

# Chapter I Introduction

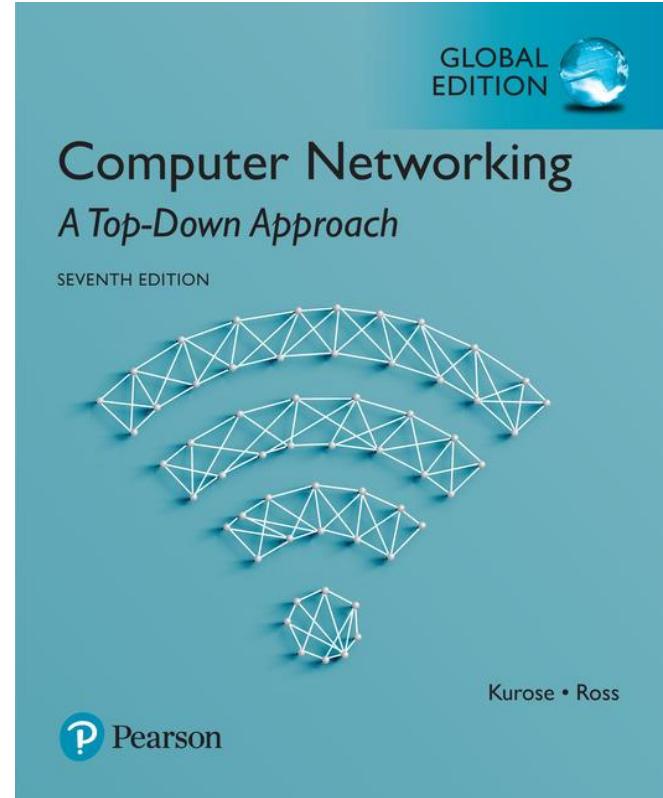
## A note on the use of these Powerpoint slides:

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

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

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016  
J.F Kurose and K.W. Ross, All Rights Reserved



## *Computer Networking: A Top Down Approach*

7<sup>th</sup> Edition, Global Edition  
Jim Kurose, Keith Ross  
Pearson  
April 2016

# 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~~ *handle more customers*
- 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

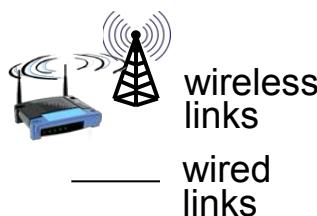
## I.6 networks under attack: security

## I.7 history

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

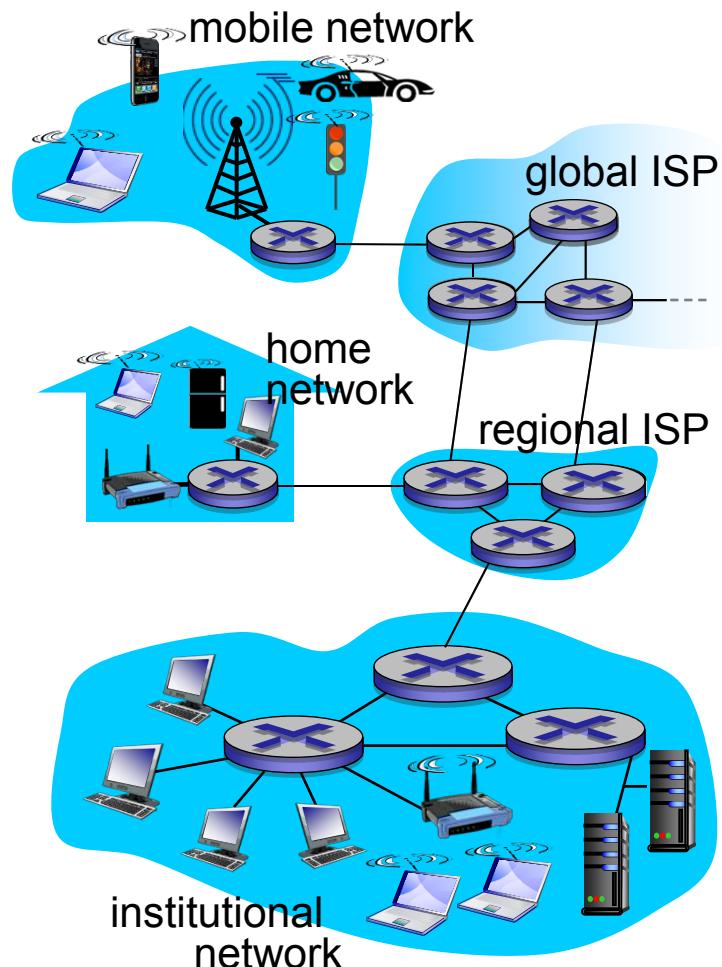


- billions of connected computing devices:
  - *hosts = end systems*
  - running *network apps*
  
- *communication links*
  - fiber, copper, radio, satellite
  - transmission rate: *bandwidth*: 넓은 대역폭



router  
Data Segmentation  
for reliable fast job scheduling

- packet switches: forward packets (chunks of data)
  - routers and switches  
more reliability & low cost.



# “Fun” Internet-connected devices



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



Web-enabled toaster +  
weather forecaster



Internet  
refrigerator



Slingbox: watch,  
control cable TV remotely



sensorized,  
bed  
mattress



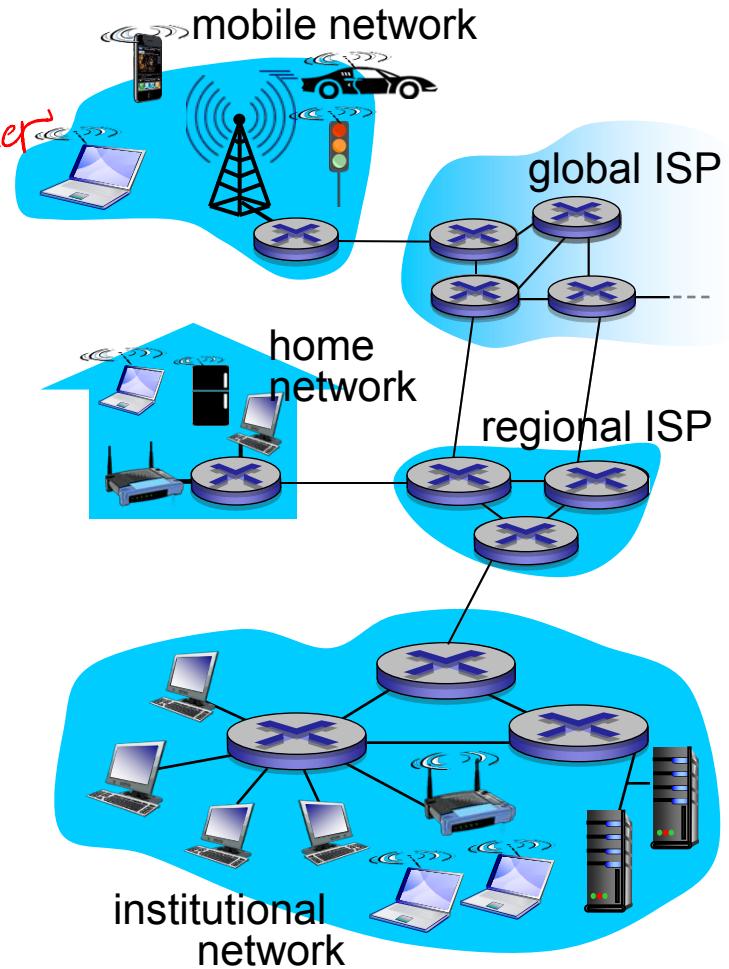
Tweet-a-watt:  
monitor energy use



Internet phones

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

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

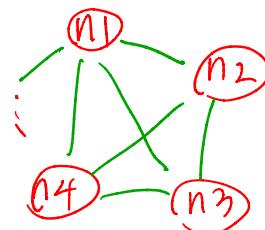


node

{ n<sub>1</sub>, n<sub>2</sub>, n<sub>3</sub>, ... }

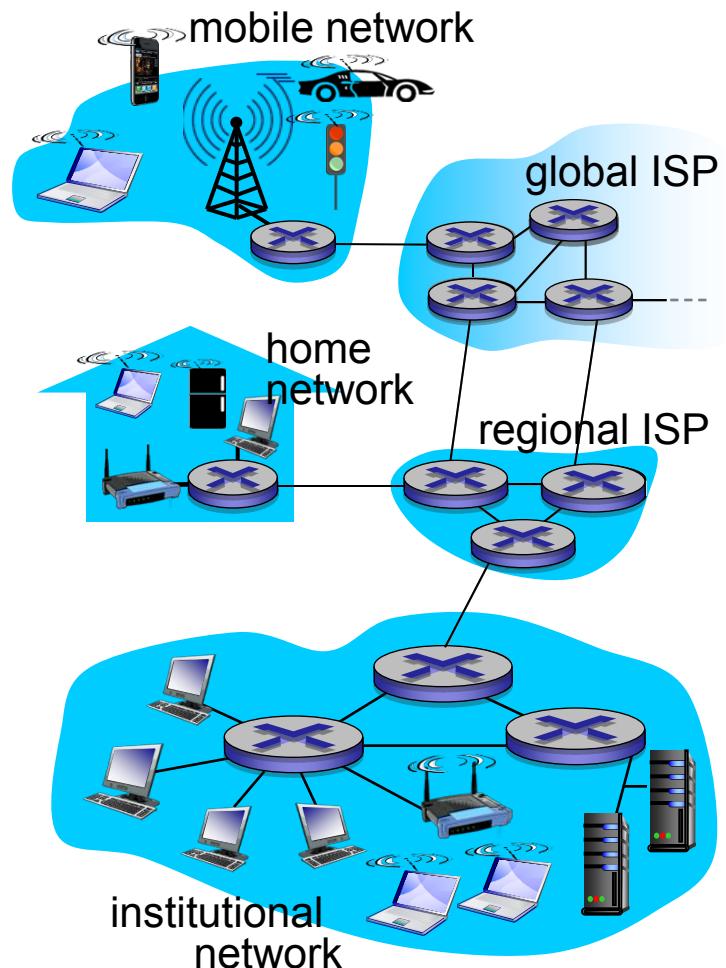
edge

{ (n<sub>1</sub>, n<sub>2</sub>), (n<sub>2</sub>, n<sub>3</sub>) }  
(n<sub>1</sub>, n<sub>3</sub>) ... }



# 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 messages sent

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

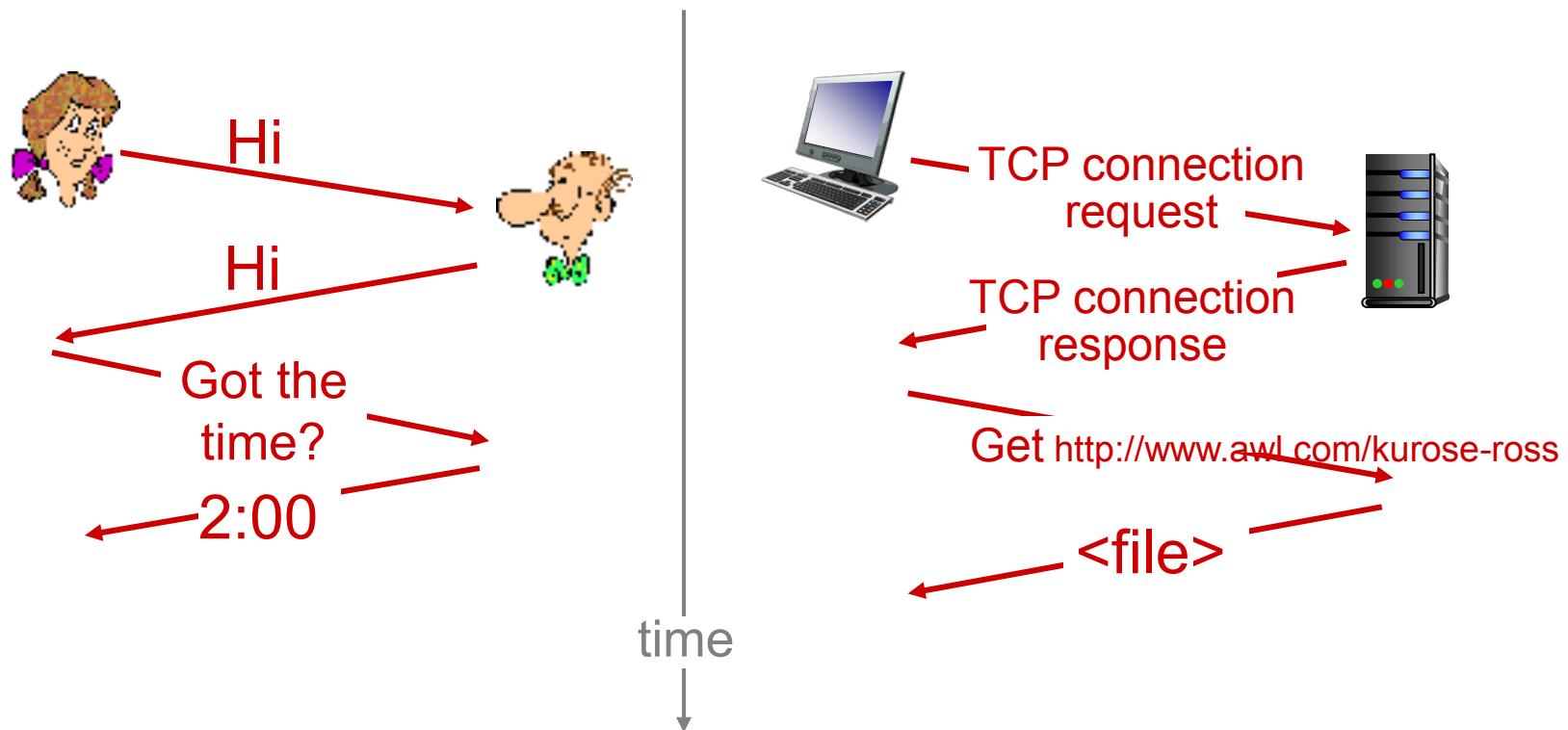
## *network protocols:*

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

*protocols define format, order of messages sent and received among network entities, and actions taken on message 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

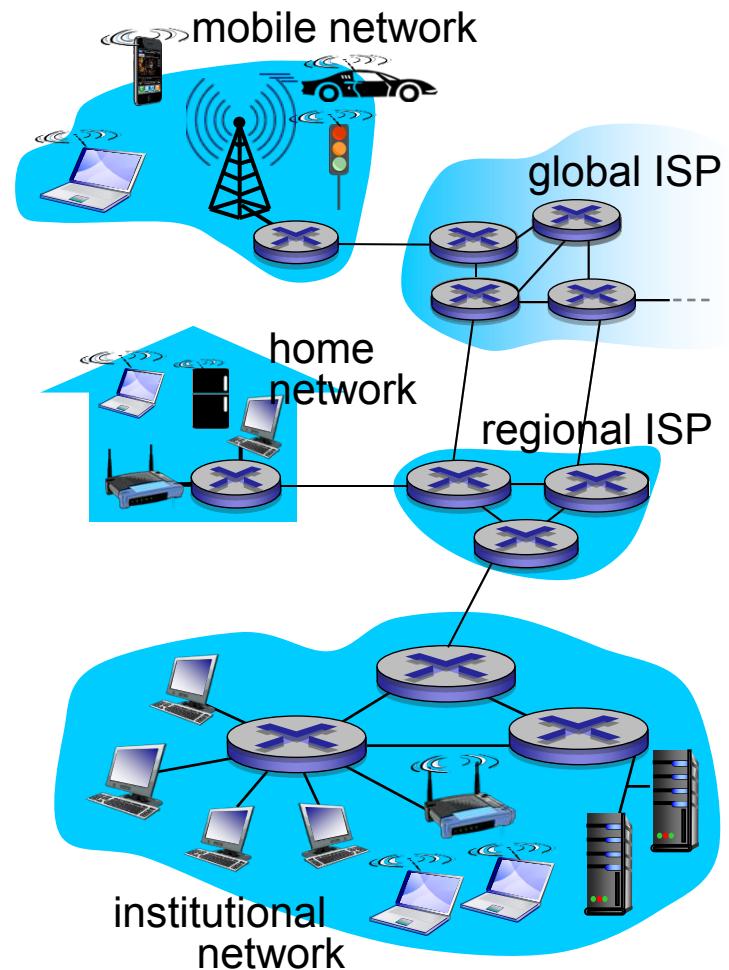
I.5 protocol layers, service models

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?*

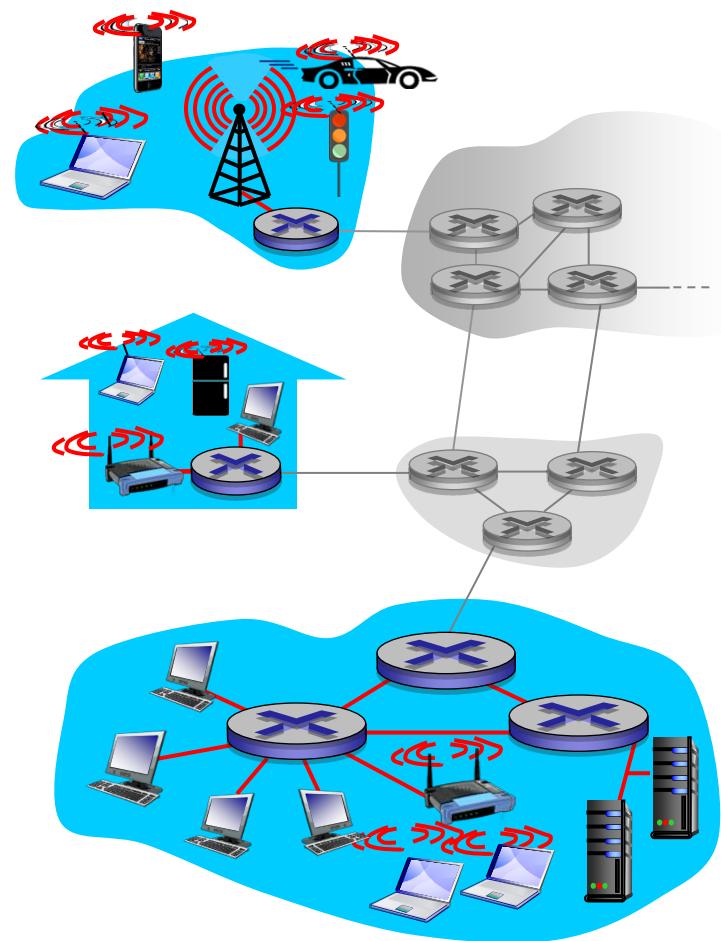
- residential access nets
- institutional access networks (school, company)
- mobile access networks

*keep in mind:*

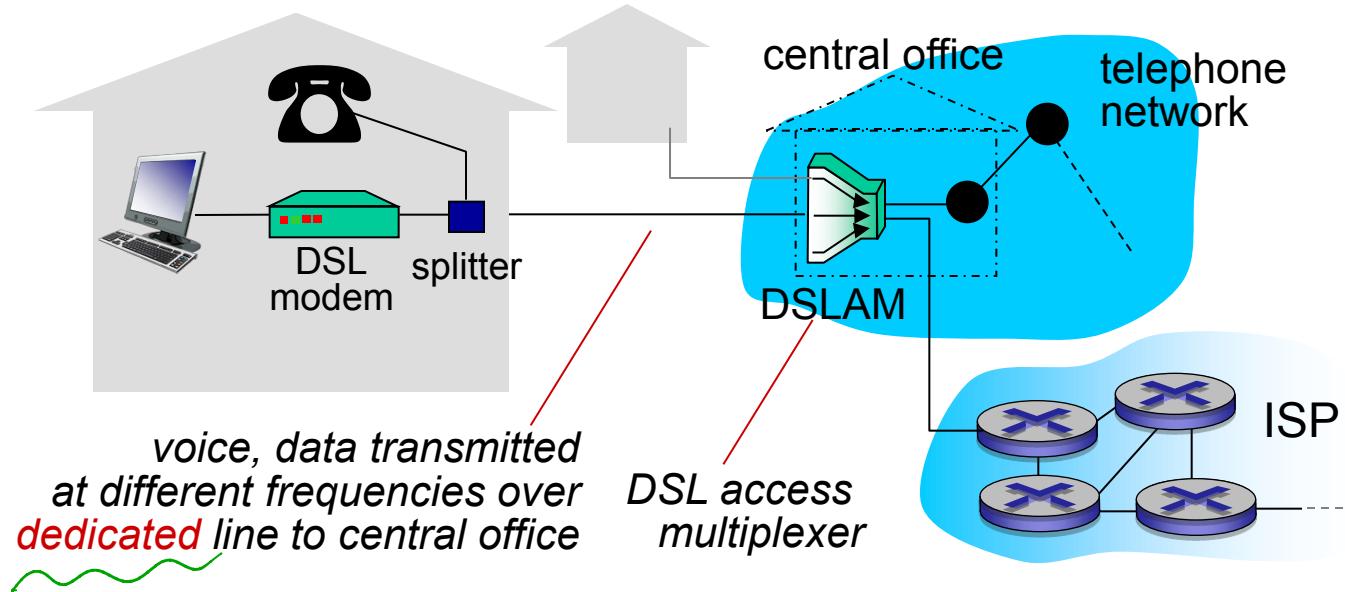
- **bandwidth (bits per second) of access network?**
- **shared or dedicated?**

the channel is  
shared to users

Token? Collision Free?



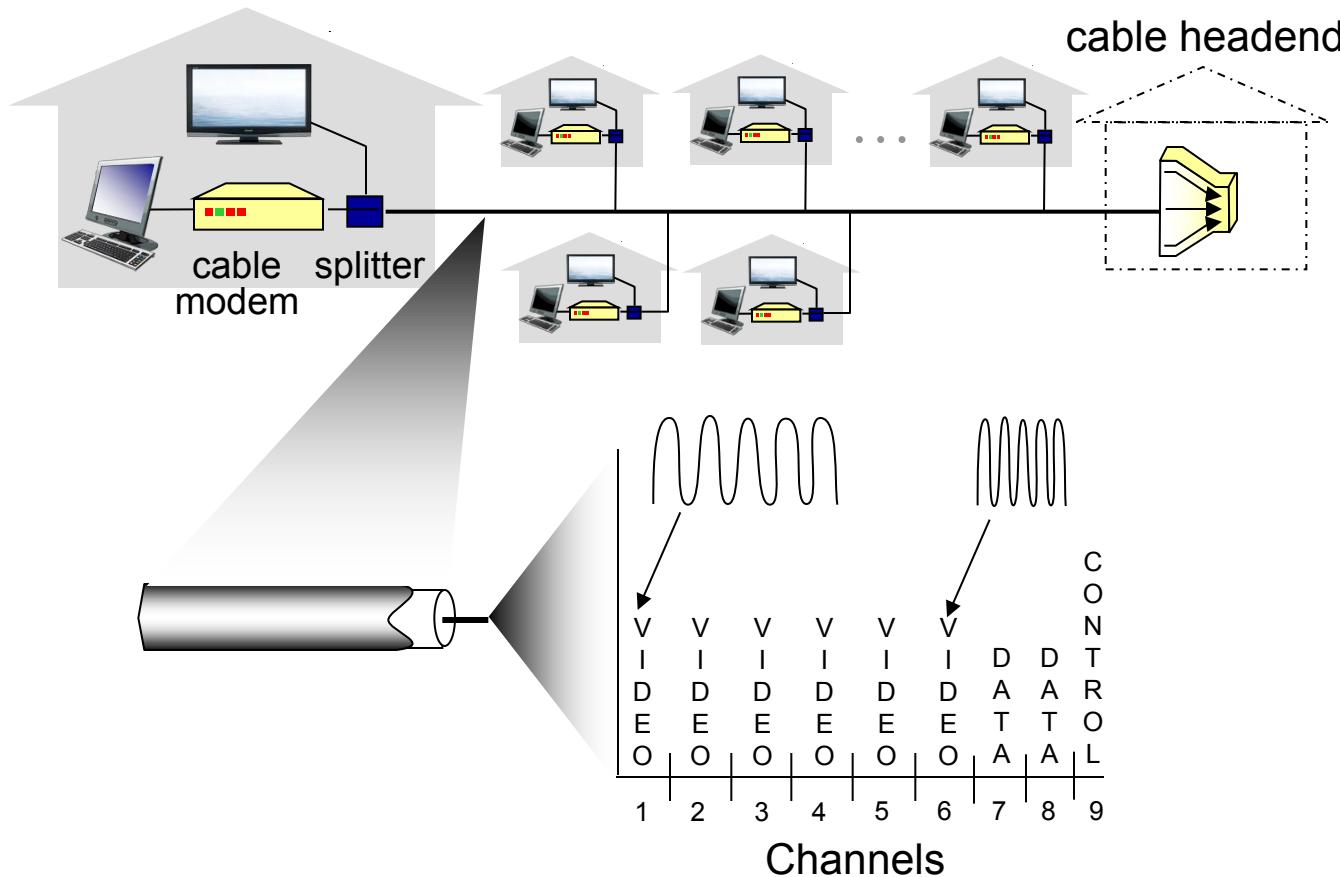
# Access network: digital subscriber line (DSL)



- 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)

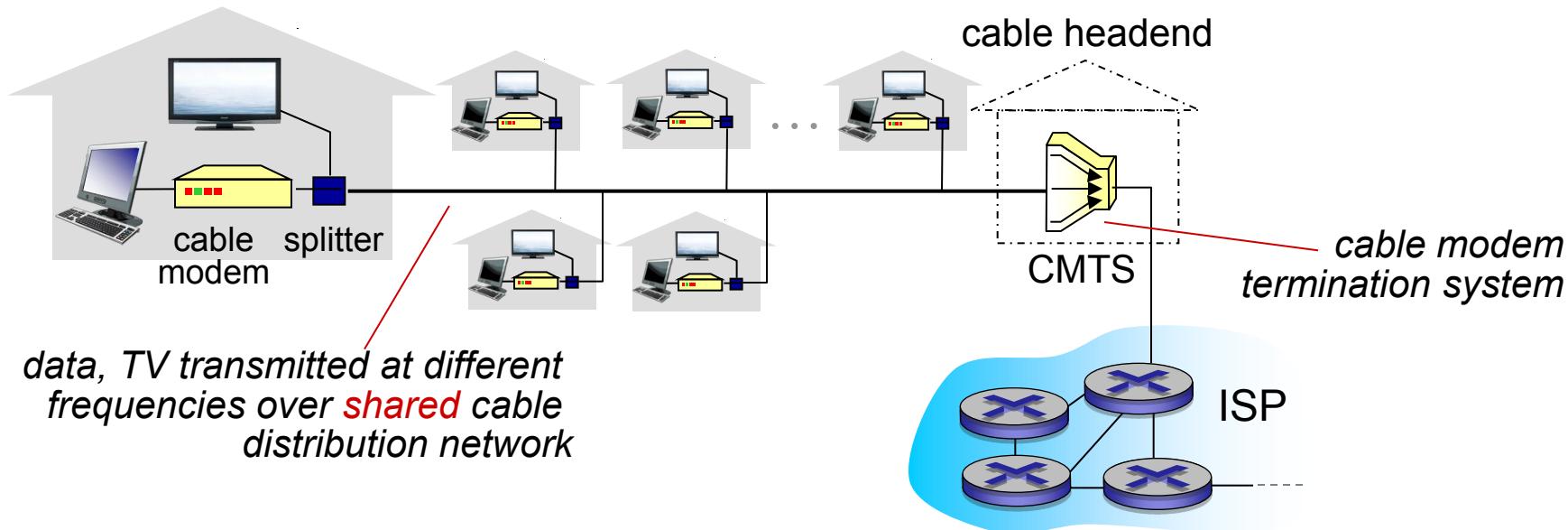
# Access network: cable network

Shared



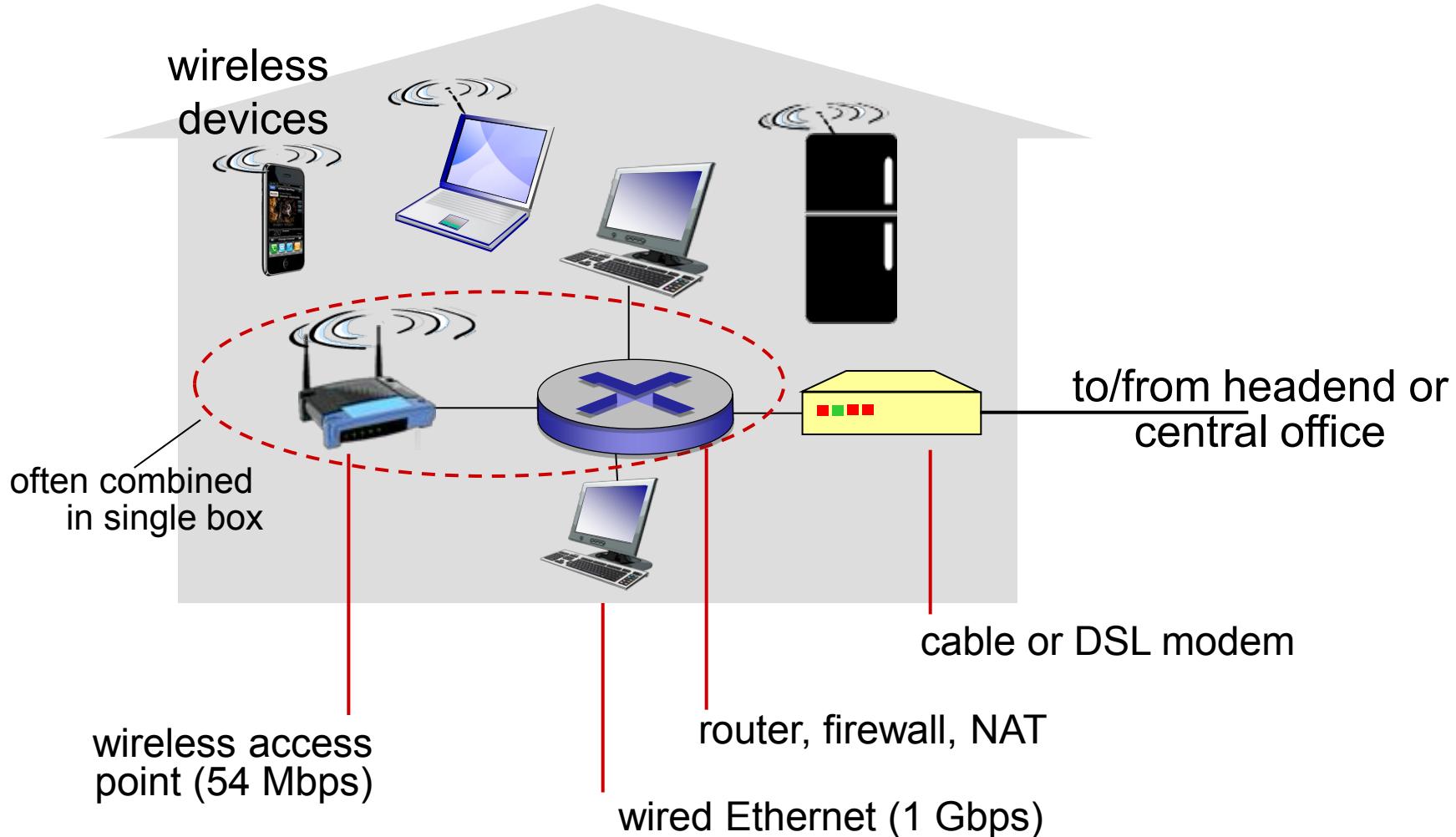
*frequency division multiplexing:* different channels transmitted in different frequency bands

# Access network: cable network

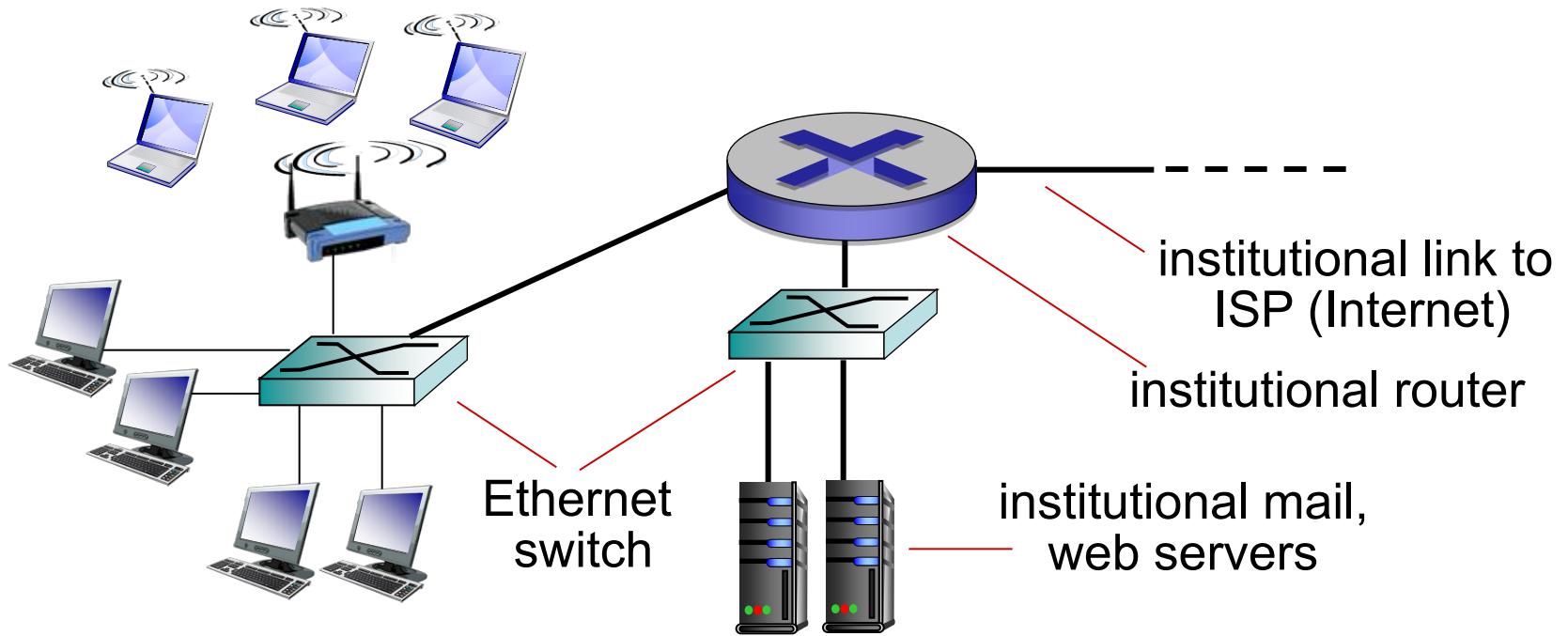


- HFC: hybrid fiber coax
  - asymmetric: up to 30Mbps downstream transmission rate, 2 Mbps upstream transmission rate
- network of cable, fiber attaches homes to ISP router
  - homes **share access network** to cable headend
  - unlike DSL, which has dedicated access to central office

# Access network: home network



# Enterprise access networks (Ethernet)



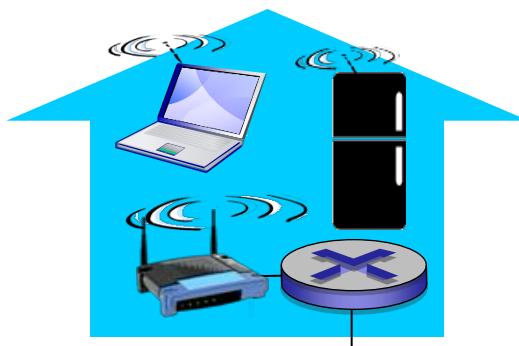
- typically used in companies, universities, etc.
- 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- today, end systems typically connect into Ethernet switch

# 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/n (WiFi): 11, 54, 450 Mbps transmission rate



*to Internet*

## wide-area wireless access

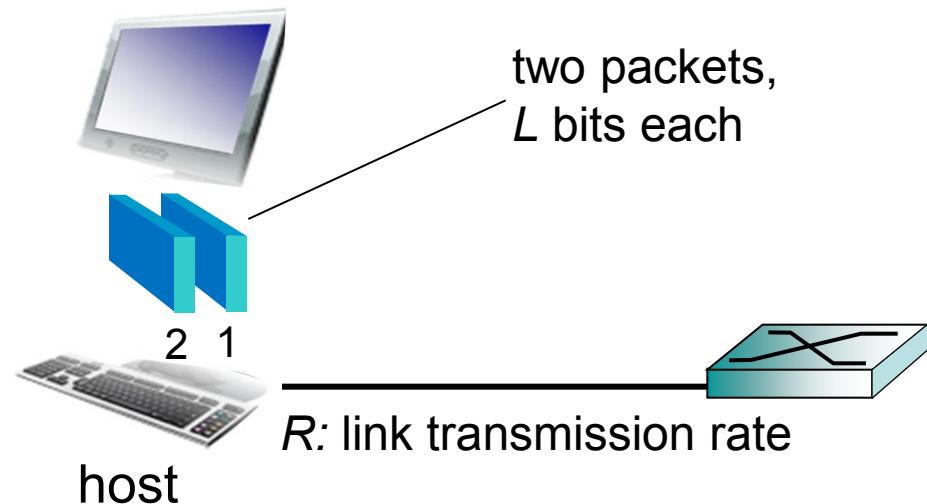
- provided by telco (cellular) operator, 10's km
- between 1 and 10 Mbps
- 3G, 4G: LTE



# 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*



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

# 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
  - 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)
  - 54 Mbps
- wide-area (e.g., cellular)
  - 4G cellular: ~ 10 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

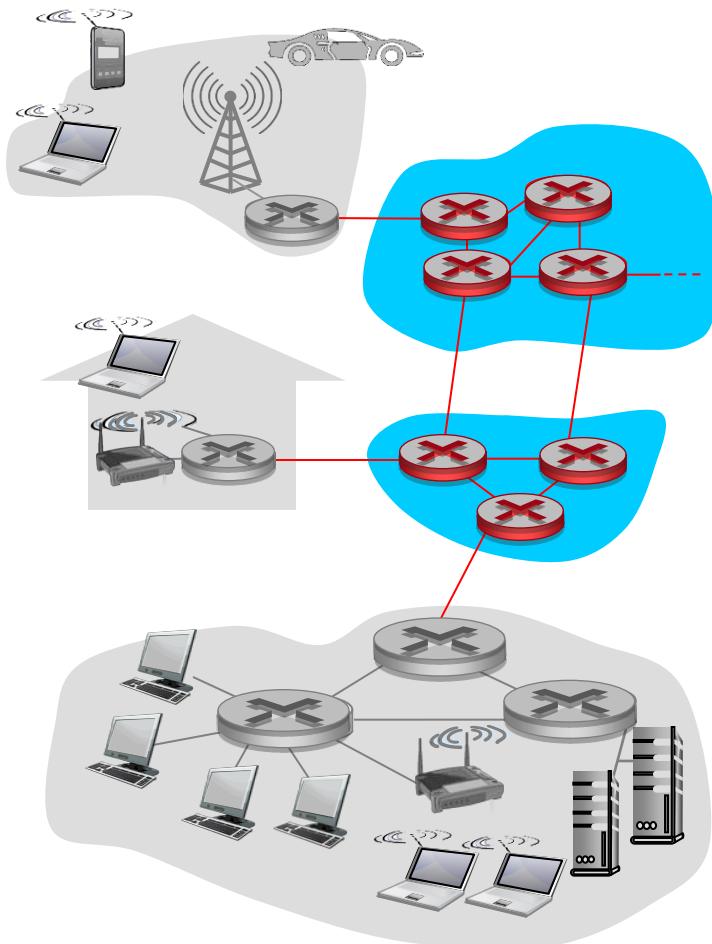
I.5 protocol layers, service models

I.6 networks under attack: security

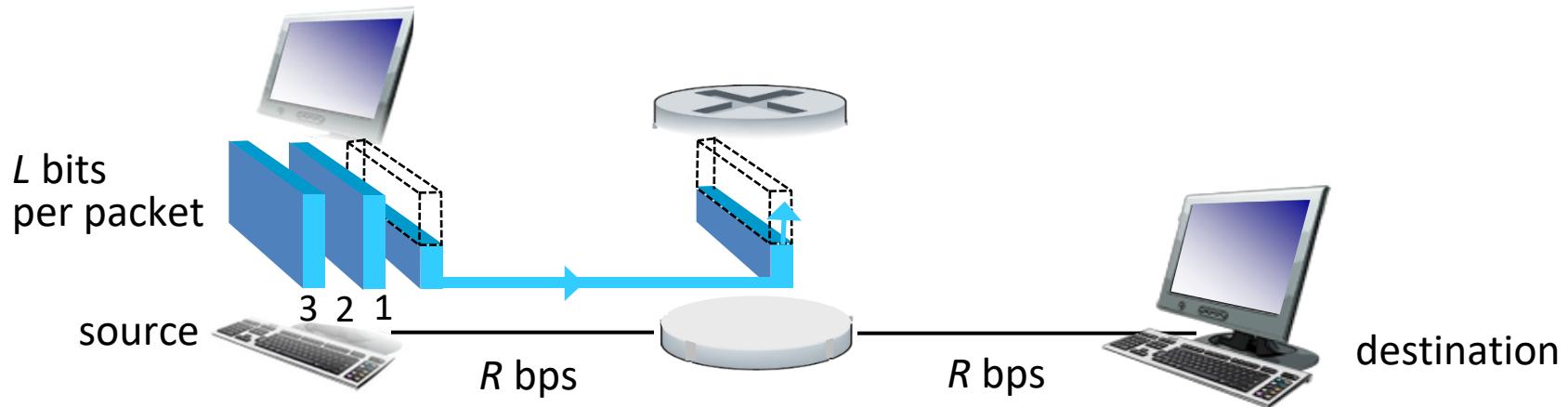
I.7 history

# 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**



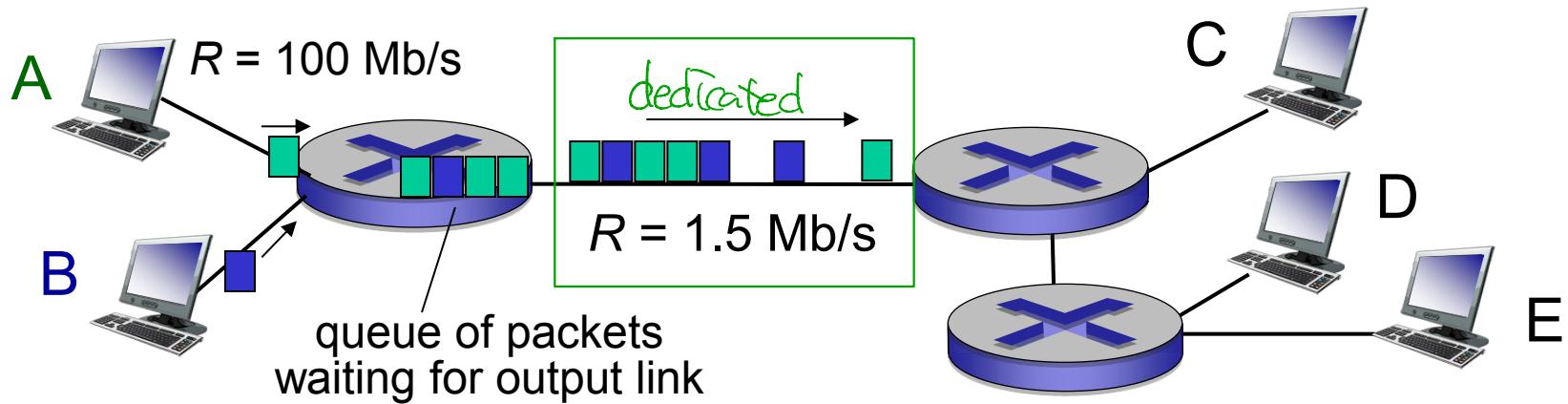
# Packet-switching: store-and-forward



- 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:*
- $L = 7.5 \text{ Mbits}$
  - $R = 1.5 \text{ Mbps}$
  - one-hop transmission delay = 5 sec
- } more on delay shortly ...

# Packet Switching: queueing delay, loss



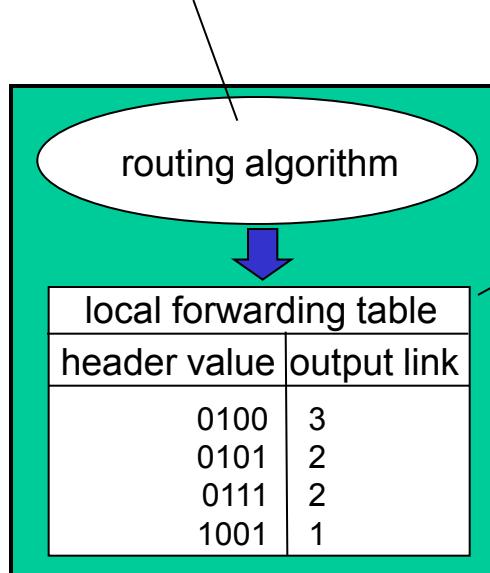
## queueing and loss:

- 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

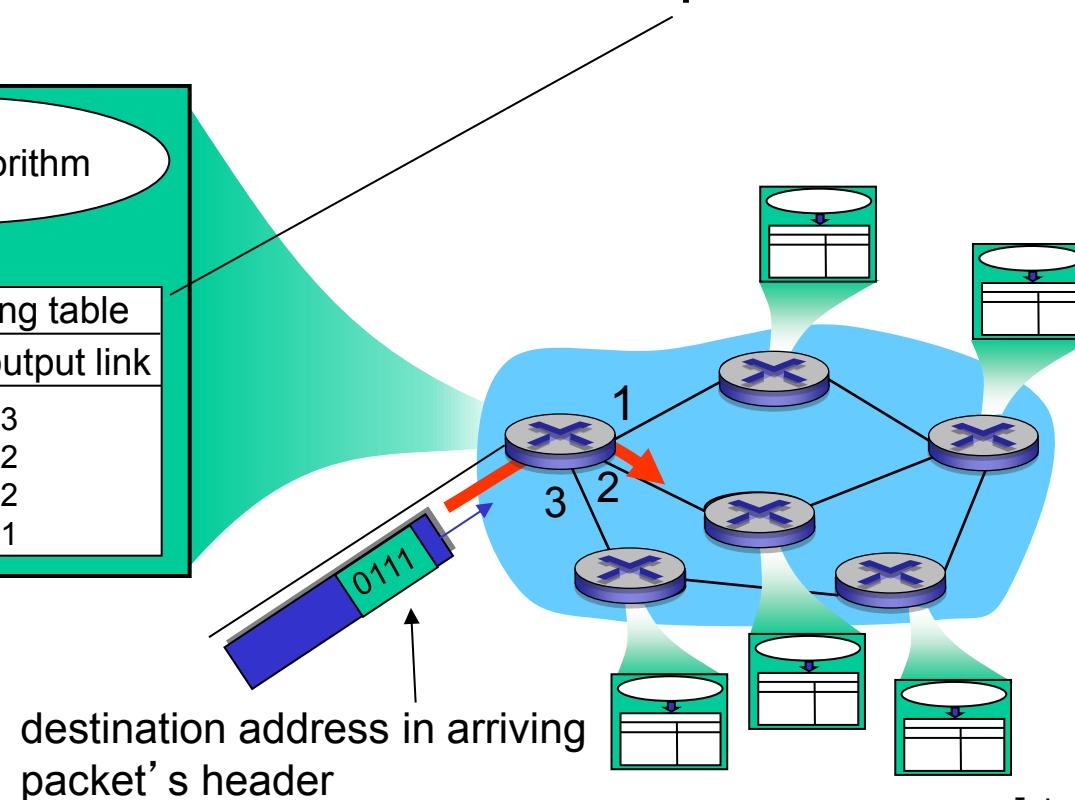
# Two key network-core functions

**routing:** determines source-destination route taken by packets

- *routing algorithms*



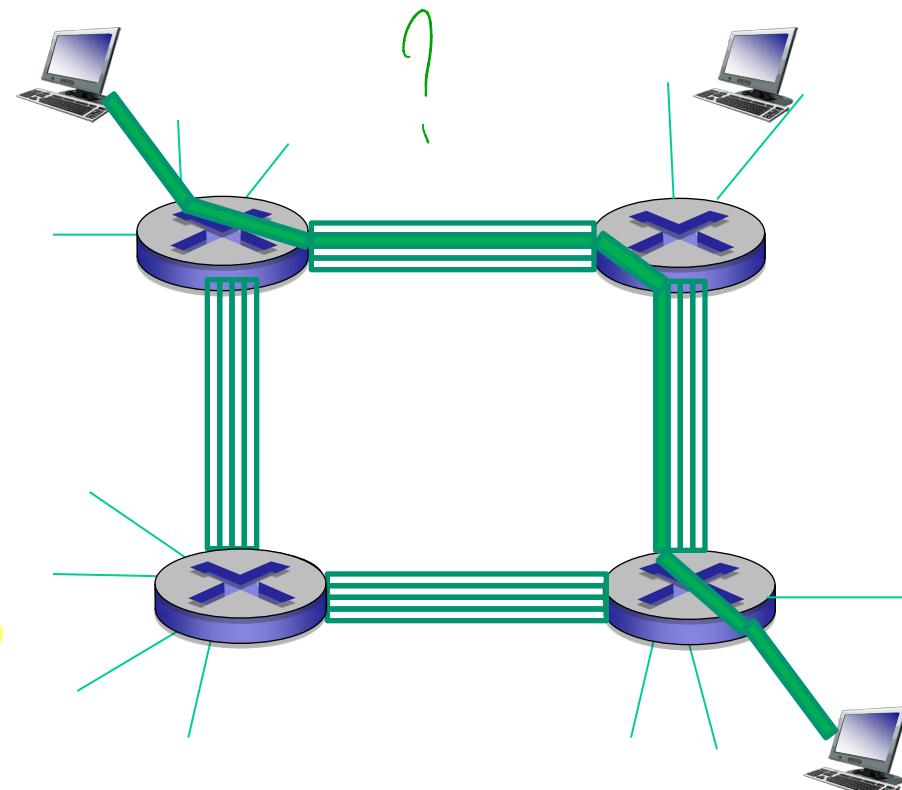
**forwarding:** move packets from router's input to appropriate router output



# Alternative core: circuit switching

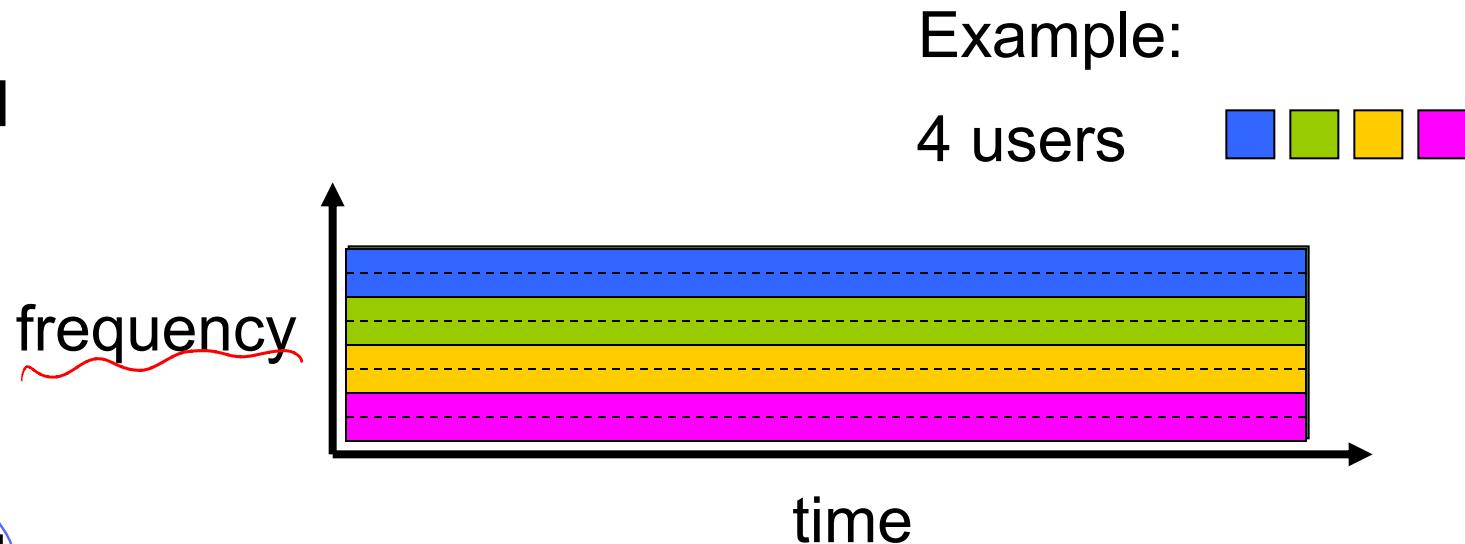
end-end resources allocated  
to, reserved for “call”  
between source & dest:

- in diagram, each link has four circuits.
  - call gets 2<sup>nd</sup> circuit in top link and 1<sup>st</sup> 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

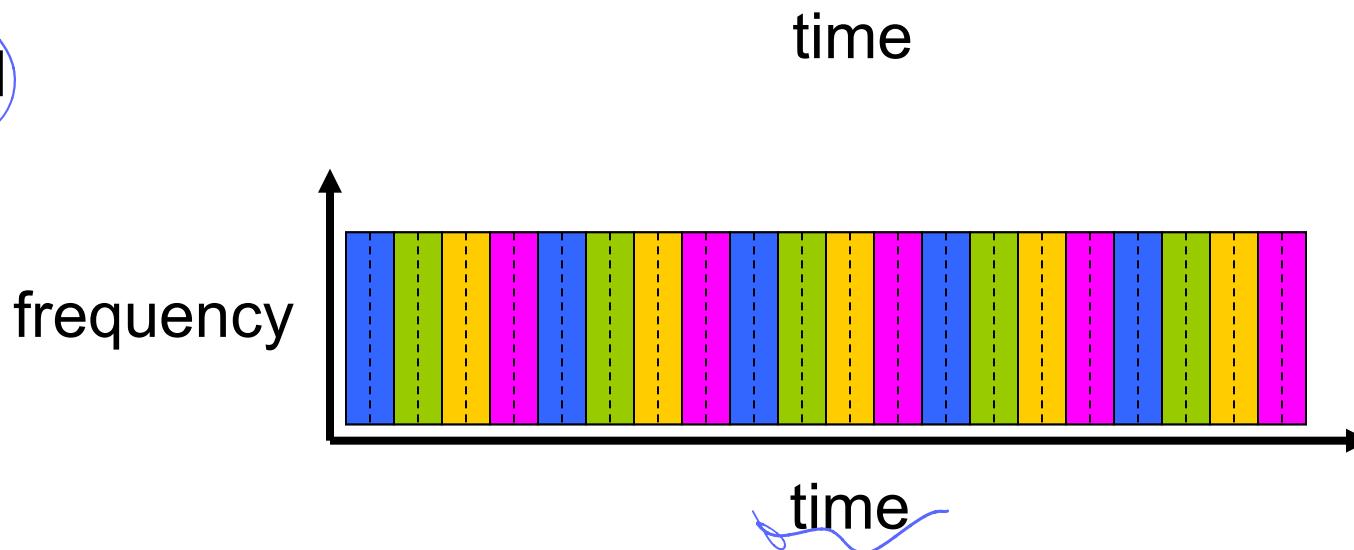


# Circuit switching: FDM versus TDM

FDM



TDM



# Packet switching versus circuit switching

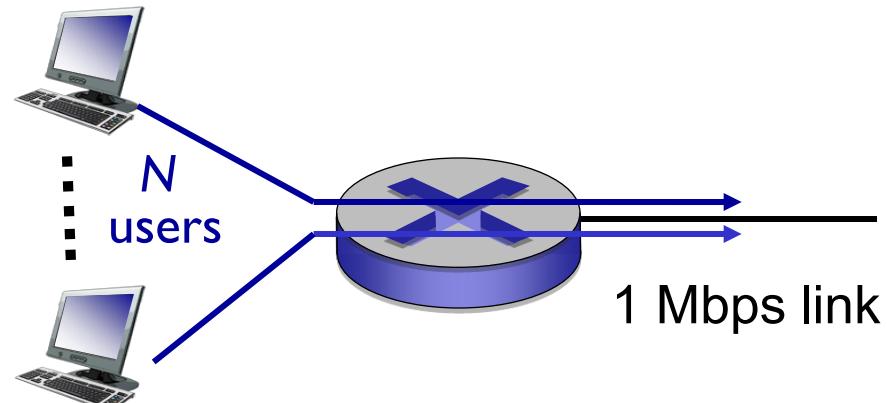
*packet switching allows more users to use network!*

example:

- 1 Mb/s link
- each user:

- 100 kb/s when “active”
  - active **10% of time**
- 

- **circuit-switching:**
  - 10 users  $1M/10 = 100k$
- **packet switching:** low cost.
  - with 35 users, **probability** > 10 active at same time is less than .0004 \*  $\sum_{i=1}^{35} {}^i C_3 (0.1)^i (1-0.1)^{35-i}$ : the probability that more than 11 users are active.



Q: how did we get value 0.0004?

Q: what happens if > 35 users ?

handle with buffer  
more user, larger buffer

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

$$\sum_{i=1}^{35} {}^i C_3 (0.1)^i (1-0.1)^{35-i}$$
 : the probability that more than 11 users are active.

# 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)

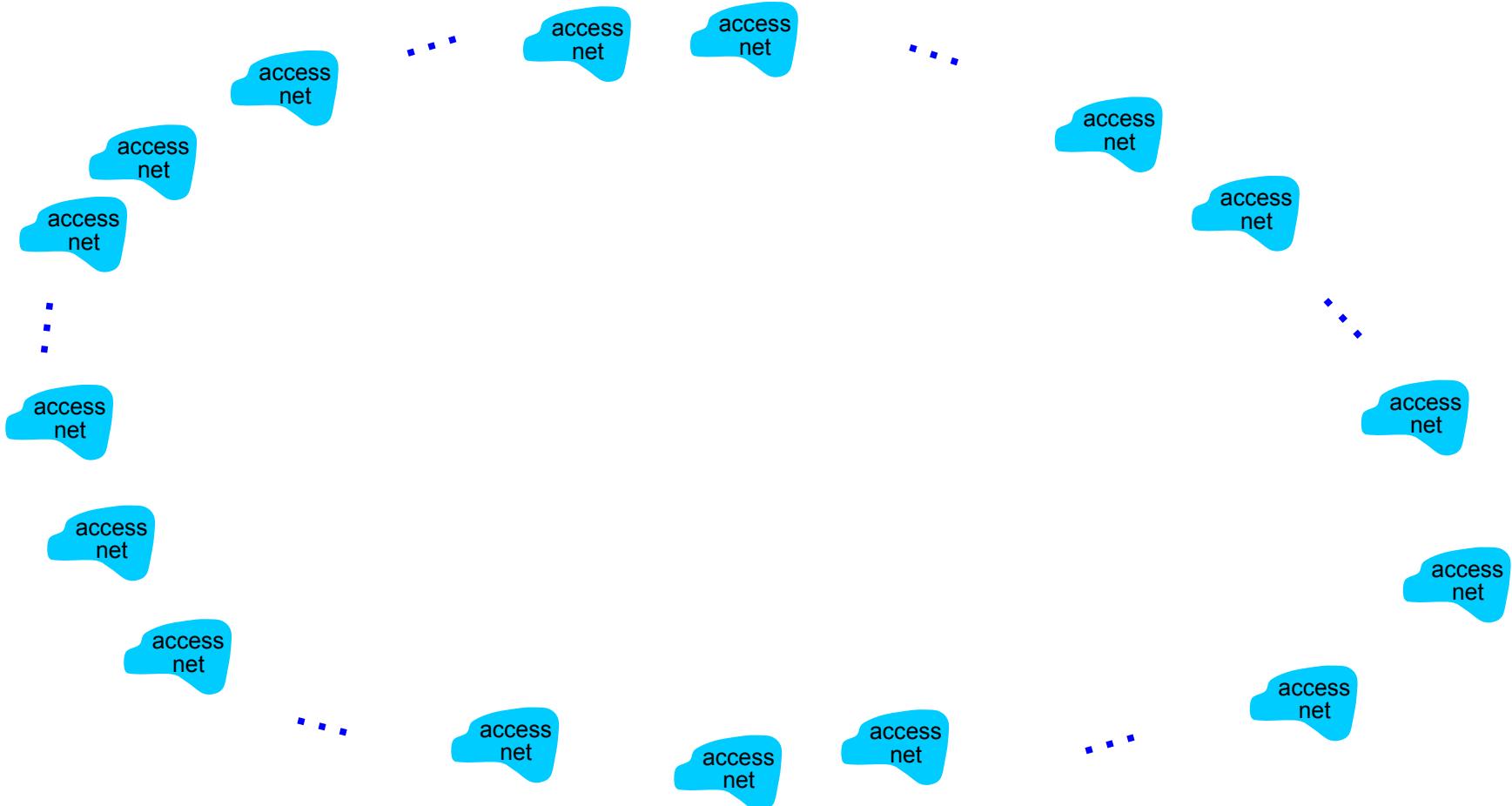
Q: human analogies of reserved resources (circuit switching)  
versus on-demand allocation (packet-switching)?

# 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

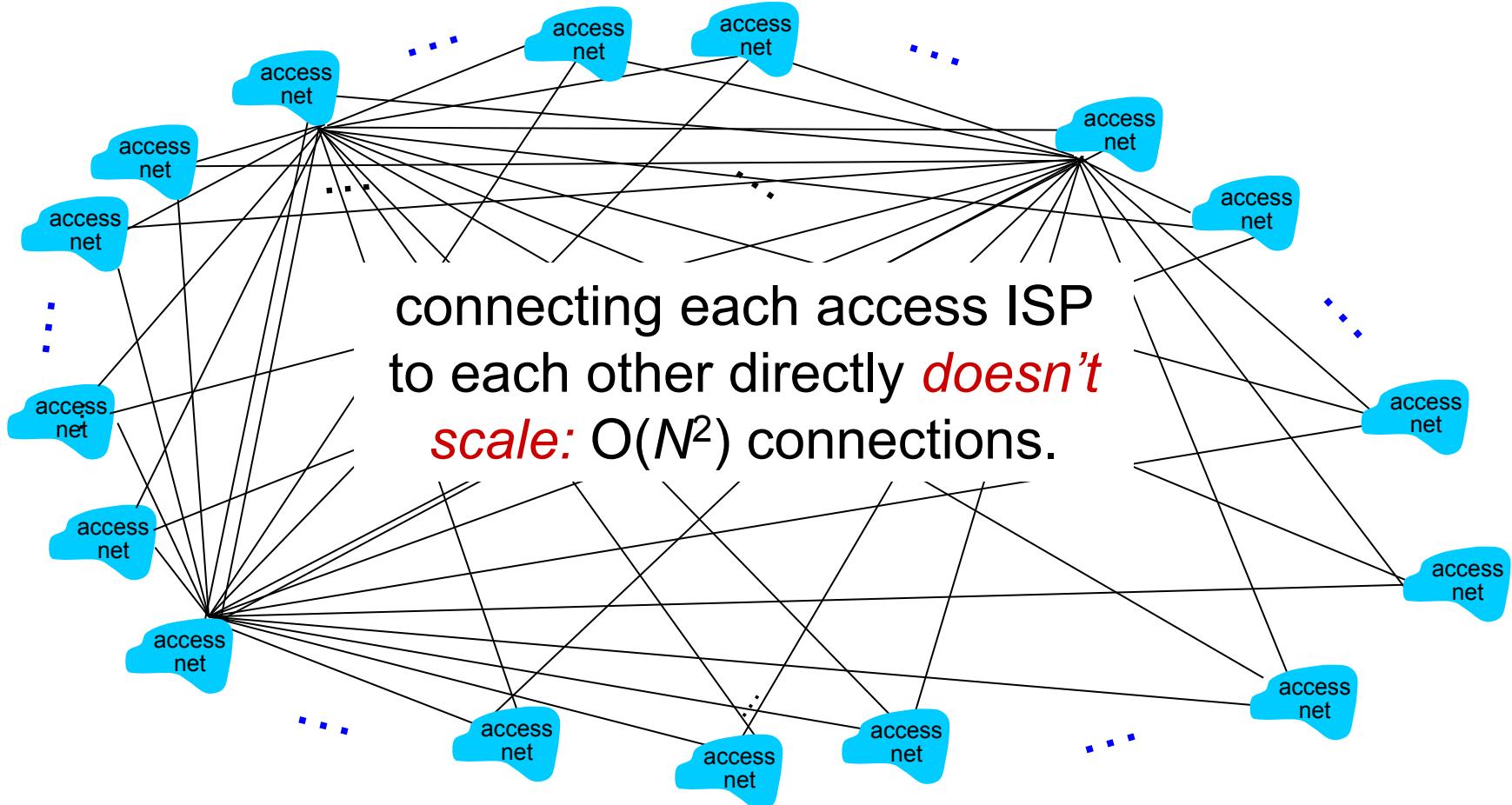
# Internet structure: network of networks

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



# Internet structure: network of networks

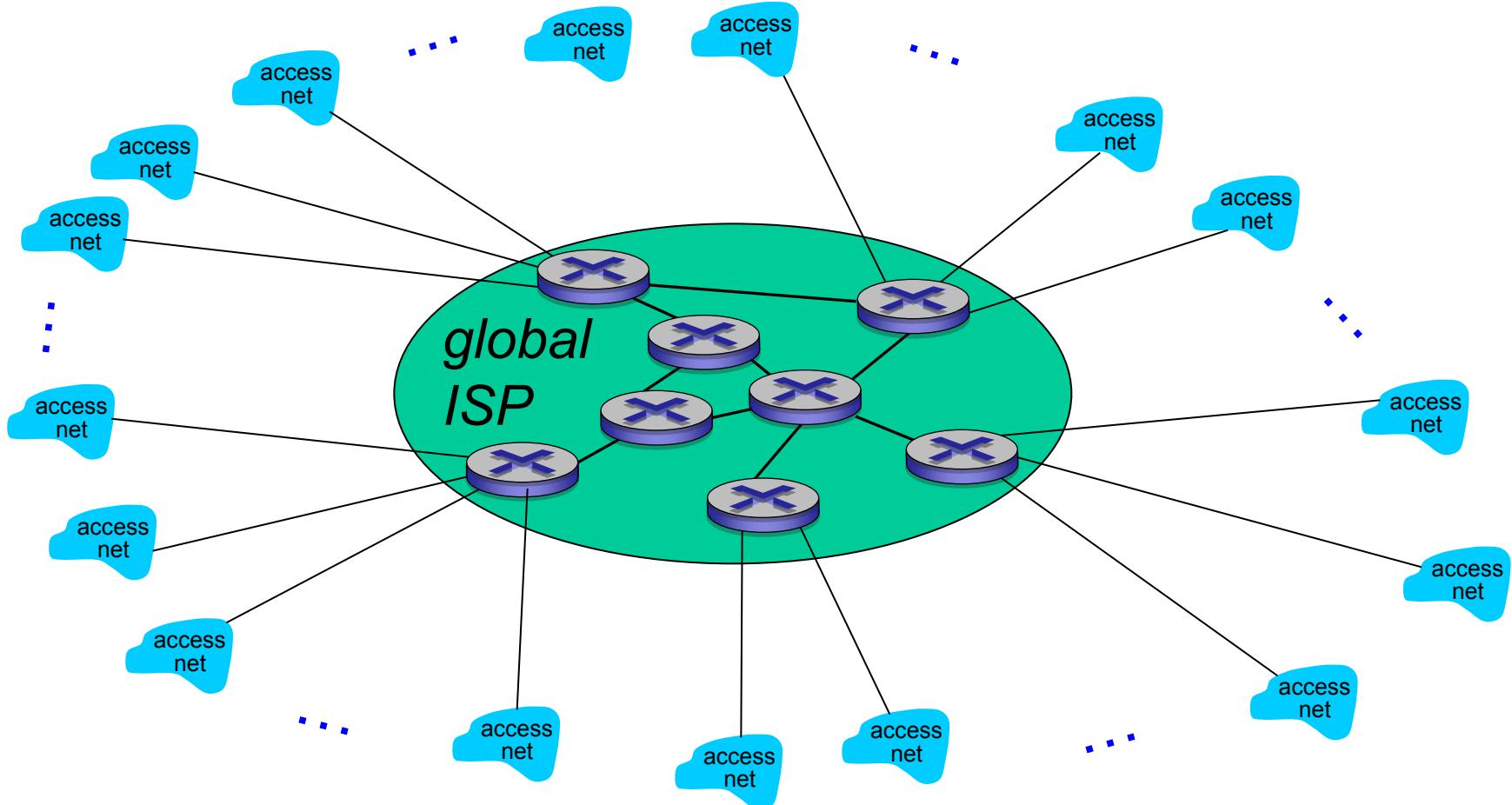
*Option:* connect each access ISP to every other access ISP?



# Internet structure: network of networks

*Option:* connect each access ISP to one *global transit ISP*?

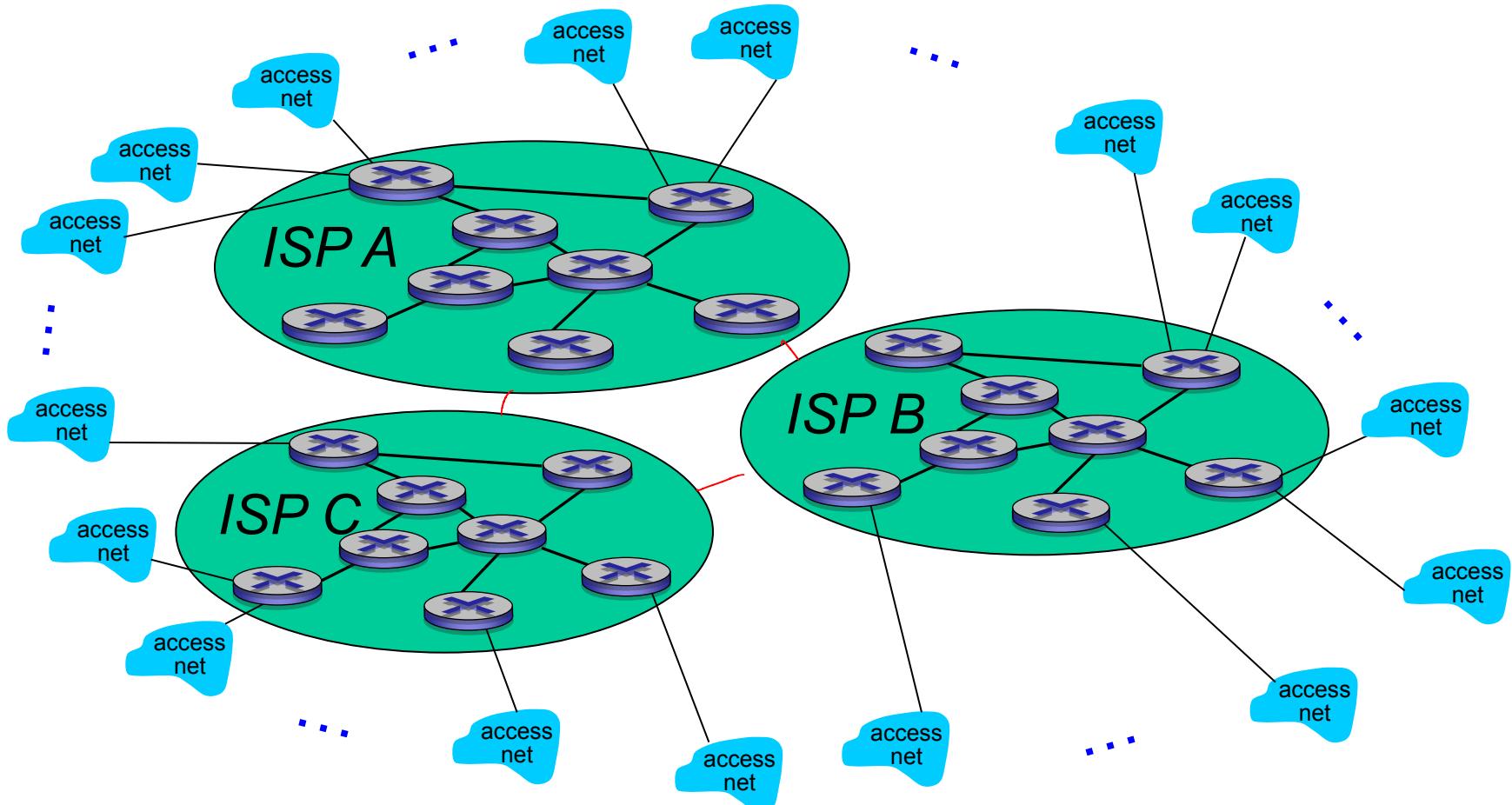
*Customer and provider ISPs have economic agreement.*



# Internet structure: network of networks

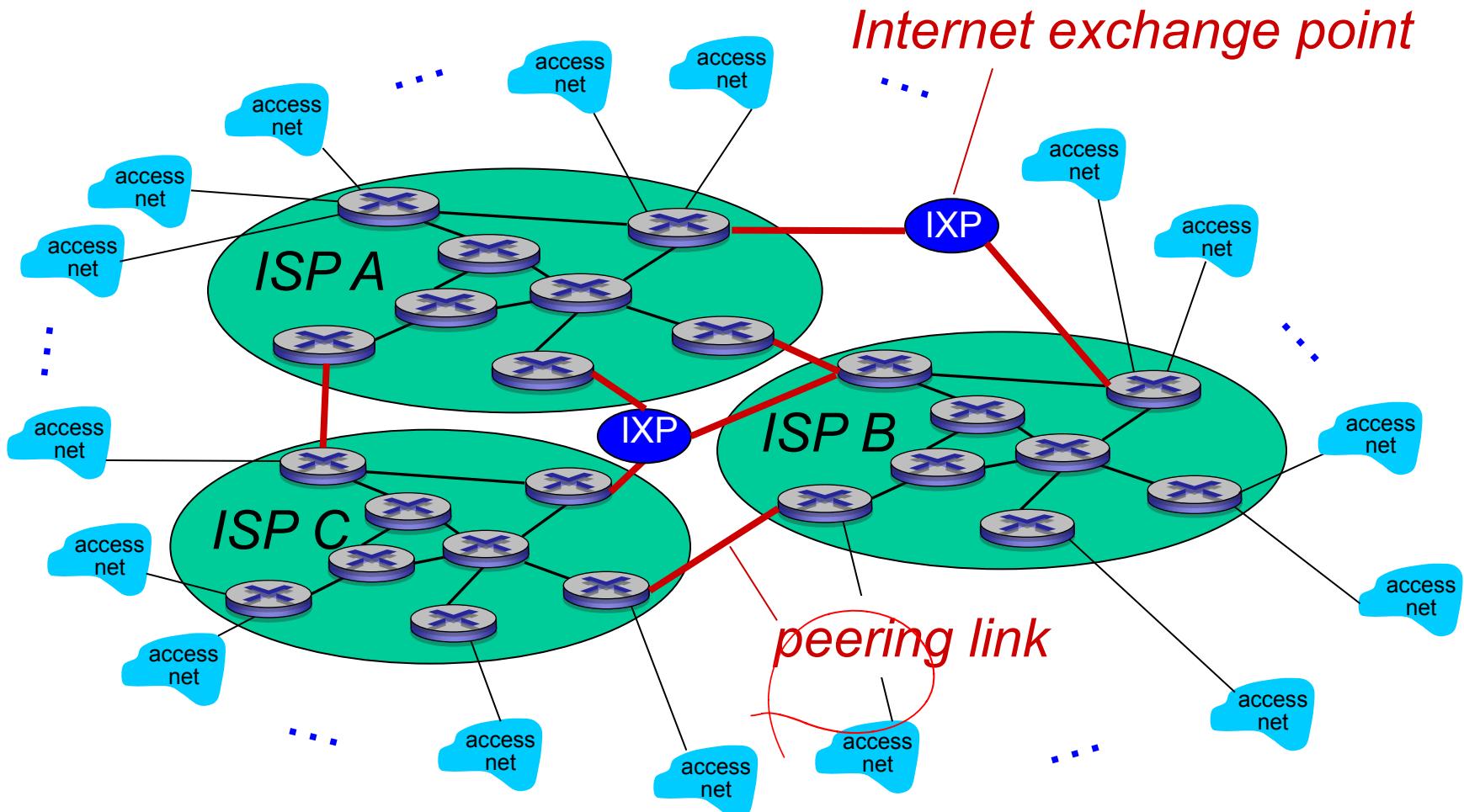
But if one global ISP is viable business, there will be competitors

....



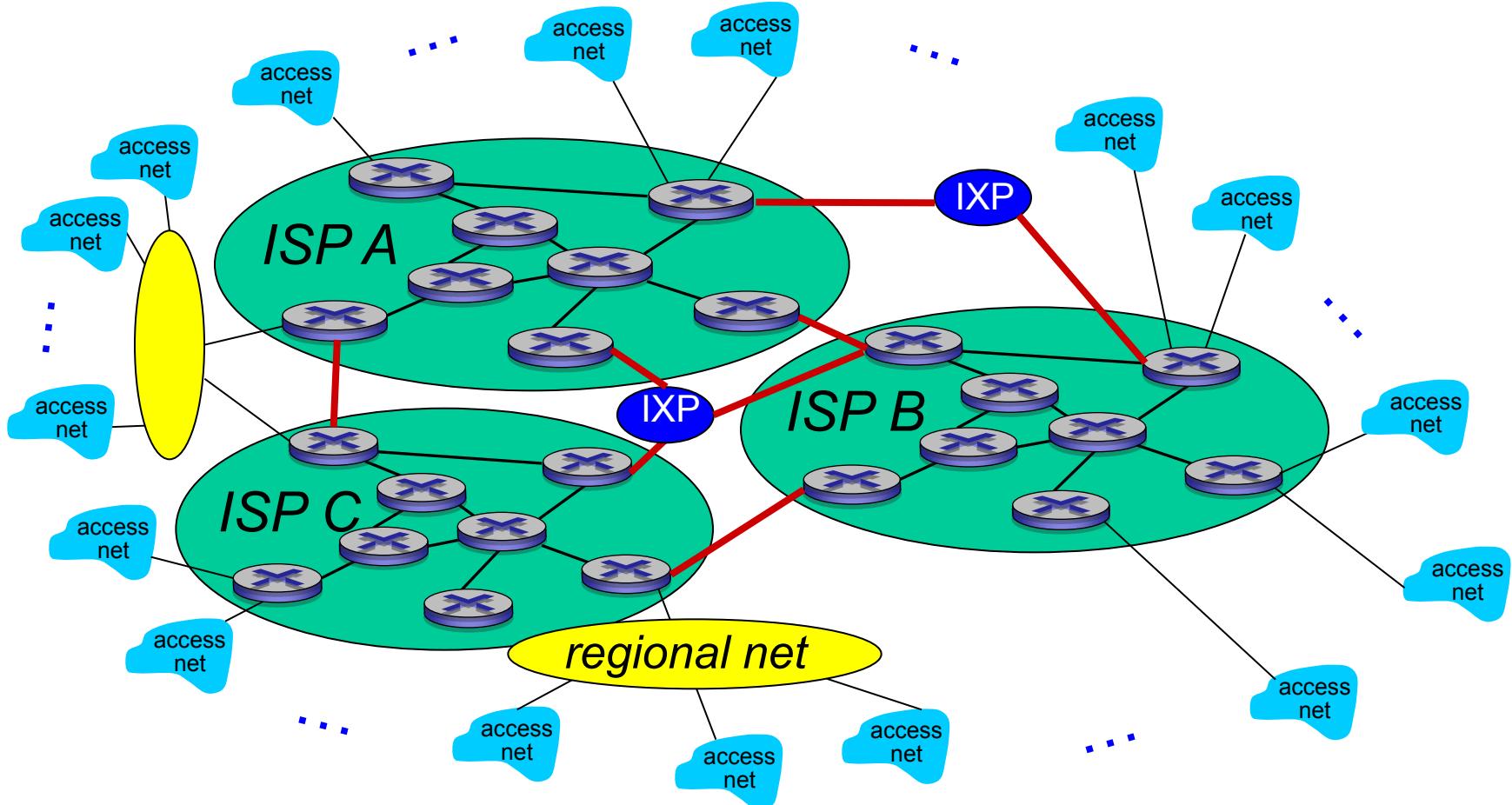
# Internet structure: network of networks

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



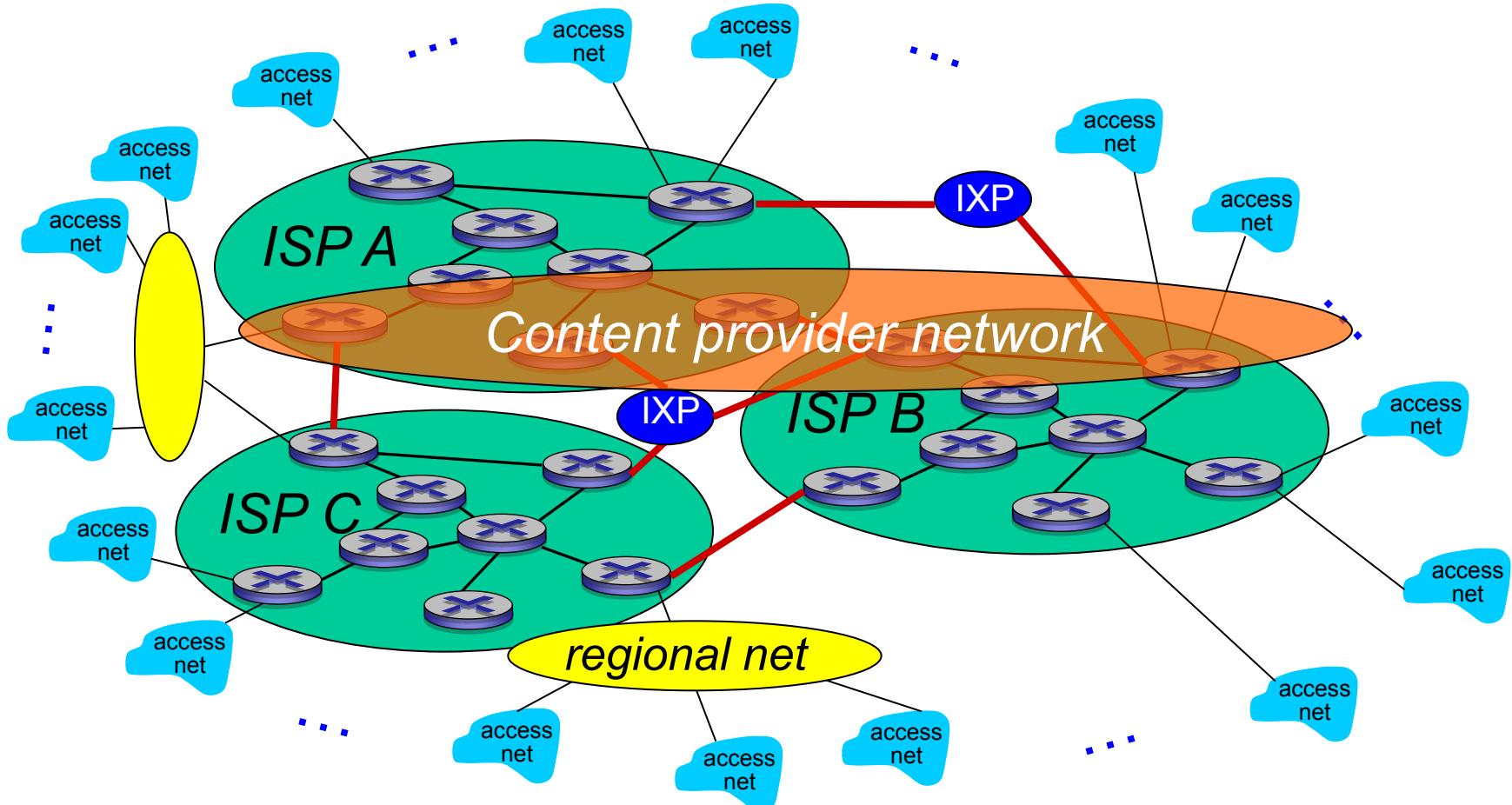
# Internet structure: network of networks

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

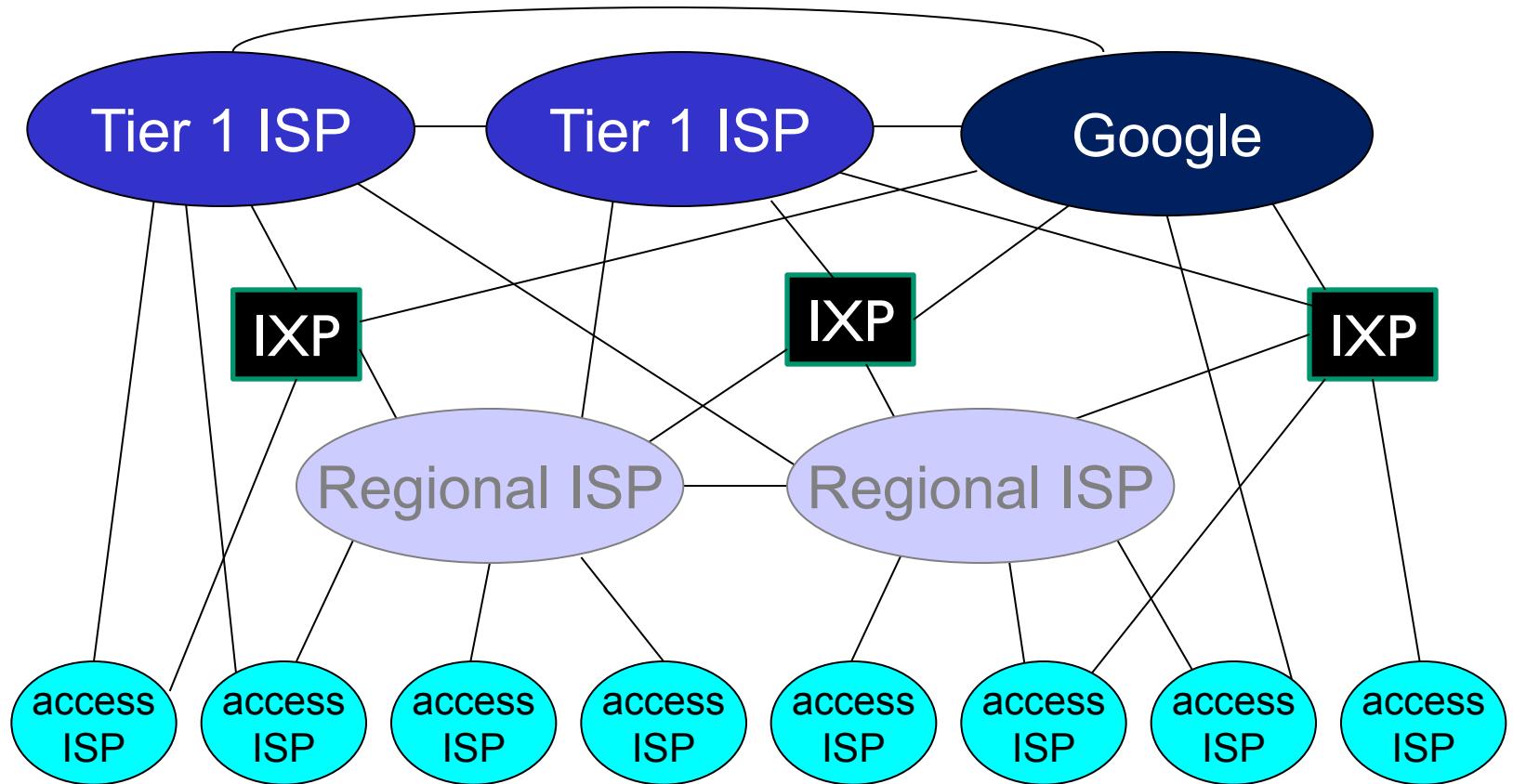


# Internet structure: network of networks

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

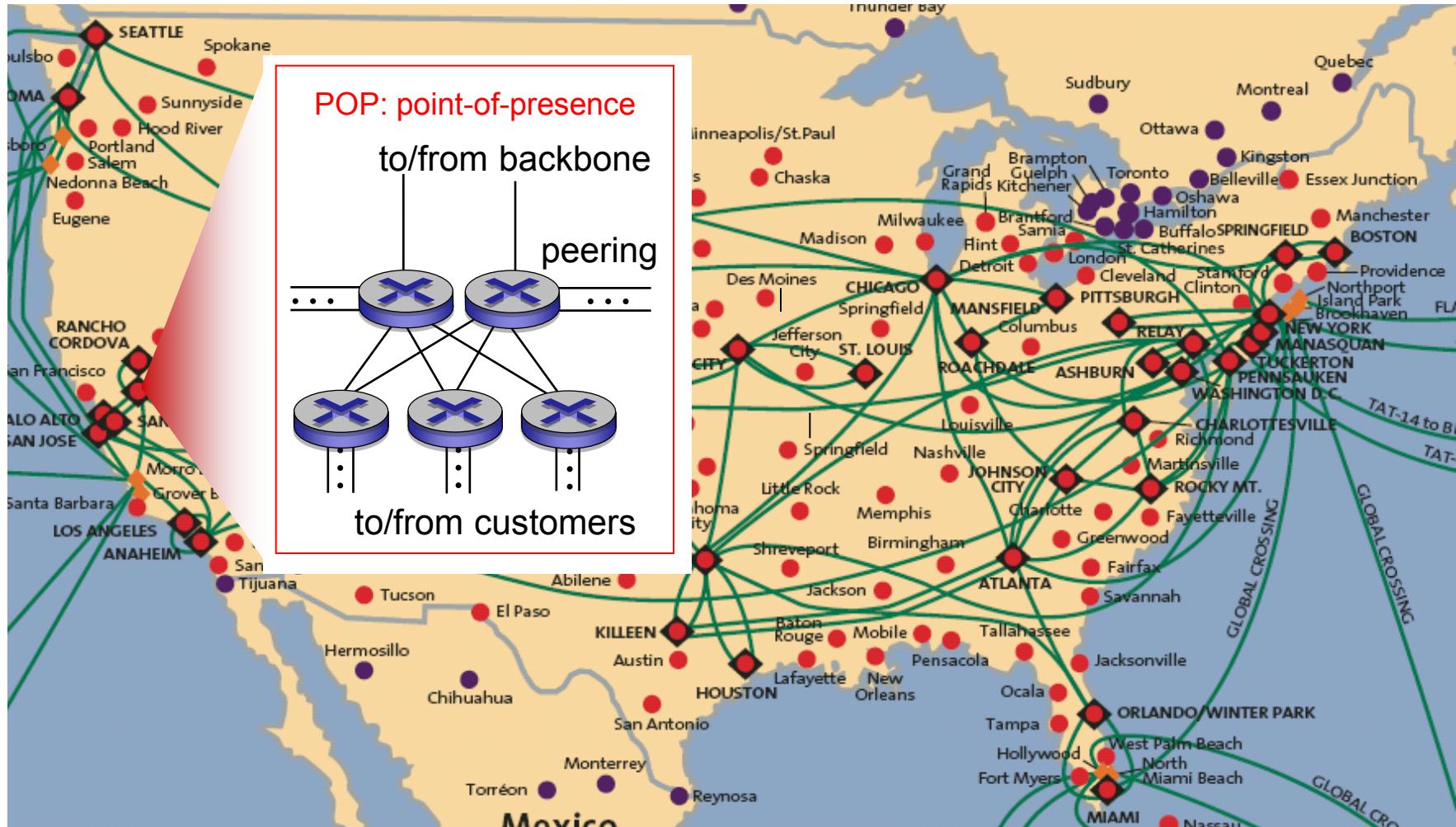


# Internet structure: network of networks



- 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

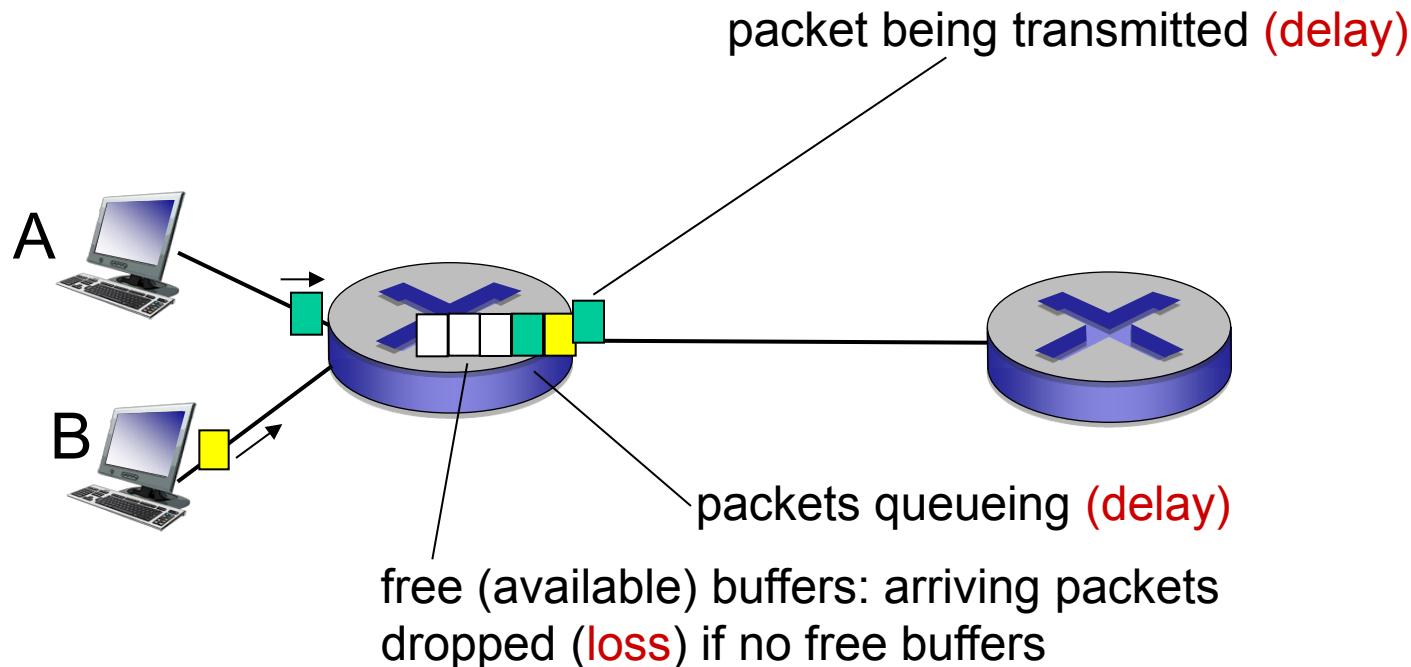
I.7 history

# How do loss and delay occur?

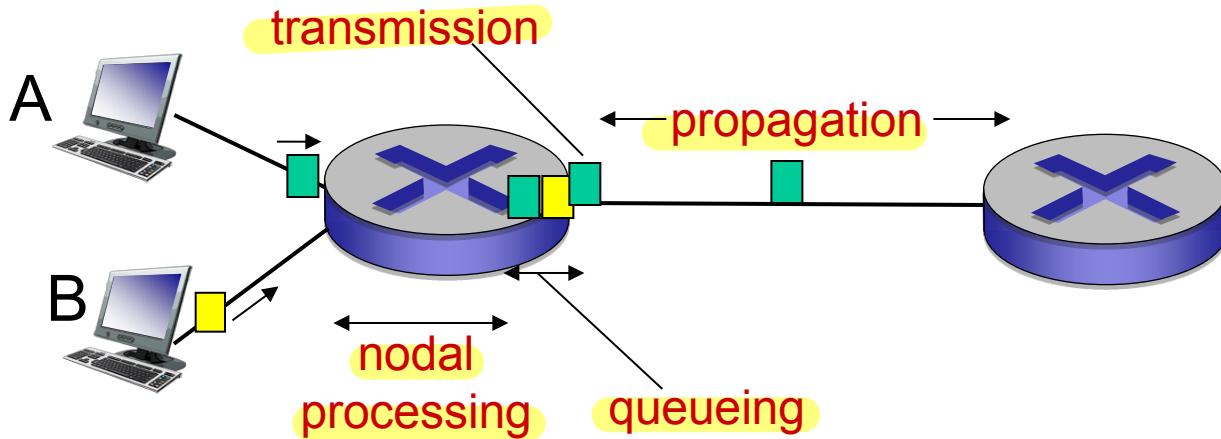
active 100Mbps	link 1Mbps
10%	

packets **queue** in router **buffers**

- packet arrival rate to link (temporarily) exceeds output link capacity 1.5Mbps 1Mbps
- packets queue, wait for turn



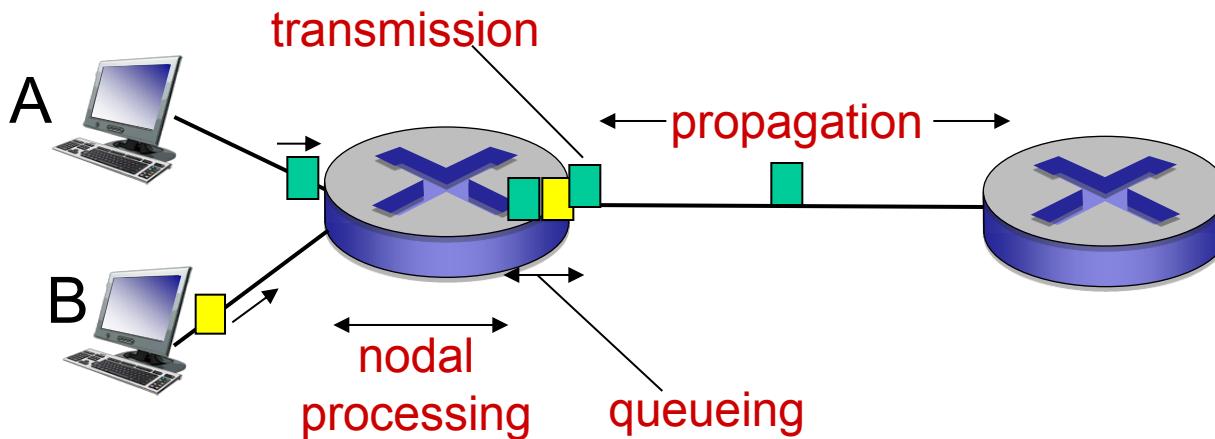
# Four sources of packet delay



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

- ①  $d_{\text{proc}}$ : nodal processing
- check bit errors error control
  - determine output link routing
  - typically < msec
- ②  $d_{\text{queue}}$ : queueing delay
- time waiting at output link for transmission
  - depends on congestion level of router

# Four sources of packet delay



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

③  $d_{\text{trans}}$ : transmission delay:

▪ L: packet length (bits) 1kb

▪ R: link bandwidth (bps) 1Mbps

▪  $d_{\text{trans}} = L/R$  ←  $d_{\text{trans}}$  and  $d_{\text{prop}}$  →  
 $10^3/10^6 = 10^{-3}$  very different

④  $d_{\text{prop}}$ : propagation delay:

▪ d: length of physical link

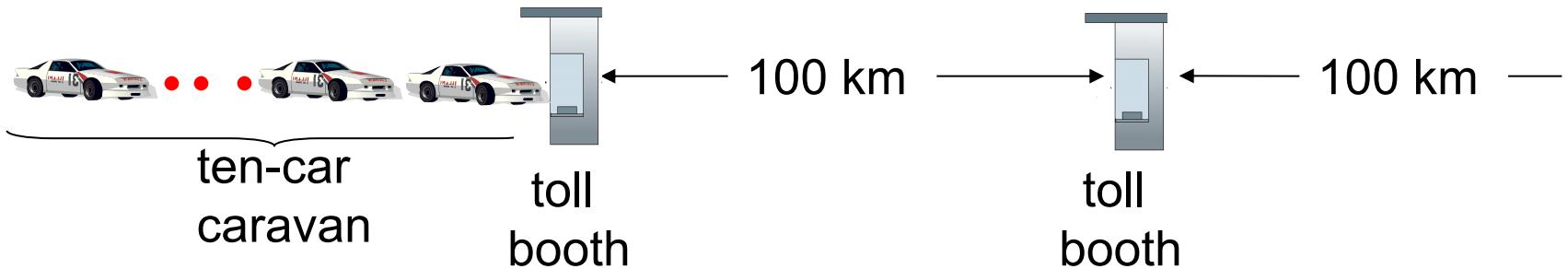
▪ s: propagation speed ( $\sim 2 \times 10^8$  m/sec)

▪  $d_{\text{prop}} = d/s$

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

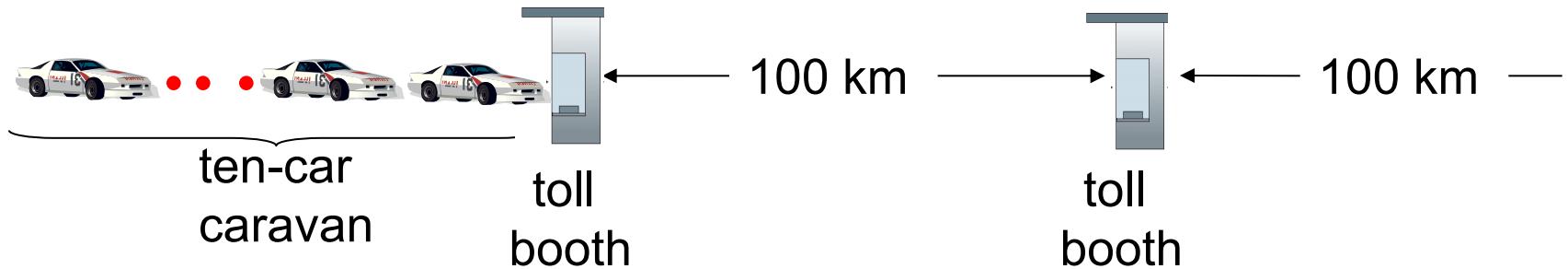
\* Check out the Java applet for an interactive animation on trans vs. prop delay

# 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 booth:  
 $100\text{km}/(100\text{km/hr})= 1\text{ hr}$
- 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, first car arrives at second booth; three cars still at first booth → ↗  
error control ↗



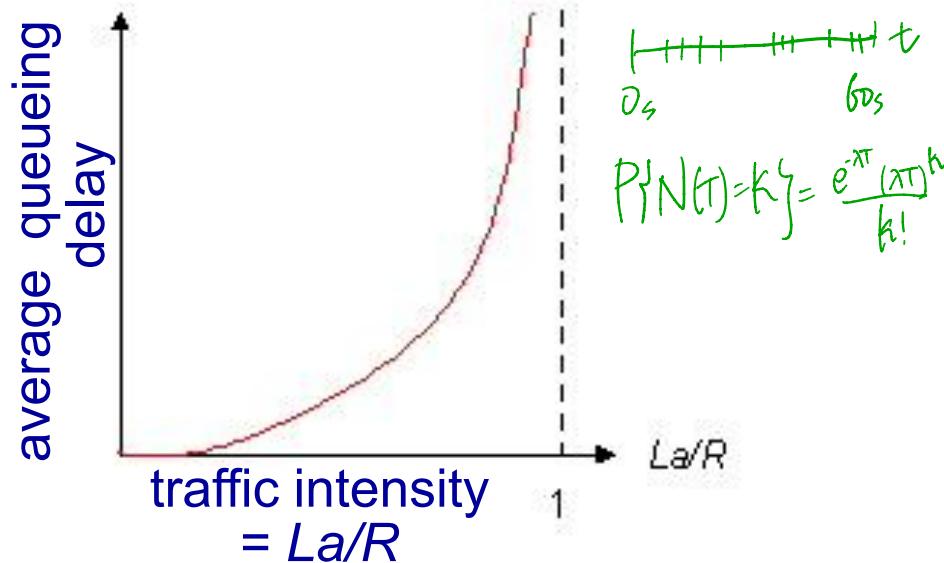
generalized processor sharing  $\Rightarrow W$  is bounded.

# Queueing delay (revisited)

Poisson Process in arrival packets

10 packets/min  $\Rightarrow$  randomly distributed.

- $R$ : 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!



$La/R \sim 0$



$La/R \rightarrow 1$

traffic intensity.  $\rho = \frac{\lambda}{\mu} = \frac{\text{average arrival rate}}{\text{service rate}}$

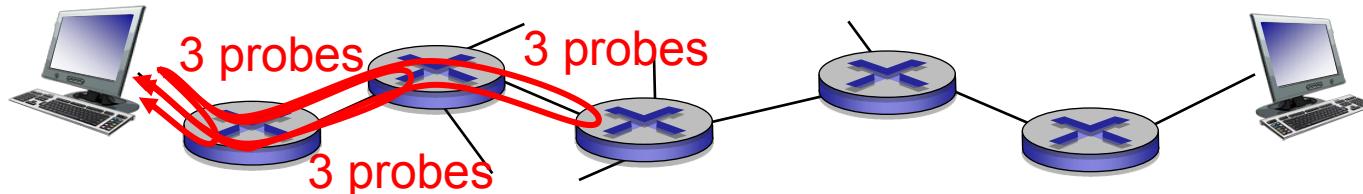
\* Check online interactive animation on queuing and loss

$$\tilde{W}_q = \frac{\rho}{1-\rho} \quad \begin{array}{c} \omega \\ \diagup \quad \diagdown \\ \rho \end{array}$$

wait time  
in queue

# “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 error msg 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](http://www.traceroute.org)

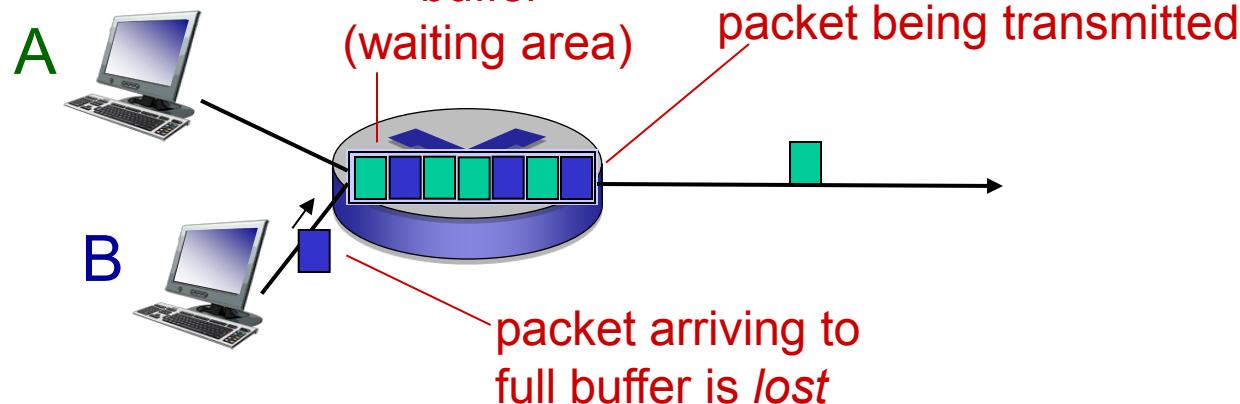
# Packet loss

- 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**

TCP  $\Rightarrow$  how can retransmission is possible  
Sequence number  $\Rightarrow$  find lost packet

TCP or IP

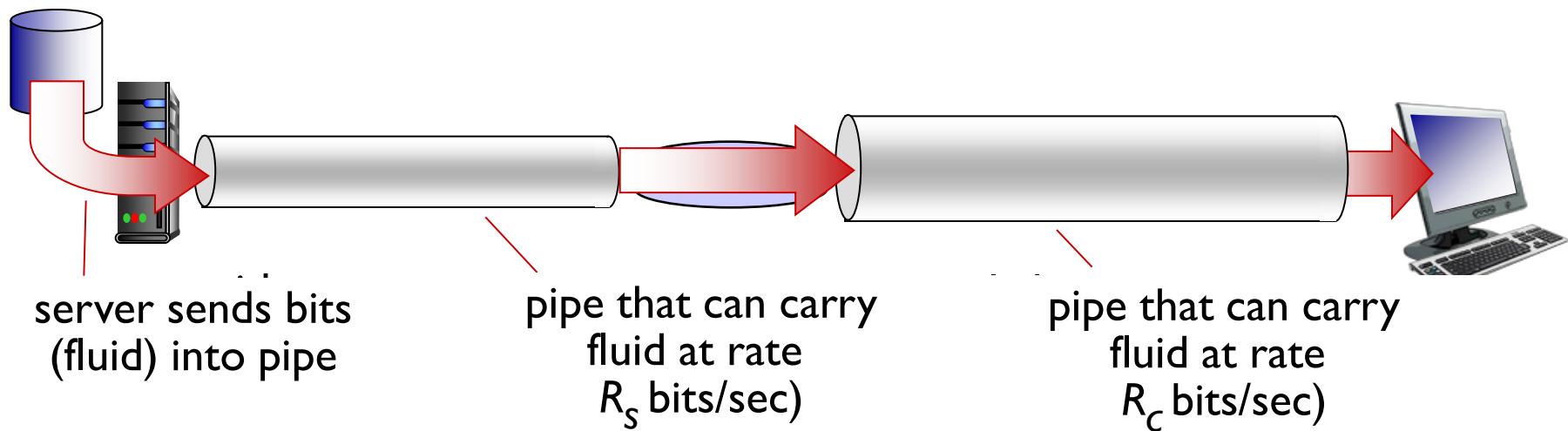
UDP  $\Rightarrow$  No retransmission  
real time delivery



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

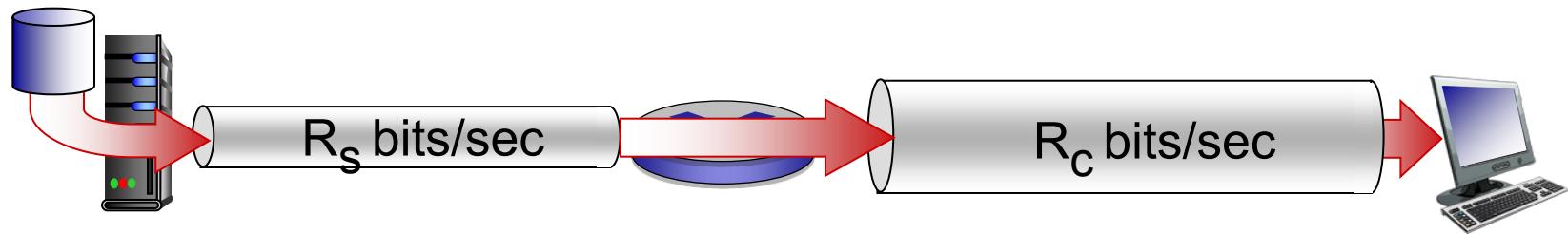
# Throughput

- **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

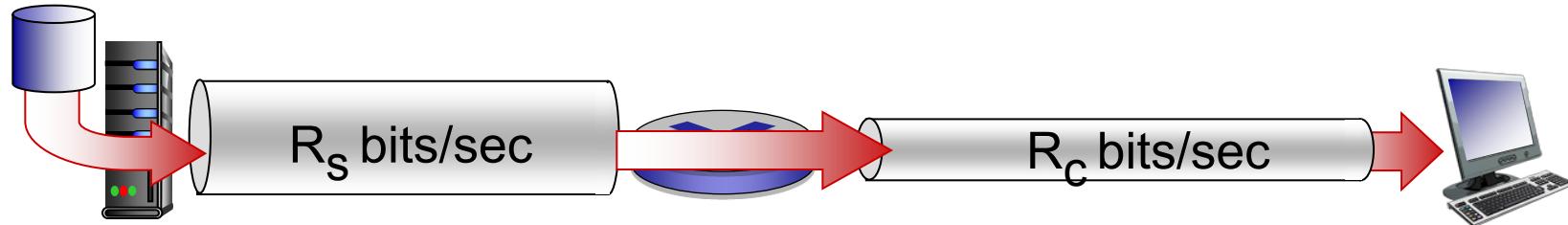


# Throughput (more)

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



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

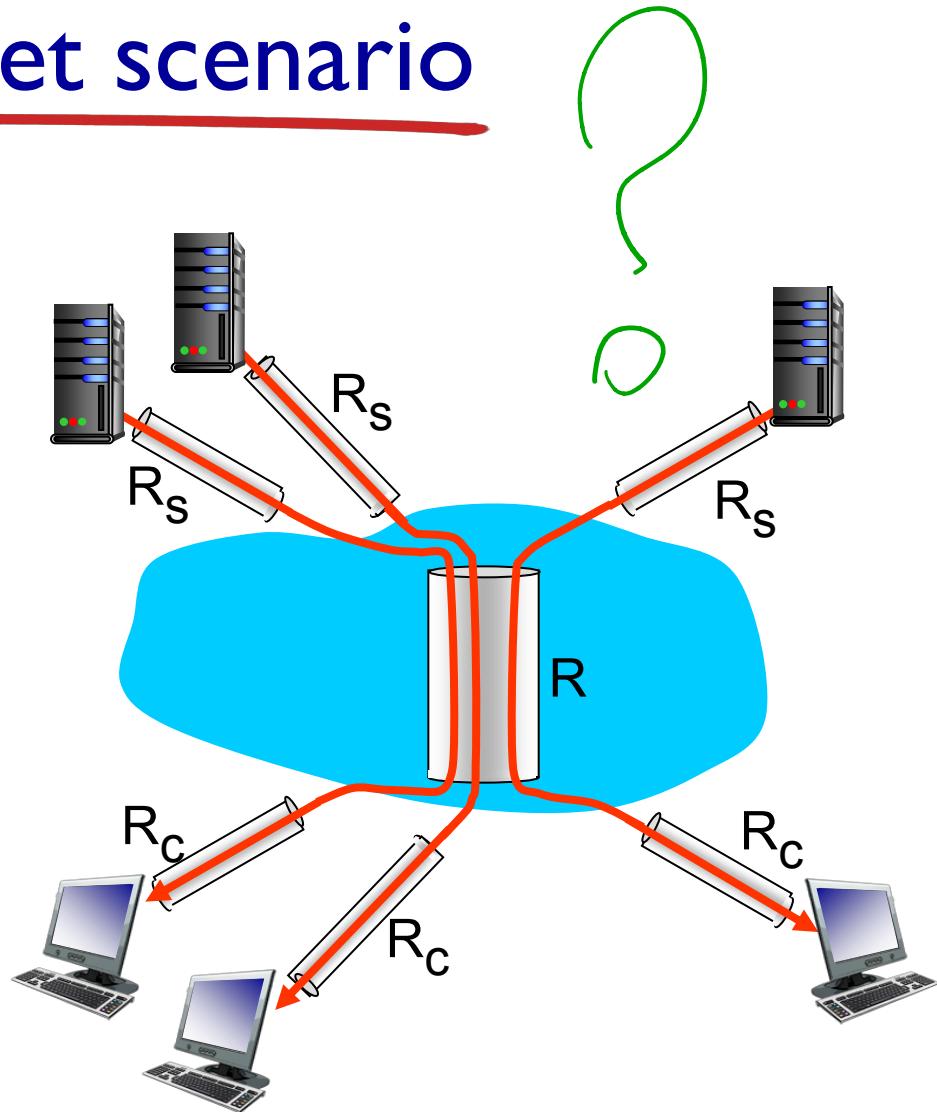


**bottleneck link**

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

# Throughput: Internet scenario

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



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

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# 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

$$F = B \times D^t = B \text{ Mb}$$

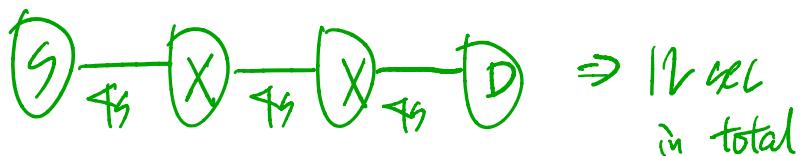
$$R = 2 \text{ Mbps}$$

why segmentation?

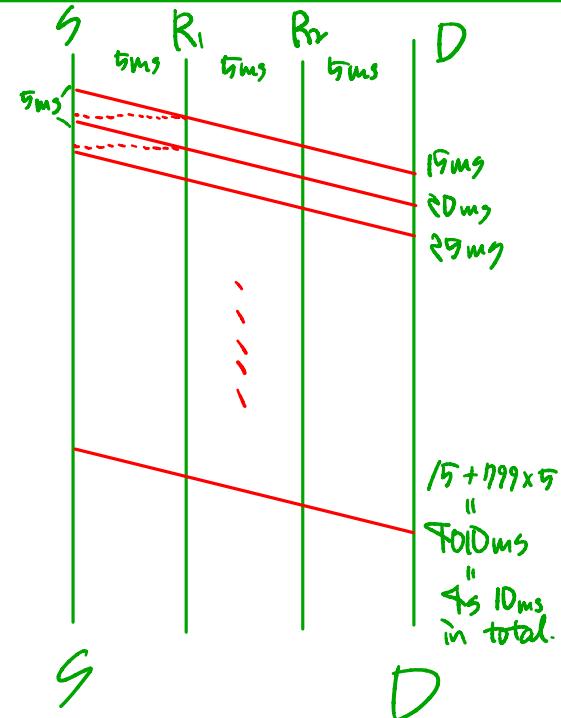
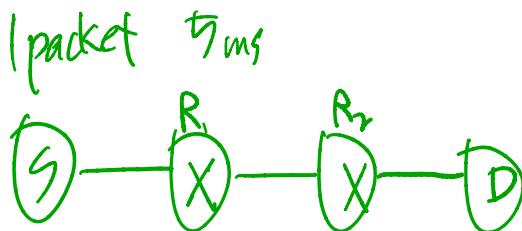
- 1) transmission time reduced
- 2) error control
- 3) job scheduling

a. no seg

$$\frac{B \times 10^6}{2 \times 10^6} = 4 \text{ sec}$$



b.  $10^4$  bit/packet  
 $\Rightarrow 800$  packets



# Protocol “layers”

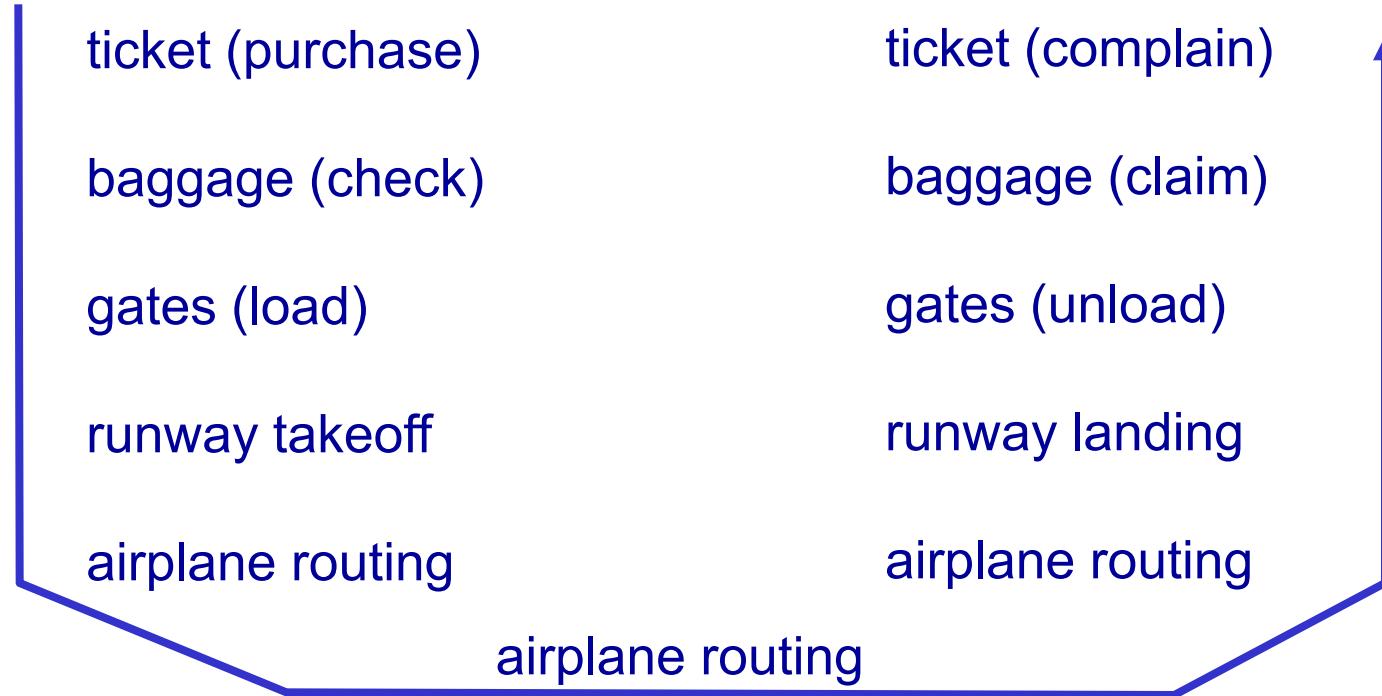
*Networks are complex,  
with many “pieces”:*

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

**Question:**  
is there any hope of  
organizing structure of  
network?

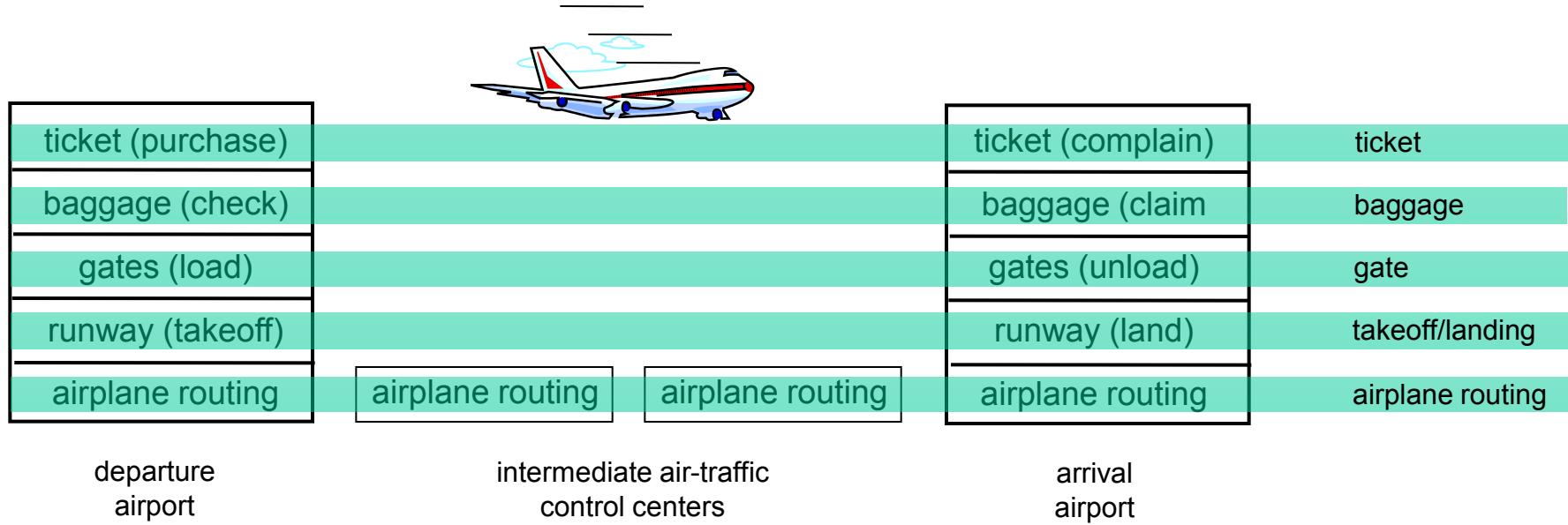
.... or at least our  
discussion of networks?

# Organization of air travel



- a series of steps

# Layering of airline functionality



**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:

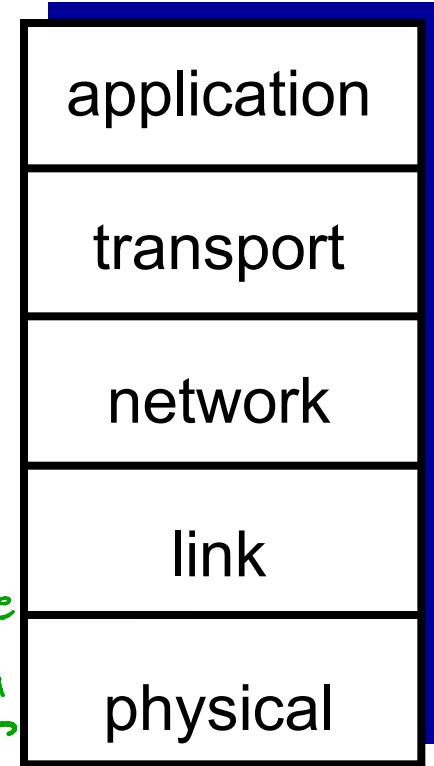
- explicit structure allows identification, relationship of complex system's pieces
  - layered *reference model* for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in gate procedure doesn't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

# Internet protocol stack

Network → not reliable  
:: packet switching  
    delay  
    loss  
    unordered

Goal) re-transmit!

5



- **application:** supporting network applications
  - FTP, SMTP, HTTP
- **transport:** process-process data transfer
  - TCP, UDP
    - logical connection
    - port # assign
    - RDT (Right Data Transmission)
  - congestion
  - flow control
- **network:** routing of datagrams from source to destination
  - IP, routing protocols
- **link:** data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP ... various media!
    - wired
    - wireless
- **physical:** bits “on the wire”

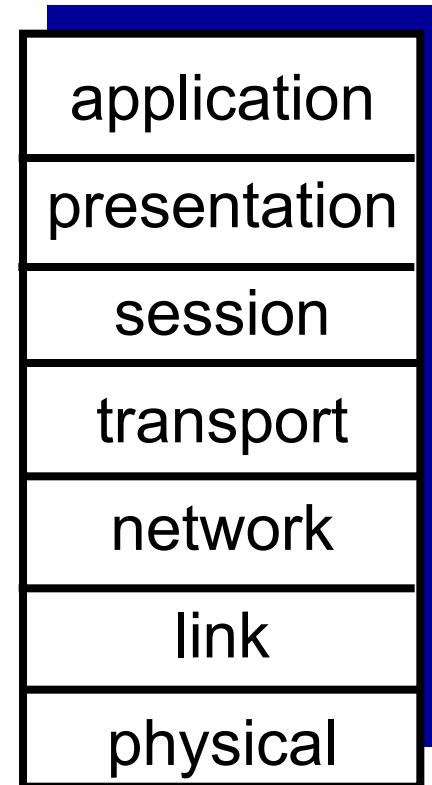
Seg.  
MSB

{

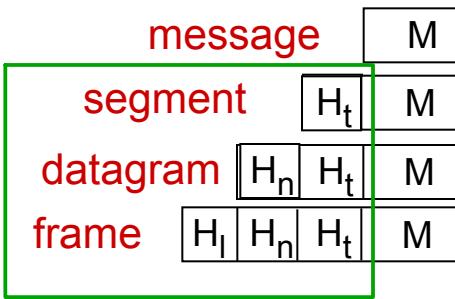
error/collision-free  
shared/dedicated

# ISO/OSI reference model

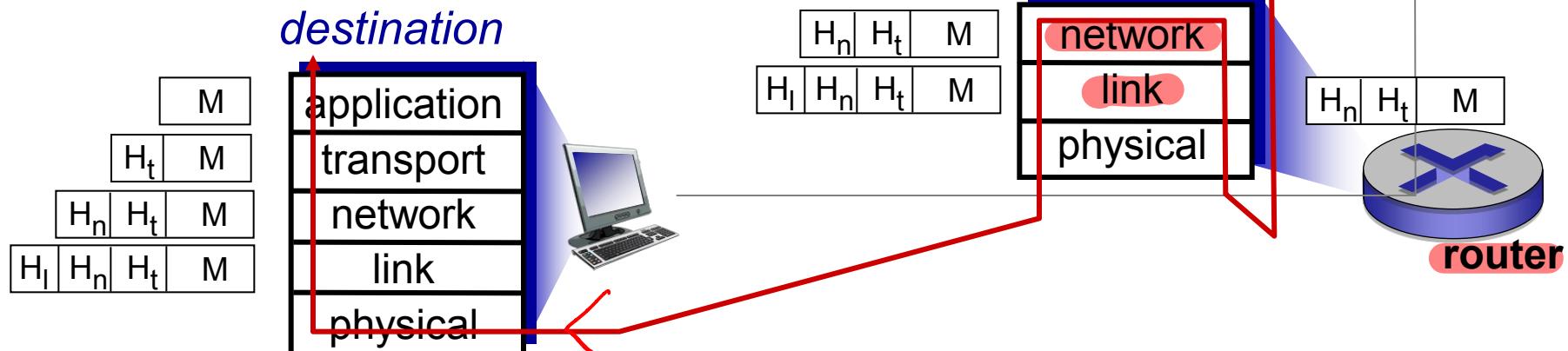
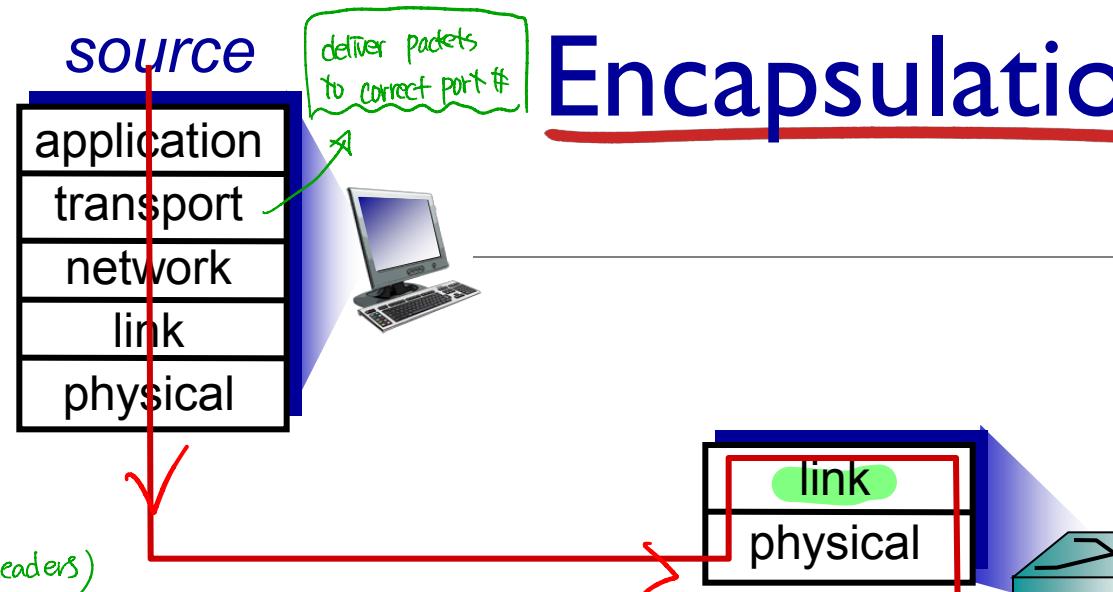
- ***presentation:*** allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- ***session:*** synchronization, checkpointing, recovery of data exchange
- Internet stack “missing” these layers!
  - these services, *if needed*, must be implemented in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



# Encapsulation



disadvantage  
⇒ too much overhead  
due to redundant info (headers)  
& unordered delivery



# 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!

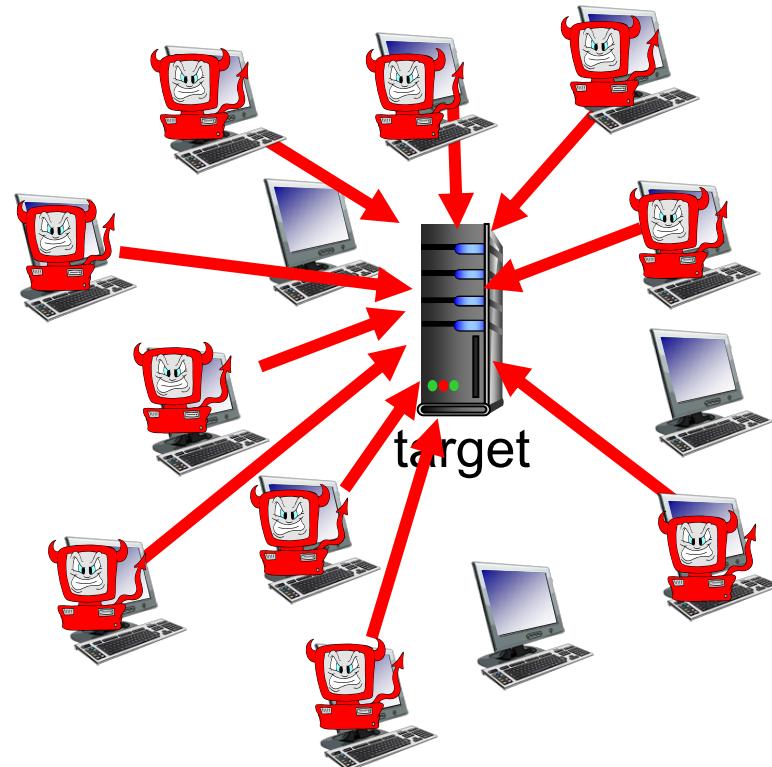
# Bad guys: put malware into hosts via Internet

- 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

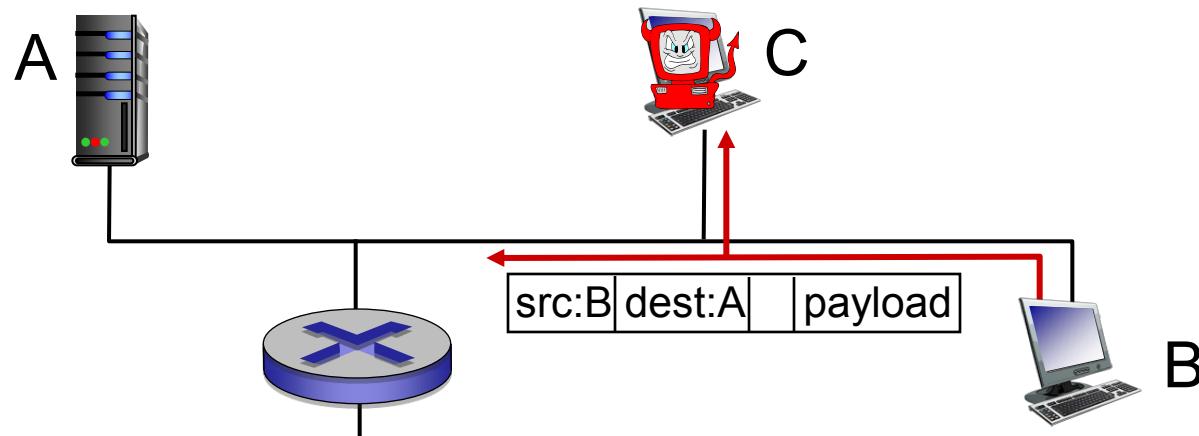
1. select target
2. break into hosts around the network (see botnet)
3. send packets to target from compromised hosts



# Bad guys can sniff packets

*packet “sniffing”:*

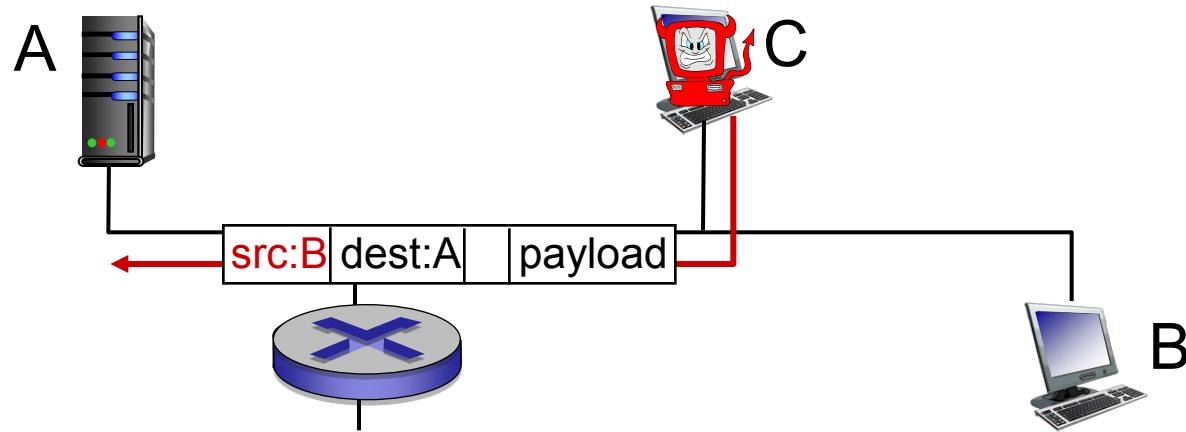
- broadcast media (shared Ethernet, wireless)
- promiscuous network interface reads/records all packets (e.g., including passwords!) passing by



- 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

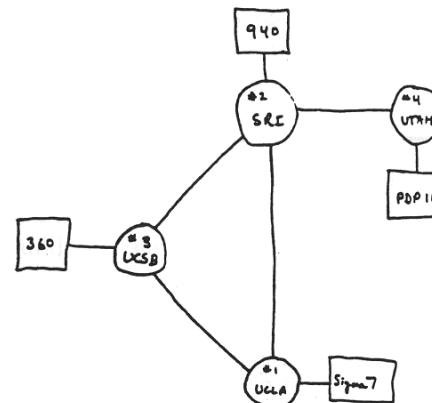
I.6 networks under attack: security

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*

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

# Introduction: summary

*covered a “ton” of material!*

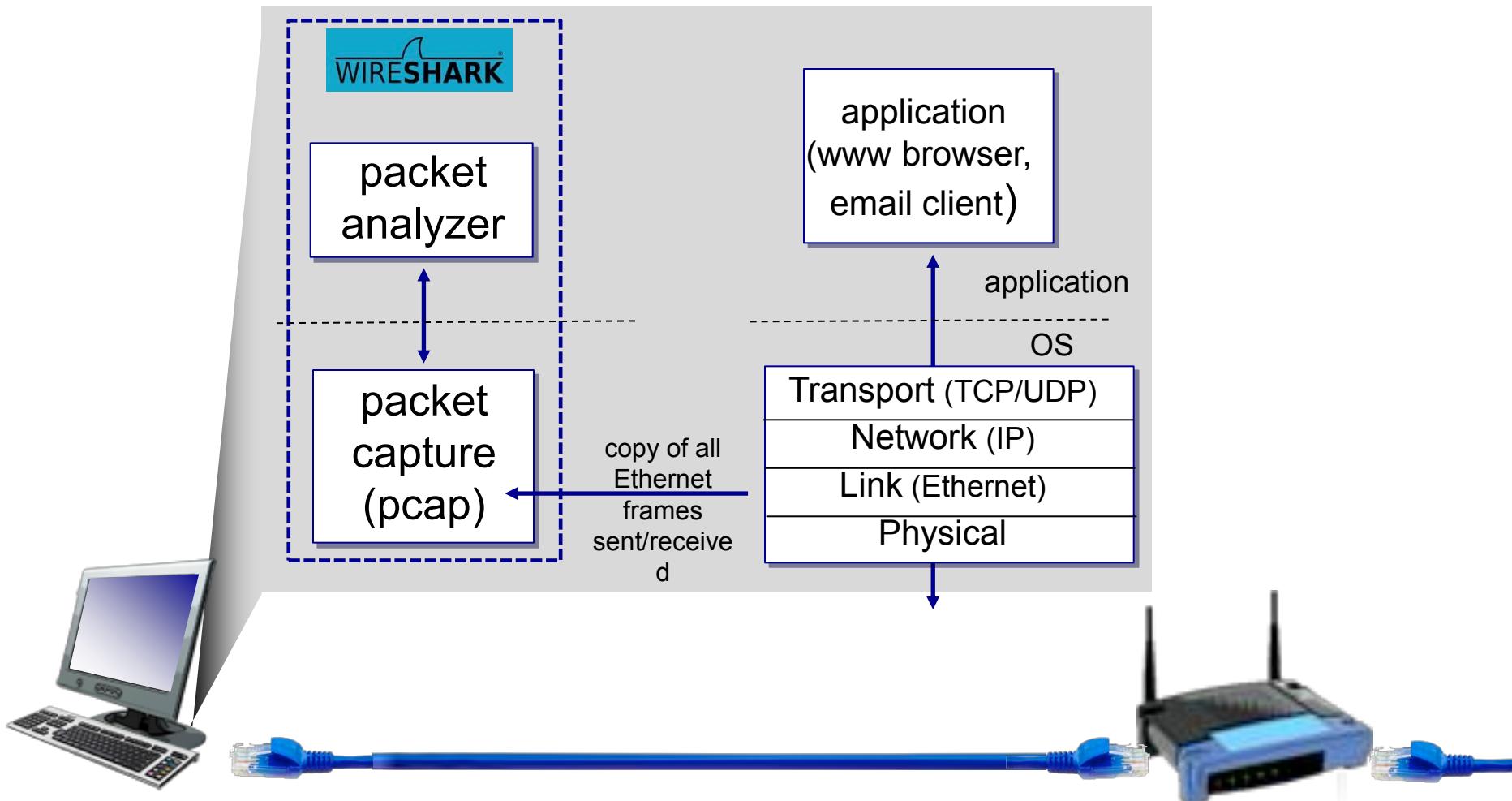
- Internet overview
- what’s a protocol?
- network edge, core, access network
  - packet-switching versus circuit-switching
  - Internet structure
- performance: loss, delay, throughput
- layering, service models
- security
- history

*you now have:*

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

# Chapter I

# Additional Slides



# Chapter 3

## Transport Layer

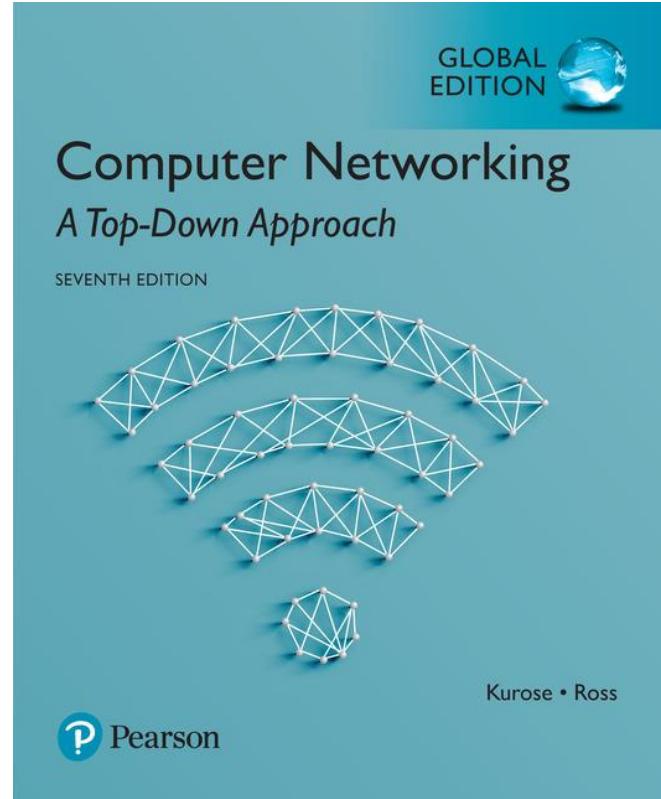
### A note on the use of these Powerpoint slides:

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

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

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016  
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer  
Networking: A Top  
Down Approach*

7<sup>th</sup> Edition, Global Edition  
Jim Kurose, Keith Ross  
Pearson  
April 2016

# Chapter 3: Transport Layer

## our goals:

- understand principles behind transport layer services:
  - multiplexing, demultiplexing *] port # issue*
  - reliable data transfer *loss delay unordered*
  - flow control *overflow of buffer*
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - ~~TCP congestion control~~

# Chapter 3 outline

issues  
loss  
delay  
unordered

## 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

consider issues

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

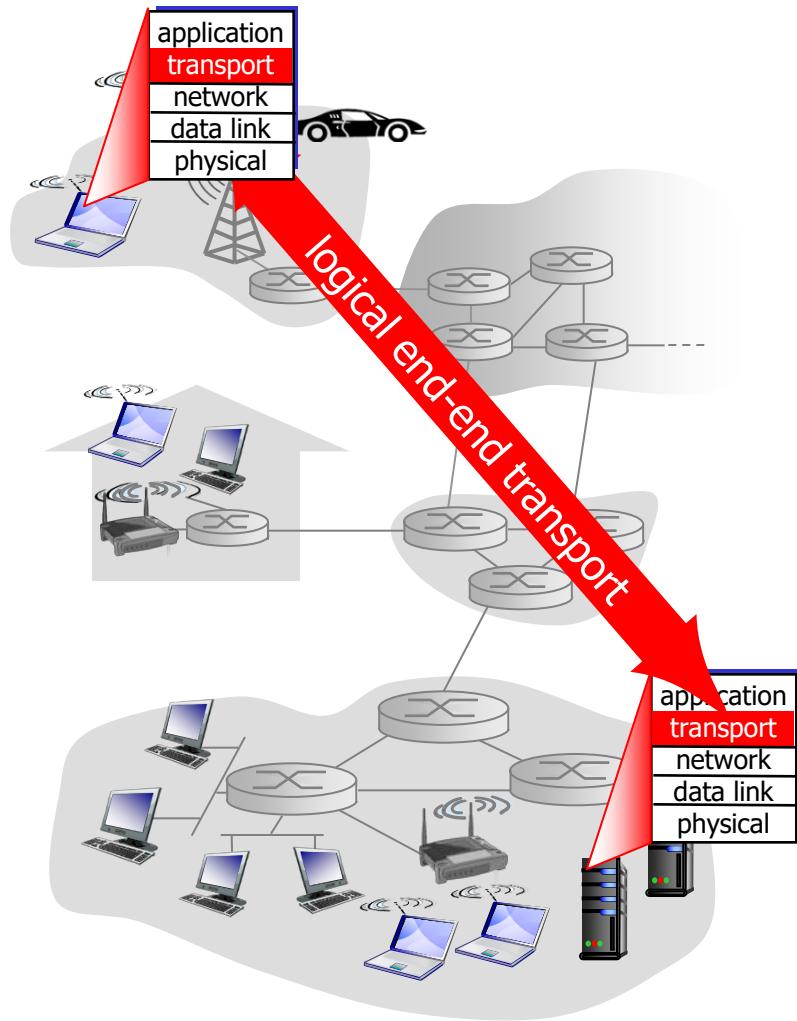
## 3.6 principles of congestion control

## 3.7 TCP congestion control

Transport Layer only for end-system.

# 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  
    → IP address
- *transport layer*: → port number  
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

error sensitive  
→ connection oriented  
(before sending data  
3-way handshaking)

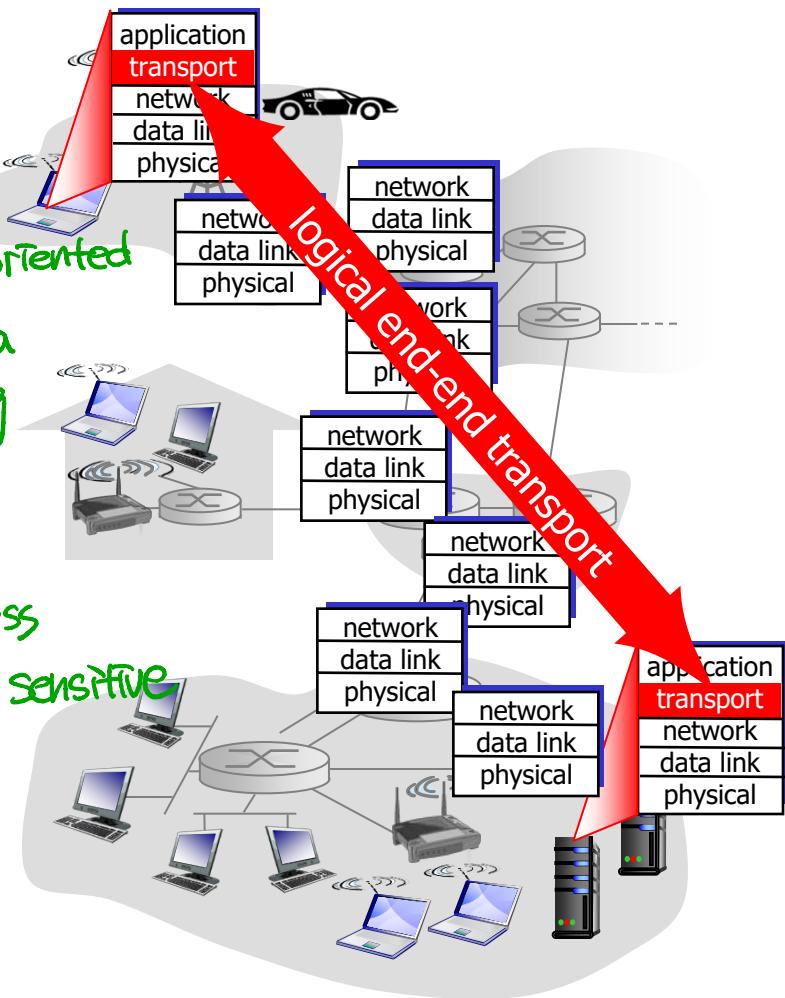
- unreliable, unordered delivery: UDP

- no-frills extension of “best-effort” IP

→ connectionless  
real time, delay sensitive

- 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

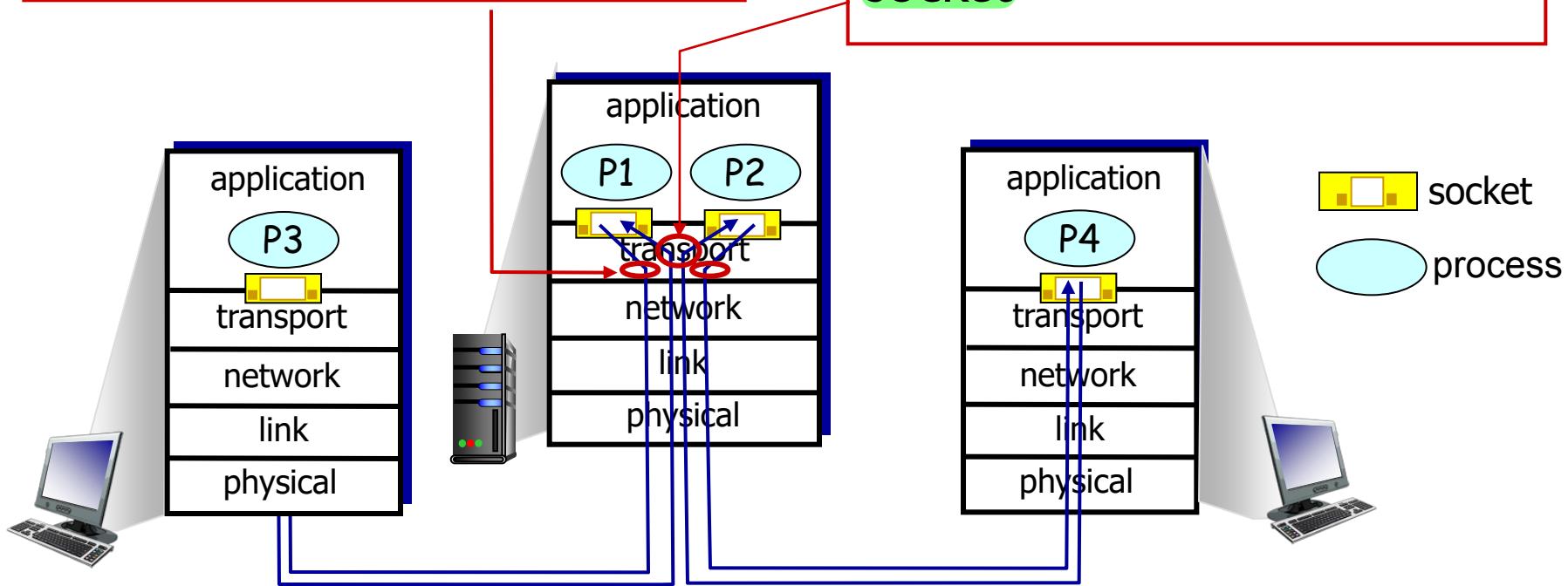
# Multiplexing/demultiplexing

- multiplexing at sender:

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

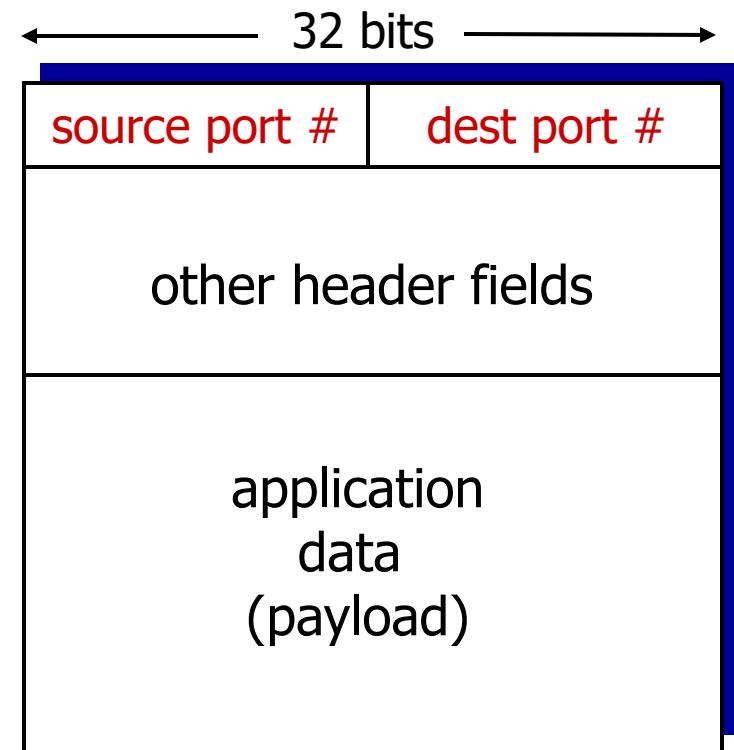
port number , 16 bit

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  $\Rightarrow$  Network layer
  - 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

UDP

# 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 #

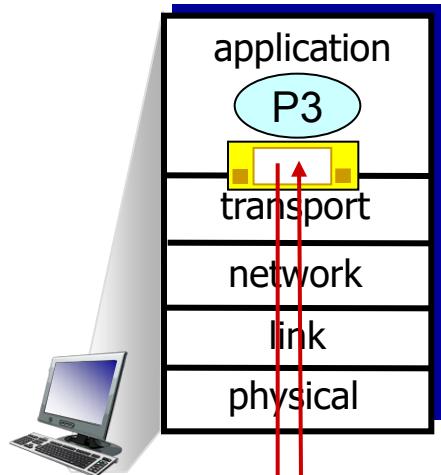


IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

UDP

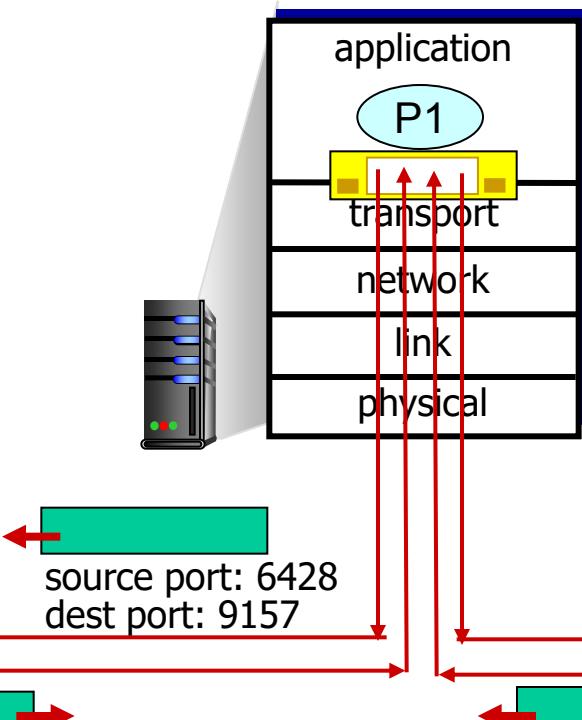
# Connectionless demux: example

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

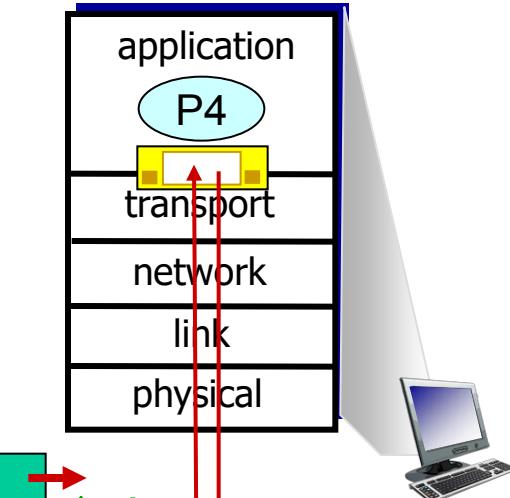


## DatagramSocket

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



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

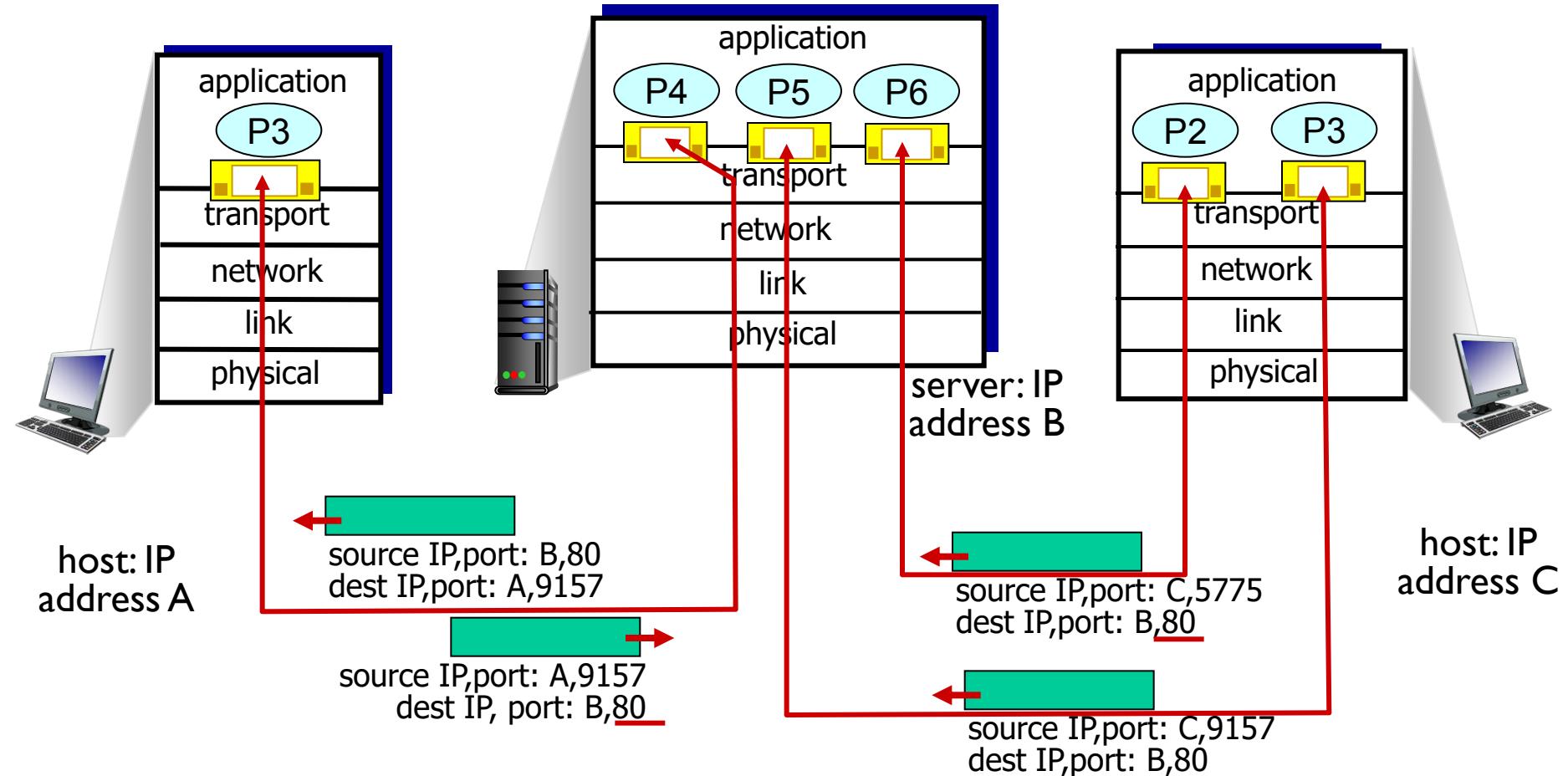


# Connection-oriented demux

- 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

TCP

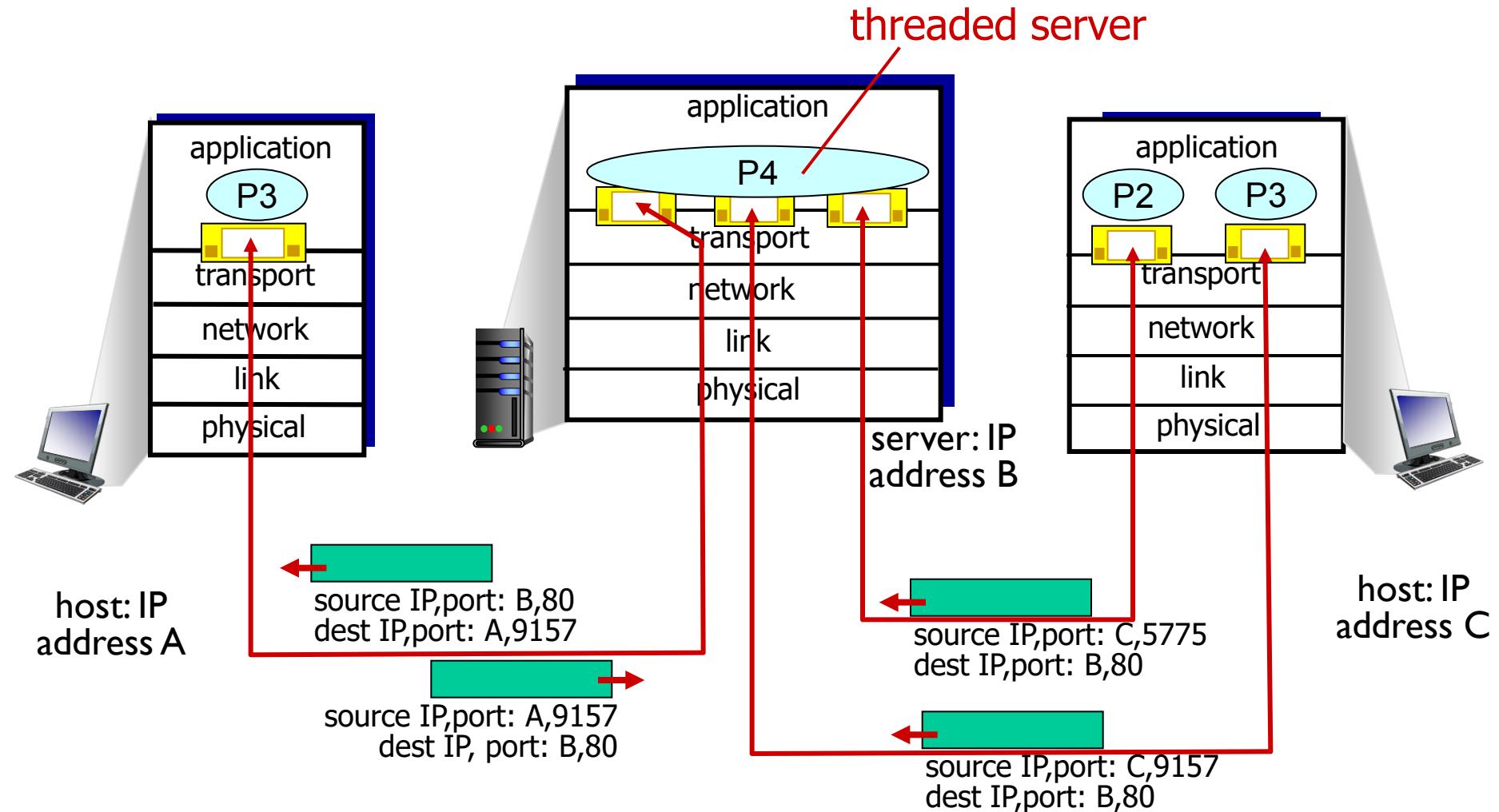
# Connection-oriented demux: example



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

TCP

# 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 deco*

- “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:

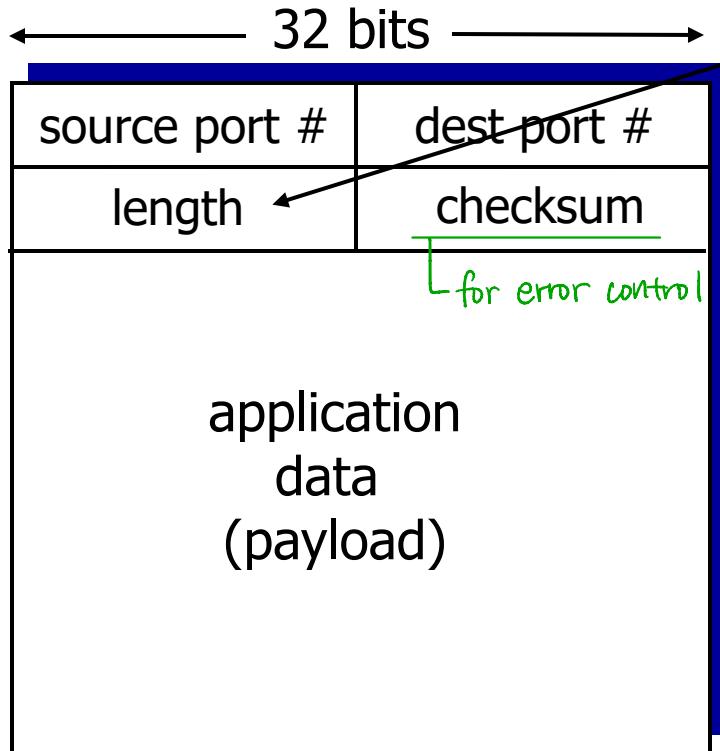
- streaming multimedia apps (*loss tolerant, rate sensitive*)  
*delay >*

- 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

problems in link  
not transport/network layer

## 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

src port #	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
dst port #	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
<hr/>																

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

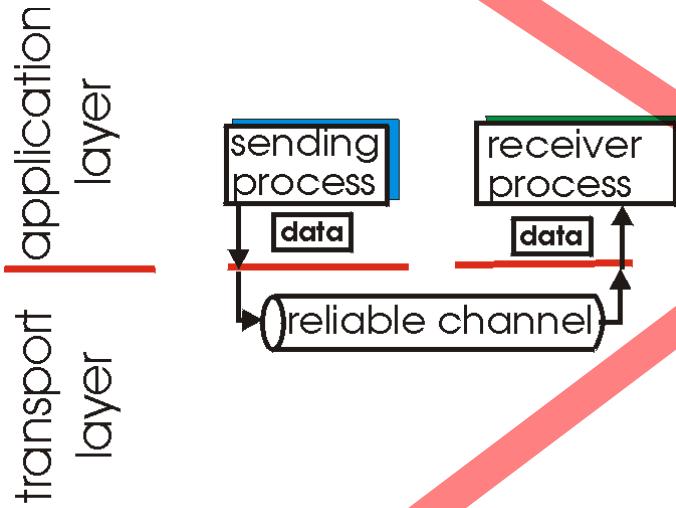
- segment structure
- reliable data transfer
- flow control
- connection management

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!

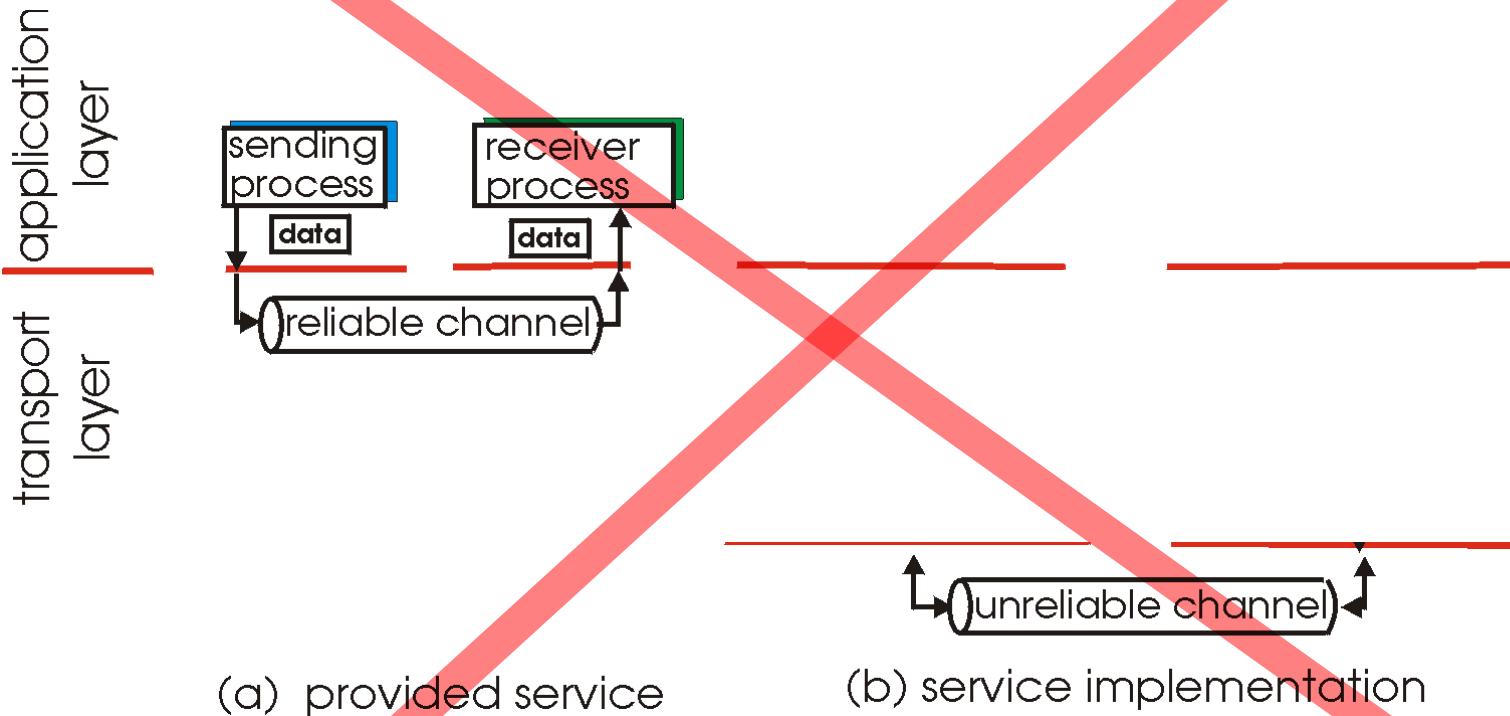


(a) provided service

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

# Principles of reliable data transfer

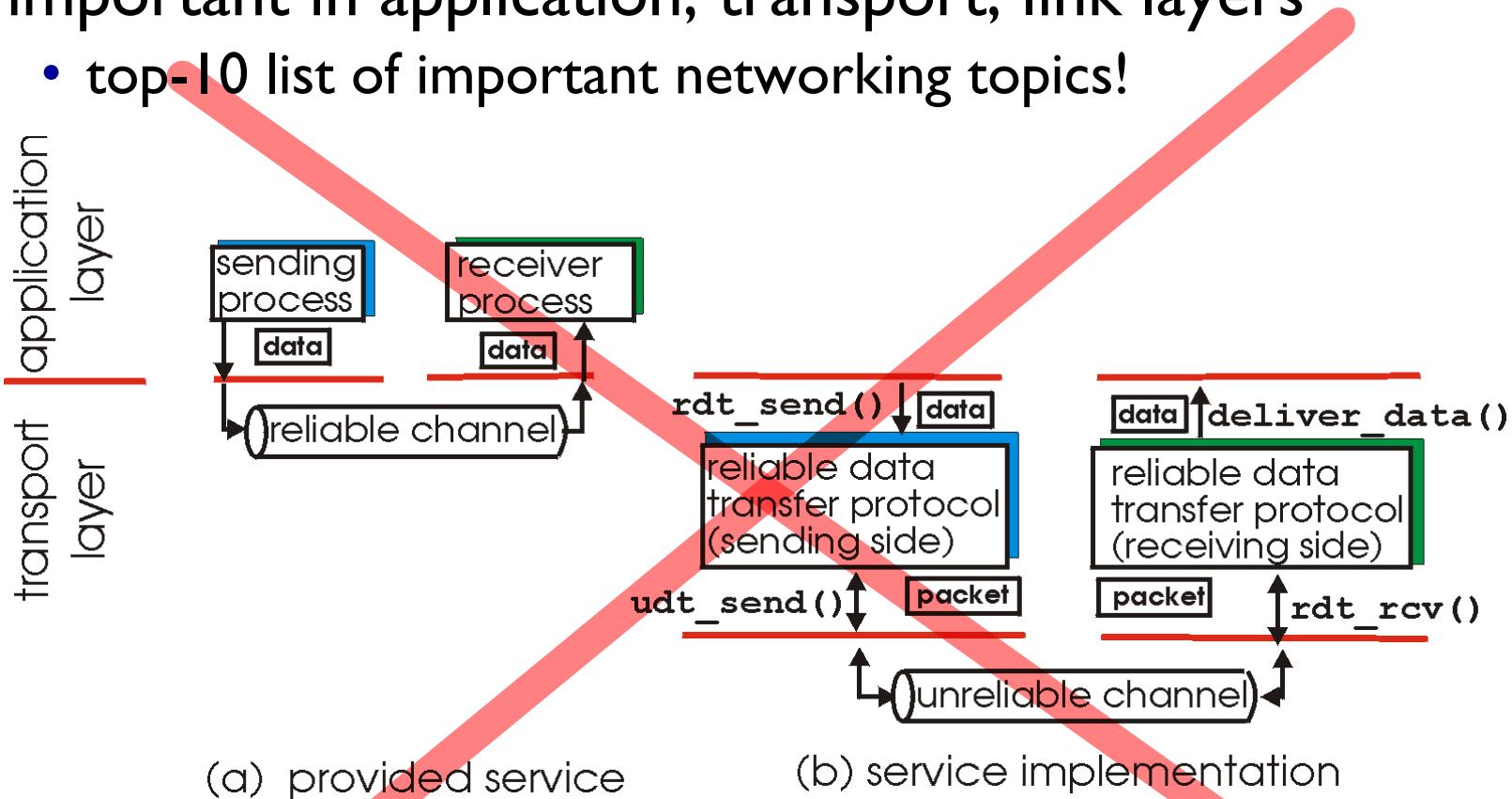
- important in application, transport, link layers
  - top-10 list of important networking topics!



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

# Principles of reliable data transfer

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

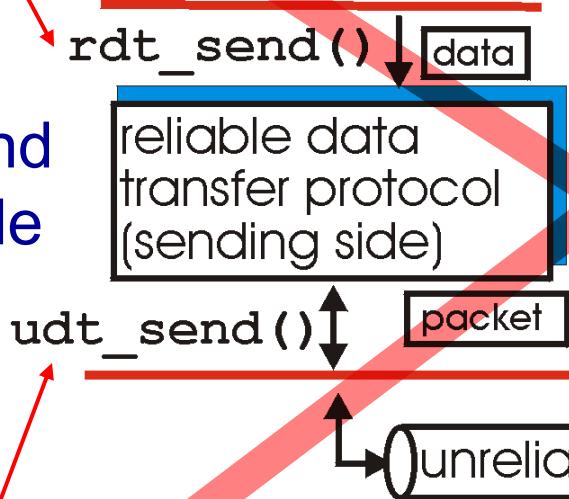


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

# Reliable data transfer: getting started

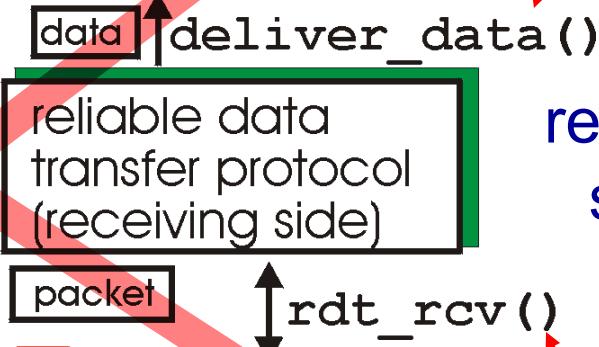
**`rdt_send()`**: called from above,  
(e.g., by app.). Passed data to  
deliver to receiver upper layer

send  
side



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

receive  
side



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

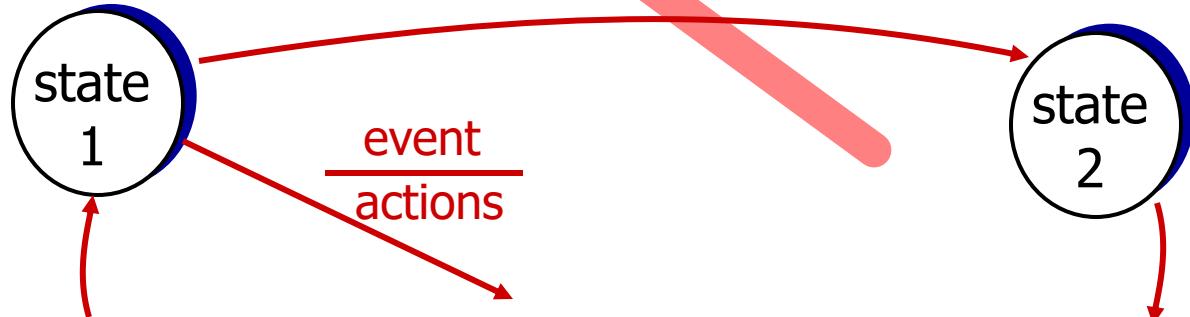
**`rdt_recv()`**: called when packet  
arrives on rcv-side of channel

# Reliable data transfer: getting started

we'll:

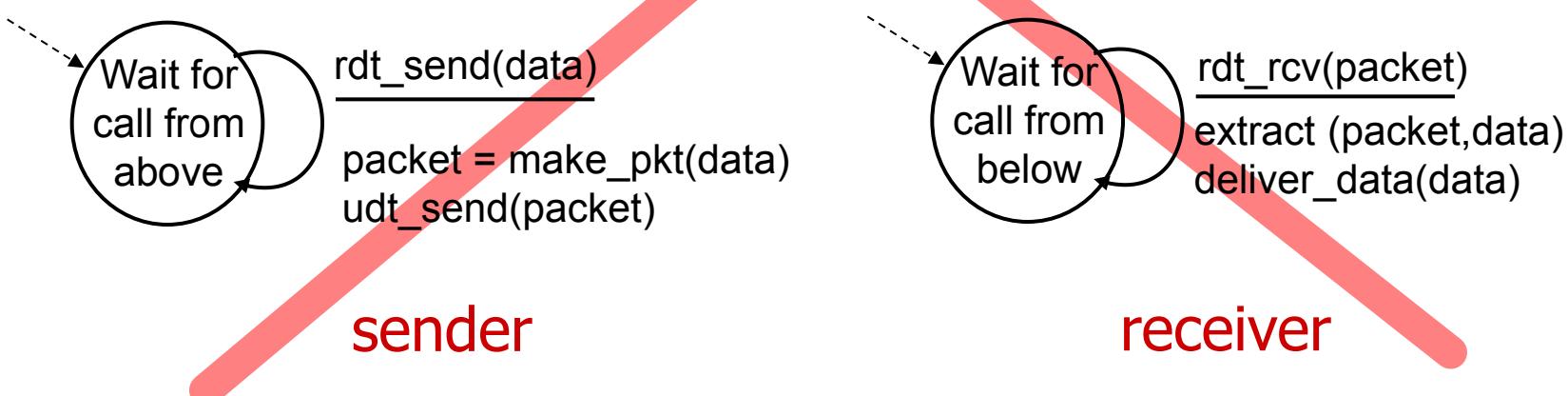
- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver

**state:** when in this “state” next state uniquely determined by next event



# rdt 1.0: reliable transfer over a reliable channel

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



# rdt2.0: channel with bit errors

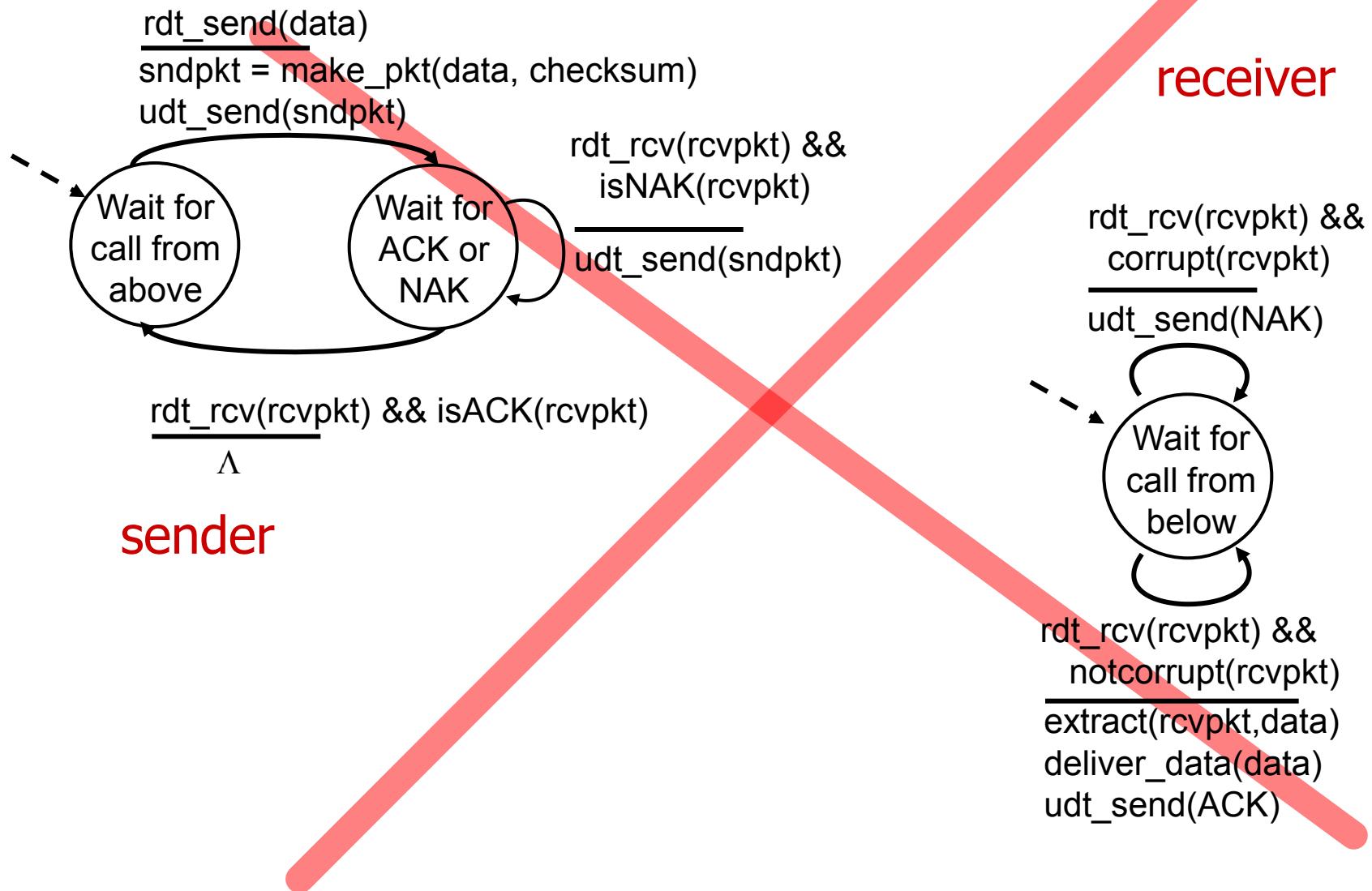
- underlying channel may flip bits in packet
  - checksum to detect bit errors
- the question: how to recover from errors:

*How do humans recover from “errors”  
during conversation?*

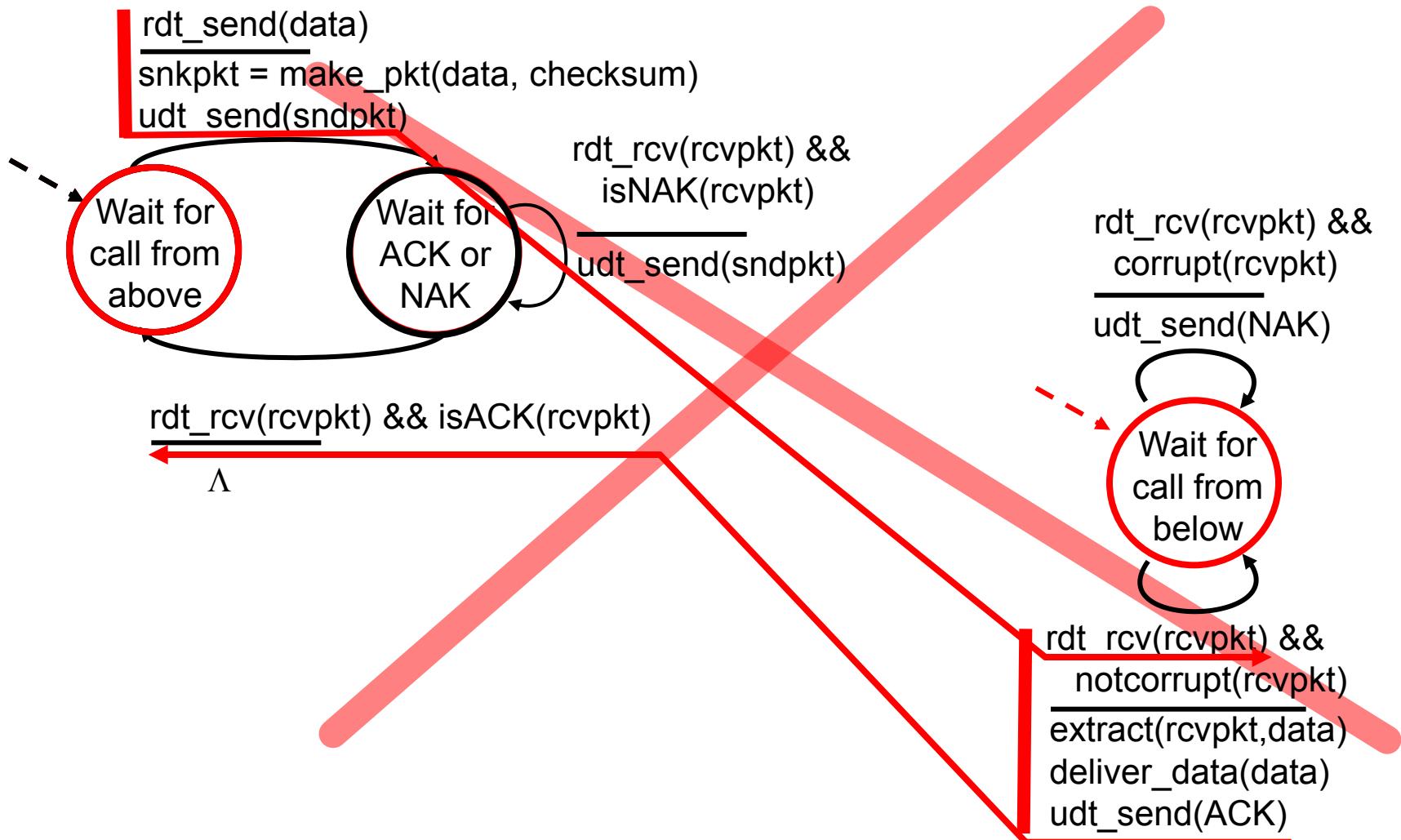
# rdt2.0: channel with bit errors

- 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

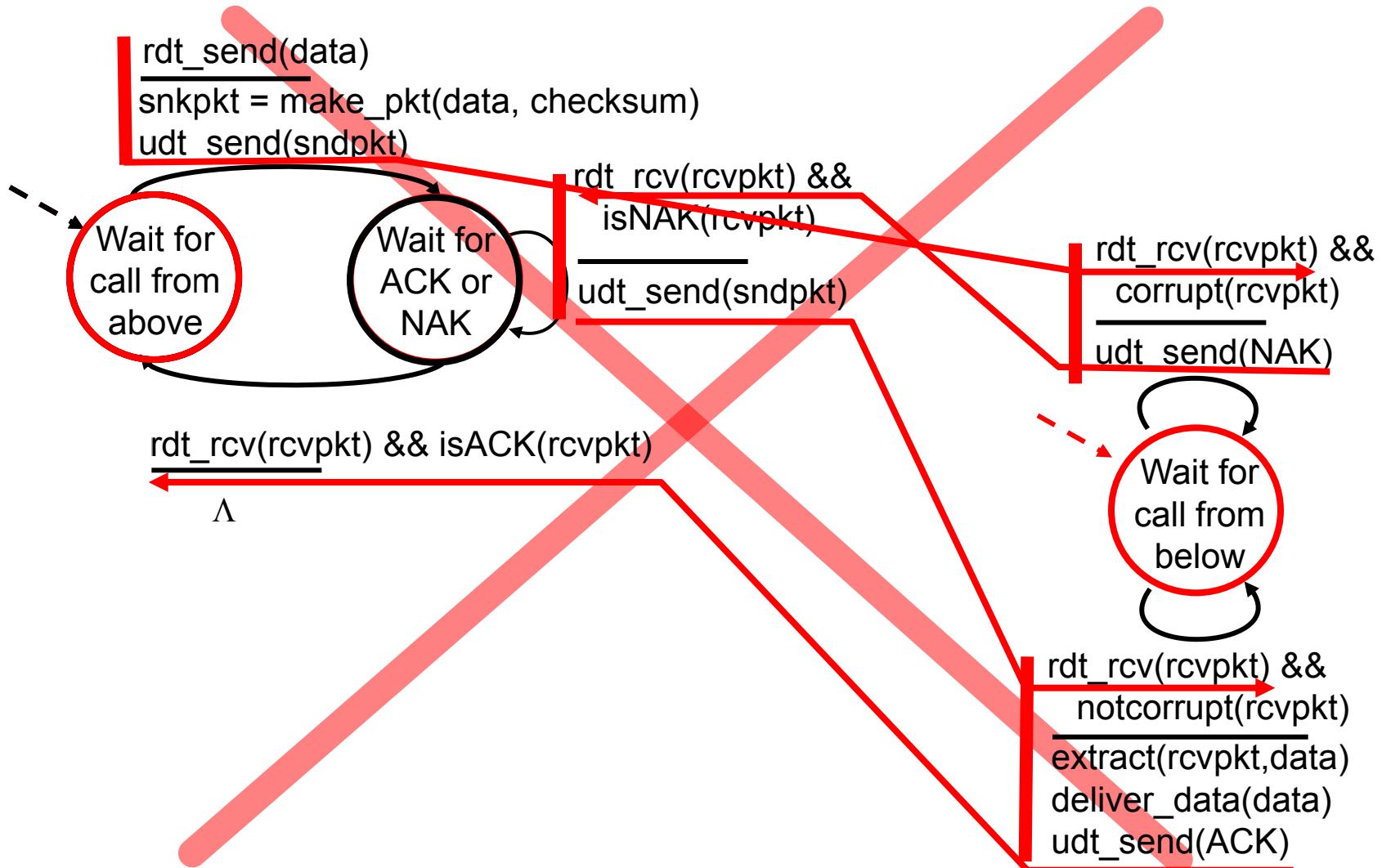
# rdt2.0: FSM specification



# rdt2.0: operation with no errors



# rdt2.0: error scenario



# rdt2.0 has a fatal flaw!

## what happens if ACK/NAK corrupted?

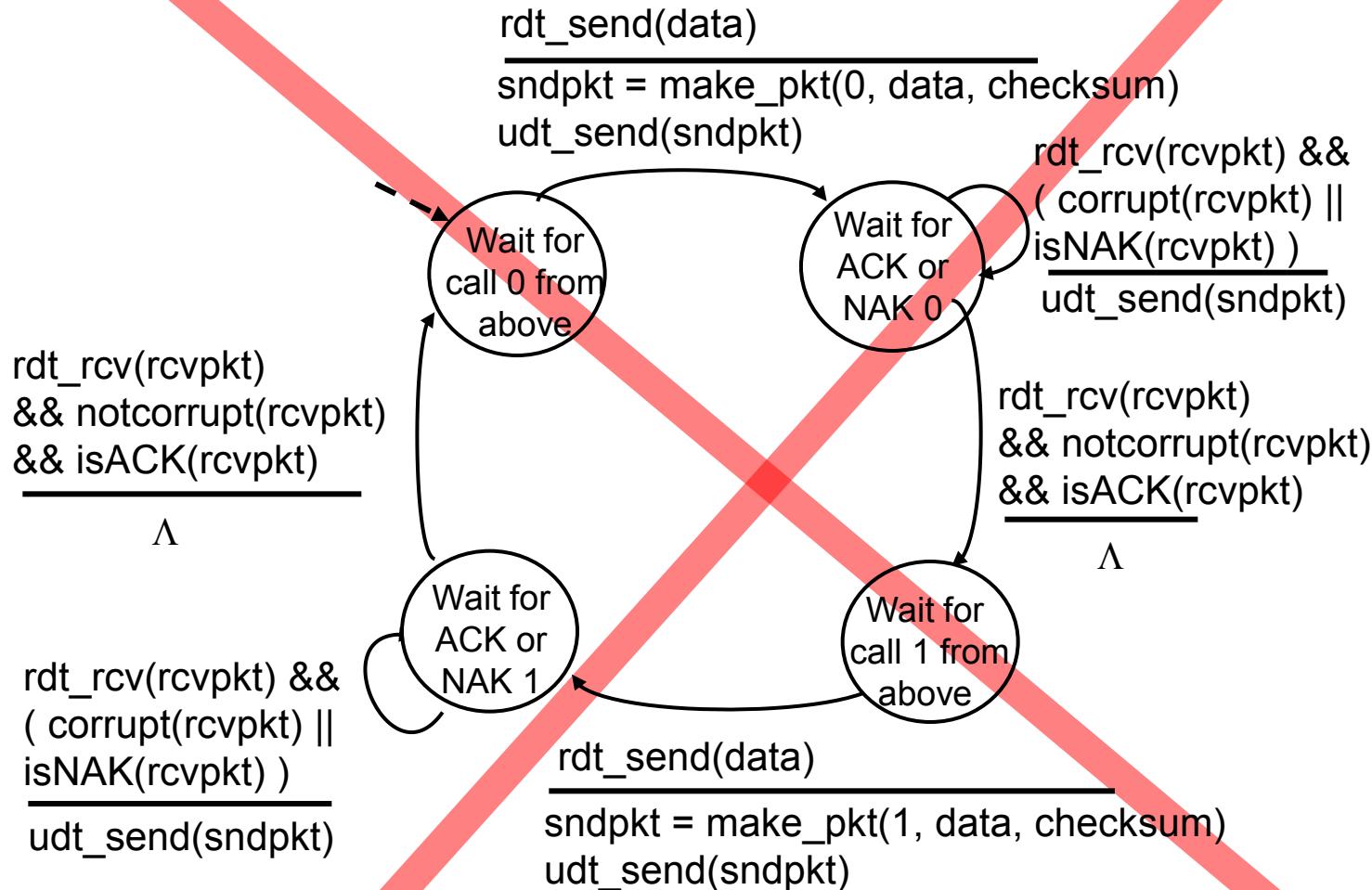
- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

## handling duplicates:

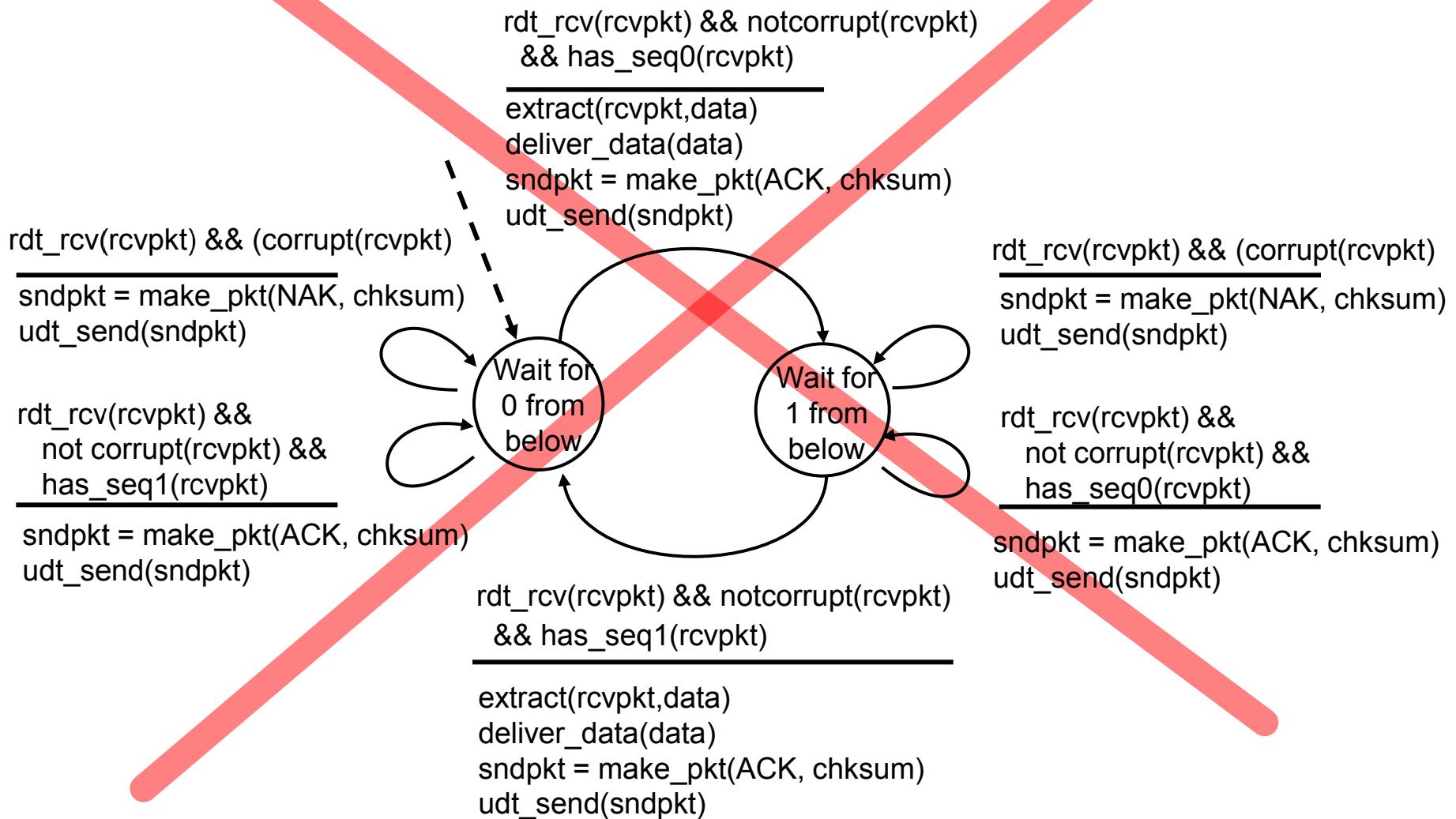
- sender retransmits current pkt if ACK/NAK corrupted
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

**stop and wait**  
sender sends one packet,  
then waits for receiver  
response

# rdt2.1: sender, handles garbled ACK/NAKs



# rdt2.1: receiver, handles garbled ACK/NAKs



# rdt2.1: discussion

## sender:

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

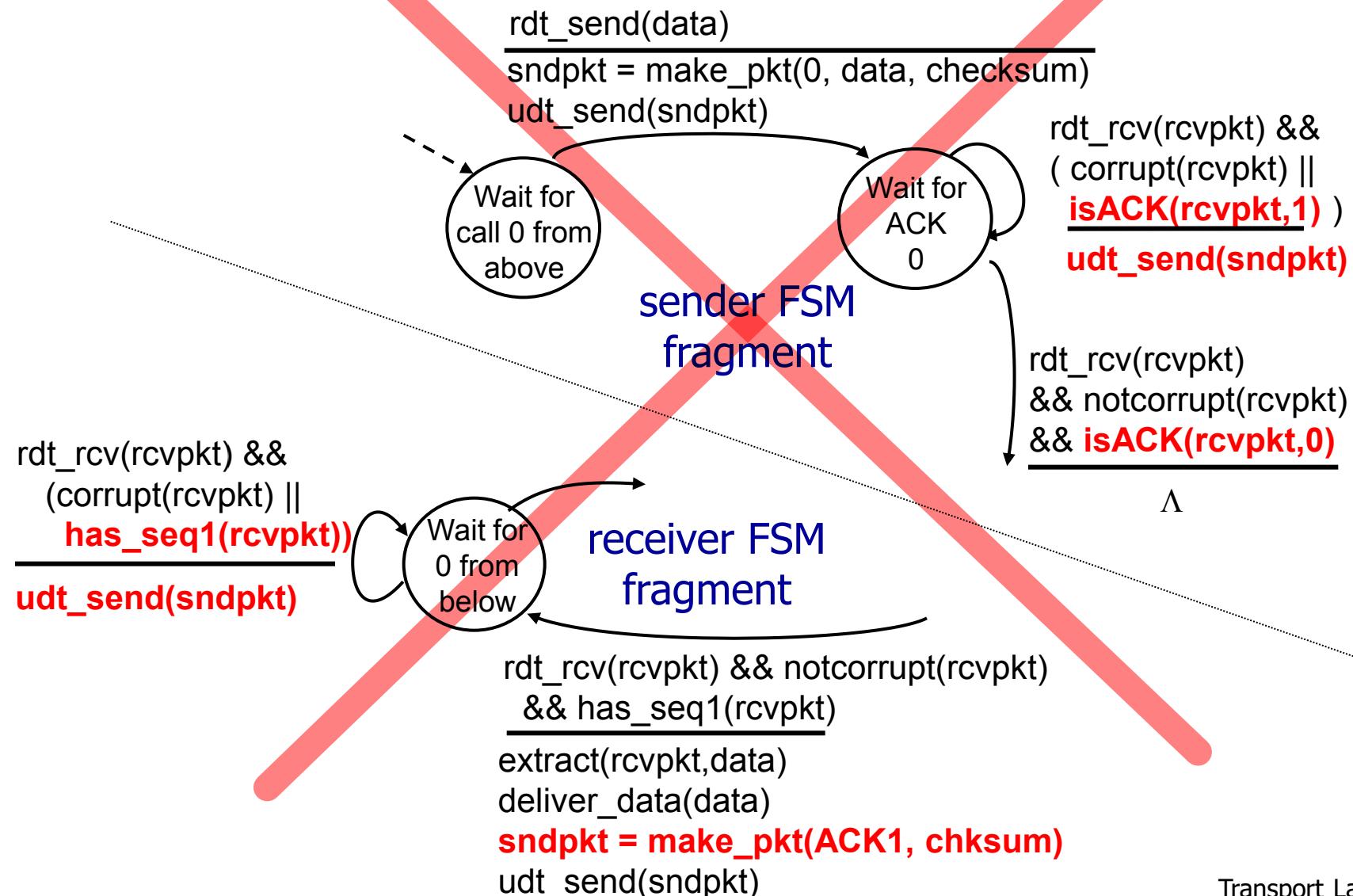
## receiver:

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

## 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*

# rdt2.2: sender, receiver fragments



# rdt3.0: channels with errors and loss

## new assumption:

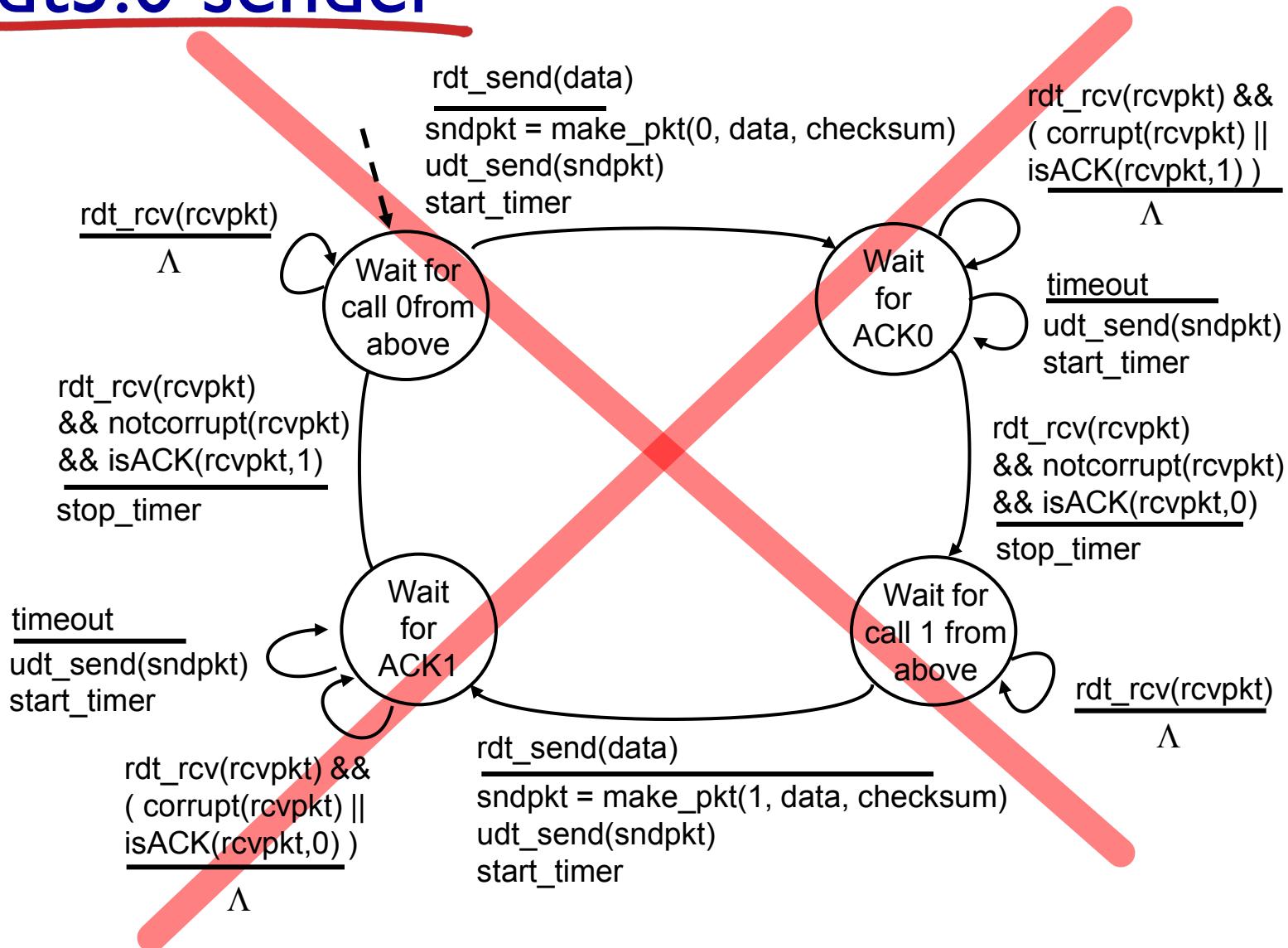
underlying channel can also lose packets (data, ACKs)

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

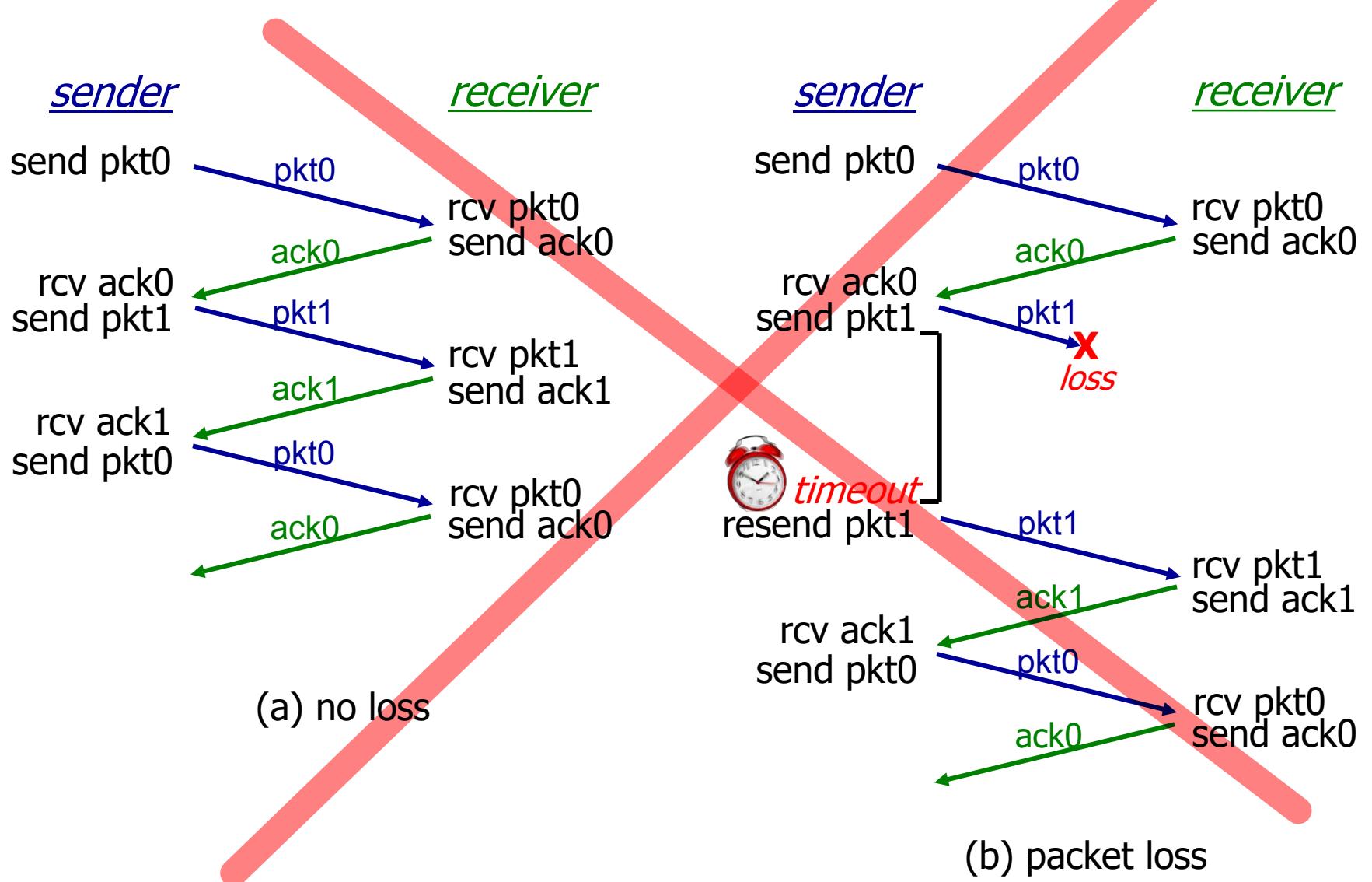
## approach: sender waits “reasonable” amount of time for ACK

- 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

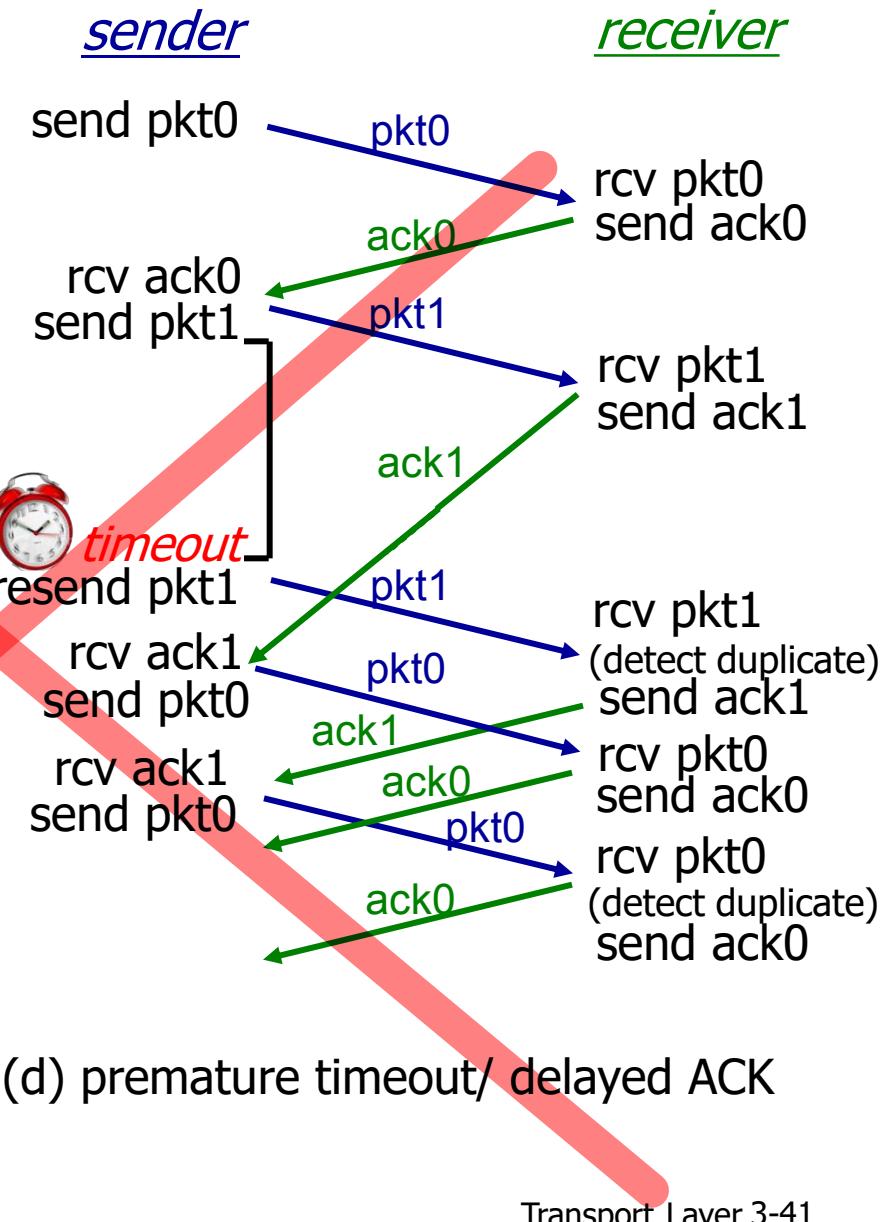
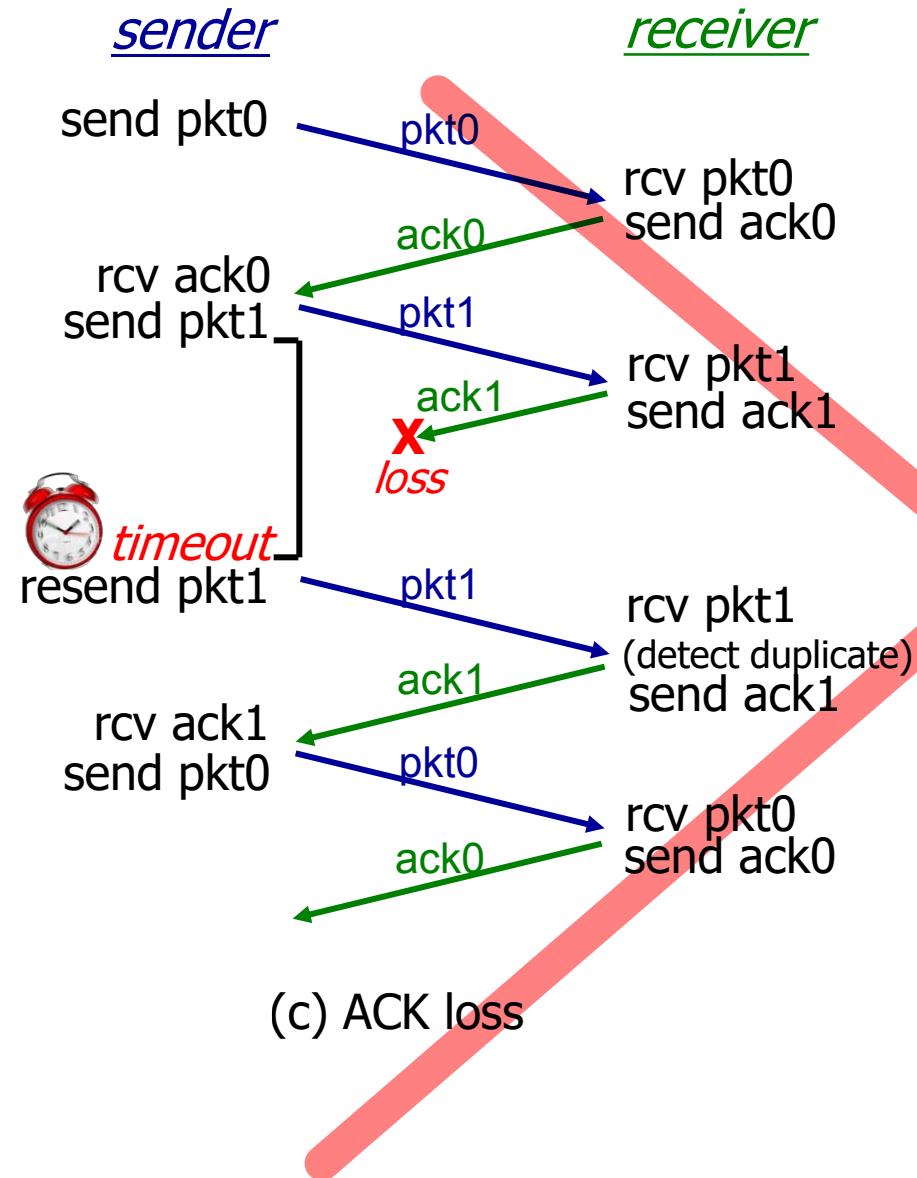
# rdt3.0 sender



# rdt3.0 in action



# rdt3.0 in action



# Performance of rdt3.0

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

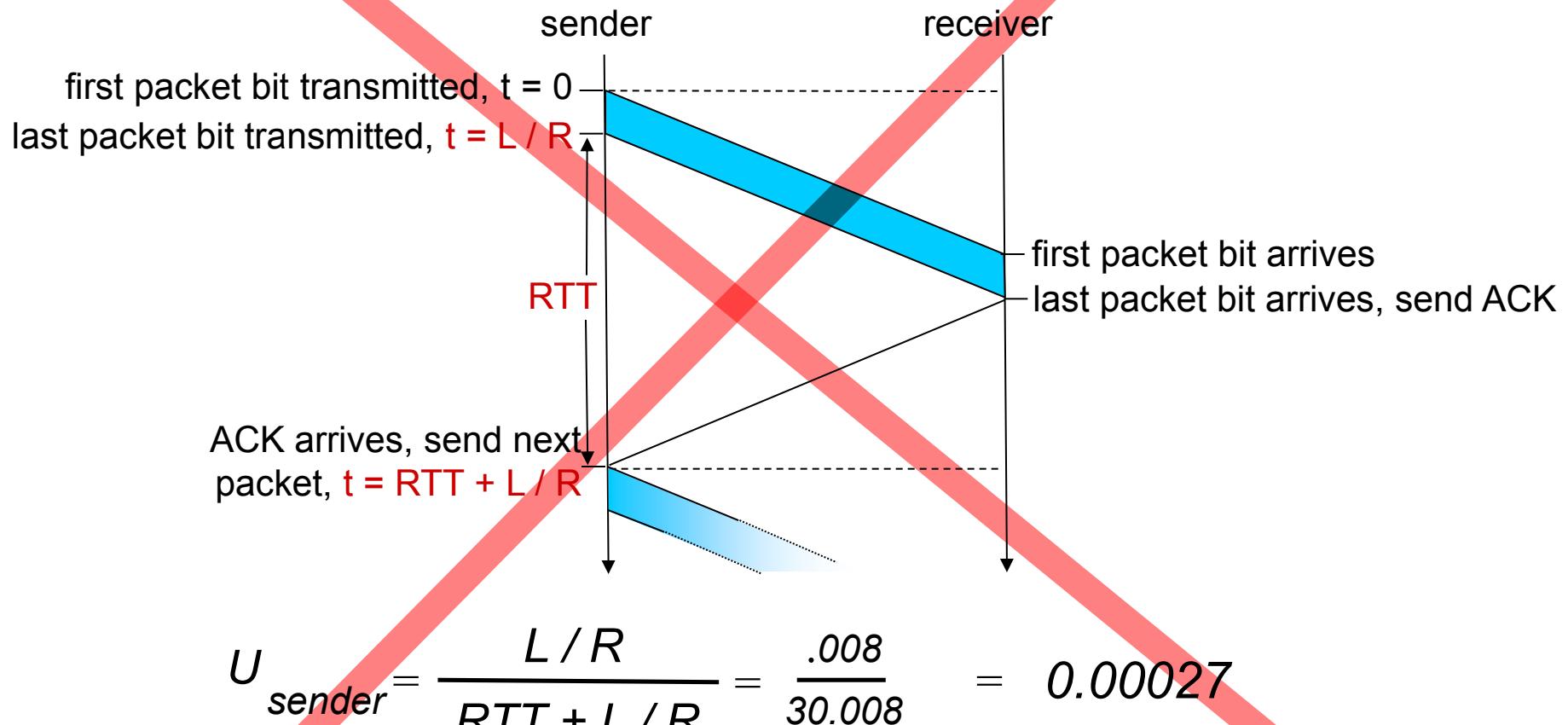
$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

- $U_{\text{sender}}$ : *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!

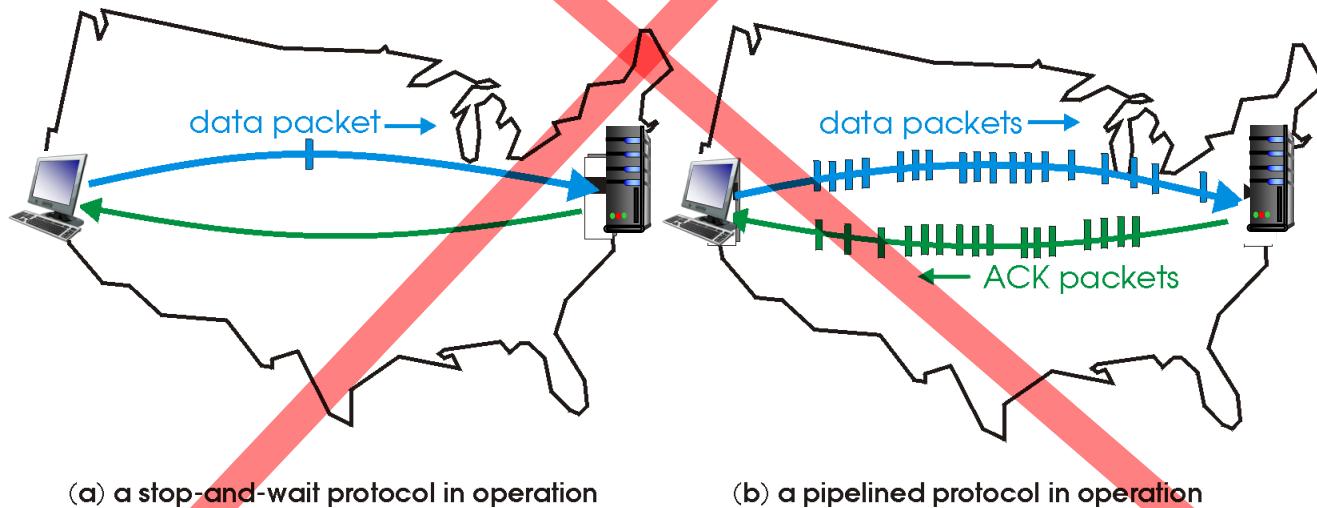
# rdt3.0: stop-and-wait operation



# Pipelined protocols

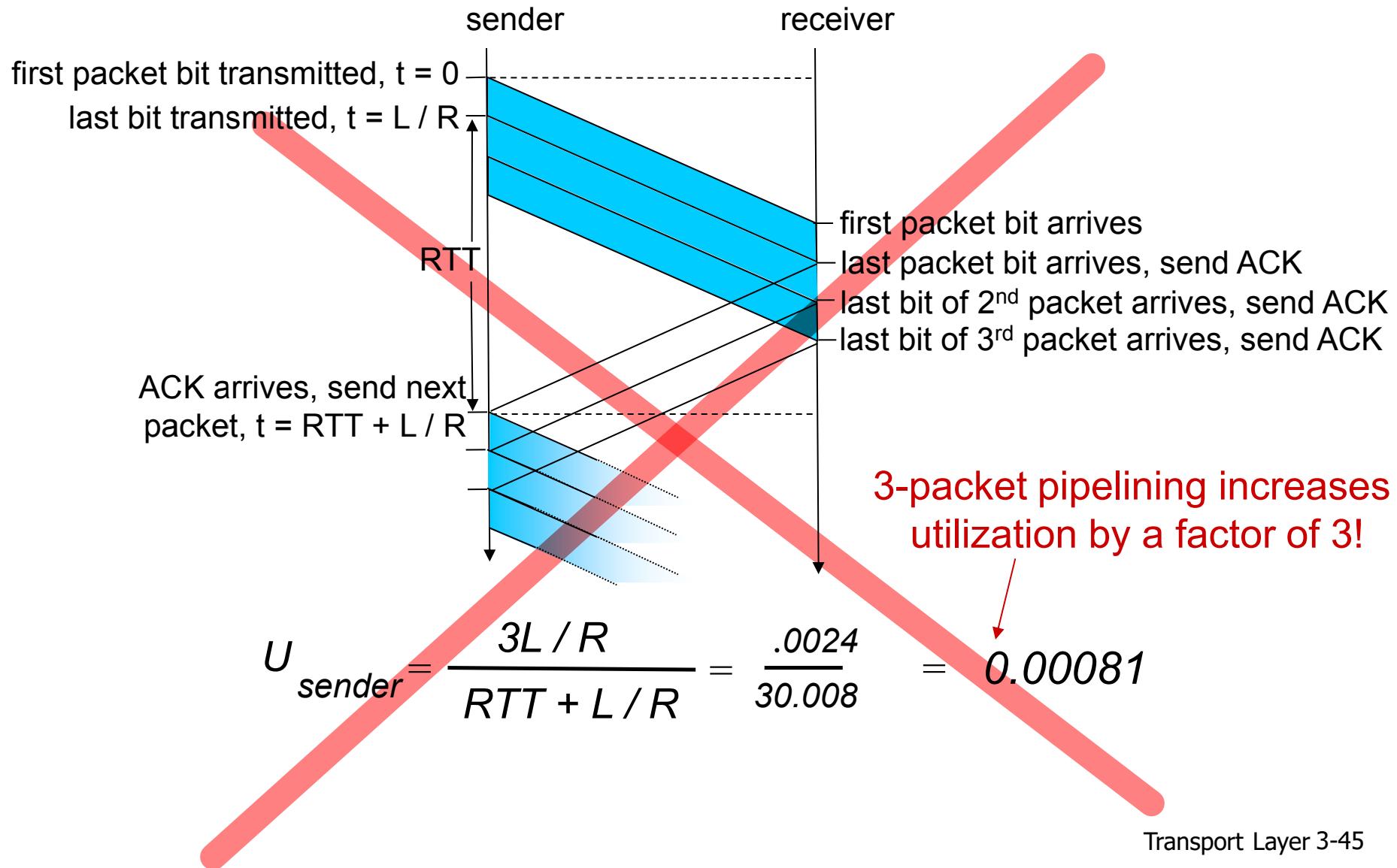
**pipelining:** sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- 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:

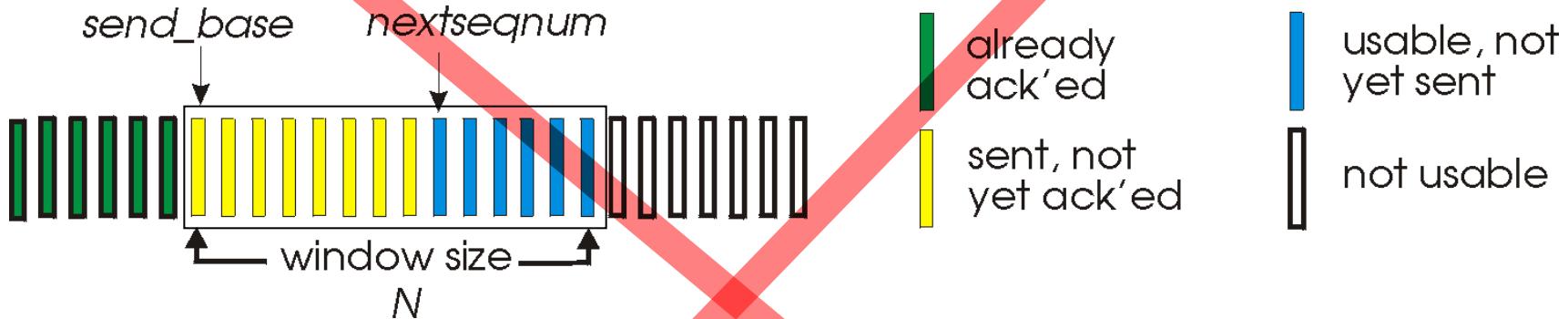
- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

## Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit *only* that unacked packet

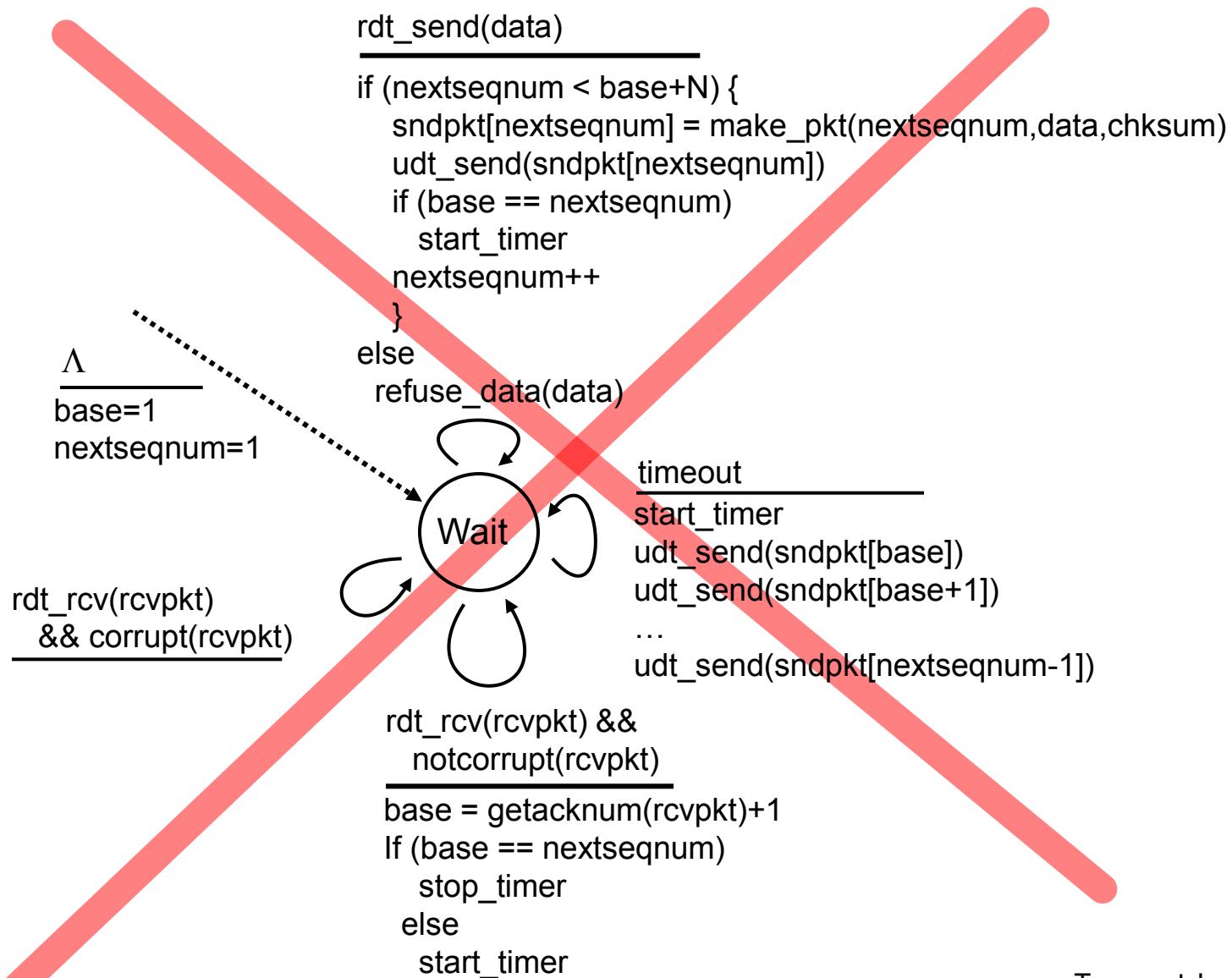
# Go-Back-N: sender

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

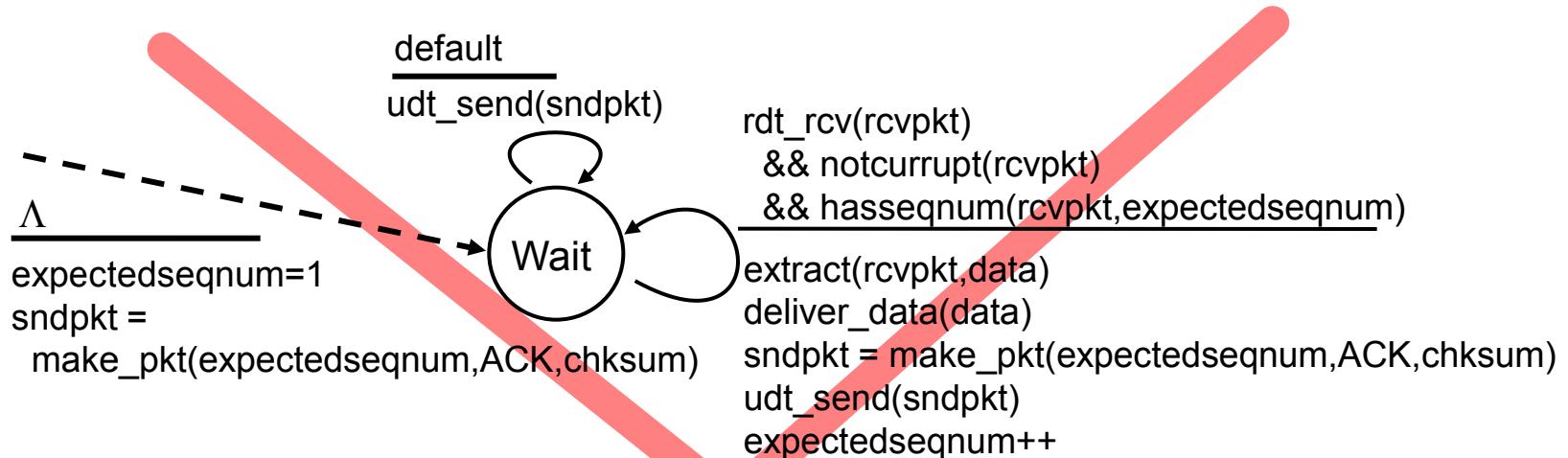


- ACK(n):ACKs all pkts up to, including seq # n - “*cumulative ACK*”
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- $timeout(n)$ : retransmit packet n and all higher seq # pkts in window

# GBN: sender extended FSM



# GBN: receiver extended FSM



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

sender

send pkt0  
send pkt1  
send pkt2  
send pkt3  
(wait)

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

rcv ack0, send pkt4  
rcv ack1, 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



*pkt 2 timeout*

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

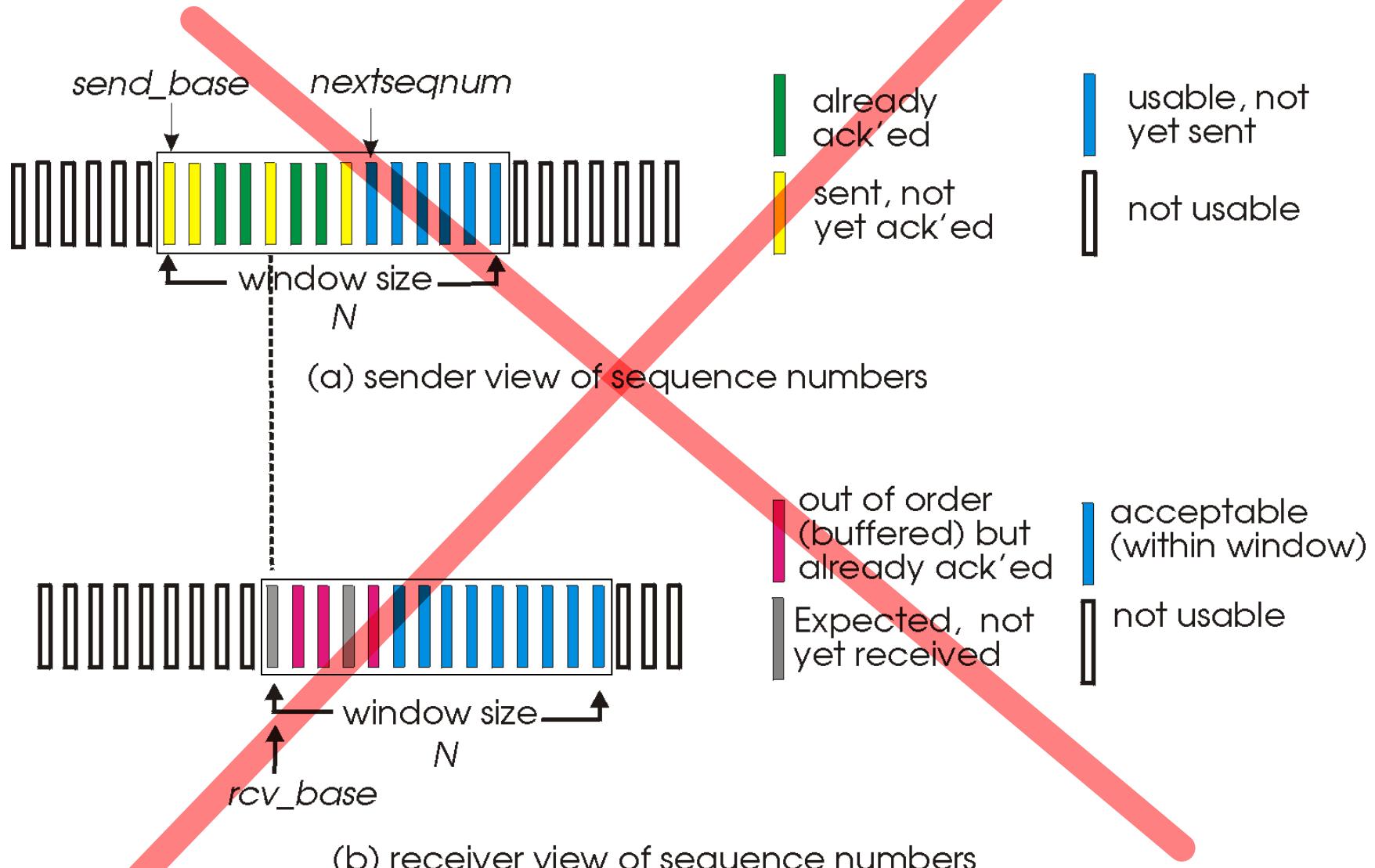
send pkt2  
send pkt3  
send pkt4  
send pkt5

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



# 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 smallest unACKed pkt, advance window base to next unACKed seq #

receiver

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

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

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

- ACK(n)

**otherwise:**

- ignore

# 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
0	1	2	3	4	5	6	7	8

rcv ack0, send pkt4  
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

rcv pkt2; deliver pkt2,  
pkt3, pkt4, pkt5; send ack2

*Q: what happens when ack2 arrives?*

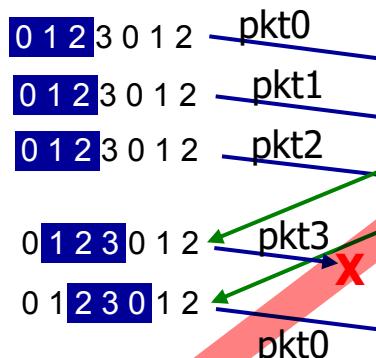
# Selective repeat: dilemma

example:

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

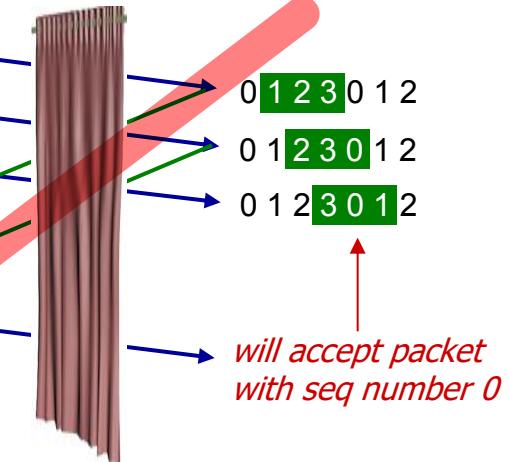
Q: what relationship  
between seq # size  
and window size to  
avoid problem in (b)?

sender window  
(after receipt)

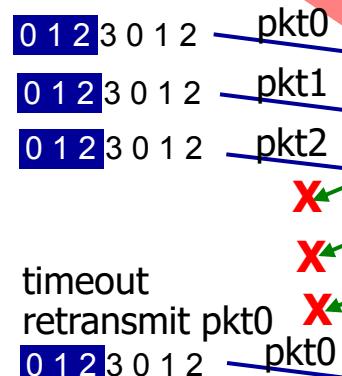


(a) no problem

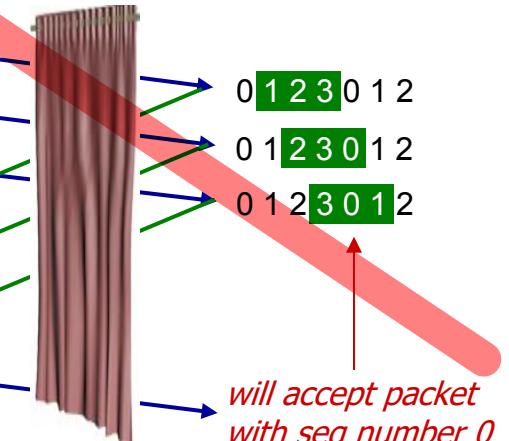
receiver window  
(after receipt)



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

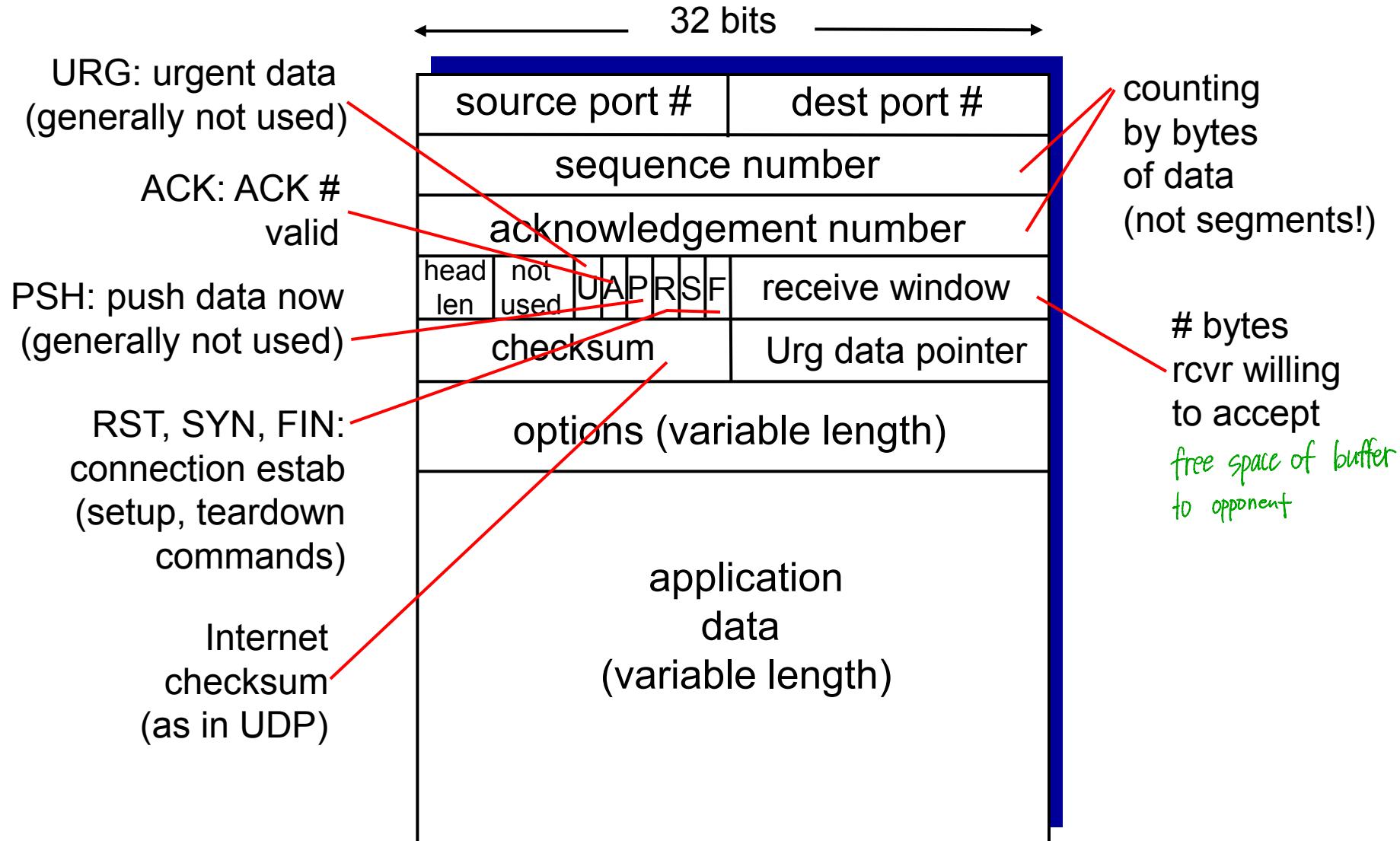
# TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- **point-to-point:**
  - one sender, one receiver
- **reliable, in-order byte steam:**
  - no “message boundaries”
- **pipelined:**
  - TCP congestion and flow control set window size

flow control  
congestion control
- **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

data byte count number?

## 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

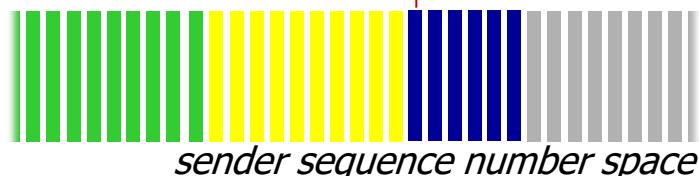
- A: TCP spec doesn’t say,  
- up to implementor

outgoing segment from sender

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

window size

$N$



sent  
ACKed

sent, not-  
yet ACKed  
("in-  
flight")

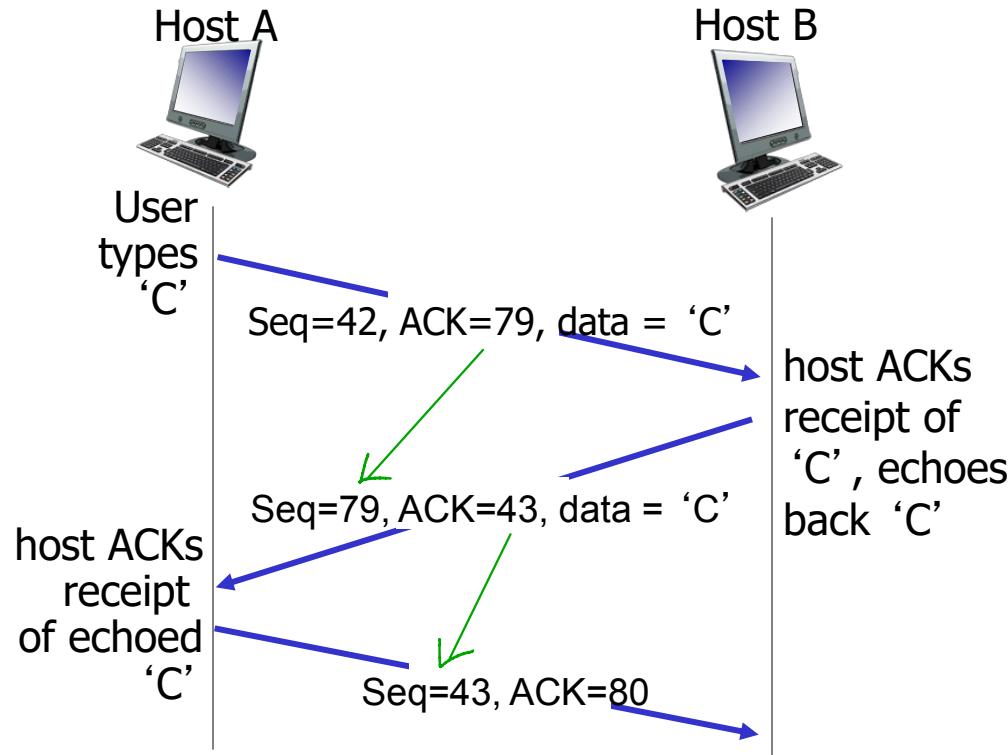
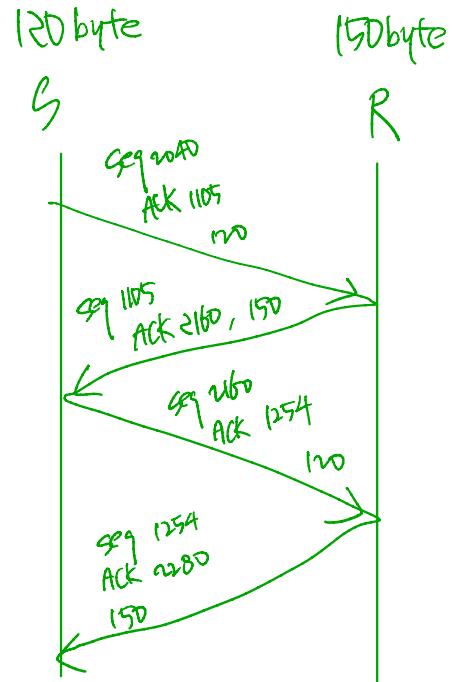
usable  
but not  
yet sent

not  
usable

incoming segment to sender

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

# TCP seq. numbers, ACKs



simple telnet scenario

# TCP round trip time, timeout

RTT measured by ping

avoid deadlock

**Q:** how to set TCP timeout value?

- longer than RTT
  - but RTT varies not constant
- 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**

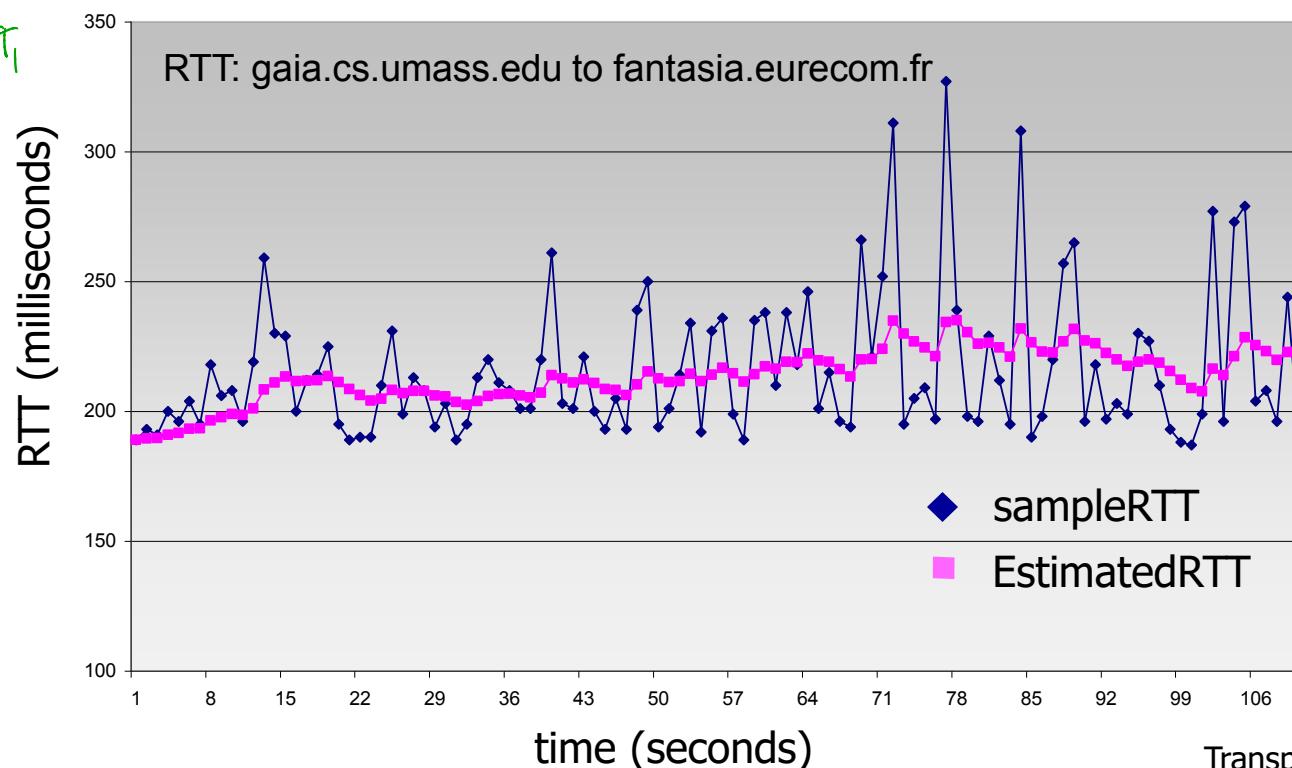
# TCP round trip time, timeout

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

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

SampleRTT<sub>v</sub> = EstimatedRTT<sub>v</sub>

EstimatedRTT<sub>v+1</sub>  
= (1- $\alpha$ ) EstimatedRTT<sub>v</sub>  
+  $\alpha$  SampleRTT<sub>v</sub>



# 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”

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# 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  $\text{Seq } \#$
  - cumulative acks  $\text{Ack } \#$
  - single retransmission timer  $\text{Timeout}$
- 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:*

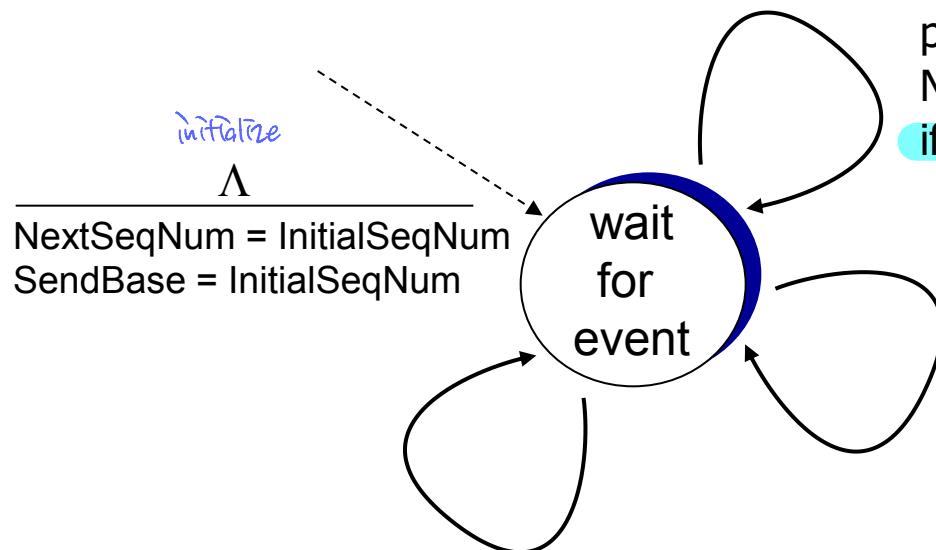
- retransmit segment that caused timeout

- 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 sender (simplified)



data received from application above  
create segment, seq. #: NextSeqNum  
pass segment to IP (i.e., "send")  
 $\text{NextSeqNum} = \text{NextSeqNum} + \text{length(data)}$   
if (timer currently not running) ) suppose Simple Timer  
start timer

timeout  $\Rightarrow$  retransmission.  
retransmit not-yet-acked segment  
with smallest seq. #  
start timer the segment owns timer

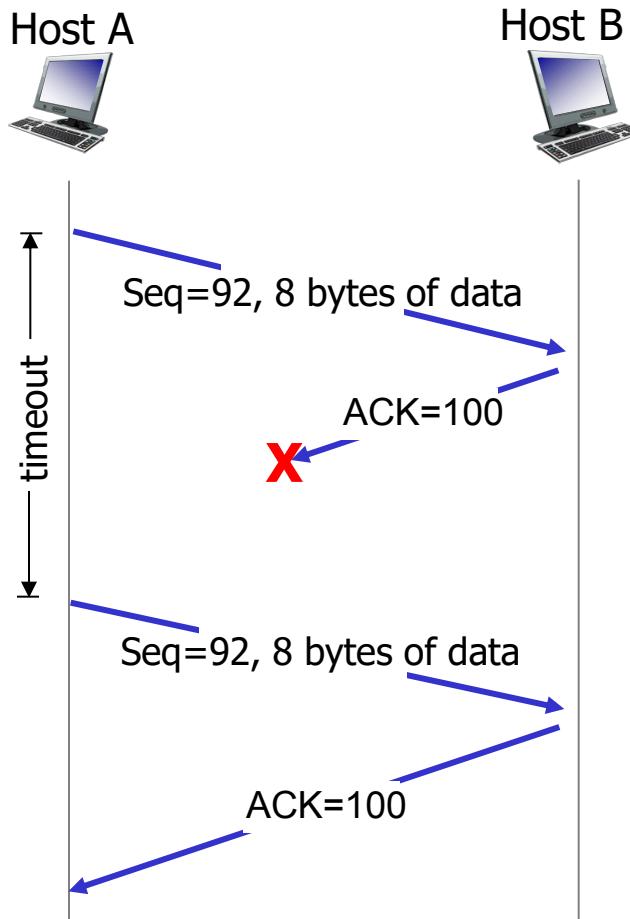
ACK received, with ACK field value y

---

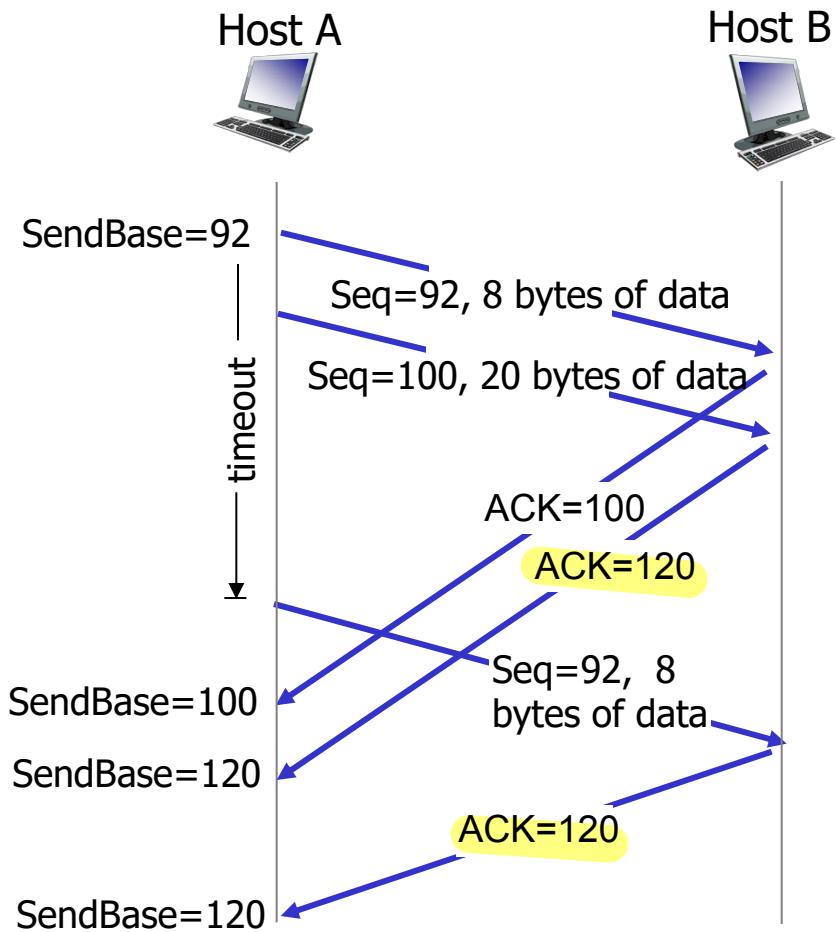
```
if (y > SendBase) { \Rightarrow ACK from new segment.
    SendBase = y
    /* SendBase-1: last cumulatively ACKed byte */
    if (there are currently not-yet-acked segments)
        start timer assigned to not-yet-acked seg
    else stop timer
}
else \Rightarrow duplicate ACK ... loss of seg
```

*View of Sender*

# TCP: retransmission scenarios

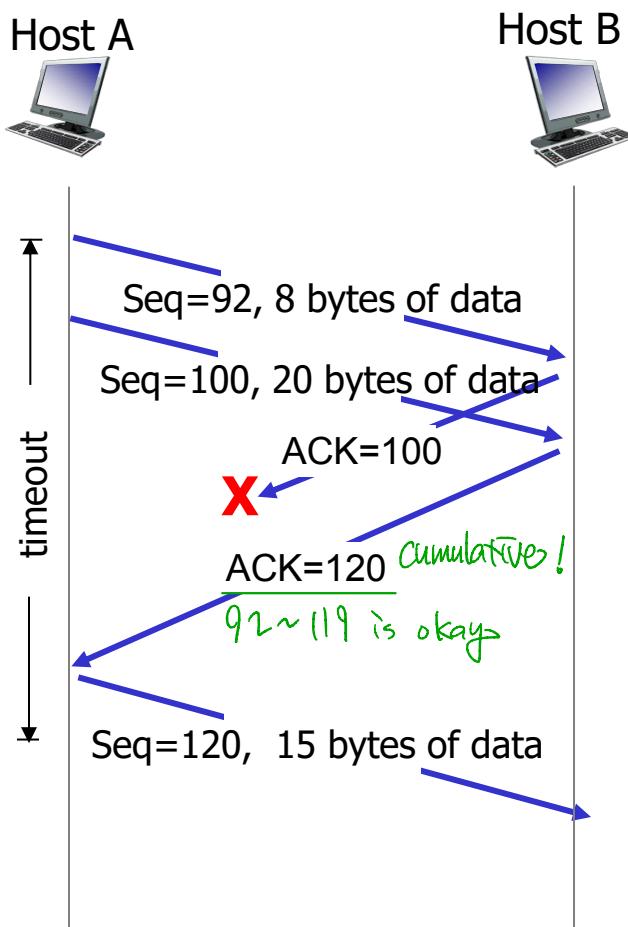


lost ACK scenario



premature timeout

# TCP: retransmission scenarios



cumulative ACK

# 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

y = Genbase

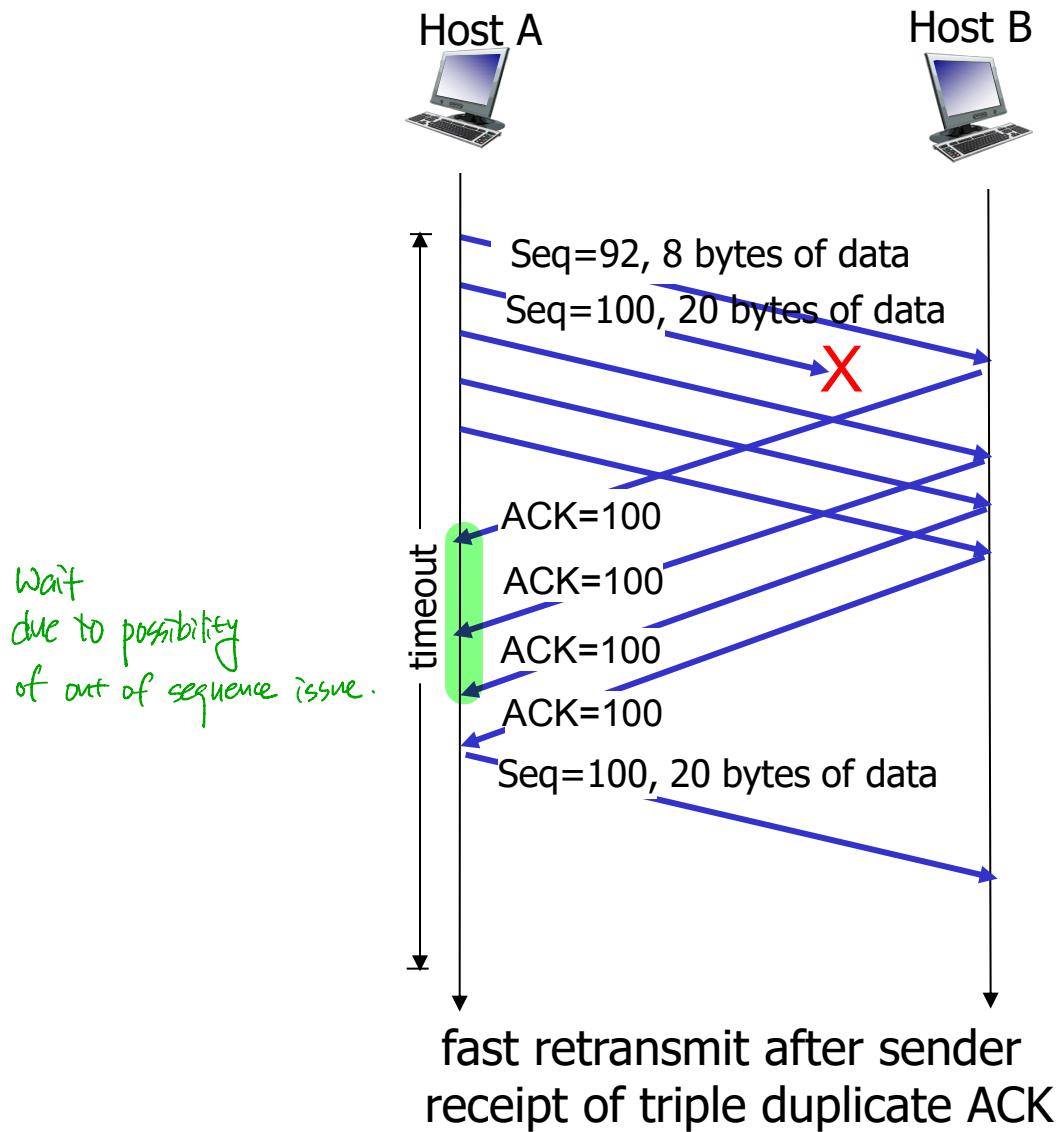
# TCP fast retransmit

- 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



fast retransmit  
algorithm  
↓  
~~pic~~ else part

# 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

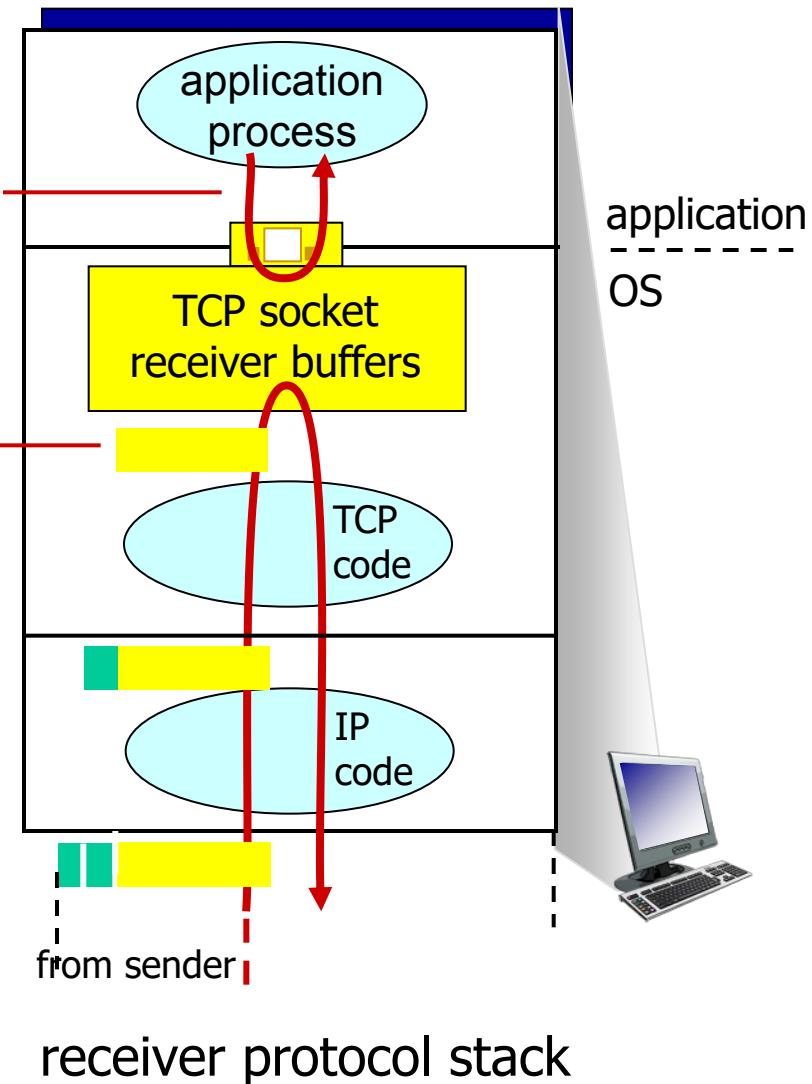
3.7 TCP congestion control

# TCP flow control

application may  
remove data from  
TCP socket buffers ....

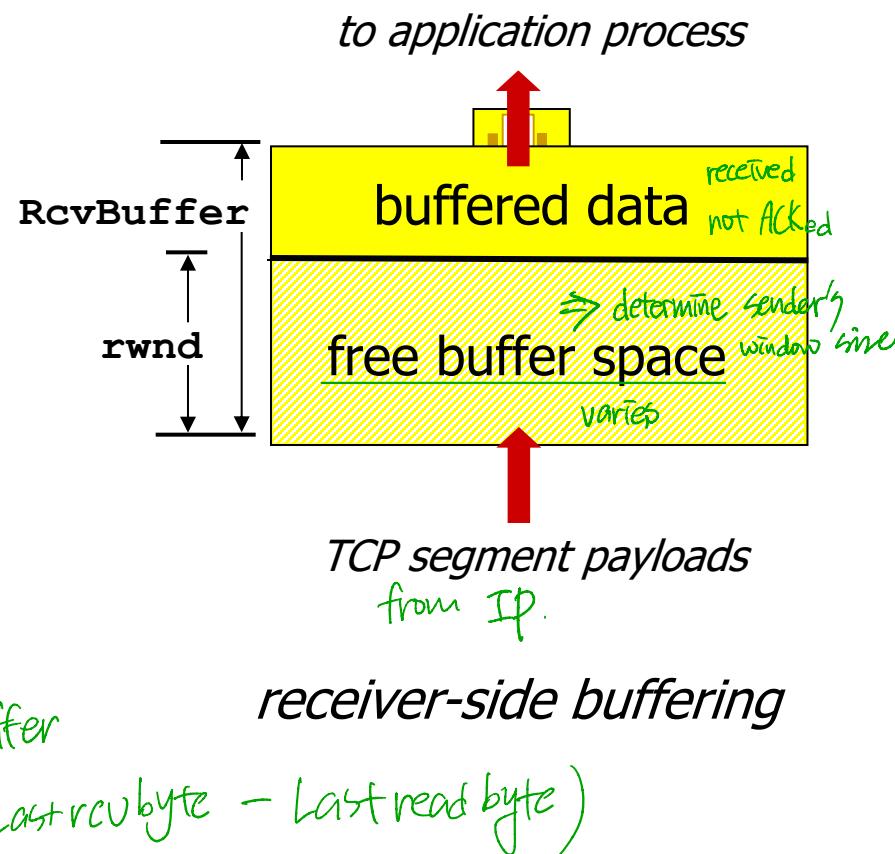
... slower than TCP  
receiver is delivering  
(sender is sending)

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



# 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 =  $RcvBuffer - (Last\ rcv\ byte - Last\ read\ byte)$
- guarantees receive buffer will not overflow



# 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

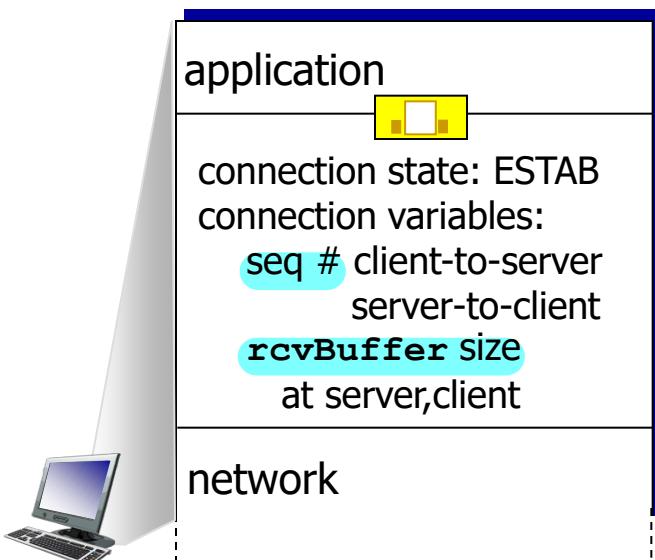
3.6 principles of congestion control

3.7 TCP congestion control

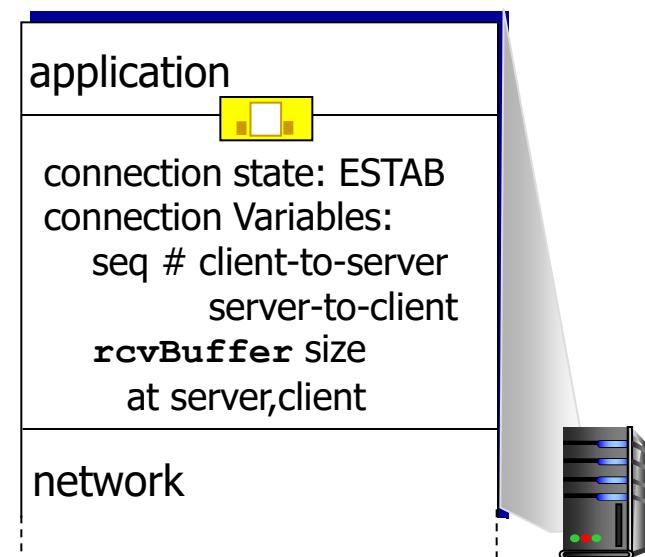
# Connection Management

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");
```

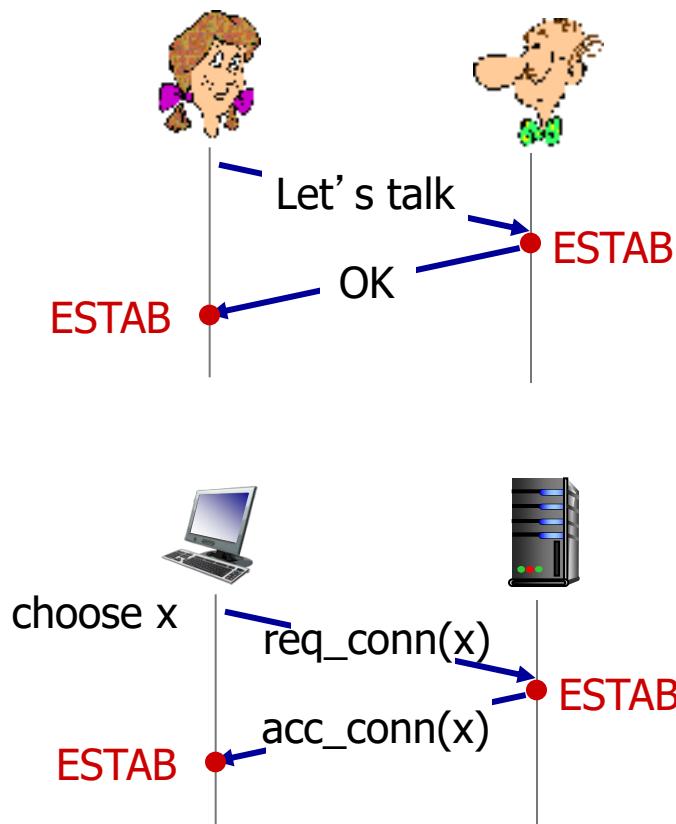


```
Socket connectionSocket =  
    welcomeSocket.accept();
```

# Agreeing to establish a connection

X

2-way handshake:



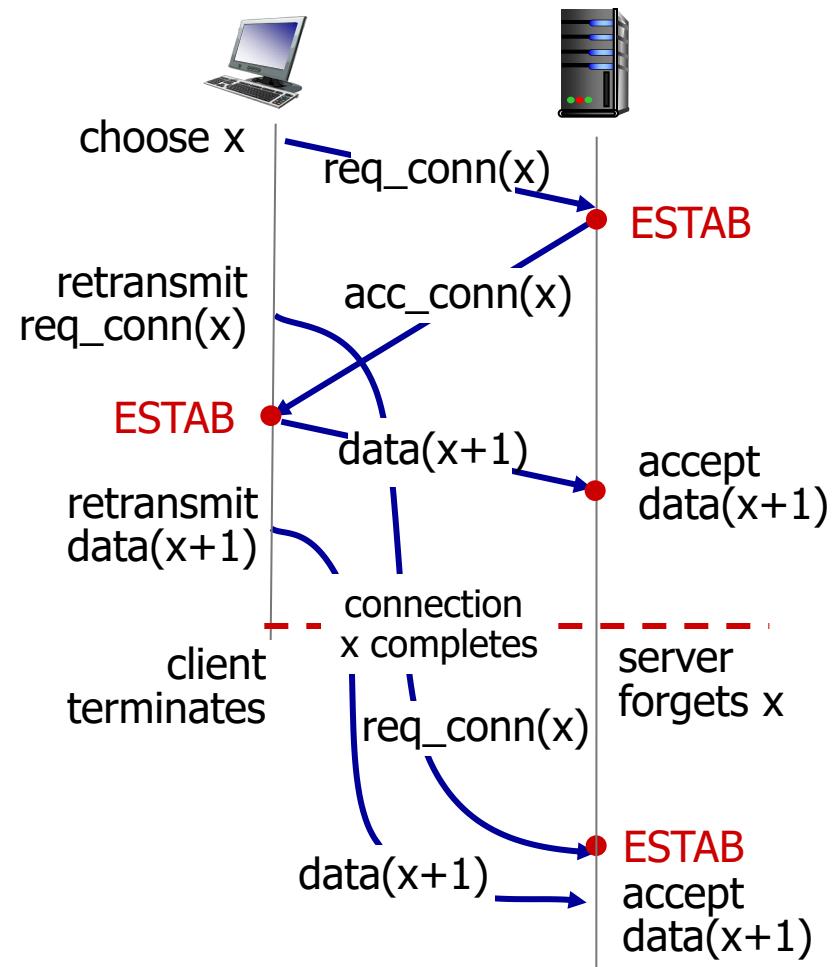
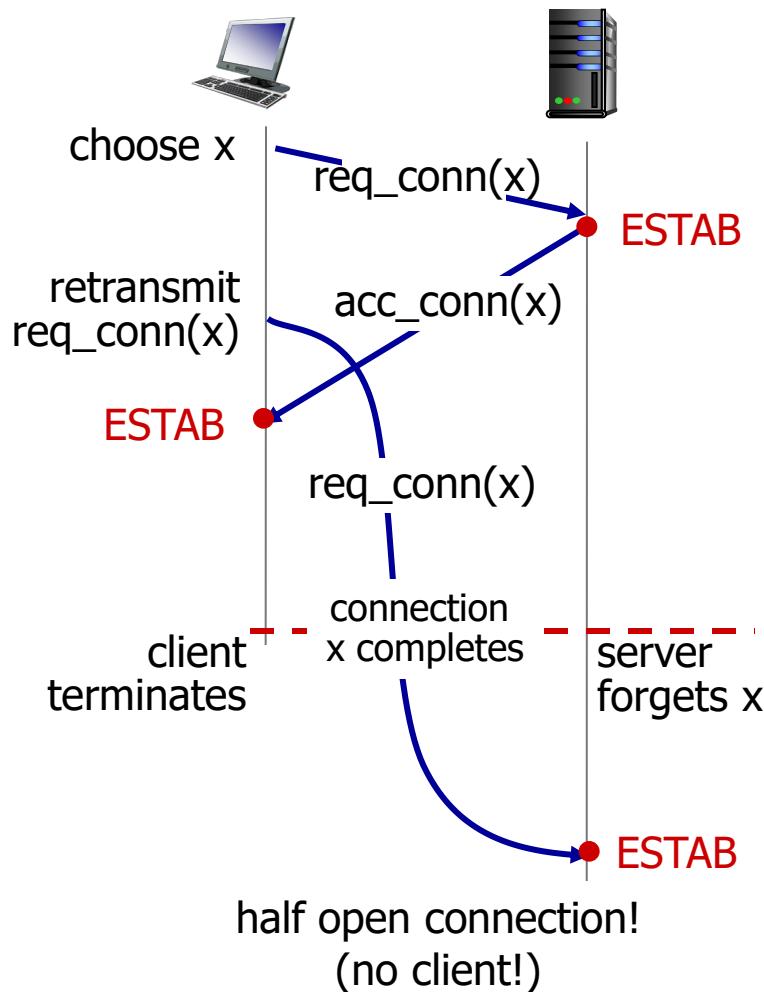
**Q:** will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. `req_conn(x)`) due to message loss
- message reordering
- can't "see" other side

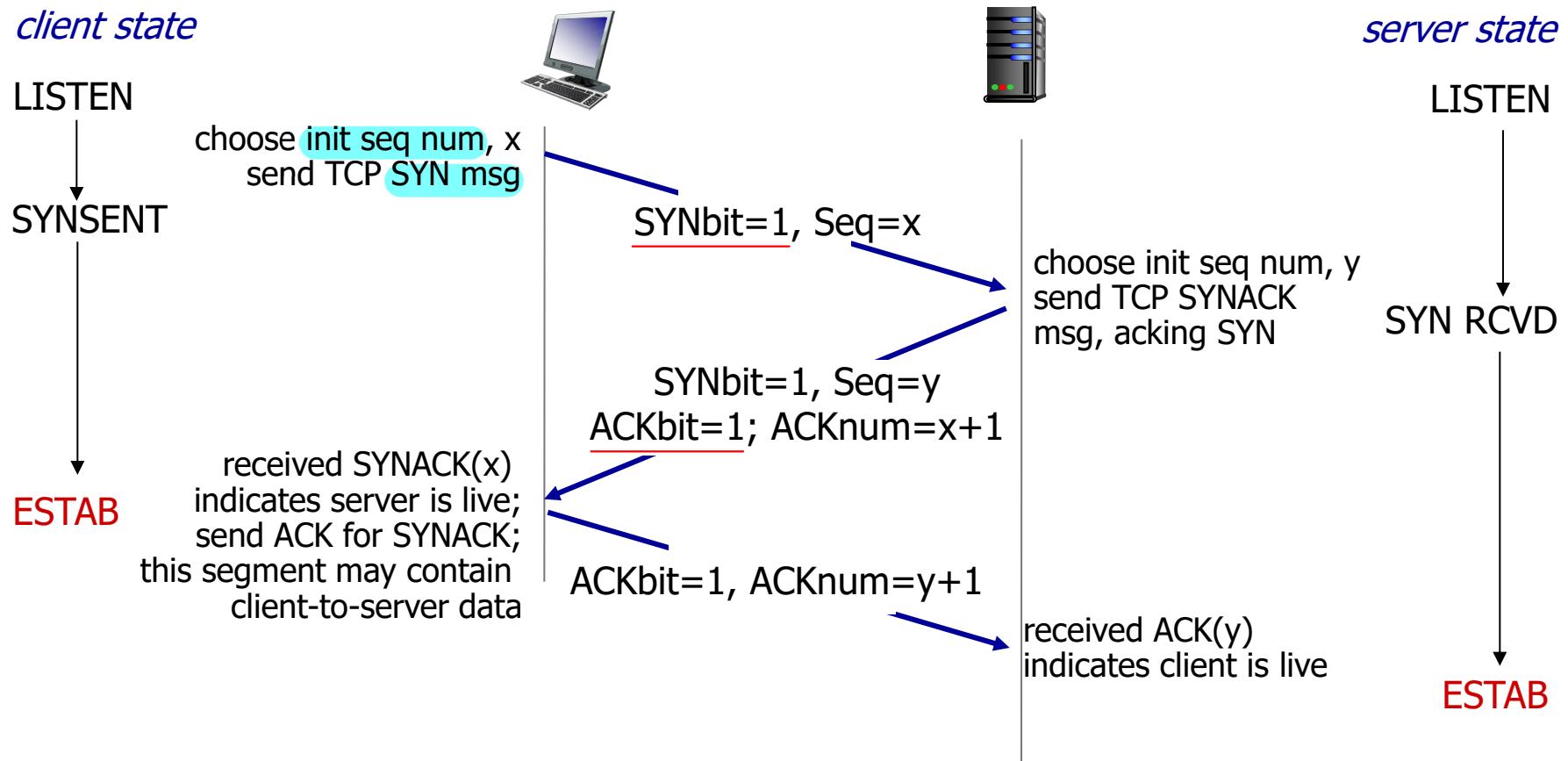
# Agreeing to establish a connection



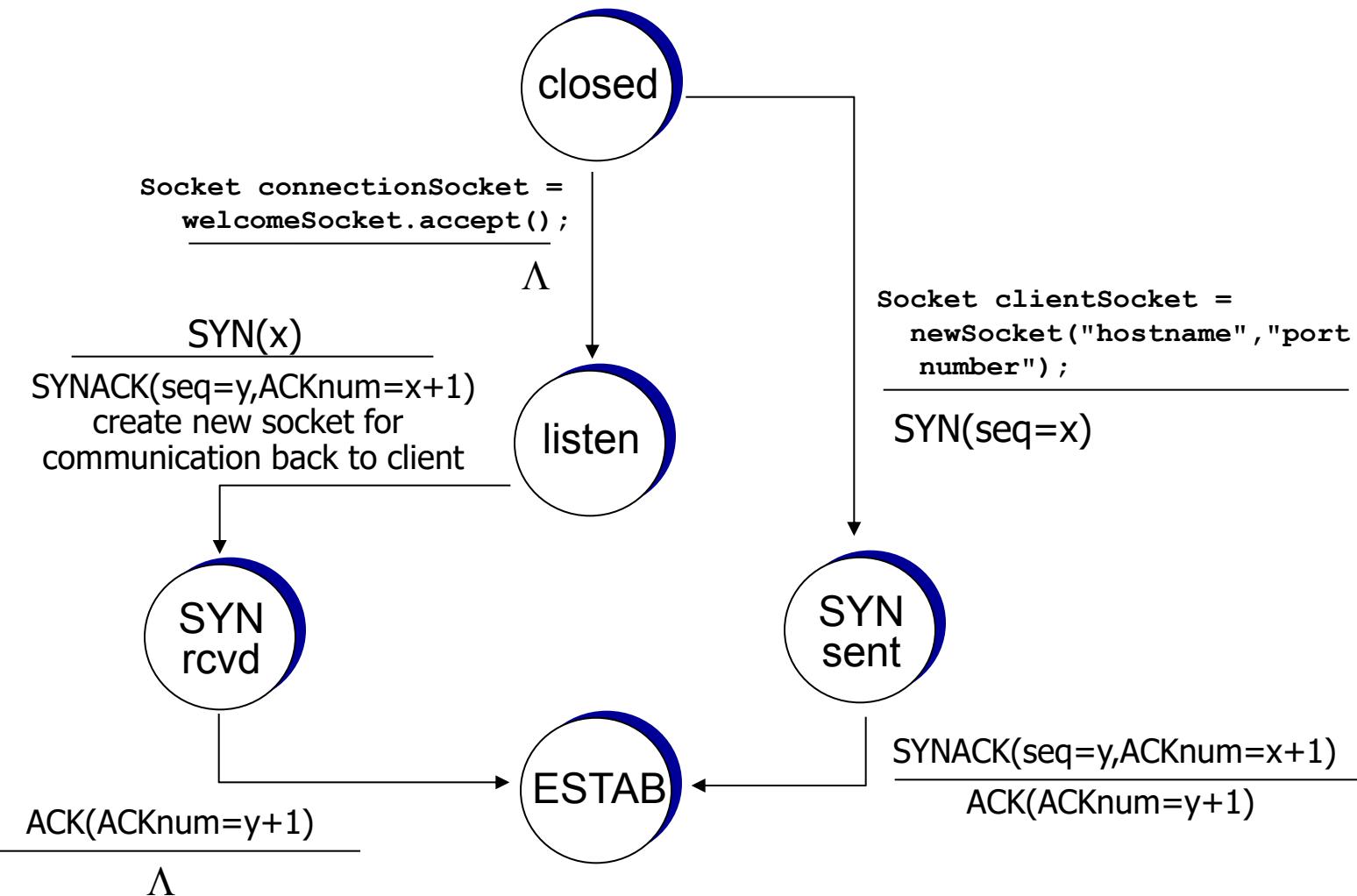
2-way handshake failure scenarios:



# TCP 3-way handshake



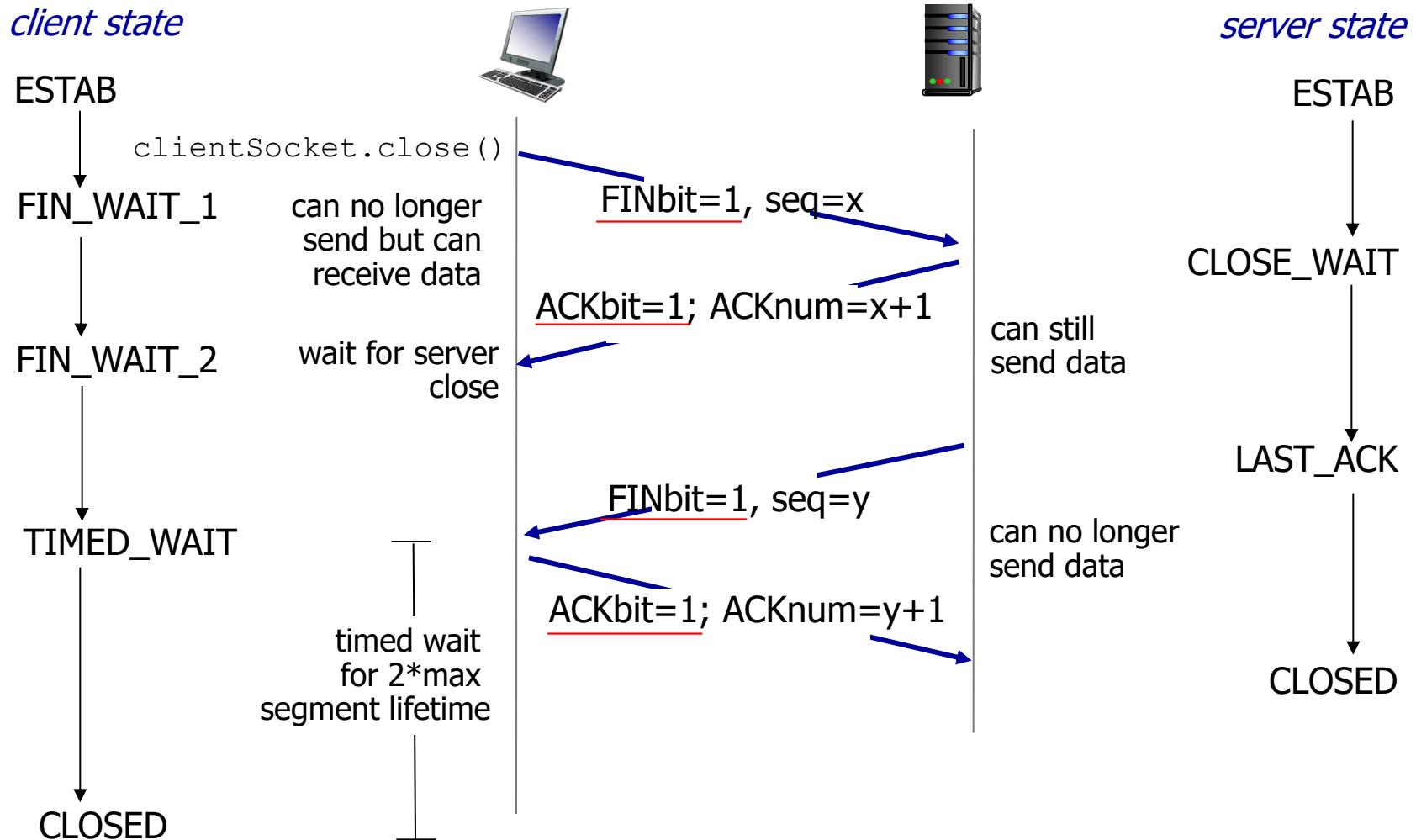
# TCP 3-way handshake: FSM



# 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

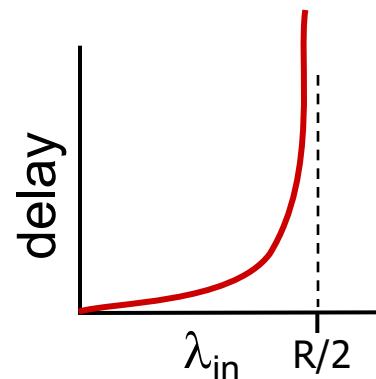
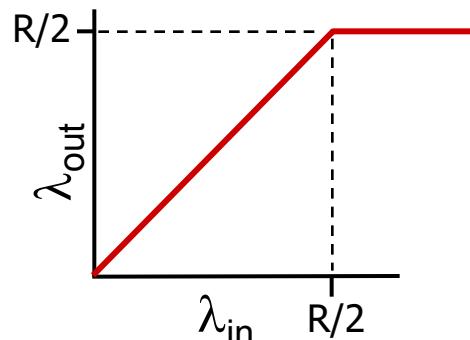
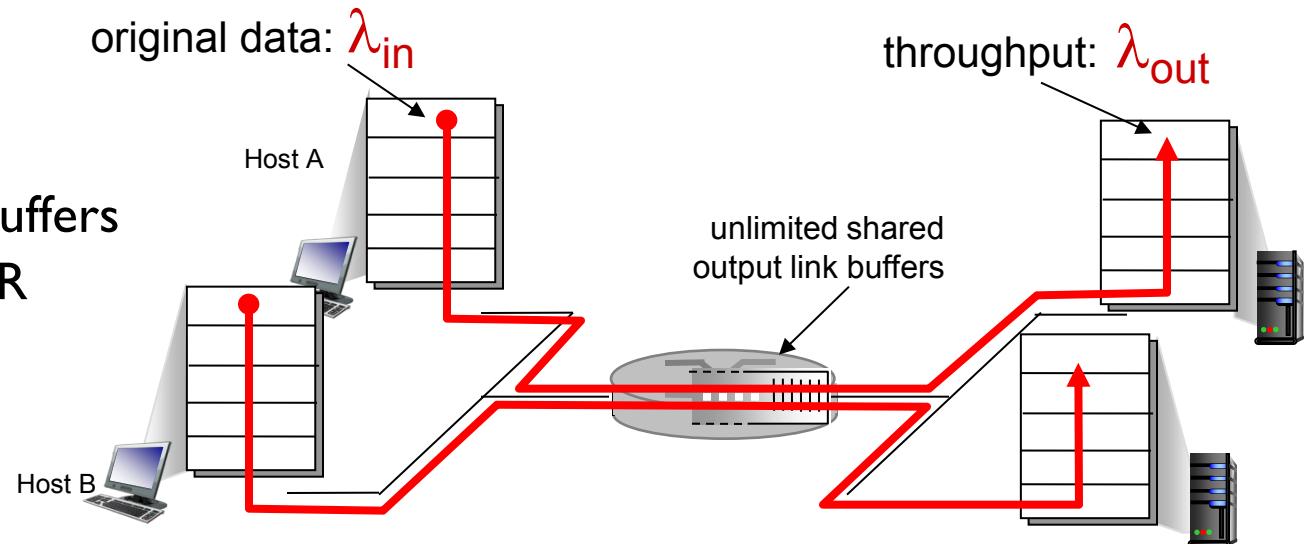
# Principles of congestion control

*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

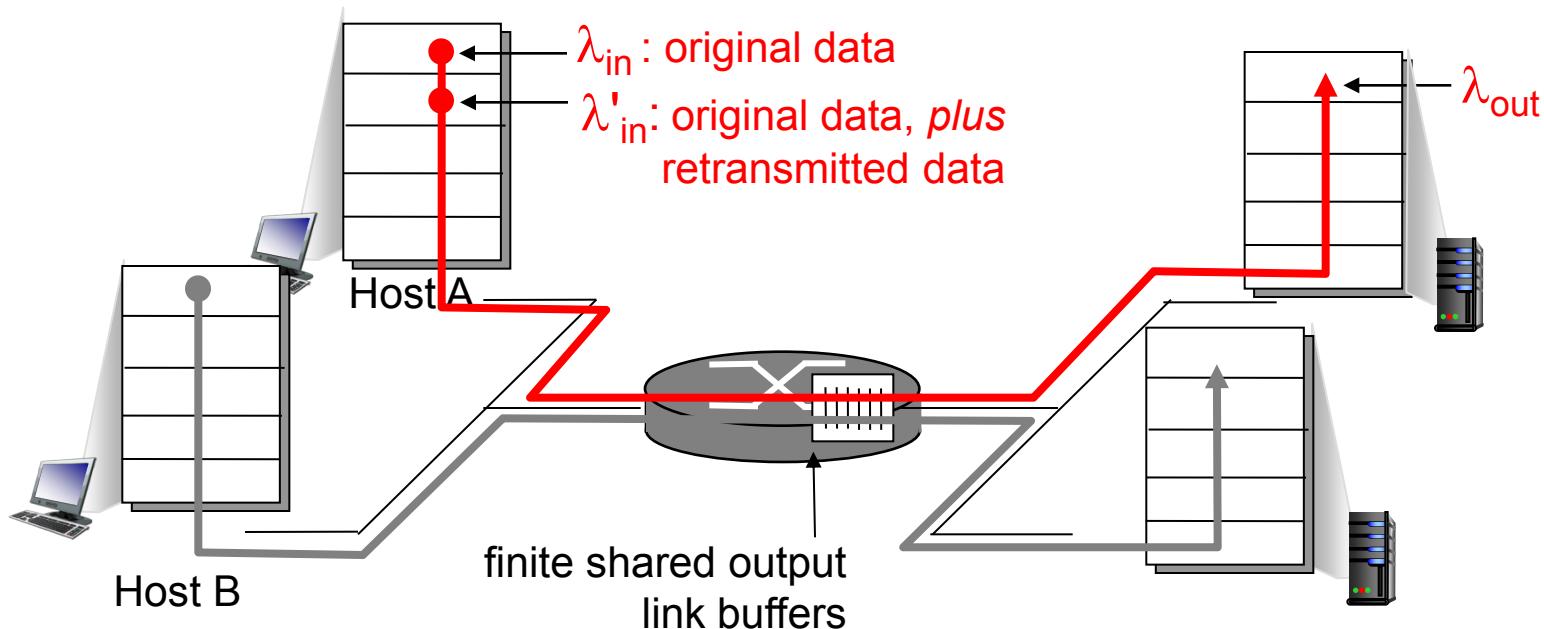
- two senders, two receivers
- one router, infinite buffers
- output link capacity:  $R$
- no retransmission



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

# Causes/costs of congestion: scenario 2

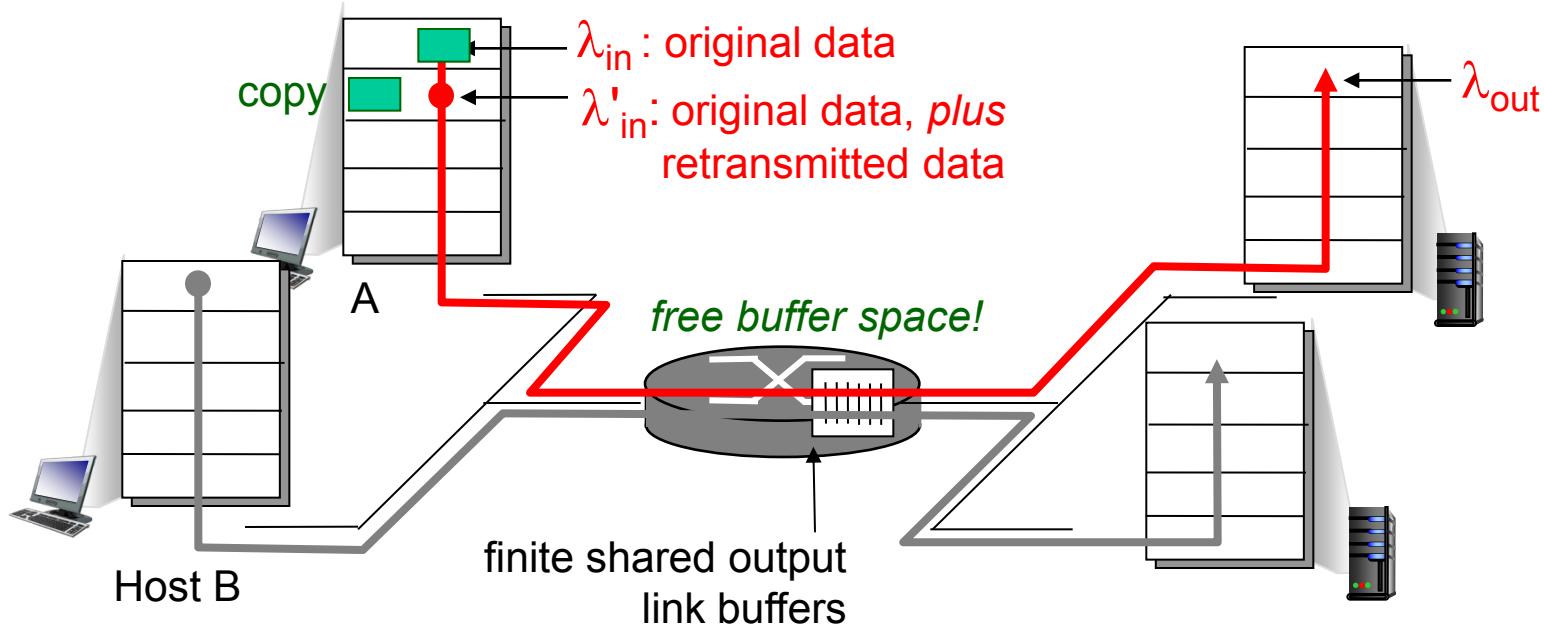
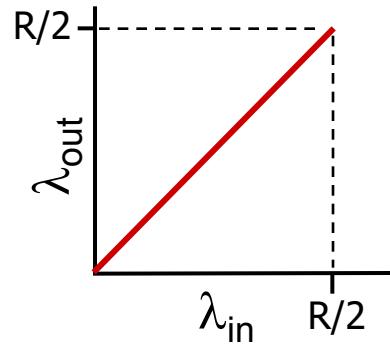
- one router, *finite* buffers
- sender retransmission of timed-out packet
  - application-layer input = application-layer output:  $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* :  $\lambda'_{in} \geq \lambda_{in}$



# Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- sender sends only when router buffers available

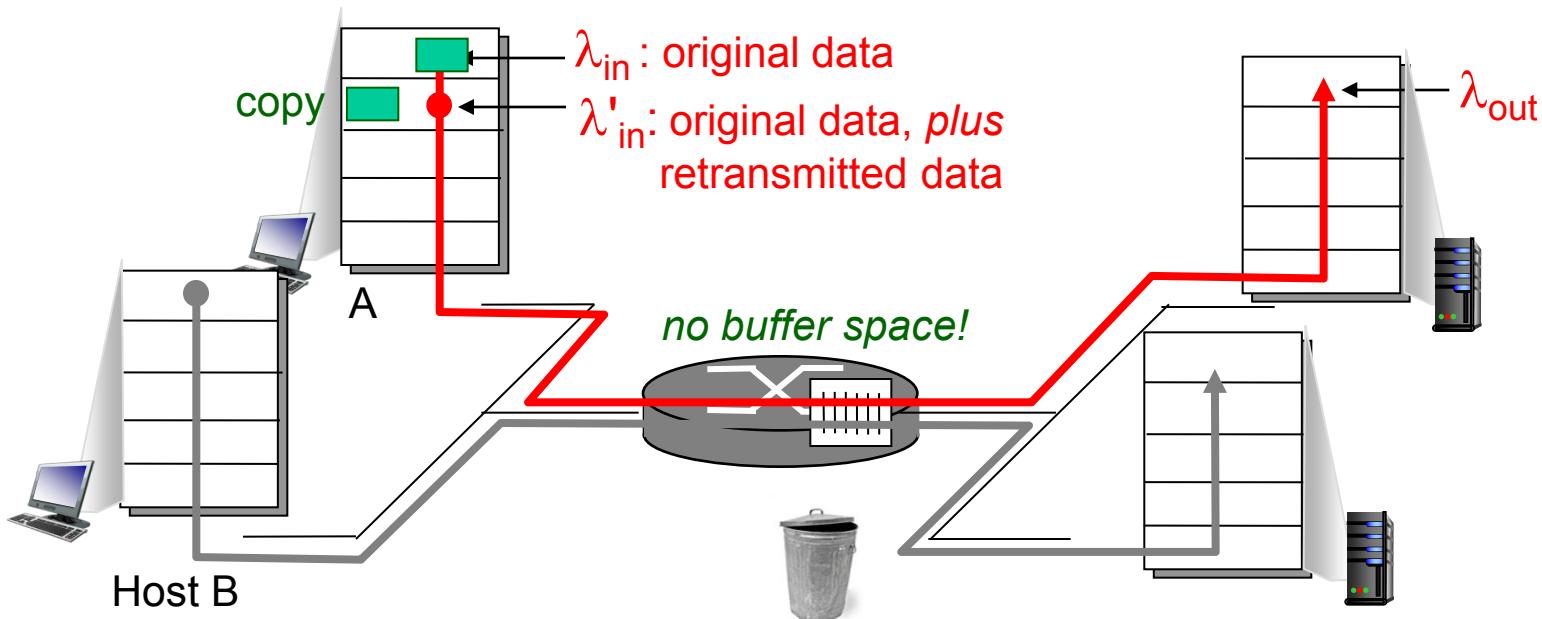


# Causes/costs of congestion: scenario 2

Idealization: *known loss*

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

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

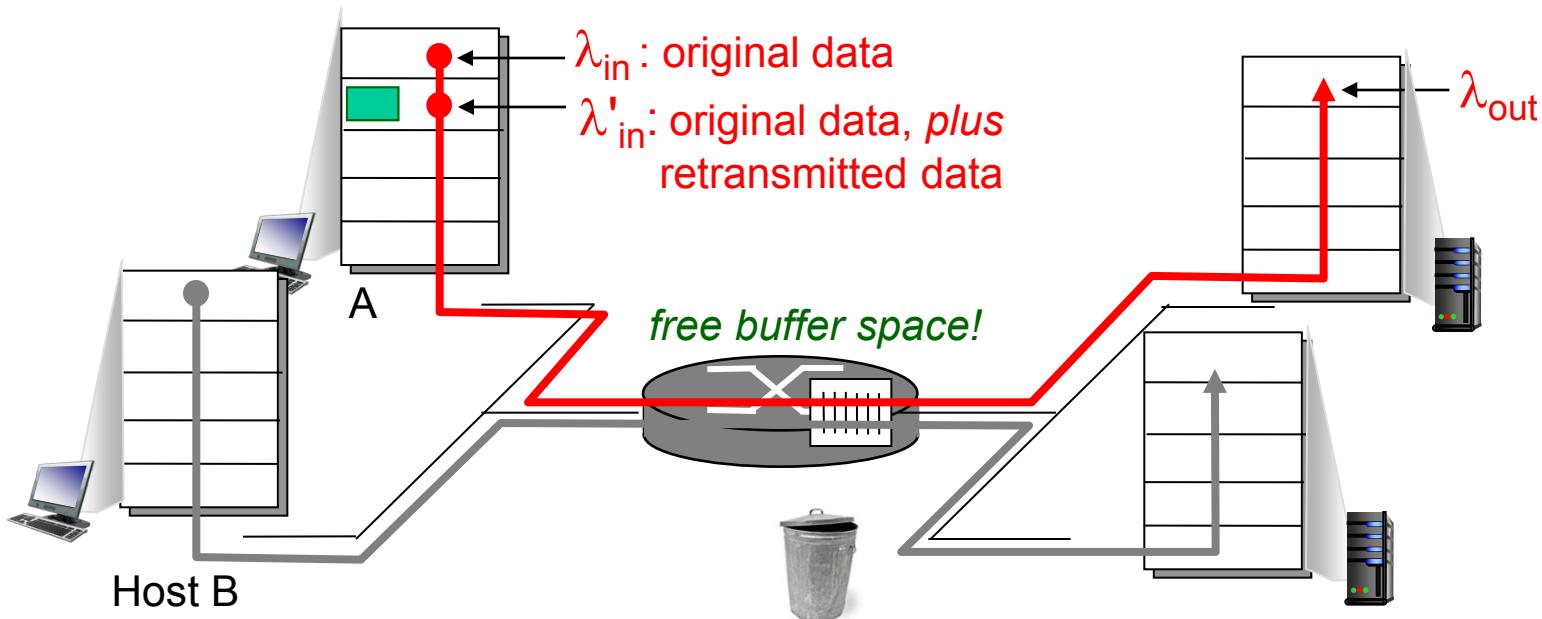
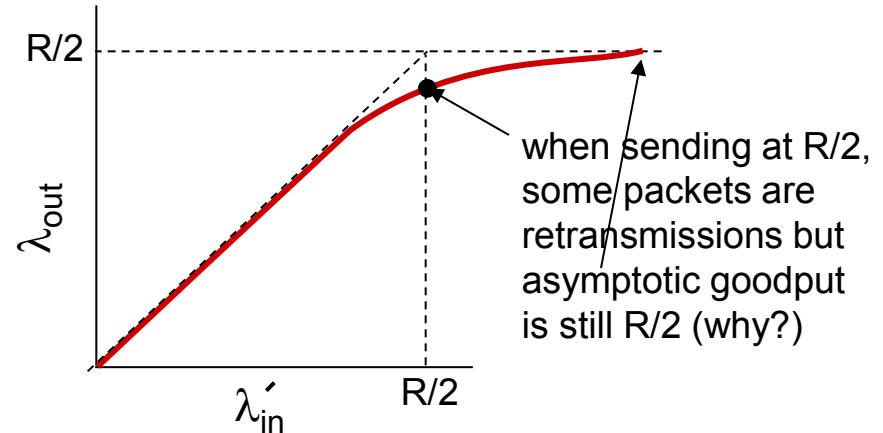


# Causes/costs of congestion: scenario 2

Idealization: *known loss*

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

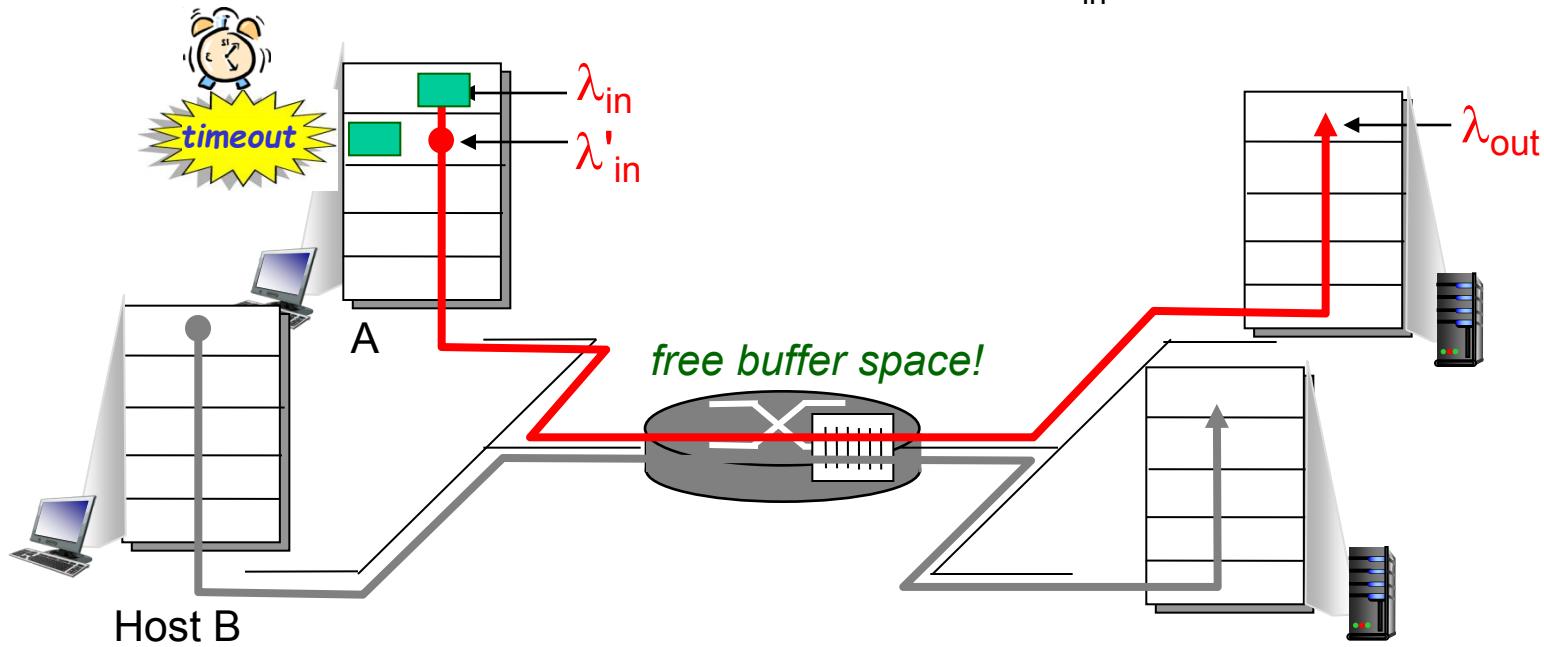
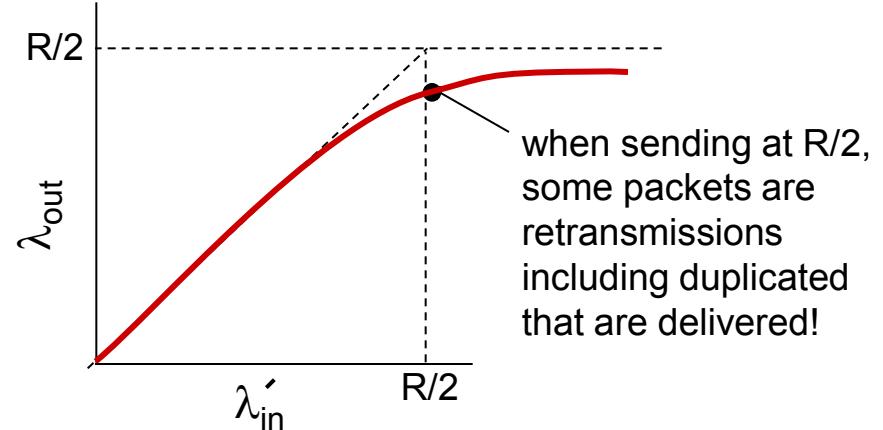
- sender only resends if  
packet *known* to be lost



# Causes/costs of congestion: scenario 2

## Realistic: *duplicates*

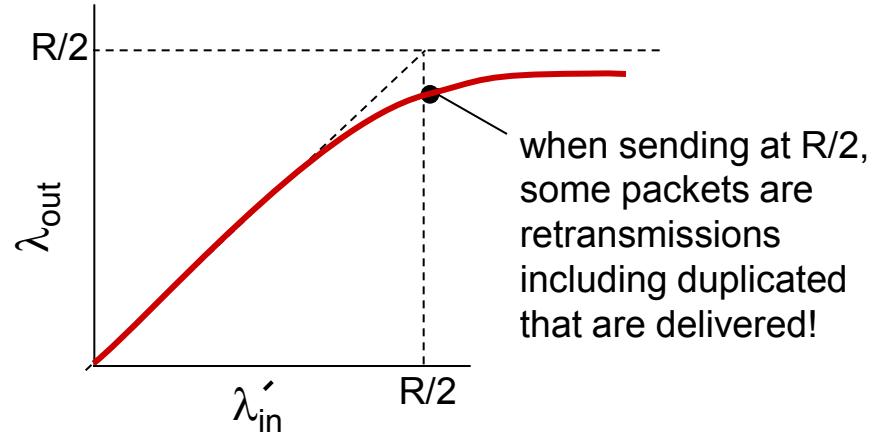
- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered



# 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:

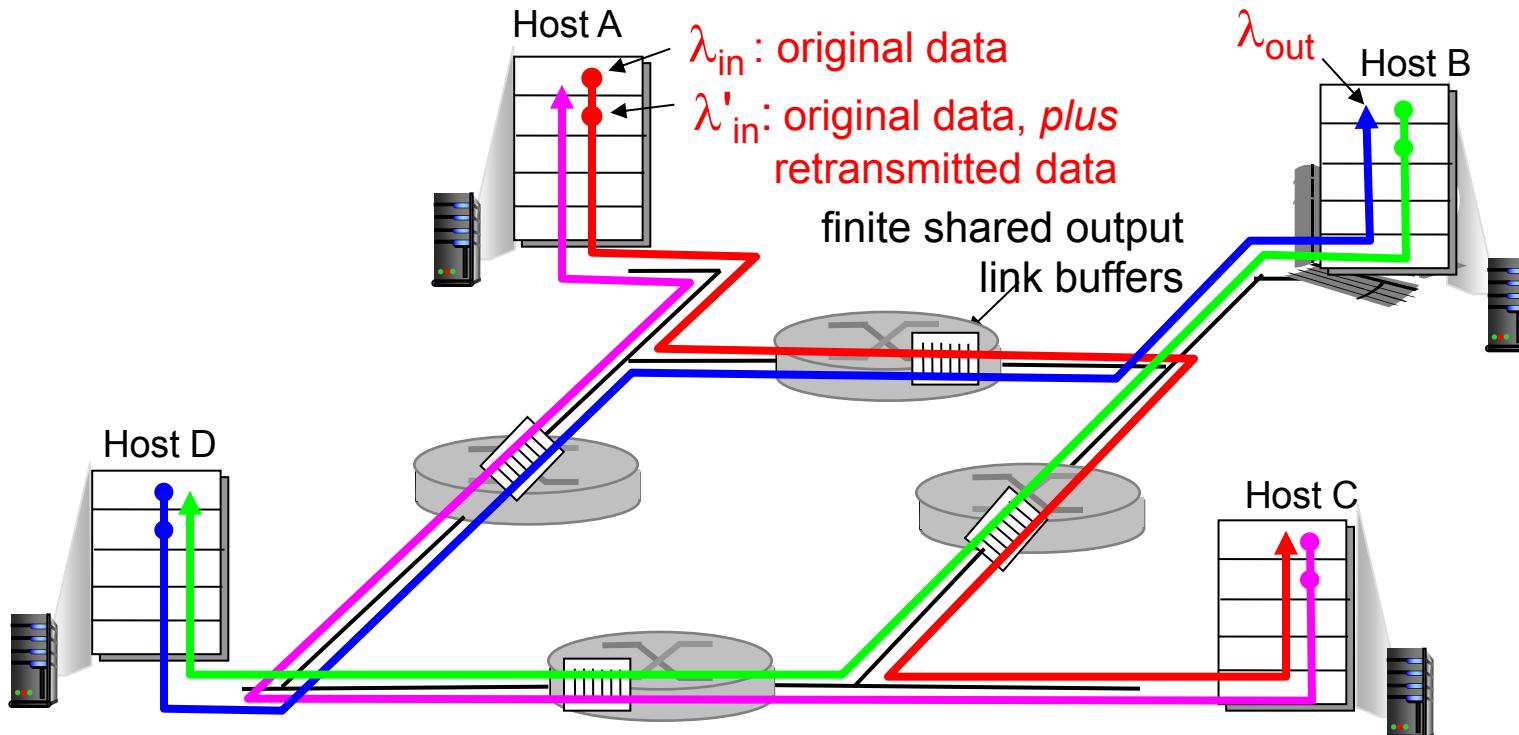
- more work (retrans) for given “goodput”
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

# Causes/costs of congestion: scenario 3

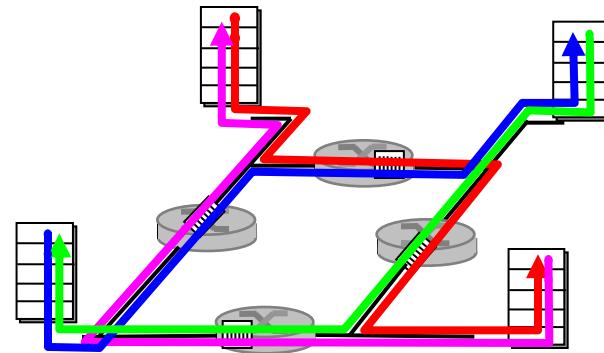
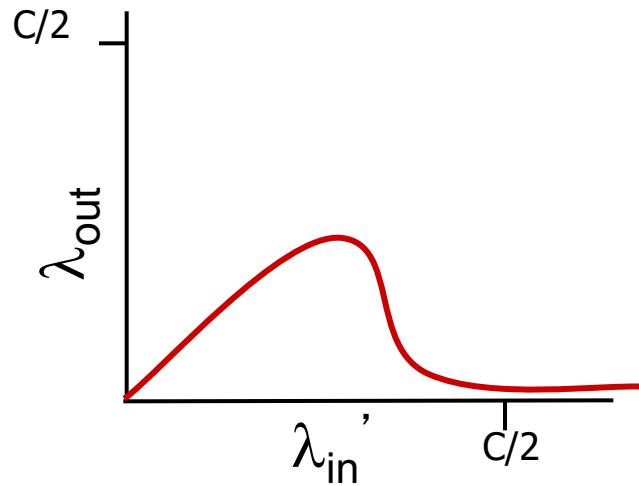
- four senders
- multihop paths
- timeout/retransmit

Q: what happens as  $\lambda_{in}$  and  $\lambda'_{in}$  increase ?

A: as red  $\lambda_{in}$  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!

# 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

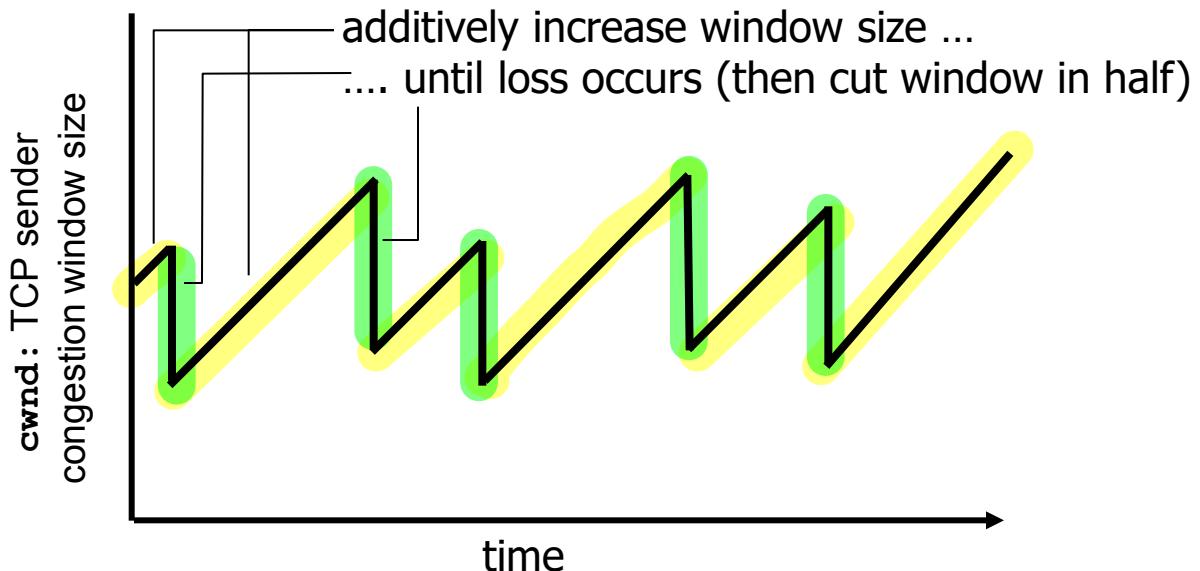


inside of network

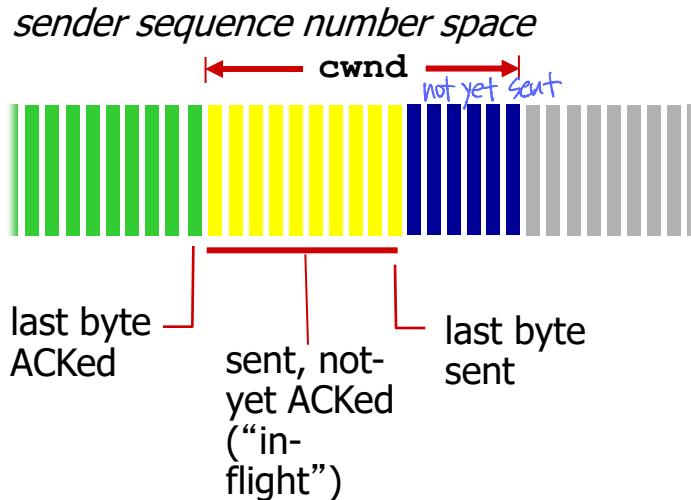
# TCP congestion control: additive increase multiplicative decrease

- **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:**

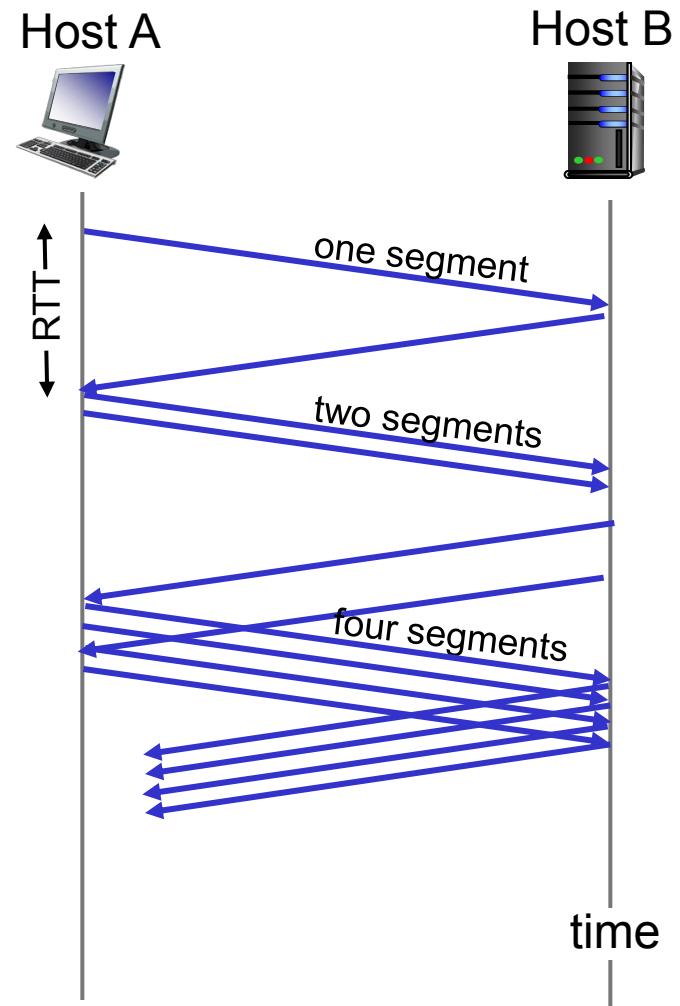
- roughly: send cwnd bytes, wait RTT for ACKS, then send more bytes

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

- **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



# How TCP sender perceive network congestion

Timeout : more seconds  
↳ duplicate ACK

## 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)

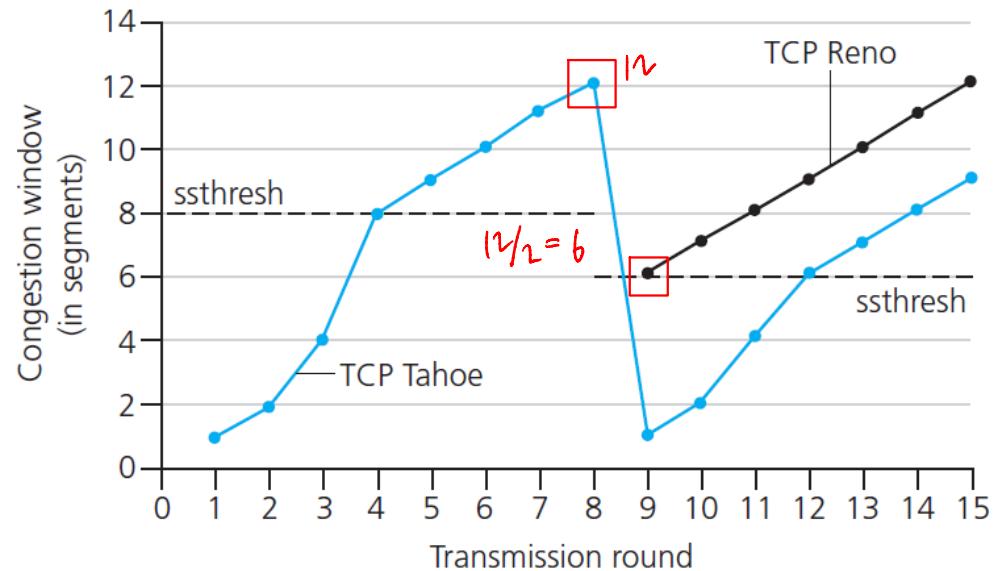
# TCP: switching from slow start to CA

**Q:** when should the exponential increase switch to linear?

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

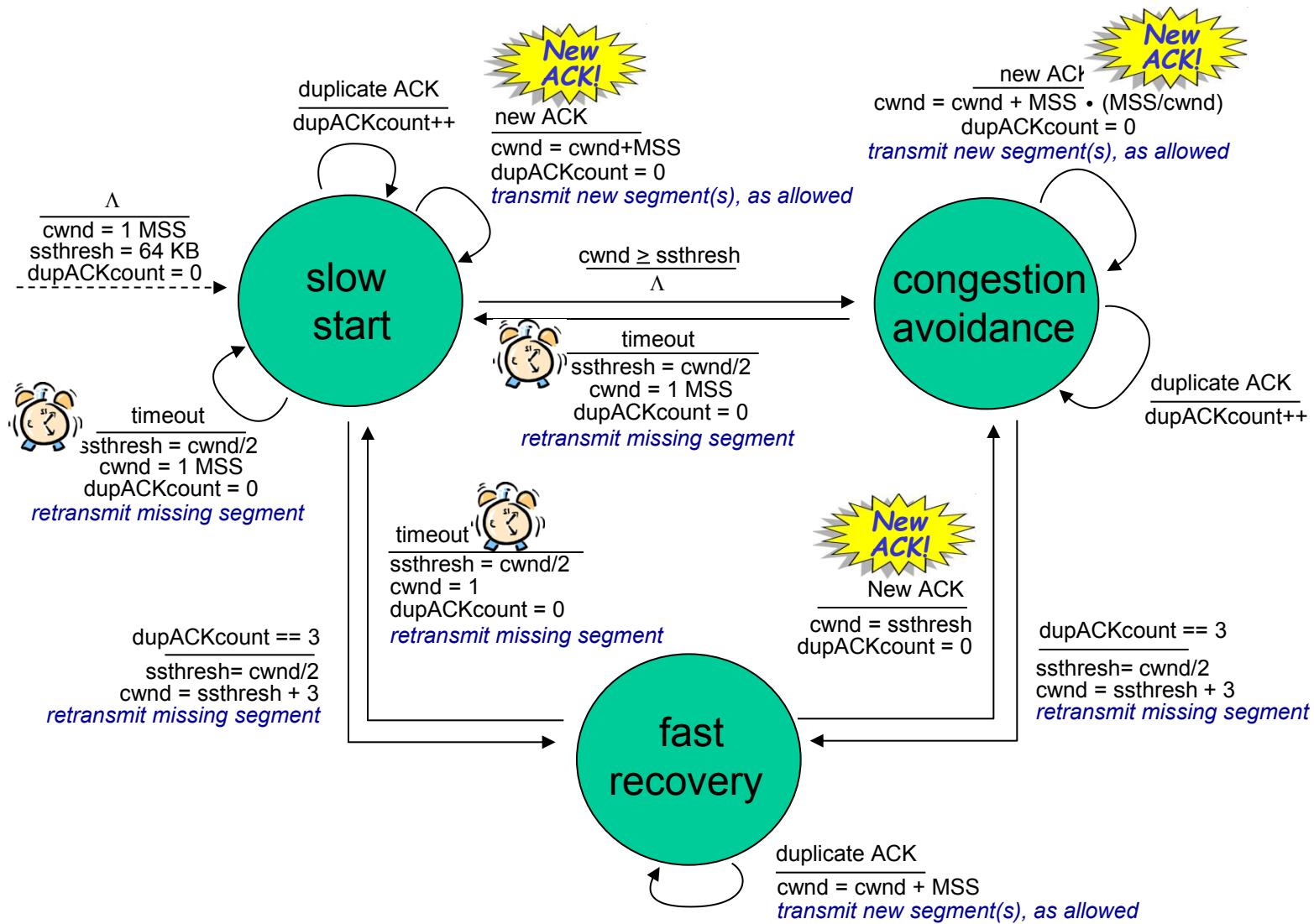
## Implementation:

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



\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

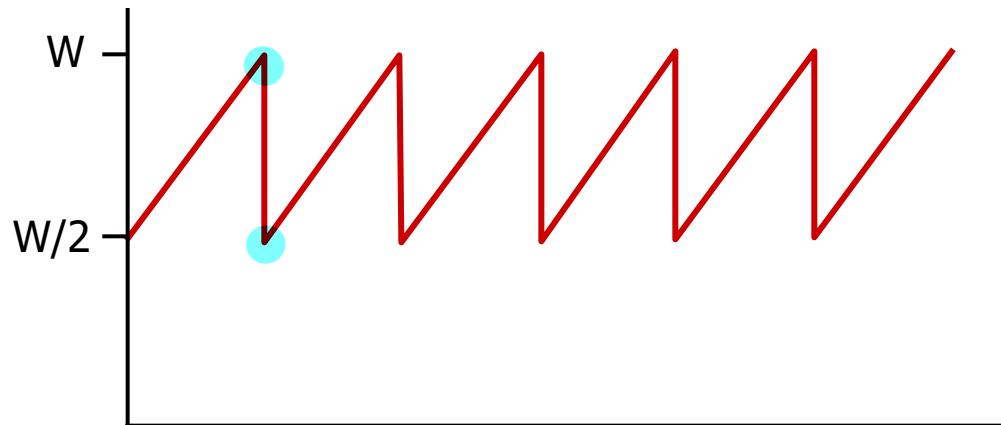
# Summary: TCP Congestion Control



# TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume always data to send
- **W**: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is  $\frac{3}{4} W = (W + \frac{1}{2}W)/2$
  - avg. thruput is  $\frac{3}{4}W$  per RTT

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



## X 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]:

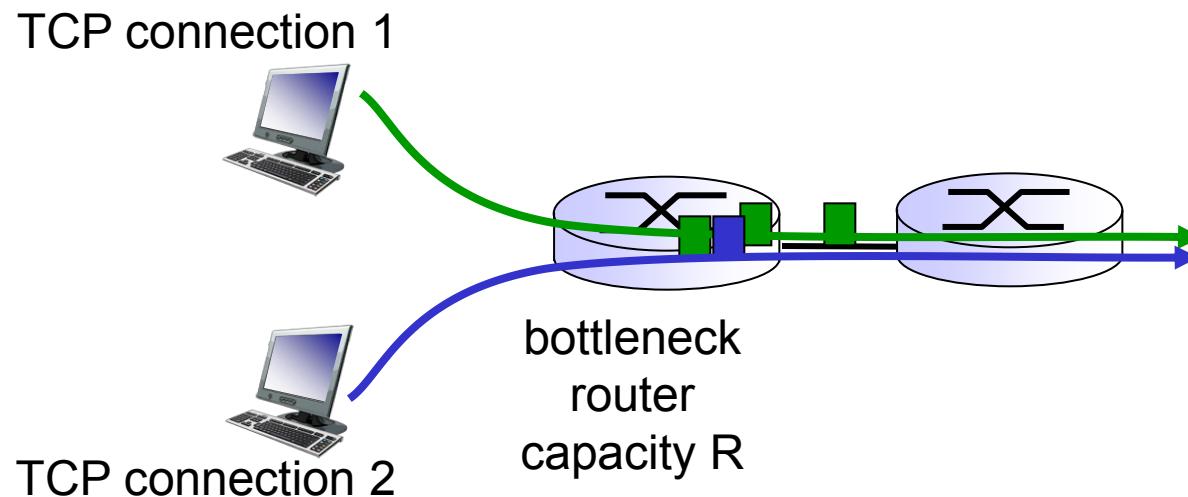
$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

→ 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

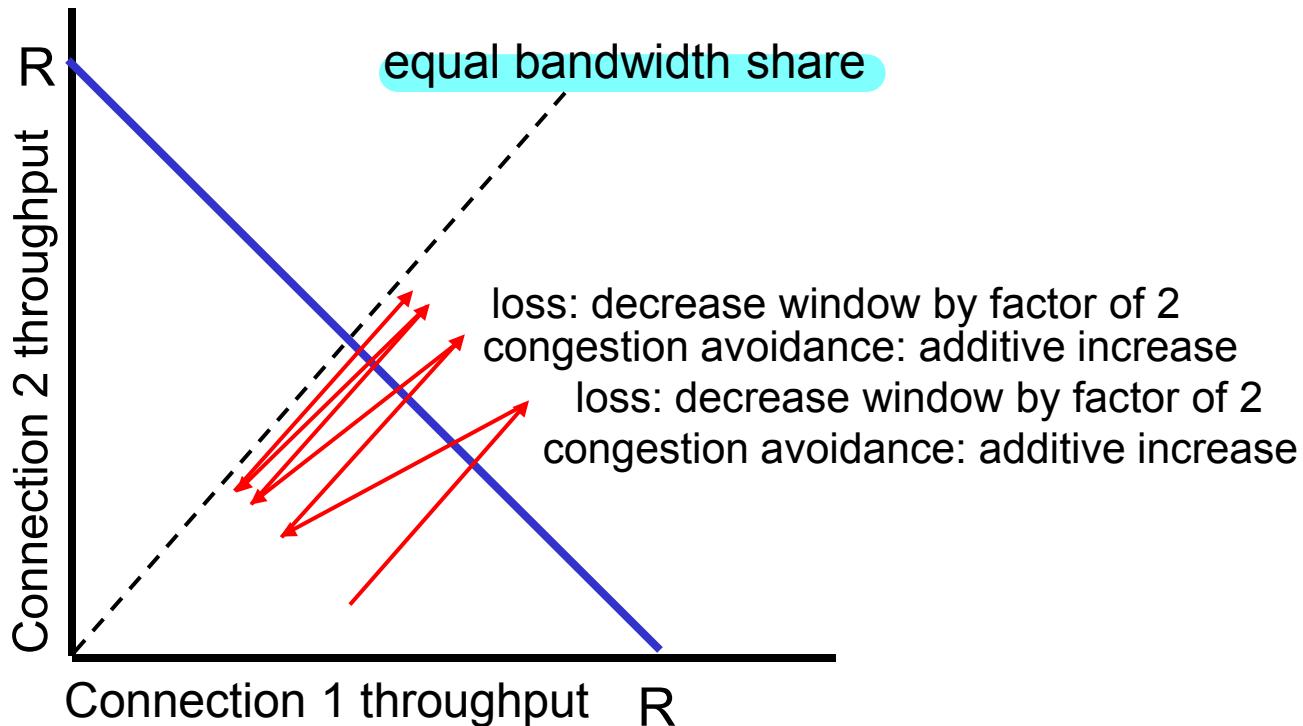
*fairness goal:* if  $K$  TCP sessions share same bottleneck link of bandwidth  $R$ , each should have average rate of  $R/K$



# Why is TCP fair?

two competing sessions:

- additive increase gives slope of 1, as throughput 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*

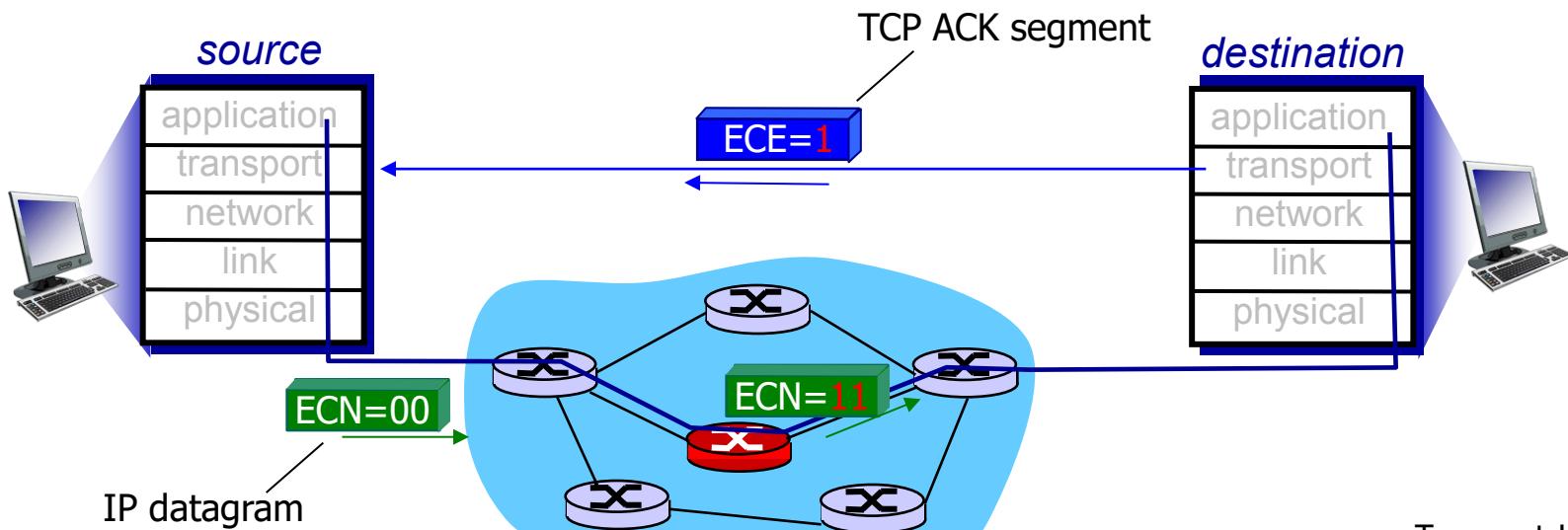
- application can open multiple parallel connections between two hosts
- web browsers do this
- e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate  $R/10$
  - new app asks for 11 TCPs, gets  $R/2$



# Explicit Congestion Notification (ECN)

*network-assisted congestion control:*

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
- congestion indication carried to receiving host
- receiver (seeing congestion indication in IP datagram) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion



# 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”
- two network layer chapters:
  - data plane
  - control plane

# Chapter 4

## Network Layer: The Data Plane

*routing*

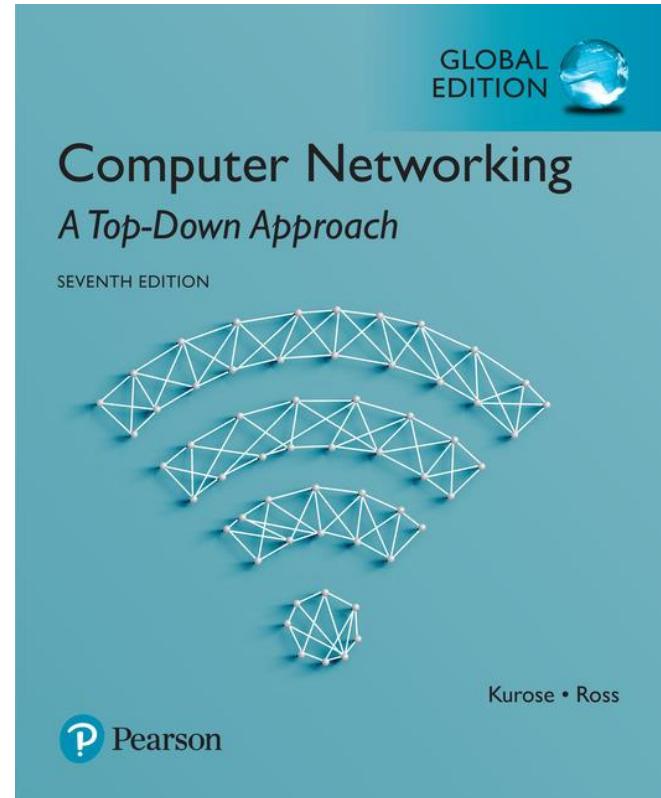
### A note on the use of these Powerpoint slides:

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

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

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016  
J.F Kurose and K.W. Ross, All Rights Reserved



## Computer Networking: A Top Down Approach

7<sup>th</sup> Edition, Global Edition  
Jim Kurose, Keith Ross  
Pearson  
April 2016

# Chapter 4: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

## 4.4 Generalized Forward and SDN

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

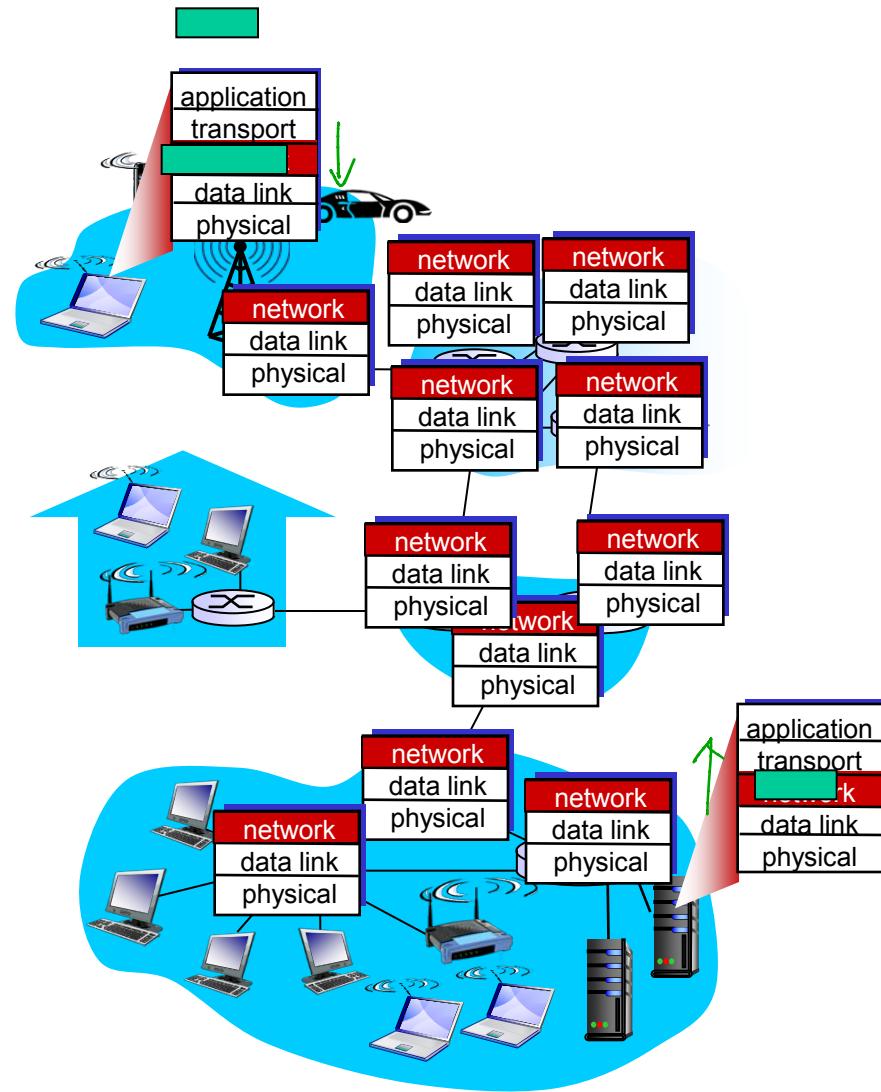
# Chapter 4: network layer

## *chapter goals:*

- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - **forwarding** versus routing data plane control plane
  - how a router works HWD.
  - generalized forwarding → SDN?
- instantiation, implementation in the Internet

# 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

*network-layer functions:*

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

*analogy: taking a trip*

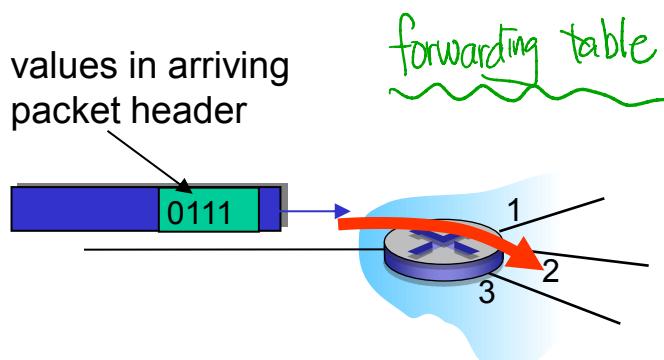
- **forwarding:** process of getting through single interchange
- **routing:** process of planning trip from source to destination

*Big Picture -*

# Network layer: data plane, control plane

## Data plane

- local, per-router function
- determines how datagram arriving on router input port is forwarded to router output port
- forwarding function

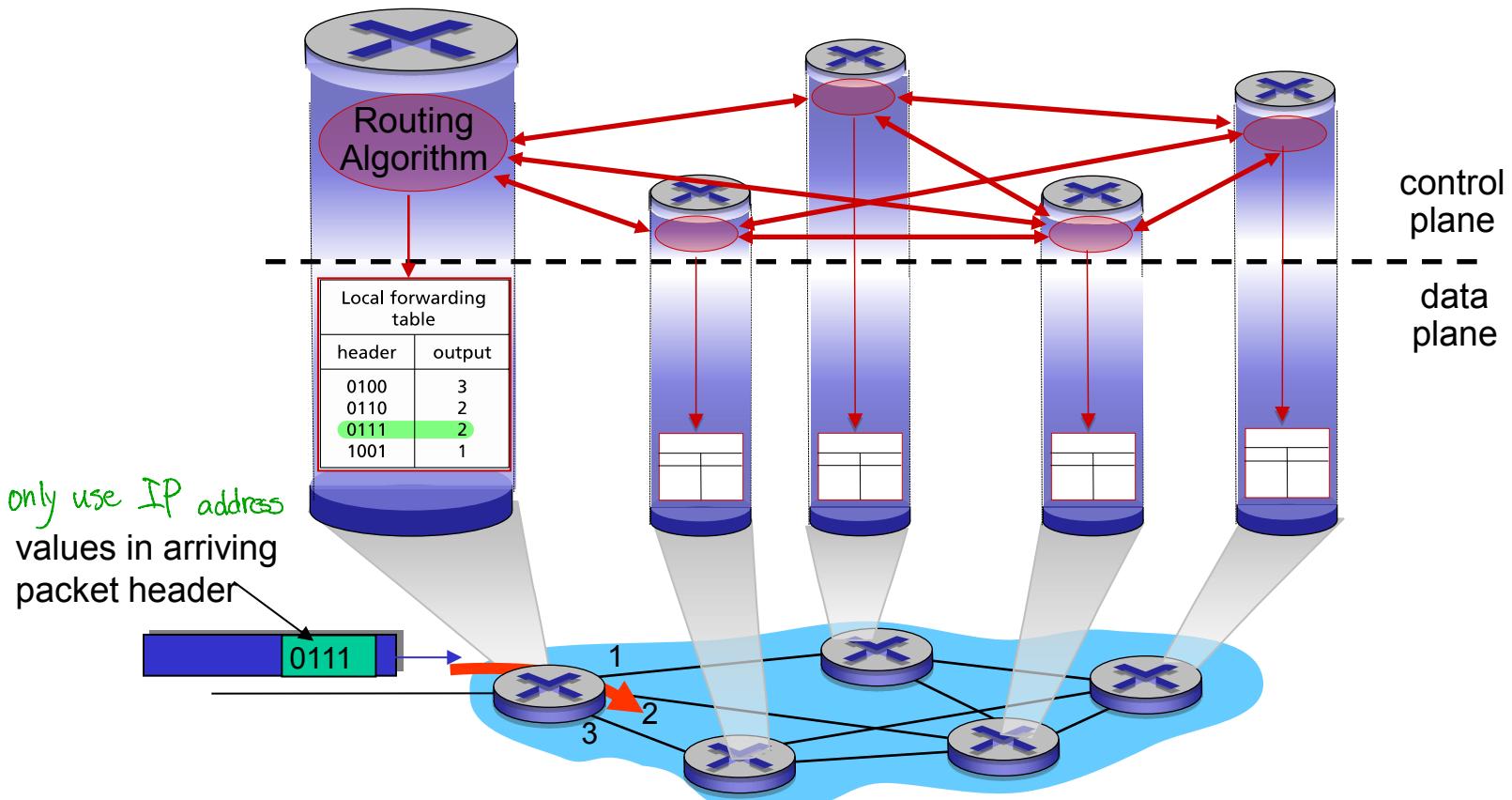


## Control plane

- network-wide logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms*: implemented in routers communicate each other
  - *software-defined networking (SDN)*: implemented in (remote) servers centralized

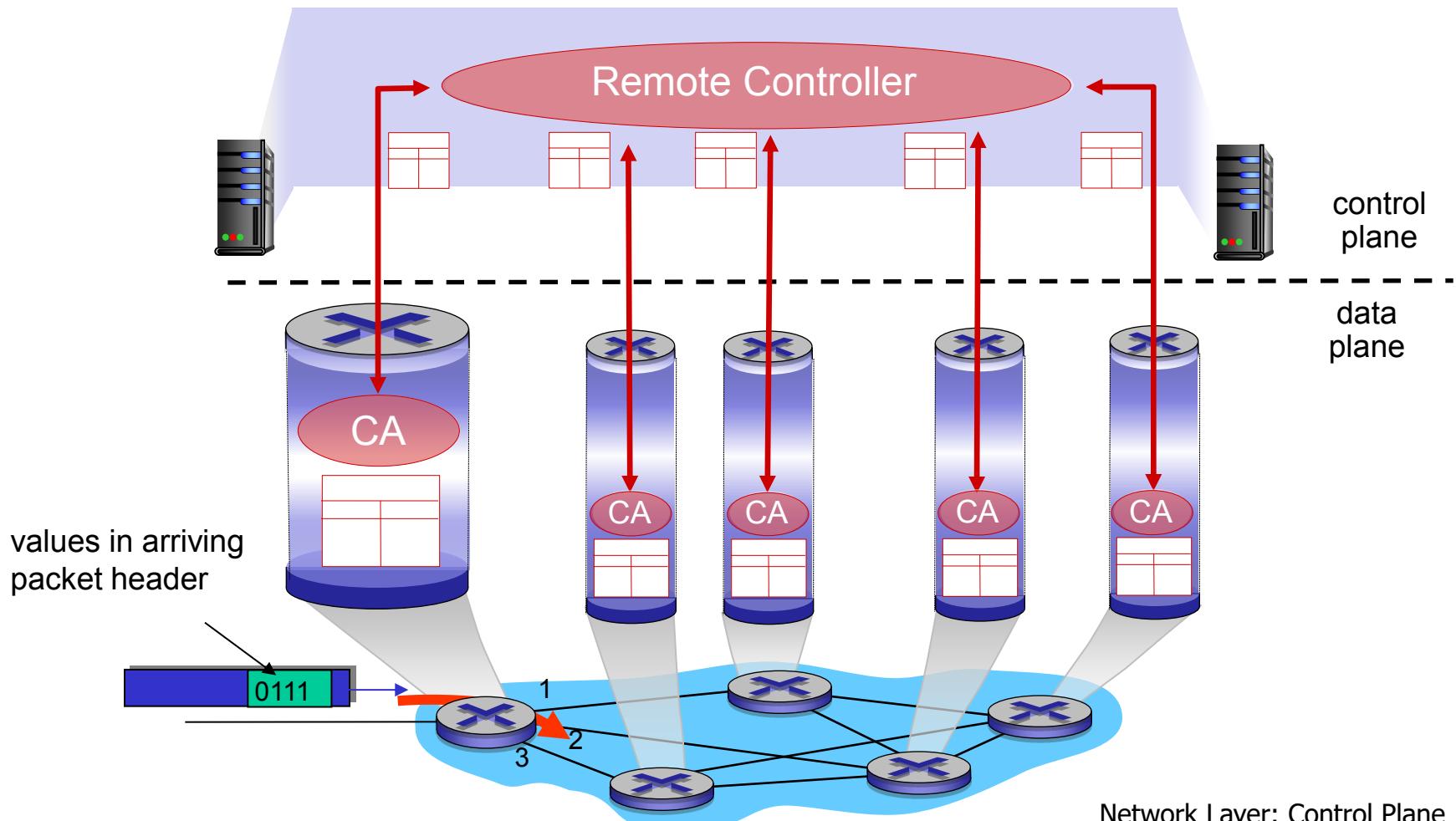
# Per-router control plane

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



# Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs)  $\Rightarrow$  logical



# 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  
*with some specified delay*

*example services for a flow  
of datagrams:*

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing  
*↳ Jitter requirement*

# 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 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

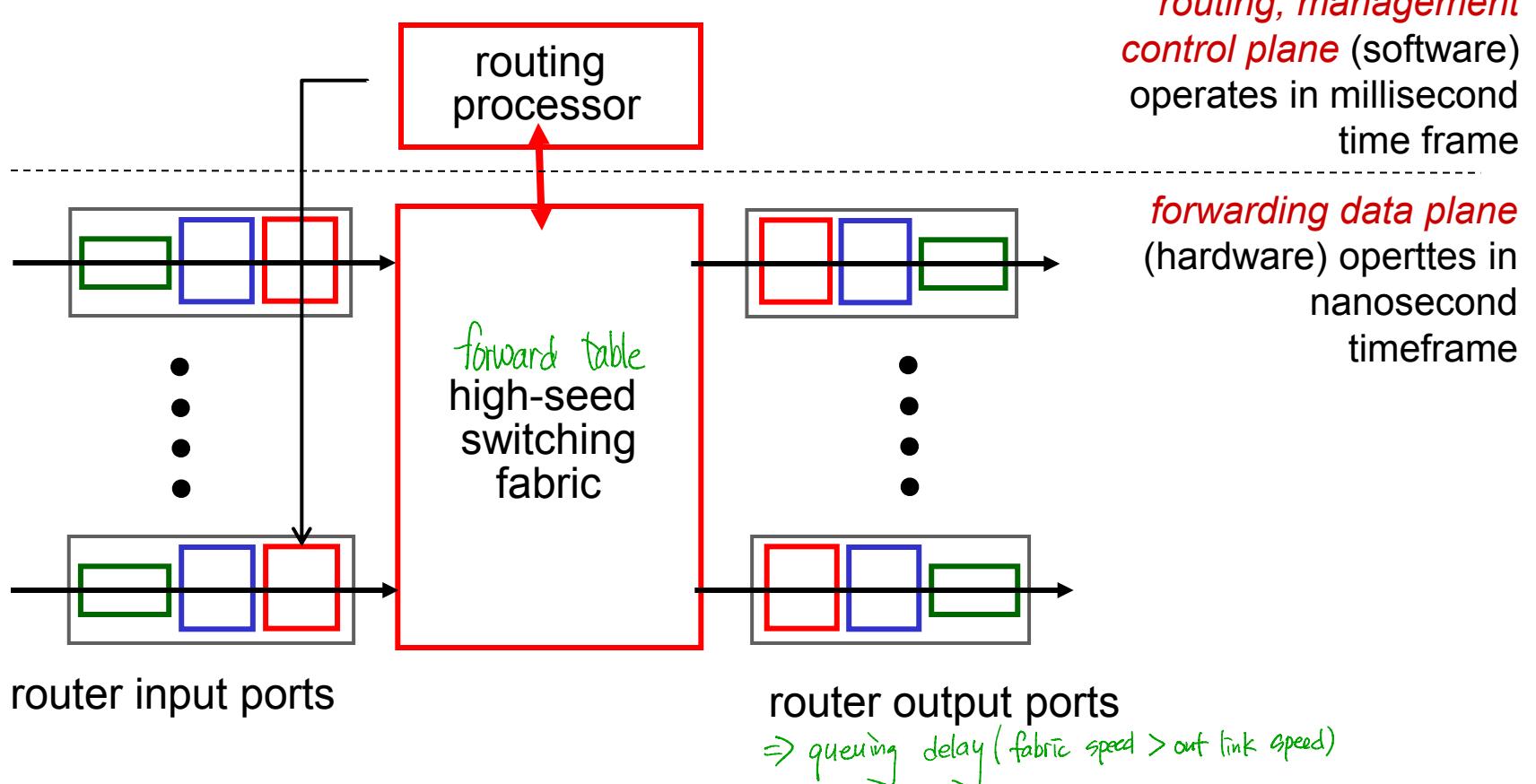
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

## 4.4 Generalized Forward and SDN

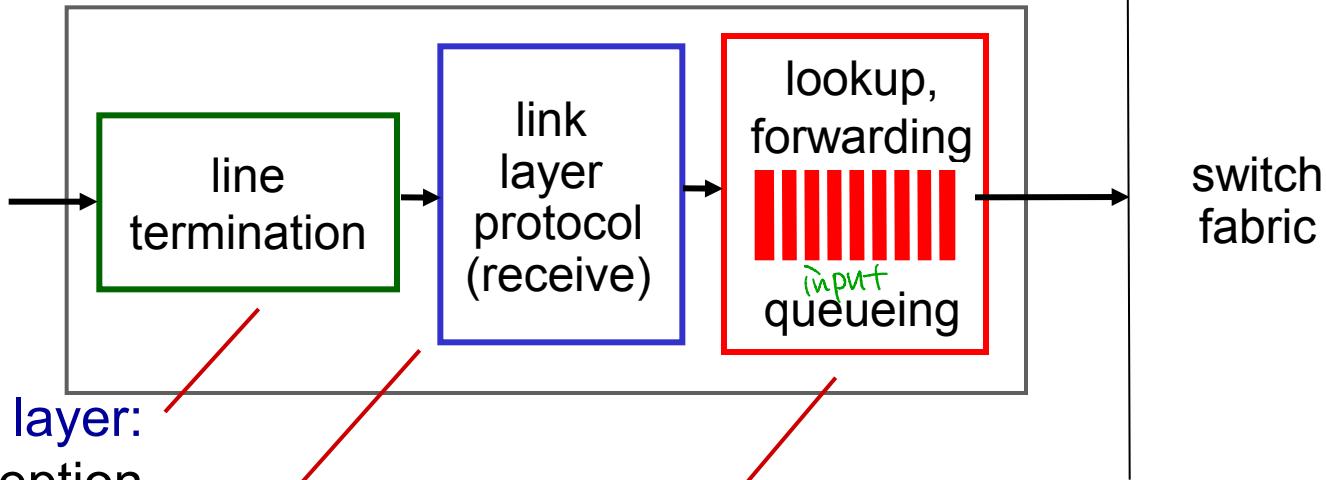
- match
- action
- OpenFlow examples of match-plus-action in action

# Router architecture overview

- high-level view of generic router architecture:



# Input port functions



physical layer:

bit-level reception

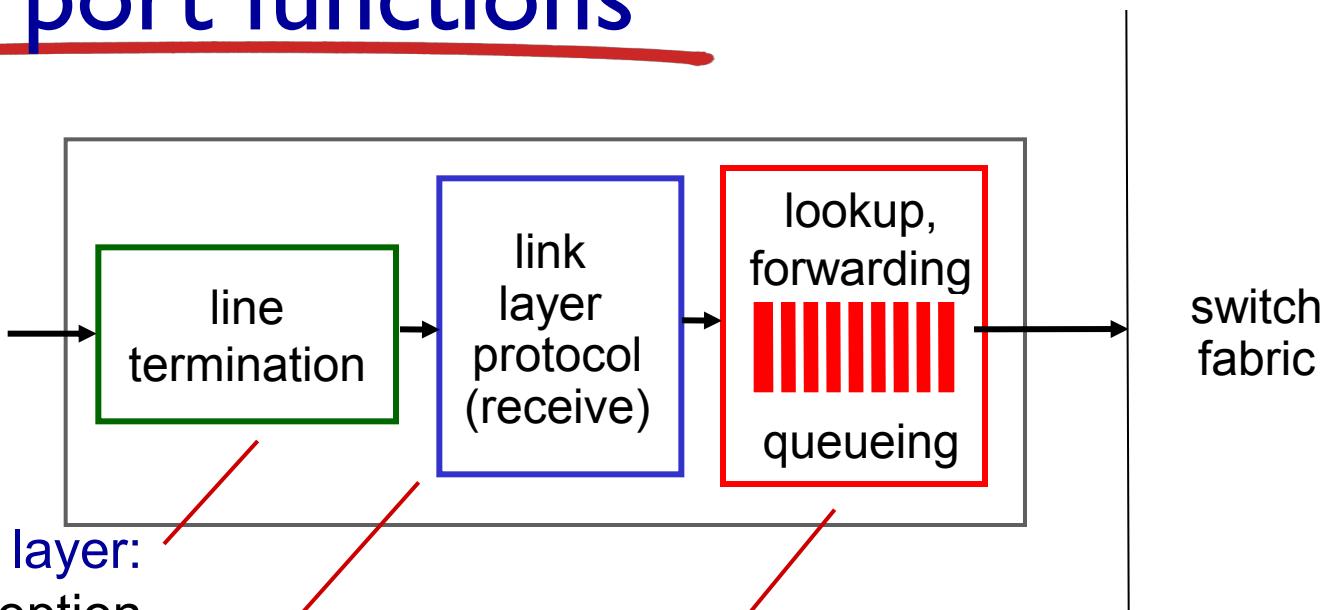
data link layer:

e.g., Ethernet  
see chapter 5

decentralized switching:

- using header field values, 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

# Input port functions



physical layer:

bit-level reception

data link layer:

e.g., Ethernet  
see chapter 5

decide out port

decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory (“*match plus action*”)
- **destination-based forwarding**: forward based only on destination IP address (traditional)
- **generalized forwarding**: forward based on any set of header field values (SDN)

# Destination-based forwarding

*forwarding table*

Destination Address Range	Link Interface
11001000 00010111 00010 <b>0000</b> 00000000 through 11001000 00010111 00010 <b>1111</b> 11111111	0
11001000 00010111 00011000 <b>00000000</b> through 11001000 00010111 00011000 <b>11111111</b>	1
11001000 00010111 00011 <b>001</b> 00000000 through 11001000 00010111 00011 <b>111</b> 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? ①

DA: 11001000 00010111 00011000 10101010

which interface? ②

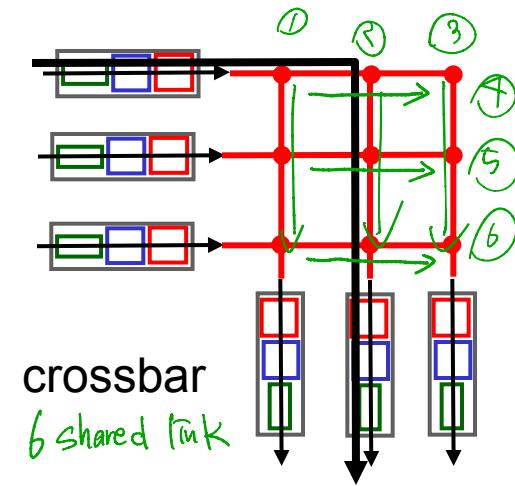
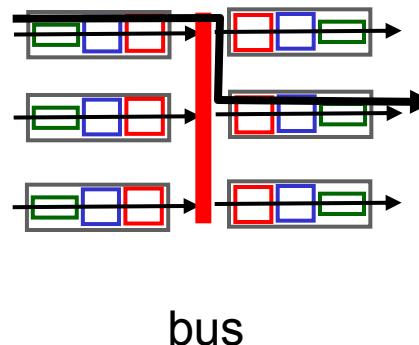
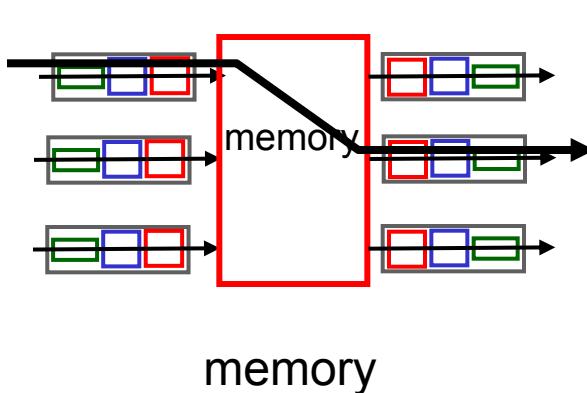
# Longest prefix matching



- we'll see *why* longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  - *content addressable*: present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: can up ~1M routing table entries in TCAM

# 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

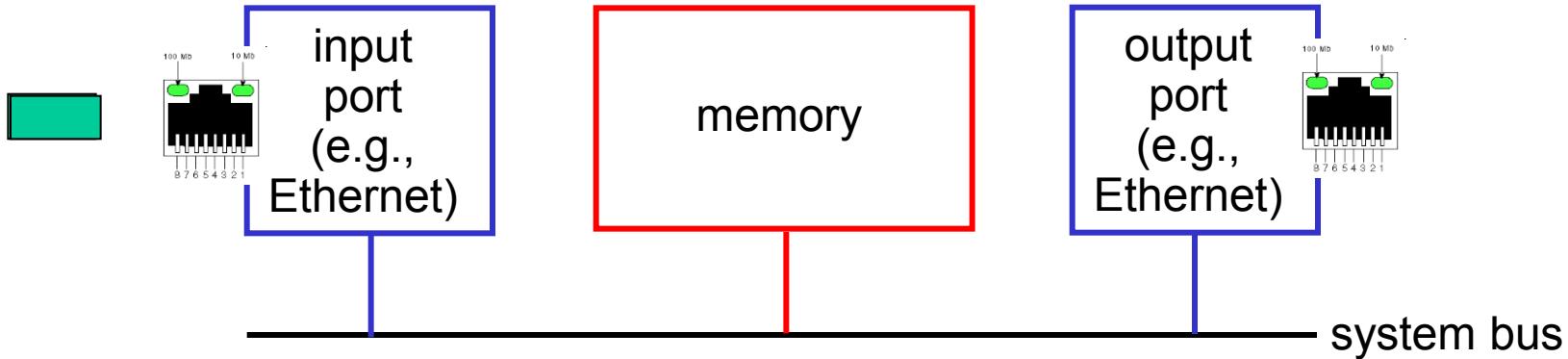


# Switching via memory



*first generation routers:*

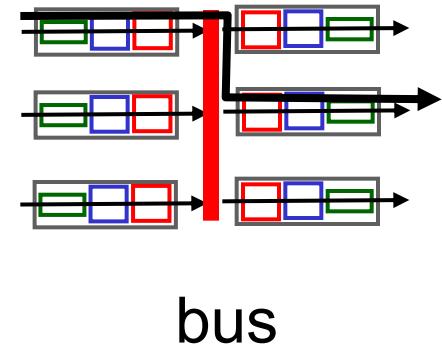
- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



# Switching via a bus

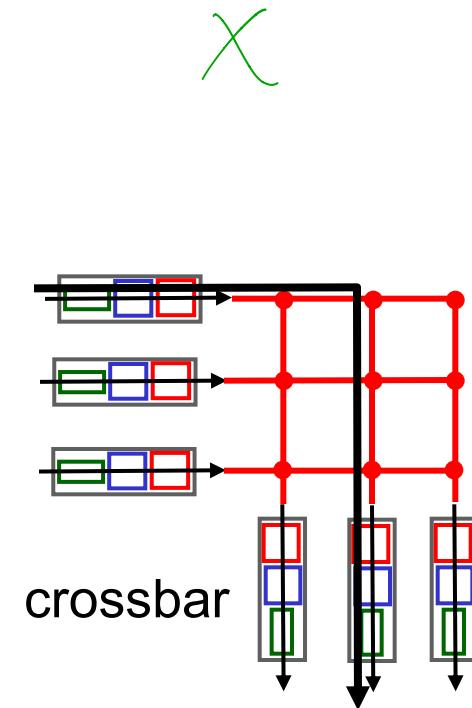


- datagram from input port memory to output port memory via a shared bus
- *bus contention*: switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers



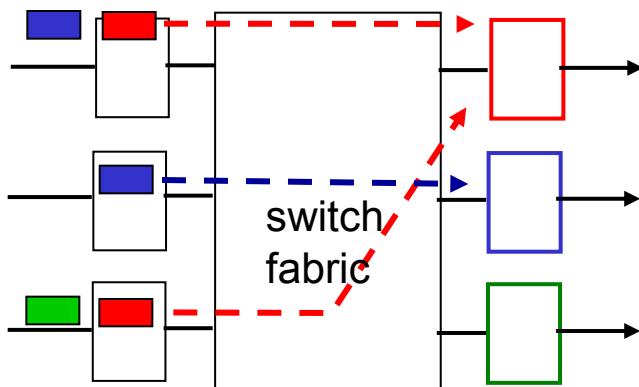
# Switching via interconnection network

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

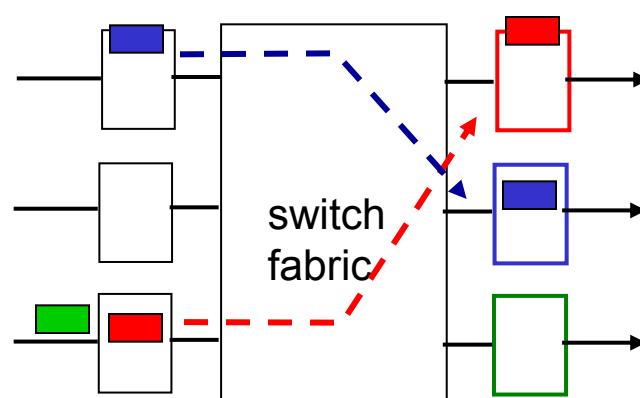


# Input port queuing

- 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*

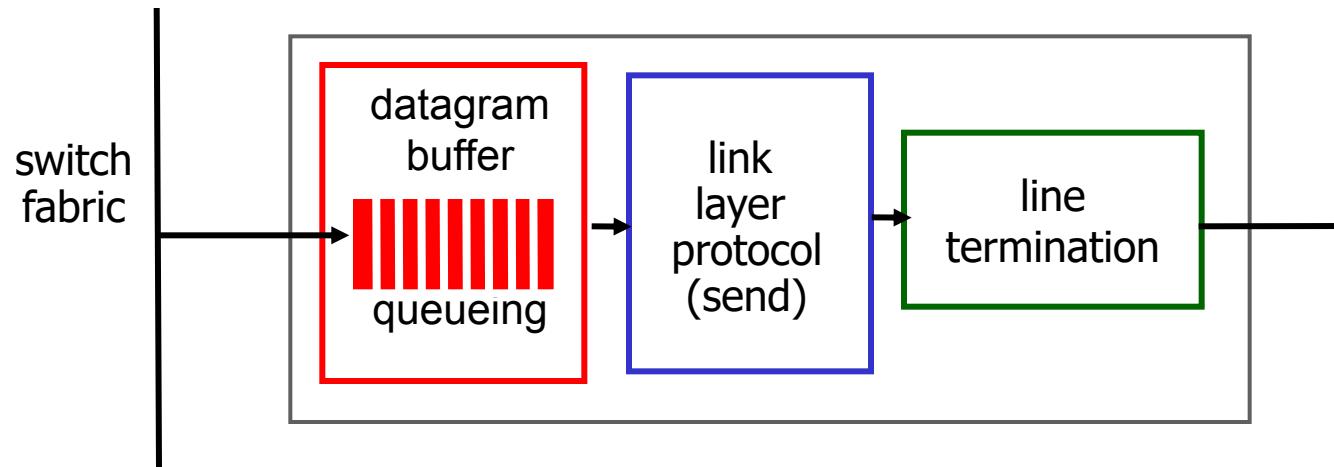


one packet time later:  
green packet experiences HOL blocking



# Output ports

*This slide is HUGELY important!*

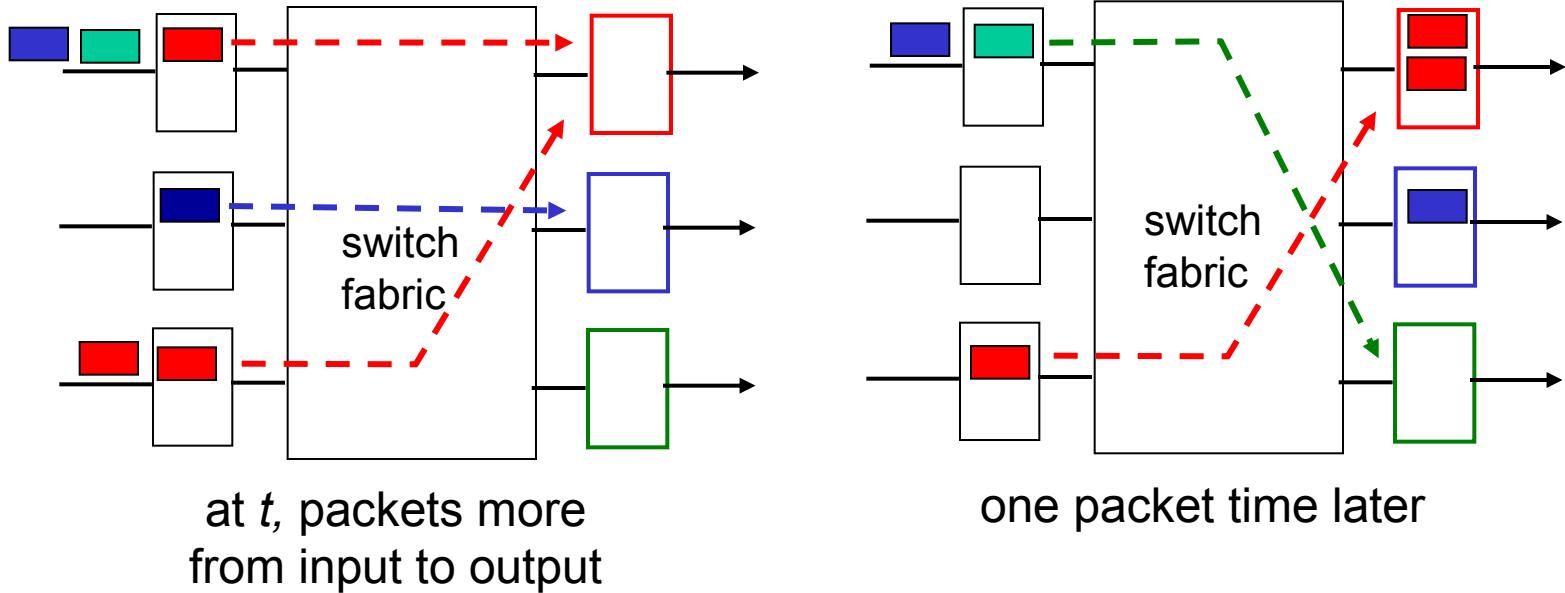


- **buffering** required from fabric faster rate
- **scheduling** datagrams

Datagram (packets) can be lost due to congestion, lack of buffers

Priority scheduling – who gets best performance, network neutrality

# Output port queueing



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

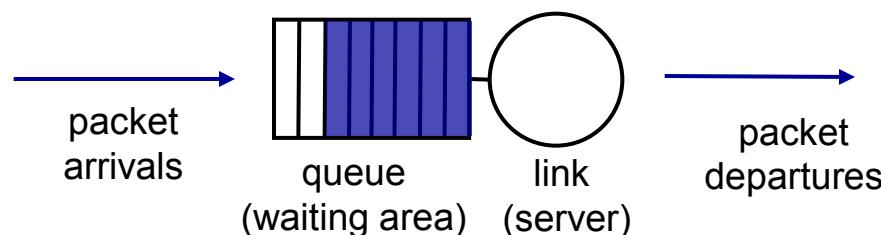
# How much buffering?

- 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

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

# Scheduling mechanisms

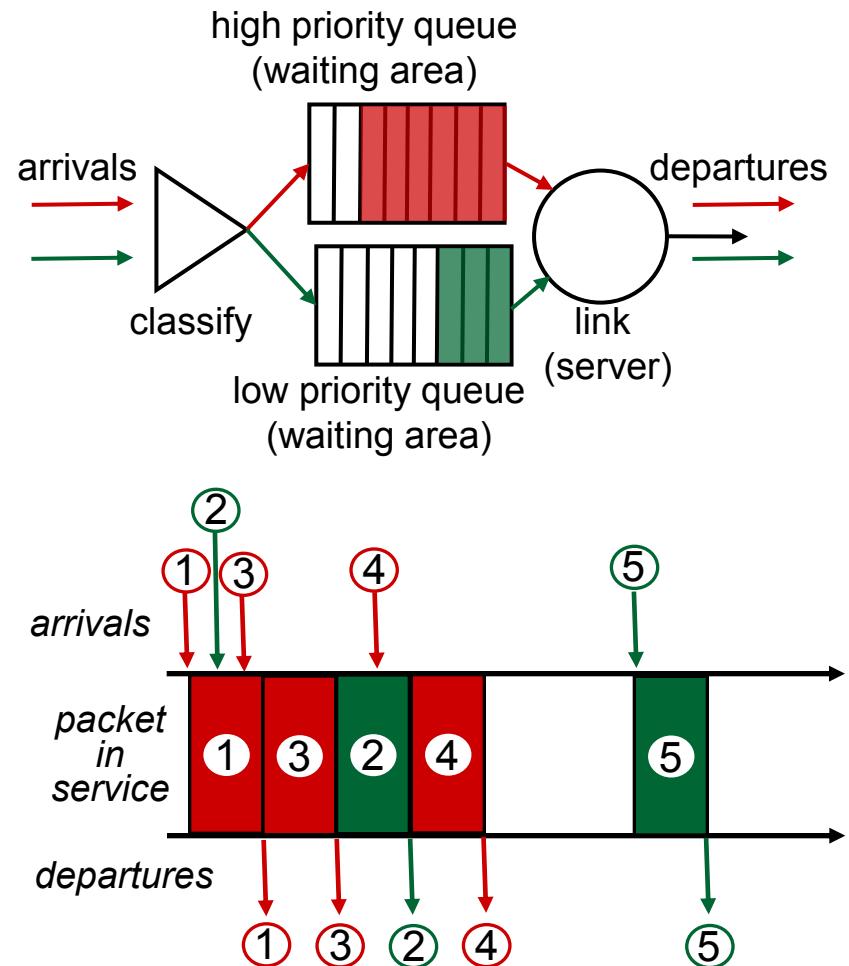
- *scheduling*: choose next packet to send on link
- *FIFO (first in first out) scheduling*: send in order of arrival to queue
  - real-world example?
  - *discard policy*: if packet arrives to full queue: who to discard?
    - *tail drop*: drop arriving packet
    - *priority*: drop/remove on priority basis
    - *random*: drop/remove randomly



# Scheduling policies: priority

*priority scheduling:* send  
highest priority  
queued packet

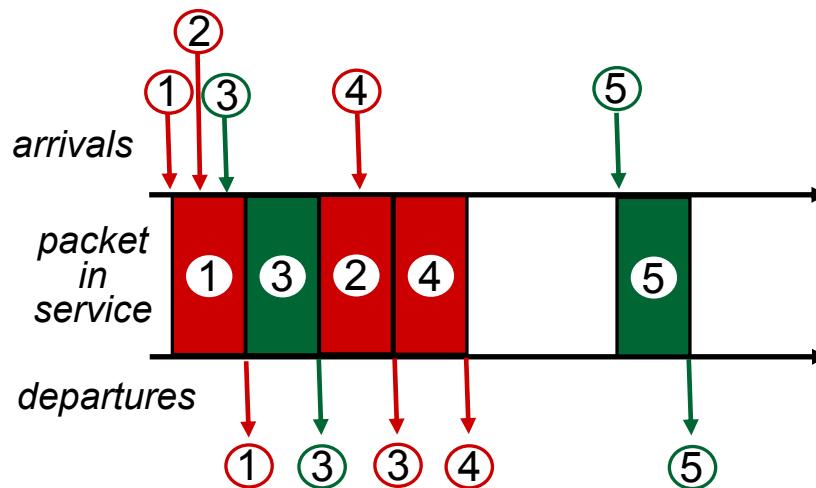
- multiple classes, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?



# Scheduling policies: still more

*Round Robin (RR) scheduling: cycle*

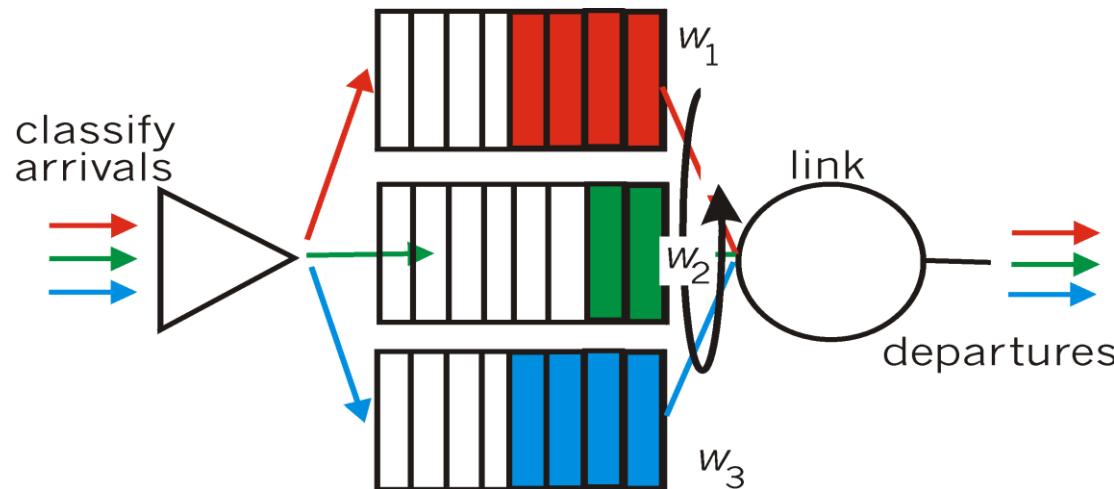
- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
- real world example?



# Scheduling policies: still more

★ **Weighted Fair Queuing (WFQ): RR + Priority**

- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?



# Chapter 4: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

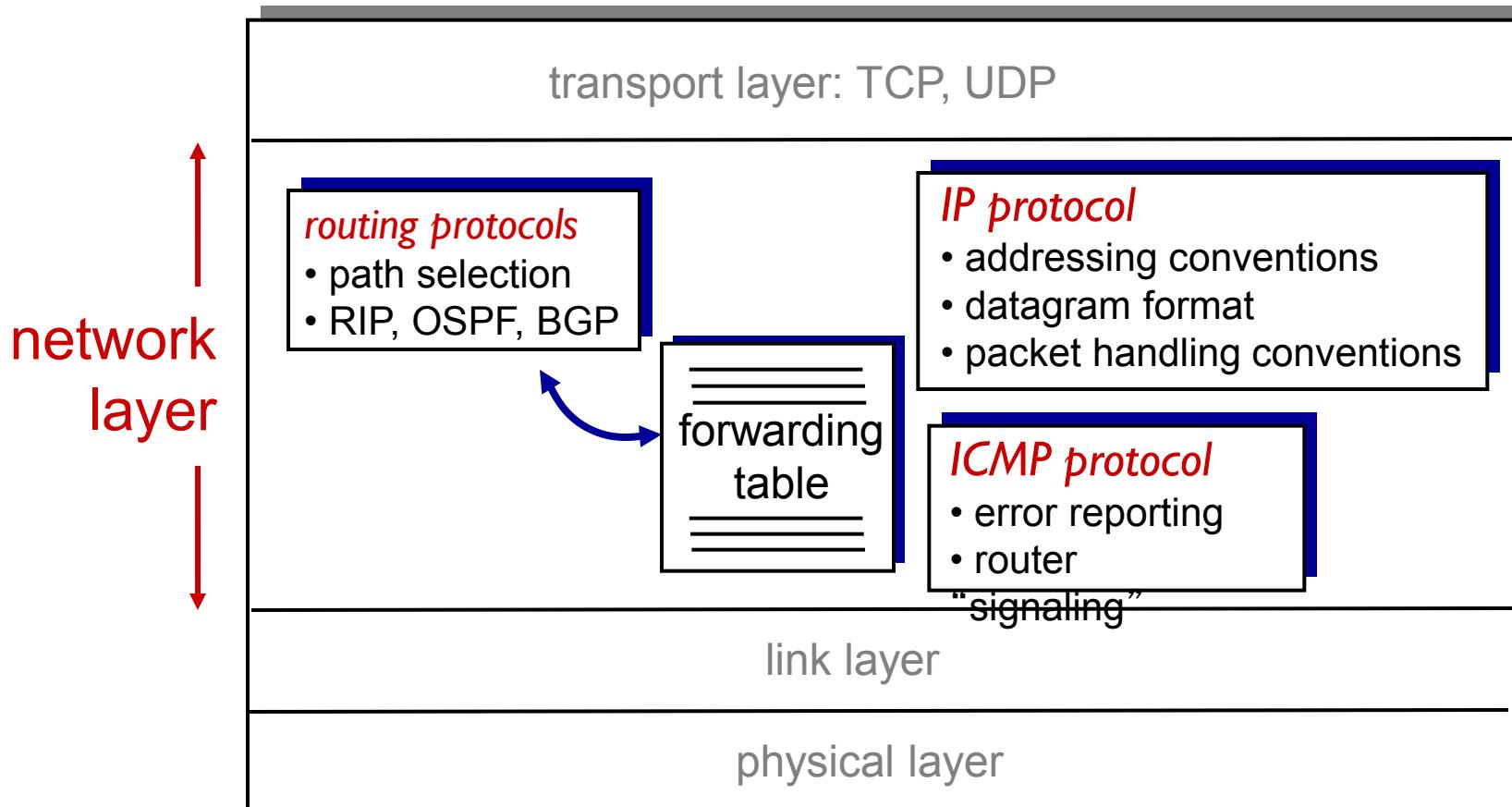
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

## 4.4 Generalized Forward and SDN

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

# The Internet network layer

host, router network layer functions:



# IP datagram format

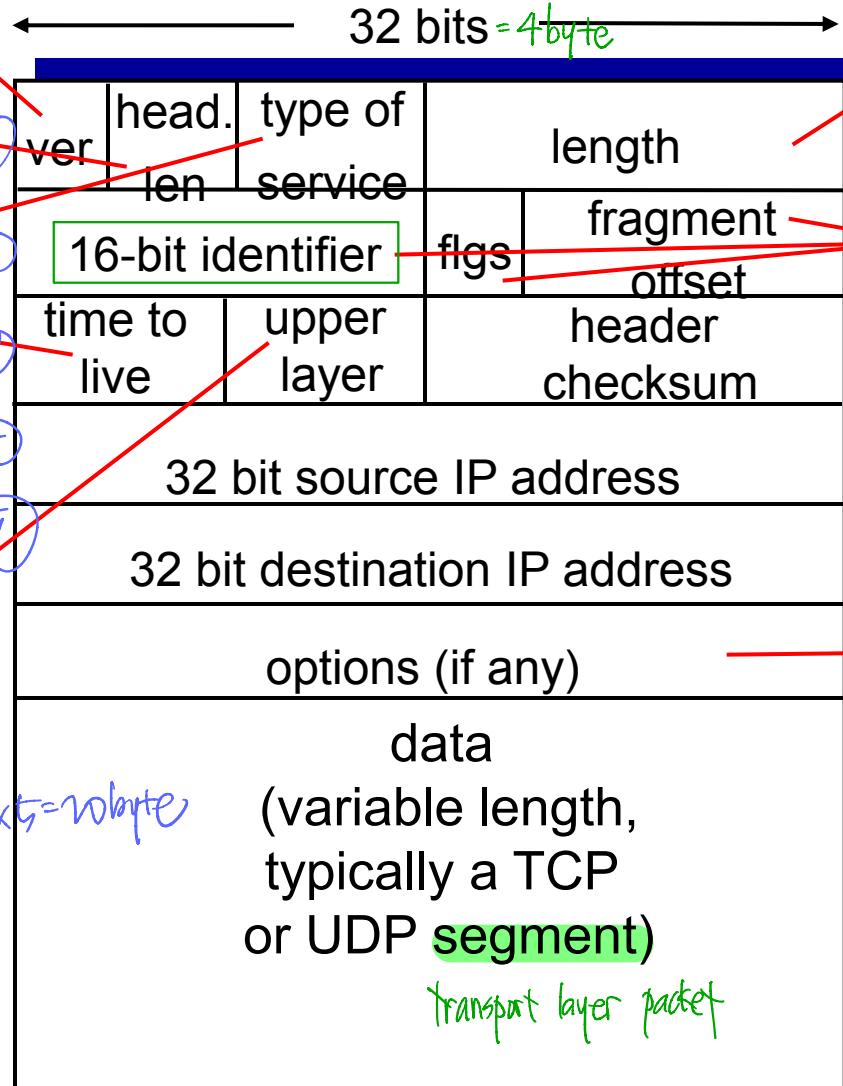
IP protocol version  
number

header length  
(bytes)

“type” of data

handling  
looping  
max number  
remaining hops  
(decremented at  
each router)

TCP/UDP  
upper layer protocol  
to deliver payload to



IPv4 : 32bit  $\Rightarrow 2^{32}$  addresses ... limited

IPv6 : 128bit  $\Rightarrow 2^{128}$  addresses  $\rightarrow \infty$  [high cost for transition!]

total datagram  
length (bytes)

for  
fragmentation/  
reassembly

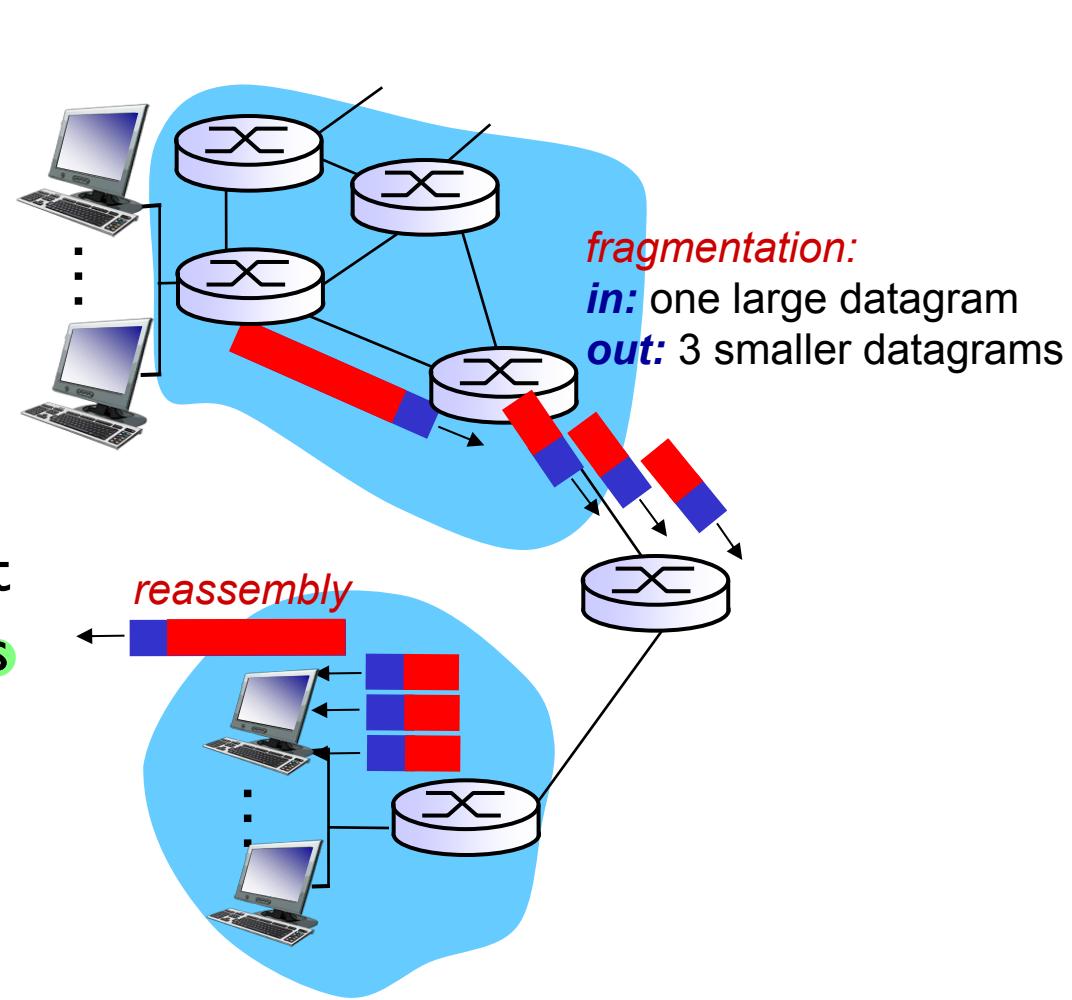
datagram size differs  
depending on protocol  
(wifi, ethernet, satellite ...)

*how much overhead?*

- ❖ 20 bytes of TCP = 4 bytes = 20 bytes
- ❖ 20 bytes of IP
- ❖ = 40 bytes + app layer overhead

# 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

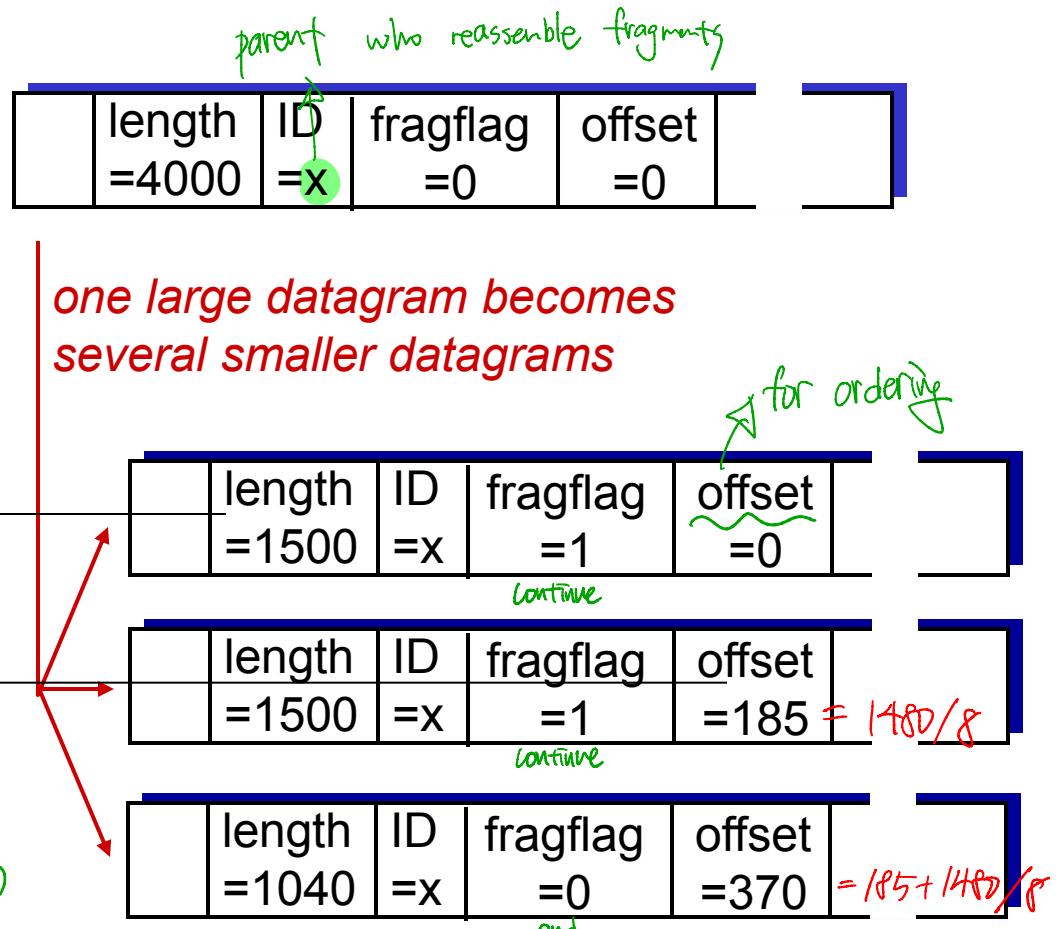
example:  $3980 + 20$  (data) (header)

- ❖ 4000 byte datagram
- ❖ MTU = 1500 bytes

$1480 + 20$  (data) (header)  
1480 bytes in data field

$$\text{offset} = 1480/8$$

$1480 + 20$   
 $1020 + 20$   
 $\Sigma = 3180$



# Chapter 4: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

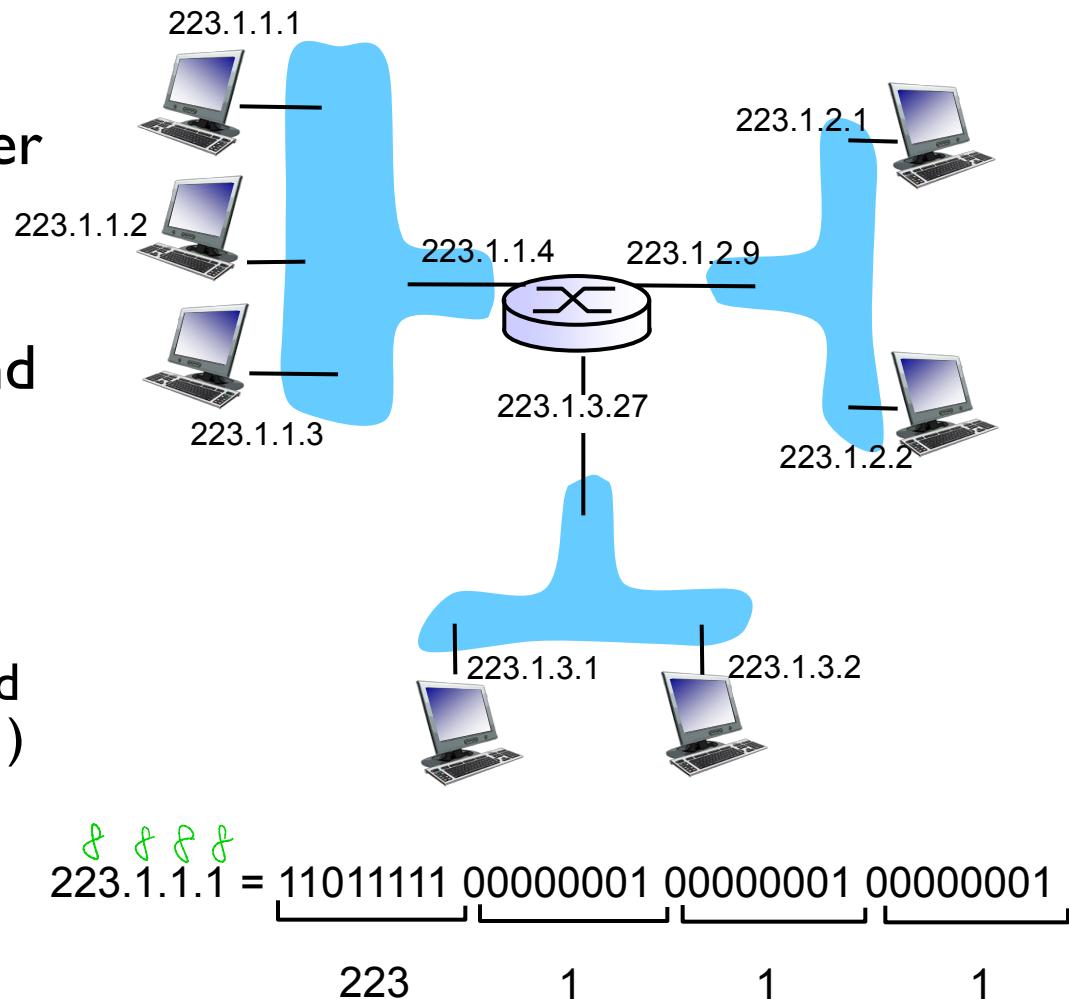
## 4.4 Generalized Forward and SDN

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

# IP addressing: introduction

| IP address per interface

- **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**



# IP addressing: introduction

*Q: how are interfaces actually connected?*

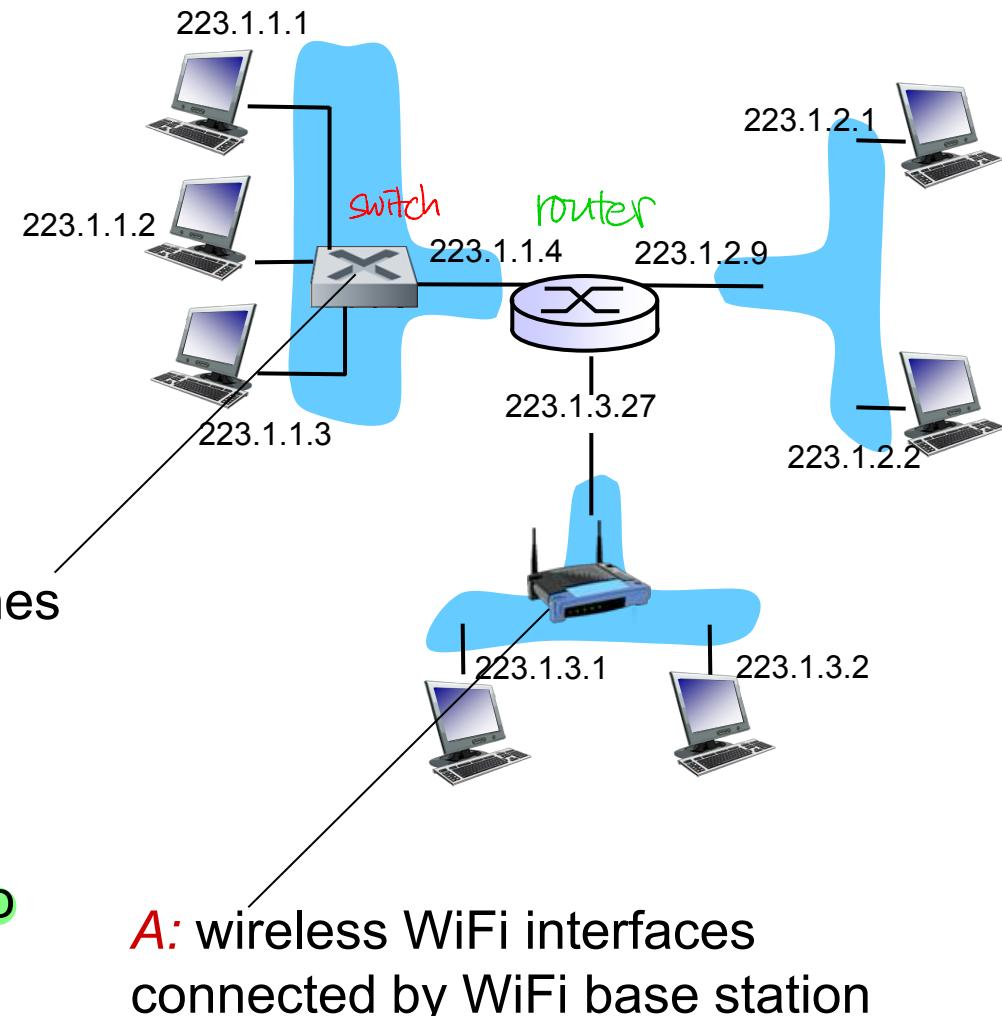
*A: we'll learn about that in chapter 5, 6.*

*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)

router → network layer

switch → link layer



*A: wireless WiFi interfaces connected by WiFi base station*

# Subnets

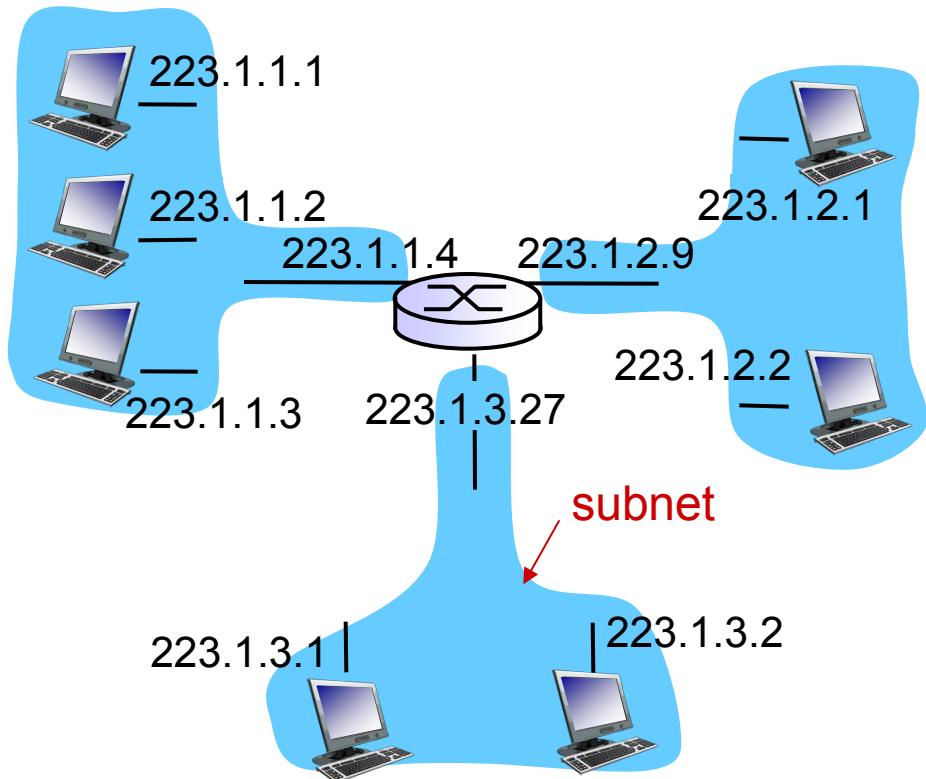
## ■ IP address:

- **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**

using MAC address

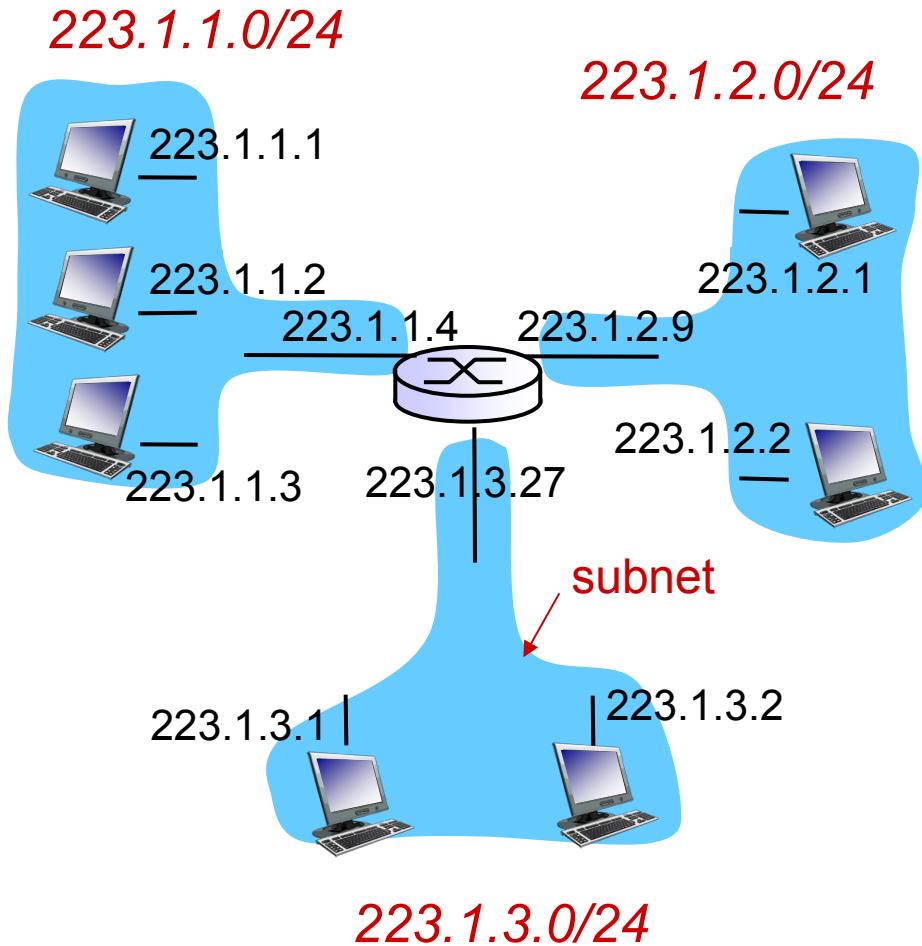


network consisting of 3 subnets

# 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**

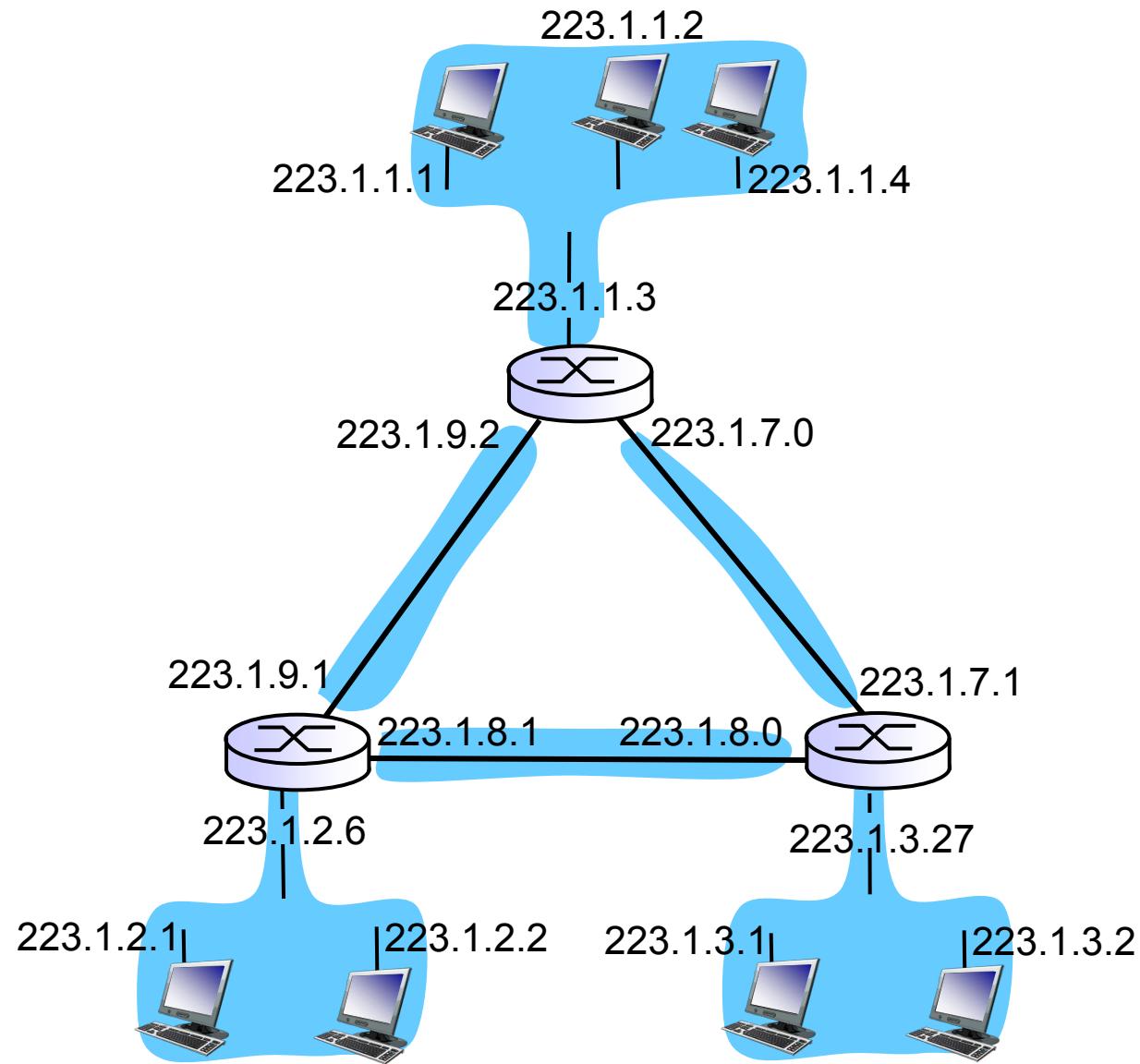


subnet mask: /24

# Subnets

how many?

6



# IP addressing: CIDR

## CIDR: Classless InterDomain Routing

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





# IP addresses: how to get one?

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

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

# DHCP: Dynamic Host Configuration Protocol

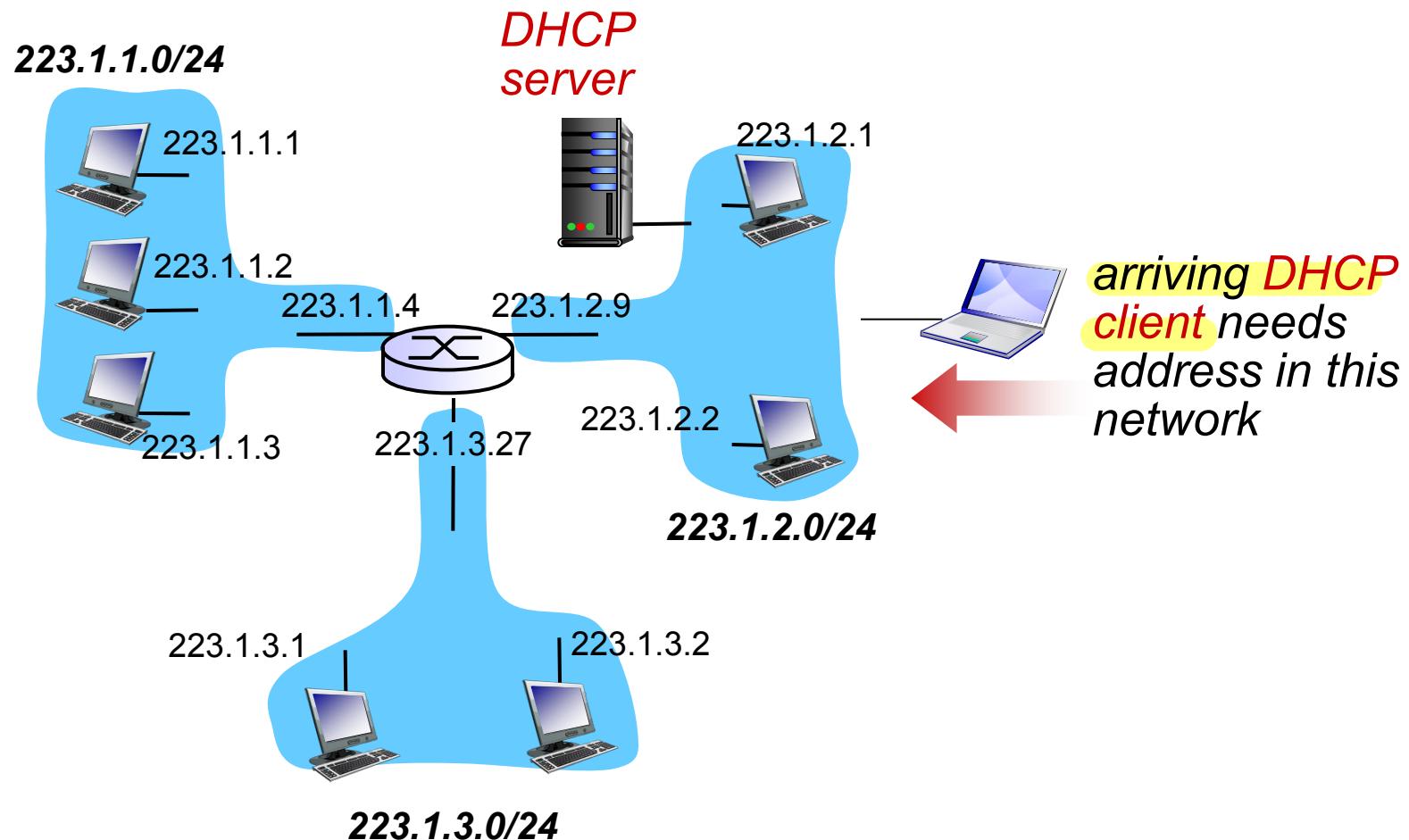
**goal:** allow host to dynamically obtain its IP address from network server when it joins network

- 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)

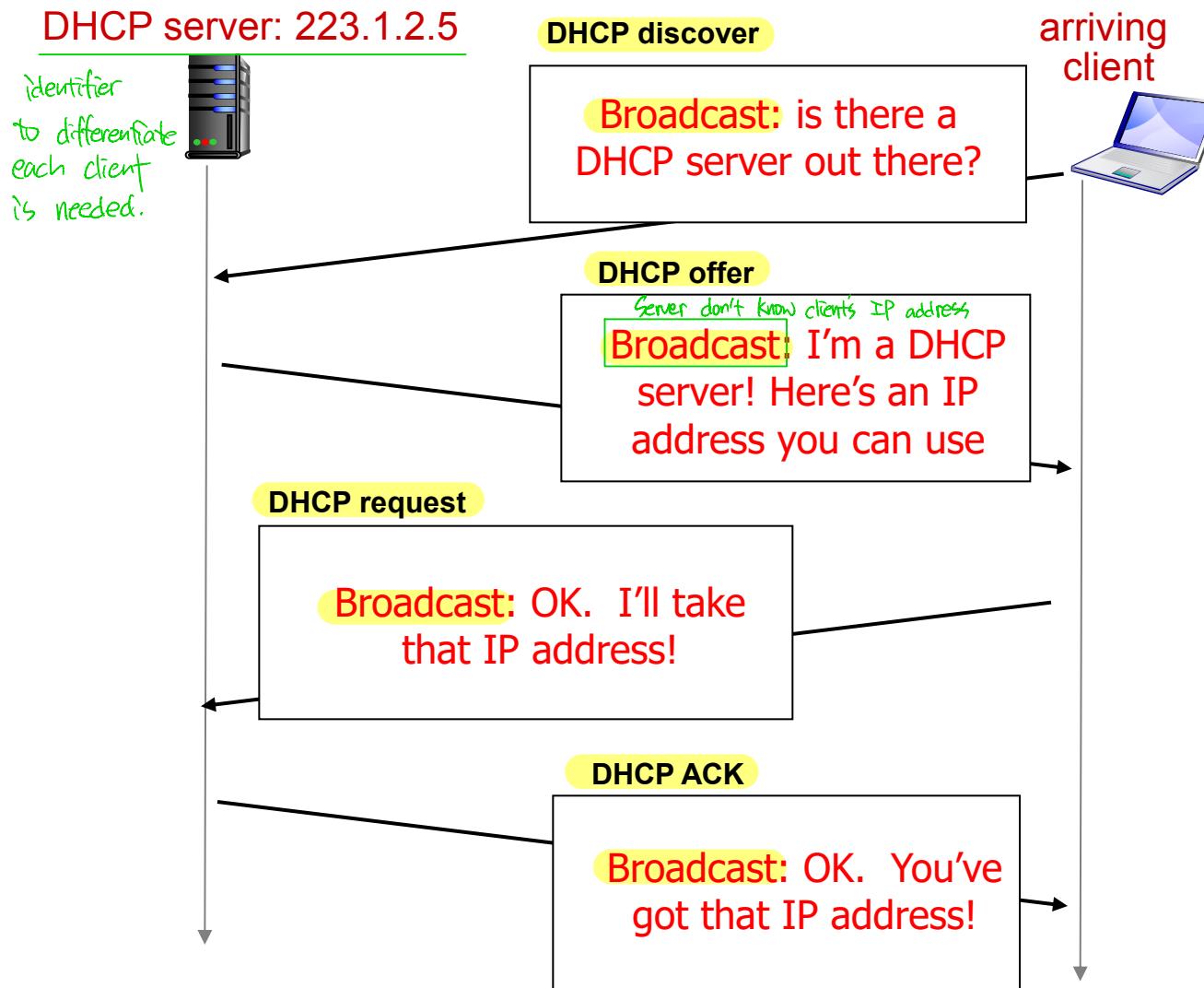
**DHCP overview:**

- host broadcasts “**DHCP discover**” msg [optional] *request*
- 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

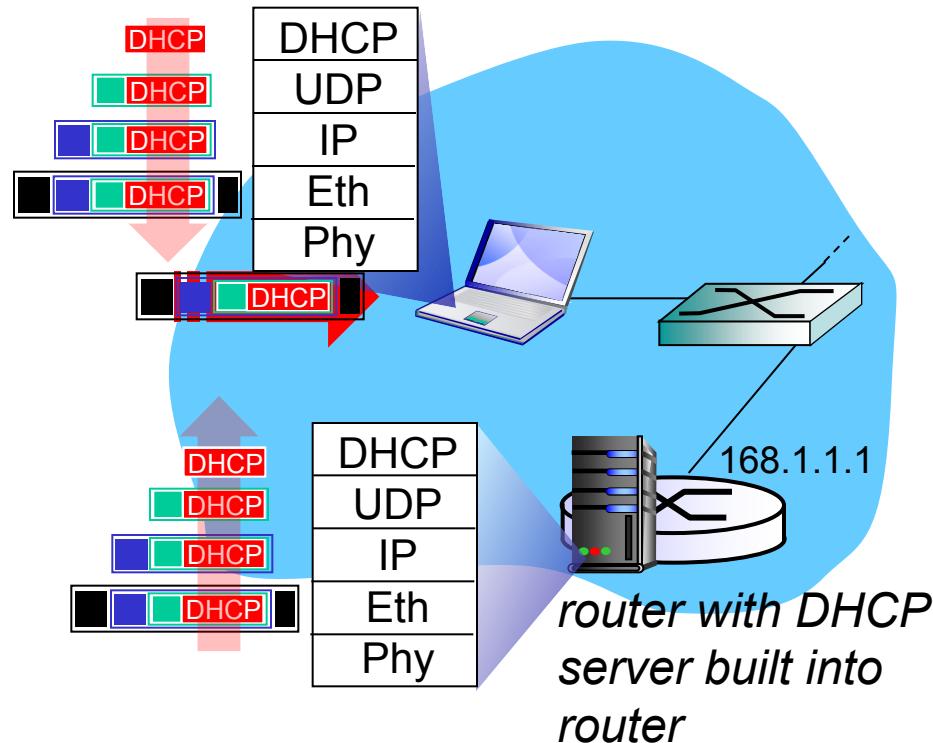


# DHCP: more than IP addresses

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

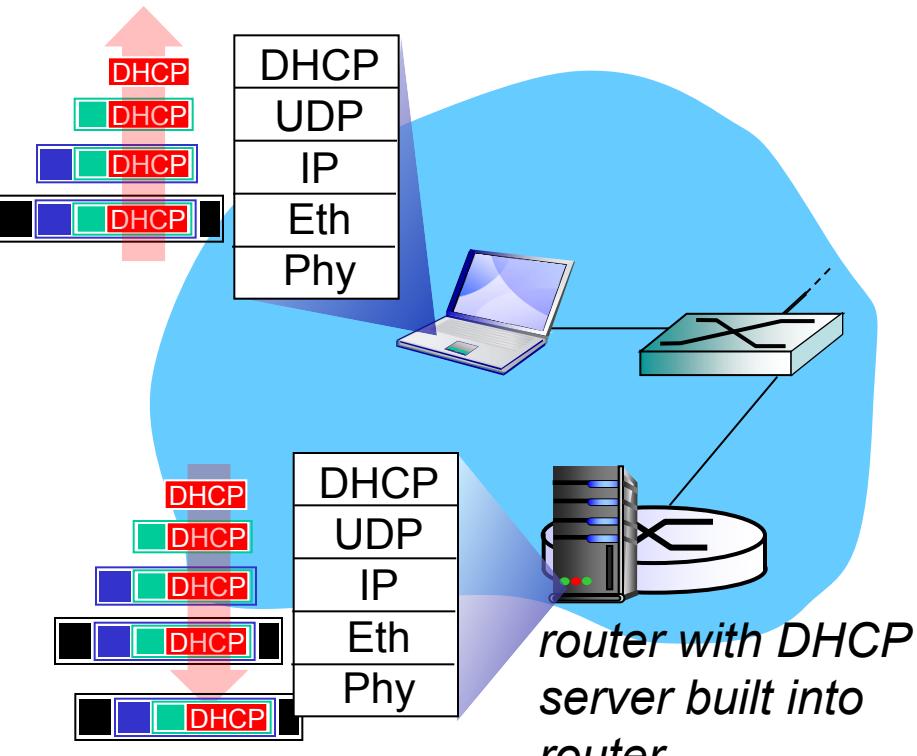
- 
- address of first-hop router for client
  - name and IP address of DNS sever
  - network mask (indicating network versus host portion of address)

# DHCP: example



- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP *⇒ to get MAC*
- 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, UDP demuxed to DHCP

# DHCP: example



- DCP 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 DSN 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?

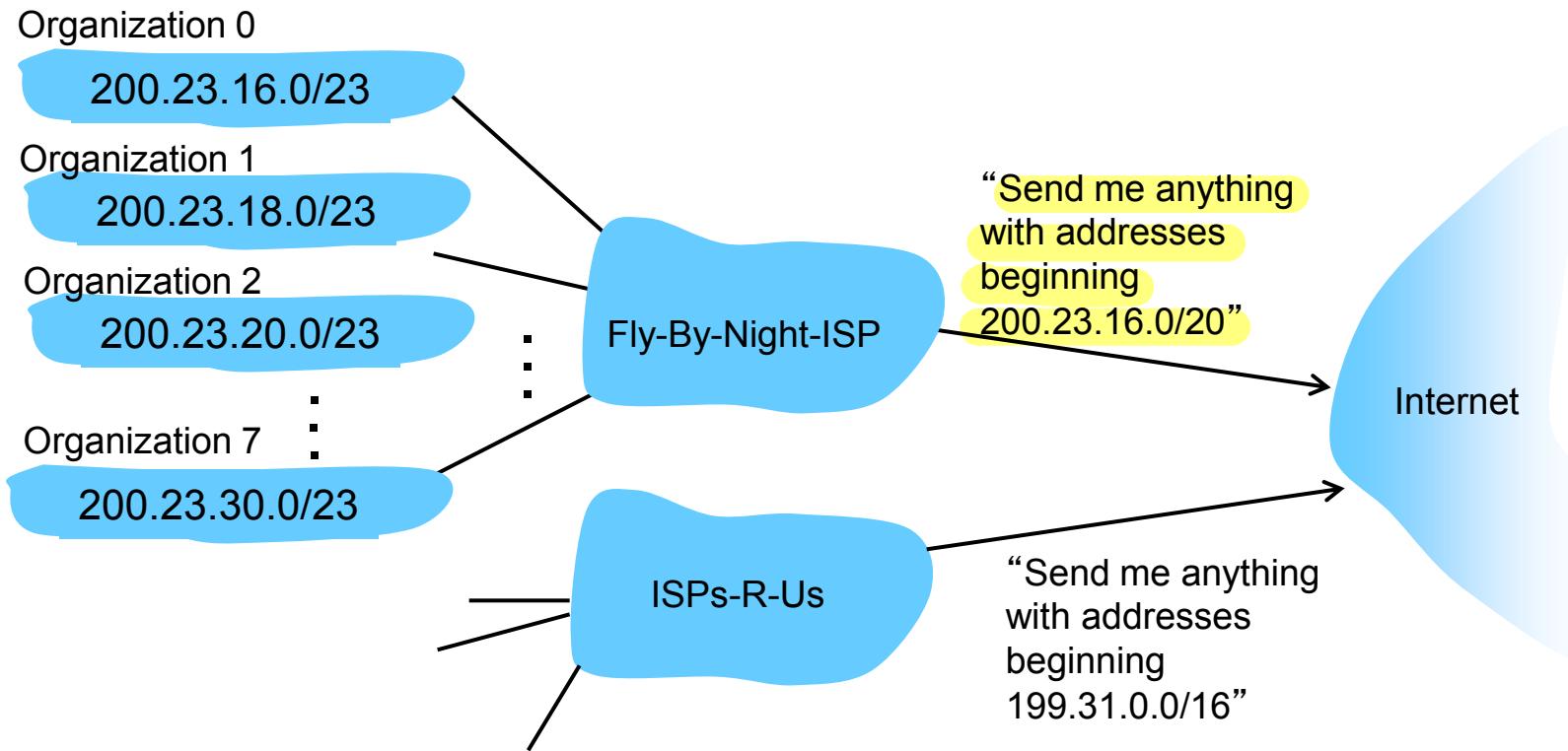
**A:** gets allocated portion of its provider ISP's address space

ISP's block	<u>11001000 00010111 00010000 00000000</u>	200.23.16.0/20 <sup>+3</sup>
Organization 0	<u>11001000 00010111 00010000</u> 00000000	200.23.16.0/23
Organization 1	<u>11001000 00010111 00010010</u> 00000000	200.23.18.0/23
Organization 2	<u>11001000 00010111 00010100</u> 00000000	200.23.20.0/23
...	.....	....
Organization 7	<u>11001000 00010111 00011110</u> 00000000	200.23.30.0/23

<sup>2^3</sup>

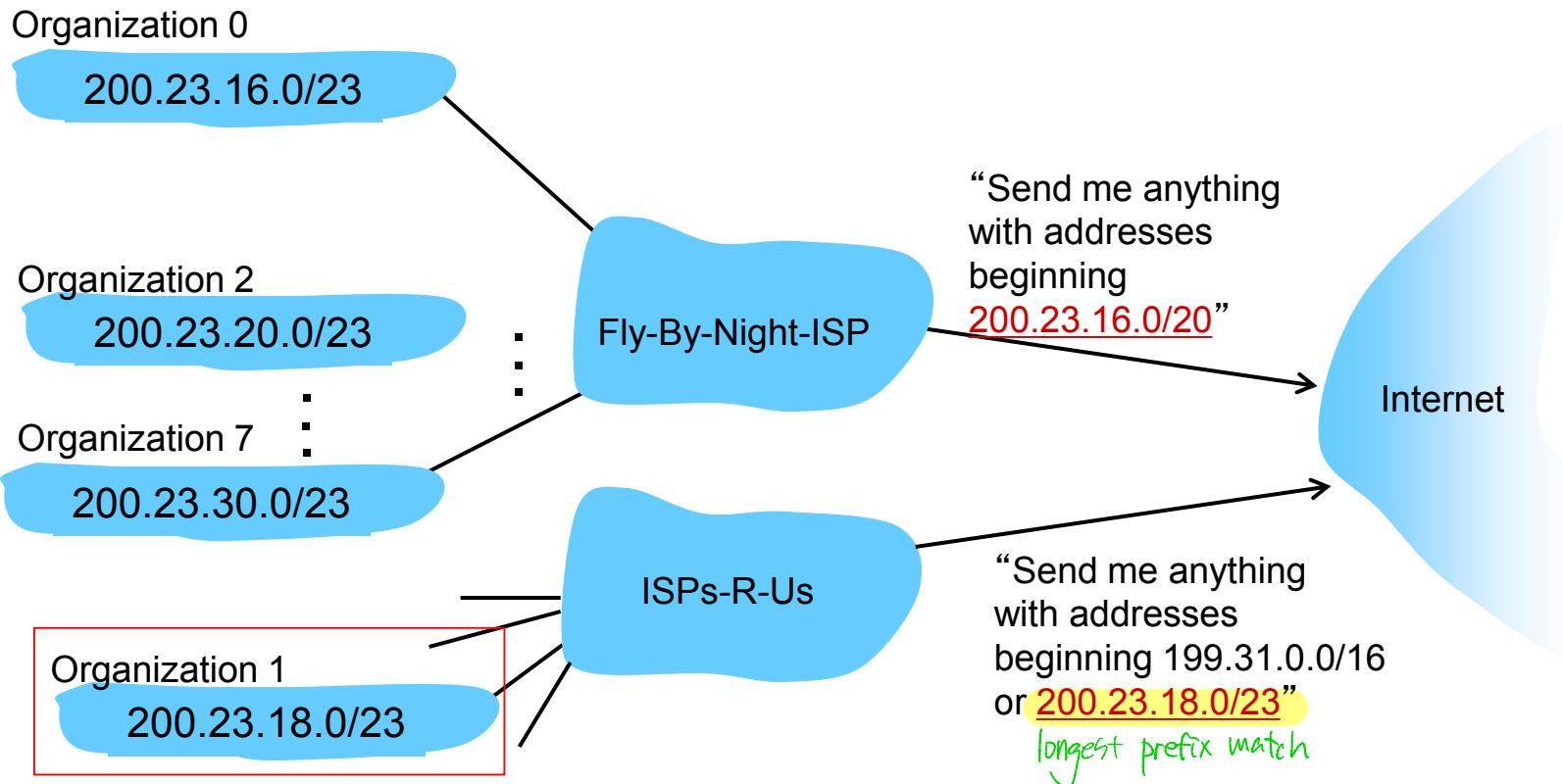
# Hierarchical addressing: route aggregation

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...

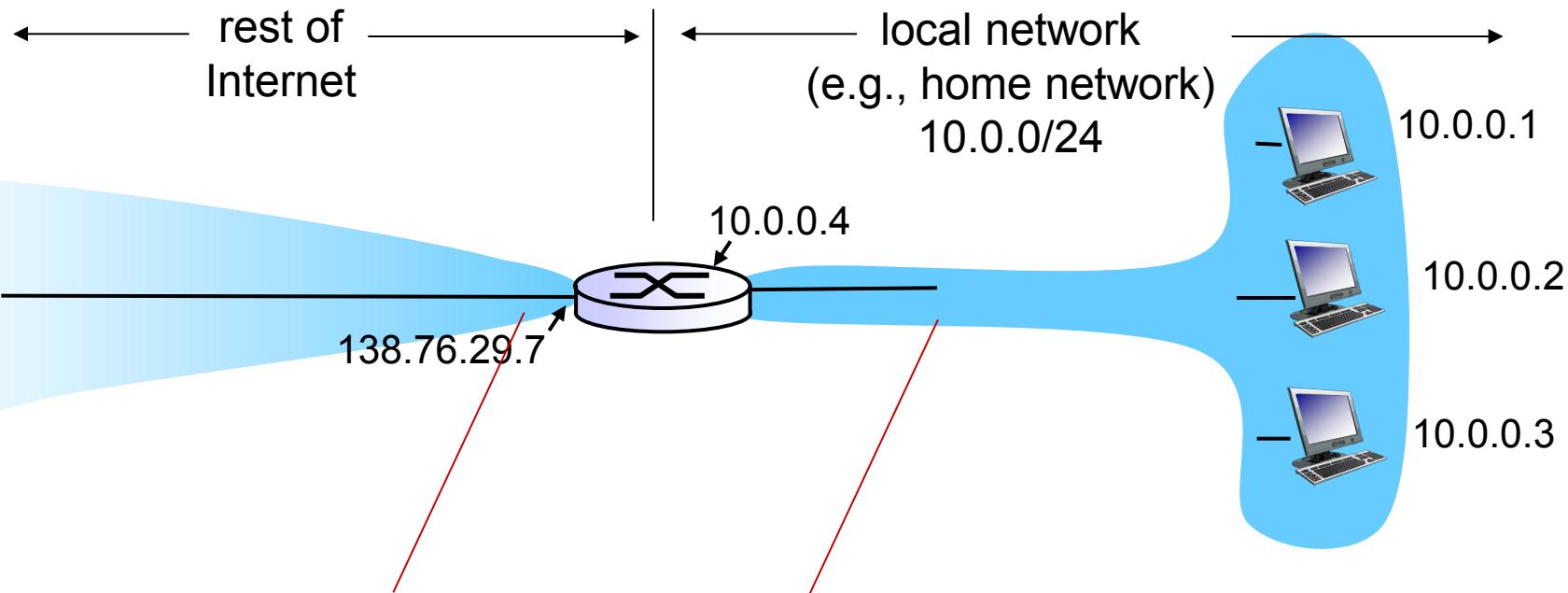
**Q:** how does an ISP get block of addresses?

**A:** ICANN: Internet Corporation for Assigned Names and Numbers <http://www.icann.org/>

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

middle box  $\Rightarrow$  handle all info (port, IP, MAC) at the same time

# 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)

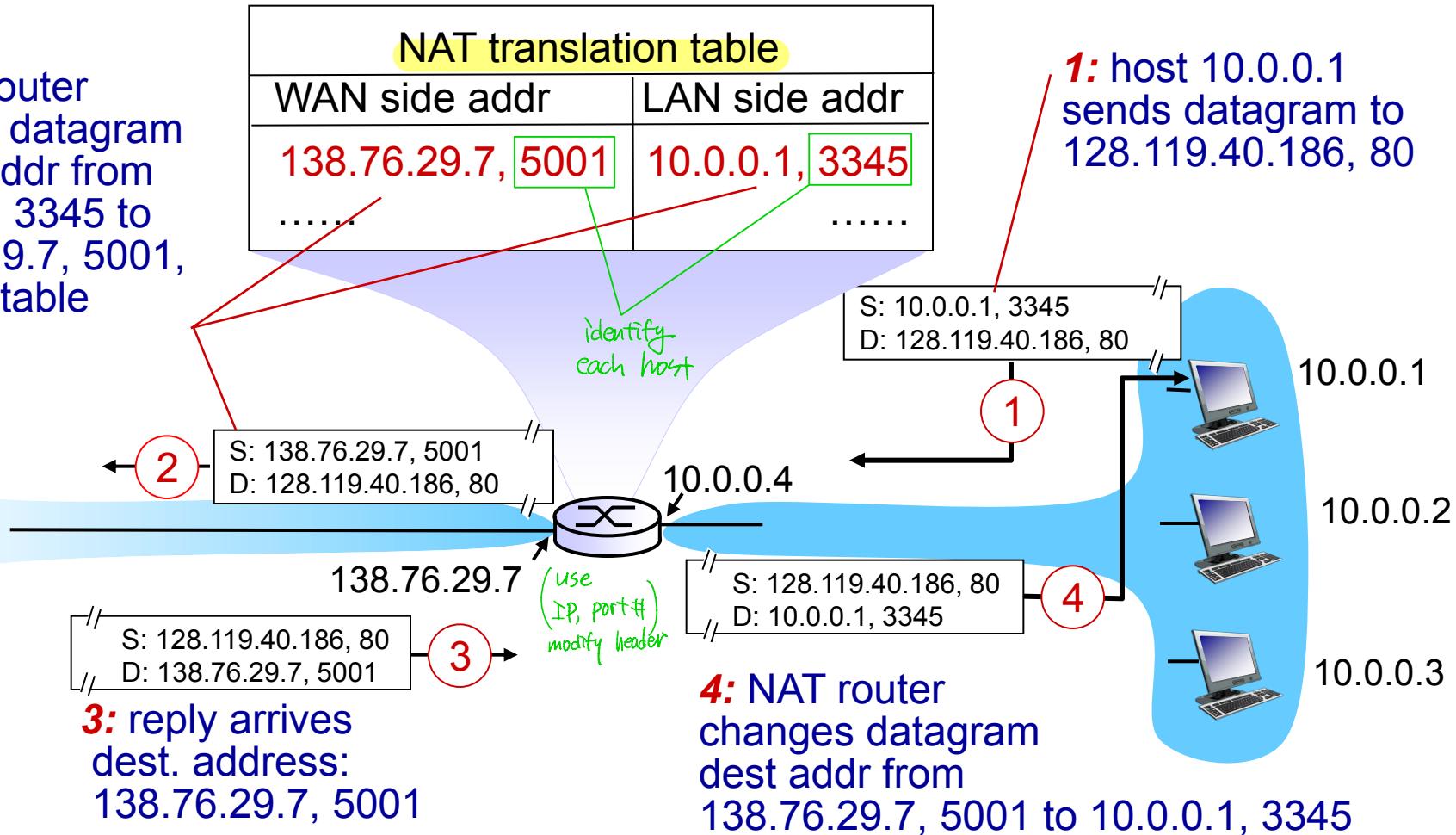
# NAT: network address translation

*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



\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# 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
  - address shortage should be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
    - NAT traversal: what if client wants to connect to server behind NAT?



# Chapter 4: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- **IPv6**

## 4.4 Generalized Forward and SDN

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

# IPv6: 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

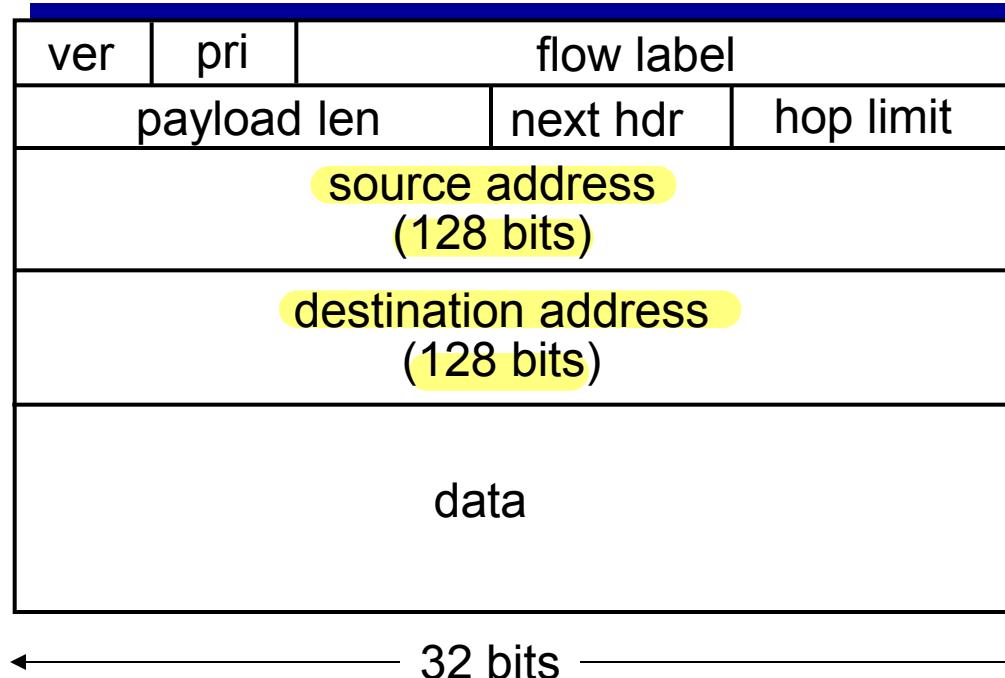
# IPv6 datagram format

*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

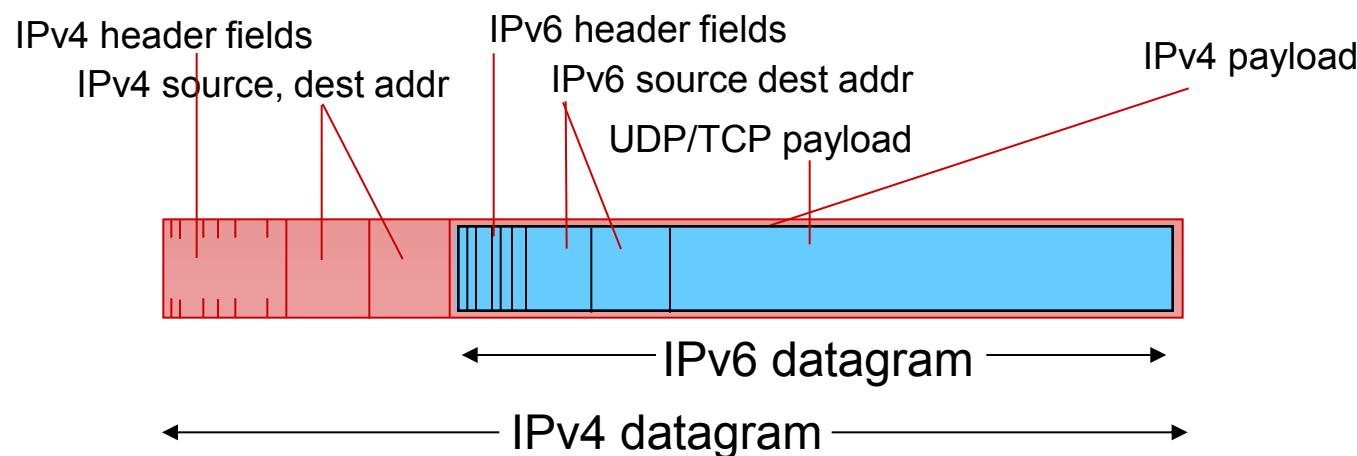


# Other changes from IPv4

- **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

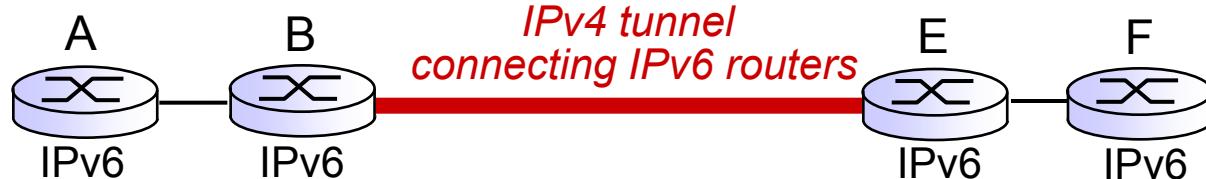
# Transition from IPv4 to IPv6

- 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

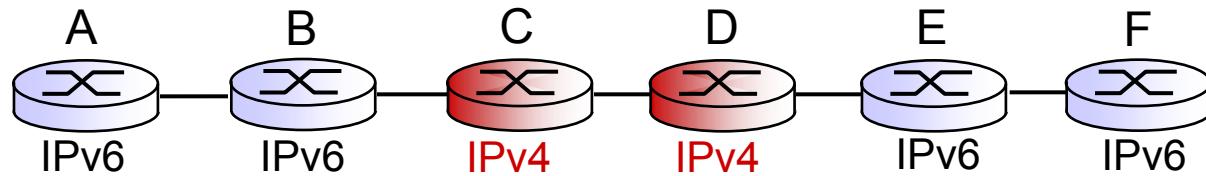


# Tunneling

logical view:

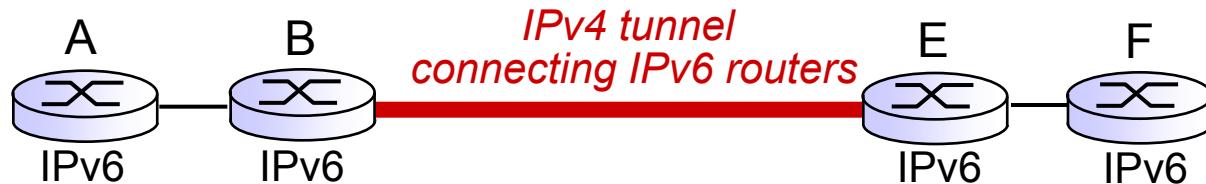


physical view:

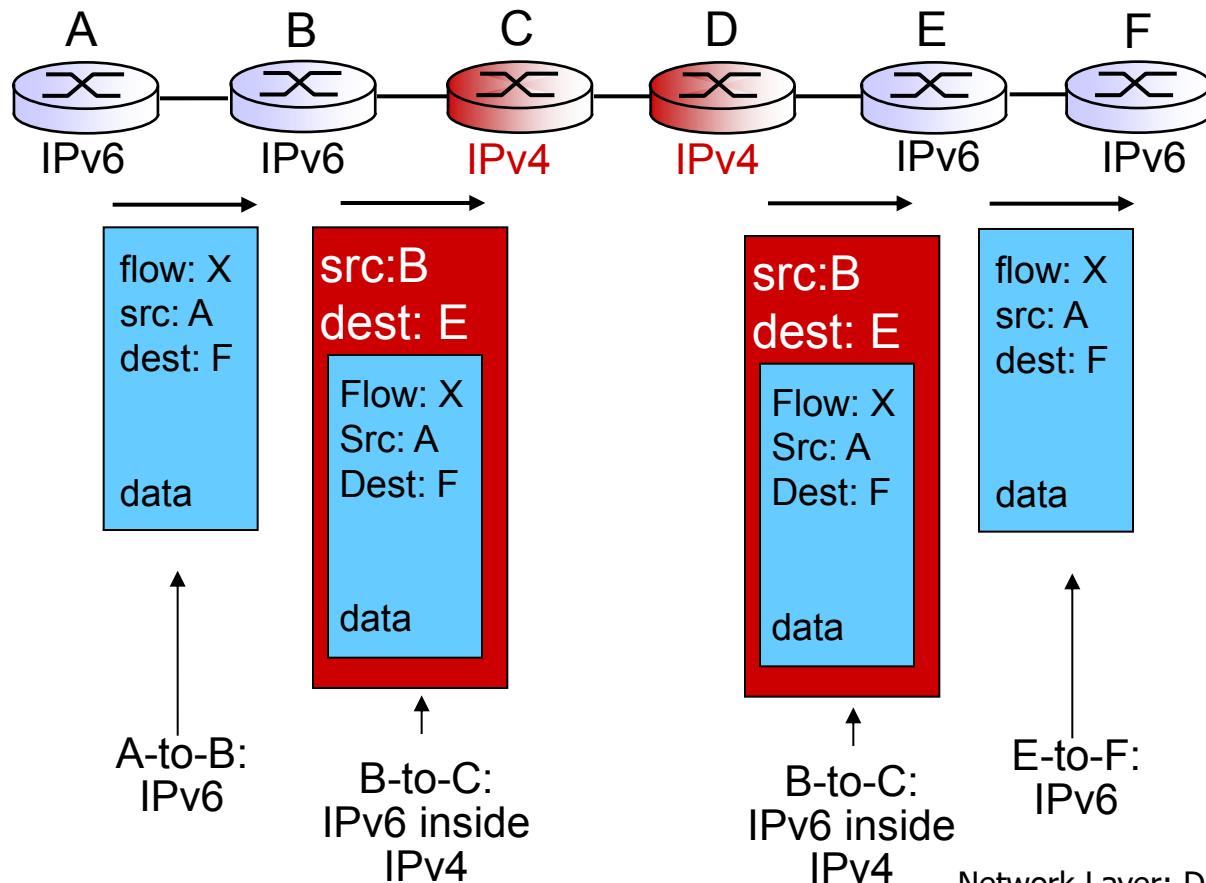


# Tunneling

logical view:



physical view:



# IPv6: adoption

- Google: 8% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable
- *Long (long!) time for deployment, use*
  - 20 years and counting!
  - think of application-level changes in last 20 years: WWW, Facebook, streaming media, Skype, ...
  - *Why?*

# Chapter 4: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

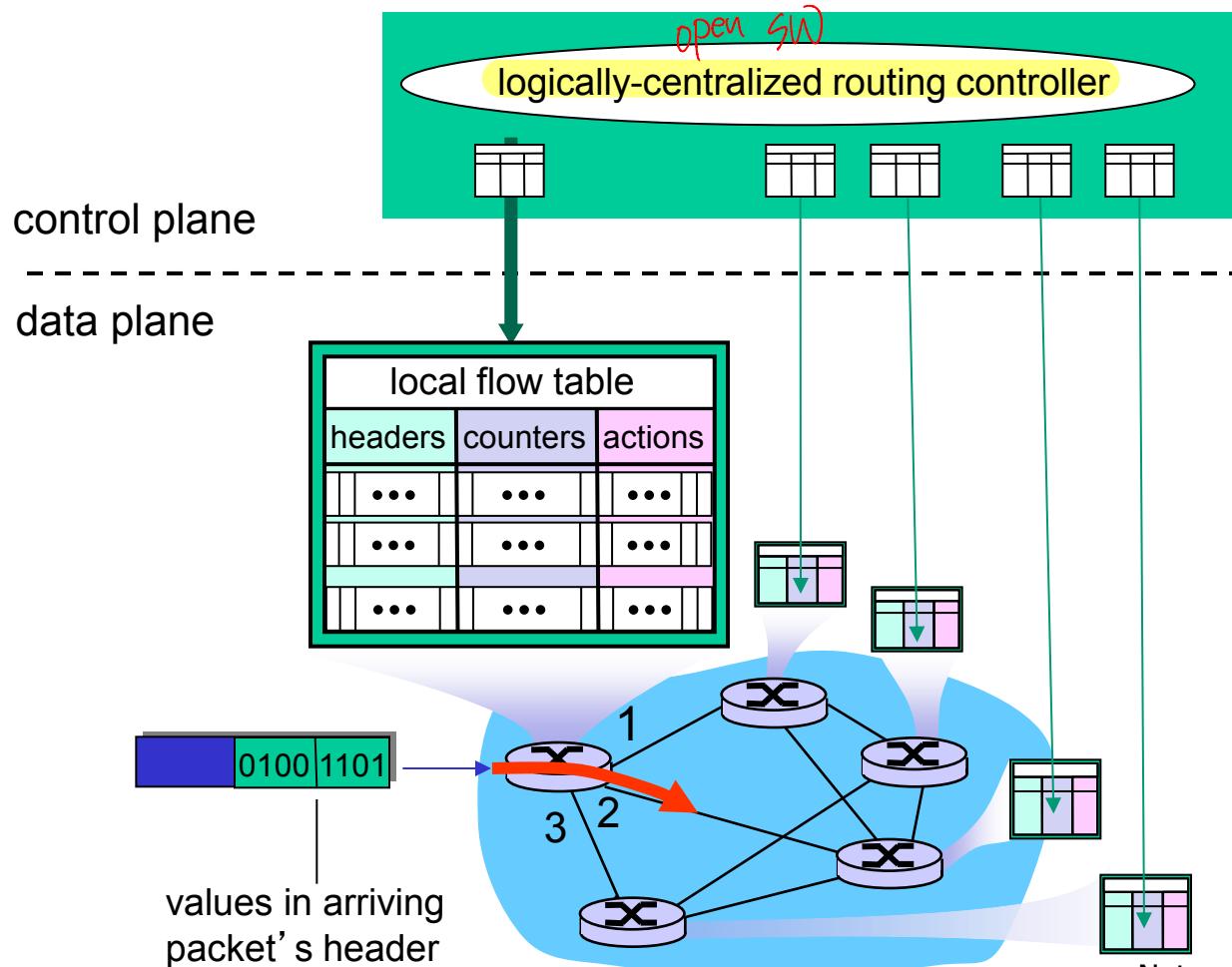
## 4.4 Generalized Forward and SDN

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

# Generalized Forwarding and SDN

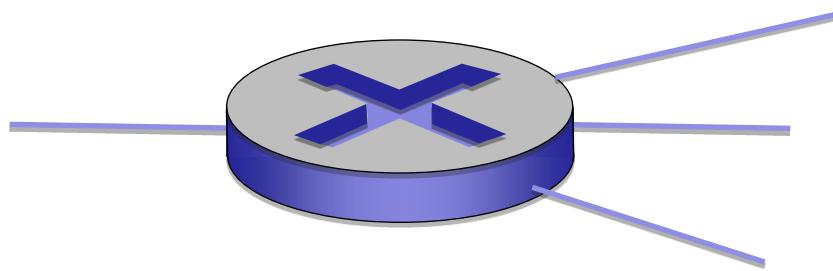
forwarding  
+  
other actions

- Each router contains a *flow table* that is computed and distributed by a *logically centralized routing controller*



# OpenFlow data plane abstraction

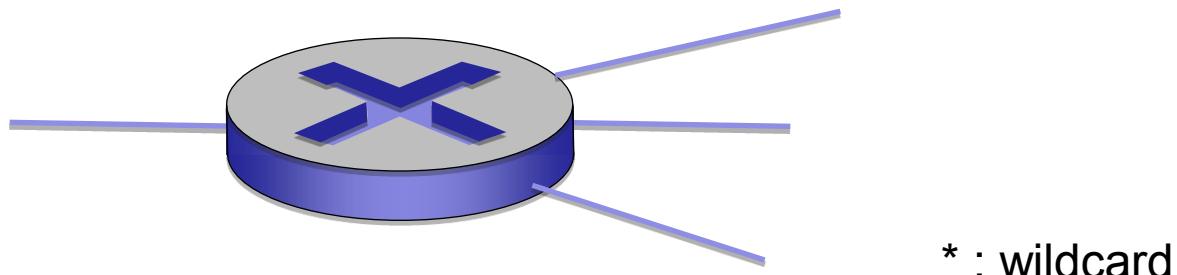
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern*: match values in packet header fields
  - *Actions*: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters*: #bytes and #packets



*Flow table in a router (computed and distributed by controller) define router's match+action rules*

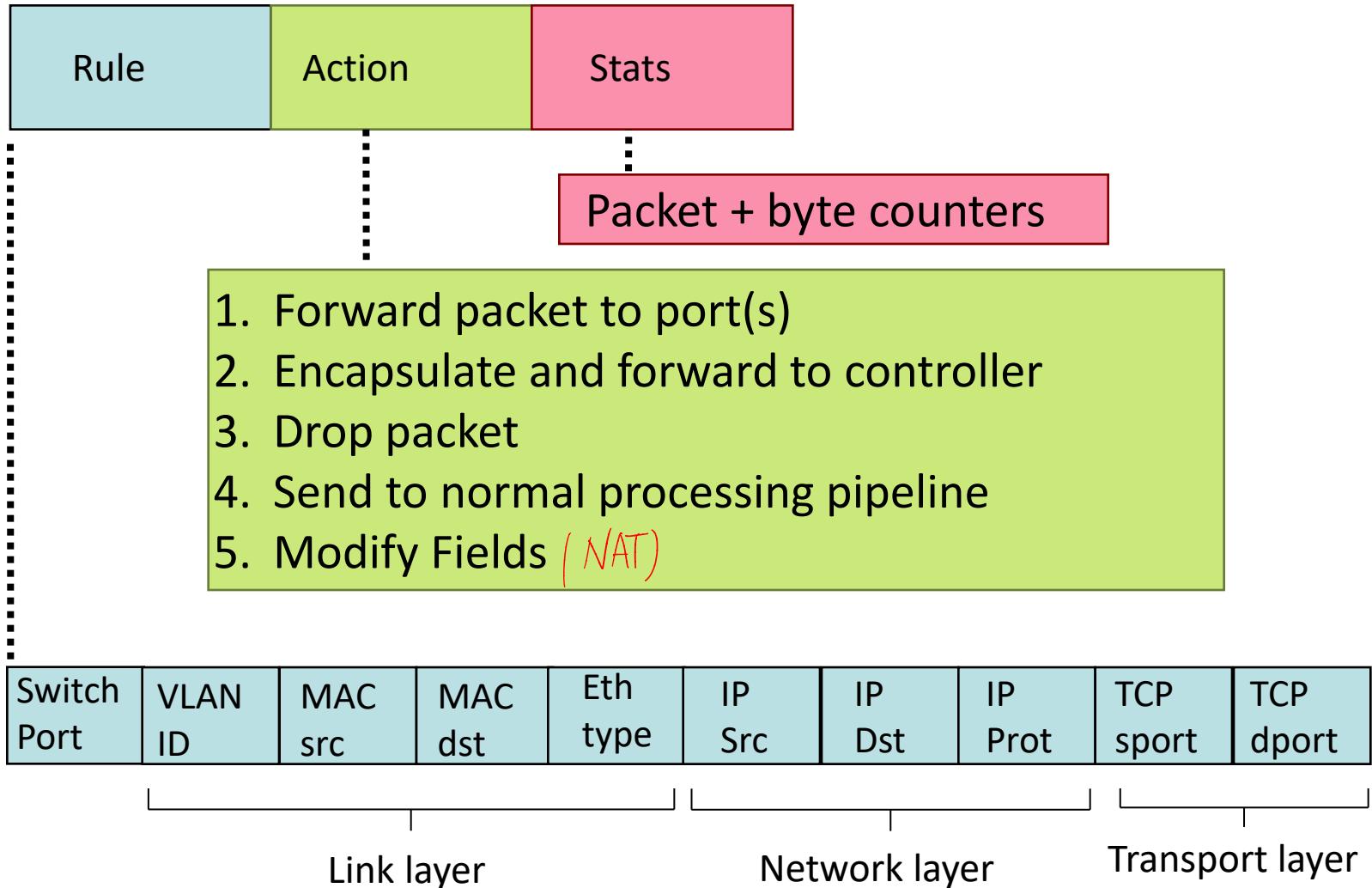
# OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern*: match values in packet header fields
  - *Actions*: *for matched packet*: drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters*: #bytes and #packets



1.  $\text{src}=1.2.*.*$ ,  $\text{dest}=3.4.5.* \rightarrow \text{drop}$
2.  $\text{src} = *.*.*.*$ ,  $\text{dest}=3.4.*.* \rightarrow \text{forward}(2)$
3.  $\text{src}=10.1.2.3$ ,  $\text{dest} = *.*.*.* \rightarrow \text{send to controller}$

# OpenFlow: Flow Table Entries



# Examples

## Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

*IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6*

## Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	*	22 drop

*do not forward (block) all datagrams destined to TCP port 22*

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

*do not forward (block) all datagrams sent by host 128.119.1.1*

# Examples

SDN  $\Rightarrow$  packet switch  
(router X, switch X)

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	port3

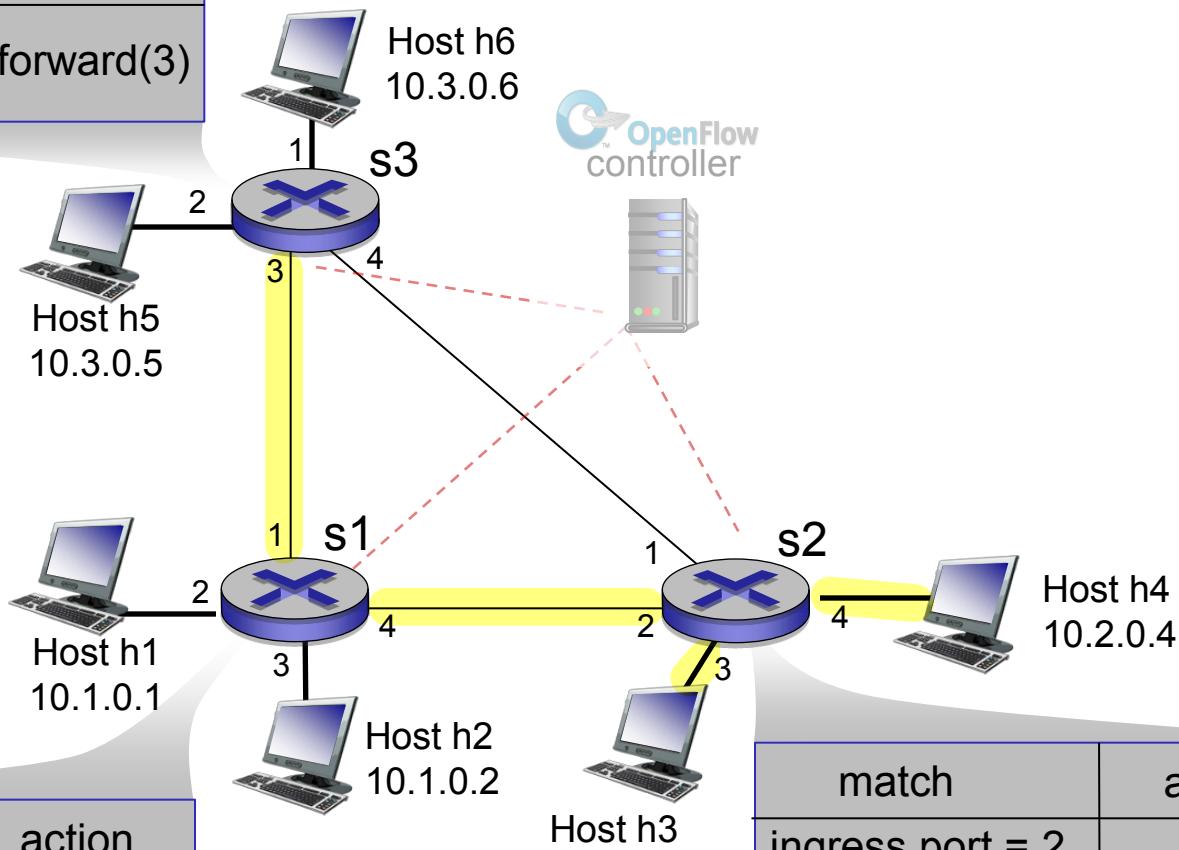
layer 2 frames from MAC address 22:A7:23:11:E1:02  
should be forwarded to output port 6

# OpenFlow abstraction

- *match+action*: unifies different kinds of devices
- Router
  - *match*: longest destination IP prefix
  - *action*: forward out a link
- Switch
  - *match*: destination MAC address
  - *action*: forward or flood
- Firewall
  - *match*: IP addresses and TCP/UDP port numbers
  - *action*: permit or deny
- NAT
  - *match*: IP address and port
  - *action*: rewrite address and port

# OpenFlow example

match	action
IP Src = 10.3.*.*	
IP Dst = 10.2.*.*	forward(3)



**Example:** datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

match	action
ingress port = 1	
IP Src = 10.3.*.*	forward(4)
IP Dst = 10.2.*.*	

match	action
ingress port = 2	
IP Dst = 10.2.0.3	forward(3)
ingress port = 2	
IP Dst = 10.2.0.4	forward(4)

# Chapter 4: done!

4.1 Overview of Network layer: data plane and control plane

4.2 What's inside a router

4.3 IP: Internet Protocol

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

4.4 Generalized Forward and SDN

- match plus action
- OpenFlow example

**Question:** how do forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

**Answer:** by the control plane (next chapter)

# Chapter 5

## Network Layer: The Control Plane

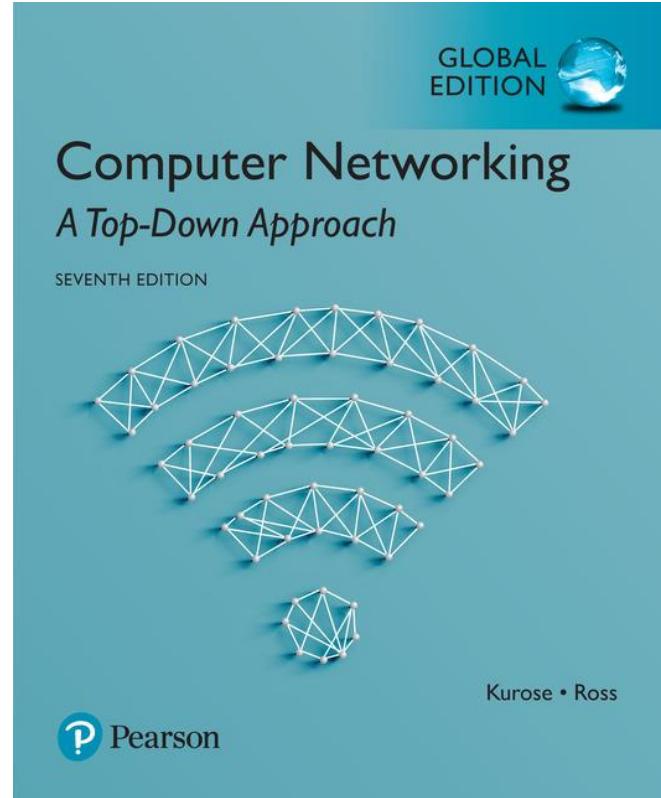
### A note on the use of these Powerpoint slides:

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

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

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016  
J.F Kurose and K.W. Ross, All Rights Reserved



## Computer Networking: A Top Down Approach

7<sup>th</sup> Edition, Global Edition  
Jim Kurose, Keith Ross  
Pearson  
April 2016

# Chapter 5: network layer control plane

*chapter goals:* understand principles behind network control plane

- traditional routing algorithms
- SDN controllers
- Internet Control Message Protocol
- network management

and their instantiation, implementation in the Internet:

- OSPF, BGP, OpenFlow, ODL and ONOS controllers, ICMP, SNMP

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# Network-layer functions

*Recall: two network-layer functions:*

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

*data plane*

*control plane*

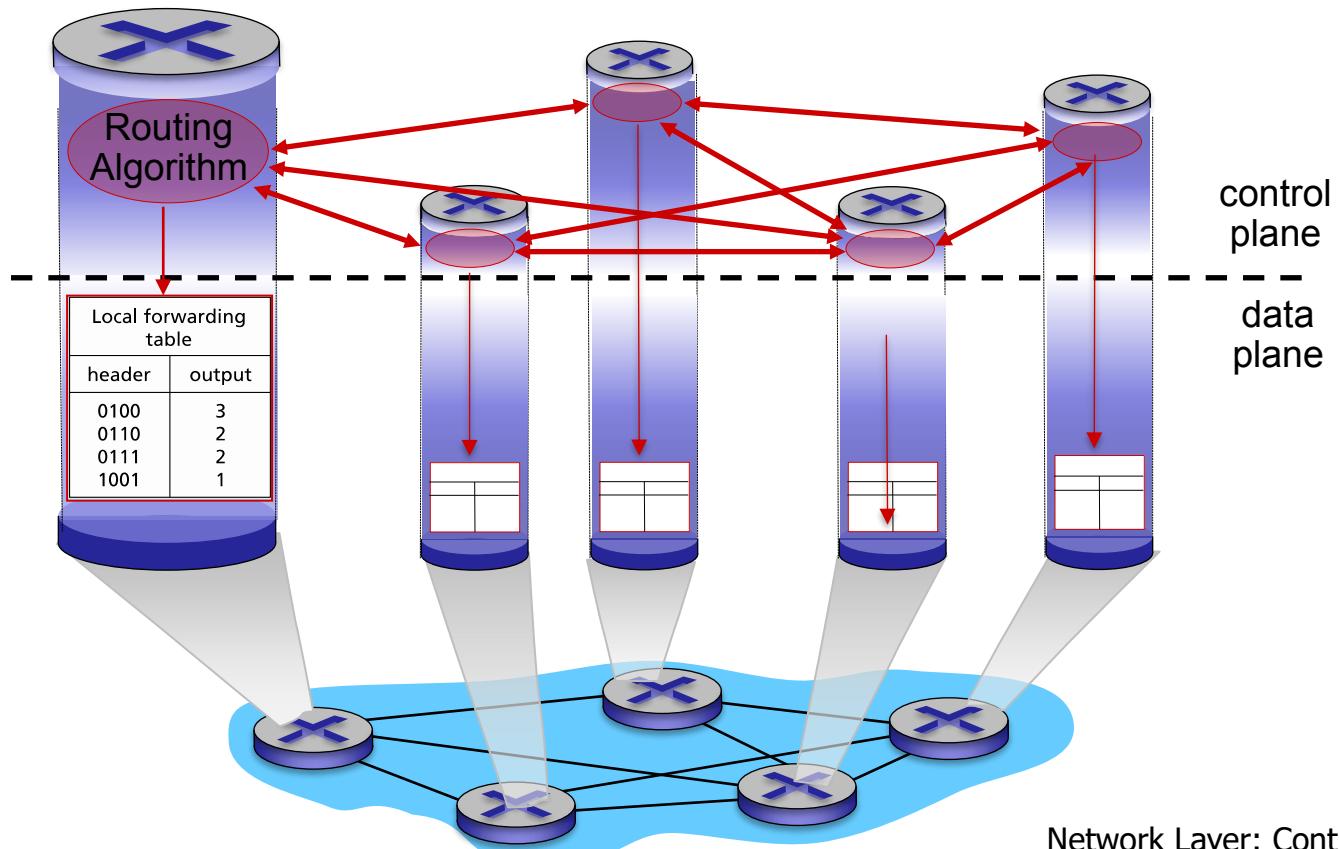
*Two approaches to structuring network control plane:*

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

algorithm < shortest path  
distance vector routing

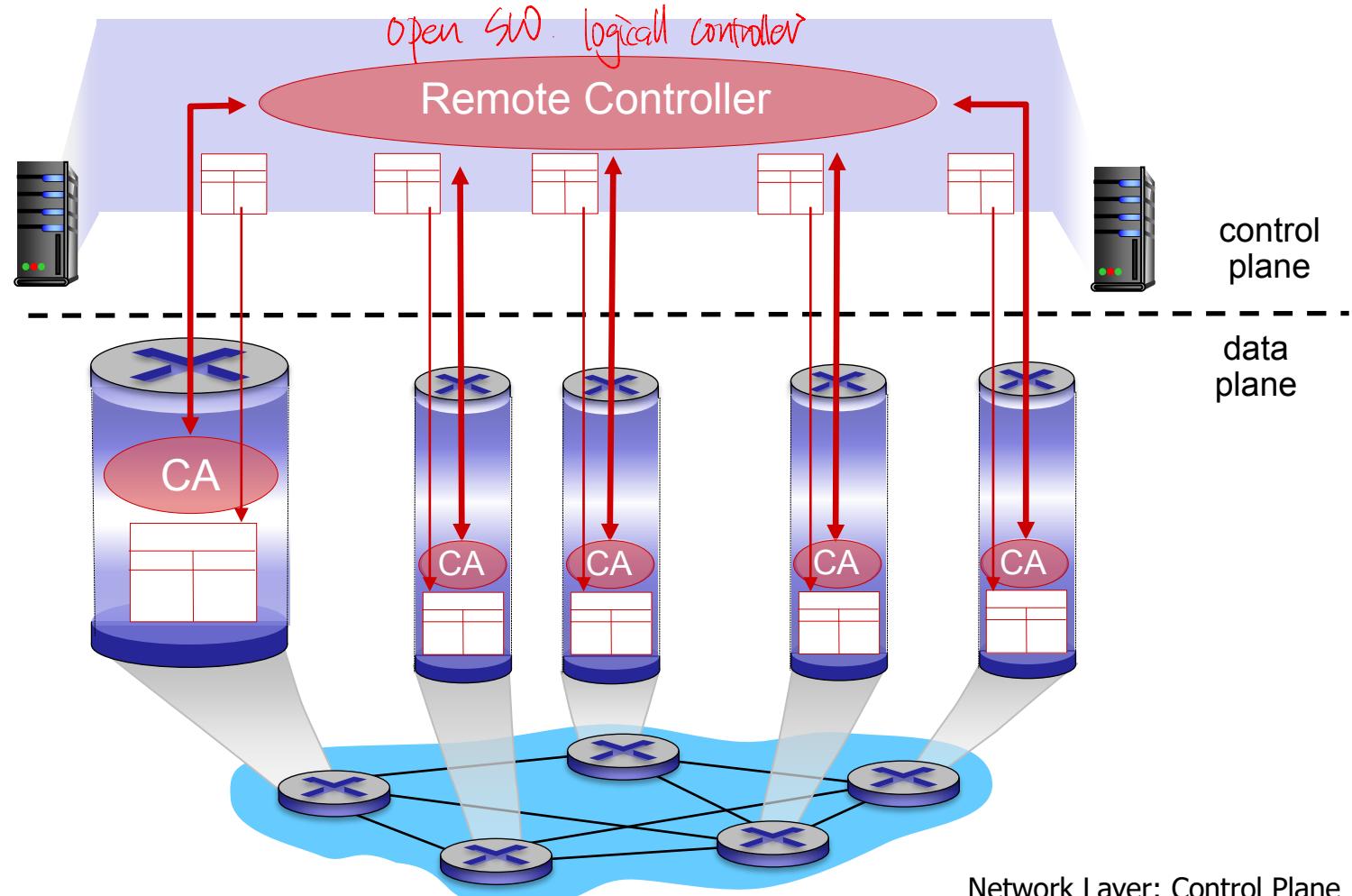
# Per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



# Logically centralized control plane (SDN)

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

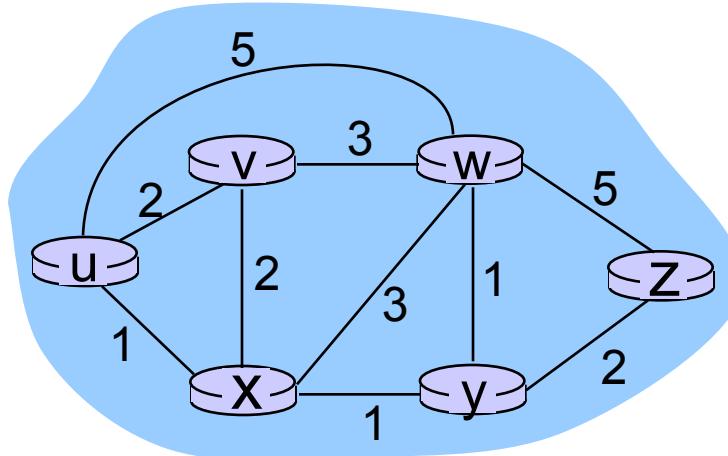
5.7 Network management and SNMP

# Routing protocols

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

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

# Graph abstraction of the network



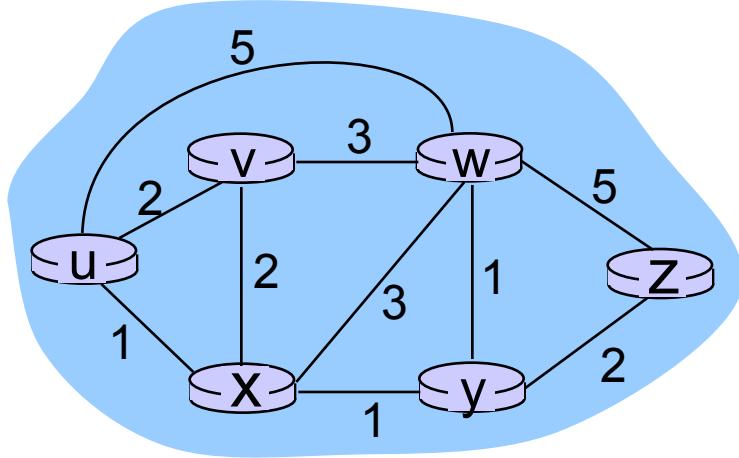
graph:  $G = (N, E)$

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

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

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

$$C(u-x-y-z) = 4$$

**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:* shortest path

- all routers have complete topology, link cost info by flooding
- “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:* update info slowly

- routes change slowly over time

*dynamic:* update info quickly

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

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state

- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# 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  $v$  \*/

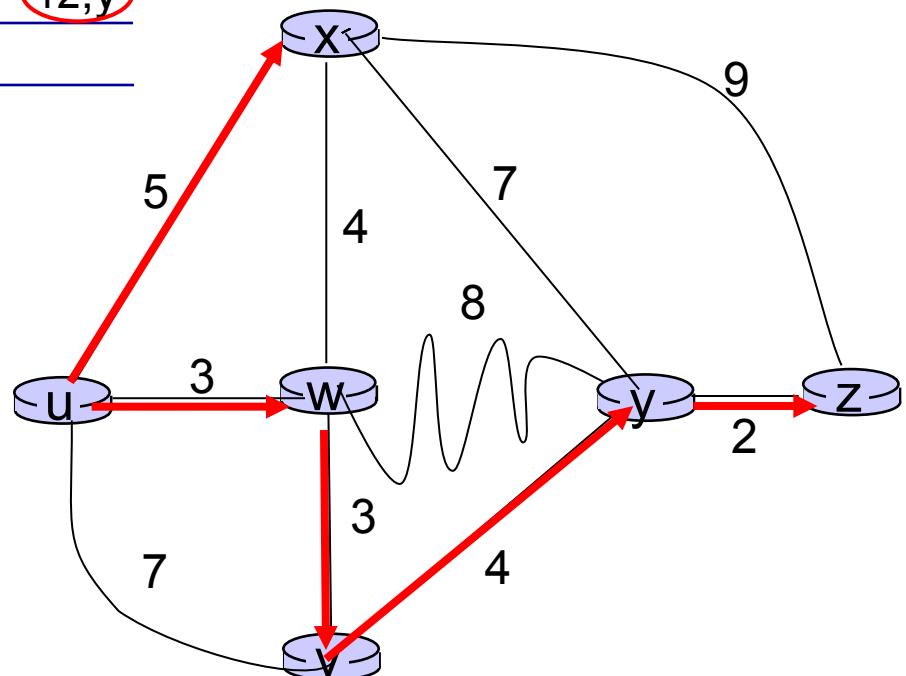
15 **until all nodes in  $N'$**

# 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	$\infty$	$\infty$
1	uw	6,w	5,u	11,w	$\infty$	
2	uwx	6,w		11,w	14,x	
3	UWXV			10,v	14,x	
4	UWXVY				12,y	
5	UWXVYZ					

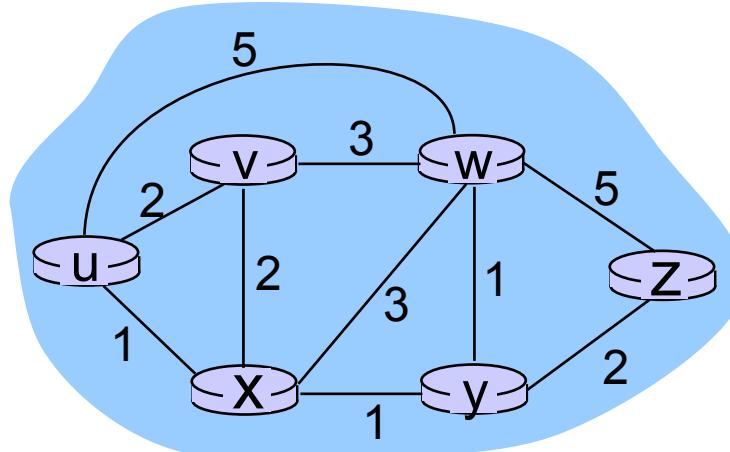
**notes:**

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



# Dijkstra's algorithm: another example

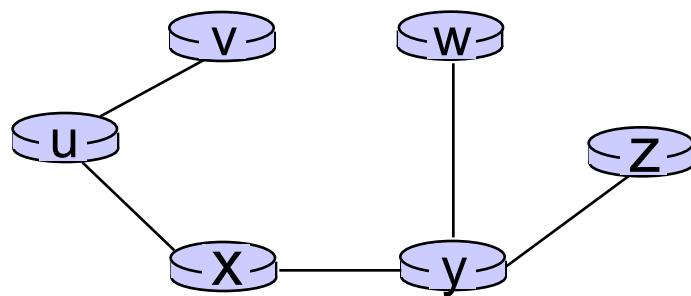
Step	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	$\infty$	$\infty$
1	ux	2,u	4,x		2,x	$\infty$
2	uxy	2,u	3,y		4,y	
3	uxyv		3,y		4,y	
4	uxyvw					4,y
5	uxyvwz					



\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# 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)

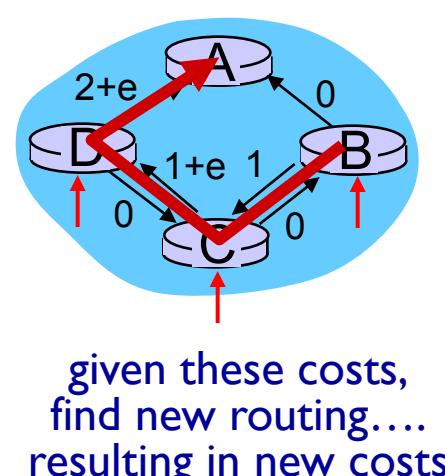
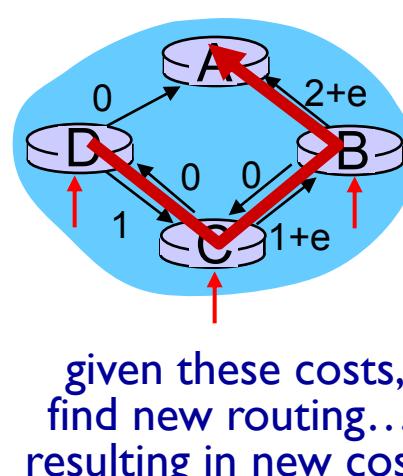
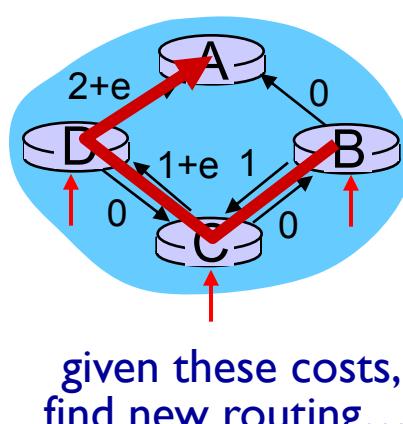
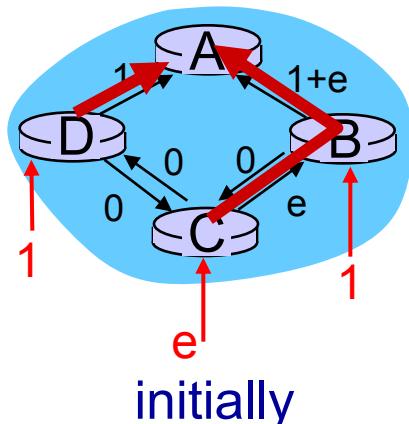
# Dijkstra's algorithm, discussion

*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:



# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- **distance vector**

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# Distance vector algorithm

*Bellman-Ford equation (dynamic programming)*

let

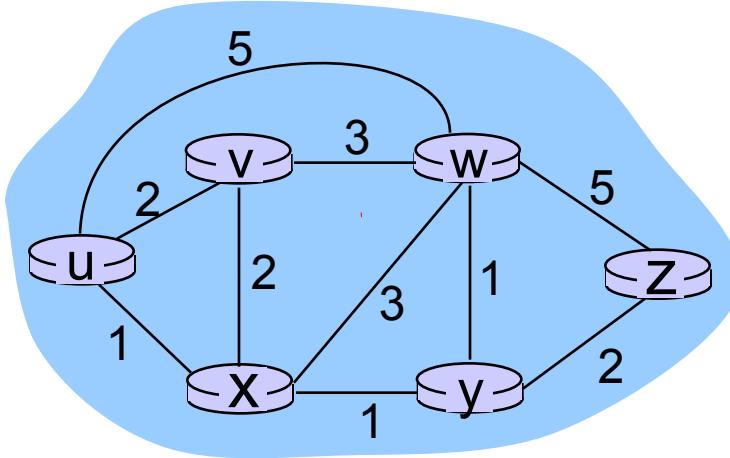
$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$

then

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

cost from neighbor  $v$  to destination  $y$   
cost to neighbor  $v$   
*min taken over all neighbors  $v$  of  $x$*   
*directly connected*

# Bellman-Ford example



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

B-F equation says:

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

when  $d$  is changed  
 $\Rightarrow$  advertisement

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:*

*wait* for (change in local link cost or msg from neighbor)

*recompute* estimates

if DV to any dest has changed, *notify* neighbors

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

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

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

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

**node x  
table**

	cost to		
	x	y	z
x	0	2	7
y	$\infty$	$\infty$	$\infty$
z	$\infty$	$\infty$	$\infty$

**node y  
table**

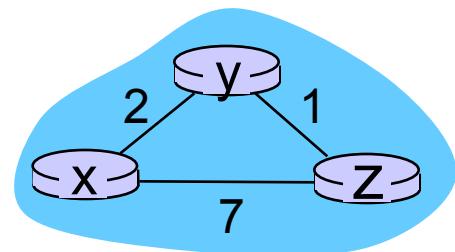
	cost to		
	x	y	z
x	$\infty$	$\infty$	$\infty$
y	2	0	1
z	$\infty$	$\infty$	$\infty$

**node z  
table**

	cost to		
	x	y	z
x	$\infty$	$\infty$	$\infty$
y	$\infty$	$\infty$	$\infty$
z	7	1	0

	cost to		
	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**

	x	y	z
from	0	2	7
x	0	2	7
y	$\infty$	$\infty$	$\infty$
z	$\infty$	$\infty$	$\infty$

**node y  
table**

	x	y	z
from	$\infty$	$\infty$	$\infty$
x	$\infty$	$\infty$	$\infty$
y	2	0	1
z	$\infty$	$\infty$	$\infty$

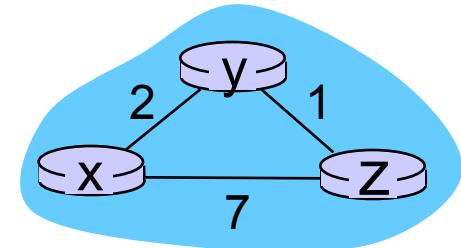
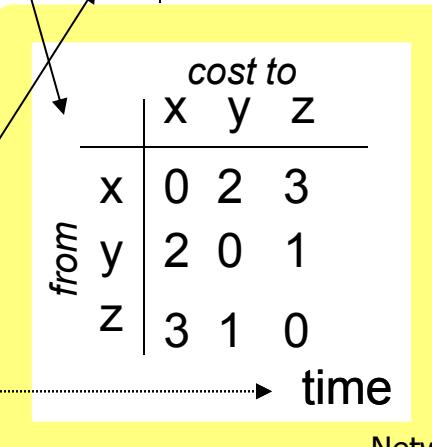
**node z  
table**

	x	y	z
from	$\infty$	$\infty$	$\infty$
x	$\infty$	$\infty$	$\infty$
y	$\infty$	$\infty$	$\infty$
z	7	1	0

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

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

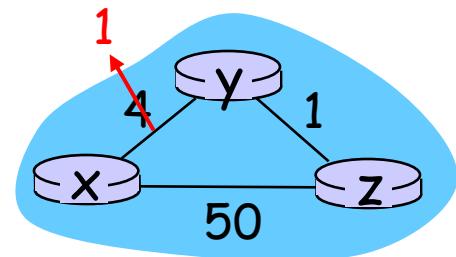
	x	y	z
from	0	2	3
x	0	2	3
y	2	0	1
z	3	1	0



# 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.

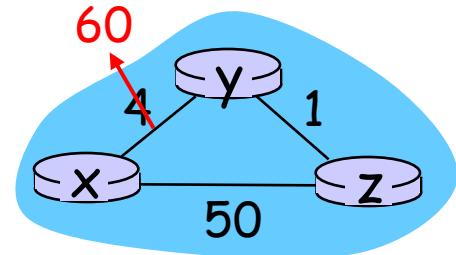
$t_2$ : y receives z' s update, updates its distance table. y' s least costs do *not* change, so y does *not* send a message to z.

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# 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 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# Making routing scalable

our routing study thus far - idealized

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

**scale:** with billions of destinations:

- can't store all destinations 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

# Internet approach to scalable routing

aggregate routers into regions known as “autonomous systems” (AS) (a.k.a. “domains”)

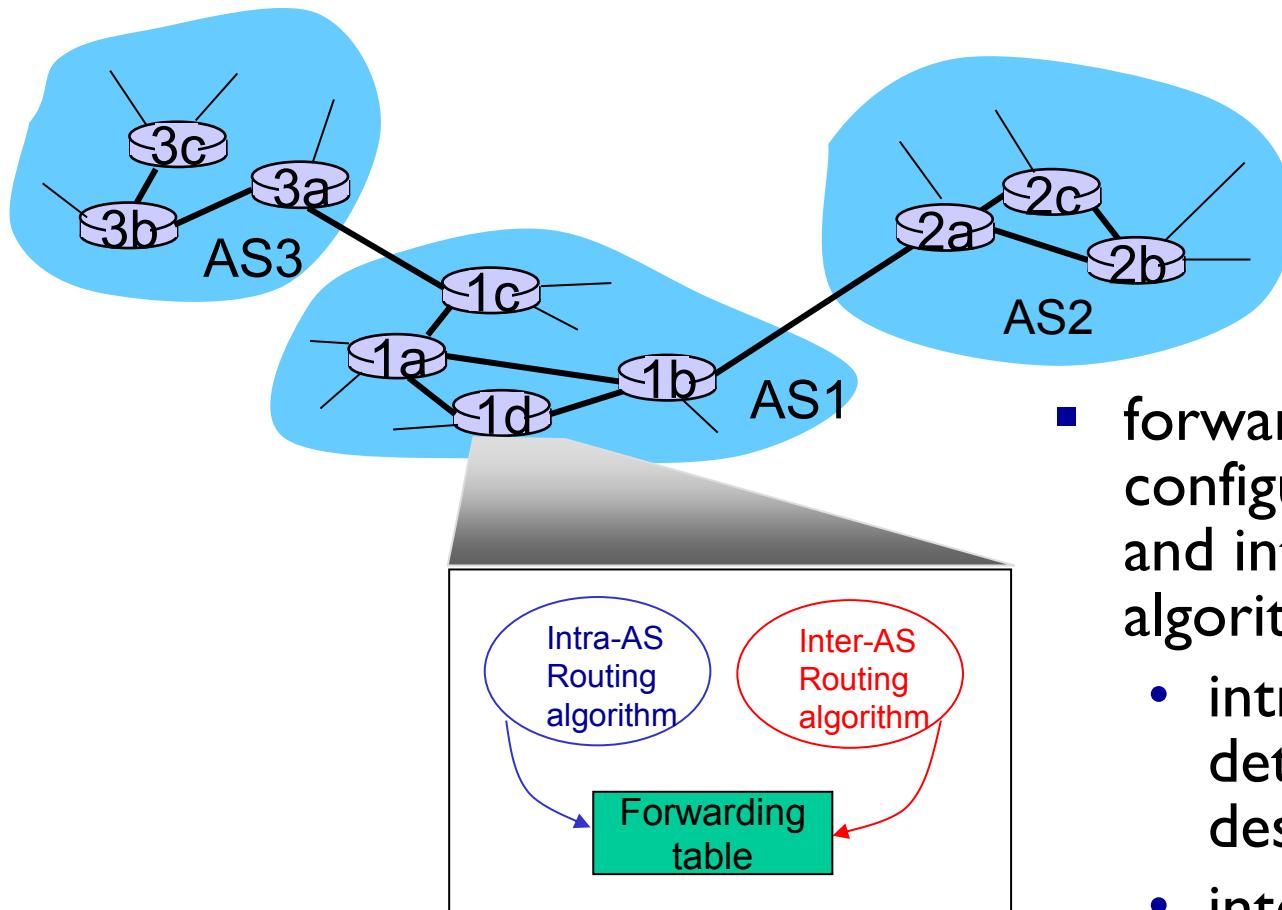
## intra-AS routing

- routing among hosts, routers in same AS (“network”)
- all routers in AS must run *same* intra-domain protocol
- routers in *different* AS can run *different* intra-domain routing protocol
- gateway router: at “edge” of its own AS, has link(s) to router(s) in other AS’es

## inter-AS routing

- routing among AS’es
- gateways perform inter-domain routing (as well as intra-domain routing)

# Interconnected ASes



- **forwarding table**  
configured by both intra-  
and inter-AS routing  
algorithm
  - **intra-AS routing**  
determine entries for  
destinations within AS
  - **inter-AS & intra-AS**  
determine entries for  
external destinations

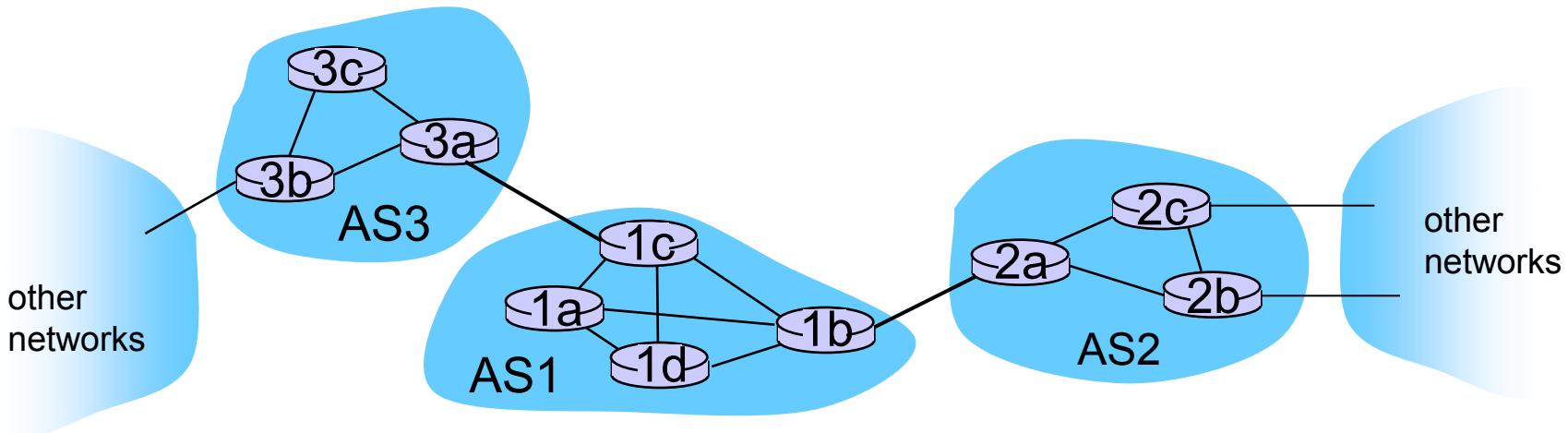
# Inter-AS tasks

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

*AS1 must:*

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

*job of inter-AS 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 (IS-IS protocol essentially same as OSPF)
  - IGRP: Interior Gateway Routing Protocol  
**(Cisco proprietary for decades, until 2016)**

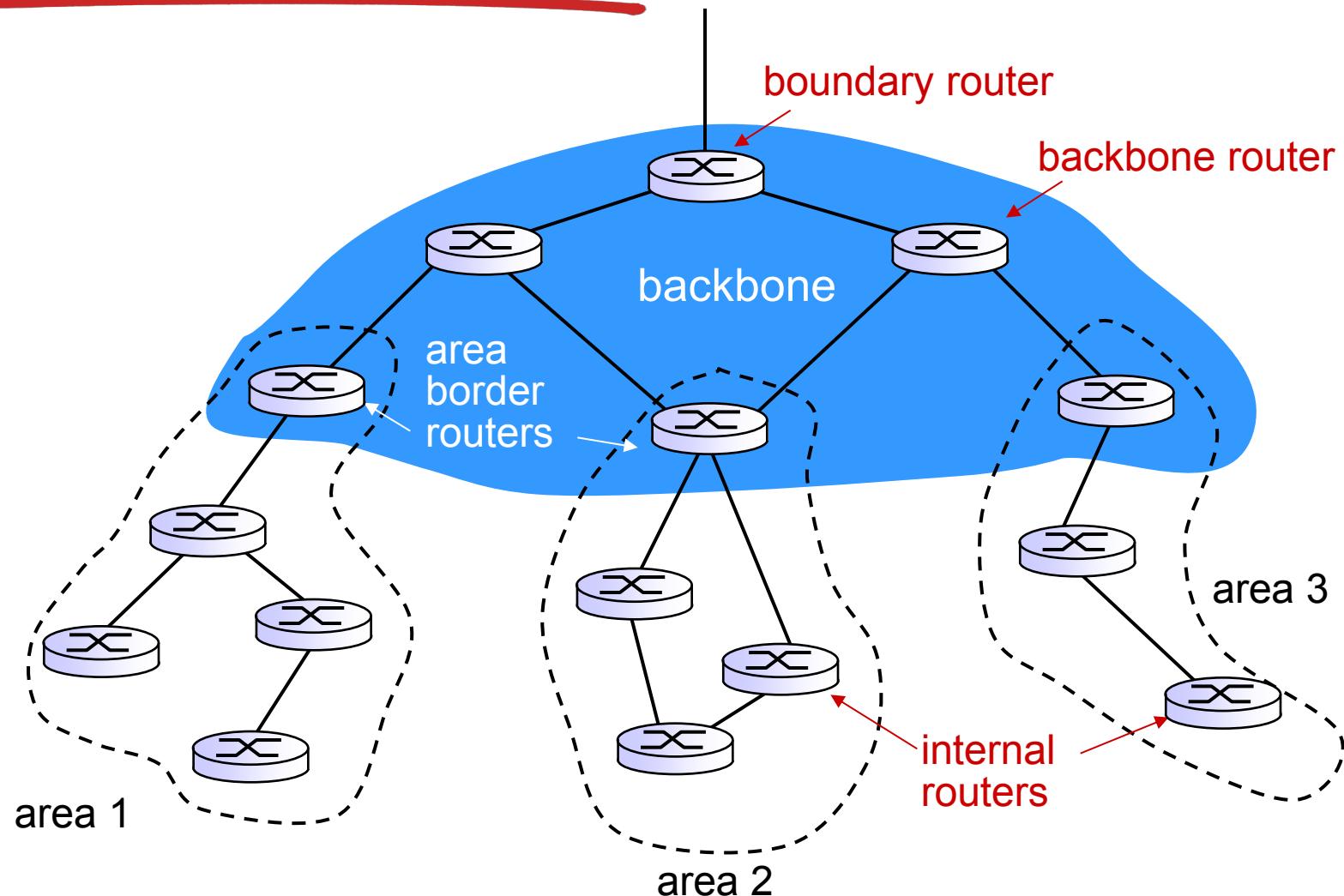
# OSPF (Open Shortest Path First)

- “open”: publicly available
- uses link-state algorithm
  - link state packet dissemination
  - topology map at each node
  - route computation using Dijkstra’s algorithm
- router floods OSPF link-state advertisements to all other routers in *entire AS*
  - carried in OSPF messages directly over IP (rather than TCP or UDP)
  - link state: for each attached link
- *IS-IS routing* protocol: nearly identical to OSPF

# OSPF “advanced” features

- **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 **multi-cast** 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' es.

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state

- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

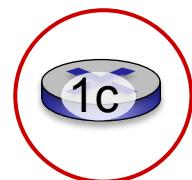
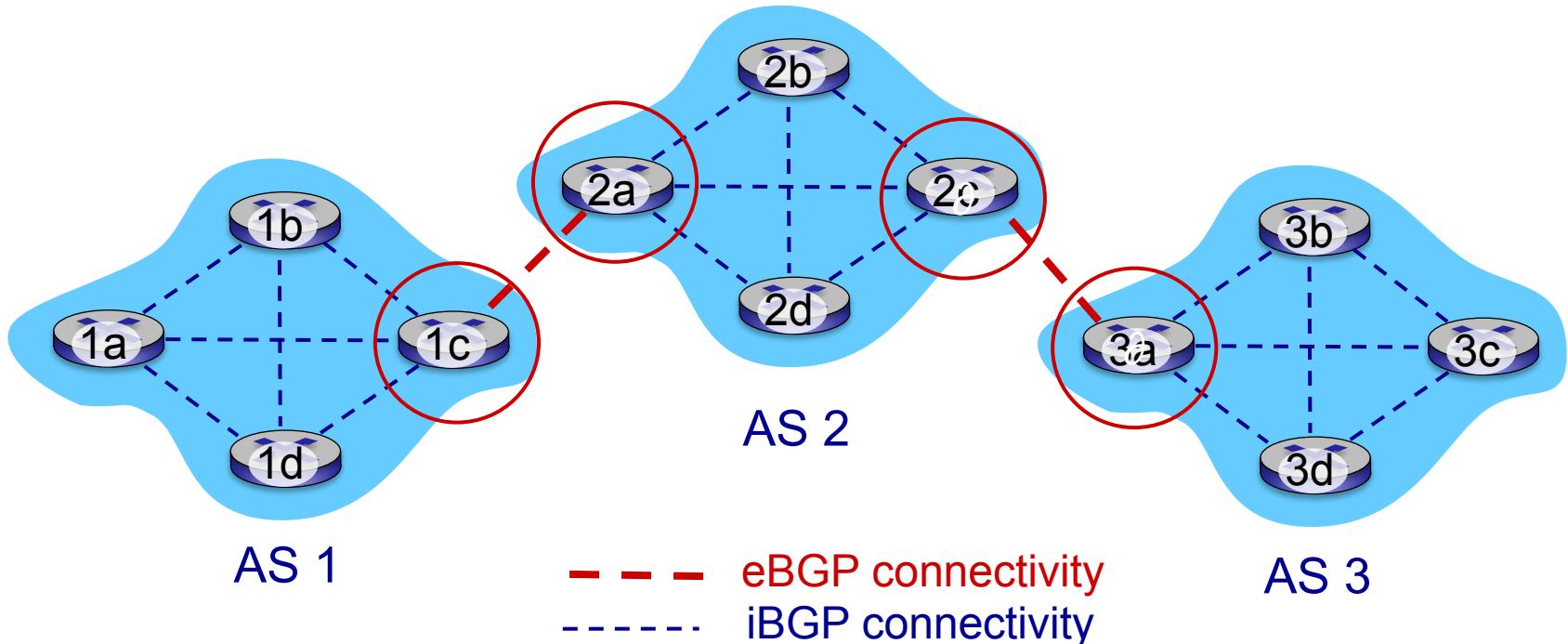
5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# 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 ASes
  - **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*”

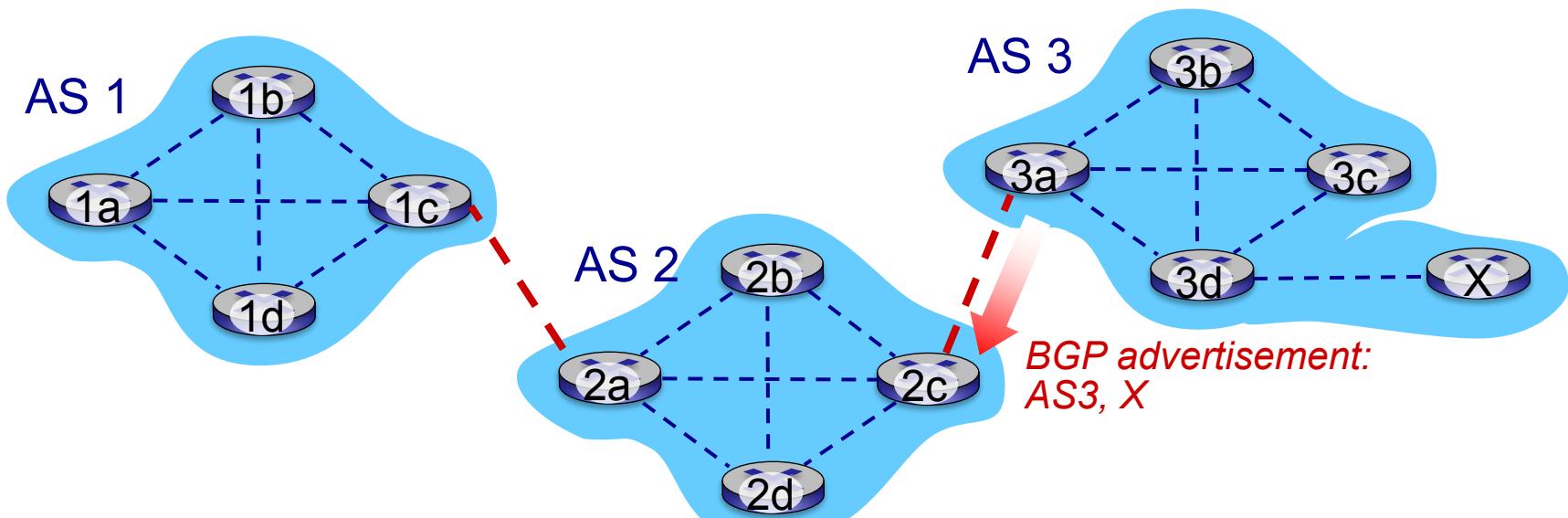
# eBGP, iBGP connections



gateway routers run both eBGP and iBGP protocols

# BGP basics

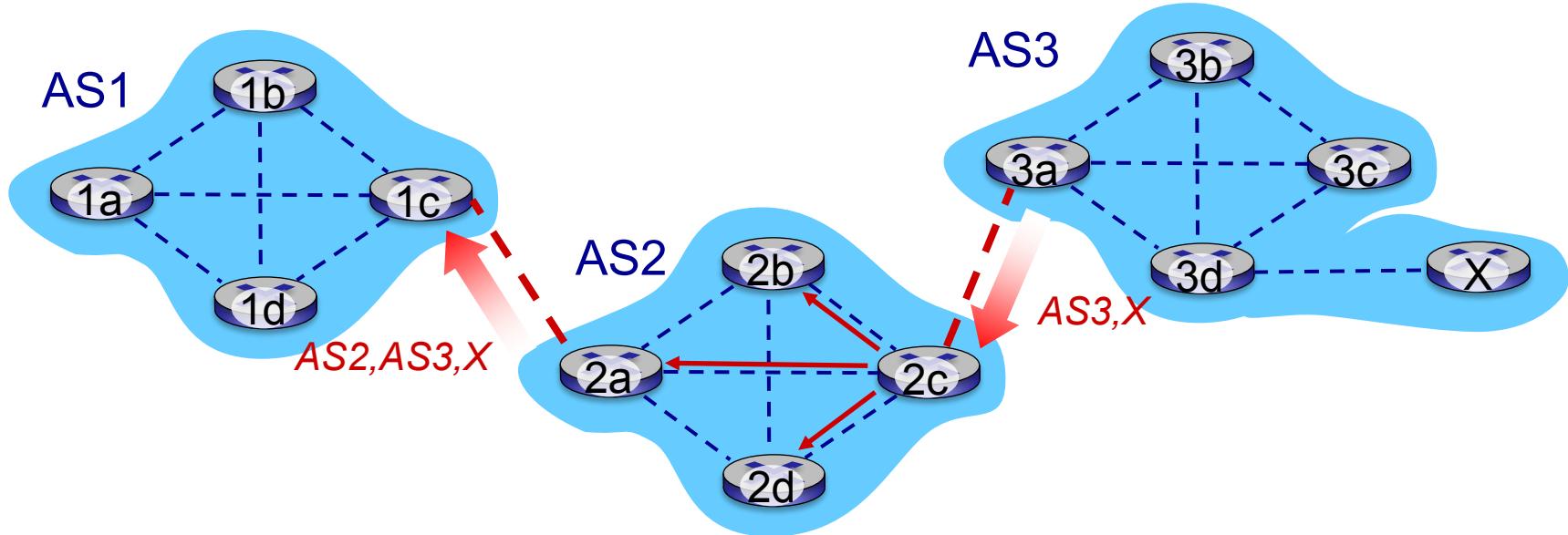
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a “path vector” protocol)
- when AS3 gateway router 3a advertises path **AS3,X** to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



# Path attributes and BGP routes

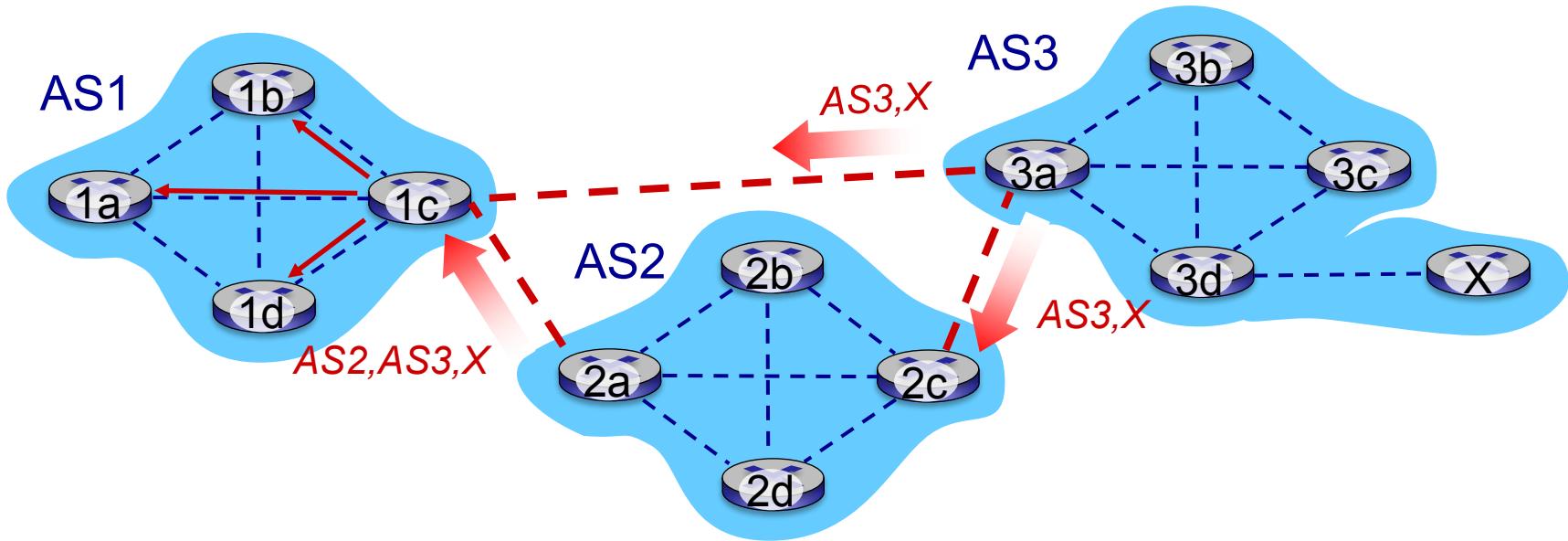
- advertised prefix includes BGP attributes
  - prefix + attributes = “route”
- two important attributes:
  - **AS-PATH**: list of ASes through which prefix advertisement has passed
  - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS
- *Policy-based routing*:
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

# BGP path advertisement



- AS2 router 2c receives path advertisement **AS3,X** (via eBGP) from AS3 router 3a
- Based on AS2 policy, AS2 router 2c accepts path **AS3,X**, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path **AS2, AS3,X** to AS1 router 1c

# BGP path advertisement



gateway router may learn about **multiple** paths to destination:

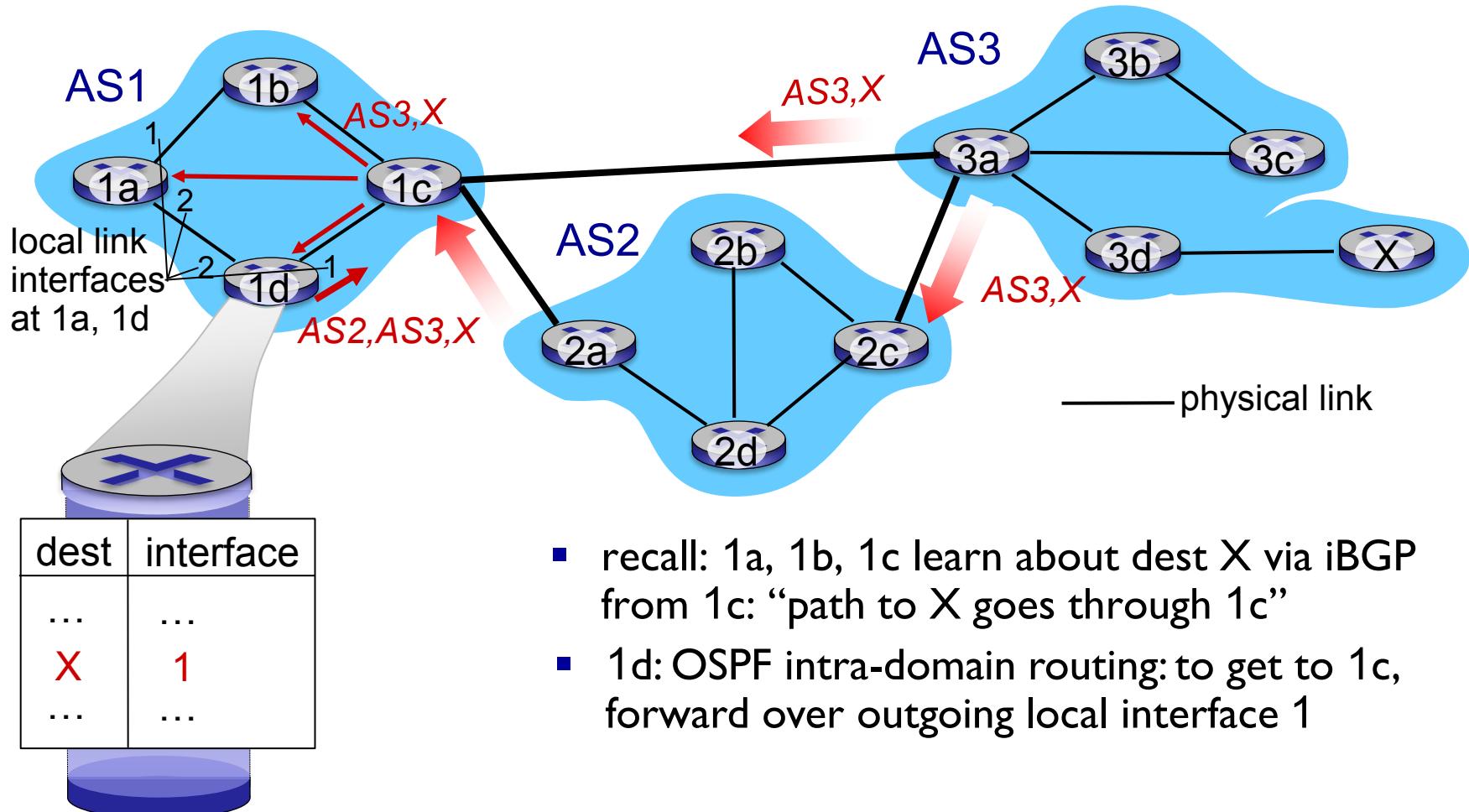
- AS1 gateway router 1c learns path **AS2,AS3,X** from 2a
- AS1 gateway router 1c learns path **AS3,X** from 3a
- Based on policy, AS1 gateway router 1c chooses path **AS3,X, and advertises path within AS1 via iBGP**

# BGP messages

- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - **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, OSPF, forwarding table entries

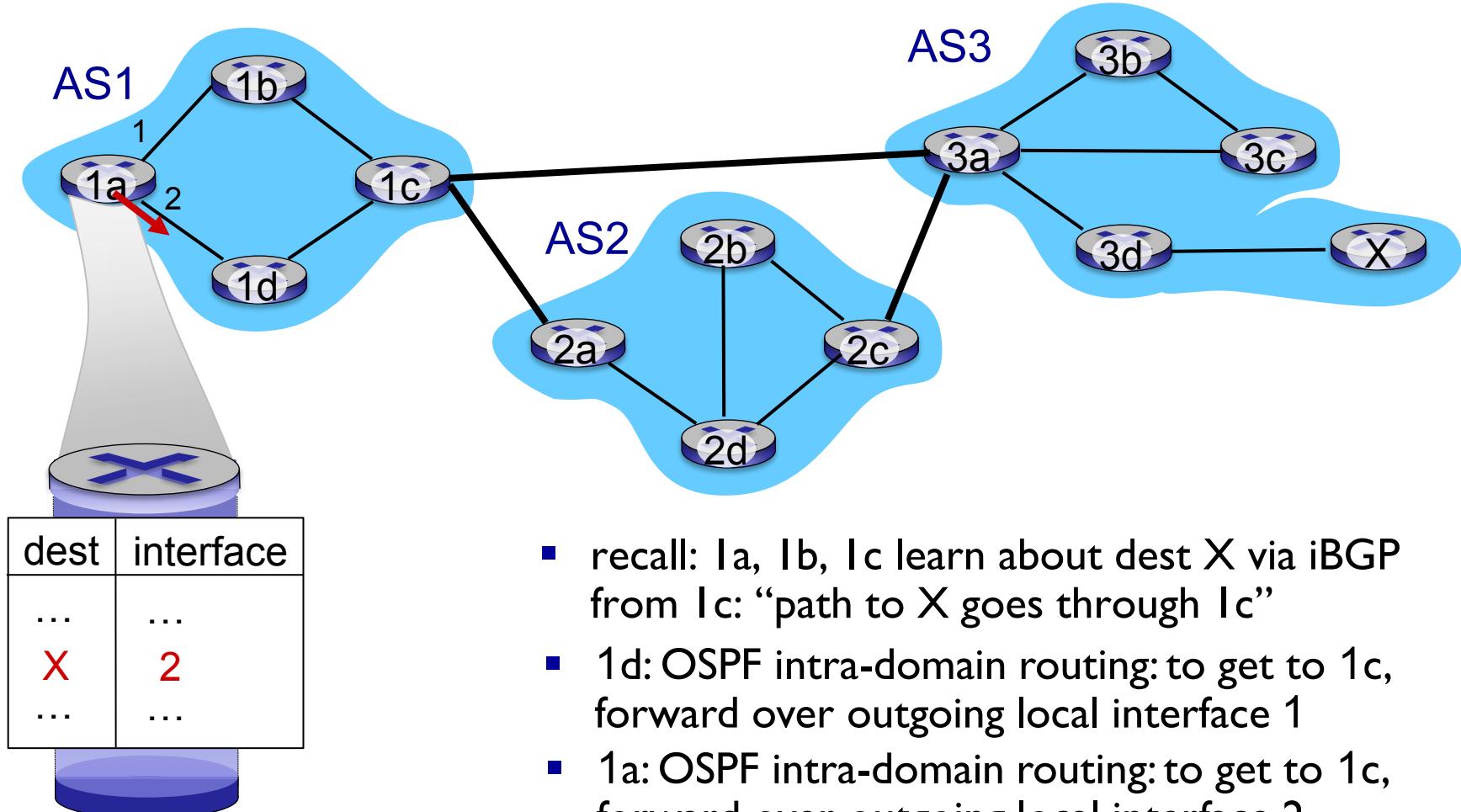
Q: how does router set forwarding table entry to distant prefix?



- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1

# BGP, OSPF, forwarding table entries

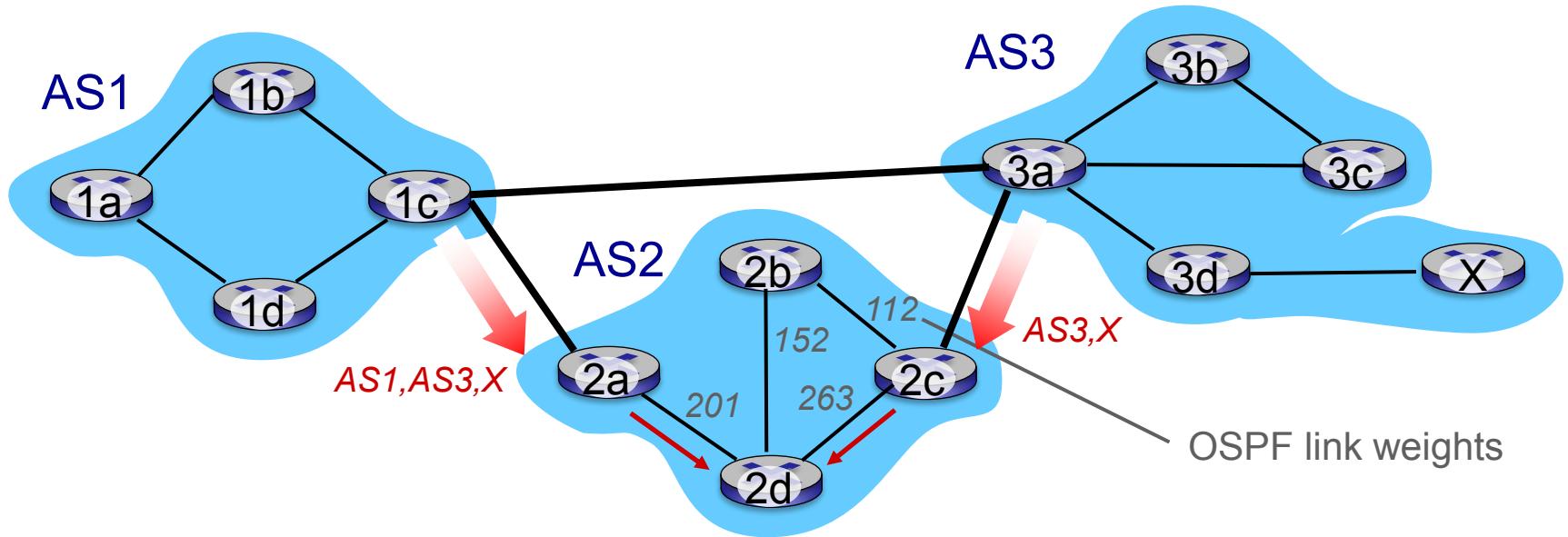
Q: how does router set forwarding table entry to distant prefix?



# BGP route selection

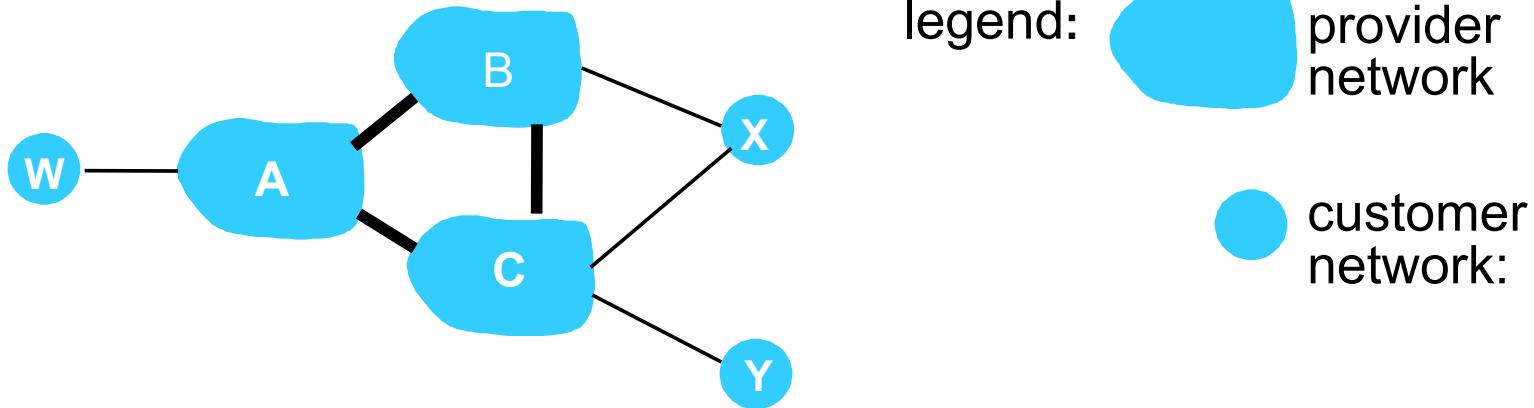
- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

# Hot Potato Routing



- 2d learns (via iBGP) it can route to X via 2a or 2c
- *hot potato routing*: choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

# BGP: achieving policy via advertisements



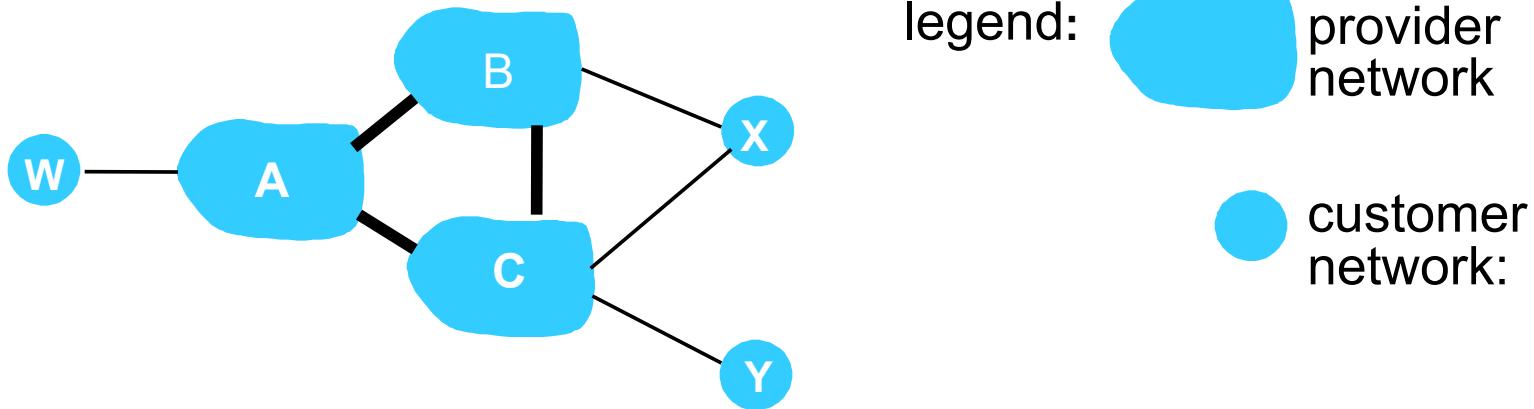
legend:

- provider network
- customer network:

Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A advertises path Aw to B and to C
- B *chooses not to advertise* BAw to C:
  - B gets no “revenue” for routing CBAw, since none of C, A, w are B’s customers
  - C does not learn about CBAw path
- C will route CAw (not using B) to get to w

# BGP: achieving policy via advertisements



Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

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

# 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 5: outline

5.1 introduction

5.2 routing protocols

- link state

- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

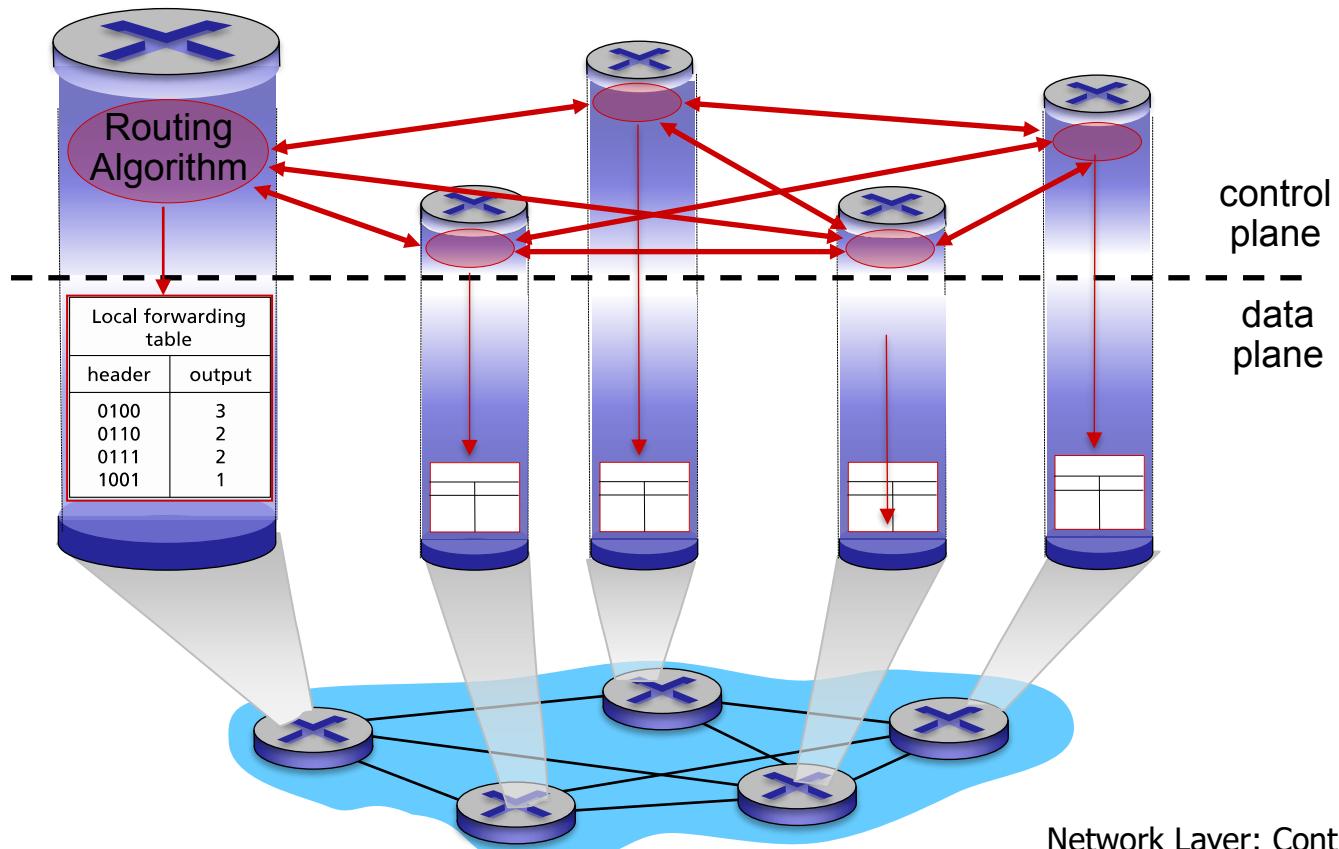
5.7 Network management and SNMP

# Software defined networking (SDN)

- Internet network layer: historically has been implemented via distributed, per-router approach
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

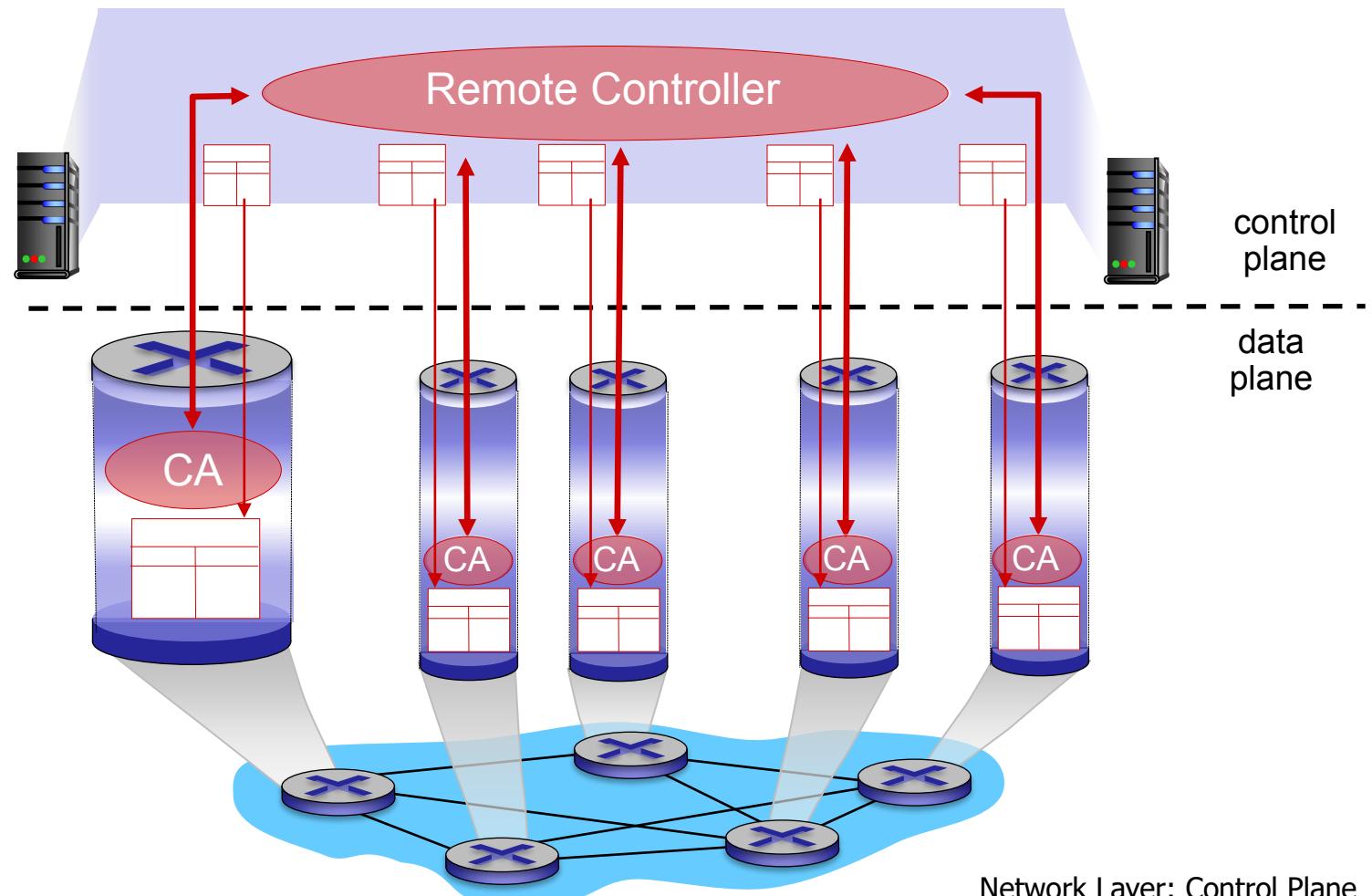
# Recall: per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



# Recall: logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables

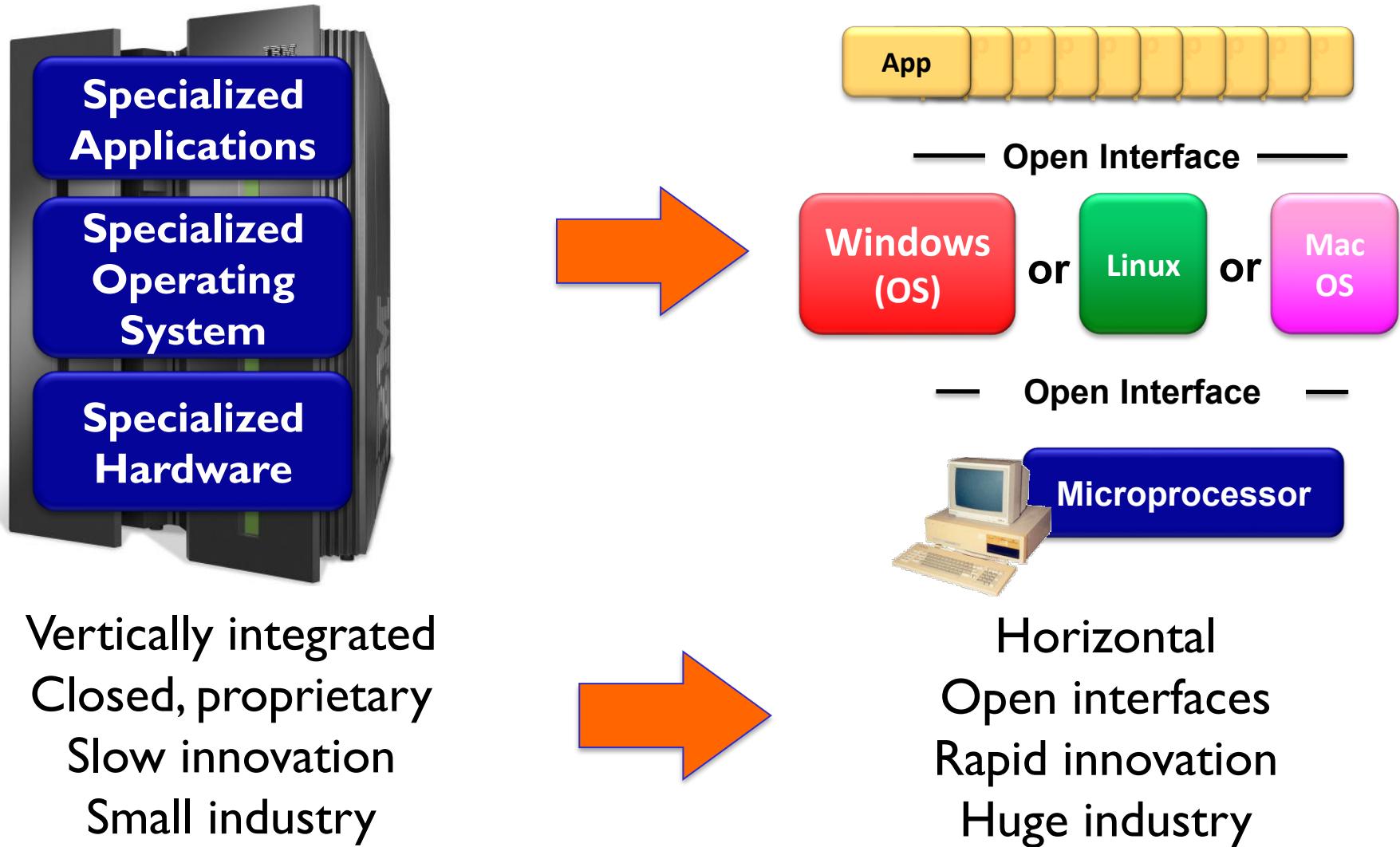


# Software defined networking (SDN)

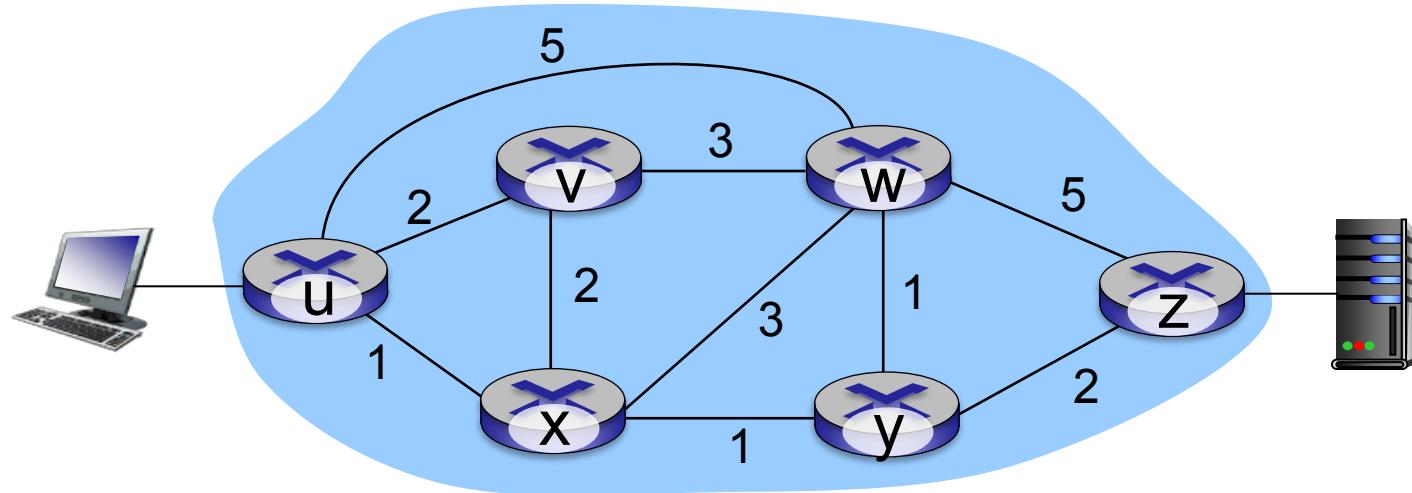
*Why* a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows “programming” routers
  - centralized “programming” easier: compute tables centrally and distribute
  - distributed “programming”: more difficult: compute tables as result of distributed algorithm (protocol) implemented in each and every router
- open (non-proprietary) implementation of control plane

# Analogy: mainframe to PC evolution\*



# Traffic engineering: difficult traditional routing

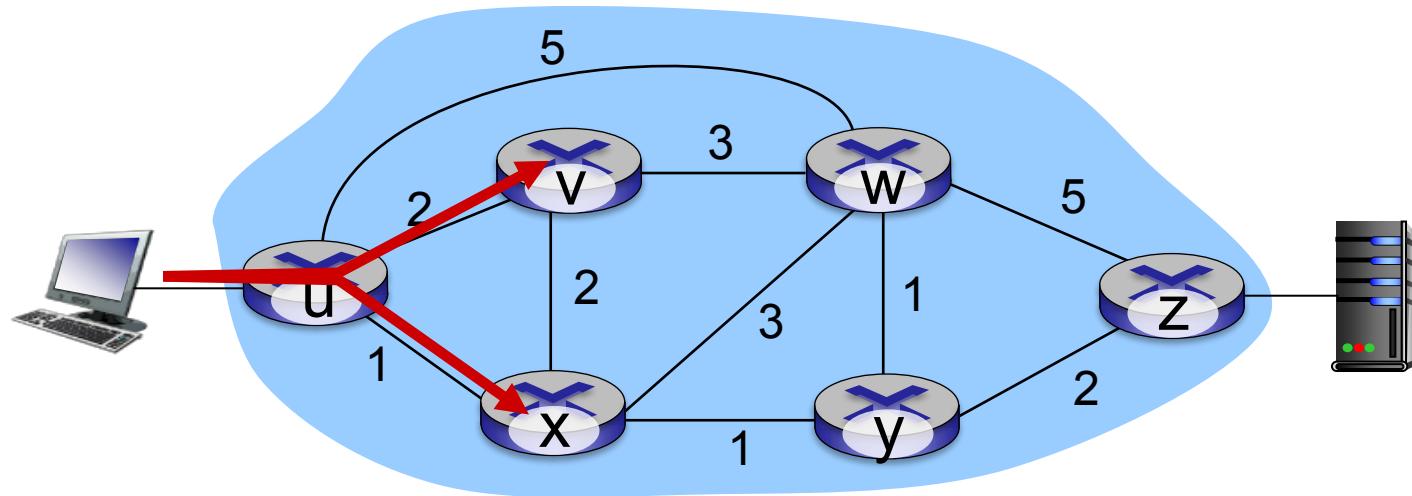


Q: what if network operator wants u-to-z traffic to flow along  $uvwz$ , x-to-z traffic to flow  $xwyz$ ?

A: need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

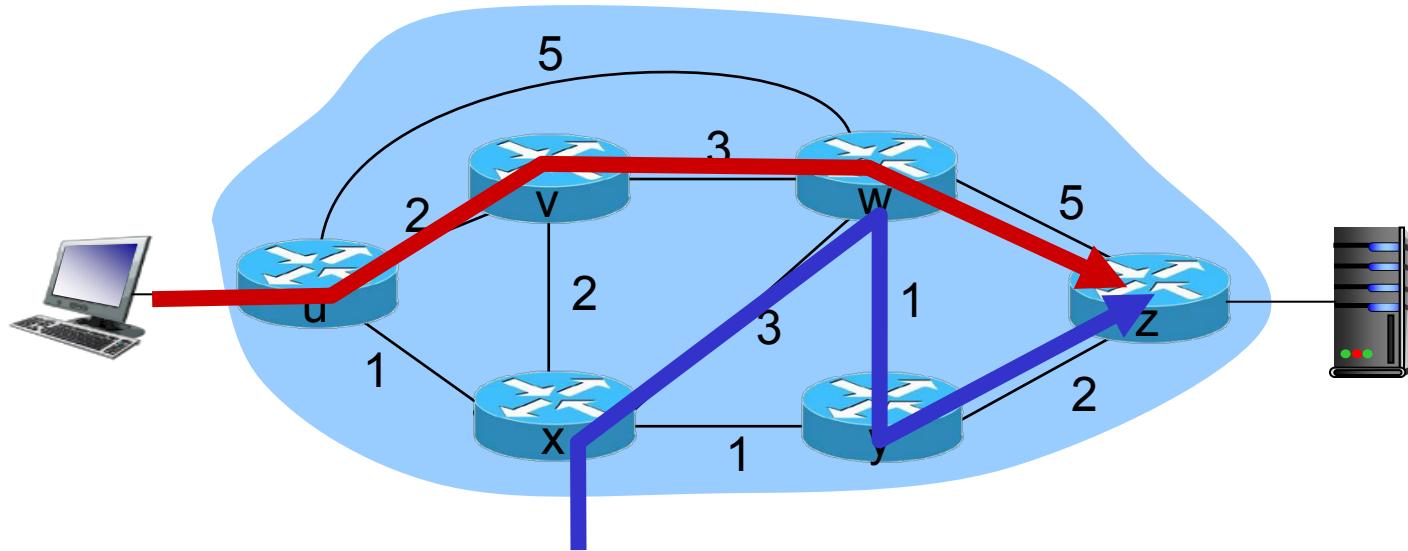
*Link weights are only control “knobs”: wrong!*

# Traffic engineering: difficult



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?  
A: can't do it (or need a new routing algorithm)

# Traffic engineering: difficult



Q: what if w wants to route blue and red traffic differently?

A: can't do it (with destination based forwarding, and LS, DV routing)

# Software defined networking (SDN)

4. programmable control applications

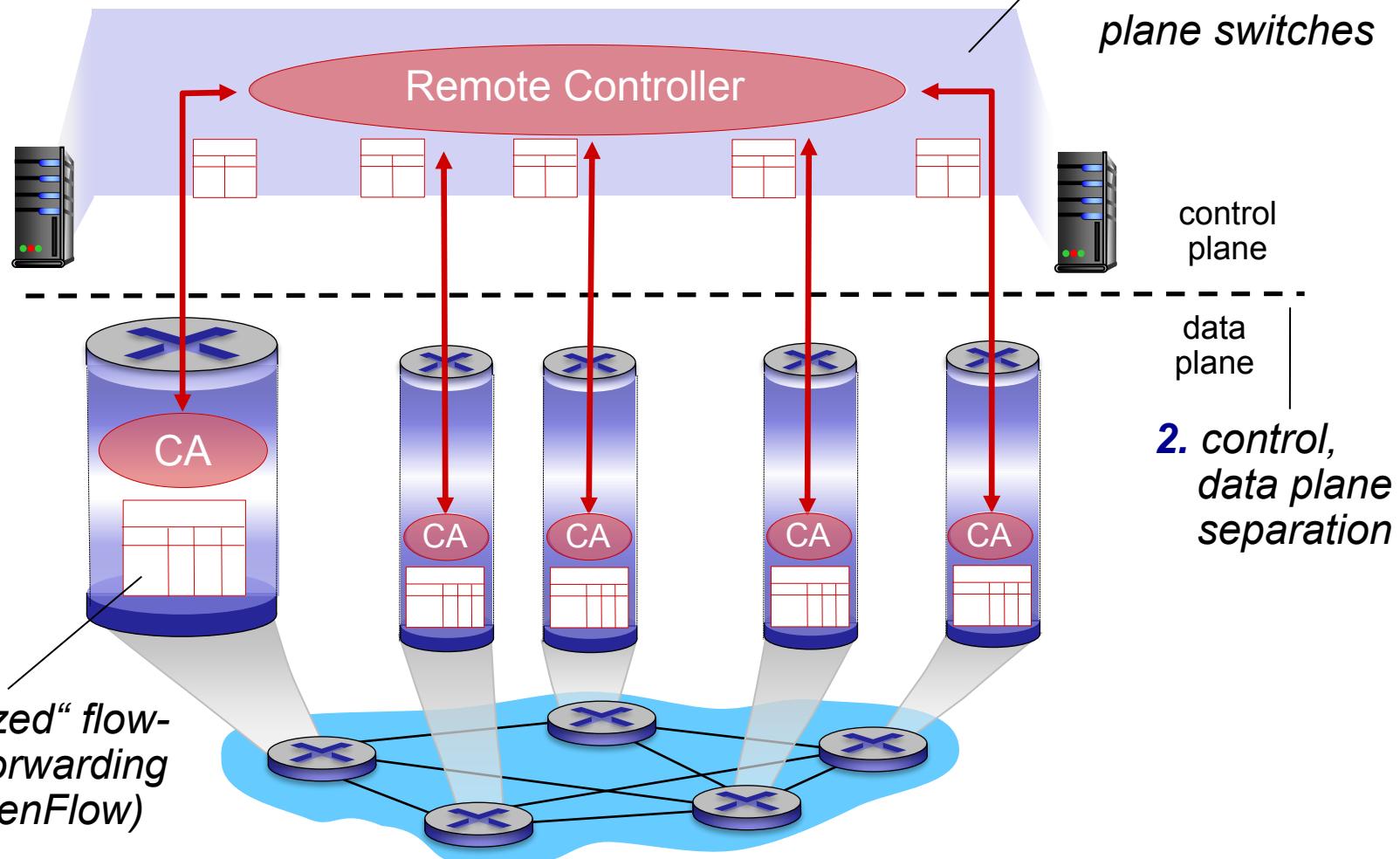
routing

access control

...

load balance

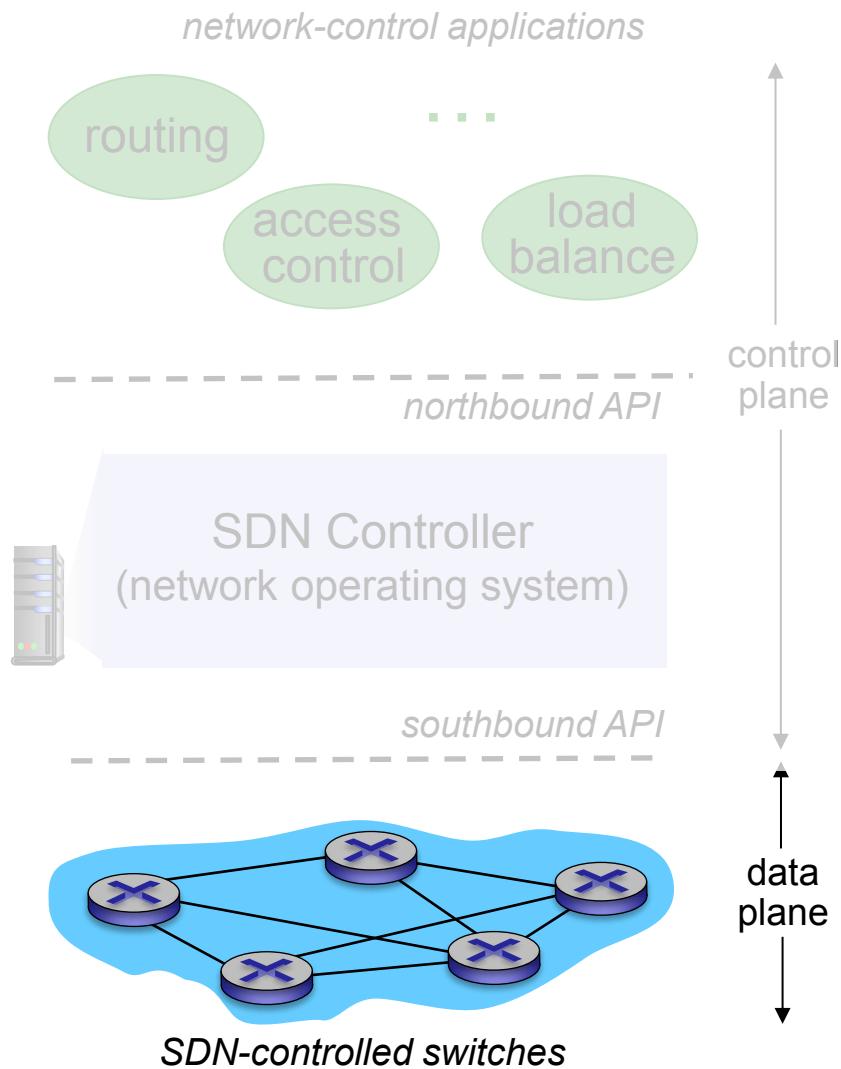
3. control plane functions external to data-plane switches



# SDN perspective: data plane switches

## *Data plane switches*

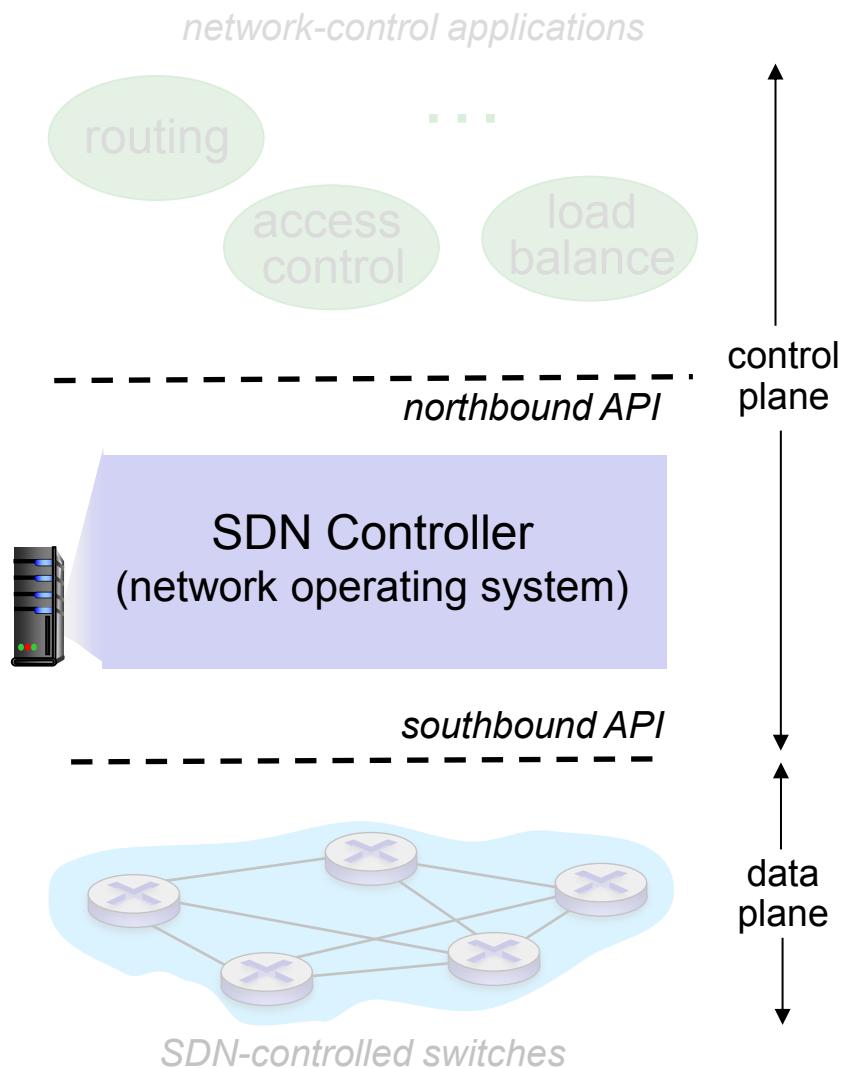
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)



# SDN perspective: SDN controller

## *SDN controller (network OS):*

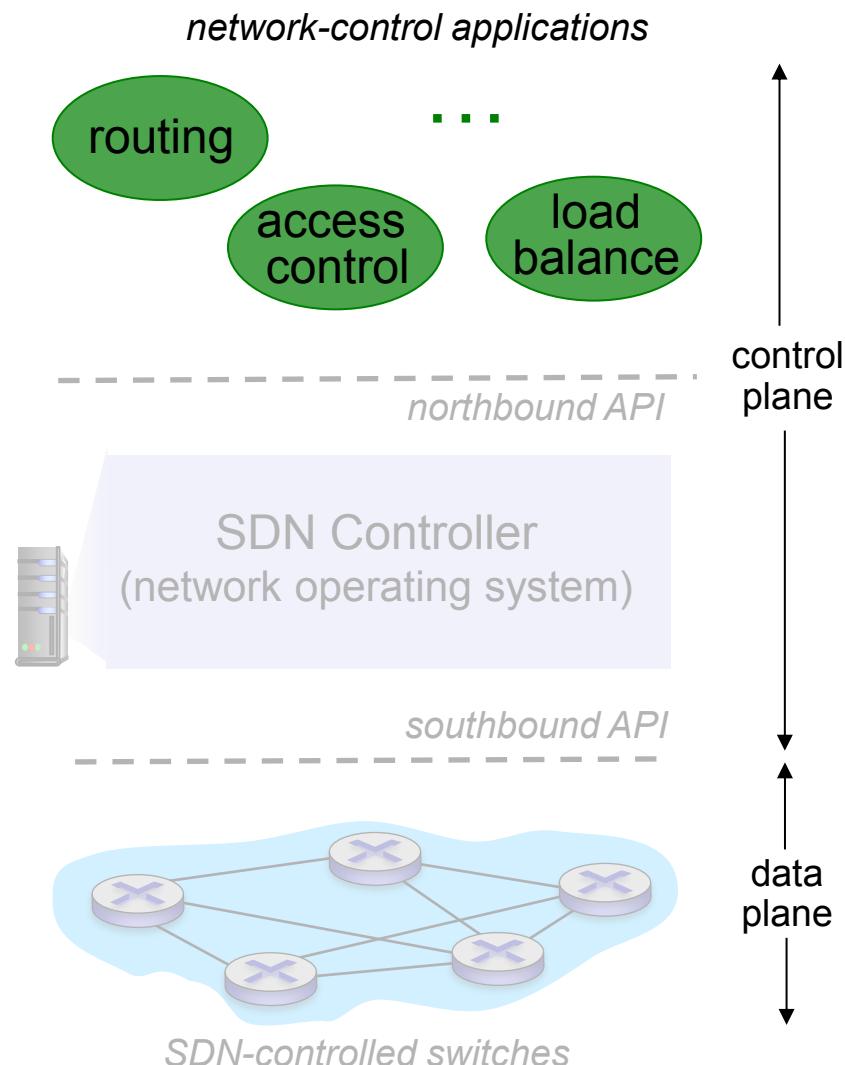
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



# SDN perspective: control applications

## *network-control apps:*

- “brains” of control:  
implement control functions  
using lower-level services, API  
provided by SDN controller
- *unbundled*: can be provided by  
3<sup>rd</sup> party: distinct from routing  
vendor, or SDN controller

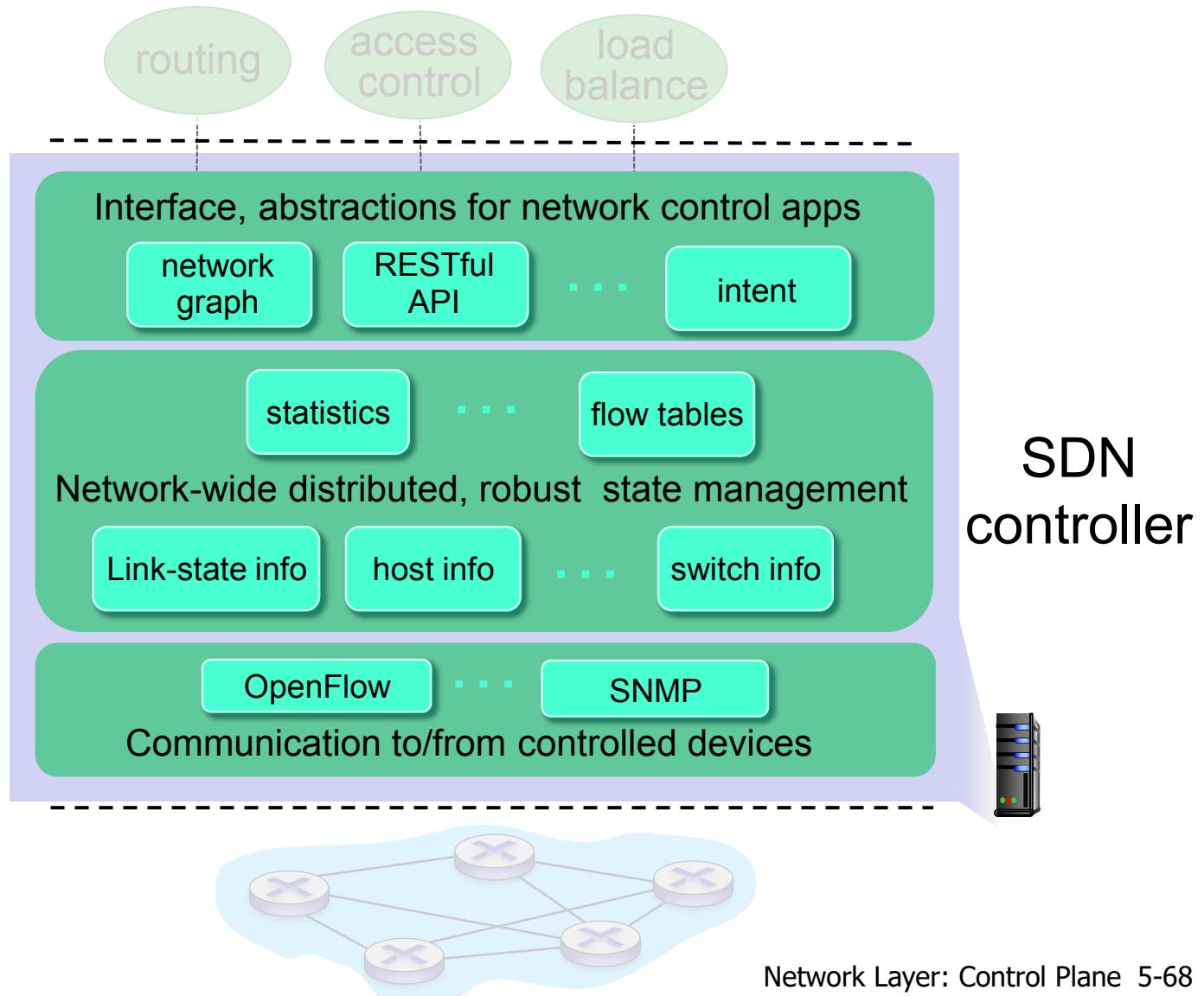


# Components of SDN controller

Interface layer to network control apps: abstractions API

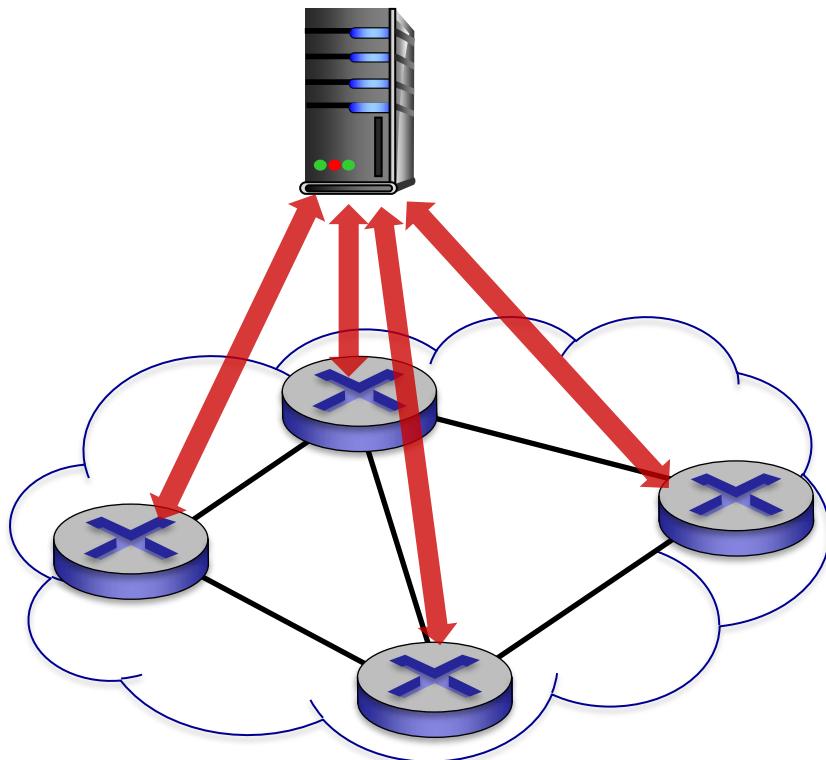
Network-wide state management layer: state of networks links, switches, services: a *distributed database*

*communication layer:* communicate between SDN controller and controlled switches



# OpenFlow protocol

OpenFlow Controller

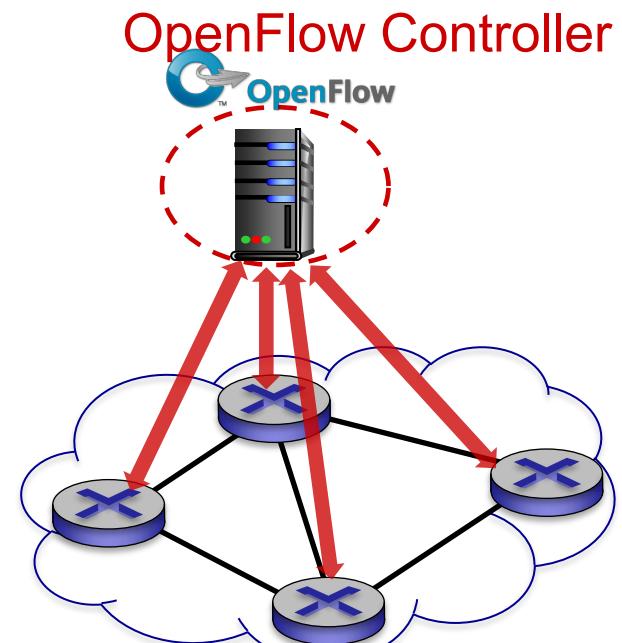


- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

# OpenFlow: controller-to-switch messages

## Key controller-to-switch messages

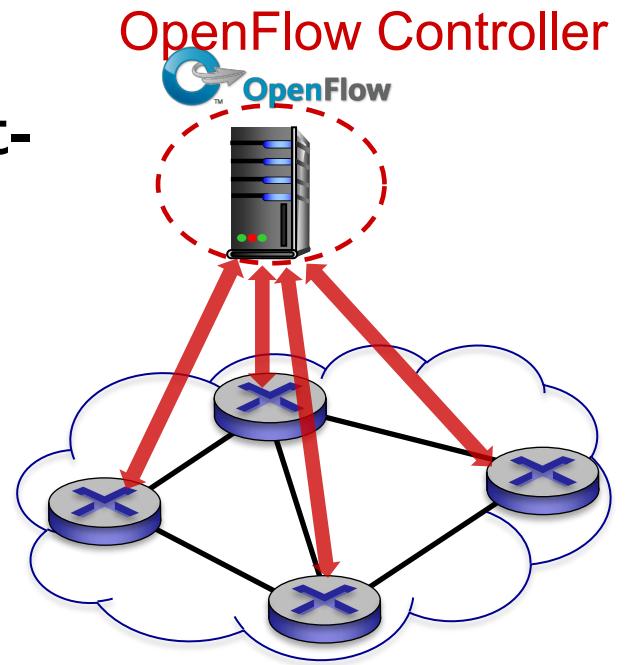
- **features**: controller queries switch features, switch replies
- **configure**: controller queries/sets switch configuration parameters
- **modify-state**: add, delete, modify flow entries in the OpenFlow tables
- **packet-out**: controller can send this packet out of specific switch port



# OpenFlow: switch-to-controller messages

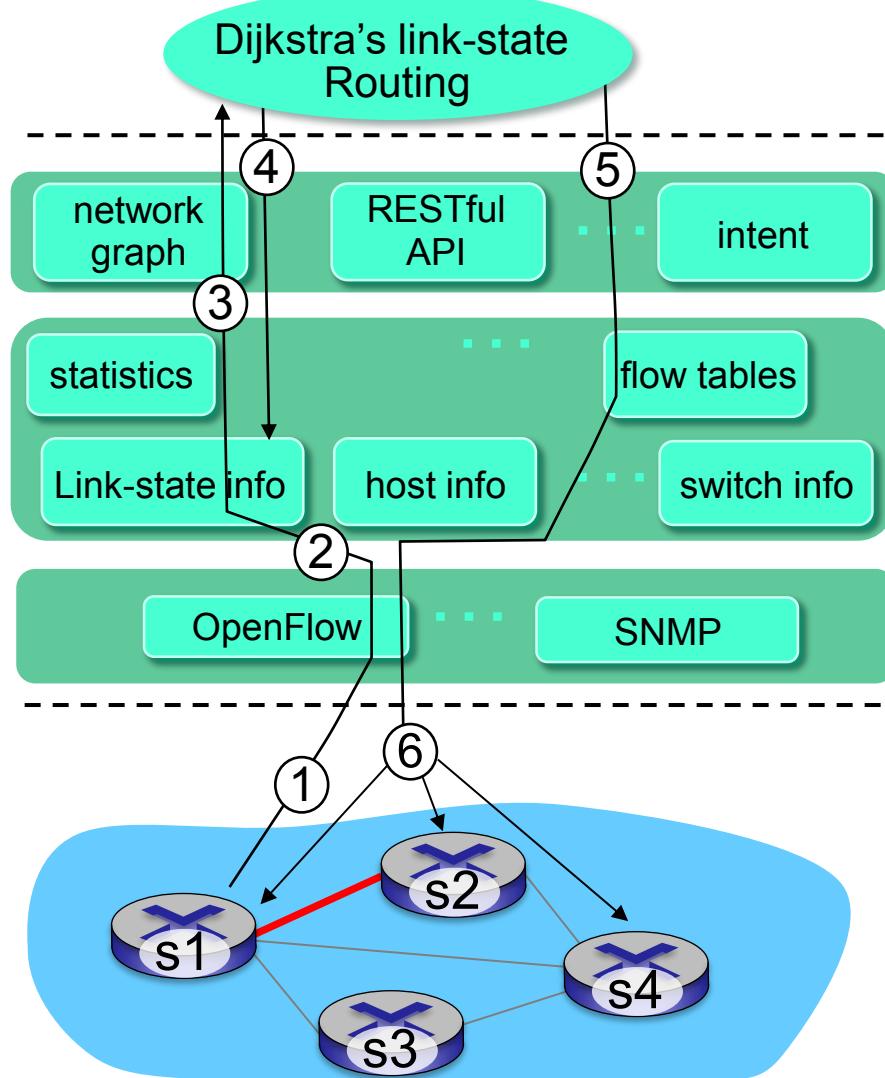
## *Key switch-to-controller messages*

- **packet-in:** transfer packet (and its control) to controller. See packet-out message from controller
- **flow-removed:** flow table entry deleted at switch
- **port status:** inform controller of a change on a port.



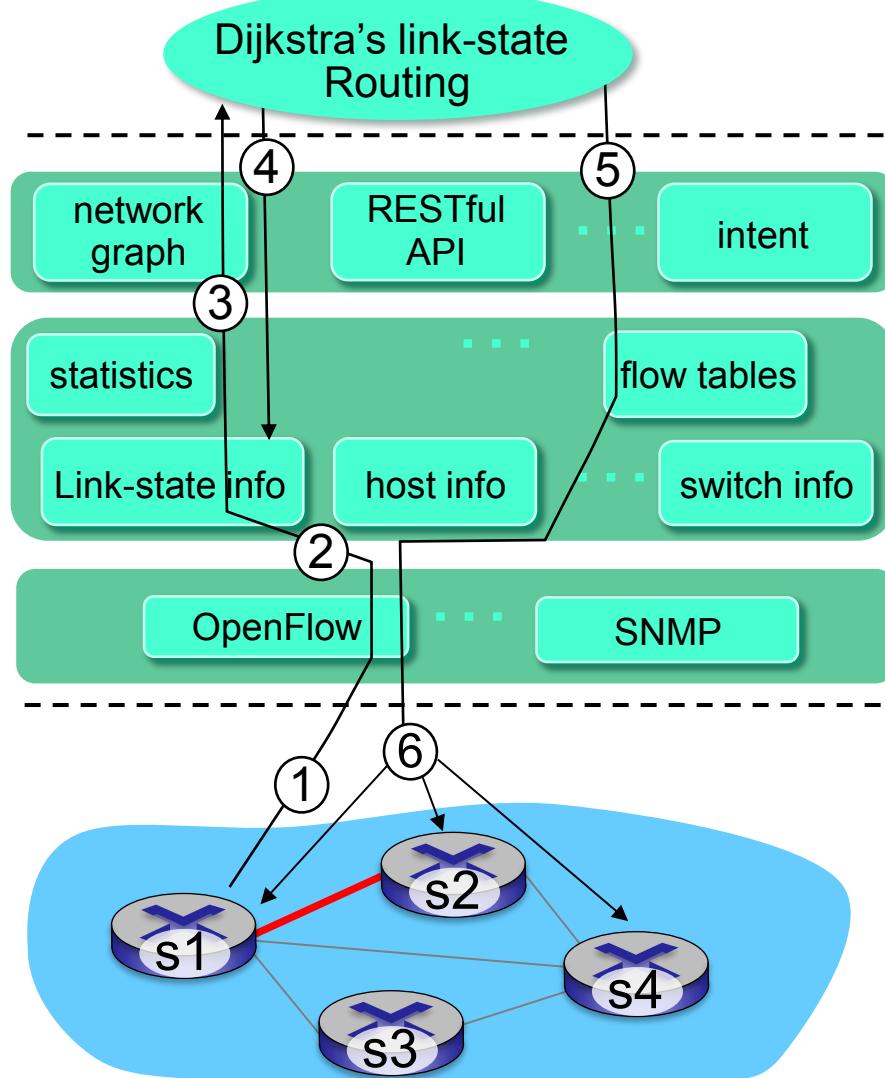
Fortunately, network operators don't “program” switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

# SDN: control/data plane interaction example



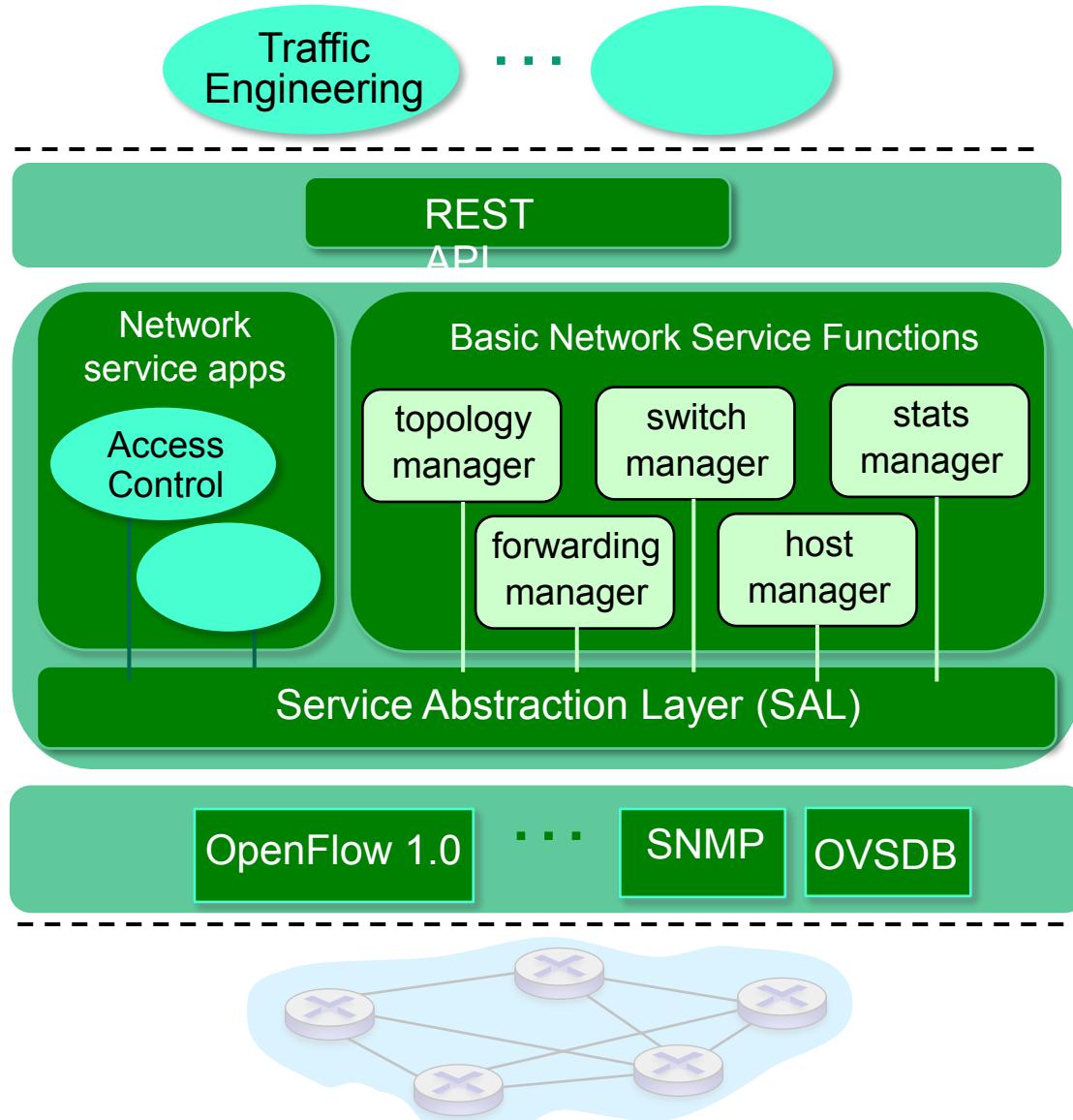
- ① SI, experiencing link failure using OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It is called.
- ④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

# SDN: control/data plane interaction example



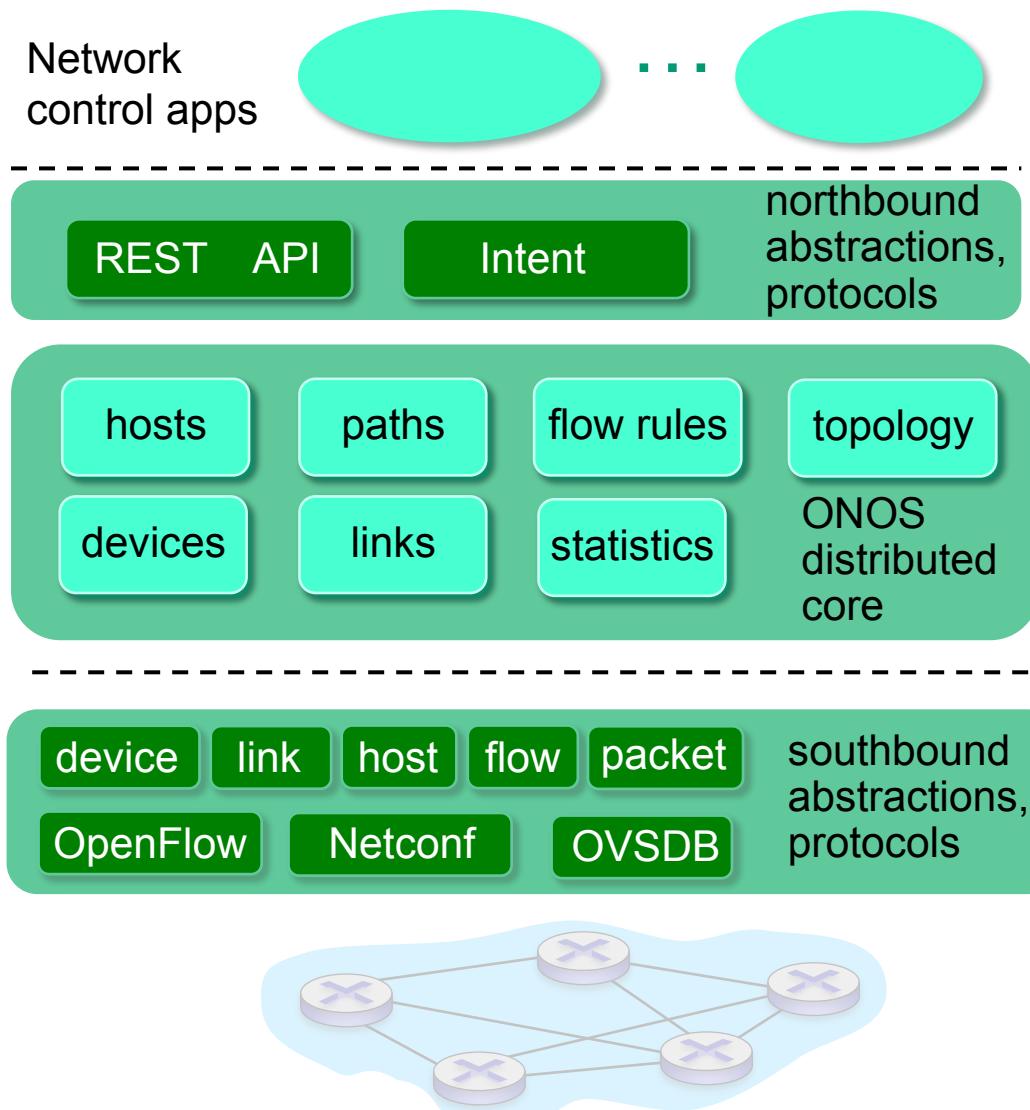
- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ Controller uses OpenFlow to install new tables in switches that need updating

# OpenDaylight (ODL) controller



- ODL Lithium controller
- network apps may be contained within, or be external to SDN controller
- Service Abstraction Layer: interconnects internal, external applications and services

# ONOS controller



- **control apps separate from controller**
- **intent framework:** high-level specification of service: what rather than how
- **considerable emphasis on distributed core:** service reliability, replication performance scaling

# SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: “baked in” from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state

- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# ICMP: internet control message protocol

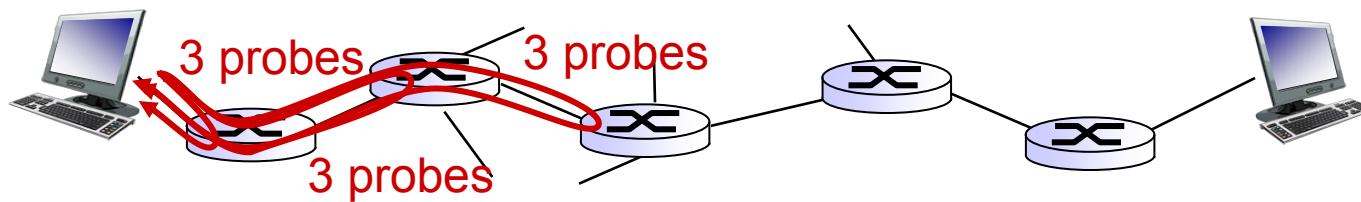
---

- 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)
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 destination
    - first set has TTL = 1
    - second set has TTL=2, etc.
    - unlikely port number
  - when datagram in  $n$ th set arrives to  $n$ th router:
    - router discards datagram and sends source ICMP message (type 11, code 0)
    - ICMP message include name of router & IP address
  - when ICMP message 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



# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state

- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs:  
BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# What is network management?

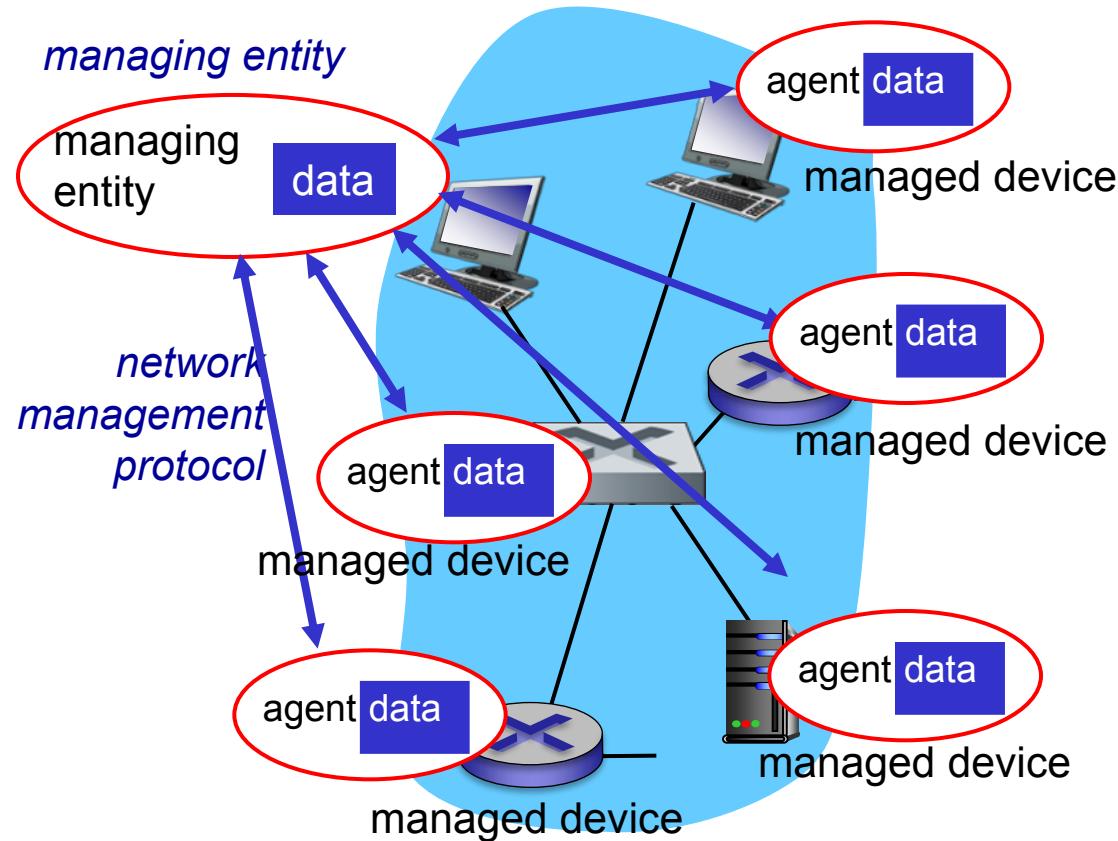
- autonomous systems (aka “network”): 1000s of interacting hardware/software components
- other complex systems requiring monitoring, control:
  - jet airplane
  - nuclear power plant
  - others?



"Network management includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

# Infrastructure for network management

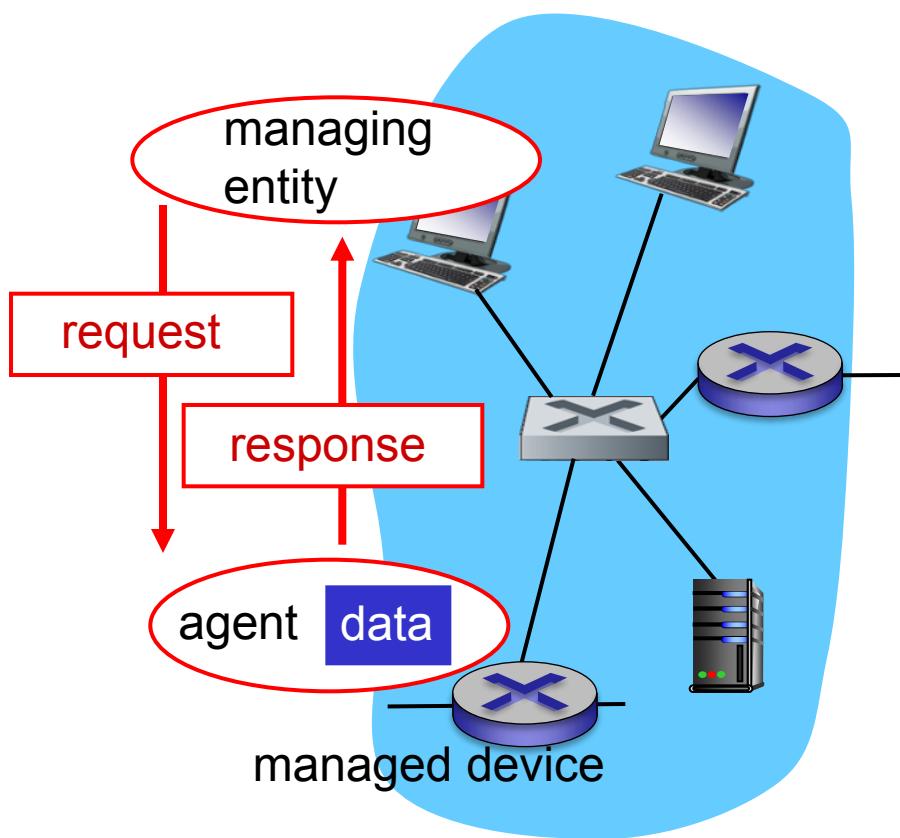
definitions:



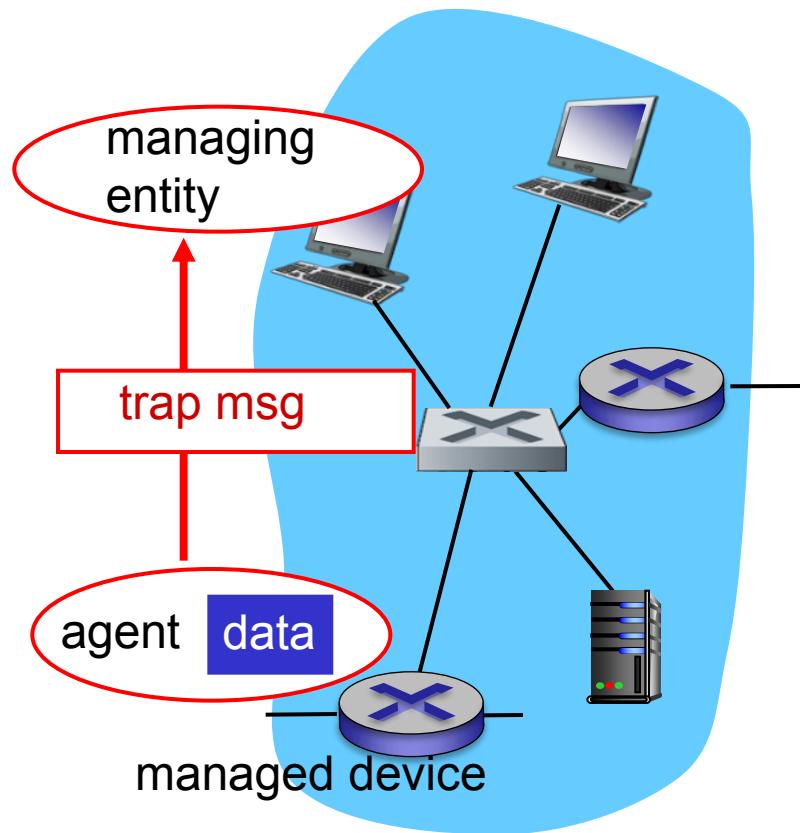
*managed devices* contain *managed objects* whose data is gathered into a *Management Information Base (MIB)*

# SNMP protocol

Two ways to convey MIB info, commands:



request/response mode

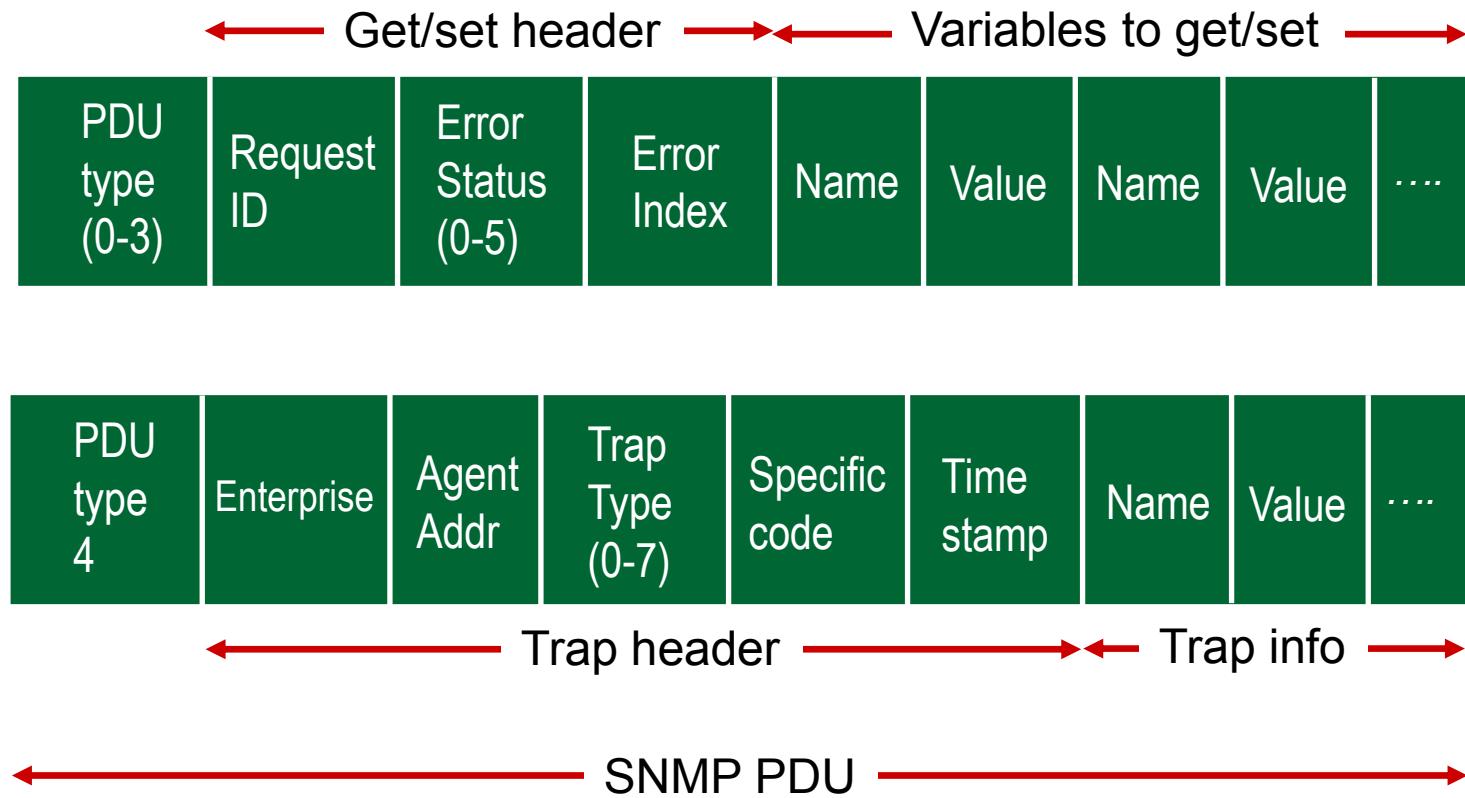


trap mode

# SNMP protocol: message types

<u>Message type</u>	<u>Function</u>
GetRequest GetNextRequest GetBulkRequest	manager-to-agent: “get me data” (data instance, next data in list, block of data)
InformRequest	manager-to-manager: here’s MIB value
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

# SNMP protocol: message formats



*More on network management: see earlier editions of text!*

# Chapter 5: summary

*we've learned a lot!*

- approaches to network control plane
  - per-router control (traditional)
  - logically centralized control (software defined networking)
- traditional routing algorithms
  - implementation in Internet: OSPF, BGP
- SDN controllers
  - implementation in practice: ODL, ONOS
- Internet Control Message Protocol
- network management

*next stop: link layer!*