The outline and scheme of 《Computer Network》

- Chapter 1 Computer Networks and the Internet
- Chapter 2 Application Layer [应用层]
- Chapter 3 Transport Layer [传输层]
- Chapter 4 Network Layer [网络层]
- Chapter 5 Link Layer: Links, Access Networks, and LANs [链路层]
- Chapter 6 Wireless and Mobile Networks [无线网络]

Chapter 1: roadmap

Questions

1. **什么是internet?**

A.两个描述方式

a. 描述因特网的具体构成，即构成因特网的基本硬件和软件组件

主机(端系统)、通信链路、分组交换机(路由器—用于网络核心、链路层交换机—用于接入网)、因特网服务提供商(ISP)、协议

b.为分布式应用提供服务的联网基础设施：

因特网为应用程序的平台，其API规定了运行在两个端系统上的软件交付数据的方式

B.网络协议

协议定义了在两个或多个通信实体之间交换的报文格式和次序，以及报文发送和/或接收一条报文或其他事件所采取的动作。

掌握计算机网络领域知识的过程就是理解网络协议的构成、原理和工作方式的过程。

2. **什么是 protocol?**

Answer:

- 一个协议定义了在两个或多个通信实体之间交换的报文格式和次序，以及报文发送和/或接收一条报文或其他事件所采取的动作。
- > A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.

3. **ADSL 和 HFC 的异同点? (DSL 和 HFC 的异同点?)**

Answer:

同:

1. 主干线路都是光纤

异:

1. ADSL用户接入仍使用电话线,HFC 用户接入使用同轴电缆
2. ADSL 上行信道256Kbps, 下行信道 1Mbps; HFC 带宽取决于用户的数量

4. **分组交换和电路交换的异同点?**

5. **四种时延是什么?**

6. **5层模型**

Answer

1. 应用层
2. 传输层
3. 网络层
4. 链路层
5. 物理层

7. **OSI 7层模型**

Answer

1. 应用层
2. 表示层
3. 会话层
4. 运输层
5. 网络层

6. 链路层
7. 物理层

1.1 what *is* the Internet?

1.6

What's the Internet: "nuts and bolts" view

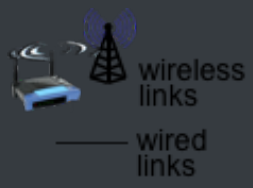
> 在本书中，我们使用一种特定的计算机网络，即公共因特网，作为讨论计算机网络及 其协议的主要载体：但什么是因特网？回答这个问题有两种方式：其一，我们能够描述因特网的具体构成，即构成因特网的基本硬件和软件组件；其二，我们能够根据为分布式应用 提 供 服 务 的 联 网 基 础 设 施 来 描 述 因 特 网 我 们 先 从 描 述 因 特 网 的 具 体 构 成 开 始 ， 并 用图1-1举例说明我们的讨论、

1. millions of connected computing devices:

- - *hosts[主机] = end systems[端系统]*
  - running *network apps*

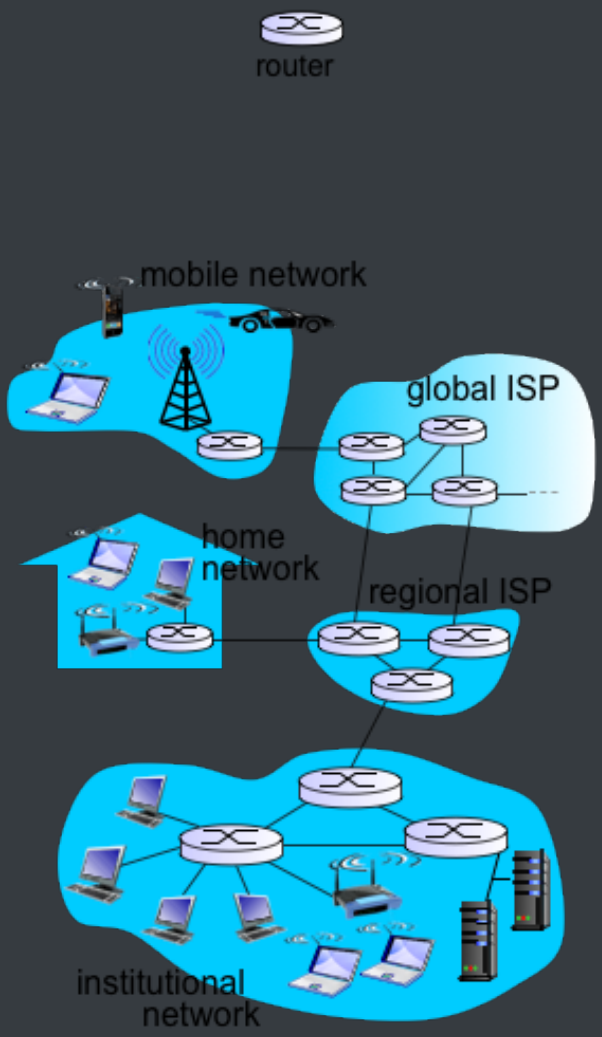
> 因特网是一个世界范围的计算机网络，即它是一个互联了遍及全世界的数以亿计的计算设备的网络 在不久前，这些计算设备多数是传统的桌面PC、Linux工作站以及所谓的服务 器（它们用于存储和 传输Web页面和电子邮件报文等信息）。然而， 越来越多的非 传统的因特网端系统 （如便携机、智能手机 、平板电脑 、电视、游戏机、Weh相机、汽年 、环境传感设备、数字相框、家用电器）和安全系统，正在与因特网 相连。的确 ， 在许多非传统设 备连接到因特网的情况下，计算机网络这个术语开始听起来有些过时了，用因特网术语來说.所有这些设备都称为 主机（ host ）或端系统（ end system ） ,到2 0 1 1年7月为止，有大约5亿 台端系 统与因特网连接，这并未将 智能手机 、便 携机 和仅断断 续续 与因特网 连接的设备计算在内［ISC2011］。总体来说，因特网用户数估计超过20亿［HTI 2011］。

1. *communication links*
2. 2 fiber, copper, radio, satellite
3. 3 transmission rate: *bandwidth*



> 端系统（ end system ）通过通信链路（communication link）和分组交换机（packet switch）连接到一起。在1 . 2节中，我们将介绍许多类型的通信链路，它们由不 同类型的物理媒体组成。这些物理媒体包括同轴电缆、铜线、光纤和无线电频谱。不同的链路能够以不同的速率传输数据，链路的传输速率以比特/秒度量（ｂｉｔ/ｓ，或ｂｐｓ）。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分组（ｐａｃｋｅｔ）。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。

2. *Packet switches:* forward packets (chunks of data)
3. 1 *routers* and *switches*



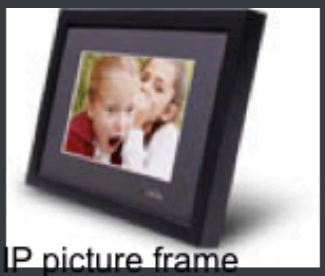
分组交换机(packet switch）从它的一条人通信链路接收到达的分组，并从它的一条出通信链路转发该分组。市面上流行着各种类型、各具特色的分组交换机，但在当今的因特网中，两种最著名的类 型是 路由 器（rou ter）和链路层交换机（link - laye r s witch）。这两种类 型的交换机 朝着最终目的地转发 分组。链路层交换机通常用于接人网中，而路由器通常用于网络核心中。从发送端系统到接收端系统，一个分组所经历的一系列通信链路和分组交换机称为通过该网络的路径（route或path）。因特网所承载的精确通信量是难以估算的，不过思科公 司［Cisco VNI 2011］估计，全球因特网流量在2012年每月大约为40EB （10"字节）。

用于传送分组的分组交换网络在许多方面类似于承载运输车辆的运输网络，该网络包 括了高速公路、公路和立交桥。例如，考虑下列情况，一个工厂需要将大量货物搬运到数千公里以外的某个目的地仓库。在工厂中，货物要分开并装上卡车车队。然后，每辆卡车独立地通过高速公路、公路和立交桥组成的网络向该仓库运送货物。在目的地仓库，卸下 这些货物，并且与一起装载的同一批货物的其余部分堆放在一起。因此，在许多方面，分 组类似于卡车，通信链路类似于高速公路和公路，分组交换机类似于立交桥，而端系统类似于建筑物。就像卡车选取运输网络的一条路径前

> 行一样，分组则选取计算机网络的一条路径前行。
>
> 端系统通过因特网服务提供商（Internet Service Provider, ISP）接人因特网，包括如本地电缆或电话公 司那样的住宅区ISP、公 司I S P、大学I S P，以及那些在机场、旅馆、咖啡店和 其 他 公 共场所提供W i F i接人的I S P。每个I S P是一个由多个分组交换机和多段通信 链路组成的网络。 各I S P为端系统提供了各种不同类型的网络接入，包括如线缆调制解调器或D S L那样的住宅宽带接人 、高速局域网接人 、无线接人 和5 6 k b p s拨号调制解调器接人 。I S P也 为 内容提供者提供因特网接人服务，将W e b站点直接接人因特网。因特网就是将端系统彼此互联，因此为端系统提供接人的I S P也必须互联。 低层的I S P通过国家的、 国际的髙层 IS P （如 Le ve l 3 Co mm un ic at io ns、AT &T、Sp ri nt 和 NT T）互联起来。高层 ISP 是由通过高速光纤链路互联的高速路由器组成的。无论是高层还是低层I S P网络，它们每个都是独立管理的，运行着IP协议（详情见后），遵从一定的命名和地址习惯。我们将在1.3节中更为详细地考察ISP及其互联的情况。

1.7

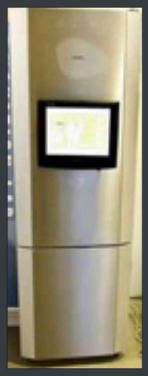
“Fun” internet appliances



IP picture frame
http://www.ceiva.com/

Web-enabled toaster +
weather forecaster

Tweet-a-watt:
monitor energy use



Internet
refrigerator

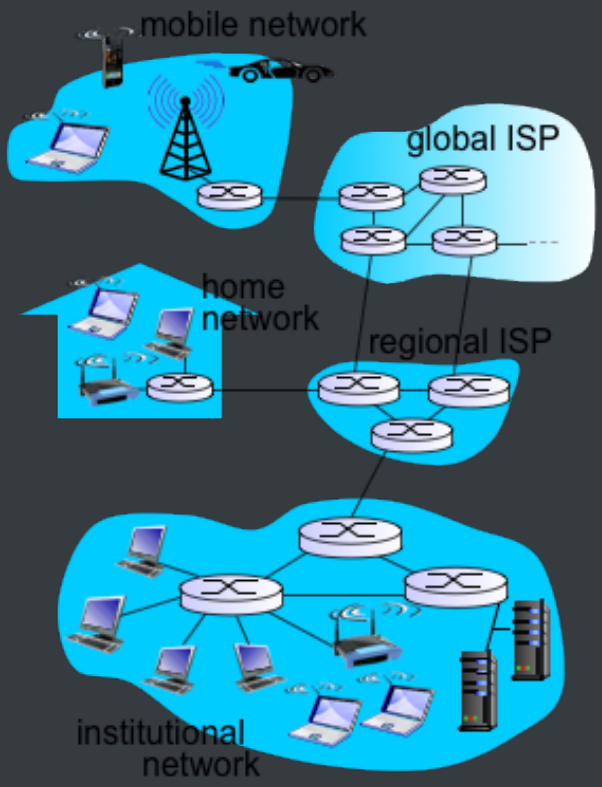
Slingbox: watch,
control cable TV remotely

Internet phones

1.8

What's the Internet: "nuts and bolts" view

1. *Internet:* "network of networks"
2.
    - Interconnected[相互连接的🔗] ISPs
3. *protocols* control sending, receiving of msgs
4.
    - e.g., TCP, IP, HTTP, Skype, 802.11
5. *Internet standards*
6.
    - RFC: Request for comments
    - IETF: Internet Engineering Task Force



端系统、 分组交换机和 其 他因特网部件都要运行一系列 协议（p r o t o c o l ），这些协议控制因特网中信息的接收和发送。

TCP（Transmission Control Protocol ,传输控制协议）和I P （I n t e r n e t P r o t o c o l ,网际协议）是因特网中两个最为重要的协议。I P协议定义了在路 由 器 和端 系 统 之 间 发 送 和 接 收 的 分 组格 式。因特 网 的主要协议 统 称为T C P / I P。我们 在这一章中就开始接触这些 协议。 但这仅 仅是个开始，本书的许多地方与计算机网络协议有关。 鉴于因特网协议的重要性，每个人就各个协议及其作用取得一致认识是很重要的，这样 人 们 就 能 够 创 造 协 同 工 作 的 系 统 和 产 品 。 这正 是 标 准 发 挥 作 用 的 地 方 . 因 特 网 标 准 (Internet standard)由因特网工程任务组(Internet Engineering Task Force, IETF) IETF 2 0 1 2 ］研发:

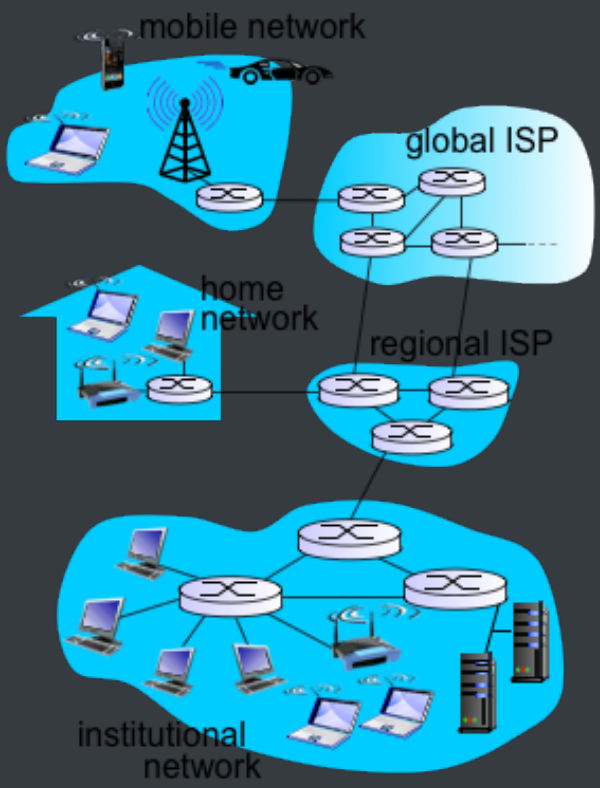
I E T F的标准文档称为请求评论(Request For Comment , R F C) RFC最初是作为 普 通 的 请 求 评 论 (因 此 而 得 名) ， 以 解 决 因 特 网 先 驱 者 们 面 临 的 网络 和 协 议 问 题 ［A ll ma n 20 11 ］ RF C文档 往往 是技 术性 很强 并相当详细的。它们定义了 TC P、IP、HT TP (用 于 W e b) 和 S M T P (用 于 电 子 邮 件) 等协 议 。 目 前 已 经 有 将 近 6 0 0 0 个 R F C • 其 他 组 织 也在制定用于网络组件的标准，最引人注目的是针对网络链路的标准 。 例如，I E E E 8 0 2 LAN/MAN标准化

> 委员会［IEEE 802 202］制定了以太网和无线WiFi的标准

1.9

What’s the Internet: a service view

1. *Infrastructure that provides services to applications:*
2. - Web, VoIP, email, games, e-commerce, social nets, …
3. *provides programming interface to apps*
4. - hooks that allow sending and receiving app programs to “connect” to Internet
   - provides service options, analogous to postal service



前面的讨论已经辨识了构成因特网的许多部件 。 但是我们 也能从 一 个完全 不同的角度，即 从 为应用程序提供服务的基础设施的角度来描述因特网。这些应用程序包括电子邮 件 、 Web冲 浪 、 即 时 讯 息 、 社 交 网 络 、 IP语 音(VoIP) 、 流 式 视 频 、 分 布 式 游 戏、对 等(peer-to-peer,P2P)文 件 共 享 、 因 特 网 电 视 、 远 程 注 册 等 等 。 这 些 应 用 程序称为 分 布式应用程序(distributedapplication),因为它们涉及多台相互 交换数据的端系统。重 要 的是，因特网应用程序运行在端系统上，即它们并不运行在网络核心中的分组交换机中。尽管分组交换机促进端系统之间的数据交换，但它们并不 关心作 为数据的源或宿的应用程序。

我们稍深人地探讨一下为应用程序提供服务的基础设施的含义。为此，假定你对分布式因特网应用程序有一个激动人心的新思想，它可能大大地造福于人类，或者它可能直接使你富有和出名。你将如何将这种思想转换成为一种实际的因特网应用程序呢? 因为应用程序运行在端系统上，所以 你将需要编写运行在端系统上的一 些软件 。 例如，你 可能用 Java、C或 python 编写软件。此时，因为你在研发一种分布式因特网应用程序，运行在不 同端系统上的软件将需要互相发送数据在这里我们碰到一 个核心问题，这导致了 另 一种描述因特网的方法，即将因特网描述为应用程序的平台。运行在一个端系统上的应用程序

怎样才能指令因特网向运行在另一个端系统上的软件发送数据呢? 与因特网相连的端系统提供了一个应用程序编程接口(Ap pl ic at io n Pr og ra mming Interface, API)，该API规定了运行在一个端系统上的软件请求因特网基础设施向运行在另一个端系统上的 特定目 的地软件 交 付数 据的方 式。因特网API是一套发送软件必须遵循的规则集合，因 此 因特 网 能够 将 数 据交 付给 目 的地、我们将在第2章详细讨论因特网API,此时，我们 做 一 个简单的类比，在本书 中我们将经常使用这个类比。假定Alice使用邮政服务向 Bob 发 一 封 信 。 当 然 ， Alice不 能 只 是 写了 这 封 信 (相 关 数 据) 然 后把该信 丢 出窗外。相反，邮政服务要求Alice将信放人 一 个 信封中；在信封的中央写 上Bob的全 名 、地址和邮政编码；封上信封；在信封的右上角贴上邮票；最后将该信封丢进一个邮局的邮政服务邮箱中 。因此，该邮政服务有自己的“邮政服务API”或一套规则，这是Alice必须遵循的，

这样邮政服务才能将自己的信 件 交 付给Bob..同理，因特网也有一个发送数据的程序必须遵循的API,使因特网向接收数据的程序交付数据。 当然，邮政服务向顾客提供了多种服务，如特快专递、挂号、普通服务等。同样的，因特网向应用程序提供 了多种服务当你研发 一种因特网应用程序时，也必须为 你的应用程序选择其中的一种因特网服务。我们将在第2章中描述因特网服务.

1.10

human protocols:

- “what’s the time?”
- “I have a question”
- introductions

… specific msgs sent

… specific actions taken when msgs received, or other events

> 也许理解计算机网络协议概念的一 个最容易办法是，先 与某些 人类活动进行类比，因为我们 人类无时无刻 不在执行协议。考虑当你想要向某人询问时间时将要怎样做 。图1 - 2中显示了 一种典型的 交 互过 程。 人类协议 （ 至少说是 好的 行为方式） 要求一方首先 进 行 问 候 （ 图 1 - 2 中 的 第 一 个 “ 你 好 ” ） ， 以 开 始 与 另 一 个 人 的 通 信 。 对 “ 你 好 ” 的 典型 响 应 是 返 回 一 个 " 你 好 ” 报 文 。 此 人 用 一 个 热 情 的 “ 你 好 ” 进 行 响 应 ， 隐 含 着 一 种指 示 ， 表 明 能 够 继 续 向 那 人 询 问 时 间 了 。 对 最 初 的 “ 你 好 ” 的 不 同 响 应 （ 例 如 “ 不 要烦 我 ！ ”， 或 “ 我 不 会 说 英 语 ”， 或 某 些 不 合 时 宜 的 回 答 ） 也 许 表 明 了 一 个 勉 强的或不能进行的通信 。在此情况 下，按照人类协议，发话者也许将不能够询问时间了 。有时，问的问题根本得不 到 任 何回答，在此情况 下，发话者通常会放弃向这个 人询问时间。注意在我们 人类协议中，有我们 发送的特定报文，也有我们根据接收到的应答报文或其 他 事 件采取的动作（例如在某个给定的时间内没有回答）n显然，发送和接收的报文，以及这些报文发送和接收或其 他 事 件 出现时所采取的动 作，这些在一 个 人类协议中起到 了核心作用。如果人 们 使用不 同的协议（例如，如果一 个 人讲礼貌，而另 一 人 不讲礼貌，或一 个 人明白时间这个概念，而另一 人 却 不知道），该协议就不能互 动，因而不能完成有用 的 工 作 。 在 网 络 中 这 个 道 理 同 样 成 立 。 即 为 了 完 成 一 项 工 作 ， 要 求 两 个 （ 或 多 个 ） 通信实体运行相同的协议。

> 我们再考虑第二个人类类比的例子。假定你正在大学课堂里上课（例如上的是计算机网络课程）。教师正在唠唠叨叨地讲述协议，而你困惑不解。这名教师停下来问：“同学们有什么问题吗？”（教师发送出一个报文，该报文被所有没有睡觉的学生接收到了。）你举起了手（向教师发送了一个隐含的报文），这位教师面带微笑地示意你说：“请讲……”（教师发出的这个报文鼓励你提出问题，教师喜欢被问问题。）接着你就问了问题（即向该教师传输了你的报文）。教师听取了你的问题（即接收了你的问题报文）并加以回答（向你传输了回答报文）。我们再一次看到了报文的发送和接收，以及这些报文发送和接收时所采取的一系列约定俗成的动作，这些是这个“提问与回答”协议的核心。

network protocols:

- machines rather than humans
- all communication activity in Internet governed by protocols

*protocols define format**, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt*

> 网络协议类似于人类协议，除了交换报文和采取动作的实体是某些设备的硬件或软件组件（这些设备可以是计算机、智能手机、平板电脑、路由器或其他具有网络能力的设备）。在因特网中，凡是涉及两个或多个远程通信实体的所有活动都受协议的制约。例如，在两台物理上连接的计算机中，硬件实现的协议控制了在两块网络接口卡间的"线上”的比特流；在端系统中，拥塞控制协议控制了在发送方和接收方之间传输的分组发送的速率。协议在因特网中到处运行，因此本书的大量篇幅与计算机网络协议有关。以大家可能熟悉的一个计算机网络协议为例，考虑当你向一个Web服务器发出请求（即你在Web浏览器中键人一个Web网页的URL）时所发生的情

> 况。图1 - 2右半部分显示 了这种情形。首先，你的计算机将向该Web服务器发送一条连接请求报文，并等待回答。 该 W e b 服 务 器 将 最 终 能 接 收 到 连 接 请 求 报 文 ， 并 返 回 一 条 连 接 响 应 报 文 。 得 知 请 求 该 W e b文档正常以 后，计算机则在一条G E T报文中 发送要从这台W e b服务 器 上 取回的网页 名字。最后，Web服务器向计算机返回该Web网页（文件）

1.11

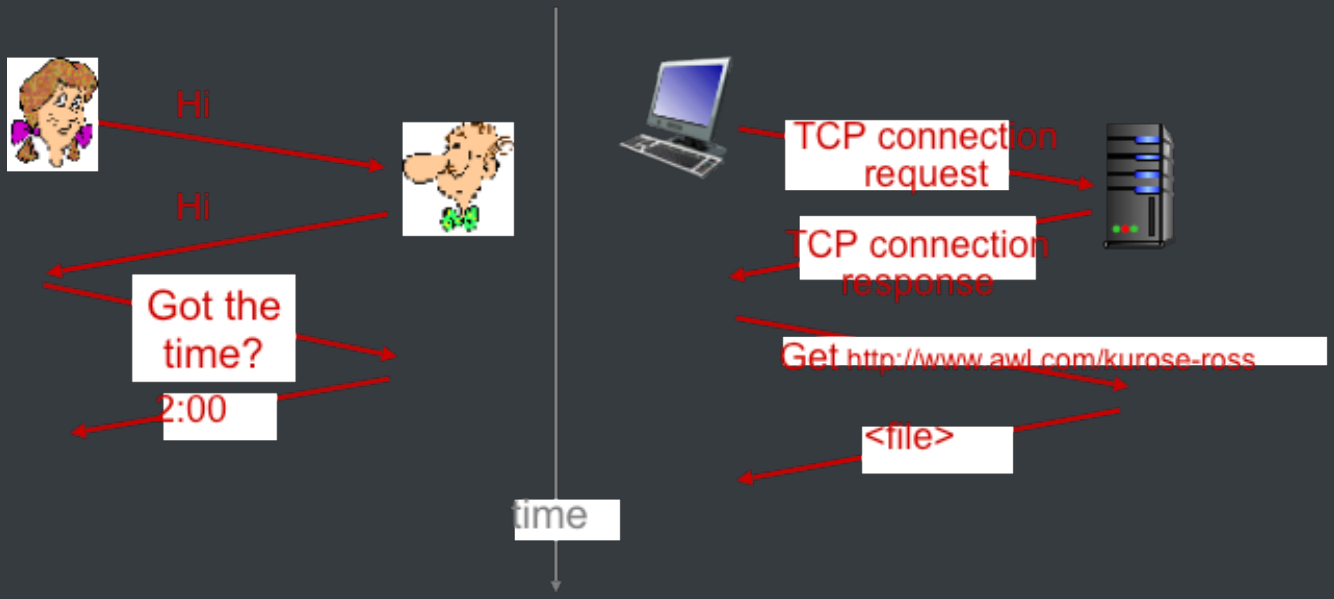
⭐ What's a protocol?

Answer:

- [defines, format and the order exchanged between entities, actions on transmission and receopt] A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the trans-mission and/or receipt of a message or other event.

> 一个协议定义了在两个或多个通信实体之间交换的报文格式和次序，以及报文发送和/或接收一条报文或其他事件所采取的动作。

a human protocol and a computer network protocol:



TCP 三次握手🤝

Q: other human protocols?

1.2 network edge [网络边缘] : end systems, access networks, links

2.*网络边缘**

a.端系统:位于边缘的与因特网相连的计算机和其他设备

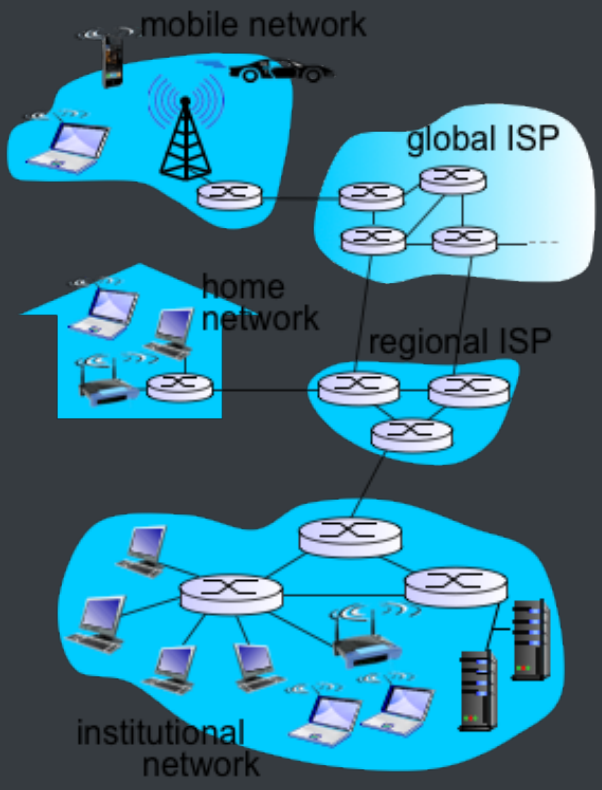
b.接入网:将端系统连接到边缘路由器的物理链路

接入类型	接入技术		物理媒体
家庭	DSL		双绞铜线
	电缆		同轴电缆
	FTTH（光纤到户）		光纤
	拨号		陆地无线电信道
	卫星		卫星无线电信道
企业或家庭	以太网		
	Wifi		
广域无线接入	3G		
	LTE		

1.13

A closer look at network structure

- *network edge:*
- - hosts: clients and servers
  - servers often in data centers
- *access networks, physical media:* wired, wireless communication links
- *network core:*
- - interconnected routers
  - network of networks



1.14

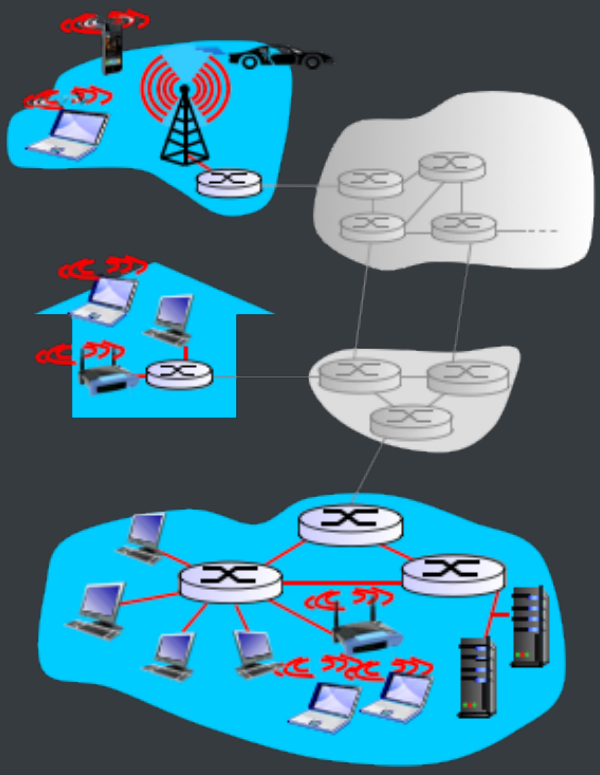
Access networks and physical media

Q: How to connect end systems to edge router?

- residential access nets
- institutional access networks (school, company)
- mobile access networks

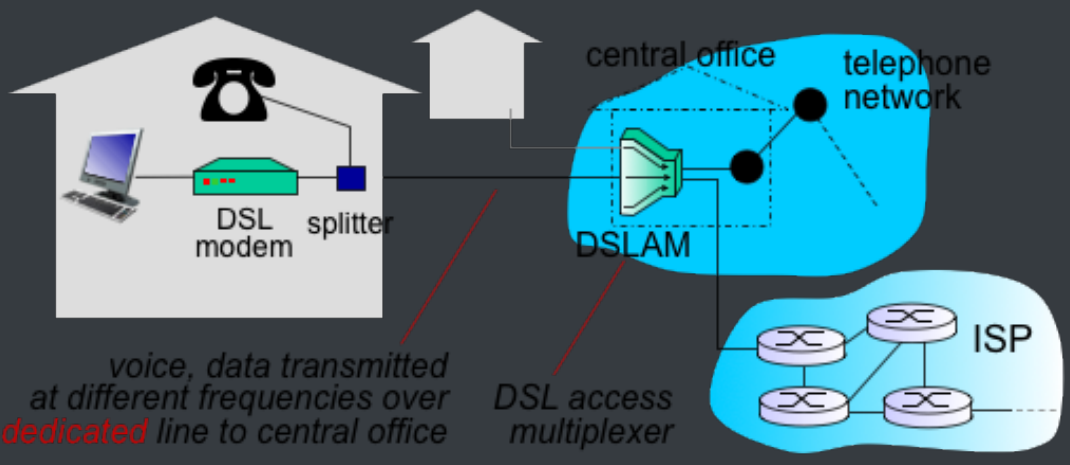
keep in mind:

- bandwidth (bits per second) of access network?
- shared or dedicated?



1.15

Access net: digital subscriber line (DSL)

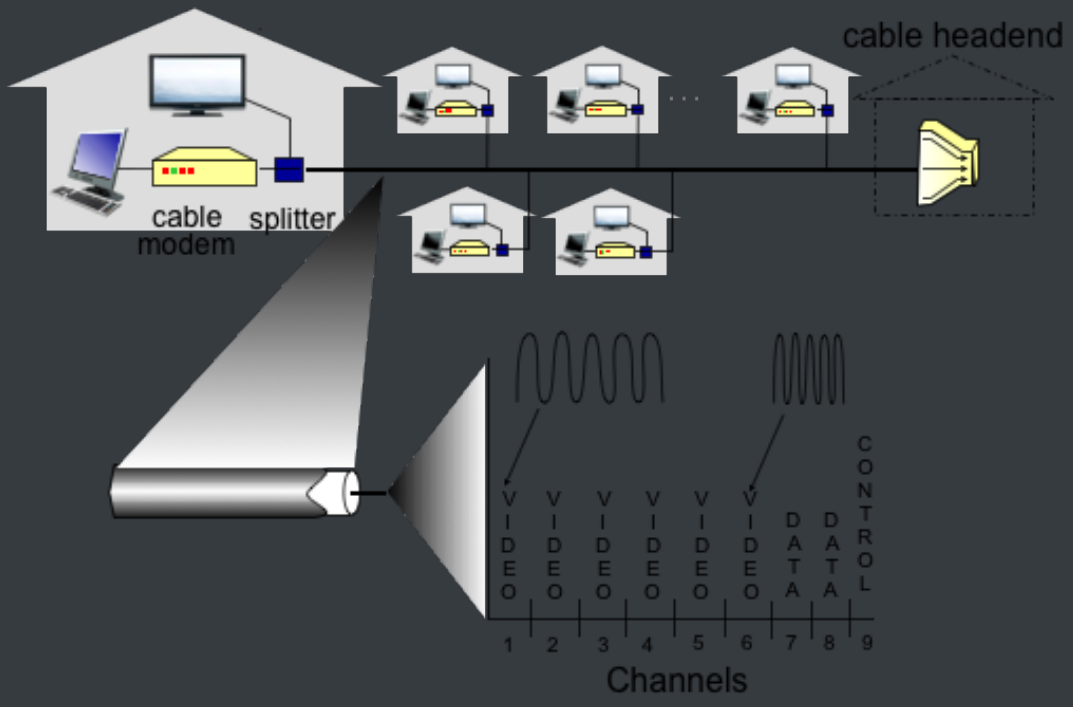


- 1. use *existing* telephone line to central office DSLAM
  2.
     - data over DSL phone line goes to Internet
     - voice over DSL phone line goes to telephone net
  3. < 2.5 Mbps upstream transmission rate (typically < 1 Mbps)

4. < 24 Mbps downstream transmission rate (typically < 10 Mbps)

1.16

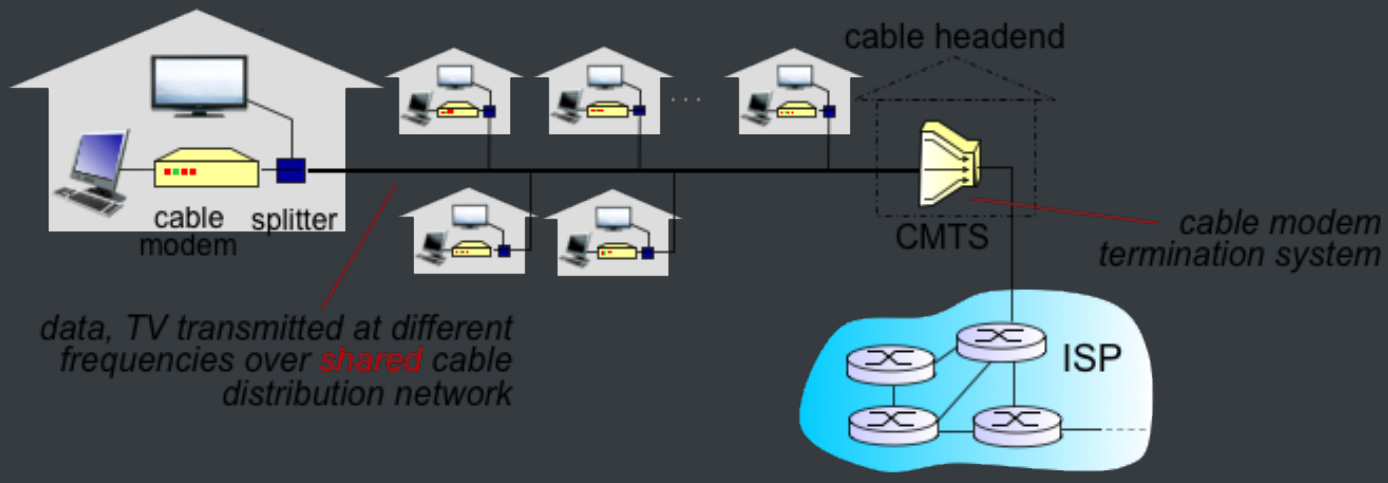
Access net: cable network



frequency division multiplexing: different channels transmitted in different frequency bands

1.17

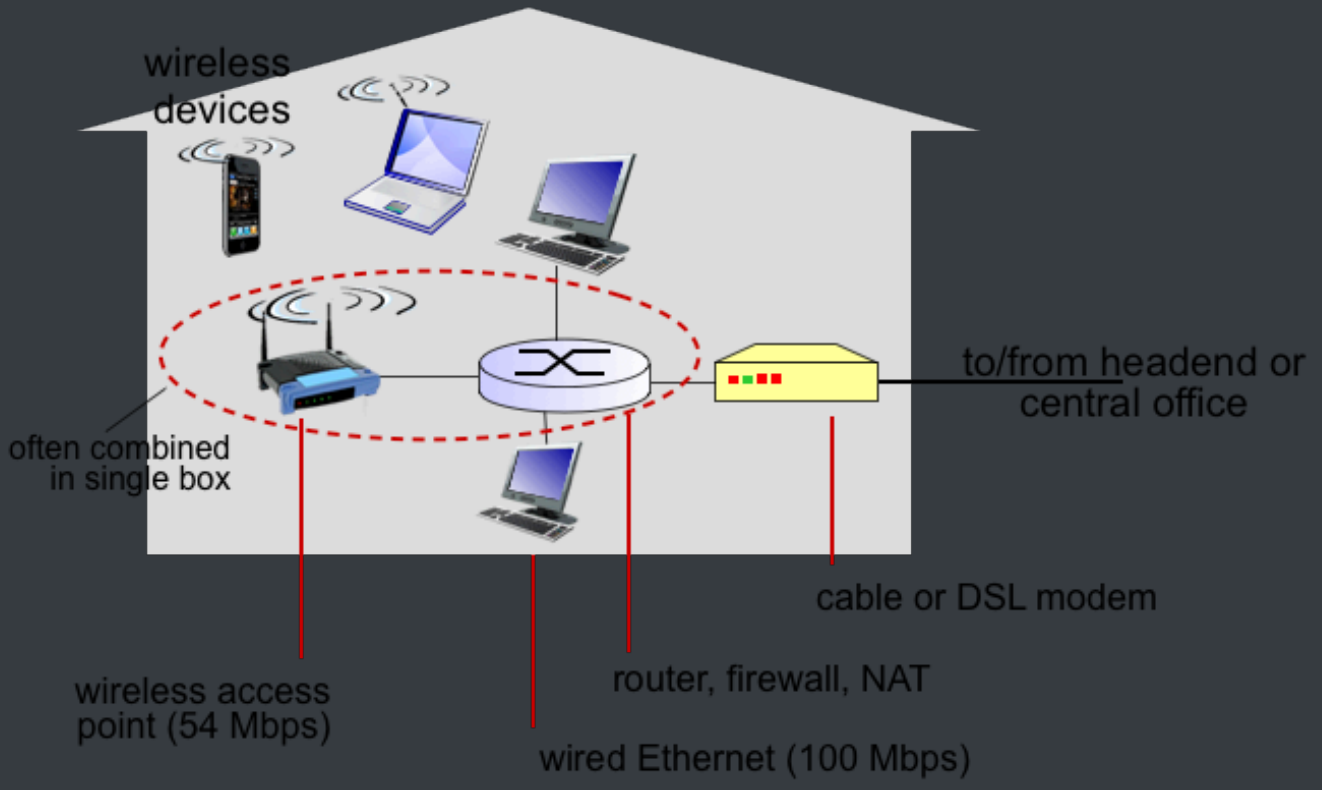
Access net: cable network



- HFC: hybrid fiber coax
- - asymmetric: up to 30Mbps downstream transmission rate, 2 Mbps upstream transmission rate
- network of cable, fiber attaches homes to ISP router
- - homes *share access network* to cable headend
  - unlike DSL, which has dedicated access to central office

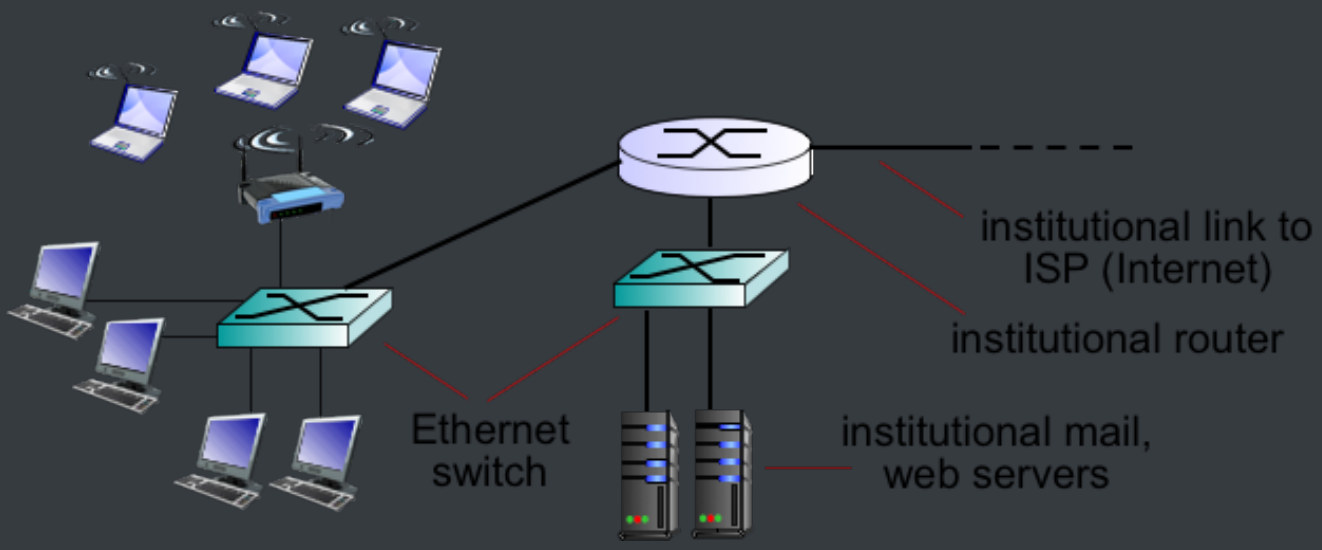
1.18

Access net: home network



1.19

Enterprise access networks (Ethernet)



- typically used in companies, universities, etc
-
  - 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
  - today, end systems typically connect into Ethernet switch

1.20

Wireless access networks

- shared *wireless* access network connects end system to router
- - via base station aka “access point”

wireless LANs:

- - within building (100 ft)
  - 802.11b/g (WiFi): 11, 54 Mbps transmission rate

wide-area wireless access

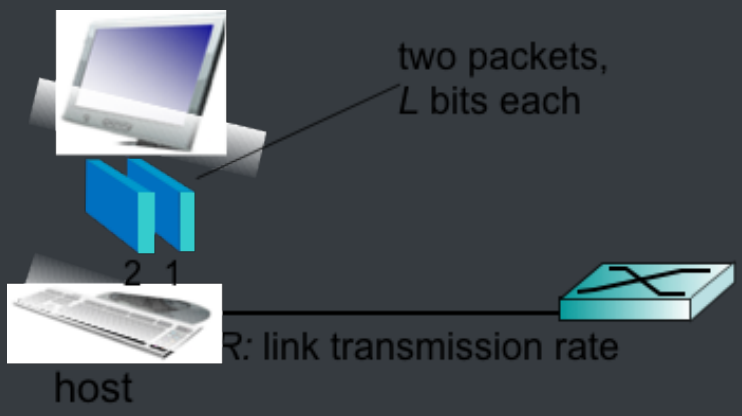
- - provided by telco (cellular) operator, 10’s km
  - between 1 and 10 Mbps
  - 3G, 4G: LTE

1.21

Host: sends *packets* of data

host sending function:

- ❖takes application message
- ❖breaks into smaller chunks, known as *packets*, of length *L* bits
- ❖transmits packet into access network at *transmission rate R*
- - link transmission rate, aka link *capacity, aka link bandwidth*



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

$$
\begin{aligned}
&packet\ transmission\ delay \\
&= time\ needed\ to\ transmit\ L\ bit\ packet\ into\ link \\
&= \frac{L(bits)}{R(bits/sec)}
\end{aligned}
\tag{1}
$$

1.22

Physical media

- bit: propagates between transmitter/receiver pairs
- physical link: what lies between transmitter & receiver
- guided media:
    - signals propagate in solid media: copper, fiber, coax
- unguided media:
    - signals propagate freely, e.g., radio

*twisted pair (TP**)*

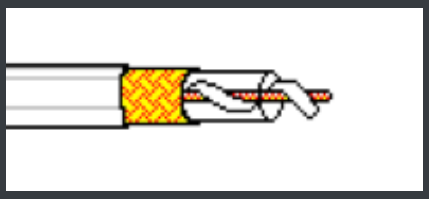
- two insulated copper wires
    - Category 5: 100 Mbps, 1 Gpbs Ethernet
    - Category 6: 10Gbps

1.23

coaxial cable:

- two concentric copper conductors
- bidirectional

- broadband:
- - multiple channels on cable
  - HFC (hybrid fiber coax)



fiber optic cable:

- glass fiber carrying light pulses, each pulse a bit
- high-speed operation:
- - high-speed point-to-point transmission (e.g., 10's-100's Gpbs transmission rate)
- low error rate:
- - repeaters spaced far apart
  - immune to electromagnetic noise

1.24

Physical media: radio

- signal carried in electromagnetic spectrum
- no physical "wire"
- bidirectional

- propagation environment effects:
- - reflection
  - obstruction by objects
  - interference

radio link types:

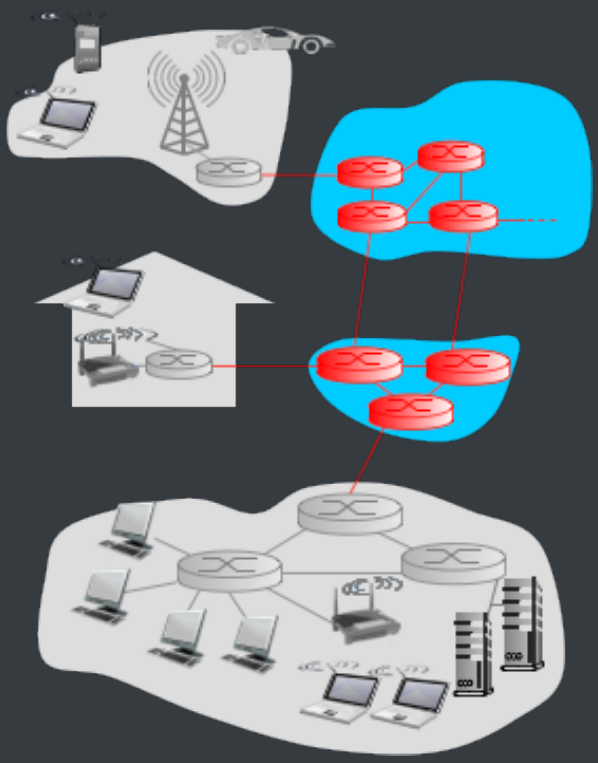
- terrestrial microwave
- - e.g. up to 45 Mbps channels
- LAN (e.g., WiFi)
- - 11Mbps, 54 Mbps
- wide-area (e.g., cellular)
- - 3G cellular: ~ few Mbps
- satellite
- - Kbps to 45Mbps channel (or multiple smaller channels)
  - 270 msec end-end delay
  - geosynchronous versus low altitude

1.3 network core : packet switching, circuit switching, network structure

1.26

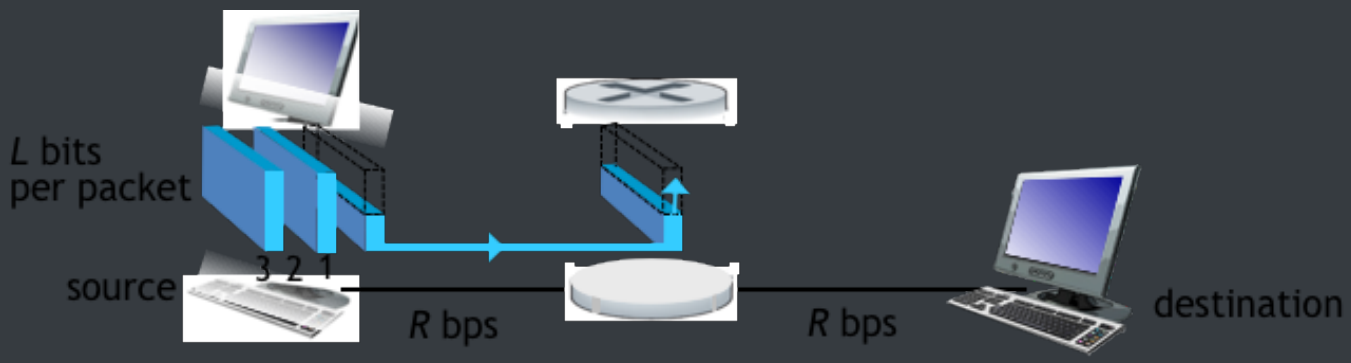
The network core

1. mesh of interconnected routers
2. packet-switching: hosts break application-layer messages into *packets*
3. 1. forward packets from one router to the next, across links on path from source to destination
   2. each packet transmitted at full link capacity



1.27

Packet-switching: store-and-forward



- takes *L*/*R* seconds to transmit (push out) *L*-bit packet into link at *R* bps
- *store and forward:* entire packet must arrive at router before it can be transmitted on next link
- end-end delay = 2*L*/*R* (assuming zero propagation delay)
    - more on delay shortly …

one-hop numerical example:

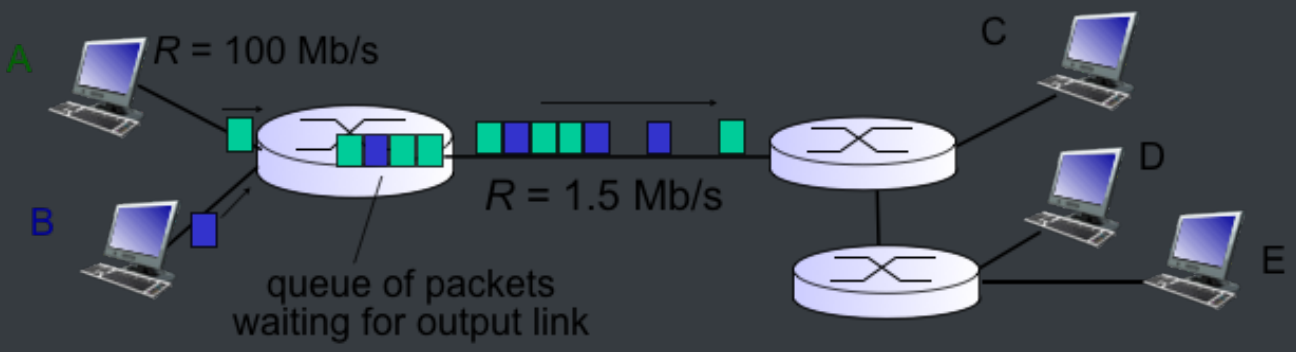
> 传输延迟

- L = 7.5 Mbits
- R = 1.5 Mbps
- one-hop transmission delay = 5 sec

1.28

Packet Switching: queueing delay, loss

> queueing delay[排队延迟]



queuing and loss:

- If arrival rate (in bits) to link exceeds transmission rate of link for a period of time:
-
    - packets will queue, wait to be transmitted on link
    - packets can be dropped (lost) if memory (buffer) fills up

1.29

Two key network-core functions

routing: determines source-destination route taken by packets

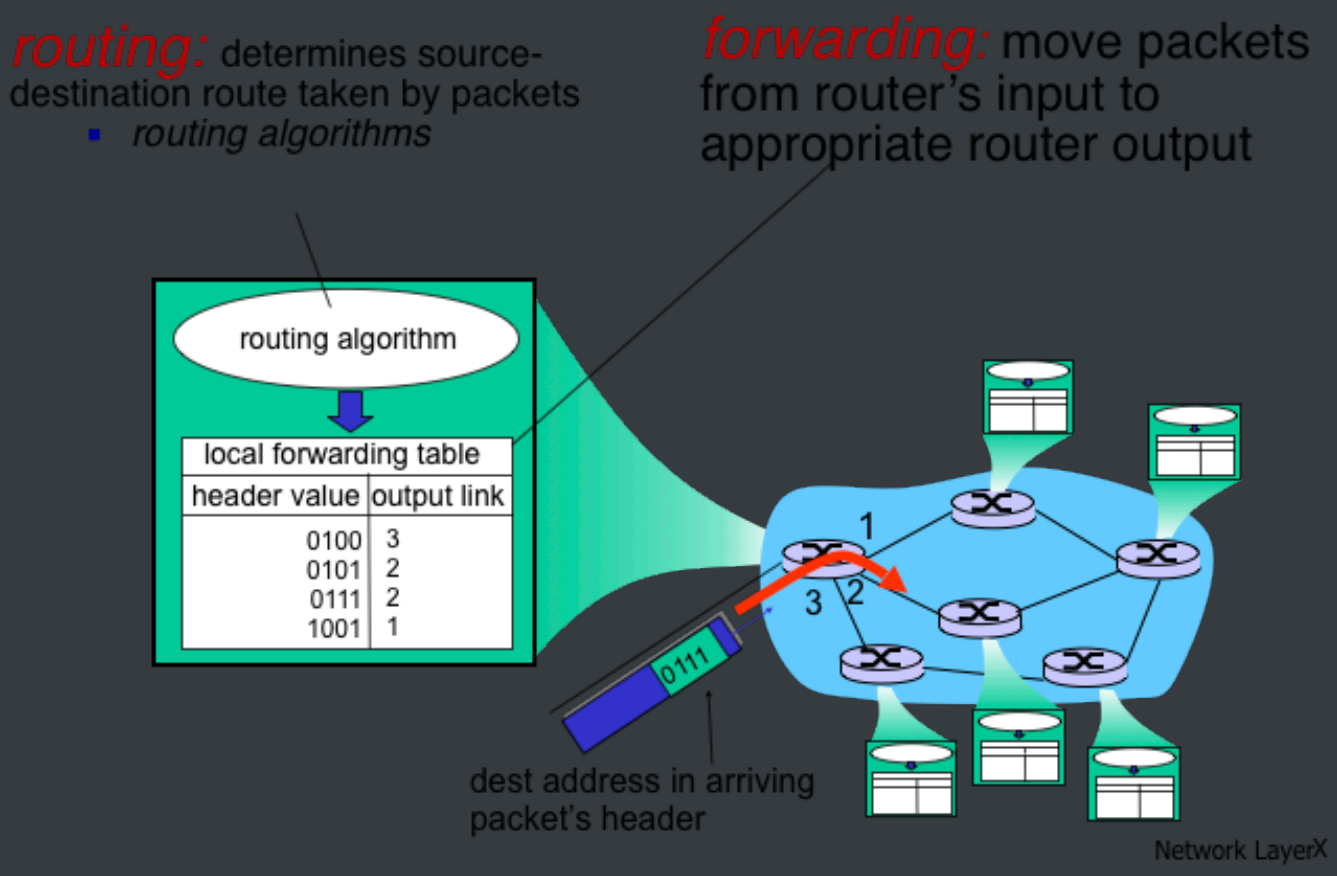
> routing 路由

- *routing algorithms*

> 路由算法
>
> 通常是根据 ip 地址

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

> forwarding 转发



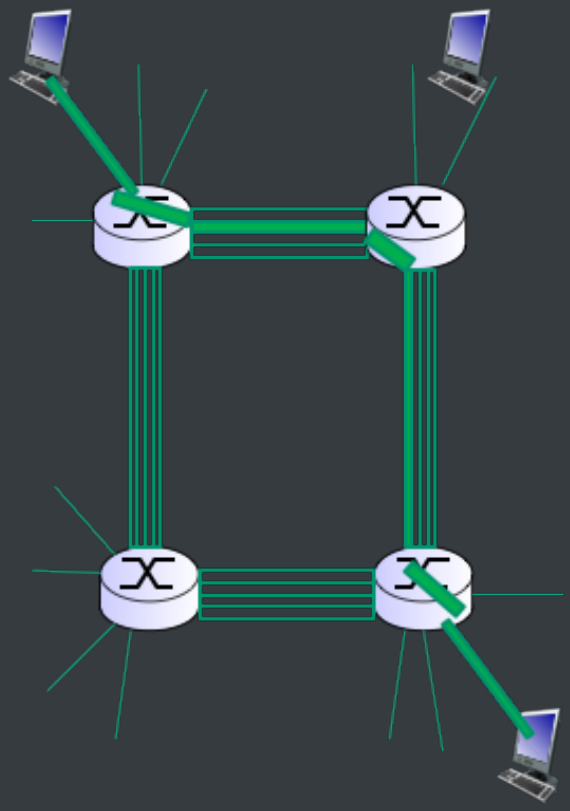
1.30

Alternative core: circuit switching

> 电路交换, 类似电话网络

end-end resources allocated to, reserved for “call” between source & dest:

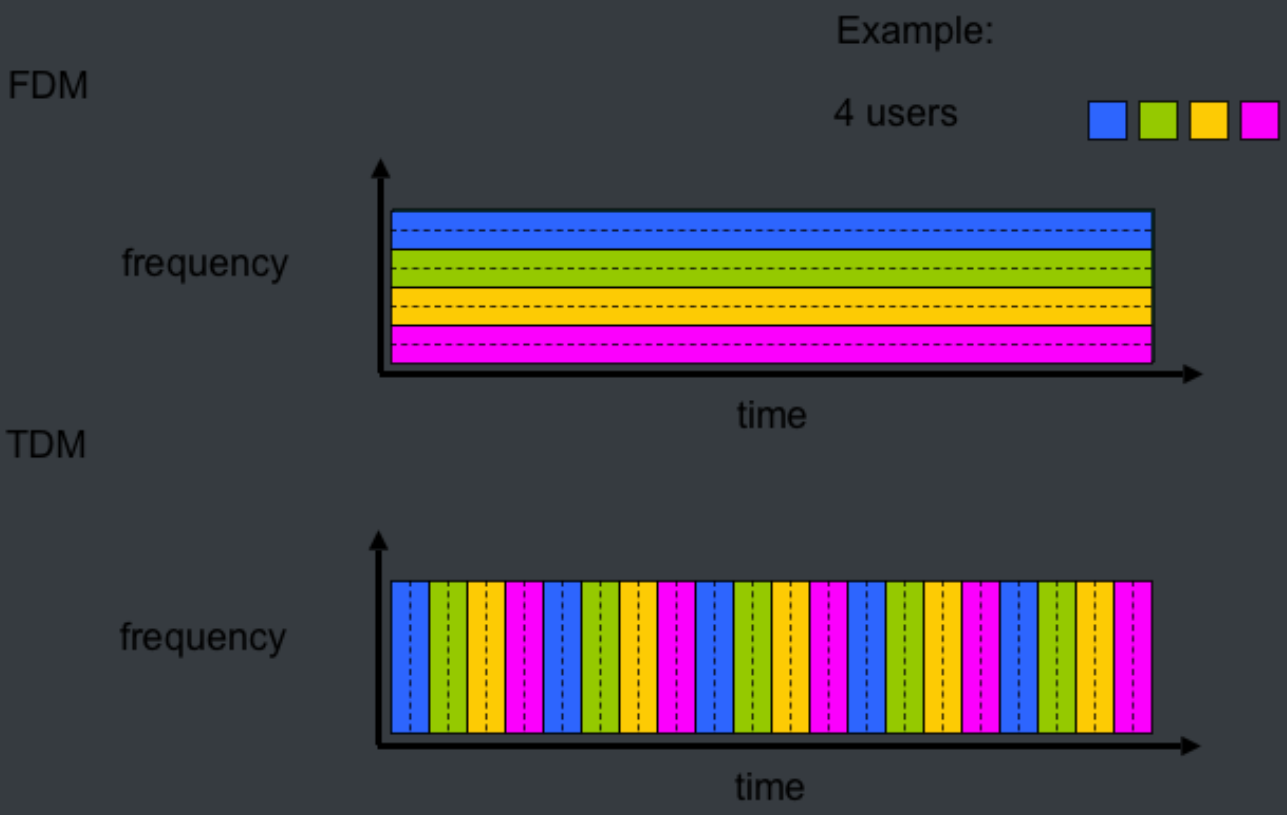
1. In diagram, each link has four circuits.
2. - call gets 2nd circuit in top link and 1st circuit in right link.
3. dedicated resources: no sharing
4. - circuit-like (guaranteed) performance
5. circuit segment idle if not used by call *(no sharing)*
6. Commonly used in traditional telephone networks



1.31

Circuit switching: FDM versus TDM

> 电路交换, 类似电话网络



1.32

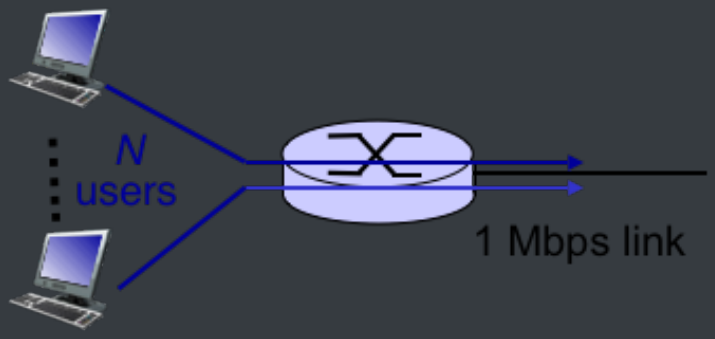
packet switching allows more users to use network!

example:

- 1 Mb/s link
- each user:
- - 100 kb/s when “active”
  - active 10% of time
-
- *circuit-switching:*
- - 10 users
- *packet switching:*
- - with 35 users, probability > 10 active at same time is less than .0004

Q: how did we get value 0.0004?

Q: what happens if > 35 users ?



1.33

Packet switching versus circuit switching

Is packet switching a “slam dunk winner?”

1. great for bursty data
2. - resource sharing
   - simpler, no call setup
3. excessive congestion possible: packet delay and loss
4. - protocols needed for reliable data transfer, congestion control
5. *Q:* How to provide circuit-like behavior?
6. - bandwidth guarantees needed for audio/video apps

- still an unsolved problem (chapter 7)

Q: human analogies of reserved resources (circuit switching) versus on-demand allocation (packet-switching)?

1.34

Internet structure: network of networks

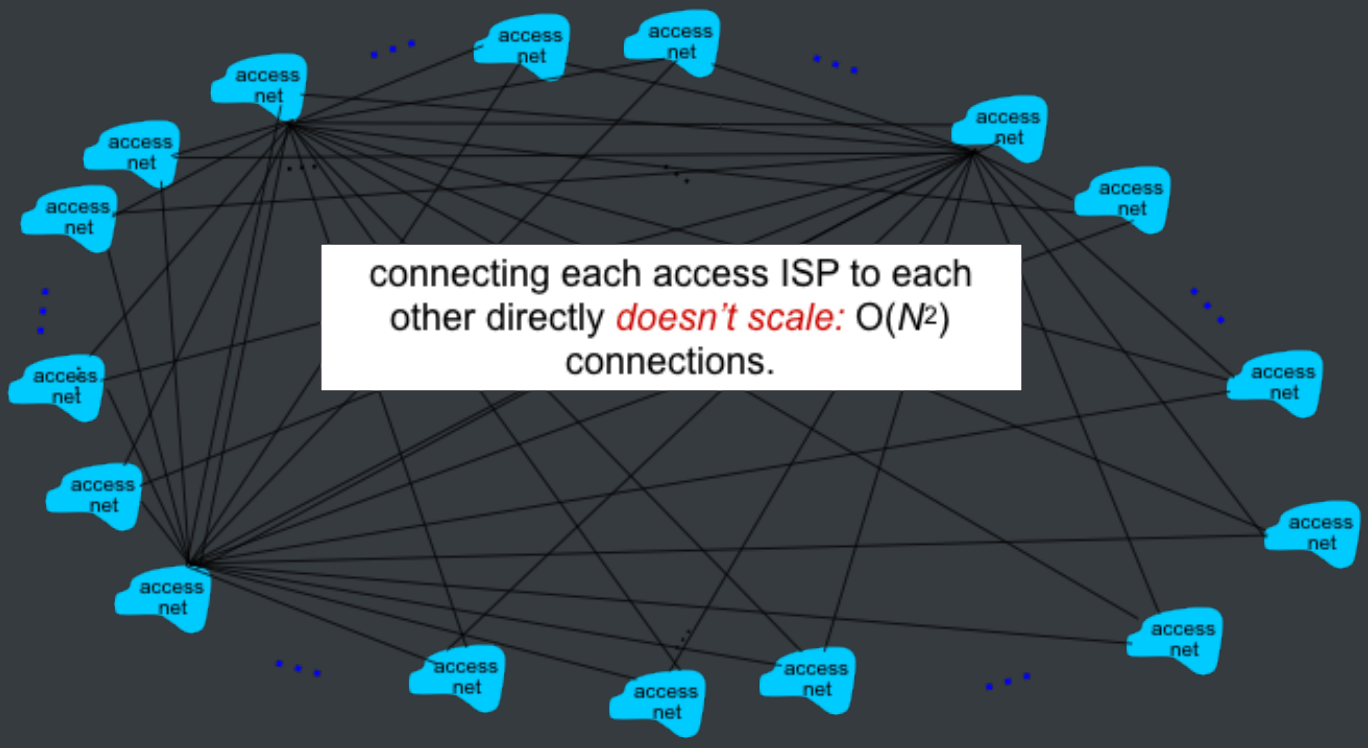
1. End systems connect to Internet via access ISPs (Internet Service Providers)
2. - Residential, company and university ISPs
3. Access ISPs in turn must be interconnected.
4. - So that any two hosts can send packets to each other
5. Resulting network of networks is very complex
6. - Evolution was driven by economics and national policies
7. Let's take a stepwise approach to describe current Internet structure

1.35

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

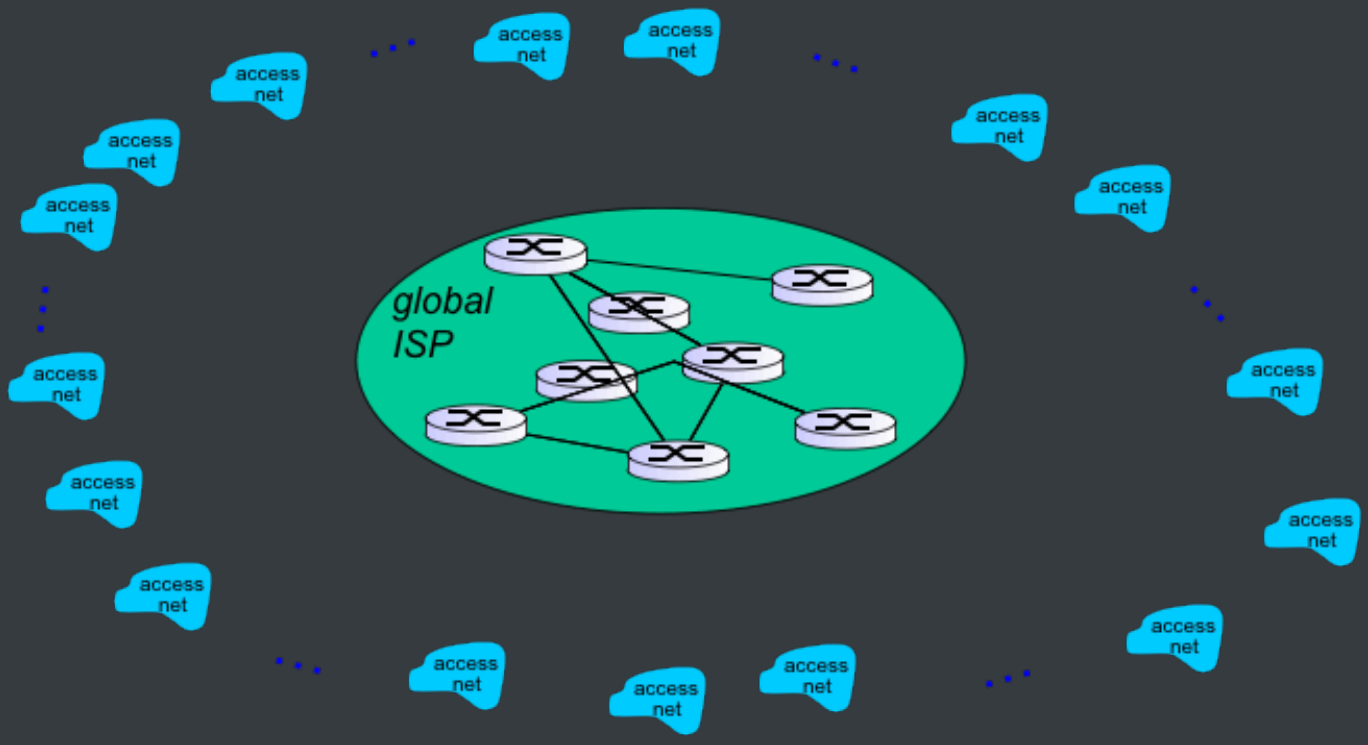
1.36

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



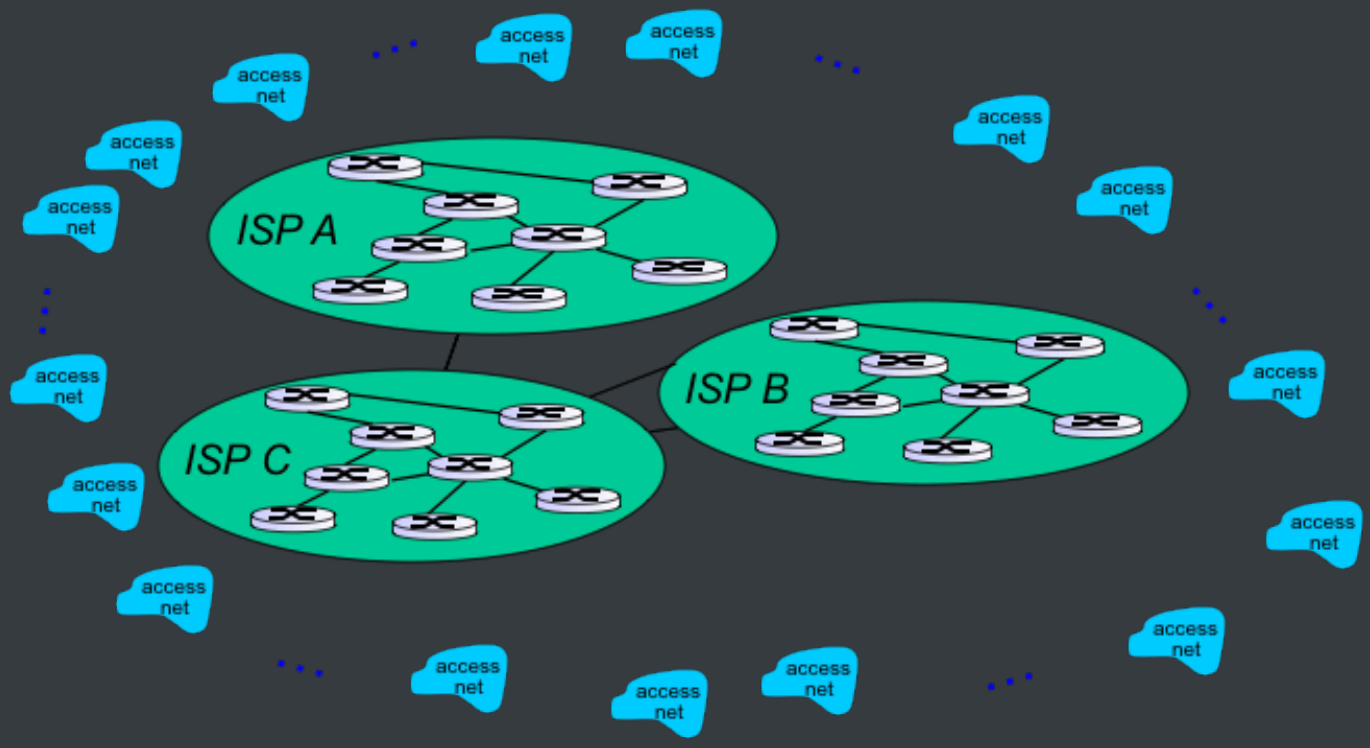
1.37

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



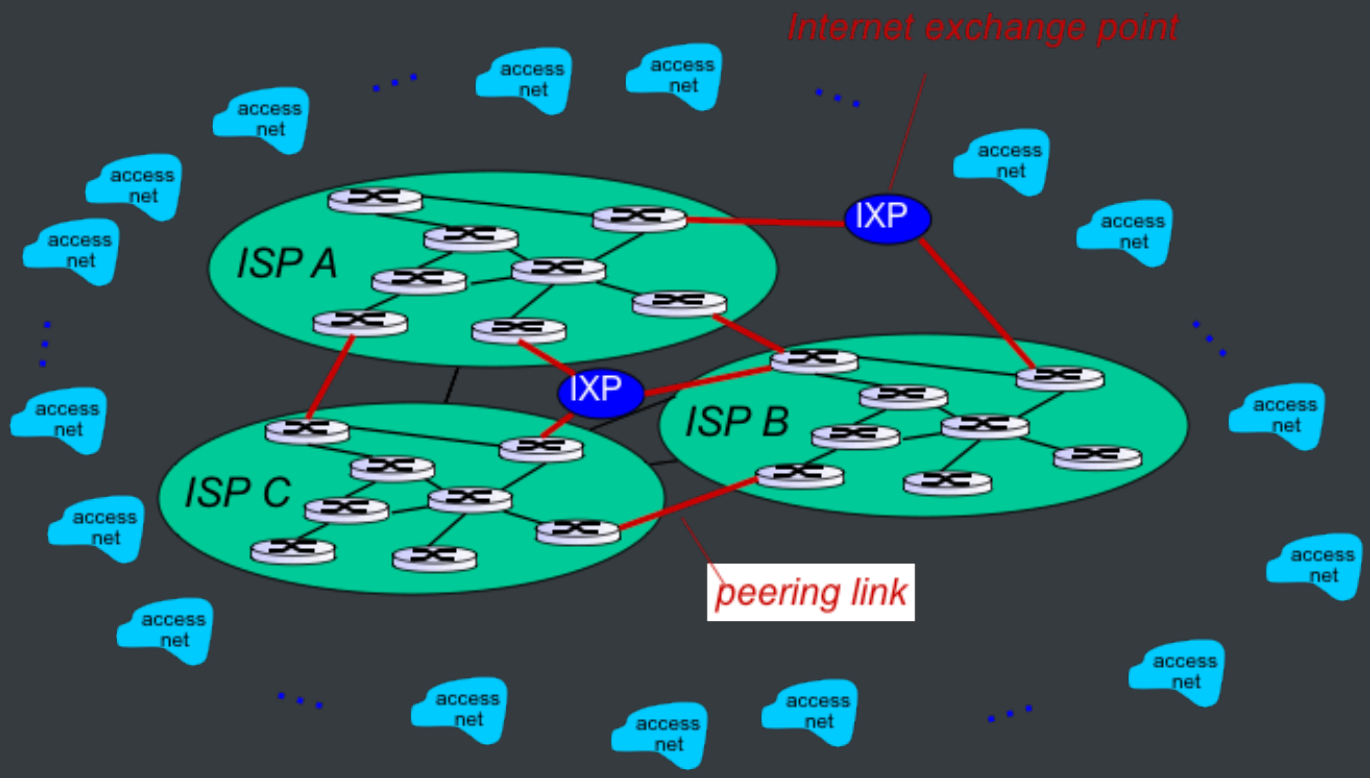
1.38

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



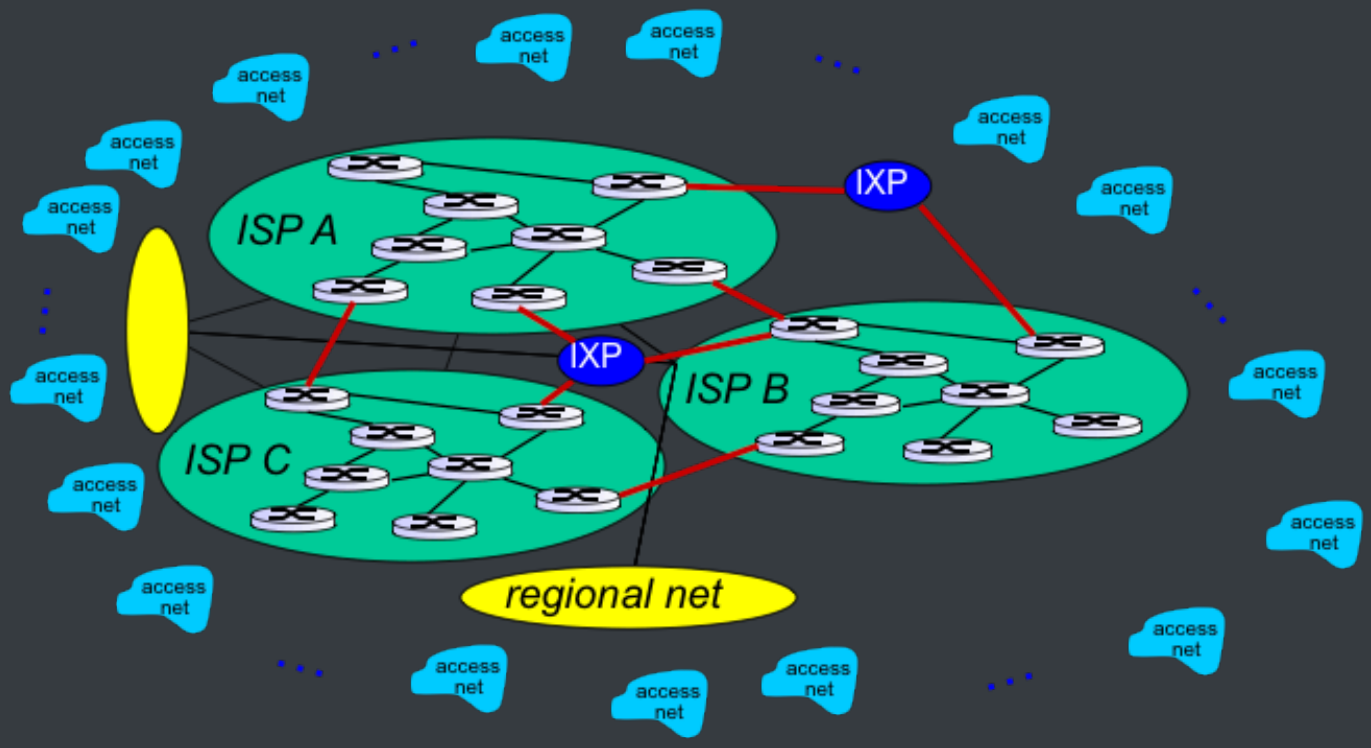
1.39

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



1.40

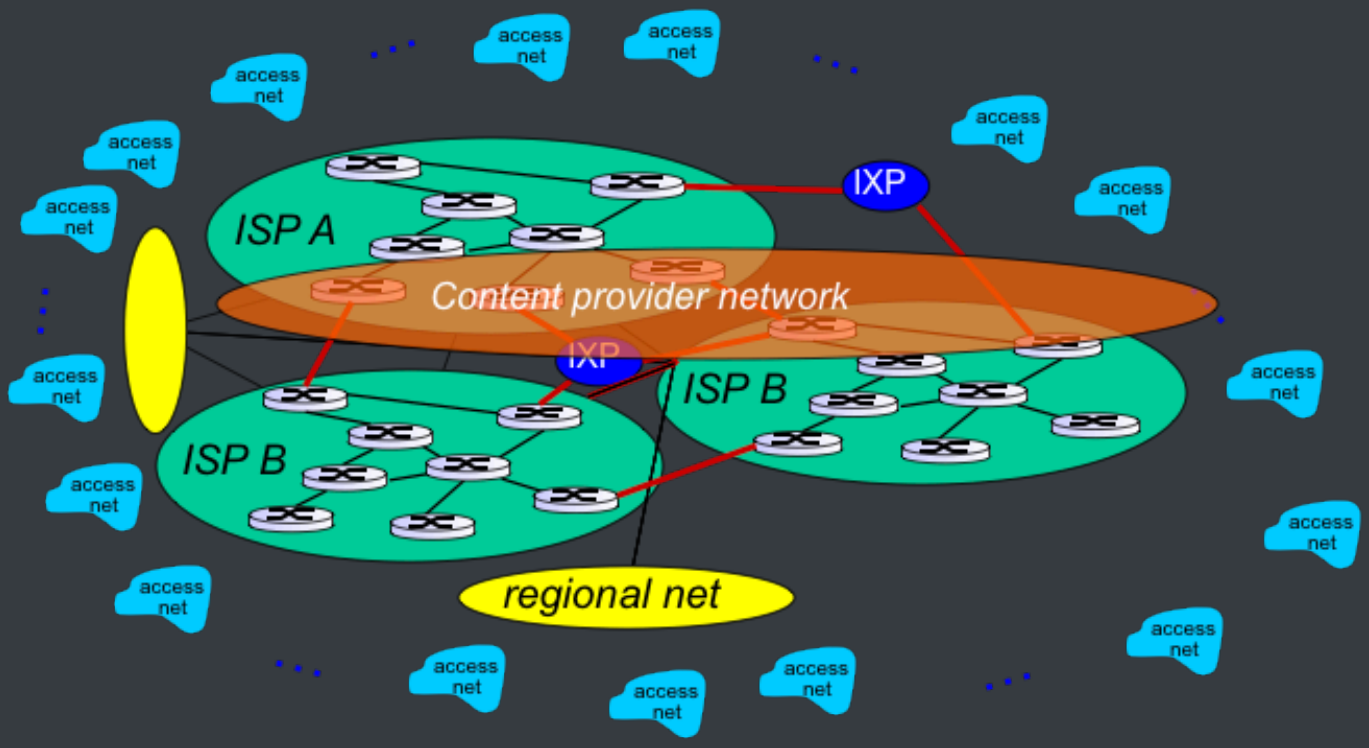
… and regional networks may arise to connect access nets to ISPS



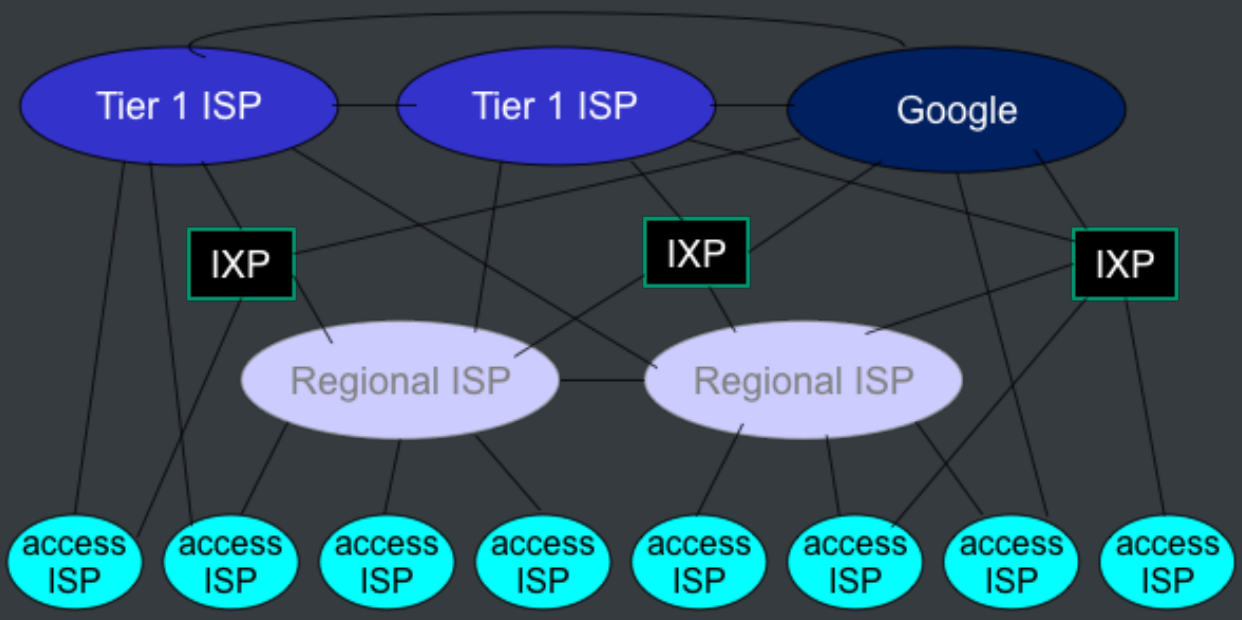
1.41

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

> CDN(Content Delivery Network)是指内容分发网络，也称为内容传送网络，这个概念始于1996年，是美国麻省理工学院的一个研究小组为改善互联网的服务质量而提出

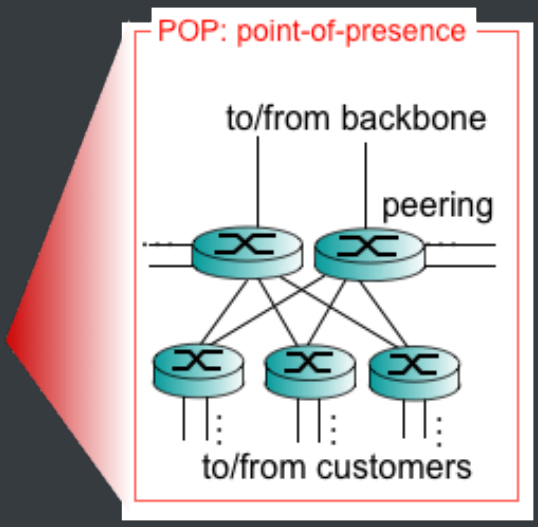


1.42



- at center: small # of well-connected large networks
-
  - “tier-1” commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
  - content provider network (e.g, Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs

Tier-1 ISP: e.g., Sprint



1.4 delay, loss, throughput in networks

4.分组交换网中的时延、丢包和吞吐量

A.结点总时延

结点总时延=结点处理时延+排队时延+传输时延+传播时延

a.结点处理时延

检查分组首部和决定将分组导向何处所需要的时间

b.排队时延

在输出队列中等待传输的时延，取决于到达队列的速率、链路的传输速率和到达流量的性质(周期性到达还是突发形式到达)

1.流量强度：

令a表示分组到达队列的平均速率(单位：pkt/s)；R为链路传输速率，即从链路推出比特的速率(b/s)；每个分组由比特组成。

则比率La/R为流量强度。

若La/R＞1,排队队列趋向于无界增加，排队时延趋向于无穷大；

若La/R≦1,排队时延随着流量强度的增加而增加。

设计系统时流量强度不能大于1

2.丢包

队列容量有限。满队列情况下，新分组将会被丢弃。

c.传输时延

将所有分组的比特推(传输)向链路的时间。与分组长度和链路传输速率有关

d.传播时延

链路起点向目标路由器传播所需要的时间。与路由器之间的距离和传播速率有关

B.端到端时延

假设源主机和目的主机之间有N-1台路由器。每台路由器和源主机的处理时延d(proc)、路由器和源主机的传输时延d(trans)、链路的传播时延d(prop).其中d(trans)=L/R.L为分组长度、R为路由器和源主机的输出速率。

则端到端时延d(end-end)=N(d(proc)+d(trans)+d(prop))

C.计算机网络中的吞吐量

文件传输过程中，接收方接收文件的速率。如果文件由F比特组成，接收文件需要T秒，则平均吞吐量为F/T bps

假设服务器和文件之间有N条链路，其传输速率分别是R1、R2、R3……RN。则文件传输的吞吐量为min{R1,R2……RN}。即整条路径的最小传输速率。

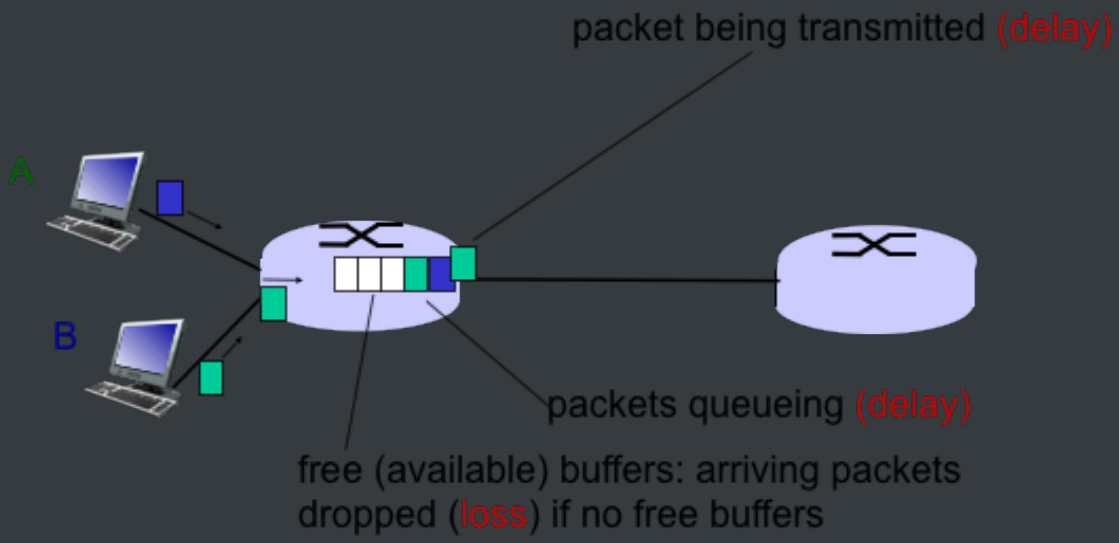
吞吐量不仅取决于沿着路径的传输速率，而且取决于干扰流量

1.45

How do loss and delay occur?

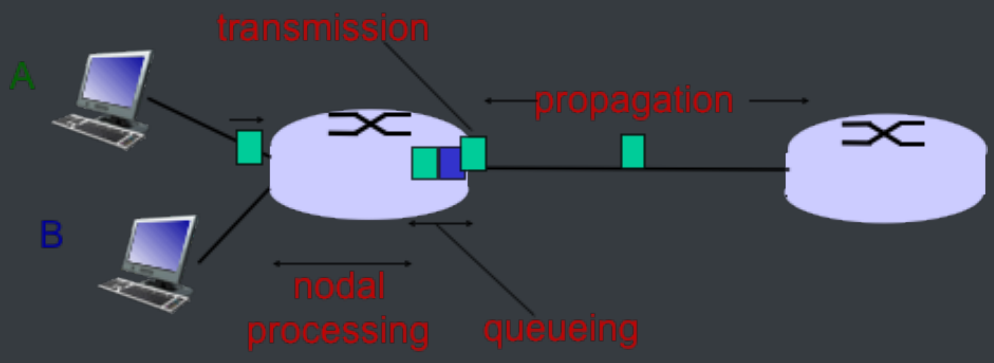
packets *queue* in router buffers

- packet arrival rate to link (temporarily) exceeds output link capacity
- packets queue, wait for turn



1.46

⭐Four sources of packet delay



1. ==d_{proc}: nodal processing ==

> 1.节点时延

1. check bit errors
2. determine output link
3. typically < msec
4. d_{queue}: queueing delay

> 排队时延

1. time waiting at output link for transmission
2. depends on congestion level of router

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

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

1.47

3. *d*trans: transmission delay:

> 传输时延

- *L*: packet length (bits)
- *R*: link *bandwidth (bps)*
- *d**trans = *L*/*R*

4. *d*prop: propagation delay:

> 传播时延

- *d*: length of physical link
- *s*: propagation speed in medium (~2x108 m/sec)

- *d*prop = *d*/*s*

d_{trans}: transmission delay:
- *L*: packet length (bits)
- *R*: link *bandwidth (bps)*
- d_{trans} = *L*/*R*

d_{prop}: propagation delay:
- *d*: length of physical link
- *s*: propagation speed in medium (~2x10⁸ m/sec)
- d_{prop} = *d*/*s*

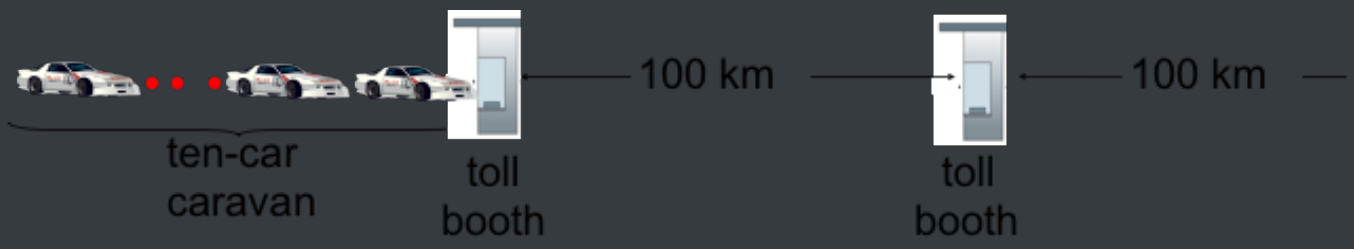
d_{trans} and d_{prop} *very* different

* Check out the Java applet for an interactive animation on trans vs. prop delay

1.48

Caravan analogy

> 卡车定理传输时延



- cars “propagate” at 100 km/hr
- toll booth takes 12 sec to service car (bit transmission time)
- car~bit; caravan ~ packet

Q: How long until caravan is lined up before 2nd toll booth?

- time to “push” entire caravan through toll booth onto highway = 12*10 = 120 sec
- time for last car to propagate from 1st to 2nd toll both: 100km/(100km/hr)= 1 hr
- *A:* 62 minutes

1.49

> 改变速度

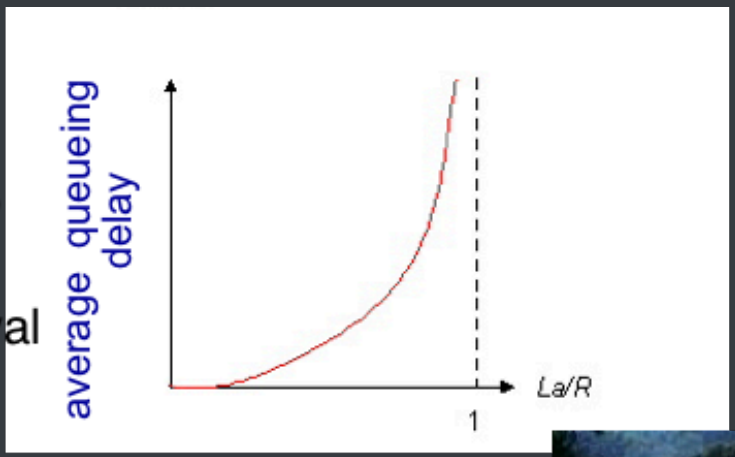
- suppose cars now “propagate” at 1000 km/hr
- and suppose toll booth now takes one min to service a car
- *Q:* Will cars arrive to 2nd booth before all cars serviced at first booth?

1.50

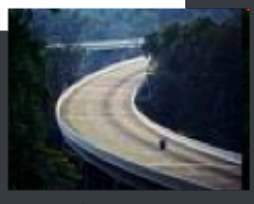
Queueing delay (revisited)

> 排队时延

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



traffic intensity = La/R



La/R ~ 0

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

La/R -> 1

* Check out the Java applet for an interactive animation on queuing and loss

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

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

1.51

win

```
tracert www.baidu.com
```

linux/unix

```
traceroute www.baidu.com
```

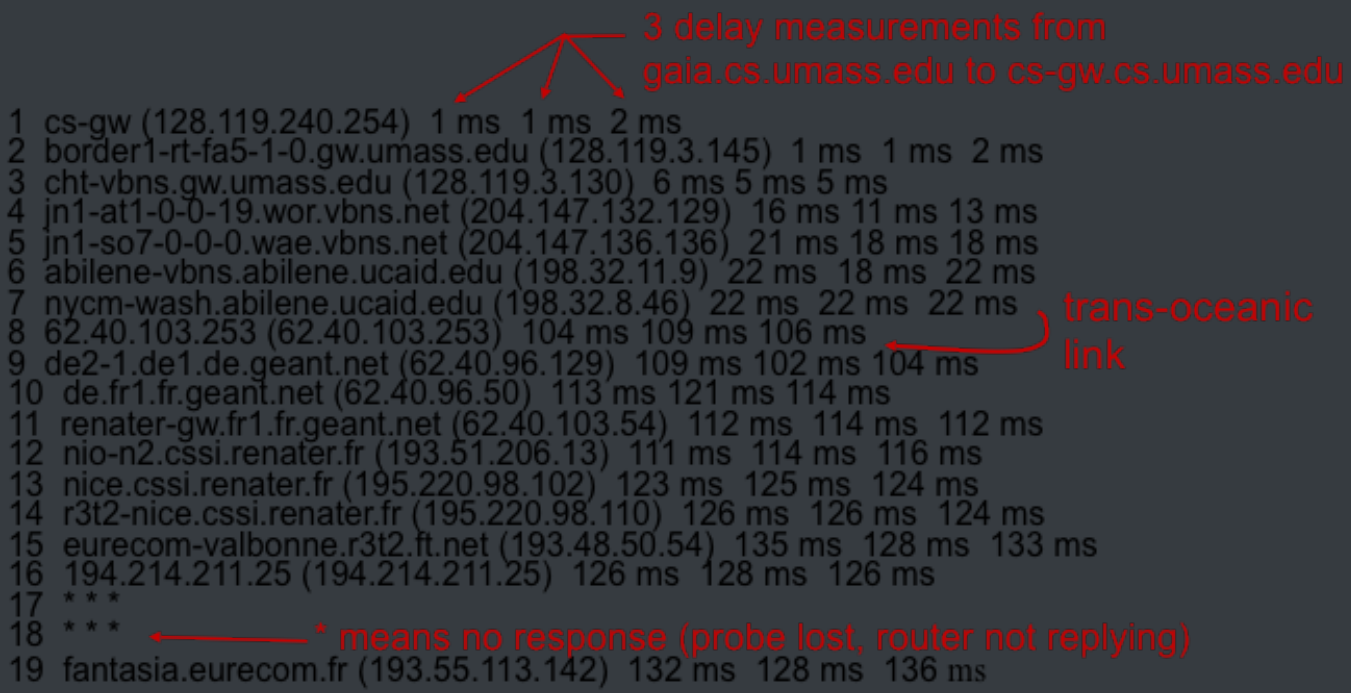
“Real” Internet delays and routes

> 跳数

- what do “real” Internet delay & loss look like?
- traceroute program: provides delay measurement from source to router along end-end Internet path towards destination. For all *i:*
-
    - sends three packets that will reach router *i* on path towards destination
    - router *i* will return packets to sender
    - sender times interval between transmission and reply.

1.52

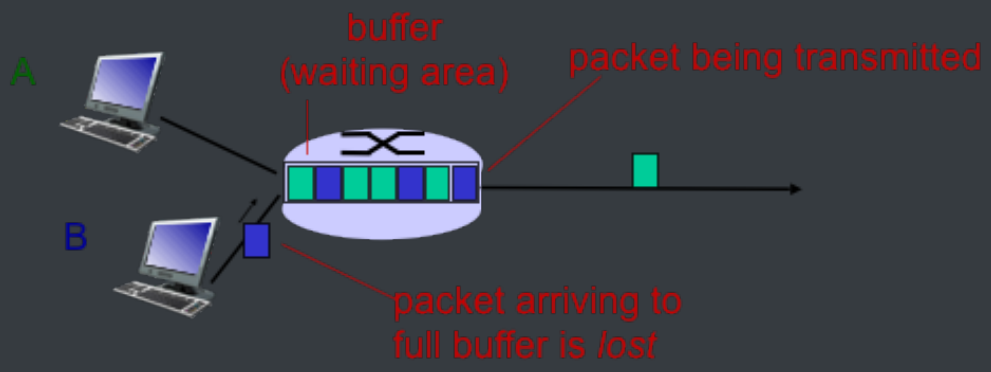
traceroute: gaia.cs.umass.edu to www.eurecom.fr



1.53

Packet loss

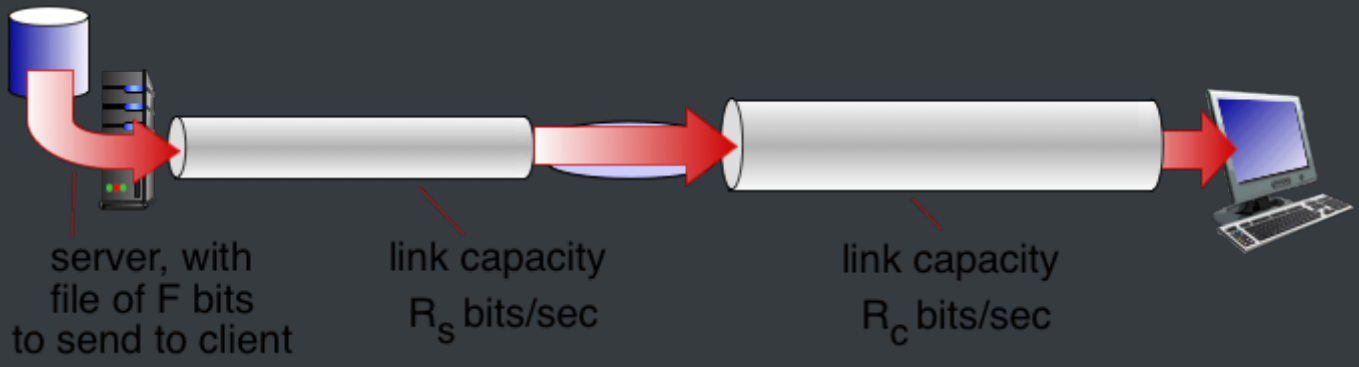
1. queue (aka buffer) preceding link in buffer has finite capacity
2. packet arriving to full queue dropped (aka lost)
3. lost packet may be retransmitted by previous node, by source end system, or not at all



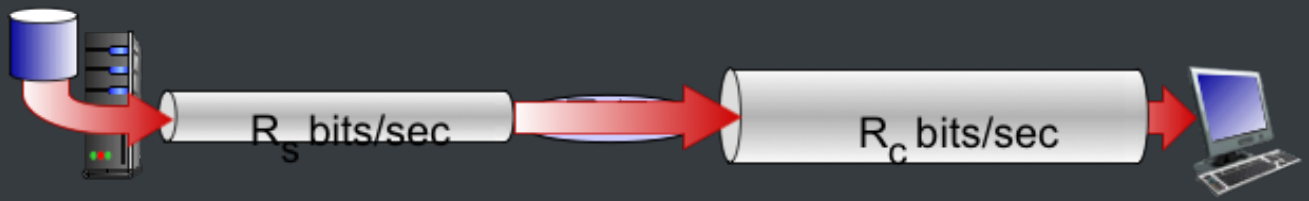
1.54

Throughput

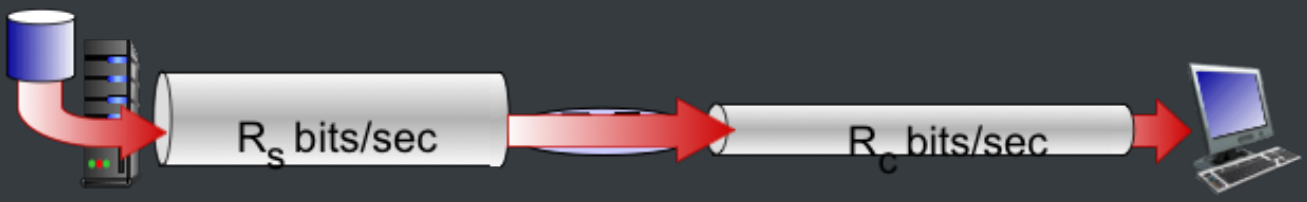
- *throughput:* rate (bits/time unit) at which bits transferred between sender/receiver
- 1. *instantaneous:* rate at given point in time
  2. *average:* rate over longer period of time



- *R**s < R**c* What is average end-end throughput?



- *R**s > R**c* What is average end-end throughput?



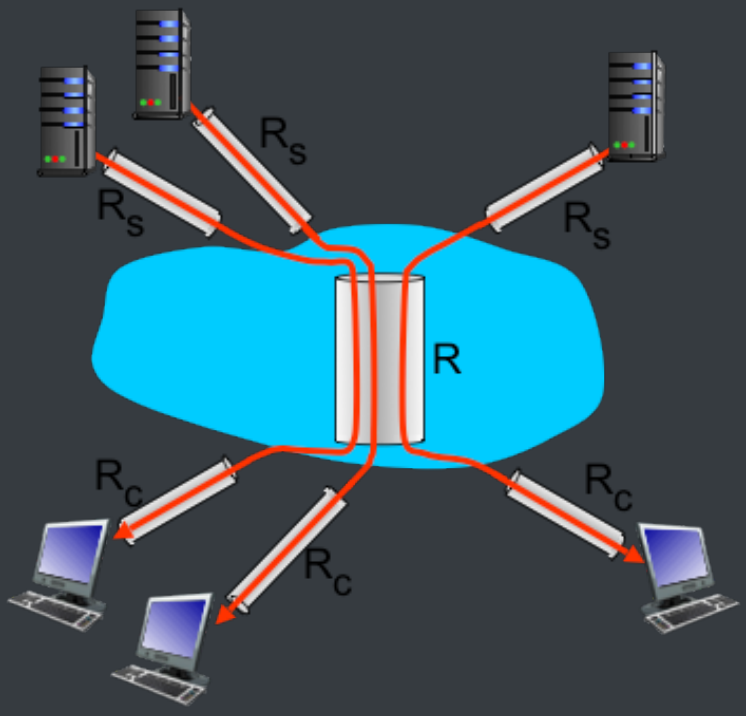
- *bottleneck link*

link on end-end path that constrains end-end throughput

1.56

Throughput: Internet scenario

- per-connection end-end throughput: min(Rc,Rs,R/10)
- in practice: Rc or Rs is often bottleneck



1.5 protocol layers[协议层次], service models[服务模型]

5.协议层次及其服务模型

A.分层的体系结构

a.协议分层（因特网五层协议栈）（概述）

1.应用层

网络应用程序及他们的应用层协议存留于应用层中。不同端系统中的应用程序通过应用层协议交换信息分组（报文）。应用层协议有：HTTP、SMTP、FTP、DNS。

2.传输层

传输层在应用程序端点之间传送报文段（应用层报文）。传输层协议有：TCP、UDP

3.网络层

网络层将数据报（网络层分组）从一台主机移动到另一台主机。网络层协议：IP协议、路由选择协议

4.链路层

将帧（链路层分组）从一个结点移动到路径的下一个结点。链路层协议：以太网、Wifi和电缆接入网的DOCSIS协议

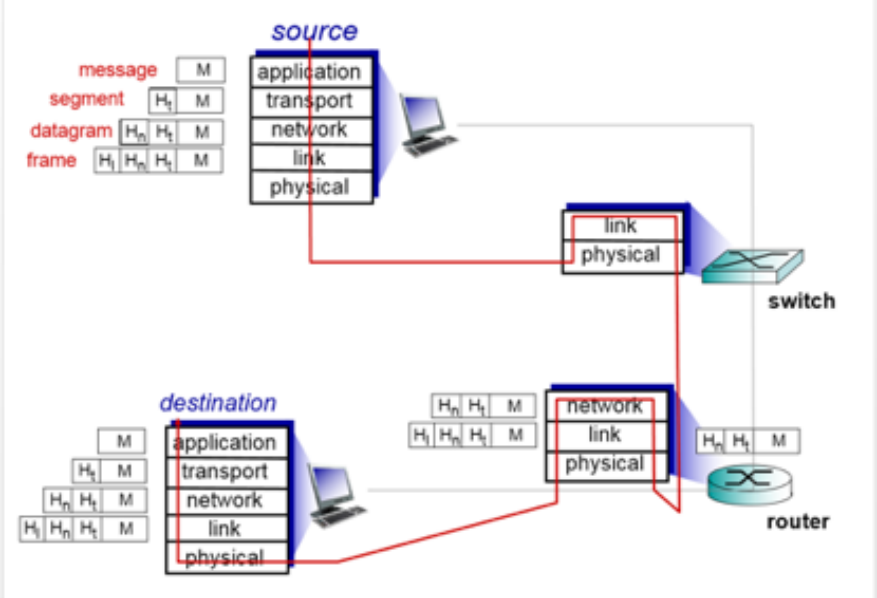
5.物理层

将帧中的一个个比特一个结点移动到路径的下一个结点。物理层协议与链路传输媒体相关。

b.OSI模型

【7层】自上而下：应用层、表示层、会话层、传输层、网路层、链路层、物理层

B.封装



每一层，一个分组具有两种类型的字段：首部字段和有效载荷字段。有效字段通常来源于上一层的分组。

1.58

Protocol “layers”

Networks are complex,

with many “pieces”:

- 1. hosts
  2. routers
  3. links of various media
  4. applications
  5. protocols
  6. hardware, software

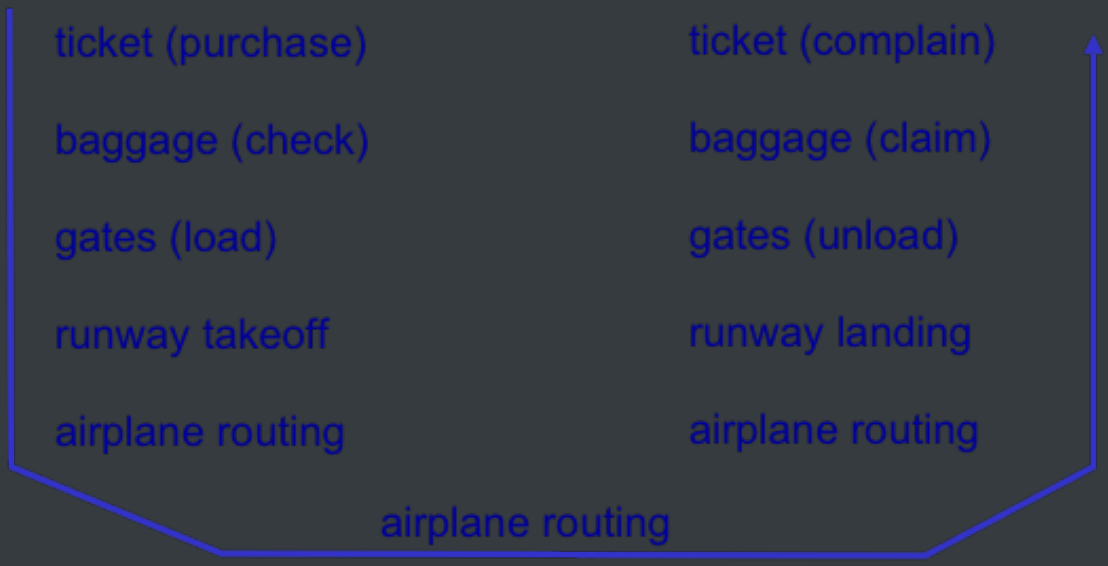
Question:

- is there any hope of *organizing* structure of network?

- …. or at least our discussion of networks?

1.59

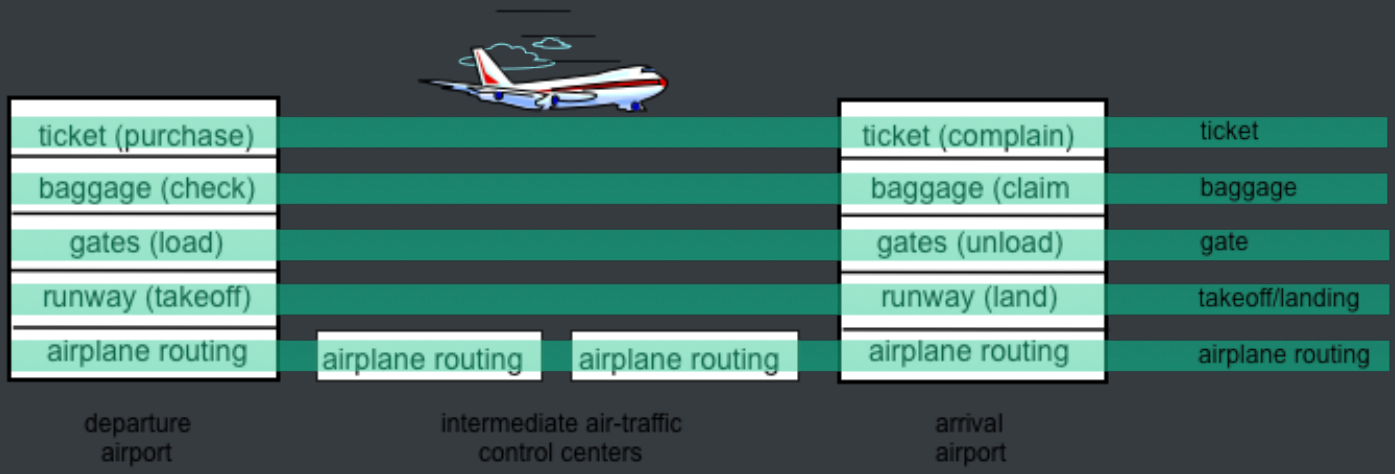
Organization of air travel



- a series of steps

1.60

Layering of airline functionality



layers: each layer implements a service

- 1. via its own internal-layer actions
  2. relying on services provided by layer below

1.61

Why layering?

dealing with complex systems:

1. explicit structure allows identification, relationship of complex system's pieces
2. - layered *reference model* for discussion
3. modularization eases maintenance, updating of system
4. - change of implementation of layer's service transparent to rest of system
   - e.g., change in gate procedure doesn't affect rest of system
5. layering considered harmful?

1.62

⭐Internet protocol stack

1. *application:* supporting network applications

    > 应用层

2.
    - FTP, SMTP, HTTP
3. *transport:* process-process data transfer

    > 传输层

4.
    - TCP, UDP
5. *network:* routing of datagrams from source to destination

    > 网络层

6.
    - IP, routing protocols
7. *link:* data transfer between neighboring network elements

    > 链路层

8.
    - Ethernet, 802.11 (WiFi), PPP
9. *physical:* bits “on the wire”

    > 物理层

1.63

ISO/OSI reference model

1. *presentation:* allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions

    > 表示层

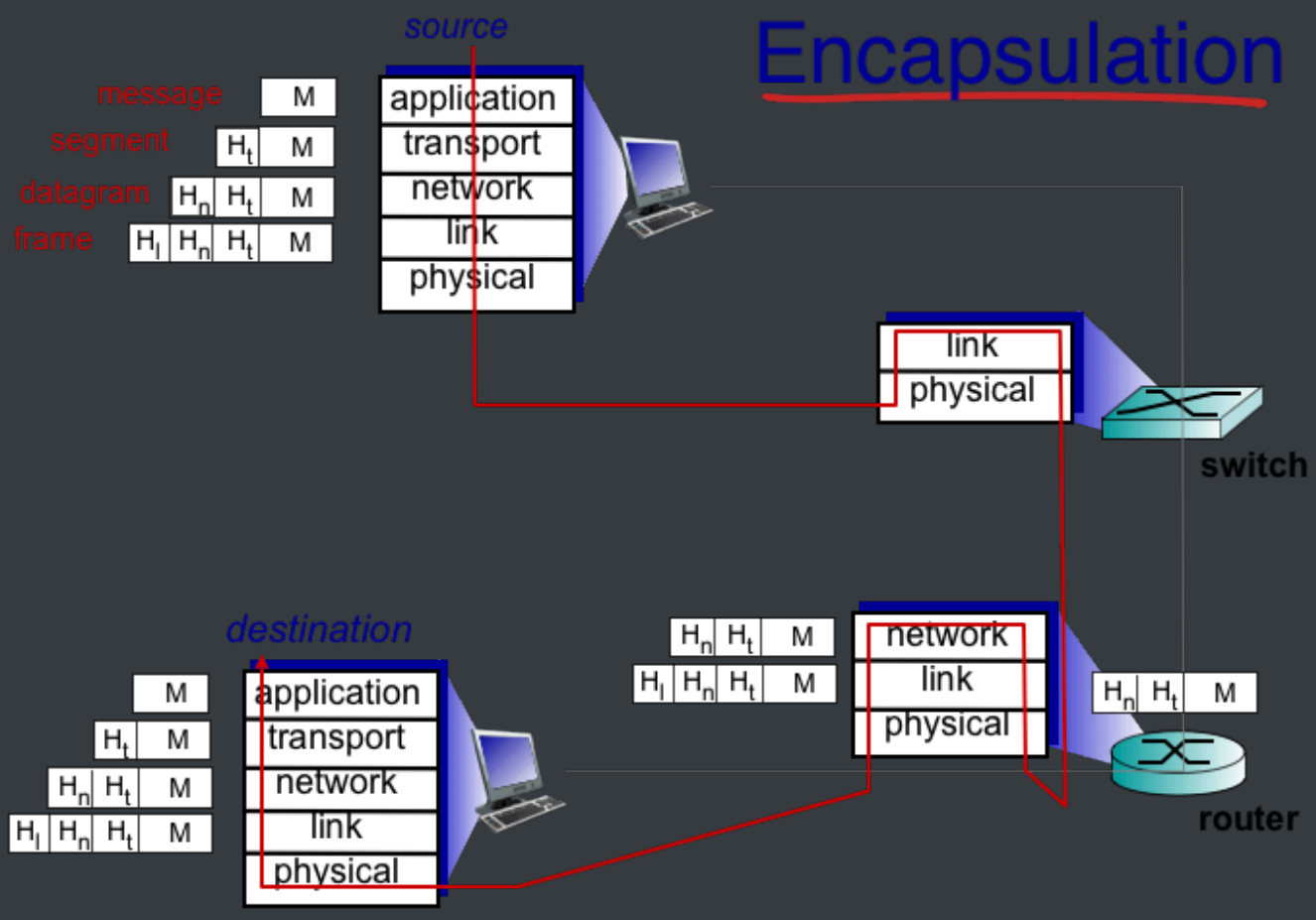
2. *session:* synchronization, checkpointing, recovery of data exchange

> 会话层

3. Internet stack “missing” these layers!
4.
   - these services, *if needed,* must be implemented in application
   - needed?

1.64

Encapsulation[封装📦]



1.6 networks under attack: security

6.面对攻击的网络

网络安全领域主要探讨以下问题：hacker如何攻击计算机网络，以及我们如何防御以免受他们的攻击，或者是更好的是设计能够事先免除这样的攻击的新型体系结构。

A.恶意软件(自我复制)

病毒：是一种需要某种形式的用户交互来感染用户设备的恶意软件。

蠕虫：是一种无需任何明显用户交互就能进入设备的恶意软件

B.拒绝服务攻击(DDOS)

1.弱点攻击。向一台目标主机运行的易受攻击的应用程序或操作系统发送精细制作的报文

2.带宽洪泛。攻击者向目标主机发送大量的分组，使得目标的接入链路拥塞，合法分组无法到达服务器。

3.连接洪泛。攻击者在目标主机中创建大量的半开或全开TCP连接使主机停止接受合法的连接。

C.分组嗅探器

在无线传输设备的附近放置一台被动的接收机，就能得到传输的每个分组的副本

D.IP哄骗

将具有虚假源地址的分组注入因特网的能力被称为IP哄骗。

1.66

Network security

1. field of network security:
2.
    1. how bad guys can attack computer networks
    2. how we can defend networks against attacks
    3. how to design architectures that are immune to attacks
3. Internet not originally designed with (much) security in mind
4.
    1. *original vision:* “a group of mutually trusting users attached to a transparent network” ☺
    2. Internet protocol designers playing “catch-up”
    3. security considerations in all layers!

1.67

Bad guys: put malware into hosts via Internet

1. malware can get in host from:
2.
    - *virus:* self-replicating infection by receiving/executing object (e.g., e-mail attachment)
    - *worm:* self-replicating infection by passively receiving object that gets itself executed
3. spyware malware can record keystrokes, web sites visited, upload info to collection site
4. infected host can be enrolled in botnet, used for spam. DDoS attacks

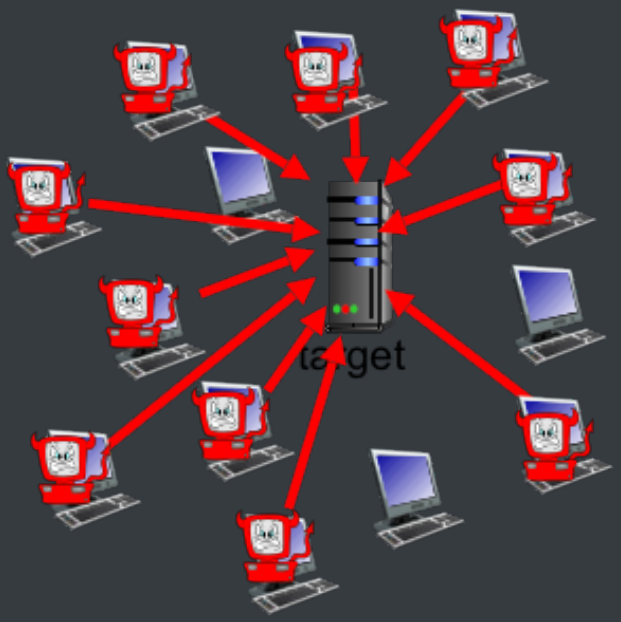
1.68

Bad guys: attack server, network infrastructure

⭐Denial of Service (DoS)

Denial of Service (DoS): attackers make resources (server, bandwidth) unavailable to legitimate traffic by overwhelming resource with bogus traffic

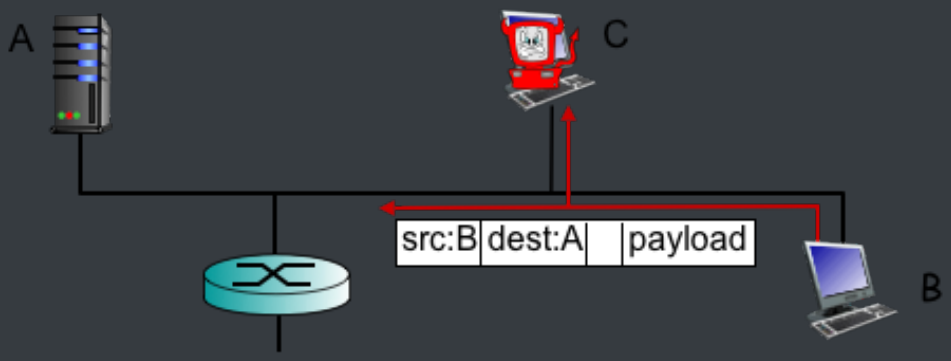
1. select target
2. break into hosts around the network (see botnet)
3. send packets to target from compromised hosts



1.69 Bad guys can sniff packets

packet "sniffing":

- 1. broadcast media (shared ethernet, wireless)
  2. promiscuous network interface reads/records all packets (e.g., including passwords!) passing by

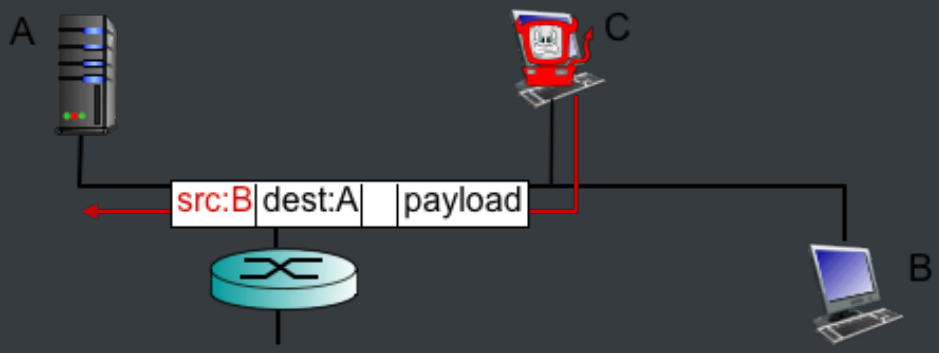


- wireshark software used for end-of-chapter labs is a (free) packet-sniffer

1.70

Bad guys can use fake addresses

IP spoofing: send packet with false source address



1.7 history

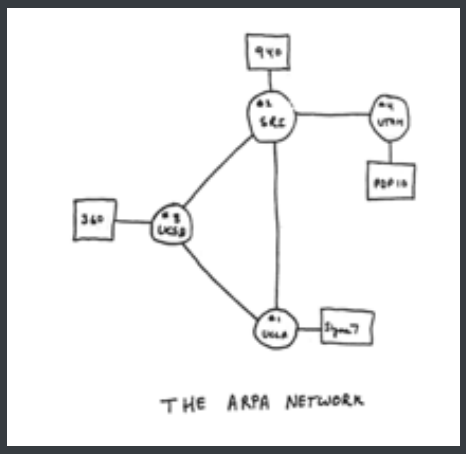
1.72

Internet history

1961-1972: Early packet-switching principles

- 1961: Kleinrock - queueing theory shows effectiveness of packet-switching
- 1964: Baran - packet-switching in military nets

- 1967: ARPAnet conceived by Advanced Research Projects Agency
- 1969: first ARPAnet node operational
- 1972:
-
    - ARPAnet public demo
    - NCP (Network Control Protocol) first host-host protocol
    - first e-mail program
    - ARPAnet has 15 nodes



1972-1980: Internetworking, new and proprietary nets

- 1970: ALOHAnet satellite network in Hawaii
- 1974: Cerf and Kahn - architecture for interconnecting networks
- 1976: Ethernet at Xerox PARC
- late70's: proprietary architectures: DECnet, SNA, XNA
- late 70's: switching fixed length packets (ATM precursor)
- 1979: ARPAnet has 200 nodes

Cerf and Kahn's internetworking principles:

-
    1. minimalism, autonomy - no internal changes required to interconnect networks
    2. best effort service model
    3. stateless routers

4. decentralized control

define today's Internet architecture

1980-1990: new protocols, a proliferation of networks

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

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

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

late 1990's – 2000's:

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

2005-present

- ~750 million hosts
- - Smartphones and tablets
- Aggressive deployment of broadband access
- Increasing ubiquity of high-speed wireless access
- Emergence of online social networks:
- - Facebook: soon one billion users
- Service providers (Google, Microsoft) create their own networks
- - Bypass Internet, providing “instantaneous” access to search, emai, etc.
- E-commerce, universities, enterprises running their services in “cloud” (eg, Amazon EC2)

1.8 Introduction : summary

covered a “ton” of material!

- Internet overview
- what’s a protocol?
- network edge, core, access network
- - packet-switching versus circuit-switching
  - Internet structure
- performance: loss, delay, throughput
- layering, service models
- security
- history

you now have:

- context, overview, “feel” of networking
- more depth, detail *to follow!*

Chapter 2 Application Layer

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail : SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

> 应用层是我们学习协议非常好的起点，它最为我们所熟悉。我们熟悉的很多应用就是建立在这些将要学习的协议基础上的。通过对应用层的学习，将有助于我们认知协议有关知识，将使我们了解到很多问题，这些问题当我们学习运输层、网络层及数据链路层协议时也同样会碰到。

> 研发网 络应用 程序的核 心 是写 出能够运行在不 同的端系统和通过网络彼此通信的程序。 例如，在W e b应用程序中，有两 个 互相通信的不 同的程序：一 个是运行在用户主机（桌面机、膝上机、平板电脑、智能电话等）上的浏览器程序；另一个是运行在Web服务器主机上的W e b 服务 器程序。 另 一 个 例子是P 2 P文件 共 享系统，在参 与文件 共 享的社区中的每台主机中都有一个程序.:，在这种情况下，在各台主 机中的这些程序可能都 是类似的或相同的。
>
> 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、J a v a或P y t h o n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」即使你要为网络核心设备写应用程序软件，你也不能做到这一 点。如 我们在第1章所知 ，以 及如 图1 - 2 4所 显示 的 那样，网络核心设备并不在应用层上 起作用，而仅在较低层起作用，特别是位 于网络层及 下面层次这种基本设计，也 即将应用软件限制在端系统（如阁2 - 1所示）的方法，促进了大量的网络应用程序的迅速研发和部署。

2.1 principles of network applications

our goals:

- conceptual, implementation aspects of network application protocols
- 1. transport-layer service models
  2. client-server paradigm
  3. peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
- 1. HTTP
  2. FTP
  3. SMTP / POP3 / IMAP
  4. DNS
- creating network applications
- - socket API

> 当进行软件编码之前，应当对应用程序有一个宽泛的体系结构计划。记住应用程序的体系结构明显不同于网络的体系结构（例如在第1章中所讨论的5层因特网体系结构）。
>
> 从应用程序研发者的角度看，网络体系结构是固定的，并为应用程序提供了特定的服务集合。在另一方面，应用程序体系结构（application architectur e）由应用程序研发者设 计，规定了如何在各种端系统上组织该应用程序„在选择应用程序体系结构时，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：
>
> 客户-服务器体系结构
>
> 对等（P2P）体系结构。
>
> 在客户-服务器体系结构（client-server architectur e）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 - 个 经 典 的 例 子 是 W e b 应 用 程序， 其 中 总 是 打 开 的 W e b 服 务 器 服 务 于 来 自 浏 览 器 （ 运 行 在 客 户 主 机 h ） 的 请 求 : ， 当W e b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。 值得 注 意的是利用客 户- 服务 器 体系结构.客户相互 之间不直接通信；例如，在W e b 应用中 两 个浏览器并不直接通信 。客户-服务 器 体系结构的另一个特征是该服务器具有固 定 的 . 周 知 的地 址， 该地 址称为1 P地 址（我们将 很 快讨 论它〉 。因为该服务 器 具有固 定的、 周知的地址，并且因为该服务 器总是打开的，客户总是能够通过向该服务器的地址 发 送 分 组 来 与 其 联 系 。 具 有 客 户- 服务 器 体系 结构的 非常著名的应用 程序包括W e b 、 FTP、Telnet和电子邮件。图2-2a中显示了这种客户-服务器体系结构：

2.4

Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing

- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

2.5

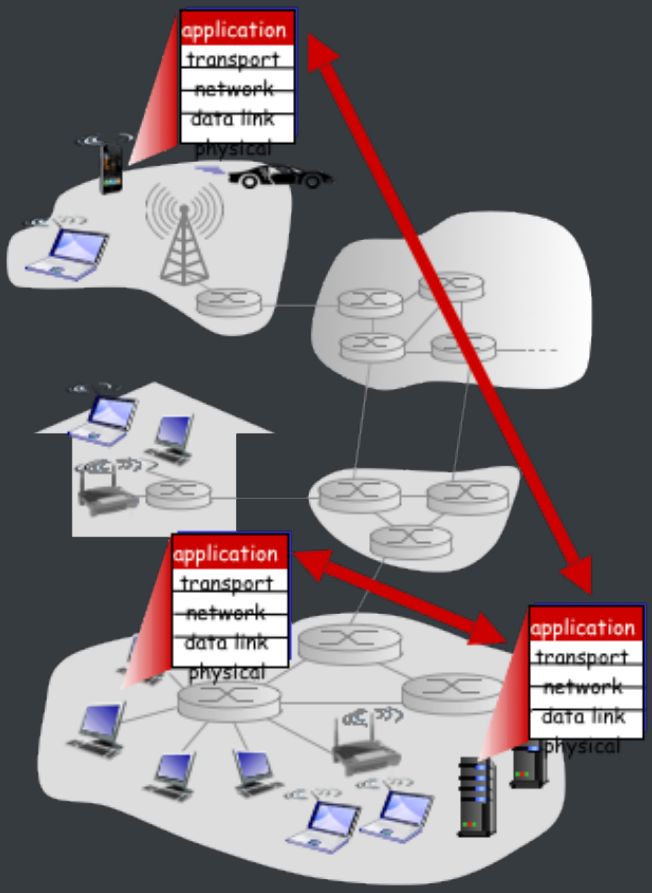
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



2.6

⭐Application architectures

possible structure of applications:

1. client-server
2. peer-to-peer (P2P)

2.7

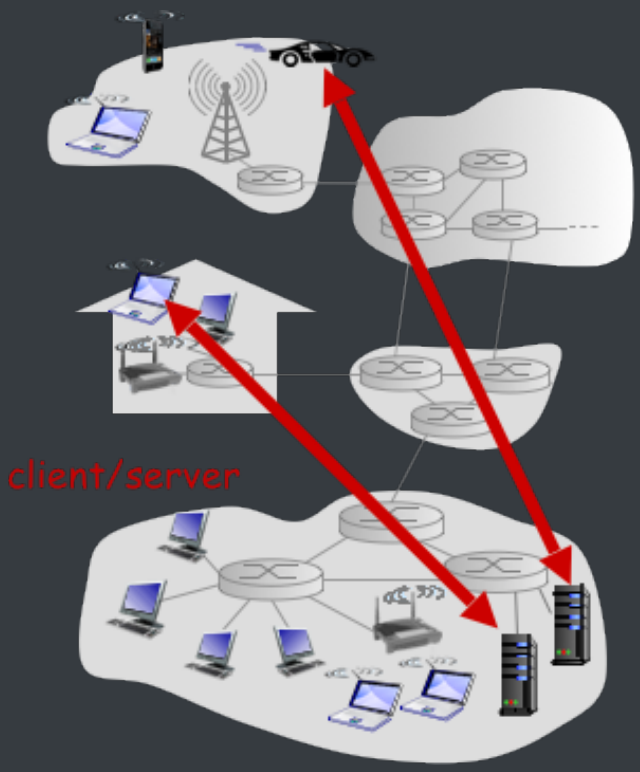
⭐Client-server architecture

server:

- always-on host
- permanent IP address
- data centers for scaling

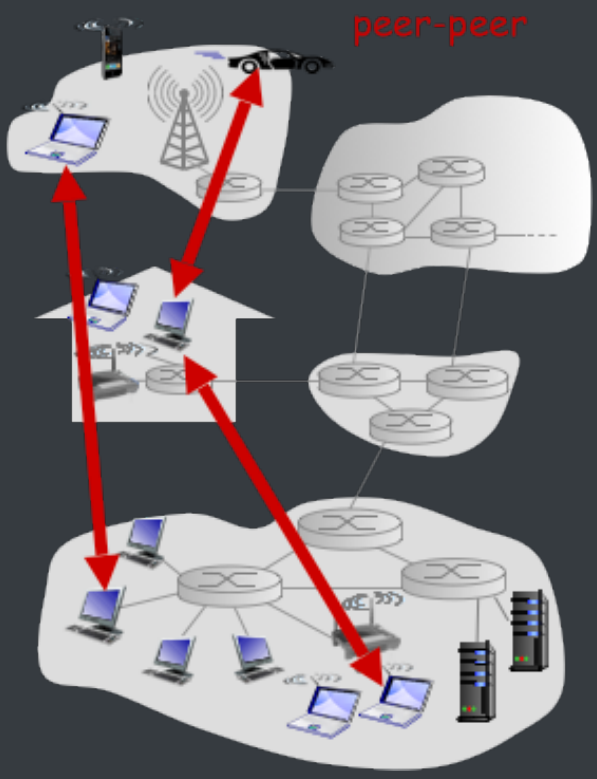
clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other



⭐P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
- - - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
- - - complex management



2.9

Processes communicating **进程通信**

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

client process: process that initiates communication

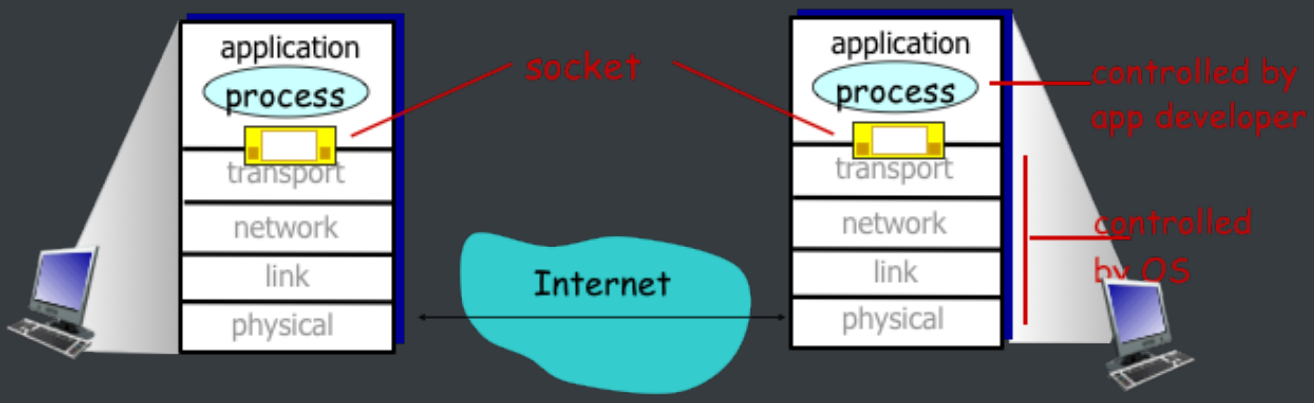
server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

2.10

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
-
    - sending process shoves message out door
    - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



2.11

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
- *A:* no, *many* processes can be running on same host

1. *identifier* includes both IP address and port numbers associated with process on host.
2. example port numbers:
    - HTTP server: 80
    - mail server: 25
3. to send HTTP message to gaia.cs.umass.edu web server:
    - IP address: 128.119.245.12
    - port number: 80
4. more shortly…

2.12

App-layer protocol defines

- types of messages exchanged,
- - e.g., request, response
- message syntax:
- - what fields in messages & how fields are delineated
- message semantics
- - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

2.13

What transport service does an app need?

1. data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

2. timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

3. throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

> 3. 定时 运 输层协议也能提供定时保证。如同 具有吞 吐量保证那样，定时保证

> 能够以多种形式实现。 一 个 保证的例子如：发送方注人进套接字中的每个比特到达接收方的套接字不 迟于1００ｍｓ。这种服务将对交 互式实时应用程序有吸引力，如因特网电话、虚拟环境、 电 话 会 议 和 多 方 游 戏 ， 所有 这 些服务 为 了有 效 性而 要求 数 据交 付有严格的时间 限制 (参 见第7章，[Ｇａｕｔｈｉｅｒ１９９９;Ｒａｍｊｅｅ１９９４])。 例如，在因特网电话中，较长的时延会导致会话中 出现不自然的停顿；在多方游戏和虚拟互 动环境中，在做 出 动 作并看到来自环境 (如 来 自 位 于 端 到 端 连 接 中 另 一 端 点 的 玩 家) 的 响 应 之 间 ， 较 长 的 时 延 使 得 它 失 去 真实感。对于非实时的应用，较低的时延总比较高的时延好，但对端到端的时延没有严 格的约束。

4. security

- encryption, data integrity, …

> 4. 安全性 最 后 ， 运 输 协 议 能 够 为 应 用 程 序 提 供 一 种 或 多 种 安 全 性 服 务 ， 例 如 ， 在 发 送 主 机中，运输协议能够加密由发送进程传输的所有数据，在接收主机中，运输层协议能够在将数据交 付给接收进程之 前解密这些数据。这种服务将在发送和接收进程之间提供机密性，以防该数据以某种方式在这两 个进程之间被观察到 。运输协议还能提供除了机密性以外的其 他安全性 服务，包括 数 据 完 整 性和端点鉴别，我们将 在第8章中详 细讨论这些 主题。

2.14

Transport service requirements: common apps

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

> 至此，我们已经考虑了计算机网络能够一般性地提供的运输服务。现在我们要更为具 体地考察由因特网提供的运输服务类型。因特网（更一般的是T C P / I P网络）为应用程序提供 两 个运输层协议，即U D P和T C P。当你（作 为 一 个软件开发者）为因特网创建一 个 新的应用时，首先要做出的决定是，选择UDP还是选择TCP。每个协议为调用它们的应用程序提供了不同的服务集合。图2-4显示了某些所选的应用程序的服务要求。

2.15

Internet transport protocols services

TCP service:

1. *reliable transport* between sending and receiving process
2. *flow control:* sender won’t overwhelm receiver

3. *congestion control:* throttle sender when network overloaded
4. *does not provide:* timing, minimum throughput guarantee, security
5. *connection-oriented:* setup required between client and server processes

UDP service:

1. *unreliable data transfer* between sending and receiving process
2. *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, orconnection setup,

> UDP是一种不提供不必要服务的轻量级运输协议，它仅提供最小服务。UDP是无连接的，因此在两个进程通信前没有握手过程。UDP协议提供一种不可靠数据传送服务，也 就是说，当进程将一个报文发送进UDP套接字时，UDP协议并不保证该报文将到达接收进程。不仅如此，到达接收进程的报文也可能是乱序到达的。 UDP没有包括拥塞控制机制，所以UDP的发送端可以用它选定的任何速率向其下层（网络层）注人数据。（然而，值得注意的是实际端到端吞吐量可能小于这种速率，这可能是因为中间链路的带宽受限或因为拥塞而造成的。）

Q: why bother? Why is there a UDP?

2.16

Internet apps: application, transport protocols

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

> 至此，我们已经考虑了计算机网络能够一般性地提供的运输服务。现在我们要更为具 体地考察由因特网提供的运输服务类型。因特网（更一般的是TCP /IP网络）为应用程序提供两个运输层协议，即UDP和TCP。当你（作为一个软件开发者）为因特网创建一个 新的应用时，首先要做出的决定是，选择UDP还是选择TCP。每个协议为调用它们的应用程序提供了不同的服务集合。图2-4显示了某些所选的应用程序的服务要求。

2.17

Securing TCP

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- Apps use SSL libraries, which “talk” to TCP

SSL socket API

-
    - cleartext passwds sent into socket traverse Internet encrypted
    - See Chapter 7

> 无论TCP还是UDP都没有提供任何加密机制，这就是说发送进程传进其套接字的数据，与经网络传送到目的进程的数据相同。因此，举例来说如果某发送进程以明文方式（即没有加密）发送了一个口令进入它的套接字，该明文口令将经过发送方与接收方之间的所有链路传送，这就可能在任何中间链路被嗅探和发现，因为隐私和其他安全问题对许多应用而言已经成为至关重要的问题，所以因特网界已经研制了 TCP 的加强版本，称为安全套接字层（Secure Sockets Layer，SSL）。用SSL加强后的TCP不仅能够做传统的TCP所能做的一切，而且提供了关键的进程到进程的安全性服务，包括加密、数据完整性和端点鉴别。我们强调SSL不是与TCP和UD P在相同层次上的第三种因特网运输协议，而是一种对TCP的加强，这种强化是在应用层上实现的。特别是，如果一个应用程序要使用SS L的服务，它需要在该应用程序的客户端和服务器端包括SSL代码（利用现有的、高度优化 的库和类SS L有它 自己的套接 字API ,这类 似于 传统的TCP 套接字API,当一个应用使用SS L时，发送进程向SS L套接字传递明文数据；在发送主机中的SSL则加密该数据并将加密的数据传递给TCP套接字。加密的数据经因特网传送到接收进程中的TCP套接字。该接收套接字将加密数据传递给SSL，由其进行解密。最后，SSL通过它的SSL套接字将明文数据传递给接收进程。我们将在第8章中更为详细地讨论SSL。

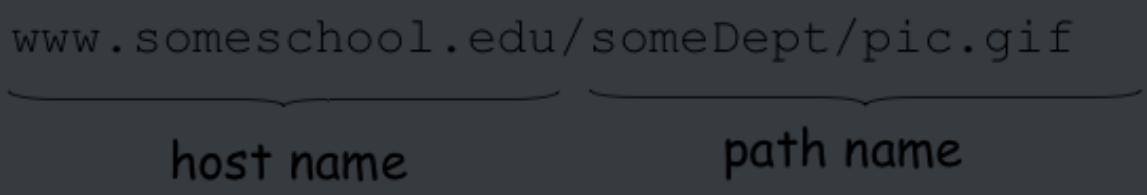
2.2 Web and HTTP(Hypertext Transfer Protocol)

2.19

Web and HTTP

First, a review…

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,



2.20

HTTP overview

⭐HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
-
    - *client**:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
    - *server:* Web server sends (using HTTP protocol) objects in response to requests

2.21

HTTP overview (continued)

uses TCP:

1. client initiates TCP connection (creates socket) to server, port 80
2. server accepts TCP connection from client
3. HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
4. 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

2.22

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
- - - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

2.23

Non-persistent HTTP

suppose user enters URL:

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

2.25

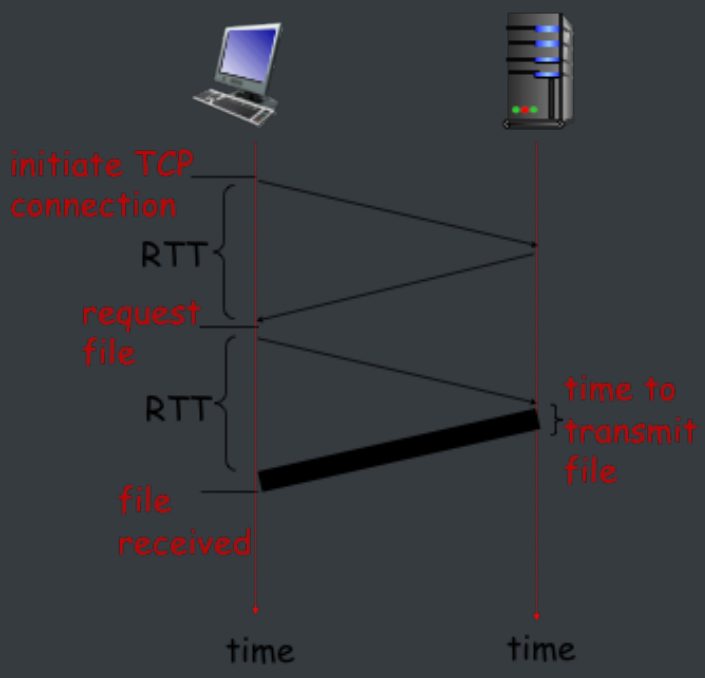
Non-persistent HTTP: response time

⭐RTT(Round-Trip Time)

(definition): time for a small packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time = 2RTT+ file transmission time



2.26

Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

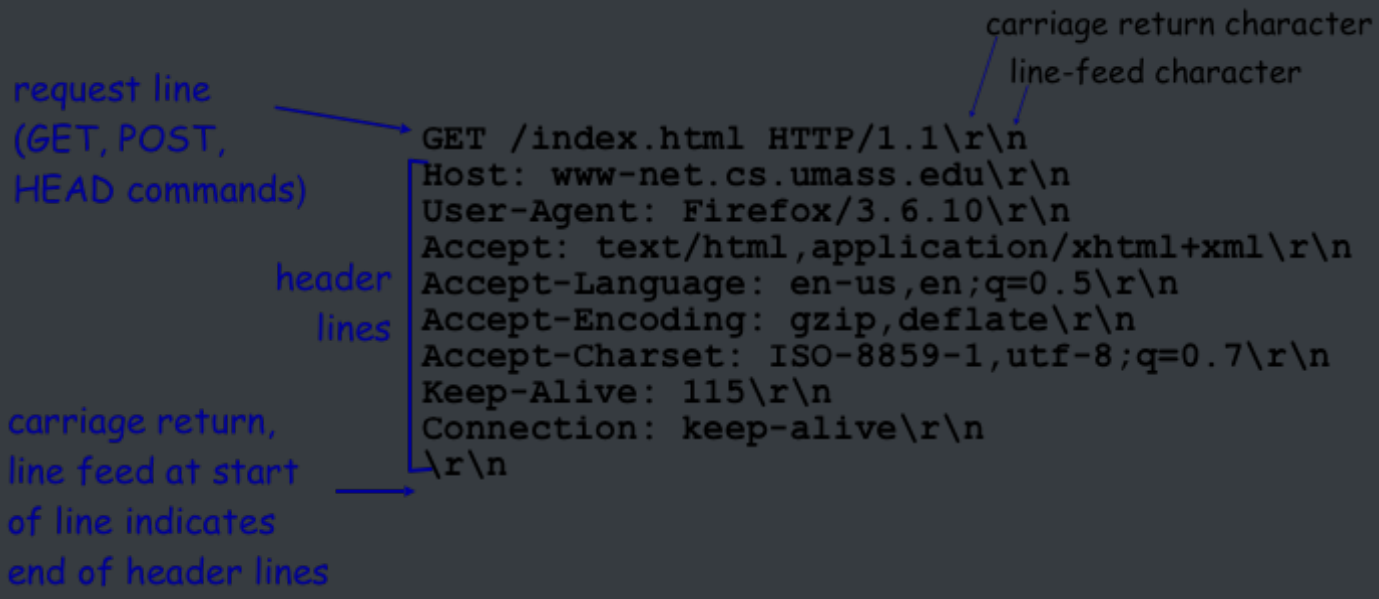
persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

2.27

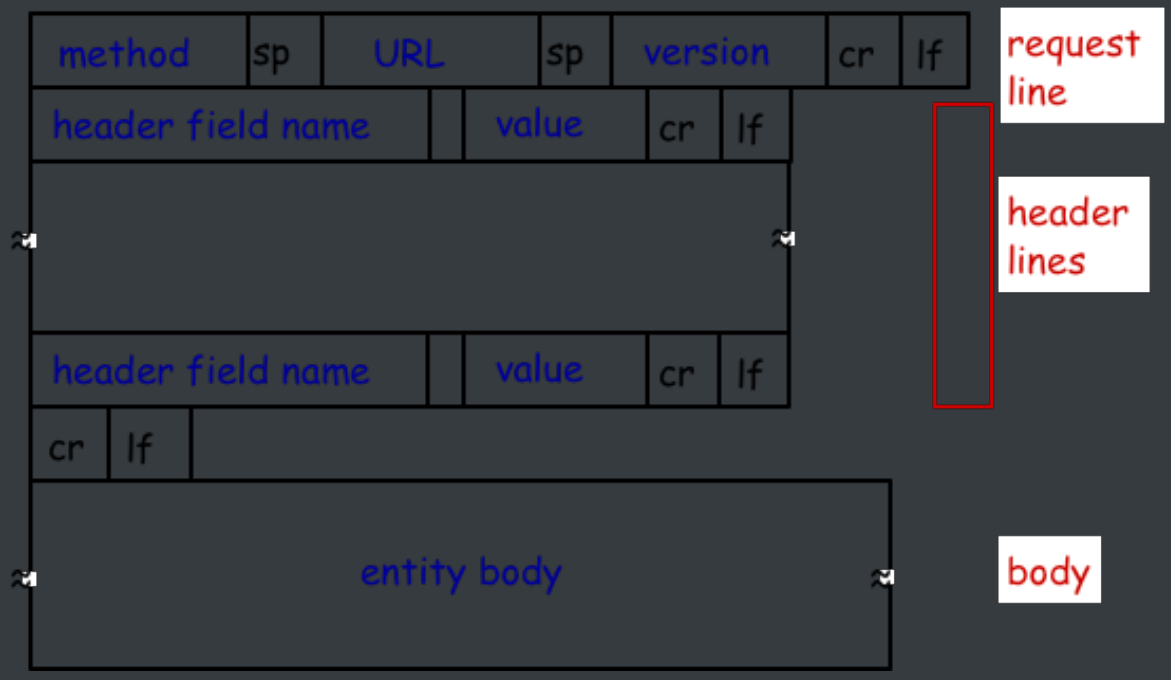
HTTP request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
-
    - ASCII (human-readable format)



2.28

HTTP request message: general format



2.29

Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

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

2.30

Method types

HTTP/1.0:

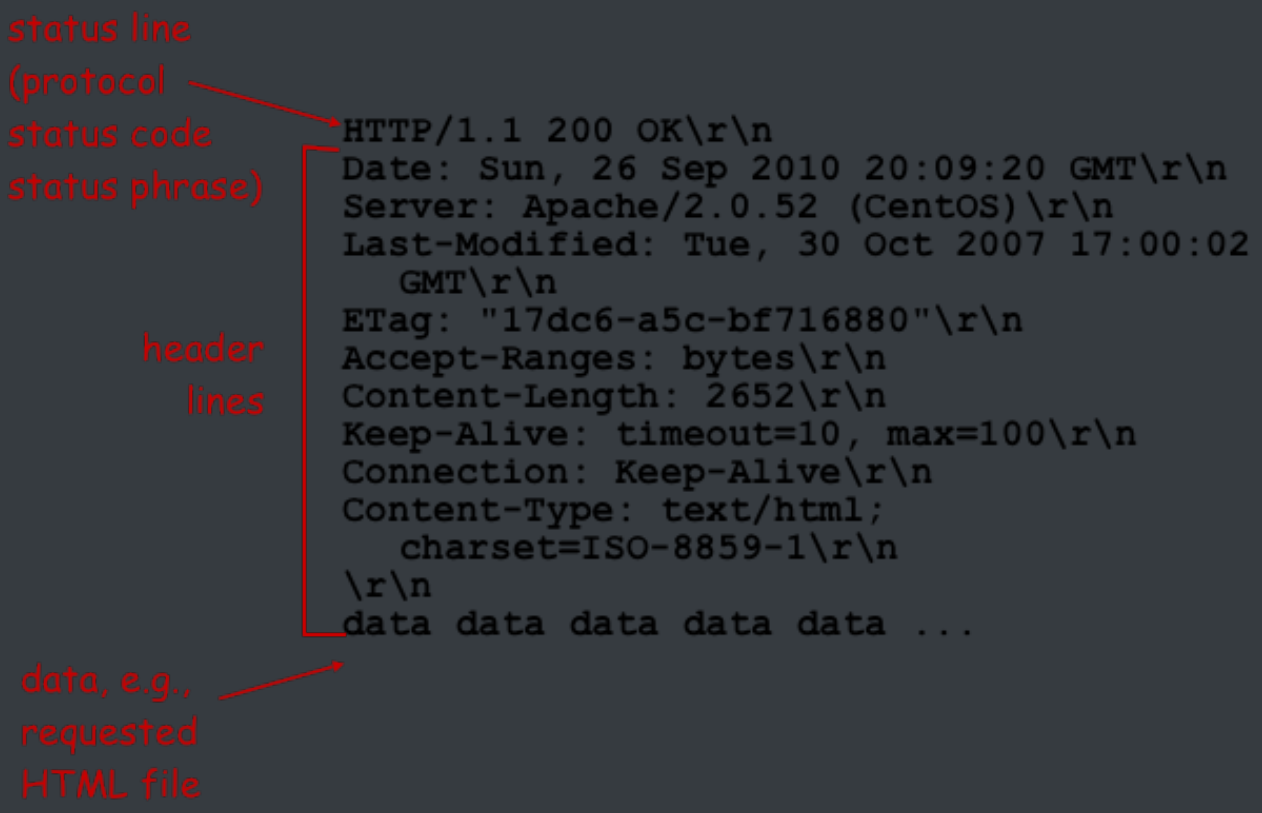
- GET
- POST
- HEAD
    - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
    - uploads file in entity body to path specified in URL field
- DELETE
    - deletes file specified in the URL field

2.31

HTTP response message



2.32

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

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

400 Bad Request

- request msg not understood by server

404 Not Found

- - requested document not found on this server

505 HTTP Version Not Supported

2.33

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet cis.poly.edu 80

opens TCP connection to port 80

(default HTTP server port) at cis.poly.edu.

anything typed in sent

to port 80 at cis.poly.edu

2. type in a GET HTTP request:

GET /~ross/ HTTP/1.1

Host: cis.poly.edu

2.34

User-server state: cookies

many Web sites use cookies

four components:

1) cookie header line of HTTP *response* message

2) cookie header line in next HTTP *request* message

3) cookie file kept on user's host, managed by user's browser

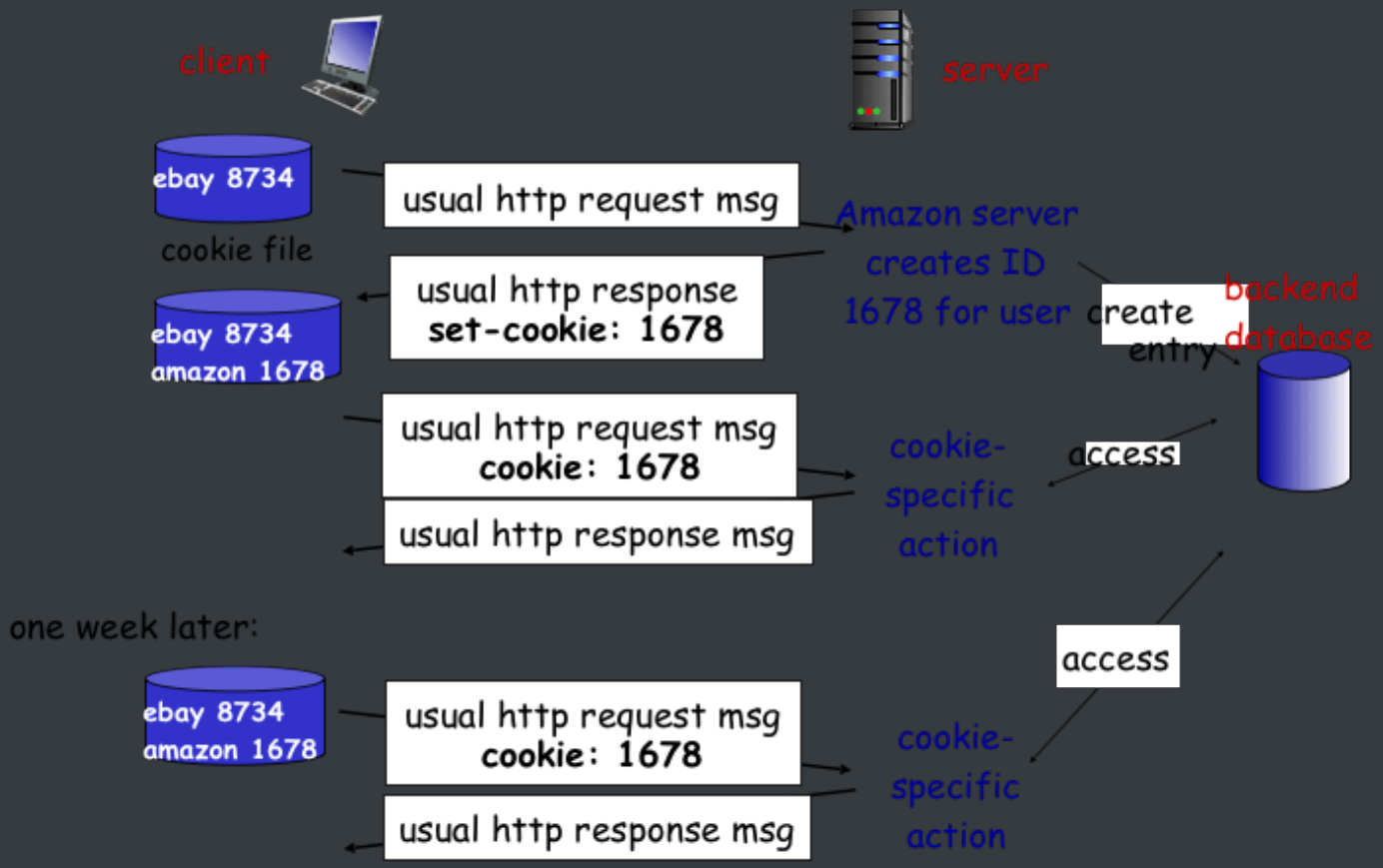
4) back-end database at Web site

example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
-
    - unique ID
    - entry in backend database for ID

2.35

Cookies: keeping "state" (cont.)



2.36

what cookies can be used for:

1. authorization
2. shopping carts
3. recommendations
4. user session state (Web e-mail)

how to keep “state”:

1. protocol endpoints: maintain state at sender/receiver over multiple transactions
2. cookies: http messages carry state

cookies and privacy:

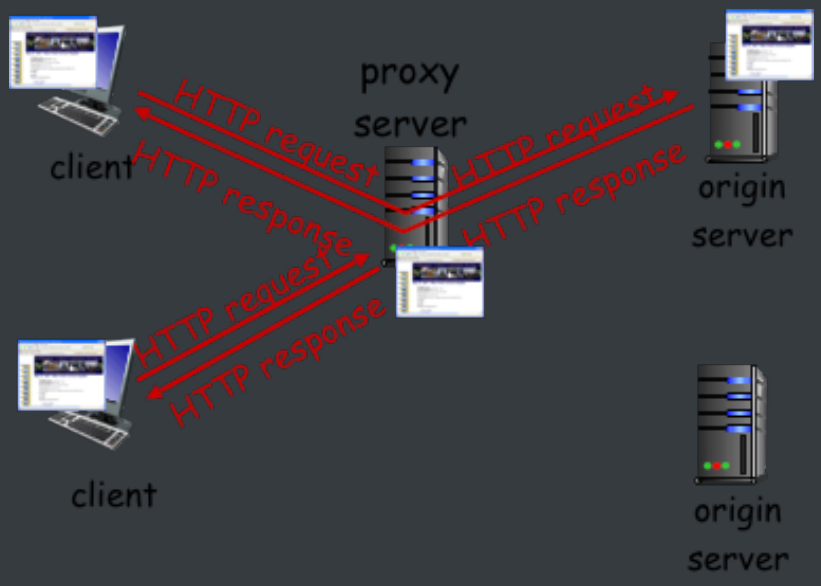
1. cookies permit sites to learn a lot about you
2. you may supply name and e-mail to sites

2.37

Web caches (proxy server)

goal: satisfy client request without involving origin server

1. user sets browser: Web accesses via cache
2. browser sends all HTTP requests to cache
3.
    - object in cache: cache returns object
    - else cache requests object from origin server, then returns object to client



2.38

More about Web caching

1. cache acts as both client and server

2.
   - server for original requesting client
   - client to origin server
3. typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

1. reduce response time for client request
2. reduce traffic on an institution's access link
3. Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

2.39

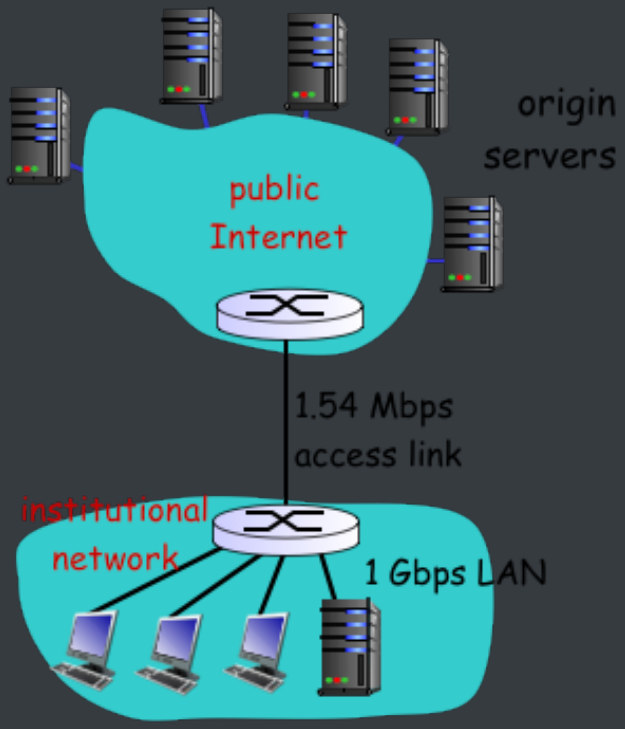
Caching example:

assumptions:

1. avg object size: 100K bits
2. avg request rate from browsers to origin servers:15/sec
3. avg data rate to browsers: 1.50 Mbps
4. RTT from institutional router to any origin server: 2 sec
5. access link rate: 1.54 Mbps

consequences:

1. LAN utilization: 15%
2. access link utilization = 99%
3. total delay = Internet delay [problem!] access delay + LAN delay = 2 sec + minutes + usecs



2.40

Caching example: fatter access link

assumptions:

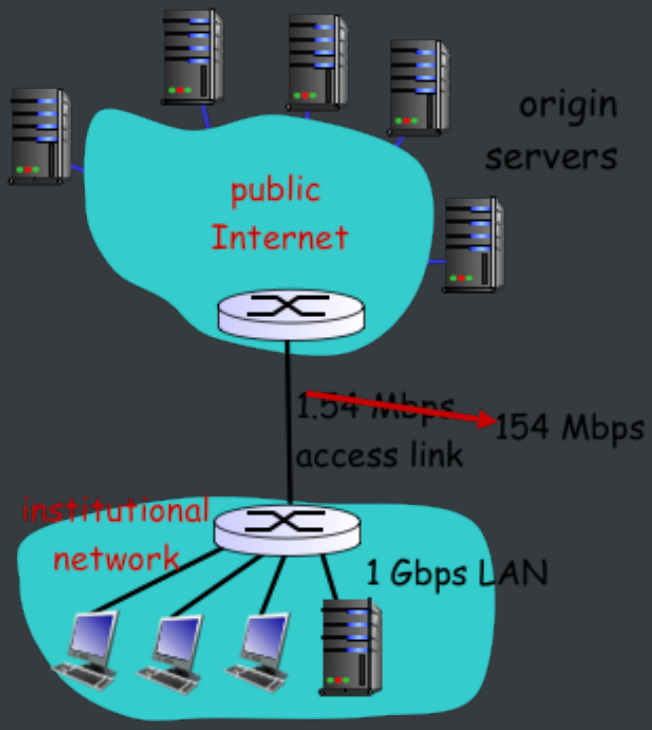
- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps → 154 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 99% → 9.9%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + usecs → msecs

Cost: increased access link speed (not cheap!)



2.41

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

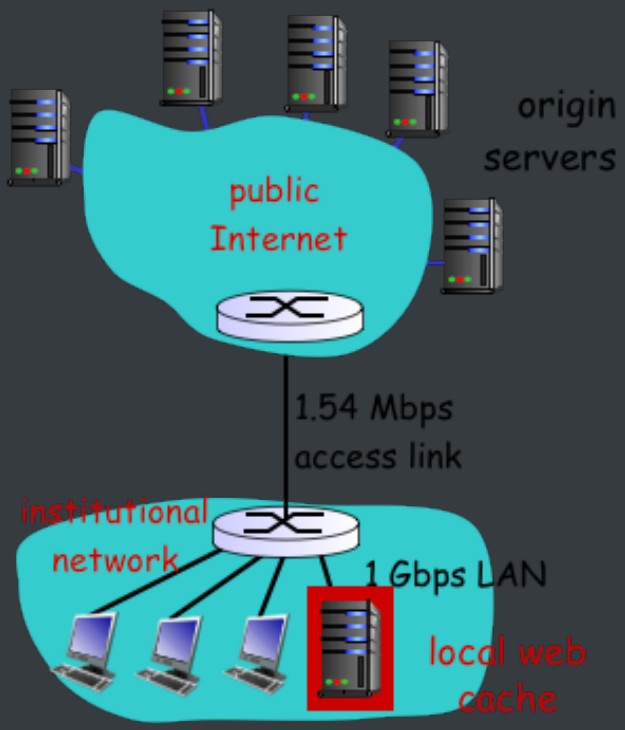
consequences:

- LAN utilization: 15%
- access link utilization = 100%
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + minutes + usecs

How to compute link utilization, delay?

Cost: web cache (cheap!)



2.42

Caching example: install local cache

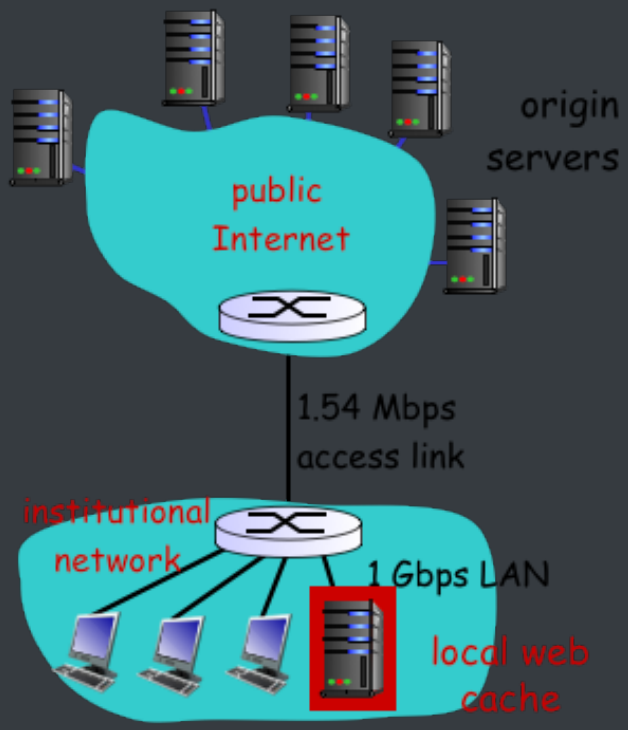
Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4

- - 40% requests satisfied at cache, 60% requests satisfied at origin

- access link utilization:
- - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
- - utilization = 0.9/1.54 = .58

- total delay
- - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)



2.43

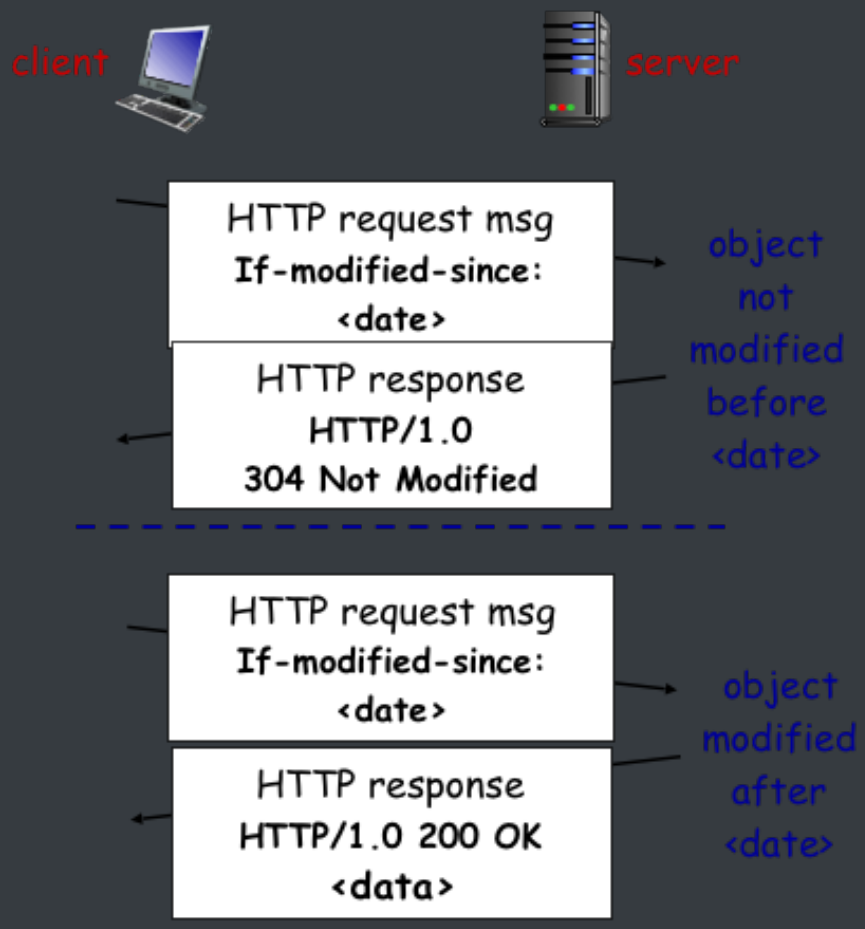
Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
-
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request

```
1  If-modified-since: <date>
```

- *server:* response contains no object if cached copy is up-to-date:

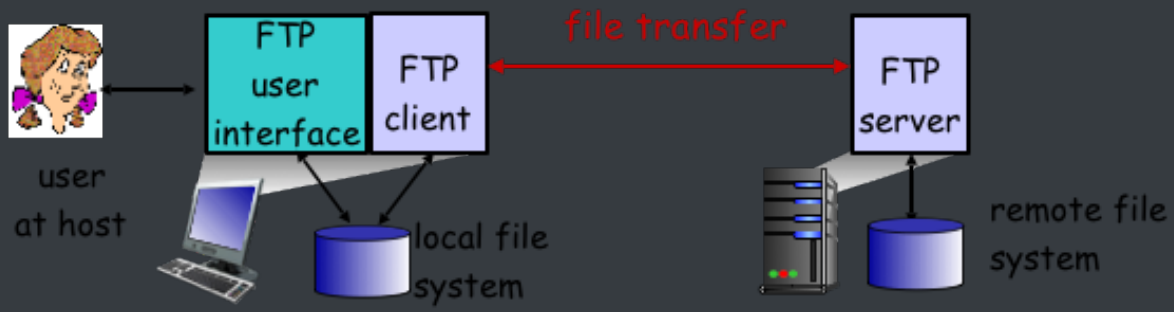
```
1  HTTP/1.0 304 Not Modified
```



2.3 FTP(Fire Transfer Protocol)

2.45

FTP: the file transfer protocol



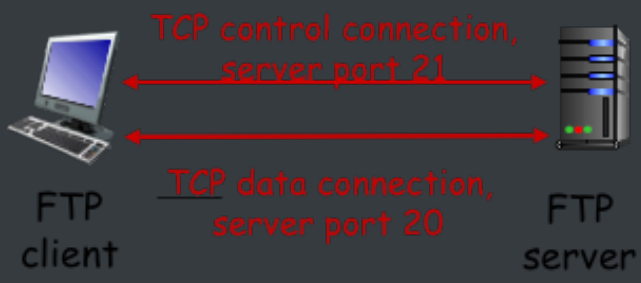
1. transfer file to/from remote host
2. client/server model [CS 模式]
3.
   - *client:* side that initiates transfer (either to/from remote)
   - *server:* remote host
4. ftp: RFC 959
5. ftp server: port 21

> 在一个典型的FTP会话中，用户坐在一台主机（本地主机）前面，向一台远程主机传输（或接收来自远程主机的）文件。为使用户能访问它的远程账户，用户必须提供一个用户标识和口令。在提供了这种授权信息后，用户就能从本地文件系统向远程主机文件系统传送文件，反之 亦然=如图2- 14所示，用户通过一个FT P用户代理与FTP交互 。该用户首先提供远程主机的主机名，使本地主机的FTP客户进程建立一个到远程主机FTP服务器进程的TCP连接。该用户接着提供用户标识和口令，作为FT P命令的一部分在该TCP连接上传送。一旦该服务器向该用户授权，用户可以将存放在本地文件系统中的一个或者多个文件复制到远程文件系统（反之亦然）。

2.46

FTP[File Transfer Protocol]: separate control, data connections

1. FTP client contacts FTP server at port 21, using TCP
2. client authorized over control connection
3. client browses remote directory, sends commands over control connection
4. when server receives file transfer command, *server* opens *2**nd* TCP data connection (for file) *to* client
5. after transferring one file, server closes data connection
6. server opens another TCP data connection to transfer another file
7. control connection: *“out of band”*
8. FTP server maintains “state”: current directory, earlier authentication



> HTTP和FTP都是文件传输协议，并且有很多共同的特点，例如，它们都运行在TCP 上、然而，这两个应用层协议也有一些重要的区别。其中最显著的就是FTP使用了两个并行的TC P连接来传输文件，一个是控制连接（control connection），一个是数据连接（data connection） 控制连接用于在两主机之间传输控制信息，如用户标识、口令、改变远程目 录的命令 以及 “存放 （put）"和“ 获取（get）”文件的命令。数据 连接用于实际发送一个 文件。 因为FT P协议使用一个 独立 的控制连接，所以我们也称FT P的控制信息是带外（ou t-o f-b and ）传送的。如你所知，HTT P协议是在传输文件的同一个TCP连接中发送请求和响应首部行的。因此，HTTP也可以说是带内（in-band）发送控制信息的、FTP协议的控制连接和数据连接如图2-15所示。

> 当用户主机与远程主机开始一个FTP会 话时，FTP的客户（用户）端首先在服务器 21号端口与服务器（远程主机）端发起一个 用于控制的TCP连接。FTP的客户端也通过该控制连接发送用户的标识和口令，发送改变远程目录的命令。当FTP的服务器端从该连接上收到一个文件传输的命令后（无论是向 还是来自远程主机），就发起一个到客户端的TCP数据连接。FTP在该数据连接上准确地传送一个文件，然后关闭该连接。在同一个会话期间，如果用户还需要传输另一个文件, FTP则打开另一个数据连接。因而对FTP传输而言，控制连接贯穿了整个用户会话期间，但是对会话中的每一次文件传输都需要建立一个新的数据连接（即数据连接是非持续的）。
>
> FTP服务器必须在整个会话期间保留用户的状态（state）。特别是，服务器必须把特定的用户账户与控制连接联系起来，随着用户在远程目录树上徘徊，服务器必须追踪用户在远程目录树上的当前位置，对每个进行中的用户会话的状态信息进行追踪，大大限制了FTP同时维持的会话总数。而另一方面，前面讲过HTTP是无状态的，即它不必对任何用户状态进行追踪。

2.47

FTP commands, responses

sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

> 我们通过简要地讨论几个常见的FTP命令和回答来结束本节。从客户到服务器的命令和从服务器到 客户 的回答， 都是以7比特ASC II格式 在控 制连接上传 送的 。因此 ， 与 HTTP协议的命令类似，FTP协议的命令也是人可读的。为了区分连续的命令，每个命令后跟回车换行符。每个命令由4个大写字母ASCII字符组成，有些还具有可选参数。一些较为常见的命令如下：・USER username：用于向服务器传送用户标识。・PASS password：用于向服务器发送用户口令、・UST：用于请求服务器回

> 送当前远程目录中的所有文件列表。该文件表是经一 个（新建且非持续连接）数据连接传送的，而不是在控制TCP连接上传送。• RETR filename；用于从远程主机当前目录检索（即gel）文件。该命令引起远程主机发起一个数据连接，并经该数据连接发送所请求的文件。• STOR filename：用于在远程主机的当前目录上存放（即put）文件。

sample return codes

- status code and phrase (as in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can’t open data connection**
- **452 Error writing file**

> 贯穿控制连接，在用户发出的命令和FTP发送的命令之间通常有一一对应关系。每个命令都对应着一个从服务器发向客户的回答。回答是一个3位的数字，后跟一个可选信 息。这与HTTP响应报文状态行的状态码和状态信息的结构相同。一些典型的回答连同它们可能的报文如下所示：• 331 Username OK, Password required （用户名 OK，需要口令）。• 125 Data connection already open； transfer starting （数据连接已经打开，开始传送）• 425 Can’t open data connection （无法打开数据连接）• 452 Error writing file （写文件差错）。 有兴趣学习其他FTP命令和回答的读者请阅读RFC 959。

2.4 electronic mail : SMTP, POP3, IMAP

> 自从有了因特网，电子邮件就在因特网上流行起来。当因特网还在襁褓中时，电子邮件已经成为最为流行的应用程序［Segaller l998］，年复一年，它变得越来越精细，越来越强大。它是当今因特网上最重要和实用的应用程序之一。 与普通邮件一样，电子邮件是一种异步通信媒介，即当人们方便时就可以收发邮件，不必与他人的计划进行协调。与普通邮件相比，电子邮件更为快速并且易于分发，而且价 格便宜。现代电子邮件具有许多强大的特性，包括具有附件、超链接、HTML格式文本和图片的报文。

> Web电子邮件 1995年12月，在Web “发 明”后 仅过了几年，Sabeer B hatia和Jack S mith拜访了因特网风险投资人Dr ap er Fisher Jurvetson,提出了研发一个免费的基于Web的电子邮件 系统的建议：其基本想法是为任何想要的人分配一个免费的电子邮件账户，并且使得这个账户可以在Web上使用。通过用公司的15%份额作为交换，Dra per Fi she r J urv ets on向 Bh at ia和Sm it h提供了资金，后者组建了一家公司，叫做Hotmail。3个全职员工和14个 兼职人员为了自己拥有的股份而工作，1996年7月，他们研发并提供了该服务，： 之后的一个月内，他们就拥有了 100 000名用户。199 7年12月，在启动该服务不到18个月内，Hotmail就拥有超过1200万个用户，并且以4亿美元的价格被微软公司收购，Hotmail公司的成功常被归结于它的“先行者优势（first-mover advantage）"和它固有的电子邮件“病毒行销（viral marketing） 策略，： （也许正在阅读本书的某些学生将会成为这种新人 之--------构思并开发具有“先行者优势”和“病毒行销”策略特征的因特网服务，） Web电子邮件继续兴盛，每年都变得更为复杂和功能强大。当今最为流行的服务之 一是谷歌的Gmail,它提供了千兆字节的免费存储、先进的垃圾邮件过滤和病毒检测、 电子邮件加密（使用SSL）、对第三方电子邮件服务的邮件接纳和面向搜索的界面。社 交网络如脸谱中的异步信息转发在近年来也已经变得流行。

2.49

Electronic mail

Three major components:

1. user agents
2. mail servers
3. simple mail transfer protocol: SMTP

User Agent

1. a.k.a. “mail reader”
2. composing, editing, reading mail messages
3. e.g., Outlook, Thunderbird, iPhone mail client

4. outgoing, incoming messages stored on server

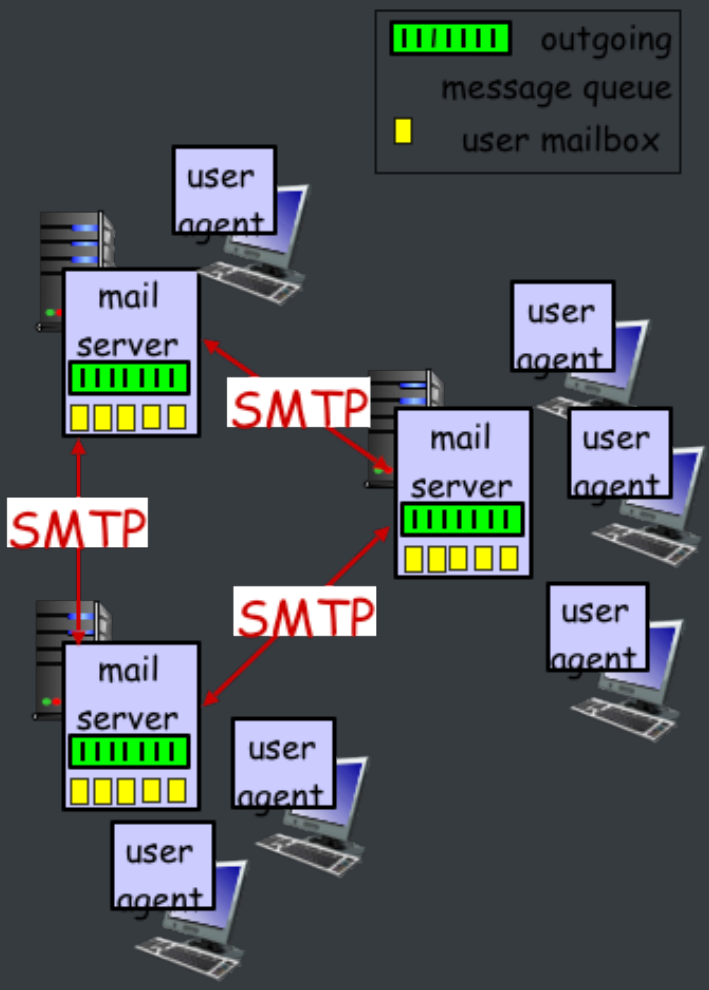


图2- 16给出了因特网电子邮件系统的总体情况。从该图中我们可以看到它有3个主 要组 成部分：用 户代理（use ragent）、邮件服务器 （mailse rv er ）和简单邮件传输协议（Si mp le M ail Tr an sf er P ro to co l, S MT P） o下面我们结合发送方 Al ic e发电子邮件给接收方 Bob的例子，对每个组成部分进行描述。用户代理允许用户阅读、回复、转发、保存和撰写报文 微软的Ou tl oo k和Ap pl e Ma il是电子邮件用户代理的例子。当Al ic e完成邮件撰写 时.她的邮 件代 理向其 邮件服务器 发送邮件，此时邮件放在邮件服务器的外出报文队列中。邮件服务器形成了电子邮件体系结构的核心。每个接收方（如Bob）在其中的某个邮件服务器上有一个邮箱（mailbox）。Bob的邮箱管理和维护着发送给他的报文。一个典型的邮件发送过程是：从发送方的用户代理开始，传输到发送方的邮件服务器，再传输到接收方的邮件服务器，然后在这里被分发到接收方的邮箱中。当Boh要在他的邮箱中读取该报文时，包含他邮箱的邮件服务器（使用用户名和口令）来鉴别Bob。Alice的邮箱也必须能处理Bob 的邮件服务器的故障 如果Alice的服务器不能将邮件交付给Bob的服务器，Al ic e的邮件服务器在一个报文队列（message queue）中保持该报文并在以后尝试再次发 送。通常每30分钟左右进行一次尝试；如果几天后仍不能成功，眼务器

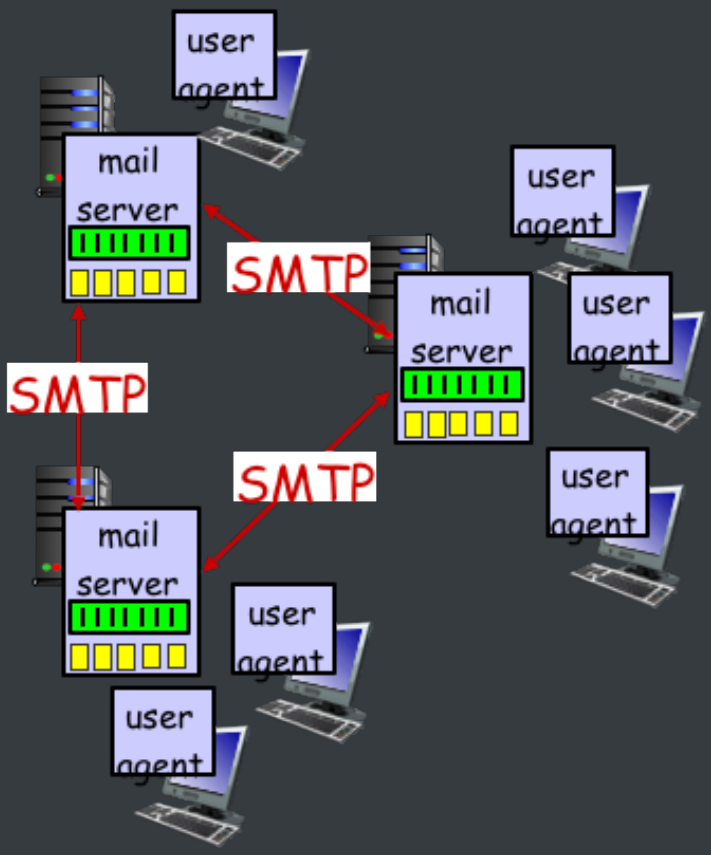
> 就删除该报文并以电子邮件的形式通知发送方（Alice），

2.50

Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
-
    - client: sending mail server
    - “server”: receiving mail server



> SMT P是因特网电子邮件中主要的应用层协议。它使用TCP可靠数据传输服务，从发 送方的邮件服务器向接收方的邮件服务器发送邮件。像大多数应用层协议一样，SMTP也 有两个部分：运行在发送方邮件服务器的客户端和运行在接收方邮件服务器的服务器端。
>
> 每台邮件服务器上既运行SMT P的客户端也运行SMT P的服务器端。当一个邮件服务器向 其他邮件服务器发送邮件时，它就表现为SMTP的客户；当邮件服务器从其他邮件服务器上接收邮件时，它就表现为一个SMTP的服务器

2.51

Electronic Mail: SMTP [RFC 2821]

1. uses TCP to reliably transfer email message from client to server, port 25
2. direct transfer: sending server to receiving server

3. three phases of transfer
4. - handshaking (greeting)
   - transfer of messages
   - closures
5. command/response interaction (like HTTP, FTP)
6. - commands: ASCII text
   - response: status code and phrase
7. messages must be in 7-bit ASCI

> RF C 53 21给出了 SM TP的定义。SM TP是因特网电子邮件应用的核心，如前所述，SMTP用于从发送方的邮件服务器发送报文到接收方的邮件服务器。SMT P问世的时间比HT TP要长得多（初始的SM TP协议的RF C可追溯到19 82年，而SM TP在此之前很长一段时间就已经出现了）。尽管电子邮件应用在因特网上的独特地位可以证明SMTP有着众多非常出色的性质，但它所具有的某种陈旧特征表明它仍然是一种继承的技术。例如，它限制所有邮件报文的体部分（不只是其首部）只能采用简单的7比特AS CI I表示。在20世 纪80年代早期，这种限制是明智的，因为当时传输能力不足，没有人会通过电子邮件发 送大的附件或是大的图片、声音或者视频文件。然而，在今天的多媒体时代，7位ASC II 的限 制的确 有点痛苦，即在用SM TP传送邮件之 前，需要将二进制多媒体数据编码为 AS CI I码，并且在使用SM TP传输后要求将相应的ASCII码邮件解码还原为多媒体数据。 2.2节讲过，使用HTTP传送前不需要将多媒体数据编码为ASCII码。

2.52

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message “to” bob@someschool.edu

> @ is the sign that never exists in person's name

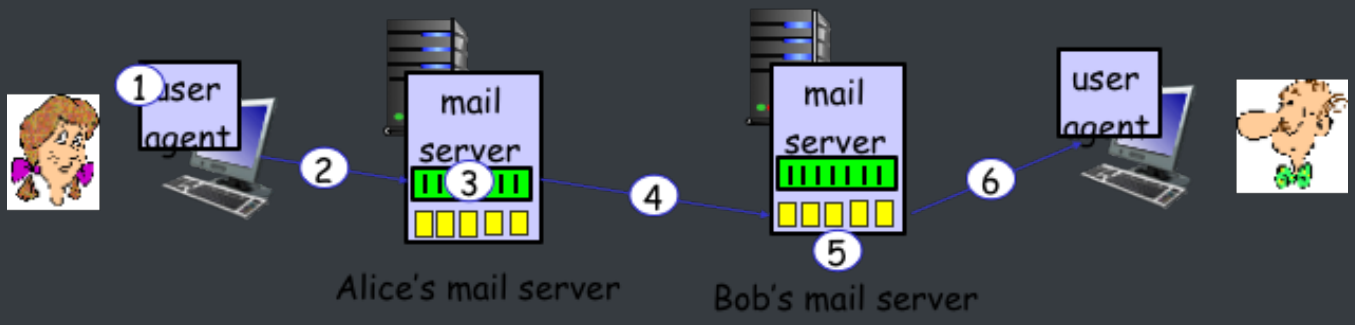
2) Alice’s UA sends message to her mail server; message placed in message queue

3) client side of SMTP opens TCP connection with Bob's mail server

4) SMTP client sends Alice's message over the TCP connection

5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his user agent to read message



> 为了 描述SM TP的基本操作.我们观察一种常见的情景。假设Al ic e想给Bo b发送一 封简单的ASCII报文-·Alice调用她的邮件代理程序并提供Bob的邮件地址（例如bob@ som eschool . edu）， 撰写报文.然后指示用户代理发送该报文。·Alire的用户代理把报文发给她的邮件服务器，在那里该报文被放在报文队列中，·运行在Al ic e的邮件服务器 上的SM TP客户端发现了报文队列中的这个报文，它就 创建一个到运行在Bob的邮件服务器上的SMTP服务器的TCP连接。·在经过一些初始SMTP握手后，SMTP客户通过该TCP连接发送Alice的报文。 ·在Boh的邮件服务器上，SMTP的服务器端接收该报文。Bob的邮件服务器然后将 该报文放人Bob的邮箱中-·在Boh方便的时候，他调用用户代理阅读该报文。 图2-17总结了上述这个情况。 观察到下述现象是重要的：SMTP 一般不使用中间邮件服务器发送邮件，即使这两个 邮件服务器位于地球的两端也是这样。假设Alice的邮件服务器在中国香港，而Bob的服务器在美国圣路易斯，那么这个TCP连接也是从香港服务器到圣路易斯服务器之间的直接 相连。特别是，如果Bob的邮件服务器没有开机，该报文会保留在Ali ce的邮件服务器上 并等待进行新的尝试，这意味着邮件并不在中间的某个邮件服务器存留。

2.53

Sample SMTP interaction

```
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
```

> 我们现在仔细观察一下，SMTP是如何将一个报文从发送邮件服务器传送到接收邮件 服务器的。我们将看到，SMTP与人类面对面交往的行为方式有许多类似性、首先，客户 SMT P （运行在发送邮件服务器上）在25号端口建立一个到服务器SMTP（运行在接收邮 件服务器上）的TCP连接。如果服务器没有开机，客户会在稍后继续尝试连接。一旦连 接建立，服务器和客户执行某些应用层的握手，就像人们在互相交流前先进行自我介绍一 样，SMT P的客户和服务器在传输信息前先相互介绍。在SMTP握手的阶段，SMTP客户指 示发送方的邮件地址（产生报文的那个人）和接收方的邮件地址。一旦该SMTP客户和服 务器彼此介绍之后，客户发送该报文。SMTP能依赖TCP提供的可靠数据传输无差错地将 邮件投递到接收服务器。该客户如果有另外的报文要发送到该服务器，就在该相同的TCP 连接上重复这种处理；否则，它指示TCP关闭连接。 接下来我们分析一个在SMTP客户（C）和SMTP服务器（S）之间交换报文脚本的例 子。客户 的主机名为 crepes, fr,服务器 的主机名为 hambur ge r. e du o以C:开头的AS CI I码 文本行正是客户交给其TCP套接字的那些行，以S:开头的ASCII码则是服务器发送给其 TCP套接字的那些行。一旦创建了 TCP连接，就开始了下列过程

> 在上例中，客户从邮件服务器crepes, fr向邮件服务器hamburger, ed u发送了一个报文 （**D o y ou lik e k etc hup ? H ow abo ut pic kle s?" ） o 作为对话的一部分，该客户发送了 5 条命 令：HELO （是 HELLO 的缩写）、MAIL FROM，RCPTTO、DATA 以及 QUIT。这些命令都 是自解释的。该客户通过发送一个只包含一个句点的行，向服务器指示该报文结束了。 220 hamburger.edu HELO crepes.fr 250 Hello crepes.fr, pleased to meet you MAIL FROM: <alicegcrepes.fr> 250 alice@crepes.fr ... Sender ok RCPT TO: <bob0hamburger.edu> 250 bob®hamburger.edu ... Recipient ok DATA 354 Enter mail, end with "・"on a line by itself Do you like ketchup? How about pickles? （按照ASCII码的表示方法，每个报文以CRLF. CRLF结束，其中的CR和LF分别表示回 车和换行。）服务器对每条命令做出回答，其中每个回答含有一个回答码和一些（可选的）英文解释，我们在这里指出SMTP用的是持续连接：如果发送邮件服务器有几个报文 发往同一个接收邮件服务器，它可以通过同一个TCP连接发送这些所有的报文。对每个报 文，该客户用一个新的MAI L F ROM： cr epe s, fr开始，用一个独立的句点指示该邮件的结 束，并且仅当所有邮件发送完后才发送QUIT.

2.54

Try SMTP interaction for yourself:

1. '''shell telnet servername 25 '''
2. see 220 reply from server
3. enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

> 我们强烈推荐你使用Telnet与一个SMTP服务器进行一次直接对话。使用的命令是
>
> '''shell telnet serverName 25 '''

> 其中serverName是本地邮件服务器的名称。当你这么做时，就直接在本地主机与邮件服务 器之间建立一个TC P连接。输完上述命令后，你立即会从该服务器收到22 0回答。接下来，在适当的时机发出 HEL O、MAI L F ROM, RCP T T O、DAT A、CRL F. CRL F 以及 QUI T 等SM TP命令：强烈推荐你做本章后面的编程作业3。在该作业中，你将在SMTP的客户端实现一个简单的用户代理，它允许你经本地邮件服务器向任意的接收方发送电子邮件报文。

2.55

SMTP: final words

1. SMTP uses persistent connections
2. SMTP requires message (header & body) to be in 7-bit ASCII
3. SMTP server uses CRLF.CRLF to determine end of message

comparison with HTTP:

1. HTTP: pull
2. SMTP: push
3. both have ASCII command/response interaction, status codes
4. HTTP: each object encapsulated in its own response msg
5. SMTP: multiple objects sent in multipart msg

> 我们简要地比较一下SM TP和HT TP .这两个 协议都用于从 一台 主机向另一台主机传 送文件：HTTP从Web服务器向Web客户（通常是一个 浏览器）传送文件（也称为对象）；SMTP从一个邮件服务器向另一个邮件服务器传送文件（即电子邮件报文）。当进行文件传送时，持续的HTTP和SMTP都使用持续连接。因此，这两个协议有一些共同特征。 然而，两者之间也有一些重要的区别。首先，HT TP主要是一个拉协议（pull protocol), 即在方便的时候，某些人在Web服务器上装载信息，用户使用HTTP从该服务器拉取这些信息。特别是TCP连接是由想接收文件的机器发起的。另一方面，SMTP基本上是一个推协议（push protocol）,即发送邮件服务器把文件推向接收邮件服务器。特别是，这个TCP 连接是由要发送该文件的机器发起的。

> 第二个区别就是我们前面间接地提到过的，SMTP要求每个报文（包括它们的体）使 用7比特ASCII码格式 。如果 某报 文包含 了非7比特ASC II字符（ 如具有重音的法文字符）或二进制数据（如图形文件），则该报文必须按照7比特ASCII码进行编码、HTTP数据则不受这种限制。
>
> 第三个重要区别是如何处理一个既包含文本又包含图形（也可能是其他媒体类型）的文档。如我们在2. 2节知道的那样，HTTP把每个对象封装到它自己的HTTP响应报文中，而SMTP则把所有报文对象放在一个报文之中:

2.56

Mail message format

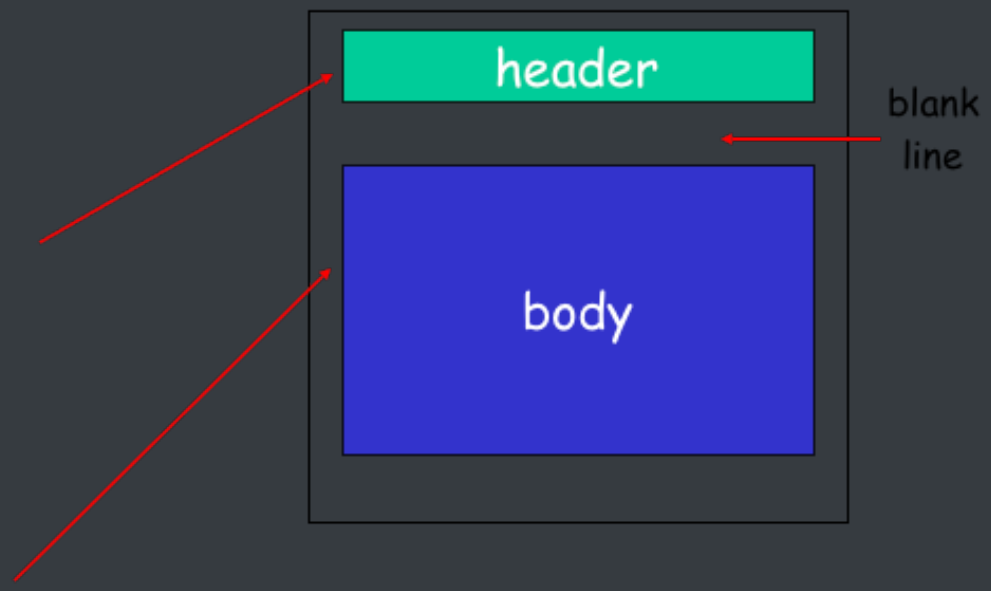
SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

- header lines, e.g.,
    - To:
    - From:
    - Subject:

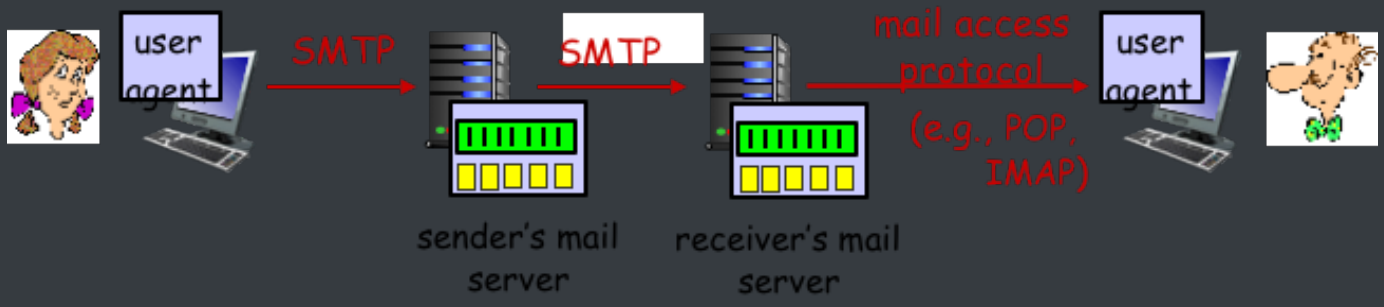
different from SMTP MAIL FROM, RCPT TO: commands!

- Body: the “message”
    - ASCII characters only



2.57

Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
-
    - POP: Post Office Protocol [RFC 1939]: authorization, download
    - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
    - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

> 当Alice给Bob写一封邮寄时间很长的普通信件时，她可能要在信的上部包含各种各 样的环境首部信息，如Bob的地址、她自己的回复地址以及日期等。同样，当一个人给另一个人发送电子邮件时，一个包含环境信息的首部位于报文体前面。这些环境信息包括在一系列首部行中，这些行由RFC 532 2定义。首部行和该报文的体用空行（即回车换行）进行分隔。RFC 5322定义了邮件首部行和它们的语义解释的精确格式。如同HTTP协议，每个首部行包含了可读的文本，是由关键词后跟冒号及其值组成的。某些关键词是必需的，另一 些则是可选的。每个首部必须含有一个Fr om :首部行和一 个To :首部行；一个 首部也许包含一个Subject：首部行以及其他可选的首部行。注意到下列事实是重要的：这些首部行不同 于我们在2. 4 . 1节所学到的SMTP命令（即使那里也包含了某些相同的词汇，如fr om和to ）. .那节中的命令是SMTP握手协议的一部分；本节中研究的首部行则是邮件报文自身的一部分。 一个典型的报文首部看起来如下：
>
> '''html From: alice@crepes.fr To: bob@hamburger.edu Subject: Searching for the meaning of life. '''
>
> 在报文首部之后，紧接着一个空白行，然后是以ACSII格式表示的报文体你应当用 Te ln et向邮件服务器发送包含一些首部行的报文，包括Subject：首部行„为此，输人命令 telnet serverName 25 ,如在2. 4. 1节中讨论的那样.

2.58

POP3 protocol

authorization phase

- client commands:
    - **user:** declare username
    - **pass:** password
- server responses

- - **+OK**
  - **-ERR**

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

transaction phase, client:

- **list:** list message numbers
- **retr:** retrieve message by number
- **dele:** delete
- **quit**

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

2.59

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 “download and delete” mode
- - Bob cannot re-read e-mail if he changes client
- POP3 “download-and-keep”: copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
-
    - names of folders and mappings between message IDs and folder name

2.5 DNS

2.61

DNS: domain name system

people: many identifiers:

-
    - SSN, name, passport #

Internet hosts, routers:

-
    - IP address (32 bit) - used for addressing datagrams
    - “name”, e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*

- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
-
    - note: core Internet function, implemented as application-layer protocol
    - complexity at network's "edge"

2.62

DNS: services, structure

DNS services

- hostname to IP address translation
- host aliasing
-
    - canonical, alias names
- mail server aliasing
- load distribution
-
    - replicated Web servers: many IP addresses correspond to one name

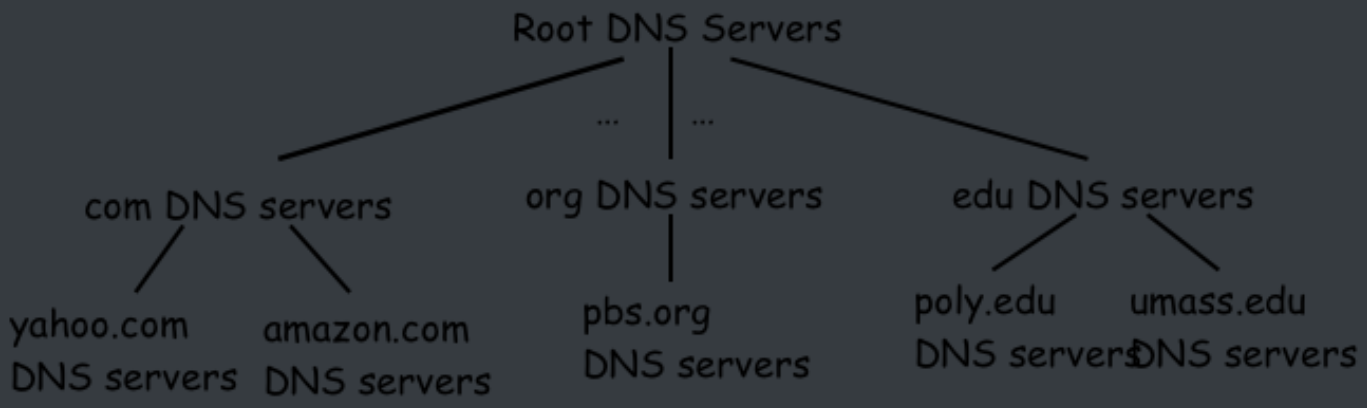
why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

2.63

DNS: a distributed, hierarchical database



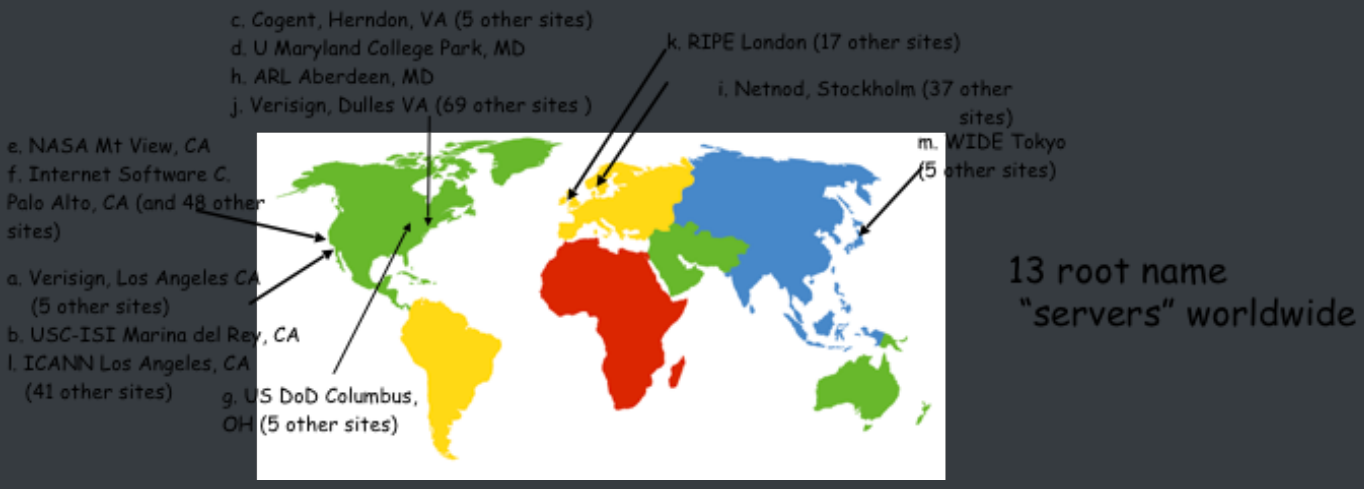
*client wants IP for www.amazon.com; 1**st approx:*

- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

2.64

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
-
    - contacts authoritative name server if name mapping not known
    - gets mapping
    - returns mapping to local name server



2.65

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

2.66

Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one

- also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
- has local cache of recent name-to-address translation pairs (but may be out of date!)
- acts as proxy, forwards query into hierarchy

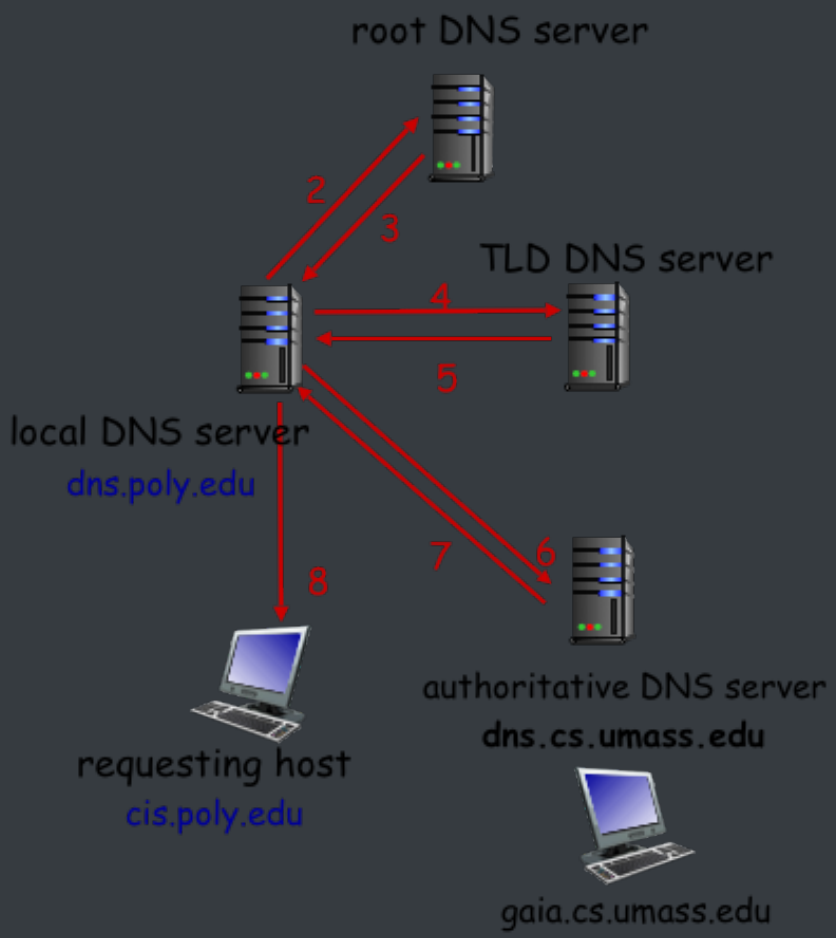
2.67

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 to contact
- “I don’t know this name, but ask this server”

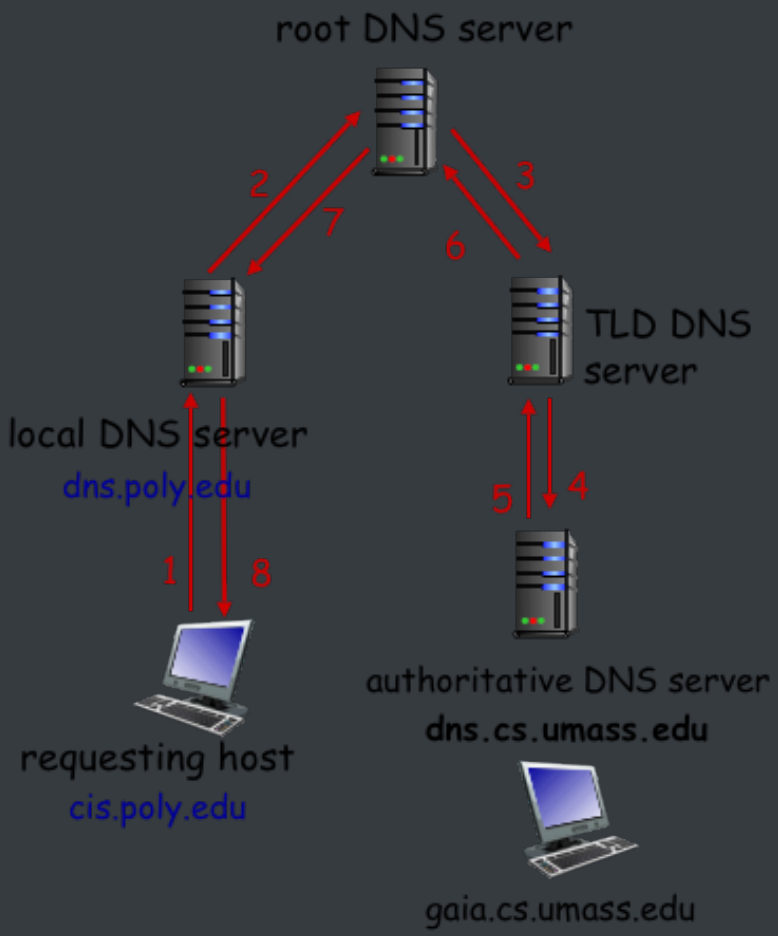


2.68

DNS name resolution example

recursive query:

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



2.69

DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
- - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
- - - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
- - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
- - RFC 2136

2.70

DNS records

DNS: distributed db storing resource records (RR)

$$RR\ format: (name, value, type, ttl) \tag{2}$$

type=A

- - **name** is hostname
  - **value** is IP address

type=NS

- - **name** is domain (e.g., foo.com)
  - **value** is hostname of authoritative name server for this domain

type=CNAME

- - **name** is alias name for some “canonical” (the real) name
  - **www.ibm.com** is really **servereast.backup2.ibm.com**
- - **value** is canonical name

type=MX

- - **value** is name of mailserver associated with **name**

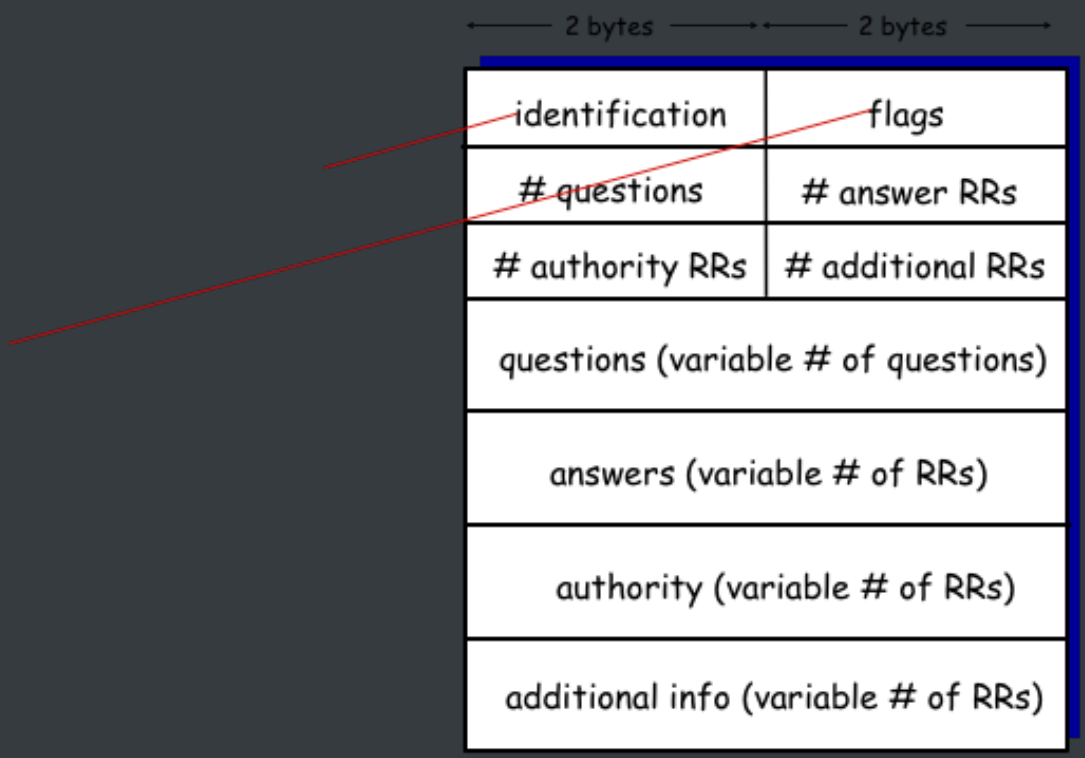
2.71

DNS protocol, messages

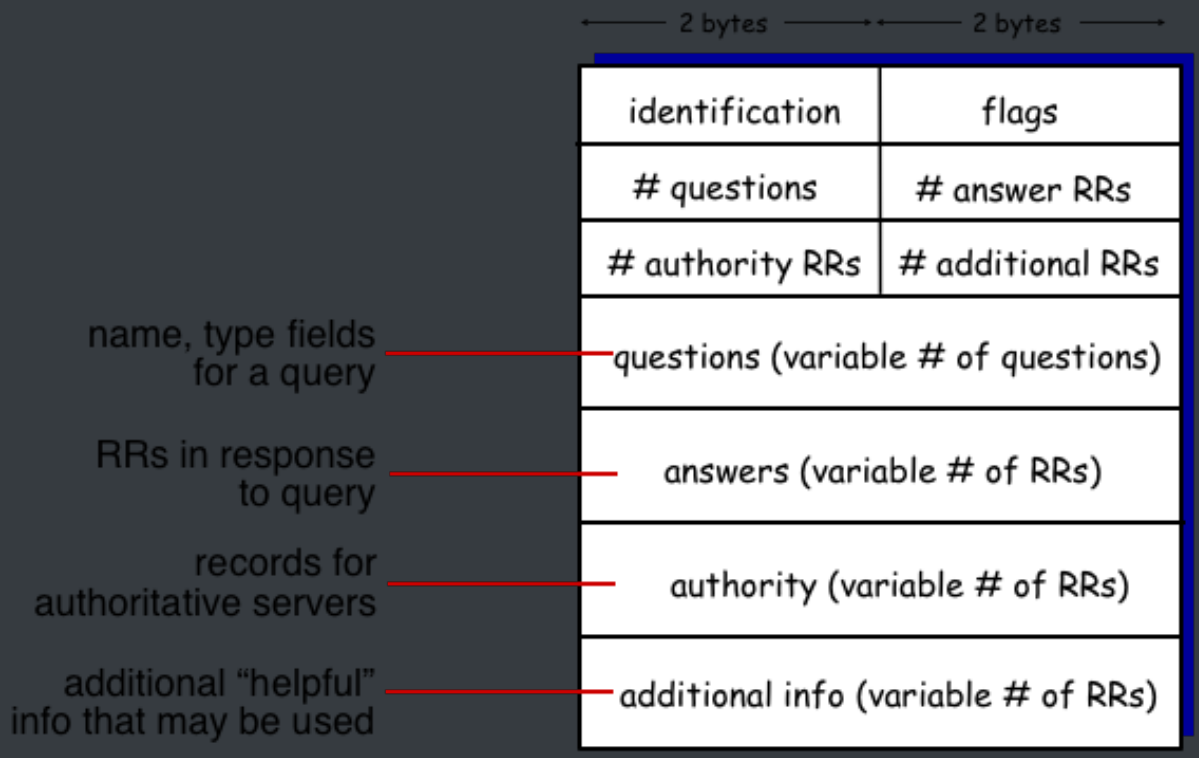
- *query* and *reply* messages, both with same *message format*

msg header

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



2.72



2.73

Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
-
    - provide names, IP addresses of authoritative name server (primary and secondary)
    - registrar inserts two RRs into .com TLD server: **(networkutopia.com, dns1.networkutopia.com, NS)**

(dns1.networkutopia.com, 212.212.212.1, A)

- create authoritative server type A record for www.networkuptopia.com; type MX record for networkutopia.com

2.74

Attacking DNS

DDoS attacks

- Bombard root servers with traffic
- - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
- - Potentially more dangerous

Redirect attacks

- Man-in-middle
- - Intercept queries
- DNS poisoning
- - Send bogus relies to DNS server, which caches

Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

2.6 P2P applications

2.76

Pure P2P architecture

1. [no always-on] *no* always-on server
2. [end-systems driectly] arbitrary end systems directly communicate
3. [intermittently, change IP] peers are intermittently connected and change IP

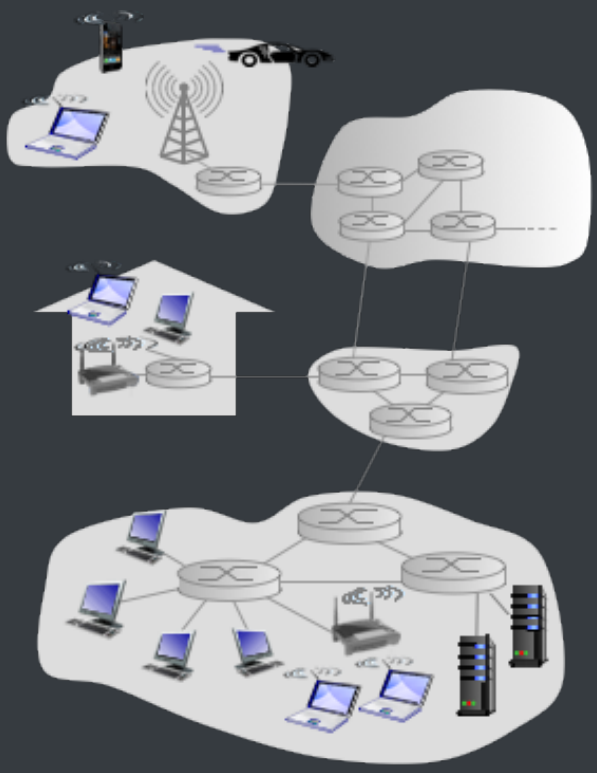
addresses

examples:

- file distribution (BitTorrent)

  Streaming (KanKan)

  VoIP (Skype)

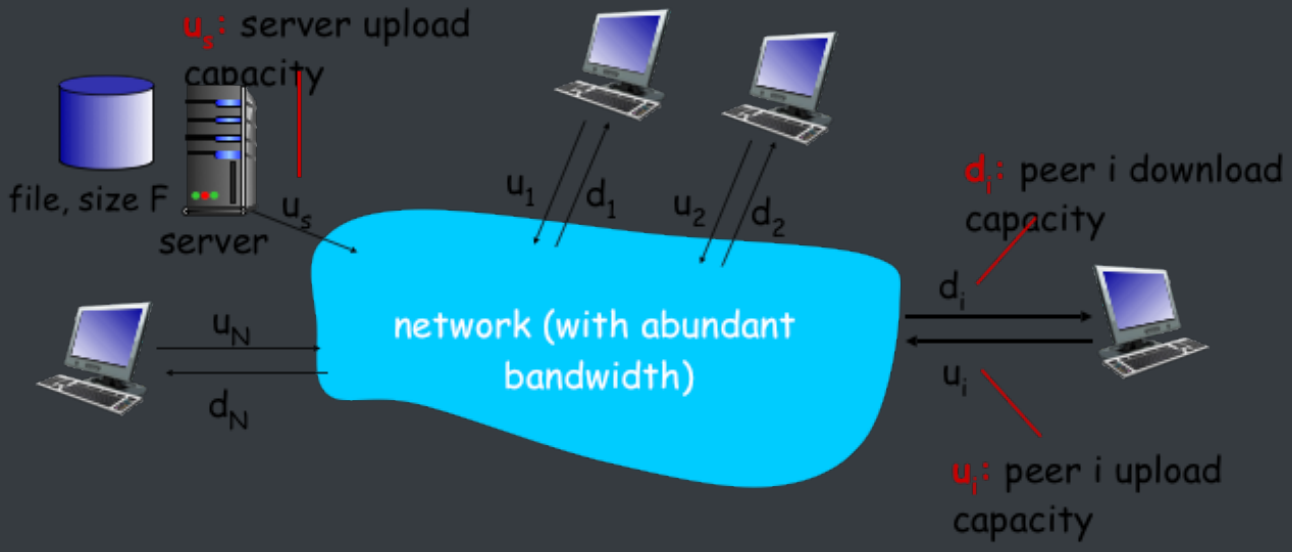


2.77

File distribution: client-server vs P2P

*Question**:* how much time to distribute file (size *F*) from one server to *N peers*?

- peer upload/download capacity is limited resource



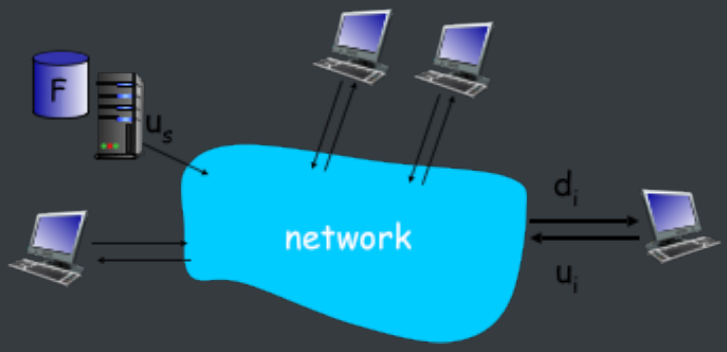
2.78

File distribution time: client-server

- *server transmission:* must sequentially send (upload) *N* file copies:
- - time to send one copy: *F/u**s*
  - time to send N copies: *NF/u**s*
- *client:* each client must download file copy
- - d_{min} = min client download rate
  - min client download time: $F/dmin$

time to distribute F to N clients using client-server approach

$$D_{c-s} \geq max\{N\frac{F}{u_s}, \frac{F}{d_{min}}\} \quad (3)$$



2.79

File distribution time: P2P

- *server transmission:* must upload at least one copy
- - time to send one copy: *F/u**s*
- *client:* each client must download file copy
- - min client download time: F/dmin
- *clients:* as aggregate must download *NF* bits
- - max upload rate (limting max download rate) is us + Σui

time to distribute F to N clients using P2P approach

$$D_{P2P} > max\{\frac{F}{u_s}, \frac{F}{d_{min}}, N\frac{F}{(u_s + \sum_{i=1}^{N} u_i)}\} \qquad (4)$$

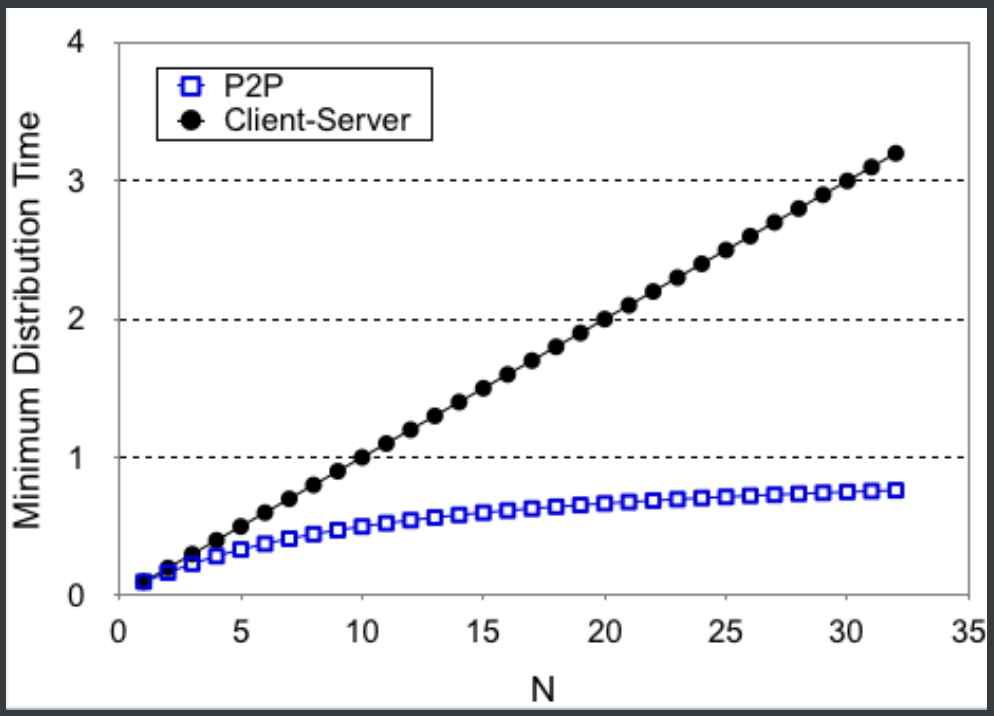
increases linearly in N …

… but so does this, as each peer brings service capacity

2.80

Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, us = 10u, dmin ≥ us



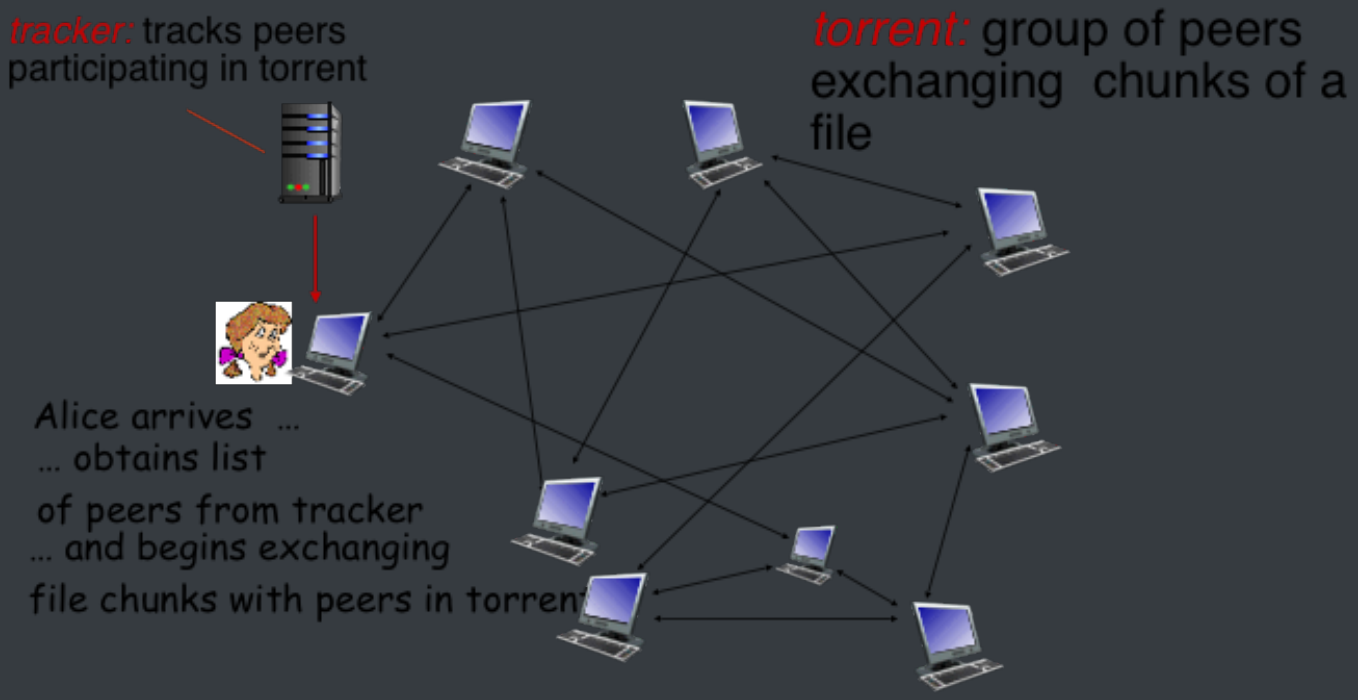
2.81

P2P file distribution: BitTorrent

- file divided into 256Kb chunks[片]
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

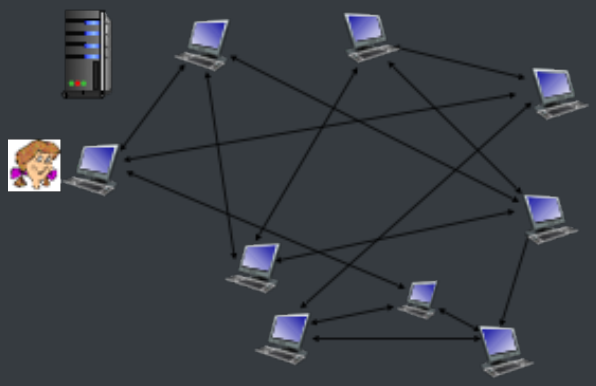
torrent: group of peers exchanging chunks of a file



2.82

P2P file distribution: BitTorrent

- peer joining torrent:
-
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

2.83

BitTorrent: requesting, sending file chunks

requesting chunks[厚片]:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
-
    - other peers are choked by Alice (do not receive chunks from her)
    - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
-
    - “optimistically unchoke” this peer
    - newly chosen peer may join top 4

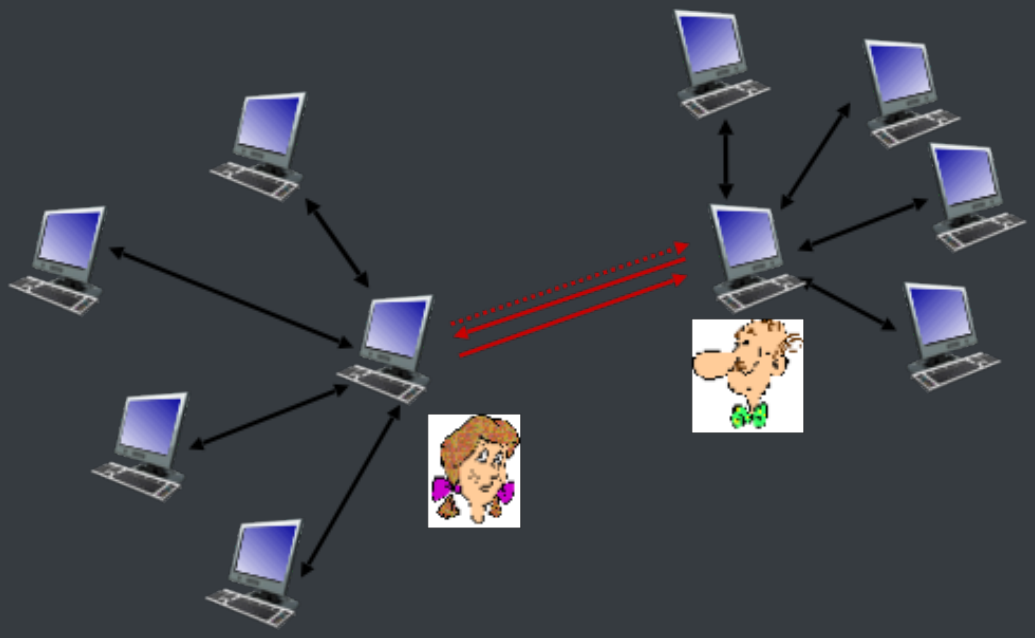
2.84

BitTorrent: tit-for-tat

(1) Alice “optimistically unchokes” Bob

(2) Alice becomes one of Bob’s top-four providers; Bob reciprocates

(3) Bob becomes one of Alice’s top-four providers



higher upload rate: find better trading partners, get file faster !

2.85

⭐Distributed Hash Table (DHT) [分布式哈希表]

1. Hash table
2. DHT paradigm[范例]
3. Circular DHT and overlay networks

4. Peer churn

2.86

Simple Database

Simple database with(key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	…….
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

2.87

Hash Table

- More convenient to store and search on numerical representation of key
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		……
Lisa Kobayashi	9290124	177-23-0199

2.88

Distributed Hash Table (DHT)

- Distribute (key, value) pairs over millions of peers

  pairs are evenly distributed over peers
- Any peer can query database with a key

  database returns value for the key

  To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

2.89

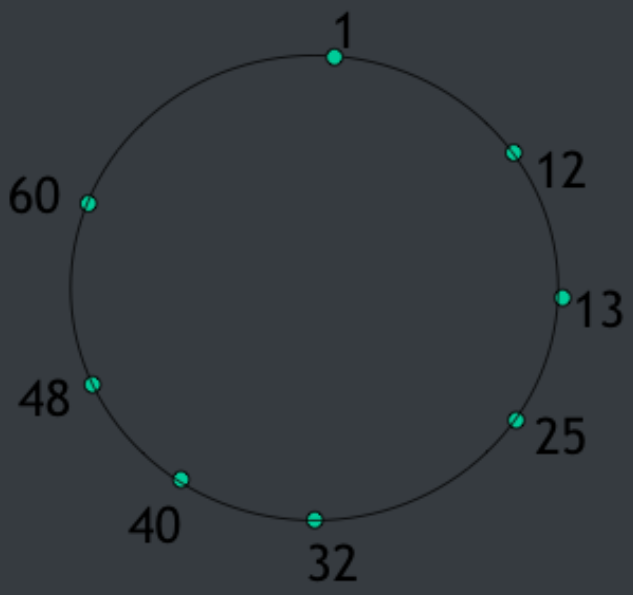
Assign key-value pairs to peers

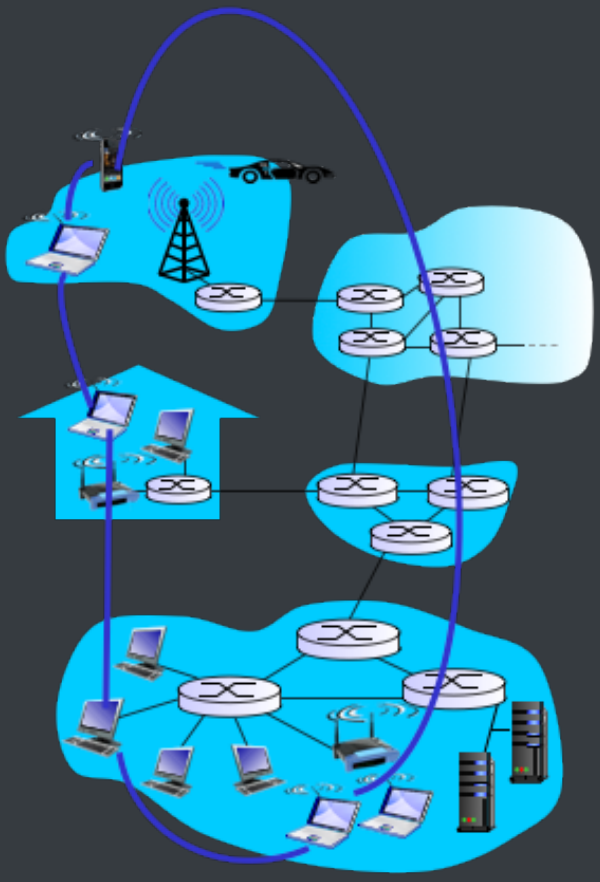
- rule: assign key-value pair to the peer that has the *closest* ID.
- convention: closest is the *immediate successor* of the key.
- e.g., ID space {0,1,2,3,…,63}
- suppose 8 peers: 1,12,13,25,32,40,48,60
-
    - If key = 51, then assigned to peer 60
    - If key = 60, then assigned to peer 60
    - If key = 61, then assigned to peer 1

2.90

Circular DHT

- each peer *only* aware of immediate successor and predecessor.





2.91

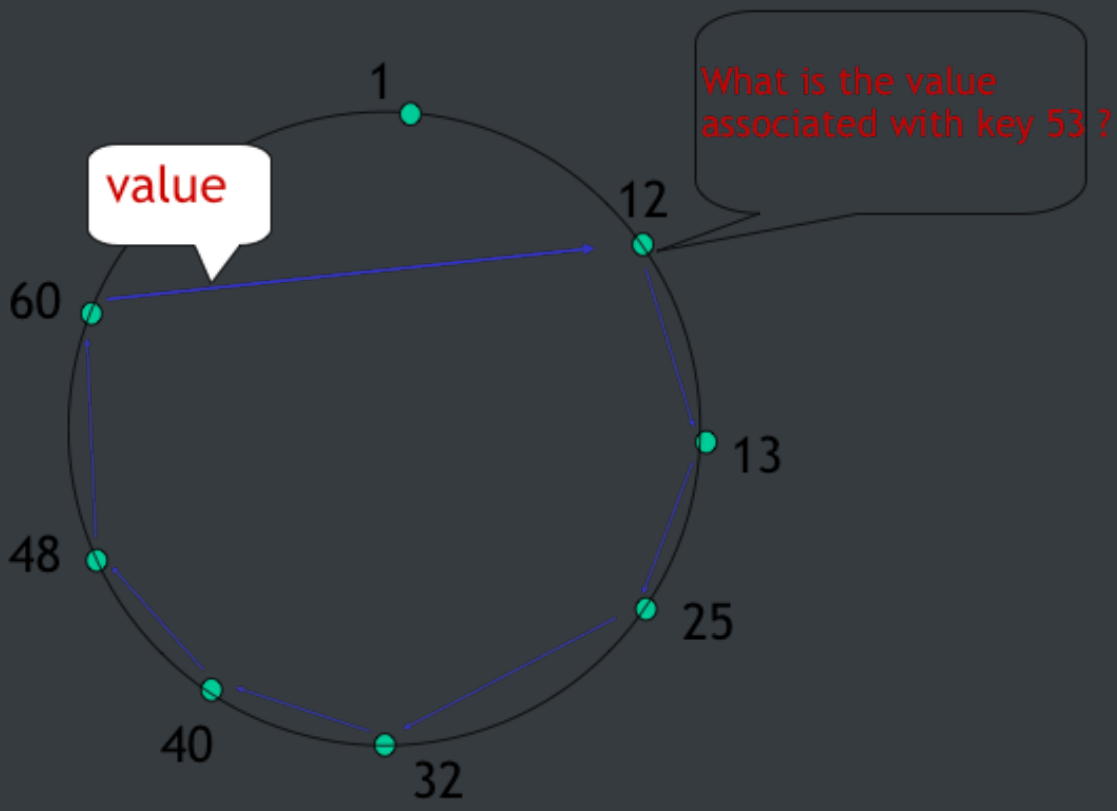
Resolving a query

O(N) messages

on avgerage to resolve

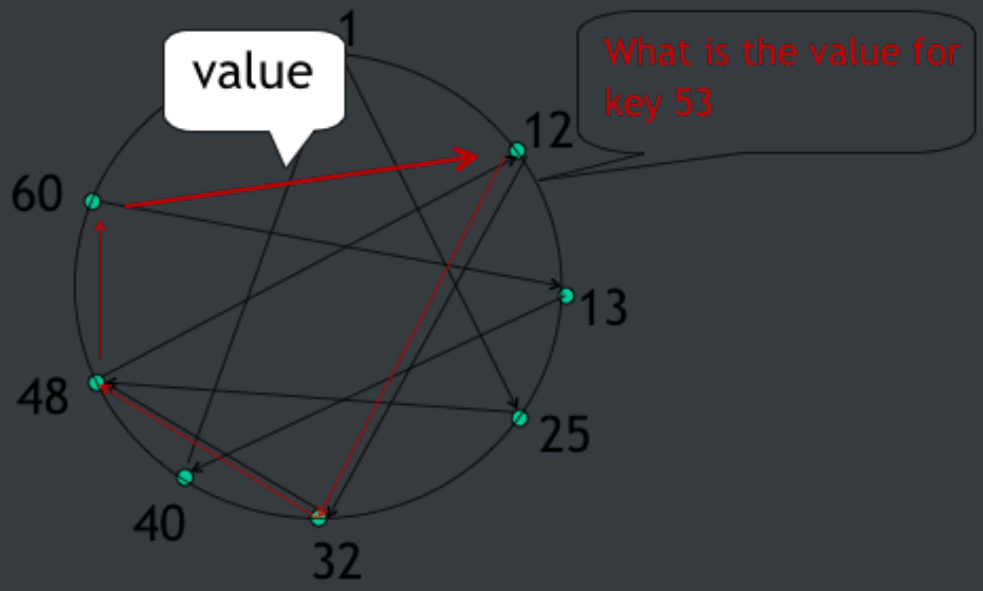
query, when there

are *N* peers



2.92

Circular DHT(Distributed Hash Table) with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.
- possible to design shortcuts with *O(log N)* neighbors, *O(log N)* messages in query

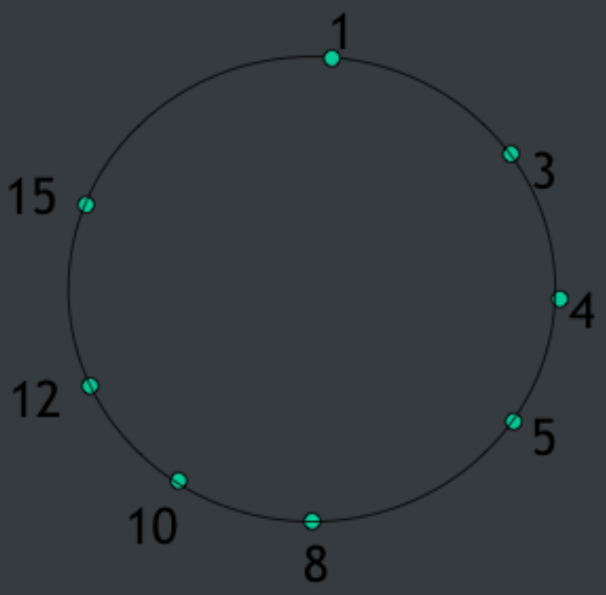
2.93

Peer churn

handling peer churn:

1. peers may come and go (churn)
2. each peer knows address of its two successors
3. each peer periodically pings its two successors to check aliveness
4. if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves



2.94

Peer churn

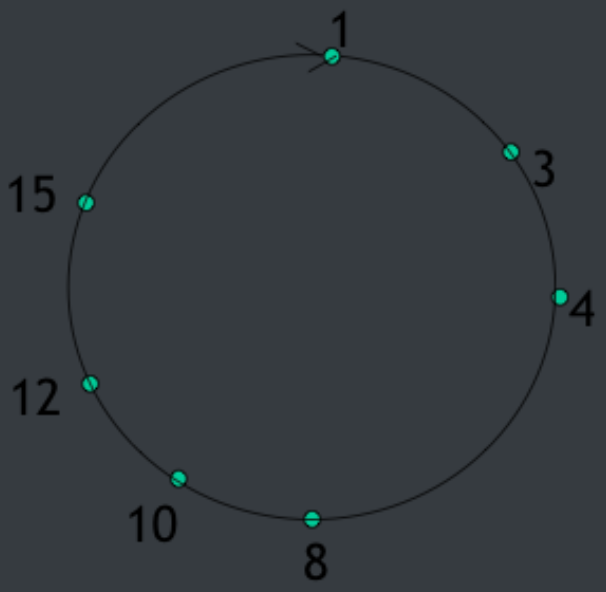
handling peer churn:

- peers may come and go (churn)

- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

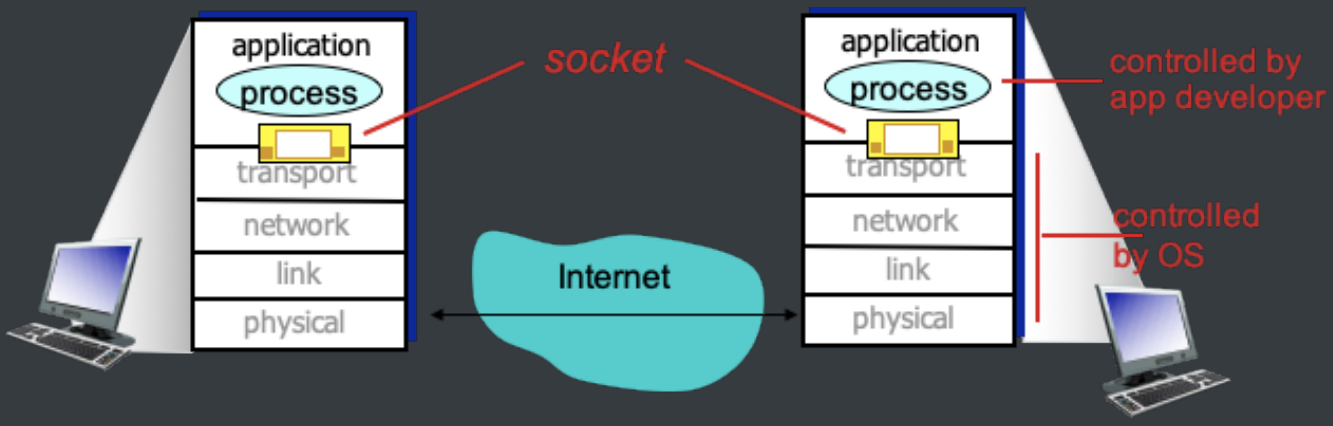


2.7 socket programming with UDP and TCP

2.96

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



2.97

Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the

server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

2.98

Socket programming *with UDP*

UDP: no “connection” between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

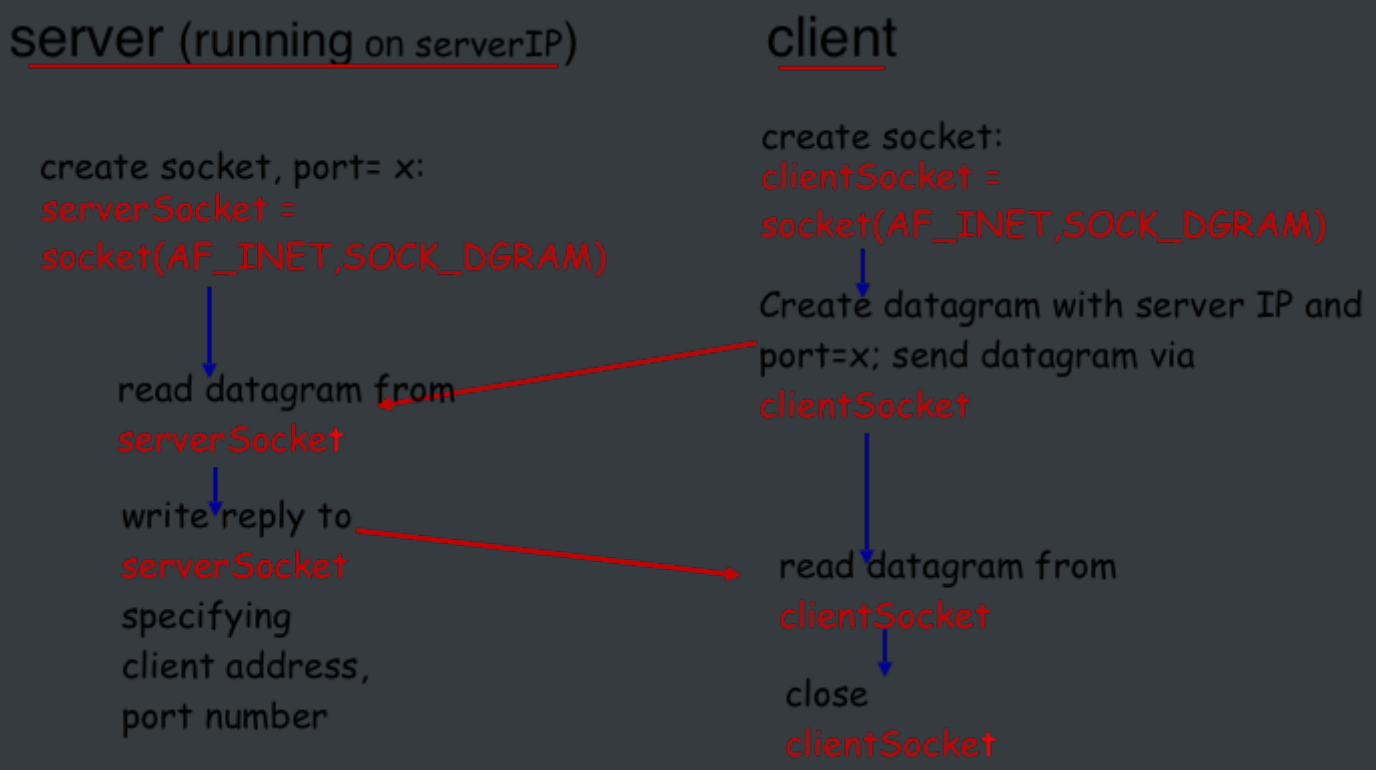
UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server

2.99

Client/server socket interaction: UDP



Example app: Python UDP client

```
1  from socket import *
2  # include Python's socket library
3  serverName = 'hostname'
4  serverPort = 12000
5  clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
6  # create UDP socket for server
7  message = raw_input('Input lowercase sentence:')
8  # get user keyboard input
9  clientSocket.sendto(message,(serverName, serverPort))
10 # Attach server name, port to message; send into socket
11 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
12 # read reply characters from socket into string
13 print modifiedMessage
```

```
14 # print out received string and close socket
15 clientSocket.close()
16
```

2.101

Python UDPServer

```
1  from socket import *
2  serverPort = 12000
3  serverSocket = socket(AF_INET, SOCK_DGRAM)
4  # create UDP socket
5  serverSocket.bind(('', serverPort))
6  # bind socket to local port number 12000
7  print “The server is ready to receive”
8  while 1:
9  # loop forever
10     message, clientAddress = serverSocket.recvfrom(2048)
11     # Read from UDP socket into message, getting client’s address
   (client IP and port)
12
13     modifiedMessage = message.upper()
14     serverSocket.sendto(modifiedMessage, clientAddress)
15     # send upper case string back to this client
16
```

2.102

Socket programming with TCP

1. client must contact server

- [server, first running] server process must first be running
- [server, created socket(door)] server must have created socket (door) that welcomes client's contact

2. client contacts server by:

- [client creates TCP socket, specifies IP, port number] Creating TCP socket, specifying IP address, port number of server process
- [client TCP connects to server TCP] *when client creates socket:* client TCP establishes connection to server TCP

when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client

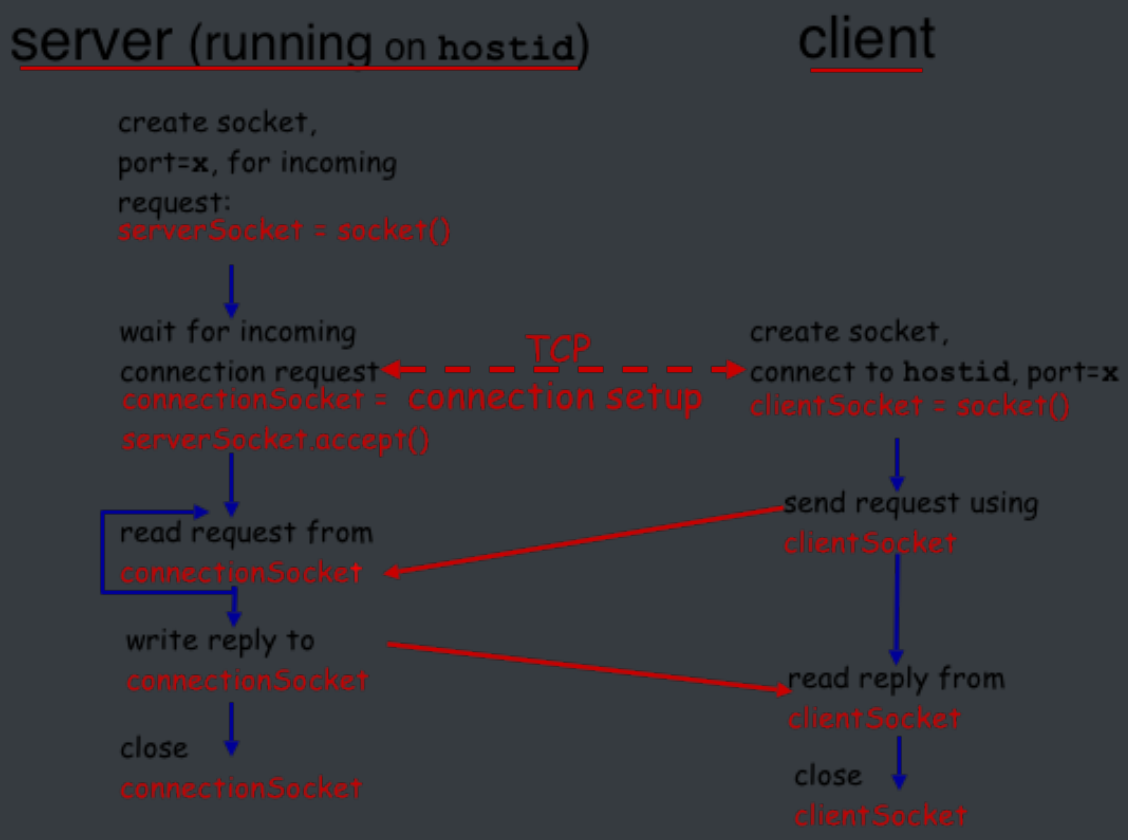
- allows server to talk with multiple clients
- source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

2.103

Client/server socket interaction: TCP



2.104

Python TCP Client

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
# create TCP socket for server, remote port 12000
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
# No need to attach server name, port
modifiedSentence = clientSocket.recv(1024)
```

```
print ‘From Server:’, modifiedSentence
clientSocket.close()
```

2.105

Python TCP Server

```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
# create TCP welcoming socket
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
# server begins listening for  incoming TCP requests
print ‘The server is ready to receive’
while 1:
# loop forever
     connectionSocket, addr = serverSocket.accept()
     # server waits on accept() for incoming requests, new socket created on return
     sentence = connectionSocket.recv(1024)
     # read bytes from socket (but not address as in UDP)
     capitalizedSentence = sentence.upper()
     connectionSocket.send(capitalizedSentence)
     # close connection to this client (but not welcoming socket)s
     connectionSocket.close()
```

Chapter 2: summary

our study of network apps now complete!

1. application architectures
2.
    - client-server
    - P2P
3. application service requirements:
4.
    - reliability, bandwidth, delay
5. Internet transport service model
6.
    - connection-oriented[面向链接的], reliable[可靠的]: TCP
    - unreliable[不可靠的], datagrams[面向数据报的]: UDP

- specific protocols:
-
    1. HTTP
    2. FTP
    3. SMTP, POP, IMAP
    4. DNS
    5. P2P: BitTorrent, DHT
- socket programming: TCP, UDP sockets

most importantly: learned about protocols!

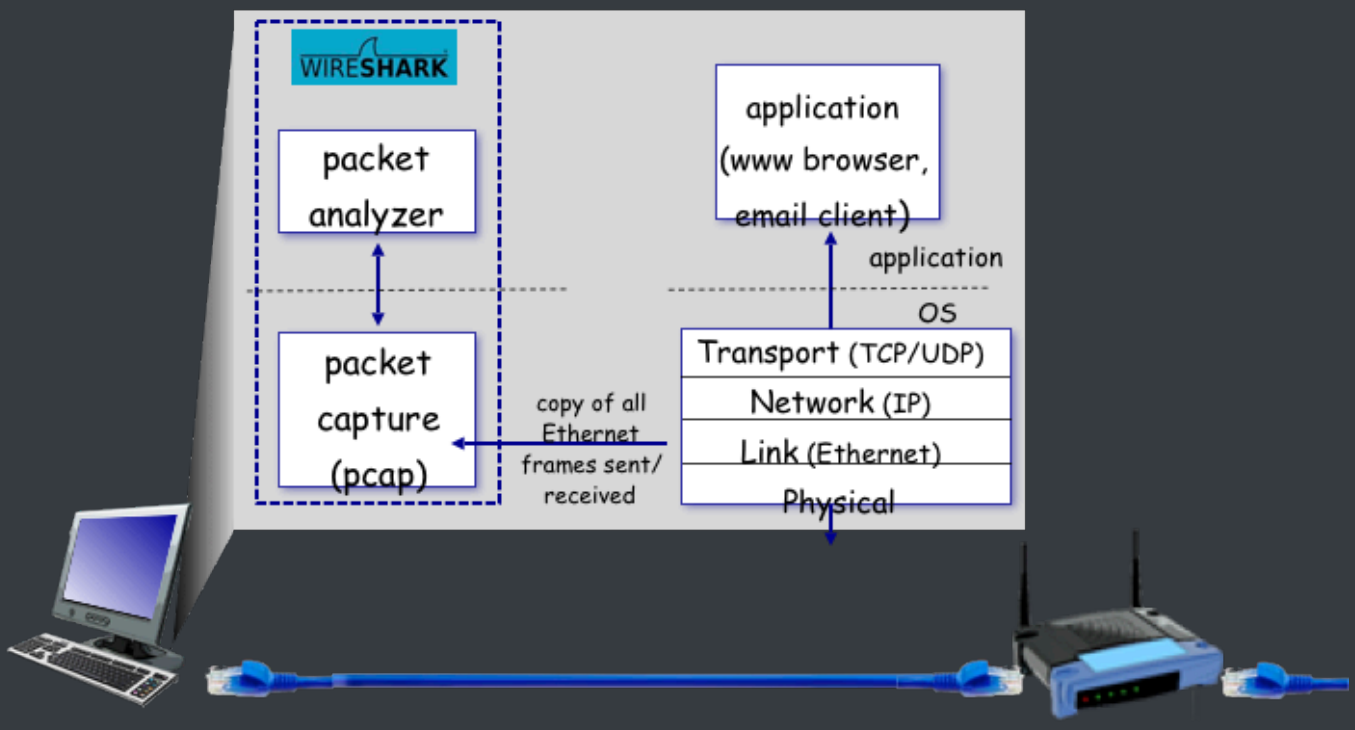
1. typical request/reply message exchange:
2.
    1. client requests info or service
    2. server responds with data, status code
3. message formats:
4.
    1. headers: fields giving info about data
    2. data: info being communicated

important themes:

1. control vs. data msgs
2.
    - in-band, out-of-band
3. centralized vs. decentralized
4. stateless vs. stateful
5. reliable vs. unreliable msg transfer
6. “complexity at network edge”

Chapter 1 Additional Slides

2.109



Chapter 3 Transport Layer

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- 1. segment structure
  2. reliable data transfer
  3. flow control
  4. connection management

3.6 principles of congestion control

3.7 TCP congestion control

our goals:

- understand principles behind transport layer services:
- 1. multiplexing, demultiplexing
  2. reliable data transfer

3. flow control
4. congestion control

- learn about Internet transport layer protocols:
-
    1. UDP: connectionless transport
    2. TCP: connection-oriented reliable transport
    3. TCP congestion control

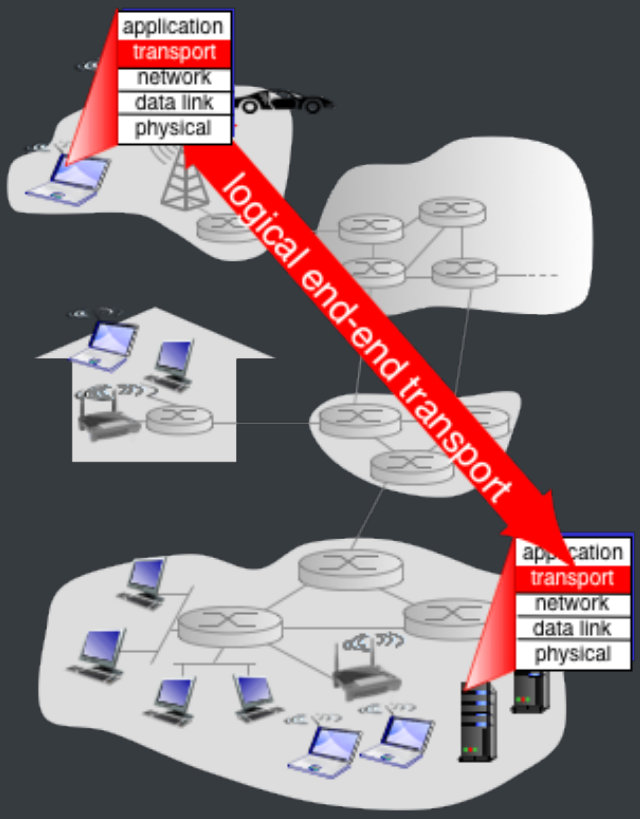
> 我们在本章采用的教学方法是，交替地讨论运输层的原理和这些 原理在现有的协议中是如何实现的。 与往常一 样，我们将特别关注因特网协议，即TCP和UDP运输层协议。
>
> 我们将从讨论运输层和网络层的关系开始。这就进人了研究运输层第一个关键功能的阶段，即将网络层的在两个端系统之间的交付服务扩展到运行在两个不同端系统上的应用层进程之间的交付服务。我们将在讨论因特网的无连接运输协议UDP时阐述这个功能。
>
> 然后我们重新回到原理学习上，面对计算机网络中最为基础性的问题之一，即两个实体 怎 样 才 能 在 一 种 会 丢 失 或 损 坏 数 据 的 媒 体 上 可 靠 地 通 信 .通 过 一系列复 杂 性不断 增加 （从而更真实！）的情况，我们将逐步建 立起 一套被运输协议用来解决这些问题 的技术. 然后，我们将说明这些原理是如何体现在因特网面向连接的运输协议TCP中的。

3.4

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
-
    - send side: breaks app messages into *segments*, passes to network layer

    - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
    - Internet: TCP and UDP



3.1 transport-layer services

3.5

Transport vs. network layer

- *network layer:* logical communication between hosts[主机]
- *transport layer:* logical communication between processes
    - relies on, enhances, network layer services

household[家庭] analogy[类推]:

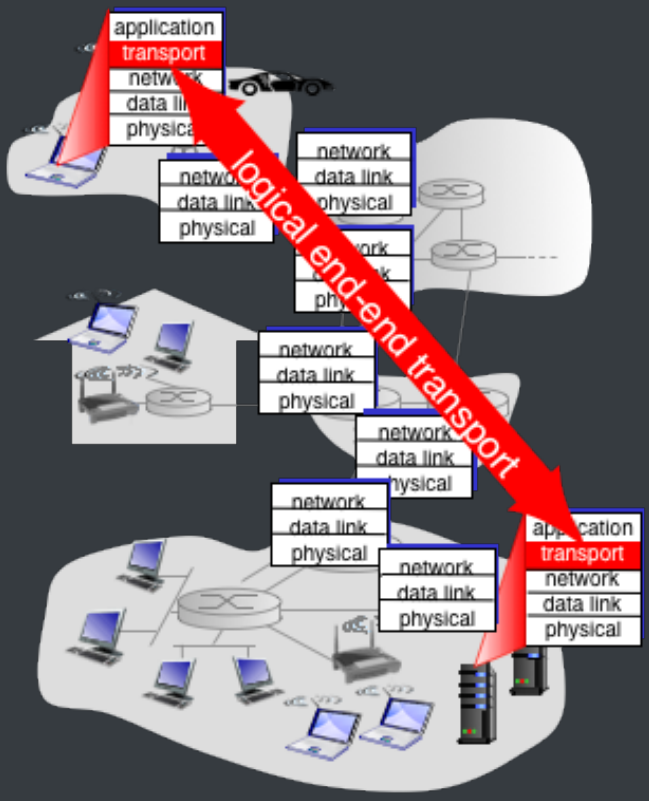
12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

3.6

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
    - congestion control
    - flow control
    - connection setup
- unreliable, unordered delivery: UDP
    - no-frills extension of "best-effort" IP
- services not available:
    - delay guarantees
    - bandwidth guarantees



3.2 multiplexing[多路复用] and demultiplexing[多路分解]

3.8

Multiplexing/demultiplexing

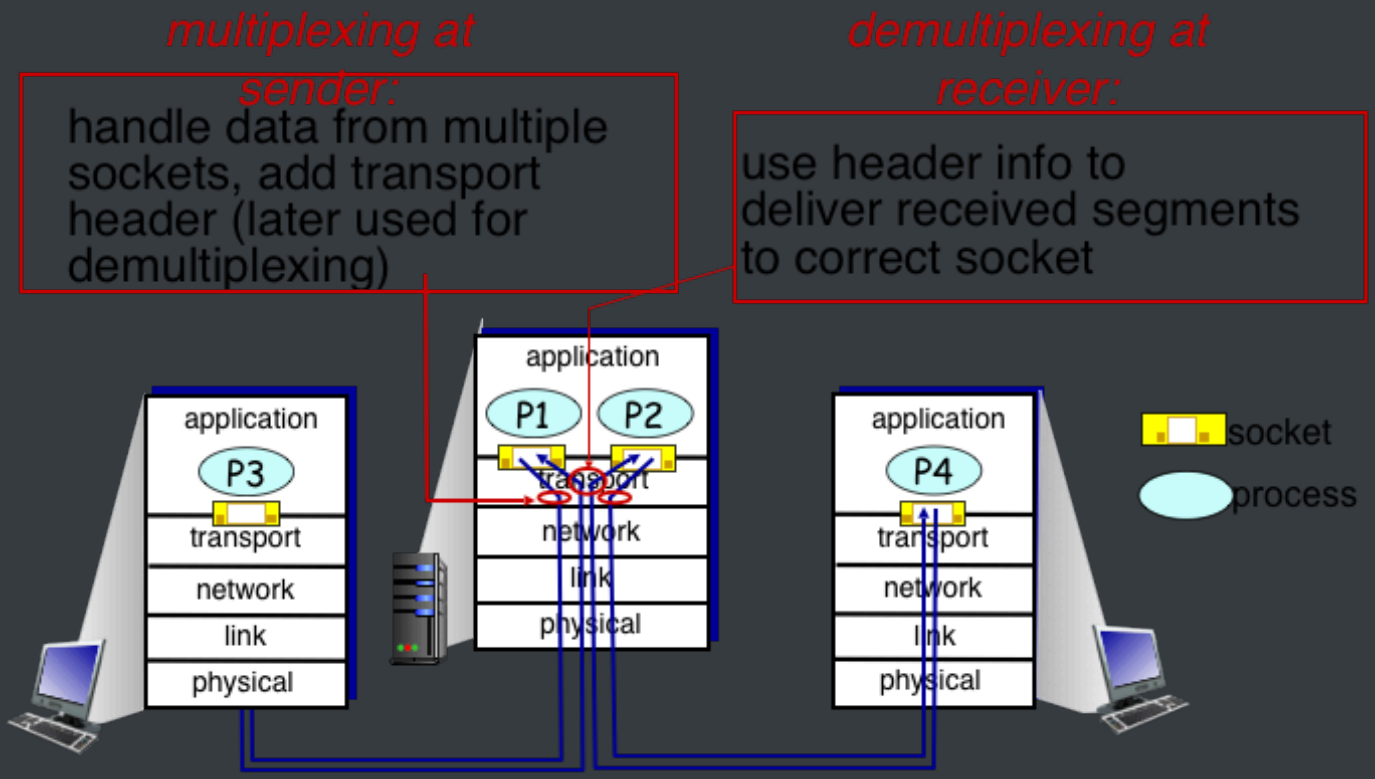
> 网络层提供的主机到 主机交付服务延伸到为运行在主机上的应用程序提供进程到进程的交付服务。

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

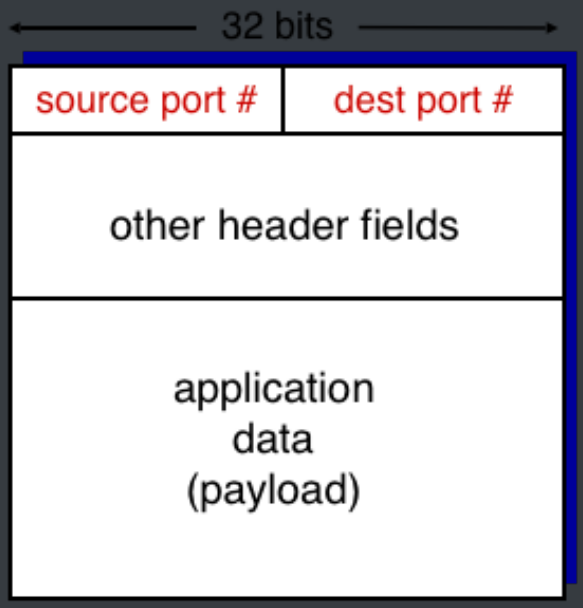
use header info to deliver received segments to correct socket



3.9

How demultiplexing works

- host receives IP datagrams
-
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

3.10

Connectionless demultiplexing

- *recall:* created socket has host-local port #:

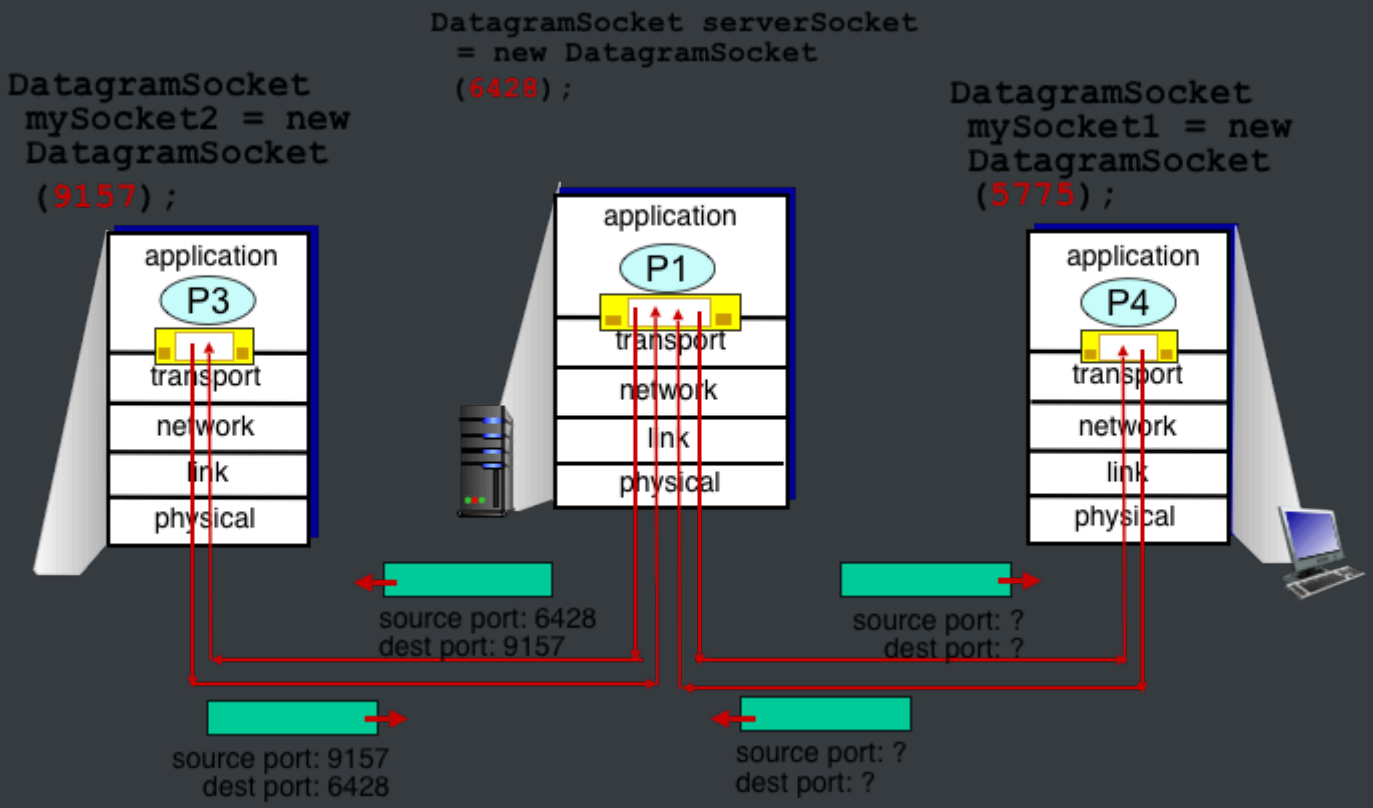
DatagramSocket mySocket1 = new DatagramSocket(12534);**

- when host receives UDP segment:
- - checks destination port # in segment
  - directs UDP segment to socket with that port #
- *recall:* when creating datagram to send into UDP socket, must specify
- - destination IP address
  - destination port #

IP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

3.11

Connectionless demux: example



3.12

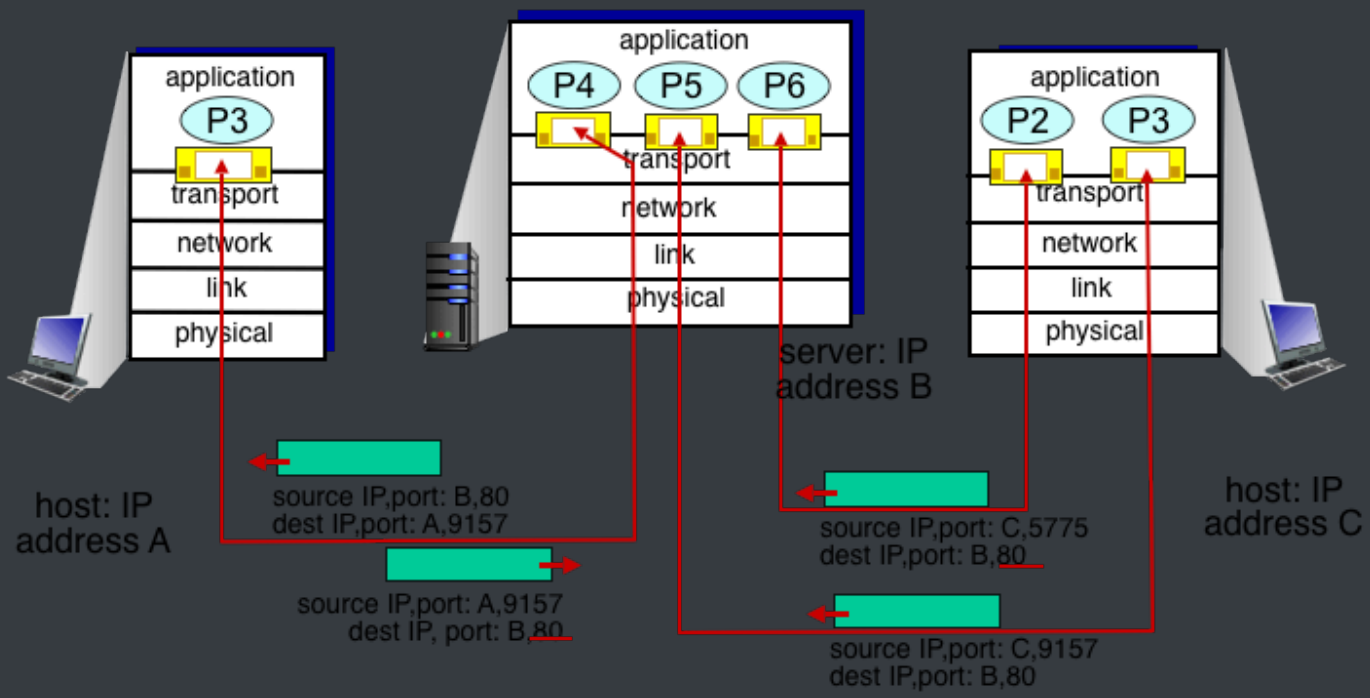
Connection-oriented demux

- TCP socket identified by 4-tuple:
-
    - source IP address
    - source port number
    - dest IP address
    - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
-
    - each socket identified by its own 4-tuple

- web servers have different sockets for each connecting client
- - non-persistent HTTP will have different socket for each request

3.13

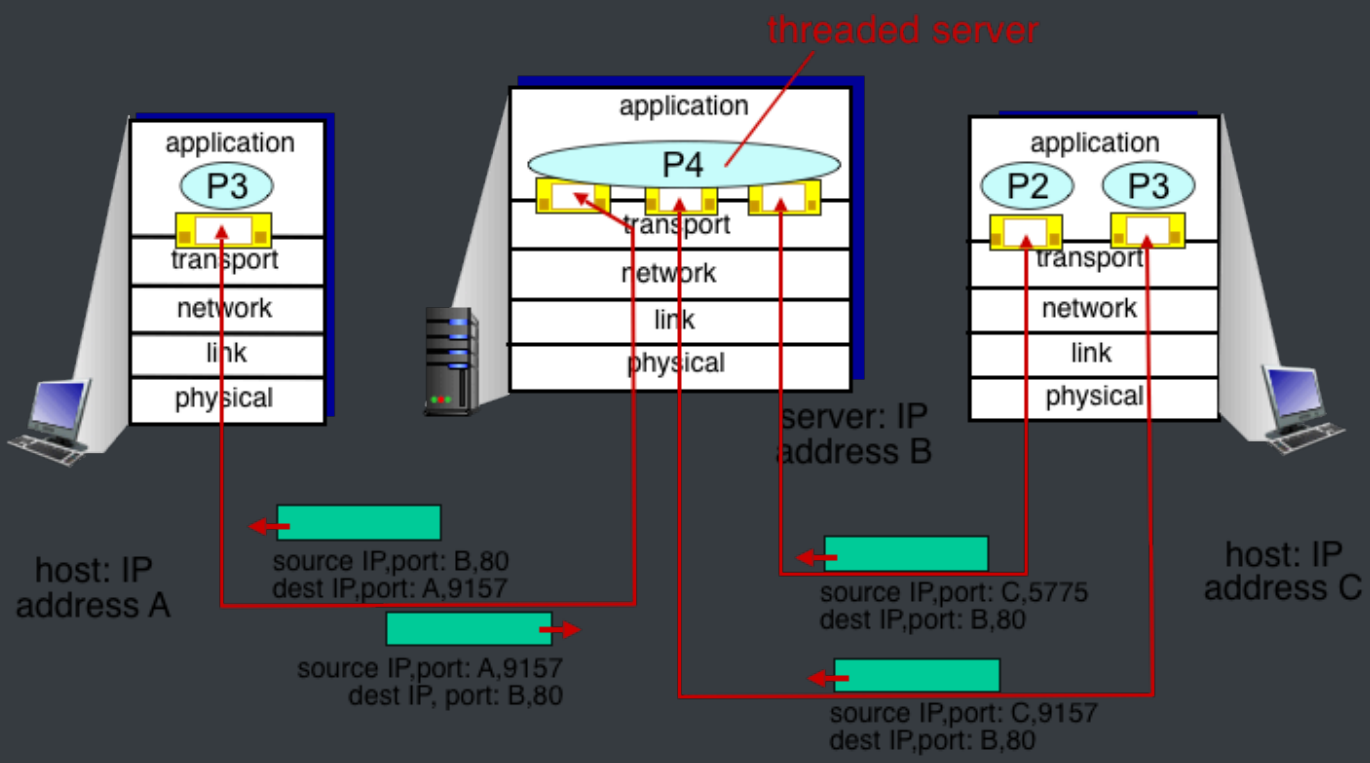
Connection-oriented demux: example



three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

3.14

Connection-oriented demux: example



3.3 connectionless transport: UDP

3.16

UDP: User Datagram Protocol [RFC 768]

- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
- - lost
  - delivered out-of-order to app
- *connectionless:*
- - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others
- UDP use:
- - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS

        - SNMP
    - reliable transfer over UDP:
    -
        - add reliability at application layer
        - application-specific error recovery!

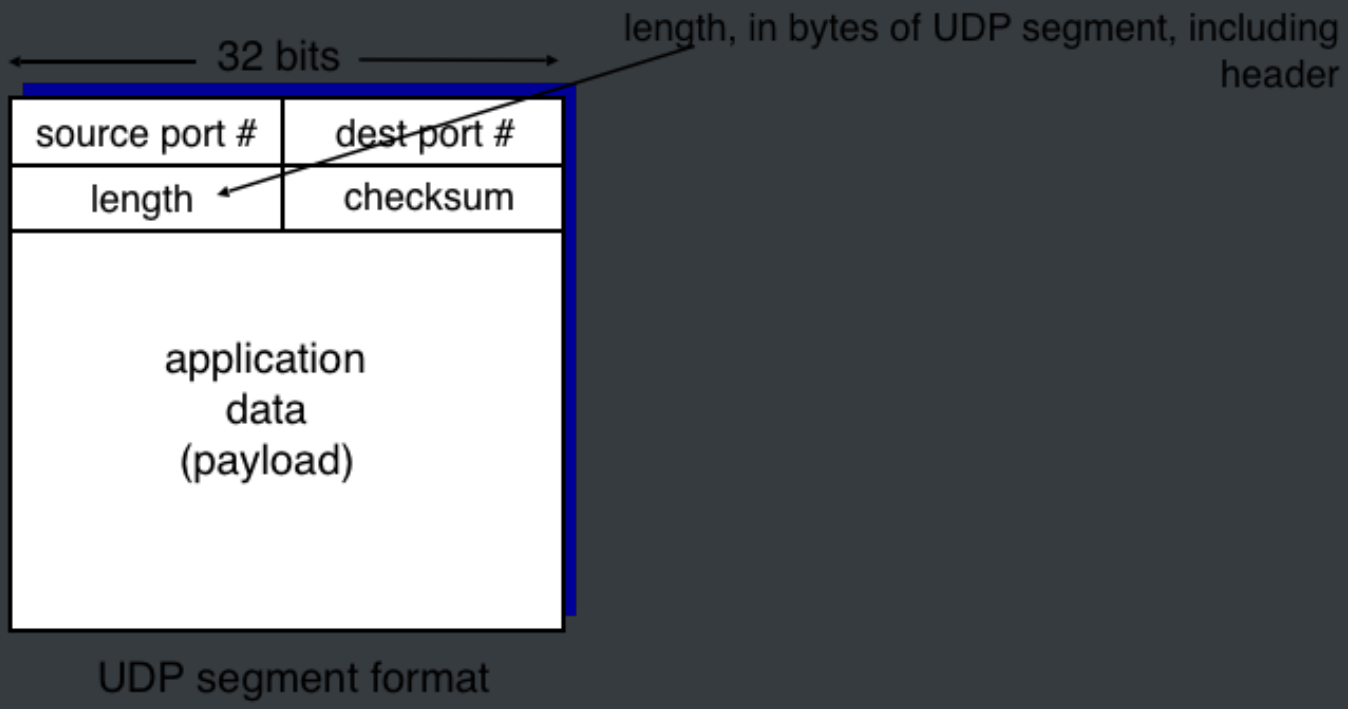
3.17

Internet apps: application, transport protocols

Application	**Application layer protocol**	**Underlying transport protocol**
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary	TCP or UDP
Internet telephony	(e.g. RealNetworks)	

3.18

UDP: segment header



UDP segment format

why is there a UDP?

1. no connection establishment (which can add delay)
2. simple: no connection state at sender, receiver
3. small header size
4. no congestion control: UDP can blast away as fast as desired

'''http 0632 0035 001C E217 '''

发送端口号 0632

'''http 0000 0110 0011 0010 '''

目的端口号 0035 == 53

'''http 0000 0000 0011 0101 '''

$$= 2^5 + 2^4 + 2^2 + 2^0 = 32 + 16 + 5 = 53 \quad (5)$$

端口号是53

payload 28 bytes

3.19

UDP checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one’s complement sum) of segment contents
- sender puts checksum value into UDP checksum field

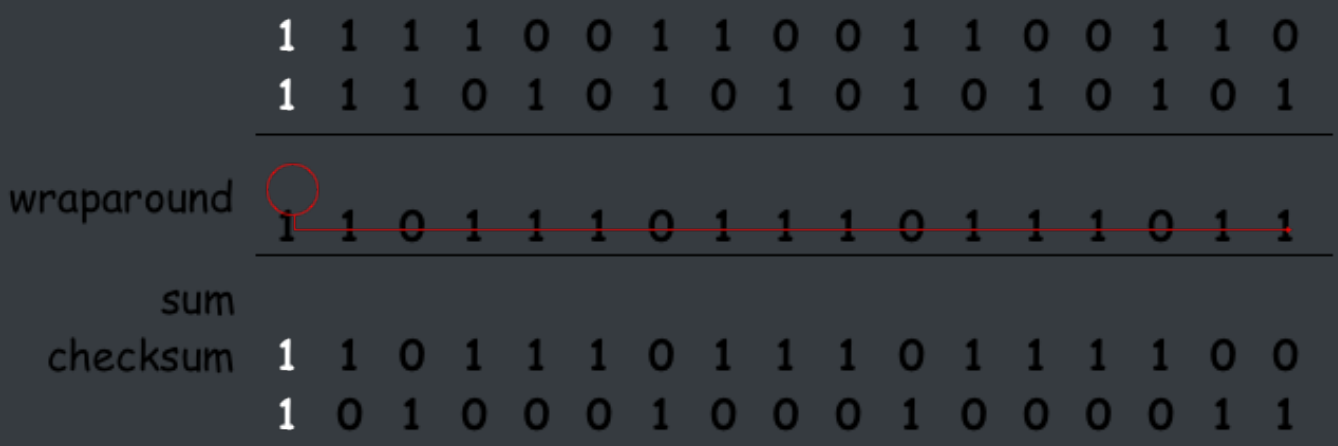
receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
-
    - NO - error detected
    - YES - no error detected. *But maybe errors nonetheless?* More later ….

3.20

Internet checksum: example

example: add two 16-bit integers



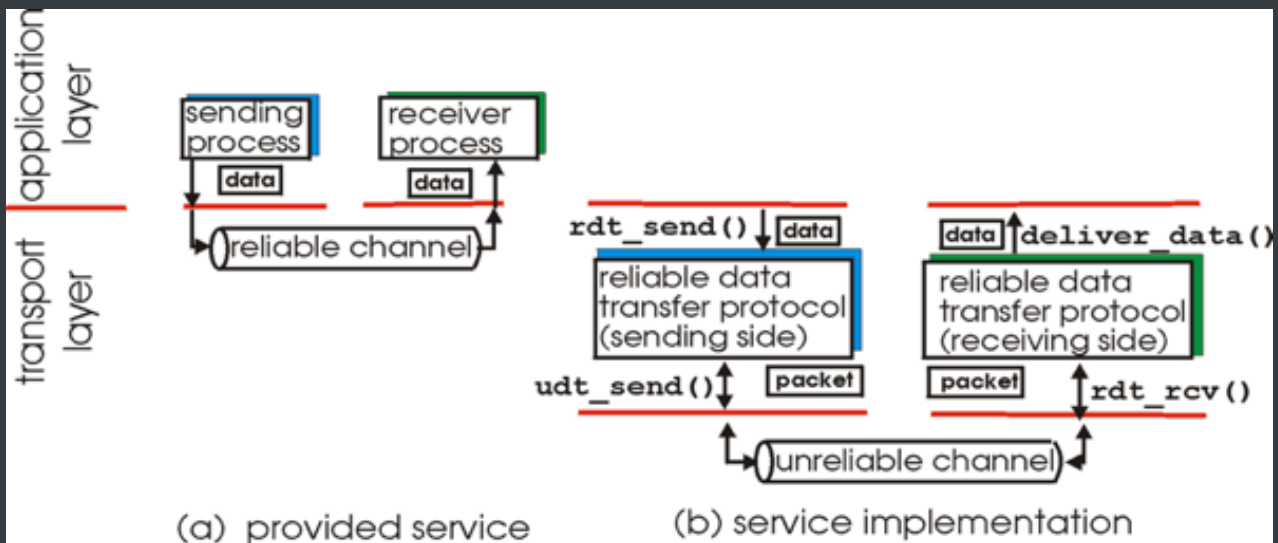
Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

3.4 principles of reliable data transfer

3.22

Principles of reliable data transfer

- important in application, transport, link layers
- - top-10 list of important networking topics!

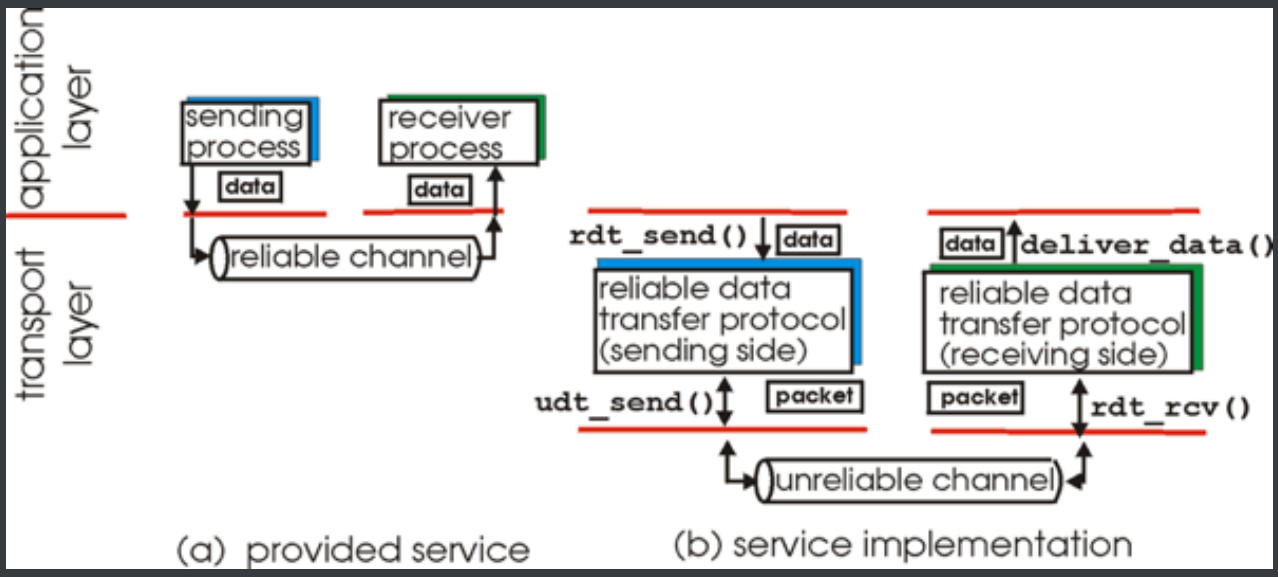


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

3.23

Principles of reliable data transfer

- important in application, transport, link layers
- - top-10 list of important networking topics!

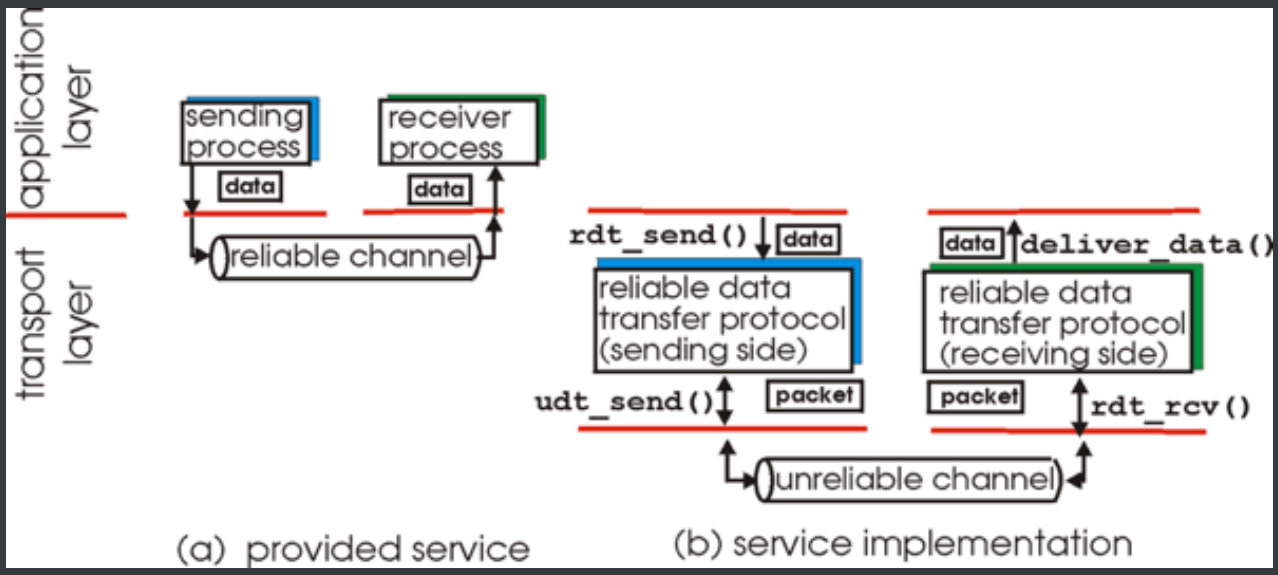


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

3.24

Principles of reliable data transfer

- important in application, transport, link layers
- - top-10 list of important networking topics!



- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

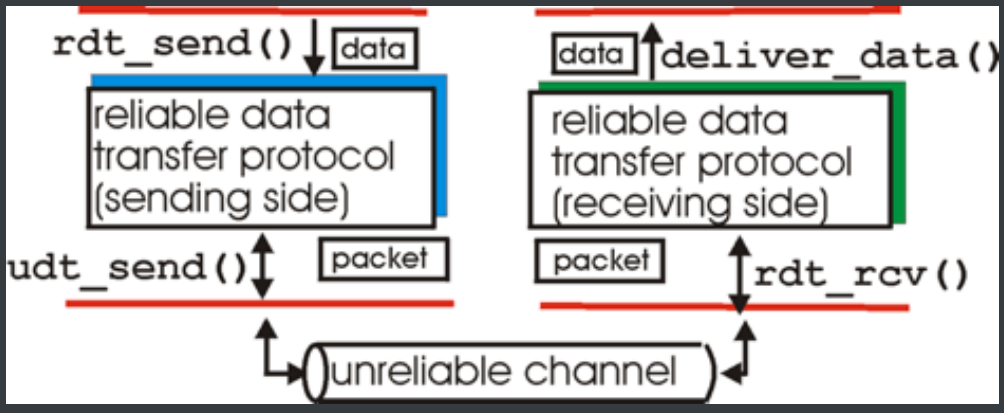
3.25

Reliable data transfer: getting started

send side

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

udt_send(): called by rdt, to transfer packet over unreliable channel to receiver



receive side

deliver_data(): called by **rdt** to deliver data to upper

rdt_rcv(): called when packet arrives on rcv-side of channel

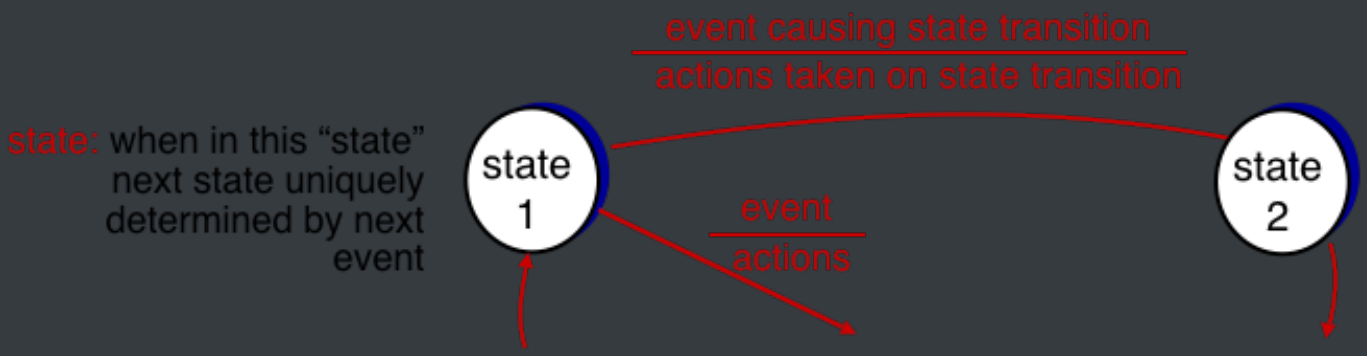
3.26

Reliable data transfer: getting started

we'll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
- - - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver

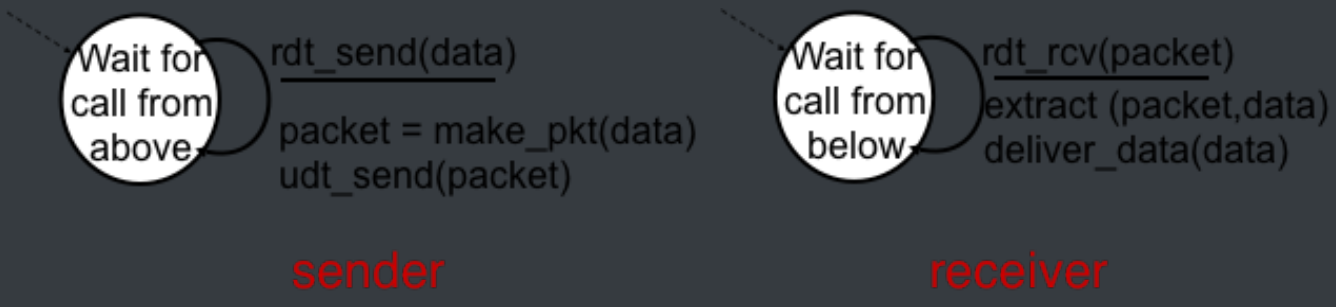
state: when in this "state" next state uniquely determined by next event



3.27

rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
- - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
- - sender sends data into underlying channel
  - receiver reads data from underlying channel



3.28

rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
- - checksum to detect bit errors
- *the* question: how to recover from errors:

***How do humans recover from "errors"* during conversation?**

3.29

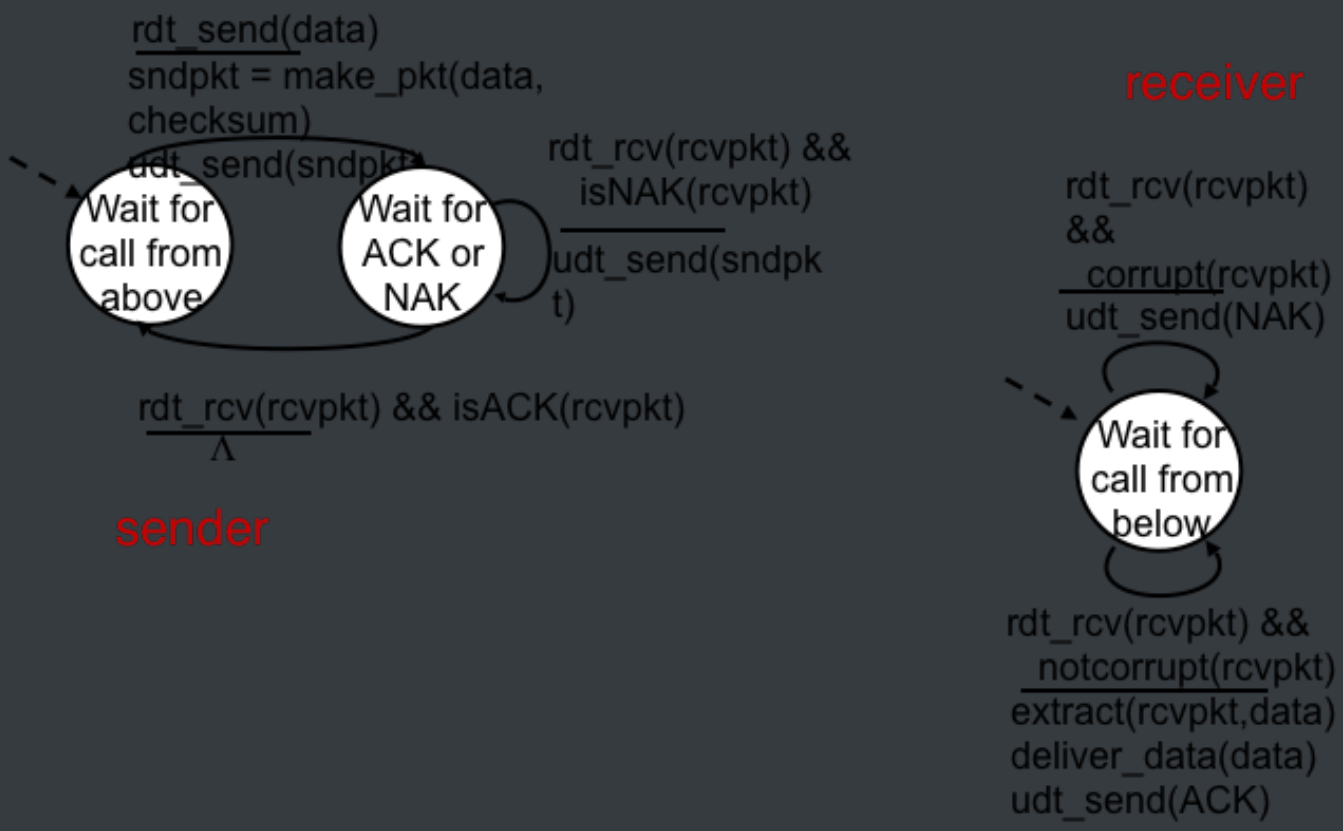
rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
- - checksum to detect bit errors

- *the* question: how to recover from errors:
- - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- new mechanisms in **rdt2.0** (beyond **rdt1.0**):
- - error detection
  - feedback: control msgs (ACK,NAK) from receiver to sender

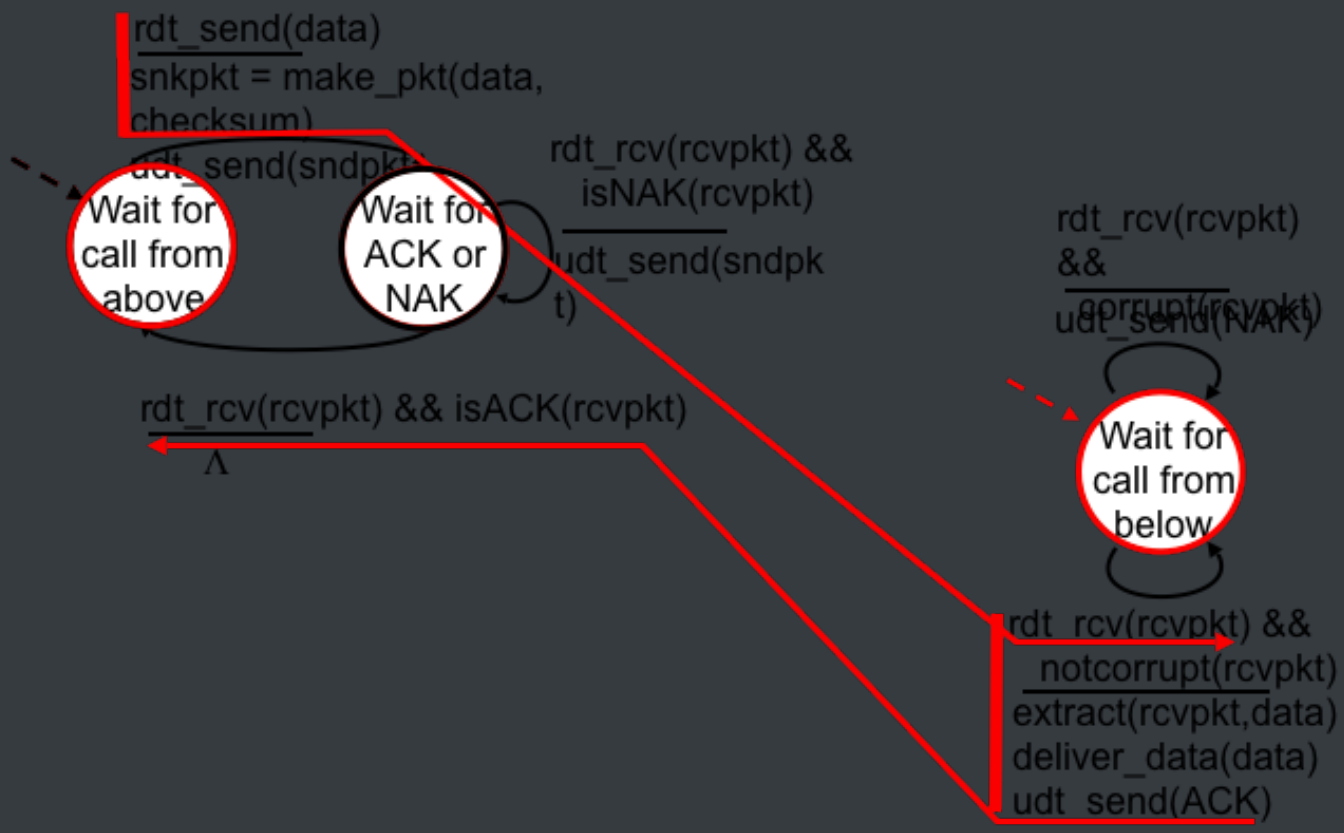
3.30

rdt2.0: FSM specification



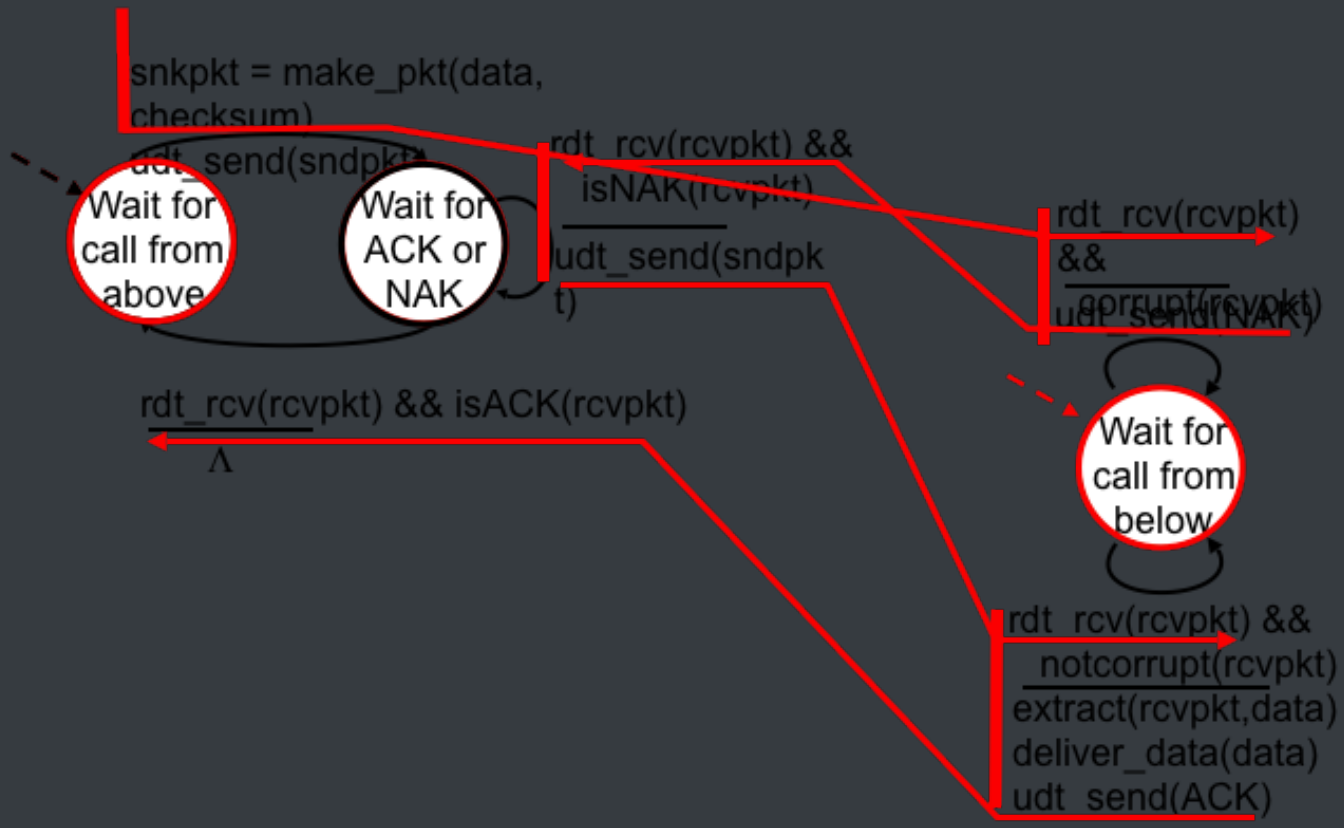
3.31

rdt2.0: operation with no errors



3.32

rdt2.0: error scenario



3.33

rdt2.0 has a fatal flaw!

what happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

handling duplicates:

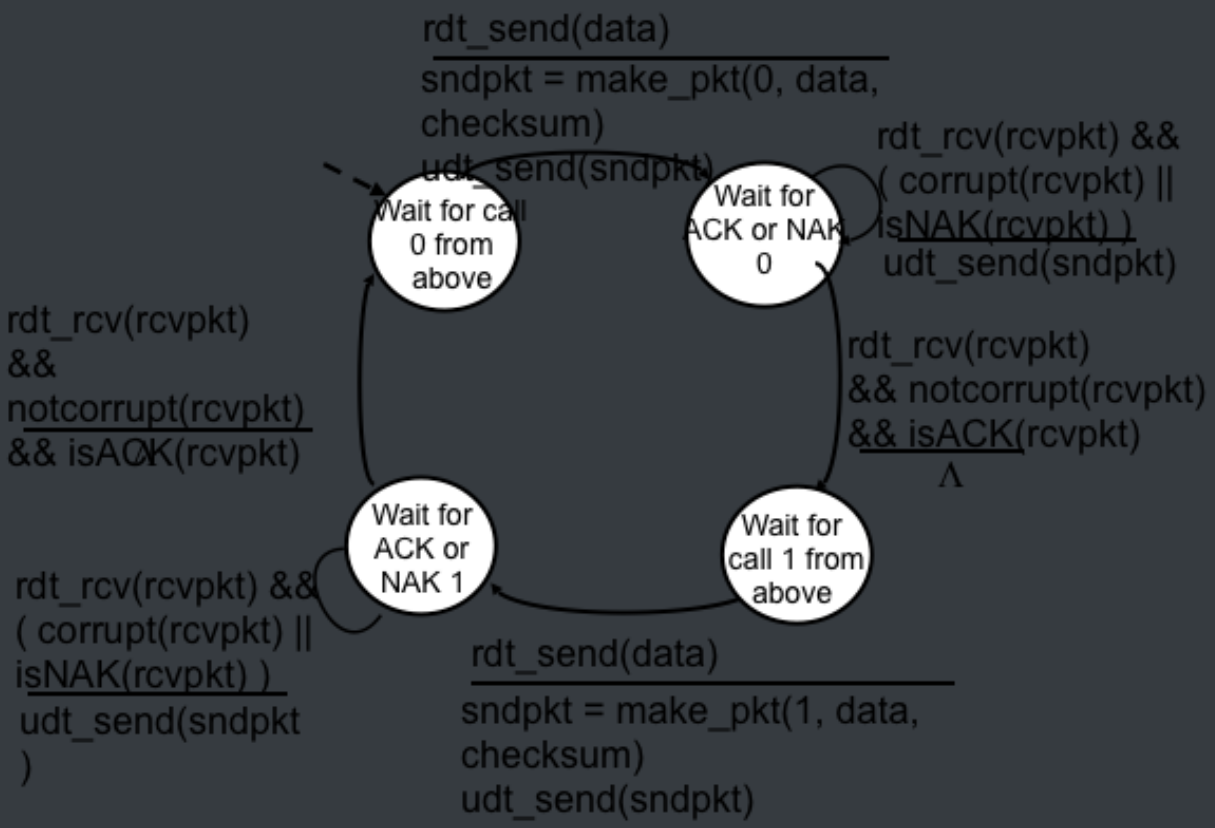
- sender retransmits current pkt if ACK/NAK corrupted
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait

- sender sends one packet, then waits for receiver response

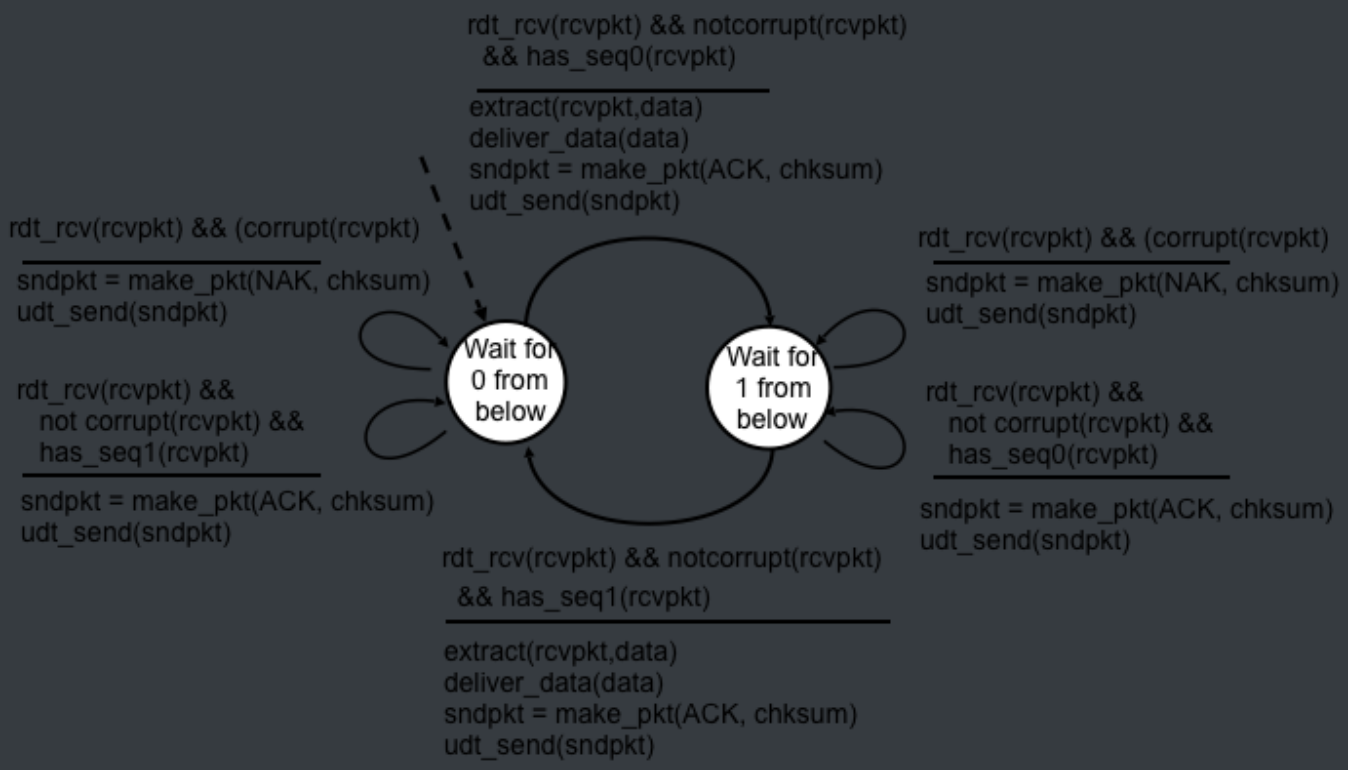
3.34

rdt2.1: sender, handles garbled ACK/NAKs



3.35

rdt2.1: receiver, handles garbled ACK/NAKs



3.36

rdt2.1: discussion

sender:

- seq # added to pkt
- two seq. #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
- - state must "remember" whether "expected" pkt should have seq # of 0 or 1

receiver:

- must check if received packet is duplicate
- - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

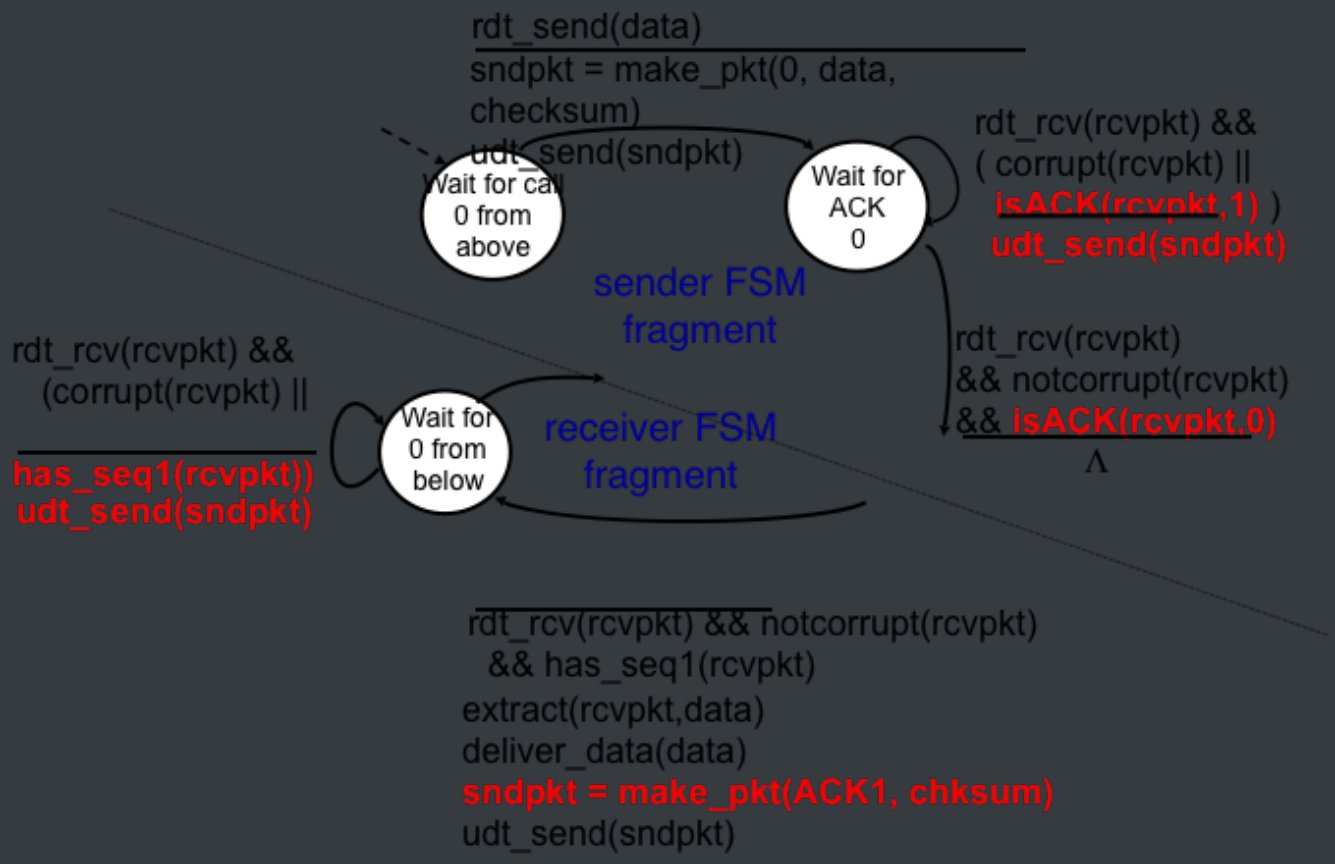
3.37

rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
- - - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

3.38

rdt2.2: sender, receiver fragments



3.39

rdt3.0: channels with errors *and* loss

new assumption: underlying channel can also lose packets (data, ACKs)

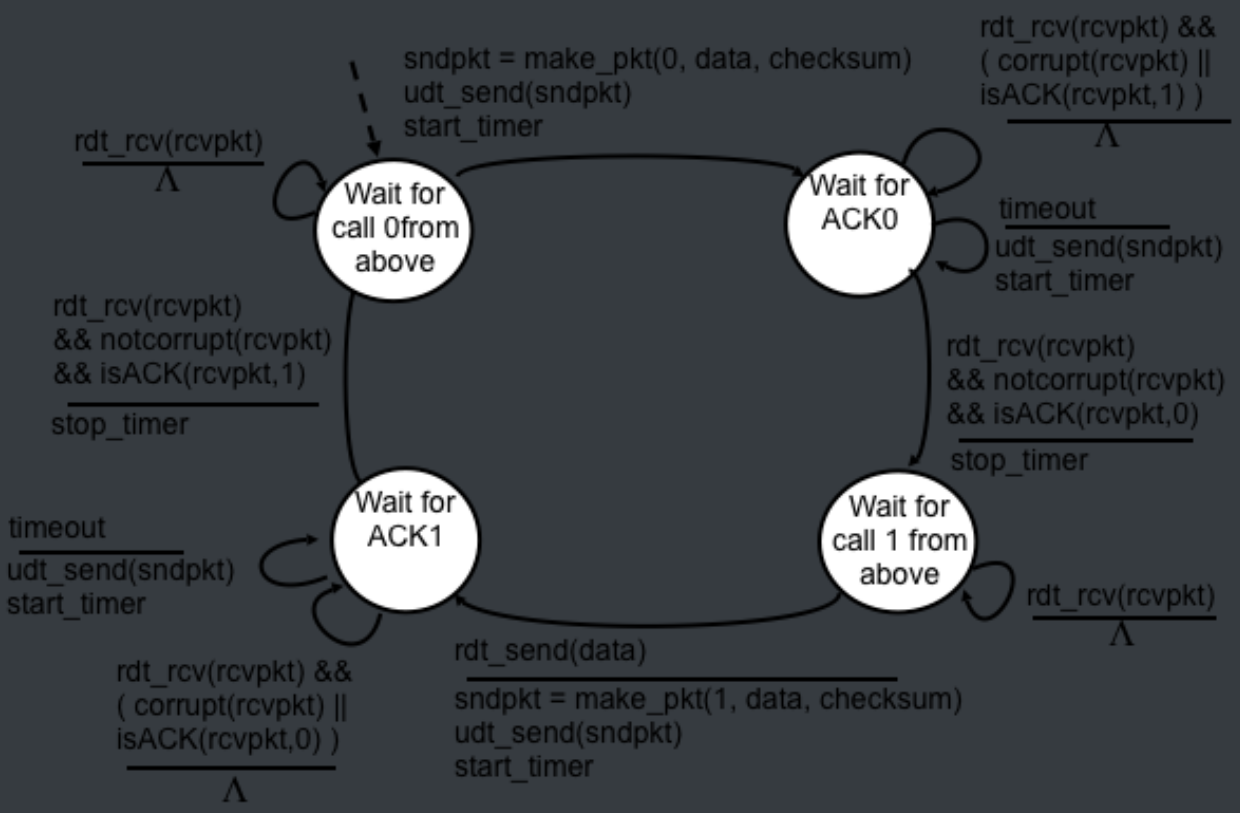
- - checksum, seq. #, ACKs, retransmissions will be of help … but not enough

approach: sender waits “reasonable” amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
- - retransmission will be duplicate, but seq. #’s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

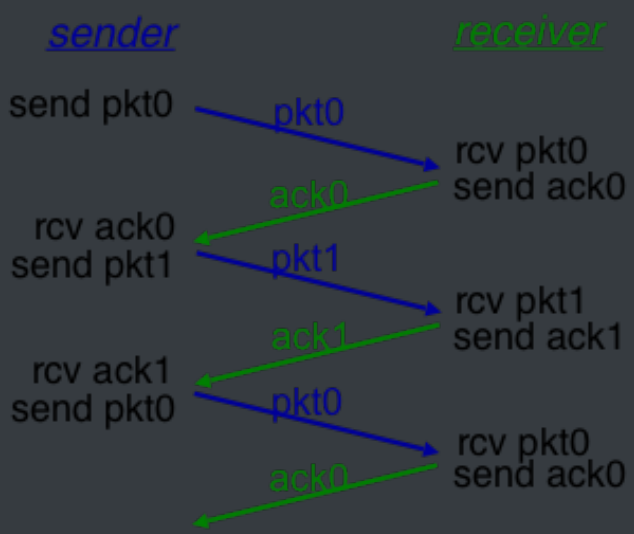
3.40

rdt3.0 sender

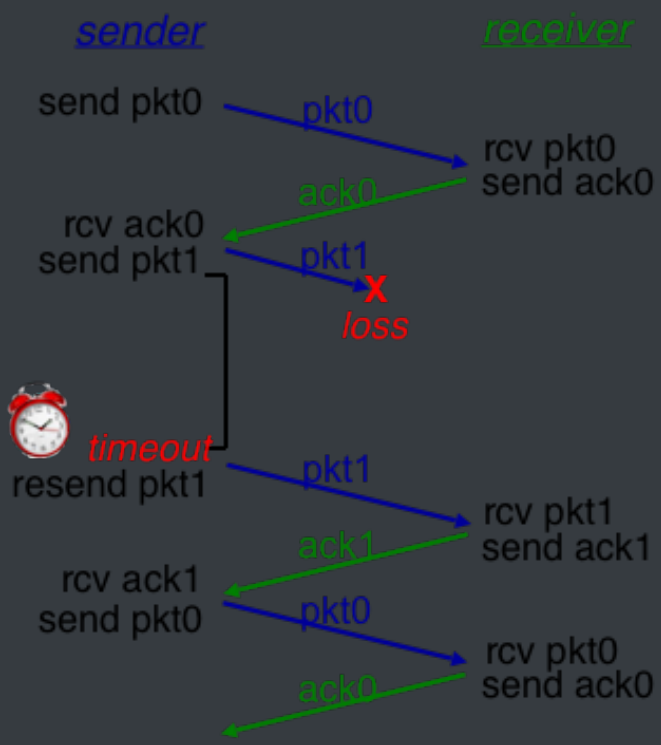


3.41

rdt3.0 in action



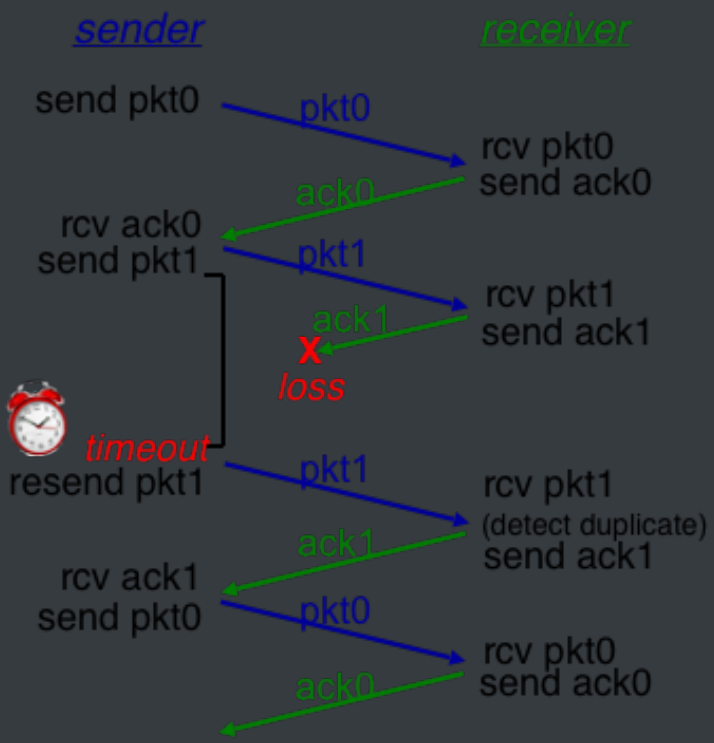
(a) no loss



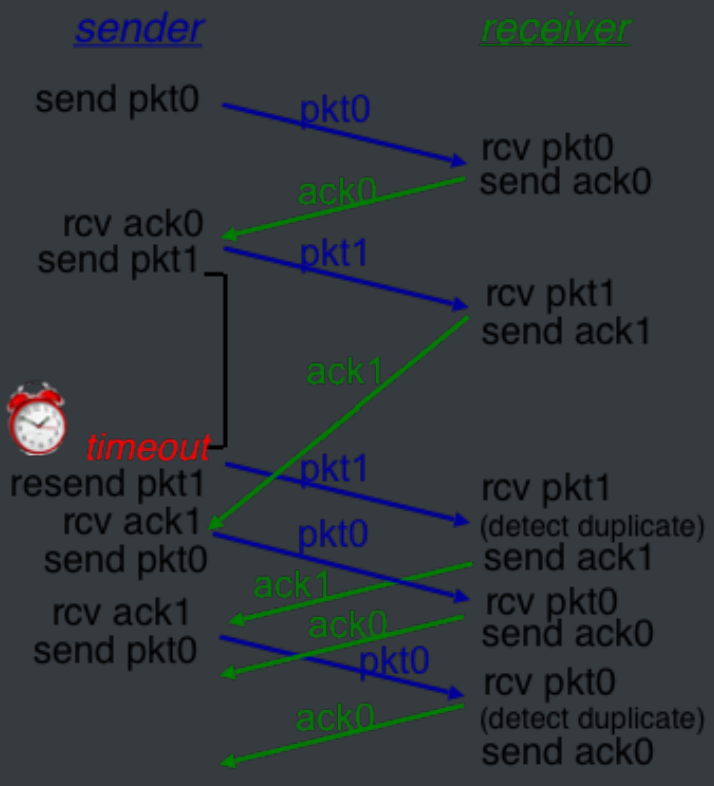
(b) packet loss

3.42

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

Performance of rdt3.0

- rdt3.0 is correct, but performance stinks
- e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

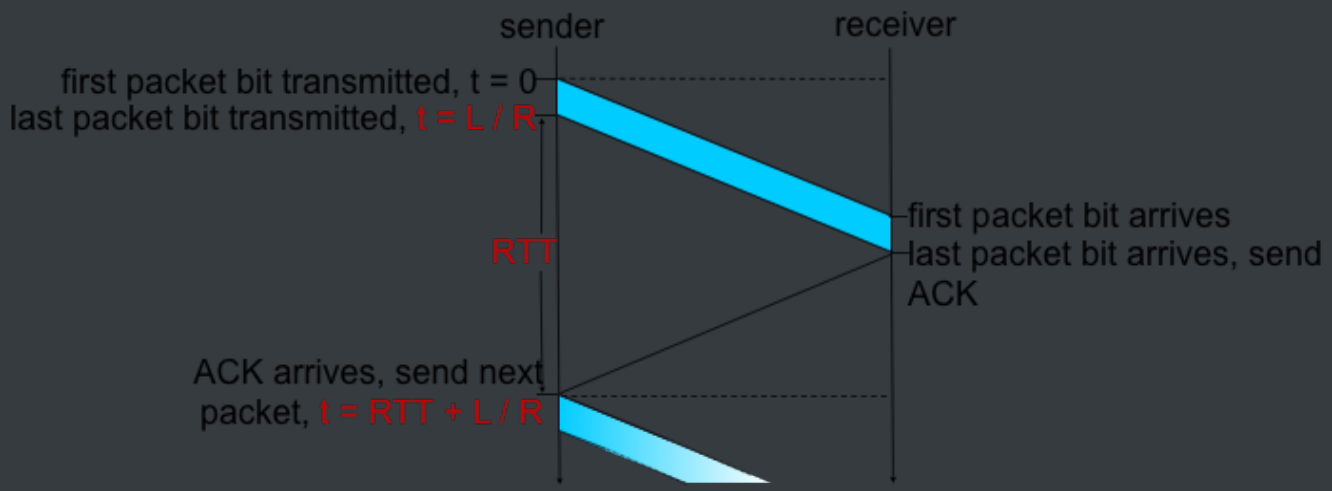
- U sender: *utilization* – fraction of time sender busy sending

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

-
  - if RTT=30 msec, 1KB pkt every 30 msec: 267Kb/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!

3.44

rdt3.0: stop-and-wait operation



first packet bit transmitted, t = 0

last packet bit transmitted, t = L / R

first packet bit arrives

last packet bit arrives, send ACK

ACK arrives, send next packet, t = RTT + L / R

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

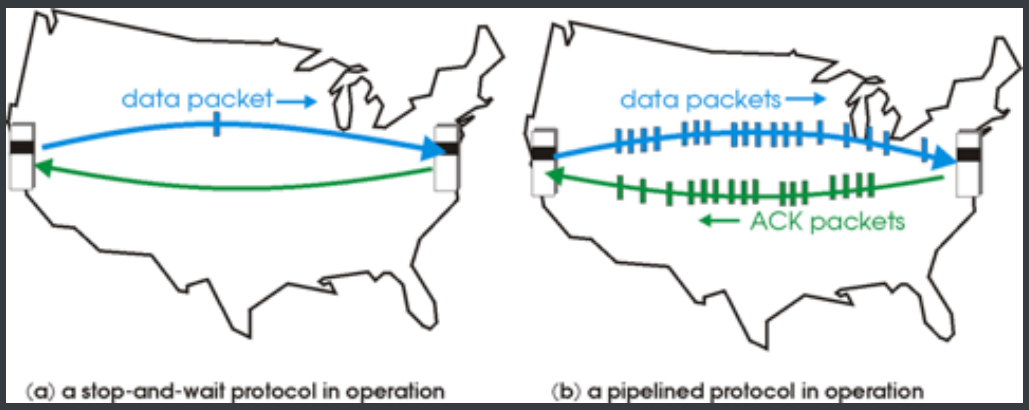
3.45

Pipelined protocols

pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- range of sequence numbers must be increased

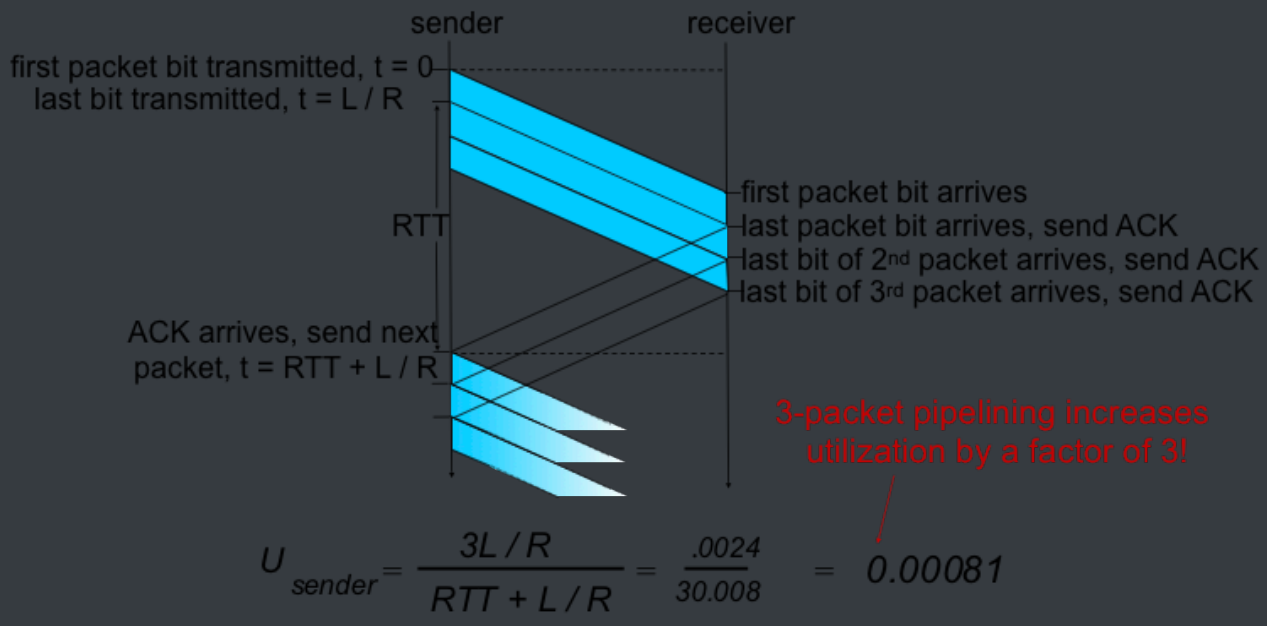
- buffering at sender and/or receiver



- two generic forms of pipelined protocols: *go-Back-N, selective repeat*

3.46

Pipelining: increased utilization



3-packet pipelining increases utilization by a factor of 3!

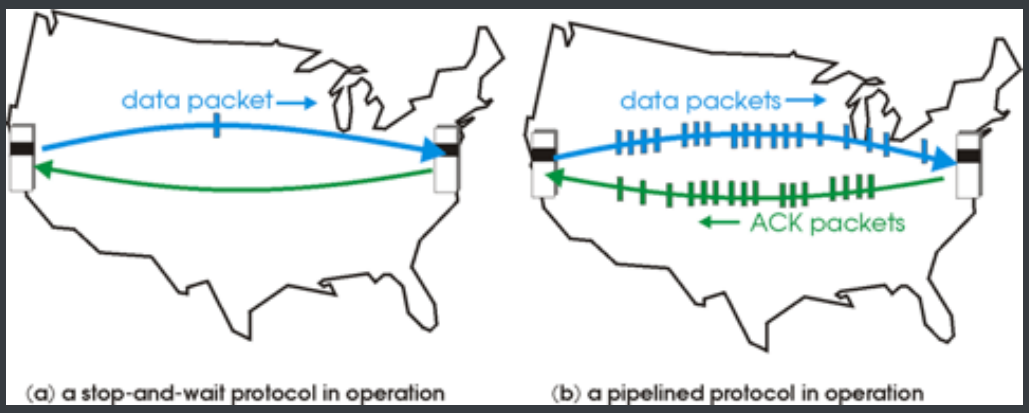
$$U_{sender} = \frac{\frac{3L}{R}}{RTT + \frac{L}{R}} = \frac{0.0024}{30.008} = 0.00081 \qquad (6)$$

3.47

Pipelined protocols

pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- - range of sequence numbers must be increased
  - buffering at sender and/or receiver



- two generic forms of pipelined protocols: *go-Back-N, selective repeat*

3.48

Pipelined protocols: overview

Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
- - doesn’t ack packet if there’s a gap
- sender has timer for oldest unacked packet
- - when timer expires, retransmit *all* unacked packets

Selective Repeat:

- sender can have up to N unack’ed packets in pipeline

- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
- - when timer expires, retransmit only that unacked packet

3.49

Why is window up to N?

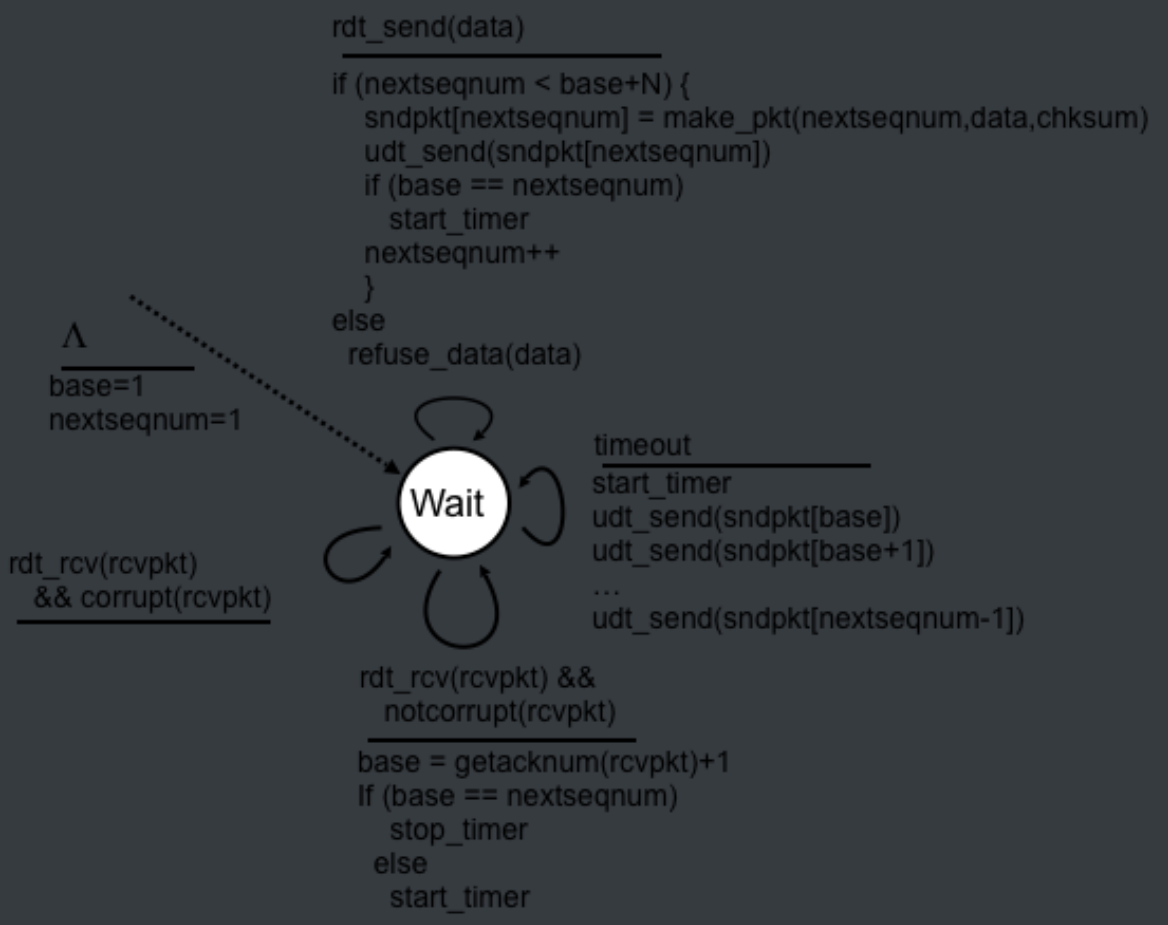
Go-Back-N: sender

- k-bit seq # in pkt header
- “window” of up to N, consecutive unack’ed pkts allowed

- ACK(n): ACKs all pkts up to, including seq # n - *“cumulative ACK”*
- - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

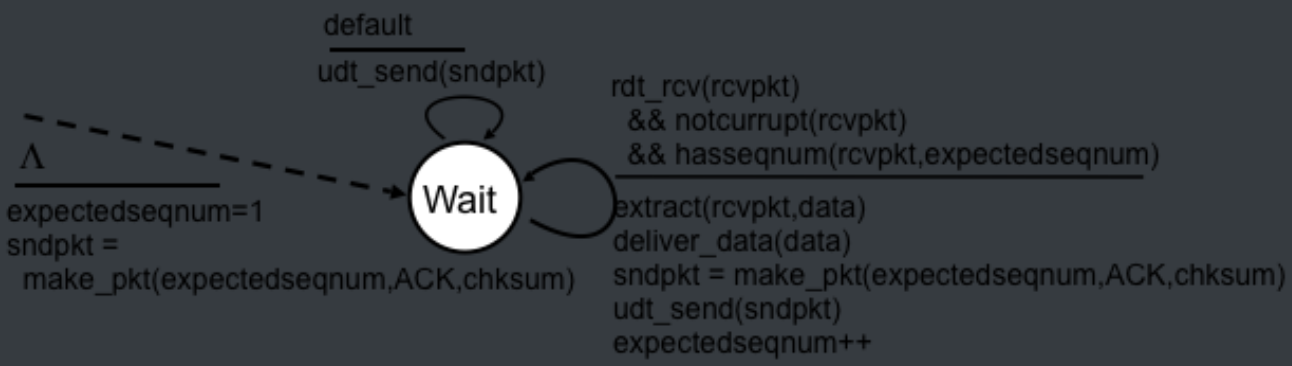
3.50

GBN: sender extended FSM



3.51

GBN: receiver extended FSM



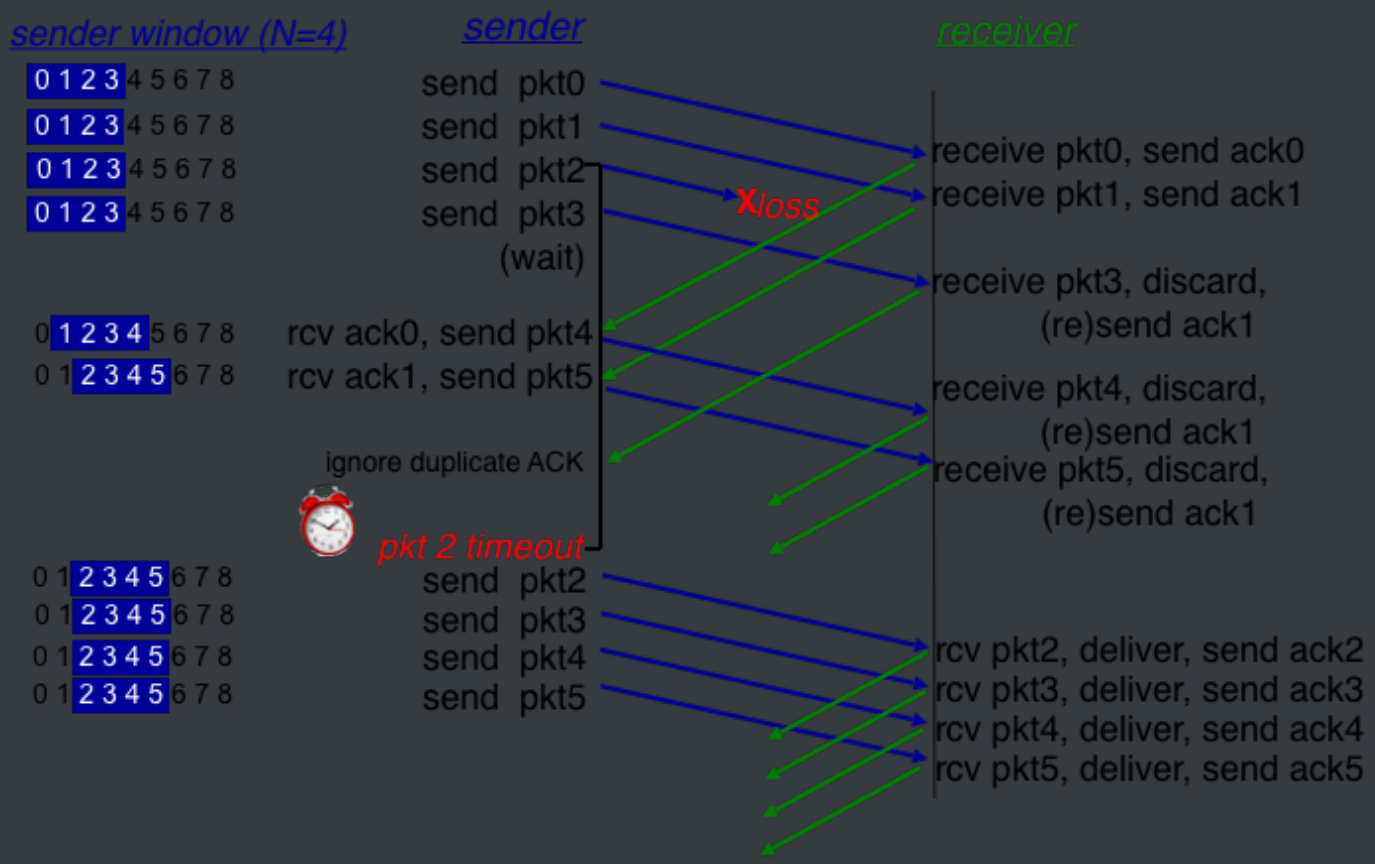
Why not buffering?

ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- - may generate duplicate ACKs
  - need only remember **expectedseqnum**
- out-of-order pkt:
- - discard (don’t buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

3.52

GBN in action



3.5 connection-oriented transport: TCP segment structure

- 1. segment structure
  2. reliable data transfer
  3. flow control
  4. connection management

TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- point-to-point:
    - one sender, one receiver
- reliable, in-order *byte steam:*
    - no “message boundaries”
- pipelined:
    - TCP congestion and flow control set window size
- full duplex data:
    - bi-directional data flow in same connection
    - MSS: maximum segment size
- connection-oriented:
    - handshaking (exchange of control msgs) inits sender, receiver state before data exchange

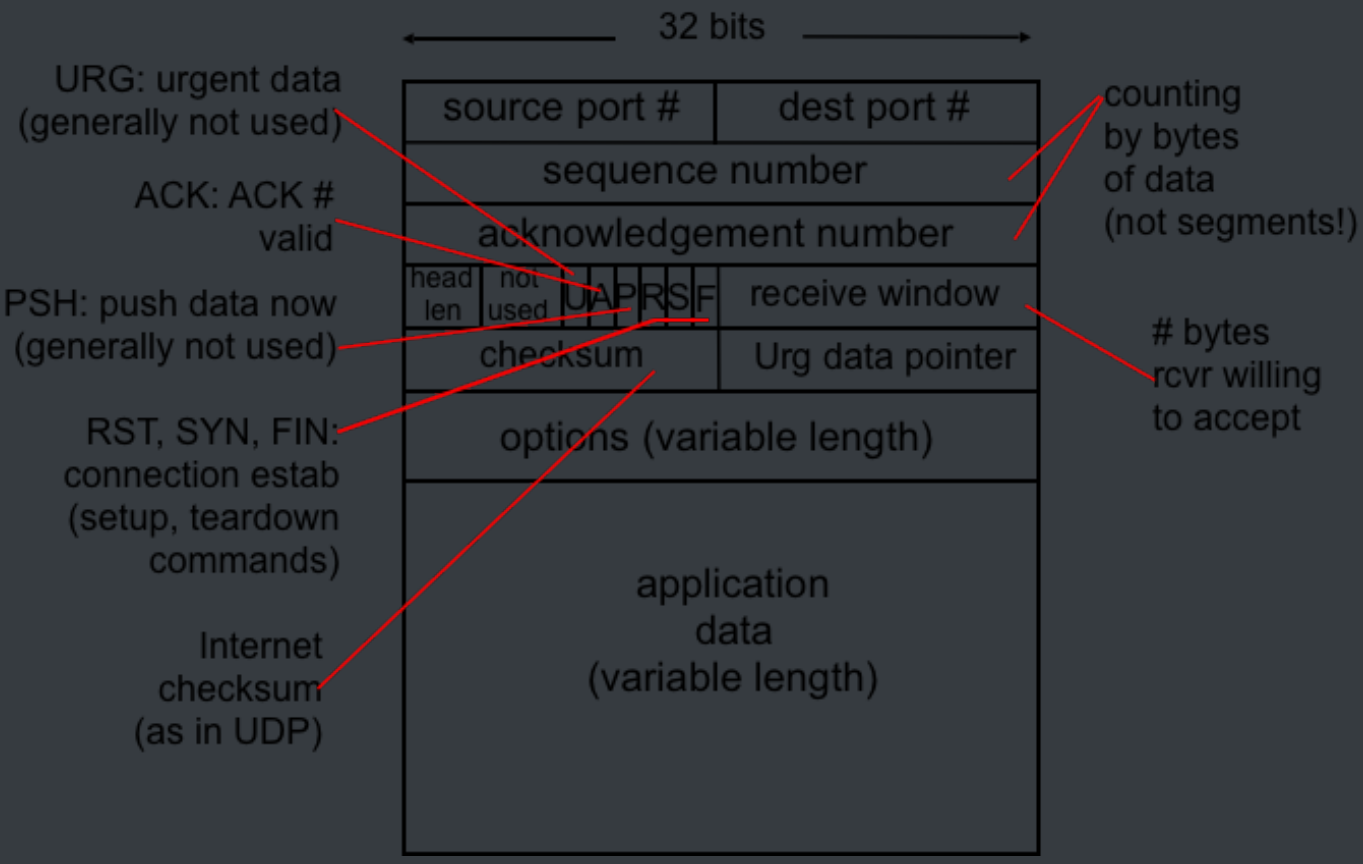
    > TCP被称为是面向连接的（conn ection-oriented）， 这是因为在一个应用进程 可以开始 向 另 一 个应用 进 程发送数 据之 前，这两 个进程必须先相互 “握手”，即它们必须相互 发送 某些预备报文段，以建立确保数据传输的参数。作为TCP连接建立的一部分，连接的双方 都将初始化 与TCP连接相关的许多TCP状态变量

- flow controlled:
    - sender will not overwhelm receiver

> 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，T C P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。

3.66

TCP segment structure



3.67

TCP seq. numbers, ACKs

sequence numbers:

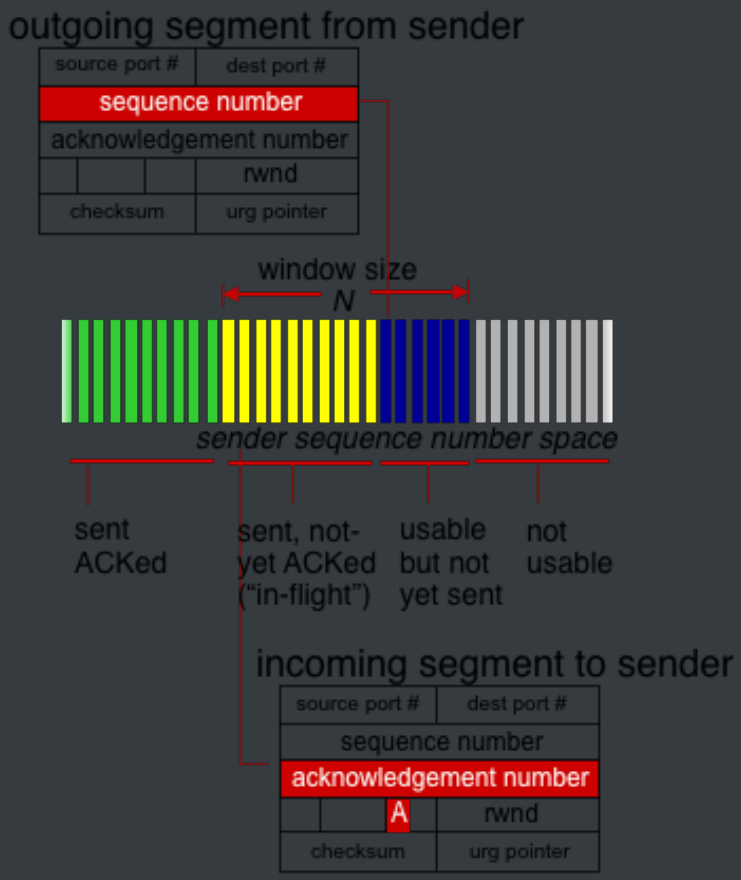
- byte stream “number” of first byte in segment’s data

acknowledgements:

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

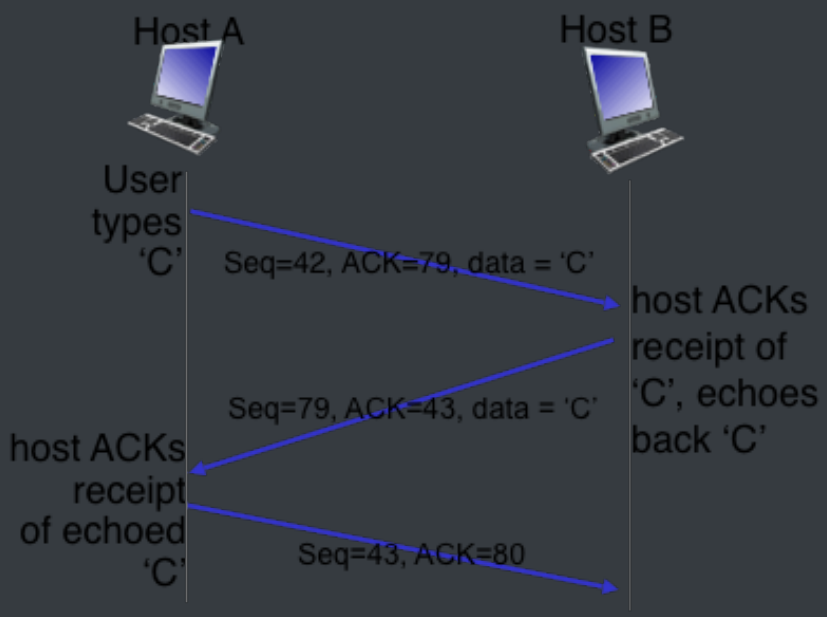
Q: how receiver handles out-of-order segments

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



3.68

TCP seq. numbers, ACKs



simple telnet scenario

3.69

TCP round trip time, timeout

Q: how to set TCP timeout value?

1. longer than RTT
2.
    - but RTT varies
3. *too short:* premature timeout, unnecessary retransmissions
4. *too long:* slow reaction to segment loss

Q: how to estimate RTT?

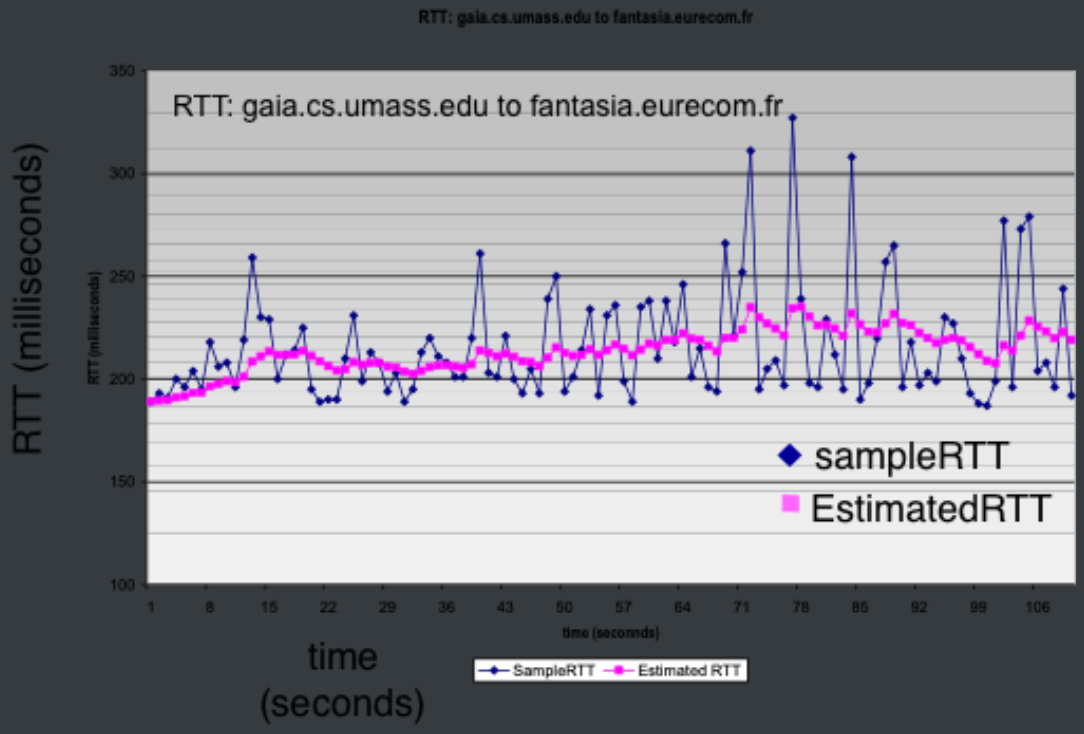
1. **SampleRTT**: measured time from segment transmission until ACK receipt
2.
    - ignore retransmissions
3. **SampleRTT** will vary, want estimated RTT "smoother"
4.
    - average several *recent* measurements, not just current **SampleRTT**

3.70

TCP round trip time, timeout

$$EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT \quad (7)$$

- [平均値]exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: α = 0.125



3.71

TCP round trip time, timeout

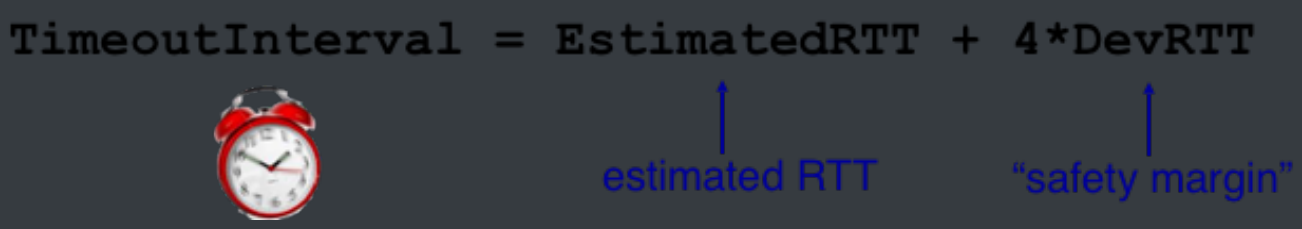
- timeout interval: **EstimatedRTT** plus “safety margin”
- - large variation in **EstimatedRTT ->** larger safety margin

- estimate SampleRTT deviation from EstimatedRTT:

$$DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT| \quad (8)$$

(typically, β = 0.25)

$$TimeoutInterval = EstimatedRTT + 4 * DevRTT \quad (9)$$



3.5 connection-oriented transport: TCP reliable data transfer

3.73

TCP reliable data transfer

- TCP creates rdt service on top of IP’s unreliable service
- - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
- - timeout events
  - duplicate acks
- let’s initially consider simplified TCP sender:
- - ignore duplicate acks
  - ignore flow control, congestion control

3.74

TCP sender events:

data rcvd from app:

- create segment with seq
- seq is byte-stream number of first data byte in segment
- start timer if not already running
-
    - think of timer as for oldest unacked segment
    - expiration interval: **TimeOutInterval**

timeout:

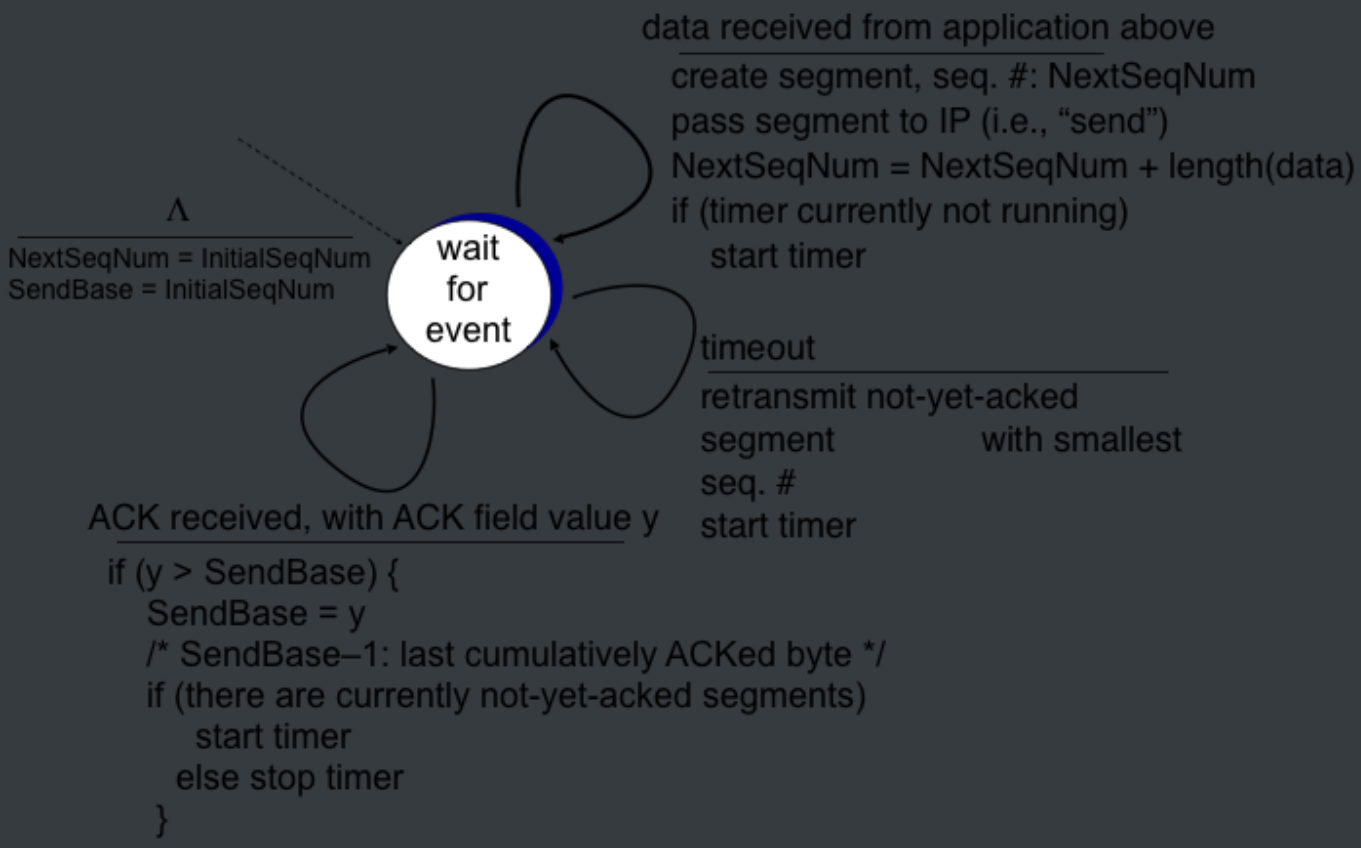
1. retransmit segment that caused timeout
2. restart timer

ack rcvd:

- if ack acknowledges previously unacked segments
-
    - update what is known to be ACKed
    - start timer if there are still unacked segments

3.75

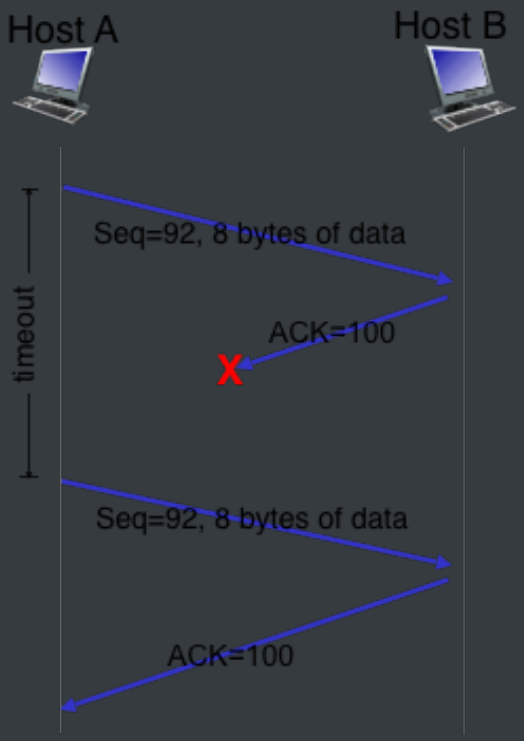
TCP sender (simplified)



3.76

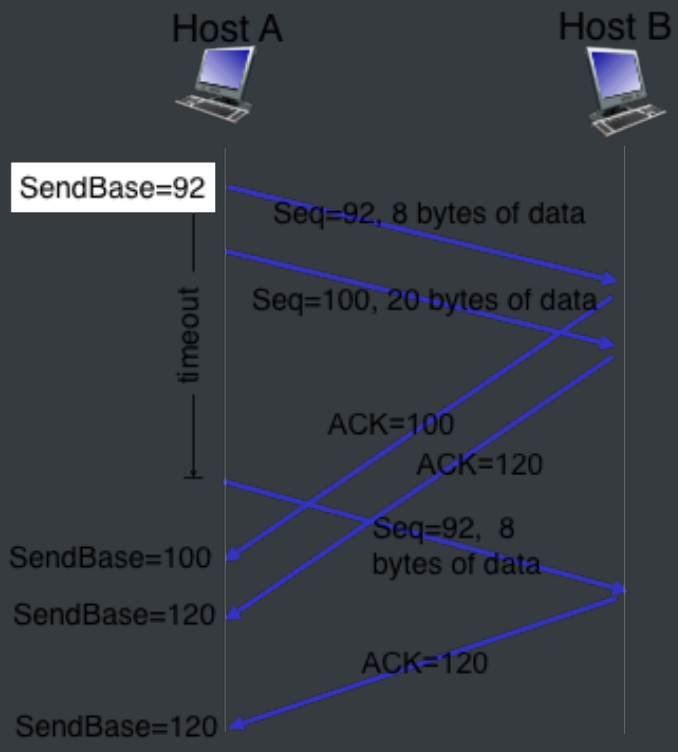
TCP: retransmission scenarios

lost ACK scenario



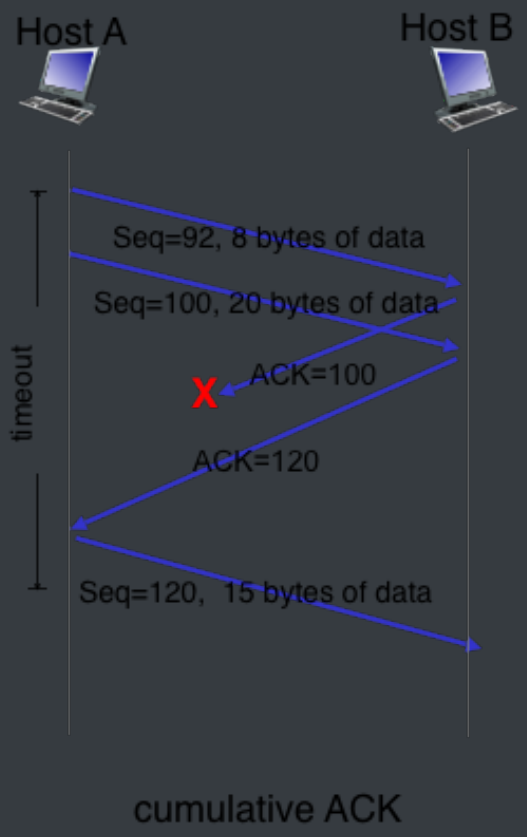
lost ACK scenario

premature[过早地] timeout



premature timeout

cumulative ACK



cumulative ACK

3.78

TCP ACK generation [RFC 1122, RFC 2581]

event at receiver	*TCP receiver action*
1.1 arrival of in-order segment with with expected seq #	1.1 delayed ACK. Wait up to 500 msdelayed ACK.
1.2 All data up toexpected seq #, already ACKed	1.2 If no next segment,for next segment.
2.1 arrival of in-order segment with expected seq #	2.1 immediately send single cumulative ACK
2.2 One other segment has ACK pending	2.2 ACKing both in-order segments
3 arrival of out-of-order segment higher-than-expect seq # Gap detected	3. immediately send duplicate ACK, indicating seq. # of next expected byte
4. arrival of segment that partially or completely fills gap	4. immediate send ACK, provided that segment starts at lower end of gap

3.79

TCP fast retransmit[快速SOON重传, 三次重复的ACK(fast retransmit after sender receipt of triple duplicate ACK)]

- time-out period often relatively long:
- - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
- - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

if sender receives 3 ACKs for same data (“triple duplicate ACKs”), resend unacked segment with smallest seq likely that unacked segment lost, so don’t wait for timeout

3.80

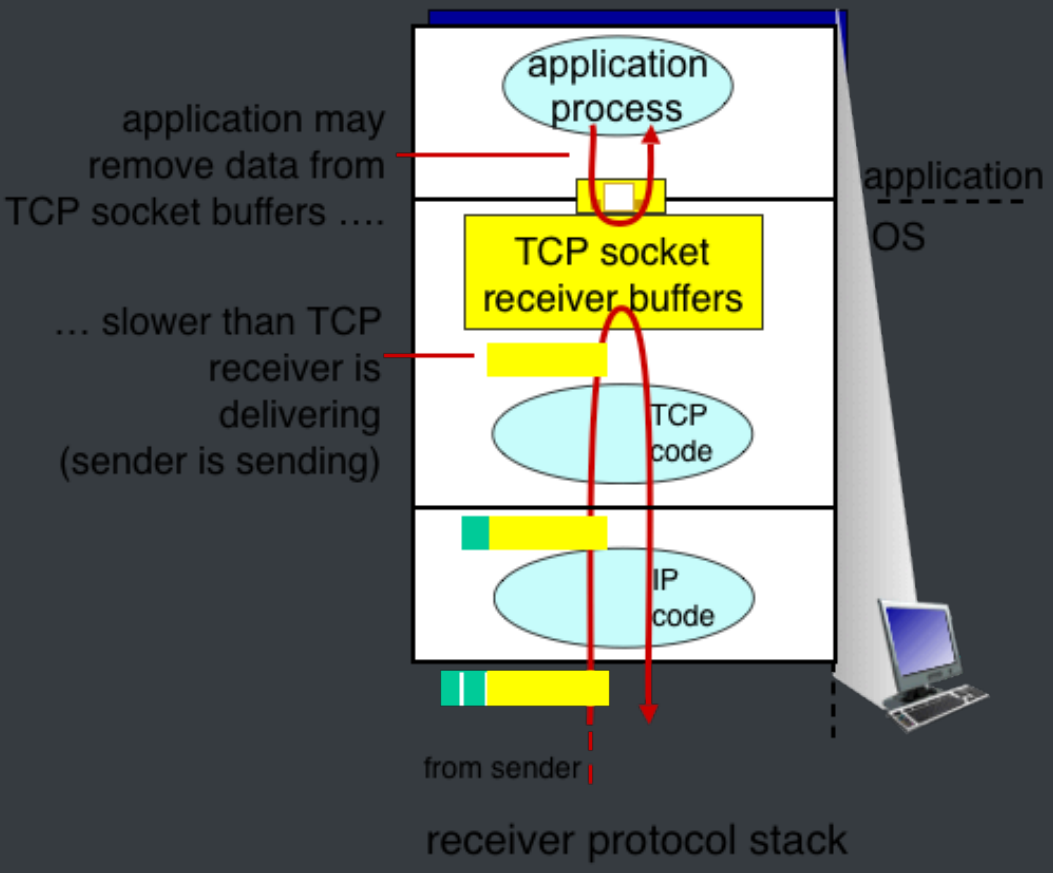
TCP fast retransmit

fast retransmit after sender receipt of triple duplicate ACK

3.5 TCP: flow control[流量控制]

3.82

TCP: flow control

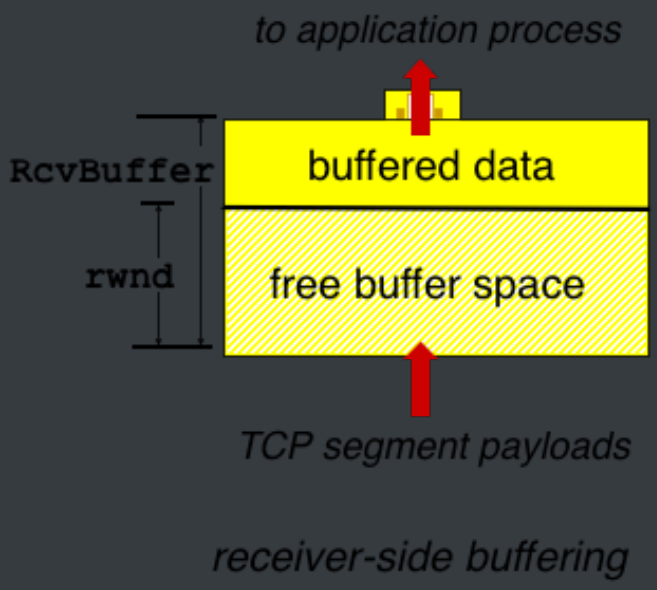


3.83

TCP: flow control

1. receiver "advertises" free buffer space by including $rwnd$ value in TCP header of reciver-to-sender segments
    1. 1 $RcvBuffer$ size set via socket option (typical default is 4096 bytes)
    2. 2 many operating systems autoadjust $RcvBuffer$
2. Sender Limits amount of unacked ("in-flight") data to reciver $rwndvalue$

3. guarantees receive buffer will not overflow



receiver-side buffering

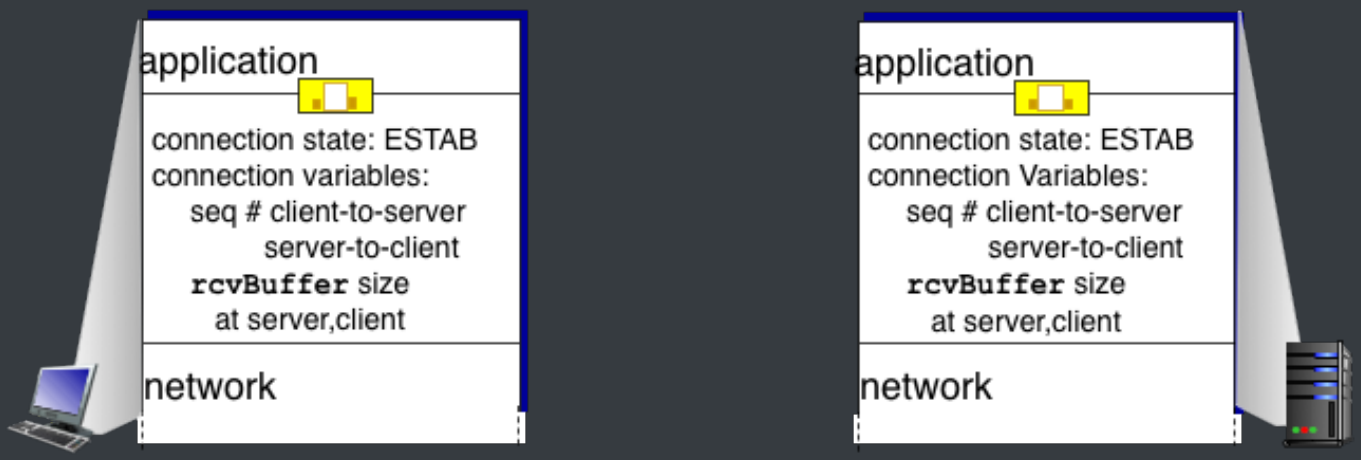
3.5 TCP: connection management[链接管理]

3.85

Connection Management

before exhanging data, sender/receiver "handshake"

1. agree to establish ocnnection (each knowing the other willing to establish connection)
2. agree on connection parameters

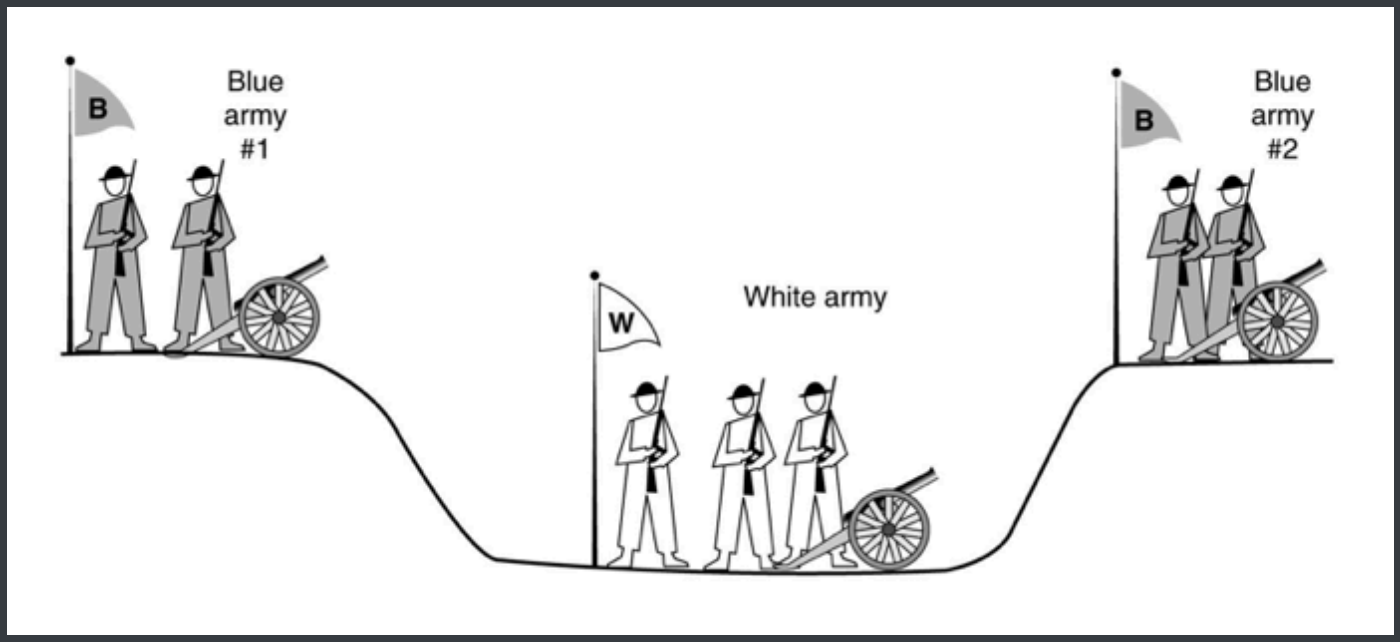



```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

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

3.86

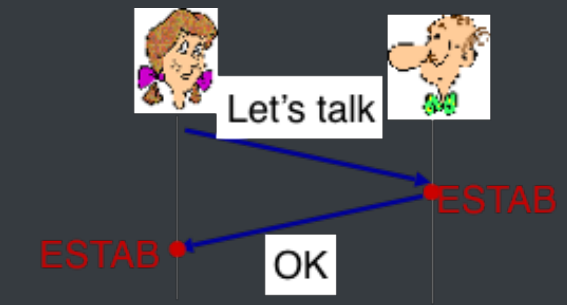
Complexity of Protocol

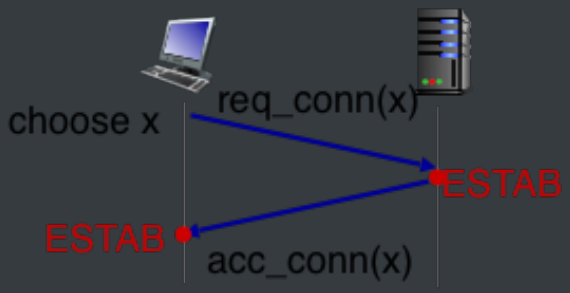


3.87

Agreeing to establish a connection

2-way handshake





Q: Will 2-way handshake always work in network?

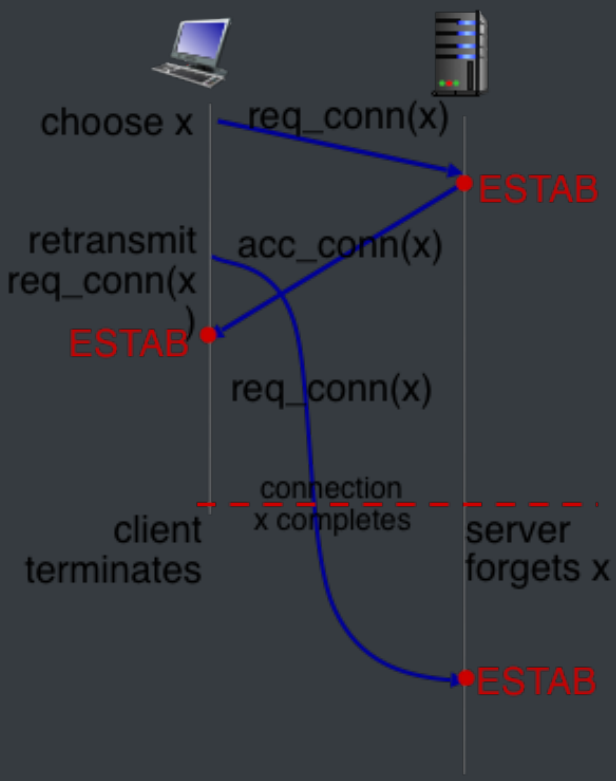
1. [variable delays] variable delays[变化的延迟]
2. [retransmitted message loss] retransmitted messages (e.g. $req - conn(x)$) due to message loss
3. [mesage reordering] message reordering[重新安排]
4. [cannot see] cannot "see" other sider

3.88

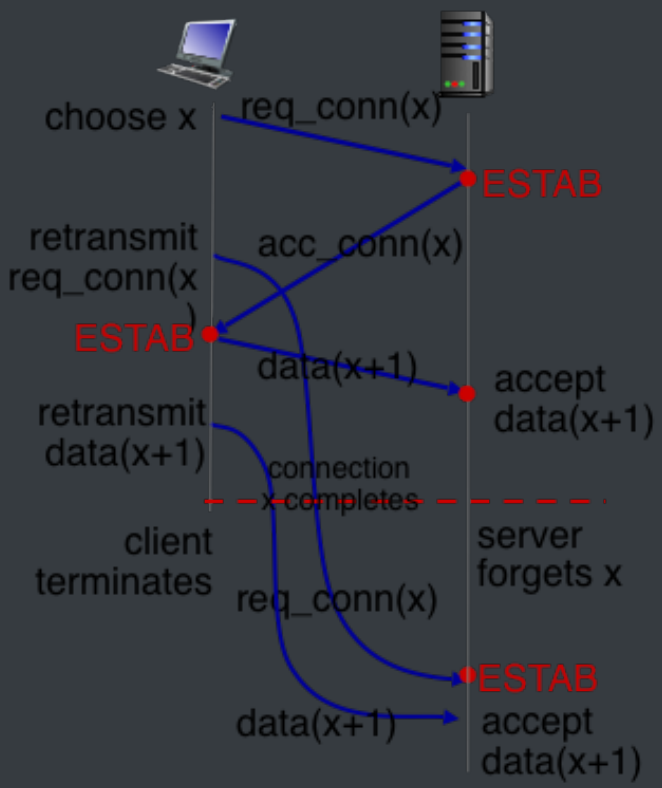
2-way handshake failure scenarios:

1. half open connection! (no client!)

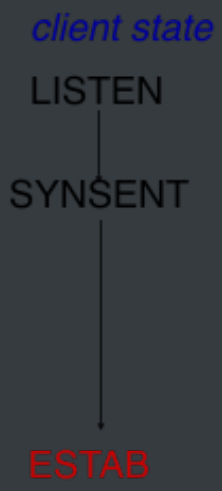
> 提前判断超时, 重发 $req\_conn(x)$ 导致 server 没发 $acc\_conn(x)$

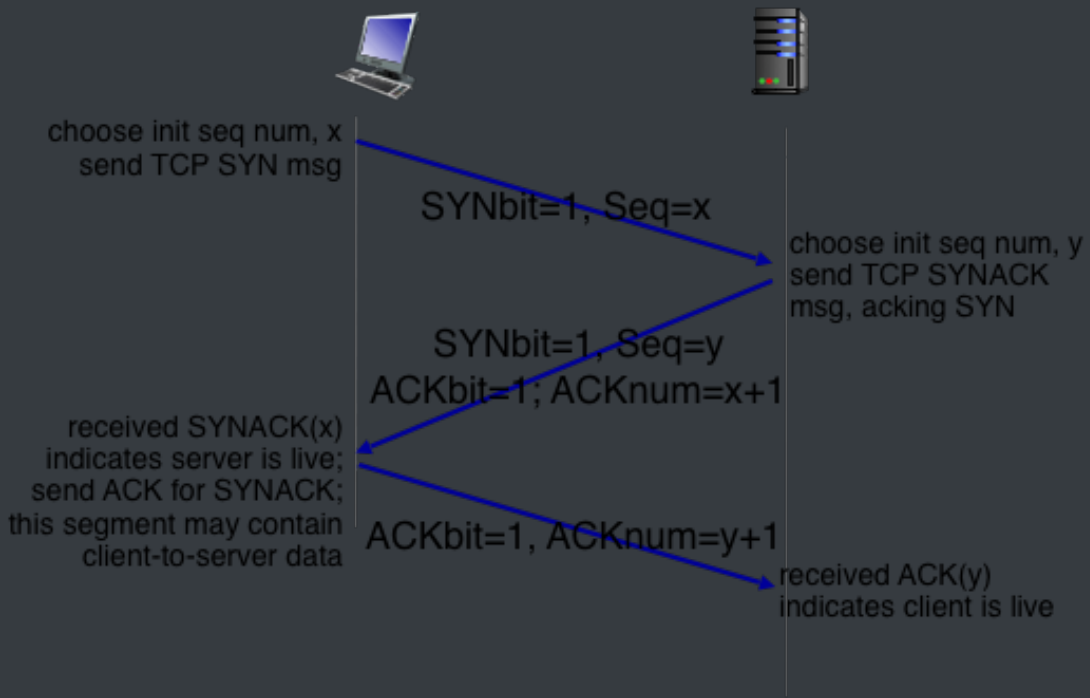


提前判断超时, 重发的 $req\_conn(x)$ 导致 server 没发 $acc\_conn(x)$



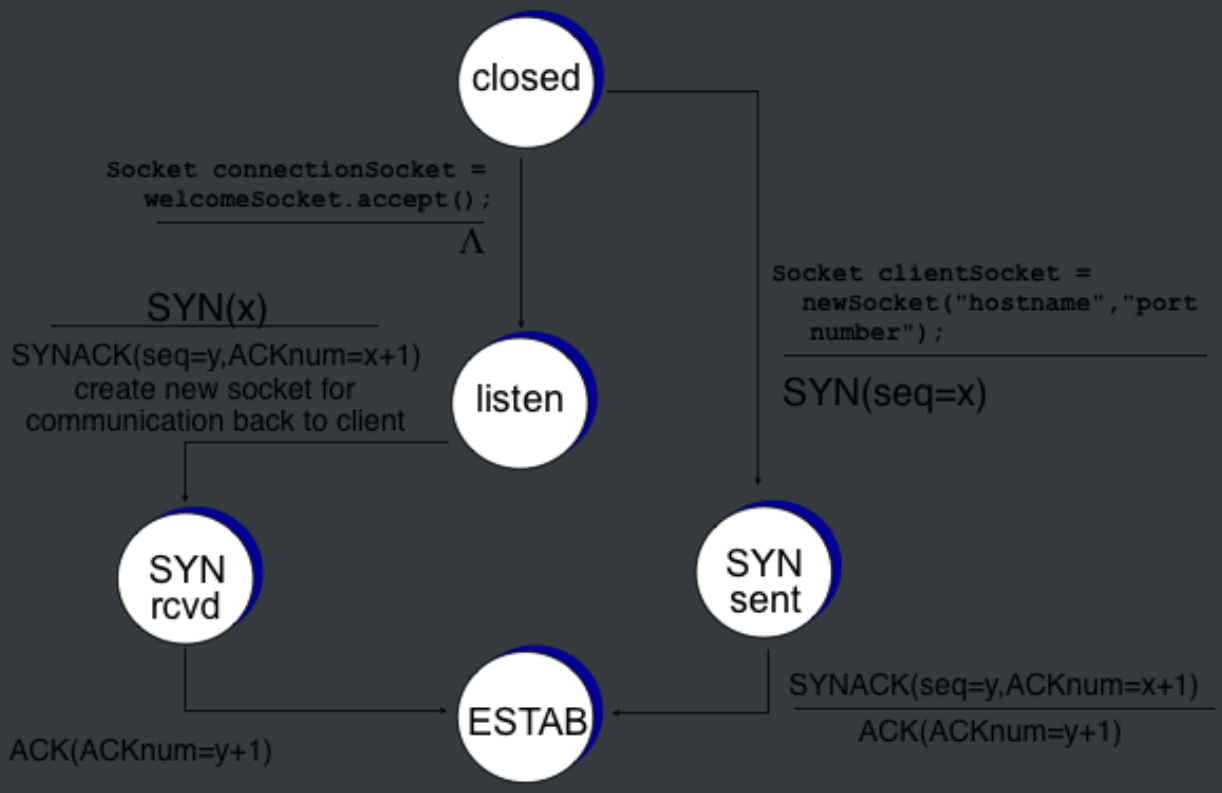
TCP 3-way Handshake





3.90

⭐⭐TCP 3-way Handshake: FSM (Finite State Machine)

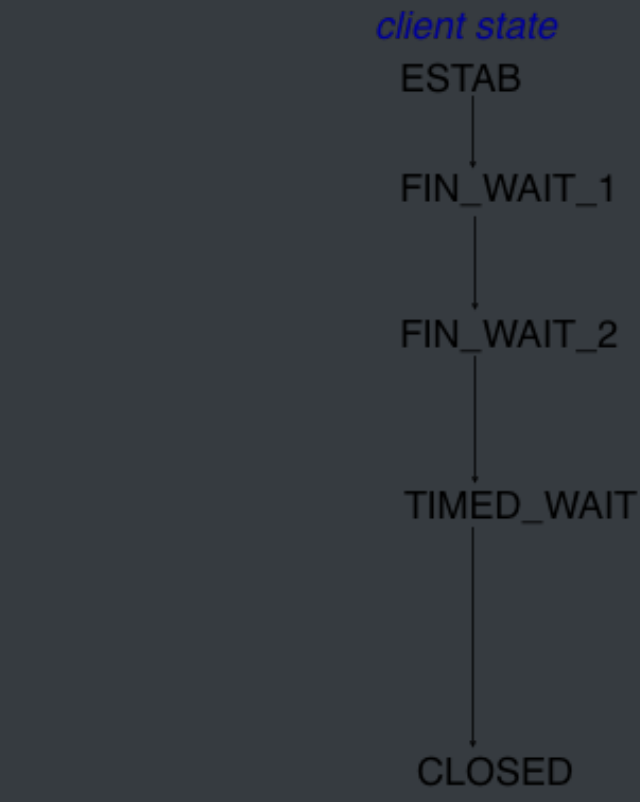


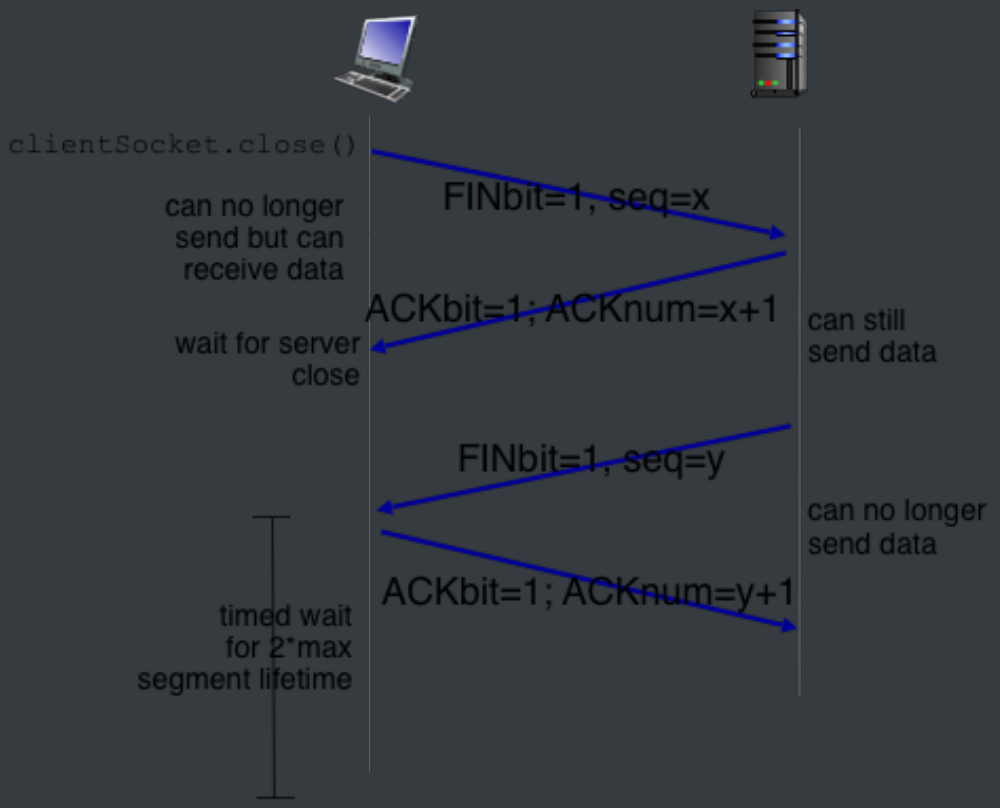
3.91

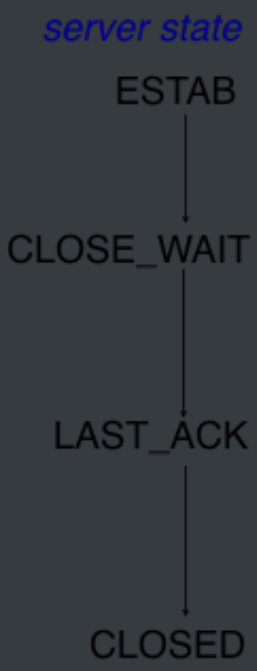
⭐ TCP: closing a connection

1. Client, server each close their side of connection
    1. send TCP segment with FIN bit = 1
2. respond to received FIN with ACK
    1. on receiving FIN, ACK can be combined with own FIN
3. simultaneous FIN exchanges can be handled

3.92


client state
ESTAB
FIN_WAIT_1
FIN_WAIT_2
TIMED_WAIT
CLOSED


clientSocket.close()
FINbit=1, seq=x
can no longer send but can receive data
ACKbit=1; ACKnum=x+1
can still send data
wait for server close
FINbit=1, seq=y
can no longer send data
ACKbit=1; ACKnum=y+1
timed wait for 2*max segment lifetime



3.6 Principles of congestion control

3.93

congestion:

1. informally: “too many sources sending too much data too fast for network to handle”
2. different from flow control!

3. manifestations[表现]:

    3. 1 lost packets (buffer overflow at routers)
    4. 2 long delays (queueing in router buffers)

4. a top-10 problem!

3.94

Principles of congestion control

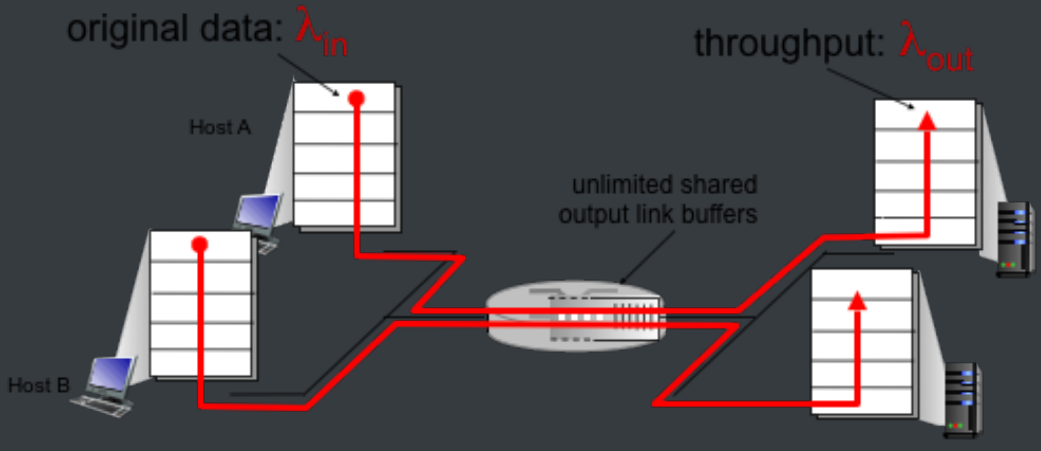
congestion:

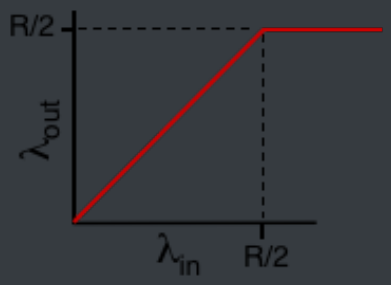
- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
-
    - lost packets (buffer overflow at routers)
    - long delays (queueing in router buffers)
- a top-10 problem!

3.95

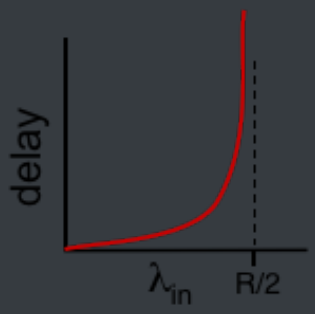
Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- output link capacity: R
- no retransmission





- maximum per-connection throughput: R/2

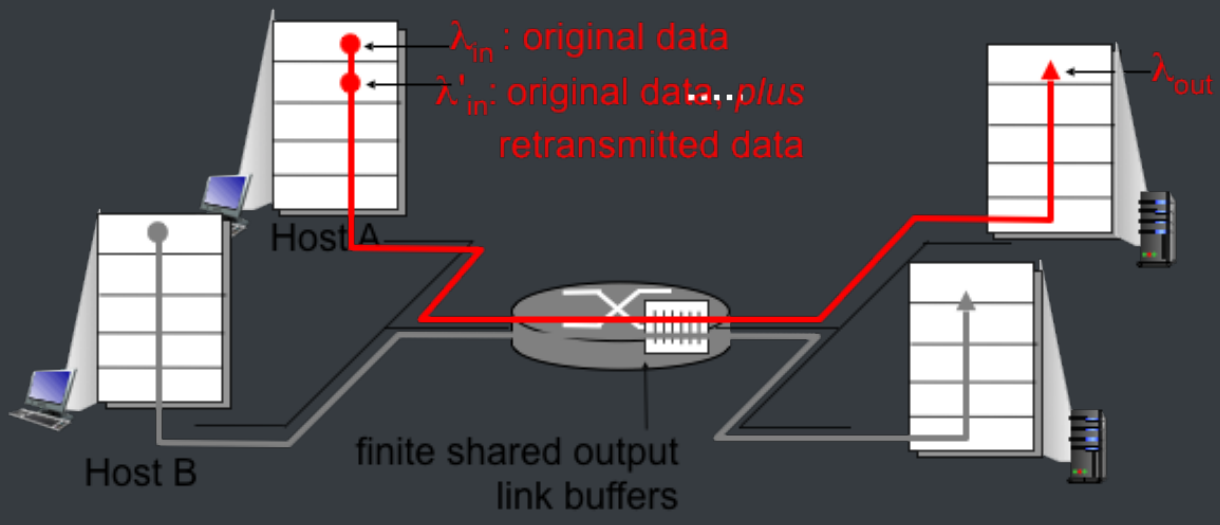


- large delays as arrival rate, λ_{in}, approaches capacity

3.96

Causes/costs of congestion: scenario 2

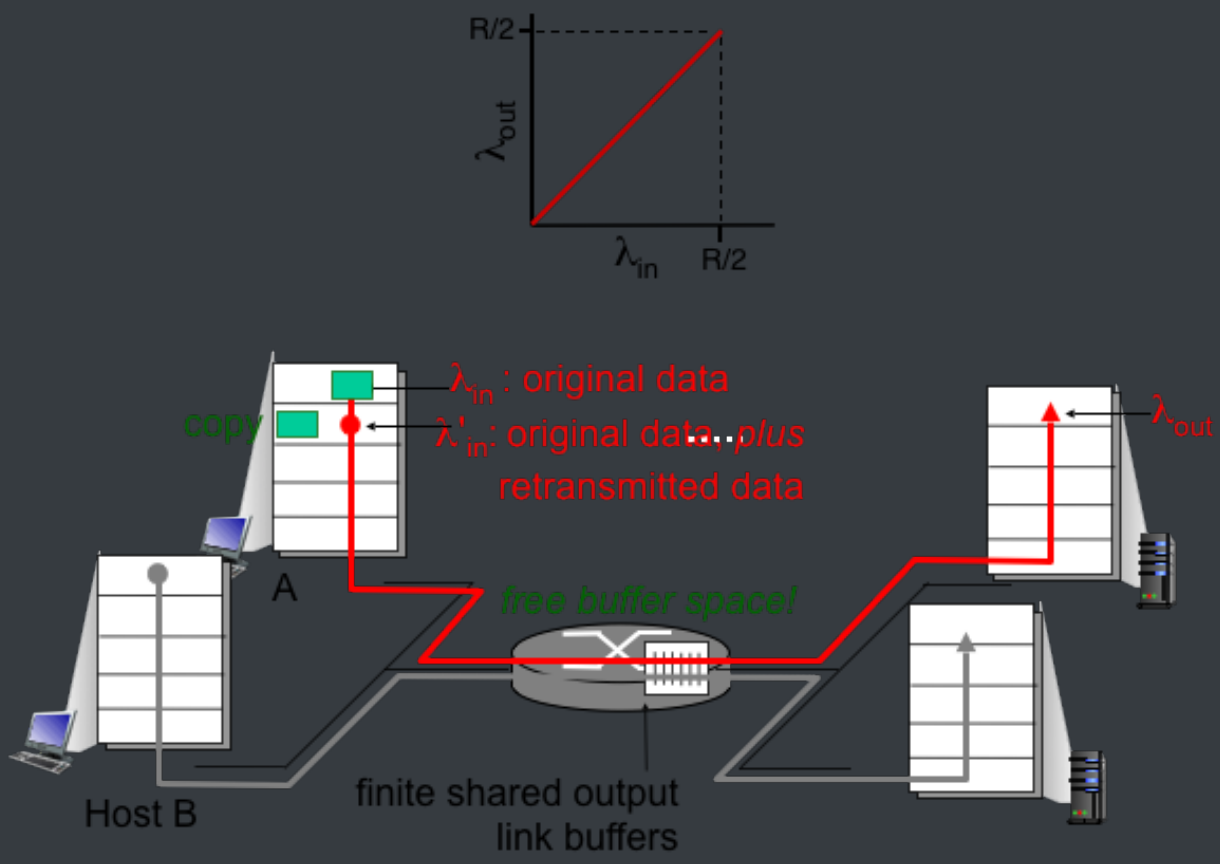
- one router, *finite* buffers
- sender retransmission of timed-out packet
-
  - application-layer input = application-layer output: λin = λout
  - transport-layer input includes *retransmissions* : λin λin



3.97

Causes/costs of congestion: scenario 2

idealization: perfect knowledge sender sends only when router buffers available

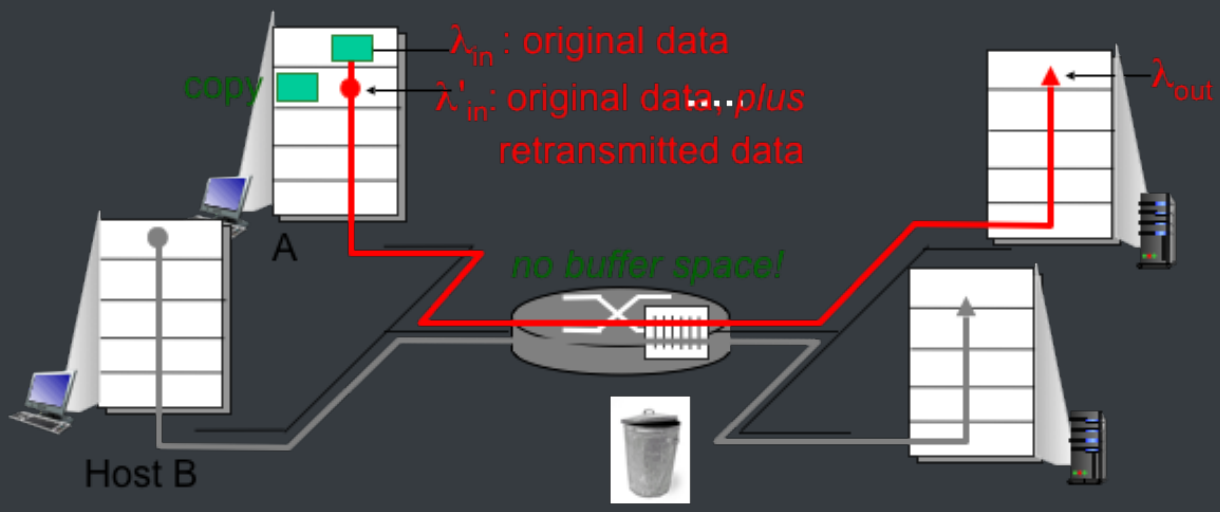


3.98

Causes/costs of congestion: scenario 2

Idealization: known loss packets can be lost, dropped at router due to full buffers

- sender only resends if packet *known* to be lost



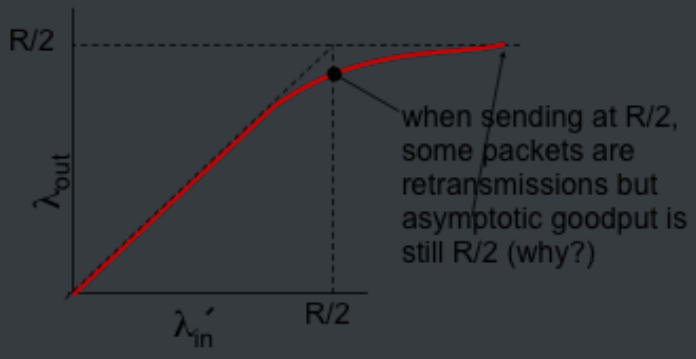
3.99

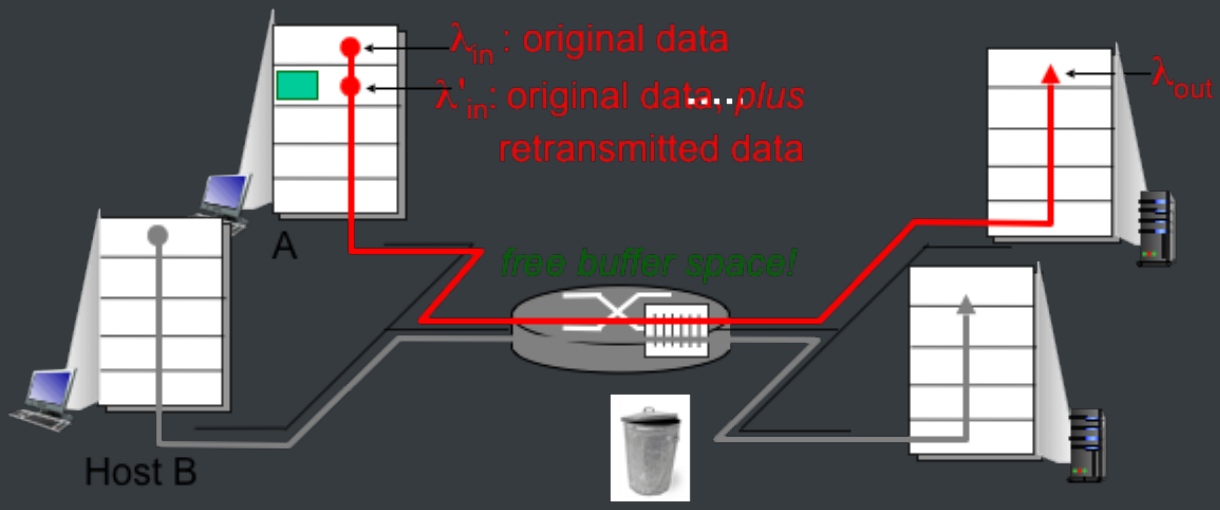
Causes/costs of congestion: scenario 2

Idealization: known loss packets can be lost, dropped at router due to full buffers

- sender only resends if packet *known* to be lost

when sending at R/2, some packets are retransmissions but asymptotic goodput is still R/2 (why?)





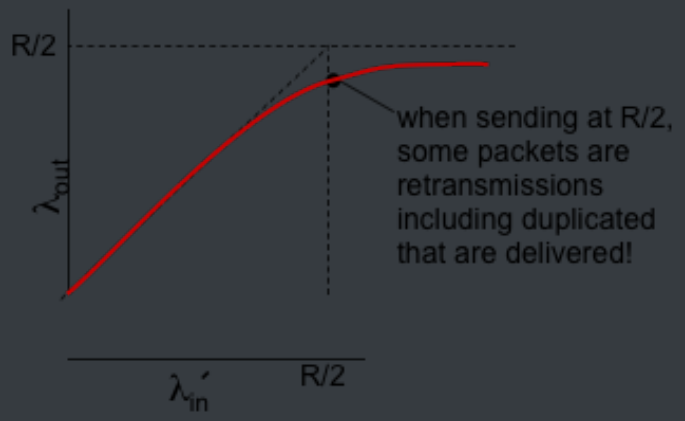
3.100

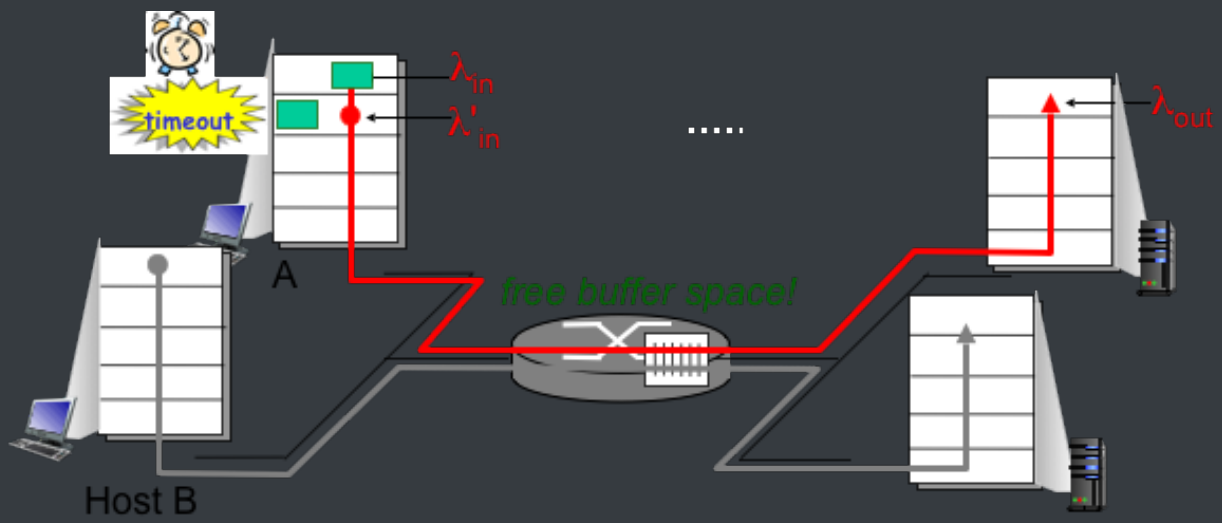
Causes/costs of congestion: scenario 2

Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered

when sending at R/2, some packets are retransmissions including duplicated that are delivered!

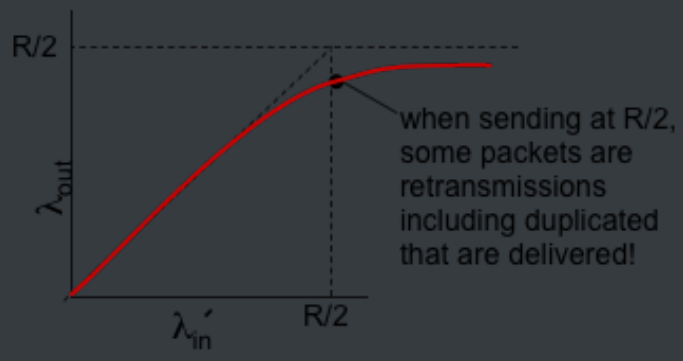




3.101

Causes/costs of congestion: scenario 2

when sending at R/2, some packets are retransmissions including duplicated that are delivered!



Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered

“costs” of congestion:

- more work (retrans) for given “goodput”
- unneeded retransmissions: link carries multiple copies of pkt
- - decreasing goodput

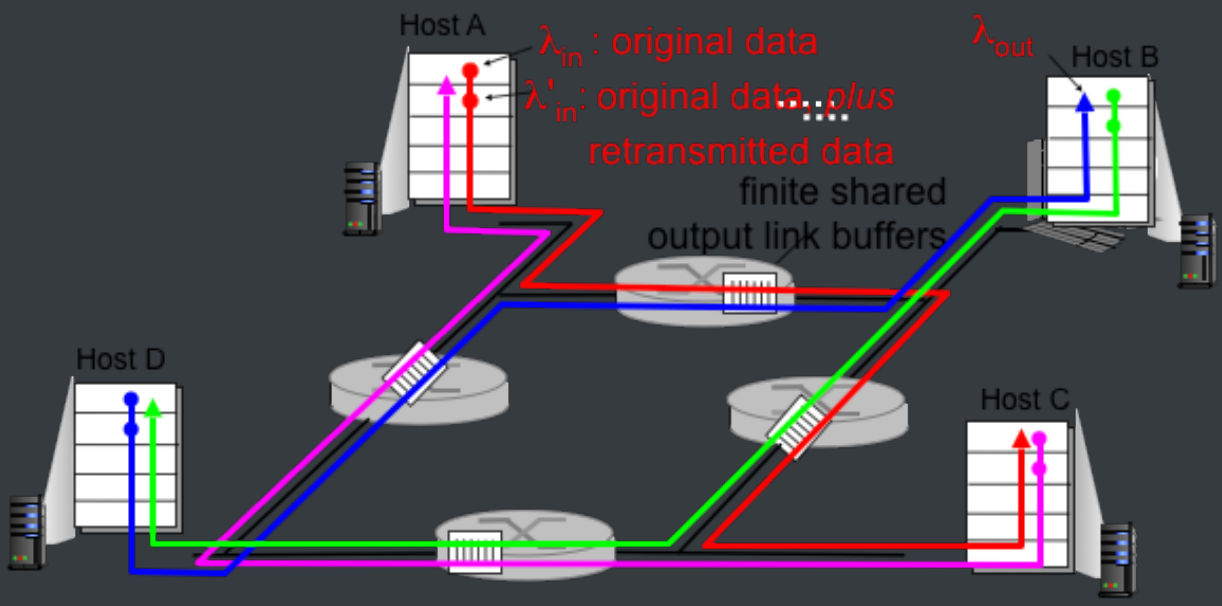
3.102

Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit

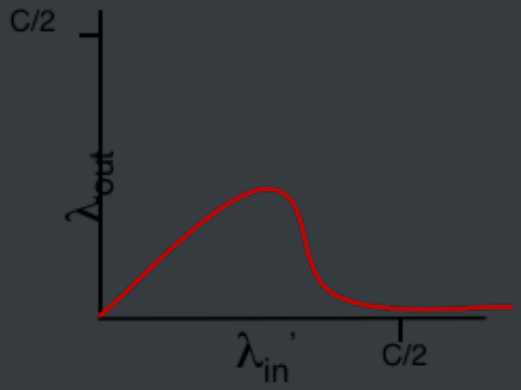
Q: what happens as λin and λin’ increase ?

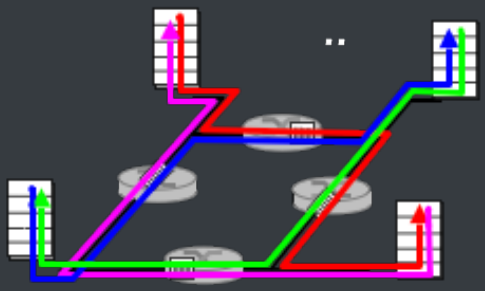
A: as red λin' increases, all arriving blue pkts at upper queue are dropped, blue throughput is 0.



3.103

Causes/costs of congestion: scenario 3





another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

3.104

Approaches towards congestion control[拥塞控制]

two broad approaches towards congestion control:

A. [端对端拥塞控制] end-end congestion control:

1. [no feedback] no explicit feedback from network
2. [loss delay] congestion inferred[推断] from end-system observed loss, delay
3. [taken by TCP] approach taken by TCP

B. network-assisted congestion control:

- routers provide feedback to end systems
- - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate for sender to send at

3.105

Case study: ATM ABR congestion control

ABR: available bit rate[可变位速率]:

- “elastic service”
- if sender’s path “underloaded”:
- - sender should use available bandwidth
- if sender’s path congested:
- - sender throttled to minimum guaranteed rate

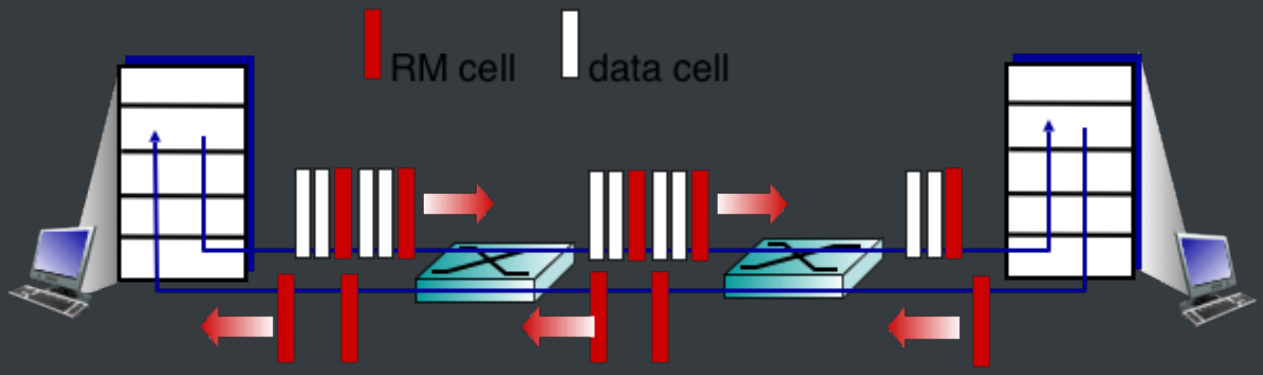
RM (resource management) cells:

- sent by sender, interspersed with data cells
- bits in RM cell set by switches (“*network-assisted*”)

- - *NI bit:* no increase in rate (mild congestion) [不增加速率]
  - *CI bit:* congestion indication [拥塞发生]
- RM cells returned to sender by receiver, with bits intact [控制分组位不变, 发回接收端]

3.106

Case study: ATM ABR congestion control



- two-byte ER (explicit rate) field in RM cell
- - congested switch may lower ER value in cell
  - senders' send rate thus max supportable rate on path
- EFCI bit in data cells: set to 1 in congested switch
- - if data cell preceding RM cell has EFCI set, receiver sets CI bit in returned RM cell

3.7 TCP congestion control

3.108 end-end control (no network assistance)

- sender limits transmission:

LastByteSent-LastByteAcked ≤ Cwnd

Roughly,

$$rate = \frac{cwnd}{RTT} Bytes/sec \tag{10}$$

How does sender perceive congestion?

1. loss event = timeout *or* 3 duplicate acks
2. TCP sender reduces rate (**Cwnd**) after loss event
3. **Cwnd** is dynamic, function of perceived network congestion

three mechanisms:

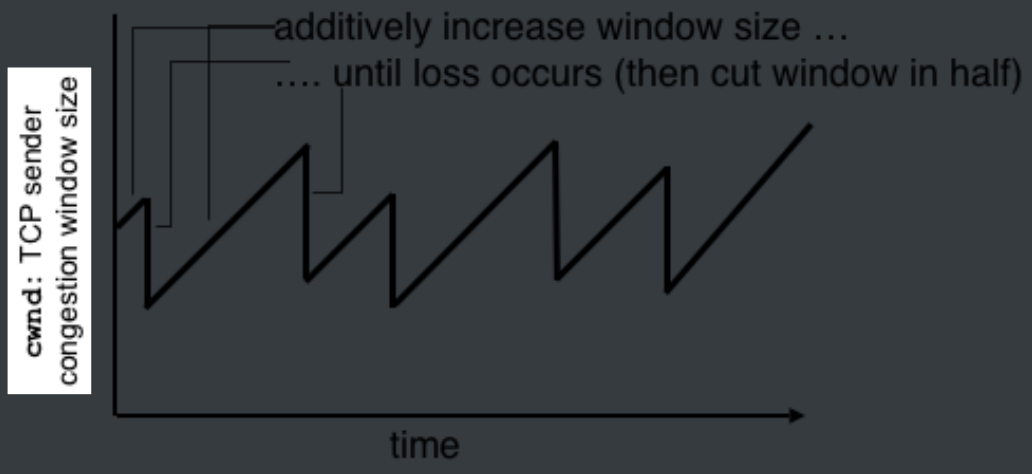
1. AIMD
2. slow start
3. conservative after timeout events

3.109

TCP congestion control: additive increase multiplicative decrease

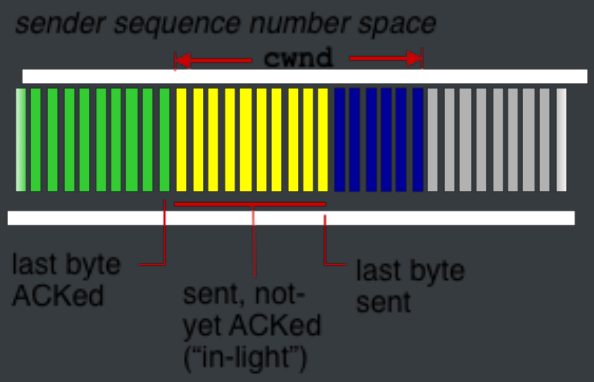
approach: sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs additive increase: increase cwnd by 1 MSS every RTT until loss detected multiplicative decrease: cut cwnd in half after loss

AIMD saw tooth behavior: probing for bandwidth



3.110

TCP Congestion Control: details



sender limits transmission

$$LastByteSent - LastByteAcked \leq cwnd[congestionwindow] \quad (11)$$

cwnd is dynamic, function of perceived network congestion

TCP sending rate: roughly: send cwnd bytes, wait RTT for ACKS, then send more bytes

$$rate \approx \frac{cwnd}{RTT} bytes/sec \quad (12)$$

3.111

TCP Slow Start

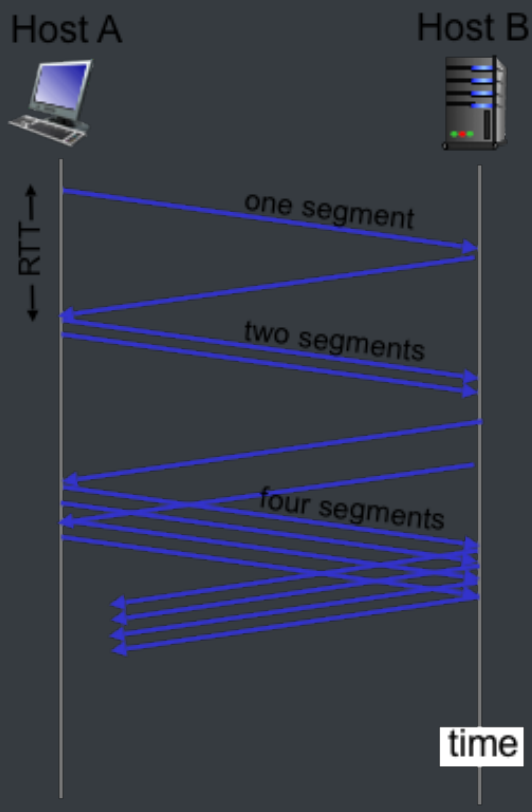
when connection begins, increase rate exponentially until first loss event:

initially cwnd = 1 MSS

double cwnd every RTT

done by incrementing cwnd for every ACK received

summary: initial rate is slow but ramps up exponentially fast



3.112

TCP: detecting, reacting to loss

loss indicated by timeout:

cwnd set to 1 MSS;

window then grows exponentially (as in slow start) to threshold, then grows linearly

loss indicated by 3 duplicate ACKs: TCP RENO

dup ACKs indicate network capable of delivering some segments

ssthresh is cut in half window

cwnd set to ssthresh+ 3*MSS then grows linearly

TCP Tahoe always sets cwnd to 1 (timeout or 3 duplicate acks)

3.113

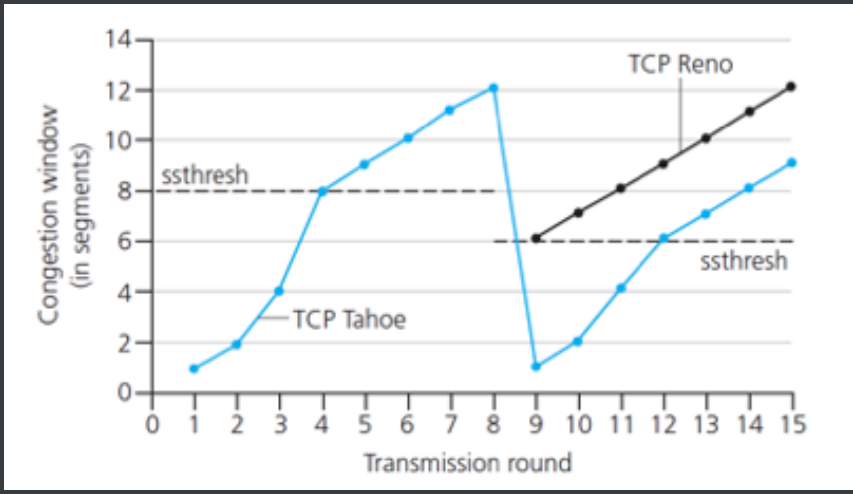
TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

A: when cwnd gets to 1/2 of its value before timeout.

Implementation:

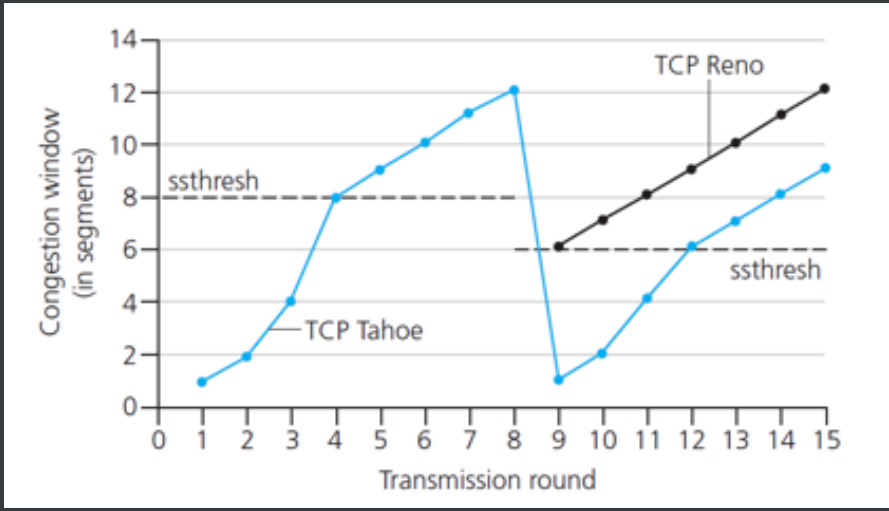
1. variable ssthresh
2. on loss event, ssthresh is set to 1/2 of cwnd just before loss event



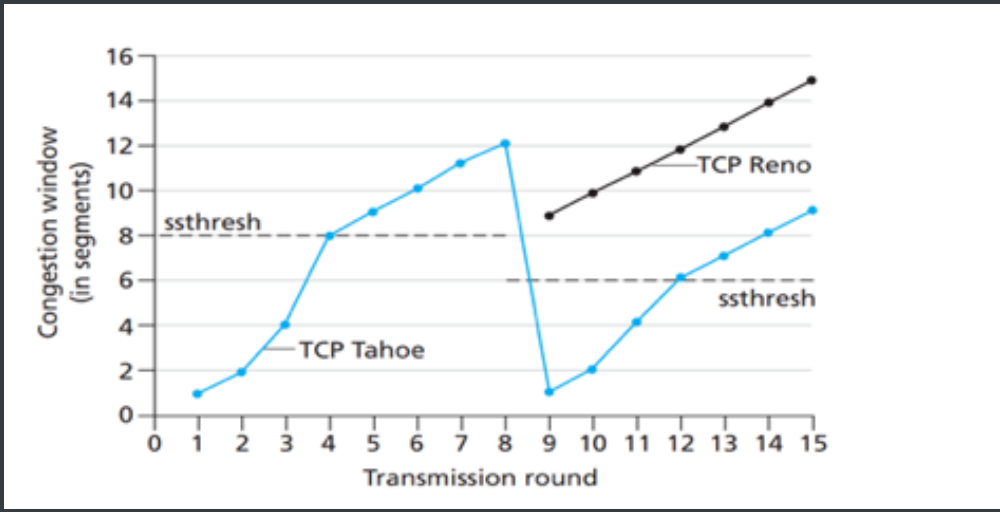
3.114

Diffrent states

congestion avoidance

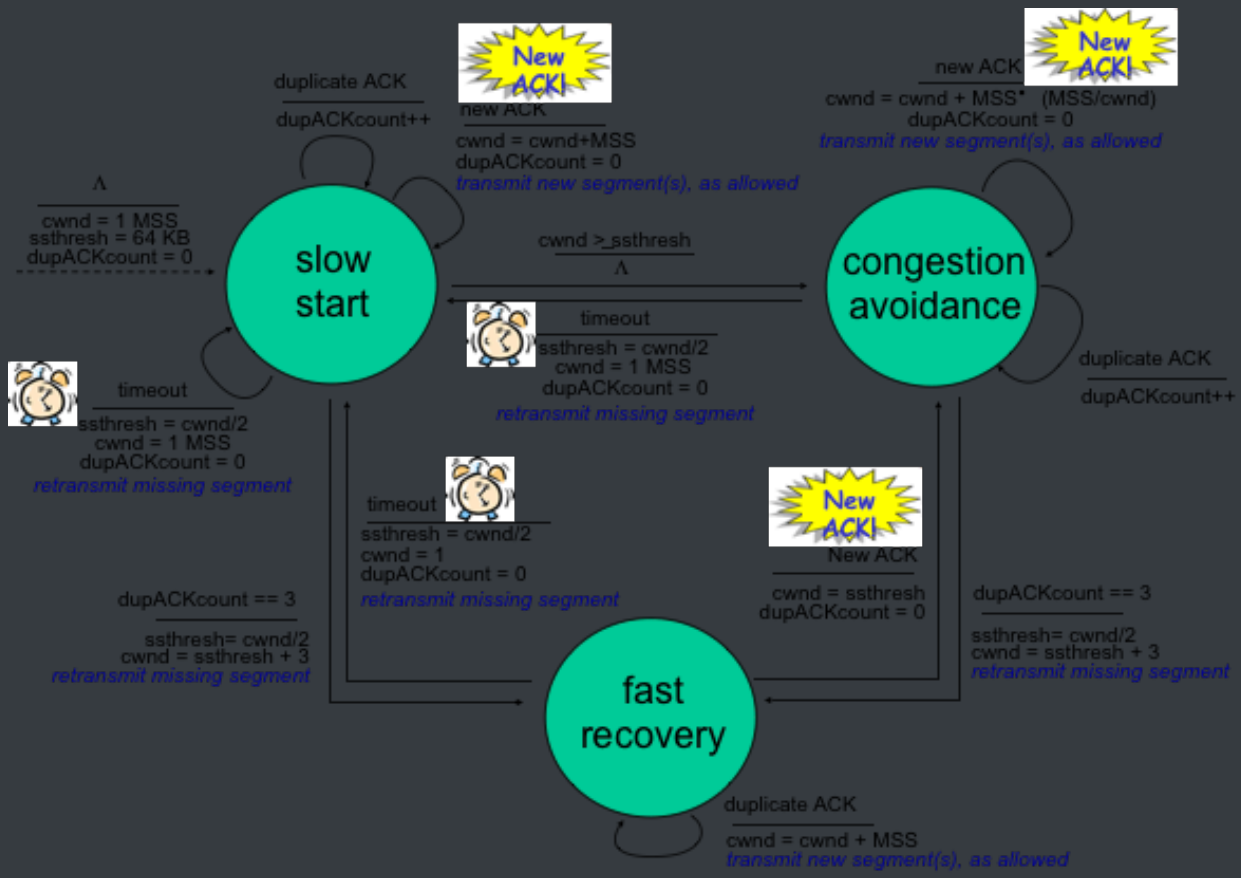


Fast recovery： cwnd = ssthresh + 3



3.115

Summary: TCP Congestion Control



3.116

TCP throughput

avg. TCP thruput as function of window size, RTT?

- ignore slow start, assume always data to send

W: window size (measured in bytes) where loss occurs

- avg. window size (# in-flight bytes) is ¾ W
- avg. thruput is 3/4W per RTT

3.117

TCP Futures: TCP over “long, fat pipes”

example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput

requires W = 83,333 in-flight segments

throughput in terms of segment loss probability, L [Mathis 1997]:

$$TCPthroughput = \frac{1.22 \cdot MSS}{RTT\sqrt{L}} \qquad (13)$$

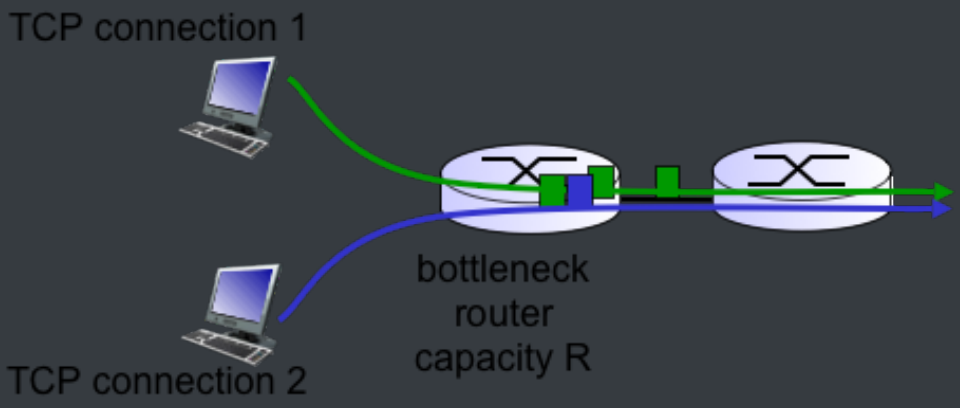
➜ to achieve 10 Gbps throughput, need a loss rate of L = 2·10-10 – a very small loss rate!

new versions of TCP for high-speed

3.118

TCP Fairness

fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



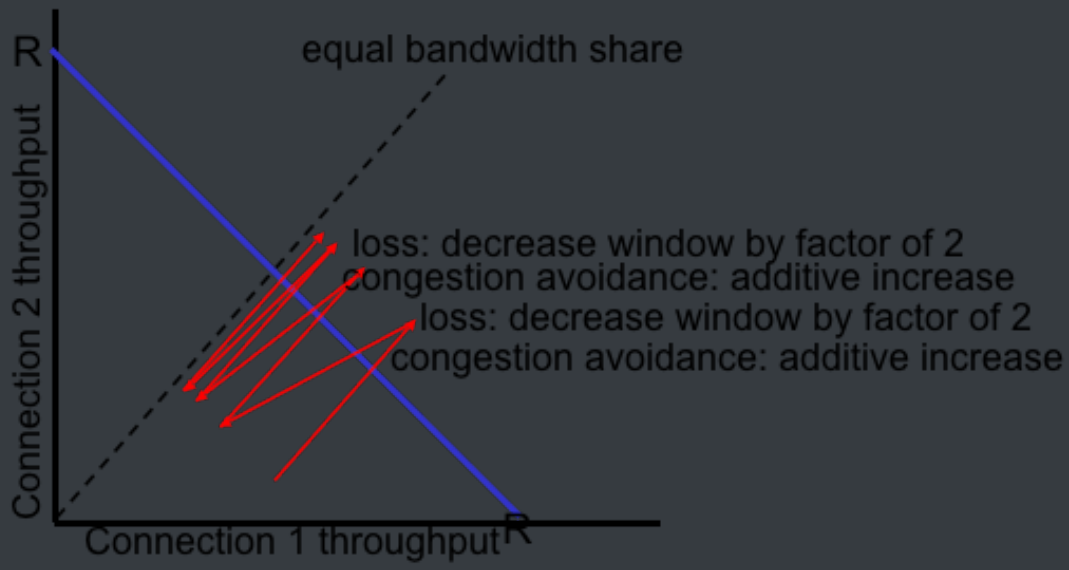
3.119

Why is TCP fair?

two competing sessions:

additive increase gives slope of 1, as throughout increases

multiplicative decrease decreases throughput proportionally



3.120

Fairness (more)

Fairness and UDP

multimedia apps often do not use TCP

- do not want rate throttled by congestion control

instead use UDP:

- send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

1. application can open multiple parallel connections between two hosts
2. web browsers do this
3. e.g., link of rate R with 9 existing connections:

- 3. 1 new app asks for 1 TCP, gets rate R/10
- 3. 2 new app asks for 11 TCPs, gets R/2

Chapter 3: summary

- principles behind transport layer services:
- - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
- - UDP

    - TCP

next:

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

Chapter 4 Network Layer

4.2

Chapter goal

understand principles behind network layer services:

1. netwrok layer service models [网络服务模型]
2. forwarding versus routing [转发路由选择]
3. how a router workd [路由器工作原理]
4. routing (path selection) [路由选择算法]
5. boradcast multicast [广播和多播]

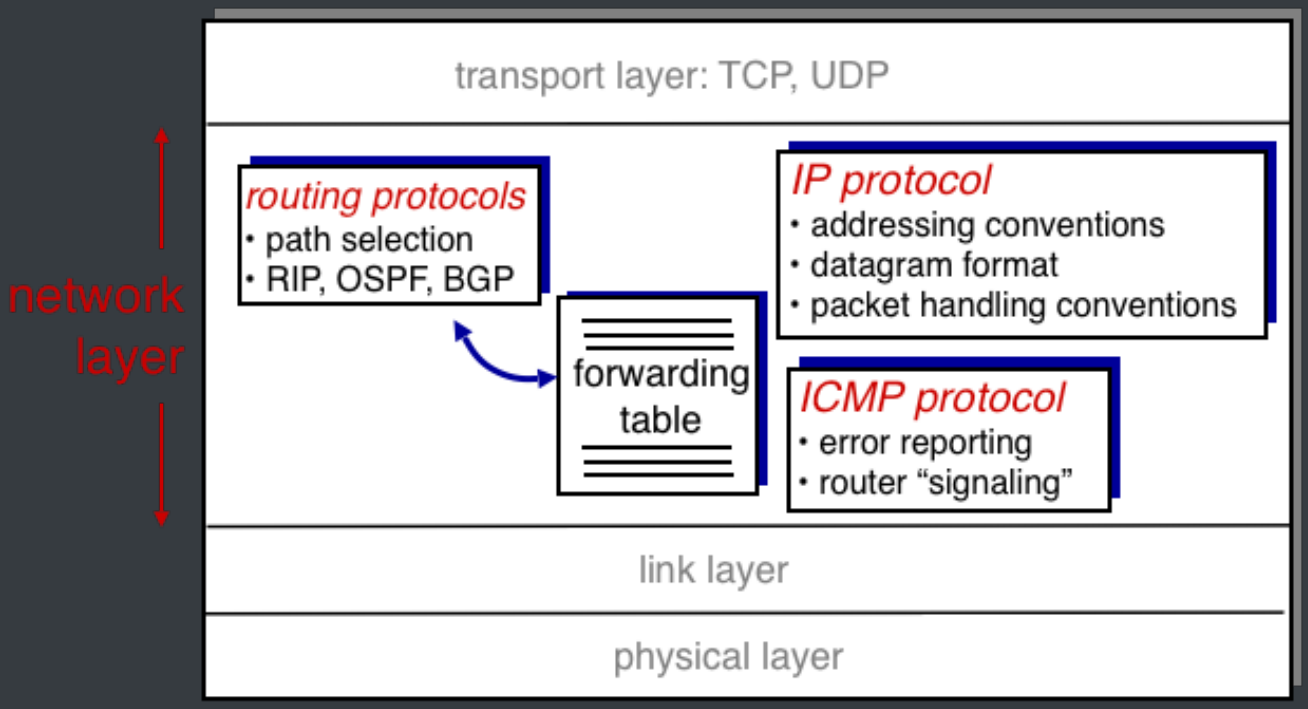
4.4 IP: Datagram Format

1. datagram format [数据包格式]
2. IPV4 addressing [Internet Protocol version 4 选址]
3. ICMP [Internet Control Message Protocol]
4. IPV6 [Internet Protocol version 6]

4.33

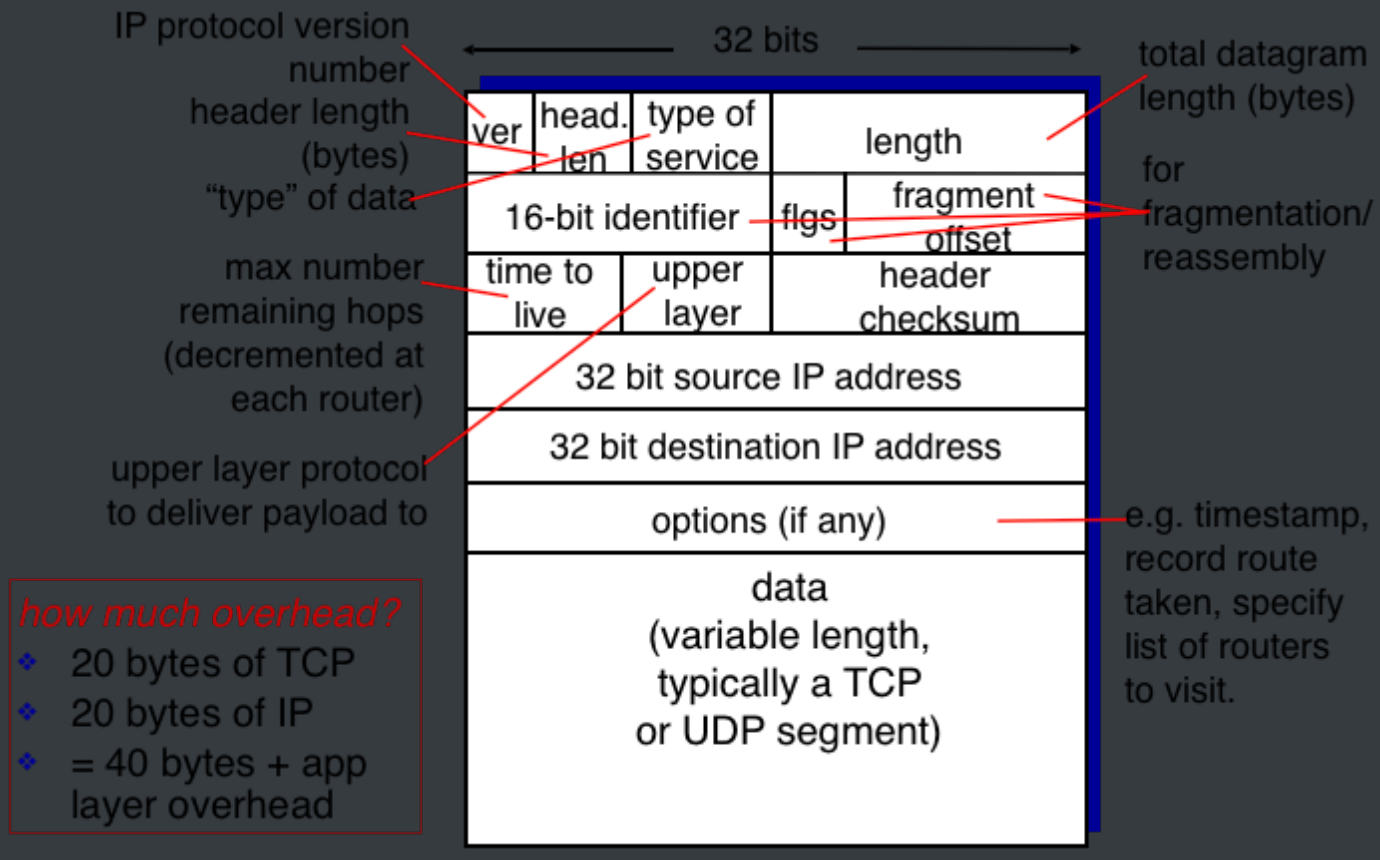
> 肖云：网络层三大组件

host, router network layer functions:



> ipv5 时间非常短 length 首部长度 20length +40options （ 0 - 60 bytes ）， 最长可以达到60bytes？ 20 btes TTL time to live: 寿命。一直在找主机host， 会造成资源浪费， 设置最大的跳数。每过一个路由器 - 1， 到达最大跳数不在往前传了。upper layer 上层协议， 6是UDP

4.34



4.35

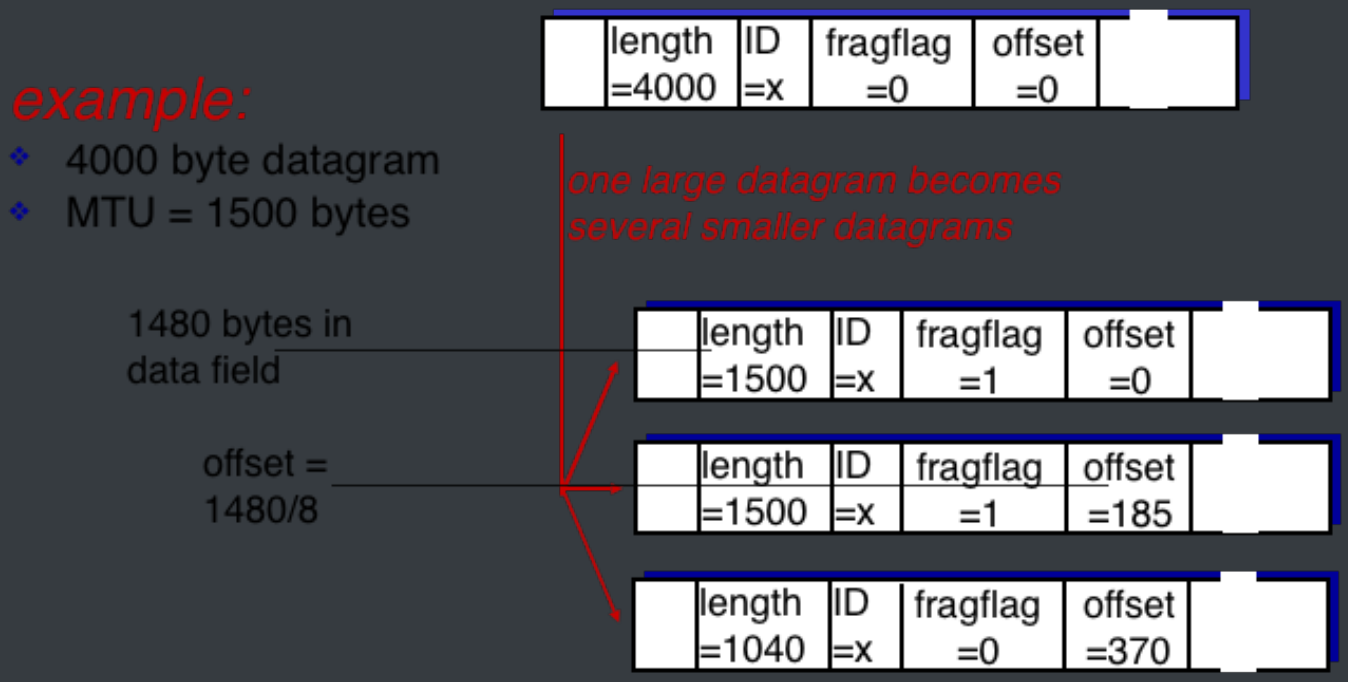
IP Fragmentation, reassembly [IP 数据报分片]

1. network lins have MTU (maximum transfer size) - largest possible link-level fram
   - different likn types, diffreent MTU

2. Large IP datagram divided (" fragmented") within net
    1. one datagram becomes several datagrams
    2. "reassembled" only at final destination
    3. IP header bits used to identify, order related fragments

4.36

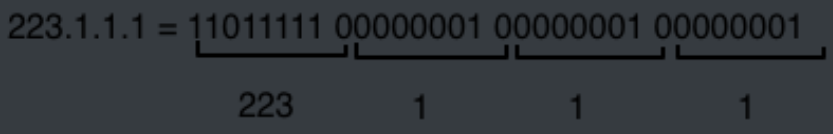
IP fragmentation, reassembly [IP 数据报分片]



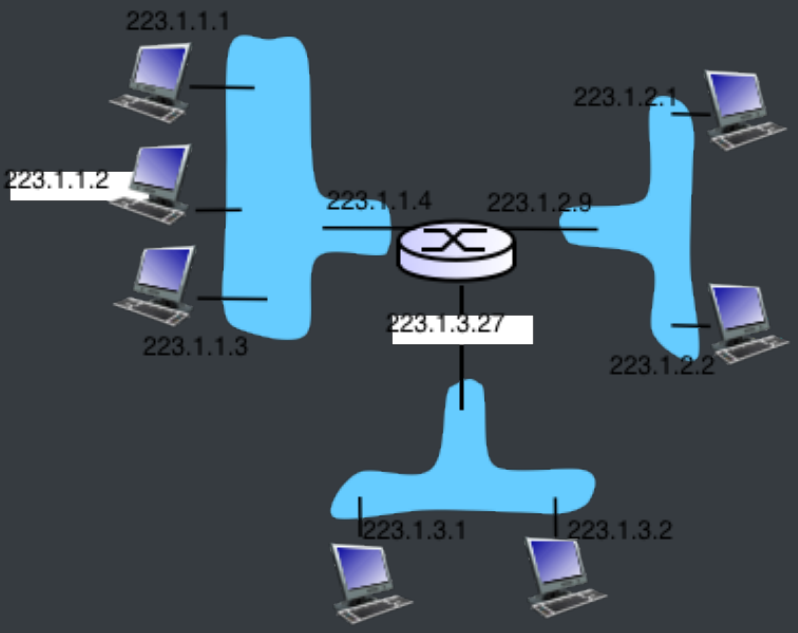
4.4 IP:IPv4 Addressing [IP 地址]

4.38

1. IP Address: 32-bit identifier for host, router interface
2. Interface: connection between host/router and physical link
    1. router's typically have multiple interfaces
    2. host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
3. IP addresses associated with each interface

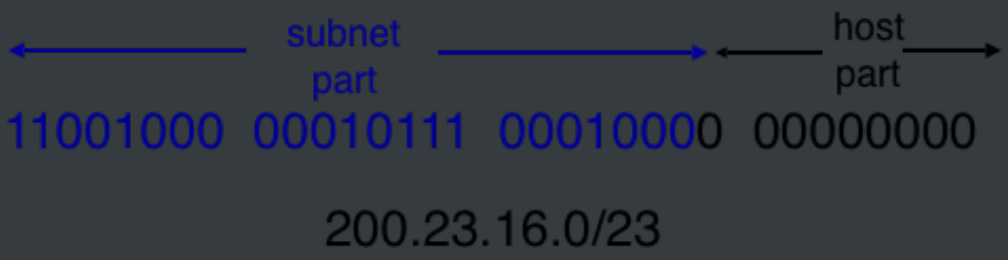


> 3个蓝色阴影 - 3个子网 ipv4 - 32bits 2^{32} ipv6 - 40bits 2^{40}



4.43

IP addressing: CIDR[**Classless Interdomain Routing**]



4.44

IP addresses: how to get one?

Q: How does a *host* get IP address?

- hard-coded by system admin in a file
    - Windows: control-panel->network->configuration->tcp/ip->properties
    - UNIX: /etc/rc.config
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
    - “plug-and-play”

4.45

DHCP: Dynamic Host COnfiuration Protocol

goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/“on”)
- support for mobile users who want to join network (more shortly)

DHCP overview:

- host broadcasts “DHCP discover” msg [optional]
- DHCP server responds with “DHCP offer” msg [optional]
- host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg

4.46

4.47

4.48

4.49

4.50

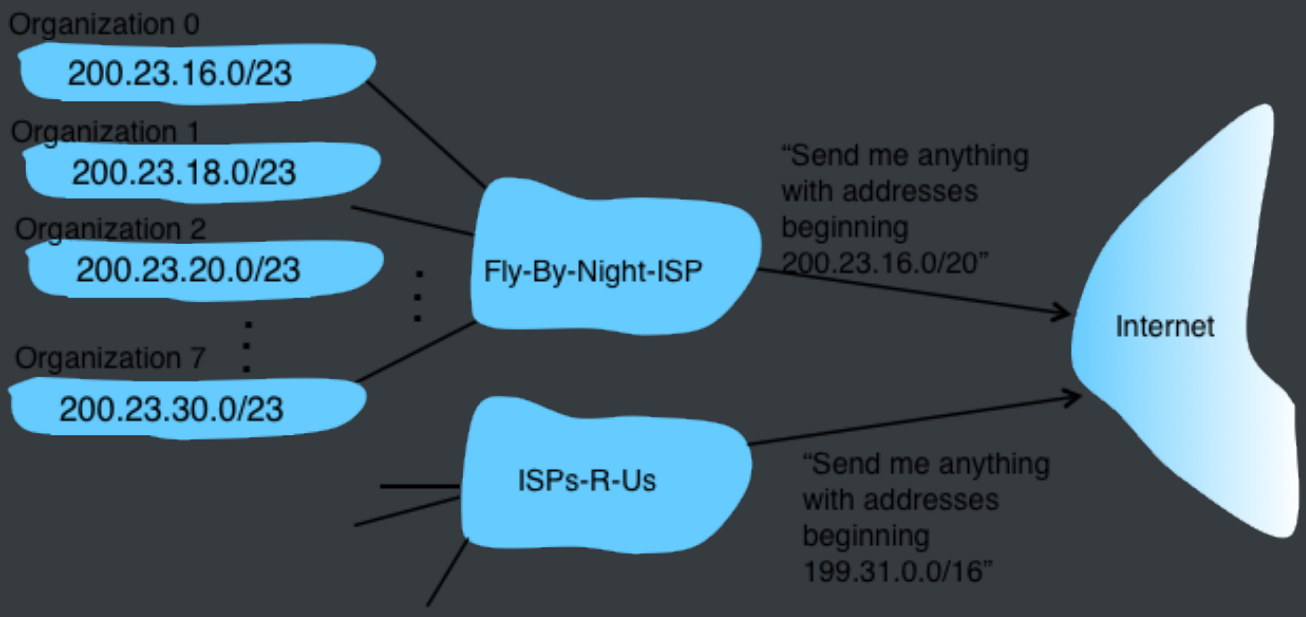
4.51

4.52

4.53

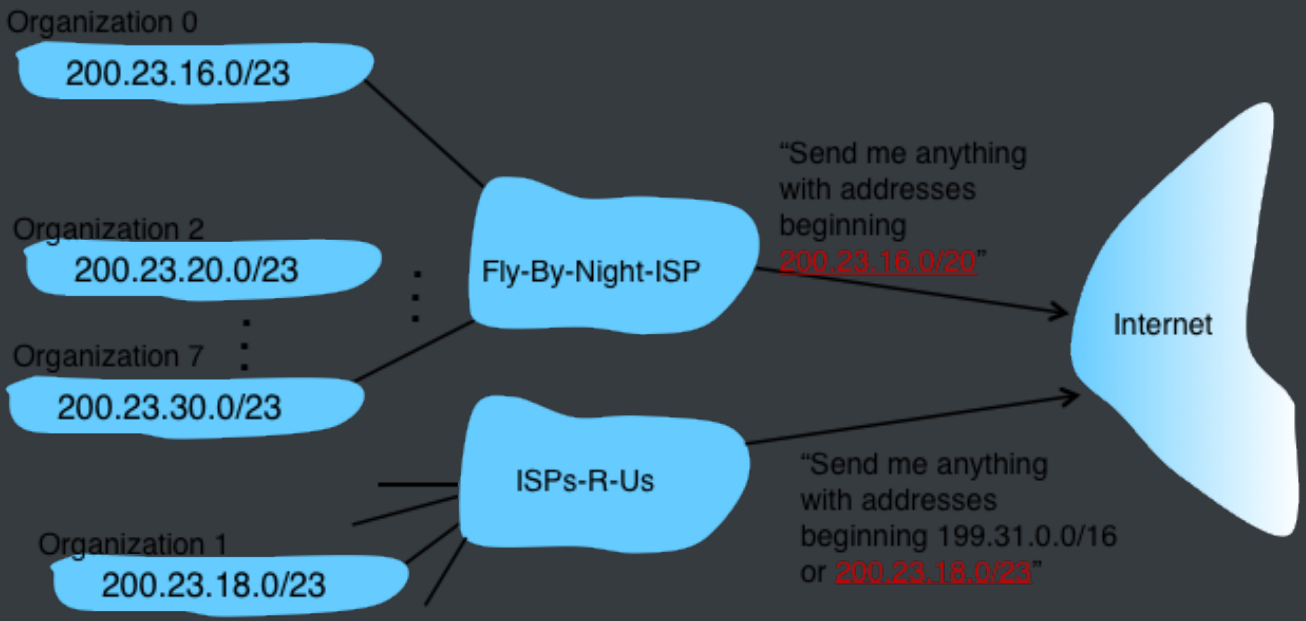
Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



4.54

ISPs-R-Us has a more specific route to Organization 1



4.55

IP addressing: the last word...

Q: how does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned

Names and Numbers http://www.icann.org/

- - allocates addresses
  - manages DNS
  - assigns domain names, resolves disputes

4.56

4.57

4.58

4.59

4.60

4.61

4.62

4.63

4.4 IP:ICMP, IPv6

4.65

ICMP: internet control message protocol

used by hosts & routers to communicate network-level information error reporting: unreachable host, network, port, protocol echo request/reply (used by ping) network-layer “above” IP: ICMP msgs carried in IP datagrams ICMP message: type, code plus first 8 bytes of IP datagram causing error

Type Code description
0 0 echo reply (ping)
3 0 dest. network unreachable
3 1 dest host unreachable
3 2 dest protocol unreachable
3 3 dest port unreachable
3 6 dest network unknown
3 7 dest host unknown
4 0 source quench (congestion
control - not used)

4.66

Traceroute and ICMP

- source sends series of UDP segments to dest

- - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- when *n*th set of datagrams arrives to nth router:
- - router discards datagrams
  - and sends source ICMP messages (type 11, code 0)
  - ICMP messages includes name of router & IP address
- when ICMP messages arrives, source records RTTs

stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP “port unreachable” message (type 3, code 3)
- source stops

4.67

IPV6: motivation [研究动机]

initial motivation: 32-bit address space soon to be completely allocated.

additional motivation:

1. header format helps speed processing/forwarding
2. header changes to facilitate QoS

IPv6 datagram format:

1. fixed-length 40 byte header
2. no fragmentation allowed

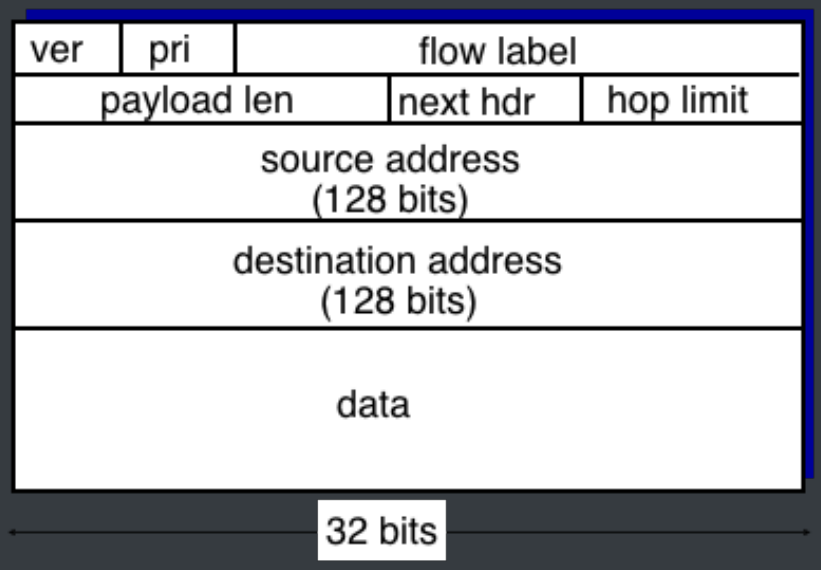
4.68

IPv6 datagram format

priority: identify priority among datagrams in flow

flow Label: identify datagrams in same “flow.” (concept of“flow” not well defined).

next header: identify upper layer protocol for data



4.69

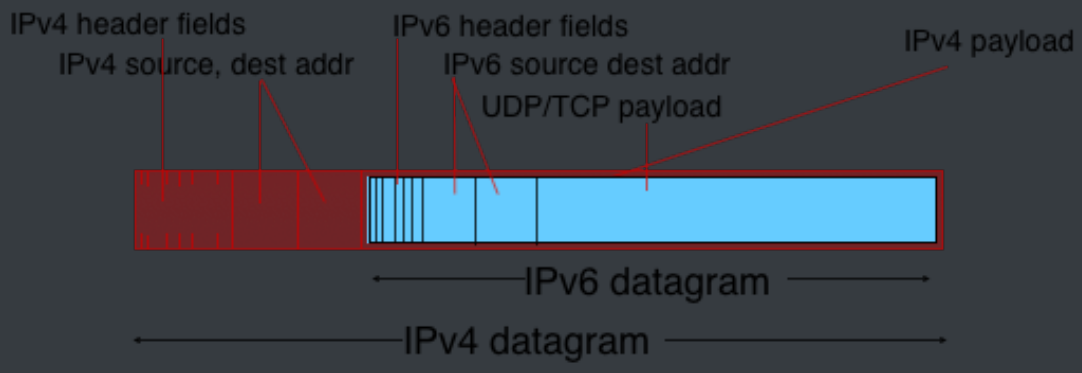
- *checksum*: removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by “Next Header” field
- *ICMPv6:* new version of ICMP
-
    - additional message types, e.g. “Packet Too Big”
    - multicast group management functions

4.70

Transition from IPv4 to IPv6

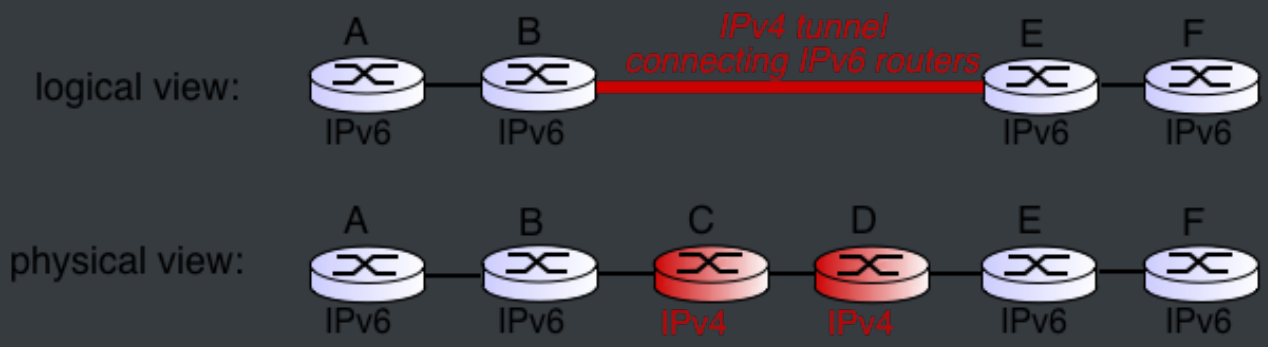
> 固定长度： 40bytes IPv6 数据报更短一点

1. not all routers can be upgraded simulateously
    1. no flag days
    2. how will network operate with IPv4 to IPv6
2. tunelling: IPv6 datagram carried as payload in IPv4 datagram amoung IPv4 routers



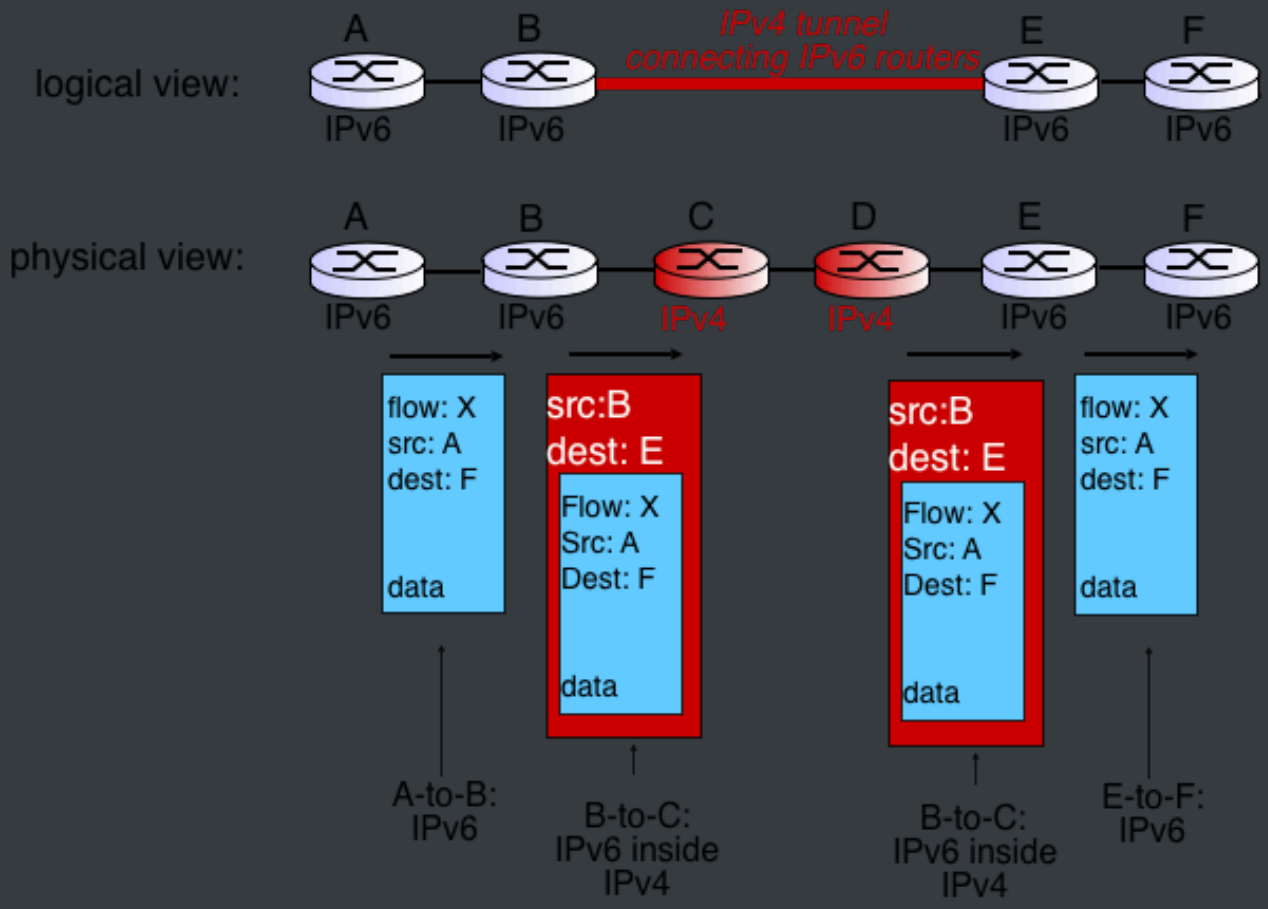
4.71

Tunneling



4.72

Tunneling



4.73

IPv6: adoption

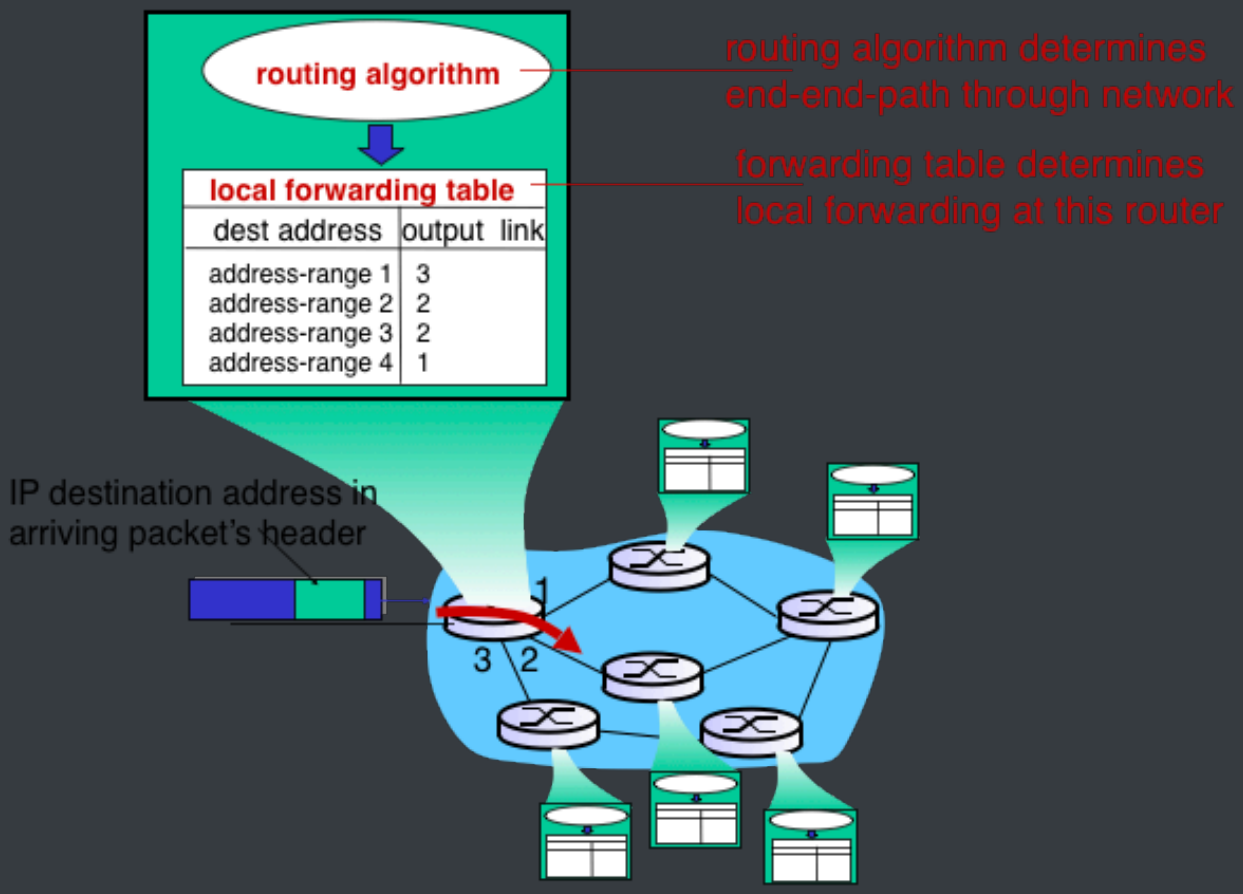
- US National Institutes of Standards estimate [2013]:
  - ~3% of industry IP routers
  - ~11% of US gov't routers

- *Long (long!) time for deployment, use*
  - 20 years and counting!
  - think of application-level changes in last 20 years: WWW, Facebook, …
  - *Why?*

4.5 Routing Algorithms: Link State

4.75

Interplay between routing, forwarding



4.76

Graph abstraction

graph: G = (N,E)

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

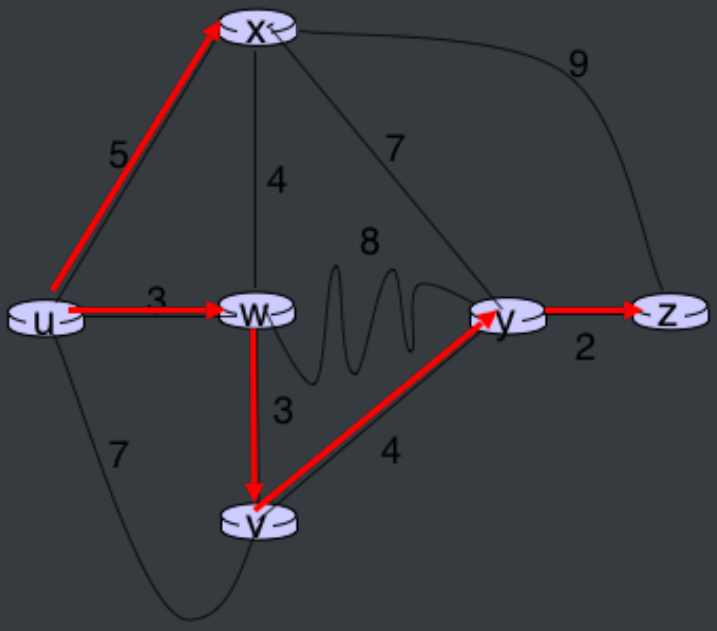
E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

4.78

4.82

> N' [节点集合， 不是路径]
>
> > 每个点的邻近一个路径

N' [节点集合， 不是路径]	D(v)	D(w)	D(x)	D(y)	D(z)
{ u }	7,u	3,u	5,u	∞	∞
{ uw } (3, u)	6(3+3) ,w	3, u	7(3+4), w; **[5, u]** < 7(3+4), w	11(3+ 8), w	∞
{ uwx } (5, u)	∞; [6(3+3) ,w] < ∞	9 (5+4), x	5, u	12(5 + 7), x; [11(3+ 8), w] < 12(5 + 7), x;	14 (5+9), x



0. 准备开始

N' [节点集合， 不是路径]	D(v)	D(w)	D(x)	D(y)	D(z)
∞	∞	∞	∞	∞	∞

1. 第一个节点集合的最短路计算 （ 邻近1个节点，顺序可以打乱）

1.1 第一个节点集合的最短路比较 （ 邻近1个节点，顺序可以打乱）

1.2 第一个节点集合的最短路结果 （ 邻近1个节点，顺序可以打乱）

1.2 第一个节点集合的最短路去除已经用的值 （ 邻近1个节点，顺序可以打乱）

4.5 Routing Algorithms: Distance Vector

4.5 Routing Algorithms:Hiericial Rounting

Chapter 5 Link Layer

5.3 Multiple Access Protocols

5.19

Given: boradcast channel of rate R bps

desiderate

1. [效率, 有效性] one note, R
2. [公平] M nodes, R/M
3. [分布式, 完全分散] Fully decentralized,

    > 击鼓传花🥁， 没有一个特殊的节点协调传输，没有同步
    >
    > ❓ 非同步的 电路 有吗

    1. no special node to coordineate transmissions
    2. no synchronization of clocks, slots
4. [简单性] Simple

5.20

5.21

TDMA: time division multiple access

> 1. 只有一个节点时，没法全速传输

> 2. 公平性满足
> 3. 高度分散
> 4. 很简单

1. access to channel in "rounds"
2. each station gets fixed length slot
3. unused slots go idle
4. example: 6-station LAN, 1, 3, 4 have pkt, slots 2,5,6 idle

5.22

FDMA: frequency division multiple access

1. channel spectrum divided into frequency bands
2. each station assigned fixed frequency band
3. unused transmission time in frequency bands go idle
4. example:

5.23

> 资源划分类的协议 资源不划分: 随机访问类的协议

2. [重传] two or more transmitting nods - collision
3. [判断有无传完]
4. [碰撞避免的]

5.24

Slotted ALOHA[夏威夷打招呼，想发就发]

> 不满足完全分散 追求效率，导致分散性降低

5.25 C: 碰撞时期 E: 空闲时期 S: 成功时期

5.26 Slotted ALOHA efficiency: long-run fraction of successful slots(many nodes, all with many frames to send)

5.27 Pure (unslotted) ALOHA

> 产生冲突的概率增加了 不划分时期， 完全分散，追求分散性，导致效率降低

5.29 CSMA(carier sense multiple access)[载波侦听]

listen before transimic

5.30

collisions can still occur:

propagation delay means two nodes may not hear each other's transmission

5.31

CSMA/CD [碰撞检测 的 载波侦听]

1. [碰撞检测] collisions detected within short time
2. [colliding transmissions aborted, reducing channel wastage

1.1

> 有线: 检测信号强度

1.2

> 无线: 在无线中信号衰减，一边发很难一边检测

2.1 human analogy: the polite conversationalist

>

5.32

> 充分让其他节点知道数据frame产生故障 collision detect/abort time

5.33

Ethernet CSMA/CD algorithm

5. [M次碰撞后，怎么去选择碰撞的时间] after aborting, NIC enters binary (exponential) backoff:

> 0-5152 去选择等待时间， 回退到第2步， 碰撞时间多，可选时间多了，选择时间短的先发， 长的后发。目的是避免冲突。

- after mth collision, NIC chooses K at random from { 0, 1, 2, ..., $2^m - 1$}. NIC waits K 512 bit times, returns to Step 2.
-

5.34

CSMA/CD efficiency

$$efficiency = \frac{1}{1 + \frac{5t_{prop}}{t_{trans}}} \tag{14}$$

> 一个节点做了载波侦听后传输一个无限大的比特，t_{trans} 无限大，效率== 1

5.35

random access MAC protocols

- high load: collision overhead

> 碰撞负担就很大

5.36

polling [轮循协议]

> master 轮流询问，发 poll 报文 高负债，低负载效率都很好

1. master node “invites” slave nodes to transmit in turn
2.

5.37

token

> 短的报文， 令牌环

5.38

> 不同频段传输，划分时期

5.40

> channel partitioning

5.4 LAN

⭐ [Discussion Problem]5.7 a day in the life of a web request

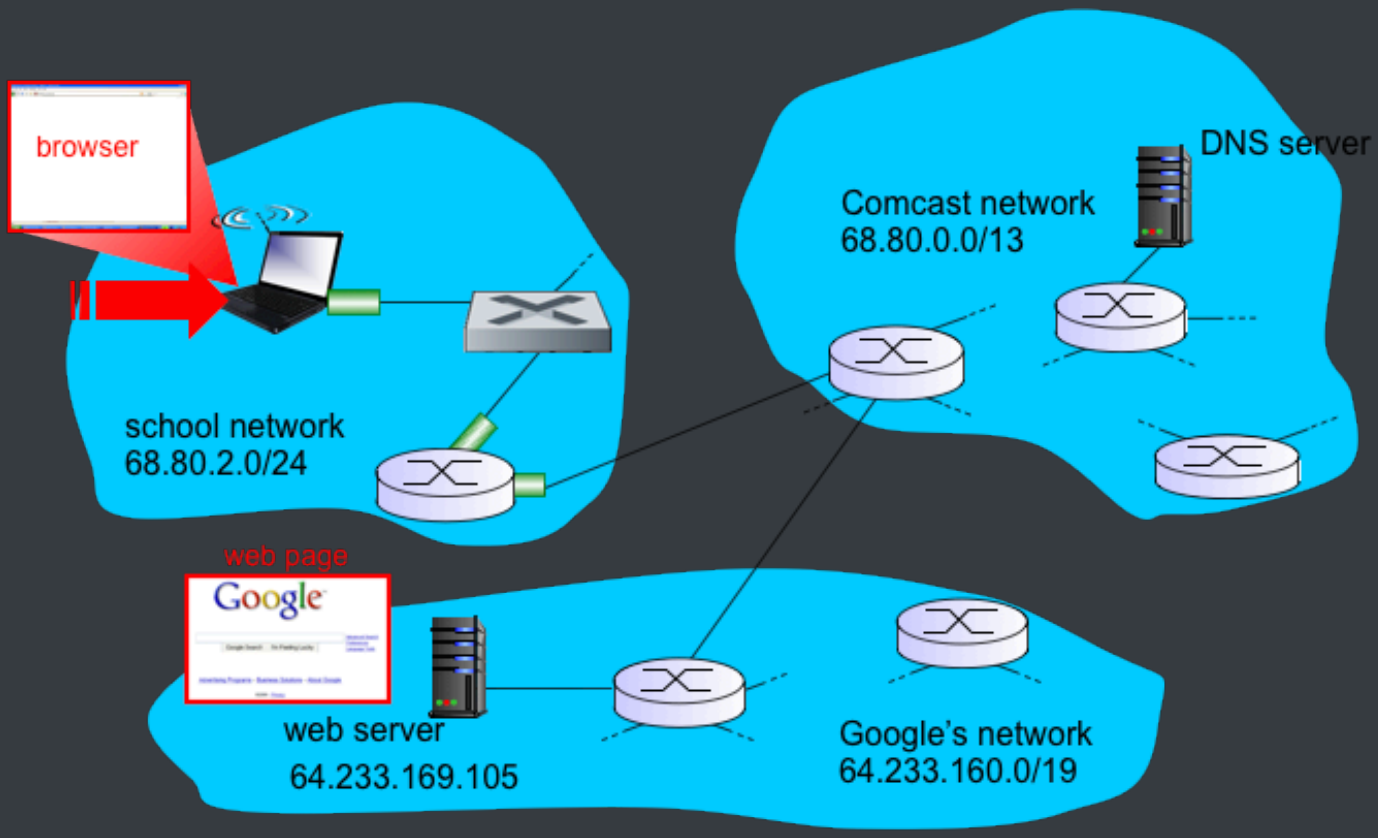
5.88

Synthesis: a day in the life of a web request

- journey down protocol stack complete!
- - - application, transport, network, link
- putting-it-all-together: synthesis!
- - - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
    - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

5.89

A day in the life: scenario



CE265 Chapter 6 Wireless Mobility

> 我现在的要求呢
>
> 就是很简单
>
> 电子笔记每个都要有批注，方便进行理解和校准，
>
> 同时先过一遍中文书，再过一遍英文书，通读，skim，不求甚解
>
> skimming 的同时补充以md中的笔记，用来应对ielts的备考
>
> 同时用以证明自己的理解

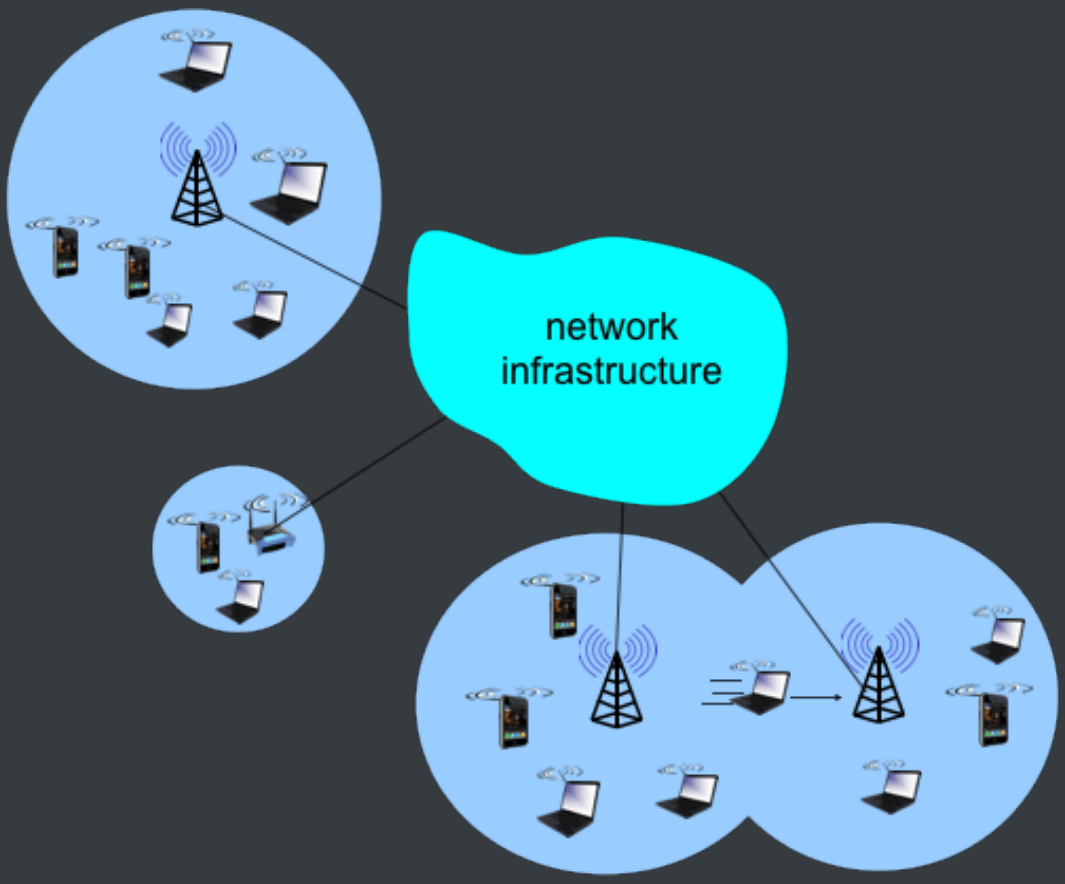
6.2

Background:

- # wireless (mobile) phone subscribers now exceeds # wired phone subscribers (5-to-1)!
- # wireless Internet-connected devices equals # wireline Internet-connected devices
- - laptops, Internet-enabled phones promise anytime untethered Internet access
- two important (but different) challenges
- - *wireless:* communication over wireless link
  - *mobility:* handling the mobile user who changes point of attachment to network

6.1 Introduction

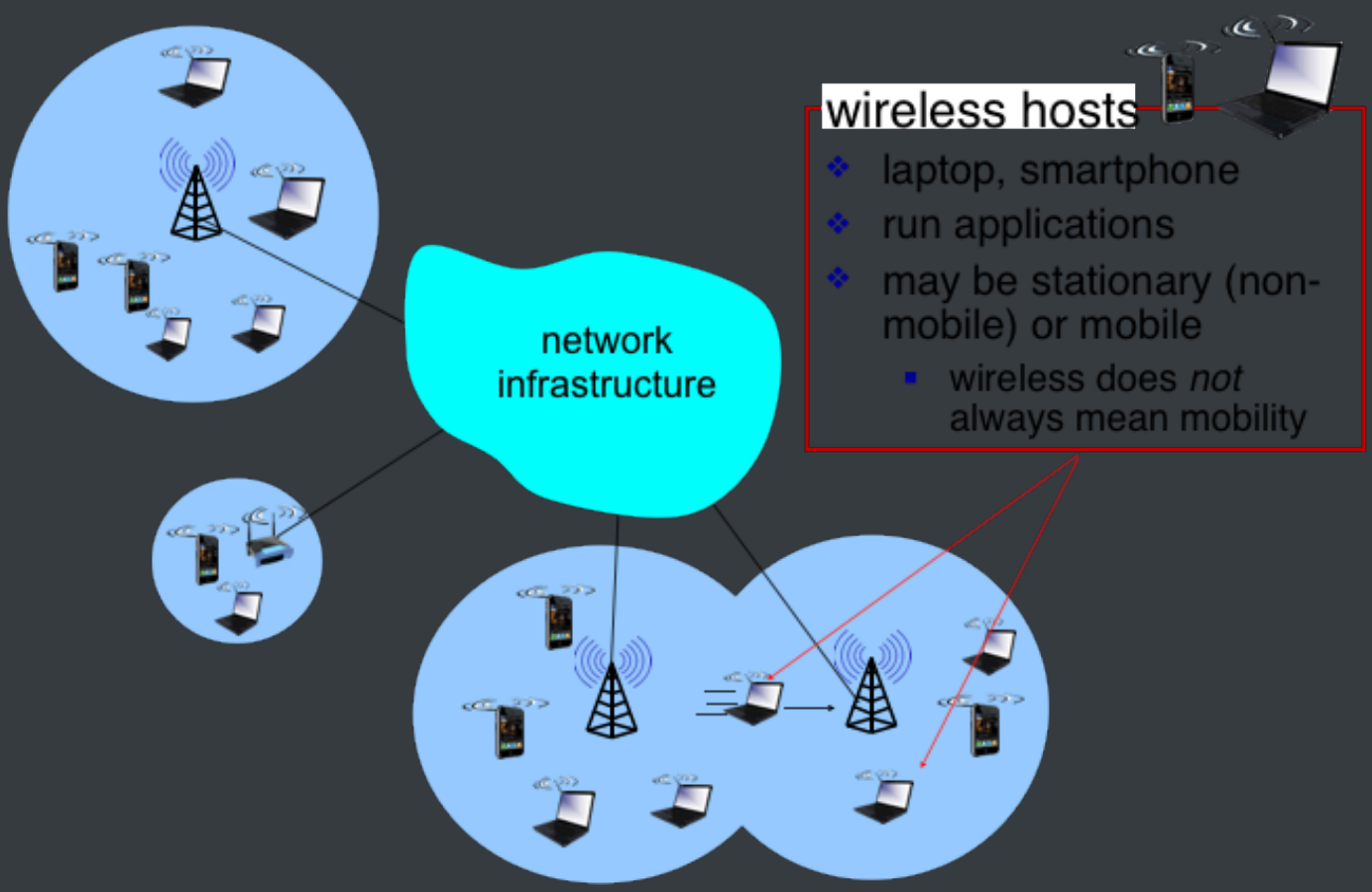
6.4 Elements of a wireless network



6.5 wireless hosts

wireless hosts

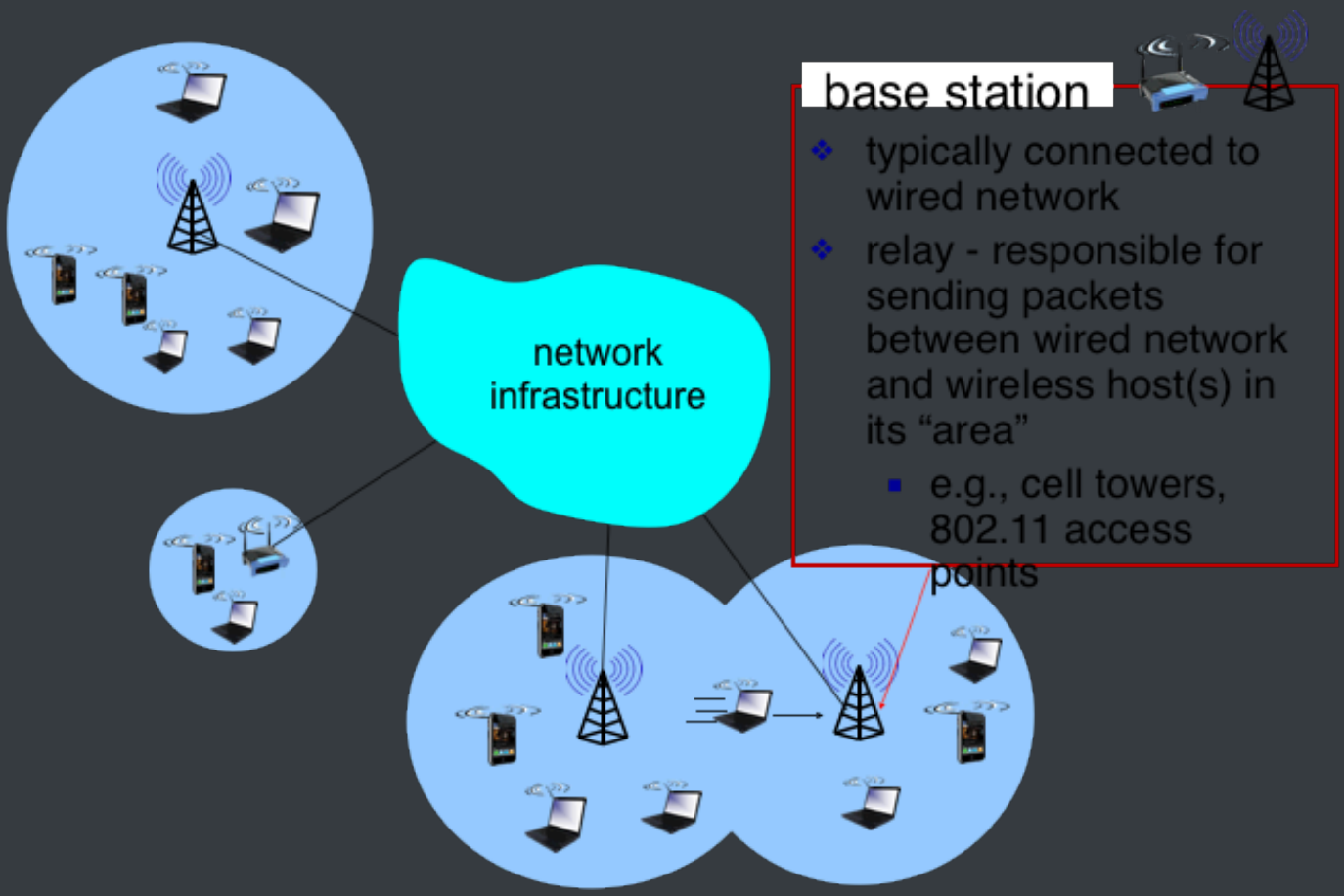
- laptop, smartphone
- run applications
- may be stationary (non-mobile) or mobile
- - wireless does *not* always mean mobility



6.6 base station

base station

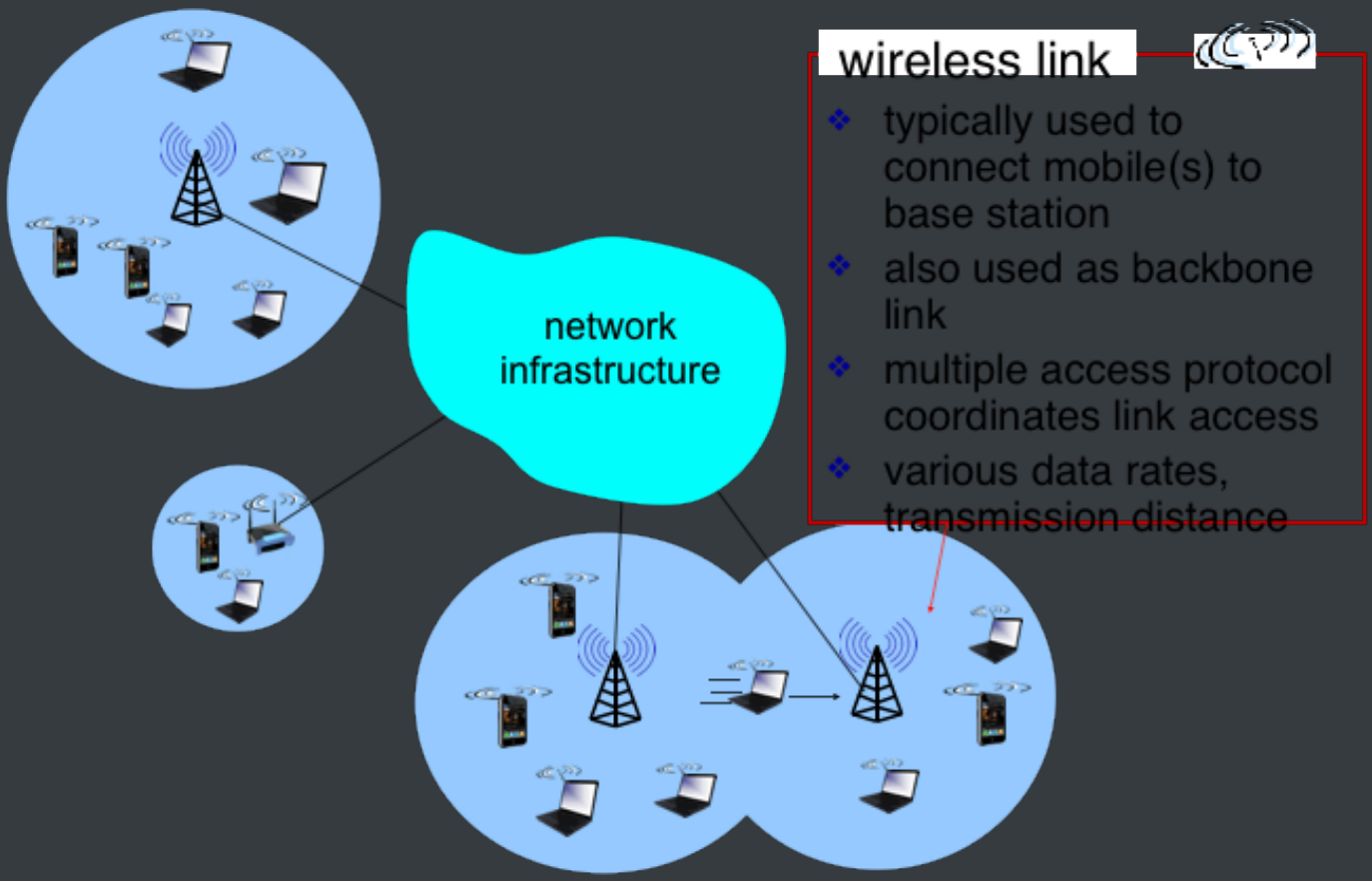
- typically connected to wired network
- relay - responsible for sending packets between wired network and wireless host(s) in its “area”
- - e.g., cell towers, 802.11 access points



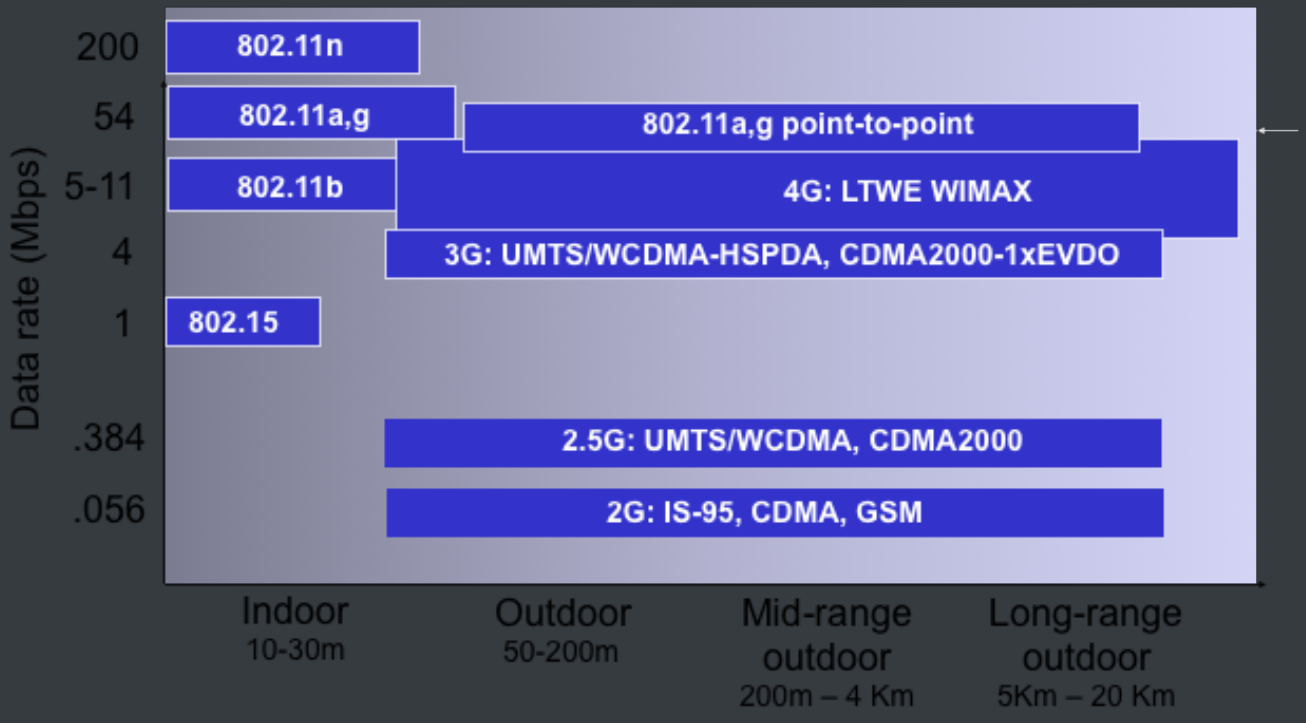
6.7 wireless link

wireless link

- typically used to connect mobile(s) to base station
- also used as backbone link
- multiple access protocol coordinates link access
- various data rates, transmission distance



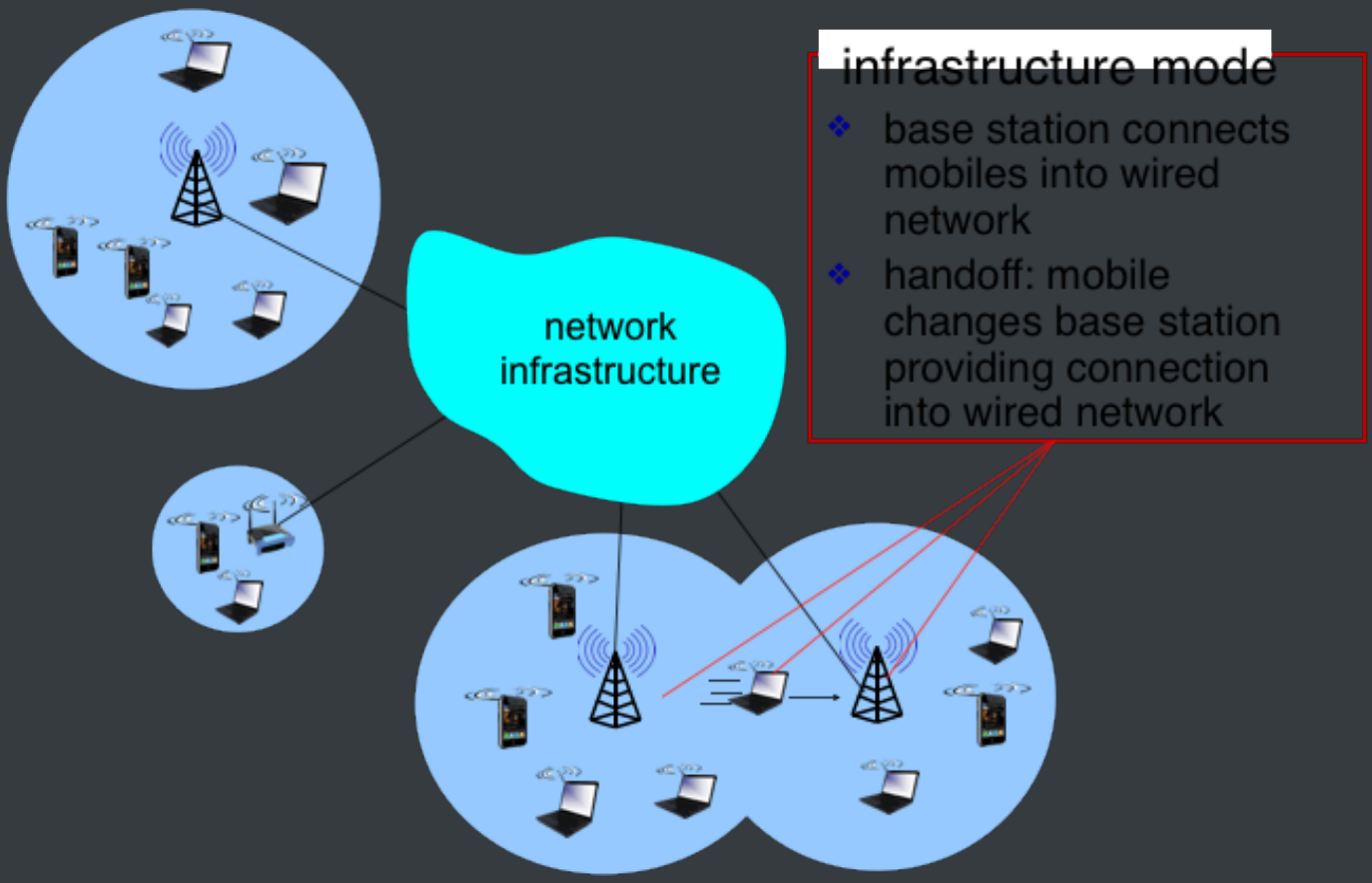
6.8 Characteristics of selected wireless links



6.9 infrastructure mode

infrastructure mode

- base station connects mobiles into wired network
- handoff: mobile changes base station providing connection into wired network



6.10 ad hoc mode

ad hoc mode(点对点)

> ad hoc(for this, for this purpose only),
>
> 为某种目的设置的
>
> 省去ap base stations

- no base stations
- nodes can only transmit to other nodes within link coverage
- nodes organize themselves into a network: route among themselves

6.11 Wireless network taxonomy[分类]

	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, WiMAX, cellular) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: *mesh net*
no infrastructure	no base station, no connection to larger Internet (Bluetooth, ad hoc nets)	no base station, no connection to larger Internet. May have to relay to reach other a given wireless node MANET, VANET

6.2 Wireless links, characteristics, CDMA

6.13 Wireless Link Characteristics (1)

> 衰减的信号强度
>
> 来自其他信号源的干扰
>
> 多路传播📣

important differences from wired link ….

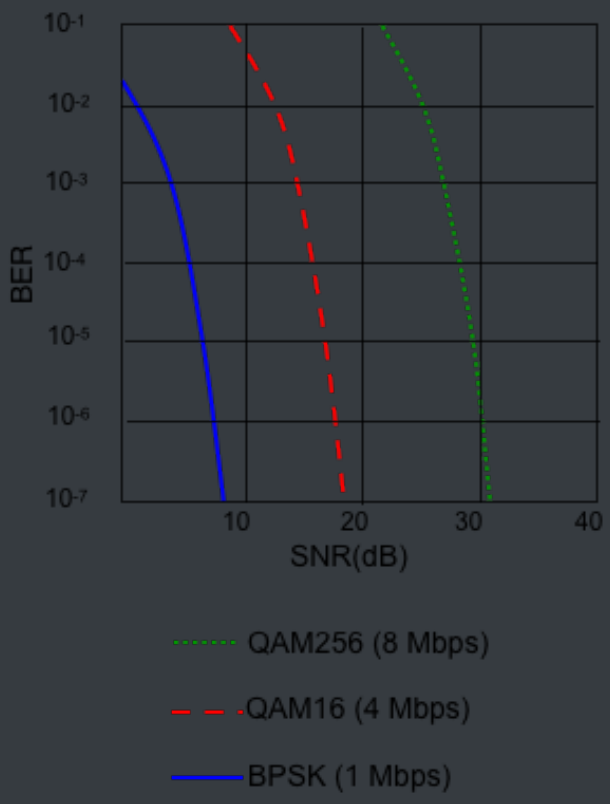
- - *decreased signal strength:* radio signal attenuates as it propagates through matter (path loss)
  - *interference from other sources:* standardized wireless network frequencies (e.g., 2.4 GHz) shared by other devices (e.g., phone); devices (motors) interfere as well
  - *multipath propagation:* radio signal reflects off objects ground, arriving ad destination at slightly different times

…. make communication across (even a point to point) wireless link much more “difficult”

6.14 Wireless Link Characteristics (2)

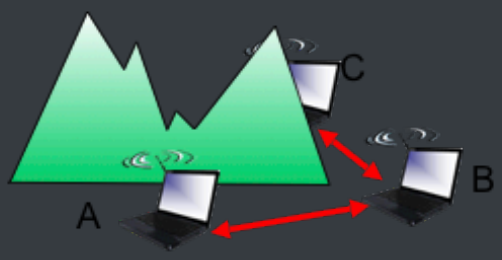
> SNR (Signal to Noise)

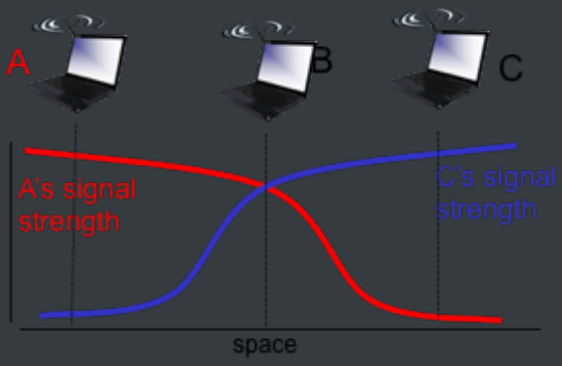
- SNR: signal-to-noise ratio
- - larger SNR – easier to extract signal from noise (a “good thing”)
- *SNR versus BER tradeoffs*
- - *given physical layer:* increase power -> increase SNR->decrease BER
  - *given SNR:* choose physical layer that meets BER requirement, giving highest thruput
  - - SNR may change with mobility: dynamically adapt physical layer (modulation technique, rate)



6.15 Wireless network characteristics

Multiple wireless senders and receivers create additional problems (beyond multiple access):





Hidden terminal problem

- B, A hear each other
- B, C hear each other
- A, C can not hear each other means A, C unaware of their interference at B

Signal attenuation:

- B, A hear each other
- B, C hear each other
- A, C can not hear each other interfering at B

Hidden terminal problem

- B, A hear each other
- B, C hear each other
- A, C can not hear each other means A, C unaware of their interference at B

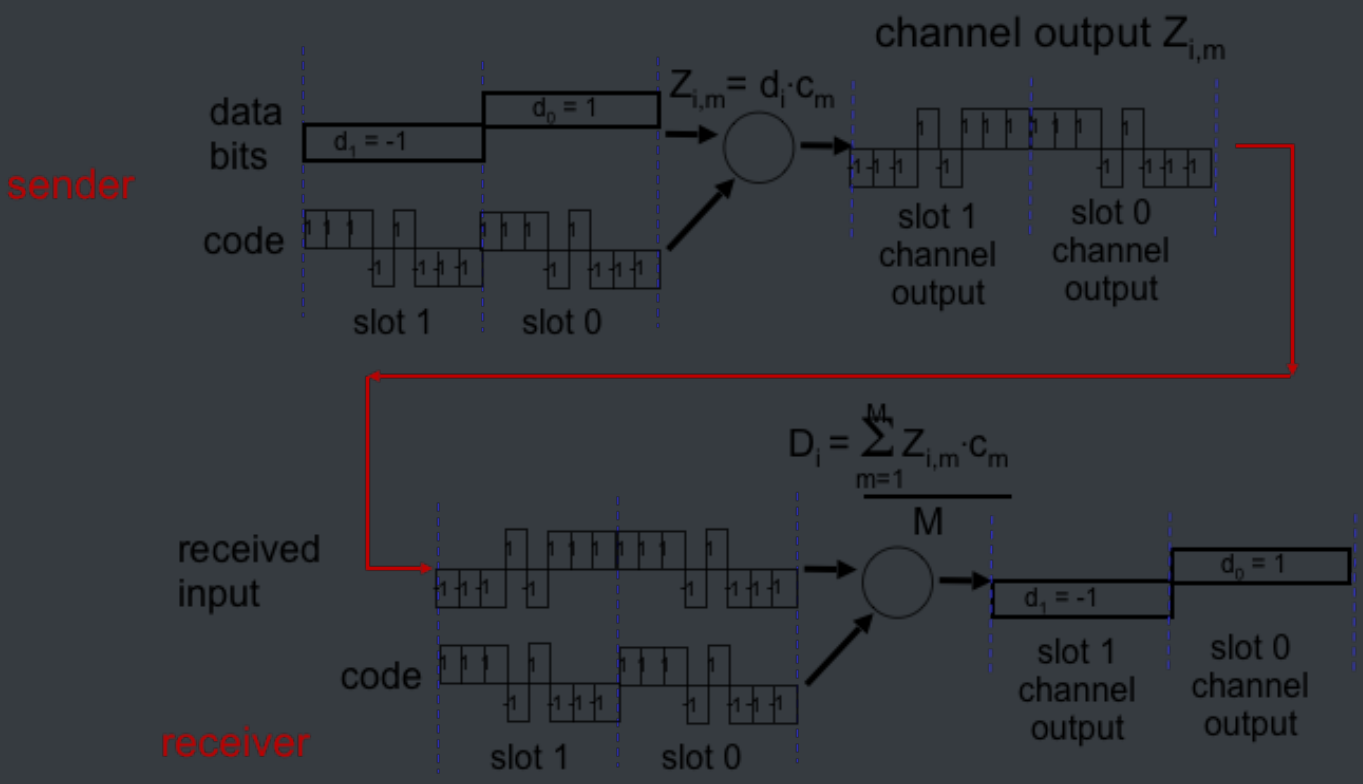
Signal attenuation:

- B, A hear each other
- B, C hear each other
- A, C can not hear each other interfering at B

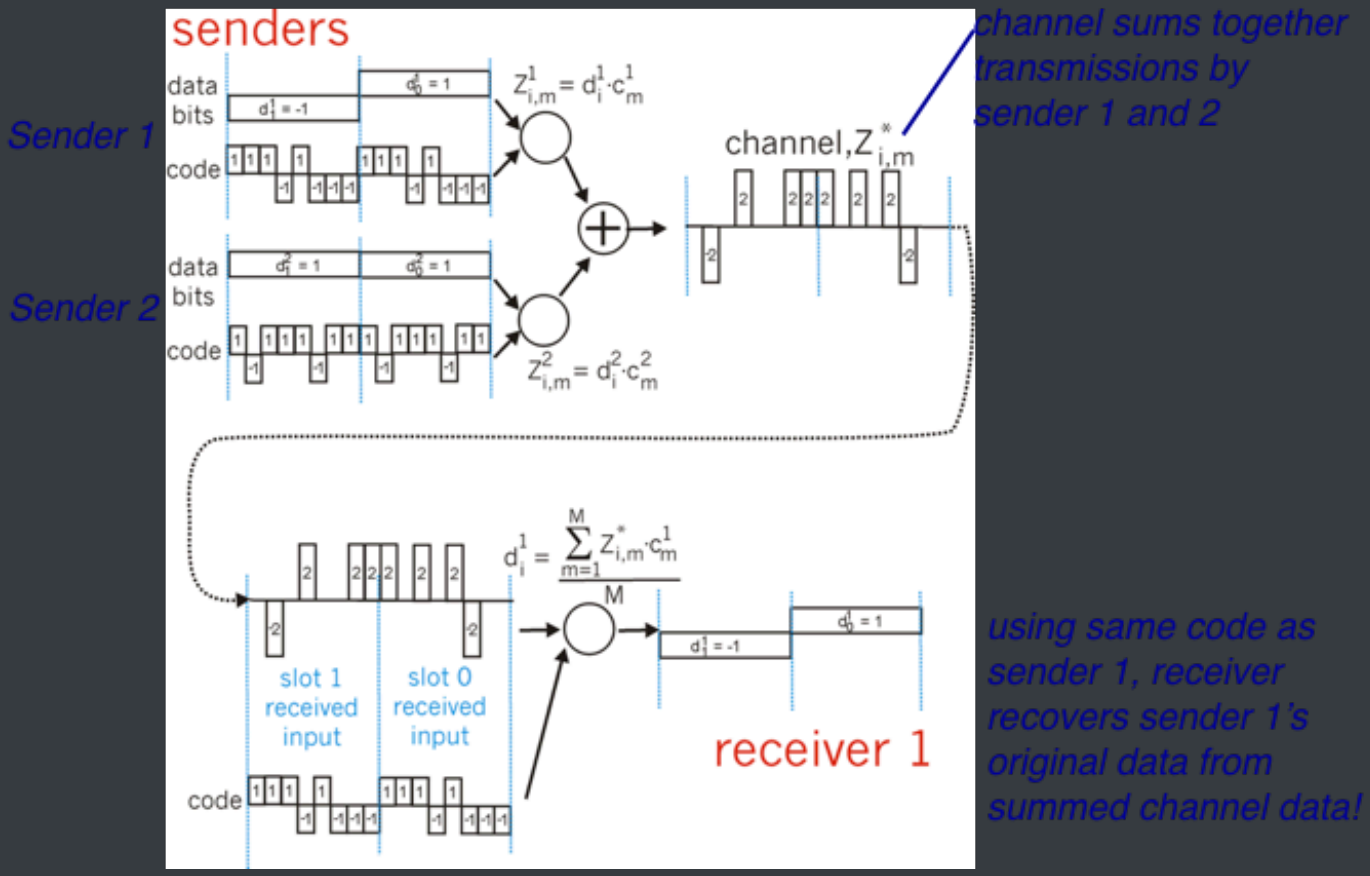
6.16 Code Division Multiple Access (CDMA)

- unique “code” assigned to each user; i.e., code set partitioning
-
  - all users share same frequency, but each user has own “chipping” sequence (i.e., code) to encode data
  - allows multiple users to “coexist” and transmit simultaneously with minimal interference (if codes are “orthogonal”)
- *encoded signal* = (original data) X (chipping sequence)
- *decoding:* inner-product of encoded signal and chipping sequence

6.17 CDMA encode/decode



6.18 CDMA: two-sender interference



6.3 IEEE 802.11 wireless LANs (“Wi-Fi”)

6.20 IEEE 802.11 Wireless LAN

> 区分了 IEEE 802.11 小字母后缀的一些Wi-Fi频段，记忆为主
>
> 802.11b 2.4-5GHz
>
> > DSSS (direct sequence spread spectrum)

802.11b

- ❖2.4-5 GHz unlicensed spectrum
- ❖up to 11 Mbps
- ❖direct sequence spread spectrum (DSSS) in physical layer
- - all hosts use same chipping code

802.11a

- - 5-6 GHz range
  - up to 54 Mbps

802.11g

- - 2.4-5 GHz range
  - up to 54 Mbps

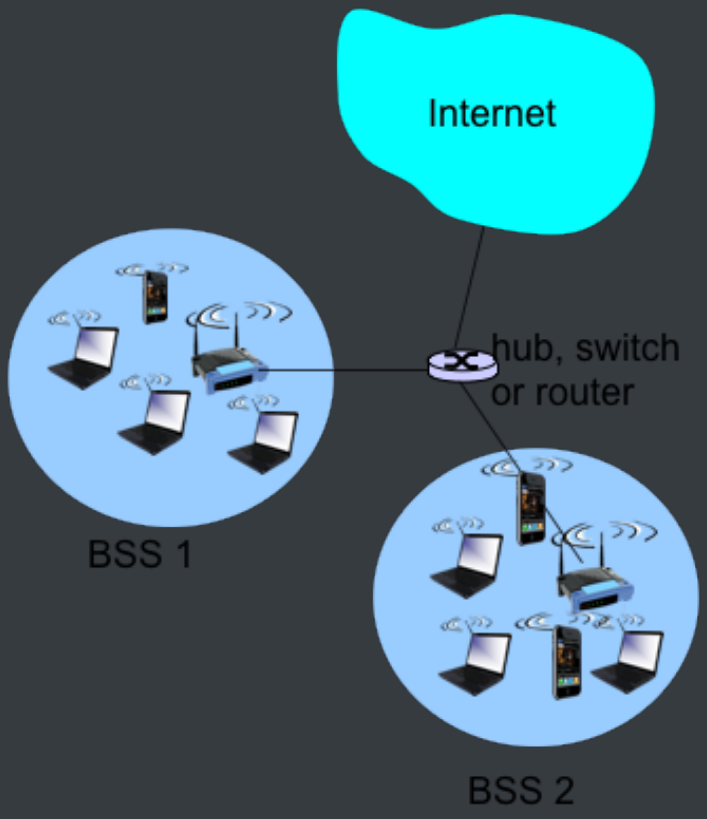
802.11n: multiple antennae

- - 2.4-5 GHz range
  - up to 200 Mbps
- all use CSMA/CA for multiple access
- all have base-station and ad-hoc network versions

6.21 802.11 LAN architecture

- wireless host communicates with base station
- - base station = access point (AP)
- Basic Service Set (BSS) (aka “cell”) in infrastructure mode contains:
- - wireless hosts
  - access point (AP): base station

- ad hoc mode: hosts only



6.22 802.11: Channels, association

- 802.11b: 2.4GHz-2.485GHz spectrum divided into 11 channels at different frequencies
-
  - AP admin chooses frequency for AP
  - interference possible: channel can be same as that chosen by neighboring AP!
- host: must *associate* with an AP
-
  - scans channels, listening for *beacon frames* containing AP's name (SSID) and MAC address
  - selects AP to associate with
  - may perform authentication [Chapter 8]
  - will typically run DHCP to get IP address in AP's subnet

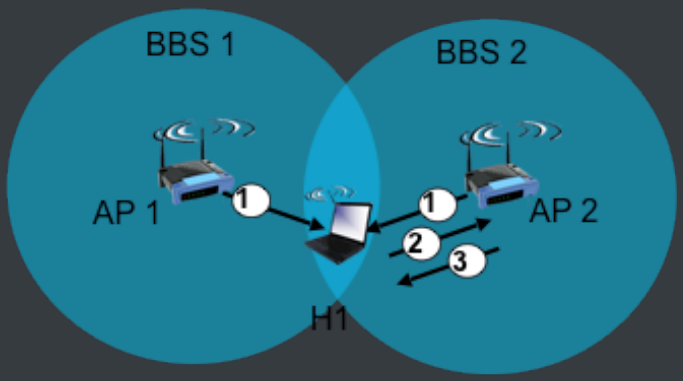
6.23 802.11: passive/active scanning

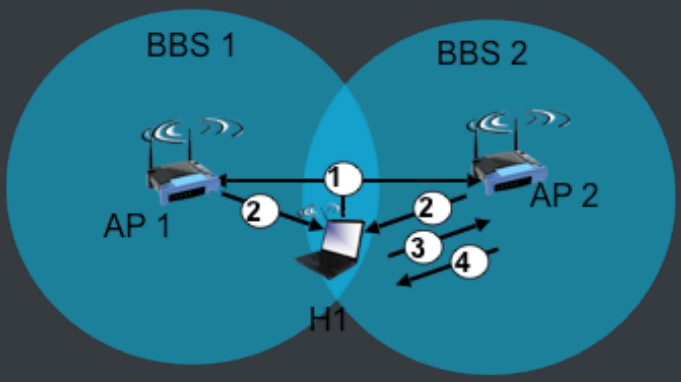
passive scanning:

1. beacon frames sent from APs
2. association Request frame sent: H1 to selected AP
3. association Response frame sent from selected AP to H1

active scanning:

1. Probe Request frame broadcast from H1
2. Probe Response frames sent from APs
3. Association Request frame sent: H1 to selected AP
4. Association Response frame sent from selected AP to H1





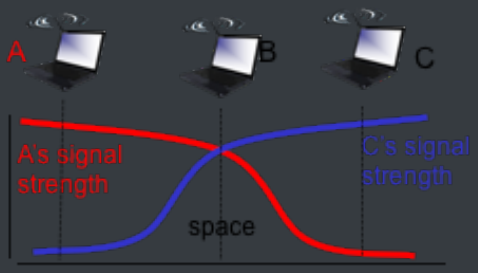
passive scanning:
(1) beacon frames sent from APs
(2) association Request frame sent: H1 to selected AP
(3) association Response frame sent from selected AP to H1

active scanning:
(1) Probe Request frame broadcast from H1
(2) Probe Response frames sent from APs
(3) Association Request frame sent: H1 to selected AP
(4) Association Response frame sent from selected AP to H1

6.24 IEEE 802.11: multiple access

- avoid collisions: 2+ nodes transmitting at same time
- 802.11: CSMA - sense before transmitting

- - don’t collide with ongoing transmission by other node
- 802.11: *no* collision detection!
- - difficult to receive (sense collisions) when transmitting due to weak received signals (fading)
  - can’t sense all collisions in any case: hidden terminal, fading
  - goal: *avoid collisions**:* CSMA/C(ollision)A(voidance)



6.25

IEEE 802.11 MAC Protocol: CSMA/CA

> CSMA载波侦听
>
> 不做碰撞的检测

802.11 sender

1 if sense channel idle for DIFD then transmit entire frame

2 if senes channel busy then

2.1 start random backoff time

2.2 timer counts down while channel idle

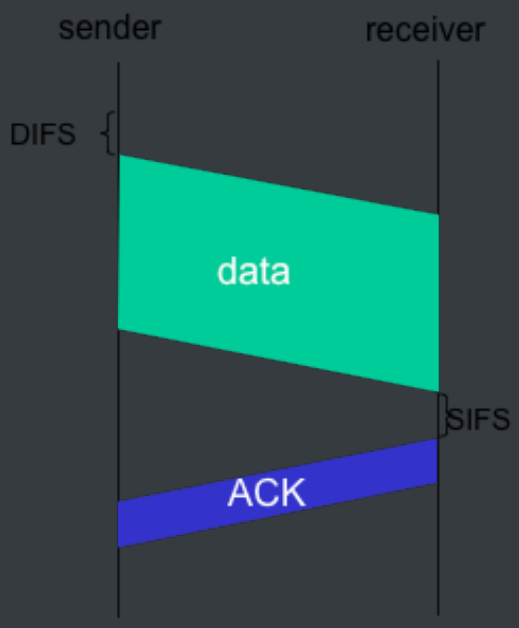
2.3 transmit when timer expires

802.11 receiver

if frame received OK

return ACK after SIFS (ACK needed due to hidden terminal problem)

> 一次性传整个数据frame
>
> SIFS， 状态切换， 从听转为发
>
> 基础模式，有可能碰撞💥



6.26

> 预约模式

> 利用预约的节点传输数据frame
>
> 避免长数据碰撞💥
>
> RTS (Request to send) 这些请求frame是很短的，开销是可以忍的
>
> CTS(Clear to send) 是 基站发出的， 利用广播的信道发送的
>
> CTS其他节点会被告知，延迟发送RTS

Avoiding collisions (more)

idea: allow sender to “reserve” channel rather than random access of data frames: avoid collisions of long data frames

- sender first transmits *small* request-to-send (RTS) packets to BS using CSMA
- - RTSs may still collide with each other (but they’re short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
- - sender transmits data frame
  - other stations defer transmissions

avoid data frame collisions completely using small reservation packets!

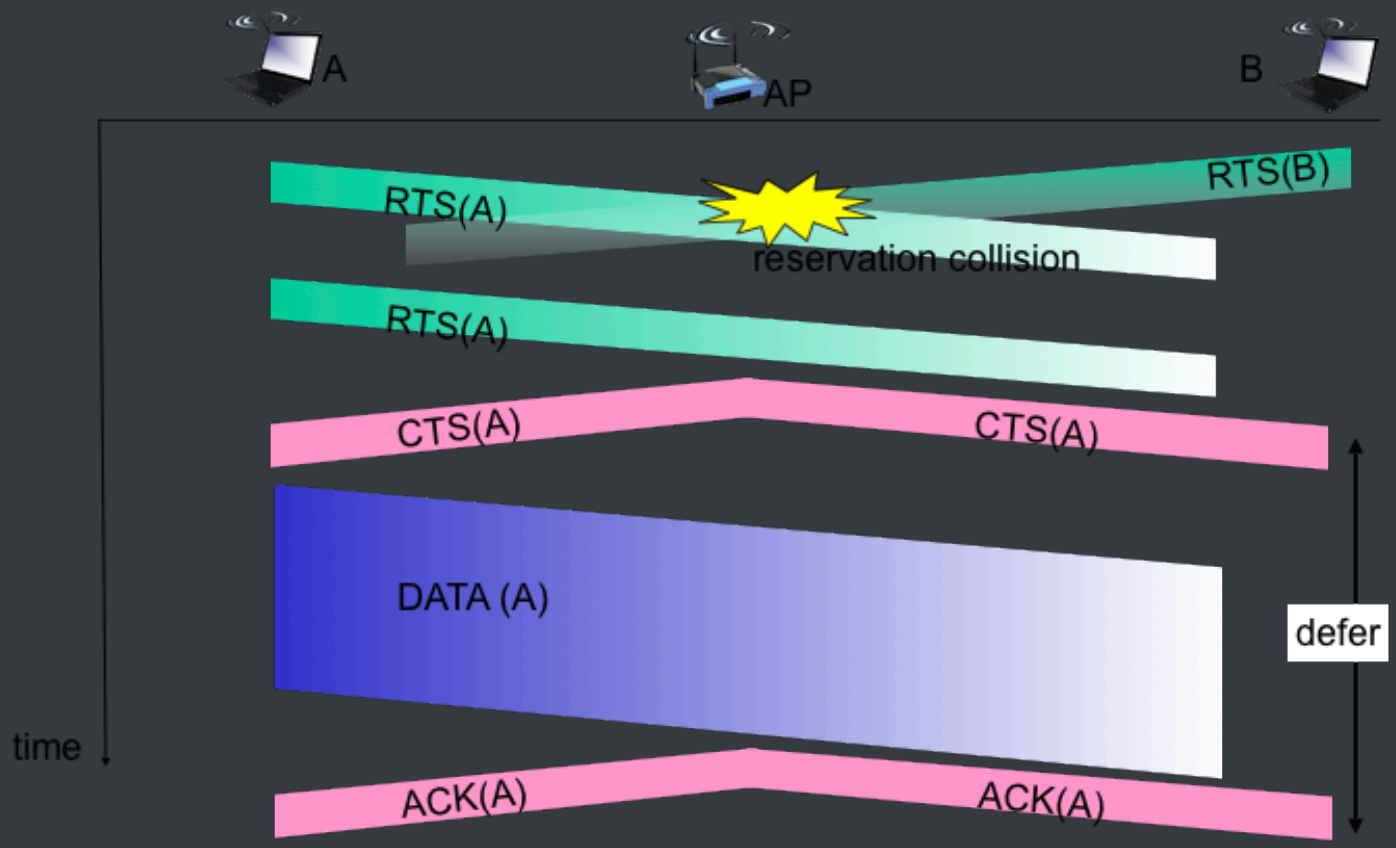
6.27

Collision Avoidance: RTS-CTS exchange

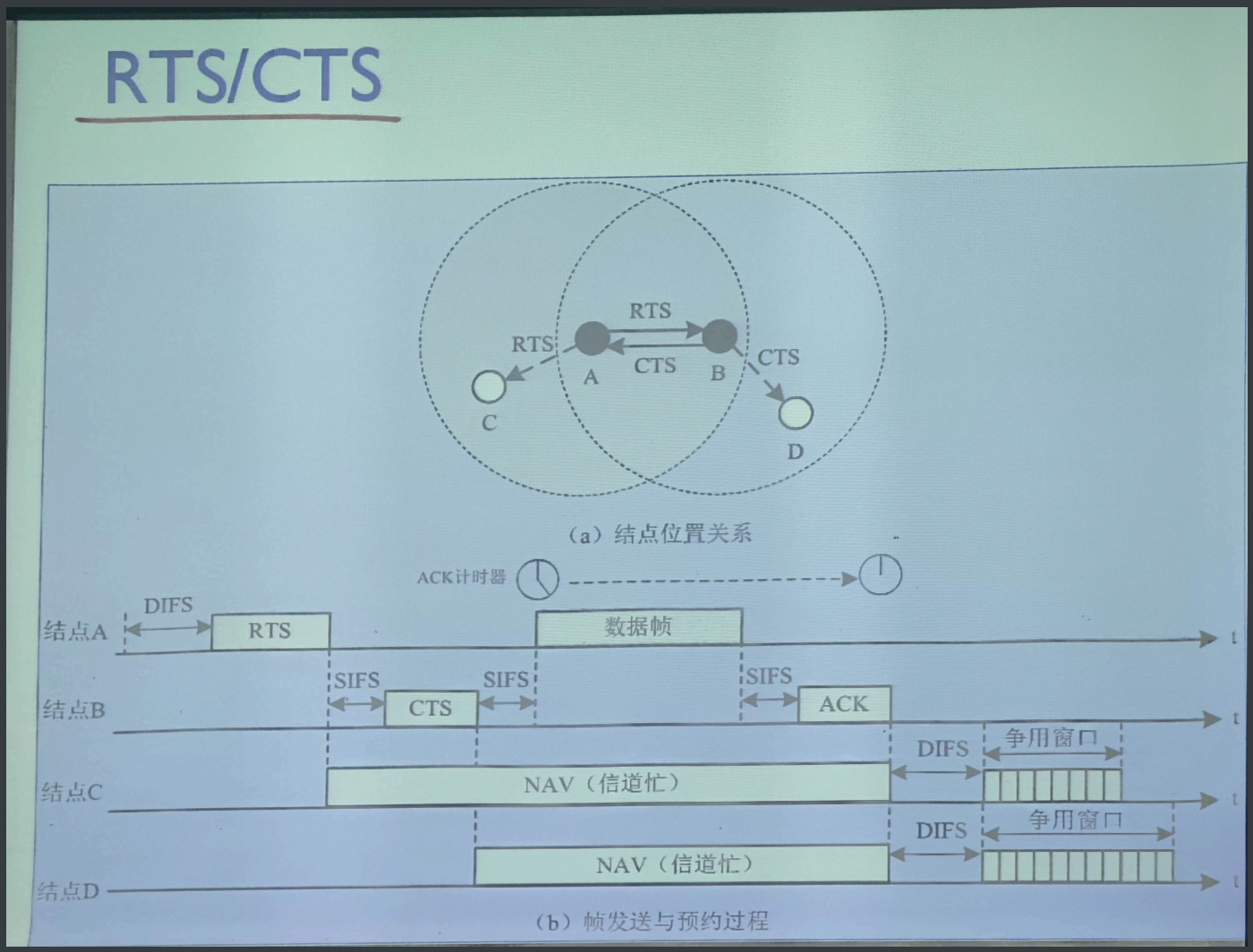
> RTS 发送
>
> B， CTS 响应
>
> 收到数据后，B，发ACK

❓ 为什么是折的，不是只有B发CTS(clear to send), ACK 吗

大，长的数据DATA, 发一个小的RTS预约模式还是划算的



四个节点

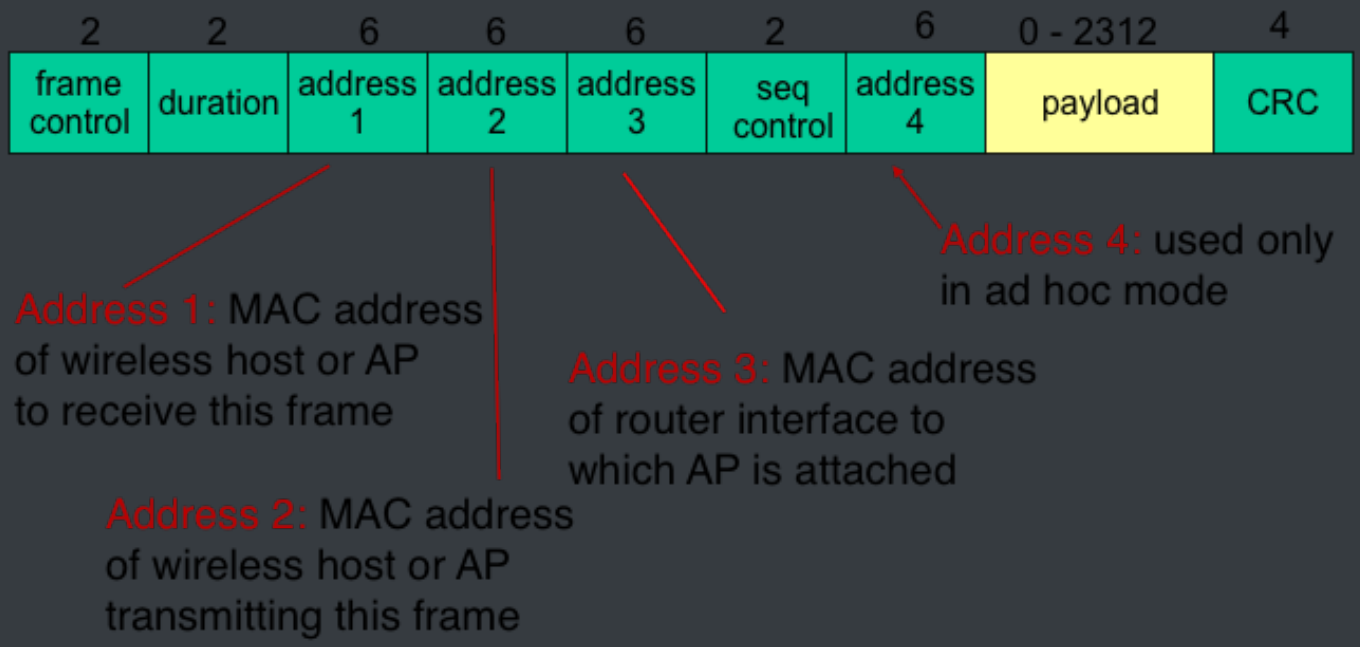


6.28

802.11 frame: addressing

> 少不了的是 payload 和 CRC
>
> payload: 包含上层协议，通常长度是1500字节，
>
> frame control 类型，子类型，RTS, CTS, DATA, ACK

> 地址1 目的地址
>
> 地址 2 源地址，
>
> 地址3 路由器MAC地址，跨越不同子网
>
> 地址4 ad hoc 地址

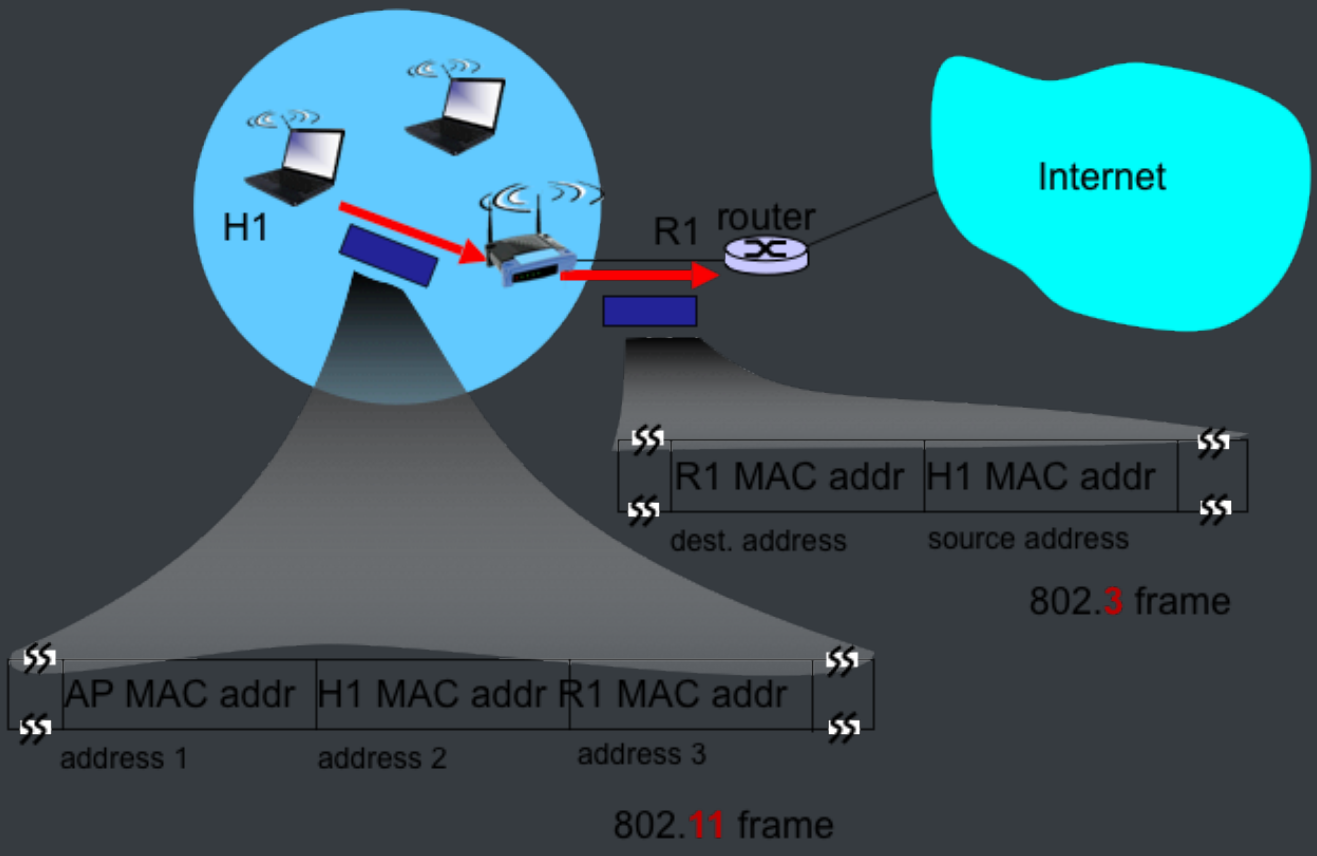


Address 1: MAC(media access control)[媒体存取控制] address of wireless host or AP to receive this frame

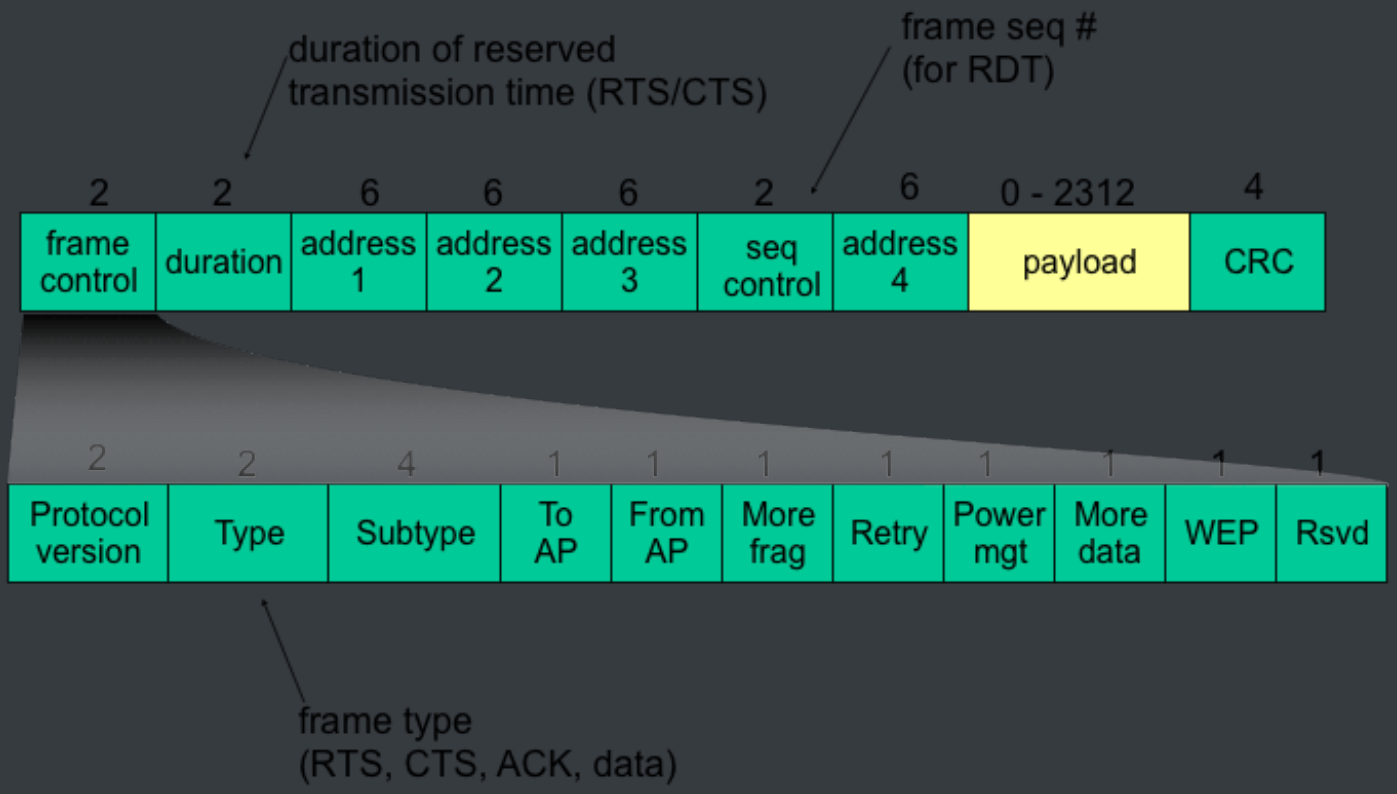
>

Address 2: MAC(media access control)[媒体存取控制] address of wireless host or AP to transmitting this frame

6.29 802.11 frame: addressing



6.30 802.11 frame: more



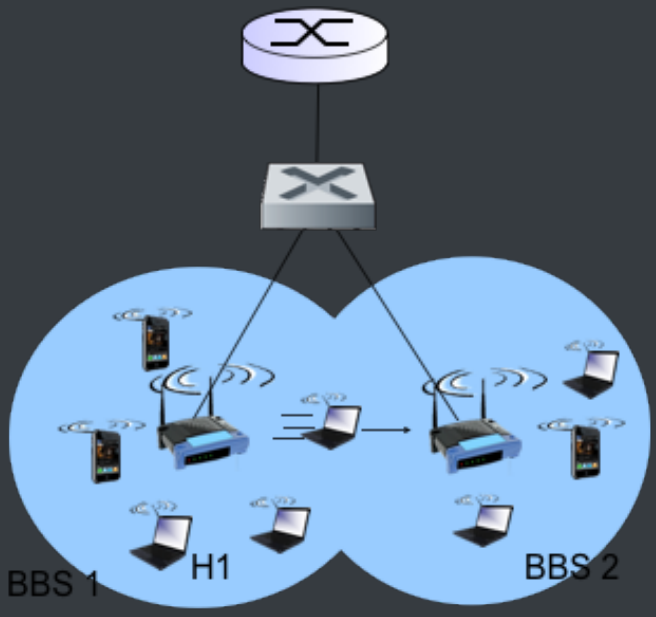
6.31 802.11: mobility within same subnet

> 只有一台交换机连接，位于一个子网
>
> 移动时位于一个子网，IP不变，可持续发送数据
>
> 主机探测到AP2信号📶强，产生关联

802.11: mobility within same subnet

- H1 remains in same IP subnet: IP address can remain same

- switch: which AP is associated with H1?
- - self-learning (Ch. 5): switch will see frame from H1 and “remember” which switch port can be used to reach H1



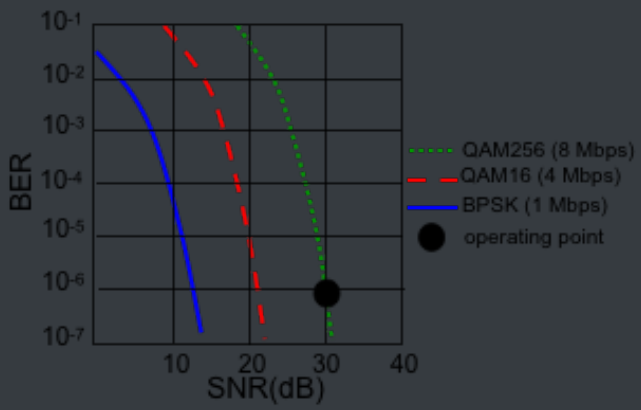
6.32 802.11: advanced capabilities

> 速率自适应
>
> 拥塞控制

802.11: advanced capabilities

Rate adaptation

- base station, mobile dynamically change transmission rate (physical layer modulation technique) as mobile moves, SNR(信道比) varies



\1. SNR decreases, BER increase as node moves away from base station

\2. When BER becomes too high, switch to lower transmission rate but with lower BER

6.33 802.11: advanced capabilities

power management

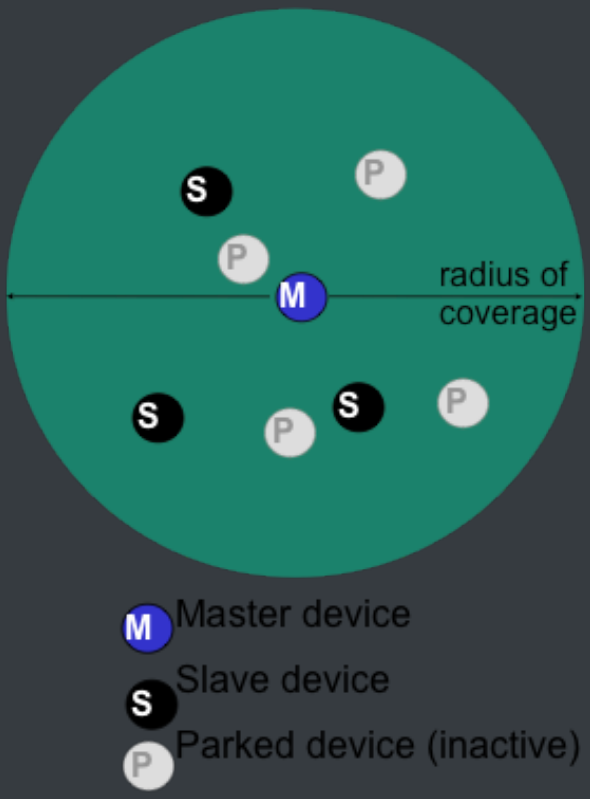
> AP 清楚哪些是休眠状态，就不传输数据了
>
> 250 微秒， 节点节省自己的能源
>
> 没有frame要发送和接受的99%处于睡眠zzz状态, 非常节省能源

- node-to-AP(Access Point)[接入点]: “I am going to sleep until next beacon[信号灯] frame”
    - AP knows not to transmit frames to this node
    - node wakes up before next beacon frame
- beacon frame: contains list of mobiles with AP-to-mobile frames waiting to be sent
    - node will stay awake if AP-to-mobile frames to be sent; otherwise sleep again until next beacon frame

6.34 802.15: personal area network

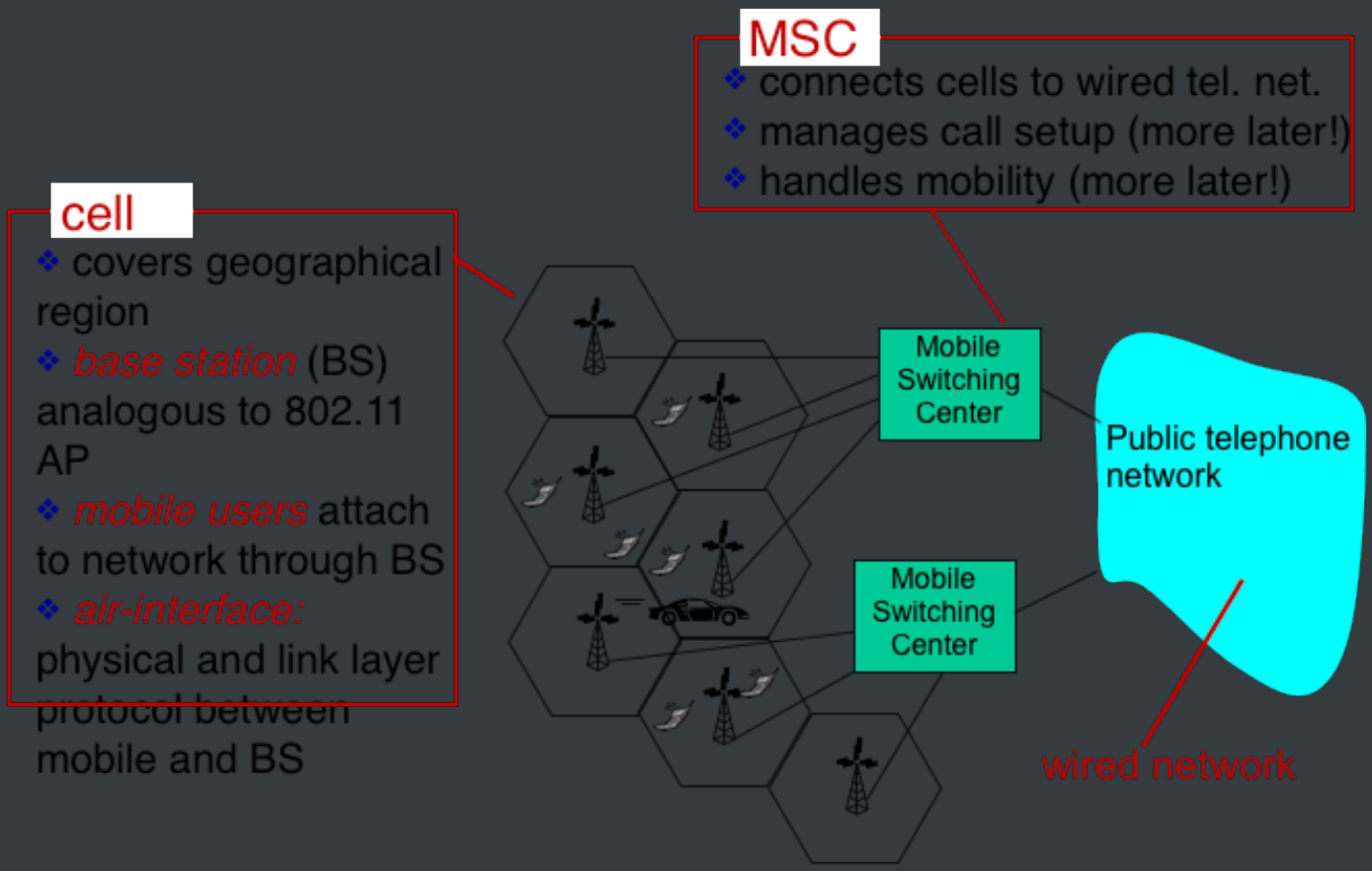
> 小范围用于 802.15

- less than 10 m diameter
- replacement for cables (mouse, keyboard, headphones)
- ad hoc: no infrastructure
- master/slaves:
-
  - slaves request permission to send (to master)
  - master grants requests
- 802.15: evolved from Bluetooth specification
-
  - 2.4-2.5 GHz radio band
  - up to 721 kbps



6.4 Cellular Internet access

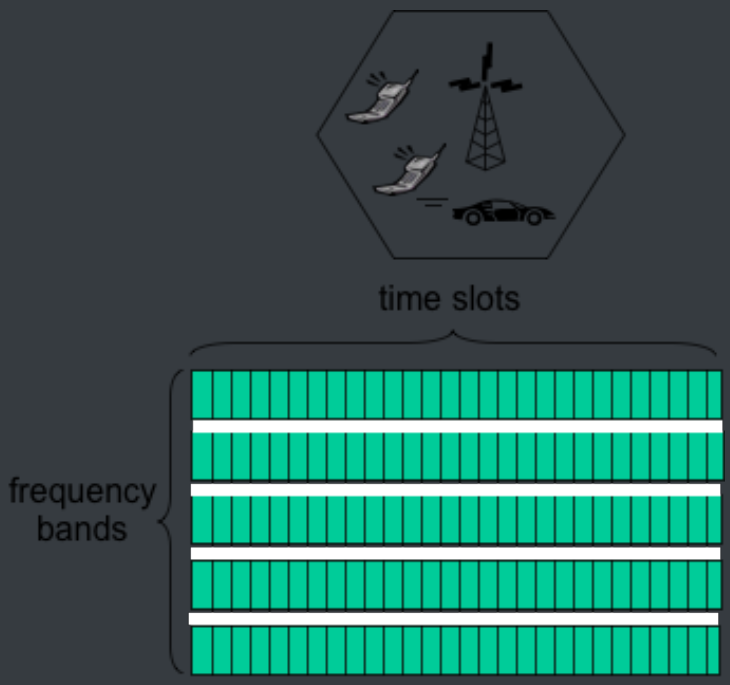
6.36 Components of cellular network architecture



6.37 Cellular networks: the first hop

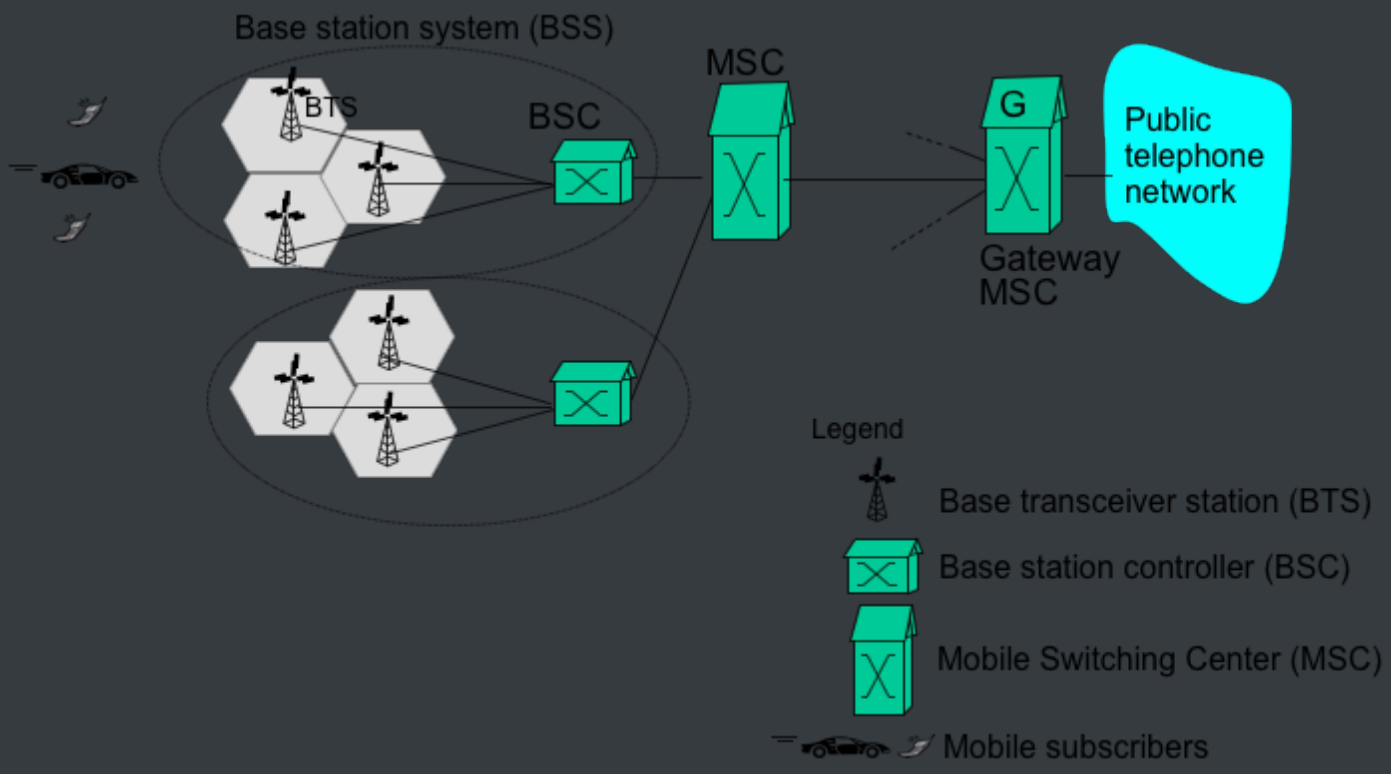
Two techniques for sharing mobile-to-BS radio spectrum

- combined FDMA/TDMA: divide spectrum in frequency channels, divide each channel into time slots
- CDMA: code division multiple access



6.38 2G (voice) network architecture

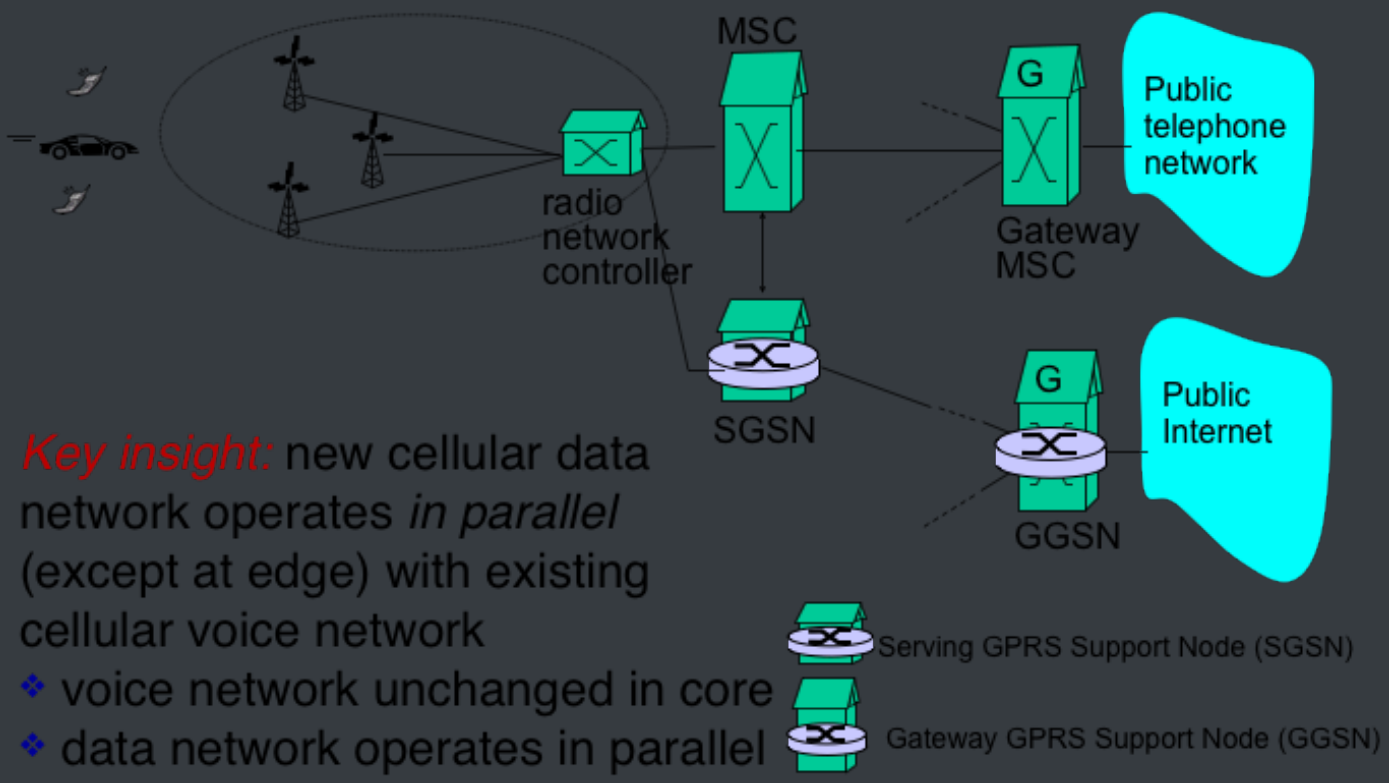
> BTS(Base Transceiver station)
>
> BSC(Base Station Controler)
>
> MSC(Mobile Switching Center)



6.39 3G (voice+data) network architecture

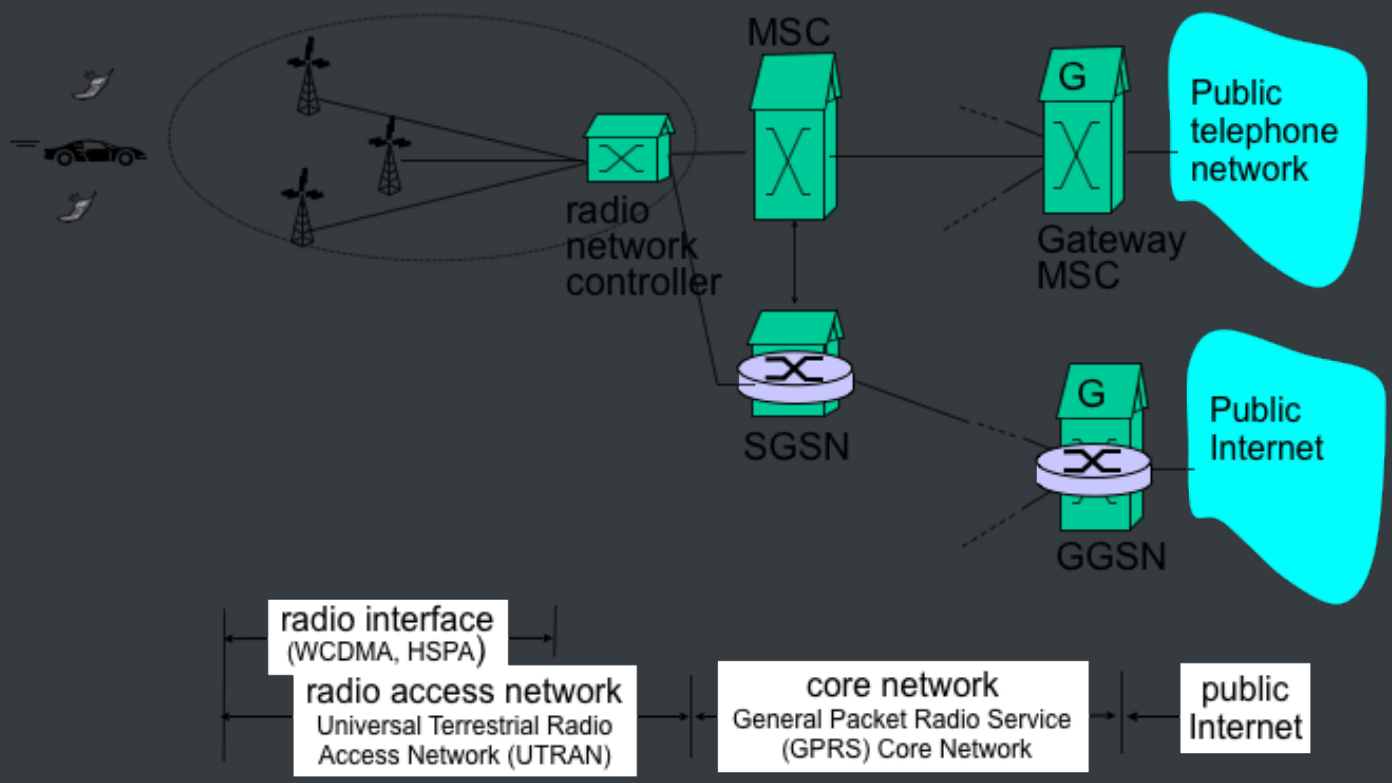
Key insight: new cellular data network operates *in parallel* (except at edge) with existing cellular voice network

- ❖ voice network unchanged in core
- ❖ data network operates in parallel



6.40 3G (voice+data) network architecture

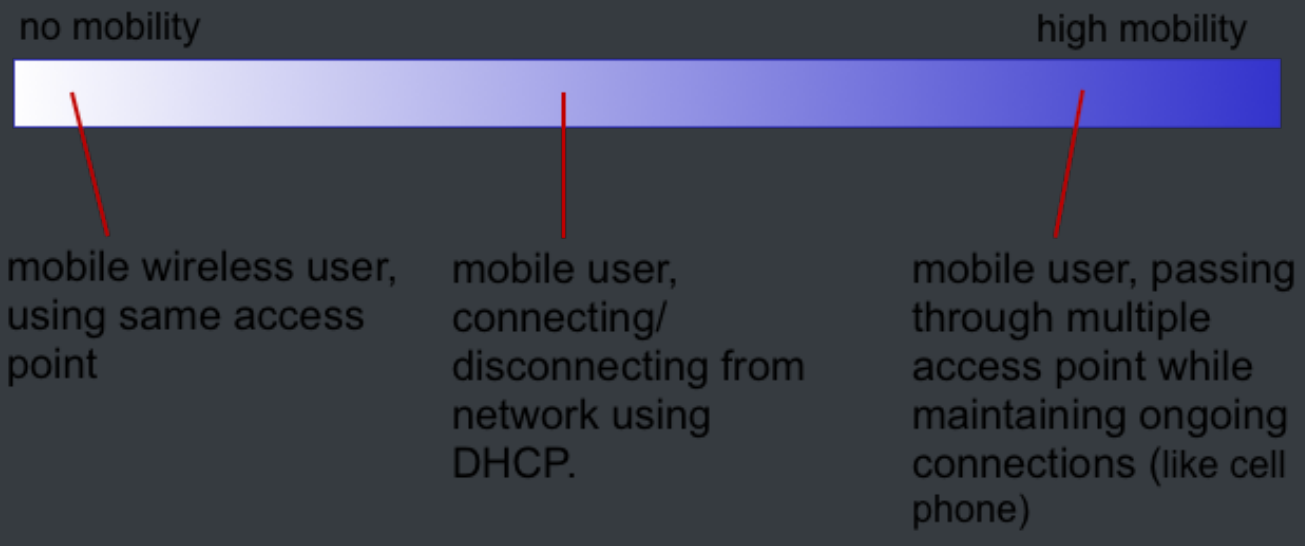
> 射频
>
> 核心网
>
> 公众网络



6.5 [Mobility] Principles: addressing and routing to mobile users

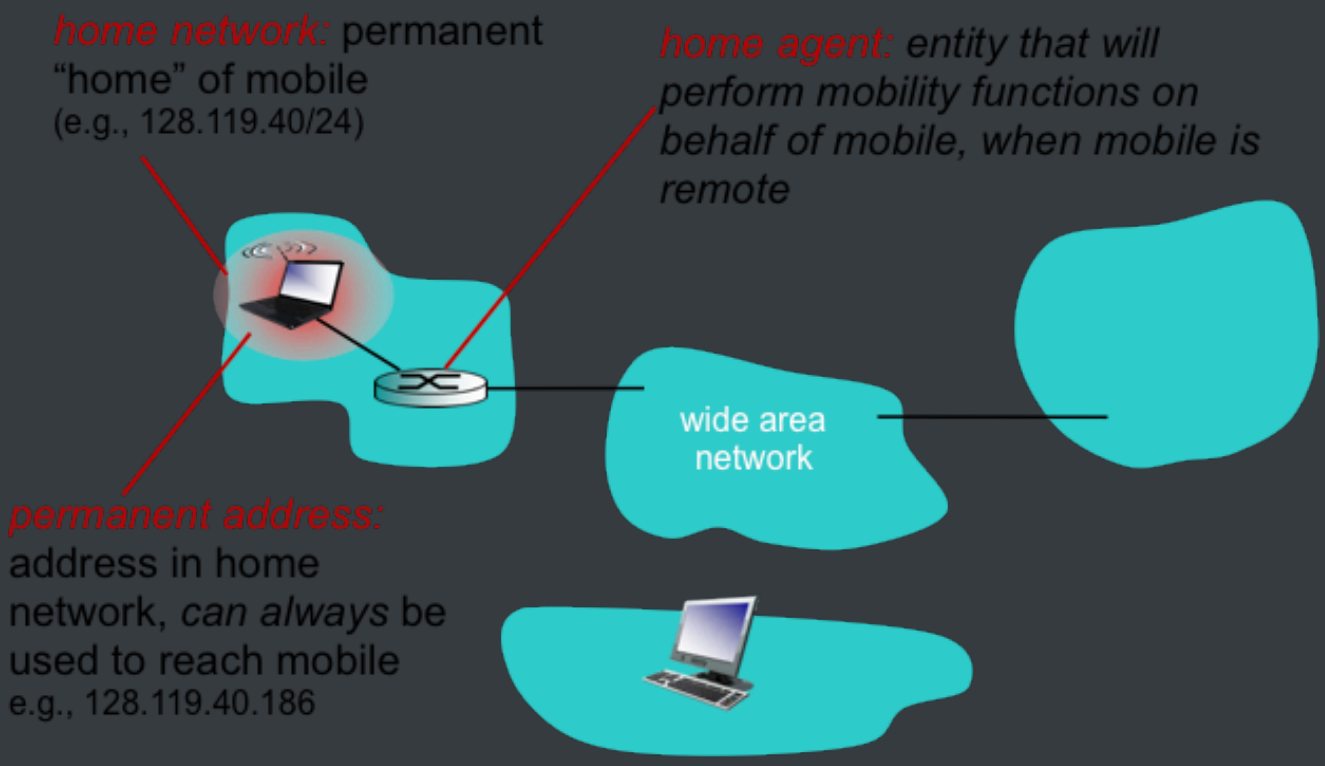
6.42 What is mobility?

- spectrum[光谱] of mobility[移动性], from the *network* perspective:



6.43 Mobility: vocabulary

> Home network[家乡网络], 永久固定的
>
> Home agent[家乡代理], 先转到代理路由器上



1. Home network[家乡网络] *home network:* permanent "home" of mobile (e.g., 128.119.40/24)
2. *home agent: entity that will perform mobility functions on behalf of mobile, when mobile is remote*
3. *permanent address:* address in home network, *can always* be used to reach mobile. e.g., 128.119.40.186

6.46 Mobility: approaches

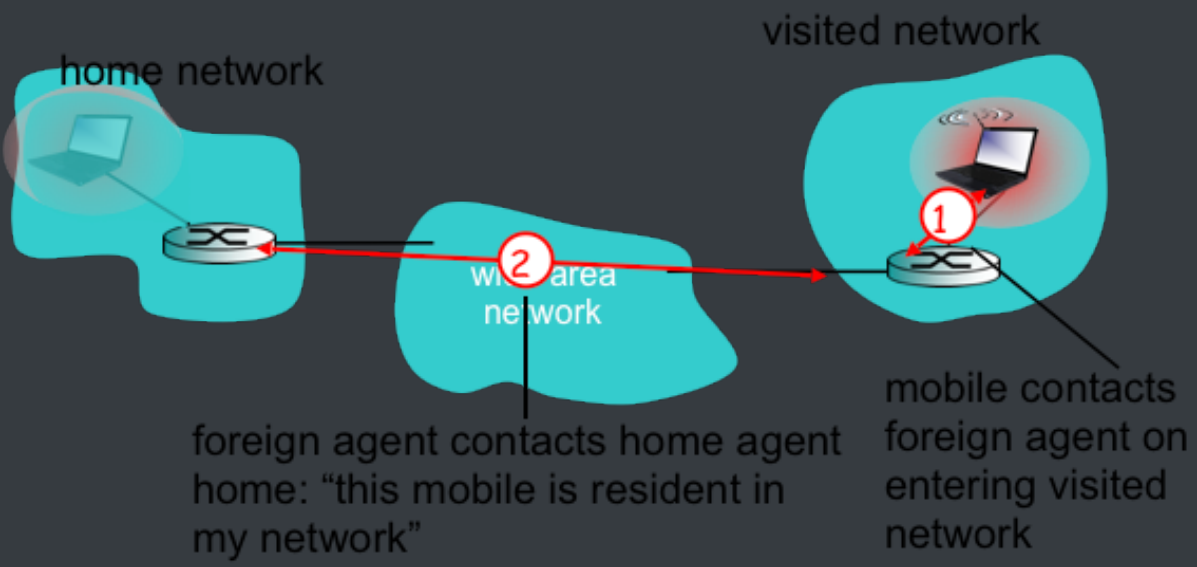
> 路由器工作量大，需要知道各个移动用户所在的位置，节点
>
> 网络的设计：边缘是复杂的，核心是简单的
>
> 无论直接路由，还是间接路由， 这种违反了设计

- 🚫 *let routing handle it:* routers advertise permanent address of mobile-nodes-in-residence via usual routing table exchange.
- - routing tables indicate where each mobile located
  - no changes to end-systems
- *let end-systems handle it:*
- - *indirect routing:* communication from correspondent to mobile goes through home agent, then forwarded to remote
  - *direct routing:* correspondent gets foreign address of mobile, sends directly to mobile

6.47 not scalable to millions of mobiles

not scalable to millions of mobiles

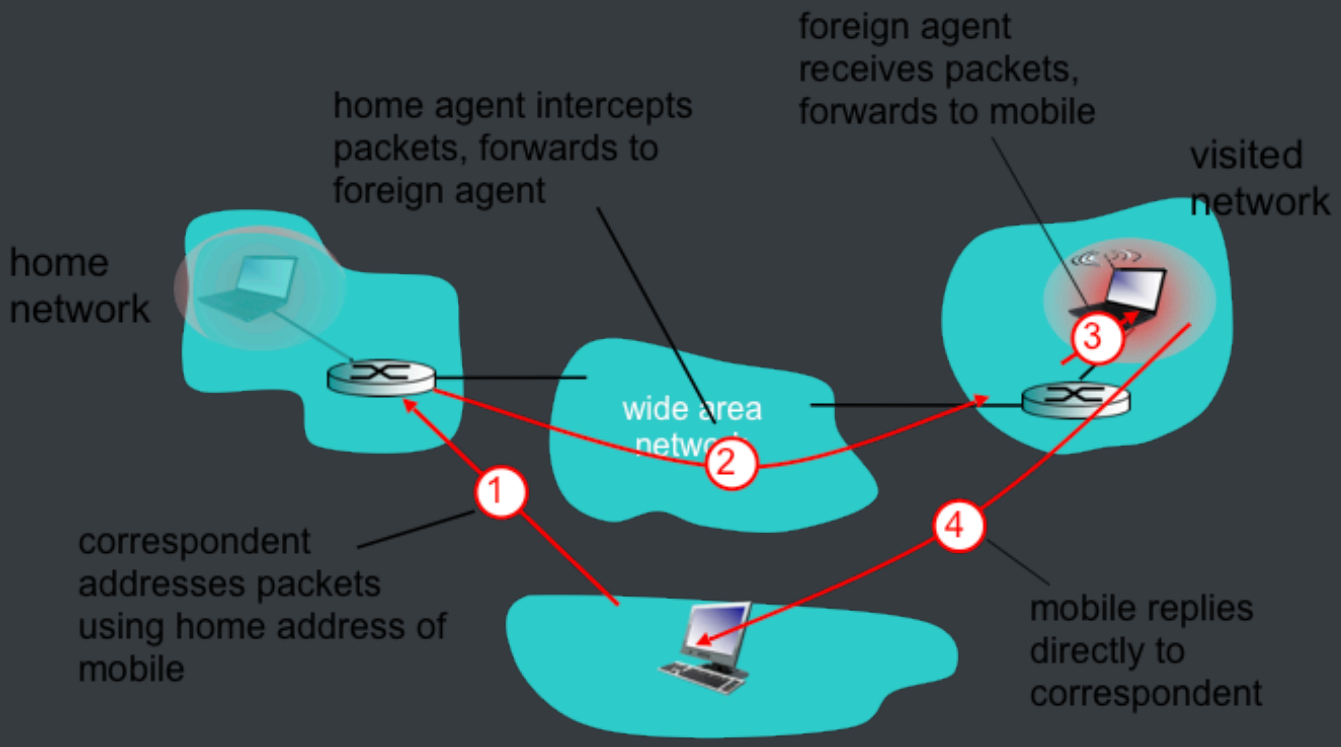
6.48 Mobility: registration



end result:

- foreign agent knows about mobile
- home agent knows location of mobile

6.49 Mobility via indirect routing



6.50 Indirect Routing: comments

> 三角网络
>
> 外部代理告知家乡网络

- mobile uses two addresses:
    - permanent address: used by correspondent (hence mobile location is *transparent* to correspondent)
    - care-of-address: used by home agent to forward datagrams to mobile
- foreign agent functions may be done by mobile itself
- triangle routing: correspondent-home-network-mobile
    - inefficient when

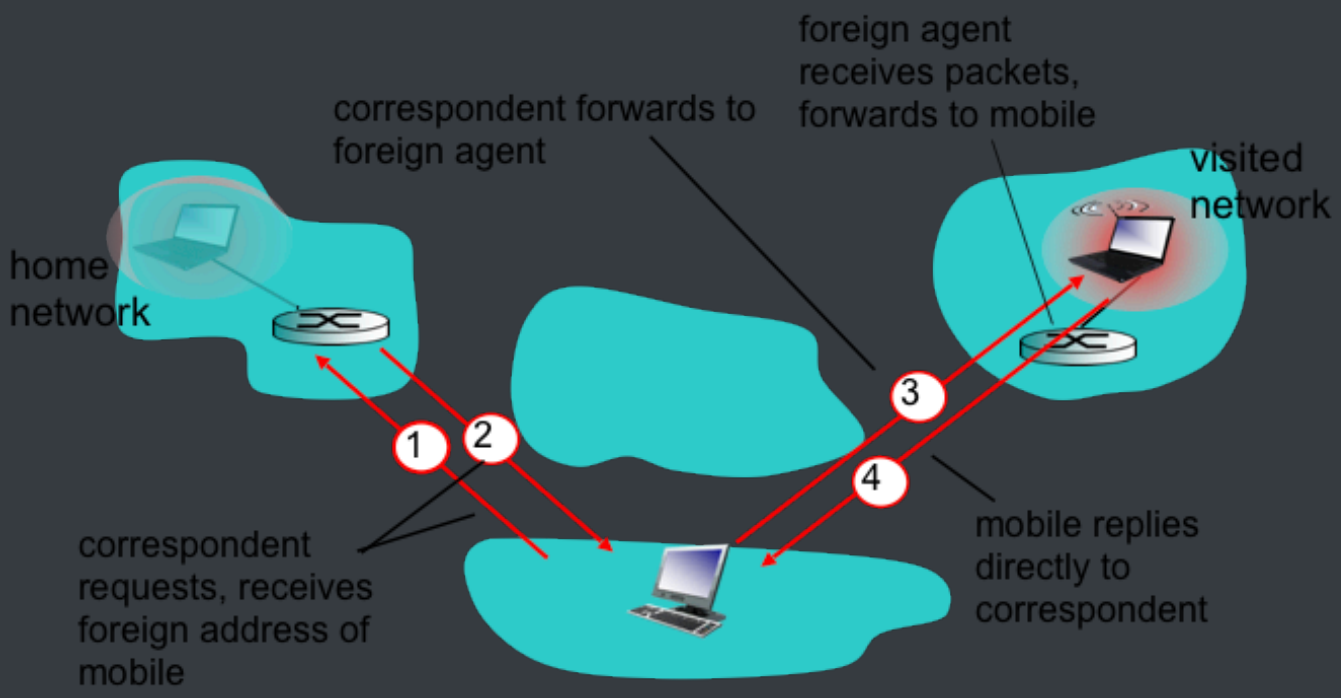
correspondent, mobile are in same network

6.51 Indirect routing: moving between networks

> 记住家乡网络的ip
>
> 间接路由选择有一定透明性

- suppose mobile user moves to another network
    - registers with new foreign agent
    - new foreign agent registers with home agent
    - home agent update care-of-address for mobile
    - packets continue to be forwarded to mobile (but with new care-of-address)
- mobility, changing foreign networks transparent: *on going connections can be maintained!*

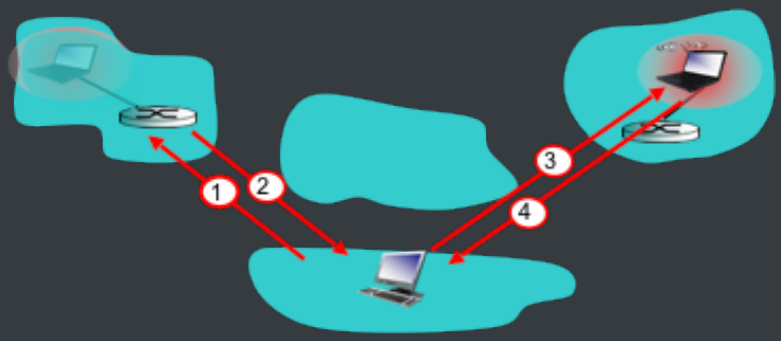
6.52 Mobility via direct routing



6.53 Mobility via direct routing: comments

> 克服三角路由
>
> 但不透明

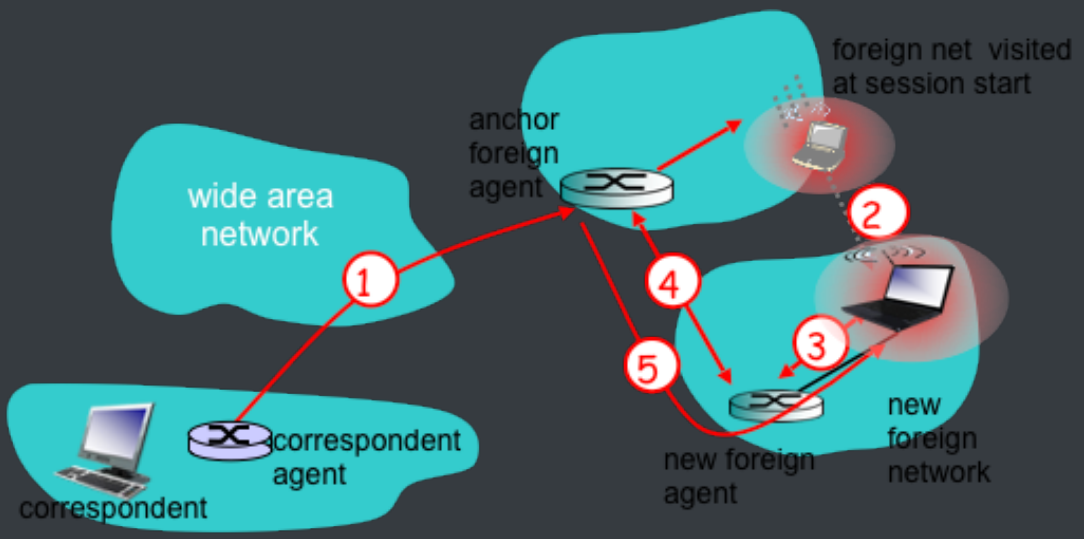
- overcome triangle routing problem
- *non-transparent to correspondent:* correspondent must get care-of-address from home agent
-
  - what if mobile changes visited network?



6.54 Accommodating mobility with direct routing

> 锚⚓的外部代理
>
> :为什么要有锚⚓

- anchor foreign agent: FA in first visited network
- data always routed first to anchor FA
- when mobile moves: new FA arranges to have data forwarded from old FA (chaining)



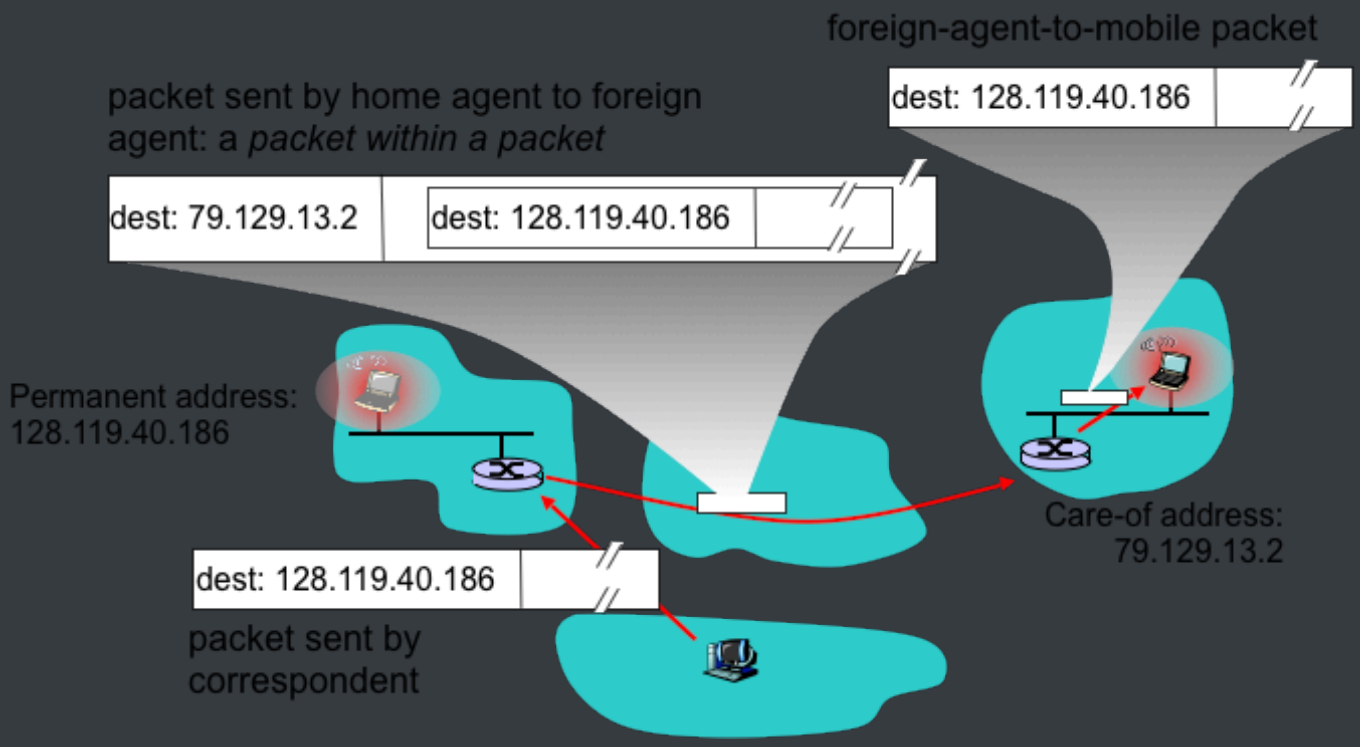
6.6 Mobile IP

6.56 Mobile IP

- RFC 3344

- has many features we've seen:
- - home agents, foreign agents, foreign-agent registration, care-of-addresses, encapsulation (packet-within-a-packet)
- three components to standard:
- - indirect routing of datagrams
  - agent discovery
  - registration with home agent

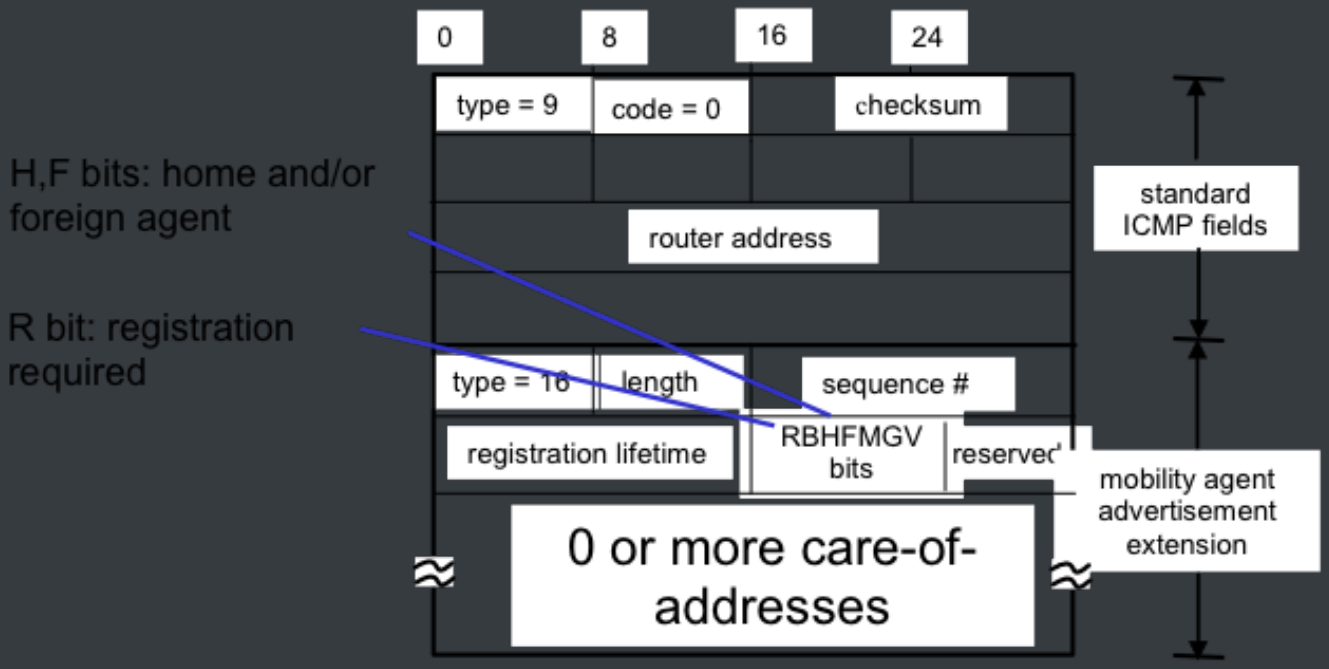
6.57 Mobile IP: indirect routing



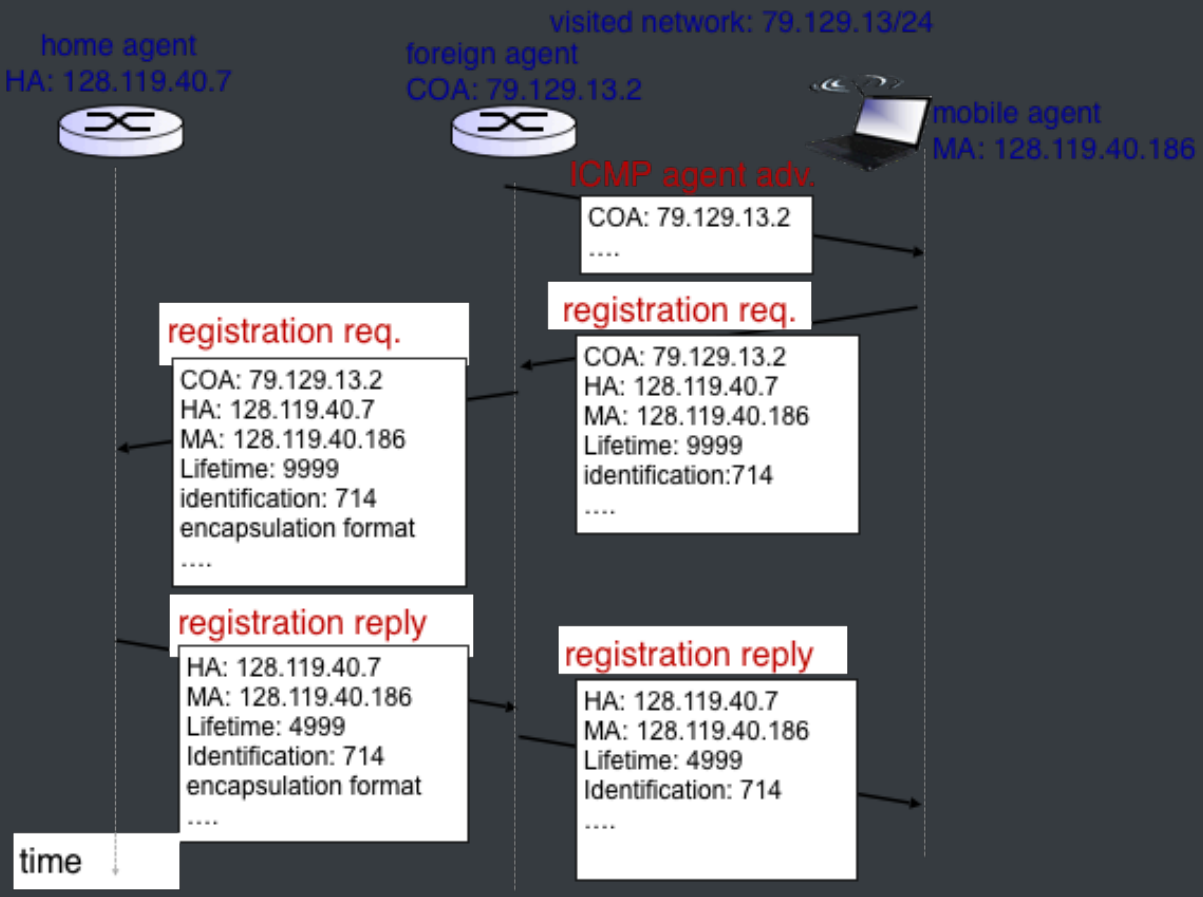
6.58 Mobile IP: agent discovery

- *agent advertisement:* foreign/home agents advertise service by broadcasting ICMP messages (typefield = 9)

- *agent advertisement:* foreign/home agents advertise service by broadcasting ICMP messages (typefield = 9)



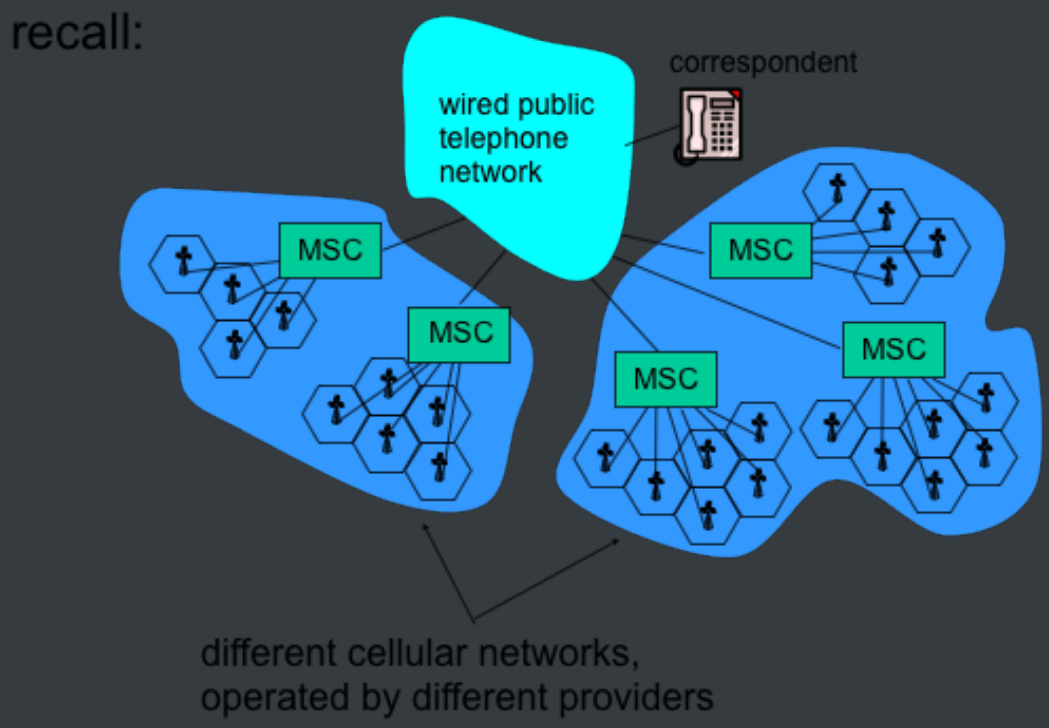
6.59 Mobile IP: registration example



6.7 Handling mobility in cellular networks

6.60 Components of cellular network architecture

> 蜂窝网络， 也是来自早期的电信网络



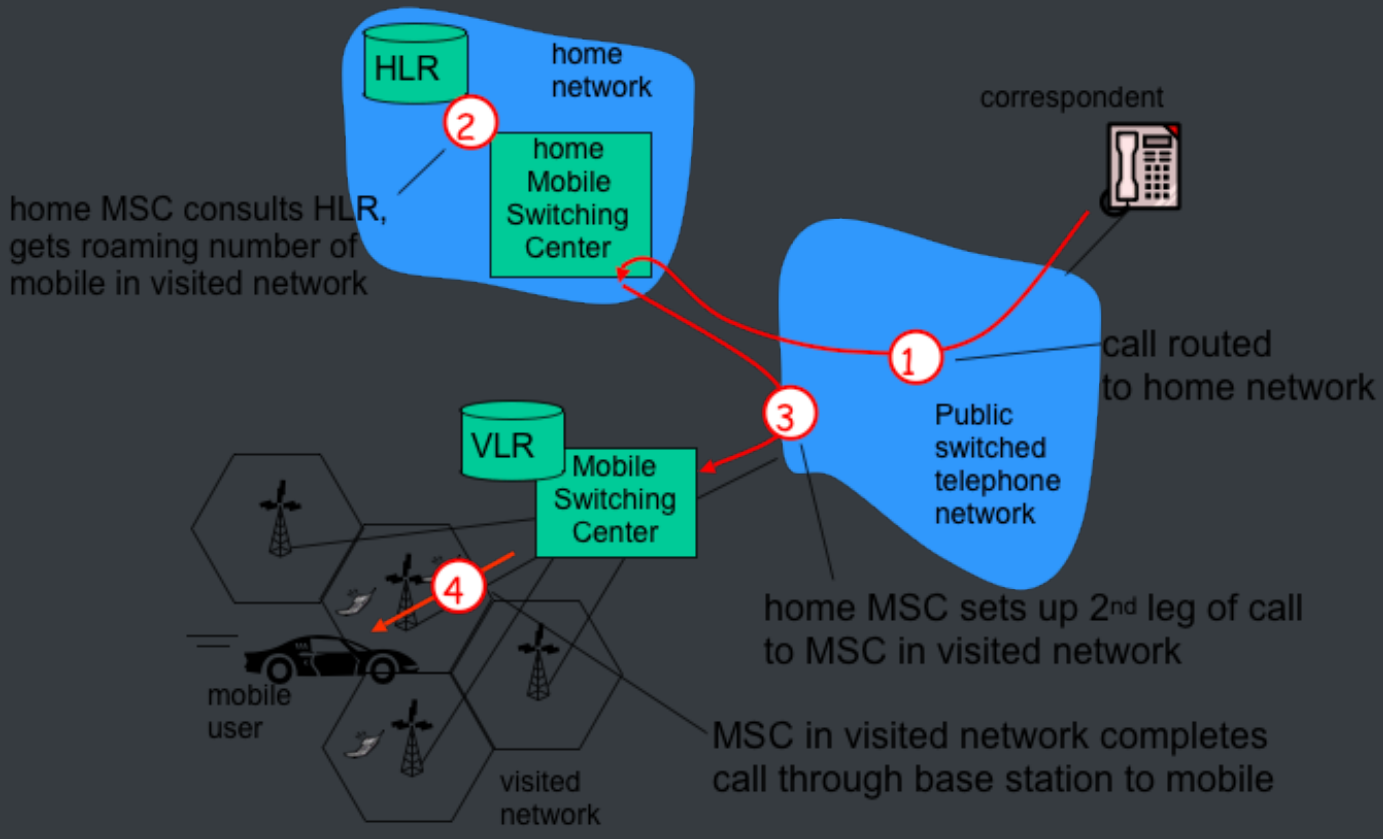
6.61 Handling mobility in cellular networks

- *home network*: network of cellular provider you subscribe to (e.g., Sprint PCS, Verizon)
- - *home location register (HLR):* database in home network containing permanent cell phone #, profile information (services, preferences, billing), information about current location (could be in another network)
- *visited network:* network in which mobile currently resides
- - *visitor location register (VLR):* database with entry for each user currently in network
  - could be home network

6.8 Mobility and higher-layer protocols

6.62 GSM: indirect routing to mobile

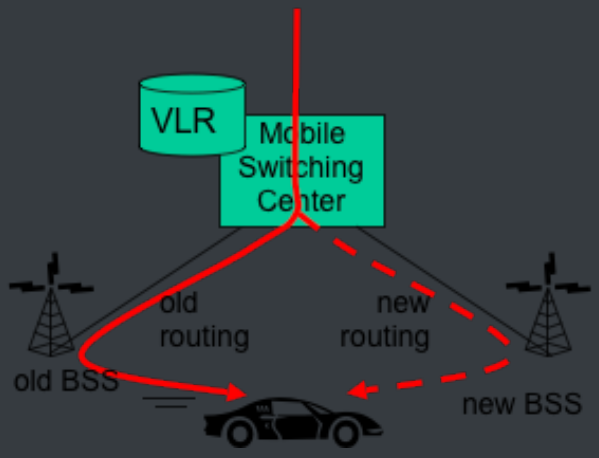
> HLR (Home Location Register)
>
> VLR (Visitor Location Register)
>
> MSC (Mobile Switching Center)



6.63 GSM: handoff with common MSC

- *handoff goal:* route call via new base station (without interruption)
- reasons for handoff:

- - stronger signal to/from new BSS (continuing connectivity, less battery drain)
  - load balance: free up channel in current BSS
  - GSM doesnt mandate why to perform handoff (policy), only how (mechanism)
- handoff initiated by old BSS



6.64 GSM: handoff with common MSC

\1. old BSS informs MSC of impending handoff, provides list of 1+ new BSSs

\2. MSC sets up path (allocates resources) to new BSS

\3. new BSS allocates radio channel for use by mobile

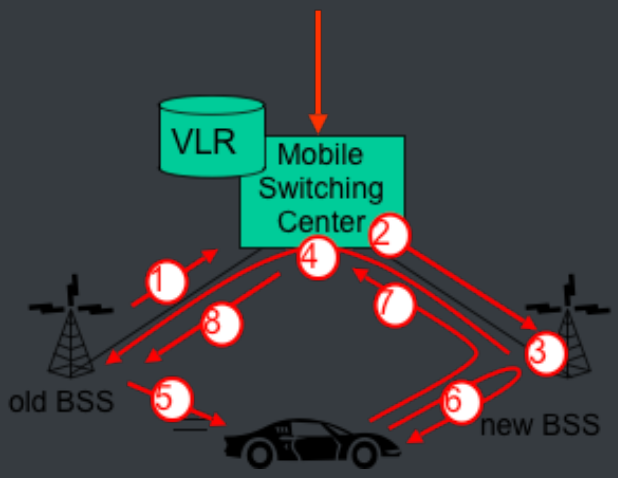
\4. new BSS signals MSC, old BSS: ready

\5. old BSS tells mobile: perform handoff to new BSS

\6. mobile, new BSS signal to activate new channel

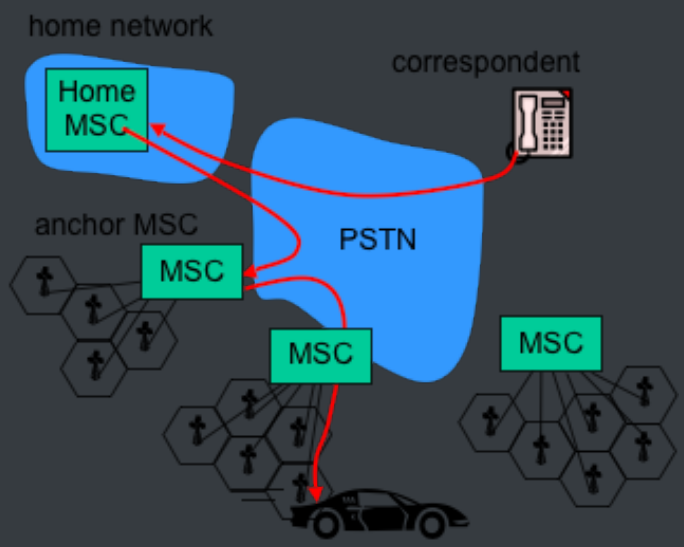
\7. mobile signals via new BSS to MSC: handoff complete. MSC reroutes call

8 MSC-old-BSS resources released



6.65 GSM: handoff between MSCs

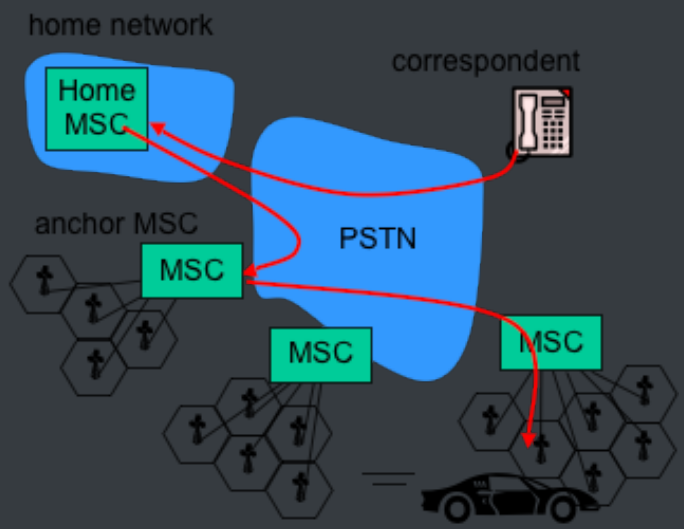
- *anchor MSC:* first MSC visited during call
- - call remains routed through anchor MSC
- new MSCs add on to end of MSC chain as mobile moves to new MSC
- optional path minimization step to shorten multi-MSC chain



(a) before handoff

6.66 GSM: handoff between MSCs

- *anchor MSC:* first MSC visited during call
- - call remains routed through anchor MSC
- new MSCs add on to end of MSC chain as mobile moves to new MSC
- optional path minimization step to shorten multi-MSC chain



(b) after handoff

6.67 Mobility: GSM versus Mobile IP

GSM element	Comment on GSM element	Mobile IP element
Home system	Network to which mobile user's permanent phone number belongs	**Home network**
Gateway Mobile Switching Center, or "home MSC". Home Location Register (HLR)	Home MSC: point of contact to obtain routable address of mobile user. HLR: database in home system containing permanent phone number, profile information, current location of mobile user, subscription information	**Home agent**
Visited System	Network other than home system where mobile user is currently residing	**Visited network**
Visited Mobile services Switching Center. Visitor Location Record (VLR)	Visited MSC: responsible for setting up calls to/from mobile nodes in cells associated with MSC. VLR: temporary database entry in visited system, containing subscription information for each visiting mobile user	**Foreign agent**
Mobile Station Roaming Number (MSRN), or "roaming number"	Routable address for telephone call segment between home MSC and visited MSC, visible to neither the mobile nor the correspondent.	**Care-of-address**

6.68 Wireless, mobility: impact on higher layer protocols

- logically, impact *should* be minimal …
- - best effort service model remains unchanged
  - TCP and UDP can (and do) run over wireless, mobile
- … but performance-wise:
- - packet loss/delay due to bit-errors (discarded packets, delays for link-layer retransmissions), and handoff
  - TCP interprets loss as congestion, will decrease congestion window un-necessarily
  - delay impairments for real-time traffic

        - limited bandwidth of wireless links

6.9 Summary

Wireless

- wireless links:
    - capacity, distance
    - channel impairments
    - CDMA
- IEEE 802.11 (“Wi-Fi”)
    - CSMA/CA reflects wireless channel characteristics
- cellular access
    - architecture
    - standards (e.g., GSM, 3G, 4G LTE)

Mobility

- principles: addressing, routing to mobile users
    - home, visited networks
    - direct, indirect routing
    - care-of-addresses
- case studies
    - mobile IP
    - mobility in GSM
- impact on higher-layer protocols