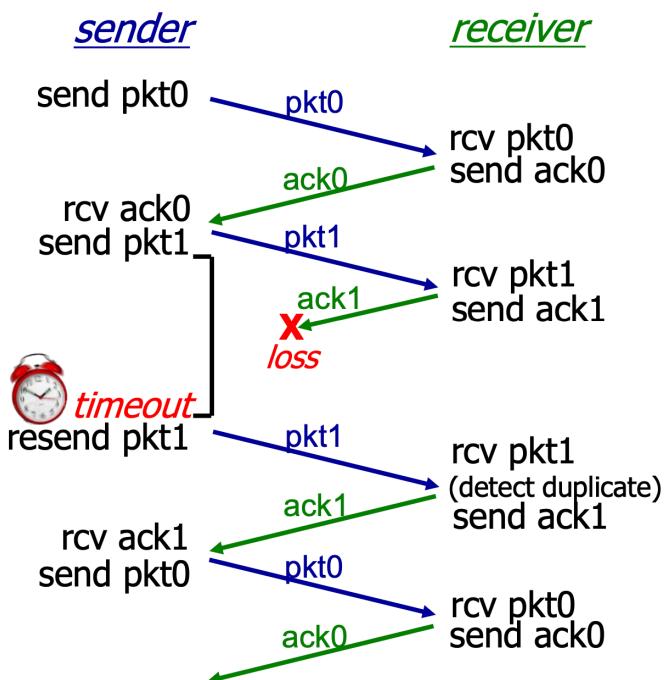


(a) no loss

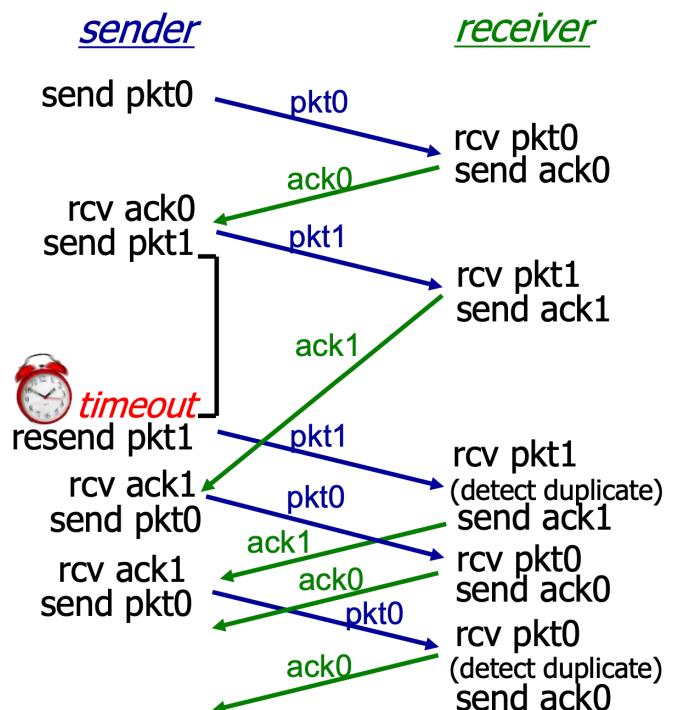


(b) packet loss

- 过早超时(延迟的ACK)也能够正常工作; 但是效率较低, 一半的分组和确认是重复的;
- 设置一个合理的超时时间也是比较重要的;



(c) ACK loss



(d) premature timeout/ delayed ACK

RDT 3.0 stop-and-wait operation

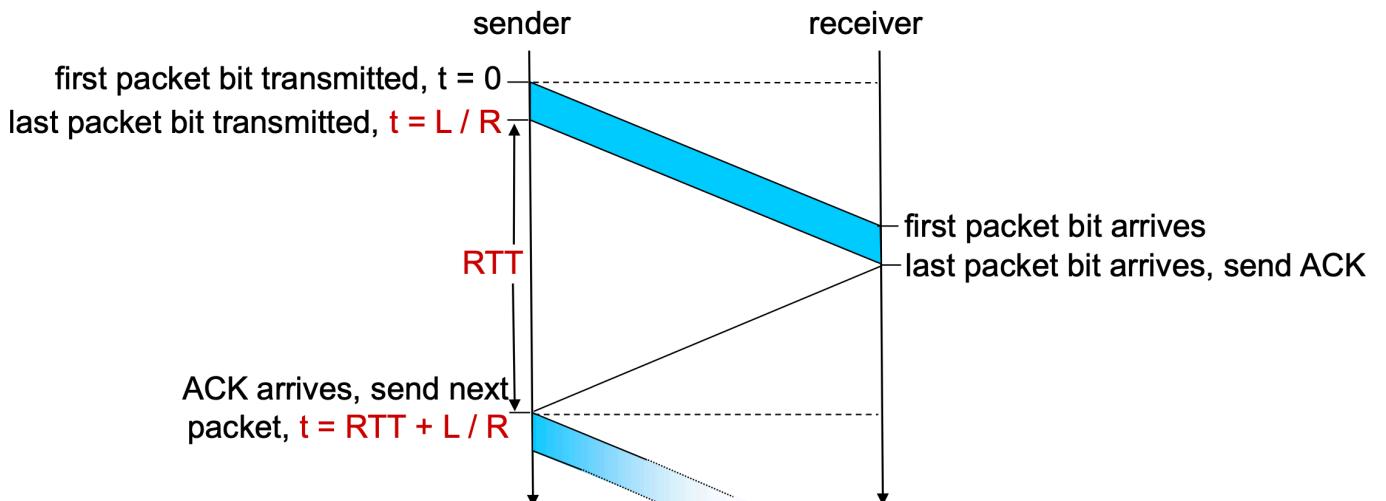
rdt3.0可以工作，但链路容量比较大的情况下，性能很差。链路容量比较大，一次发一个PDU的不能够充分利用链路的传输能力

example:

- 在这两个端系统之间的光速往返传播时延RTT大约为30毫秒。
- 假设 ACK 分组很小(以便我们可以忽略其发送时间)，接收方一旦收到一个数据分组的最后 1 比特后立即发送 ACK
- 假定彼此通过一条发送速率R为 1 Gbps (每秒 10^9 比特)的信道相连。
- 包括首部字段和数据的分组长 L 为 1000 字节 (8000 比特)，发送一个分组进入 1 Gbps 链路实际所需时间是：

□ 例：1 Gbps 的链路，15 ms 端-端传播延时，分组大小为 1 kB：

$$T_{transmit} = \frac{L \text{ (分组长度, 比特)}}{R \text{ (传输速率, bps)}} = \frac{8\text{kb/pkt}}{10^9 \text{ b/sec}} = 8\mu\text{s}$$



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

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

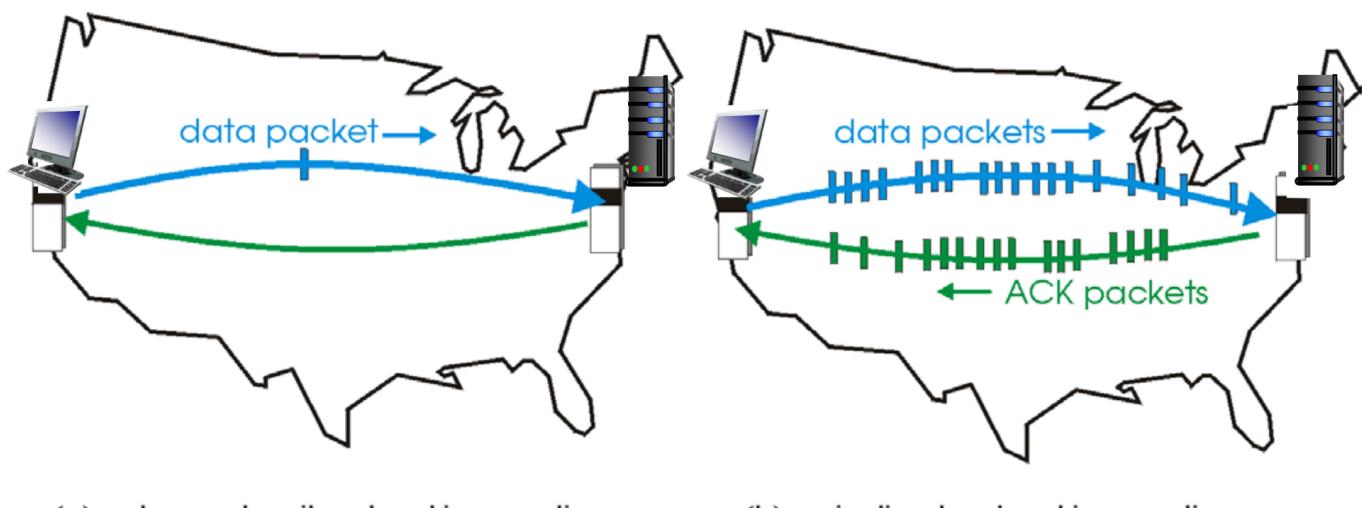
$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

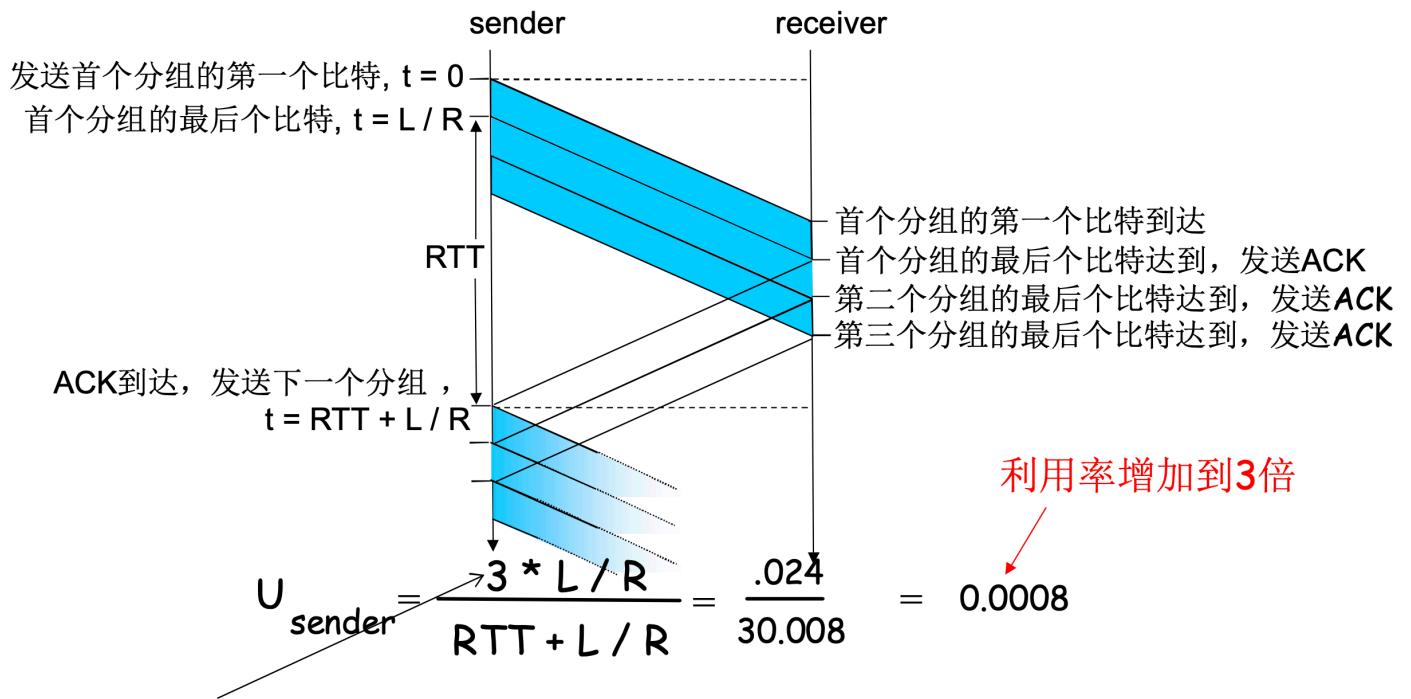
- U_{sender} : 利用率 - 忙于发送的时间比例
- 每30ms发送1KB的分组 $\rightarrow 270\text{kbps}=33.75\text{kB/s}$ 的吞吐量 (在1 Gbps 链路上)
- 瓶颈在于: 网络协议限制了物理资源的利用!

RDT 3.0 Pipelined protocols

流水线协议 (Pipelined protocols): 为了提高信道的利用率, 我们需要能够批量发送分组。当发送方窗口 >1 , 我们称之为流水线协议。

- range of sequence numbers must be increased ($[0, 1] \Rightarrow [0, N-1]$). 增加序号的范围, 用多个bit表示分组的序号
- 所需序号范围和对缓冲的要求取决于数据传输协议如何处理丢失、损坏及延时过大的分组。解决流水线的差错恢复有两种基本方法是 回退N步(Go-Back- N, GBN) 和 选择重传(Selective Repeat, SR)
- buffering at sender and/or receiver. 在发送方/接收方要有缓冲区
 - sender buffer: 缓冲那些已发送但没有确认的分组, 可能需要重传
 - receiver buffer: 缓存那些已正确接收的分组, 上层用户取用数据的速率 \neq 接收到的数据速率。接收到的数据可能乱序, 排序交付(可靠)





- 增加n,能提高链路利用率
- 但当达到某个n,其u=100%时,无法再通过增加n, 提高利用率
- 瓶颈转移了->链路带宽

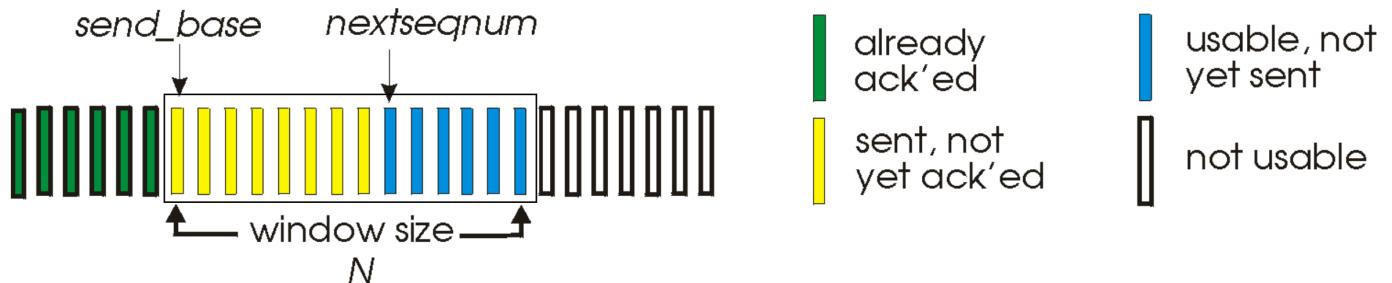
Transport Layer 3-49

GBN (Go-Back-N)

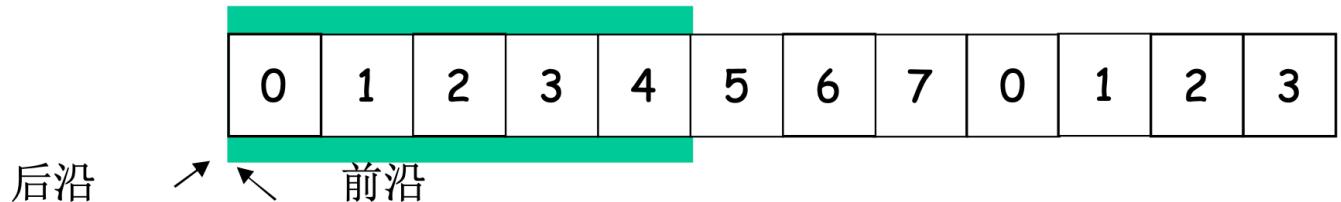
先明白一个概念 **WS (window size)** 代表可发送, 或者可接收的窗口的长度。

- 在 stop-and-wait operation 中, 发送方只能发送一个分组, 等待确认后, 再发送下一个, 所以: 发送方窗口 =1, 接收方窗口=1。
- 但是在 Pipelined protocols 中的 GBN (Go-Back-N), 那些已被发送但还未被确认的分组的许可序号范围可以被看成是一个在序号范围内长度为 N 的窗口。随着协议的运行, 该窗口在序号空间向前滑动。因此, N 常被称为窗口长度 (window size)。接收方窗口=1, 发送方窗口=N。
- GBN 协议也常被称为 滑动窗口协议 (*sliding-window protocol*)

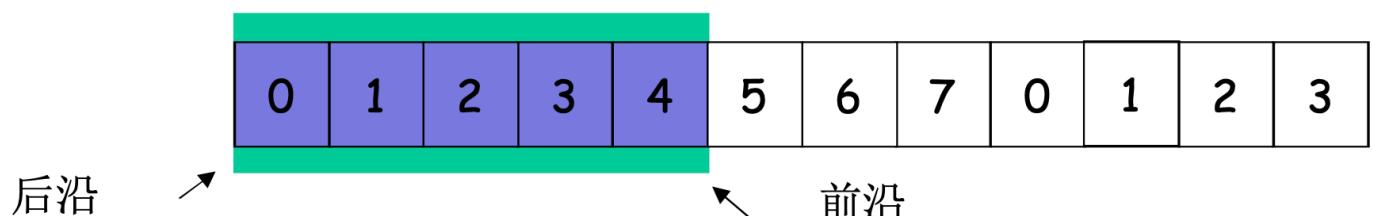
sending window



假设一开始没有发送任何一个分组, **window size = 5**



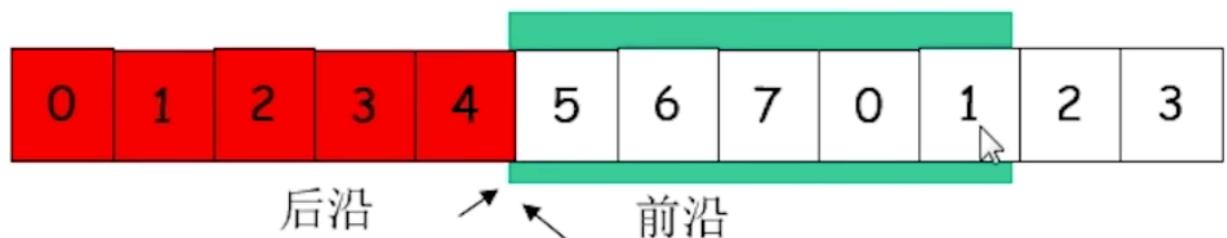
如果发送 packet 0 1 2 3 4, nextseqnum(前沿) 往后移5位:



收到 packet 0 的确认(ACK), send_base(后延) 往后移1位:



如果收到 packet 1 2 3 4 的确认(ACK), send_base(后延) 往后移4位:



receiving window

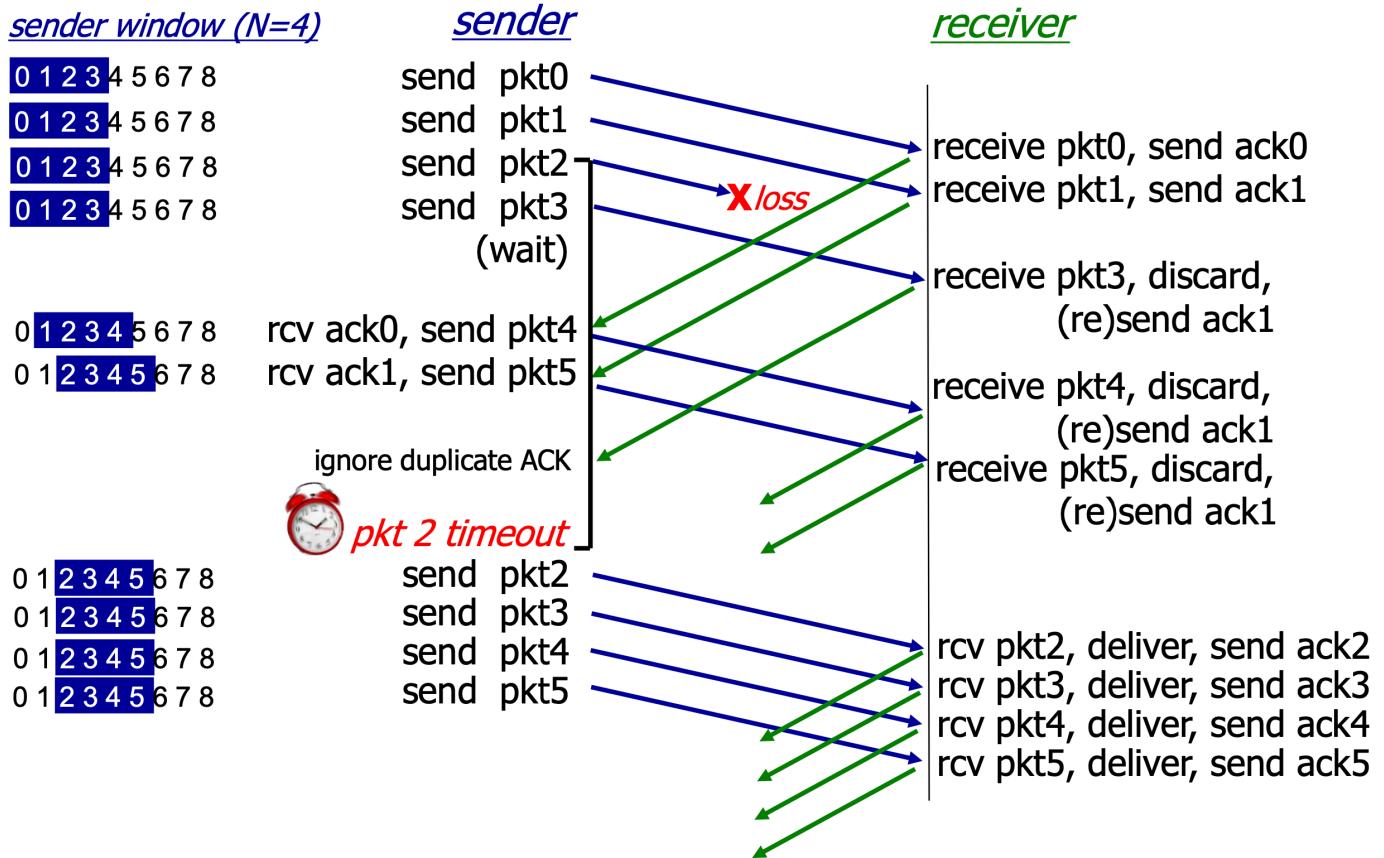
接收窗口 (receiving window) = 接收缓冲区

Receiver view of sequence number space:

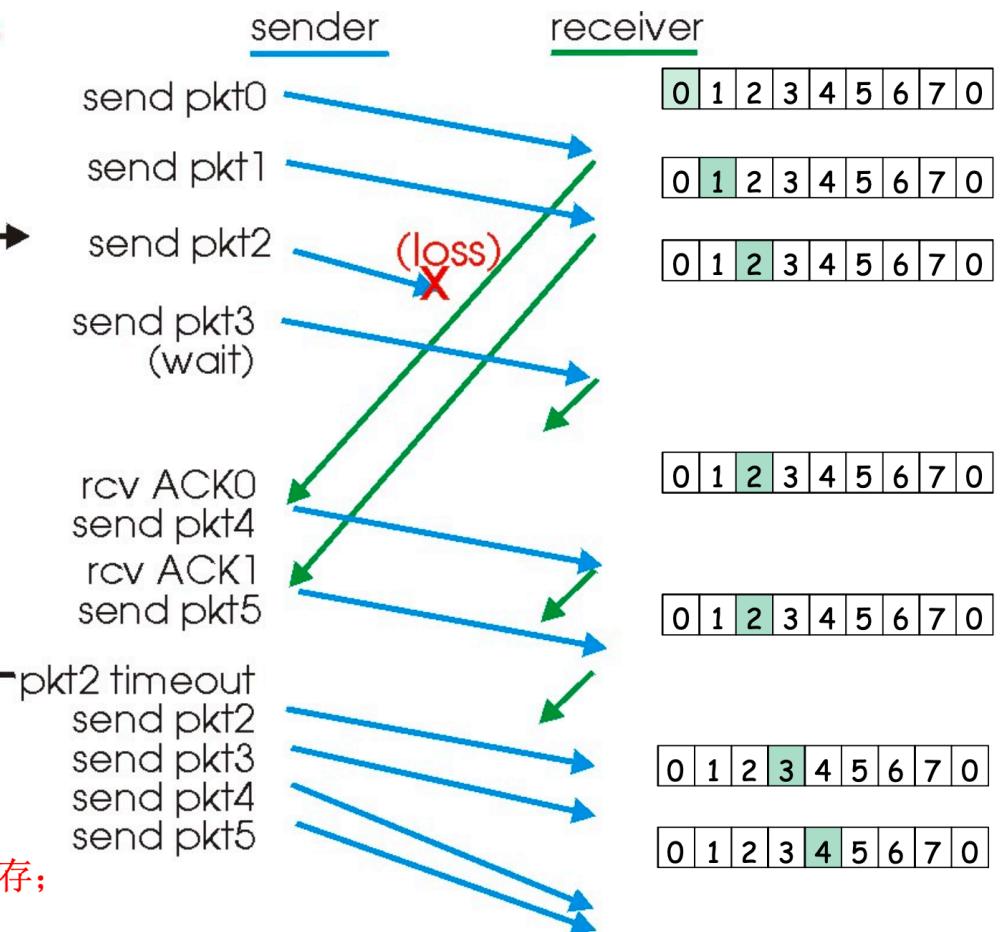
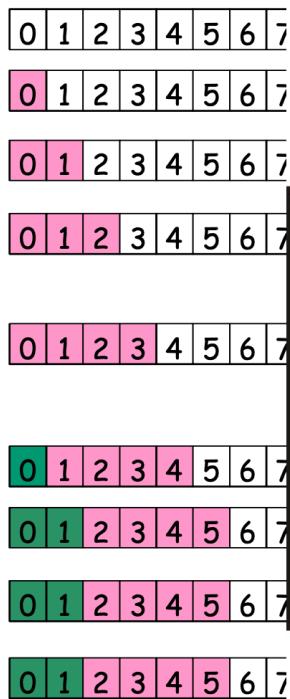


- **sender sets timer for oldest in-flight pkt:** 发送方只为最早的未经确认的分组维护一个定时器, 如果产生超时, 将重新发送该分组后的所有分组, 也就是顾名思义的回退N步。
- **discard (don't buffer) out-of-order pkt 丢弃乱序:** 接收方窗口只有1, 如果接收到比较大的序号分组, 都会选择丢弃。

- 在接收端，乱序的不缓存。因此哪个n分组丢失了GB到那个分组n，即使n以后的分组传送都是正确的



运行中的GBN

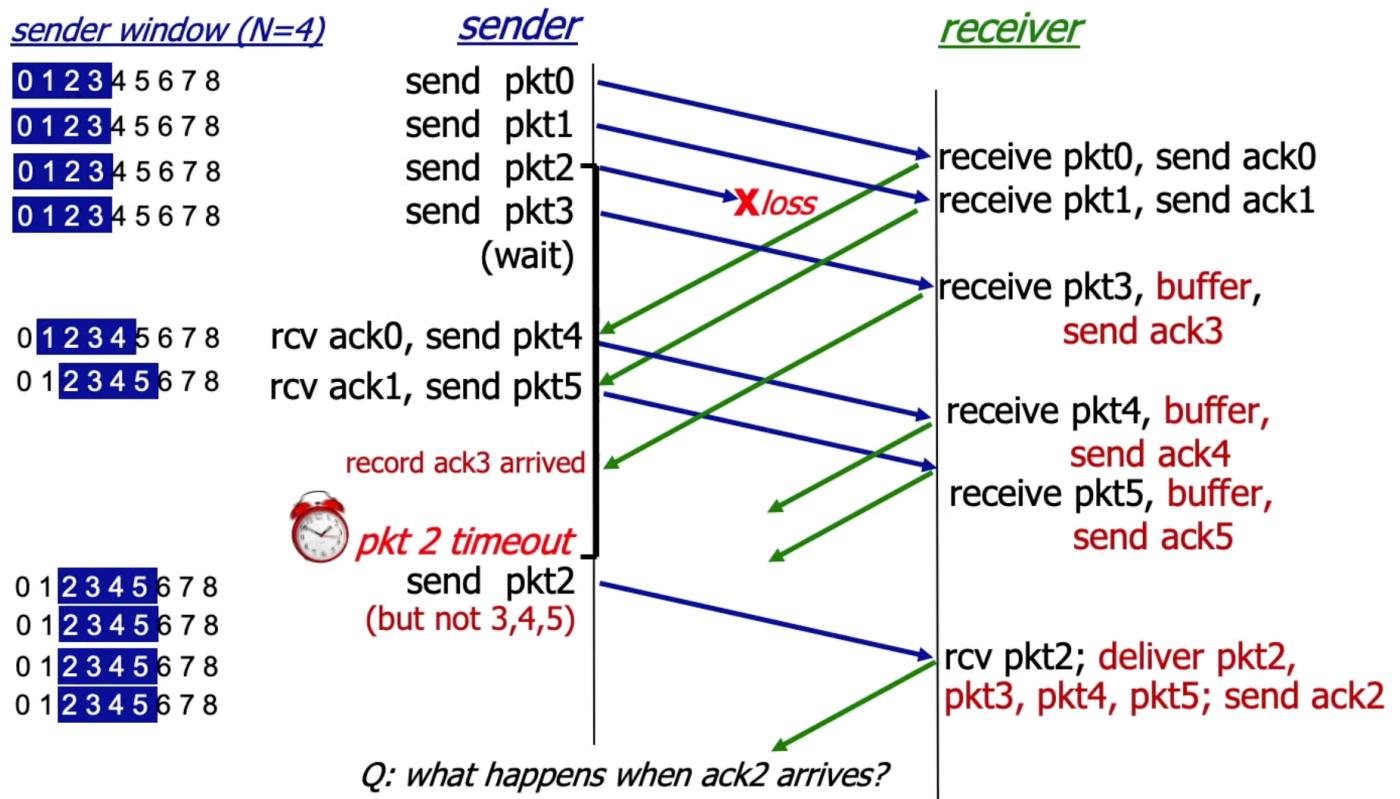
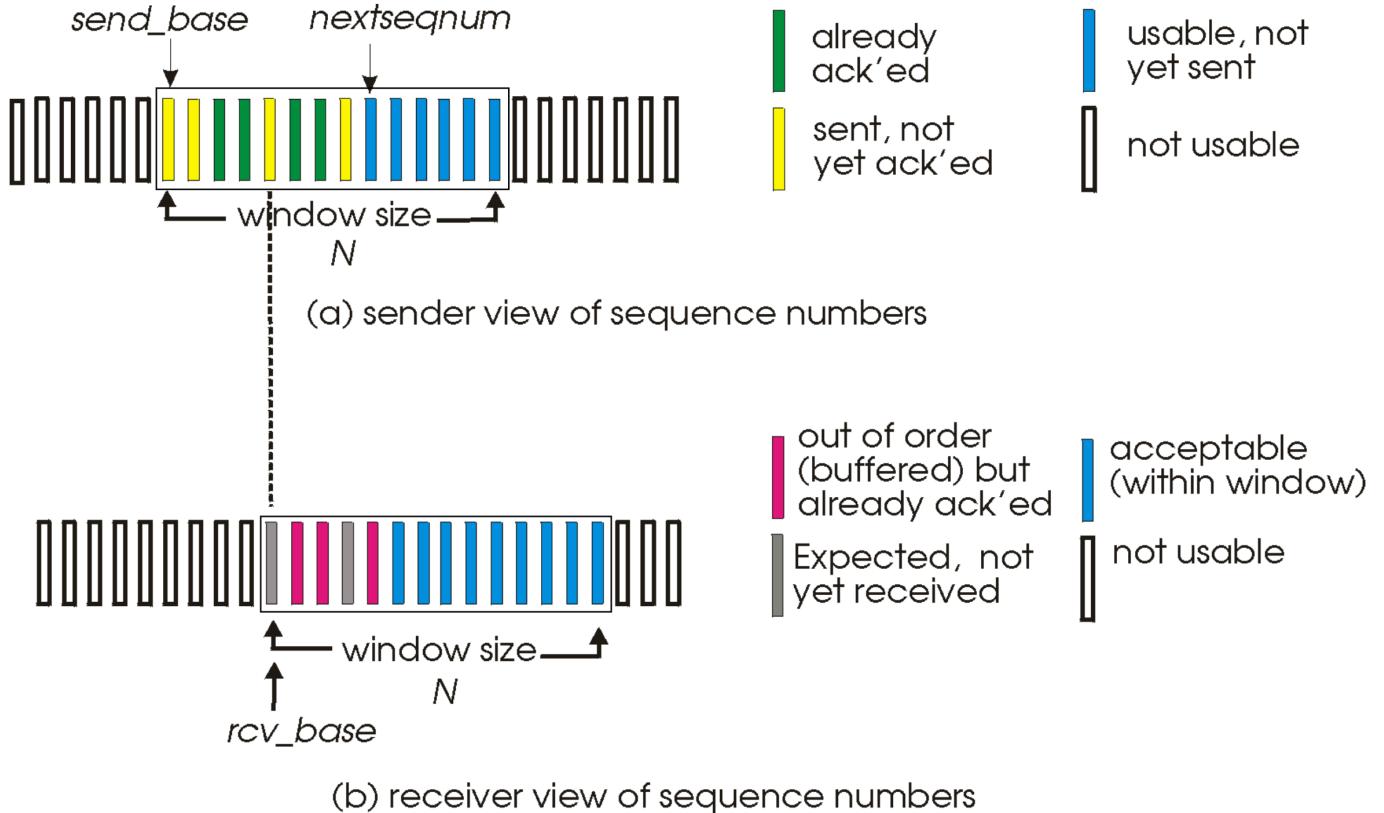


在接收端，乱序的不缓存；
因此哪个n分组丢失了
GB到那个分组n；
即使n以后的分组传送都是正确的

SR (Selective Repeat)

接收方窗口=N，发送方窗口=N。

- **sender maintains timer for each unacked packet:** 发送方为每个分组都设置了重传定时器，发送方只重发那些没有收到 ACK 的packets. 接收方单独确认所有正确接收的 packets.
- **out-of-order to buffer:** 接收方可以乱序接收分组，当最头部分组整体接收完毕，滑动窗口可以整体后移。收到乱序分组可以缓存packets，以便最终 in-order delivery to upper layer (按顺序交付给上层)。也就是说，packet 有序：交给上层。无序：先缓存，等有序了再交给上层



发送方

从上层接收数据:

- 如果下一个可用于该分组的序号可在发送窗口中，则发送

timeout(n):

- 重新发送分组n，重新设定定时器

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

- 将分组n标记为已接收
- 如n为最小未确认的分组序号，将base移到下一个未确认序号

接收方

分组n [rcvbase, rcvbase+N-1]

- 发送ACK(n)
- 乱序: 缓存
- 有序: 该分组及以前缓存的序号连续的分组交付给上层，然后将窗口移到下一个仍未被接收的分组

分组n [rcvbase-N, rcvbase-1]

- ACK(n)

其它:

- 忽略该分组

Pipelined Summary

□ 相同之处

- 发送窗口 >1
- 一次能够发送多个未经确认的分组

□ 不同之处

- GBN : 接收窗口尺寸=1
 - 接收端: 只能顺序接收
 - 发送端: 从表现来看, 一旦一个分组没有发成功, 如: 0,1,2,3,4; 假如1未成功, 2,3,4都发送出去了, 要返回1再发送; GB1
- SR: 接收窗口尺寸 >1
 - 接收端: 可以乱序接收
 - 发送端: 发送0,1,2,3,4, 一旦1未成功, 2,3,4已发送, 无需重发, 选择性发送1

流水线协议：总结

Go-back-N:

- 发送端最多在流水线中有N个未确认的分组
- 接收端只是发送累计型确认 *cumulative ack*
 - 接收端如果发现gap，不确认新到来的分组
- 发送端拥有对最老的未确认分组的定时器
 - 只需设置一个定时器
 - 当定时器到时时，重传所有未确认分组

Selective Repeat:

- 发送端最多在流水线中有N个未确认的分组
- 接收方对每个到来的分组单独确认 *individual ack* (非累计确认)
- 发送方为每个未确认的分组保持一个定时器
 - 当超时定时器到时，只是重发到时的未确认分组

3.5 connection-oriented transport: TCP

既然我们已经学习了可靠数据传输的基本原理，我们就可以转而学习 TCP 了。TCP 是因特网运输层的面向连接的可靠的运输协议。我们在本节中将看到，为了提供可靠数据传输，TCP 依赖于前一节所讨论的许多基本原理、其中包括差错检测、重传、累积确认、定时器以及用于序号和确认号的首部字段。TCP 定义在 RFC 793、RFC 1122、RFC 1323、RFC 2018 以及 RFC 2581 中。

TCP: overview

RFCs: 793, 1122, 2018, 5681, 7323

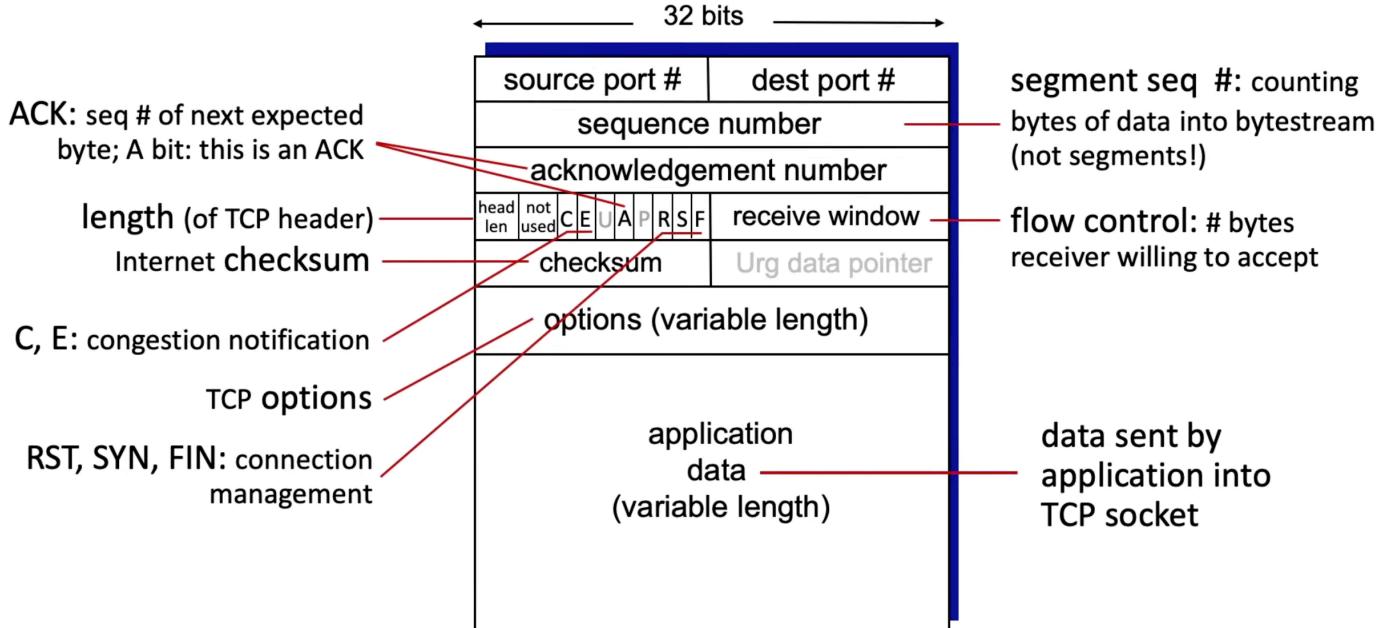
- **point-to-point:**
 - one sender, one receiver
- **reliable, in-order byte steam:**
 - no “message boundaries”
- **full duplex data:**
 - bi-directional data flow in same connection
 - MSS: maximum segment size
- **cumulative ACKs**
- **pipelining:**
 - TCP congestion and flow control set window size
- **connection-oriented:**
 - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- **flow controlled:**
 - sender will not overwhelm receiver

TCP segment structure

与 UDP 一样，TCP 首部包括源端口号和目的端口号，它被用于多路复用/分解来自或送到上层应用的数据。另外，同 UDP 一样，TCP 首部也包括检验和字段 (checksum field)。TCP 报文段首部还包含下列字段：

- **32 bits 的序号字段 (sequence number field):** TCP 连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。
- **32 比特的确认号字段 (acknowledgment number field):** 是期望收到对方的下一个报文段的数据的第一个字节的序号。
- **16 bits 的接收窗口字段 (receive window field):** 该字段用于流量控制。作为接收方让发送方设置发送窗口的依据，单位为字节。窗口值经常在动态变化着，此字段明确指出现在允许对方发送的数据量。注意：不是滑动窗口的大小，滑动窗口的大小是固定的，他是用来记录接受缓冲区的大小，如果大小为0，就不会发送数据了。以此达到流量控制。
- **4 bits 的首部长度字段 (header length field):** 该字段指示了以 32 比特的字为单位的 TCP 首部长度。由于 TCP 选项字段的原因，TCP 首部的长度是可变的。(通常 选项字段为空，所以 TCP 首部的典型长度是 20 字节)
- **可选与变长的选项字段 (options field):** 该字段用于发送方与接收方 协商最大报文段长度 (MSS) 时，或在高速网络环境下用作窗口调节因子时使用。首部字段中还定义了一个时间戳选项。可参见 RFC 854 和 RFC 1323 了解其他细节。
 - **MSS (Maximum Segment Size)** 是 TCP 报文段中的数据字段的最大长度。数据字段加上 TCP 首部才等于整个的 TCP 报文段。所以，MSS 是“TCP 报文段长度减去 TCP 首部长度”。
- **6 bits 的标志字段 (flag field):**
 - **紧急 URG:** 当 URG=1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。
 - **确认 ACK:** 只有当 ACK =1 时确认号字段才有效。当 ACK =0 时，确认号无效。
 - **推送 PSH (PuSH):** 接收 TCP 收到 PSH =1 的报文段，就尽快地交付接收应用进程，而不再等到整个缓存都填满了后再向上交付。

- **复位 RST (ReSeT):** 当 RST=1时，表明TCP连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。
- **同步SYN:** 同步SYN=1表示这是一个连接请求或连接接受报文。与ACK配合实现。
- **终止FIN (FINish):** 用来释放一个连接。FIN=1表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。
- 所以，响应ACK = 请求SEQ + 请求字节数



TCP sequence number, ACK

- **TCP sequence number:** segment 的第一个字节在 data stream 的编号，代表当前报文的发送序号。
- **ACK:** 期望从另一方收到的下一个字节的序号，累积确认。代表接收方需要的数据将从对方哪个序号开始。

sequence number:

- byte “number” of first byte in segment’s data stream

acknowledgement:

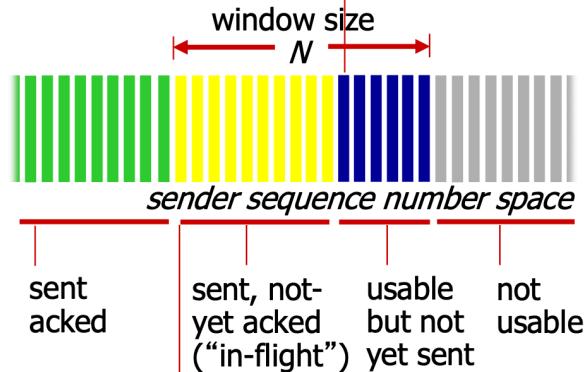
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- **A:** TCP spec doesn’t say, - up to implementer

outgoing segment from sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer



incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer

TCP序号和确认号之间的关系

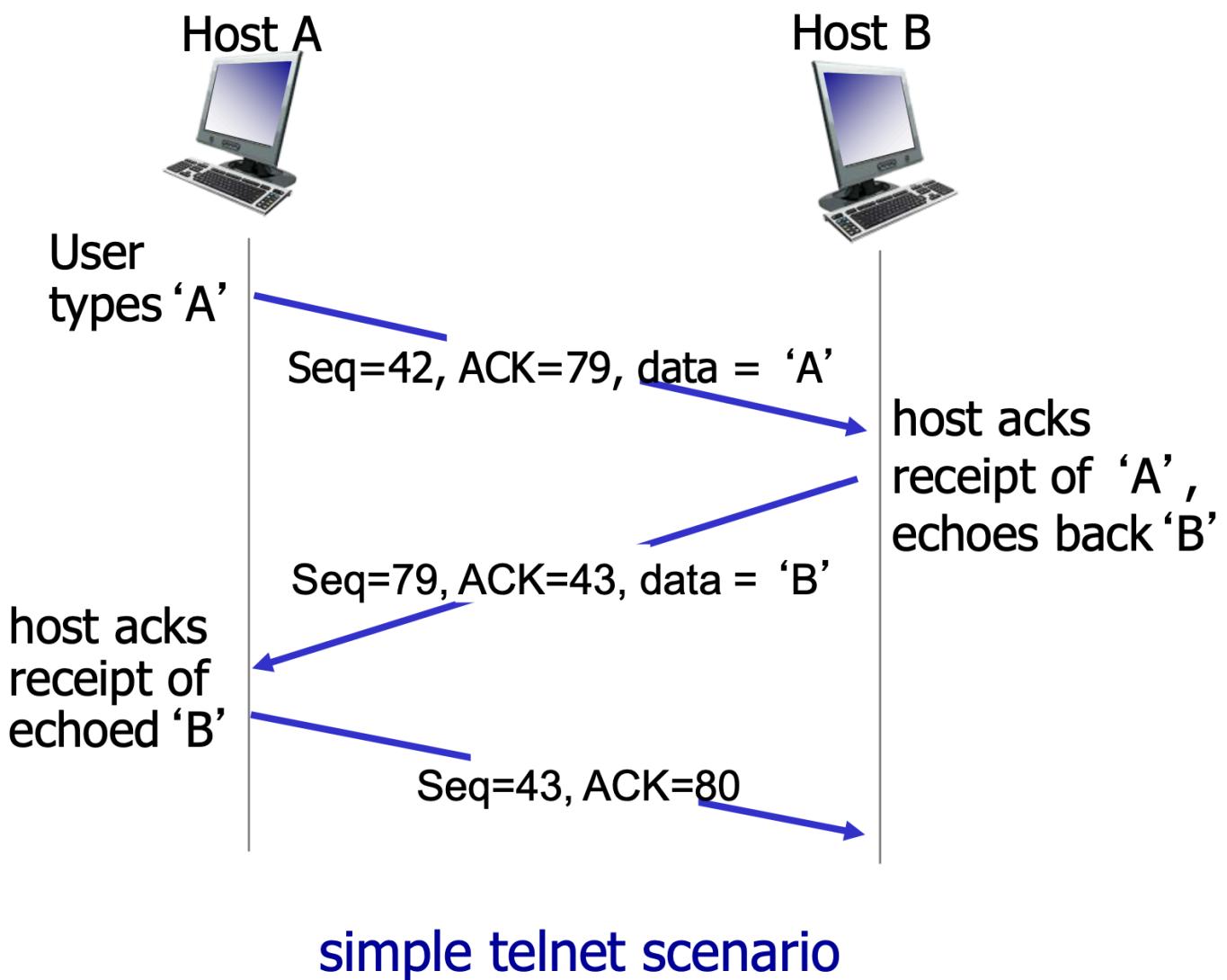
之前讲的都是单向传递，在这里 Telnet 是双向数据传递，互相发送数据和确认，两个互为接受方和发送方。

作为接收方a对上一次b发送的数据的确认。Seq是自己传给对方的字节流的数据开始点 (序号)，Ack是希望对方传给自己的字节流开始点 (确认号)。客户端发出的每一个字符到了服务器，服务器要回转回来，然后客户端要给出相应的确认。

假设客户和服务器的起始序号分别是 42 和 79。前面讲过，一个报文段的序号就是该报文段数据字段首字节的序号。因此，客户发送的第一个报文段的序号为 42, 服务器发送的第一个报文段的序号为 79。前面讲过，确认号就是主机正在等待的数据的下一个字节序号。在 TCP 连接建立后但没有发送任何数据之前，该客户等待字节 79, 而该服务器等待字节 42。

- Host A:
 - 在它的数据字段里包含一字节的字符 'A' 的 ASCII 码。
 - 第一个报文段的序号字段里是 42。
 - 另外，由于 A 还没有接收到来自 B 的任何数据，因此该第一个报文段中的确认号字段中是 79。即，当前的报文段数据是从第42个字节开始，希望从B那里获取从第79个字节开始的报文段
- Host B:
 - 首先它是为所收到数据提供一个确认，通过在确认号字段中填入 43, 告诉 A 它已经成功地收到字节 42 及以前的所有字节，现在正等待着字节 43 的出现。
 - 该报文段的第二个目的是回显字符'B'。因此，在第二个报文段的数据字段里填入的是字符 'B' 的 ASCII 码。
 - 第二个报文段的序号为 79, 它是该 TCP 连接上从服务器到客户的数据流的起始序号，这也正是 B 要发送的第一个字节的数据。即，收到42。返回ACK43(接收到了42号及以前的报文，希望从A那里获取从第43个字节开始的报文段)，并且发送第79个报文

- Host A: 该报文段的数据字段为空。收到79之后, 返回ACK80, 说我接受到了字节流中序号为 79 及以前的字节, 希望你发送下一个从80开始的, 并且发送第43个报文



TCP round trip time, timeout

- **Problem:** TCP 和 rdt 协议一样采用超时/重传机制来处理报文段的丢失问题。但是问题是定时器的超时时间设置。
- 设置 **timeout** 的间隔必须 > RTT

Q: 怎样设置TCP超时？

- 比RTT要长
 - 但RTT是变化的
- 太短：太早超时
 - 不必要的重传
- 太长：对报文段丢失反应太慢，消极

Q: 怎样估计RTT？

- SampleRTT**: 测量从报文段发出到收到确认的时间
 - 如果有重传，忽略此次测量
- SampleRTT**会变化，因此估计的RTT应该比较平滑
 - 对几个最近的测量值求平均，而不是仅用当前的**SampleRTT**

EstimatedRTT

- **SampleRTT**: 比如发了50个报文，我抽取的5个报文作为SampleRTT进行统计。
- **EstimatedRTT**: SampleRTT的均值，一旦获得一个新 SampleRTT 时，TCP 就会根据下列公式来更新 EstimatedRTT

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- weighted moving average
- typical value: $\alpha = 0.125$

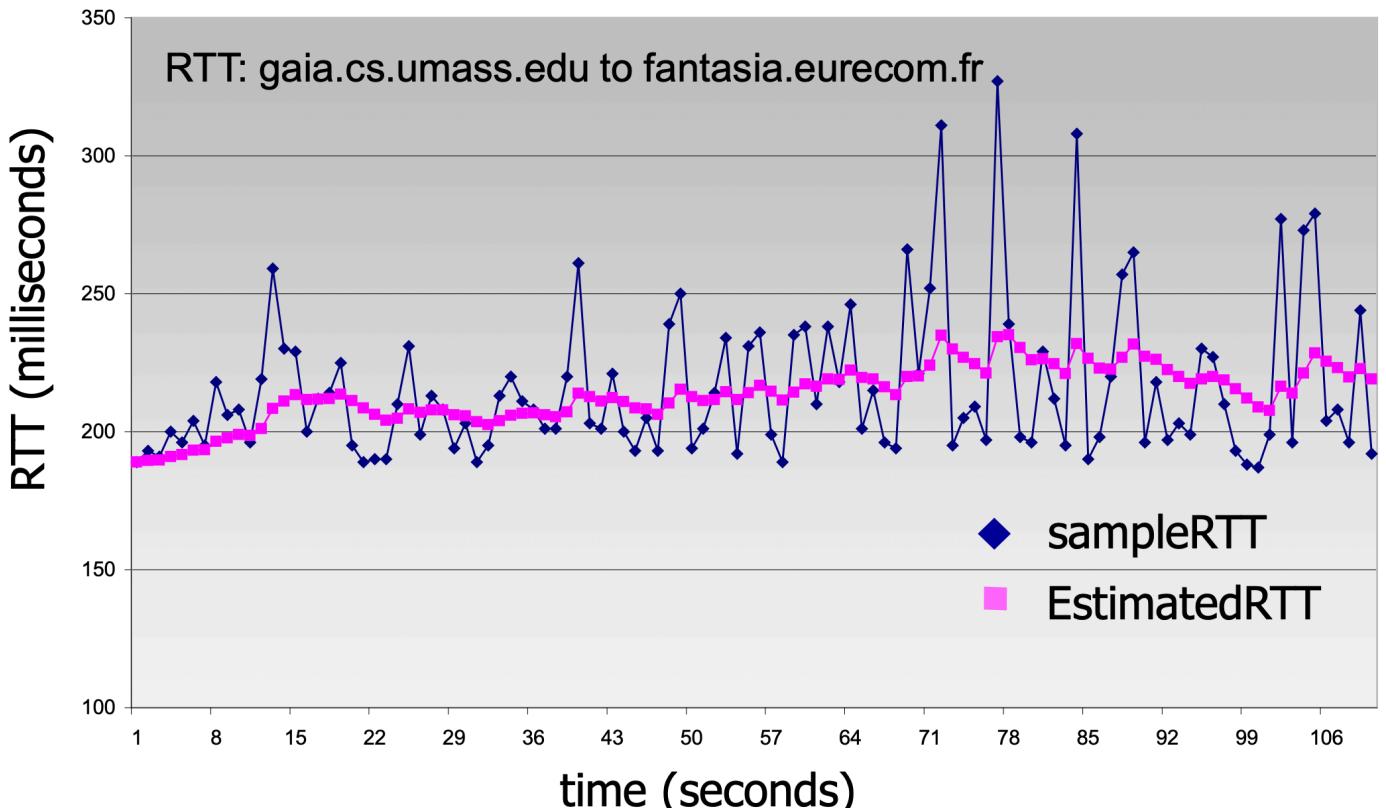
- 上面的公式是以编程语言的语句方式给出的，即 EstimatedRTT 的新值是由以前的 EstimatedRTT 值与 SampleRTT 新值 **加权** 组合而成的。在 [RFC 6298] 中给出的 α 参考值是 $\alpha=0.125$ (即 $1/8$)，这时上面的公式变为：

$$\text{EstimatedRTT} = 0.875 * \text{EstimatedRTT} + 0.125 * \text{SampleRTT}$$

EstimatedRTT 是一个 SampleRTT 值的加权平均值。这个加权平均 对最近的样本赋予的权值要大于对旧样本赋予的权值。这是很自然的，因为越近的样本越能更好地反映网络的当前拥塞情况

从统计学观点讲，这种平均被称为 **指数加权移动平均 (Exponential Weighted Moving Average, EWMA)**。在 EWMA 中的“指数”一词看起来是指一个给定的 SampleRTT 的权值在更新的过程中呈指数型快速衰减

下图显示了当 $\alpha=1/8$ 时，在 gaia.cs.umass.edu (在美国马萨诸塞州的 Amherst) 与 fantasia.eurecom.fr (在法国南部) 之间的一条 TCP 连接上的 SampleRTT 值与 EstimatedRTT 的值。显然 SampleRTT 的变化在 EstimatedRTT 的计算中趋于平缓了。



DevRTT

使用DevRTT的原因是我们想要一个更好的 safety margin

- 除了估算RTT外，测量RTT的变化也是有价值的。
- [RFC6298] 定义了 **RTT偏差DevRTT**，用于估算 SampleRTT一般会偏离 EstimatedRTT 的程度：
- 就是先计算一个往返时间的估计值，在此基础上加上一个偏差范围的四倍，相当于计算了短时间内能覆盖99%以上情况的超时时间

- timeout interval: EstimatedRTT plus “safety margin”**
 - large variation in EstimatedRTT -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:**

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



↑
estimated RTT

↑
“safety margin”

TCP reliable data transfer

TCP是一种流水线的传输方式，那么TCP属于GBN 还是属于SR呢，其实TCP是GBN和SR的混合体。

TCP: 可靠数据传输

- TCP在IP不可靠服务的基础上建立了rdt
 - 管道化的报文段
 - GBN or SR
 - 累积确认（像GBN）
 - 单个重传定时器（像GBN）
 - 是否可以接受乱序的，没有规范
- 首先考虑简化的TCP发送方：
 - 忽略重复的确认
 - 忽略流量控制和拥塞控制
- 通过以下事件触发重传
 - 超时（只重发那个最早的未确认段：SR）
 - 重复的确认
 - 例子：收到了ACK50,之后又收到3个ACK50

TCP发送方事件:

从应用层接收数据:

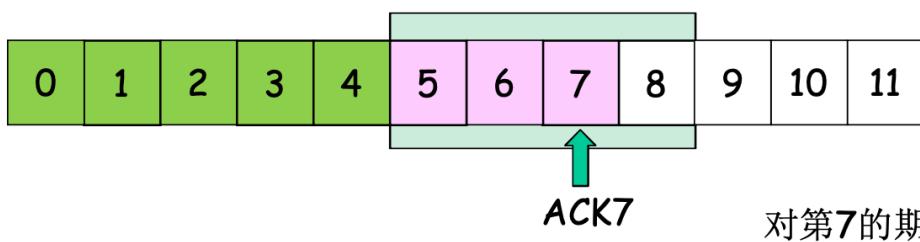
- 用nextseq创建报文段
- 序号nextseq为报文段首字节的字节流编号
- 如果还没有运行，启动定时器
 - 定时器与最早未确认的报文段关联
 - 过期间隔：
TimeOutInterval

超时:

- 重传后沿最老的报文段
- 重新启动定时器

收到确认:

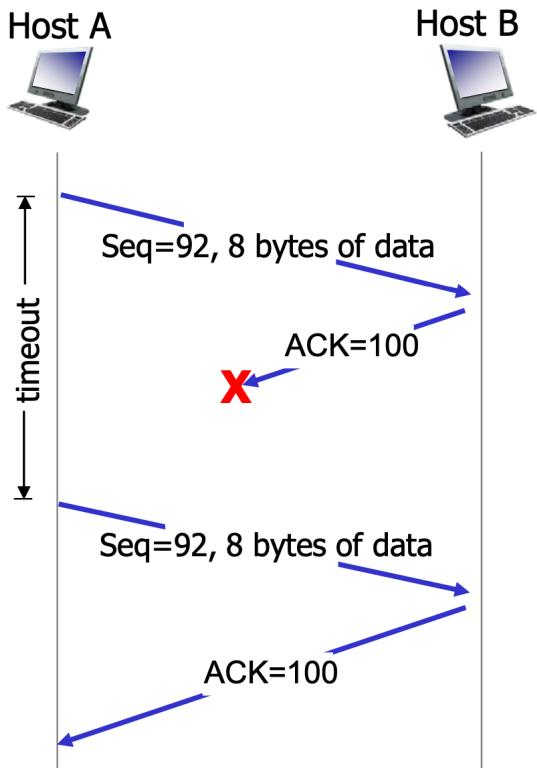
- 如果是对尚未确认的报文段确认
 - 更新已被确认的报文序号
 - 如果当前还有未被确认的报文段，重新启动定时器



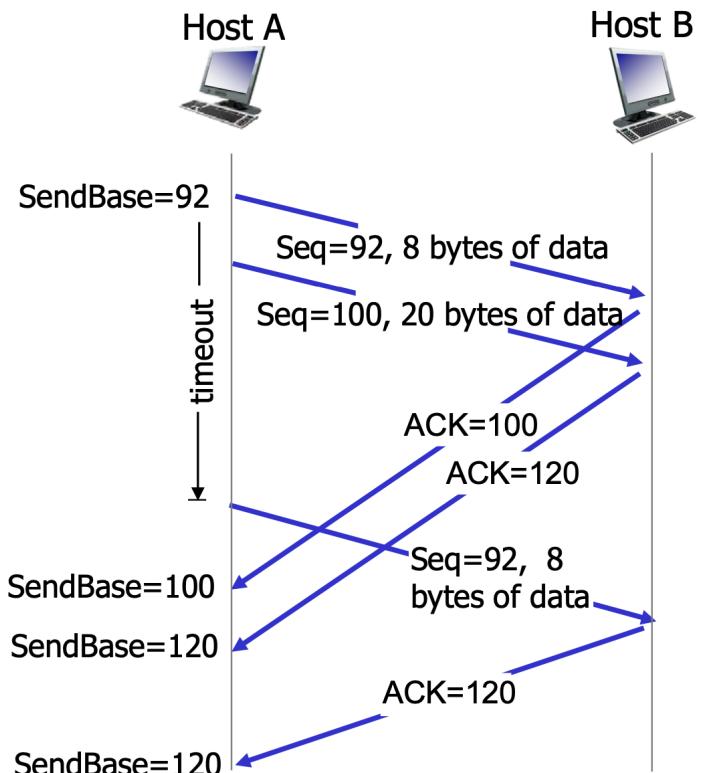
对第7的期待

- 顺序接收：因为有了分组编号，接收方收到乱序的分组后，不会选择丢弃，而是缓存起来，排好序。返回的ACK仍然是第一个未收到的分组序号。
- 超时重传：tcp发送方只为最早的分组设置一个定时器，当分组未收到ACK触发超时重传，并获得ACK应答后，才开始为下一个未确认分组开始计时。
- 选择确认：发送方未接收到分组的ACK时，只会重新发送单个分组，而不会发送所有分组。

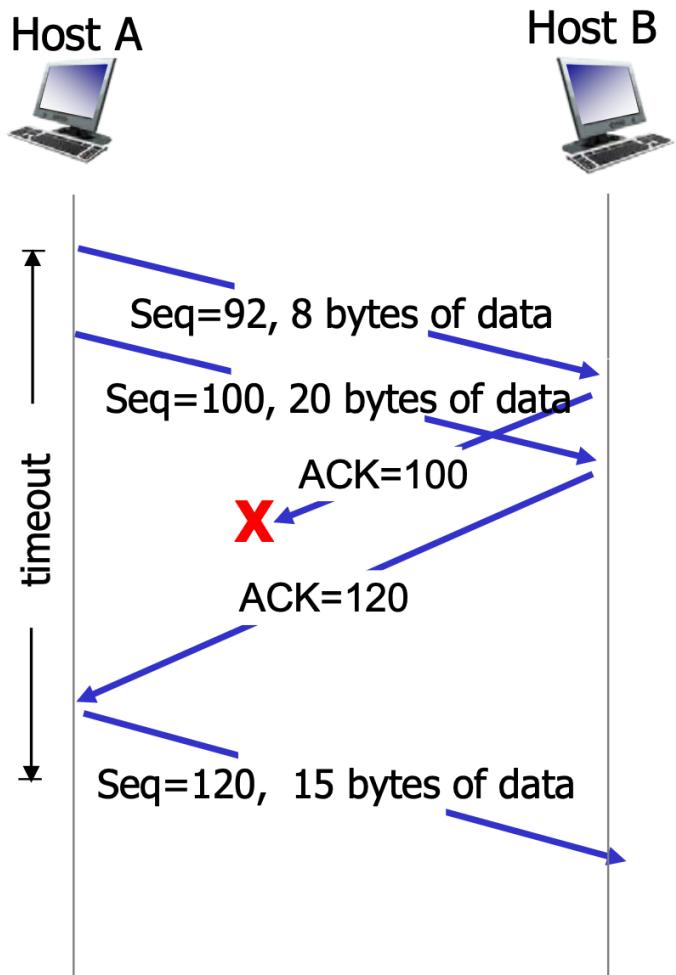
TCP: retransmission scenarios



lost ACK scenario



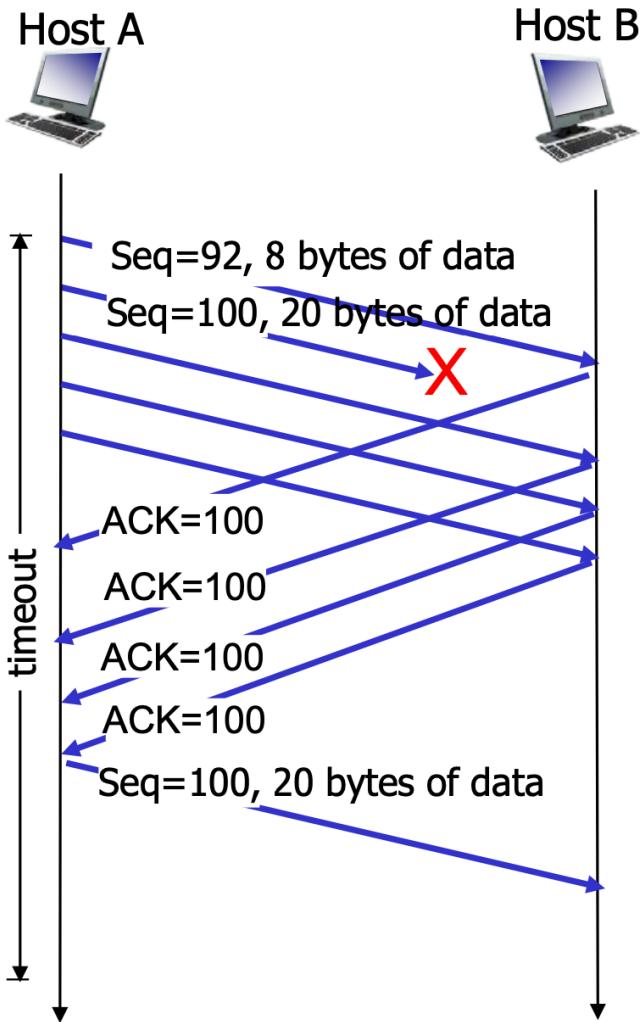
premature timeout



cumulative ACK

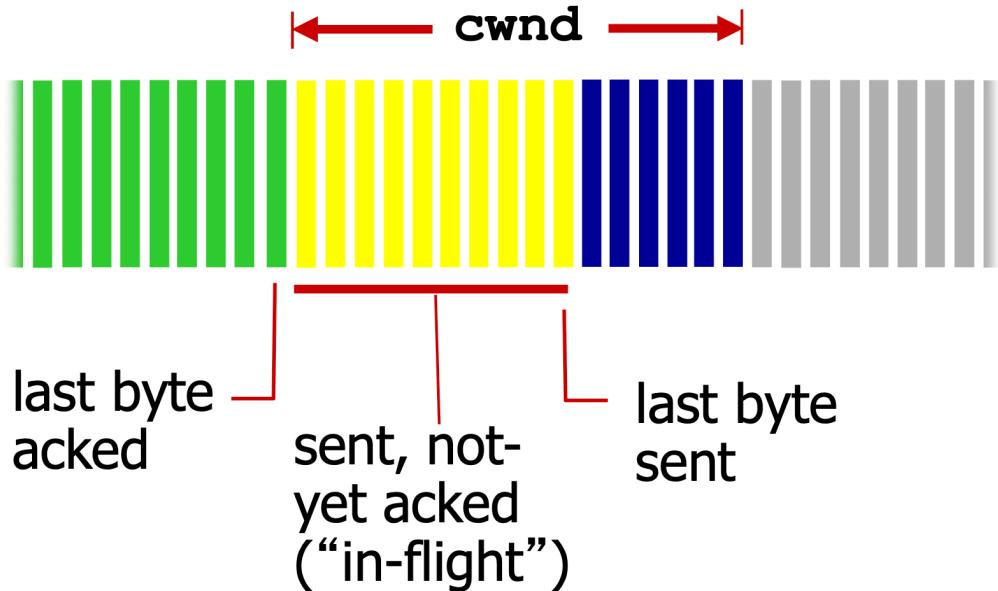
TCP fast retransmit

快速重传 (TCP fast retransmit): 基于接收方ACK确认机制，ACK序号为按顺序第一个未收到的分组。发送方接收到三次相同的ACK序号后，会触发快速重传机制，重传ACK序号标识的分组。



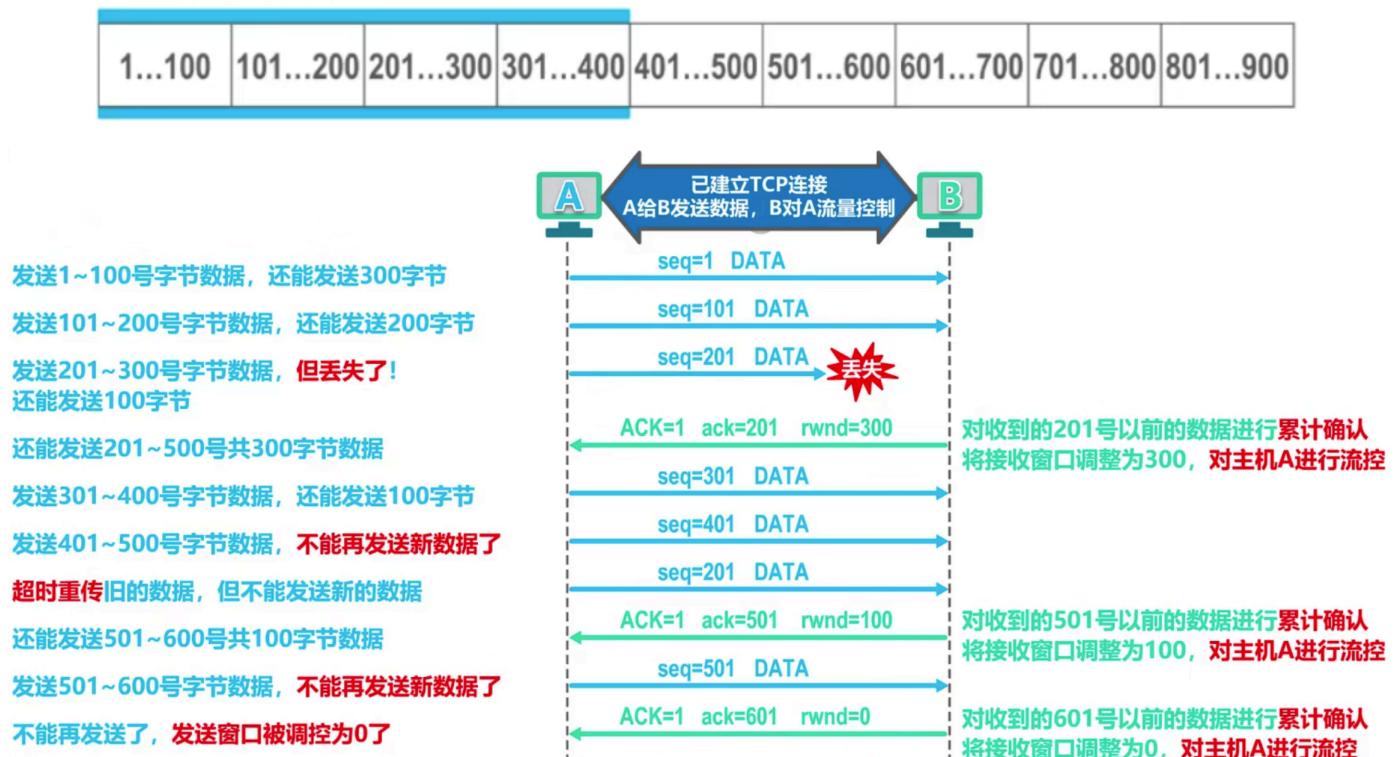
TCP flow control

- 一般来说，我们总是希望数据传输得更快一些
- 但如果发送方把数据发送得过快，接收方就可能来不及接收，这就会造成数据的丢失
- 所谓流量控制(**flow control**) 就是让发送方的发送速率不要太快，要让接收方来得及接收
- 利用滑动窗口机制 (**rwnd**) 可以很方便地在TCP连接上实现对发送方的流量控制。
 - TCP接收方利用自己的接收窗口的大小来限制发送方发送窗口的大小。
 - TCP发送方收到接收方的 $rwnd = 0$ 通知后，应启动持续计时器。持续计时器超时后，向接收方发送 $rwnd$ 探测报文。
- TCP发送方的发送窗口 = $\min \{ \text{自身拥塞窗口}, \text{TCP接收方的接收窗口} \}$

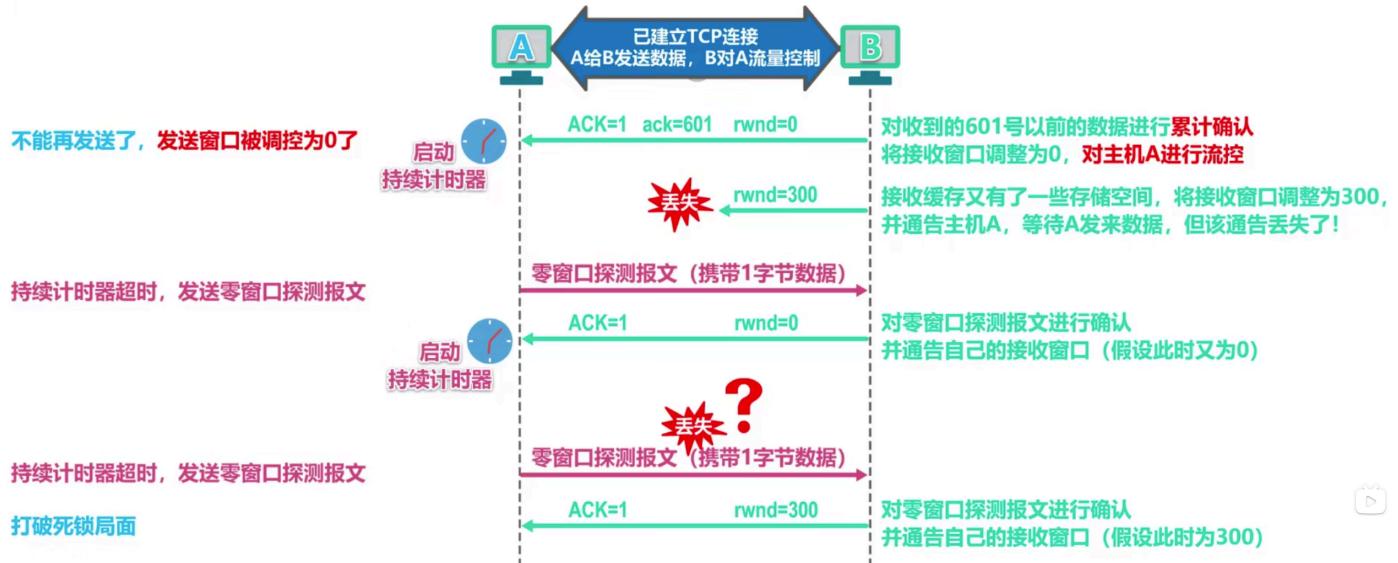


Example:

initial 发送窗口 = 400, -> initial rwnd = 400



TCP发送方收到接收方的 $rwnd = 0$ 通知后，应启动持续计时器。持续计时器超时后，向接收方发送 $rwnd$ 探测报文。



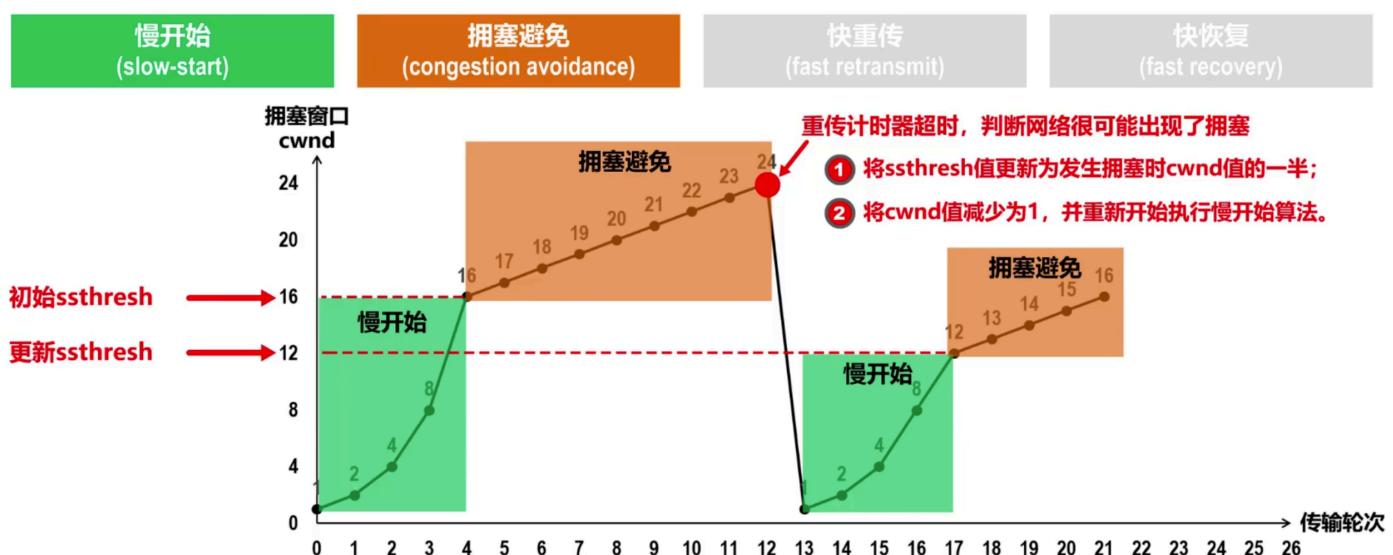
3.6 TCP congestion control

有时, 发送方和接收方性能后很好, 结果是中途的网络带宽不行, 网络中堵塞了。这时候如果还是依然大量的发送消息, 反而会造成更大面积的网络拥塞。

- manifestations (表现为):
 - lost packets (buffer overflow at routers) 丢包 (路由器缓冲区溢出)
 - long delays (queueing in router buffers) 长时间延迟 (在路由器缓冲区排队)

TCP Tahoe

- slow-start
- congestion avoidance



■ “慢开始”是指一开始向网络注入的报文段少, 并不是指拥塞窗口cwnd增长速度慢;

■ “拥塞避免”并非指完全能够避免拥塞, 而是指在拥塞避免阶段将拥塞窗口控制为按线性规律增长, 使网络比较不容易出现拥塞;

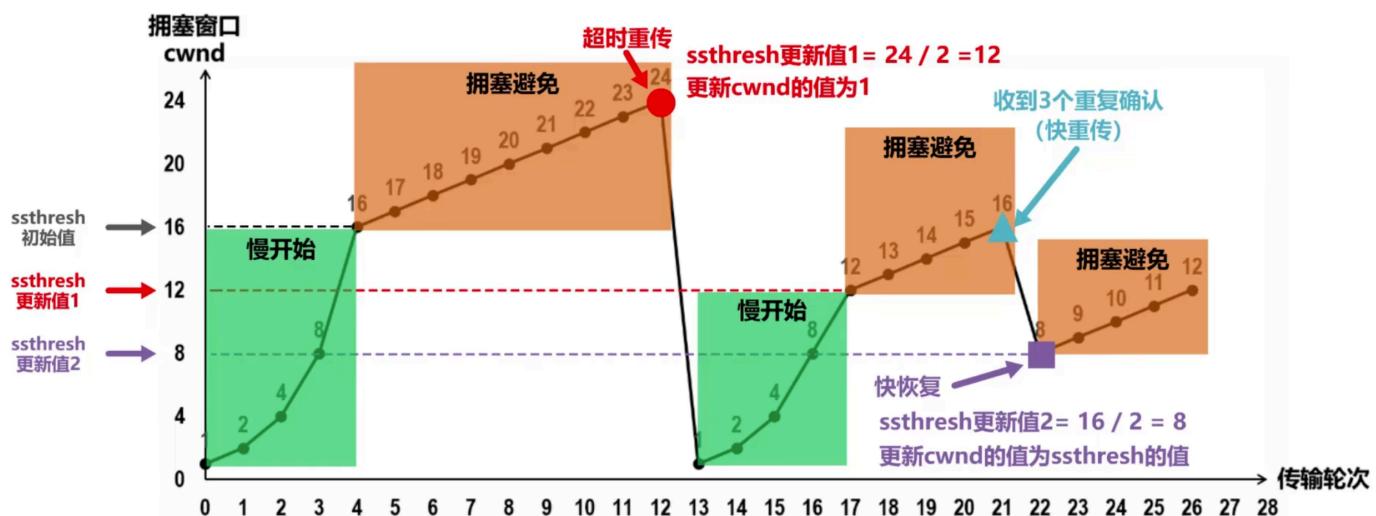
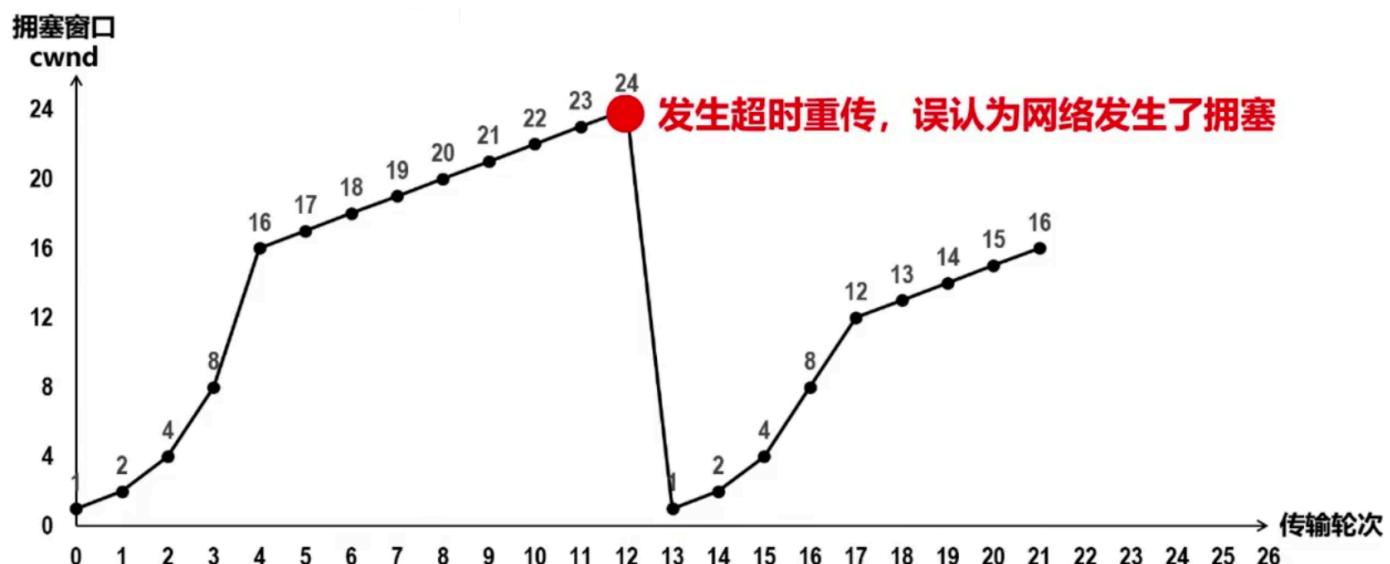
TCP Reno

- fast retransmit
- fast recovery

有时，个别报文段会在网络中丢失，但实际上网络并未发生拥塞。这将导致发送方超时重传，并误认为网络发生了拥塞，发送方把拥塞窗口 cwnd 又设置为最小值 1，并错误地启动慢开始算法，因而降低了传输效率。

- 采用快重传算法可以让发送方尽早知道发生了个别报文段的丢失。
- 所谓快重传，就是使发送方尽快进行重传，而不是等超时重传计时器超时再重传。
 - 要求接收方不要等待自己发送数据时才进行捎带确认，而是要立即发送确认；
 - 即使收到了失序的报文段也要立即发出对已收到的报文段的重复确认。
 - 发送方一旦收到3个连续的重复确认，就将相应的报文段立即重传，而不是等该报文段的超时重传计时器超时再重传。

之前的例子中，当拥塞窗口值 $cwnd = 24$ 时发生了超时重传，而此时网络并没有发生拥塞，但是发送方却误认为网络发生了拥塞，于是发送方把拥塞窗口 $cwnd$ 减少到 1，并错误的启动慢开始算法，降低了传输效率。



04. Network (Data Plane) 端到端

Network layer 将段(segment)从一台发送主机移动到一台接收主机，是主机之间的逻辑通信，能够被分解为两个相互作用的部分，即 **data plane** (数据平面) 和 **control plane** (控制平面)

Two key network-layer functions:

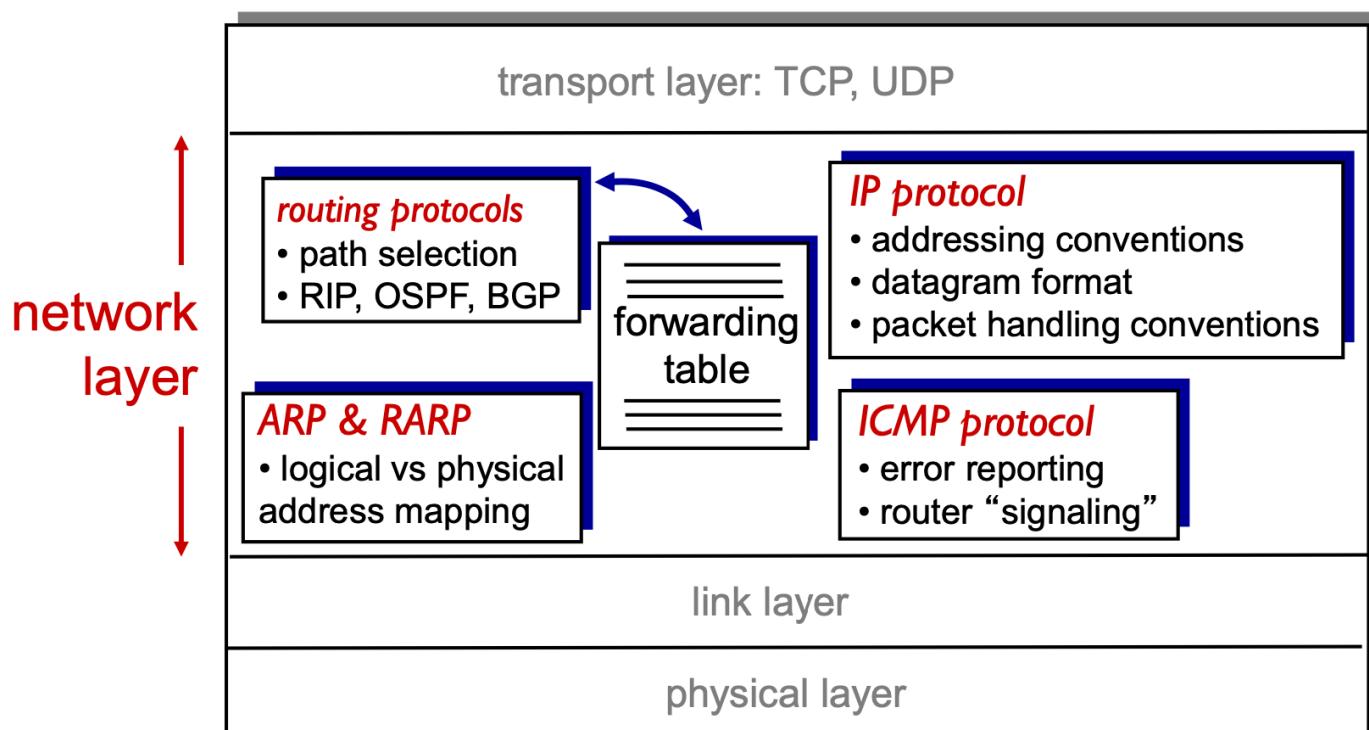
- forwarding: 将分组从路由器的输入接口转发到合适的输出接口，通过单个路口的过程。就是说从不同的端口收来的分组，然后通过合适的端口打出去。是局部功能（转发是在数据平面中实现的唯一功能）
- routing: 使用路由算法来决定分组从发送主机到目标接收主机的路径，从出发地到目的地的行程规划过程。是全局功能。（路由选择在网络层的控制平面中实现）



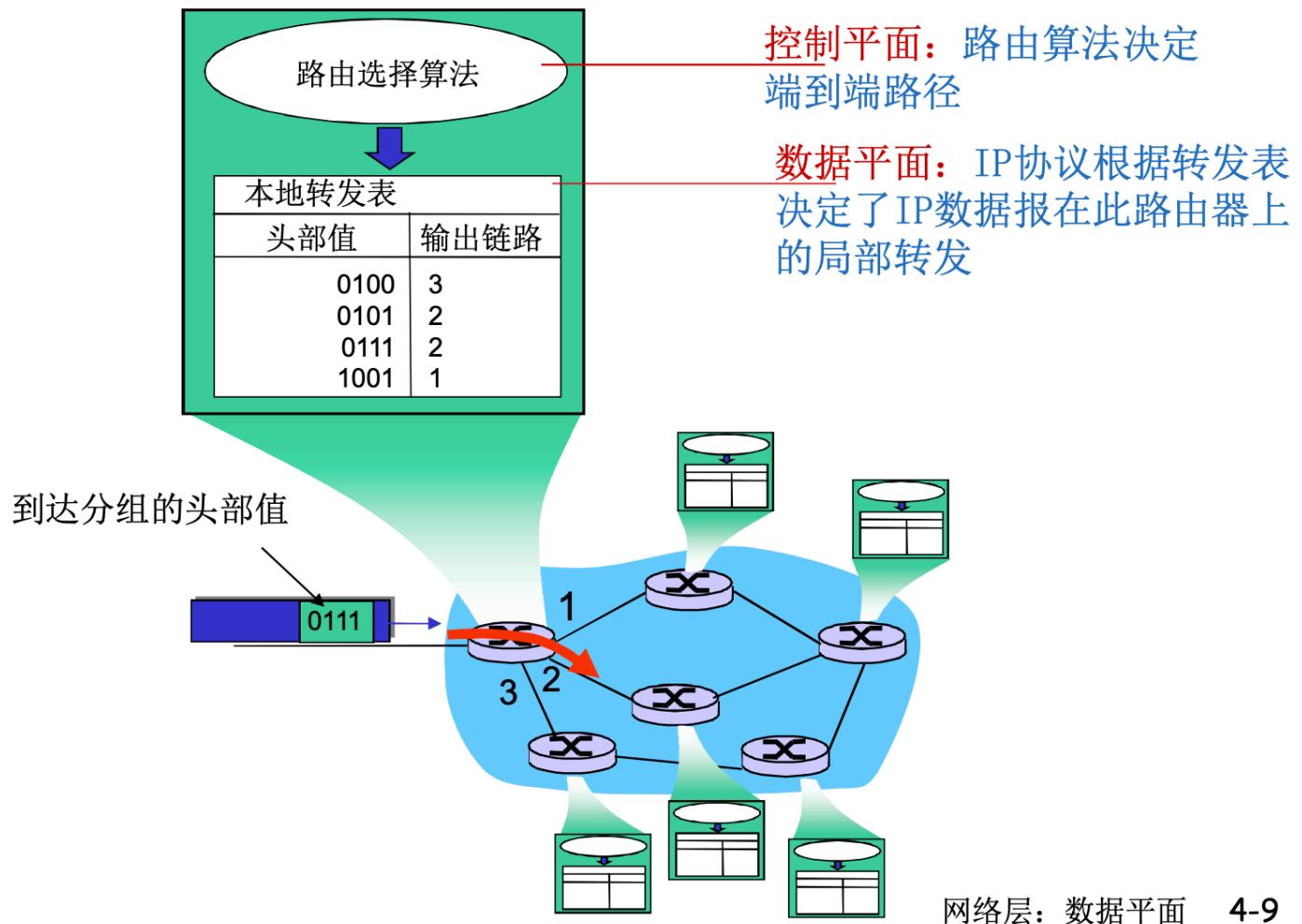
forwarding



routing

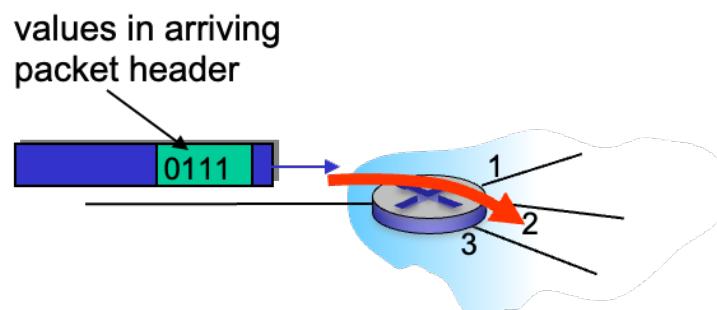


4.1 Overview of network layer



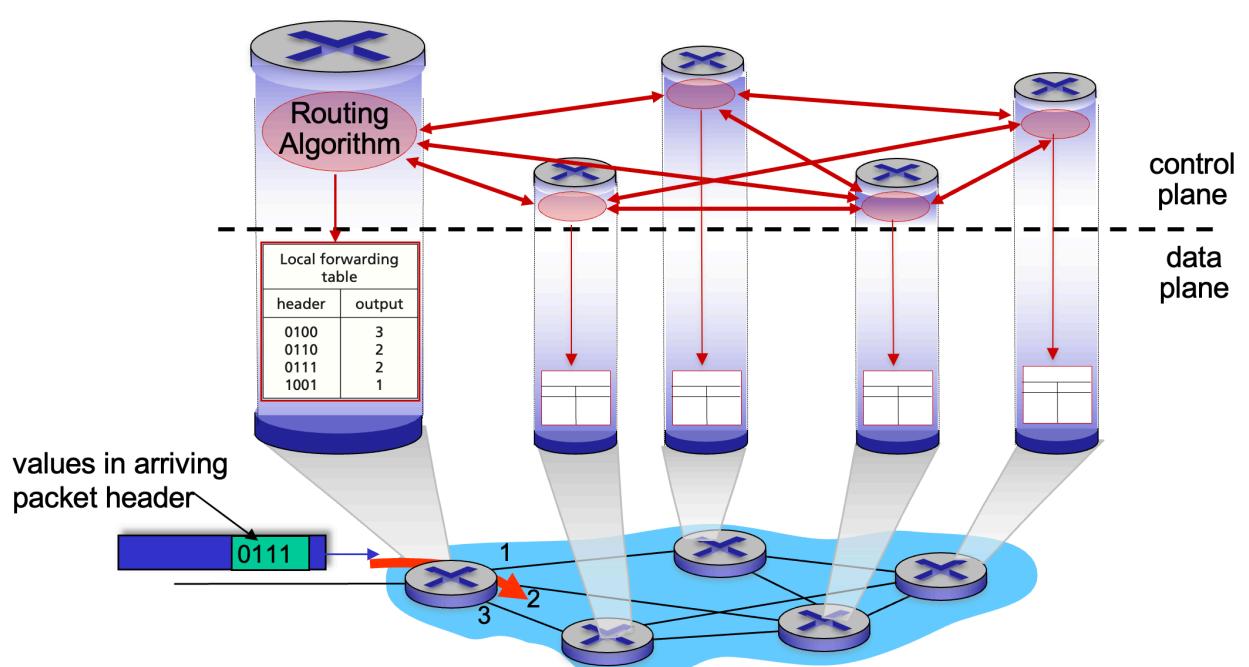
data plane

- 本地, 每个路由器功能
- 确定到达路由器输入端口的 datagram 如何 **forward** 到路由器输出端口
- 转发功能:
 - 传统方式: 基于目标地址 + 转发表
 - SDN方式: 基于多个字段 + 流表

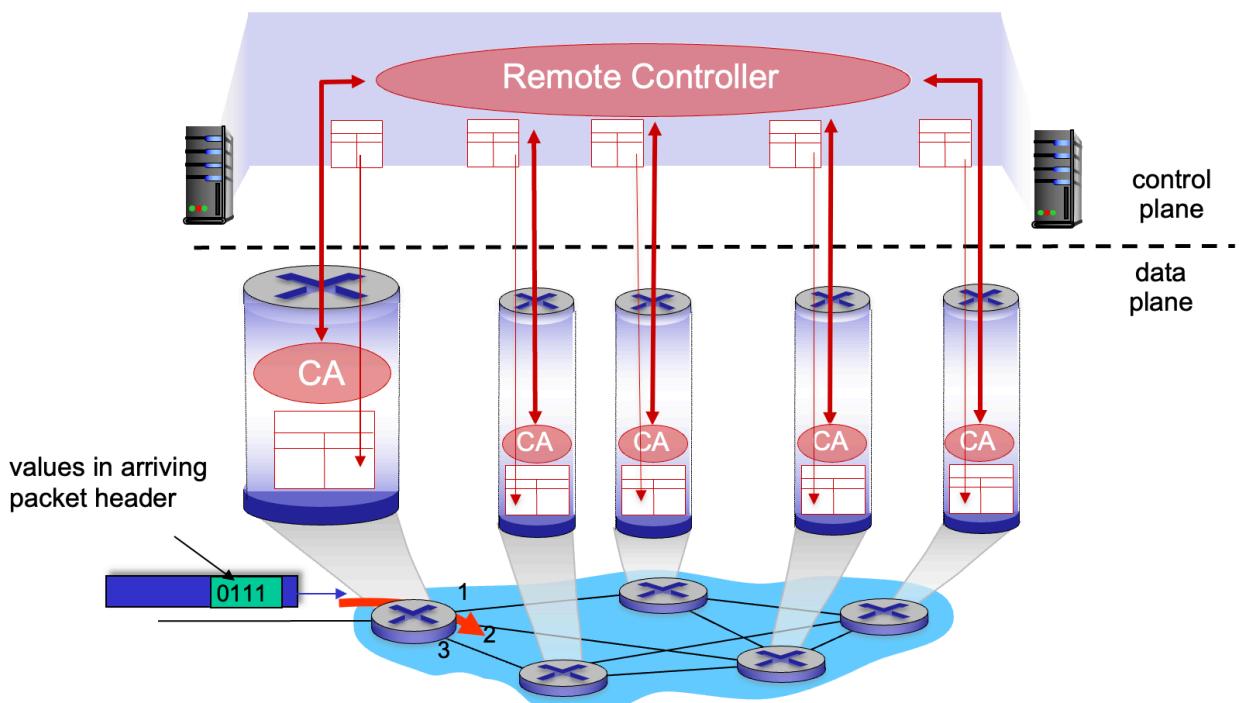


control plane

- network-wide logic (网络范围内的逻辑)
- 确定 datagram 如何沿着从源主机到目标主机的 end-to-end 路径在路由器之间路由
- two control-plane approaches:
 - **traditional routing algorithms(Per-router):** implemented in routers (在路由器中实现)
在每一个路由器中的单独路由器算法元件，在控制平面进行交互



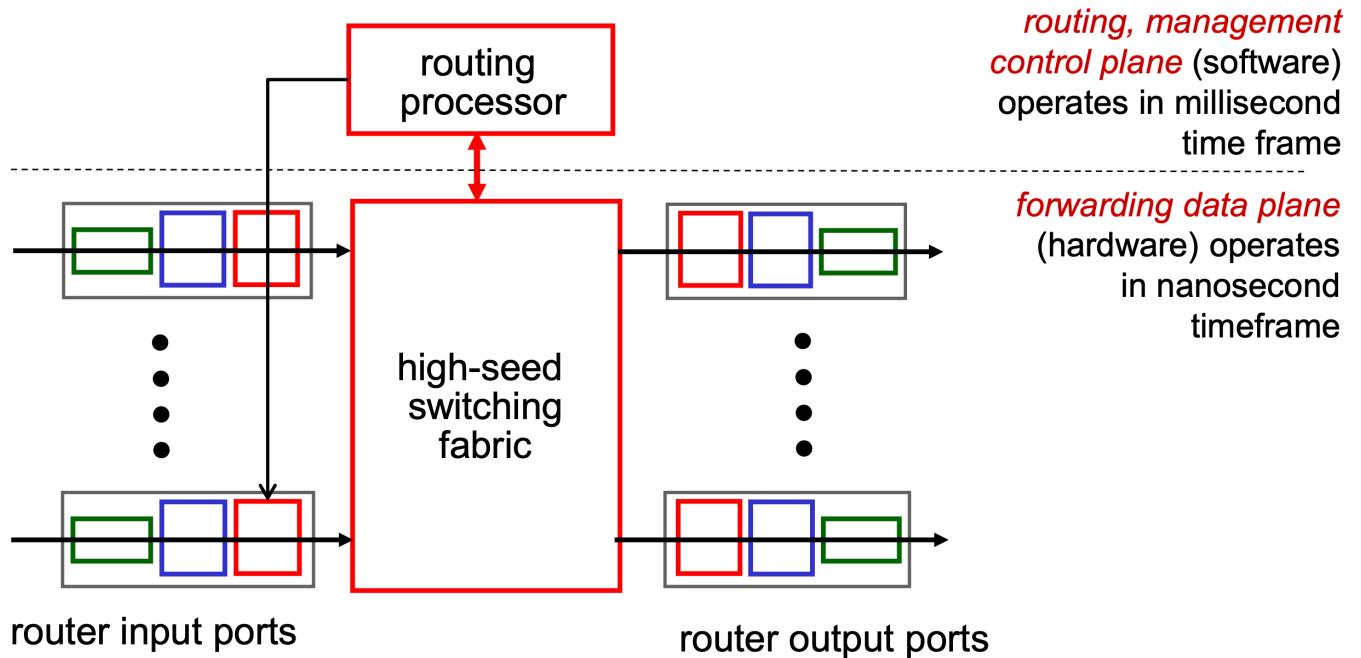
- **software-defined networking (SDN):** implemented in (remote) servers (在远程的服务器中实现)
一个不同的 (通常是远程的) controller interacts with local control agents (CAs)



4.2 What's inside a router

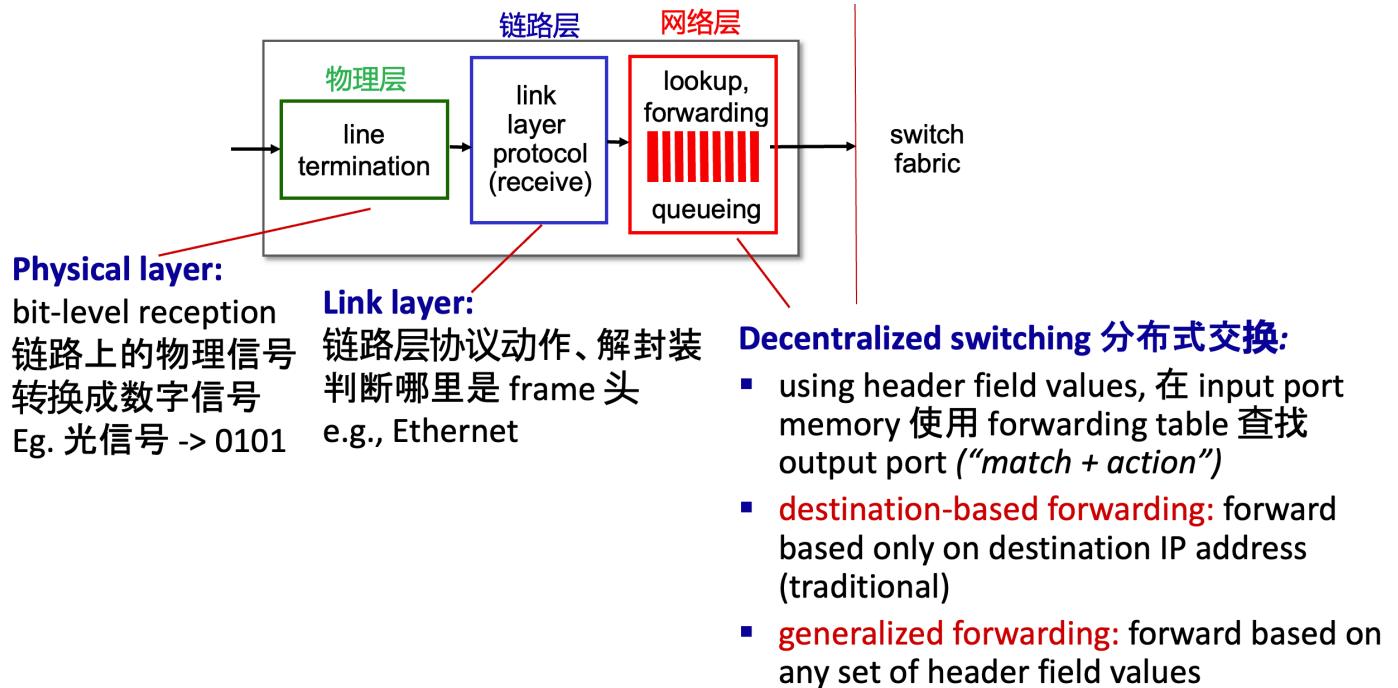
Router architecture overview (路由器体系架构):

- **high-level view of generic router architecture:**



Input port functions:

- link layer 把frame当中的数据部分提取出来数据部分 (可能是一个IP的分组)
- 交给 network layer 实体中的队列 (**queuing**: if datagrams arrive faster than forwarding rate into switch fabric (queueing delay and loss due to input buffer overflow))
- 排到对头后按照路由器交下来的路由表，找到合适的端口，把它放出去 (根据 **header field values**，使用输入端口内存中的转发表查找输出端口("匹配+动作"))，完成输入端口处理
 - **destination-based forwarding (传统转发方式):** 根据 datagram 头部的目标IP地址，来查路由表转发
 - **generalized forwarding (SDN):** 查这个分组的多个层面的头部信息，匹配相应流表的字段，然后找到匹配的流表来action
- 通过fabric做一个局部的交换，从输入端口转到输出端口。

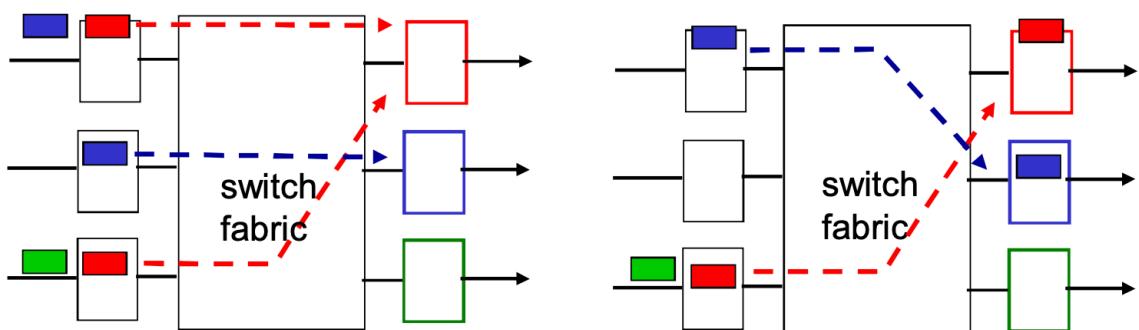


Input port queuing

所以分组有可能会因为其他分组正在输出同一个端口而导致堵塞，如果一直堵着有可能会造成丢失

所以说 network layer 输入端口缓存的目的就是为了匹配瞬间的输入速率和输出速率的不一致性，需要一个queue来匹配不一致性

- 当交换机构的速率小于输入端口的汇聚速率时 → 在输入端口可能要排队
 - 排队延迟以及由于输入缓存溢出造成丢失！
- Head-of-the-Line (HOL) blocking: 排在队头的数据报阻止了队列中其他数据报向前移动



输出端口竞争：
只能有一个红色分组被传递，交
换到一个输出端口。
下面红色的分组被阻塞

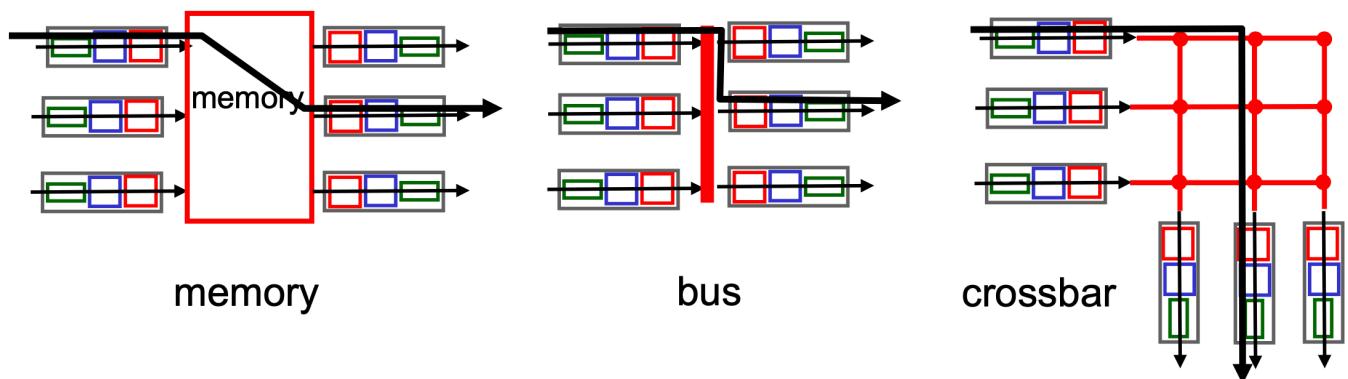
一个分组时间：
绿色分组经历了头
端阻塞

网络层：数据平面

Switching fabrics

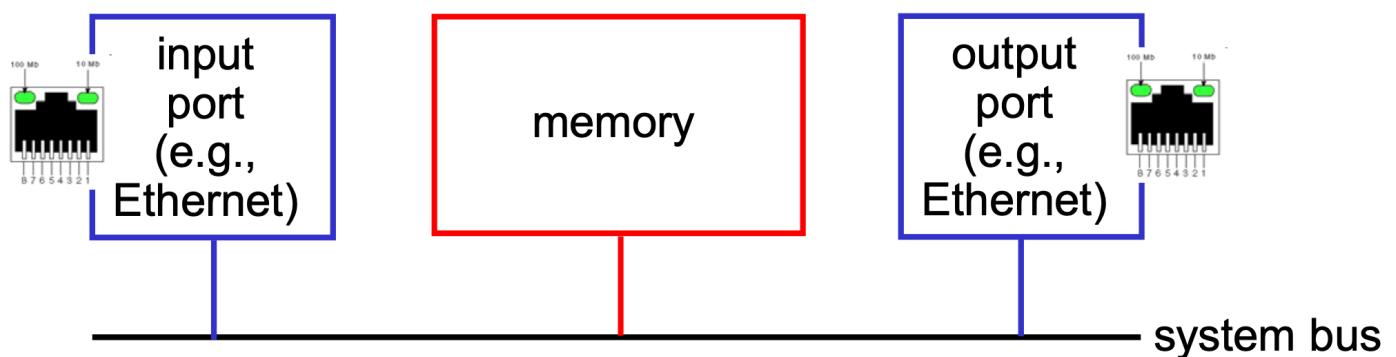
- 然后我们来看一下**fabric**怎么样 将 **packet** 从 **input buffer** 传输到合适的 **output buffer**
- switching rate:** 分组可以按照 switching rate 从输入传输到输出
 - 运行速度经常是输入/输出链路速率的若干倍, 如果有N个输入端口, 交换机构的 switching rate 必须是输入线路速度的N倍, 才不会成为瓶颈
- 有三种常见的fabric的工作模式:

■ three types of switching fabrics



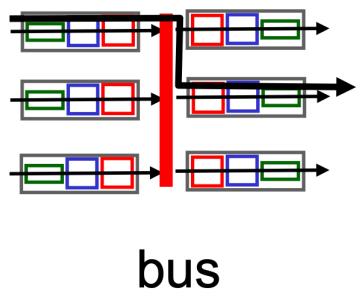
Switching via memory

- 第一代路由器就是在CPU直接控制下的交换, 采用传统的计算机
- 分组被拷贝到系统内存, CPU从分组的头部提取出目标地址, 查找转发表, 找到对应的输出端口, 拷贝到输出端口
- Problems:
 - 转发速率被内存的带宽限制 (**datagram** 通过**BUS**两遍), 所以 系统总线(bus) 本身就会成为fabric交换速度的瓶颈, 带宽每分钟交换或者是转发分组的速率很低
 - 一次只能转发一个分组



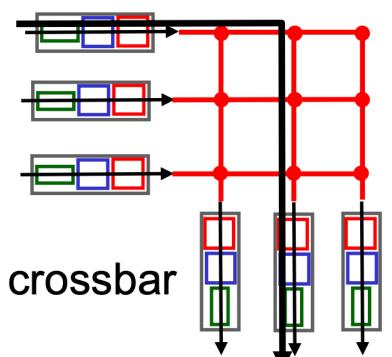
Switching via a bus

- datagram 只要通过一次 share bus (共享总线, 不是系统总线), 从输入端口转发到输出端口
 - Problem:
 - bus contention (总线竞争): 交换速度受限于总线带宽
 - 1次处理一个分组
 - 对于接入或企业级路由器速度足够 (但不适合区域或骨干网络)
- **datagram from input port memory to output port memory via a shared bus**
 - **bus contention:** switching speed limited by bus bandwidth
 - **Cisco 5600: 32 Gbps bus - sufficient speed for access and enterprise routers**



Switching via interconnection network

- 通过互联网络 (Banyan, crossbar等) 的交换, 将多个处理器连接成多处理器, 同时并发转发多个分组, 克服总线带宽限制
 - 当分组从左上角端口到达, 转给下面端口; 控制器短接相应的两个总线
 - advanced design: 将数据报分片为固定长度的信元, 通过交换网络交换
- **banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor**
 - **advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.**
 - **Cisco 12000: switches 60 Gbps through the interconnection network**



destination-based forwarding (传统转发方式):

forwarding table	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Longest prefix matching 最长前缀匹配

longest prefix matching

当给定目标地址查找转发表时，采用**最长地址前缀匹配**的目标地址表项

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

examples:

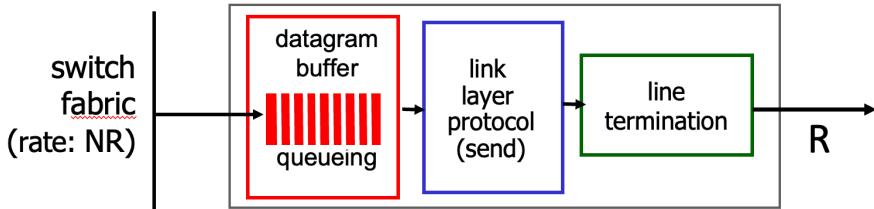
DA: 11001000 00010111 00010110 10100001

which interface?

DA: 11001000 00010111 00011000 10101010

which interface?

Output ports & queueing



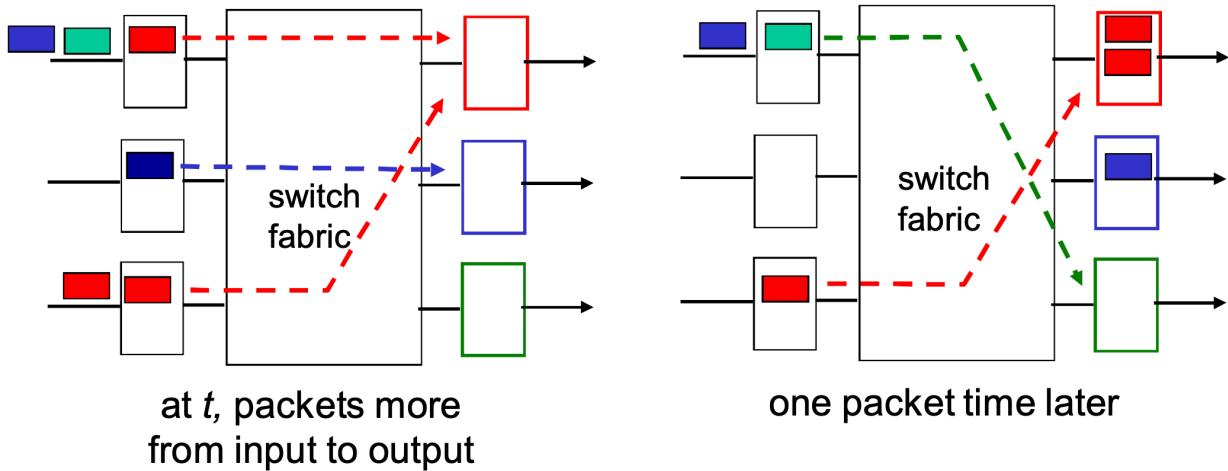
- **Buffering** required when datagrams arrive from fabric faster than link transmission rate.

Datagrams can be lost due to **congestion, lack of buffers**

- **Drop policy:** datagrams 可能会被丢弃, 由于堵塞, 缓冲区没有空间, 哪些应该被丢弃呢

Priority scheduling – who gets best performance, network neutrality

- **Scheduling discipline** 调度规则 chooses among queued datagrams for transmission 选择排队的数据报进行传输



- 假设交换速率 R_{switch} 是 R_{line} 的 N 倍 (N : 输入端口的数量)
- 当多个输入端口同时向输出端口发送时, 缓冲该分组 (当通过交换网络到达的速率超过输出速率则缓存)
- **排队带来延迟, 由于输出端口缓存溢出则丢弃数据报!**

4.3 IP: Internet Protocol

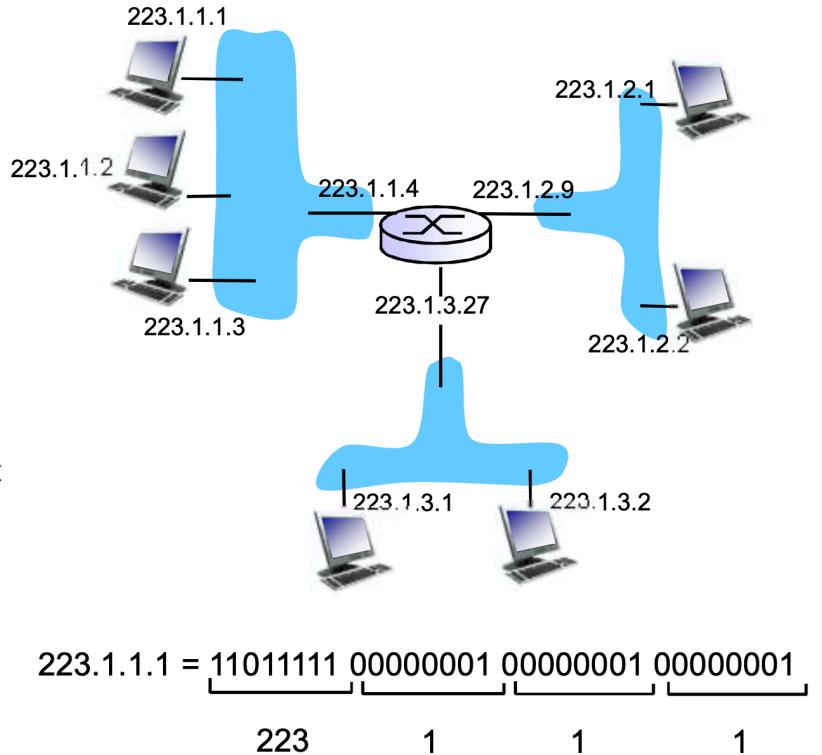
IPv4 addressing

IP = 网络号 + 主机号, IP 地址具有以下结构:

- **subnet part** (网络号, 子网部分): devices in same subnet have common **high order bits**
- **host part** (主机号, 主机部分): remaining **low order bits**

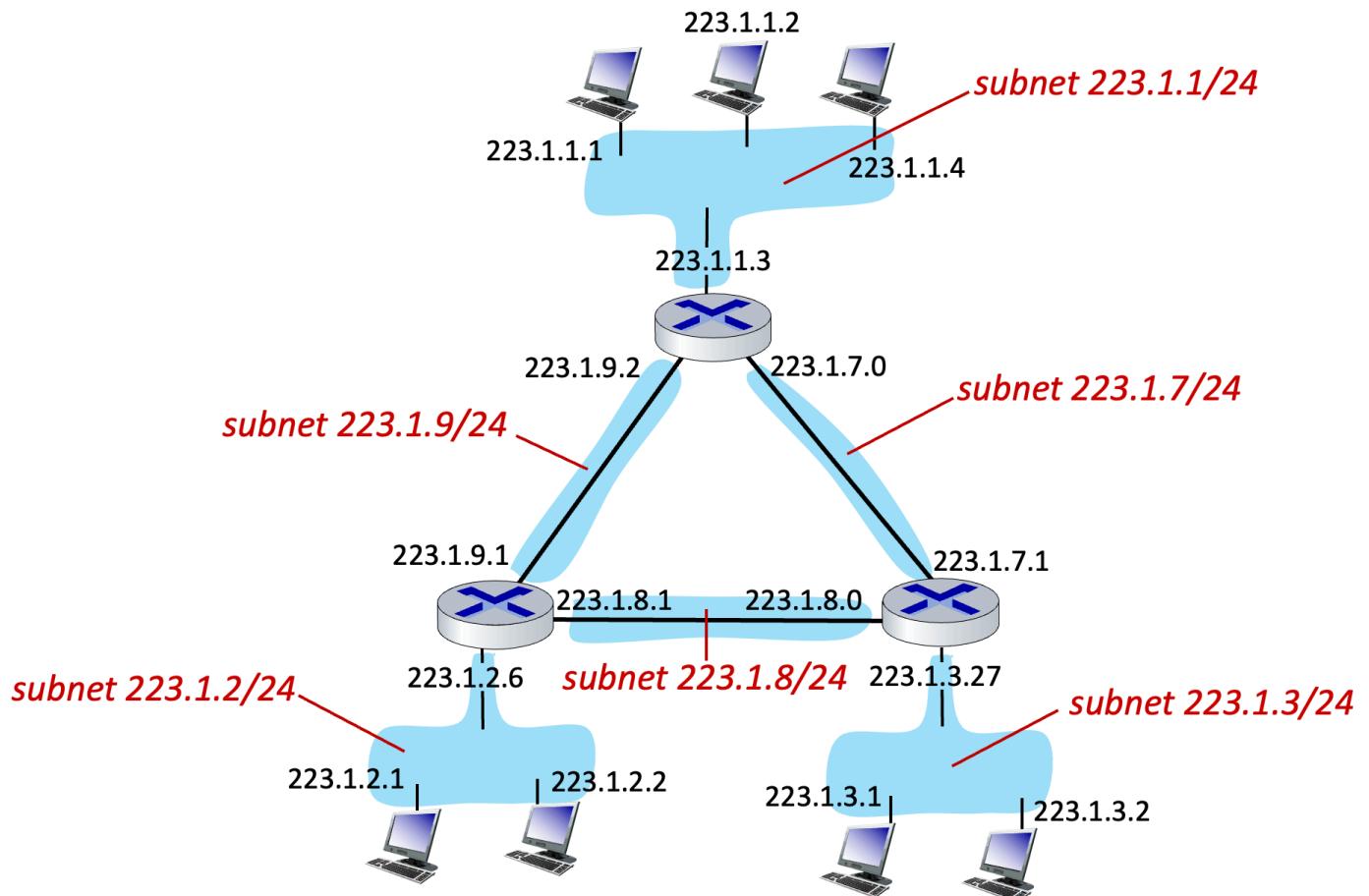
首先要知道的是 IP地址 不能标识主机 或者 路由器本身, 而是一个 **interface** (接口), 主机/路由器上的链路层接口

- **IP 地址**: 32位标示，对主机或者路由器的接口编址
- **接口**: 主机/路由器和物理链路的连接处
 - 路由器通常拥有多个接口
 - 主机也有可能有多个接口
 - IP地址和每一个接口关联
- **一个IP地址和一个接口相关联**



Subnets

- 不需要经过 intervening router (中间的路由器)，就可以物理上相互到达的 device interfaces (设备接口)
- 通过在主机号字段中拿一部分作为子网号，把两级 IP 地址划分为三级 IP 地址。
- IP 地址 = {< 网络号 >, < 子网号 >, < 主机号 >}

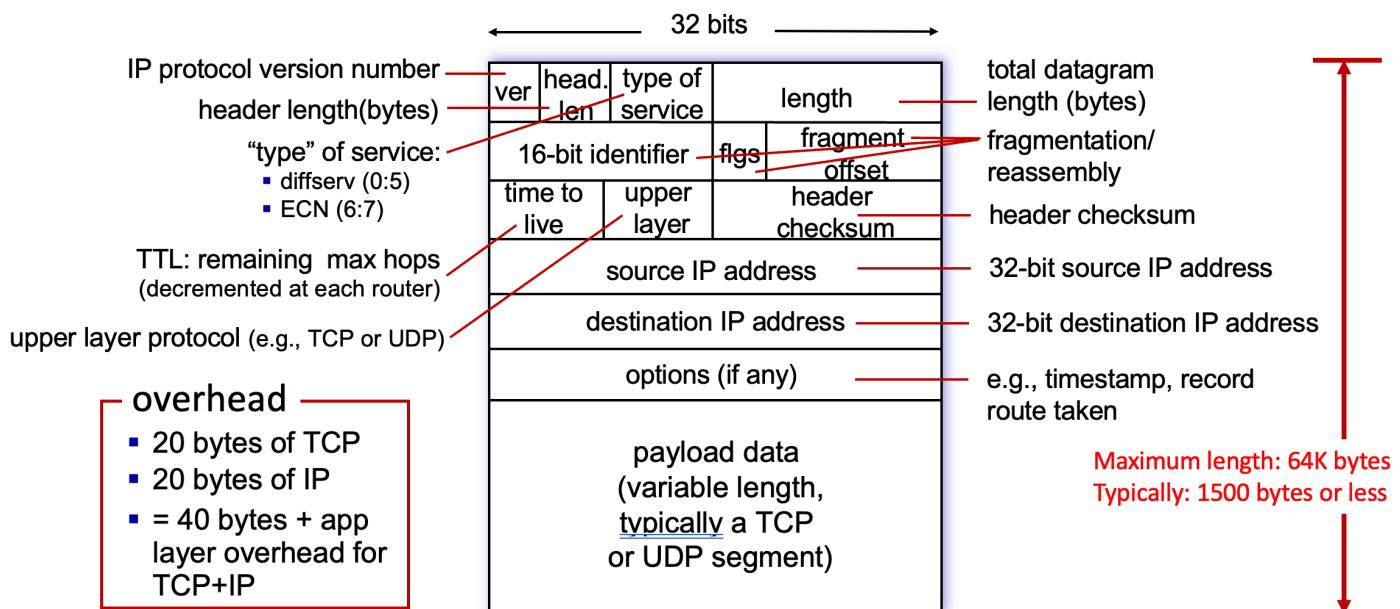


/24 代表前面24位，高位的24位是网络号，意味着后面8位数是主机号

- address format: **a.b.c.d/x**, where x is # bits in subnet portion of address

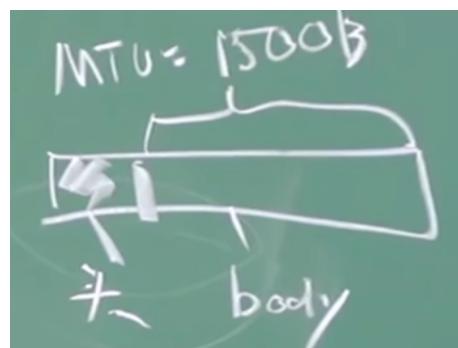


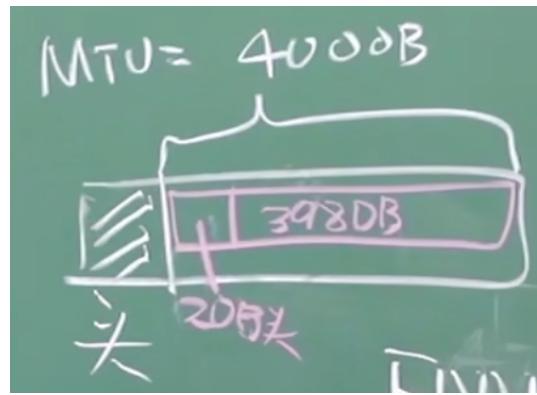
IP Datagram format



IP 分片和重组(Fragmentation & Reassembly)

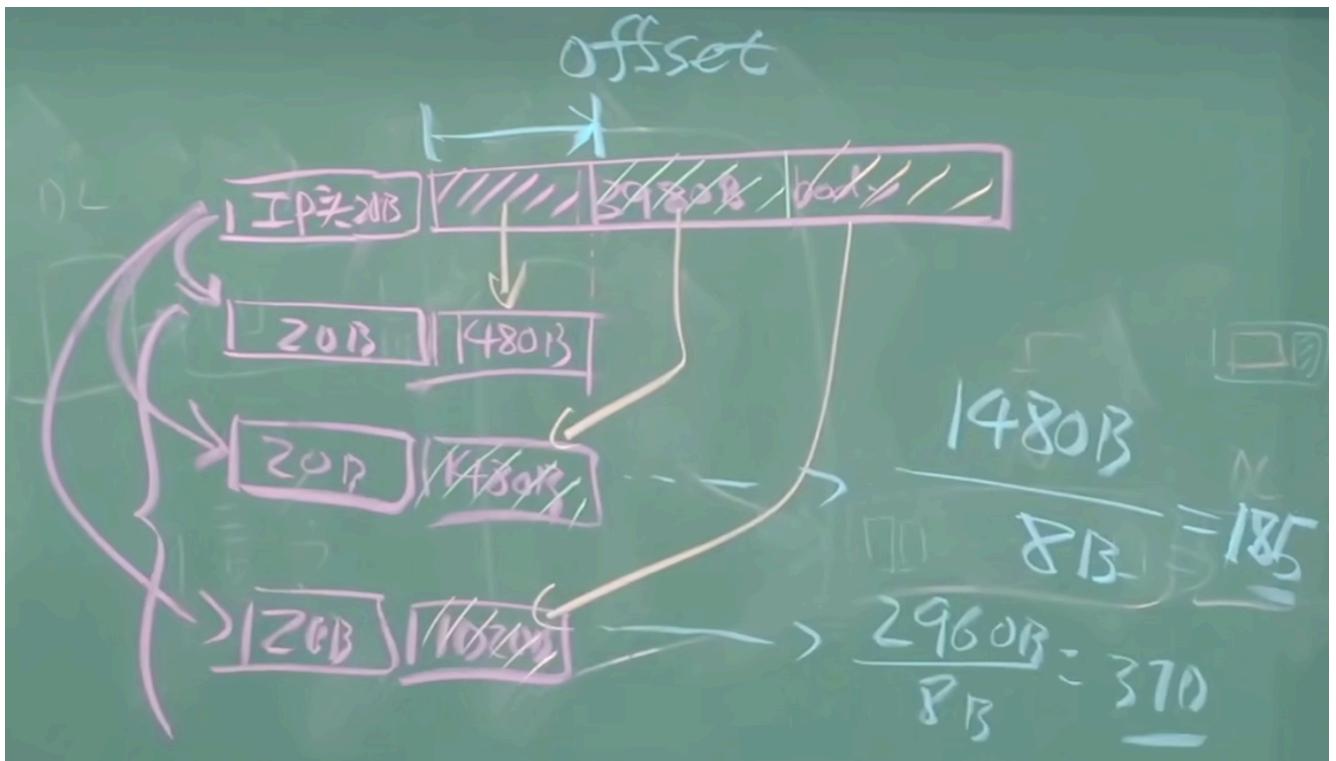
- 网络链路有MTU (最大传输单元) - 链路层帧所携带的最大数据长度





- 不能直接分片

- 分片后加上头部，相同的ID
- offset (偏移量) 当作序号，第一个分组的 offset = 0，第二个分组的 offset = 1480bits = 185Bytes，第二个分组的 offset = 2960bit = 370Bytes



例子

□ 4000 字节数据报

- 20字节头部
- 3980字节数据

	length =4000	ID =x	fragflag =0	offset =0	
--	-----------------	----------	----------------	--------------	--

一个大的数据报变成若干小的数据报

□ MTU = 1500 bytes

□ 第一片: 20字节头部+1480字节数据

- 偏移量: 0

	length =1500	ID =x	fragflag =1	offset =0	
--	-----------------	----------	----------------	--------------	--

□ 第二片: 20字节头部+1480字节数据 (1480字节应用数据)

- 偏移量: $1480/8=185$

	length =1500	ID =x	fragflag =1	offset =185	
--	-----------------	----------	----------------	----------------	--

□ 第三片: 20字节头部+1020字节数据 (应用数据)

- 偏移量: $2960/8=370$

	length =1040	ID =x	fragflag =0	offset =370	
--	-----------------	----------	----------------	----------------	--

偏移 (以8字节为单位) = $1480/8$

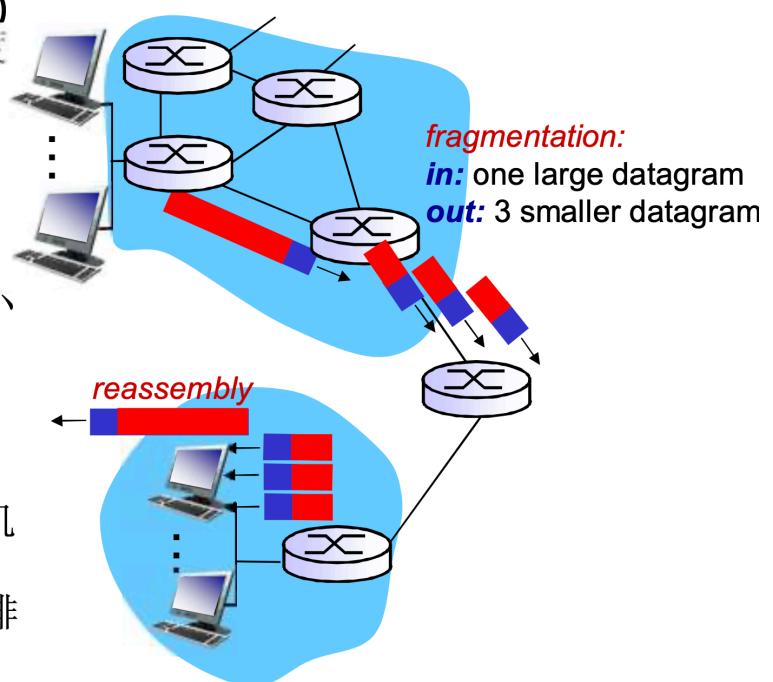
□ 网络链路有MTU (最大传输单元)

-链路层帧所携带的最大数据长度

- 不同的链路类型
- 不同的MTU

□ 大的IP数据报在网络上被分片 (“fragmented”)

- 一个数据报被分割成若干小的数据报
 - 相同的ID
 - 不同的偏移量
 - 最后一个分片标记为0
- “重组” 只在最终的目标主机进行
- IP头部的信息被用于标识，排序相关分片



Host如何获得一个IP地址?

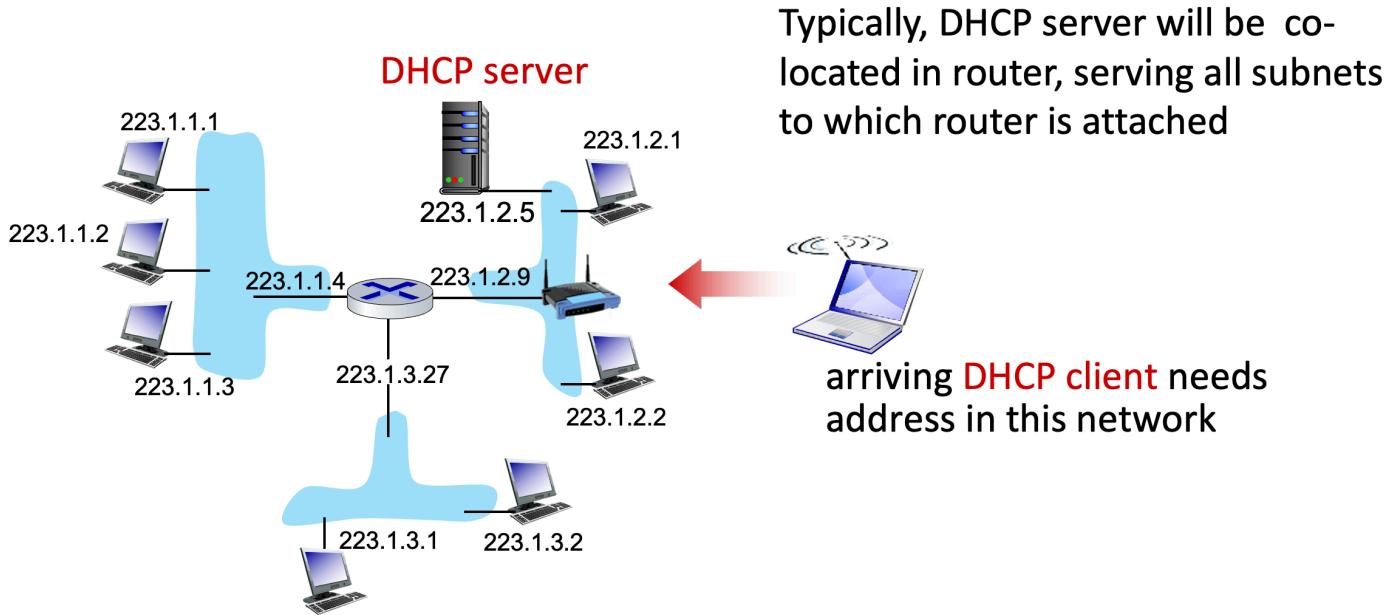
两种方法:

- **config file:** 系统管理员将地址配置在一个文件中
 - Wintel: control-panel -> network -> configuration -> tcp/ip -> properties
 - UNIX: /etc/rc.config
- **DHCP:** Dynamic Host Configuration Protocol: 从服务器中动态获得一个IP地址 (plug-and-play)

DHCP: Dynamic Host Configuration Protocol (UDP)

- 以便使用DHCP分配即插即用IP地址，网络将需要有一个DHCP服务器来执行该功能
- 当有一个客户端出现，他是一个想要IP地址的主机，将从DHCP服务器请求并接收IP地址，使用DHCP协议。
- 当主机离开网络时，他会放弃他的IP地址，然后可以被另一个主机重用或回收，或者可能由该主机在以后的时间点续订。
- 作用:** 允许主机在加入网络的时候，动态地从服务器那里获得IP地址
 - 可以更新对主机在用IP地址的租用期-租期快到了
 - 重新启动时，允许重新使用以前用过的IP地址
 - 支持移动用户加入到该网络(短期在网)

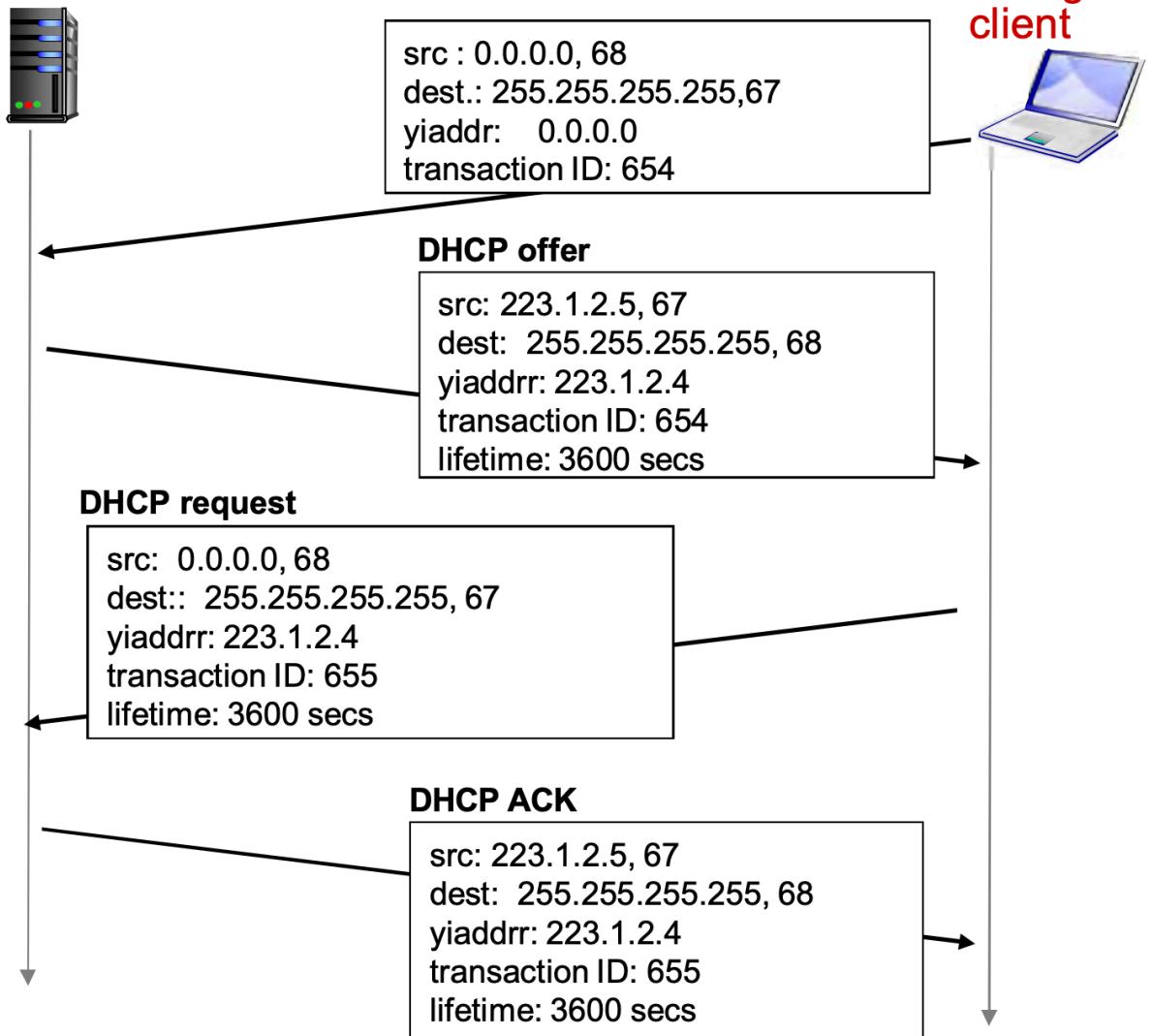
通常DHCP服务器将配置在路由器中，并为该路由器所连接的所有子网提供服务



4 important DHCP message overview:

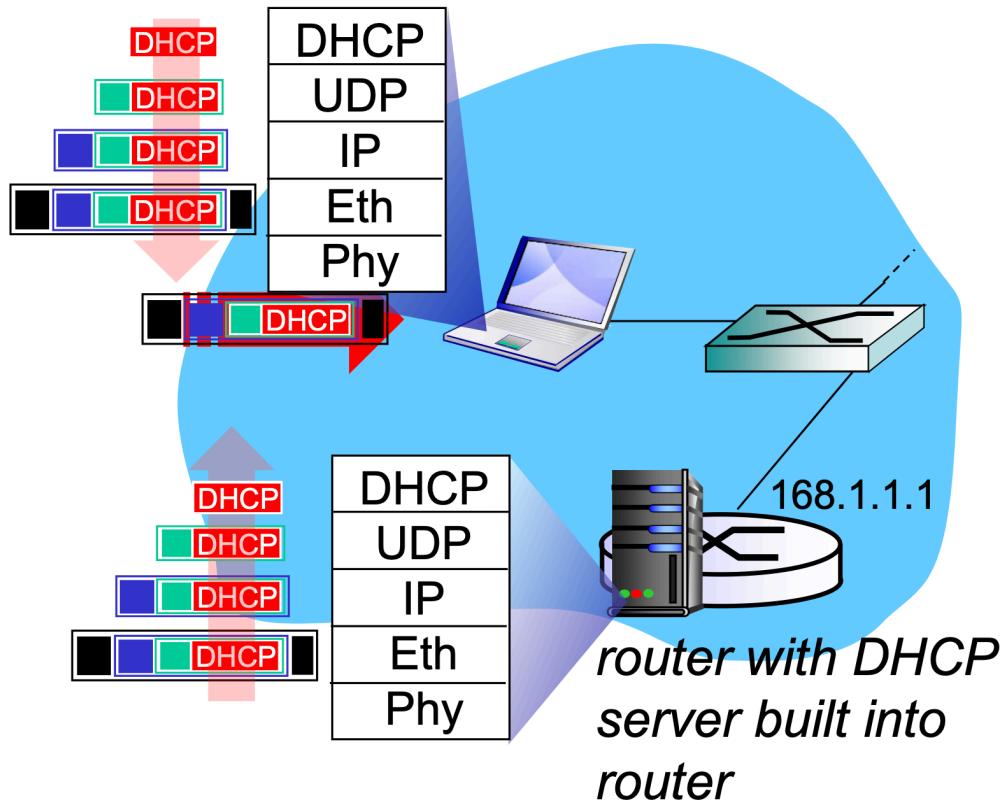
- host broadcasts (主机广播): **DHCP discover message** [optional] (广播: DHCP server在那里嘛)
- DHCP server responds (DHCP服务器提供报文相应): **DHCP offer message** [optional] (我是DHCP server, 这是你可以用的IP)
- host requests IP address (主机请求IP地址): **DHCP request message** (好的, 我接受这个IP地址)
- DHCP server sends address (DHCP服务器发送地址): **DHCP ack message** (好的, 你现在拥有这个IP地址)

DHCP server: 223.1.2.5



DHCP: 不仅仅可以返回IP addresses

- DHCP can return allocated **IP address** on subnet (可以返回IP地址)
- **address of first-hop router** for client (第一跳路由器的IP地址)
- **name and IP address of DNS sever** (DNS服务器的域名和IP地址)
- **network mask** (子网掩码, 为了划分网络号和主机号的)



一个ISP如何获得一个地址块?

ICANN: Internet Corporation for Assigned Names and Numbers

- 分配 addresses
- 管理 DNS
- 分配 domain names, 解决冲突

05. Network (Control Plane)

我们将关注单个路由器在每个路由器函数的数据平面中到更广泛的网络视图，研究路由问题，比如说如何确定从原地址到目的地，我们还将研究网络管理和网络配置的问题。

- **forwarding:** move packets from router's input to appropriate router output
- **routing:** determine route taken by packets from source to destination

data plane 数据平面

control plane 控制平面

5.1 Intro: control plane

2 种构建 network control plane 的方法:

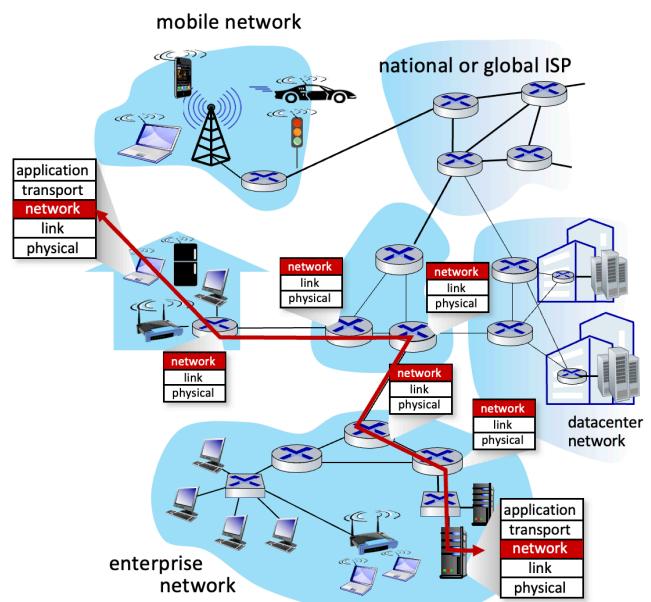
- per-router control (traditional)
- logically centralized control (software defined networking)

5.2 routing protocols

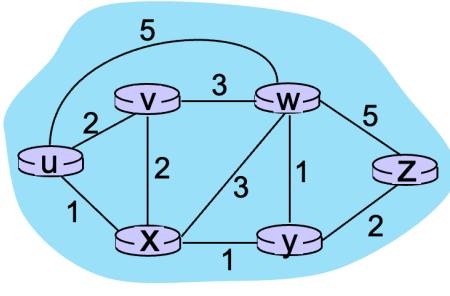
- **routing:** 路由就是计算从一个网络到其他网络如何走的问题，按照某种指标（传输延迟, 所经过的站点数目等）找到一条从源节点到目标节点的较好路径
- 任何路由算法的目标都是确定一条好的路径或一条好的路由：

Routing protocol goal: determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets traverse from given initial source host to final destination host
- **“good”:** least “cost”, “fastest”, “least congested”
- routing: a “top-10” networking challenge!



Graph abstraction: link costs 边和路径的代价



graph: $G = (N, E)$

$c_{w,z}$: cost of **direct** link connecting w and $z = 5$
 $c_{u,z}$: cost of **indirect** link connecting u and $z = \infty$

- 代价可能总为 1
- 或是 链路带宽的倒数
- 或是 拥塞情况的倒数

N : set of **routers** = { u, v, w, x, y, z }

E : set of **links** = { $(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)$ }

cost of path $(x_1, x_2, x_3, \dots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

key question: what is the least-cost path between u and z ?

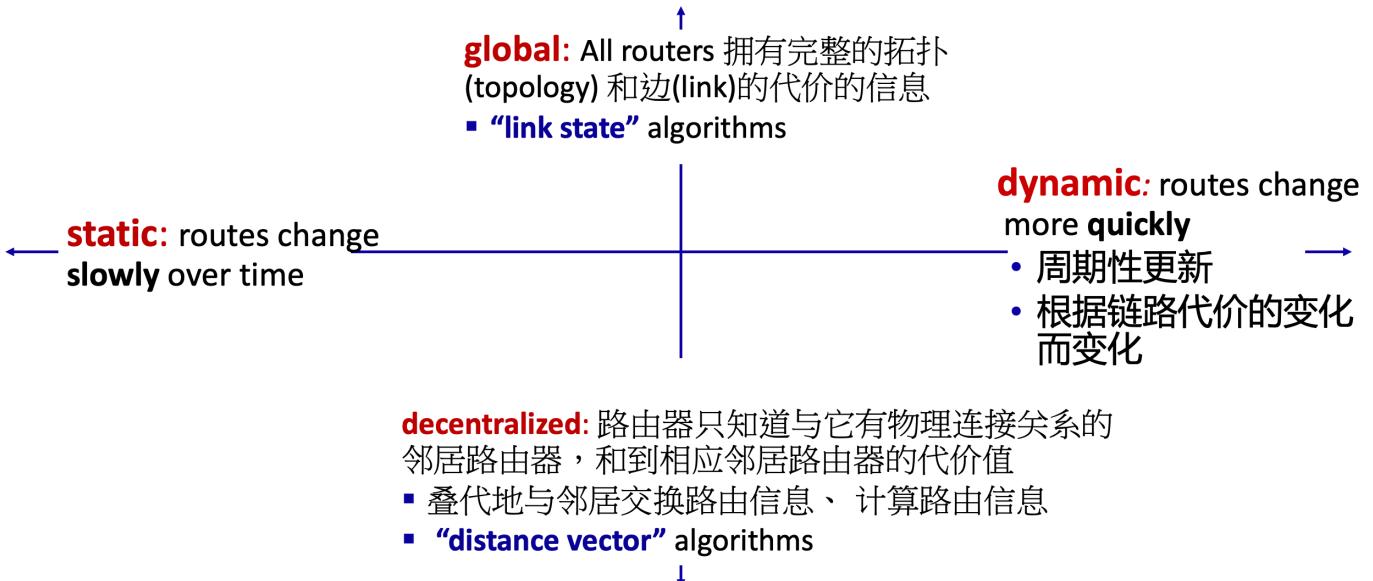
routing algorithm: algorithm that finds that least cost path

Routing algorithm classification

Routing algorithm 分为 Static or Dynamic, 也可以分为 Global or Decentralized.

尽管它们有相同的共同目标, 即计算最短路径, 最小成本路径, 但他们实际上计算最短路径的方式不一样:

- **Static:** routes change **slowly** over time
- **Dynamic:** routes change more **quickly**, 周期性更新, 根据链路代价的变化而变化
- **Global/centralized:** all routers 拥有完整的拓扑(topology) 和边(link)的代价的信息, 算法必须知道网络中每个链路的成本 ("link state" algorithms)
- **Decentralized:** 路由器只知道与它有物理连接关系的邻居路由器, 路由器以迭代、分布式的方式计算出最低开销路径。没有节点拥有关于所有网络链路开销的完整信息 ("distance vector" algorithms)



5.2.1 Link state (Dijkstra)

缺点: Dijkstra算法起初选了一个最优的path，于是都用这个path传，最后会导致这个path的堵塞，Dijkstra算法就会继续选择另一个最优的path.....一直这样反反复复，切换path的行为就叫做路径的振荡

Dijkstra's link-state routing algorithm

- 链路状态算法中，每个结点(经广播)与所有其他结点交谈，但它仅告诉它们与它直接相连链路的费用。
- **centralized:** network topology, 所有节点都知道每个链路的成本，通过“link state broadcast”实现，所有节点具有相同信息
- **computes least cost paths** from one node (“source”) to all other nodes
 - gives **forwarding table** for that node
- **iterative:** after k iterations, know least cost path to k destinations

2类节点：

- **临时节点(tentative node):** 还没有找到从源节点到此节点的最优路径的节点
- **永久节点(permanent node) N'**:已经找到了从源节点到此节点的最优路径的节点

c(i,j): 从节点*i* 到 *j*链路代价(初始状态下非相邻节点之间的链路代价为 ∞)

D(v): 从源节点到节点*V*的当前路径代价(节点的代价)

p(v): 从源到节点*V*的路径前序节点

N': 当前已经知道最优路径的的节点集合(永久节点的集合)

□ LS路由选择算法的工作原理

○ 节点标记：每一个节点使用($D(v), p(v)$) 如：

(3,B)标记

⑩ $D(v)$ 从源节点由已知最优路径到达本节点的距离

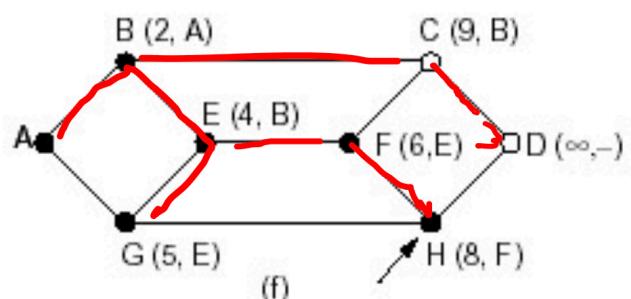
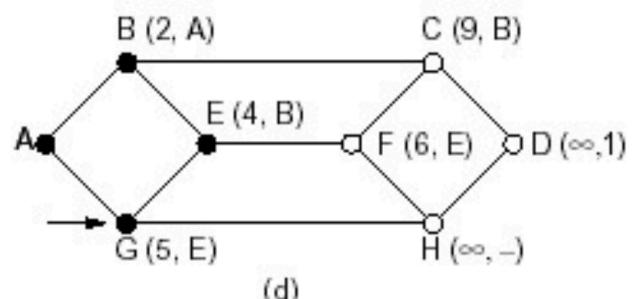
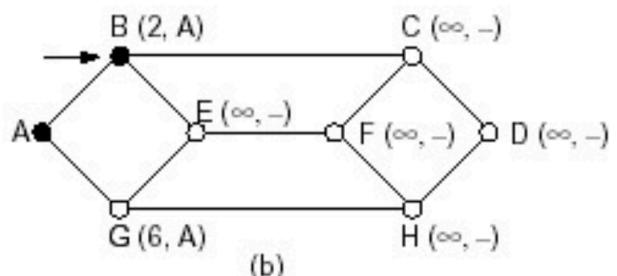
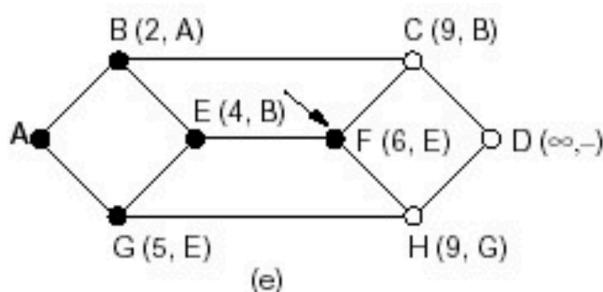
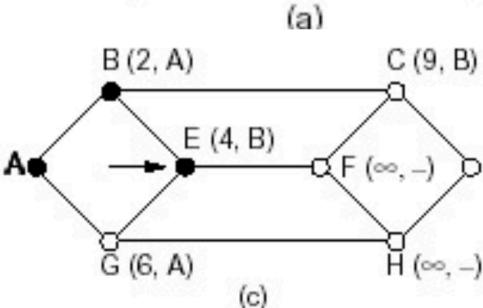
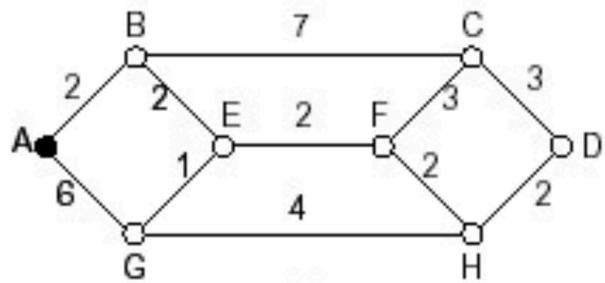
⑩ $P(v)$ 前序节点来标注

algorithm complexity: n nodes, each iteration needs to check all nodes, w, not in N' , $n(n+1)/2$ comparisons:
 $O(n^2)$

Dijkstra's link-state routing algorithm

```
1 Initialization:  
2    $N' = \{u\}$     // 计算从U到所有其他节点的最小代价路径, 除了源节点外, 所有节点都为临时节点  
3   for all nodes v  
4     if v adjacent to u          // U最初只知道直接邻居的直接路径开销  
5       then  $D(v) = c_{u,v}$       // 但可能不是最低成本  
6     else  $D(v) = \infty$         // 节点代价除了与源节点代价相邻的节点外, 都为∞  
7  
8 Loop  
9   find w not in  $N'$  such that  $D(w)$  is a minimum // 从所有临时节点中找到一个代价最小的  
10  add w to  $N'$  // 然后将他变成永久节点(当前节点)W  
11  update  $D(v)$  for all v adjacent to w and not in  $N'$ : // 所有在临时节点集合中的邻节点(V)  
12   $D(v) = \min(D(v), D(w) + c(w,v))$  // 如  $D(v) > D(w) + c(w,v)$ , 则重新标注此点,  $(D(W)+C(W,V), W)$   
13  /* 新的到v的最小路径代价要么是旧的到v的最小路径代价要么是已知的  
14  到w的最小代价路径加上从w到v的直接代价 */  
15 until all nodes in  $N'$ 
```

Examples 01



Examples 02

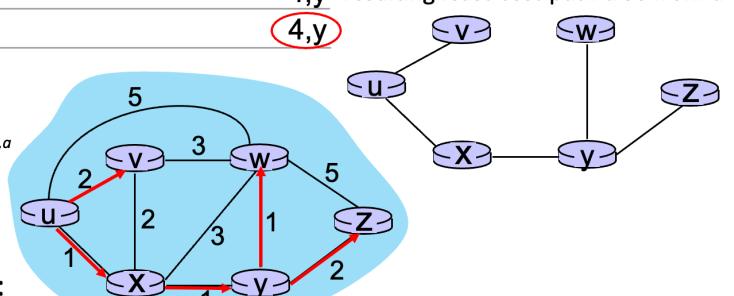
Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyw				4,y	resulting least-cost-path tree from u
5	uxvwz					

Initialization (step 0): For all a : if a adjacent to then $D(a) = c_{u,a}$

find a not in N' such that $D(a)$ is a minimum
add a to N'

update $D(b)$ for all b adjacent to a and not in N' :

$$D(b) = \min(D(b), D(a) + c_{a,b})$$



resulting forwarding table in u

destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

algorithm complexity: n nodes

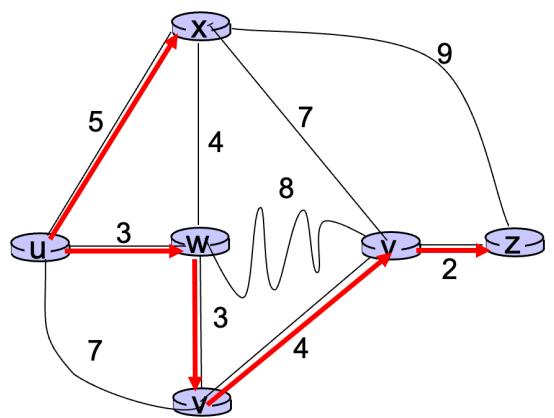
- each of n iteration: need to check all nodes, w , not in N
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n \log n)$

route from u to v directly

route from u to all other destinations via x

Examples 03

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w	5,u	11,w	∞	
2	uwx	6,w		11,w	14,x	
3	uwxv		10,v	14,x		
4	uwxvy			12,y		
5	uwxvyz					

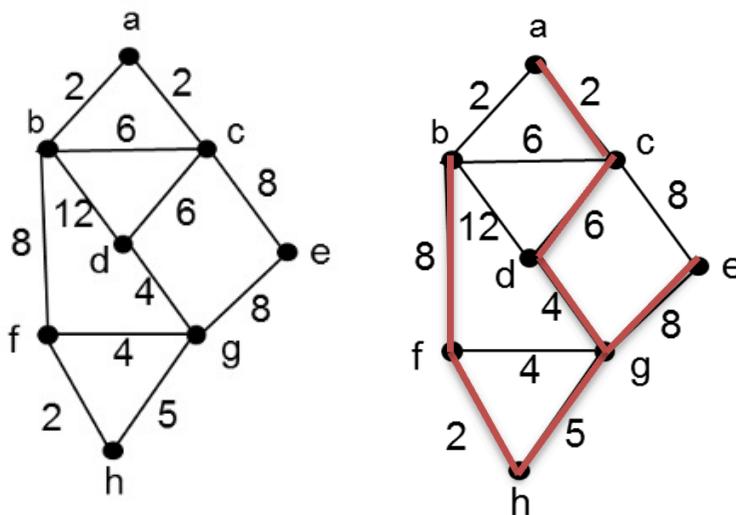


notes:

- construct least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

Examples 04

Consider the network below. Please use Dijkstra's shortest-path algorithm to compute the shortest path from node h to all network nodes.



$h \rightarrow f \rightarrow b \rightarrow a: 12$
 $h \rightarrow f \rightarrow b: 10$
 $h \rightarrow f \rightarrow b \rightarrow a \rightarrow c: 14$
 $h \rightarrow g \rightarrow d: 9$
 $h \rightarrow g \rightarrow e: 13$
 $h \rightarrow f: 2$
 $h \rightarrow g: 5$

Step	N'	$D(f), p(f)$	$D(g), p(g)$	$D(e), p(e)$	$D(c), p(c)$	$D(d), p(d)$	$D(b), p(b)$	$D(a), p(a)$
0	h	2,h	5,h	infinite	infinite	infinite	infinite	infinite
1	hf		5,h	infinite	infinite	infinite	10,f	infinite
2	hfg			13,g	infinite	9,g	10,f	infinite
3	hfgd			13,g	15,d		10,f	infinite
4	hfgdb			13,g	15,d			12,b
5	hfgdba			13,g	14,a			
6	hfgdbae				14,a			
7	hfgdbaec							

5.2.2 Distance vector (Bellman-Ford equation)

- 距离向量算法中，每个结点仅与它的直接相连的邻居交谈，但它为其邻居提供了从它自己到网络中(它所知道的)所有其他结点的最低费用估计。

Bellman-Ford equation

Let $D_x(y)$: cost of **least-cost path** from x to y .

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

least-cost-path cost from neighbor v to destination y

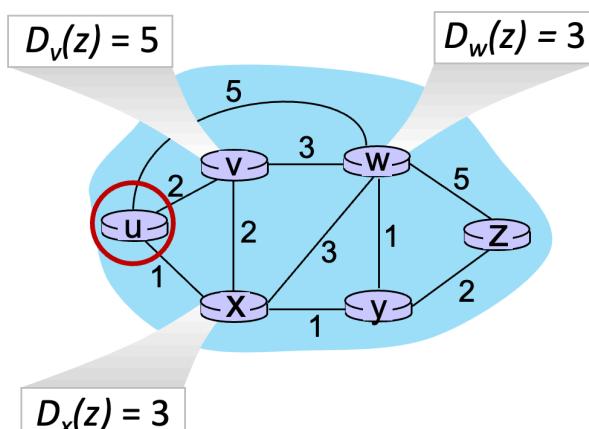
取所有x的neighbor最小的neighbor, 称为v direct cost of link from x to neighbor v

Suppose that u 's neighboring nodes, x, v, w , know that for destination z :

明显的, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

Bellman-Ford equation says:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,x} + D_x(z), \\ &\quad c_{u,w} + D_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$



那个能够达到目标 z 最小代价的节点 x , 就在到目标节点的下一条路径上, 在转发表中使用

核心思路:

- ❖ 每个节点都将自己的距离矢量估计值传送给邻居，定时或者 DV有变化时，让对方去算
- ❖ 当x从邻居收到DV时，自己运算，更新它自己的距离矢量
 - ❖ 采用B-F equation:
$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\}$$
 对于每个节点 $y \in N$

X往y的代价 x到邻居v代价 v声称到y的代价
- ❖ $D_x(y)$ 估计值最终收敛于实际的最小代价值 $d_x(y)$
 - ❖ 分布式、迭代算法

each node:



Iterative (迭代), asynchronous(异步式):
每次 local iteration 会导致:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: 每个节点仅仅当他的DV changes时，才向邻居通告

- neighbors then notify their neighbors – only if necessary
- no notification received, no actions taken!

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

node x table

	cost to		
	x	y	z
from	0	2	7
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

node y table

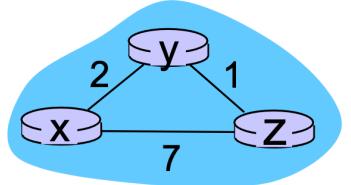
	cost to		
	x	y	z
from	∞	∞	∞
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

	cost to		
	x	y	z
from	∞	∞	∞
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	cost to		
	x	y	z
from	0	2	3
x	0	2	3
y	2	0	1
z	7	1	0

	cost to		
	x	y	z
from	0	2	3
x	0	2	3
y	2	0	1
z	3	1	0



time

Distance vector: link cost changes

在 Distance vector 中，当节点检测到本地链路开销变化，更新路由信息，重新计算 distance vector，如果DV发生变化，通知邻居

"good news travels fast"

- t0: y检测链路开销变化，更新DV，通知邻居。
- t1: z接收y的更新，更新它的表，计算新的最小代价到x，将它的DV发送给它的邻居。
- T2: y接收z的更新，更新它的距离表。Y的最小代价不变，所以Y不会向z发送消息。

"bad news travels slow" – count-to-infinity problem:

- many iterations before algorithm stabilizes

- y sees direct link to x has new cost 60, but z has said it has a path at cost of 5. So y computes “my new cost to x will be 6, via z); notifies z of new cost of 6 to x.
 - z learns that path to x via y has new cost 6, so z computes “my new cost to x will be 7 via y), notifies y of new cost of 7 to x.
 - y learns that path to x via z has new cost 7, so y computes “my new cost to x will be 8 via y), notifies z of new cost of 8 to x.
 - z learns that path to x via y has new cost 8, so z computes “my new cost to x will be 9 via y), notifies y of new cost of 9 to x.
- ...

5.2.3 Link State & Distance Vector 算法的比较

- **Link State:** 需要全局信息。因此，当在每台路由器中实现时，每个节点(经广播)与所有其他节点通信，但仅告诉它们与它直接相连链路的开销。我们通过快速比较它们各自的属性来总结所学的链路状态与距离向量算法。(会有振荡: oscillations)
- **Distance Vector:** 每个节点仅与它的直接相连的邻居交谈，但它为其邻居提供了从它自己到网络中(它所知道的)所有其他节点的最低开销估计。
- 记住 N 是节点(路由器)的集合，而 E 是边(链路)的集合。

LS 和 DV 算法的比较

消息复杂度 (DV胜出)

- LS: 有 n 节点, E 条链路, 发送报文 $O(nE)$ 个
 - 局部的路由信息；全局传播
- DV: 只和邻居交换信息
 - 全局的路由信息，局部传播

收敛时间 (LS胜出)

- LS: $O(n^2)$ 算法
 - 有可能震荡
- DV: 收敛较慢
 - 可能存在路由环路
 - count-to-infinity 问题

健壮性: 路由器故障会发生什么 (LS胜出)

LS:

- 节点会通告不正确的链路代价
- 每个节点只计算自己的路由表
- 错误信息影响较小，局部，路由较健壮

DV:

- DV 节点可能通告对全网所有节点的不正确路径代价
 - ⑩ 距离矢量
- 每一个节点的路由表可能被其它节点使用
 - ⑩ 错误可以扩散到全网

2种路由选择算法都有其优缺点，而且在互联网上都有应用

5.3 intra-AS routing

Making routing scalable:

- **scale:** 拥有数十亿个目的地，不能将所有目的地存储在路由表中!
- **administrative autonomy:** 因特网=网络中的网络，每个网络管理员可能都想控制自己网络中的路由。因特网是 ISP 的网络，其中每个 ISP 都有它自己的路由器网络。ISP 通常希望按自己的意愿运行路由器。一个组织应当能够按自己的愿望运行和管理其网络，还要能将其网络与其他外部网络连接起来。

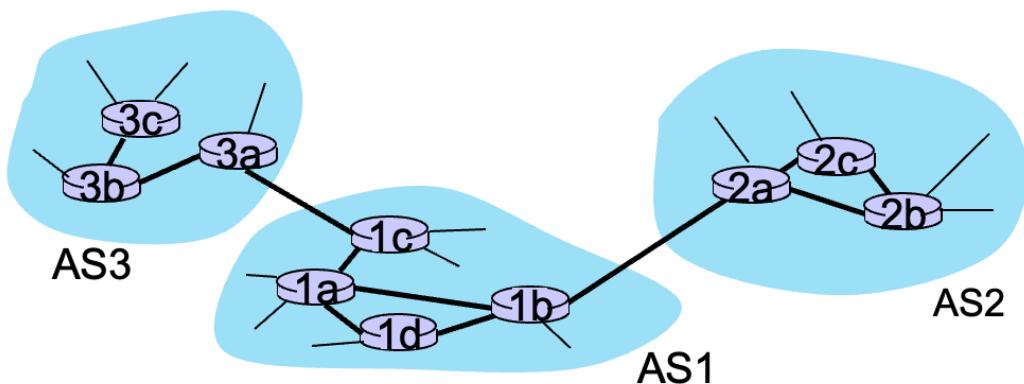
这两个问题都可以通过将路由器组织进 **自治系统 (Autonomous System , AS)** 来解决。

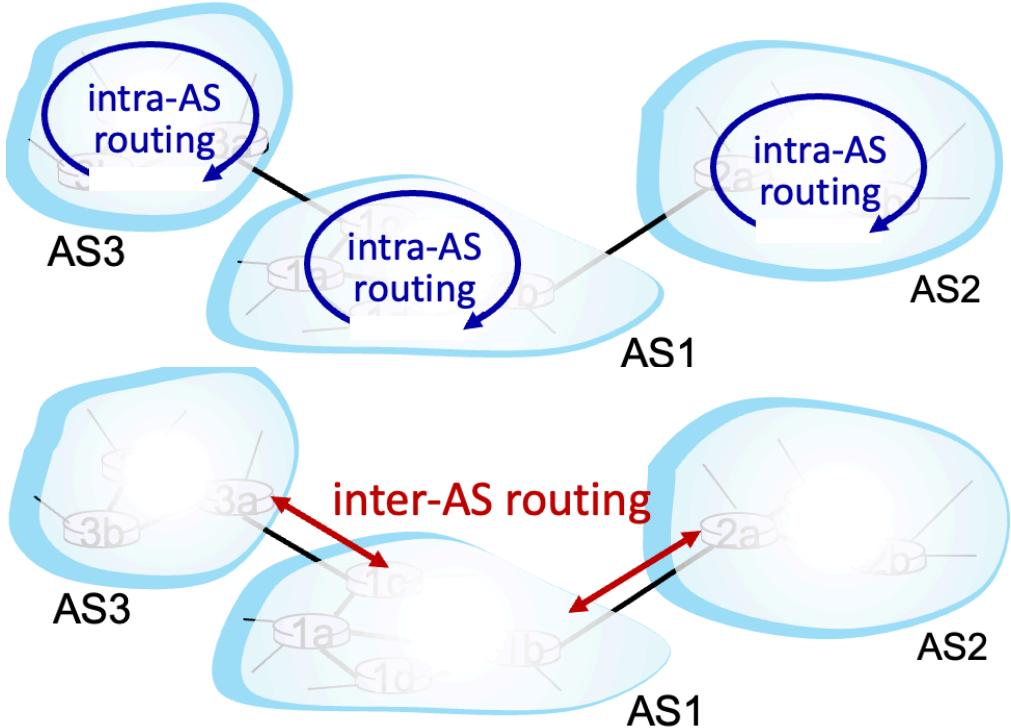
- 其中每个 **Autonomous System** 由一组通常处在相同管理控制下的路由器组成。
- 通常在一个ISP中的路由器以及互联它们的链路构成一个 **Autonomous System**
- 然而，某些 ISP 将它们的网络划分为多个 **Autonomous System**。特别是某些一级 ISP 在其整个网络中使用一个庞大的 **Autonomous System**，而其他 ISP 则将它们的 ISP 拆分为数十个互联的 **Autonomous System**
- 一个自治系统由其全局唯一的 **自治系统 (Autonomous System , AS)号 (ASN)** 所标识 [RFC1930]。就像IP 地址那样，AS 号由 ICANN 区域注册机构所分配 [ICANN 2016] 。

Internet approach to scalable routing

将 routers 聚合到称为 "autonomous systems" (AS) 的区域 (又名 "domains")

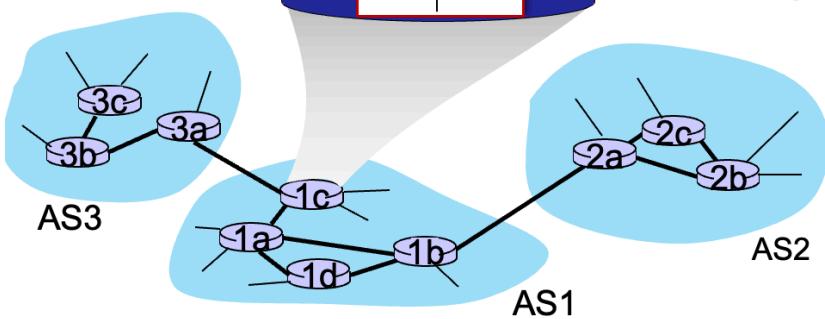
- intra-AS (aka “intra-domain”)** 内部: **inter-AS (aka “inter-domain”)** 域间:
- routing among *within same AS (“network”)* routing *among AS'es*
- AS 中的所有 routers 必须运行相同的 intra-domain protocol
 - 不同 AS 中的 routers 可以运行不同的 intra-domain routing protocols
 - **gateway router(网关路由器):** 在其自己的 AS 的“edge”，具有指向其他 AS 中的路由器的 links





forwarding table 通过 intra- and inter-AS routing algorithms 进行配置

- intra-AS routing determine AS 内目标的条目
- inter-AS & intra-AS determine 外部目的地的条目



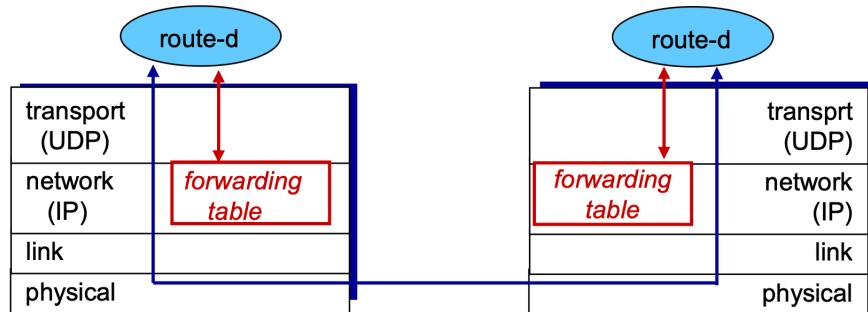
5.4 Three intra domain routing protocols

RIP

RIP: Routing Information Protocol [RFC 1723] (Distance vector 算法)

- 在 1982 年发布的 BSD-UNIX 中实现
- distance metric: # hops (max = 15 hops), 每条 link cost = 1
- DV 每隔 30 秒和邻居交换一次 DV in response message (advertisement 通告)
- 每个 advertisement 包括: 最多 25 个 destination subnets (in IP addressing sense)

- routing tables managed by **application-level** process called route-d (daemon)
 - advertisements periodically sent in UDP packets



OSPF (LS)

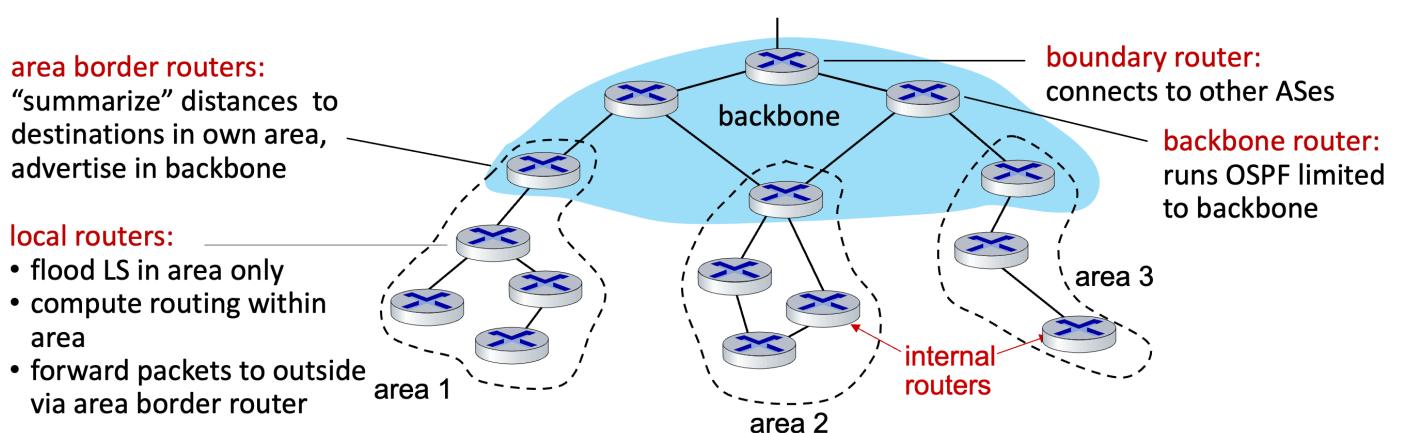
OSPF: Open Shortest Path First [RFC 2328]

- "open": **publicly available** 指路由选择协议规范是公众可用的(与之相反的是 Cisco 的 EIGRP 协议)
- 使用 **link-state routing**
 - 每个路由器将 OSPF link-state 通告 (advertisements) (直接通过IP而不是使用TCP/UDP)发送到整个AS中的所有其他路由器
 - 多链路 costs metrics 可能: bandwidth, delay
 - 每个路由器都有完整的 topology (拓扑结构), 采用Dijkstra算法计算转发表
- IS-IS路由协议: 几乎和OSPF一样
- 安全:所有 OSPF messages authenticated (消息验证, 防止恶意入侵)
- 支持在单个 AS 中的层次结构:

Hierarchical OSPF

▪ two-level hierarchy: local area, backbone.

- link-state advertisements flooded only in area, or backbone
- each node has detailed area topology; only knows direction to reach other destinations



IGRP

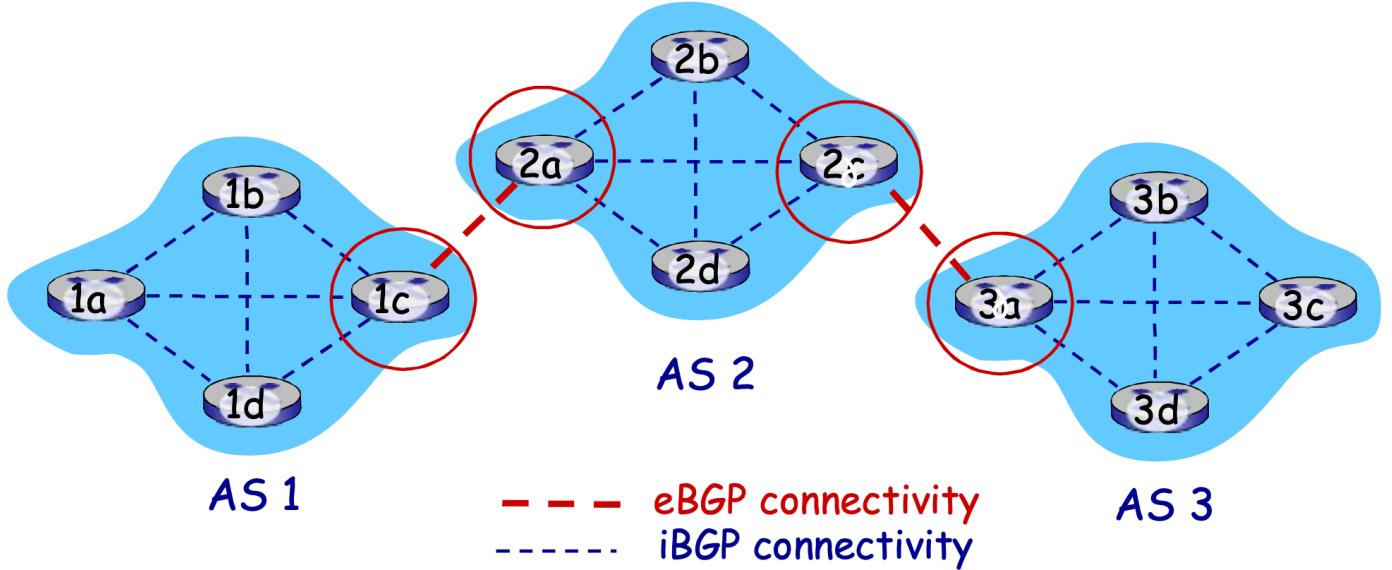
IGRP: Interior Gateway Routing Protocol

- DV based
- formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])

5.4 routing among the ISPs: BGP (DV)

互联网AS间路由: BGP

- **BGP (Border Gateway Protocol):** 自治区域间路由协议“事实上的”标准
 - “将互联网各个AS粘在一起的胶水”
- BGP 提供给每个AS以下方法:
 - **eBGP:** 从相邻的ASes那里获得子网可达信息
 - **iBGP:** 将获得的子网可达信息传遍到AS内部的所有路由器
 - 根据子网可达信息和策略来决定到达子网的“好”路径
- 允许子网向互联网其他网络通告“**我在这里**”
- 基于距离矢量算法（路径矢量）
 - 不仅仅是距离矢量，还包括到达各个目标网络的详细路径（AS序号的列表）能够避免简单DV算法的路由环路问题



网关路由器同时运行eBGP和iBGP协议

06. Link Layer and LANs 点到点传输层功能

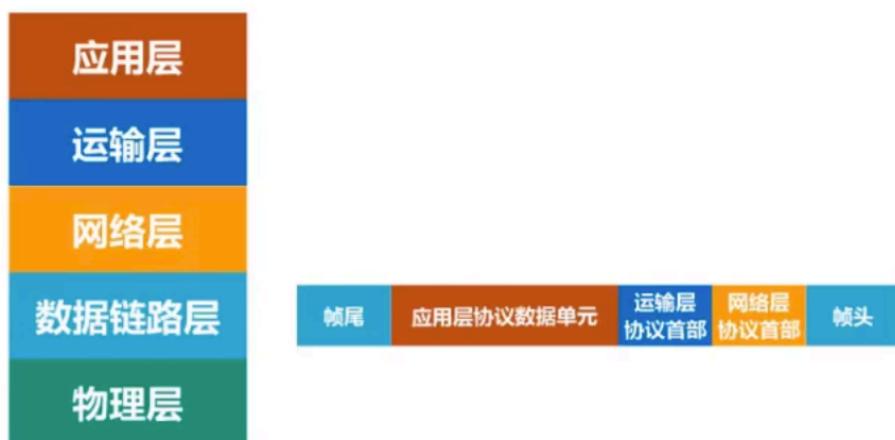
- hosts and routers = nodes
- 沿着通信路径,连接个相邻节点的 communication path = links (wired, wireless, LANs)
- 第二层协议数据单元帧 frame, encapsulates datagram
- 数据报(分组)在不同的链路上以不同的链路协议传送:
 - 第一跳链路: Ethernet 以太网 (先坐飞机)
 - 中间链路: 帧中继链路 (然后高铁)
 - 最后一跳: 802.11 (最后轿车)
- 不同的链路协议提供不同的服务 (比如在链路层上提供(或没有)可靠数据传送)

- **链路 (Link)** 就是从一个结点到相邻结点的一段物理线路，而中间没有任何其他的交换结点。
- **数据链路 (Data Link)** 是指把实现通信协议的硬件和软件加到链路上，就构成了数据链路。
- **数据链路层以帧为单位传输和处理数据。**



6.1 services

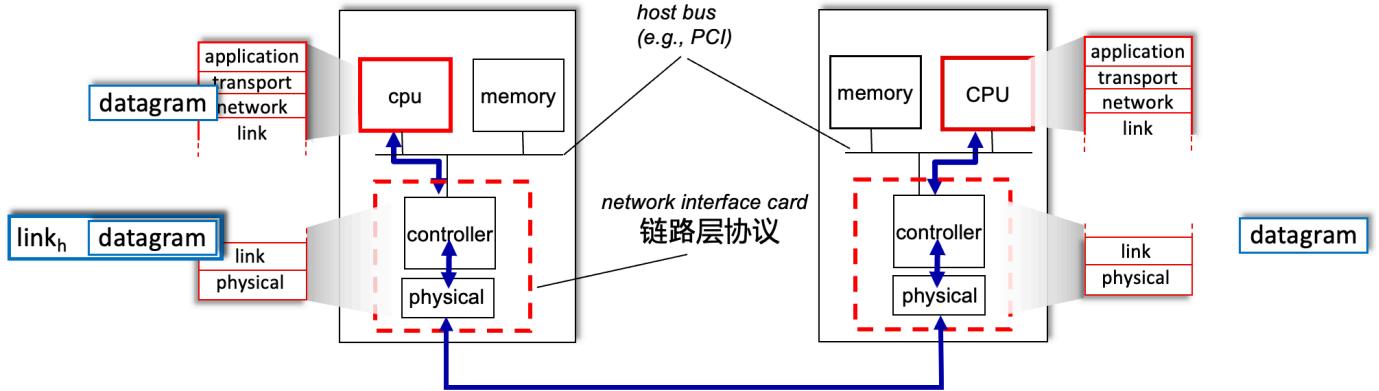
- **framing, link access:**
 - 将数据报封装在帧中，加上 **header** (帧头)、**trailer** (帧尾部)
 - 如果采用的是共享性介质，信道接入获得信道访问权
 - 在帧头部使用“MAC”(物理)地址来标示源和目的 (不同于IP地址)



- **reliable delivery between adjacent nodes:**
 - 在低出错率的链路上(光纤和双绞线电缆)很少使用
 - 在无线链路经常使用:出错率高
- **flow control:** 使得相邻的发送和接收方节点的速度匹配
- **error detection:** 差错由信号衰减和噪声引起，接收方检测出的错误: 通知发送端进行重传或丢弃帧
- **error correction:** 接收端检查和纠正bit错误，不通过重传来纠正错误
- **half-duplex and full-duplex:**
 - 半双工 (half-duplex): 链路可以双向传输，但一次只能收或只能发
 - 全双工 (full-duplex): 但一次可以同时收或发

Network interface card 网卡

- 链路层功能在 适配器 adaptor (network interface card NIC) 上实现或者在一个 芯片组 (chip) 上
- 接到主机的系统总线上，硬件、软件和固件的综合体



❖ 发送方：

- 在帧中封装数据报
- 加上差错控制编码，实现RDT和流量控制功能等

适配器是半自治的
实现了链路和物理层功能

❖ 接收方

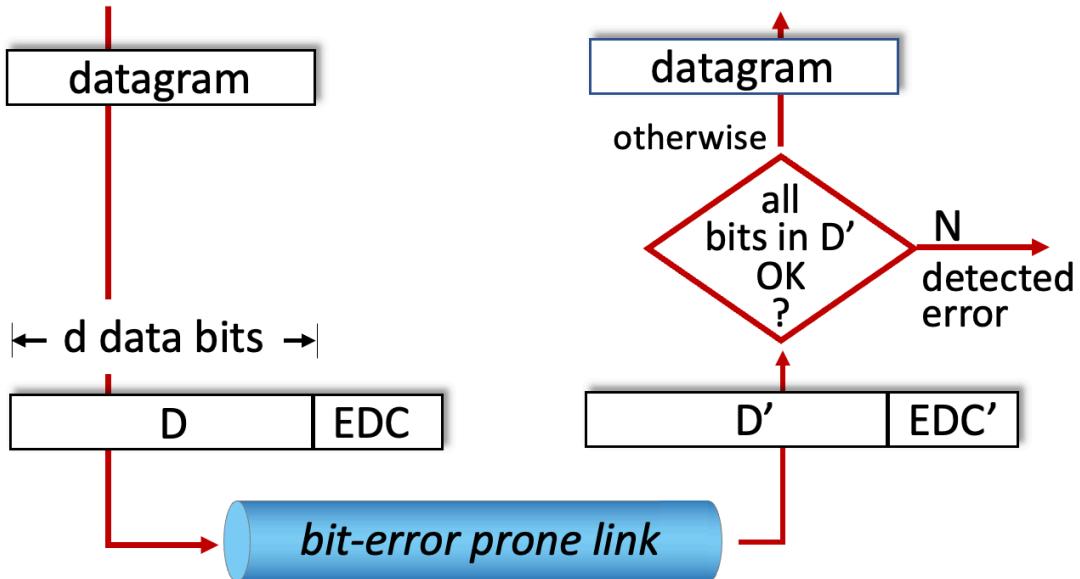
- 检查有无出错，执行 rdt 和流量控制功能等
- 解封装数据报，将至交给上层

6.2 error detection, correction

6.2.1 Error detection 错误检测

- EDC: 差错检测和纠正位(冗余位) error detection and correction bits (e.g., redundancy)
- D: 数据由差错检测保护，可以包含头部字段 data protected by error checking, may include header fields

错误检测不是100%可靠的！协议会漏检一些错误，但是很少，更长的EDC字段可以得到更好的检测和纠正效果。

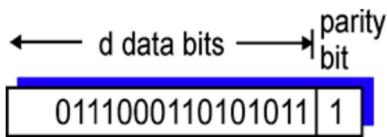


6.2.2 Parity checking 奇偶校验

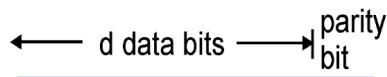
Even Parity (偶校验): 发送方只需包含一个附加的比特, 选择它的值, 使得这 $d+1$ 比特 (初始信息加上一个校验比特) 中1的总数是偶数

single bit parity:

- detect single bit errors



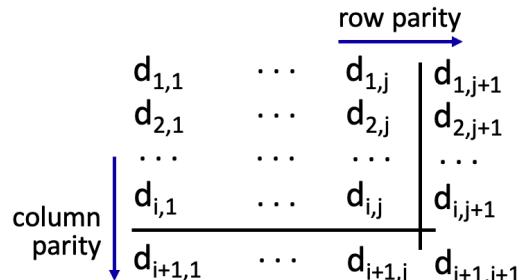
no error



parity error

two-dimensional bit parity:

- detect *and correct* single bit errors



no errors:	1 0 1 0 1 1
	1 1 1 1 0 0
	0 1 1 1 0 1
	1 0 1 0 1 0

detected and correctable single-bit error:	1 0 1 0 1 1
	1 0 1 1 0 0
	0 1 1 1 0 1
	1 0 1 0 1 0

6.2.3 Internet checksum 因特网检验和

与后面要讨论的常用于链路层的 CRC 相比, 它们提供相对弱的差错保护

目标: 检测在传输报文段时的错误（如位翻转），（注：
仅仅用在传输层）

发送方:

- 将报文段看成16-bit整数
- 报文段的校验和: 和 (1'的补码和)
- 发送方将checksum的值放在‘UDP校验和’字段

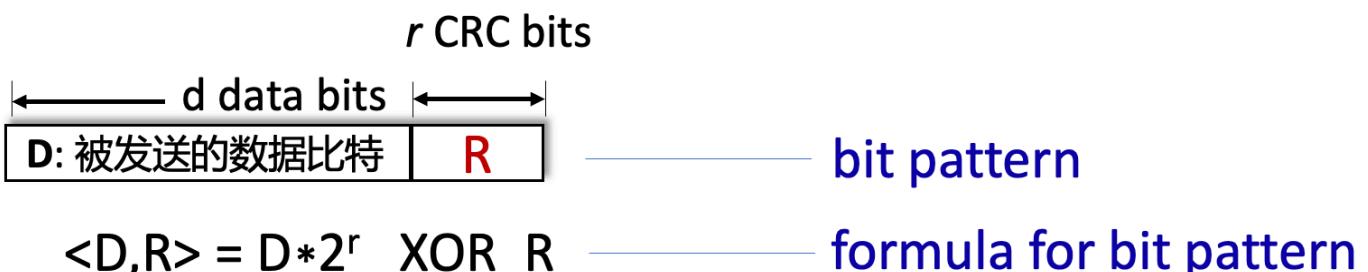
接收方:

- 计算接收到的报文段的校验和
- 检查是否与携带校验和字段值一致：
 - 不一致：检出错误
 - 一致：没有检出错误，但可能还是有错误

有更简单的检查方法
全部加起来看是不是全1

6.2.4 Cyclic Redundancy Check (CRC) 循环冗余校验

- 强大的差错检测码
- 将数据比特 D, 看成是二进制的数据
- 生成多项式 G: 双方协商 r+1位模式(r次方), 生成和检查所使用的位模式



所有 CRC 计算采用 模2算术来做 = 异或 XOR (相同为0, 不同为1)

乘以 2^k 就是以一种比特模式左移 k 个位置, 假设 D=101110, d=6 (位数), G=1001 和 r=3 (向左移3位) 的情况下的计算过程。在这种情况下传输的 9 个比特是 101110011。你应该自行检查一下这些计算，并核对一下 $D * 2^k = nG \text{ XOR } R$ 的确成立。

需要:

$$D \cdot 2^r \text{ XOR } R = nG$$

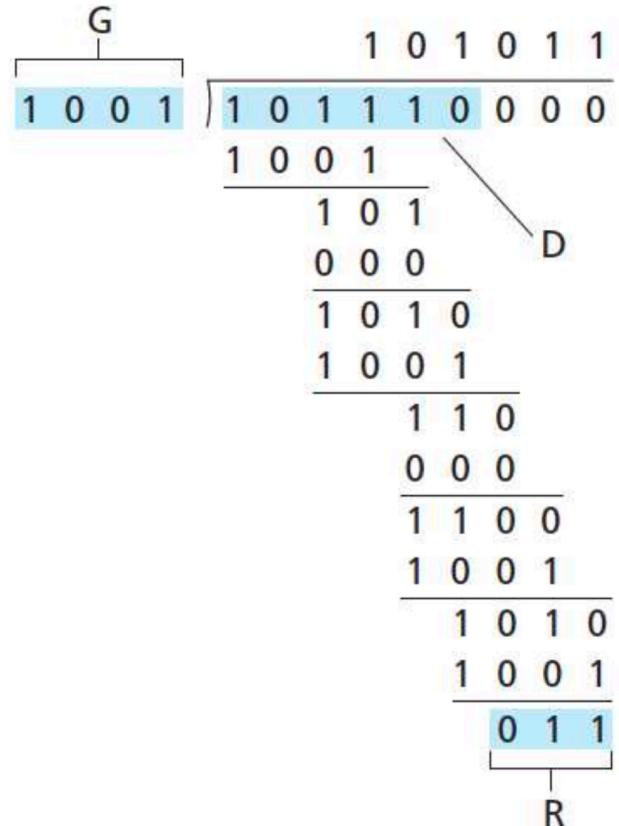
等价于:

$$D \cdot 2^r = nG \text{ XOR } R$$

等价于:

两边同除 G

得到余数 $R=..$



$$R = \text{remainder} \left[\frac{D \cdot 2^r}{G} \right]$$

6.3 multiple access protocols

- 分布式算法 -> 决定把这个 共享式的channel 分配给哪个 node 来使用, 即: 决定节点什么时候可以发送?
- 关于共享控制的通信必须用借助信道本身传输
 - 没有带外的信道, 各节点使用其协调信道使用
 - 用于传输控制信息
- 节点通过这些协议来规范它们在共享的广播信道上的传输行为
- 两种类型的链路: 点对点, 广播 broadcast (共享线路或媒体)
- 2个或更多站点同时传送: 冲突(collision), 多个节点在同一个时刻发送, 则会收到2个或多个信号叠加
- 所以当多个节点处于活跃状态时, 为了确保广播信道执行有用的工作, 以某种方式协调活跃节点的传输是必要的。这种协调工作由多路访问协议负责。

理想的 multiple access protocol:

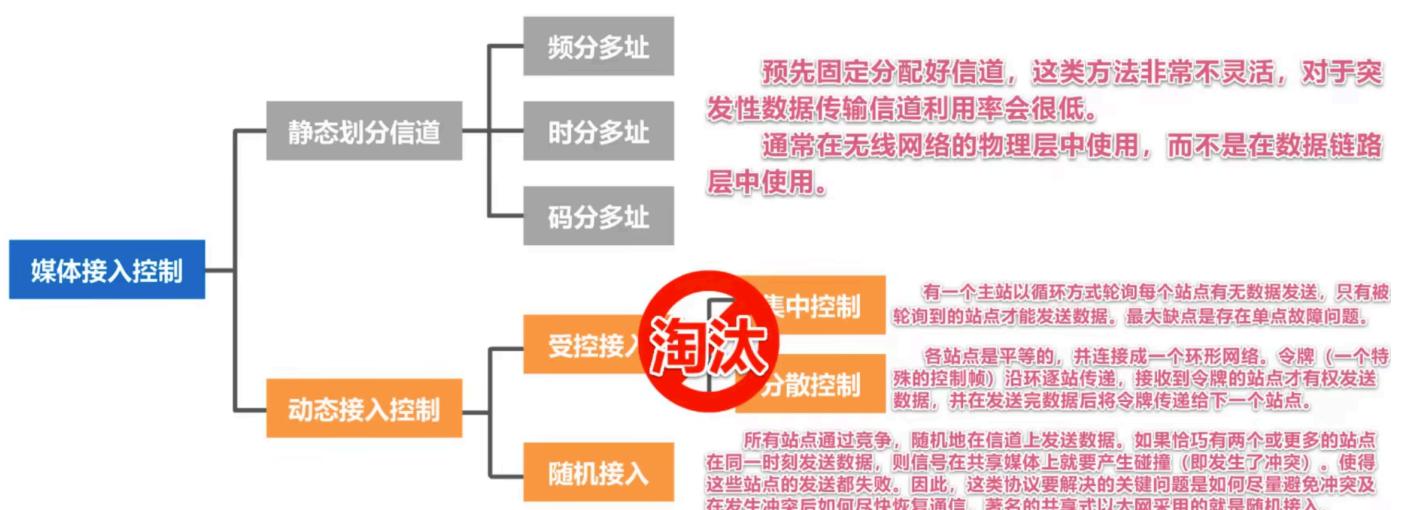
给定: multiple access channel (MAC) of rate R bps, 必要条件:

- 当仅有一个节点发送数据时, 该节点具有 R bps 的吞吐量
- 当有 M 个节点发送数据时, 每个可以以 R/M 的平均速率发送
- 协议是分散的; 这就是说不会因某主节点故障而使整个系统崩溃
- 协议是简单的, 使实现不昂贵

6.3.1 MAC protocols

three broad classes:

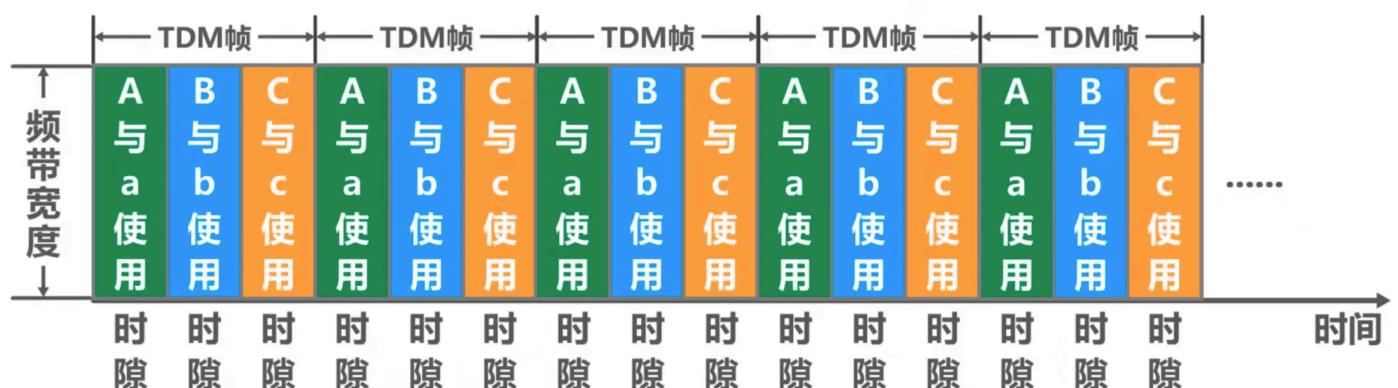
- **channel partitioning 信道划分**
 - 把 channel 划分成更小的 "pieces" (time slots, frequency, code)
 - 分配 pieces 给每个节点专用
- **random access 随机访问，想用的时候就用**
 - channel 不划分，允许 collisions
 - 冲突后恢复
- **taking turns 依次轮流**
 - 节点依次轮流，但是有很多数据传输的节点可以获得较长的信道使用权



6.3.2 channel partitioning

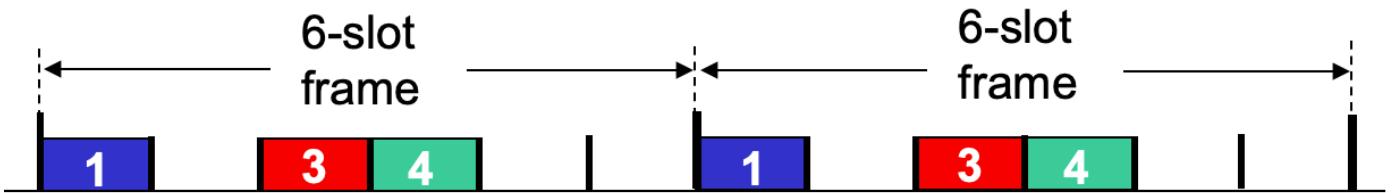
Channel partitioning MAC protocols: TDMA

将时间划分为时间帧 (time frame), 并进一步划分每个时间帧为 N 个时隙 (slot)



- 轮流使用信道，信道的时间分为周期
- 每个站点(station) 使用每周期中固定的时隙 (length = packet transmission time) 传输帧
- 如果站点无帧传输，时隙空闲 -> 浪费

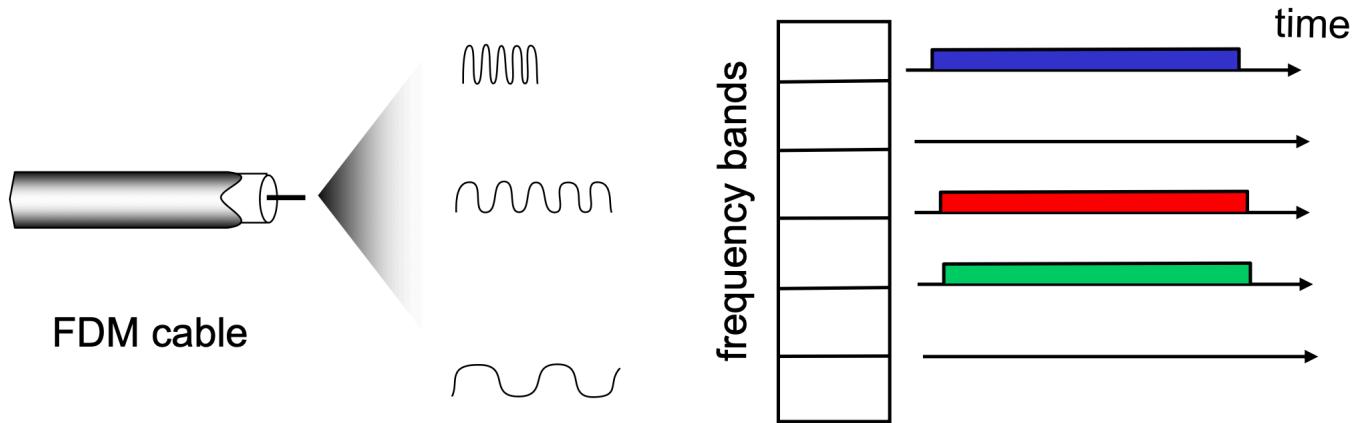
下面 6-station LAN, 1、3、4 have packets to send, slot 2、5、6 空闲



Channel partitioning MAC protocols: FDMA

- 信道的有效频率范围被分成一个个小的频段
- 每个站点被分配一个固定的频段
- 分配给站点的频段如果没有被使用，则空闲

下面 6-station LA, 1、3、4 have packets to send, frequency bands (频段) 2、5、6 idle (空闲)



6.3.3 Random access protocols

- 当节点有帧要发送时
 - 以信道带宽的全部 **R bps** 发送
 - 没有节点间的预先协调
- 两个或更多 node 同时传输 → 可能有 collision, 涉及碰撞的每个节点将反复地重发它的帧 (packet), 直到该帧无碰撞地通过为止

随机存取协议 (random access MAC protocol) 规定:

- 如何检测冲突
- 如何从冲突中恢复(如:通过稍后的重传)

examples of random access MAC protocols:

- unslotted ALOHA, slotted ALOHA
- CSMA, CSMA/CD, CSMA/CA

Slotted ALOHA

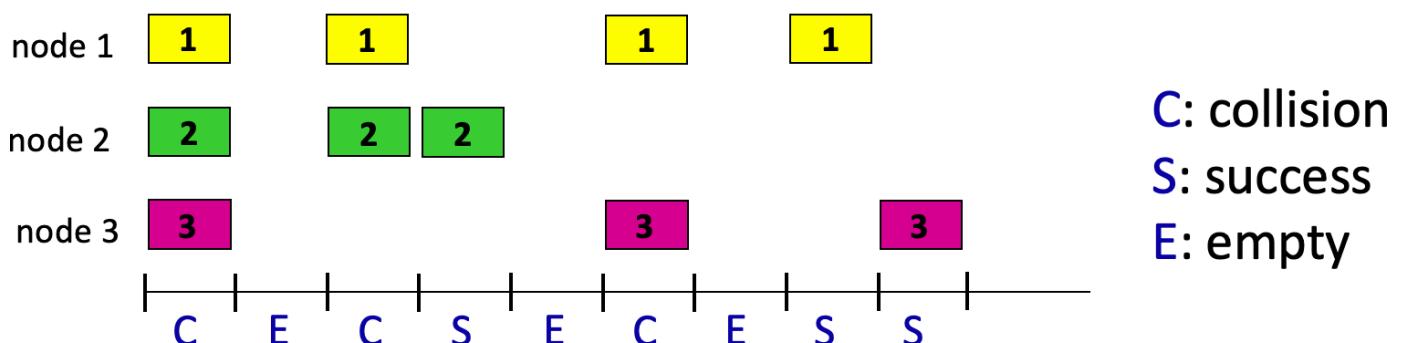
我们以最简单的随机接入协议之一：时隙 ALOHA (Slotted ALOHA) 协议，开始我们对 随机接入协议 (Random access protocols) 的学习

假设

- 所有帧是等长的
- 时间被划分成相等的时隙，每个时隙可发送一帧
- 节点只在时隙开始时发送帧
- 节点在时钟上是同步的
- 如果两个或多个节点在一个时隙传输，所有的站点都能检测到冲突

运行

- 当节点获取新的帧，在下一个时隙传输
- 传输时没有检测到冲突，成功
 - 节点能够在下一时隙发送新帧
- 检测时如果检测到冲突，失败
 - 节点在每一个随后的时隙以概率 p 重传帧直到成功



优点

- 节点可以以信道带宽全速连续传输
- 高度分布：仅需要节点之间在时隙上的同步
- 简单

缺点

- 存在冲突，浪费时隙
- 即使有帧要发送，仍然有可能存在空闲的时隙
- 节点检测冲突的时间<帧传输的时间
 - 必须传完
- 需要时钟上同步

效率: 当有很多节点，每个节点有很多帧要发送时， $x\%$ 的时隙是成功传输帧的时隙

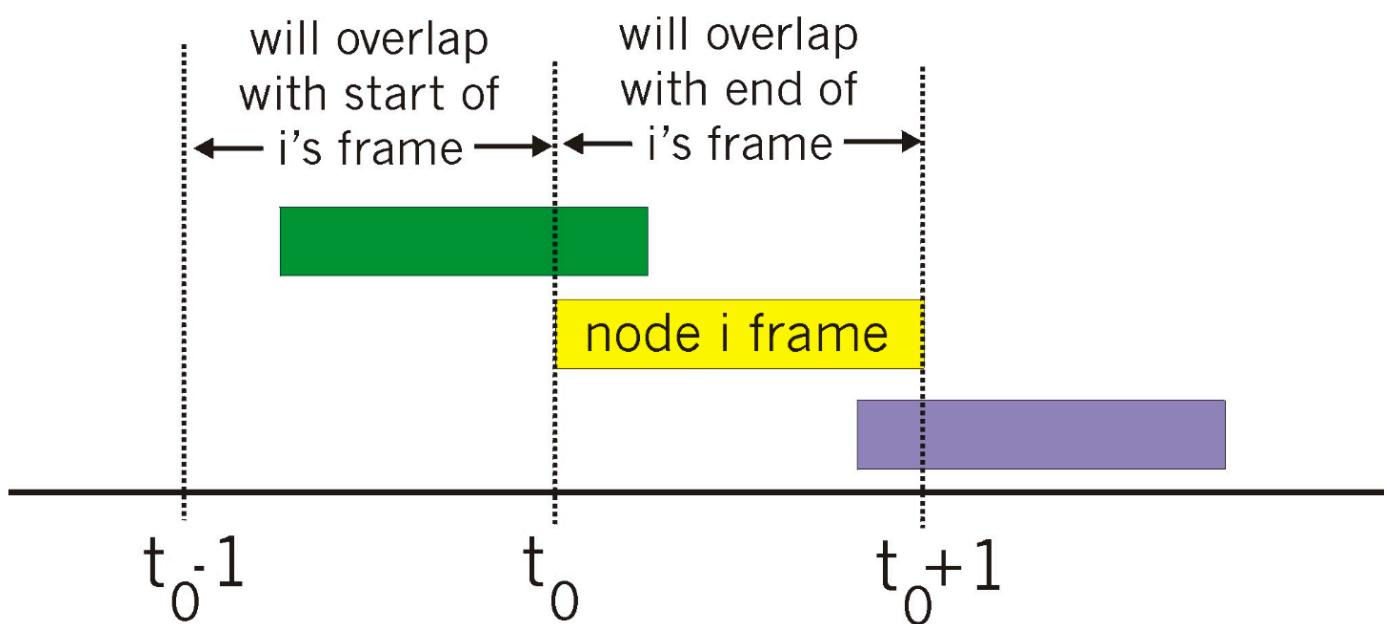
- 假设N个节点，每个节点都有很多帧要发送，在每个时隙中的传输概率是p
- 一个节点成功传输概率是 $p(1-p)^{N-1}$
- 任何一个节点的成功概率是 $= Np(1-p)^{N-1}$

- N个节点的最大效率：求出使 $f(P)=Np(1-p)^{N-1}$ 最大的 p^*
- 代入 P^* 得到最大 $f(p^*)=Np^*(1-p^*)^{N-1}$
- N为无穷大时的极限为 $1/e=0.37$

最好情况：信道利用率37%

Pure (unslotted) ALOHA 效率比时隙ALOHA更差！

- 简单、无须节点间在时间上同步，每个node各发各的不管别人
- 当有帧需要传输：马上传输
- 冲突的概率增加：
 - 帧在 t_0 发送，和其它在 $[t_0 - 1, t_0 + 1]$ 区间内开始发送的帧冲突
 - 和当前帧冲突的区间(其他帧在此区间开始传输)增大了一倍



6.3.4 CSMA (carrier sense multiple access) 礼貌的对话人

CSMA (载波侦听多路访问): **listen before transmit**, 在传输前先侦听信道, 不打断别人正在进行的说话:

- 如果侦听到信道 idle 空闲, 传送整个帧
- 如果侦听到信道 busy 忙, 推迟传送

但是冲突仍然可能发生: 由 propagation delay 造成: 两个节点可能侦听不到正在进行的传输

冲突: 整个冲突帧的传输时间都被浪费了, 是无效的传输(红黄区域)

传播延迟(距离)决定了冲突的概率

CSMA/CD (collision detection)

6.4 LANs

- addressing, ARP and

RARP

- Ethernet • switches

6.5 a day in the life of a web request