

Chapter I

Intro to Networks

© *The materials in these notes are adapted from Computer Networking: A Top Down Approach, 6th edition, by Jim Kurose, Keith Ross*

Chapter I: introduction

our goal:

- ❖ get “feel” and terminology
- ❖ more depth, detail
later in course
- ❖ approach:
 - use Internet as example

overview:

- ❖ what’s the Internet?
- ❖ what’s a protocol?
- ❖ network edge; hosts, access net, physical media
- ❖ network core: packet/circuit switching, Internet structure
- ❖ performance: loss, delay, throughput
- ❖ security
- ❖ protocol layers, service models
- ❖ history

Chapter I: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

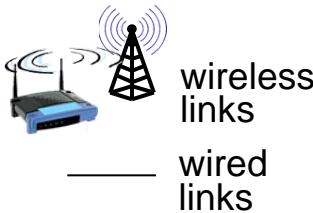
I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

I.6 networks under attack: security

I.7 history

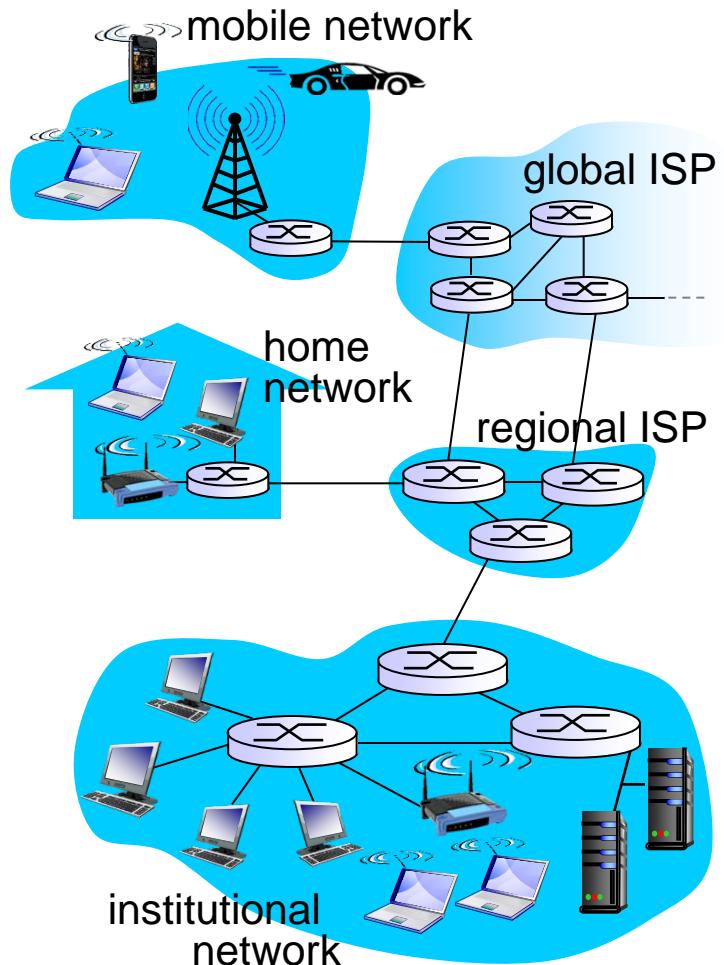
What's the Internet: “nuts and bolts” view



- ❖ millions of connected computing devices:
 - *hosts = end systems*
 - running *network apps*

- ❖ communication links
 - fiber, copper, radio, satellite
 - transmission rate: *bandwidth*

- ❖ *Packet switches: forward packets (chunks of data)*
 - *routers and switches*



“Fun” internet appliances



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



Web-enabled toaster +
weather forecaster



Internet
refrigerator



Slingbox: watch,
control cable TV remotely



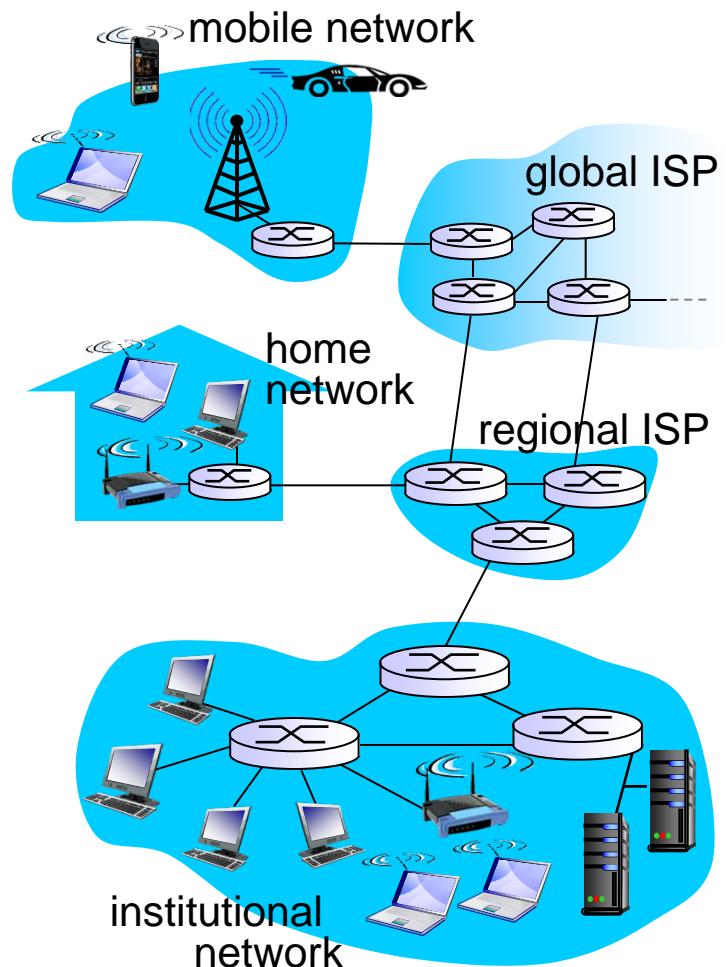
Tweet-a-watt:
monitor energy use



Internet phones

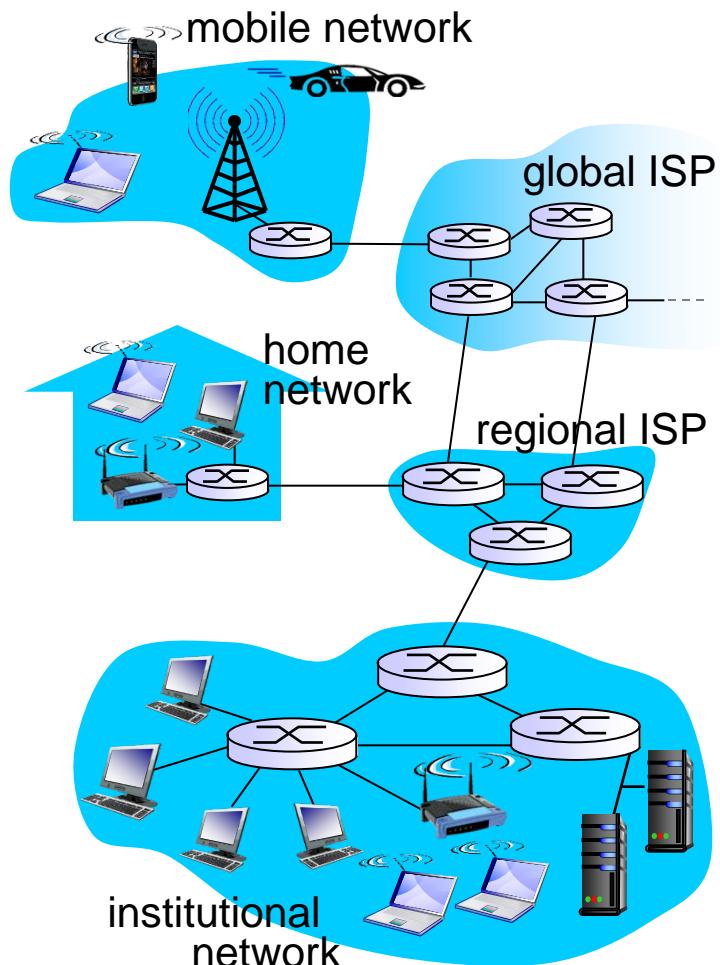
What's the Internet: “nuts and bolts” view

- ❖ *Internet: “network of networks”*
 - Interconnected ISPs
- ❖ *protocols* control sending, receiving of msgs
 - e.g., TCP, IP, HTTP, Skype, 802.11
- ❖ *Internet standards*
 - RFC: Request for comments
 - IETF: Internet Engineering Task Force



What's the Internet: a service view

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



What's a protocol?

human protocols:

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

... specific msgs sent

... specific actions taken
when msgs received, or
other events

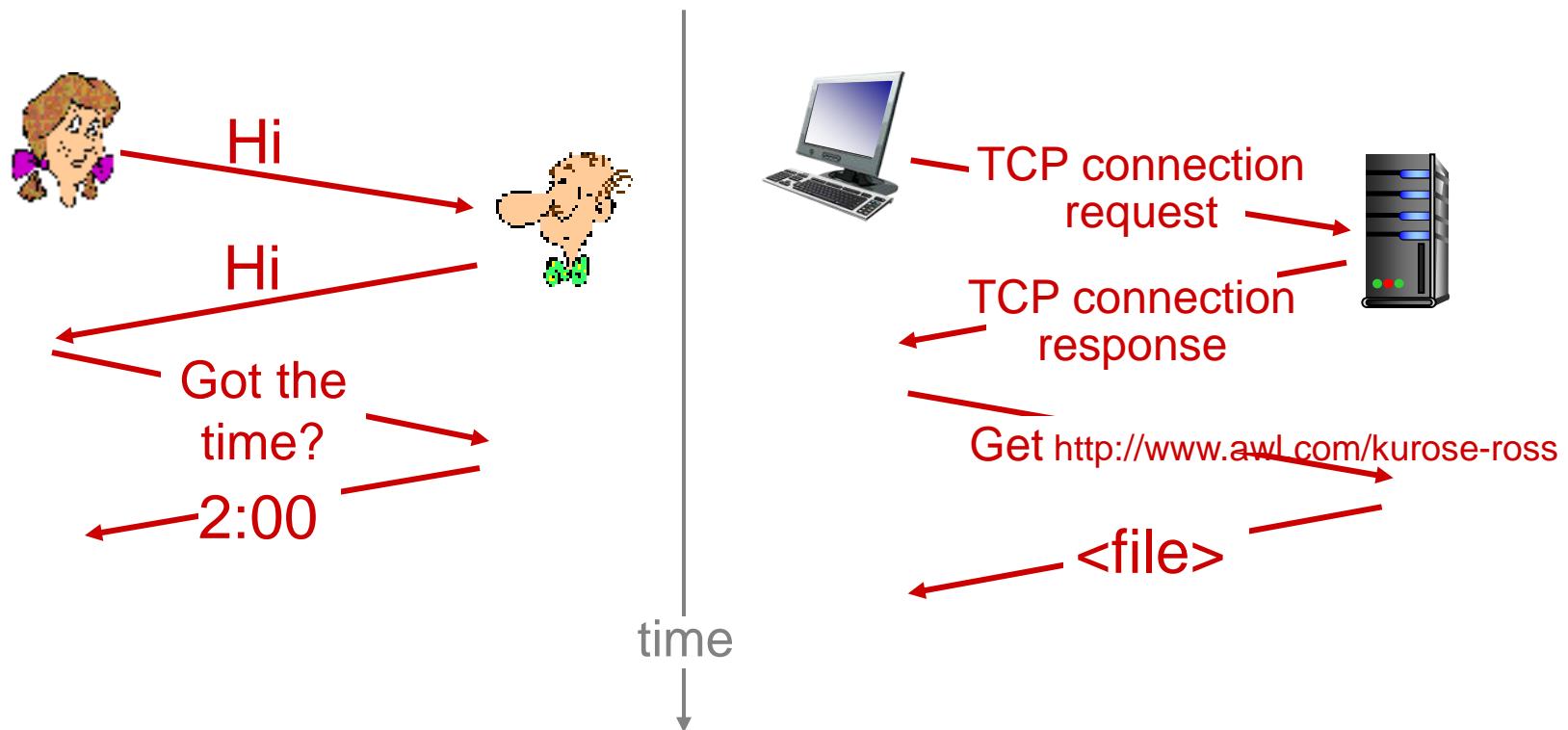
network protocols:

- ❖ 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*

What's a protocol?

a human protocol and a computer network protocol:



Q: other human protocols?

Chapter I: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

I.6 networks under attack: security

I.7 history

A closer look at network structure:

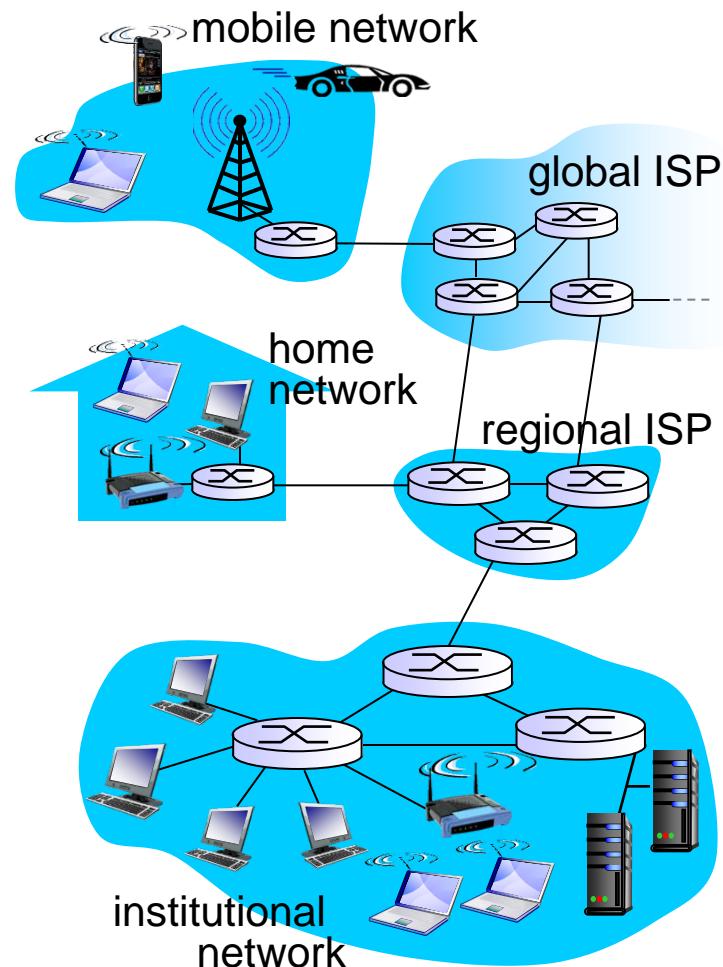
❖ *network edge:*

- hosts: clients and servers
- servers often in data centers

❖ *access networks, physical media:* wired, wireless communication links

❖ *network core:*

- interconnected routers
- network of networks



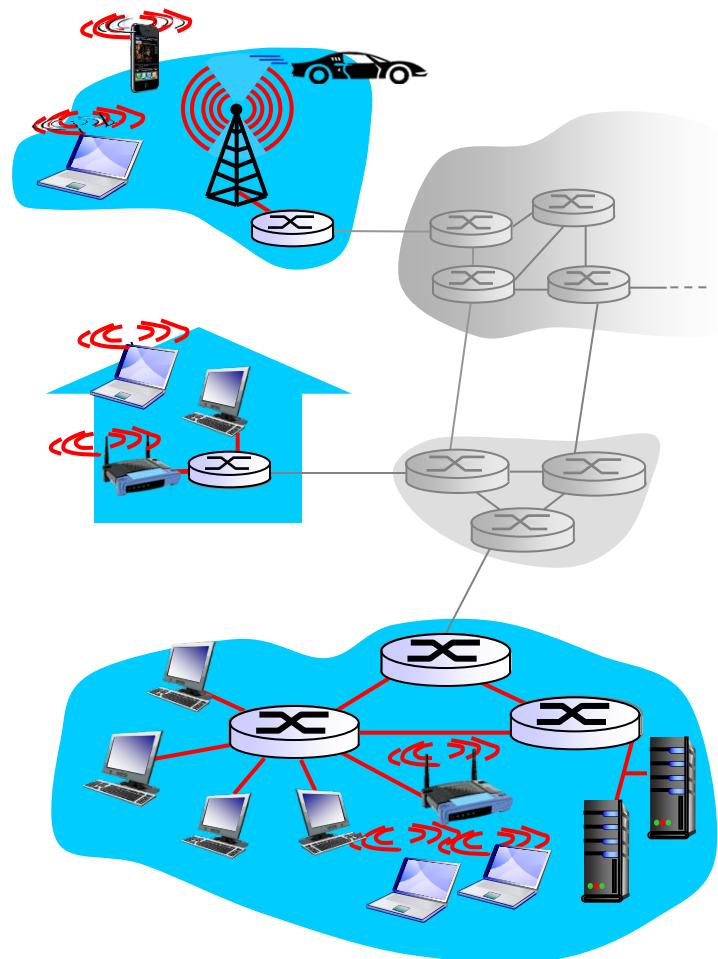
Access networks and physical media

Q: How to connect end systems to edge router?

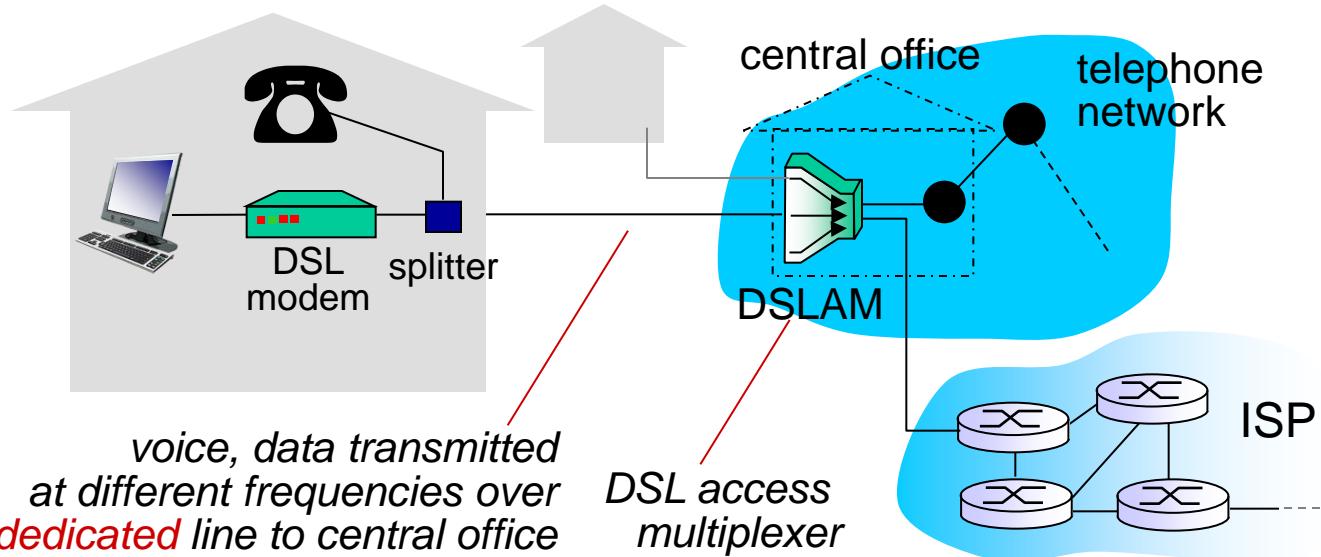
- ❖ residential access nets
- ❖ institutional access networks (school, company)
- ❖ mobile access networks

keep in mind:

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

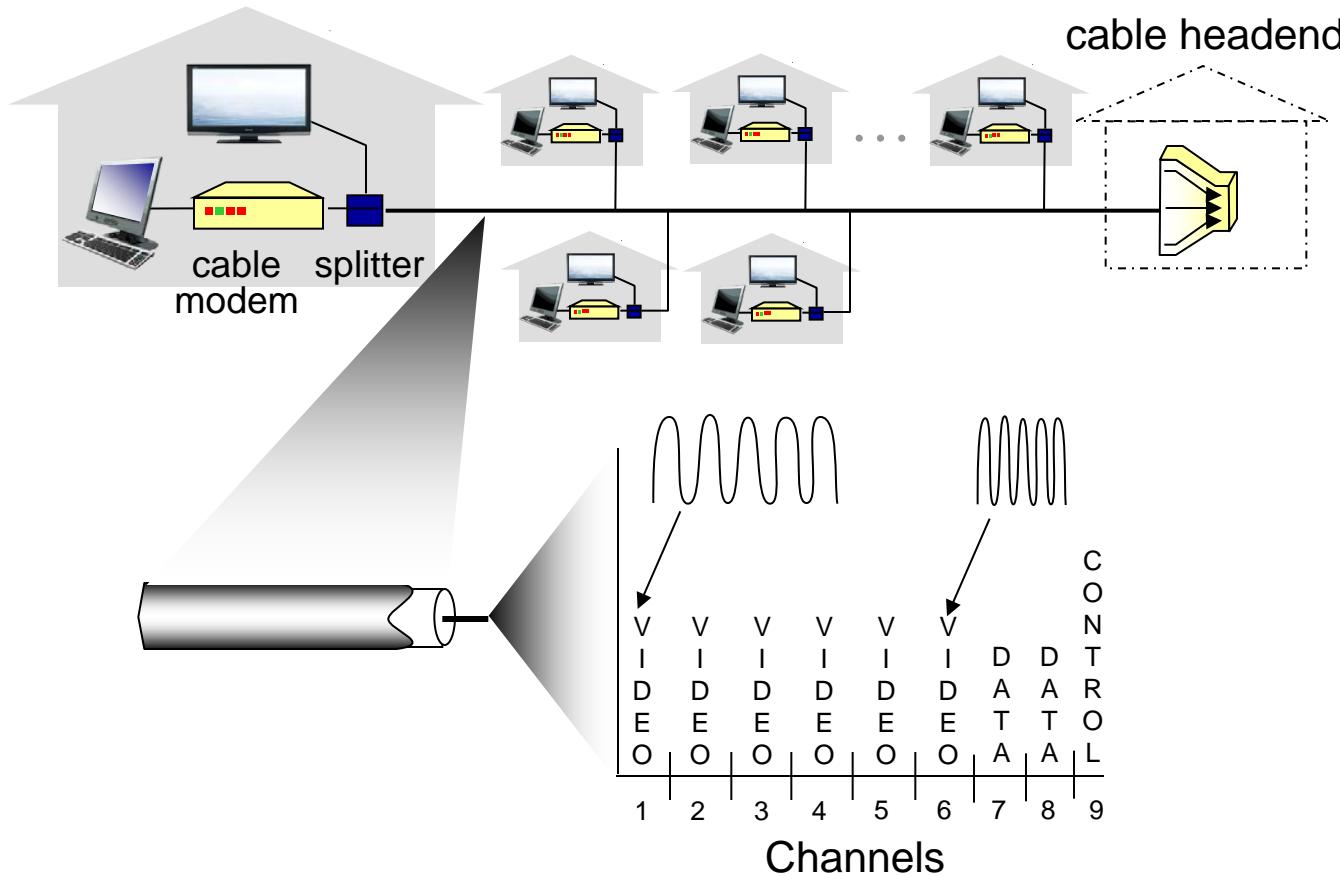


Access net: digital subscriber line (DSL)



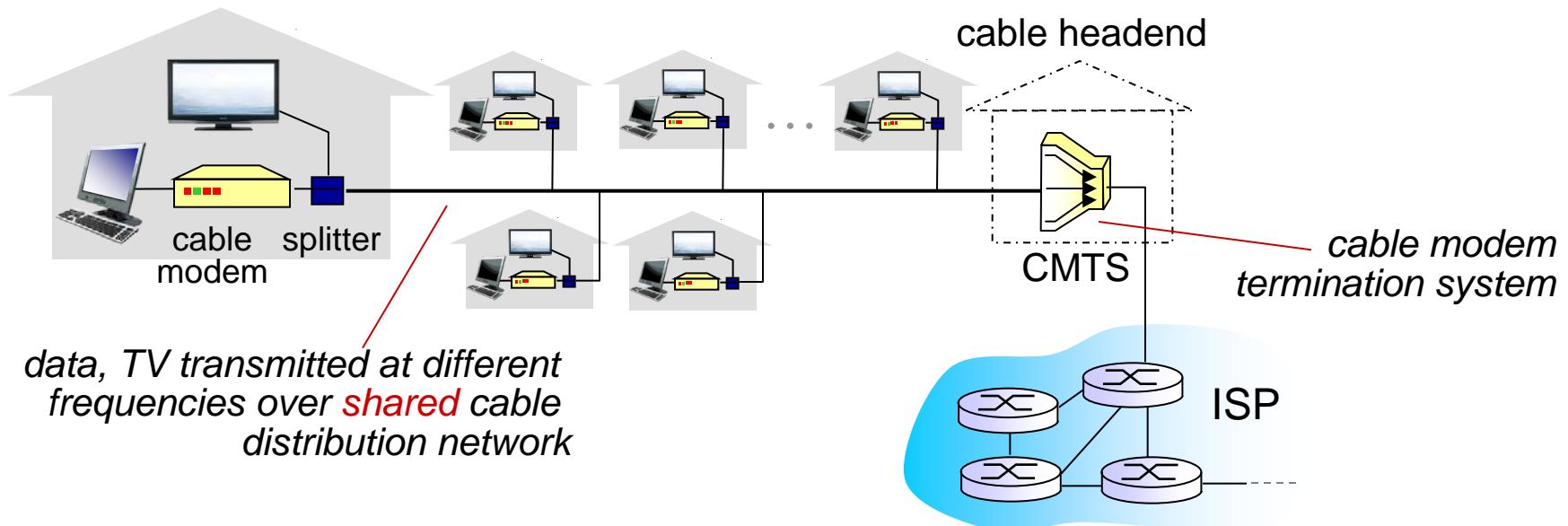
- ❖ use **existing** telephone line to central office DSLAM
 - data over DSL phone line goes to Internet
 - voice over DSL phone line goes to telephone net
- ❖ < 2.5 Mbps upstream transmission rate (typically < 1 Mbps)
- ❖ < 24 Mbps downstream transmission rate (typically < 10 Mbps)

Access net: cable network



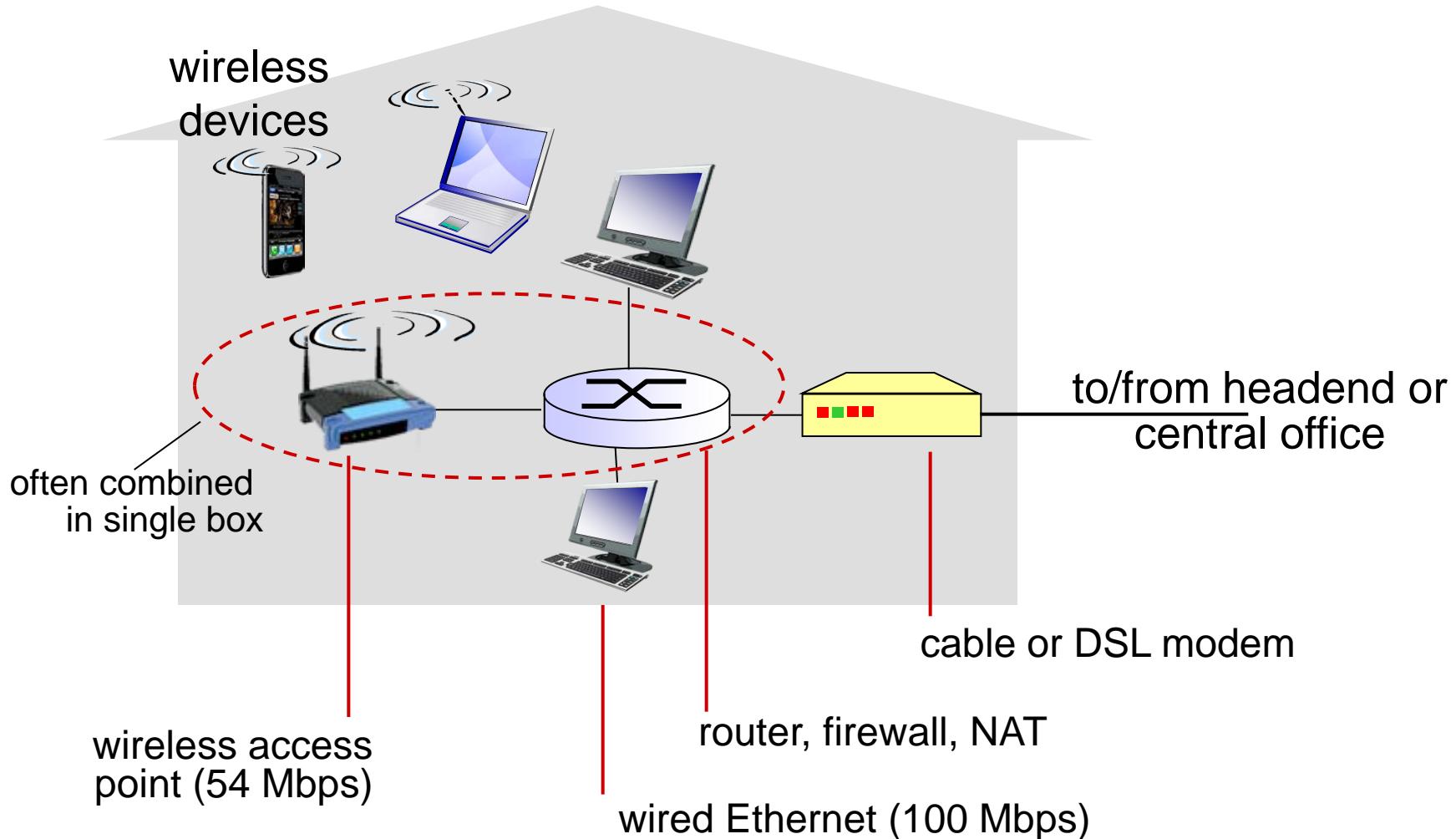
frequency division multiplexing: different channels transmitted in different frequency bands

Access net: cable network

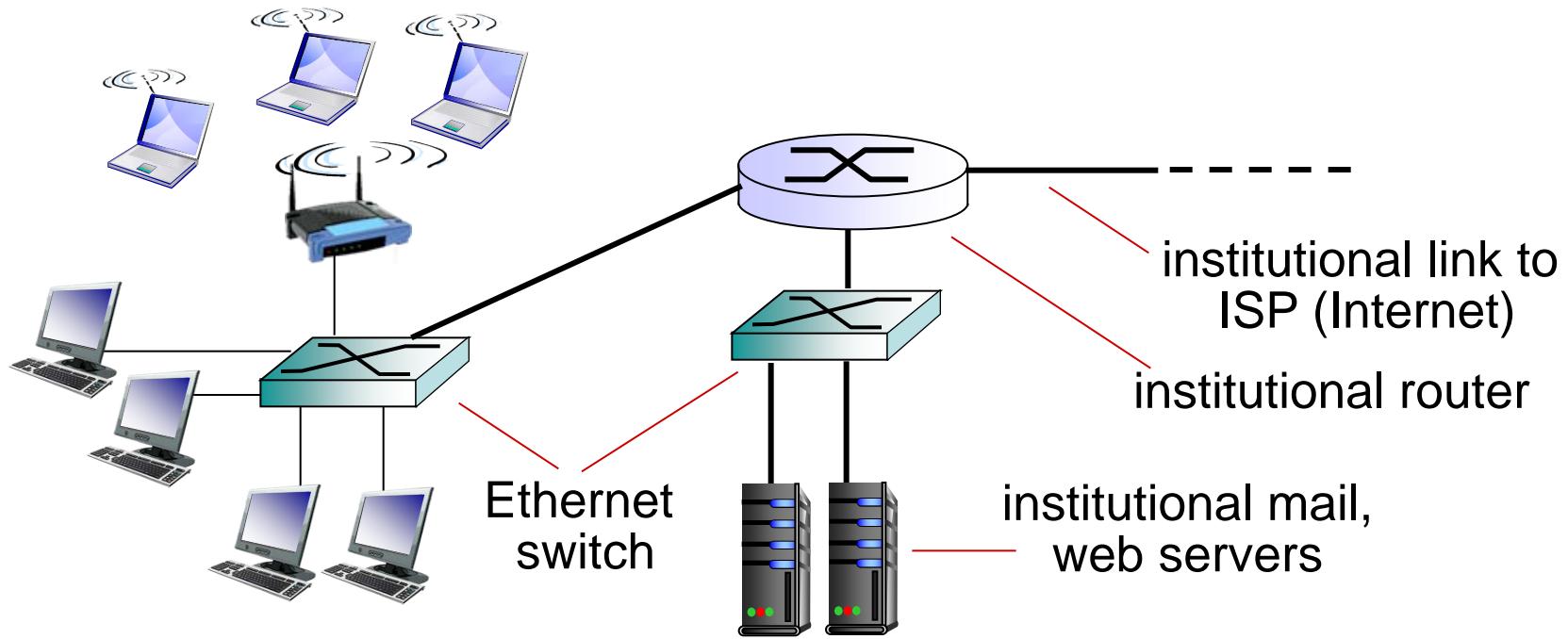


- ❖ HFC: hybrid fiber coax
 - asymmetric: up to 30Mbps downstream transmission rate, 2 Mbps upstream transmission rate
- ❖ network of cable, fiber attaches homes to ISP router
 - homes **share access network** to cable headend
 - unlike DSL, which has dedicated access to central office

Access net: home network



Enterprise access networks (Ethernet)



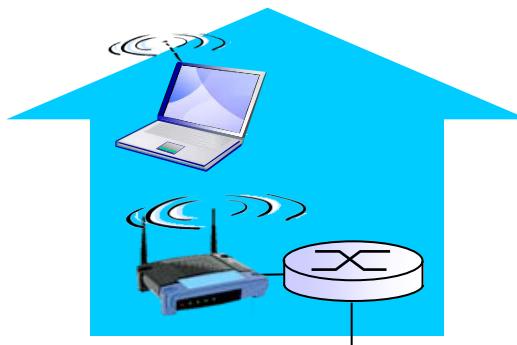
- ❖ typically used in companies, universities, etc
- ❖ 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- ❖ today, end systems typically connect into Ethernet switch

Wireless access networks

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

wireless LANs:

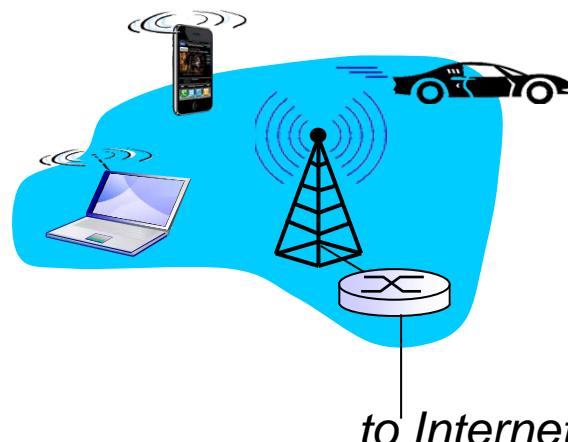
- within building (100 ft)
- 802.11b/g (WiFi): 11, 54 Mbps transmission rate



to Internet

wide-area wireless access

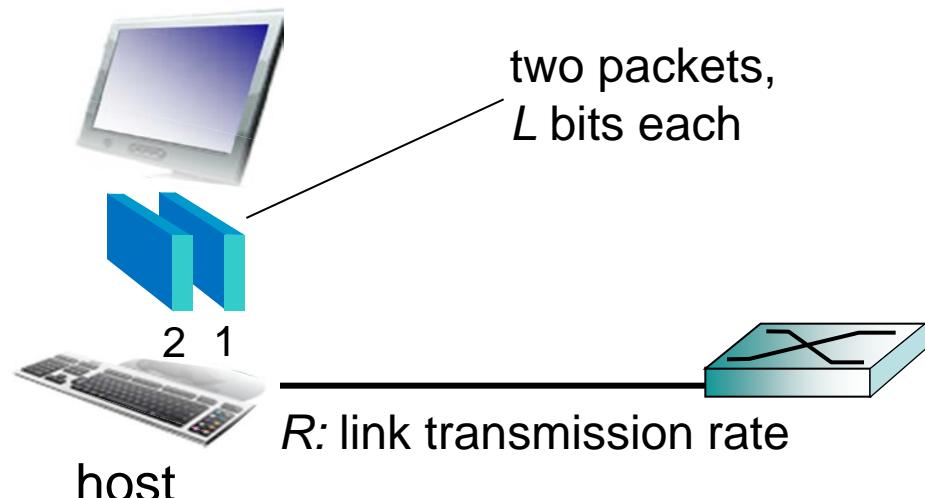
- provided by telco (cellular) operator, 10's km
- between 1 and 10 Mbps
- 3G, 4G: LTE



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*



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

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 Gbps Ethernet
 - Category 6: 10Gbps



Physical media: coax, fiber

coaxial cable:

- ❖ two concentric copper conductors
- ❖ bidirectional
- ❖ broadband:
 - multiple channels on cable
 - HFC



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 Gbps transmission rate)
- ❖ low error rate:
 - repeaters spaced far apart (up to 100 km)
 - immune to electromagnetic noise



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

Chapter I: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

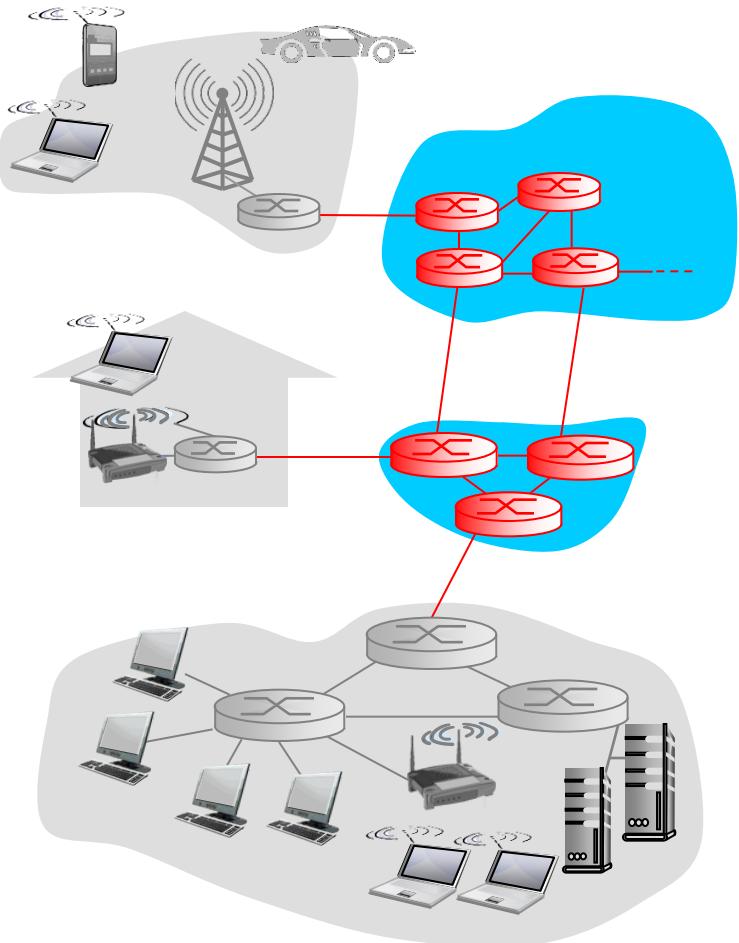
I.5 protocol layers, service models

I.6 networks under attack: security

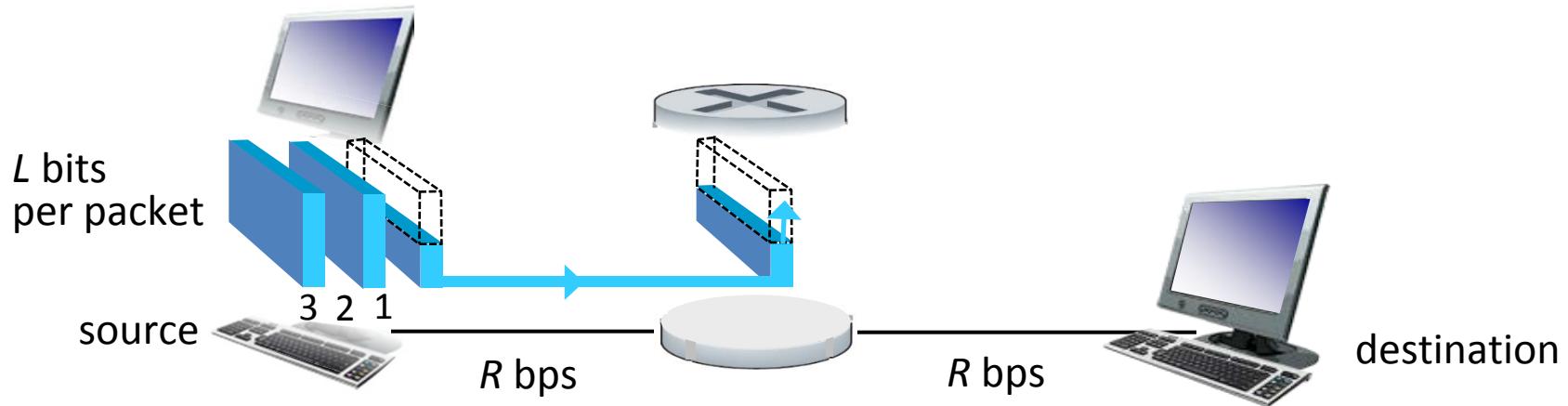
I.7 history

The network core

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



Packet-switching: store-and-forward



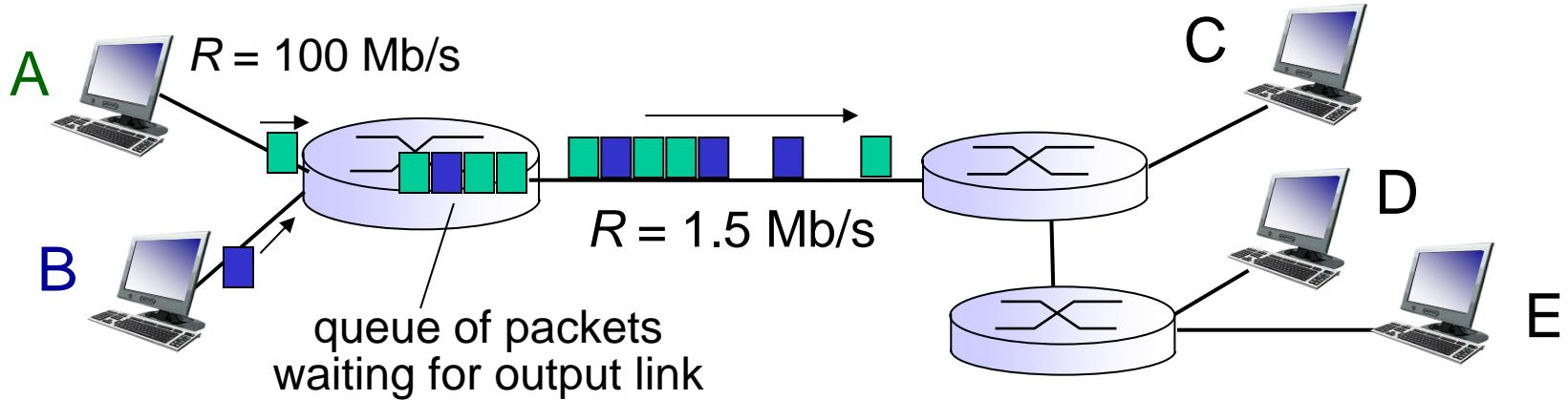
- ❖ takes L/R seconds to transmit (push out) L -bit packet into link at R bps
- ❖ **store and forward:** entire packet must arrive at router before it can be transmitted on next link
- ❖ end-end delay = $2L/R$ (assuming zero propagation delay)

one-hop numerical example:

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

} more on delay shortly ...

Packet Switching: queueing delay, loss



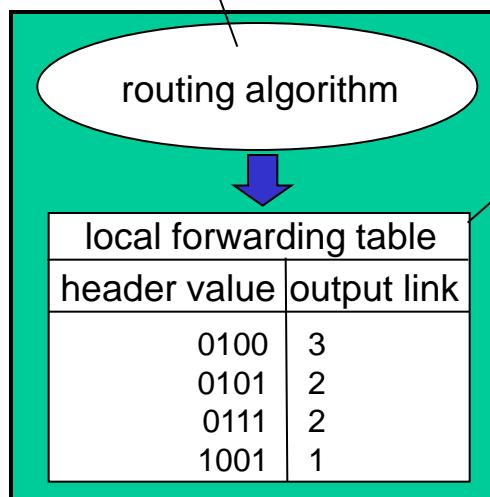
queuing and loss:

- ❖ 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

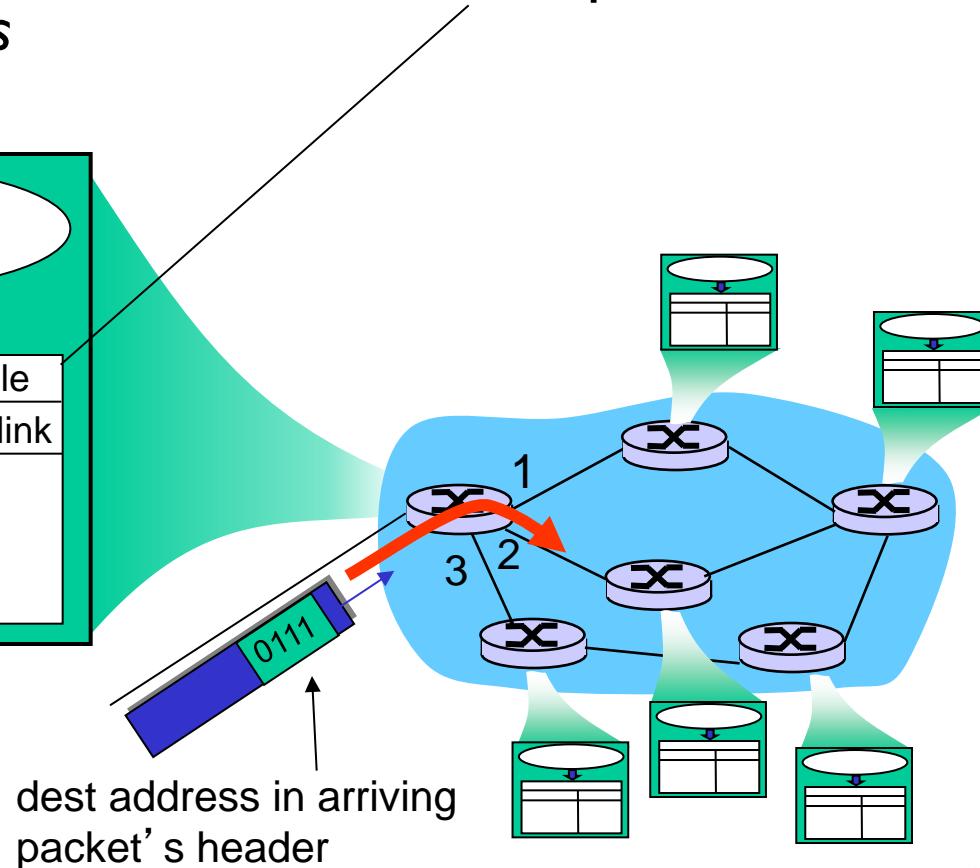
Two key network-core functions

routing: determines source-destination route taken by packets

- *routing algorithms*



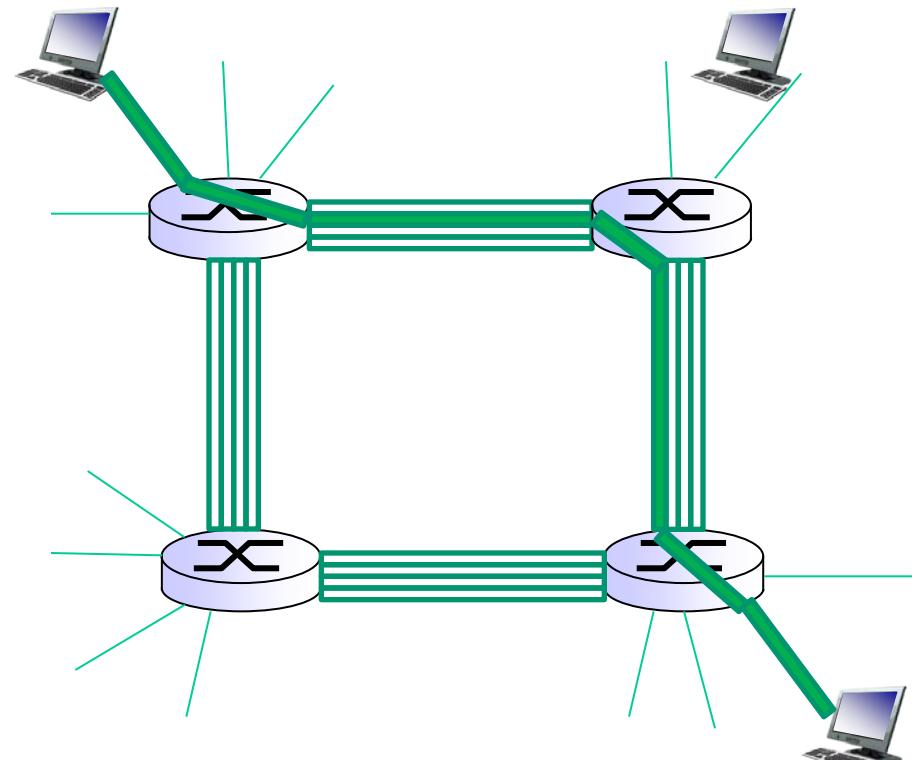
forwarding: move packets from router's input to appropriate router output



Alternative core: circuit switching

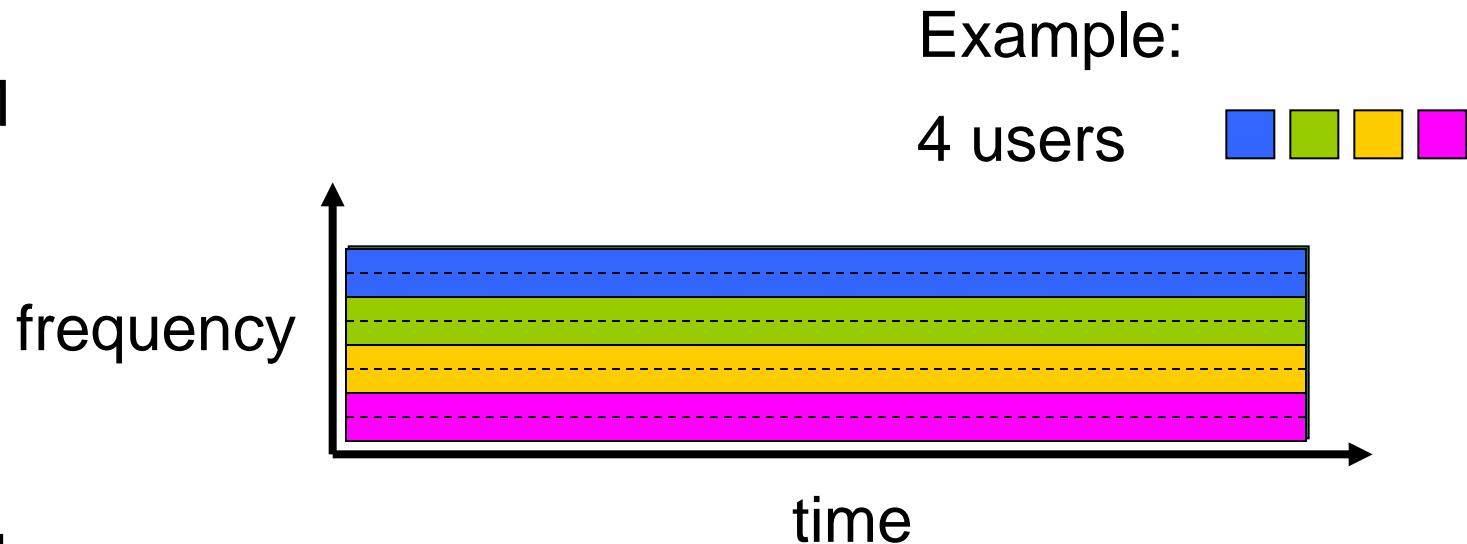
end-end resources allocated
to, reserved for “call”
between source & dest:

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

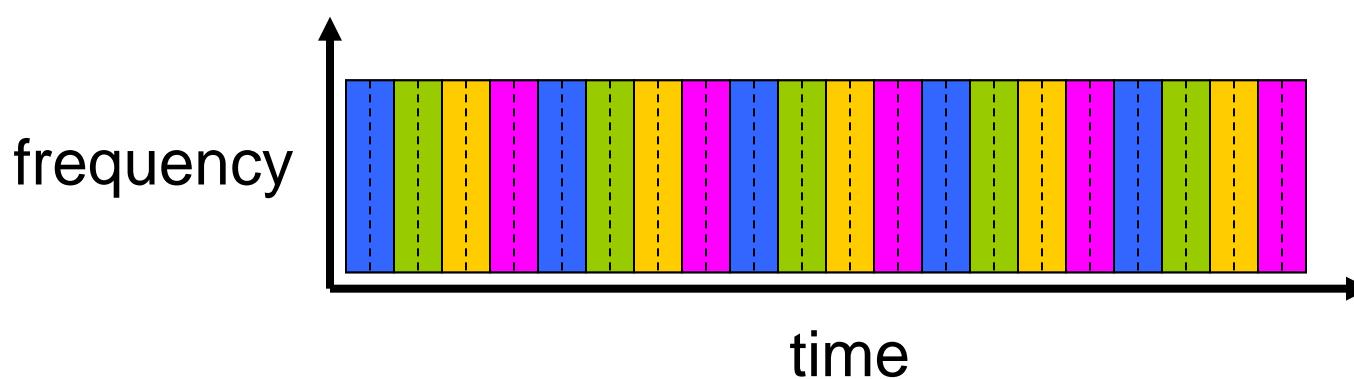


Circuit switching: FDM versus TDM

FDM



TDM

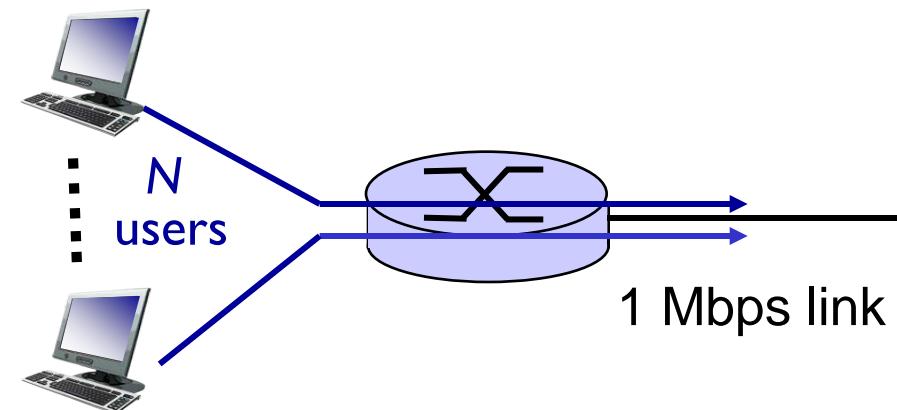


Packet switching versus circuit switching

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 ?

* Check out the online interactive exercises for more examples

Packet switching versus circuit switching

is packet switching a “slam dunk winner?”

- ❖ great for bursty data
 - resource sharing
 - simpler, no call setup
- ❖ excessive congestion possible: packet delay and loss
 - protocols needed for reliable data transfer, congestion control
- ❖ Q: How to provide circuit-like behavior?
 - bandwidth guarantees needed for audio/video apps
 - still an unsolved problem (chapter 7)

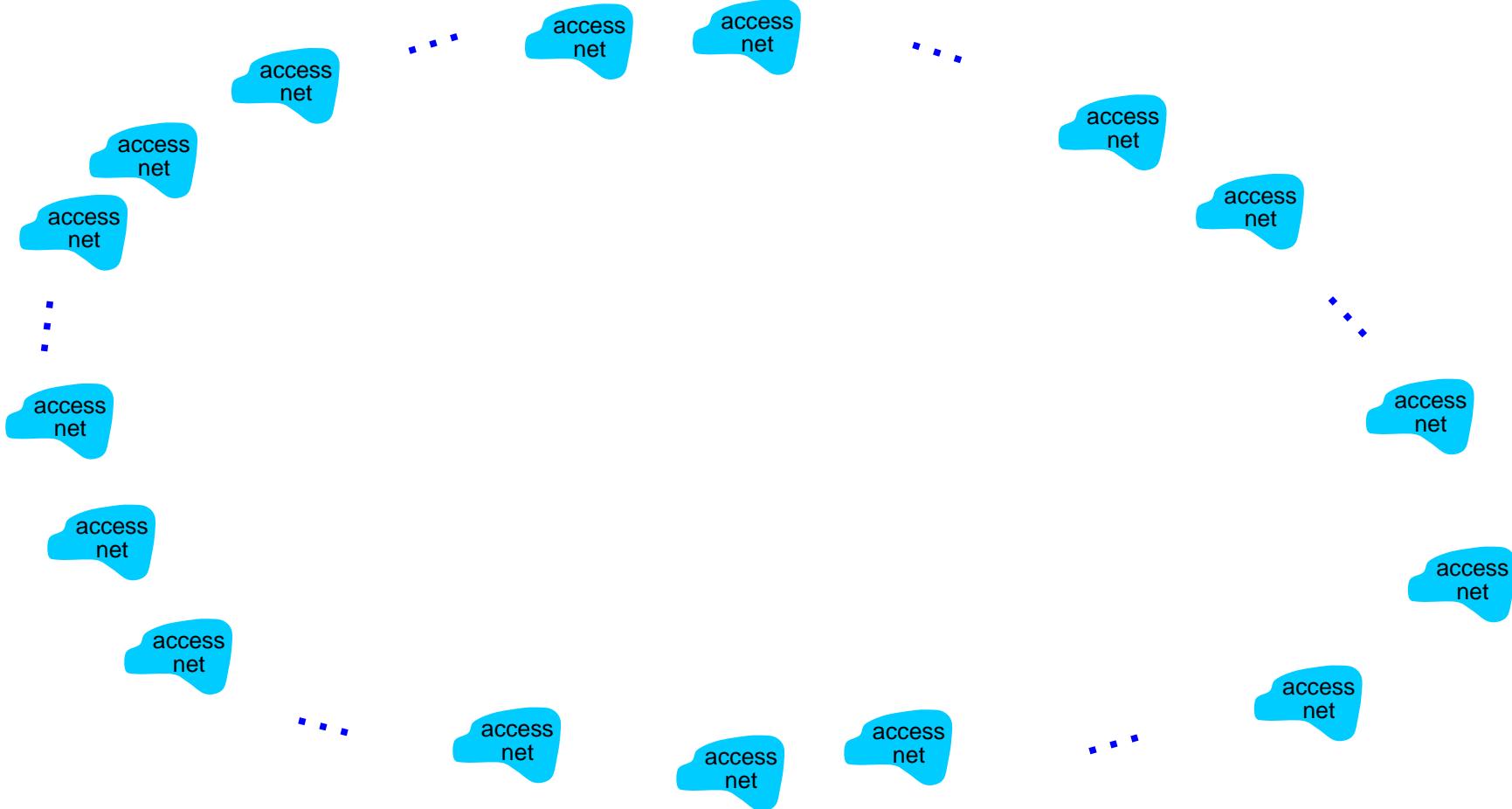
Q: human analogies of reserved resources (circuit switching) versus on-demand allocation (packet-switching)?

Internet structure: network of networks

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

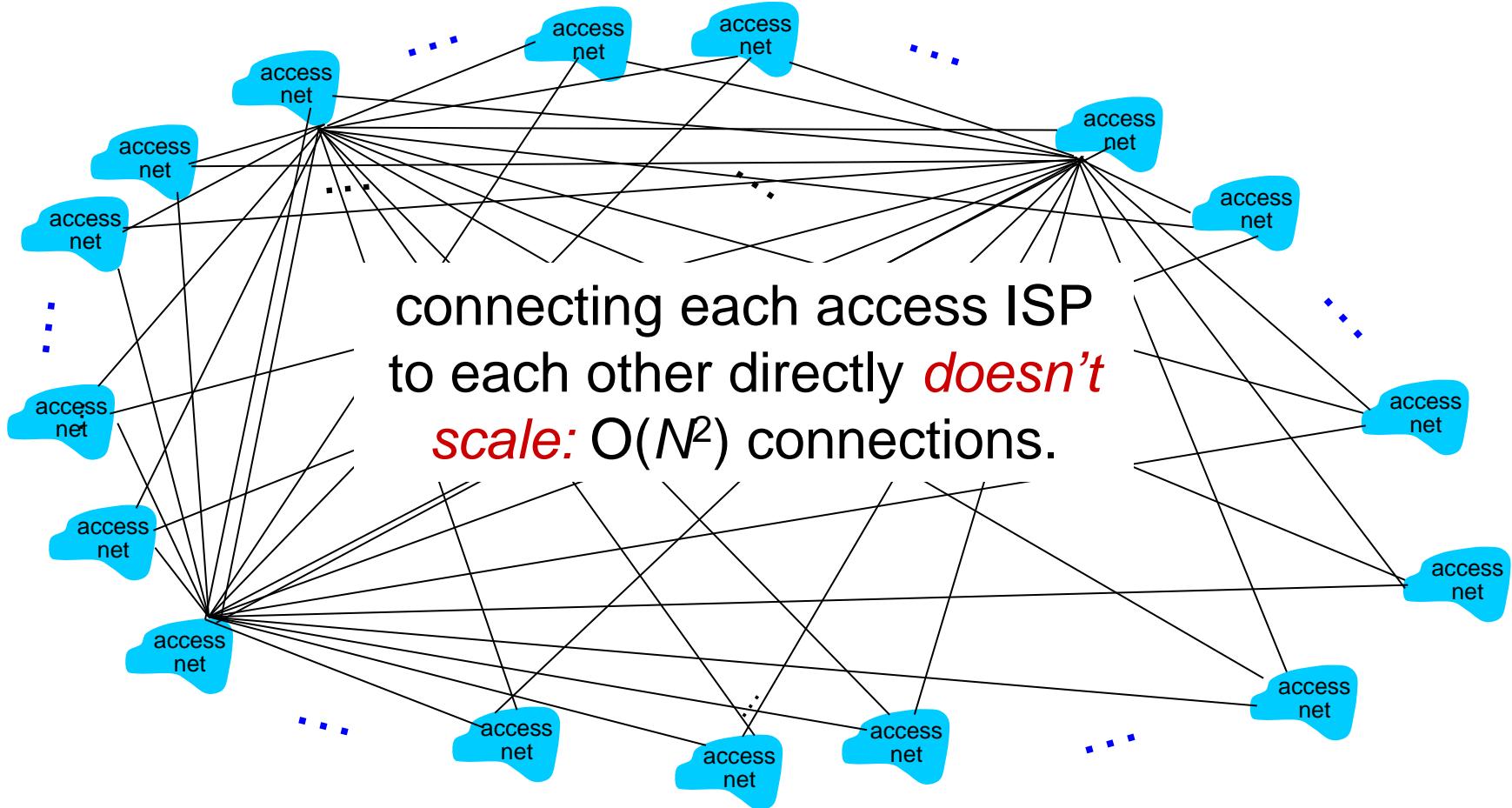
Internet structure: network of networks

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



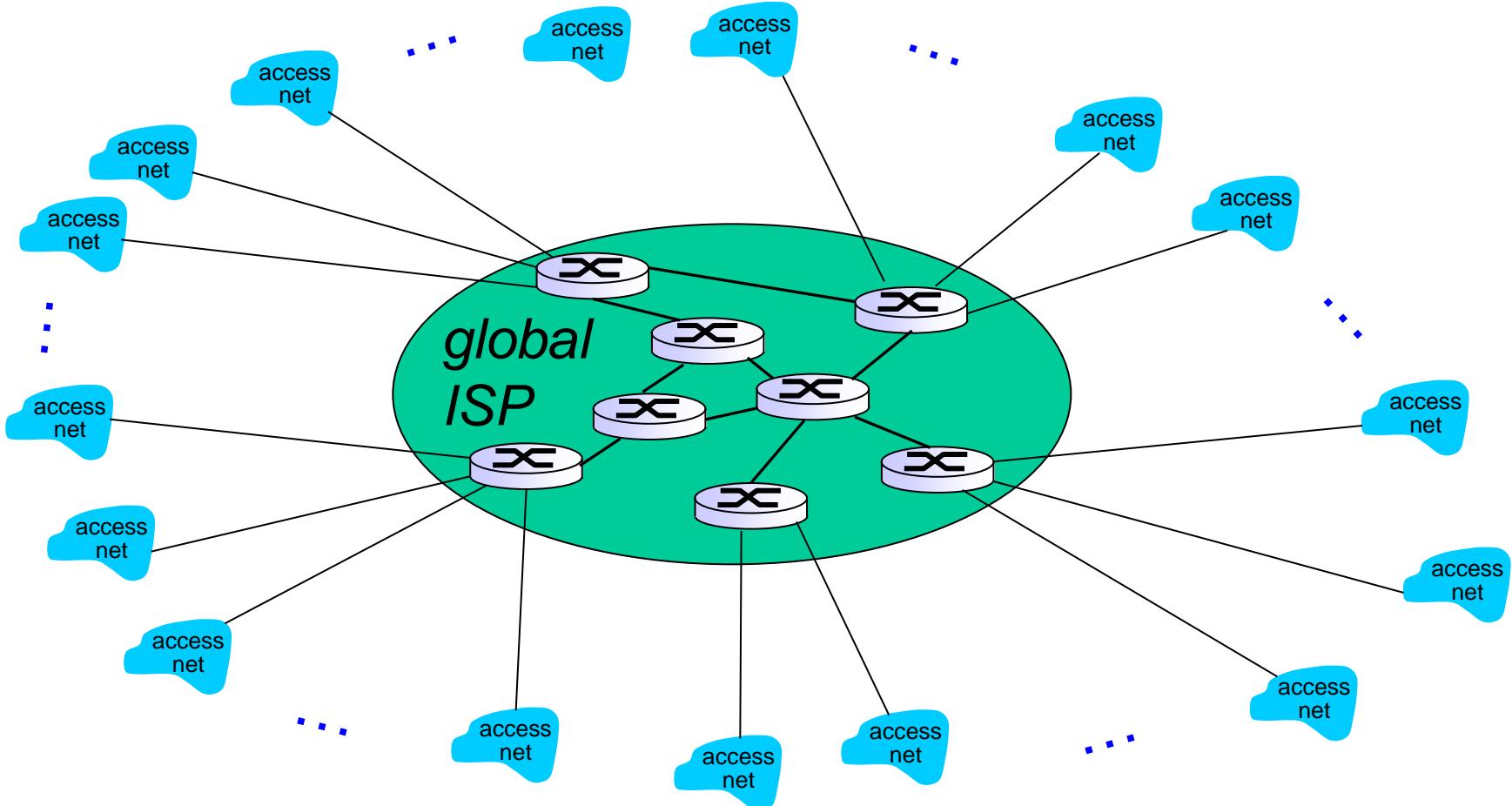
Internet structure: network of networks

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



Internet structure: network of networks

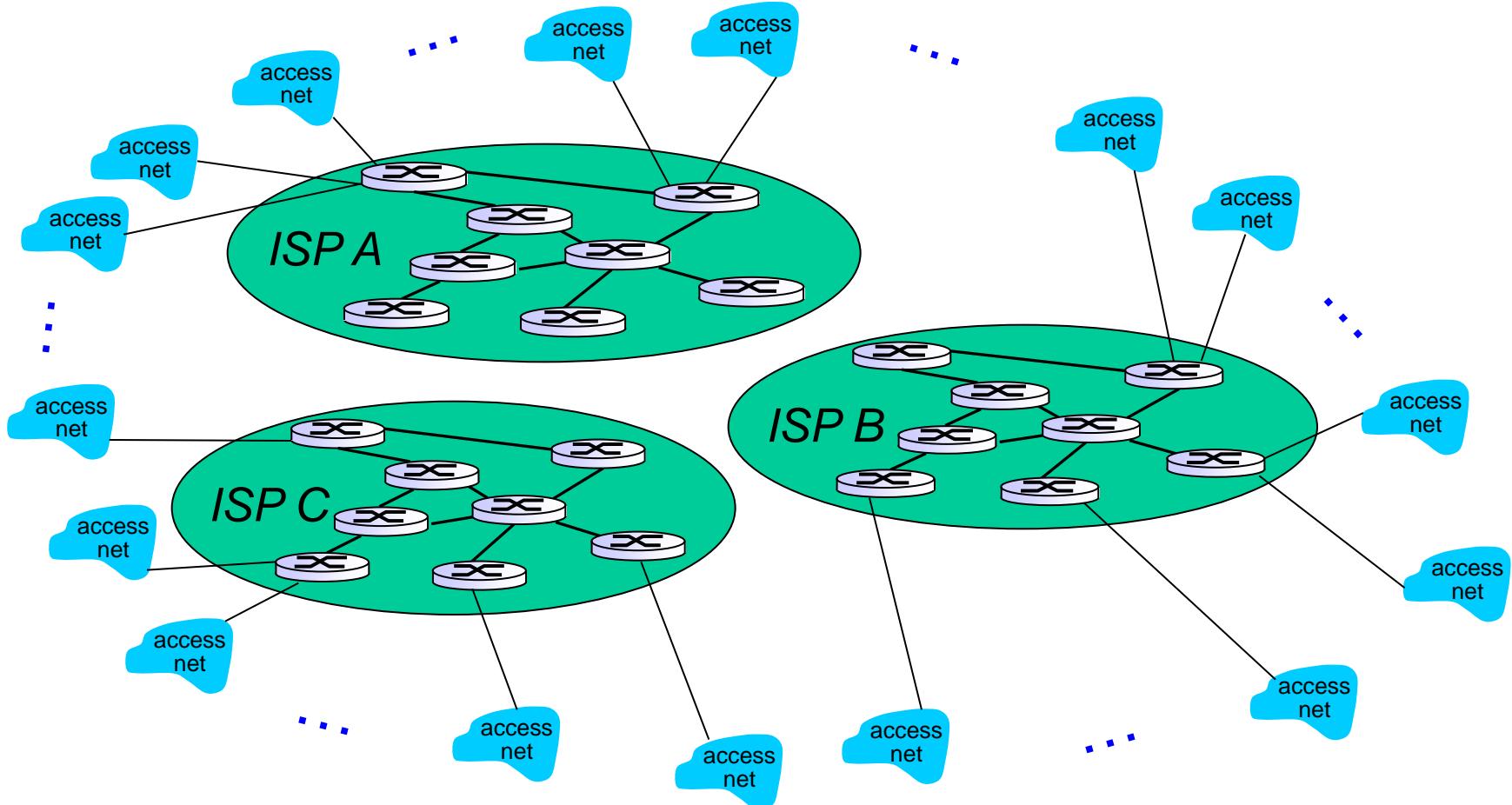
Option: connect each access ISP to a global transit ISP? *Customer and provider ISPs have economic agreement.*



Internet structure: network of networks

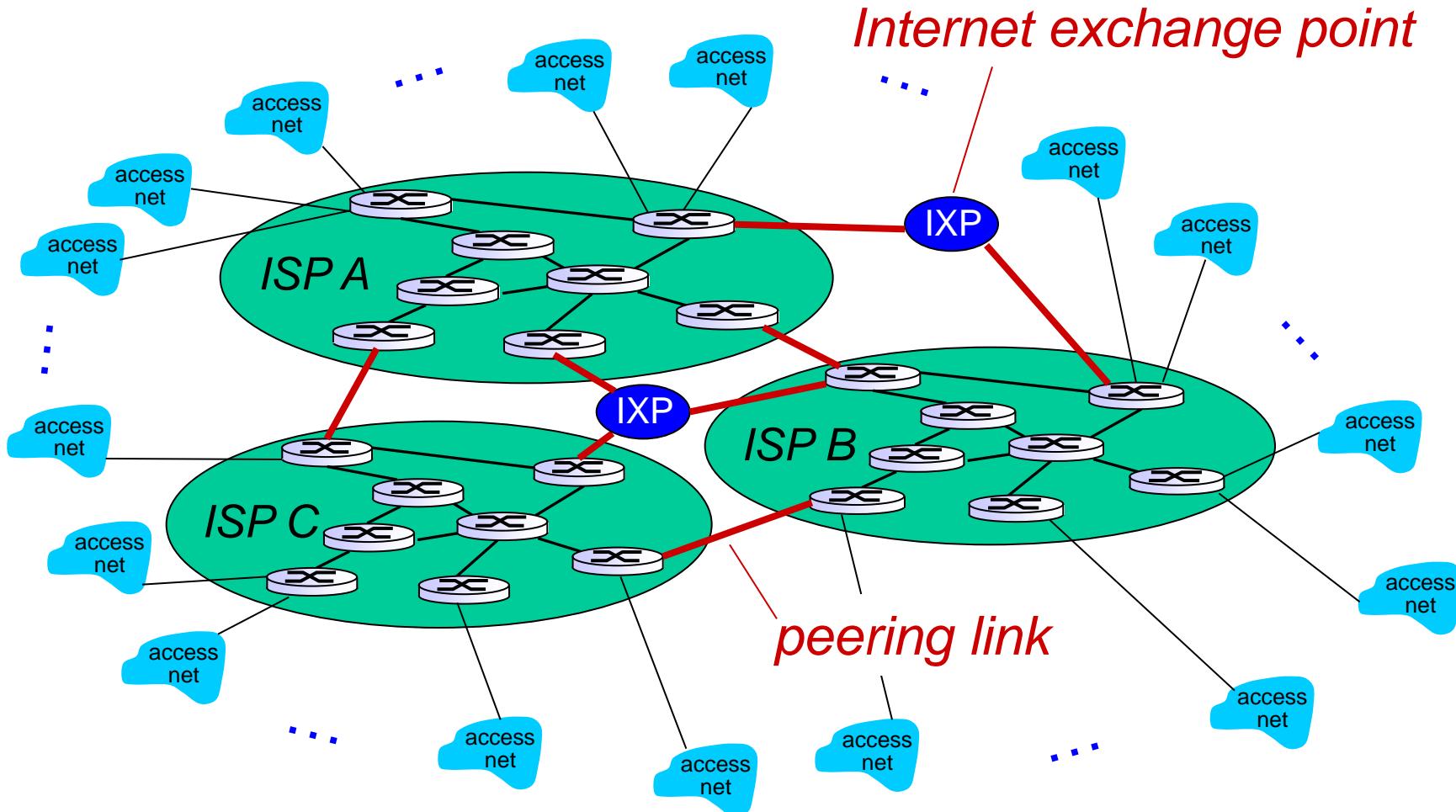
But if one global ISP is viable business, there will be competitors

....



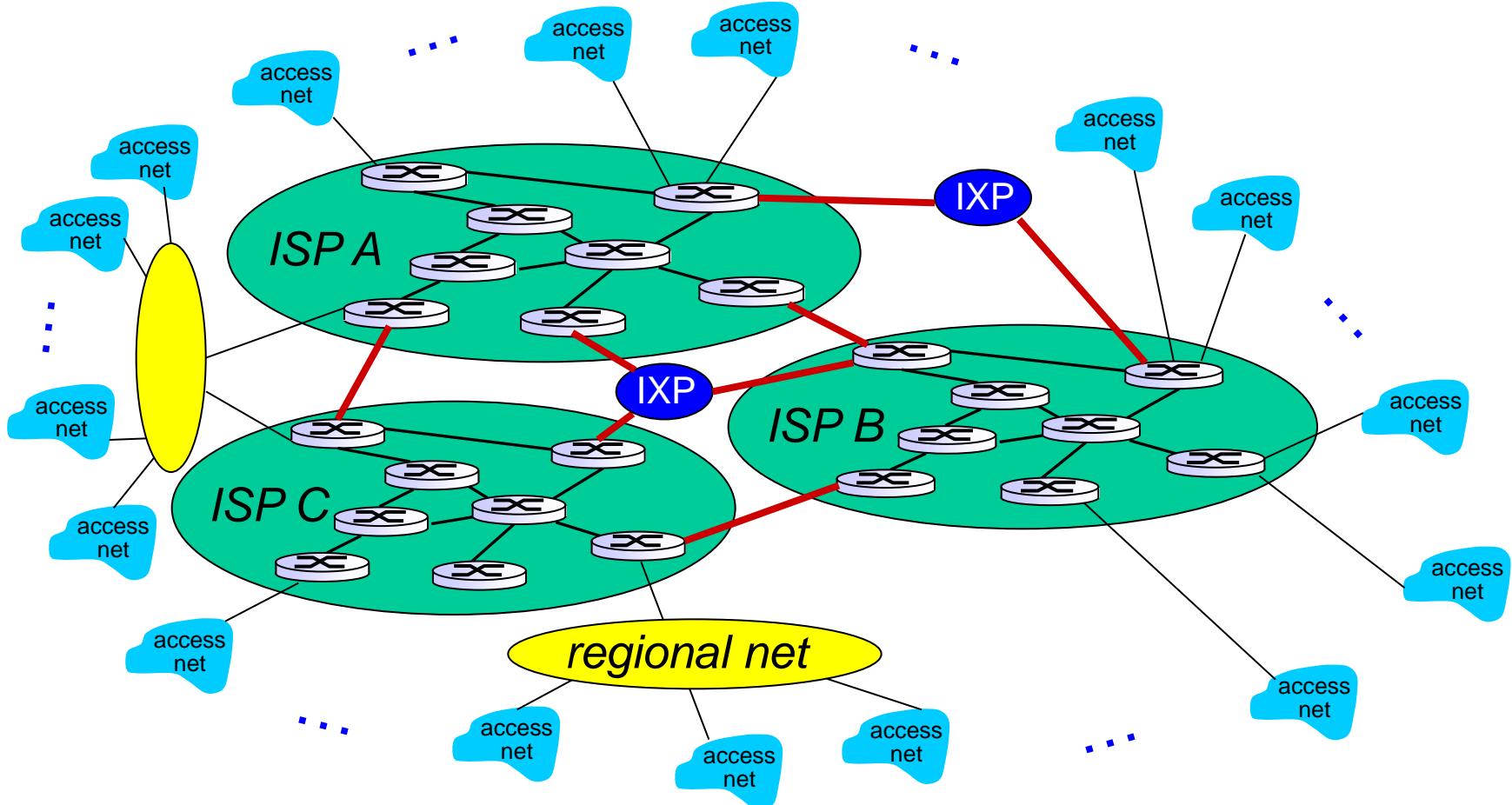
Internet structure: network of networks

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



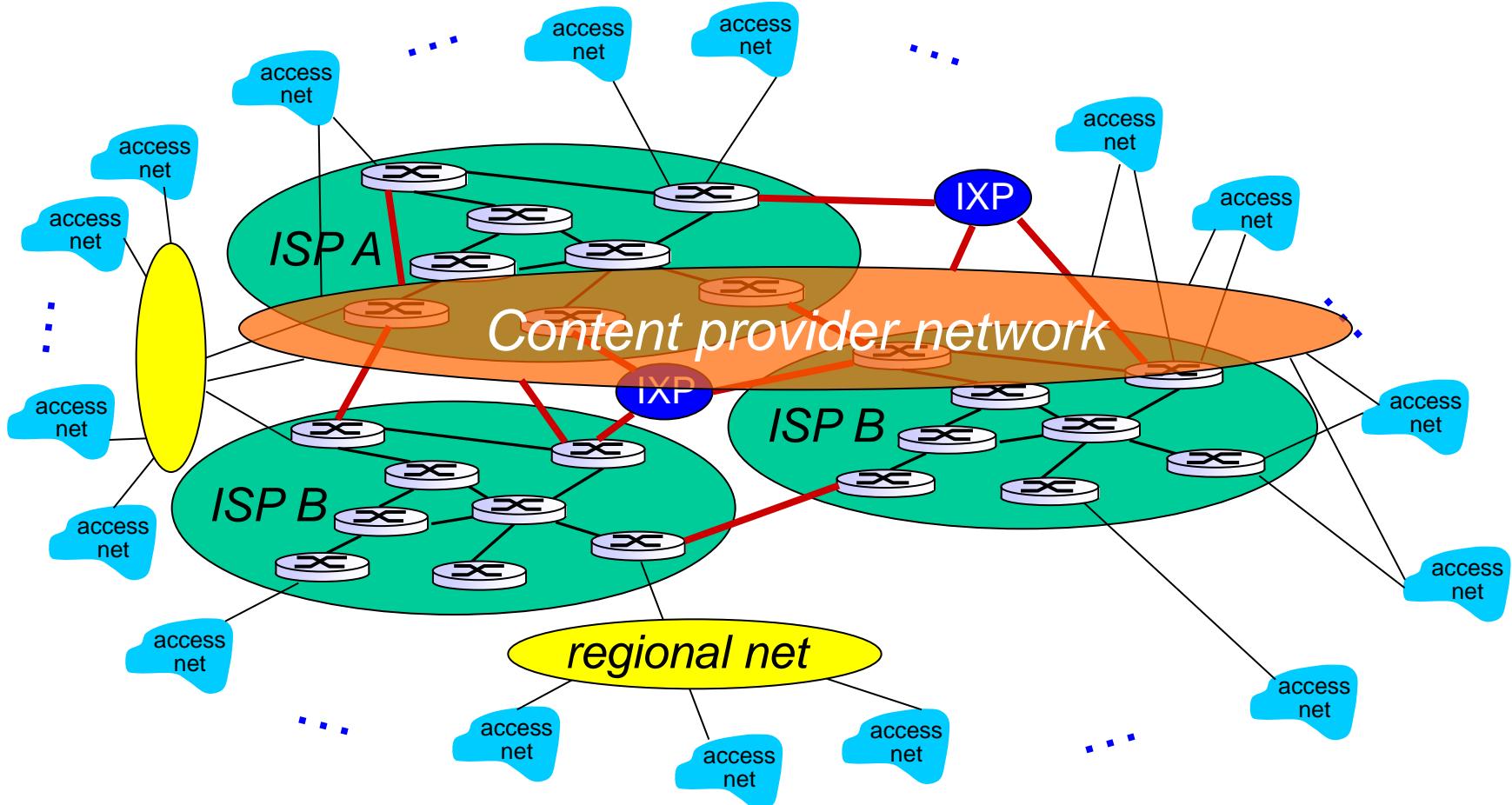
Internet structure: network of networks

... and regional networks may arise to connect access nets to ISPs

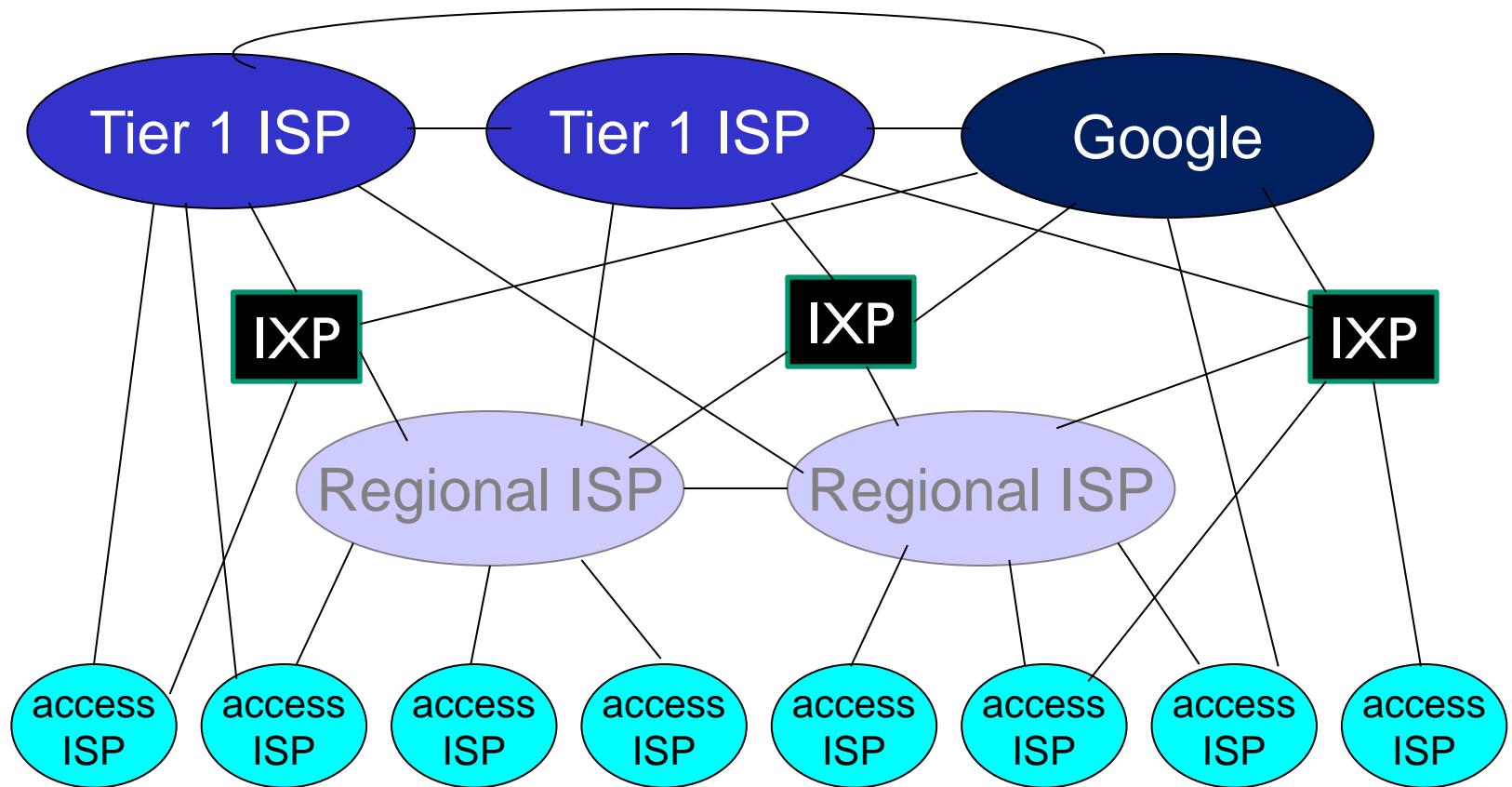


Internet structure: network of networks

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

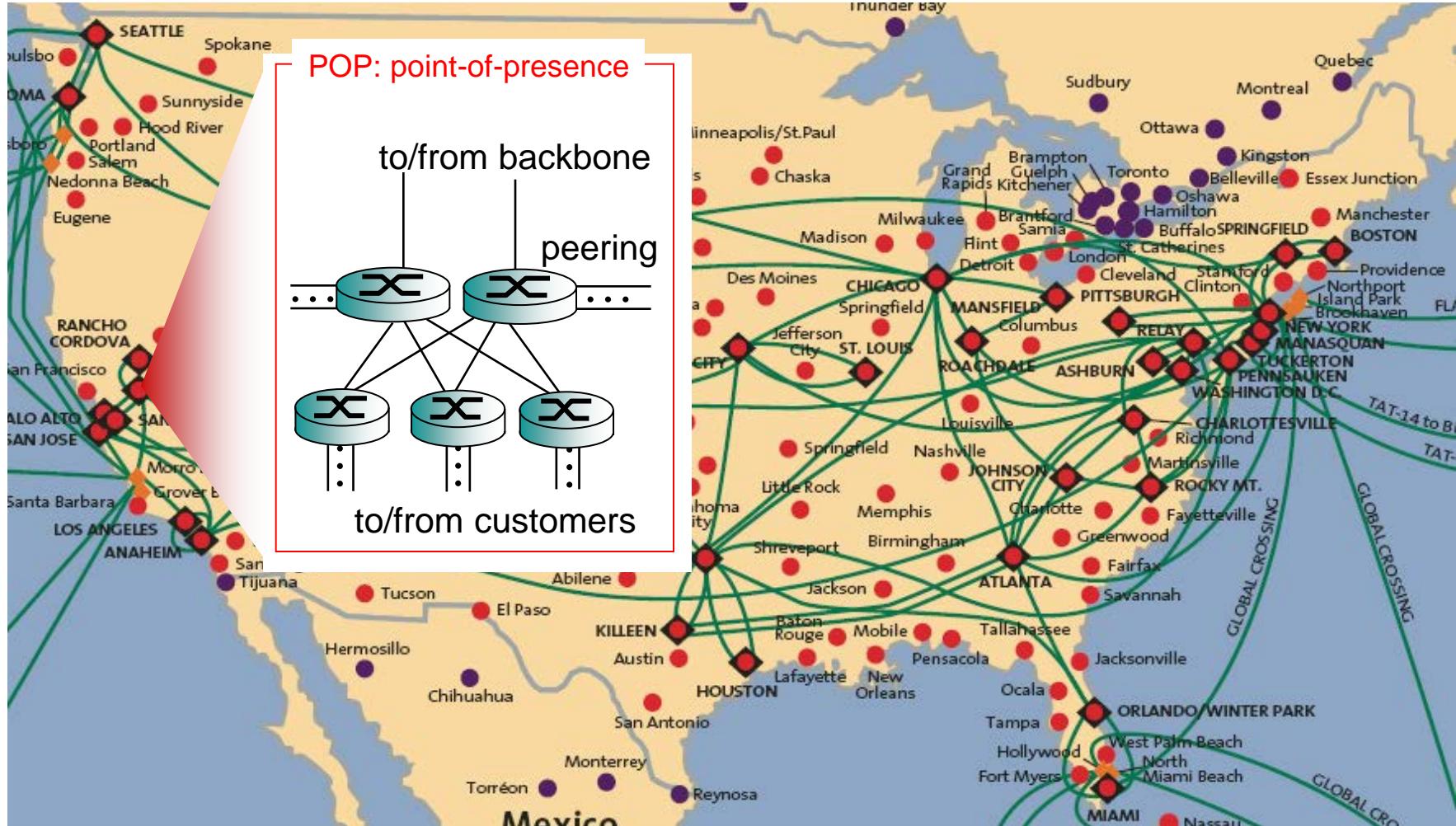


Internet structure: network of networks



- ❖ 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 its data centers to Internet, often bypassing tier-1, regional ISPs

Tier-1 ISP: e.g., Sprint



Chapter I: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

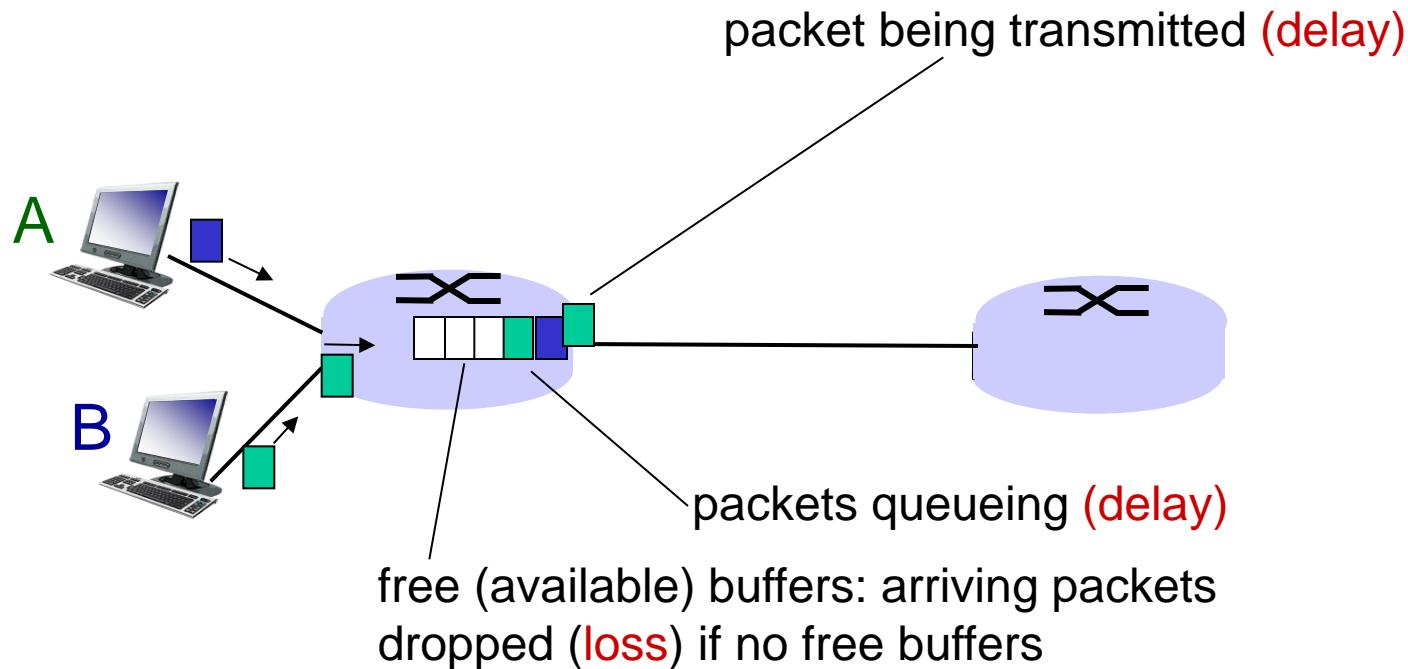
I.6 networks under attack: security

I.7 history

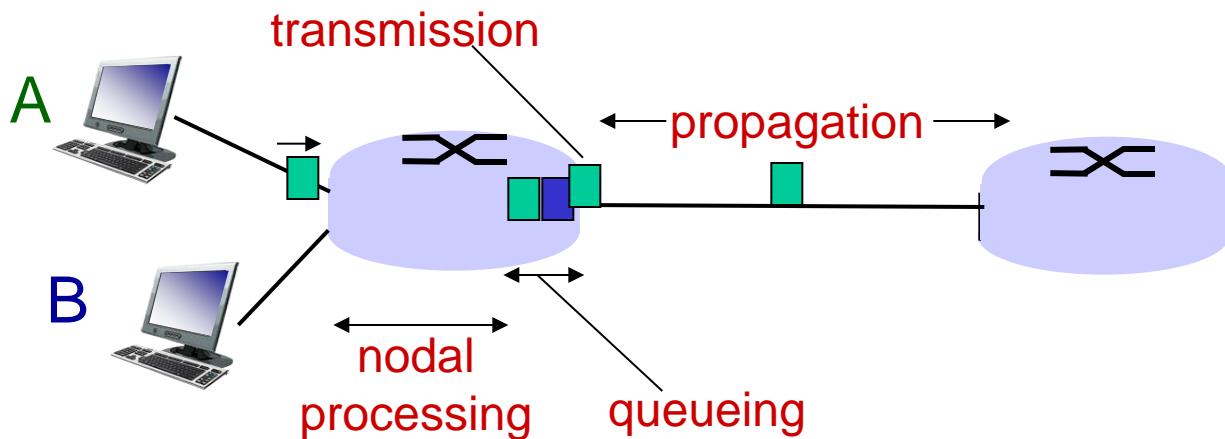
How do loss and delay occur?

packets queue in router buffers

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



Four sources of packet delay



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

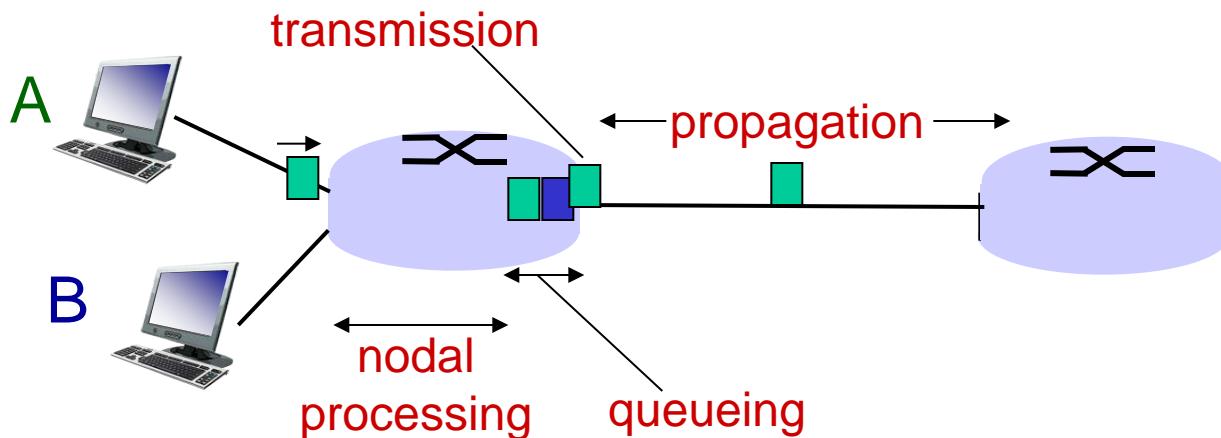
d_{proc} : nodal processing

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

d_{queue} : queueing delay

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

Four sources of packet delay



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

d_{trans} : transmission delay:

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

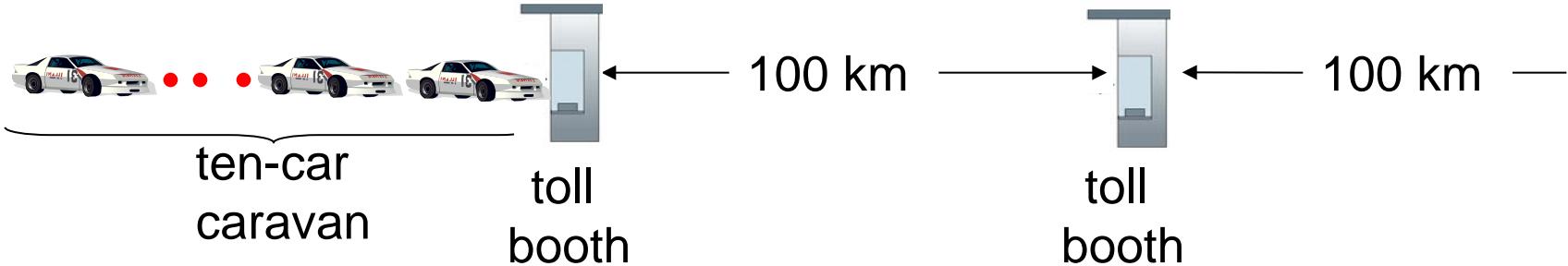
d_{trans} and d_{prop}
very different

d_{prop} : propagation delay:

- d : length of physical link
- s : propagation speed in medium ($\sim 2 \times 10^8 \text{ m/sec}$)
- $d_{\text{prop}} = d/s$

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

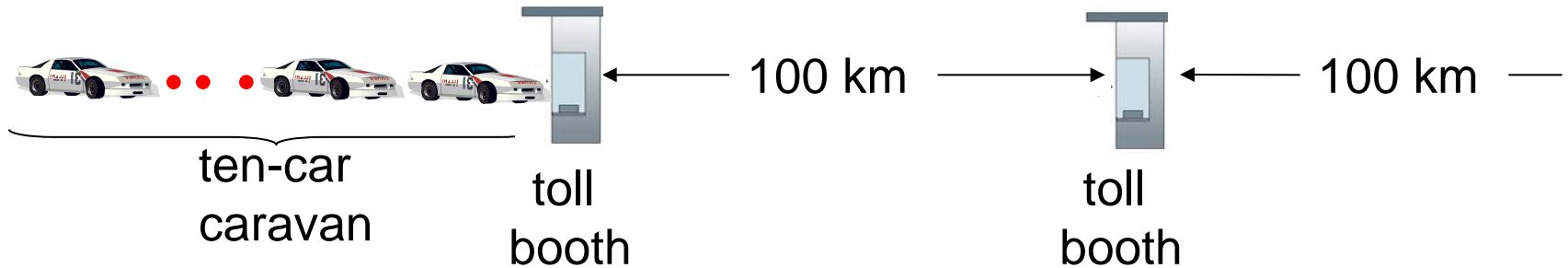
Caravan analogy



- ❖ 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:
 $100\text{km}/(100\text{km/hr}) = 1\text{ hr}$
- A: 62 minutes

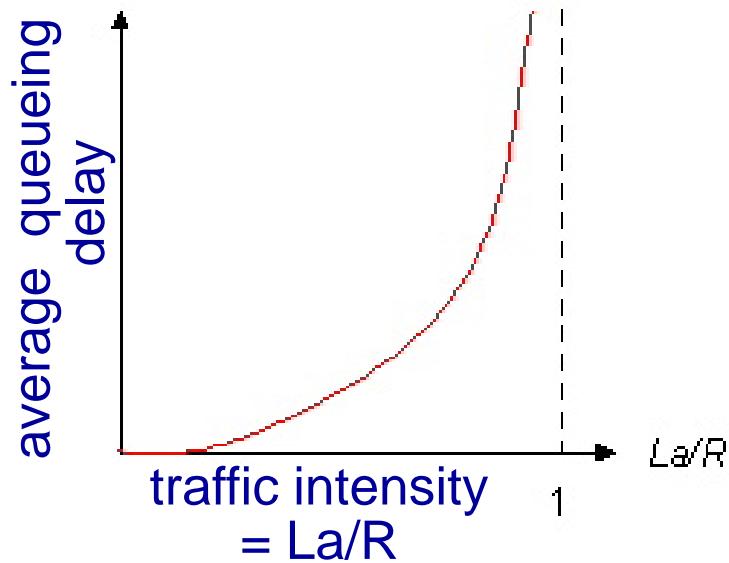
Caravan analogy (more)



- ❖ 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?
 - **A: Yes!** after 7 min, 1st car arrives at second booth; three cars still at 1st booth.

Queueing delay (revisited)

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



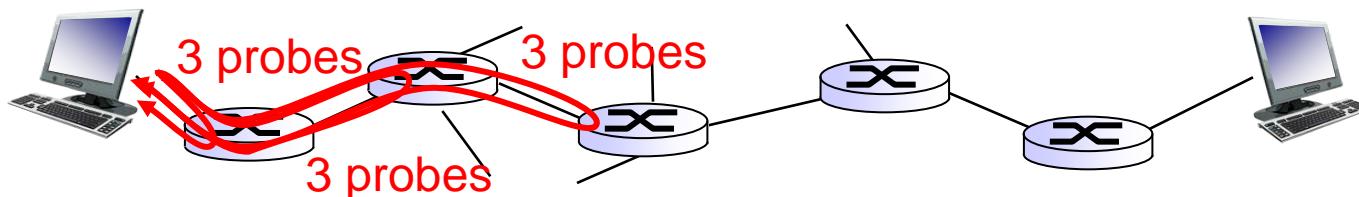
- ❖ $La/R \sim 0$: avg. queueing delay small
- ❖ $La/R \rightarrow 1$: avg. queueing delay large
- ❖ $La/R > 1$: more “work” arriving than can be serviced, average delay infinite!



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

“Real” Internet delays and routes

- ❖ 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.



“Real” Internet delays, routes

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

3 delay measurements from
gaia.cs.umass.edu to cs-gw.cs.umass.edu

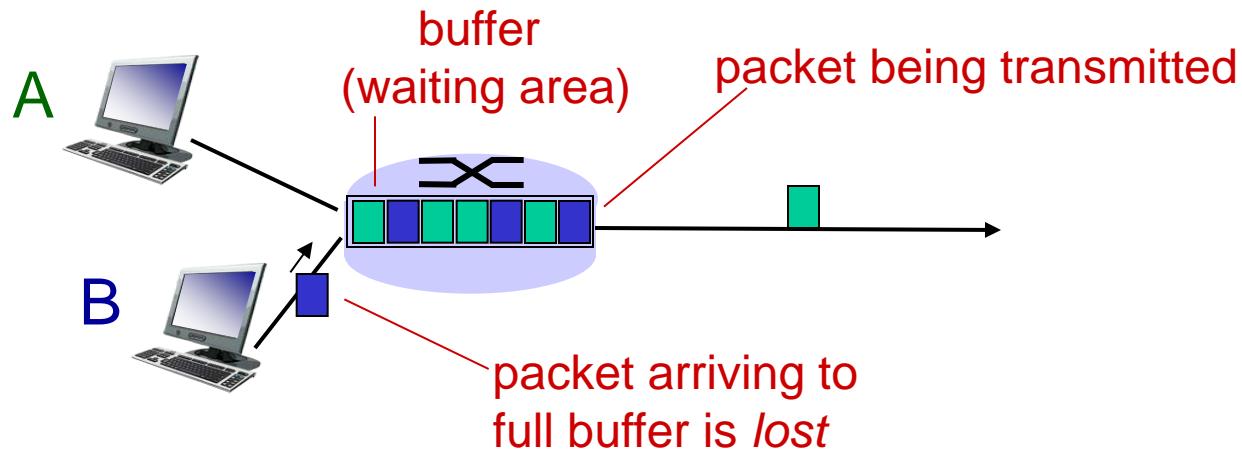
1	cs-gw (128.119.240.254)	1 ms	1 ms	2 ms
2	border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145)	1 ms	1 ms	2 ms
3	cht-vbns.gw.umass.edu (128.119.3.130)	6 ms	5 ms	5 ms
4	jn1-at1-0-0-19.wor.vbns.net (204.147.132.129)	16 ms	11 ms	13 ms
5	jn1-so7-0-0-0.wae.vbns.net (204.147.136.136)	21 ms	18 ms	18 ms
6	abilene-vbns.abilene.ucaid.edu (198.32.11.9)	22 ms	18 ms	22 ms
7	nycm-wash.abilene.ucaid.edu (198.32.8.46)	22 ms	22 ms	22 ms
8	62.40.103.253 (62.40.103.253)	104 ms	109 ms	106 ms
9	de2-1.de1.de.geant.net (62.40.96.129)	109 ms	102 ms	104 ms
10	de.fr1.fr.geant.net (62.40.96.50)	113 ms	121 ms	114 ms
11	renater-gw.fr1.fr.geant.net (62.40.103.54)	112 ms	114 ms	112 ms
12	nio-n2.cssi.renater.fr (193.51.206.13)	111 ms	114 ms	116 ms
13	nice.cssi.renater.fr (195.220.98.102)	123 ms	125 ms	124 ms
14	r3t2-nice.cssi.renater.fr (195.220.98.110)	126 ms	126 ms	124 ms
15	eurecom-valbonne.r3t2.ft.net (193.48.50.54)	135 ms	128 ms	133 ms
16	194.214.211.25 (194.214.211.25)	126 ms	128 ms	126 ms
17	***			
18	***	* means no response (probe lost, router not replying)		
19	fantasia.eurecom.fr (193.55.113.142)	132 ms	128 ms	136 ms

trans-oceanic link

* Do some traceroutes from exotic countries at www.traceroute.org

Packet loss

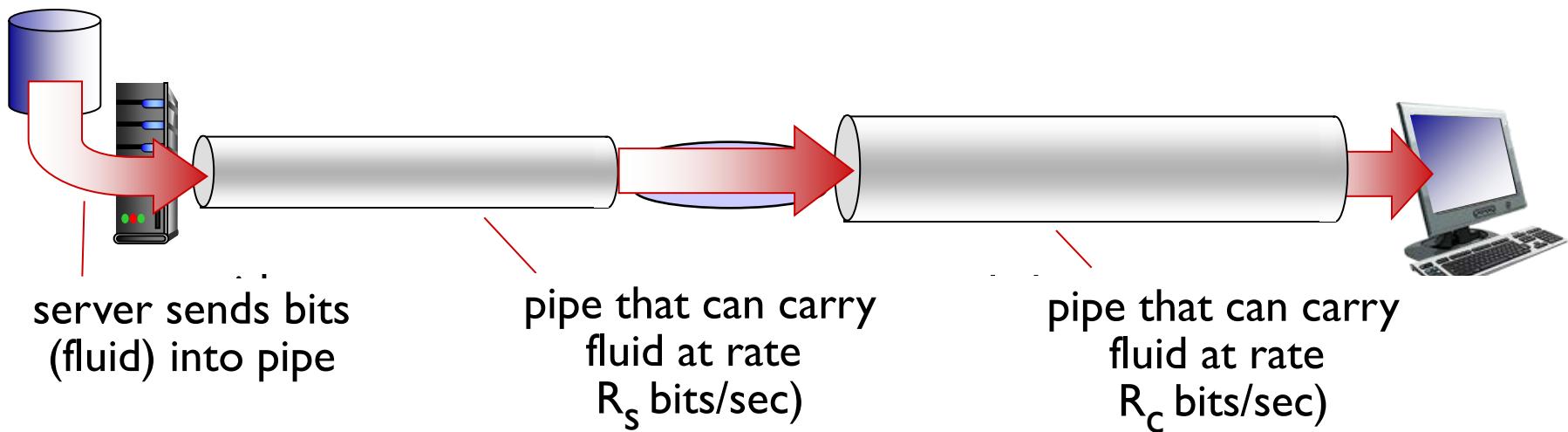
- ❖ queue (aka buffer) preceding link in buffer has finite capacity
- ❖ packet arriving to full queue dropped (aka lost)
- ❖ lost packet may be retransmitted by previous node, by source end system, or not at all



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

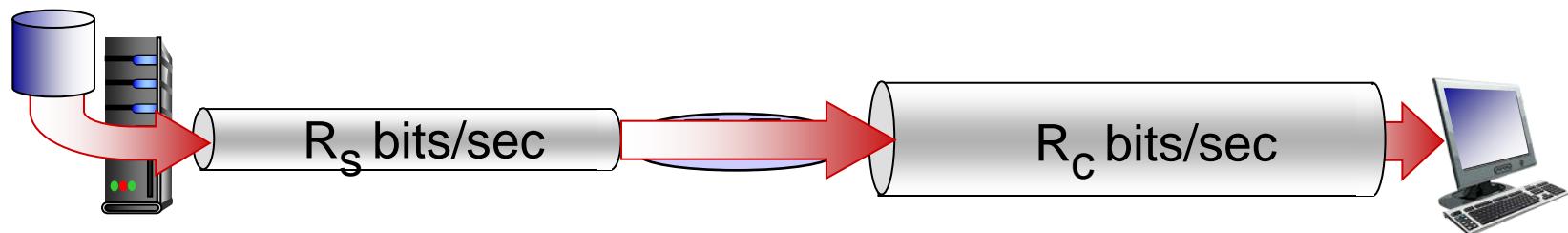
Throughput

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

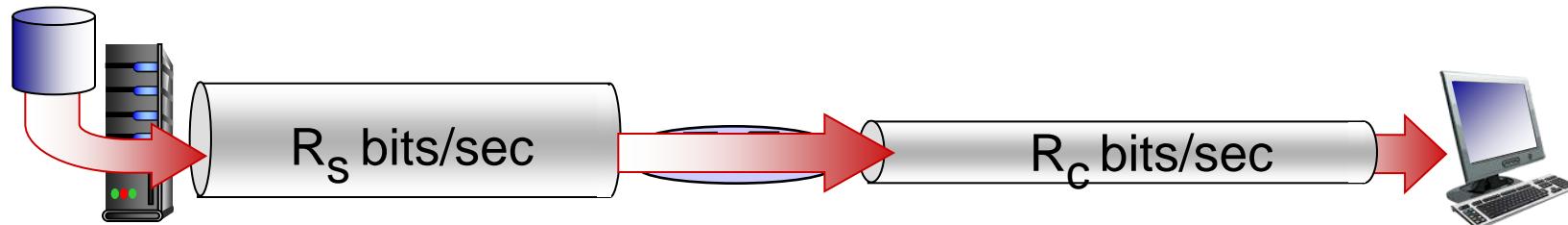


Throughput (more)

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



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

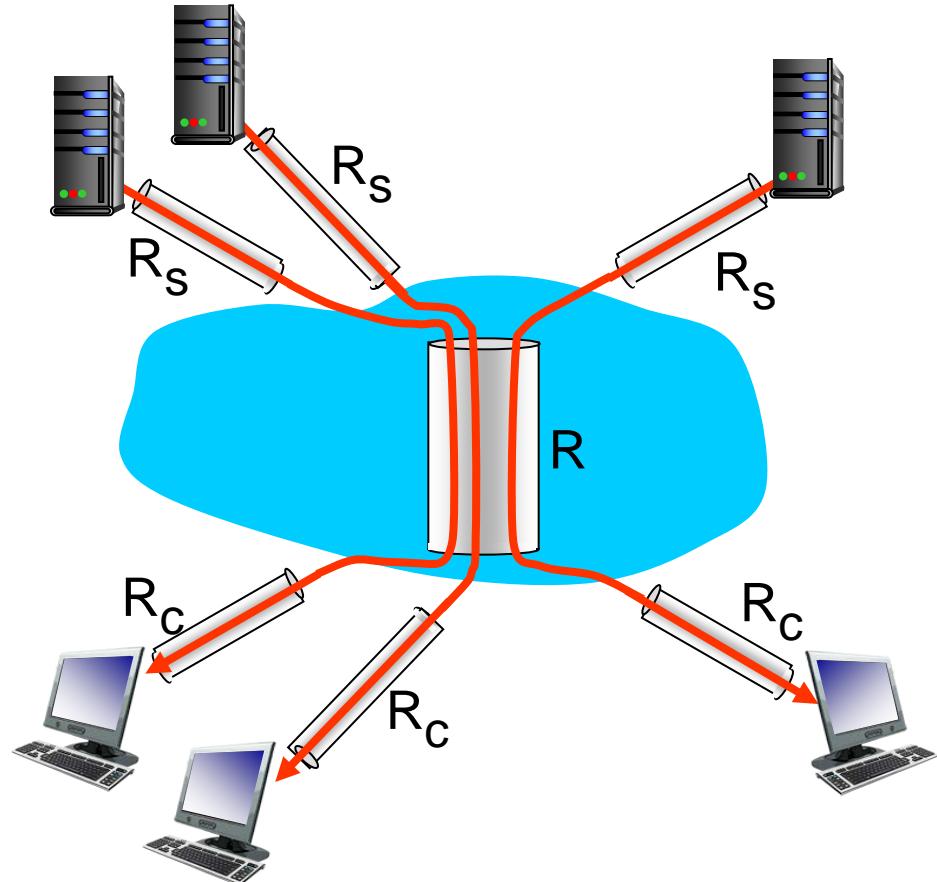


bottleneck link

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

Throughput: Internet scenario

- ❖ per-connection end-end throughput: $\min(R_c, R_s, R/10)$
- ❖ in practice: R_c or R_s is often bottleneck



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

Chapter I: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

I.6 networks under attack: security

I.7 history

Protocol “layers”

*Networks are complex,
with many “pieces”:*

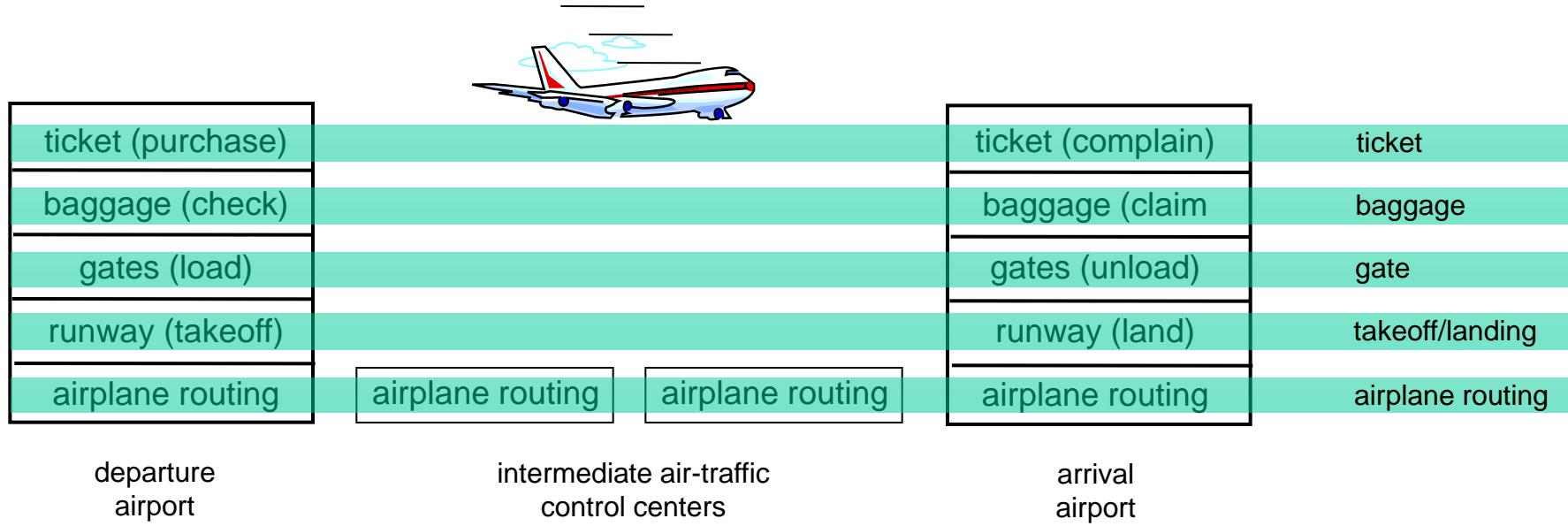
- hosts
- routers
- links of various media
- applications
- protocols
- hardware, software

Question:

is there any hope of
organizing structure of
network?

.... or at least our
discussion of networks?

Layering of airline functionality



layers: each layer implements a service

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

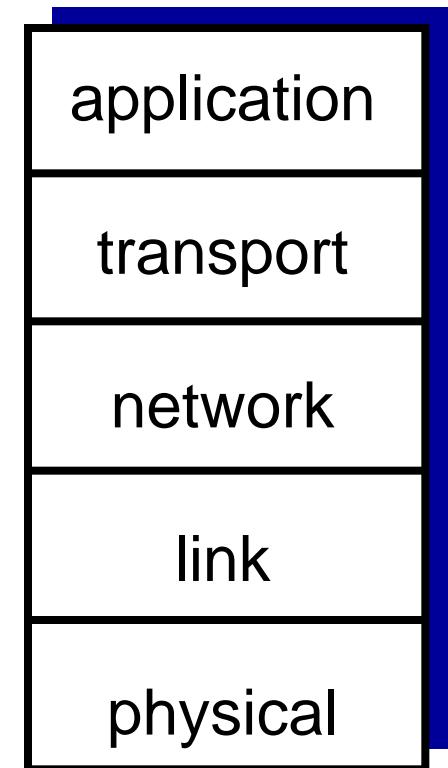
Why layering?

dealing with complex systems:

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

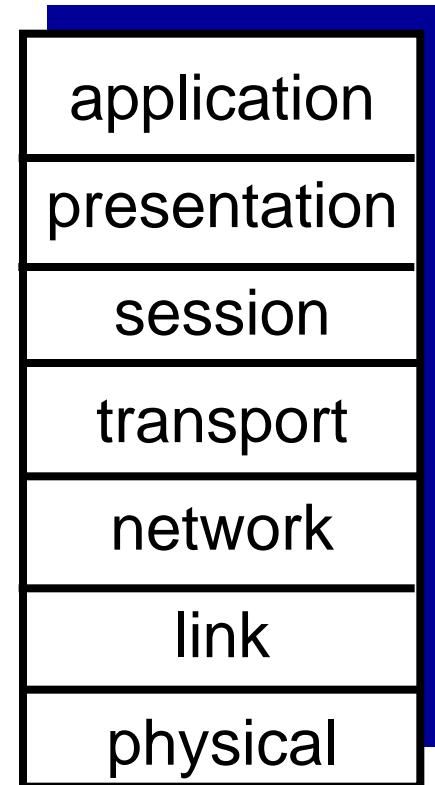
Internet protocol stack

- ❖ *application*: supporting network applications
 - FTP, SMTP, HTTP
- ❖ *transport*: process-process data transfer
 - TCP, UDP
- ❖ *network*: routing of datagrams from source to destination
 - IP, routing protocols
- ❖ *link*: data transfer between neighboring network elements
 - Ethernet, 802.11 (WiFi), PPP
- ❖ *physical*: bits “on the wire”

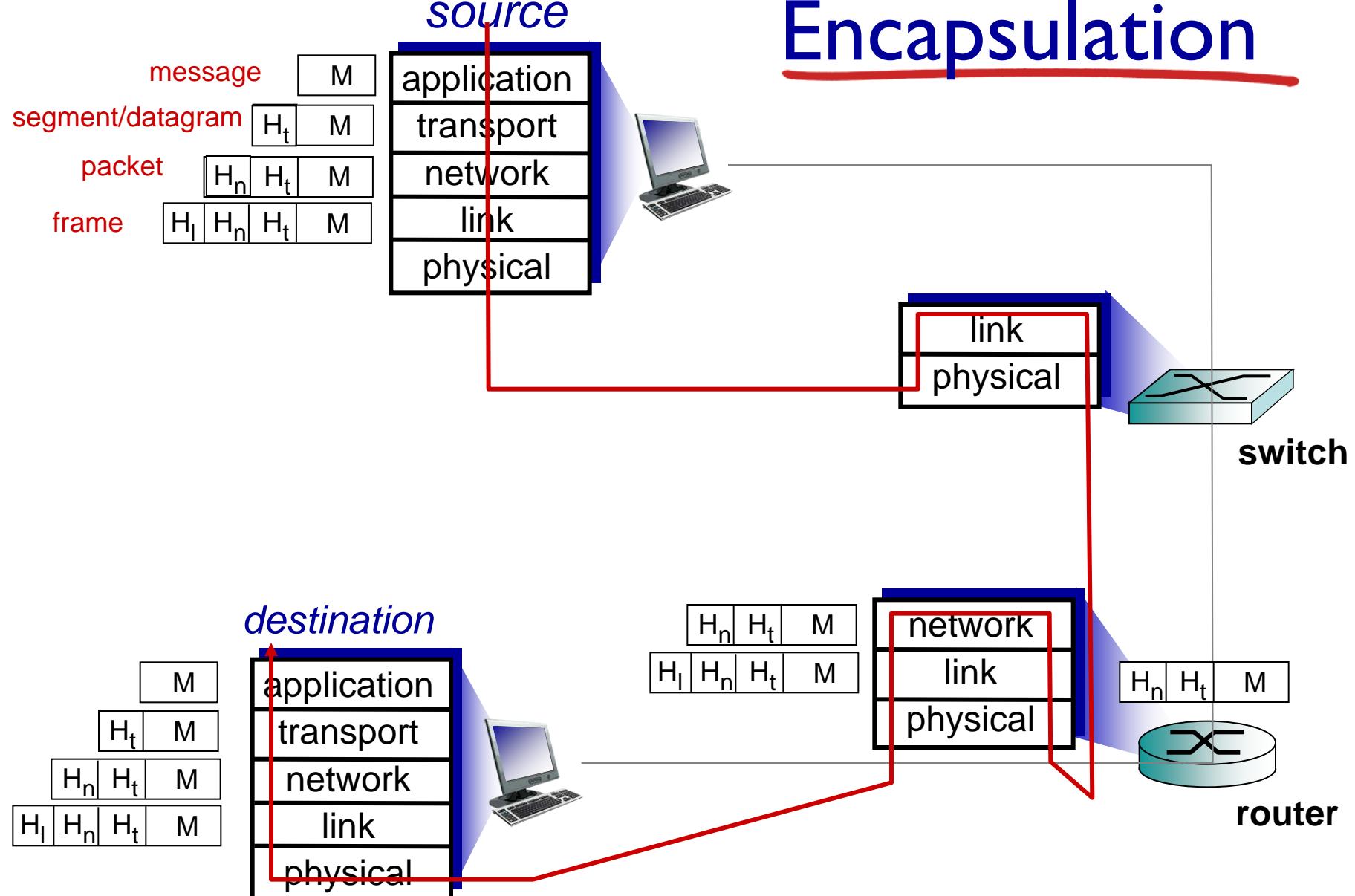


ISO/OSI reference model

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



Encapsulation



Chapter I: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

I.6 networks under attack: security

I.7 history

Network security

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

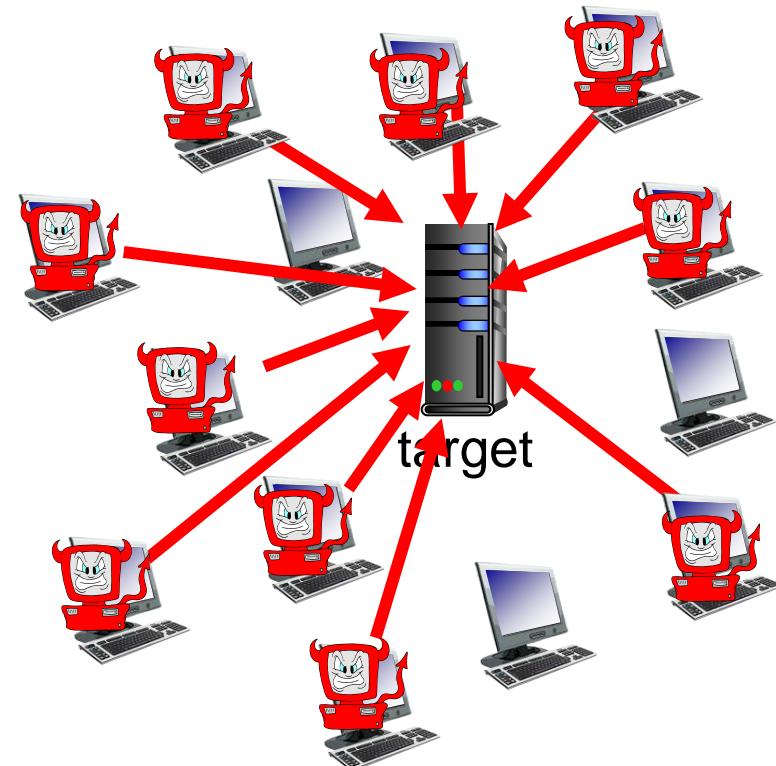
Bad guys: put malware into hosts via Internet

- ❖ malware can get in host from:
 - **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
- ❖ **spyware malware** can record keystrokes, web sites visited, upload info to collection site
- ❖ infected host can be enrolled in **botnet**, used for spam. DDoS attacks

Bad guys: attack server, network infrastructure

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

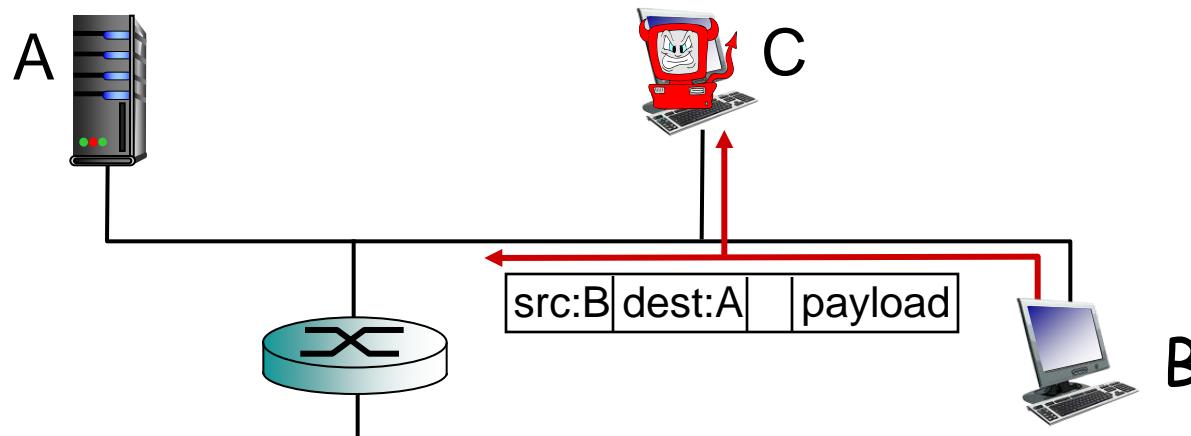
1. select target
2. break into hosts around the network (see botnet)
3. send packets to target from compromised hosts



Bad guys can sniff packets

packet “sniffing”:

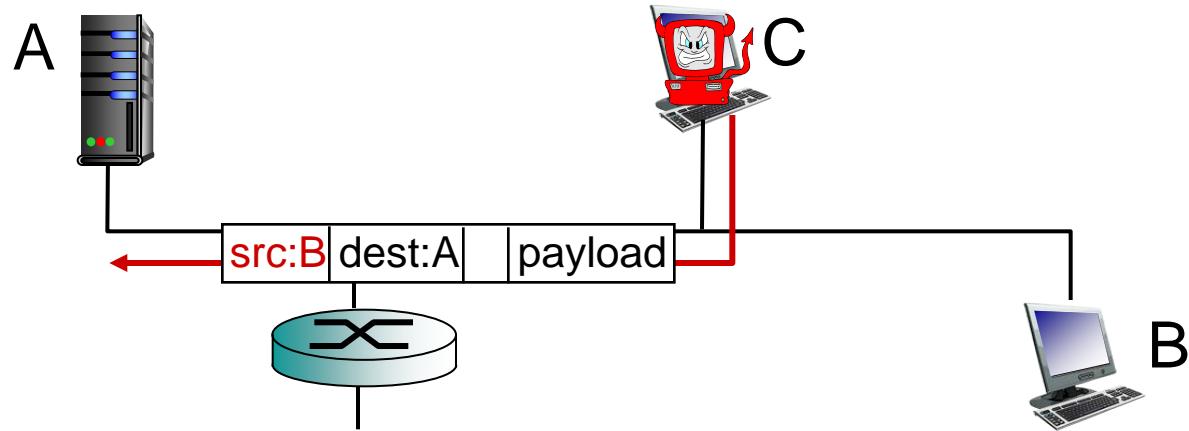
- broadcast media (shared ethernet, wireless)
- promiscuous network interface reads/records all packets (e.g., including passwords!) passing by



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

Bad guys can use fake addresses

IP spoofing: send packet with false source address



... lots more on security (throughout, Chapter 8)

Chapter I: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

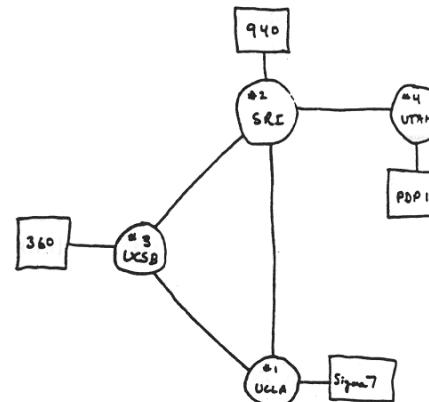
I.6 networks under attack: security

I.7 history

Internet history

1961-1972: Early packet-switching principles

- ❖ 1961: Kleinrock - queueing theory shows effectiveness of packet-switching
- ❖ 1964: Baran - packet-switching in military nets
- ❖ 1967: ARPAnet conceived by Advanced Research Projects Agency
- ❖ 1969: first ARPAnet node operational
- ❖ 1972:
 - ARPAnet public demo
 - NCP (Network Control Protocol) first host-host protocol
 - first e-mail program
 - ARPAnet has 15 nodes



Internet history

1972-1980: Internetworking, new and proprietary nets

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

Cerf and Kahn's
internetworking principles:

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

define today's Internet architecture

Internet history

1980-1990: new protocols, a proliferation of networks

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

Internet history

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

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

late 1990's – 2000's:

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

Internet history

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, email, etc.
- ❖ E-commerce, universities, enterprises running their services in “cloud” (eg, Amazon EC2)

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

© *The materials in these notes are adapted from Computer Networking: A Top Down Approach, 6th edition, by Jim Kurose, Keith Ross*

Chapter 2: outline

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

Chapter 2: application layer

our goals:

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

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
- ❖ ...
- ❖ ...

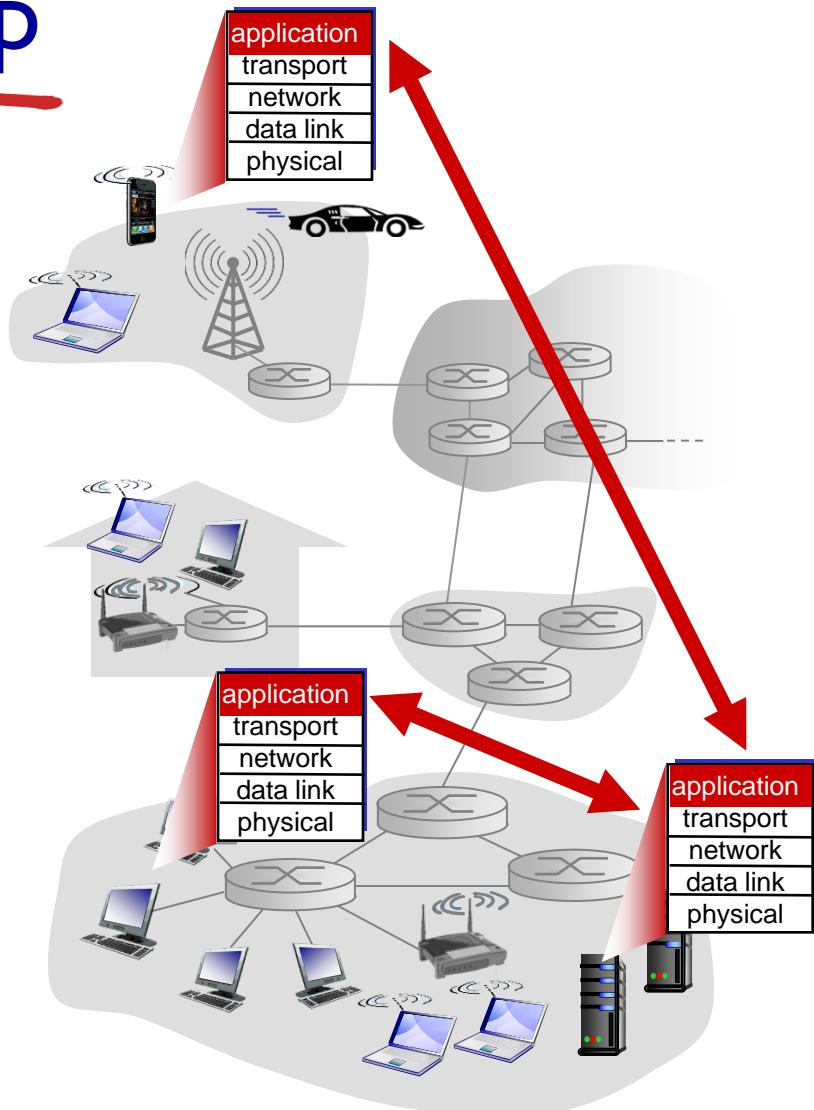
Creating a network app

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

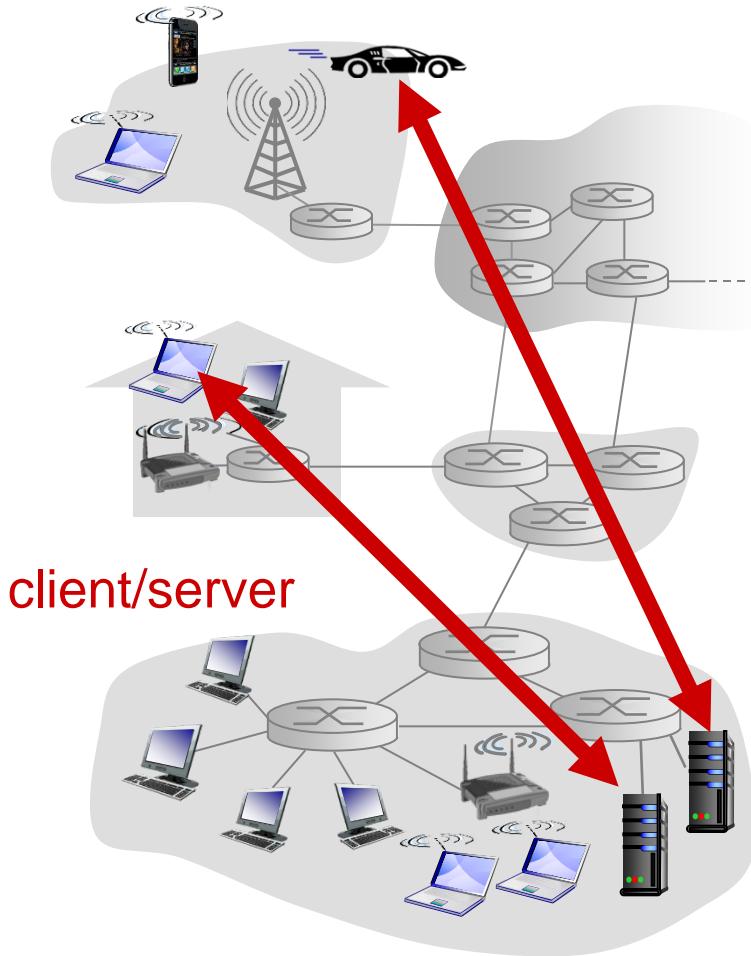


Application architectures

possible structure of applications:

- ❖ client-server
- ❖ peer-to-peer (P2P)

Client-server architecture



server:

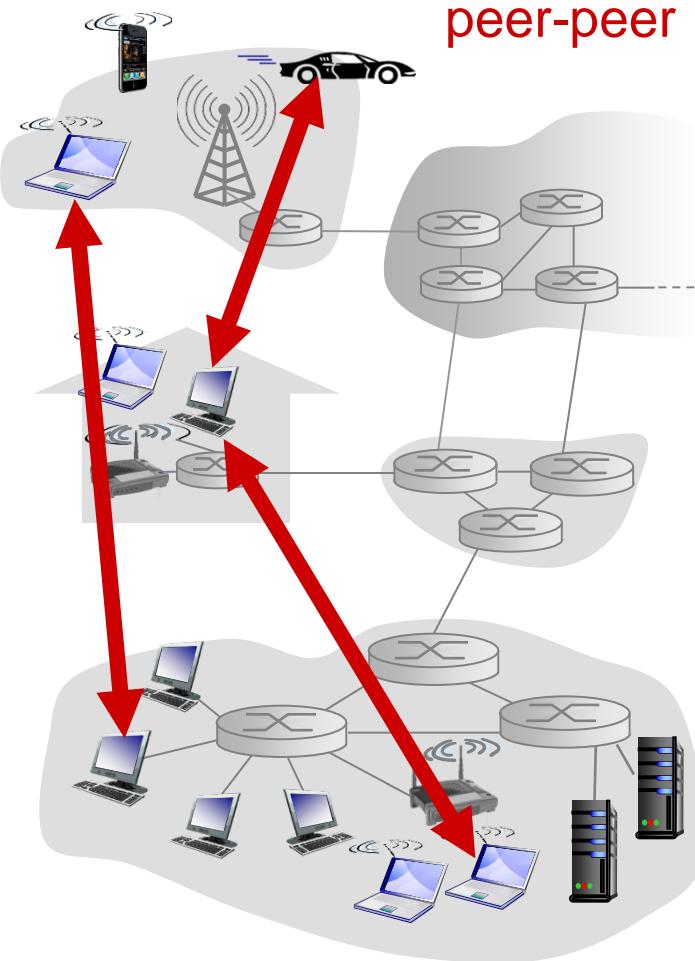
- ❖ always-on host
- ❖ permanent IP address
- ❖ data centers for scaling

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



Processes communicating

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

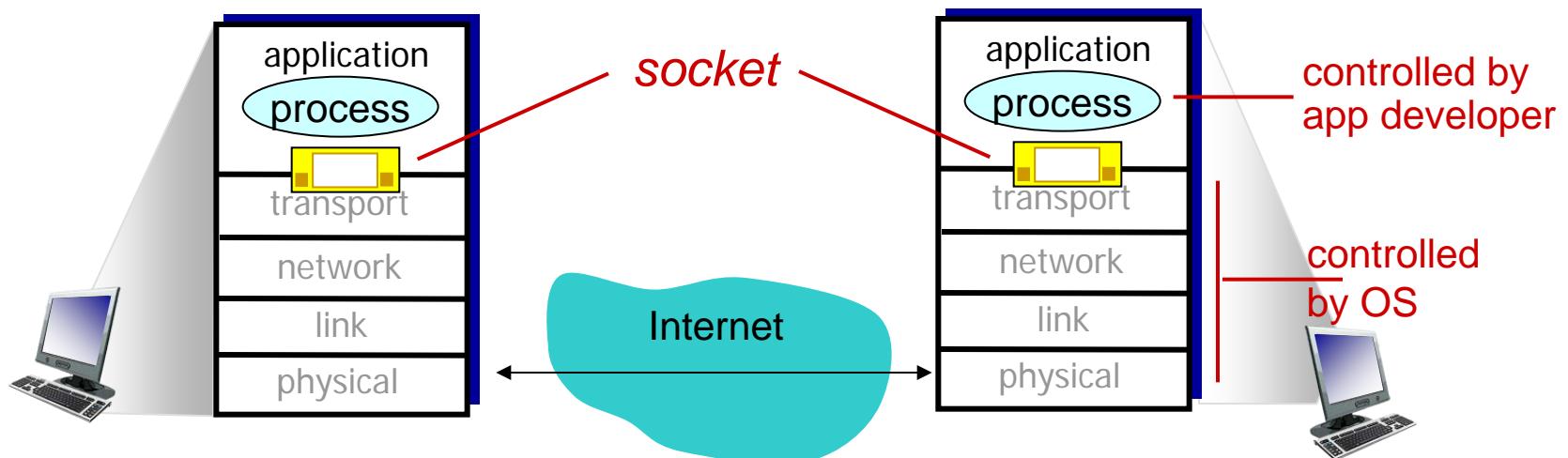
client process: process that initiates communication

server process: process that waits to be contacted

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

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



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

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

What transport service does an app need?

data integrity

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

timing

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

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

security

- ❖ encryption, data integrity,

...

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 video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	Elastic	yes and no

Internet transport protocols services

TCP service:

- ❖ *reliable transport* between sending and receiving process
- ❖ *flow control*: sender won't overwhelm receiver
- ❖ *congestion control*: throttle sender when network overloaded
- ❖ *does not provide*: timing, minimum throughput guarantee, security
- ❖ *connection-oriented*: setup required between client and server processes

UDP service:

- ❖ *unreliable data transfer* between sending and receiving process
- ❖ *does not provide*: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

Q: why bother? Why is there a UDP?

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

Securing TCP

TCP & UDP

- ❖ no encryption
- ❖ clear-text passwds sent into socket traverse Internet in clear text

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

- ❖ clear-text passwds sent into socket traverse Internet encrypted
- ❖ See Chapter 8

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

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

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.,

www.someschool.edu/someDept/pic.gif

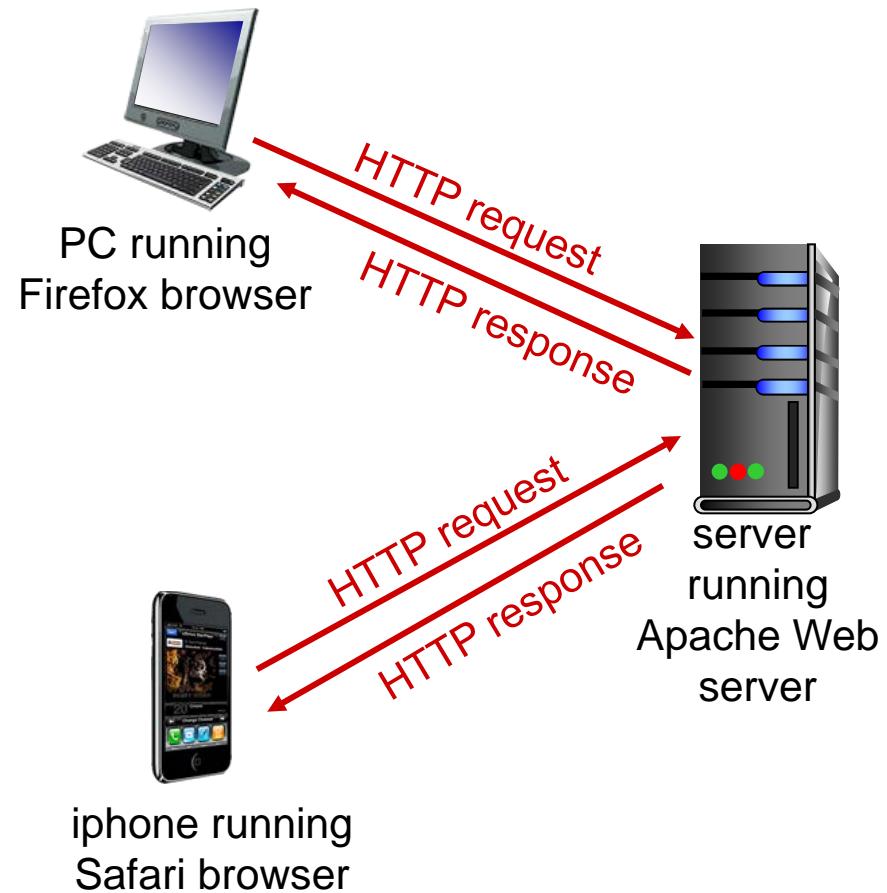
host name

path name

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



HTTP overview (continued)

uses TCP:

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

HTTP is “stateless”

- ❖ server maintains no information about past client requests

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

aside

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

Non-persistent HTTP

suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

(contains text,
references to 10
jpeg images)

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

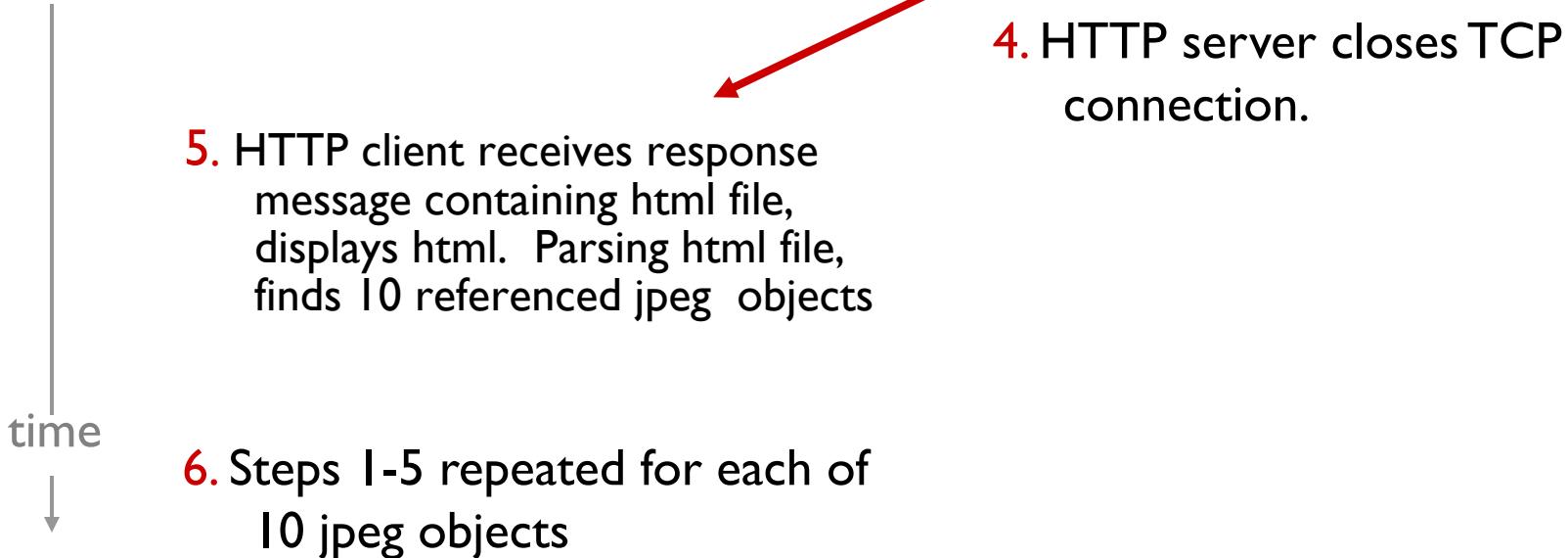
Ib. HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80. “accepts” connection, notifying client

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

time
↓

Non-persistent HTTP (cont.)

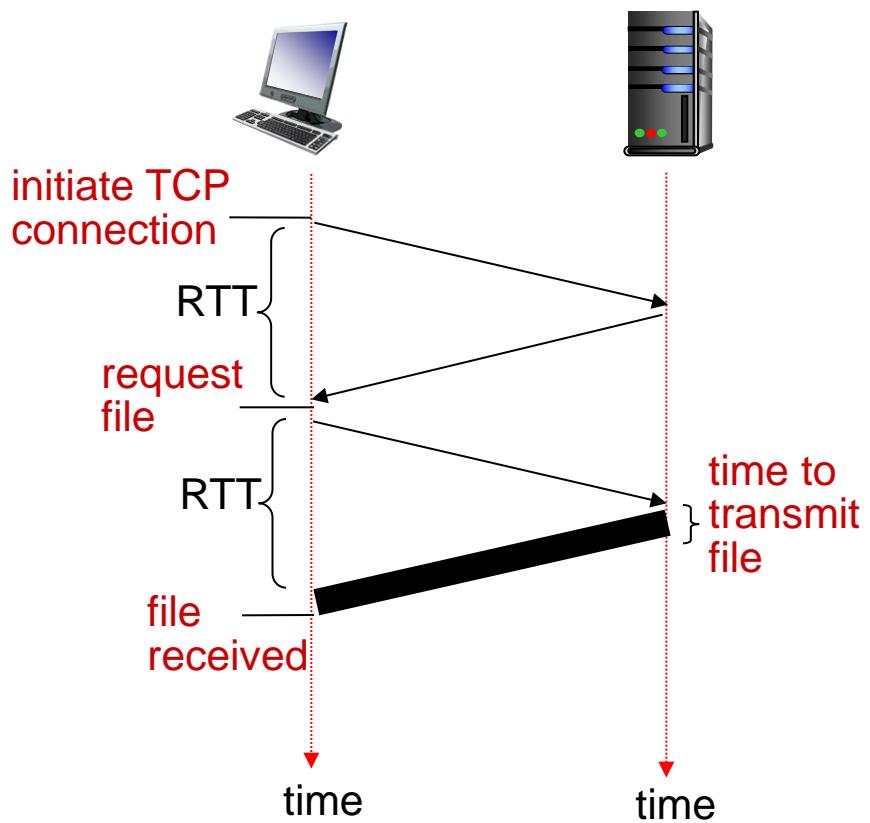


Non-persistent HTTP: response time

RTT (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 = $2\text{RTT} + \text{file transmission time}$



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

HTTP request message

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

request line
(GET, POST,
HEAD commands)

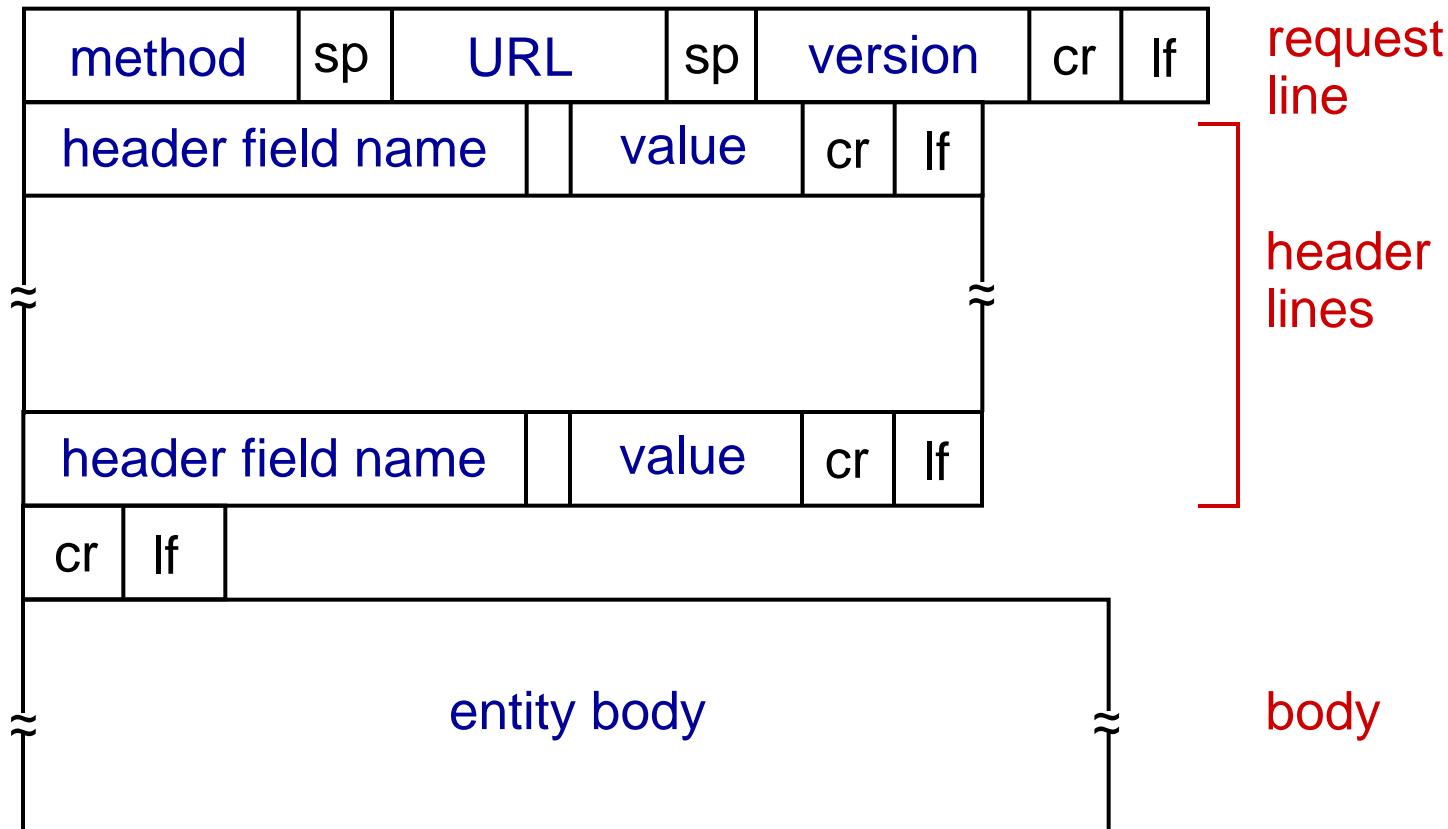
header
lines

carriage return,
line feed at start
of line indicates
end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return character
line-feed character

HTTP request message: general format



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:

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

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

HTTP response message

status line

(protocol

status code

status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS)\r\nLast-Modified: Tue, 30 Oct 2007 17:00:02  
GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-  
1\r\n\r\n
```

```
data data data data data ...
```

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

Trying out HTTP (client side) for yourself

I. Telnet to your favorite Web server:

telnet H

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**

by typing this in (hit carriage
return twice), you send
this minimal (but complete)
GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

User-server state: cookies

many Web sites use cookies

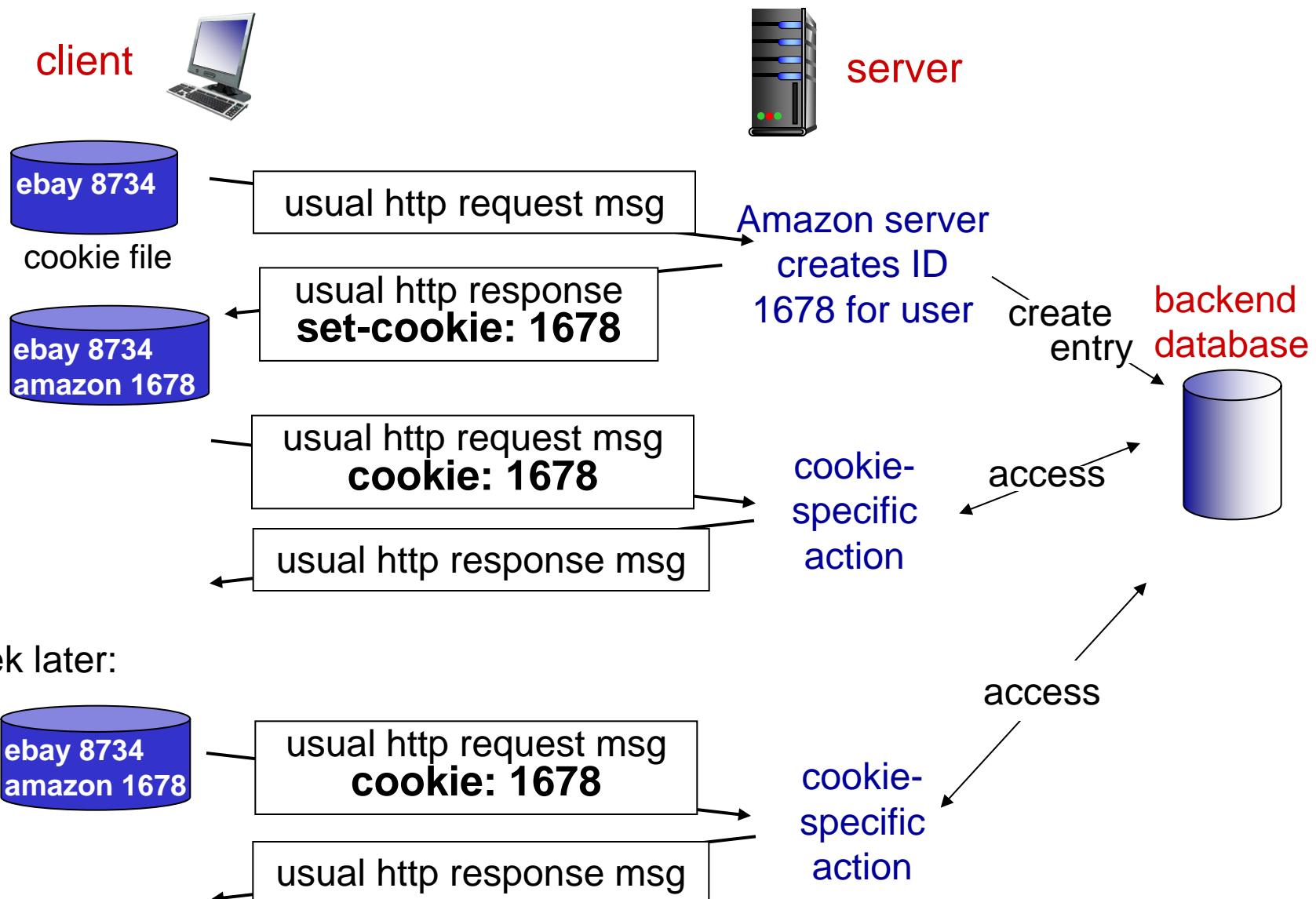
four components:

- 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

Cookies: keeping “state” (cont.)



Cookies (continued)

*what cookies can be used
for:*

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

*aside
cookies and privacy:*

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

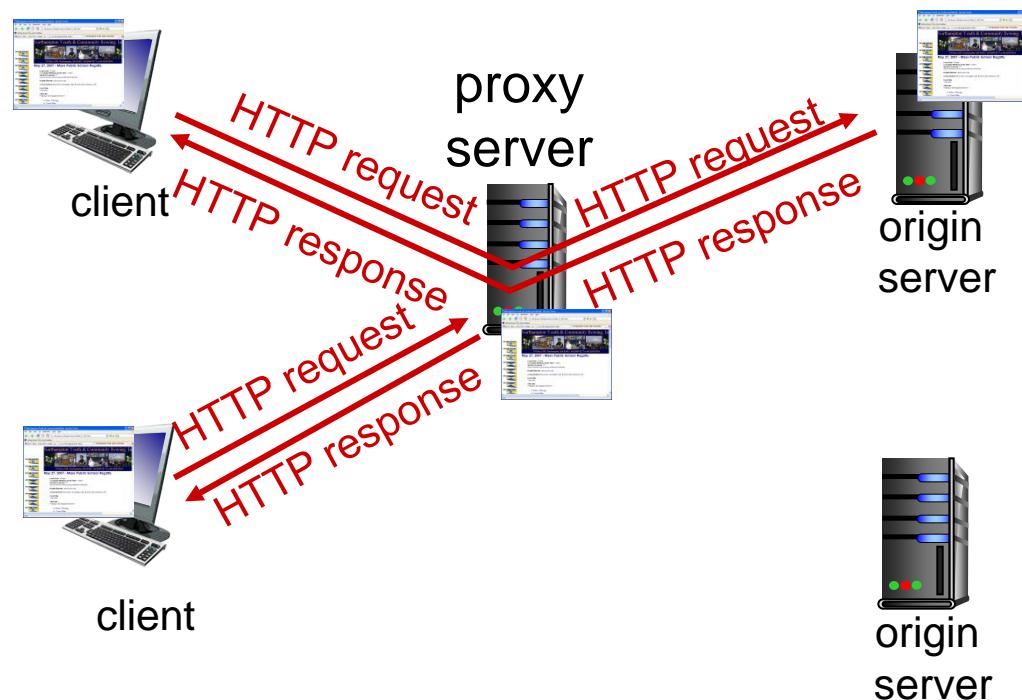
how to keep “state”:

- ❖ protocol endpoints: maintain state at sender/receiver over multiple transactions
- ❖ cookies: http messages carry state

Web caches (proxy server)

goal: satisfy client request without involving origin server

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



More about Web caching

- ❖ cache acts as both client and server
 - server for original requesting client
 - client to origin server
 - ❖ typically cache is installed by ISP (university, company, residential ISP)
- why Web caching?*
- ❖ reduce response time for client request
 - ❖ reduce traffic on an institution's access link
 - ❖ Internet dense with caches: enables “poor” content providers to effectively deliver content (so too does P2P file sharing)

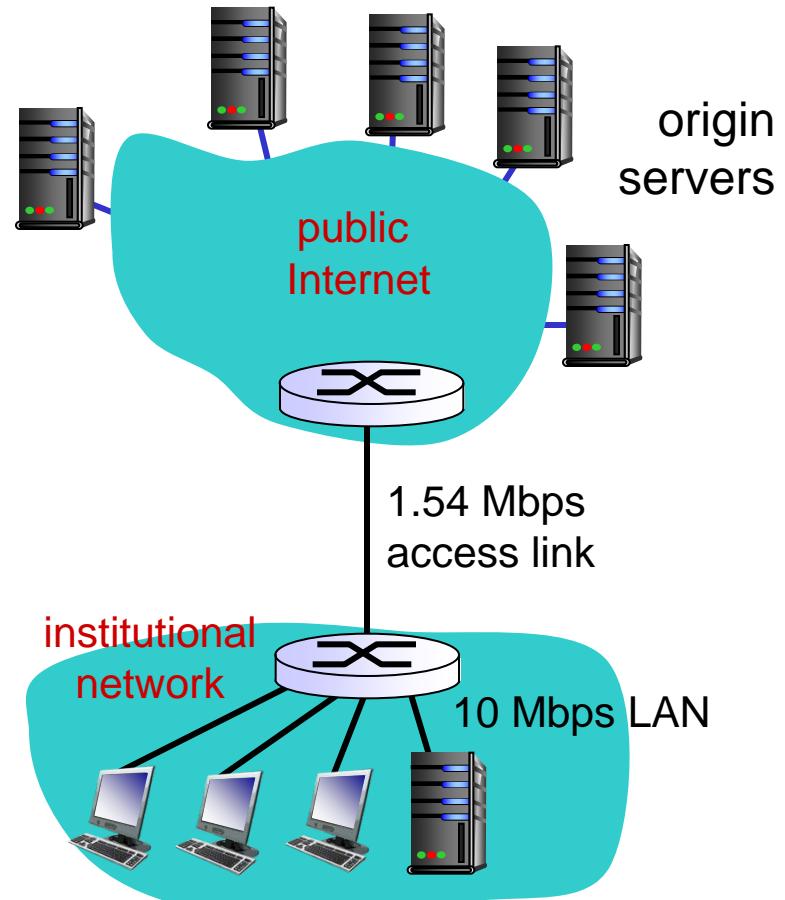
Caching example:

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:

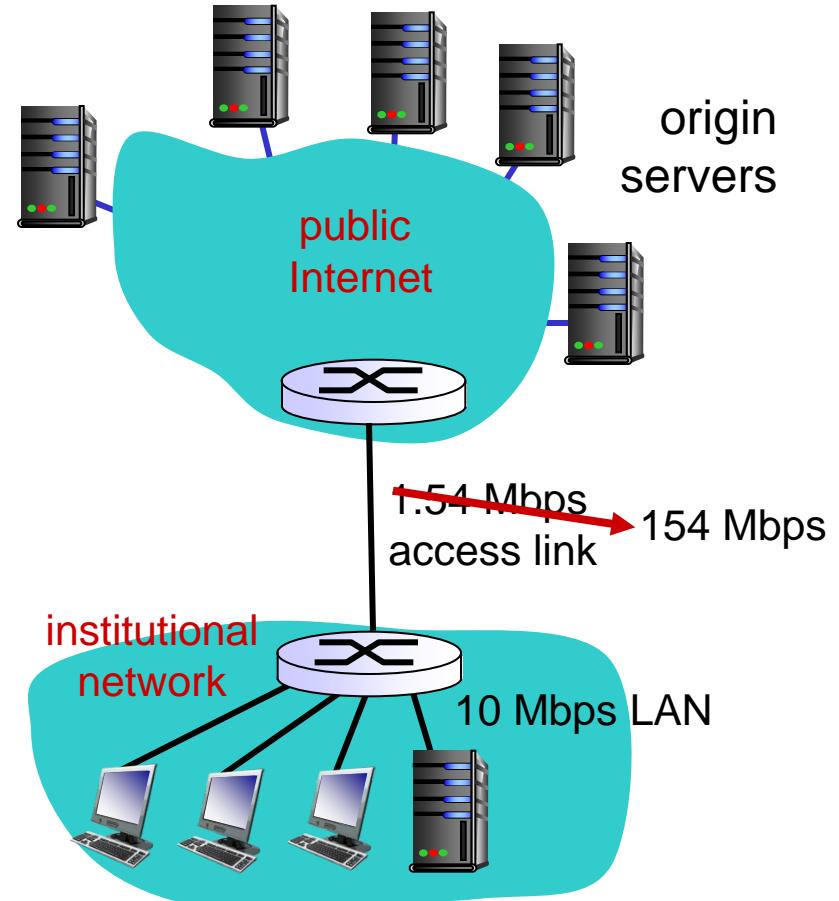
- ❖ LAN utilization: 15% *problem!*
- ❖ access link utilization = **99%**
- ❖ total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + usecs



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~~ \rightarrow 154 Mbps



consequences:

- ❖ LAN utilization: 15%
- ❖ access link utilization = ~~99%~~ \rightarrow 0.99%
- ❖ total delay = Internet delay + access delay + LAN delay
 $= 2 \text{ sec} + \cancel{\text{minutes}} + \cancel{\text{usecs}} \rightarrow \text{msecs}$

Cost: increased access link speed (not cheap!)

Caching example: install local cache

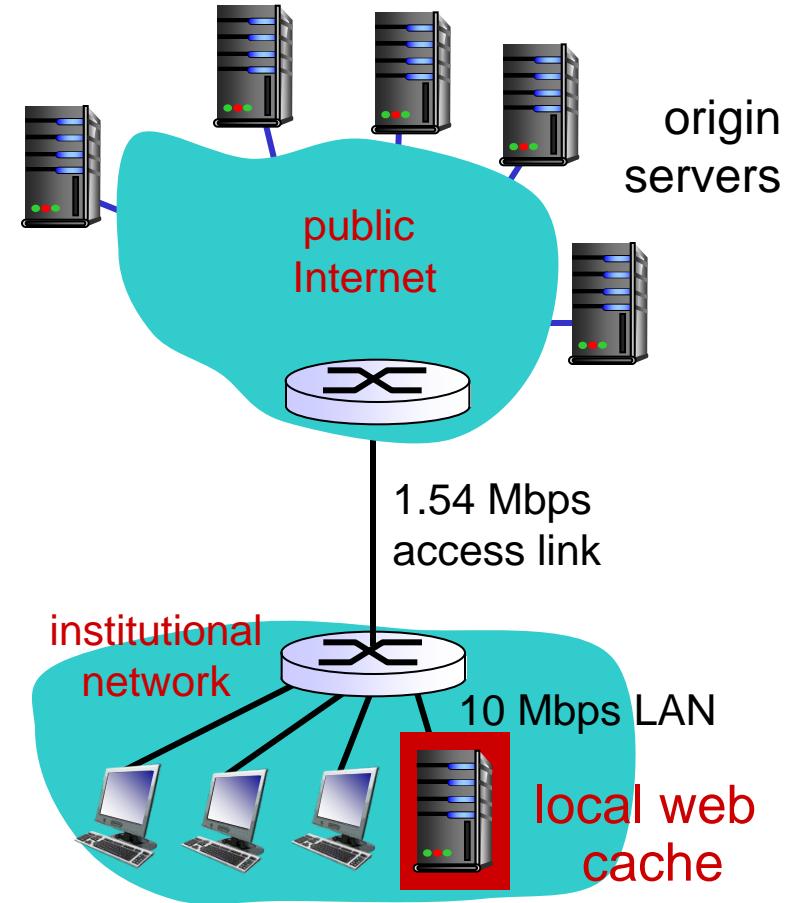
assumptions:

- ❖ 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 = ?
- ❖ total delay = ?

How to compute link utilization, delay?

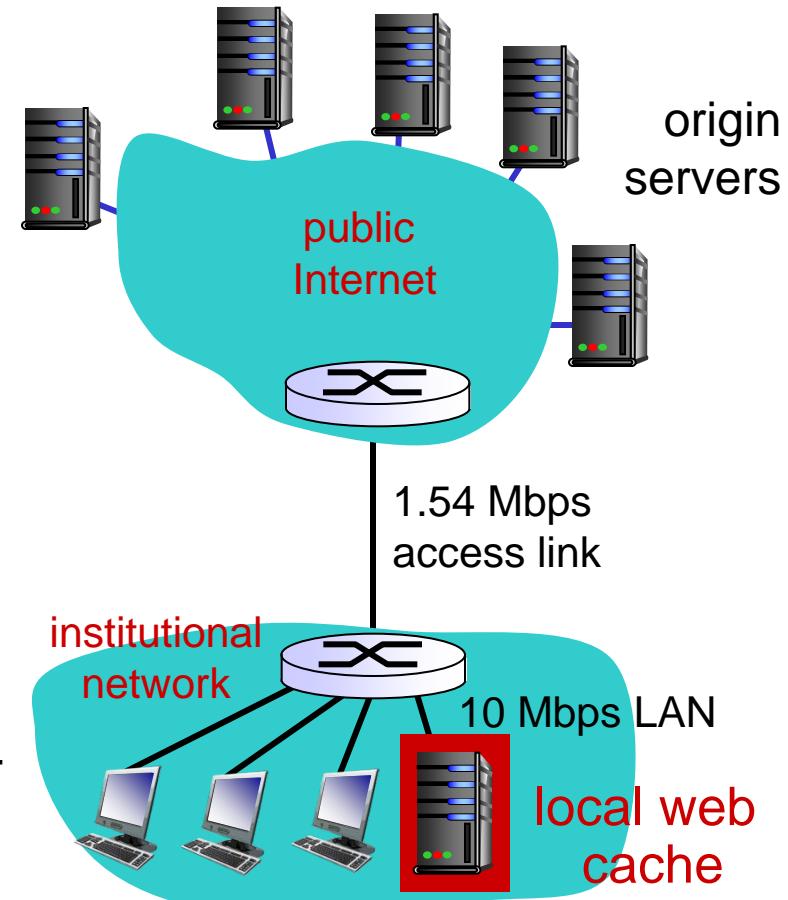


Cost: web cache (cheap!)

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
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
 - utilization = $0.9 / 1.54 = .58$
- ❖ total delay
 - $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
 - $= 0.6 (2.01) + 0.4 (0.01)$
 - $= \sim 1.2 \text{ secs}$
 - less than with 1.54 Mbps link (and cheaper too!)



Conditional GET

- ❖ **Goal:** don't send object if cache has up-to-date cached version

- no object transmission delay

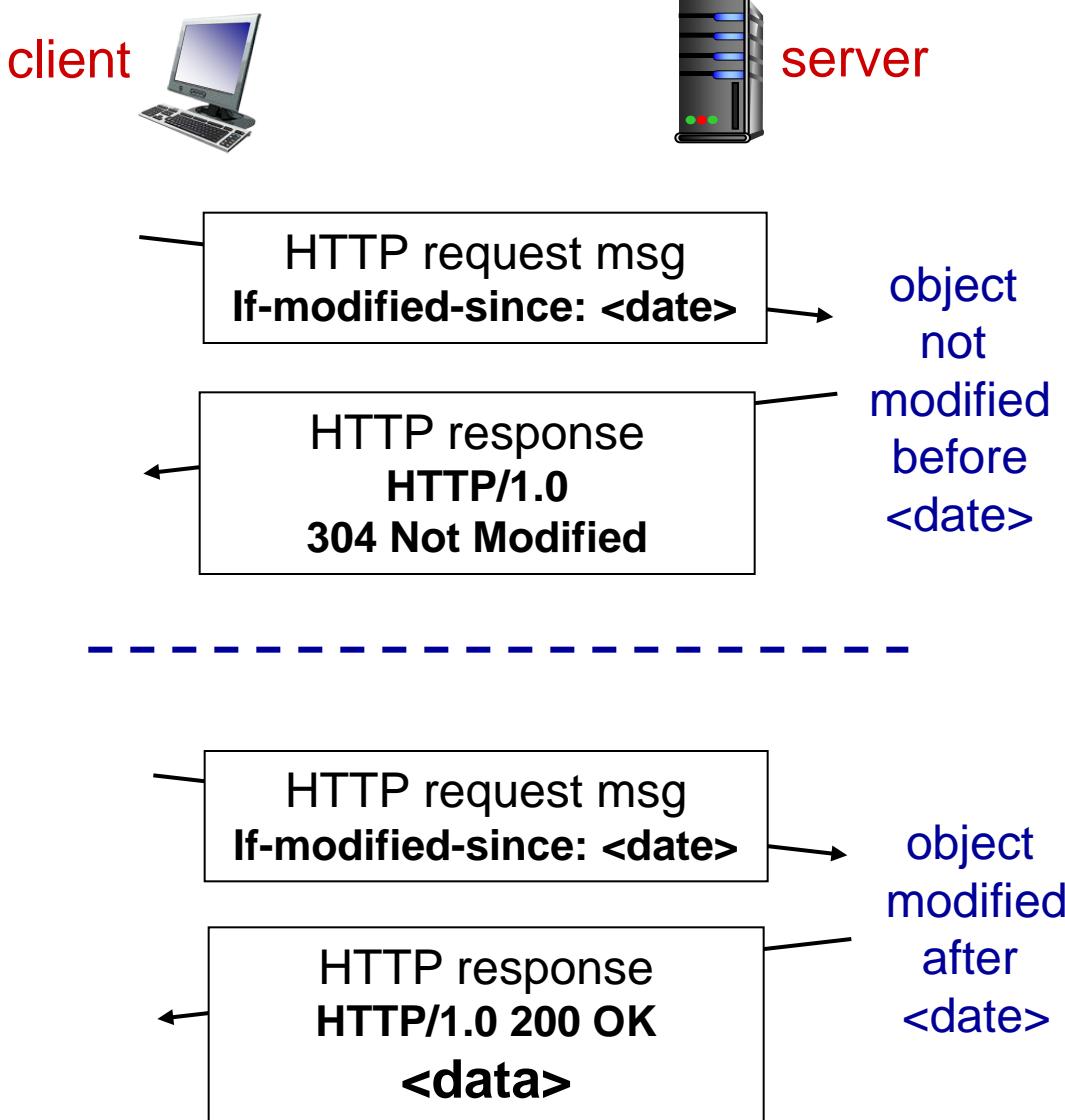
- lower link utilization

- ❖ **cache:** specify date of cached copy in HTTP request

If-modified-since:
<date>

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

HTTP/1.0 304 Not Modified



Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

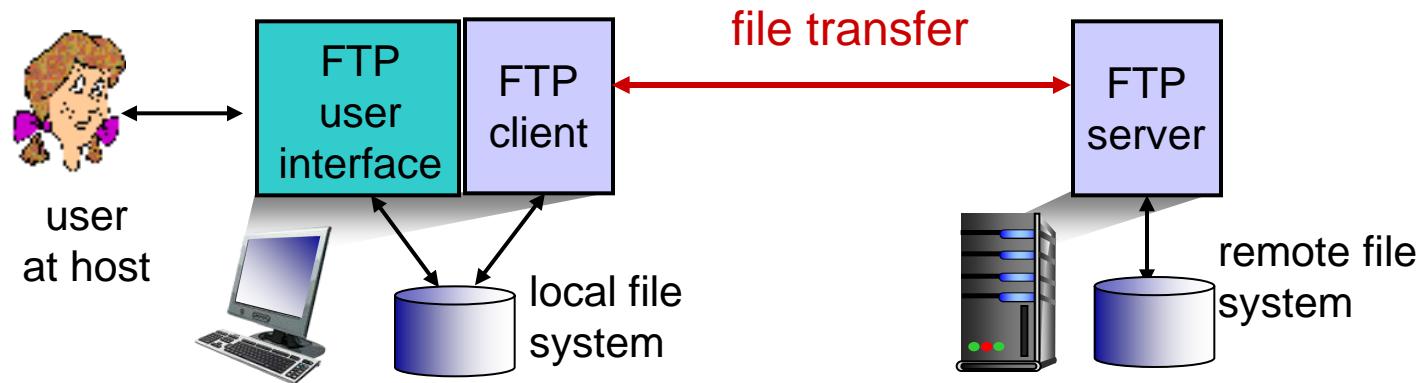
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

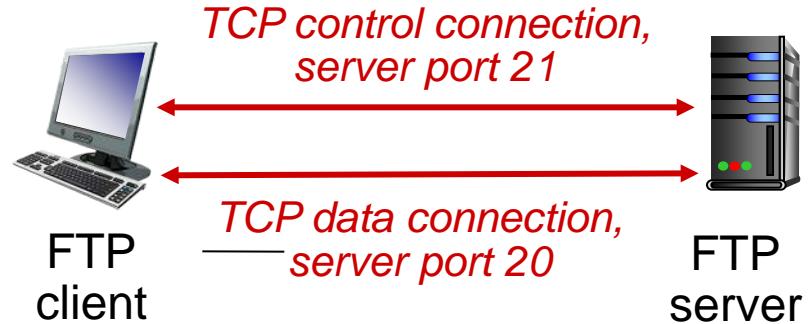
FTP: the file transfer protocol



- ❖ transfer file to/from remote host
- ❖ client/server model
 - *client*: side that initiates transfer (either to/from remote)
 - *server*: remote host
- ❖ ftp: RFC 959
- ❖ ftp server: port 21

FTP: separate control, data connections

- ❖ FTP client contacts FTP server at port 21, using TCP
- ❖ client authorized over control connection
- ❖ client browses remote directory, sends commands over control connection
- ❖ when server receives file transfer command, *server* opens 2nd TCP data connection (for file) to client
- ❖ after transferring one file, server closes data connection



- ❖ server opens another TCP data connection to transfer another file
- ❖ control connection: “*out of band*”
- ❖ FTP server maintains “state”: current directory, earlier authentication

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

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**

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

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

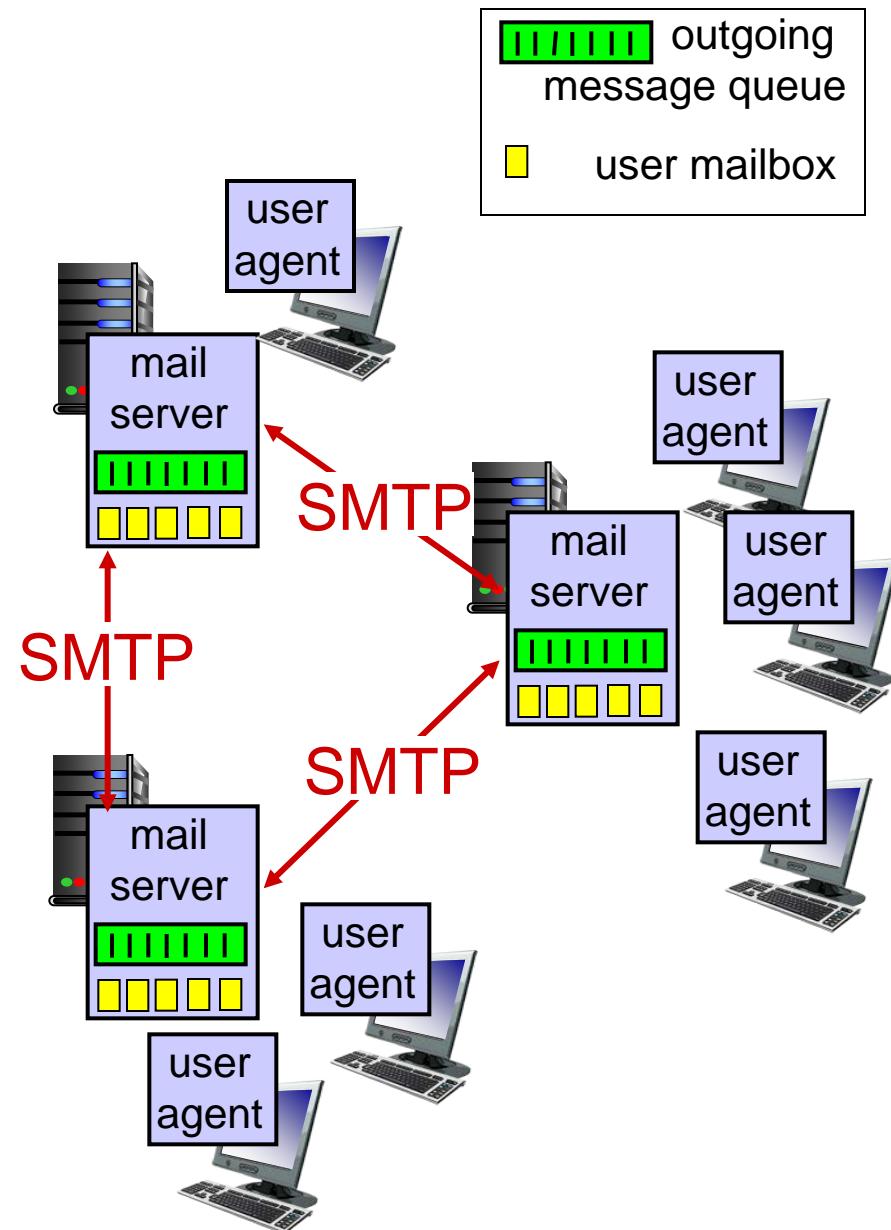
Electronic mail

Three major components:

- ❖ user agents
- ❖ mail servers
- ❖ simple mail transfer protocol: SMTP

User Agent

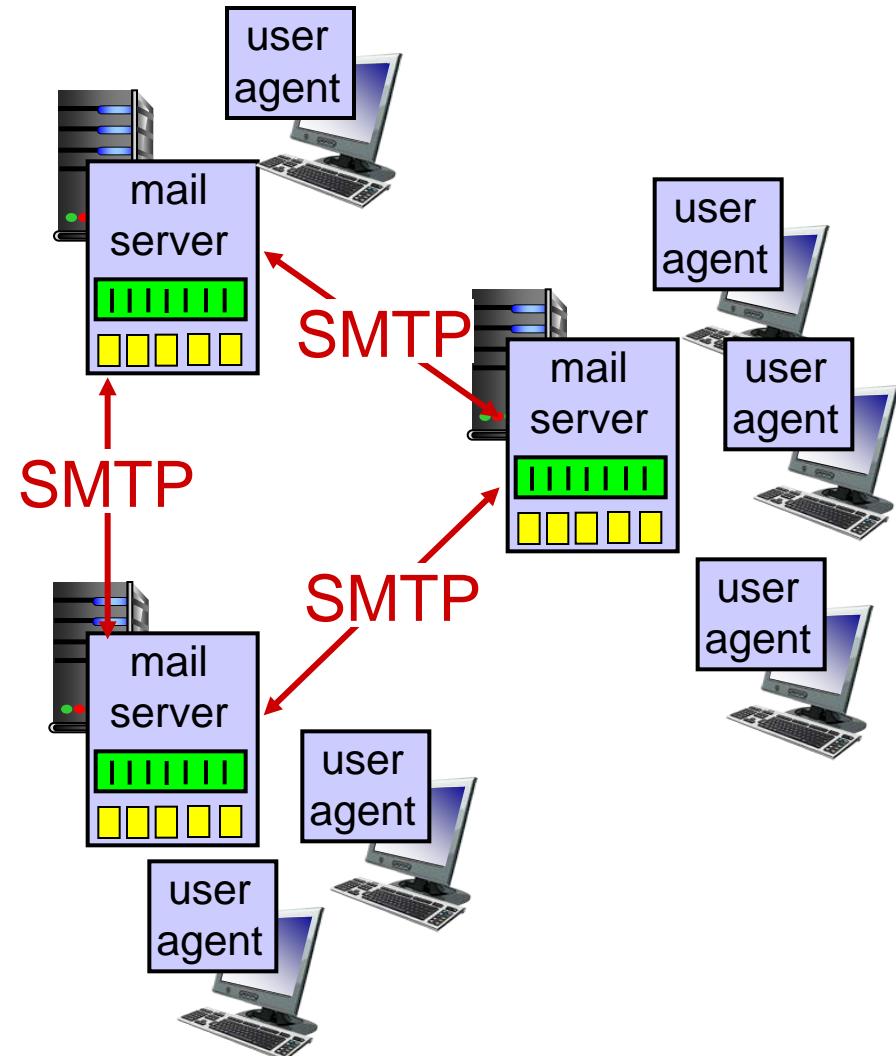
- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server



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

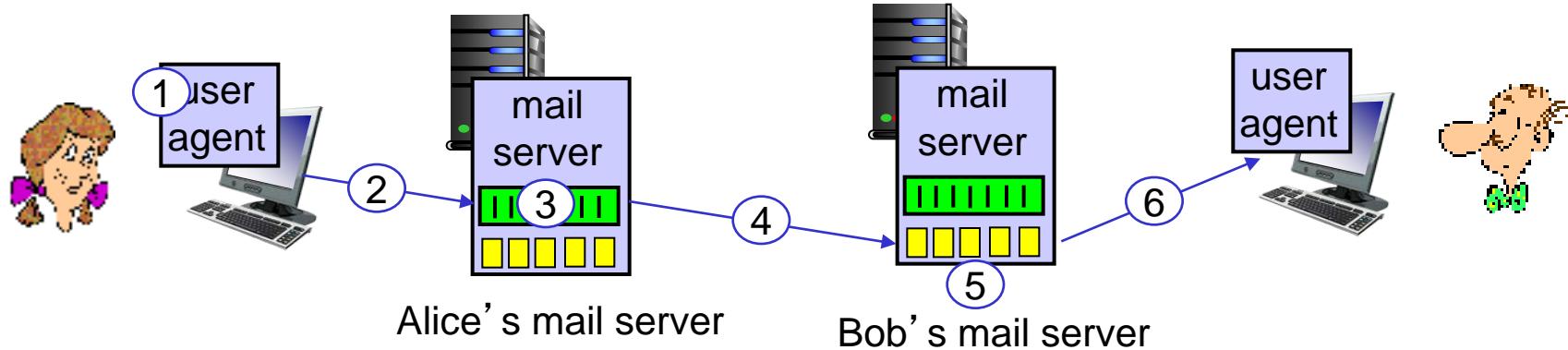


Electronic Mail: SMTP [RFC 2821]

- ❖ uses TCP to reliably transfer email message from client to server, port 25
- ❖ direct transfer: sending server to receiving server
- ❖ three phases of transfer
 - handshaking (greeting)
 - transfer of messages
 - closure
- ❖ command/response interaction (like HTTP, FTP)
 - commands: ASCII text
 - response: status code and phrase
- ❖ messages must be in 7-bit ASCII

Scenario: Alice sends message to Bob

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



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
```

Try SMTP interaction for yourself:

- ❖ `telnet servername 25`
- ❖ see 220 reply from server
- ❖ enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

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

SMTP: final words

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

comparison with HTTP:

- ❖ HTTP: pull
- ❖ SMTP: push
- ❖ both have ASCII command/response interaction, status codes
- ❖ HTTP: each object encapsulated in its own response msg
- ❖ SMTP: multiple objects sent in multipart msg

Mail message format

SMTP: protocol for
exchanging email msgs

RFC 822: standard for text
message format:

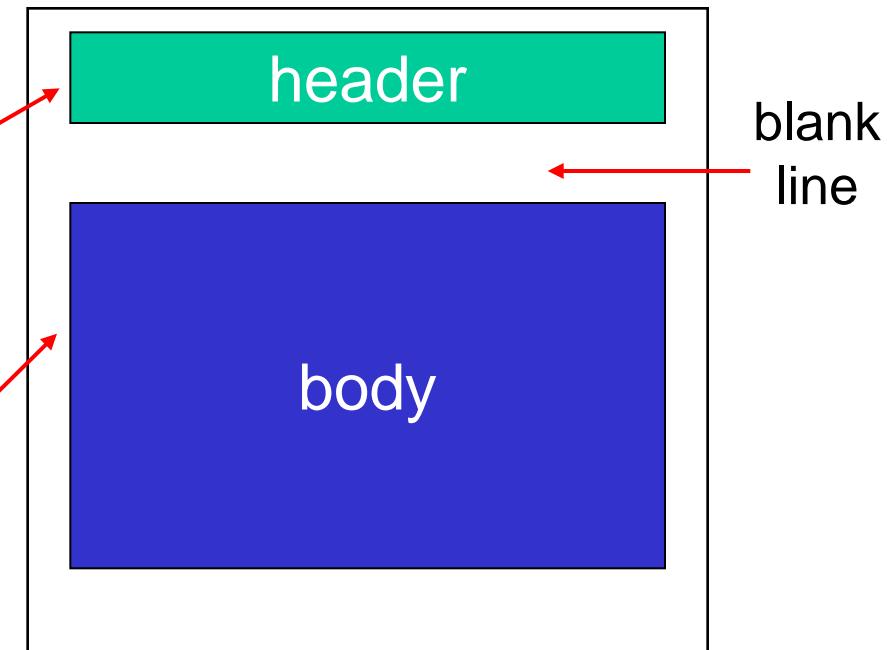
❖ header lines, e.g.,

- To:
- From:
- Subject:

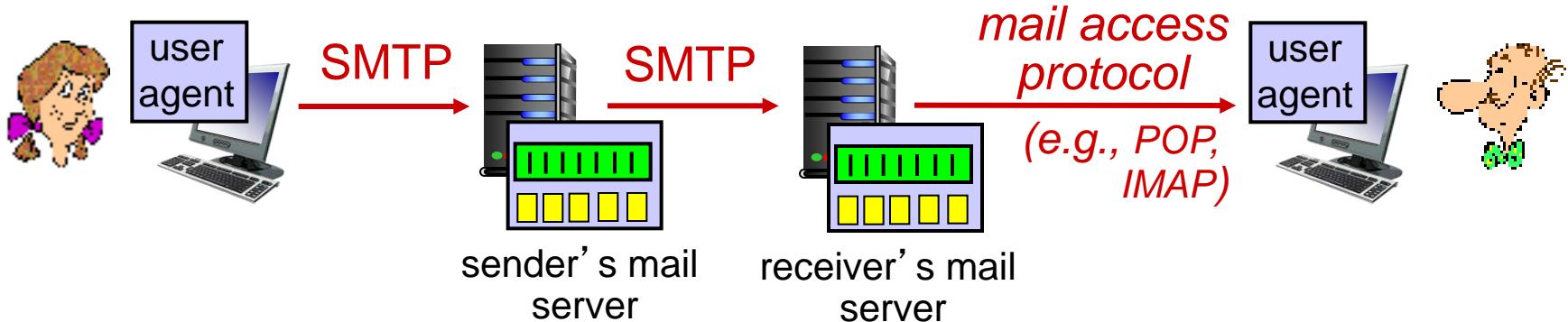
*different from SMTP MAIL
FROM, RCPT TO:
commands!*

❖ Body: the “message”

- ASCII characters only



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.

POP3 protocol

authorization phase

- ❖ client commands:
 - **user**: declare username
 - **pass**: password
- ❖ server responses
 - +OK
 - -ERR

transaction phase, client:

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

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

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

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

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

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

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”

DNS: services, structure

DNS services

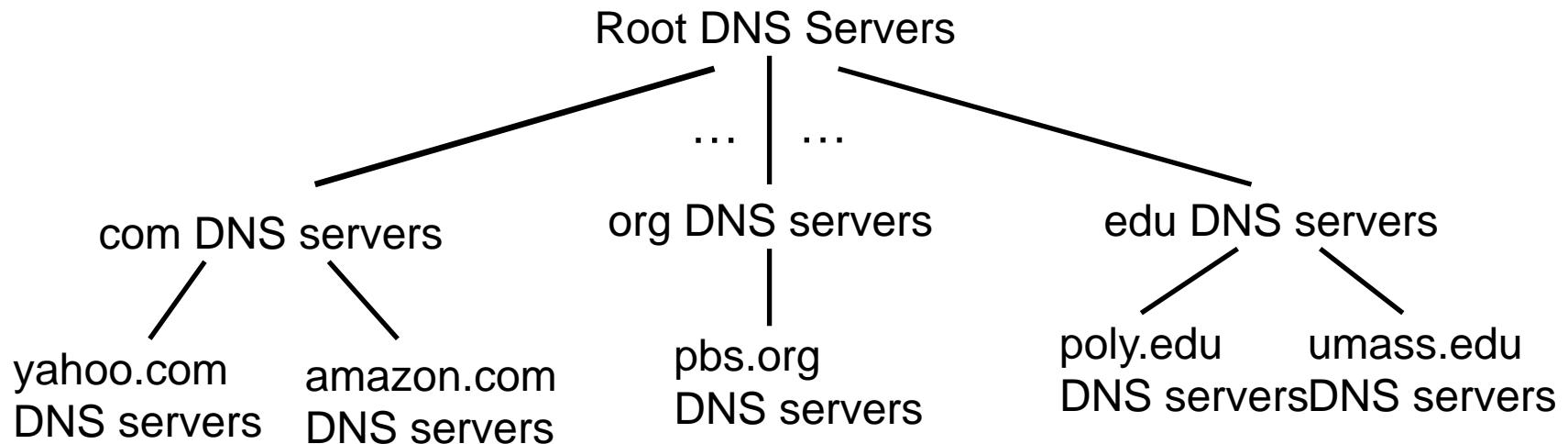
- ❖ hostname to IP address translation
- ❖ host aliasing
 - canonical, alias names
- ❖ mail server aliasing
- ❖ load distribution
 - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

A: *doesn't scale!*

DNS: a distributed, hierarchical database

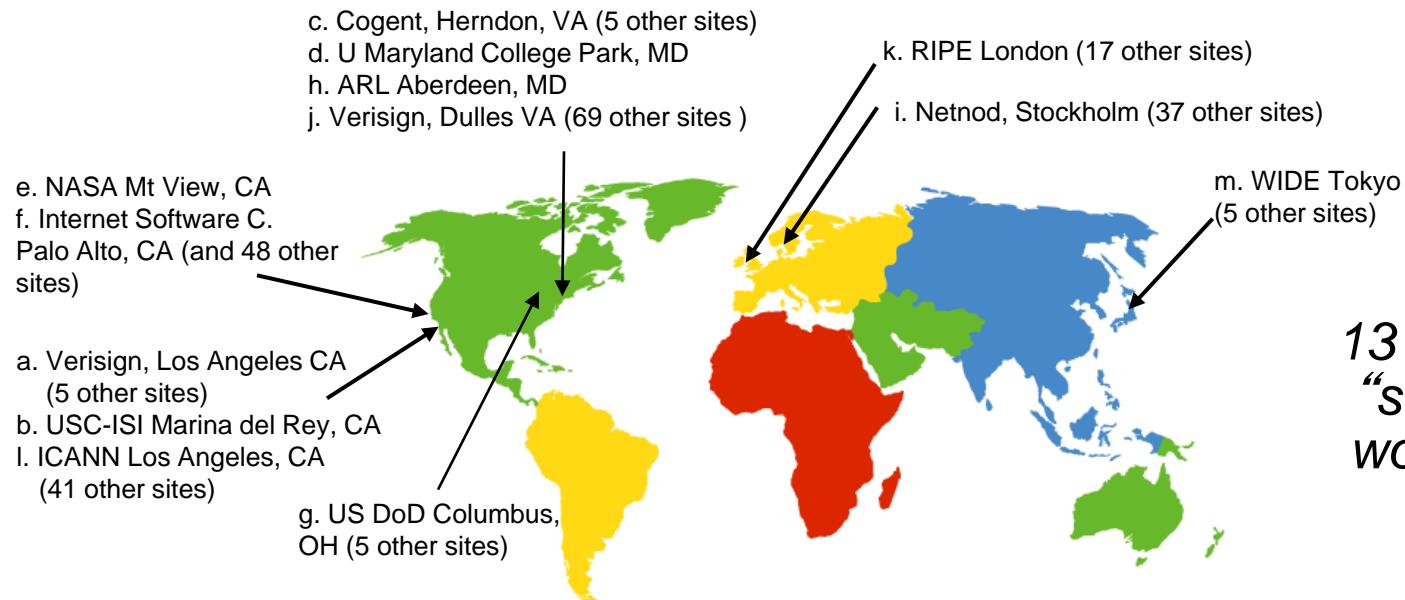


client wants IP for www.amazon.com; 1st approx:

- ❖ 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

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



*13 root name
“servers”
worldwide*

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

Local DNS name server

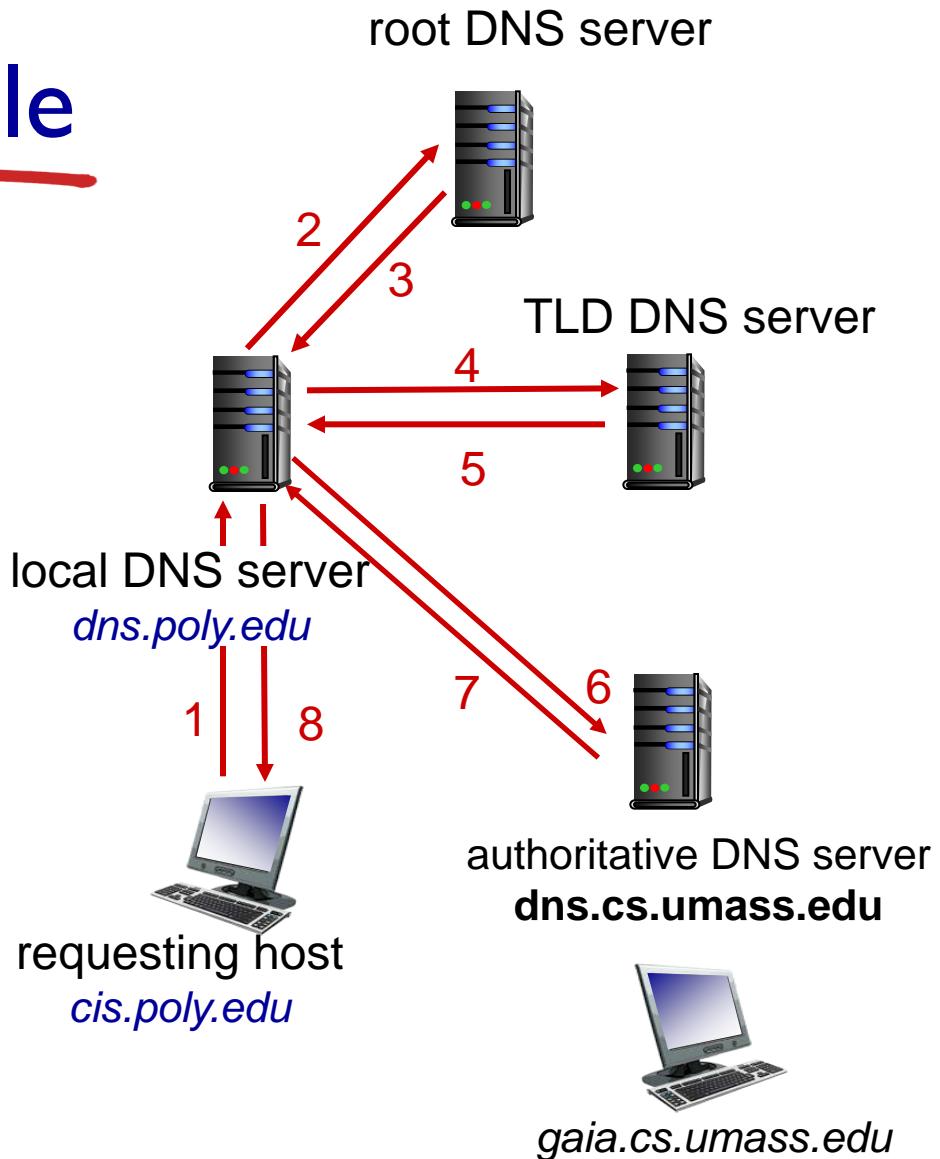
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
 - also called “default name server”
- ❖ when host makes DNS query, query is sent to its local DNS server
 - has local cache of recent name-to-address translation pairs (but may be out of date!)
 - acts as proxy, forwards query into hierarchy

DNS name resolution example

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

iterated query:

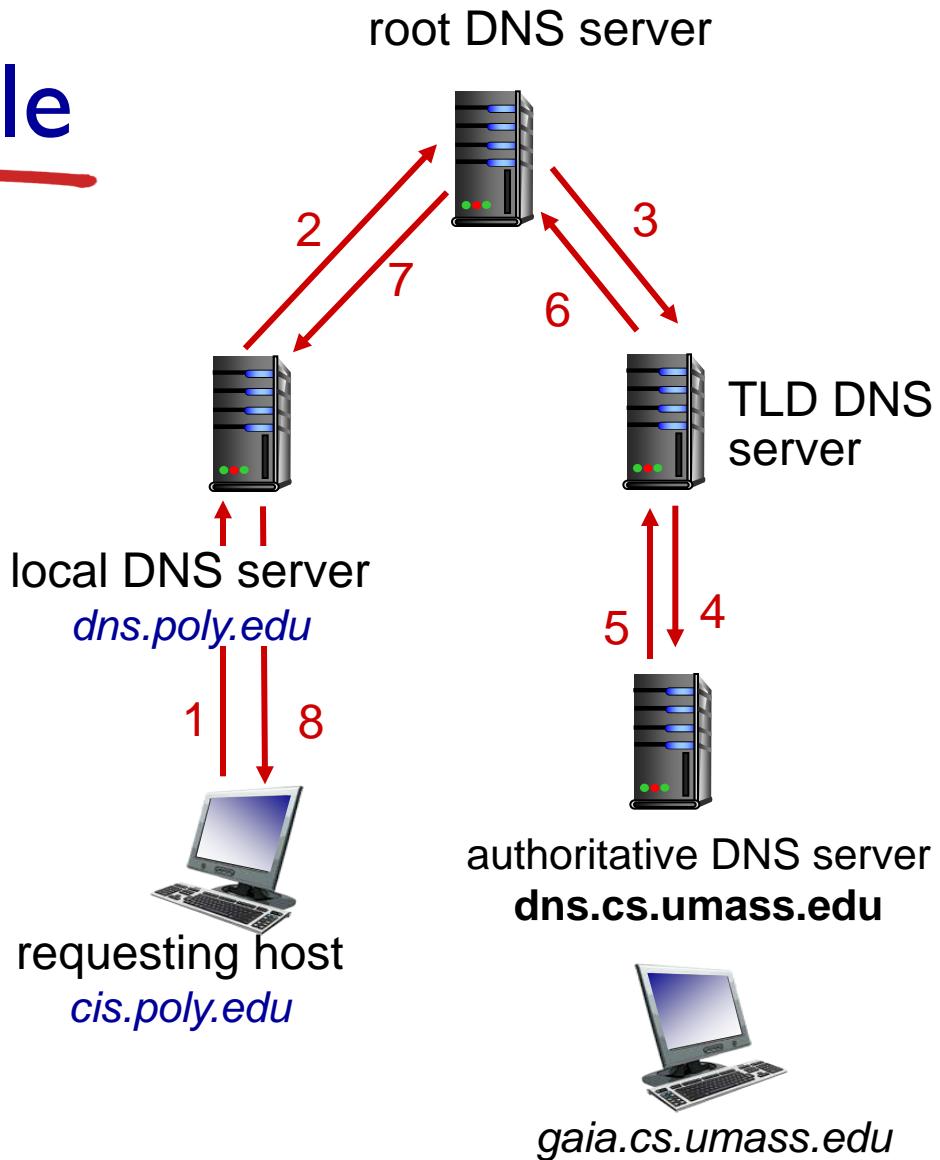
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



DNS name resolution example

recursive query:

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



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

DNS records

DNS: distributed db storing resource records (**RR**)

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

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

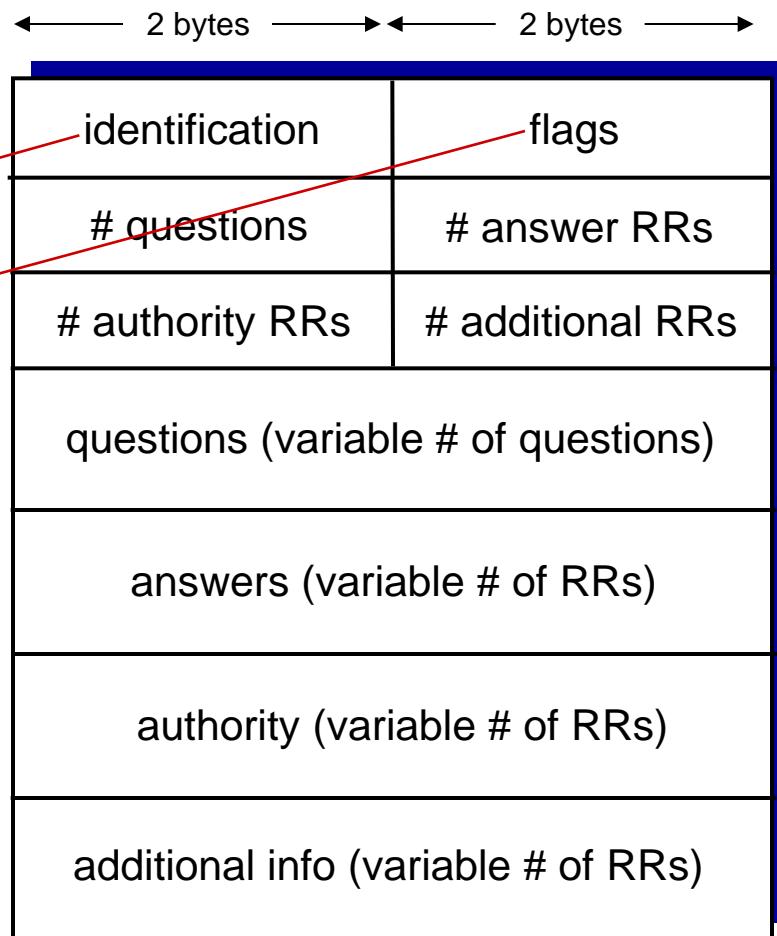
- **value** is name of mailserver associated with **name**

DNS protocol, messages

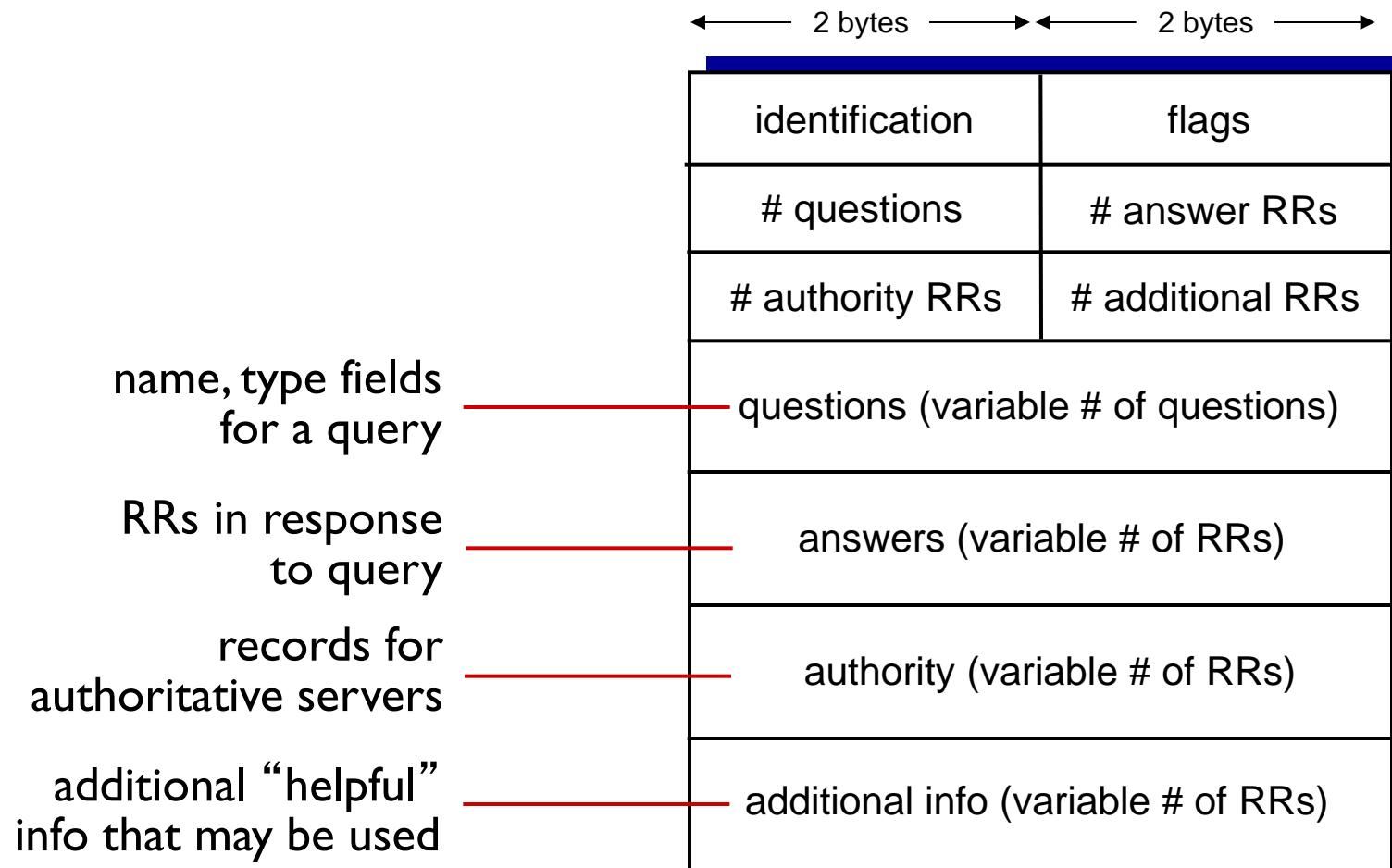
- ❖ *query* and *reply* messages, both with same *message format*

msg header

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



DNS protocol, messages



```
$ dig redhat.com

; <>> DiG 9.7.3-RedHat-9.7.3-2.el6 <>> redhat.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62863
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 3

;; QUESTION SECTION:
;redhat.com.           IN      A

;; ANSWER SECTION:
redhat.com.        37      IN      A      209.132.183.81

;; AUTHORITY SECTION:
redhat.com.        73      IN      NS     ns4.redhat.com.
redhat.com.        73      IN      NS     ns3.redhat.com.
redhat.com.        73      IN      NS     ns2.redhat.com.
redhat.com.        73      IN      NS     ns1.redhat.com.

;; ADDITIONAL SECTION:
ns1.redhat.com.    73      IN      A      209.132.186.218
ns2.redhat.com.    73      IN      A      209.132.183.2
ns3.redhat.com.    73      IN      A      209.132.176.100

;; Query time: 13 msec
;; SERVER: 209.144.50.138#53(209.144.50.138)
;; WHEN: Thu Jan 12 10:09:49 2012
;; MSG SIZE  rcvd: 164
```

Inserting records into DNS

- ❖ example: new startup “Network Utopia”
- ❖ register name `networkutopia.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.networkutopia.com`; type MX record for `networkutopia.com`

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 bypassed
- ❖ Bombard TLD servers
 - Potentially more dangerous

Redirect attacks

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

Exploit DNS for DDoS

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

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

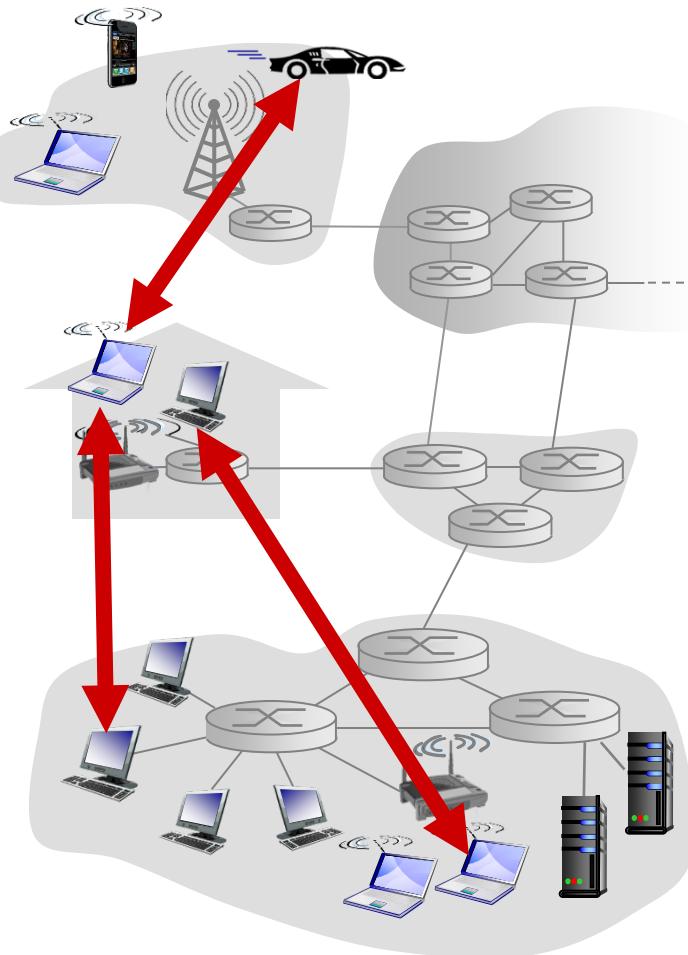
2.7 socket programming with UDP and TCP

Pure P2P architecture

- ❖ no always-on server
- ❖ arbitrary end systems directly communicate
- ❖ peers are intermittently connected and change IP addresses

examples:

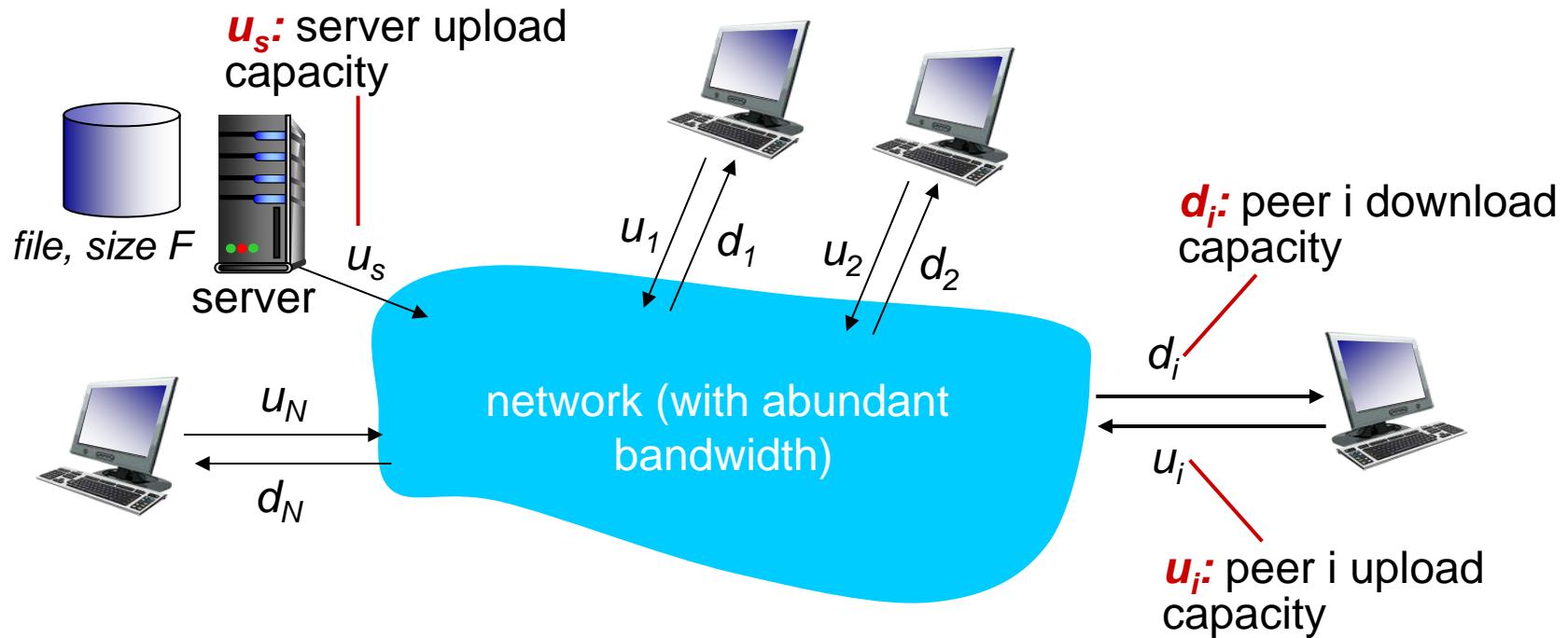
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

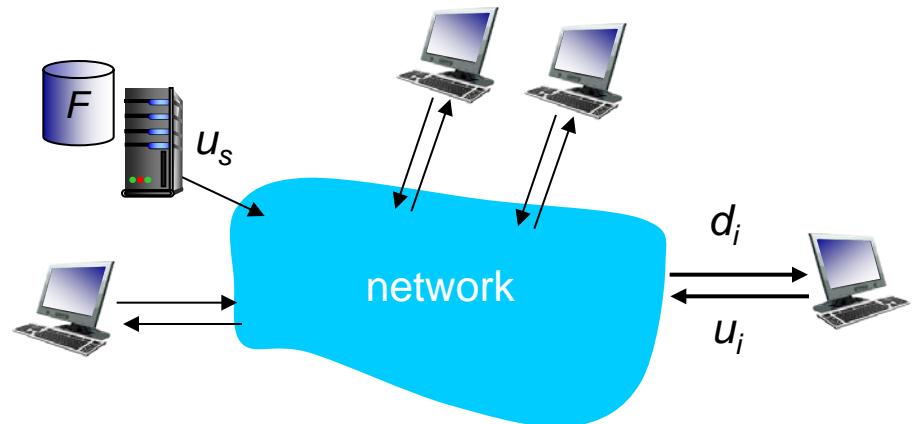
Question: how much time to distribute file (size F) from one server to N peers?

- peer upload/download capacity is limited resource



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
 - max client download time: F/d_{\min}



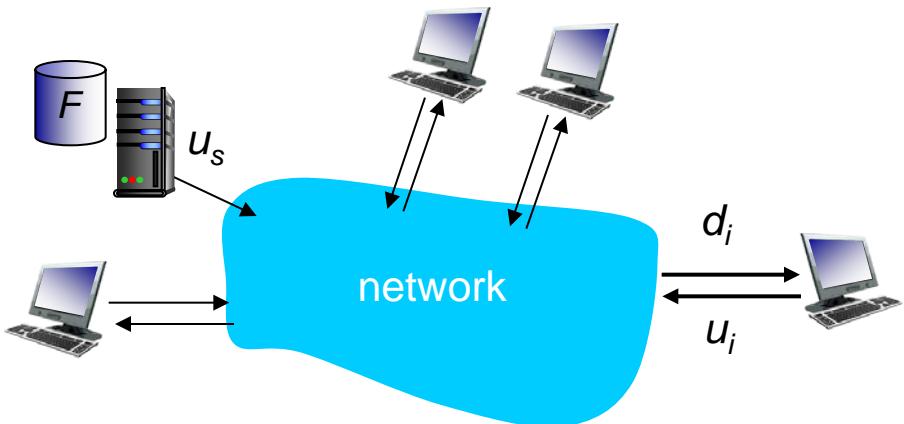
*time to distribute F
to N clients using
client-server approach*

$$D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$$

increases linearly in N

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
 - max client download time: F/d_{\min}
- ❖ **clients:** as aggregate must download NF bits
 - max upload rate (limiting max download rate) is $u_s + \sum u_i$



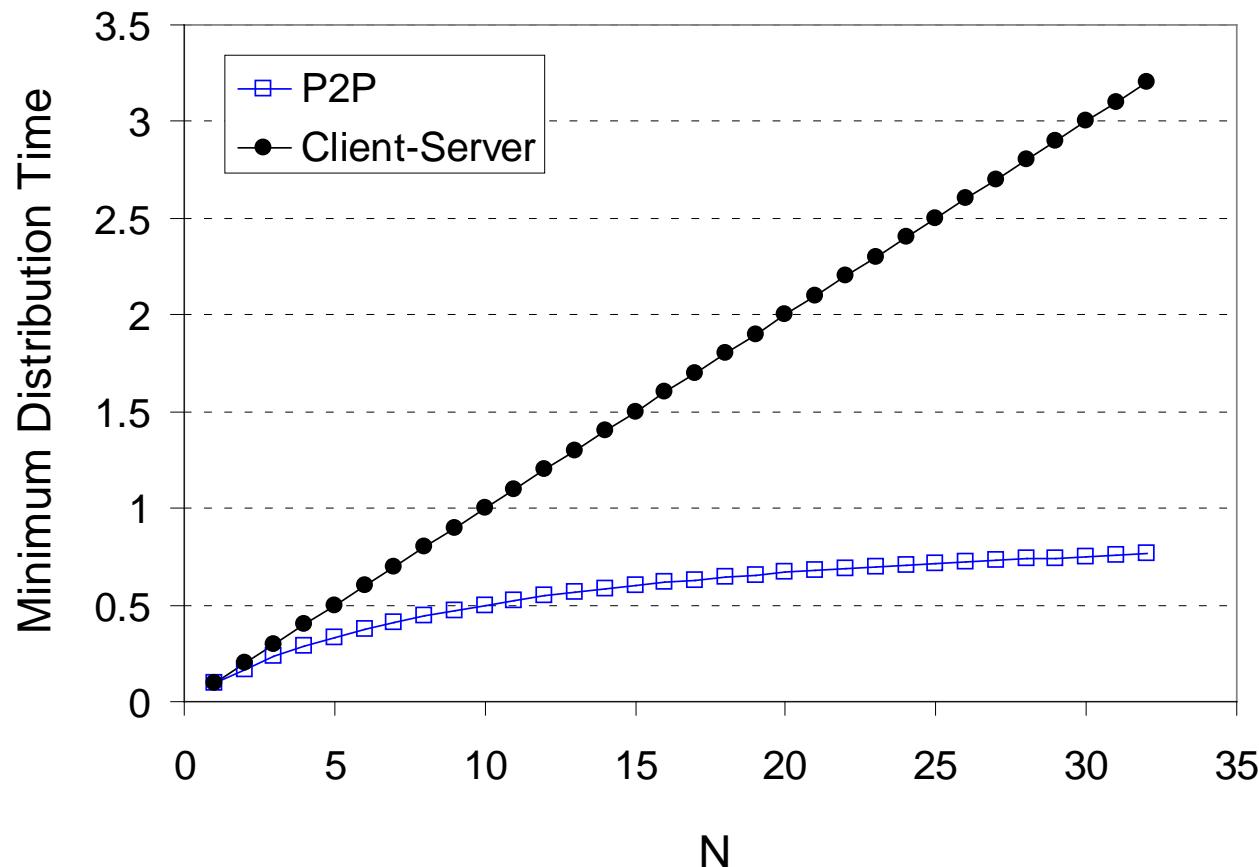
time to distribute F
to N clients using
P2P approach

$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

increases linearly in N ...
... but so does this, as each peer brings service capacity

Client-server vs. P2P: example

client upload rate = u , $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$

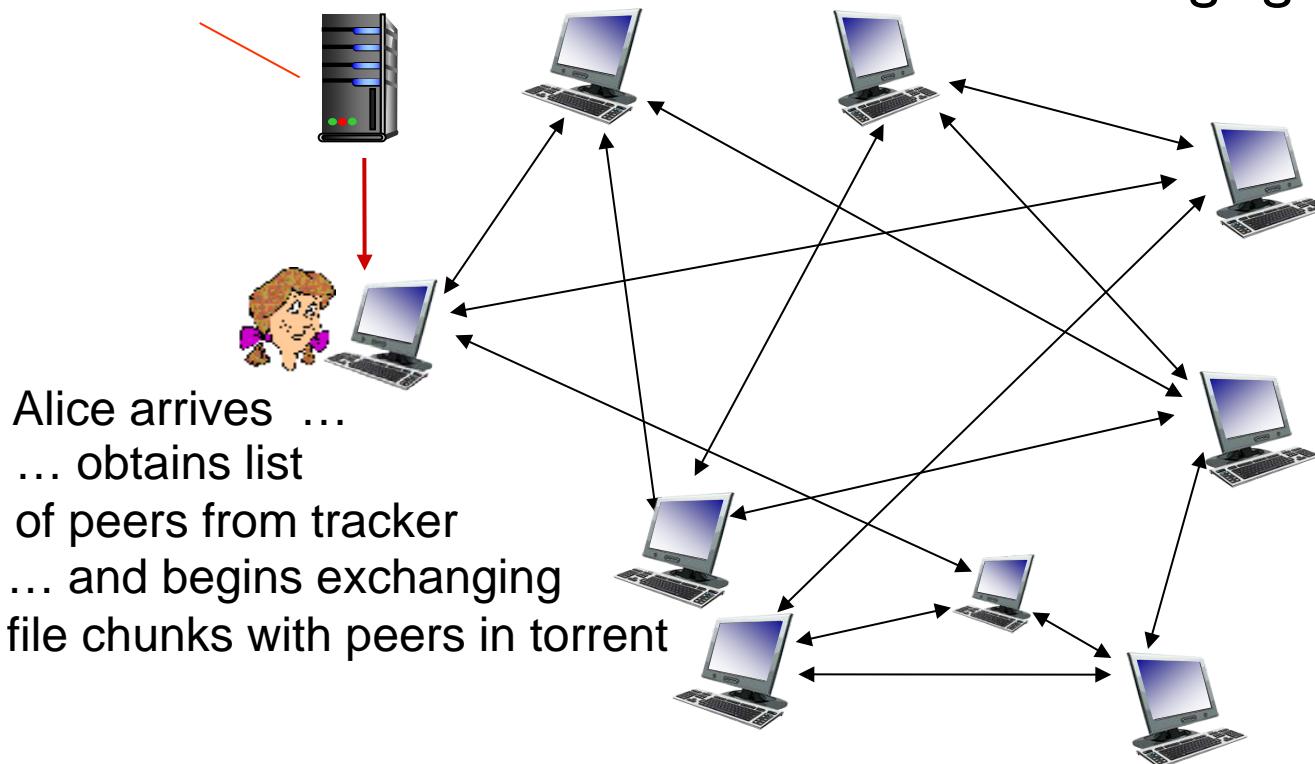


P2P file distribution: BitTorrent

- ❖ file divided into 256Kb chunks
- ❖ peers in torrent send/receive file chunks

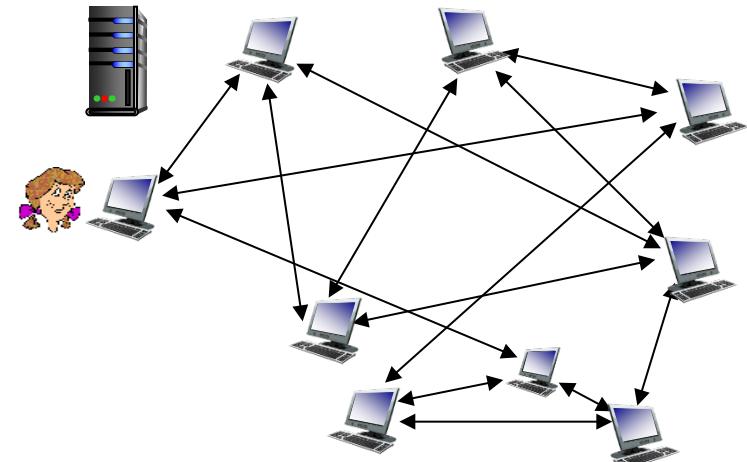
tracker: tracks peers
participating in torrent

torrent: group of peers
exchanging chunks of a file



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



BitTorrent: requesting, sending file chunks

requesting chunks:

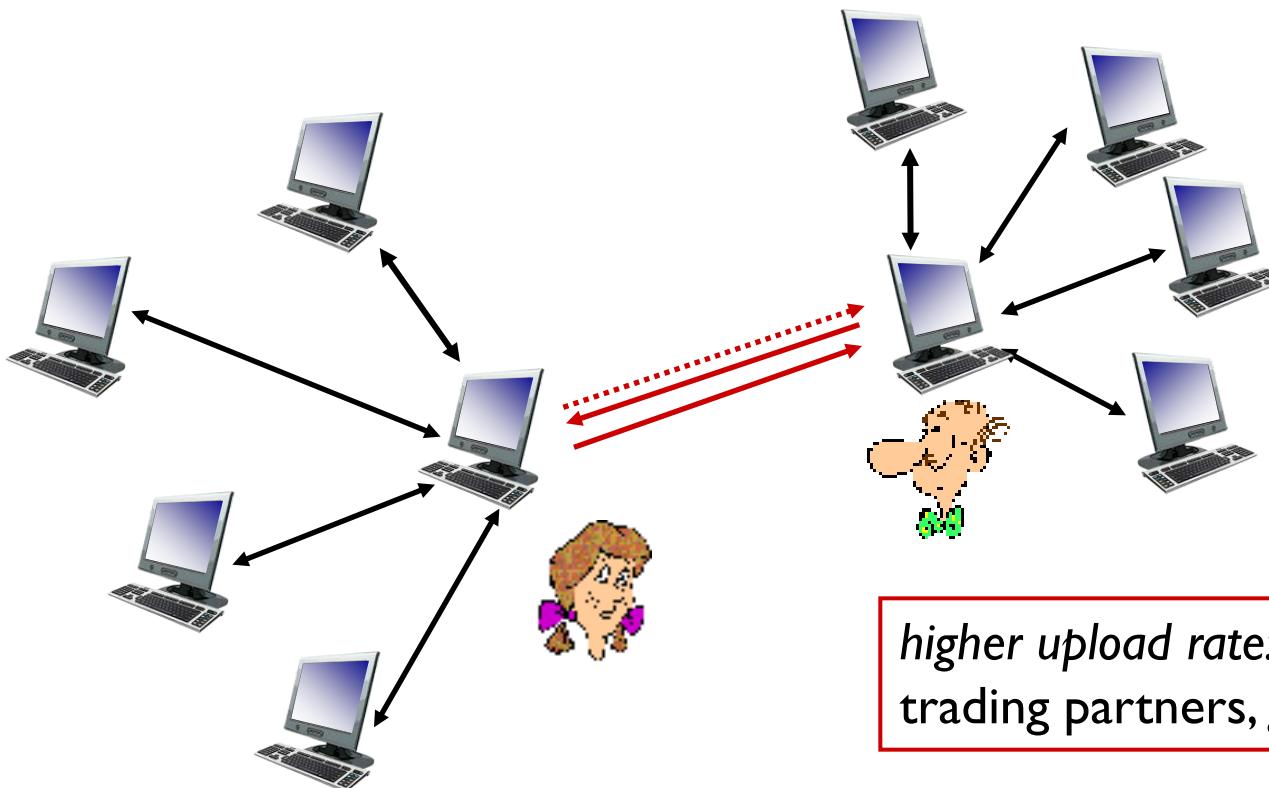
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, **rarest first**

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 every 10 secs
- ❖ every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 - newly chosen peer may join top 4

BitTorrent: tit-for-tat

- (1) Alice “optimistically unchoke” 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 !

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

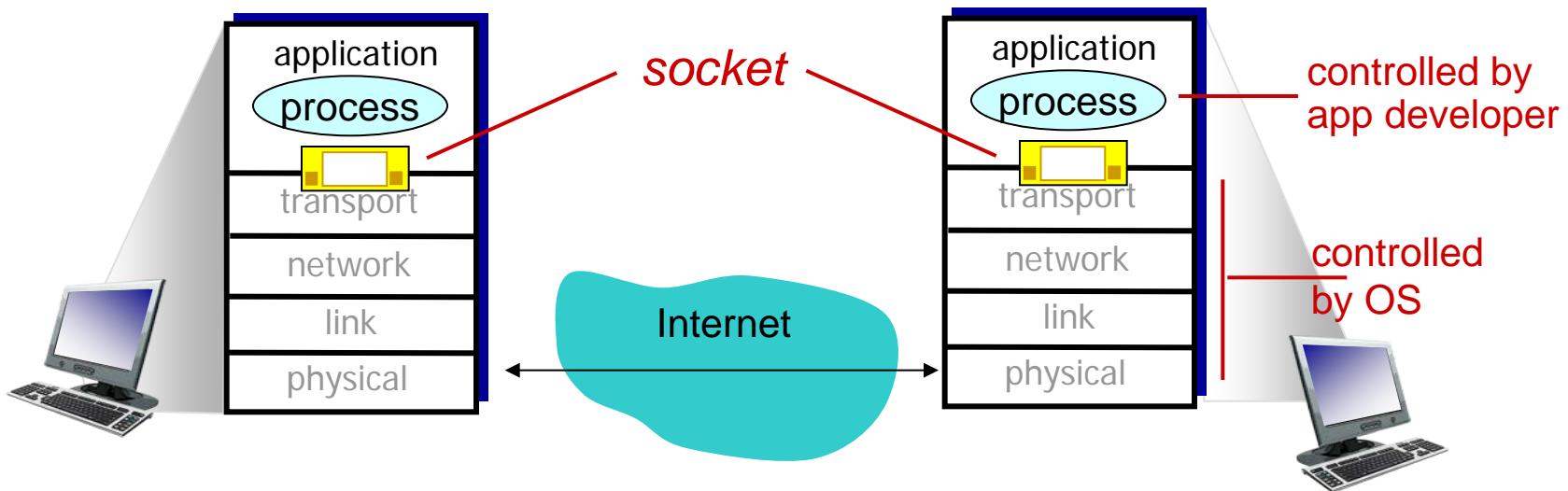
2.6 P2P applications

2.7 socket programming with UDP and TCP

Socket programming

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

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



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.

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

Client/server socket interaction: UDP

server (running on serverIP)

create socket, port= x:

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

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:

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

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket
close
clientSocket

Example app: UDP client

Python UDPCClient

```
include Python's socket  
library → from socket import *  
  
create UDP socket for  
server → clientSocket = socket(AF_INET,  
                                SOCK_DGRAM)  
  
get user keyboard  
input → message = raw_input('Input lowercase sentence:')  
  
Attach server name, port to  
message; send into socket → clientSocket.sendto(message,(serverName, serverPort))  
  
read reply characters from  
socket into string → modifiedMessage, serverAddress =  
clientSocket.recvfrom(2048)  
  
print out received string  
and close socket → print modifiedMessage  
clientSocket.close()
```

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
create UDP socket -----> serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port
number 12000 -----> serverSocket.bind(('', serverPort))
print 'The server is ready to receive'
loop forever -----> while 1:
Read from UDP socket into
message, getting client's
address (client IP and port) -----> message, clientAddress = serverSocket.recvfrom(2048)
                                            modifiedMessage = message.upper()
send upper case string -----> serverSocket.sendto(modifiedMessage, clientAddress)
back to this client
```

Socket programming with TCP

client must contact server

- ❖ server process must first be running
- ❖ server must have created socket (door) that **welcomes** client's contact

client contacts server by:

- ❖ Creating TCP socket, specifying IP address, port number of server process
- ❖ *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

Client/server socket interaction: TCP

server (running on hostid)

client

create socket,
port=x, for incoming
request:
`serverSocket = socket()`

wait for incoming
connection request
`connectionSocket =
serverSocket.accept()`

read request from
`connectionSocket`

write reply to
`connectionSocket`

close
`connectionSocket`

TCP
connection setup

create socket,
connect to **hostid**, port=x
`clientSocket = socket()`

send request using
`clientSocket`

read reply from
`clientSocket`

close
`clientSocket`

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'      //here, 127.0.0.1
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for
server, remote port 12000

→ clientSocket = socket(AF_INET, SOCK_STREAM)

No need to attach server
name, port

→ clientSocket.send(sentence)

Example app: TCP server

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(("",serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

create TCP welcoming
socket →

server begins listening for
incoming TCP requests →

loop forever →

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

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

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

Socket API in C Programming Language

- ❖ What is a socket?
 - The **point** where a **local application process** attaches to the **network**
 - An **interface** between an **application** and the **network**
 - An application creates the socket
- ❖ The interface defines operations for
 - Creating a socket
 - Attaching a socket to the network
 - Sending and receiving messages through the socket
 - Closing the socket

Socket

- ❖ **Socket Family**
 - PF_INET denotes the Internet family
 - PF_UNIX denotes the Unix pipe facility
 - PF_PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack)
- ❖ **Socket Type**
 - SOCK_STREAM is used to denote a byte stream
 - SOCK_DGRAM is an alternative that denotes a message oriented service, such as that provided by UDP

Creating a Socket

```
int sockfd = socket(address_family, type,  
protocol);
```

- ❖ The socket number returned is the socket descriptor for the newly created socket
- ❖ `int sockfd = socket (PF_INET, SOCK_STREAM, 0);`
- ❖ `int sockfd = socket (PF_INET, SOCK_DGRAM, 0);`

The combination of `PF_INET` and `SOCK_STREAM` implies TCP

Client-Serve Model with TCP

Server

- Passive open
- Prepares to accept connection, does not actually establish a connection

Server invokes

```
int bind (int socket, struct sockaddr *address,  
          int addr_len)  
int listen (int socket, int backlog)  
int accept (int socket, struct sockaddr  
*address,  
            int *addr_len)
```

Client-Serve Model with TCP

Bind

- Binds the newly created socket to the specified address i.e. the network address of the local participant (the server)
- Address is a data structure which combines IP and port

Listen

- Defines how many connections can be pending on the specified socket

Client-Serve Model with TCP

Accept

- Carries out the passive open
- Blocking operation
 - **Does not return until a remote participant has established a connection**
 - When it does, it returns a new socket that corresponds to the new established connection and the address argument contains the remote participant's address

Client-Serve Model with TCP

Client

- Application performs active open
- It says who it wants to communicate with

Client invokes

```
int connect (int socket, struct sockaddr  
*address,  
             int addr_len)
```

Connect

- Does not return until TCP has successfully established a connection at which application is free to begin sending data
- Address contains remote machine's address

Client-Serve Model with TCP

In practice

- The client usually specifies only remote participant's address and let's the system fill in the local information
- Whereas a server usually listens for messages on a well-known port
- A client does not care which port it uses for itself, the OS simply selects an unused one

Client-Serve Model with TCP

Once a **connection is established**, the application process invokes two operation

```
int send (int socket, char *msg, int msg_len,  
          int flags)
```

```
int recv (int socket, char *buff, int buff_len,  
          int  
flags)
```

Example Application: Client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_LINE 256

int main(int argc, char * argv[])
{
    FILE *fp;
    struct hostent *hp;
    struct sockaddr_in sin;
    char *host;
    char buf[MAX_LINE];
    int s;
    int len;
    if (argc==2) {
        host = argv[1];
    }
    else {
        fprintf(stderr, "usage: simplex-talk host\n");
        exit(1);
    }
```

Example Application: Client

```
/* translate host name into peer's IP address */
hp = gethostbyname(host);
if (!hp) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
}
/* build address data structure */
bzero((char *)&sin, sizeof(sin));
sin.sin_family = AF_INET; /* Internet Address*/
bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
sin.sin_port = htons(SERVER_PORT);
/* active open PF_INET is protocol family*/
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
}
if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
}
/* main loop: get and send lines of text */
while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
}
```

Example Application: Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256

int main()
{
    struct sockaddr_in sin;
    char buf[MAX_LINE];
    int len;
    int s, new_s;
    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(SERVER_PORT);

    /* setup passive open */
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("simplex-talk: socket");
        exit(1);
    }
```

Example Application: Server

```
if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("simplex-talk: bind");
    exit(1);
}
listen(s, MAX_PENDING);
/* wait for connection, then receive and print text */
while(1) {
    if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
        perror("simplex-talk: accept");
        exit(1);
    }
    while (len = recv(new_s, buf, sizeof(buf), 0))
        fputs(buf, stdout);
    close(new_s);
}
}
```

Chapter 2: summary

our study of network apps now complete!

- ❖ application architectures
 - client-server
 - P2P
- ❖ application service requirements:
 - reliability, bandwidth, delay
- ❖ Internet transport service model
 - connection-oriented, reliable: TCP
 - unreliable, datagrams: UDP
- ❖ specific protocols:
 - HTTP
 - FTP
 - SMTP, POP, IMAP
 - DNS
 - P2P: BitTorrent, DHT
- ❖ socket programming: TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

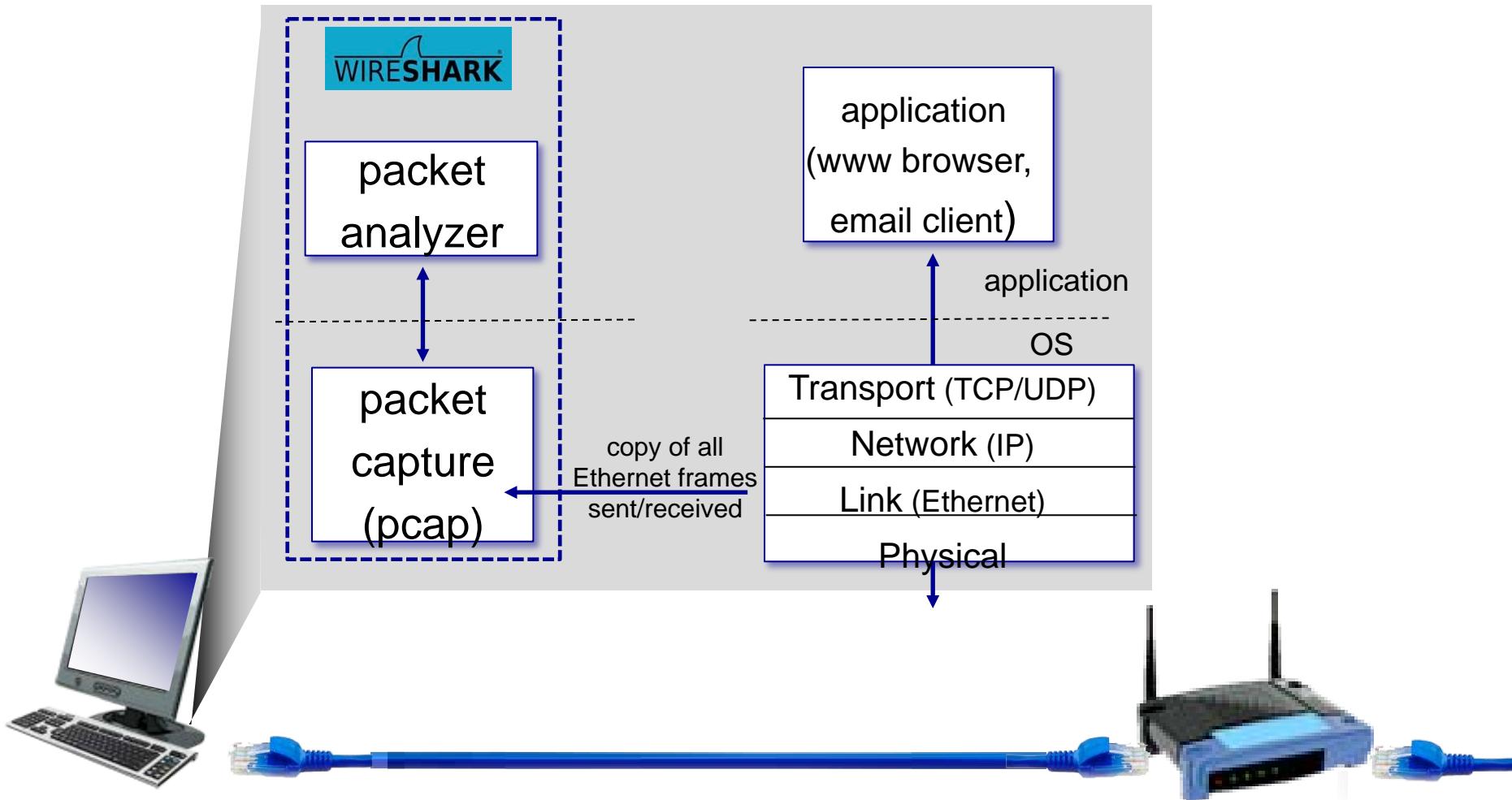
- ❖ typical request/reply message exchange:
 - client requests info or service
 - server responds with data, status code
- ❖ message formats:
 - headers: fields giving info about data
 - data: info being communicated

important themes:

- ❖ control vs. data msgs
 - in-band, out-of-band
- ❖ centralized vs. decentralized
- ❖ stateless vs. stateful
- ❖ reliable vs. unreliable msg transfer
- ❖ “complexity at network edge”

Chapter 2

Additional Slides



Chapter 3

Transport Layer

© The materials in these notes are adapted from *Computer Networking: A Top Down Approach*, 6th edition, by Jim Kurose, Keith Ross

Chapter 3: Transport Layer

our goals:

- ❖ understand principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
 - Error control
- ❖ learn about Internet transport layer protocols:
 - UDP: connectionless transport
 - TCP: connection-oriented reliable transport
 - TCP congestion control

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

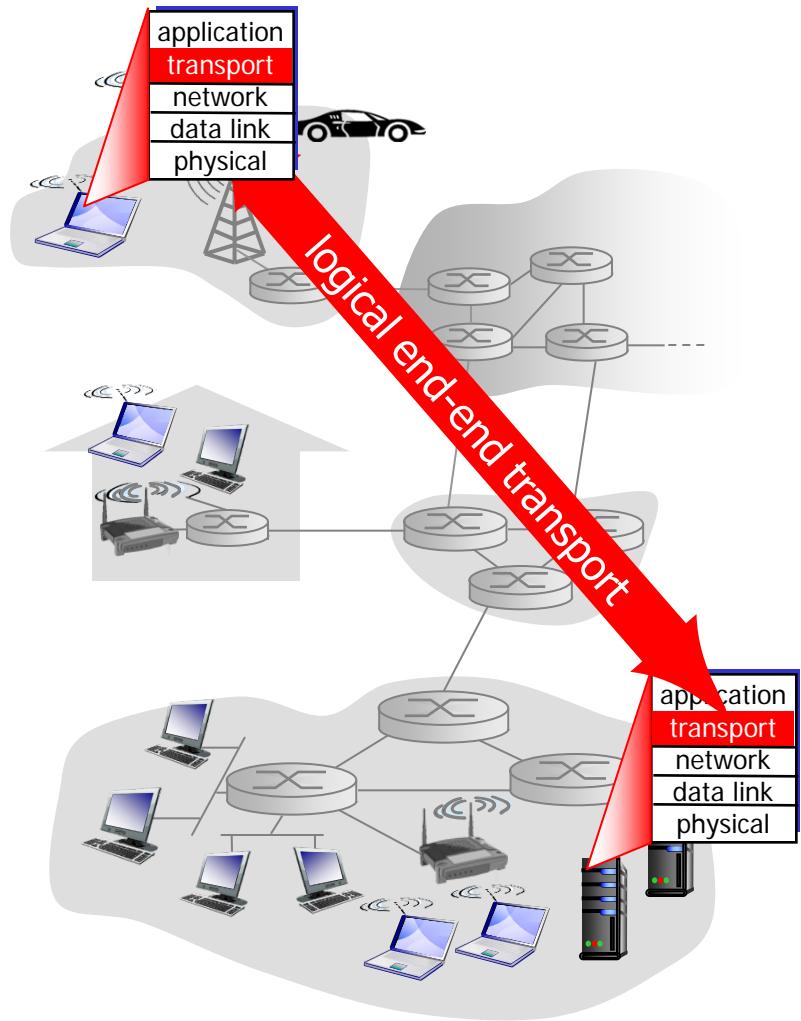
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

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



Transport vs. network layer

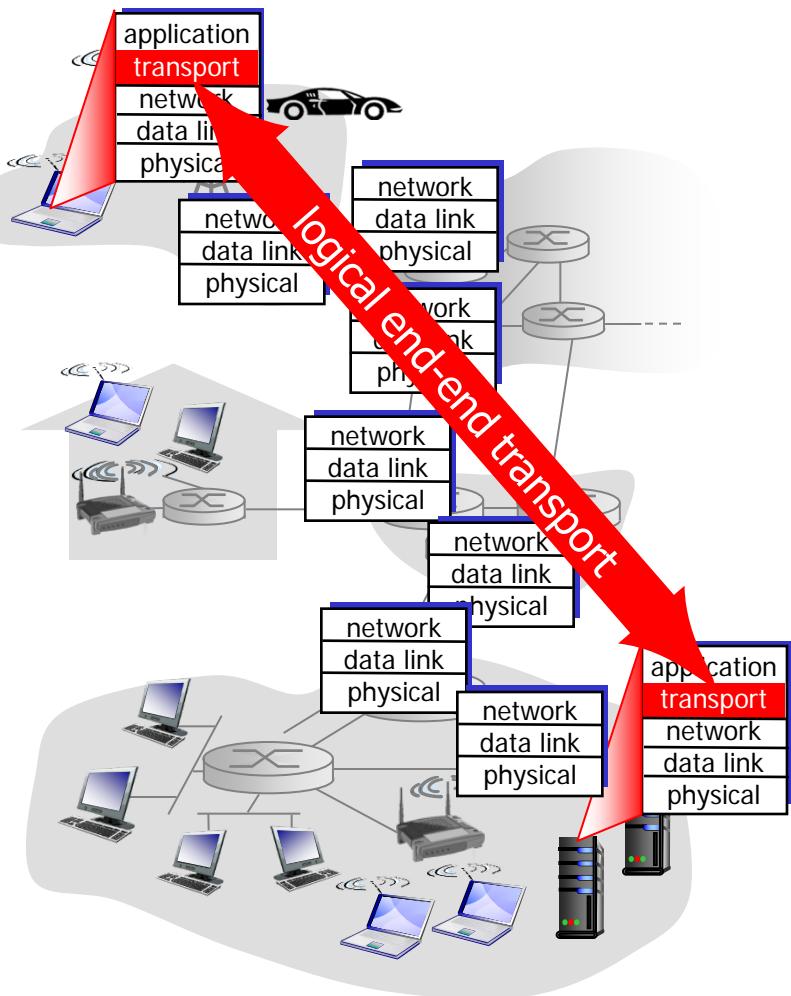
- ❖ *network layer*: logical communication between **hosts**
- ❖ *transport layer*: logical communication between **processes**
 - relies on, enhances, network layer services

household analogy:

- 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

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



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

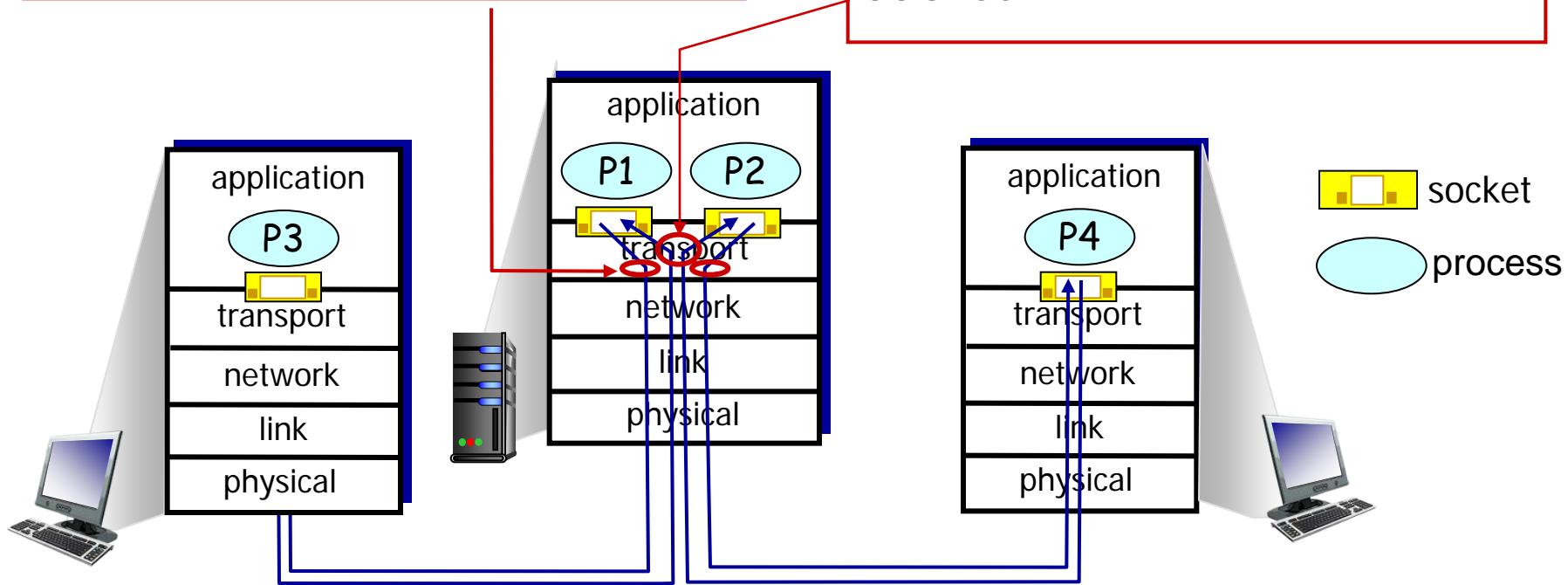
Multiplexing/demultiplexing

multiplexing at sender:

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

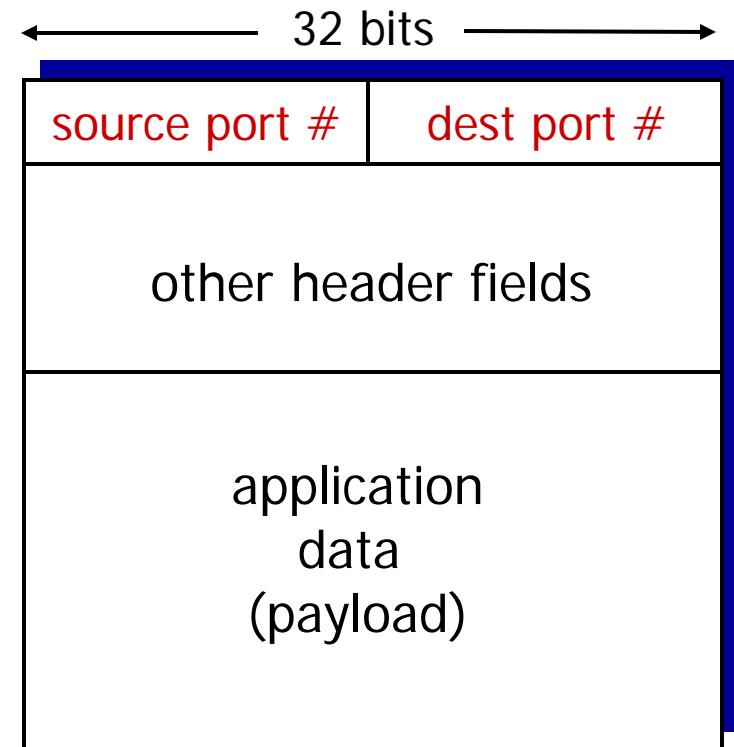
demultiplexing at receiver:

use header info to deliver received segments to correct socket



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

Connectionless demultiplexing

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

```
DatagramSocket mySocket1  
= new DatagramSocket(1234);
```

- ❖ *recall:* when creating datagram to send into UDP socket, must specify
 - destination IP address
 - destination port #

-
- ❖ when host receives UDP segment:

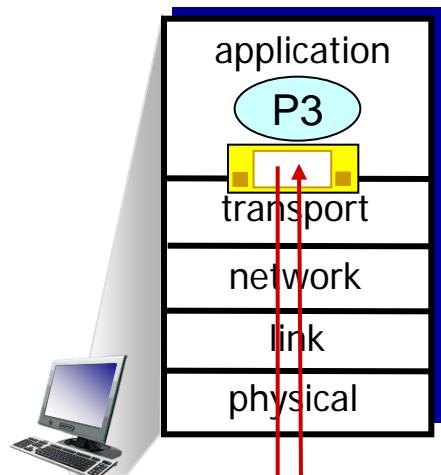
- checks destination port # in segment
- directs UDP segment to socket with that port #



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

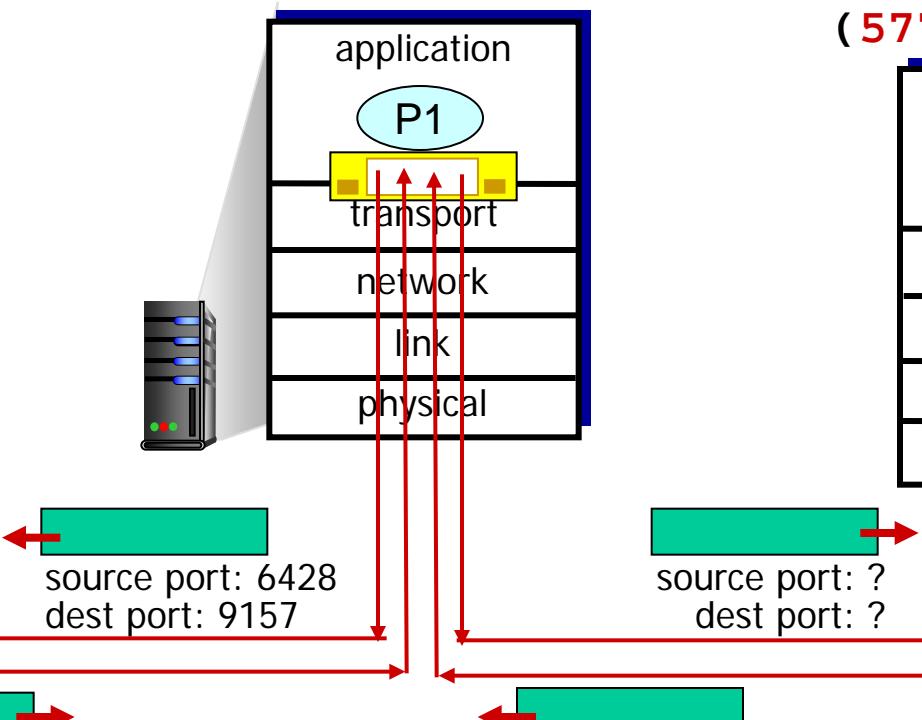
Connectionless demux: example

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

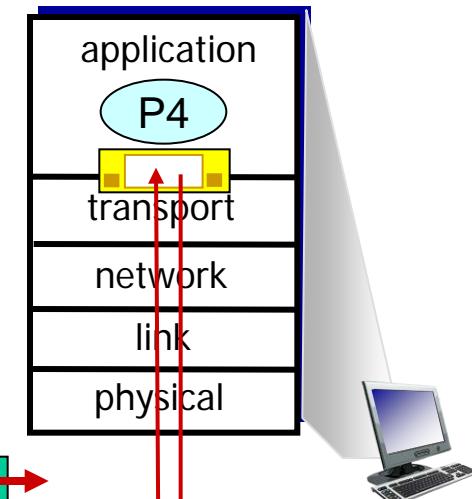


DatagramSocket

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



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



source port: 9157
dest port: 6428

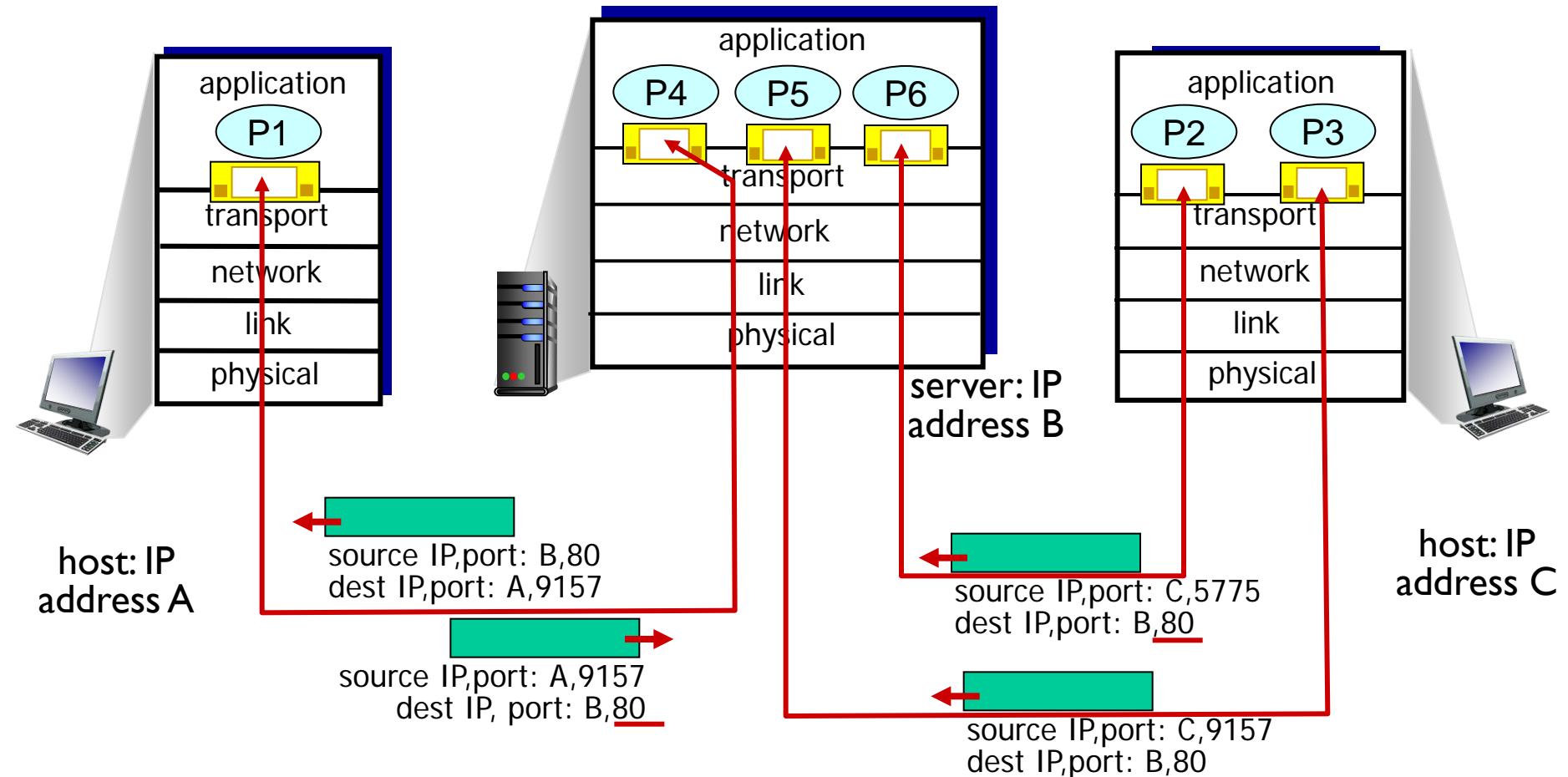
source port: ?
dest port: ?

source port: ?
dest port: ?

Connection-oriented demux

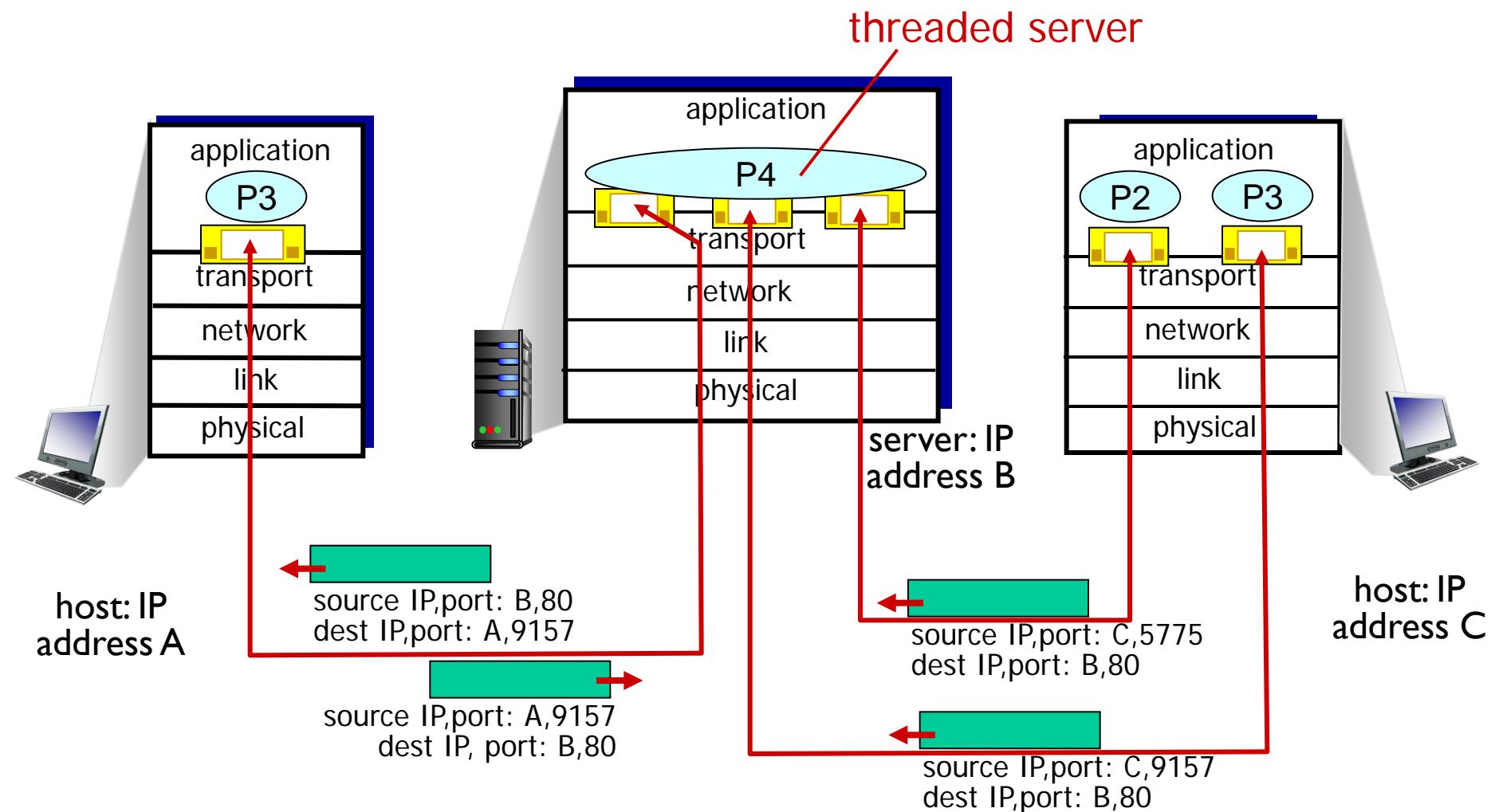
- ❖ TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- ❖ demux: receiver uses all four values to direct segment to appropriate socket
- ❖ server host may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple
- ❖ web servers have different sockets for each connecting client
 - non-persistent HTTP will have different socket for each request

Connection-oriented demux: example



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

Connection-oriented demux: example



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

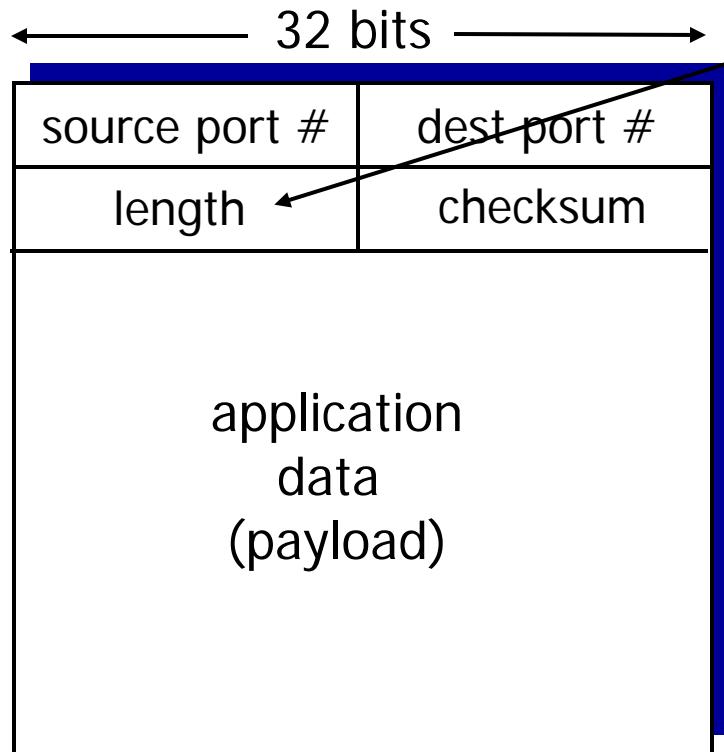
3.6 principles of congestion control

3.7 TCP congestion control

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!

UDP: segment header



UDP segment format

length, in bytes of
UDP segment,
including header

why is there a UDP?

- ❖ no connection establishment (which can add delay)
- ❖ simple: no connection state at sender, receiver
- ❖ small header size
- ❖ no congestion control: UDP can blast away as fast as desired

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
....

Internet checksum: example

example: add two 16-bit integers

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

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

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

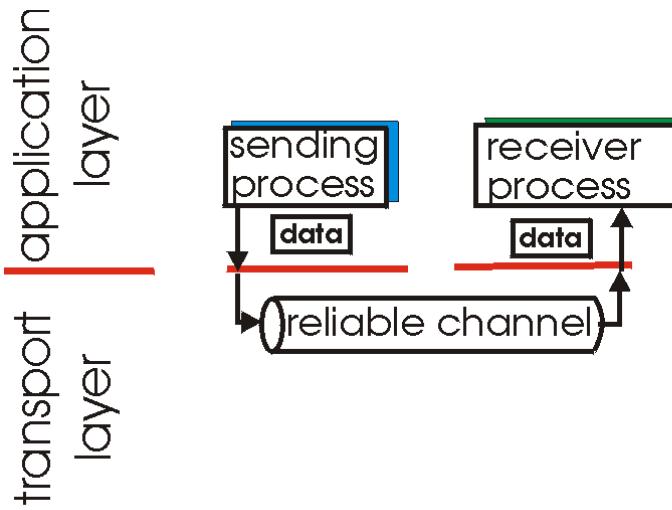
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

Principles of reliable data transfer

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

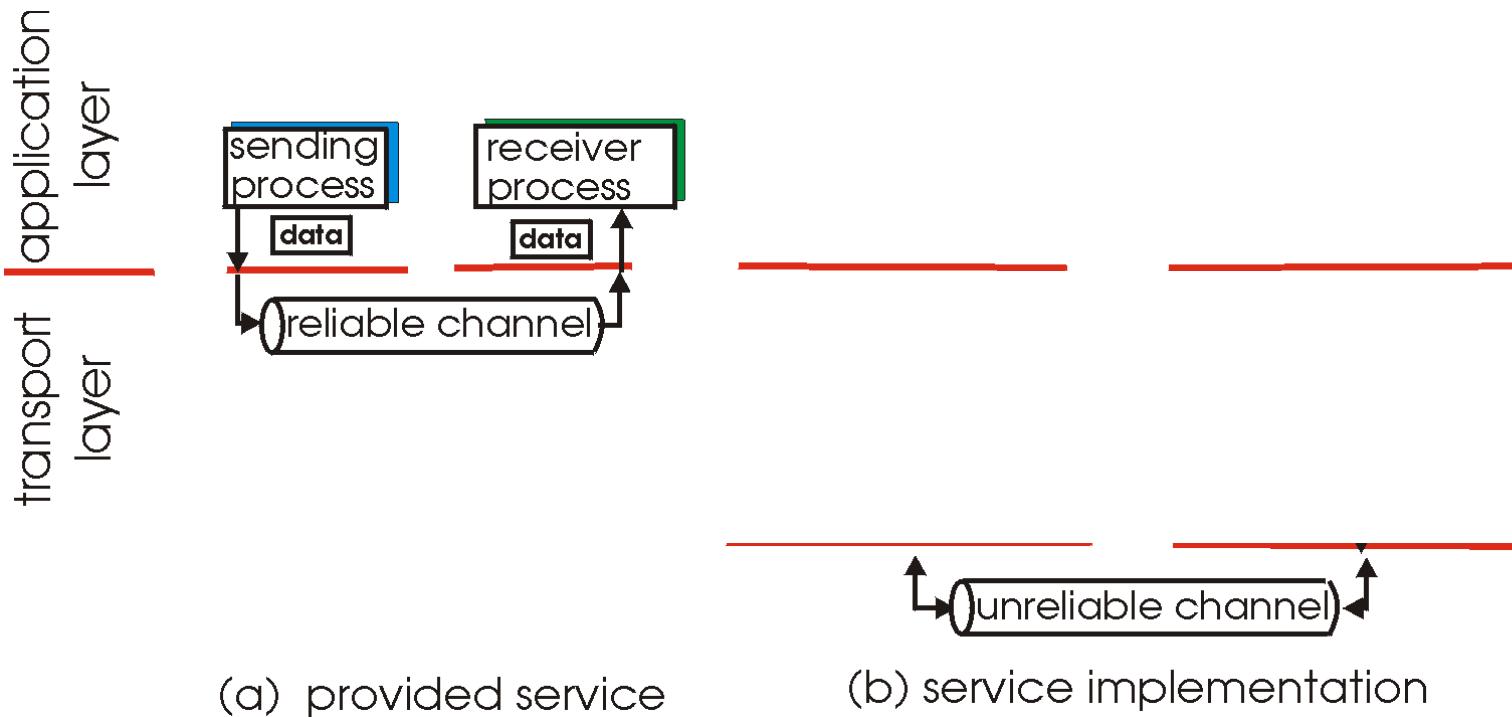


(a) provided service

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

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)

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!

rdt1.0: reliable transfer over a reliable channel

- ❖ underlying channel perfectly reliable
 - no bit errors
 - no loss of packets

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?*

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

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

rdt2.1: discussion

sender:

- ❖ seq # added to pkt
- ❖ two seq. #'s (0,1) will suffice. Why?
- ❖ must check if received ACK/NAK corrupted

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

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*

rdt3.0: channels with errors and loss

new assumption:

underlying channel can
also lose packets
(data, ACKs)

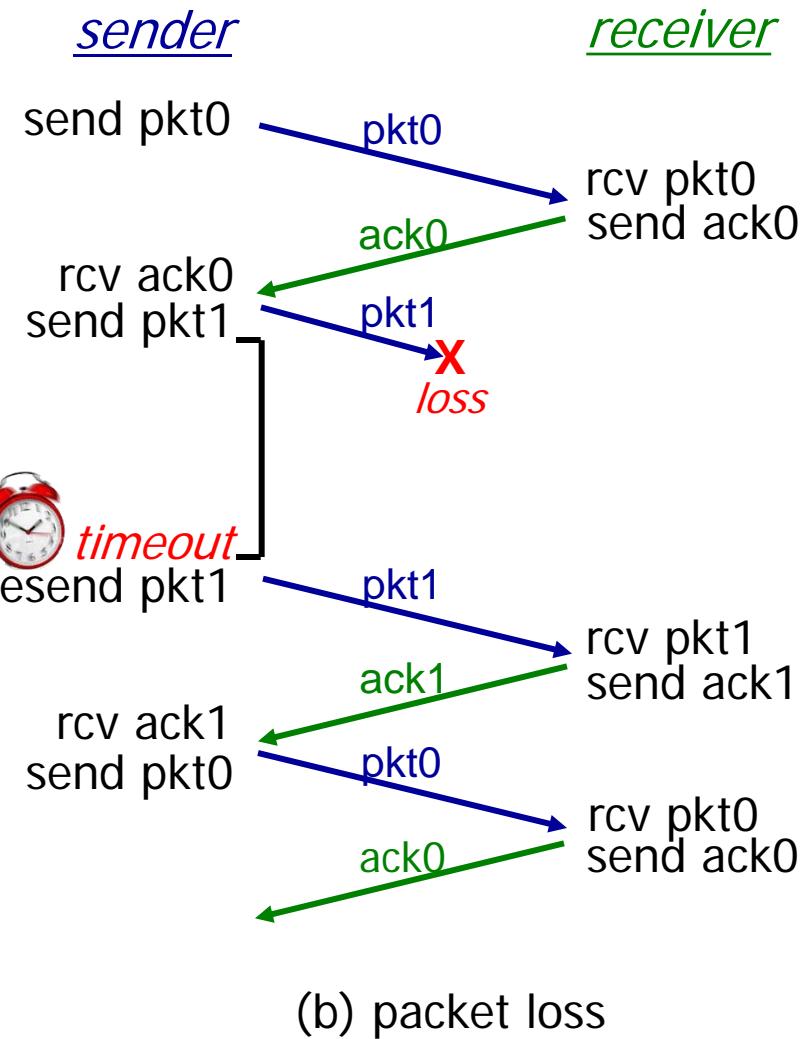
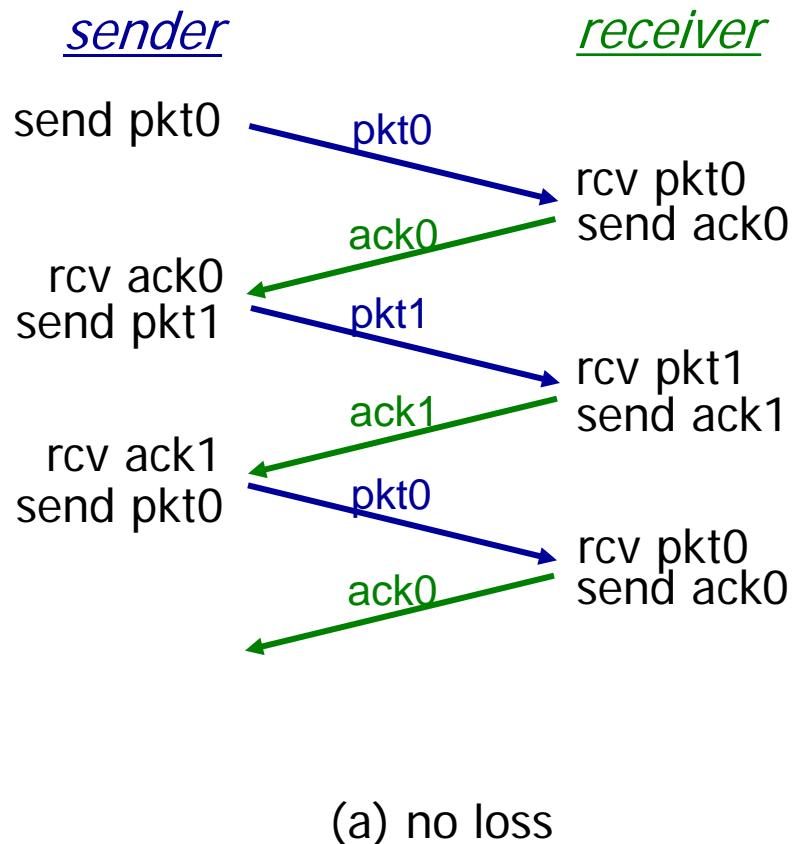
- checksum, seq. #,
ACKs, retransmissions
will be of help ... but
not enough

approach: sender waits

“reasonable” amount of
time for ACK

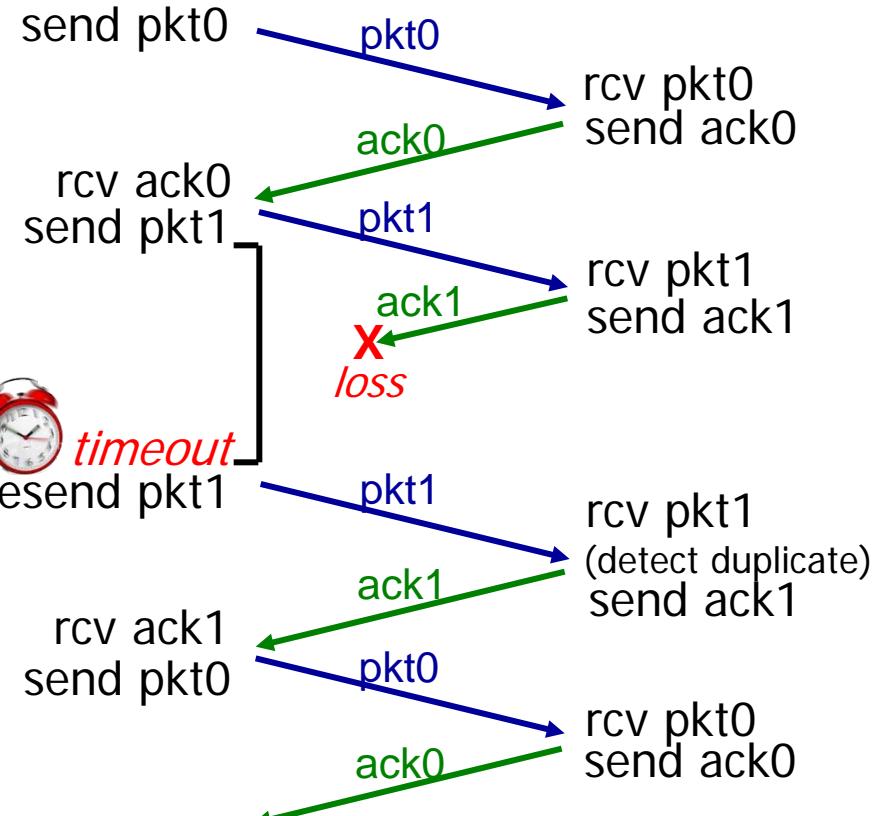
- ❖ retransmits if no ACK
received in this time
- ❖ if pkt (or ACK) just delayed
(not lost):
 - retransmission will be
duplicate, but seq. #'s
already handles this
 - receiver must specify seq
of pkt being ACKed
- ❖ requires countdown timer

rdt3.0 in action



rdt3.0 in action

sender



(c) ACK loss

sender

send pkt0

rcv ack0
send pkt1

timeout
resend pkt1

rcv ack1
send pkt0

rcv ack1
send pkt0

rcv ack1
send pkt0

rcv ack0
send pkt0

rcv ack0
send pkt0

receiver

rcv pkt0
send ack0

rcv pkt1
send ack1

rcv pkt1
(detect duplicate)
send ack1

rcv pkt0
send ack0

rcv pkt0
(detect duplicate)
send ack0

alarm clock
timeout

rcv ack1
send pkt0

rcv ack1
send pkt0

rcv ack1
send pkt0

rcv ack0
send pkt0

rcv ack0
send pkt0

(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:

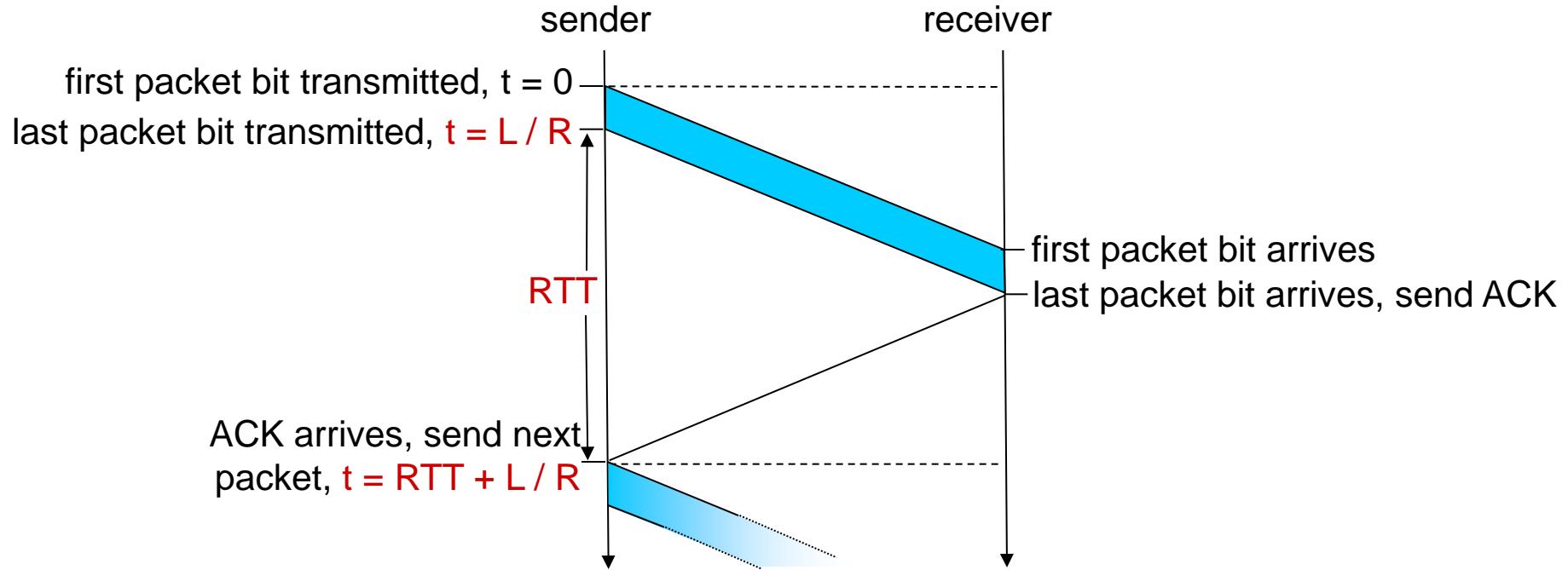
$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

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

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

- if RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link
- ❖ network protocol limits use of physical resources!

rdt3.0: stop-and-wait operation



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

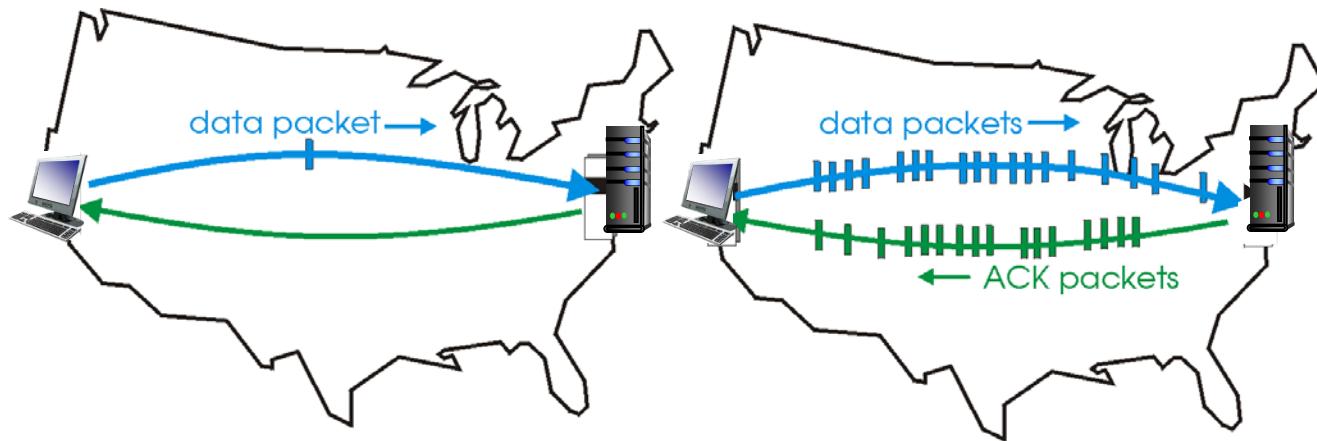
Learn from the rdt versions

- ❖ rdt 1.0: based on a completely reliable channel
- ❖ rdt 2.0: error detection (checksum) with ACK/NAK
- ❖ rdt 2.1: avoid duplicate packets (retransmission) caused by corrupted ACK/NAK, with sequence numbers
- ❖ rdt 2.2: ACK only/NAK free
- ❖ rdt 3.0: packet/ACK loss, with countdown timer

Pipelined protocols

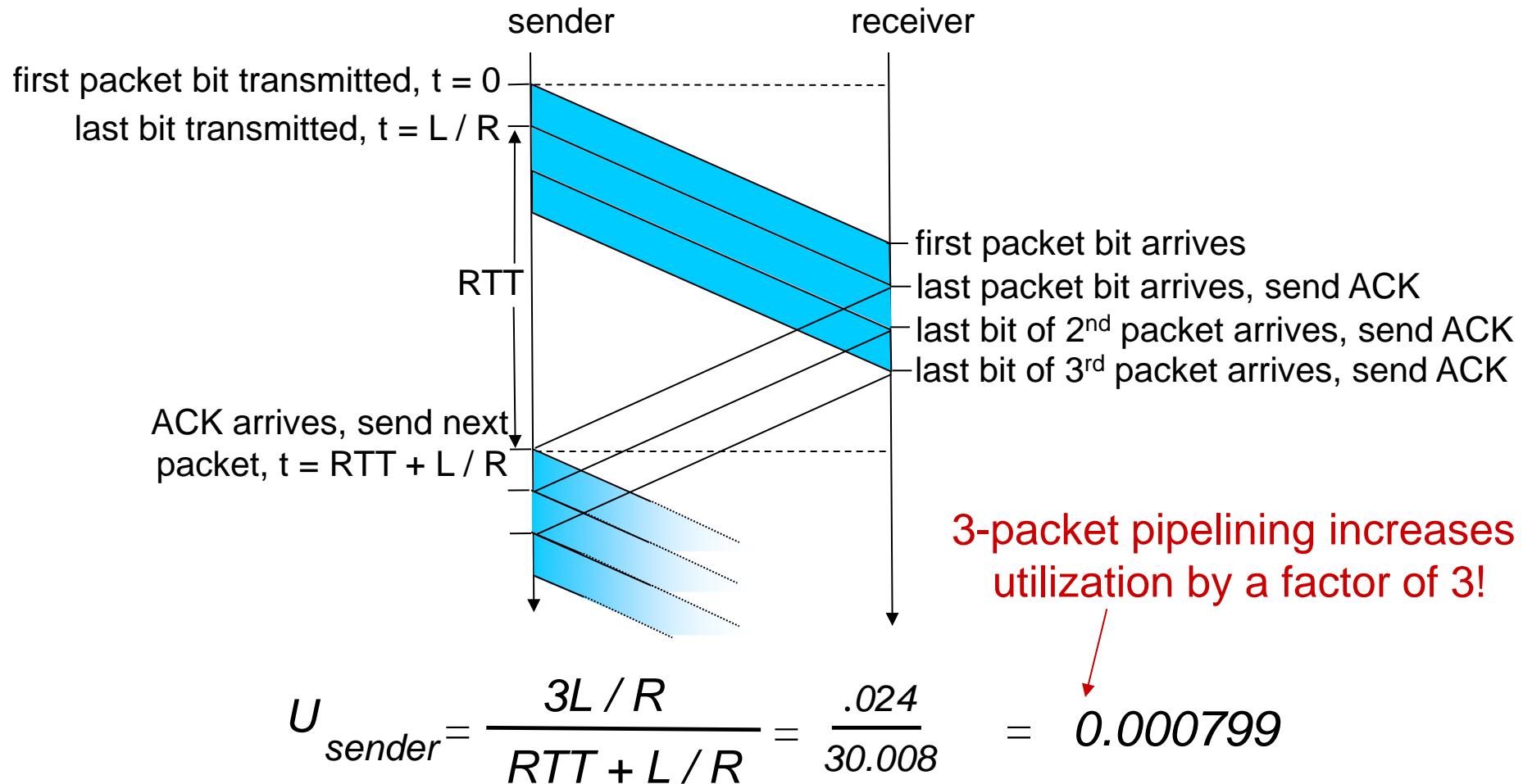
pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

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



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

Pipelining: increased utilization



Pipelined protocols: overview

Go-back-N:

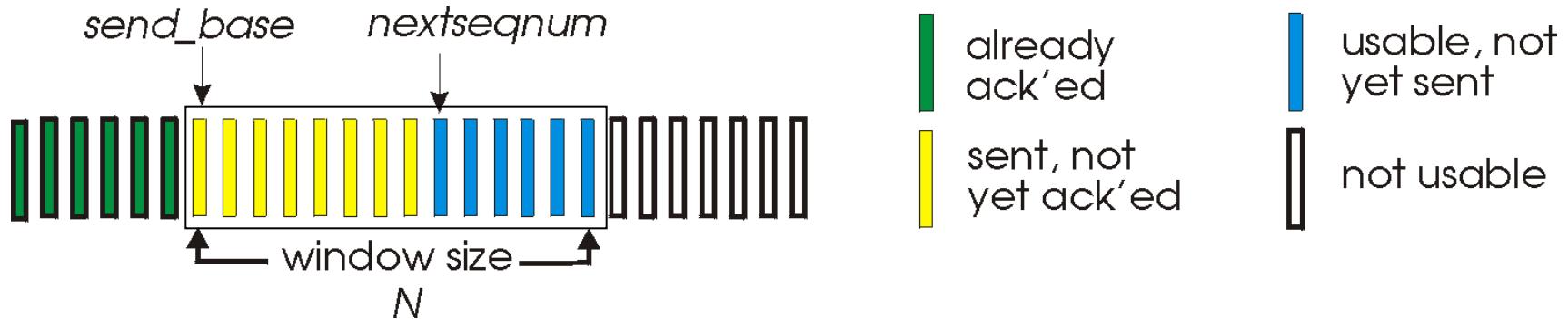
- ❖ sender can have up to N unacked packets in pipeline
- ❖ receiver only sends *cumulative ack*
 - doesn't ack packet if there's a gap
- ❖ sender has timer for oldest unacked packet
 - when timer expires, retransmit *all* unacked packets

Selective Repeat:

- ❖ sender can have up to N unacked 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

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

GBN: receiver

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 #

GBN in action

sender window (N=4)

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

rcv ack0, send pkt4
rcv ack1, send pkt5

ignore duplicate ACK



pkt 2 timeout

send pkt2
send pkt3
send pkt4
send pkt5

receiver

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, discard,
(re)send ack1

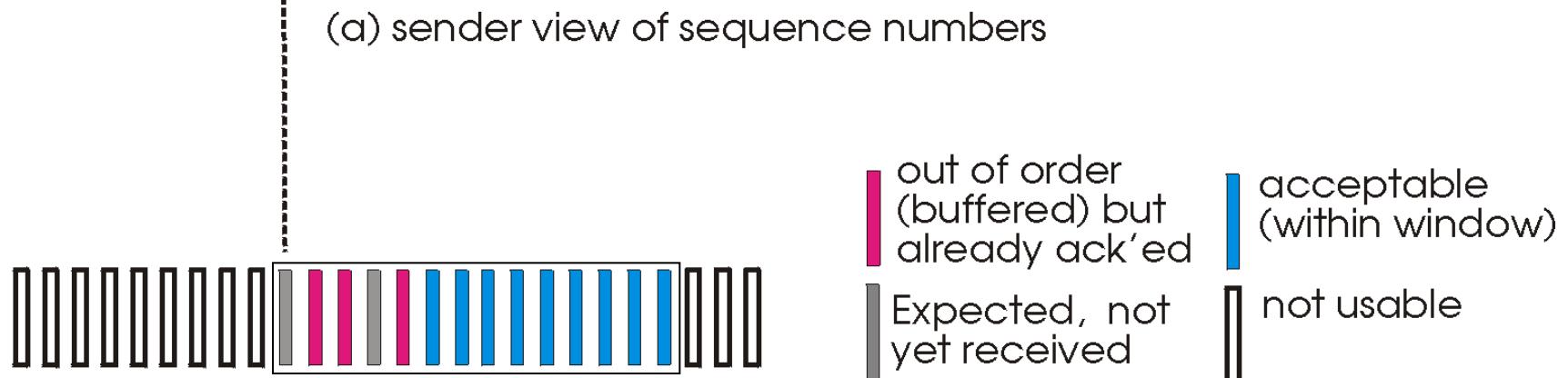
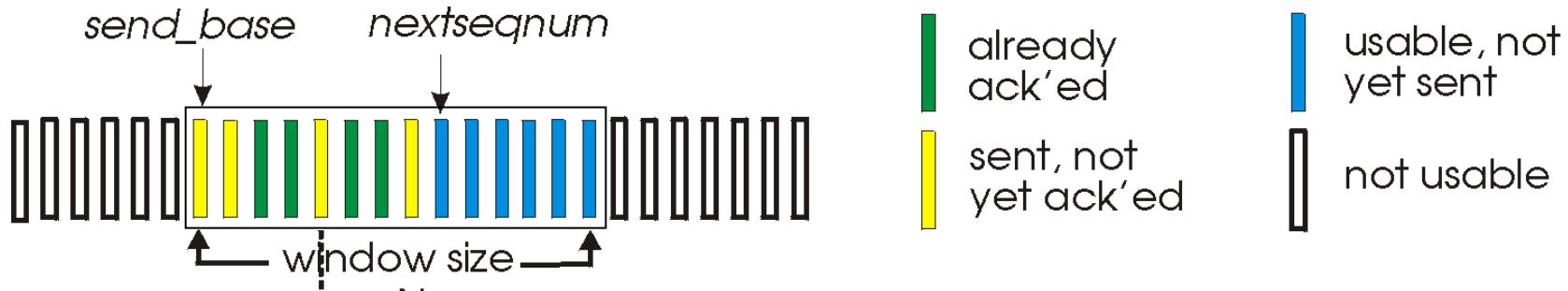
receive pkt4, discard,
(re)send ack1
receive pkt5, discard,
(re)send ack1

rcv pkt2, deliver, send ack2
rcv pkt3, deliver, send ack3
rcv pkt4, deliver, send ack4
rcv pkt5, deliver, send ack5

Selective repeat

- ❖ receiver *individually* acknowledges all correctly received pkts
 - buffers pkts, as needed, for eventual in-order delivery to upper layer
- ❖ sender only resends pkts for which ACK not received
 - sender timer for each unACKed pkt
- ❖ sender window
 - N consecutive seq #'s
 - limits seq #'s of sent, unACKed pkts

Selective repeat: sender, receiver windows



(b) receiver view of sequence numbers

Selective repeat

sender

data from above:

- ❖ if next available seq # in window, send pkt

timeout(n):

- ❖ resend pkt n, restart timer

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

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

receiver

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

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

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

- ❖ ACK(n)

otherwise:

- ❖ ignore

Selective repeat in action

sender window (N=4)

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

receiver

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, buffer,
send ack3

receive pkt4, buffer,
send ack4
receive pkt5, buffer,
send ack5

0 1 2 3 4 5 6 7 8 rcv ack0, send pkt4
0 1 2 3 4 5 6 7 8 rcv ack1, send pkt5

record ack3 arrived



pkt 2 timeout

send pkt2

record ack4 arrived

record ack5 arrived

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

Q: what happens when ack2 arrives?

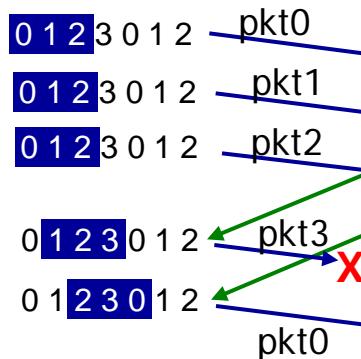
Selective repeat: dilemma

example:

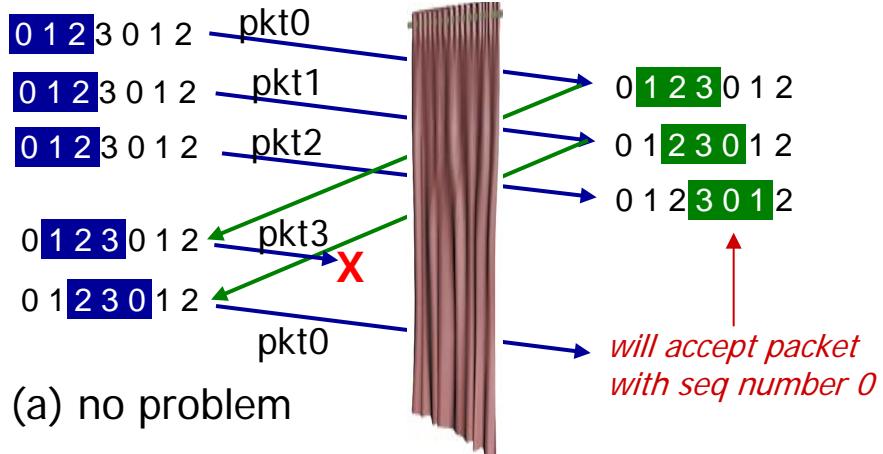
- ❖ seq #'s: 0, 1, 2, 3
- ❖ window size=3
- ❖ receiver sees no difference in two scenarios!
- ❖ duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?

sender window
(after receipt)

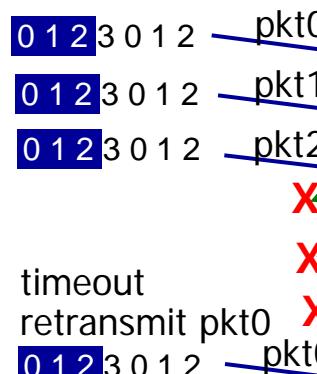


receiver window
(after receipt)



(a) no problem

*receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!*



(b) oops!

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

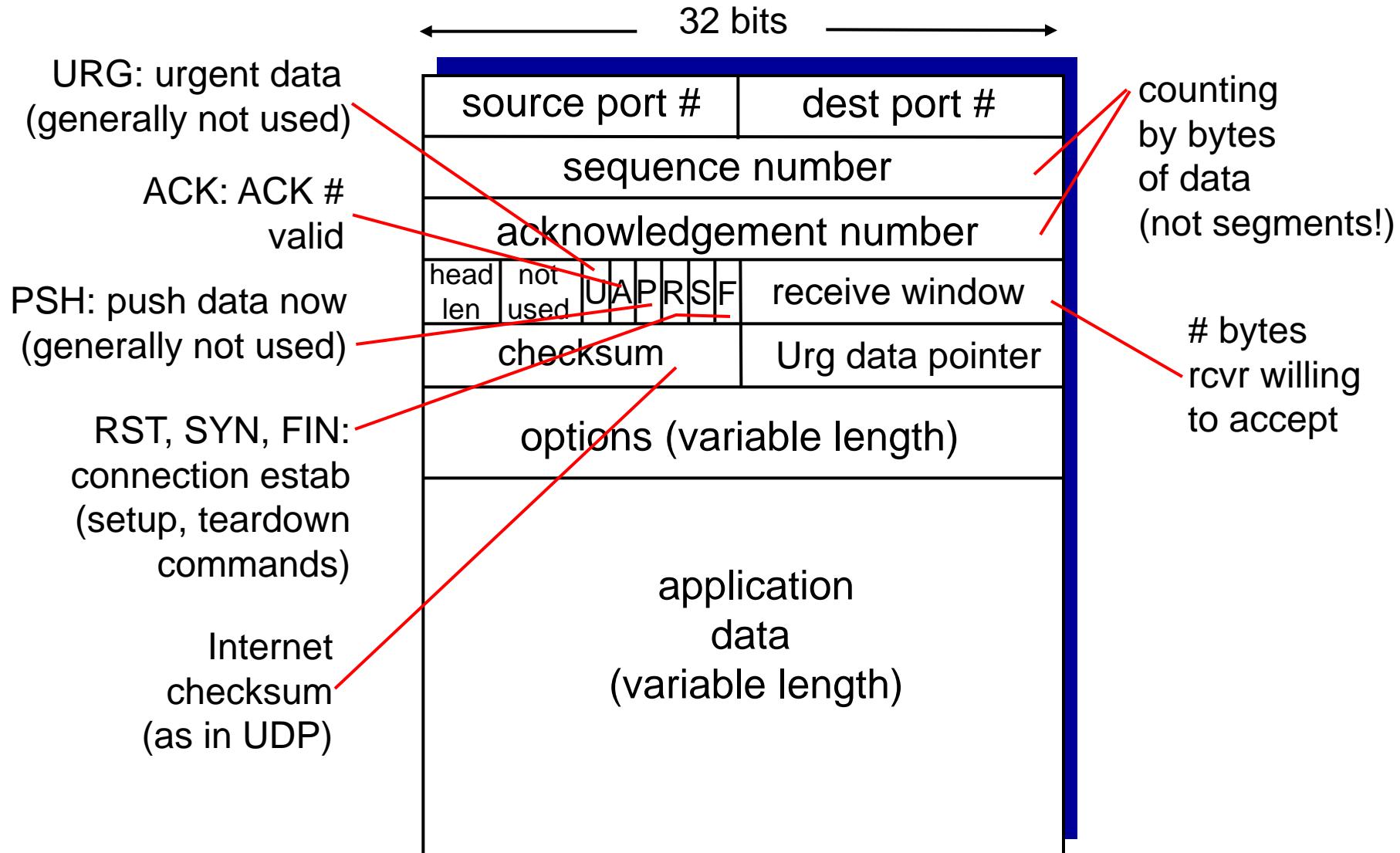
3.7 TCP congestion control

TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- ❖ **point-to-point:**
 - one sender, one receiver
- ❖ **reliable, in-order *byte stream*:**
 - 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
- ❖ **flow controlled:**
 - sender will not overwhelm receiver

TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

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

acknowledgements:

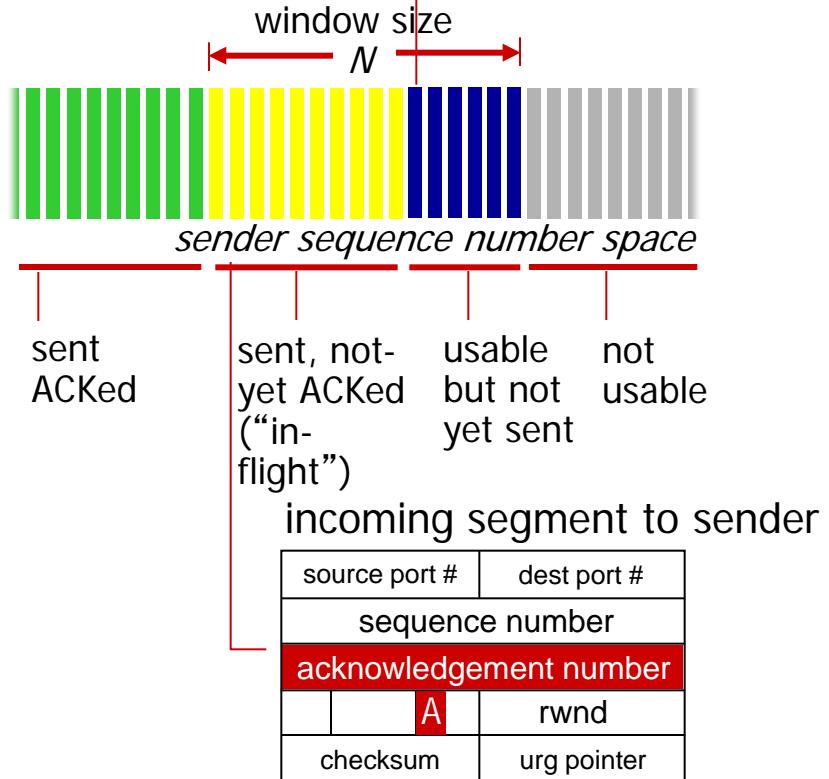
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

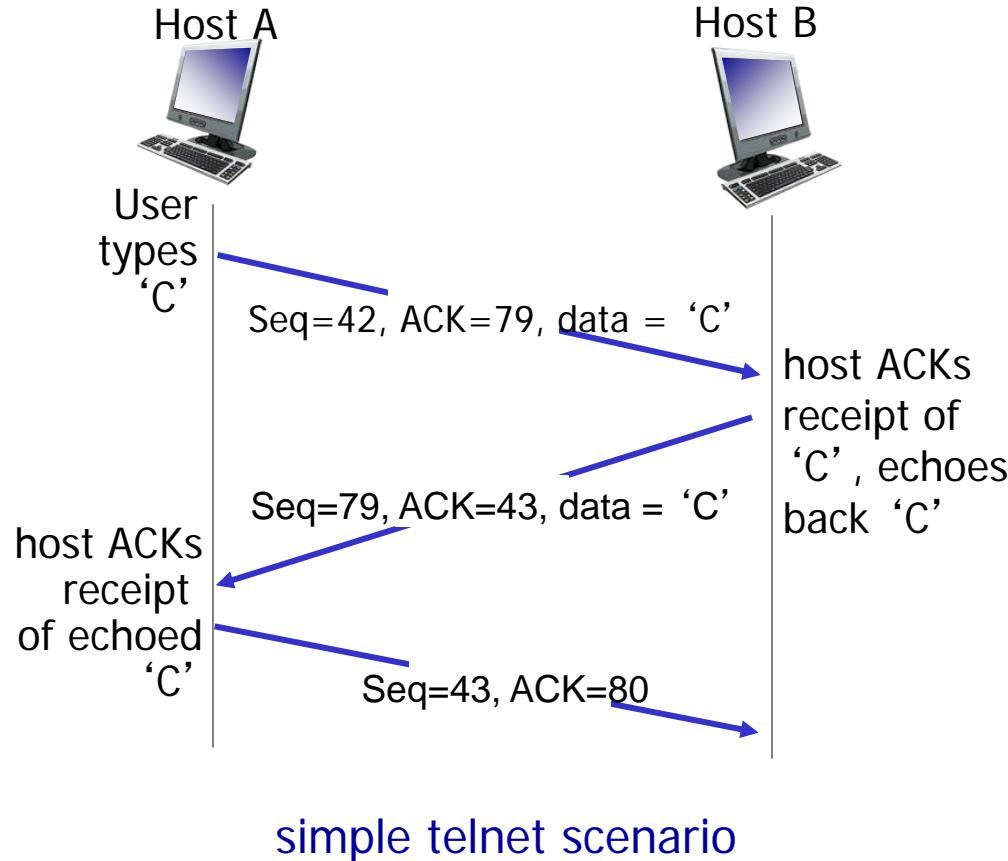
- A: TCP spec doesn’t say,
- up to implementor

outgoing segment from sender

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



TCP seq. numbers, ACKs



TCP round trip time, timeout

Q: how to set TCP timeout value?

- ❖ longer than RTT
 - but RTT varies
- ❖ too short: premature timeout, unnecessary retransmissions
- ❖ too long: slow reaction to segment loss

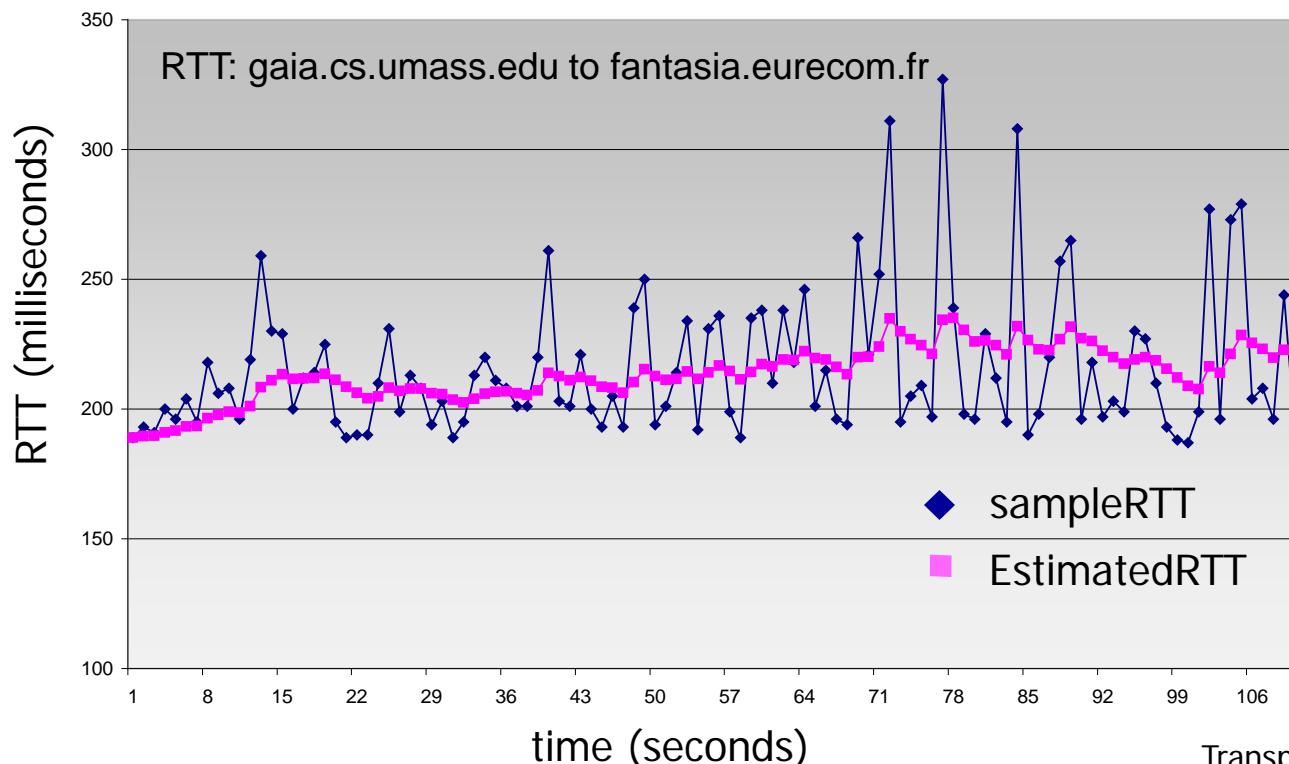
Q: how to estimate RTT?

- ❖ **SampleRTT**: measured time from segment transmission until ACK receipt
 - ignore retransmissions
- ❖ **SampleRTT** will vary, want estimated RTT “smoother”
 - average several *recent* measurements, not just current **SampleRTT**

TCP round trip time, timeout

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

- ❖ exponential weighted moving average
- ❖ influence of past sample decreases exponentially fast
- ❖ typical value: $\alpha = 0.125$



TCP round trip time, timeout

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

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

(typically, $\beta = 0.25$)

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



↑
estimated RTT

↑
“safety margin”

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- **reliable data transfer**
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

TCP 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

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`

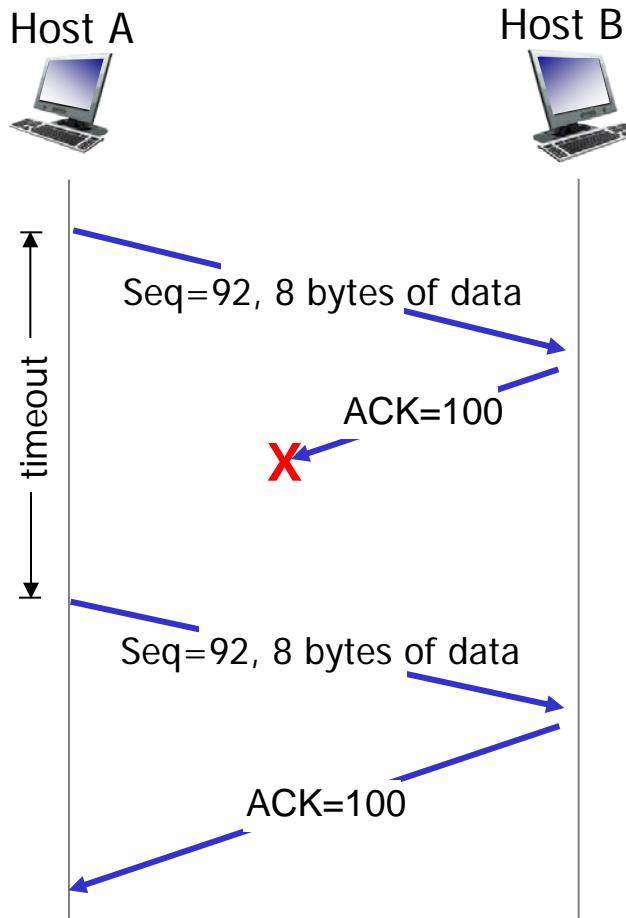
timeout:

- ❖ retransmit segment that caused timeout
- ❖ restart timer

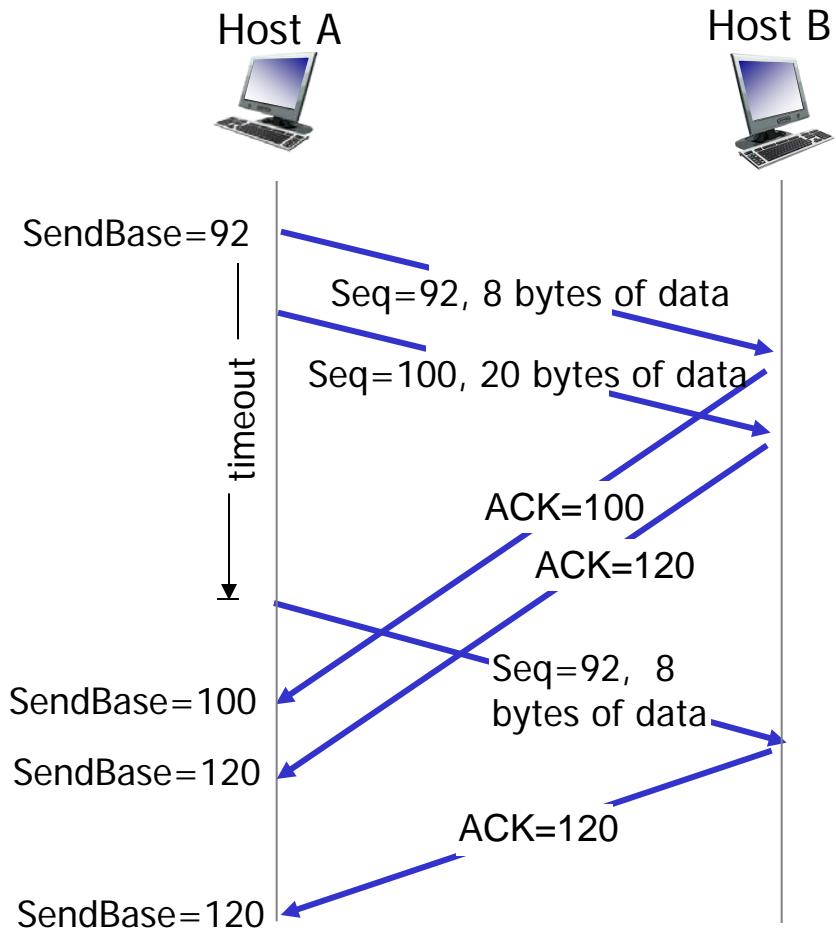
ack rcvd:

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

TCP: retransmission scenarios

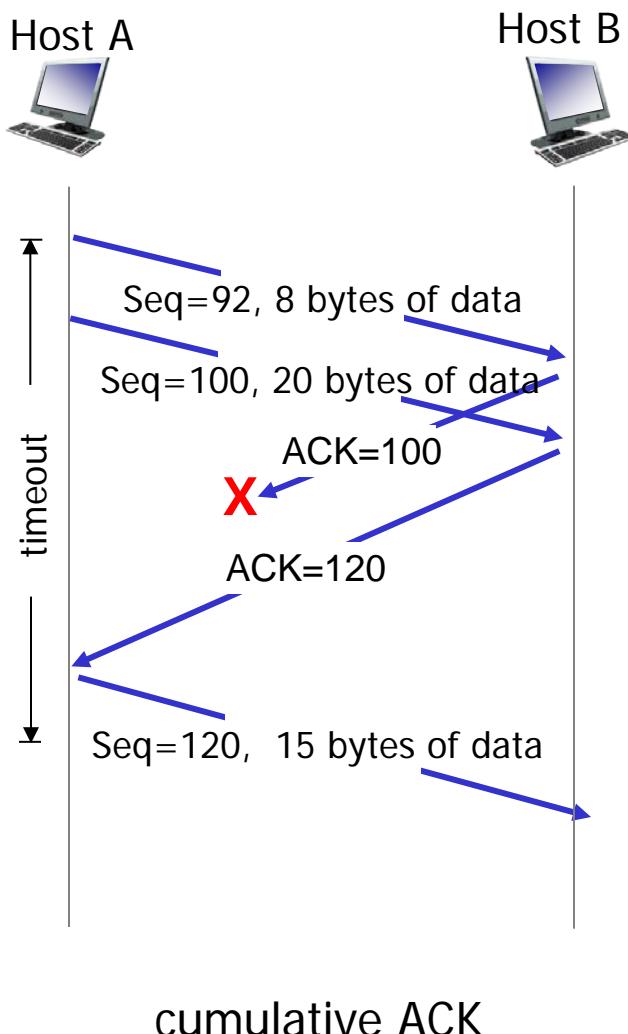


lost ACK scenario



premature timeout

TCP: retransmission scenarios



TCP ACK generation [RFC 1122, RFC 2581]

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

TCP fast retransmit

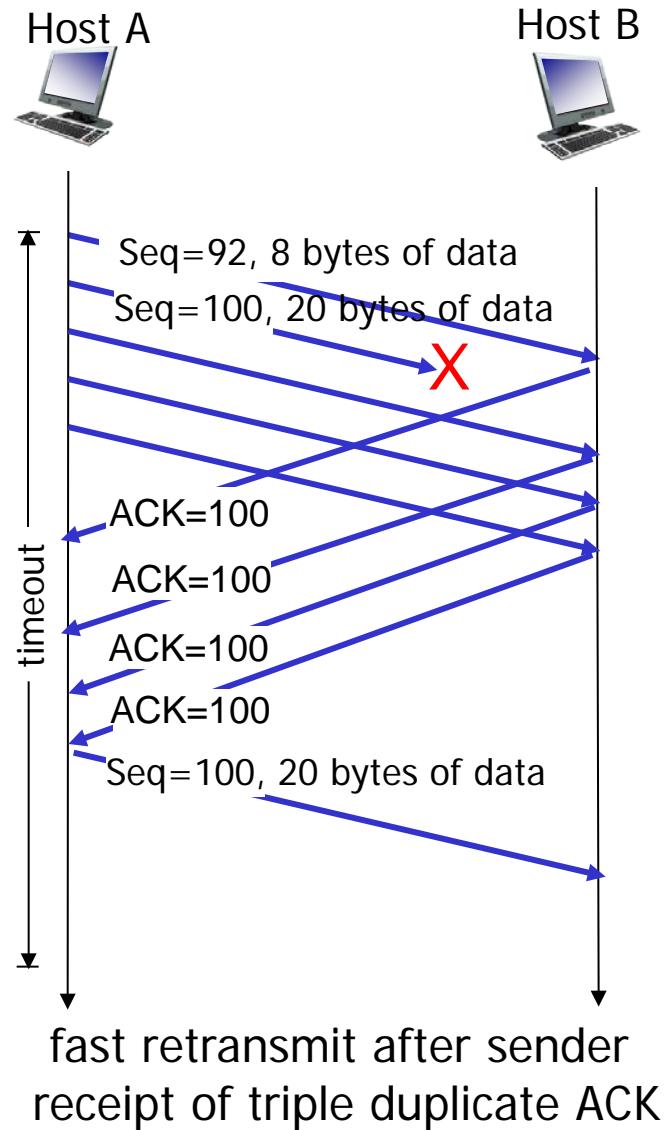
- ❖ time-out period often relatively long:
 - long delay before resending lost packet
- ❖ detect lost segments via duplicate ACKs.
 - sender often sends many segments back-to-back
 - if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

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

TCP fast retransmit



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- **flow control**
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

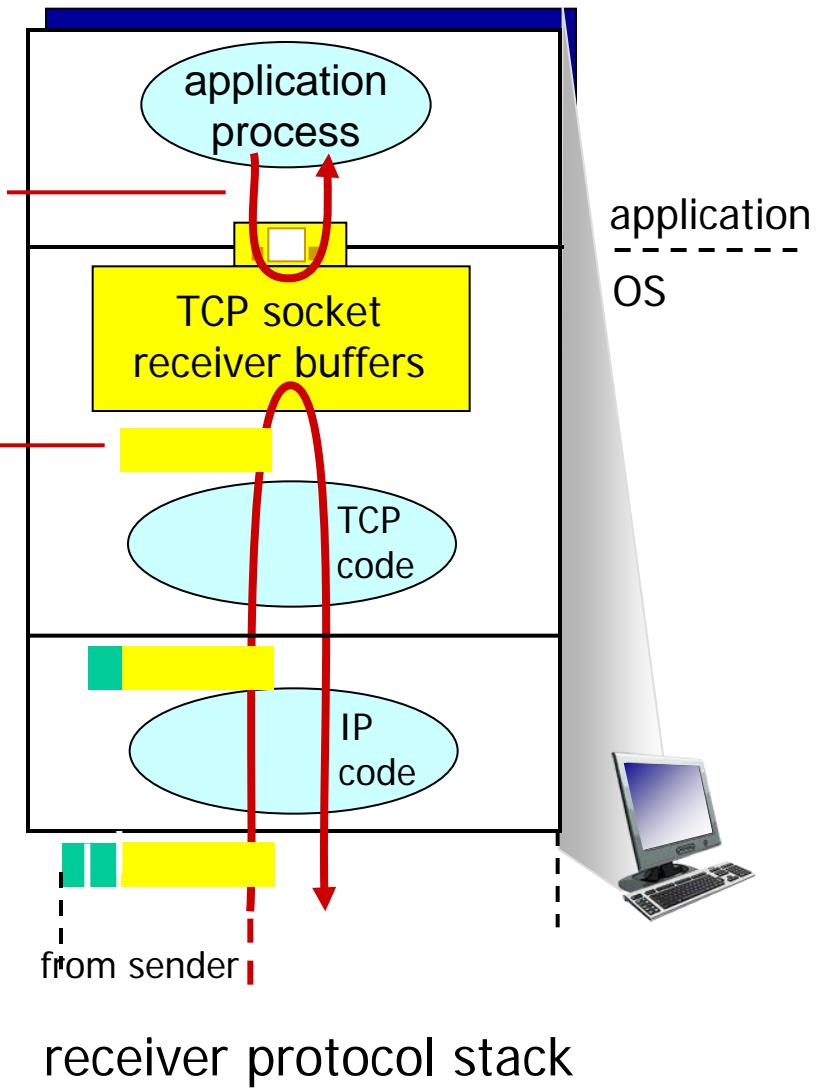
TCP flow control

flow control

receiver controls sender, so
sender won't overflow
receiver's buffer by transmitting
too much, too fast

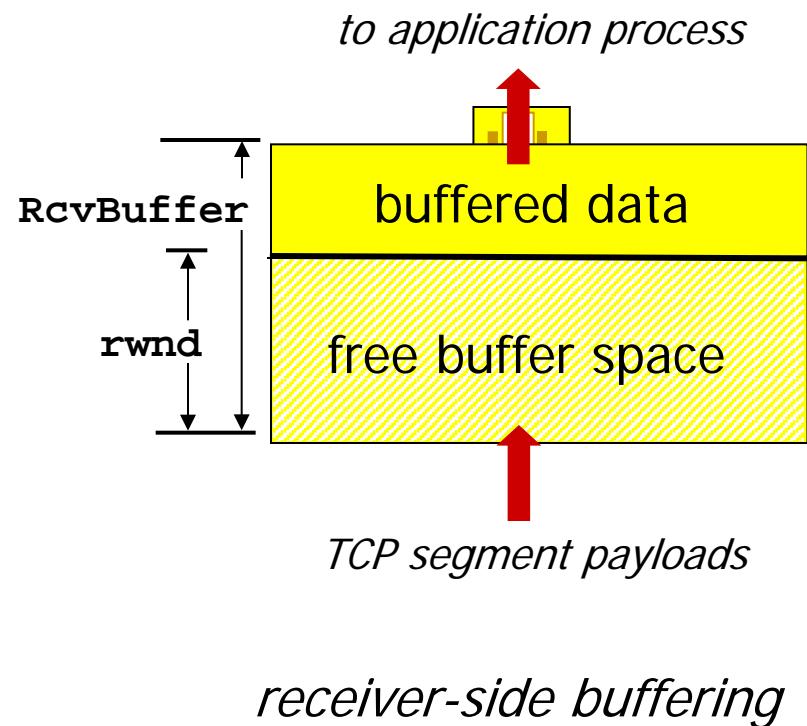
application may
remove data from
TCP socket buffers

... slower than TCP
receiver is delivering
(sender is sending)



TCP flow control

- ❖ receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - many operating systems autoadjust **RcvBuffer**
- ❖ sender limits amount of unacked (“in-flight”) data to receiver’s **rwnd** value
- ❖ guarantees receive buffer will not overflow



receiver-side buffering

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

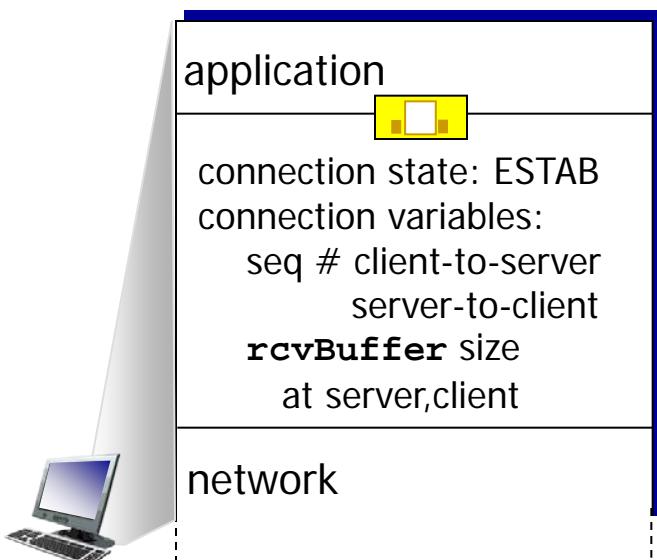
3.6 principles of congestion control

3.7 TCP congestion control

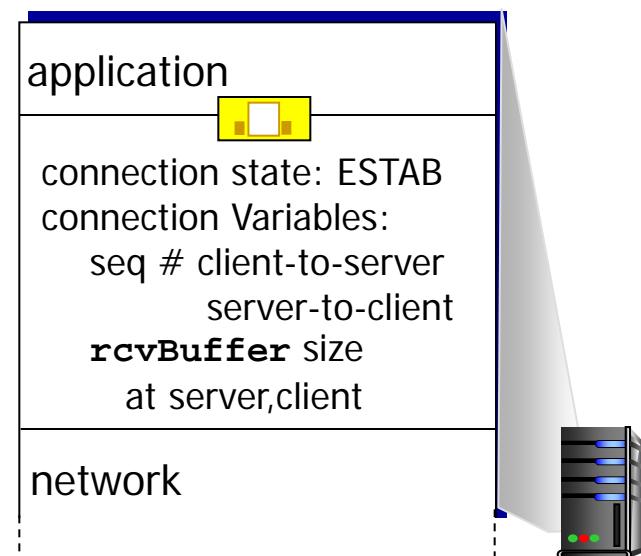
Connection Management

before exchanging data, sender/receiver “handshake”:

- ❖ agree to establish connection (each knowing the other willing to establish connection)
- ❖ agree on connection parameters



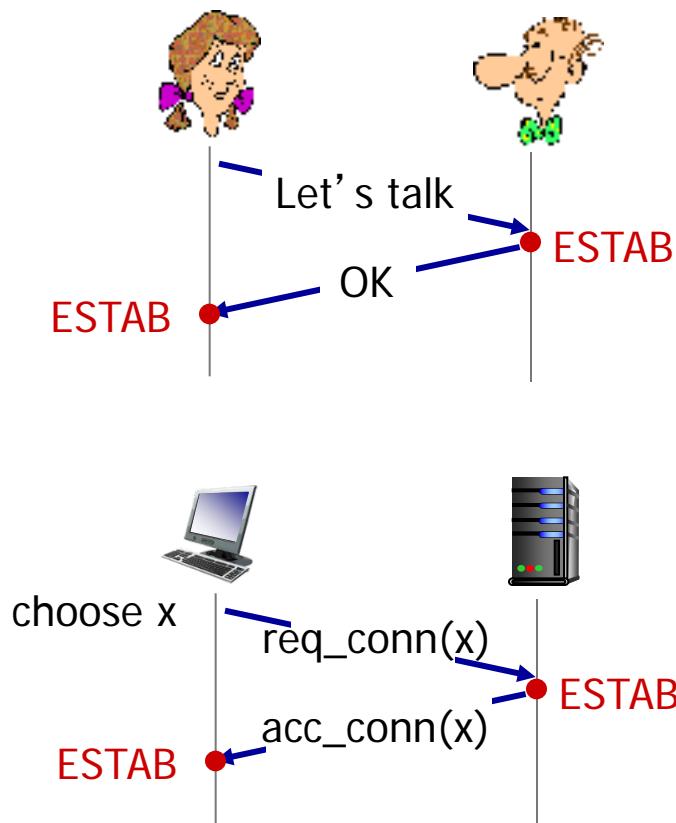
```
Socket clientSocket =  
    newSocket("hostname", "port  
    number");
```



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

Agreeing to establish a connection

2-way handshake:

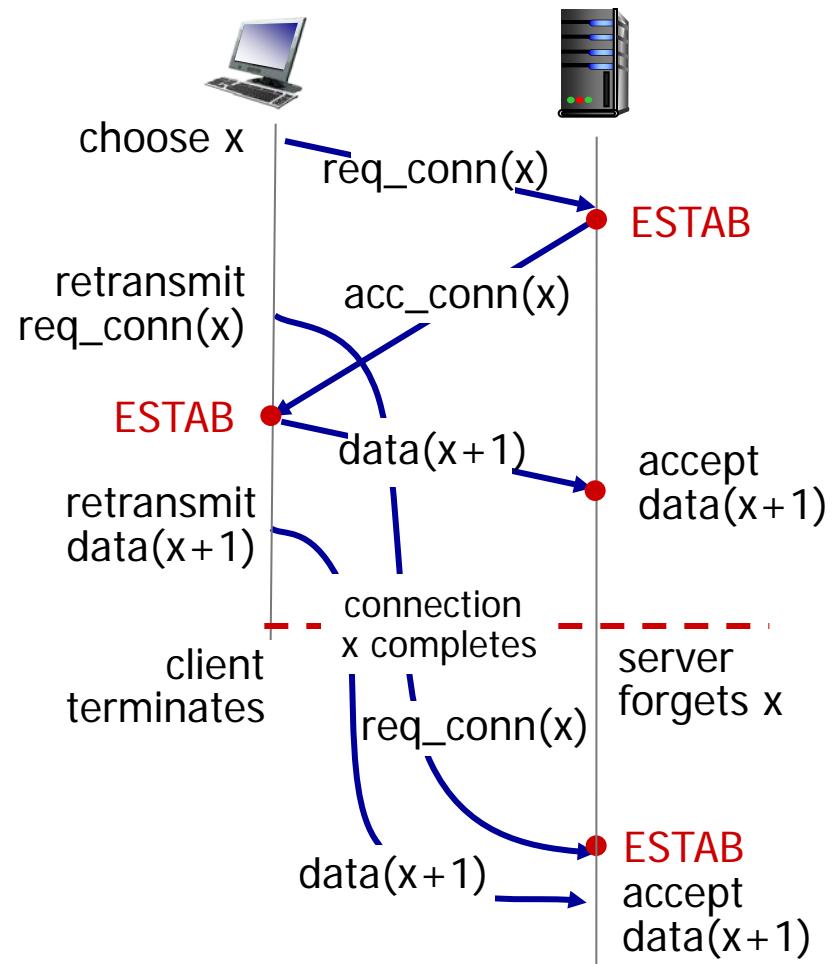
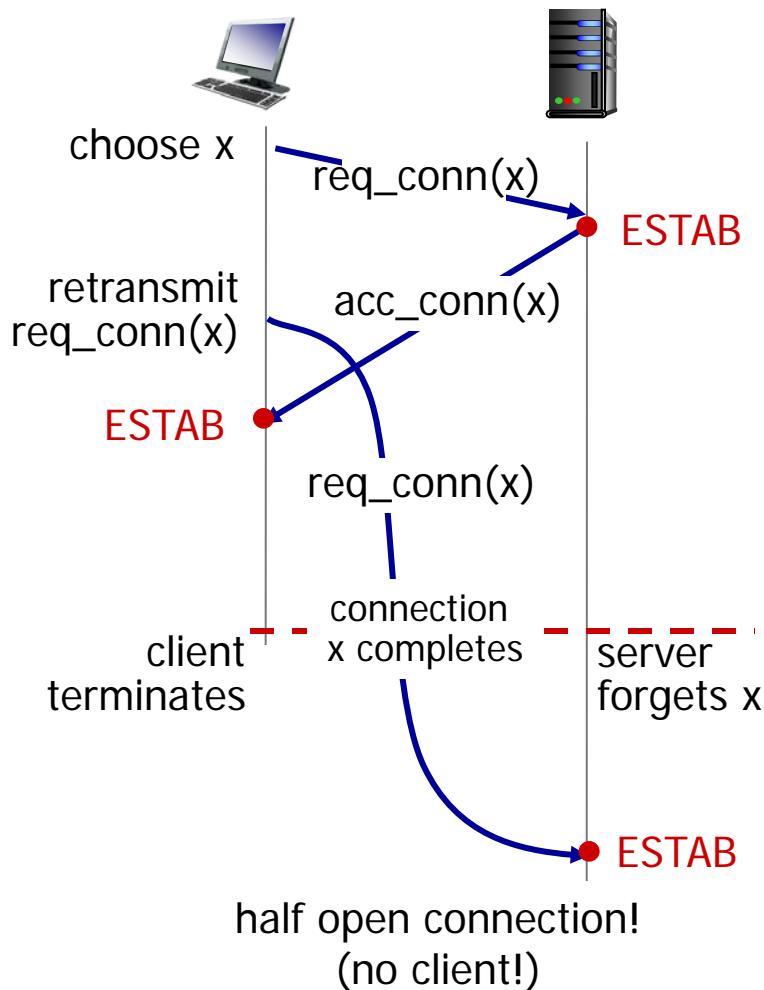


Q: will 2-way handshake always work in network?

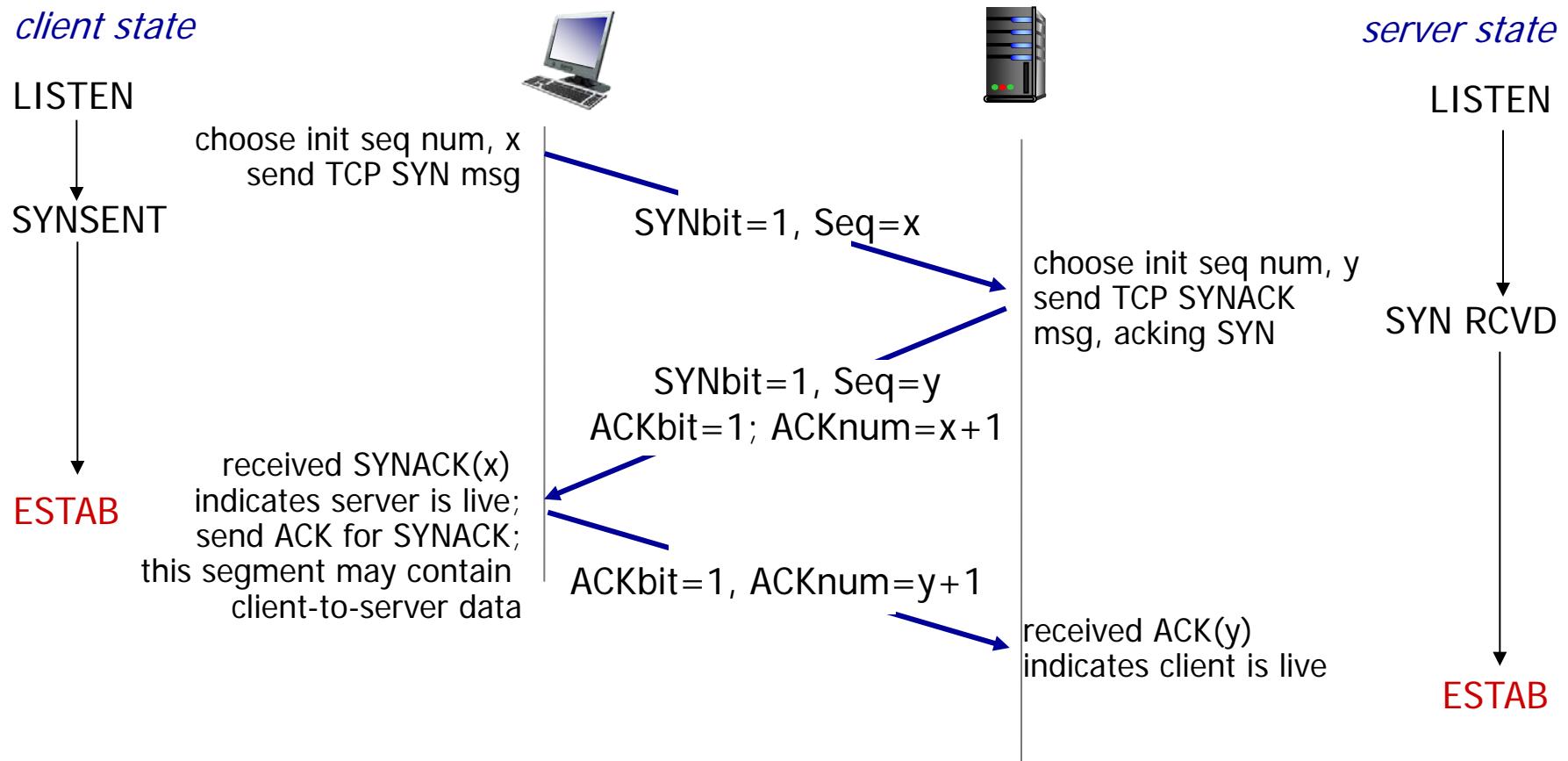
- ❖ variable delays
- ❖ retransmitted messages (e.g. $\text{req_conn}(x)$) due to message loss
- ❖ message reordering
- ❖ can't "see" other side

Agreeing to establish a connection

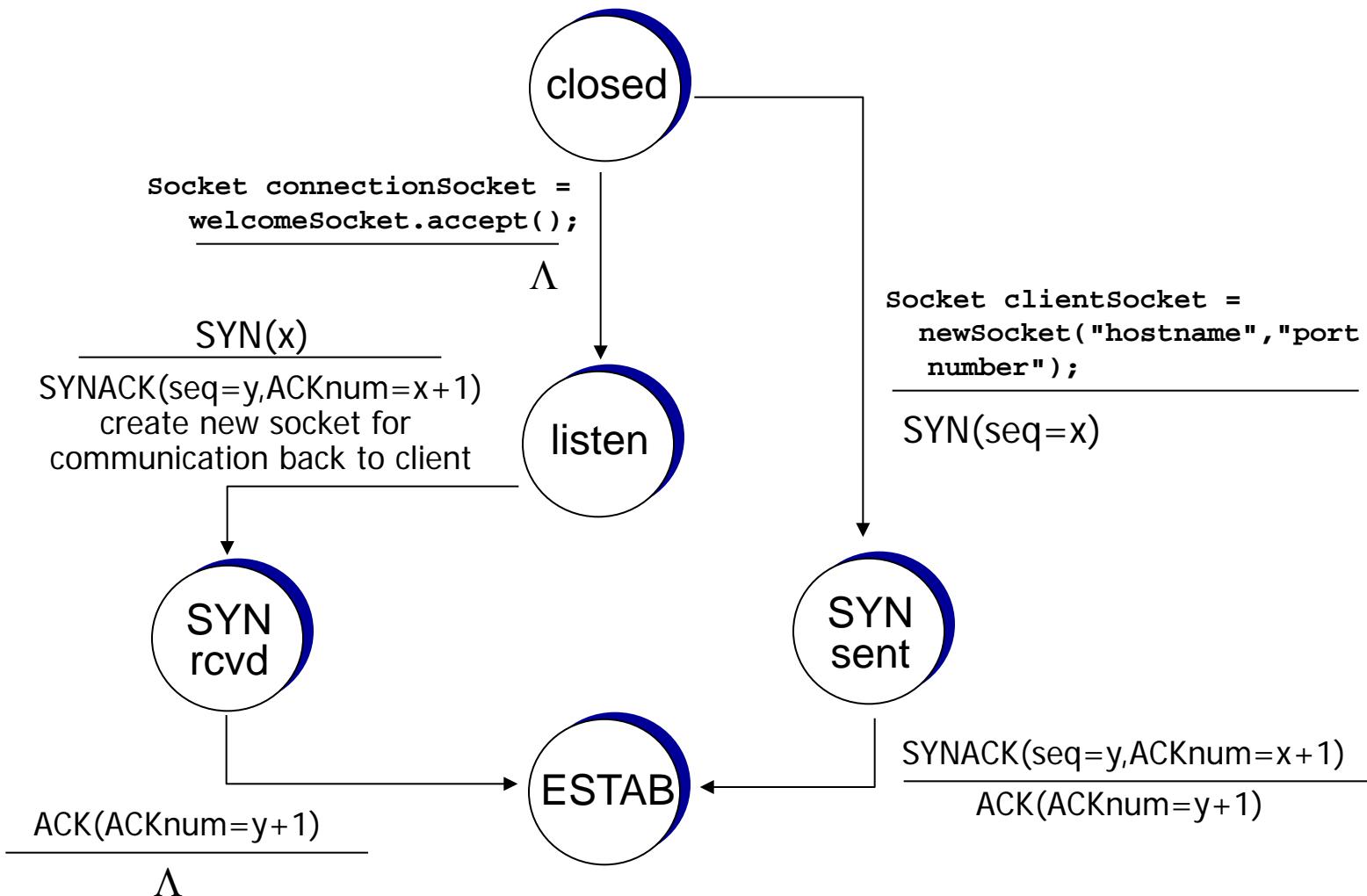
2-way handshake failure scenarios:



TCP 3-way handshake



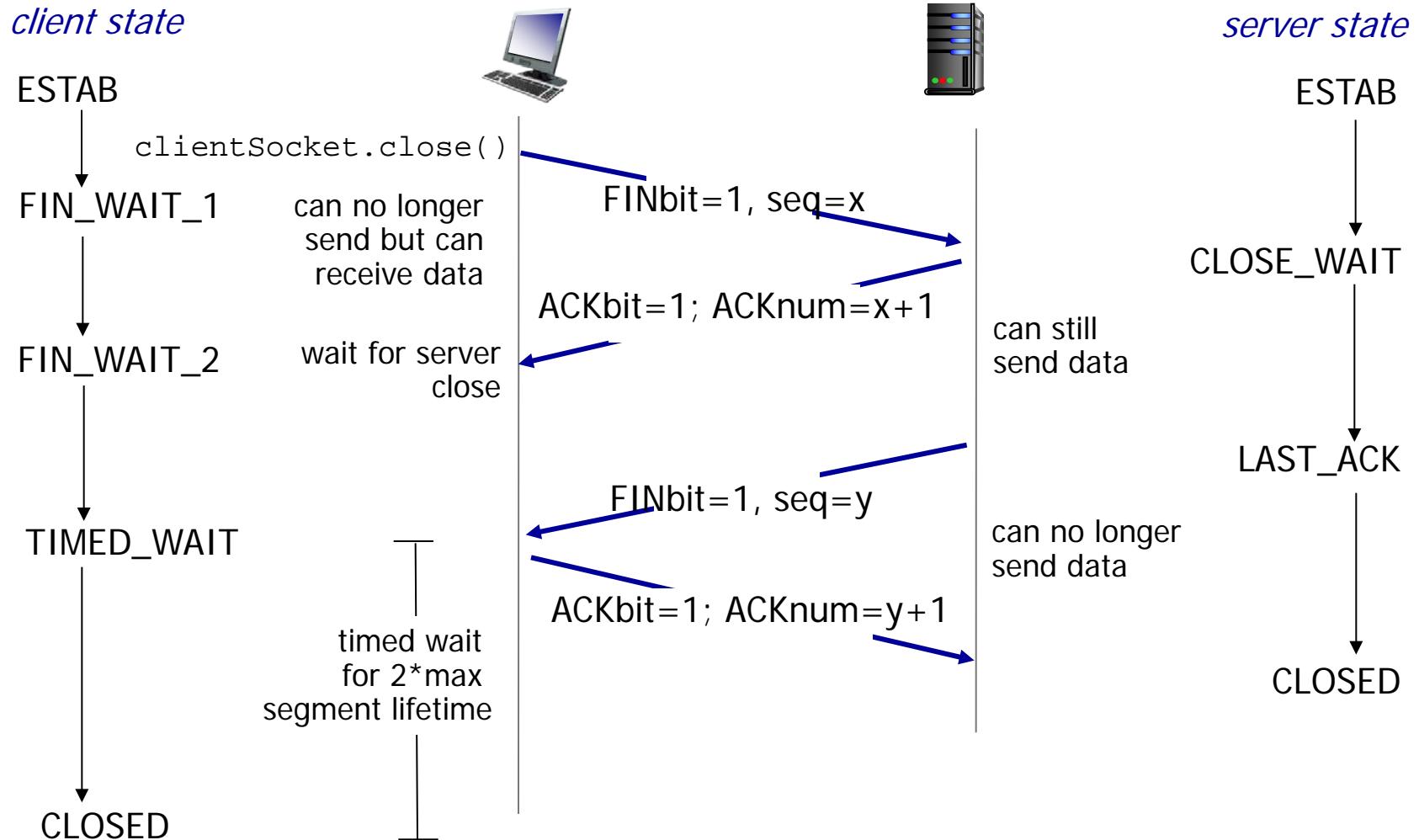
TCP 3-way handshake: FSM



TCP: closing a connection

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

TCP: closing a connection



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

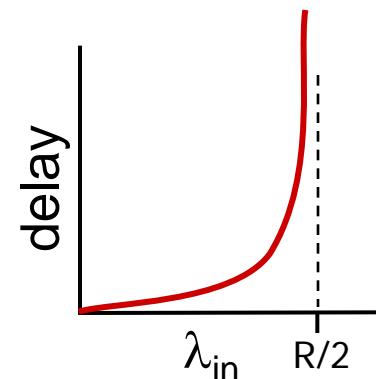
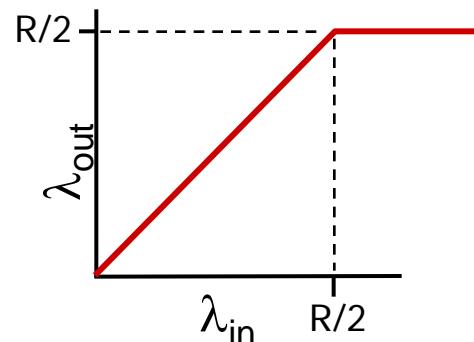
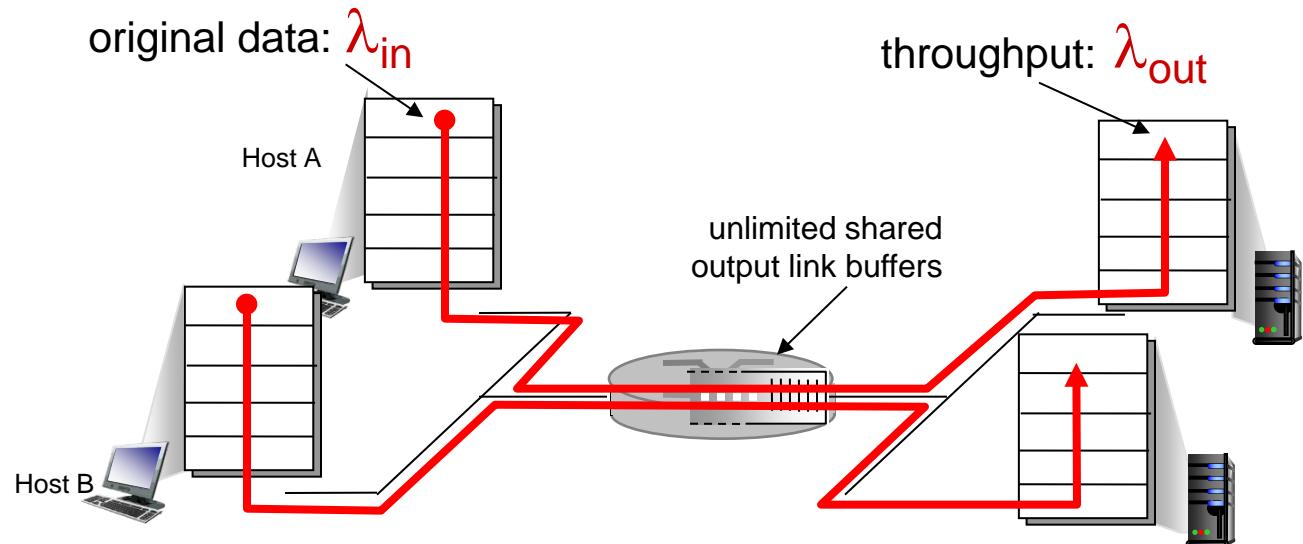
Principles of congestion control

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!

Causes/costs of congestion: scenario I

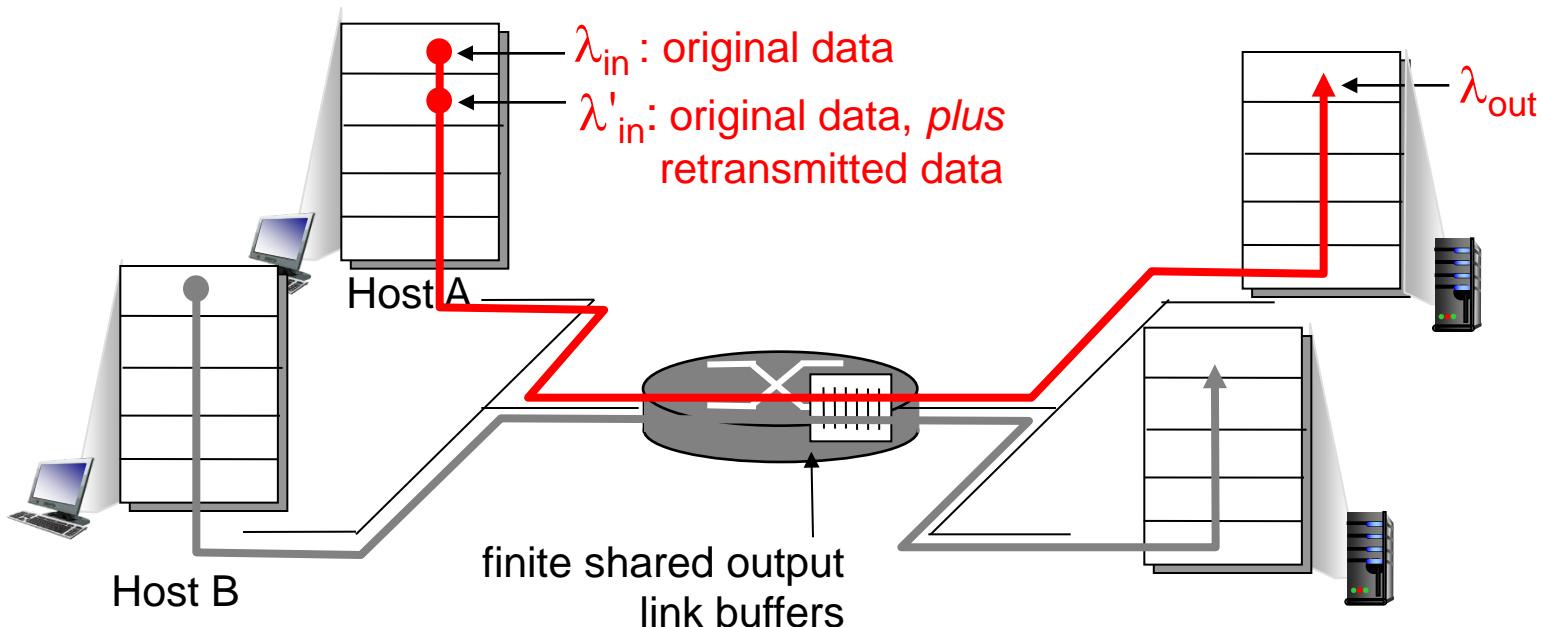
- ❖ two senders, two receivers
- ❖ one router, infinite buffers
- ❖ Shared output link capacity: R
- ❖ no retransmission



- ❖ maximum per-connection throughput: $R/2$
- ❖ large delays as arrival rate, λ_{in} , approaches capacity

Causes/costs of congestion: scenario 2

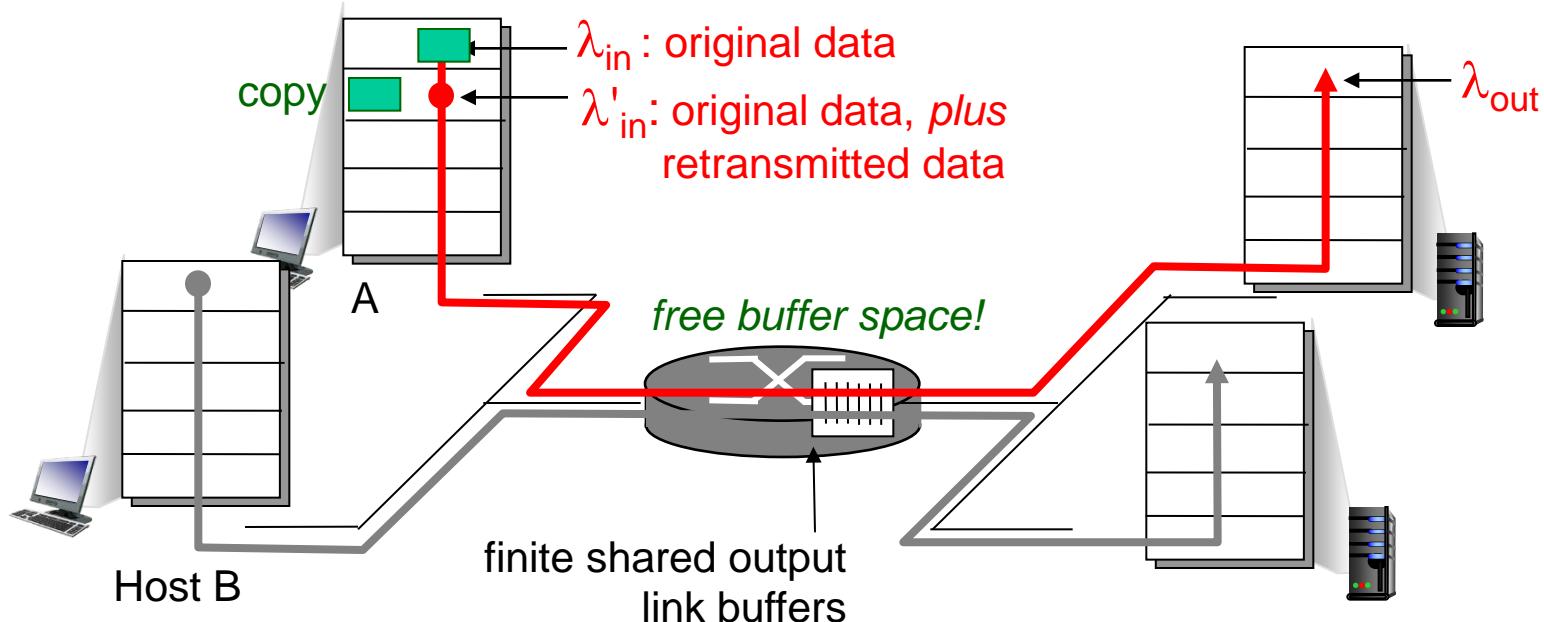
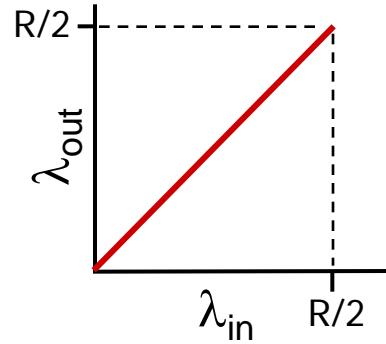
- ❖ one router, *finite* buffers
- ❖ sender retransmission of timed-out packet
 - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
 - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$



Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- ❖ sender sends only when router buffers available

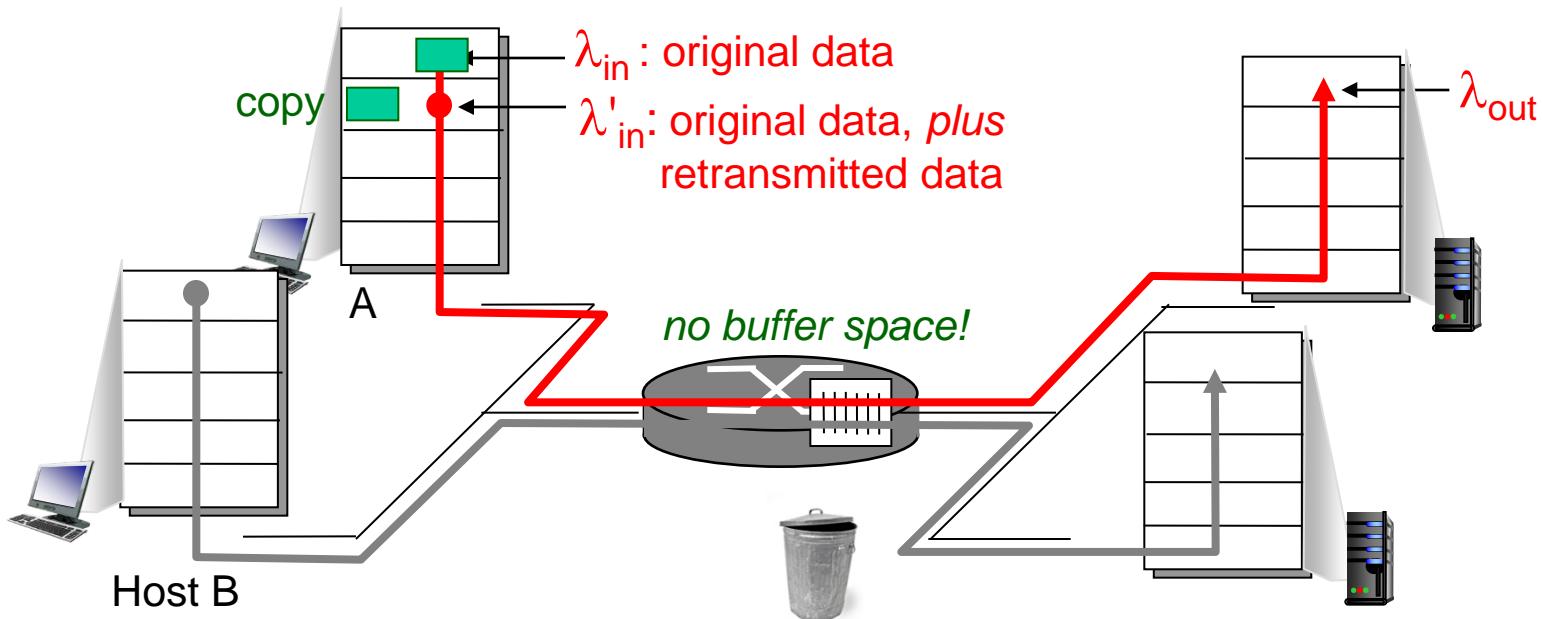


Causes/costs of congestion: scenario 2

Idealization: *known loss*

packets can be lost,
dropped at router due
to full buffers

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

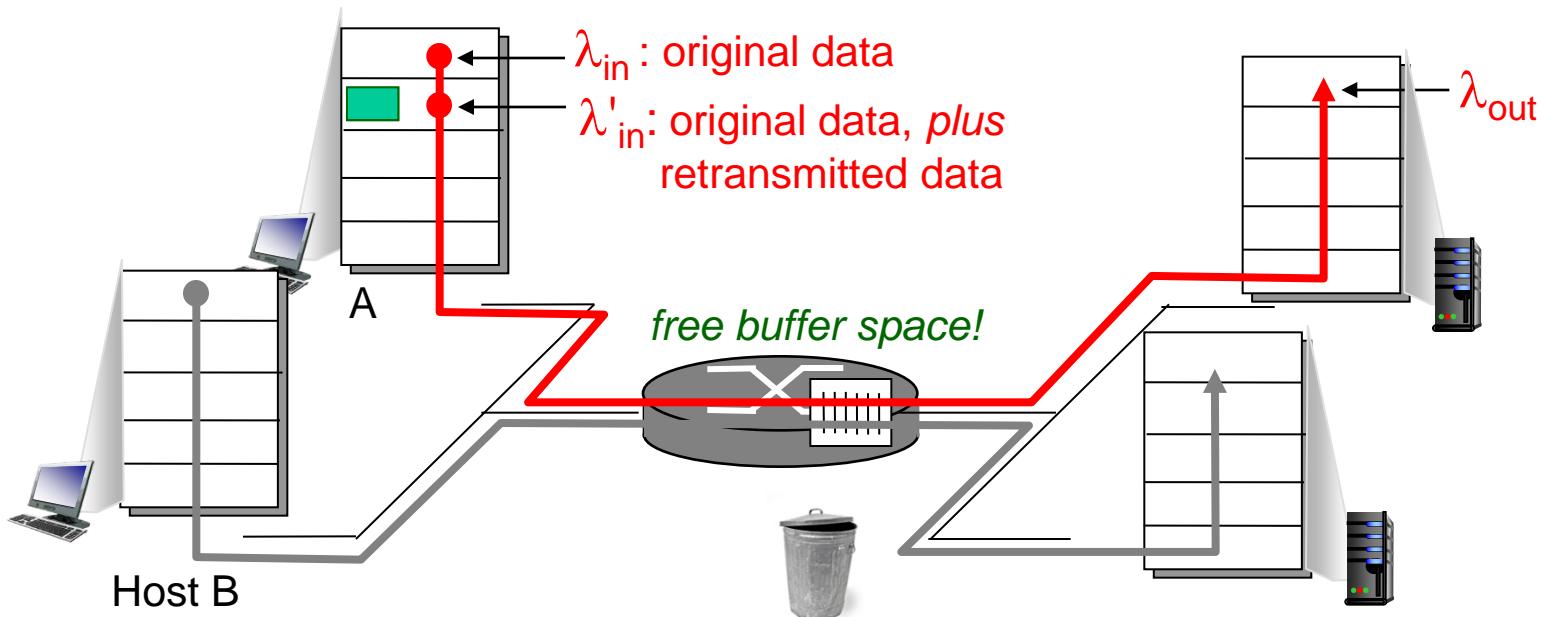
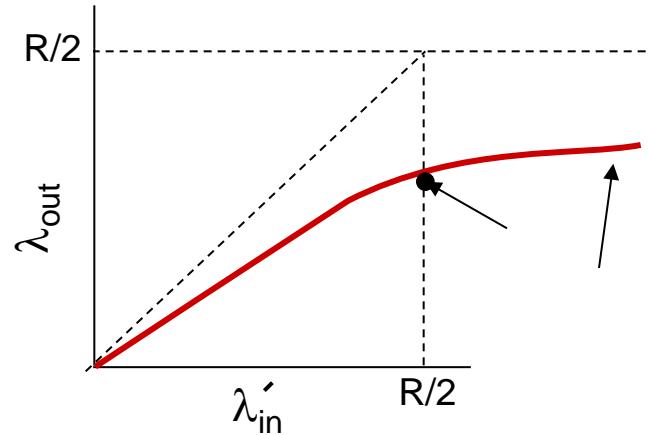


Causes/costs of congestion: scenario 2

Idealization: known loss

packets can be lost,
dropped at router due
to full buffers

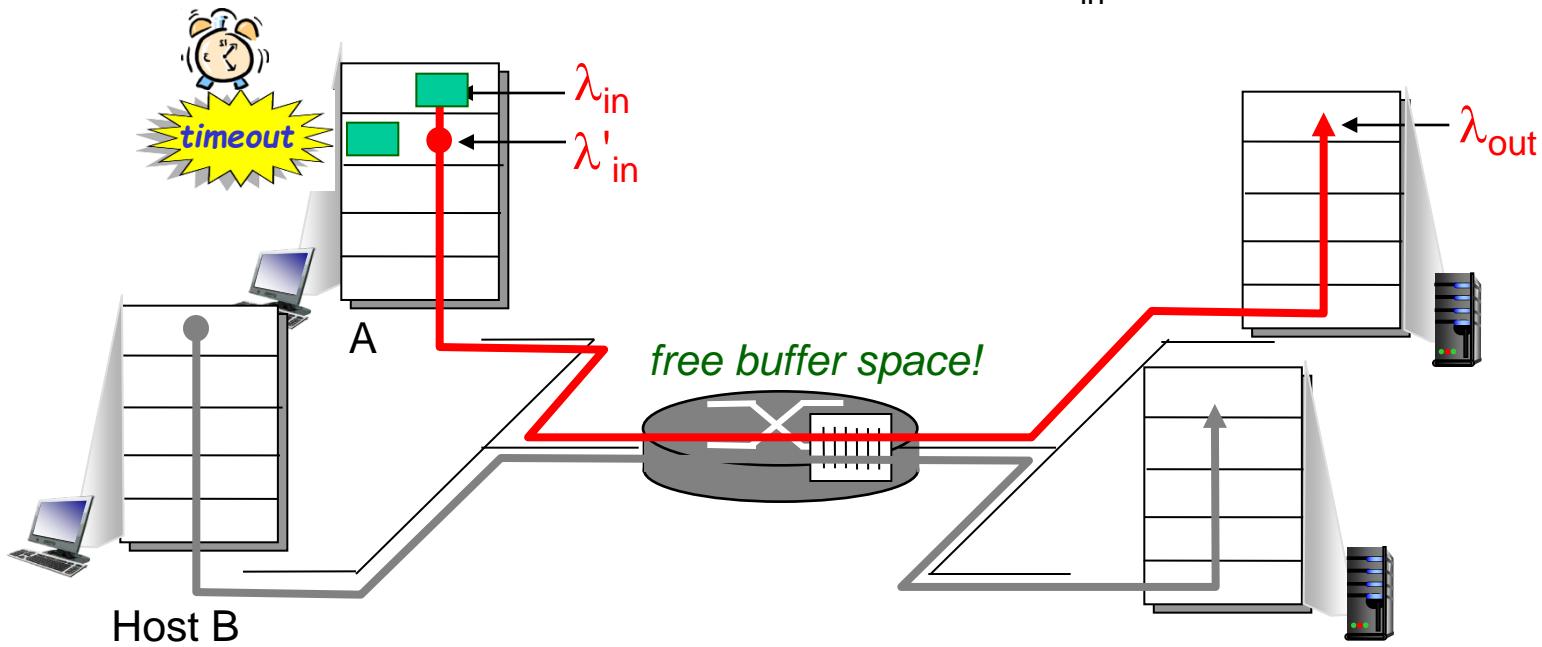
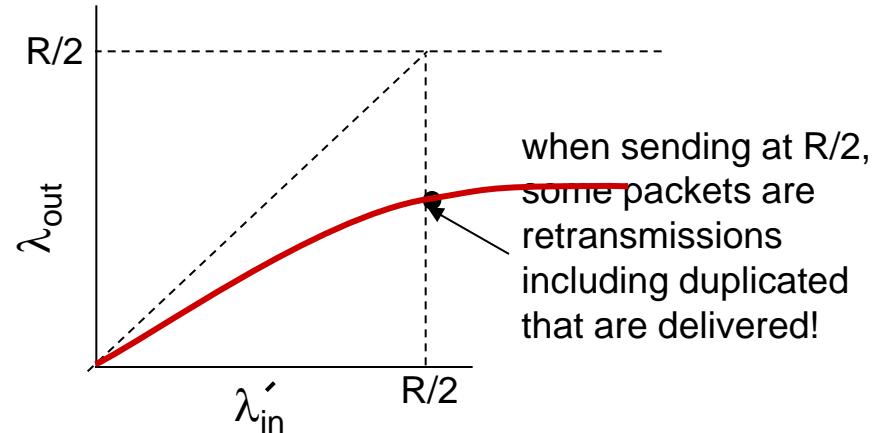
- ❖ sender only resends if
packet known to be lost



Causes/costs of congestion: scenario 2

Realistic: *duplicates*

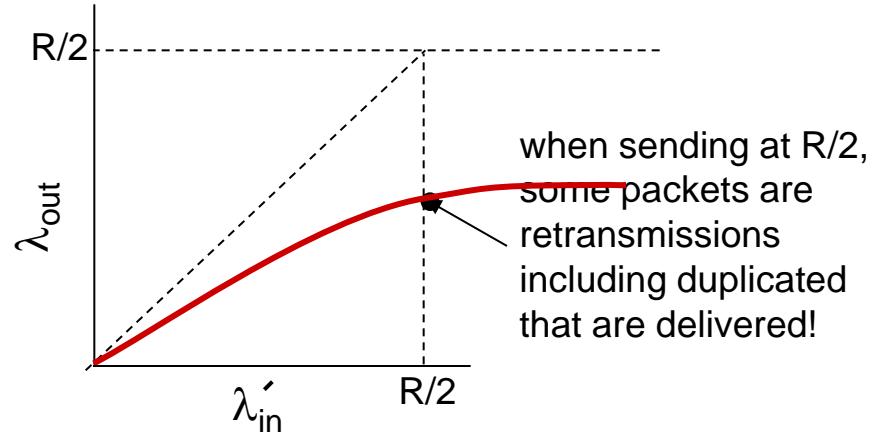
- ❖ packets can be lost, dropped at router due to full buffers
- ❖ sender times out prematurely, sending **two** copies, both of which are delivered



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



“costs” of congestion:

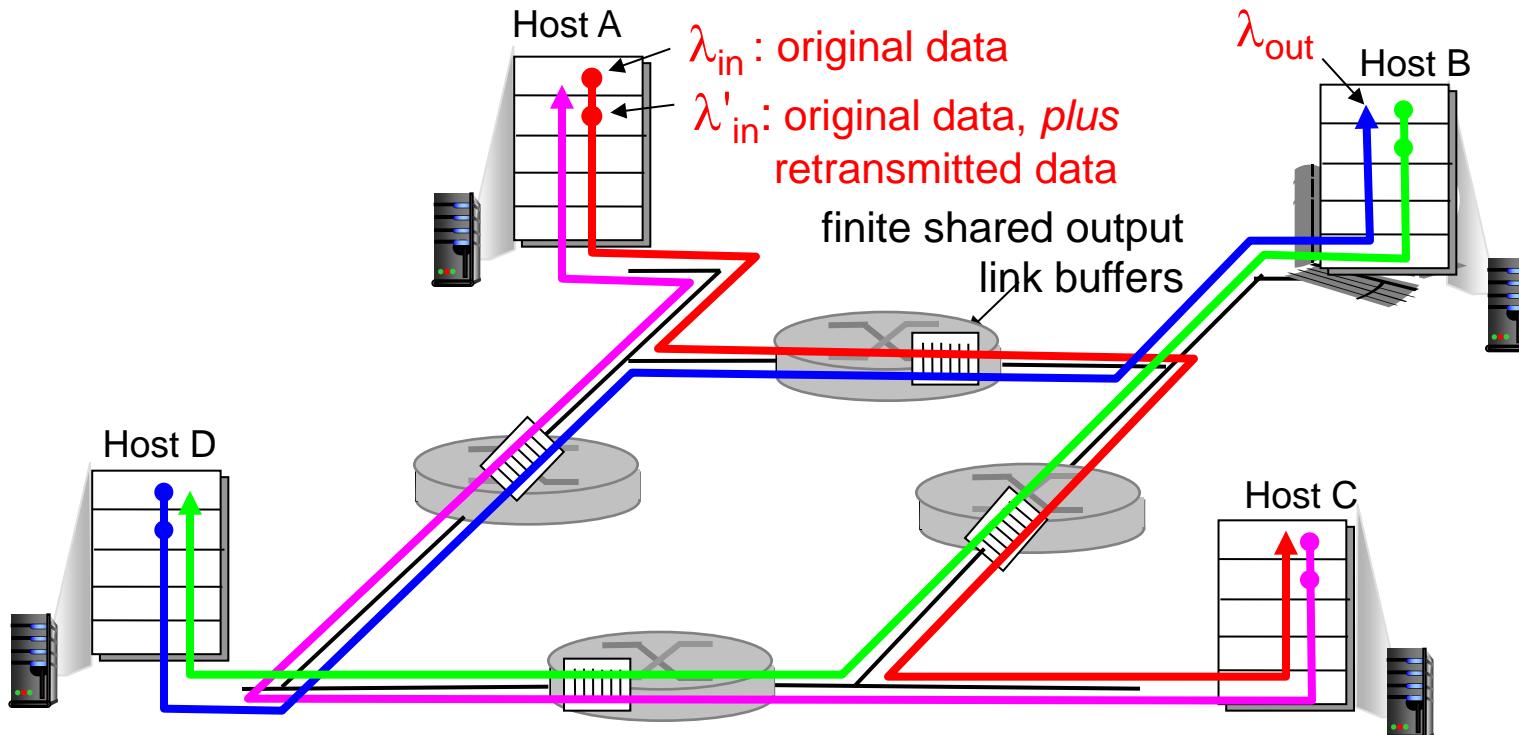
- ❖ more work (retrans) for given “goodput”
- ❖ unneeded retransmissions: link carries multiple copies of pkt
 - decreasing goodput

Causes/costs of congestion: scenario 3

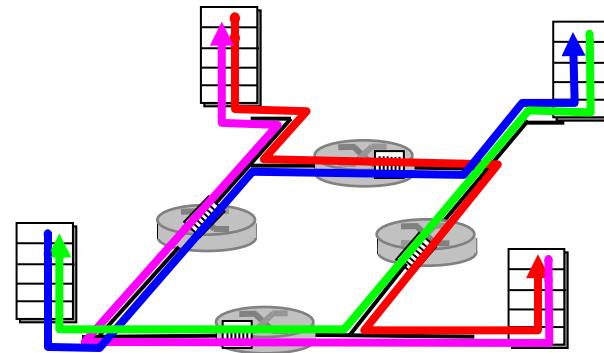
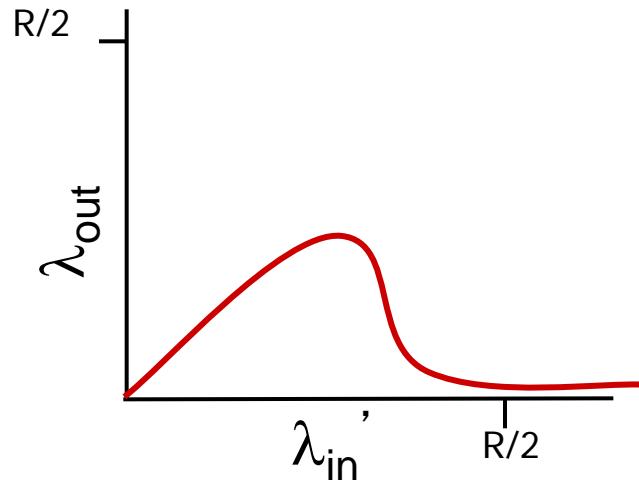
- ❖ four senders
- ❖ multihop paths
- ❖ timeout/retransmit

Q: what happens as λ_{in} and λ'_{in} increase ?

A: as red λ_{in} increases, all arriving blue pkts at upper queue are dropped, blue throughput $\rightarrow 0$



Causes/costs of congestion: scenario 3



another “cost” of congestion:

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

Approaches towards congestion control

two broad approaches towards congestion control:

end-end congestion control:

- ❖ no explicit feedback from network
- ❖ congestion inferred from end-system observed loss, delay
- ❖ approach taken by TCP

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

Case study: ATM ABR congestion control

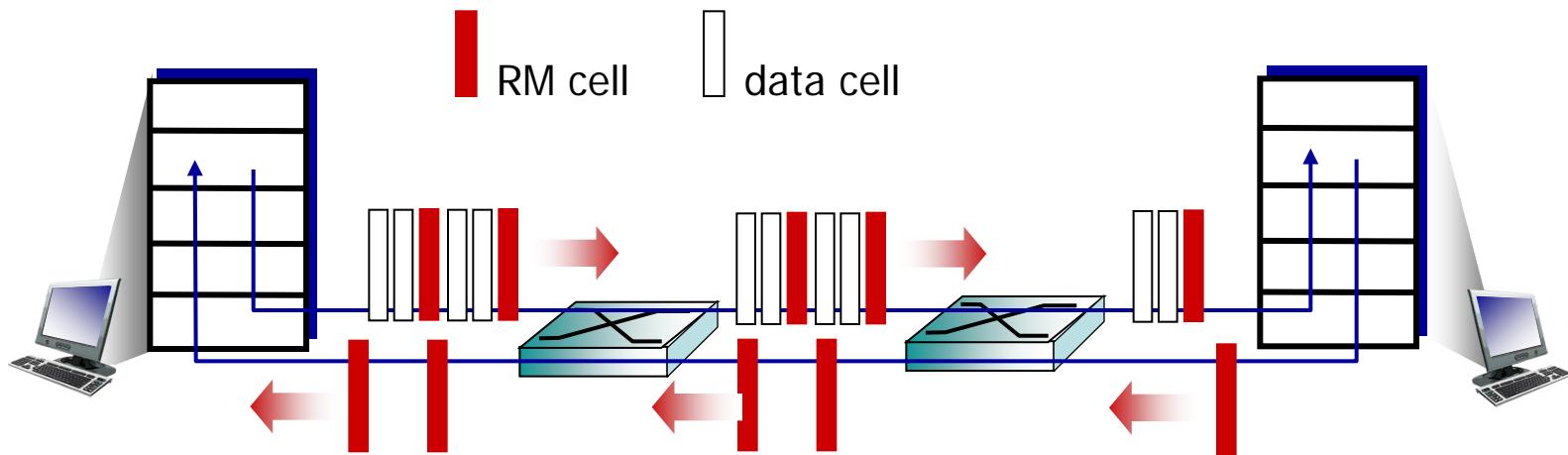
ABR: available bit rate:

- ❖ “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)
 - *CI bit*: congestion indication
- ❖ RM cells returned to sender by receiver, with bits intact

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 min 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

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

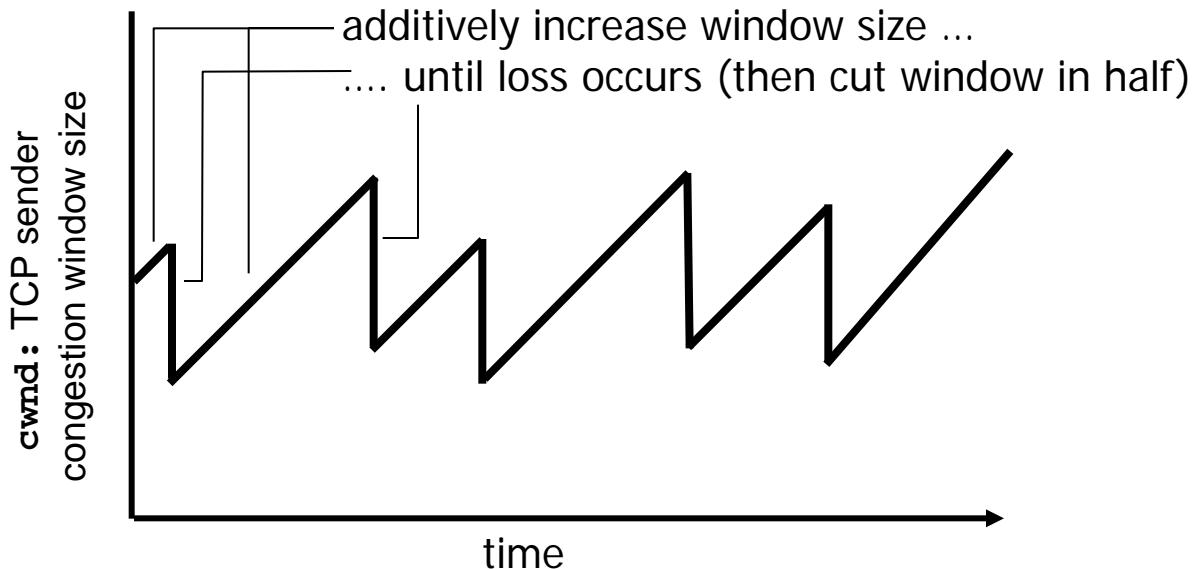
3.6 principles of congestion control

3.7 TCP congestion control

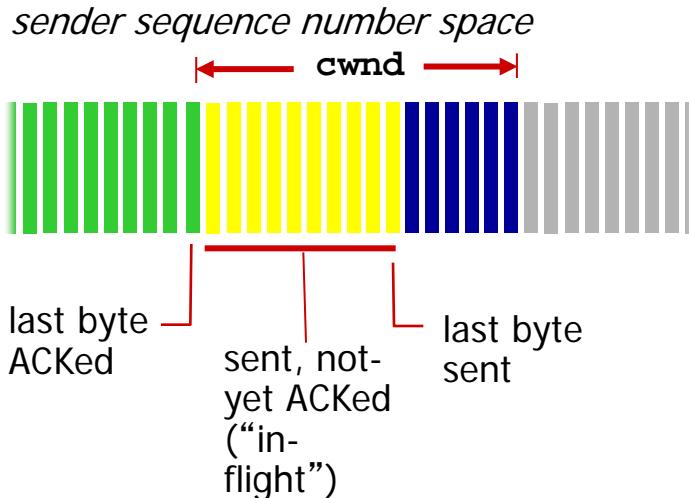
TCP congestion control: additive increase multiplicative decrease

- ❖ *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



TCP Congestion Control: details



- ❖ **sender limits transmission:**

$$\frac{\text{LastByteSent} - \text{LastByteAcked}}{\text{cwnd}} \leq 1$$

TCP sending rate:

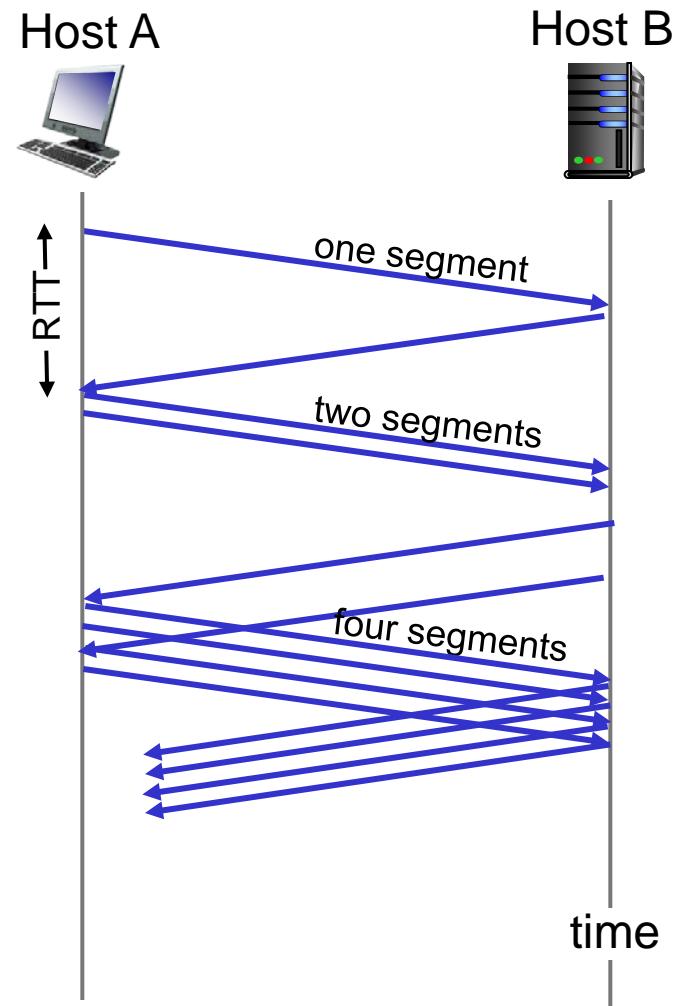
- ❖ *roughly*: send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- ❖ **cwnd is dynamic, function of perceived network congestion**

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



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
 - **cwnd** is cut in half window then grows linearly
- ❖ TCP Tahoe always sets **cwnd** to 1 (timeout or 3 duplicate acks)

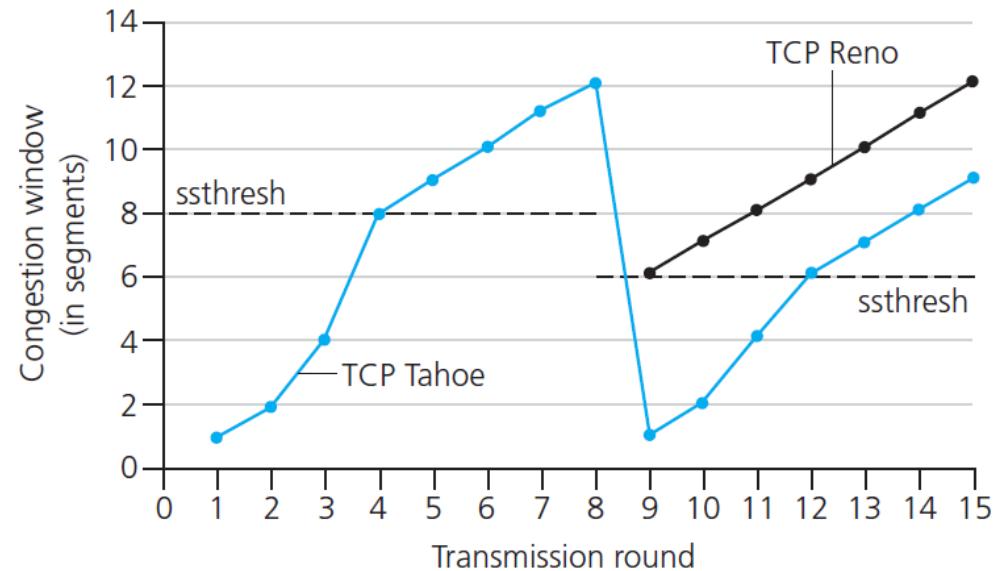
TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

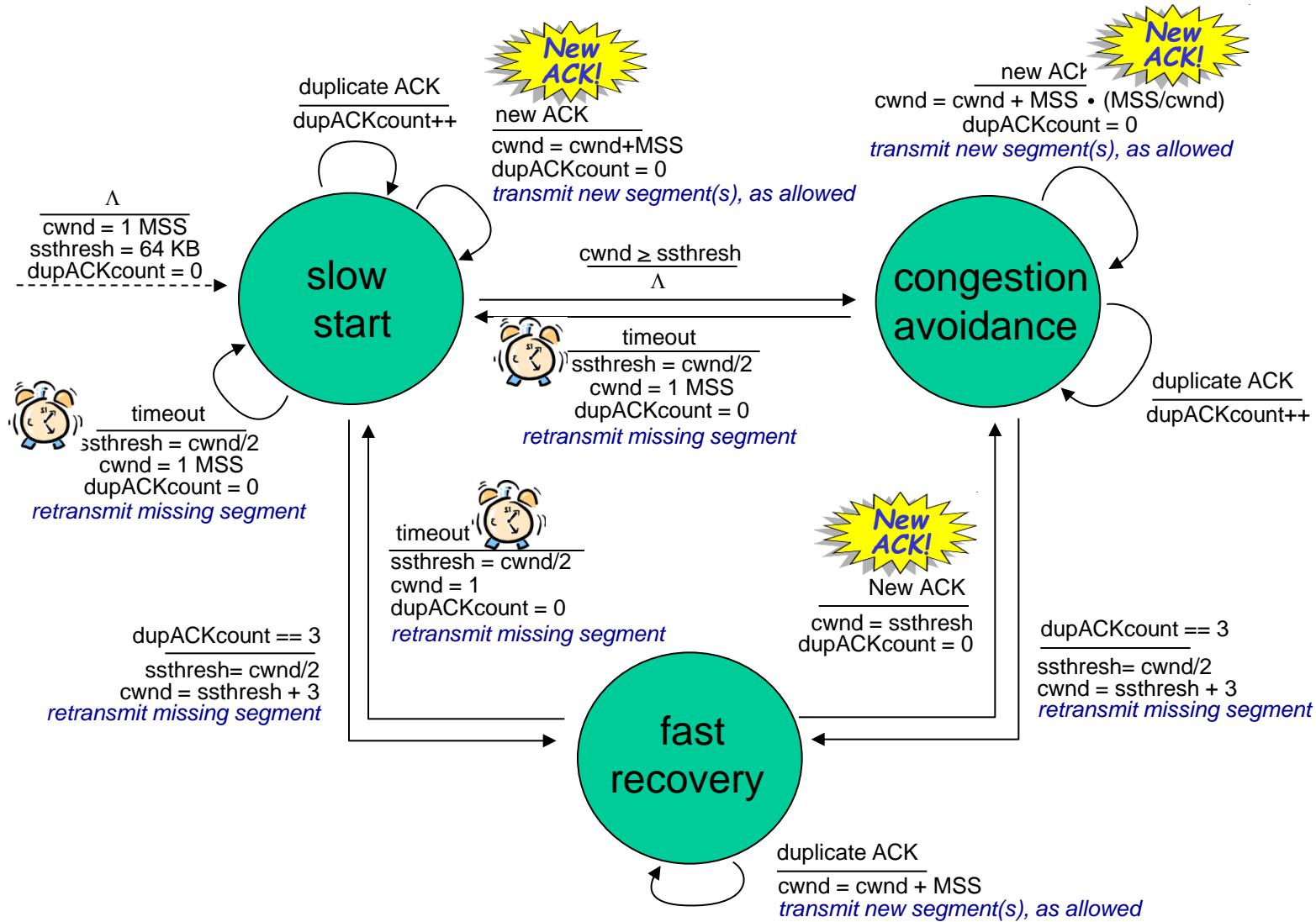
A: when **cwnd** gets to 1/2 of its value before timeout.

Implementation:

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



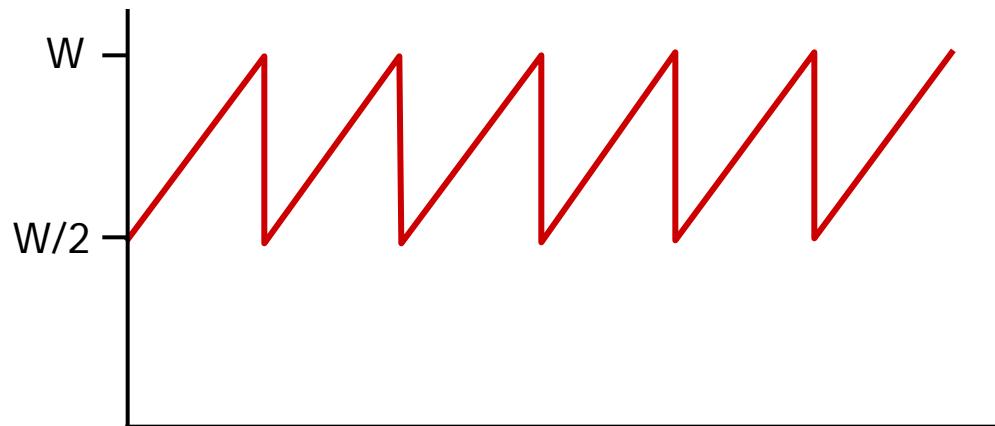
Summary: TCP Congestion Control



TCP throughput

- ❖ avg. TCP throughput 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 $\frac{3}{4}W$
 - avg. throughput is $\frac{3}{4}W$ per RTT

$$\text{avg TCP throughput} = \frac{3}{4} \frac{W}{\text{RTT}} \text{ bytes/sec}$$



TCP Futures: TCP over “long, fat pipes”

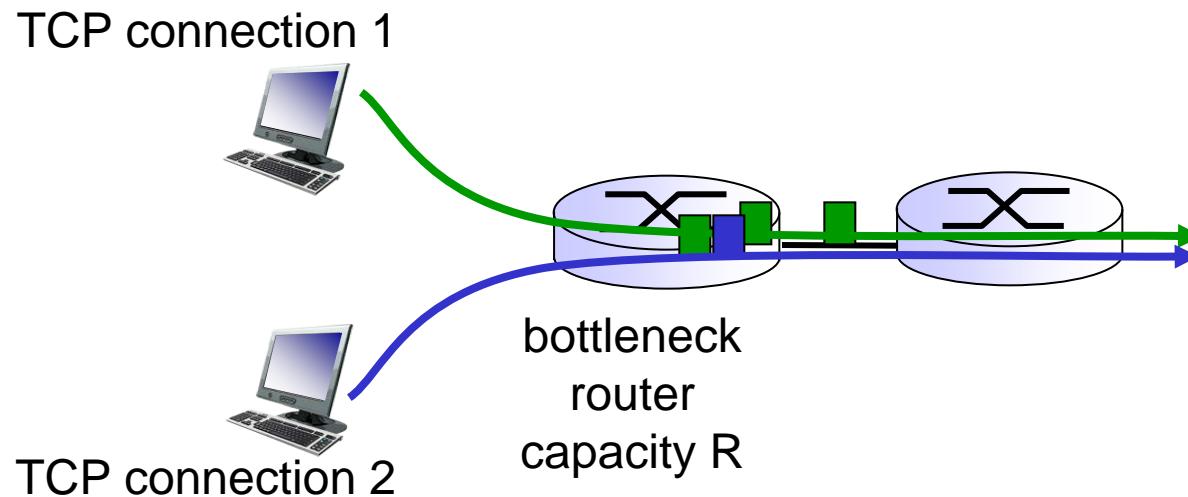
- ❖ example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- ❖ requires $W = 83,333$ in-flight segments
- ❖ throughput in terms of segment loss probability, L [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

- to achieve 10 Gbps throughput, need a loss rate of $L = 2 \cdot 10^{-10}$ – *a very small loss rate!*
- ❖ new versions of TCP for high-speed

TCP Fairness

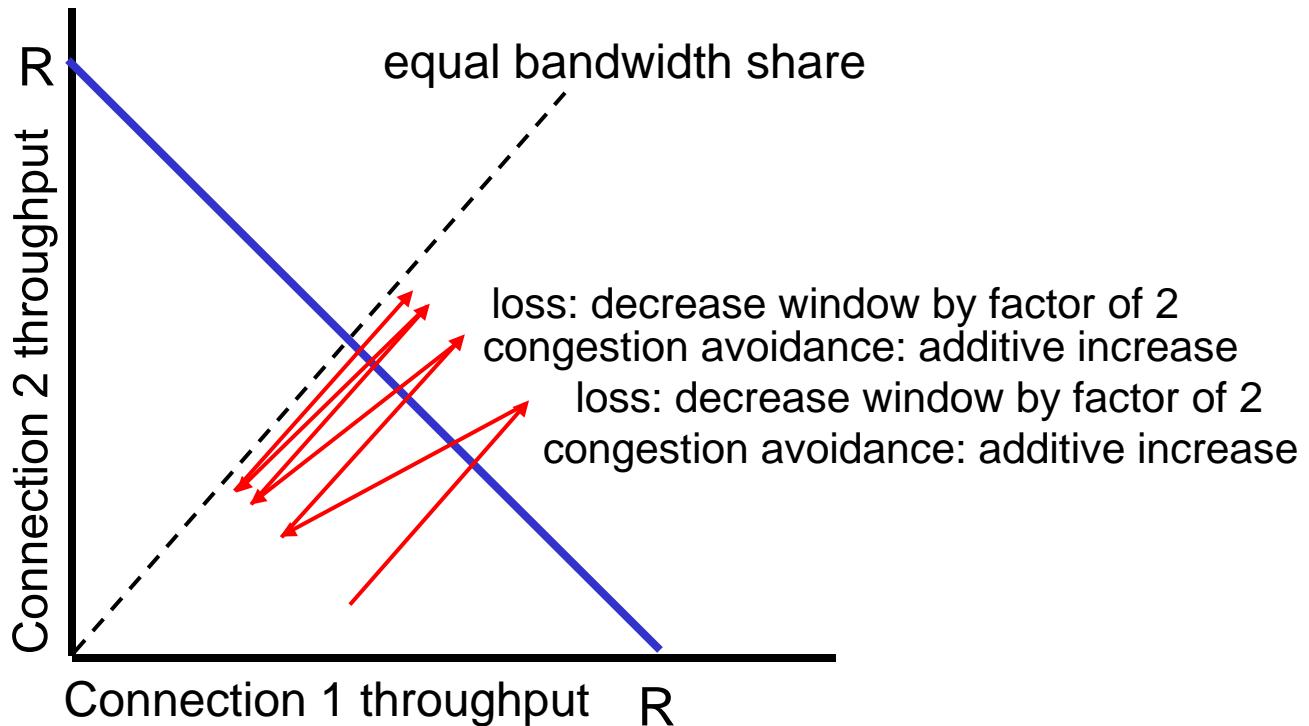
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R , each should have average rate of R/K



Why is TCP fair?

two competing sessions:

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



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

- ❖ application can open multiple parallel connections between two hosts
- ❖ web browsers do this
- ❖ e.g., link of rate R with 9 existing connections:
 - new app asks for 1 TCP, gets rate $R/10$
 - 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

© *The materials in these notes are adapted from Computer Networking: A Top Down Approach, 6th edition, by Jim Kurose, Keith Ross*

Chapter 4: network layer

chapter goals:

- ❖ understand principles behind network layer services:
 - network layer service models
 - forwarding versus routing
 - how a router works
 - routing (path selection)
 - broadcast, multicast
- ❖ instantiation, implementation in the Internet

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

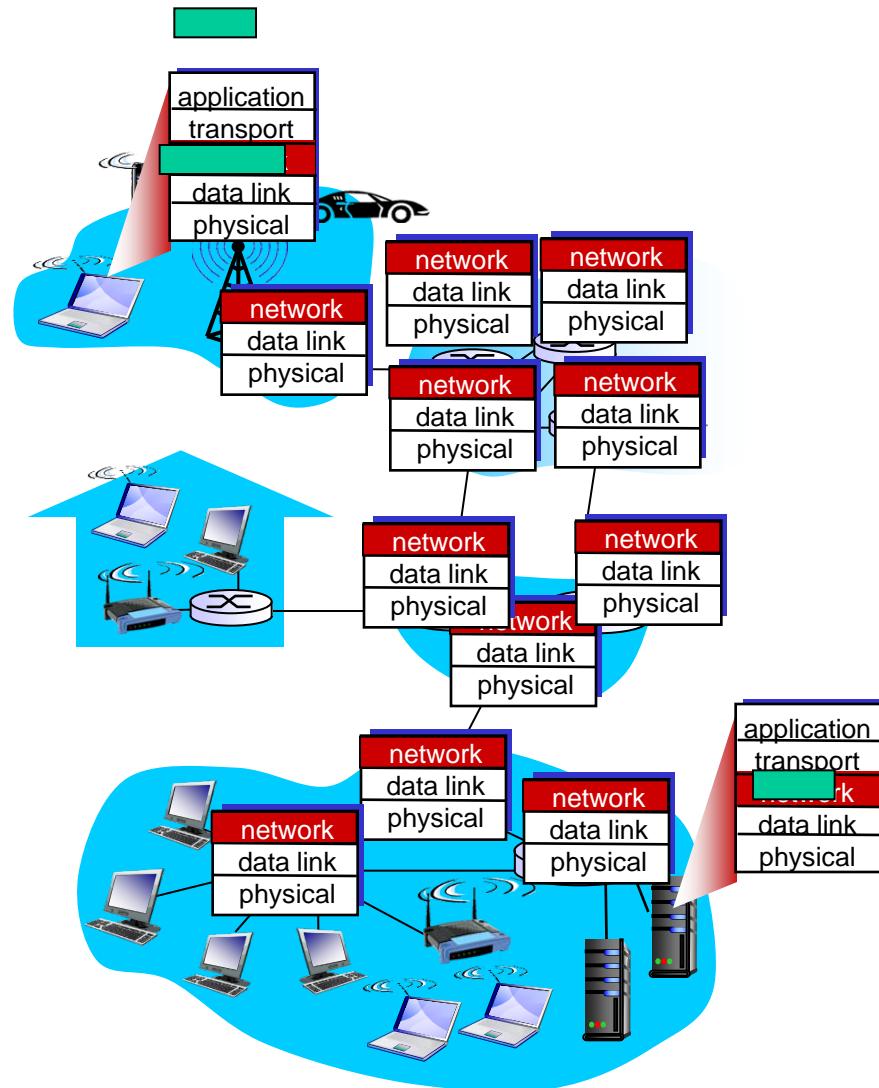
4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

Network layer

- ❖ transport segment from sending to receiving host
- ❖ on sending side encapsulates segments into datagrams
- ❖ on receiving side, delivers segments to transport layer
- ❖ network layer protocols in **every** host, router
- ❖ router examines header fields in all IP datagrams passing through it



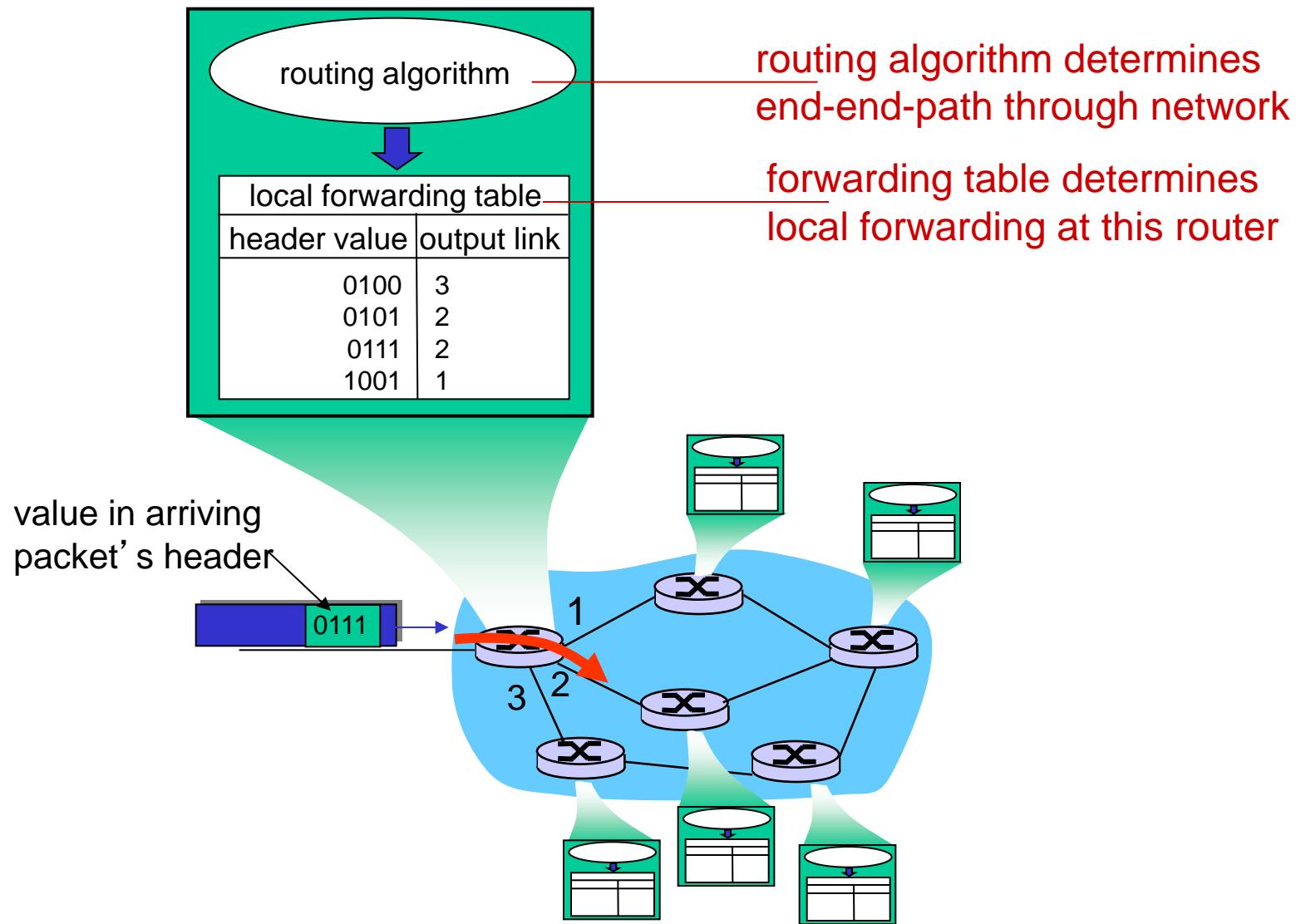
Two key network-layer functions

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

analogy:

- ❖ *routing*: process of planning trip from source to dest
- ❖ *forwarding*: process of getting through single interchange

Interplay between routing and forwarding



Connection setup

- ❖ 3rd important function in some network architectures:
 - ATM, frame relay, X.25
- ❖ before datagrams flow, two end hosts *and* intervening routers establish virtual connection
 - routers get involved
- ❖ network vs transport layer connection service:
 - **network:** between two hosts (may also involve intervening routers in case of VCs)
 - **transport:** between two processes

Network service model

Q: What *service model* for “channel” transporting datagrams from sender to receiver?

example services for individual datagrams:

- ❖ guaranteed delivery
- ❖ guaranteed delivery with less than 40 msec delay

example services for a flow of datagrams:

- ❖ in-order datagram delivery
- ❖ guaranteed minimum bandwidth to flow
- ❖ restrictions on changes in inter-packet spacing

Network layer service models:

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

Connection, connection-less service

- ❖ *datagram* network provides network-layer *connectionless* service
- ❖ *virtual-circuit* network provides network-layer *connection* service
- ❖ analogous to TCP/UDP connection-oriented / connectionless transport-layer services, but:
 - *service*: host-to-host
 - *no choice*: network provides one or the other
 - *implementation*: in network core

Virtual circuits

“source-to-dest path behaves much like telephone circuit”

- performance-wise
- network actions along source-to-dest path

- ❖ call setup for each call *before* data can flow
- ❖ each packet carries VC identifier (not destination host address)
- ❖ every router on source-dest path maintains “state” for each passing connection
- ❖ link, router resources (bandwidth, buffers) may be *allocated* to VC (dedicated resources = predictable service)

VC implementation

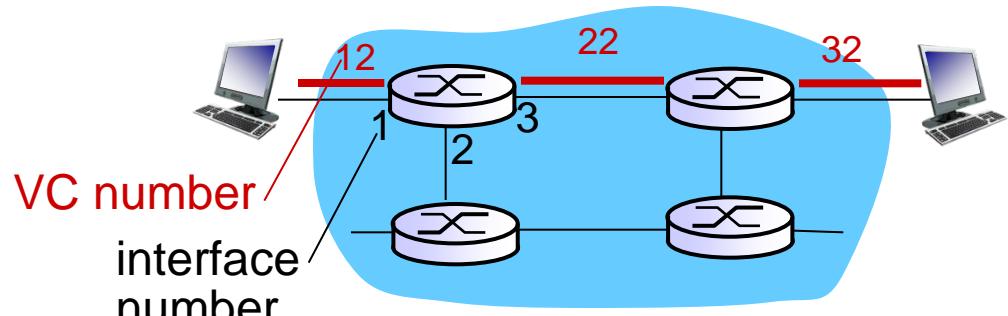
a VC consists of:

1. *path* from source to destination
 2. *VC numbers*, one number for each link along path
 3. *entries in forwarding tables* in routers along path
- ❖ packet belonging to VC carries VC number (rather than dest address)
 - ❖ VC number can be changed on each link.
 - new VC number comes from forwarding table

VC forwarding table

forwarding table in northwest router:

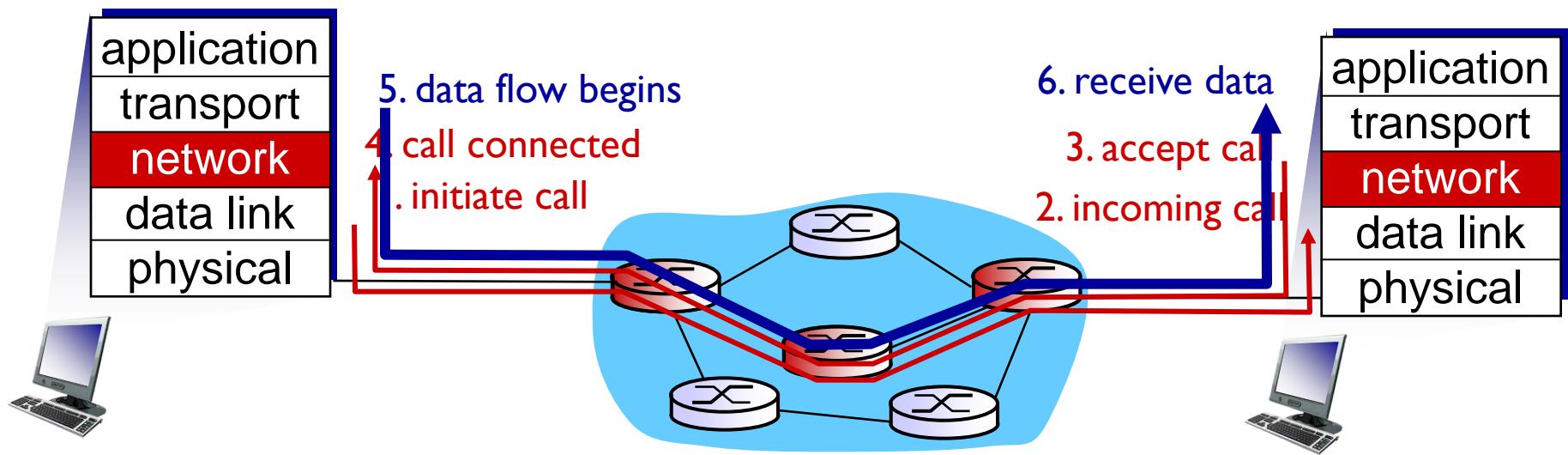
Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
...



VC routers maintain connection state information!

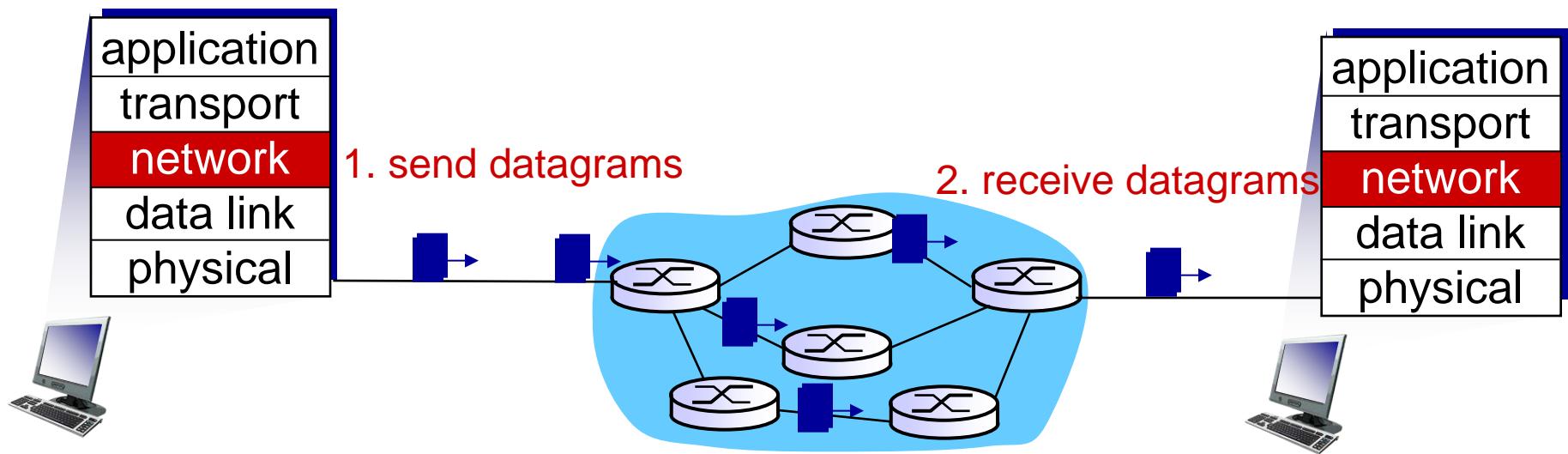
Virtual circuits: signaling protocols

- ❖ used to setup, maintain teardown VC
- ❖ used in ATM, frame-relay, X.25
- ❖ not used in today's Internet

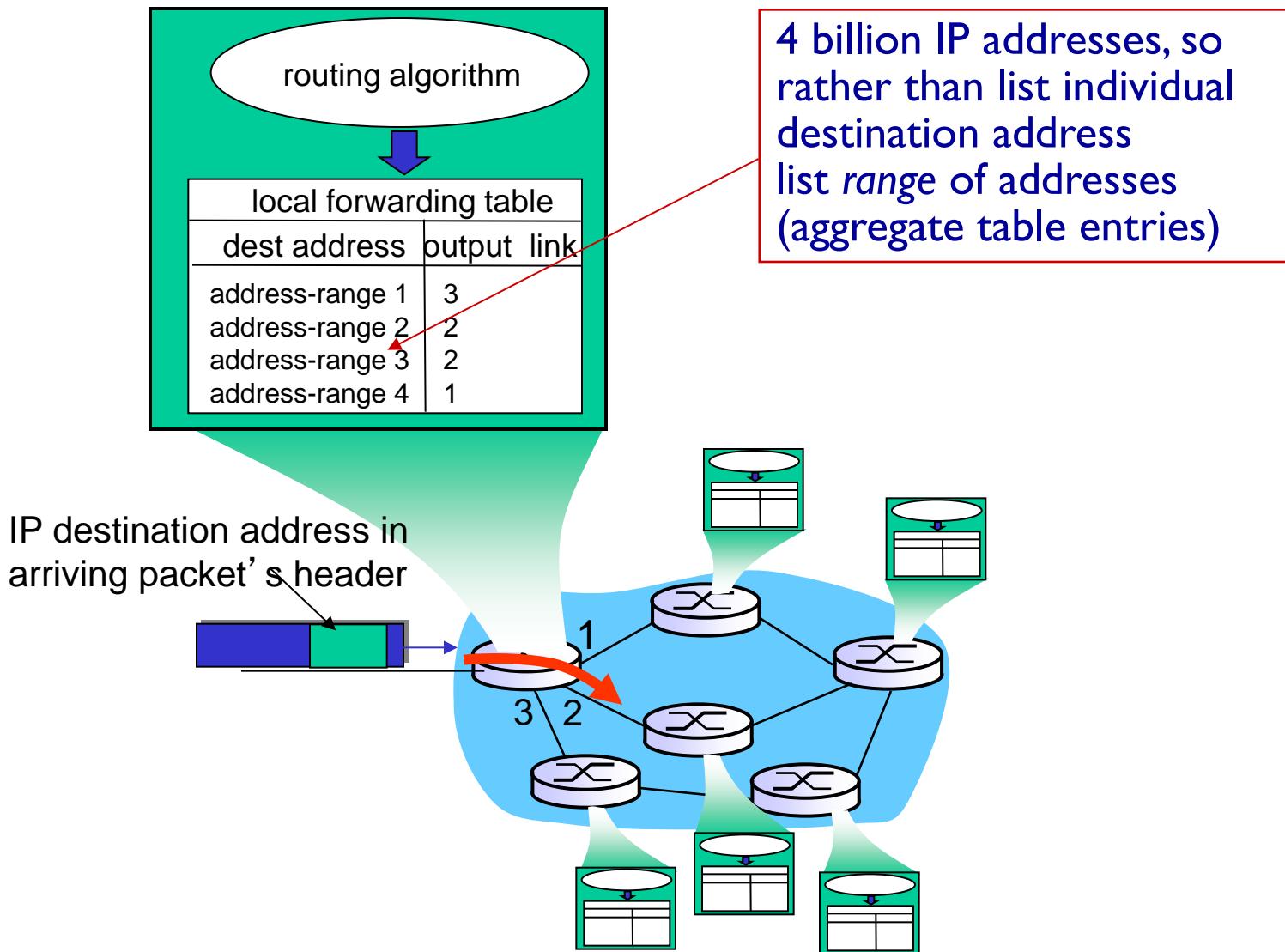


Datagram networks

- ❖ no call setup at network layer
- ❖ routers: no state about end-to-end connections
 - no network-level concept of “connection”
- ❖ packets forwarded using destination host address



Datagram forwarding table



Datagram forwarding table

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

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

longest prefix matching —

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

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

examples:

DA: 11001000 00010111 00010110 10100001

which interface?

DA: 11001000 00010111 00011000 10101010

which interface?

Datagram or VC network: why?

Internet (datagram)

- ❖ data exchange among computers
 - “elastic” service, no strict timing req.
- ❖ many link types
 - different characteristics
 - uniform service difficult
- ❖ “smart” end systems (computers)
 - can adapt, perform control, error recovery
 - ***simple inside network, complexity at “edge”***

ATM (VC)

- ❖ evolved from telephony
- ❖ human conversation:
 - strict timing, reliability requirements
 - need for guaranteed service
- ❖ “dumb” end systems
 - telephones
 - ***complexity inside network***

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

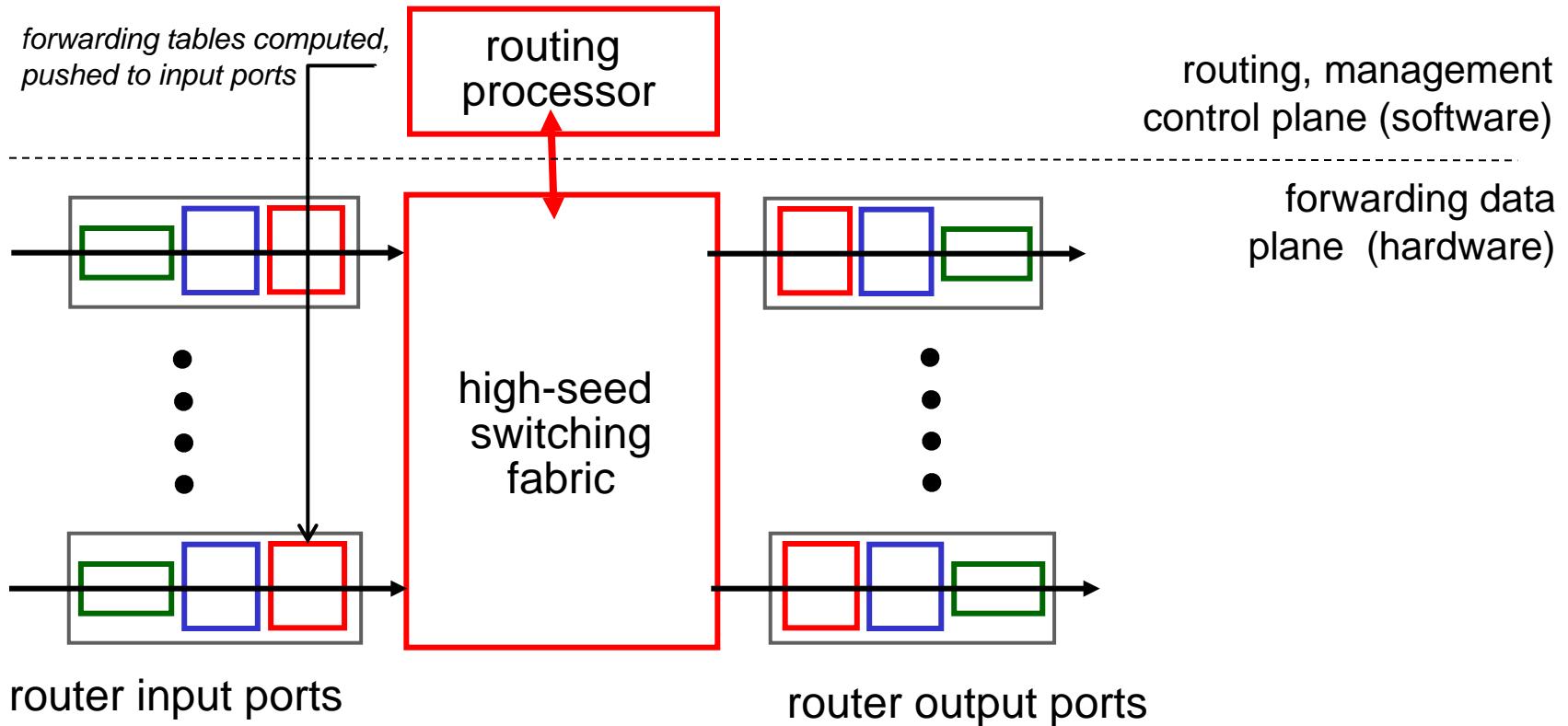
- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

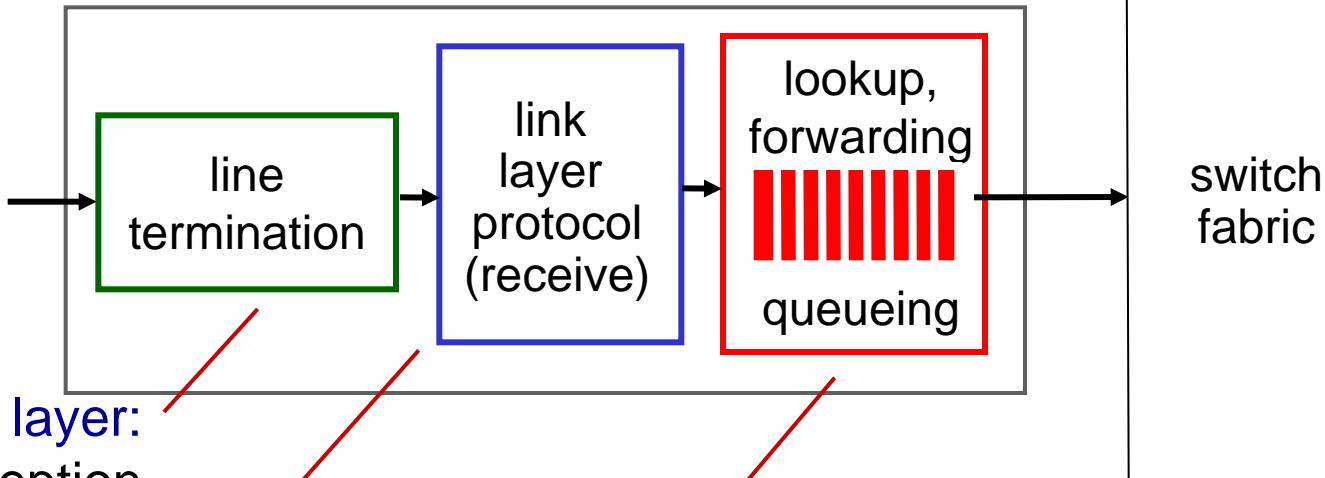
Router architecture overview

two key router functions:

- ❖ run routing algorithms/protocol (RIP, OSPF, BGP)
- ❖ *forwarding* datagrams from incoming to outgoing link



Input port functions



physical layer:

bit-level reception

data link layer:

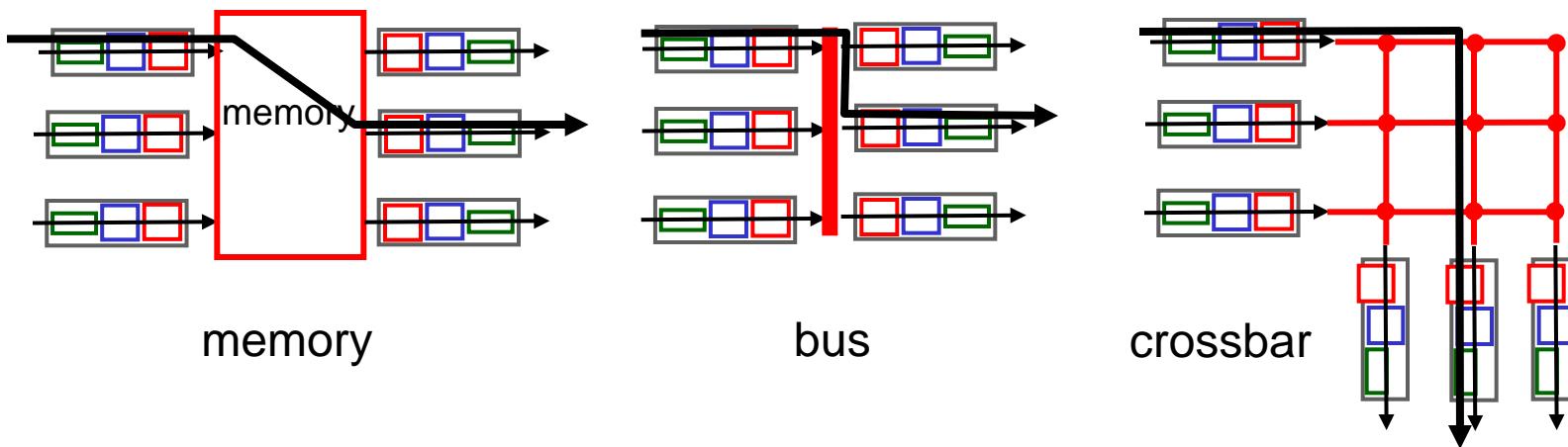
e.g., Ethernet
see chapter 5

decentralized switching:

- ❖ given datagram dest., lookup output port using forwarding table in input port memory (“*match plus action*”)
- ❖ goal: complete input port processing at ‘line speed’
- ❖ queuing: if datagrams arrive faster than forwarding rate into switch fabric

Switching fabrics

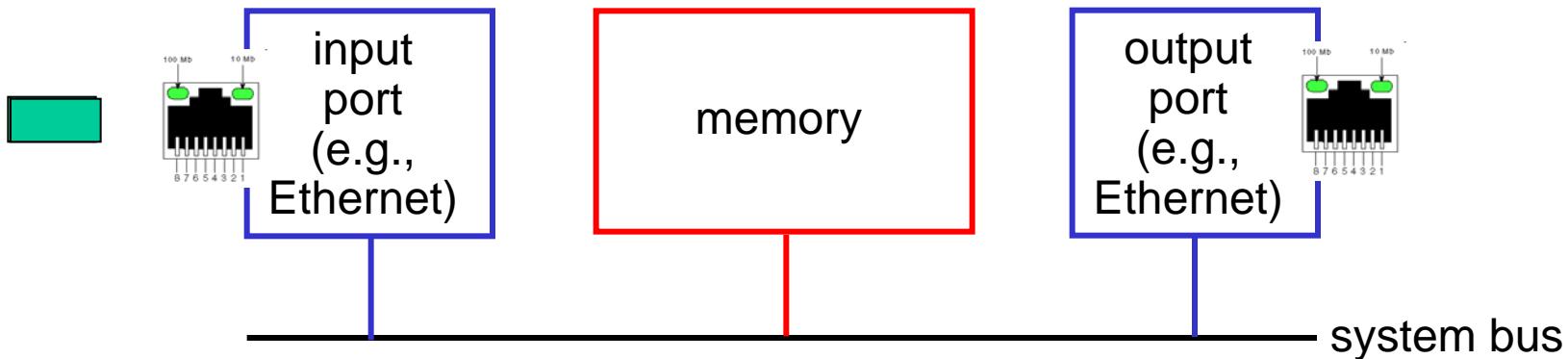
- ❖ transfer packet from input buffer to appropriate output buffer
- ❖ switching rate: rate at which packets can be transferred from inputs to outputs
 - often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate desirable
- ❖ three types of switching fabrics



Switching via memory

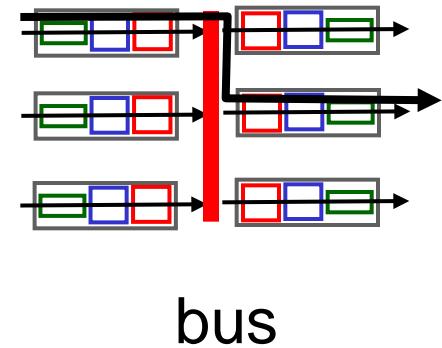
first generation routers:

- ❖ traditional computers with switching under direct control of CPU
- ❖ packet copied to system's memory
- ❖ speed limited by memory bandwidth (2 bus crossings per datagram)



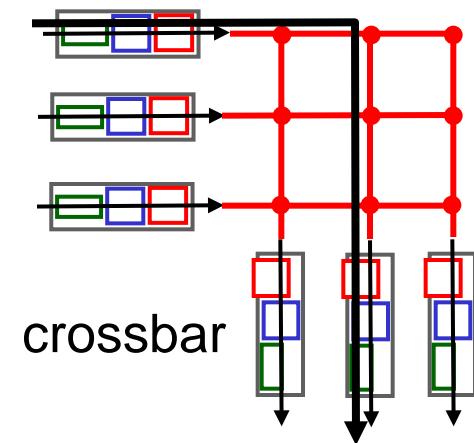
Switching via a bus

- ❖ datagram from input port memory to output port memory via a shared bus
- ❖ *bus contention*: switching speed limited by bus bandwidth
- ❖ 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers

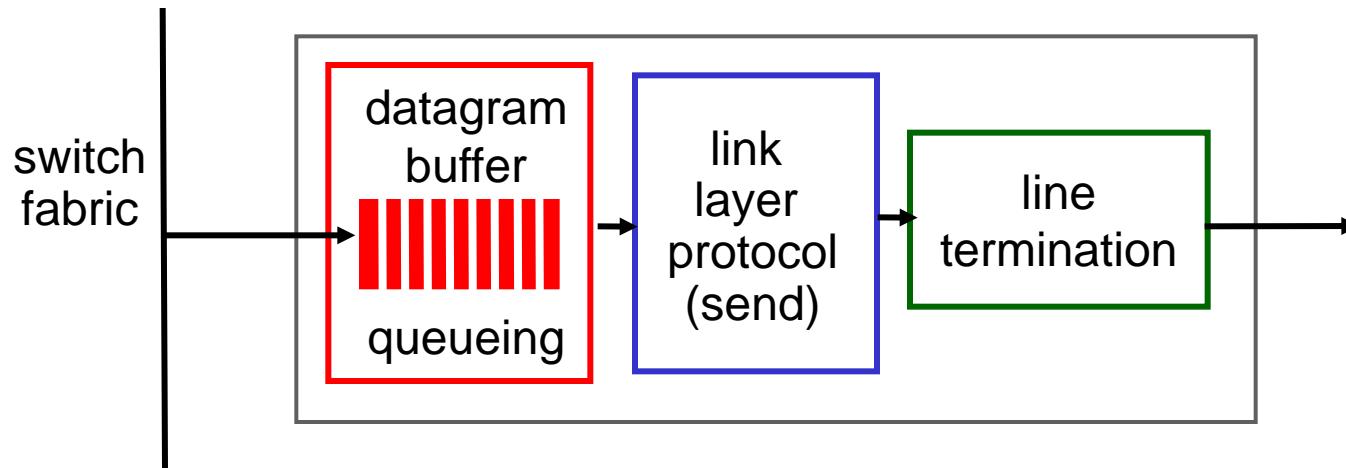


Switching via interconnection network

- ❖ overcome bus bandwidth limitations
- ❖ Crossbar and other interconnection nets initially developed to connect processors in multiprocessor
- ❖ advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- ❖ Cisco 12000: switches 60 Gbps through the interconnection network

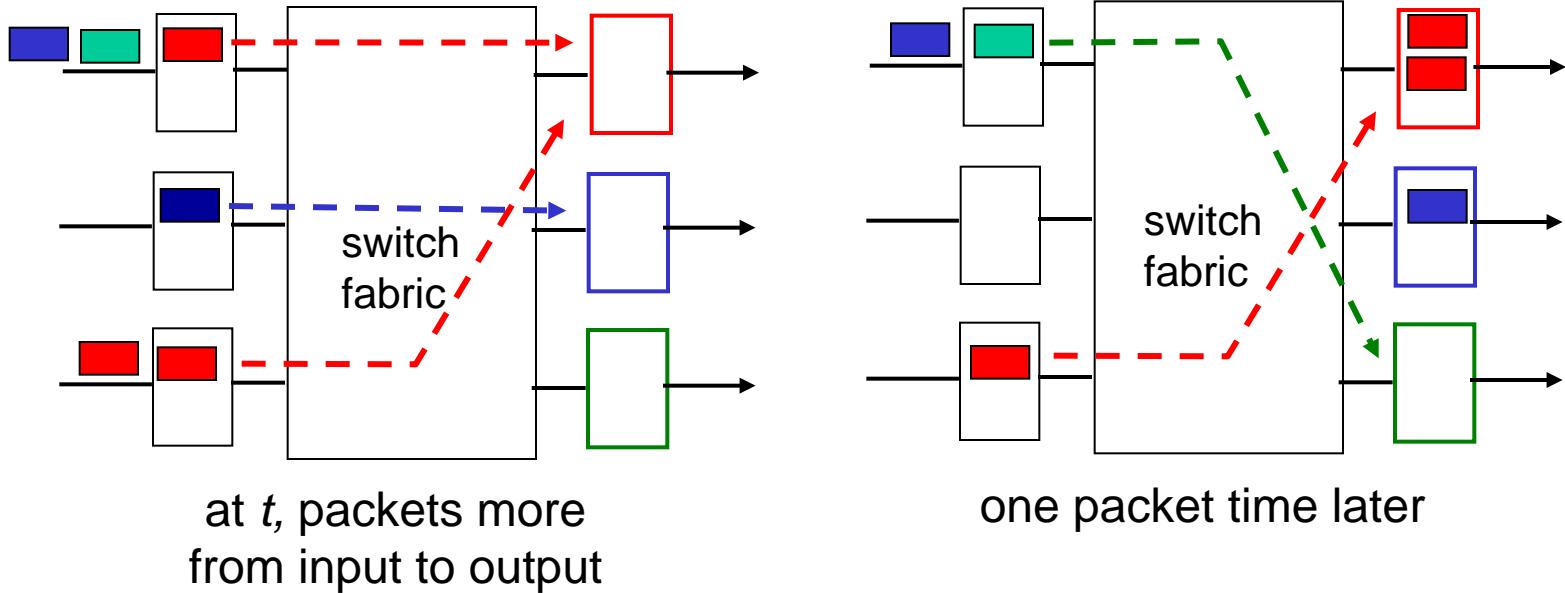


Output ports



- ❖ *buffering* required when datagrams arrive from fabric faster than the transmission rate
- ❖ *scheduling discipline* chooses among queued datagrams for transmission

Output port queueing



- ❖ buffering when arrival rate via switch exceeds output line speed
- ❖ *queueing (delay) and loss due to output port buffer overflow!*

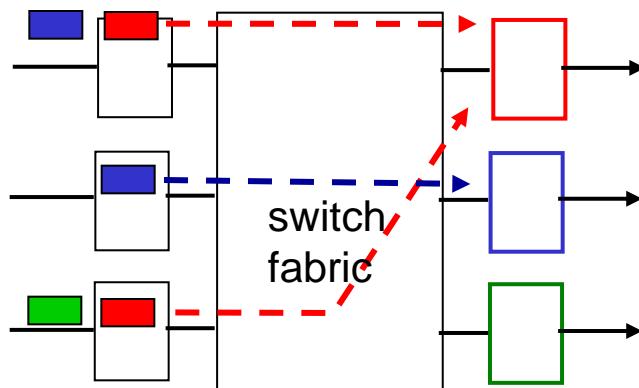
How much buffering?

- ❖ RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
 - e.g., $C = 10 \text{ Gpbs}$ link: 2.5 Gbit buffer
- ❖ recent recommendation: with N flows, buffering equal to

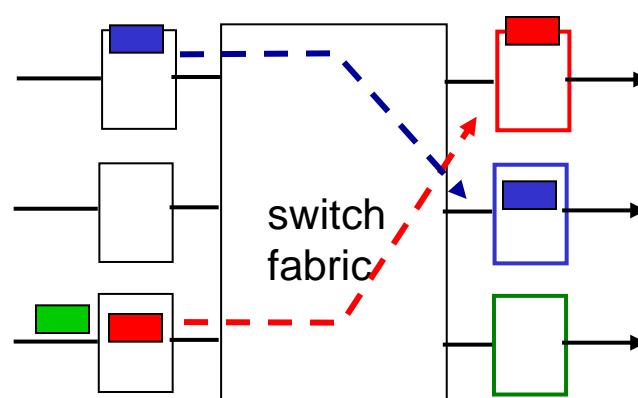
$$\frac{\text{RTT} \cdot C}{\sqrt{N}}$$

Input port queuing

- ❖ fabric slower than input ports combined -> queueing may occur at input queues
 - *queueing delay and loss due to input buffer overflow!*
- ❖ **Head-of-the-Line (HOL) blocking:** queued datagram at front of queue prevents others in queue from moving forward



output port contention:
only one red datagram can be
transferred.
lower red packet is blocked



one packet time later:
green packet
experiences HOL
blocking

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

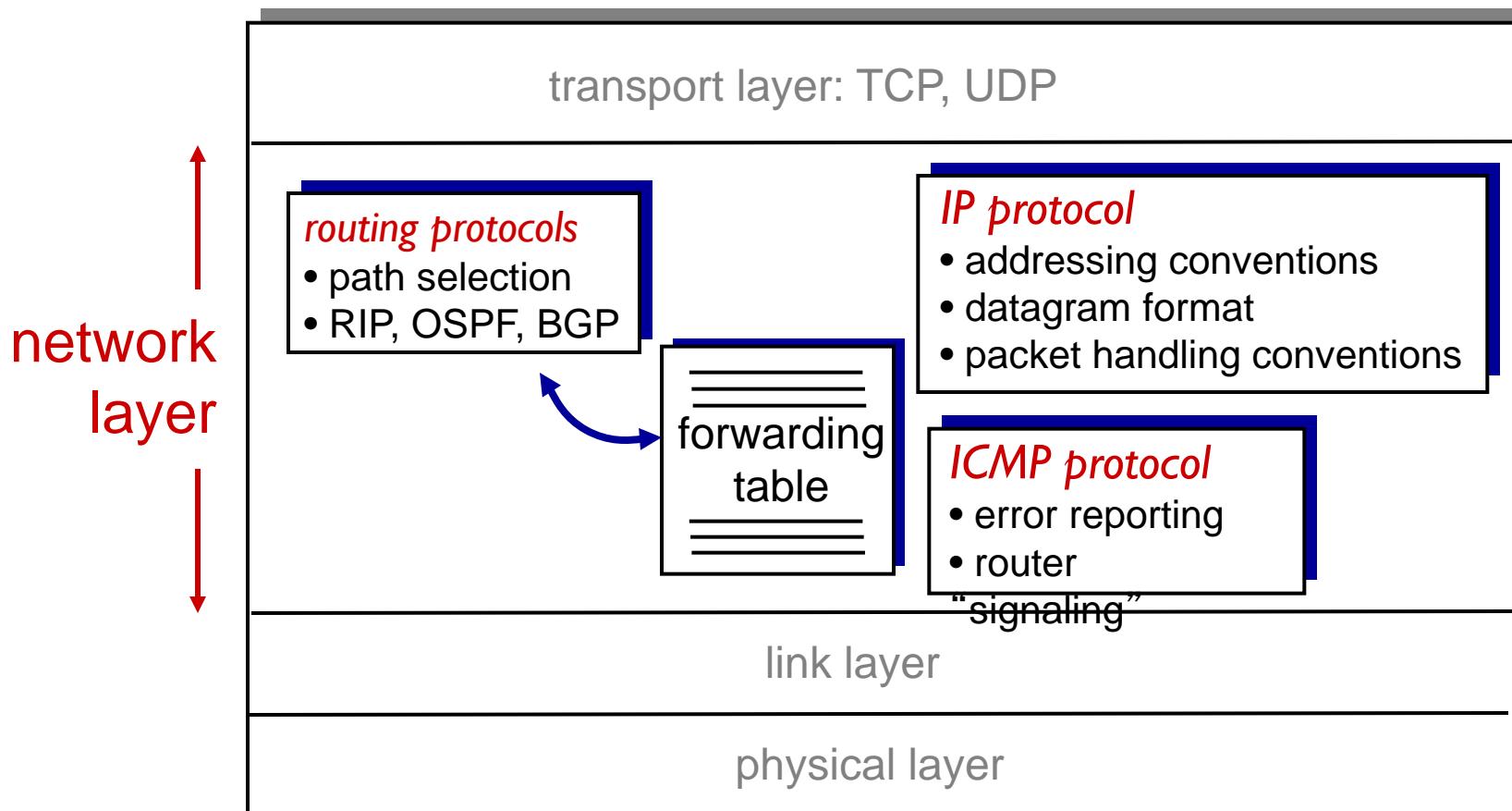
4.6 routing in the Internet

- RIP
- OSPF
- BGP

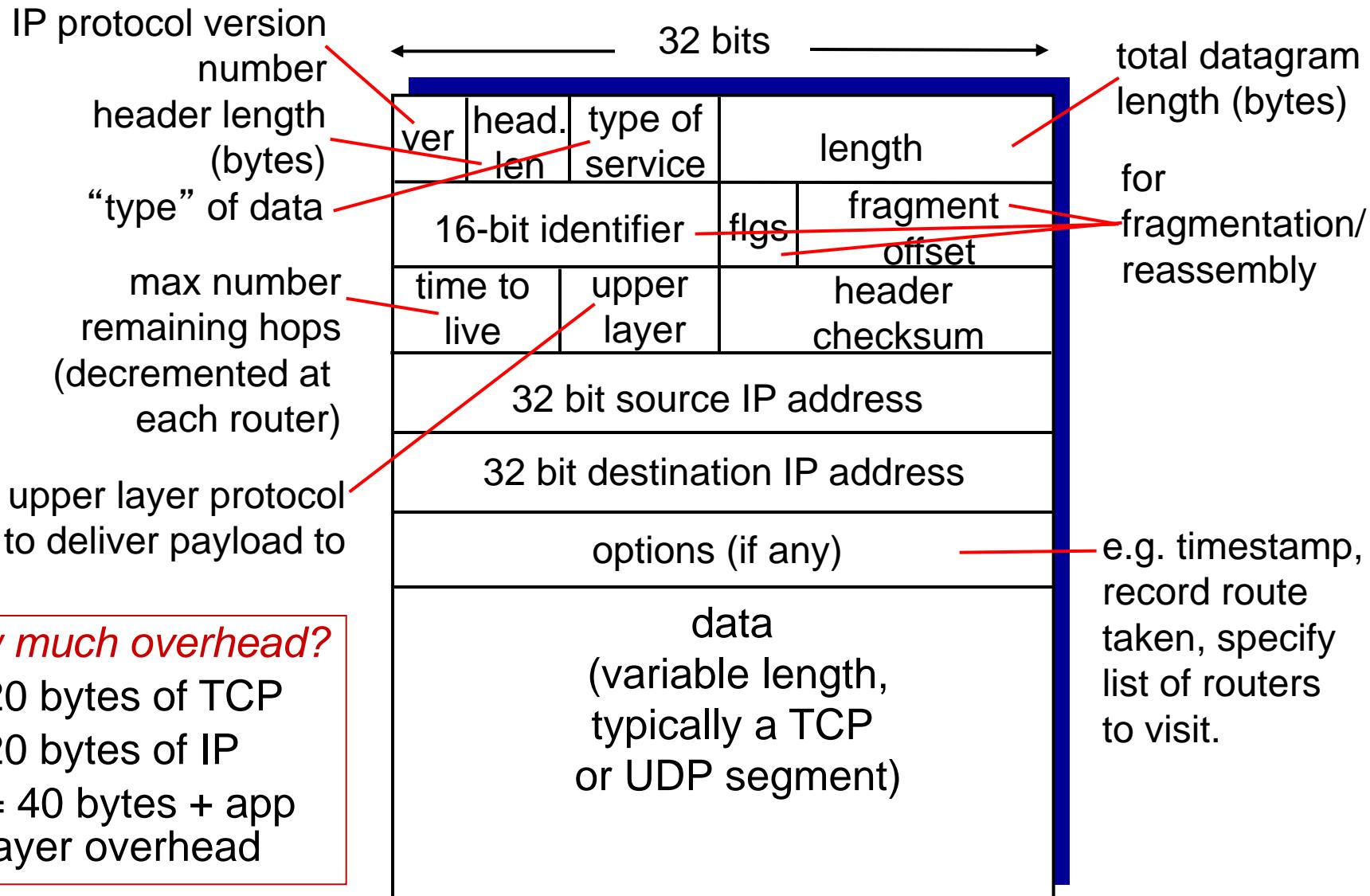
4.7 broadcast and multicast
routing

The Internet network layer

host, router network layer functions:

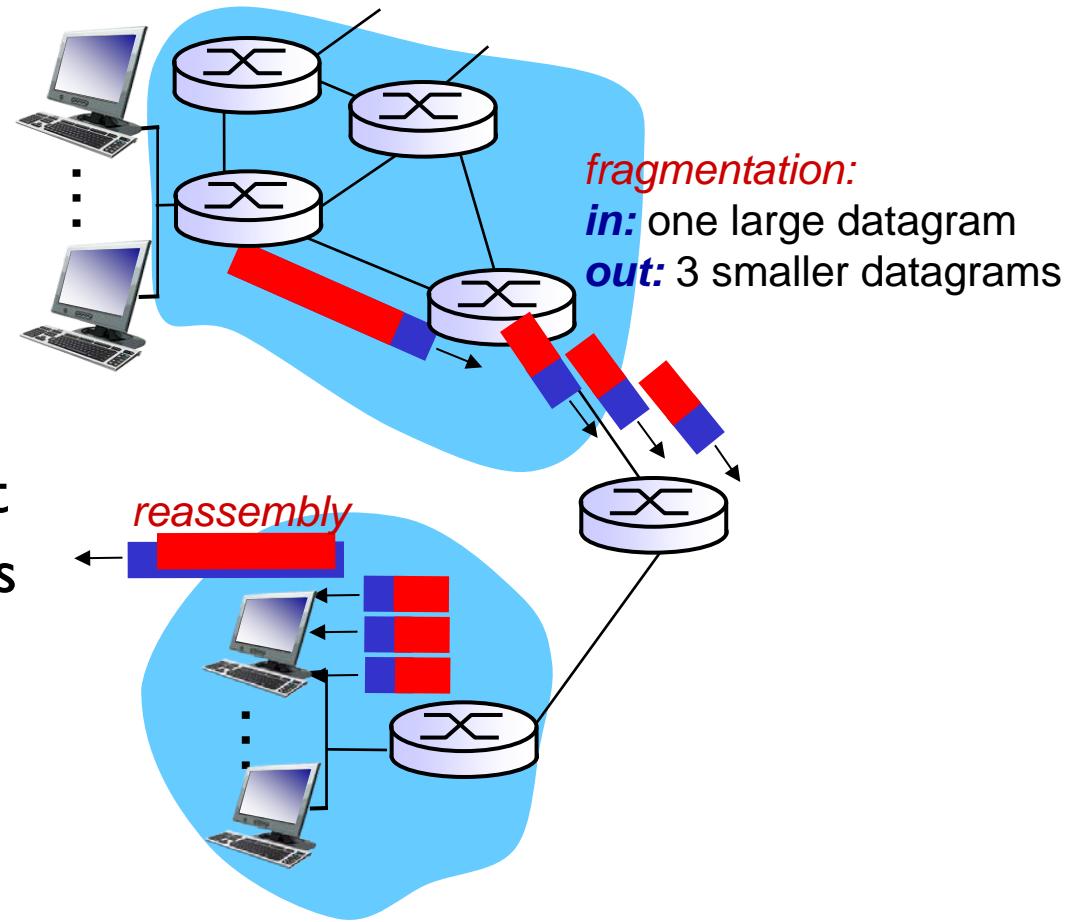


IP datagram format



IP fragmentation, reassembly

- ❖ network links have MTU (max.transfer size) - largest possible link-level frame
 - different link types, different MTUs
- ❖ large IP datagram divided (“fragmented”) within net
 - one datagram becomes several datagrams
 - “reassembled” only at final destination
 - IP header bits used to identify, order related fragments



IP fragmentation, reassembly

example:

- ❖ 4000 byte datagram
- ❖ MTU = 1500 bytes

1480 bytes in
data field

offset =
 $1480/8$

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

*one large datagram becomes
several smaller datagrams*

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

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

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

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

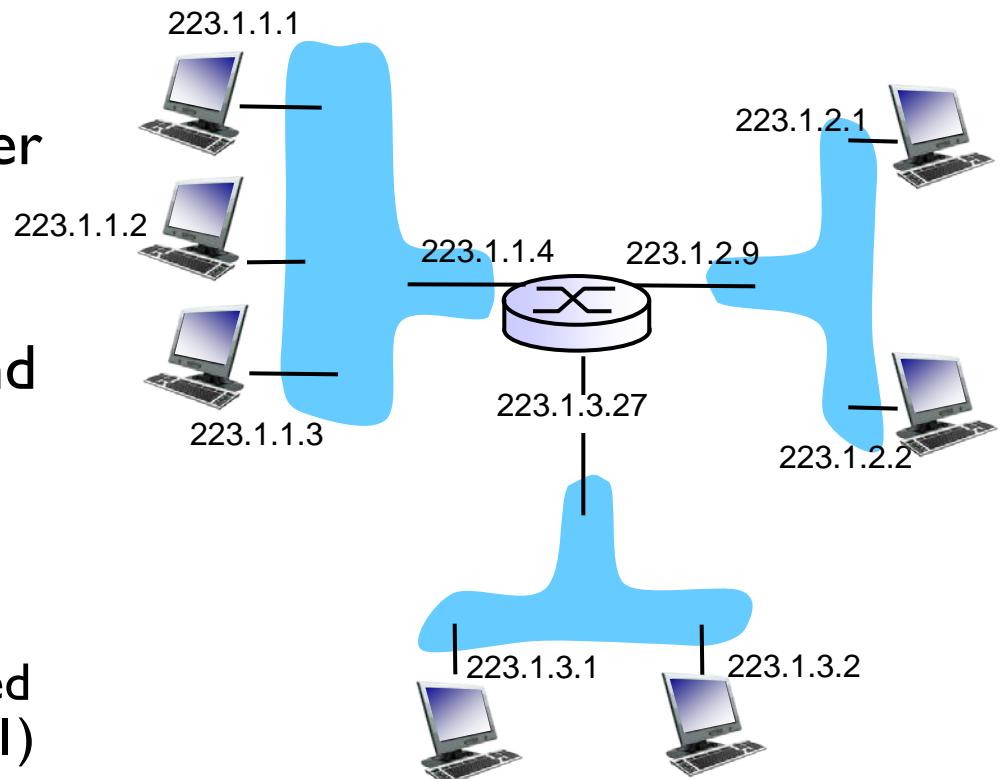
4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

IP addressing: introduction

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



$223.1.1.1 = \underbrace{11011111}_\text{223} \underbrace{00000001}_\text{1} \underbrace{00000001}_\text{1} \underbrace{00000001}_\text{1}$

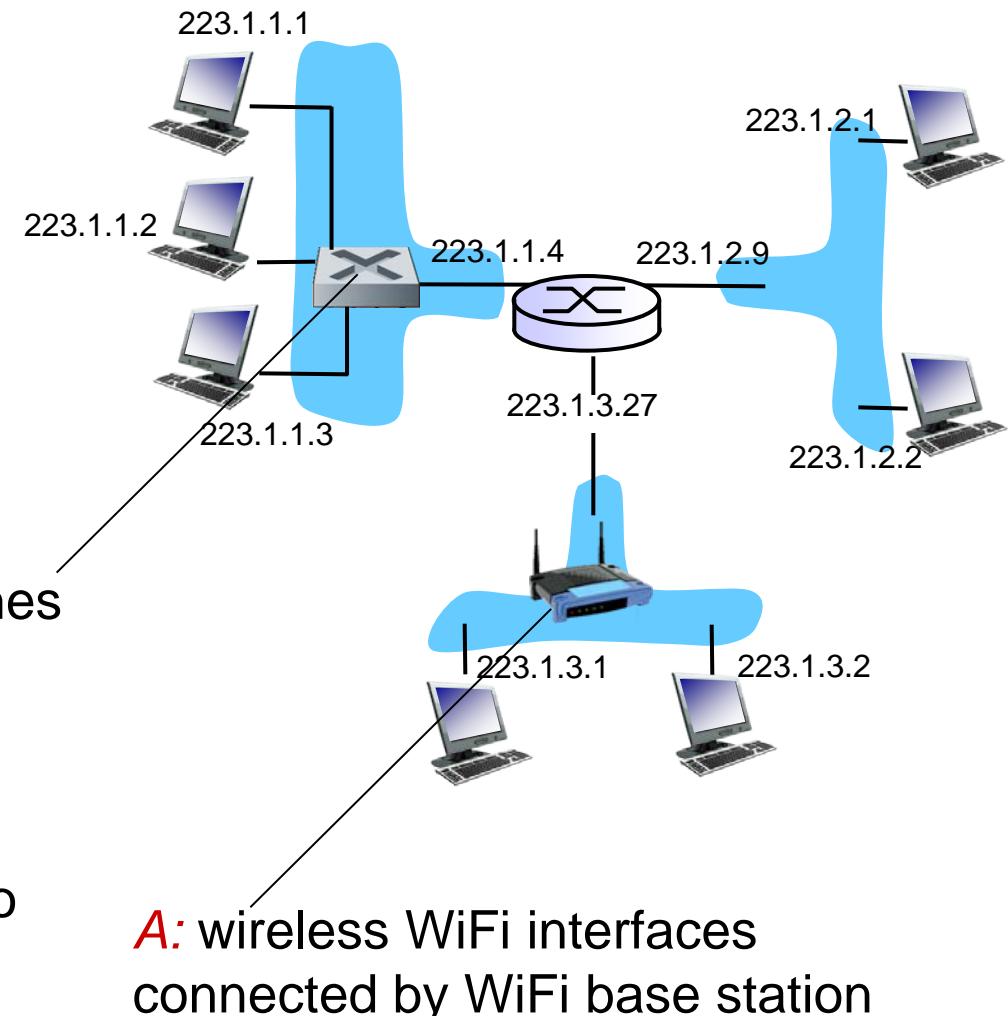
IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapter 5, 6.

A: wired Ethernet interfaces connected by Ethernet switches

For now: don't need to worry about how one interface is connected to another (with no intervening router)



A: wireless WiFi interfaces connected by WiFi base station

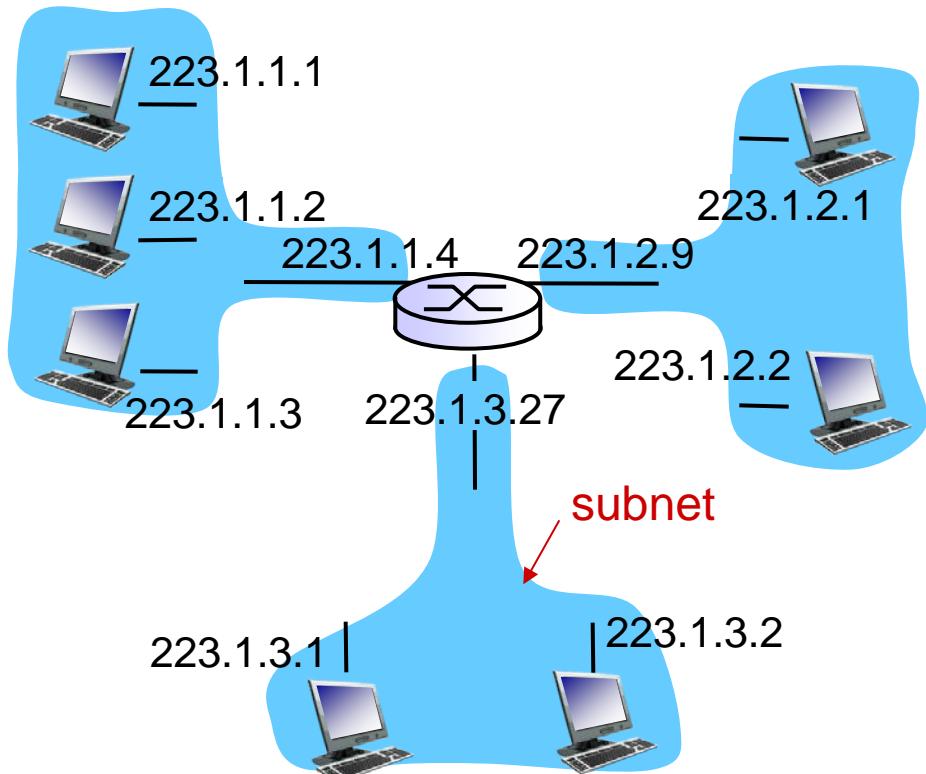
Subnets

❖ IP address:

- subnet part - high order bits
- host part - low order bits

❖ what's a subnet ?

- device interfaces with same subnet part of IP address
- can physically reach each other *without intervening router*

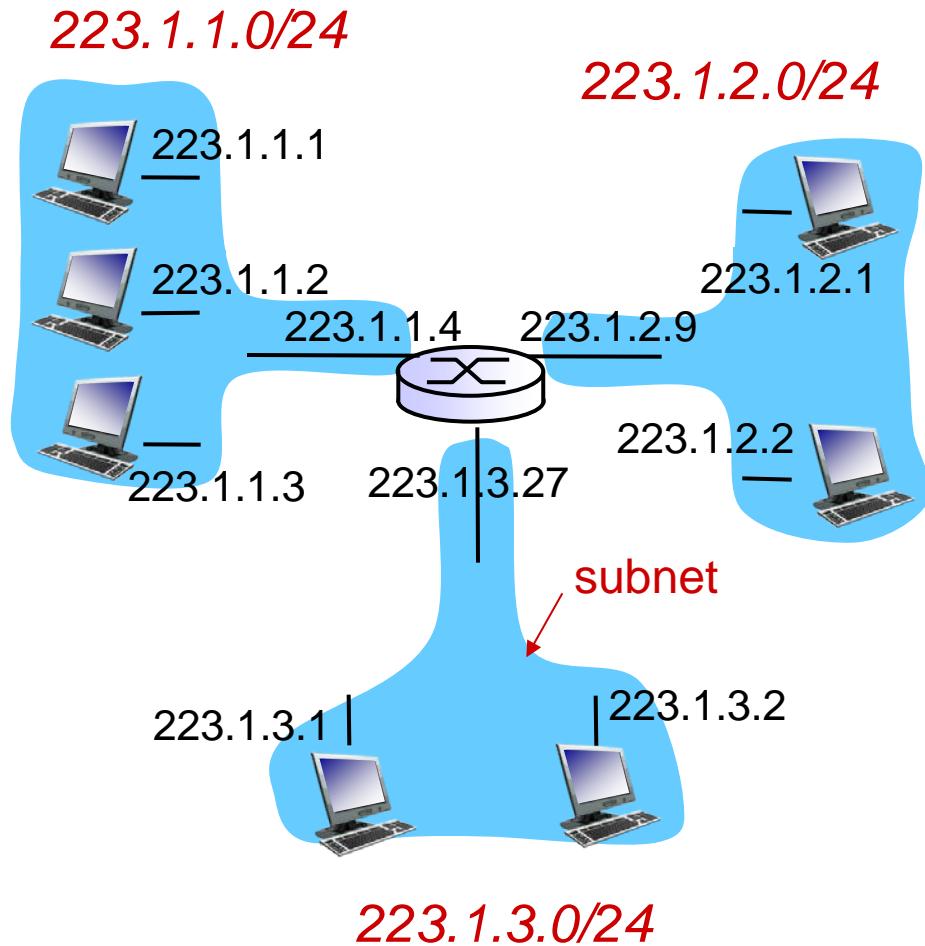


network consisting of 3 subnets

Subnets

recipe

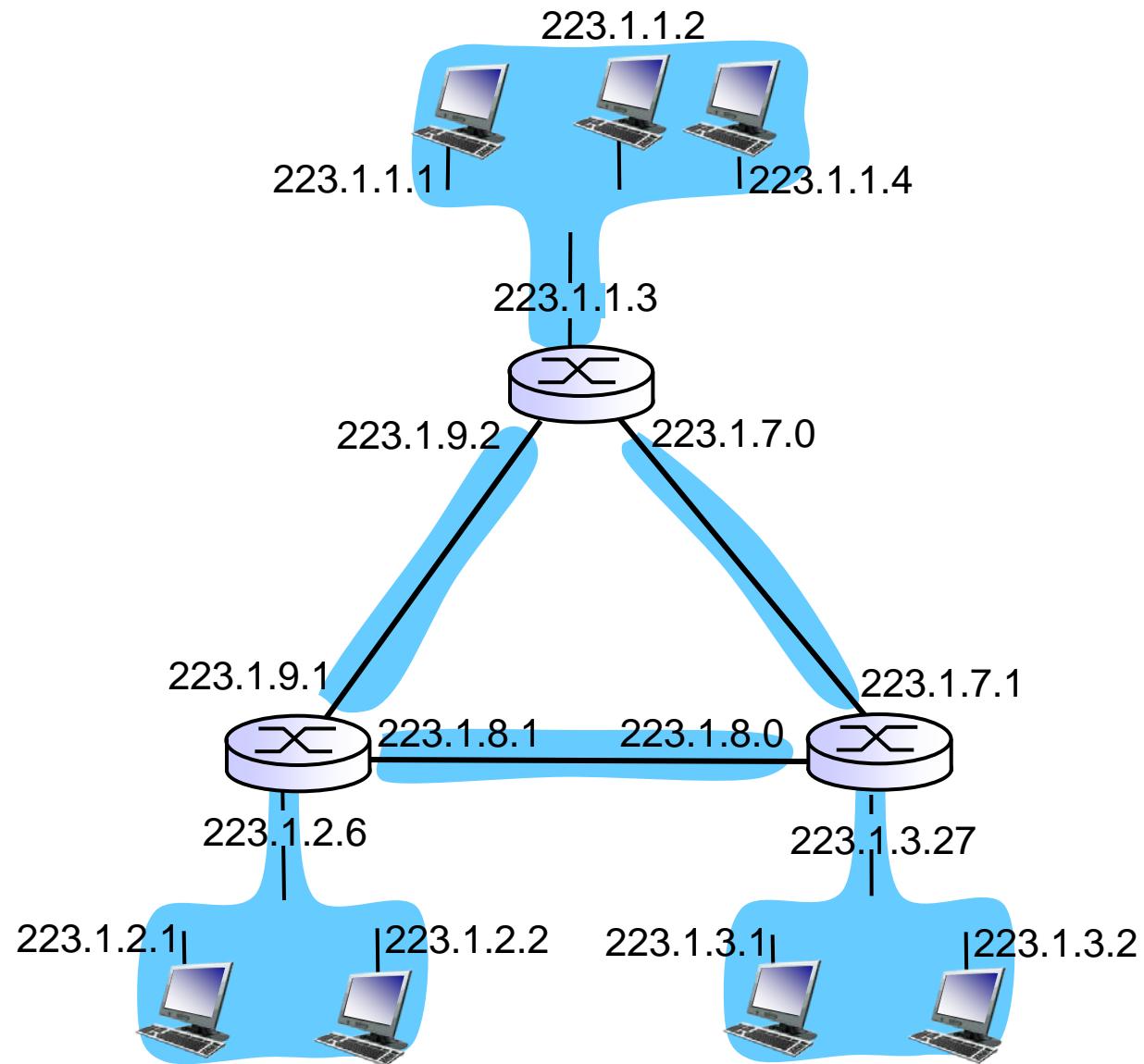
- ❖ to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- ❖ each isolated network is called a *subnet*



subnet mask: /24

Subnets

how many?



IP addressing: CIDR

CIDR: Classless Inter Domain Routing

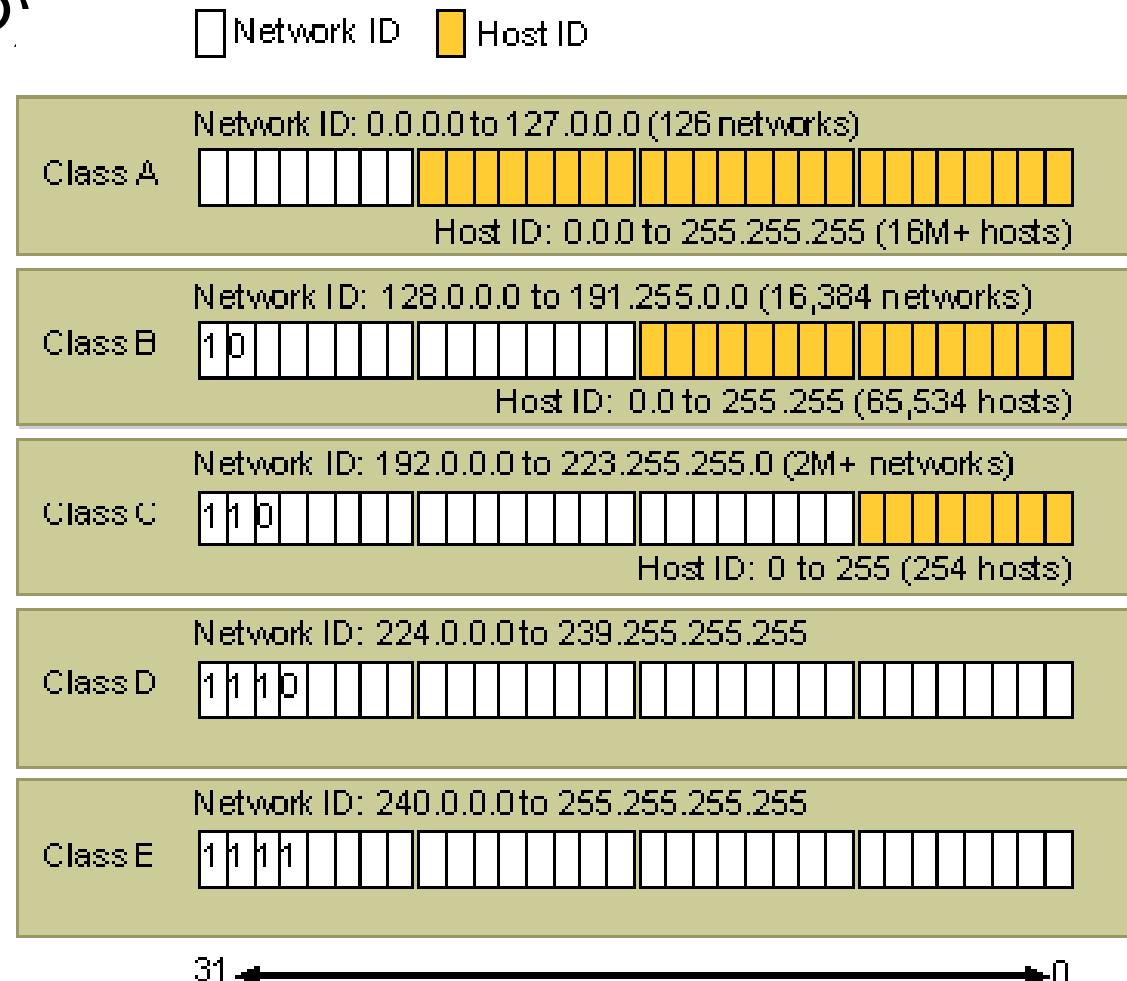
- subnet portion of address of arbitrary length
- address format: **a.b.c.d/x**, where x is # bits in subnet portion of address



Network Layer

❖ IP addresses are **hierarchical!**

- Prefix (Network ID)
- Suffix (Host ID)
- 121.21.21.21
- 192.92.92.0/24



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 a server
 - “plug-and-play”

DHCP: Dynamic Host Configuration Protocol

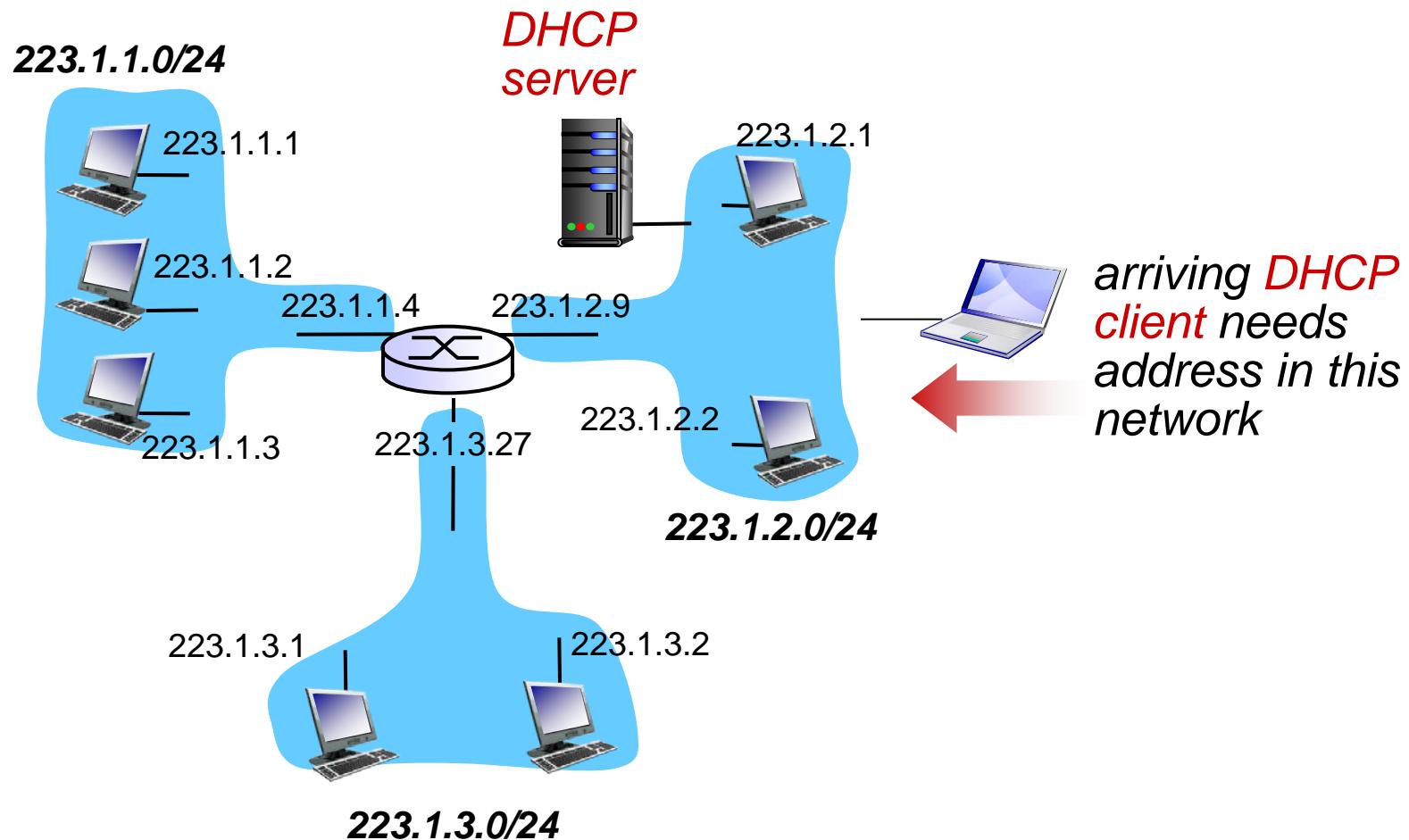
goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- 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

DHCP client-server scenario



DHCP client-server scenario

DHCP server: 223.1.2.5



DHCP discover

src : 0.0.0.0, 68
dest.: 255.255.255.255,67
yiaddr: 0.0.0.0
transaction ID: 654

arriving
client



DHCP offer

src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 654
lifetime: 3600 secs

DHCP request

src: 0.0.0.0, 68
dest:: 255.255.255.255, 67
yiaddr: 223.1.2.4
transaction ID: 655
lifetime: 3600 secs

DHCP ACK

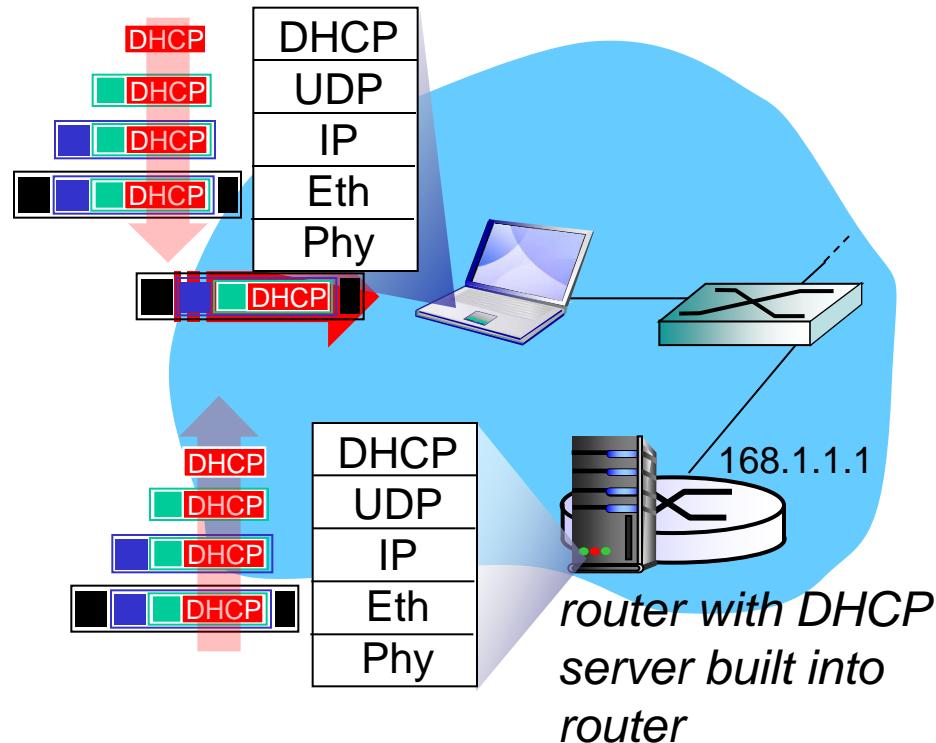
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 655
lifetime: 3600 secs

DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

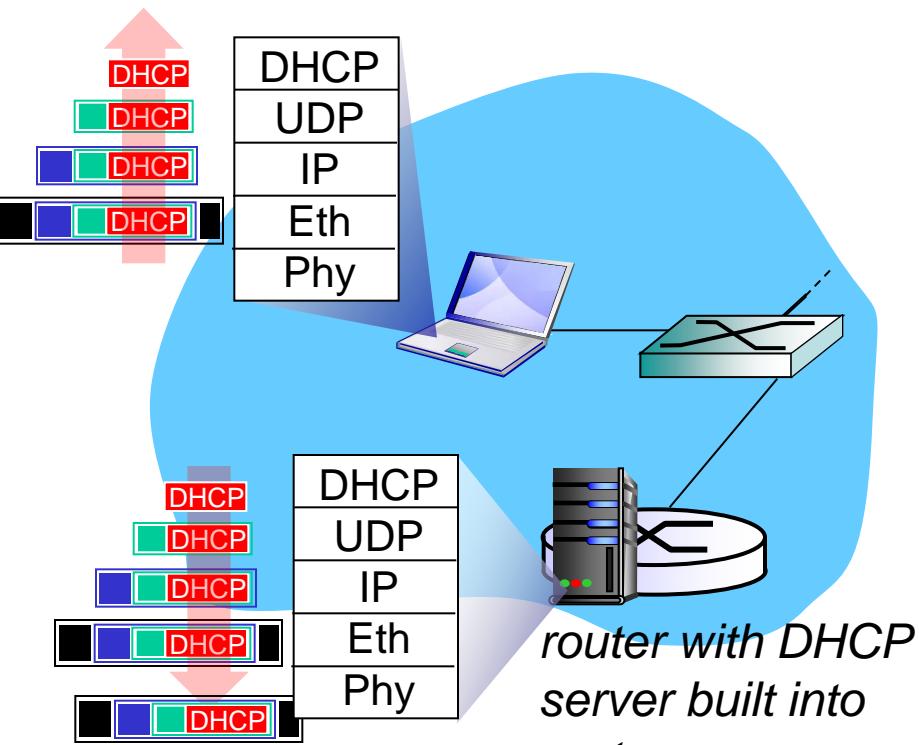
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

DHCP: example



- ❖ connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- ❖ DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- ❖ Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- ❖ Ethernet demuxed to IP, demuxed to UDP, demuxed to DHCP

DHCP: example



- ❖ DHCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- ❖ encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client
- ❖ client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

DHCP: Wireshark output (home LAN)

Message type: **Boot Request (1)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

Transaction ID: 0x6b3a11b7

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

Client IP address: 0.0.0.0 (0.0.0.0)

Your (client) IP address: 0.0.0.0 (0.0.0.0)

Next server IP address: 0.0.0.0 (0.0.0.0)

Relay agent IP address: 0.0.0.0 (0.0.0.0)

Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)

Server host name not given

Boot file name not given

Magic cookie: (OK)

Option: (t=53,l=1) **DHCP Message Type = DHCP Request**

Option: (61) Client identifier

Length: 7; Value: 010016D323688A;

Hardware type: Ethernet

Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)

Option: (t=50,l=4) Requested IP Address = 192.168.1.101

Option: (t=12,l=5) Host Name = "nomad"

Option: (55) Parameter Request List

Length: 11; Value: 010F03062C2E2F1F21F92B

1 = Subnet Mask; 15 = Domain Name

3 = Router; 6 = Domain Name Server

44 = NetBIOS over TCP/IP Name Server

.....

request

reply

Message type: **Boot Reply (2)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

Transaction ID: 0x6b3a11b7

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

Client IP address: 192.168.1.101 (192.168.1.101)

Your (client) IP address: 0.0.0.0 (0.0.0.0)

Next server IP address: 192.168.1.1 (192.168.1.1)

Relay agent IP address: 0.0.0.0 (0.0.0.0)

Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)

Server host name not given

Boot file name not given

Magic cookie: (OK)

Option: (t=53,l=1) DHCP Message Type = DHCP ACK

Option: (t=54,l=4) Server Identifier = 192.168.1.1

Option: (t=1,l=4) Subnet Mask = 255.255.255.0

Option: (t=3,l=4) Router = 192.168.1.1

Option: (6) Domain Name Server

Length: 12; Value: 445747E2445749F244574092;

IP Address: 68.87.71.226;

IP Address: 68.87.73.242;

IP Address: 68.87.64.146

Option: (t=15,l=20) Domain Name = "hsd1.ma.comcast.net."

IP addresses: how to get one?

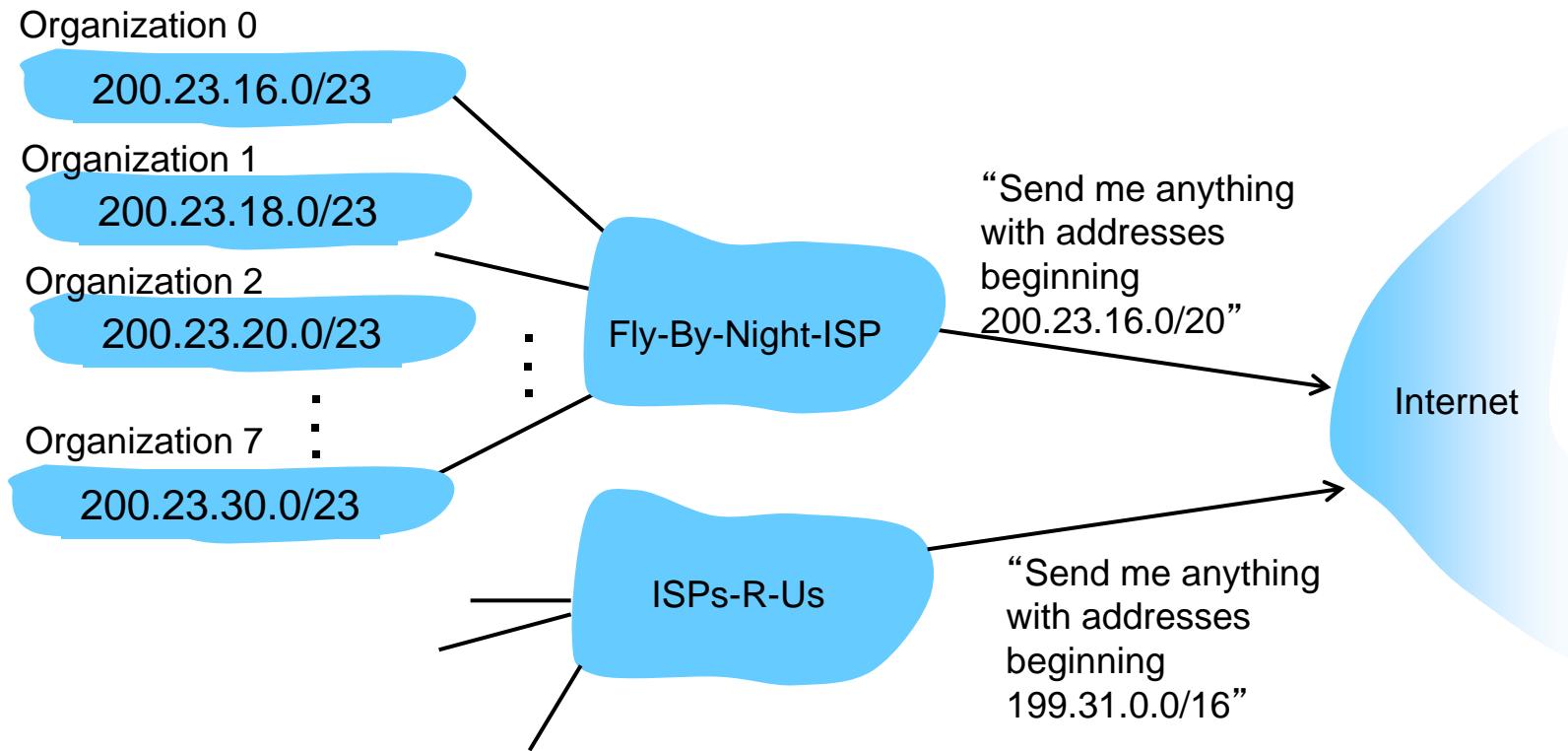
Q: how does *network* get subnet part of IP addr?

A: gets allocated portion of its provider ISP's address space

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

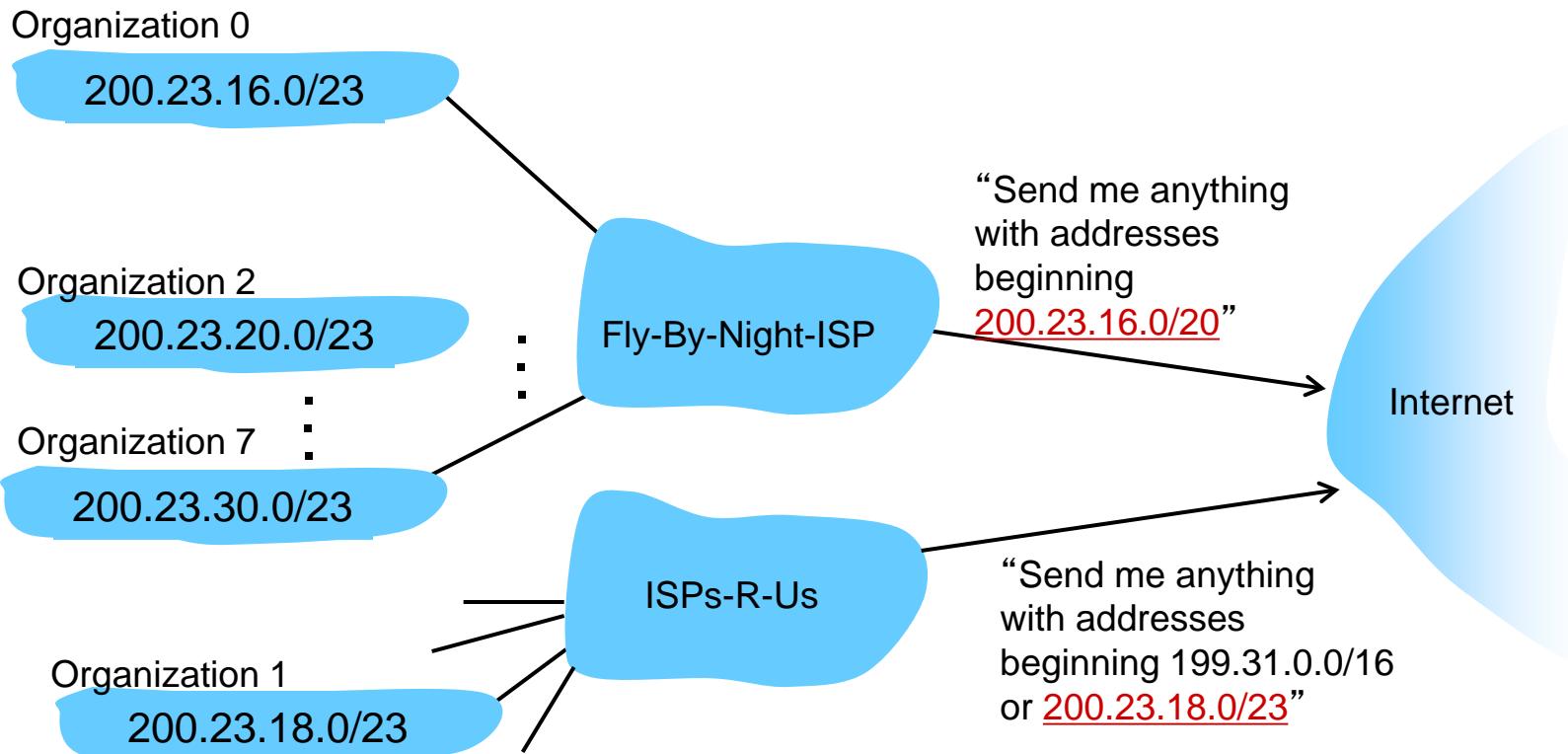
Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

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



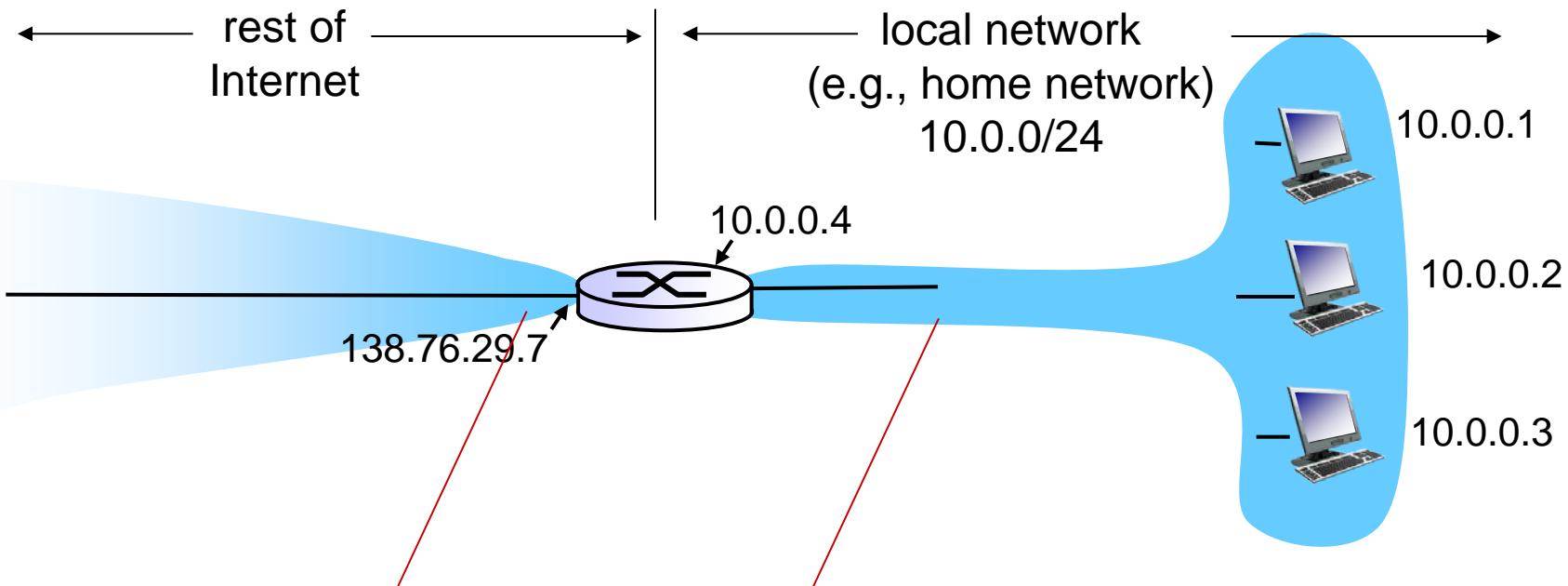
IP addressing: the last word...

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

NAT: network address translation



all datagrams *leaving* local network have *same* single source NAT IP address:
138.76.29.7, different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

NAT: network address translation

motivation: local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

NAT: network address translation

implementation: NAT router must:

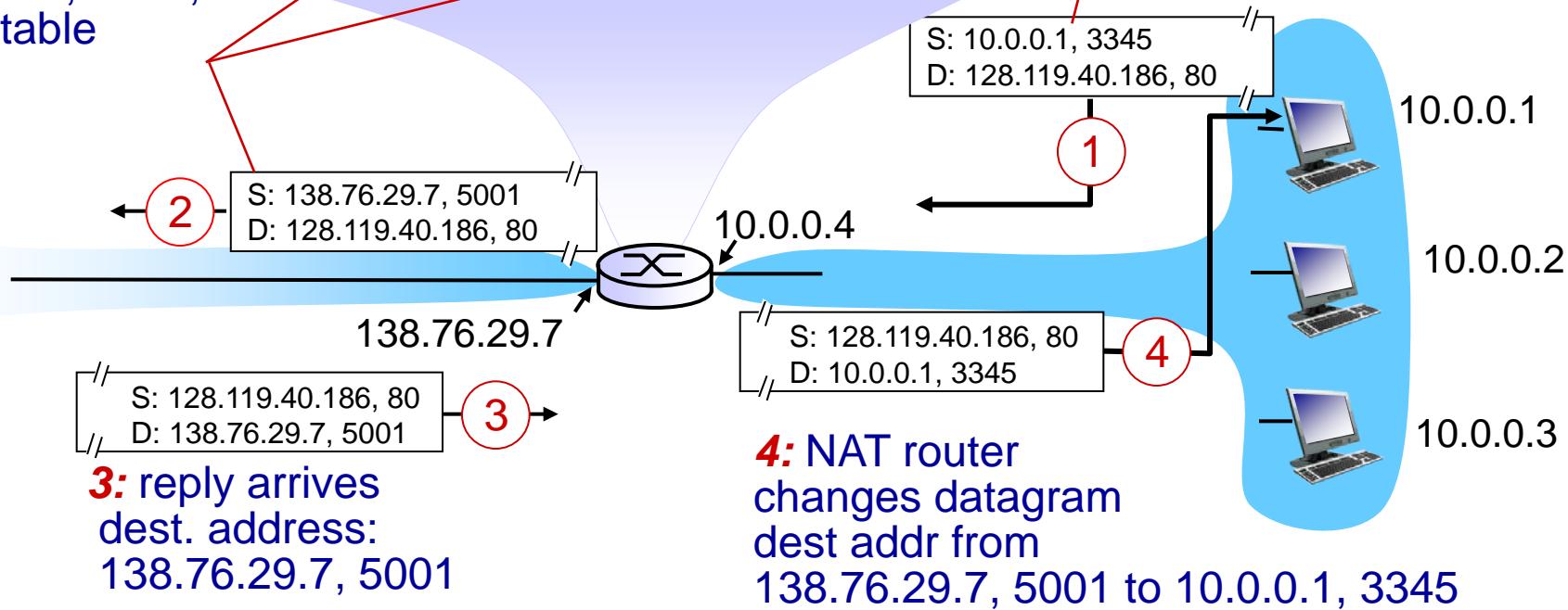
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
 . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr
- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT: network address translation

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
.....

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

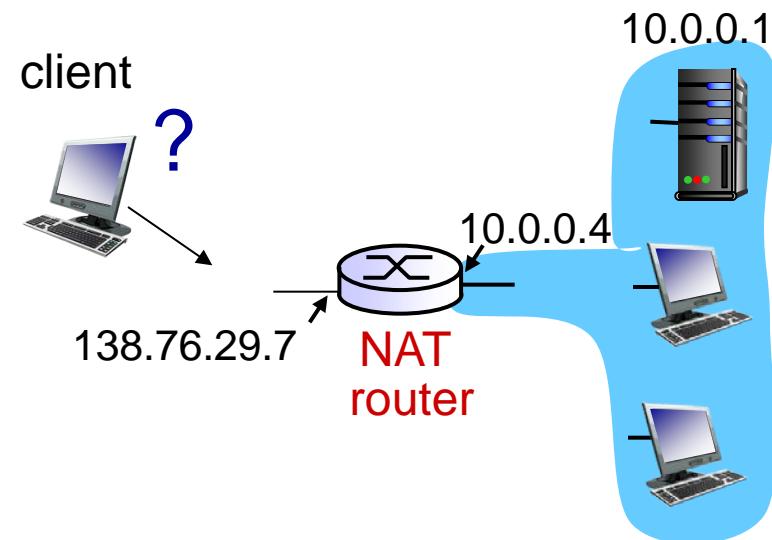


NAT: network address translation

- ❖ 16-bit port-number field:
 - 60,000 simultaneous connections with a single LAN-side address!
- ❖ NAT is controversial:
 - routers should only process up to layer 3
 - violates end-to-end argument
 - NAT possibility must be taken into account by app designers, e.g., P2P applications
 - address shortage should instead be solved by IPv6

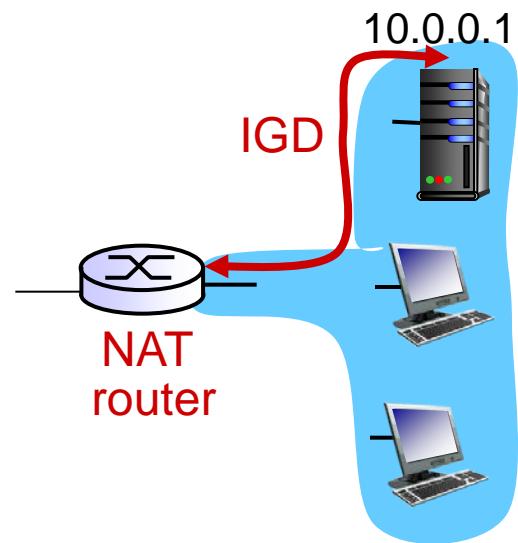
NAT traversal problem

- ❖ client wants to connect to server with address 10.0.0.1
 - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
 - only one externally visible NATed address: 138.76.29.7
- ❖ *solution 1:* statically configure NAT to forward incoming connection requests at given port to server
 - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



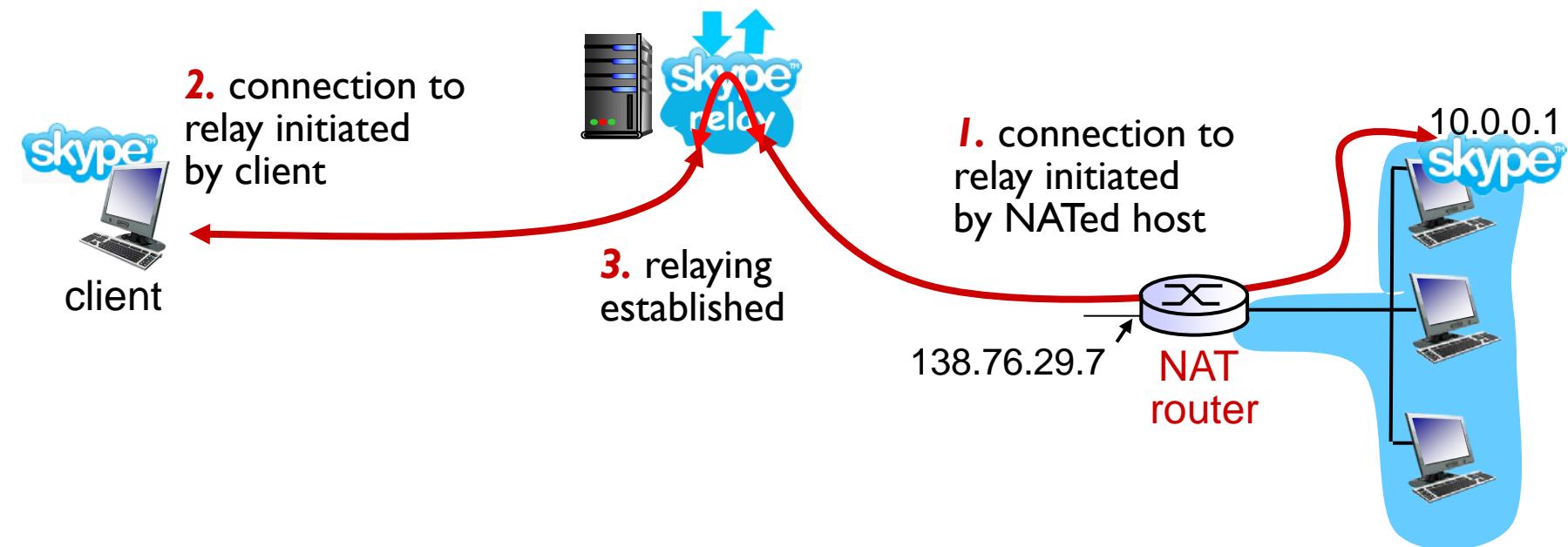
NAT traversal problem

- ❖ **solution 2:** Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
 - ❖ learn public IP address (138.76.29.7)
 - ❖ add/remove port mappings (with lease times)
- i.e., automate static NAT port map configuration



NAT traversal problem

- ❖ *solution 3:* relaying (used in Skype)
 - NATed client establishes connection to relay
 - external client connects to relay
 - relay bridges packets between two connections



Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

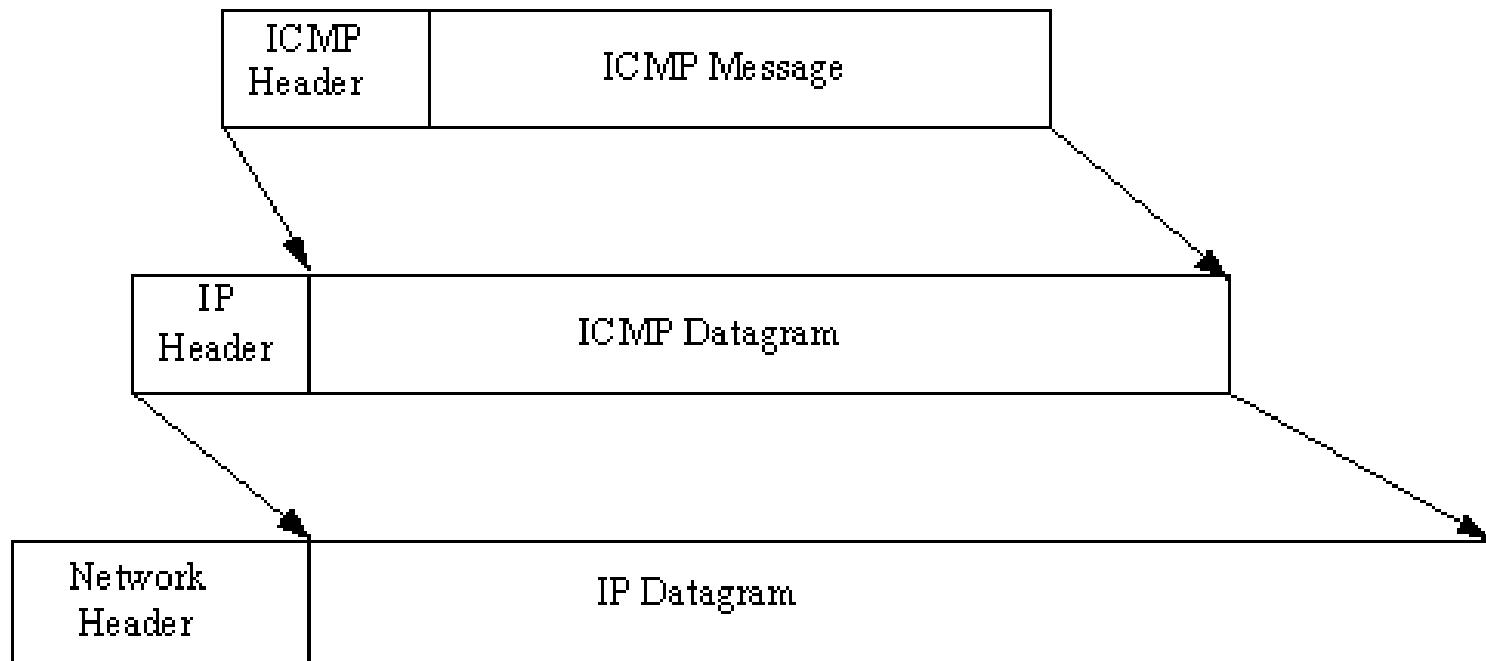
- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

ICMP: internet control message protocol

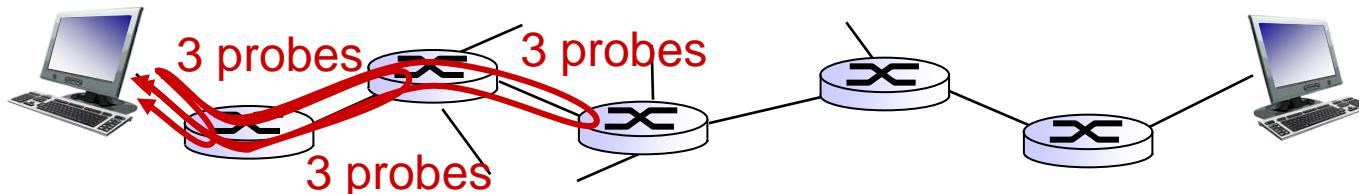
- ❖ used by hosts & routers to communicate network-level information
 - error reporting: unreachable host, network, port, protocol
 - echo request/reply (used by ping)
- ❖ network-layer “above” IP:
 - ICMP msgs carried in IP datagrams
- ❖ **ICMP message:** type, code plus first 8 bytes of IP datagram 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. (RFC6633))
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header



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 *n*th 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



IPv6: motivation

- ❖ *initial motivation:* 32-bit address space soon to be completely allocated.
- ❖ additional motivation:
 - header format helps speed processing/forwarding
 - header changes to facilitate QoS

IPv6 datagram format:

- fixed-length 40 byte header
- no fragmentation allowed

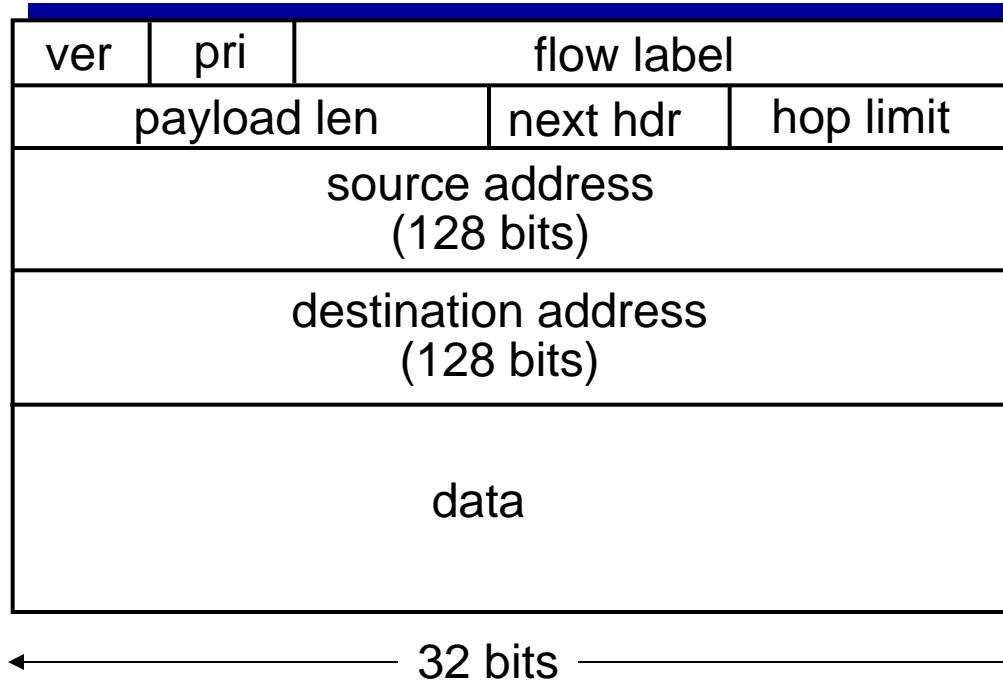
IPv6 datagram format

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

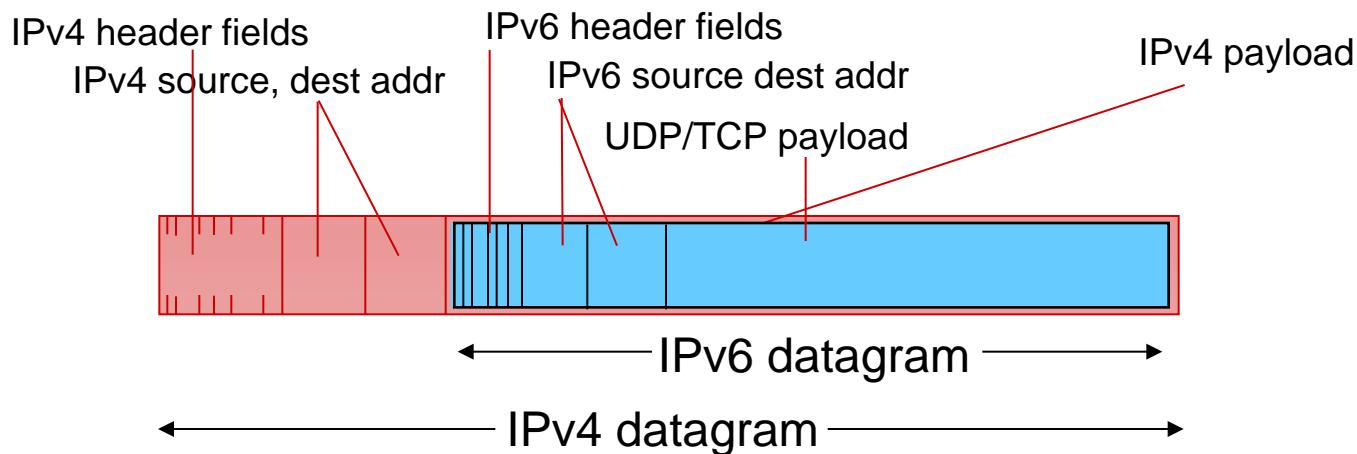


Other changes from IPv4

- ❖ *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

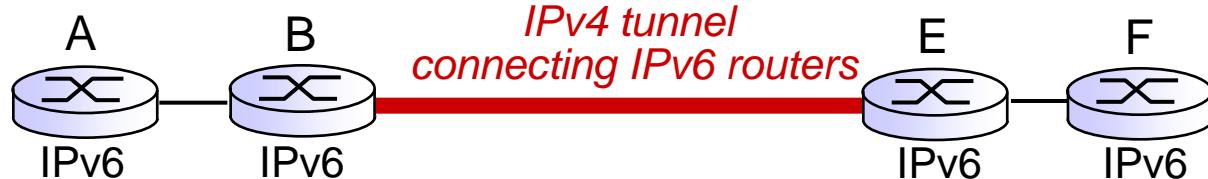
Transition from IPv4 to IPv6

- ❖ not all routers can be upgraded simultaneously
 - no “flag days”
 - how will network operate with mixed IPv4 and IPv6 routers?
- ❖ *tunneling*: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers

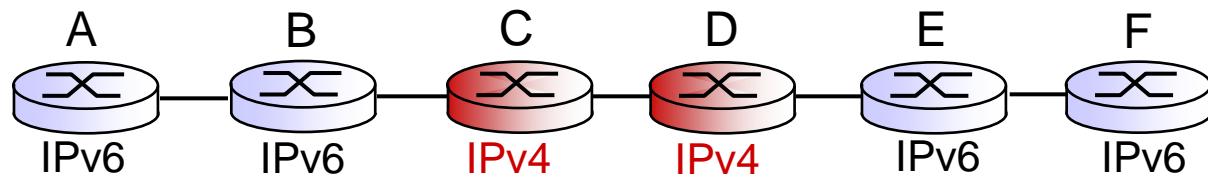


Tunneling

logical view:



physical view:

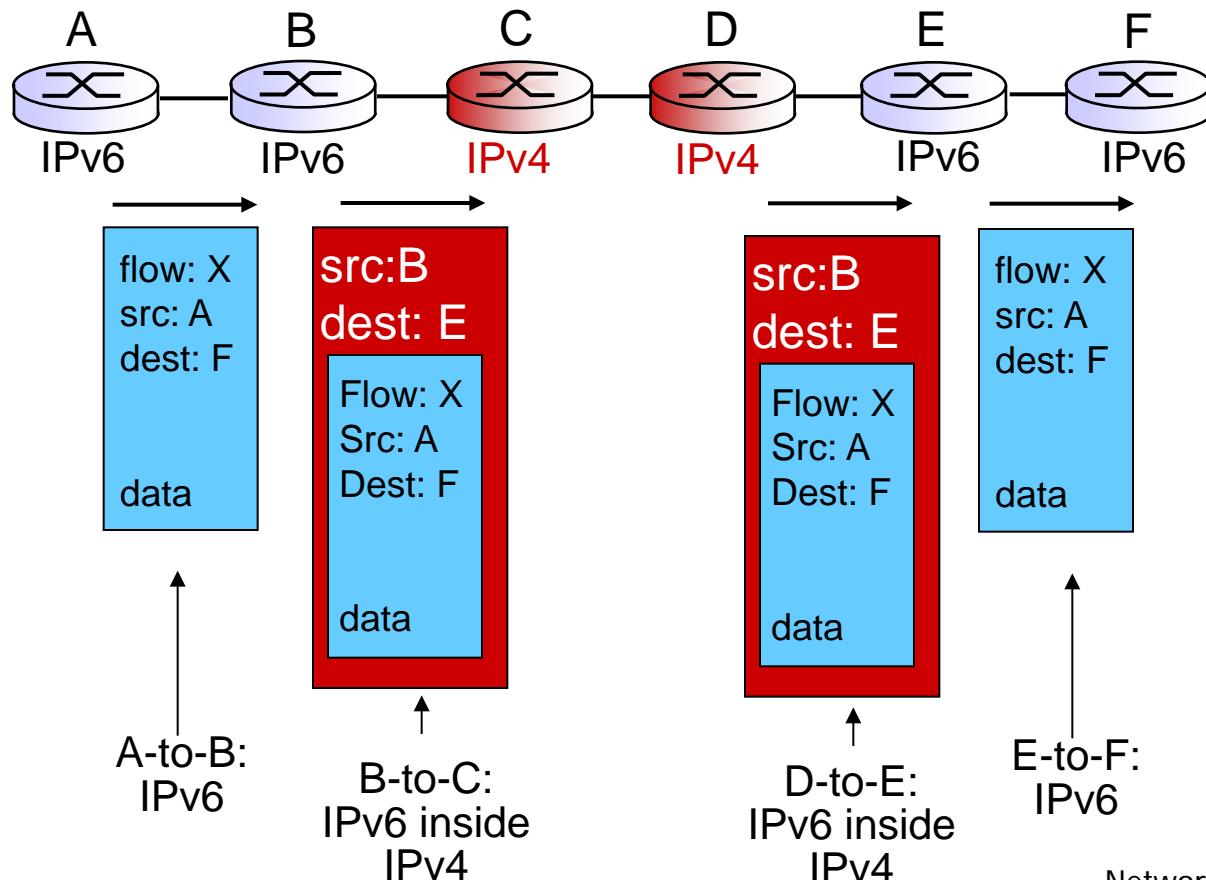


Tunneling

logical view:



physical view:



Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

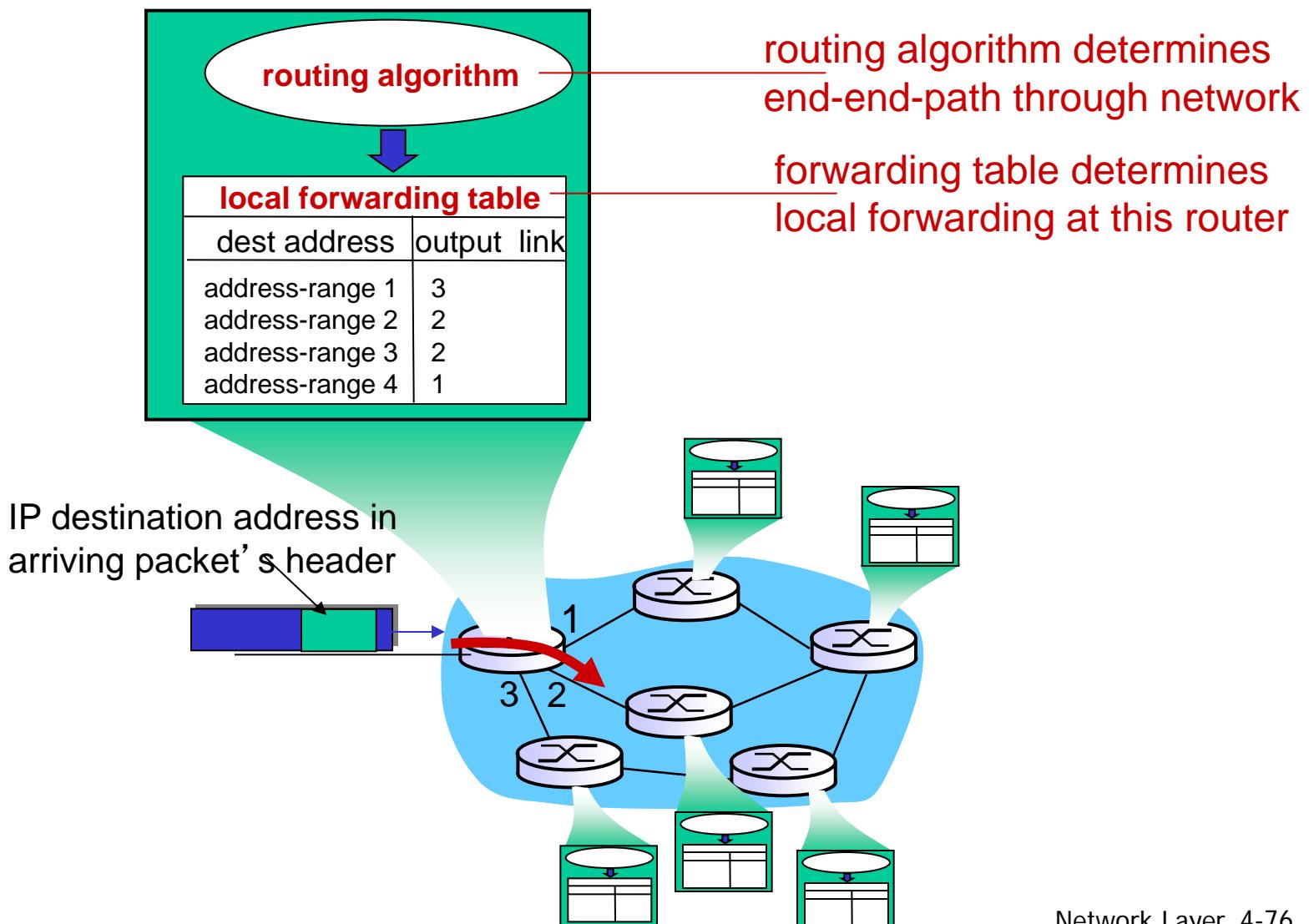
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

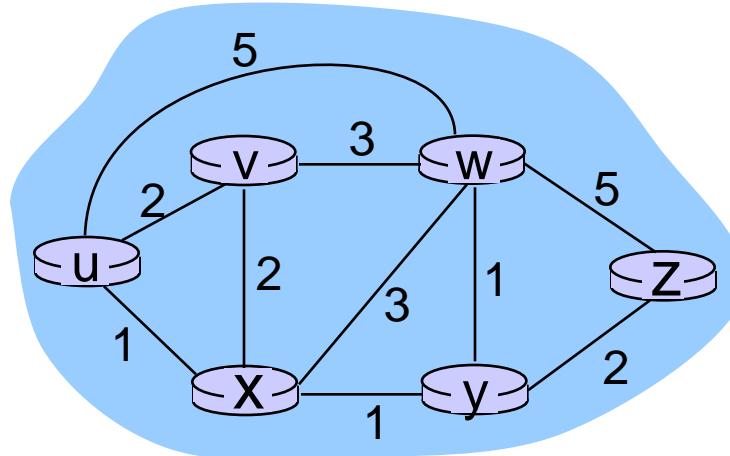
- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

Interplay between routing, forwarding



Graph abstraction



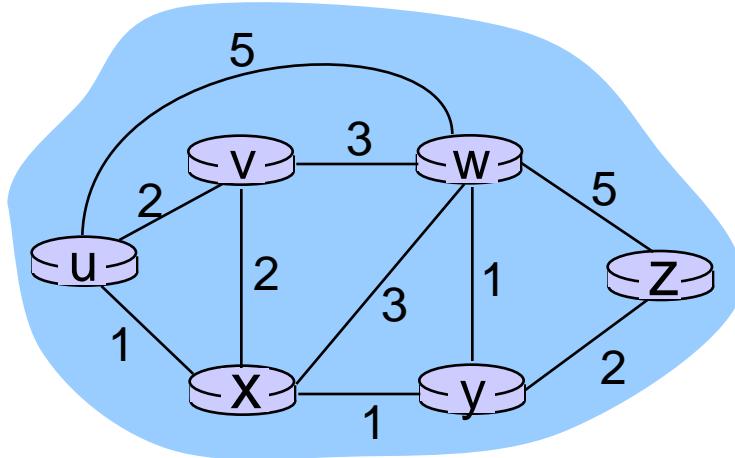
graph: $G = (N, E)$

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

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

aside: graph abstraction is useful in other network contexts, e.g., P2P, where N is set of peers and E is set of TCP connections

Graph abstraction: costs



$c(x,x')$ = cost of link (x,x')
e.g., $c(w,z) = 5$

cost could always be 1, or
inversely related to bandwidth,
or inversely related to
congestion

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

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

Routing algorithm classification

Q: global or decentralized information?

global:

- ❖ all routers have complete topology, link cost info
- ❖ “link state” algorithms

decentralized:

- ❖ router knows physically-connected neighbors, link costs to neighbors
- ❖ iterative process of computation, exchange of info with neighbors
- ❖ “distance vector” algorithms

Q: static or dynamic?

static:

- ❖ routes change slowly over time

dynamic:

- ❖ routes change more quickly
 - periodic update
 - in response to link cost changes

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

A Link-State Routing Algorithm

Dijkstra's algorithm

- ❖ net topology, link costs known to all nodes
 - accomplished via “link state broadcast”
 - all nodes have same info
- ❖ computes least cost paths from one node (‘source’) to all other nodes
 - gives *forwarding table* for that node
- ❖ iterative: after k iterations, know least cost path to k dest.’s

notation:

- ❖ $c(x,y)$: link cost from node x to y ; $= \infty$ if not direct neighbors
- ❖ $D(v)$: current value of cost of path from source to dest. v
- ❖ $p(v)$: predecessor node along path from source to v
- ❖ N' : set of nodes whose least cost path definitively known

Dijkstra's Algorithm

1 **Initialization:**

2 $N' = \{u\}$

3 for all nodes v

4 if v adjacent to u

5 then $D(v) = c(u,v)$

6 else $D(v) = \infty$

7

8 **Loop**

9 find w not in N' such that $D(w)$ is a minimum

10 add w to N'

11 update $D(v)$ for all v adjacent to w and not in N' :

12 $D(v) = \min(D(v), D(w) + c(w,v))$

13 /* new cost to v is either old cost to v or known

14 shortest path cost to w plus cost from w to v */

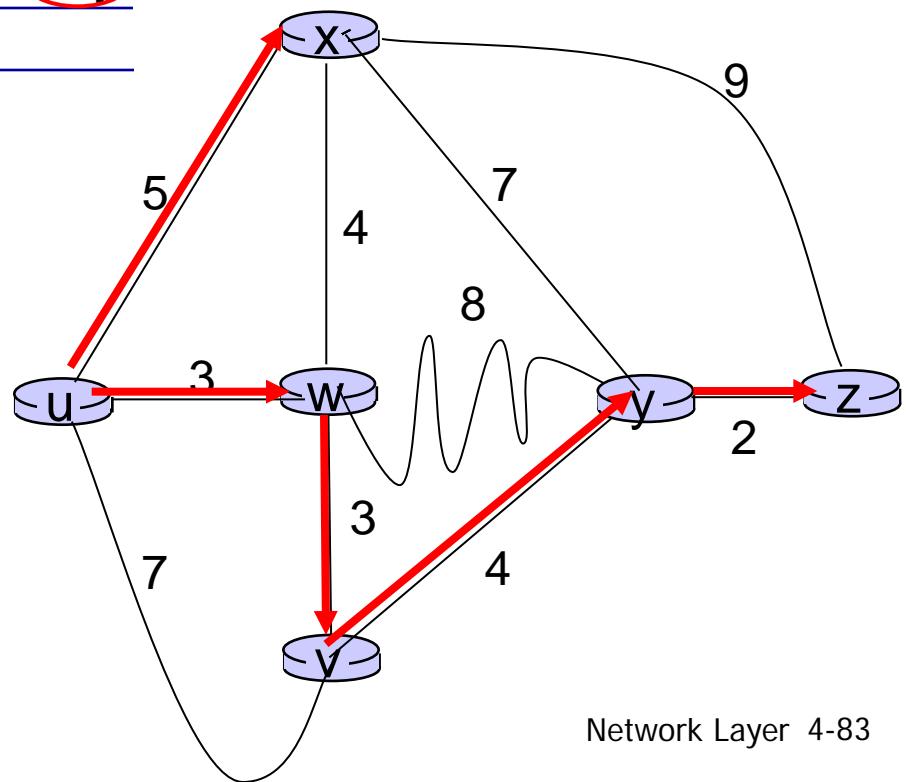
15 **until all nodes in N'**

Dijkstra's algorithm: example

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

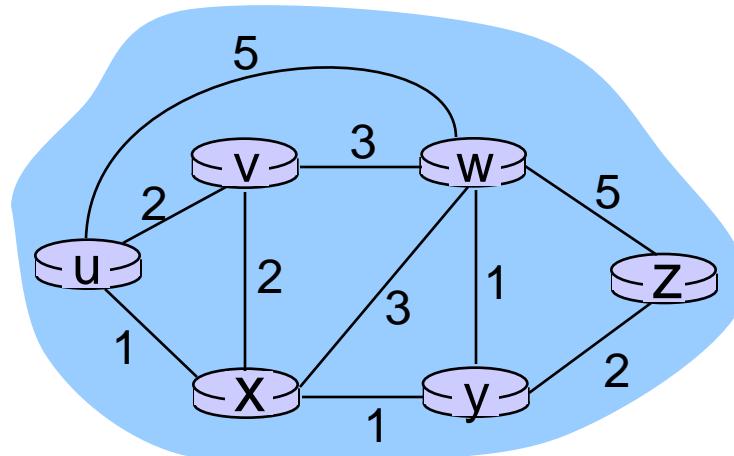
notes:

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



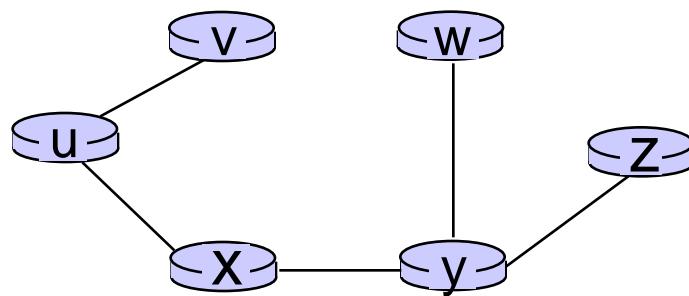
Dijkstra's algorithm: another example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y		4,y	
3	uxyv		3,y		4,y	
4	uxyvw					4,y
5	uxyvwz					



Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:



resulting forwarding table in u:

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

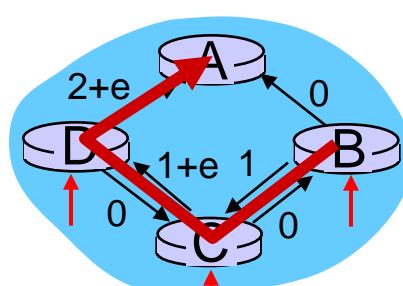
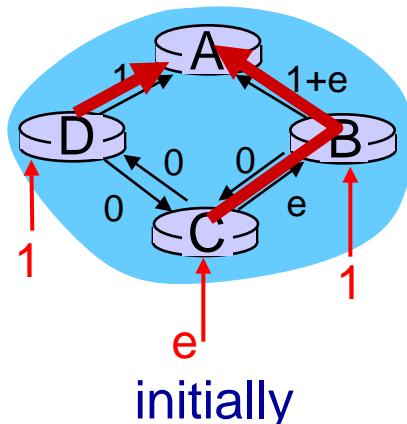
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

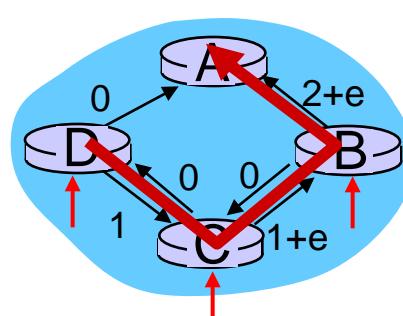
- ❖ each iteration: need to check all nodes, w, not in N
- ❖ $n(n+1)/2$ comparisons: $O(n^2)$
- ❖ more efficient implementations possible: $O(n \log n)$

oscillations possible:

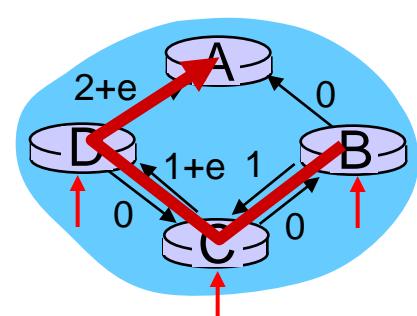
- ❖ e.g., support link cost equals amount of carried traffic:



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- **distance vector**
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$

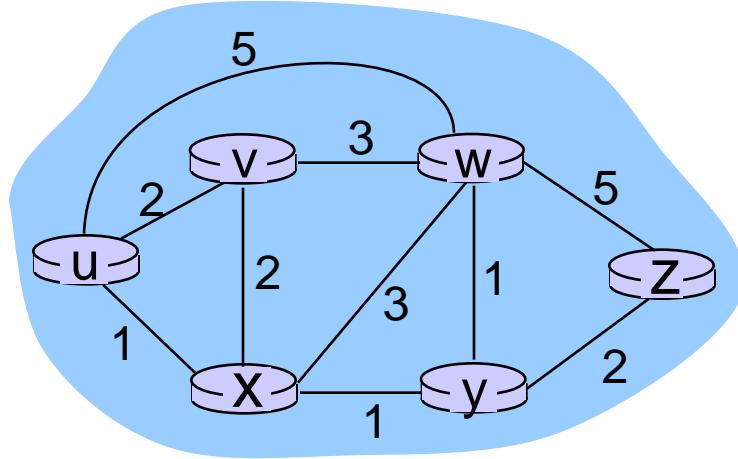
then

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

 v cost from neighbor v to destination y
cost to neighbor v

 \min taken over all neighbors v of x

Bellman-Ford example



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$\begin{aligned}d_u(z) &= \min \{ c(u,v) + d_v(z), \\&\quad c(u,x) + d_x(z), \\&\quad c(u,w) + d_w(z) \} \\&= \min \{ 2 + 5, \\&\quad 1 + 3, \\&\quad 5 + 3 \} = 4\end{aligned}$$

node achieving minimum is next
hop in shortest path, used in forwarding table

Distance vector algorithm

- ❖ $D_x(y)$ = estimate of least cost from x to y
 - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- ❖ node x :
 - knows cost to each neighbor v : $c(x,v)$
 - maintains its neighbors' distance vectors. For each neighbor v , x maintains
 $\mathbf{D}_v = [D_v(y): y \in N]$

Distance vector algorithm

key idea:

- ❖ from time-to-time, each node sends its own distance vector estimate to neighbors
- ❖ when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$

- ❖ under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

Distance vector algorithm

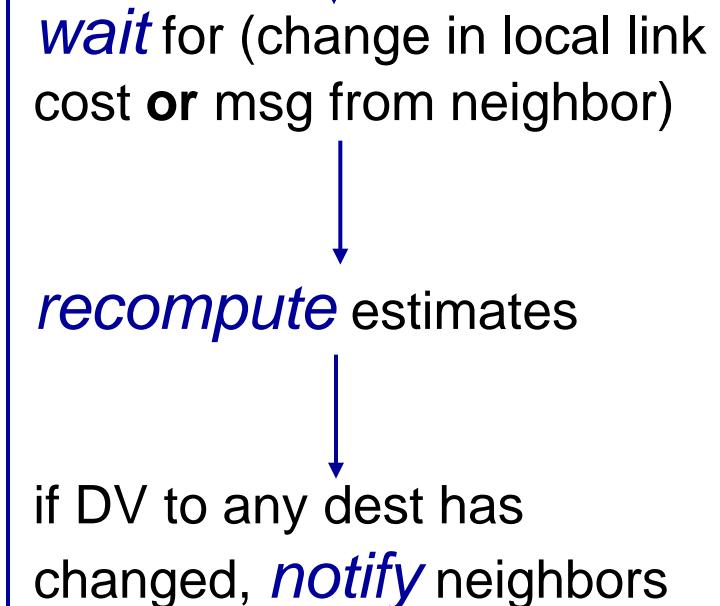
iterative, asynchronous:

- each local iteration caused by:
 - ❖ local link cost change
 - ❖ DV update message from neighbor

distributed:

- ❖ each node notifies neighbors *only* when its DV changes
 - neighbors then notify their neighbors if necessary

each node:



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

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

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

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

**node x
table**

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

**node y
table**

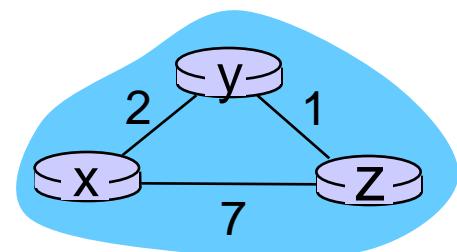
	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

**node z
table**

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

time



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

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

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

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

**node x
table**

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

**node y
table**

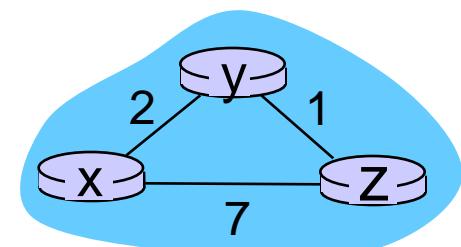
	cost to		
	x	y	z
from	x	∞ ∞ ∞	
y	2 0 1		
z	∞ ∞ ∞		

**node z
table**

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

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

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

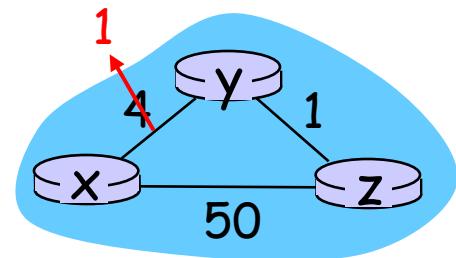


time

Distance vector: link cost changes

link cost changes:

- ❖ node detects local link cost change
- ❖ updates routing info, recalculates distance vector
- ❖ if DV changes, notify neighbors



**“good
news
travels
fast”**

t_0 : y detects link-cost change, updates its DV, informs its neighbors.

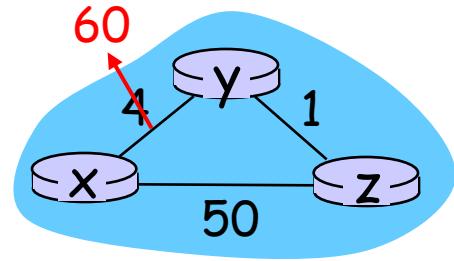
t_1 : z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV.

t_2 : y receives z' s update, updates its distance table. y' s least costs do *not* change, so y does *not* send a message to z.

Distance vector: link cost changes

link cost changes:

- ❖ node detects local link cost change
- ❖ *bad news travels slow* - “count to infinity” problem!
- ❖ 44 iterations before algorithm stabilizes: see text



poisoned reverse:

- ❖ If Z routes through Y to get to X :
 - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- ❖ will this completely solve count to infinity problem?

Comparison of LS and DV algorithms

message complexity

- ❖ **LS:** with n nodes, E links, $O(nE)$ msgs sent
- ❖ **DV:** exchange between neighbors only
 - convergence time varies

speed of convergence

- ❖ **LS:** $O(n^2)$ algorithm requires $O(nE)$ msgs
 - may have oscillations
- ❖ **DV:** convergence time varies
 - may be routing loops
 - count-to-infinity problem

robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its own table

DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
 - error propagate thru network

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast
routing

Hierarchical routing

our routing study thus far - idealization

- ❖ all routers identical
- ❖ network “flat”
- ... *not true in practice*

scale: with 600 million destinations:

- ❖ can't store all dest's in routing tables!
- ❖ routing table exchange would swamp links!

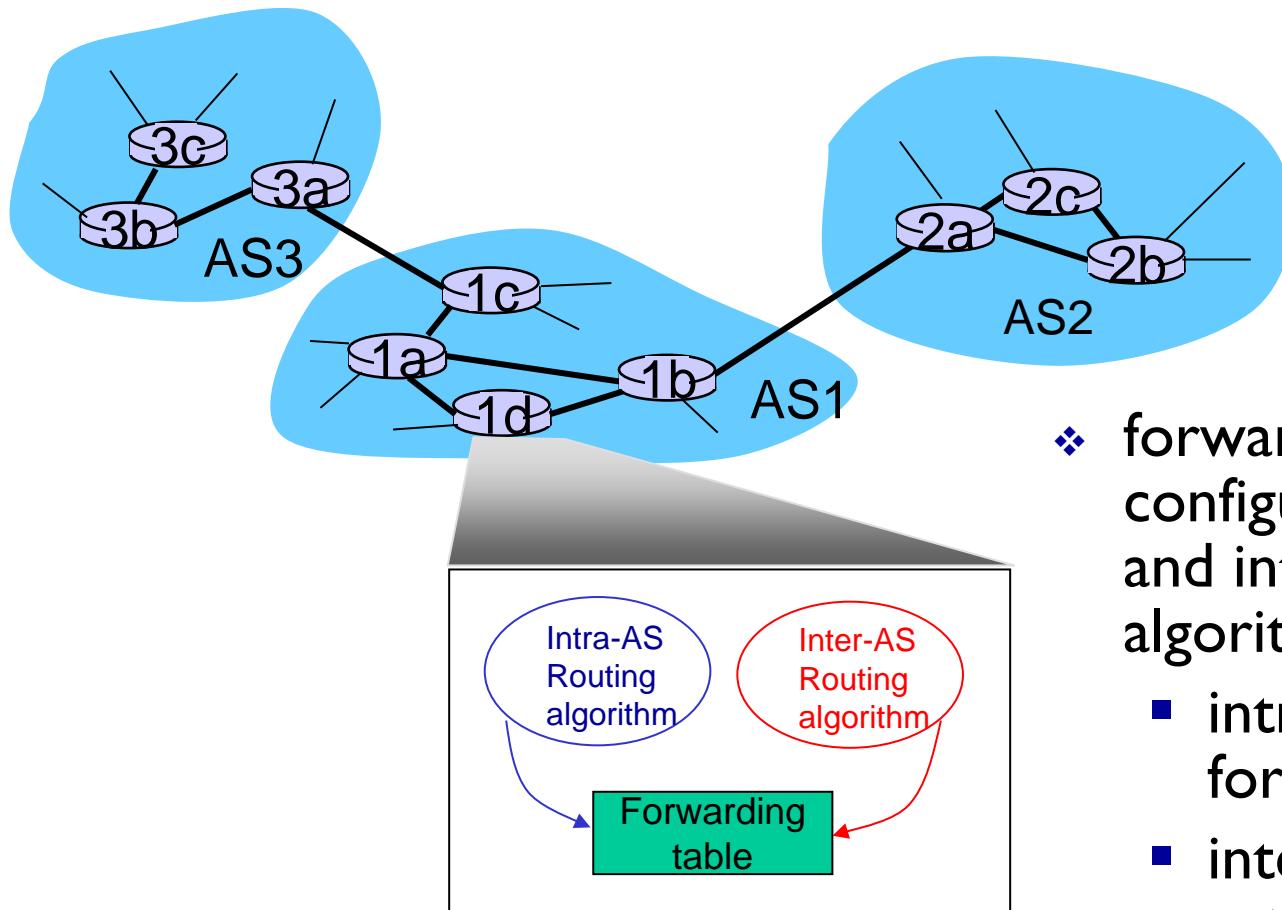
administrative autonomy

- ❖ internet = network of networks
- ❖ **each network admin may want to control routing in its own network**

Hierarchical routing

- ❖ aggregate routers into regions, “autonomous systems” (AS)
 - ❖ routers in same AS run same routing protocol
 - “intra-AS” routing protocol
 - routers in different AS can run different intra-AS routing protocol
- gateway router:*
- ❖ at “edge” of its own AS
 - ❖ has link to router in another AS

Interconnected ASes



- ❖ **forwarding table**
configured by both intra-
and inter-AS routing
algorithm
 - intra-AS sets entries
for internal dests
 - inter-AS & intra-AS
sets entries for
external dests

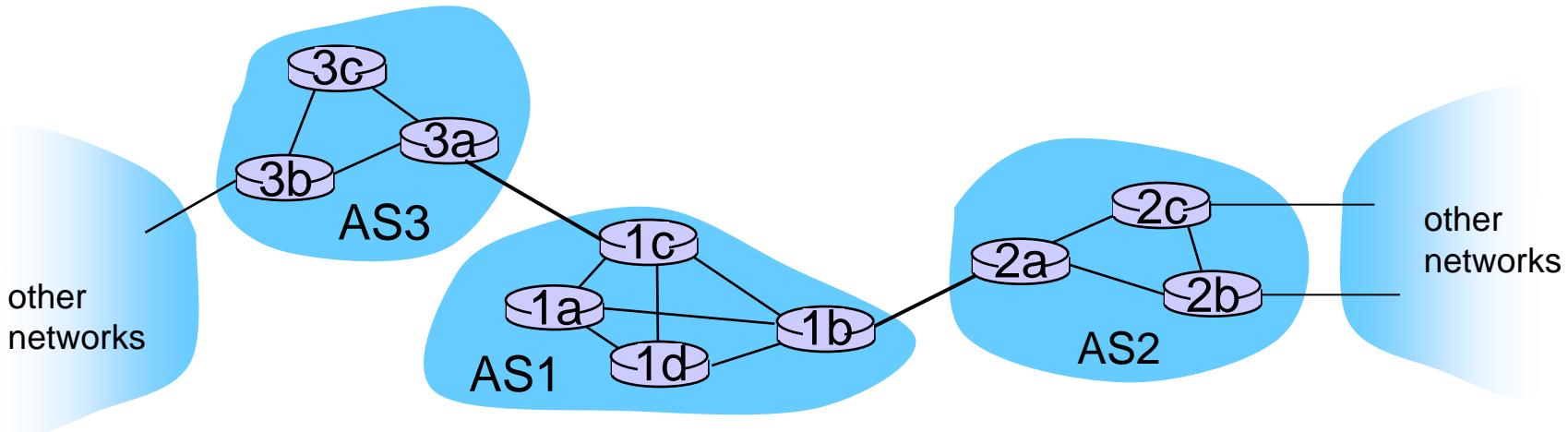
Inter-AS tasks

- ❖ suppose router in AS1 receives datagram destined outside of AS1:
 - router should forward packet to gateway router, but which one?

AS1 must:

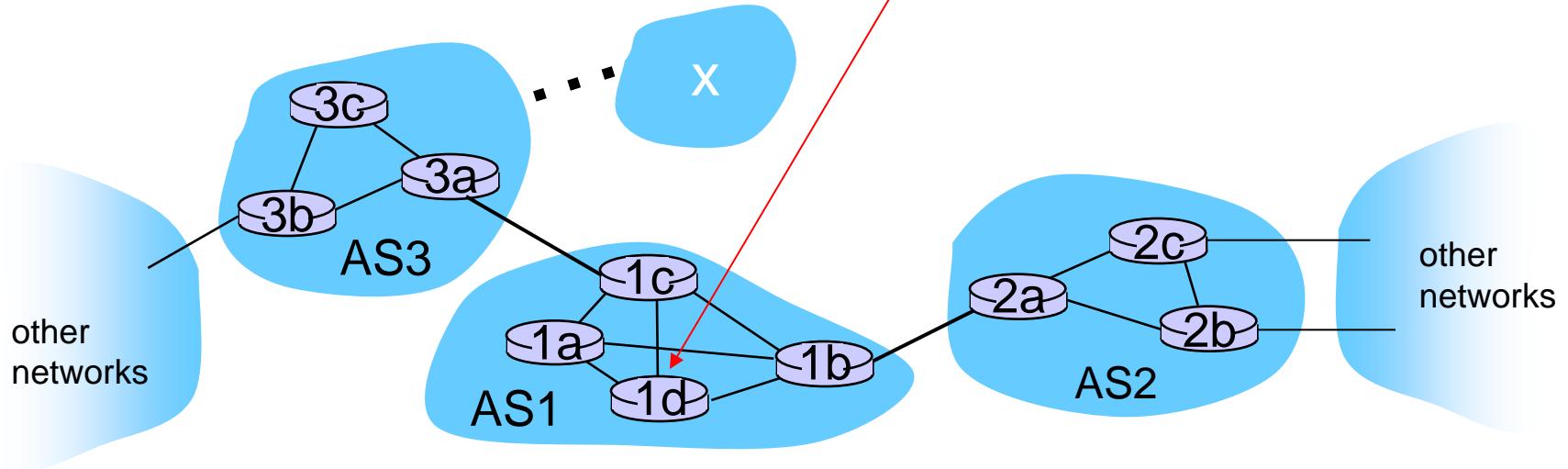
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



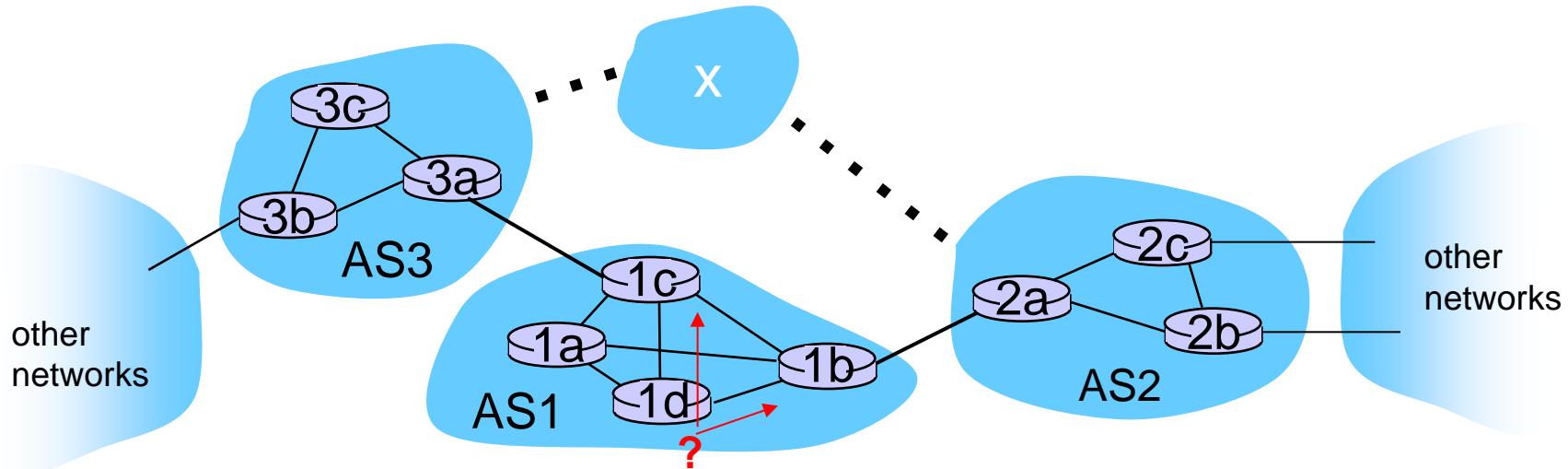
Example: setting forwarding table in router 1d

- ❖ suppose AS1 learns (via inter-AS protocol) that subnet **X** reachable via AS3 (gateway 1c), but not via AS2
 - inter-AS protocol propagates reachability info to all internal routers
- ❖ router 1d determines from intra-AS routing info that its interface **I** is on the least cost path to 1c
 - installs forwarding table entry **(x,I)**



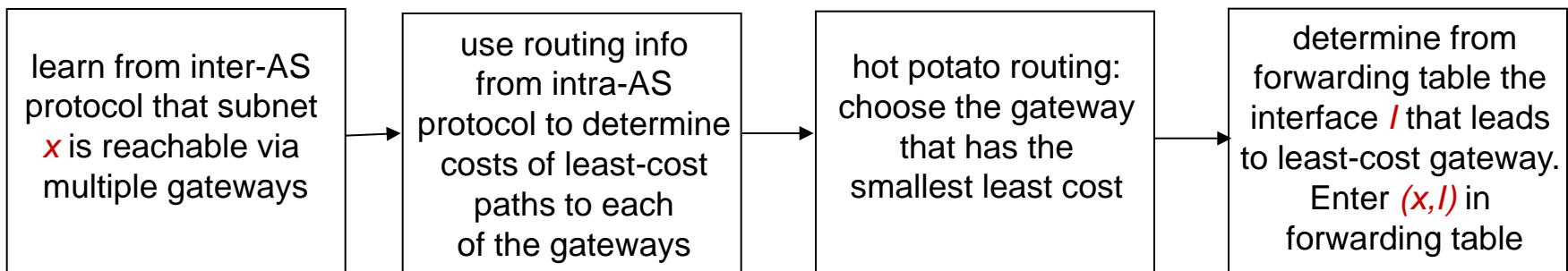
Example: choosing among multiple ASes

- ❖ now suppose AS1 learns from inter-AS protocol that subnet **x** is reachable from AS3 and from AS2.
- ❖ to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest **x**
 - this is also job of inter-AS routing protocol!



Example: choosing among multiple ASes

- ❖ now suppose AS1 learns from inter-AS protocol that subnet **x** is reachable from AS3 *and* from AS2.
- ❖ to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest **x**
 - this is also job of inter-AS routing protocol!
- ❖ *hot potato routing*: send packet towards closest of two routers.



Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

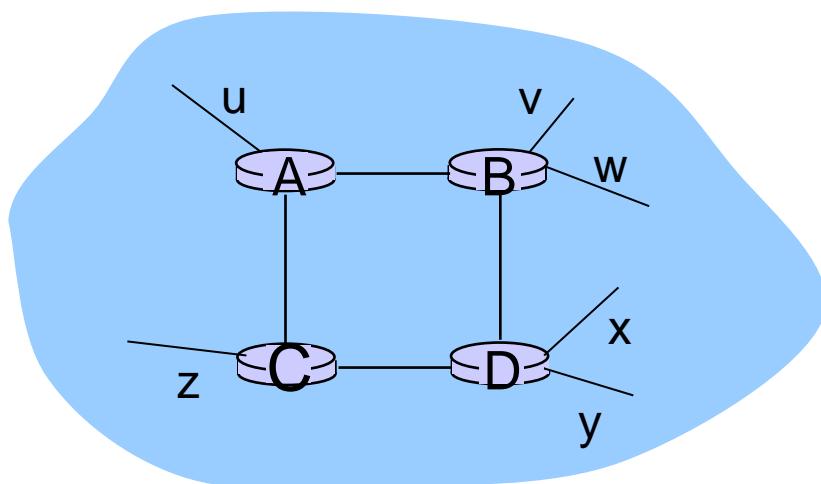
4.7 broadcast and multicast
routing

Intra-AS Routing

- ❖ also known as *interior gateway protocols (IGP)*
- ❖ most common intra-AS routing protocols:
 - RIP: Routing Information Protocol
 - OSPF: Open Shortest Path First
 - IGRP: Interior Gateway Routing Protocol
(Cisco proprietary)

RIP (Routing Information Protocol)

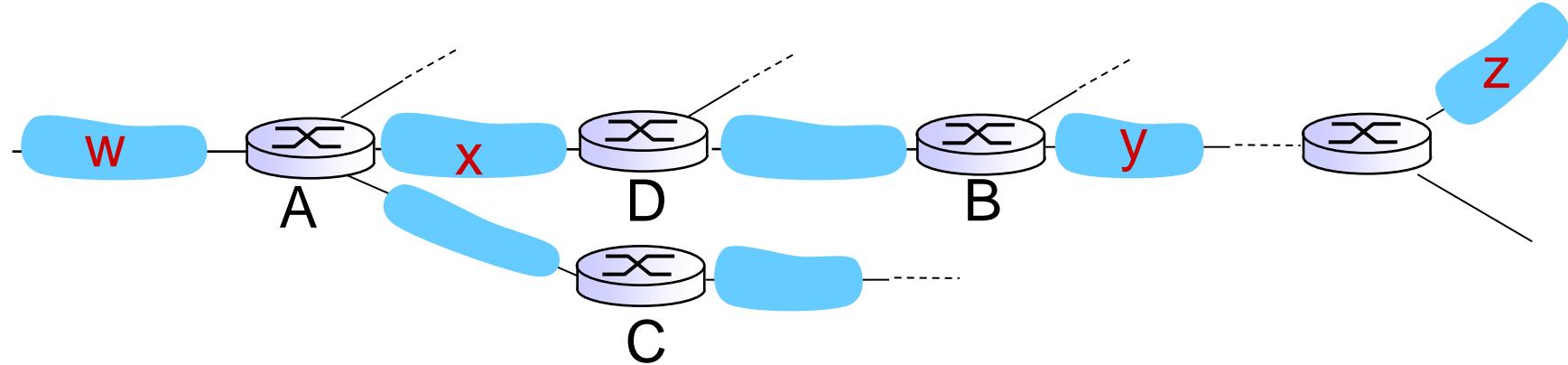
- ❖ included in BSD-UNIX distribution in 1982
- ❖ distance vector algorithm
 - distance metric: # hops (max = 15 hops), each link has cost 1
 - DVs exchanged with neighbors every 30 sec in response message (aka **advertisement**)
 - each advertisement: list of up to 25 destination **subnets** (*in IP addressing sense*)



from router A to destination **subnets**:

<u>subnet</u>	<u>hops</u>
u	1
v	2
w	2
x	3
y	3
z	2

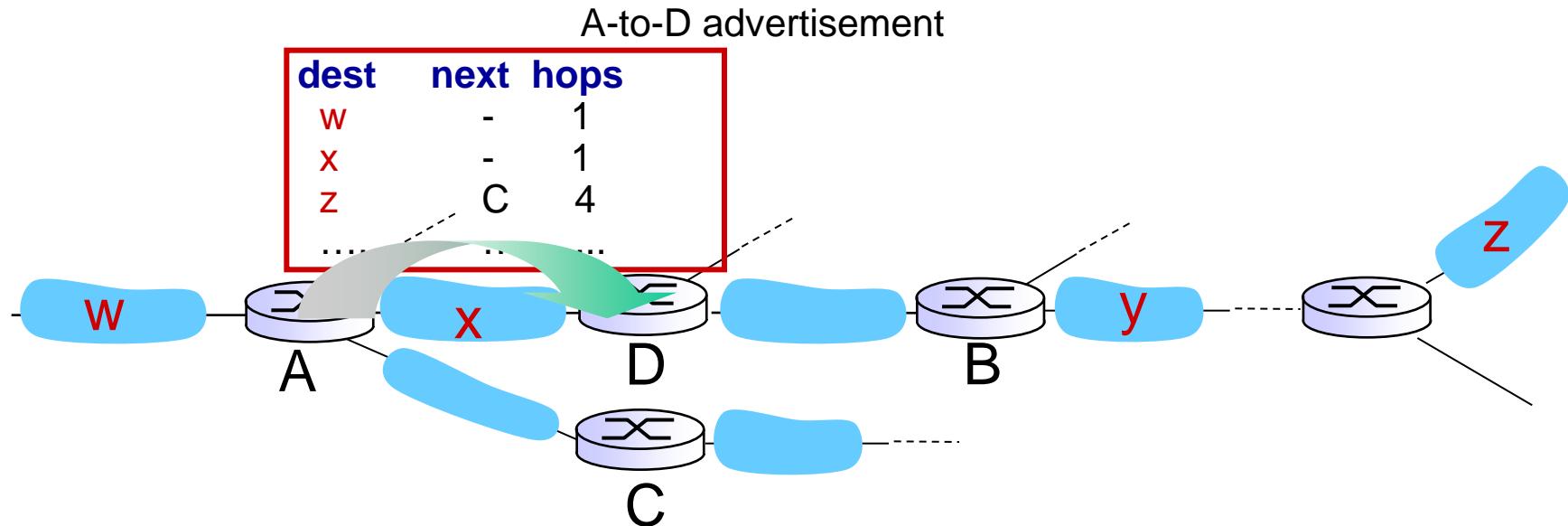
RIP: example



routing table in router D

destination subnet	next router	# hops to dest
W	A	2
y	B	2
z	B	7
X	--	1
....

RIP: example



destination subnet	next router	# hops to dest
W	A	2
y	B	2
Z	B → A	5
X	--	1
....

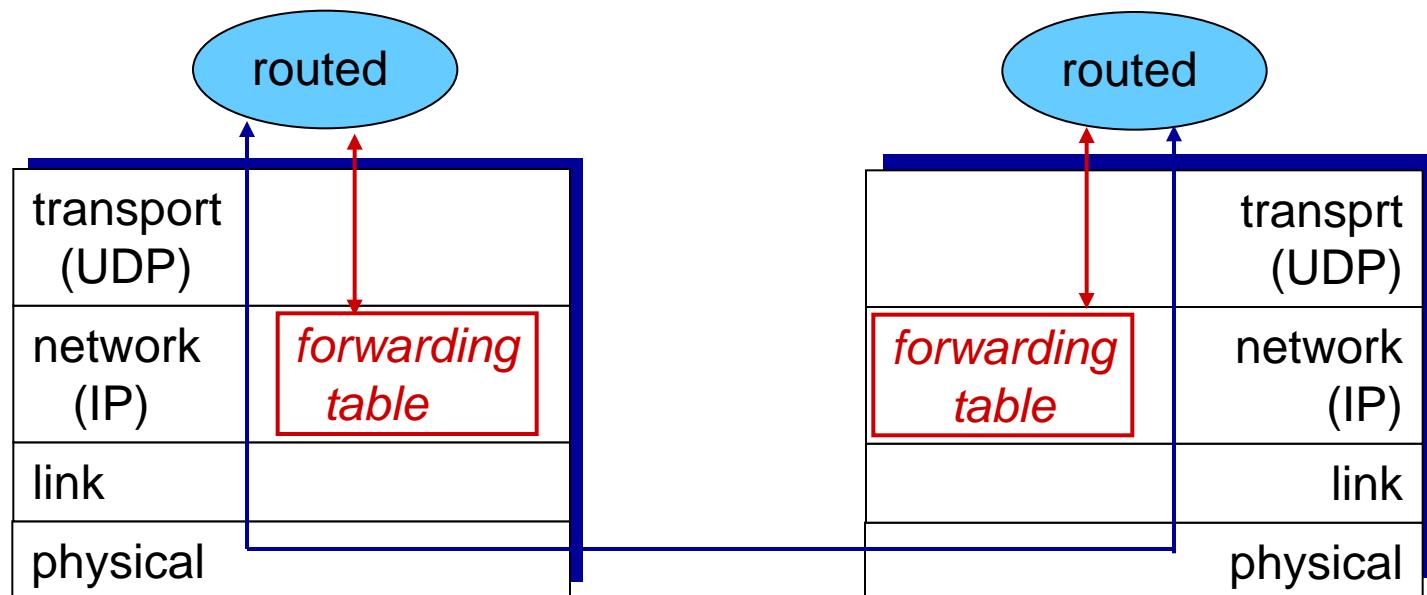
RIP: link failure, recovery

if no advertisement heard after 180 sec -->
neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly (?) propagates to entire net
- *poison reverse* used to prevent ping-pong loops (infinite distance = 16 hops)

RIP table processing

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



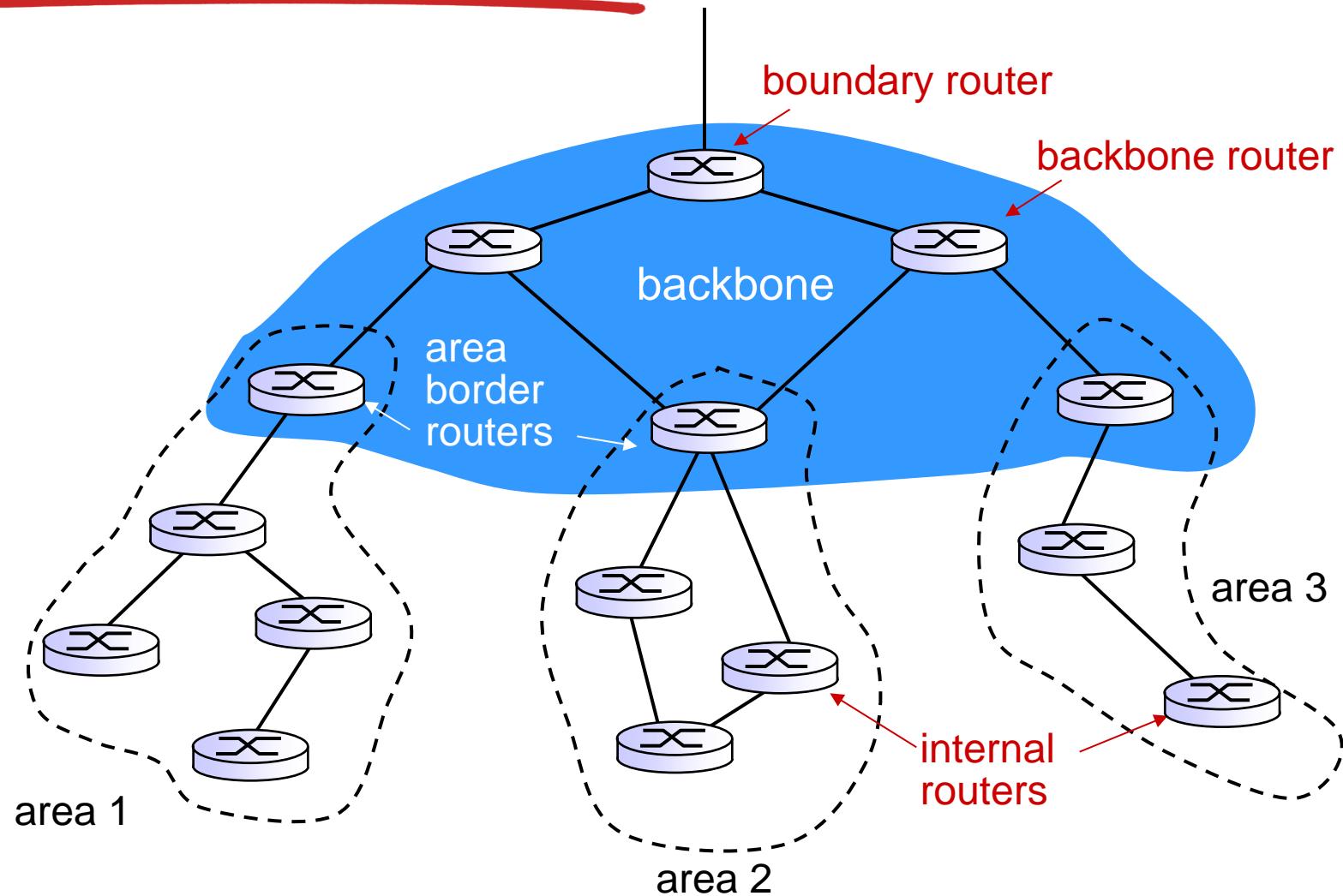
OSPF (Open Shortest Path First)

- ❖ “open”: publicly available
- ❖ uses link state algorithm
 - LS packet dissemination
 - topology map at each node
 - route computation using Dijkstra’s algorithm
- ❖ OSPF advertisement carries one entry per neighbor
- ❖ advertisements flooded to *entire* AS
 - carried in OSPF messages directly over IP (rather than TCP or UDP)
- ❖ *IS-IS routing* protocol: nearly identical to OSPF

OSPF “advanced” features (not in RIP)

- ❖ **security**: all OSPF messages authenticated (to prevent malicious intrusion)
- ❖ **multiple same-cost paths** allowed (only one path in RIP)
- ❖ for each link, multiple cost metrics for different **TOS** (e.g., satellite link cost set “low” for best effort ToS; high for real time ToS)
- ❖ integrated uni- and **multicast** support:
 - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- ❖ **hierarchical** OSPF in large domains.

Hierarchical OSPF



Hierarchical OSPF

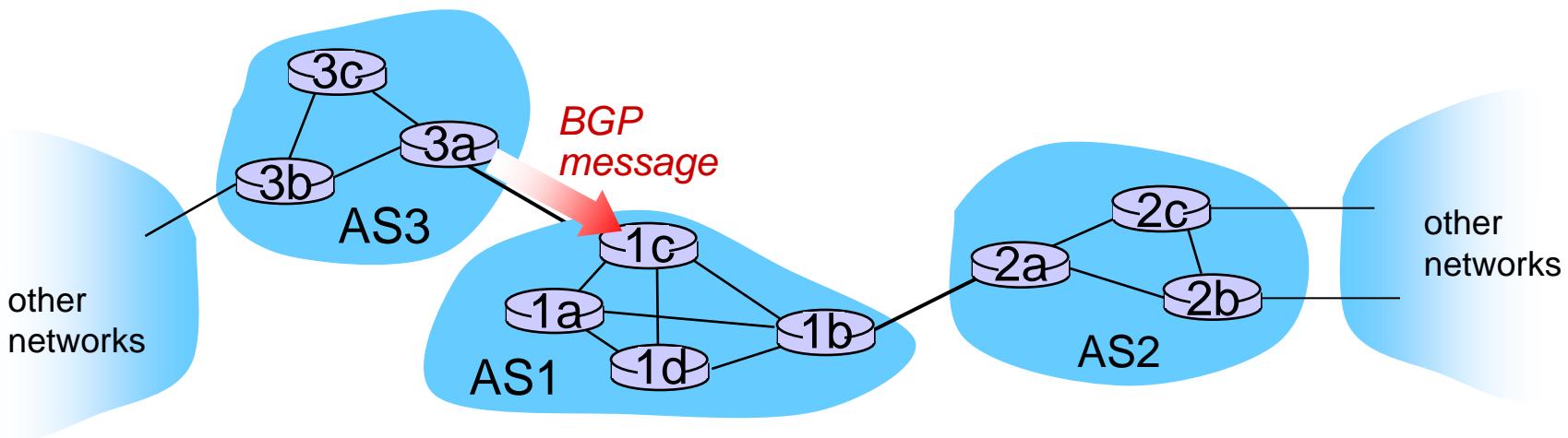
- ❖ *two-level hierarchy*: local area, backbone.
 - link-state advertisements only in area
 - each node has detailed area topology; only know direction (shortest path) to nets in other areas.
- ❖ *area border routers*: “summarize” distances to nets in own area, advertise to other Area Border routers.
- ❖ *backbone routers*: run OSPF routing limited to backbone.
- ❖ *boundary routers*: connect to other AS’ s.

Internet inter-AS routing: BGP

- ❖ **BGP (Border Gateway Protocol):** *the de facto* inter-domain routing protocol
 - “glue that holds the Internet together”
- ❖ BGP provides each AS a means to:
 - **eBGP:** obtain **subnet reachability information** from neighboring ASs.
 - **iBGP:** propagate reachability information to **all AS-internal routers**.
 - determine “good” routes to other networks based on reachability information and policy.
- ❖ allows subnet to **advertise its existence** to rest of Internet: “*I am here*”

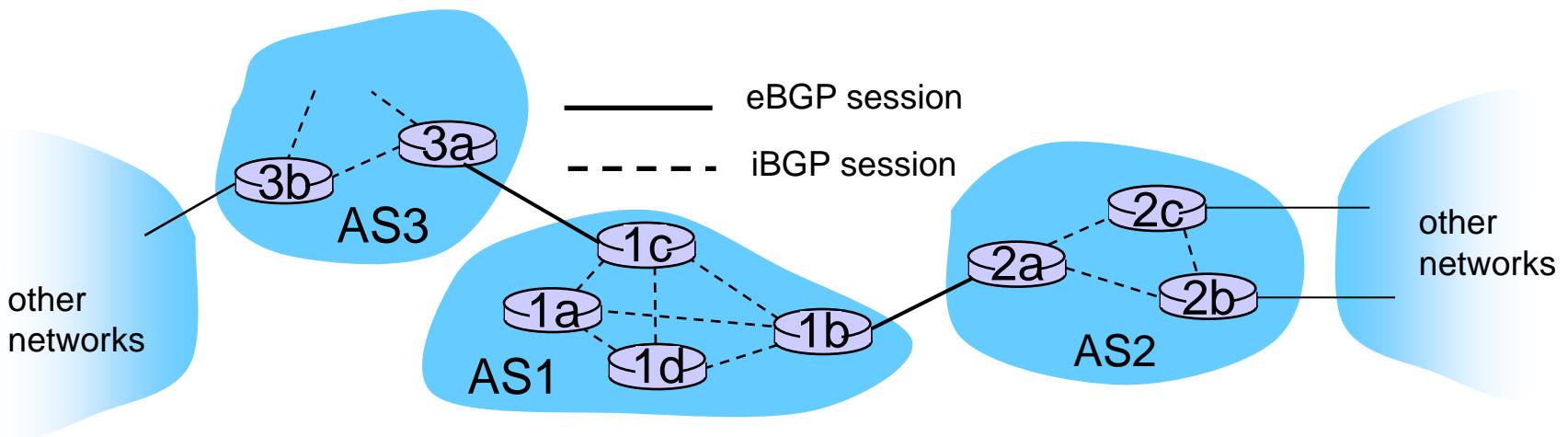
BGP basics

- ❖ **BGP session:** two BGP routers (“peers”) exchange BGP messages:
 - advertising *paths* to different destination network prefixes (“path vector” protocol)
 - exchanged over semi-permanent TCP connections
 - ❖ when AS3 advertises a prefix to AS1:
 - AS3 *promises* it will forward datagrams towards that prefix
 - AS3 can aggregate prefixes in its advertisement



BGP basics: distributing path information

- ❖ using eBGP session between 3a and 1c, AS3 sends prefix reachability info to AS1.
 - 1c can then use iBGP do distribute new prefix info to all routers in AS1
 - 1b can then re-advertise new reachability info to AS2 over 1b-to-2a eBGP session
- ❖ when router learns of new prefix, it creates entry for prefix in its forwarding table.



Path attributes and BGP routes

- ❖ advertised prefix includes BGP attributes
 - prefix + attributes = “route”
- ❖ two important attributes:
 - **AS-PATH**: contains ASs through which prefix advertisement has passed: e.g., AS 67, AS 17
 - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS. (may be multiple links from current AS to next-hop-AS)
- ❖ gateway router receiving route advertisement uses **import policy** to accept/decline
 - e.g., never route through AS x
 - **policy-based** routing

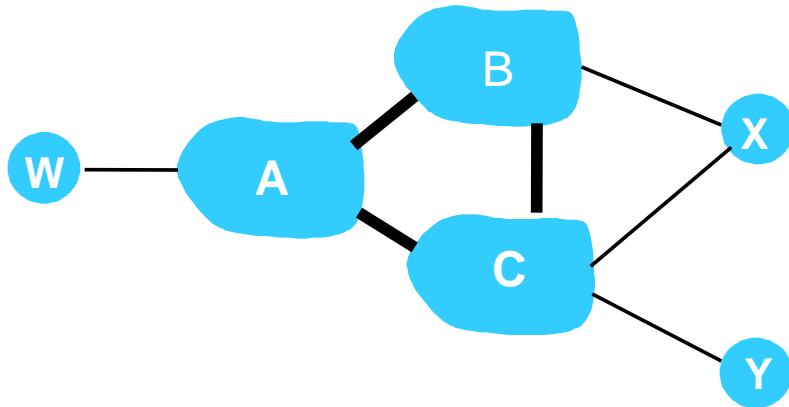
BGP route selection

- ❖ router may learn about more than 1 route to destination AS, selects route based on:
 1. local preference value attribute: policy decision
 2. shortest AS-PATH
 3. closest NEXT-HOP router: hot potato routing
 4. additional criteria

BGP messages

- ❖ BGP messages exchanged between peers over TCP connection
- ❖ BGP messages:
 - **OPEN**: opens TCP connection to peer and authenticates sender
 - **UPDATE**: advertises new path (or withdraws old)
 - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
 - **NOTIFICATION**: reports errors in previous msg; also used to close connection

BGP routing policy

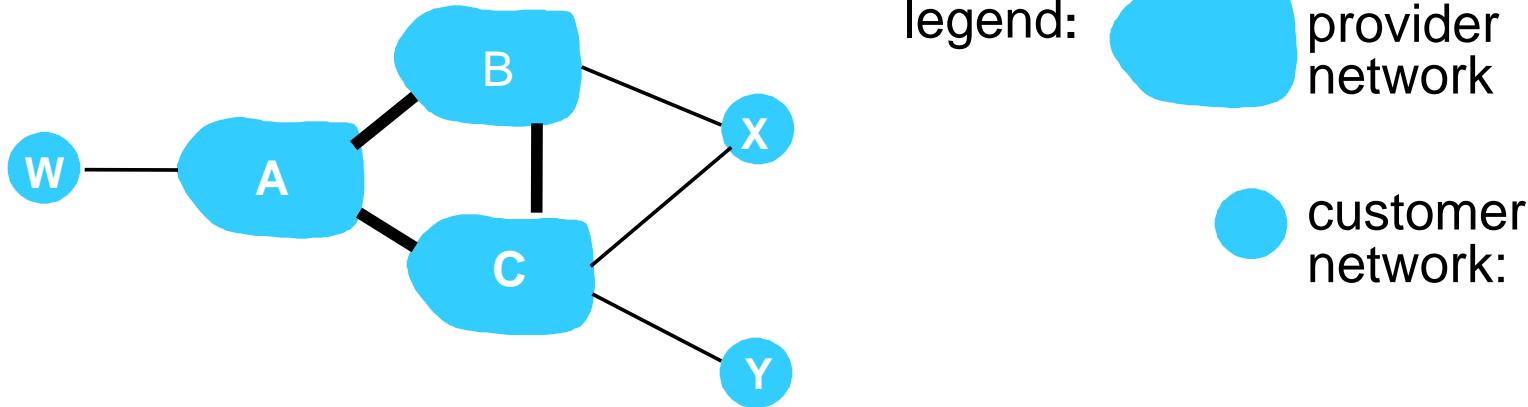


legend:

- provider network
- customer network:

- ❖ A,B,C are *provider networks*
- ❖ X,W,Y are customer (of provider networks)
- ❖ X is *dual-homed*: attached to two networks
 - X does not want to route from B via X to C
 - .. so X will not advertise to B a route to C

BGP routing policy (2)



- ❖ A advertises path AW to B
- ❖ B advertises path BAW to X
- ❖ Should B advertise path BAW to C?
 - No way! B gets no “revenue” for routing CBAW since neither W nor C are B’s customers
 - B wants to force C to route to w via A
 - B wants to route *only* to/from its customers! Unless it has to.

Why different Intra-, Inter-AS routing ?

policy:

- ❖ inter-AS: admin wants control over how its traffic routed, who routes through its net.
- ❖ intra-AS: single admin, so no policy decisions needed

scale:

- ❖ hierarchical routing saves table size, reduced update traffic

performance:

- ❖ intra-AS: can focus on performance
- ❖ inter-AS: policy may dominate over performance

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

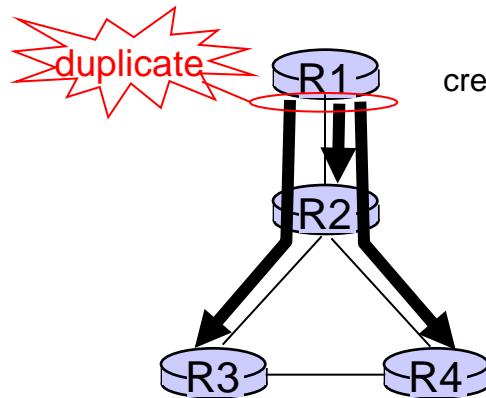
4.6 routing in the Internet

- RIP
- OSPF
- BGP

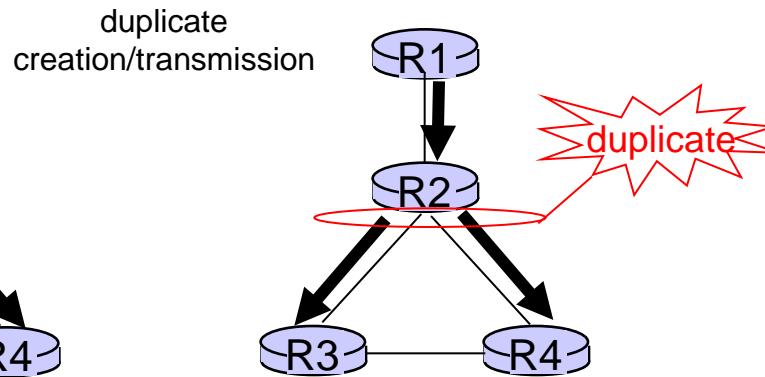
4.7 broadcast and multicast
routing

Broadcast routing

- ❖ deliver packets from source to all other nodes
- ❖ source duplication is inefficient:



source
duplication



in-network
duplication

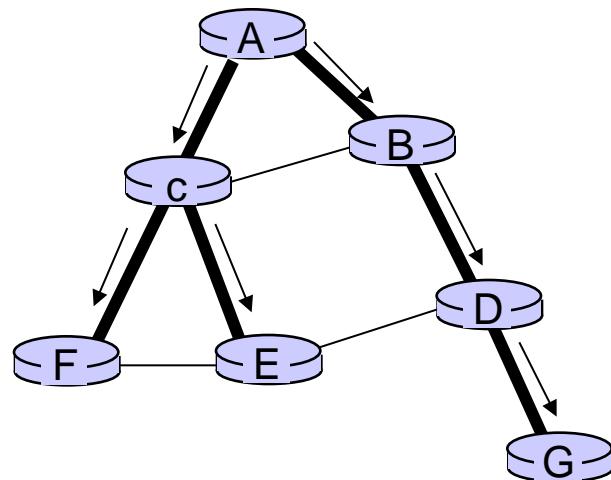
- ❖ source duplication: how does source determine recipient addresses?

In-network duplication

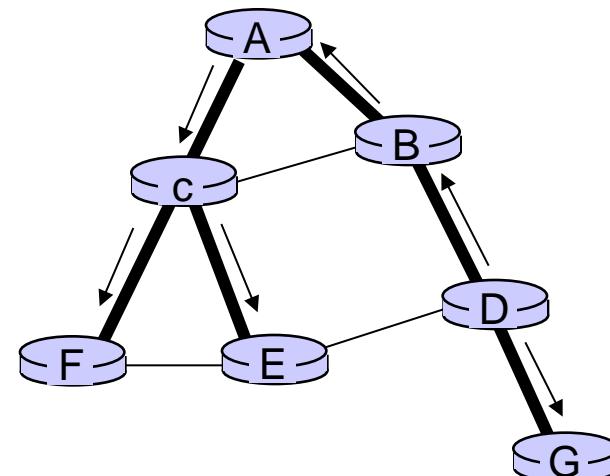
- ❖ **flooding:** when node receives broadcast packet, sends copy to all neighbors
 - problems: cycles & broadcast storm
- ❖ **controlled flooding:** node only broadcasts pkt if it hasn't broadcast same packet before
 - node keeps track of packet ids already broadacsted
 - or reverse path forwarding (RPF): only forward packet if it arrived on shortest path between node and source
- ❖ **spanning tree:**
 - no redundant packets received by any node

Spanning tree

- ❖ first construct a spanning tree
- ❖ nodes then forward/make copies only along spanning tree



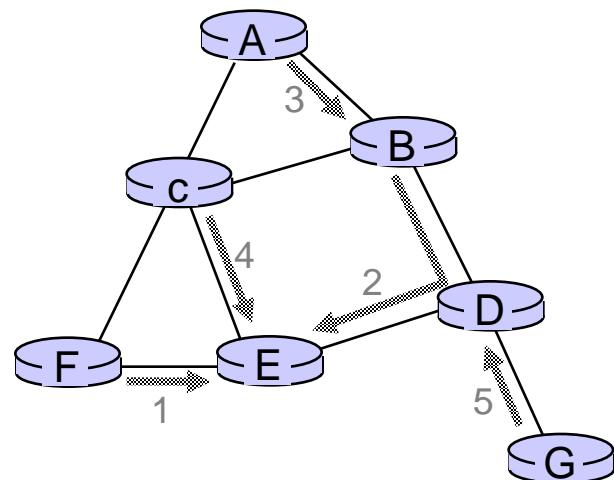
(a) broadcast initiated at A



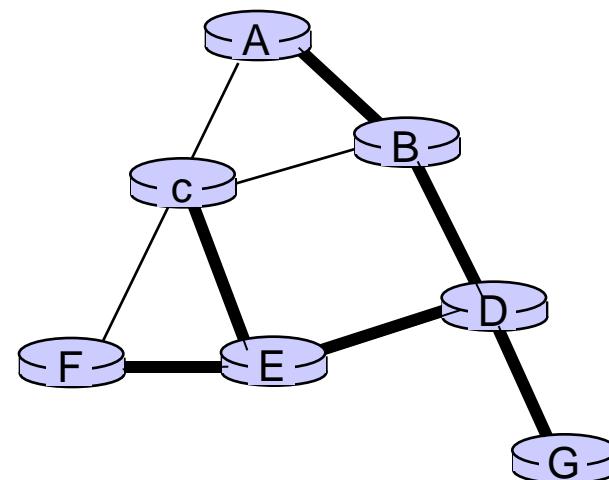
(b) broadcast initiated at D

Spanning tree: creation

- ❖ center node
- ❖ each node sends unicast join message to center node
 - message forwarded until it arrives at a node already belonging to spanning tree



(a) stepwise construction of spanning tree (center: E)

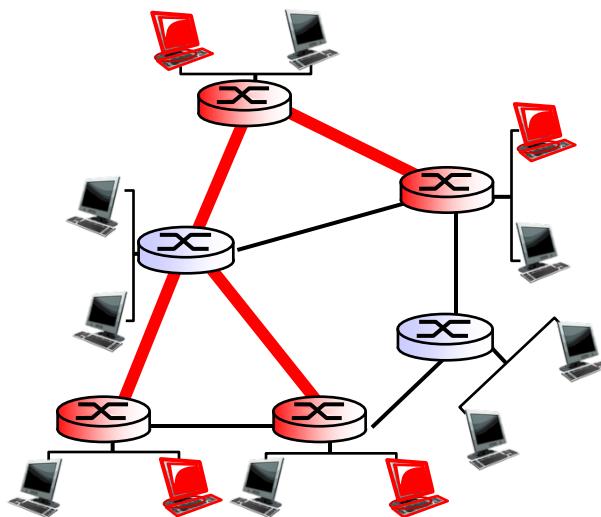


(b) constructed spanning tree

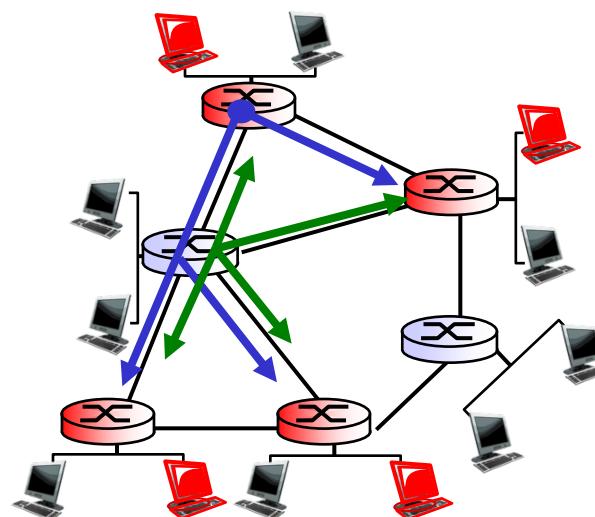
Multicast routing: problem statement

goal: find a tree (or trees) connecting routers having local mcast group members

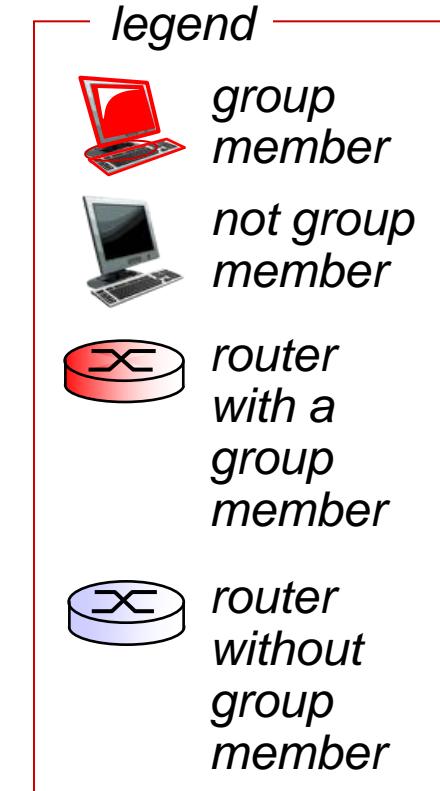
- ❖ **tree:** not all paths between routers used
- ❖ **shared-tree:** same tree used by all group members
- ❖ **source-based:** different tree from each sender to rcvs



shared tree



source-based trees



Approaches for building mcast trees

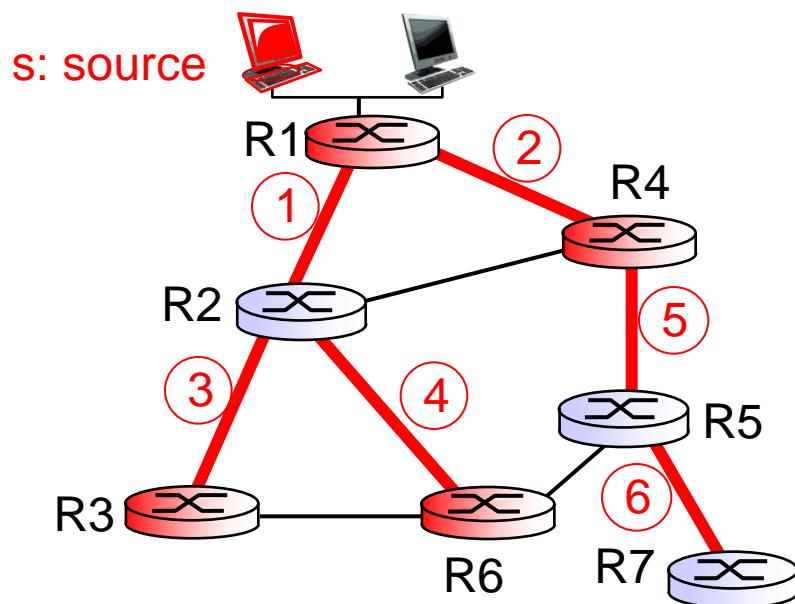
approaches:

- ❖ *source-based tree*: one tree per source
 - shortest path trees
 - reverse path forwarding
- ❖ *group-shared tree*: group uses one tree
 - minimal spanning (Steiner)
 - center-based trees

...we first look at basic approaches, then specific protocols adopting these approaches

Shortest path tree

- ❖ mcast forwarding tree: tree of shortest path routes from source to all receivers
 - Dijkstra's algorithm



LEGEND

- router with attached group member
- router with no attached group member
- link used for forwarding, i indicates order link added by algorithm

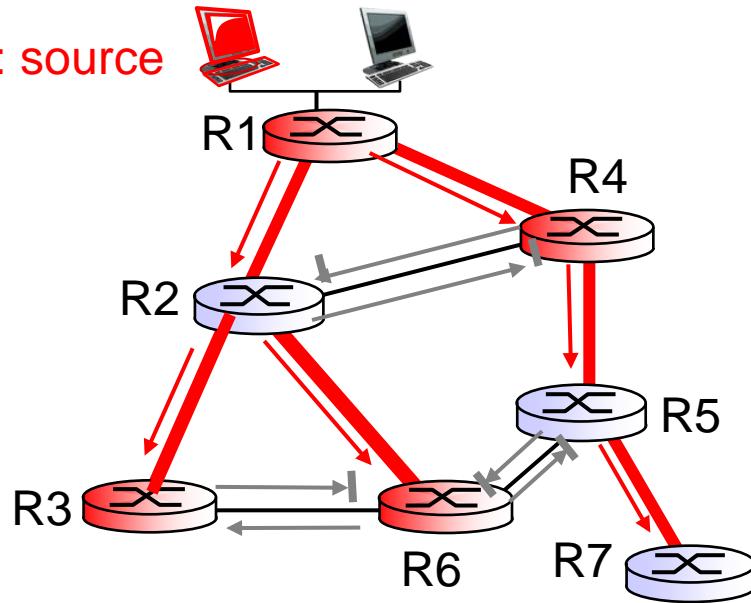
Reverse path forwarding

- ❖ rely on router's knowledge of unicast shortest path from it to sender
- ❖ each router has simple forwarding behavior:

if (mcast datagram received on incoming link on
shortest path back to center)
then flood datagram onto all outgoing links
else ignore datagram

Reverse path forwarding: example

s: source



LEGEND



router with attached
group member



router with no attached
group member



datagram will be forwarded

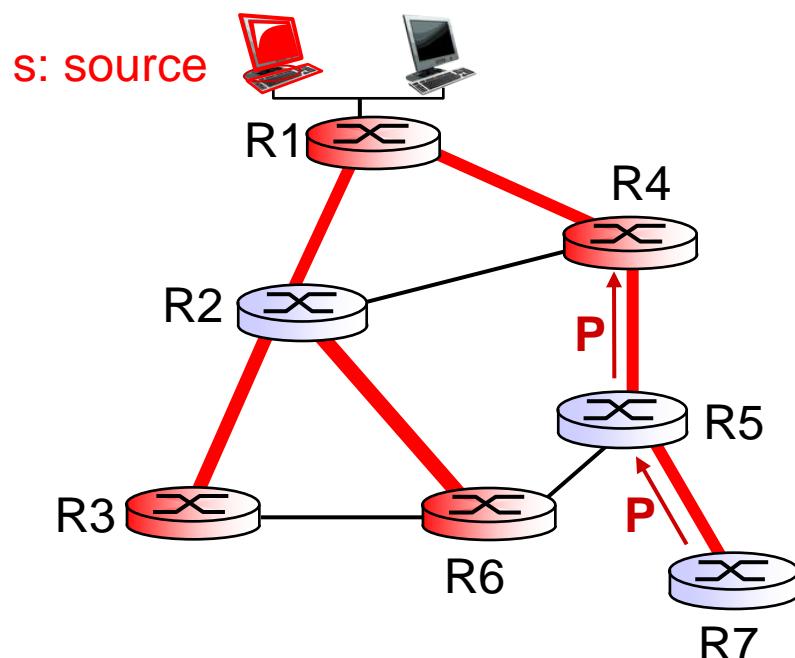


datagram will not be
forwarded

- ❖ result is a source-specific reverse SPT
 - may be a bad choice with asymmetric links

Reverse path forwarding: pruning

- ❖ forwarding tree contains subtrees with no mcast group members
 - no need to forward datagrams down subtree
 - “prune” msgs sent upstream by router with no downstream group members



LEGEND

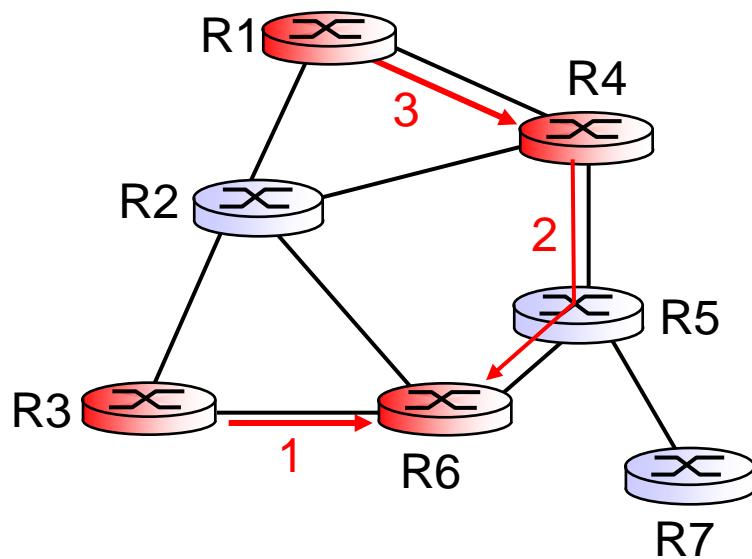
- router with attached group member
- router with no attached group member
- prune message
- links with multicast forwarding

Shared tree: Center-based trees

- ❖ single delivery tree shared by all
- ❖ one router identified as “*center*” of tree
- ❖ to join:
 - edge router sends unicast *join-msg* addressed to center router
 - *join-msg* “processed” by intermediate routers and forwarded towards center
 - *join-msg* either hits existing tree branch for this center, or arrives at center
 - path taken by *join-msg* becomes new branch of tree for this router

Center-based trees: example

suppose R6 chosen as center:

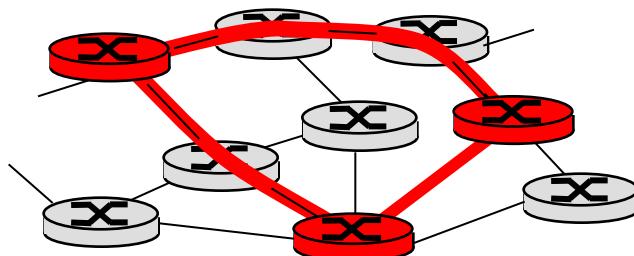


LEGEND

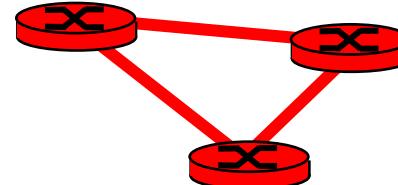
- router with attached group member
- router with no attached group member
- 1 path order in which join messages generated

Tunneling

Q: how to connect “islands” of multicast routers in a “sea” of unicast routers?



physical topology



logical topology

- ❖ mcast datagram encapsulated inside “normal” (non-multicast-addressed) datagram
- ❖ normal IP datagram sent thru “tunnel” via regular IP unicast to receiving mcast router (recall IPv6 inside IPv4 tunneling)
- ❖ receiving mcast router unencapsulates to get mcast datagram

Chapter 4: done!

4.1 introduction

4.2 virtual circuit and
datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format, IPv4 addressing, ICMP, IPv6

- ❖ understand principles behind network layer services:
 - network layer service models, forwarding versus routing how a router works, routing (path selection), broadcast, multicast
- ❖ instantiation, implementation in the Internet

4.5 routing algorithms

- link state, distance vector, hierarchical routing

4.6 routing in the Internet

- RIP, OSPF, BGP

4.7 broadcast and multicast
routing

Chapter 5

Link Layer

© *The materials in these notes are adapted from Computer Networking: A Top Down Approach, 6th edition, by Jim Kurose, Keith Ross*

Chapter 5: Link layer

our goals:

- ❖ understand principles behind link layer services:
 - error detection, correction
 - sharing a broadcast channel: multiple access
 - link layer addressing
 - local area networks: Ethernet, VLANs
- ❖ instantiation, implementation of various link layer technologies

Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

5.5 link virtualization:
MPLS

5.6 data center
networking

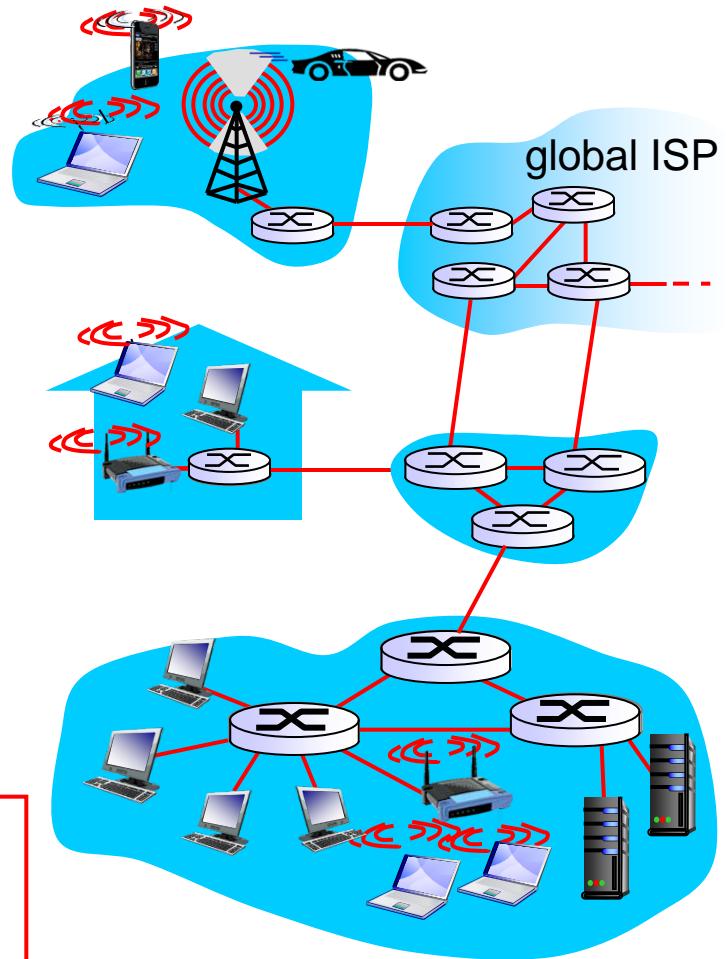
5.7 a day in the life of a
web request

Link layer: introduction

terminology:

- ❖ hosts and routers: **nodes**
- ❖ communication channels that connect adjacent nodes along communication path: **links**
 - wired links
 - wireless links
 - LANs
- ❖ layer-2 packet: **frame**, encapsulates datagram

data-link layer has responsibility of transferring datagram from one node to ***physically adjacent*** node over a link



Link layer: context

- ❖ datagram transferred by different link protocols over different links:
 - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- ❖ each link protocol provides different services
 - e.g., may or may not provide rdt over link

transportation analogy:

- ❖ trip from Princeton to Lausanne
 - limo: Princeton to JFK
 - plane: JFK to Geneva
 - train: Geneva to Lausanne
- ❖ tourist = **datagram**
- ❖ transport segment = **communication link**
- ❖ transportation mode = **link layer protocol**
- ❖ travel agent = **routing algorithm**

Link layer services

- ❖ *framing, link access:*

- encapsulate datagram into frame, adding header, trailer
- channel access if shared medium
- “MAC” addresses used in frame headers to identify source, dest
 - different from IP address!

- ❖ *reliable delivery between adjacent nodes*

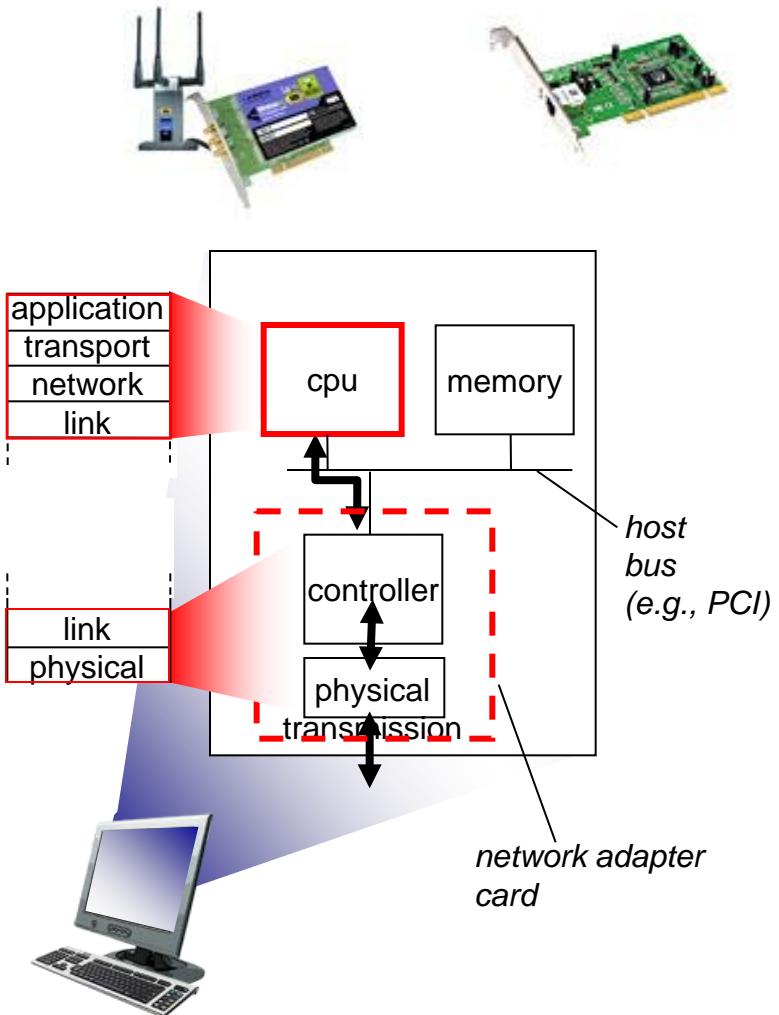
- we learned how to do this already (chapter 3)!
- seldom used on low bit-error link (fiber, some twisted pair)
- used in wireless links: high error rates
 - *Q:* why both link-level and end-end reliability?

Link layer services (more)

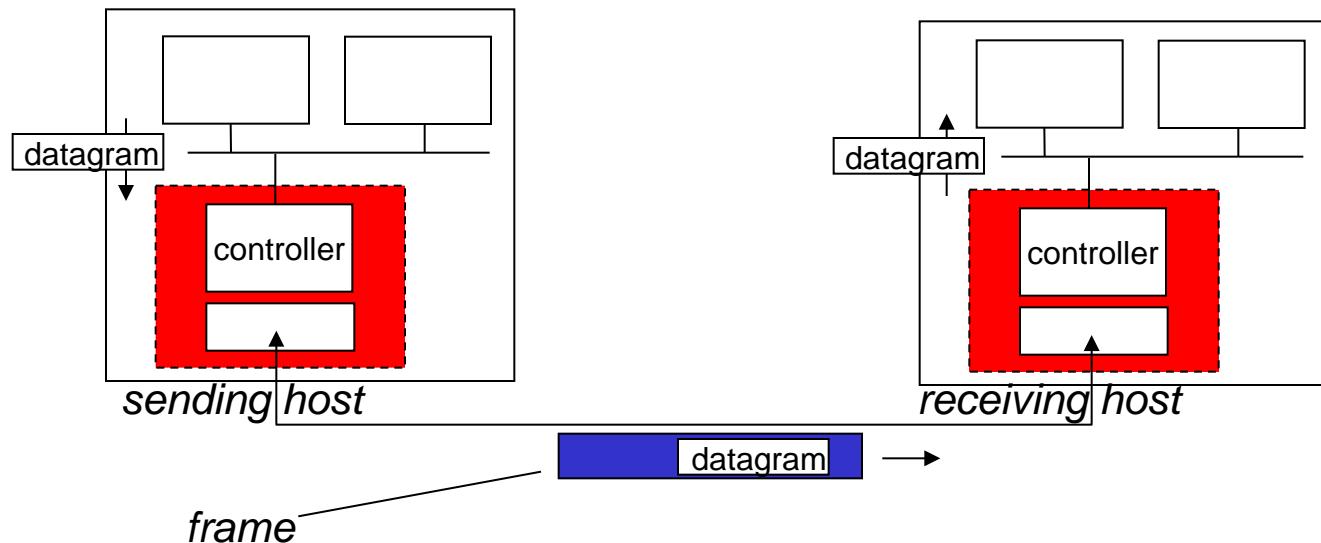
- ❖ *flow control:*
 - pacing between adjacent sending and receiving nodes
- ❖ *error detection:*
 - errors caused by signal attenuation, noise.
 - receiver detects presence of errors:
 - signals sender for retransmission or drops frame
- ❖ *error correction:*
 - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- ❖ *half-duplex and full-duplex*
 - with half duplex, nodes at both ends of link can transmit, but not at same time

Where is the link layer implemented?

- ❖ in each and every host
- ❖ link layer implemented in “adaptor” (aka *network interface card* NIC) or on a chip
 - Ethernet card, 802.11 card; Ethernet chipset
 - implements link, physical layer
- ❖ attaches into host’s system buses
- ❖ combination of hardware, software, firmware



Adaptors communicating



- ❖ sending side:
 - encapsulates datagram in frame
 - adds error checking bits, rdt, flow control, etc.
- ❖ receiving side
 - looks for errors, rdt, flow control, etc
 - extracts datagram, passes to upper layer at receiving side

Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

5.5 link virtualization:
MPLS

5.6 data center
networking

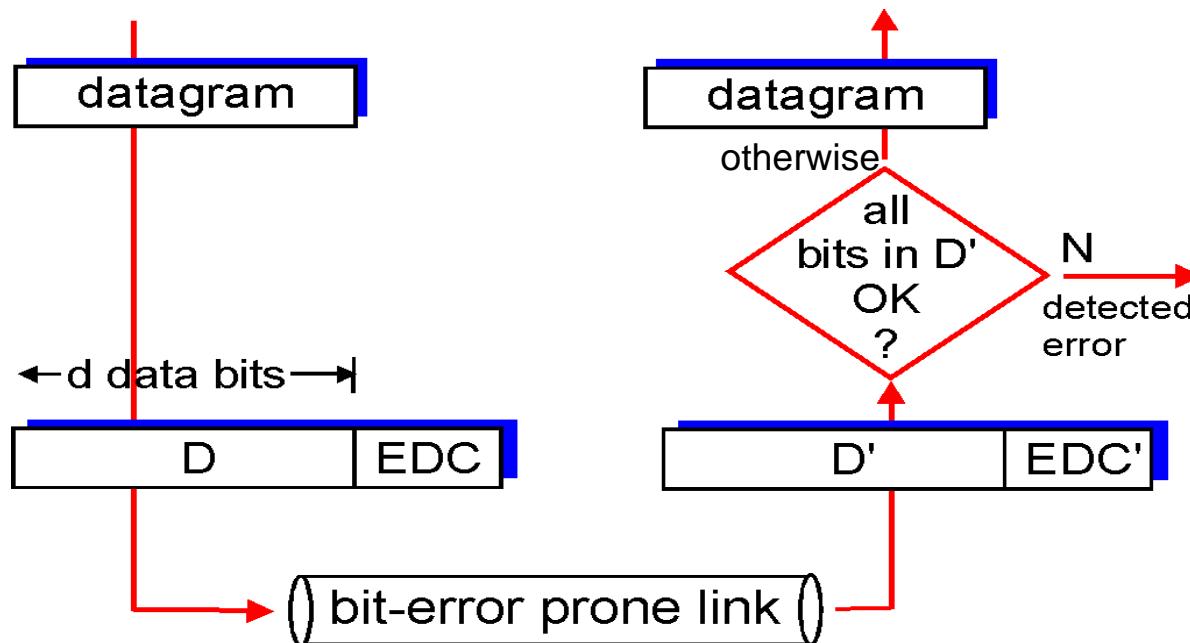
5.7 a day in the life of a
web request

Error detection

EDC= Error Detection and Correction bits (redundancy)

D = Data protected by error checking, may include header fields

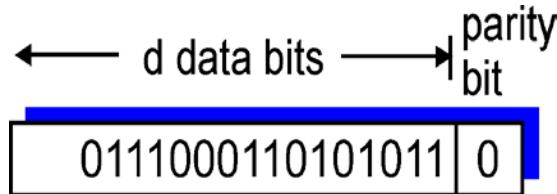
- Error detection not 100% reliable!
 - protocol may miss some errors, but rarely
 - larger EDC field yields better detection and correction



Parity checking

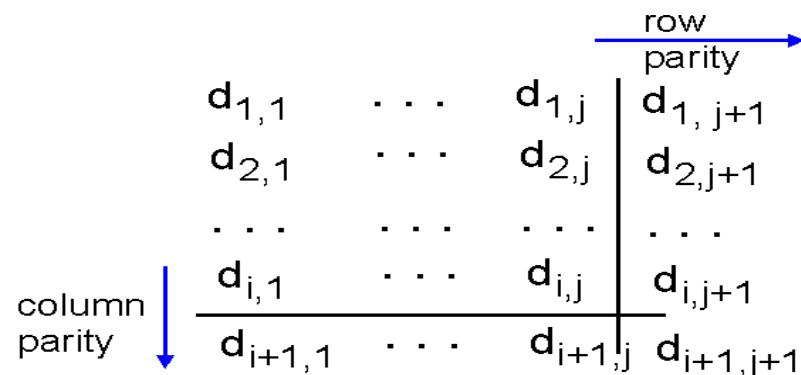
single bit parity:

- ❖ detect odd number of bit errors



two-dimensional bit parity:

- ❖ can correct single bit errors and detect multiple bit errors



1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
<hr/>					
0	0	1	0	1	0

no errors

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
<hr/>					
0	0	1	0	1	0

*correctable
single bit error*

Internet checksum (review)

goal: detect “errors” (e.g., flipped bits) in transmitted packet
(note: used at transport layer *only*)

sender:

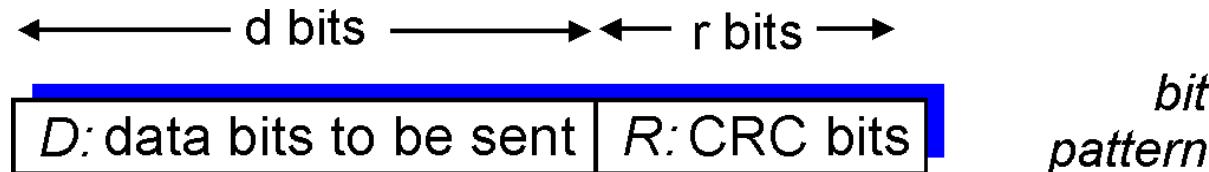
- ❖ treat segment contents as sequence of 16-bit integers
- ❖ checksum: addition ($1's$ complement sum) of segment contents
- ❖ sender puts checksum value into UDP checksum field

receiver:

- ❖ compute checksum of received segment
- ❖ check if computed checksum equals checksum field value:
 - NO - error detected
 - YES - no error detected.

Cyclic redundancy check

- ❖ more powerful error-detection coding
- ❖ view data bits, **D**, as a binary number
- ❖ choose $r+1$ bit pattern (generator), **G**
- ❖ goal: choose r CRC bits, **R**, such that
 - $\langle D, R \rangle$ exactly divisible by G (modulo 2)
 - receiver knows G, divides $\langle D, R \rangle$ by G. If non-zero remainder: error detected!
 - can detect all burst errors less than $r+1$ bits
- ❖ widely used in practice (Ethernet, 802.11 WiFi, ATM)



$$D * 2^r \text{ XOR } R$$

mathematical formula

CRC example

want:

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

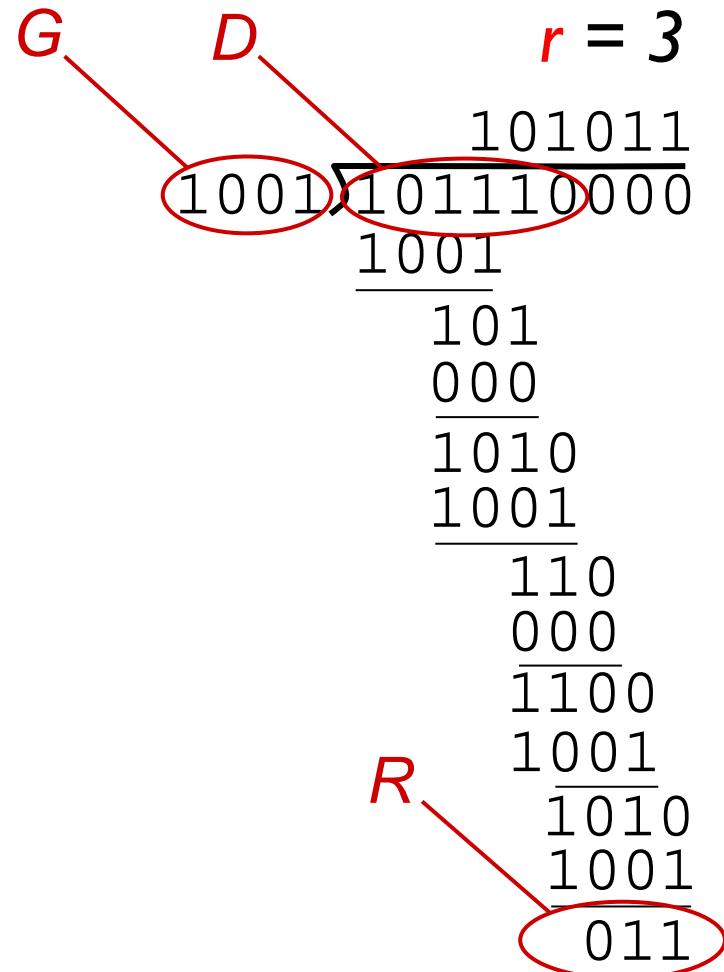
equivalently:

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

equivalently:

if we divide $D \cdot 2^r$ by
G, want remainder R
to satisfy:

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



Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

5.5 link virtualization:
MPLS

5.6 data center
networking

5.7 a day in the life of a
web request

Multiple access links, protocols

two types of “links”:

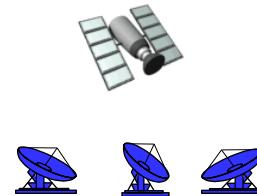
- ❖ **point-to-point**
 - PPP for dial-up access
 - point-to-point link between Ethernet switch, host
- ❖ **broadcast (shared wire or medium)**
 - old-fashioned Ethernet
 - upstream HFC
 - 802.11 wireless LAN



shared wire (e.g.,
cabled Ethernet)



shared wireless
(e.g., 802.11 WiFi)



satellite



humans at a
cocktail party
(shared air, acoustical)

Multiple access protocols

- ❖ single shared broadcast channel
- ❖ two or more simultaneous transmissions by nodes:
interference
 - **collision** if node receives two or more signals at the same time

multiple access protocol

- ❖ distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- ❖ communication about channel sharing must use channel itself!
 - no out-of-band channel for coordination

An ideal multiple access protocol

given: broadcast channel of rate R bps

desiderata:

1. when one node wants to transmit, it can send at rate R .
2. when M nodes want to transmit, each can send at average rate R/M
3. fully decentralized:
 - no special node to coordinate transmissions
 - no synchronization of clocks, slots
4. simple

MAC protocols: taxonomy

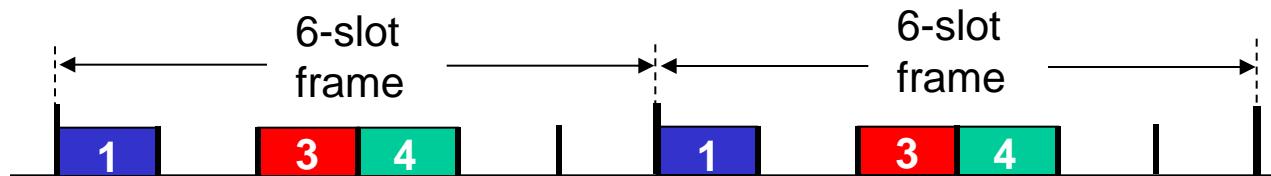
three broad classes:

- ❖ *channel partitioning*
 - divide channel into smaller “pieces” (time slots, frequency, code)
 - allocate piece to node for exclusive use
- ❖ *random access*
 - channel not divided, allow collisions
 - “recover” from collisions
- ❖ *“taking turns”*
 - nodes take turns, but nodes with more to send can take longer turns

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

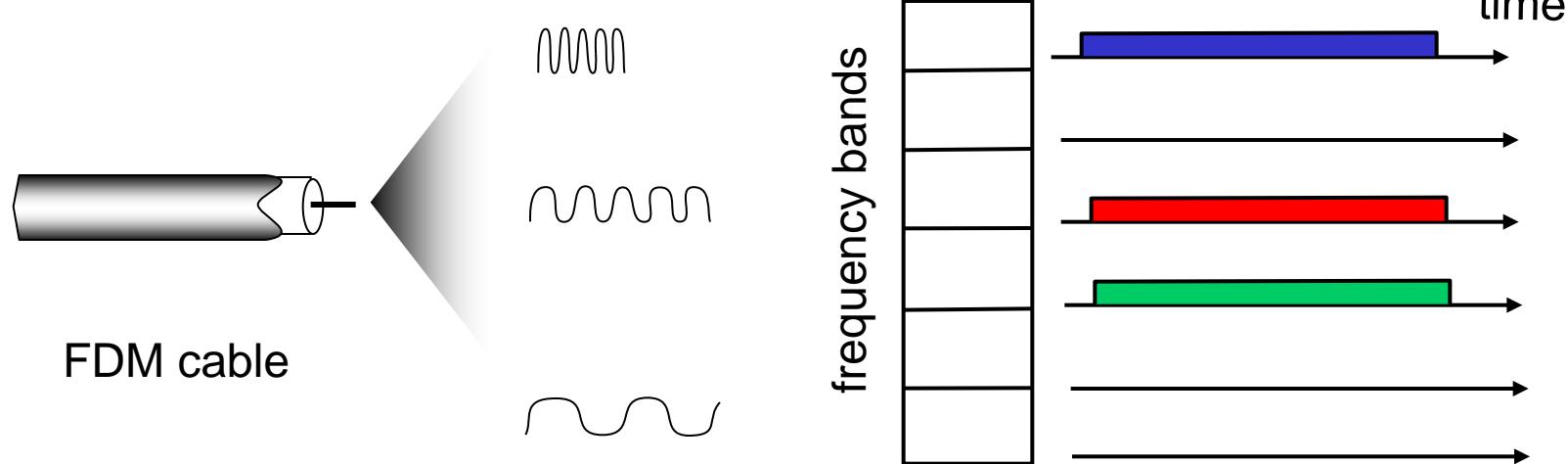
- ❖ access to channel in "rounds"
- ❖ each station gets fixed length slot (length = pkt trans time) in each round
- ❖ unused slots go idle
- ❖ example: 6-station LAN, 1,3,4 have pkt, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

- ❖ channel spectrum divided into frequency bands
- ❖ each station assigned fixed frequency band
- ❖ unused transmission time in frequency bands go idle
- ❖ example: 6-station LAN, 1,3,4 have pkt, frequency bands 2,5,6 idle



Random access protocols

- ❖ when node has packet to send
 - transmit at full channel data rate R.
 - no *a priori* coordination among nodes
- ❖ two or more transmitting nodes → “collision”,
- ❖ **random access MAC protocol** specifies:
 - how to detect collisions
 - how to recover from collisions (e.g., via delayed retransmissions)
- ❖ examples of random access MAC protocols:
 - slotted ALOHA
 - ALOHA
 - CSMA, CSMA/CD, CSMA/CA

Slotted ALOHA

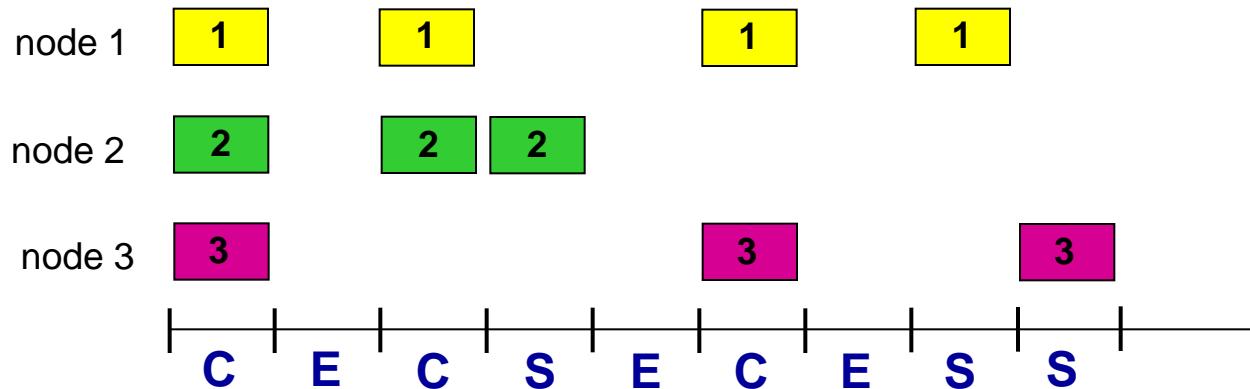
assumptions:

- ❖ all frames same size
- ❖ time divided into equal size slots (time to transmit 1 frame)
- ❖ nodes start to transmit only at slot beginning
- ❖ nodes are synchronized
- ❖ if 2 or more nodes transmit in slot, all nodes detect collision

operation:

- ❖ when node obtains fresh frame, transmits in next slot
 - *if no collision:* node can send new frame in next slot
 - *if collision:* node retransmits frame in each subsequent slot with prob. p until success

Slotted ALOHA



Pros:

- ❖ single active node can continuously transmit at full rate of channel
- ❖ highly decentralized: only slots in nodes need to be in sync
- ❖ simple

Cons:

- ❖ collisions, wasting slots
- ❖ idle slots
- ❖ nodes must be able to detect collision in less than time to transmit packet
- ❖ clock synchronization

Slotted ALOHA: efficiency

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

- ❖ suppose: N nodes with many frames to send, each transmits in slot with probability p
- ❖ prob that given node has success in a slot = $p(1-p)^{N-1}$
- ❖ prob that *any* node has a success = $Np(1-p)^{N-1}$

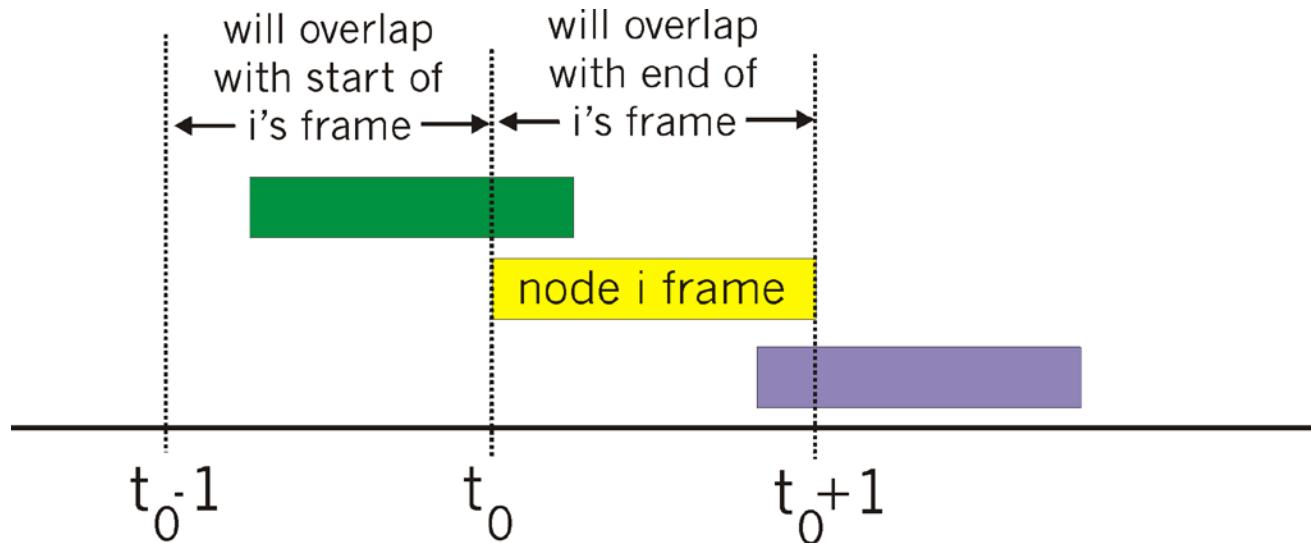
- ❖ max efficiency: find p^* that maximizes $Np(1-p)^{N-1}$
- ❖ for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives:
max efficiency = 1/e = .37

at best: channel used for useful transmissions 37% of time!

!

Pure (unslotted) ALOHA

- ❖ unslotted Aloha: simpler, no synchronization
- ❖ when frame first arrives
 - transmit immediately
- ❖ collision probability increases:
 - frame sent at t_0 collides with other frames sent in $[t_0 - l, t_0 + l]$



Pure ALOHA efficiency

$$\begin{aligned} P(\text{success by given node}) &= P(\text{node transmits}) \cdot \\ &\quad P(\text{no other node transmits in } [t_0-l, t_0]) \cdot \\ &\quad P(\text{no other node transmits in } [t_0-l, t_0]) \\ &= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1} \\ &= p \cdot (1-p)^{2(N-1)} \\ \dots \text{ choosing optimum } p \text{ and then letting } n &\rightarrow \infty \\ &= 1/(2e) = .18 \end{aligned}$$

even worse than slotted Aloha!

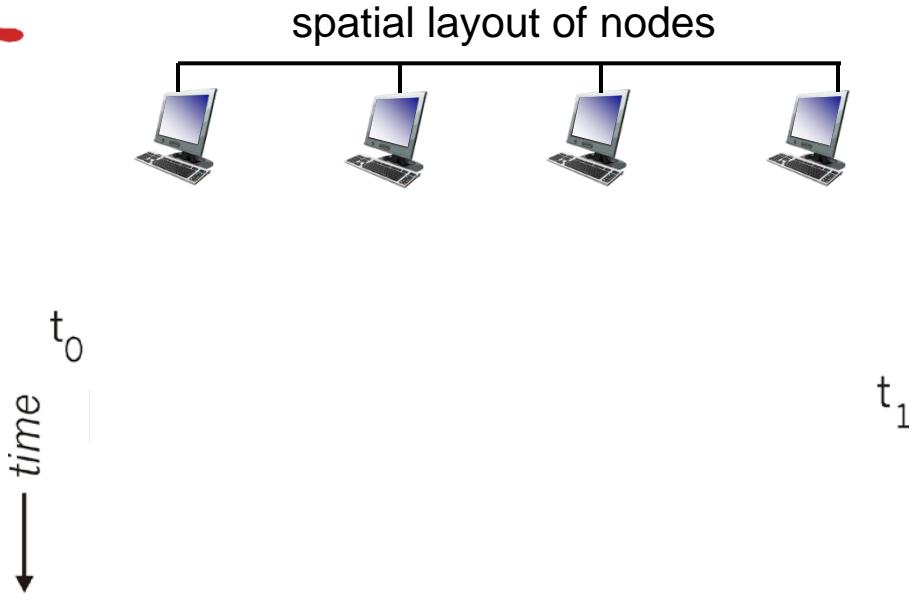
CSMA (carrier sense multiple access)

CSMA: listen before transmit:

- if channel sensed idle: transmit entire frame
 - ❖ if channel sensed busy, defer transmission
 - ❖ human analogy: don't interrupt others!

CSMA collisions

- ❖ **collisions can still occur:** propagation delay means two nodes may not hear each other's transmission
- ❖ **collision:** entire packet transmission time wasted
 - distance & propagation delay play role in determining collision probability

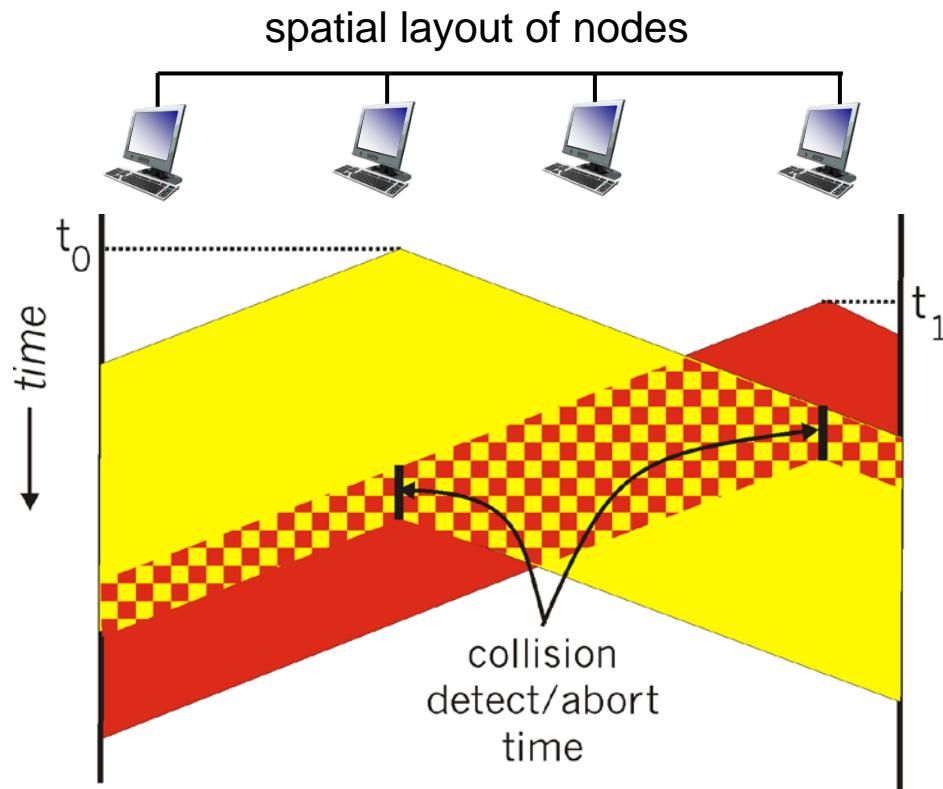


CSMA/CD (collision detection)

CSMA/CD: carrier sensing, deferral as in CSMA

- collisions detected within short time
- colliding transmissions aborted, reducing channel wastage
- ❖ collision detection:
 - easy in wired LANs: measure signal strengths, compare transmitted, received signals
 - difficult in wireless LANs: received signal strength overwhelmed by local transmission strength
- ❖ human analogy: the polite conversationalist

CSMA/CD (collision detection)



Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts and sends jam signal
5. After aborting, NIC enters *binary (exponential) backoff*:
 - after m th collision, NIC chooses K at random from $\{0, 1, 2, \dots, 2^m - 1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2
 - longer backoff interval with more collisions

CSMA/CD efficiency

- ❖ T_{prop} = max prop delay between 2 nodes in LAN
- ❖ t_{trans} = time to transmit max-size frame

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

- ❖ efficiency goes to 1
 - as t_{prop} goes to 0
 - as t_{trans} goes to infinity
- ❖ better performance than ALOHA: and simple, cheap, decentralized!

“Taking turns” MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, I/N bandwidth allocated even if only 1 active node!

random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

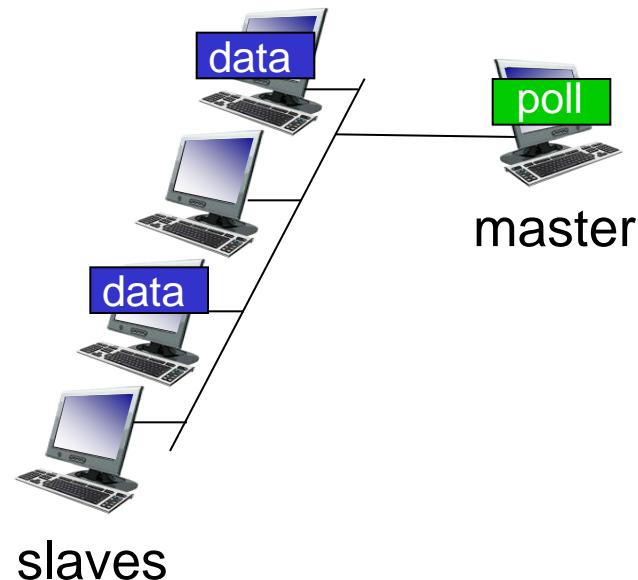
“taking turns” protocols

look for best of both worlds!

“Taking turns” MAC protocols

polling:

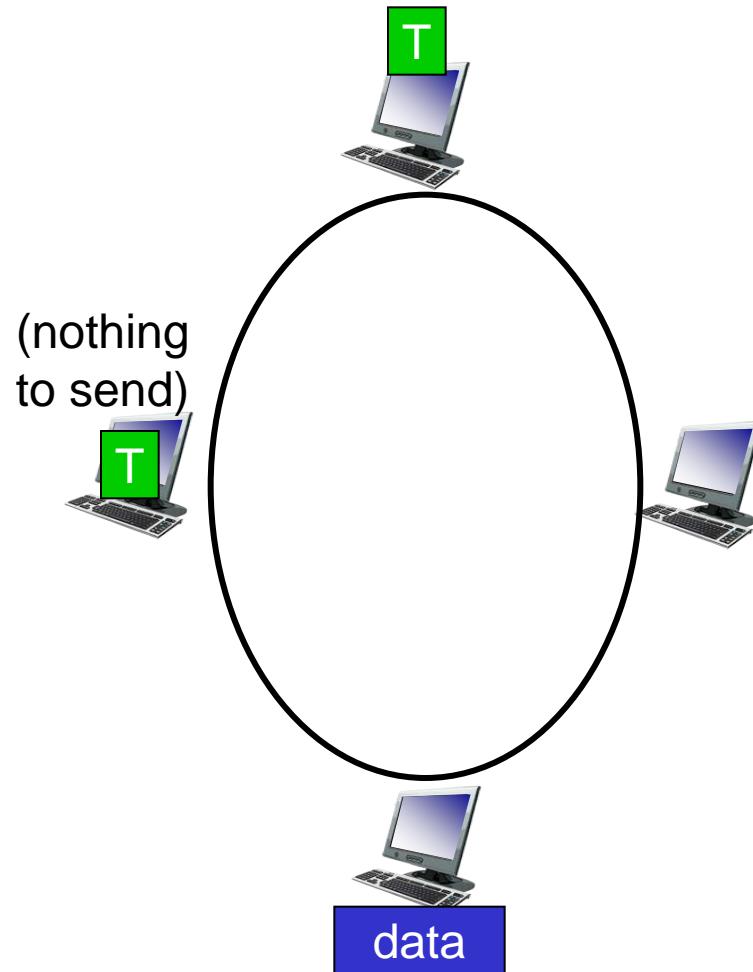
- ❖ master node “invites” slave nodes to transmit in turn
- ❖ typically used with “dumb” slave devices
- ❖ concerns:
 - polling overhead
 - latency
 - single point of failure (master)



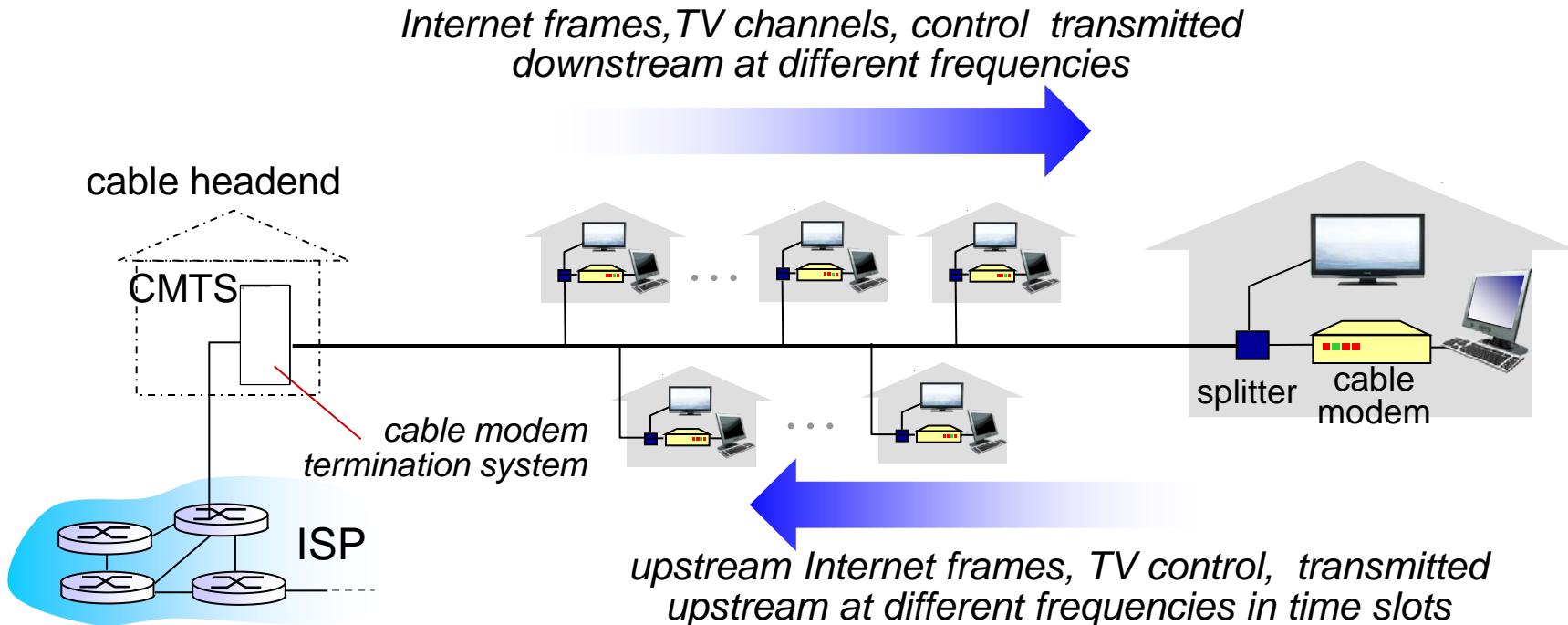
“Taking turns” MAC protocols

token passing:

- ❖ control **token** passed from one node to next sequentially.
- ❖ token message
- ❖ concerns:
 - token overhead
 - latency
 - single point of failure (token)

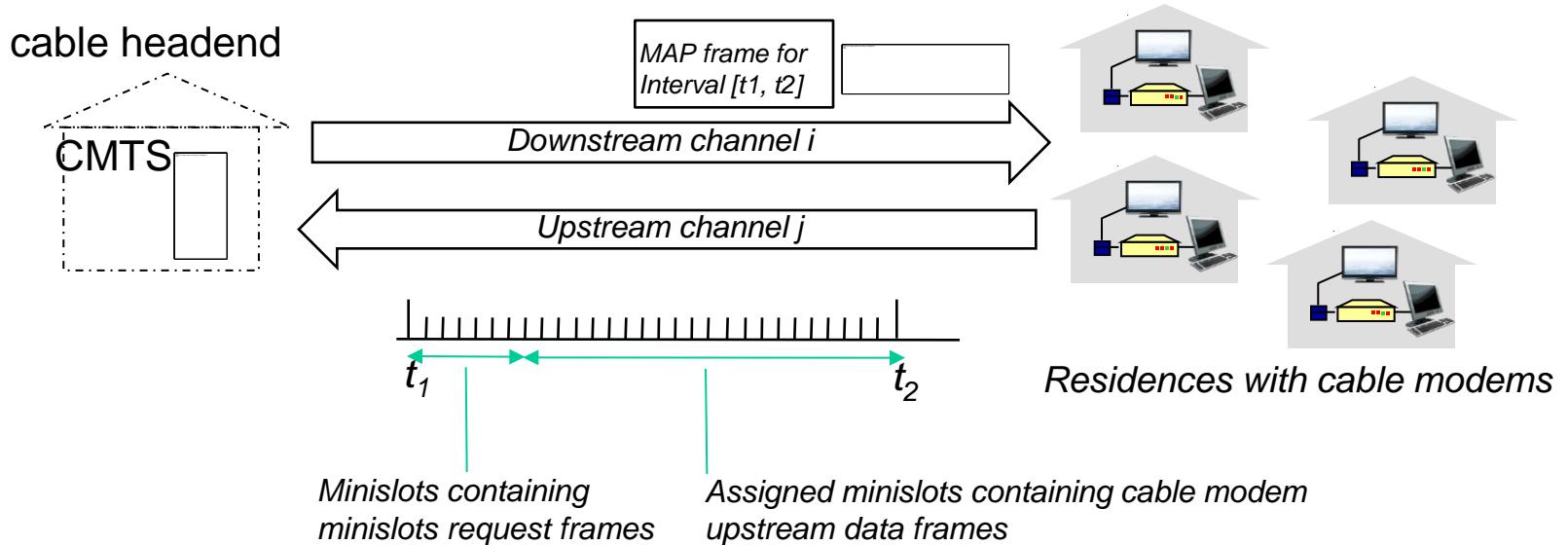


Cable access network



- ❖ **multiple** 40Mbps downstream (broadcast) channels
 - **single** CMTS transmits into channels
- ❖ **multiple** 30 Mbps upstream channels
 - **multiple access:** **all** users contend for certain upstream channel time slots (others assigned)

Cable access network



DOCSIS: data over cable service interface spec

- ❖ FDM over upstream, downstream frequency channels
- ❖ TDM upstream: some slots assigned, some have contention
 - downstream MAP frame: assigns upstream slots
 - request for upstream slots (and data) transmitted random access (binary backoff) in selected slots

Summary of MAC protocols

- ❖ *channel partitioning*, by time, frequency or code
 - Time Division, Frequency Division
- ❖ *random access* (dynamic),
 - ALOHA, S-ALOHA, CSMA, CSMA/CD
 - carrier sensing: easy in some technologies (wire), hard in others (wireless)
 - CSMA/CD used in Ethernet
 - CSMA/CA used in 802.11
- ❖ *taking turns*
 - polling from central site, token passing
 - bluetooth, FDDI, token ring

Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

5.5 link virtualization:
MPLS

5.6 data center
networking

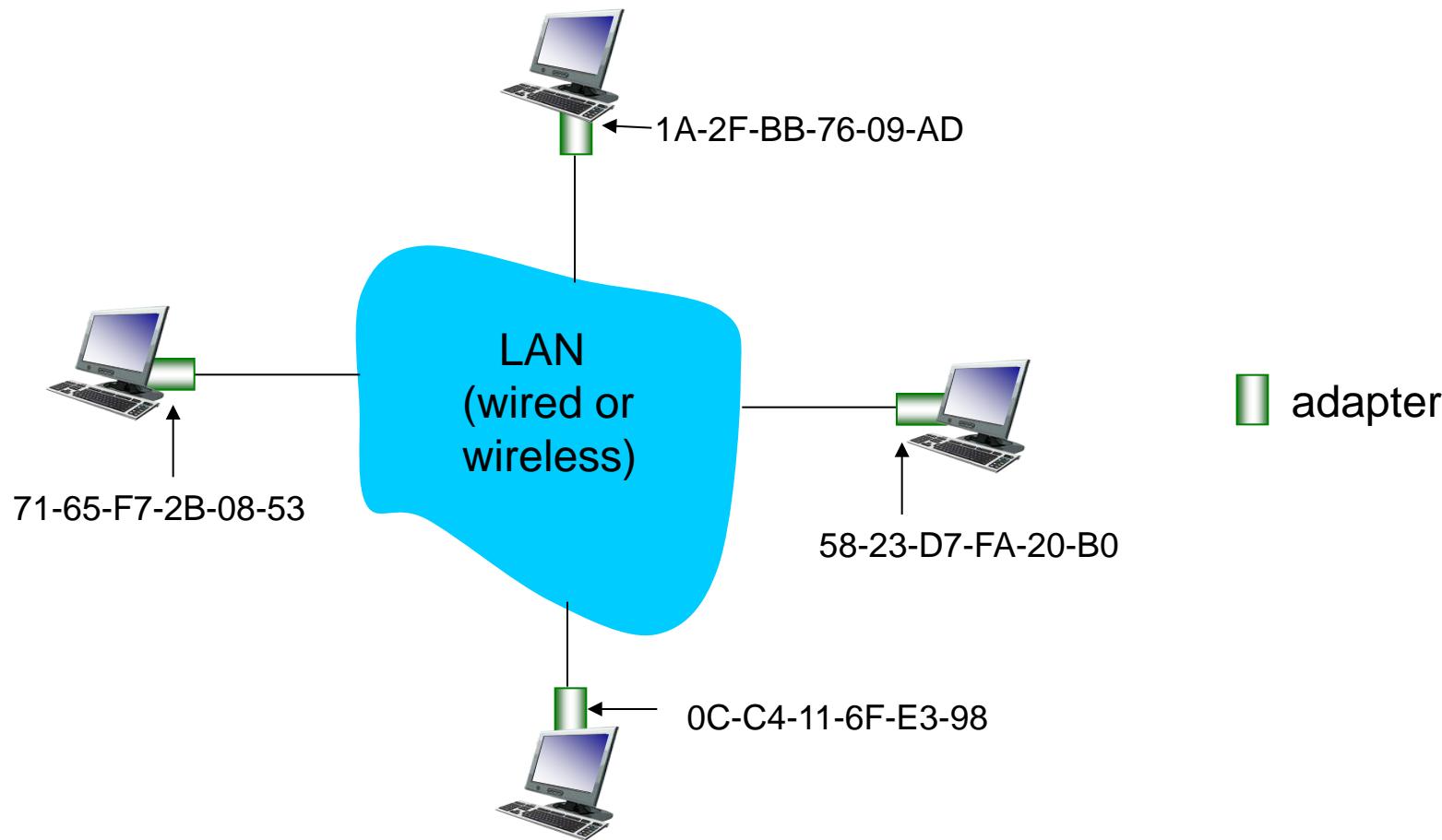
5.7 a day in the life of a
web request

MAC addresses and ARP

- ❖ 32-bit IP address:
 - *network-layer address for interface*
 - used for layer 3 (network layer) forwarding
- ❖ MAC (or LAN or physical or Ethernet) address:
 - function: *used ‘locally’ to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
 - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
 - e.g.: IA-2F-BB-76-09-AD
 - hexadecimal (base 16) notation
 - (each “number” represents 4 bits)

LAN addresses and ARP

each adapter on LAN has unique *LAN* address

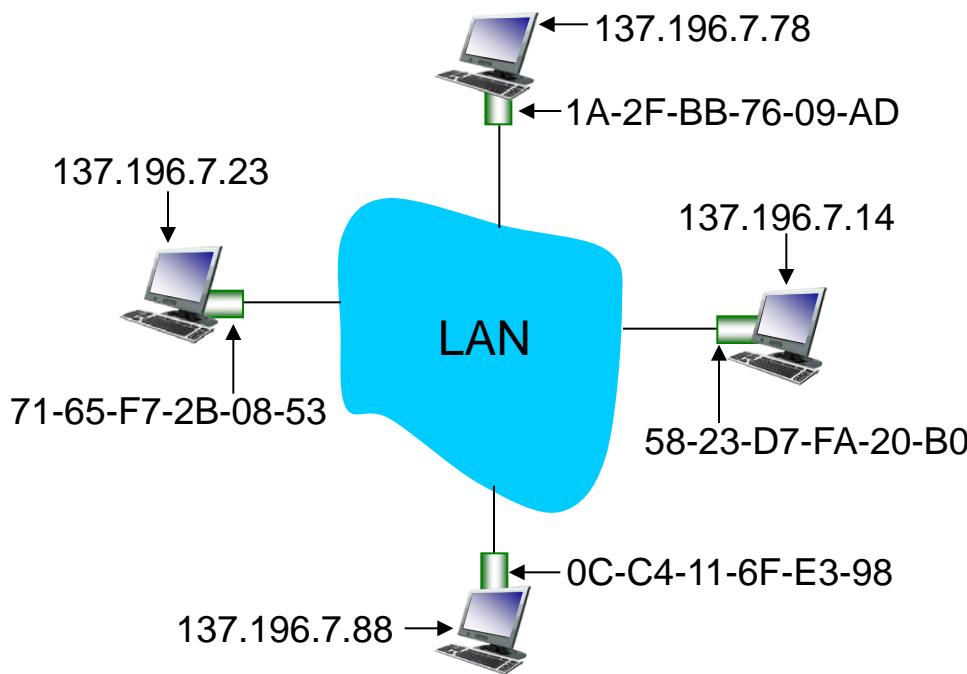


LAN addresses (more)

- ❖ MAC address allocation administered by IEEE
- ❖ manufacturer buys portion of MAC address space (to assure uniqueness)
- ❖ analogy:
 - MAC address: like Social Security Number
 - IP address: like postal address
- ❖ MAC flat address → portability
 - can move LAN card from one LAN to another
- ❖ IP hierarchical address *not* portable
 - address depends on IP subnet to which node is attached

ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?



ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:
<IP address; MAC address; TTL>
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

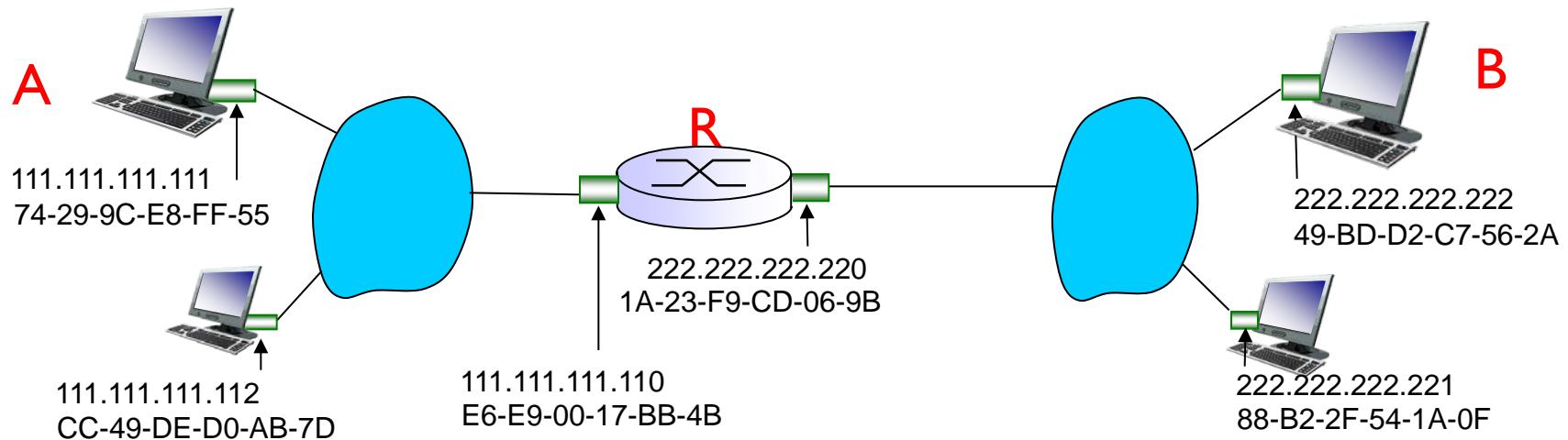
ARP protocol: same LAN

- ❖ A wants to send datagram to B
 - B's MAC address not in A's ARP table.
- ❖ A **broadcasts** ARP query packet, containing B's IP address
 - dest MAC address = FF-FF-FF-FF-FF-FF
 - all nodes on LAN receive ARP query
- ❖ B receives ARP packet, replies to A with its (B's) MAC address
 - frame sent to A's MAC address (unicast)
- ❖ A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
 - soft state: information that times out (goes away) unless refreshed
- ❖ ARP is “plug-and-play”:
 - nodes create their ARP tables *without intervention from net administrator*

Addressing: routing to another LAN

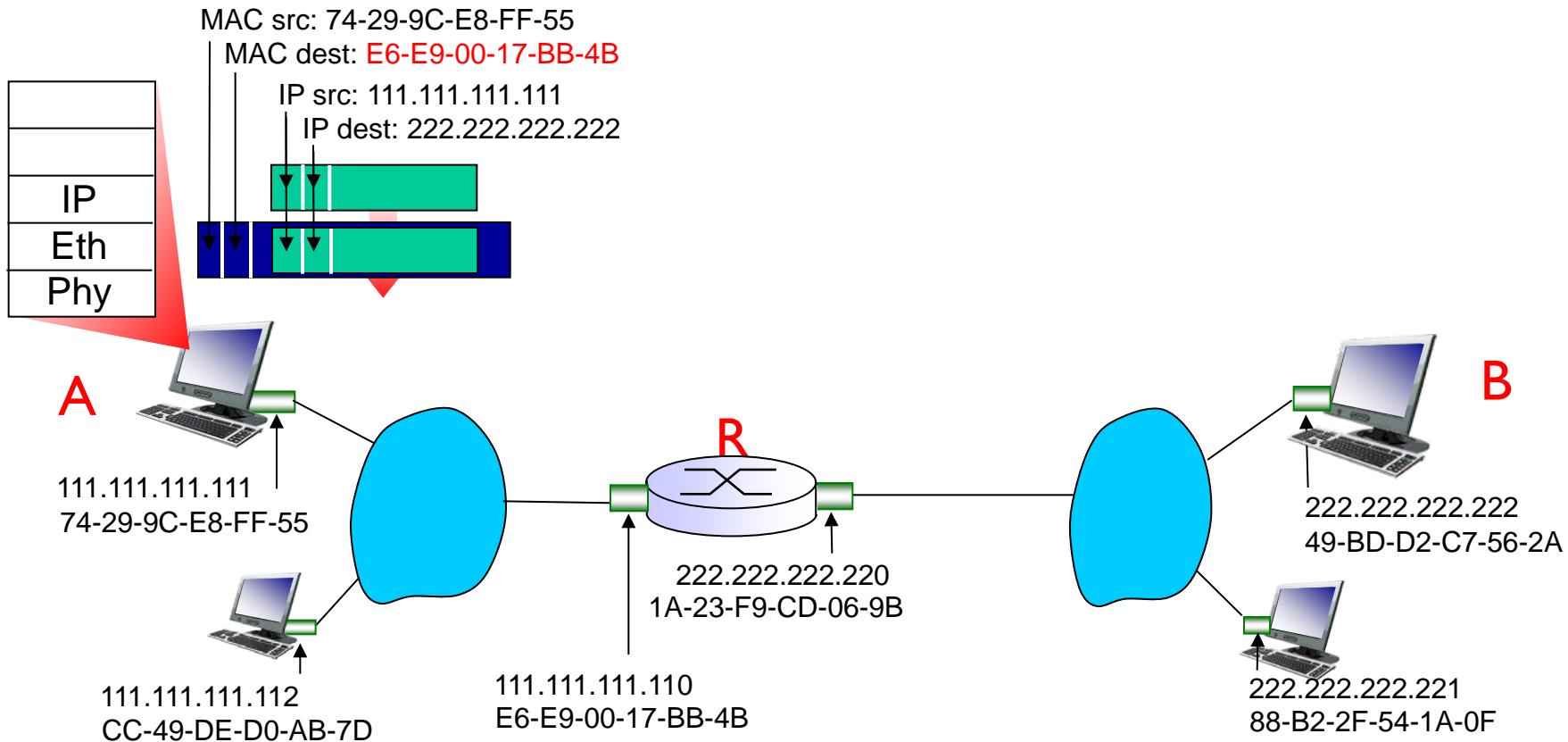
walkthrough: send datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)



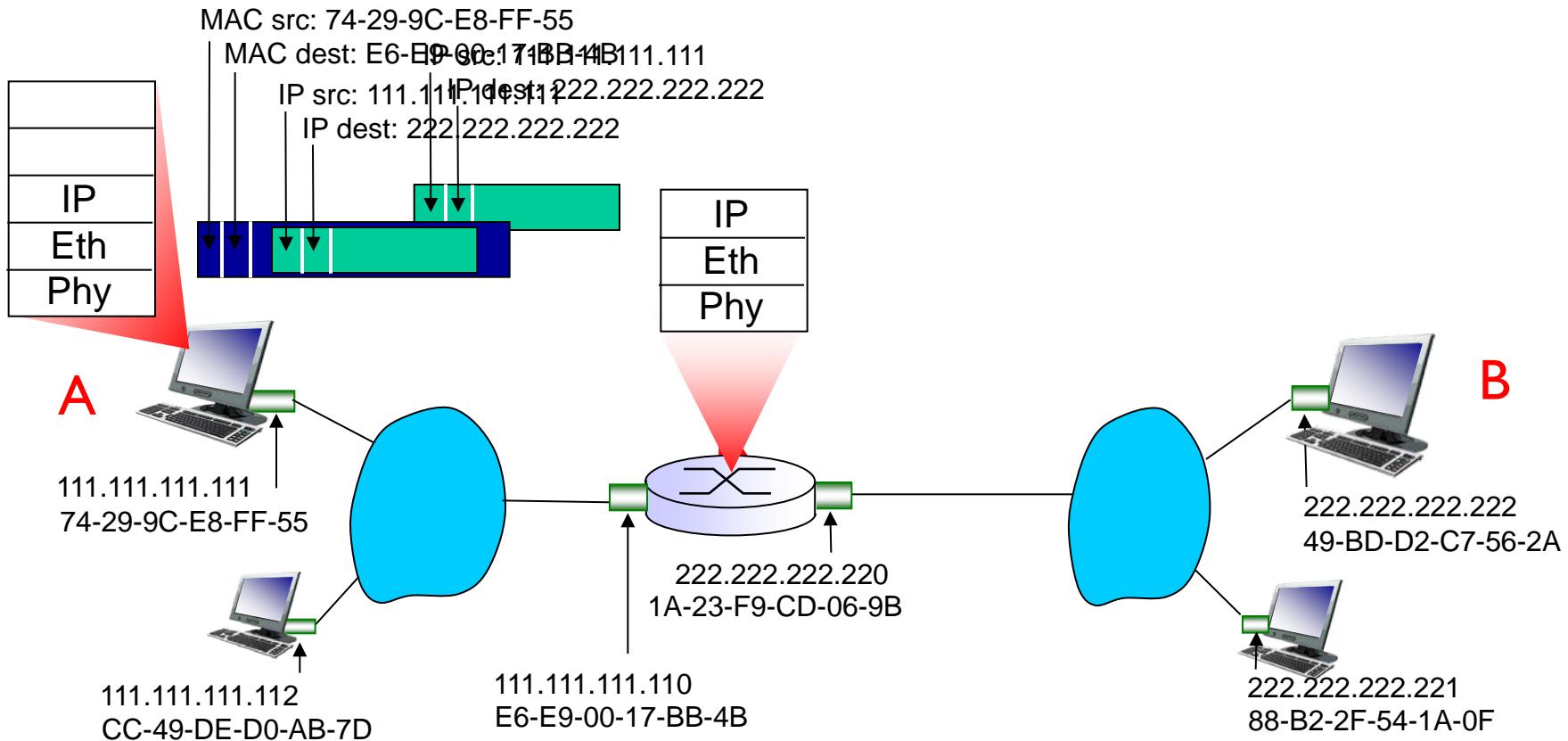
Addressing: routing to another LAN

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



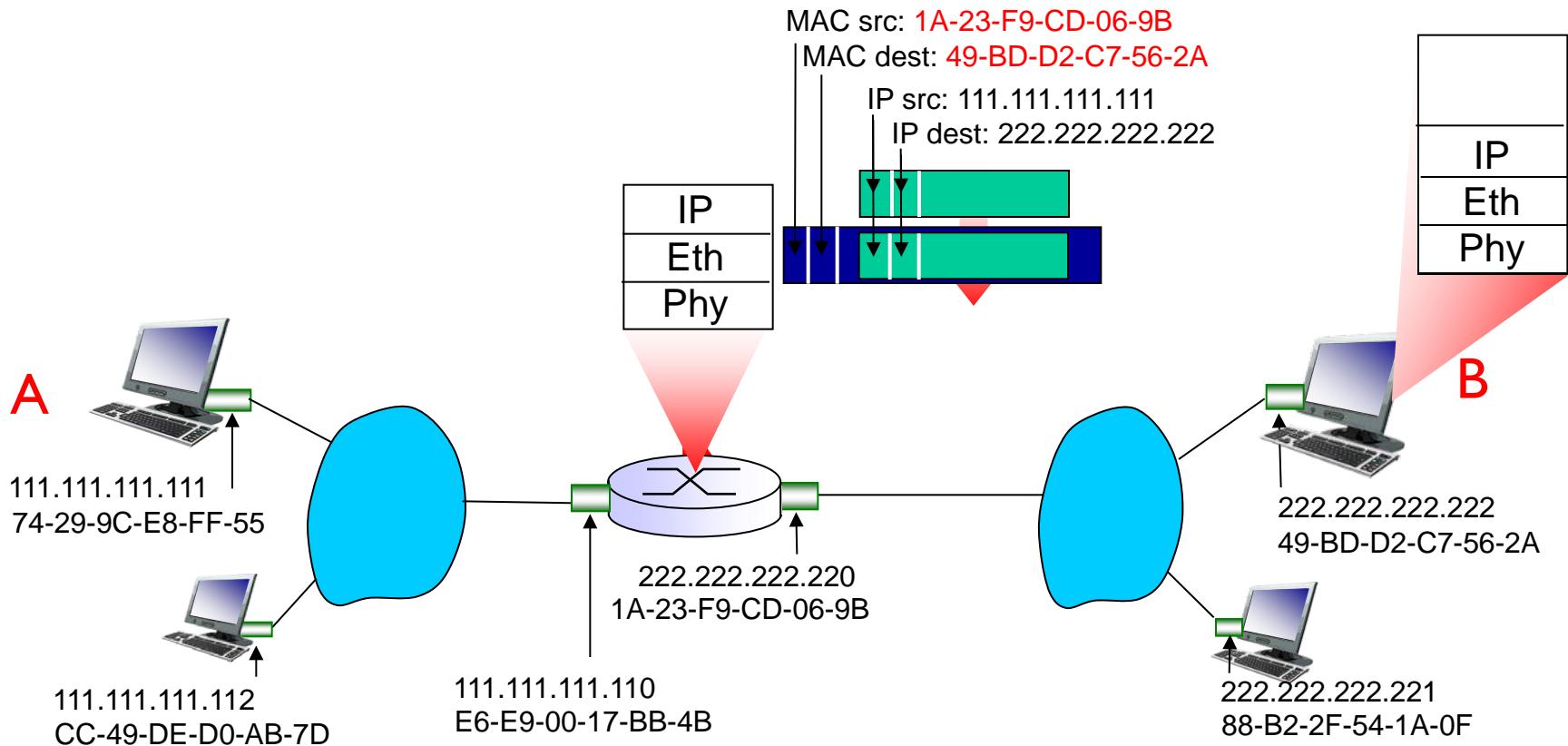
Addressing: routing to another LAN

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



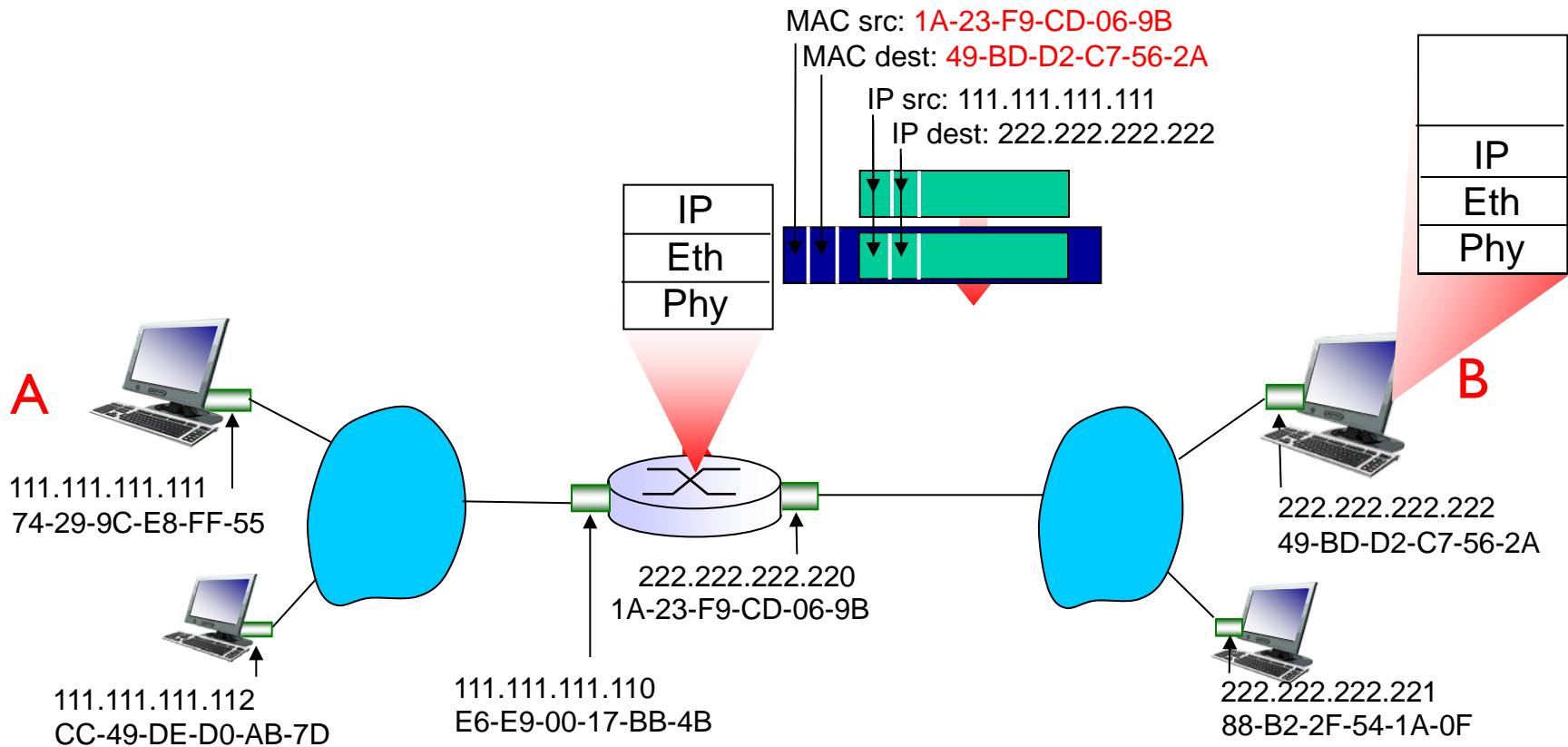
Addressing: routing to another LAN

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



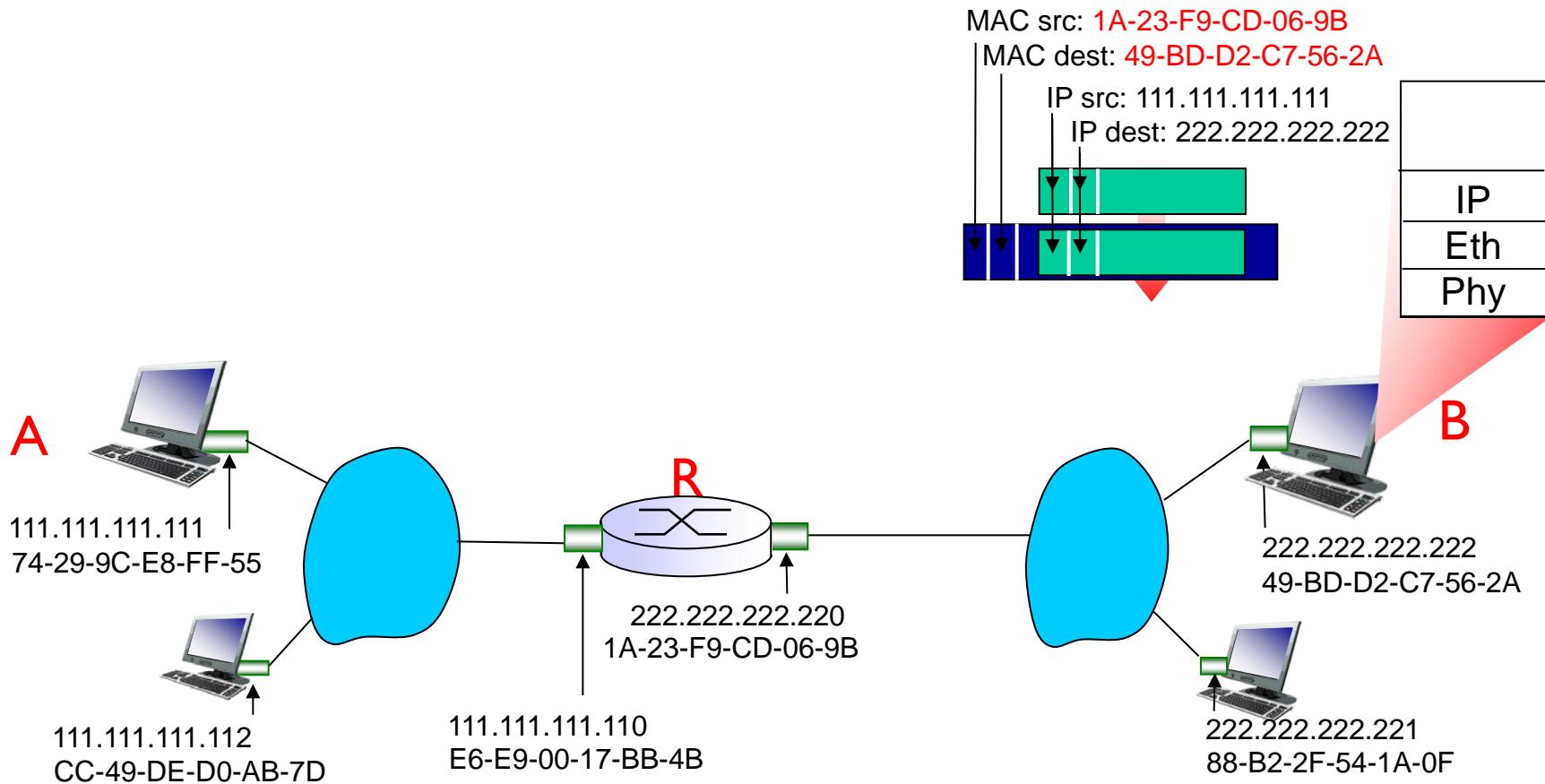
Addressing: routing to another LAN

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



Addressing: routing to another LAN

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



Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- **Ethernet**
- switches
- VLANS

5.5 link virtualization:
MPLS

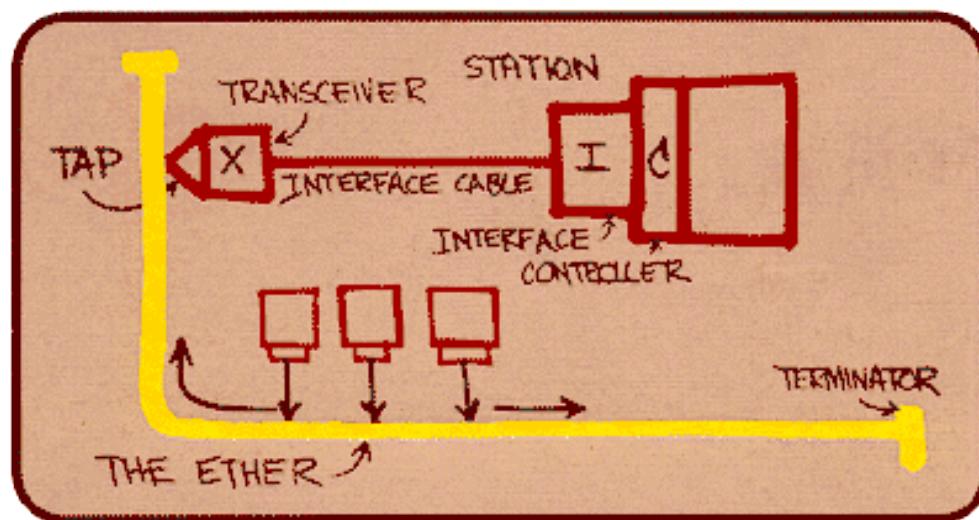
5.6 data center
networking

5.7 a day in the life of a
web request

Ethernet

“dominant” wired LAN technology:

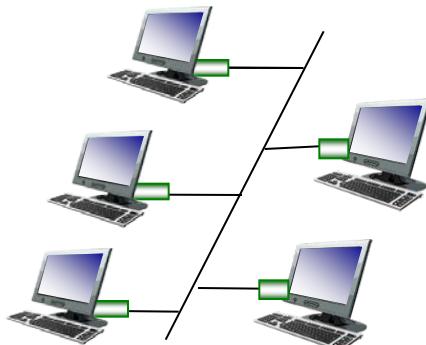
- ❖ cheap \$20 for NIC
- ❖ first widely used LAN technology
- ❖ simpler, cheaper than token LANs and ATM
- ❖ kept up with speed race: 10 Mbps – 10 Gbps



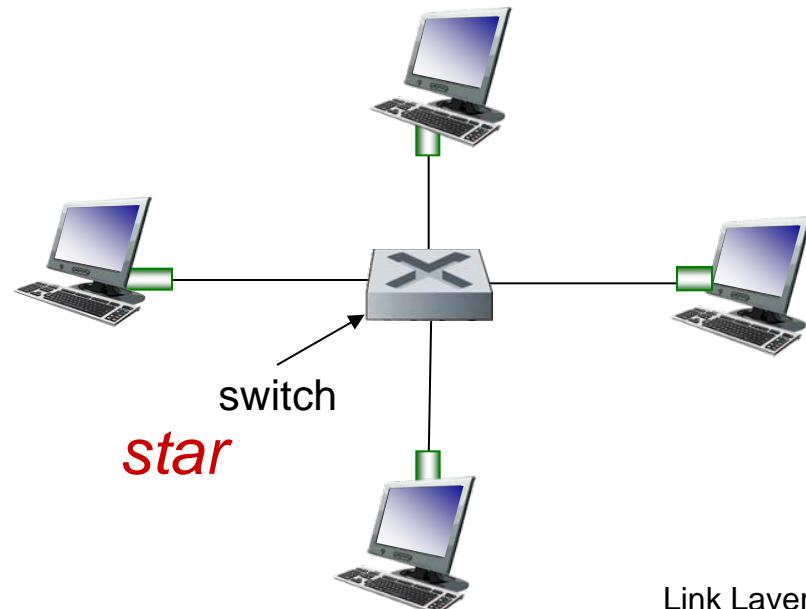
Metcalfe's Ethernet sketch

Ethernet: physical topology

- ❖ **bus:** popular through mid 90s
 - all nodes in same collision domain (can collide with each other)
- ❖ **star:** prevails today
 - active **switch** in center
 - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)



bus: coaxial cable



star

Ethernet frame structure

sending adapter encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**



preamble:

- ❖ 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- ❖ used to synchronize receiver, sender clock rates

Ethernet frame structure (more)

- ❖ **addresses:** 6 byte source, destination MAC addresses
 - if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
 - otherwise, adapter discards frame
- ❖ **type:** indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- ❖ **CRC:** cyclic redundancy check at receiver
 - error detected: frame is dropped



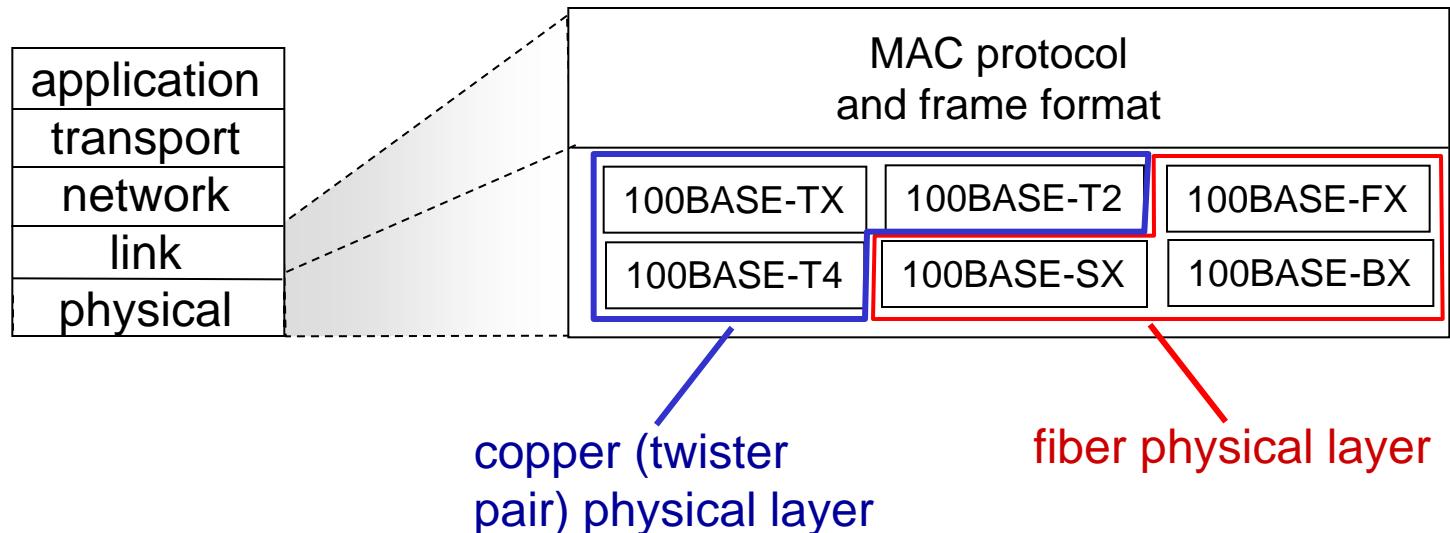
Ethernet: unreliable, connectionless

- ❖ **connectionless:** no handshaking between sending and receiving NICs
- ❖ **unreliable:** receiving NIC doesn't send acks or nacks to sending NIC
 - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- ❖ Ethernet's MAC protocol: unslotted ***CSMA/CD wth binary backoff***

802.3 Ethernet standards: link & physical layers

- ❖ *many* different Ethernet standards

- common MAC protocol and frame format
- different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1 Gbps, 10G bps
- different physical layer media: fiber, cable



Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

5.5 link virtualization:
MPLS

5.6 data center
networking

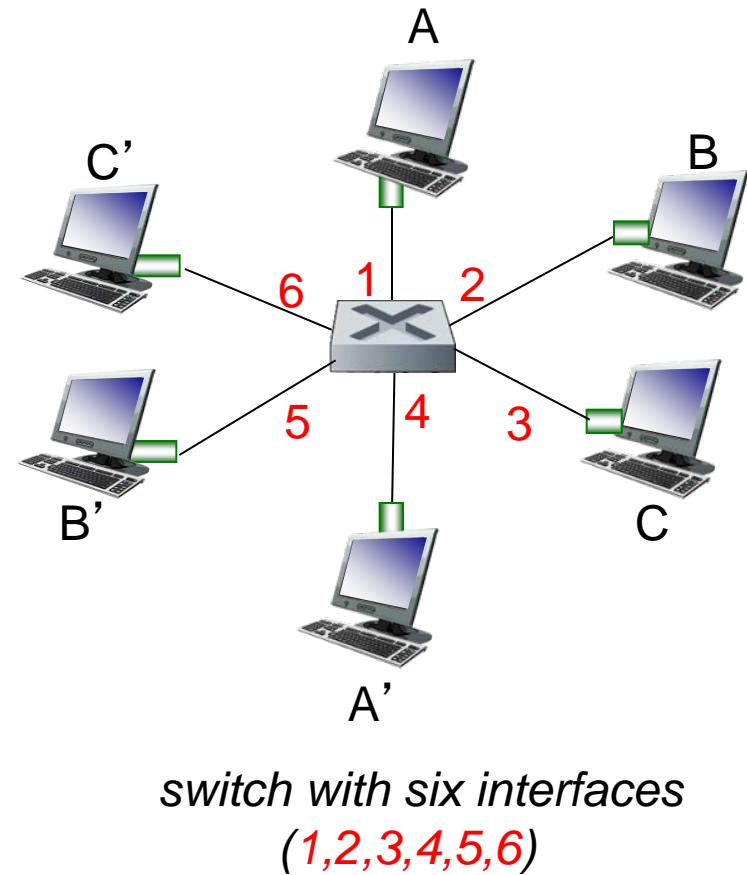
5.7 a day in the life of a
web request

Ethernet switch

- ❖ **link-layer device: takes an *active role***
 - store, forward Ethernet frames
 - examine incoming frame's MAC address, **selectively forward** frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- ❖ ***transparent***
 - hosts are unaware of presence of switches
- ❖ ***plug-and-play, self-learning***
 - switches do not need to be configured

Switch: multiple simultaneous transmissions

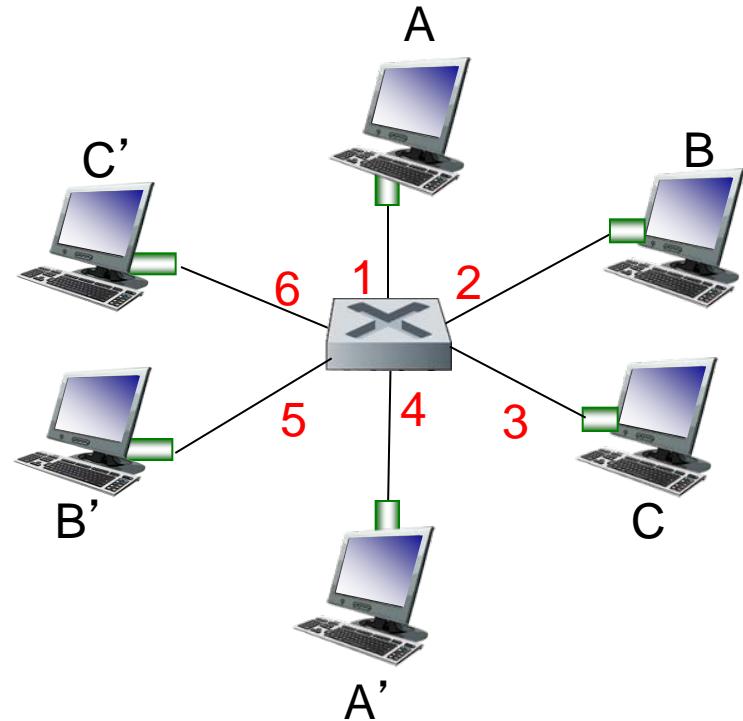
- ❖ hosts have dedicated, direct connection to switch
- ❖ switches **buffer** packets
- ❖ Ethernet protocol used on each incoming link, but no collisions; full duplex
 - each link is its own collision domain
- ❖ *switching*: A-to-A' and B-to-B' can transmit simultaneously, without collisions



Switch forwarding table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- ❖ **A:** each switch has a *switch table*, each entry:
 - (MAC address of host, interface to reach host, time stamp)
 - looks like a *routing table*!



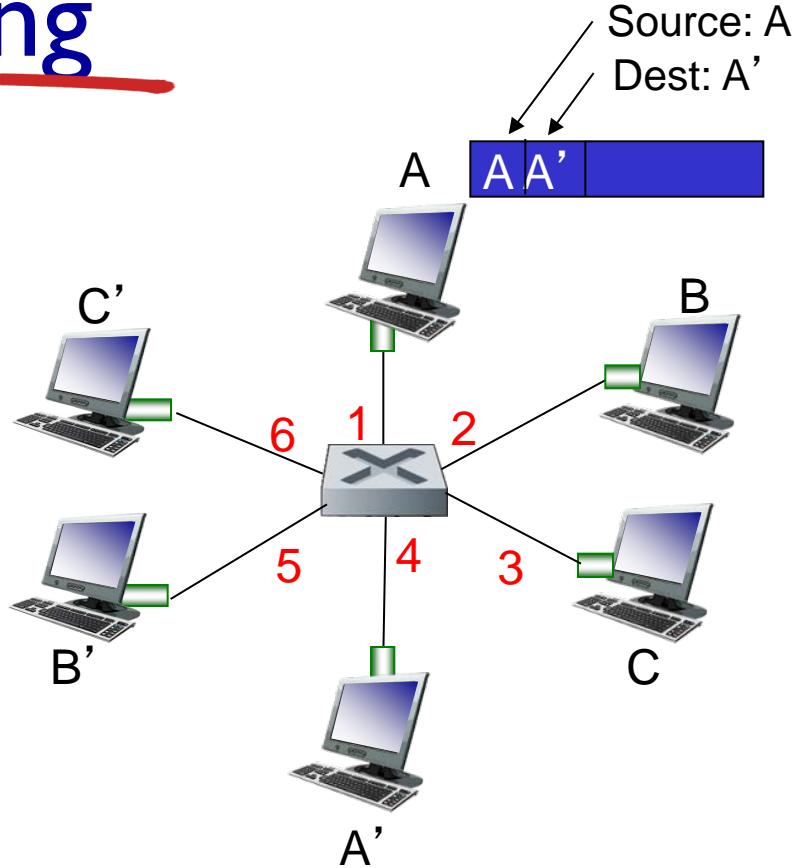
Q: how are entries created, maintained in switch table?

- something like a *routing protocol*?

switch with six interfaces
(1,2,3,4,5,6)

Switch: self-learning

- switch *learns* which hosts can be reached through which interfaces
 - when frame received, switch “learns” location of sender: incoming LAN segment
 - records sender/location pair in switch table



MAC addr	interface	TTL
A	1	60

*Switch table
(initially empty)*

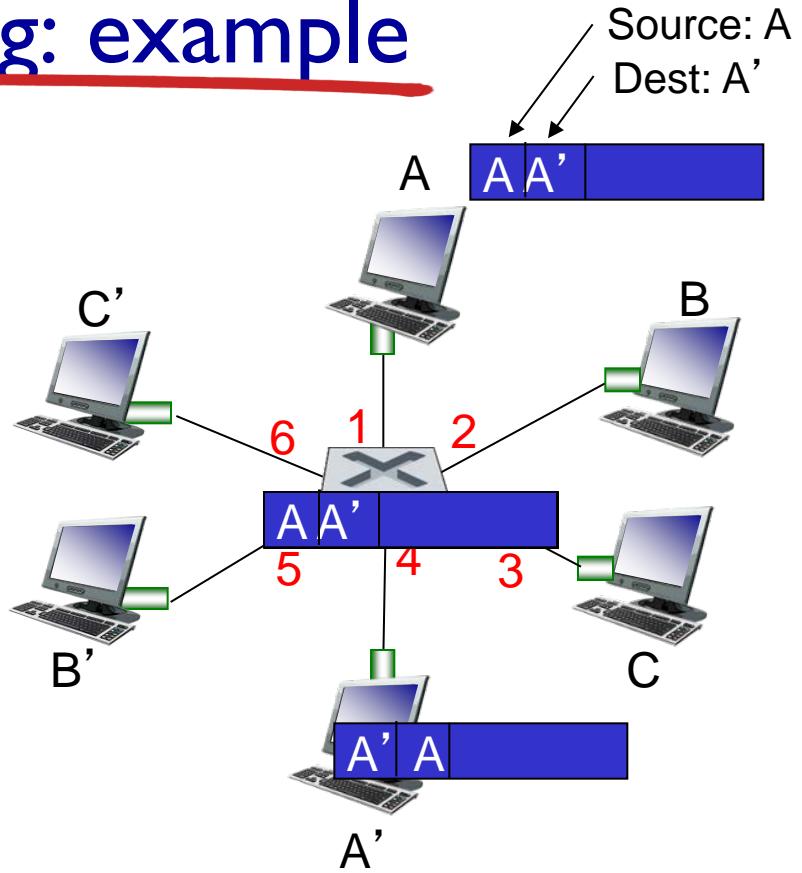
Switch: frame filtering/forwarding

when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
 then {
 if destination on segment from which frame arrived
 then drop frame
 else forward frame on interface indicated by entry
 }
 else flood /* forward on all interfaces except arriving
 interface */

Self-learning, forwarding: example

- ❖ frame destination, A', location unknown: *flood*
- ❖ destination A location known: *selectively send on just one link*

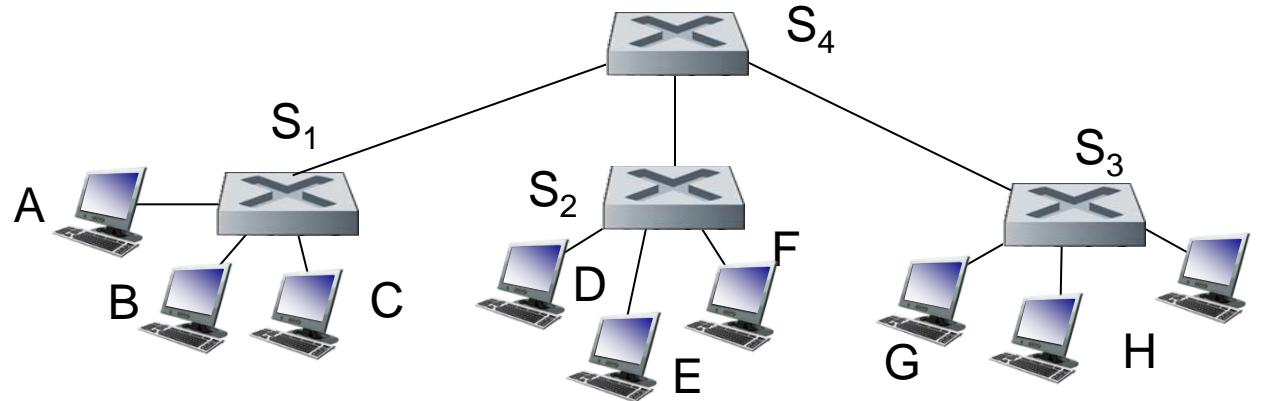


*switch table
(initially empty)*

MAC addr	interface	TTL
A	1	60
A'	4	60

Interconnecting switches

- ❖ switches can be connected together

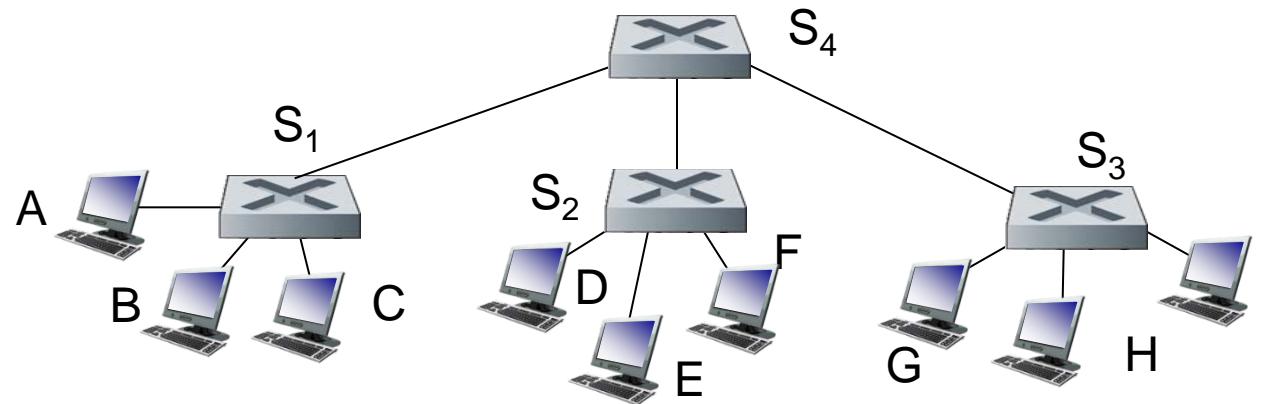


Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3 ?

- ❖ A: self learning! (works exactly the same as in single-switch case!)

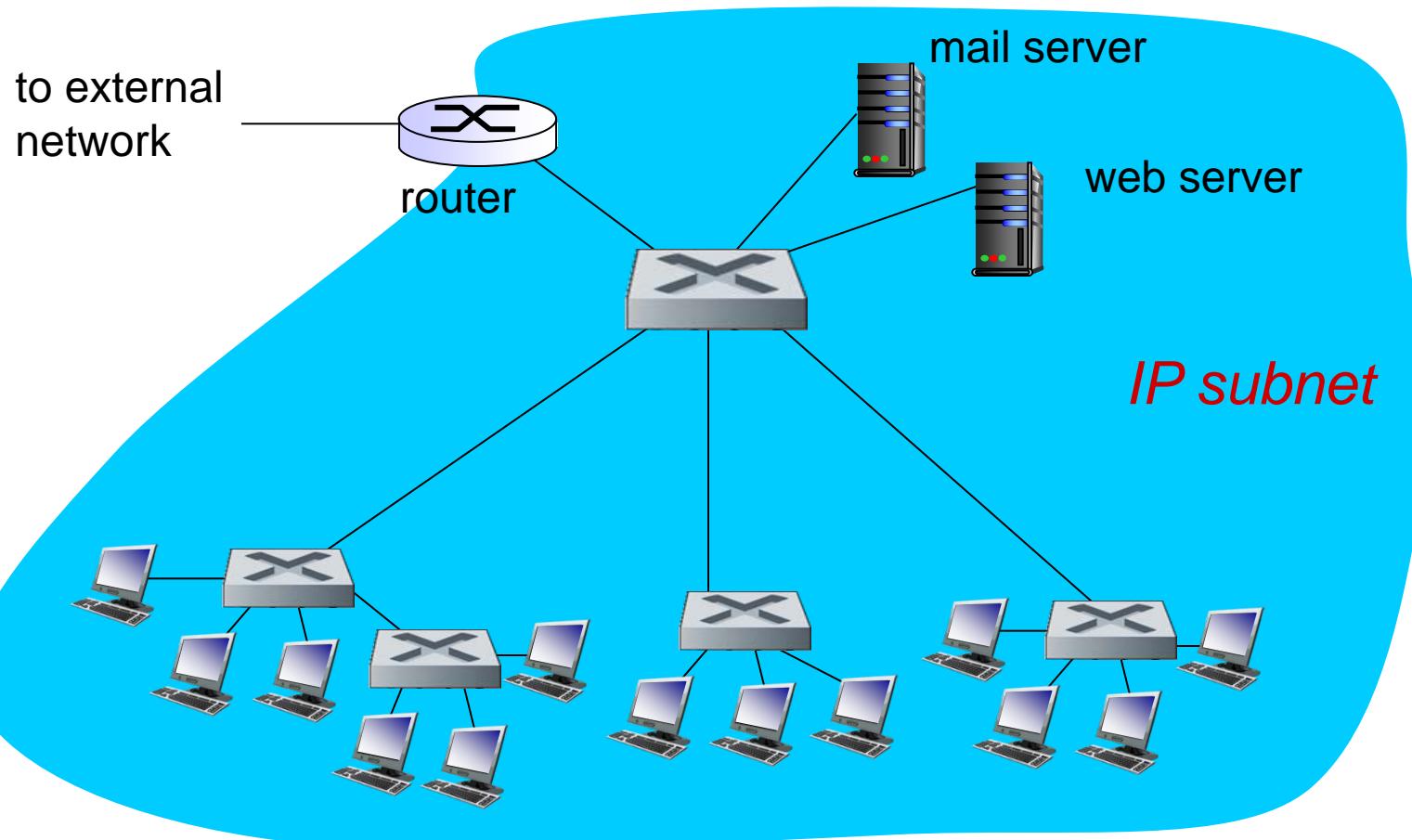
Self-learning multi-switch example

Suppose C sends frame to I, I responds to C



- ❖ Q: show switch tables and packet forwarding in S_1, S_2, S_3, S_4

Institutional network



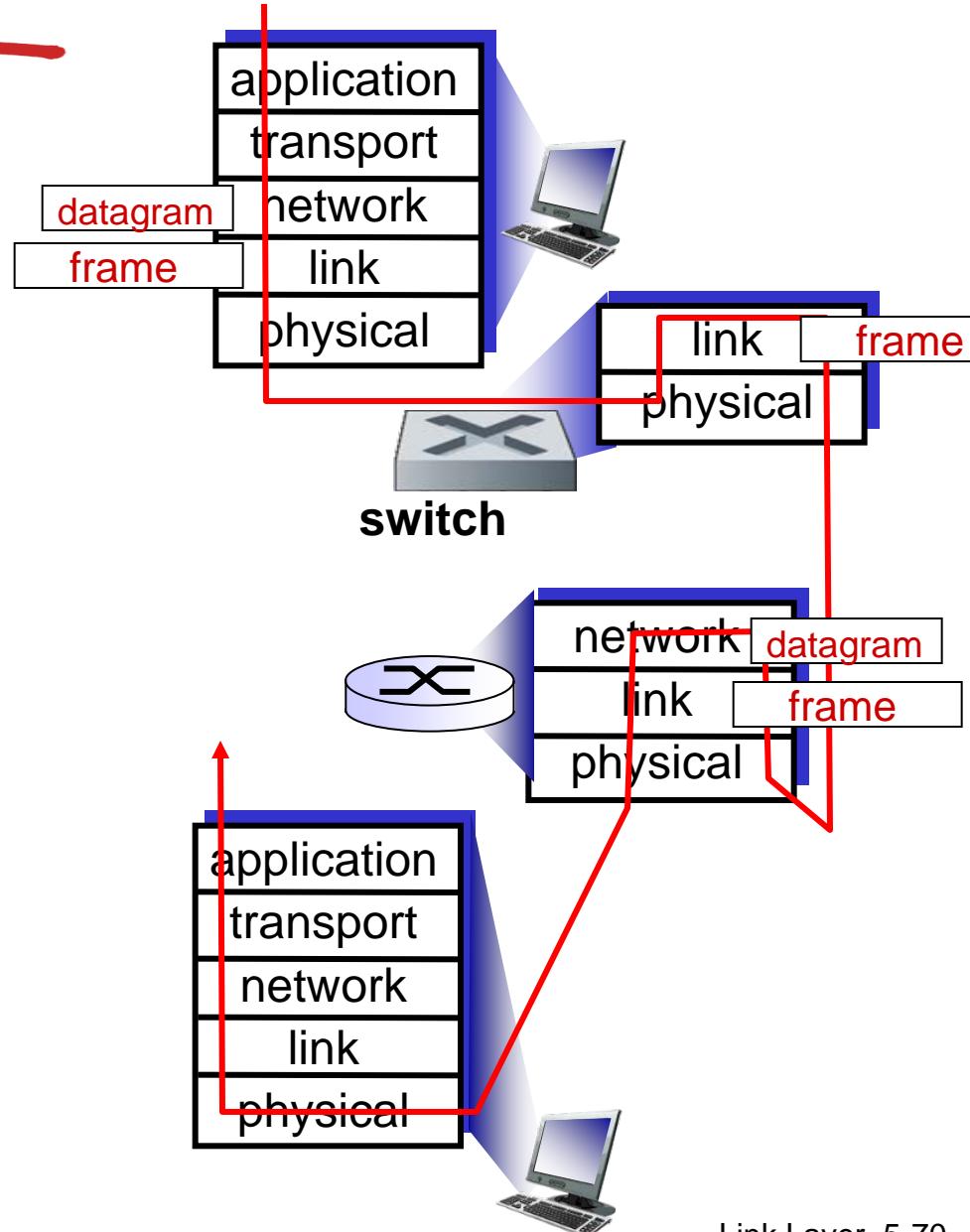
Switches vs. routers

both are store-and-forward:

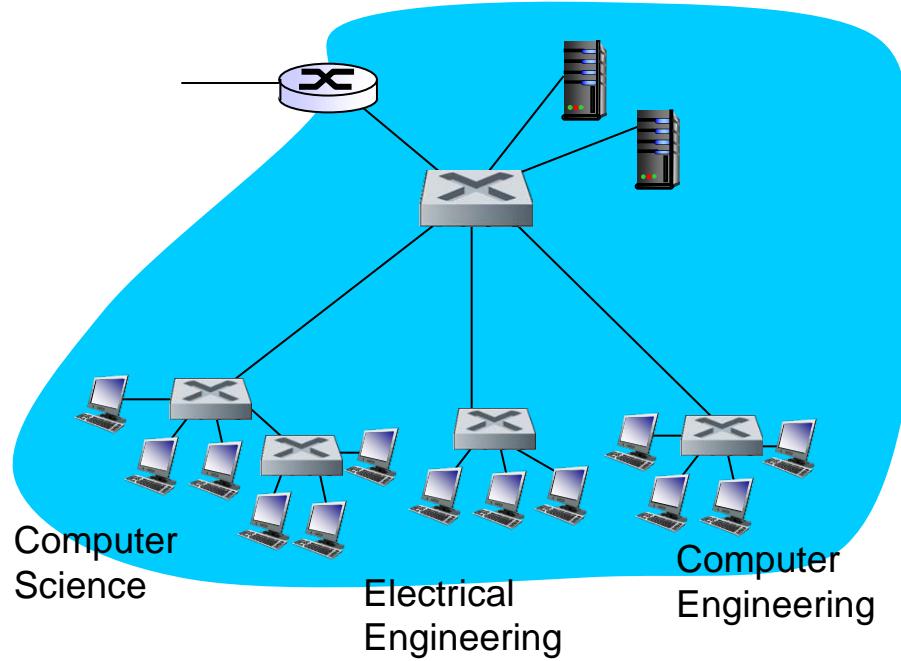
- **routers**: network-layer devices (examine network-layer headers)
- **switches**: link-layer devices (examine link-layer headers)

both have forwarding tables:

- **routers**: compute tables using routing algorithms, IP addresses
- **switches**: learn forwarding table using flooding, learning, MAC addresses



VLANs: motivation



consider:

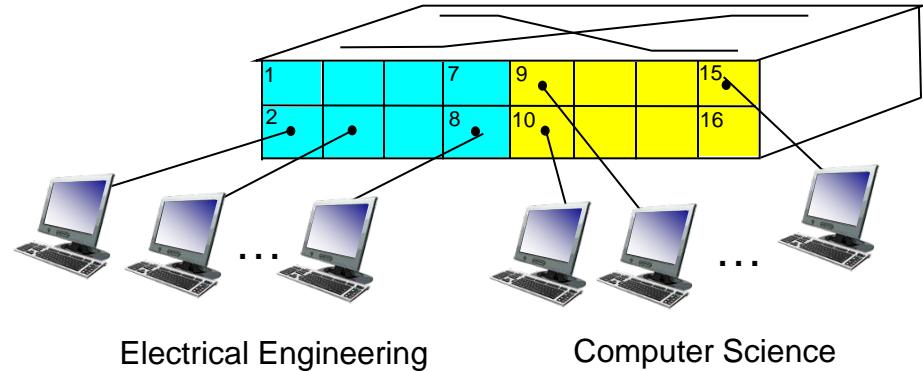
- ❖ CS user moves office to EE, but wants connect to CS switch?
- ❖ single broadcast domain:
 - all layer-2 broadcast traffic (**ARP**, **DHCP**, unknown location of destination MAC address) must cross entire LAN
 - security/privacy, efficiency issues

VLANs

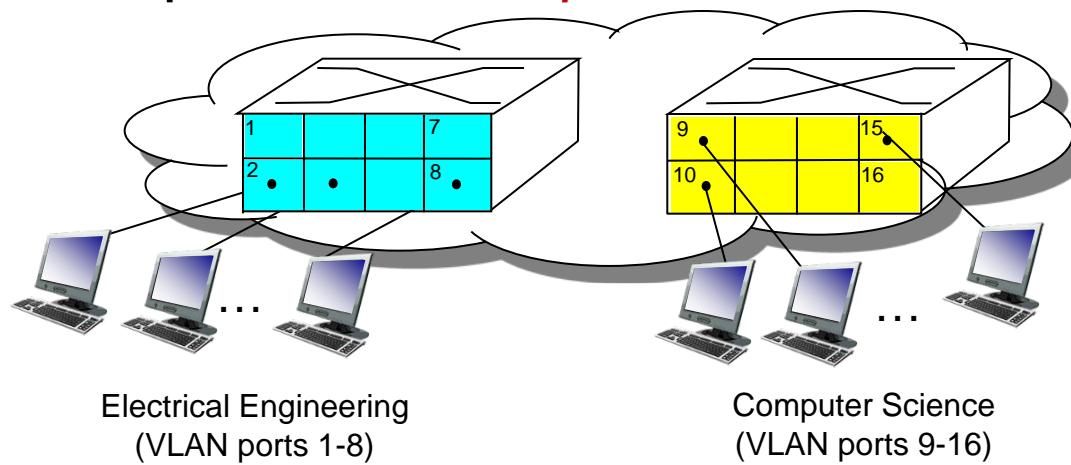
Virtual Local Area Network

switch(es) supporting VLAN capabilities can be configured to define multiple **virtual** LANS over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that ***single*** physical switch



... operates as ***multiple*** virtual switches



Port-based VLAN

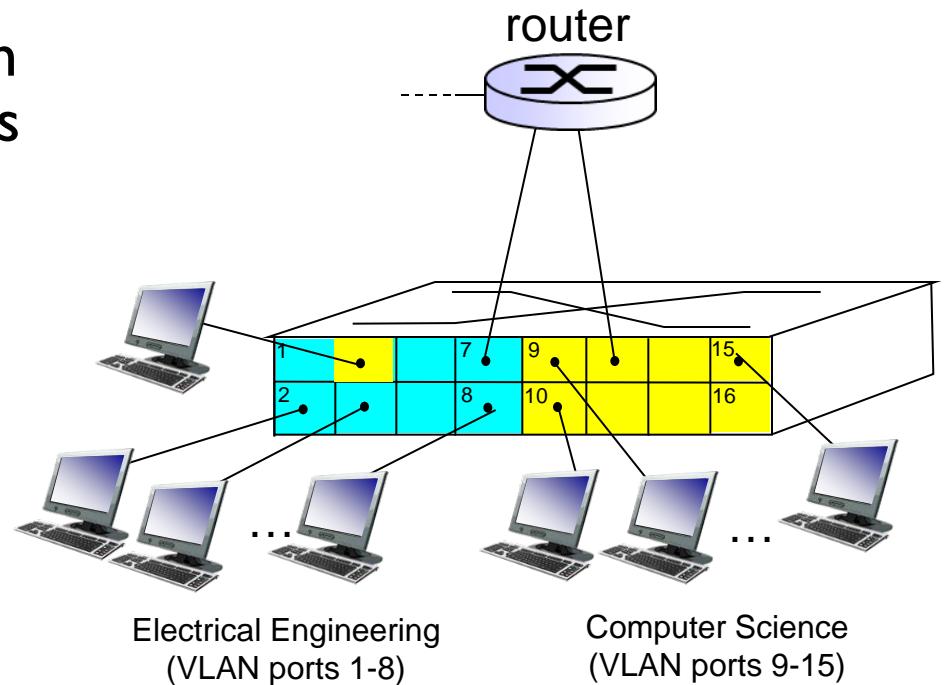
- ❖ ***traffic isolation:*** frames to/from ports 1-8 can *only* reach ports 1-8

- can also define VLAN based on MAC addresses of endpoints, rather than switch port

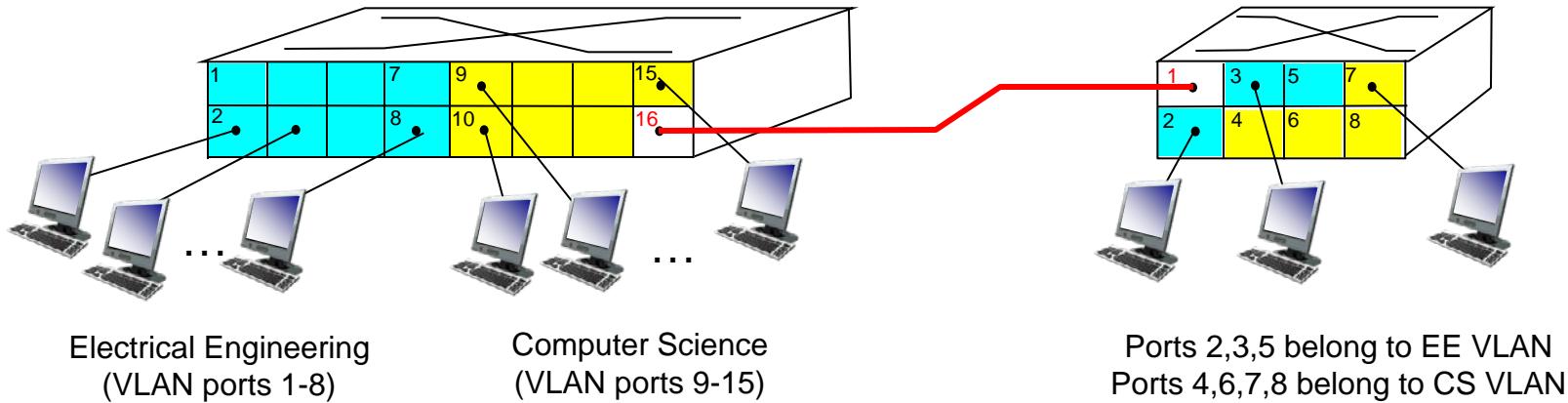
- ❖ ***dynamic membership:*** ports can be dynamically assigned among VLANs

- ❖ ***forwarding between VLANs:*** done via routing (just as with separate switches)

- in practice vendors sell combined switches plus routers

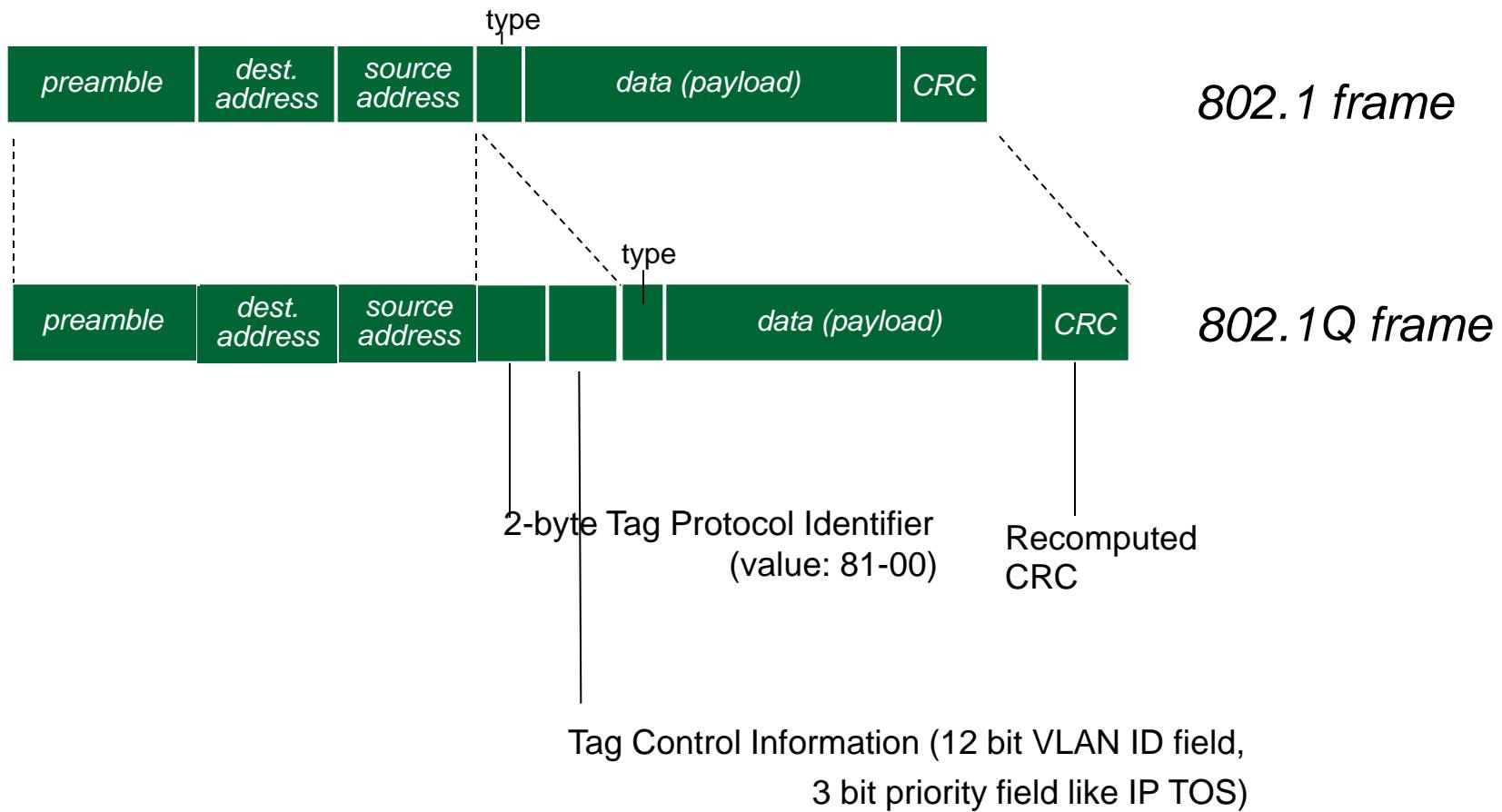


VLANS spanning multiple switches



- ❖ ***trunk port:*** carries frames between VLANS defined over multiple physical switches
 - frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
 - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

802.1Q VLAN frame format



Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

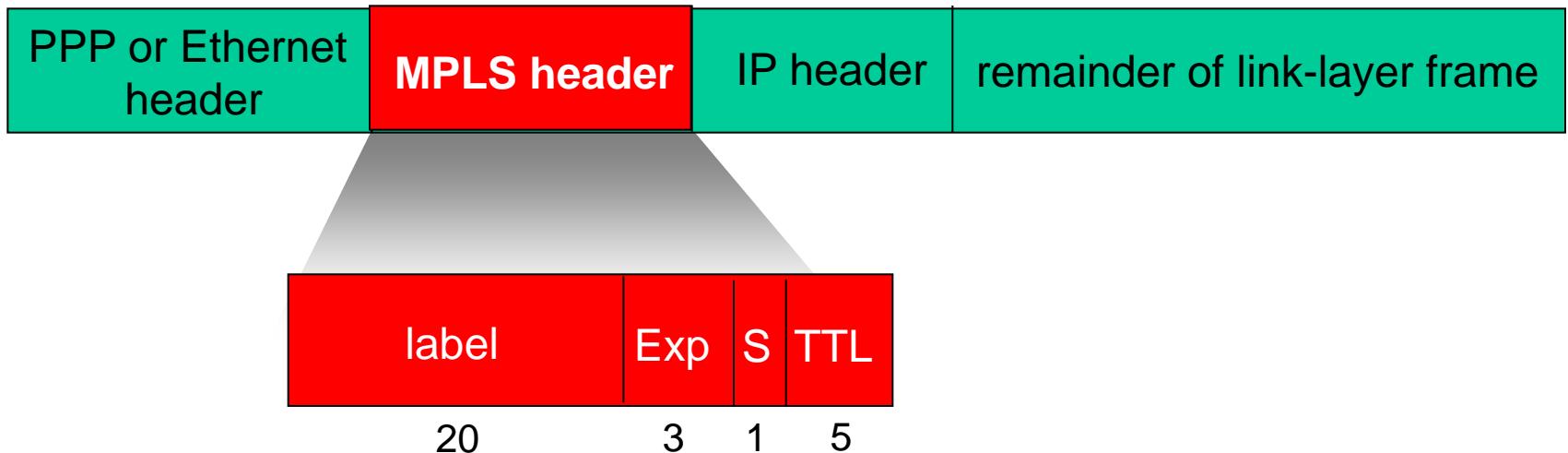
5.5 link virtualization:
MPLS

5.6 data center
networking

5.7 a day in the life of a
web request

Multiprotocol label switching (MPLS)

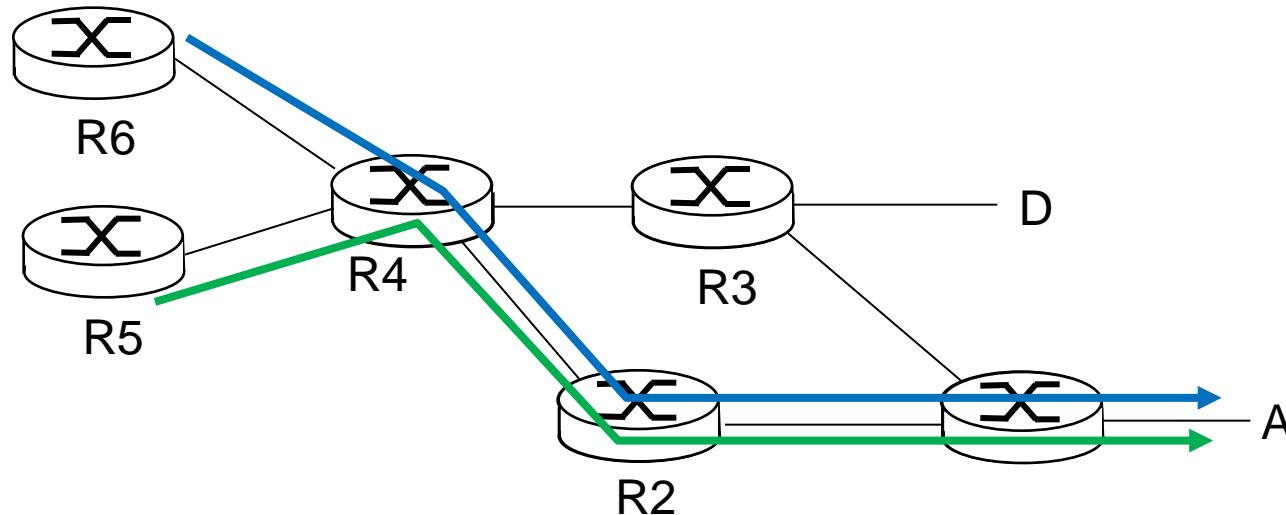
- ❖ initial goal: high-speed IP forwarding using fixed length label (instead of IP address)
 - fast lookup using fixed length identifier (rather than shortest prefix matching)
 - borrowing ideas from Virtual Circuit (VC) approach
 - but IP datagram still keeps IP address!



MPLS capable routers

- ❖ a.k.a. label-switched router
- ❖ forward packets to outgoing interface based only on label value (*don't inspect IP address*)
 - MPLS forwarding table distinct from IP forwarding tables
- ❖ **flexibility:** MPLS forwarding decisions can differ from those of IP
 - use destination *and* source addresses to route flows to same destination differently (traffic engineering)
 - re-route flows quickly if link fails: pre-computed backup paths (useful for VoIP)

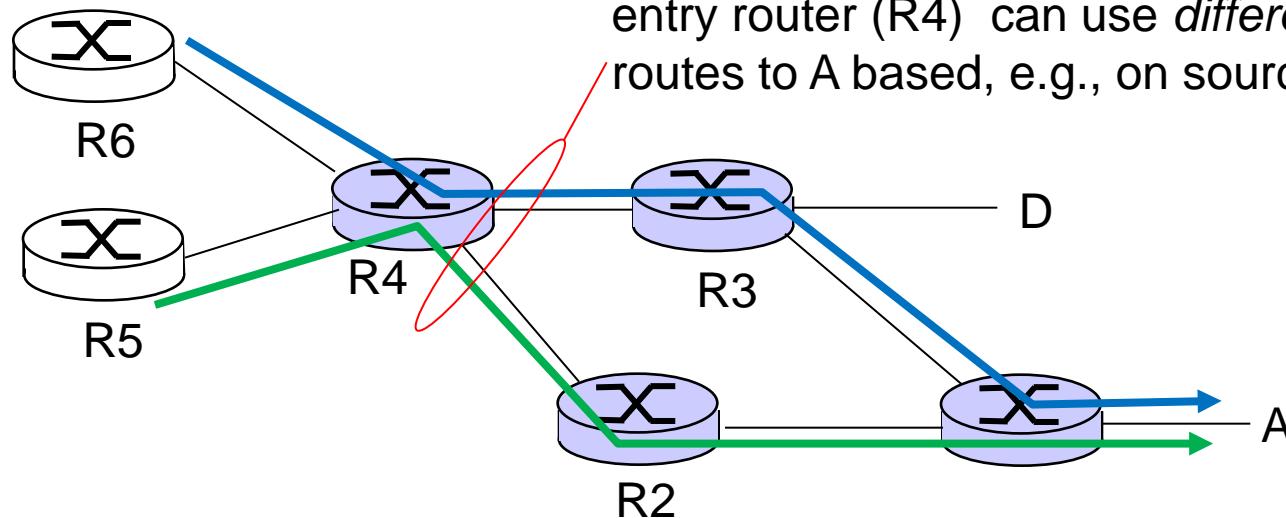
MPLS versus IP paths



- ❖ *IP routing*: path to destination determined by destination address alone



MPLS versus IP paths



- ❖ **IP routing:** path to destination determined by destination address alone

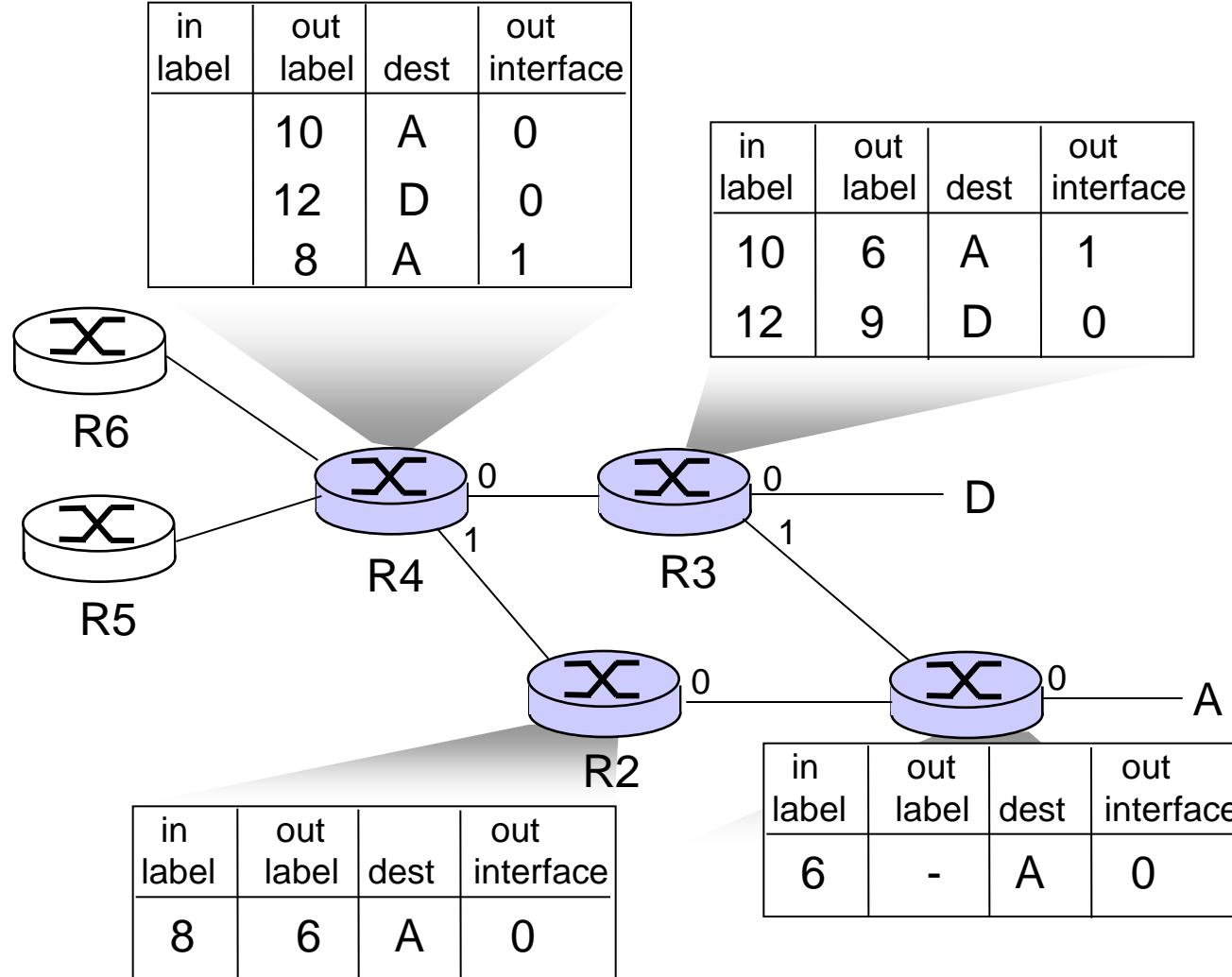


- ❖ **MPLS routing:** path to destination can be based on source *and* dest. address



- **fast reroute:** precompute backup routes in case of link failure

MPLS forwarding tables



Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

5.5 link virtualization:
MPLS

5.6 data center
networking

5.7 a day in the life of a
web request

Data center networks

- ❖ 10's to 100's of thousands of hosts, often closely coupled, in close proximity:
 - e-business (e.g. Amazon)
 - content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
 - search engines, data mining (e.g., Google)
- ❖ challenges:
 - multiple applications, each serving massive numbers of clients
 - managing/balancing load, avoiding processing, networking, data bottlenecks

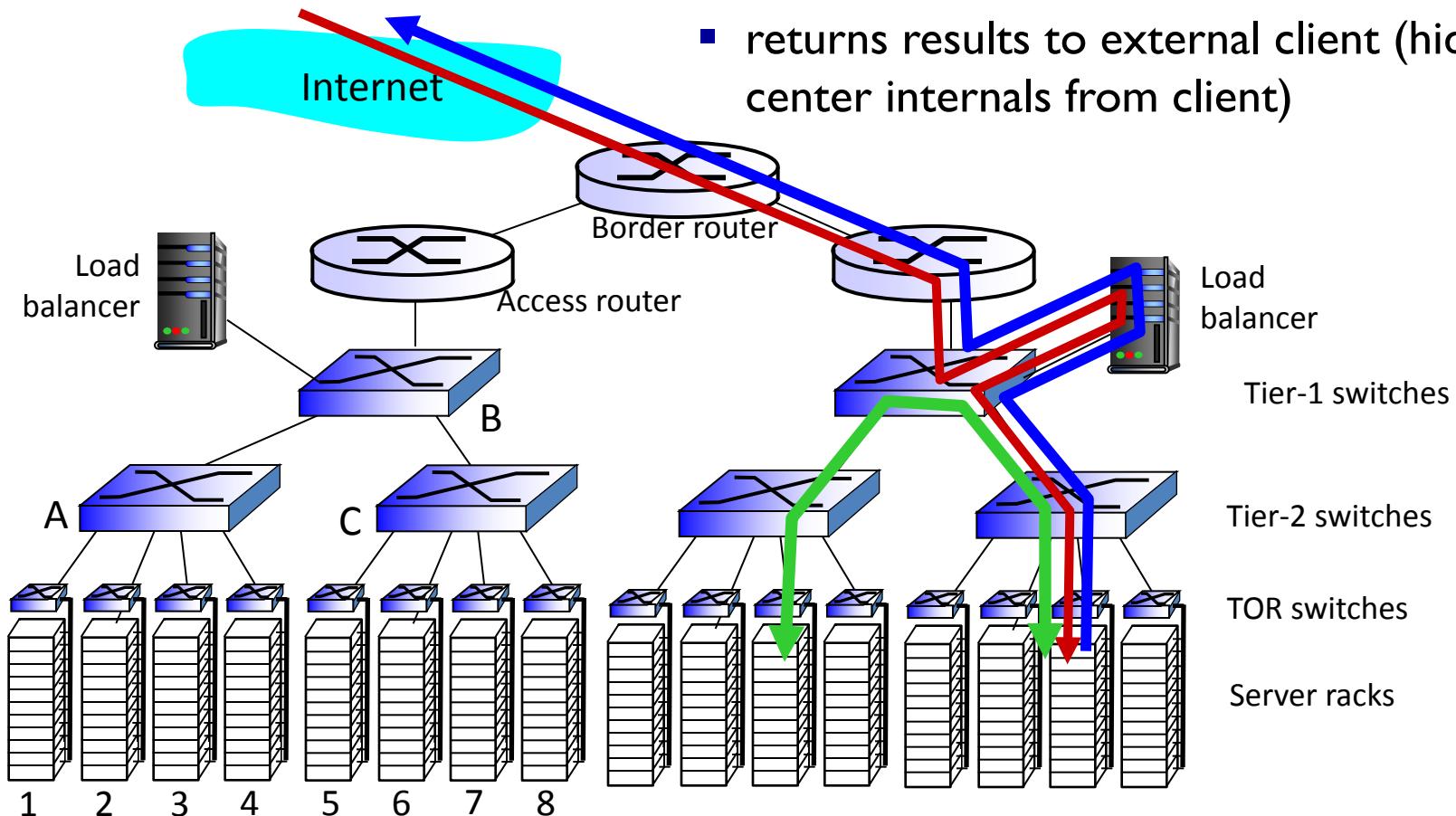


Inside a 40-ft Microsoft container,
Chicago data center

Data center networks

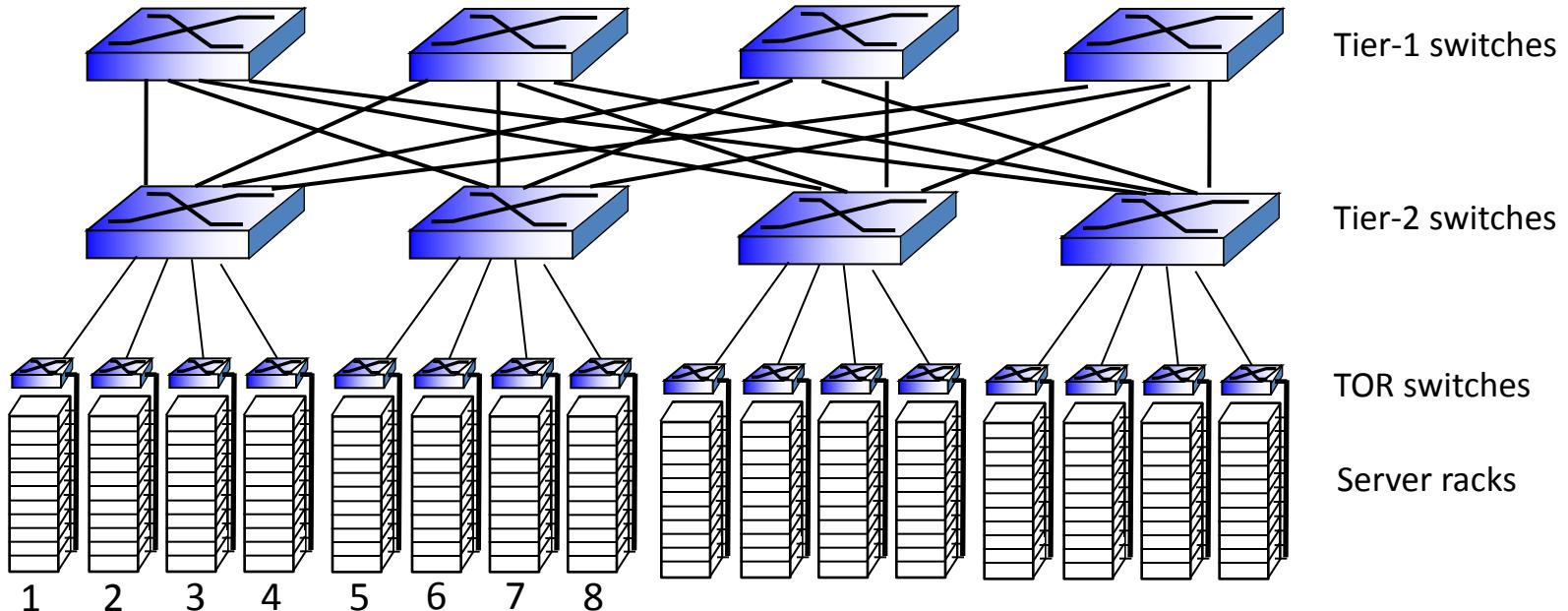
load balancer: application-layer routing

- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)



Data center networks

- ❖ rich interconnection among switches, racks:
 - increased throughput between racks (multiple routing paths possible)
 - increased reliability via redundancy



Link layer, LANs: outline

5.1 introduction, services

5.2 error detection,
correction

5.3 multiple access
protocols

5.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

5.5 link virtualization:
MPLS

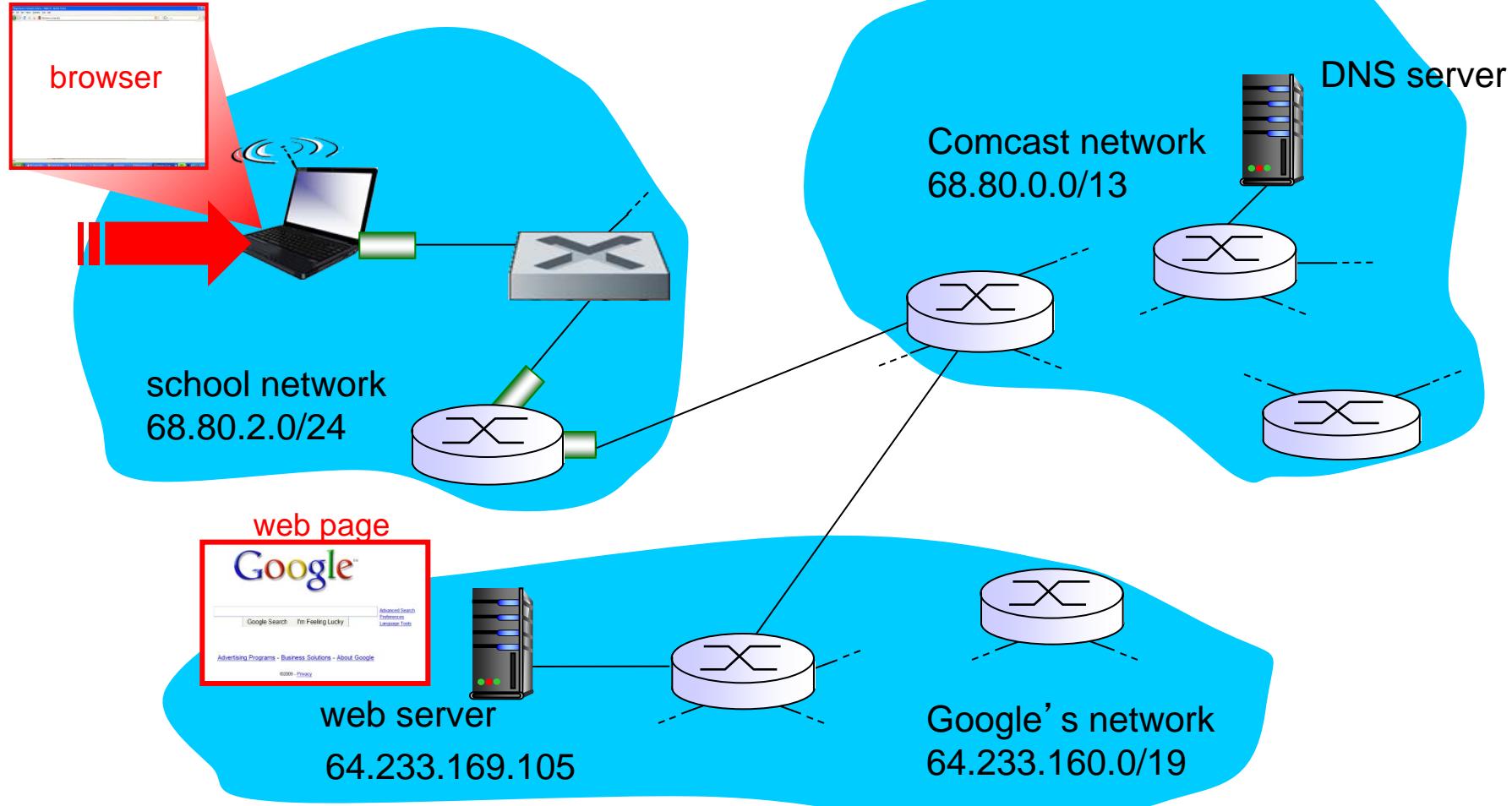
5.6 data center
networking

5.7 a day in the life of a
web request

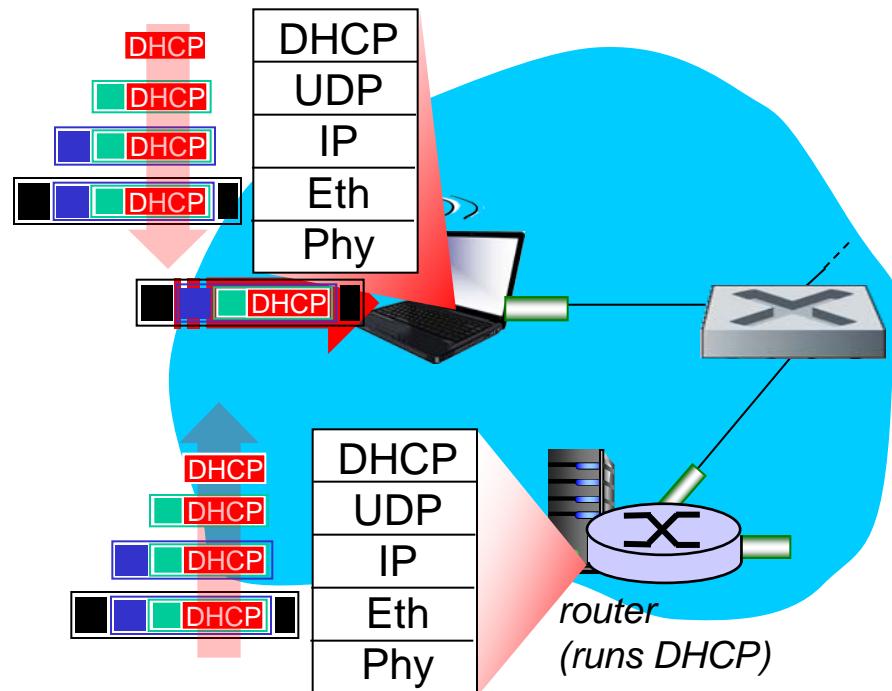
Synthesis: a day in the life of a web request

- ❖ 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

A day in the life: scenario

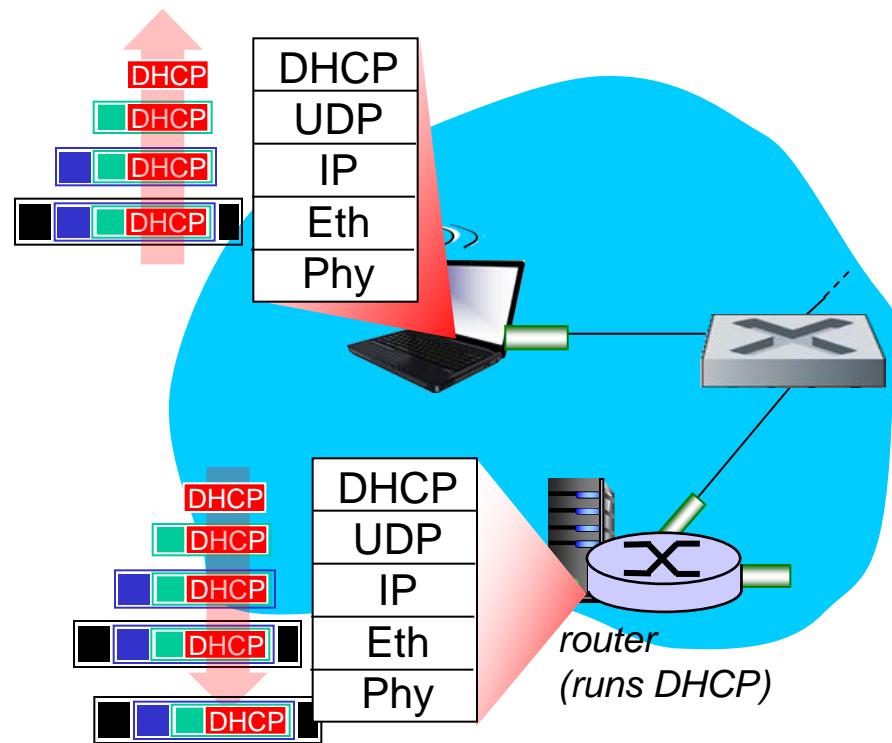


A day in the life... connecting to the Internet



- ❖ connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use **DHCP**
- ❖ DHCP request *encapsulated* in **UDP**, encapsulated in **IP**, encapsulated in **802.3 Ethernet**
- ❖ Ethernet frame *broadcast* (dest: FFFFFFFFFFFF) on LAN, received at router running **DHCP** server
- ❖ Ethernet *demuxed* to IP demuxed, UDP demuxed to DHCP

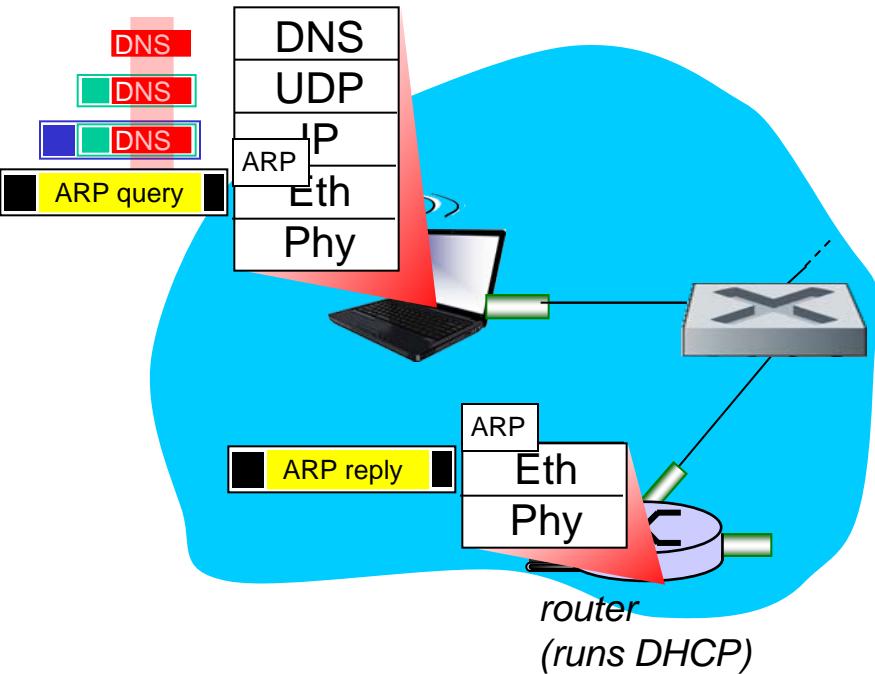
A day in the life... connecting to the Internet



- ❖ DHCP server formulates **DHCP ACK** containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- ❖ encapsulation at DHCP server, frame forwarded (*switch learning*) through LAN, demultiplexing at client
- ❖ DHCP client receives DHCP ACK reply

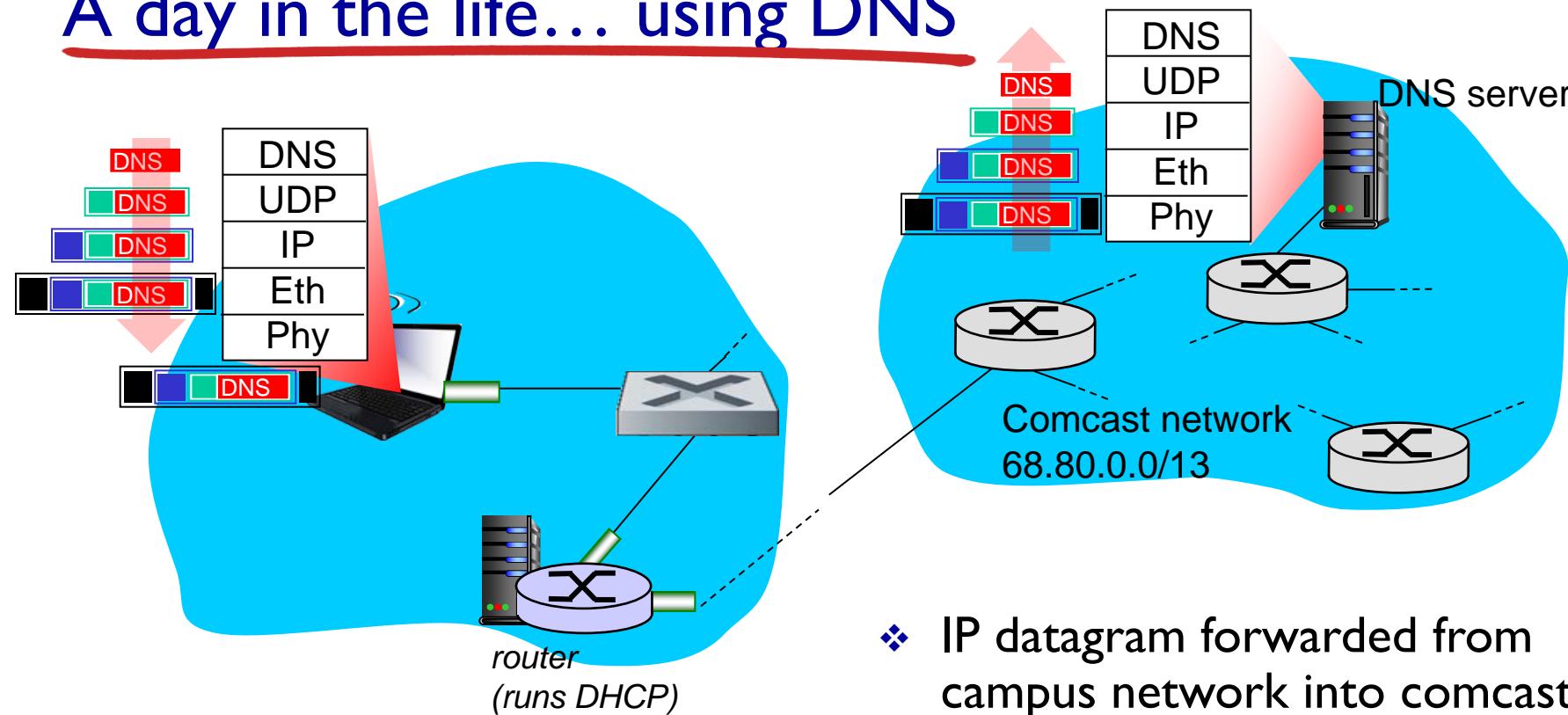
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life... ARP (before DNS, before HTTP)



- ❖ before sending **HTTP** request, need IP address of www.google.com: **DNS**
- ❖ DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: **ARP**
- ❖ **ARP query** broadcast, received by router, which replies with **ARP reply** giving MAC address of router interface
- ❖ client now knows MAC address of first hop router, so can now send frame containing DNS query

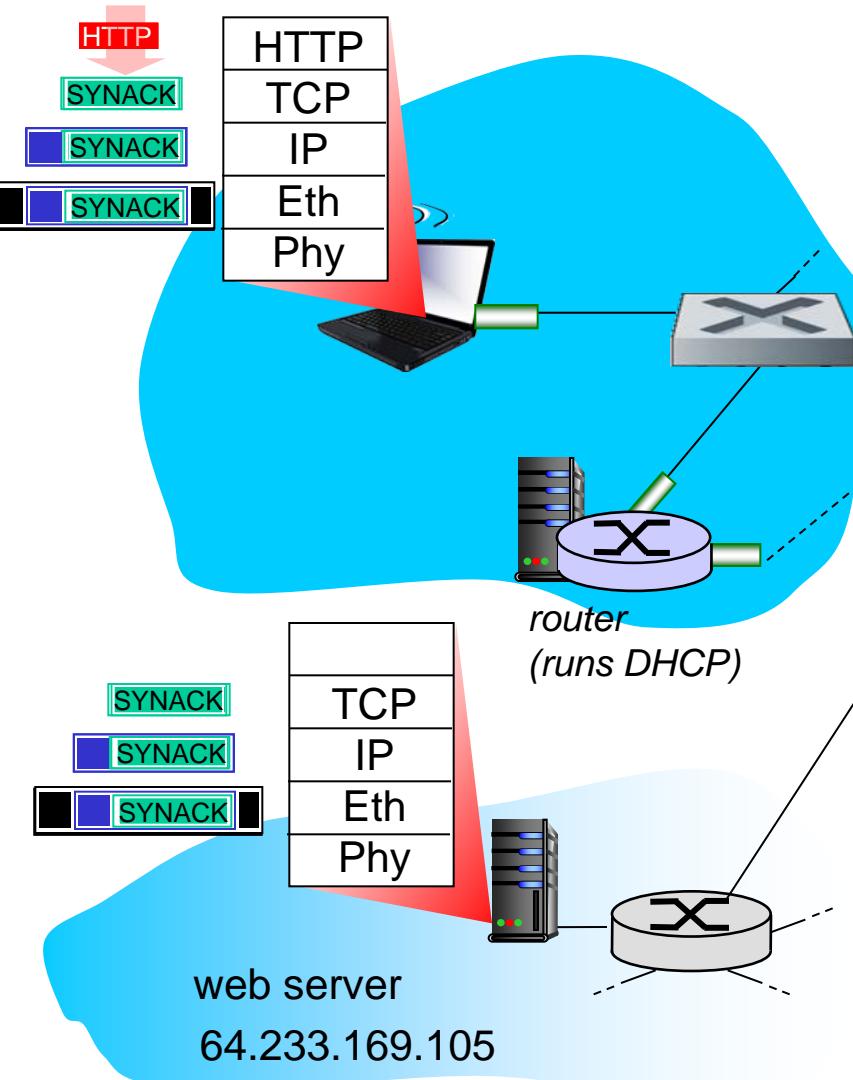
A day in the life... using DNS



- ❖ IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

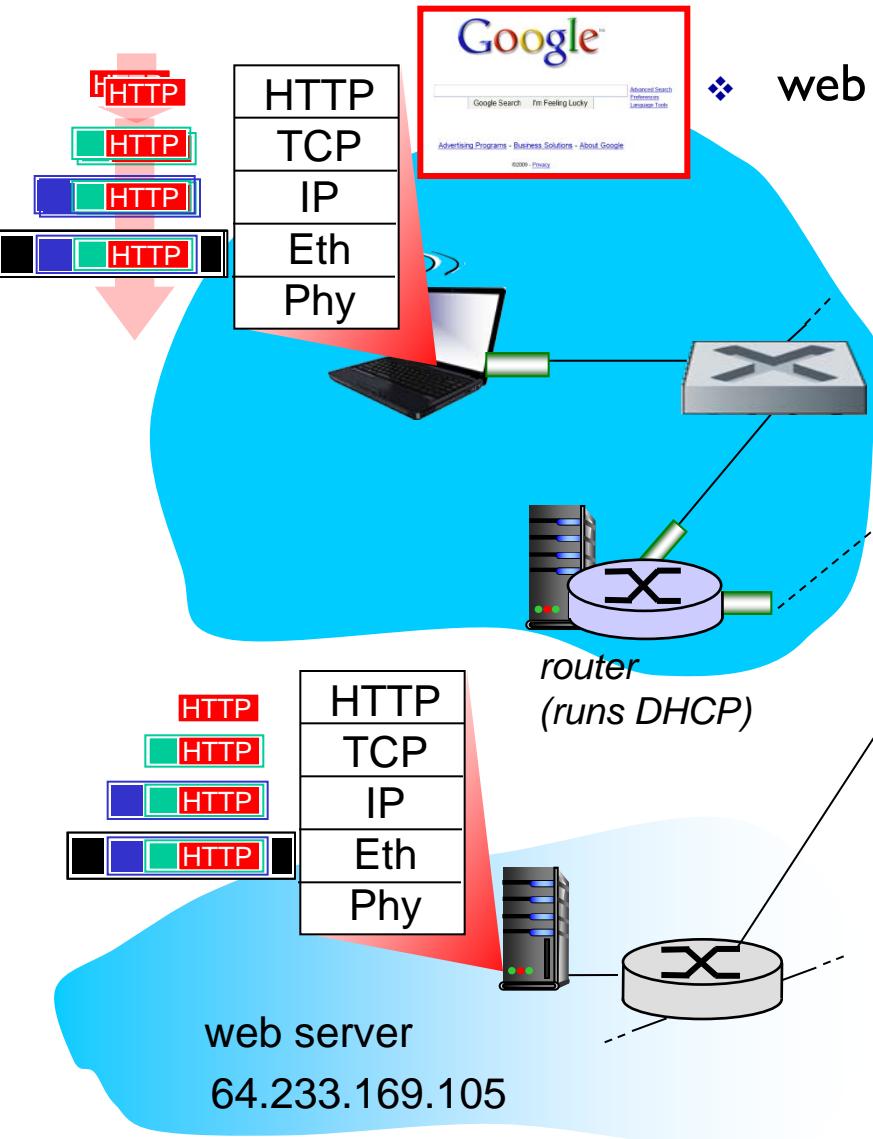
- ❖ IP datagram forwarded from campus network into comcast network, routed (tables created by *RIP, OSPF, IS-IS* and/or *BGP* routing protocols) to DNS server
- ❖ demux' ed to DNS server
- ❖ DNS server replies to client with IP address of www.google.com

A day in the life...TCP connection carrying HTTP



- ❖ to send HTTP request, client first opens **TCP socket** to web server
- ❖ TCP **SYN segment** (step 1 in 3-way handshake) *inter-domain routed* to web server
- ❖ web server responds with **TCP SYNACK** (step 2 in 3-way handshake)
- ❖ TCP **connection established!**

A day in the life... HTTP request/reply



❖ web page *finally* (!!!) displayed

- ❖ *HTTP request* sent into TCP socket
- ❖ IP datagram containing HTTP request routed to www.google.com
- ❖ web server responds with *HTTP reply* (containing web page)
- ❖ IP datagram containing HTTP reply routed back to client

Chapter 5: Summary

- ❖ principles behind data link layer services:
 - error detection, correction
 - sharing a broadcast channel: multiple access
 - link layer addressing
- ❖ instantiation and implementation of various link layer technologies
 - Ethernet
 - switched LANS, VLANs
 - virtualized networks as a link layer: MPLS
- ❖ synthesis: a day in the life of a web request

Chapter 5: let's take a breath

- ❖ journey down protocol stack *complete* (except PHY)
- ❖ solid understanding of networking principles, practice
- ❖ could stop here but *lots* of interesting topics!
 - wireless
 - multimedia
 - security
 - network management

Chapter 6

Network Security

© *The materials in these notes are adapted from Computer Networking: A Top Down Approach, 8th edition, by Jim Kurose, Keith Ross*

Security: overview

Chapter goals:

- understand principles of network security:
 - cryptography and its *many* uses beyond “confidentiality”
 - authentication
 - message integrity
- security in practice:
 - firewalls and intrusion detection systems
 - security in application, transport, network, link layers

Chapter 6 outline

- What is network security?
- Principles of cryptography
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Operational security: firewalls and IDS



What is network security?

confidentiality: only sender, intended receiver should “understand” message contents

- sender encrypts message
- receiver decrypts message

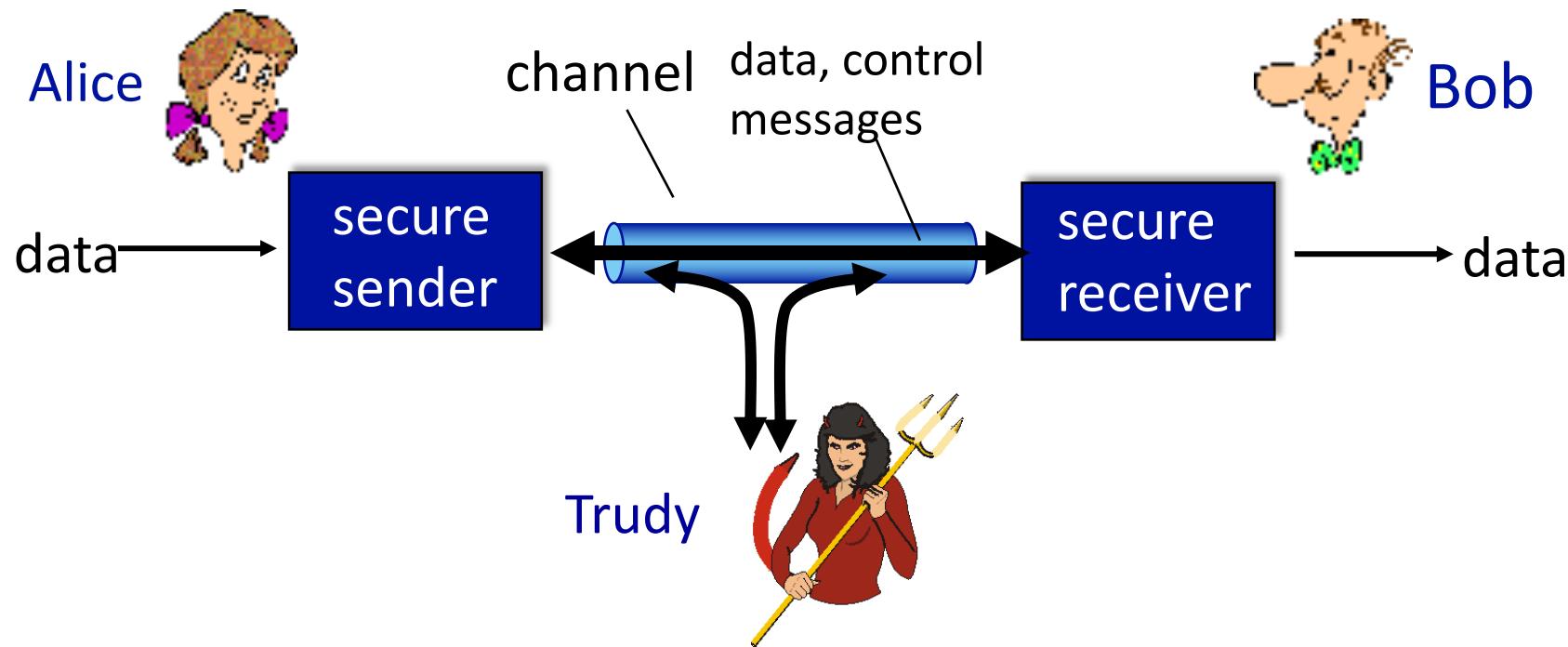
authentication: sender, receiver want to confirm identity of each other

message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

access and availability: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Friends and enemies: Alice, Bob, Trudy

Who might Bob and Alice be?

- ... well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- BGP routers exchanging routing table updates
- other examples?

There are bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: A lot! (recall section 1.6)

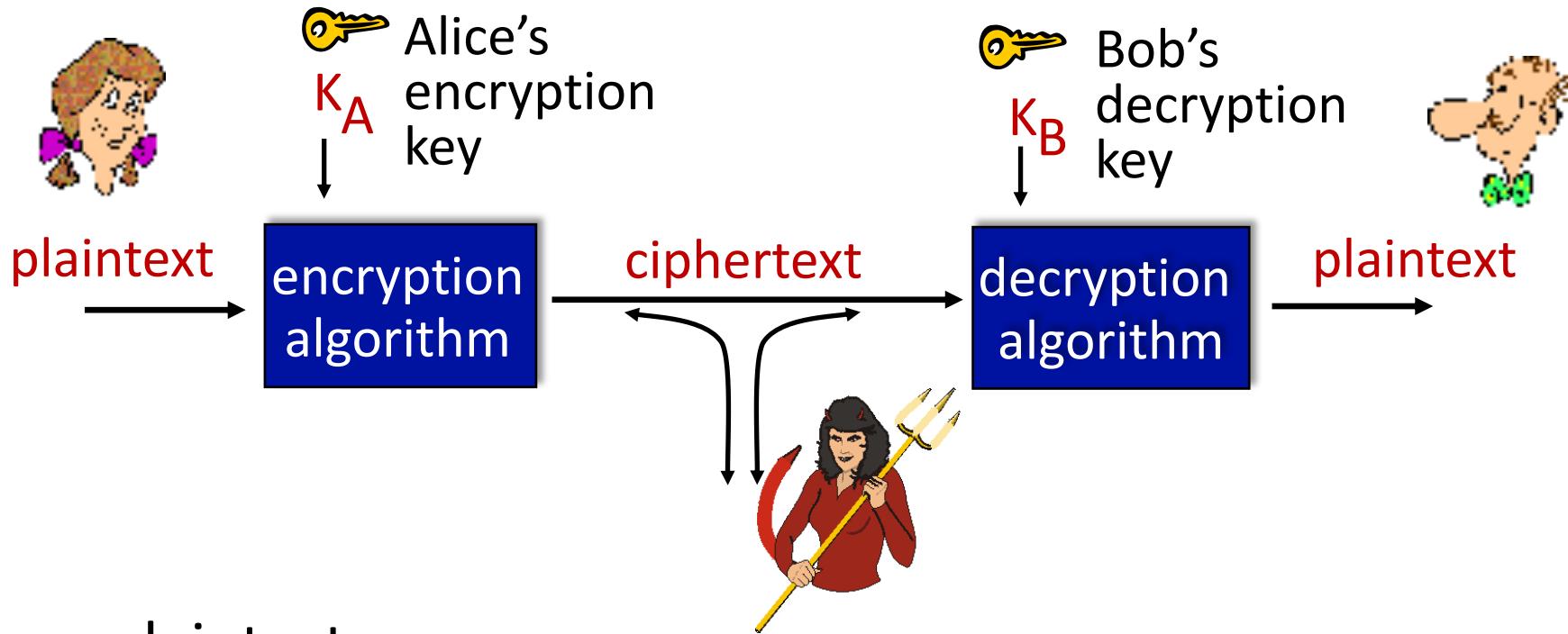
- **eavesdrop**: intercept messages
- actively **insert** messages into connection
- **impersonation**: can fake (spoof) source address in packet (or any field in packet)
- **hijacking**: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service**: prevent service from being used by others (e.g., by overloading resources)

Chapter 6 outline

- What is network security?
- **Principles of cryptography**
- Message integrity, authentication
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Operational security: firewalls and IDS



The language of cryptography

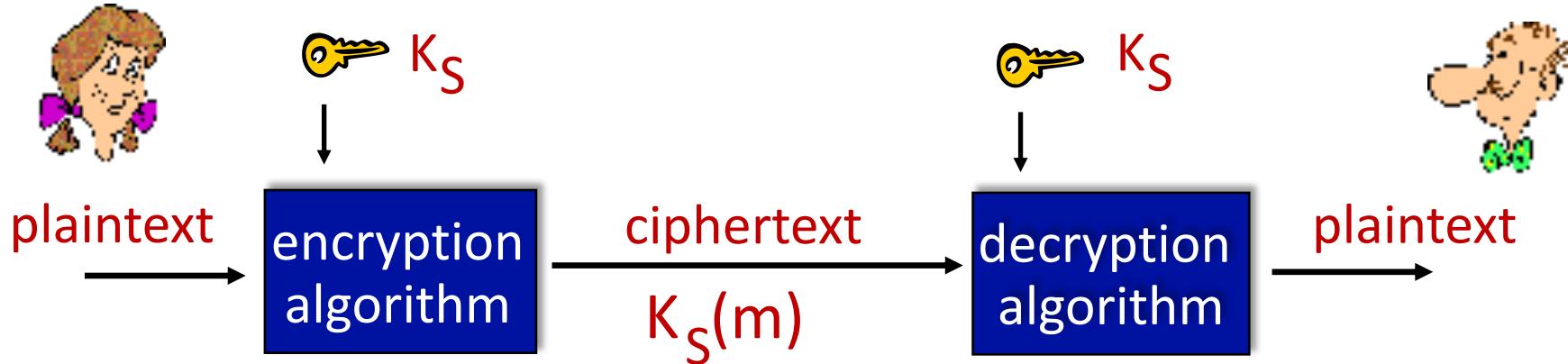


m : plaintext message

$K_A(m)$: ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

Simple encryption scheme

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

plaintext: abcdefghijklmnopqrstuvwxyz

ciphertext: mnbvctxzasdfghjklpoiuytrewq

e.g.: Plaintext: bob. i love you. alice

ciphertext: nkn. s gktc wky. mgsbc



Encryption key: mapping from set of 26 letters
to set of 26 letters

Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
 - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
 - no known good analytic attack
- making DES more secure:
 - 3DES: encrypt 3 times with 3 different keys

AES: Advanced Encryption Standard

- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

Does symmetric encryption suffice?

- Symmetric encryption is usually fast, but it has limitations!
 - The key is known by **both** sender and receiver
 - Key distribution/agreement is potentially **vulnerable**
 - Key is not attached with identity, thus against **non-repudiation**
 - Key management problem
 - Ex. when Alice wants to securely communicate with 100 others, Alice needs to remember 100 Keys, which is **hard to manage**



Public Key Cryptography

symmetric key crypto:

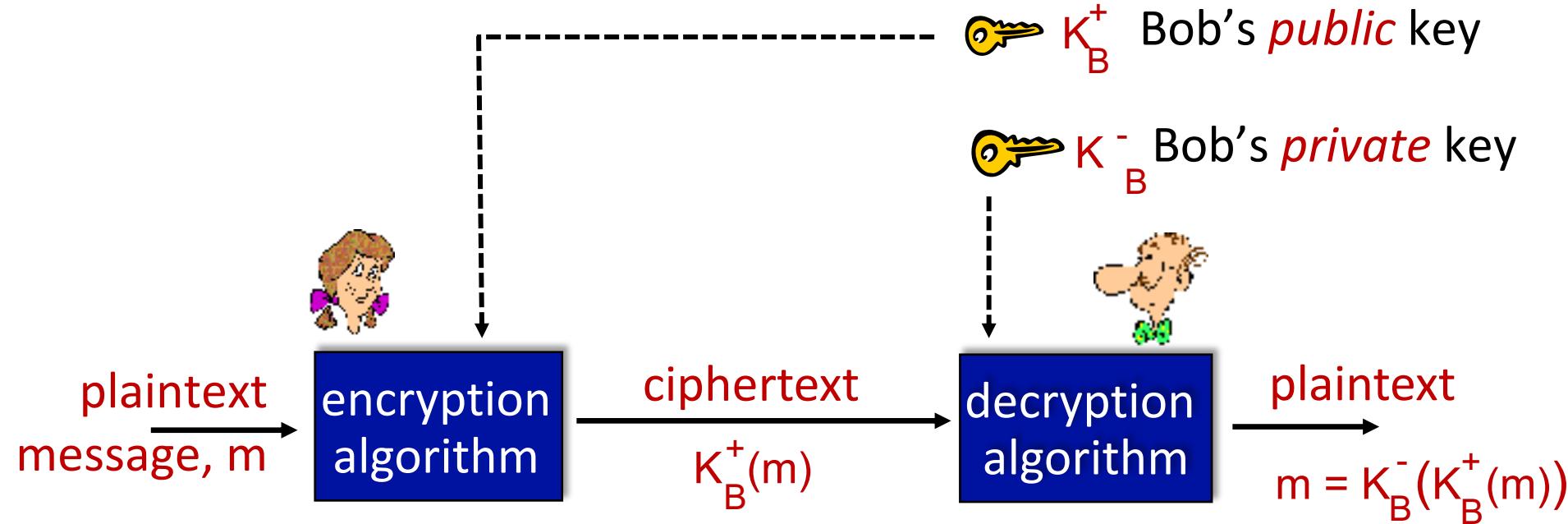
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

public key crypto

- *radically* different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver



Public Key Cryptography



Wow - public key cryptography revolutionized 2000-year-old (previously only symmetric key) cryptography!

- similar ideas emerged at roughly same time, independently in US and UK (classified)

Public key encryption algorithms

requirements:

- ① need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

- ② given public key K_B^+ , it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

RSA Algorithm-Key Generation

- Pick two large primes **p** and **q**
 - 1. $p=3, q=11$
 - 2. $n = 3*11 = 33$
 - 3. $\phi(n) = (2*10) = 20$
 - 4. $e = 7 \mid \text{GCD}(20,7) = 1$
 - 5. $d = 7^{-1} \bmod 20$
 - $d \mid d7 \bmod 20 = 1$
 - $d = 3$
 - “Euclid’s Algorithm”
 - Number theory
 - 6. k+ is {7, 33} and k- is {3, 33}
- Calculate **n** = **pq**
- Pick **e** such that it is relatively prime to **phi(n)** = $(q-1)(p-1)$
 - Euler’s Totient Function
- Calculate **d** = $e^{-1} \bmod \phi(n)$
 - i.e. $de \bmod \phi(n) = 1$
- Public key **k+** is **{e,n}** and private key **k-** is **{d,n}**

RSA Encryption with Public Key

- $E(k+, P): C = P^e \bmod n$
- $D(k-, C): P = C^d \bmod n$
- Example: Data: “4” (encoding of actual data)
 - $E(\{7,33\},4) = 4^7 \bmod 33 = 16384 \bmod 33 = 16$
 - $D(\{3,33\},16) = 16^3 \bmod 33 = 4096 \bmod 33 = 4$

magic happens!

RSA Encryption with Private Key

- $E(k+, P): C = P^d \bmod n$
- $D(k-, C): P = C^e \bmod n$
- Example: Data: “4” (encoding of actual data)
 - $E(\{3,33\},4) = 4^3 \bmod 33 = 64 \bmod 33 = 31$
 - $D(\{7,33\},31) = 31^7 \bmod 33 = 27,512,614,111 \bmod 33 = 4$

magic happens!

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

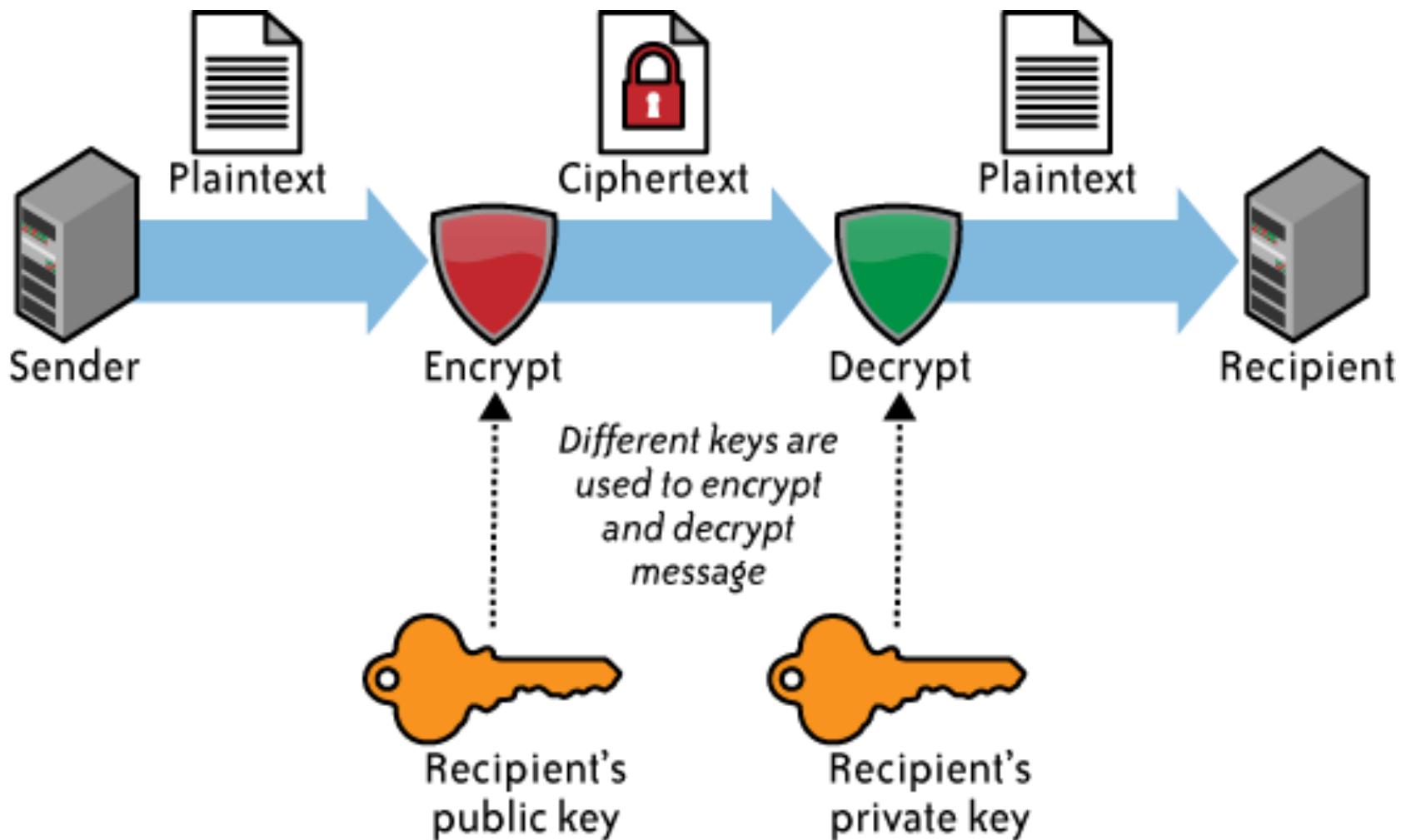
use public key
first, followed
by private key

use private key
first, followed
by public key

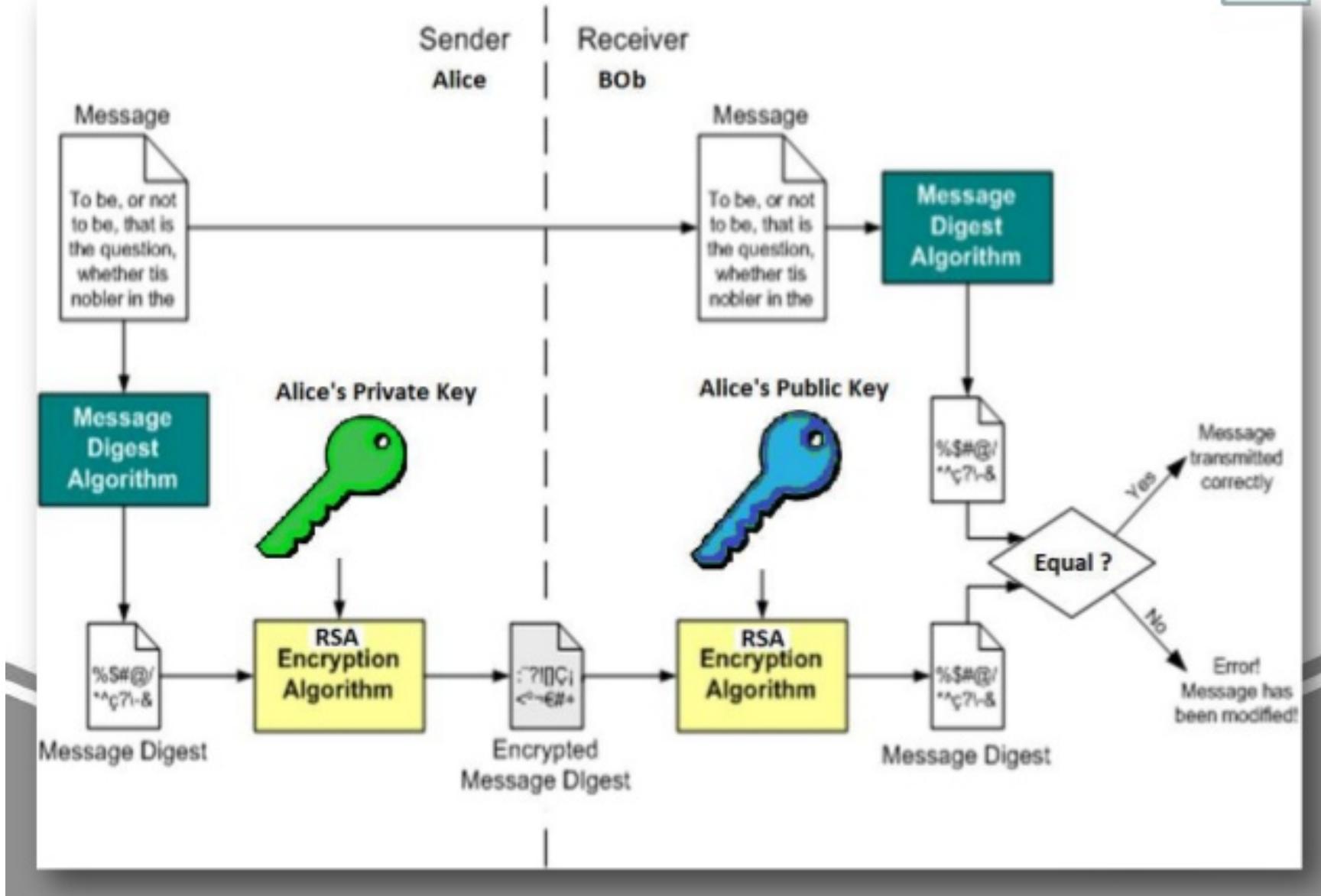
result is the same!

How to Use RSA

- Given message M, first uses recipient's public key against the message, then the paired private key will be used to restore the original message.
 - For **confidentiality**
- Given message M, the sender's private key is first used to sign the message, then the paired public key will be used by receiver to verify the signed message
 - This is the idea of **digital signatures**
 - For **content integrity**
 - For **authentication (source integrity)**
 - For **non-repudiation**



Digital Signature on RSA



Why is RSA secure?

- suppose you know Bob's public key (n, e). How hard is it to determine d ?
- essentially need to find factors of n without knowing the two factors p and q
 - fact: factoring a big number is hard

RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

session key, K_s

- Bob and Alice use RSA to exchange a symmetric session key K_s
- once both have K_s , they use symmetric key cryptography

Chapter 6 outline

- What is network security?
- Principles of cryptography
- **Authentication**, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Operational security: firewalls and IDS



Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”



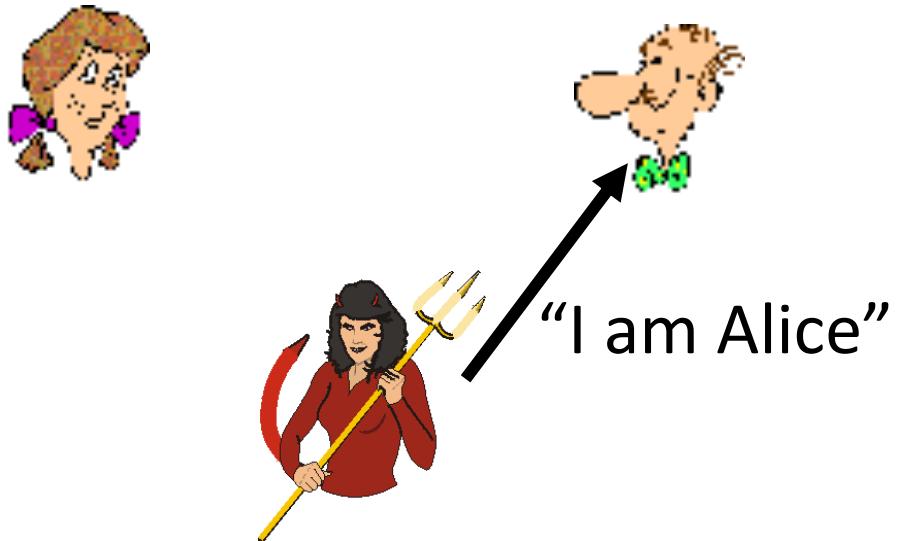
failure scenario??



Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”



in a network, Bob can not “see” Alice, so Trudy simply declares herself to be Alice

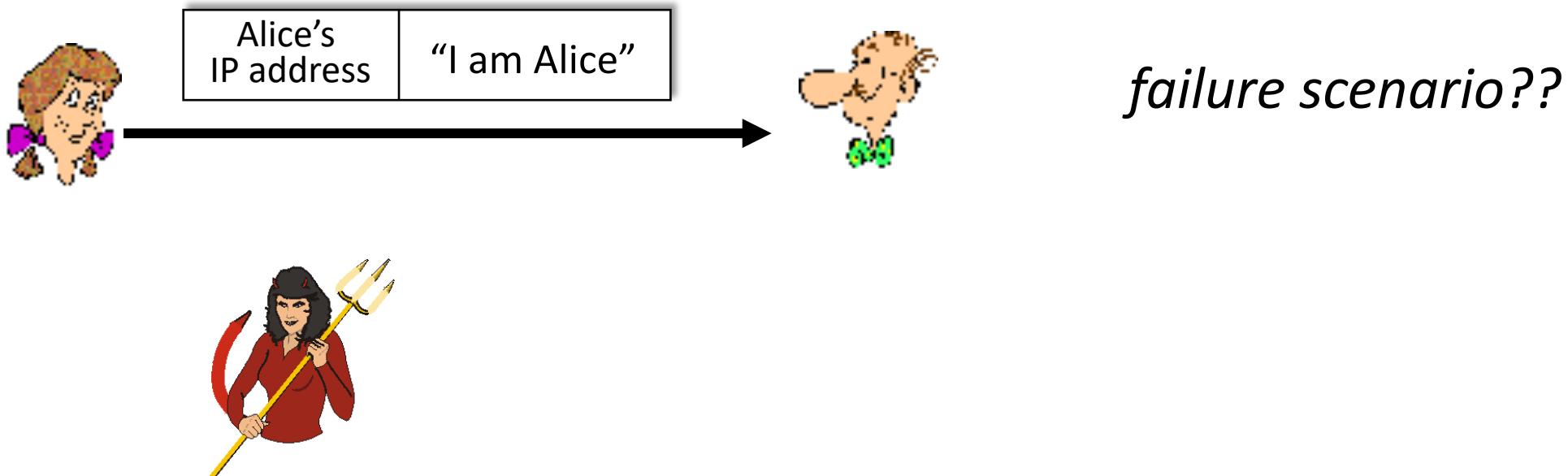


"On the Internet, nobody knows you're a dog."

Authentication: another try

Goal: Bob wants Alice to “prove” her identity to him

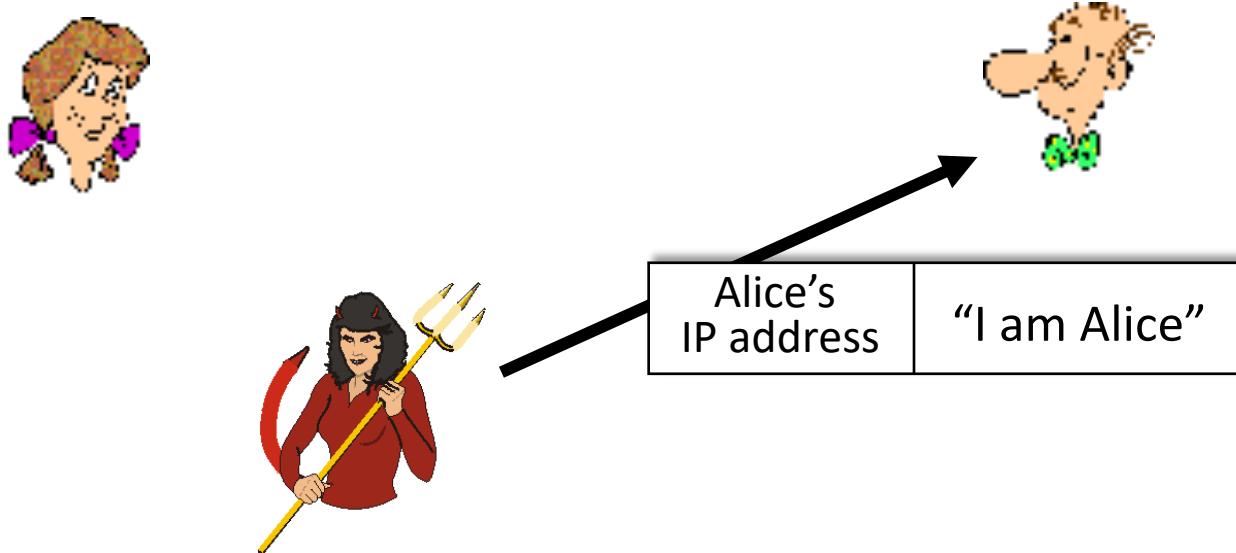
Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



Authentication: another try

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address

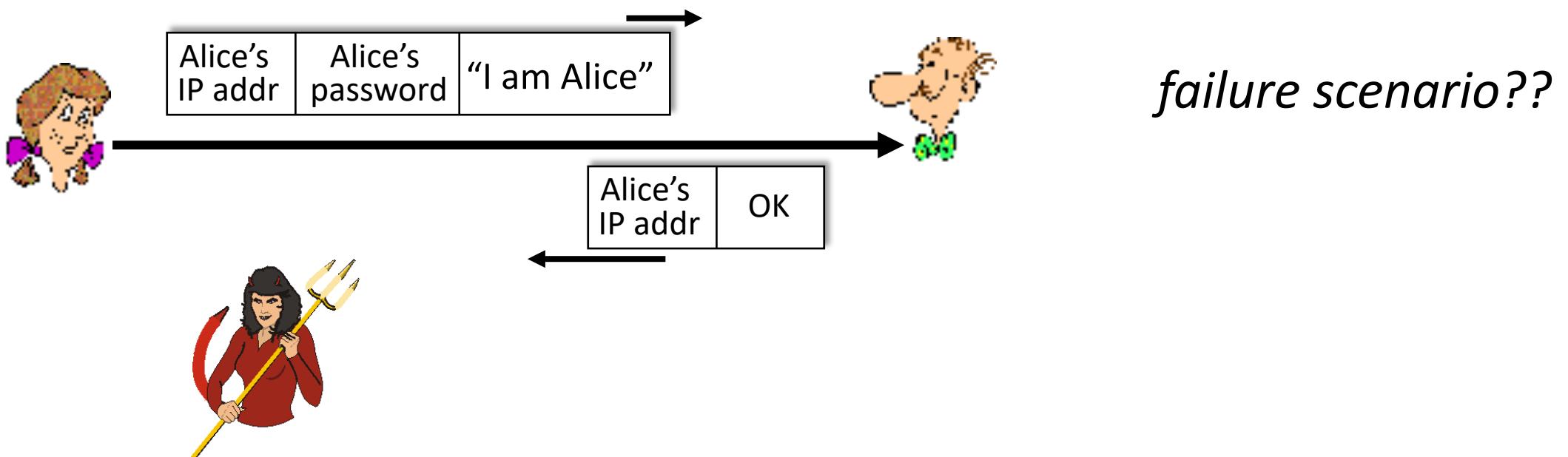


*Trudy can create
a packet “spoofing”
Alice’s address*

Authentication: a third try

Goal: Bob wants Alice to “prove” her identity to him

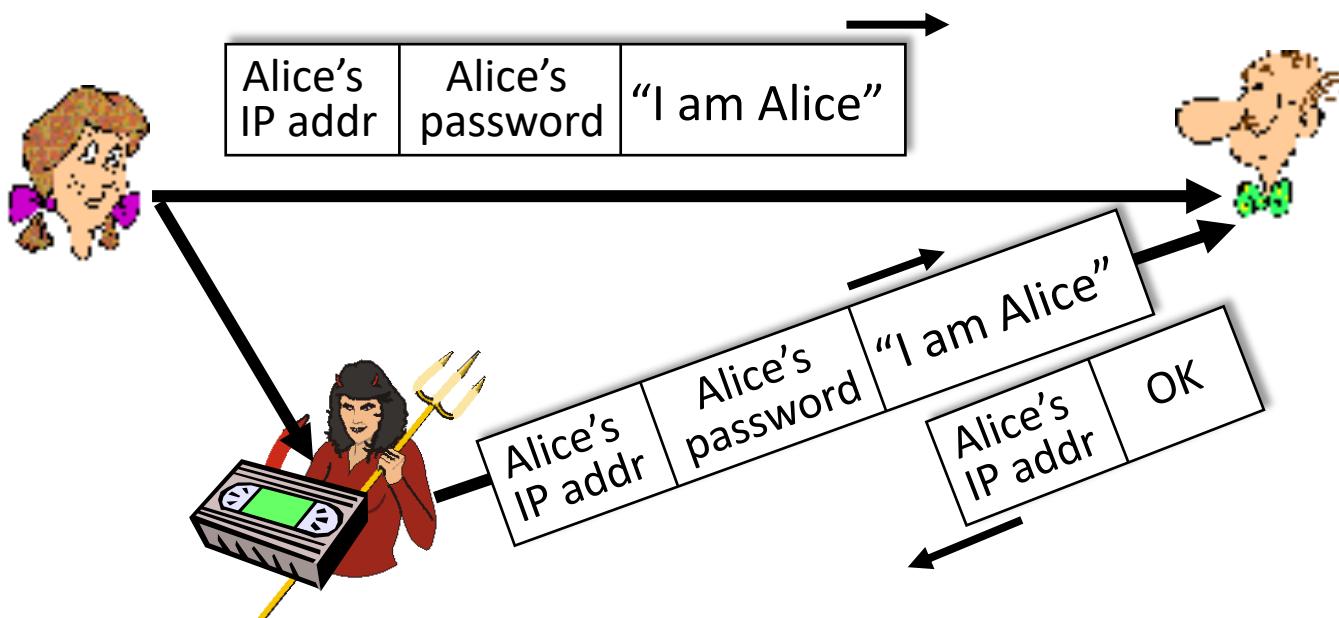
Protocol ap3.0: Alice says “I am Alice” Alice says “I am Alice” and sends her secret password to “prove” it.



Authentication: a third try

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap3.0: Alice says “I am Alice” Alice says “I am Alice” and sends her secret password to “prove” it.

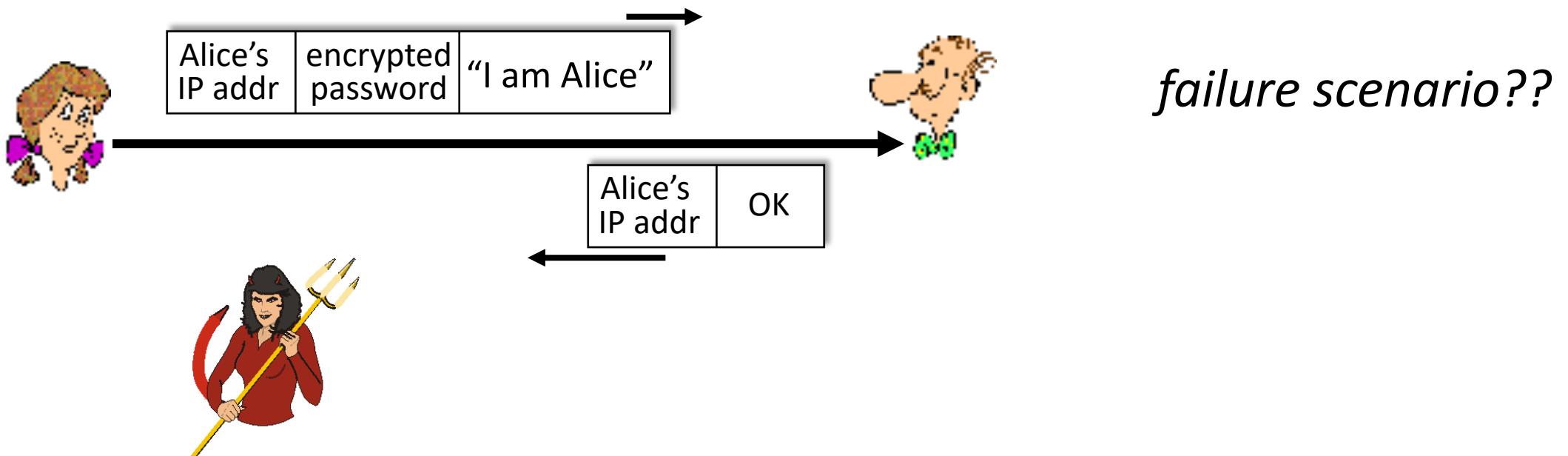


*playback attack:
Trudy records
Alice's packet
and later
plays it back to Bob*

Authentication: a modified third try

Goal: Bob wants Alice to “prove” her identity to him

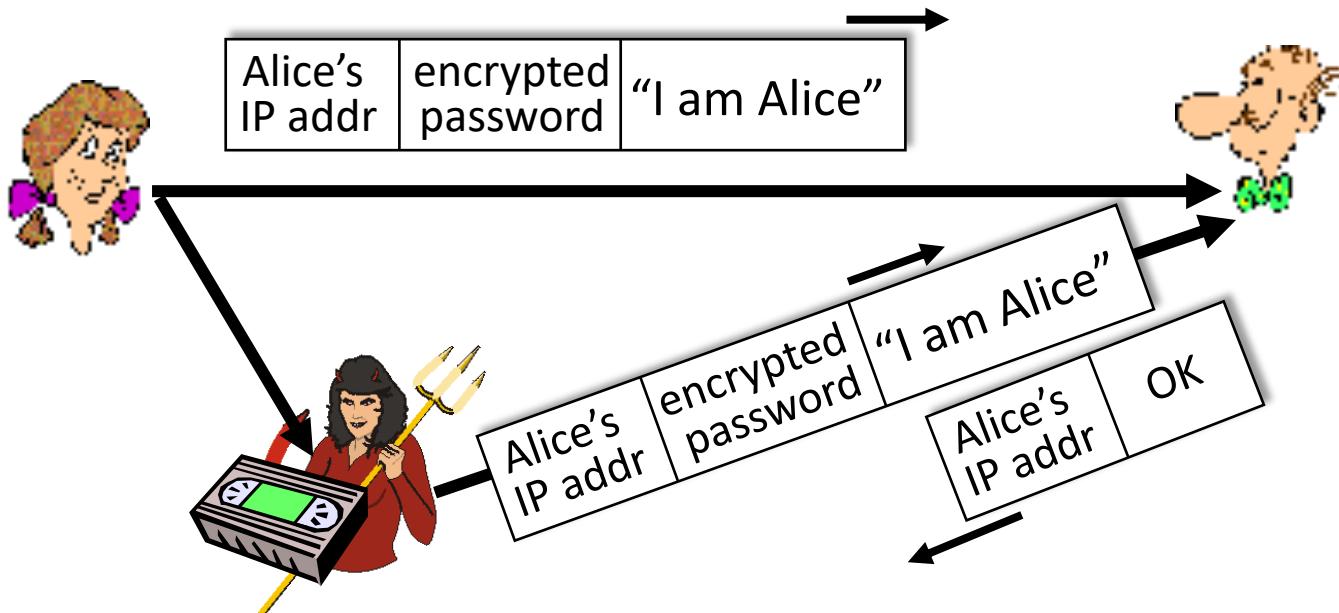
Protocol ap3.0: Alice says “I am Alice” Alice says “I am Alice” and sends her encrypted secret password to “prove” it.



Authentication: a modified third try

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap3.0: Alice says “I am Alice” Alice says “I am Alice” and sends her encrypted secret password to “prove” it.



playback attack still works: Trudy records Alice's packet and later plays it back to Bob

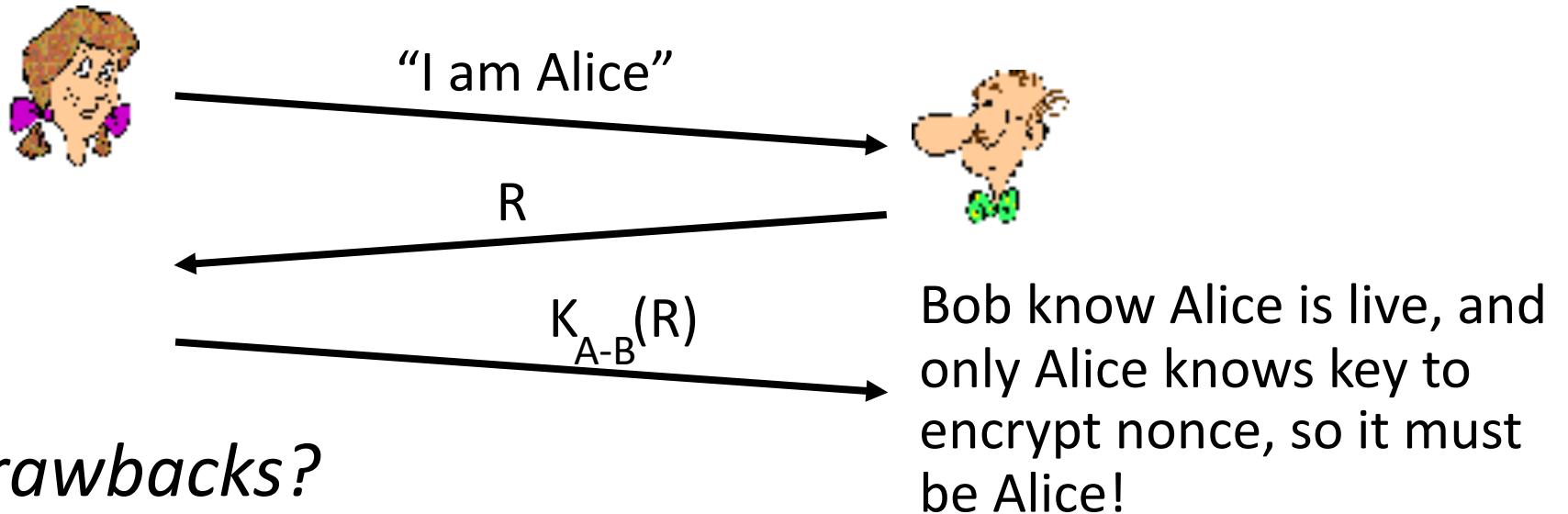
Authentication: a fourth try

Goal: avoid playback attack

nonce: number (R) used only **once-in-a-lifetime**

protocol ap4.0: to prove Alice “live”, Bob sends Alice nonce, R

- Alice must return R , encrypted with shared secret key

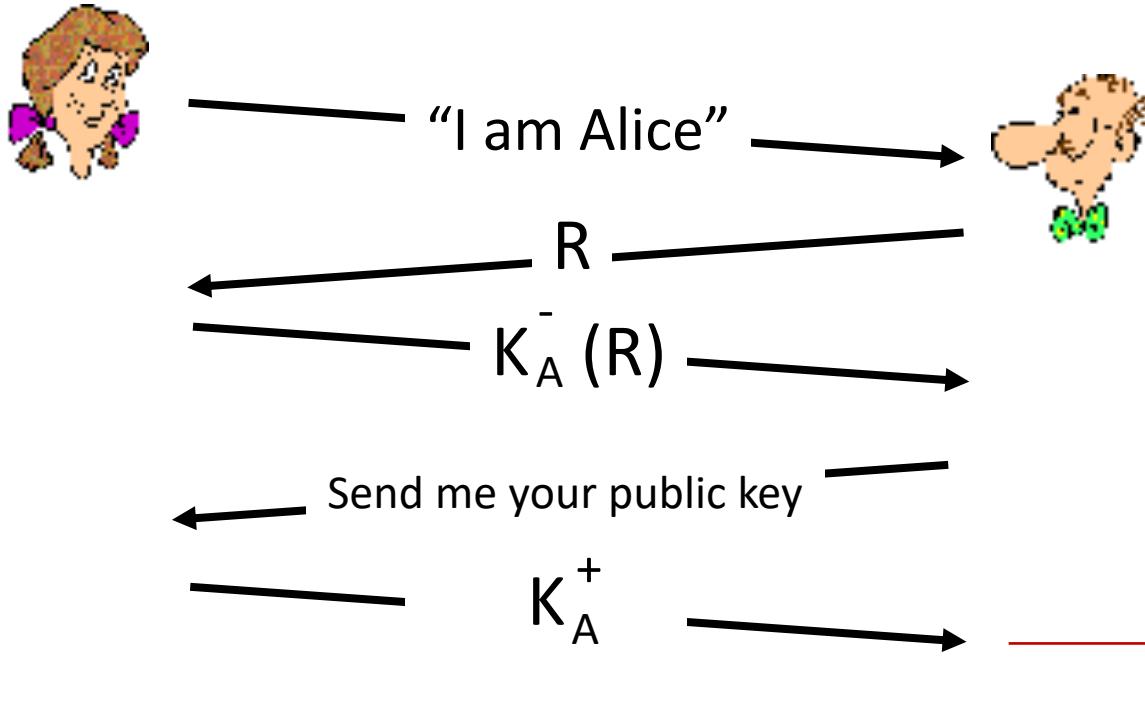


Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key - can we authenticate using public key techniques?

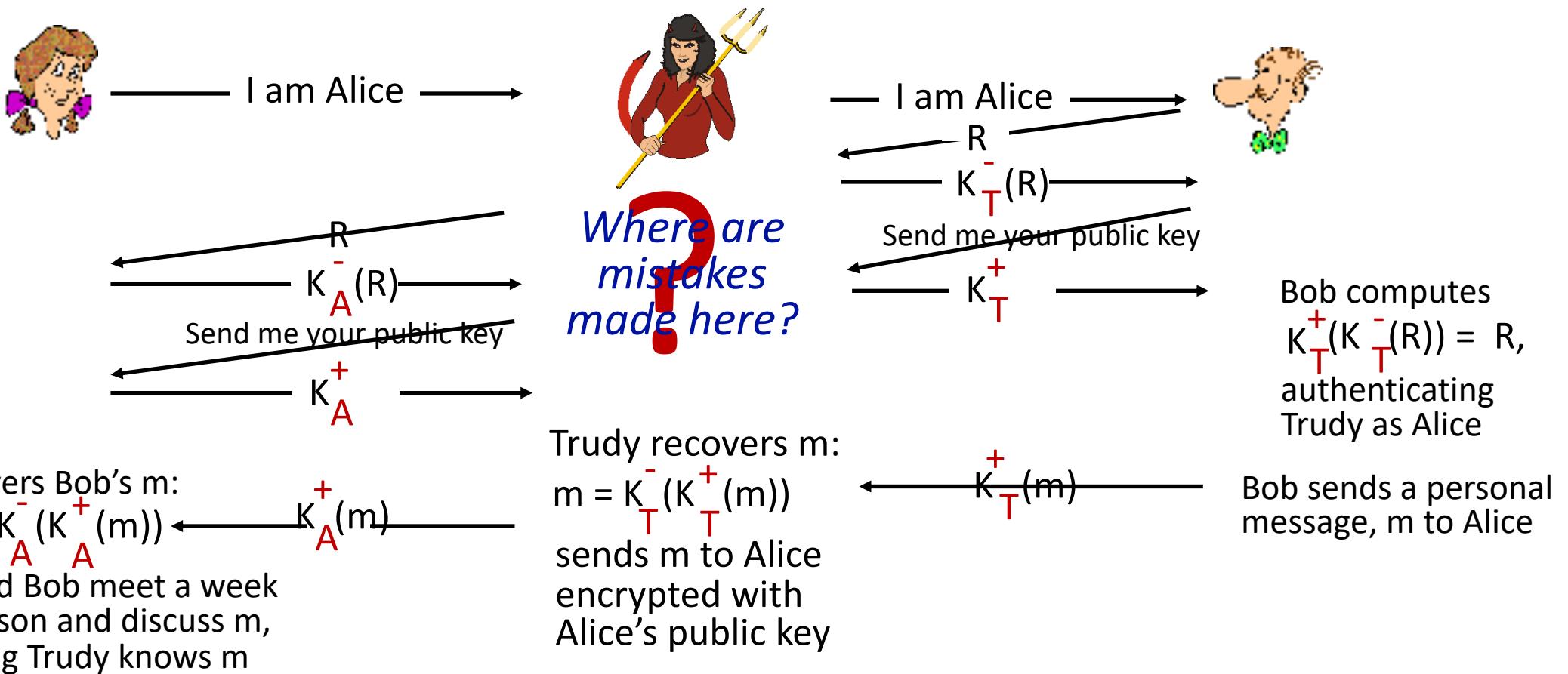
ap5.0: use nonce, public key cryptography



Bob computes
 $K_A^+ (K_A^-(R)) = R$
and knows only Alice could have the private key, that encrypted R such that
 $K_A^+ (K_A^-(R)) = R$

Authentication: ap5.0 – there's still a flaw!

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Chapter 6 outline

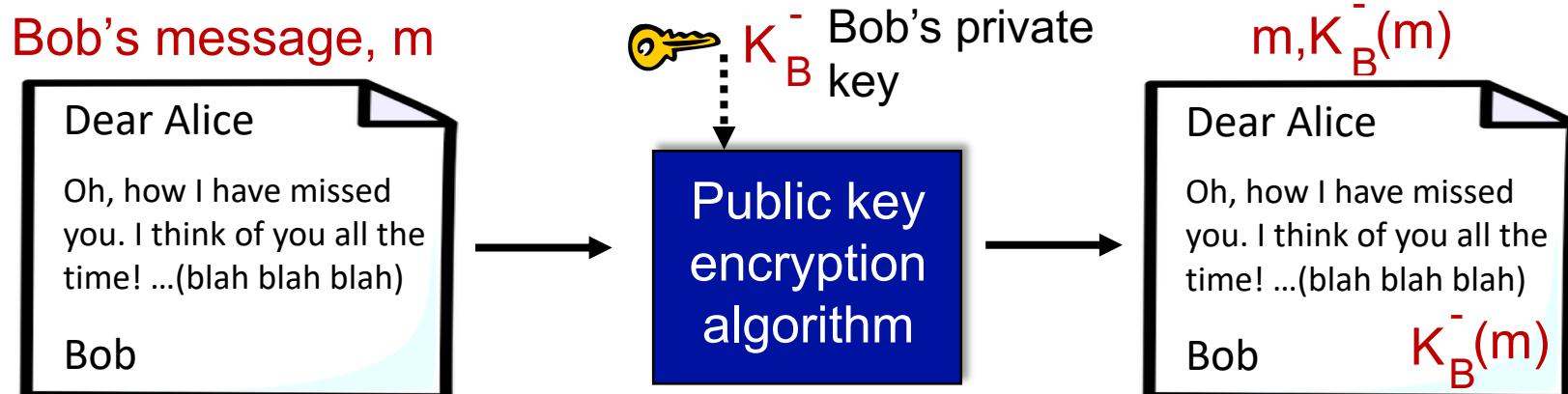
- What is network security?
- Principles of cryptography
- Authentication, **message integrity**
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- Operational security: firewalls and IDS



Digital signatures

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document: he is document owner/creator.
- *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
- simple digital signature for message m :
 - Bob signs m by encrypting with his private key K_B^- , creating “signed” message, $K_B^-(m)$



Digital signatures

- suppose Alice receives msg m , with signature: $m, \bar{K}_B(m)$
- Alice verifies m signed by Bob by applying Bob's public key \bar{K}_B^+ to $\bar{K}_B(m)$ then checks $K_B^+(\bar{K}_B(m)) = m$.
- If $K_B^+(\bar{K}_B(m)) = m$, whoever signed m must have used Bob's private key

Alice thus verifies that:

- Bob signed m
- no one else signed m
- Bob signed m and not m'

non-repudiation:

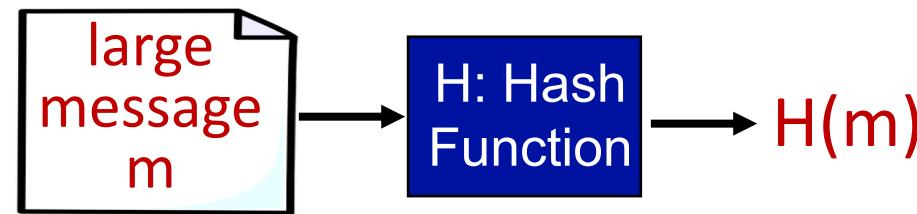
- ✓ Alice can take m , and signature $\bar{K}_B(m)$ to court and prove that Bob signed m

Message digests

computationally expensive to public-key-encrypt long messages

goal: fixed-length, easy- to-compute digital “fingerprint”

- apply hash function H to m , get fixed size message digest, $H(m)$

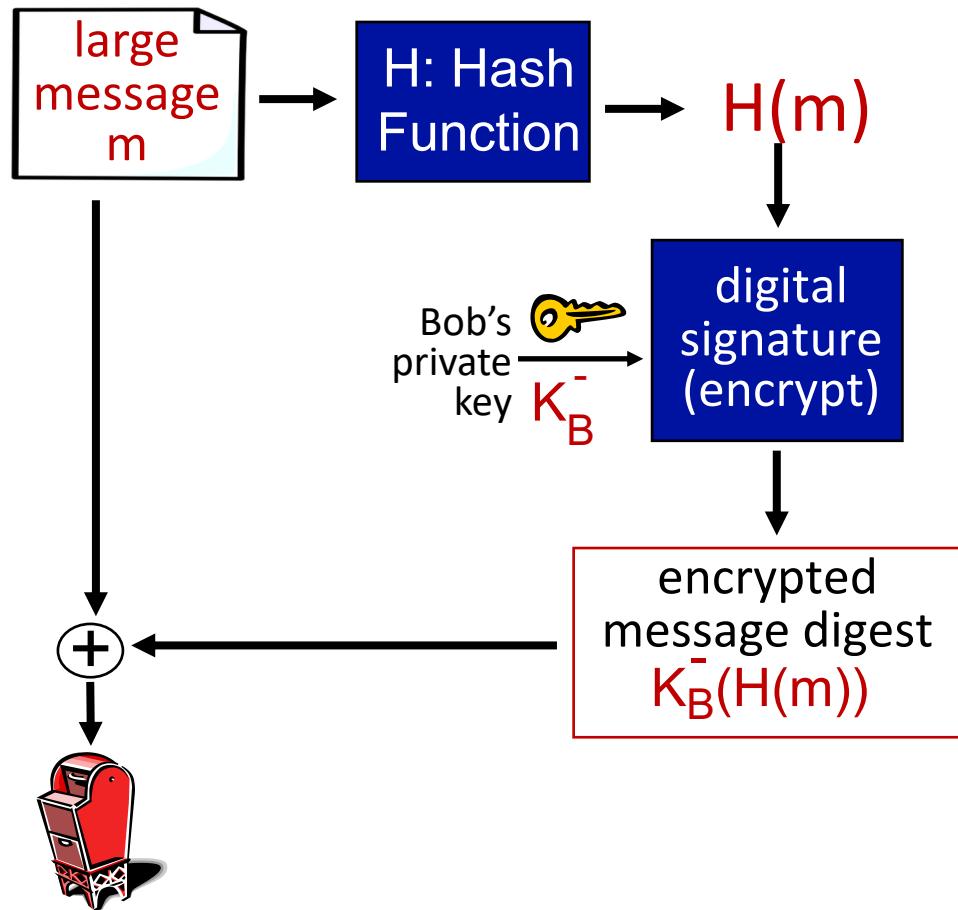


Hash function properties:

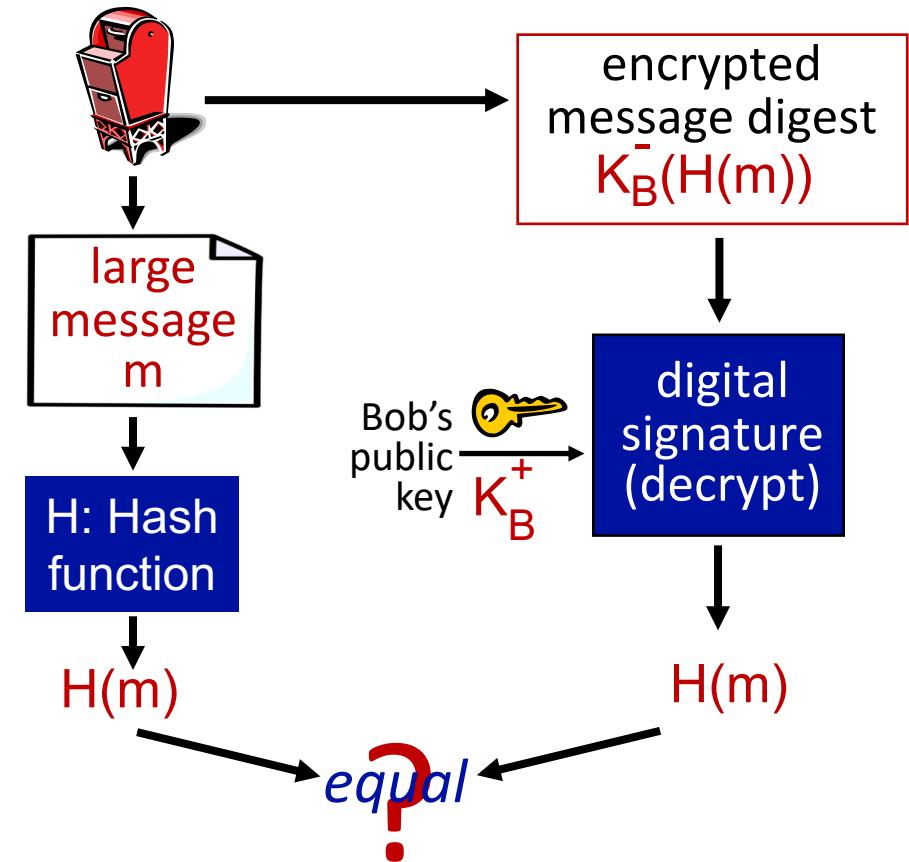
- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest x , computationally infeasible to find m such that $x = H(m)$

Digital signature = signed message digest

Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:

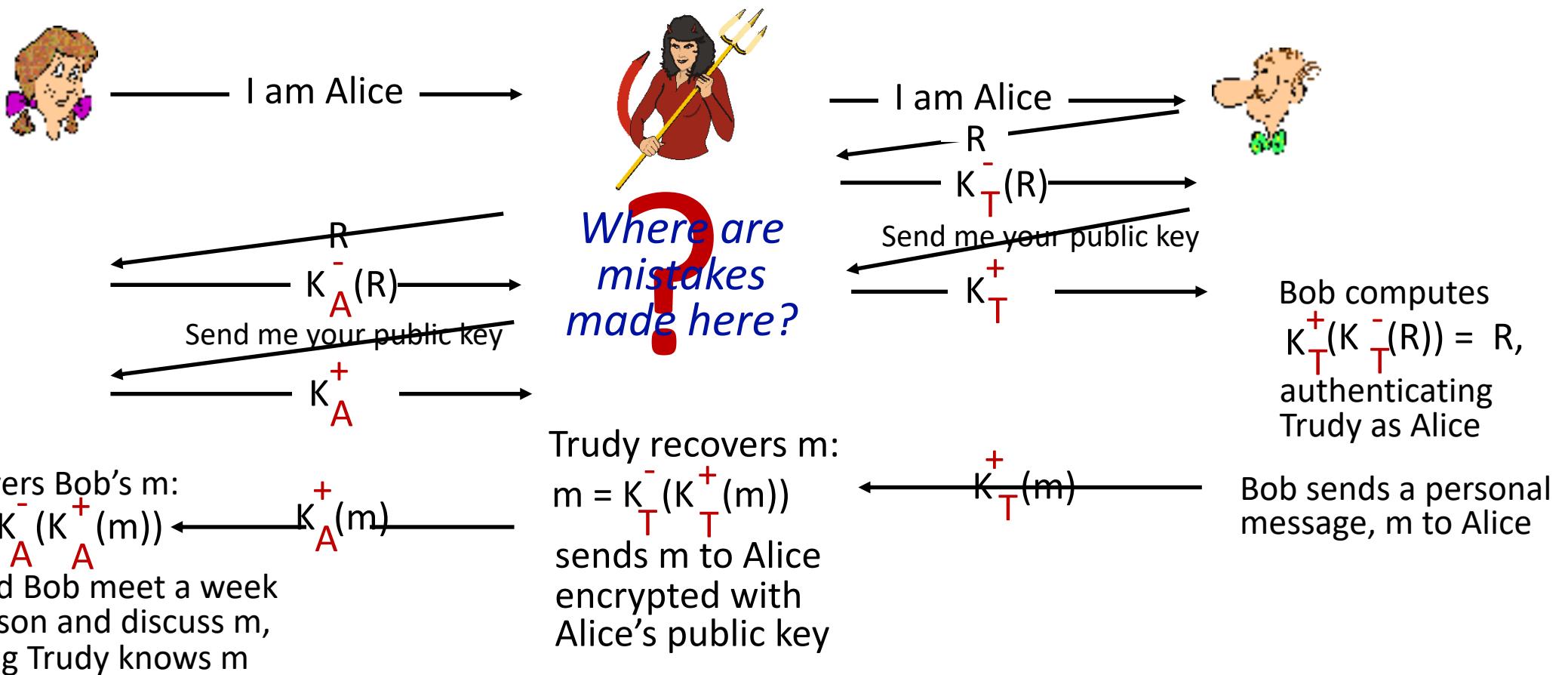


Hash function algorithms

- MD5 hash function widely used (RFC 1321)
 - computes 128-bit message digest in 4-step process.
 - arbitrary 128-bit string x , appears difficult to construct msg m whose MD5 hash is equal to x
- SHA-1 is also used
 - US standard [NIST, FIPS PUB 180-1]
 - 160-bit message digest

Authentication: ap5.0 – let's fix it!!

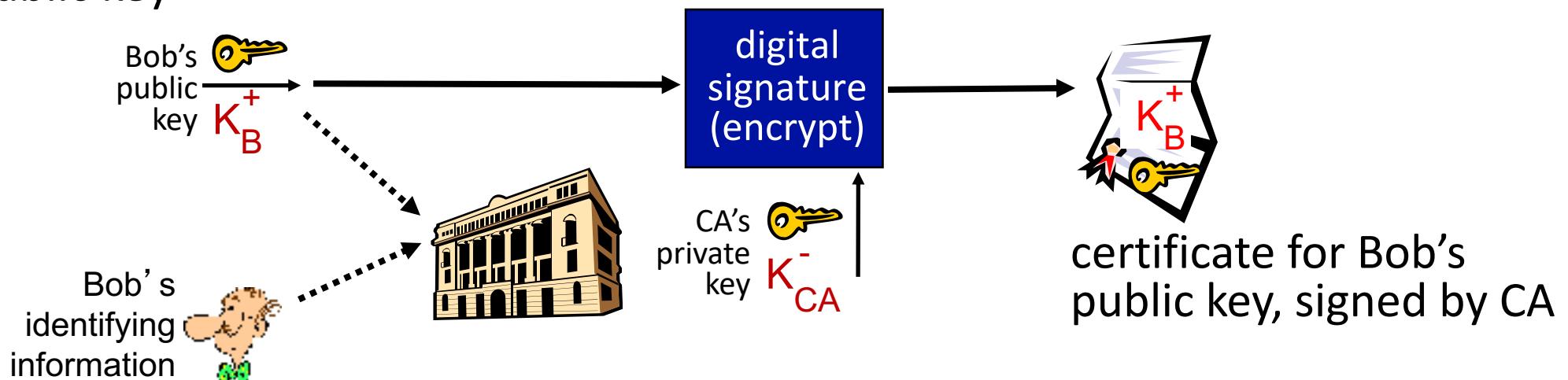
Recall the problem: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Need for certified public keys

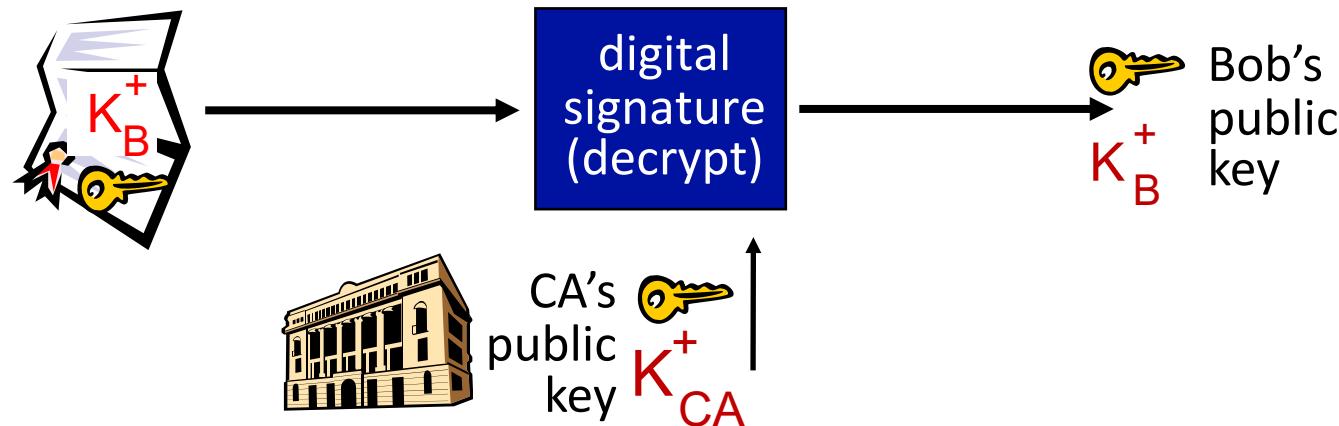
Public key Certification Authorities (CA)

- certification authority (CA): binds public key to particular entity, E
- entity (person, website, router) registers its public key with CE provides “proof of identity” to CA
 - CA creates certificate binding identity E to E’s public key
 - certificate containing E’s public key digitally signed by CA: CA says “this is E’s public key”



Public key Certification Authorities (CA)

- when Alice wants Bob's public key:
 - gets Bob's certificate (Bob or elsewhere)
 - apply CA's public key to Bob's certificate, get Bob's public key



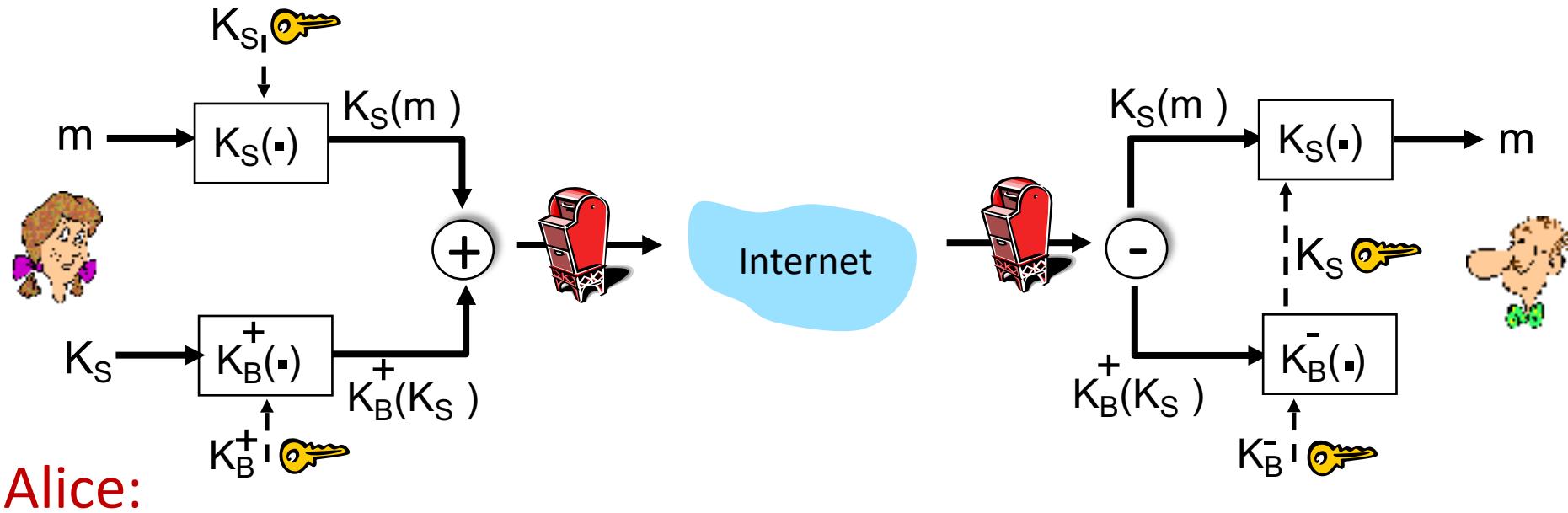
Chapter 6 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- **Securing e-mail**
- Securing TCP connections: TLS
- Network layer security: IPsec
- Operational security: firewalls and IDS



Secure e-mail: confidentiality

Alice wants to send *confidential* e-mail, m , to Bob.

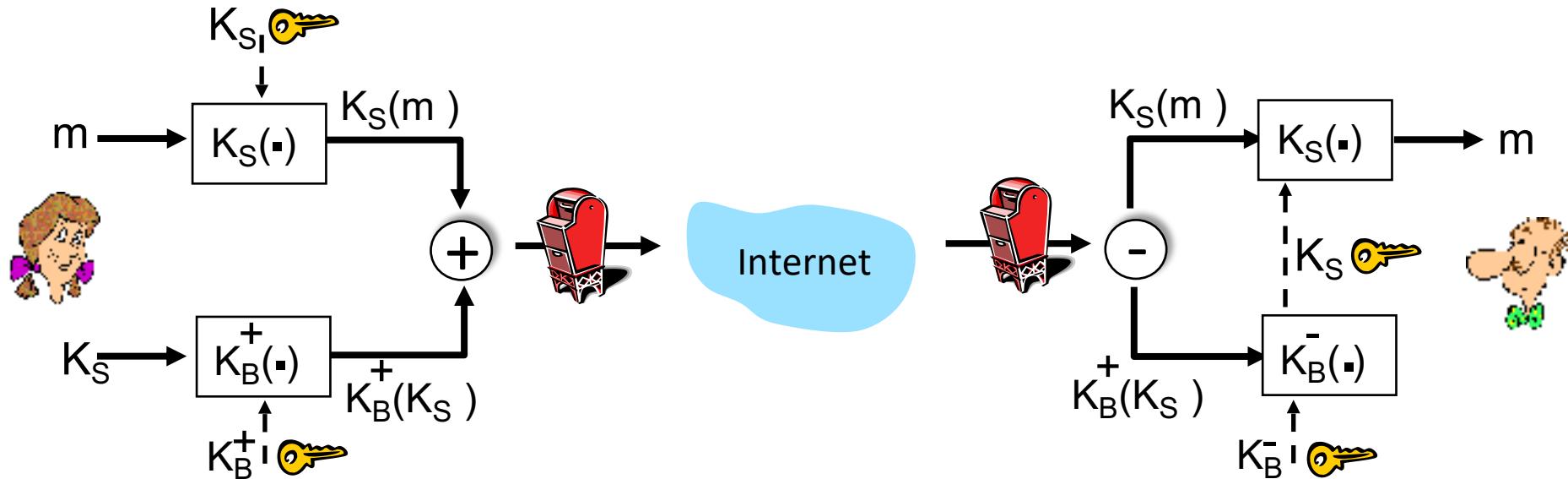


Alice:

- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K_B^+(K_S)$ to Bob

Secure e-mail: confidentiality (more)

Alice wants to send *confidential* e-mail, m , to Bob.

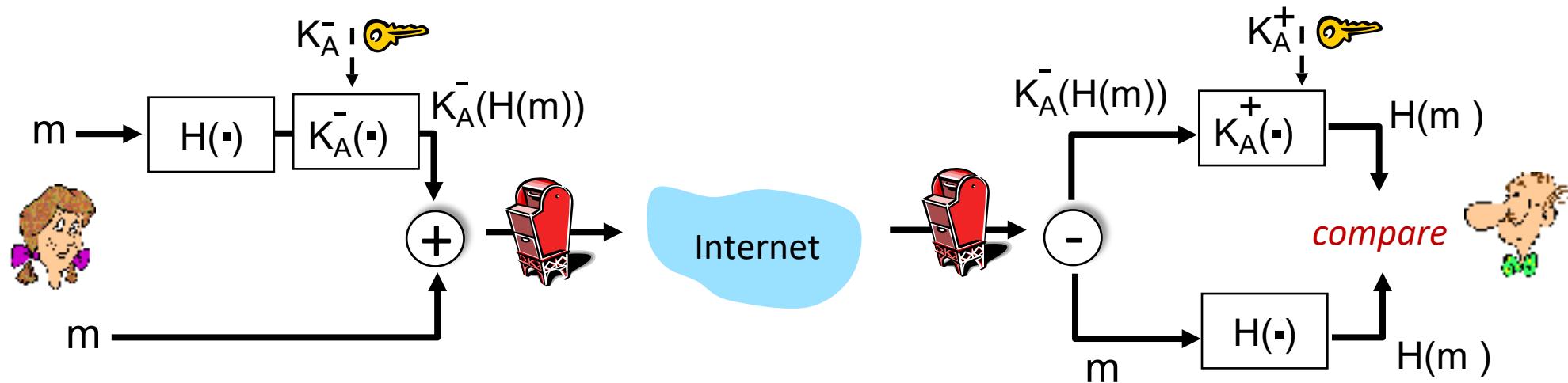


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail: integrity, authentication

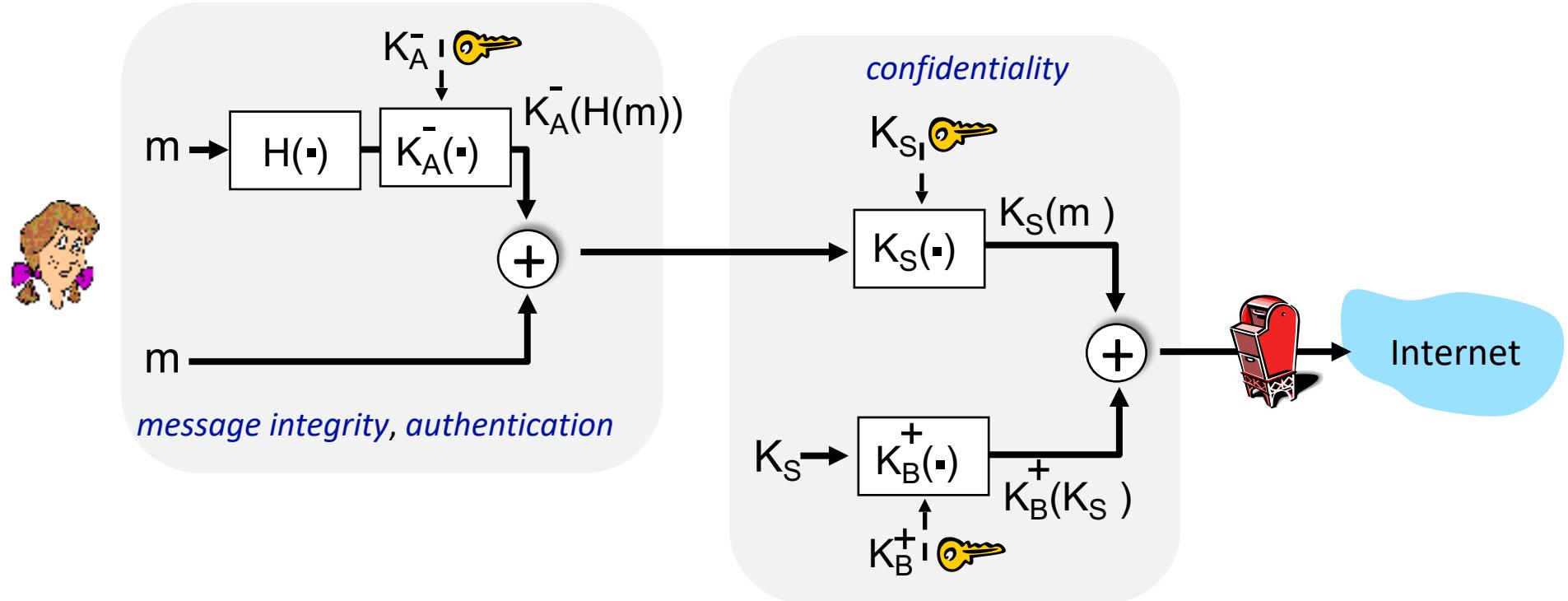
Alice wants to send m to Bob, with *message integrity, authentication*



- Alice digitally signs hash of her message with her private key, providing integrity and authentication
- sends both message (in the clear) and digital signature

Secure e-mail: integrity, authentication

Alice sends m to Bob, with *confidentiality, message integrity, authentication*



Alice uses three keys: her private key, Bob's public key, new symmetric key

What are Bob's complementary actions?

Chapter 6 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS**
- Network layer security: IPsec
- Operational security: firewalls and IDS

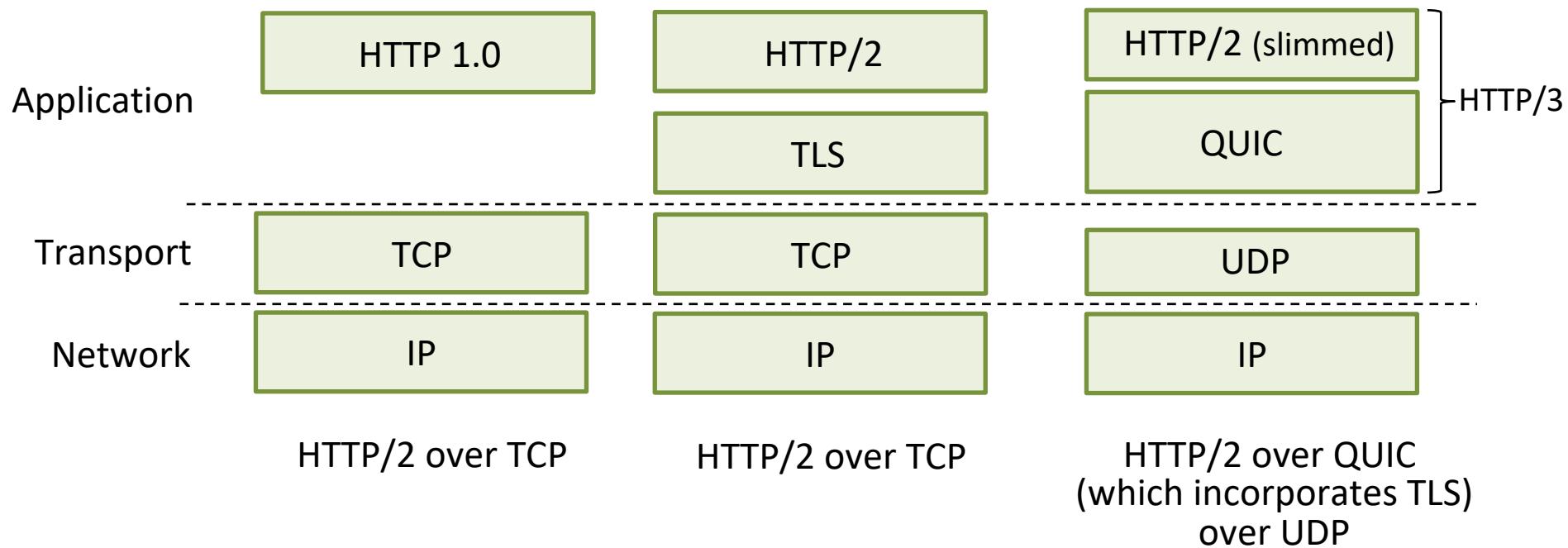


Transport-layer security (TLS)

- widely deployed security protocol above the transport layer
 - supported by almost all browsers, web servers: https (port 443)
 - provides:
 - **confidentiality**: via *symmetric encryption*
 - **integrity**: via *cryptographic hashing*
 - **authentication**: via *public key cryptography*
 - history:
 - early research, implementation: secure network programming, secure sockets
 - secure socket layer (SSL) deprecated [2015]
 - TLS 1.3: RFC 8846 [2018]
- 
- all techniques we have studied!*

Transport-layer security (TLS)

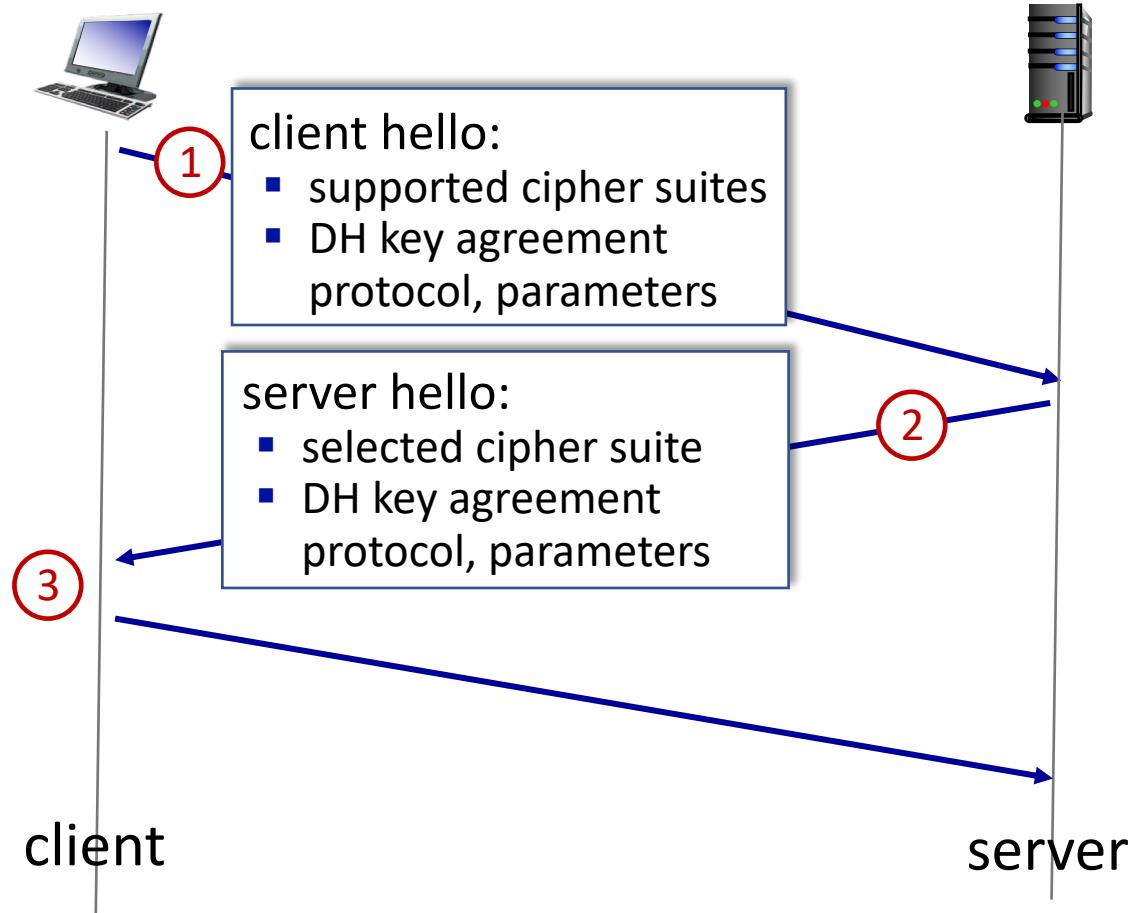
- TLS provides an API that *any* application can use
- an HTTP view of TLS:



TLS: 1.3 cipher suite

- “cipher suite”: algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
 - only 5 choices, rather than 37 choices
 - *requires* Diffie-Hellman (DH) for key exchange, rather than DH or RSA
 - combined encryption and authentication algorithm (“authenticated encryption”) for data rather than serial encryption, authentication
 - 4 based on AES
 - HMAC uses SHA (256 or 284) cryptographic hash function

TLS 1.3 handshake: 1 RTT



- ① client TLS hello msg:
 - guesses key agreement protocol, parameters
 - indicates cipher suites it supports
- ② server TLS hello msg chooses
 - key agreement protocol, parameters
 - cipher suite
 - server-signed certificate
- ③ client:
 - checks server certificate
 - generates key
 - can now make application request (e.g., HTTPS GET)

Chapter 6 outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- **Network layer security: IPsec**
- Operational security: firewalls and IDS



VPN Security Requirements

- Confidentiality
 - hide packet content
- Content authentication (integrity)
 - if a packet is changed, replaced, or faked?
- Origin authentication (authentication)
 - is the real sender lying about its identity?
- Anti-replay:
 - a bad guy replaying a packet sent by others
- The goal of IPSec is to satisfy the 4 requirements!

Confidentiality

- Confidentiality is given through:
 - **Encryption**
 - Topic covered
 - In VPN, **triple DES** is used (in ESP)
 - To do encryption, key exchange must be done firstly
 - we need IKE!

Integrity

- Content Integrity is given through:
 - One-way hash function
 - **Keyed** hash function (HMAC)
 - $\text{HMAC}(k, d) = h(k \text{ XOR } h(k \text{ XOR } d))$
 - Digital signature

- Origin Integrity is given through:
 - Digital certificate
 - Private key, public key
 - Kerberos

Anti-replay

- Anti-replay is given through:

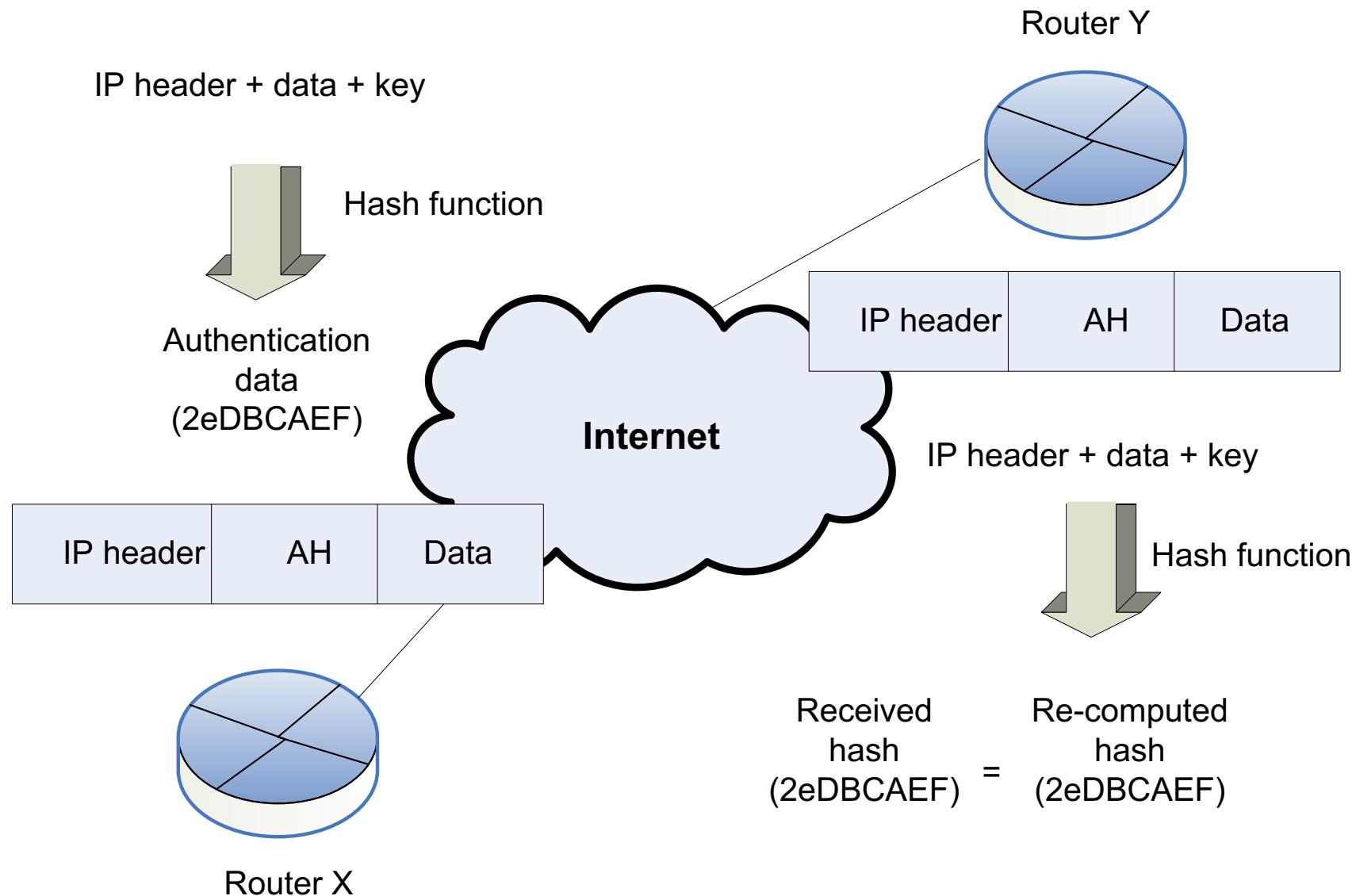
- One-time password
- Sequence number
- Timestamp

IPSec

- IPSec: Internet Protocol Security

- A framework of open standards
- Authentication Header (AH) + Encapsulating Security Payloads (ESP) + Internet Key Exchange (IKE)
 - AH gives you integrity + authentication
 - Authenticated ESP (tunnel mode) gives you confidentiality + integrity + authentication
 - IKE takes care of key exchange
 - Without key exchange, NONE of the 3 can be provided

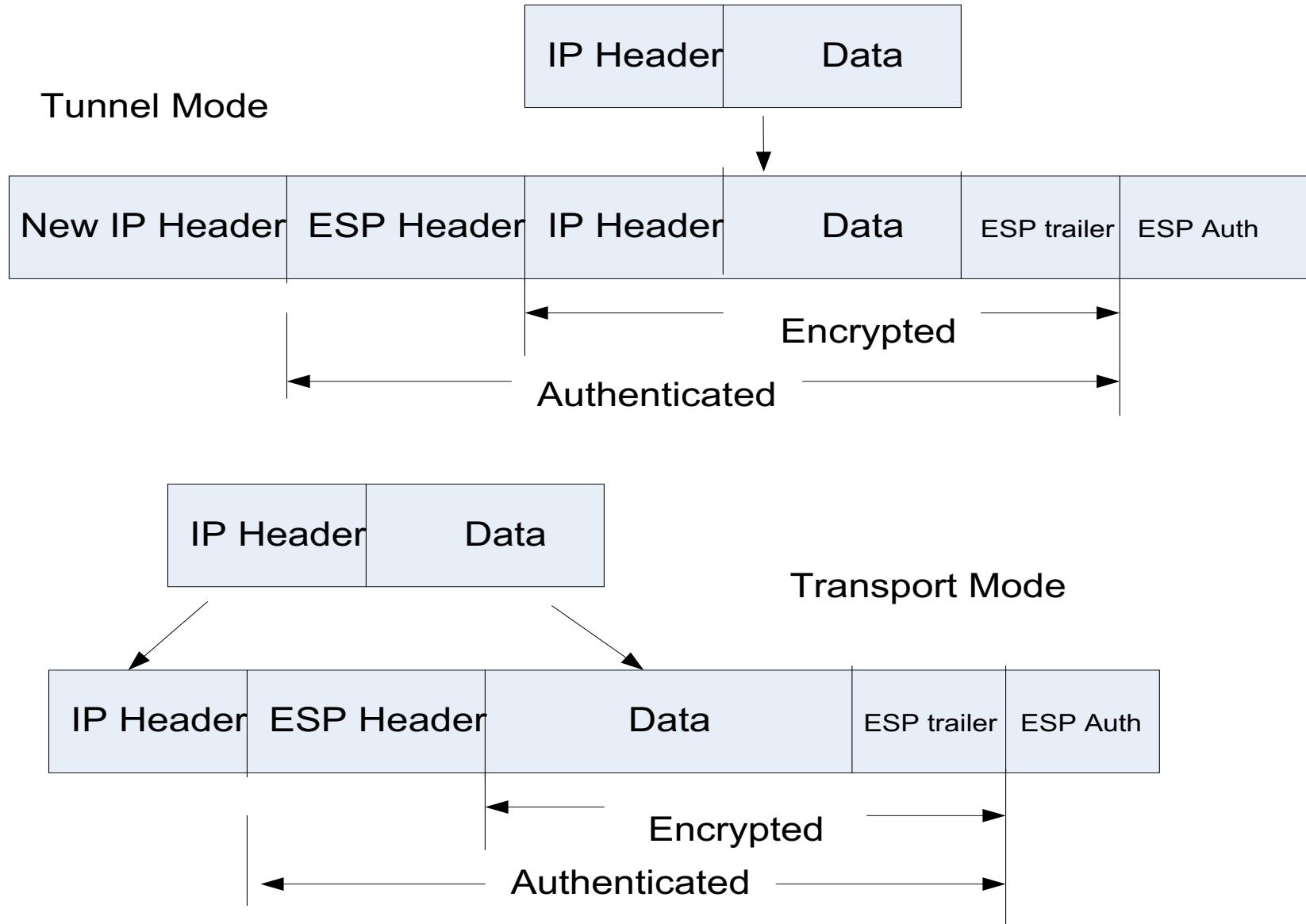
AH



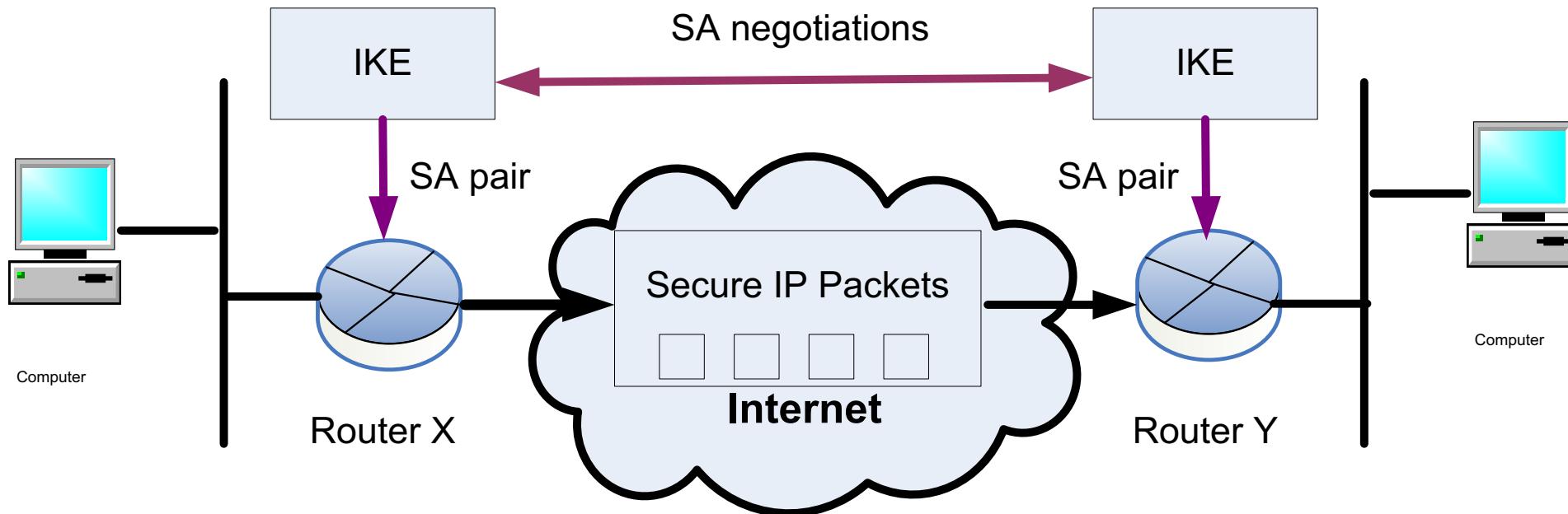
ESP

- ESP can be used in 2 modes:
 - Transport mode
 - Encrypt traffic between two peer devices
 - Tunnel mode
 - Encrypt traffic between two subnets

ESP



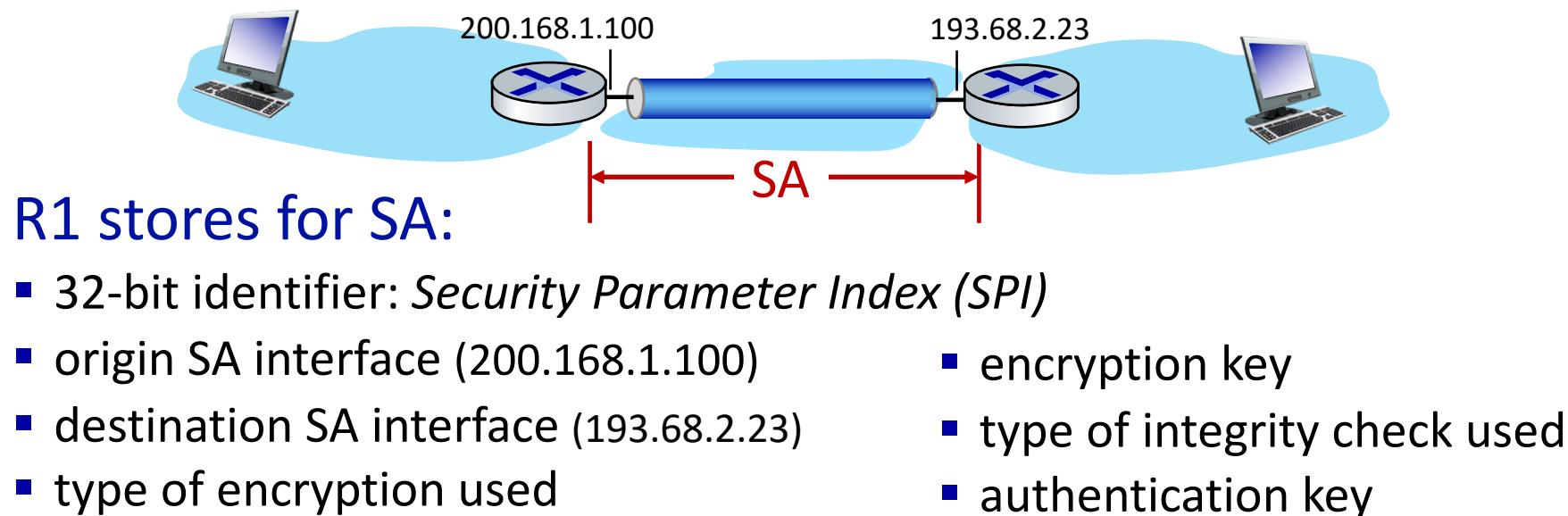
IKE



Both AH and ESP need keys!! But, IKE is optional.

Security associations (SAs)

- before sending data, **security association (SA)** established from sending to receiving entity (directional)
- ending, receiving entities maintain *state information* about SA
 - recall: TCP endpoints also maintain state info
 - IP is connectionless; IPsec is connection-oriented!



IKE: Internet Key Exchange

- *previous examples:*

Example SA:

SPI: 12345

Source IP: 200.168.1.100

Dest IP: 193.68.2.23

Protocol: ESP

Encryption algorithm: 3DES-cbc

HMAC algorithm: MD5

Encryption key: 0x7aeaca...

HMAC key: 0xc0291f...

Chapter 6 outline

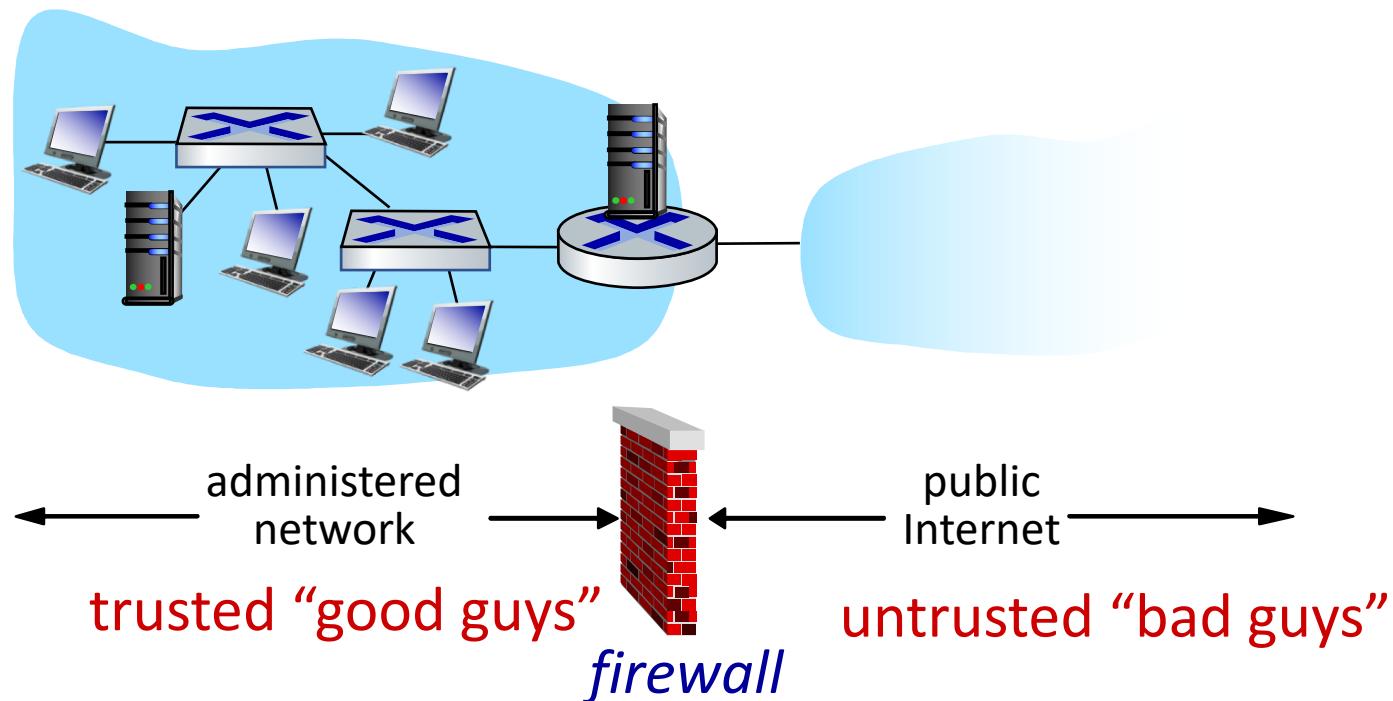
- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec
- **Operational security: firewalls and IDS**



Firewalls

firewall

isolates organization's internal network from larger Internet, allowing some packets to pass, blocking others



Firewalls: why

prevent denial of service attacks:

- SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

prevent illegal modification/access of internal data

- e.g., attacker replaces CIA’s homepage with something else

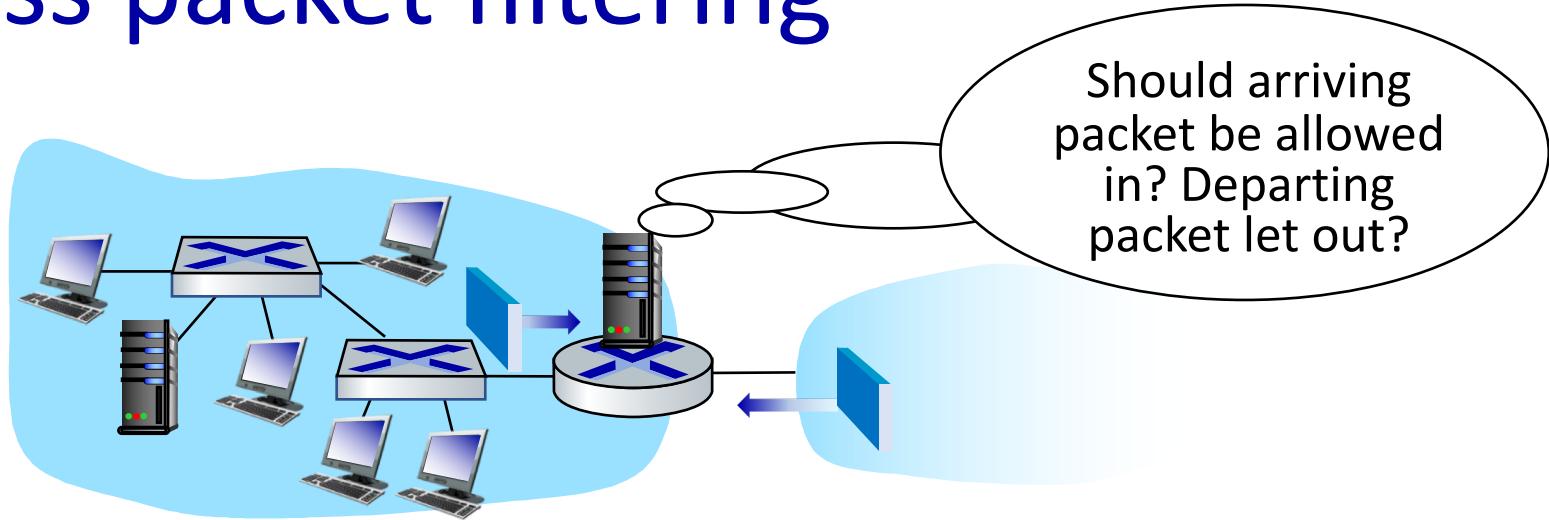
allow only authorized access to inside network

- set of authenticated users/hosts

three types of firewalls:

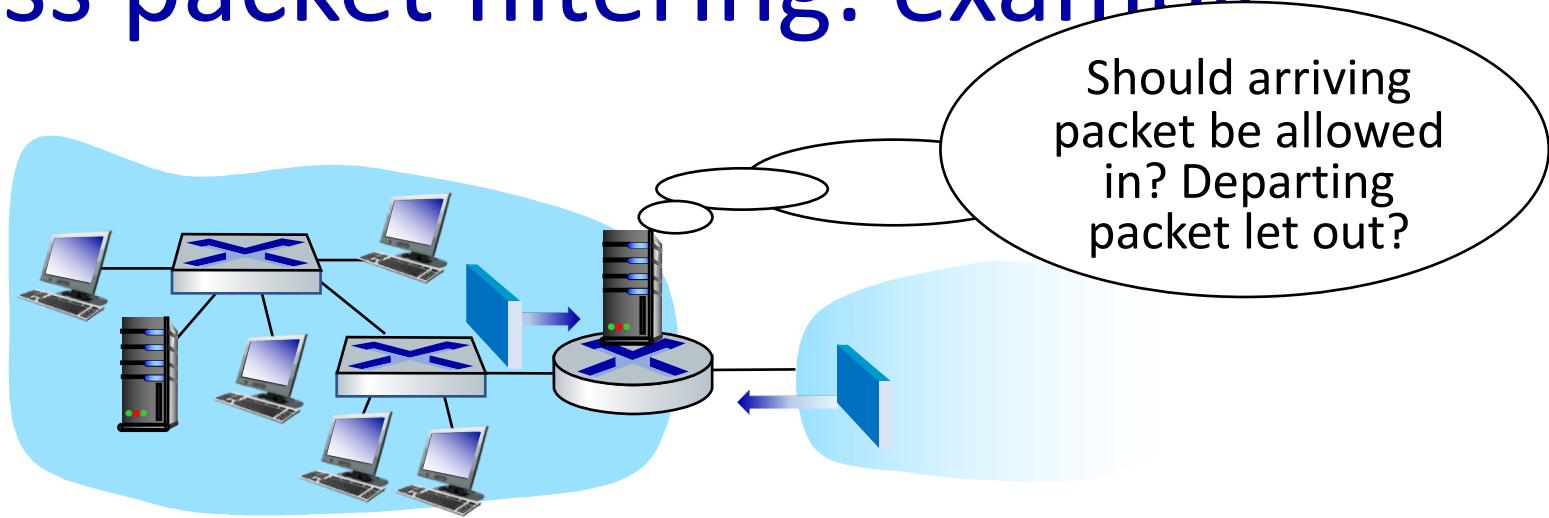
- stateless packet filters
- stateful packet filters
- application gateways

Stateless packet filtering



- internal network connected to Internet via router **firewall**
- filters **packet-by-packet**, decision to forward/drop packet based on:
 - source IP address, destination IP address
 - TCP/UDP source, destination port numbers
 - ICMP message type
 - TCP SYN, ACK bits

Stateless packet filtering: example



- **example 1:** block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
 - **result:** all incoming, outgoing UDP flows and telnet connections are blocked
- **example 2:** block inbound TCP segments with ACK=0
 - **result:** prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside

Stateless packet filtering: more examples

Policy	Firewall Setting
no outside Web access	drop all outgoing packets to any IP address, port 80
no incoming TCP connections, except those for institution's public Web server only.	drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
prevent Web-radios from eating up the available bandwidth.	drop all incoming UDP packets - except DNS and router broadcasts.
prevent your network from being used for a smurf DoS attack.	drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255)
prevent your network from being tracerouted	drop all outgoing ICMP TTL expired traffic

Access Control Lists

ACL: table of rules, applied top to bottom to incoming packets: (action, condition) pairs

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

Stateful packet filtering

- *stateless packet filter*: heavy handed tool

- admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

action	source address	dest address	protocol	source port	dest port	flag bit
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK

- *stateful packet filter*: track status of every TCP connection

- track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
 - timeout inactive connections at firewall: no longer admit packets

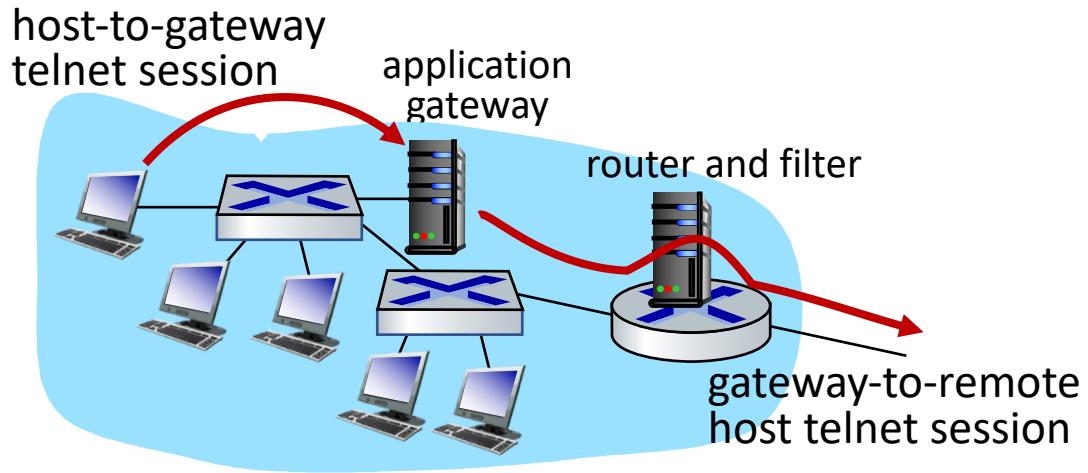
Stateful packet filtering

ACL augmented to indicate need to check connection state table before admitting packet

action	source address	dest address	proto	source port	dest port	flag bit	check connection
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

Application gateways

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside



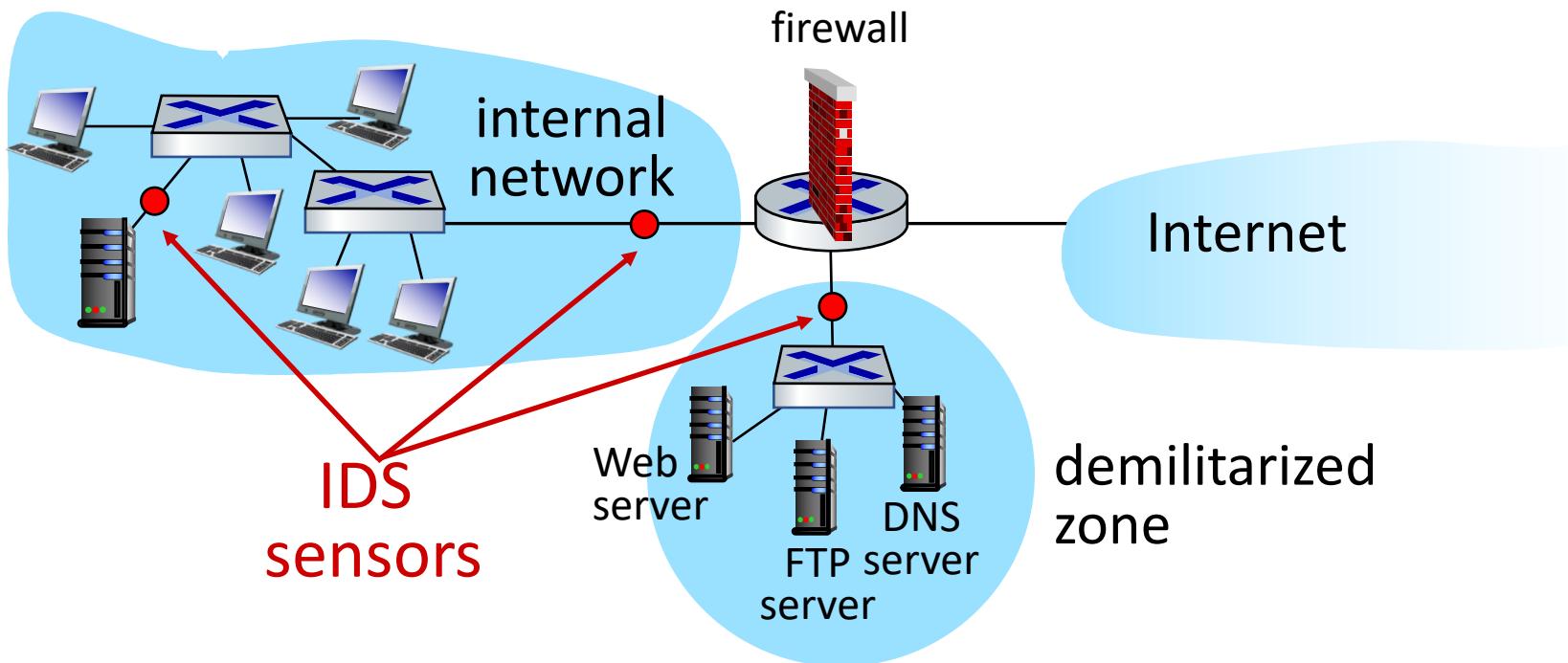
1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host
 - gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway

Intrusion detection systems

- packet filtering:
 - operates on TCP/IP headers only
 - no correlation check among sessions
- IDS: intrusion detection system
 - deep packet inspection: look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
 - examine correlation among multiple packets
 - port scanning
 - network mapping
 - DoS attack

Intrusion detection systems

multiple IDSs: different types of checking at different locations



Network Security (summary)

basic techniques.....

- cryptography (symmetric and public key)
- message integrity
- end-point authentication

.... used in many different security scenarios

- secure email
- secure transport (TLS)
- IP sec

operational security: firewalls and IDS

