

# Computer Networks and Applications

COMP 3331/COMP 9331

Week 1

## Introduction to Computer Networks

Reading Guide: Chapter 1, Sections 1.1 - 1.4

# Acknowledgment

- ❖ Majority of lecture slides are from the author's lecture slide set
  - Enhancements + *additional material*

# I. Introduction

## *Goals:*

- ❖ get “feel” and terminology
- ❖ defer depth and detail to *later* in course
- ❖ understand concepts using the Internet as example

# I. Introduction: 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

### I.6 networks under attack: security

### I.7 history

Hobbe's Internet Timeline - <http://www.zakon.org/robert/internet/timeline/>

# Quiz: What is the Internet?

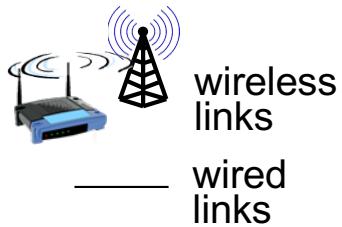


- A. One single homogenous network
- B. An interconnection of different computer networks
- C. An infrastructure that provides services to networked applications
- D. Something else (answer in comments on Zeeting)

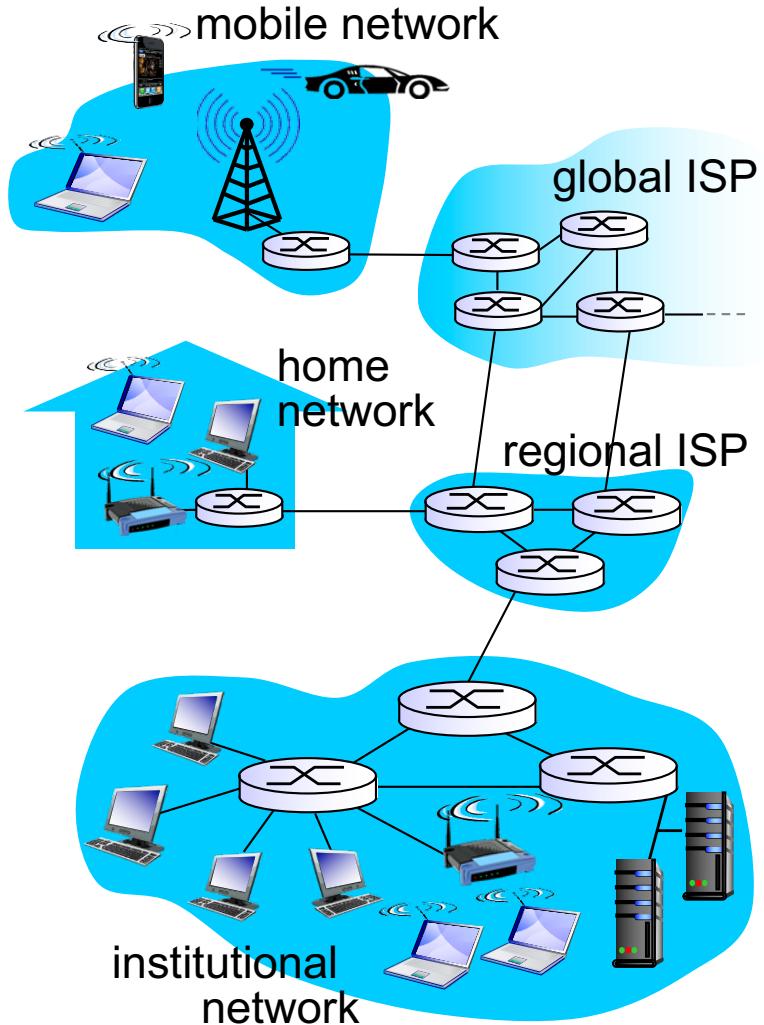
**Answers: B and C are valid answers**

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)

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



- ❖ millions of connected computing devices:
  - *hosts = end systems*
  - running *network apps*
- ❖ *communication links*
  - fiber, copper, radio, satellite
  - transmission rate: *bandwidth*
- ❖ *Packet switches: forward packets (chunks of data)*
  - *routers and link layer switches*



# “Fun” Internet appliances



Internet  
refrigerator



Picture frame



Web-enabled toaster +  
weather forecaster



car



Networked TV Set top Boxes



sensorized,  
bed  
mattress



pacemaker

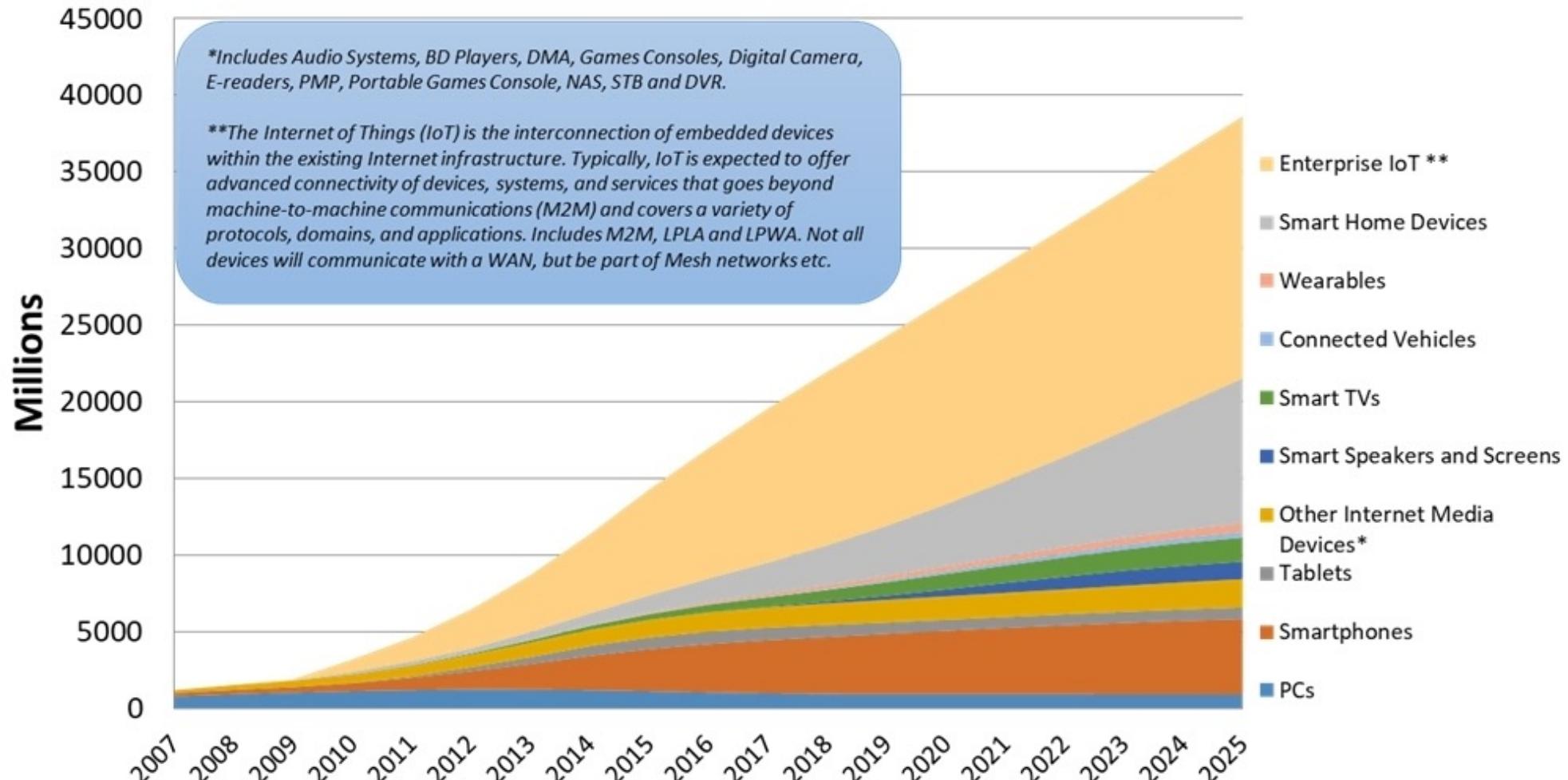


Tweet-a-watt:  
monitor energy use



Smart Lightbulbs

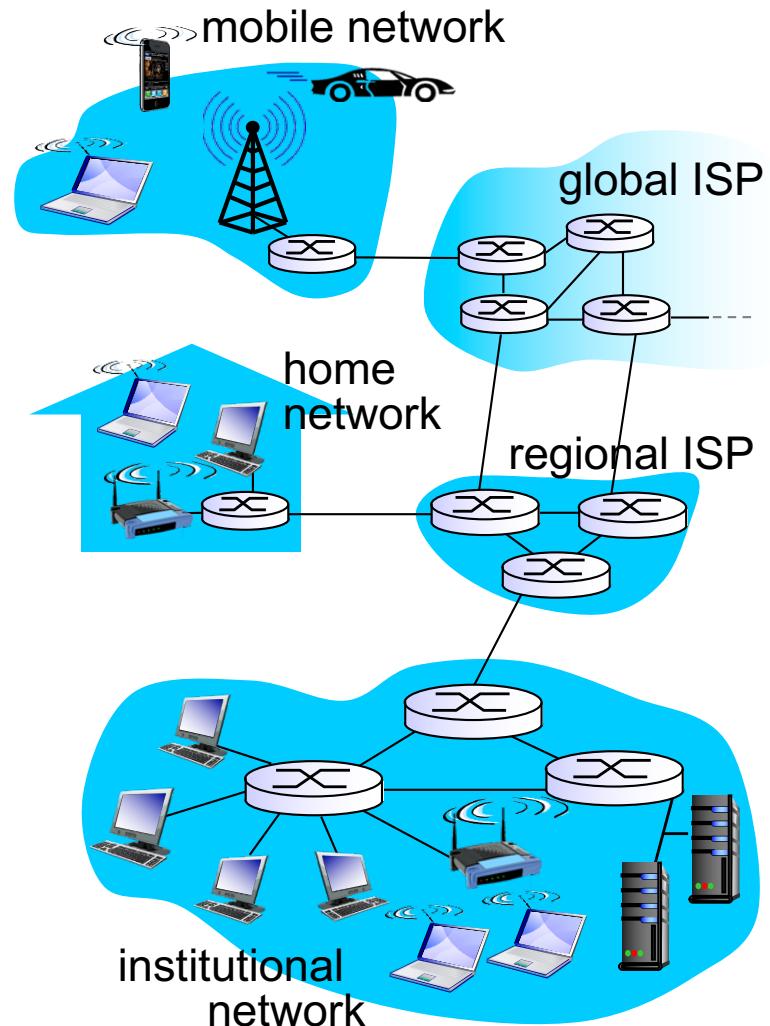
## Global Connected and IoT Device Installed Base Forecast



Source – Strategy Analytics research services, May 2019: IoT Strategies, Connected Home Devices, Connected Computing Devices, Wireless Smartphone Strategies, Wearable Device Ecosystem, Smart Home Strategies

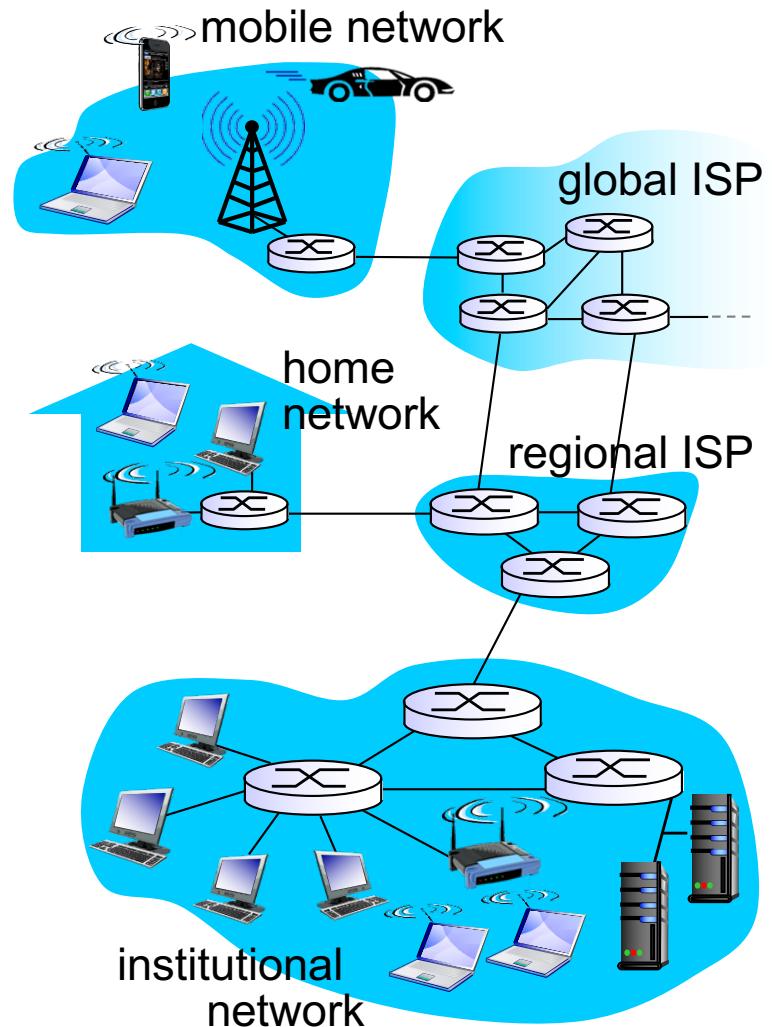
# What's the Internet: “nuts and bolts” view

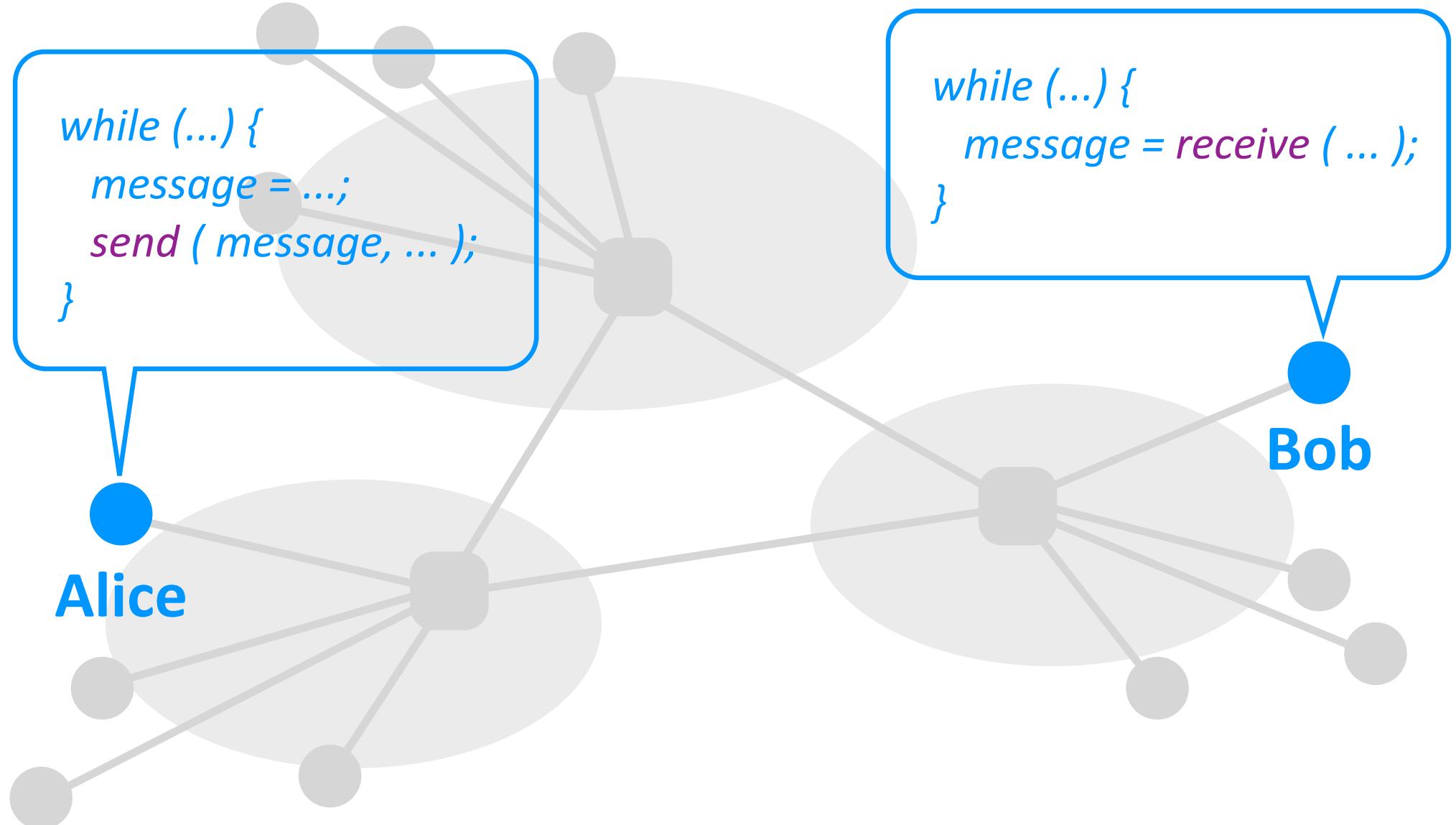
- ❖ *Internet: “network of networks”*
  - Interconnected ISPs
- ❖ *protocols* control sending, receiving of msgs
  - e.g., TCP, IP, HTTP, Skype, 802.11
- ❖ *Internet standards*
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force



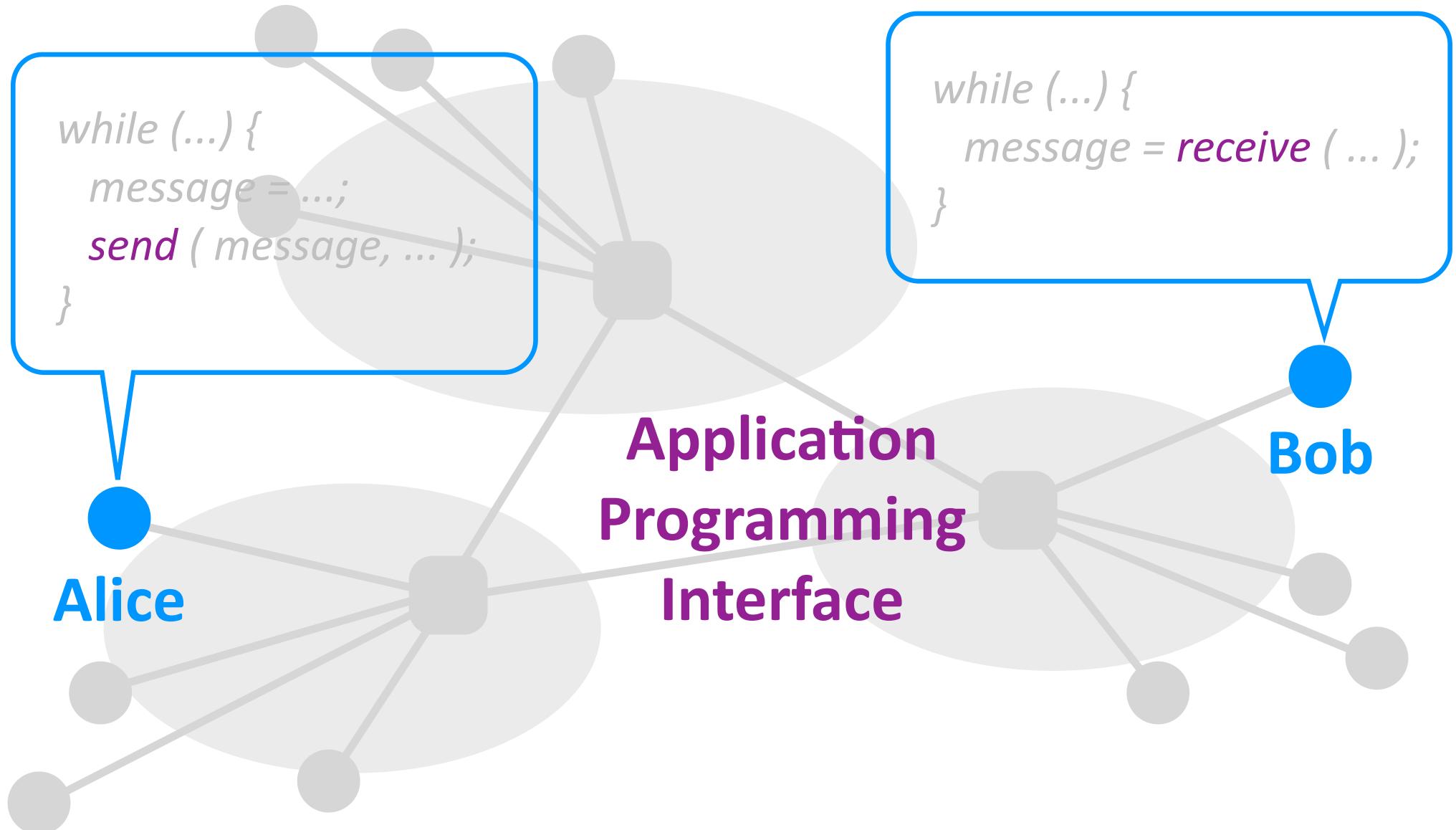
# What's the Internet: a service view

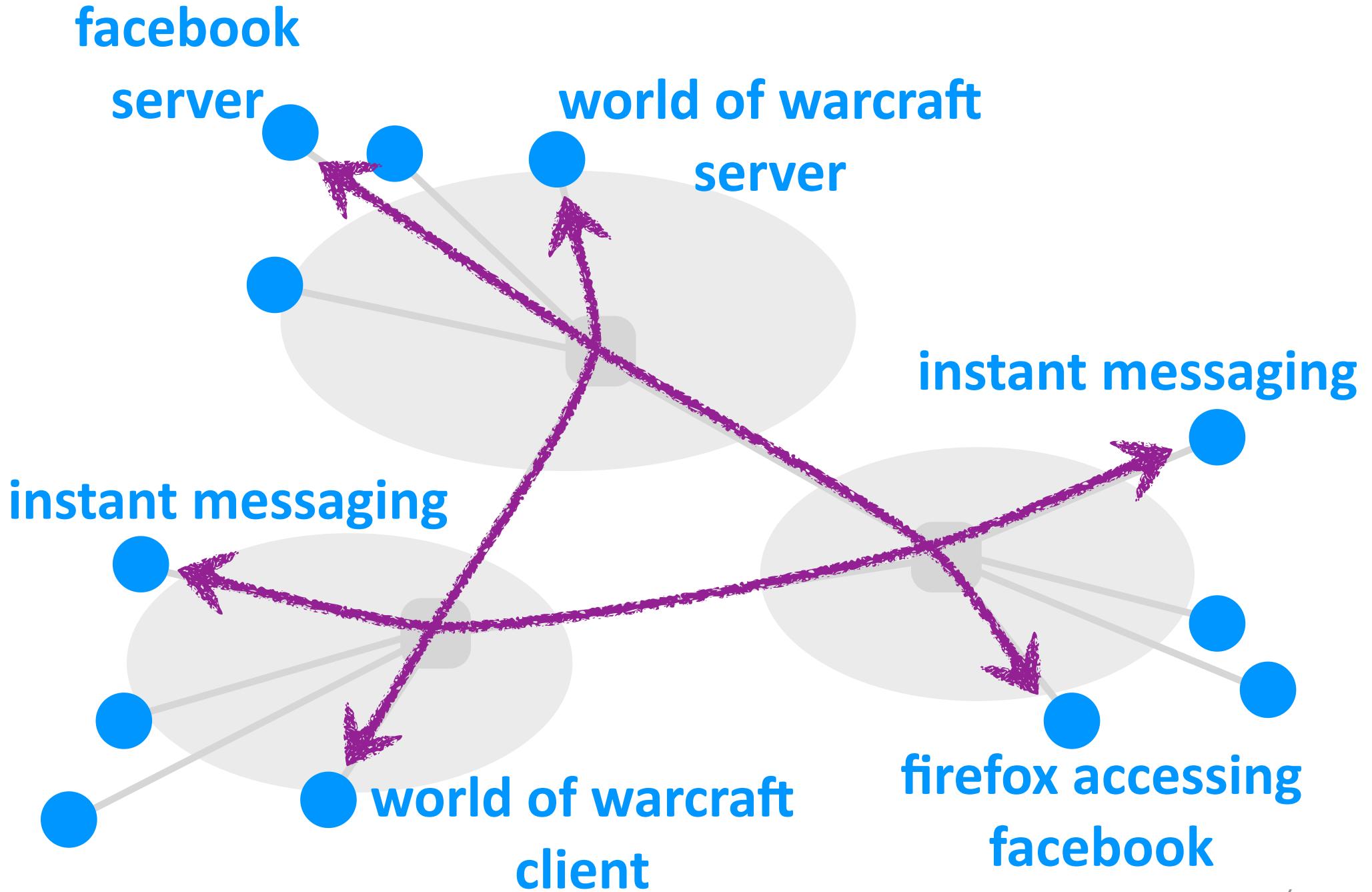
- ❖ *Infrastructure that provides services to applications:*
  - Web, VoIP, email, games, e-commerce, social nets, ...
- ❖ *provides programming interface to apps*
  - hooks that allow sending and receiving app programs to “connect” to Internet
  - provides service options, analogous to postal service





# Application Programming Interface





# What's a protocol?

## *human protocols:*

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

... specific msgs sent

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

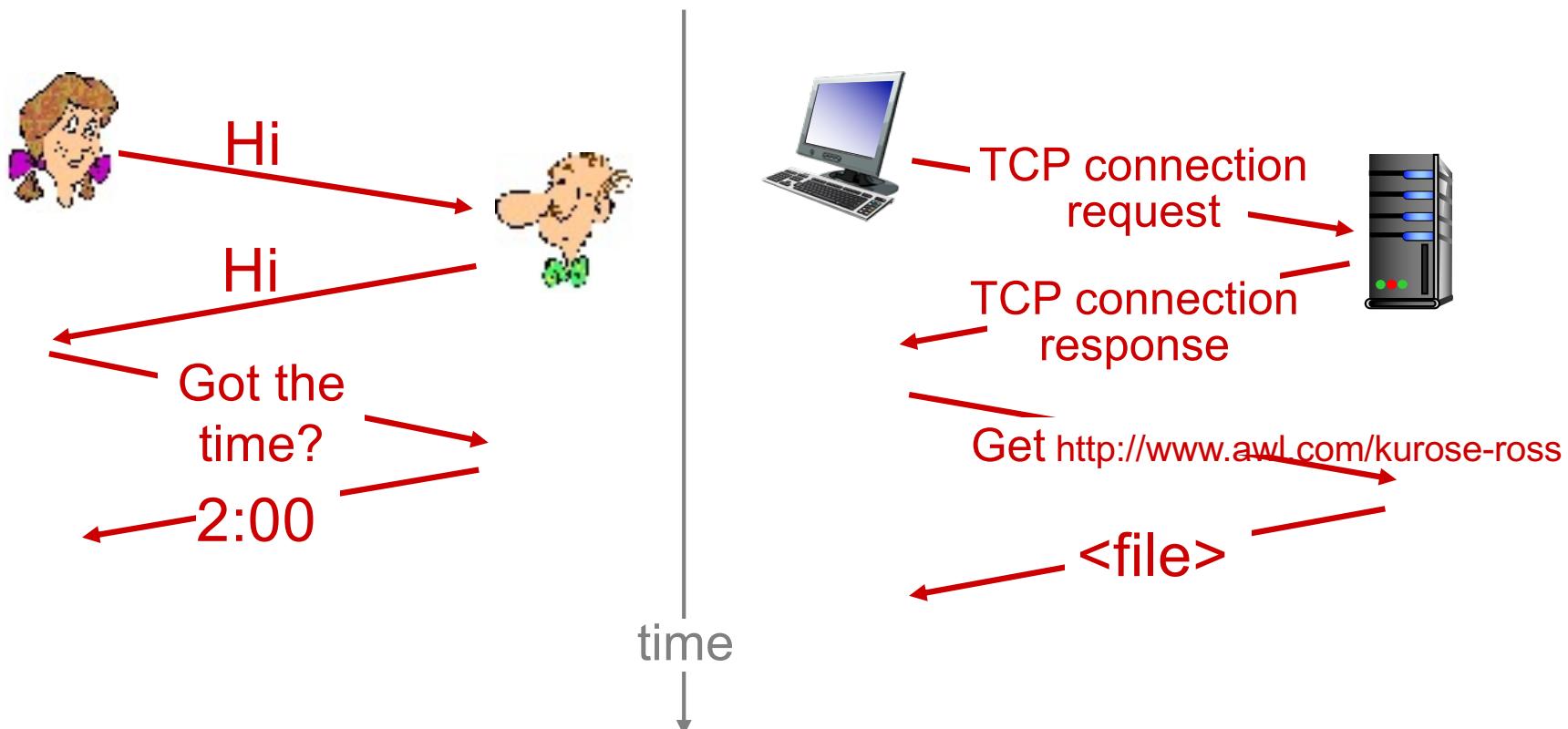
## *network protocols:*

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

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

# What's a protocol?

a human protocol and a computer network protocol:



Q: other human protocols?



## Quiz: Internet of Things

How many Internet-connected devices do you have in your home (include your computers, phones, tablets)?

- A. Less than 10
- B. Between 10 to 20
- C. Between 20 to 50
- D. Between 50 to 100
- E. More than 100

Open a browser and type: **[www.zeeplings.com/salil](http://www.zeeplings.com/salil)**

# I. Introduction: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

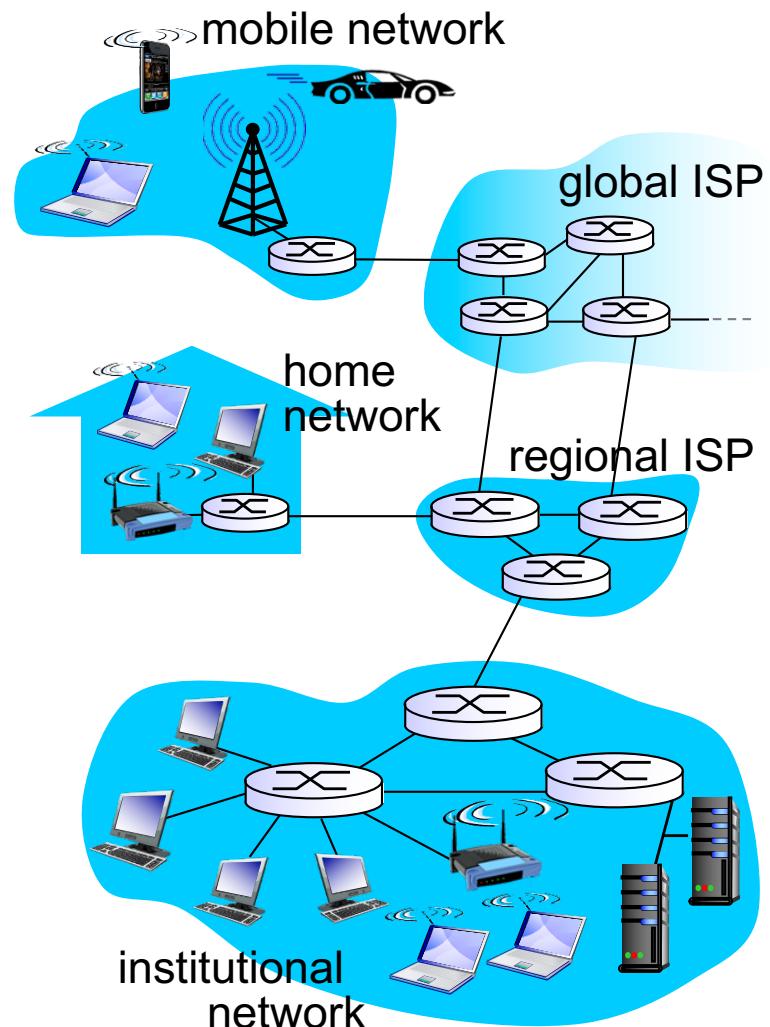
I.5 protocol layers, service models

I.6 networks under attack: security

I.7 history

# A closer look at network structure:

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



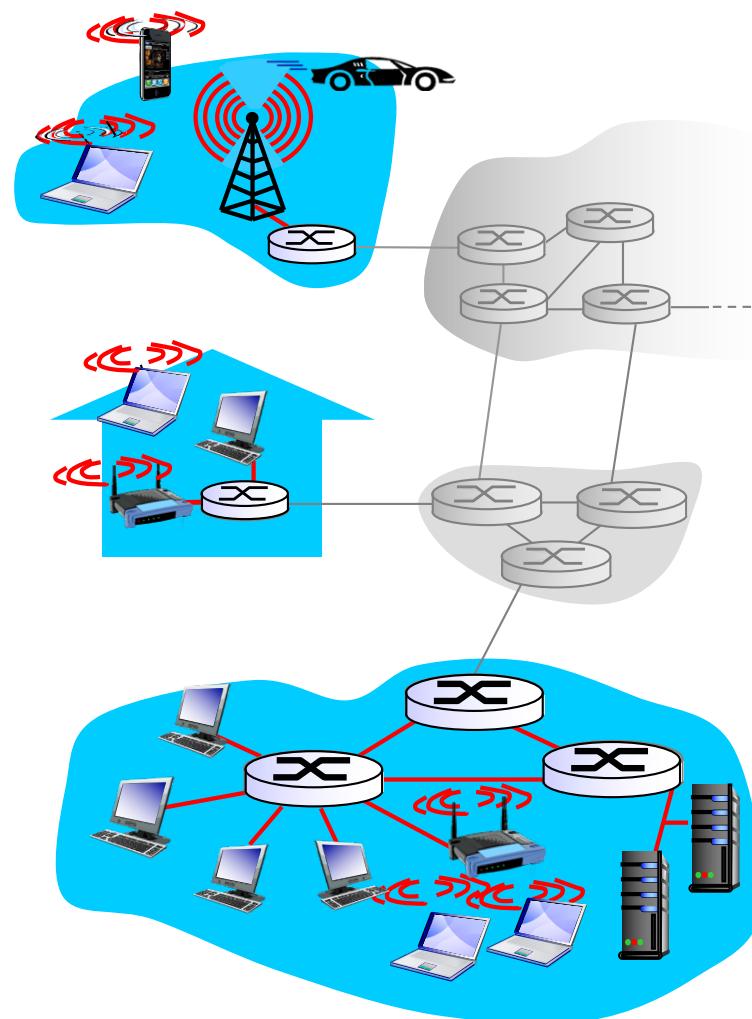
# Access networks and physical media

*Q: How to connect end systems to edge router?*

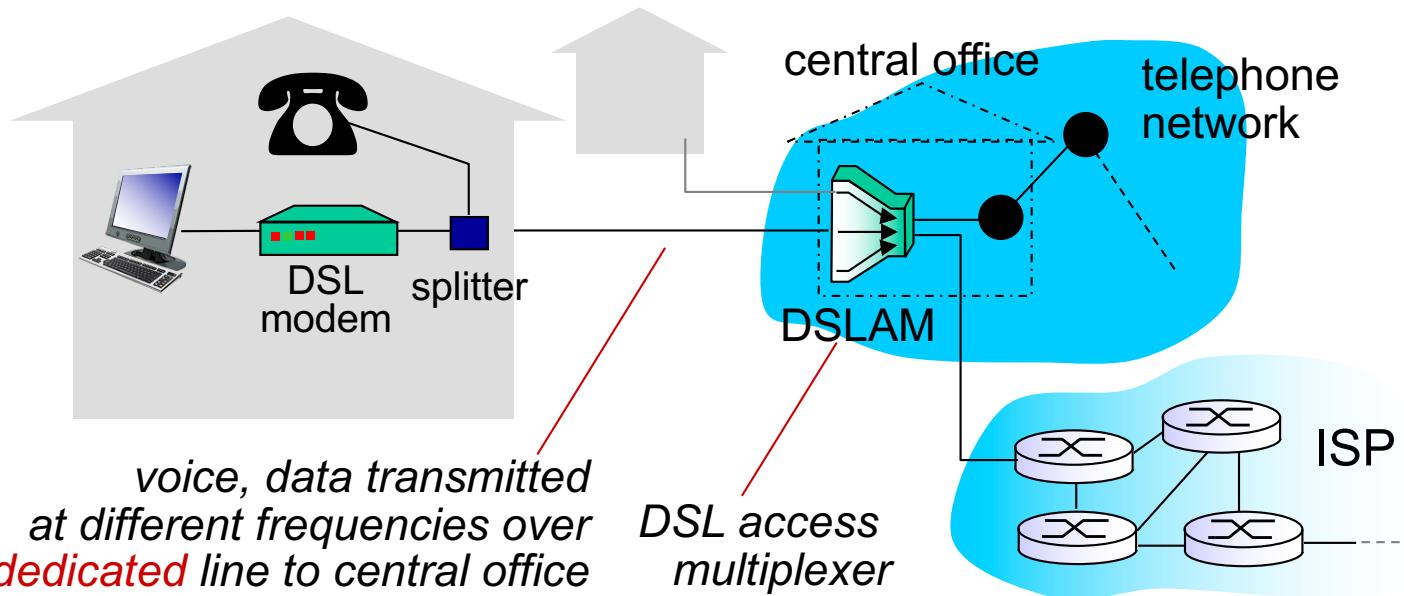
- ❖ residential access nets
- ❖ institutional access networks (school, company)
- ❖ mobile access networks

*keep in mind:*

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

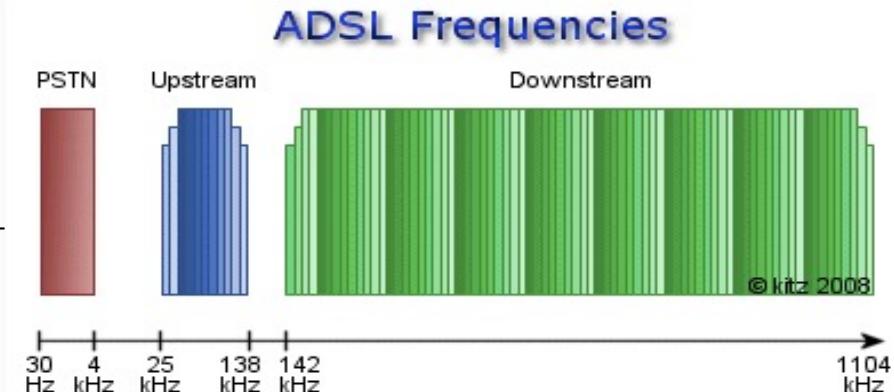
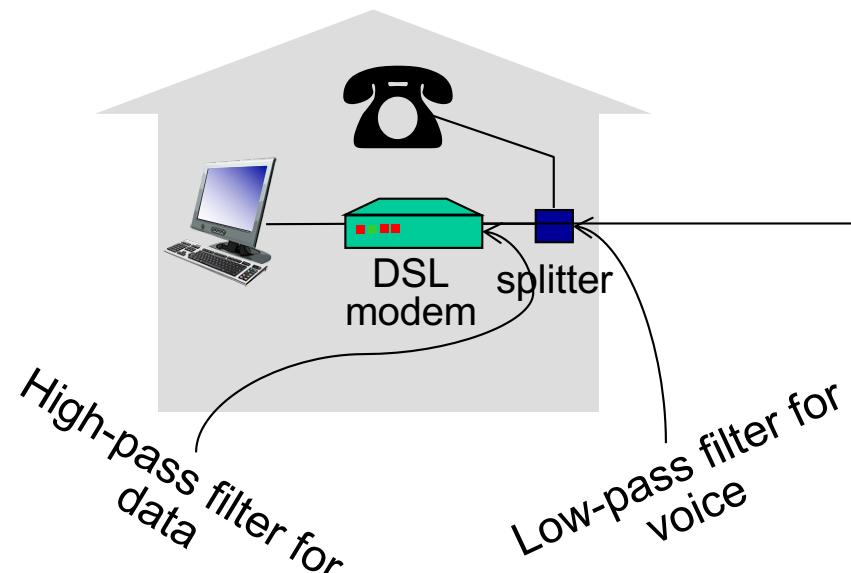


# Access net: digital subscriber line (DSL)



- ❖ use **existing** telephone line to central office DSLAM
  - data over DSL phone line goes to Internet
  - voice over DSL phone line goes to telephone net

# Access net: digital subscriber line (DSL)

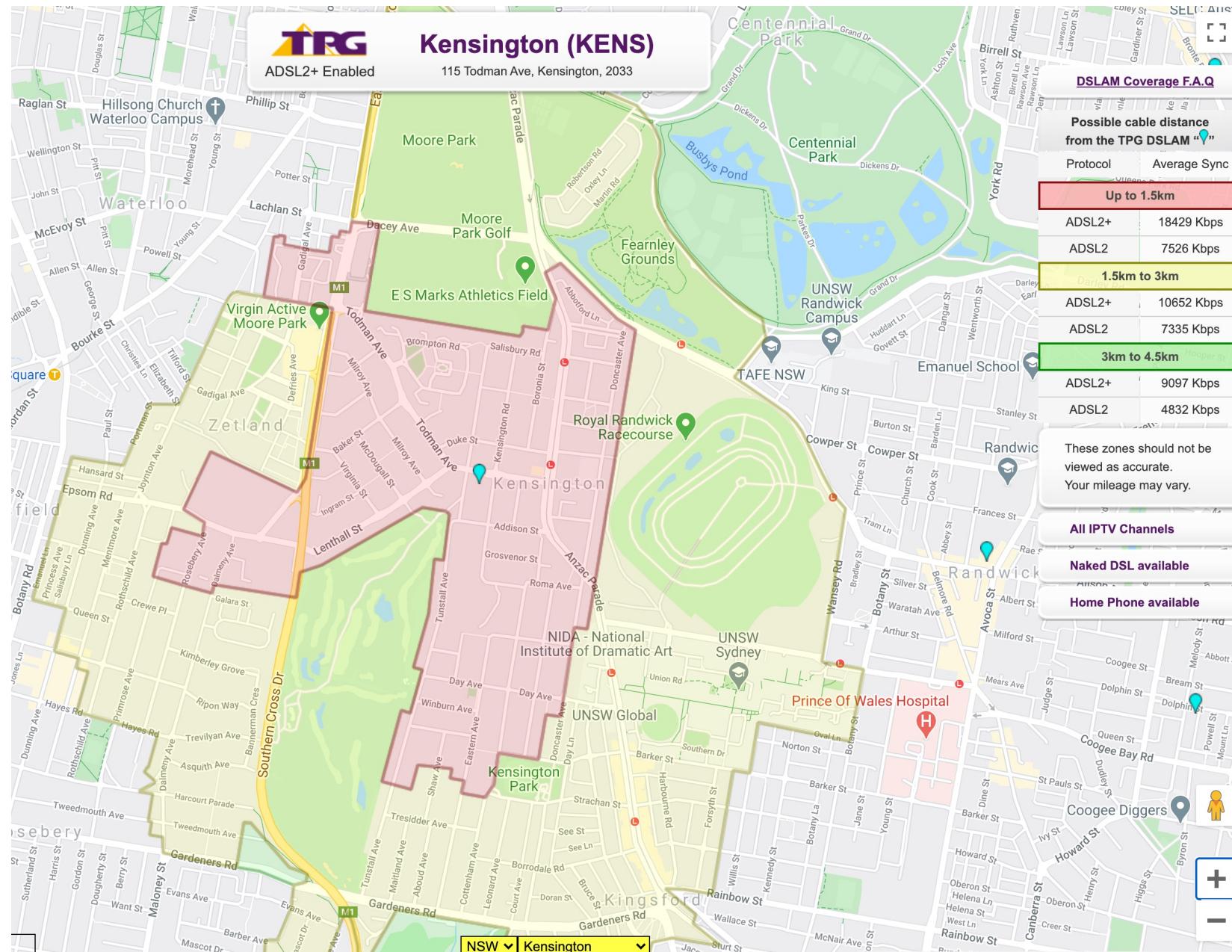


ADSL over POTS

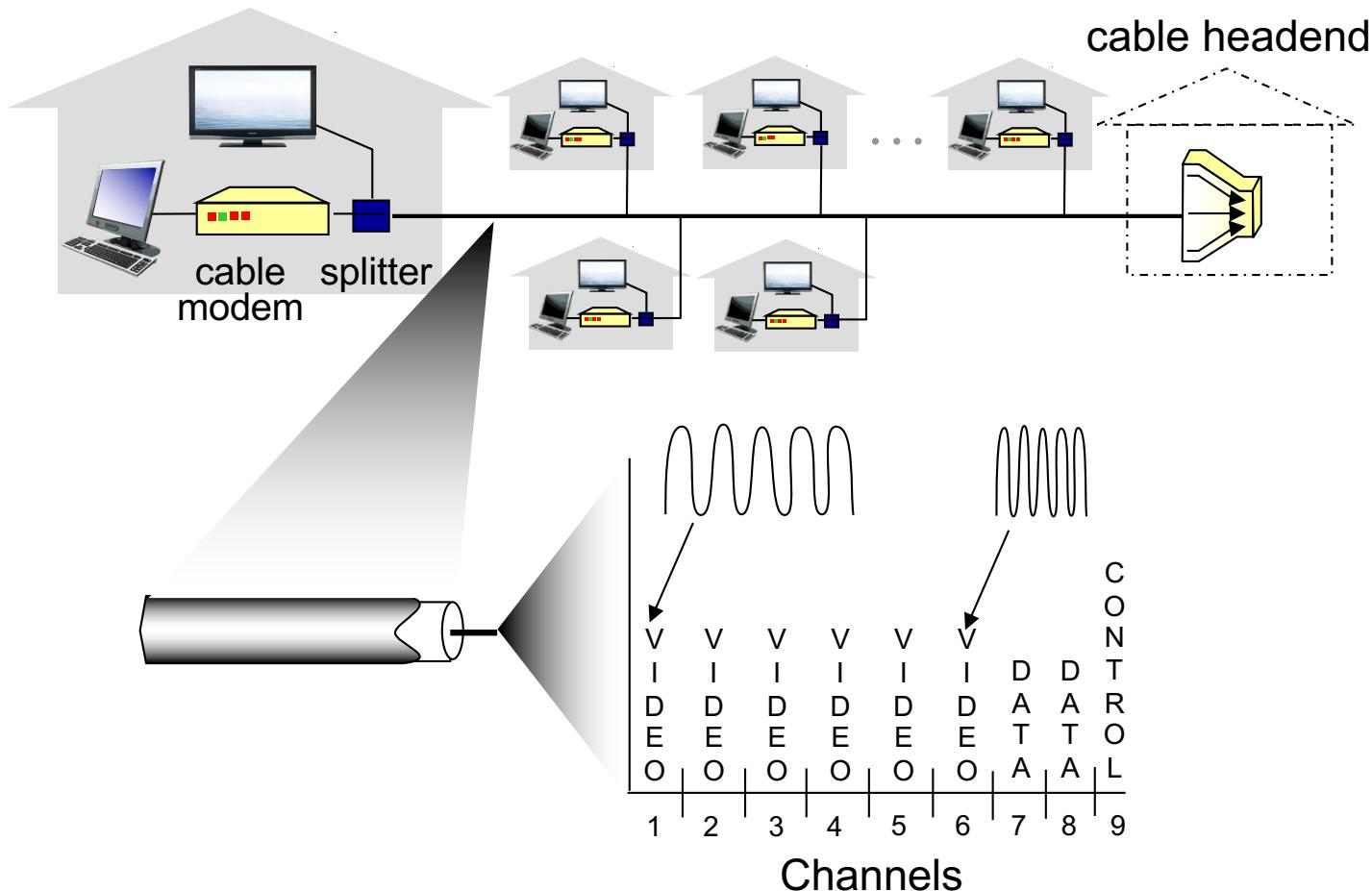
*voice, data transmitted  
at different frequencies over  
dedicated line to central office*

- Different data rates for upload and download (ADSL)
  - < 2.5 Mbps upstream transmission rate (typically < 1 Mbps)
  - < 24 Mbps downstream transmission rate (typically < 10 Mbps)

# Access net: digital subscriber line (DSL)

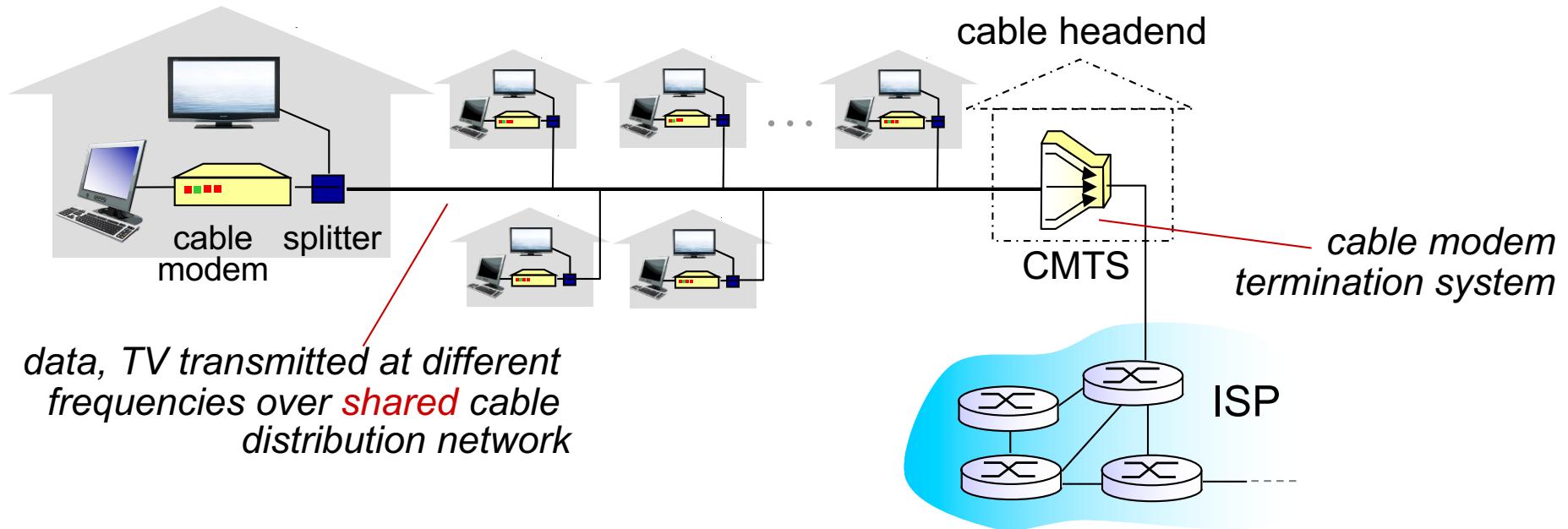


# Access net: cable network



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

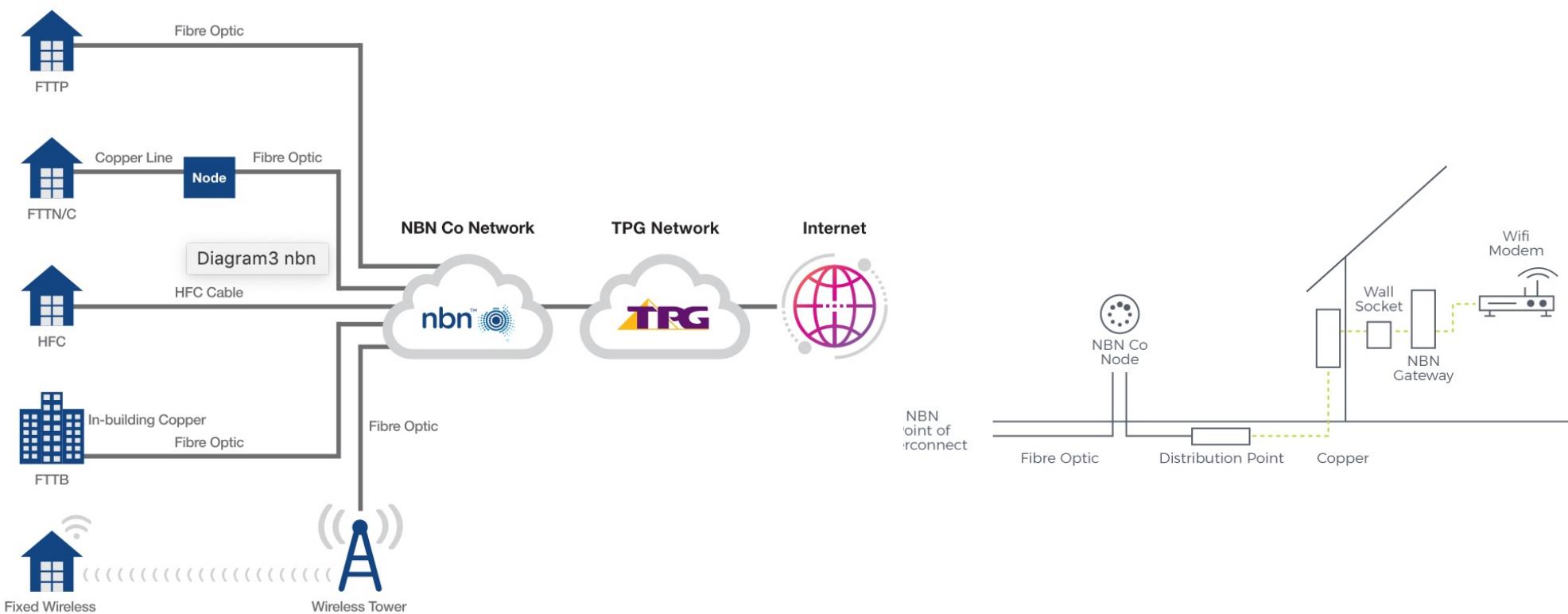
# Access net: cable network



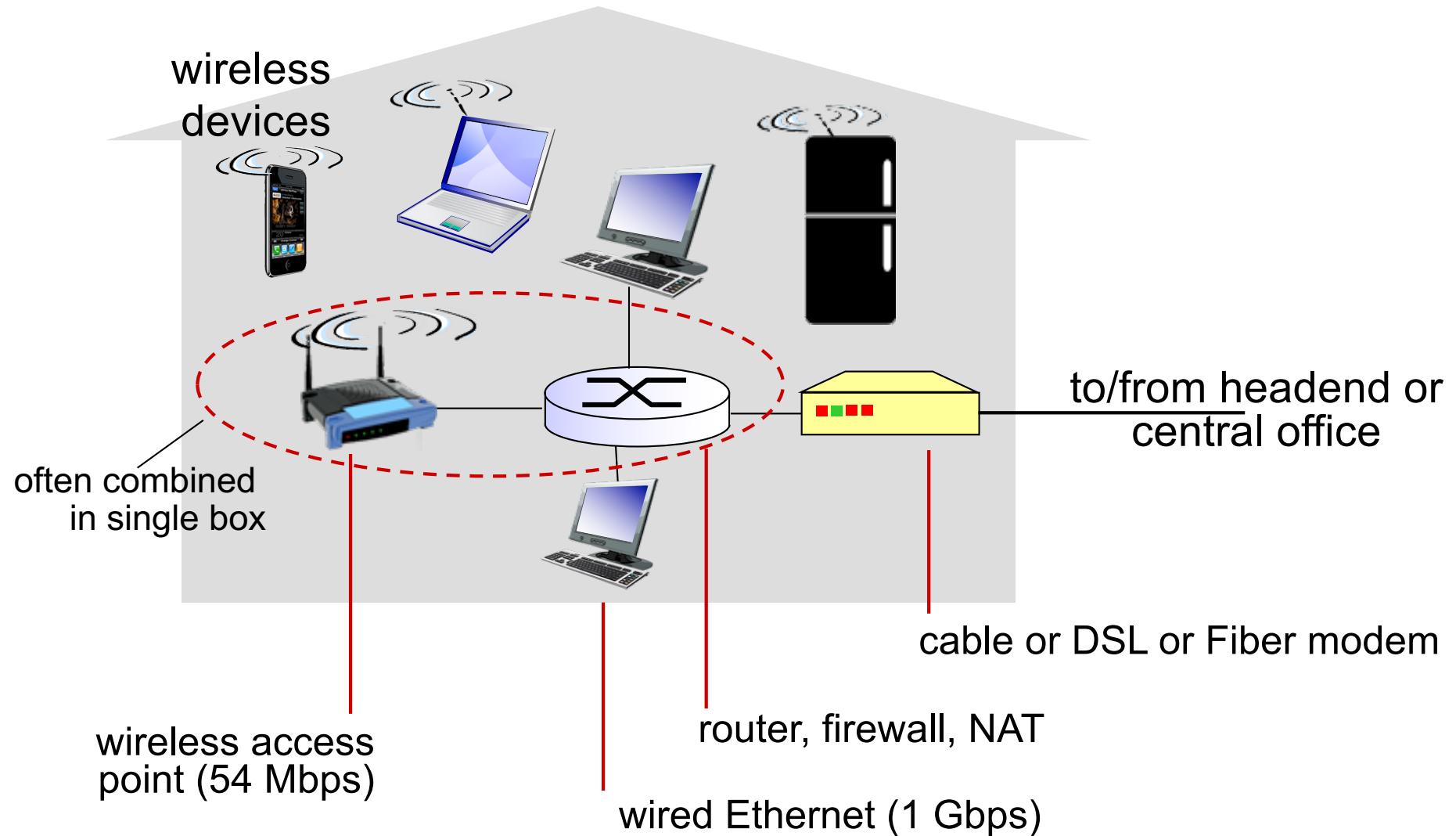
- ❖ HFC: hybrid fiber coax
  - asymmetric: up to 30Mbps downstream transmission rate, 2 Mbps upstream transmission rate
- ❖ network of cable, fiber attaches homes to ISP router
  - homes **share access network** to cable headend
  - unlike DSL, which has dedicated access to central office

# Fiber to the home/premise/curb

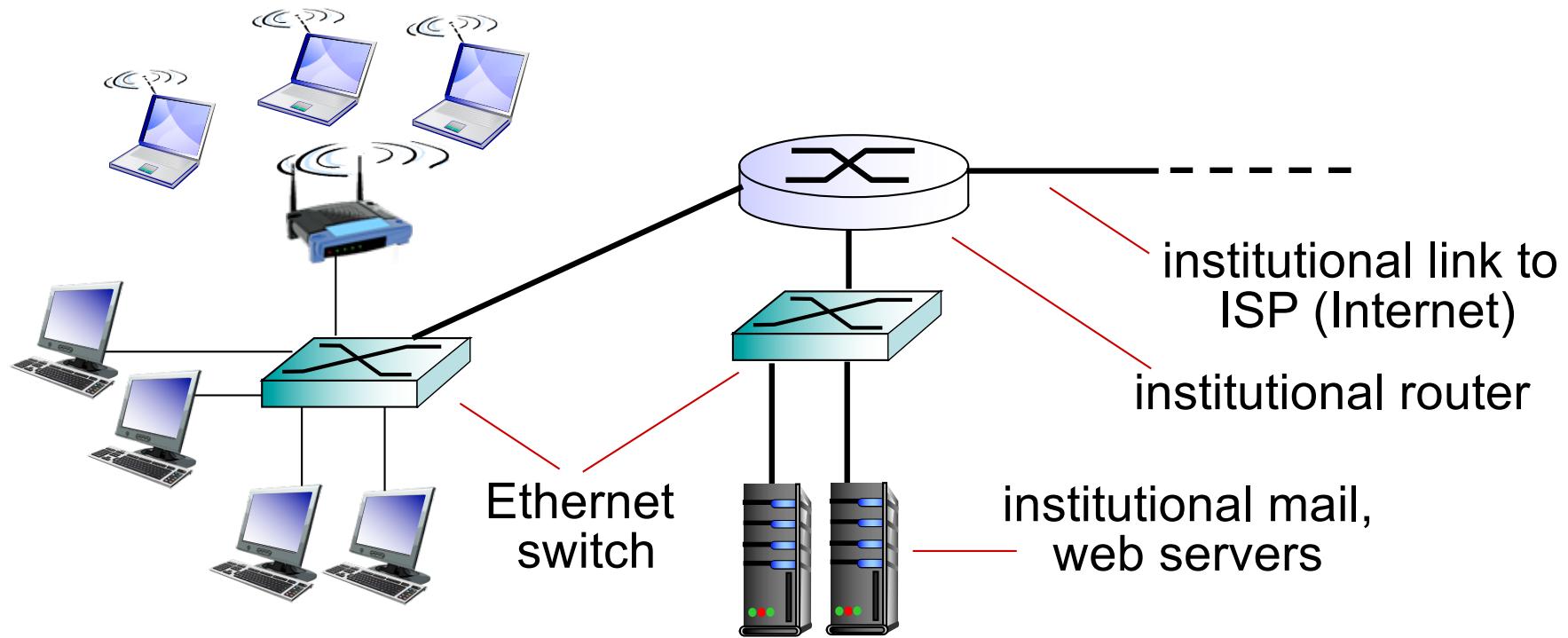
- ❖ Fully optical fiber path all the way to the home (or premise or curb)
  - e.g., NBN, Google, Verizon FIOS
  - ~30 Mbps to 1 Gbps



# Access net: home network



# Enterprise access networks (Ethernet)



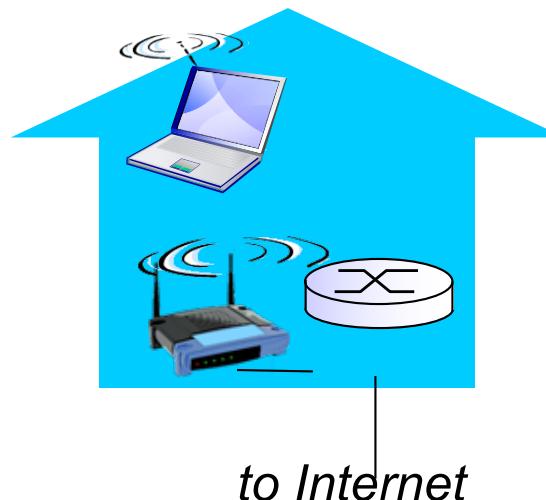
- ❖ typically used in companies, universities, etc
- ❖ 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- ❖ today, end systems typically connect into Ethernet switch

# Wireless access networks

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

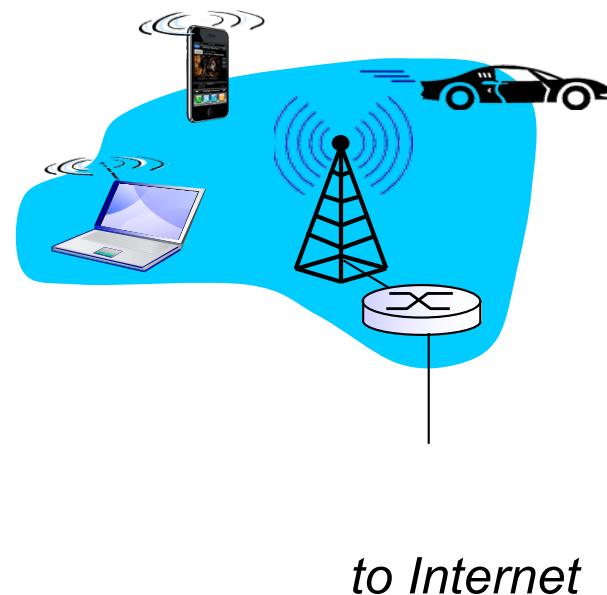
## wireless LANs:

- within building (100 ft)
- 802.11b/g/n (WiFi): 11, 54, 300 Mbps transmission rate
- 802.11ac: 1 Gbps(2.4GHz) + 4.34Gbps (5GHz)
- 802.11ax: WiFi 6



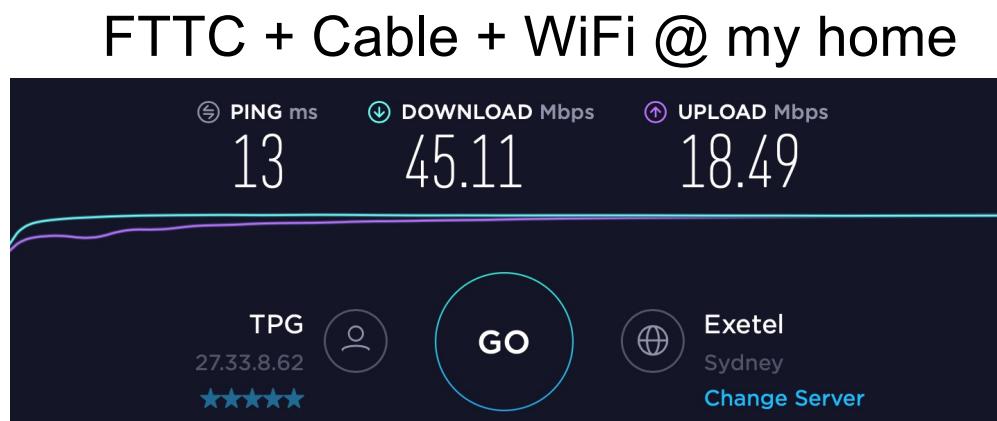
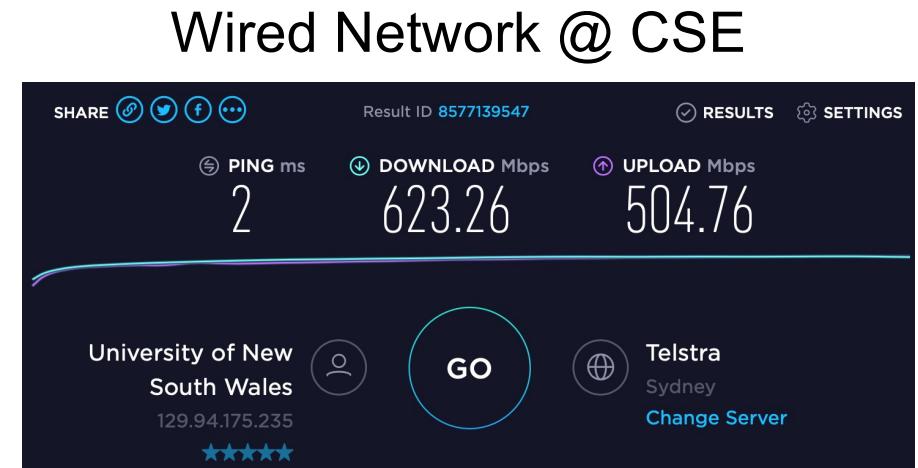
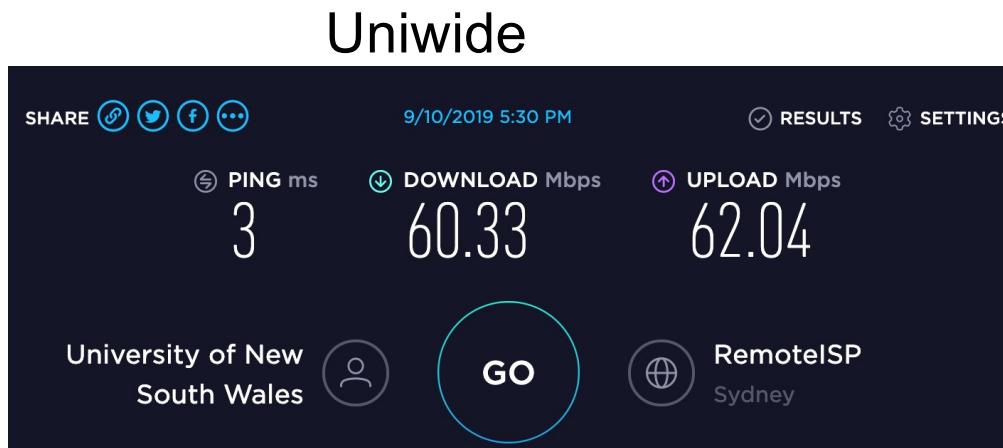
## wide-area wireless access

- provided by telco (cellular) operator, 10's km
- between 10 and 100 Mbps
- 4G, 5G



# Sample results

Can you explain the differences?





## Quiz: Your access network

Your residential ISP provides connectivity using the following technology:

- A. DSL
- B. Cable
- C. Fiber to the home/premise/curb
- D. Mobile (3G/4G/5G)
- E. Satellite
- F. Something Else (type in Zeetings comment)

Open a browser and type: **[www.zheetings.com/salil](http://www.zheetings.com/salil)**

# Physical media

Self Study

- ❖ **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

# Physical media: twisted pair, coax, fiber

## *twisted pair (TP)*

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



## *coaxial cable:*

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



Self Study

## *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

Self Study

- ❖ signal carried in electromagnetic spectrum, i.e., no physical “wire”
- ❖ propagation environment effects:
  - reflection
  - obstruction by objects
  - interference

## *radio link types:*

- ❖ **terrestrial microwave**
  - e.g. up to 45 Mbps channels
- ❖ **LAN** (e.g., WiFi)
  - 11Mbps, 54 Mbps, 450 Mbps, Gbps
- ❖ **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 earth-orbiting (LEO)

# I. Introduction: 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

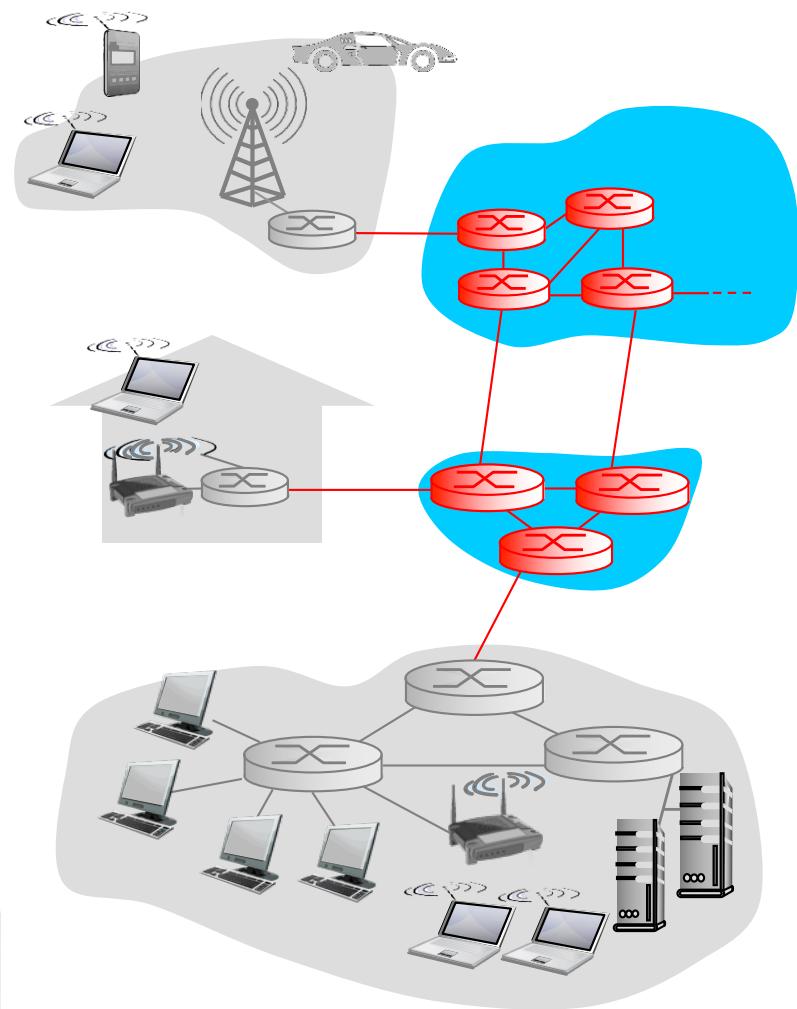
I.7 history

# The network core

- ❖ mesh of interconnected routers/switches
- ❖ Two forms of switched networks:
  - Circuit switching: used in the legacy telephone networks



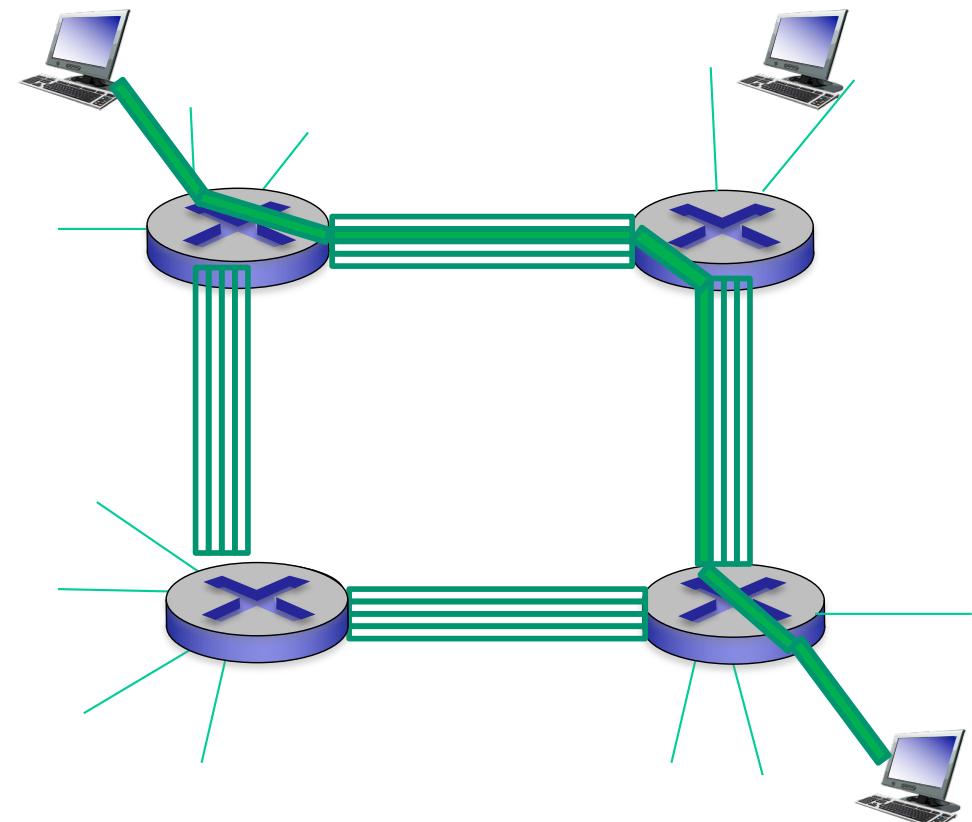
- Packet switching: used in the Internet



# Circuit Switching

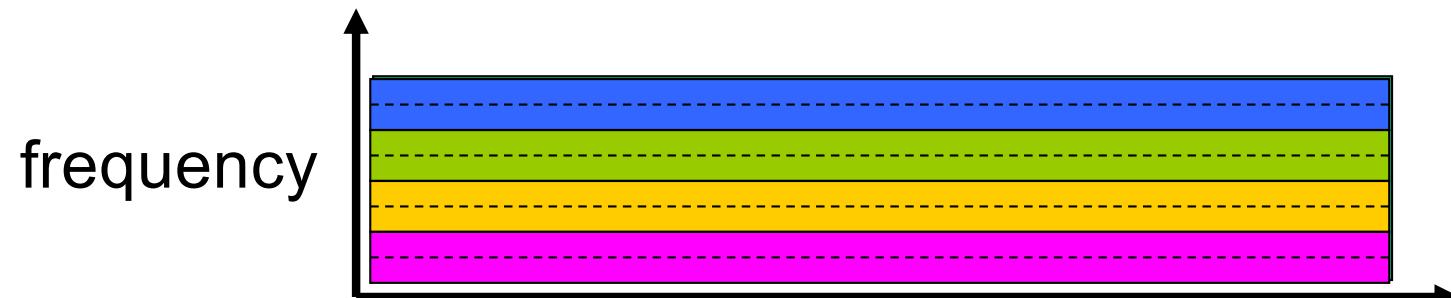
end-end resources allocated to, reserved for “call” between source & dest:

- 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

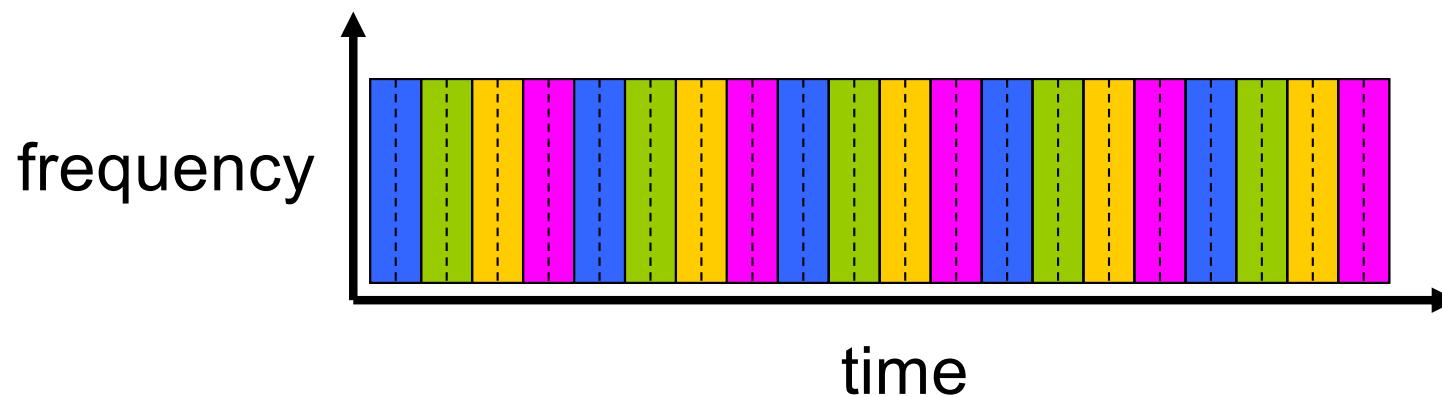


Example:

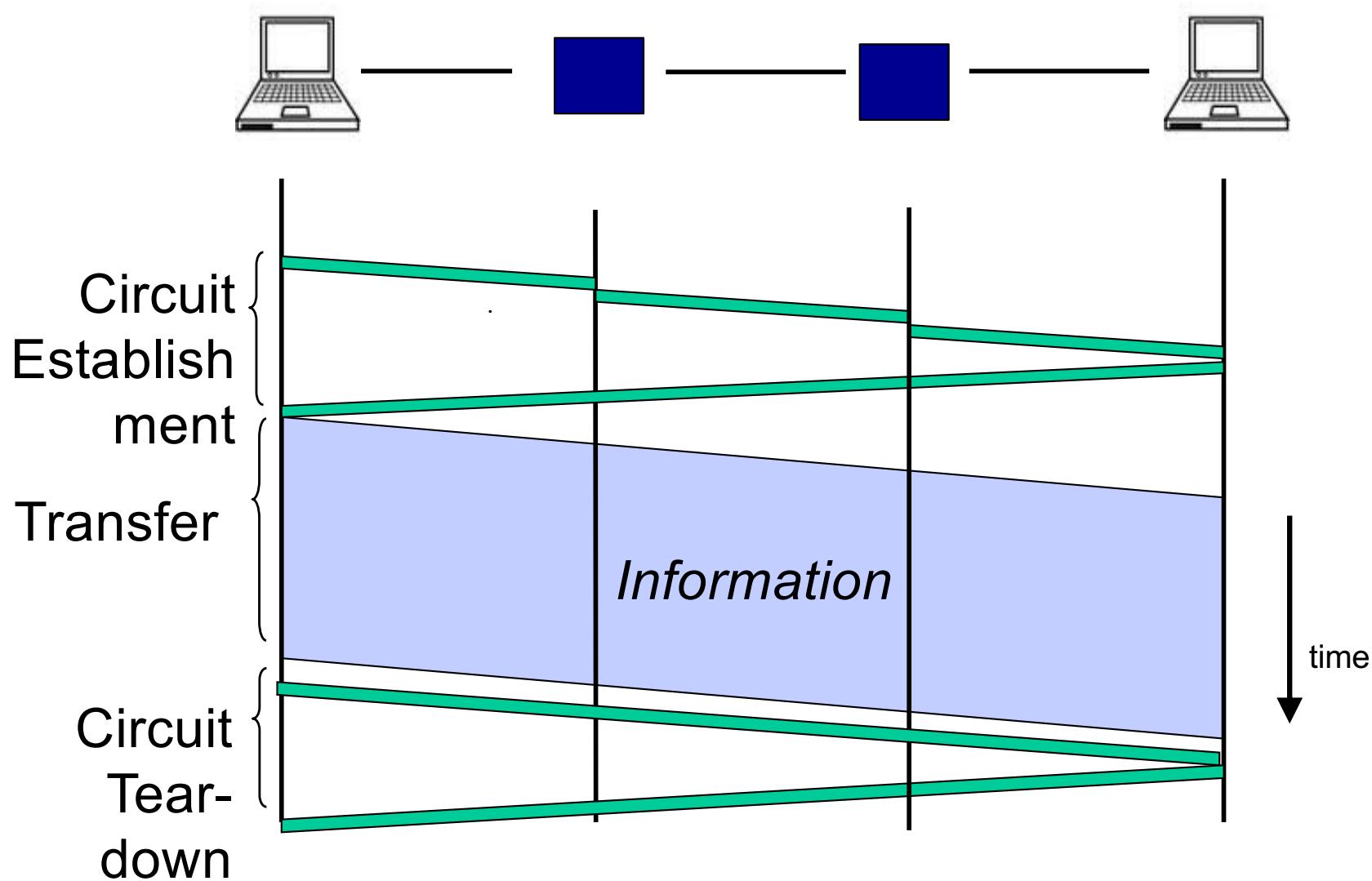
4 users



TDM



# Timing in Circuit Switching



# Why circuit switching is not feasible?

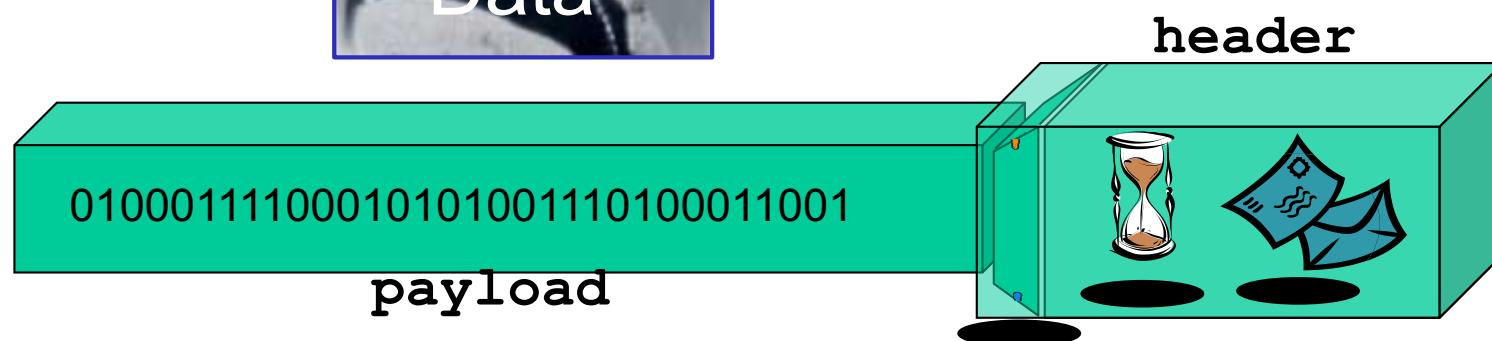
- **Inefficient**
  - Computer communications tends to be very bursty. For example, viewing a sequence of web pages
  - Dedicated circuit cannot be used or shared in periods of silence
  - Cannot adopt to network dynamics
- **Fixed data rate**
  - Computers communicate at very diverse rates. For example, viewing a video vs using telnet or web browsing
  - Fixed data rate is not useful
- **Connection state maintenance**
  - Requires per communication state to be maintained that is a considerable overhead
  - Not scalable

# Packet Switching

- ❖ Data is sent as chunks of formatted bits (**Packets**)
- ❖ Packets consist of a “**header**” and “**payload**”



1. Internet Address
2. Age (TTL)
3. Checksum to protect header



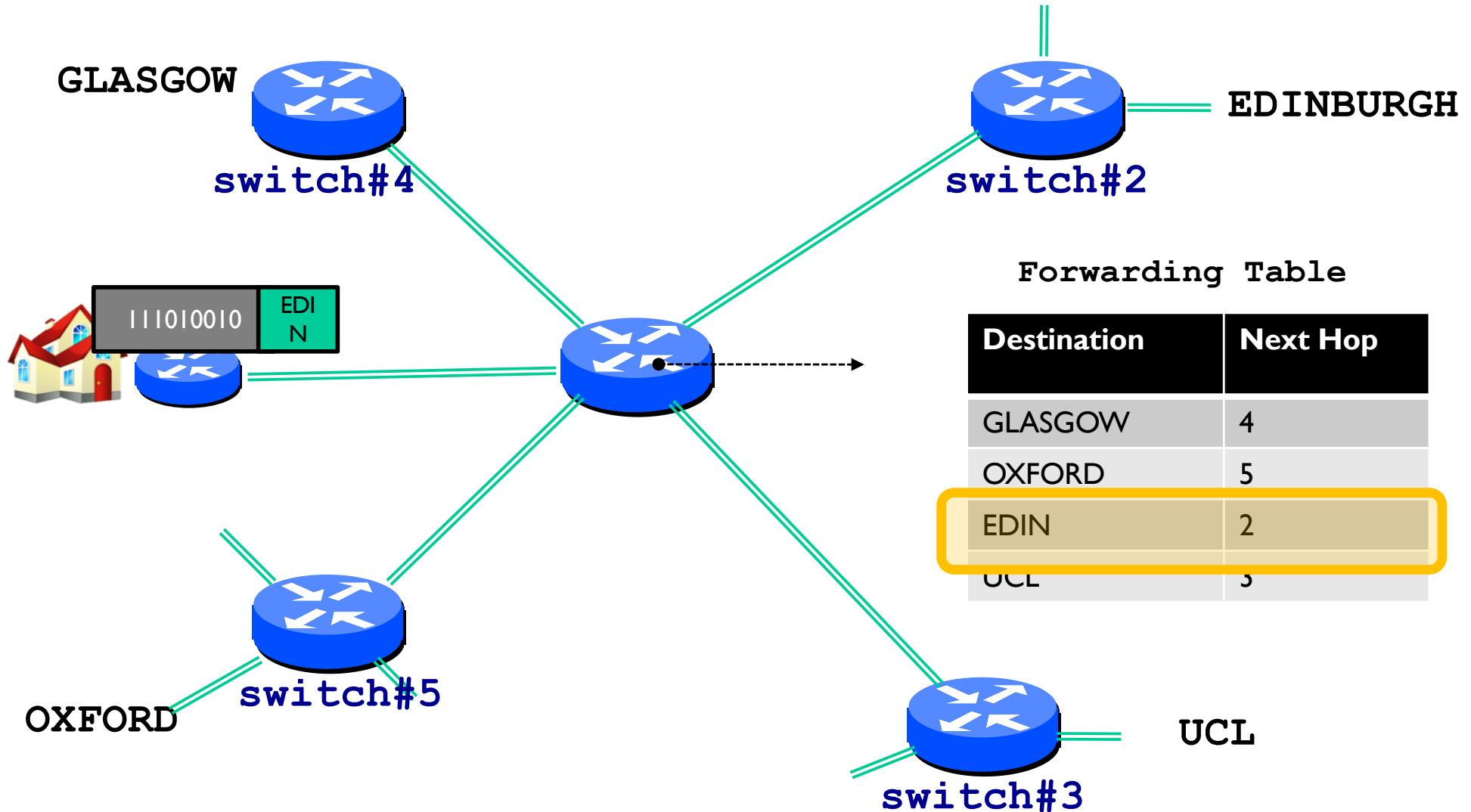
# Packet Switching

- ❖ Data is sent as chunks of formatted bits (**Packets**)
- ❖ Packets consist of a “**header**” and “**payload**”
  - payload is the data being carried
  - header holds instructions to the network for how to handle packet (think of the header as an API)

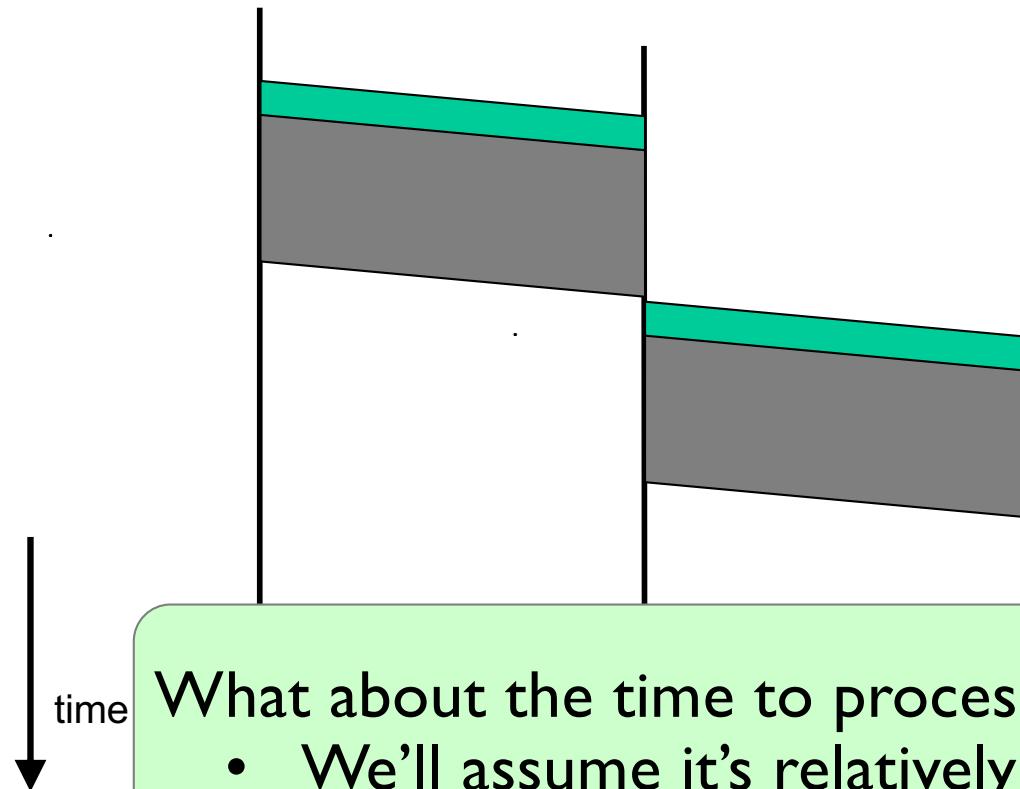
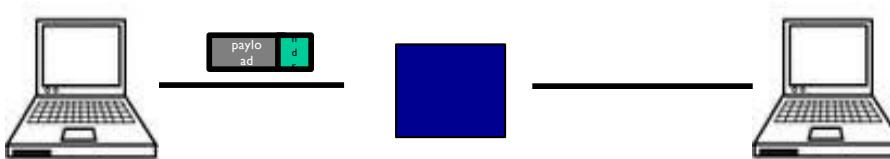
# Packet Switching

- ❖ Data is sent as chunks of formatted bits (Packets)
- ❖ Packets consist of a “header” and “payload”
- ❖ Switches “**forward**” packets based on their headers

# Switches forward packets

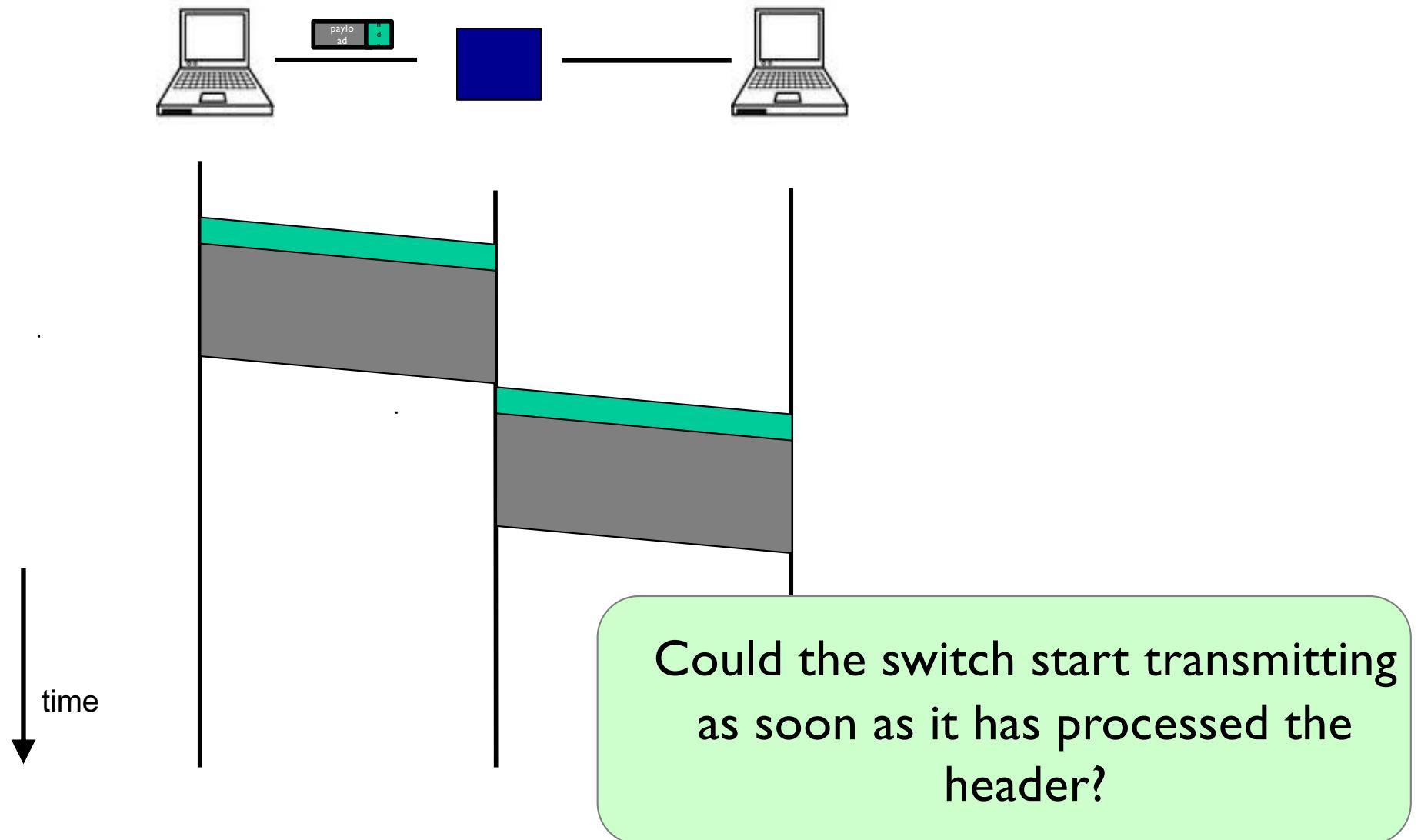


# Timing in Packet Switching

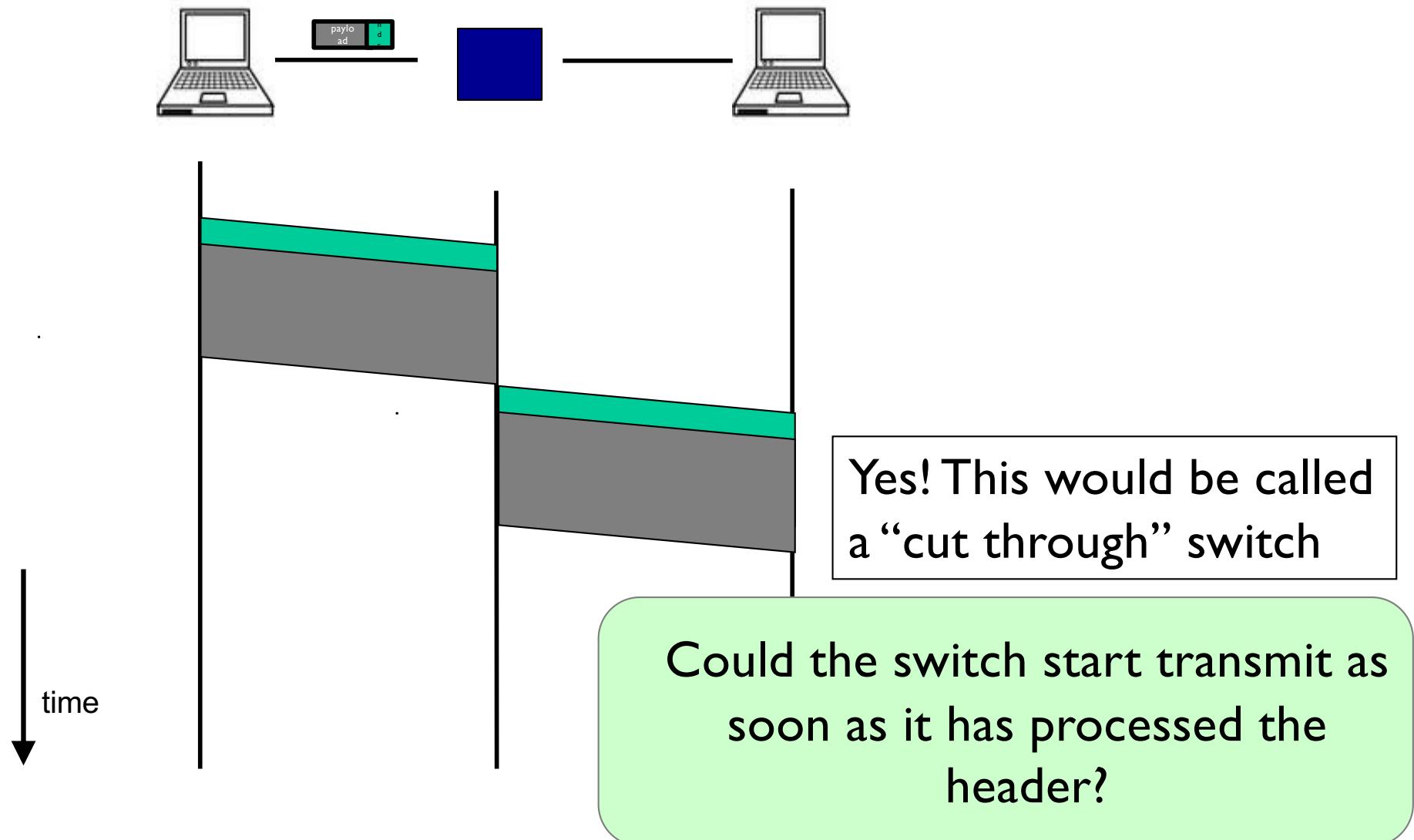


What about the time to process the packet at the switch?  
• We'll assume it's relatively negligible (mostly true)

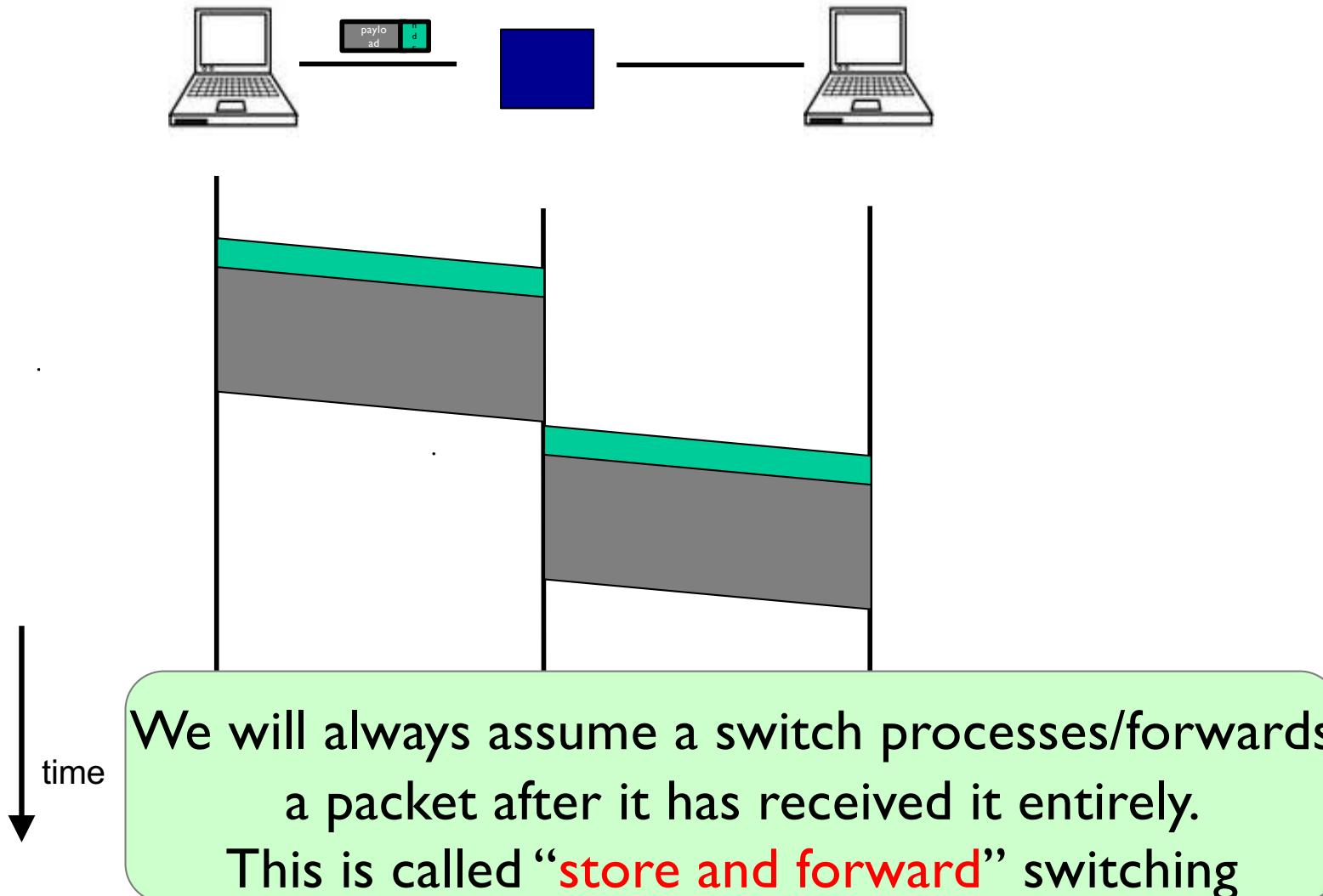
# Timing in Packet Switching



# Timing in Packet Switching



# Timing in Packet Switching



# Packet Switching

- ❖ Data is sent as chunks of formatted bits (Packets)
- ❖ Packets consist of a “header” and “payload”
- ❖ Switches “**forward**” packets based on their headers

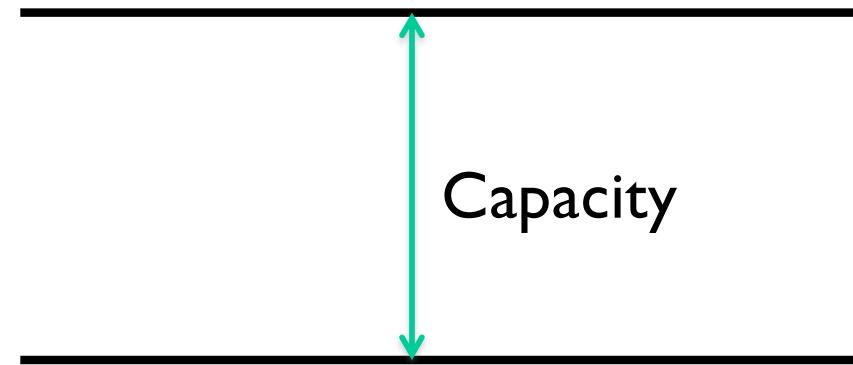
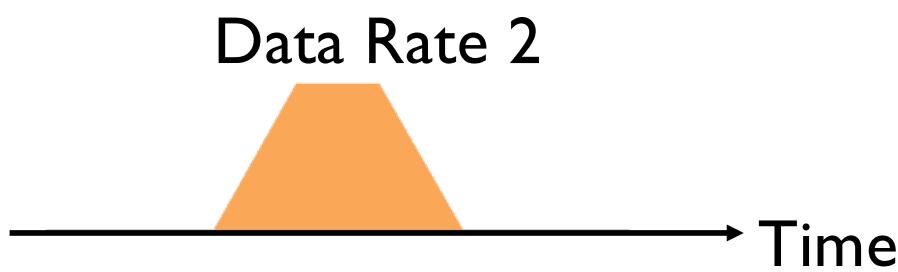
# Packet Switching

- ❖ Data is sent as chunks of formatted bits (Packets)
- ❖ Packets consist of a “header” and “payload”
- ❖ Switches “forward” packets based on their headers
- ❖ Each packet travels independently
  - no notion of packets belonging to a “circuit”

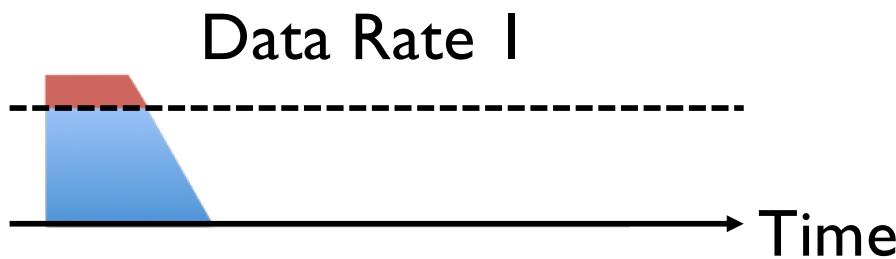
# Packet Switching

- ❖ Data is sent as chunks of formatted bits (Packets)
- ❖ Packets consist of a “header” and “payload”
- ❖ Switches “forward” packets based on their headers
- ❖ Each packet travels independently
- ❖ No link resources are reserved in advance. Instead, packet switching leverages **statistical multiplexing**

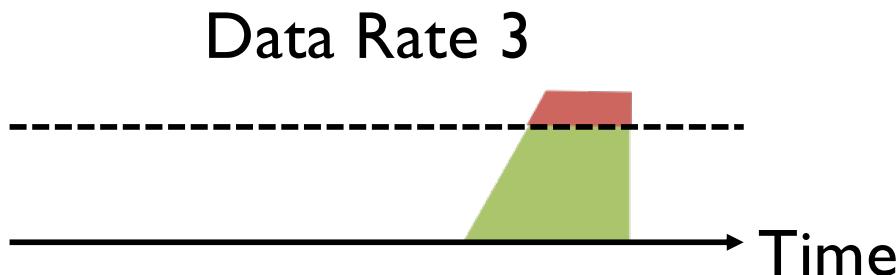
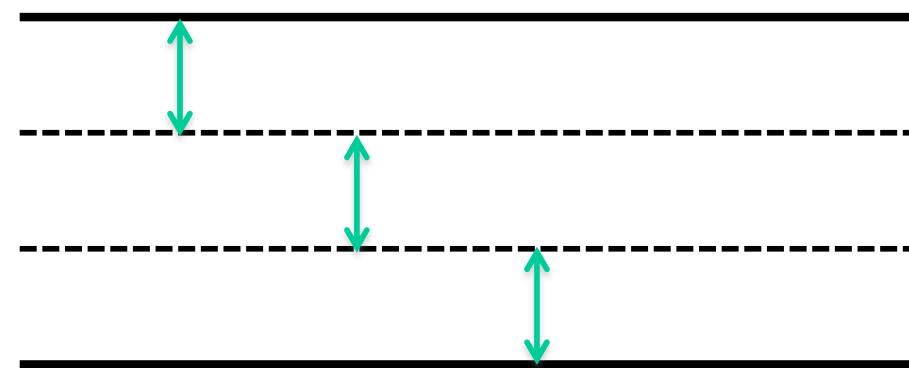
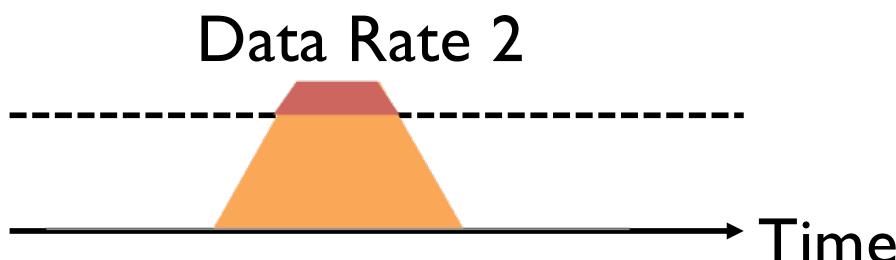
# Three Flows with Bursty Traffic



# When Each Flow Gets 1/3<sup>rd</sup> of Capacity



like circuit switching



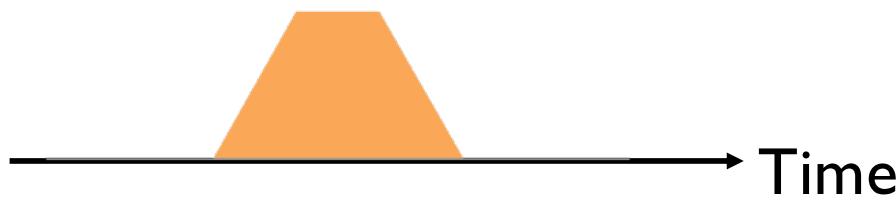
Overloaded

# When Flows Share Total Capacity

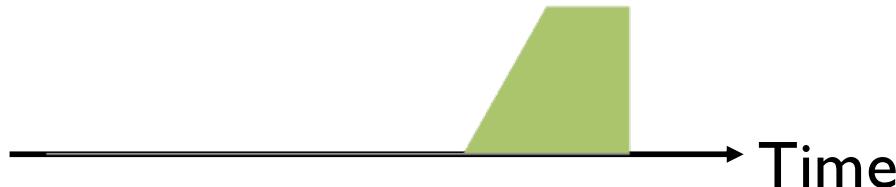
packet switching



No Overloading



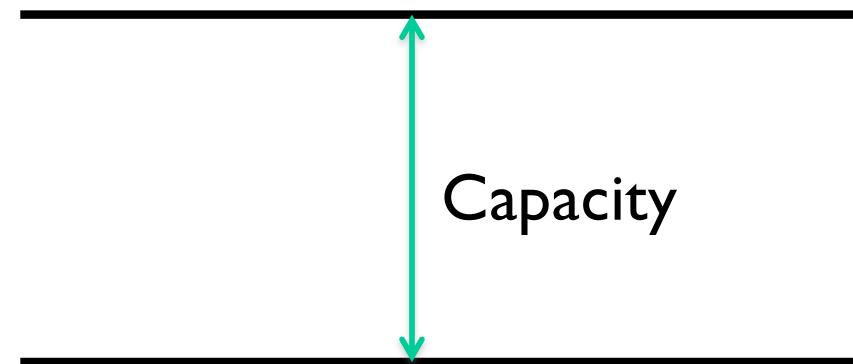
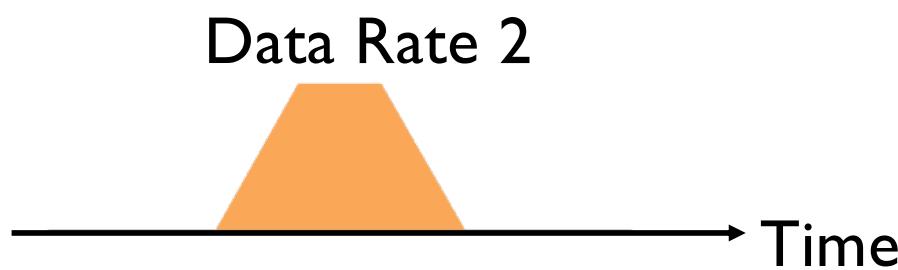
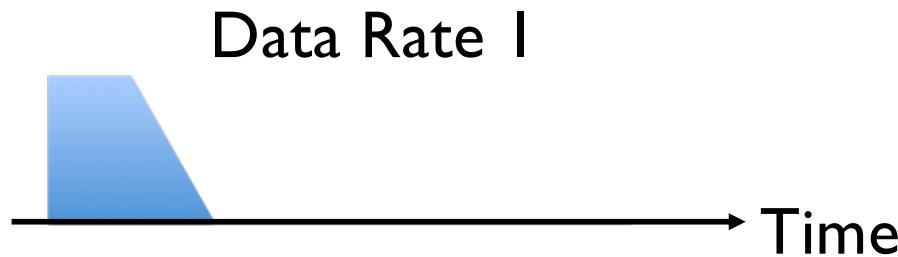
Statistical multiplexing relies on the assumption that not all flows burst at the same time



Very similar to insurance, and has same failure case

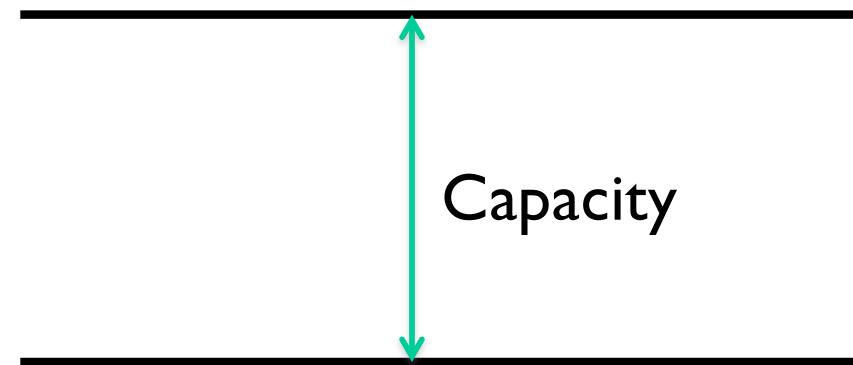
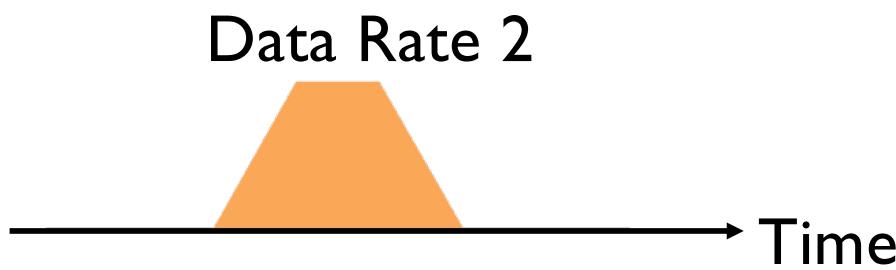
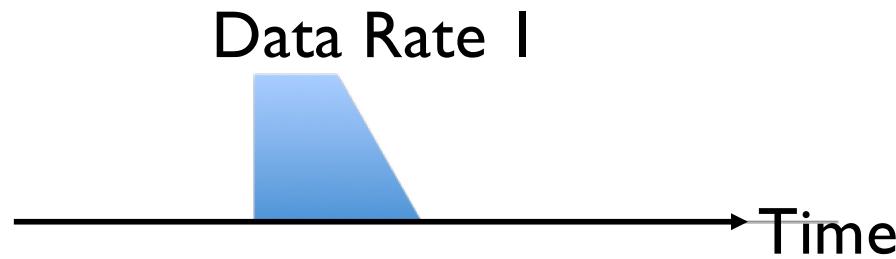
# Three Flows with Bursty Traffic

---



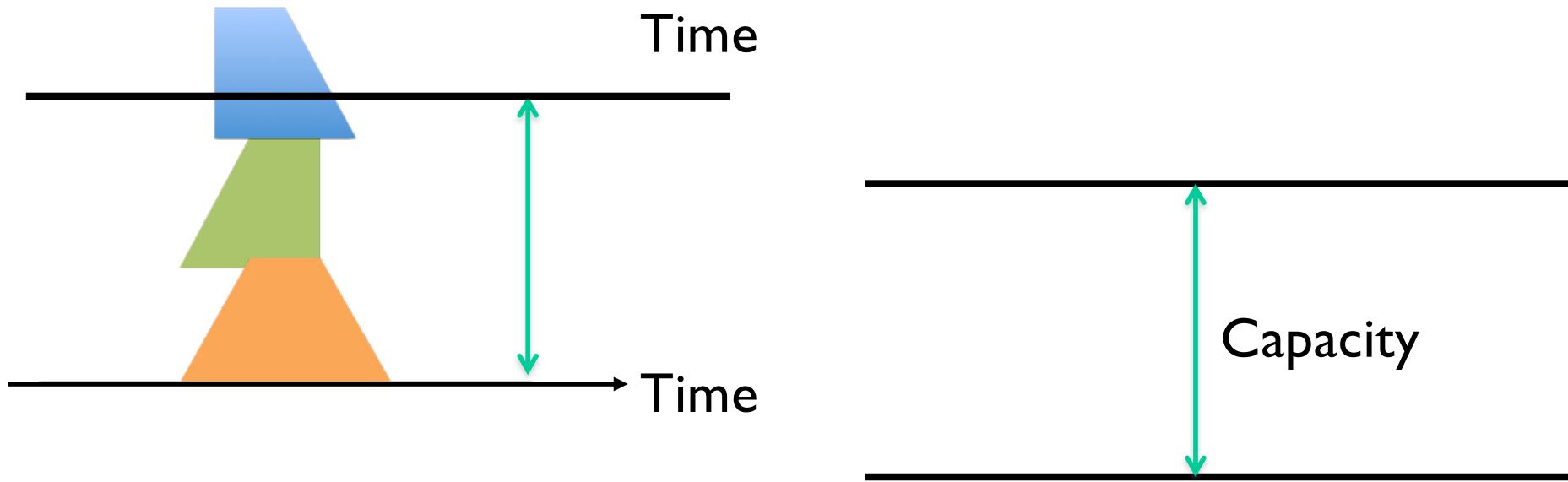
# Three Flows with Bursty Traffic

---



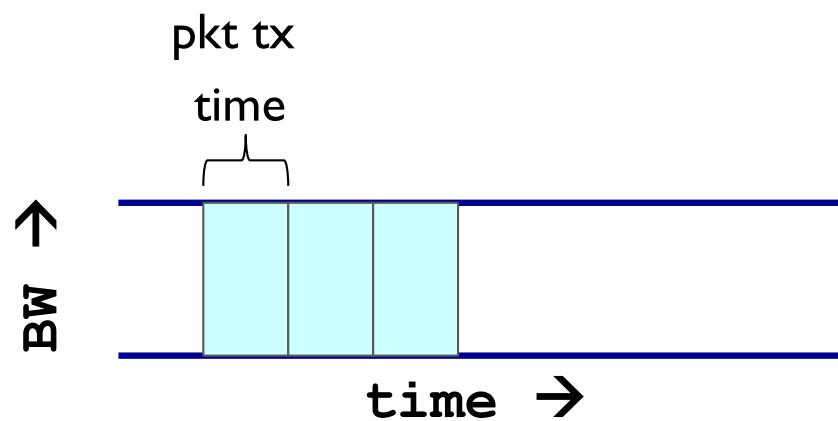
# Three Flows with Bursty Traffic

Data Rate 1+2+3 >> Capacity



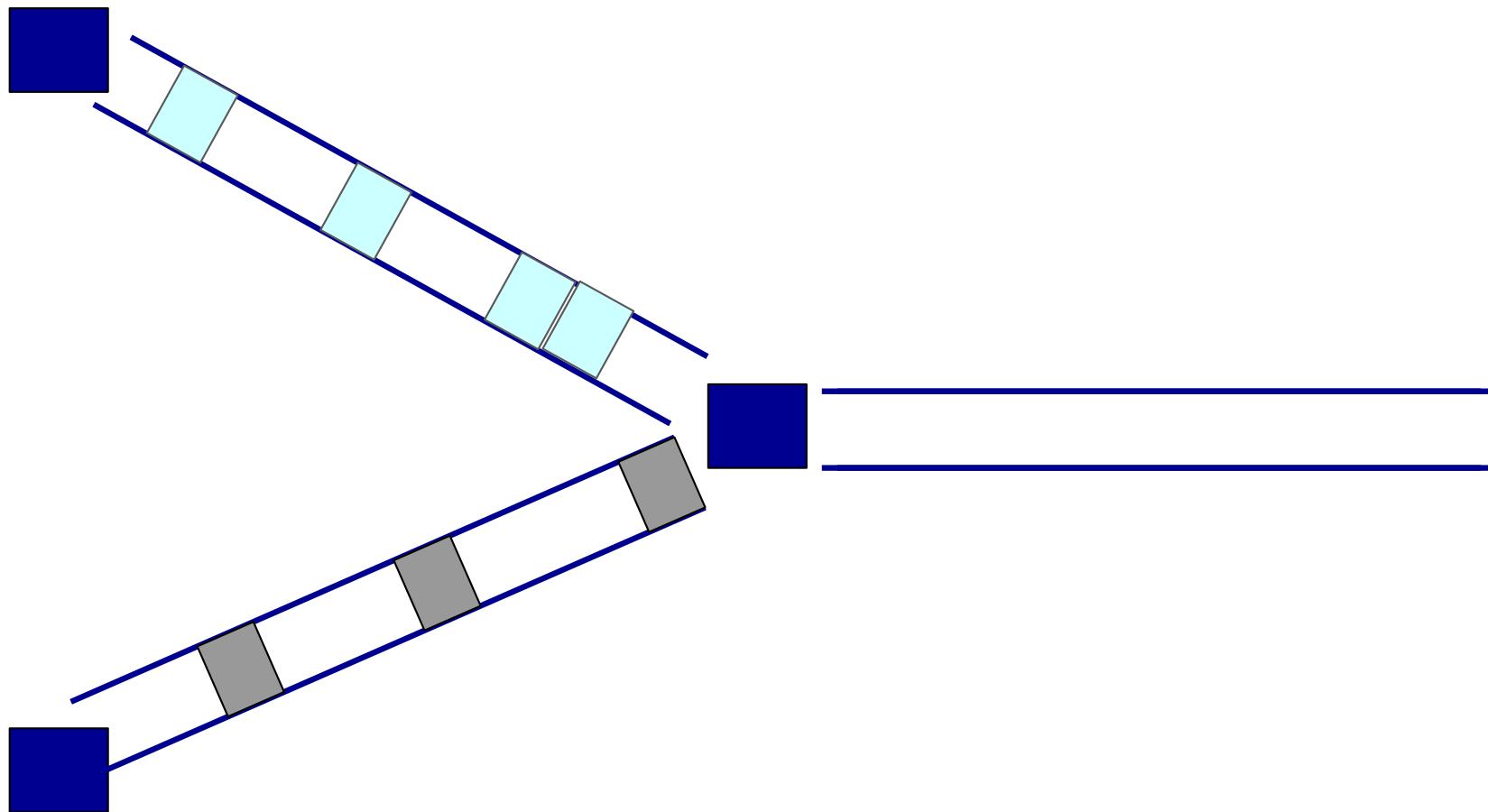
What do we do under overload?

# Statistical multiplexing: pipe view

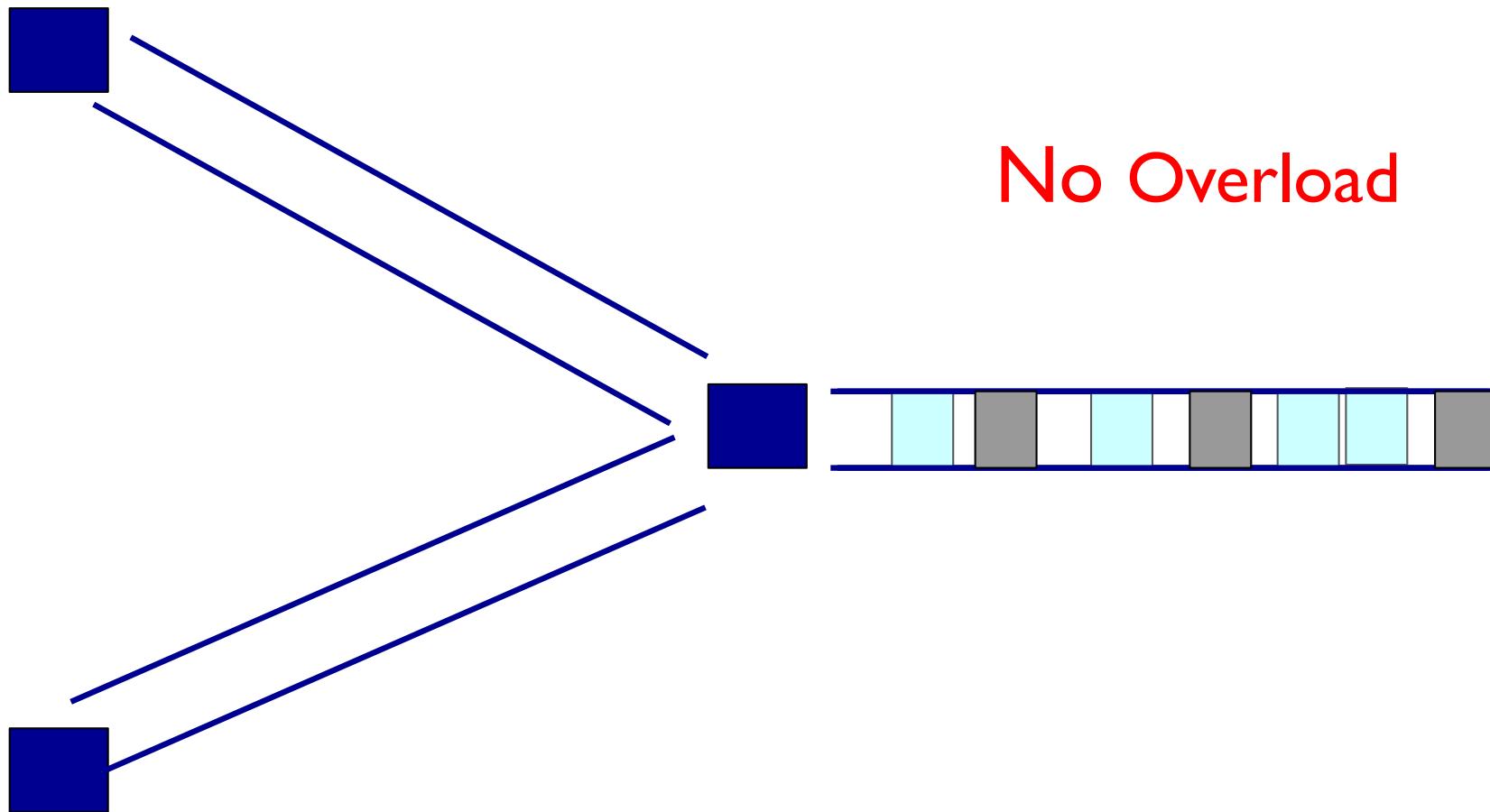


# Statistical multiplexing: pipe view

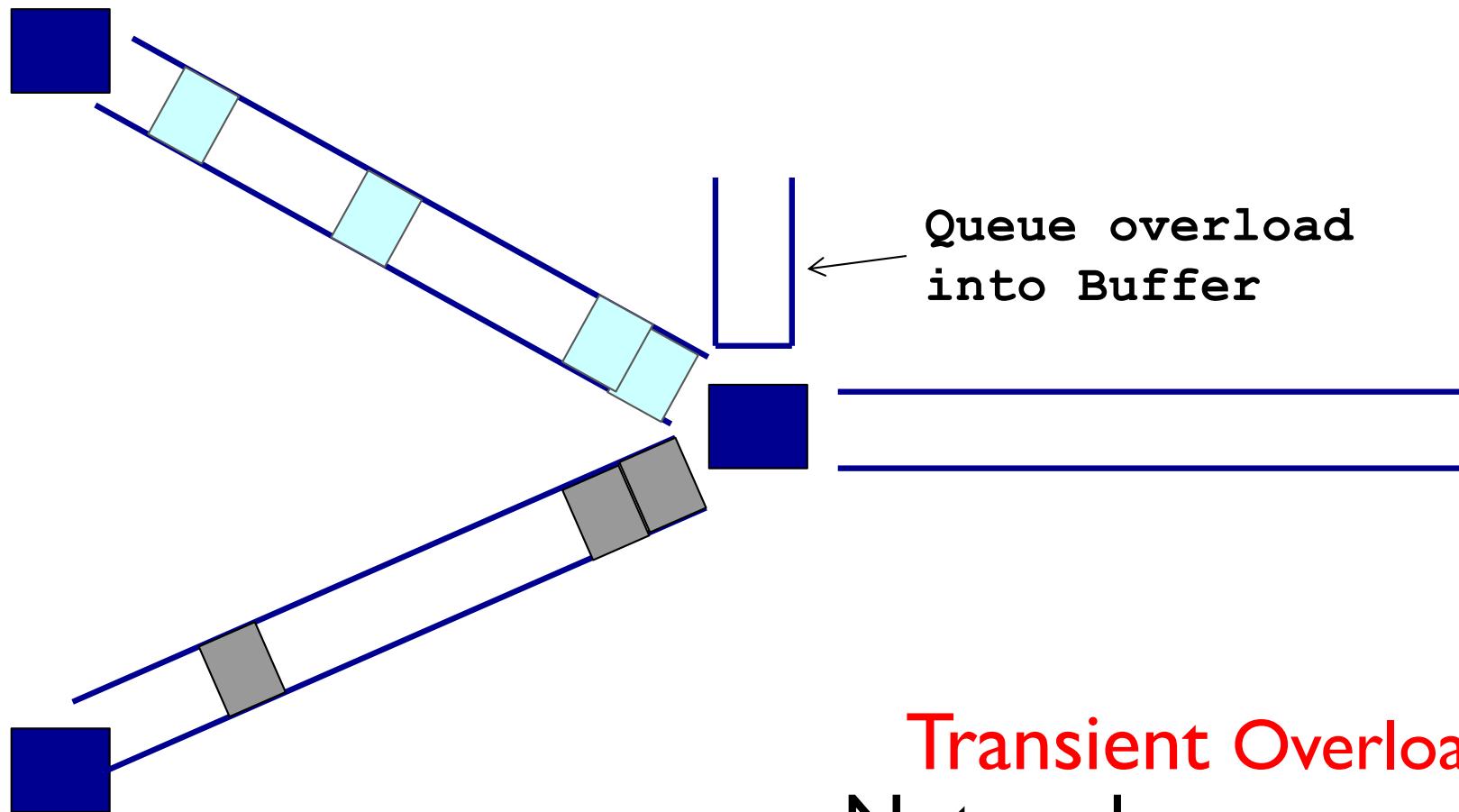
---



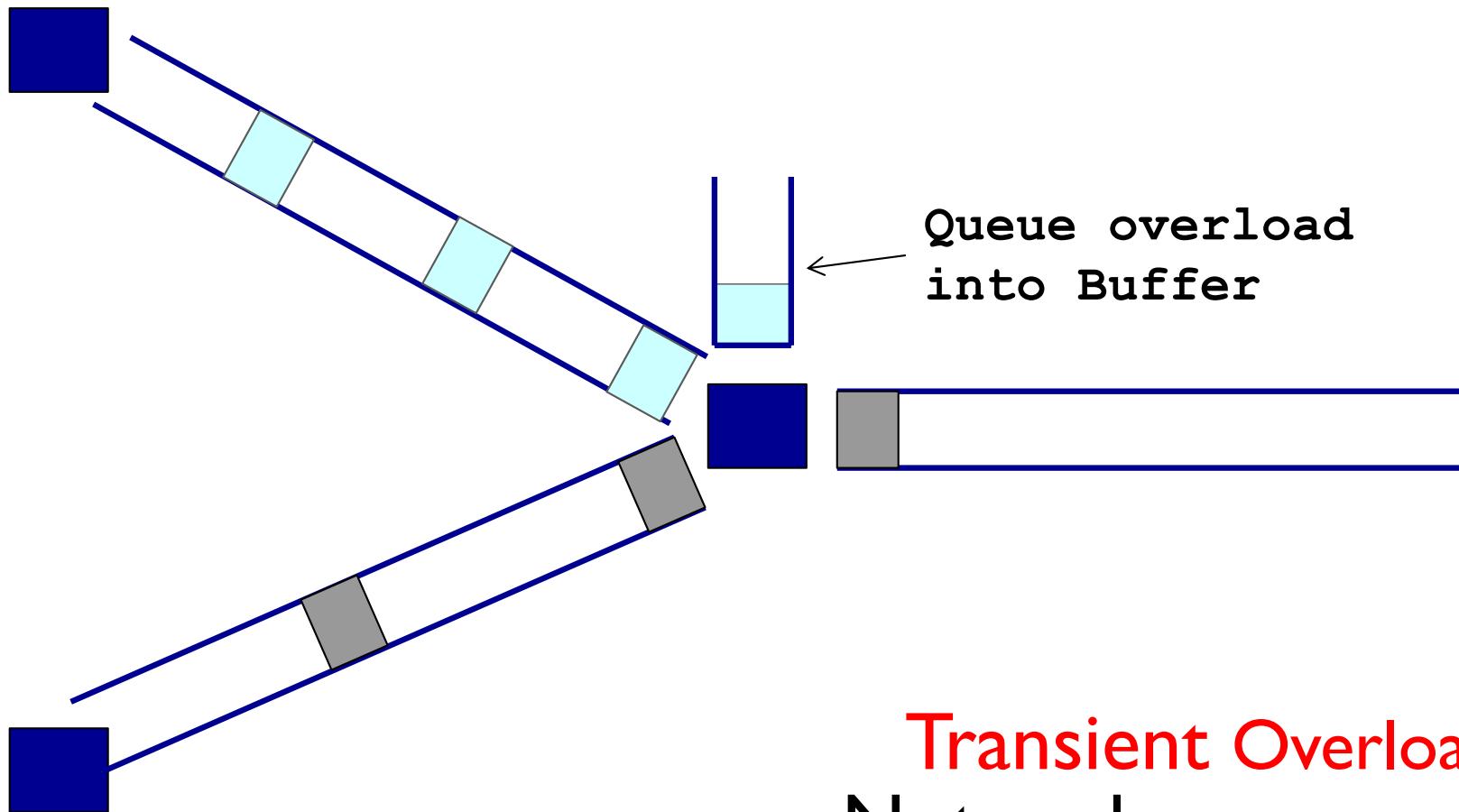
# Statistical multiplexing: pipe view



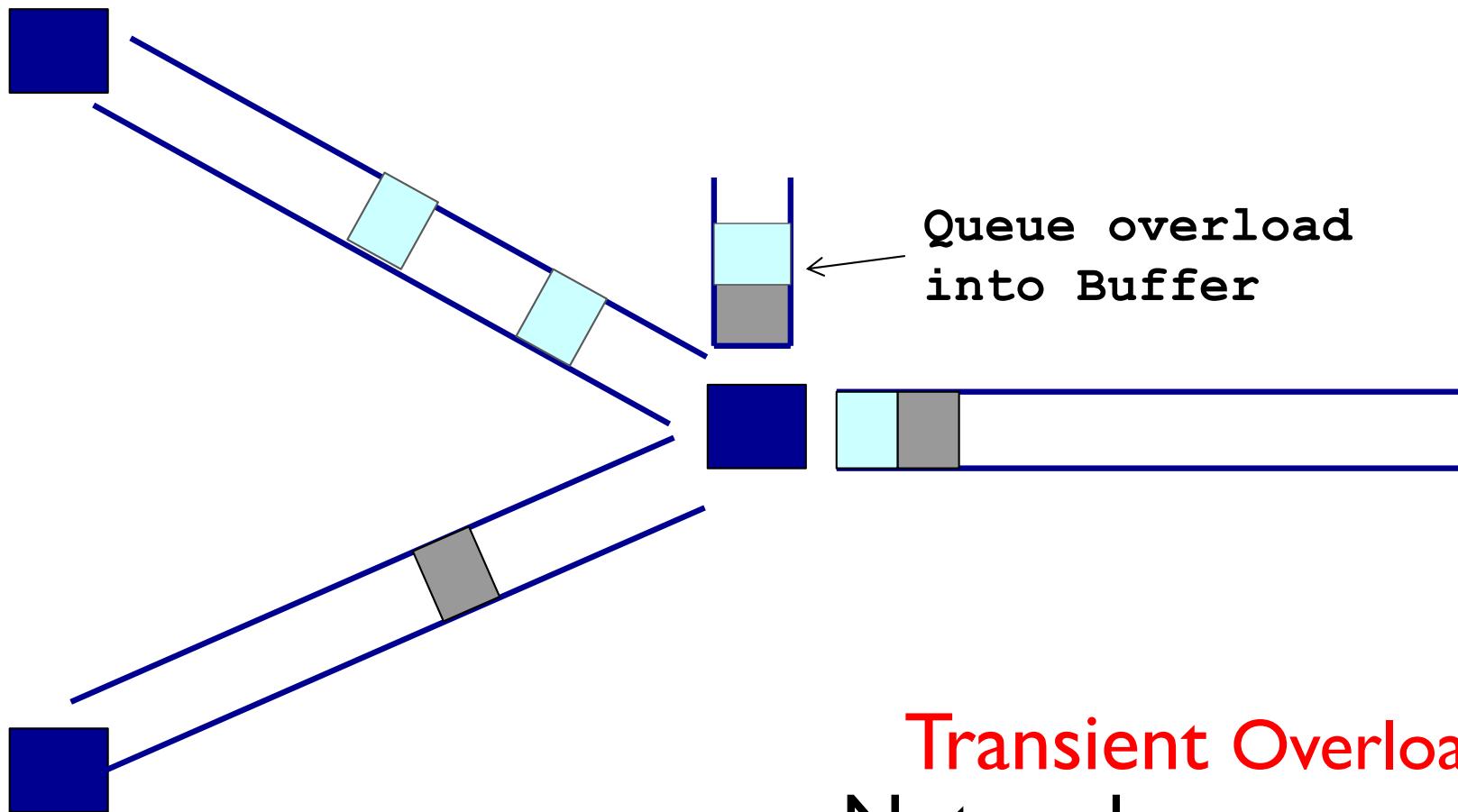
# Statistical multiplexing: pipe view



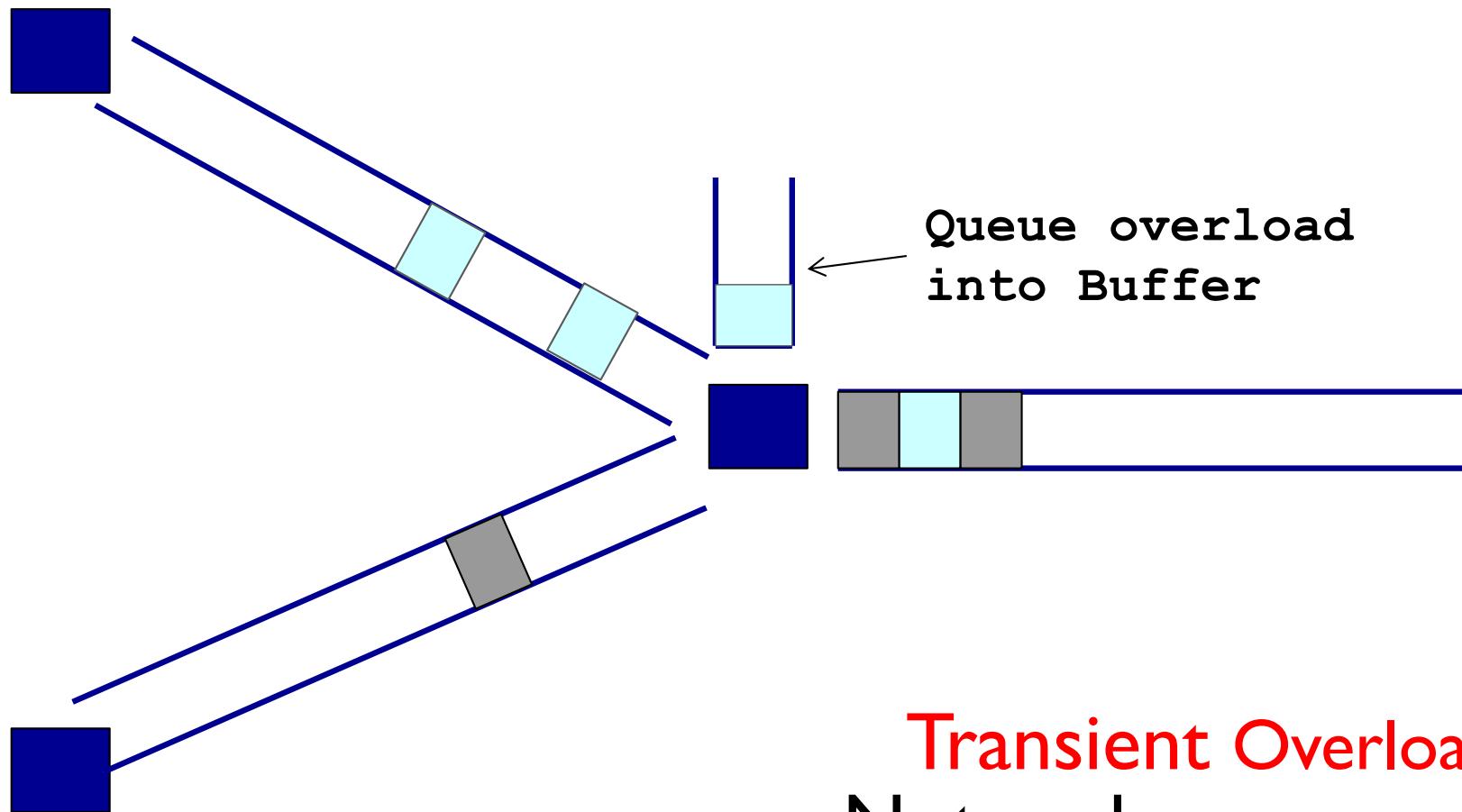
# Statistical multiplexing: pipe view



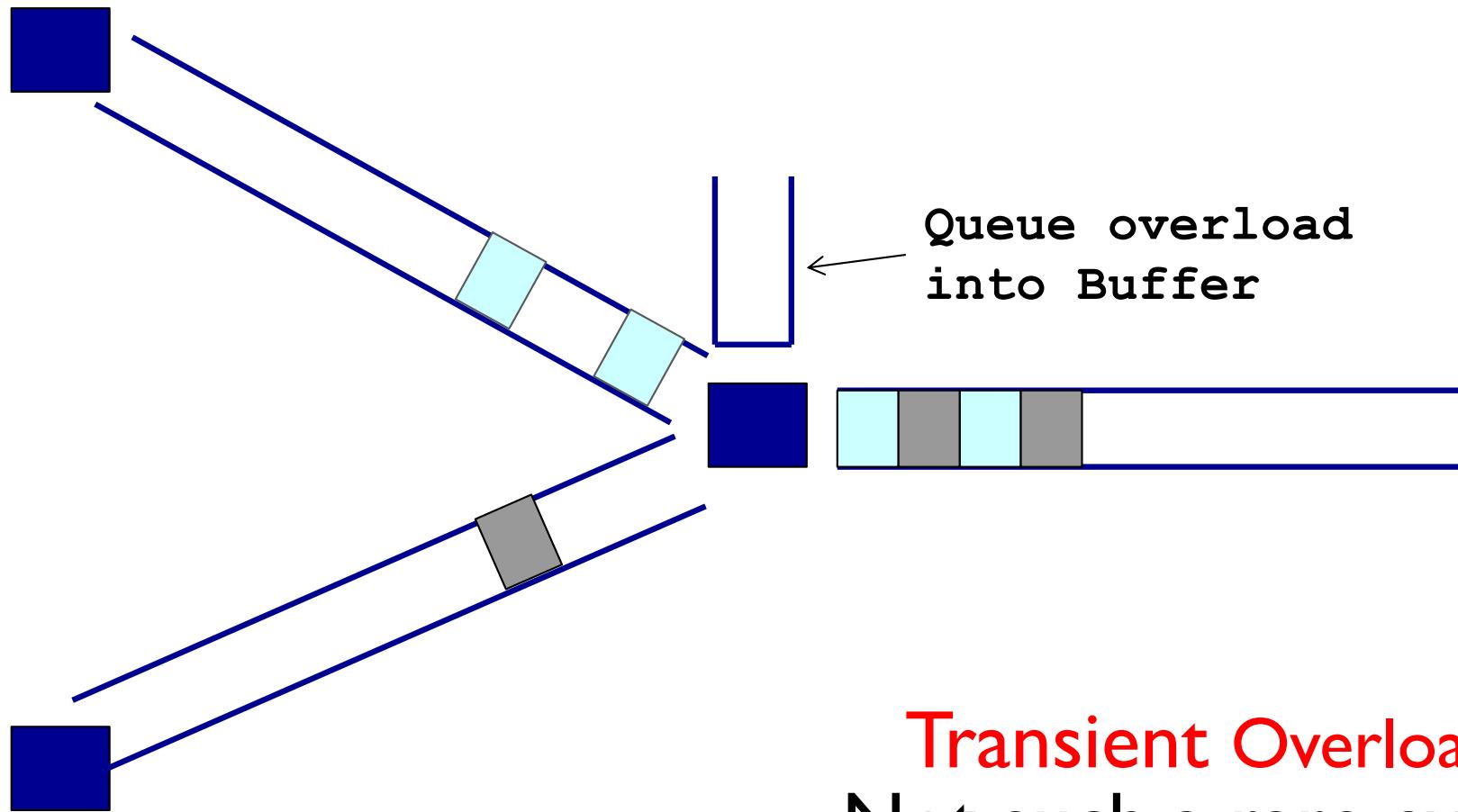
# Statistical multiplexing: pipe view



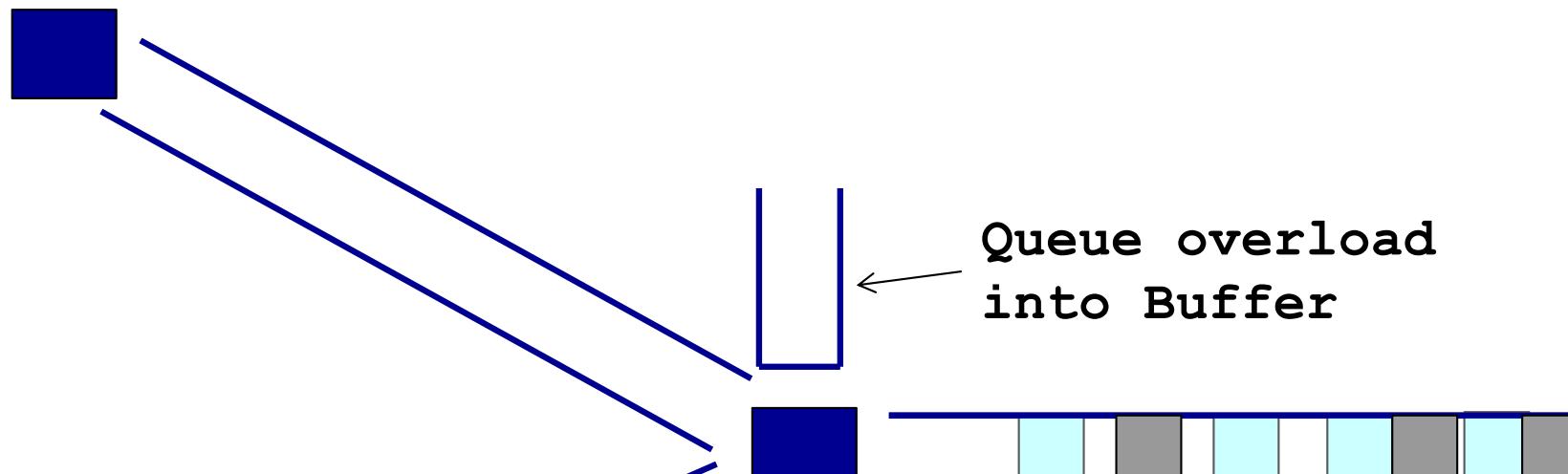
# Statistical multiplexing: pipe view



# Statistical multiplexing: pipe view

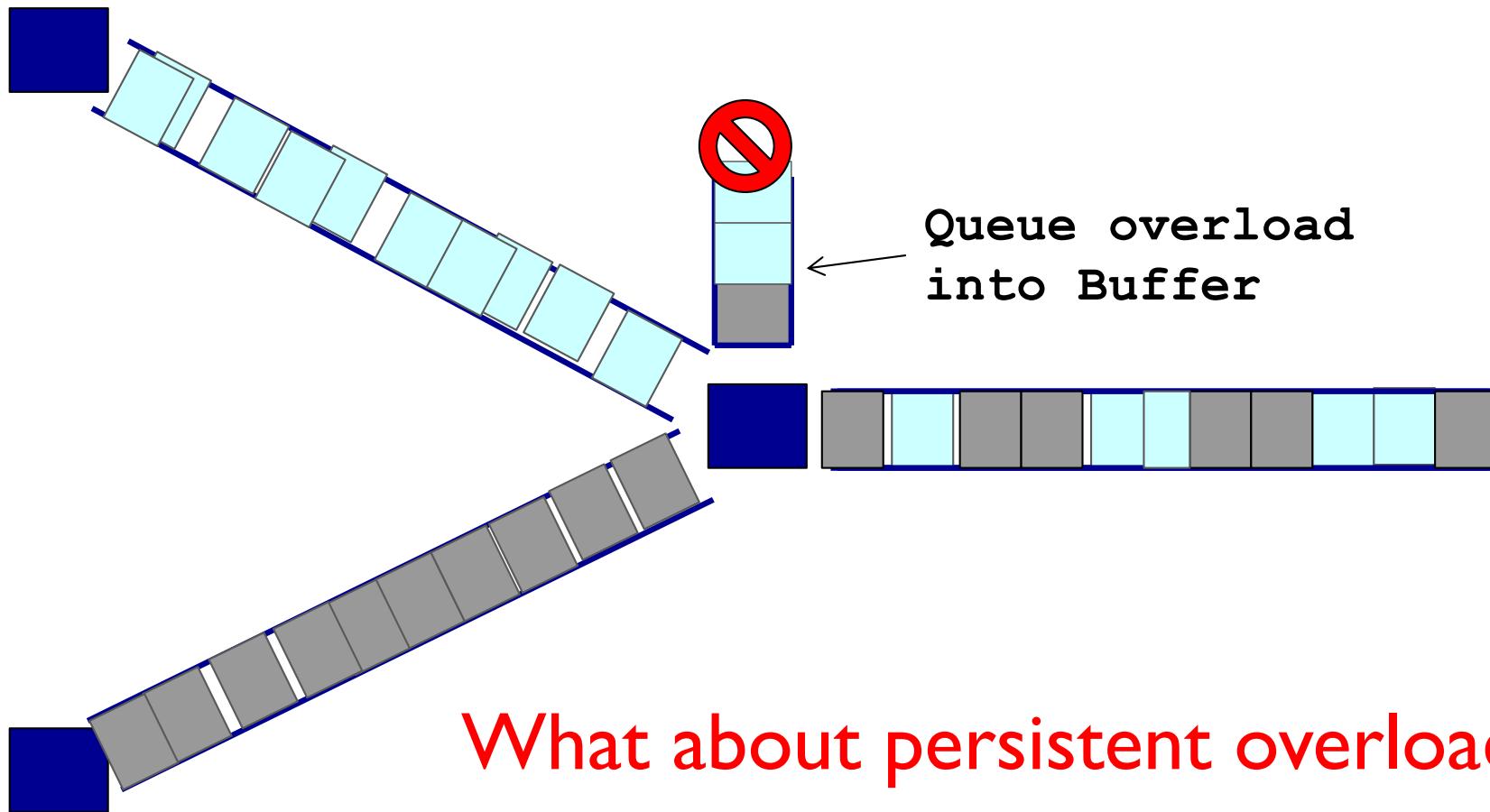


# Statistical multiplexing: pipe view



Buffer absorbs transient bursts

# Statistical multiplexing: pipe view

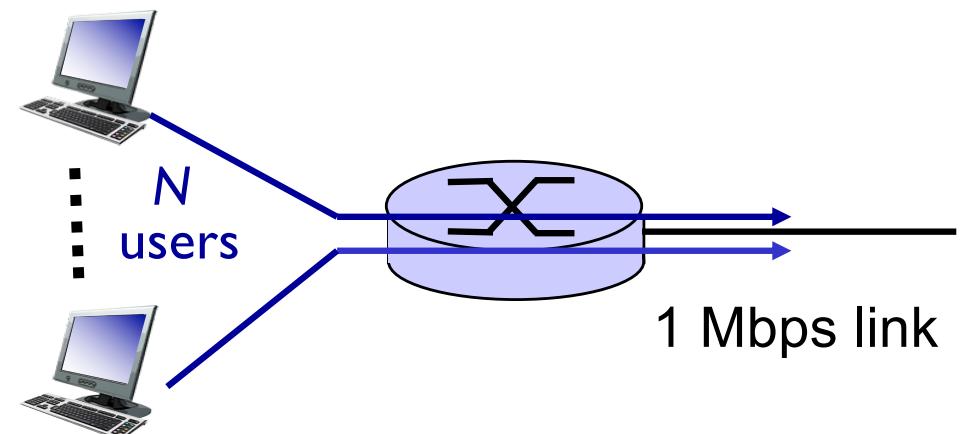


# Packet switching versus circuit switching

*packet switching allows more users to use network!*

example:

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



❖ *circuit-switching:*

- 10 users

❖ *packet switching:*

- with 35 users, probability > 10 active at same time is less than .0004

*Q: how did we get value 0.0004?*

*Q: what happens if > 35 users say 70?*

**Hint: Bernoulli Trials and Binomial Distribution**

# Binomial Probability Distribution

- ❖ A fixed number of observations (trials),  $n$ 
  - E.g., 5 tosses of a coin
- ❖ Binary random variable
  - E.g., head or tail in a coin toss
  - Often called as success or failure
  - Probability of success is  $p$  and failure is  $(1-p)$
- ❖ Constant probability for each observation

# Binomial Distribution: Example

---

- ❖ Q: What is the probability of observing exactly 3 heads in a sequence of 5 coin tosses
- ❖ A:
  - One way to get exactly 3 heads is: HHHTT
  - Probability of this sequence occurring =  $(1/2) \times (1/2) \times (1/2) \times (1-1/2) \times (1-1/2) = (1/2)^5$
  - Another way to get exactly 3 heads is: THHHT
  - Probability of this sequence occurring =  $(1-1/2) \times (1/2) \times (1/2) \times (1/2) \times (1-1/2) = (1/2)^5$
  - How many such unique combinations exist?

# Binomial Distribution: Example

$$\binom{5}{3}$$

ways to  
arrange 3  
heads in  
5 trials

$${}_5C_3 = 5!/3!2! = 10$$

Outcome	Probability
THHHT	$(1/2)^3 \times (1/2)^2$
HHHTT	$(1/2)^3 \times (1/2)^2$
TTHHH	$(1/2)^3 \times (1/2)^2$
HTTHH	$(1/2)^3 \times (1/2)^2$
HHTTH	$(1/2)^3 \times (1/2)^2$
THTHH	$(1/2)^3 \times (1/2)^2$
HTHTH	$(1/2)^3 \times (1/2)^2$
HHTHT	$(1/2)^3 \times (1/2)^2$
THHTH	$(1/2)^3 \times (1/2)^2$
<u>HTHHT</u>	<u><math>(1/2)^3 \times (1/2)^2</math></u>

$$10 \text{ arrangements} \times (1/2)^3 \times (1/2)^2$$

The probability  
of each unique  
outcome (note:  
they are all  
equal)

$$P(3 \text{ heads and 2 tails}) = 10 \times (1/2)^5 = 0.3125$$

# Binomial Distribution

---

Note the general pattern emerging → if you have only two possible outcomes (call them 1/0 or yes/no or success/failure) in  $n$  independent trials, then the probability of exactly  $X$  “successes” =

$$\binom{n}{X} p^X (1-p)^{n-X}$$

$n$  = number of trials

$X$  = # successes out of  $n$  trials

$p$  = probability of success

$1-p$  = probability of failure

# Packet switching versus circuit switching

---

- ❖ Let's revisit the earlier problem
- ❖  $N = 35$  users
- ❖  $\text{Prob } (\# \text{ active users} > 10) = 1 - \text{Prob } (\# \text{ active} = 10)$   
    –  $\text{Prob } (\# \text{ active} = 9)$   
    –  $\text{Prob } (\# \text{ active} = 8)$   
    ...  
    –  $\text{Prob } (\# \text{ active} = 0)$

where  $\text{Prob } (\# \text{ active} = 10) = C(35,10) \times 0.1^{10} \times 0.9^{25}$

- ❖  $\text{Prob } (\# \text{ active users} > 10) = 0.0004$  (approx)

# 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

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



## Quiz: Switching

In \_\_\_\_\_ resources are allocated on demand

- A. Packet switching
- B. Circuit switching
- C. Both
- D. None

**Answer: A**

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)



## Quiz: Switching

A message from device A to B consists of packet X and packet Y. In a circuit switched network, packet Y's path \_\_\_\_\_ packet X's path

- A. is the same
- B. is independent
- C. is always different from

**Answer: A**

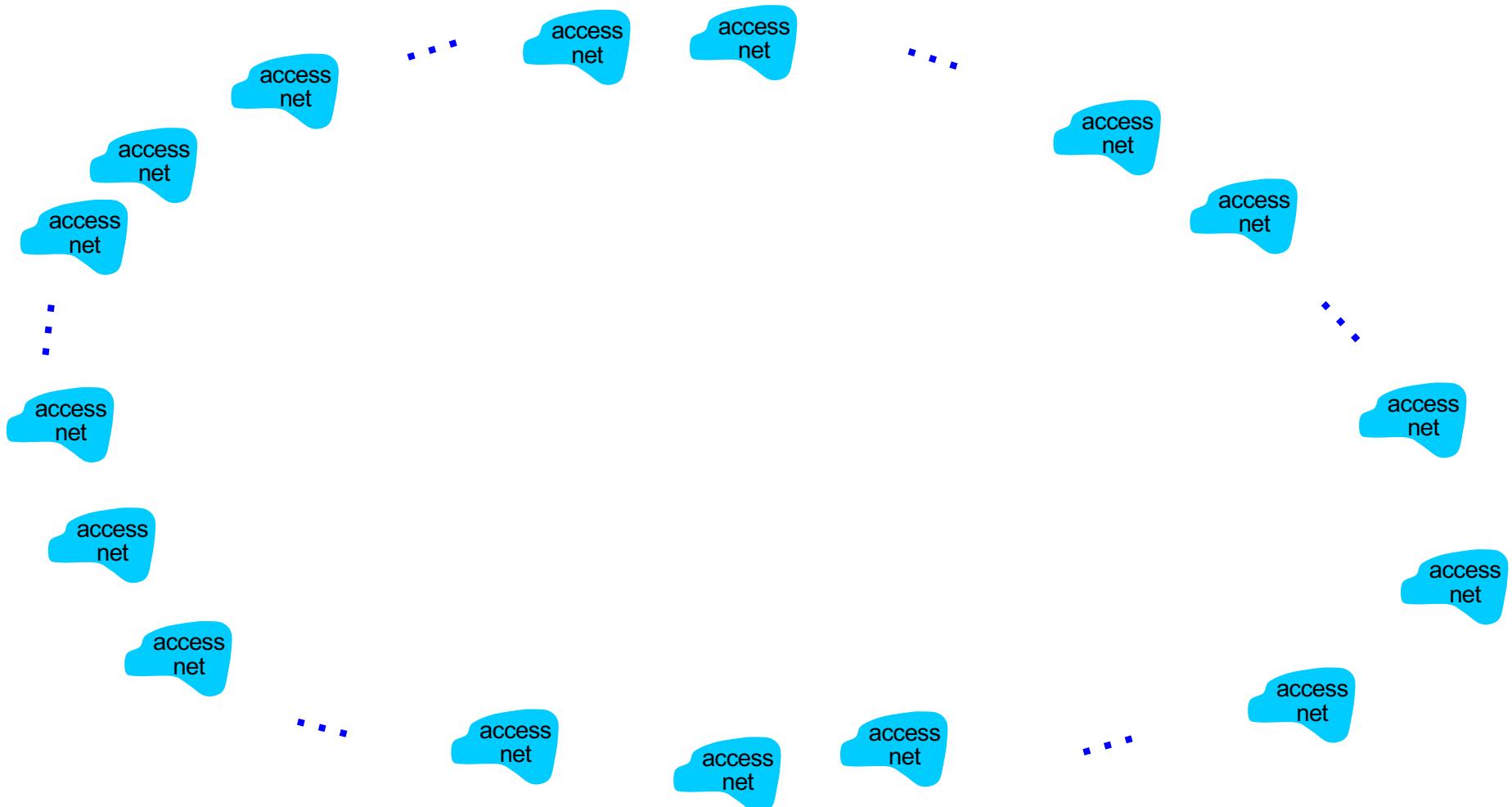
Open a browser and type: **[www.zeetings.com/salil](http://www.zeetings.com/salil)**

# Internet structure: network of networks

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

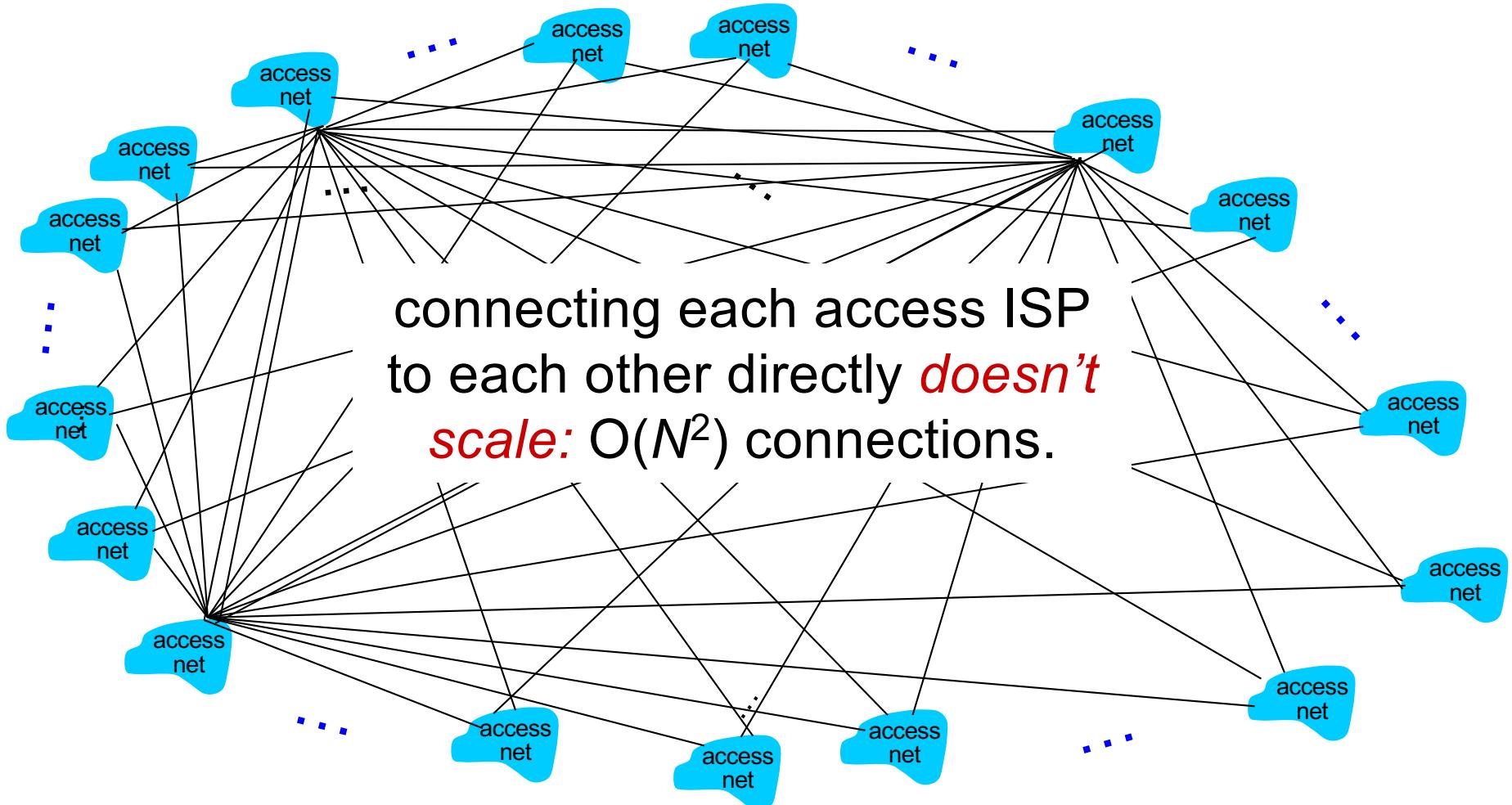
# Internet structure: network of networks

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



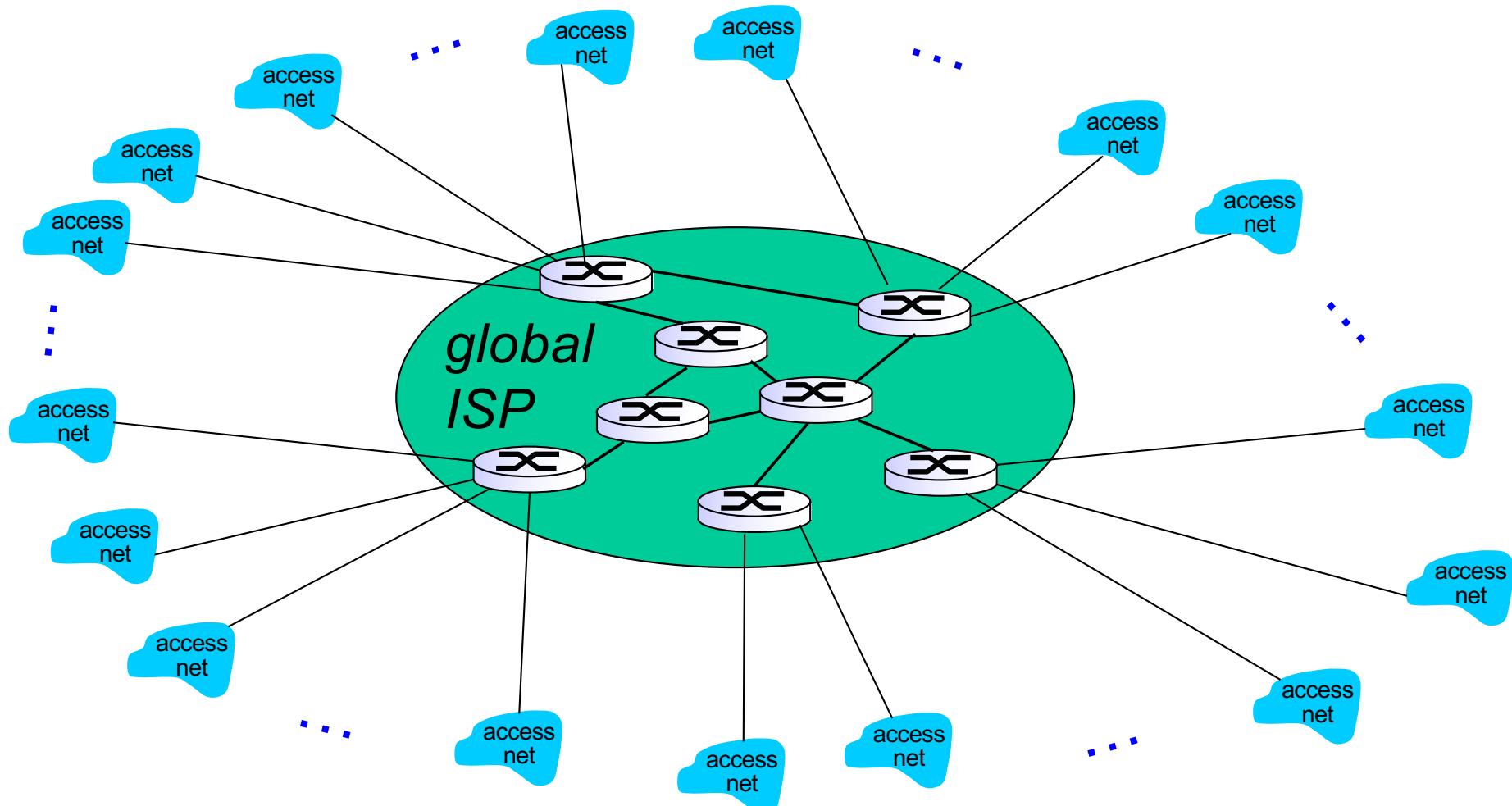
# Internet structure: network of networks

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



# Internet structure: network of networks

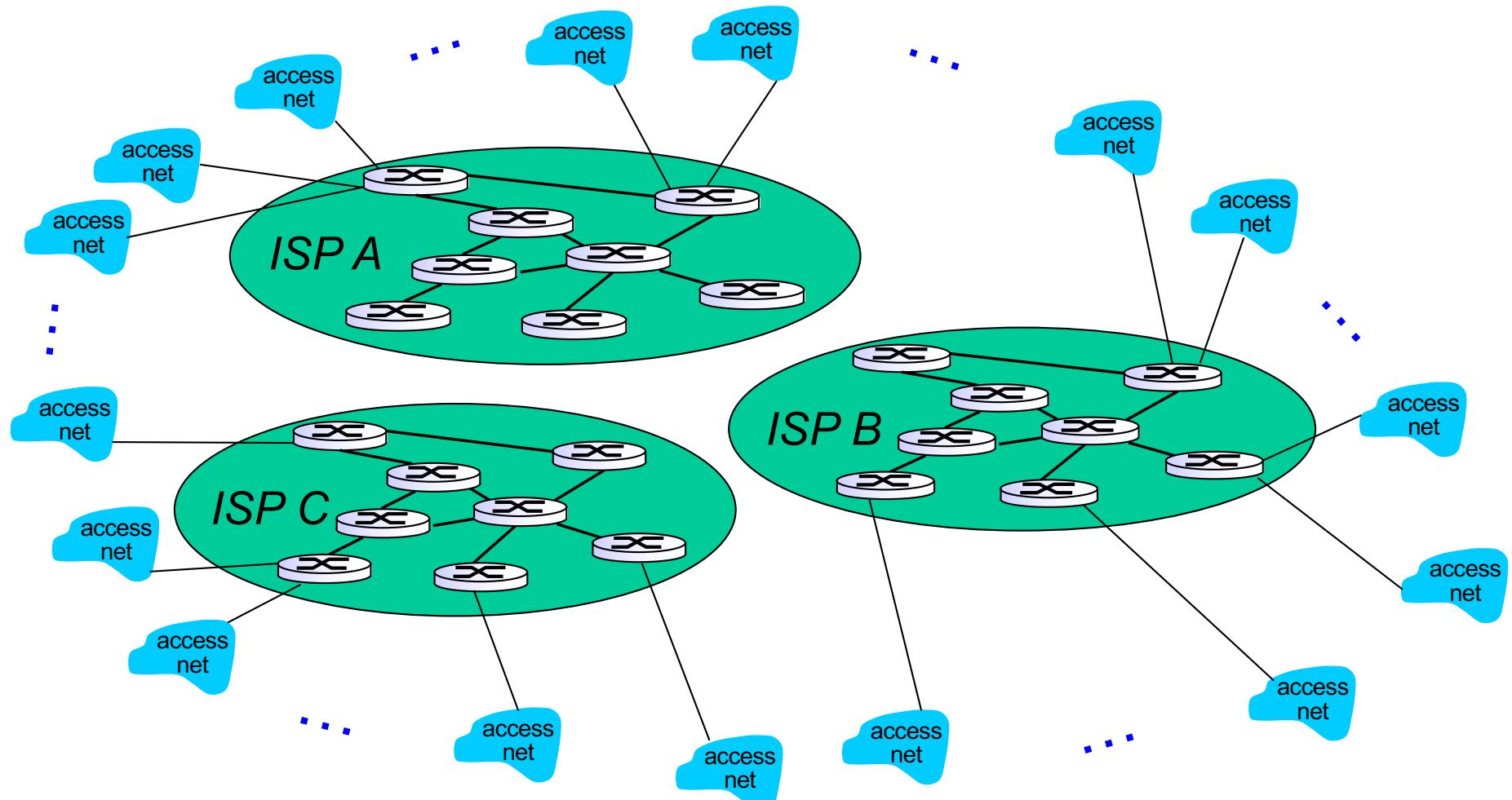
*Option: connect each access ISP to a global transit ISP? Customer and provider ISPs have economic agreement.*



# Internet structure: network of networks

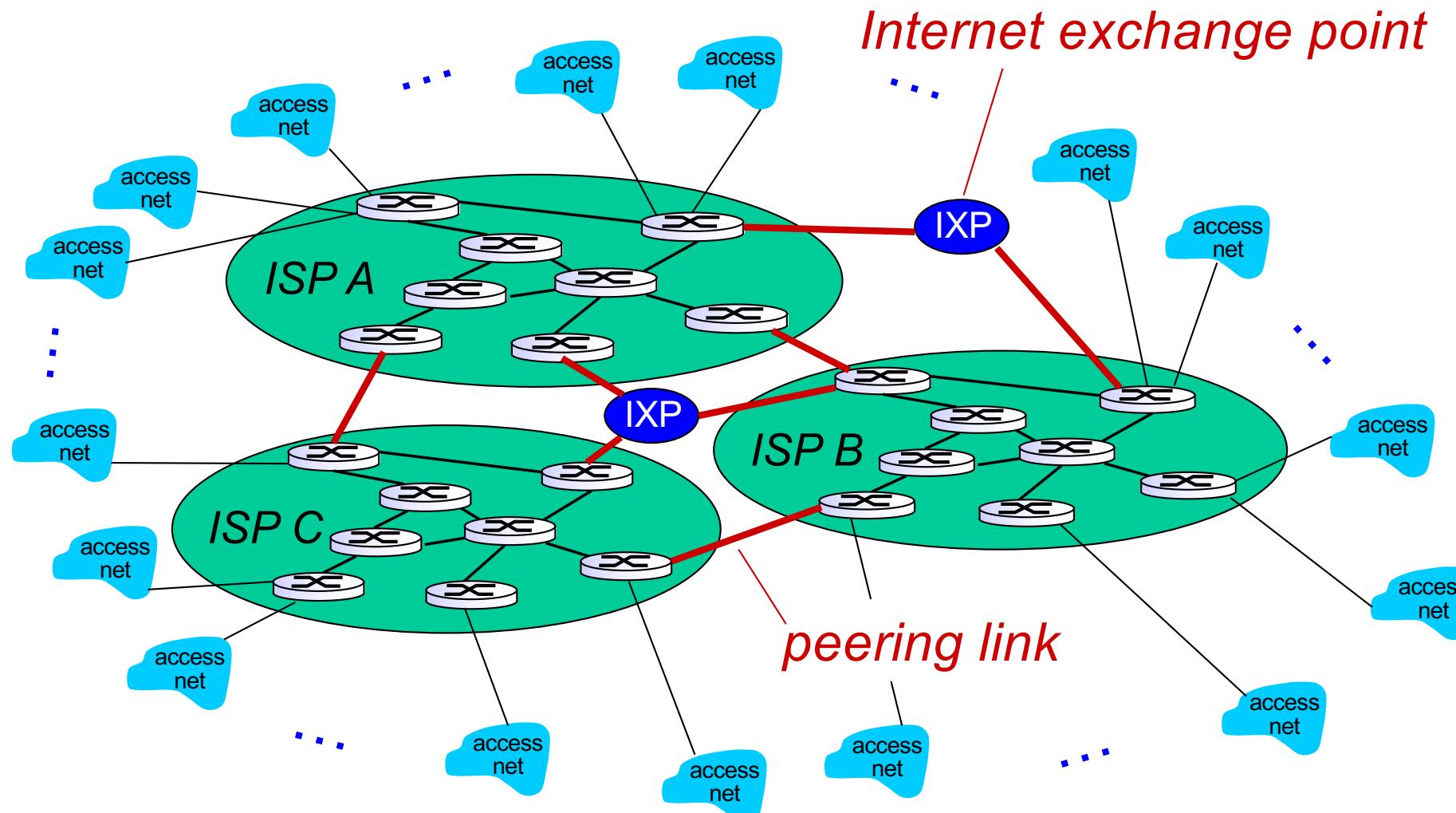
But if one global ISP is viable business, there will be competitors

....



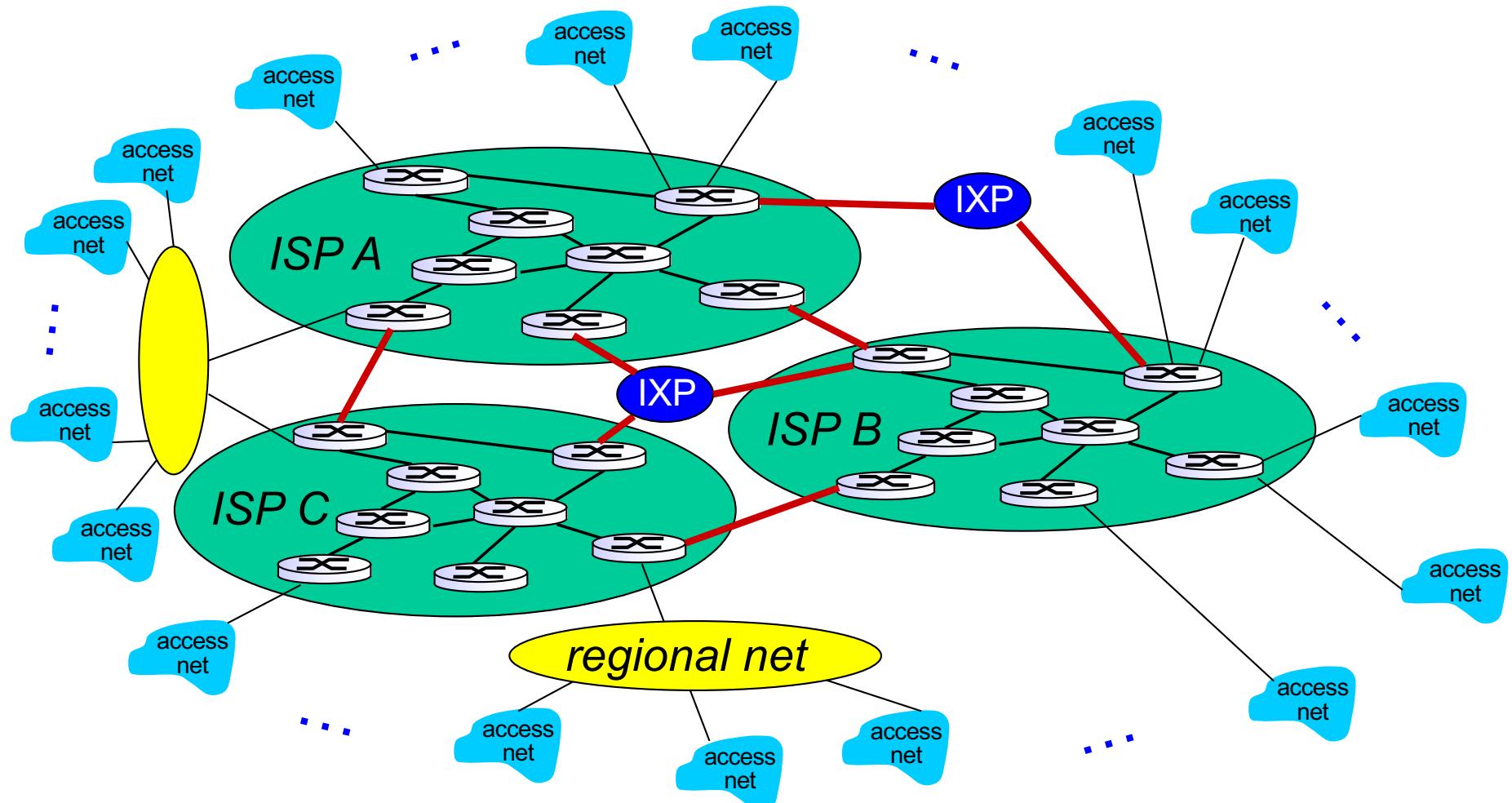
# Internet structure: network of networks

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



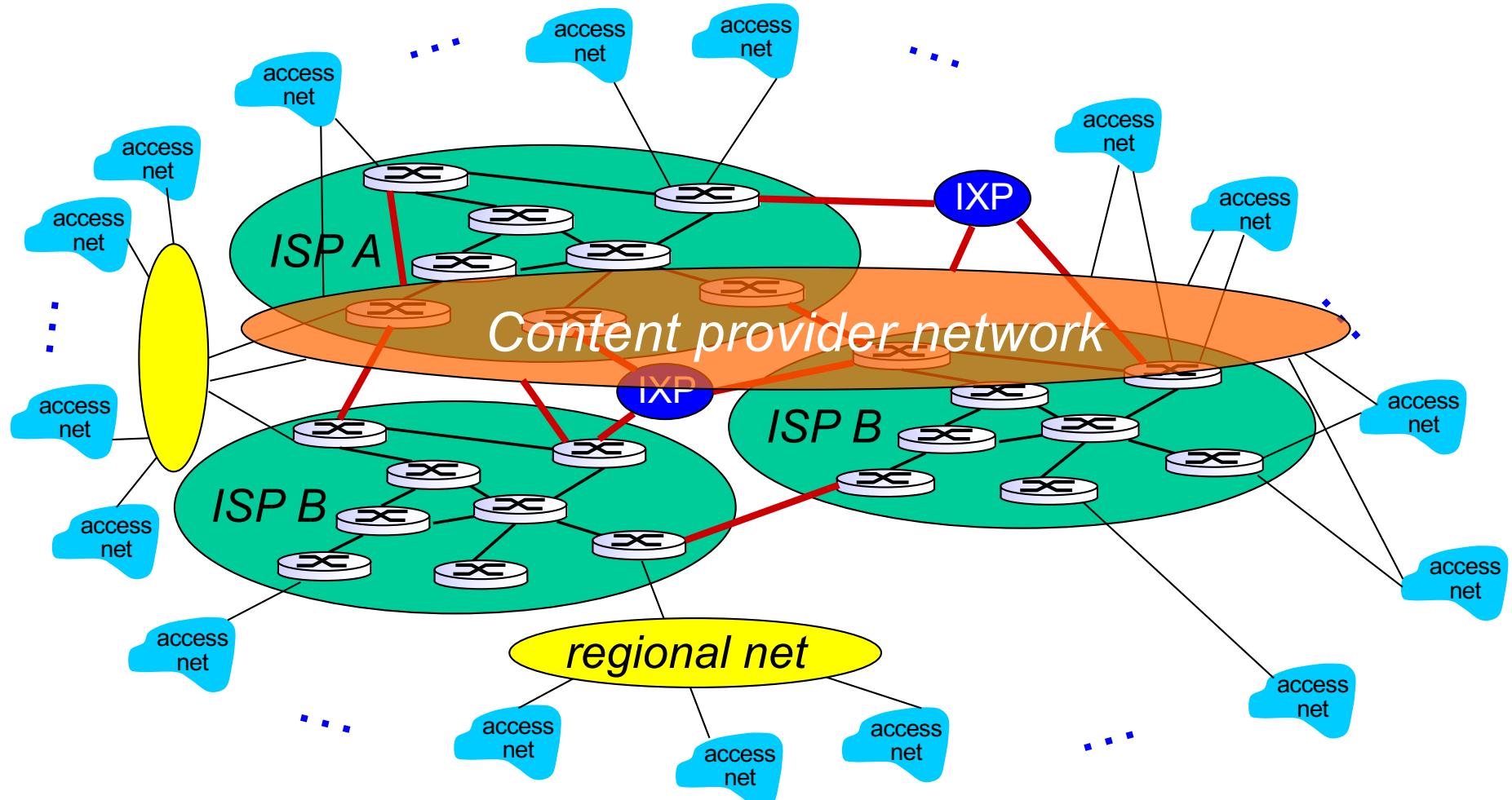
# Internet structure: network of networks

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

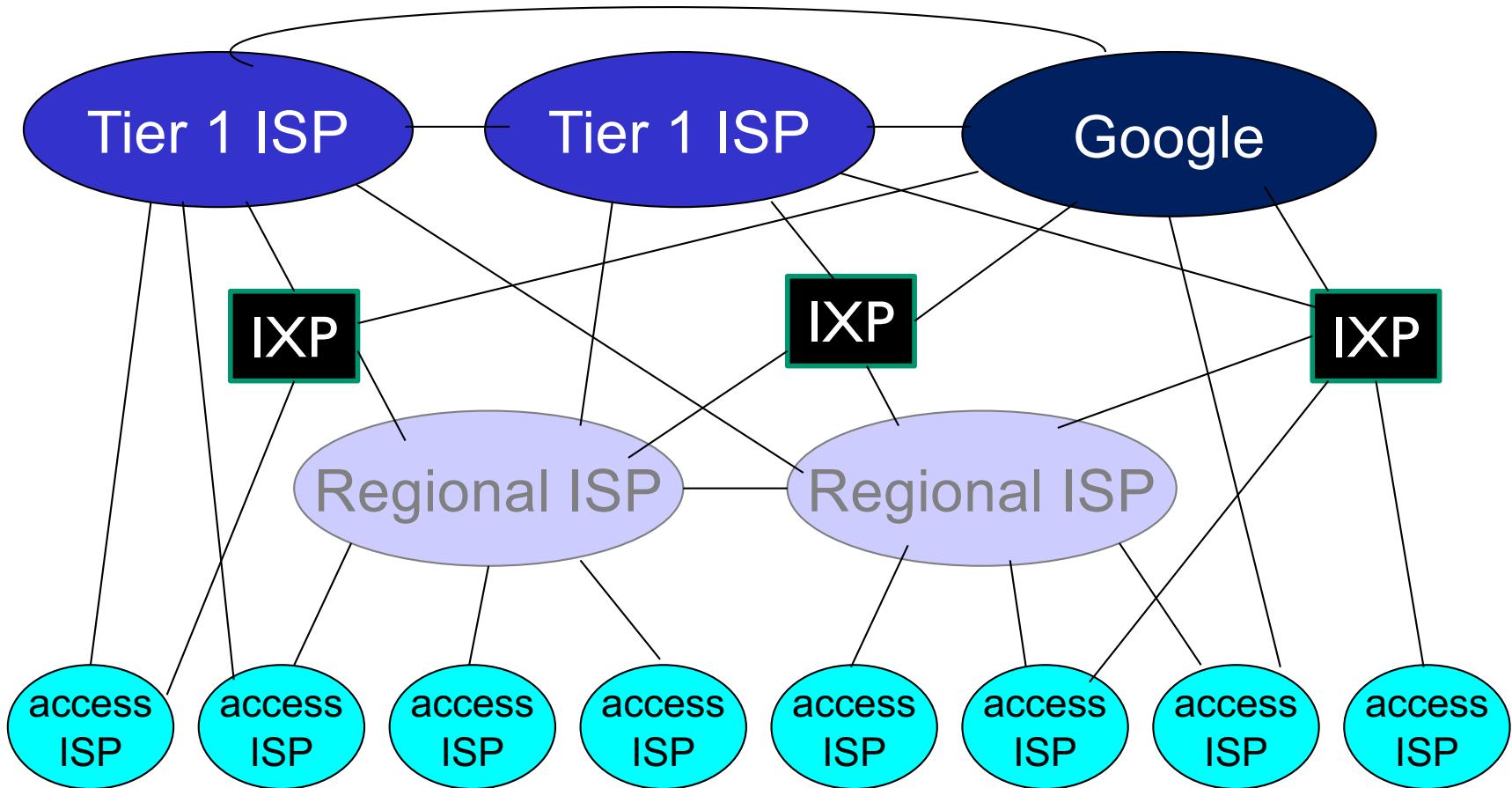


# Internet structure: network of networks

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



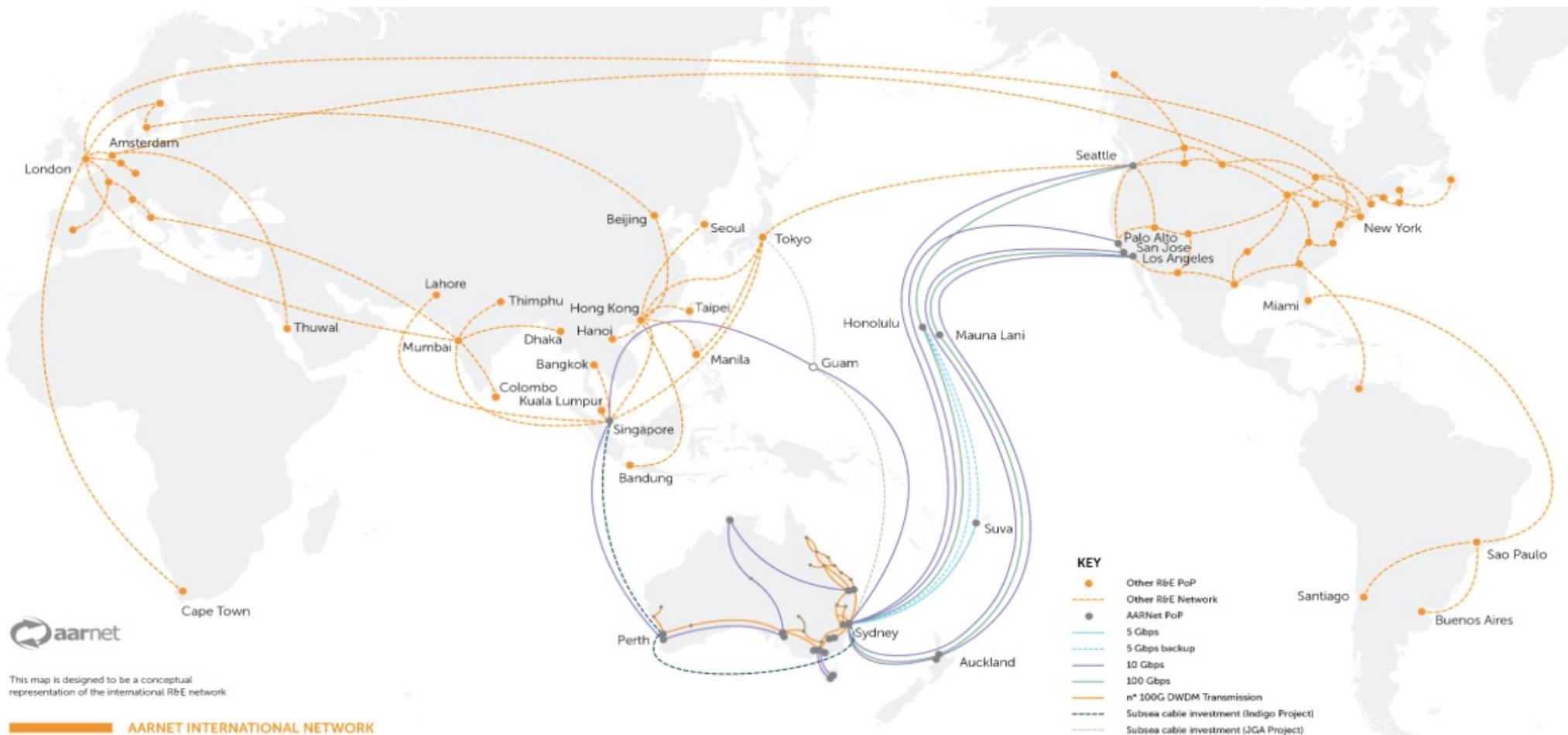
# Internet structure: network of networks



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

# AARNET: Australia's Academic and Research Network

- ❖ <https://www.aarnet.edu.au/>
- ❖ <https://www.submarinecablemap.com>



# I. Introduction: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

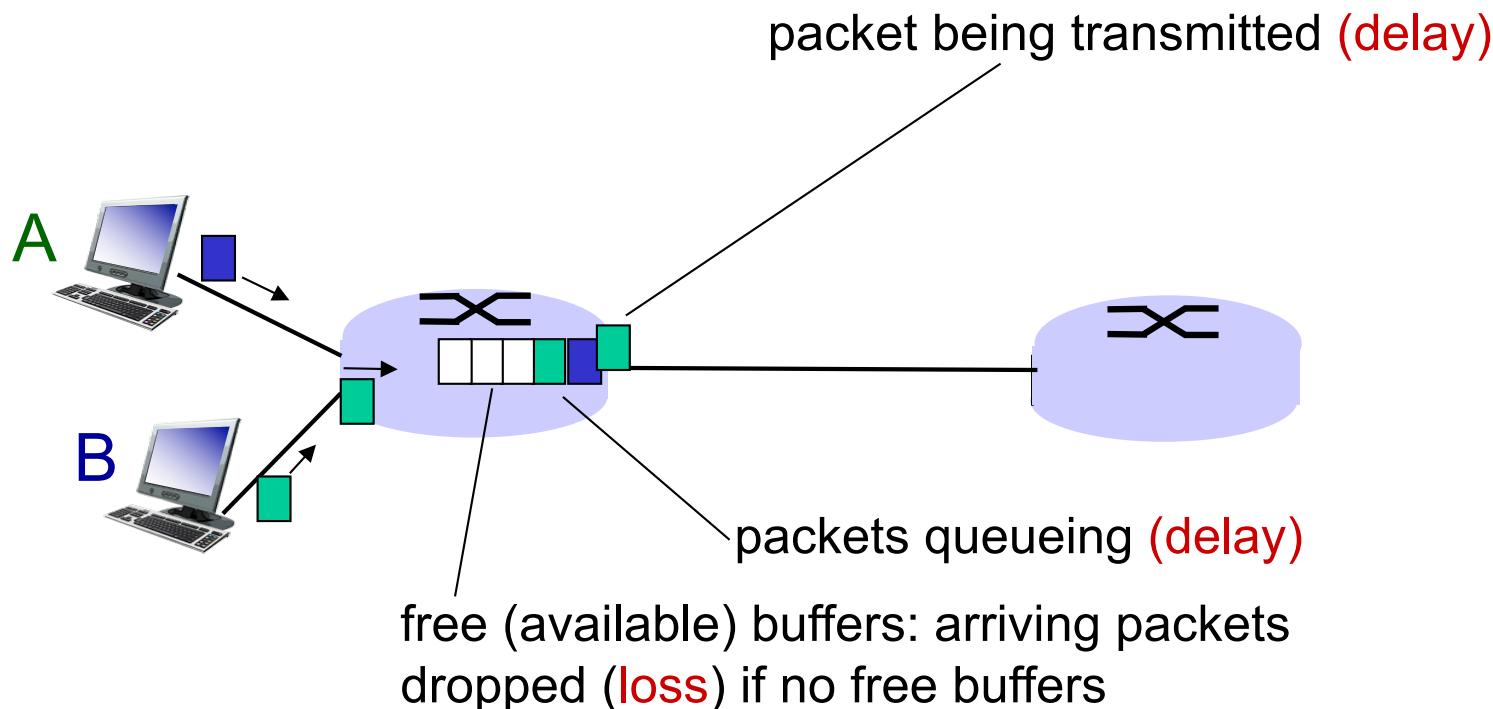
I.6 networks under attack: security

I.7 history

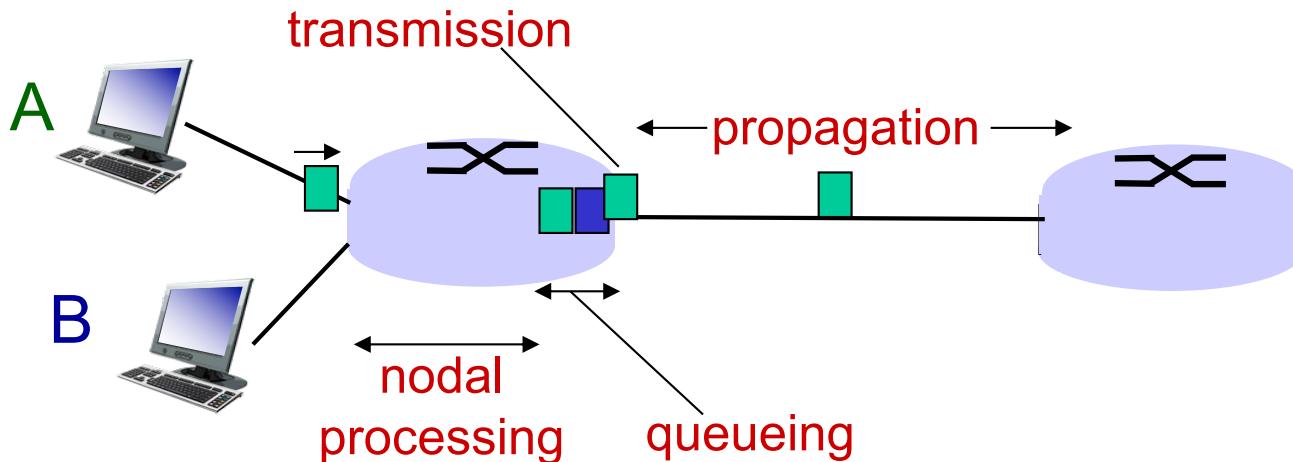
# How do loss and delay occur?

Packets queue in router buffers

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



# Four sources of packet delay



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

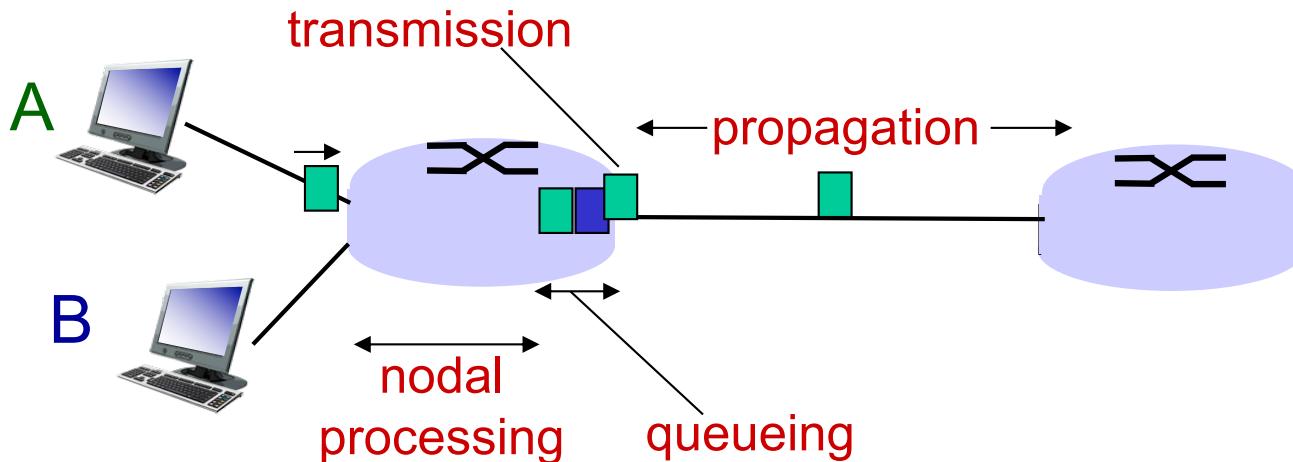
$d_{\text{proc}}$ : nodal processing

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

$d_{\text{queue}}$ : queueing delay

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

# Four sources of packet delay



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

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

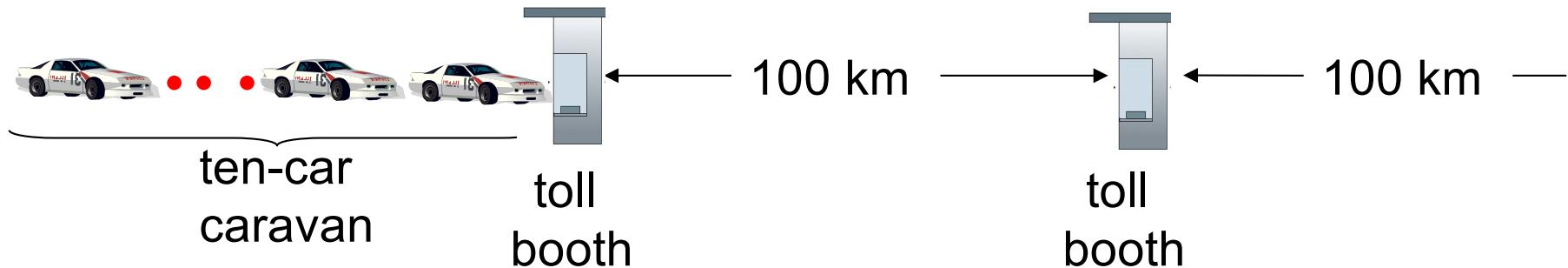
- $L$ : packet length (bits)
- $R$ : link bandwidth ( $\text{bps}$ )
- $d_{\text{trans}} = L/R$

$d_{\text{trans}}$  and  $d_{\text{prop}}$   
very different

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

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

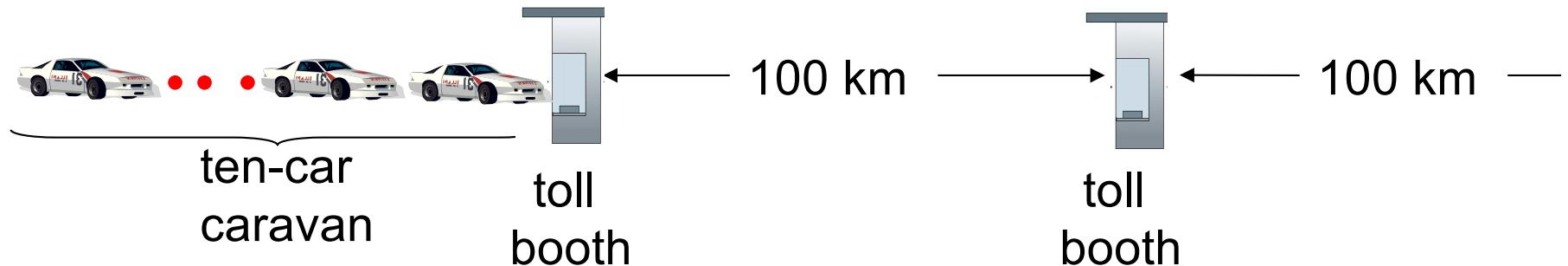
# Caravan analogy



- Car ~bit; Caravan ~ packet
- Cars “propagate” at 100 km/hr
- Toll booth takes 12 sec to service car (bit transmission time)
- Q: How long until caravan is lined up before 2nd toll booth?

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

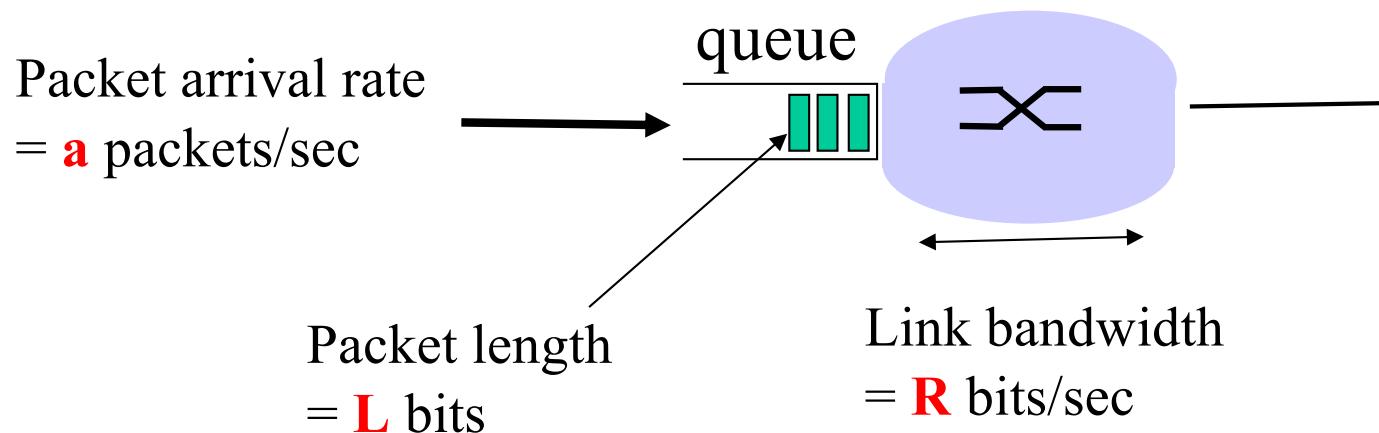
# Caravan analogy (more)



- Suppose cars now “propagate” at 1000 km/hr
- And suppose toll booth now takes one min to service a car
- **Q: Will cars arrive to 2nd booth before all cars serviced at first booth?**
  - **A: Yes!** after 7 min, 1st car arrives at second booth; three cars still at 1st booth.

Interactive Java Applet – Propagation vs transmission delay  
<https://www2.tkn.tu-berlin.de/teaching/rn/animations/propagation/>

# Queueing delay (more insight)

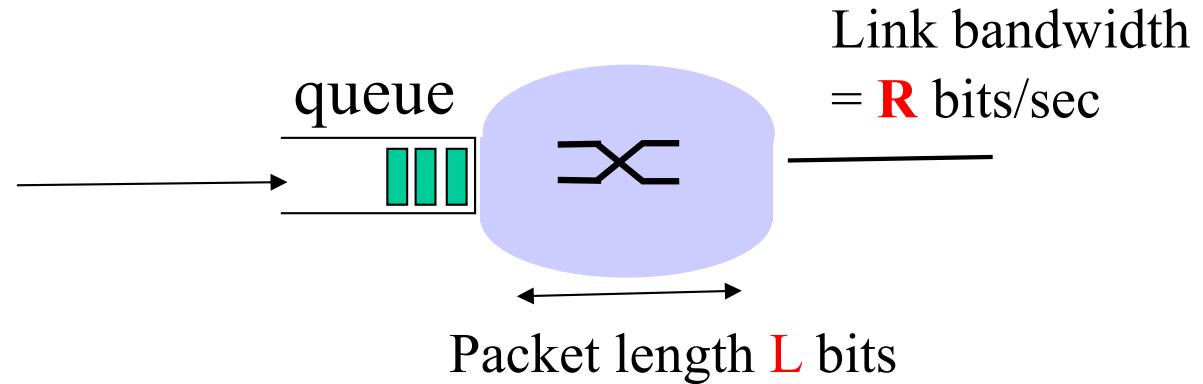


- ❖ Every second:  $aL$  bits arrive to queue
- ❖ Every second:  $R$  bits leave the router
- ❖ Question: what happens if  $aL > R$  ?
- ❖ Answer: queue will fill up, and packets will get dropped!!

$aL/R$  is called traffic intensity

# Queueing delay: illustration

1 packet arrives  
every  $L/R$  seconds



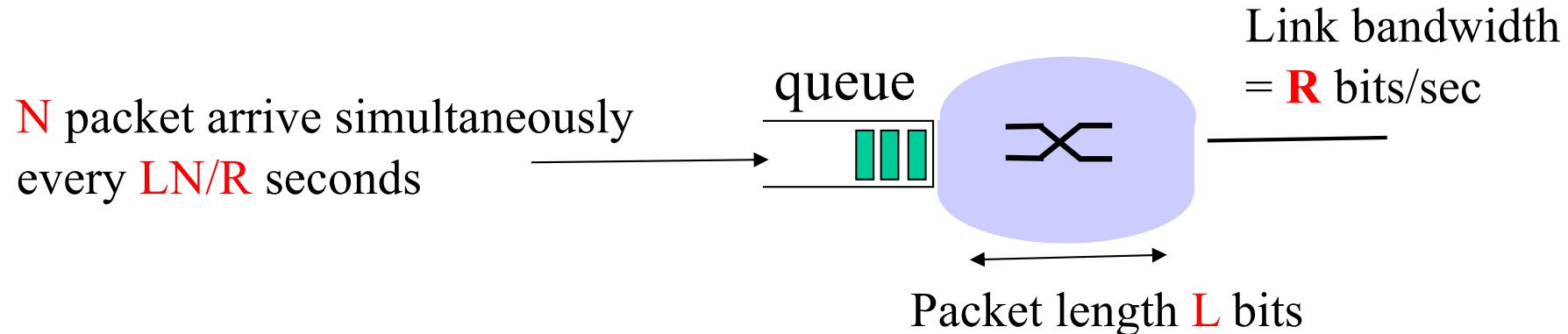
**Arrival rate:**  $a = 1/(L/R) = R/L$  (packet/second)



**Traffic intensity** =  $aL/R = (R/L)(L/R) = 1$

**Average queueing delay** = 0  
(queue is initially empty)

# Queueing delay: illustration



Arrival rate:  $a = N/(LN/R) = R/L$  packet/second

Traffic intensity =  $aL/R = (R/L)(L/R) = 1$

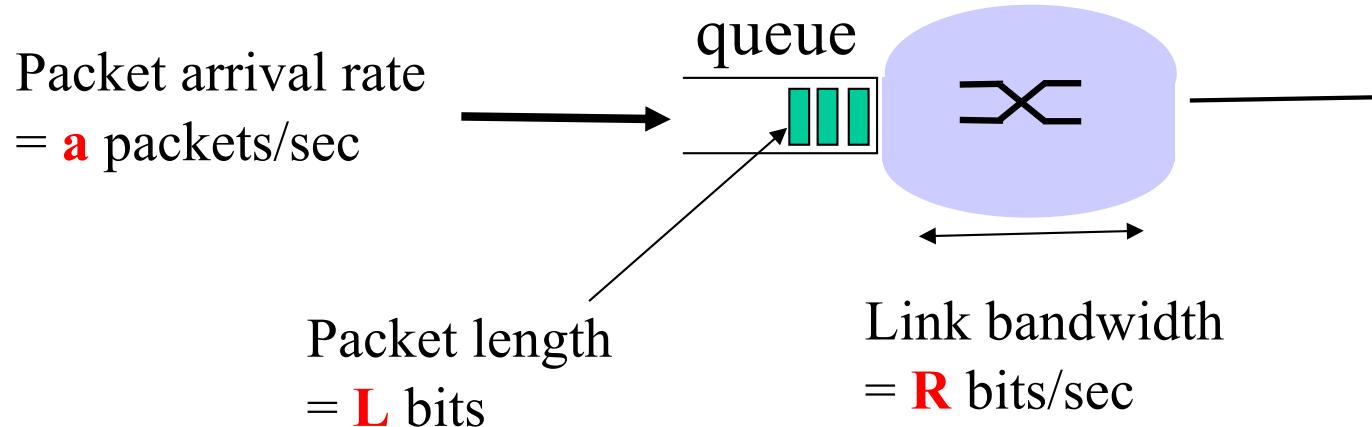


Average queueing delay (queue is empty at time 0) ?

$$\{0 + L/R + 2L/R + \dots + (N-1)L/R\}/N = L/(RN)\{1+2+\dots+(N-1)\} = L(N-1)/(2R)$$

Note: traffic intensity is same as previous scenario, but queueing delay is different

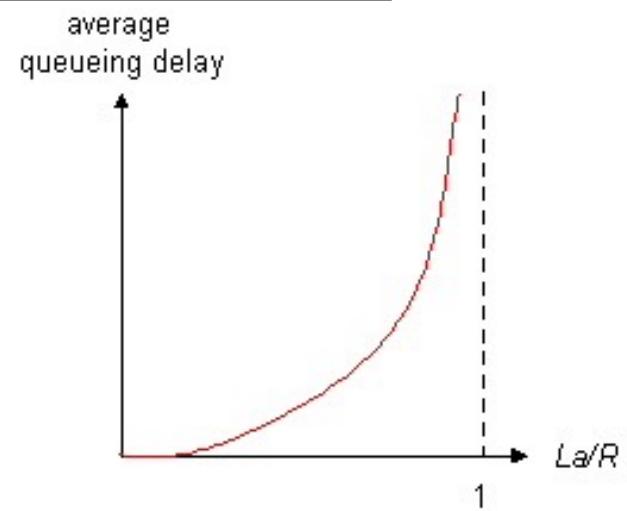
# Queueing delay: behaviour



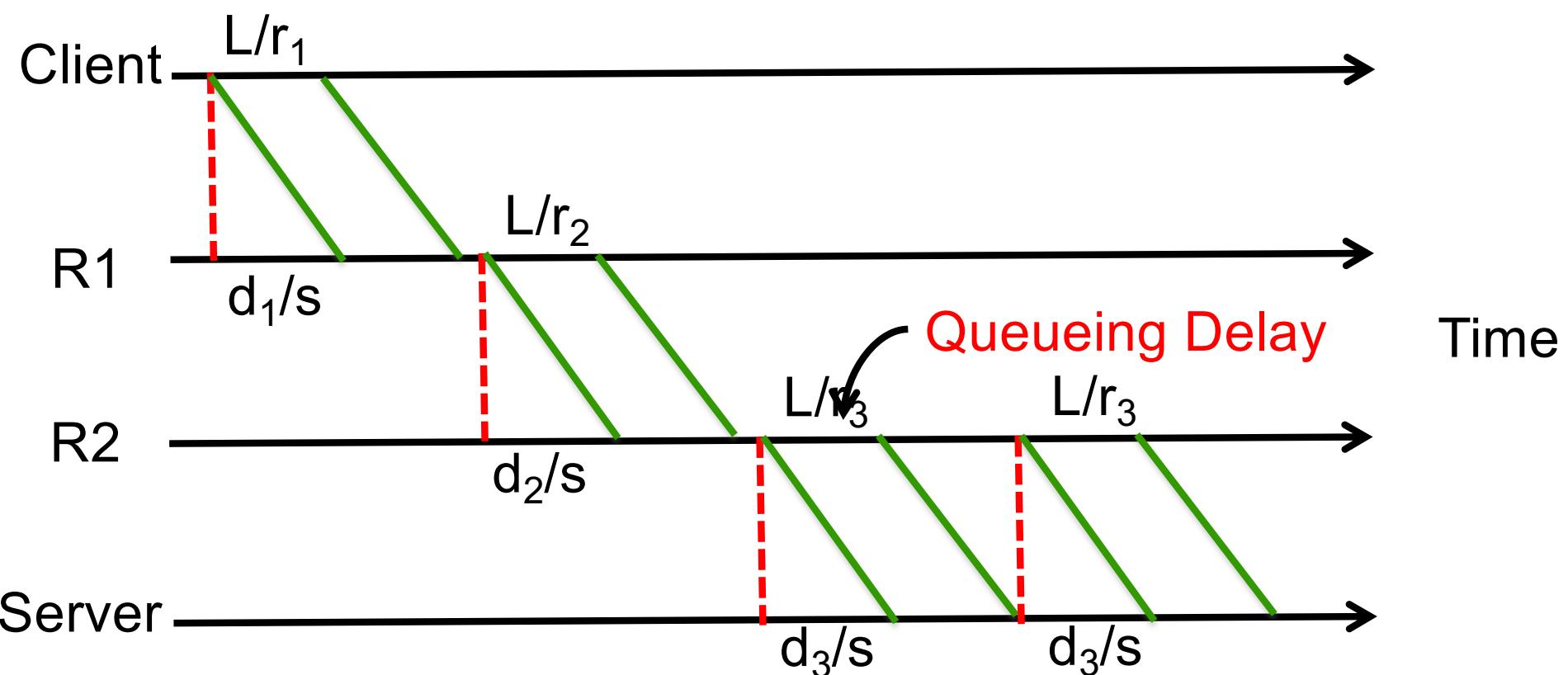
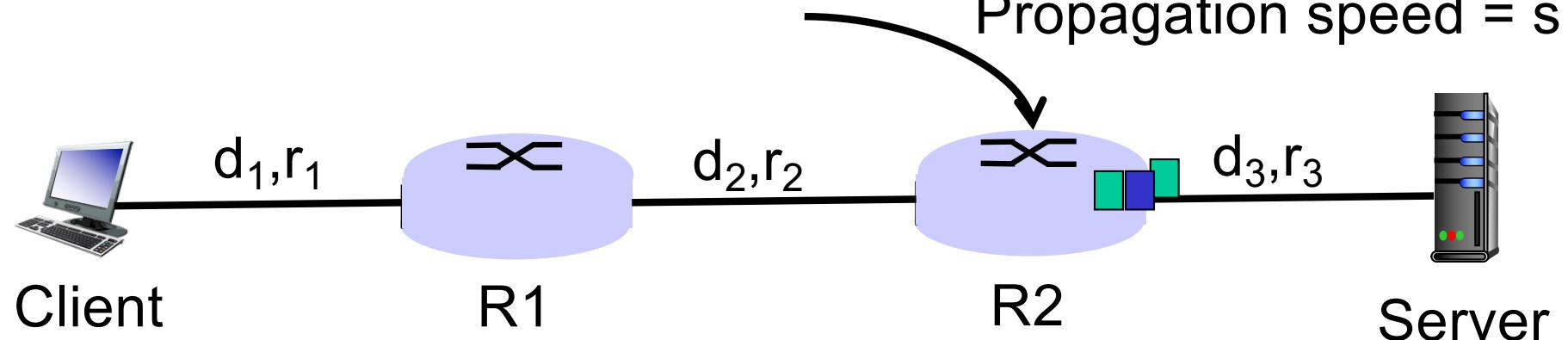
Interactive Java Applet:

<http://computerscience.unicam.it/marcantoni/reti/applet/QueuingAndLossInteractive/1.html>

- $La/R \sim 0$ : avg. queueing delay small
- $La/R \rightarrow 1$ : delays become large
- $La/R > 1$ : more “work” than can be serviced, average delay infinite!  
(this is when  $a$  is random!)



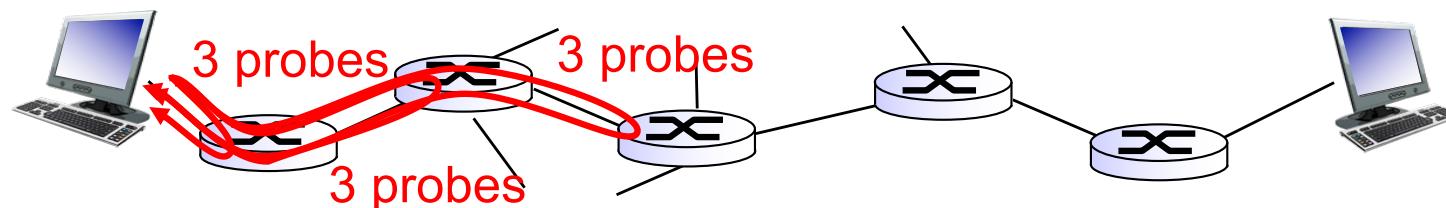
# End to End Delay



In the picture,  $r_1 = r_2 = r_3$ , you may wish to consider what happens when this is not the case <sup>96</sup>

# “Real” Internet delays and routes

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



# “Real” Internet delays, routes

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

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

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

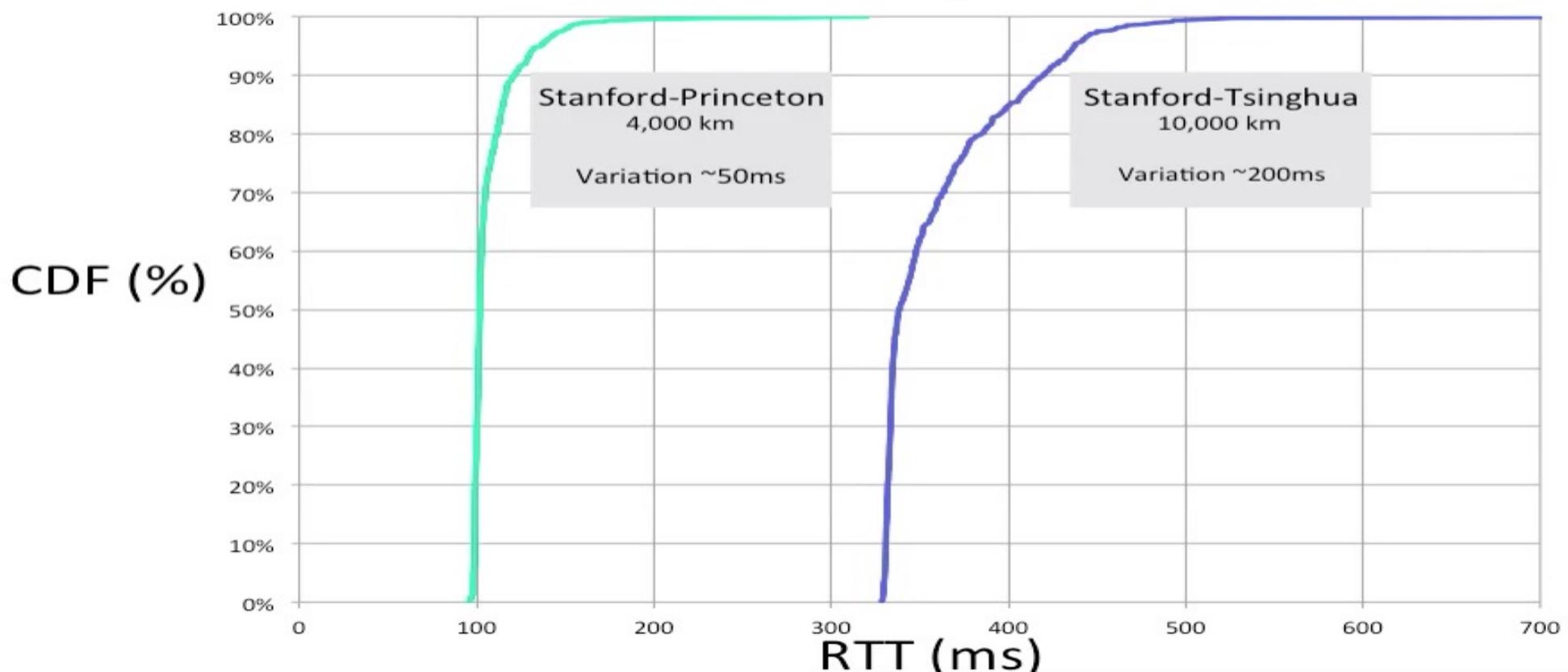
trans-oceanic link

\* Do some traceroutes from countries at [www.traceroute.org](http://www.traceroute.org)

# “Real” delay variations

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

*End-to-end delay = sum of all  $d_{\text{nodal}}$  along the path*



## **Quiz: Propagation Delay**



Propagation delay depends on the size of the packet

- A. True
- B. False

**Answer: B**

Open a browser and type: **[www.zeetings.com/salil](http://www.zeetings.com/salil)**



## Quiz: Oh these delays

Consider a packet that has just arrived at a router. What is the correct order of the delays encountered by the packet until it reaches the next-hop router?

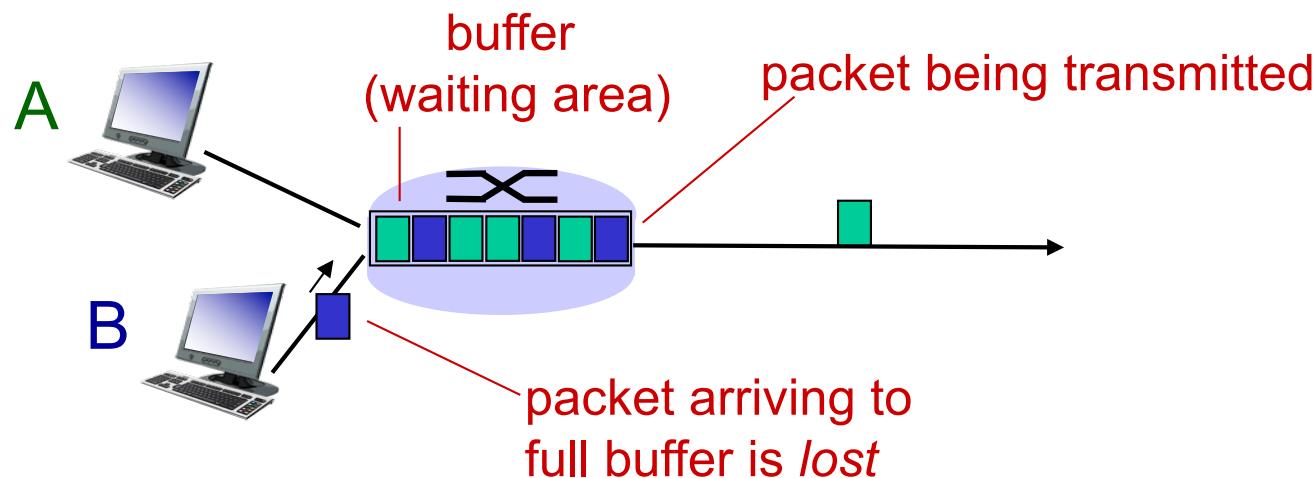
- A. Transmission, processing, propagation, queuing
- B. Propagation, processing, transmission, queuing
- C. Processing, queuing, transmission, propagation
- D. Queuing, processing, propagation, transmission

**Answer: C**

Open a browser and type: **[www.zeetings.com/salil](http://www.zeetings.com/salil)**

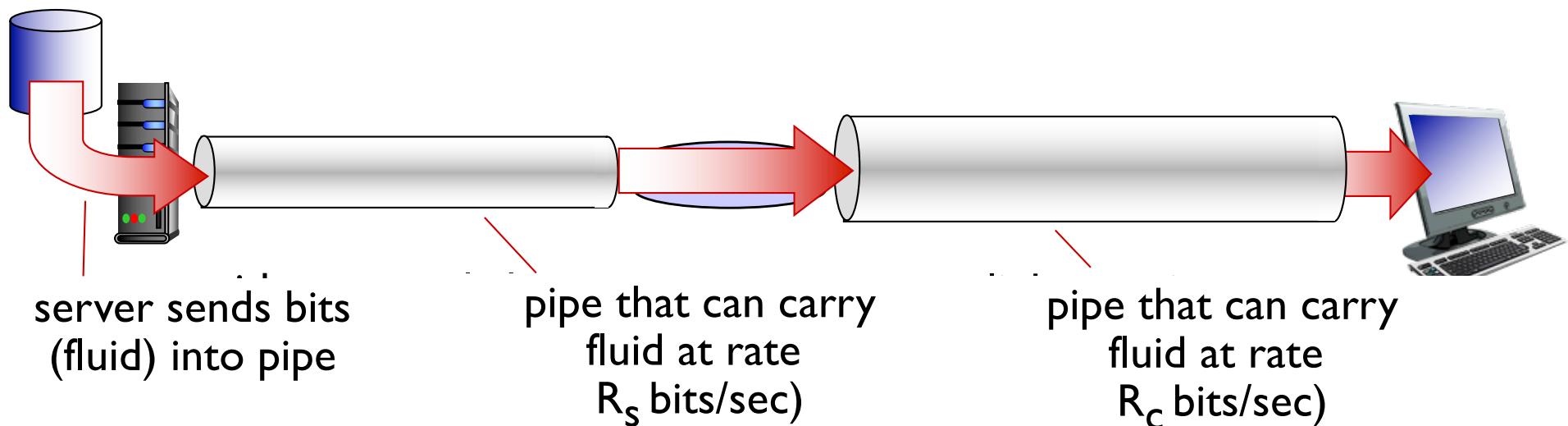
# Packet loss

- ❖ queue (aka buffer) preceding link in buffer has finite capacity
- ❖ packet arriving to full queue dropped (aka lost)
- ❖ lost packet may be retransmitted



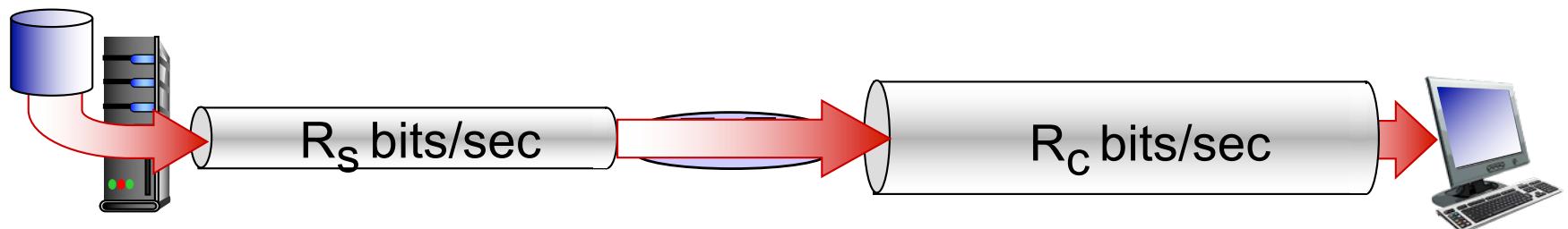
# Throughput

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

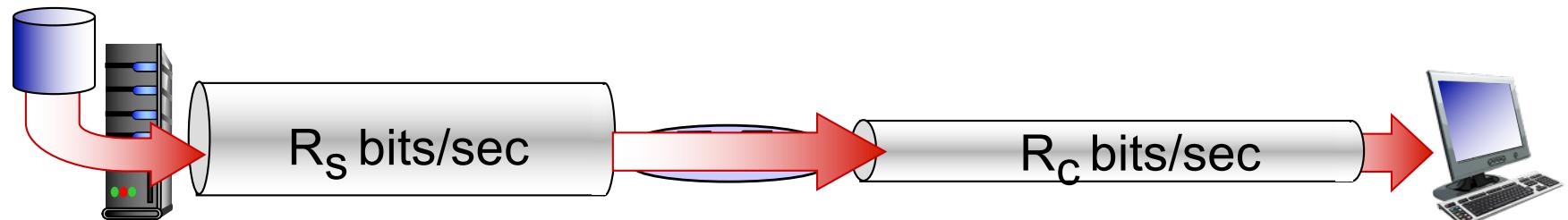


# Throughput (more)

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



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

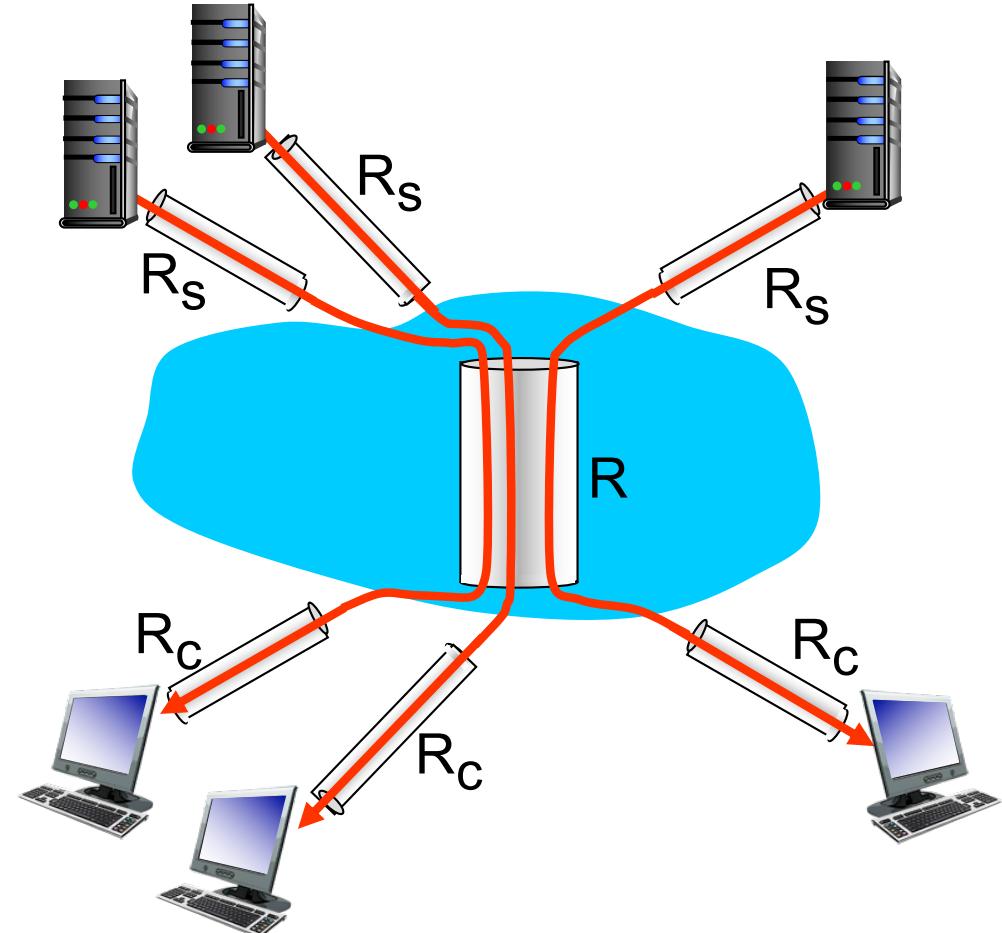


*bottleneck link*

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

# Throughput: Internet scenario

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



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

# 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
- ❖ **Next Week**
  - Protocol layers, service models
  - Application Layer

# Computer Networks and Applications

COMP 3331/COMP 9331

Week 2

Application Layer (Principles, Web,  
Email)

**Chapter 2, Sections 2.1-2.3**

## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

## 2. Application layer

### our goals:

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

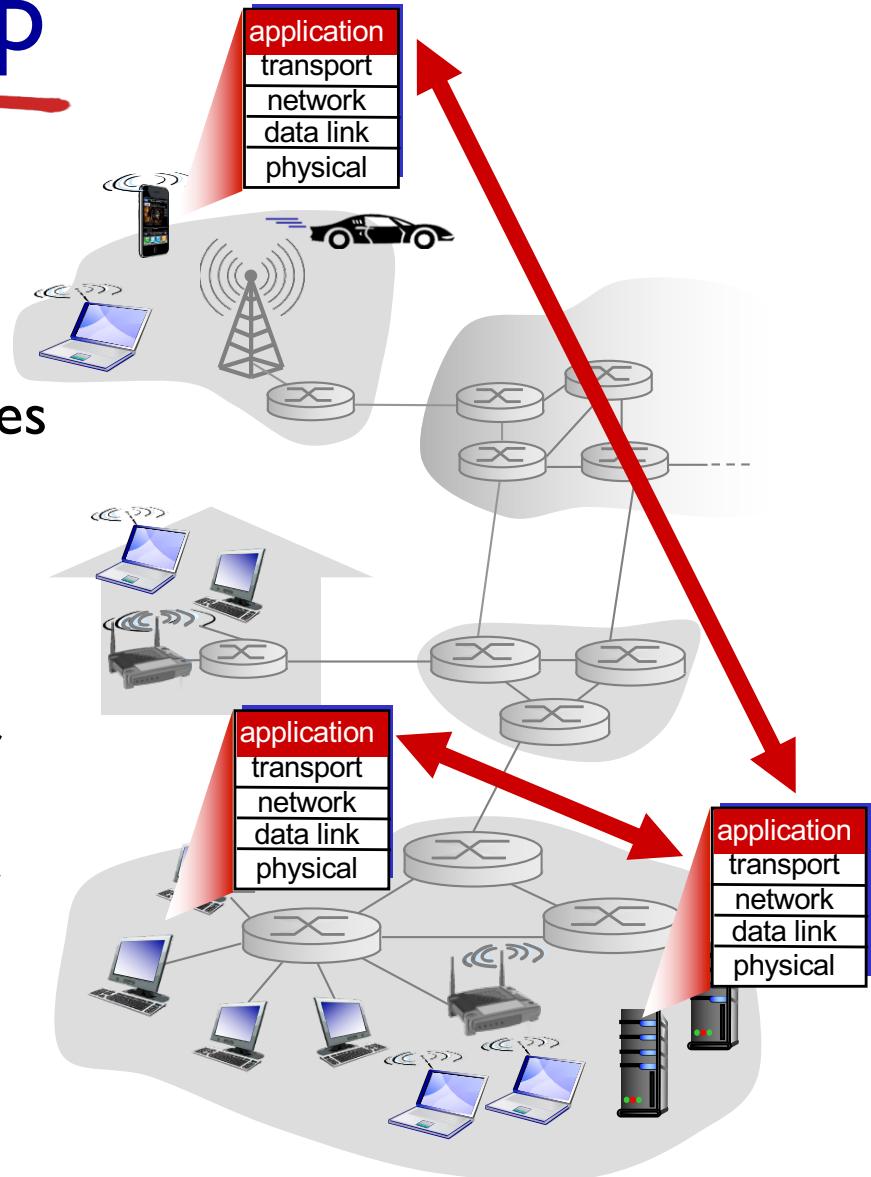
# Creating a network app

Write programs that:

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

No need to write software for network-core devices

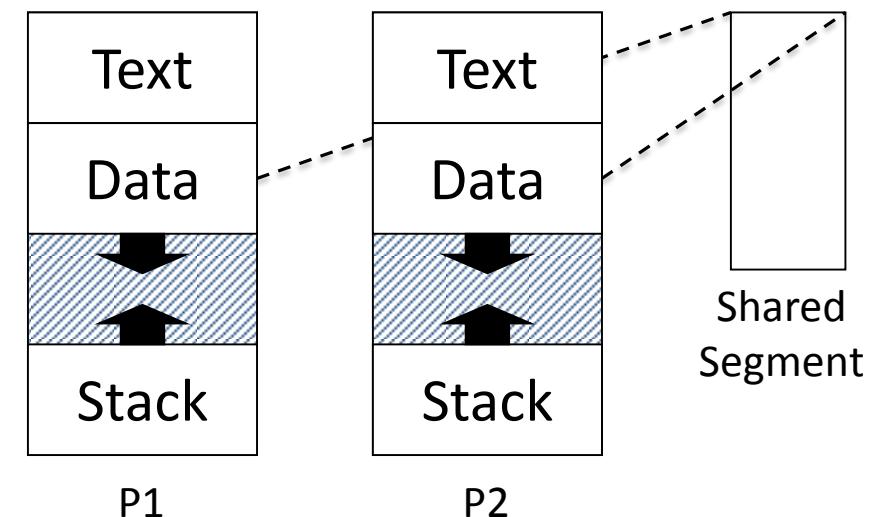
- ❖ network-core devices do not run user applications
- ❖ applications on end systems allows for rapid app development



# Interprocess Communication (IPC)

- ❖ Processes talk to each other through Inter-process communication (IPC)

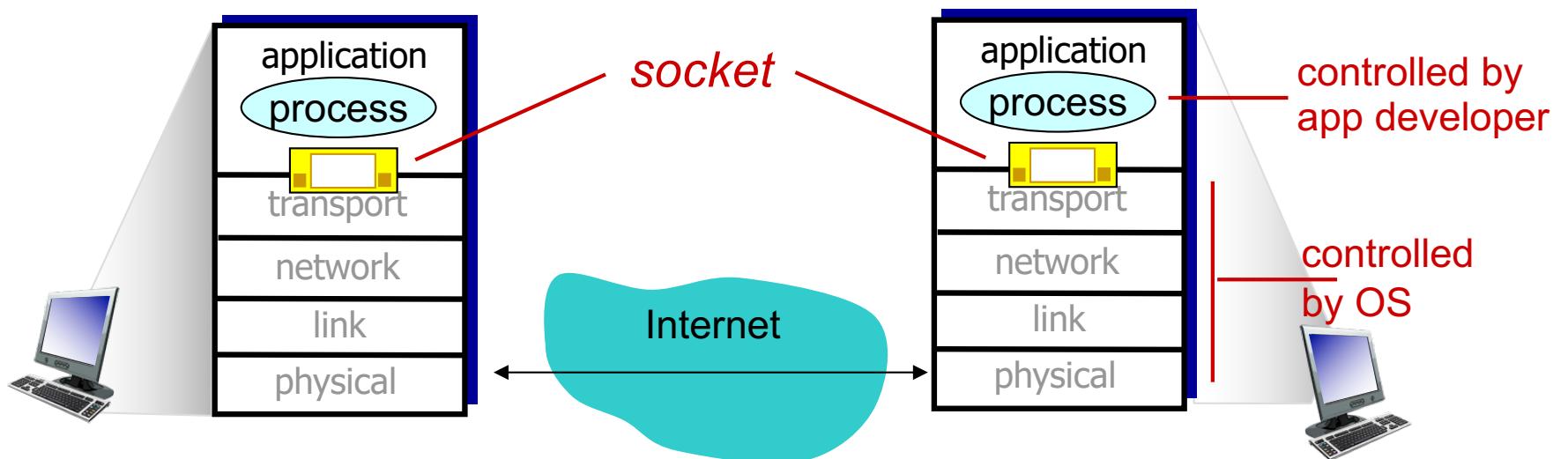
- ❖ On a single machine:
  - Shared memory



- ❖ Across machines:
  - We need other abstractions (message passing)

# Sockets

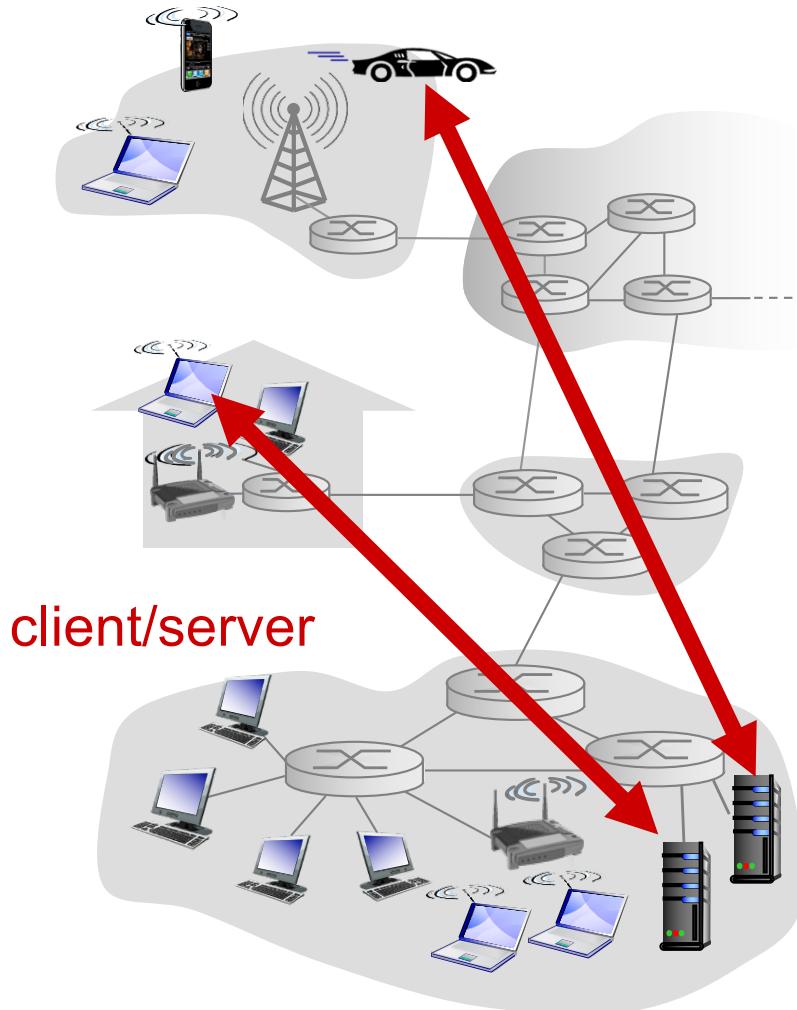
- ❖ process sends/receives messages to/from its **socket**
- ❖ socket analogous to door
  - sending process shoves message out through the door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
- ❖ Application has a few options, OS handles the details



# Addressing processes

- ❖ to receive messages, process must have *identifier*
- ❖ host device has unique 32-bit IP address
- ❖ Q: does IP address of host on which process runs suffice for identifying the process?
  - A: no, many processes can be running on same host
- ❖ *identifier* includes both IP address and port numbers associated with process on host.
- ❖ example port numbers:
  - HTTP server: 80
  - mail server: 25
- ❖ to send HTTP message to cse.unsw.edu.au web server:
  - IP address: 129.94.242.51
  - port number: 80

# Client-server architecture



## server:

- ❖ Exports well-defined request/response interface
- ❖ long-lived process that waits for requests
- ❖ Upon receiving request, carries it out

## clients:

- ❖ Short-lived process that makes requests
- ❖ “User-side” of application
- ❖ Initiates the communication

# Client versus Server

## ❖ Server

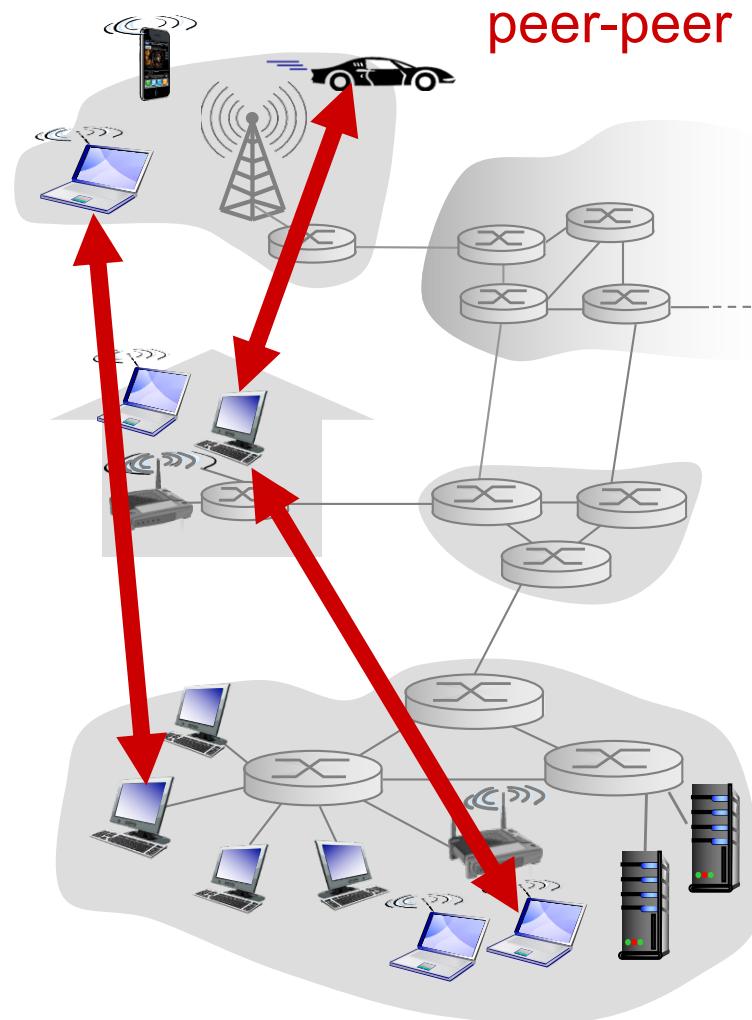
- Always-on host
- Permanent IP address (rendezvous location)
- Static port conventions (http: 80, email: 25, ssh:22)
- Data centres for scaling
- May communicate with other servers to respond

## ❖ Client

- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other

# P2P architecture

- ❖ no always-on server
  - No permanent rendezvous involved
- ❖ arbitrary end systems (peers) directly communicate
- ❖ Symmetric responsibility (unlike client/server)
- ❖ Often used for:
  - File sharing (BitTorrent)
  - Games
  - Blockchain and cryptocurrencies
  - Video distribution, video chat
  - In general: “distributed systems”



# P2P architecture: Pros and Cons

+ peers request service from other peers, provide service in return to other peers

- *self scalability* – new peers bring new service capacity, as well as new service demands

+ Speed: parallelism, less contention

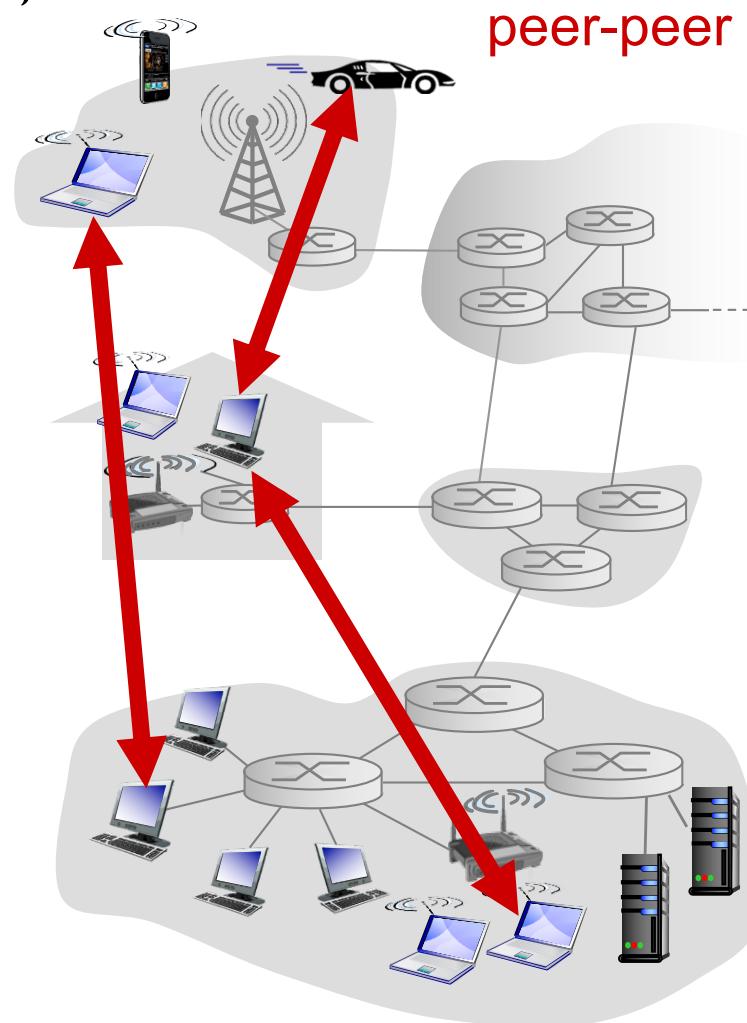
+ Reliability: redundancy, fault tolerance

+ Geographic distribution

-Fundamental problems of decentralized control

- State uncertainty: no shared memory or clock
- Action uncertainty: mutually conflicting decisions

-Distributed algorithms are complex



# App-layer protocol defines

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

**open protocols:**

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

**proprietary protocols:**

- ❖ e.g., Skype, Teams, Zoom

# What transport service does an app need?

## data integrity

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

## timing

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

## throughput

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

## security

- ❖ encryption, data integrity,

...

# Transport service requirements: common apps

<b>application</b>	<b>data loss</b>	<b>throughput</b>	<b>time sensitive</b>
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 50kbps-1Mbps video:100kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
Chat/messaging	no loss	elastic	yes and no

# Internet transport protocols services

## TCP service:

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

## UDP service:

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

Q: why bother? Why is there a UDP?

**NOTE:** More on transport later on

# Internet apps: application, transport protocols

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



## Quiz: Transport

Pick the true statement

- A. TCP provides reliability and guarantees a minimum bandwidth
- B. TCP provides reliability while UDP provides bandwidth guarantees
- C. TCP provides reliability while UDP does not
- D. Neither TCP nor UDP provides reliability

**Answer: C**

Open a browser and type: **[www.zeetings.com/salil](http://www.zeetings.com/salil)**

## 2. Application Layer: outline

### 2.1 principles of network applications

- app architectures
- app requirements

### 2.2 Web and HTTP

### 2.3 electronic mail

- SMTP

### 2.4 DNS

### 2.5 P2P applications

### 2.6 video streaming and content distribution networks (CDNs)

### 2.7 socket programming with UDP and TCP

# The Web – Precursor



Ted Nelson

- ❖ **1967, Ted Nelson, Xanadu:**
  - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
  - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- ❖ **Coined the term “Hypertext”**

# The Web – History

---



Tim Berners-Lee

- ❖ World Wide Web (WWW): a distributed database of “pages” linked through **Hypertext Transport Protocol (HTTP)**
  - First HTTP implementation - 1990
    - Tim Berners-Lee at CERN
  - HTTP/0.9 – 1991
    - Simple GET command for the Web
  - HTTP/1.0 – 1992
    - Client/Server information, simple caching
  - HTTP/1.1 – 1996
  - HTTP2.0 - 2015

<http://info.cern.ch/hypertext/WWW/TheProject.html>

# 2020 This Is What Happens In An Internet Minute



# Web and HTTP

*First, a review...*

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

www.someschool.edu/someDept/pic.gif

host name

path name

# Web and HTTP

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hyperlink Example</title>
    </head>
    <body>
        <p>Click the following link</p>
        <a href = "http://www.cnn.com" target ="_self">CNN</a>
    </body>
</html>
```

# Uniform Resource Locator (URL)

---

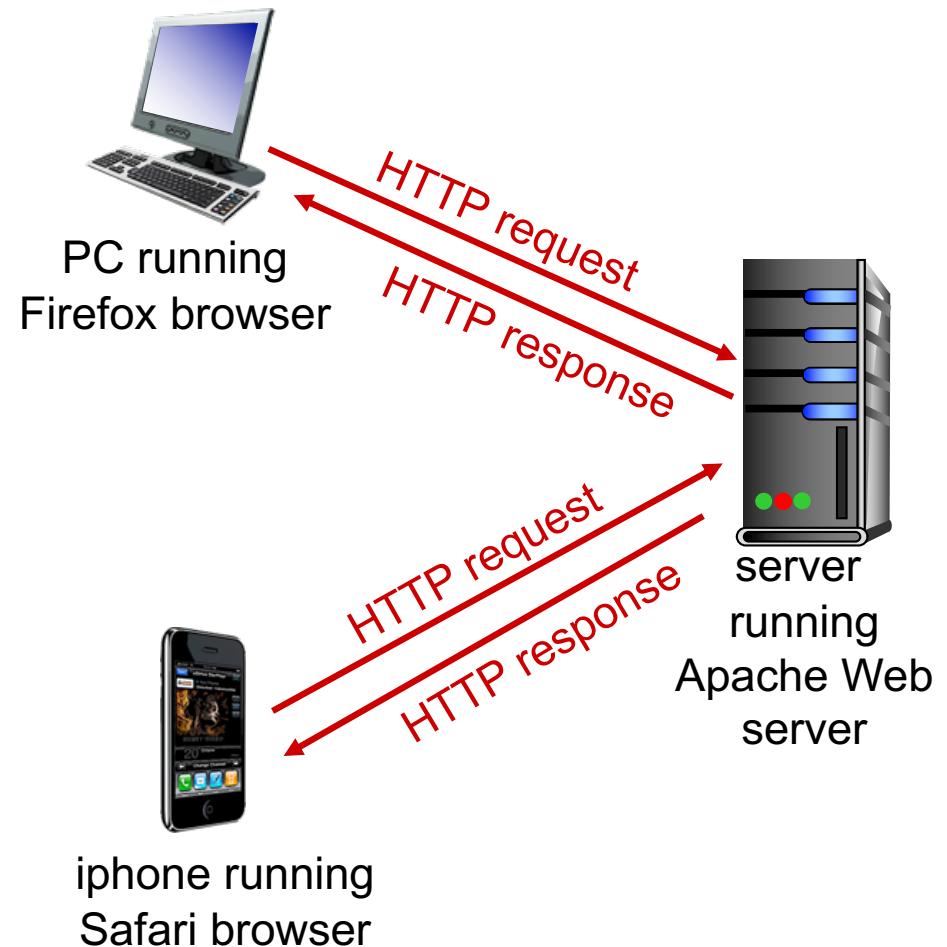
**protocol://host-name[:port]/directory-path/resource**

- ❖ *protocol*: http, ftp, https, smtp etc.
- ❖ *hostname*: DNS name, IP address
- ❖ *port*: defaults to protocol's standard port; e.g. http: 80 https: 443
- ❖ *directory path*: hierarchical, reflecting file system
- ❖ *resource*: Identifies the desired resource

# HTTP overview

## HTTP: hypertext transfer protocol

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



# HTTP overview (continued)

## *uses TCP:*

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

## *HTTP is “stateless”*

- ❖ server maintains no information about past client requests

protocols that maintain “state” are complex!

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

*aside*

# HTTP request message

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

request line  
(GET, POST,  
HEAD commands)

header  
lines

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

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

carriage return character

line-feed character

# HTTP response message

status line  
(protocol  
status code  
status phrase)

header  
lines

data, e.g.,  
requested  
HTML file

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

# HTTP response status codes

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

## **200 OK**

- request succeeded, requested object later in this msg

## **301 Moved Permanently**

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

## **400 Bad Request**

- request msg not understood by server

## **404 Not Found**

- requested document not found on this server

## **505 HTTP Version Not Supported**

## **451 Unavailable for Legal Reasons**

## **429 Too Many Requests**

## **418 I'm a Teapot**

# HTTP is all text

- ❖ Makes the protocol simple
  - Easy to delineate messages (\r\n)
  - (relatively) human-readable
  - No issues about encoding or formatting data
  - Variable length data
- ❖ Not the most efficient
  - Many protocols use binary fields
    - Sending "12345678" as a string is 8 bytes
    - As an integer, 12345678 needs only 4 bytes
  - Headers may come in any order
  - Requires string parsing/processing
- ❖ Non-text content needs to be encoded

# Request Method types (“verbs”)

## HTTP/1.0:

- ❖ GET
  - Request page
- ❖ POST
  - Uploads user response to a form
- ❖ HEAD
  - asks server to leave requested object out of response

## HTTP/1.1:

- ❖ GET, POST, HEAD
- ❖ PUT
  - uploads file in entity body to path specified in URL field
- ❖ DELETE
  - deletes file specified in the URL field
- ❖ TRACE, OPTIONS, CONNECT, PATCH
  - For persistent connections

# Uploading form input

## POST method:

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

## Get (in-URL) method:

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

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

# User-server state: cookies

many Web sites use cookies

*four components:*

- 1) cookie header line of HTTP *response* message
- 2) cookie header line in next HTTP *request* message
- 3) cookie file kept on user's host, managed by user's browser
- 4) back-end database at Web site

*example:*

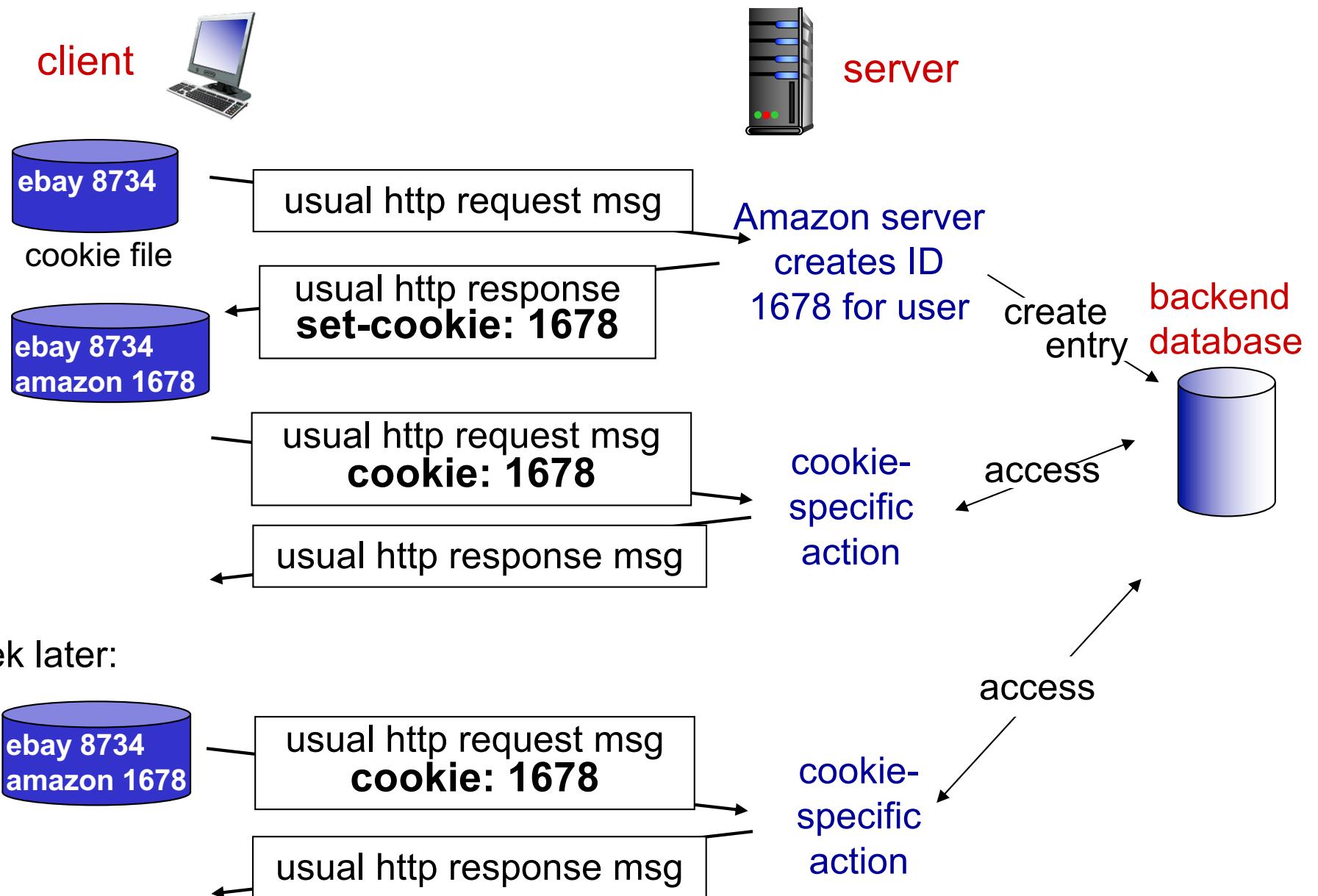
- ❖ Susan always access Internet from PC
- ❖ visits specific e-commerce site for first time
- ❖ when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

HTTP is stateless

It does not save the clients request



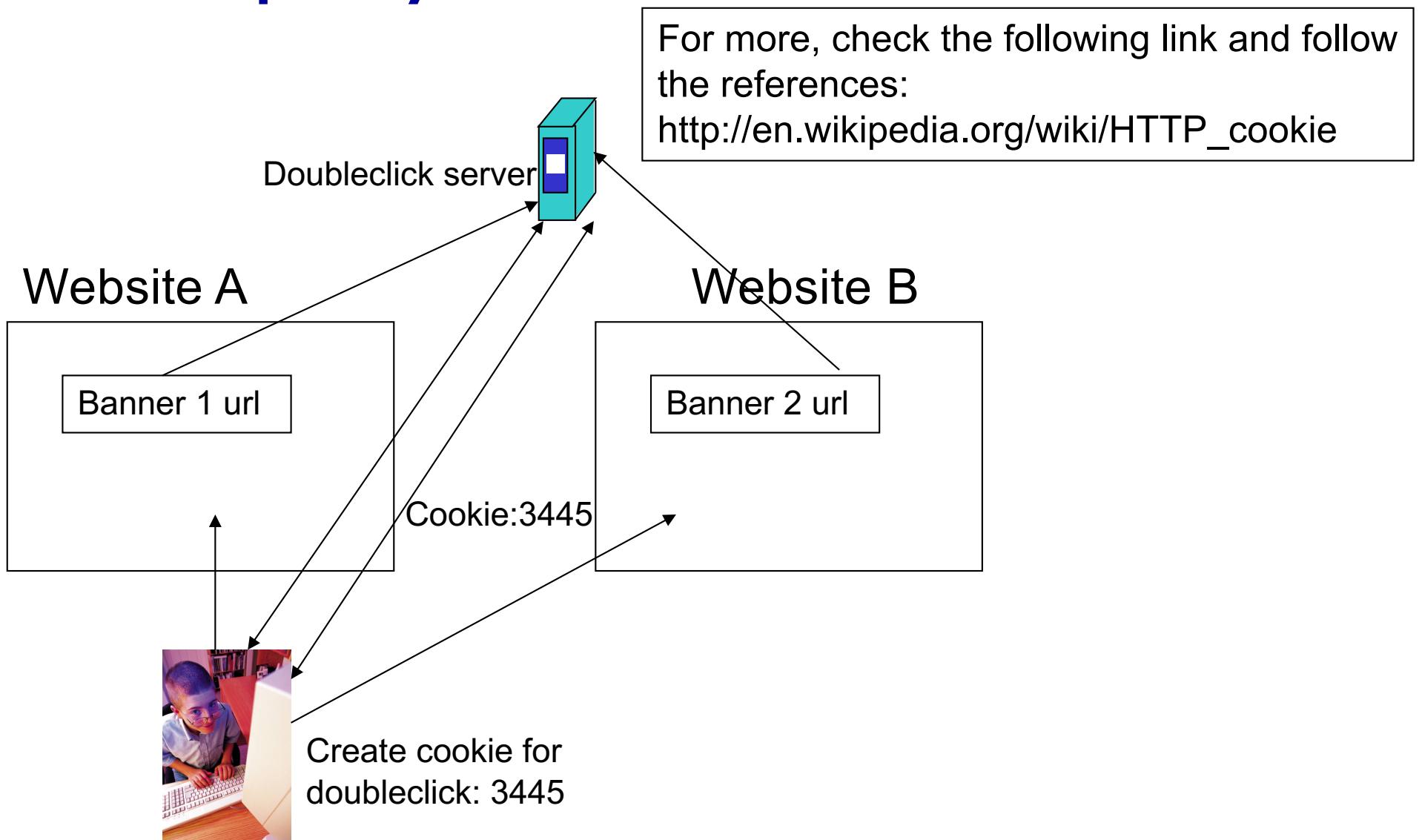
# Cookies: keeping “state” (cont.)



# The Dark Side of Cookies

- ❖ Cookies permit sites to learn a lot about you
- ❖ You may supply name and e-mail to sites (and more)
- ❖ 3<sup>rd</sup> party cookies (from ad networks, etc.) can follow you across multiple sites
  - Ever visit a website, and the next day ALL your ads are from them ?
    - Check your browser's cookie file (cookies.txt, cookies.plist)
    - Do you see a website that you have never visited
- ❖ You COULD turn them off
  - But good luck doing anything on the Internet !!

# Third party cookies



In practice the banner can be a single pixel (invisible to the user)

# Performance of HTTP

- Page Load Time (PLT) as the metric
  - From click until user sees page
  - Key measure of web performance
- Depends on many factors such as
  - page content/structure,
  - protocols involved and
  - Network bandwidth and RTT

# Performance Goals

- ❖ User
  - fast downloads
  - high availability
- ❖ Content provider
  - happy users (hence, above)
  - cost-effective infrastructure
- ❖ Network (secondary)
  - avoid overload

# Solutions?

- ❖ User
  - fast downloads
  - high availability
- ❖ Content provider
  - happy users (hence, above)
  - cost-effective infrastructure
- ❖ Network (secondary)
  - avoid overload

Improve HTTP to  
achieve faster  
downloads



# Solutions?

- ❖ User

- fast downloads
- high availability

Improve HTTP to  
achieve faster  
downloads

- ❖ Content provider

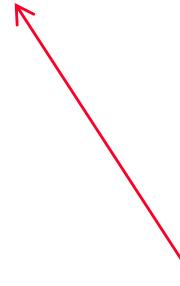
- happy users (hence, above)
- cost-effective delivery infrastructure

Caching and Replication

- ❖ Network (secondary)

- avoid overload

# Solutions?

- ❖ User
    - fast downloads
    - high availability
  - ❖ Content provider
    - happy users (hence, above)
    - cost-effective delivery infrastructure
  - ❖ Network (secondary)
    - avoid overload
- Improve HTTP to  
achieve faster  
downloads
- Caching and Replication
- Exploit economies of scale  
(Webhosting, CDNs, datacenters)
- 

# How to improve PLT

- Reduce content size for transfer
  - Smaller images, compression
- Change HTTP to make better use of available bandwidth
  - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
  - Caching and web-proxies
- Move content closer to the client
  - CDNs

# HTTP Performance

- ❖ Most Web pages have multiple objects
  - e.g., HTML file and a bunch of embedded images
- ❖ How do you retrieve those objects (naively)?
  - *One item at a time*
- ❖ New TCP connection per (small) object!

## *non-persistent HTTP*

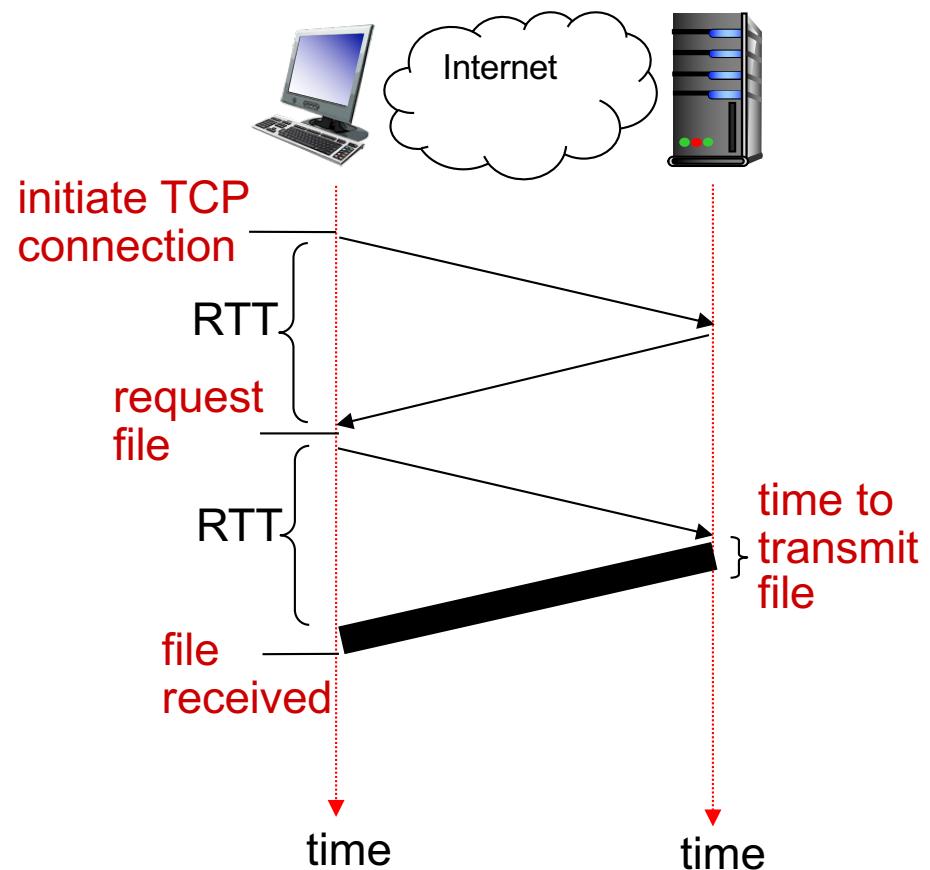
- ❖ at most one object sent over TCP connection
  - connection then closed
- ❖ downloading multiple objects required multiple connections

# Non-persistent HTTP: response time

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

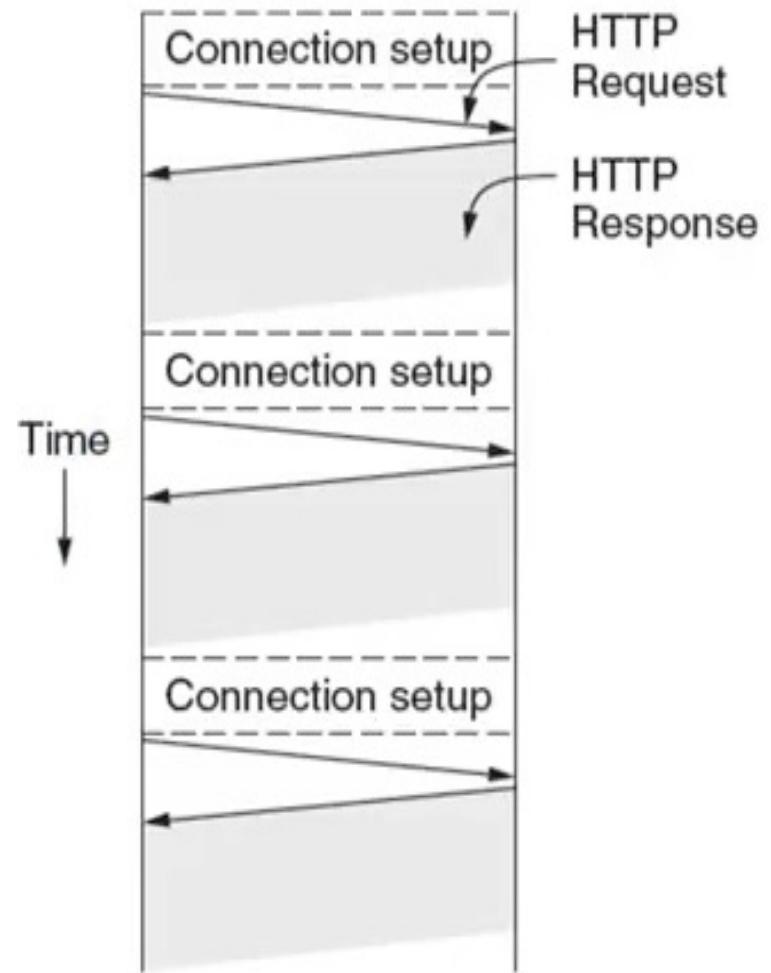
**HTTP response time:**

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



# HTTP/1.0

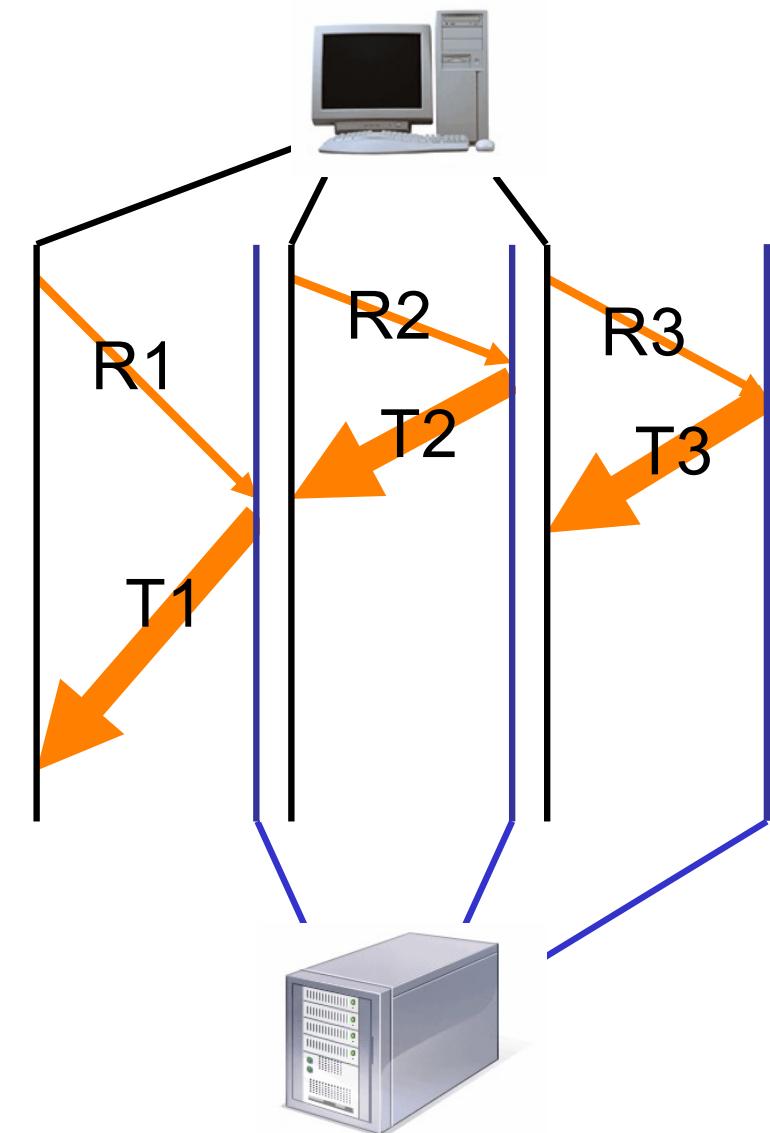
- Non-Persistent: One TCP connection to fetch one web resource
- Fairly poor PLT
- 2 Scenarios
  - Multiple TCP connections setups to the **same server**
  - Sequential request/responses even when resources are located on **different servers**
- Multiple TCP slow-start phases (more in lecture on TCP)



# Improving HTTP Performance: Concurrent Requests & Responses

---

- ❖ Use multiple connections *in parallel*
- ❖ Does not necessarily maintain order of responses



# Quiz: Parallel HTTP Connections



- ❖ What are potential downsides of parallel HTTP connections, i.e., can opening too many parallel connections be harmful and if so in what way?
  
- ❖ Answer: Extra load on the server for managing parallel connections

Open a browser and type: **[www.zeetings.com/salil](http://www.zeetings.com/salil)**

# Persistent HTTP

## Persistent HTTP

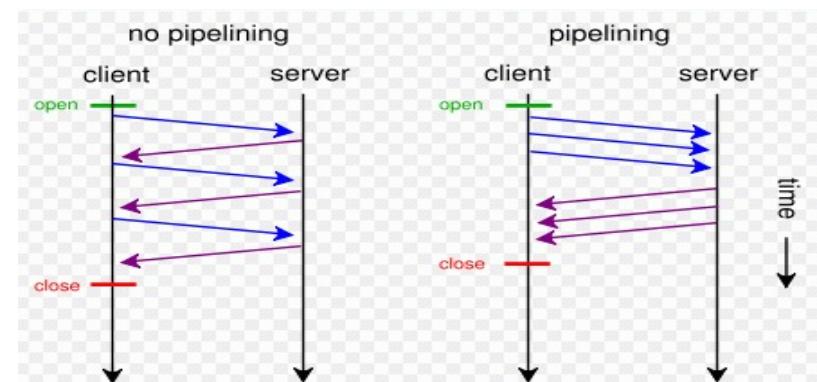
- ❖ server leaves TCP connection open after sending response
- ❖ subsequent HTTP messages between same client/server are sent over the same TCP connection
- ❖ Allow TCP to learn more accurate RTT estimate (APPARENT LATER IN THE COURSE)
- ❖ Allow TCP congestion window to increase (APPARENT LATER)
- ❖ i.e., leverage previously discovered bandwidth (APPARENT LATER)

## Persistent without pipelining:

- ❖ client issues new request only when previous response has been received
- ❖ one RTT for each referenced object

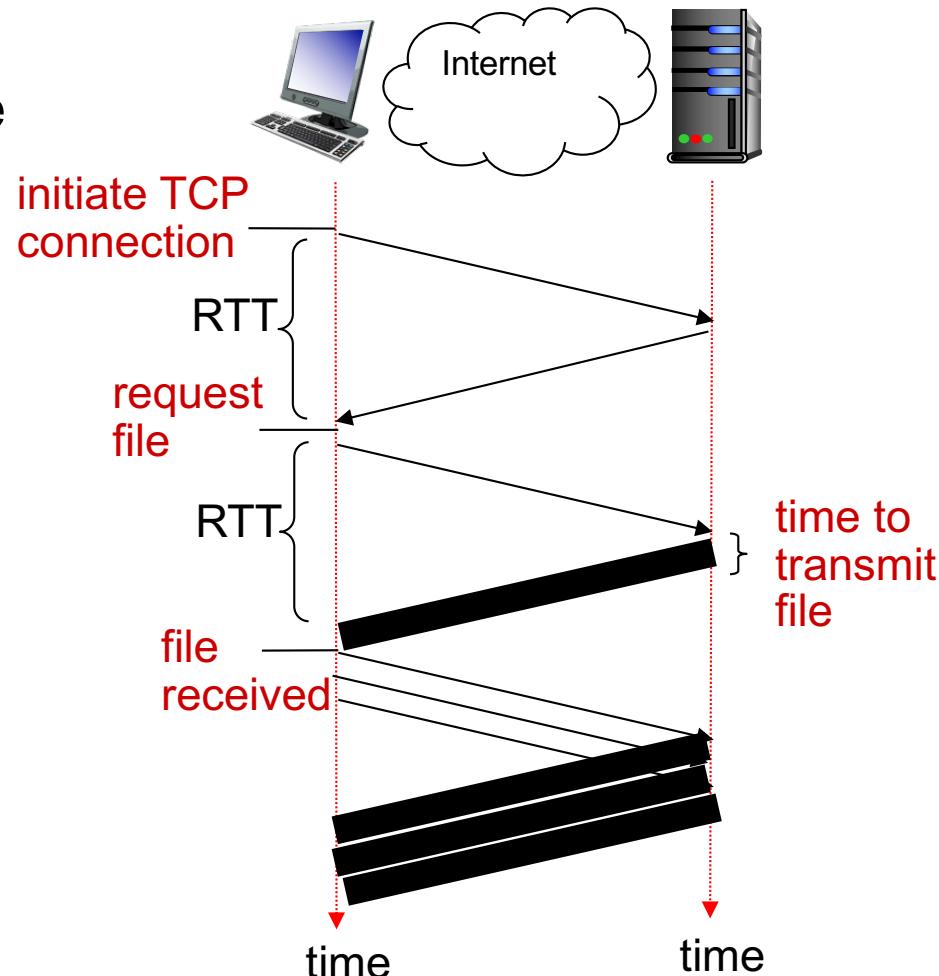
## Persistent with pipelining:

- ❖ introduced in HTTP/1.1
- ❖ client sends requests as soon as it encounters a referenced object
- ❖ as little as one RTT for all the referenced objects



# HTTP 1.1: response time with pipelining

Website with one index page and three embedded objects



# How to improve PLT

- Reduce content size for transfer
  - Smaller images, compression
- Change HTTP to make better use of available bandwidth
  - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
  - Caching and web-proxies
- Move content closer to the client
  - CDNs

# Improving HTTP Performance: Caching

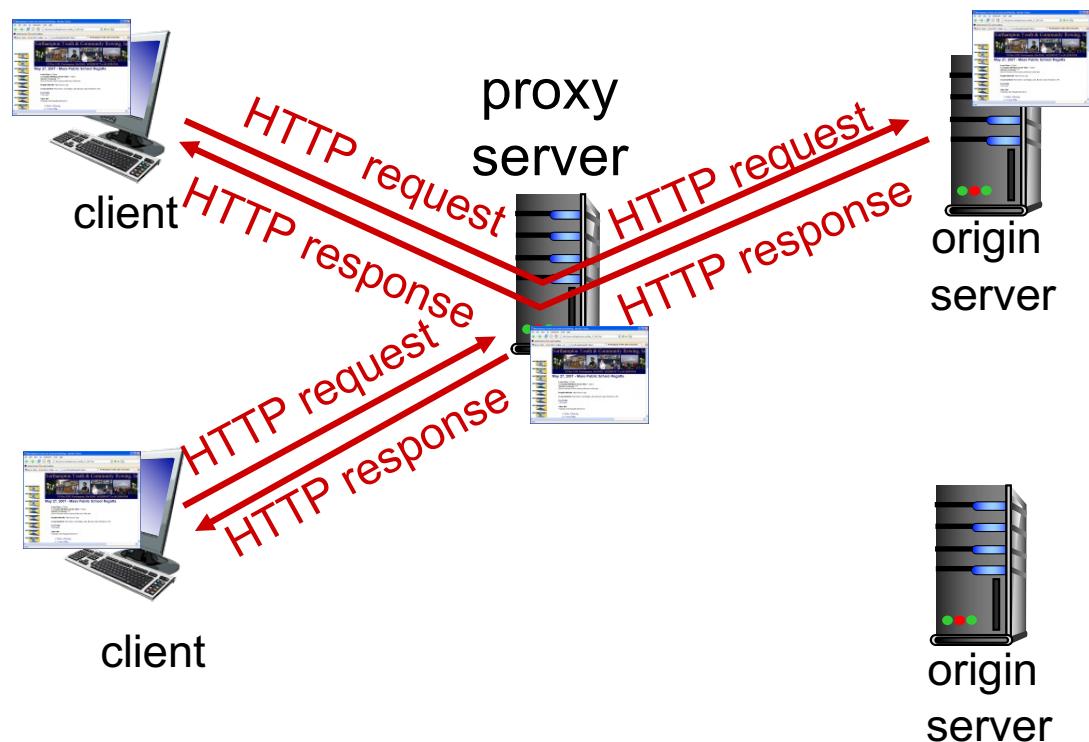
---

- Why does caching work?
  - Exploit *locality of reference*
- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests
- Trend: increase in dynamic content
  - For example, customization of web pages
  - Reduces benefits of caching
  - Some exceptions, for example, video content

# Web caches (proxy server)

**goal:** satisfy client request without involving origin server

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



# More about Web caching

- ❖ cache acts as both client and server
  - server for original requesting client
  - client to origin server
- ❖ typically, cache is installed by ISP (university, company, residential ISP)

## *why Web caching?*

- ❖ reduce response time for client request
- ❖ reduce traffic on an institution's access link
- ❖ Internet dense with caches: enables “poor” content providers to effectively deliver content

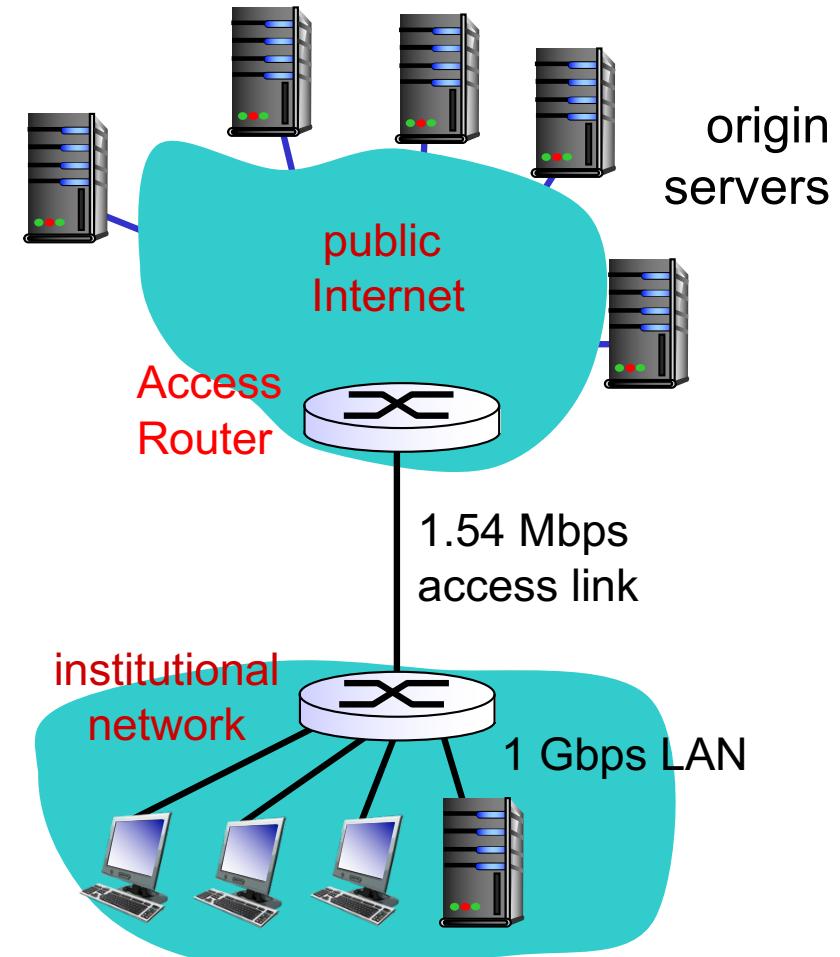
# Caching example:

## *assumptions:*

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

## *consequences:*

- ❖ LAN utilization: 0.15%
- ❖ access link utilization = **99%**
- ❖ total delay = Internet delay +  
access delay + LAN delay  
= 2 sec + minutes + usecs



*problem!*

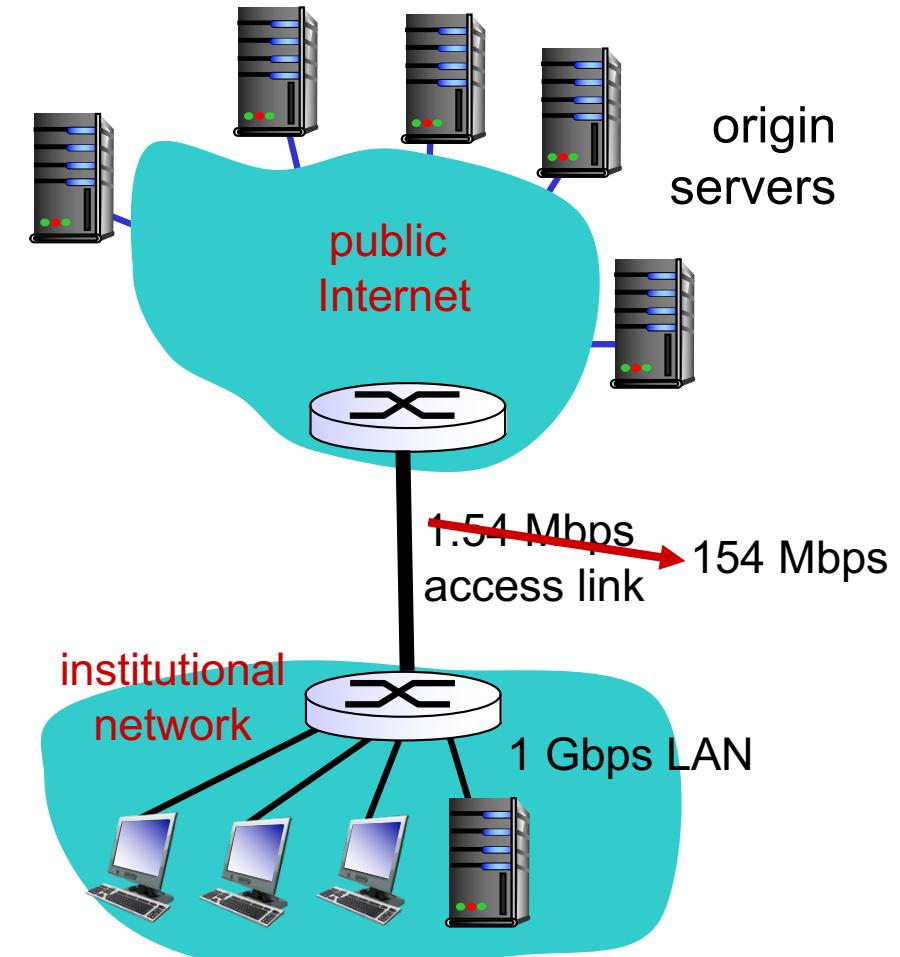
# Caching example: fatter access link

## assumptions:

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

consequences: ~~154 Mbps~~

- ❖ LAN utilization: 0.15%
- ❖ access link utilization = ~~99%~~  $\rightarrow$  0.99%
- ❖ total delay = Internet delay + access delay + LAN delay  
 $= 2 \text{ sec} + \cancel{\text{minutes}} + \cancel{\text{usecs}}$   
 $\qquad\qquad\qquad \text{msecs}$



Cost: increased access link speed (not cheap!)

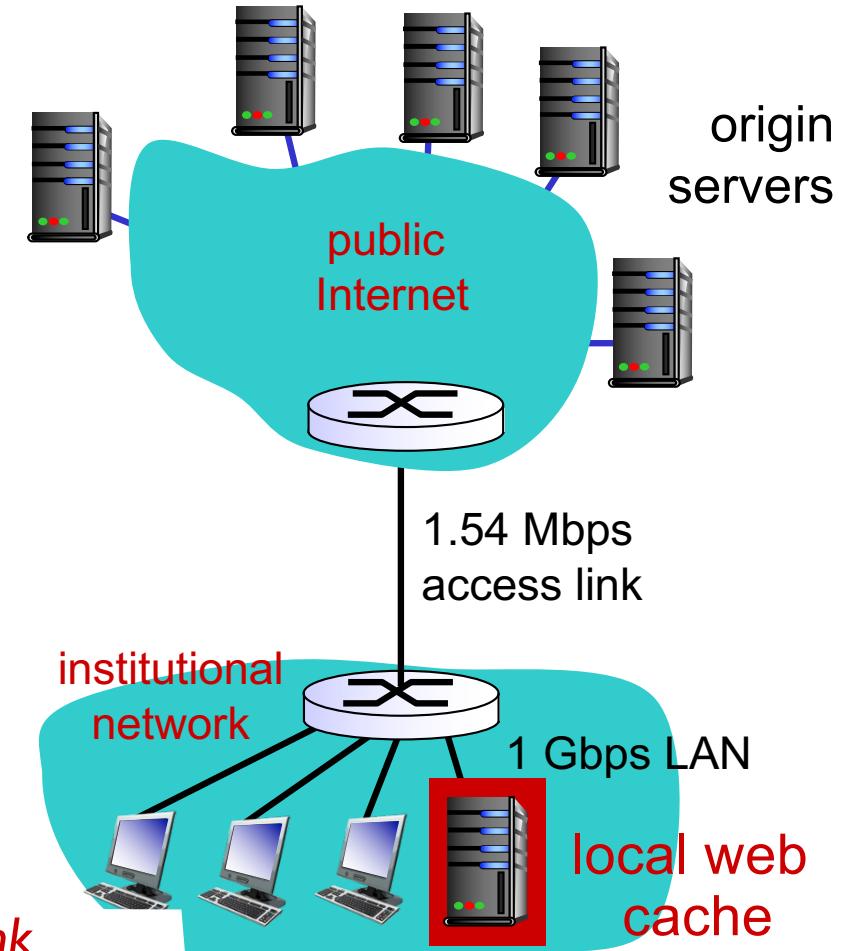
# Caching example: install local cache

## *assumptions:*

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

## *consequences:*

- ❖ LAN utilization: ?
- ❖ access link utilization = ?
- ❖ total delay = ?      *How to compute link utilization, delay?*

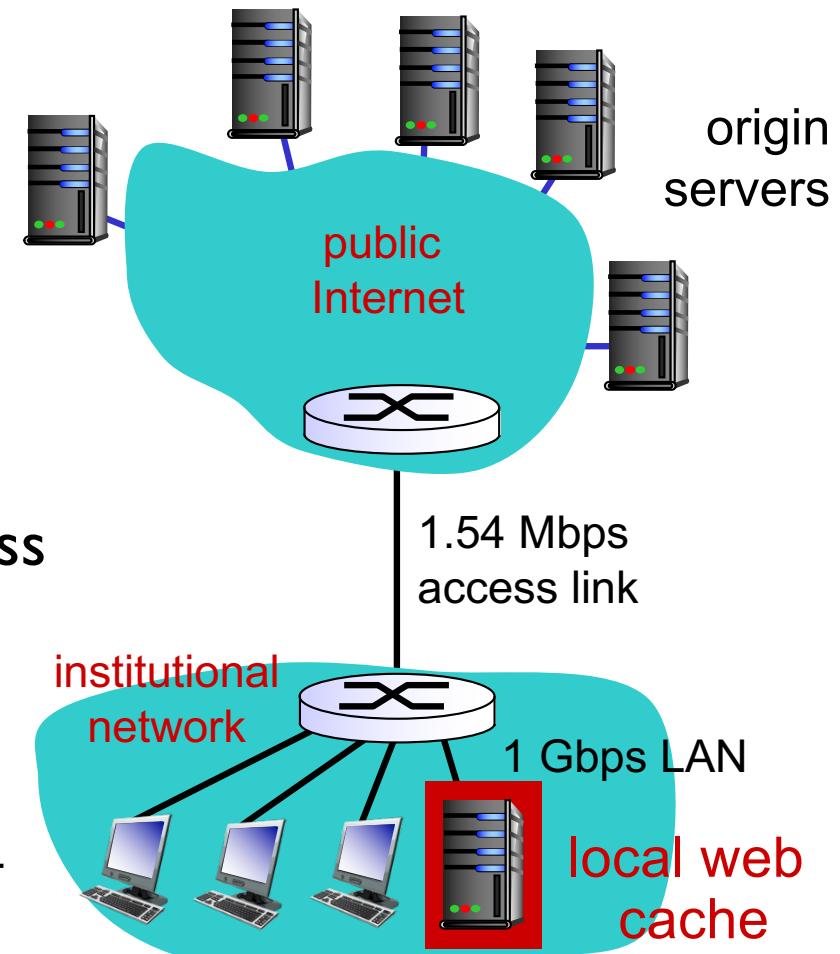


*Cost:* web cache (cheap!)

# Caching example: install local cache

*Calculating access link utilization, delay with cache:*

- ❖ suppose cache hit rate is 0.4
  - 40% requests satisfied at cache;
  - 60% requests satisfied at origin
- ❖ access link utilization:
  - 60% of requests use access link
- ❖ data rate to browsers over access link =  $0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$ 
  - utilization =  $0.9 / 1.54 = .58$
- ❖ total delay
  - $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$
  - $= 0.6 (2.01) + 0.4 (\sim \text{msecs})$
  - $= \sim 1.2 \text{ secs}$
  - less than with 154 Mbps link (and cheaper too!)



# Conditional GET

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

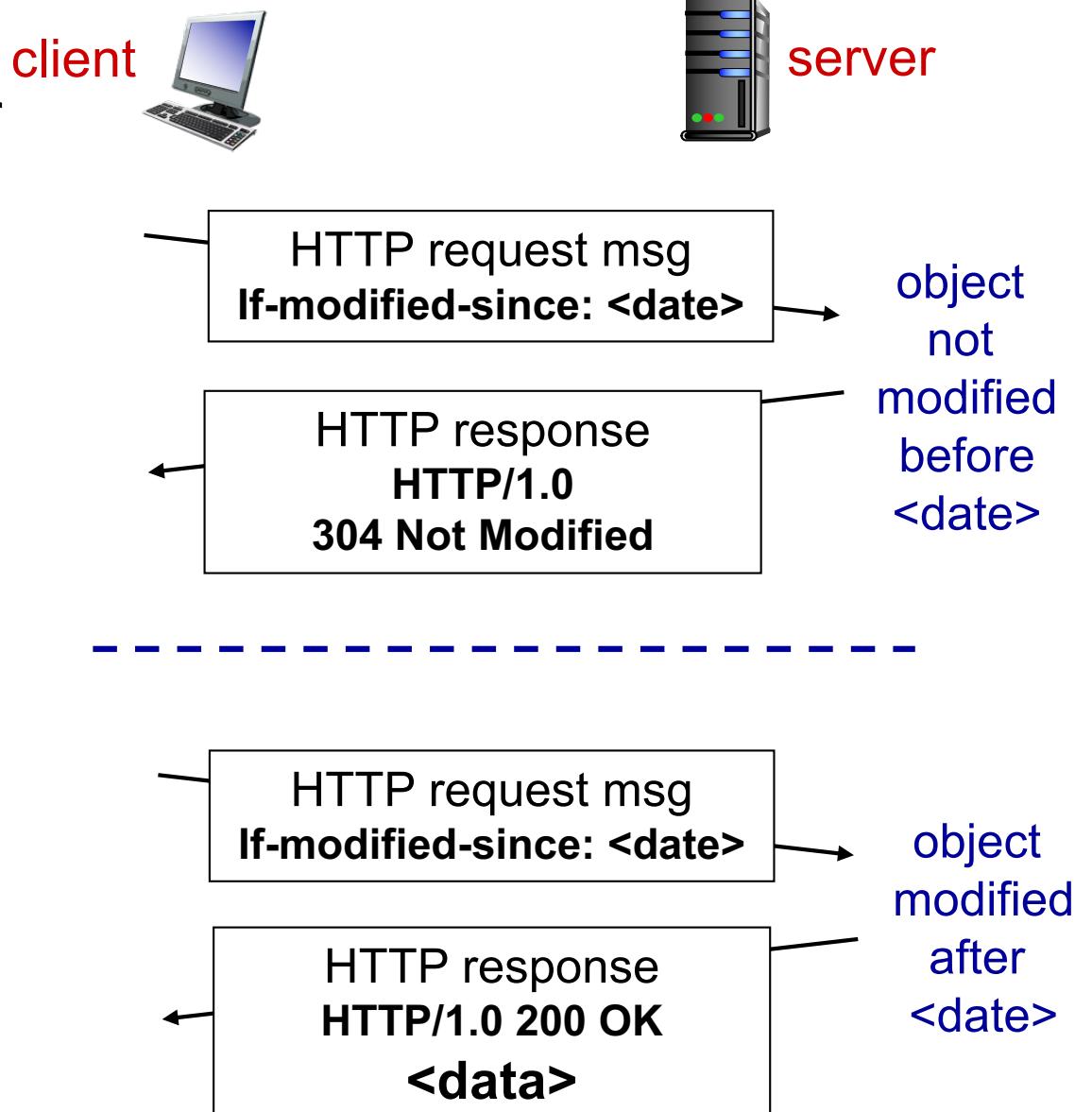
- no object transmission delay
- lower link utilization

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

**If-modified-since:**  
**<date>**

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

**HTTP/1.0 304 Not Modified**



# Example Cache Check Request

GET / HTTP/1.1

Accept: \*/\*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT

If-None-Match: "7a11f-10ed-3a75ae4a"

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT  
5.0)

Host: www.intel-iris.net

Connection: Keep-Alive

# Example Cache Check Response

HTTP/1.1 304 Not Modified

Date: Tue, 27 Mar 2001 03:50:51 GMT

Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod\_ssl/2.7.1  
OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod\_perl/1.24

Connection: Keep-Alive

Keep-Alive: timeout=15, max=100

ETag: "7a11f-10ed-3a75ae4a"

Etag: Usually used for dynamic content. The value is often a cryptographic hash of the content.

# Improving HTTP Performance: Replication

---

- Replicate popular Web site across many machines
  - Spreads load on servers
  - Places content closer to clients
  - Helps when content isn't cacheable
- Problem:
  - Want to direct client to a particular replica
    - Balance load across server replicas
    - Pair clients with nearby servers
  - Expensive
- Common solution:
  - DNS returns different addresses based on client's geo-location, server load, etc.

# Improving HTTP Performance: CDN

---

- Caching and replication as a service
- Large-scale distributed storage infrastructure (usually) administered by one entity
  - e.g., Akamai has servers in 20,000+ locations
- Combination of (pull) caching and (push) replication
  - **Pull:** Direct result of clients' requests
  - **Push:** Expectation of high access rate
- Also do some processing
  - Handle dynamic web pages
  - Transcoding

# What about HTTPS?

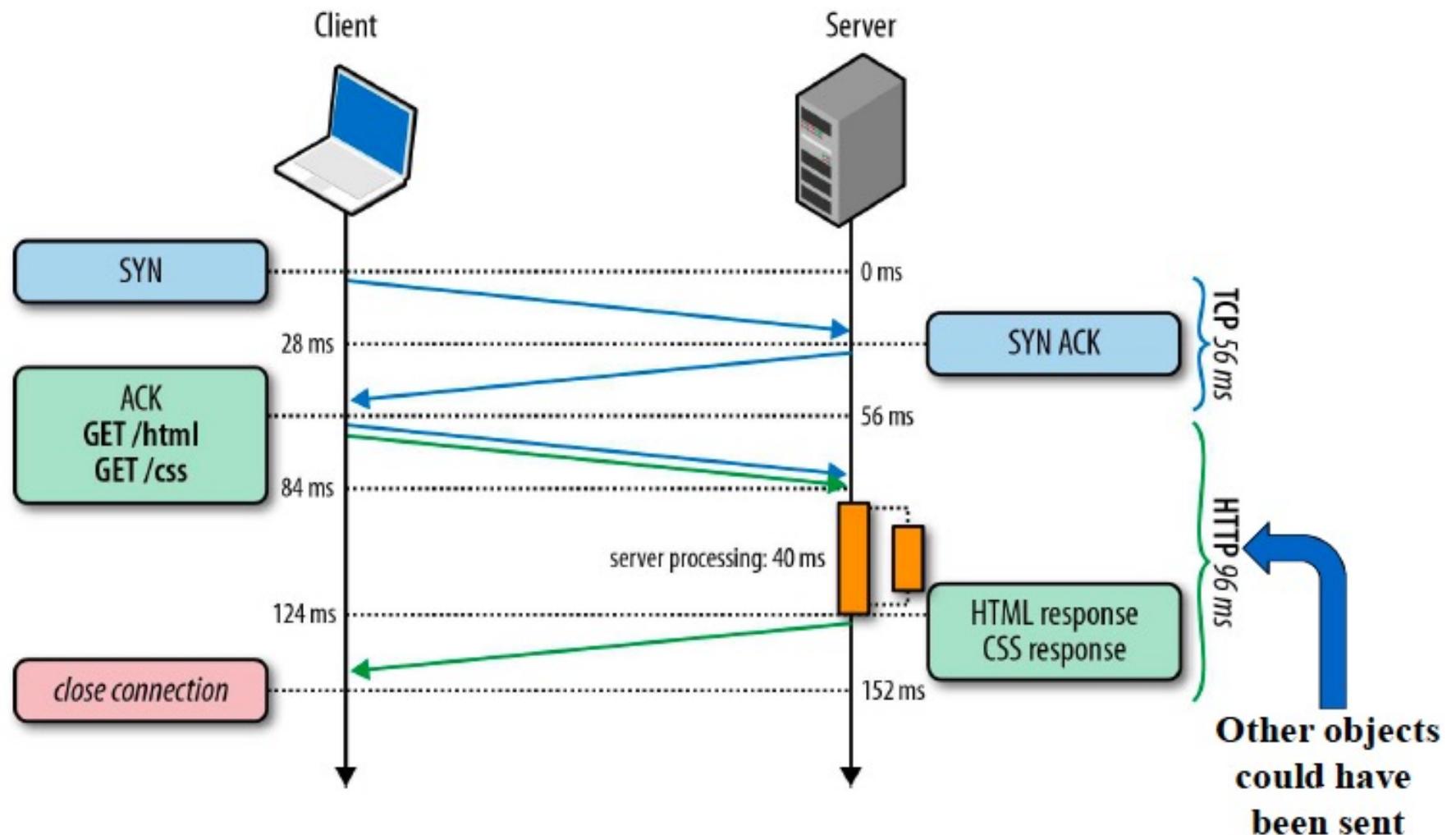
- HTTP is insecure
- HTTP basic authentication: password sent using base64 encoding (can be readily converted to plaintext)
- HTTPS: HTTP over a connection encrypted by Transport Layer Security (TLS)
- Provides:
  - Authentication
  - Bidirectional encryption
- Widely used in place of plain vanilla HTTP



# Issues with HTTP

- Head of line blocking: “slow” objects delay later requests
  - Example objects from remote storage vs from local memory
- Browsers often open multiple TCP connections for parallel transfers
  - Increases throughput and reduces impact of HOL blocking
  - Increases load on servers and network
- HTTP headers are big
  - Overheads higher for small objects
- Objects have dependencies, different priorities
  - Javascript vs images
  - Extra RTTs for “dependent” objects

# Head of Line Blocking Example



# What's on the horizon: HTTP/2

- Google SPDY (speedy) -> HTTP/2: (RFC 7540 May 2015)
- Binary instead of text
  - Efficient to parse, more compact and much less error-prone
- Responses are multiplexed over a single TCP connection
  - Server can send response data whenever it is ready
  - “Fast” objects can bypass “slow” objects – avoid HOL blocking
  - Fewer handshakes, more traffic (helps congestion control)
- Multiplexing uses prioritized flow-controlled schemes
  - Urgent responses can bypass non-critical responses
- Single TCP connection
- HTTP headers are compressed
- Push feature allows server to push embedded objects to the client without waiting for request
  - Saves RTT

More details: <https://http2.github.io/faq/>  
Demo: <http://www.http2demo.io>



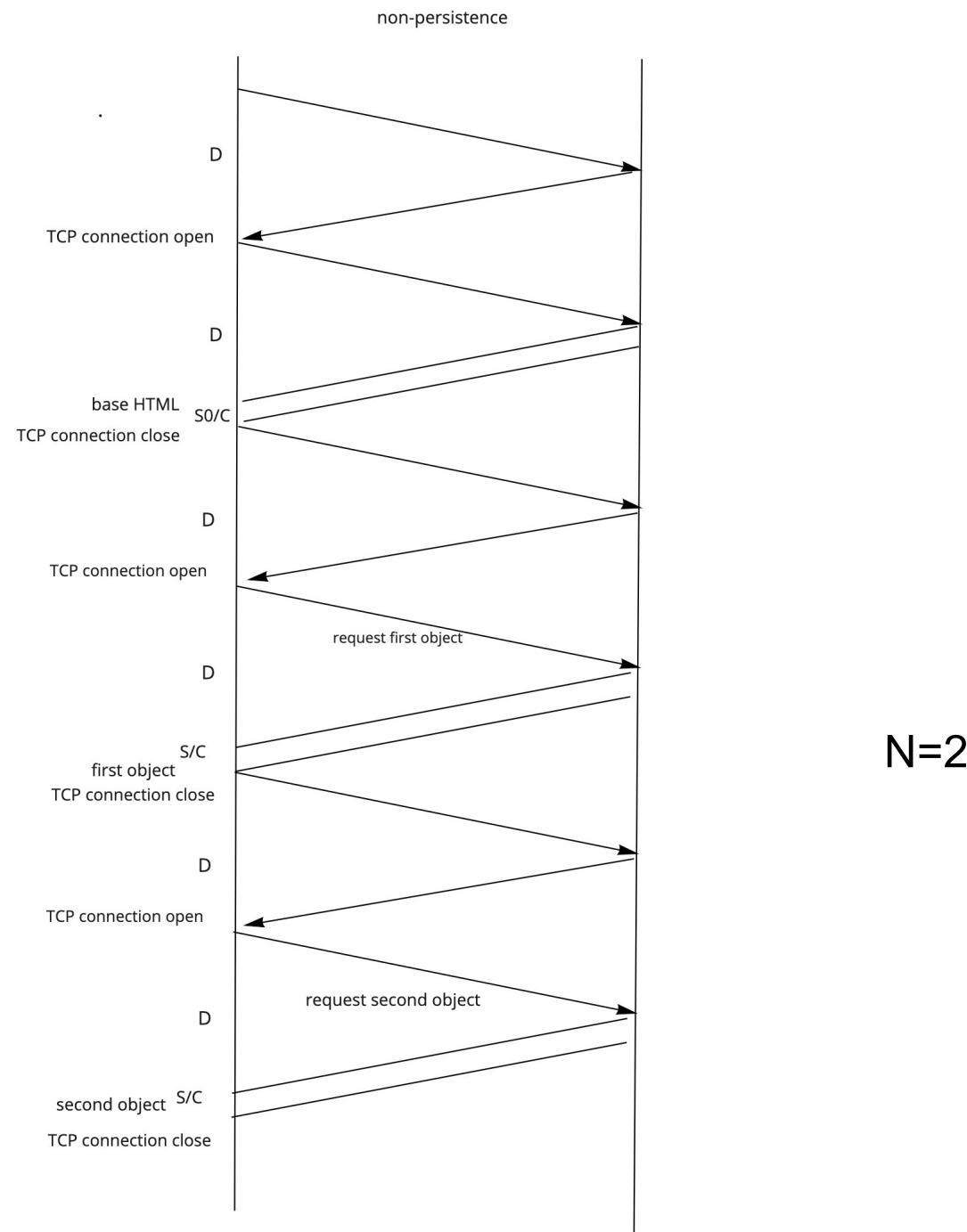
## Quiz: HTTP (1)

Consider an HTML page with a base file of size  $S_0$  bits and  $N$  inline objects each of size  $S$  bits. Assume a client fetching the page across a link of capacity  $C$  bits/s and RTT of  $D$ . How long does it take to download the page using **non-persistent HTTP (without parallelism)**?

- A.  $D + (S_0 + NS)/C$
- B.  $2D + (S_0 + NS)/C$
- C.  $N(D + S/C)$
- D.  $2D + S_0/C + N(2D + S/C)$
- E.  $2D + S_0/C + N(D + S/C)$

**Answer: D**  
(see picture on next slide)

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)



N=2



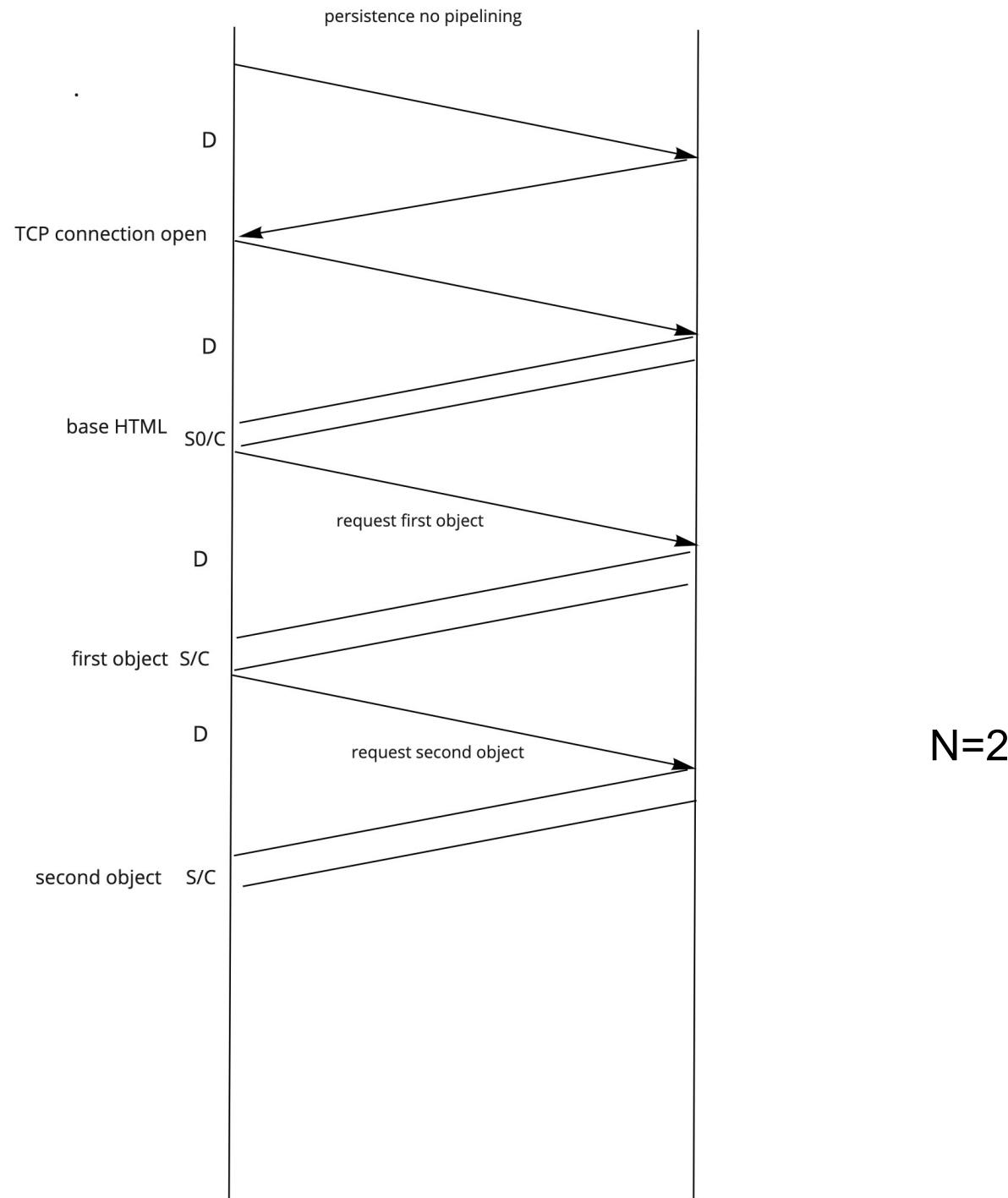
## Quiz: HTTP (2)

Consider an HTML page with a base file of size  $S_0$  bits and  $N$  inline objects each of size  $S$  bits. Assume a client fetching the page across a link of capacity  $C$  bits/s and RTT of  $D$ . How long does it take to download the page using **persistent HTTP (without parallelism or pipelining)**?

- A.  $2D + (S_0 + NS)/C$
- B.  $3D + (S_0 + NS)/C$
- C.  $N(D + S/C)$
- D.  $2D + S_0/C + N(2D + S/C)$
- E.  $2D + S_0/C + N(D + S/C)$

**Answer: E**  
(see picture on next slide)

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)





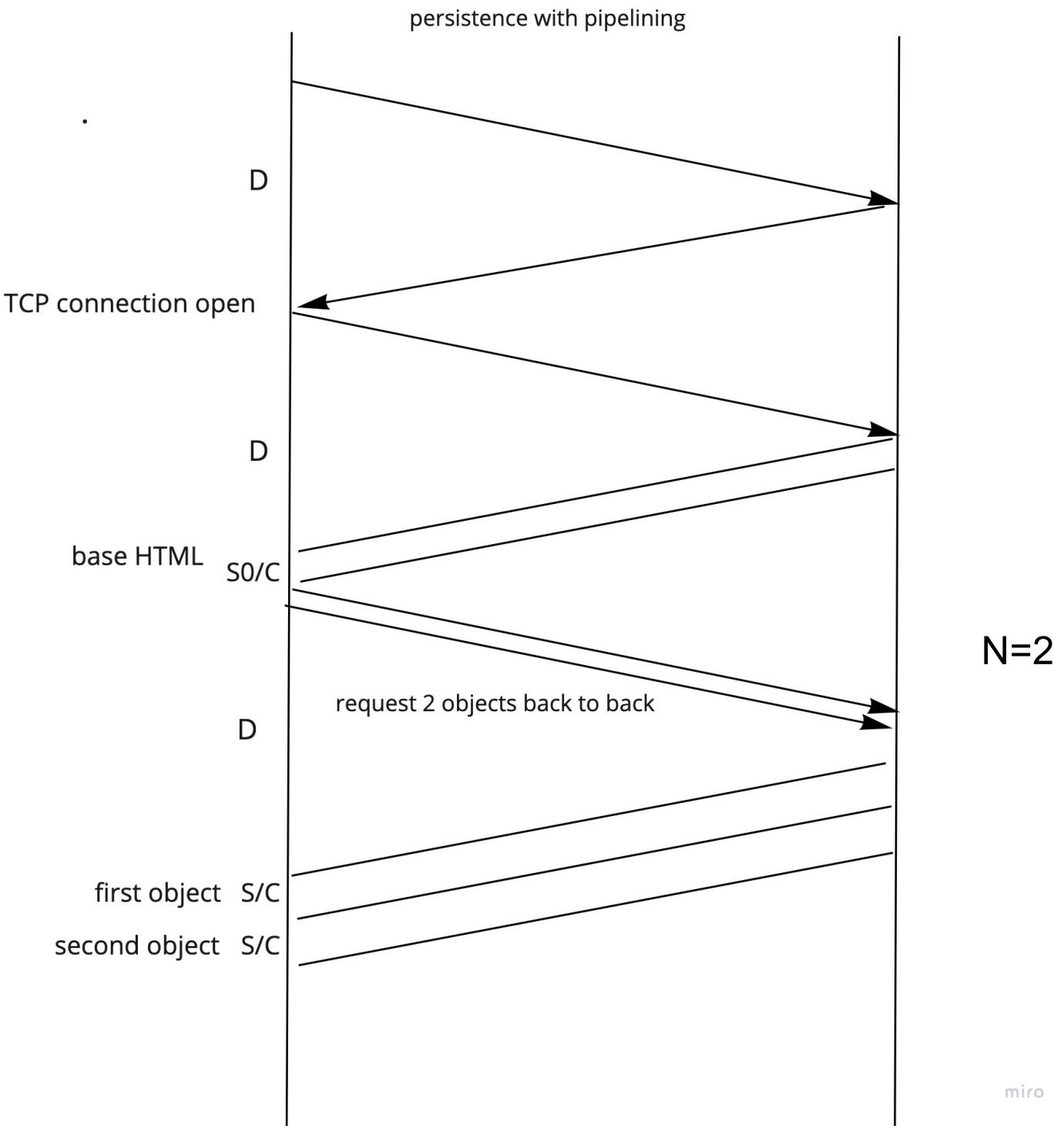
## Quiz: HTTP (3)

Consider an HTML page with a base file of size  $S_0$  bits and  $N$  inline objects each of size  $S$  bits. Assume a client fetching the page across a link of capacity  $C$  bits/s and RTT of  $D$ . How long does it take to download the page using **persistent HTTP with pipelining?**

- A.  $2D + (S_0 + NS)/C$
- B.  $4D + (S_0 + NS)/C$
- C.  $N(D + S/C)$
- D.  $3D + S_0/C + NS/C$
- E.  $2D + S_0/C + N(D + S/C)$

**Answer: D**  
(see picture on next slide)

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)



# Application Layer: outline

## 2.1 principles of network applications

- app architectures
- app requirements

## 2.2 Web and HTTP

## 2.3 electronic mail

- SMTP

## 2.4 DNS

## 2.5 P2P applications

## 2.6 video streaming and content distribution networks (CDNs)

## 2.7 socket programming with UDP and TCP

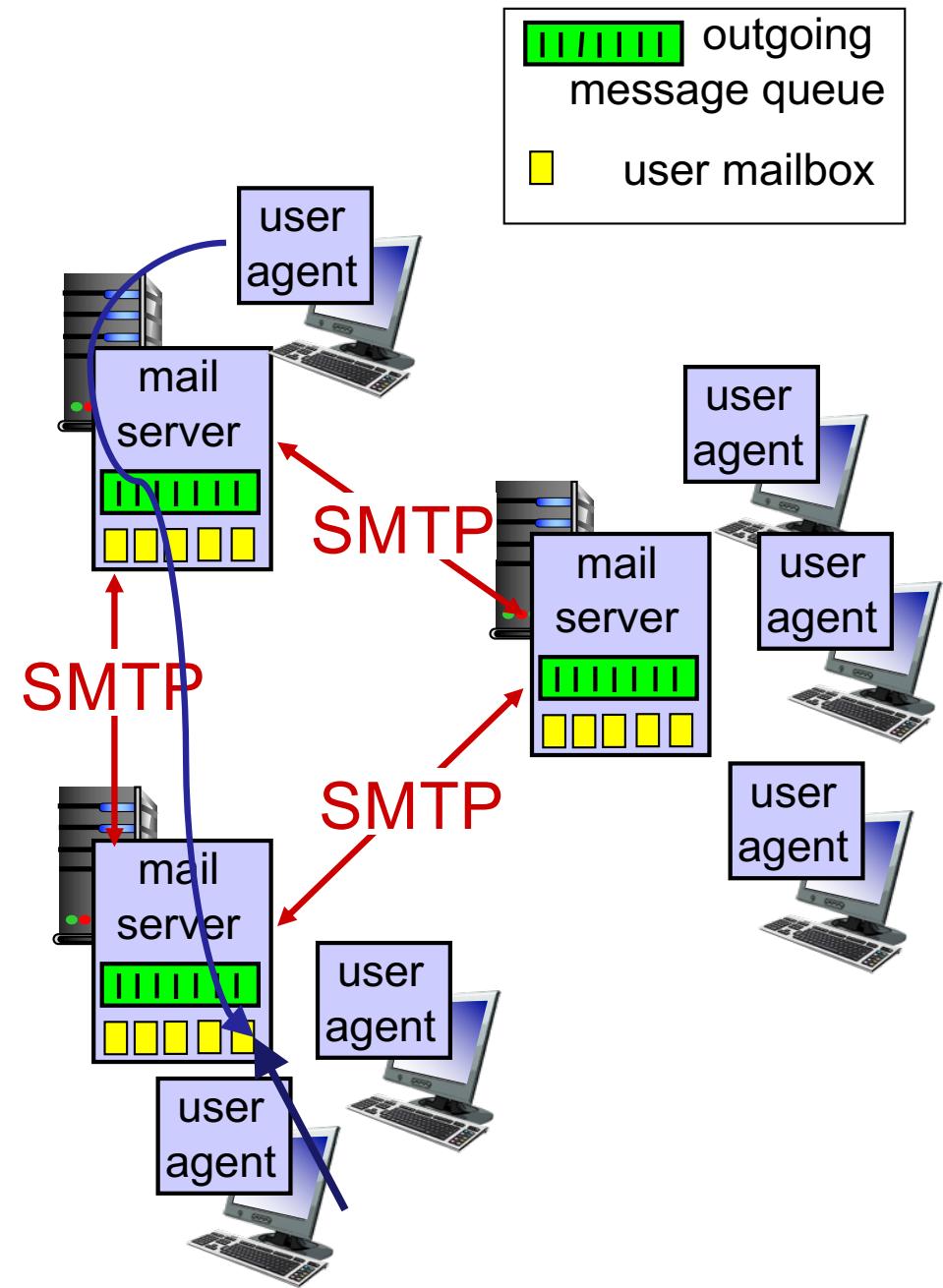
# Electronic mail

*Three major components:*

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

## User Agent

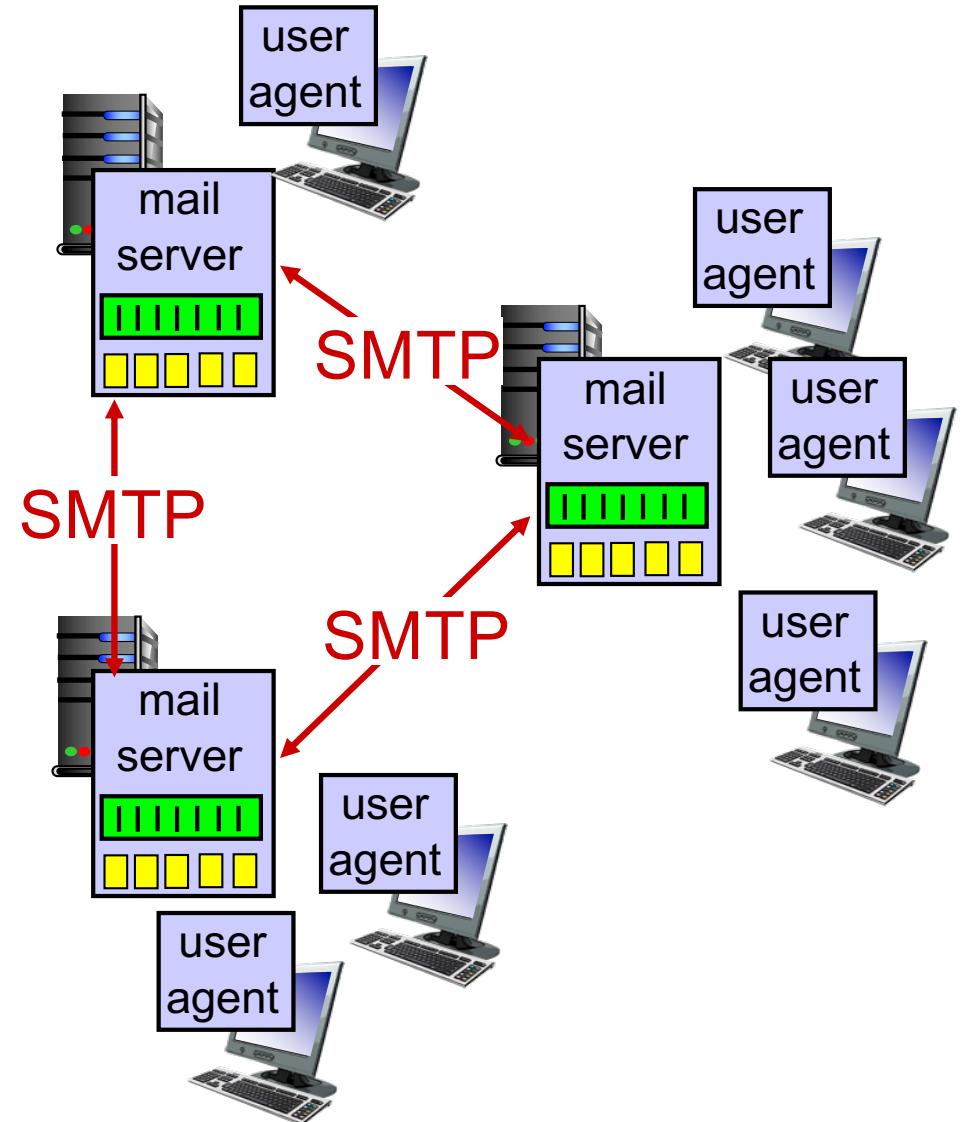
- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server



# Electronic mail: mail servers

## mail servers:

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

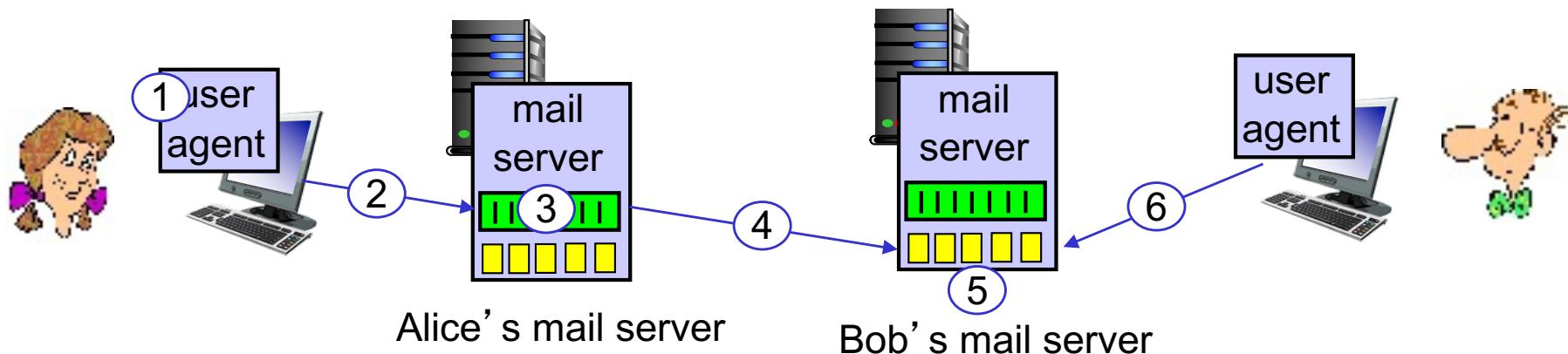


# Electronic Mail: SMTP [RFC 2821]

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

# Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message “to”  
bob@someschool.edu
- 2) Alice’s UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob’s mail server
- 4) SMTP client sends Alice’s message over the TCP connection
- 5) Bob’s mail server places the message in Bob’s mailbox
- 6) Bob invokes his user agent to read message



# Sample SMTP interaction

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

# SMTP: final words

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

## *comparison with HTTP:*

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

# Mail message format

SMTP: protocol for  
exchanging email msgs

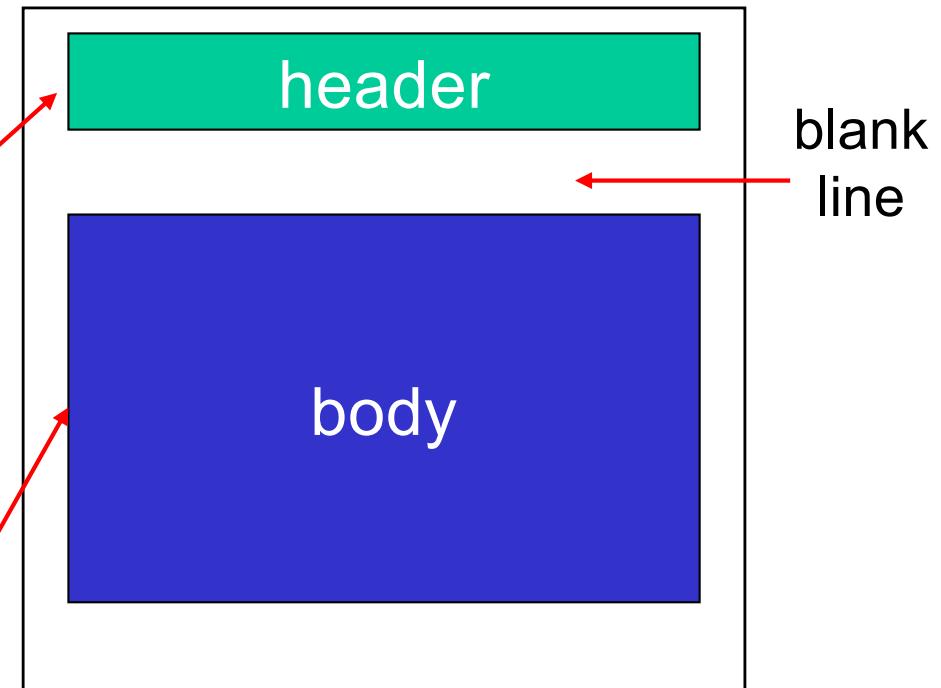
RFC 5322 (822,2822):  
standard for text message  
format (Internet Message  
Format, IMF):

- ❖ header lines, e.g.

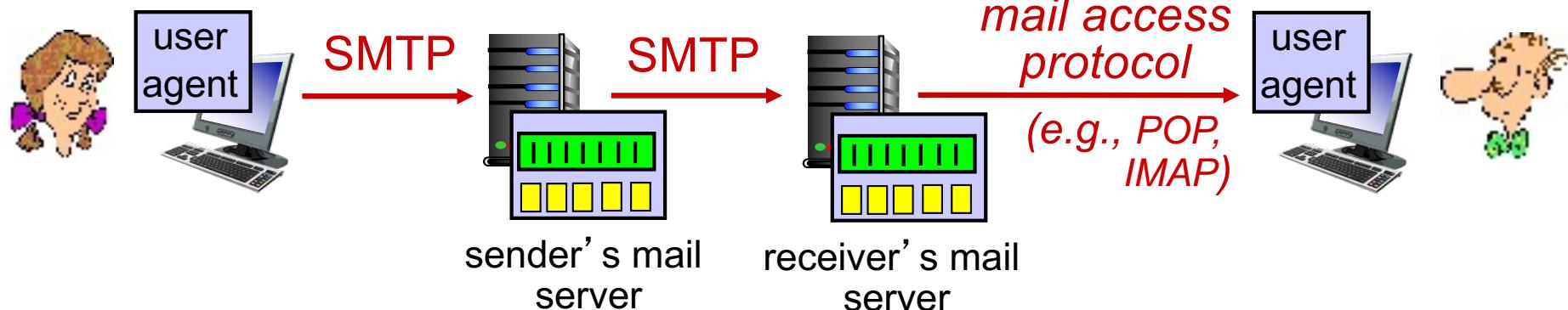
- To:
  - From:
  - Subject:

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

- ❖ Body: the “message”
  - ASCII characters only



# Mail access protocols



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

Read about POP and IMAP from the text in your own time

# Quiz: SMTP

**Why do we have Sender's mail server?**

- User agent can directly connect with recipient mail server without the need of sender's mail server? What's the catch?

**ANSWER: TO ENSURE THAT THE MAIL CAN BE DELIVERED IF THE RECEIVER'S MAIL SERVER IS DOWN MOMENTARILY**

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)

# Quiz: SMTP

**Why do we have a separate Receiver's mail server?**

- Can't the recipient run the mail server on own end system?

**ANSWER: THE RECIPIENT MAY NOT BE ALWAYS CONNECTED**

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)

# Summary

- ❖ Application Layer (Chapter 2)
  - Principles of Network Applications
  - HTTP
  - E-mail
- ❖ Next:
  - DNS
  - P2P



**Reading Exercise for next week  
Chapter 2: 2.4 – 2.7**



# I. Introduction: 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

Self study



## Quiz: Circuit Switching

Consider a circuit-switched network with  $N=100$  users where each user is independently active with probability  $p=0.2$  and when active, sends data at a rate of  $R=1\text{Mbps}$ . How much capacity must the network be provisioned with to guarantee service to all users?

- A. 100 Mbps
- B. 20 Mbps
- C. 200 Mbps
- D. 50 Mbps
- E. 500 Mbps

**Answer: A**

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)



## Quiz: Statistical Multiplexing

Consider a packet-switched network with  $N=100$  users where each user is independently active with probability  $p=0.2$  and when active, sends data at a rate of  $R=1\text{Mbps}$ . What is the expected aggregate traffic sent by all the users?

- A. 100 Mbps
- B. 20 Mbps
- C. 200 Mbps
- D. 50 Mbps
- E. 500 Mbps

**Answer: B**

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)

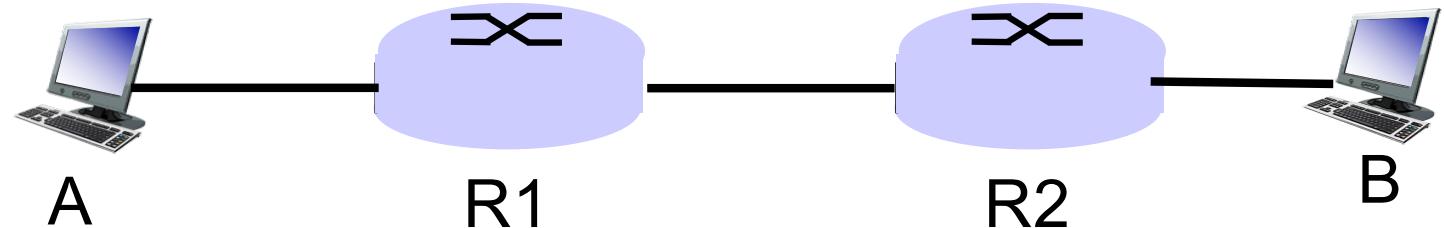


## Quiz: Delays

Consider a network connecting hosts A and B through two routers R1 and R2 like this: A-----R1-----R2-----B. Does whether a packet sent by A destined to B experiences queuing at R1 depend on the length of the link R1-R2?

- A. Yes, it does
- B. No, it doesn't

**Answer: B**



Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)

# Three (networking) design steps

- ❖ Break down the problem into tasks
- ❖ Organize these tasks
- ❖ Decide who does what

# Tasks in Networking

- ❖ What does it take to send packets across?
  
- ❖ Prepare data (Application)
- ❖ Ensure that packets get to the dst process (Transport)
- ❖ Deliver packets across global network (Network)
- ❖ Delivery packets within local network to next hop (Datalink)
- ❖ Bits / Packets on wire (Physical)

This is decomposition...

Now, how do we organize these tasks?

**Let us have an example**

# Inspiration....

- ❖ CEO A writes letter to CEO B
  - Folds letter and gives it to Executive Assistant (EA)

**Dear John,**

» EA:

**Your days are numbered.**

- » Puts letter in envelope with CEO B's full name
- » Takes to FedEx

**--Grace**

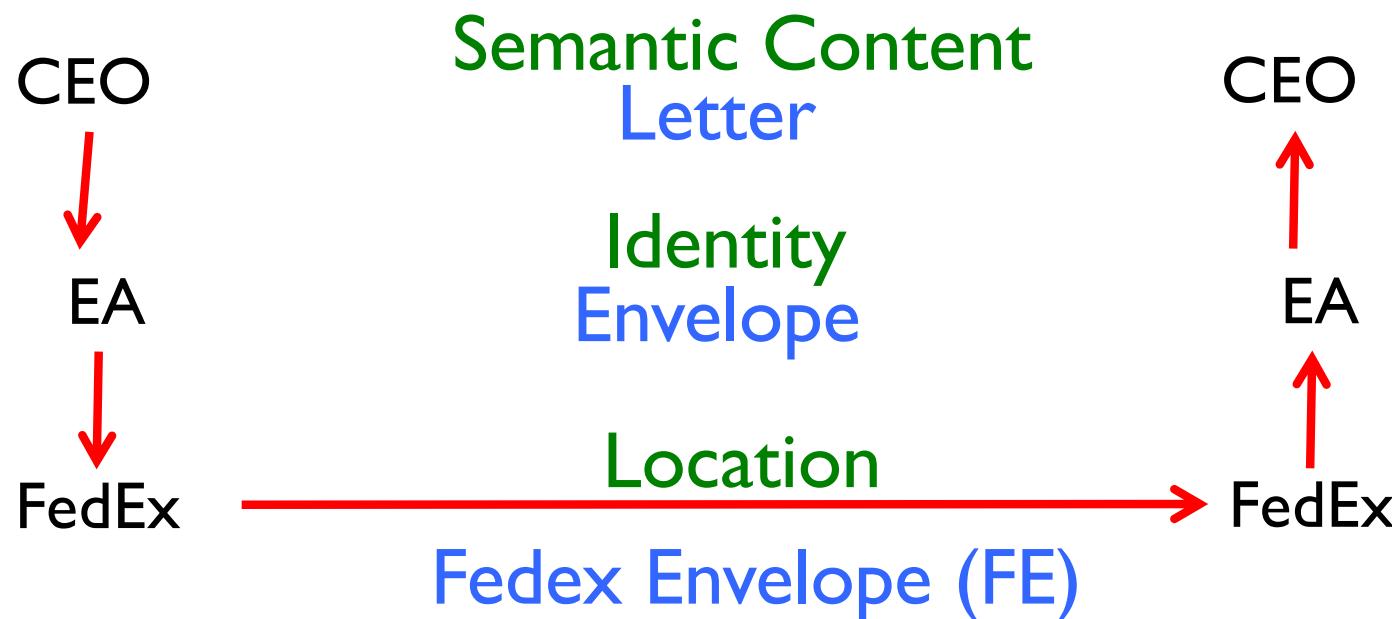
- ❖ FedEx Office
  - Puts letter in larger envelope
  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck
- ❖ FedEx delivers to other company

# The Path of the Letter

“Peers” on each side understand the same things

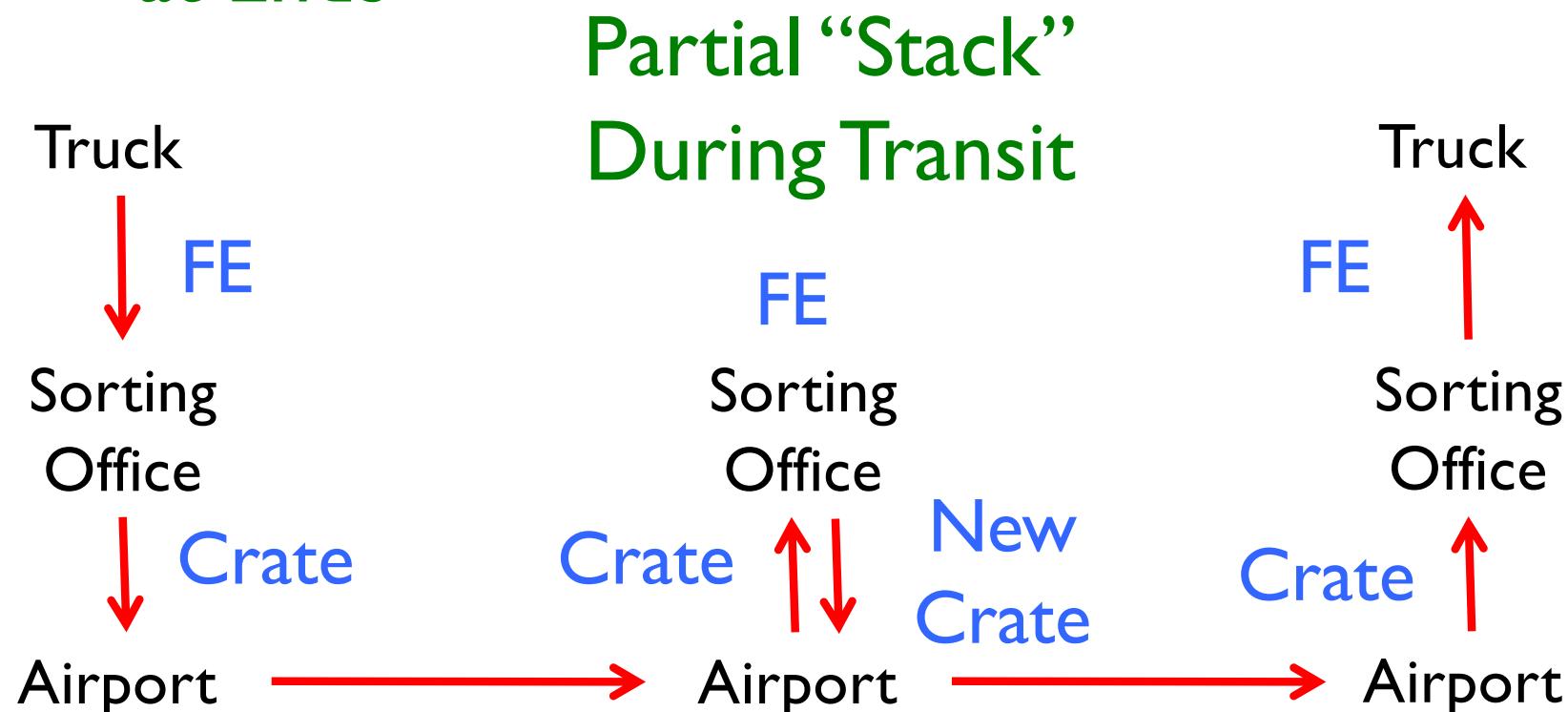
No one else needs to (abstraction)

Lowest level has most packaging



# The Path Through FedEx

Higher “Stack”  
at Ends



Deepest Packaging (Envelope+FE+Crate)  
at the Lowest Level of Transport

# In the context of the Internet

---

Applications

...built on...

Reliable (or unreliable) transport

...built on...

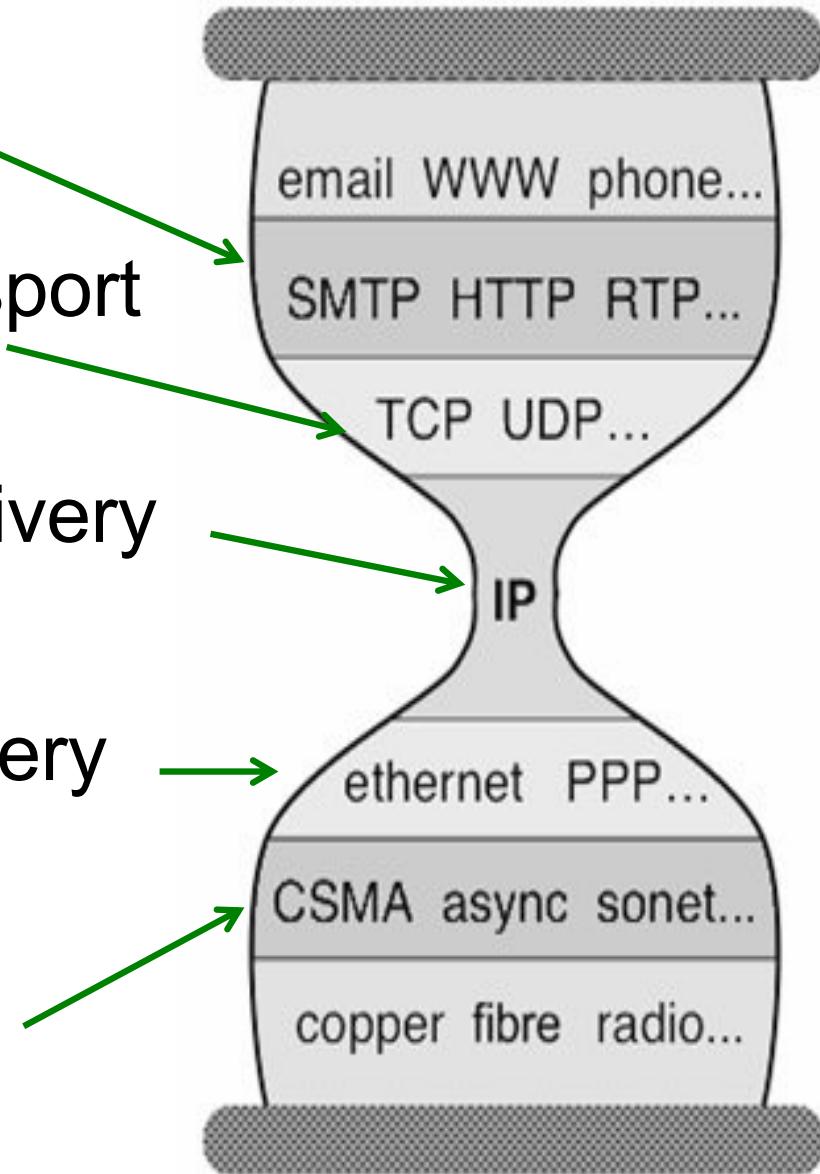
Best-effort global packet delivery

...built on...

Best-effort local packet delivery

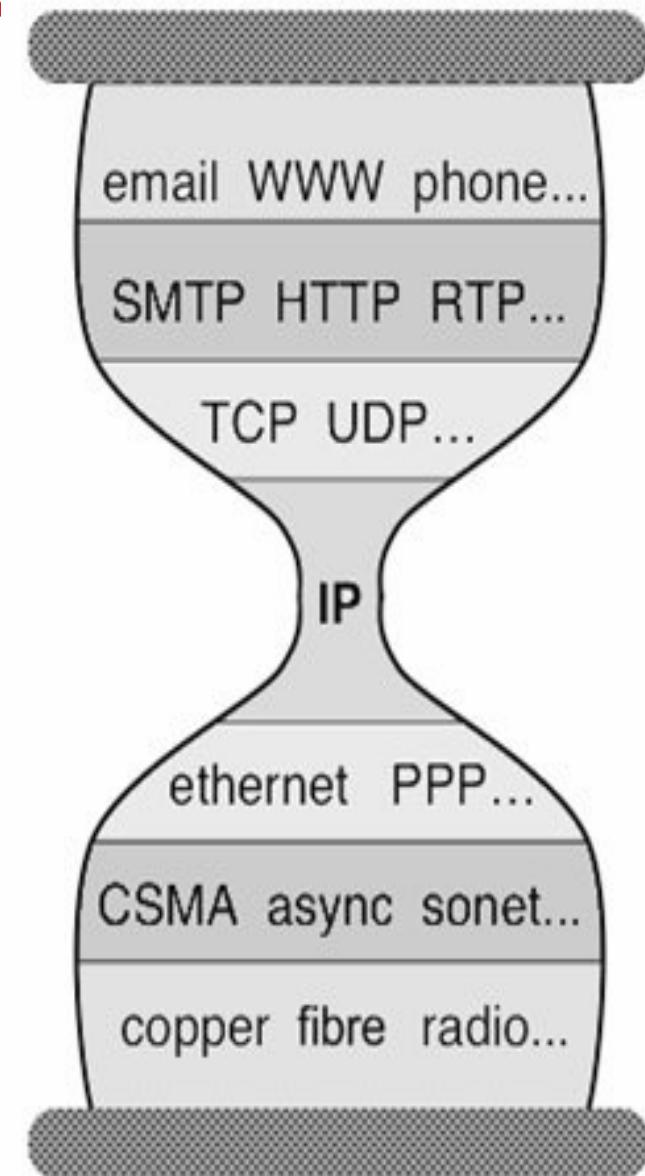
...built on...

Physical transfer of bits



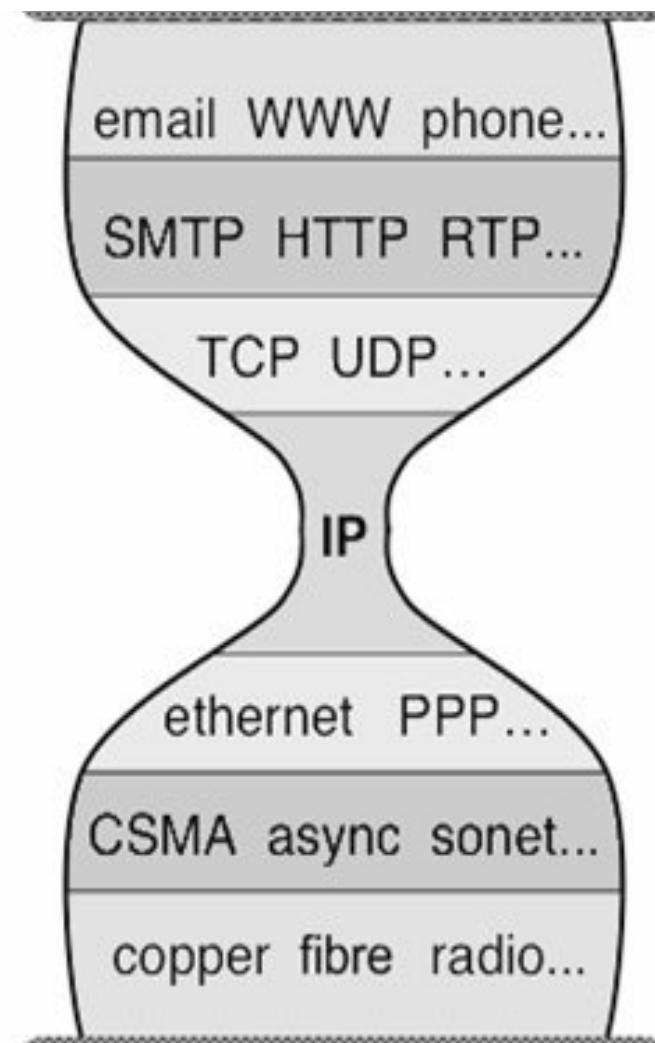
# Internet protocol stack

- ❖ *application*: supporting network applications
  - FTP, SMTP, HTTP, Skype, ..
- ❖ *transport*: process-process data transfer
  - TCP, UDP
- ❖ *network*: routing of datagrams from source to destination
  - IP, routing protocols
- ❖ *link*: data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- ❖ *physical*: bits “on the wire”

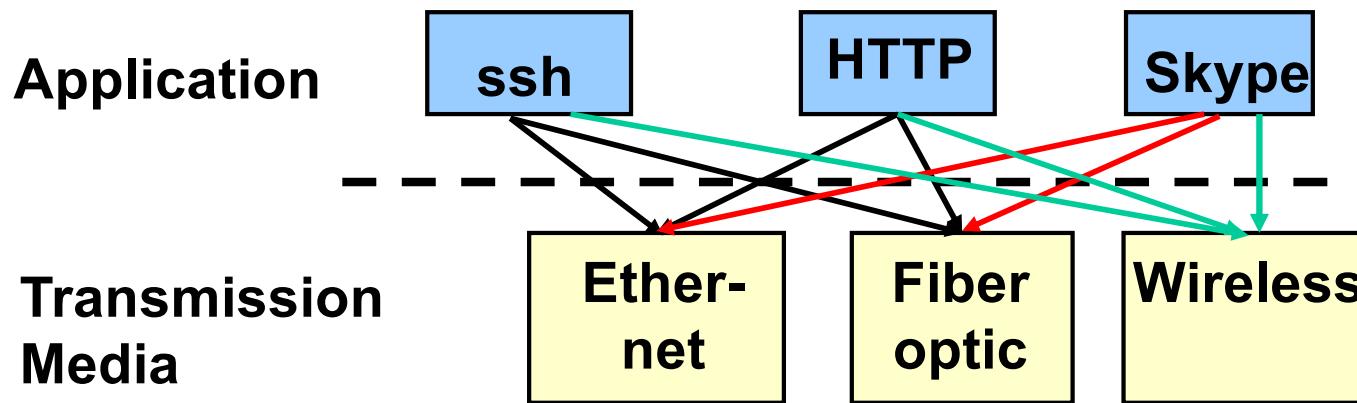


# Three Observations

- ❖ Each layer:
  - Depends on layer below
  - Supports layer above
  - Independent of others
- ❖ Multiple versions in layer
  - Interfaces differ somewhat
  - Components pick which lower-level protocol to use
- ❖ But only one IP layer
  - Unifying protocol



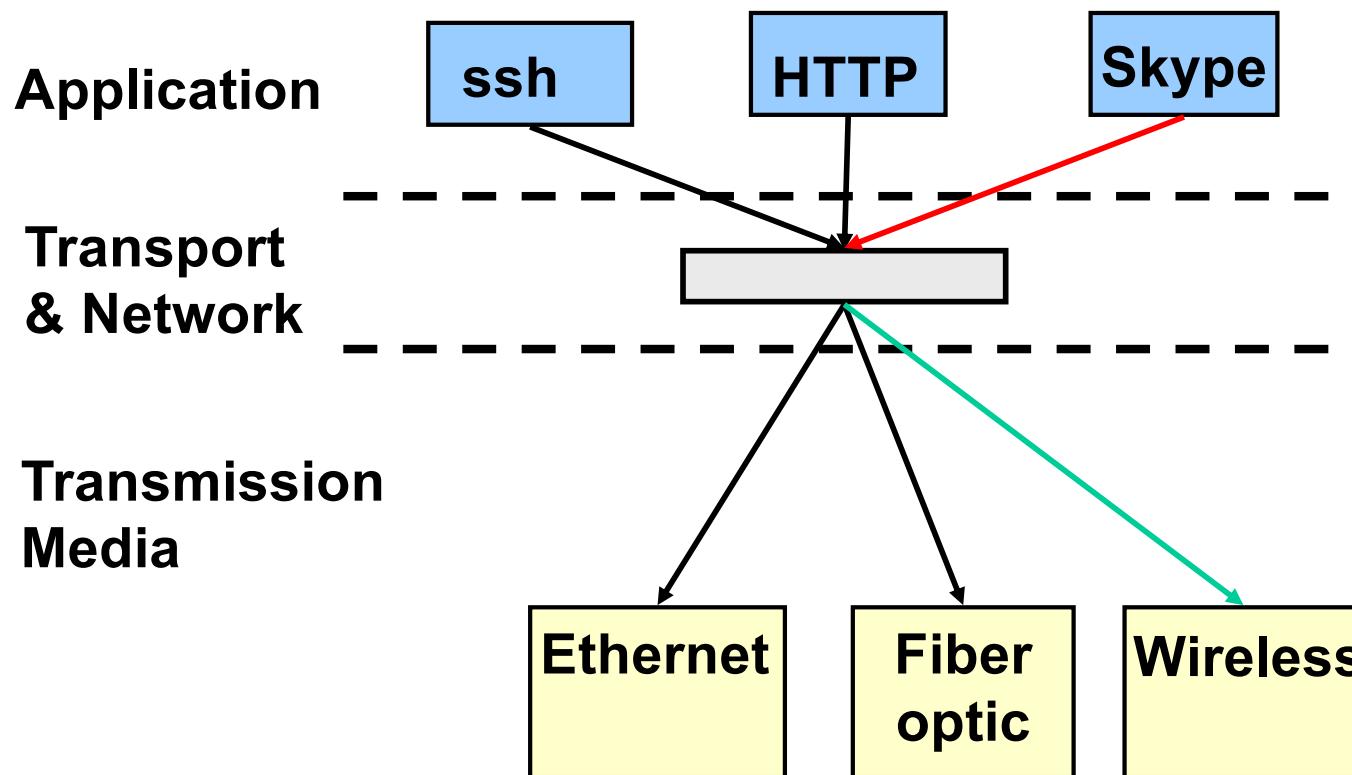
# An Example: No Layering



- ❖ No layering: each new application has to be **re-implemented for every network technology !**

# An Example: Benefit of Layering

- ❖ Introducing an intermediate layer provides a **common abstraction** for various network technologies



# Is Layering Harmful?

- ❖ Layer N may duplicate lower-level functionality
  - E.g., error recovery to retransmit lost data
- ❖ Information hiding may hurt performance
  - E.g., packet loss due to corruption vs. congestion
- ❖ Headers start to get large
  - E.g., typically, TCP + IP + Ethernet headers add up to 54 bytes
- ❖ Layer violations when the gains too great to resist
  - E.g., NAT
- ❖ Layer violations when network doesn't trust ends
  - E.g., Firewalls

# Distributing Layers Across Network

- ❖ Layers are simple if only on a single machine
  - Just stack of modules interacting with those above/below
- ❖ But we need to implement layers across machines
  - Hosts
  - Routers
  - Switches
- ❖ What gets implemented where?

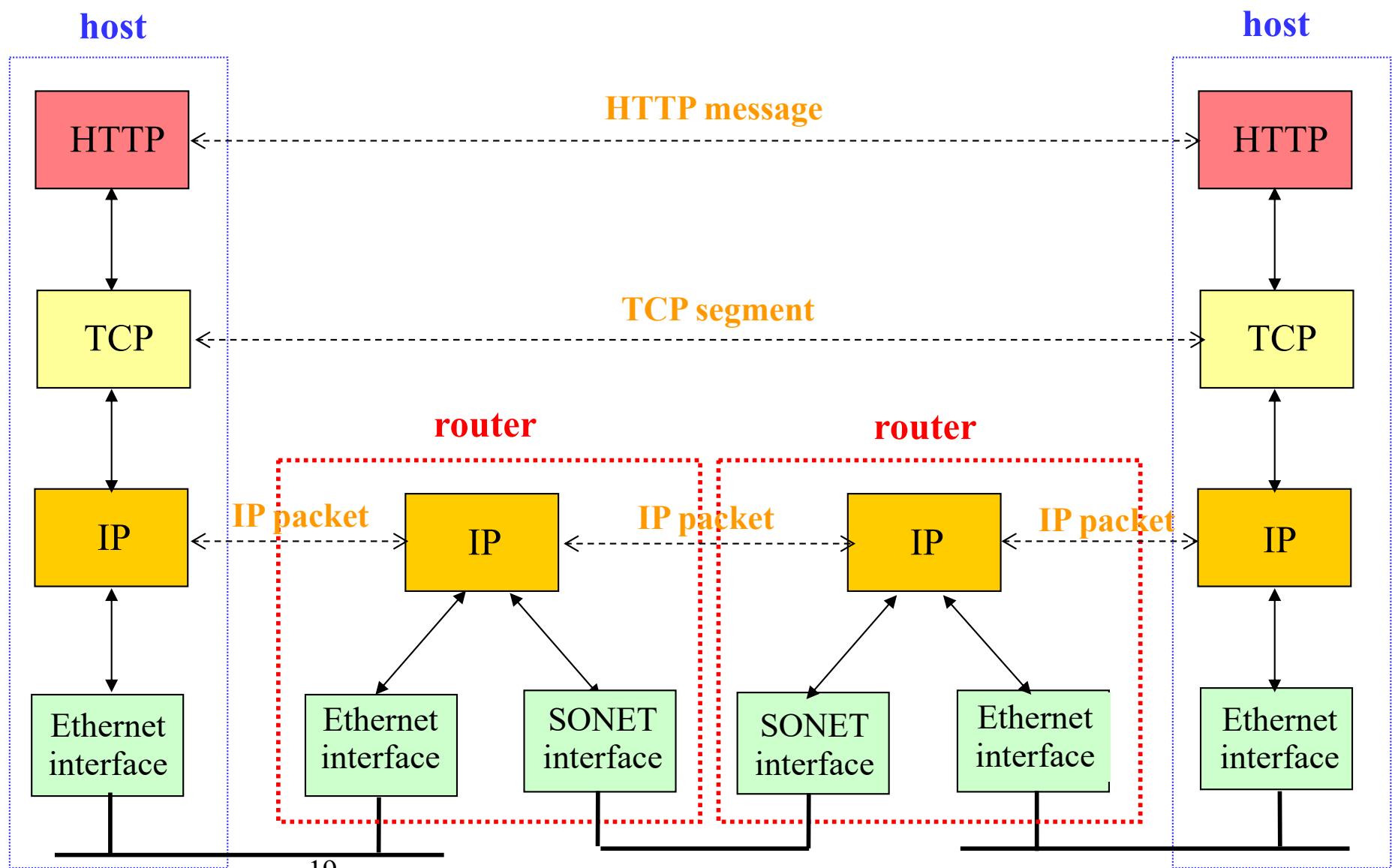
# What Gets Implemented on Host?

- ❖ Hosts have applications that generate data/messages that are eventually put out on wire
- ❖ At receiver host bits arrive on wire, must make it up to application
- ❖ Therefore, all layers must exist at host!

# What Gets Implemented on Router?

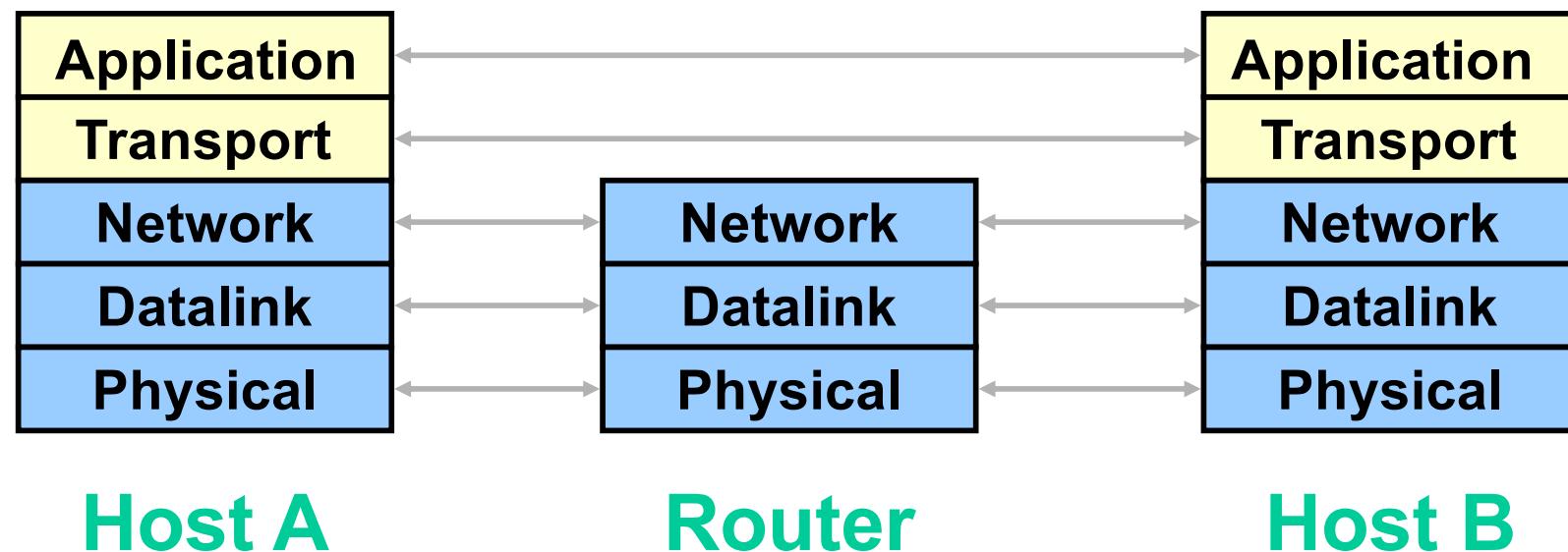
- ❖ Bits arrive on wire
  - Physical layer necessary
- ❖ Packets must be delivered to next-hop
  - datalink layer necessary
- ❖ Routers participate in global delivery
  - Network layer necessary
- ❖ Routers don't support reliable delivery
  - Transport layer (and above) **not** supported

# Internet Layered Architecture



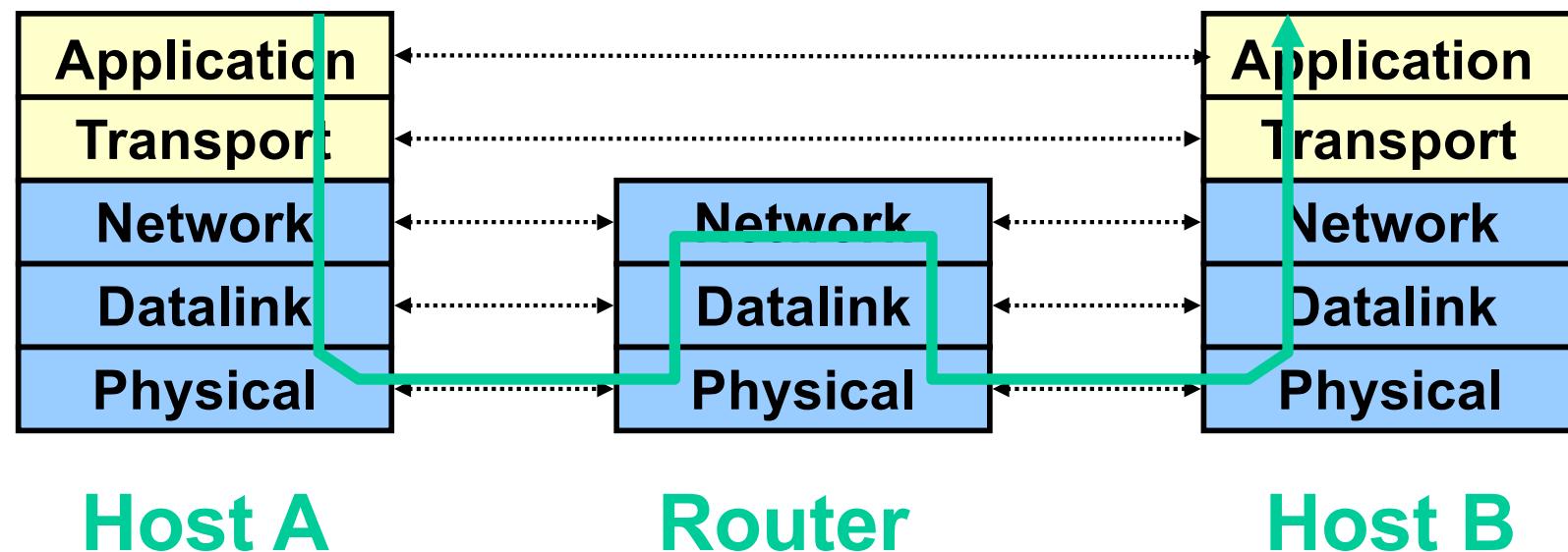
# Logical Communication

- ❖ Layers interacts with peer's corresponding layer



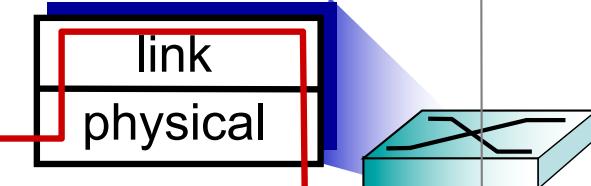
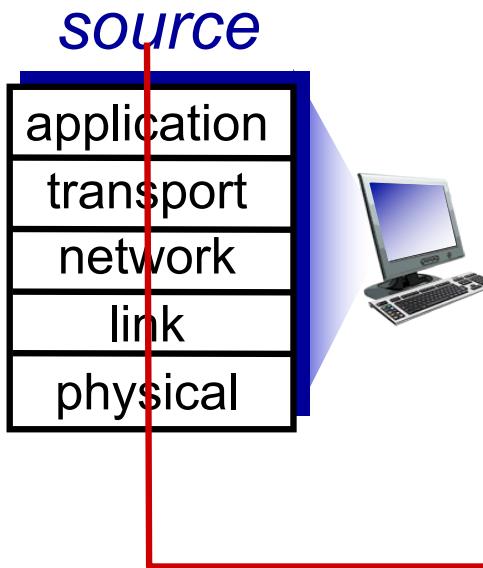
# Physical Communication

- ❖ Communication goes down to physical network
- ❖ Then from network peer to peer
- ❖ Then up to relevant layer

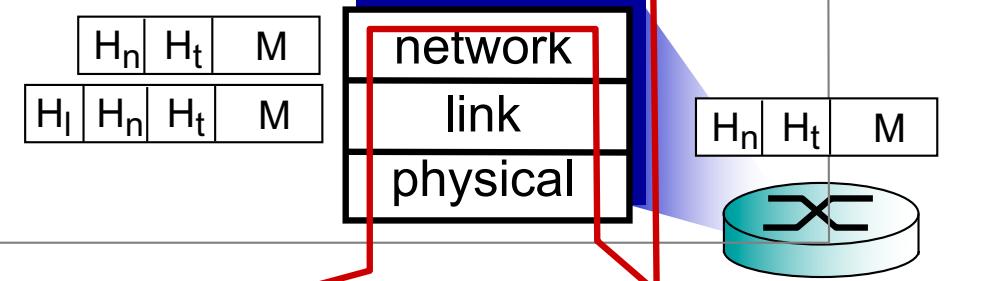
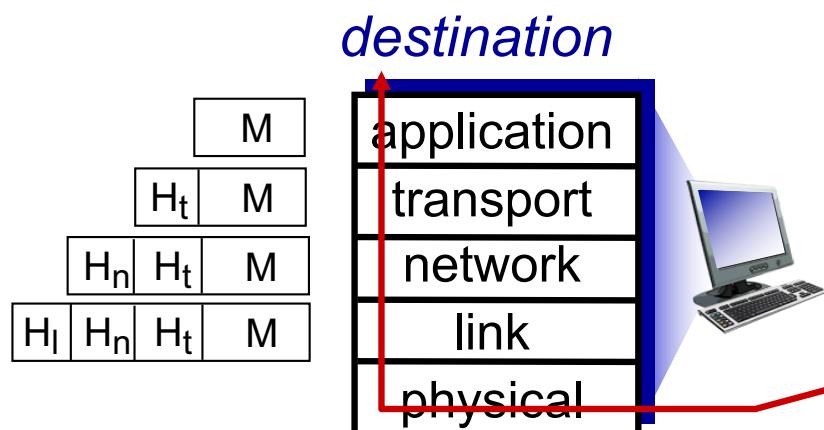


# Encapsulation

message	M
segment	H <sub>t</sub> M
datagram	H <sub>n</sub> H <sub>t</sub> M
frame	H <sub>l</sub> H <sub>n</sub> H <sub>t</sub> M



switch



router



## Quiz: Layering

What are two benefits of using a layered network model ? (Choose two)

- A. It makes it easy to introduce new protocols
- B. It speeds up packet delivery
- C. It allows us to have many different packet headers
- D. It prevents technology in one layer from affecting other layers
- E. It creates many acronyms
- F. It reminds me of cake

**Answer: A + D**

Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)

# I. Introduction: 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

Self study

# Computer Networks and Applications

COMP 3331/COMP 9331

Week 3

Application Layer (DNS, P2P, Video Streaming and CDN)

**Reading Guide: Chapter 2, Sections 2.4 -2.7**

# Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

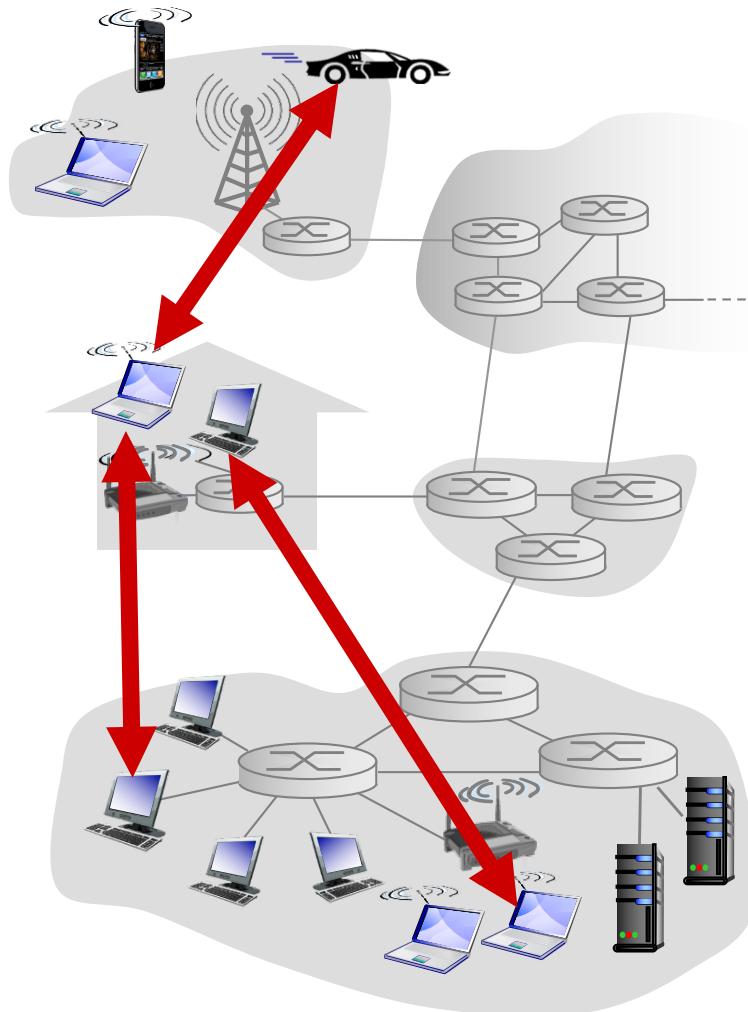
2.7 socket programming with UDP and TCP

# Pure P2P architecture

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

## *examples:*

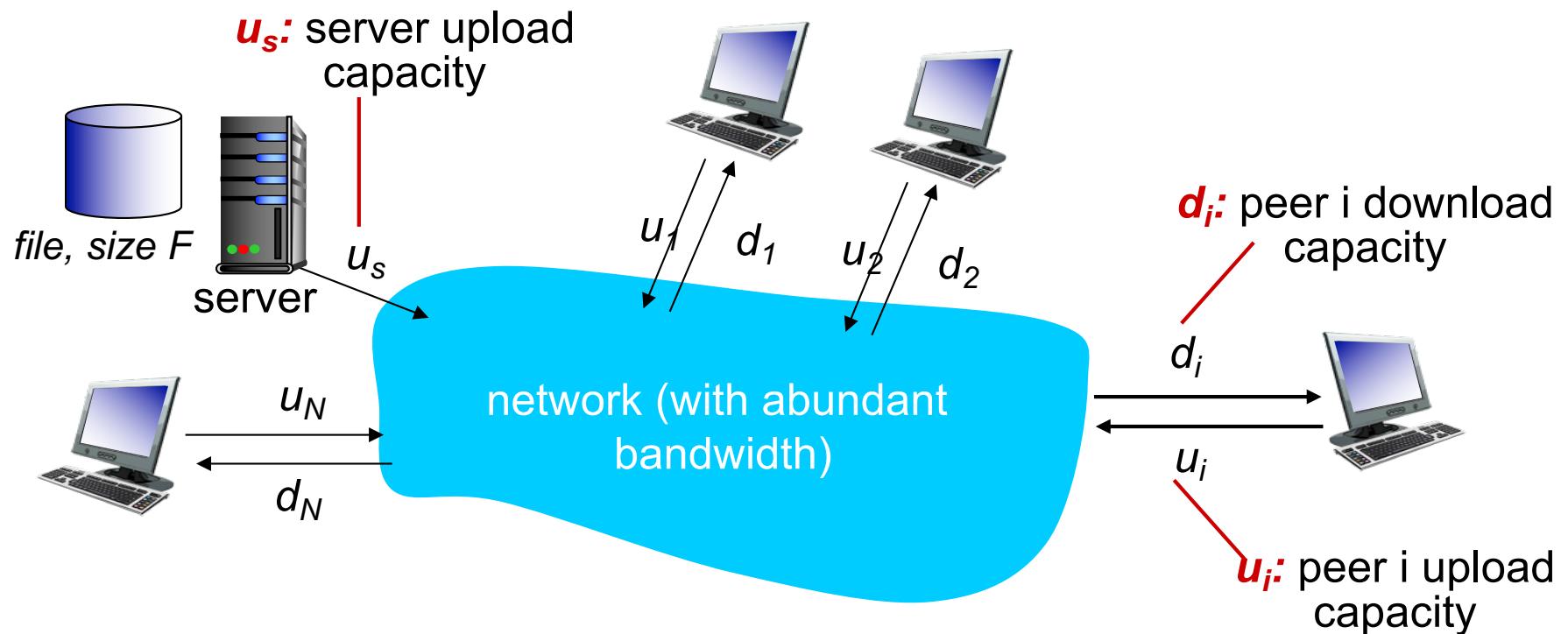
- file distribution  
(BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)
- Cryptocurrency  
(BitCoin)



# File distribution: client-server vs P2P

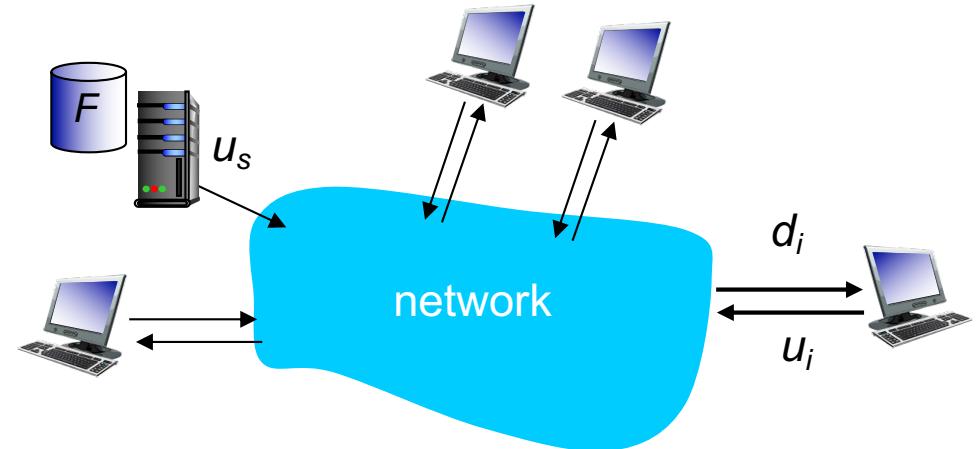
Question: how much time to distribute file (size  $F$ ) from one server to  $N$  peers?

- peer upload/download capacity is limited resource



# File distribution time: client-server

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



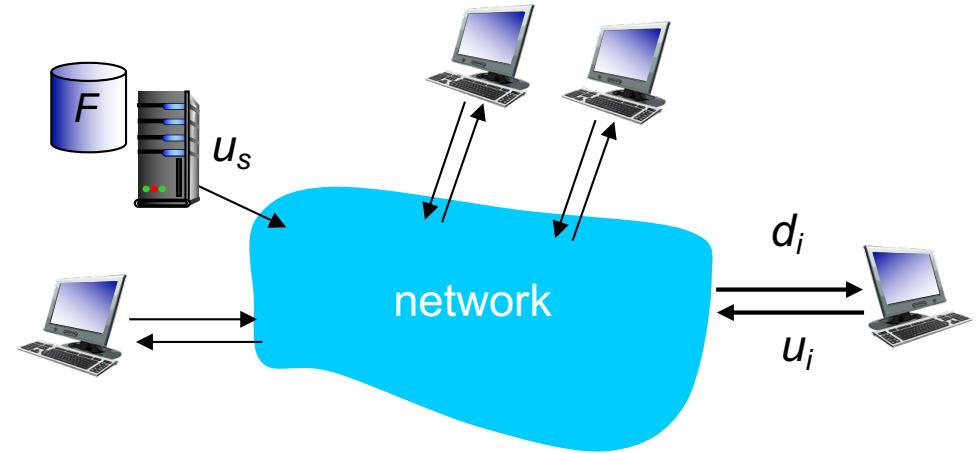
*time to distribute  $F$   
to  $N$  clients using  
client-server approach*

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

increases linearly in  $N$

# File distribution time: P2P

- ❖ **server transmission:** must upload at least one copy
  - time to send one copy:  $F/u_s$
- ❖ **client:** each client must download file copy
  - client download time:  $F/d_{\min}$
- ❖ **clients:** as aggregate must download  $NF$  bits
  - max upload rate (limiting max download rate) is  $u_s + \sum u_i$



time to distribute  $F$   
to  $N$  clients using  
P2P approach

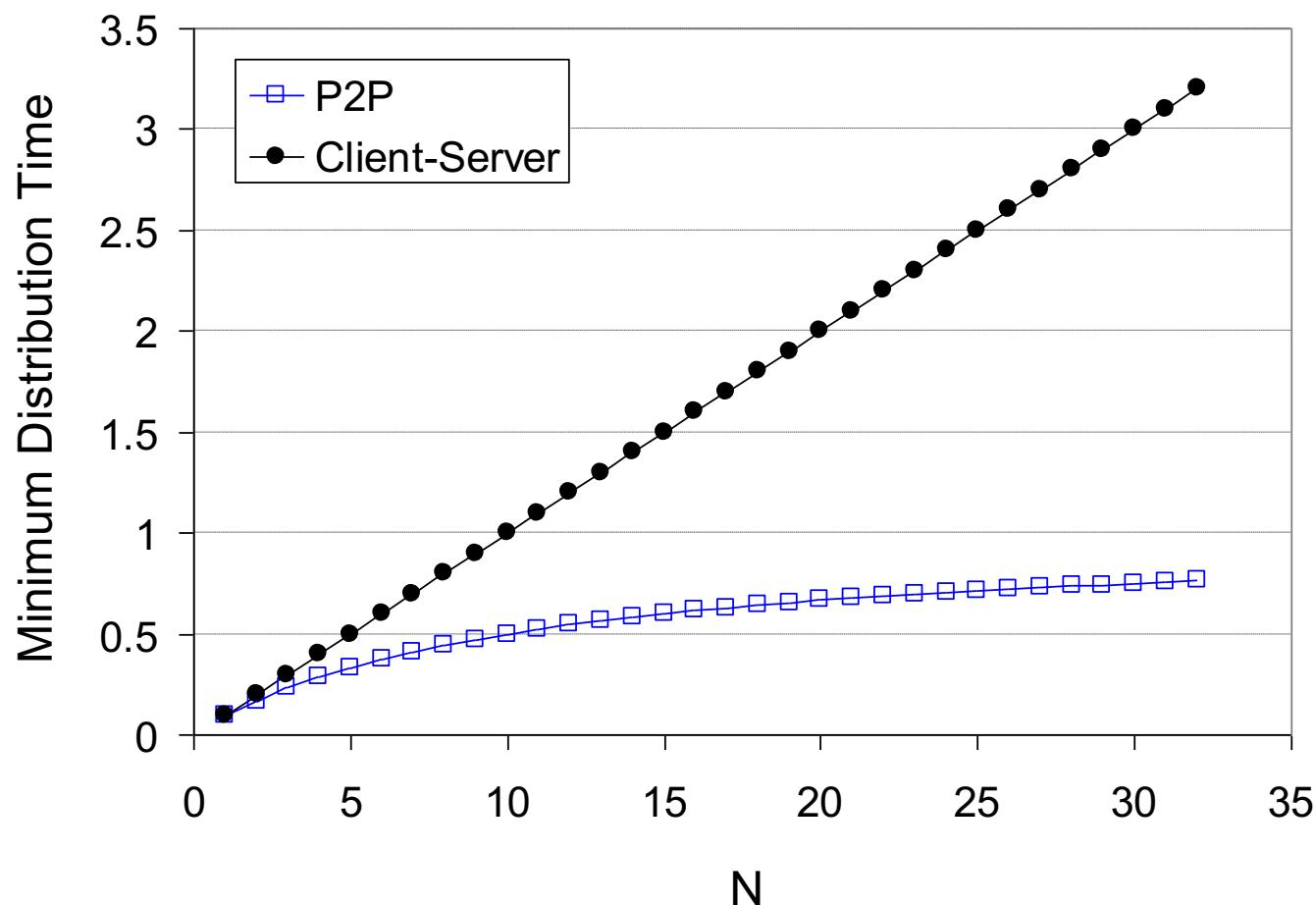
$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum_{i=1}^N u_i)\}$$

increases linearly in  $N$  ...

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

# Client-server vs. P2P: example

client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$

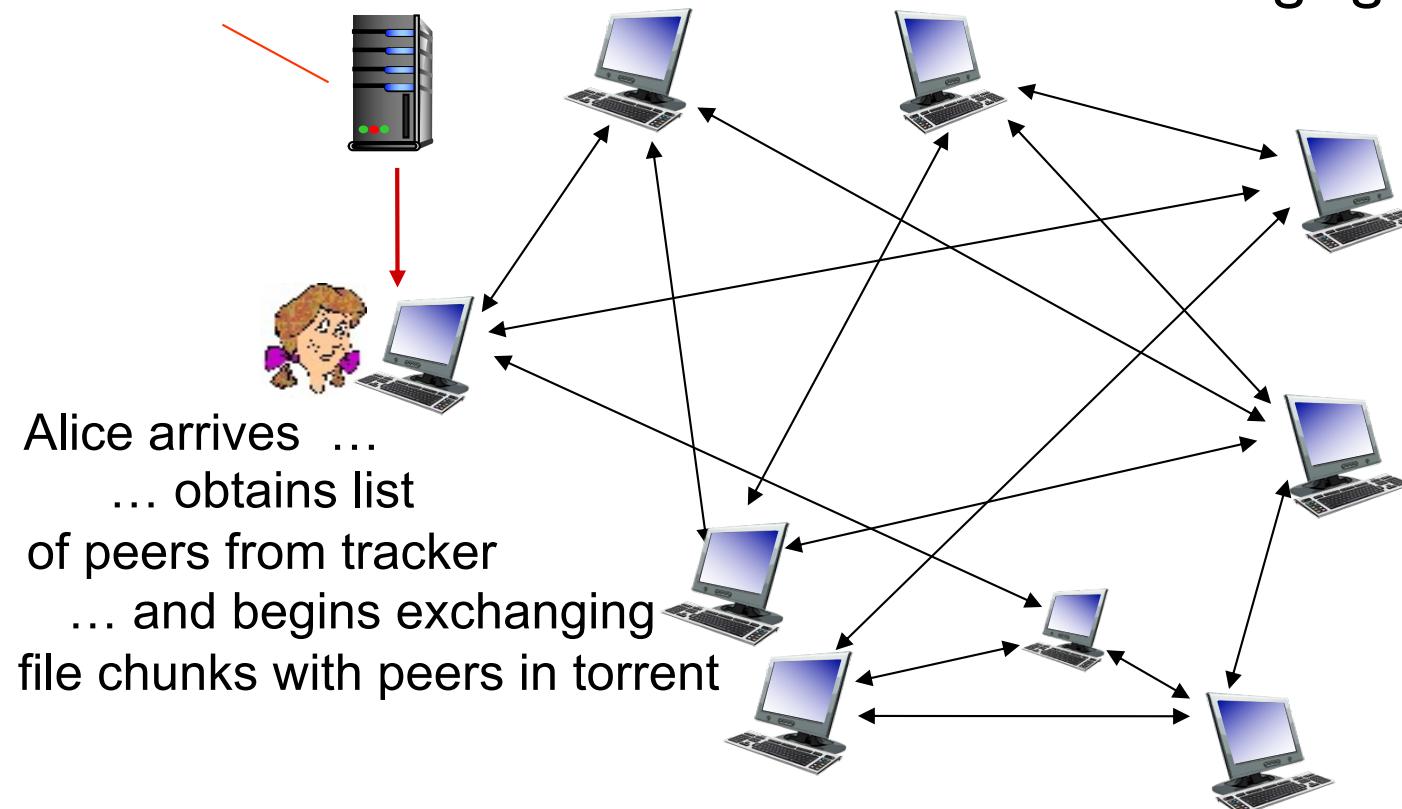


# P2P file distribution: BitTorrent

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

*tracker*: tracks peers  
participating in torrent

*torrent*: group of peers  
exchanging chunks of a file



# .torrent files

- ❖ Contains address of trackers for the file
  - Where can I find other peers?
- ❖ Contain a list of file chunks and their cryptographic hashes
  - This ensures that chunks are not modified

Title

The Boys Season 2

Walking Dead Season 10

Game of Thrones Season 8

Trackers

Tracker1-url

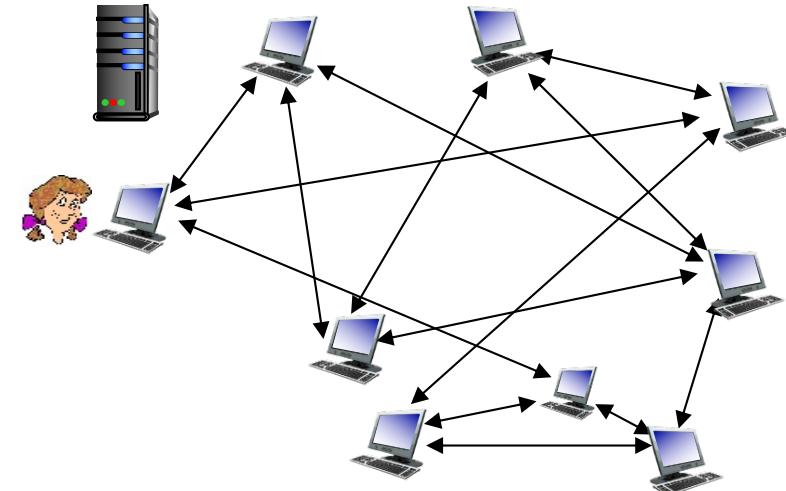
Tracker2-url

Tracker2-url, Tracker3-url

# P2P file distribution: BitTorrent

- ❖ peer joining torrent:

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



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

# BitTorrent: requesting, sending file chunks

## *requesting chunks:*

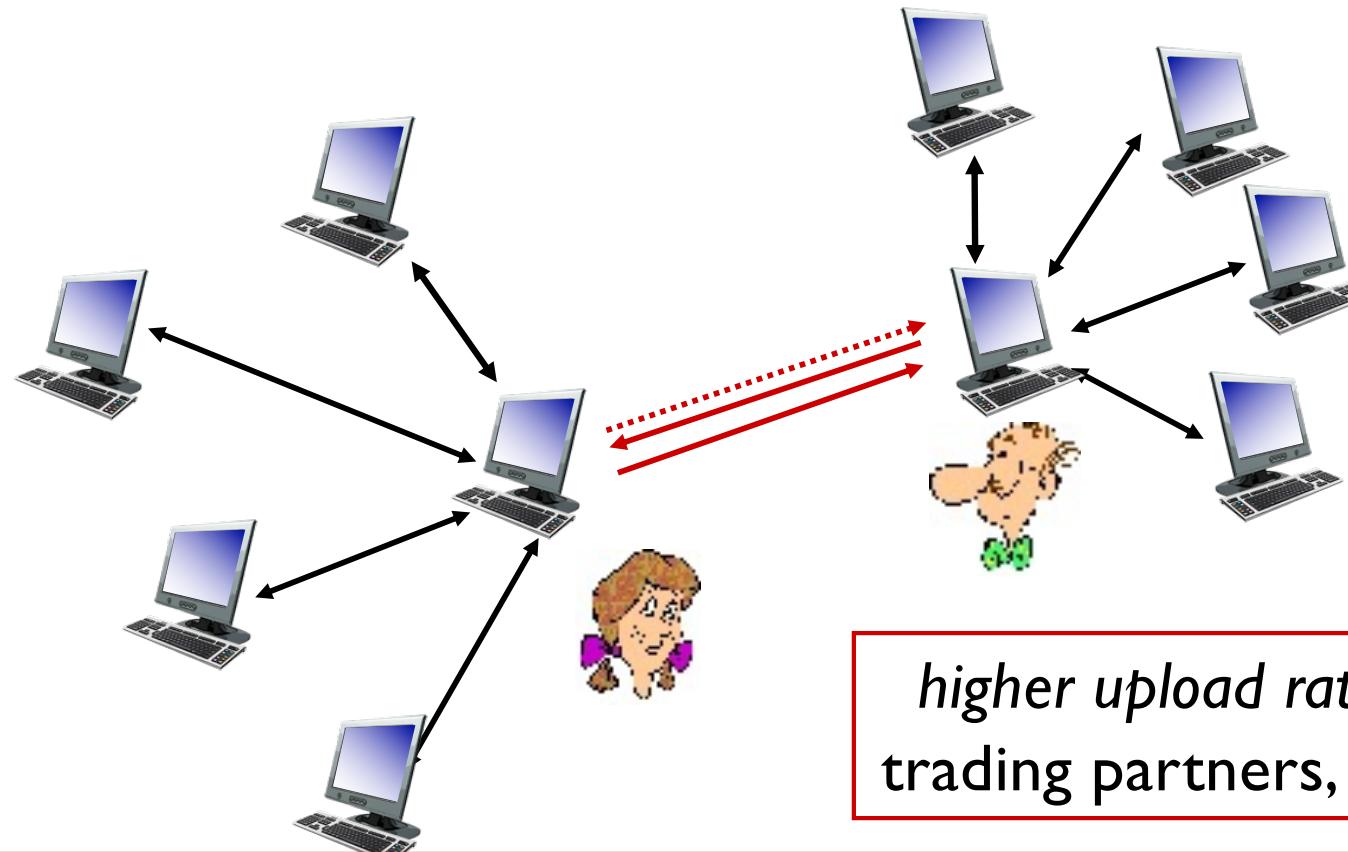
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first
- ❖ **Q:** Why rarest first?

## *sending chunks: tit-for-tat*

- ❖ Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every 10 secs
- ❖ every 30 secs: randomly select another peer, starts sending chunks
  - “optimistically unchoke” this peer
  - newly chosen peer may join top 4

# BitTorrent: tit-for-tat

- (1) Alice “optimistically unchoke” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice’s top-four providers



# Distributed Hash Table (DHT)

- ❖ DHT: a *distributed P2P database*
- ❖ database has **(key, value)** pairs; examples:
  - key: TFN number; value: human name
  - key: file name; value: IP addresses of peers (BT Tracker)
- ❖ Distribute the **(key, value)** pairs over many peers
- ❖ a peer **queries** DHT with key
  - DHT returns values that match the key
- ❖ peers can also **insert** **(key, value)** pairs

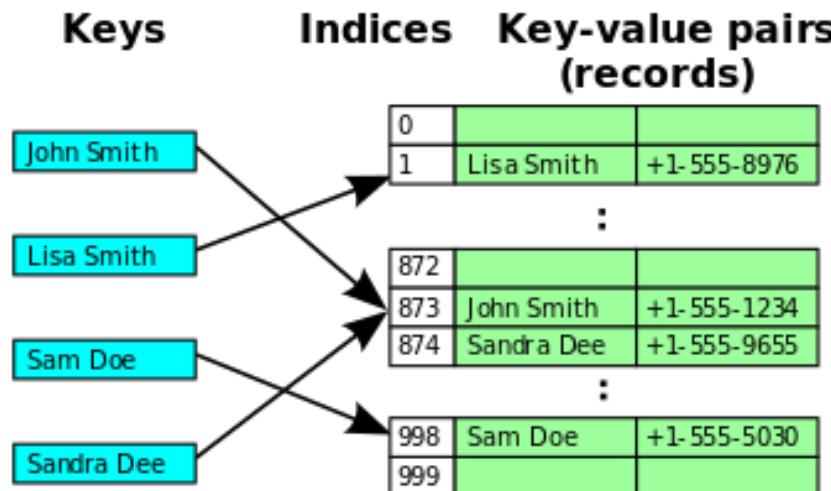
Content available in 6<sup>th</sup> Edition of the textbook Section 2.6.2, Added to Lecture Notes

# Q: how to assign keys to peers?

- ❖ basic idea:
  - convert each key to an integer
  - Assign integer value to each peer
  - put (key, value) pair in the peer that is **closest** to the key

# DHT identifiers: Consistent Hashing

- ❖ assign integer identifier to each peer in range  $[0, 2^n - 1]$  for some  $n$ -bit hash function
  - E.g., node ID is hash of its IP address
- ❖ require each key to be an integer in **same range**
- ❖ to get integer key, hash original key
  - e.g., key = **hash**("The Boys Season 2")
  - therefore, it is referred to as a *distributed "hash" table*



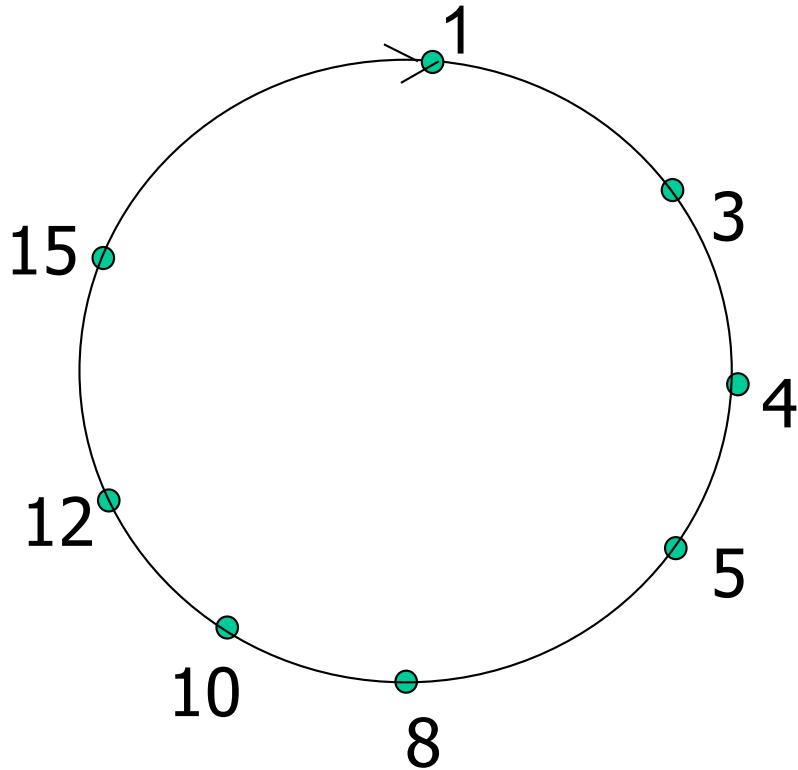
# Assign keys to peers

- ❖ rule: assign key to the peer that has the *closest* ID.
- ❖ common convention: closest is the *immediate successor* of the key.
- ❖ e.g.,  $n=4$ ; all peers & key identifiers are in the range [0-15], peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

Question: How is the peer-to-peer network organised?

One way could be to require each peer to be aware of every other peer, but this would not scale.

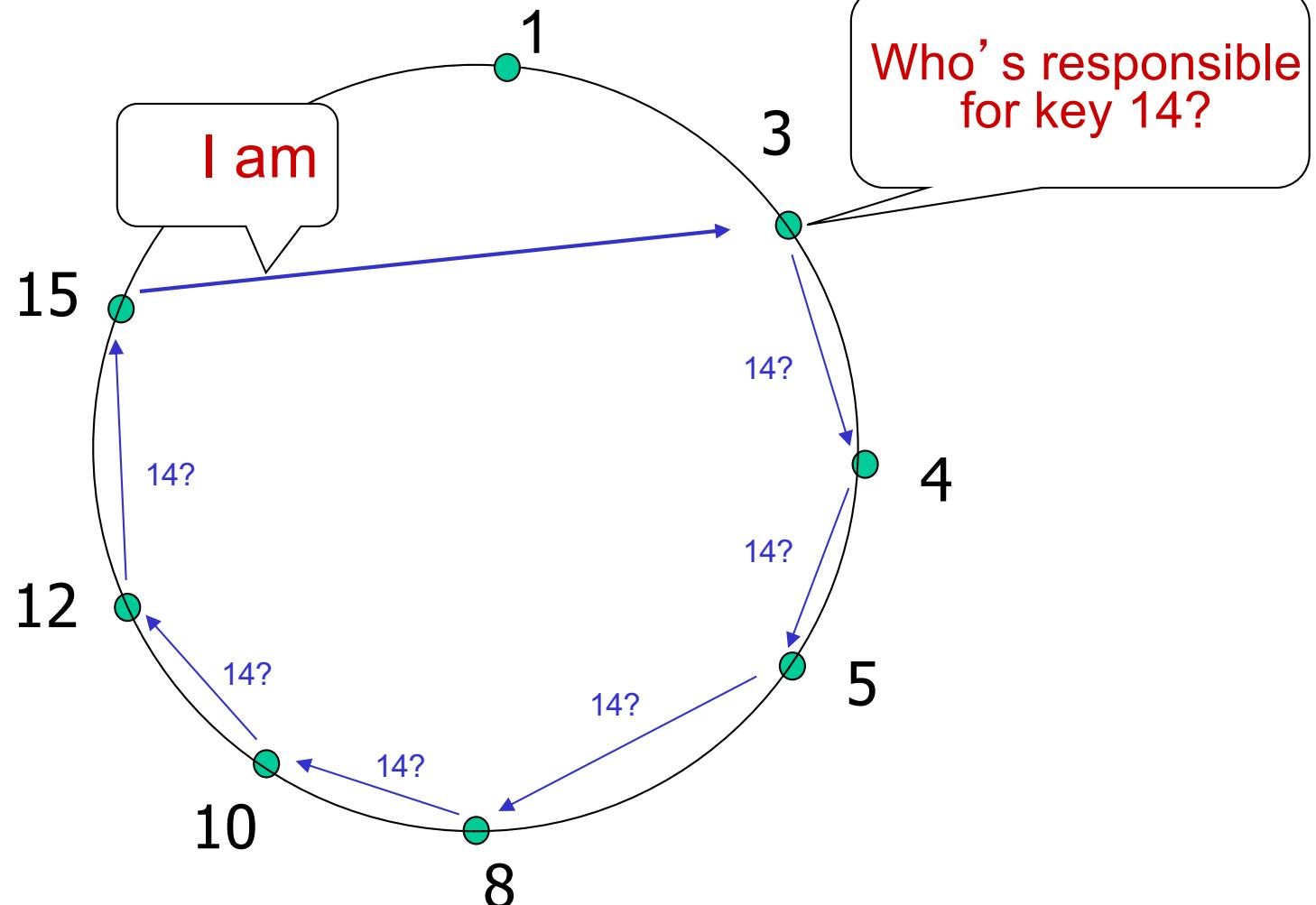
# Circular DHT (I)



- ❖ each peer *only* aware of immediate successor and predecessor.
- ❖ “overlay network”

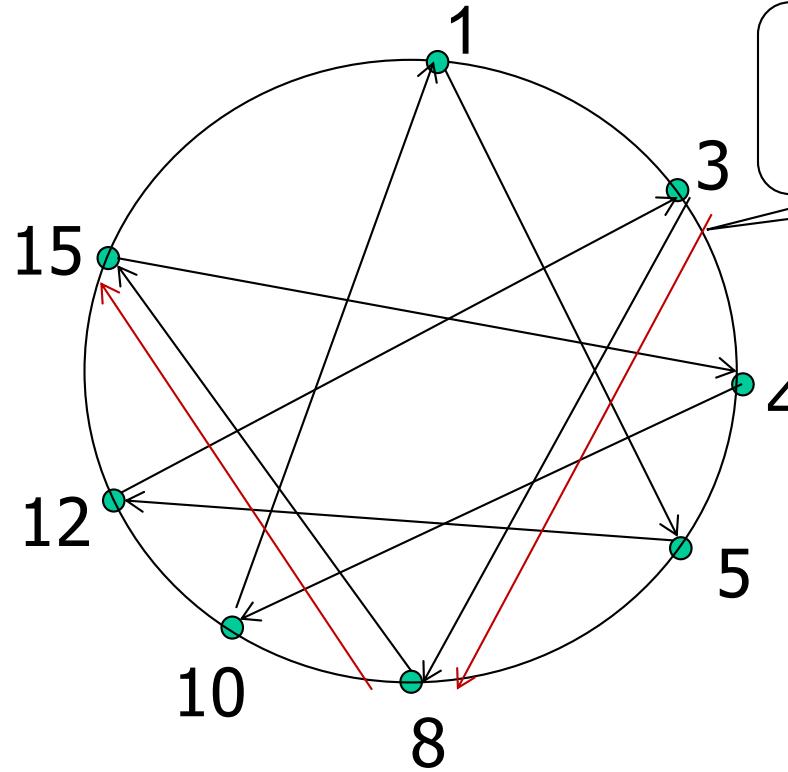
# Circular DHT (2)

Define closest as closest successor



- ❖ Each peer maintains 2 neighbours
- ❖ In this example, 6 query messages are sent
- ❖ Worst case:  $N$  messages, Average:  $N/2$  messages

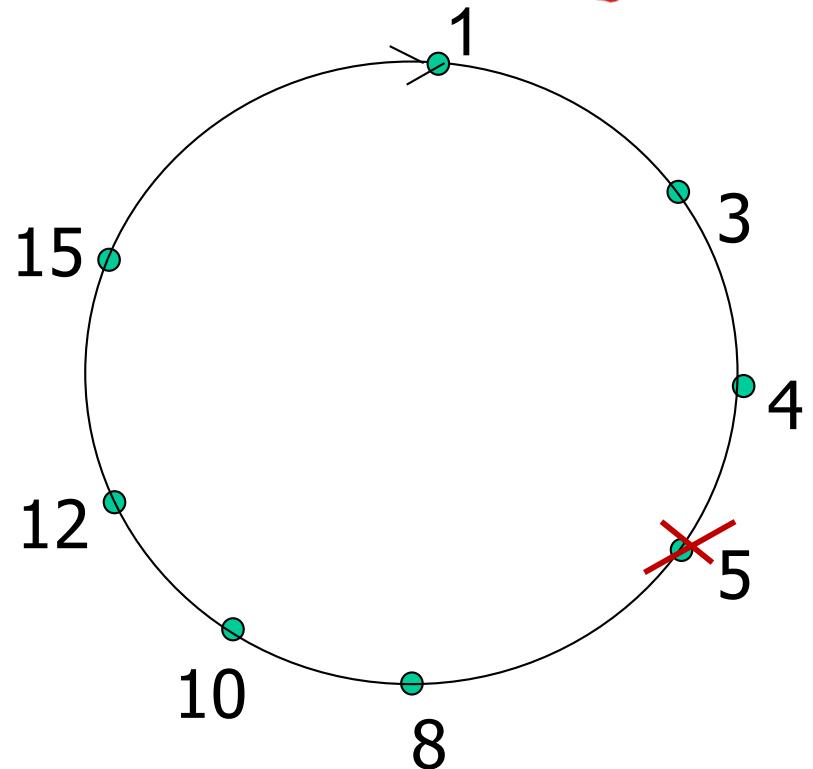
# Circular DHT with shortcuts



3 has shortcut to 8  
8 has shortcut to 15  
and so on

- ❖ each peer keeps track of IP addresses of predecessor, successor, short cuts
- ❖ reduced from 6 to 2 messages.
- ❖ possible to design shortcuts so  $O(\log N)$  neighbours,  $O(\log N)$  messages in query

# Peer churn



## handling peer churn:

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

*example: peer 5 abruptly leaves*

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

# More DHT info

- ❖ How do nodes join?
- ❖ How does cryptographic hashing work?
- ❖ How much state does each node store?

Research Papers (on the webpage):  
Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications  
**NOT MANDATORY READING**



# Quiz: BitTorrent

- ❖ BitTorrent uses tit-for-tat in each round to
  - a) Determine which chunks to download
  - b) Determine from which peers to download chunks
  - c) Determine to which peers to upload chunks
  - d) Determine which peers to report to the tracker as uncooperative
  - e) Determine whether or how long it should stay after completing download

**Answer: c**

# Quiz: BitTorrent



- ❖ Suppose Todd joins a BitTorrent torrent, but he does not want to upload any data to any other peers. Todd claims that he can receive a complete copy of the file that is shared by the swarm. Is Todd's claim possible? Why or Why not (one short sentences)?

**ANSWER:** Yes. Todd may receive chunks through the optimistic unchoke process. However, it will take Todd a much longer time to obtain the file.

# Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 **video streaming and content distribution networks (CDNs)**

2.7 socket programming with UDP and TCP

# Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1.8B YouTube users, ~140M Netflix users
- challenge: scale - how to reach ~2B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure



# Multimedia: video

- ❖ video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- ❖ digital image: array of pixels
  - each pixel represented by bits
- ❖ coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

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



frame  $i$

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

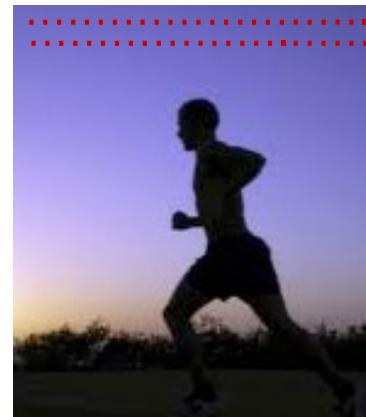


frame  $i+1$

# Multimedia: video

- **CBR: (constant bit rate):**  
video encoding rate fixed
- **VBR: (variable bit rate):**  
video encoding rate changes  
as amount of spatial,  
temporal coding changes
- **examples:**
  - MPEG I (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

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



frame  $i$

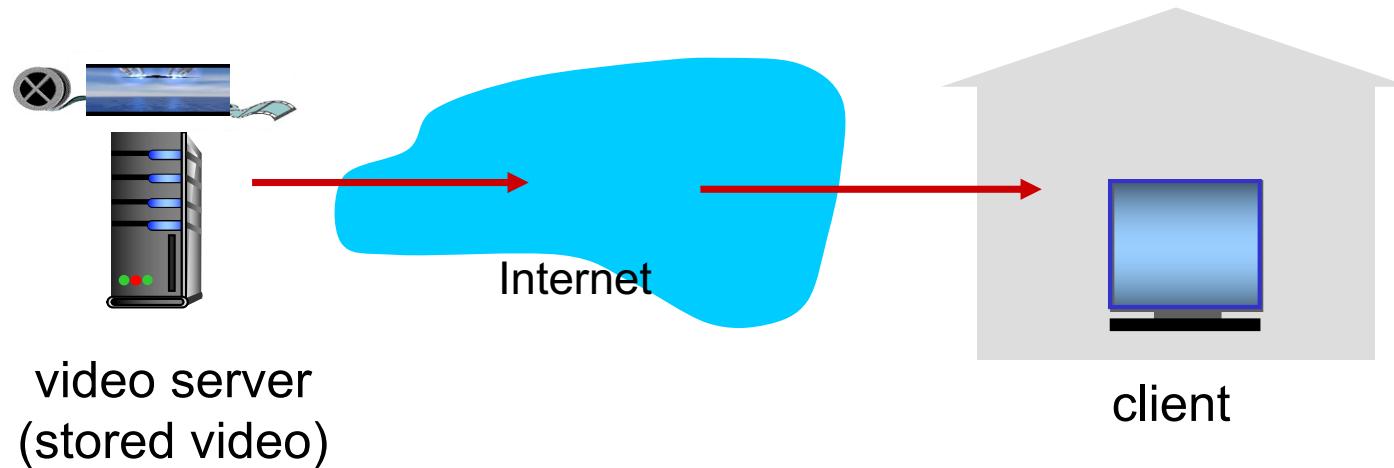
*temporal coding example:*  
instead of sending complete frame at  $i+1$ ,  
send only differences from frame  $i$



frame  $i+1$

# Streaming stored video:

simple scenario:

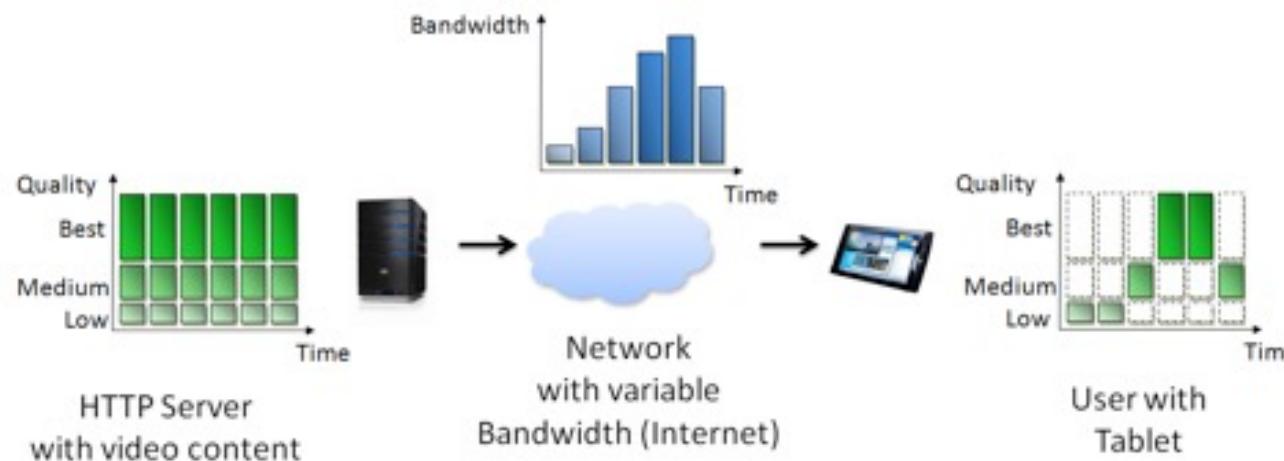


# Streaming multimedia: DASH

- ❖ *DASH: Dynamic, Adaptive Streaming over HTTP*
- ❖ *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- ❖ *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

# Streaming multimedia: DASH

- ❖ *DASH: Dynamic, Adaptive Streaming over HTTP*
- ❖ “*intelligence*” at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk from (can request from URL server that is “close” to client or has high available bandwidth)



# Content Distribution Networks (CDNs)

- *challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- *option 1*: single, large “mega-server”
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

....quite simply: this solution *doesn't scale*

# Content Distribution Networks (CDNs)

- ❖ *challenge*: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?
- ❖ *option 2*: store/serve multiple copies of videos at multiple geographically distributed sites (*CDN*)
  - *enter deep*: push CDN servers deep into many access networks
    - close to users
    - used by Akamai, thousands of locations
  - *bring home*: smaller number (10's) of larger clusters in IXPs near (but not within) access networks
    - used by Limelight

# An example

```
bash-3.2$ dig www.mit.edu

; <>> DiG 9.10.6 <>> www.mit.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17913
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 8, ADDITIONAL: 8

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags: udp: 4096
;; QUESTION SECTION:
;www.mit.edu.           IN      A

;; ANSWER SECTION:
www.mit.edu.          924    IN      CNAME   www.mit.edu.edgekey.net.
www.mit.edu.edgekey.net. 54    IN      CNAME   e9566.dscb.akamaiedge.net.
e9566.dscb.akamaiedge.net. 14    IN      A       23.77.154.132

;; AUTHORITY SECTION:
dscb.akamaiedge.net. 623    IN      NS      n0dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n2dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n7dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n6dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n1dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n3dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n5dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n4dscb.akamaiedge.net.

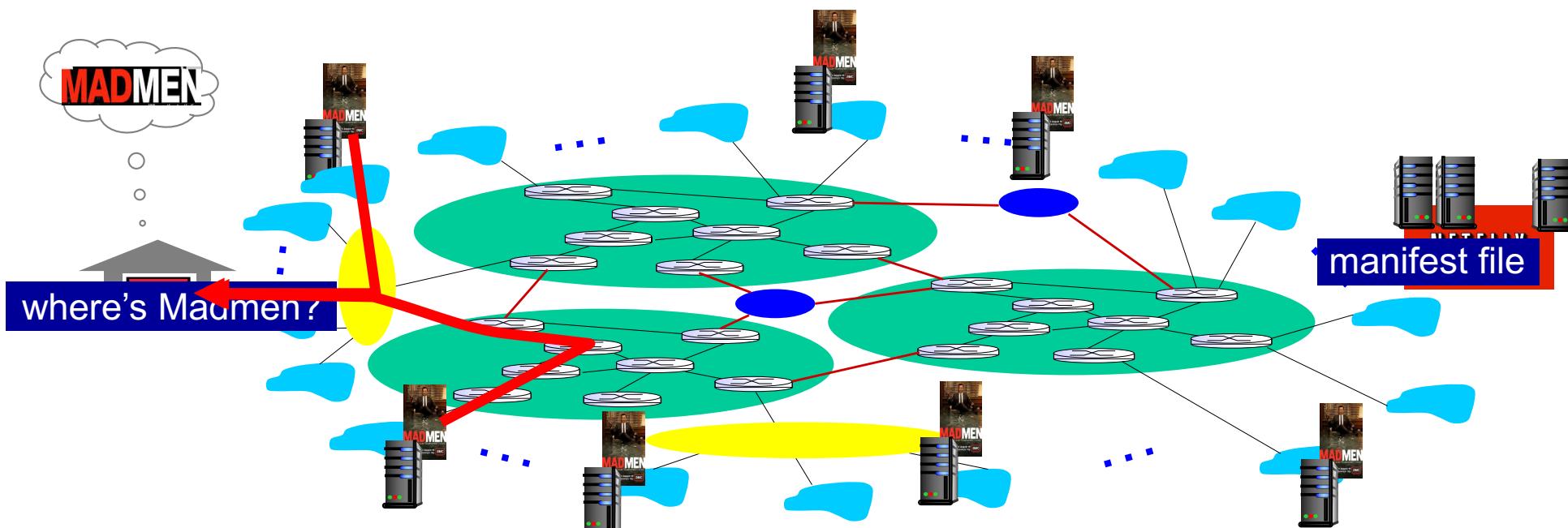
;; ADDITIONAL SECTION:
n0dscb.akamaiedge.net. 1241   IN      A       88.221.81.192
n0dscb.akamaiedge.net. 1124   IN      AAAA   2600:1480:e800::c0
n1dscb.akamaiedge.net. 842    IN      A       23.32.5.76
n2dscb.akamaiedge.net. 749    IN      A       23.32.5.84
n4dscb.akamaiedge.net. 1399   IN      A       23.32.5.177
n6dscb.akamaiedge.net. 702    IN      A       23.32.5.98
n7dscb.akamaiedge.net. 1208   IN      A       23.206.243.54

;; Query time: 46 msec
;; SERVER: 129.94.172.11#53(129.94.172.11)
;; WHEN: Mon Sep 28 13:15:28 AEST 2020
;; MSG SIZE  rcvd: 421
```

Many well-known sites are hosted by CDNs. A simple way to check using dig is shown here.

# Content Distribution Networks (CDNs)

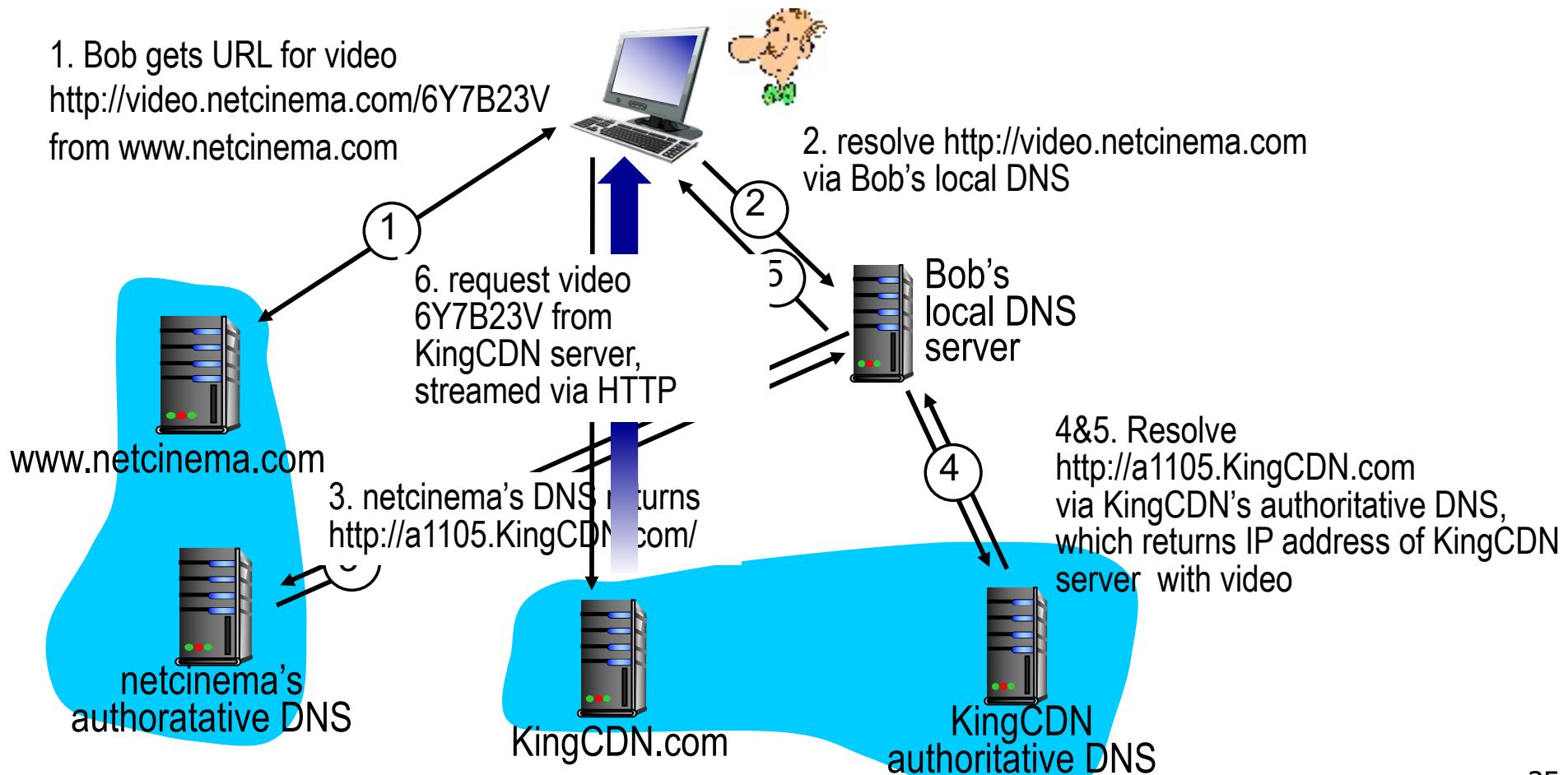
- CDN: stores copies of content at CDN nodes
  - e.g., Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



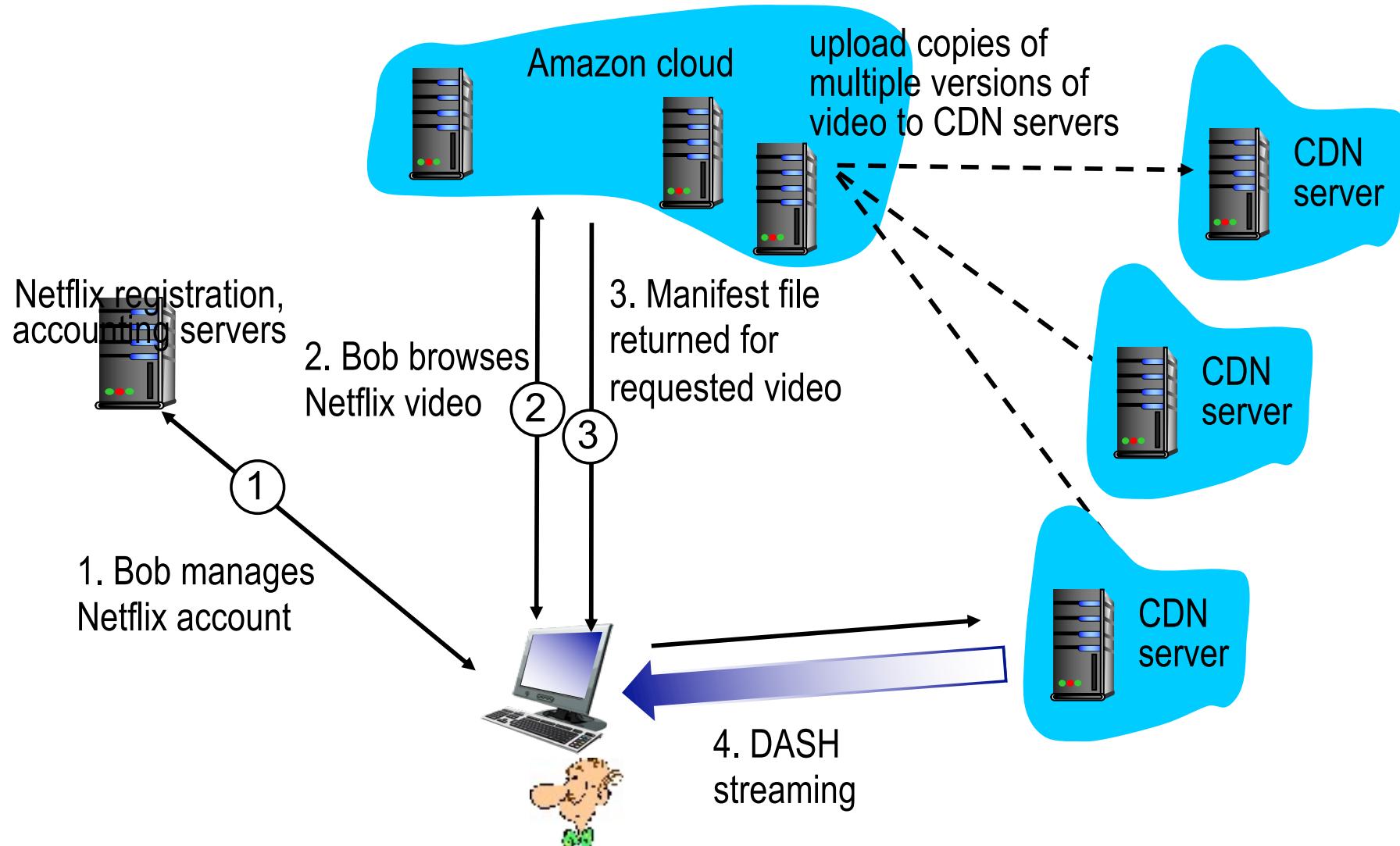
# CDN content access: a closer look

Bob (client) requests video <http://video.netcinema.com/6Y7B23V>

- video stored in CDN at managed by KingCDN.com

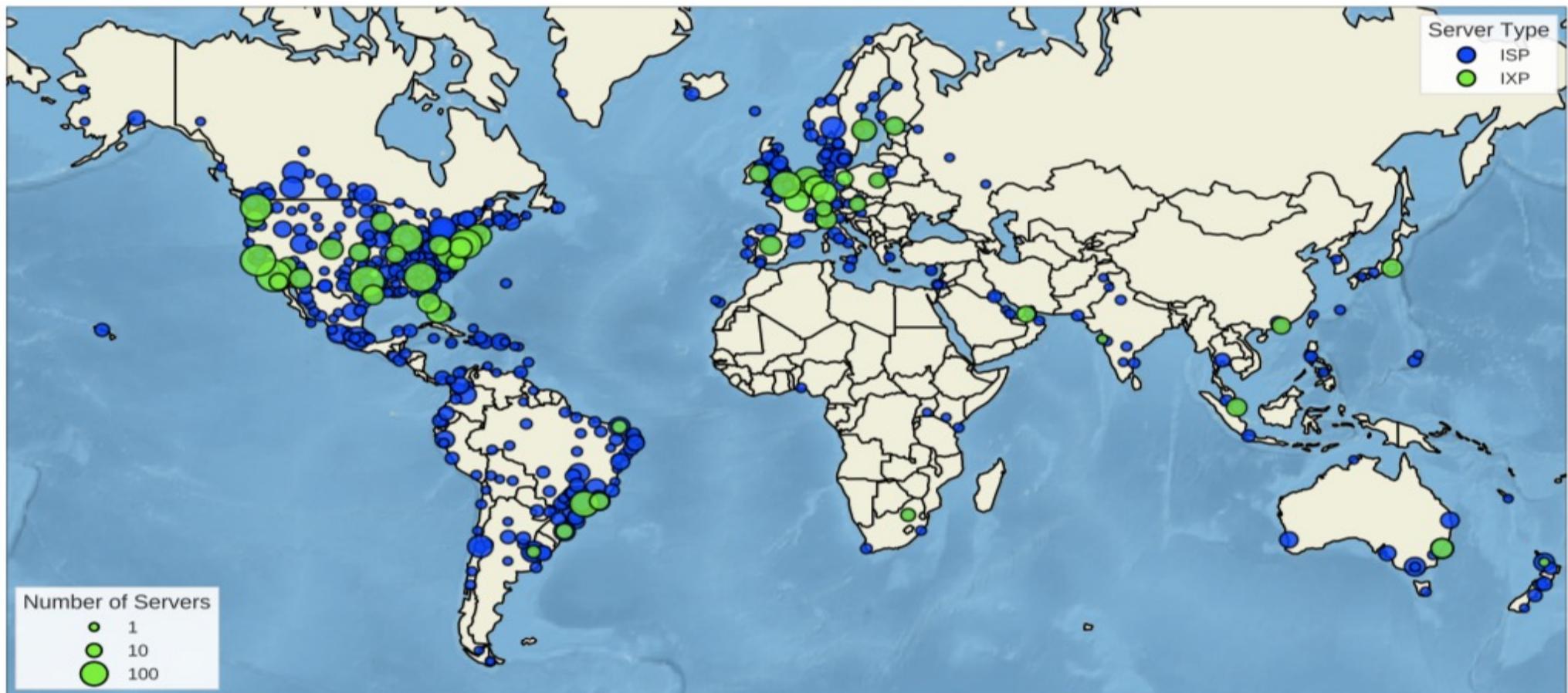


# Case study: Netflix



Uses Push caching (during offpeak)  
Preference to "deep inside" followed by "bring home"

# NetFlix servers (snap shot from Jan 2018)



Researchers from Queen Mary University of London (QMUL) traced server names that are sent to a user's computer every time they play content on Netflix to find the location of the 8492 servers (4152 ISP, 4340 IXP). They have been found to be scattered across 578 locations around the world.



## Quiz: CDN

- ❖ The role of the CDN provider's authoritative DNS name server in a content distribution network, simply described, is:
  - a) to provide an alias address for each browser access to the “origin server” of a CDN website
  - b) to map the query for each CDN object to the CDN server closest to the requestor (browser)
  - c) to provide a mechanism for CDN “origin servers” to provide paths for clients (browsers)
  - d) none of the above, CDN networks do not use DNS

## 2. Application Layer: outline

### 2.1 principles of network applications

- app architectures
- app requirements

### 2.2 Web and HTTP

### 2.3 electronic mail

- SMTP, POP3, IMAP

### 2.4 DNS

### 2.5 P2P applications

### 2.6 video streaming and content distribution networks (CDNs)

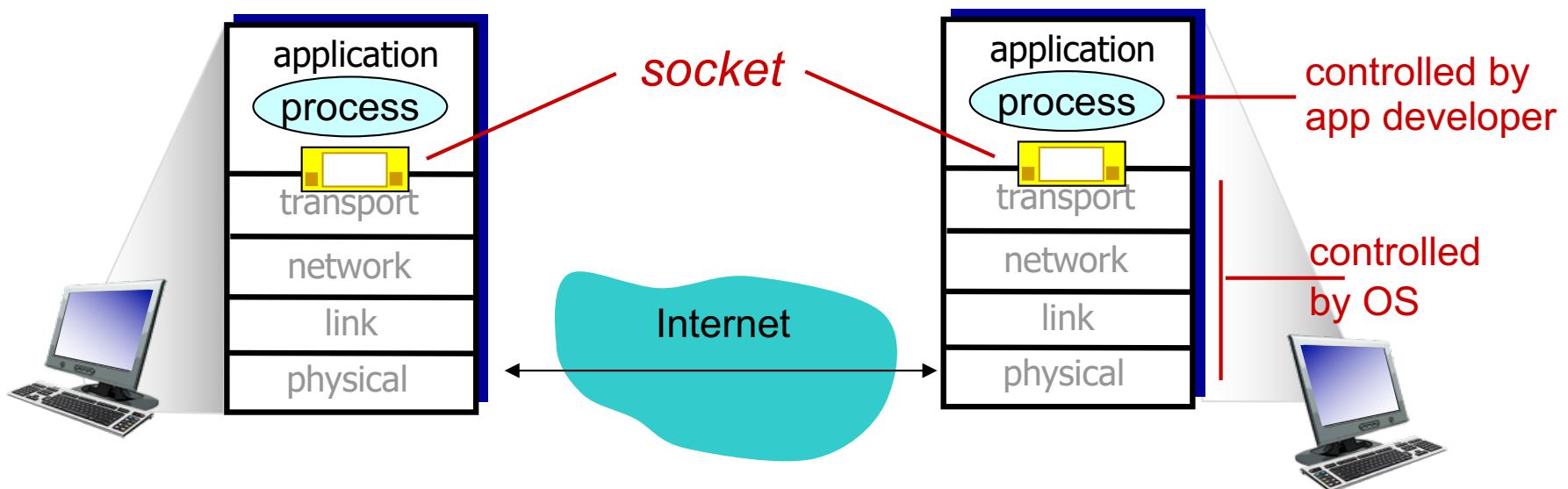
### 2.7 socket programming with UDP and TCP

Please see example code (C, Java, Python) on course website  
Labs 2 & 3 will include a socket programming exercise

# Socket programming

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

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



# Socket programming with UDP

UDP: no “connection” between client & server

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

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

Application viewpoint:

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

# Pseudo code UDP client

- ❖ Create socket
- ❖ Loop
  - (Send UDP datagram to known port and IP addr of server)
  - (Receive UDP datagram as a response from server)
- ❖ Close socket

# Pseudo code UDP server

- ❖ Create socket
- ❖ Bind socket to a specific port where clients can contact you
- ❖ Loop
  - (Receive UDP datagram from client X)
  - (Send UDP datagram as reply to client X)
- ❖ Close socket

Note: The IP address and port number of the client must be extracted from the client's message

# Socket programming with TCP

## client must contact server

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

## client contacts server by:

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

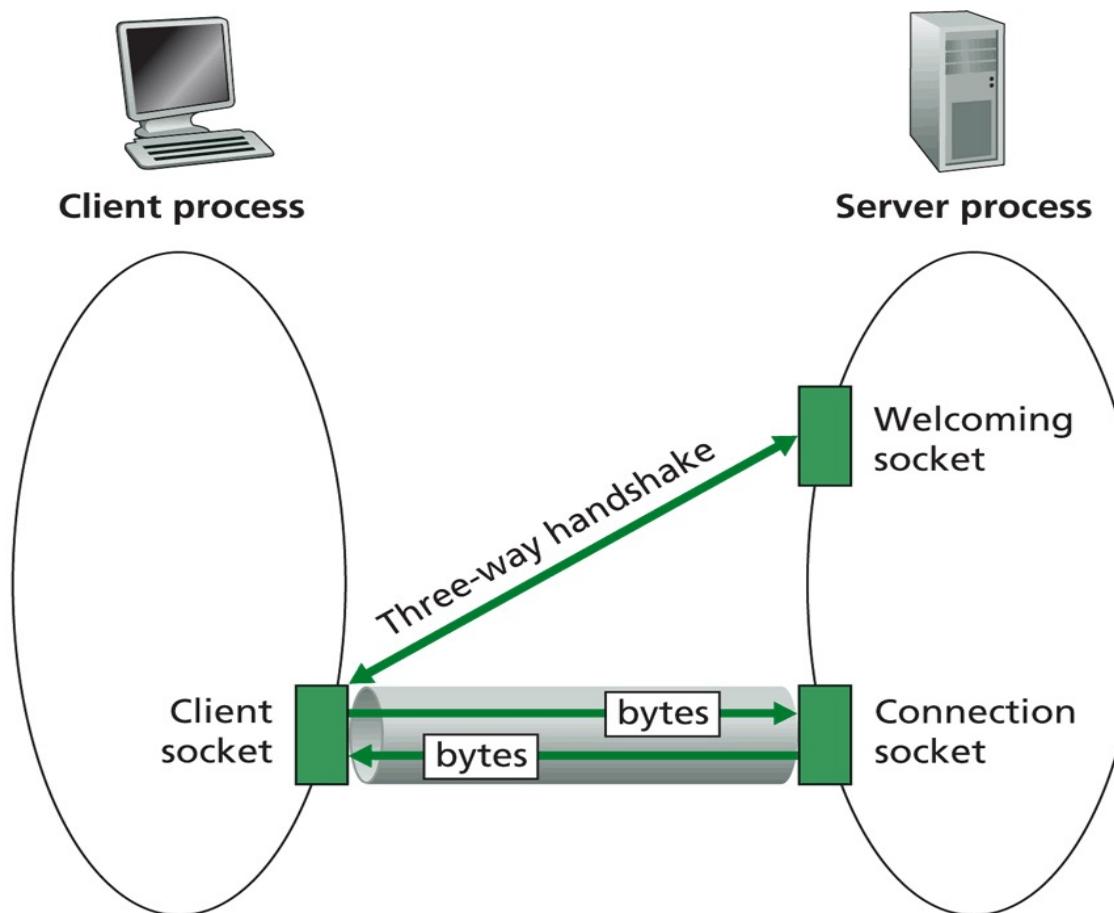
- ❖ when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client

- allows server to talk with multiple clients
- client-side port numbers used to distinguish clients (more later)

## application viewpoint:

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

# TCP Sockets



# Pseudo code TCP client

- ❖ Create socket (`ConnectionSocket`)
- ❖ Do an active connect specifying the IP address and port number of server
- ❖ Read and write data into `ConnectionSocket` to communicate with client
- ❖ Close `ConnectionSocket`

# Pseudo code TCP server

- ❖ Create socket (WelcomingSocket)
- ❖ Bind socket to a specific port where clients can contact you
- ❖ Register with the OS your willingness to listen on that socket for clients to contact you
- ❖ Loop
  - Accept new connection(ConnectionSocket)
  - Read and write data into ConnectionSocket to communicate with client
  - Close ConnectionSocket
- ❖ Close WelcomingSocket

# Queues

---

- ❖ While the server socket is busy, incoming connection requests are stored in a queue
- ❖ Once the queue fills up, further incoming connections are refused
- ❖ This is clearly a problem
  - Example: HTTP servers
- ❖ Solution
  - Concurrency

# Concurrent TCP Servers

- ❖ Benefit comes in ability to hand off interaction with a client to another process
- ❖ Parent process creates the WelcomingSocket and waits for clients to request connection
- ❖ When a connection request is received, fork off a child process to handle that connection so that the parent process can return to waiting for connections as soon as possible
- ❖ Multithreaded server: same idea, just spawn off another thread rather than a process

## 2. Application Layer: outline

### 2.1 principles of network applications

- app architectures
- app requirements

### 2.2 Web and HTTP

### 2.3 electronic mail

- SMTP, POP3, IMAP

### 2.4 DNS

### 2.5 P2P applications

### 2.6 video streaming and content distribution networks (CDNs)

### 2.7 socket programming with UDP and TCP

A nice overview <https://www.thegeeksearch.com/beginners-guide-to-dns/>

# DNS: domain name system

*people:* many identifiers:

- TFN, name, passport #

*Internet hosts, routers:*

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

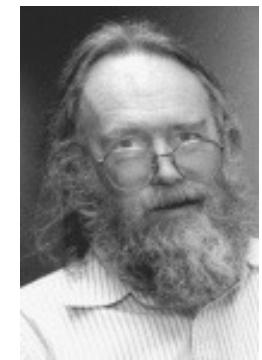
*Q:* how to map between IP address and name, and vice versa ?

*Domain Name System:*

- ❖ *distributed database* implemented in hierarchy of many *name servers*
- ❖ *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”

# DNS: History

- ❖ Initially all host-address mappings were in a hosts.txt file (in /etc/hosts):
  - Maintained by the Stanford Research Institute (SRI)
  - Changes were submitted to SRI by email
  - New versions of hosts.txt periodically FTP'd from SRI
  - An administrator could pick names at their discretion
- ❖ As the Internet grew this system broke down:
  - SRI couldn't handle the load; names were not unique; hosts had inaccurate copies of hosts.txt
- ❖ The Domain Name System (DNS) was invented to fix this



Jon Postel

<http://www.wired.com/2012/10/joe-postel/>

# DNS: services, structure

## *DNS services*

- ❖ hostname to IP address translation
- ❖ Indirection
- ❖ host aliasing
  - canonical, alias names
- ❖ mail server aliasing
- ❖ load distribution
  - replicated Web servers: many IP addresses correspond to one name
  - Content Distribution Networks: use IP address of requesting host to find best suitable server
    - Example: closest, least-loaded, etc.

## *why not centralize DNS?*

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

A: *doesn't scale!*

# Goals

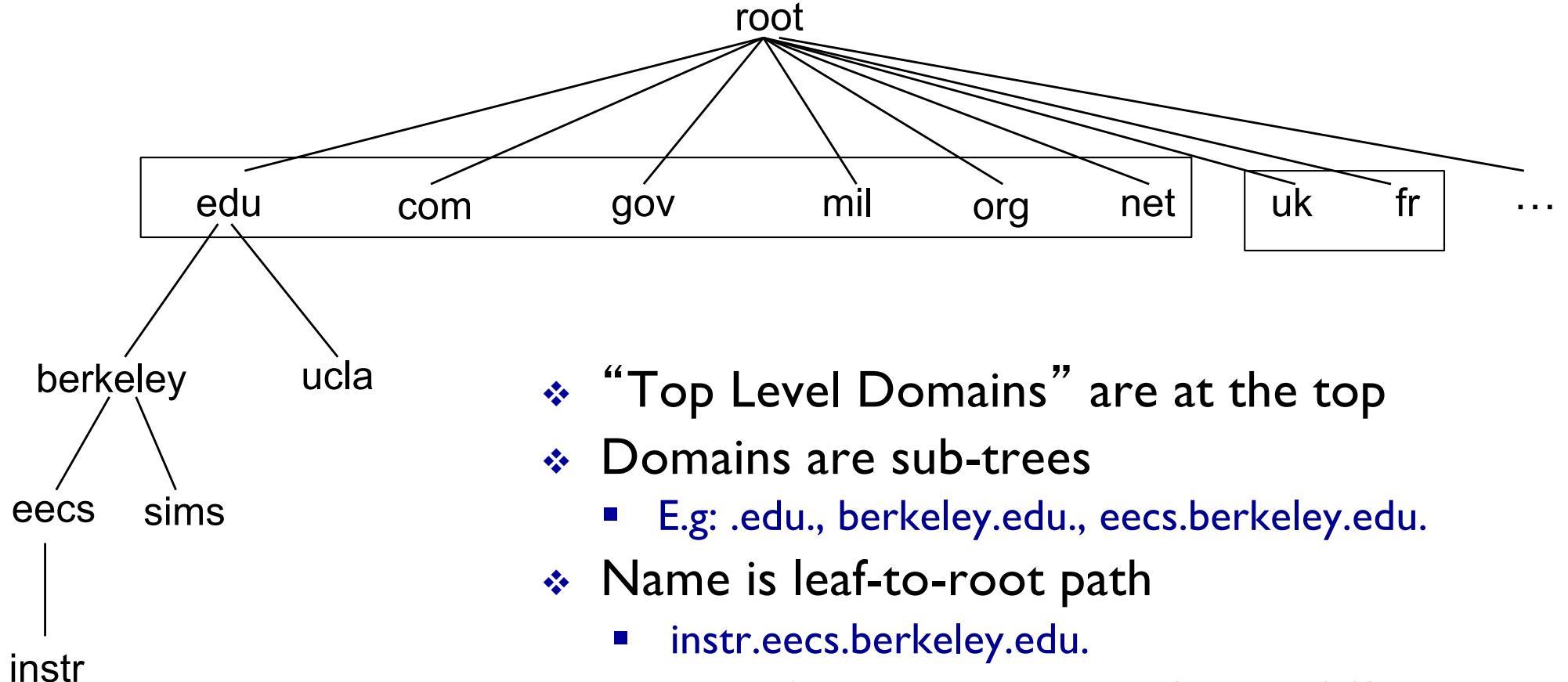
- ❖ No naming conflicts (uniqueness)
- ❖ Scalable
  - many names
  - (secondary) frequent updates
- ❖ Distributed, autonomous administration
  - Ability to update my own (domains') names
  - Don't have to track everybody's updates
- ❖ Highly available
- ❖ Lookups should be fast

# Key idea: Hierarchy

Three intertwined hierarchies

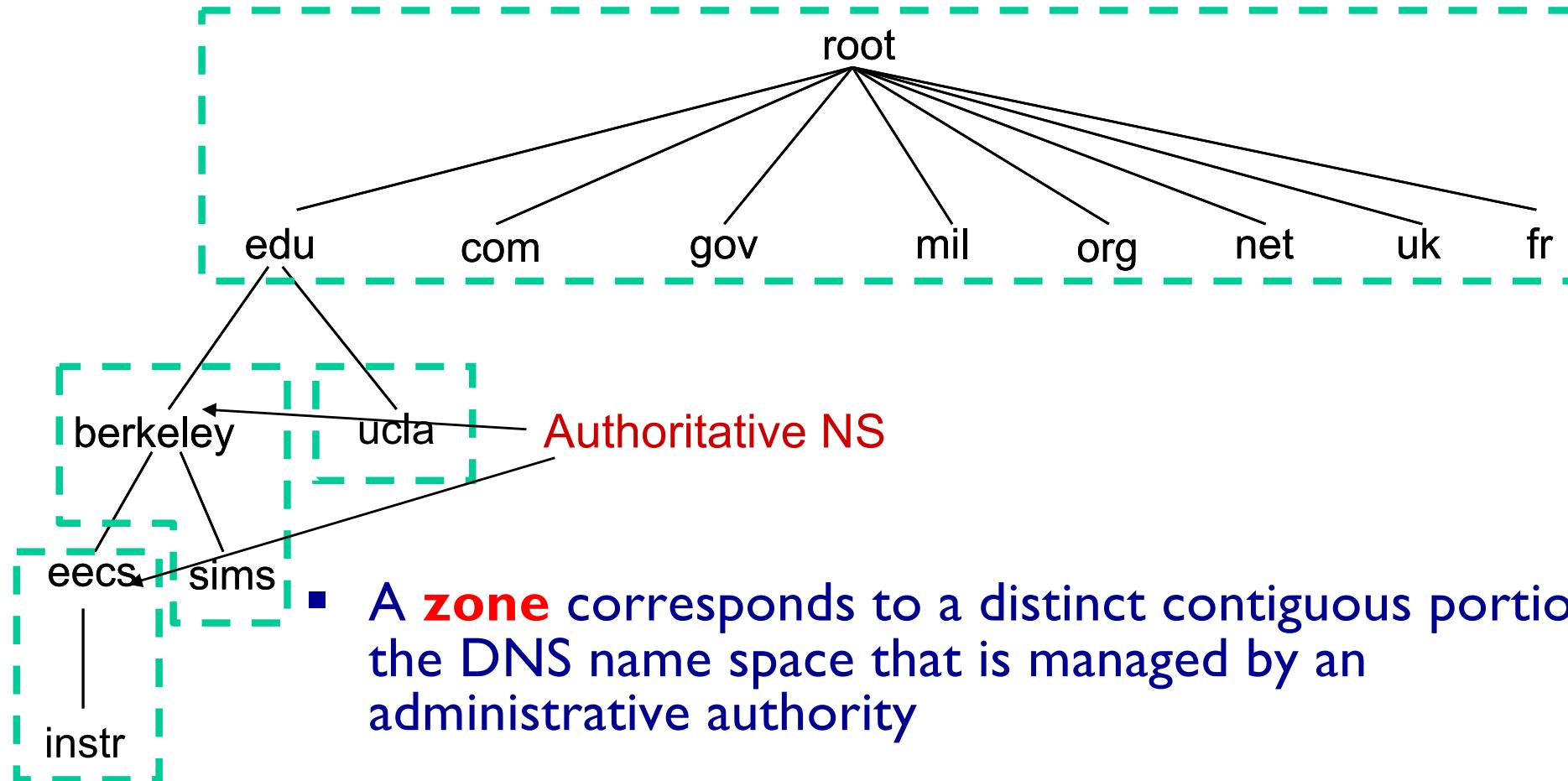
- Hierarchical namespace
  - As opposed to original flat namespace
- Hierarchically administered
  - As opposed to centralised
- (Distributed) hierarchy of servers
  - As opposed to centralised storage

# Hierarchical Namespace



- ❖ “Top Level Domains” are at the top
- ❖ Domains are sub-trees
  - E.g: .edu., berkeley.edu., eecs.berkeley.edu.
- ❖ Name is leaf-to-root path
  - instr.eecs.berkeley.edu.
- ❖ Depth of tree is arbitrary (limit 128)
- ❖ Name collisions trivially avoided
  - each domain is responsible

# Hierarchical Administration



- A **zone** corresponds to a distinct contiguous portion of the DNS name space that is managed by an administrative authority
- E.g., UCB controls names: \*.berkeley.edu and \*.sims.berkeley.edu
- ❖ E.g., EECS controls names: \*.eeecs.berkeley.edu

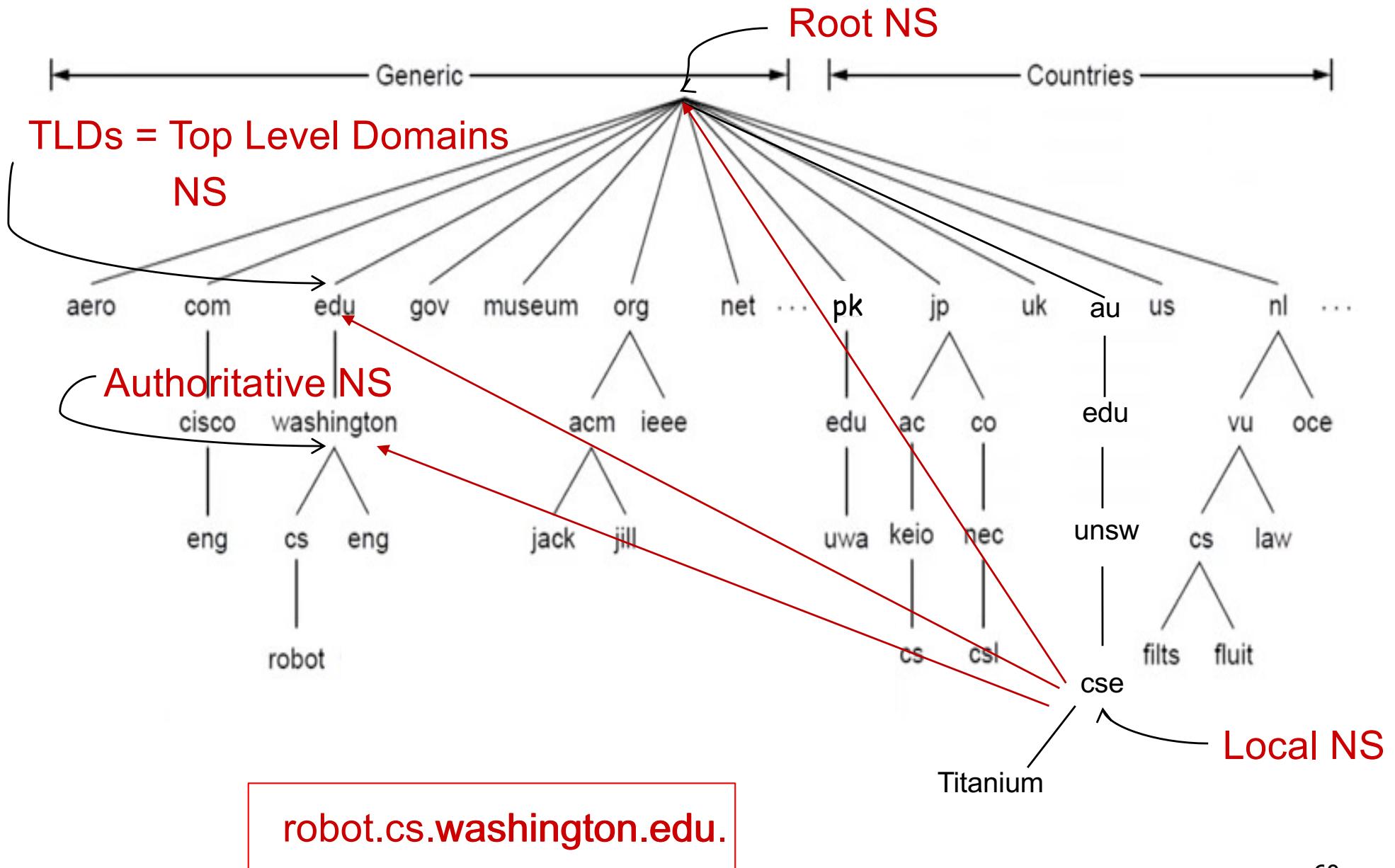
# Server Hierarchy

- ❖ Top of hierarchy: Root servers
  - Location hardwired into other servers
- ❖ Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc. (several new TLDs introduced recently)
  - Managed professionally
- ❖ Bottom Level: **Authoritative** DNS servers
  - Store the name-to-address mapping
  - Maintained by the corresponding administrative authority

# Server Hierarchy

- ❖ Each server stores a (small!) subset of the total DNS database
- ❖ An authoritative DNS server stores “resource records” for all DNS names in the domain that it has authority for
- ❖ Each server can discover the server(s) that are responsible for the other portions of the hierarchy
  - Every server knows the root server(s)
  - Root server(s) knows about all top-level domains

# DNS: a distributed, hierarchical database

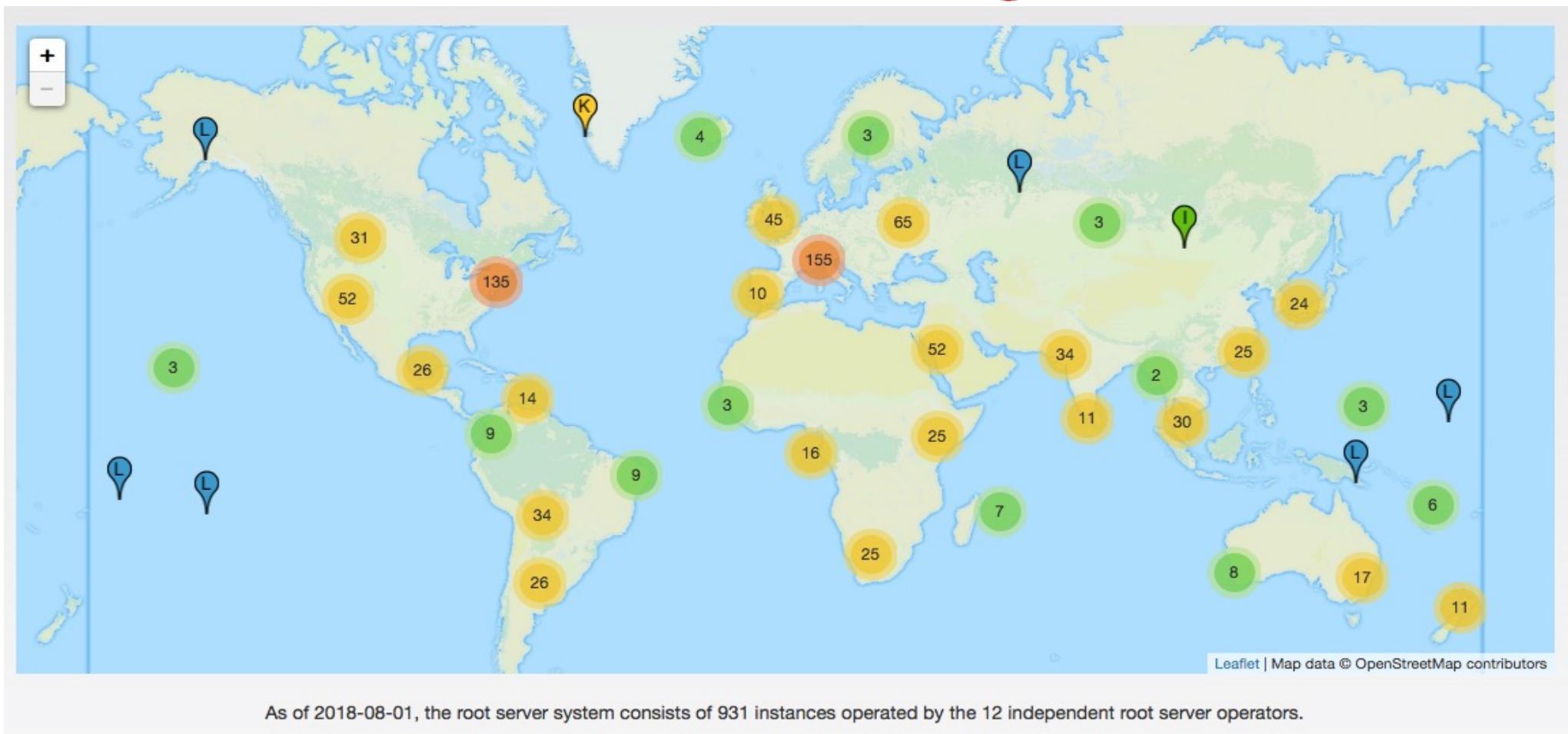


# DNS Root Servers

- 13 root servers (labeled A-M; see <http://www.root-servers.org/>)
- Replicated via any-casting (network will deliver DNS messages to the closest replica)



# DNS: root name servers



[www.root-servers.org](http://www.root-servers.org)



# TLD, authoritative servers

## *top-level domain (TLD) servers:*

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

## *authoritative DNS servers:*

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

# Local DNS name server

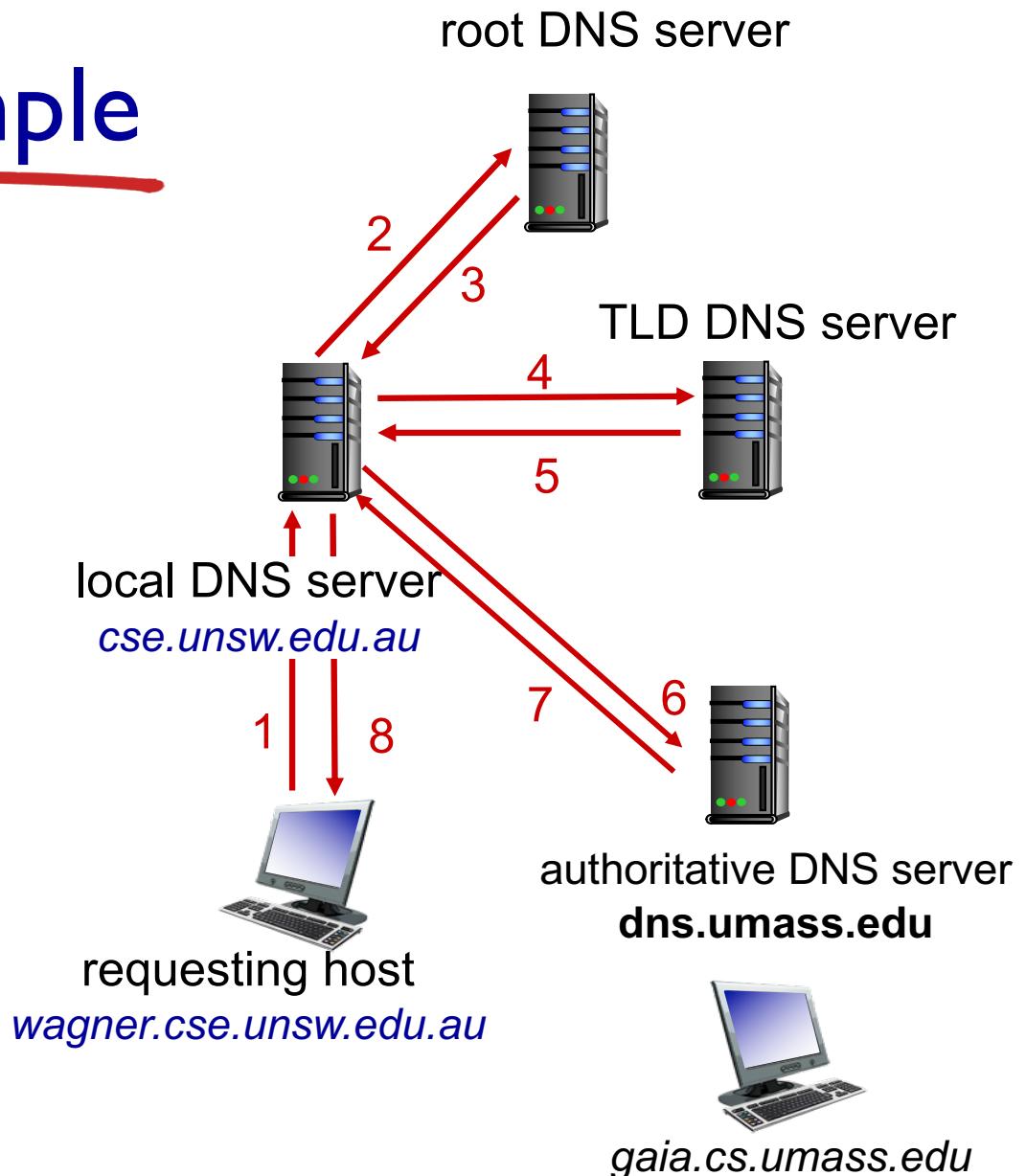
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
  - also called “default name server” or “DNS resolver”
- ❖ Hosts configured with local DNS server address (e.g.,  
`/etc/resolv.conf`) or learn server via a host configuration protocol (e.g., DHCP)
- ❖ Client application
  - Obtain DNS name (e.g., from URL)
  - Do `gethostbyname()` to trigger DNS request to its local DNS server
- ❖ when host makes DNS query, the query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

# DNS name resolution example

- ❖ host at `wagner.cse.unsw.edu.au` wants IP address for `gaia.cs.umass.edu`

## *iterated query:*

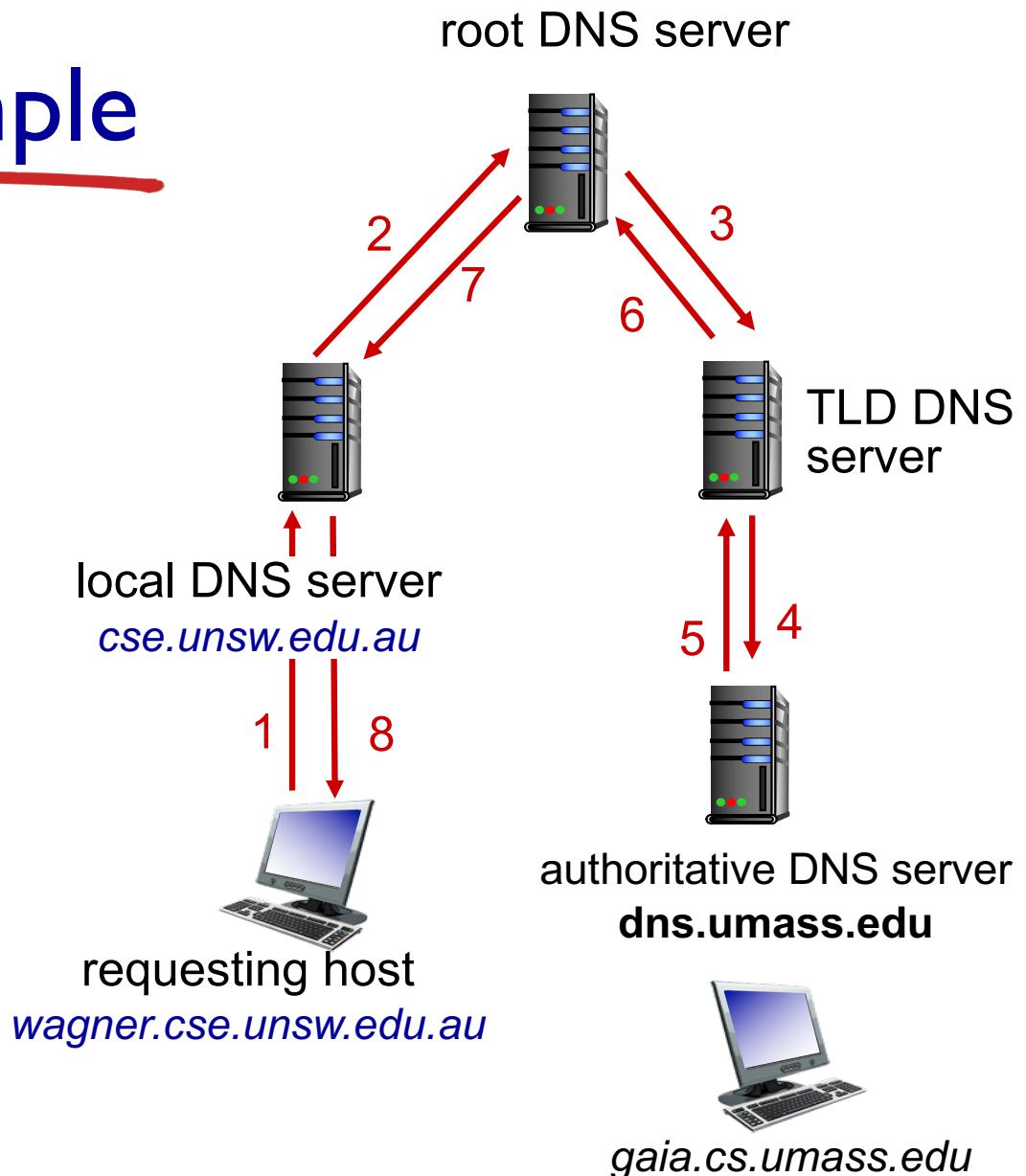
- ❖ contacted server replies with name of server to contact
- ❖ “I don’t know this name, but ask this server”



# DNS name resolution example

*recursive query:*

- ❖ puts burden of name resolution on contacted name server



# DNS: caching, updating records

- ❖ once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- ❖ Subsequent requests need not burden DNS
- ❖ cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- ❖ Negative caching (optional)
  - Remember things that don't work
  - E.g., misspellings like [www.cnn.comm](http://www.cnn.comm) and [www.cnnn.com](http://www.cnnn.com)
  - These can take a long time to fail for the first time
  - Good to remember that they don't work

# DNS records

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

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

## type=A

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

## type=NS

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

## type=CNAME

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

## type=MX

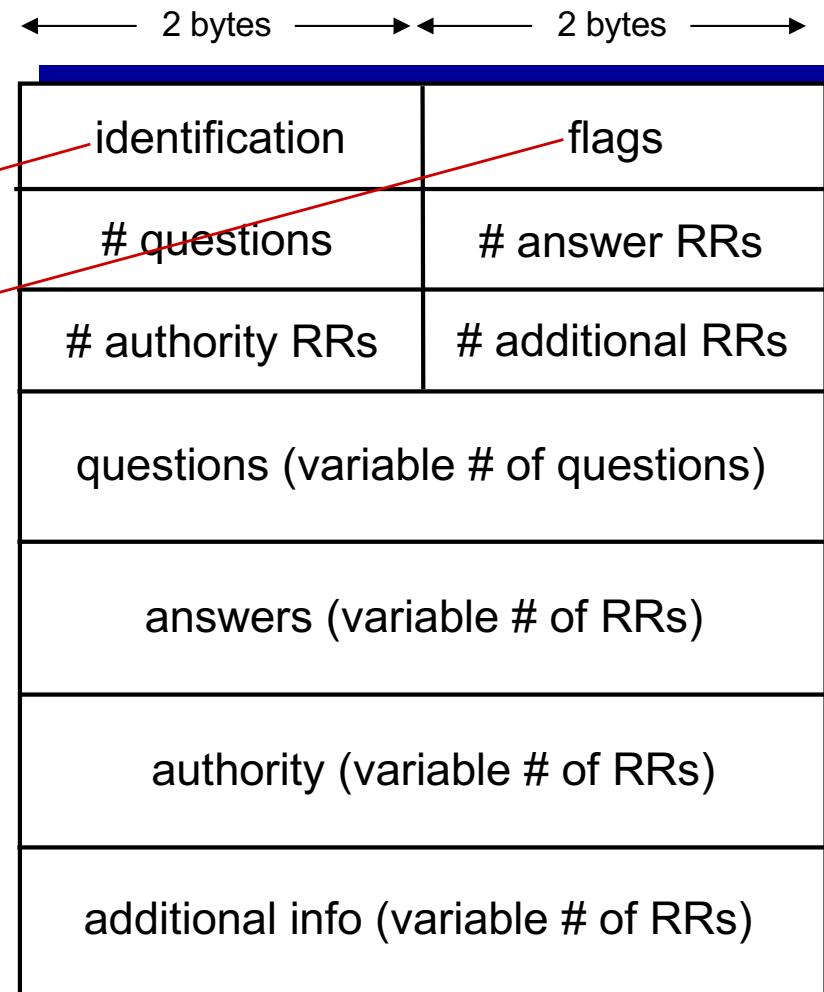
- **value** is name of mailserver  
associated with **name**

# DNS protocol, messages

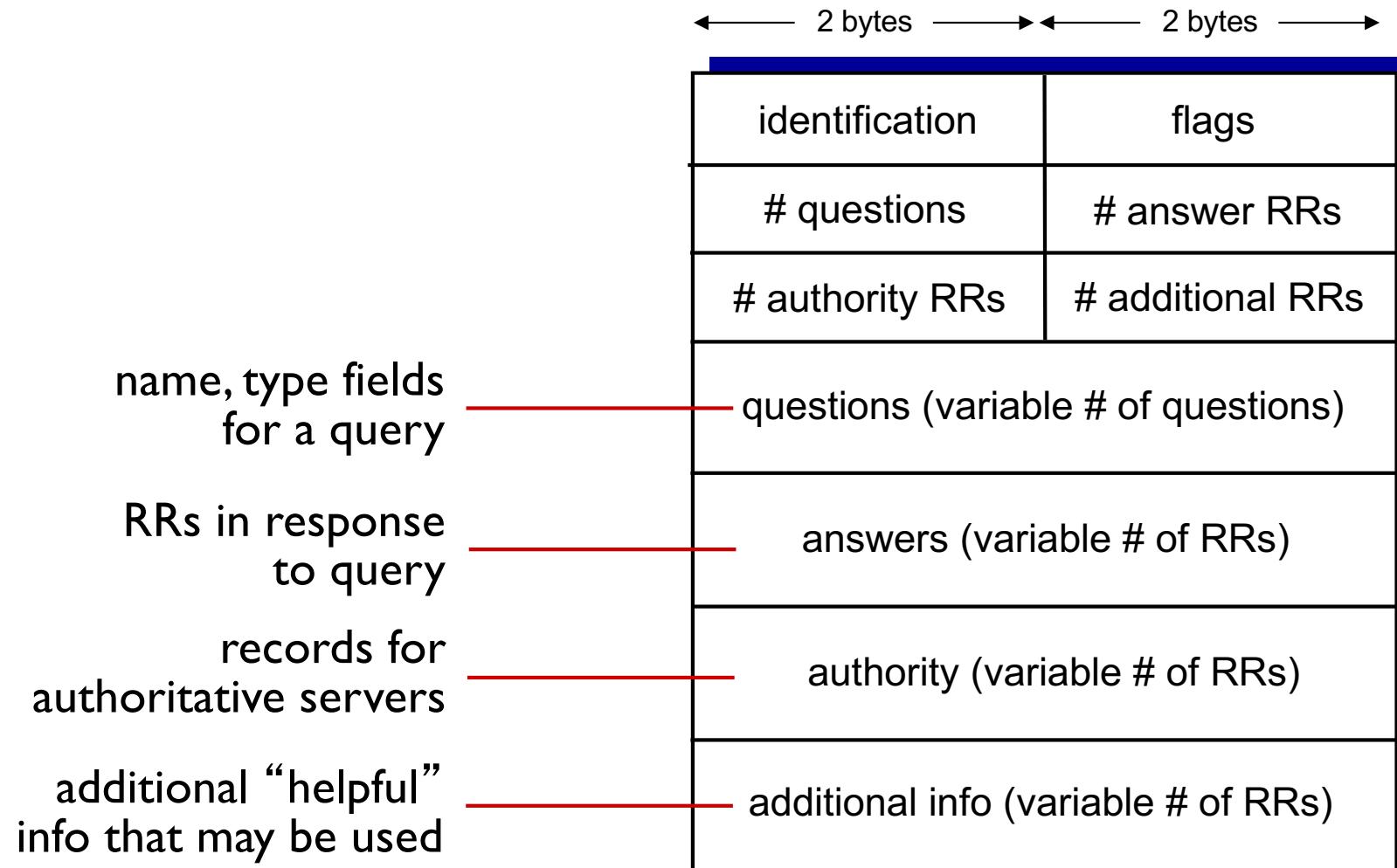
- ❖ *query* and *reply* messages, both with same *message format*

msg header

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



# DNS protocol, messages



# An Example

```
[salilk@wagner:~$ dig www.oxford.ac.uk

; <>> DiG 9.9.5-9+deb8u19-Debian <>> www.oxford.ac.uk
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23390
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 4, ADDITIONAL: 6

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.oxford.ac.uk.           IN      A

;; ANSWER SECTION:
www.oxford.ac.uk.        300     IN      A      151.101.194.133
www.oxford.ac.uk.        300     IN      A      151.101.2.133
www.oxford.ac.uk.        300     IN      A      151.101.66.133
www.oxford.ac.uk.        300     IN      A      151.101.130.133

;; AUTHORITY SECTION:
oxford.ac.uk.            86400   IN      NS     dns2.ox.ac.uk.
oxford.ac.uk.            86400   IN      NS     dns0.ox.ac.uk.
oxford.ac.uk.            86400   IN      NS     dns1.ox.ac.uk.
oxford.ac.uk.            86400   IN      NS     ns2.ja.net.

;; ADDITIONAL SECTION:
ns2.ja.net.              81448   IN      A      193.63.105.17
ns2.ja.net.              17413   IN      AAAA    2001:630:0:45::11
dns0.ox.ac.uk.          42756   IN      A      129.67.1.190
dns1.ox.ac.uk.          908     IN      A      129.67.1.191
dns2.ox.ac.uk.          908     IN      A      163.1.2.190

;; Query time: 544 msec
;; SERVER: 129.94.242.2#53(129.94.242.2)
;; WHEN: Mon Sep 28 10:55:27 AEST 2020
;; MSG SIZE  rcvd: 285
```

Try this out  
yourself. Part of  
Lab 3

# Inserting records into DNS

- ❖ example: new startup “Network Utopia”
- ❖ register name networkutopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into .com TLD server:  
(`networkutopia.com`, `dns1.networkutopia.com`, NS)  
(`dns1.networkutopia.com`, `212.212.212.1`, A)
- ❖ create authoritative server type A record for `www.networkutopia.com`; type MX record for `networkutopia.com`
- ❖ Q: Where do you insert these type A and type MX records?

A: ??

# Updating DNS records

- ❖ Remember that old records may be cached in other DNS servers (for up to TTL)
- ❖ General guidelines
  - Record the current TTL value of the record
  - Lower the TTL of the record to a low value (e.g., 30 seconds)
  - Wait the length of the previous TTL
  - Update the record
  - Wait for some time (e.g. 1 hour)
  - Change the TTL back to your previous time

# Reliability

- ❖ DNS servers are **replicated** (primary/secondary)
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- ❖ Usually, UDP used for queries
  - Need reliability: must implement this on top of UDP
  - Spec supports TCP too, but not always implemented
- ❖ DNS uses port 53
- ❖ Try alternate servers on timeout
  - **Exponential backoff** when retrying same server
- ❖ Same identifier for all queries
  - Don't care which server responds

# DNS provides indirection

- ❖ Addresses can **change** underneath
  - Move www.cnn.com to 4.125.91.21
  - Humans/Apps should be unaffected
- ❖ Name could map to **multiple** IP addresses
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers
- ❖ **Multiple names** for the same address
  - E.g., many services (mail, www, ftp) on same machine
  - E.g., aliases like www.cnn.com and cnn.com
- ❖ But this flexibility applies only within domain!

# CDN example

```

bash-3.2$ dig www.mit.edu

; <>> DiG 9.10.6 <>> www.mit.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17913
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 8, ADDITIONAL: 8

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.mit.edu.           IN      A

;; ANSWER SECTION:
www.mit.edu.          924    IN      CNAME   www.mit.edu.edgekey.net.
www.mit.edu.edgekey.net. 54    IN      CNAME   e9566.dscb.akamaiedge.net.
e9566.dscb.akamaiedge.net. 14    IN      A       23.77.154.132

;; AUTHORITY SECTION:
dscb.akamaiedge.net. 623    IN      NS      n0dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n2dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n7dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n6dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n1dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n3dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n5dscb.akamaiedge.net.
dscb.akamaiedge.net. 623    IN      NS      n4dscb.akamaiedge.net.

;; ADDITIONAL SECTION:
n0dscb.akamaiedge.net. 1241   IN      A       88.221.81.192
n0dscb.akamaiedge.net. 1124   IN      AAAA   2600:1480:e800::c0
n1dscb.akamaiedge.net. 842    IN      A       23.32.5.76
n2dscb.akamaiedge.net. 749    IN      A       23.32.5.84
n4dscb.akamaiedge.net. 1399   IN      A       23.32.5.177
n6dscb.akamaiedge.net. 702    IN      A       23.32.5.98
n7dscb.akamaiedge.net. 1208   IN      A       23.206.243.54

;; Query time: 46 msec
;; SERVER: 129.94.172.11#53(129.94.172.11)
;; WHEN: Mon Sep 28 13:15:28 AEST 2020
;; MSG SIZE  rcvd: 421

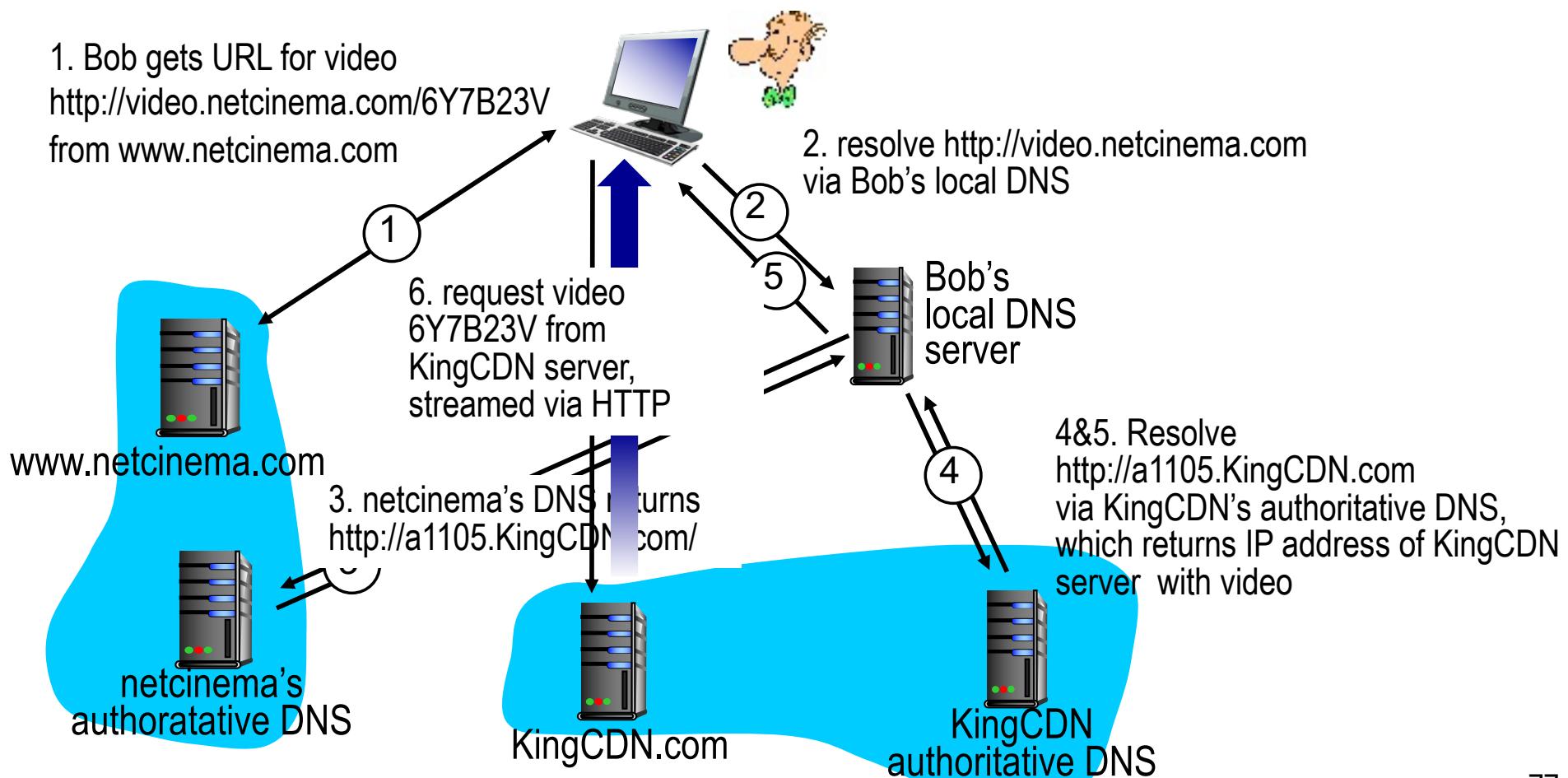
```

Many well-known sites are hosted by CDNs. A simple way to check using dig is shown here.

# CDN content access: a closer look

Bob (client) requests video <http://video.netcinema.com/6Y7B23V>

- video stored in CDN at managed by KingCDN.com



# WWW vs non-WWW domains

- ❖ E.g., www.metalhead.com or metalhead.com
- ❖ Non-www referred to as apex or naked domains (metalhead.com)
- ❖ Technically either can serve as primary (for search engines) and the other is redirected to primary (HTTP 301)
- ❖ There are 2 main advantages of using www
  - DNS requires apex domains to always point to type A and that CNAME record cannot coexist with other RR types
  - With www domains, offloading to a CDN is easy:
    - www.metalhead.com CNAME somecdn.com
    - metalhead.com A 156.23.34.252
    - Note: Some CDN providers have workarounds for the above
  - Cookies of the apex domain are automatically passed down to sub-domains (metalhead.com to static.metalhead.com and mail.metalhead.com)
    - Unnecessary cookies hurt performance
    - Also, a security issue (out of scope of our discussion)

More reading at: <https://www.bjornjohansen.com/www-or-not>

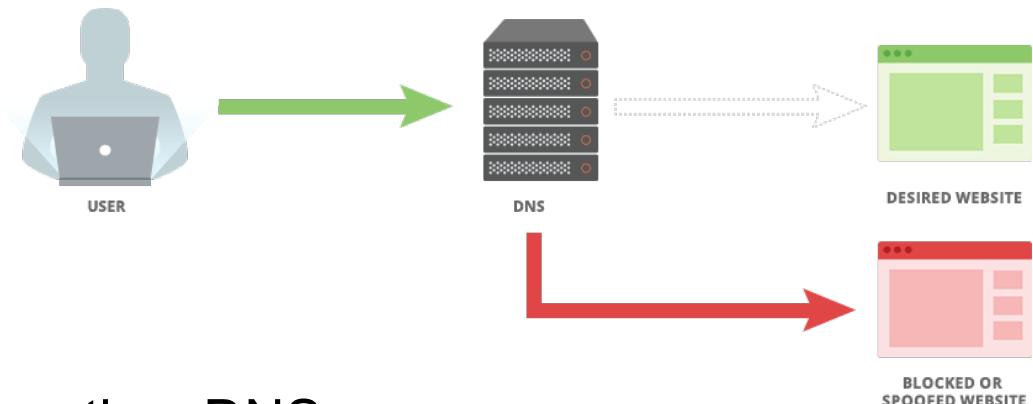
# Reverse DNS

- ❖ IP address -> domain name
- ❖ Special PTR record type to store reverse DNS entries
- ❖ Where is reverse DNS used?
  - Troubleshooting tools such as traceroute and ping
  - “Received” trace header field in SMTP e-mail
  - SMTP servers for validating IP addresses of originating servers
  - Internet forums tracking users
  - System logging or monitoring tools
  - Used in load balancing servers/content distribution to determine location of requester



# Do you trust your DNS server?

- ❖ Censorship



[https://wikileaks.org/wiki/Alternative\\_DNS](https://wikileaks.org/wiki/Alternative_DNS)

- ❖ Logging

- IP address, websites visited, geolocation data and more
- E.g., Google DNS:

<https://developers.google.com/speed/public-dns/privacy>

# Attacking DNS



## DDoS attacks

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

## Redirect attacks

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

## Exploit DNS for DDoS

- ❖ Send queries with spoofed source address: target IP

Want to dig deeper?

<http://www.networkworld.com/article/2886283/security0/top-10-dns-attacks-likely-to-infiltrate-your-network.html>



IT disaster recovery, cloud computing and information security news

## DNS attacks on the rise finds 2021 Global DNS Threat Report

Published: Wednesday, 09 June 2021 07:12

Print

EfficientIP has announced the results of its 2021 Global DNS Threat Report. The annual research, which was conducted in collaboration with IDC, sheds light on the frequency of the different types of DNS attack and the associated costs for the last year throughout the COVID-19 pandemic.

Globally, 87 percent of organizations surveyed experienced DNS attacks, with the average cost of each attack around £693,507 (€779,008). The Report shows that organizations across all industries suffered an average of 7.6 attacks this past year. These figures illustrate the pivotal role of DNS for network security, both as a threat vector and security objective.

The 2021 DNS Threat Report found that, throughout the past year during the pandemic, attackers have increasingly targeted the cloud, profiting from the reliance on off-premise working and cloud infrastructures. Around a quarter of companies have suffered a DNS attack abusing cloud misconfiguration, with almost half of companies (47 percent) suffering cloud service downtime as a result of DNS attacks.

The Threat Report, now in its seventh year, also found a sharp rise in data theft via DNS, with 26 percent of organizations reporting having sensitive customer information stolen compared to 16 percent in 2020's Threat Report.

Evidence shows attackers are targeting more organizations and diversifying their toolkits. Threat actors relied on domain hijacking, where the user is connected not to the desired service but to a fake one, more than twice as often as last year. This year phishing also continued to grow in popularity (49 percent of companies experienced phishing attempts), as did malware-based attacks (38 percent), and traditional DDoS attacks (29 percent).

Although the cost and variety of attacks remains high, there is a growing awareness of DNS security and how to combat these attacks.

76 percent of respondents in the 2021 Threat Report deemed DNS security a critical component of their network architecture. Additionally, the report found zero trust is evolving as a tool to protect networks in the remote era. 75 percent of companies are planning, implementing, or running zero trust initiatives and 43 percent of companies believe DNS domain deny and allow lists are highly valuable for improving control over access to apps.

<https://www.continuitycentral.com/index.php/news/technology/6340-dns-attacks-on-the-rise-finds-2021-global-dns-threat-report>

# DNS Cache Poisoning



- ❖ Suppose you are a bad guy  and you control the name server for drevil.com. Your name server receives a request to resolve www.drevil.com. and it responds as follows:

;; QUESTION SECTION:

;www.drevil.com. IN A

;; ANSWER SECTION:

www.drevil.com 300 IN A 129.45.212.42

;; AUTHORITY SECTION:

drevil.com 86400 IN NS dns1.drevil.com.

drevil.com 86400 IN NS google.com

;; ADDITIONAL SECTION:

google.com 600 IN A 129.45.212.222

A drevil.com machine, **not** google.com

- ❖ Solution: Do not allow DNS servers to cache IP address mappings unless they are from authoritative name servers

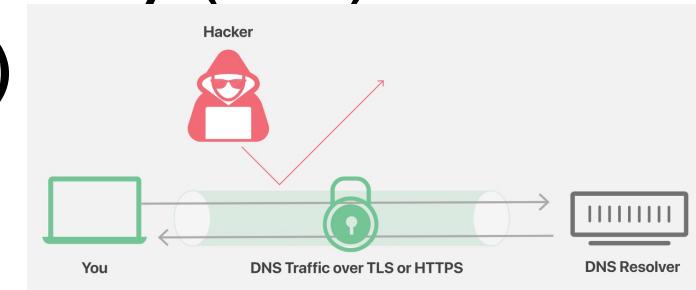
# DNSSEC

---

- ❖ Extension to improve DNS security
- ❖ Allows DNS clients to cryptographically authenticate DNS data and data integrity
- ❖ Does not guarantee availability or confidentiality
- ❖ Further details: <https://www.dnssec.net>
- ❖ Stats: <https://stats.labs.apnic.net/dnssec>

# DoH (RFC 8484) and DoT (RFC 7858)

- ❖ DoT: DNS over Transport Layer Security (TLS)
- ❖ DoH: DNS over HTTPS (or HTTP2)
- ❖ Increase user privacy and security
- ❖ DoT: port 853, DoH: port 443
- ❖ DoH traffic masked with other HTTPS traffic
- ❖ Cloudflare, Google, etc. have publicly accessible DoT resolvers and OS support is also available
- ❖ Chrome and Mozilla support DoH, OS support coming soon (or already there)
- ❖ DoT: <https://developers.google.com/speed/public-dns/docs/dns-over-tls>
- ❖ DoH: <https://developers.cloudflare.com/1.1.1.1/dns-over-https>



# Quiz: DNS



- ❖ If a local DNS server has no clue about where to find the address for a hostname then the \_\_\_\_\_
  - a) Server starts crying
  - b) Server asks the root DNS server
  - c) Server asks its neighbouring DNS server
  - d) Request is not processed

**Answer: B**

# Quiz: DNS



- ❖ Which of the following are respectively maintained by the client-side ISP and the domain name owner?
  - a) Root DNS server, Top-level domain DNS server
  - b) Root DNS server, Local DNS server
  - c) Local DNS server, Authoritative DNS server
  - d) Top-level domain DNS server, Authoritative DNS server
  - e) Authoritative DNS server, Top-level domain DNS server

Answer: C

# Quiz: DNS



- ❖ Suppose you open your email program and send an email to [salil@unsw.edu.au](mailto:salil@unsw.edu.au), your email program will trigger which type of DNS query?
  - a) A
  - b) NS
  - c) CNAME
  - d) MX
  - e) All of the above

**Answer: D**

# Quiz: DNS



❖ You open your browser and type [www.zeetings.com](http://www.zeetings.com). The minimum number of DNS requests sent by your local DNS server to obtain the corresponding IP address is:

- A. 0
- B. 1
- C. 2
- D. 3
- E. 42

**Answer: A**



## Quiz: CDN

- ❖ The role of the CDN provider's authoritative DNS name server in a content distribution network, simply described, is:
  - a) to provide an alias address for each browser access to the “origin server” of a CDN website
  - b) to map the query for each CDN object to the CDN server closest to the requestor (browser)
  - c) to provide a mechanism for CDN “origin servers” to provide paths for clients (browsers)
  - d) none of the above, CDN networks do not use DNS

Answer: B

# Summary

*our study of network apps now complete!*

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

# Computer Networks and Applications

COMP 3331/COMP 9331

Week 4

## Transport Layer Part 1

Reading Guide:  
Chapter 3, Sections 3.1 – 3.4

# Transport Layer

## our goals:

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

# Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

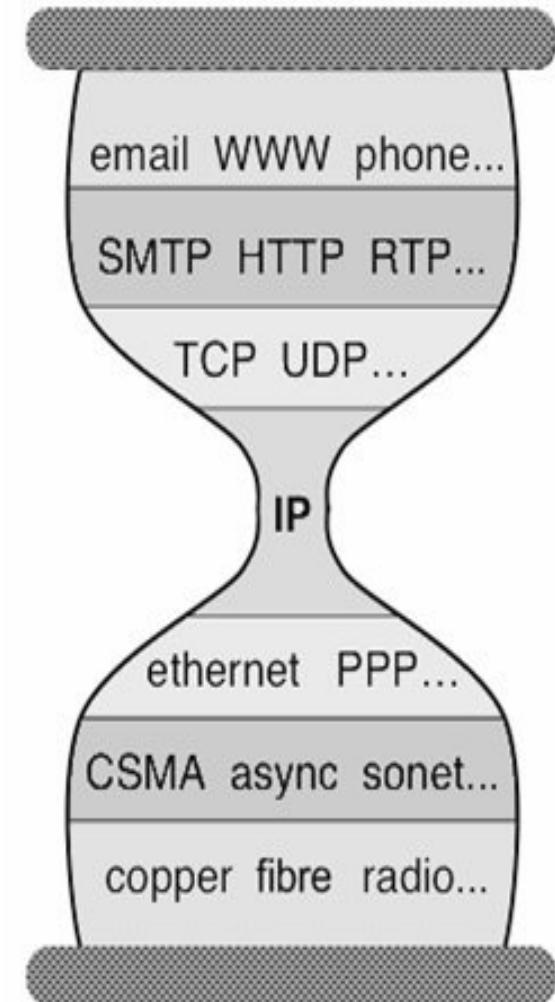
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# Transport layer

- ❖ Moving “down” a layer
- ❖ Current perspective:
  - Application layer is the boss....
  - Transport layer usually executing within the OS Kernel
  - The network layer is ours to command !!

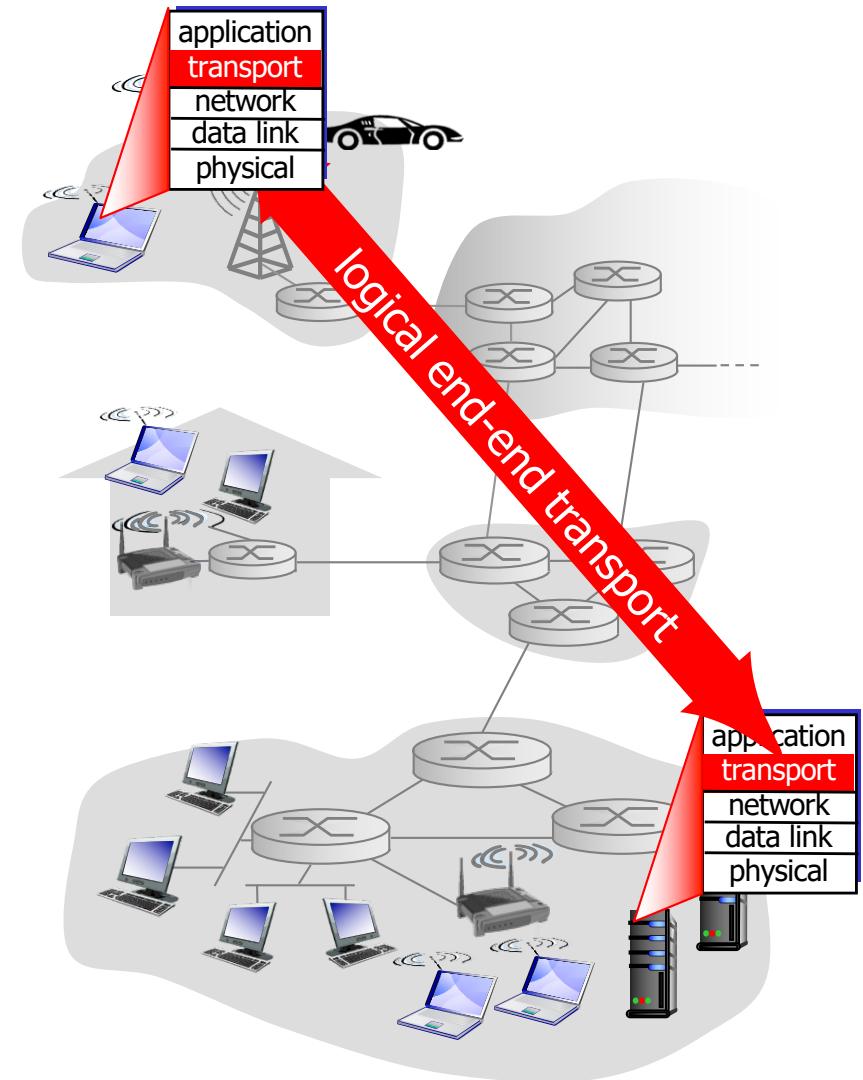


# Network layer (some context)

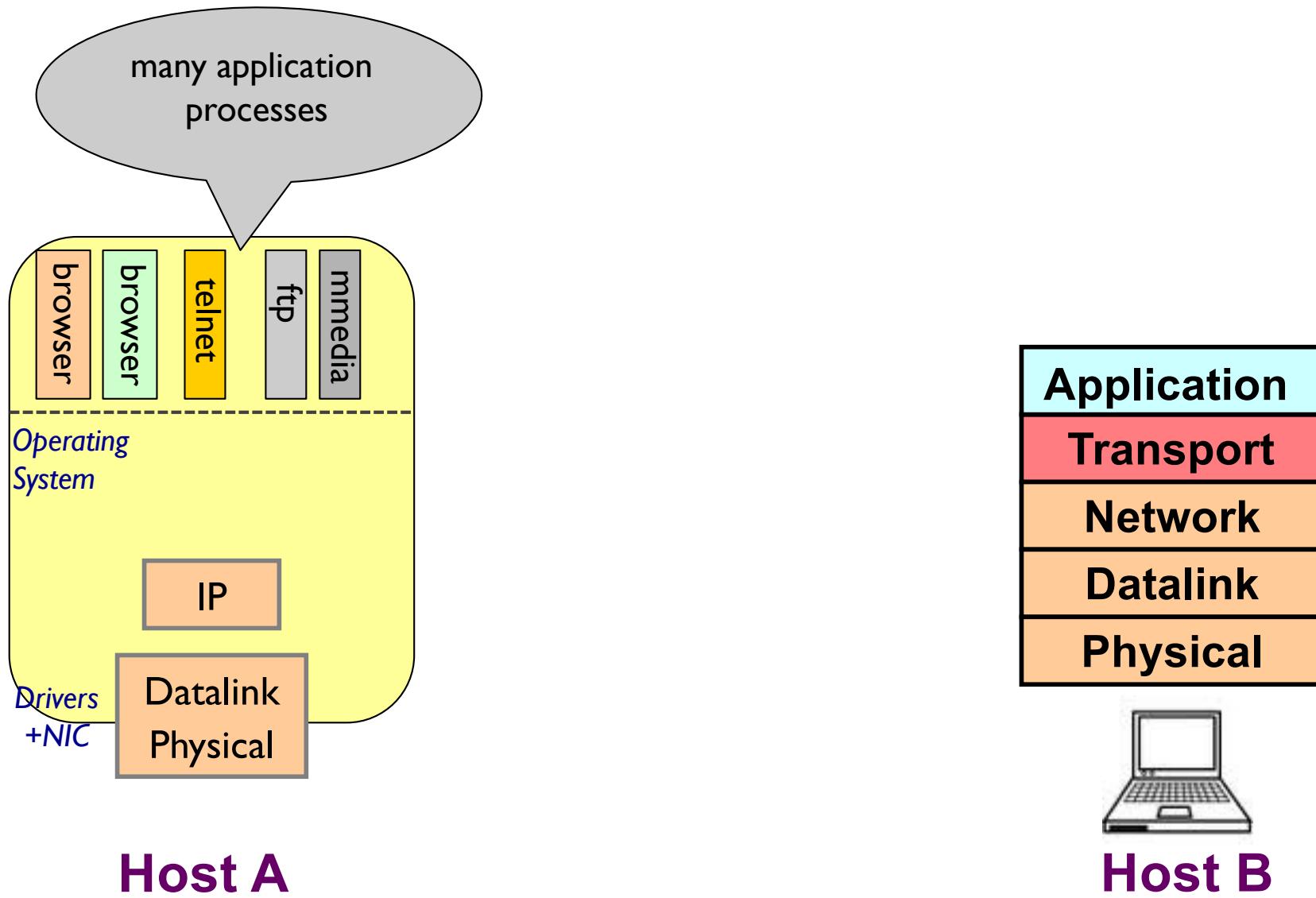
- ❖ What it does: finds paths through network
  - Routing from one end host to another
- ❖ What it doesn't:
  - Reliable transfer: “best effort delivery”
  - Guarantee paths
  - Arbitrate transfer rates
- ❖ For now, think of the network layer as giving us an “API” with one function:  
*sendtohost(data, host)*
  - Promise: the data will go to that (usually!!)

# Transport services and protocols

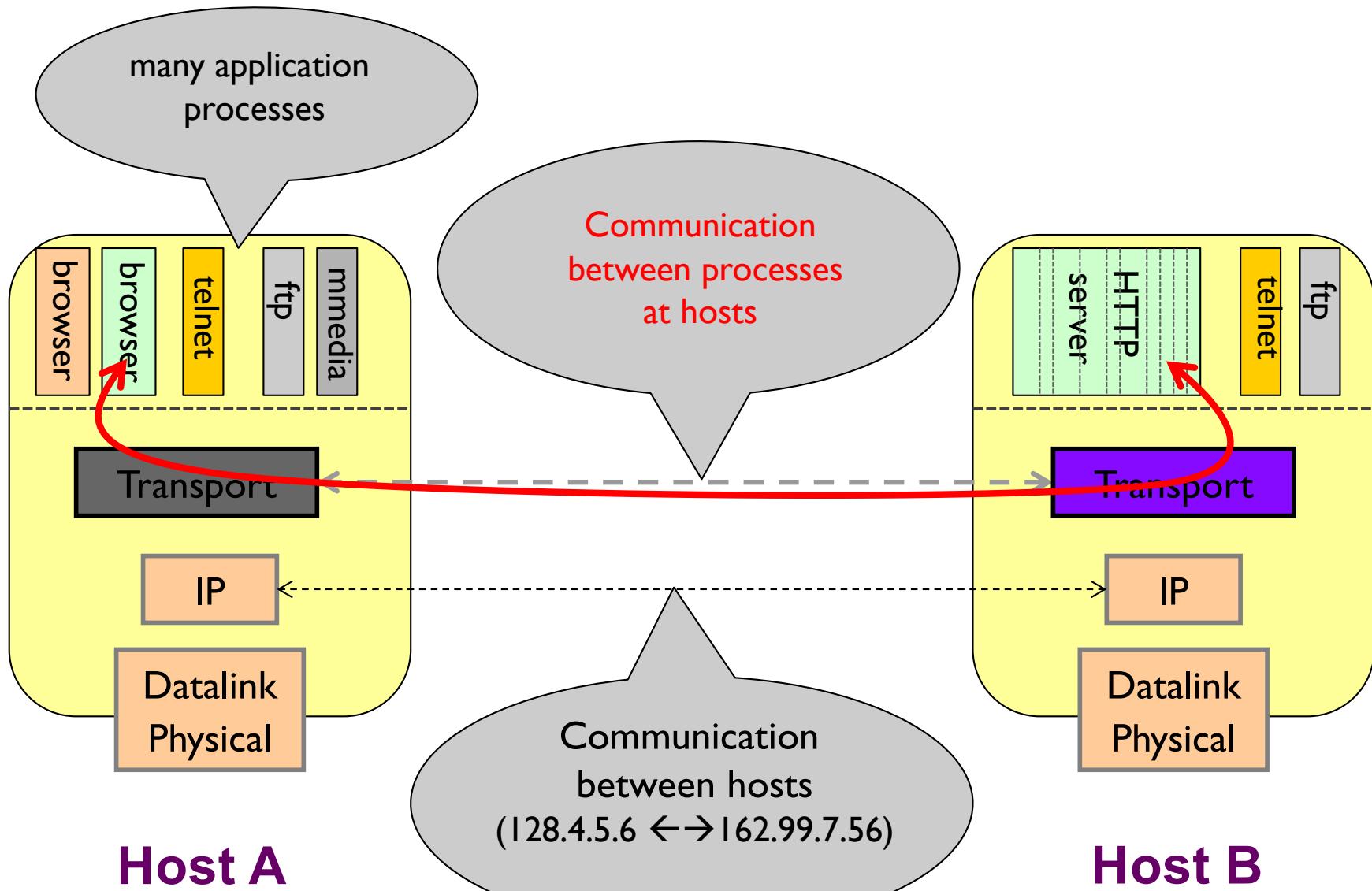
- ❖ provide *logical communication* between app processes running on different hosts
- ❖ transport protocols run in end systems
  - sender side: breaks app messages into *segments*, passes to network layer
  - receiver side: reassembles segments into messages, passes to app layer
  - Exports services to application that network layer does not provide



# Why a transport layer?



# Why a transport layer?



# Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

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

3.6 principles of congestion control

3.7 TCP congestion control

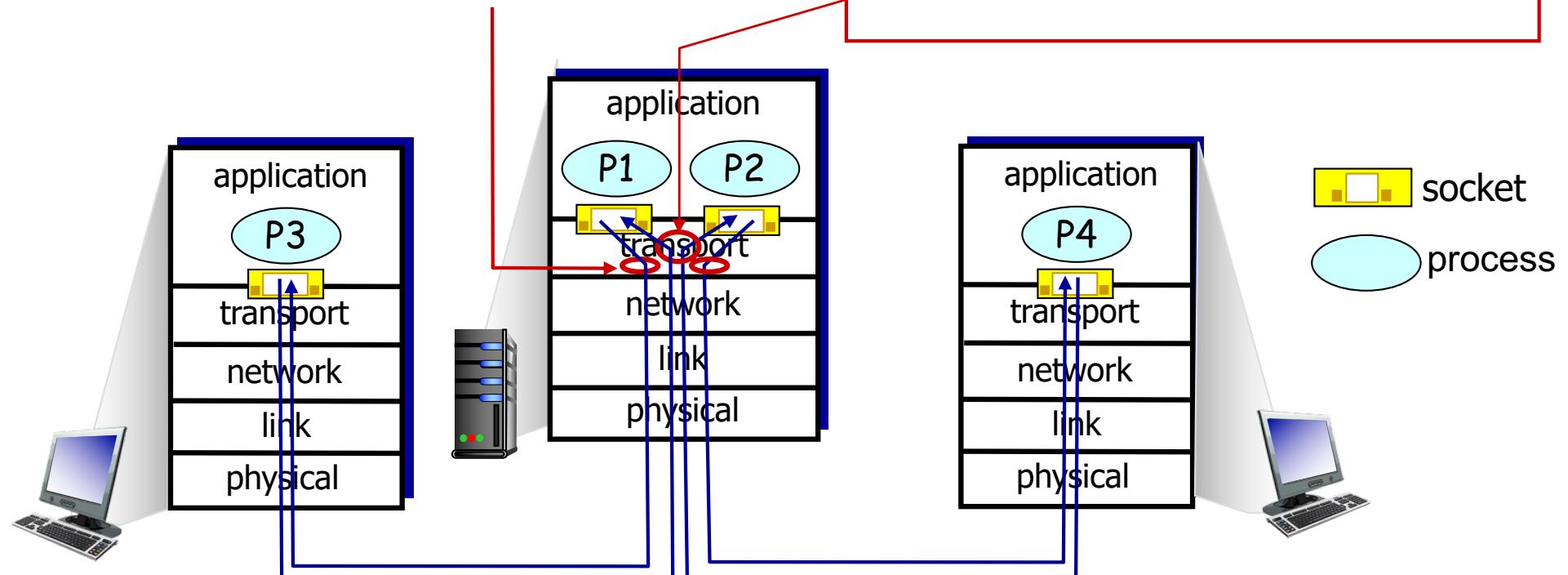
# Multiplexing/demultiplexing

*multiplexing at sender:*

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

*demultiplexing at receiver:*

use header info to deliver received segments to correct socket



**Note:** The network is a shared resource. It does not care about your applications, sockets, etc.

# Connectionless demultiplexing

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

```
DatagramSocket mySocket1  
= new DatagramSocket(12534);
```

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

- ❖ when host receives UDP segment:

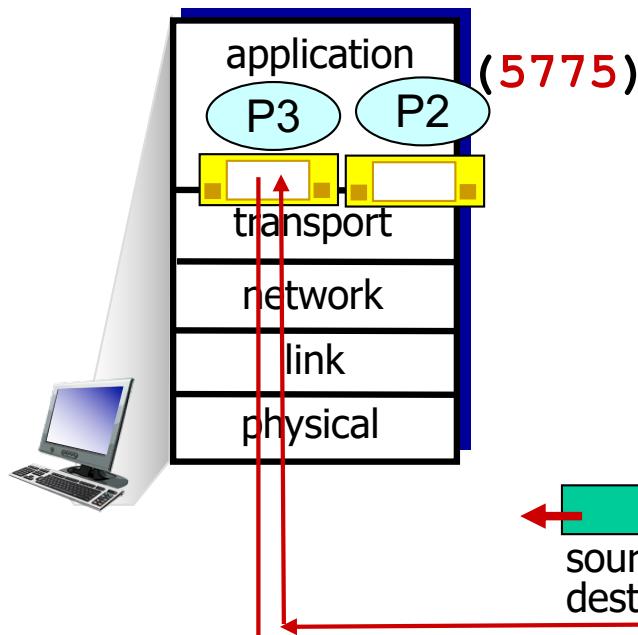
- checks destination port # in segment
- directs UDP segment to socket with that port #



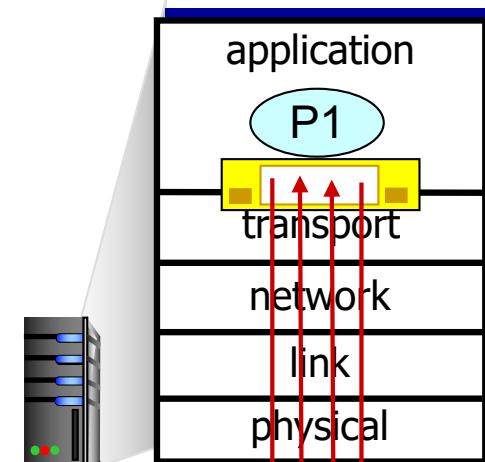
IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

# Connectionless demux: example

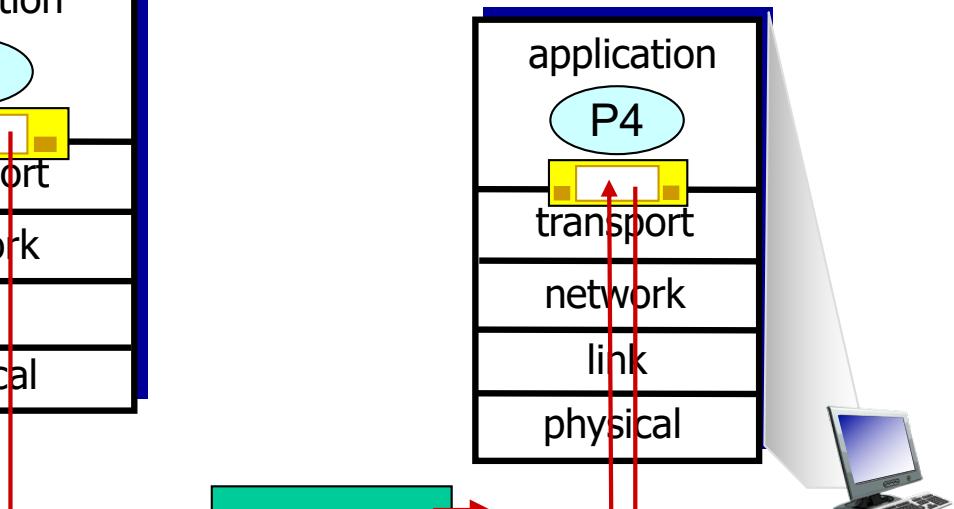
```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```



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



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



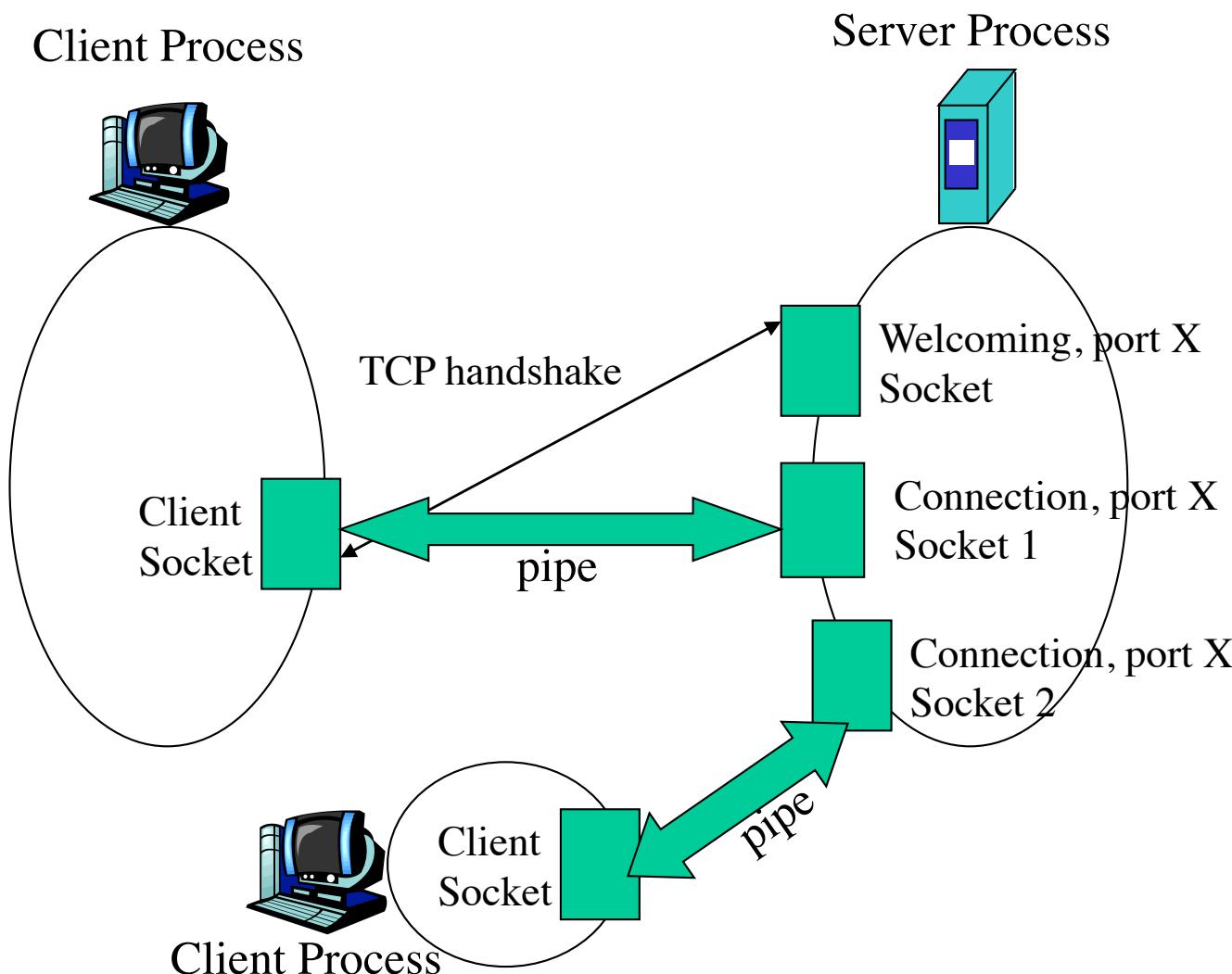
source port: 9157  
dest port: 6428

source port: ?  
dest port: ?

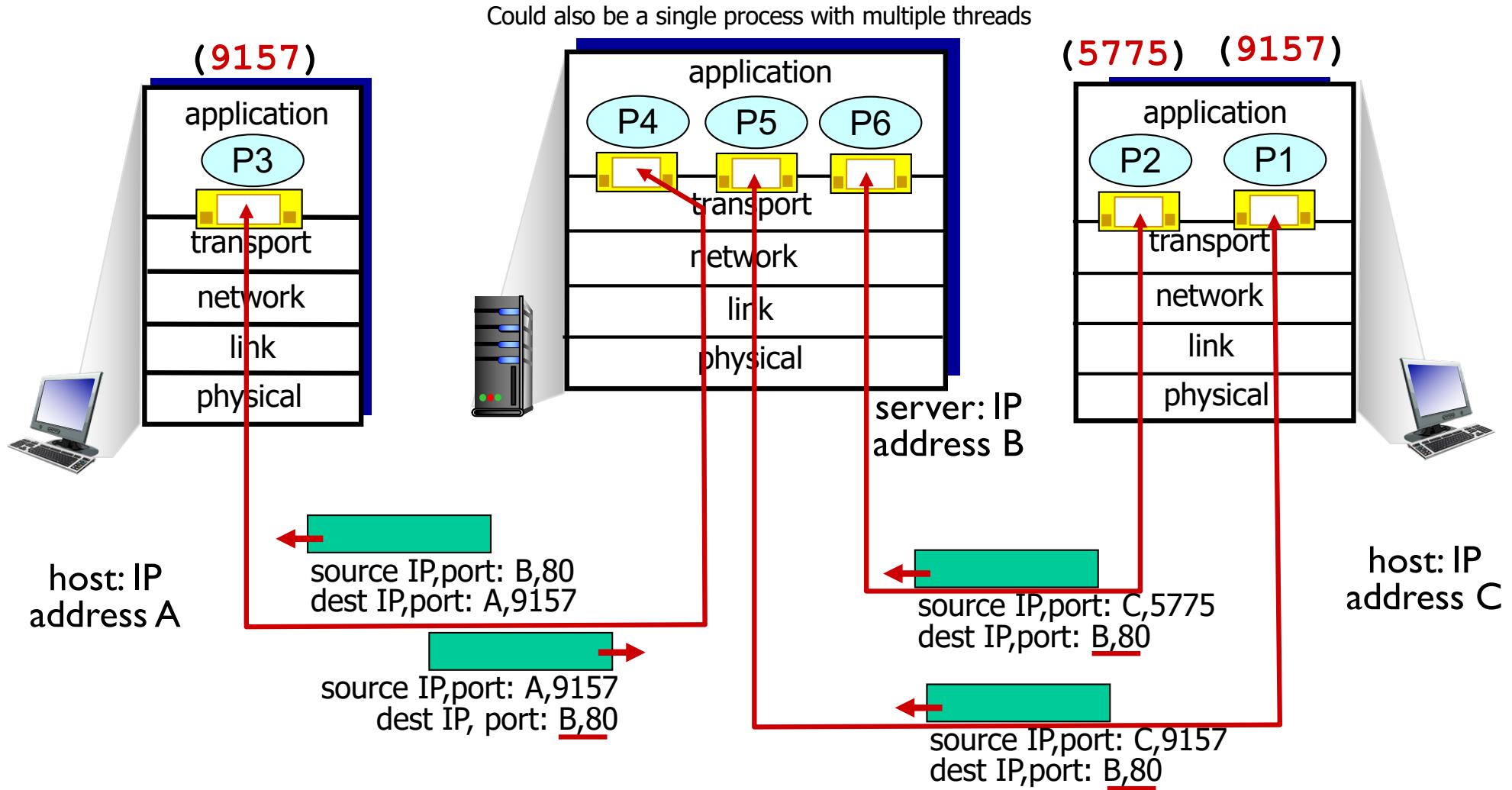
# Connection-oriented demux

- ❖ 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

# Revisiting TCP Sockets



# Connection-oriented demux: example



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

# May I scan your ports?

<http://netsecurity.about.com/cs/hackertools/a/aa121303.htm>

- ❖ Servers wait at open ports for client requests
- ❖ Hackers often perform *port scans* to determine open, closed and unreachable ports on candidate victims
- ❖ Several ports are well-known
  - <1024 are reserved for well-known apps
  - Other apps also use known ports
    - MS SQL server uses port 1434 (udp)
    - Sun Network File System (NFS) 2049 (tcp/udp)
- ❖ Hackers can exploit known flaws with these known apps
  - Example: Slammer worm exploited buffer overflow flaw in the SQL server
- ❖ How do you scan ports?
  - Nmap, Superscan, etc

<http://www.auditmypc.com/>

<https://www.grc.com/shieldsup>

# Quiz: UDP Sockets



- ❖ Suppose we use UDP instead of TCP for communicating with a web server where all requests and responses fit in a single UDP segment. Suppose 100 clients are simultaneously communicating with this web server. How many sockets are respectively active at the server and each client?

- a) 1, 1
- b) 2, 1
- c) 200, 2
- d) 100, 1
- e) 101, 1

**ANSWER: a)**

# Quiz: TCP Sockets



- ❖ Suppose 100 clients are simultaneously communicating with a traditional HTTP/TCP web server. How many sockets are active respectively at the server and each client?
    - a) 1, 1
    - b) 2, 1
    - c) 200, 2
    - d) 100, 1
    - e) 101, 1
- ANSWER: d) or e) depending on whether a welcoming socket is counted as a socket**

# Quiz: TCP Sockets



- ❖ Suppose 100 clients are simultaneously communicating with a traditional HTTP/TCP web server. Do all the TCP sockets at the server have the same server-side port number?
  - a) Yes
  - b) No

**ANSWER: a)**

# Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

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

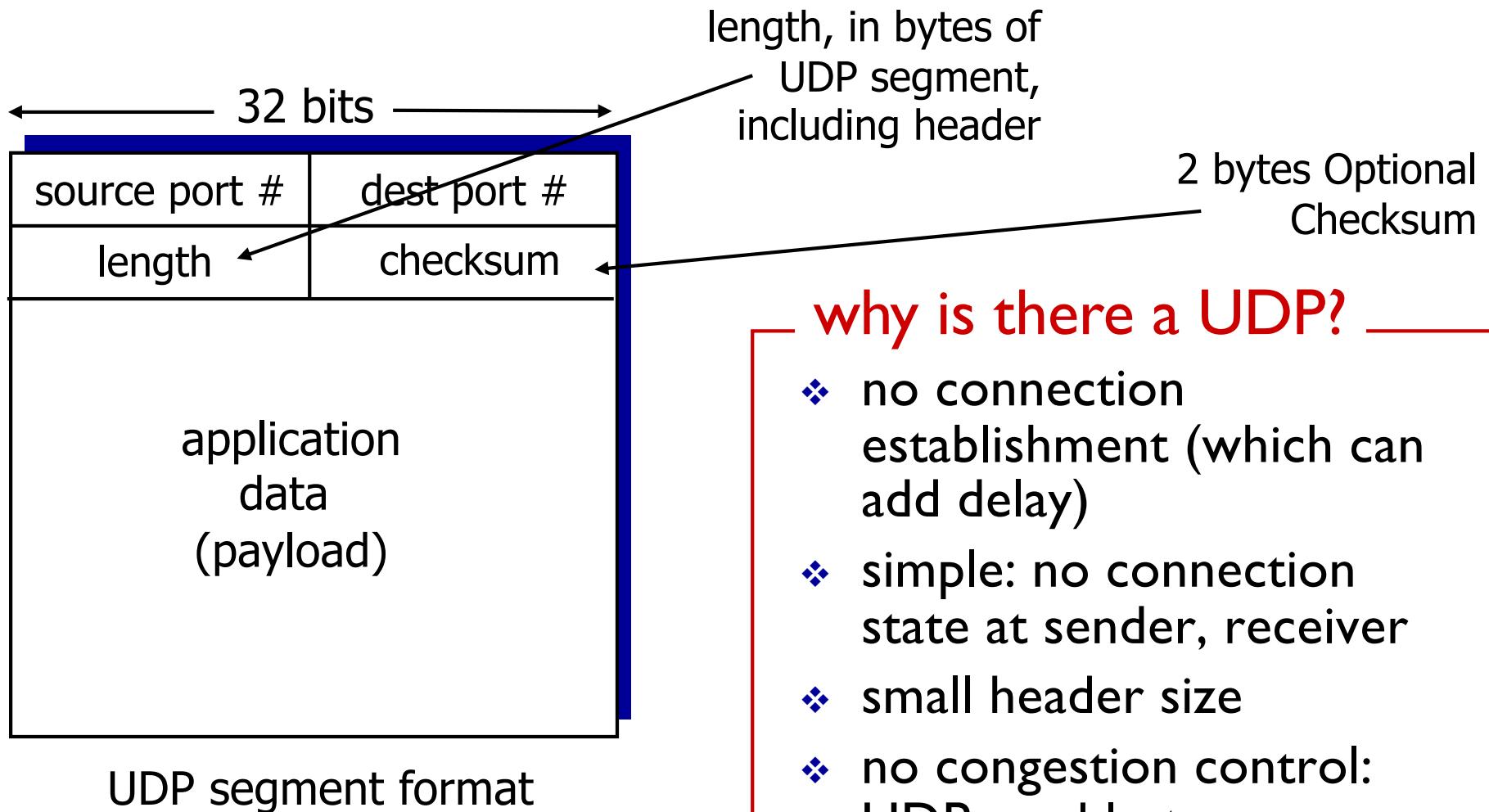
3.6 principles of congestion control

3.7 TCP congestion control

# UDP: User Datagram Protocol [RFC 768]

- ❖ “no frills,” “bare bones” Internet transport protocol
- ❖ “best effort” service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- ❖ *connectionless*:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

# UDP: segment 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
  - Router memory errors
  - Driver bugs
  - Electromagnetic interference

## 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:

- ❖ Add all the received together as 16-bit integers
- ❖ Add that to the checksum
- ❖ If the result is not 1111 1111 1111 1111, there are errors !

# Internet checksum: example

example: add two 16-bit integers

1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

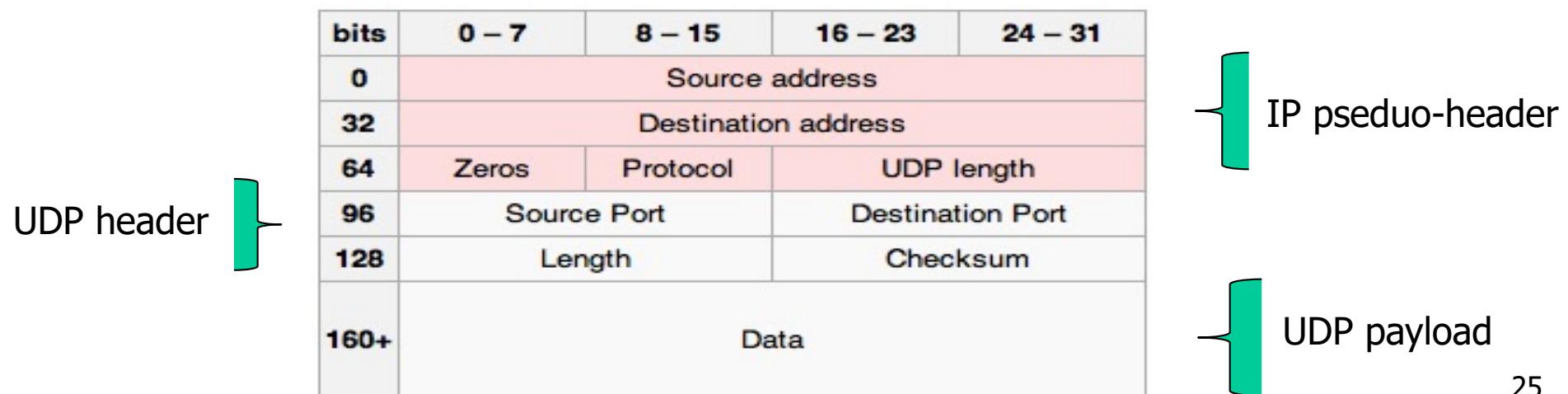
---

wraparound   
sum      1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0  
checksum    0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

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

# UDP: Checksum

- Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.
- Checksum header, data and pre-pended IP pseudo-header (some fields from the IP header)
- But the header contains the checksum itself?



# Checksum: example

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

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

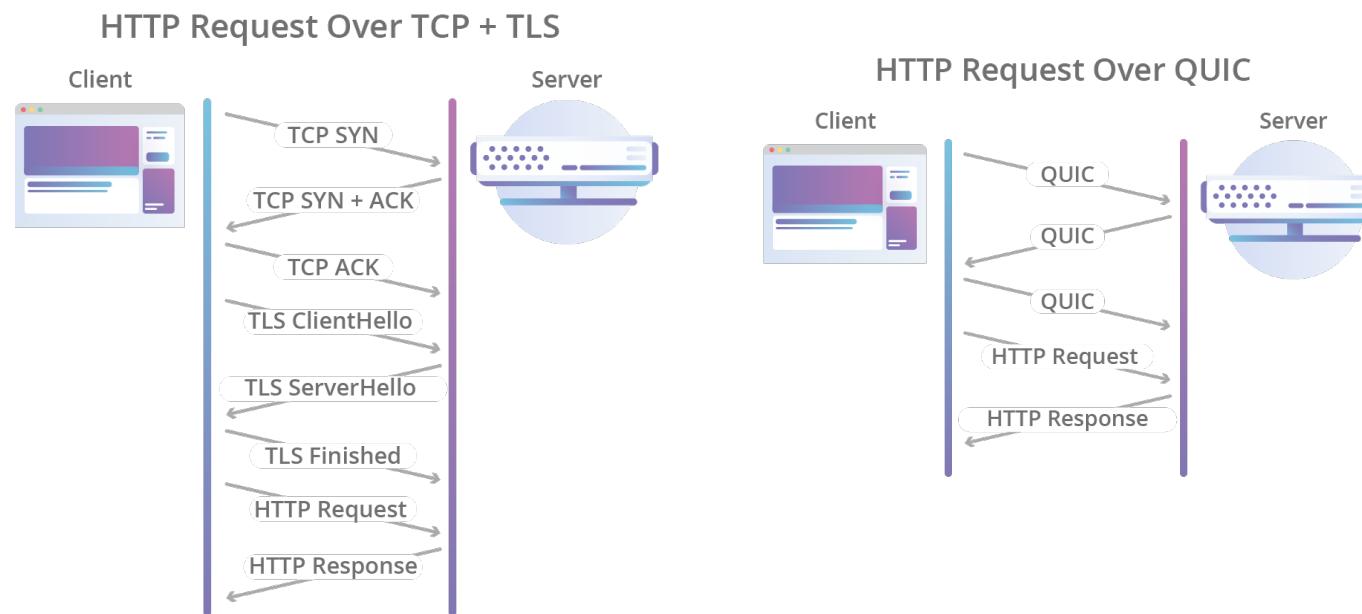
Note: TCP Checksum computation is exactly similar

# UDP Applications

- ❖ Latency sensitive/time critical
  - ❖ Quick request/response (DNS, DHCP)
  - ❖ Network management (SNMP)
  - ❖ Routing updates (RIP)
  - ❖ Voice/video chat
  - ❖ Gaming (especially FPS)
- ❖ Error correction managed by periodic messages

# QUIC: Quick UDP Internet Connections

- ❖ Core idea: HTTP/2 over UDP
  - Faster connection establishment
  - Overcomes HoL blocking due to lost packets
  - Improved congestion control
  - Forward error correction
  - Connection migration



# Transport Layer 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

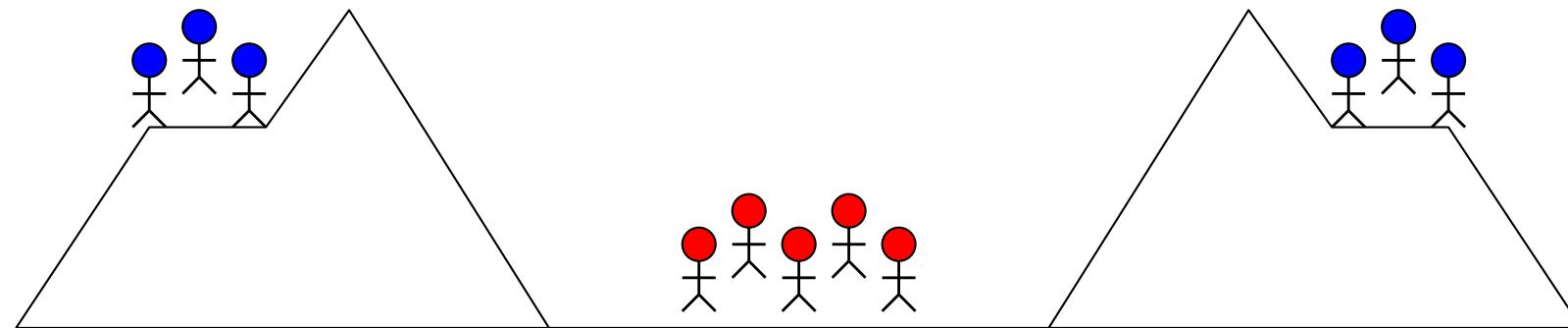
3.6 principles of congestion control

3.7 TCP congestion control

# Reliable Transport

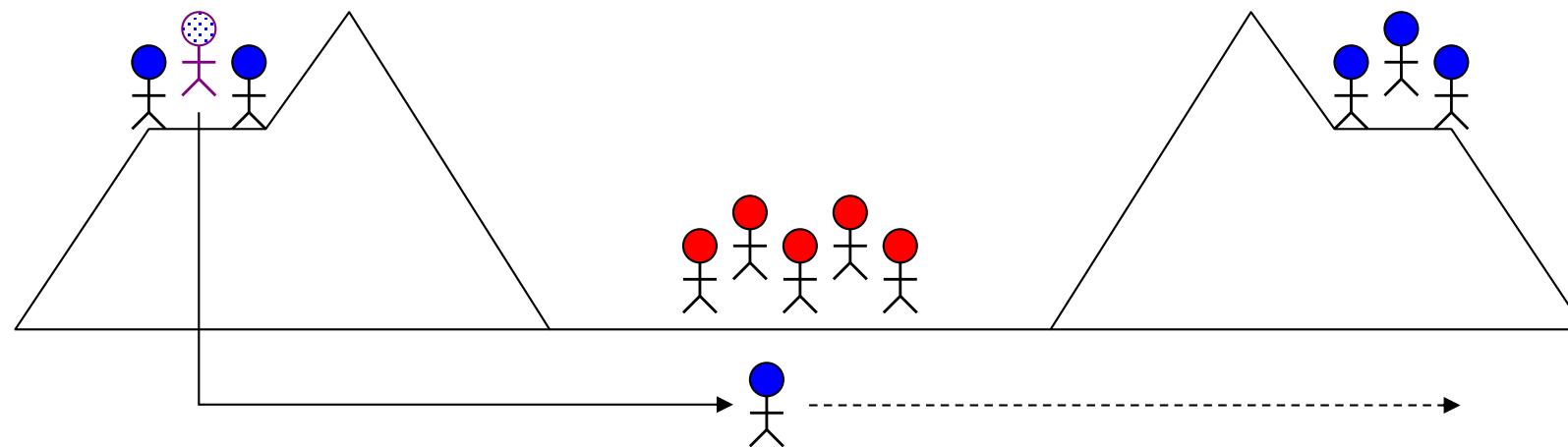
- In a perfect world, reliable transport is easy
- All the bad things best-effort can do
  - a packet is corrupted (bit errors)
  - a packet is lost (*why?*)
  - a packet is delayed (*why?*)
  - packets are reordered (*why?*)
  - a packet is duplicated (*why?*)

# The Two Generals Problem



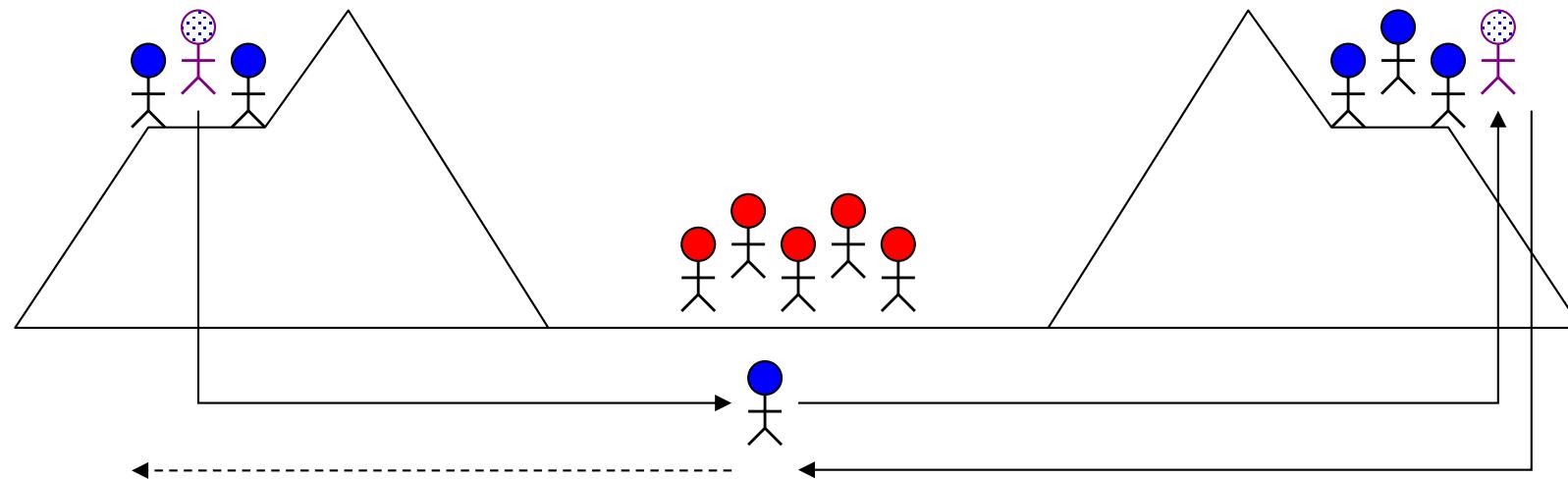
- ❖ Two army divisions (blue) surround enemy (red)
  - Each division led by a general
  - Both must agree when to simultaneously attack
  - If either side attacks alone, defeat
- ❖ Generals can only communicate via messengers
  - Messengers may get captured (unreliable channel)

# The Two Generals Problem



- ❖ How to coordinate?
  - Send messenger: “Attack at dawn”
  - What if messenger doesn’t make it?

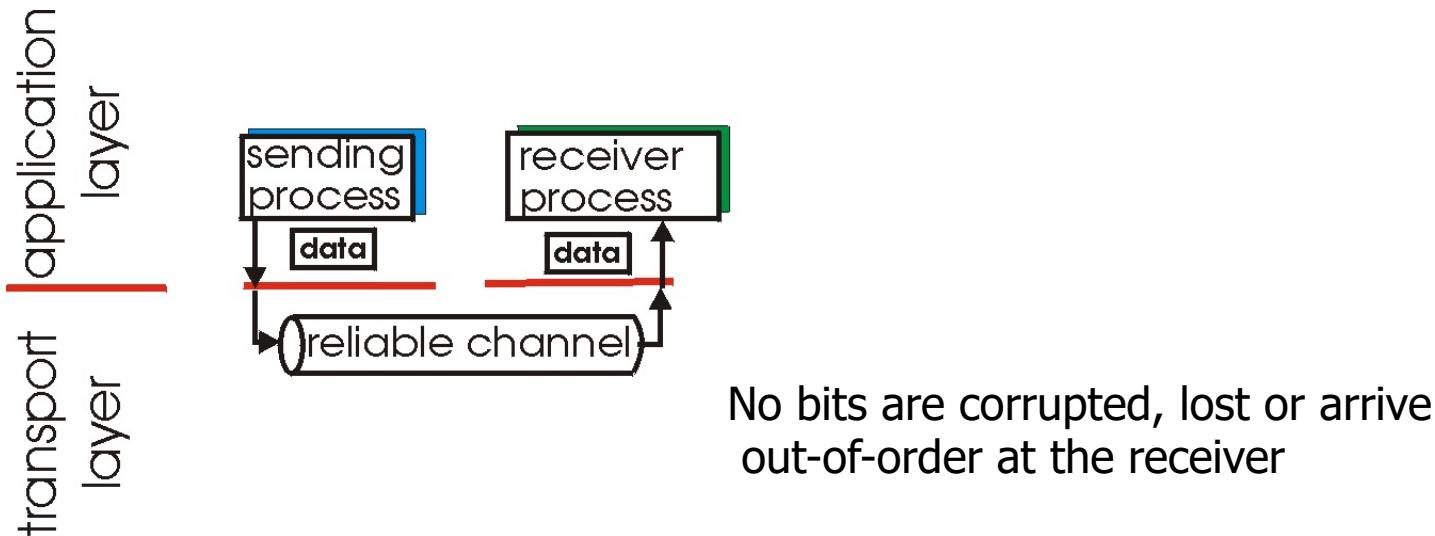
# The Two Generals Problem



- ❖ How to be sure messenger made it?
  - Send acknowledgement: “We received message”

# Principles of reliable data transfer

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

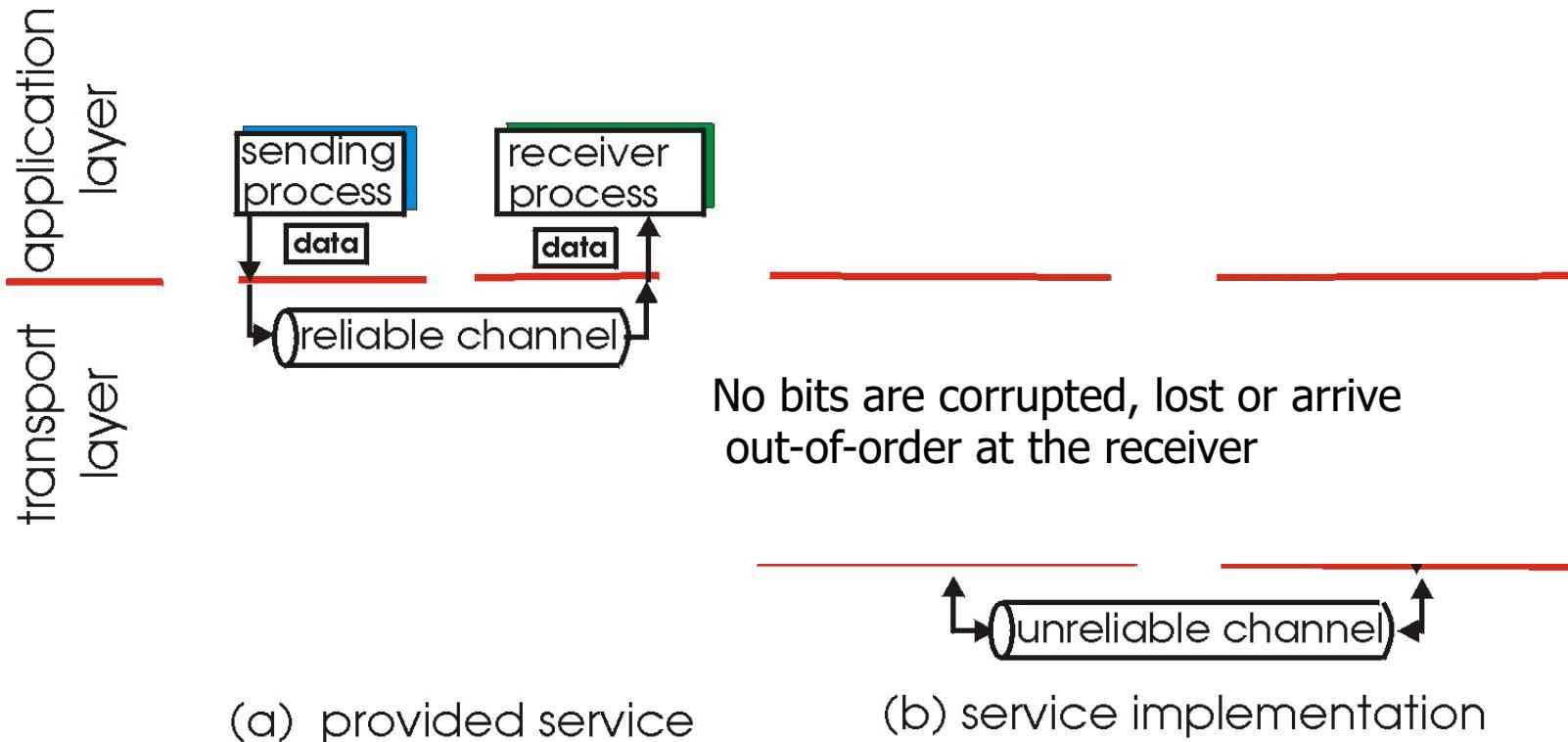


(a) provided service

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

# Principles of reliable data transfer

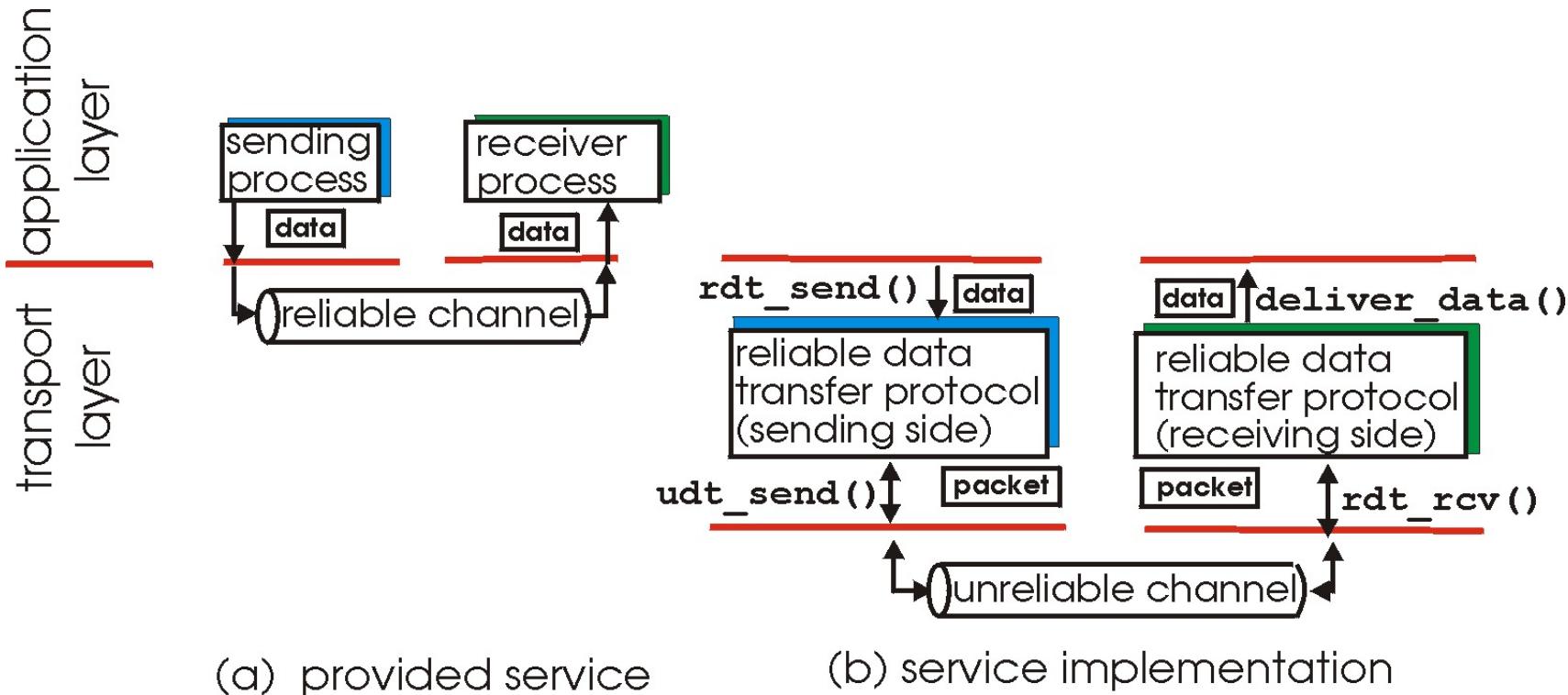
- ❖ important in application, transport, link layers
  - top-10 list of important networking topics!



- ❖ 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

We'll:

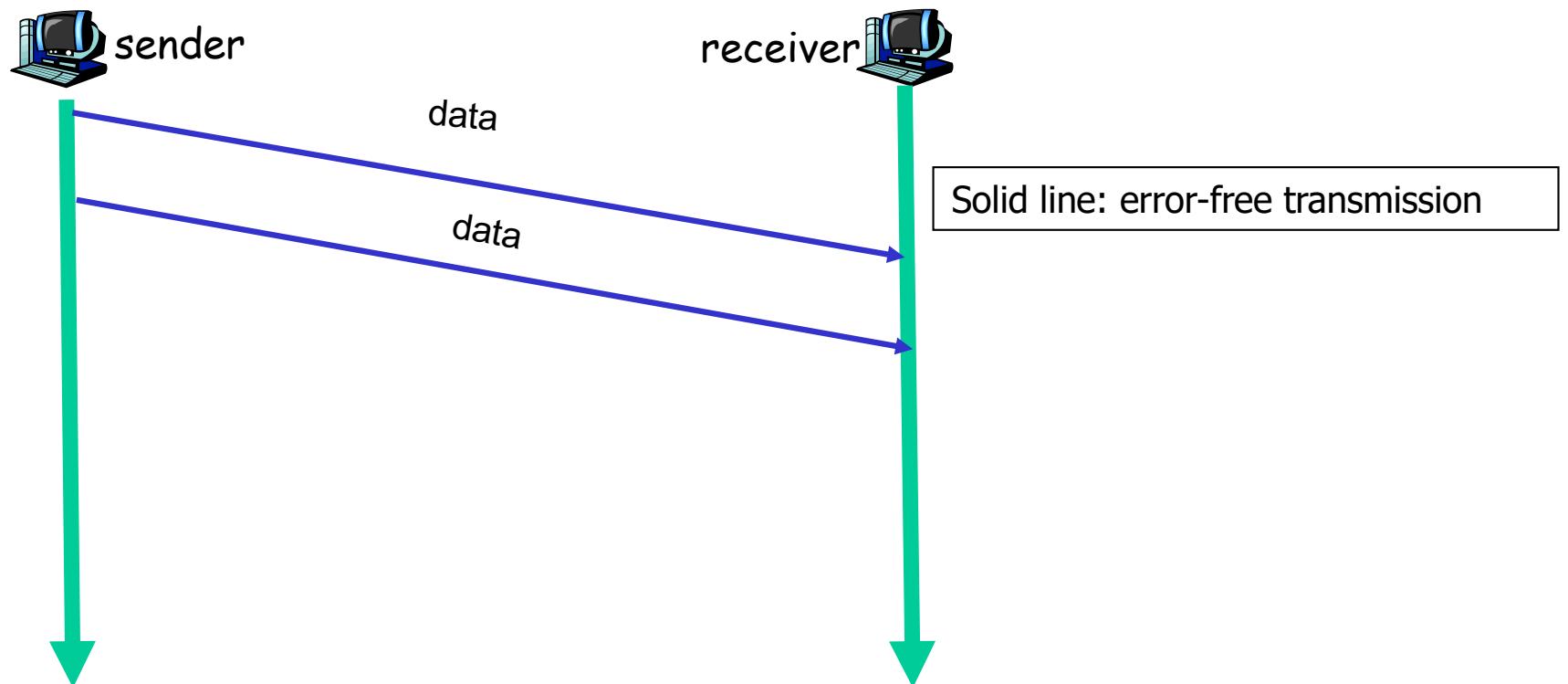
- Incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- Consider only unidirectional data transfer
  - but control info will flow on both directions!
- Channel will not re-order packets

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

## rdt1.0: reliable transfer over a reliable channel

- Underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- Transport layer does nothing !

# Global Picture of rdt1.0



## rdt2.0: channel with bit errors

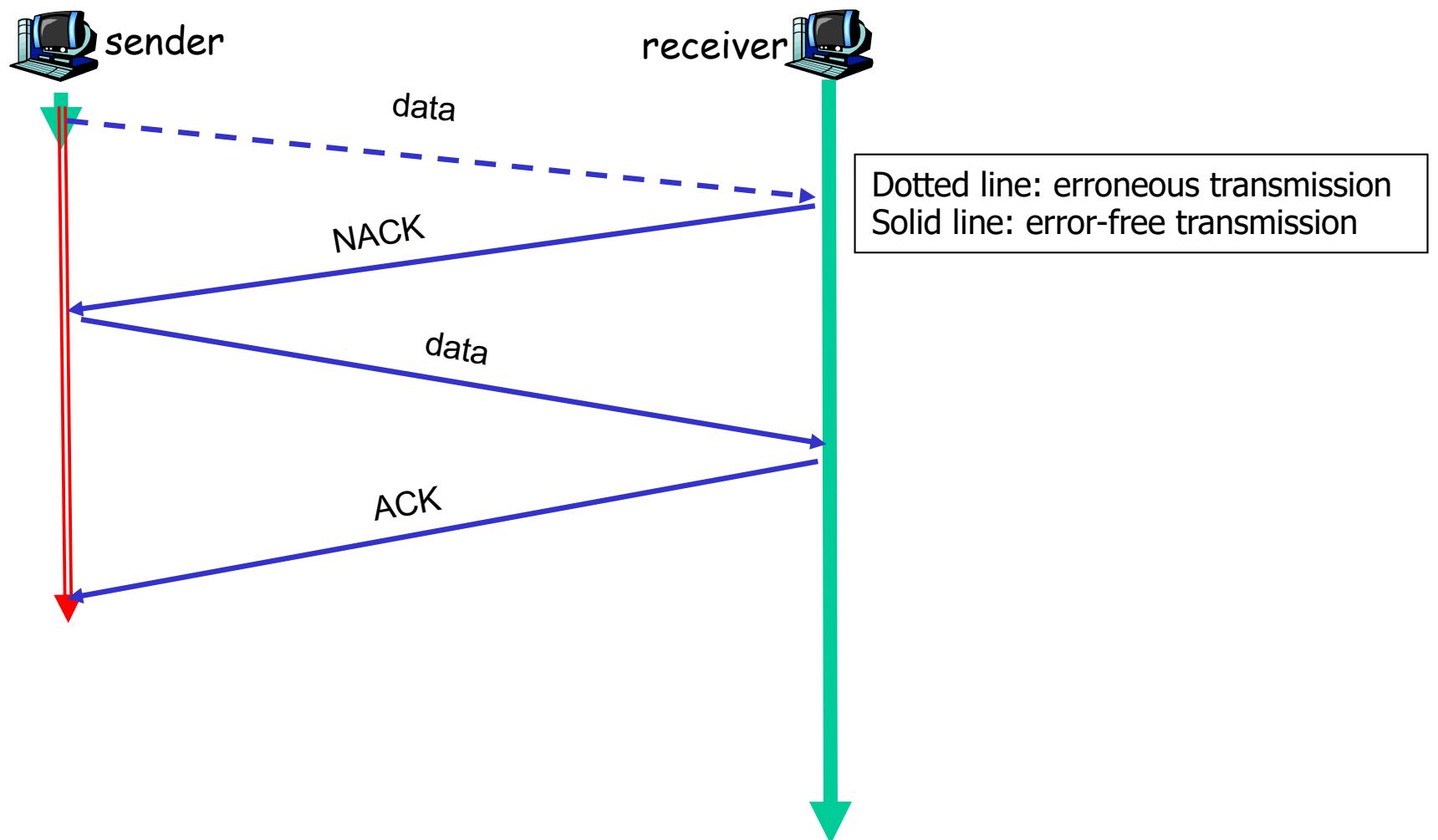
- ❖ underlying channel may flip bits in packet
  - checksum to detect bit errors
- ❖ *the question: how to recover from errors:*

*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
  - retransmission

# Global Picture of rdt2.0



# rdt2.0 has a fatal flaw!

## what happens if ACK/NAK corrupted?

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

## handling duplicates:

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

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

# rdt2.1: discussion

## sender:

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

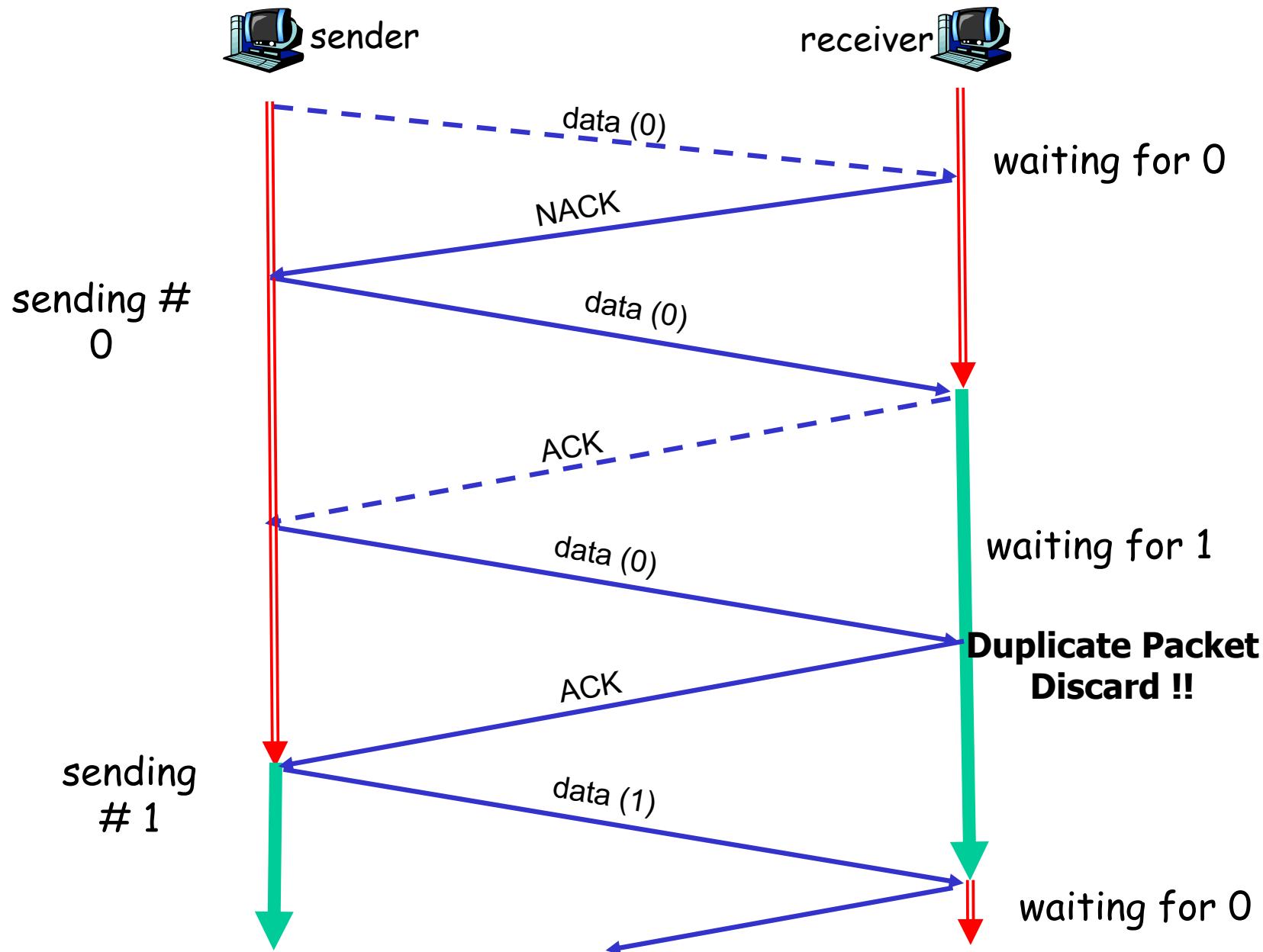
## receiver:

- ❖ 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

New Measures: Sequence Numbers, Checksum for ACK/NACK,  
Duplicate detection

# Another Look at rdt2.1

Dotted line: erroneous transmission  
Solid line: error-free transmission

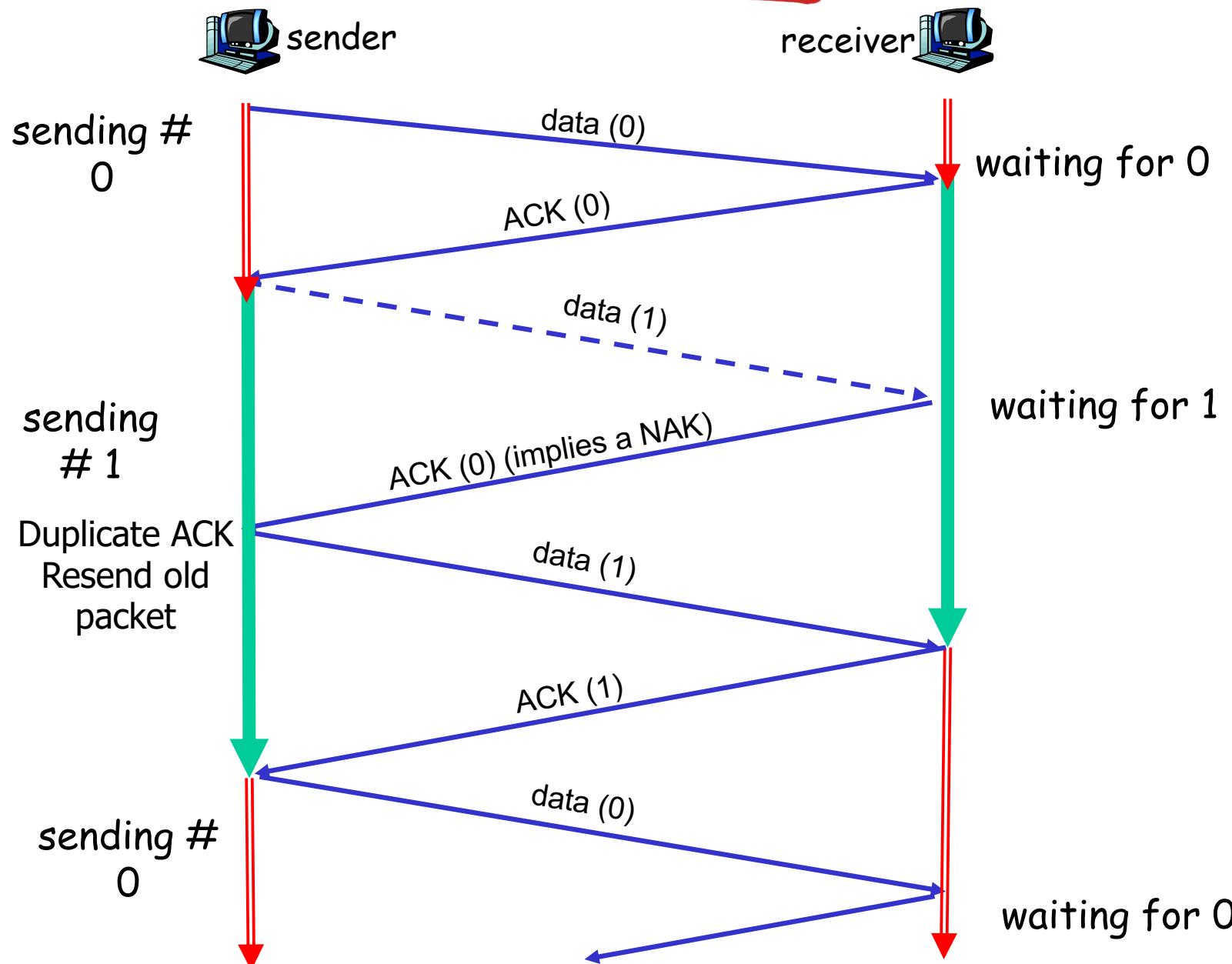


## rdt2.2: a NAK-free protocol

- ❖ 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: Example

Dotted line: erroneous transmission  
Solid line: error-free transmission



# rdt3.0: channels with errors and loss

## new assumption:

underlying channel can also loose packets (data, ACKs)

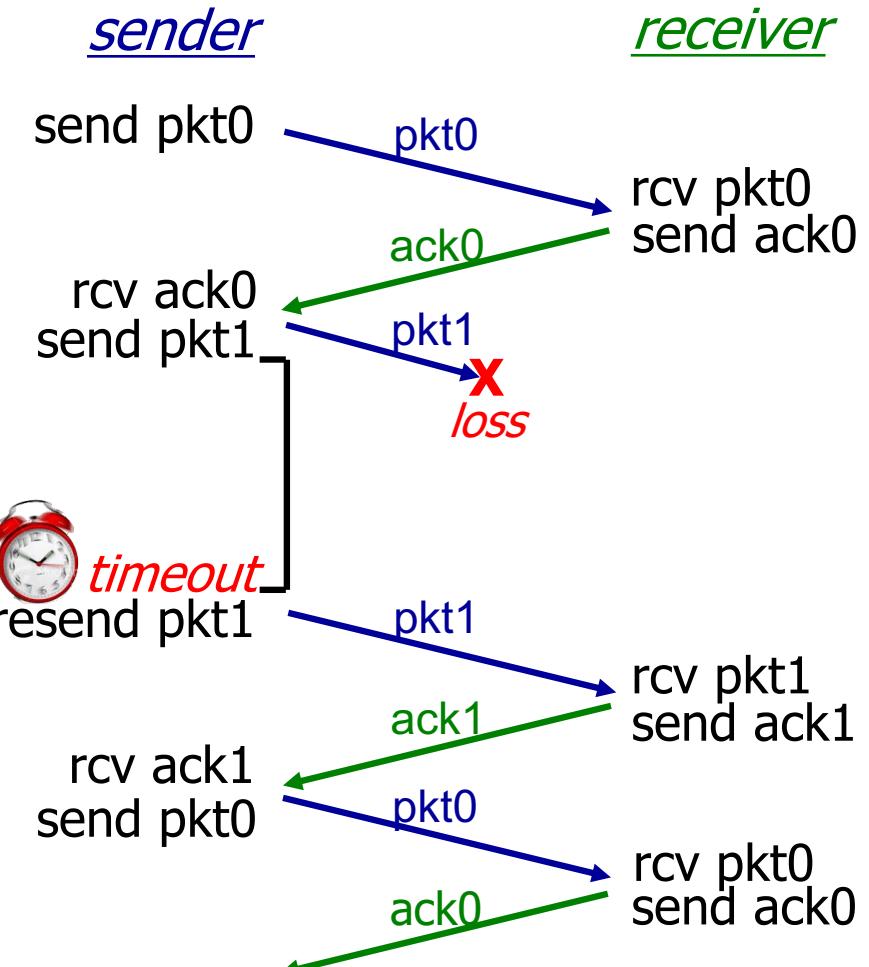
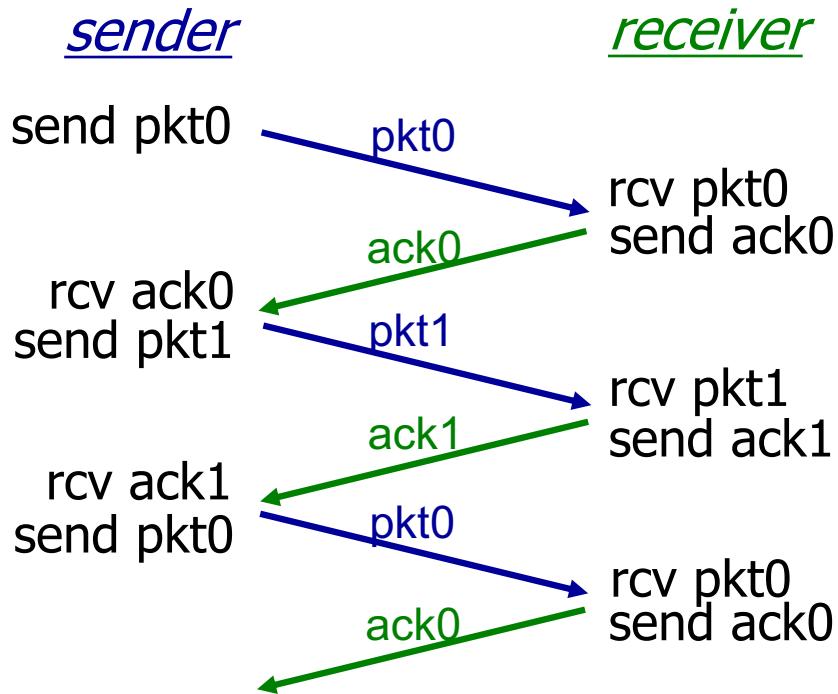
- checksum, seq. #, ACKs, retransmissions will be of help ... but not enough

## approach: sender waits

“reasonable” amount of time for ACK

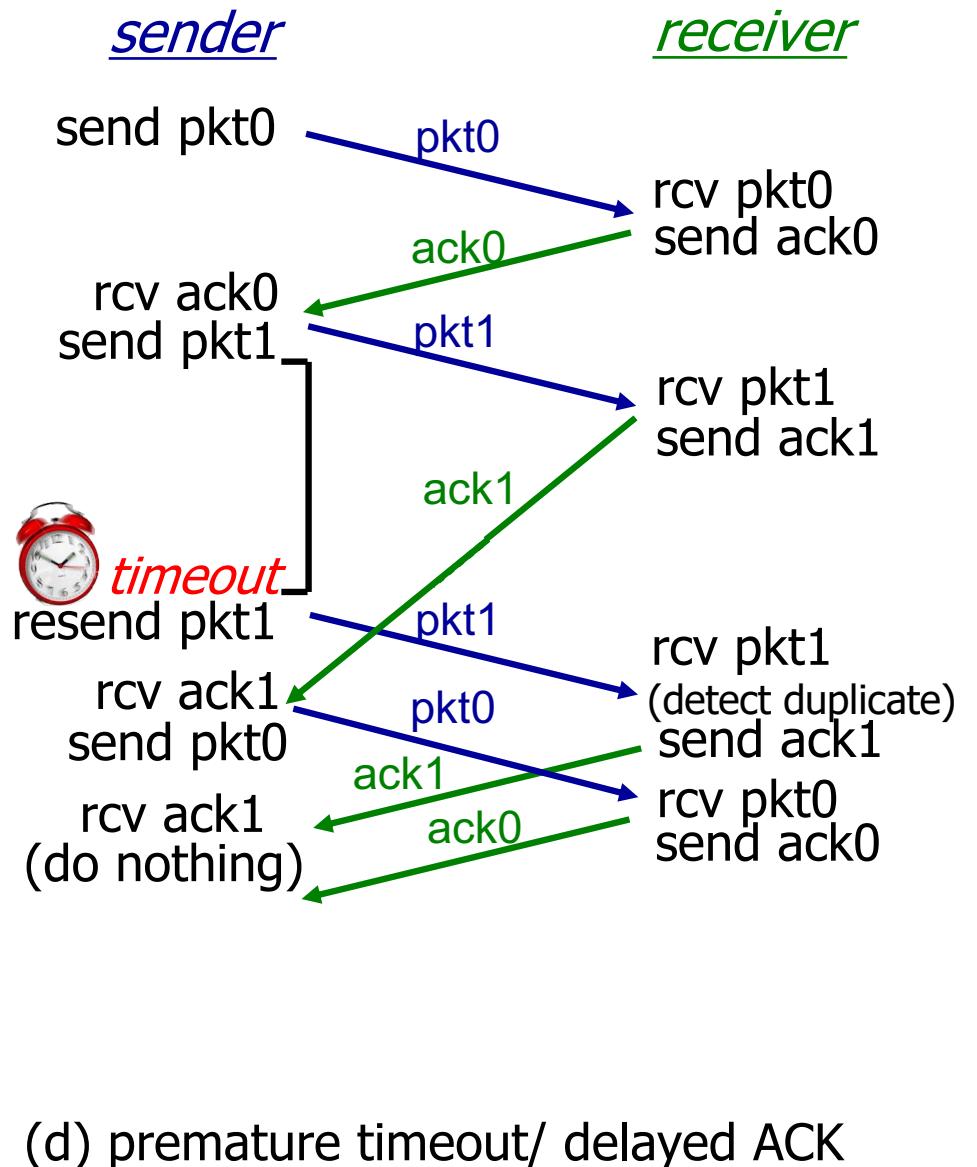
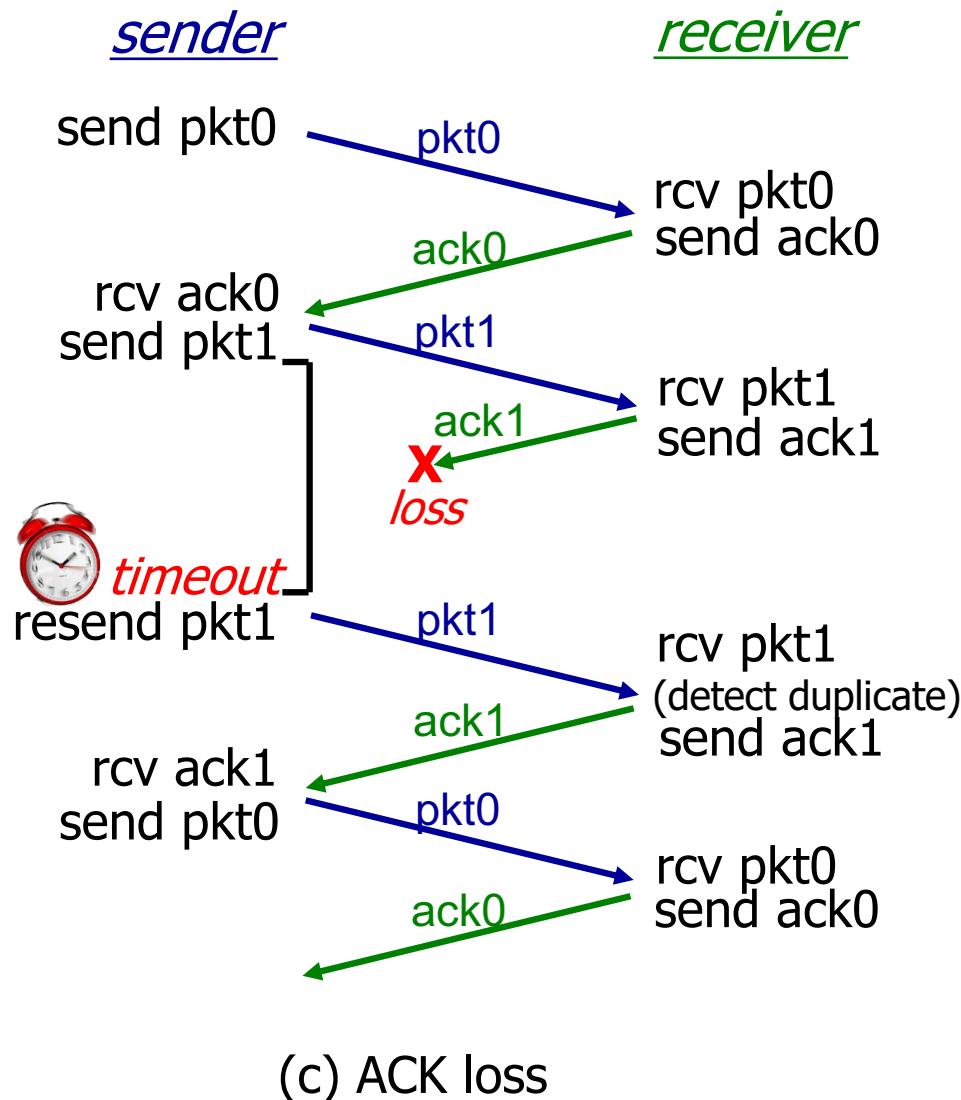
- ❖ retransmits if no ACK received in this time
- ❖ if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- ❖ requires countdown timer
- ❖ No retransmission on duplicate ACKs

# rdt3.0 in action



(b) packet loss

# rdt3.0 in action



# Quiz: Reliable Data Transfer



- ❖ Which of the following are needed for reliable data transfer with only packet corruption (and no loss or reordering)? Use only as much as is strictly needed.

**ANSWER: d)**

- a) Checksums
- b) Checksums, ACKs, NACKs
- c) Checksums, ACKs
- d) Checksums, ACKs, sequence numbers
- e) Checksums, ACKs, NACKs, sequence numbers

# Quiz: Reliable Data Transfer



- ❖ If packets (and ACKs and NACKs) could be lost which of the following is true of RDT 2.1 (or 2.2)?
  - a) Reliable in-order delivery is still achieved
  - b) The protocol will get stuck
  - c) The protocol will continue making progress but may skip delivering some messages

**ANSWER: b)**

[www.zeetings.com/salil](http://www.zeetings.com/salil)

# Quiz: Reliable Data Transfer

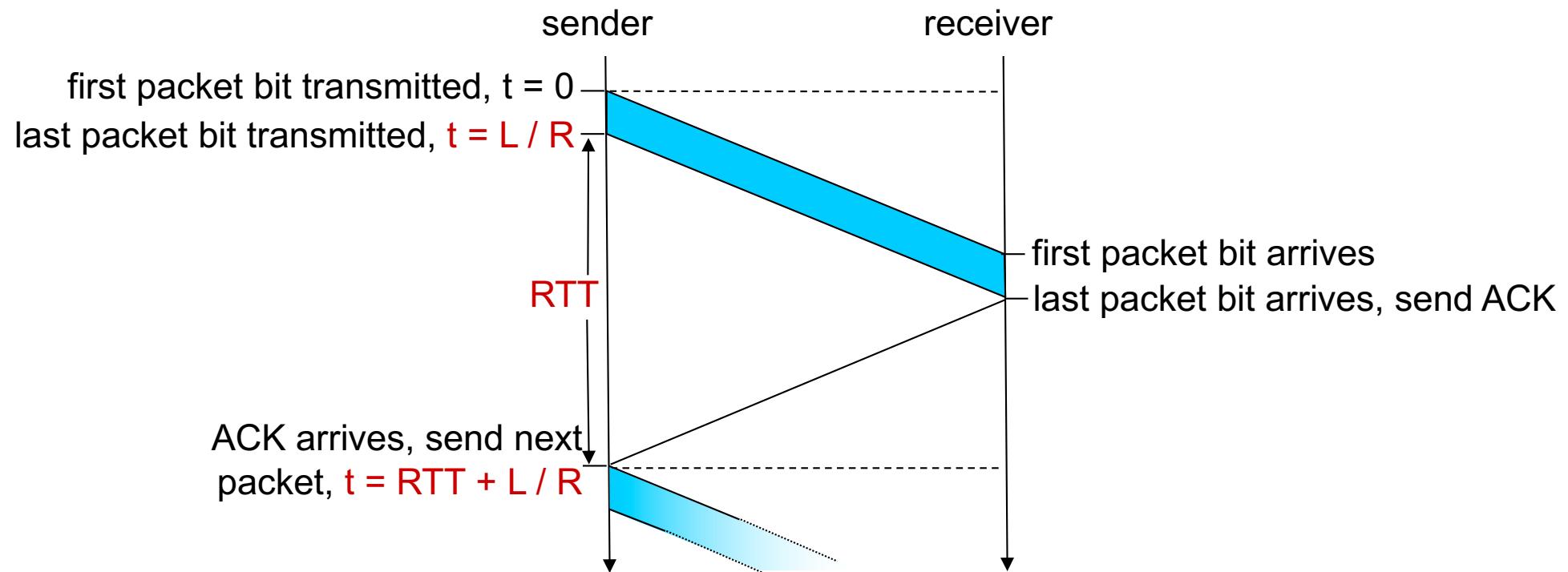


- ❖ Which of the following are needed for reliable data transfer to handle packet corruption and loss?  
Use only as much as is strictly needed.
  - a) Checksums, timeouts
  - b) Checksums, ACKs, sequence numbers
  - c) Checksums, ACKs, timeouts
  - d) Checksums, ACKs, timeouts, sequence numbers
  - e) Checksums, ACKs, NACKs, timeouts, sequence numbers

[www.zeetings.com/salil](http://www.zeetings.com/salil)

**ANSWER: d)**

## rdt3.0: stop-and-wait operation



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

## Performance of rdt3.0

- rdt3.0 is correct, but performance stinks
- e.g.: 1 Gbps link, 8000 bit packet and 30msec RTT:

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

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

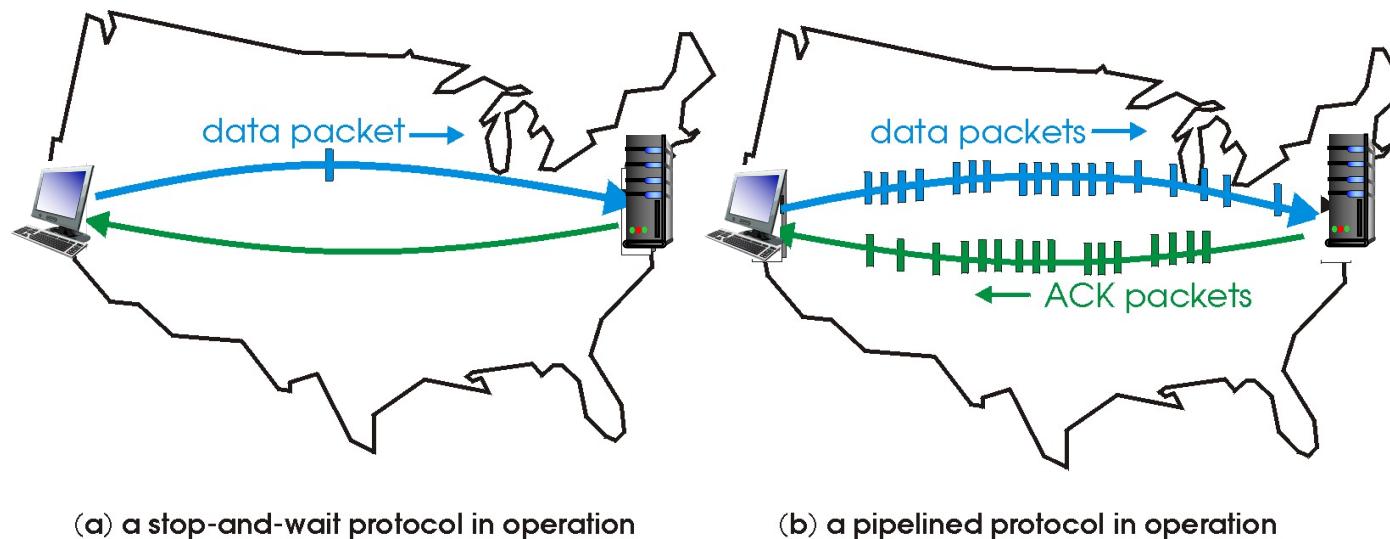
$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

- RTT=30 msec, 1KB pkt every 30.008 msec: 33kB/sec thruput over 1 Gbps link
- Network protocol limits use of physical resources!

# Pipelined protocols

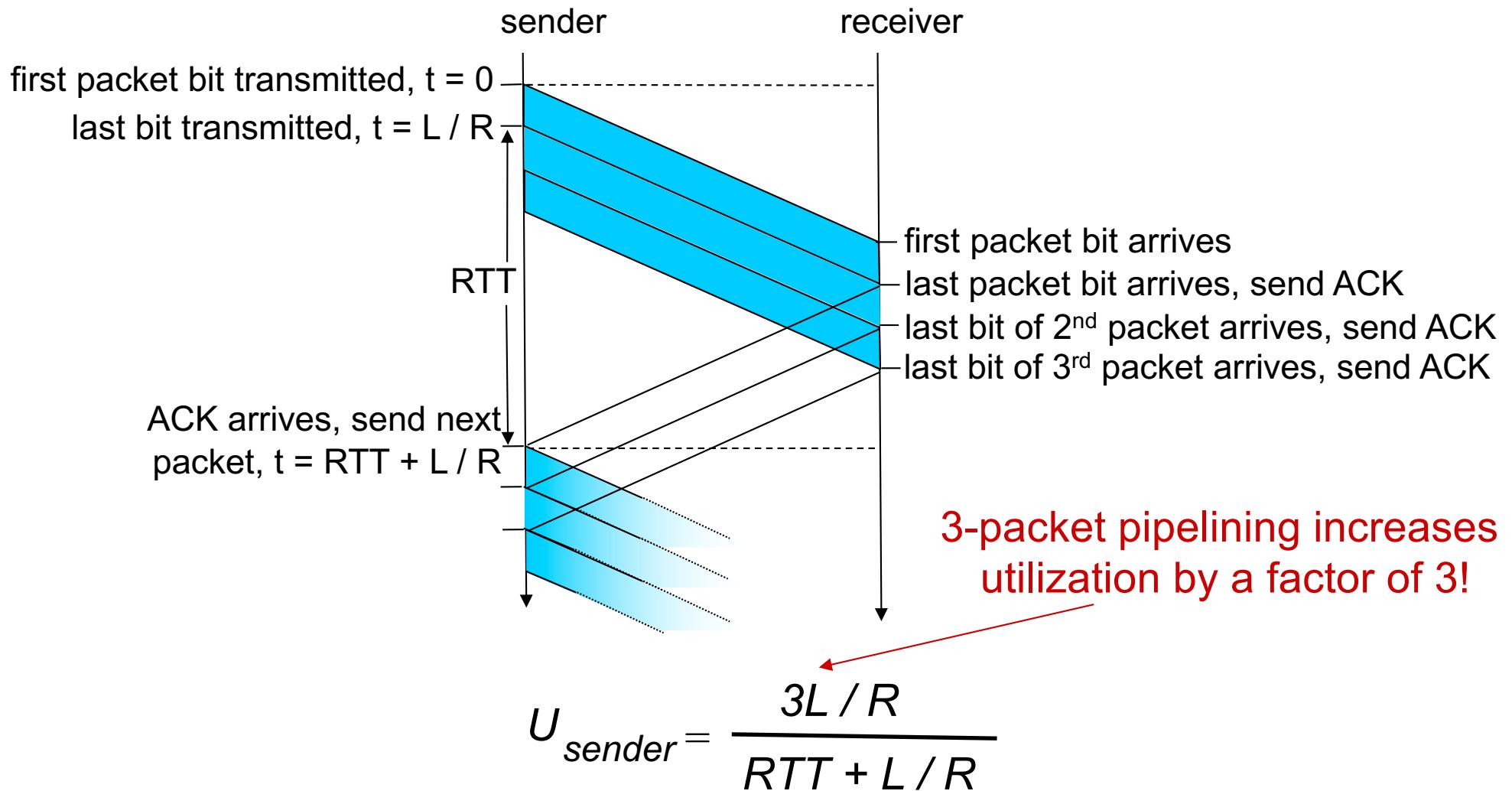
**pipelining:** sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

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



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

# Pipelining: increased utilization



# Pipelined protocols: overview

## Go-Back-N:

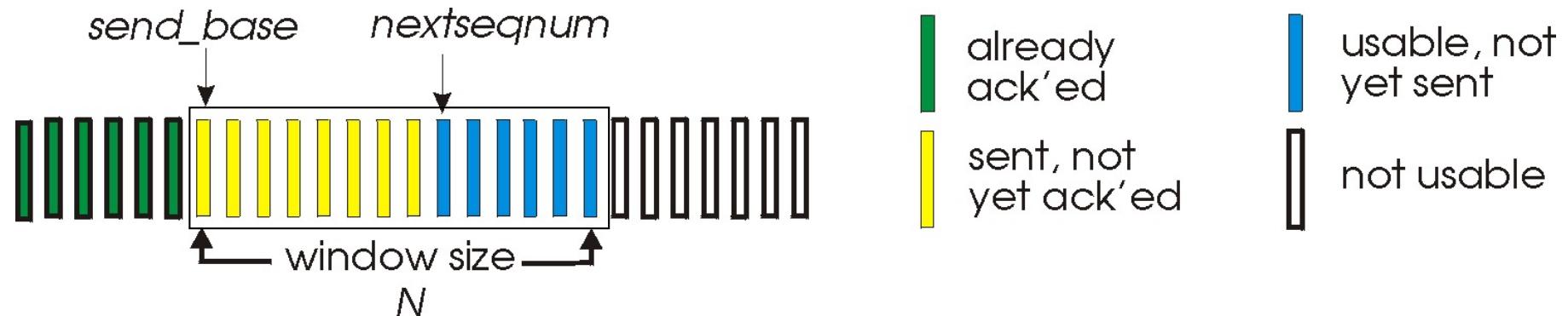
- Sender can have up to N unacked packets in pipeline
- Sender has **single timer** for oldest unacked packet, when timer expires, retransmit *all* unacked packets
- There is no buffer available at Receiver, out of order packets are discarded
- Receiver only sends **cumulative ack**, doesn't ack new packet if there's a gap

## Selective Repeat:

- Sender can have up to N unacked packets in pipeline
- Sender maintains timer for each unacked packet, when timer expires, retransmit only that unacked packet
- Receiver has buffer, can accept **out of order** packets
- Receiver sends **individual ack** for each 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

Applets: [http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_2/applets/go-back-n/go-back-n.html](http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/go-back-n/go-back-n.html)  
[http://www.ccs-labs.org/teaching/rn/animations/gbn\\_sr/](http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/)

# GBN in action

*sender window (N=4)*

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

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

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

*sender*

send pkt0  
send pkt1  
send pkt2  
send pkt3  
(wait)

rcv ack0, send pkt4  
rcv ack1, send pkt5

send pkt2  
send pkt3  
send pkt4  
send pkt5

*receiver*

receive pkt0, send ack0  
receive pkt1, send ack1

receive pkt3, discard,  
(re)send ack1

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

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



*pkt 2 timeout*

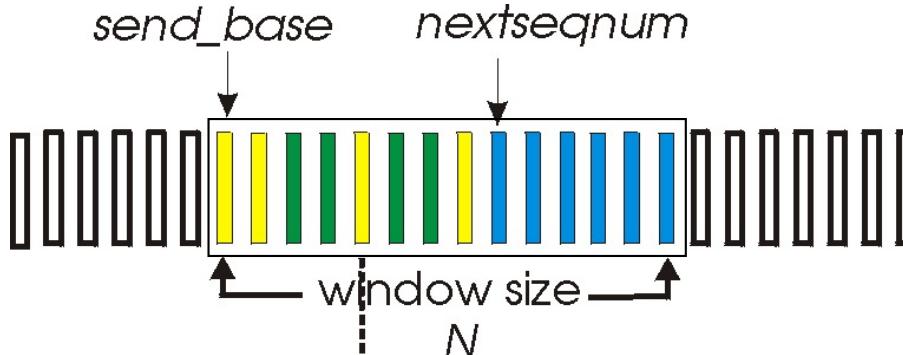
# Selective repeat

- ❖ 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

Applet: [http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_3/applets/SelectRepeat/SR.html](http://media.pearsoncmg.com/aw/aw_kurose_network_3/applets/SelectRepeat/SR.html)

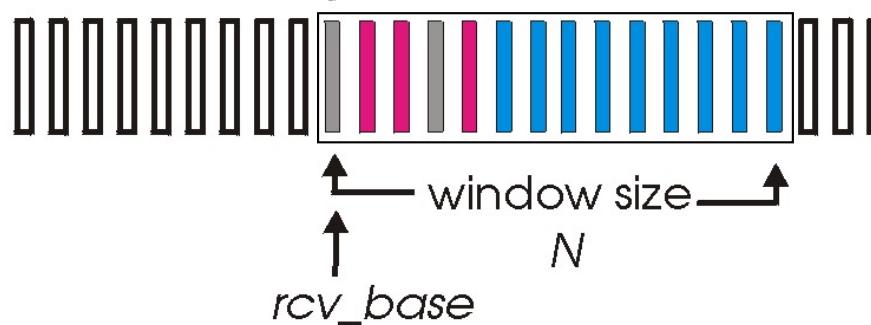
[http://www.ccs-labs.org/teaching/rn/animations/gbn\\_sr/](http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/)

# Selective repeat: sender, receiver windows



already  
ack'ed  
sent, not  
yet ack'ed  
usable, not  
yet sent  
not usable

(a) sender view of sequence numbers



out of order  
(buffered) but  
already ack'ed  
Expected, not  
yet received  
acceptable  
(within window)  
not usable

(b) receiver view of sequence numbers

# Selective repeat

## sender

### **data from above:**

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

### **timeout(n):**

- ❖ resend pkt n, restart timer

### **ACK(n) in [sendbase,sendbase+N-1]:**

- ❖ 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

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

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

*sender*

send pkt0  
send pkt1  
send pkt2  
send pkt3  
(wait)

rcv ack0, send pkt4  
rcv ack1, send pkt5

record ack3 arrived



*pkt 2 timeout*

send pkt2

record ack4 arrived

record ack5 arrived

*receiver*

receive pkt0, send ack0  
receive pkt1, send ack1

receive pkt3, buffer,  
send ack3

receive pkt4, buffer,  
send ack4

receive pkt5, buffer,  
send ack5

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

*Q: what happens when ack2 arrives?*

# Selective repeat: dilemma

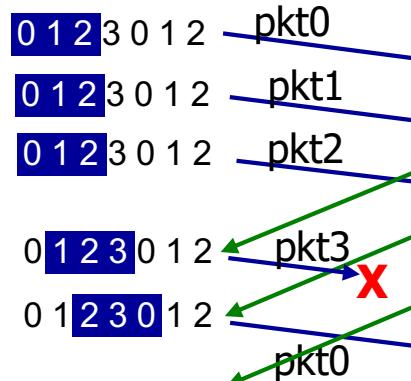
example:

- ❖ 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)?

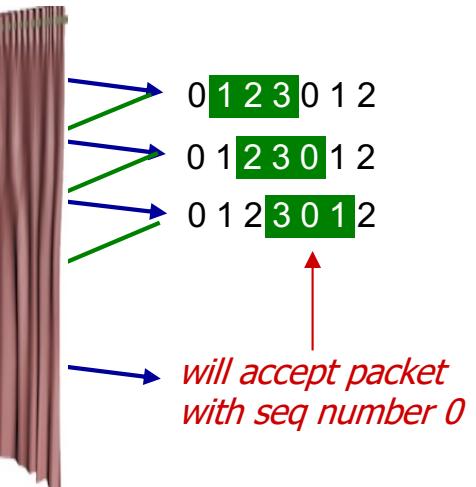
A: Sender window size  $\leq \frac{1}{2}$  of Sequence number space

sender window  
(after receipt)

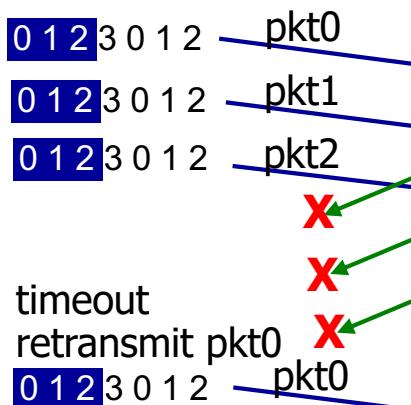


(a) no problem

receiver window  
(after receipt)

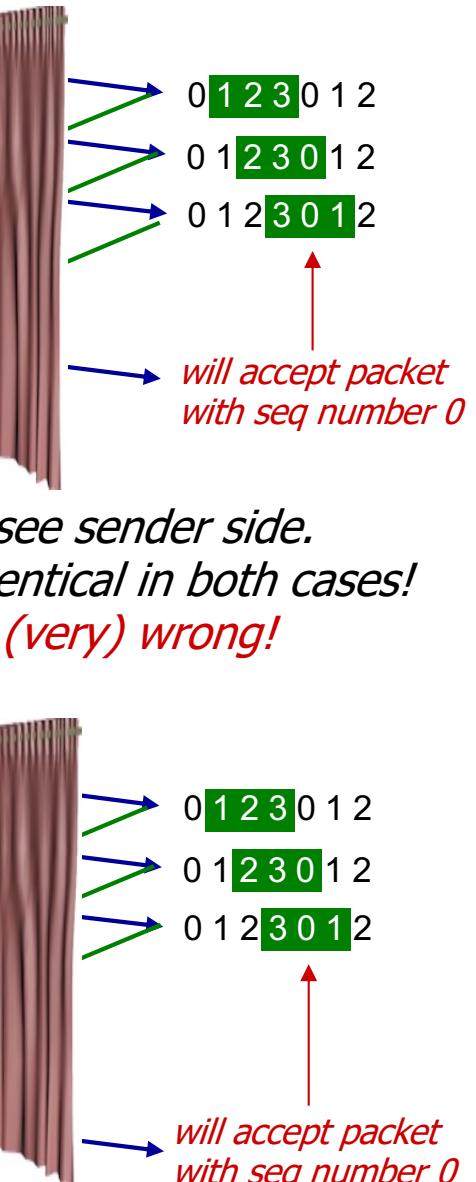


*receiver can't see sender side.  
receiver behavior identical in both cases!  
something's (very) wrong!*



(b) oops!

receiver window  
(after receipt)



# Recap: components of a solution

- ❖ Checksums (for error detection)
- ❖ Timers (for loss detection)
- ❖ Acknowledgments
  - cumulative
  - selective
- ❖ Sequence numbers (duplicates, windows)
- ❖ Sliding Windows (for efficiency)
  
- ❖ Reliability protocols use the above to decide when and what to retransmit or acknowledge

# Quiz: GBN, SR



- ❖ Which of the following is not true?
  - a) GBN uses cumulative ACKs, SR uses individual ACKs
  - b) Both GBN and SR use timeouts to address packet loss
  - c) GBN maintains a separate timer for each outstanding packet
  - d) SR maintains a separate timer for each outstanding packet
  - e) Neither GBN nor SR use NACKs

# Quiz: GBN, SR



- ❖ Suppose a receiver that has received all packets up to and including sequence number 24 and next receives packet 27 and 28. In response, what are the sequence numbers in the ACK(s) sent out by the GBN and SR receiver, respectively?
  - a) [27, 28], [28, 28]
  - b) [24, 24], [27, 28] **ANSWER: b)**
  - c) [27, 28], [27, 28]
  - d) [25, 25], [25, 25]
  - e) [nothing], [27, 28]

# Summary

- ❖ Multiplexing/Demultiplexing
- ❖ UDP
- ❖ Reliable Data Transfer
  - Stop-and-wait protocols
  - Sliding window protocols
- ❖ Up Next:
  - TCP
  - Congestion Control

# **COMP 3331/9331:** **Computer Networks and** **Applications**

**Week 5**

**Transport Layer (Continued)**

**Reading Guide: Chapter 3, Sections: 3.5 – 3.7**

# Transport Layer 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

# Practical Reliability Questions

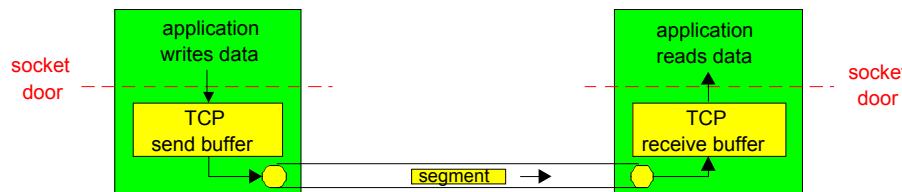
- ❖ How do the sender and receiver keep track of outstanding pipelined segments?
- ❖ How many segments should be pipelined?
- ❖ How do we choose sequence numbers?
- ❖ What does connection establishment and teardown look like?
- ❖ How should we choose timeout values?

# TCP: Overview

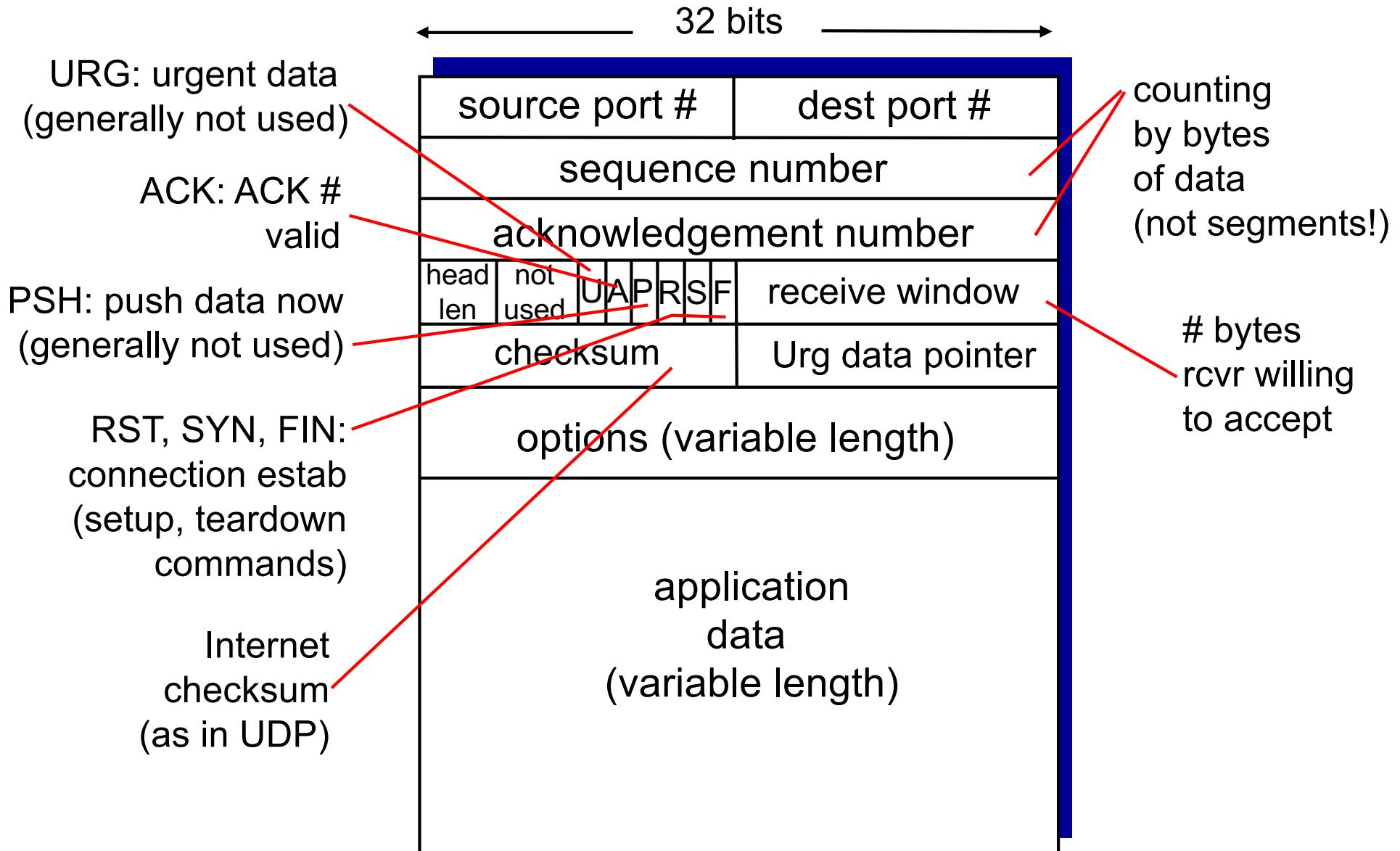
RFCs: 793, 1122, 1323, 2018, 2581

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

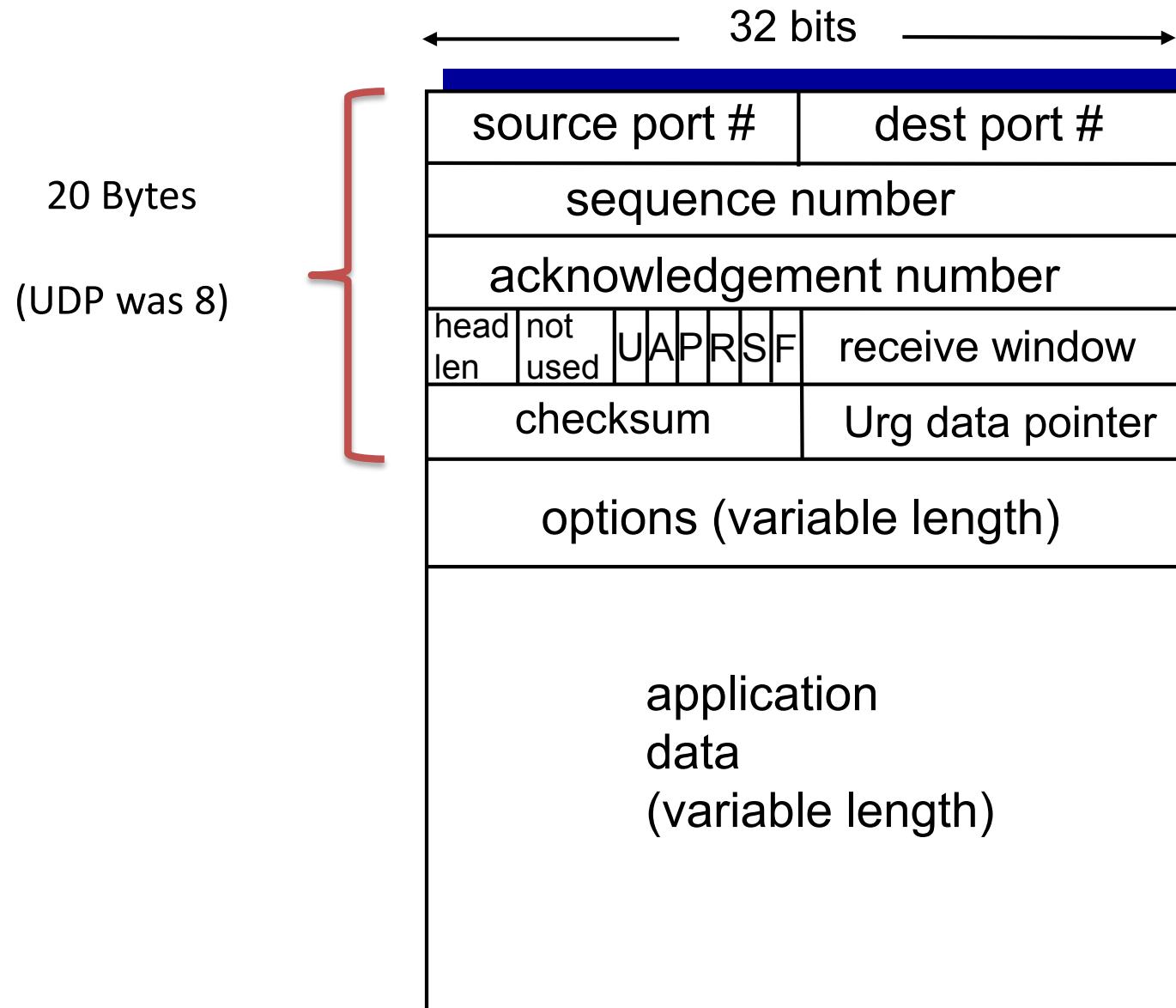
- ❖ **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- ❖ **connection-oriented:**
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- ❖ **flow controlled:**
  - sender will not overwhelm receiver



# TCP segment structure



# TCP segment structure



# Transport Layer 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

# Recall: Components of a solution for reliable transport

---

- ❖ Checksums (for error detection)
- ❖ Timers (for loss detection)
- ❖ Acknowledgments
  - Cumulative
  - Selective
- ❖ Sequence numbers (duplicates, windows)
- ❖ Sliding Windows (for efficiency)
  - Go-Back-N (GBN)
  - Selective Repeat (SR)

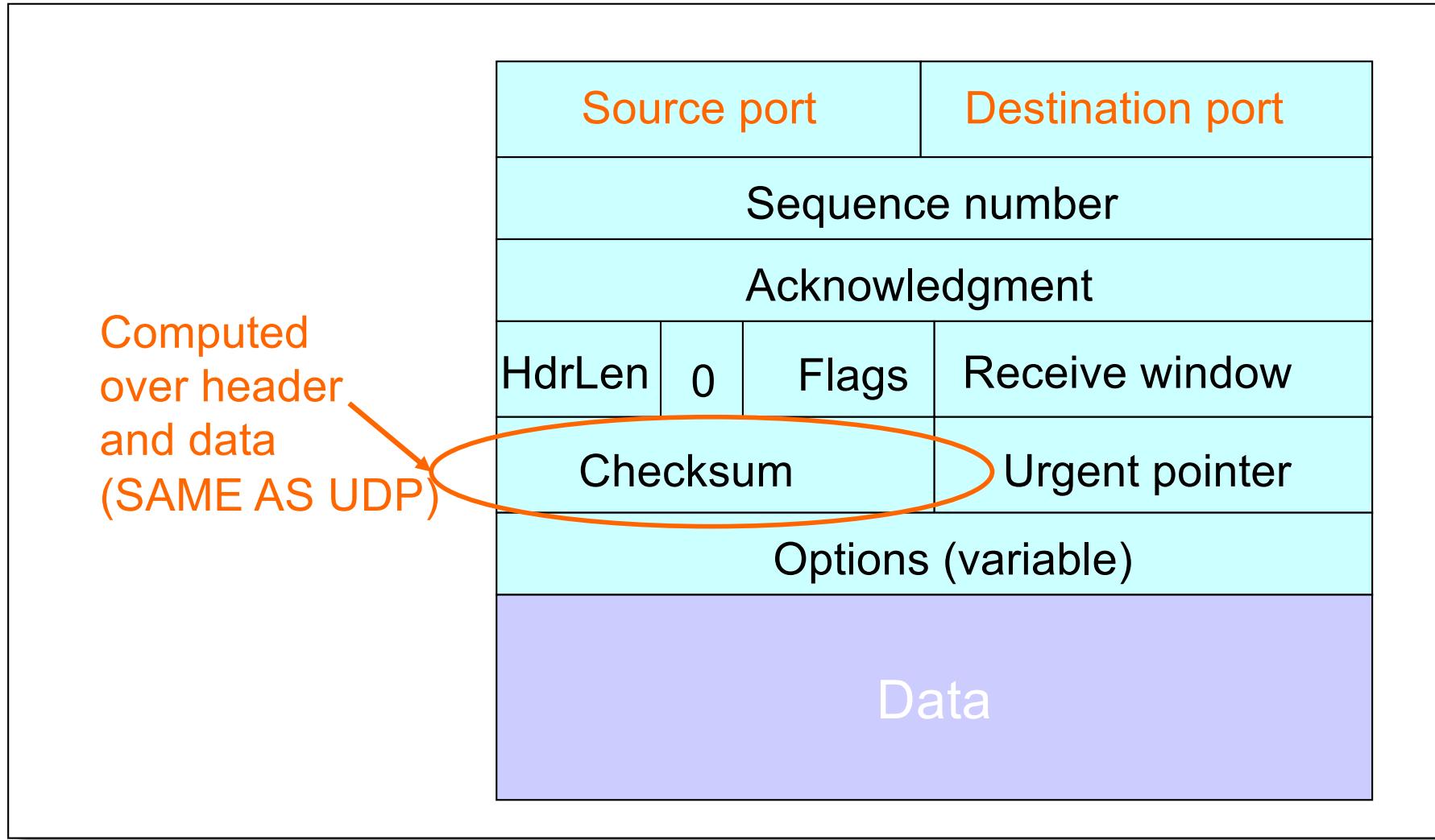
# What does TCP do?

Many of our previous ideas, but some key differences

- ❖ Checksum

# TCP Header

---



# What does TCP do?

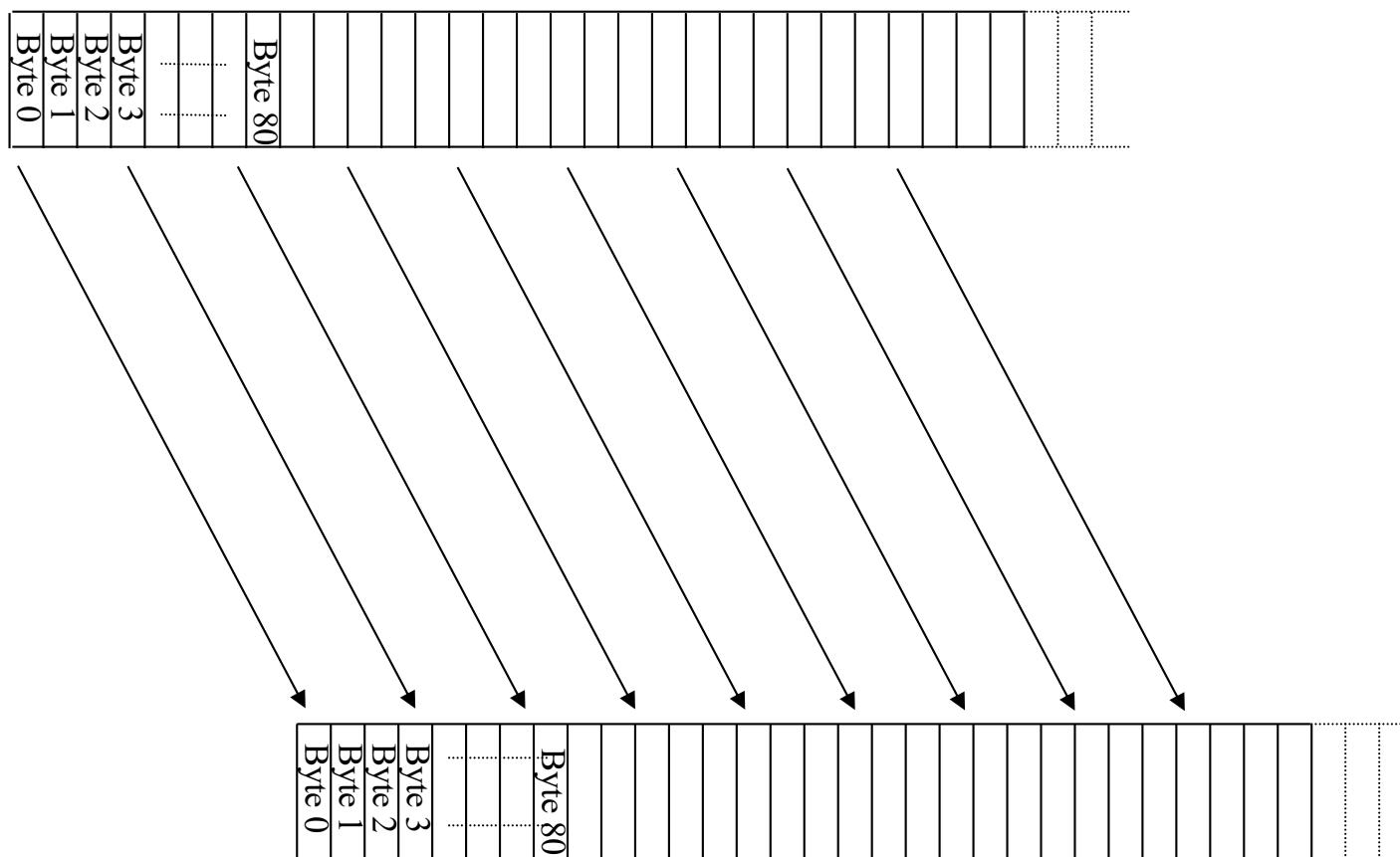
---

Many of our previous ideas, but some key differences

- ❖ Checksum
- ❖ **Sequence numbers are byte offsets**

# TCP “Stream of Bytes” Service ..

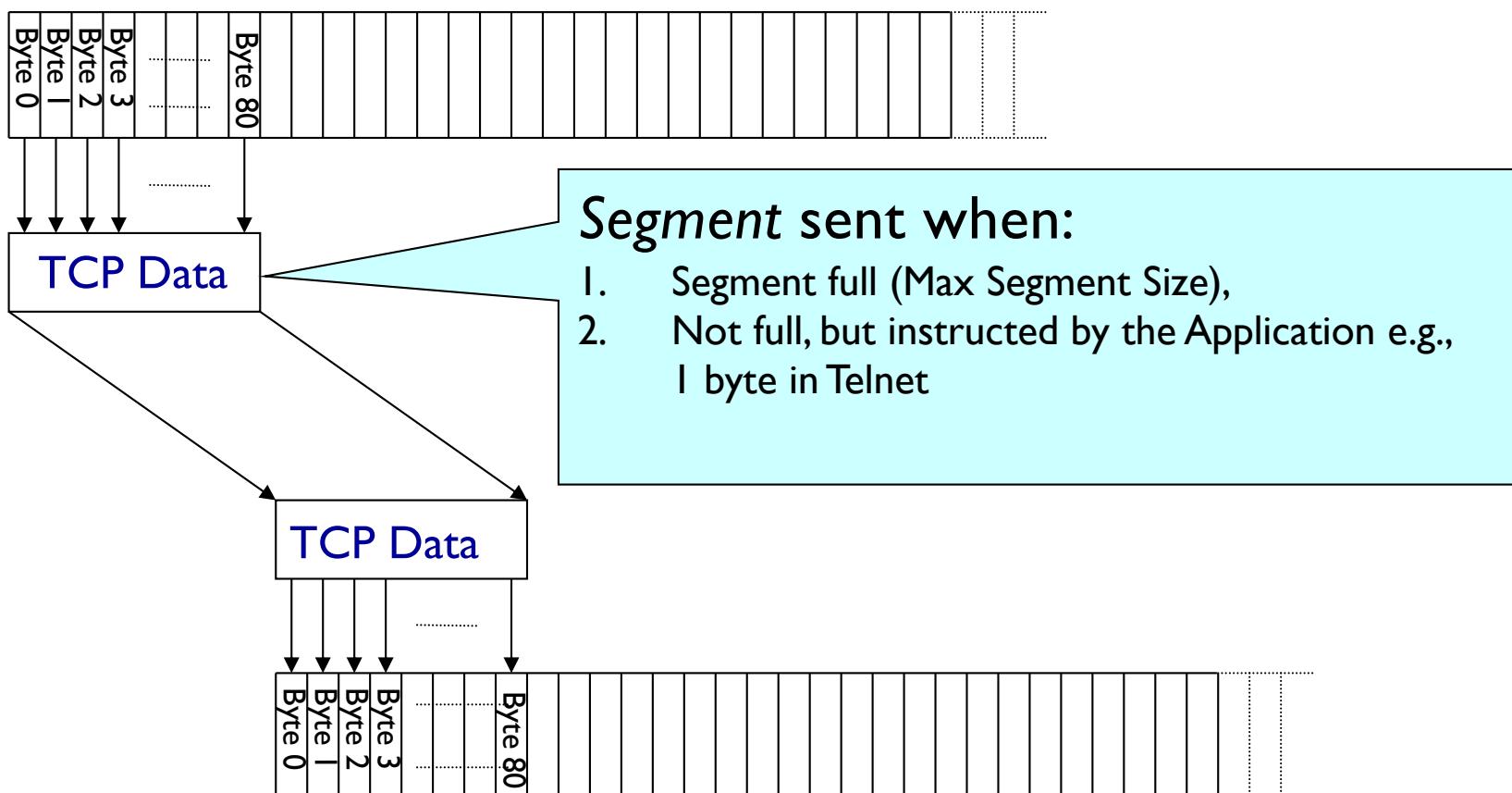
# Application @ Host A



# Application @ Host B

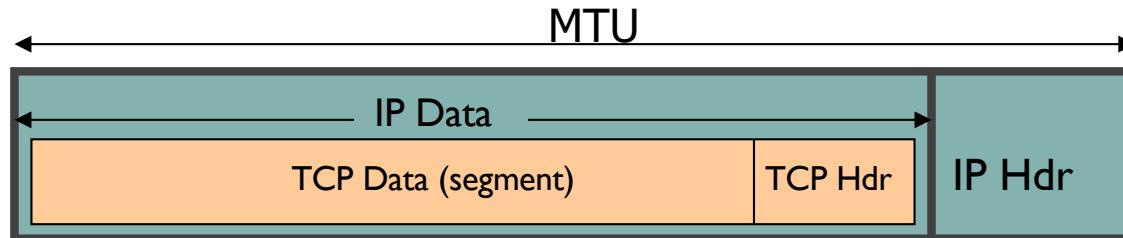
# .. Provided Using TCP “Segments”

Host A



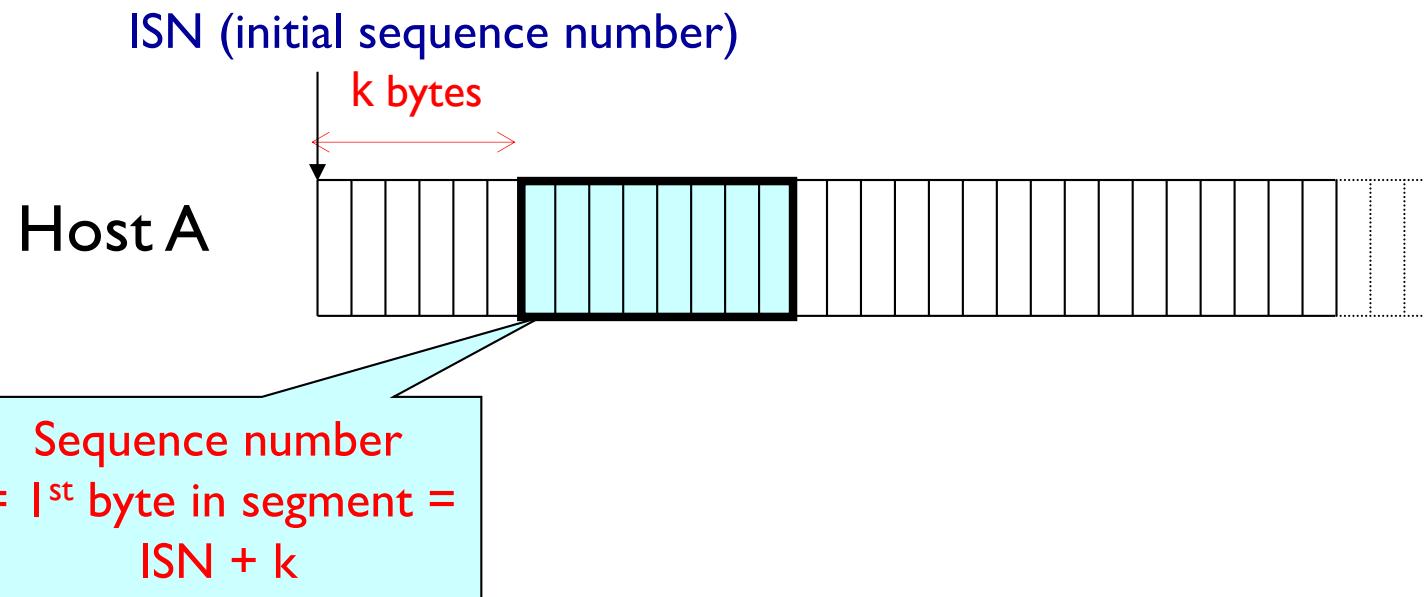
Host B

# TCP Maximum Segment Size



- ❖ IP packet
  - No bigger than Maximum Transmission Unit (**MTU**)
  - E.g., up to 1500 bytes with Ethernet
- ❖ TCP packet
  - IP packet with a TCP header and data inside
  - TCP header  $\geq$  20 bytes long
- ❖ TCP **segment**
  - No more than **Maximum Segment Size (MSS)** bytes
  - E.g., up to 1460 consecutive bytes from the stream
  - $MSS = MTU - 20 \text{ (min IP header)} - 20 \text{ ( min TCP header )}$

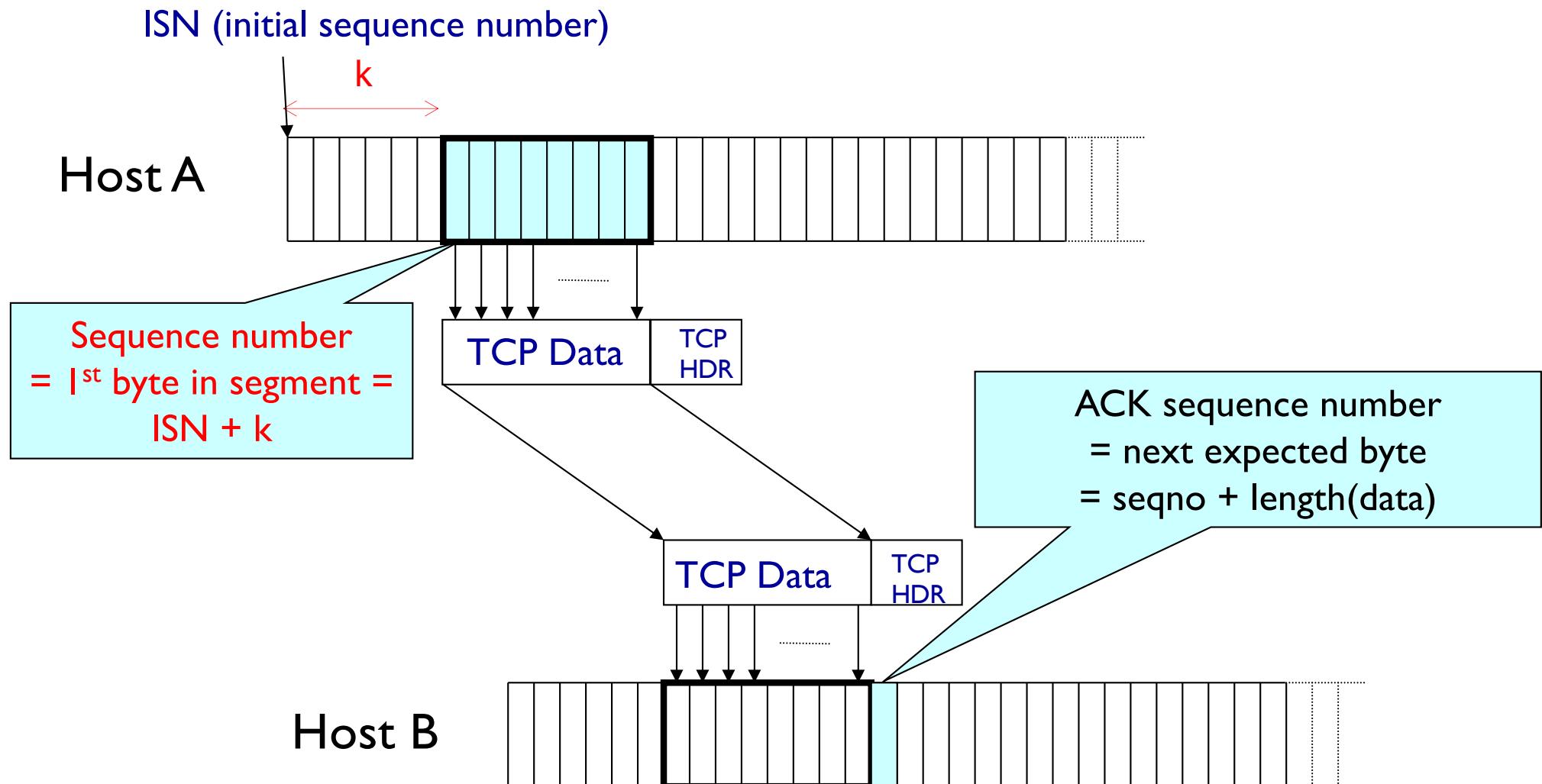
# Sequence Numbers



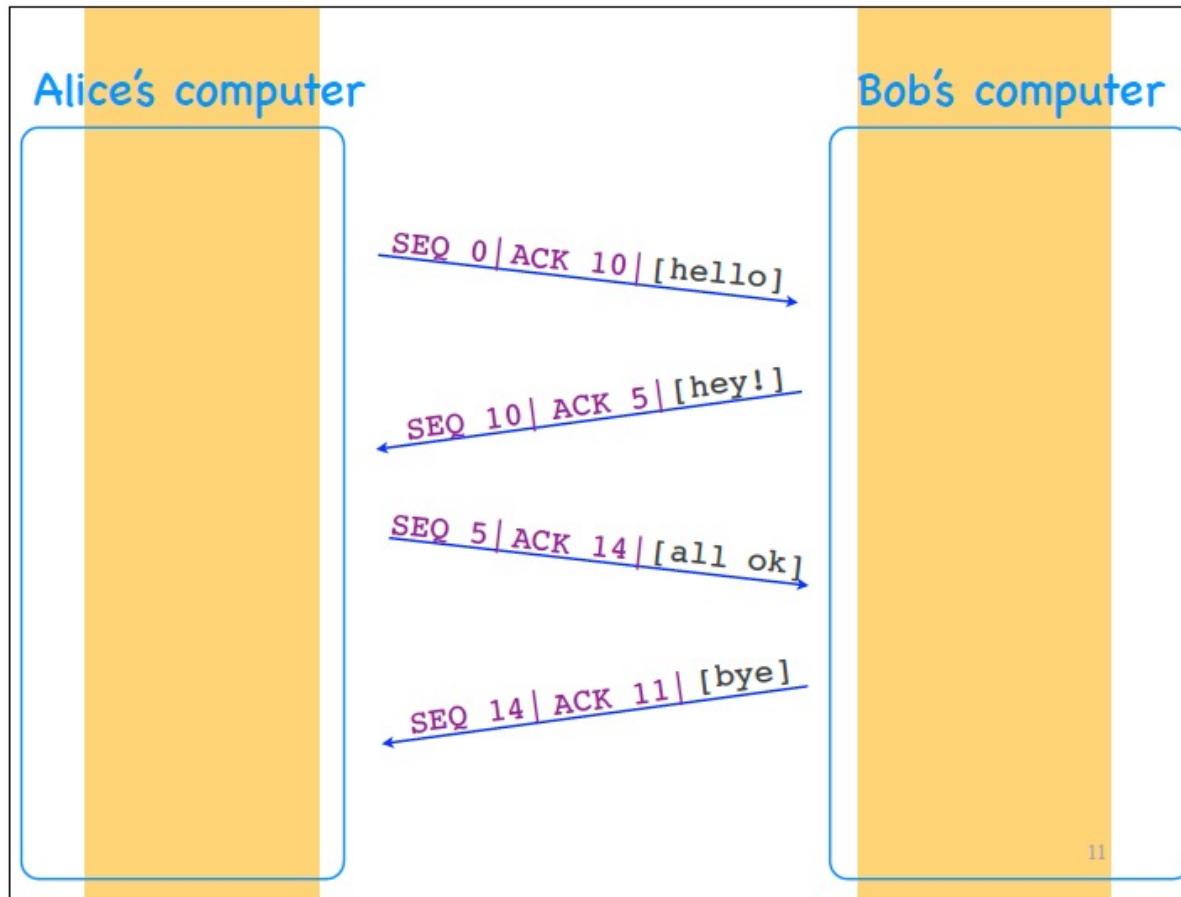
## Sequence numbers:

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

# Sequence & Ack Numbers

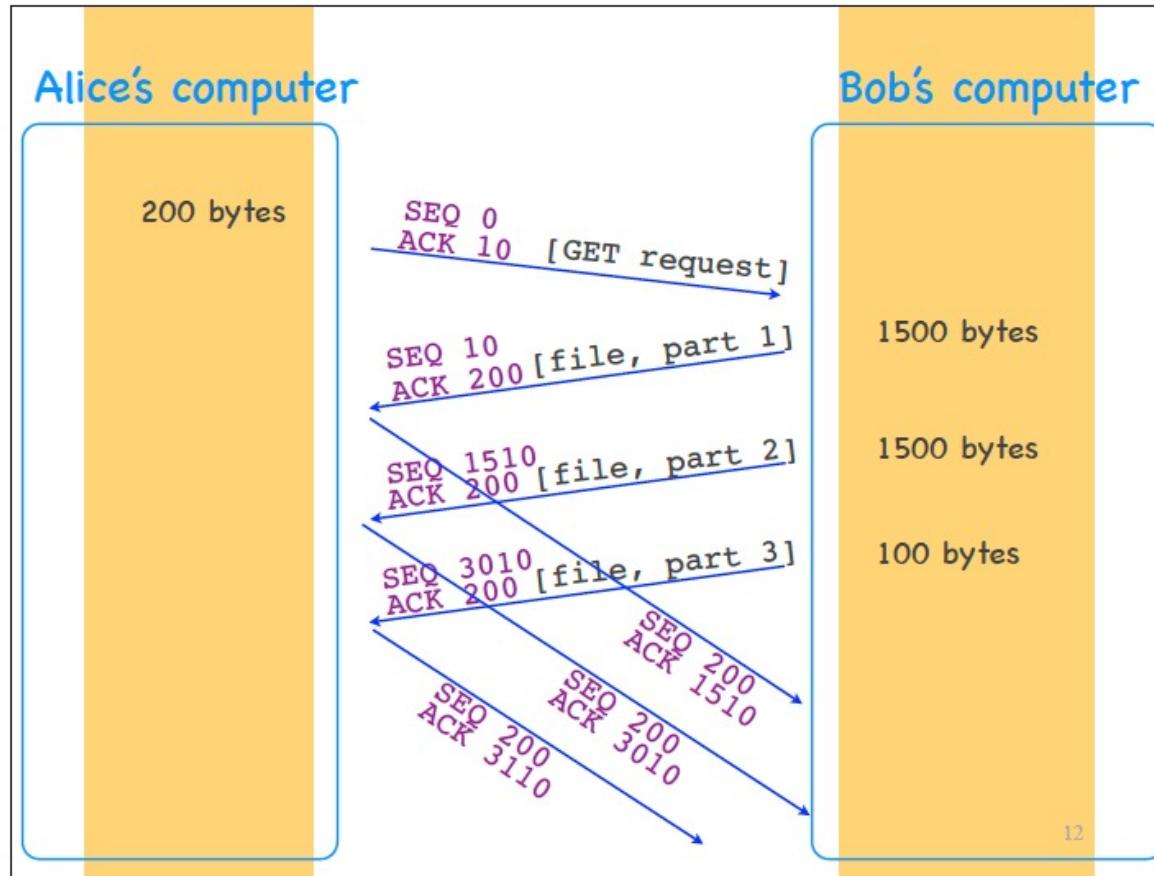


# Example



Note: Connection establishment not shown. Alice's end point selects the initial sequence number as 0 while Bob's end point selects the initial sequence number as 10

# Another Example

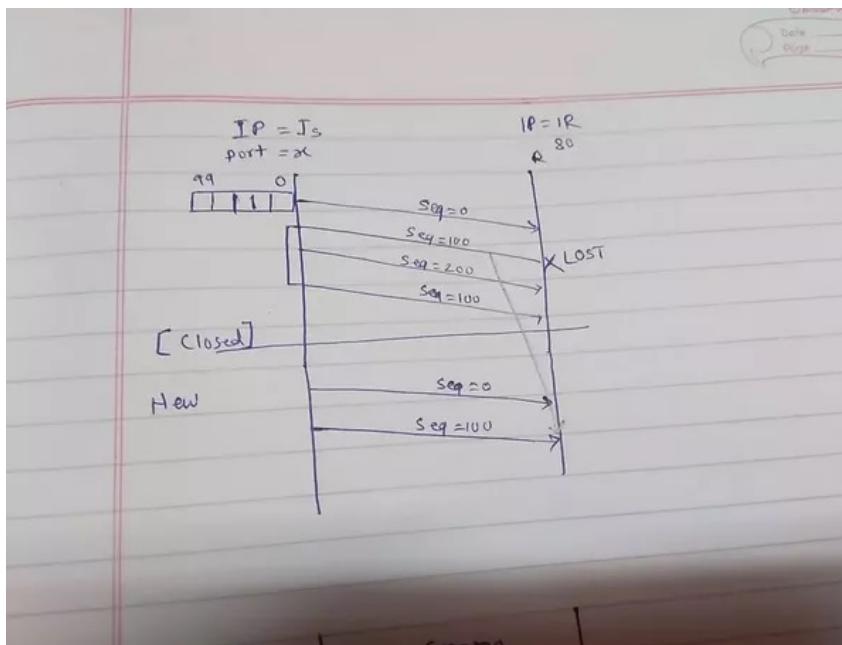


Note: Connection establishment not shown. Alice's end point selects the initial sequence number as 0 while Bob's end point selects the initial sequence number as 10

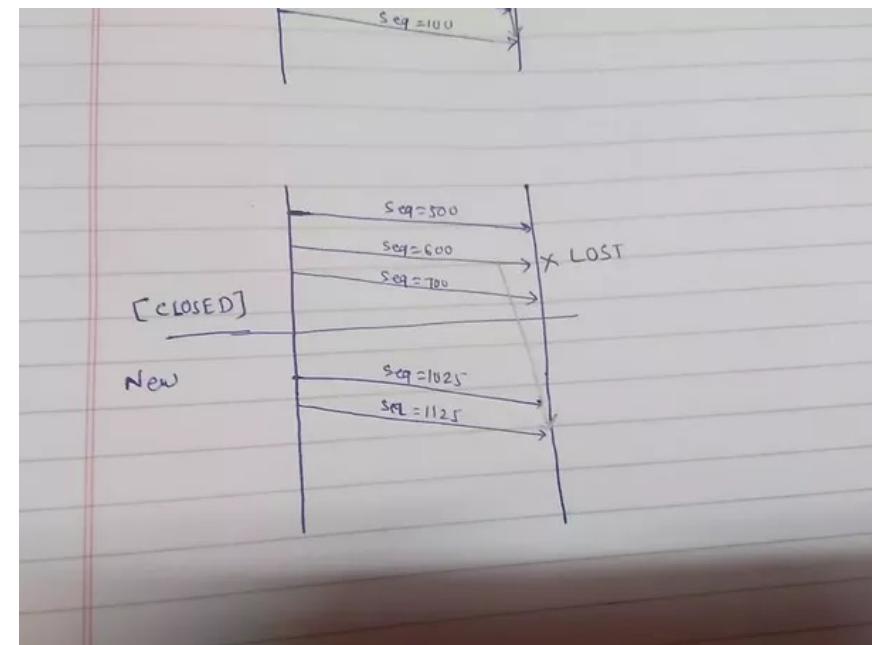
HTTP response split into 3 segments (MSS = 1500 bytes)

# Why choose random ISN?

- ❖ Avoids ambiguity with back-to-back connections between same end-points



(a) When ISN=0



(b) When ISN is random

- ❖ Potential security issue if the ISN is known

# What does TCP do?

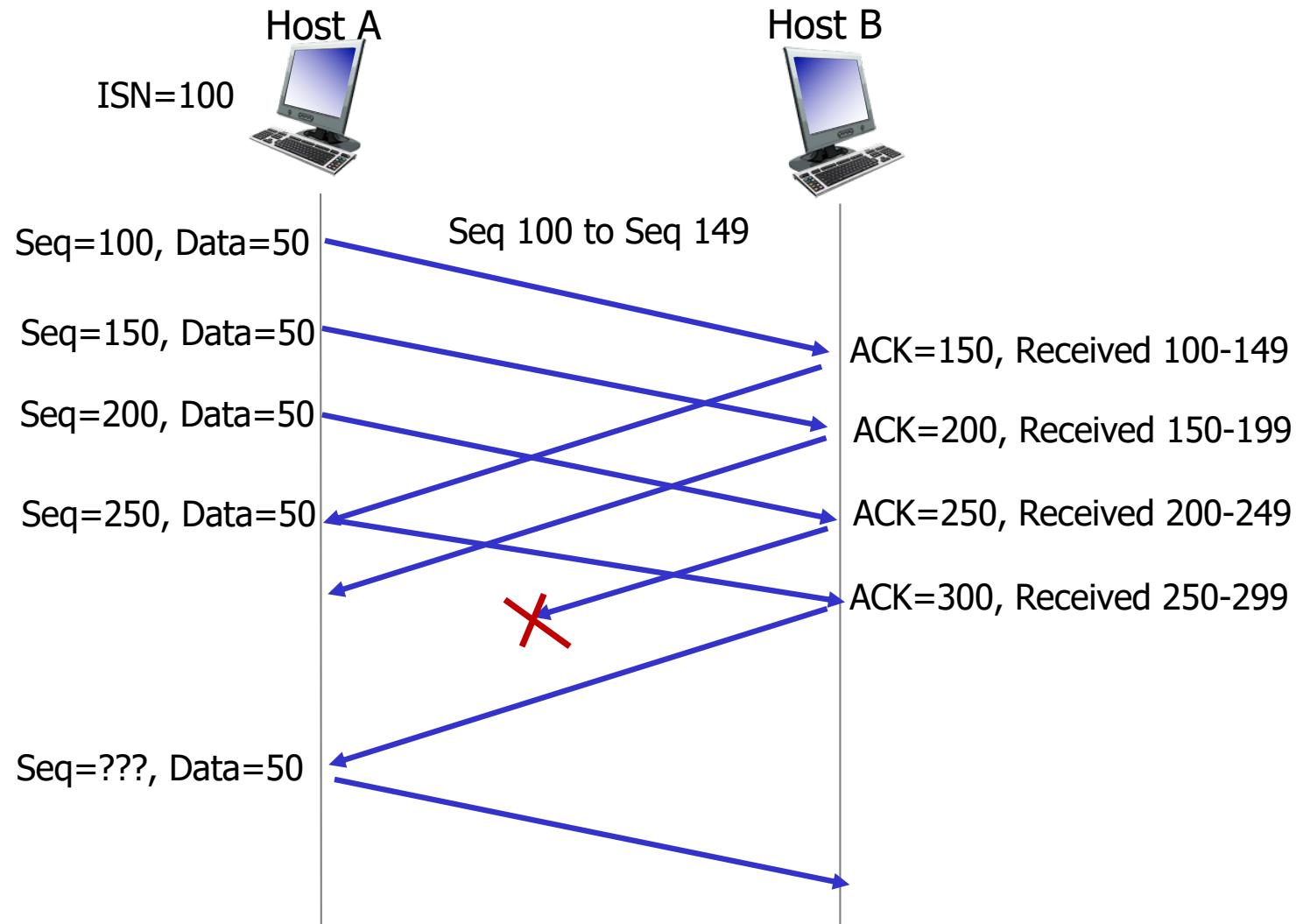
Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)

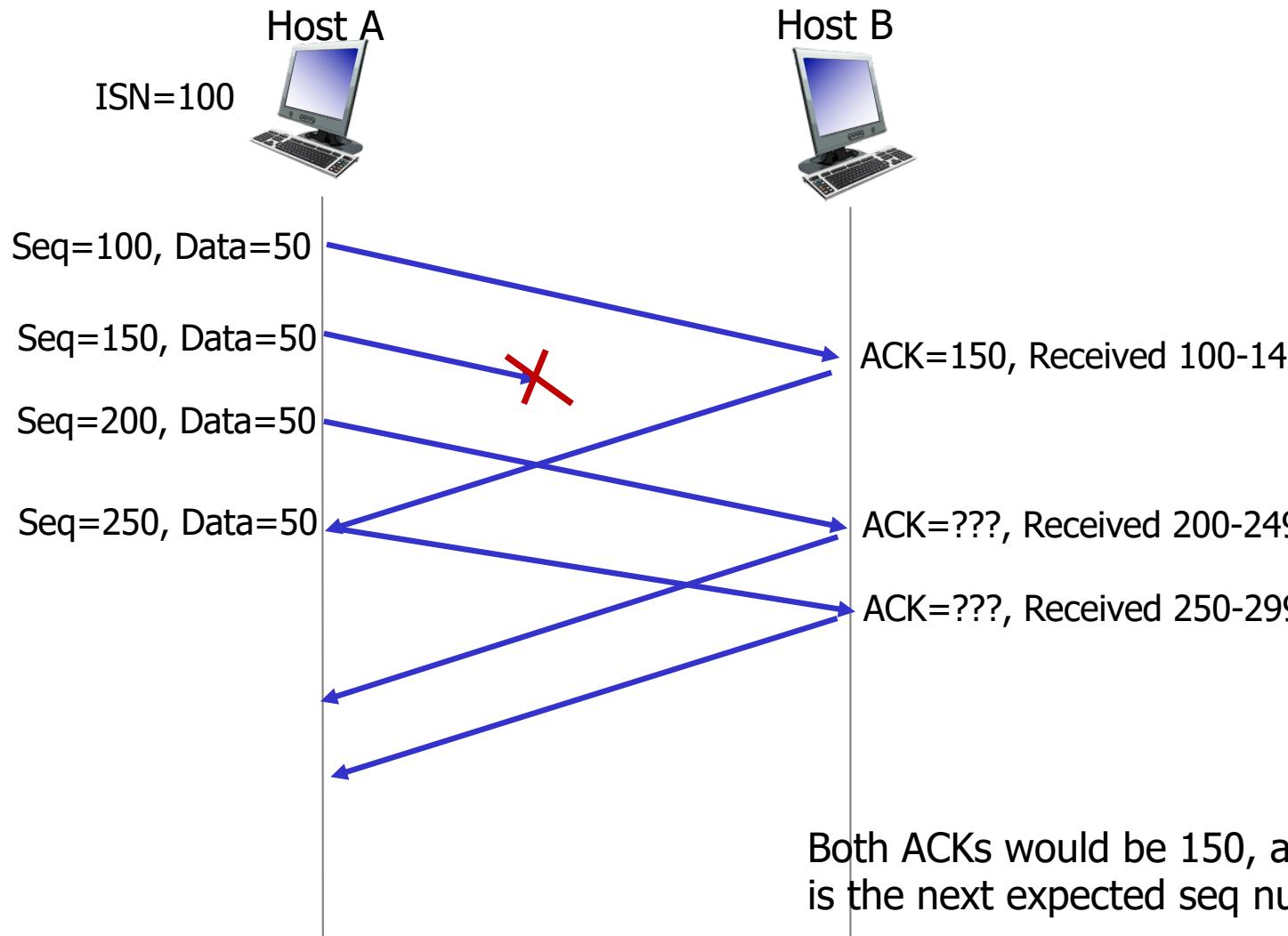
# ACKing and Sequence Numbers

- ❖ Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes [X, X+1, X+2, ..., X+B-1]
- ❖ Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges  $X+B$  (because that is next expected byte)
  - If highest in-order byte received is Y s.t.  $(Y+1) < X$ 
    - ACK acknowledges  $Y+1$
    - Even if this has been ACKed before

# TCP seq. numbers, ACKs



# TCP seq. numbers, ACKs



# Normal Pattern

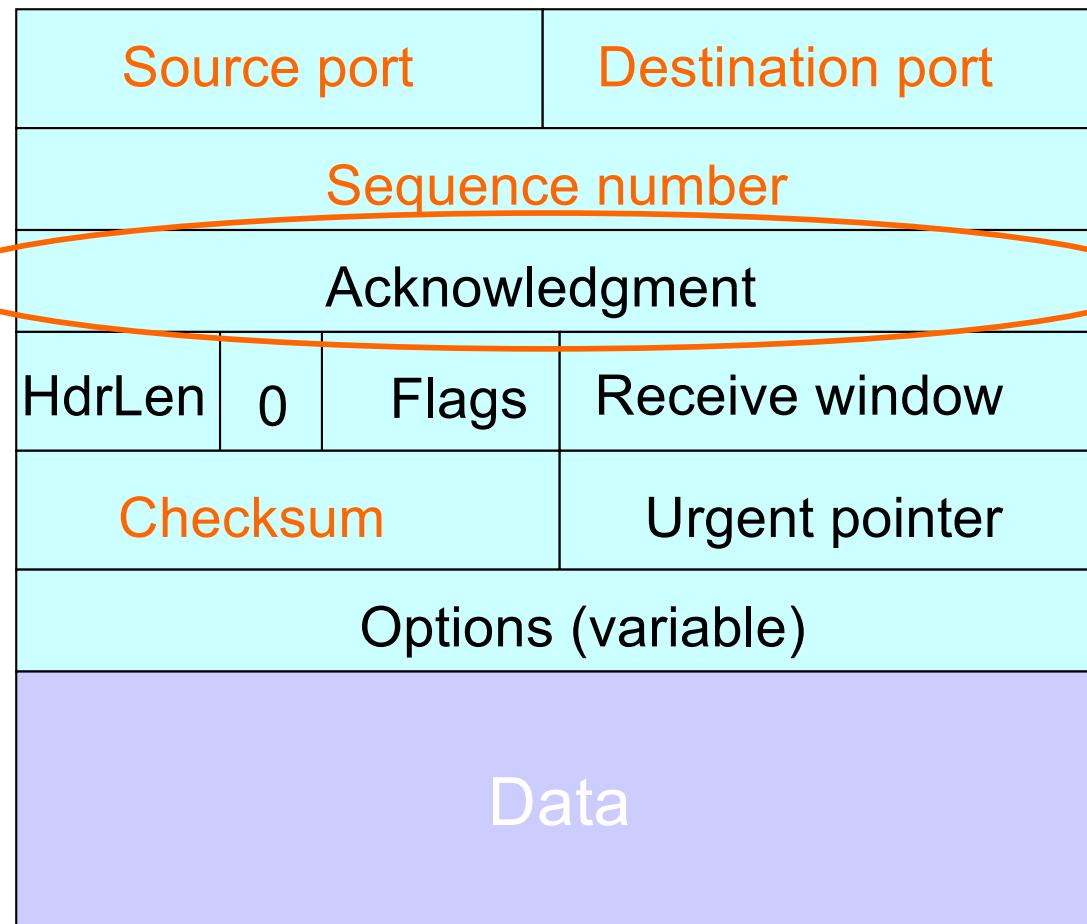
- ❖ Sender: seqno=X, length=B
- ❖ Receiver: ACK=X+B
- ❖ Sender: seqno=X+B, length=B
- ❖ Receiver: ACK=X+2B
- ❖ Sender: seqno=X+2B, length=B
  
- ❖ Seqno of next packet is same as last ACK field

# Packet Loss

- ❖ Sender: seqno=X, length=B
- ❖ Receiver: ACK=X+B
- ❖ Sender: ~~seqno=X+B, length=B~~ LOST
  
- ❖ Sender: seqno=X+2B, length=B
- ❖ Receiver: ACK = X+B

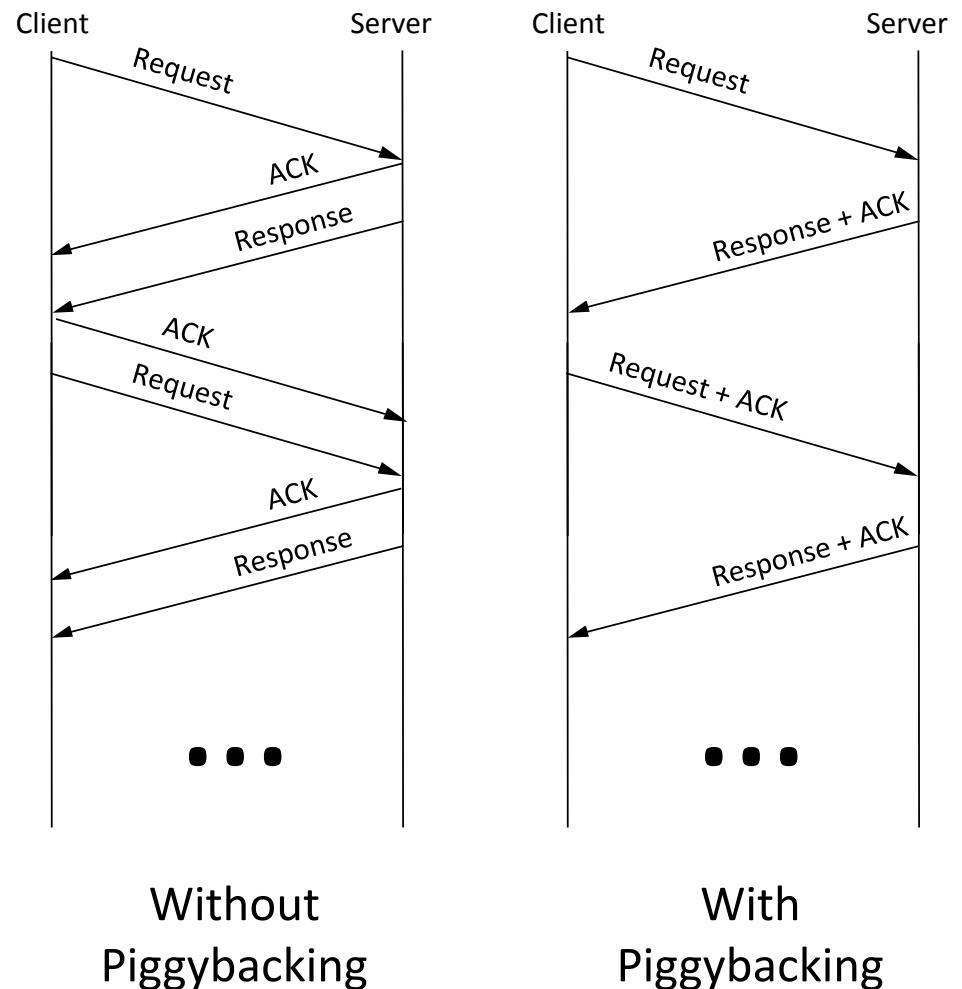
# TCP Header

Acknowledgment gives seqno just beyond highest seqno received **in order**  
(“*What Byte is Next*”)



# Piggybacking

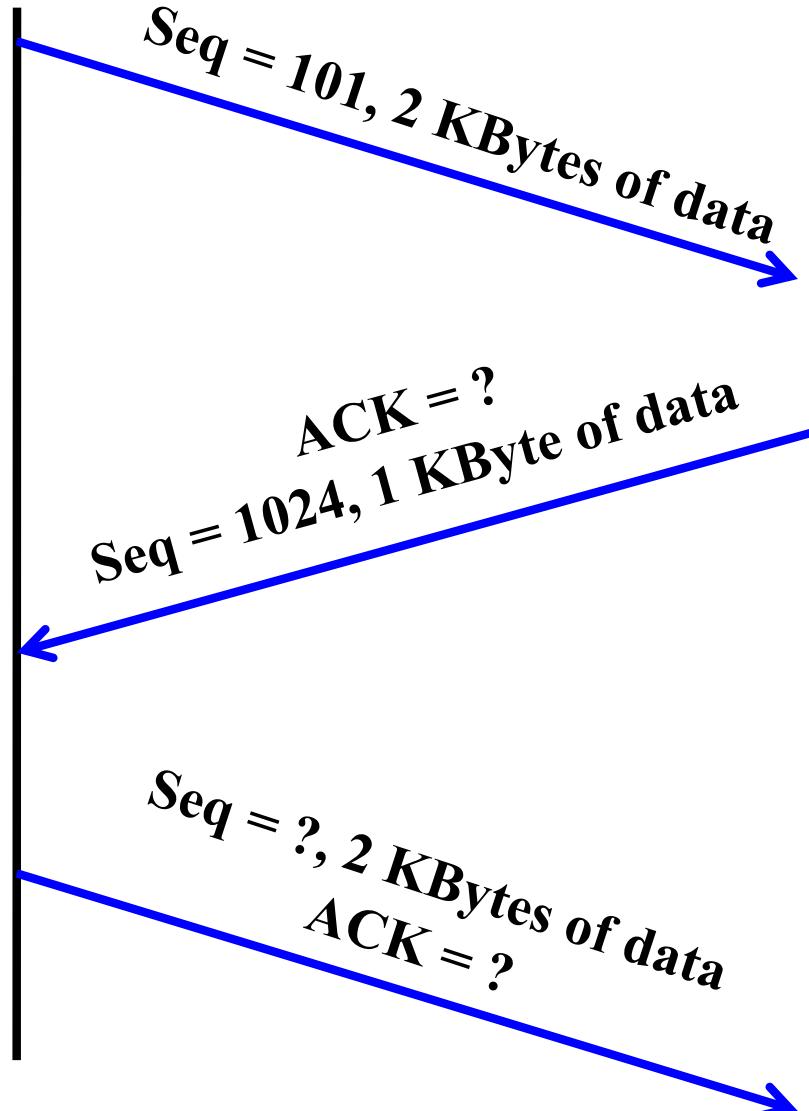
- ❖ So far, we've assumed distinct “sender” and “receiver” roles
- ❖ In reality, usually both sides of a connection send some data



Without  
Piggybacking

With  
Piggybacking

# Quiz



$$\text{ACK} = 101 + 2048 = 2149$$

**Seq = 2149**

$$\text{ACK} = 1024 + 1024 = 2048$$

# What does TCP do?

Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers **can** buffer out-of-sequence packets (like SR)

# Loss with cumulative ACKs

---

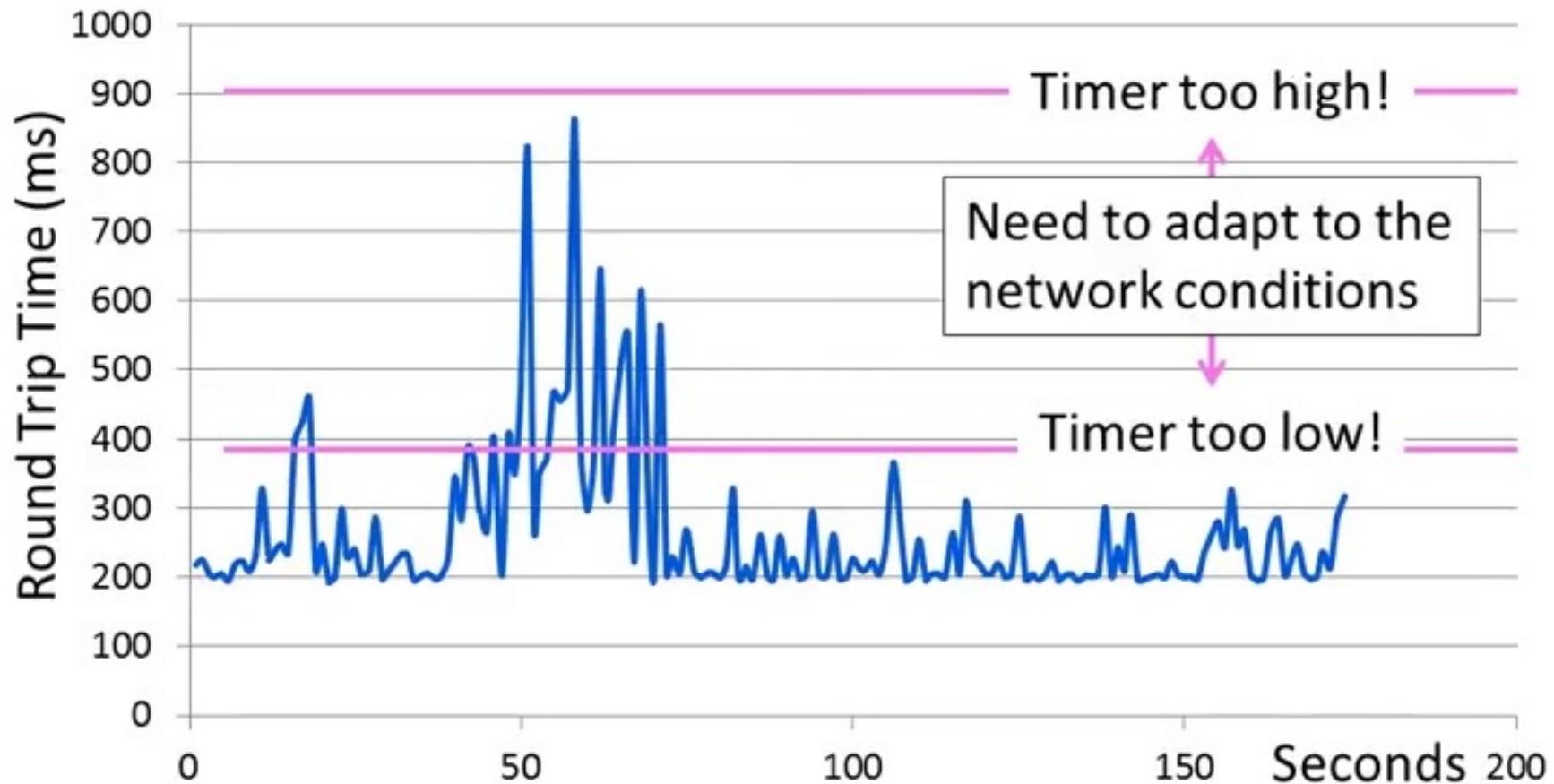
- ❖ Sender sends packets with 100Bytes and sequence numbers:
  - 100, 200, 300, 400, 500, 600, 700, 800, 900, ...
- ❖ Assume the fifth packet (seq. no. 500) is lost, but no others
- ❖ Stream of ACKs will be:
  - 200, 300, 400, 500, 500, 500, ...

# What does TCP do?

Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers do not drop out-of-sequence packets (like SR)
- ❖ Sender maintains a single retransmission timer (like GBN) and retransmits on timeout (*how much?*)

# TCP round trip time, timeout



# TCP round trip time, timeout

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

- ❖ longer than RTT
  - but RTT varies
- ❖ *too short*: premature timeout, unnecessary retransmissions
- ❖ *too long*: slow reaction to segment loss and connection has lower throughput

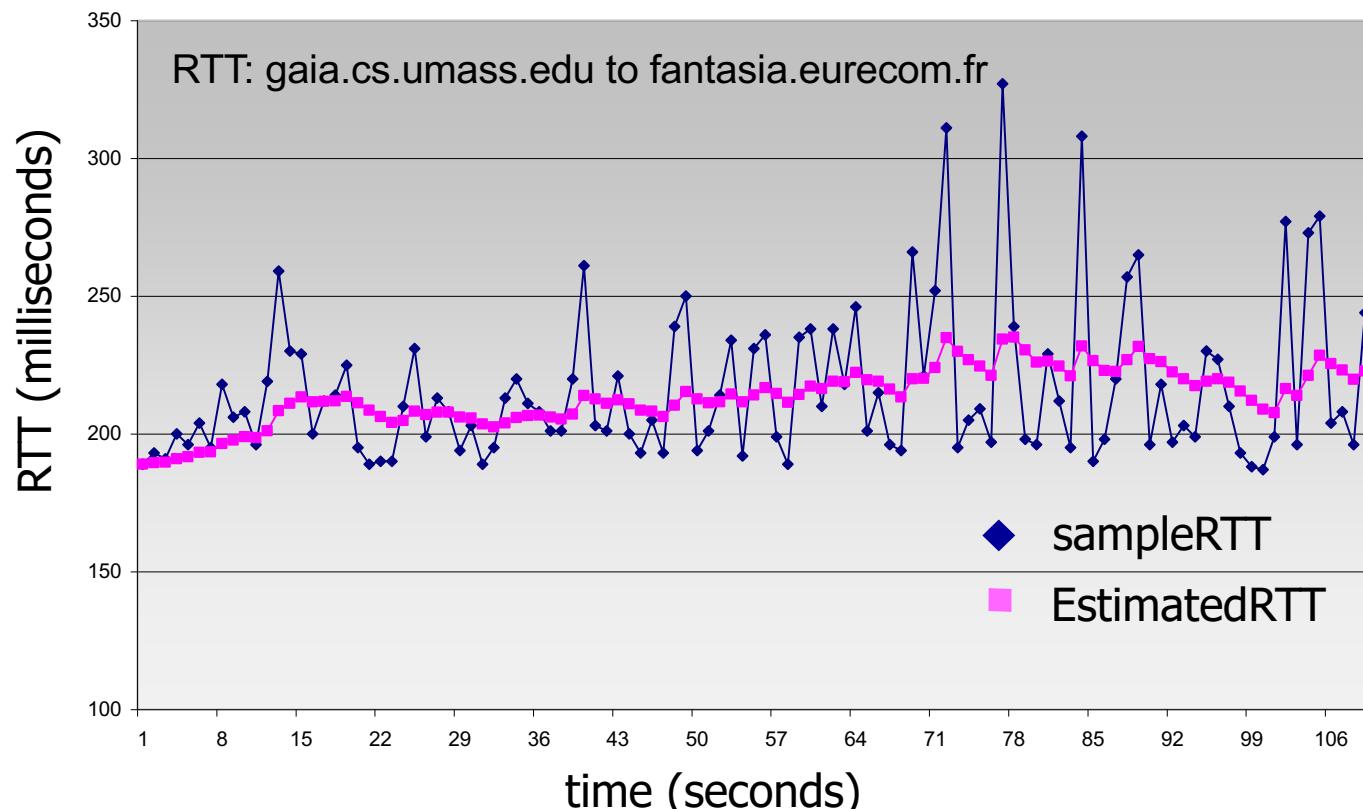
**Q:** how to estimate RTT?

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

# TCP round trip time, timeout

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

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



# TCP round trip time, timeout

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

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

(typically,  $\beta = 0.25$ )

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



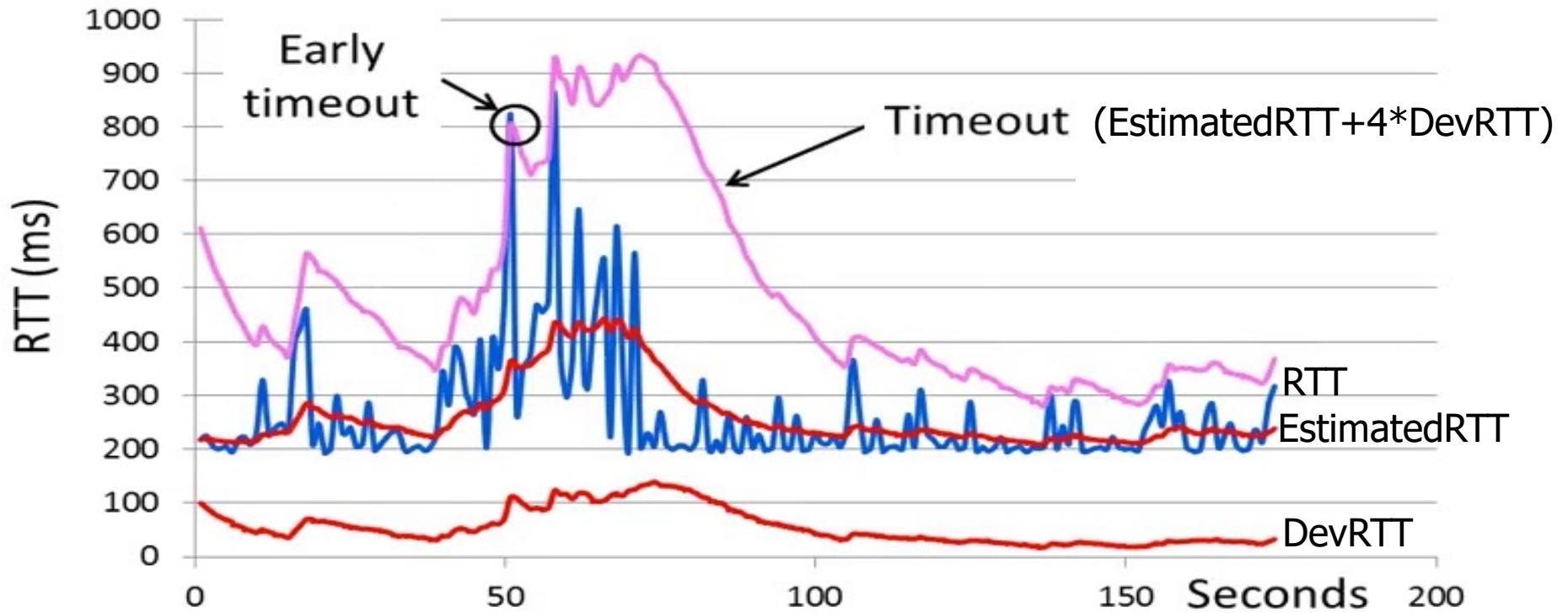
↑  
estimated RTT

↑  
“safety margin”

Practice Problem:

[http://wps.pearsoned.com/ecs\\_kurose\\_compnetw\\_6/216/55463/14198700.cw/index.html](http://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198700.cw/index.html)

# TCP round trip time, timeout



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



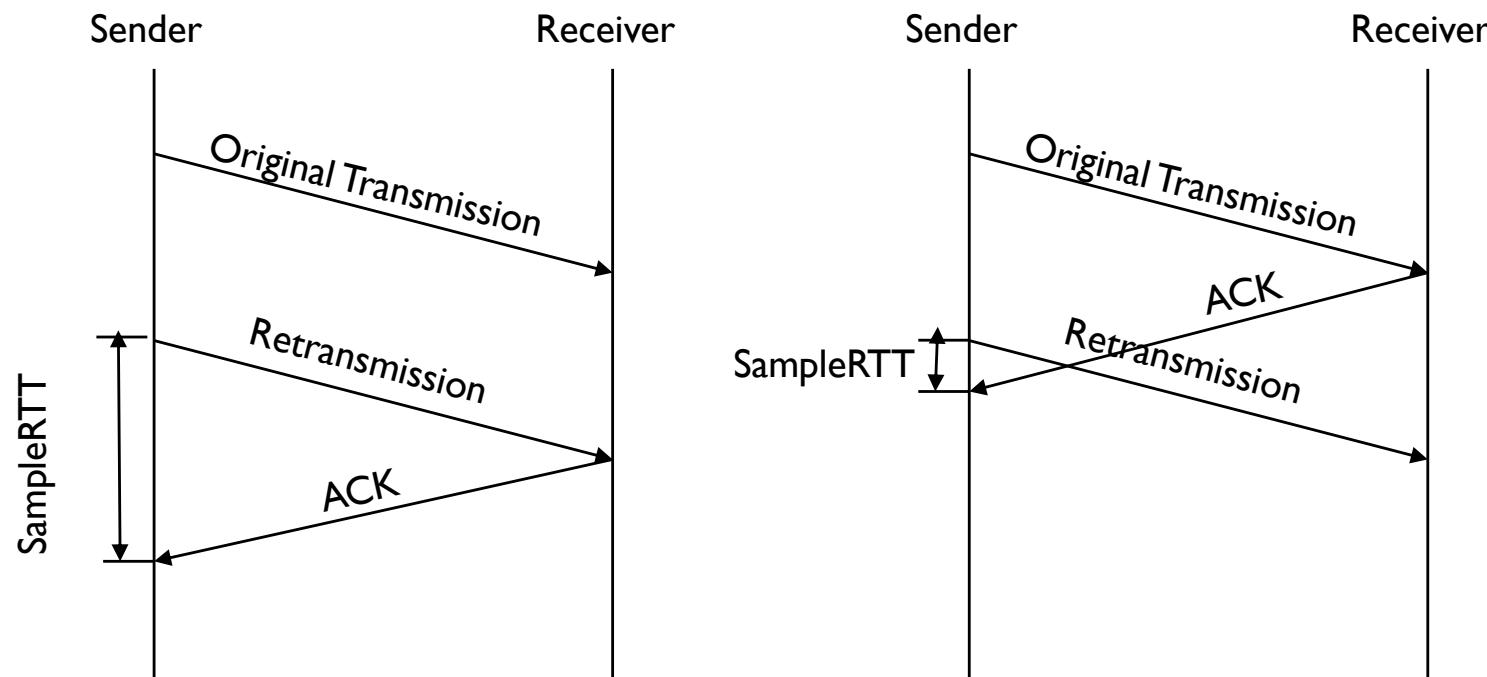
↑  
estimated RTT

↑  
“safety margin”

# Why exclude retransmissions in RTT computation?

---

- ❖ How do we differentiate between the real ACK, and ACK of the retransmitted packet?



# TCP sender events:

PUTTING IT  
TOGETHER

## *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

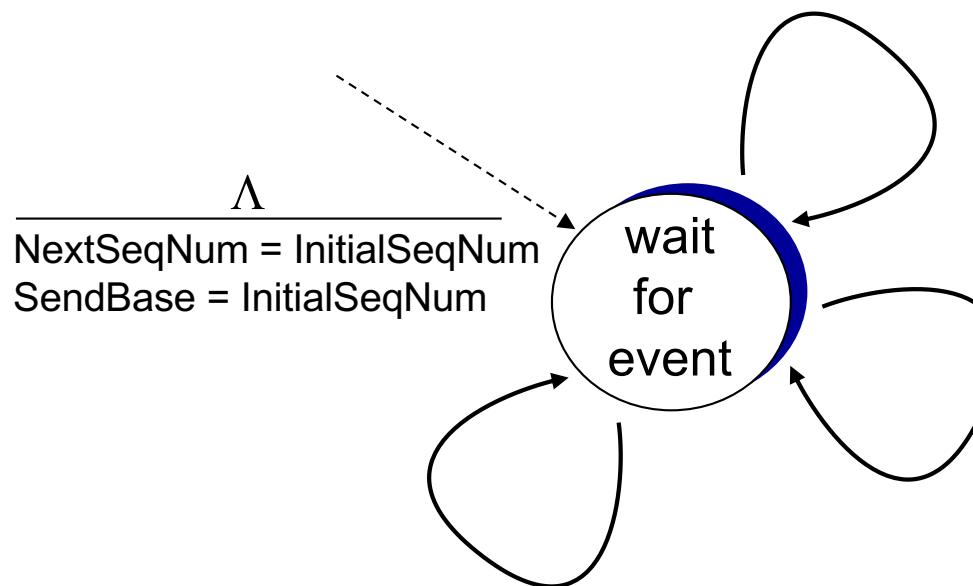
- ❖ restart timer

## *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)

PUTTING IT TOGETHER



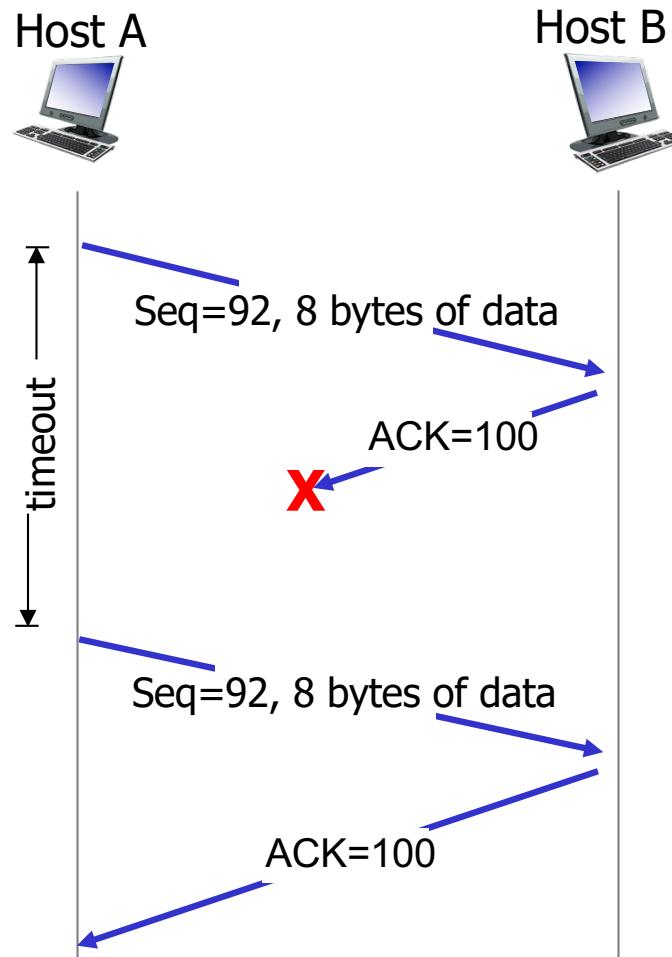
data received from application above  
create segment, seq. #: NextSeqNum  
pass segment to IP (i.e., “send”)  
 $\text{NextSeqNum} = \text{NextSeqNum} + \text{length(data)}$   
if (timer currently not running)  
start timer

---

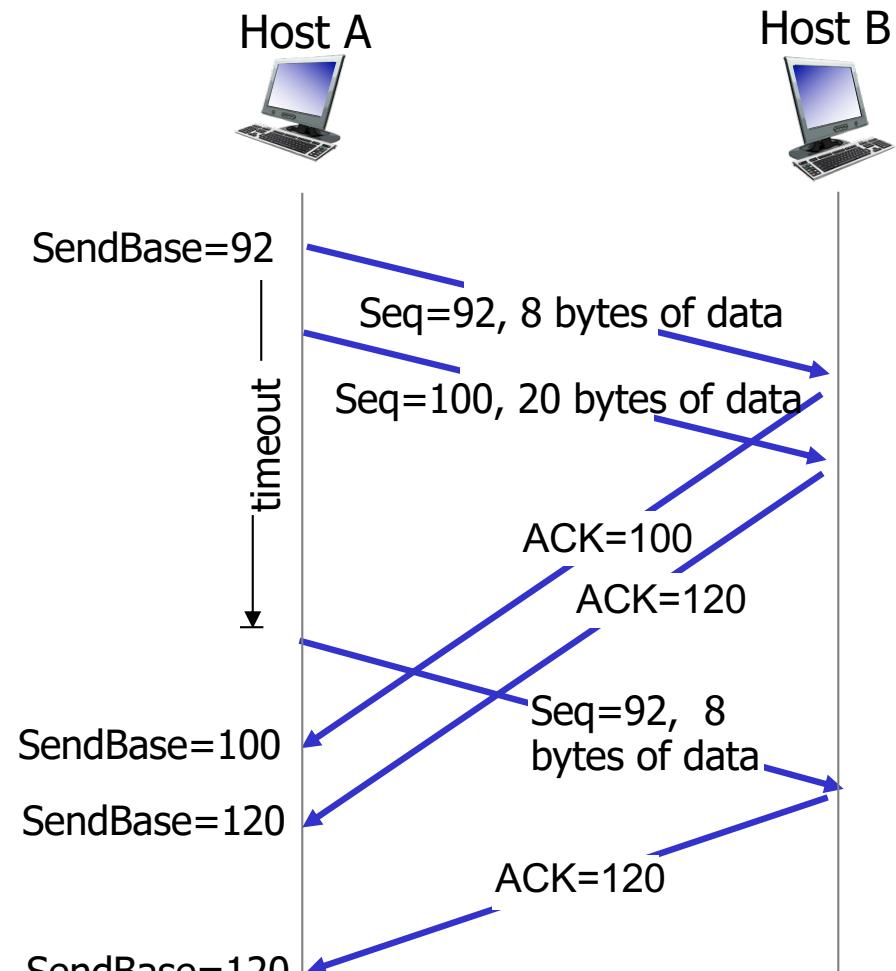
timeout  
retransmit not-yet-acked segment  
with smallest seq. #  
start timer

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

# TCP: retransmission scenarios

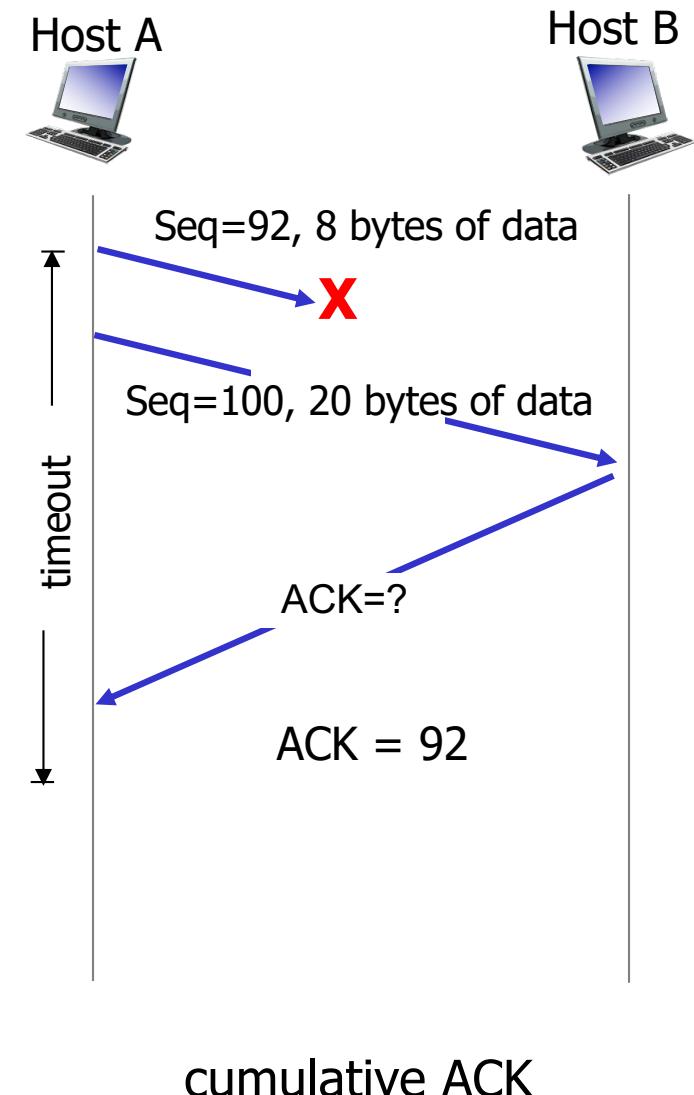
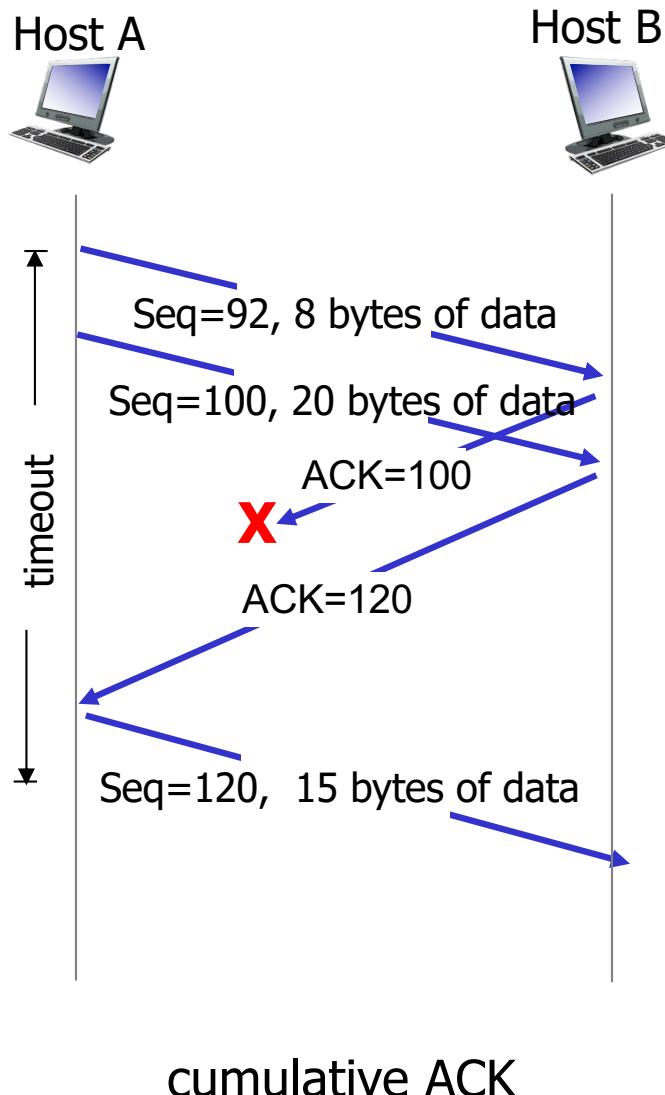


lost ACK scenario



premature timeout

# TCP: retransmission scenarios



# TCP ACK generation [RFC 1122, RFC 2581]

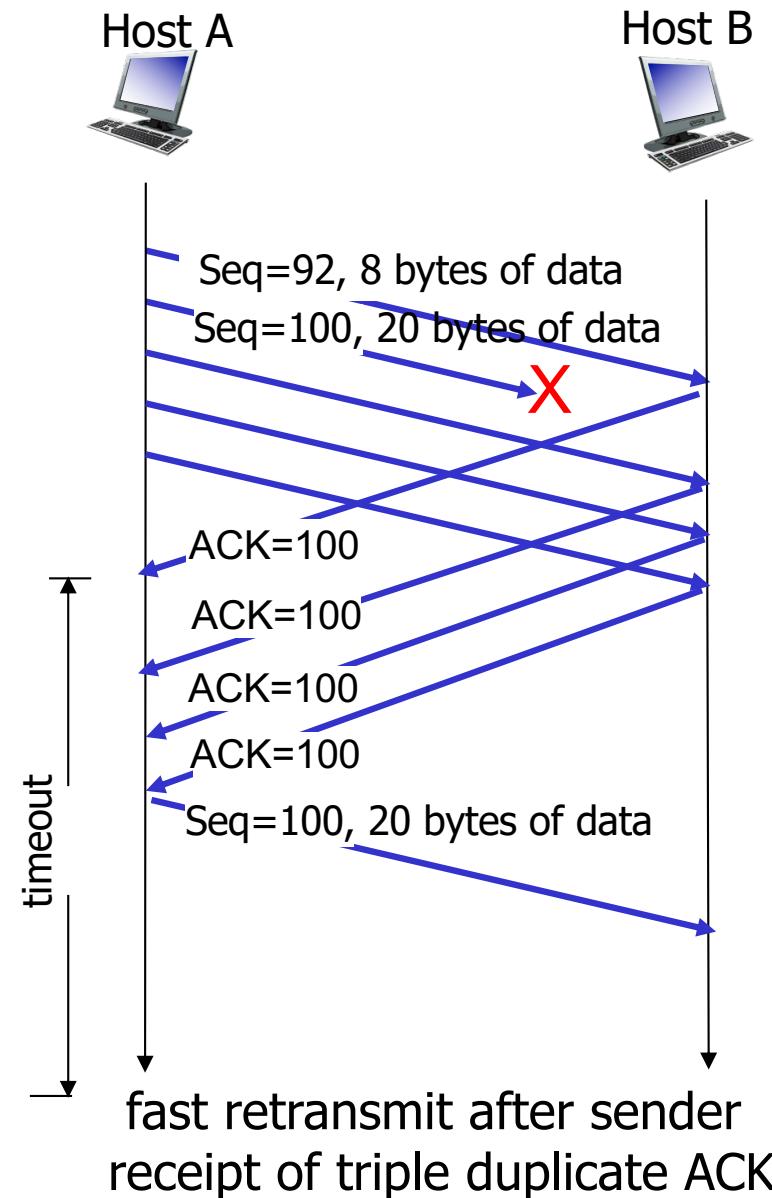
<i>event at receiver</i>	<i>TCP receiver action</i>
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	<b>delayed ACK.</b> Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single <b>cumulative ACK</b> , ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send <b>duplicate ACK</b> , 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

# What does TCP do?

Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers may not drop out-of-sequence packets (like SR)
- ❖ Sender maintains a single retransmission timer (like GBN) and retransmits on timeout
- ❖ Introduces **fast retransmit**: optimisation that uses duplicate ACKs to trigger early retransmission

# TCP fast retransmit



# TCP fast retransmit

- ❖ time-out period often relatively long:
  - long delay before resending lost packet
- ❖ “Duplicate ACKs” are a sign of an isolated loss
  - The lack of ACK progress means that packet hasn’t been delivered
  - Stream of ACKs means some packets are being delivered
  - Could trigger resend on receiving “ $k$ ” duplicate ACKs (TCP uses  $k = 3$ )

## *TCP fast retransmit*

if sender receives 3 duplicate ACKs for same data (“triple duplicate ACKs”), resend unacked segment with smallest seq #

- likely that unacked segment is lost, so don’t wait for timeout

# What does TCP do?

Most of our previous ideas, but some key differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers do not drop out-of-sequence packets (like SR)
- ❖ Sender maintains a single retransmission timer (like GBN) and retransmits on timeout
- ❖ Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission

# Quiz: TCP Sequence Numbers?



A TCP Sender is just about to send a segment of size 100 bytes with sequence number 1234 and ack number 436 in the TCP header. What is the highest sequence number up to (and including) which this sender has received all bytes from the receiver?

- A. 1233
- B. 436
- C. 435
- D. 1334
- E. 536

**ANSWER: C**

# Quiz: TCP Sequence Numbers?



A TCP Sender is just about to send a segment of size 100 bytes with sequence number 1234 and ack number 436 in the TCP header. Is it possible that the receiver has received byte number 1335?

- A. Yes
- B. No

**ANSWER: A**

# Quiz: TCP Sequence Numbers?



The following statement is true about the TCP sliding window protocol for implementing reliable data transfer

- A. It exclusively uses the ideas of Go-Back-N
- B. It exclusively uses the ideas of Selective Repeat
- C. It uses a combination of ideas of Go-Back-N and Selective-Repeat
- D. It uses none of the ideas of Go-Back-N and Selective-Repeat

**ANSWER: C**

# Transport Layer 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

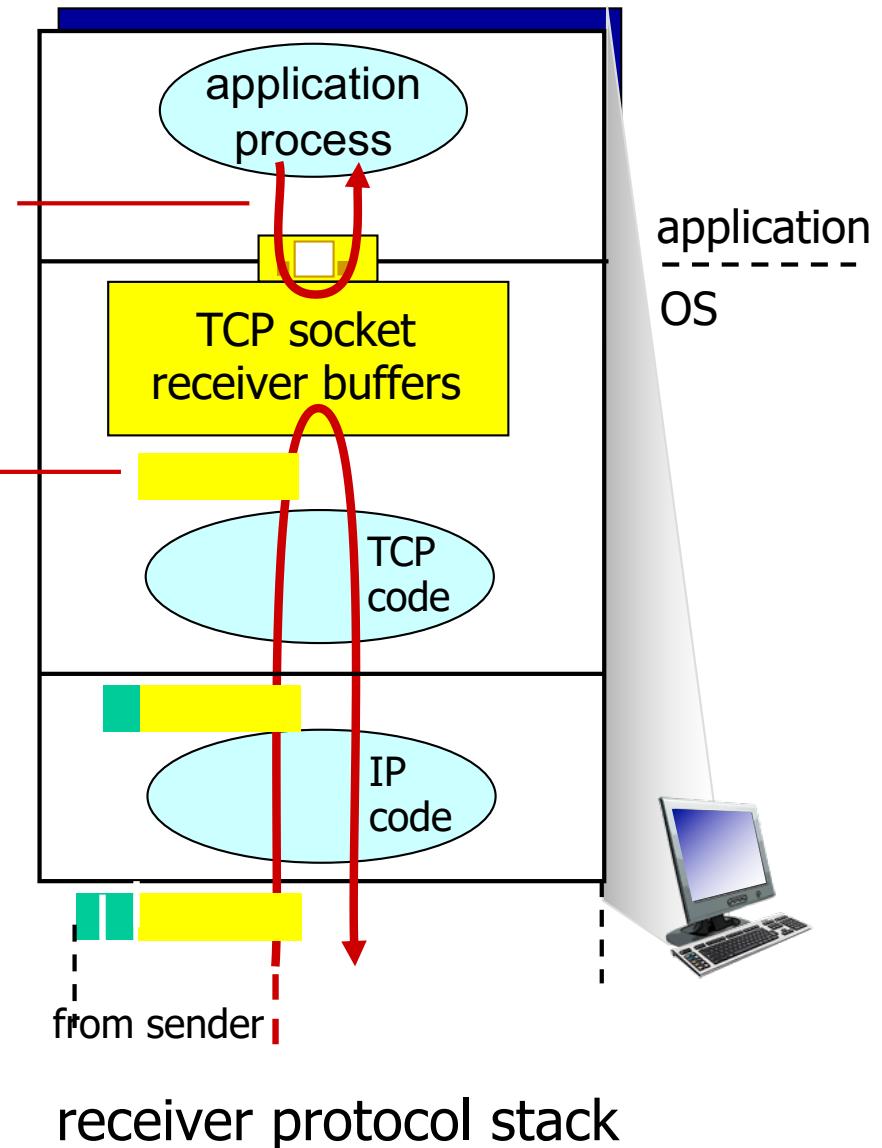
3.7 TCP congestion control

# TCP flow control

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

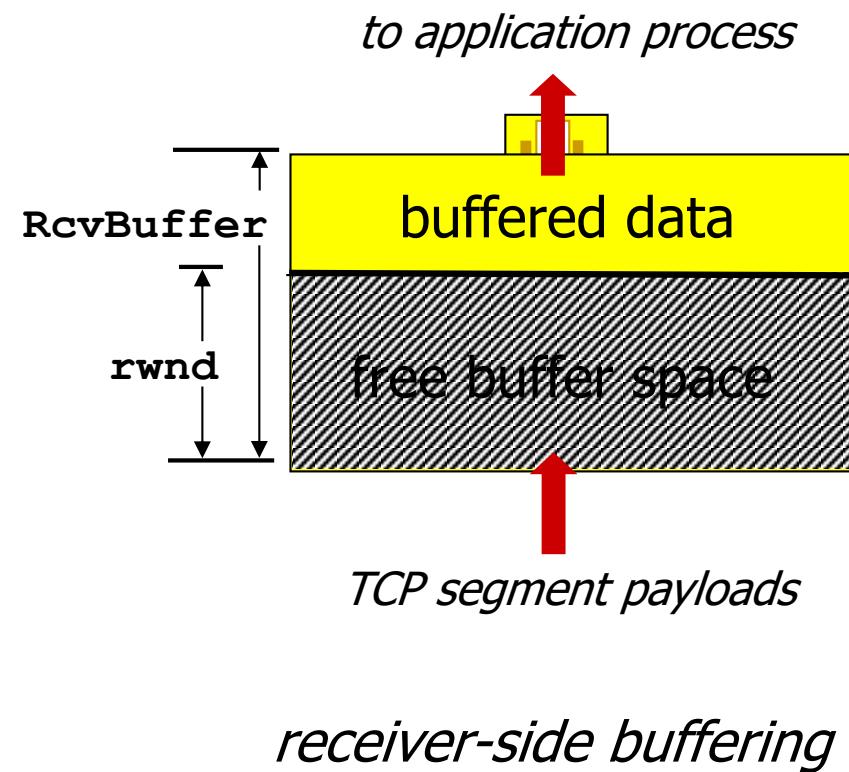
... slower than TCP  
receiver is delivering  
(sender is sending)

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

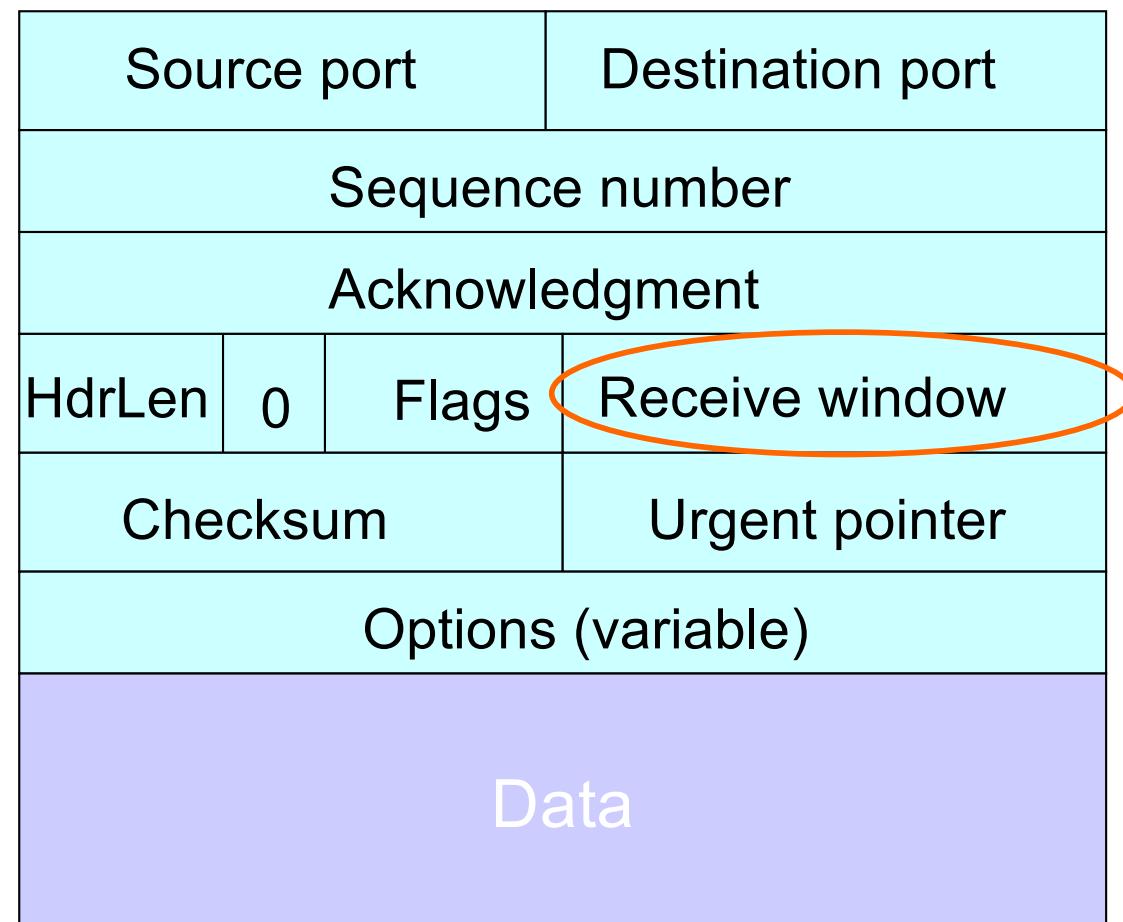


# TCP flow control

- ❖ 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
- ❖ guarantees receive buffer will not overflow



# TCP Header



# TCP flow control

- ❖ What if **rwnd** = 0?
  - Sender would stop sending data
  - Eventually the receive buffer would have space when the application process reads some bytes
  - But how does the receiver advertise the new **rwnd** to the sender?
- ❖ Sender keeps sending TCP segments with one data byte to the receiver
- ❖ These segments are dropped but acknowledged by the receiver with a zero-window size
- ❖ Eventually when the buffer empties, non-zero window is advertised

# Transport Layer 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

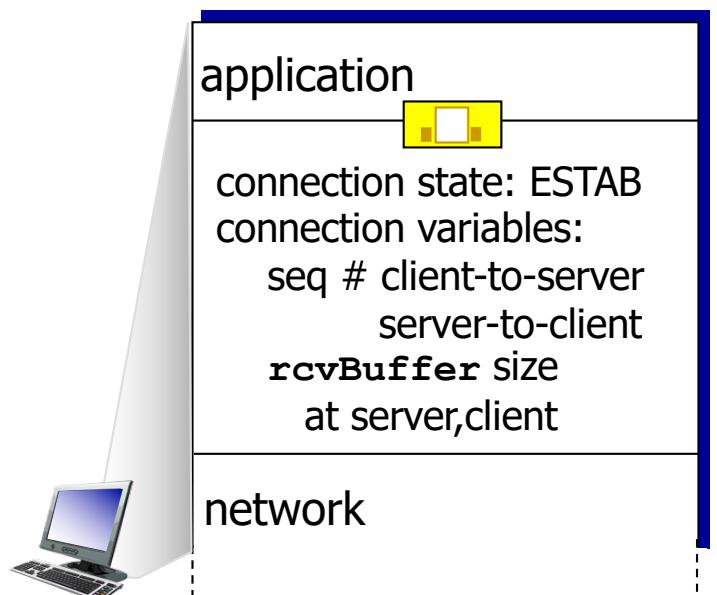
3.6 principles of congestion control

3.7 TCP congestion control

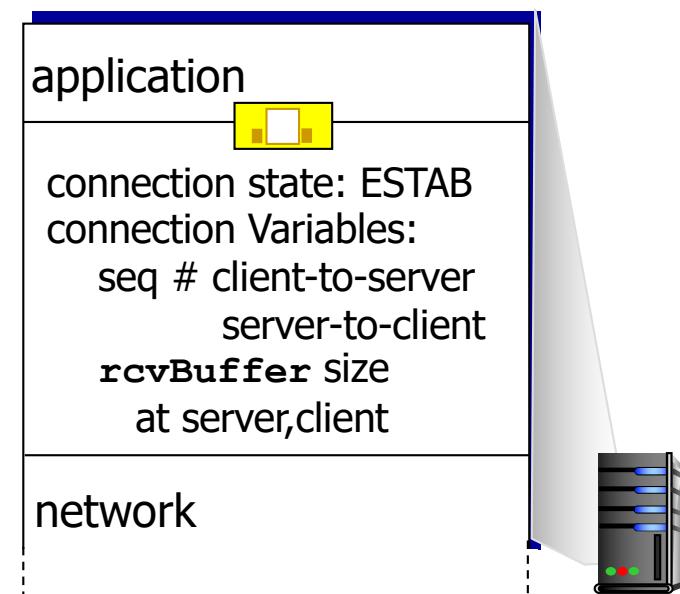
# Connection Management

before exchanging data, sender/receiver “handshake”:

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

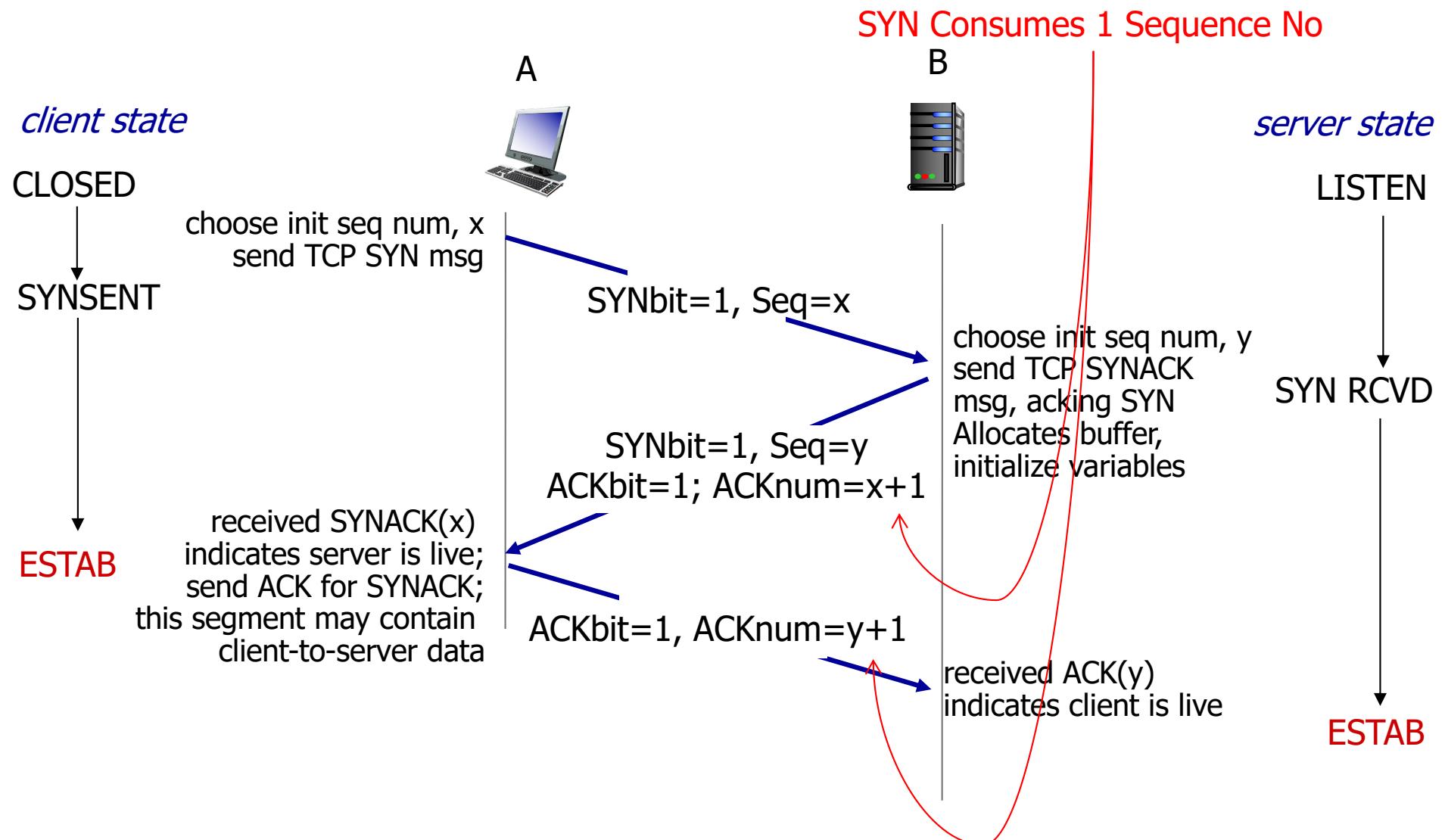


```
Socket clientSocket =  
    newSocket("hostname", "port  
    number");
```

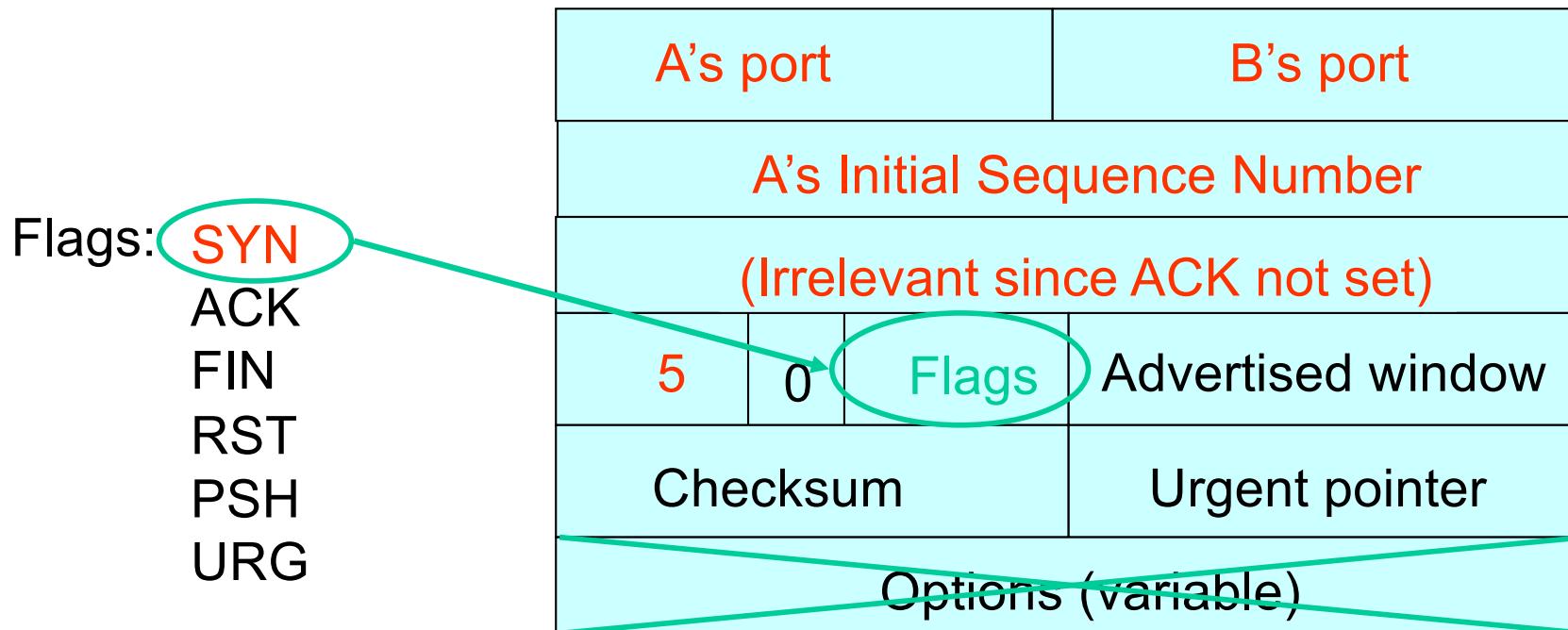


```
Socket connectionSocket =  
    welcomeSocket.accept();
```

# TCP 3-way handshake

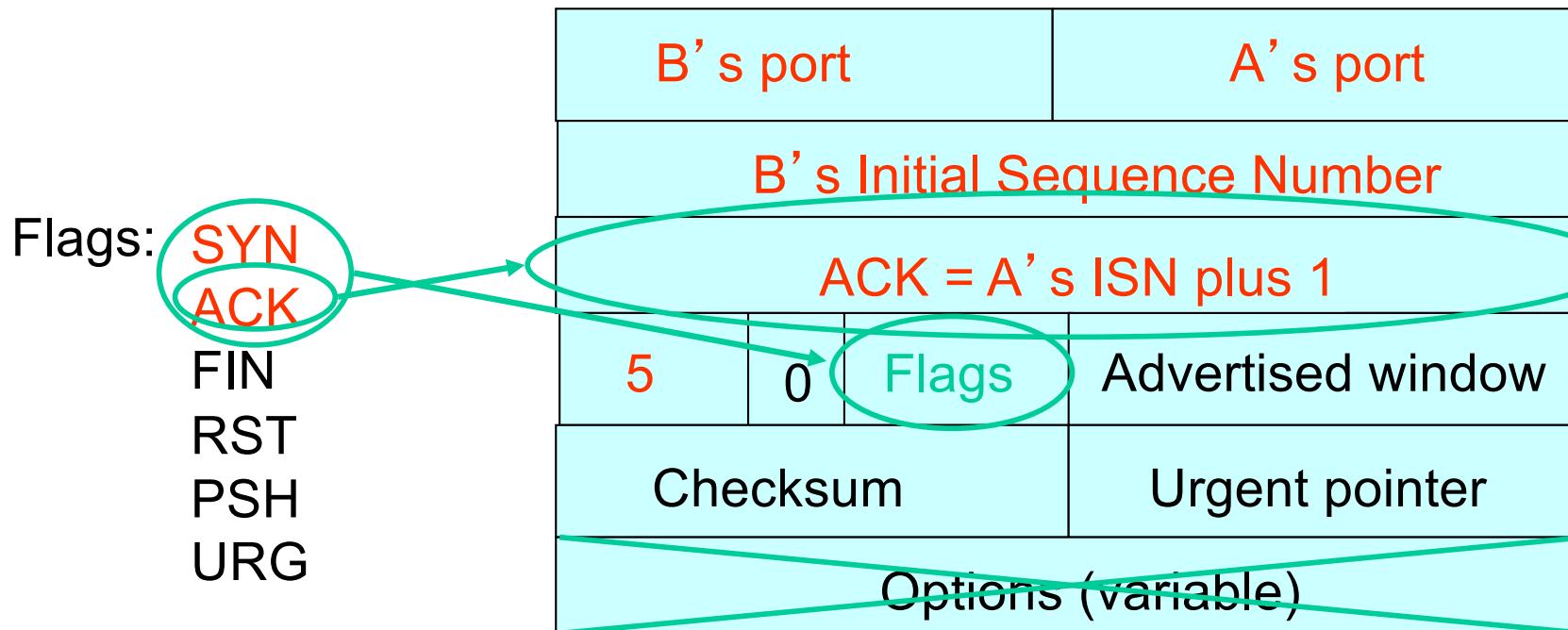


# Step 1: A's Initial SYN Packet



A tells B it wants to open a connection...

# Step 2: B's SYN-ACK Packet

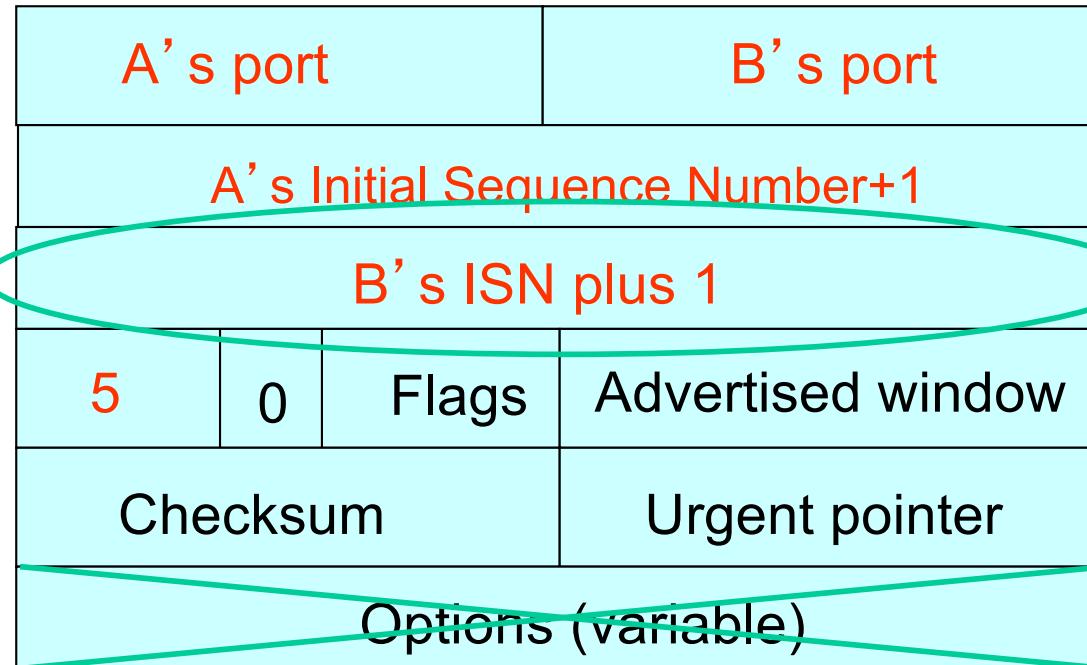


**B tells A it accepts, and is ready to hear the next byte...**

**... upon receiving this packet, A can start sending data**

# Step 3: A's ACK of the SYN-ACK

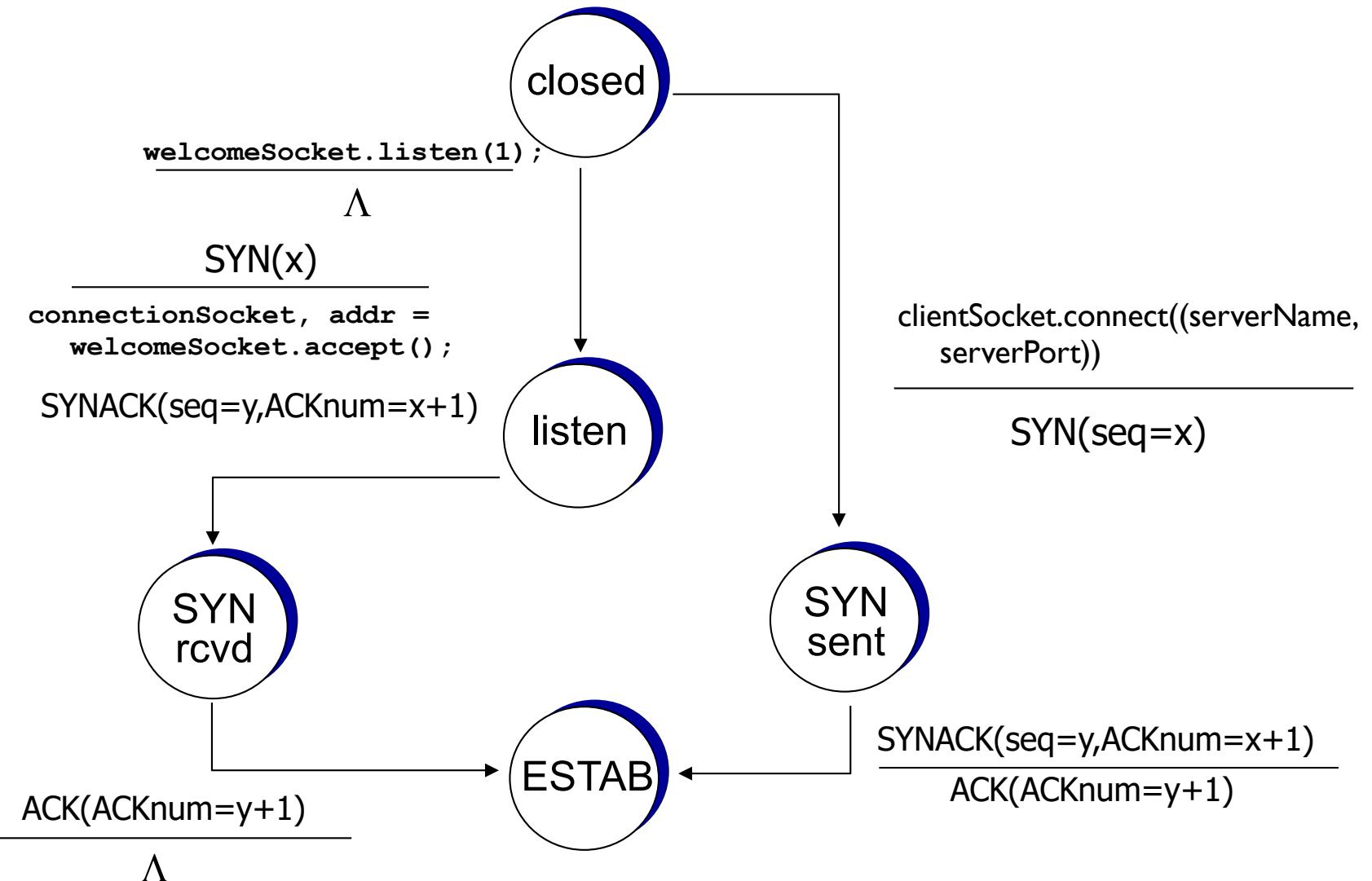
Flags: **SYN**  
**ACK**  
FIN  
RST  
PSH  
URG



**A tells B it's likewise okay to start sending**

**... upon receiving this packet, B can start sending data**

# TCP 3-way handshake: FSM



# What if the SYN Packet Gets Lost?

---

- ❖ Suppose the SYN packet gets lost
  - Packet is lost inside the network, or:
  - Server **discards** the packet (e.g., it's too busy)
- ❖ Eventually, no SYN-ACK arrives
  - Sender sets a **timer** and **waits** for the SYN-ACK
  - ... and retransmits the SYN if needed
- ❖ How should the TCP sender set the timer?
  - Sender has **no idea** how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - **SHOULD** (RFCs 1122,2988) use default of **3 second**,  
RFC 6298 use default of **1 second**

# SYN Loss and Web Downloads

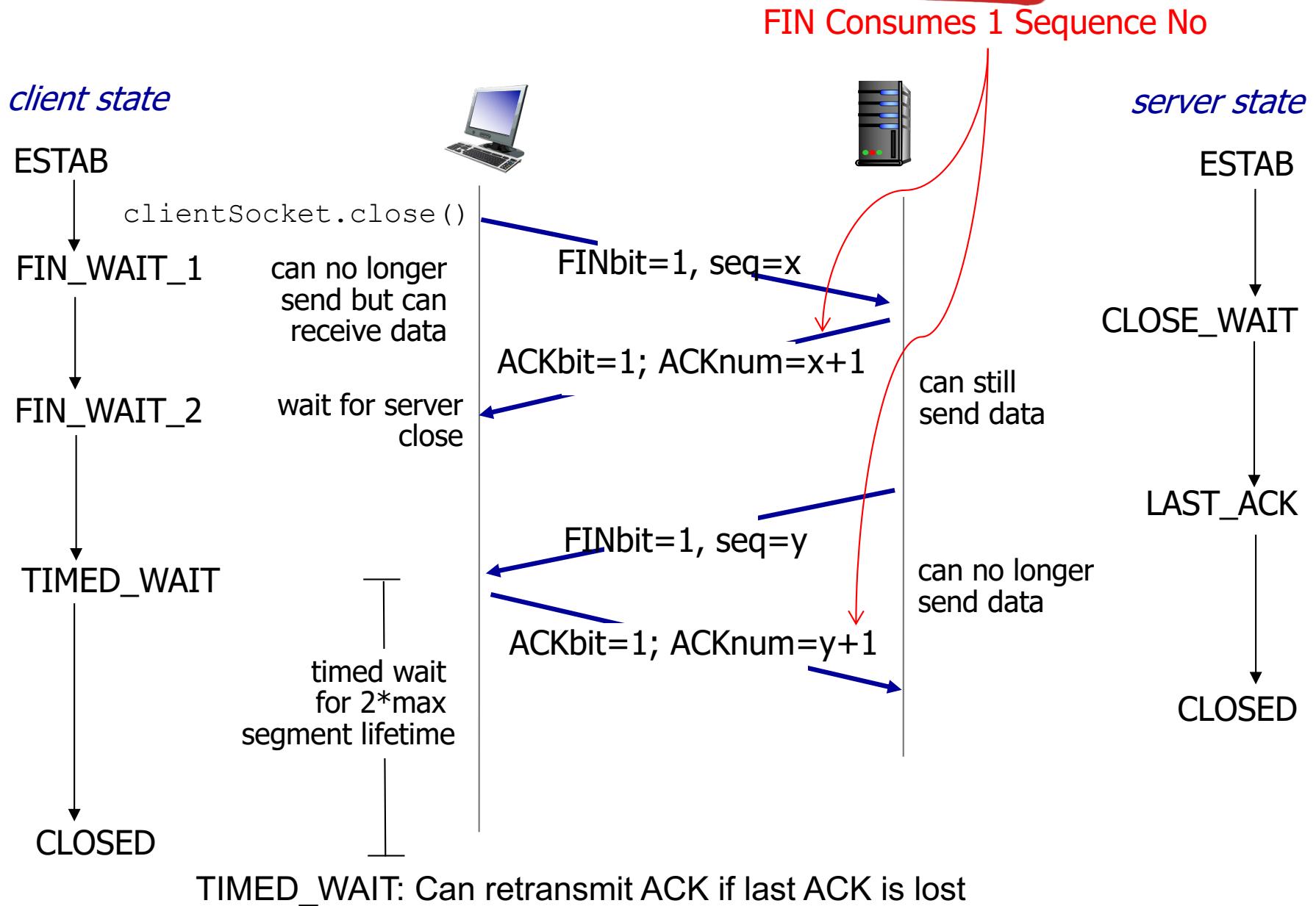
---

- ❖ User clicks on a hypertext link
  - Browser creates a socket and does a “connect”
  - The “connect” triggers the OS to transmit a SYN
- ❖ If the SYN is lost...
  - 1-3 seconds of delay: can be **very long**
  - User may become impatient
  - ... and click the hyperlink again, or click “reload”
- ❖ User triggers an “abort” of the “connect”
  - Browser creates a **new** socket and another “connect”
  - Essentially, forces a faster send of a new SYN packet!
  - Sometimes very effective, and the page comes quickly

# TCP: closing a connection

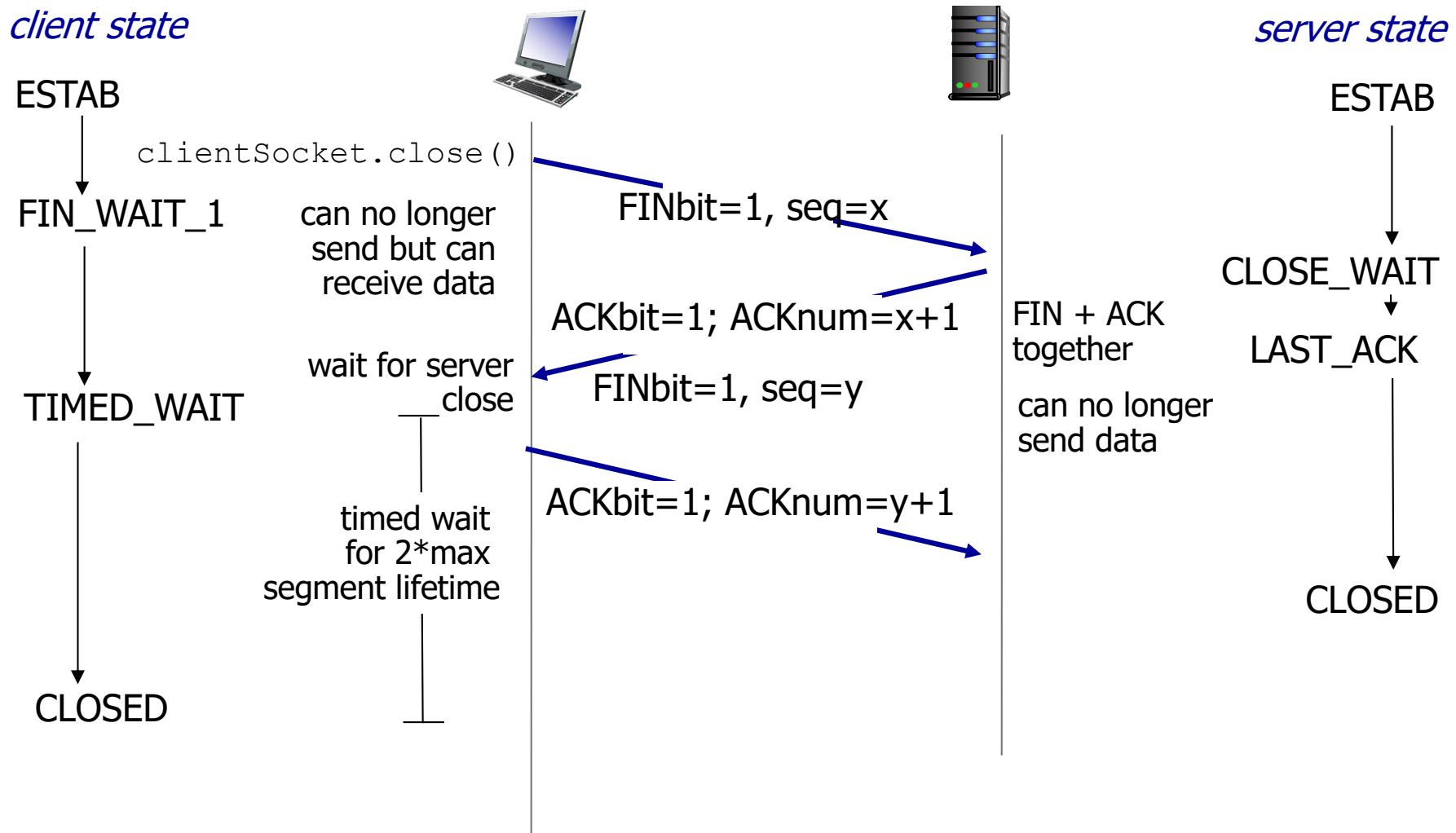
- ❖ client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- ❖ respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- ❖ simultaneous FIN exchanges can be handled

# Normal Termination, One at a Time

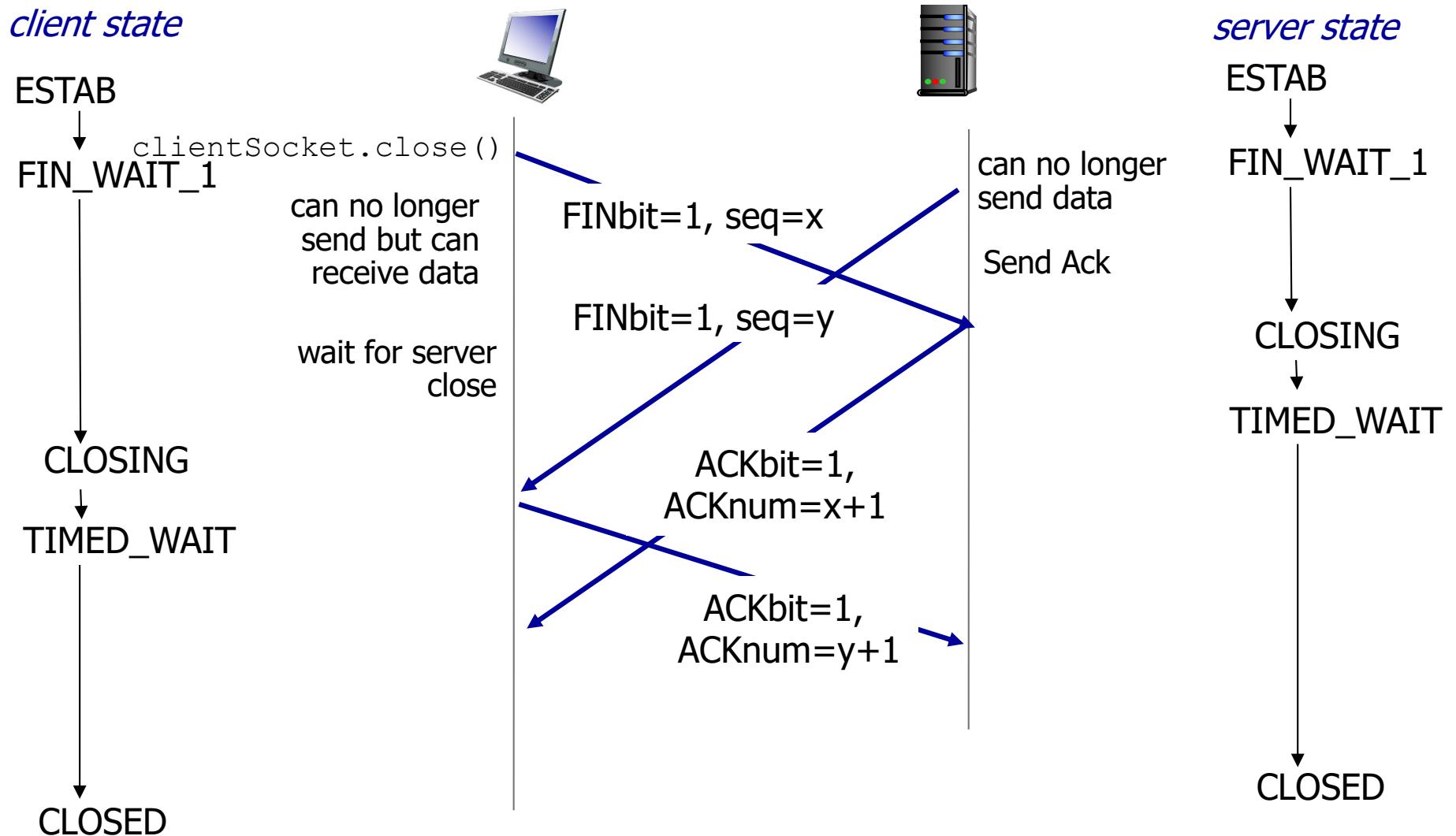


TIMED\_WAIT: Can retransmit ACK if last ACK is lost

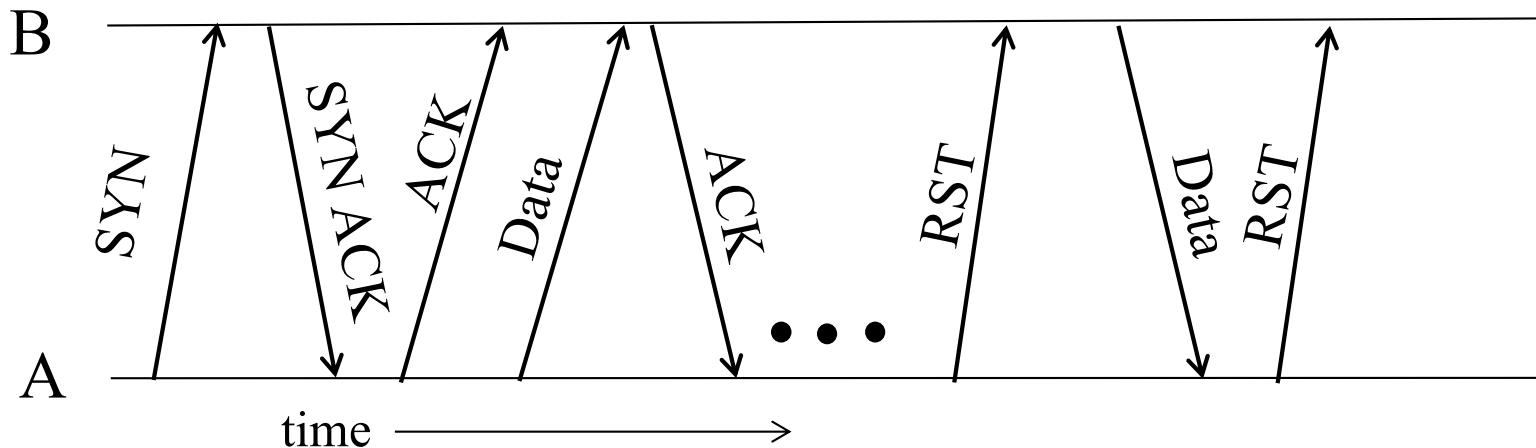
# Normal Termination, Both Together



# Simultaneous Closure

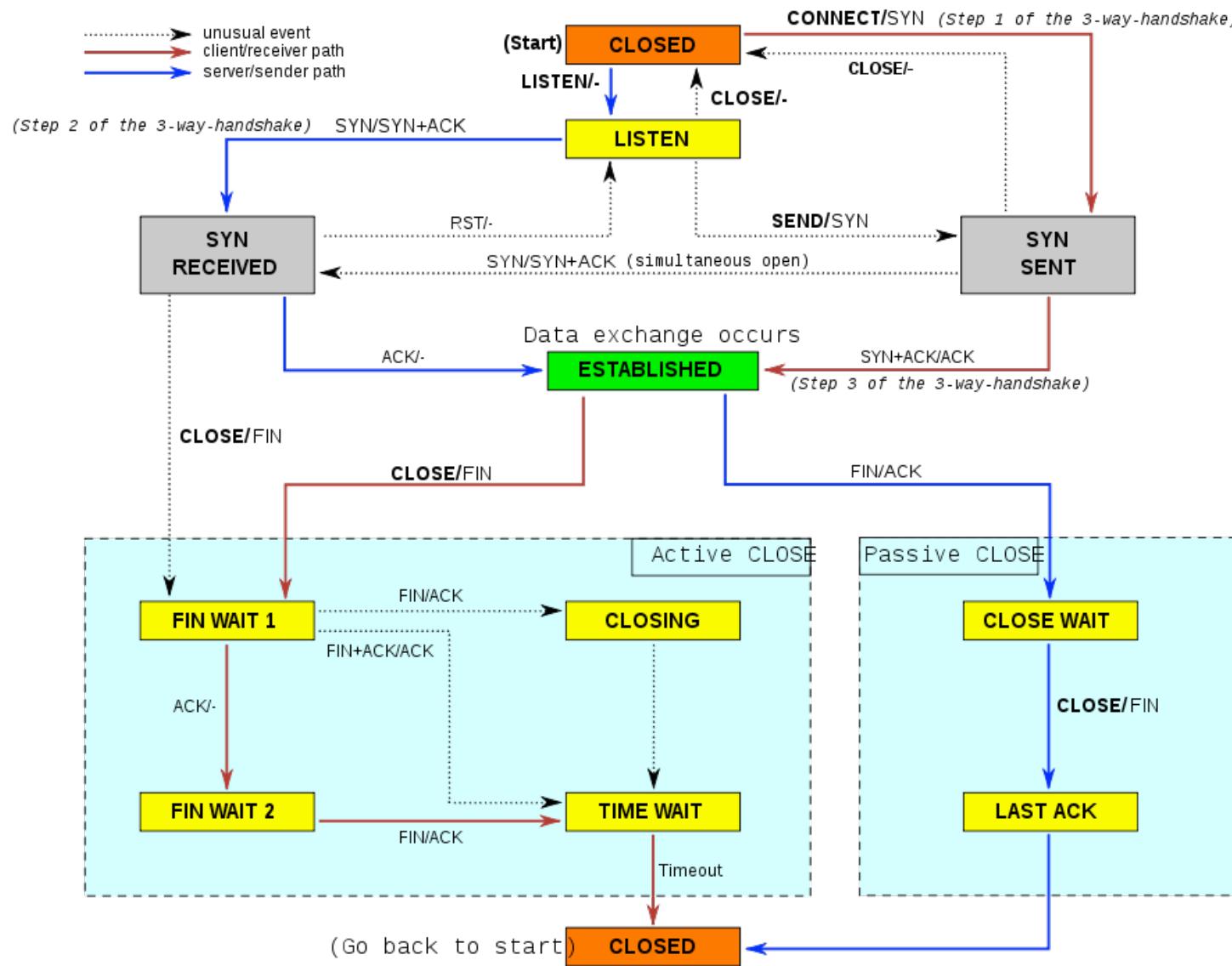


# Abrupt Termination



- ❖ A sends a RESET (**RST**) to B
  - E.g., because application process on A **crashed**
- ❖ **That's it**
  - B does **not** ack the **RST**
  - Thus, **RST** is **not** delivered **reliably**
  - And: any data in flight is **lost**
  - But: if B sends anything more, will elicit **another RST**

# TCP Finite State Machine



# TCP SYN Attack (SYN flooding)

- ❖ Miscreant creates a fake SYN packet
  - Destination is IP address of victim host (usually some server)
  - Source is some spoofed IP address
- ❖ Victim host on receiving creates a TCP connection state i.e allocates buffers, creates variables, etc and sends SYN ACK to the spoofed address (half-open connection)
- ❖ ACK never comes back
- ❖ After a timeout connection state is freed
- ❖ However for this duration the connection state is unnecessarily created
- ❖ Further miscreant sends large number of fake SYNs
  - Can easily overwhelm the victim
- ❖ Solutions:
  - Increase size of connection queue
  - Decrease timeout wait for the 3-way handshake
  - Firewalls: list of known bad source IP addresses
  - TCP SYN Cookies (explained on next slide)

# TCP SYN Cookie

---

- ❖ On receipt of SYN, server does not create connection state
- ❖ It creates an initial sequence number (*init\_seq*) that is a hash of source & dest IP address and port number of SYN packet (secret key used for hash)
  - Replies back with SYN ACK containing *init\_seq*
  - Server does not need to store this sequence number
- ❖ If original SYN is genuine, an ACK will come back
  - Same hash function run on the same header fields to get the initial sequence number (*init\_seq*)
  - Checks if the ACK is equal to (*init\_seq+1*)
  - Only create connection state if above is true
- ❖ If fake SYN, no harm done since no state was created

<http://etherealmind.com/tcp-syn-cookies-ddos-defence/>

# Quiz: TCP Connection Management?



Roughly how much time does it take for both the TCP Sender and Receiver to establish connection state since the connect() call?

- A. RTT
- B. 1.5RTT
- C. 2RTT
- D. 3RTT

**ANSWER: B**

Note that the final ACK will be typically piggybacked with the first data segment, so often the TCP connection setup is approximated to be 1RTT

# Quiz: TCP Connection Management?



Assume that one end point of the TCP connection sends a FIN segment. If it never receives an ACK, what should it do?

- A. Assume that the connection is closed and do nothing
- B. Retransmit the FIN
- C. Transmit an ACK
- D. Start crying

**ANSWER: B**

# Transport Layer: 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

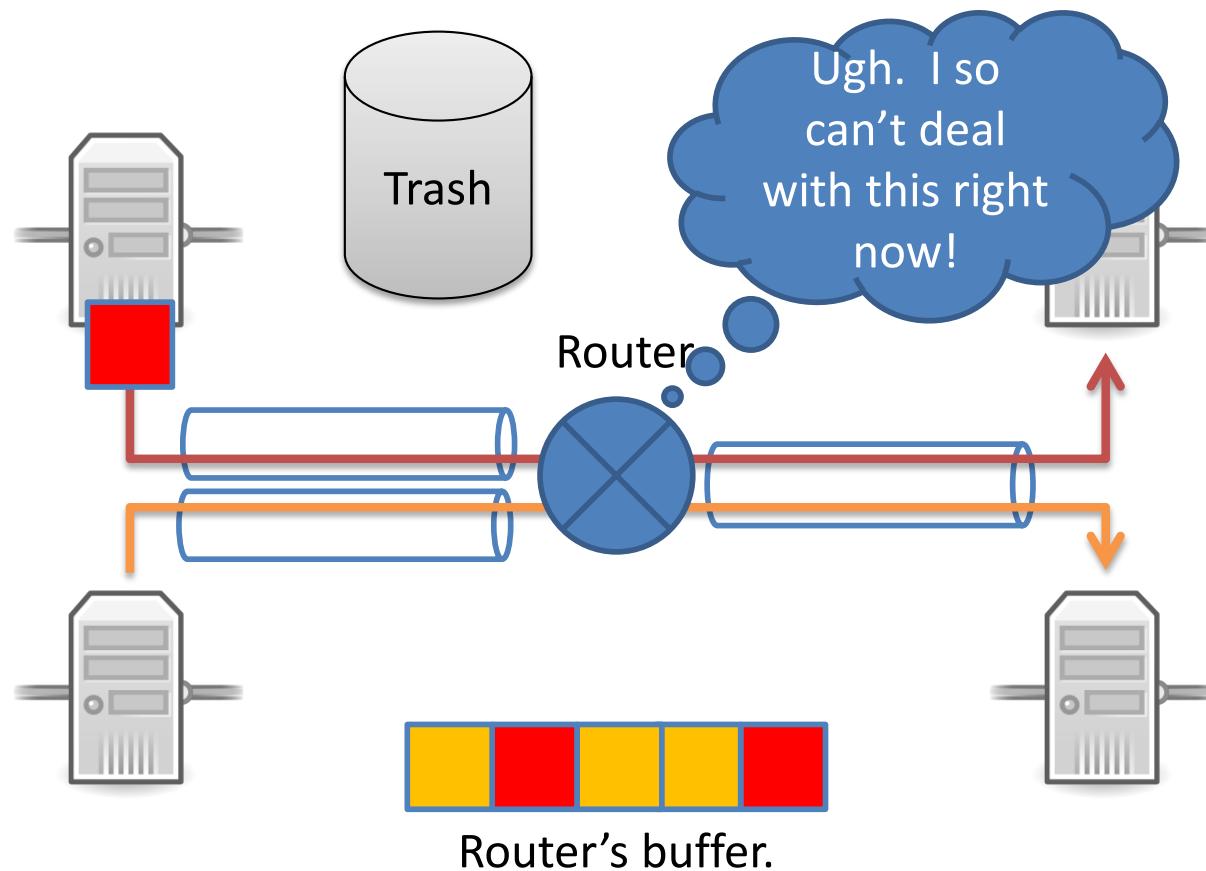
3.7 TCP congestion control

# Principles of congestion control

*congestion:*

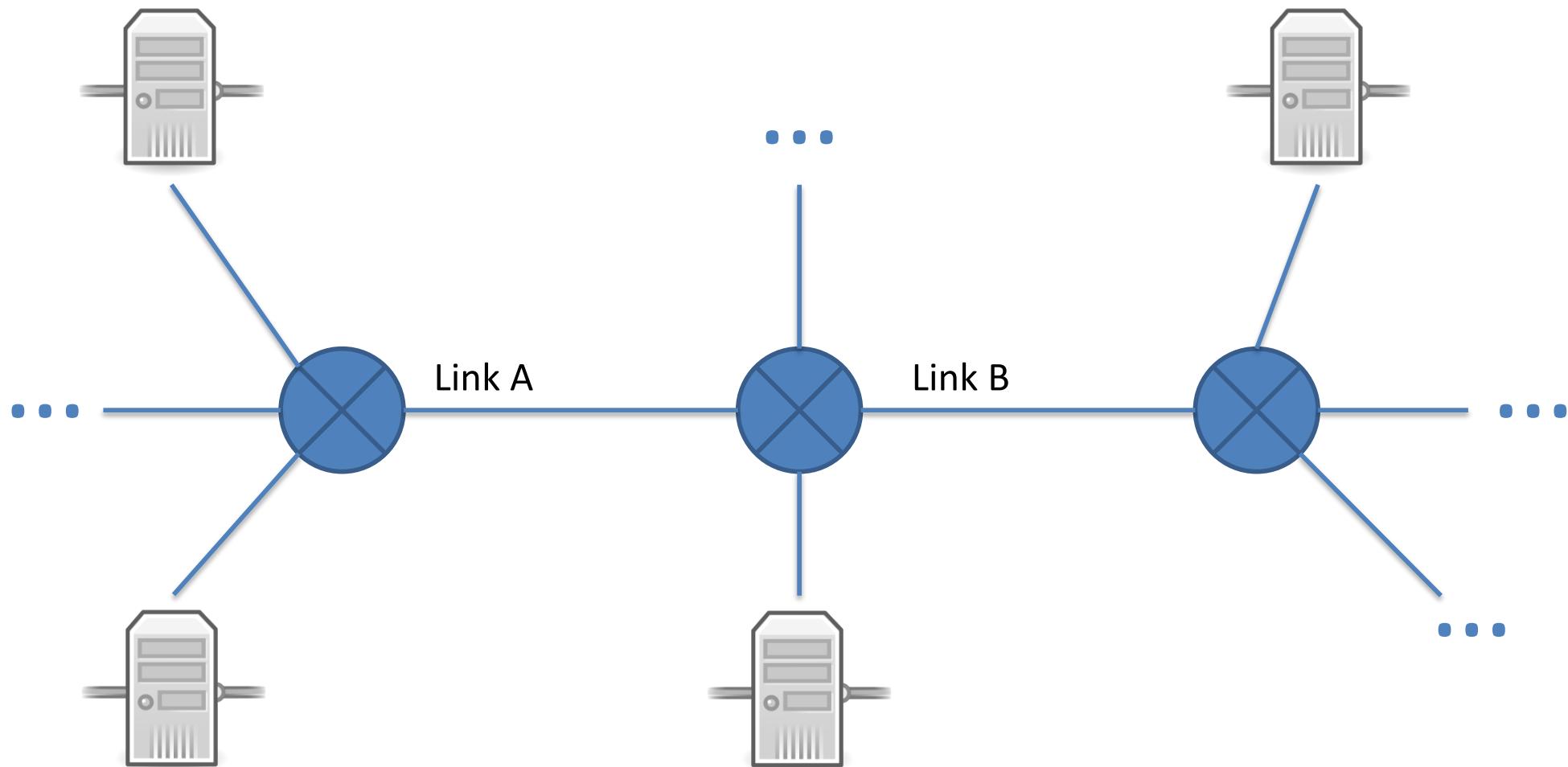
- ❖ informally: “too many sources sending too much data too fast for *network* to handle”
- ❖ different from flow control!
- ❖ manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- ❖ a top-10 problem!

# Congestion

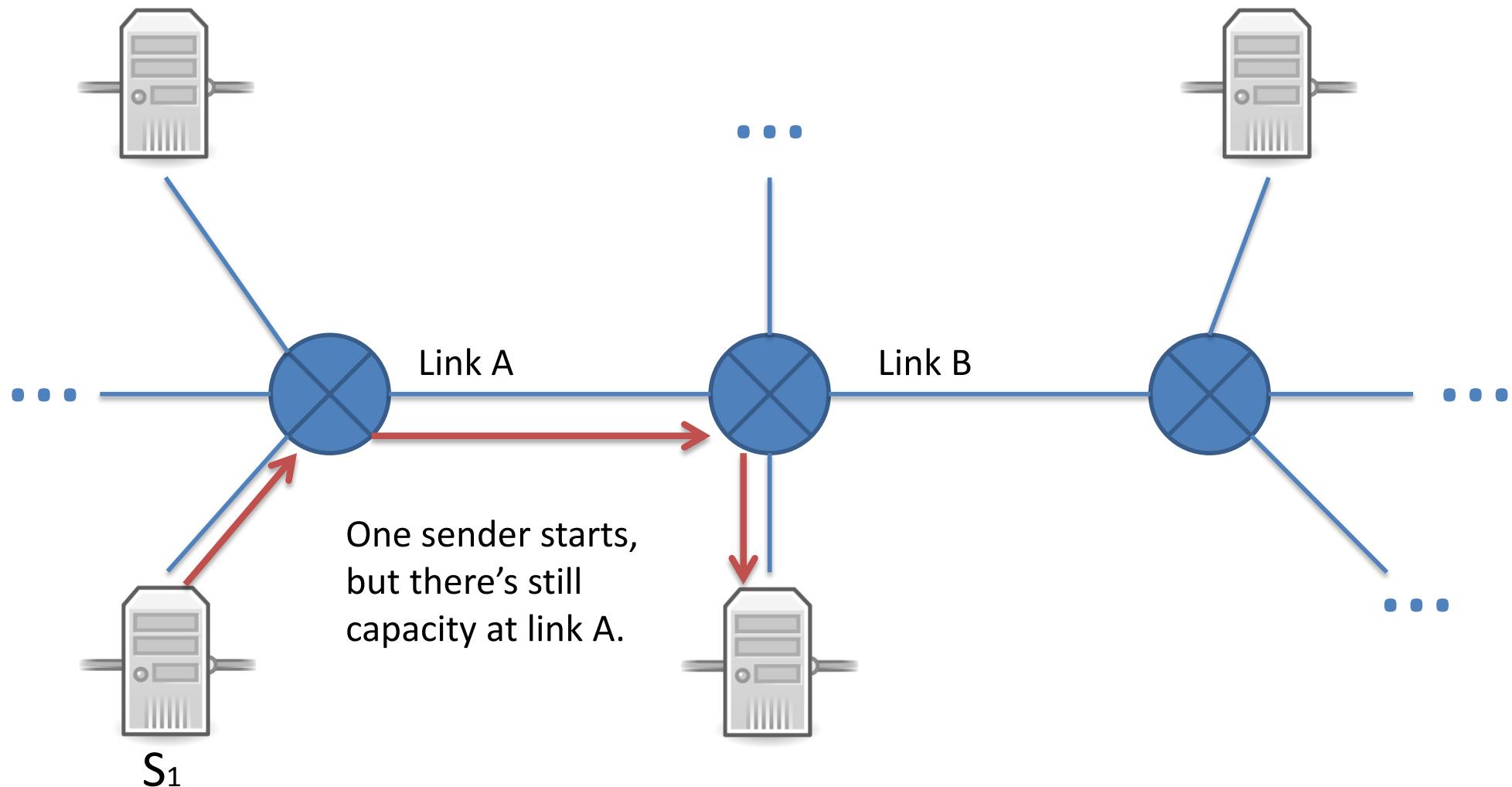


Incoming rate is faster than outgoing link can support.

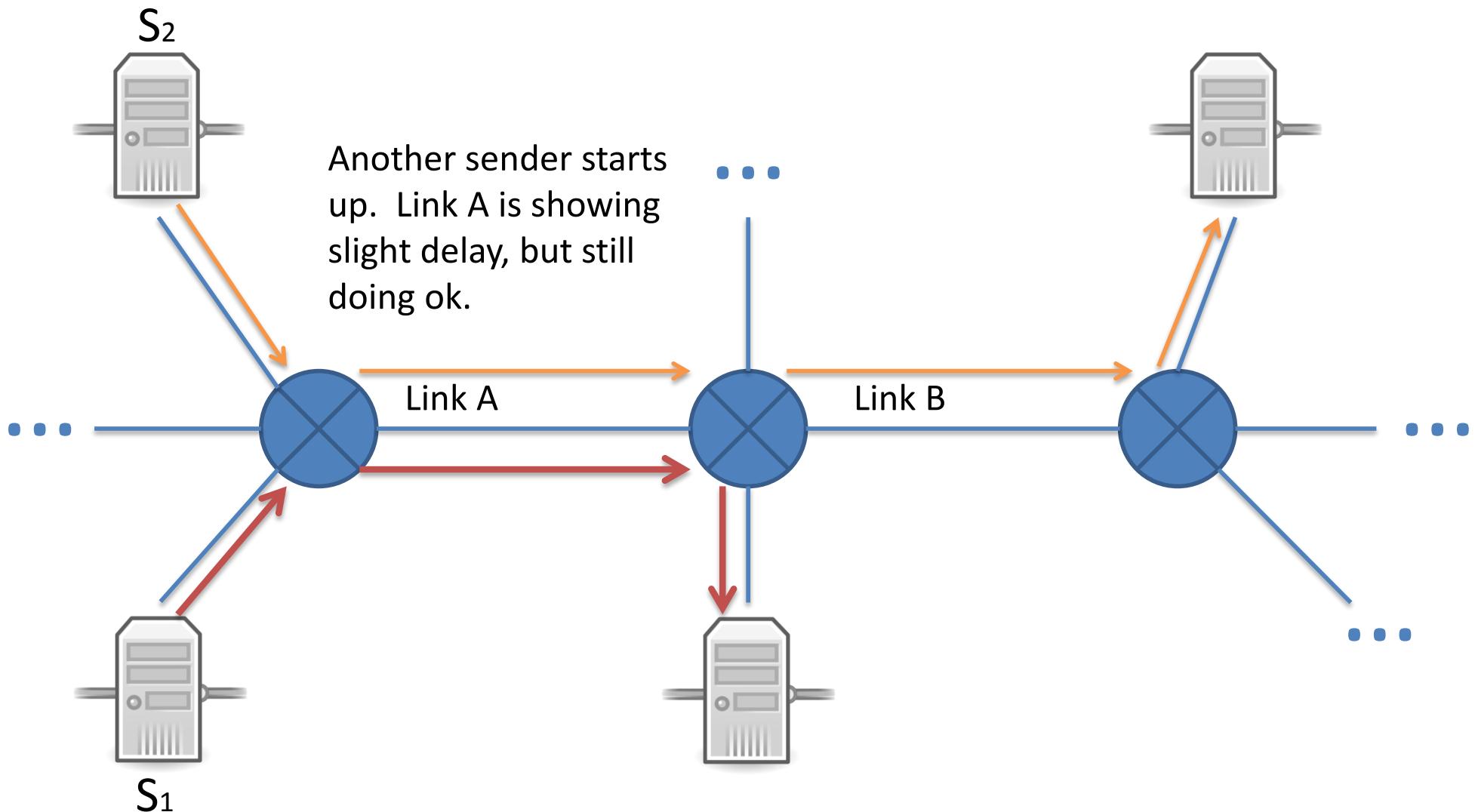
# Congestion Collapse



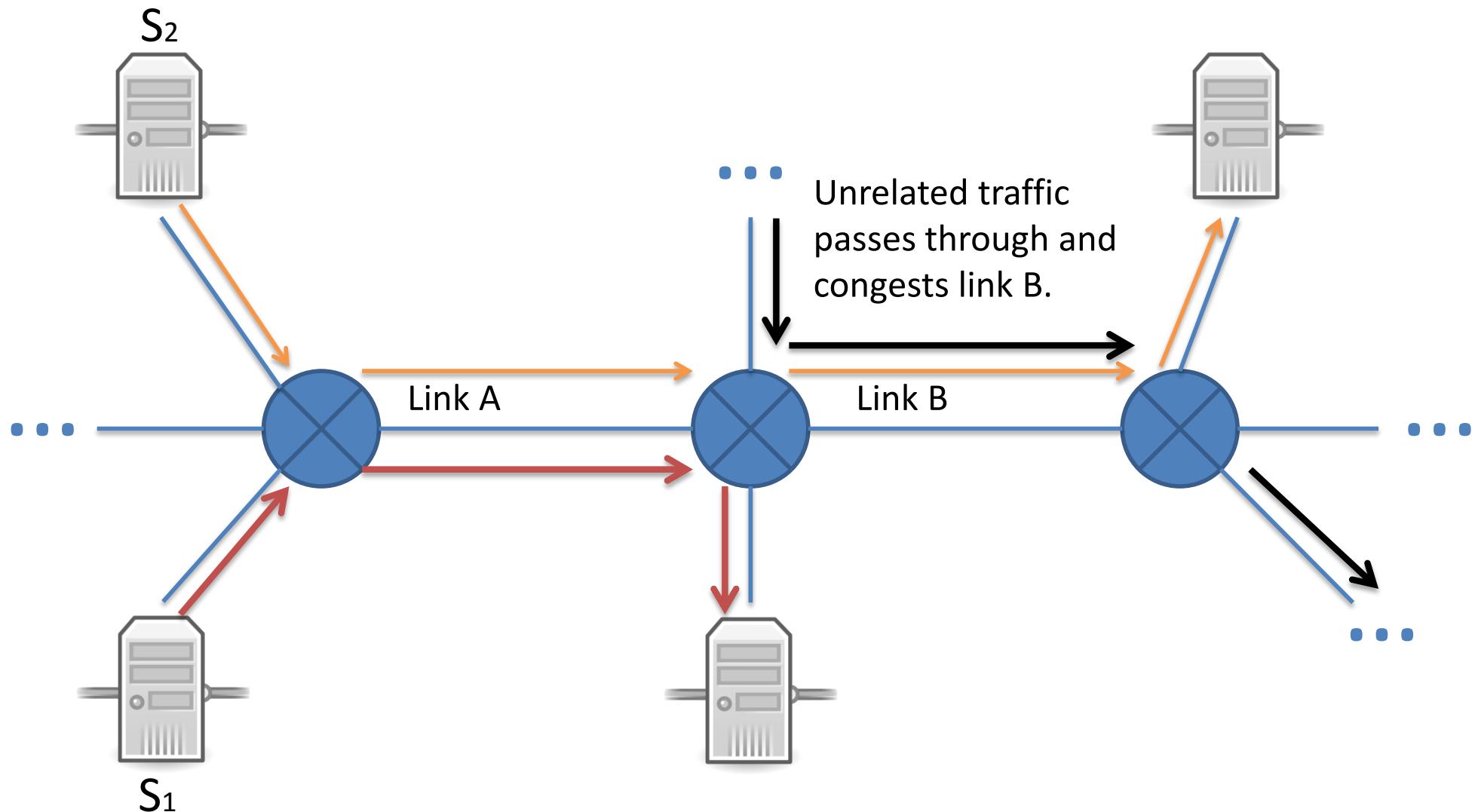
# Congestion Collapse



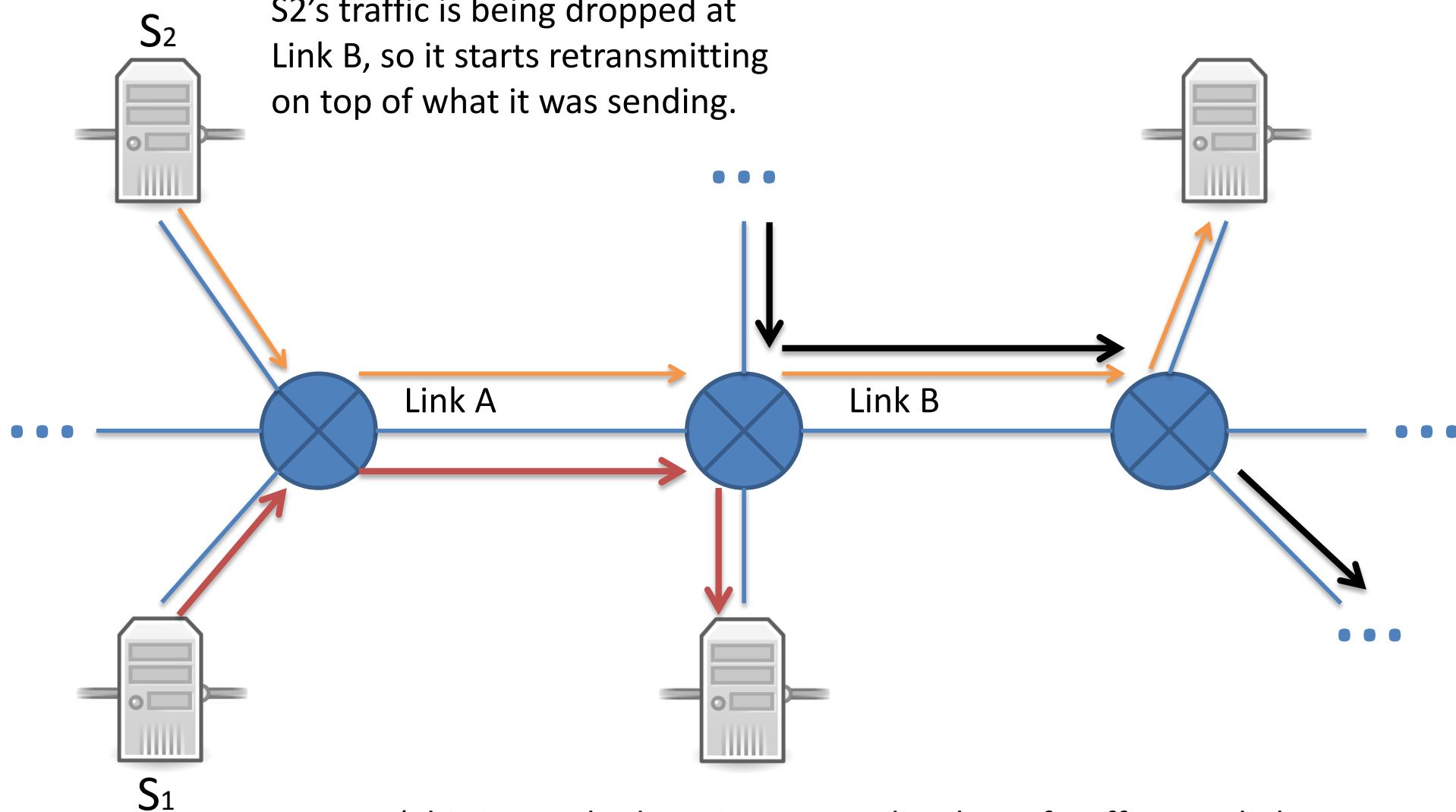
# Congestion Collapse



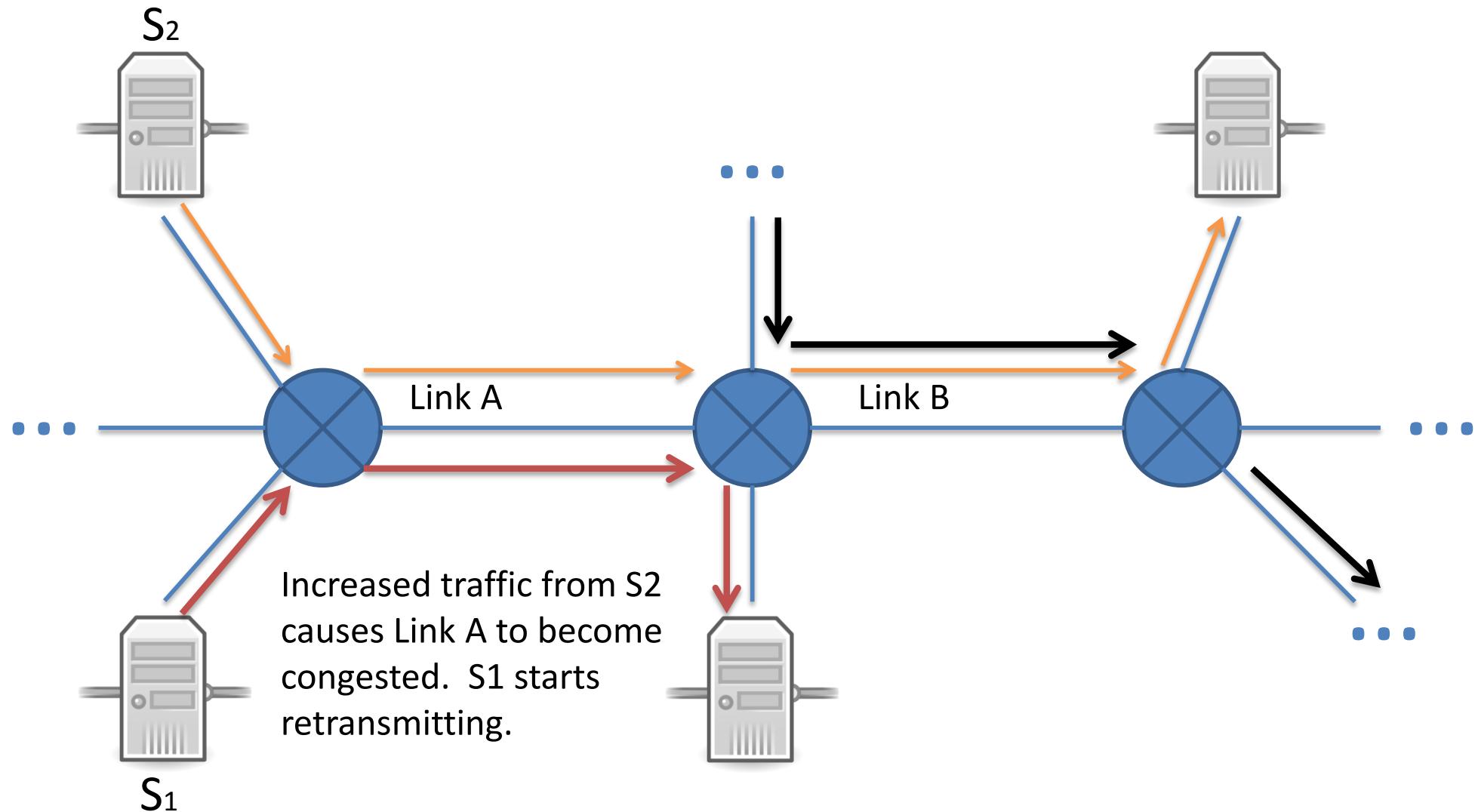
# Congestion Collapse



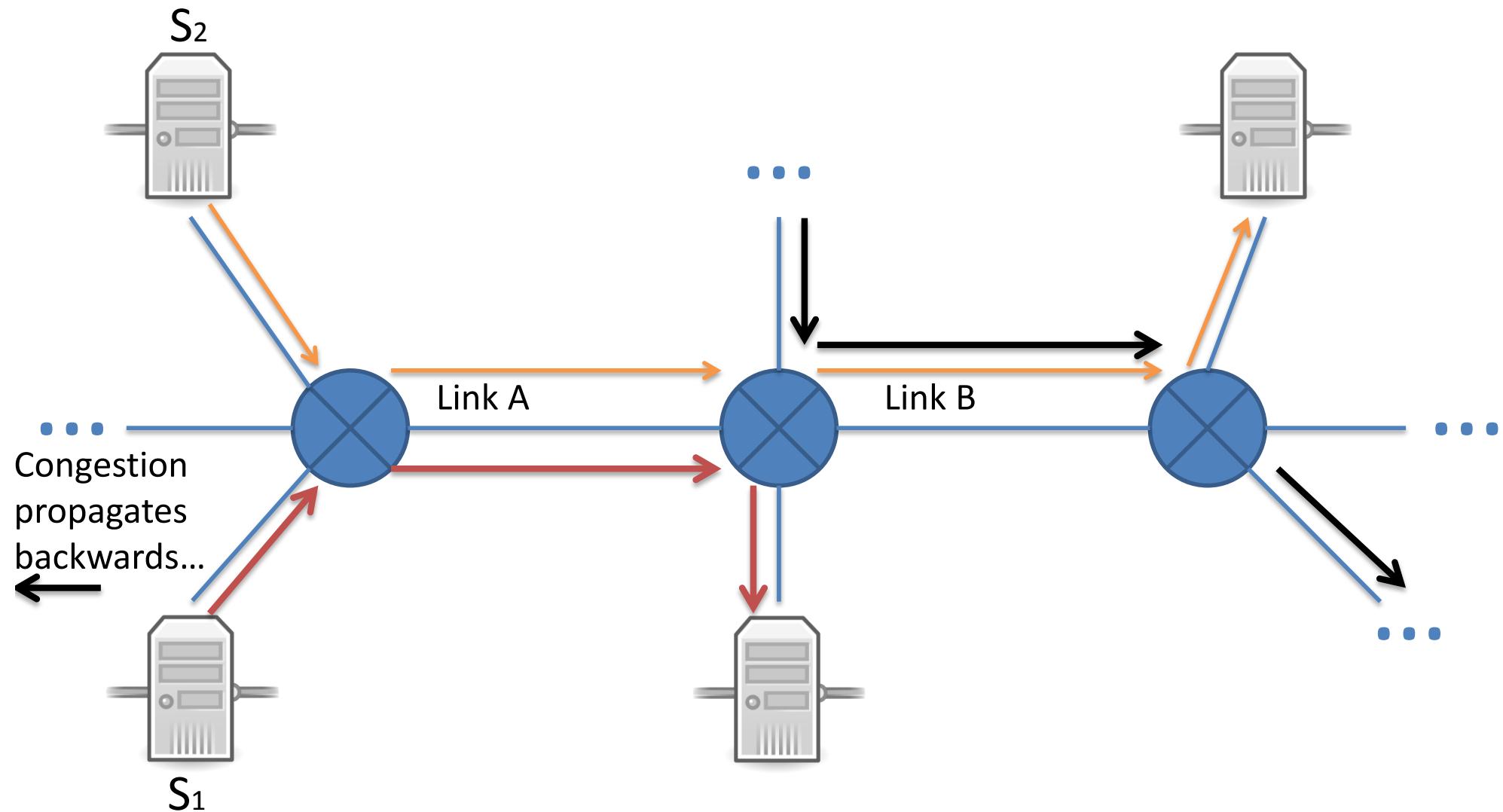
# Congestion Collapse



# Congestion Collapse



# Congestion Collapse



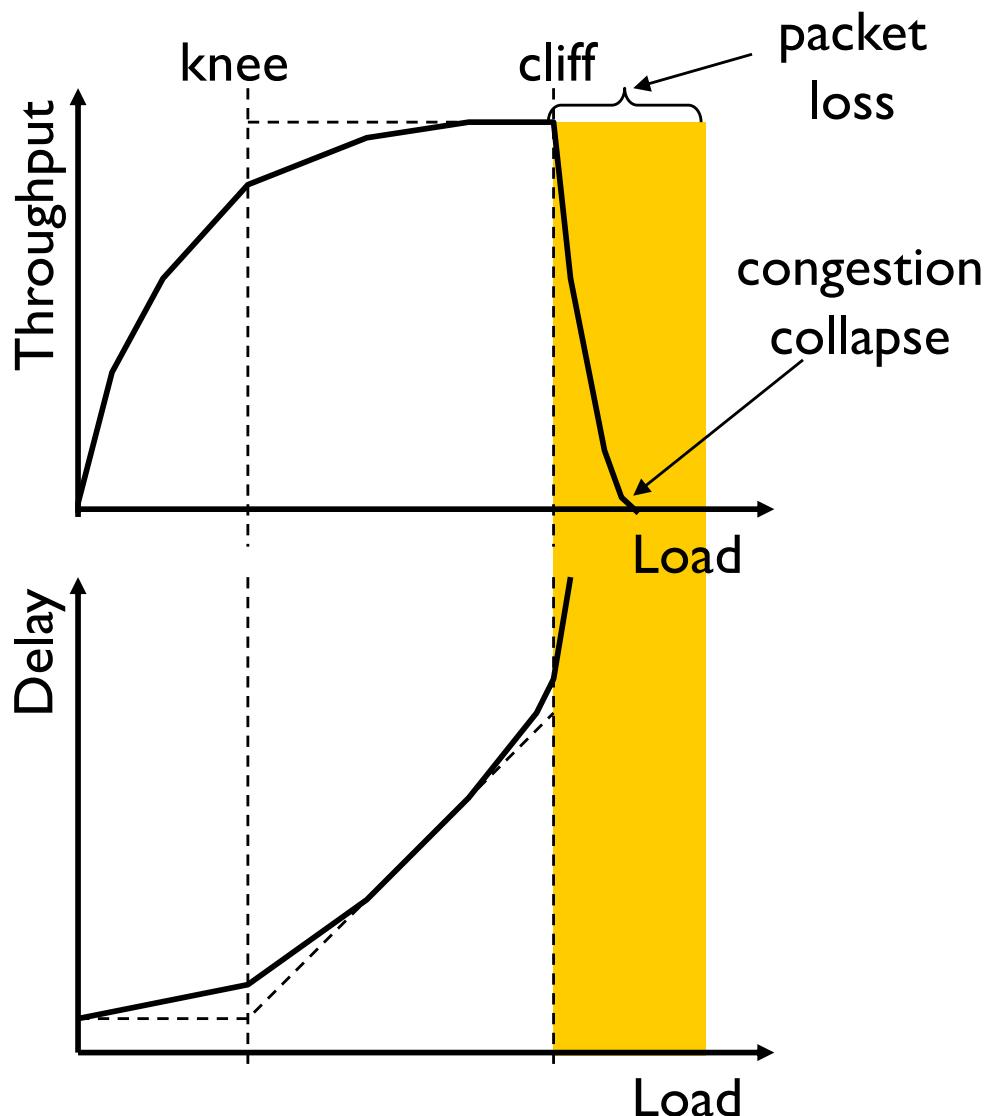
# Without congestion control

*congestion:*

- ❖ Increases delays
  - If delays > RTO, sender retransmits
- ❖ Increases loss rate
  - Dropped packets also retransmitted
- ❖ Increases retransmissions, many unnecessary
  - Wastes capacity of traffic that is never delivered
  - Increase in load results in decrease in useful work done
- ❖ Increases congestion, cycle continues ...

# Cost of Congestion

- ❖ Knee – point after which
  - Throughput increases slowly
  - Delay increases fast
  
- ❖ Cliff – point after which
  - Throughput starts to drop to zero (congestion collapse)
  - Delay approaches infinity



# Congestion Collapse

*This happened to the Internet (then NSFnet) in 1986*

- ❖ Rate dropped from a *blazing* 32 Kbps to 40bps
- ❖ This happened on and off for two years
- ❖ In 1988, Van Jacobson published “Congestion Avoidance and Control”
- ❖ The fix: senders voluntarily limit sending rate

# Approaches towards congestion control

two broad approaches towards congestion control:

## end-end congestion control:

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

## network-assisted congestion control:

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

# Transport Layer: 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

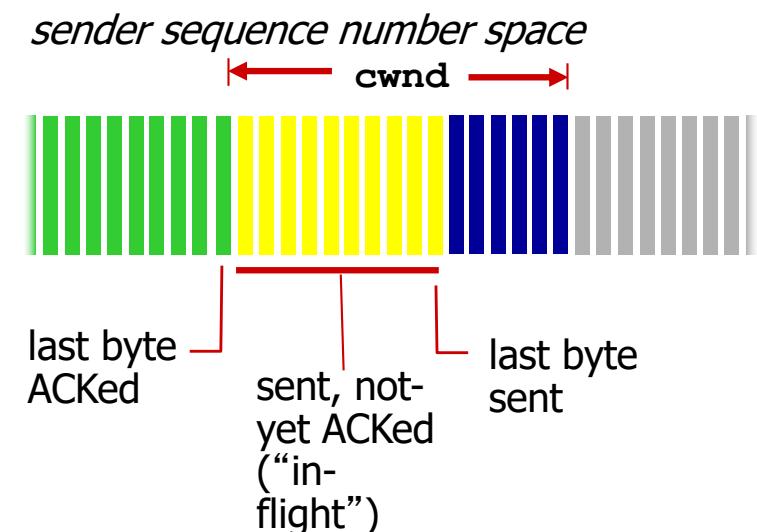
3.6 principles of congestion control

3.7 TCP congestion control

# TCP's Approach in a Nutshell

- ❖ TCP connection maintains a **window**
  - Controls number of packets in flight
- ❖ *TCP sending rate:*
  - roughly: send cwnd bytes, wait RTT for ACKs, then send more bytes

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



- ❖ **Vary window size to control sending rate**

# All These Windows...

- ❖ Congestion Window: **CWND**
  - How many bytes can be sent without overflowing routers
  - Computed by the sender using congestion control algorithm
- ❖ Flow control window: **Advertised / Receive Window (RWND)**
  - How many bytes can be sent without overflowing receiver's buffers
  - Determined by the receiver and reported to the sender
- ❖ Sender-side window = **minimum{CWND, RWND}**
  - Assume for this discussion that RWND >> CWND

# CWND

- ❖ This lecture will talk about CWND in units of MSS
  - (Recall MSS: Maximum Segment Size, the amount of payload data in a TCP packet)
  - This is only for pedagogical purposes
- ❖ Keep in mind that real implementations maintain CWND in bytes

# Two Basic Questions

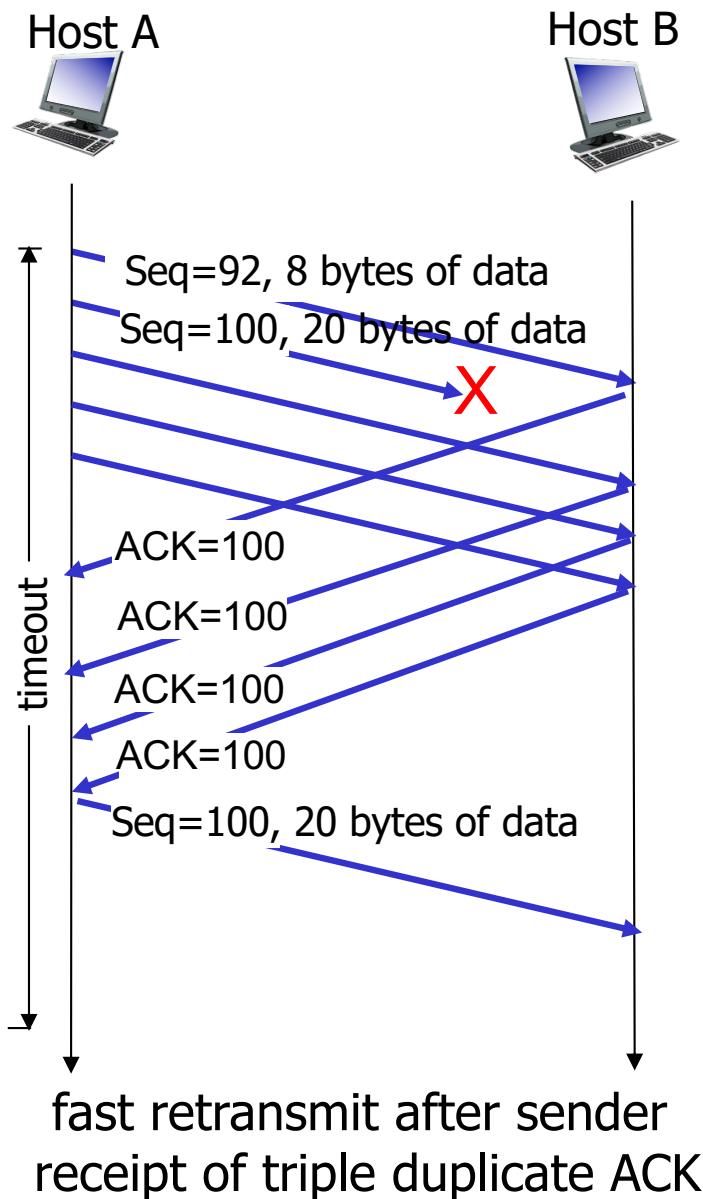
- ❖ How does the sender detect congestion?
- ❖ How does the sender adjust its sending rate?

# Detection Congestion: Infer Loss

---

- ❖ Duplicate ACKs: isolated loss
  - dup ACKs indicate network capable of delivering some segments
- ❖ Timeout: much more serious
  - Not enough dup ACKs
  - Must have suffered several losses
- ❖ Will adjust rate differently for each case

# RECAP: TCP fast retransmit (dup acks)



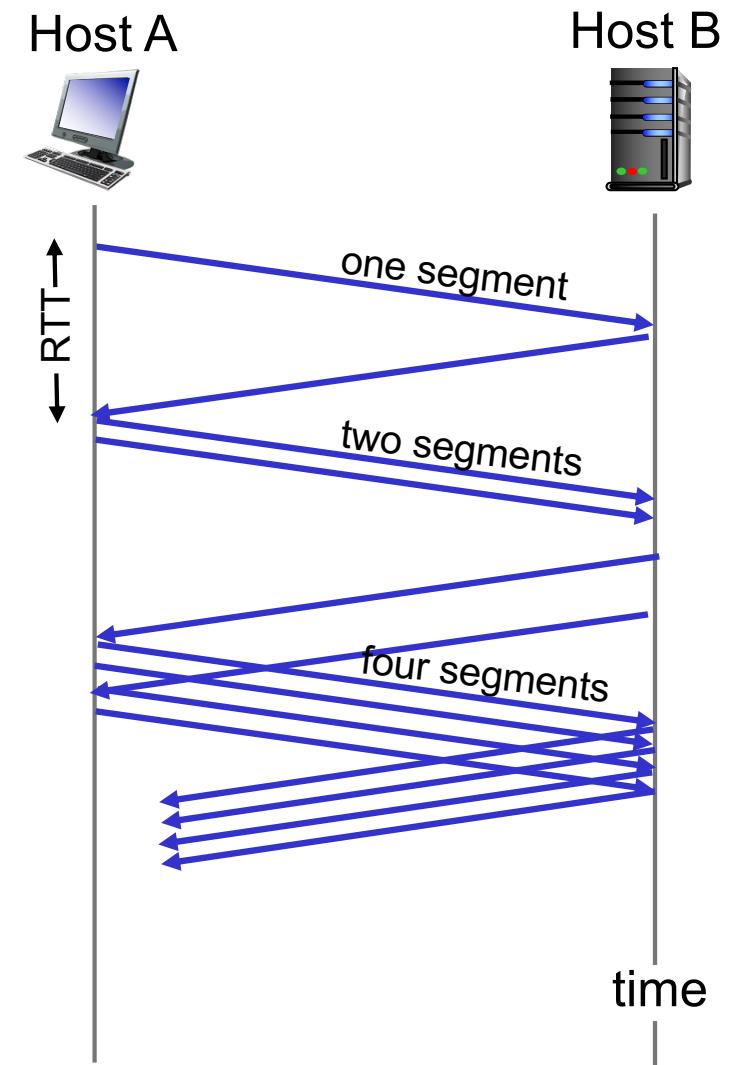
# Rate Adjustment

---

- ❖ Basic structure:
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate
- ❖ How we increase/decrease the rate depends on the phase of congestion control we're in:
  - Discovering available bottleneck bandwidth vs.
  - Adjusting to bandwidth variations

# TCP Slow Start (Bandwidth discovery)

- ❖ when connection begins, increase rate **exponentially** until **first loss event**:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT (full ACKs)
  - Simpler implementation achieved by incrementing **cwnd** for every ACK received
    - $cwnd += 1$  for each ACK
- ❖ **summary:** initial rate is slow but ramps up exponentially fast



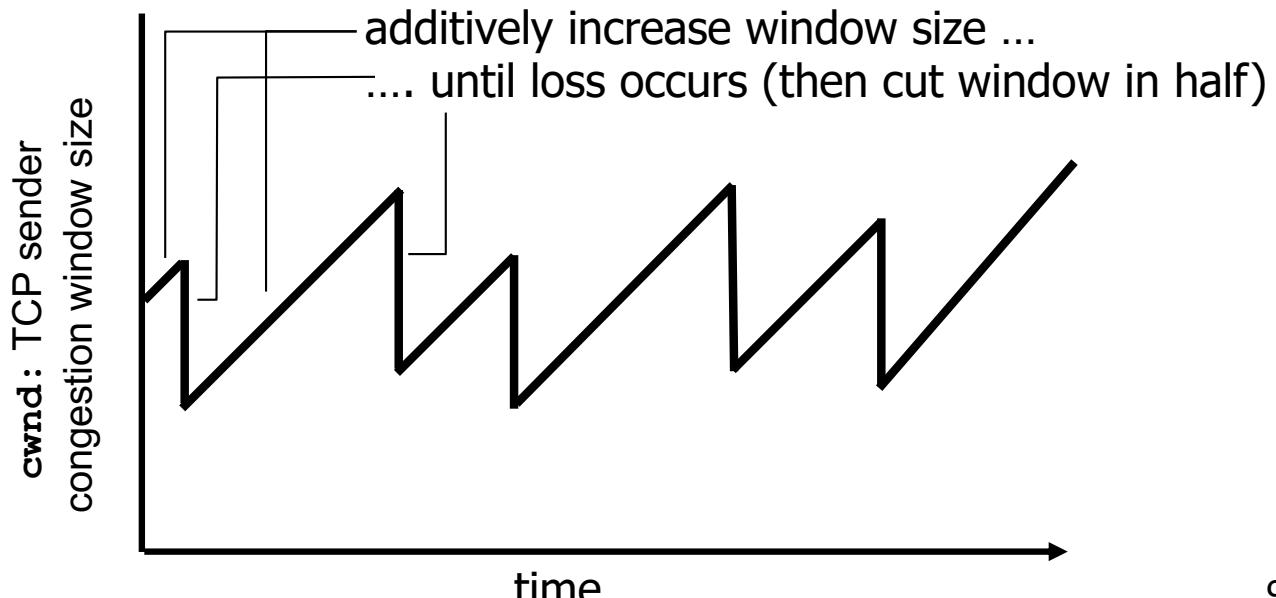
# Adjusting to Varying Bandwidth

- ❖ Slow start gave an estimate of available bandwidth
- ❖ Now, want to track variations in this available bandwidth, oscillating around its current value
  - Repeated probing (rate increase) and backoff (rate decrease)
  - Known as Congestion Avoidance (CA)
- ❖ TCP uses: “Additive Increase Multiplicative Decrease” (AIMD)

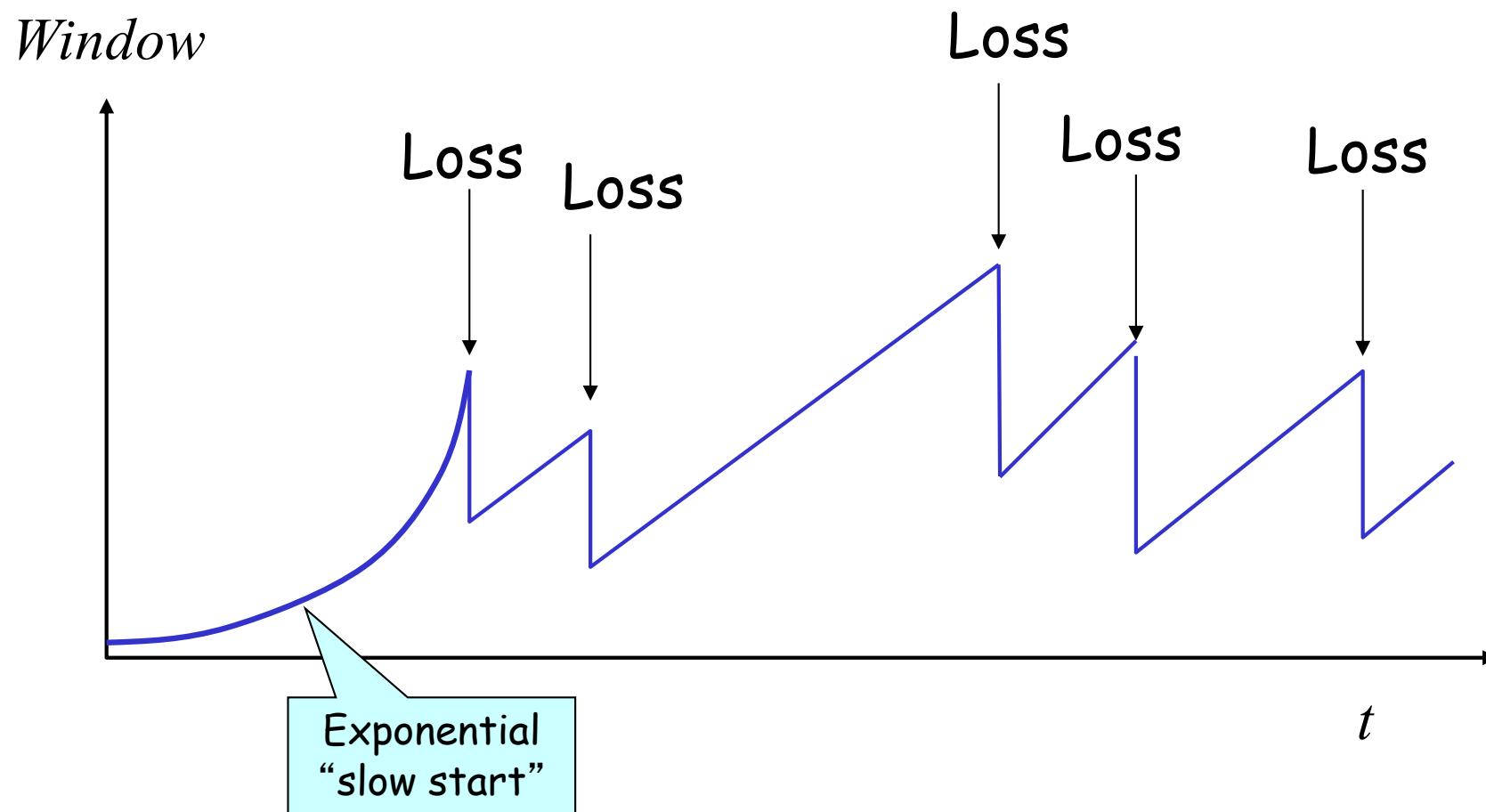
# AIMD

- ❖ **approach:** sender increases transmission rate (window size), probing for usable bandwidth, until another congestion event occurs
  - **additive increase:** increase **cwnd** by 1 MSS every RTT until loss detected
    - For each successful RTT (all ACKS),  $cwnd = cwnd + 1$
    - Simple implementation: for each ACK,  $cwnd = cwnd + 1/cwnd$  (since there are  $cwnd/MSS$  packets in a window)
  - **multiplicative decrease:** cut **cwnd** in half after loss

AIMD saw tooth behavior: probing for bandwidth



# Leads to the TCP “Sawtooth”



# Slow-Start vs. AIMD

- ❖ When does a sender stop Slow-Start and start Additive Increase?
- ❖ Introduce a “slow start threshold” (**ssthresh**)
  - Initialized to a large value
- ❖ Convert to AI when  $cwnd = ssthresh$ , sender switches from slow-start to AIMD-style increase
  - On timeout,  $ssthresh = CWND/2$

# Implementation

- ❖ State at sender

- CWND (initialized to a small constant)
- ssthresh (initialized to a large constant)
- [Also dupACKcount and timer, as before]

- ❖ Events

- ACK (new data)
- dupACK (duplicate ACK for old data)
- Timeout

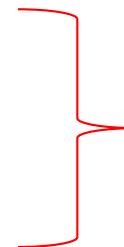
## Event: ACK (new data)

- ❖ If  $CWND < ssthresh$ 
  - $CWND += +$

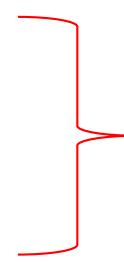
- Hence after one RTT (All ACKs with no drops):  
 $CWND = 2 \times CWND$

# Event: ACK (new data)

- ❖ If  $CWND < ssthresh$ 
  - $CWND += I$
- ❖ Else
  - $CWND = CWND + I/CWND$



*Slow start phase*



*“Congestion  
Avoidance” phase  
(additive increase)*

- Hence after one RTT (All ACKs with no drops):  
 $CWND = CWND + I$

## Event: dupACK

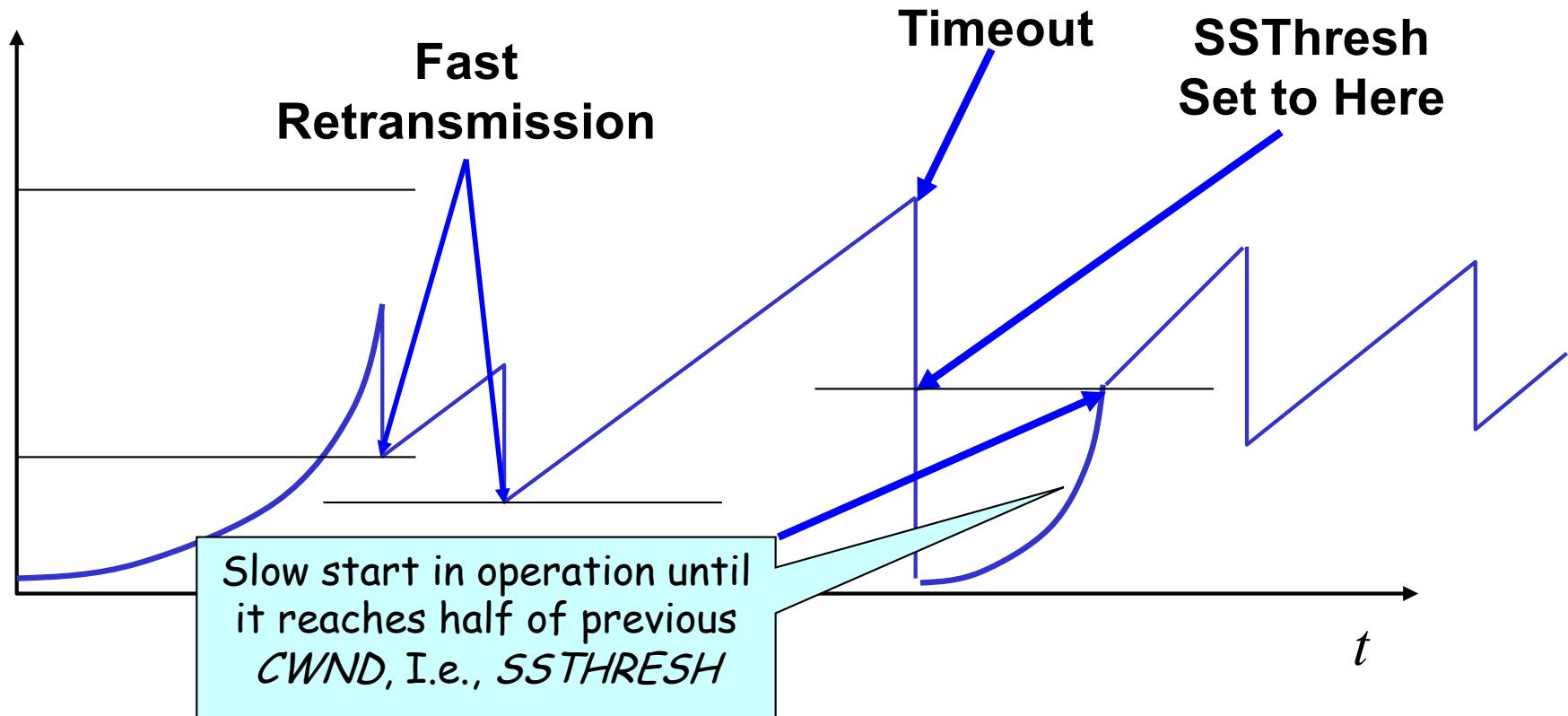
- ❖ dupACKcount ++
- ❖ If dupACKcount = 3 /\* fast retransmit \*/
  - ssthresh = CWND/2
  - **CWND = CWND/2**

# Event: TimeOut

- ❖ On Timeout
  - $ssthresh \leftarrow CWND/2$
  - $CWND \leftarrow 1$

# Example

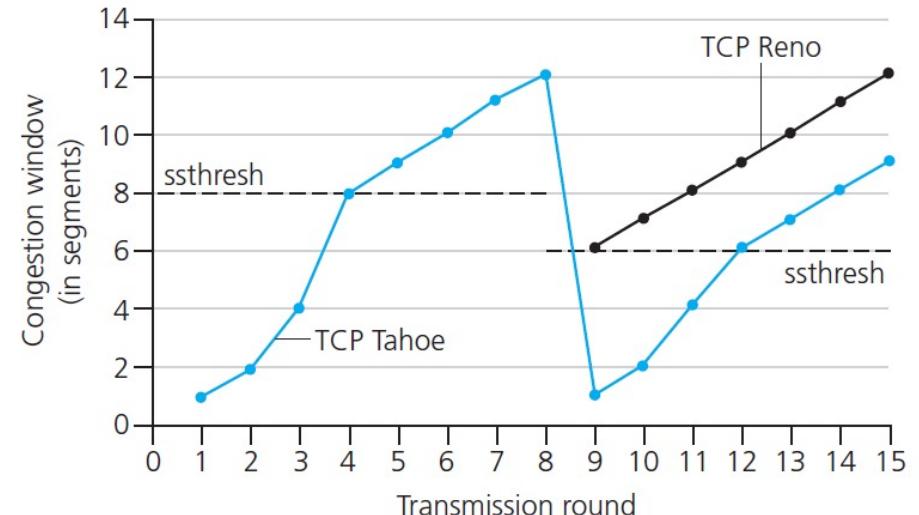
Window



Slow-start restart: Go back to  $CWND = 1$  MSS, but take advantage of knowing the previous value of  $CWND$

# TCP Flavours

- ❖ TCP-Tahoe
  - $cwnd = 1$  on triple dup ACK & timeout
- ❖ TCP-Reno
  - $cwnd = 1$  on timeout
  - $cwnd = cwnd/2$  on triple dup ACK
- ❖ TCP-newReno
  - TCP-Reno + improved fast recovery (SKIPPED)
- ❖ TCP-SACK (NOT COVERED IN THE COURSE)
  - incorporates selective acknowledgements



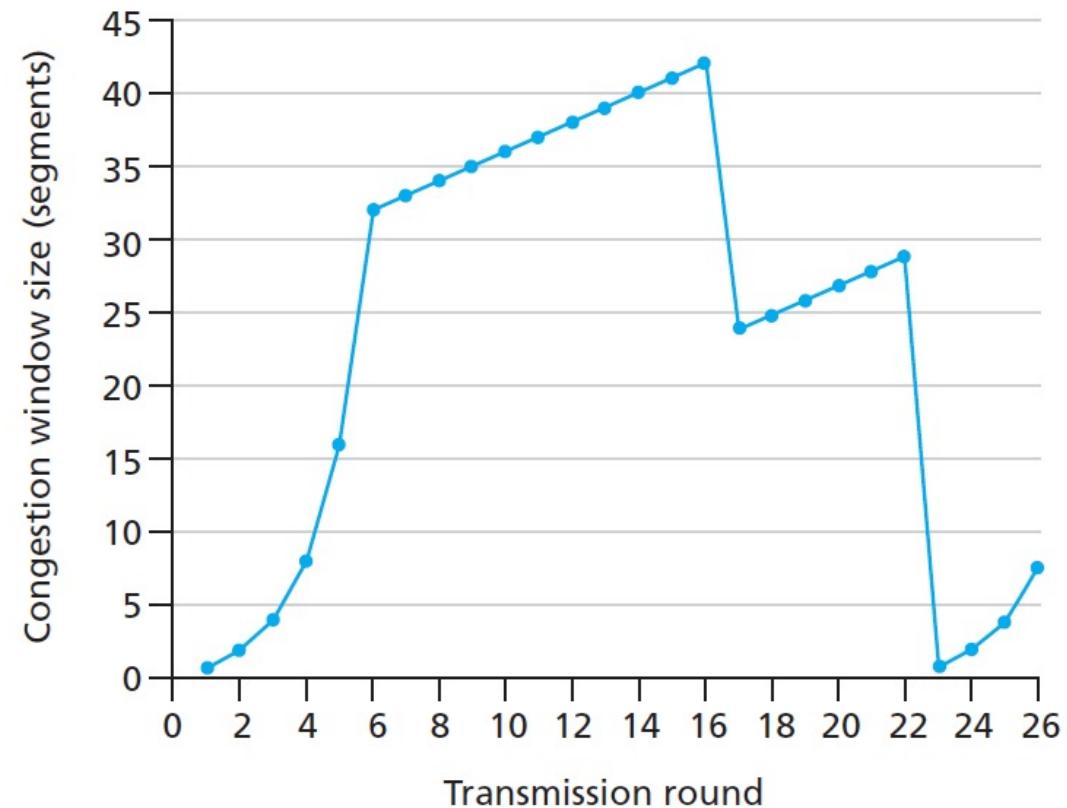
# Quiz: TCP Congestion Control?



In the figure how many congestion avoidance intervals can you identify?

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

**ANSWER: C**



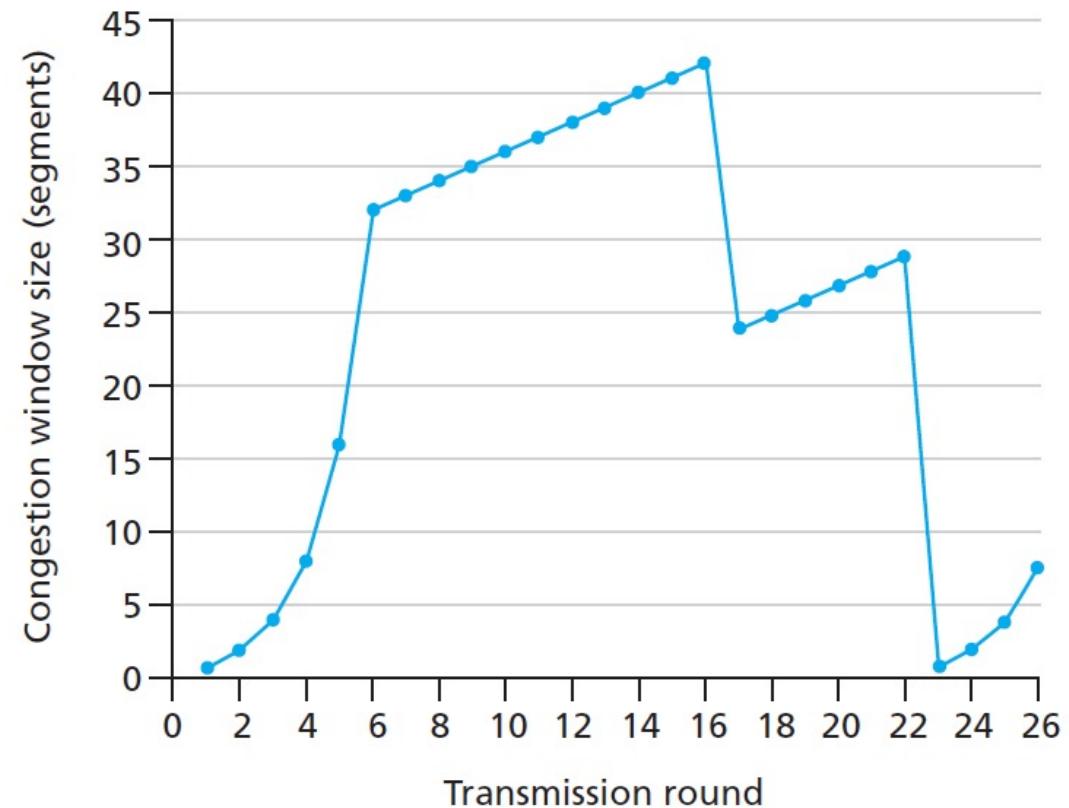
# Quiz: TCP Congestion Control?



In the figure how many slow start intervals can you identify?

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

**ANSWER: C**



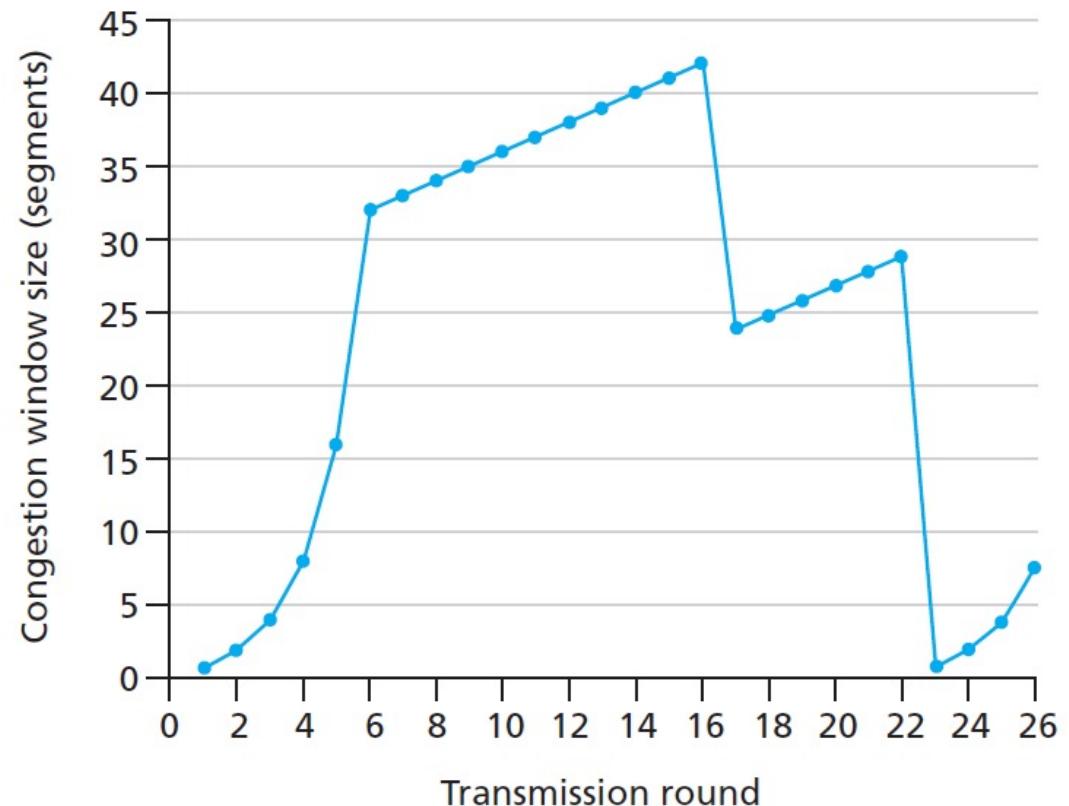
# Quiz: TCP Congestion Control?



In the figure after the 16<sup>th</sup> transmission round, segment loss is detected by \_\_\_\_\_?

- A. Triple Dup Ack
- B. Timeout

**ANSWER: A**



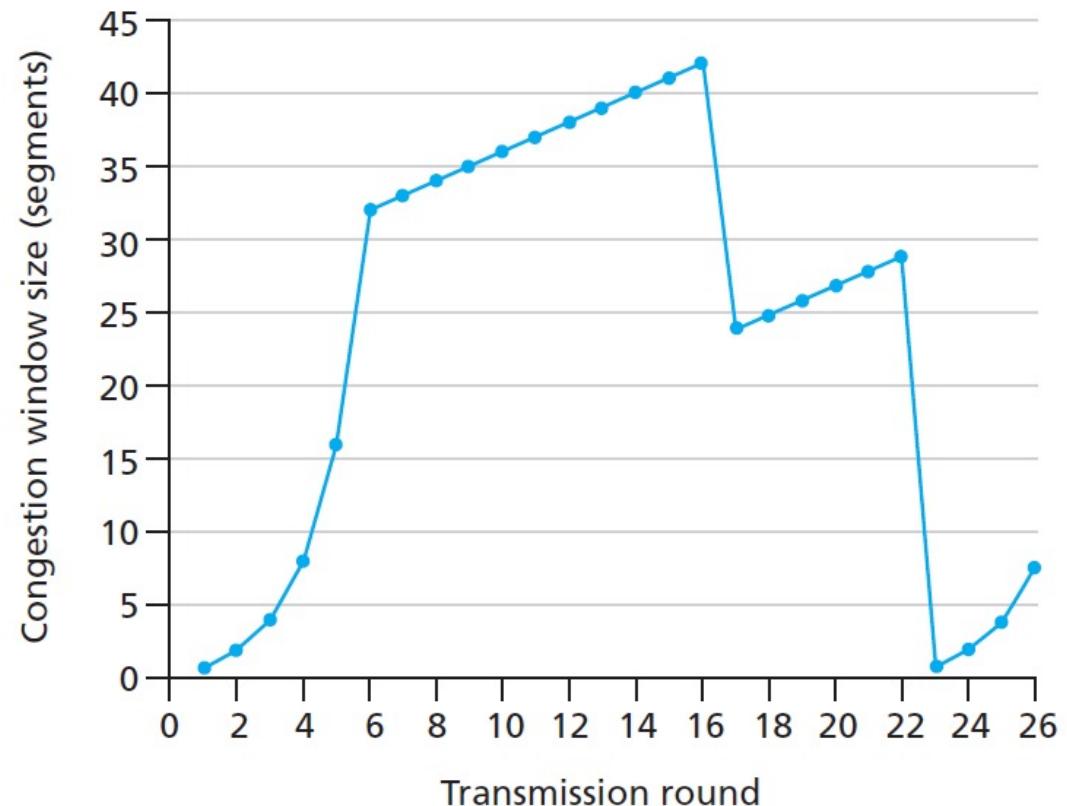
# Quiz: TCP Congestion Control?



In the figure what is the initial value of ssthresh (steady state threshold)?

- A. 0
- B. 28
- C. 32
- D. 42
- E. 64

**ANSWER: C**



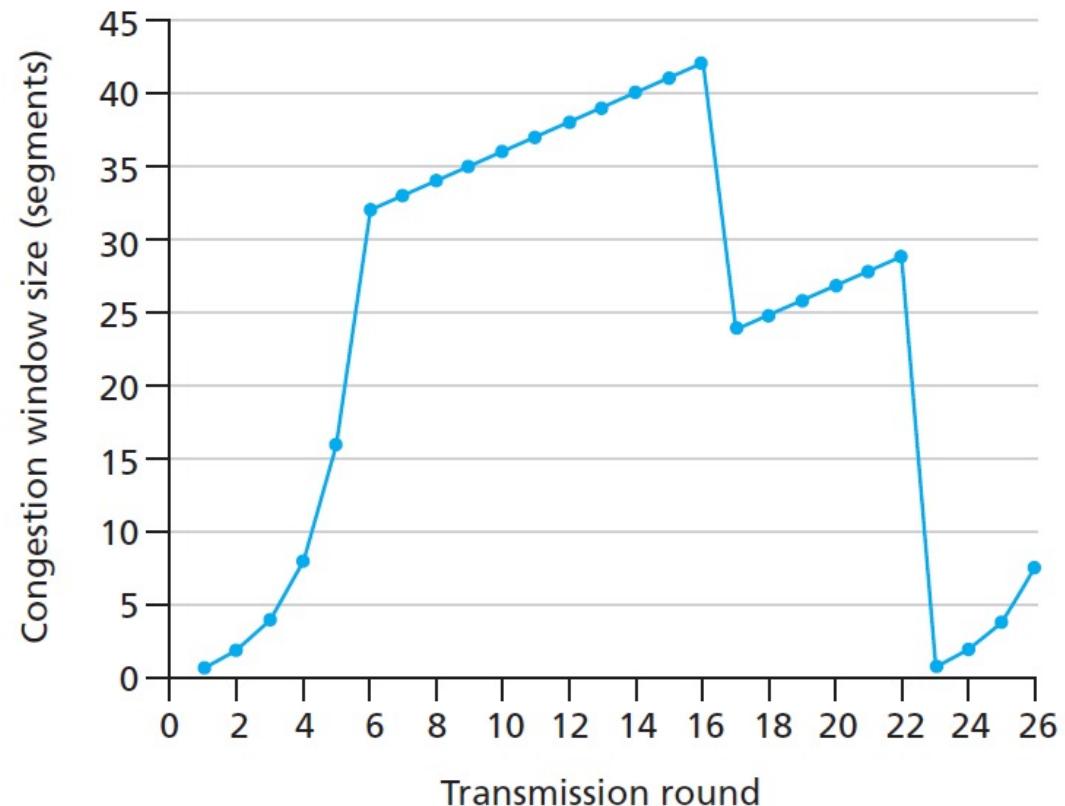
# Quiz: TCP Congestion Control?



In the figure what is the value of ssthresh (steady state threshold) at the 18<sup>th</sup> round?

- A. 1
- B. 32
- C. 42
- D. 21
- E. 20

**ANSWER: D**



# Transport Layer: 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”

COMP 3331/9331:  
Computer Networks and  
Applications

Week 7

Network Layer: Data Plane

Reading Guide: Chapter 4: Sections 4.1, 4.3

# Network Layer: outline

*Our goals:*

- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - forwarding versus routing
- instantiation, implementation in the Internet

# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

-- **Not Covered**

## 4.3 IP: Internet Protocol

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

## 4.4 Generalized forwarding and Software Defined Networking (SDN)

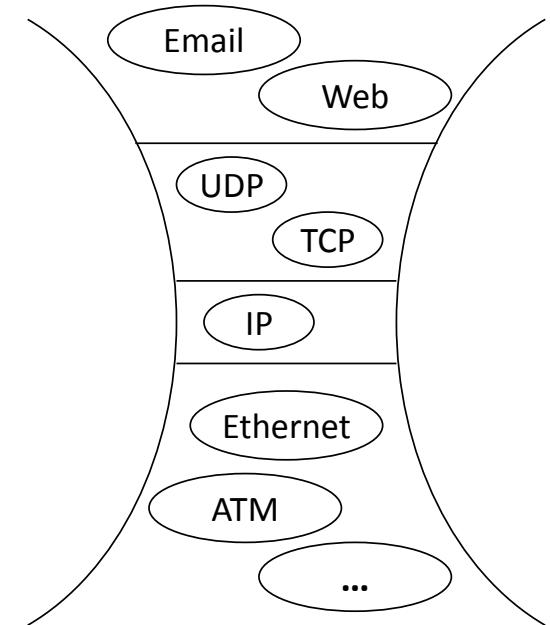
– **Not Covered**

# Some Background

- 1968: DARPAnet/ARPAnet (precursor to Internet)
  - (Defense) Advanced Research Projects Agency Network
- Mid 1970's: new networks emerge
  - SATNet, Packet Radio, Ethernet
  - All “islands” to themselves – didn't work together
- Big question: How to connect these networks?

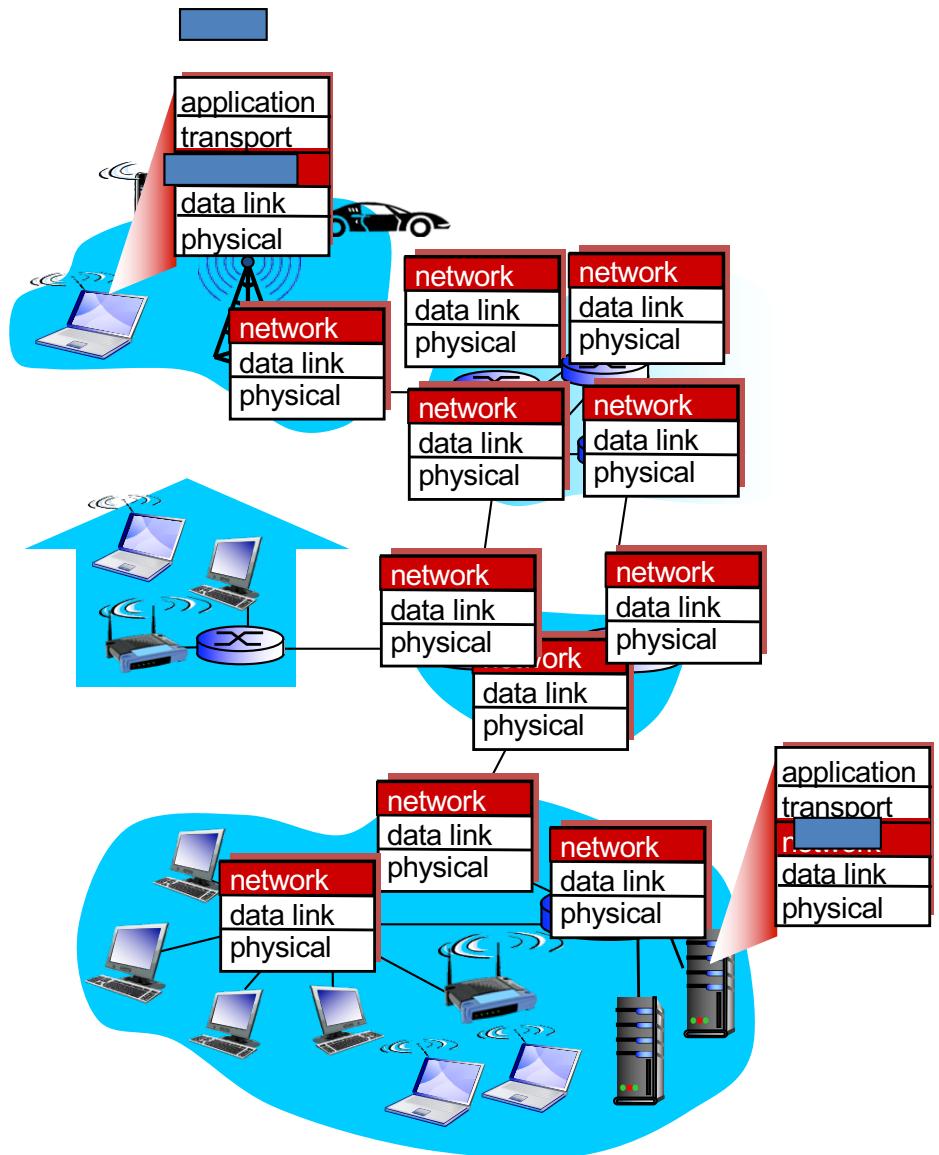
# Internetworking

- Cerf & Kahn in 1974,
  - “A Protocol for Packet Network Intercommunication”
  - Foundation for the modern Internet
- **Routers forward packets from source to destination**
  - May cross many separate networks along the way
- All packets use a common **Internet Protocol**
  - Any underlying data link protocol
  - Any higher layer transport protocol



# Network Layer

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



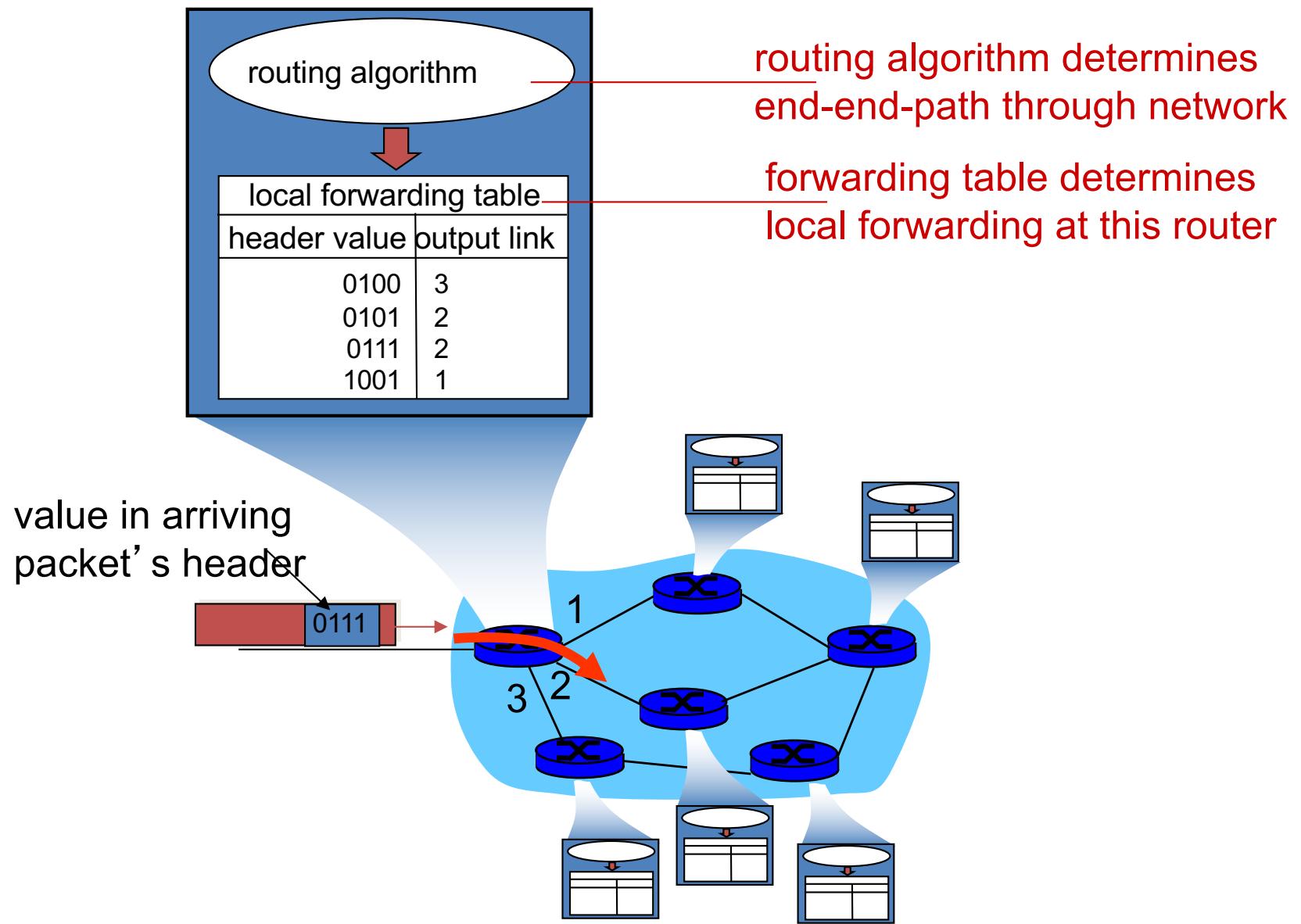
# Two key network-layer functions

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

*analogy:*

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

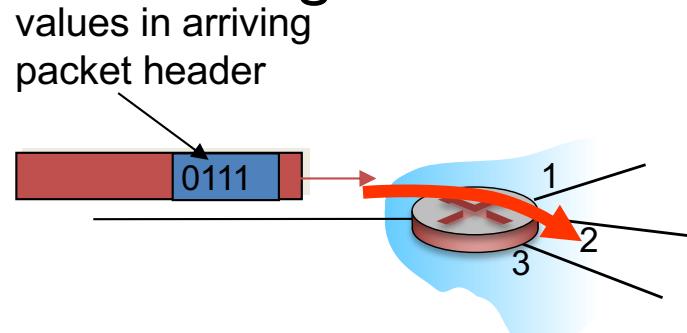
# Interplay between routing and forwarding



# Network Layer: data vs control plane

## *Data plane*

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

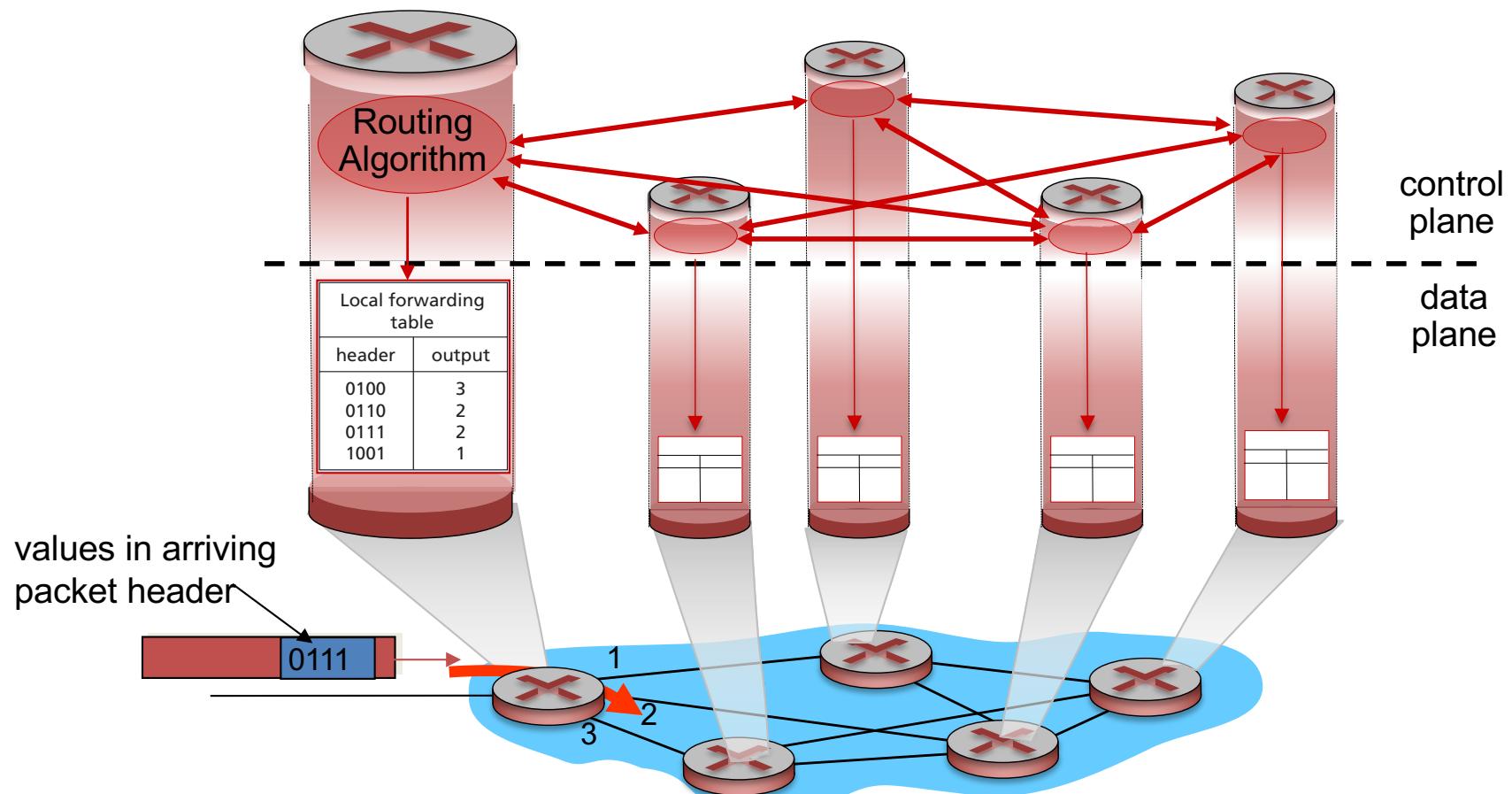


## *Control plane*

- 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
  - *software-defined networking (SDN)*: centralised (remote) servers

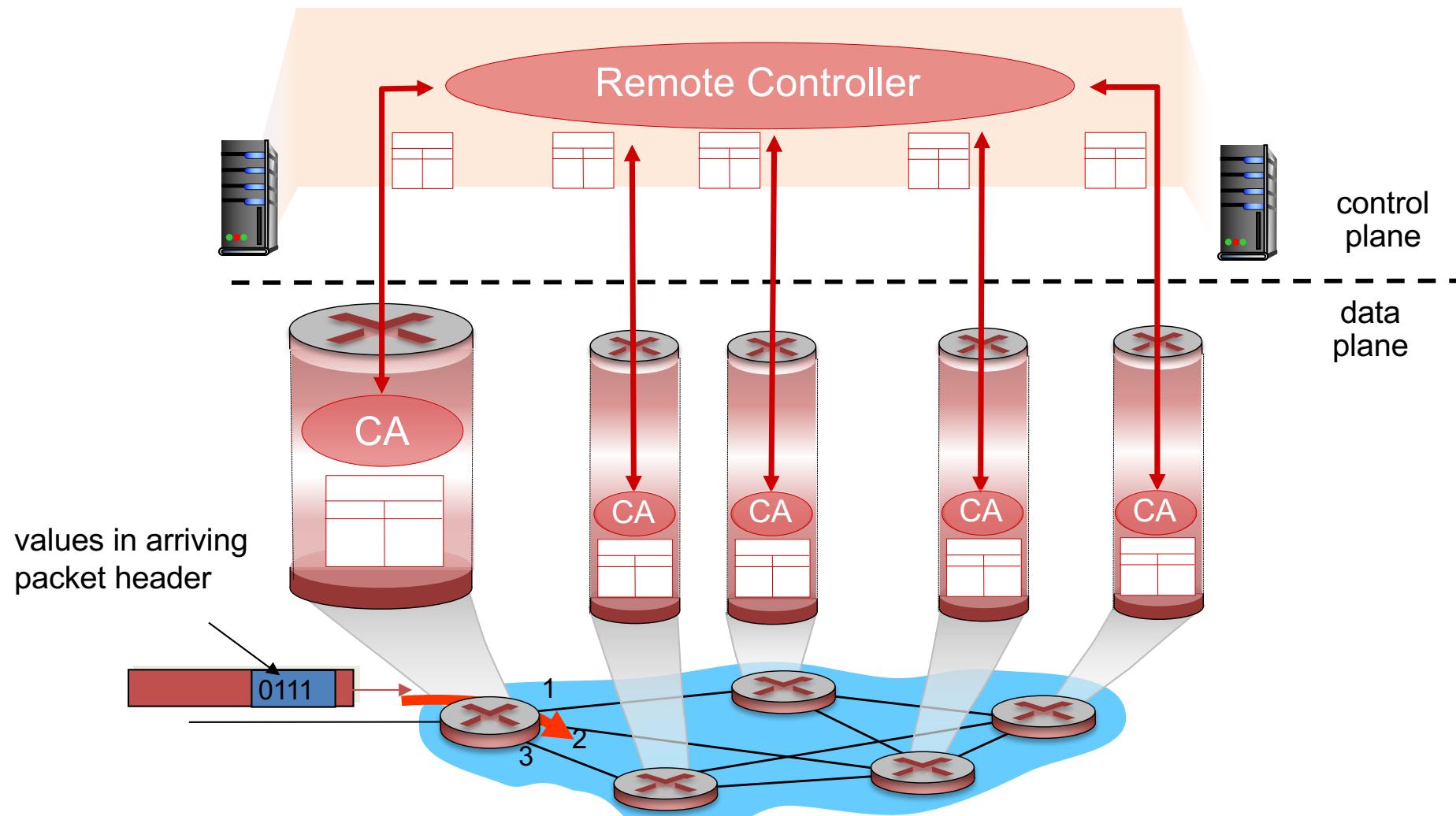
# Per-router control plane

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



# Logically centralized control plane (SDN)

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



# Network Layer: service model

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

- A. No guarantee whatsoever is provided by IP layer in TCP/IP protocol stack. It's “best effort service”.

# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

-- **Not Covered**

## 4.3 IP: Internet Protocol

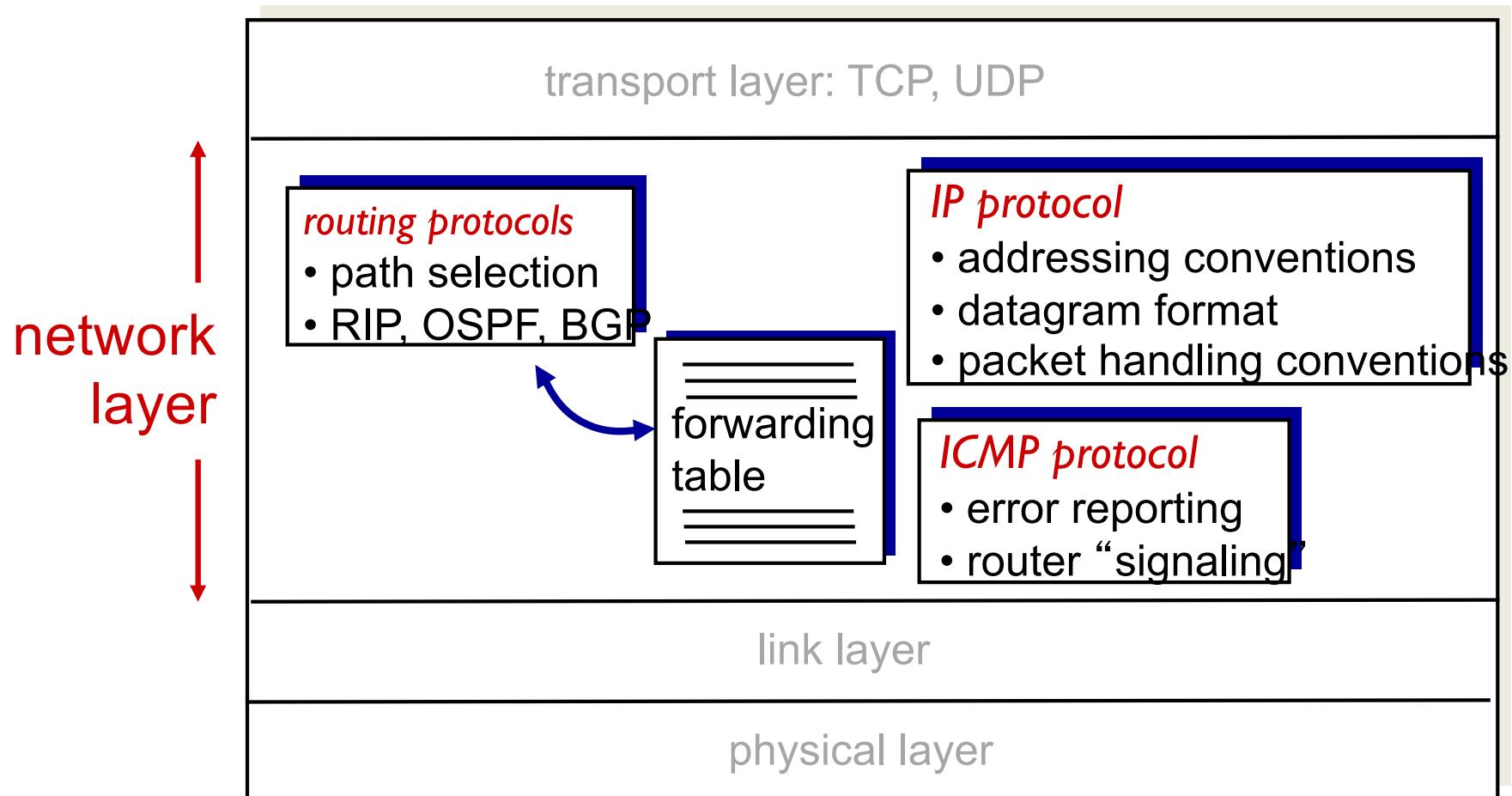
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

## 4.4 Generalized forwarding and Software Defined Networking (SDN)

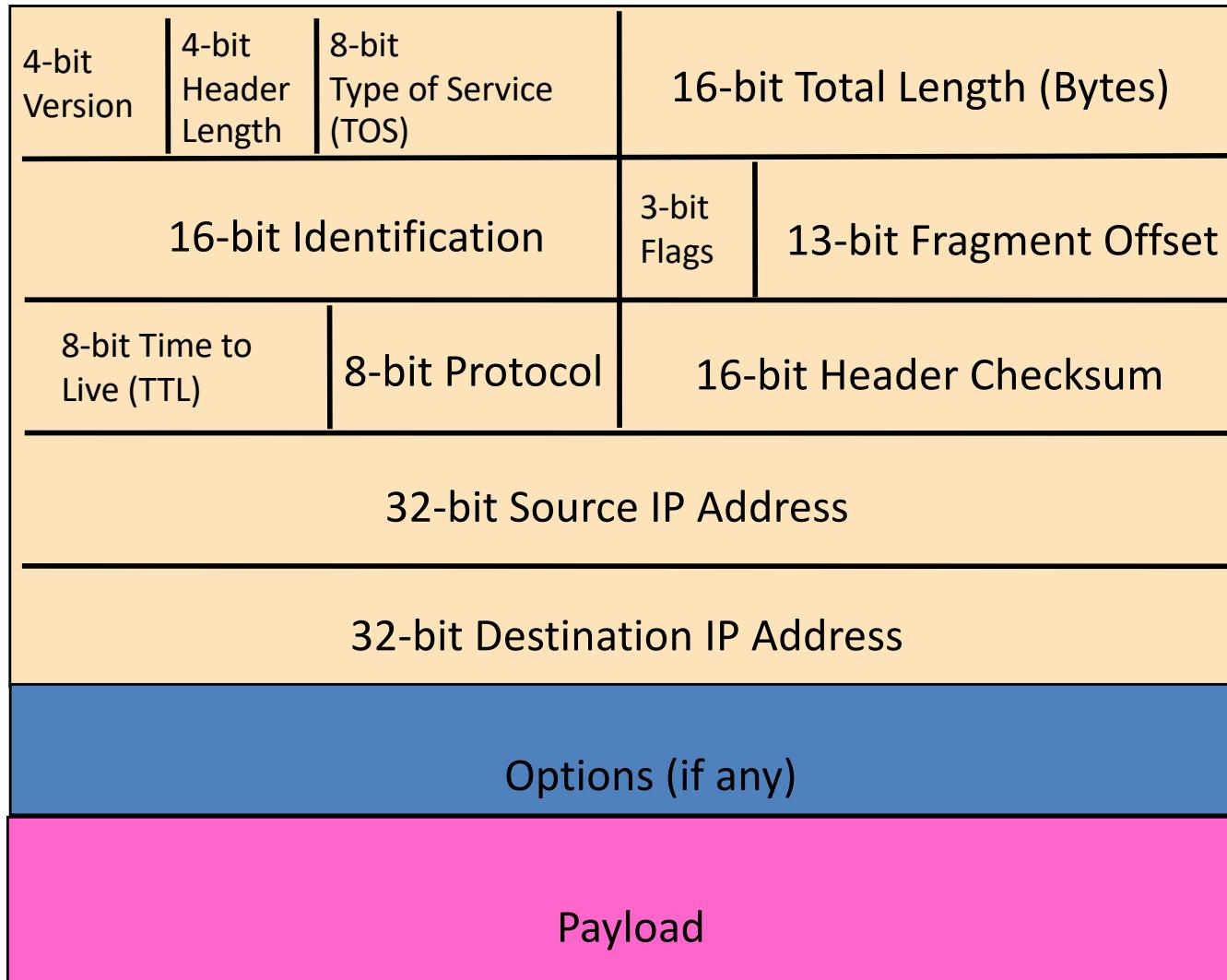
– **Not Covered**

# The Internet network layer

host, router network layer functions:

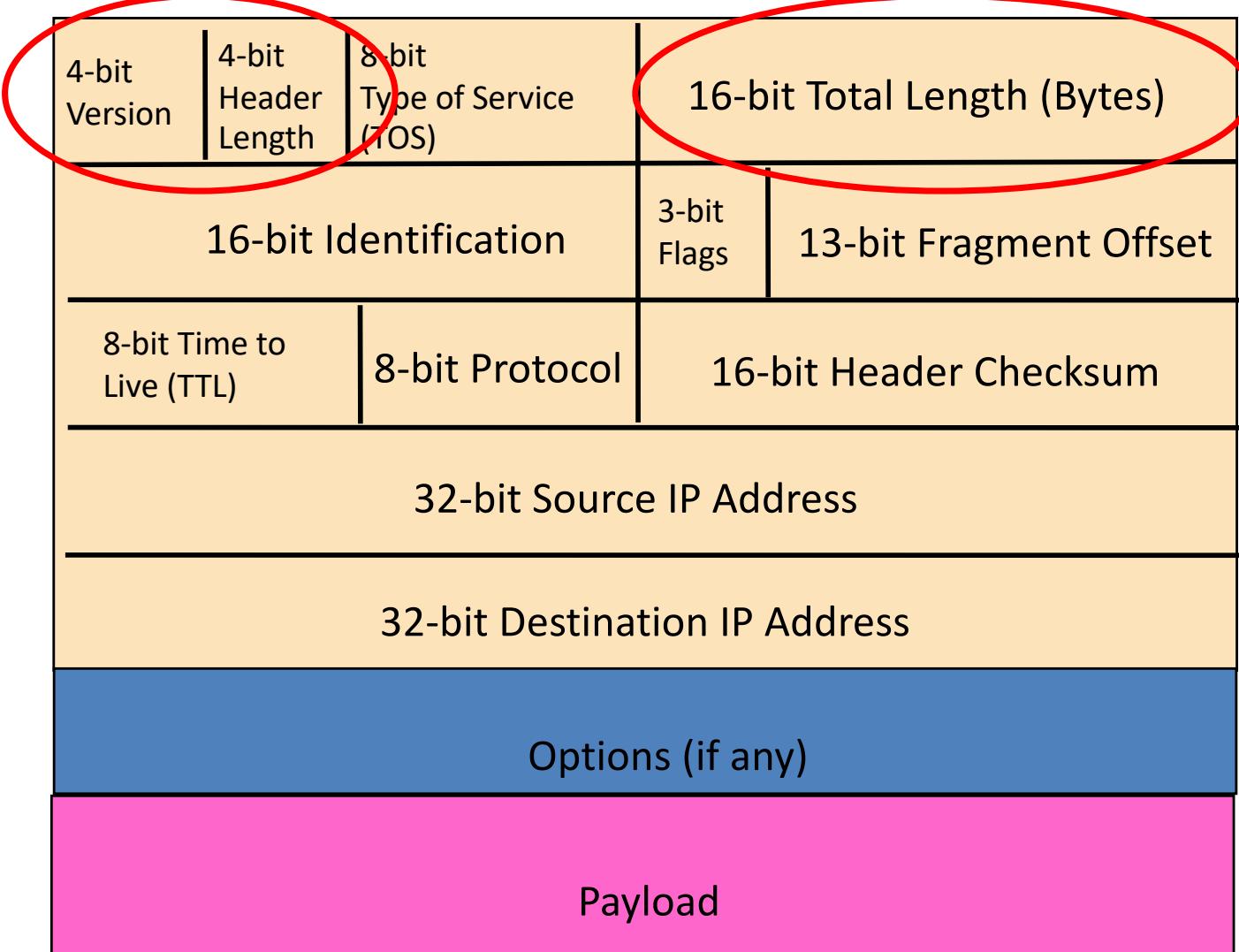


# IP Packet Structure



# Fields for Reading Packet Correctly

---



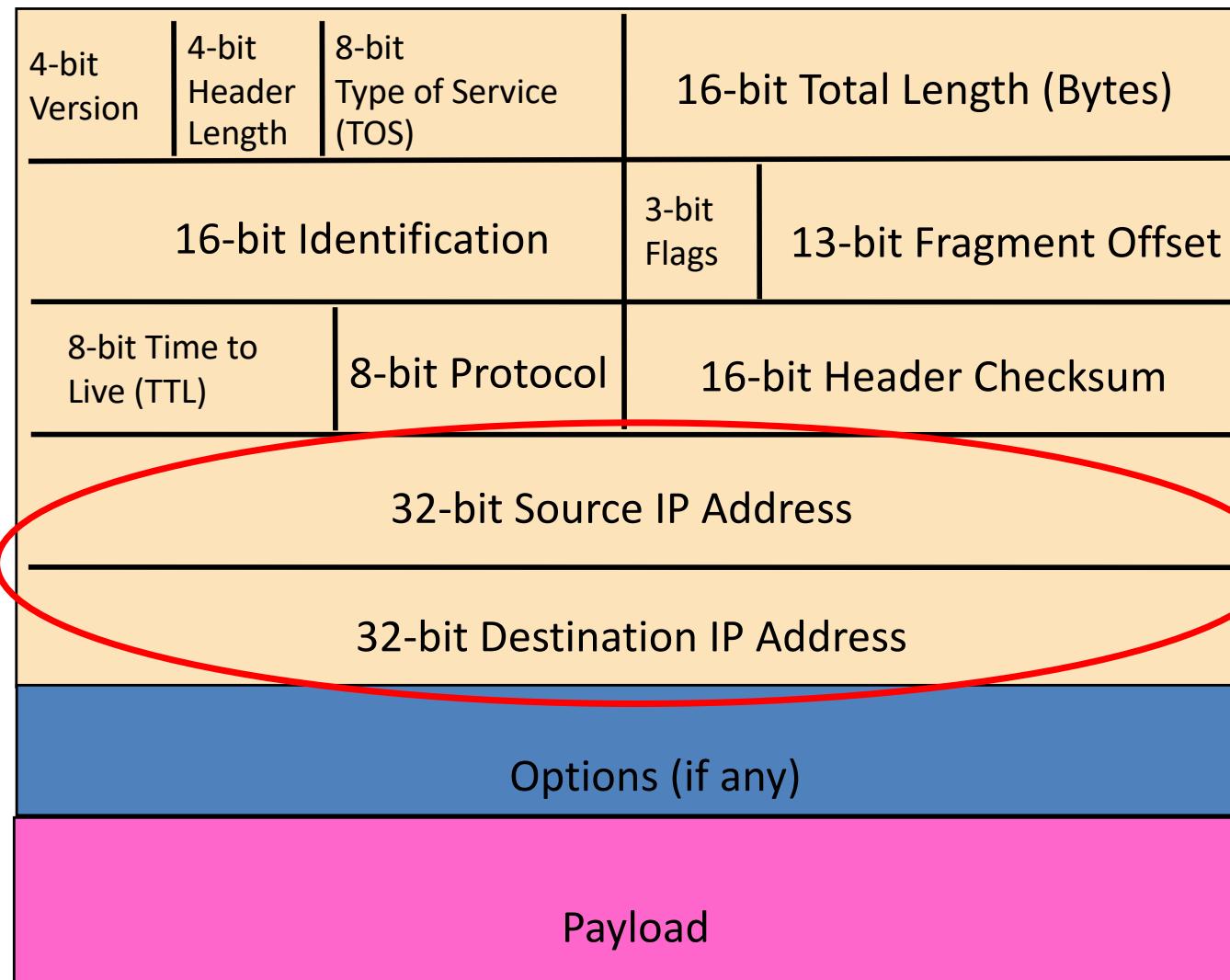
# Reading Packet Correctly

---

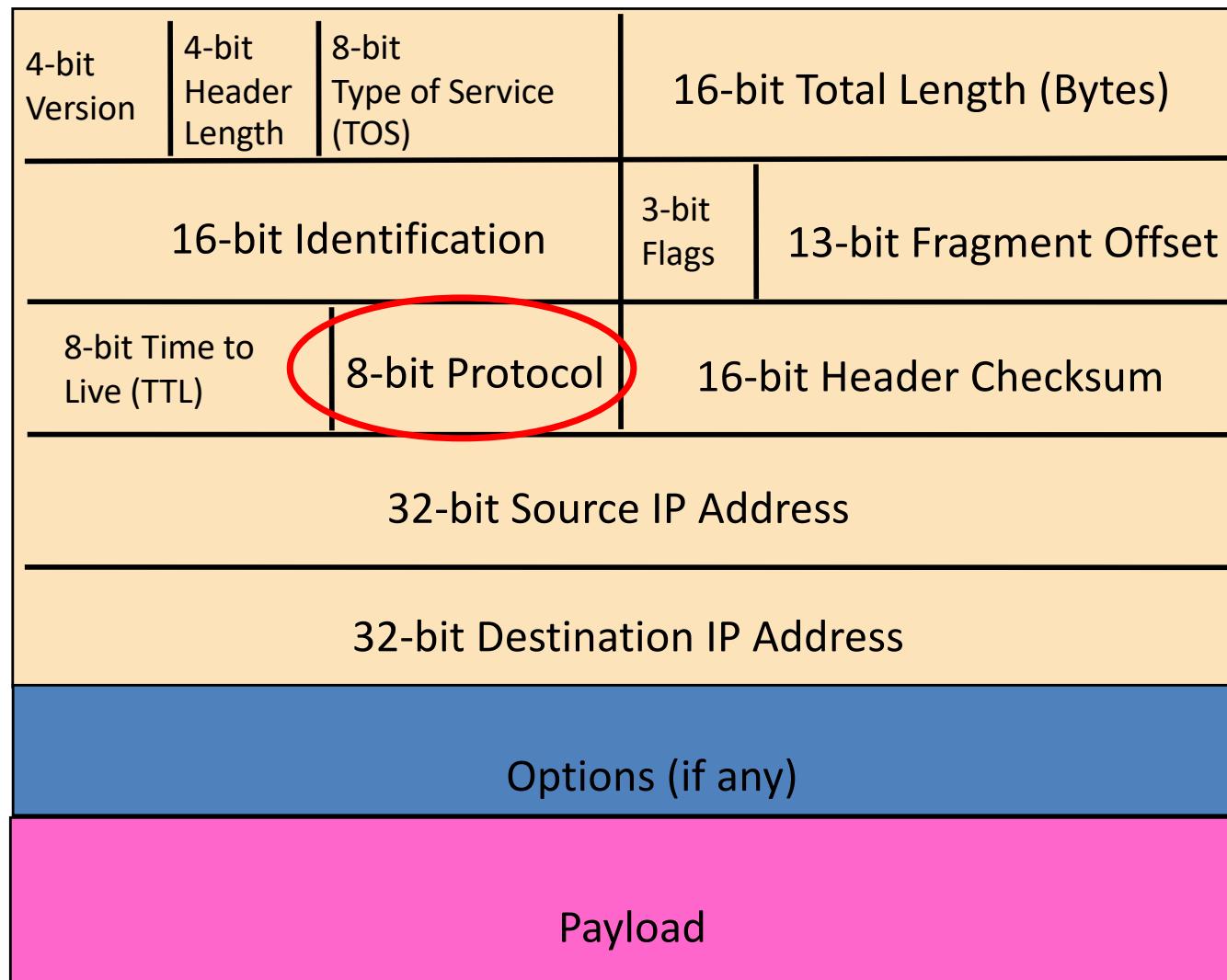
- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically, “4” (for IPv4)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically, “5” (for a 20-byte IPv4 header)
  - Can be more when IP **options** are used
- Total length (16 bits)
  - Number of bytes in the packet
  - Maximum size is 65,535 bytes ( $2^{16} - 1$ )
  - ... though underlying links may impose smaller limits

# Fields for Reaching Destination and Back

---



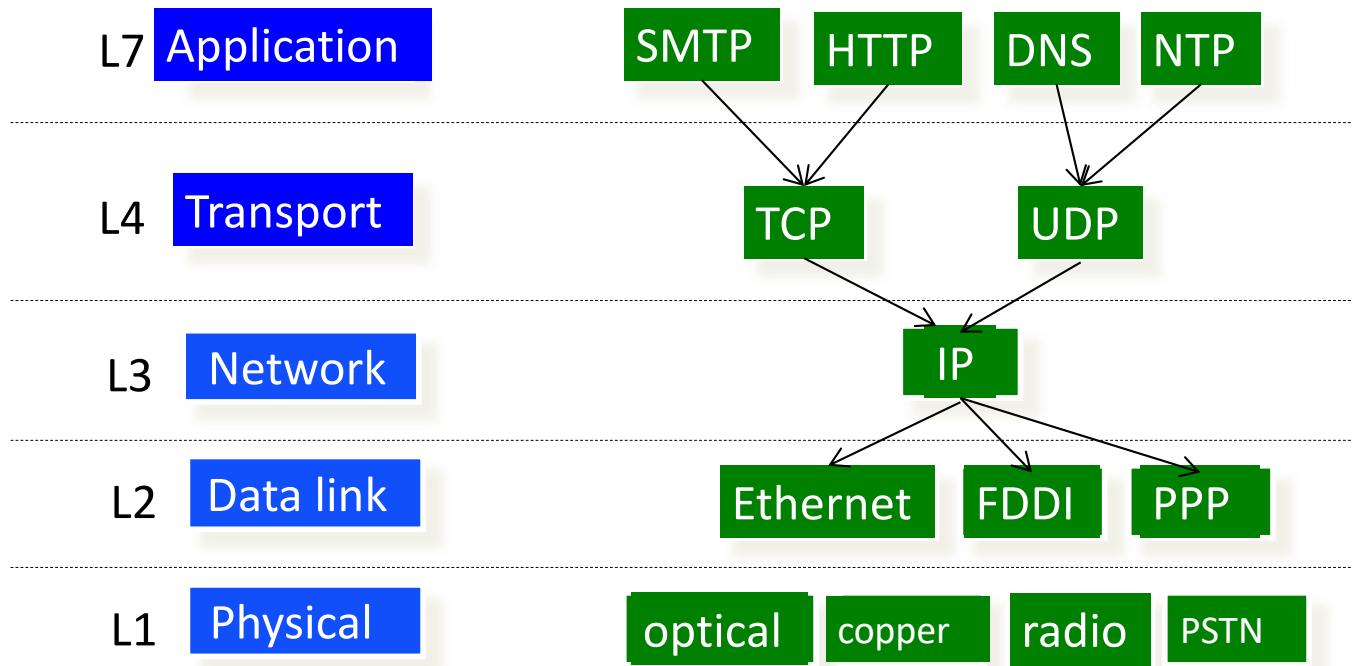
# Telling End-Host How to Handle Packet



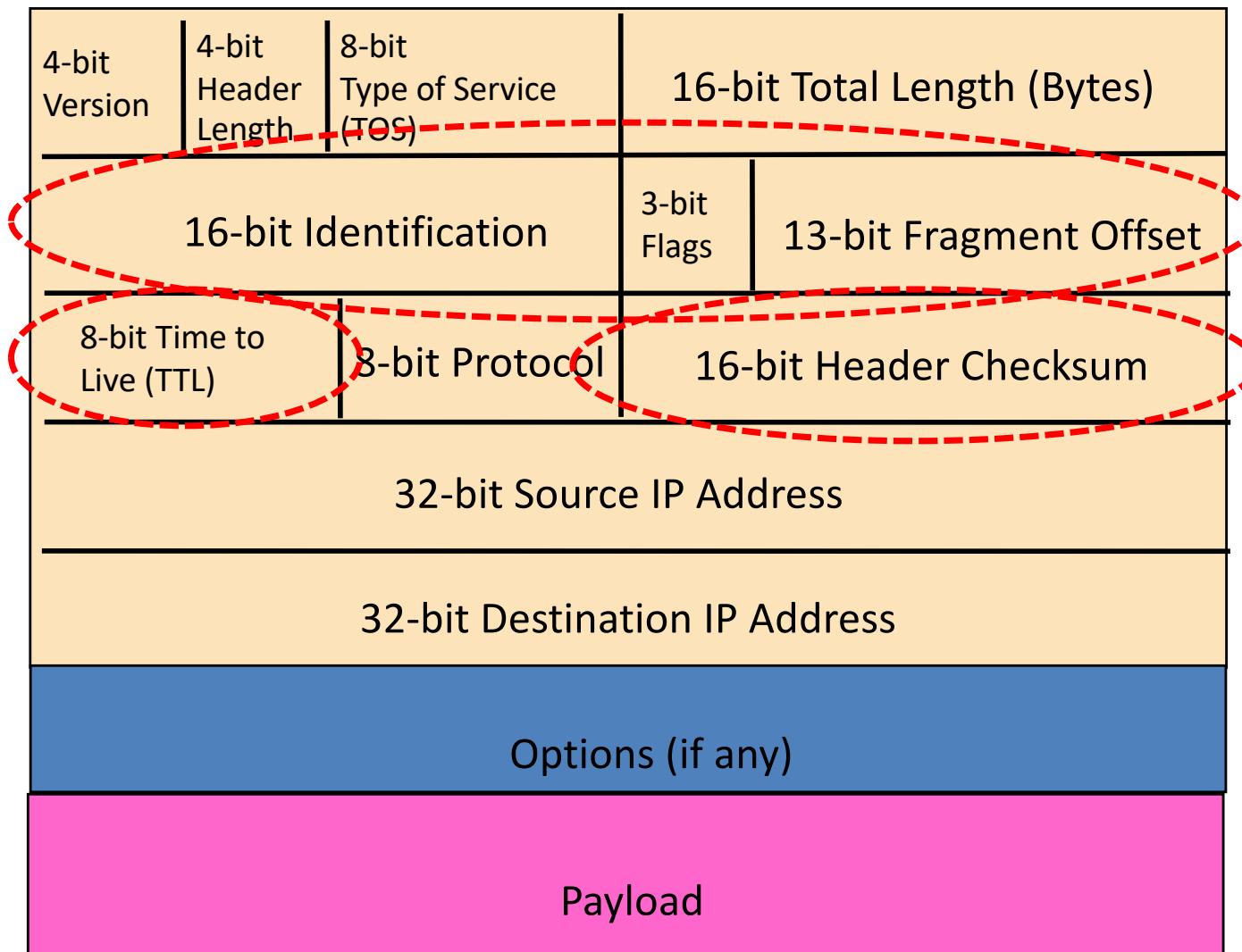
# Telling End-Host How to Handle Packet

---

- Protocol (8 bits)
  - Identifies the higher-level Transport protocol
  - Important for demultiplexing at receiving host



# Checksum, TTL and Fragmentation Fields

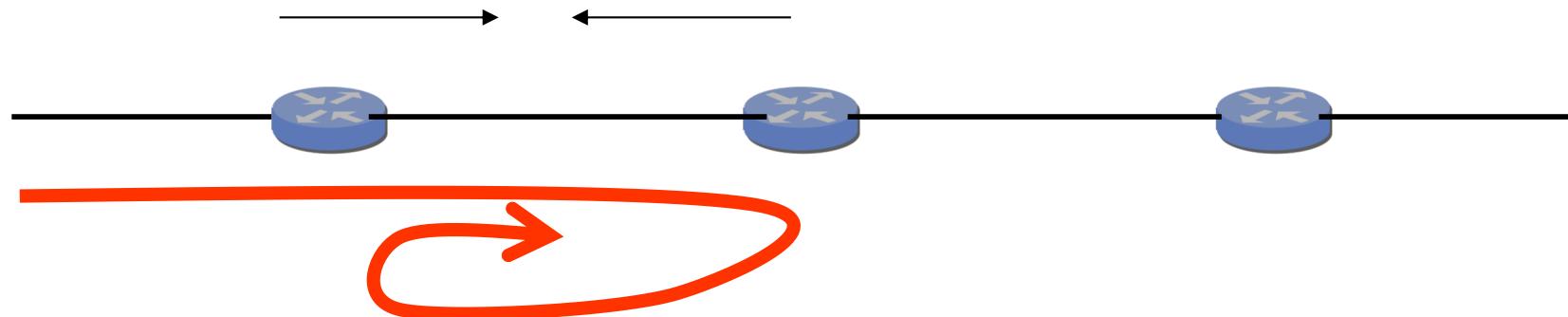


# Potential Problems

- Loop: **TTL**
- Header Corrupted: **Checksum**
- Packet too large: **Fragmentation**

# Preventing Loops (TTL)

- Forwarding loops cause packets to cycle for a long time
  - As these accumulate, eventually consume **all** capacity



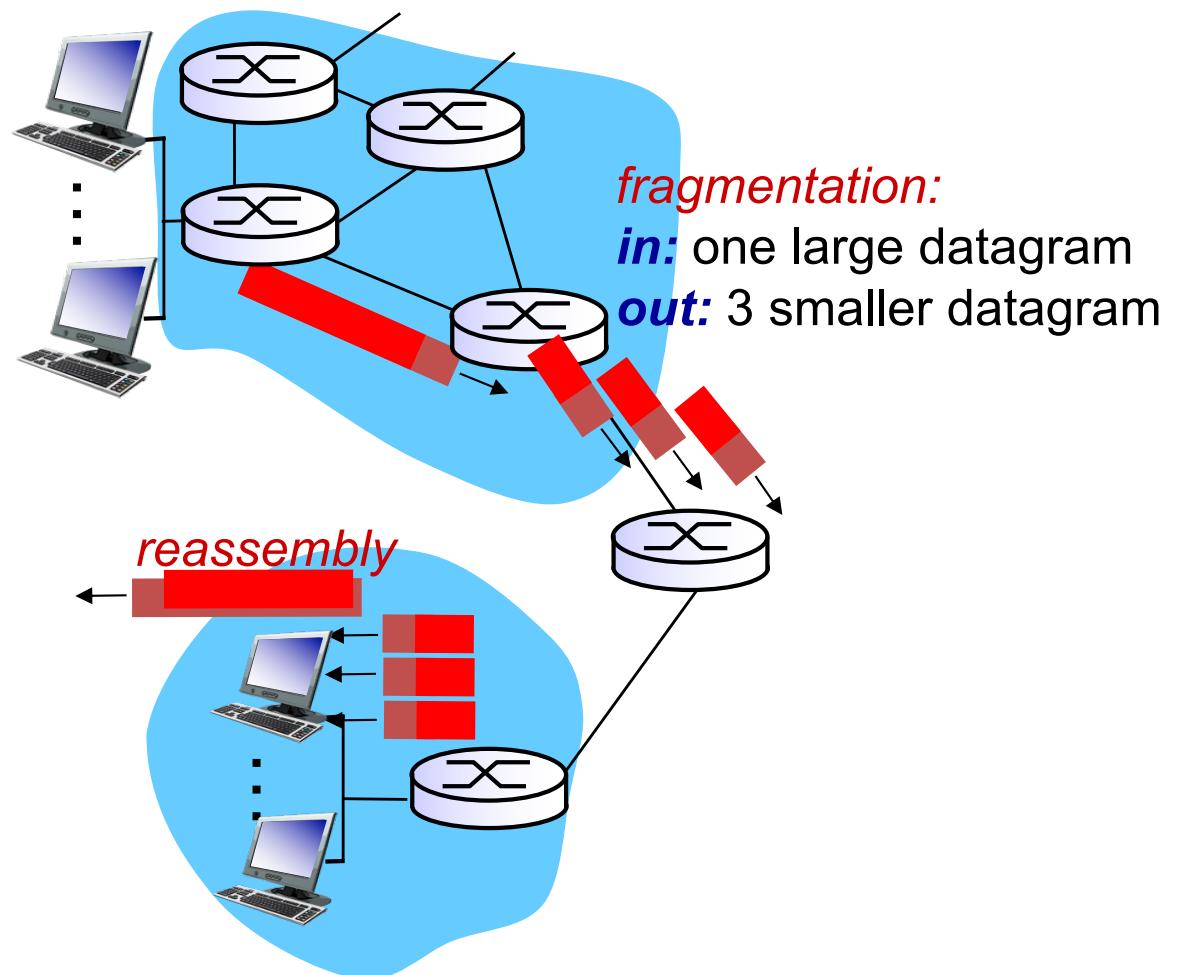
- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0
  - ...and “time exceeded” message is sent to the source
  - Recommended default value is 64

# Header Corruption (Checksum)

- Checksum (16 bits)
  - Particular form of checksum over packet header
- If not correct, router discards packets
  - So, it doesn't act on bogus information
- Checksum recalculated at every router
  - Why?
  - Why include TTL?
  - Why only header?

# 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

Note: Offset is expressed as multiple of 8 bytes

## example:

- ❖ 4000-byte datagram
- ❖ MTU = 1500 bytes

Data = 4000 –  
20 (IP header)  
=3980  
F1=1480  
F2=1480  
F3=1020

1480 bytes in  
data field

offset =  
 $1480/8$

	length =4000	ID =x	MF flag =0	offset =0	
--	-----------------	----------	---------------	--------------	--

*one large datagram becomes  
several smaller datagrams*

	length =1500	ID =x	MF flag =1	offset =0	
--	-----------------	----------	---------------	--------------	--

	length =1500	ID =x	MF flag =1	offset =185	
--	-----------------	----------	---------------	----------------	--

	length =1040	ID =x	MF flag =0	offset =370	
--	-----------------	----------	---------------	----------------	--

Applet:

[http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_2/applets/ip/ipfragmentation.html](http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/ip/ipfragmentation.html)

# IPv4 fragmentation procedure

## ➤ Fragmentation

- Router breaks up datagram in size that output link can support
- Copies IP header to pieces
- Adjust length on pieces
- Set offset to indicate position
- Set MF (More fragments) flag on pieces except the last
- Re-compute checksum

## ➤ Re-assembly

- Receiving host uses identification field with MF and offsets to complete the datagram.

## ➤ Fragmentation of fragments also supported

Taken from [TCP/IP  
Protocol Suite by  
Behroze Forouzan]

			4,020
14,567	0	000	

Bytes 0000–3,999

Original datagram

			1,420
14,567	1	000	

Bytes 0000–1,399

Fragment 1

			1,420
14,567	1	175	

Bytes 1,400–2,799

Fragment 2

			820
14,567	1	175	

Bytes 1,400–2,199

Fragment 2.1

			620
14,567	1	275	

Bytes 2,200–2,799

Fragment 2.2

MTU=820

			1,220
14,567	0	350	

Bytes 2,800–3,999

Fragment 3

MTU=1420

# IP Fragmentation Attacks ...

[https://en.wikipedia.org/wiki/IP\\_fragmentation\\_attack](https://en.wikipedia.org/wiki/IP_fragmentation_attack)

IP fragmentation exploits [\[edit\]](#)

**IP fragment overlapped** [\[edit\]](#)

The IP fragment overlapped [exploit](#) occurs when two fragments contained within the same IP datagram have offsets that indicate that they overlap each other in positioning within the datagram. This could mean that either fragment A is being completely overwritten by fragment B, or that fragment A is partially being overwritten by fragment B. Some operating systems do not properly handle fragments that overlap in this manner and may throw exceptions or behave in other undesirable ways upon receipt of overlapping fragments. This is the basis for the [teardrop Denial of service](#) attacks.

**IP fragmentation buffer full** [\[edit\]](#)

The IP fragmentation buffer full exploit occurs when there is an excessive amount of incomplete fragmented traffic detected on the protected network. This could be due to an excessive number of incomplete fragmented datagrams, a large number of fragments for individual datagrams or a combination of quantity of incomplete datagrams and size/number of fragments in each datagram. This type of traffic is most likely an attempt to bypass security measures or [Intrusion Detection Systems](#) by intentional fragmentation of attack activity.

**IP fragment overrun** [\[edit\]](#)

The IP Fragment Overrun exploit is when a reassembled fragmented datagram exceeds the declared IP data length or the maximum datagram length. By definition, no IP datagram should be larger than 65,535 bytes. Systems that try to process these large datagrams can crash, and can be indicative of a denial of service attempt.

**IP fragment overwrite** [\[edit\]](#)

Overlapping fragments may be used in an attempt to bypass Intrusion Detection Systems. In this exploit, part of an attack is sent in fragments along with additional random data; future fragments may overwrite the random data with the remainder of the attack. If the completed datagram is not properly reassembled at the IDS, the attack will go undetected.

**IP fragment too many datagrams** [\[edit\]](#)

The Too Many Datagrams exploit is identified by an excessive number of incomplete fragmented datagrams detected on the network. This is usually either a denial of service attack or an attempt to bypass security measures. An example of "Too Many Datagrams", "Incomplete Datagram" and "Fragment Too Small" is the Rose Attack.<sup>[1]</sup>

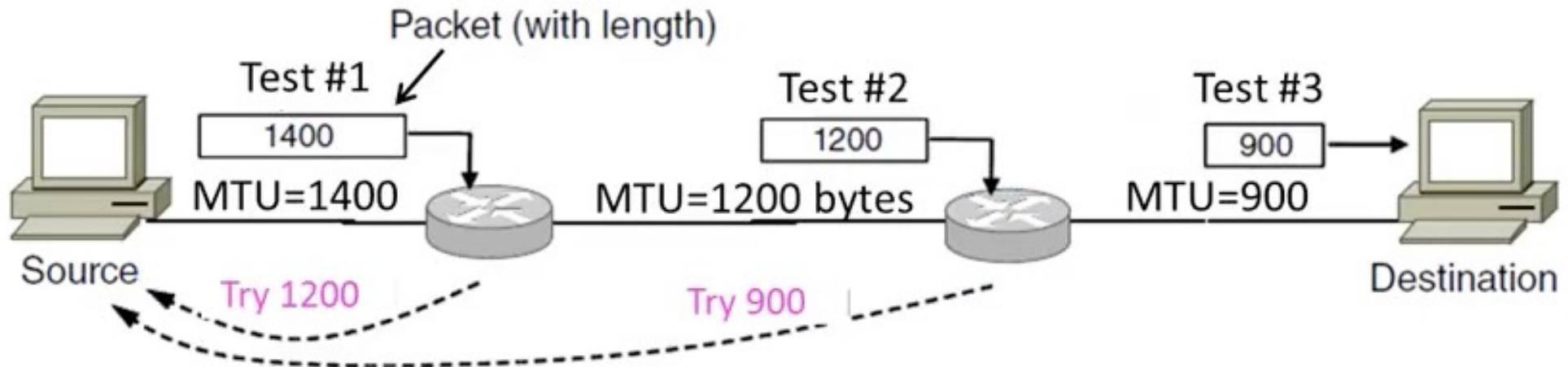
**IP fragment incomplete datagram** [\[edit\]](#)

This exploit occurs when a datagram can not be fully reassembled due to missing data. This can indicate a denial of service attack or an attempt to defeat packet filter security policies.

**IP fragment too small** [\[edit\]](#)

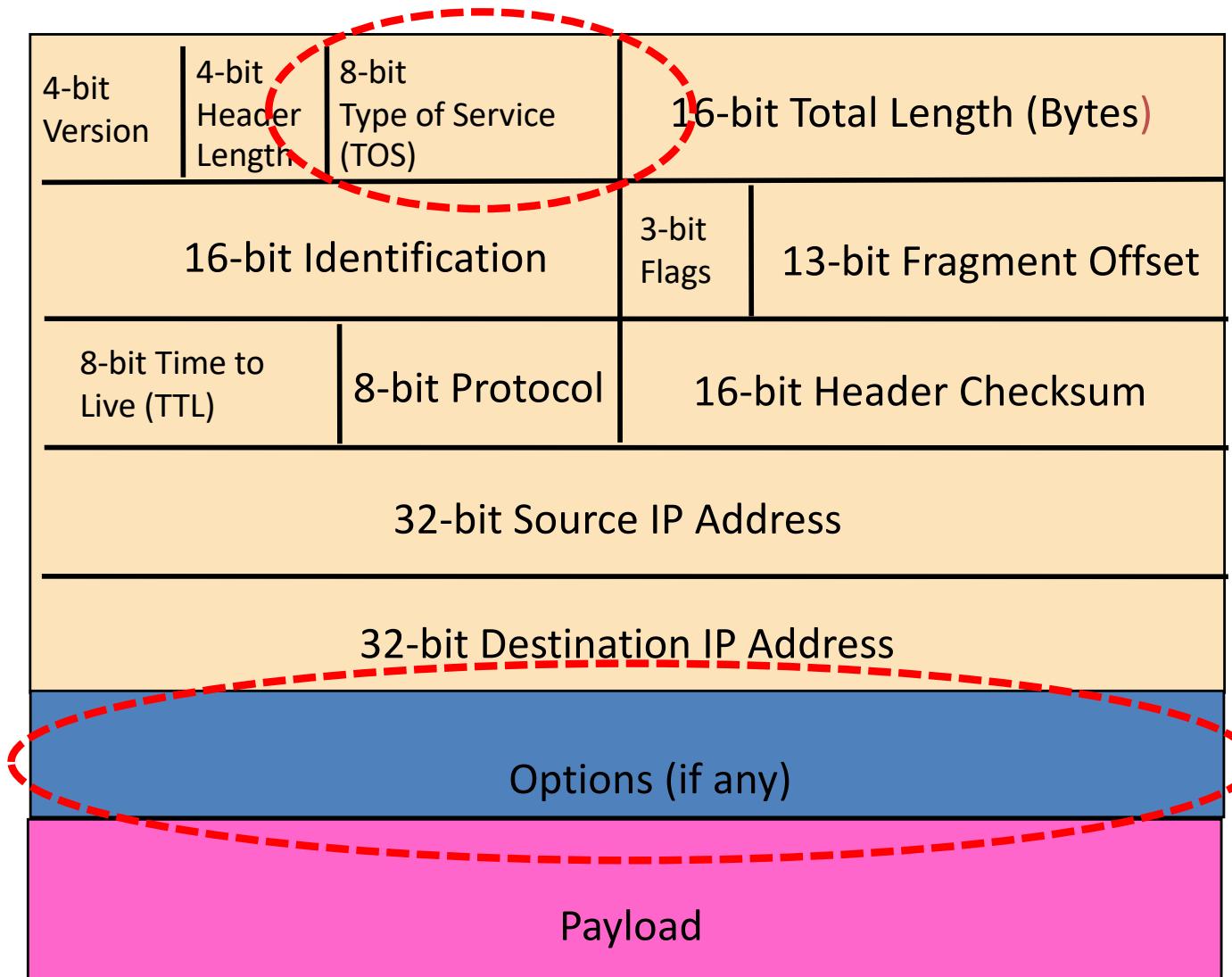
An IP Fragment Too Small exploit is when any fragment other than the final fragment is less than 400 bytes, indicating that the fragment is likely intentionally crafted. Small fragments may be used in denial of service attacks or in an attempt to bypass security measures or detection.

# Path MTU Discovery procedure



- Host
  - Sends a big packet to test whether all routers in path to the destination can support or not
  - Set DF (Do not fragment) flag
- Routers
  - Drops the packet if it is too large (as DF is set)
  - Provides feedback to Host with ICMP message telling the maximum supported size

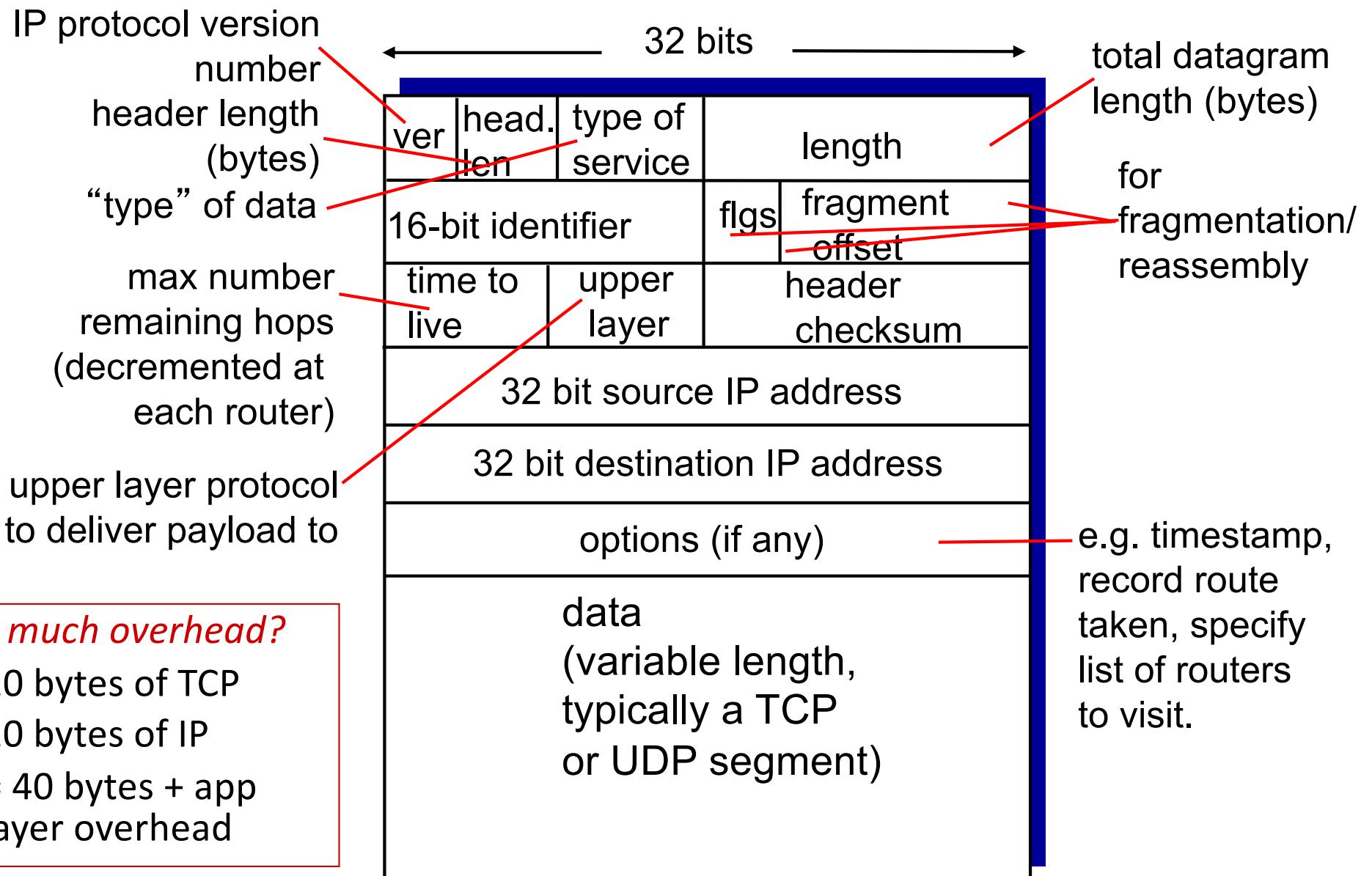
# Fields for Special Handling



# Special Handling

- “Type of Service”, or “Differentiated Services Code Point (DSCP)” (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer
  - Has been redefined several times
  - Not widely used
- Options (not often used)

# RECAP: IP datagram format



# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

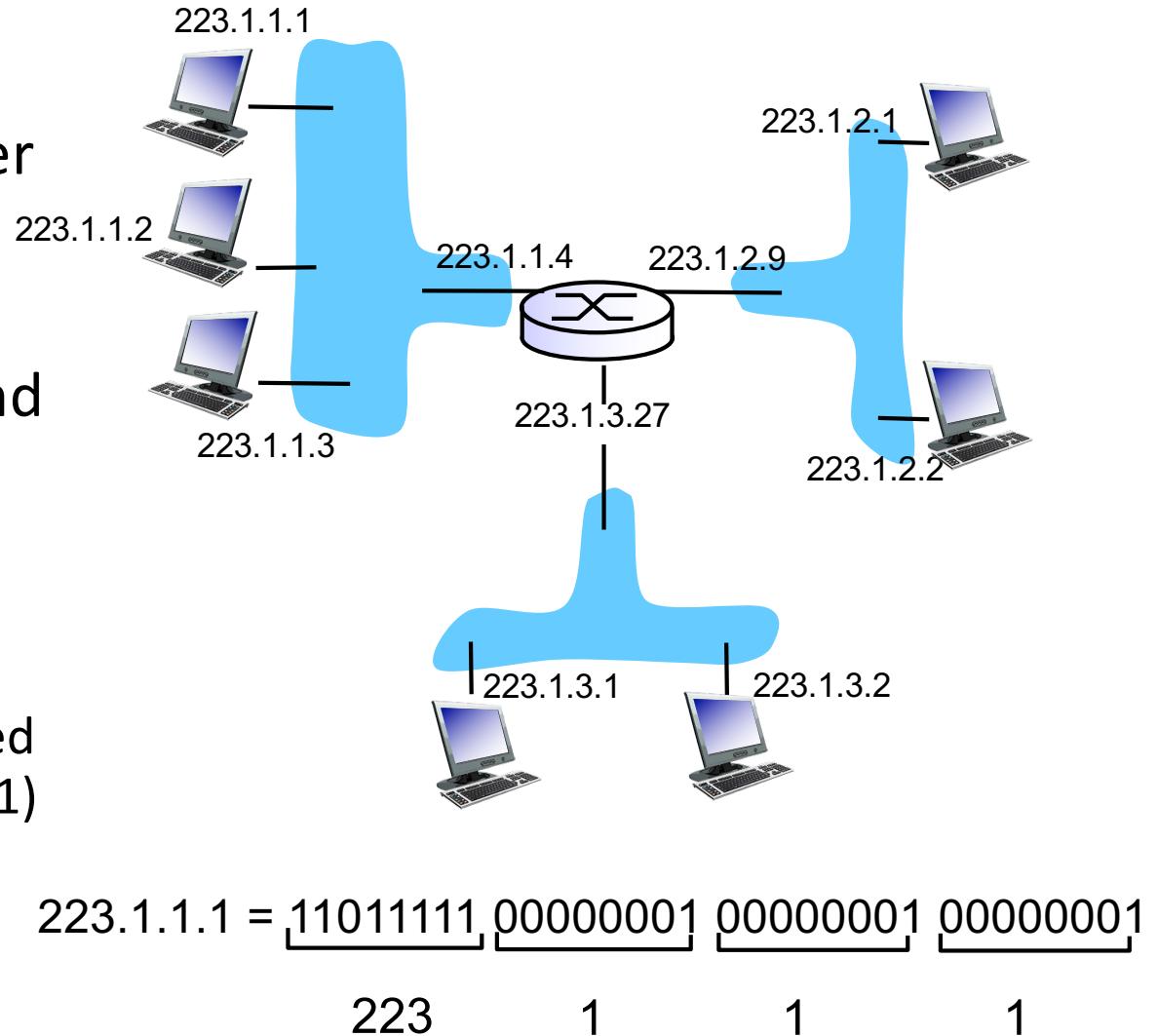
## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

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

# IP addressing: introduction

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

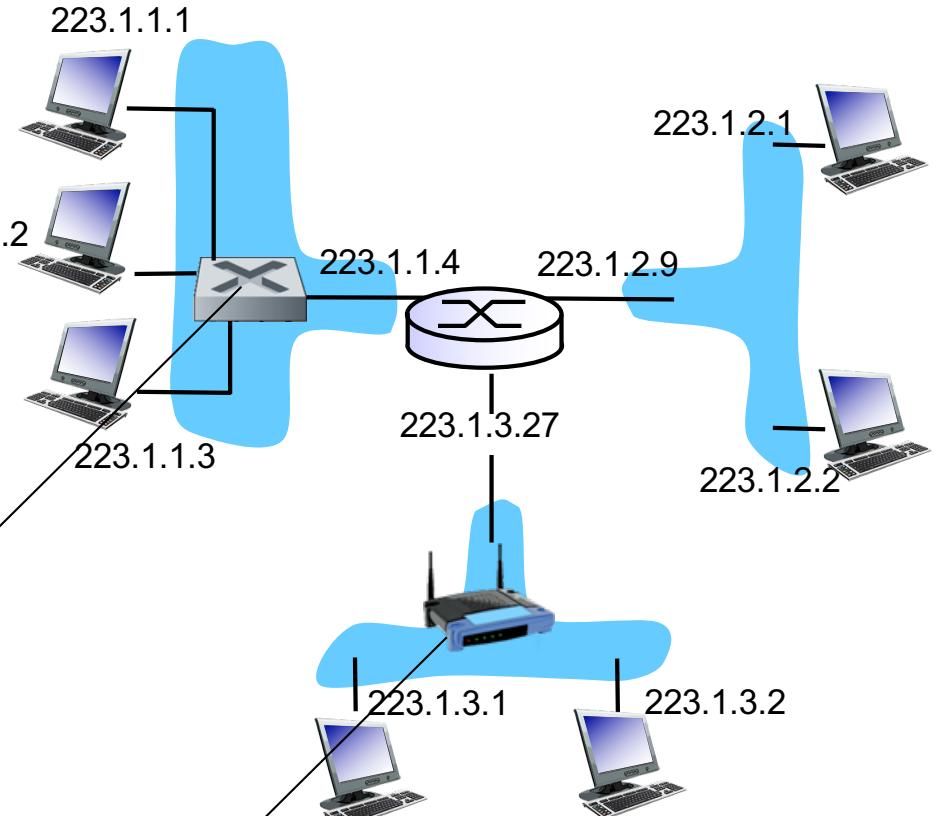


# IP addressing: introduction

*Q: how are interfaces actually connected?*

*A: we'll learn about that in the link layer*

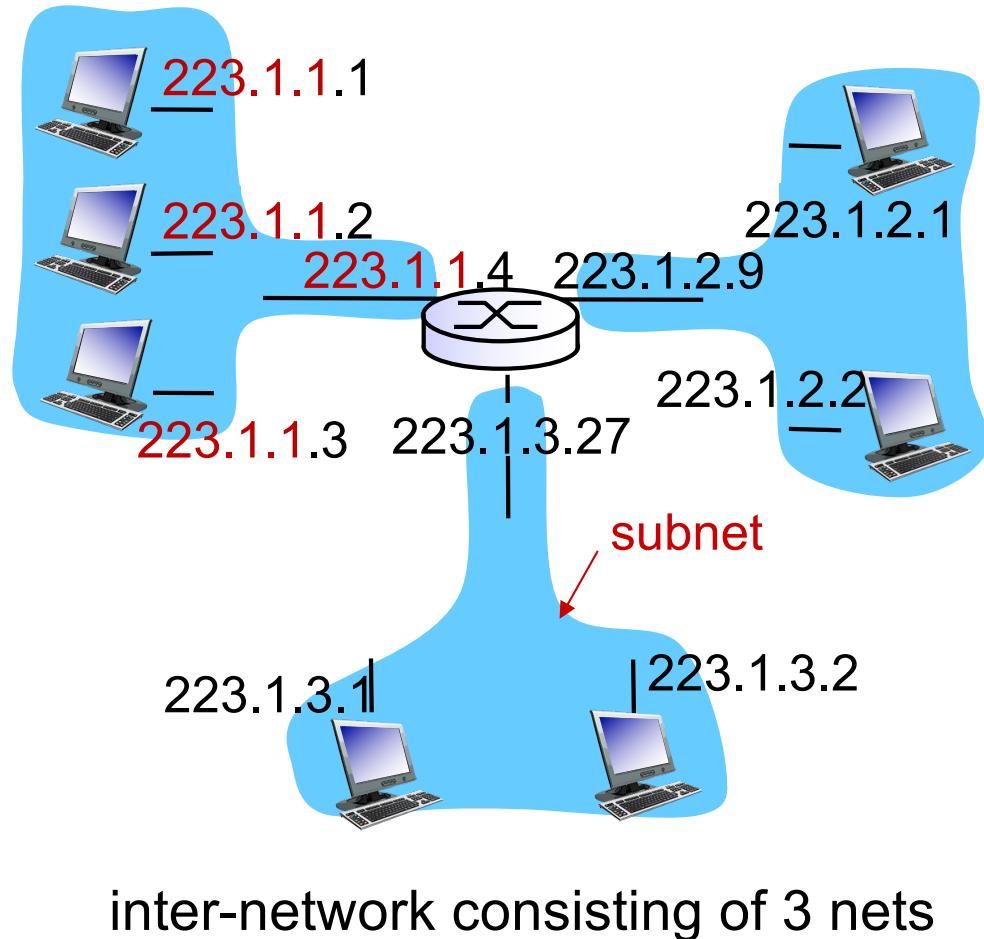
*A: wired Ethernet interfaces connected by Ethernet switches*



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

# Networks

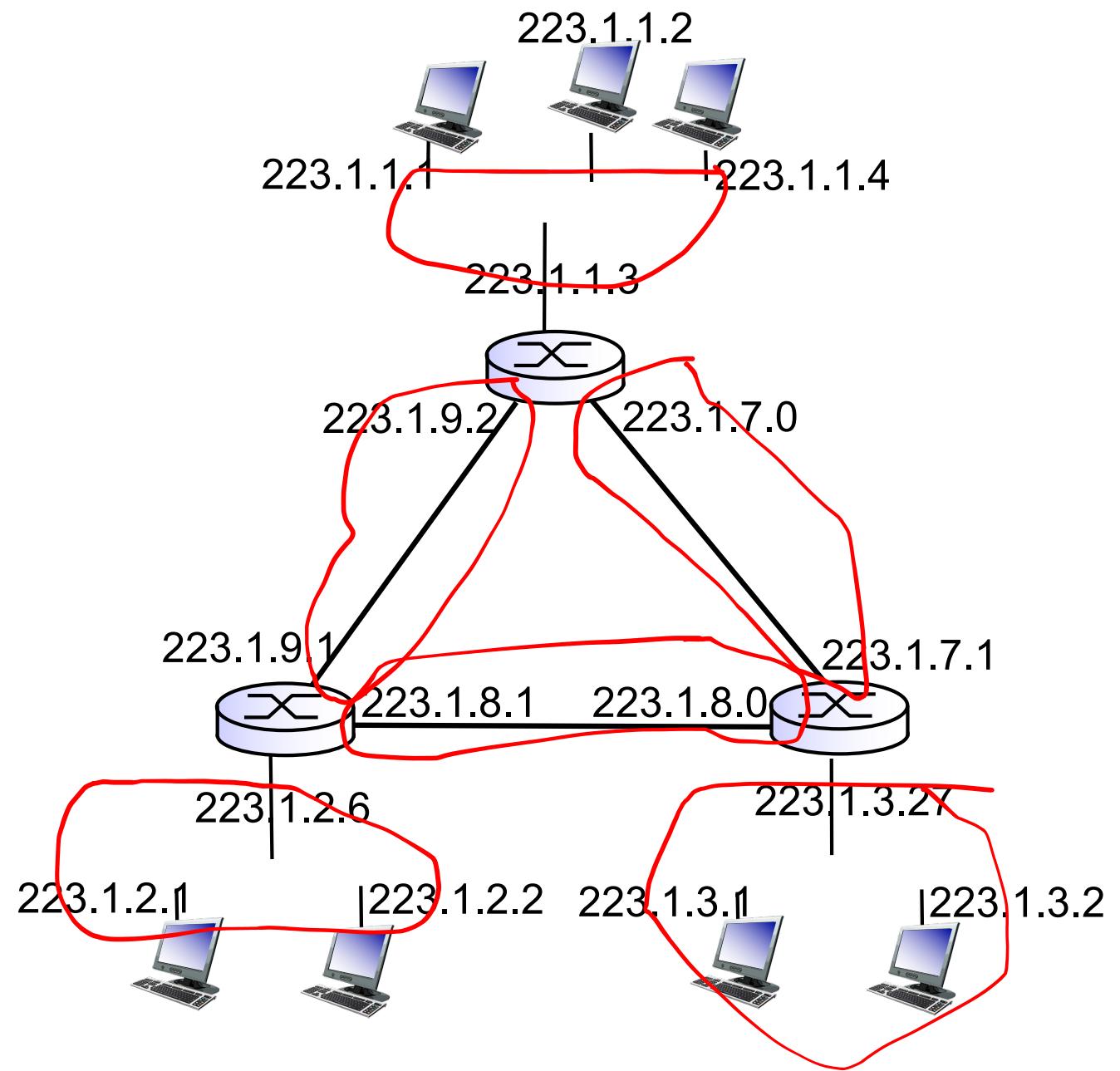
- IP address:
  - network part - high order bits
  - host part - low order bits
- *what's a network ?*
  - device interfaces with same network part of IP address
  - can physically reach each other *without intervening router*



# Networks

how many?

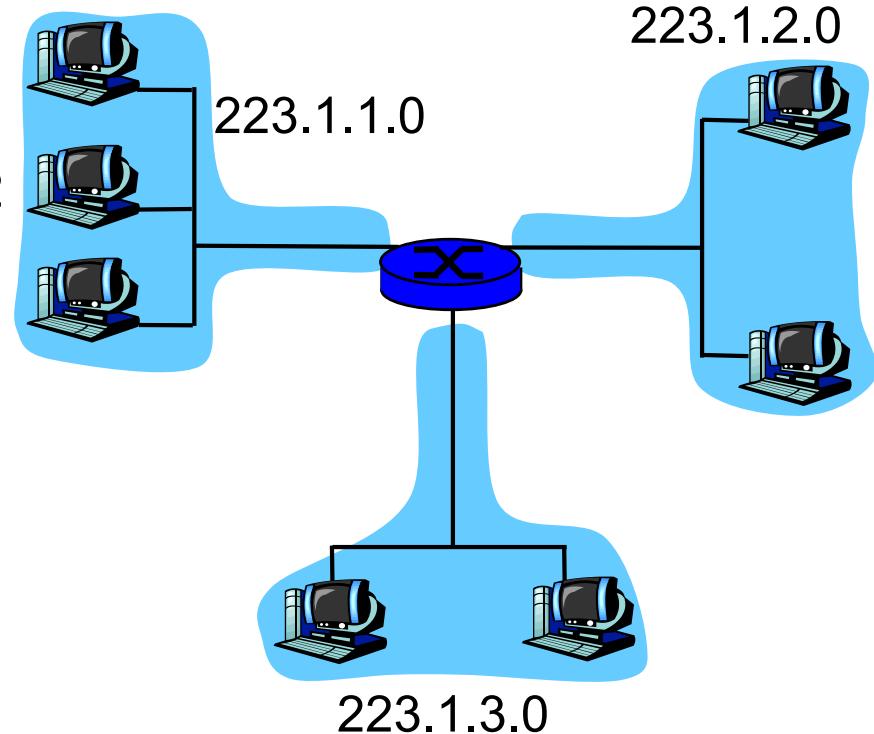
Answer: 6 as shown



# Masking

- Mask
  - Used in conjunction with the network address to indicate how many higher order bits are used for the network part of the address
    - Bit-wise AND
  - 223.1.1.0 with mask 255.255.255.0
- Broadcast Address
  - host part is all 111's
  - E.g., 223.1.1.255
- Network Address
  - Host part is all 0000's
  - E.g., 223.1.1.0
- Both are typically not assigned to any host

B: 223.1.1.2



Host B	Dot-decimal address	Binary
IP address	223.1.1.2	11111101.00000001.00000001.00000010
Mask	255.255.255.0	11111111.11111111.11111111.00000000
Network Part	223.1.1.0	11111101.00000001.00000001.00000000
Host Part	0.0.0.2	00000000.00000000.00000000.00000010

# Original Internet Addresses

- First eight bits: network address (/8)
- Last 24 bits: host address, ~16.7 million

*Assumed 256 networks were more than enough!*

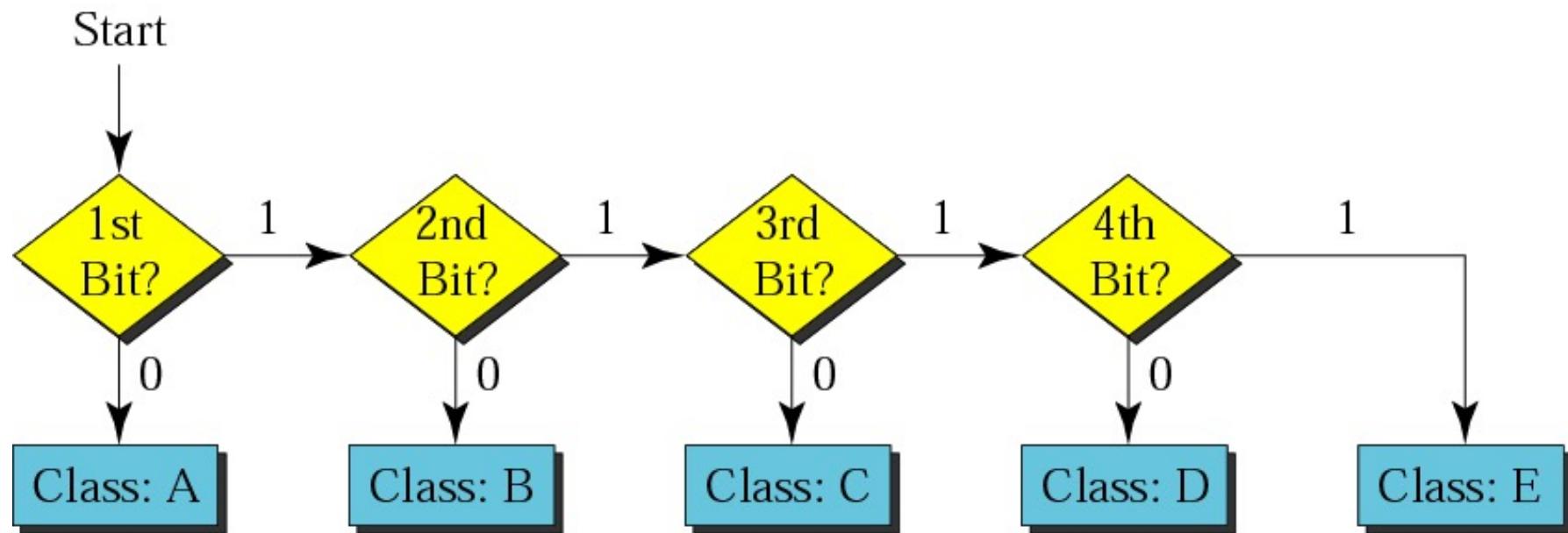
# Next design: Class-ful Addresses

Used till the introduction of CIDR 1993

	0	8	16	24	31		
Class A	0	<i>netid</i>		<i>hostid</i>		$2^7$ nets, $2^{24}$ hosts	1.0.0.0 to 127.255.255.255
Class B	10	<i>netid</i>		<i>hostid</i>		$2^{14}$ nets, $2^{16}$ hosts	128.0.0.0 to 191.255.255.255
Class C	110	<i>netid</i>		<i>hostid</i>		$2^{21}$ nets, $2^8$ hosts	192.0.0.0 to 223.255.255.255
Class D	1110		<i>multicast address</i>				224.0.0.0 to 239.255.255.255
Class E	1111		<i>reserved for future use</i>				240.0.0.0 to 255.255.255.255

Problem: Networks only come in three sizes!

# Finding the address class



# What are the issues?

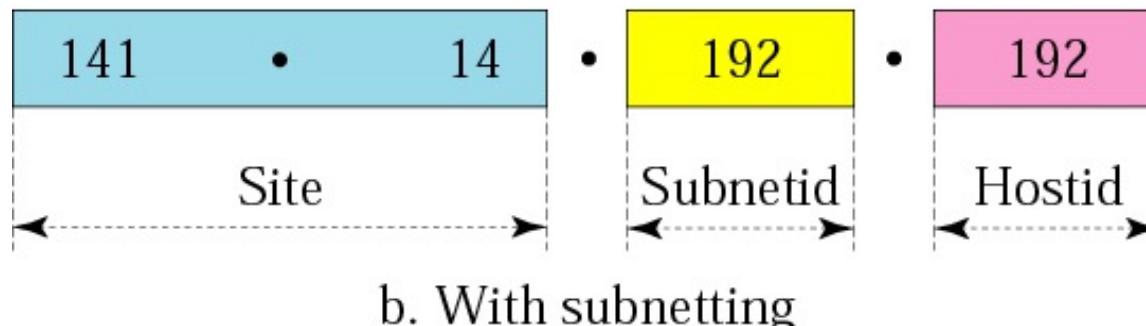
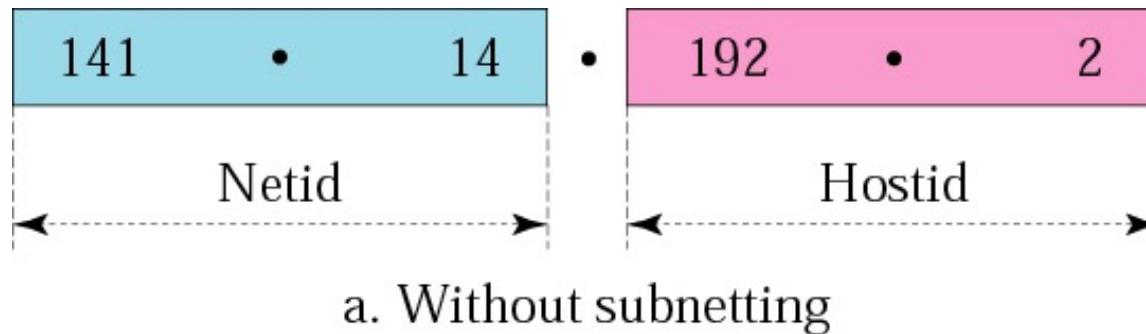
- An organization requires 6 nets each of size 30.  
Does it have to buy 6 class C address blocks?
  
- An organization requires 512 addresses? How  
many IP addresses should it buy?

# Subnetting

- Subnetting is the process of dividing the class A, B or C network into more manageable chunks that are suited to your network's size and structure.
- Subnetting allows 3 levels of hierarchy
  - netid, subnetid, hostid
- Original netid remains the same and designates the site
- Subnetting remains transparent outside the site

# Subnetting

- The process of subnetting simply extends the point where the 1's of Mask stop and 0's start
- You are sacrificing some host ID bits to gain Network ID bits



# Quiz?

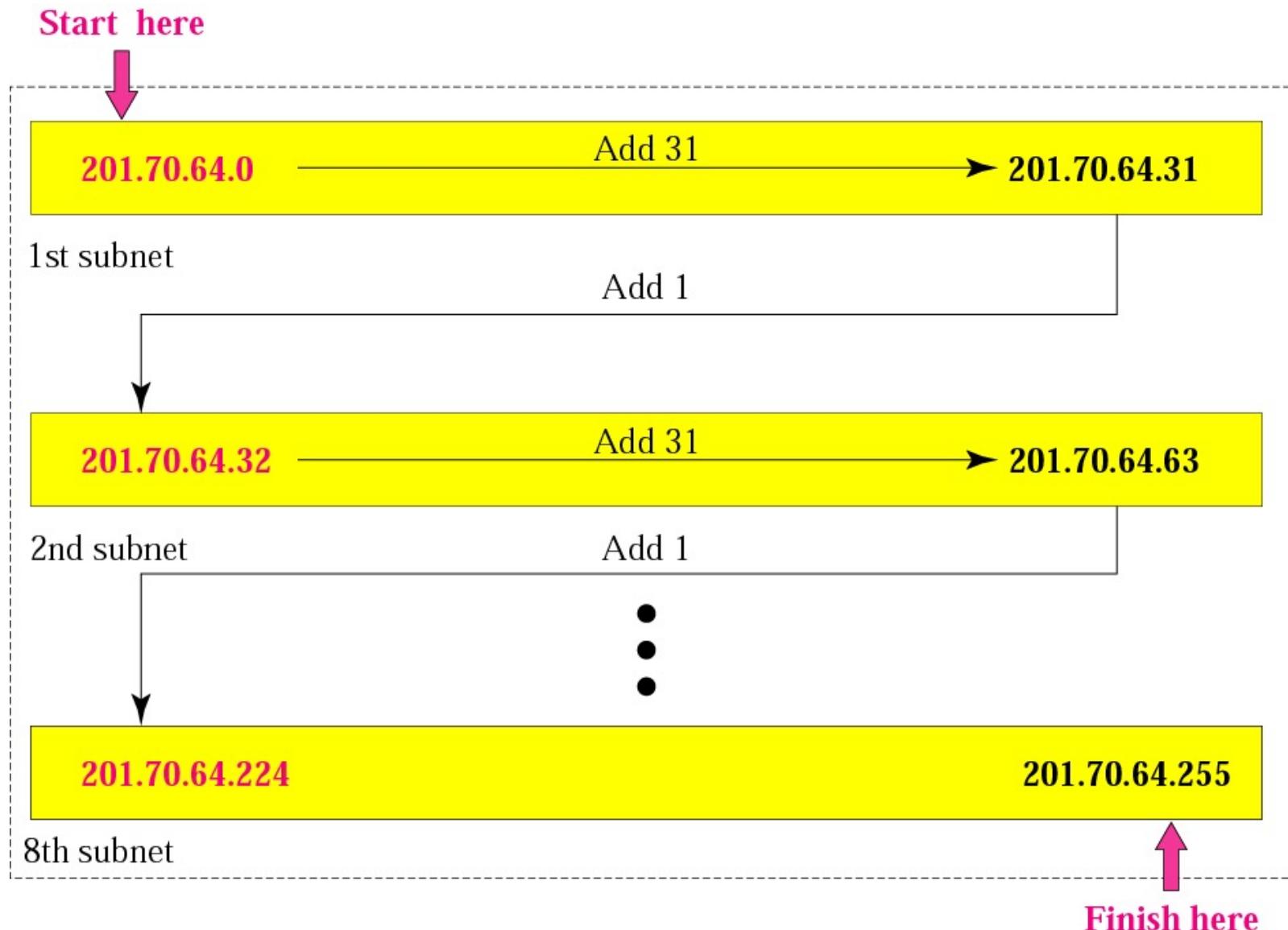
A company is granted the site address 201.70.64.0 (class C). The company needs six subnets. Design the subnets.

The company needs six subnets. 6 is not a power of 2. The next number that is a power of 2 is 8 ( $2^3$ ). We need 3 more 1s in the subnet mask. The total number of 1s in the subnet mask is 27 ( $24 + 3$ ). The mask is

11111111 11111111 11111111 11100000  
or 255.255.255.224

Number of addresses in each subnet =  $2^5$   
= 32

The number of addresses in each subnet is  $2^5$  or 32.

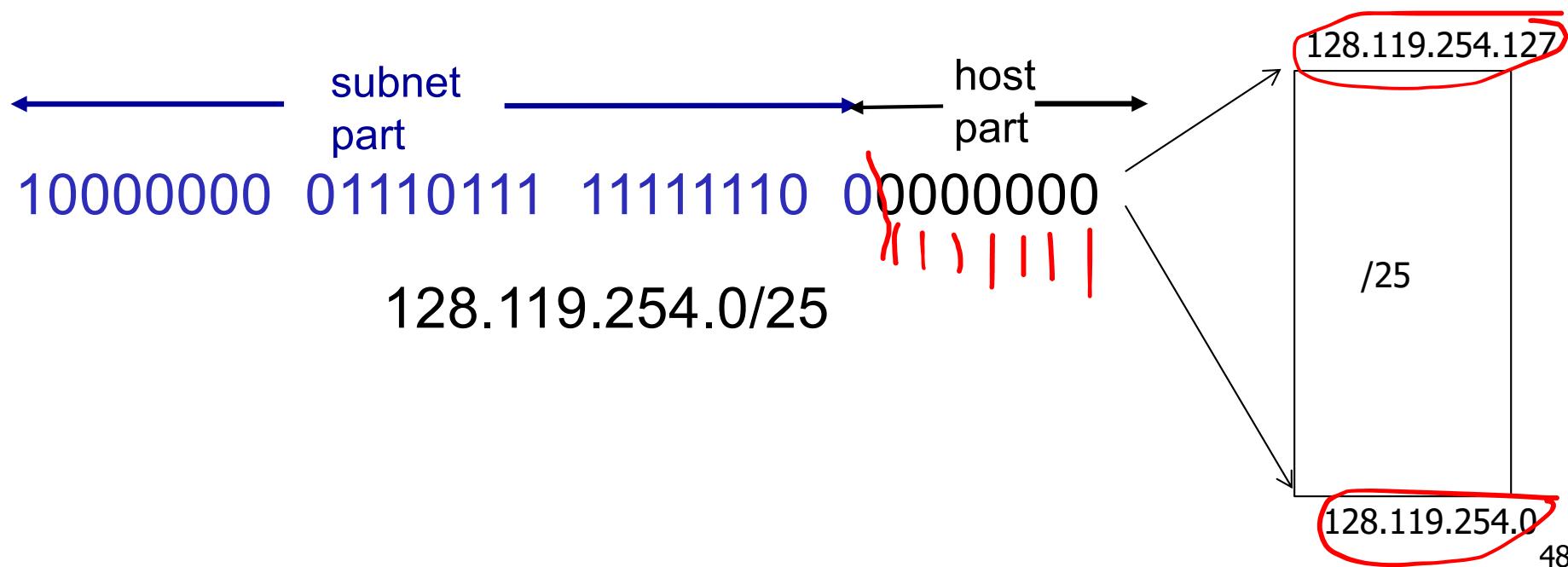


# Quiz: IP Addressing



- How many IP addresses belong to the subnet 128.119.254.0/25 ? What are the IP addresses at the two end-points of this range ?

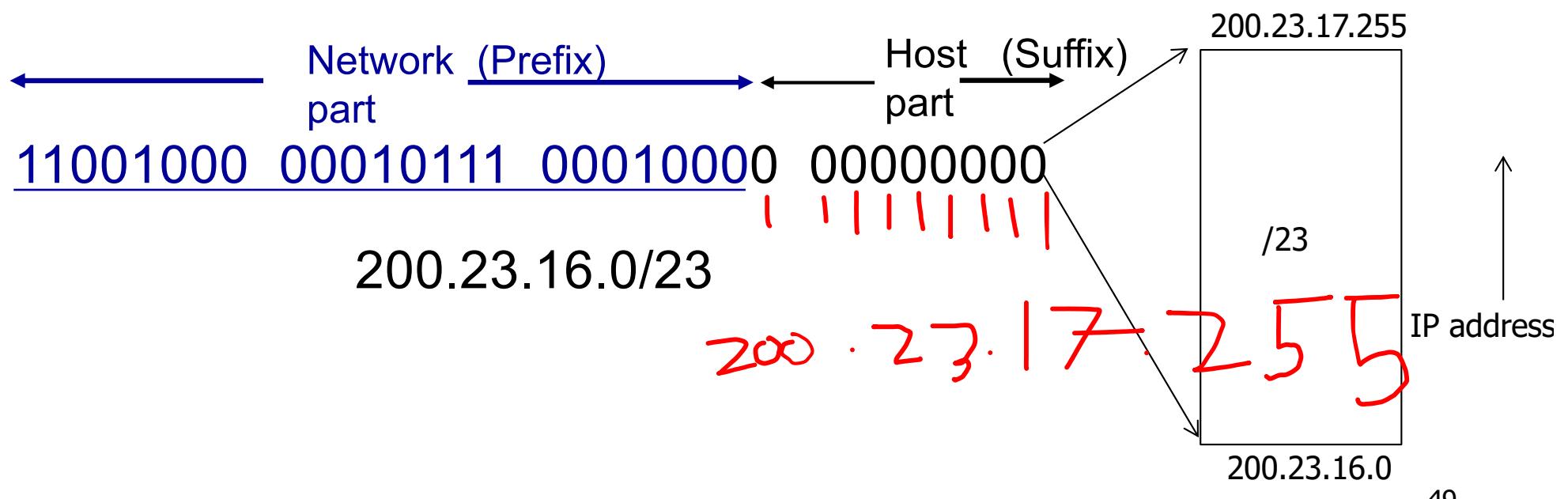
Answer:  $2^7 = 128$  addresses (126 are usable)



# Today's addressing: CIDR

## CIDR: Classless InterDomain Routing

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



# Quiz: IP Addressing



How many IP addresses belong to the subnet  
134.45.22.0/23?

[www.zeeting.com/salil](http://www.zeeting.com/salil)

- A) 32
- B) 64~~A~~
- C) 128
- D) 256
- E) 512

Answer: E

# Quiz: IP Addressing



A small organization is given a block with the beginning address and the prefix length 205.16.37~~24~~/29 (in slash notation). What are the IP addresses at the two end points?

[www.zeeting.com/salil](http://www.zeeting.com/salil)

The beginning address is 205.16.37.24. To find the last address we keep the first 29 bits and change the last 3 bits to 1s.

Beginning: 11001101 00010000 00100101 00011000

Ending : 11001101 00010000 00100101 00011111

There are only 8 addresses in this block.

205.16.37.24 to 205.16.37.31

# Quiz: IP Addressing



An ISP is granted a block of addresses starting with 190.100.0.0/16. The ISP needs to distribute these addresses to three groups of customers as follows:

1. The first group has 64 customers; each needs 256 addresses.
2. The second group has 128 customers; each needs 128 addresses.
3. The third group has 128 customers; each needs 64 addresses.

Design the sub-blocks and give the slash notation for each sub-block. Find out how many addresses are still available after these allocations.

## Group 1

For this group, each customer needs 256 addresses. This means the suffix length is 8 ( $2^8 = 256$ ). The prefix length is then  $32 - 8 = 24$ .

01: 190.100.0.0/24 → 190.100.0.255/24

02: 190.100.1.0/24 → 190.100.1.255/24

.....

64: 190.100.63.0/24 → 190.100.63.255/24

Total =  $64 \times 256 = 16,384$

## Group 2

For this group, each customer needs 128 addresses. This means the suffix length is 7 ( $2^7 = 128$ ). The prefix length is then  $32 - 7 = 25$ . The addresses are:

001: 190.100.64.0/25 → 190.100.64.127/25

002: 190.100.64.128/25 → 190.100.64.255/25

.....

128: 190.100.127.128/25 → 190.100.127.255/25

Total =  $128 \times 128 = 16,384$

## Group 3

For this group, each customer needs 64 addresses. This means the suffix length is 6 ( $2^6 = 64$ ). The prefix length is then  $32 - 6 = 26$ .

001:190.100.128.0/26 → 190.100.128.63/26

002:190.100.128.64/26 → 190.100.128.127/26

.....

128:190.100.159.192/26 → 190.100.159.255/26

Total =  $128 \times 64 = 8,192$

Number of granted addresses: 65,536

Number of allocated addresses: 40,960

Number of available addresses: 24,576

# 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

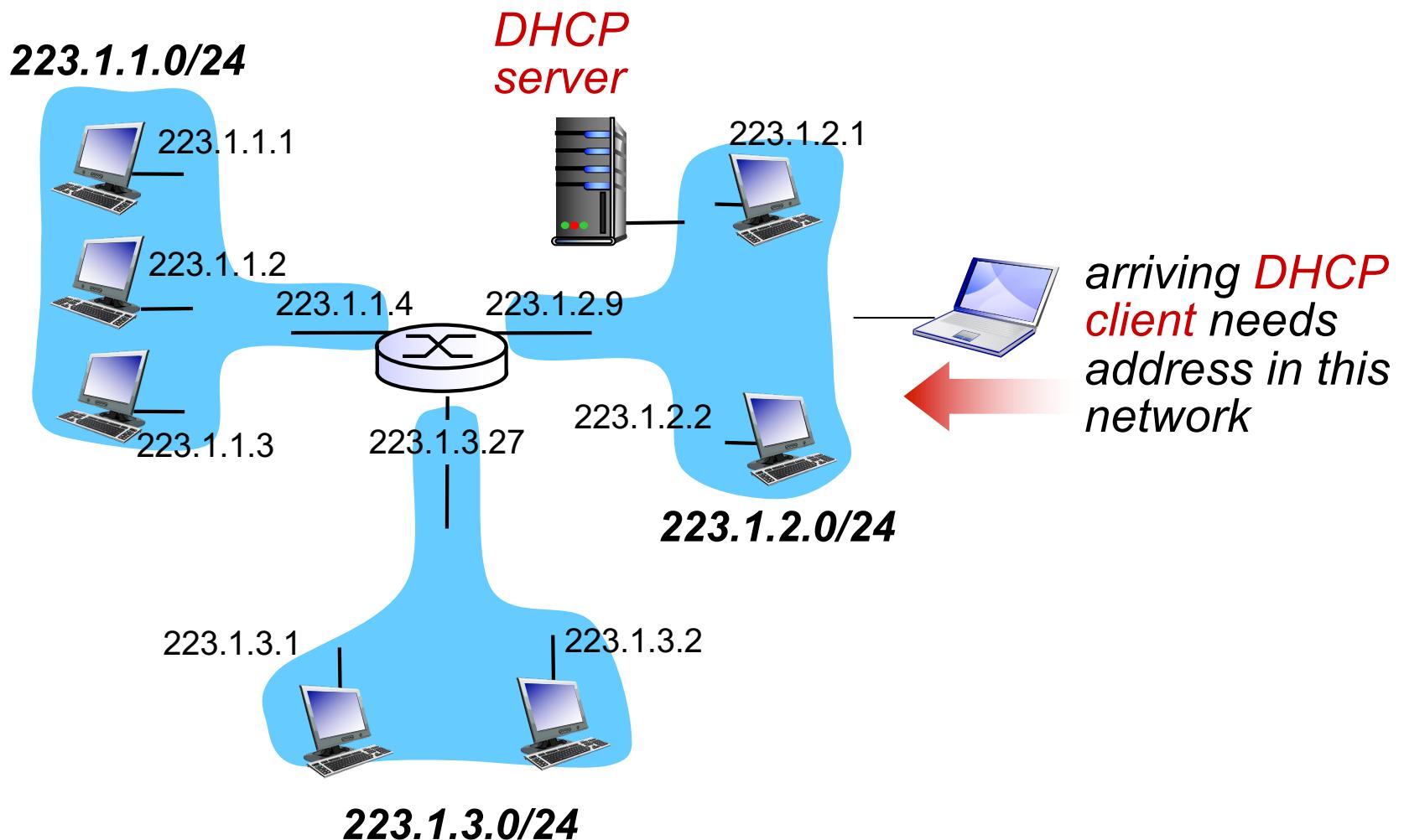
*goal:* allow host to *dynamically* obtain its IP address from network server when it joins network

- 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

*DHCP overview:*

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

# DHCP client-server scenario



# DHCP client-server scenario

DHCP server: 223.1.2.5



DHCP discover

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

arriving  
client



DHCP offer

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

DHCP request

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

DHCP ACK

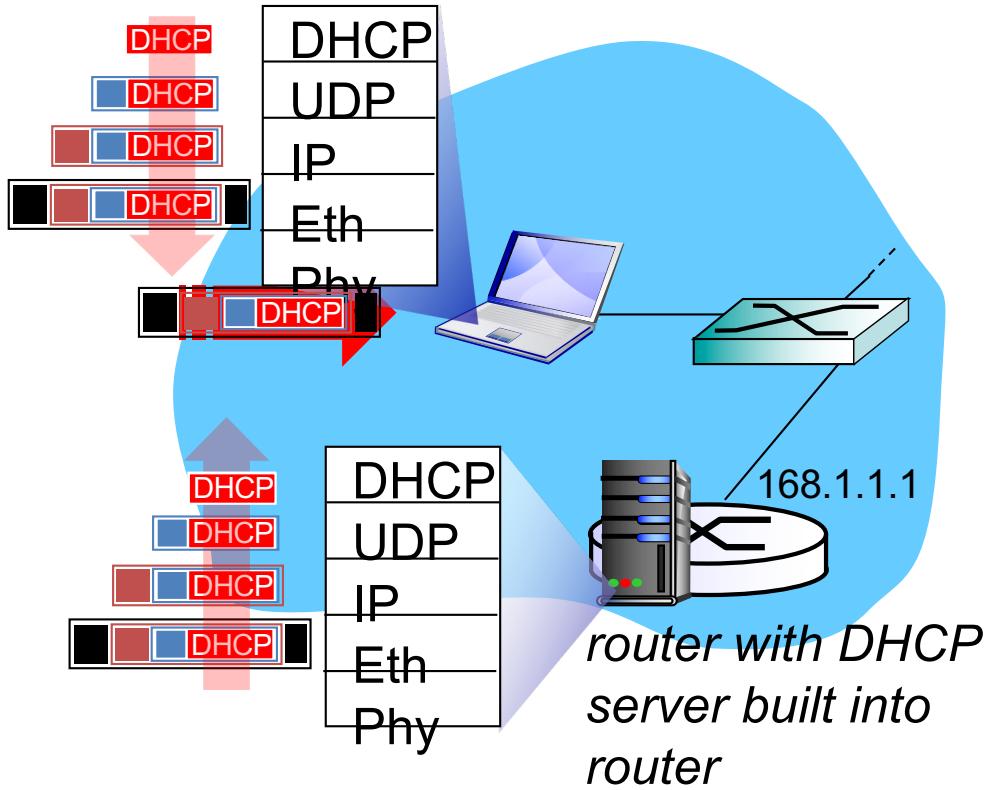
src: 223.1.2.5, 67  
dest: 255.255.255.255, 68  
yiaddr: 223.1.2.4  
transaction ID: 655  
lifetime: 3600 secs

# DHCP: more than IP addresses

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

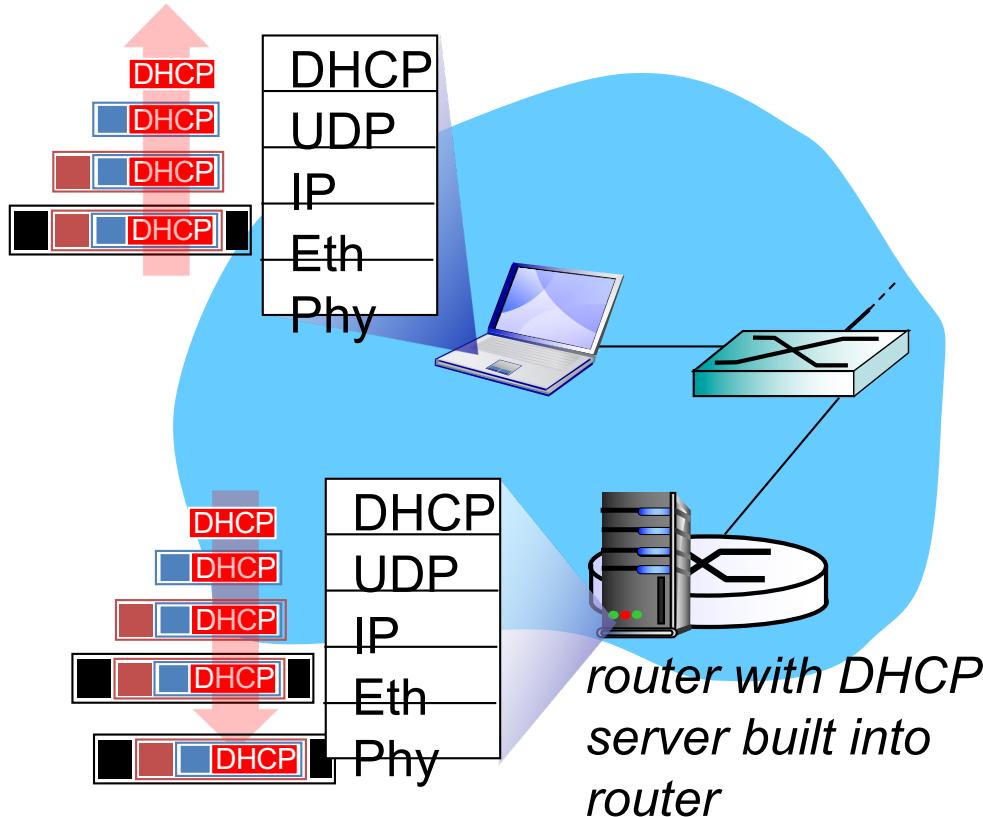
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# DHCP: example



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

# DHCP: example



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

# DHCP: Wireshark output (home LAN)

## request

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

.....

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."**

# DHCP: further details

- DHCP uses UDP and port numbers 67 (server side) and 68 (client side)
- Usually the MAC address is used to identify clients
  - DHCP server can be configured with a “registered list” of acceptable MAC addresses
- DHCP offer message includes ip address, length of lease, subnet mask, DNS servers, default gateway
- DHCP security holes
  - DoS attack by exhausting pool of IP addresses
  - Masquerading as a DHCP server
  - Authentication for DHCP - RFC 3118

# IP addresses: how to get one?

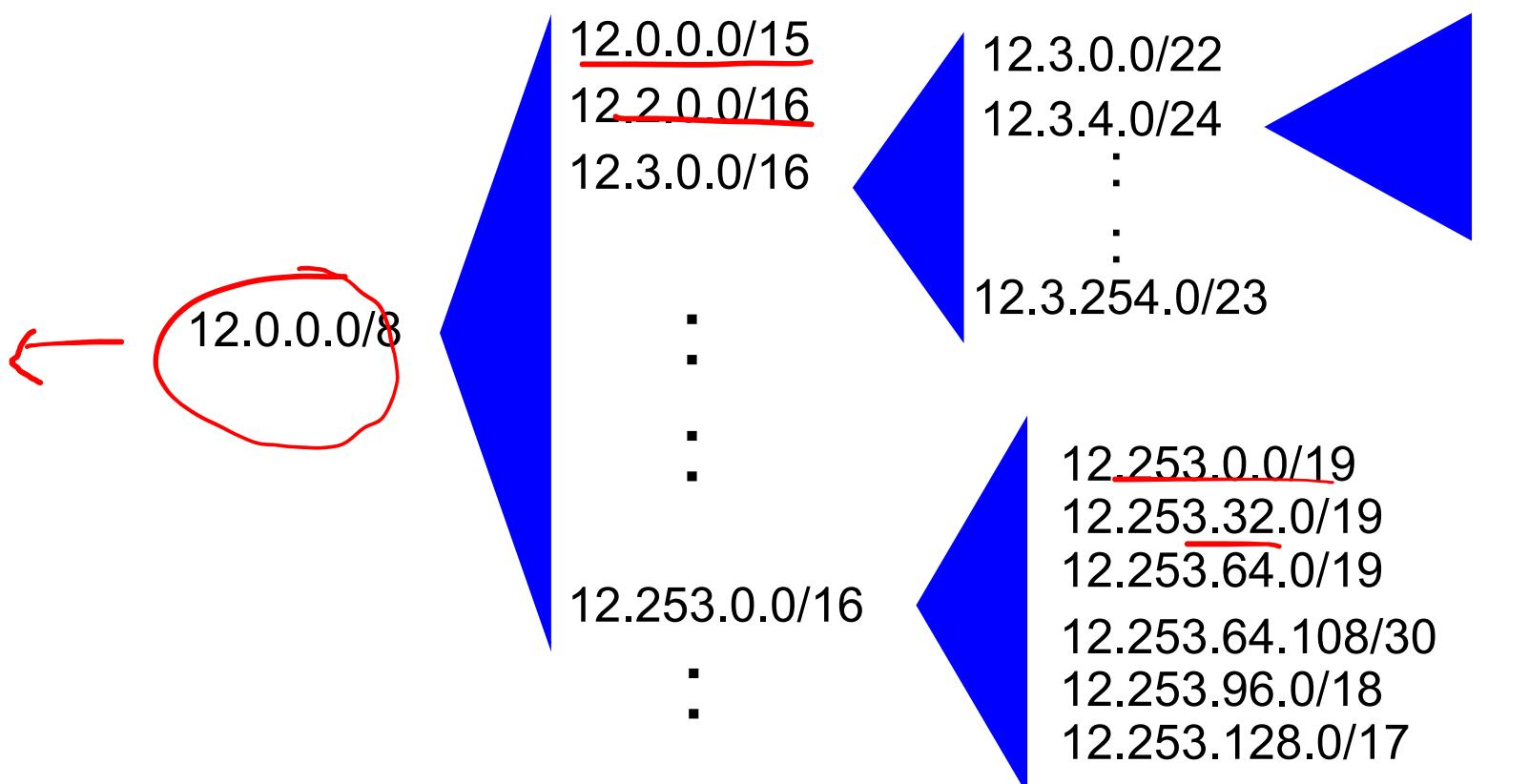
**Q:** how does *network* get subnet part of IP addr?

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

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

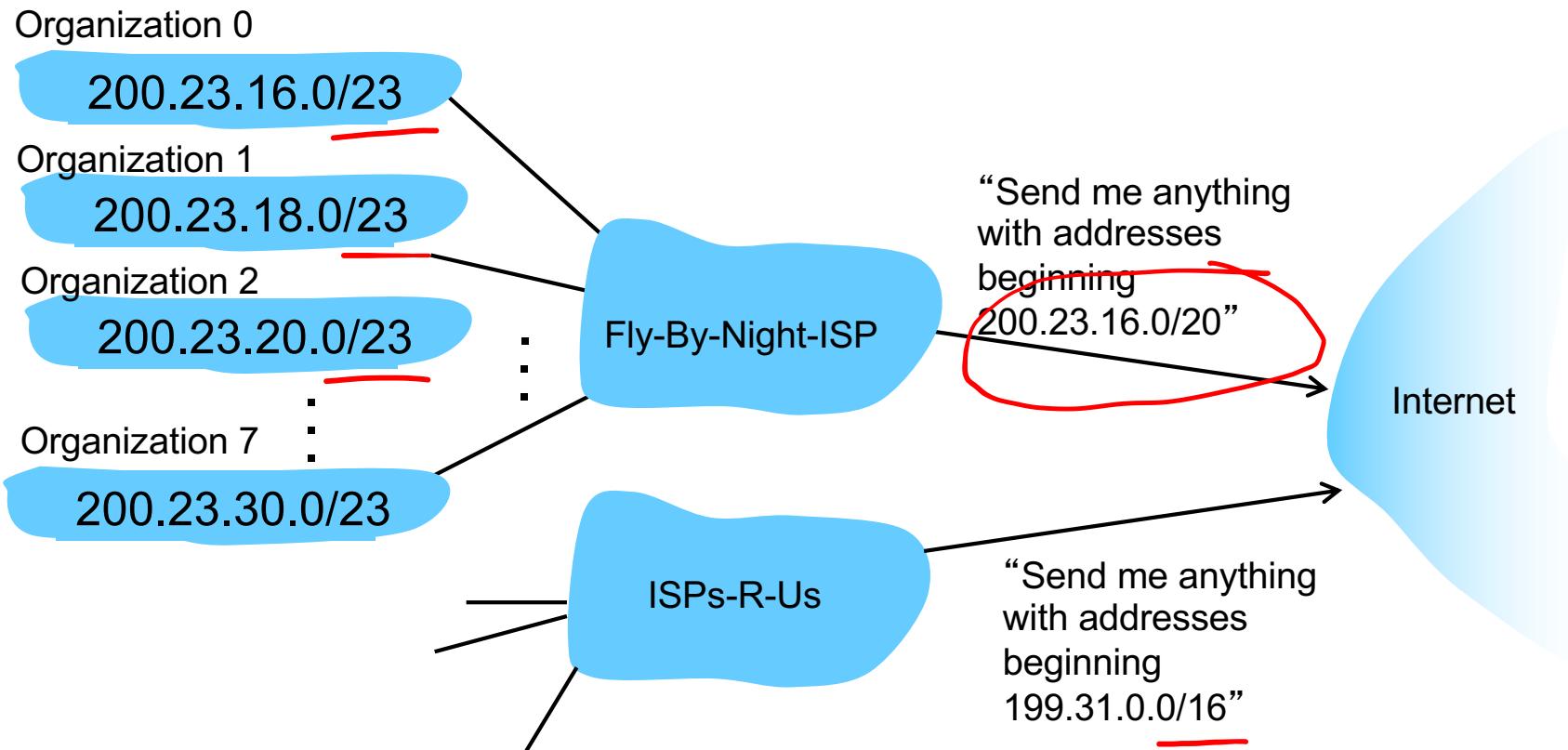
# CIDR: Addresses allocated in contiguous prefix chunks

Recursively break down chunks as get closer to host

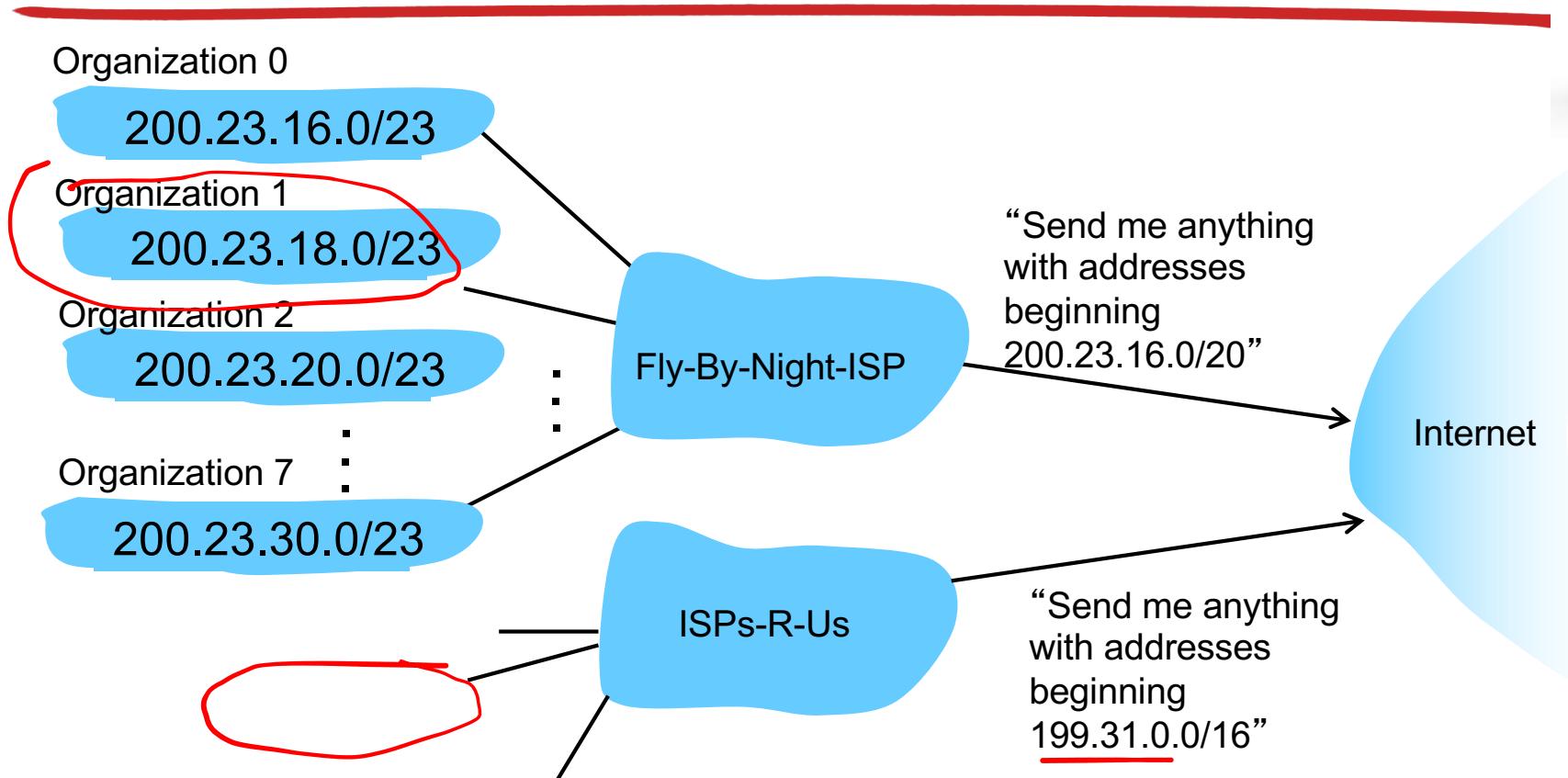


# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



# Quiz: What should we do if organization 1 decides to switch to ISPs-R-Us

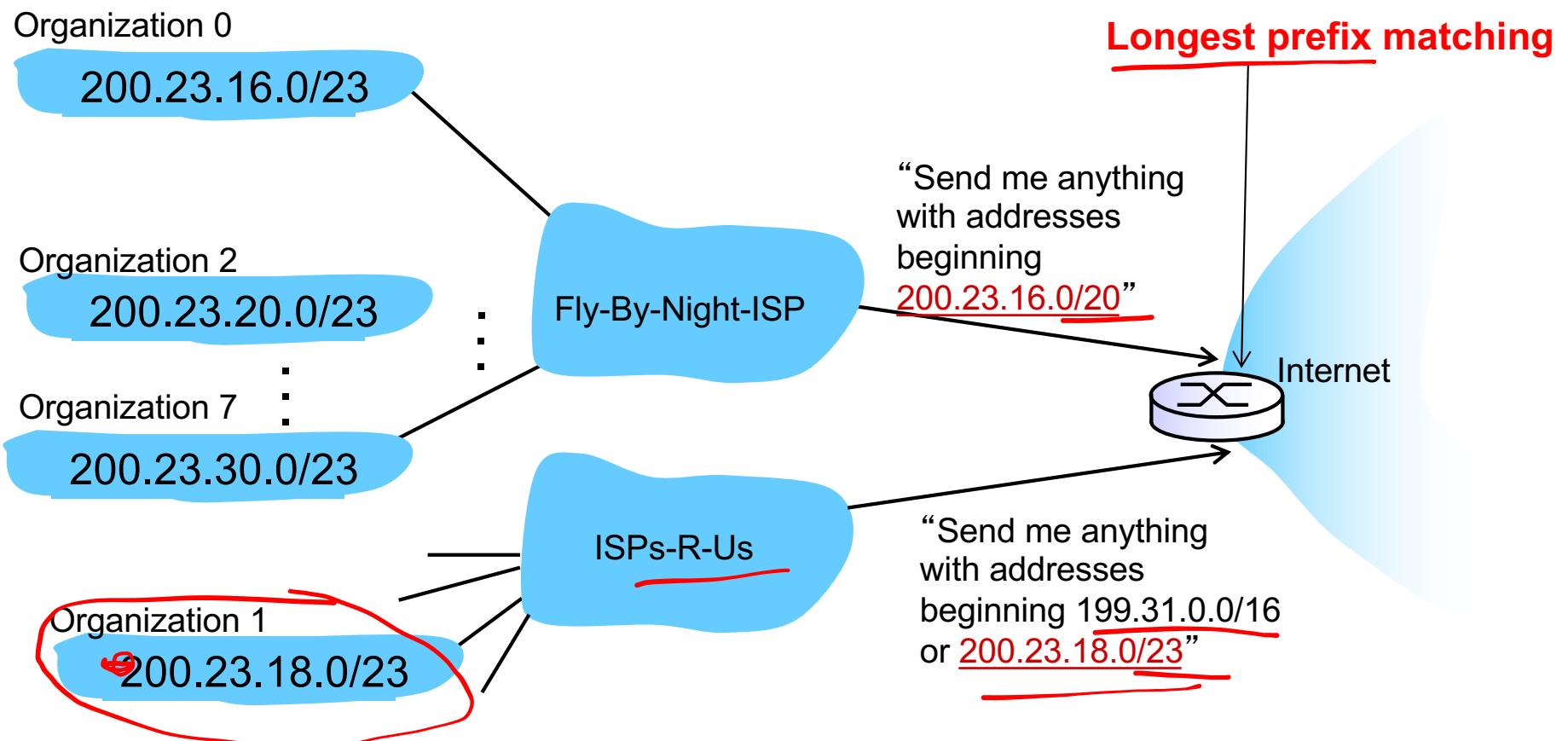


- A: Move 200.23.18.0/23 to ISPs-R-Us (and break up Fly-By-Night's/20 block).
- B: Give new addresses to Organization 1 (and force them to change all their addresses)
- C: Some other solution

Both A and B are valid. A is explained on the next 2 slides

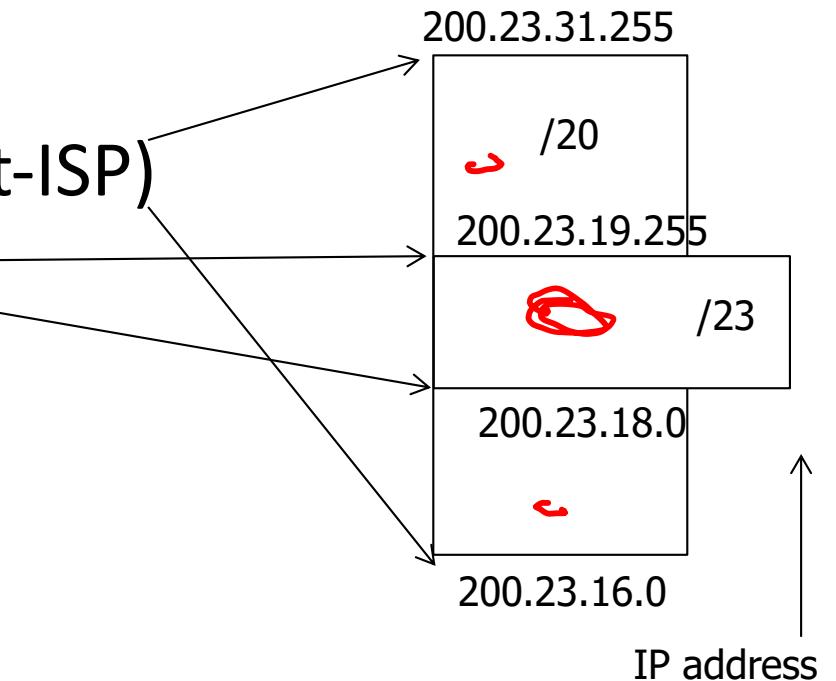
# Hierarchical addressing: more specific routes

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



## Example: continued

- But how will this work?
- Routers in the Internet will have two entries in their tables
  - ~~200.23.16.0/20~~ (Fly-by-Night-ISP)
  - ~~200.23.18.0/23~~ (ISPs-R-Us)
- Longest prefix match



# 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*** **** <i>00010</i>	0
11001000 00010111 00011000 **** <i>00011000</i>	1
11001000 00010111 00011*** **** <i>00011</i>	2
otherwise	3

examples:

DA: 11001000 00010111 00010*110* 10100001

which interface?

0

DA: 11001000 00010111 00011*000* 10101010

which interface?

1



# Quiz: Longest prefix matching

- On which outgoing interface will a packet destined to 11011001 be forwarded?

Prefix	Interface
<u>1*</u> ✓	A
<u>11*</u> ✓	B
111* ✗	C
Default	D

# More on IP addresses

Source: [www.xkcd.com](http://www.xkcd.com)

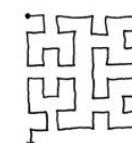
- IP addresses are allocated as blocks and have geographical significance
- It is possible to determine the geographical location of an IP address

<http://www.geobytes.com/IpLocator.htm>



THIS CHART SHOWS THE IP ADDRESS SPACE ON A PLANE USING A FRACTAL MAPPING WHICH PRESERVES GROUPING -- ANY CONSECUTIVE STRING OF IPs WILL TRANSLATE TO A SINGLE COMPACT, CONTIGUOUS REGION ON THE MAP. EACH OF THE 256 NUMBERED BLOCKS REPRESENTS ONE /8 SUBNET (CONTAINING ALL IPs THAT START WITH THAT NUMBER). THE UPPER LEFT SECTION SHOWS THE BLOCKS SOLD DIRECTLY TO CORPORATIONS AND GOVERNMENTS IN THE 1990's BEFORE THE RIRs TOOK OVER ALLOCATION.

0	1	14	15	16	19
3	2	13	12	17	18
4	7	8	11		
5	6	9	10		



= UNALLOCATED BLOCK

# 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/>



IANA works through Regional Internet Registries (RIRs):



American Registry for Internet Numbers



Latin America and Caribbean Network Information Centre



IRéseaux IP Européens Network Coordination Centre



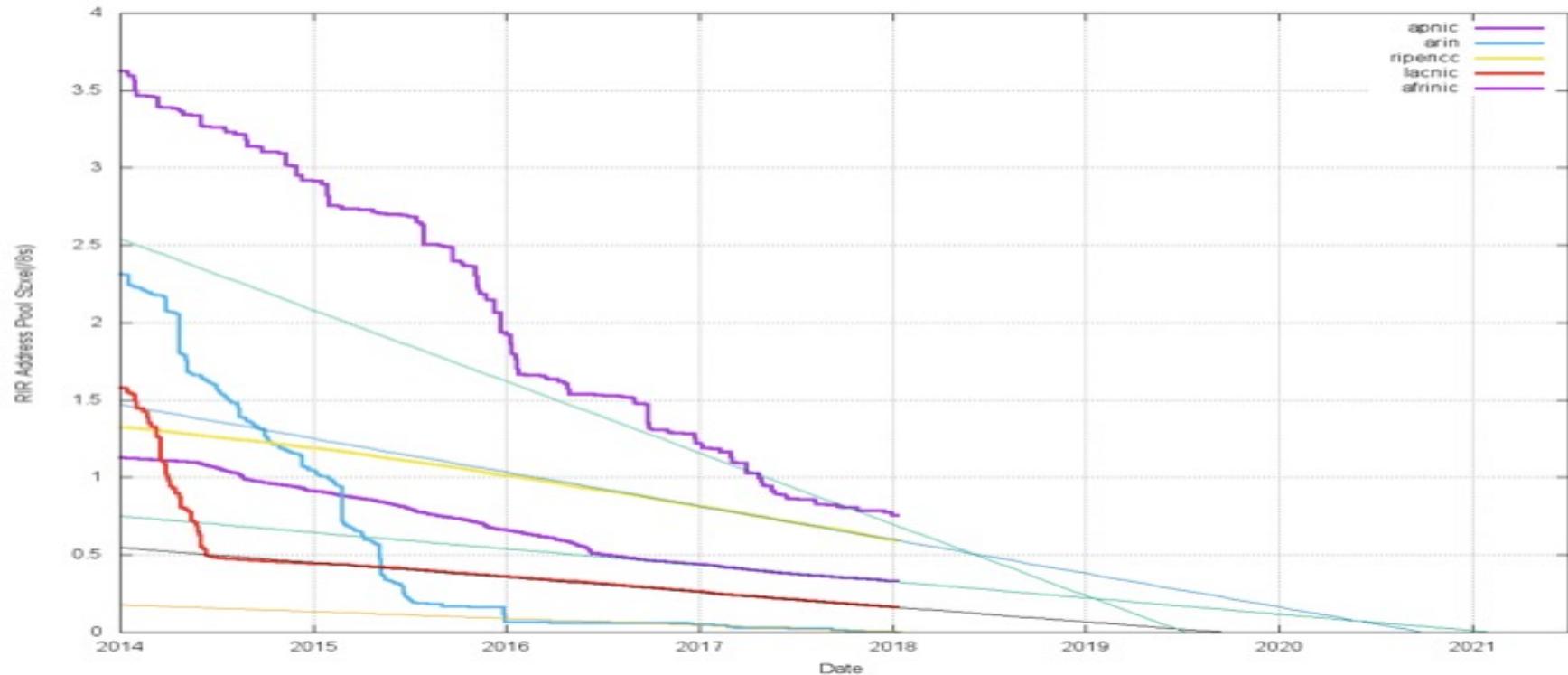
Asia-Pacific Network Information Center



African Network Information Centre



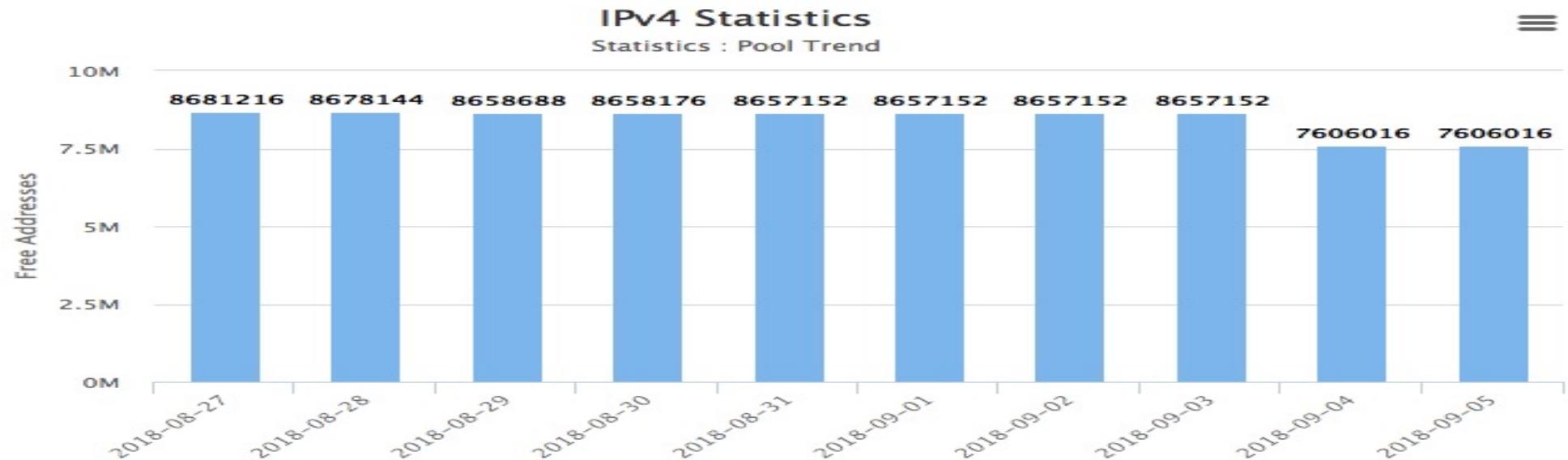
### - Address Pool Consumption Model



## IPv4 Exhaustion

In this section you can find statistics for IPv4 pool in the AfriNIC region.

[IPv4 Usage per IANA allocation](#) [IPv4 available space over time](#) [IPv4 availability by prefix size](#)



# Made-up Example in More Detail

- ICANN gives APNIC several /8s
- APNIC gives Telstra one /8, **129/8**
  - Network Prefix: **10000001**
- Telstra gives UNSW a /16, **129.94/16**
  - Network Prefix: **1000000101011110**
- UNSW gives CSE a /24, **129.94.242/24**
  - Network Prefix: **100000010101111011110010**
- CSE gives me a specific address **129.94.242.51**
  - Address: **10000001010111101111001000110011**

# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

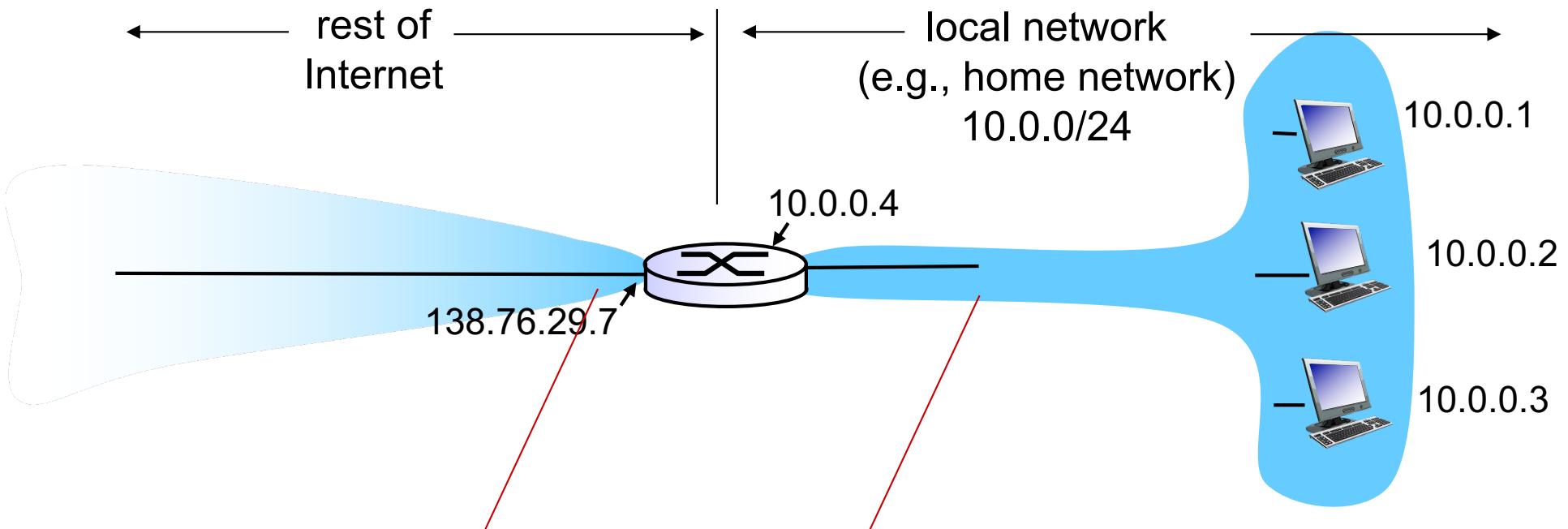
## 4.3 IP: Internet Protocol

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

# Private Addresses

- Defined in RFC 1918:
  - 10.0.0.0/8 (16,777,216 hosts)
  - 172.16.0.0/12 (1,048,576 hosts)
  - 192.168.0.0/16 (65536 hosts)
- These addresses cannot be routed
  - Anyone can use them
  - Typically used for NAT

# 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

*implementation:* NAT router must:

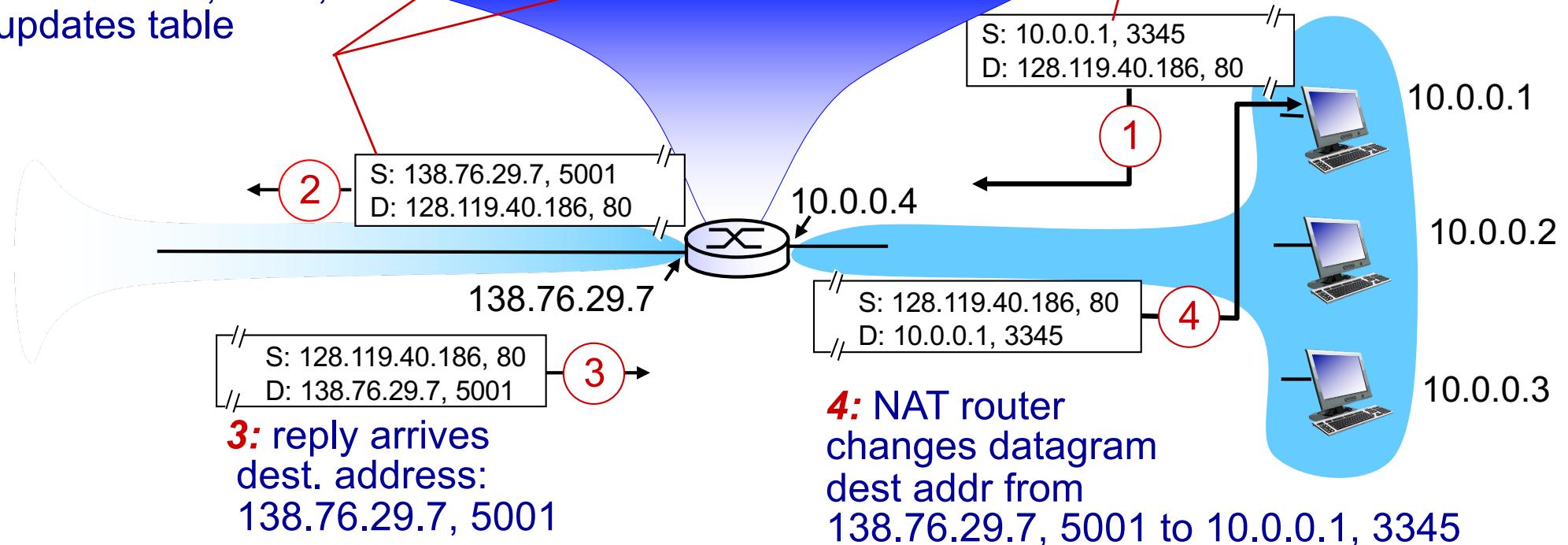
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)  
. . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr
- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP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

# NAT: network address translation

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

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

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80



# NAT Advantages

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
  - 16-bit port-number field: ~65,000 simultaneous connections with one WAN-side address!
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network

# NAT Disadvantages

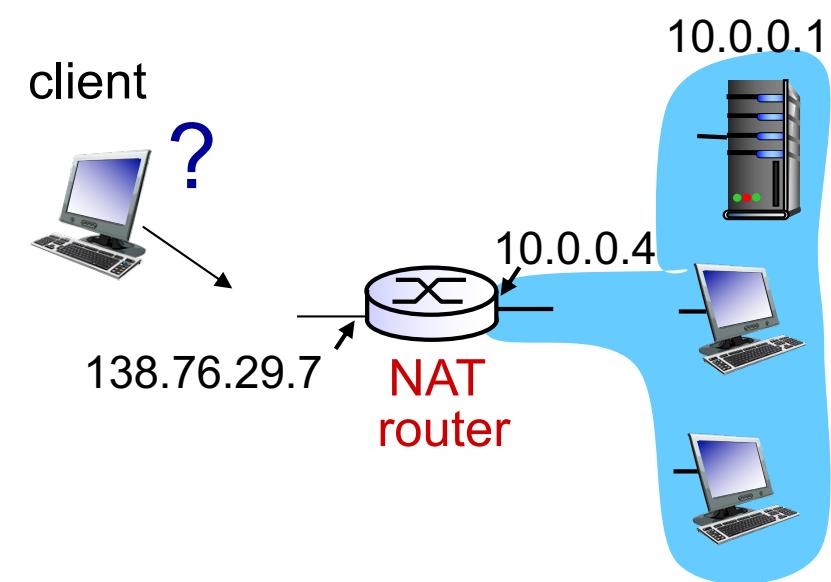
- NAT violates the architectural model of IP
  - Every IP address uniquely identifies a single node on Internet
  - routers should only process up to layer 3
- NAT changes the Internet from connection less to a kind of connection-oriented network
- NAT possibility must be taken into account by app designers, e.g., P2P applications

# NAT: Practical Issues

- NAT modifies port # and IP address
  - *Requires recalculation of TCP and IP checksum*
- Some applications embed IP address or port numbers in their message payloads
  - DNS, FTP (PORT command), SIP, H.323
  - For legacy protocols, NAT must look into these packets and translate the embedded IP addresses/port numbers
  - Duh, What if these fields are encrypted ?? (SSL/TLS, IPSEC, etc.)
  - **Q: In some cases, why may NAT need to change TCP sequence number?? (Discussion Question on Website)**
- If applications change port numbers periodically, the NAT must be aware of this
- NAT Traversal Problems
  - How to setup a server behind a NAT router?
  - How to talk to a Skype user behind a NAT router?

# NAT traversal problem

- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7
- *Solution1:* Inbound-NAT  
Statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (138.76.29.7, port 2500)  
always forwarded to 10.0.0.1 port 25000

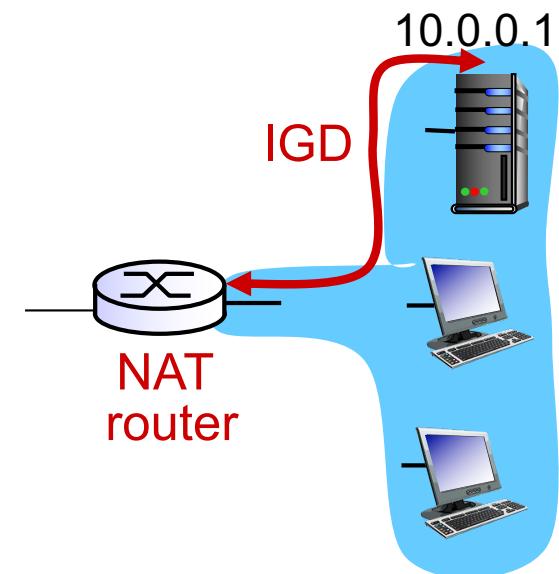


# NAT traversal problem

- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:

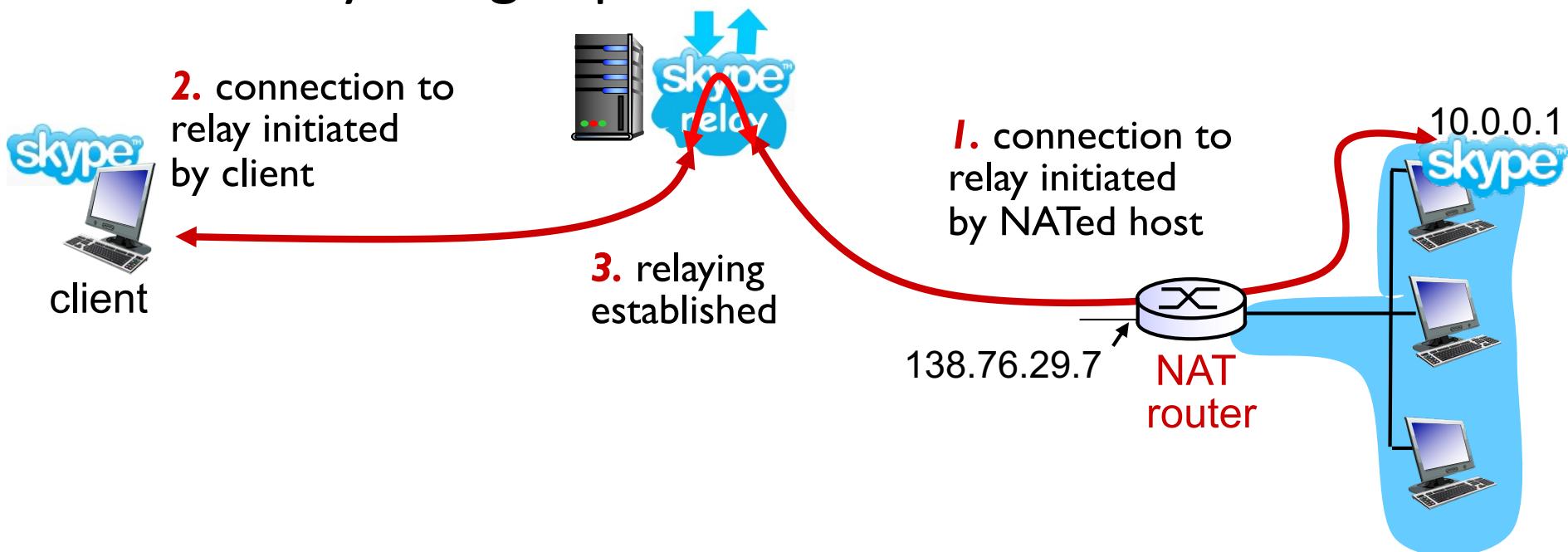
- ❖ learn public IP address (138.76.29.7)
- ❖ add/remove port mappings (with lease times)

i.e., automate static NAT port map configuration



# NAT traversal problem

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



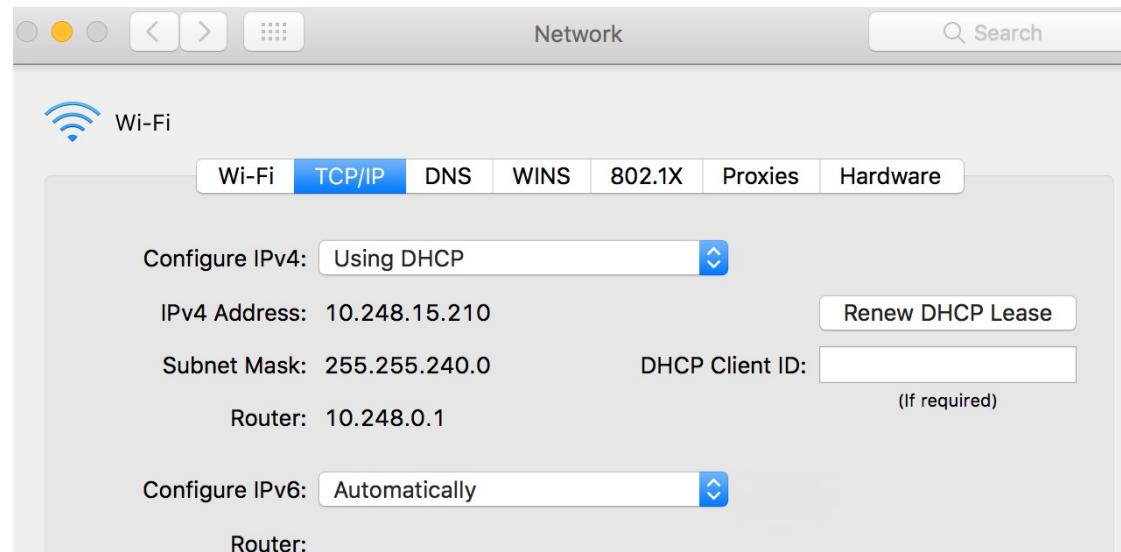
# NAT: Devil in the details

- Despite the problems, NAT has been widely deployed
- Most protocols can be successfully passed through a NAT, including VPN
- Modern hardware can easily perform NAT functions at > 100 Mbps
- IPv6 is still not widely deployed commercially, so the need for NAT is real
- After years of refusing to work on NAT, the IETF has been developing “NAT control protocols” for hosts
- Lot of practical variations
  - Full-cone NAT, Restricted Cone NAT, Port Restricted Cone NAT, Symmetric NAT, .....
  - The devil is in the detail (NOT COVERED IN THE COURSE)



# Discussion

- The picture below shows you the IP address of my machine connected to the uniwide wireless network.



- However when I ask Google it says my IP address is as noted below. Can you explain the discrepancy?

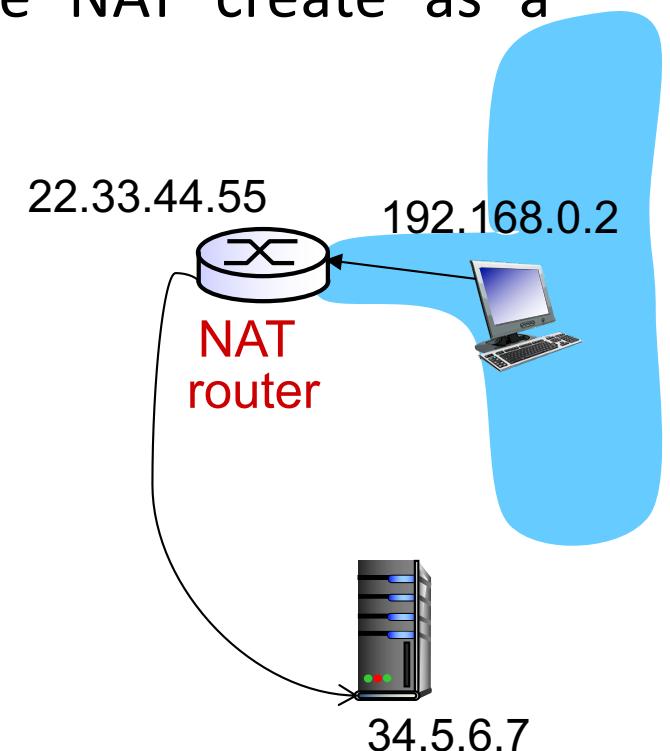
129.94.8.210  
Your public IP address



# Quiz: NAT

- A host with a private IP address 192.168.0.2 opens a TCP socket on its local port 4567 and connects to a web server at 34.5.6.7. The NAT's public IP address is 22.33.44.55. Which of the following mapping entries *could* the NAT create as a result?
  - [22.33.44.55, 4567] → [192.168.0.2, 80]
  - [34.5.6.7, 80] → [22.33.44.55, 4567]
  - [192.168.0.2, 80] → [34.5.6.7, 4567]
  - [22.33.44.55, 3967] → [192.168.0.2, 4567]

**ANSWER: D**

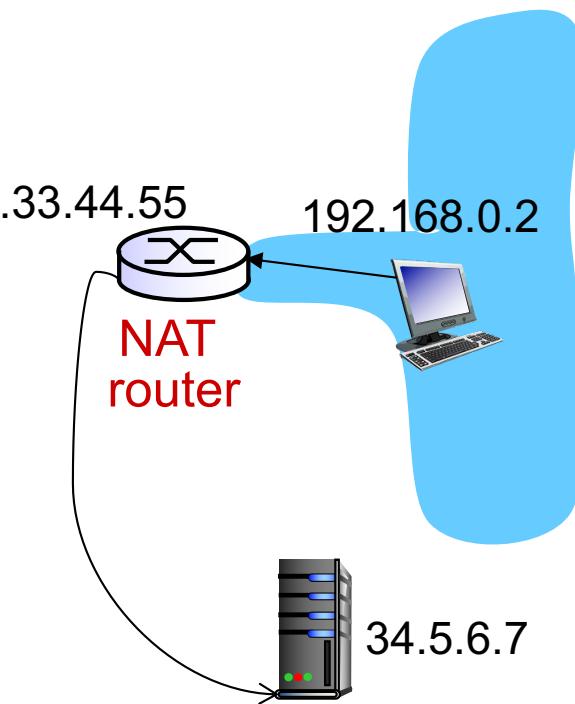


# Quiz: NAT



- A host with a private IP address 192.168.0.2 opens a TCP socket on its local port 4567 and connects to a web server at 34.5.6.7. The NAT's public IP address is 22.33.44.55. Suppose the NAT created the mapping  $[22.33.44.55, 3967] \rightarrow [192.168.0.2, 4567]$  as a result. What are the source and destination port numbers in the SYN-ACK response from the server?  
A. 80, 3967  
B. 4567, 80  
C. 3967, 80  
D. 3967, 4567  
E. 80, 4567

**ANSWER: A**



# **COMP 3331/9331: Computer Networks and Applications**

**Week 8**

**Network Layer: Data Plane + Control Plane  
(Routing)**

**Chapter 4: Section 4.3**

**Chapter 5: Section 5.1 – 5.2, 5.6**

# Network Layer, data plane: 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

# 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

<https://www.google.com/intl/en/ipv6/statistics.html>

# IPv6 datagram format

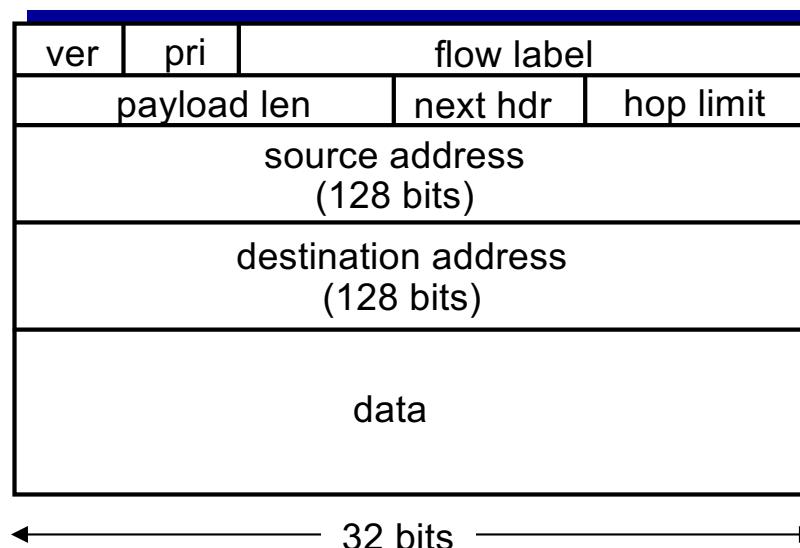
---

*priority*: identify priority among datagrams in flow (traffic class)

*flow Label*: identify datagrams in same “flow.”

(concept of “flow” not well defined).

*next header*: identify upper layer protocol for data



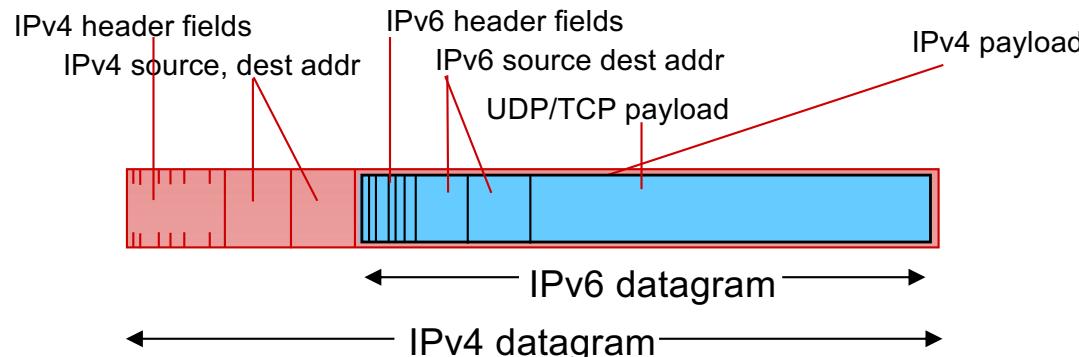
# Other changes from IPv4

- ❖ **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

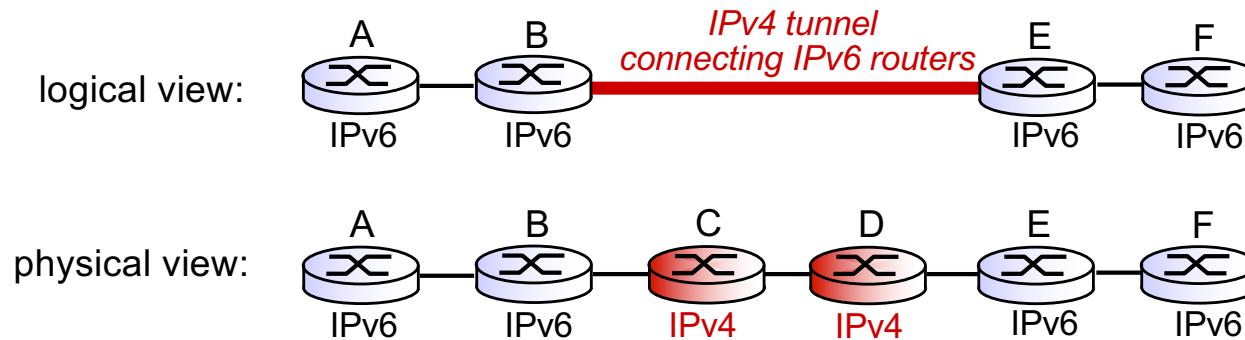
# Transition from IPv4 to IPv6

---

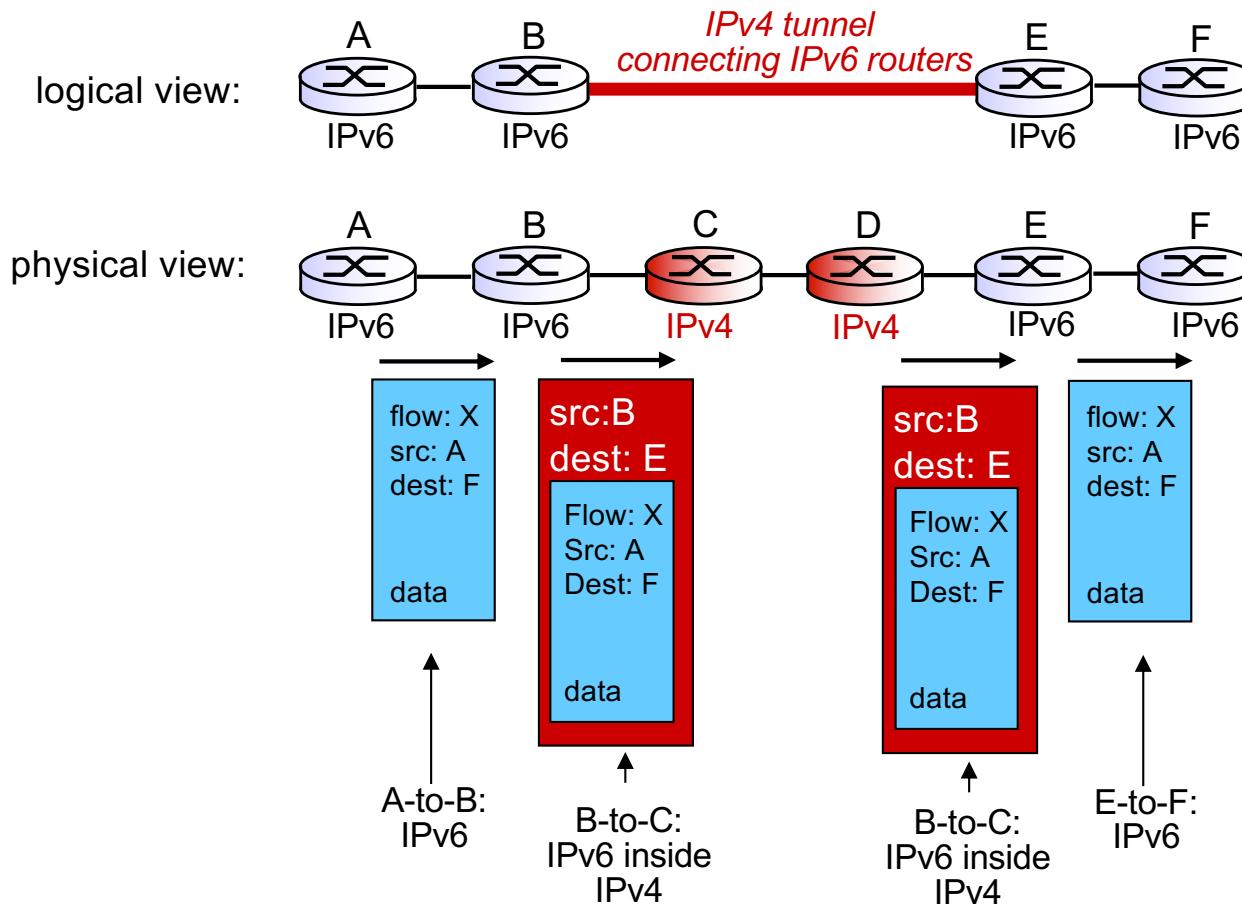
- ❖ not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?
- ❖ *tunneling*: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



# Tunneling (IPv6 over IPv4)

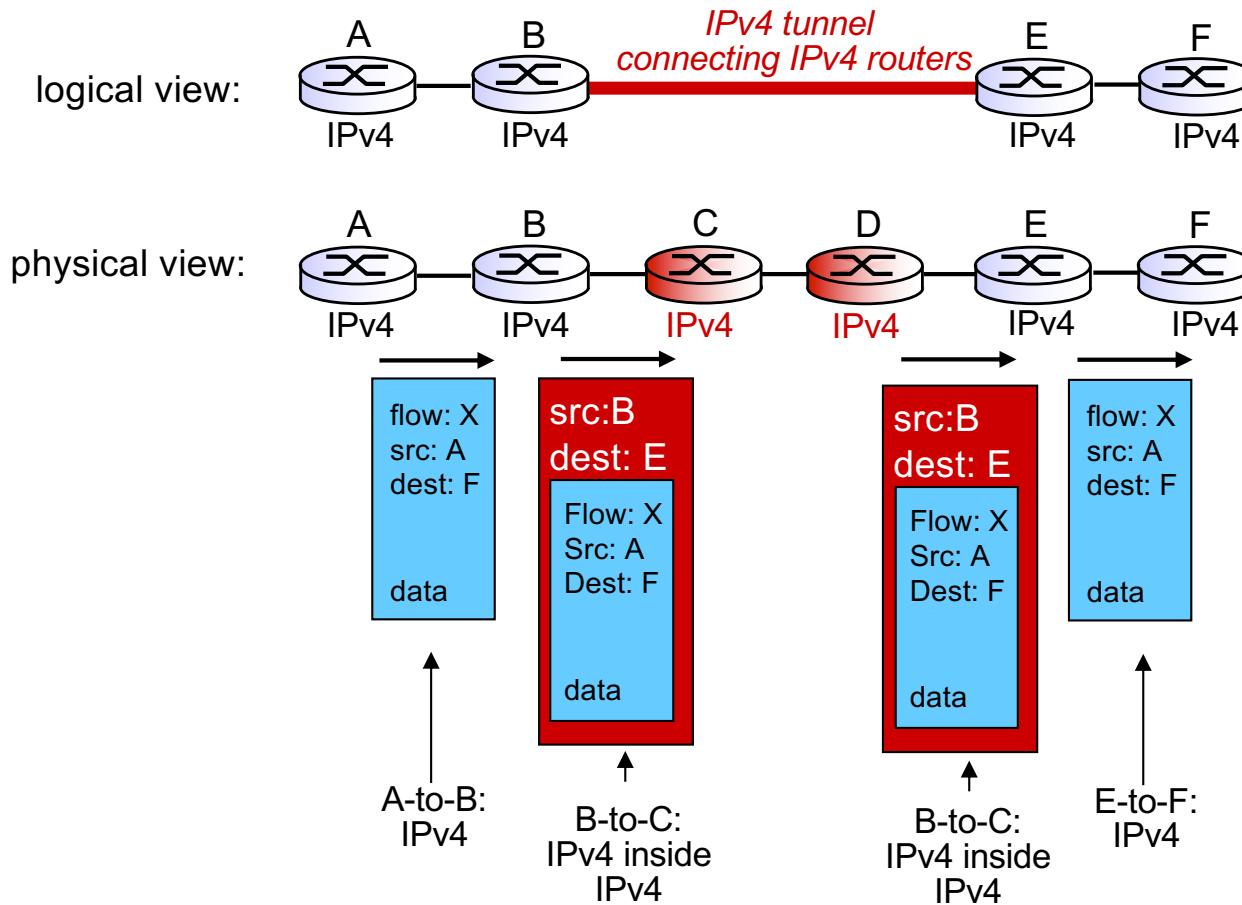


# Tunneling (IPv6 over IPv4)



# Tunneling (IPv4 over IPv4)

Used in VPNs



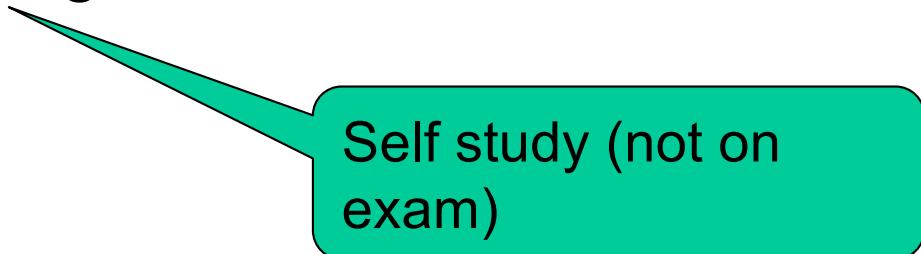
# Network layer, control plane: outline

## 5.1 introduction

## 5.2 routing protocols

- ❖ link state
- ❖ distance vector
- ❖ hierarchical routing

## 5.6 ICMP: The Internet Control Message Protocol



Self study (not on exam)

# Network-layer functions

*Recall: two network-layer functions:*

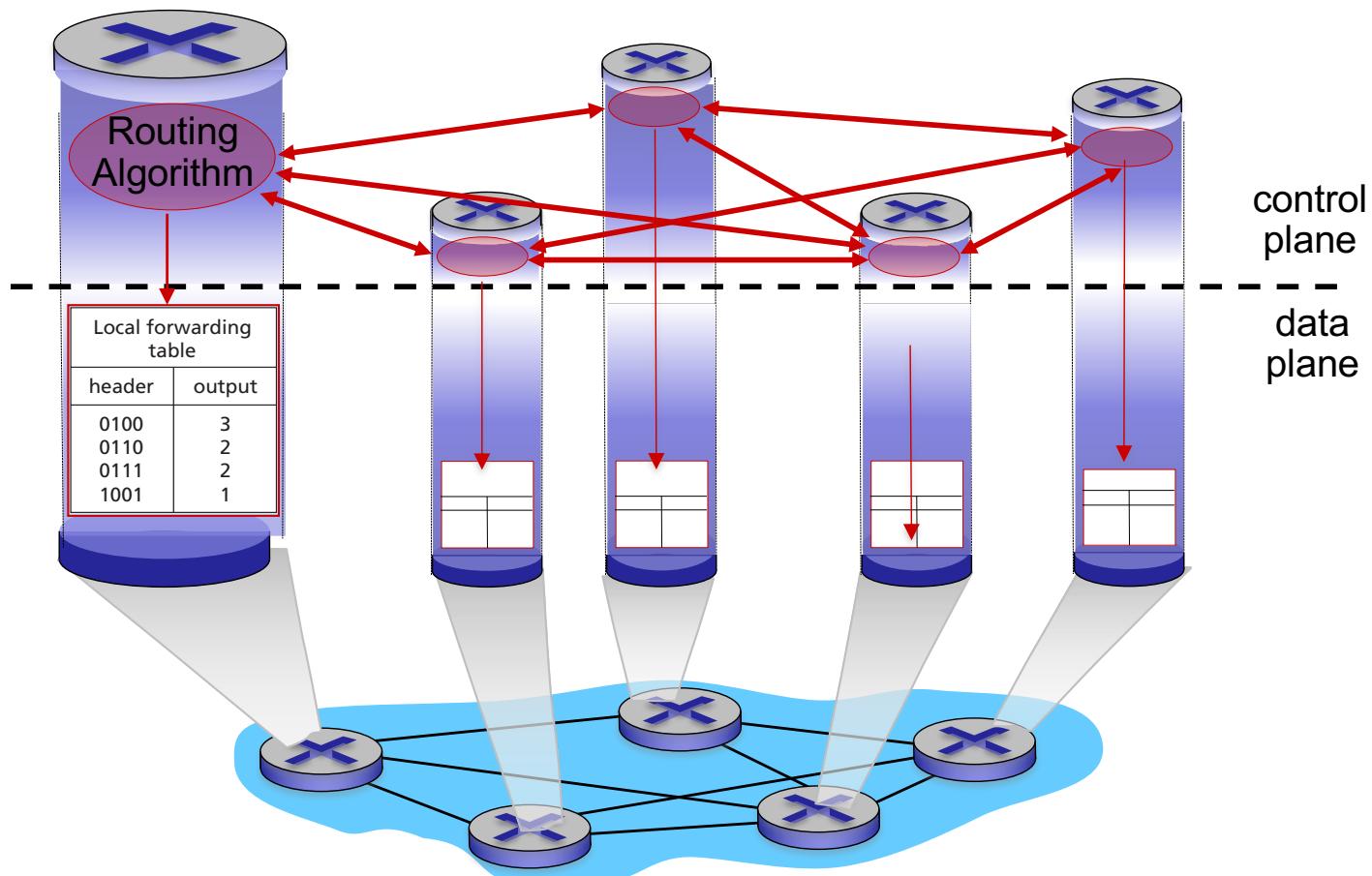
- ❖ *forwarding*: move packets from router's input to appropriate router output      ***data plane***
- *routing*: determine route taken by packets from source to destination      ***control plane***

*Two approaches to structuring network control plane:*

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

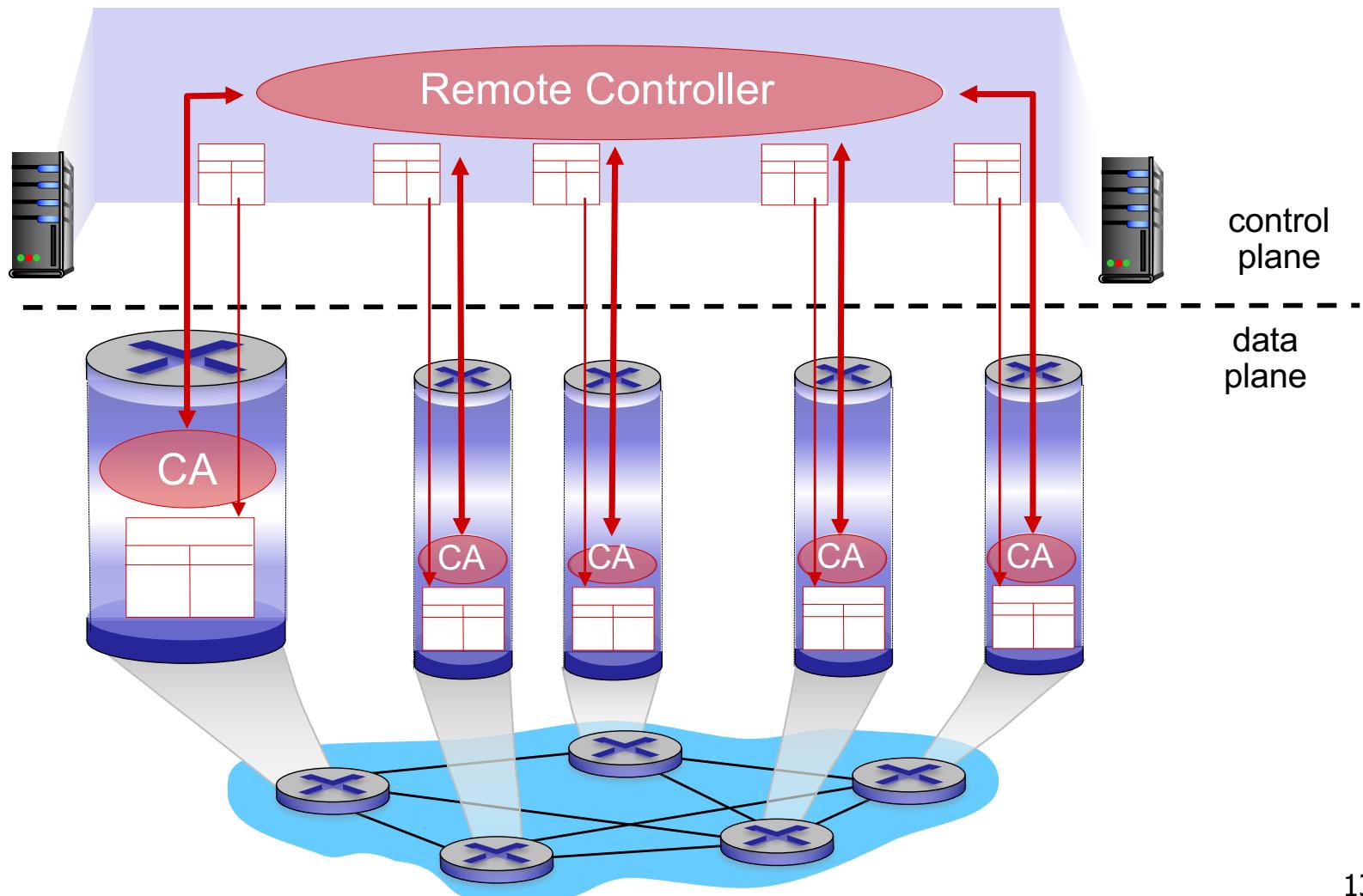
# Per-router control plane

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



# Logically centralized control plane

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



# Network layer, control plane: outline

5.1 introduction

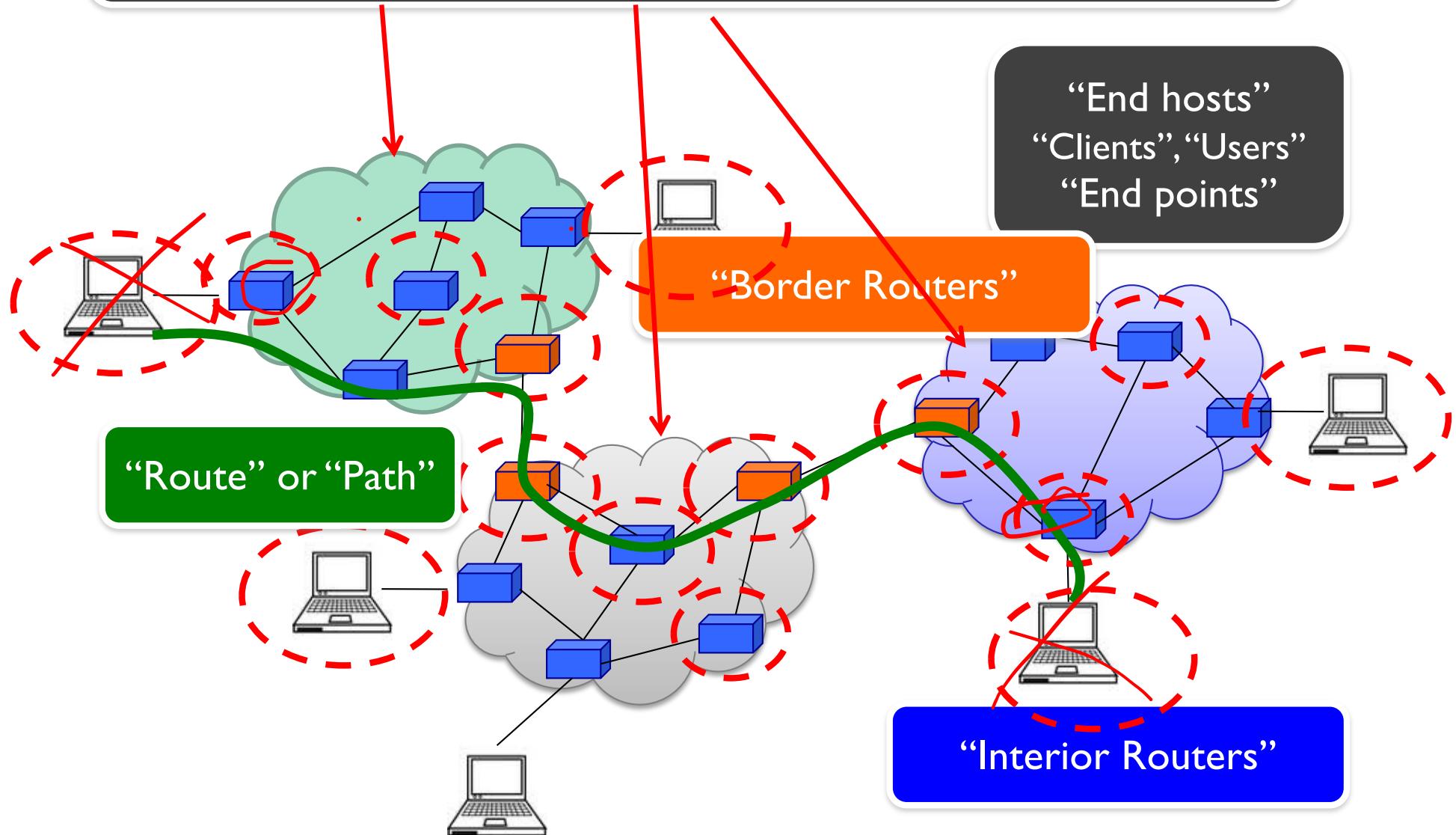
5.2 routing protocols

- ❖ link state
- ❖ distance vector
- ❖ Hierarchical routing

5.6 ICMP: The Internet  
Control Message  
Protocol

## “Autonomous System (AS) or “Domain”

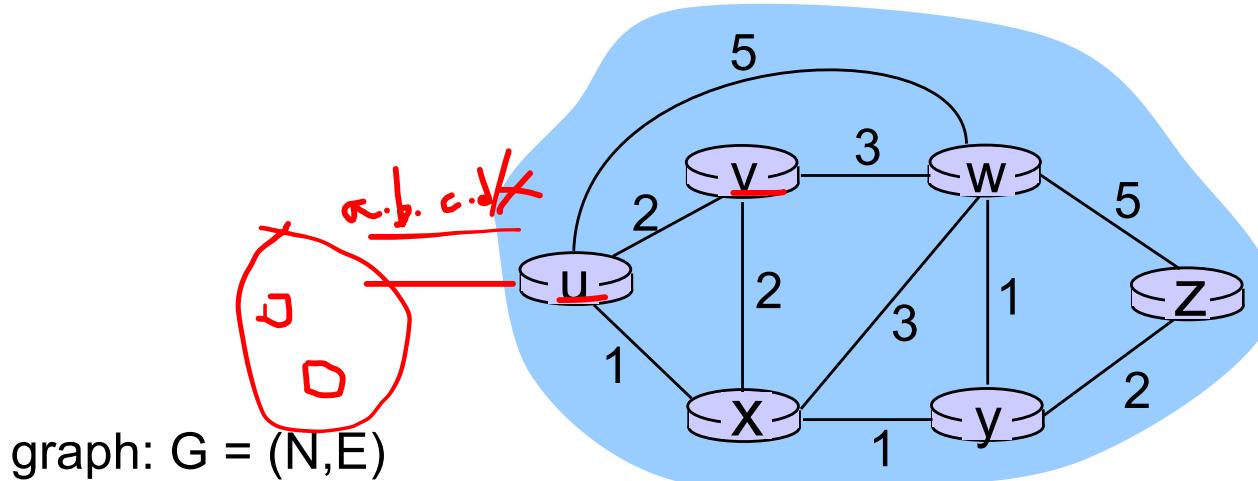
Region of a network under a single administrative entity



# Internet Routing

- ❖ Internet Routing works at two levels
- ❖ Each AS runs an **intra-domain** routing protocol that establishes routes within its domain
  - AS -- region of network under a single administrative entity
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)
- ❖ ASes participate in an **inter-domain** routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)

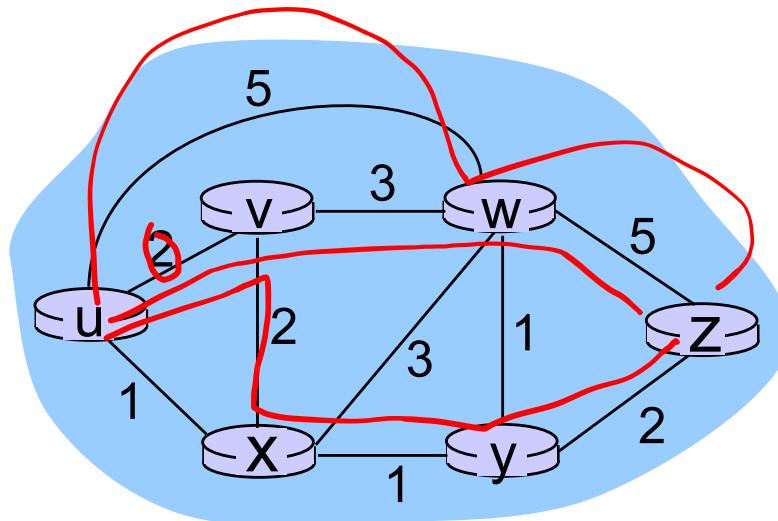
# Graph abstraction



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

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

# Graph abstraction: costs



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

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

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

## Link Cost

- ❖ Typically simple: all links are equal
- ❖ Least-cost paths => shortest paths (hop count)
- ❖ Network operators add policy exceptions
  - Lower operational costs
  - Peering agreements
  - Security concerns

# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- ❖ link state

- ❖ distance vector

- ❖ hierarchical routing

5.6 ICMP: The Internet  
Control Message  
Protocol

# Routing algorithm classes

## *Link State (Global)*

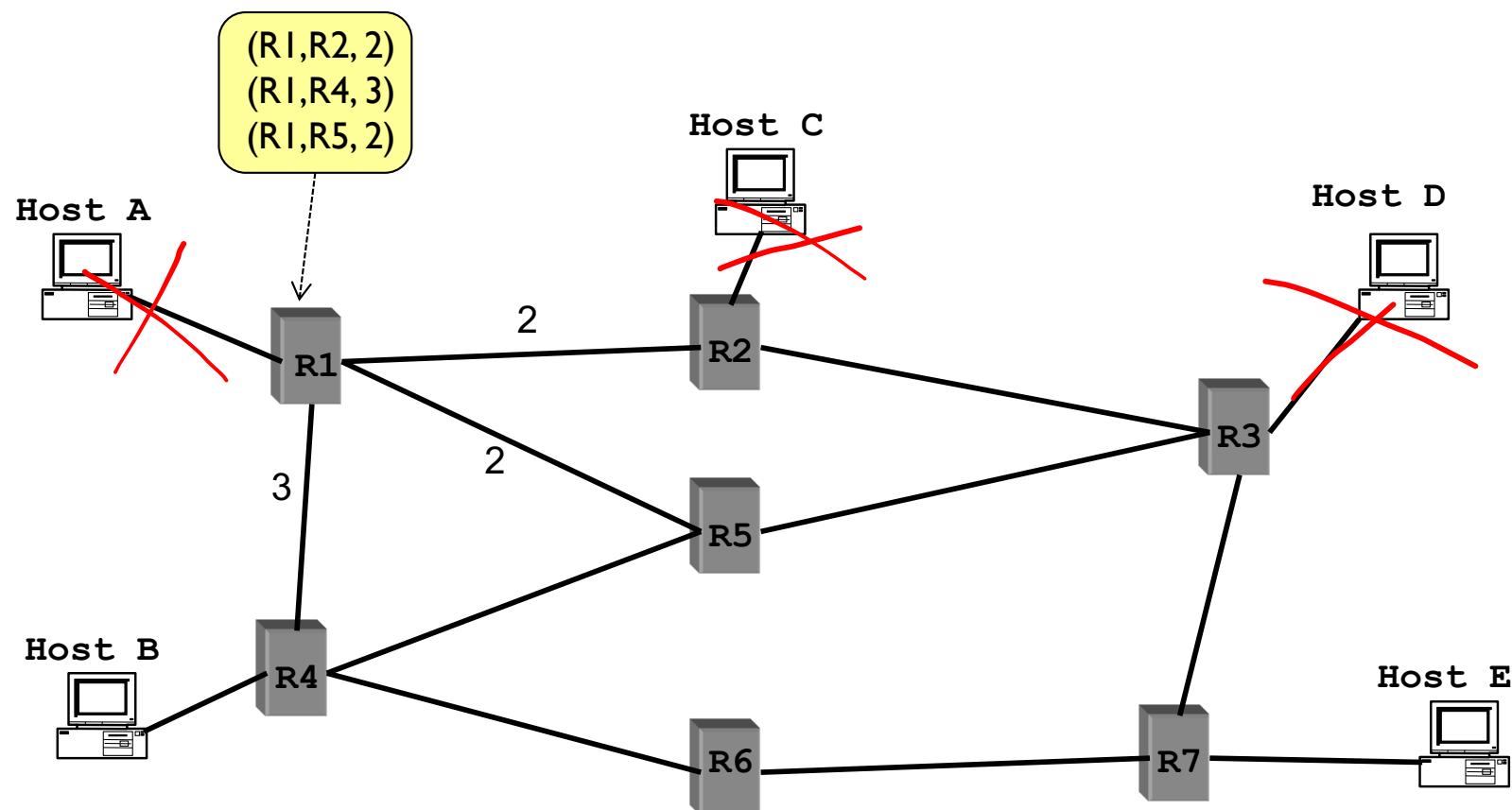
- Routers maintain cost of each link in the network
- Connectivity/cost changes flooded to all routers
- Converges quickly (less inconsistency, looping, etc.)
- Limited network sizes

## *Distance Vector (Decentralised)*

- Routers maintain next hop & cost of each destination.
- Connectivity/cost changes iteratively propagate from neighbour to neighbour
- Requires multiple rounds to converge
- Scales to large networks

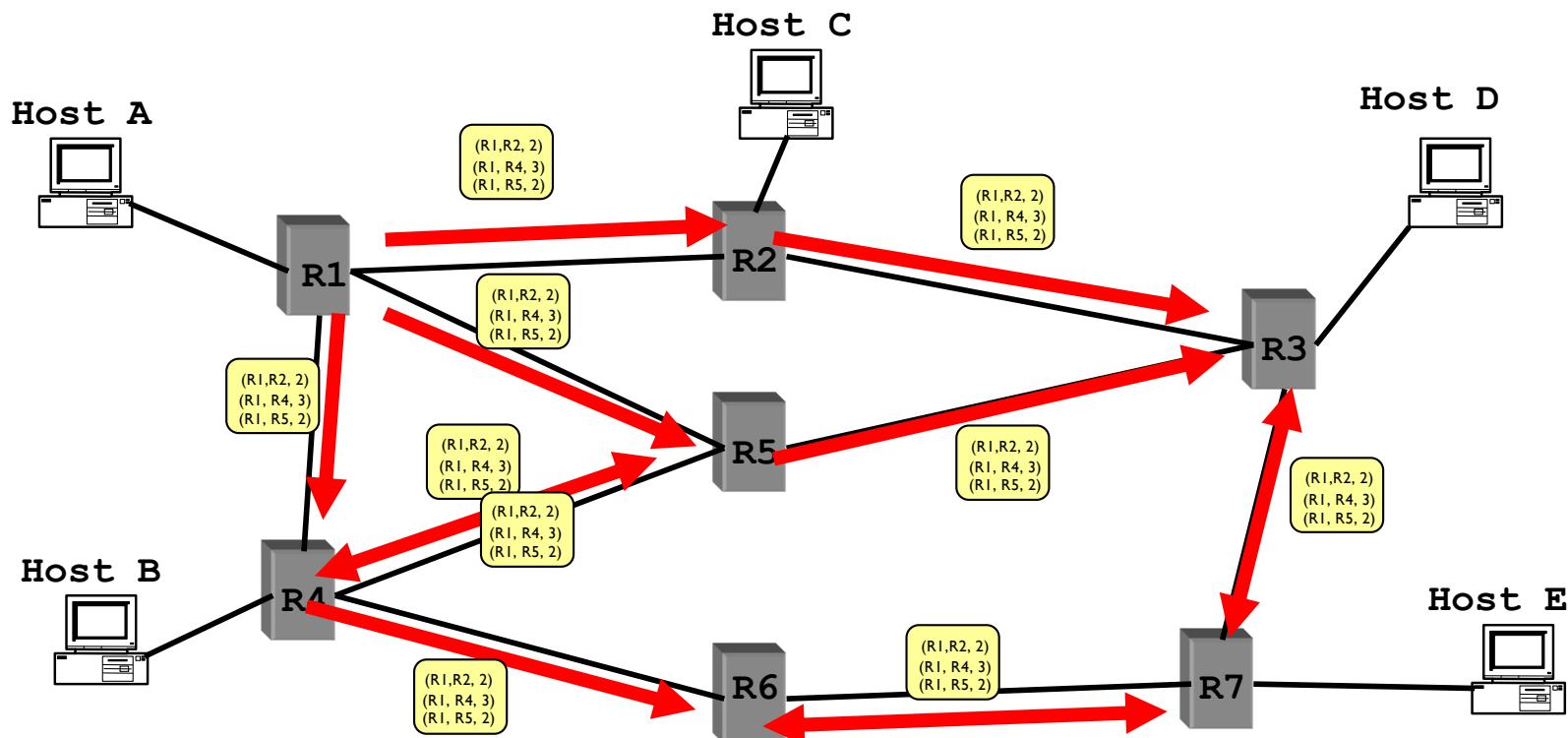
# Link State Routing

- ❖ Each node maintains its **local** “link state” (LS)
  - i.e., a list of its directly attached links and their costs



# Link State Routing

- ❖ Each node maintains its local “link state” (LS)
- ❖ Each node floods its local link state
  - on receiving a **new LS** message, a router forwards the message to all its neighbors other than the one it received the message from

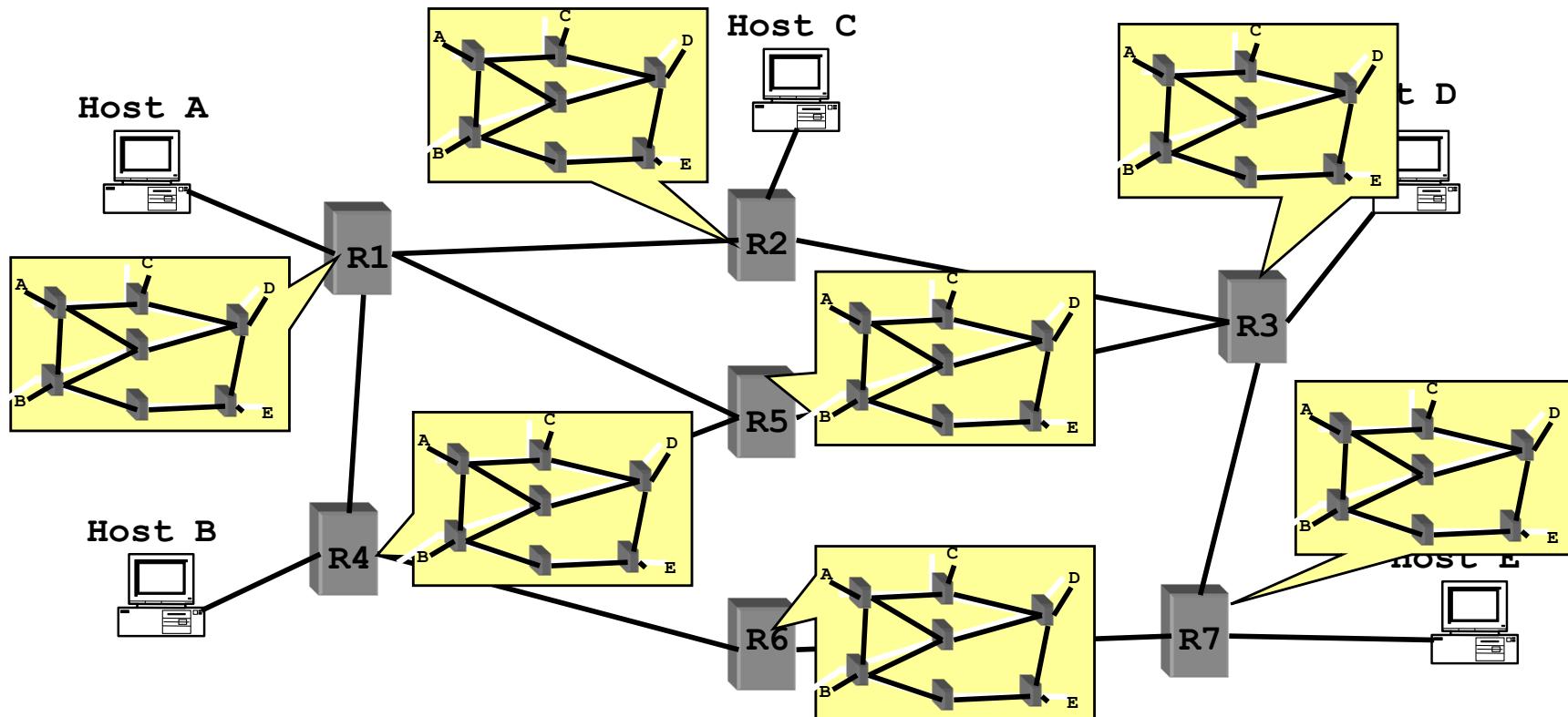


# Flooding LSAs

- ❖ Routers transmit **Link State Advertisement (LSA)** on links
  - A neighbouring router forwards out on all links except incoming
  - Keep a copy locally; don't forward previously-seen LSAs
- ❖ Challenges
  - Packet loss
  - Out of order arrival
- ❖ Solutions
  - Acknowledgements and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

# Link State Routing

- ❖ Each node maintains its local “link state” (LS)
- ❖ Each node floods its local link state
- ❖ Eventually, each node learns the entire network topology
  - Can use Dijkstra’s to compute the shortest paths between nodes



# 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:*

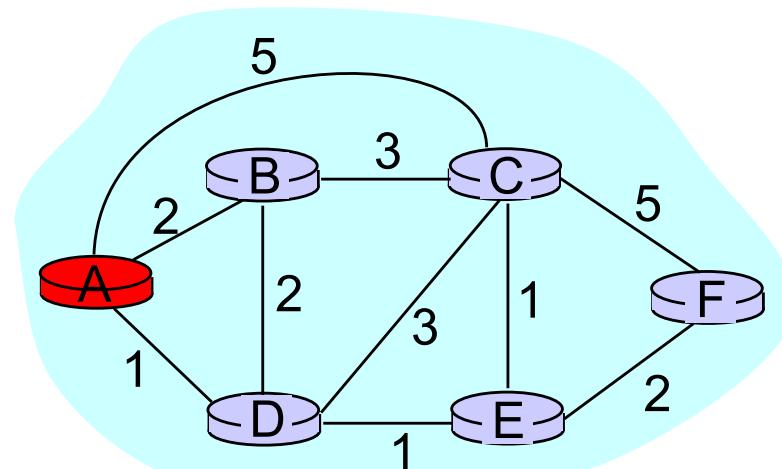
- ❖  $c(x,y)$ : link cost from node  $x$  to  $y$ ;  $= \infty$  if not direct neighbors
- ❖  $D(v)$ : current value of cost of path from source to dest.  $v$
- ❖  $p(v)$ : predecessor node along path from source to  $v$
- ❖  $N'$ : set of nodes whose least cost path definitively known

# 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  $N'$ 
```

# Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1						
2						
3						
4						
5						

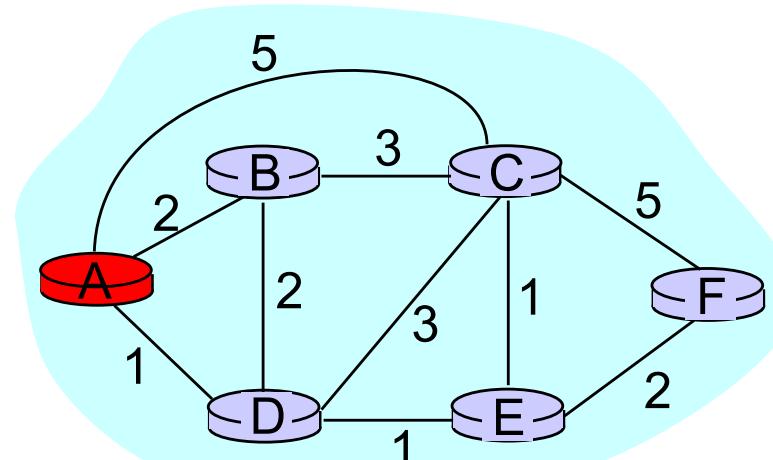


```
1 Initialization:
2    $N' = \{A\};$ 
3   for all nodes  $v$ 
4     if  $v$  adjacent to  $A$ 
5       then  $D(v) = c(A,v);$ 
6     else  $D(v) = \infty;$ 
...

```

# Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1						
2						
3						
4						
5						



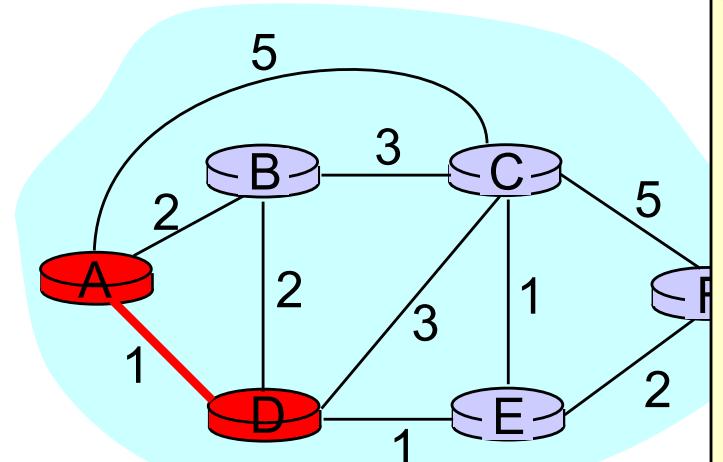
```

...
8 Loop
9   find w not in  $N'$  s.t.  $D(w)$  is a minimum;
10  add w to  $N'$ ;
11  update  $D(v)$  for all v adjacent
      to w and not in  $N'$ :
12  If  $D(w) + c(w,v) < D(v)$  then
13     $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;
14 until all nodes in  $N'$ ;

```

# Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD					
2						
3						
4						
5						



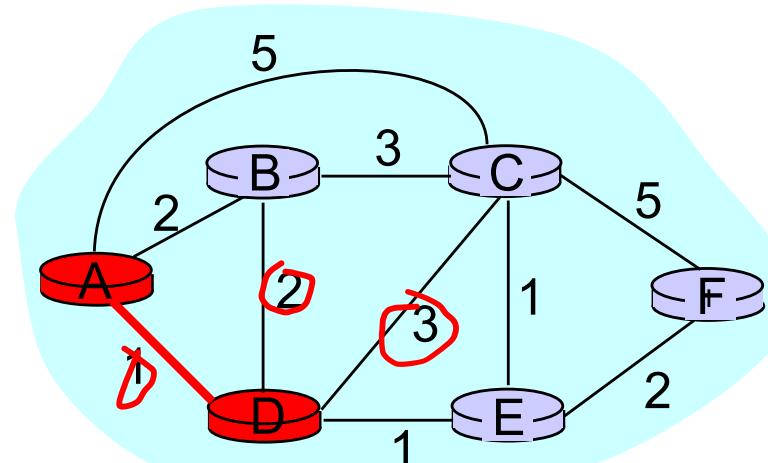
```

...
8  Loop
9    find w not in N' s.t. 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   If D(w) + c(w,v) < D(v) then
13     D(v) = D(w) + c(w,v); p(v) = w;
14   until all nodes in N';

```

# Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	0,A	4,D	X	2,D	
2						
3						
4						
5						



```

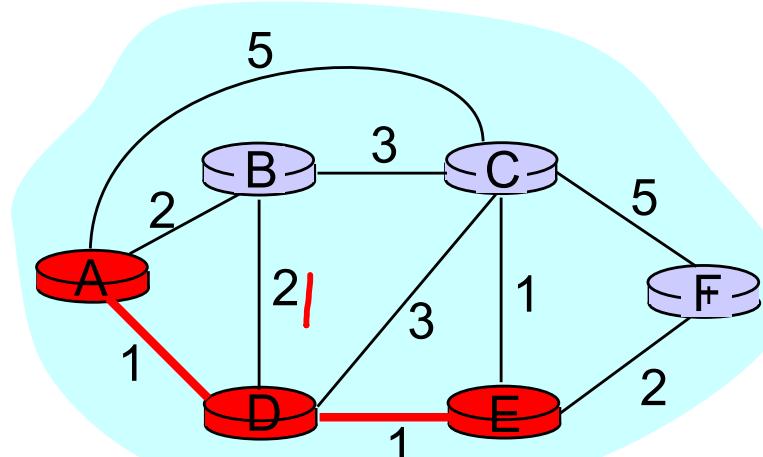
...
8  Loop
9  find w not in N' s.t. D(w) is a minimum;
10 add w to N';
11 update D(v) for all v adjacent
     to w and not in N';
12 If D(w) + c(w,v) < D(v) then
13   D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2, A	4,D			2,D
2	ADE	2, A	3,E			4,E
3						
4						
5						

→ Step 2: Set  $N' = \{A, D\}$



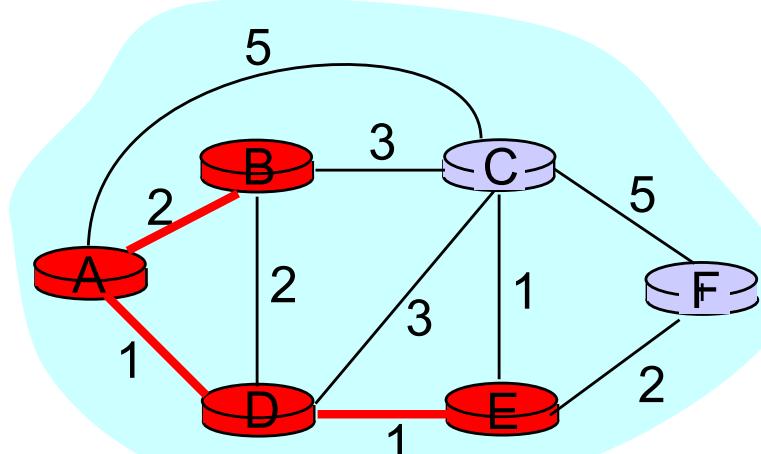
```

...
8 Loop
9   find w not in N' s.t. 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  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4						
5						



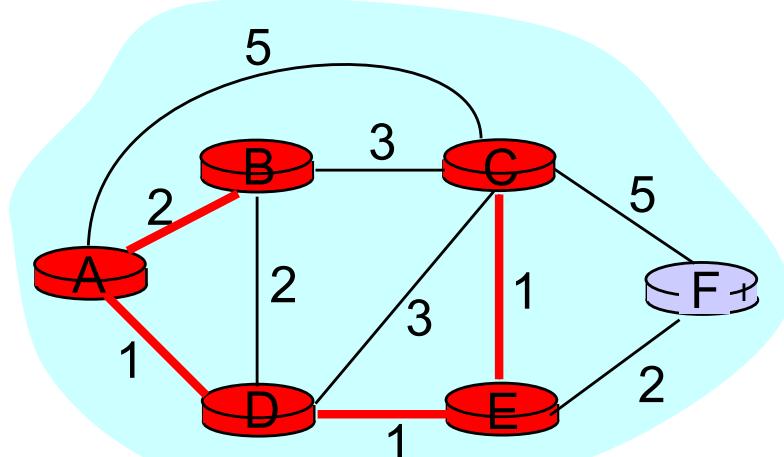
```

...
8 Loop
9   find w not in N' s.t. 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  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
→4	ADEBC					4,E
5						



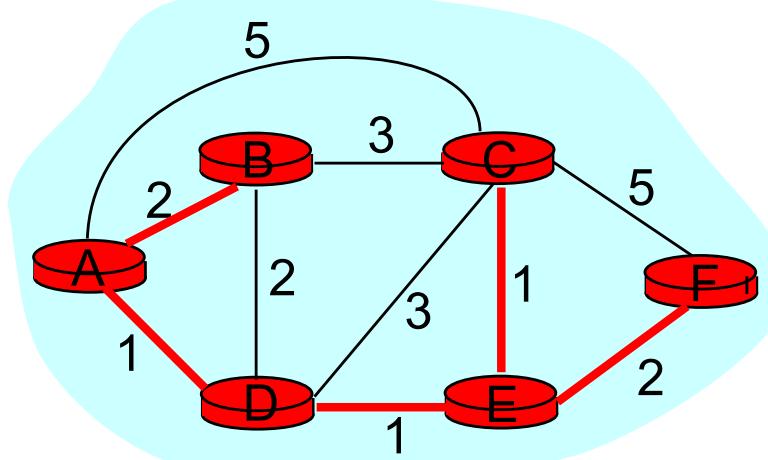
```

...
8 Loop
9   find w not in N' s.t. 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  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

# Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E		4,E	
3	ADEB		3,E		4,E	
4	ADEBC				4,E	
5	ADEBCF					



...

8 **Loop**

9 find  $w$  not in  $N'$  s.t.  $D(w)$  is a minimum;

10 add  $w$  to  $N'$ ;

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

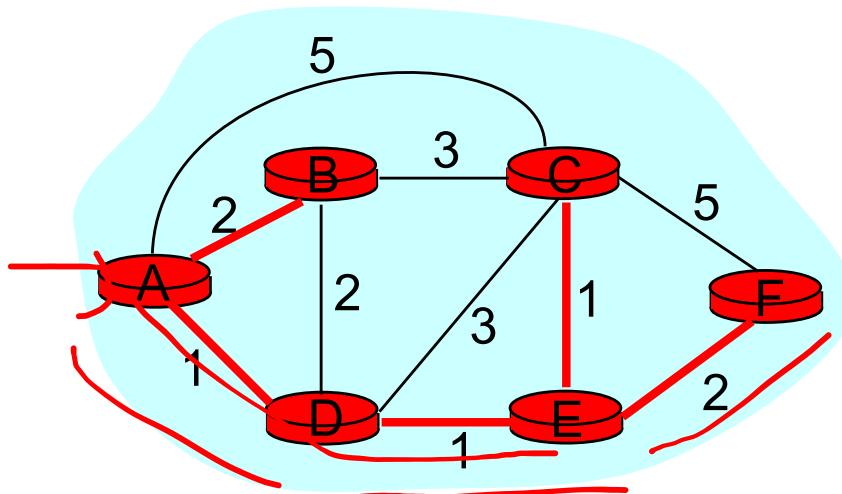
12 If  $D(w) + c(w,v) < D(v)$  then

13      $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;

14 **until all nodes in  $N'$ ;**

# Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD		4,D		2,D	
2	ADE			3,E		4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					

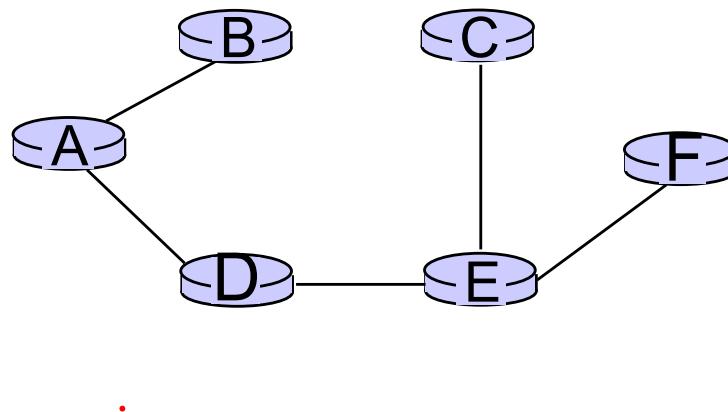


To determine path  $A \rightarrow C$  (say), work backward from C via  $p(v)$

# The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the *forwarding table*

resulting shortest-path tree from A:



Destination	Link <i>Next</i>
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

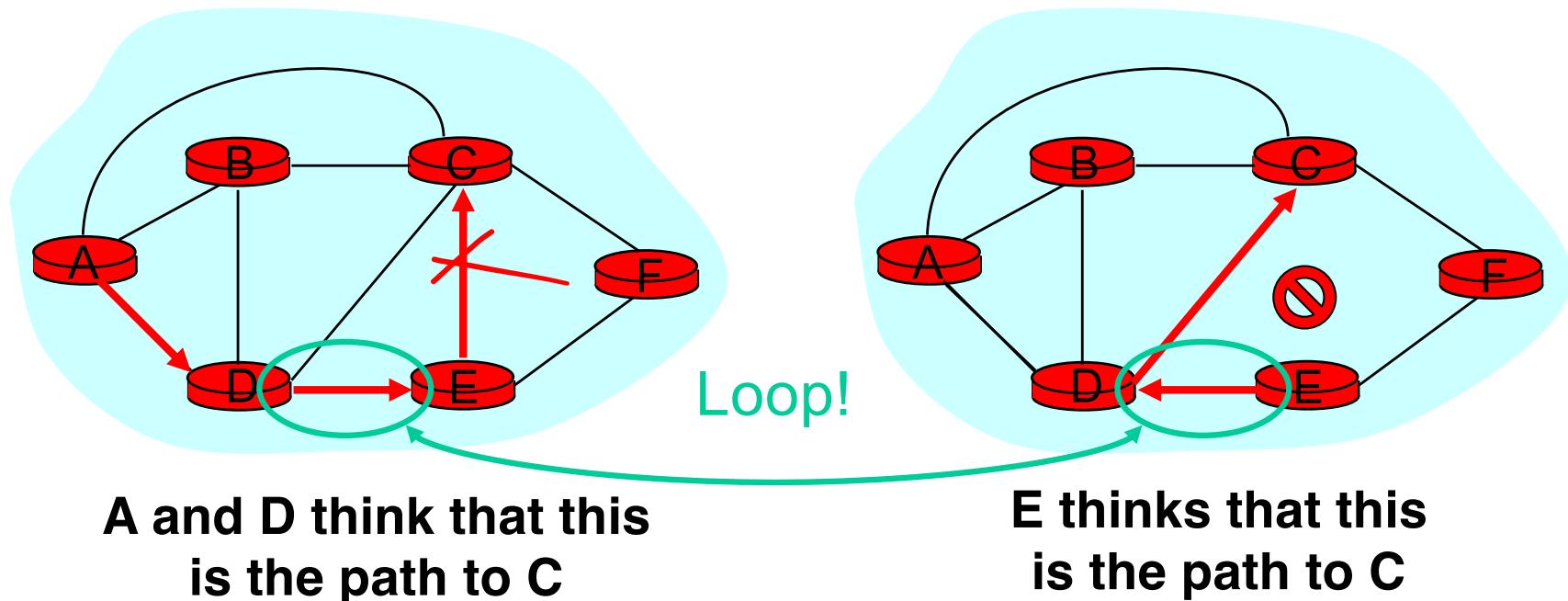
# Issue #1: Scalability

- ❖ How many messages needed to flood link state messages?
  - $O(N \times E)$ , where N is #nodes; E is #edges in graph
- ❖ Processing complexity for Dijkstra's algorithm?
  - $\underline{O(N^2)}$ , because we check all nodes w not in  $N'$  at each iteration and we have  $O(N)$  iterations
- ❖ How many entries in the LS topology database?  $O(E)$
- ❖ How many entries in the forwarding table?  $O(N)$

# Issue#2: Transient Disruptions

---

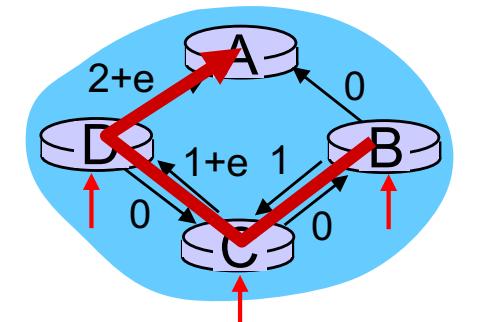
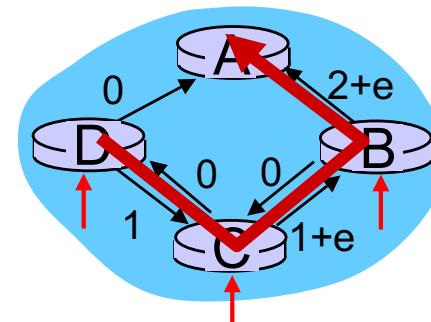
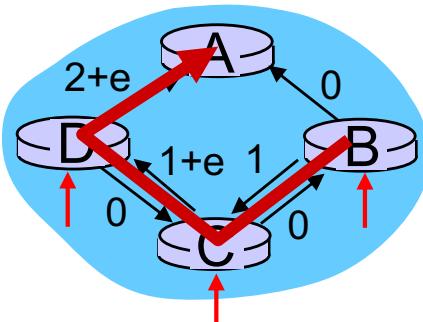
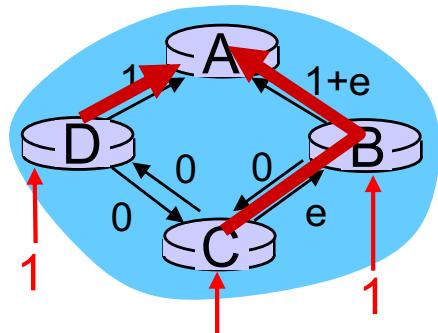
- ❖ Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause transient forwarding loops



# Oscillations

*oscillations possible:*

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



# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- ❖ link state

- ❖ distance vector

- ❖ hierarchical routing

5.6 ICMP: The Internet  
Control Message  
Protocol

# Distance vector algorithm

*Bellman-Ford equation*

let

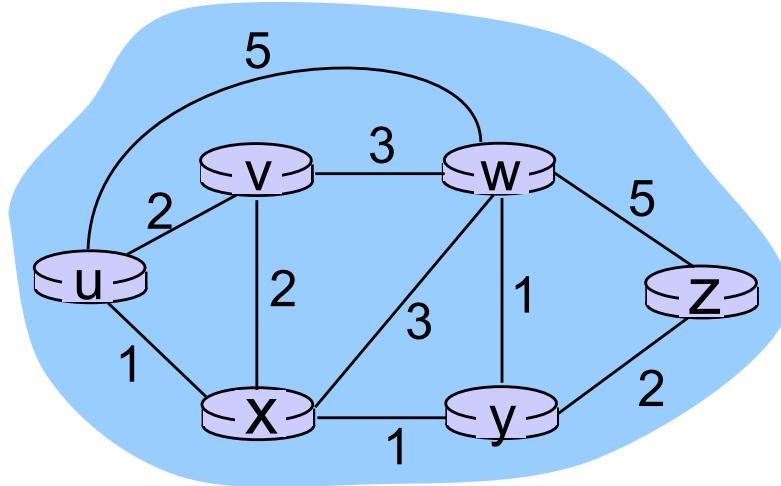
$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$

then

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

 $v$             cost from neighbor  $v$  to destination  $y$   
cost to neighbor  $v$   
 $\min$  taken over all neighbors  $v$  of  $x$

# Bellman-Ford example



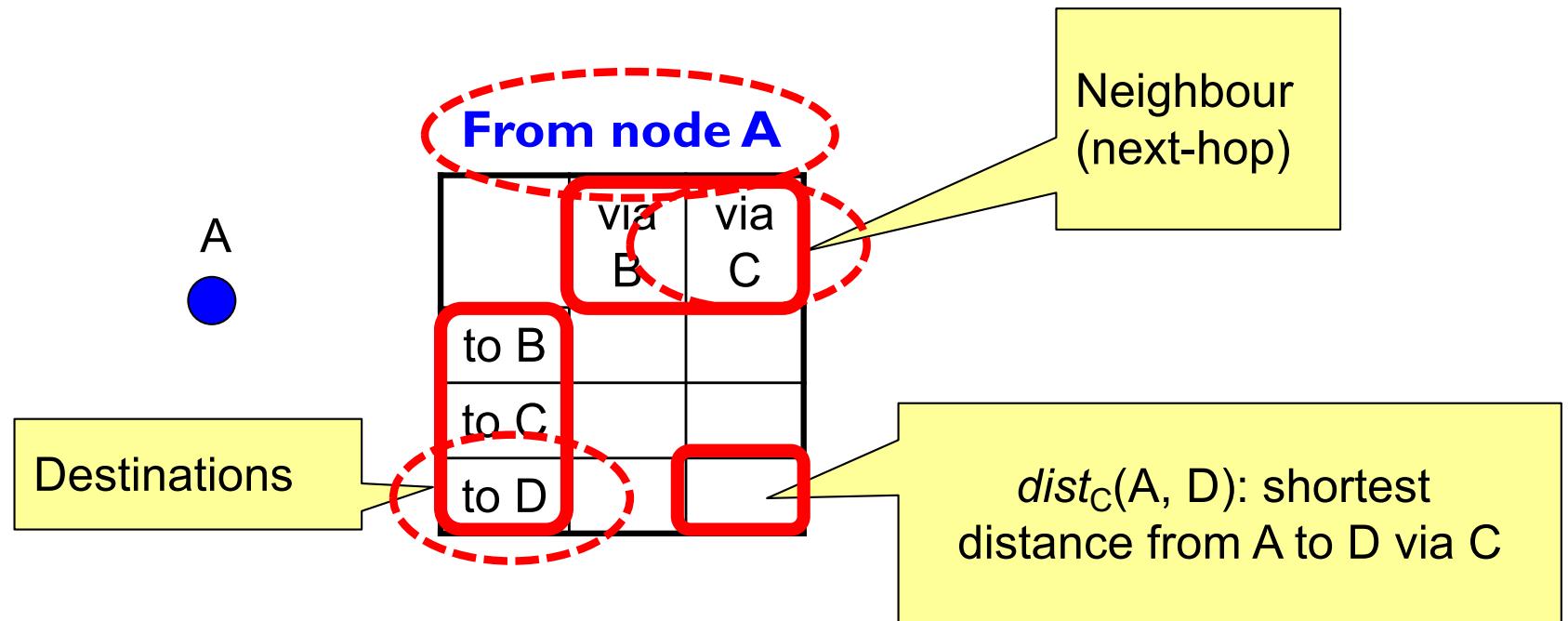
clearly,  $d_v(z) = 5$ ,  $d_x(z) = 3$ ,  $d_w(z) = 3$

B-F equation says:

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

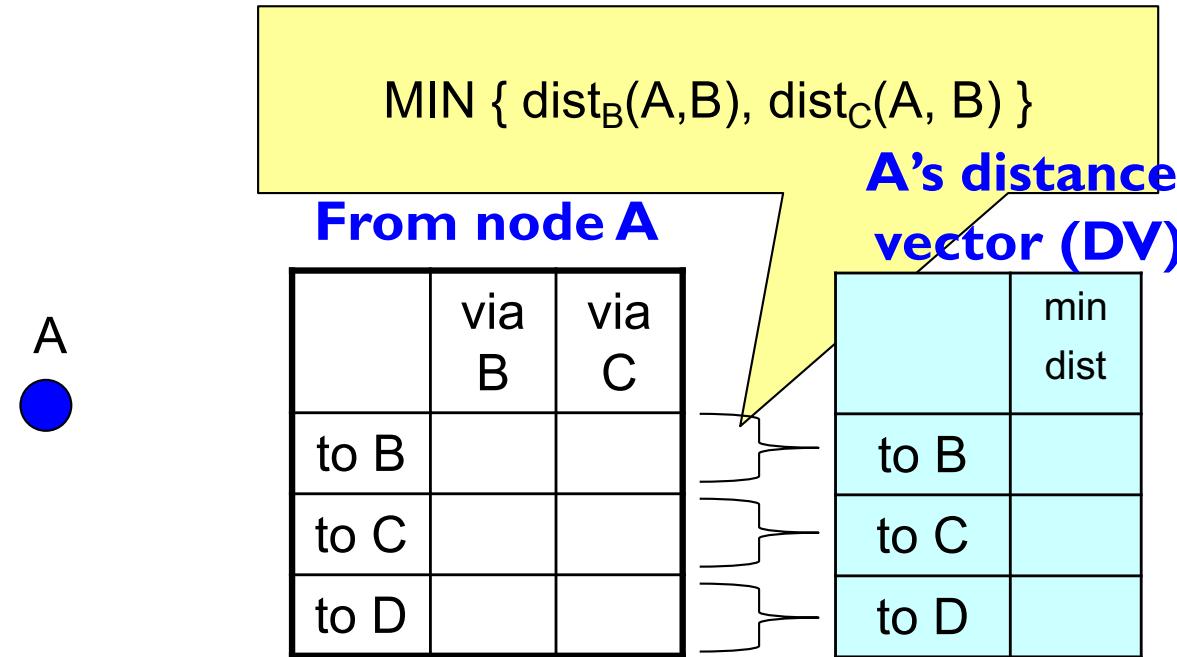
node achieving minimum is next  
hop in shortest path, used in forwarding table

# How Distance-Vector (DV) works



Each router maintains its shortest distance to every destination via each of its neighbours

# How Distance-Vector (DV) works



Each router computes its shortest distance to every destination via any of its neighbors

# How Distance-Vector (DV) works

A  
●

**From node A**

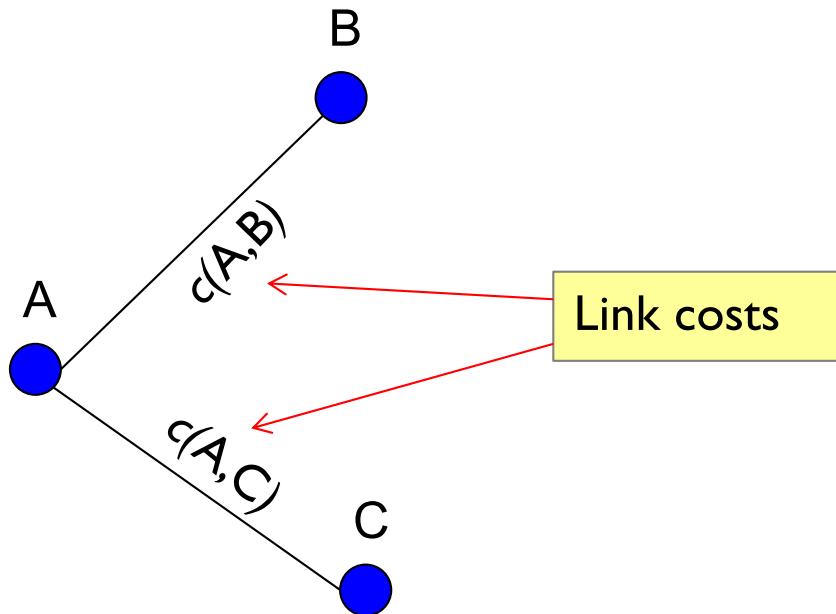
	via B	via C
to B	?	?
to C	?	?
to D	?	?

**A's DV**

	min dist
to B	?
to C	?
to D	?

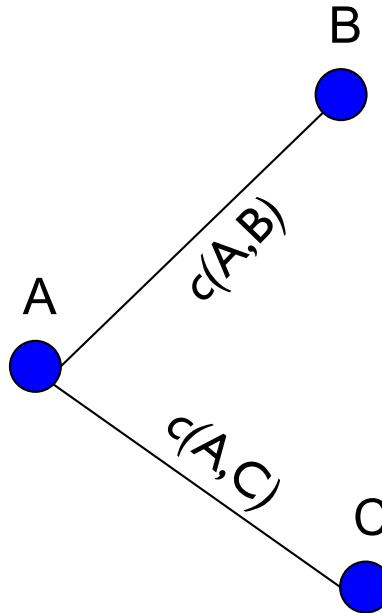
How does A initialize its dist() table and DV?

# How Distance-Vector (DV) works



How does A initialize its `dist()` table and DV?

# How Distance-Vector (DV) works



From node A

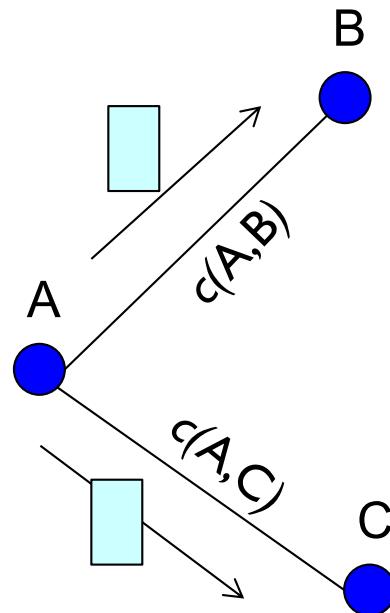
	via B	via C
to B	$c(A,B)$	$\infty$
to C	$\infty$	$c(A,C)$
to D	$\infty$	$\infty$

A's DV

	mindist
to B	$c(A,B)$
to C	$c(A,C)$
to D	$\infty$

Each router initializes its  $dist()$  table based on its immediate neighbors and link costs

# How Distance-Vector (DV) works



From node A

	via B	via C
to B	$c(A,B)$	$\infty$
to C	$\infty$	$c(A,C)$
to D	$\infty$	$\infty$

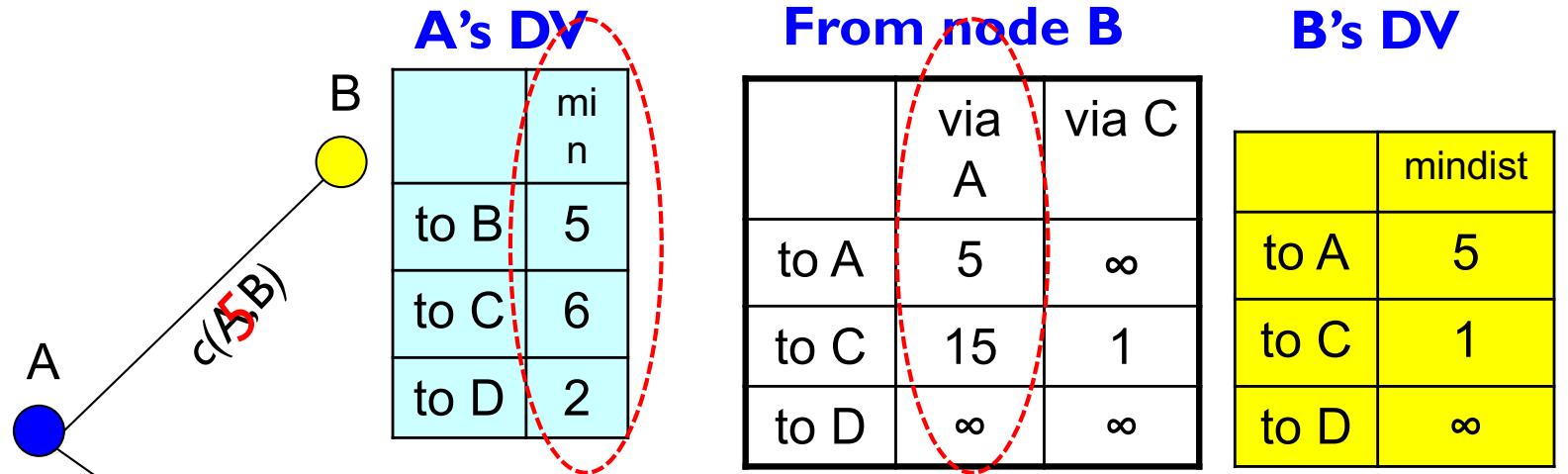
Assume that A's DV  
is as follows at  
some later time

A's DV

	mindist
to B	5
to C	6
to D	2

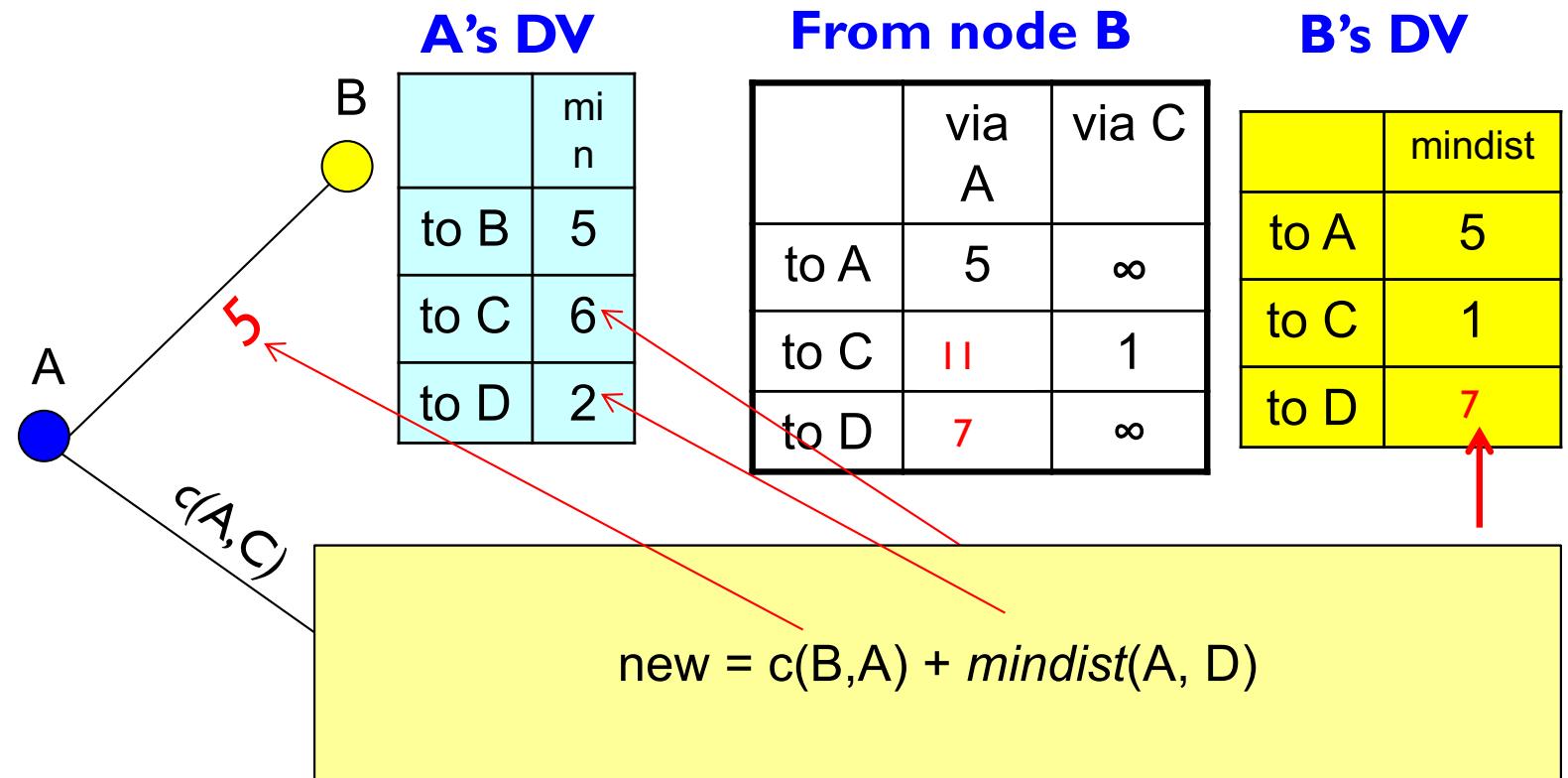
Each router sends its DV to its immediate neighbors

# How Distance-Vector (DV) works



Routers process received DVs

# How Distance-Vector (DV) works



Routers process received DVs

And repeat...

# Distance Vector Routing

- ❖ Each router knows the links to its neighbors
- ❖ Each router has provisional “shortest path” to **every** other router -- its **distance vector (DV)**
- ❖ Routers exchange this DV with their neighbors
- ❖ Routers look over the set of options offered by their neighbors and select the best one
- ❖ Iterative process converges to set of shortest paths

# Distance vector routing

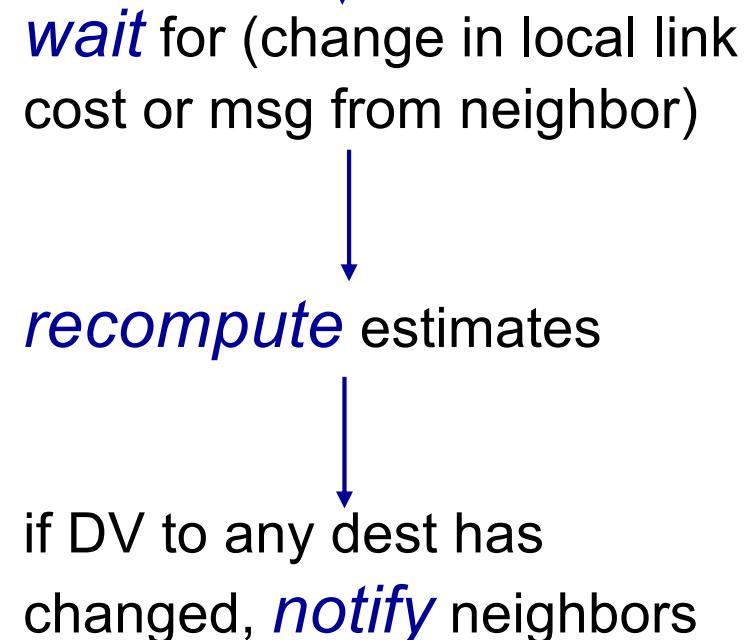
*iterative, asynchronous:*

- 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:*



# Distance Vector

- ❖  $c(i,j)$ : link cost from node  $i$  to  $j$
- ❖  $\text{dist}_z(A,V)$ : shortest dist. from  $A$  to  $V$  via  $Z$
- ❖  $\text{mindist}(A,V)$ : shortest dist. from  $A$  to  $V$

## 0 At node A

1 **Initialization:**

```
2 for all destinations V do
3     if V is neighbor of A
4          $\text{dist}_V(A, V) = \text{mindist}(A, V) = c(A, V);$ 
5     else
6          $\text{dist}_V(A, V) = \text{mindist}(A, V) = \infty;$ 
7     send  $\text{mindist}(A, *)$  to all neighbors
```

*loop:*

```
8 wait (until A sees a link cost change to neighbor V /* case 1 */
9     or until A receives  $\text{mindist}(V, *)$  from neighbor V) /* case 2 */
10 if ( $c(A, V)$  changes by  $\pm d$ ) /*  $\Leftarrow$  case 1 */
11     for all destinations Y do
12          $\text{dist}_Y(A, Y) = \text{dist}_V(A, Y) \pm d$ 
13 else /*  $\Leftarrow$  case 2: */
14     for all destinations Y do
15          $\text{dist}_Y(A, Y) = c(A, V) + \text{mindist}(V, Y);$ 
16 update  $\text{mindist}(A, *)$ 
15 if (there is a change in  $\text{mindist}(A, *)$ )
16     send  $\text{mindist}(A, *)$  to all neighbors
17 forever
```

# Distance Vector

- ❖  $c(i,j)$ : link cost from node  $i$  to  $j$
- ❖  $\text{dist}_Z(A,V)$ : shortest dist. from  $A$  to  $V$  via  $Z$
- ❖  $\text{mindist}(A,V)$ : shortest dist. from  $A$  to  $V$

## 0 At node A

1 **Initialization:**

```
2 for all destinations  $V$  do
3     if  $V$  is neighbor of  $A$ 
4          $\text{dist}_V(A, V) = \text{mindist}(A, V) = c(A, V);$ 
5     else
6          $\text{dist}_V(A, V) = \text{mindist}(A, V) = \infty;$ 
7     send  $\text{mindist}(A, *)$  to all neighbors
```

*loop:*

```
8 wait (until  $A$  sees a link cost change to neighbor  $V$  /* case 1 */
9     or until  $A$  receives  $\text{mindist}(V, *)$  from neighbor  $V$ ) /* case 2 */
10 if ( $c(A, V)$  changes by  $\pm d$ ) /*  $\Leftarrow$  case 1 */
11     for all destinations  $Y$  do
12          $\text{dist}_V(A, Y) = \text{dist}_V(A, Y) \pm d$ 
13 else /*  $\Leftarrow$  case 2: */
14     for all destinations  $Y$  do
15          $\text{dist}_V(A, Y) = c(A, V) + \text{mindist}(V, Y);$ 
16 update  $\text{mindist}(A, *)$ 
15 if (there is a change in  $\text{mindist}(A, *)$ )
16     send  $\text{mindist}(A, *)$  to all neighbors
17 forever
```

# Example: Initialization

**from Node B**

	via A	via C	via D	min dist
to A	2	$\infty$	$\infty$	2
to B	-	-	-	0
to C	$\infty$	1	$\infty$	1
to D	$\infty$	$\infty$	3	3

**from Node D**

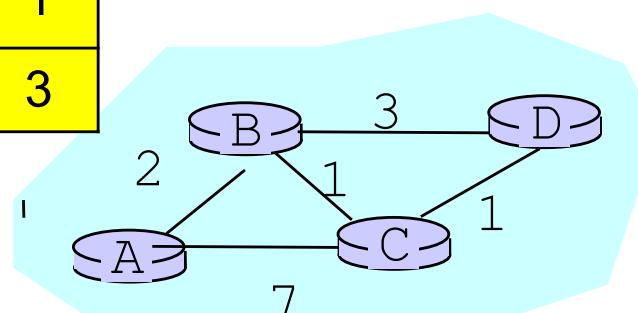
	via B	via C	min dist
to A	$\infty$	$\infty$	$\infty$
to B	3	$\infty$	3
to C	$\infty$	1	1
to D	-	-	0

**from Node A**

	via B	via C	min dist	min dist
to A	-	-	0	0
to B	2	$\infty$	2	2
to C	$\infty$	7	7	7
to D	$\infty$	$\infty$	$\infty$	$\infty$

**from Node C**

	via A	via B	via D	min dist
to A	7	$\infty$	$\infty$	7
to B	$\infty$	1	$\infty$	1
to C	-	-	-	0
to D	$\infty$	$\infty$	1	1



# Example: C sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	$\infty$	$\infty$	2
to B	-	-	-	0
to C	$\infty$	1	$\infty$	1
to D	$\infty$	$\infty$	3	3

from Node D

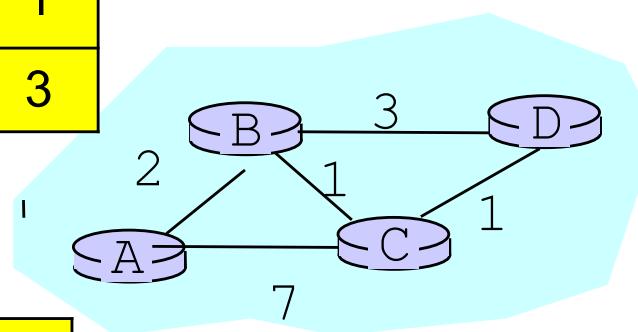
	via B	via C	min dist
to A	$\infty$	$\infty$	$\infty$
to B	3	$\infty$	3
to C	$\infty$	1	1
to D	-	-	0

from Node A

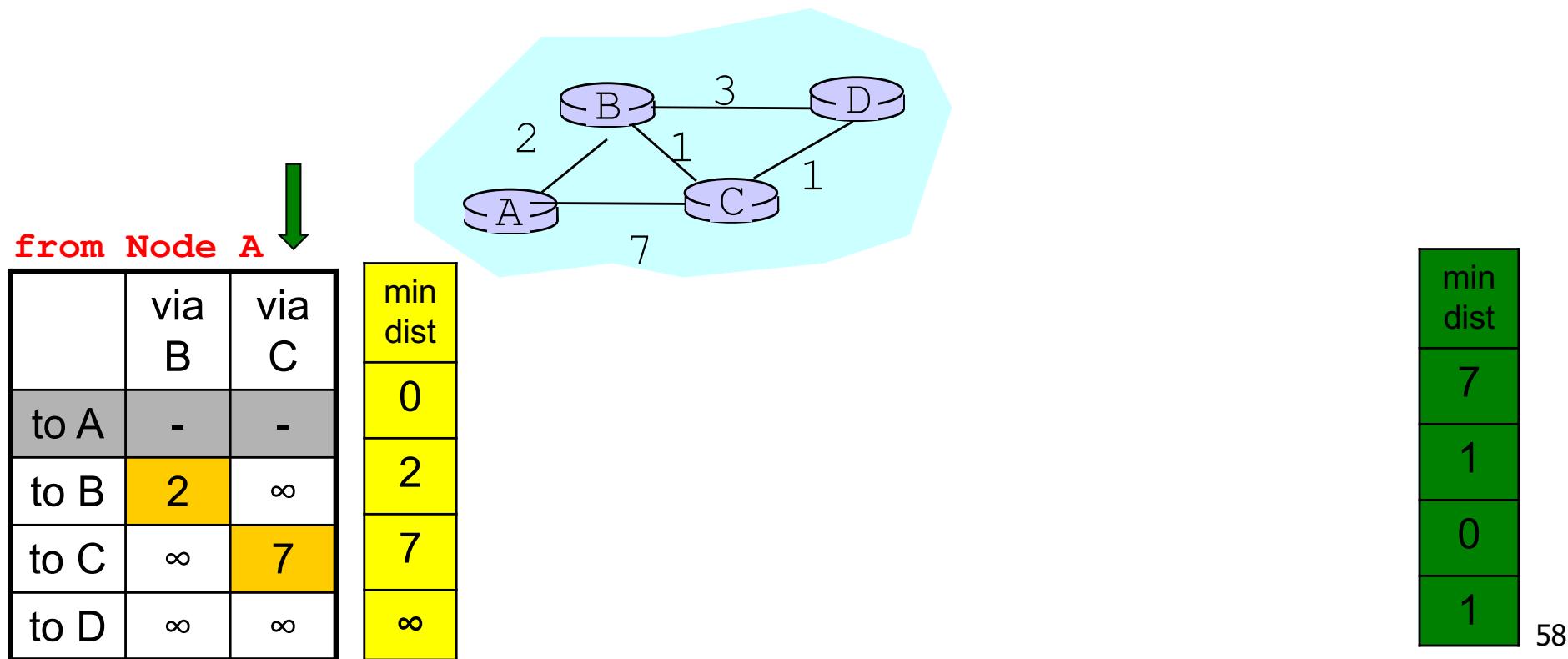
	via B	via C	min dist
to A	-	-	0
to B	2	$\infty$	2
to C	$\infty$	7	7
to D	$\infty$	$\infty$	$\infty$

from Node C

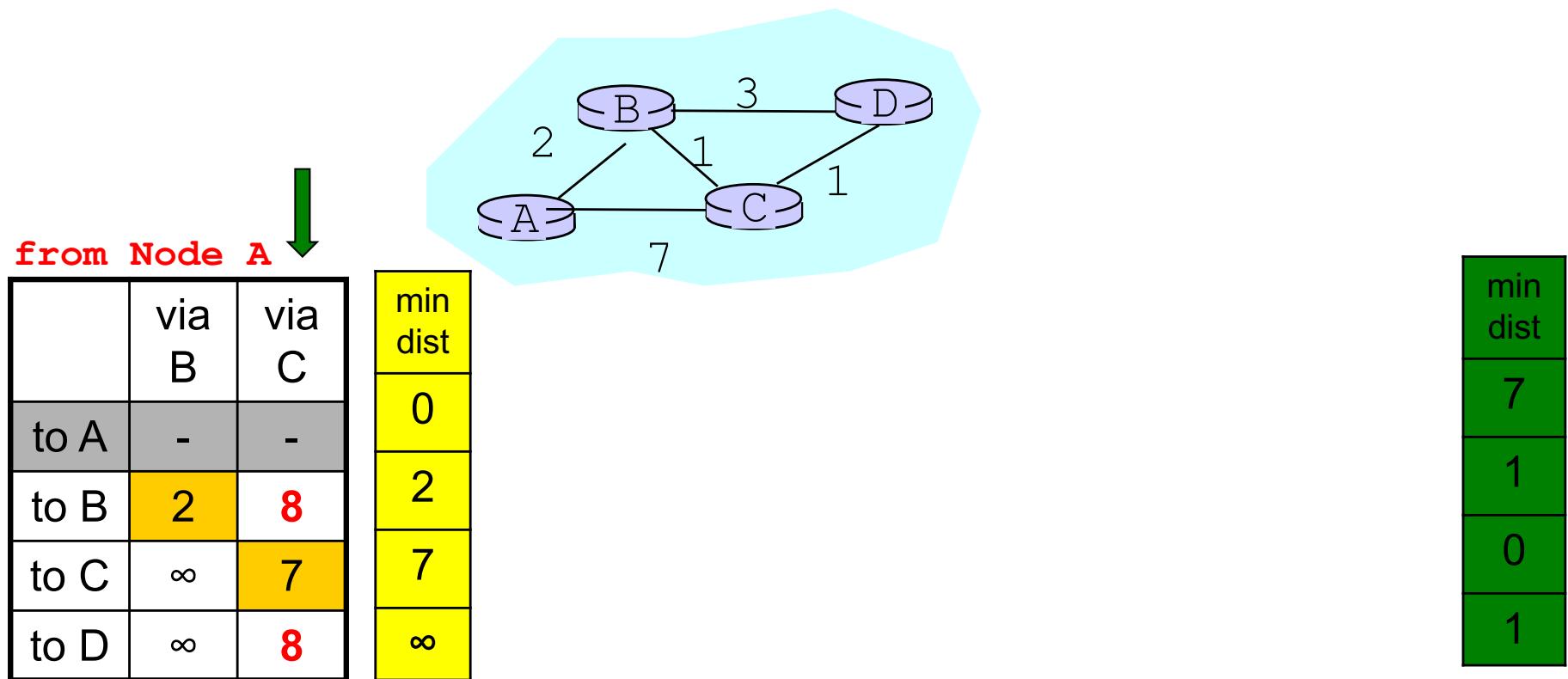
	via A	via B	via D	min dist
to A	7	$\infty$	$\infty$	7
to B	$\infty$	1	$\infty$	1
to C	-	-	-	0
to D	$\infty$	$\infty$	1	1



# Example: C sends update to A



# Example: C sends update to A



# Example: C sends update to A

from Node A

	via B	via C
to A	-	-
to B	2	8
to C	$\infty$	7
to D	$\infty$	8

min dist

0
2
7
8

# Example: now B sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	$\infty$	$\infty$	2
to B	-	-	-	0
to C	$\infty$	1	$\infty$	1
to D	$\infty$	$\infty$	3	3

from Node D

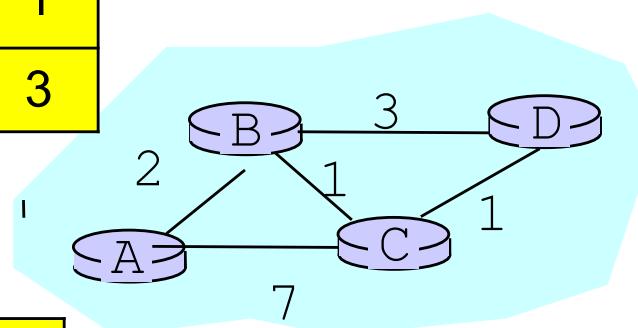
	via B	via C	min dist
to A	$\infty$	$\infty$	$\infty$
to B	3	$\infty$	3
to C	$\infty$	1	1
to D	-	-	0

from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	$\infty$	7	7
to D	$\infty$	8	8

from Node C

	via A	via B	via D	min dist
to A	7	$\infty$	$\infty$	7
to B	$\infty$	1	$\infty$	1
to C	-	-	-	0
to D	$\infty$	$\infty$	1	1



# Example: now B sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	$\infty$	$\infty$	2
to B	-	-	-	0
to C	$\infty$	1	$\infty$	1
to D	$\infty$	$\infty$	3	3

from Node D

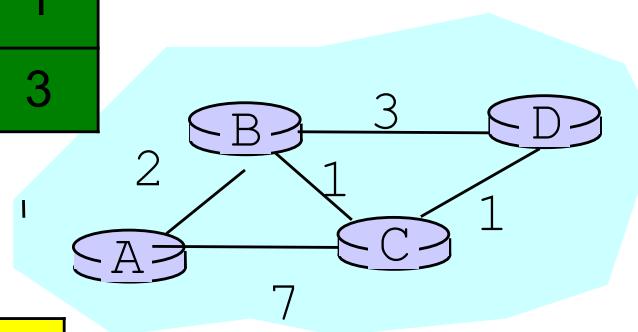
	via B	via C	min dist
to A	$\infty$	$\infty$	$\infty$
to B	3	$\infty$	3
to C	$\infty$	1	1
to D	-	-	0

from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	$\infty$	7	7
to D	$\infty$	8	8

from Node C

	via A	via B	via D	min dist
to A	7	$\infty$	$\infty$	7
to B	$\infty$	1	$\infty$	1
to C	-	-	-	0
to D	$\infty$	$\infty$	1	1



# Example: now B sends update to A

from Node B

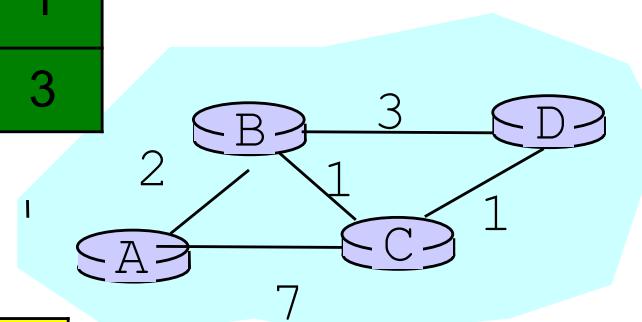
	via A	via C	via D	min dist
to A	2	$\infty$	$\infty$	2
to B	-	-	-	0
to C	$\infty$	1	$\infty$	1
to D	$\infty$	$\infty$	3	3

from Node D

	via B	via C	min dist
to A	$\infty$	$\infty$	$\infty$
to B	3	$\infty$	3
to C	$\infty$	1	1
to D	-	-	0

from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	7
to D	5	8	8



Make sure you know  
why this is 5, not 4!

from Node C

	via A	via B	via C	min dist
to A	$\infty$	$\infty$	$\infty$	7
to B	$\infty$	$\infty$	$\infty$	1
to C	-	-	-	0
to D	$\infty$	$\infty$	1	1

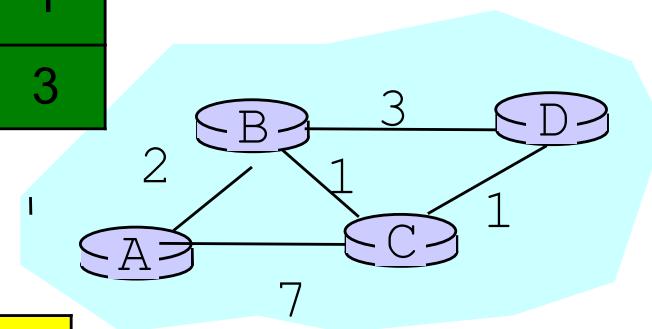
# Example: now B sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	$\infty$	$\infty$	2
to B	-	-	-	0
to C	$\infty$	1	$\infty$	1
to D	$\infty$	$\infty$	3	3

from Node D

	via B	via C	min dist
to A	$\infty$	$\infty$	$\infty$
to B	3	$\infty$	3
to C	$\infty$	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	5	8	5

from Node C

	via A	via B	via D	min dist
to A	7	$\infty$	$\infty$	7
to B	$\infty$	1	$\infty$	1
to C	-	-	-	0
to D	$\infty$	$\infty$	1	1

*All nodes know the best **two-hop** paths.*

Make sure you believe this

from Node B

	via A	via C	via D
to A	2	8	$\infty$
to B	-	-	-
to C	9	1	4
to D	$\infty$	2	3

min dist

2

0

1

2

from Node D

	via B	via C
to A	5	8
to B	3	2
to C	4	1
to D	-	-

min dist

5

2

1

0

from Node A

	via B	via C
to A	-	-
to B	2	8
to C	3	7
to D	5	8

min dist

0

2

3

5

from Node C

	via A	via B	via D
to A	7	3	$\infty$
to B	9	1	4
to C	-	-	-
to D	$\infty$	4	1

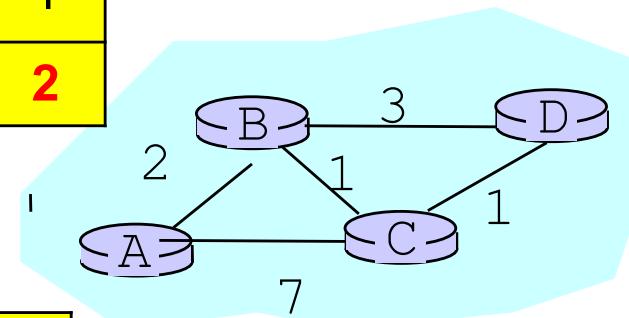
min dist

3

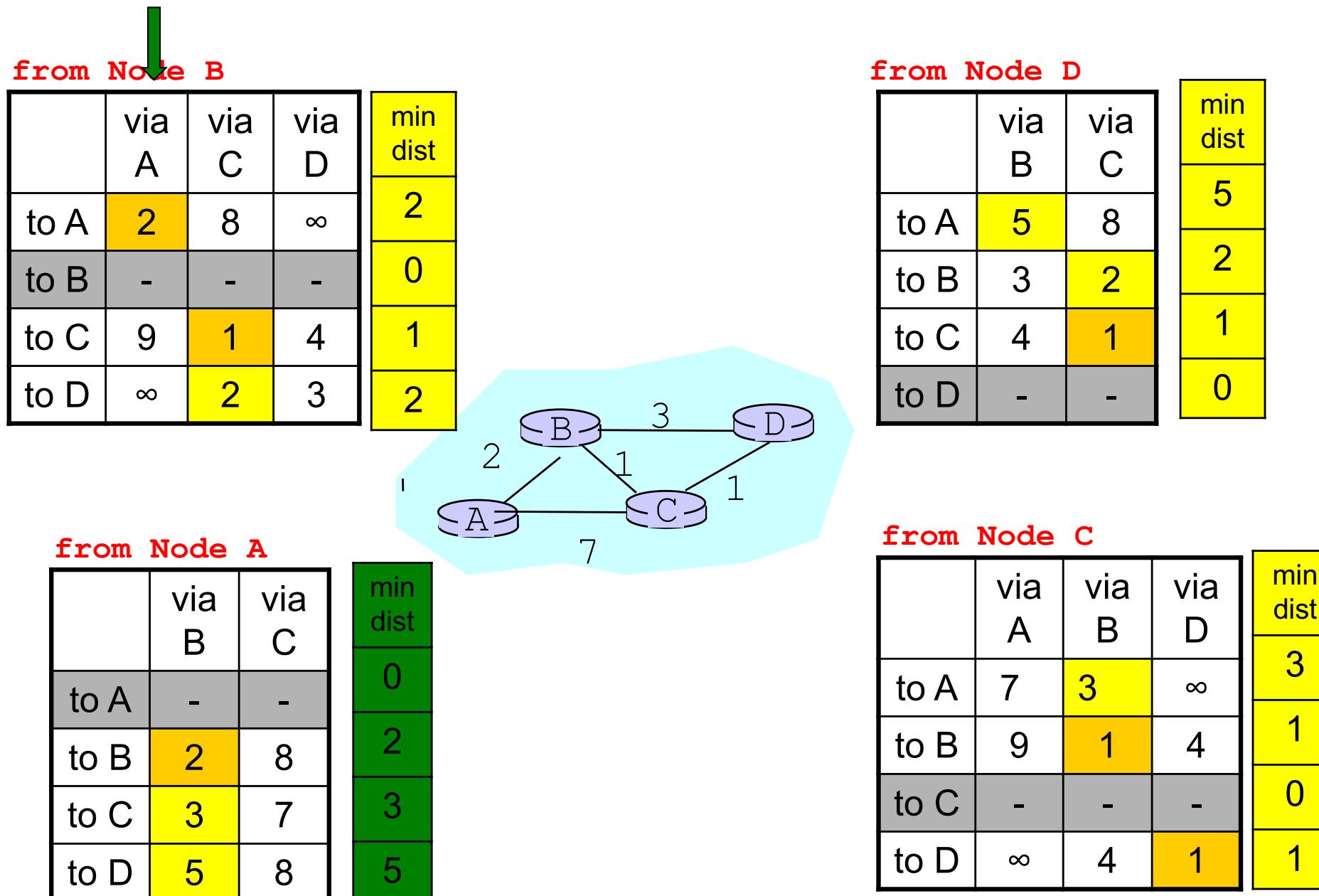
1

0

1



# Example: Now A sends update to B

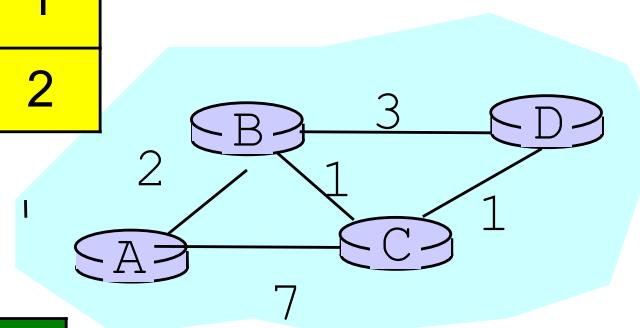


# Example: Novice

Updated

from Node B

	via A	via C	via D	min dist
to A	2	8	$\infty$	0
to B	-	-	-	1
to C	5	1	4	2
to D	7	2	3	2



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	5	8	5

from Node D

	via B	via C	min dist
to A	5	8	5
to B	3	2	2
to C	4	1	1
to D	-	-	0

from Node C

	via A	via B	via D	min dist
to A	7	3	$\infty$	3
to B	9	1	4	1
to C	-	-	-	0
to D	$\infty$	4	1	1

**Check:** All nodes know the best *three*-hop paths.

from Node B

	via A	via C	via D	min dist
to A	2	4	8	2
to B	-	-	-	0
to C	5	1	4	1
to D	7	2	3	2

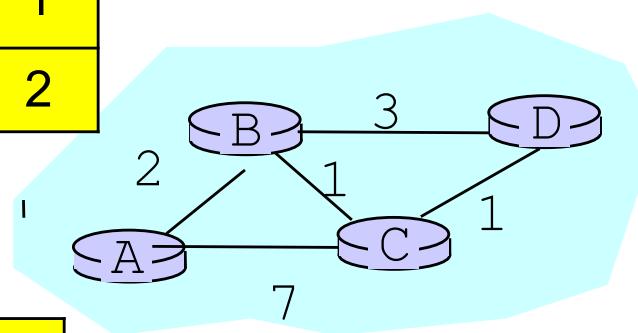
from Node D

	via B	via C	min dist
to A	5	4	4
to B	3	2	2
to C	4	1	1
to D	-	-	0

from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	4	8	4

Check



from Node C

	via A	via B	via D	min dist
to A	7	3	6	3
to B	9	1	3	1
to C	-	-	-	0
to D	12	3	1	1

# Example: End of 3<sup>nd</sup> Full Exchange

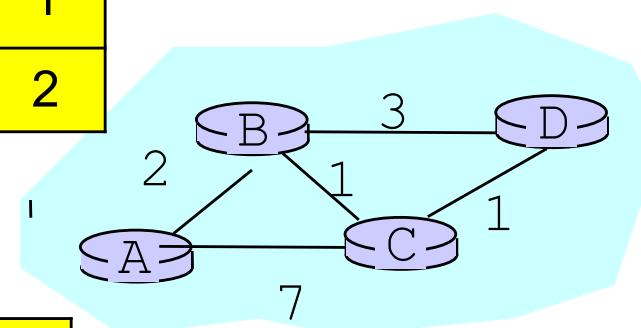
No further change in DVs → Convergence!

from Node B

	via A	via C	via D	min dist
to A	2	4	7	2
to B	-	-	-	0
to C	5	1	4	1
to D	6	2	3	2

from Node D

	via B	via C	min dist
to A	5	4	4
to B	3	2	2
to C	4	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	4	8	4

from Node C

	via A	via B	via D	min dist
to A	7	3	5	3
to B	9	1	3	1
to C	-	-	-	0
to D	11	3	1	1

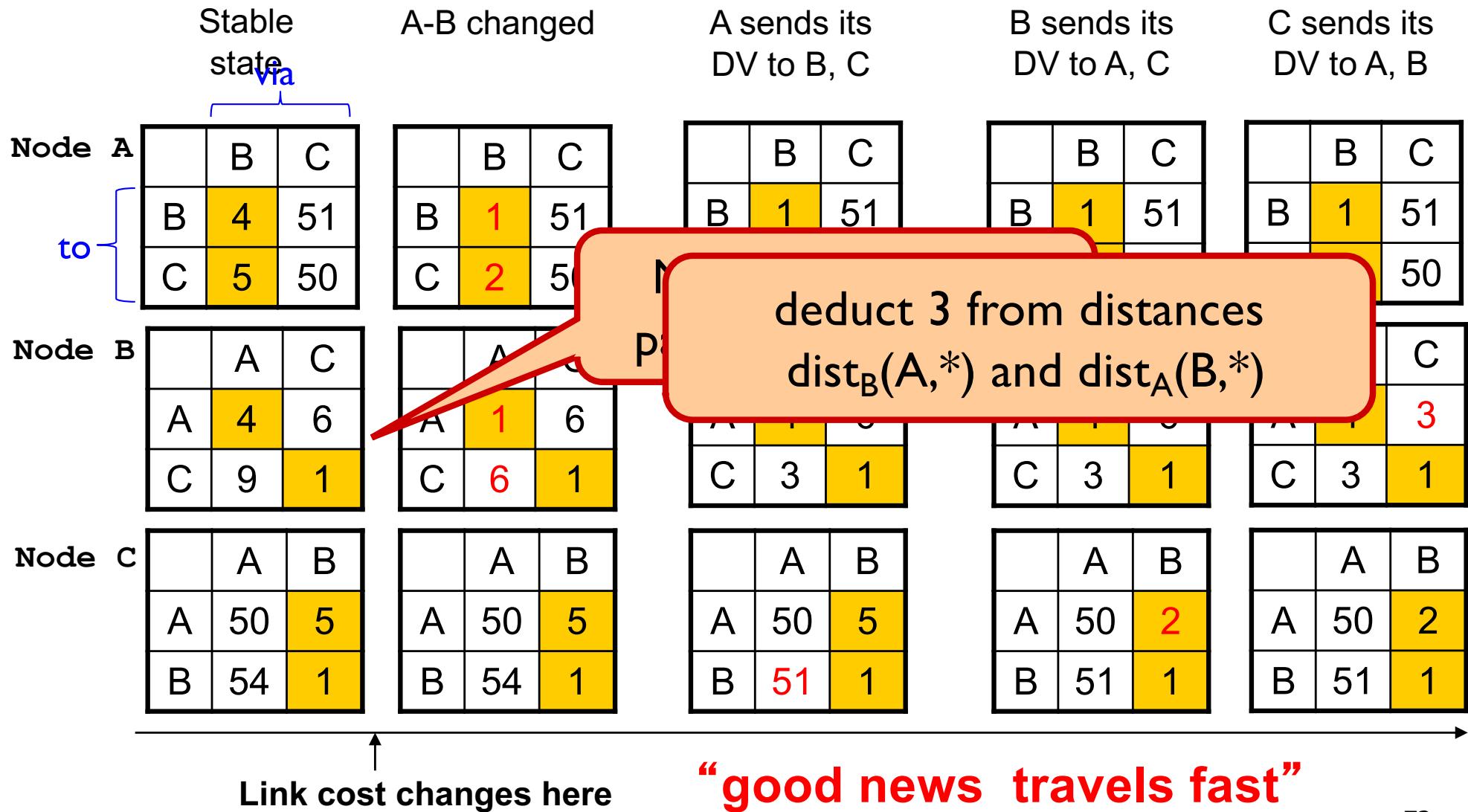
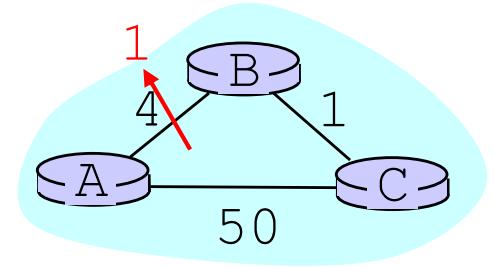
# Intuition

- ❖ Initial state: best one-hop paths
- ❖ One simultaneous round: best two-hop paths
- ❖ Two simultaneous rounds: best three-hop paths
- ❖ ...
- ❖ Kth simultaneous round: best  $(k+1)$  hop paths
  
- ❖ Must eventually converge
  - as soon as it reaches longest best path
- ❖ .....but how does it respond to changes in cost?

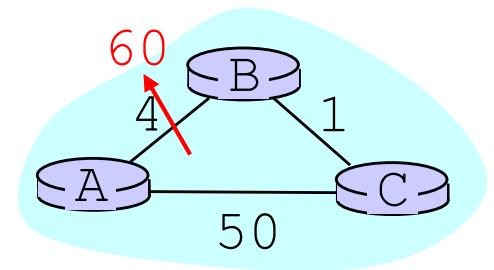
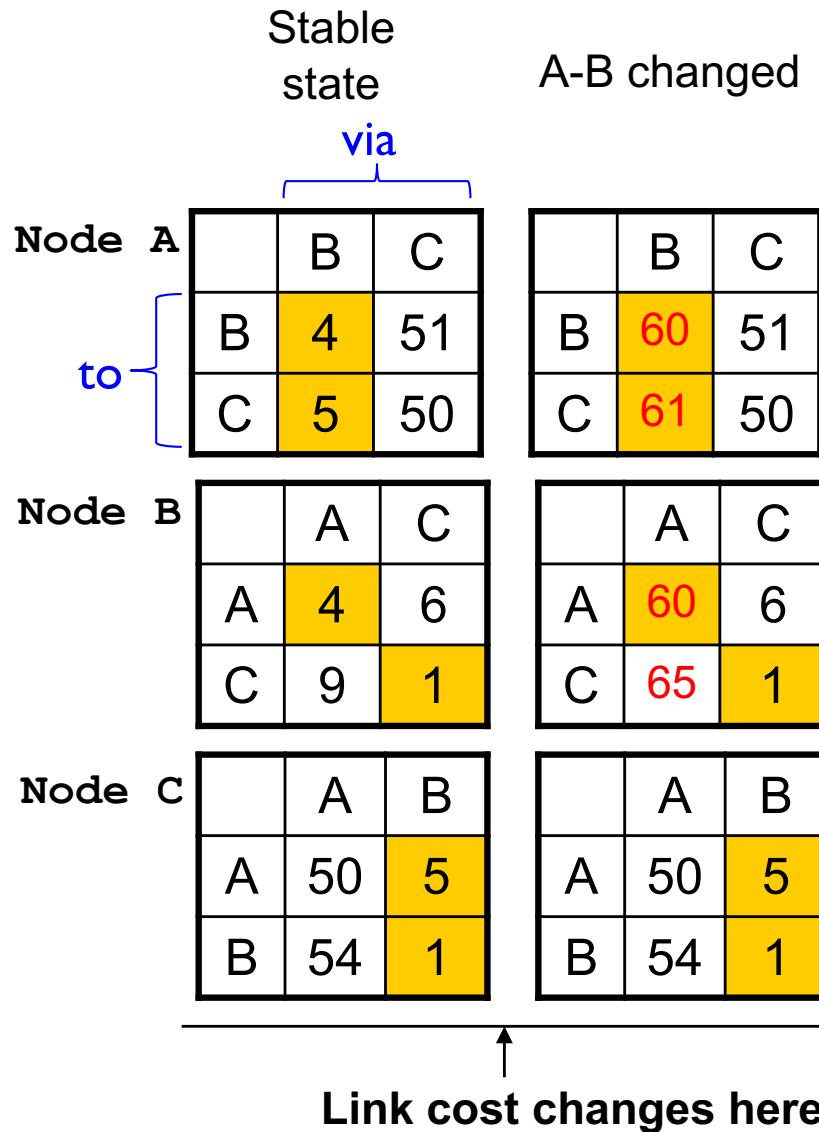
# Problems with Distance Vector

- A number of problems can occur in a network using distance vector algorithm
- Most of these problems are caused by slow convergence or routers converging on incorrect information
- *Convergence* is the time during which all routers come to an agreement about the best paths through the internetwork
  - whenever topology changes there is a period of instability in the network as the routers converge
- Reacts rapidly to good news, but leisurely to bad news

# DV: Link Cost Changes



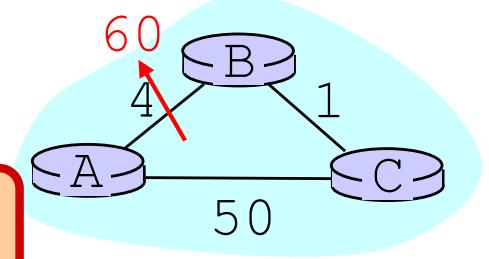
# DV: Link Cost Changes



add 56 to distances  
 $\text{dist}_B(A,*)$  and  $\text{dist}_A(B,*)$

# DV: Link Cost Changes

This is the “Counting to Infinity” Problem



	Stable state via to	A-B changed	A sends its DV to B, C	B sends its DV to A, C	C sends its DV to A, B																																													
Node A	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>4</td><td>51</td></tr> <tr><td>C</td><td>5</td><td>50</td></tr> </table>		B	C	B	4	51	C	5	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50
	B	C																																																
B	4	51																																																
C	5	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
Node B	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>4</td><td>6</td></tr> <tr><td>C</td><td>9</td><td>1</td></tr> </table>		A	C	A	4	6	C	9	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>6</td></tr> <tr><td>C</td><td>65</td><td>1</td></tr> </table>		A	C	A	60	6	C	65	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>6</td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>		A	C	A	60	6	C	110	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>6</td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>		A	C	A	60	6	C	110	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>8</td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>		A	C	A	60	8	C	110	1
	A	C																																																
A	4	6																																																
C	9	1																																																
	A	C																																																
A	60	6																																																
C	65	1																																																
	A	C																																																
A	60	6																																																
C	110	1																																																
	A	C																																																
A	60	6																																																
C	110	1																																																
	A	C																																																
A	60	8																																																
C	110	1																																																
Node C	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table>		A	B	A	50	5	B	54	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table>		A	B	A	50	5	B	54	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>101</td><td>1</td></tr> </table>		A	B	A	50	5	B	101	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>7</td></tr> <tr><td>B</td><td>101</td><td>1</td></tr> </table>		A	B	A	50	7	B	101	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>7</td></tr> <tr><td>B</td><td>101</td><td>1</td></tr> </table>		A	B	A	50	7	B	101	1
	A	B																																																
A	50	5																																																
B	54	1																																																
	A	B																																																
A	50	5																																																
B	54	1																																																
	A	B																																																
A	50	5																																																
B	101	1																																																
	A	B																																																
A	50	7																																																
B	101	1																																																
	A	B																																																
A	50	7																																																
B	101	1																																																

Link cost changes here

“bad news travels slowly”  
(not yet converged)

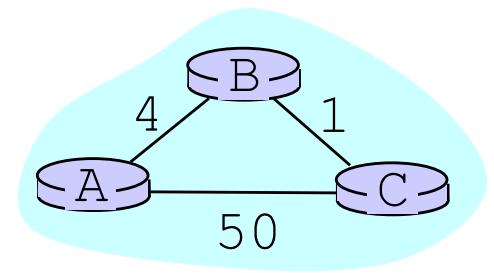
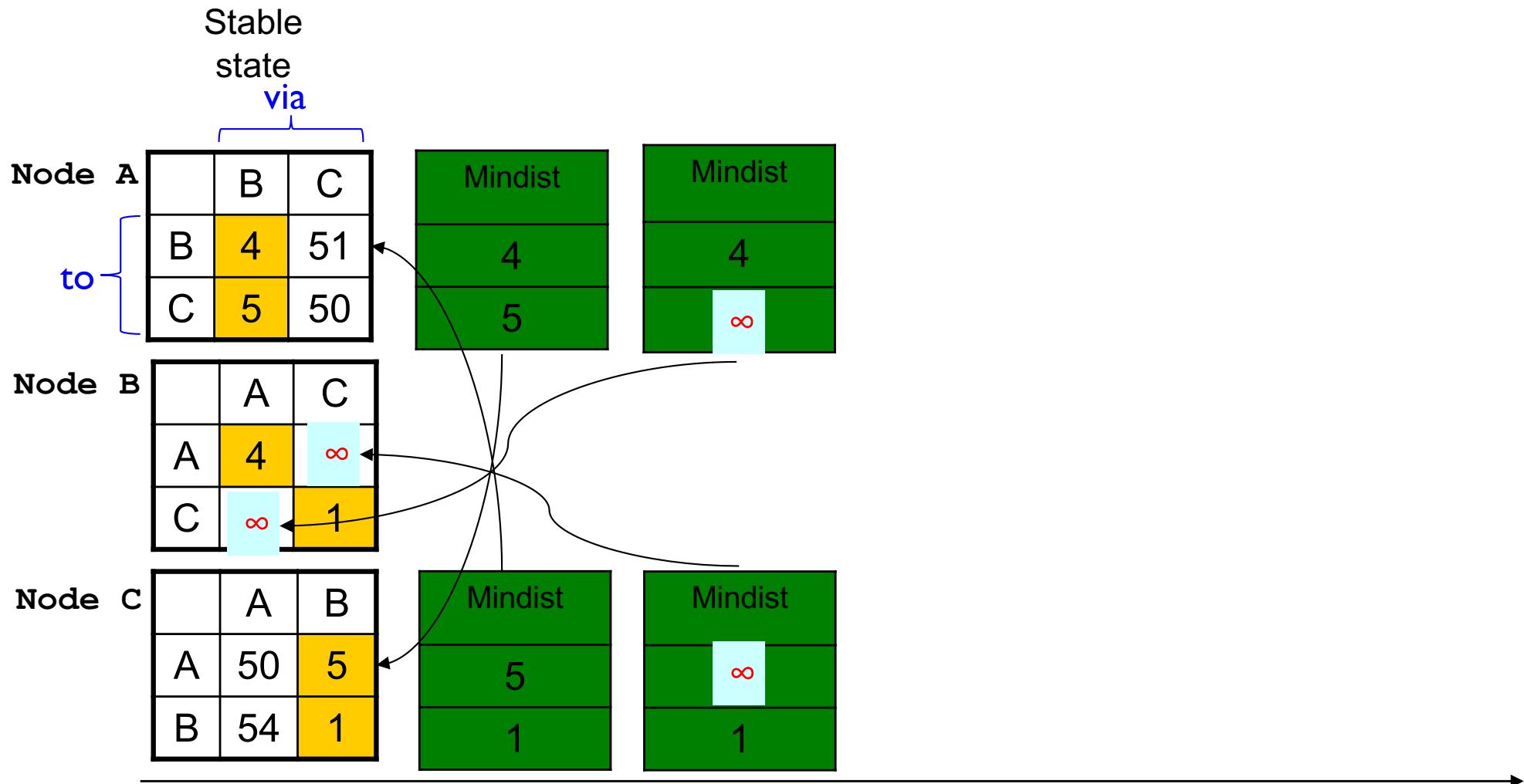
## The “Poisoned Reverse” Rule

- ❖ Heuristic to avoid count-to-infinity
- ❖ If B routes via C to get to A:
  - B tells C its (B's) distance to A is infinite  
(so C won't route to A via B)

# DV: Poisoned Reverse

If B routes through C to get to A:

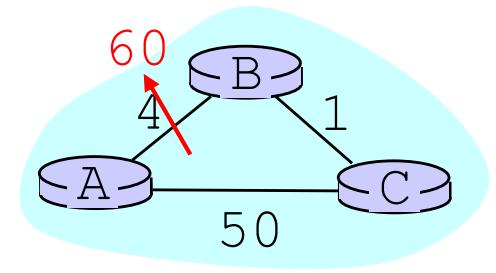
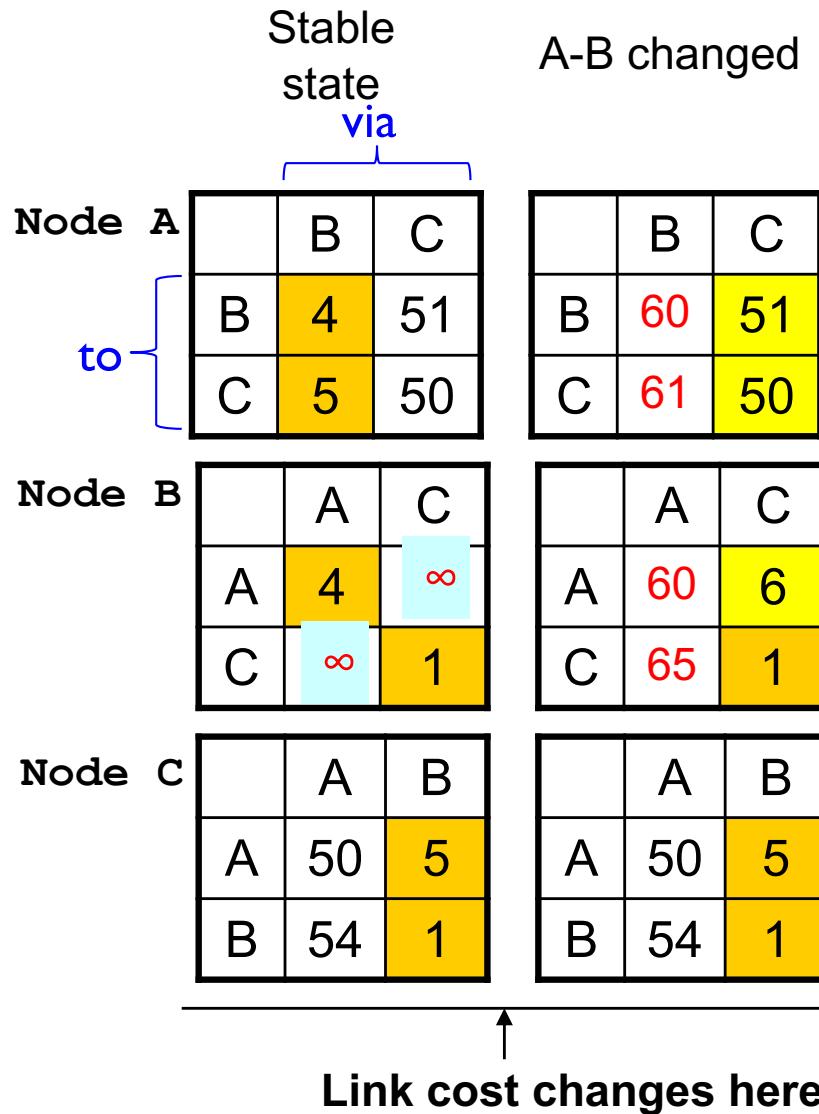
B tells C its (B's) distance to A is infinite



# DV: Poisoned Reverse

If B routes through C to get to A:

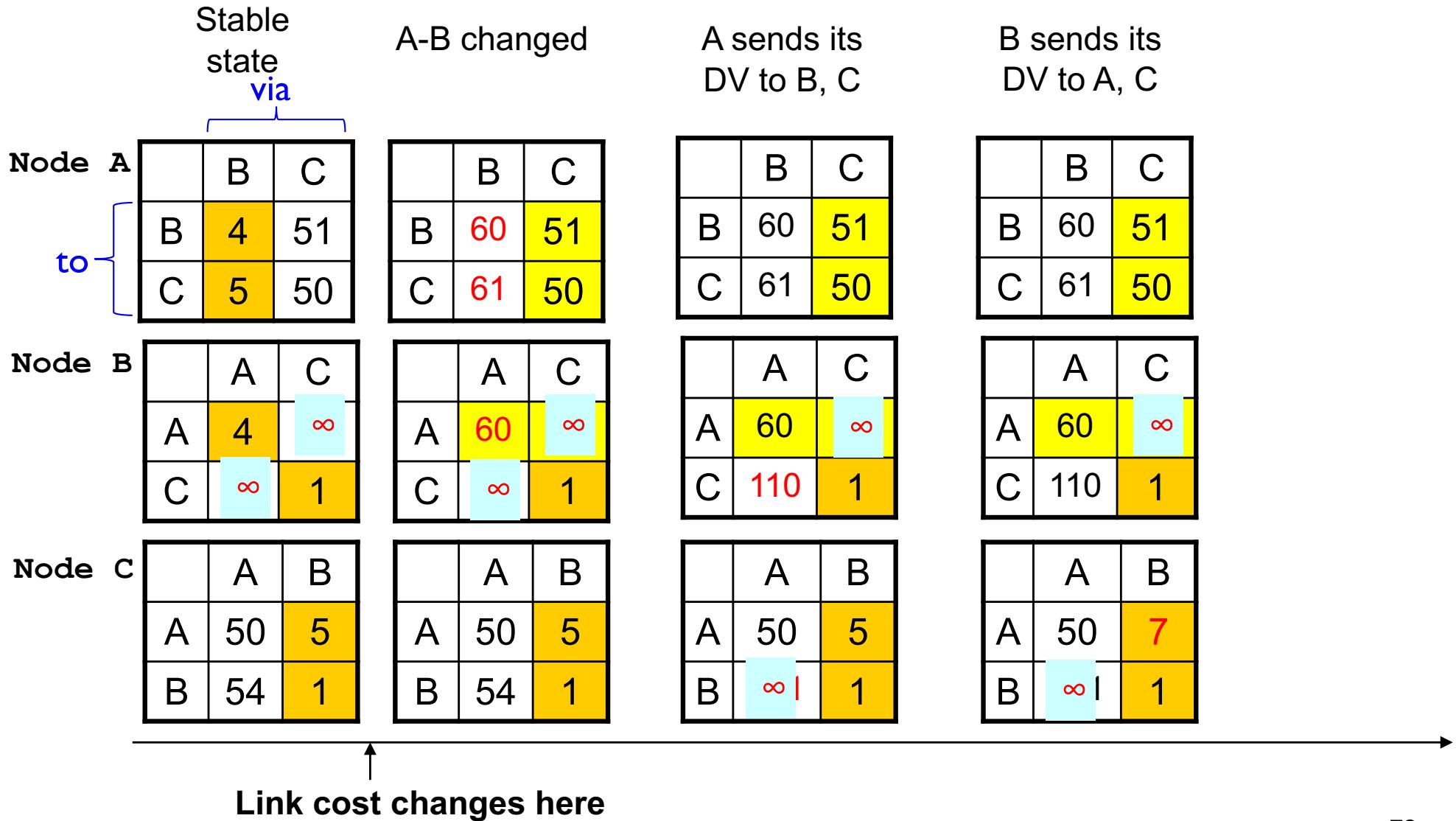
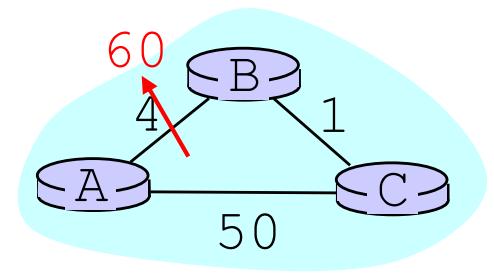
B tells C its (B's) distance to A is infinite



# DV: Poisoned Reverse

If B routes through C to get to A:

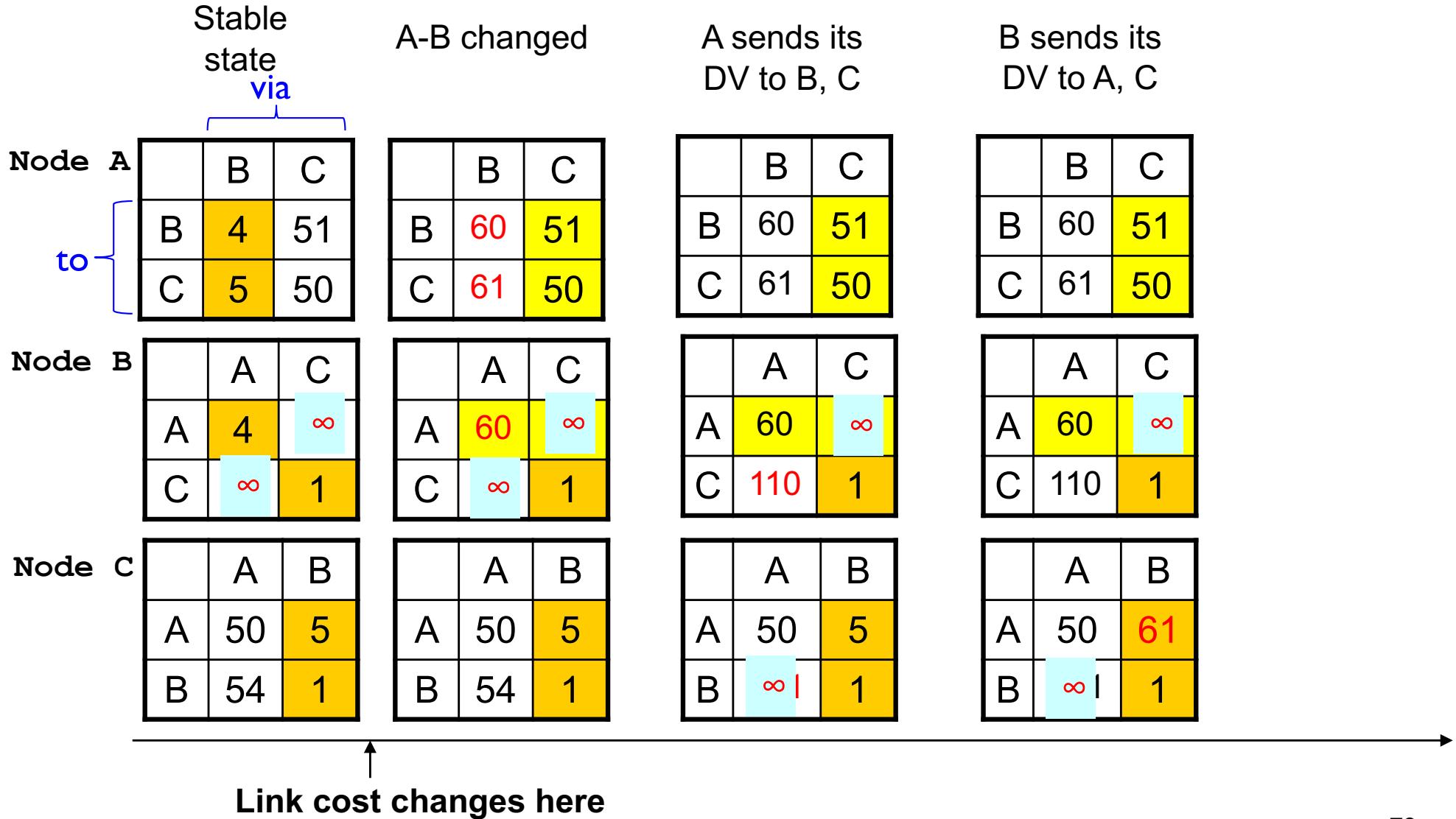
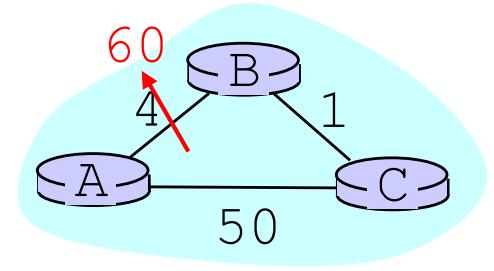
B tells C its (B's) distance to A is infinite



# DV: Poisoned Reverse

If B routes through C to get to A:

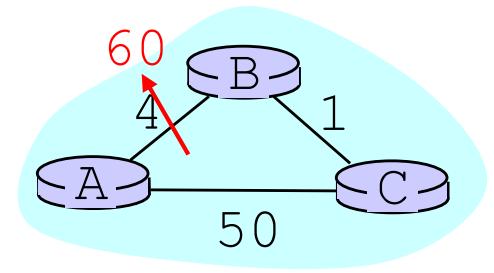
B tells C its (B's) distance to A is infinite



# DV: Poisoned Reverse

If B routes through C to get to A:

B tells C its (B's) distance to A is infinite

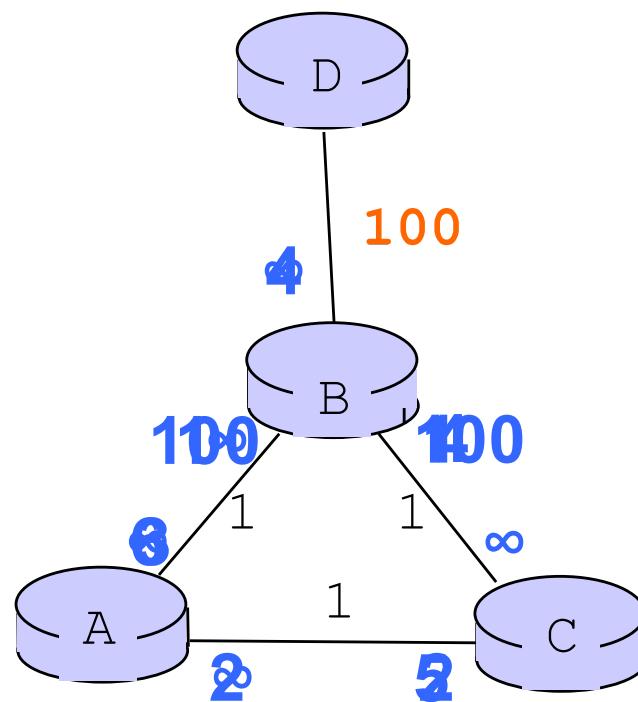


	Stable state via	A-B changed	A sends its DV to B, C	B sends its DV to A, C	C sends its DV to A, B																																													
<b>Node A</b>	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>4</td><td>51</td></tr> <tr><td>C</td><td>5</td><td>50</td></tr> </table>		B	C	B	4	51	C	5	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50	<table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table>		B	C	B	60	51	C	61	50
	B	C																																																
B	4	51																																																
C	5	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
	B	C																																																
B	60	51																																																
C	61	50																																																
<b>Node B</b>	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>4</td><td><math>\infty</math></td></tr> <tr><td>C</td><td><math>\infty</math></td><td>1</td></tr> </table>		A	C	A	4	$\infty$	C	$\infty$	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td><math>\infty</math></td></tr> <tr><td>C</td><td><math>\infty</math></td><td>1</td></tr> </table>		A	C	A	60	$\infty$	C	$\infty$	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td><math>\infty</math></td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>		A	C	A	60	$\infty$	C	110	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td><math>\infty</math></td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>		A	C	A	60	$\infty$	C	110	1	<table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>		A	C	A	60	51	C	110	1
	A	C																																																
A	4	$\infty$																																																
C	$\infty$	1																																																
	A	C																																																
A	60	$\infty$																																																
C	$\infty$	1																																																
	A	C																																																
A	60	$\infty$																																																
C	110	1																																																
	A	C																																																
A	60	$\infty$																																																
C	110	1																																																
	A	C																																																
A	60	51																																																
C	110	1																																																
<b>Node C</b>	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table>		A	B	A	50	5	B	54	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table>		A	B	A	50	5	B	54	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td><math>\infty</math></td><td>1</td></tr> </table>		A	B	A	50	5	B	$\infty$	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>61</td></tr> <tr><td>B</td><td><math>\infty</math></td><td>1</td></tr> </table>		A	B	A	50	61	B	$\infty$	1	<table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td><math>\infty</math></td></tr> <tr><td>B</td><td><math>\infty</math></td><td>1</td></tr> </table>		A	B	A	50	$\infty$	B	$\infty$	1
	A	B																																																
A	50	5																																																
B	54	1																																																
	A	B																																																
A	50	5																																																
B	54	1																																																
	A	B																																																
A	50	5																																																
B	$\infty$	1																																																
	A	B																																																
A	50	61																																																
B	$\infty$	1																																																
	A	B																																																
A	50	$\infty$																																																
B	$\infty$	1																																																

↑  
Link cost changes here

Converges after C receives another update from B

# Will Poison-Reverse Completely Solve the Count-to-Infinity Problem?



Numbers in blue denote the best cost to destination D advertised along the link

# Quiz: Link-state routing

- ❖ In link state routing, each node sends information of its direct links (i.e., link state) to \_\_\_\_\_?
  - A. Immediate neighbours
  - B. All nodes in the network
  - C. Any one neighbor
  - D. No one

**Answer: B**

# Quiz: Distance-vector routing

- ❖ In distance vector routing, each node shares its distance table with \_\_\_\_\_?
- A. All Immediate neighbours
  - B. All nodes in the network
  - C. Any one neighbor
  - D. No one

Answer: A

# Quiz: Distance-vector routing

---

- ❖ Which of the following is true of distance vector routing?
  - A. Convergence delay depends on the topology (nodes and links) and link weights
  - B. Convergence delay depends on the number of nodes and links
  - C. Each node knows the entire topology
  - D. A and C
  - E. B and C

**Answer: A**

# 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

# Real Protocols

*Link State*

Open Shortest Path  
First (OSPF)

Intermediate system to  
intermediate system (IS-  
IS)

*Distance Vector*

Routing Information  
Protocol (RIP)

Interior Gateway  
Routing Protocol  
(IGRP-Cisco)

Border Gateway  
Protocol (BGP)

# Network layer, control plane: outline

5.1 introduction

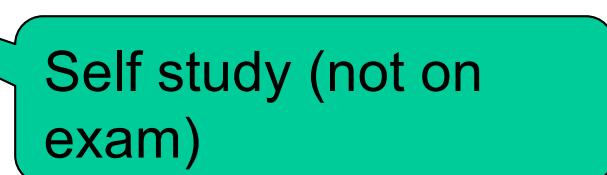
5.2 routing protocols

- ❖ link state

- ❖ distance vector

- ❖ hierarchical routing

5.6 ICMP: The Internet Control Message Protocol

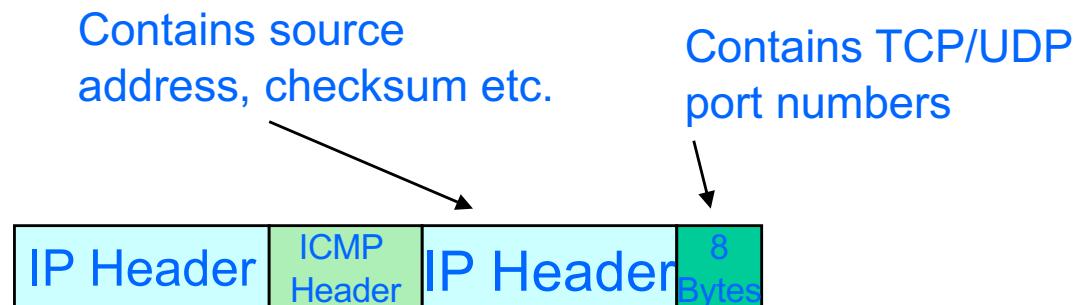


Self study (not on exam)

# ICMP: Internet Control Message Protocol

---

- ❖ Used by hosts & routers to communicate network level information
  - Error reporting: unreachable host, network, port
  - Echo request/reply (used by ping)
- ❖ Works above IP layer
  - ICMP messages carried in IP datagrams
- ❖ ICMP message: type, code plus IP header and first 8 bytes of IP datagram payload causing error



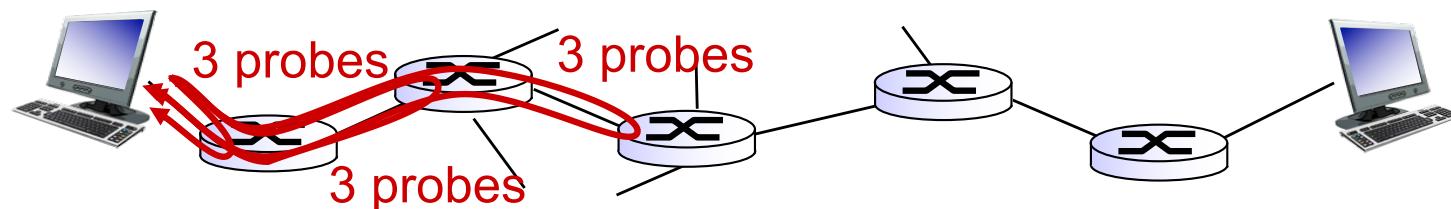
# ICMP: Internet Control Message Protocol

---

Type	Code	Description
0	0	echo reply(ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	3	dest port unreachable
3	4	frag needed; DF set
8	0	echo request(ping)
11	0	TTL expired
11	1	frag reassembly time exceeded
12	0	bad IP header

# Traceroute and ICMP

- Source sends series of UDP segments to dest
    - first set has TTL = 1
    - second set has TTL=2, etc.
    - unlikely port number
  - When  $n$ th set of datagrams arrives to  $n$ th router:
    - router discards datagrams
    - and sends source ICMP messages (type 11, code 0)
    - ICMP messages includes IP address of router
  - when ICMP messages arrives, source records RTTs
- stopping criteria:*
- UDP segment eventually arrives at destination host
  - destination returns ICMP “port unreachable” message (type 3, code 3)
  - source stops



# Summary

- ❖ Network Layer: Data Plane
  - Overview
  - IP
- ❖ Network Layer: Control Plane
  - Routing Protocols
    - Link-state
    - Distance Vector
  - ICMP

# COMP 3331/9331: Computer Networks and Applications

Week 9  
Data link Layer

Reading Guide: Chapter 6, Sections 6.1 – 6.4, 6.7

# Link layer and LANs

*our goals:*

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

# Link layer, LANs: outline

6.1 introduction, services

6.2 error detection,  
correction

6.3 multiple access  
protocols

6.4 Switched LANs

- addressing, ARP
- Ethernet
- Switches
- VLANS (**EXCLUDED**)

6.5 link virtualization:

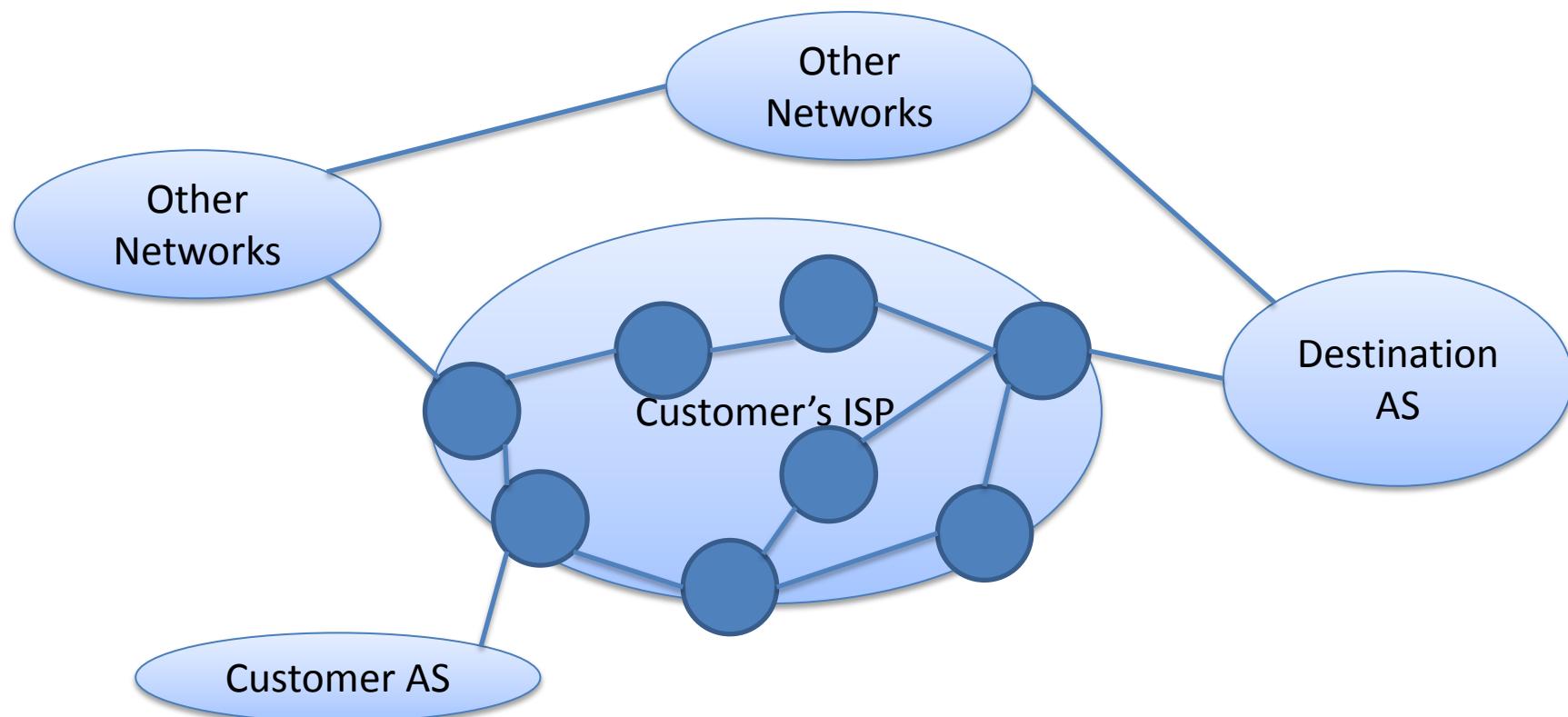
MPLS (**EXCLUDED**)

6.6 data center  
networking  
(**EXCLUDED**)

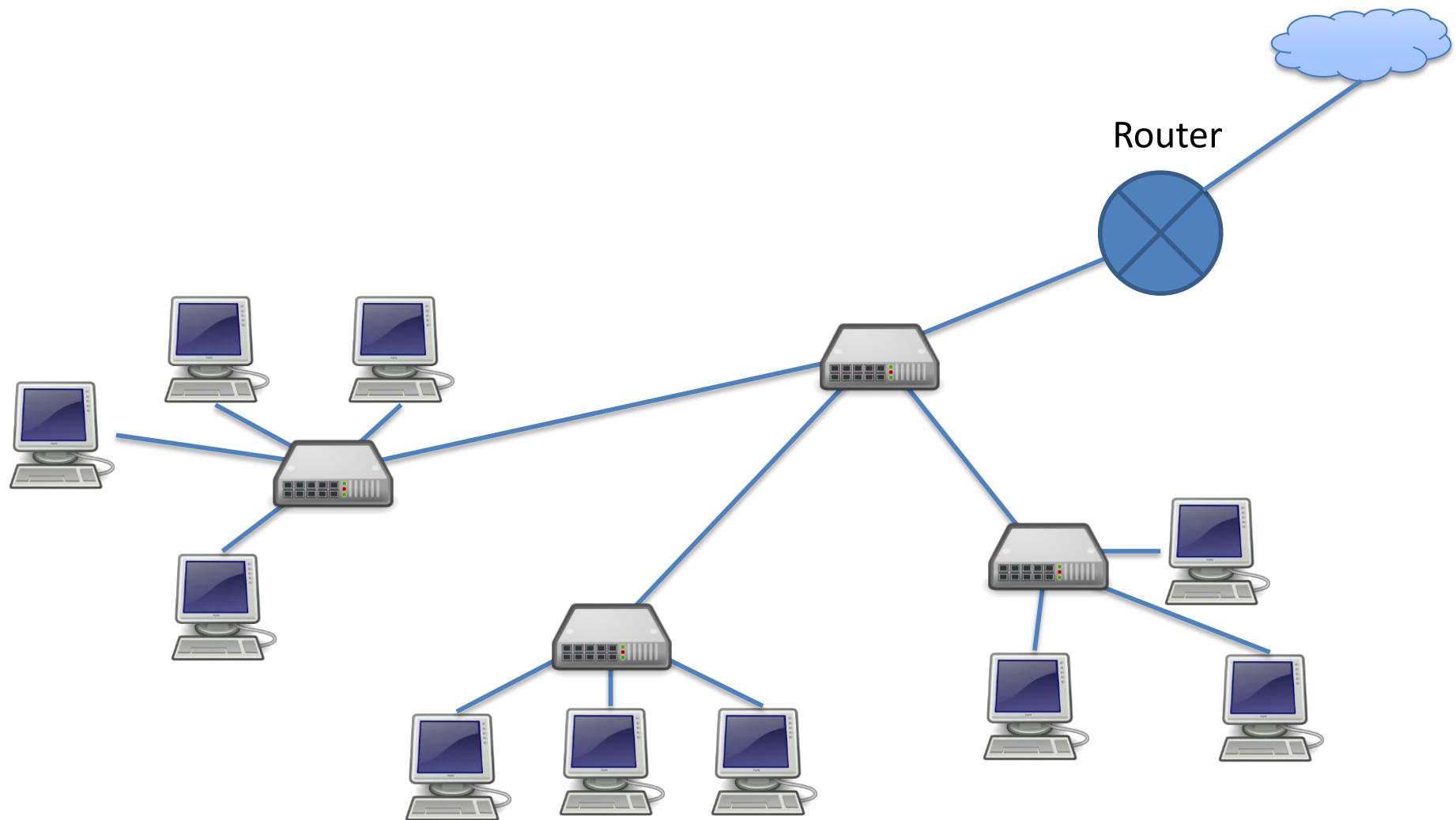
6.7 a day in the life of a  
web request

# From Macro- to Micro-

- Previously, we looked at Internet scale...



# Link layer focus: Within a Subnet

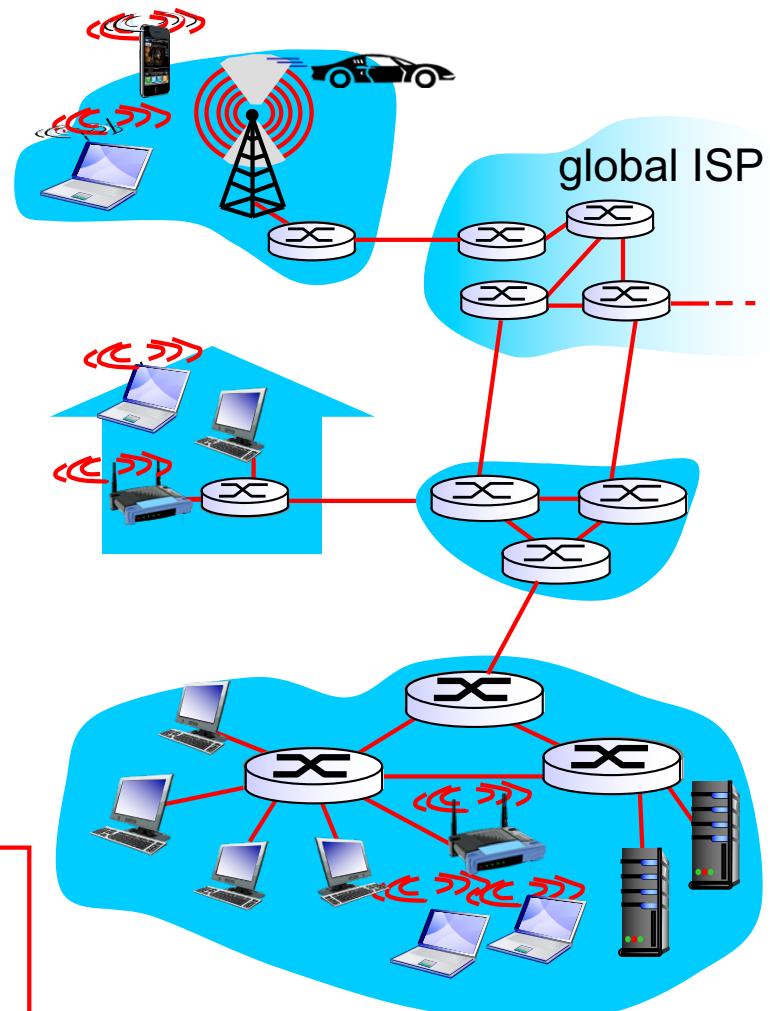


# Link layer: introduction

## *terminology:*

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

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



# Link layer: context

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

## *transportation analogy:*

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

# Link layer services

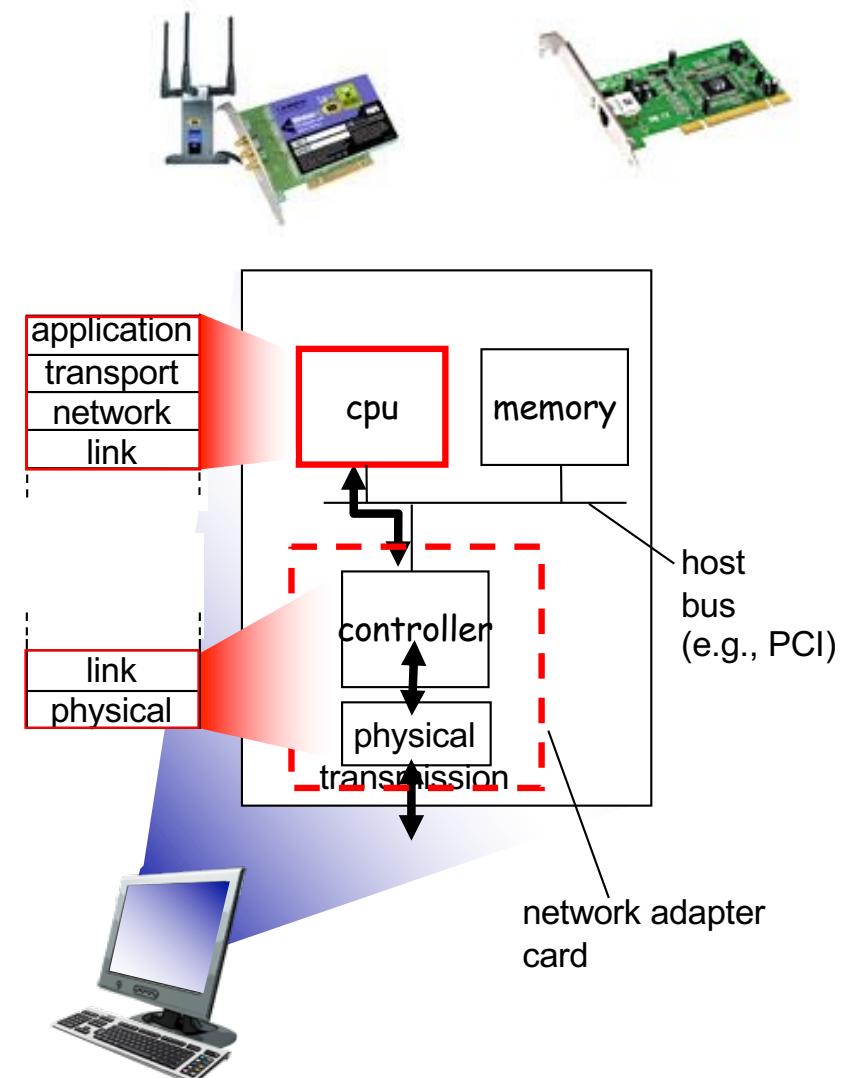
- ❖ *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - “MAC” addresses used in frame headers to identify source, dest
    - different from IP address!
- ❖ *reliable delivery between adjacent nodes*
  - we learned how to do this already (chapter 3)!
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - *Q:* why both link-level and end-end reliability?

# Link layer services (more)

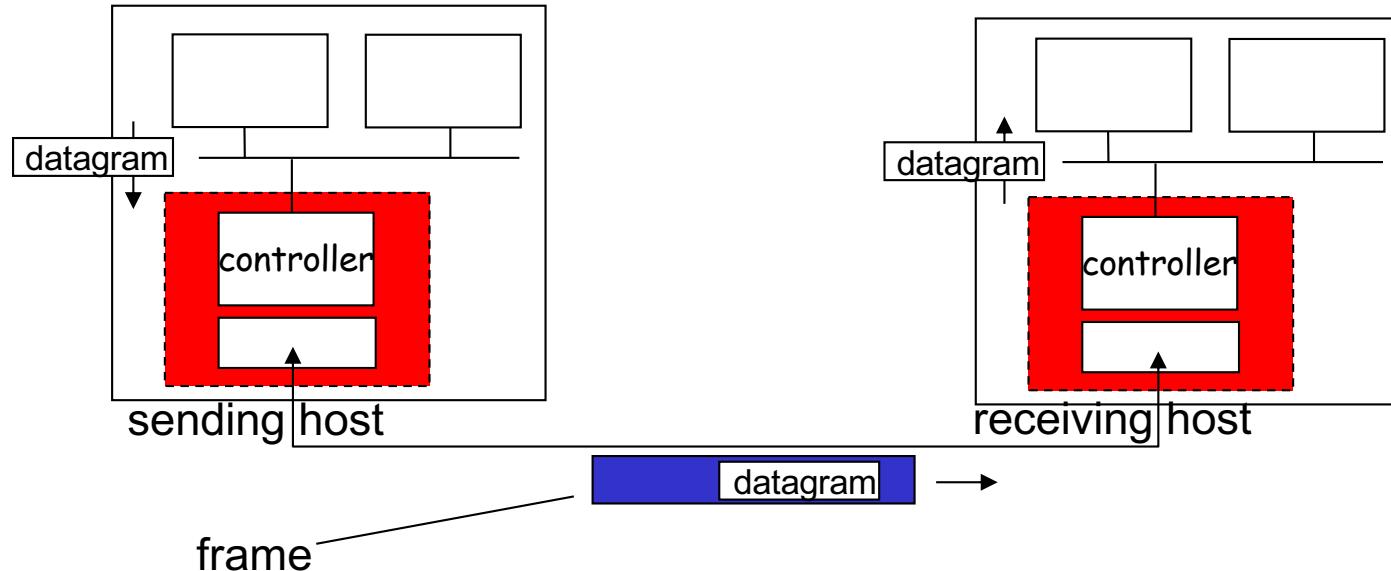
- ❖ *flow control:*
  - pacing between adjacent sending and receiving nodes
- ❖ *error detection:*
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- ❖ *error correction:*
  - receiver identifies *and corrects*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

# Where is the link layer implemented?

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



# Adaptors communicating



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

# What is framing?

- Physical layer talks in terms of bits.
- How do we identify frames within the sequence of bits?
- Need to do Framing
  - Delimit the start and end of the frame
- Ethernet Framing
  - Timing/Physical layer

# Framing in Ethernet

- Start of frame is recognized by
  - Preamble : Seven bytes with pattern 10101010
  - Start of Frame Delimiter (SFD) : 10101011

Preamble 7 Bytes	SFD 1 Byte	Dest MAC 6 Bytes	Source MAC 6 Bytes	Type/Len gth 2 Bytes	Payload 46-1500 Bytes	FCS/CRC 4 Bytes	Inter Frame Gap
---------------------	---------------	------------------------	--------------------------	----------------------------	-----------------------------	-----------------------	-----------------------

- Inter Frame Gap is 12 Bytes (96 bits) of idle state
  - 0.96 microsec for 100 Mbit/s Ethernet
  - 0.096 microsec for Gigabit/s Ethernet

# COMP 3331/9331: Computer Networks and Applications

Week 9  
Data link Layer

Reading Guide: Chapter 6, Sections 6.2 – 6.4, 6.7



**UNSW**  
SYDNEY

Australia's  
Global  
University

**“Having the lecture  
recordings up – absolute  
lifesaver when it comes  
to revision”**

**Shape  
OUR  
Future**

Tell us about your experience and  
shape the future of education at UNSW.



**Click the link in Moodle now**



*Complete your myExperience and shape  
the future of education at UNSW.*

**Click the  Experience link in Moodle**

**or login to myExperience.unsw.edu.au**

(use z1234567@**ad.unsw.edu.au** to login)

*The survey is confidential, your identity will never be released*

*Survey results are not released to teaching staff until after your results are published*

# Link layer, LANs: outline

6.1 introduction, services

6.2 error detection,  
correction

6.3 multiple access  
protocols

6.4 Switched LANs

- addressing, ARP
- Ethernet
- switches

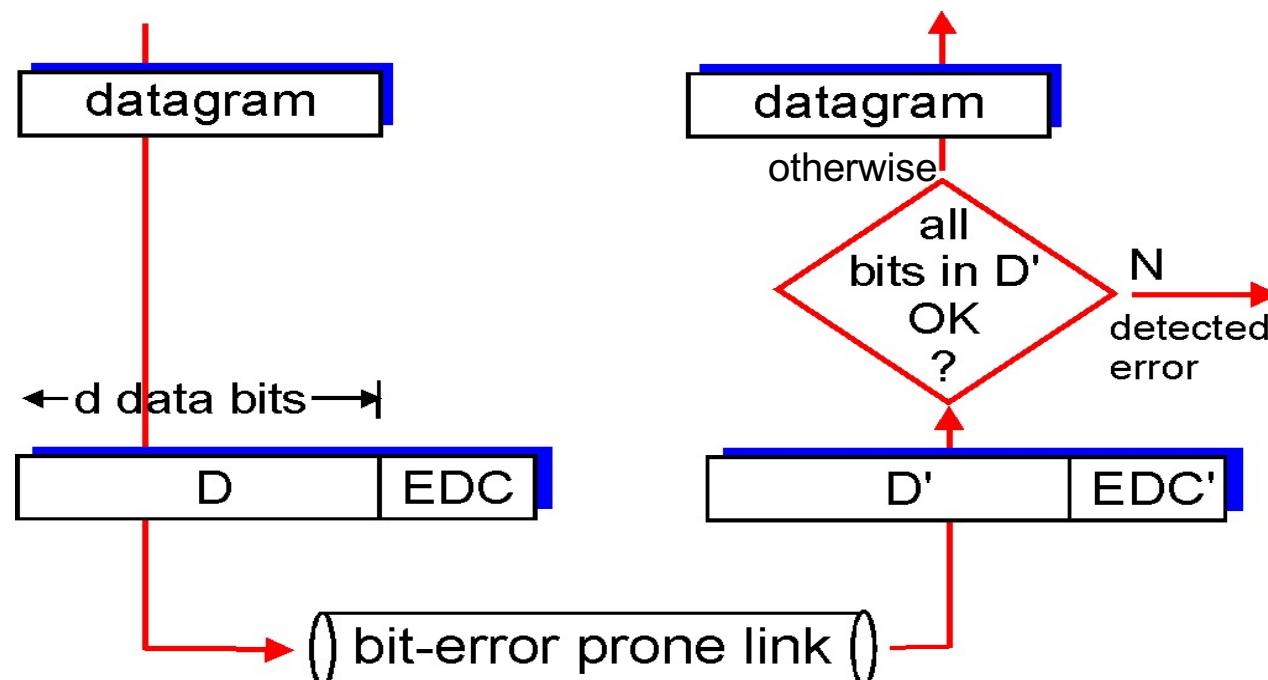
6.7 a day in the life of a  
web request

# Error detection

EDC= Error Detection and Correction bits (redundancy)

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

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



# Error Detection

- Error coding
- Add check bits to the message bits to let some errors be detected and some be corrected
- How to structure the code to detect many errors with few check bits and modest computation?
- A simple code
  - Send two copies of the same message : **101101**
  - Error if the copies are different : **101100**
  - How many errors can it correct? 0
  - How many errors can it detect? At most 3
  - How many errors will make it fail? Specific 2-bit errors
  - What is the overhead? 100% (wrt original message)

# Simple Parity - Sender

*Example uses even parity*

- Suppose you want to send the message:
  - 001011011011000110010
- For every  $d$  bits (e.g.,  $d = 7$ ), add a parity bit:
  - 1 if the number of one's is odd
  - 0 if the number of one's is even

Message chunk	Parity bit
0010110	1
1101100	0
0110010	1

– 00101101101100001100101

# Simple Parity - Receiver

- For each block of size  $d$ :
  - Count the number of 1's and compare with following parity bit.
- If an odd number of bits get flipped, we'll detect it (can't do much to correct it).
- Cost: One extra bit for every  $d$ 
  - In this example, 21  $\rightarrow$  24 bits.

# Two-Dimensional Parity

*Example uses even parity*

- Suppose you want to send the same message:
  - 001011011011000110010
- Add an extra parity byte, compute parity on “columns” too.
- Can detect 1, 2, 3-bit (and some 4-bit) errors

	Message chunk	Parity bit
	0010110	1
	1101100	0
	0110010	1
Parity byte:	1001000	0

# Forward Error Correction

*Example uses even parity*

- With two-dimensional parity, we can even *correct* single-bit errors.

The diagram illustrates a 4x9 grid of binary bits. A blue arrow labeled "Parity bits" points downwards from the grid to a row of four binary digits at the bottom, labeled "Parity byte". The grid has four rows and nine columns. The first four columns represent data, and the last five columns represent parity bits. The fifth column from the left contains a circled "0", indicating it is the bit that has been flipped. The "Parity byte" row contains the values 1, 1, 1, 1, 1, 1, 0, 0, 0, representing the calculated parity for each column.

0	0	1	0	1	1	0	1	
1	0	1	0	0	0	1	0	
1	0	0	1	0	1	1	0	
1	1	1	0	1	1	0	1	
Parity byte →								
1	1	1	1	1	1	0	0	

Exactly one bit has been flipped. Which is it?

## In practice

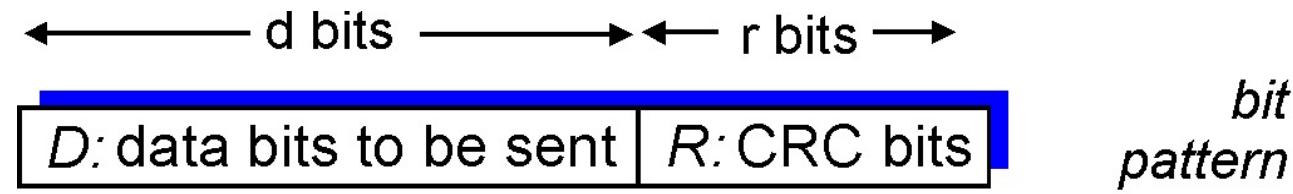
- Bit errors occur in bursts.
- We're willing to trade computational complexity for space efficiency.
  - Make the detection routine more complex, to detect error bursts, without tons of extra data
- Insight: We need hardware to interface with the network, do the computation there!

# Error Detection and Correction

- **Checksum**
  - Sum up data in N-bit words
  - Internet Checksum uses 16-bit words
- **How well checksum works?**
  - How many errors can it detect/correct?
- **What have we gained as compared to parity bit?**
  - Can now detect all burst errors up to 16

# Cyclic redundancy check

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



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

*mathematical formula*

# Cyclic redundancy check

- Sender operation
  - Extend D data bits with R zeros
  - Divide by generator G
  - Keep remainder, ignore quotient
  - Adjust R check bits by the remainder
- Receiver Procedure
  - Divide received frame by G and check for zero remainder

# A Note on Modulo-2 Arithmetic

- All calculations are modulo-2 arithmetic
- No carries or borrows in subtraction
- Addition and subtraction are identical and both are equivalent to XOR
  - $1011 \text{ XOR } 0101 = 1110$
  - $1011 - 0101 = 1110$
  - $1011 + 0101 = 1110$
- Multiplication by  $2^k$  is essentially a left shift by  $k$  bits
  - $1011 \times 2^2 = 101100$

# CRC example

want:

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

equivalently:

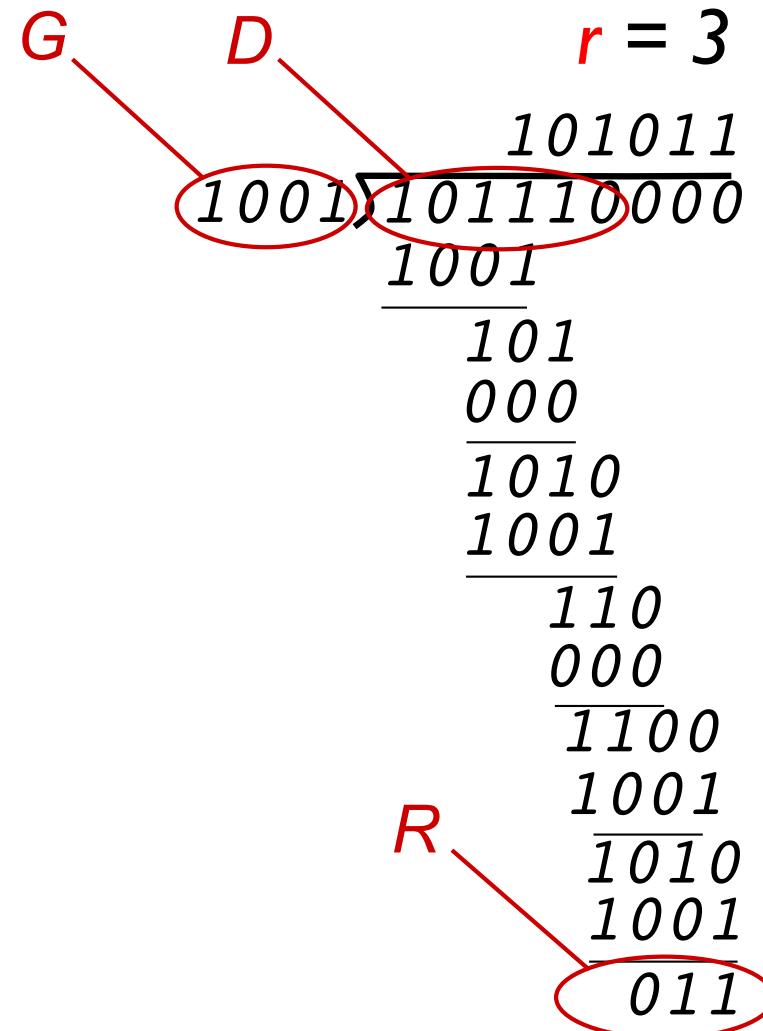
$$D \cdot 2^r = nG \text{ XOR } R$$

equivalently:

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

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

*At the sender*



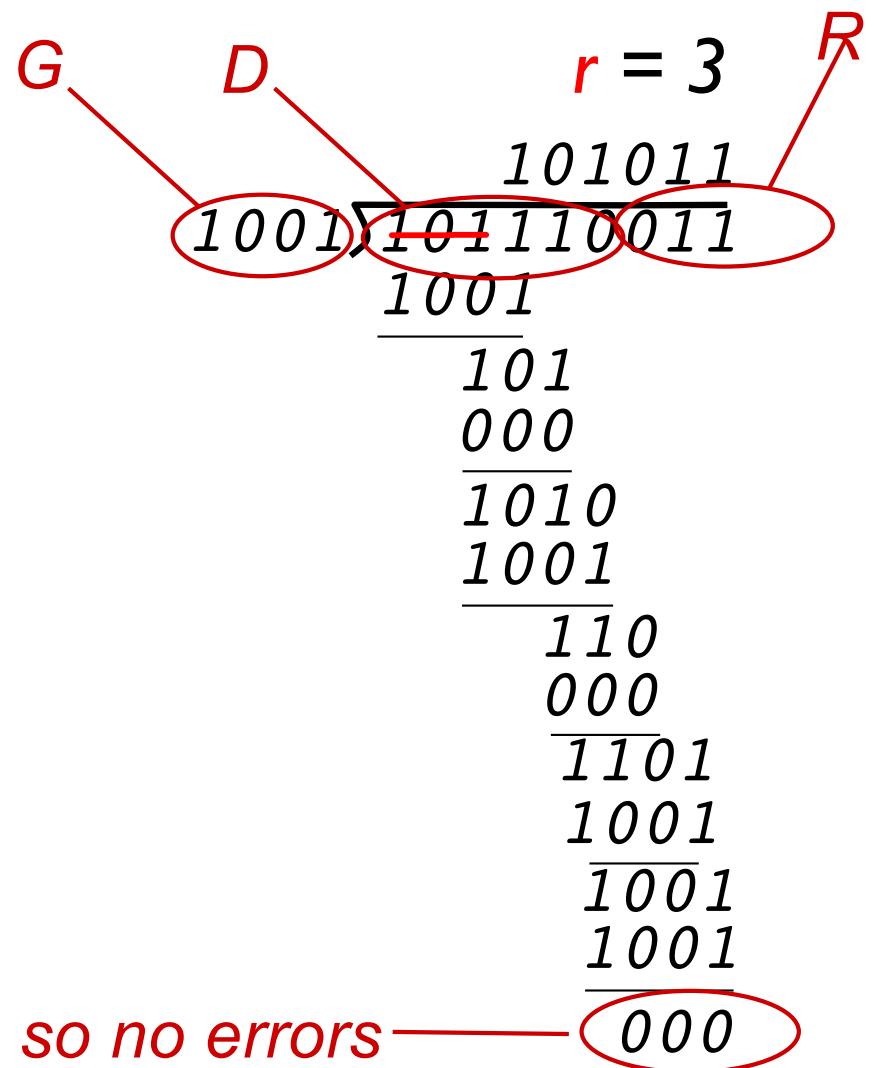
Sender sends  $\langle D, R \rangle$  into the channel

# CRC example

Receiver divides the received frame and divides by  $G$  and checks if the remainder is zero.

In this example, there are no errors, so the receiver receives  $\langle D, R \rangle$  (from previous slide)

*At the receiver*



*Remainder is zero, so no errors*

## Quiz: Error Detection/Correction



- ❖ Can these schemes respectively correct any bit errors: Internet checksums, two-dimensional parity, cyclic redundancy check (CRC)
  - a) Yes, No, No
  - b) No, Yes, Yes
  - c) No, Yes, No
  - d) No, No, Yes
  - e) No, No, No

*ANSWER: C*

# Link layer, LANs: outline

6.1 introduction, services

6.7 a day in the life of a  
web request

6.2 error detection,  
correction

6.3 multiple access  
protocols

6.4 Switched LANs

- addressing, ARP
- Ethernet
- switches

# Multiple access links, protocols

two types of “links”:

- ❖ **point-to-point**

- PPP for dial-up access
- point-to-point link between Ethernet switch, host

- ❖ ***broadcast (shared wire or medium)***

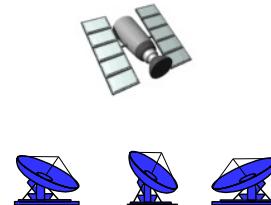
- old-fashioned Ethernet
- upstream HFC
- 802.11 wireless LAN



shared wire (e.g.,  
cabled Ethernet)



shared RF  
(e.g., 802.11 WiFi)



shared RF  
(satellite)



humans at a  
cocktail party  
(shared air, acoustical)

# Multiple access protocols

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

## *multiple access protocol*

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

# An ideal multiple access protocol

*given:* broadcast channel of rate  $R$  bps

*requirements:*

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

# MAC protocols: taxonomy

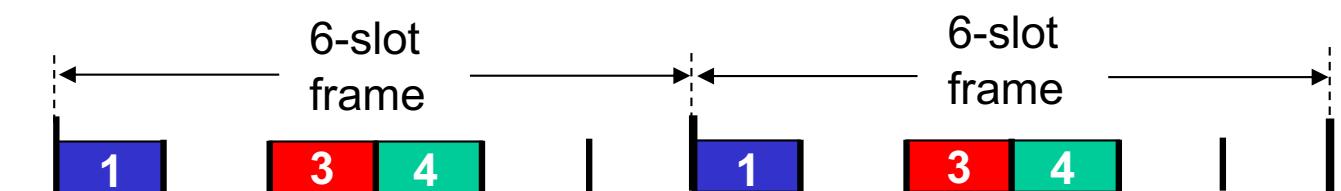
three broad classes:

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

# Channel partitioning MAC protocols: TDMA

## TDMA: time division multiple access

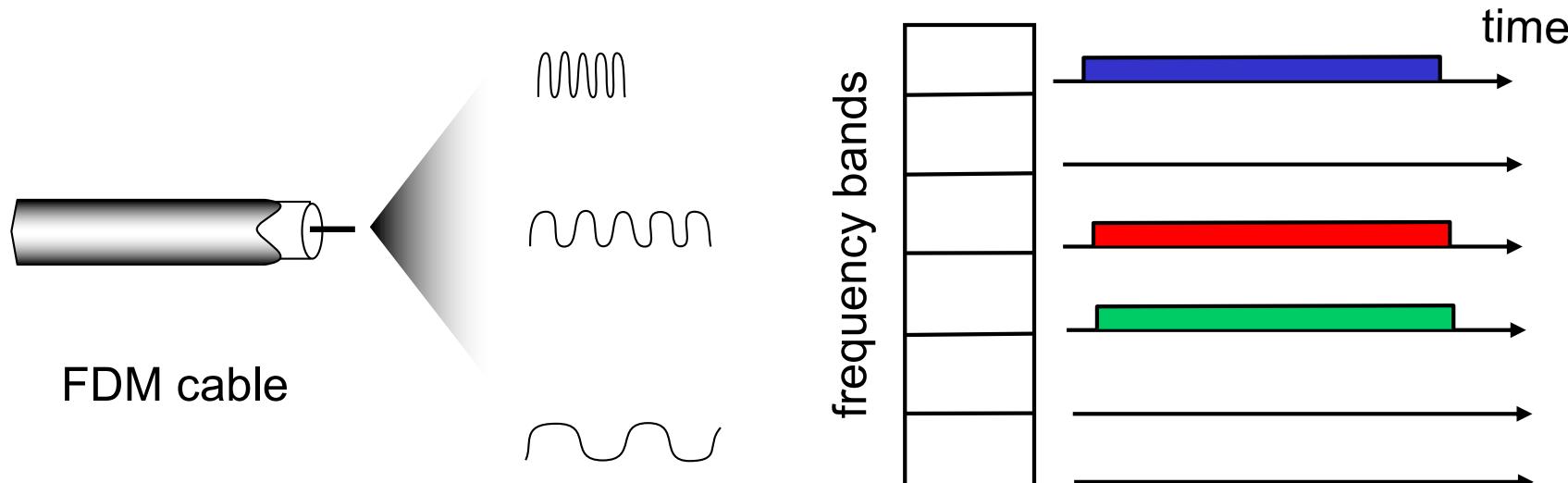
- ❖ access to channel in "rounds"
- ❖ each station gets fixed length slot (length = pkt trans time) in each round
- ❖ unused slots go idle
- ❖ example: 6-station LAN, 1,3,4 have pkt, slots 2,5,6 idle



# Channel partitioning MAC protocols: FDMA

## FDMA: frequency division multiple access

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



# Quiz: Does channel partitioning satisfy ideal properties ?

---



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

- A. 0  
B. 1  
C. 2  
D. 3  
E. 4

*ANSWER: C*

*2 and 4 from above are satisfied*

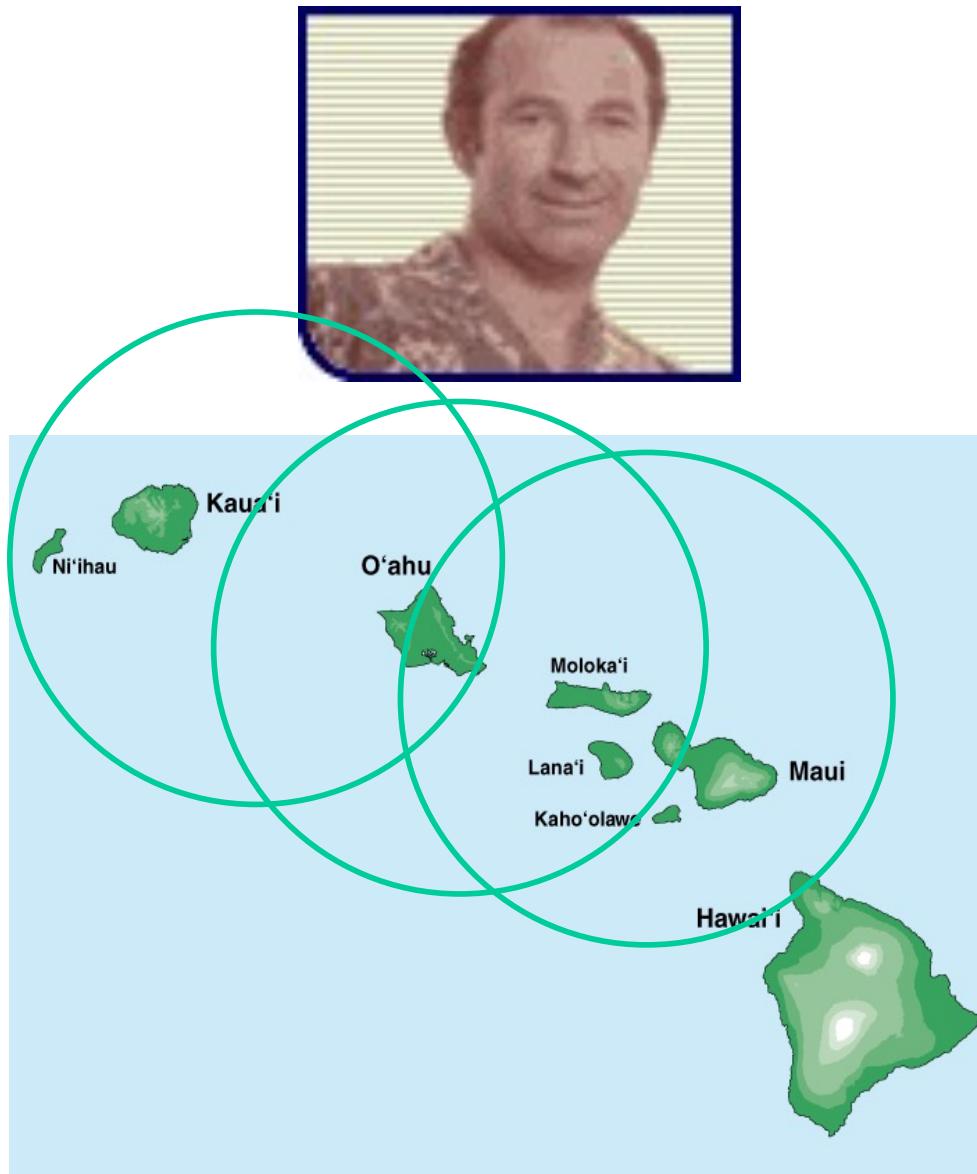
*Assuming  $M=N$  (no of nodes on the network)  
if  $M < N$ , then 2 is not satisfied)*

(Which ones?)

# Random access protocols

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

# Where it all Started: AlohaNet



- ❖ Norm Abramson left Stanford in 1970 (***so he could surf!***)
- ❖ Set up first data communication system for Hawaiian islands
- ❖ Central hub at U. Hawaii, Oahu

# Slotted ALOHA

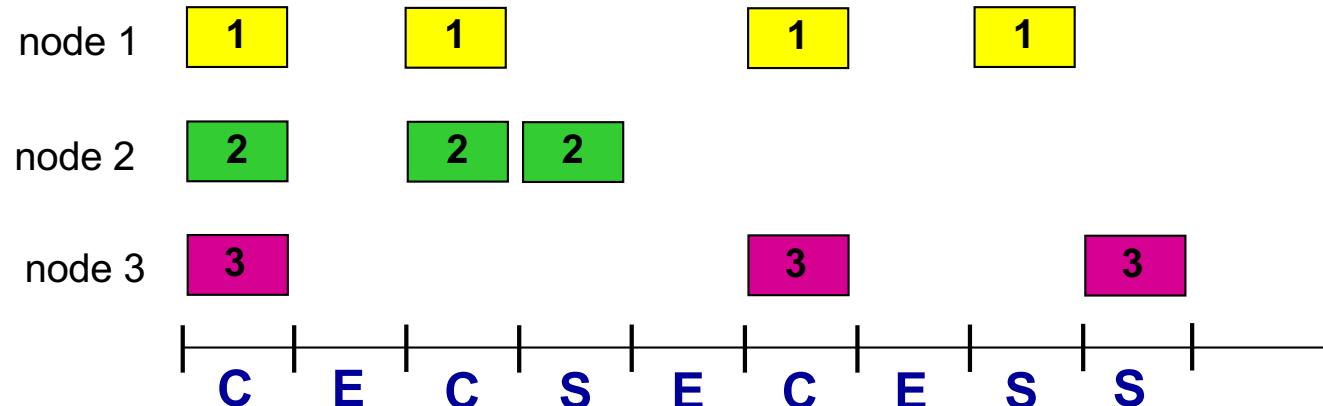
## *assumptions:*

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

## *operation:*

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

# Slotted ALOHA



## Pros:

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

## Cons:

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

# Slotted ALOHA: efficiency

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

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

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

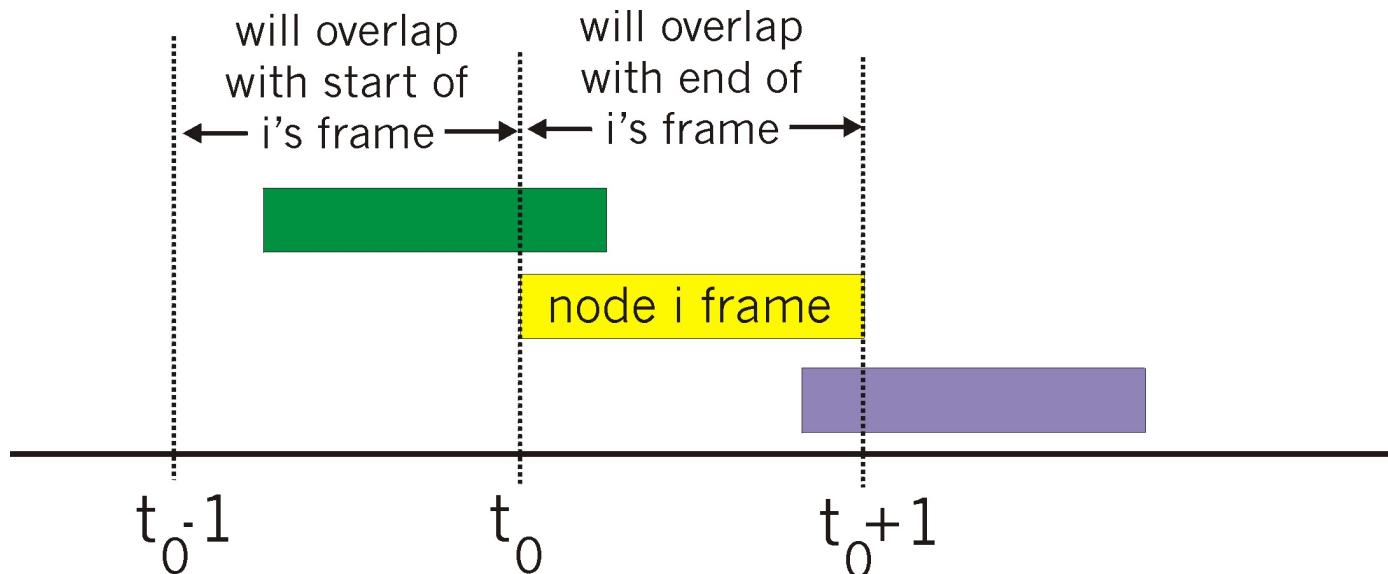
$$\text{max efficiency} = \underline{1/e = .37}$$

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

!

# Pure (unslotted) ALOHA

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



# Pure ALOHA efficiency

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

even worse than slotted Aloha!

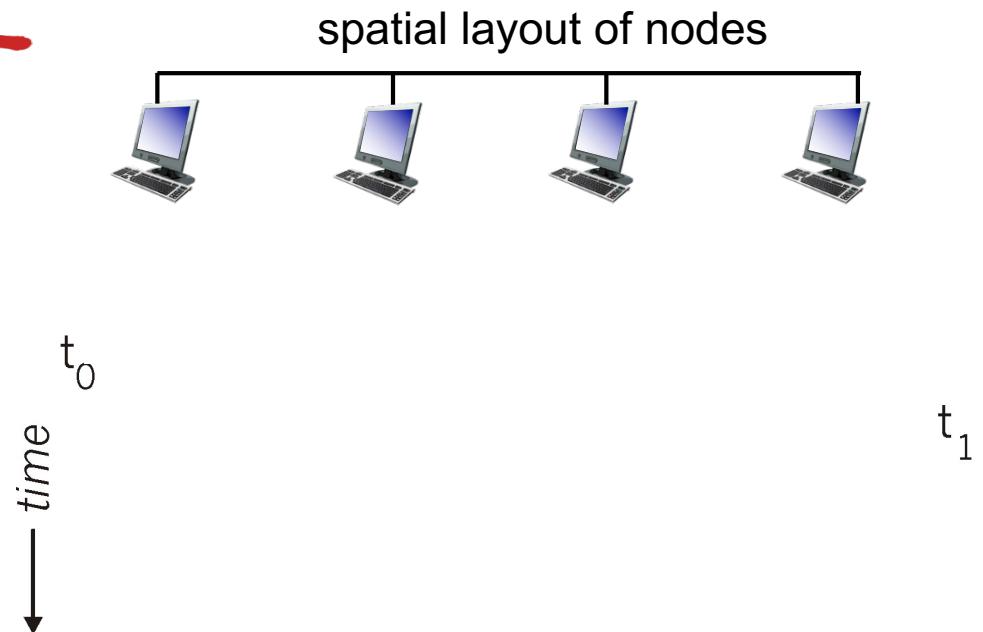
# CSMA (carrier sense multiple access)

**CSMA:** listen before transmit:

- if channel sensed idle: transmit entire frame
  - ❖ if channel sensed busy, defer transmission
  - ❖ human analogy: don't interrupt others!
  - ❖ Does this eliminate all collisions?
    - No, because of nonzero propagation delay

# CSMA collisions

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



*CSMA reduces but does not  
eliminate collisions*

*Biggest remaining problem?*

*Collisions can be detected earlier!*

•

# CSMA/CD (collision detection)

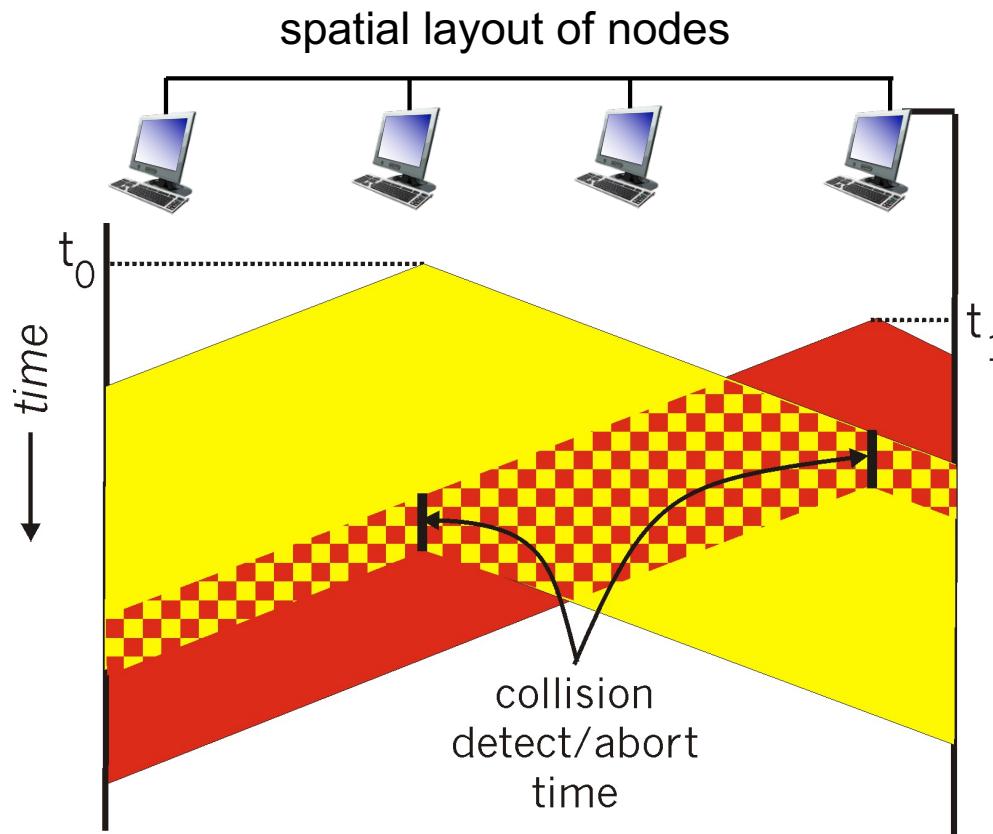
***CSMA/CD***: carrier sensing, deferral as in CSMA

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

# CSMA/CD (collision detection)

*Note: for this to work, need restrictions on minimum frame size and maximum distance.*

**Why?**



[http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_2/applets/csmacd/csmacd.html](http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/csmacd/csmacd.html)

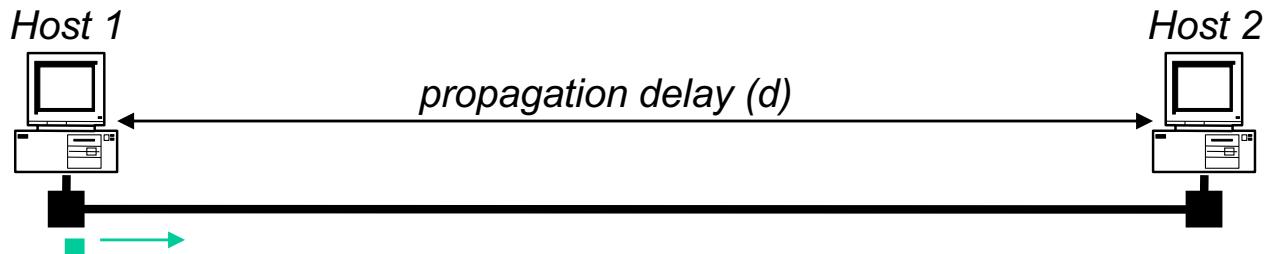
# Minimum Packet Size

---

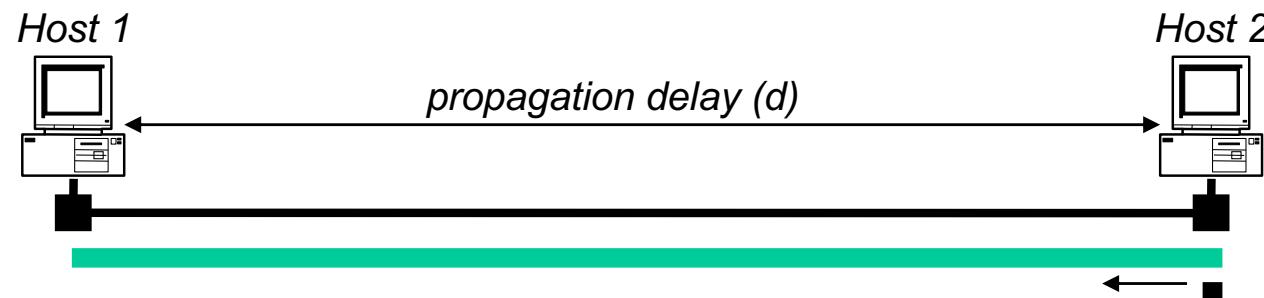
- ❖ Why enforce a minimum packet size?
- ❖ Give a host enough time to detect collisions
- ❖ In Ethernet, minimum packet size = 64 bytes (two 6-byte addresses, 2-byte type, 4-byte CRC, and 46 bytes of data)
- ❖ If host has less than 46 bytes to send, the adaptor pads (adds) bytes to make it 46 bytes
- ❖ What is the relationship between minimum packet size and the length of the LAN?

# Limits on CSMA/CD Network Length

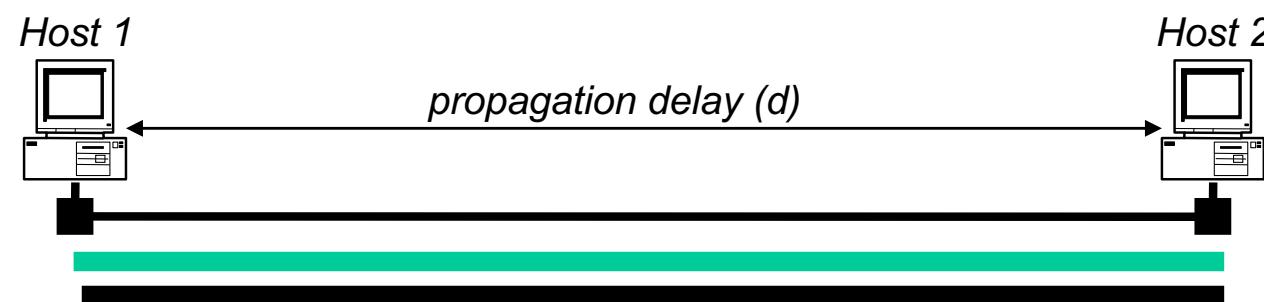
a) Time =  $t$ ; Host 1 starts to send frame



b) Time =  $t + d$ ; Host 2 starts to send a frame, just before it hears from host 1's frame



c) Time =  $t + 2*d$ ; Host 1 hears Host 2's frame → **detects collision**



$$\text{min\_frame\_size}/\text{bandwidth} = 2 * \text{LAN length}/\text{propagation speed}$$

$$\begin{aligned} \text{For } 10 \text{ Mbps LAN, LAN length} &= (\text{min\_frame\_size}) * (\text{propagation\_speed}) / (2 * \text{bandwidth}) = \\ &= (8 * 64B) * (2 * 10^8 \text{ mps}) / (2 * 10^7 \text{ bps}) = 5120 \text{ m approx} \end{aligned}$$

What about 100 mbps? 1 gbps? 10 gbps?

# Ethernet CSMA/CD algorithm

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

# Quiz: Does CSMA/CD satisfy ideal properties ?

---



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

[www.zetings.com/salil](http://www.zetings.com/salil)

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

**Answer: D**

**1, 3 and 4 are satisfied**

**2 is not satisfied as bandwidth is wasted due to collisions when multiple nodes are transmitting (neglect the overheads for channel sensing)**

(Which ones?)

# “Taking turns” MAC protocols

## channel partitioning MAC protocols:

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

## random access MAC protocols

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

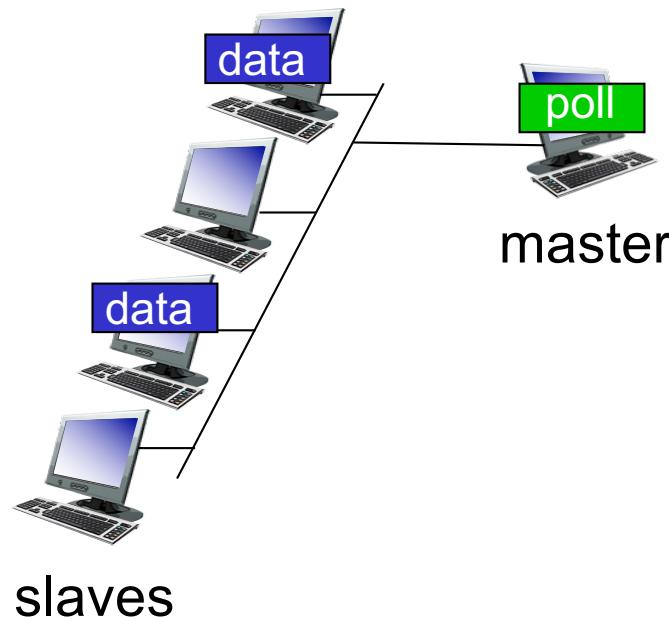
## “taking turns” protocols

look for best of both worlds!

# “Taking turns” MAC protocols

## polling:

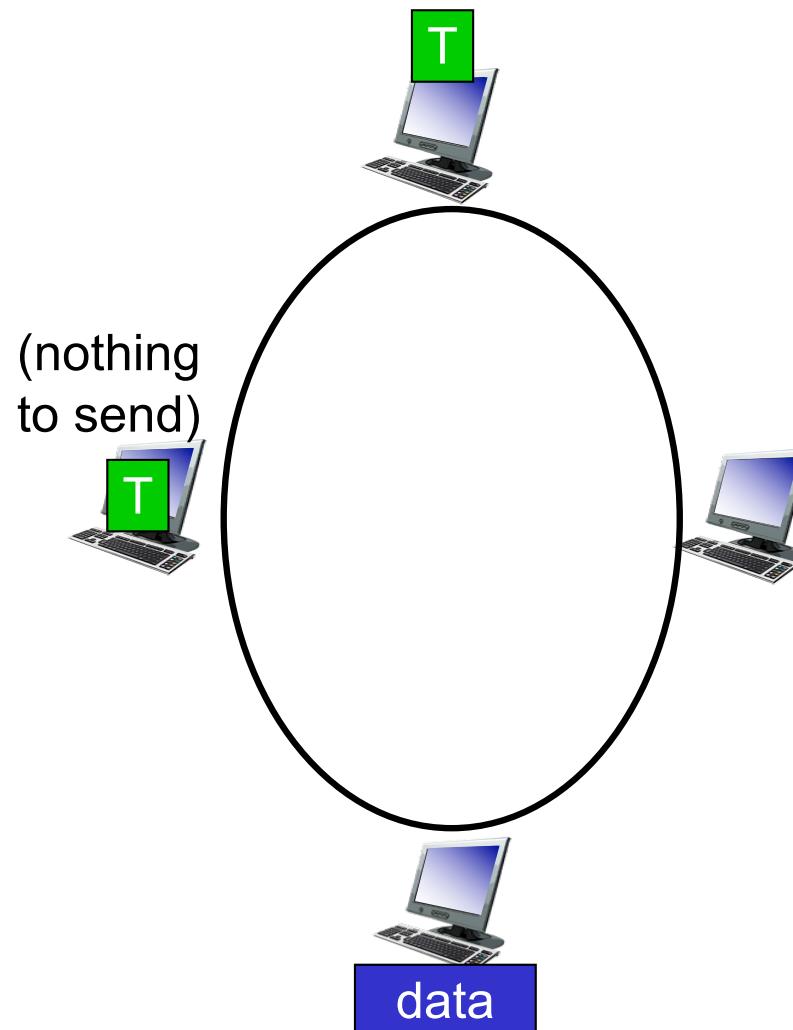
- ❖ master node “invites” slave nodes to transmit in turn
- ❖ typically used with “dumb” slave devices
- ❖ concerns:
  - polling overhead
  - latency
  - single point of failure (master)



# “Taking turns” MAC protocols

## *token passing:*

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



# Quiz: Does taking turns satisfy ideal properties ?

---



1. if only one node wants to transmit, it can send at rate R.
  2. when M nodes want to transmit, each can send at average rate  $R/M$  (fairness)
  3. fully decentralized:
    - no synchronization of clocks, slots
    - no special node to coordinate transmissions
  4. simple
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4
- (Which ones?)
- [www.zetings.com/salil](http://www.zetings.com/salil)
- Answer: D**  
**1, 2 and 4 are satisfied**  
**(neglect the overheads for polling and token passing)**

# Summary of MAC protocols

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

# Link layer, LANs: outline

6.1 introduction, services

6.7 a day in the life of a  
web request

6.2 error detection,  
correction

6.3 multiple access  
protocols

## 6.4 Switched LANs

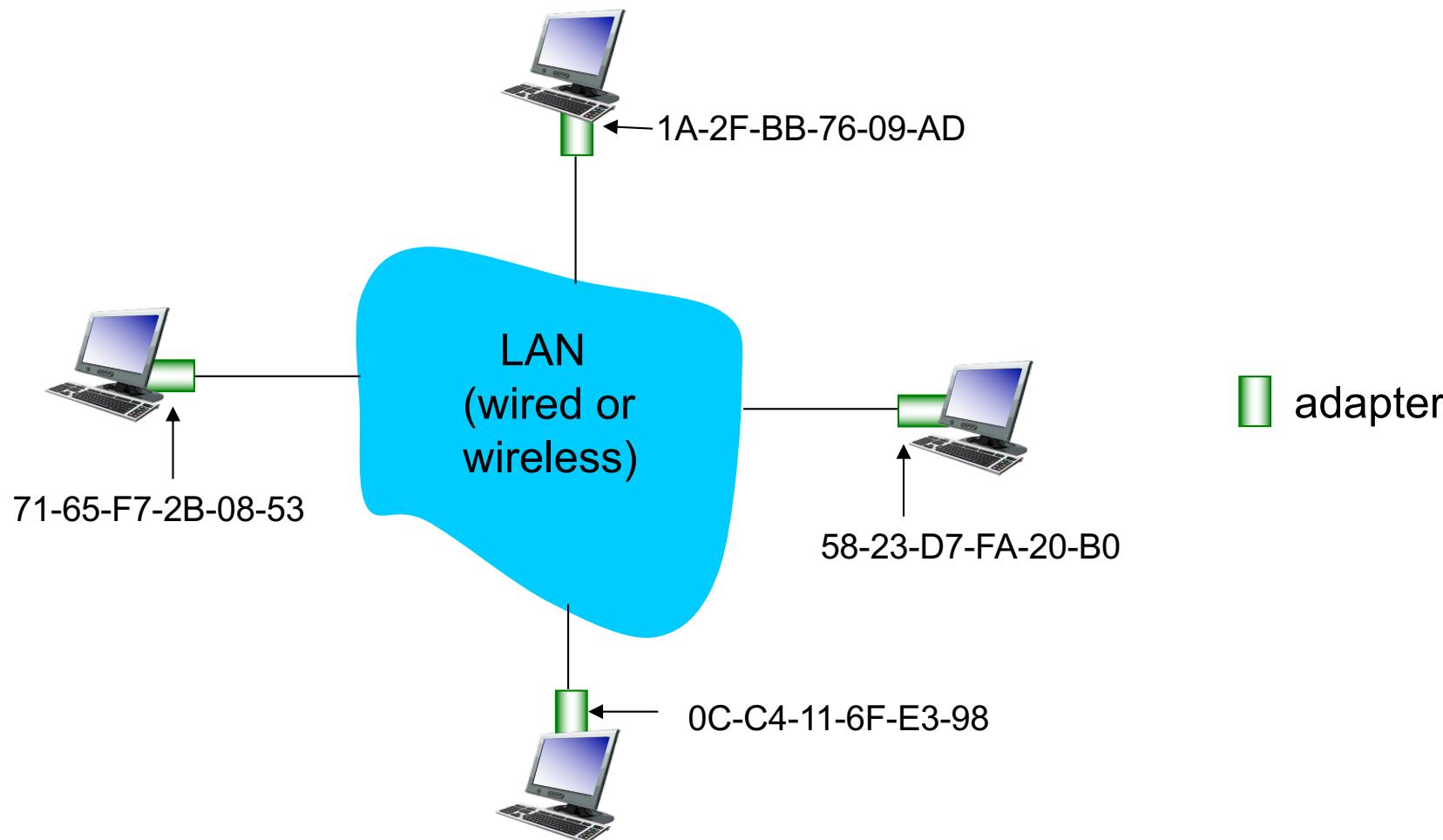
- addressing, ARP
- Ethernet
- switches

# MAC addresses and ARP

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

# LAN addresses and ARP

each adapter on LAN has unique *LAN* address



## LAN addresses (more)

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

# MAC Address vs. IP Address

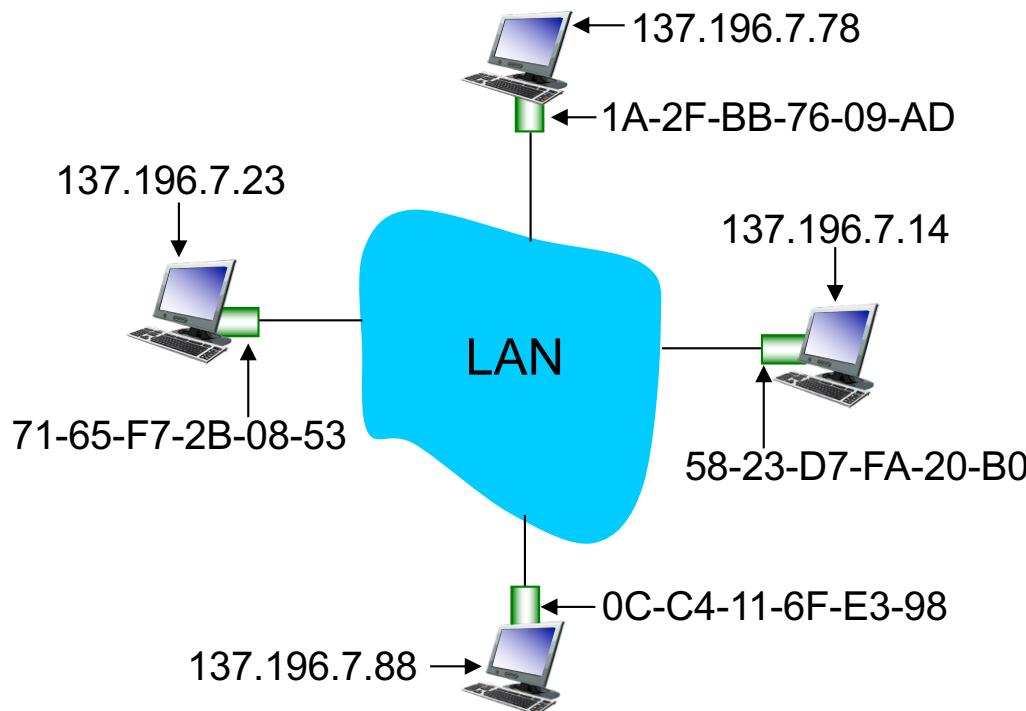
- ❖ MAC addresses (used in link-layer)
  - Hard-coded in read-only memory when adapter is built
  - Flat name space of 48 bits (e.g., 00-0E-9B-6E-49-76)
  - Portable, and can stay the same as the host moves
  - Used to get packet between interfaces on same network
- ❖ IP addresses
  - Configured, or learned dynamically
  - Hierarchical name space of 32 bits (e.g., 12.178.66.9)
  - Not portable, and depends on where the host is attached
  - Used to get a packet to destination IP subnet

# Taking Stock: Naming

Layer	Examples	Structure	Configuration	Resolution Service
App. Layer	www.cse.unsw.edu.au	organizational hierarchy	~ manual	DNS
Network Layer	129.94.242.51	topological hierarchy	DHCP	ARP
Link layer	45-CC-4E-12-F0-97	vendor (flat)	hard-coded	

# ARP: address resolution protocol

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



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

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

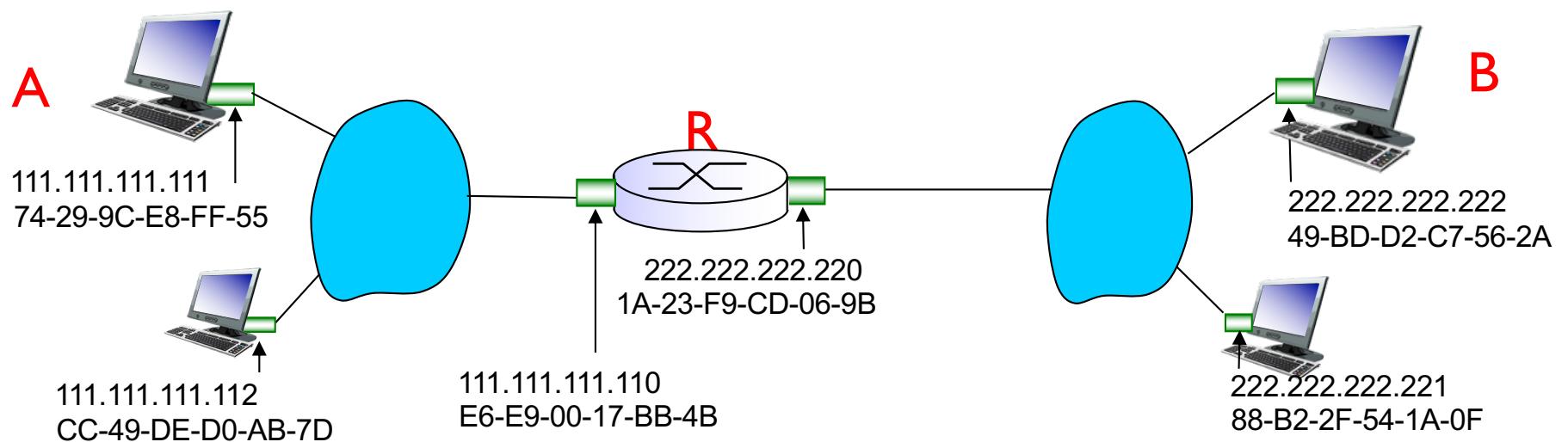
# ARP protocol: same LAN

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

# Addressing: routing to another LAN

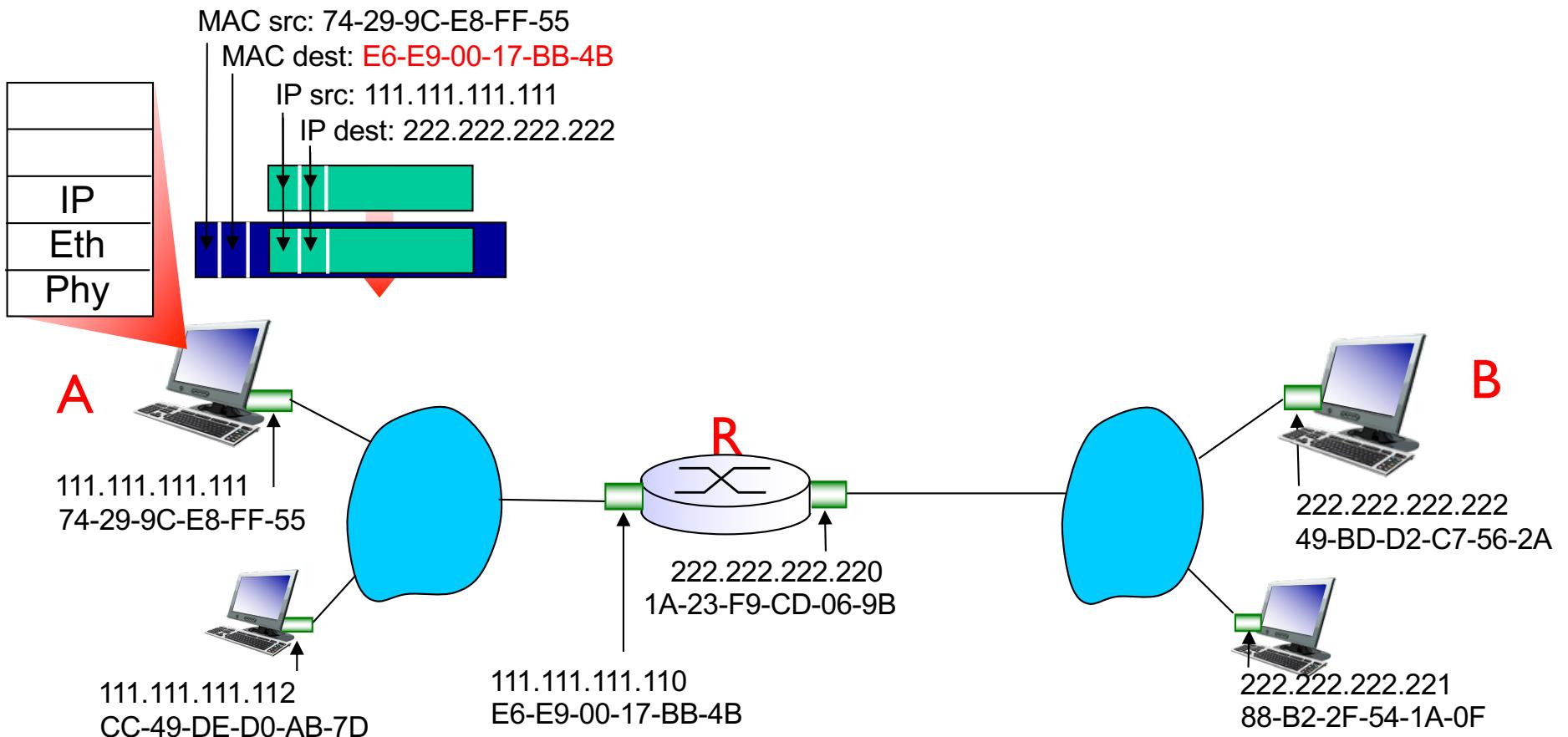
walkthrough: send datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address (how?)
  - How does A know B is not local (i.e., connected to the same LAN as A) ?
    - Subnet Mask (discovered via DHCP)
- assume A knows IP address of first hop router, R (how?)
  - Default router (discovered via DHCP)
- assume A knows R's MAC address (how?)
  - ARP



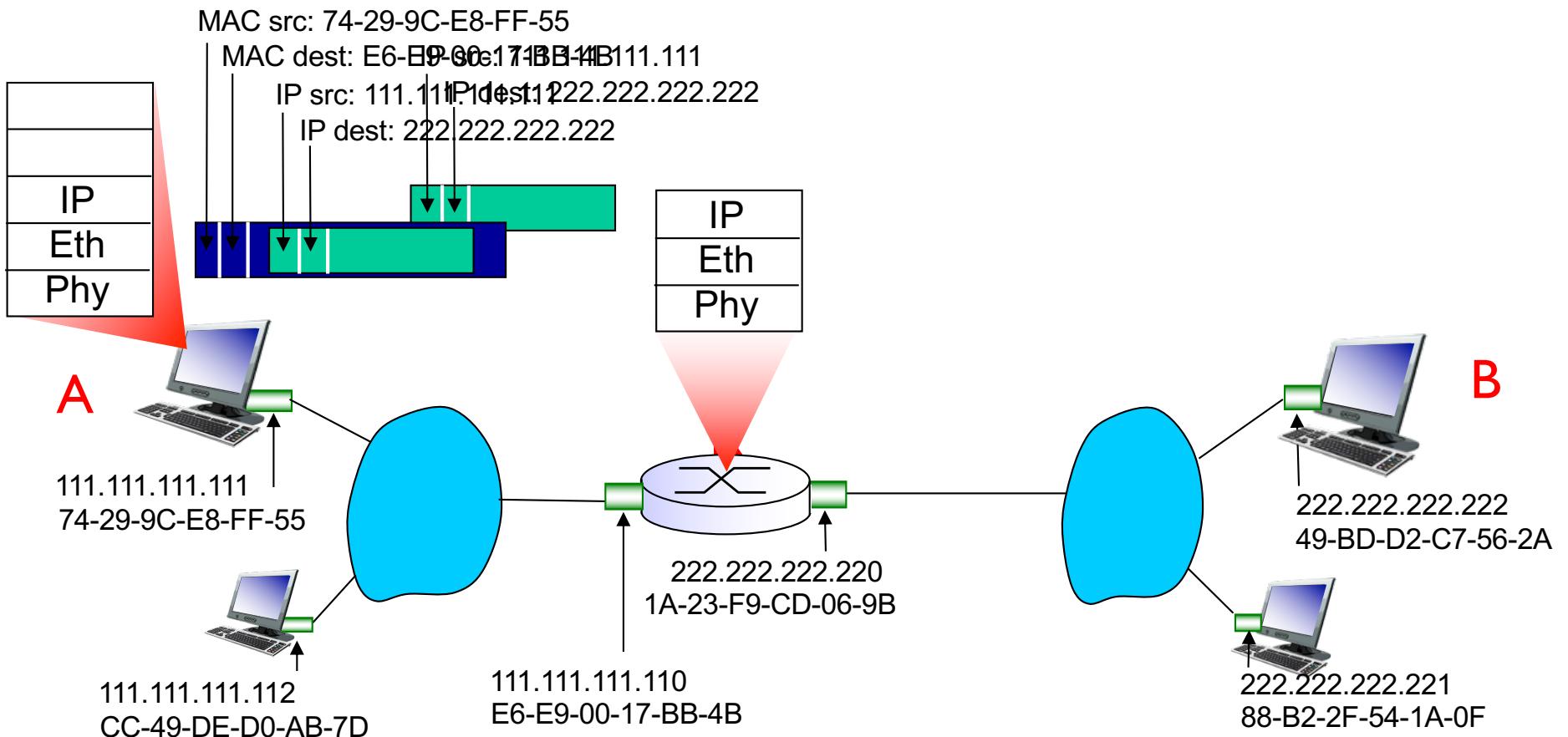
# Addressing: routing to another LAN

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



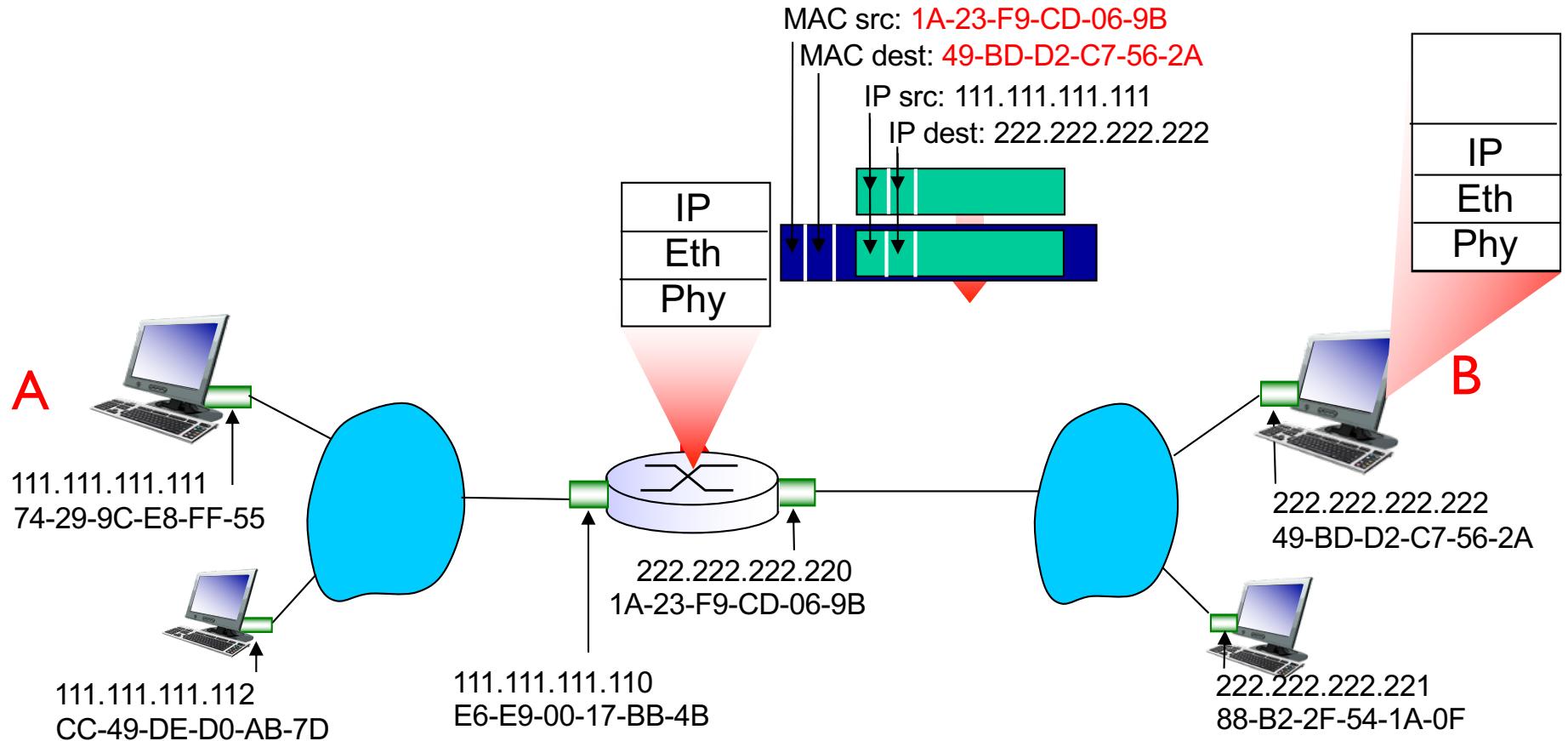
# Addressing: routing to another LAN

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



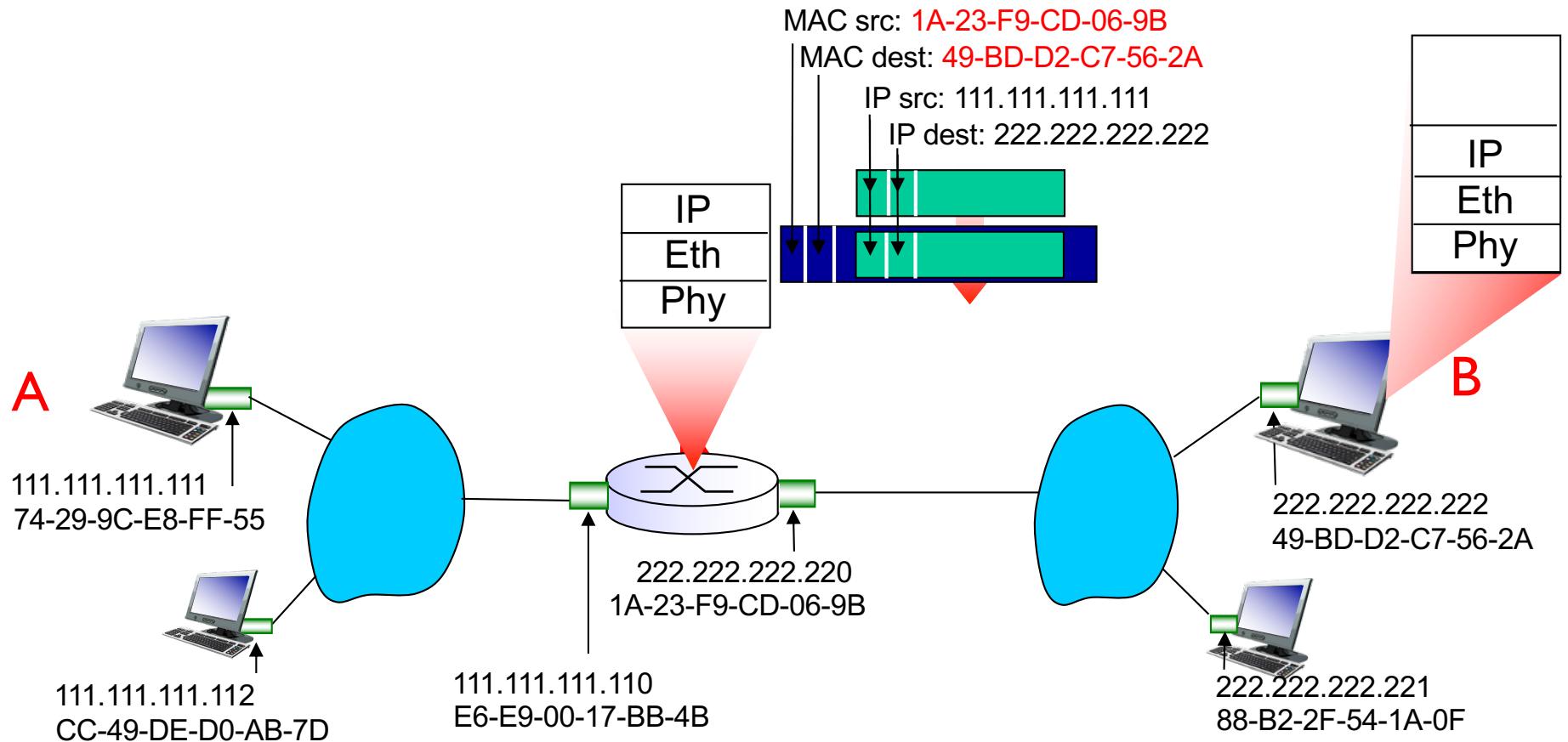
# Addressing: routing to another LAN

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



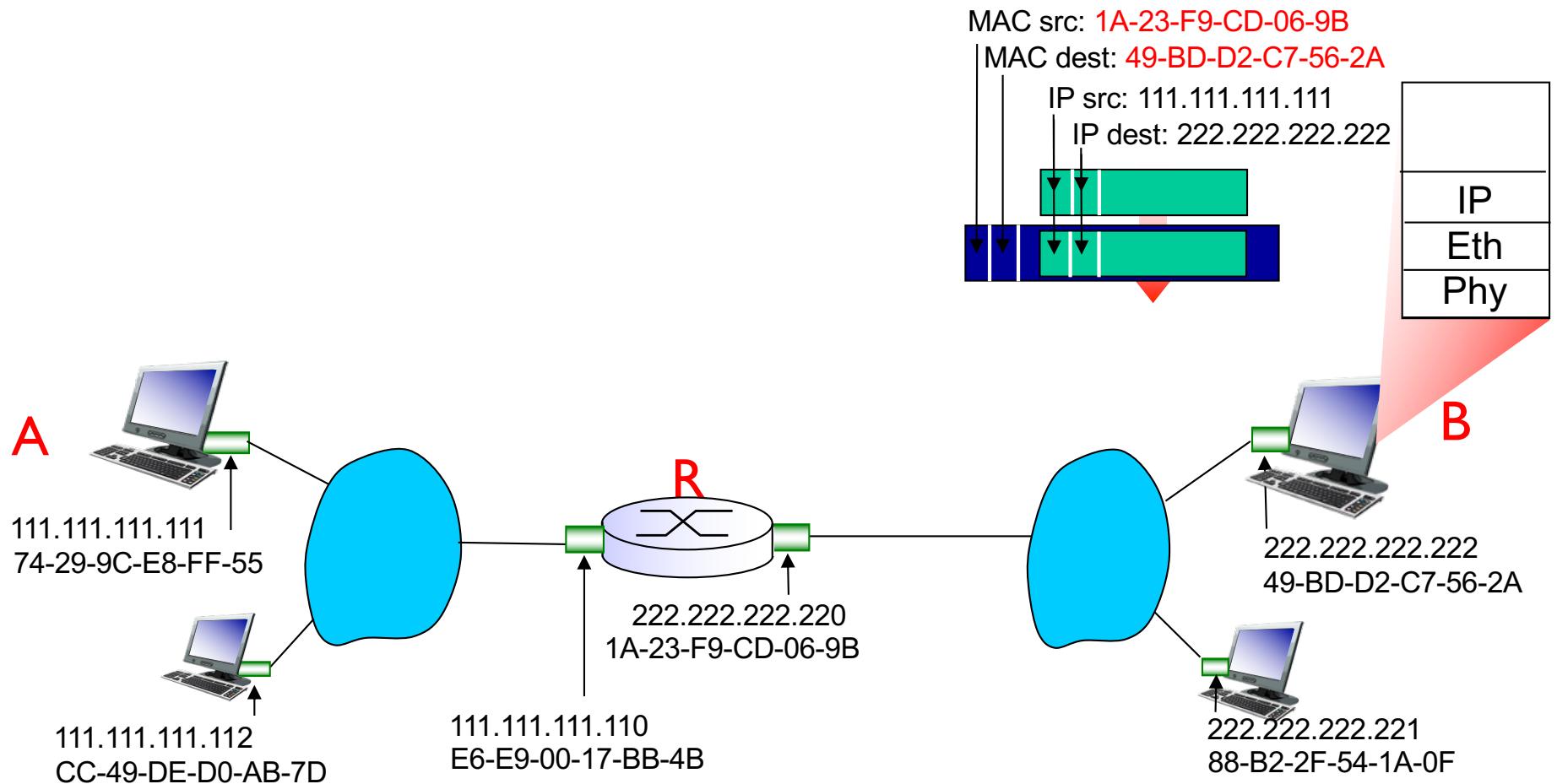
# Addressing: routing to another LAN

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



# Addressing: routing to another LAN

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



# Example ARP Table

C:\Windows\system32\cmd.exe		
C:\Users\admin>arp -a		
Interface: 192.168.150.155 --- 0xb	Internet Address	Physical Address
	192.168.150.2	00-10-db-82-4d-52
	192.168.150.10	00-0e-7f-af-6d-b8
	192.168.150.24	00-0f-fe-25-74-40
	192.168.150.32	00-0b-cd-6e-b8-2c
	192.168.150.36	00-0f-fe-3a-aa-3f
	192.168.150.42	00-0f-fe-87-1e-98
	192.168.150.48	00-0e-7f-63-8d-d1
	192.168.150.54	00-16-35-ae-3b-a9
	192.168.150.58	00-16-35-ae-39-53
	192.168.150.60	00-21-63-68-e9-29
	192.168.150.62	00-0f-fe-9b-e8-38
	192.168.150.78	00-0f-fe-3a-a7-d7
	192.168.150.90	00-0e-7f-f2-f8-e8
	192.168.150.92	00-0f-fe-3a-a7-96
	192.168.150.98	00-0f-fe-85-8d-6b
	192.168.150.114	00-0e-7f-6c-81-25
	192.168.150.144	00-22-5f-12-67-a2
	192.168.150.156	00-0f-fe-d1-7e-1e
	192.168.150.157	00-0f-fe-d1-7e-1e
	192.168.150.159	00-06-1b-c2-e1-f3
	192.168.150.208	00-19-66-32-53-25
	192.168.150.219	00-00-aa-8c-be-07
	192.168.150.221	00-0e-7f-64-5f-d0
	192.168.150.255	ff-ff-ff-ff-ff-ff
	224.0.0.22	01-00-5e-00-00-16
	224.0.0.251	01-00-5e-00-00-fb
	224.0.0.252	01-00-5e-00-00-fc
	224.0.1.134	01-00-5e-00-01-86
	239.255.255.250	01-00-5e-7f-ff-fa

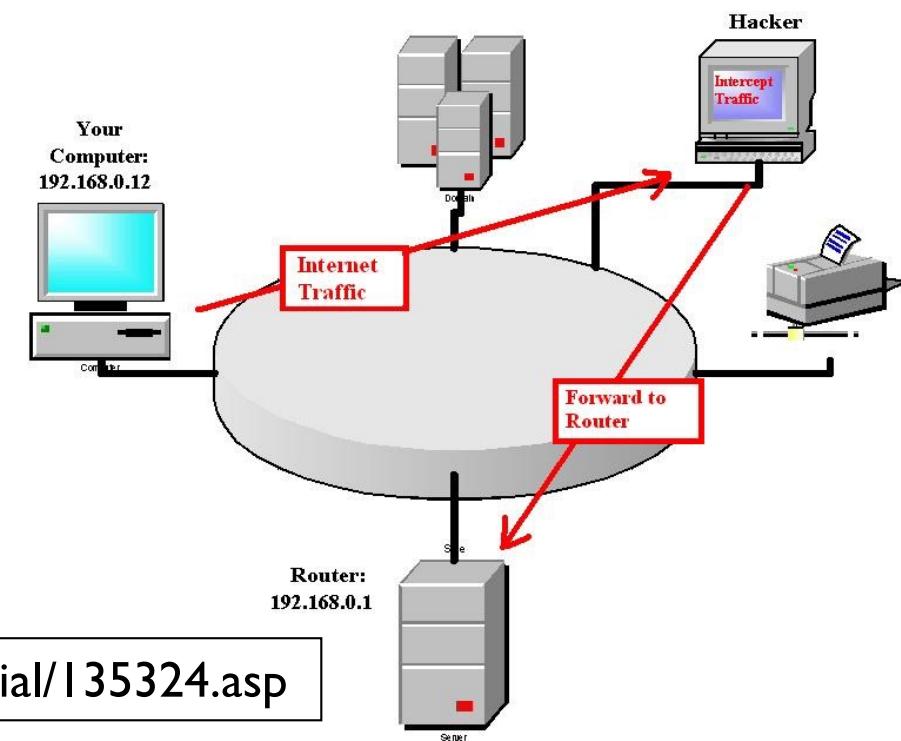
# Security Issues: ARP Cache Poisoning



- ❖ Denial of Service - Hacker replies back to an ARP query for a router NIC with a fake MAC address
- ❖ Man-in-the-middle attack - Hacker can insert his/her machine along the path between victim machine and gateway router
- ❖ Such attacks are generally hard to launch as hacker needs physical access to the network

## Solutions -

- Use Switched Ethernet with port security enabled (i.e., one host MAC address per switch port)
- Adopt static ARP configuration for small size networks
- Use ARP monitoring tools such as ARPWatch



<http://www.watchguard.com/infocenter/editorial/135324.asp>

# Link layer, LANs: outline

6.1 introduction, services

6.2 error detection,  
correction

6.3 multiple access  
protocols

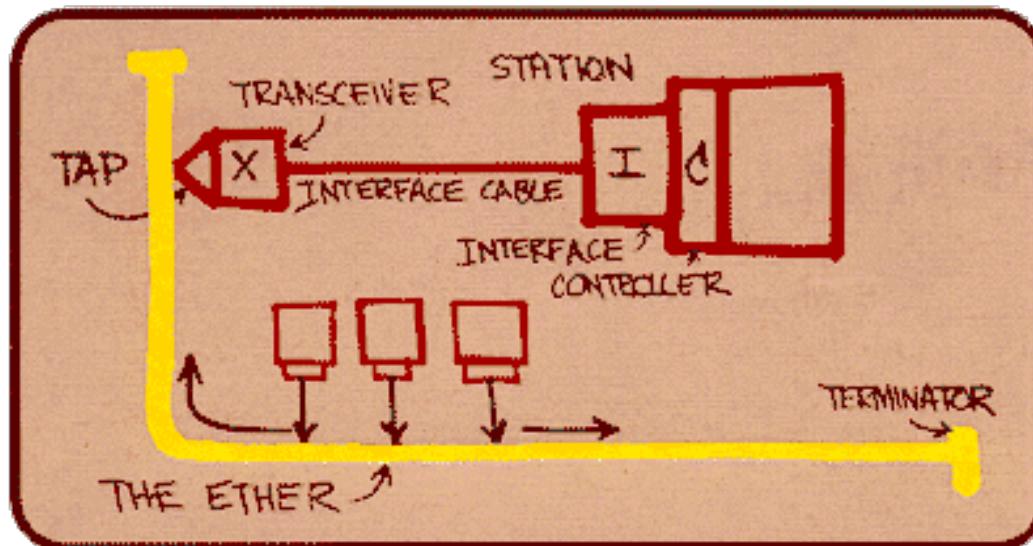
## 6.4 LANs

- addressing, ARP
- Ethernet
- switches

6.7 a day in the life of a  
web request

# Ethernet

Bob Metcalfe, Xerox PARC, visits Hawaii and gets an idea!



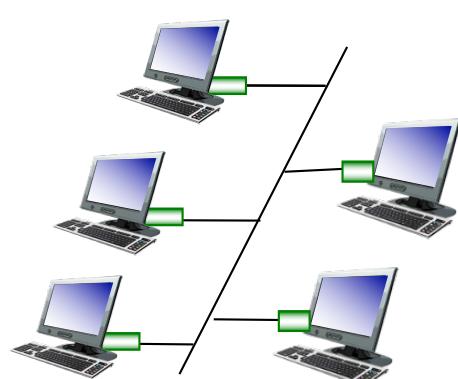
*Metcalfe's Ethernet sketch*

“dominant” wired LAN technology:

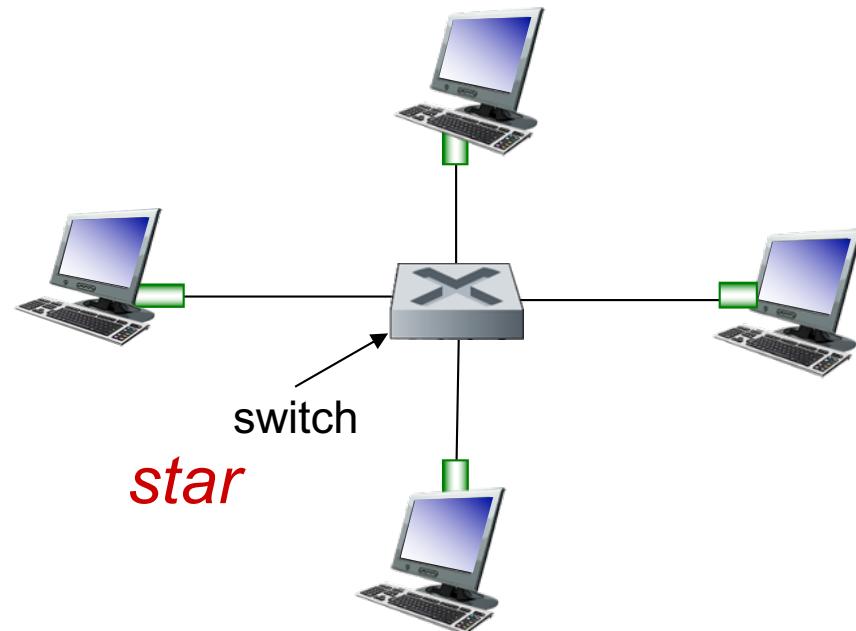
- ❖ first widely used LAN technology
- ❖ simpler, cheaper than token LANs and ATM
- ❖ kept up with speed race: 10 Mbps – 10 Gbps

# Ethernet: physical topology

- ❖ *bus*: popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
  - CSMA/CD for media access control
- ❖ *star*: prevails today
  - active *switch* in center
  - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)
  - No sharing, no CSMA/CD



*bus*: coaxial cable



# Ethernet frame structure

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

Preamble 7 Bytes	SFD 1 Byte	Dest MAC 6 Bytes	Source MAC 6 Bytes	Type/Length 2 Bytes	Payload 46-1500 Bytes	FCS/CRC 4 Bytes	Inter Frame Gap
---------------------	---------------	---------------------	-----------------------	------------------------	--------------------------	--------------------	-----------------

*preamble:*

- ❖ Start of frame is recognized by
  - Preamble : Seven bytes with pattern 10101010
  - Start of Frame Delimiter (SFD) : 10101011
- ❖ used to synchronize receiver, sender clock rates
- Inter Frame Gap is 12 Bytes (96 bits) of idle state
  - 0.96 microsec for 100 Mbit/s Ethernet
  - 0.096 microsec for Gigabit/s Ethernet

# Ethernet frame structure (more)

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



# Ethernet: unreliable, connectionless

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

# Link layer, LANs: outline

6.1 introduction, services

6.7 a day in the life of a  
web request

6.2 error detection,  
correction

6.3 multiple access  
protocols

## 6.4 LANs

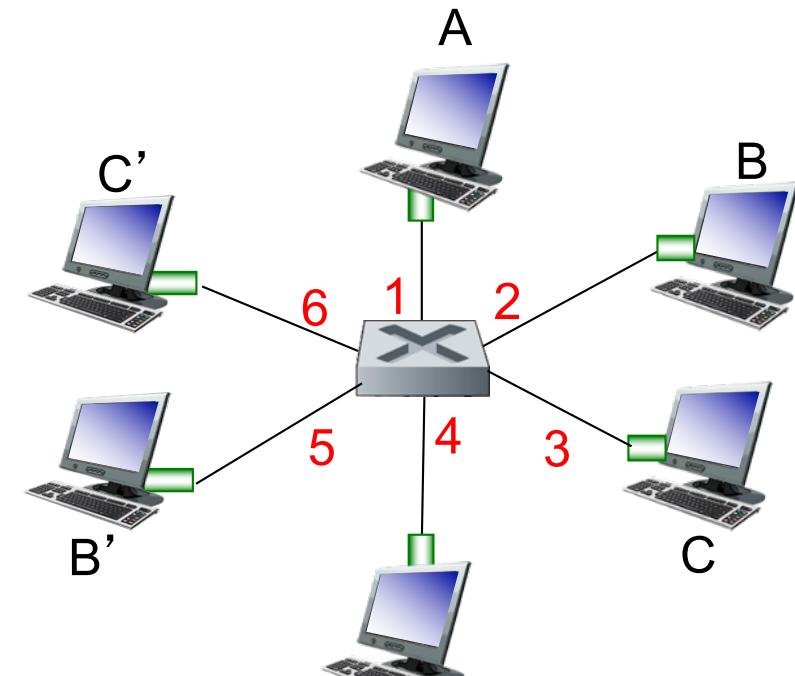
- addressing, ARP
- Ethernet
- switches

# Ethernet switch

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

# Switch: multiple simultaneous transmissions

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

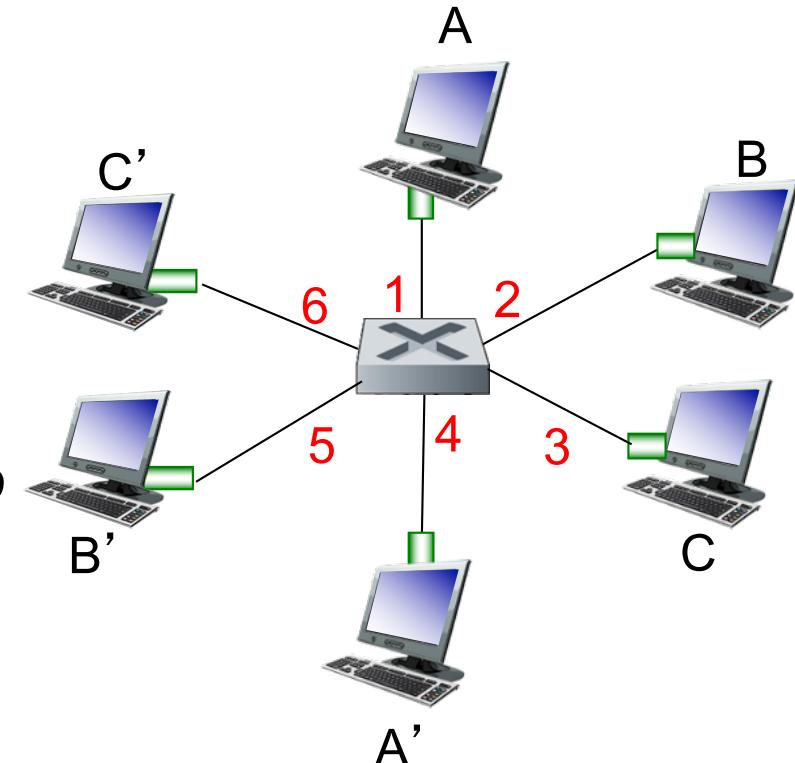


*switch with six interfaces  
(1,2,3,4,5,6)*

# Switch forwarding table

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

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



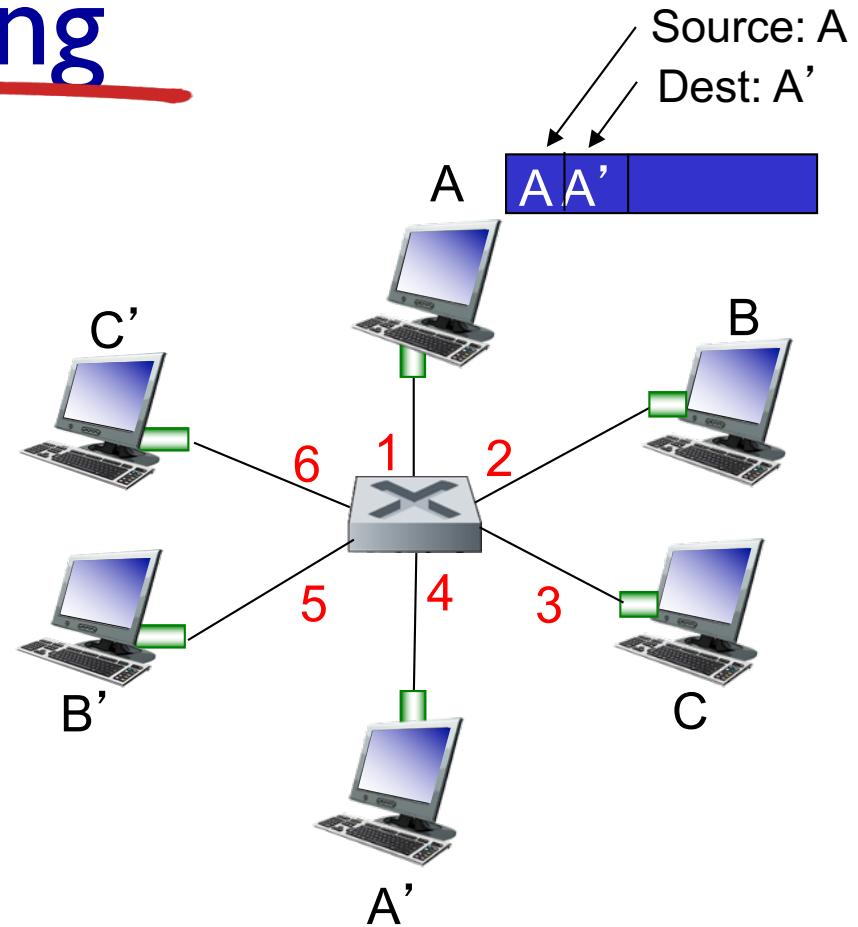
**Q:** how are entries created, maintained in switch table?

- something like a *routing protocol?*

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

# Switch: self-learning

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



MAC addr	interface	TTL
A	1	60

*Switch table  
(initially empty)*

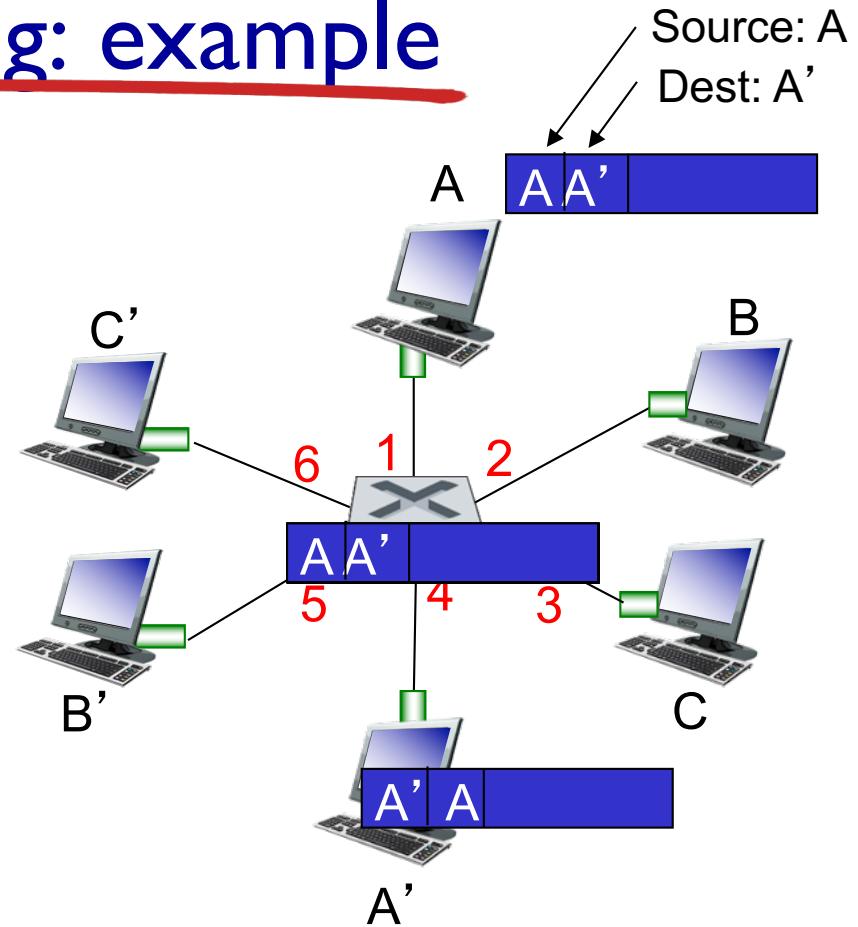
# Switch: frame filtering/forwarding

when frame received at switch:

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

## Self-learning, forwarding: example

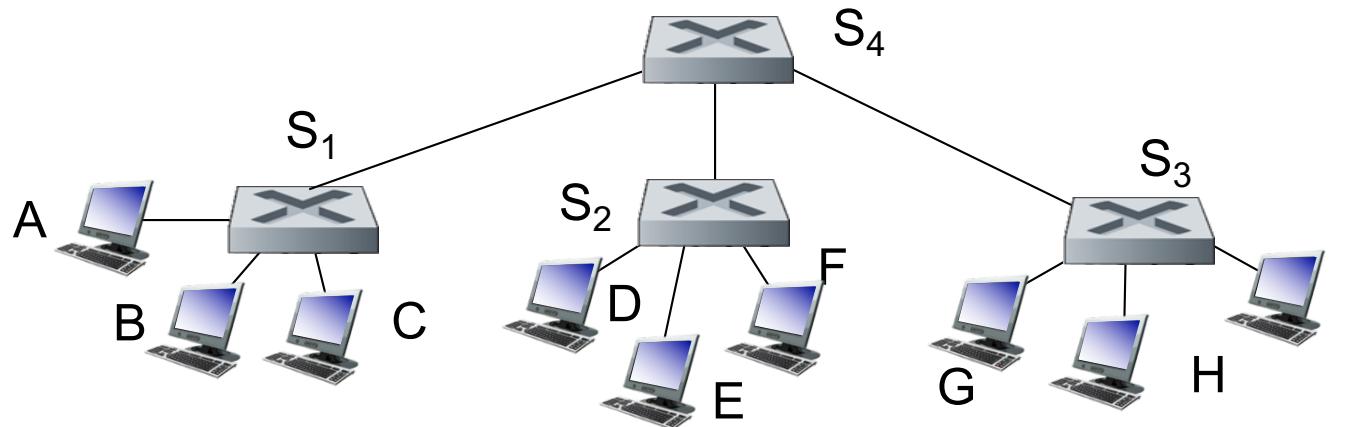
- ❖ frame destination,  $A'$ , location unknown: *flood*
- ❖ destination  $A$  location known: *selectively send on just one link*



*switch table  
(initially empty)*

# Interconnecting switches

- ❖ switches can be connected together



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

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

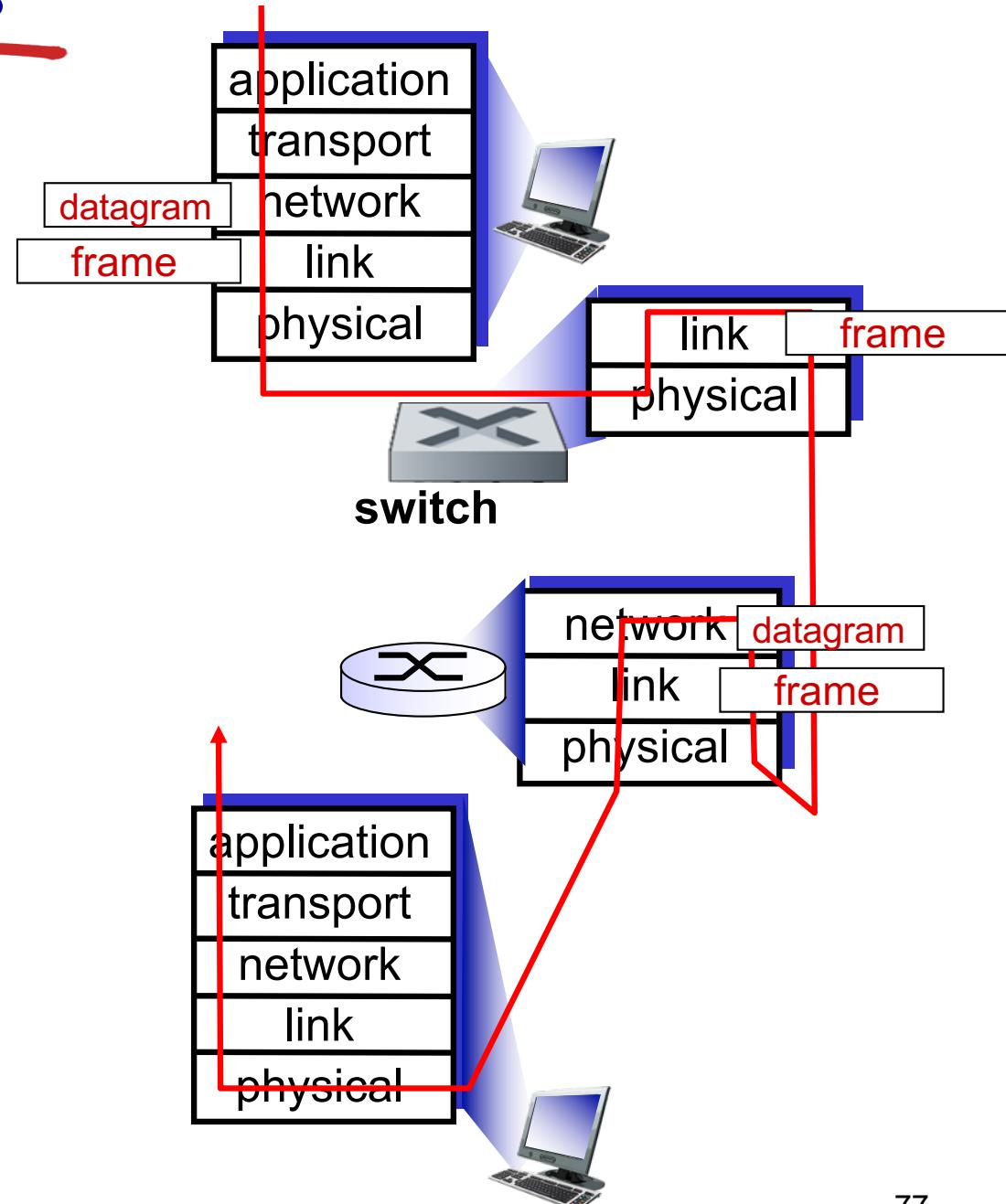
# Switches vs. routers

both are **store-and-forward**:

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

both have **forwarding tables**:

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



# Security Issues

- ❖ In a switched LAN once the switch table entries are established frames are not broadcast
  - Sniffing frames is harder than pure broadcast LANs
  - Note: attacker can still sniff broadcast frames and frames for which there are no entries (as they are broadcast)
- ❖ Switch Poisoning: Attacker fills up switch table with bogus entries by sending large # of frames with bogus source MAC addresses
- ❖ Since switch table is full, genuine packets frequently need to be broadcast as previous entries have been wiped out

# Quiz

- ❖ A switch can
  - A. Filter a frame
  - B. Forward a frame
  - C. Extend a LAN
  - D. All of the above

Answer: D

# Quiz

- ❖ The \_\_\_\_\_ will typically change from hop to hop, but the \_\_\_\_\_ will typically remain the same

Answer: D

- A. Source MAC address, destination MAC address
- B. Source IP address, destination IP address
- C. Source & destination IP addresses, source & destination MAC addresses
- D. Source & destination MAC addresses, source & destination IP addresses

# Link layer, LANs: outline

6.1 introduction, services

6.7 a day in the life of a  
web request

6.2 error detection,  
correction

6.3 multiple access  
protocols

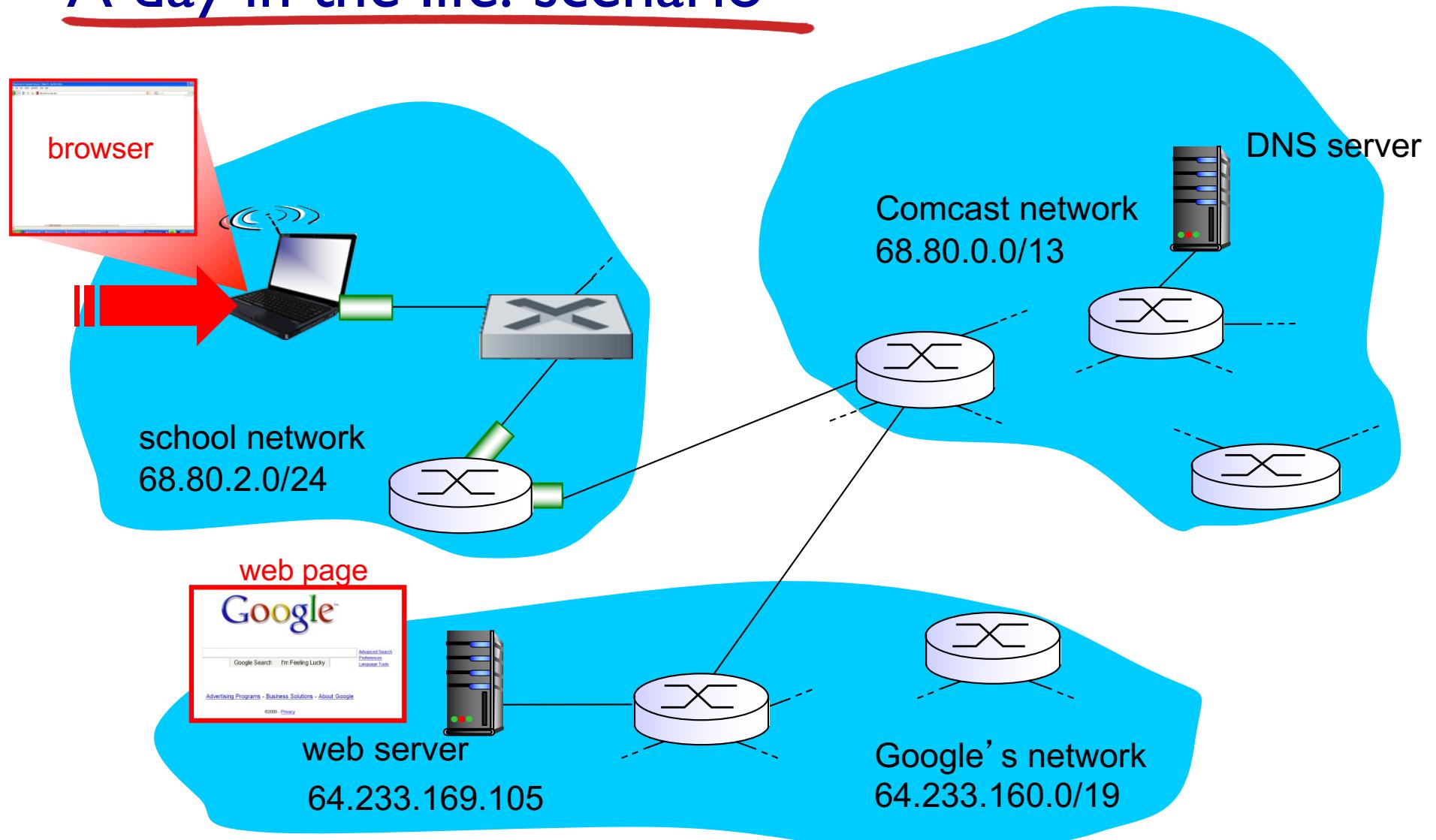
6.4 LANs

- addressing, ARP
- Ethernet
- switches

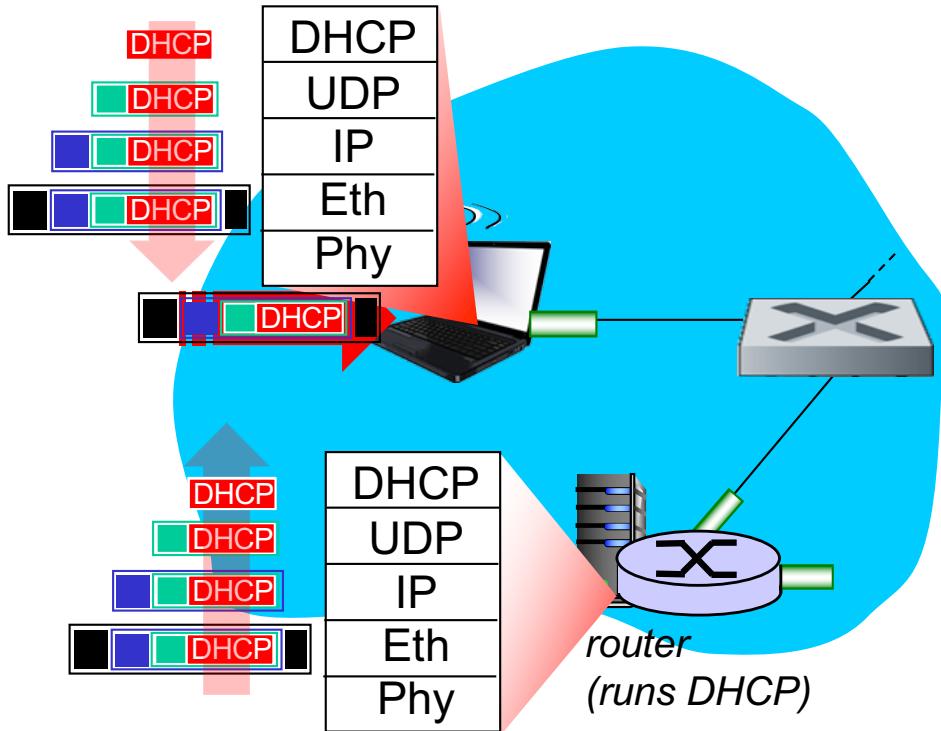
## Synthesis: a day in the life of a web request

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

## A day in the life: scenario

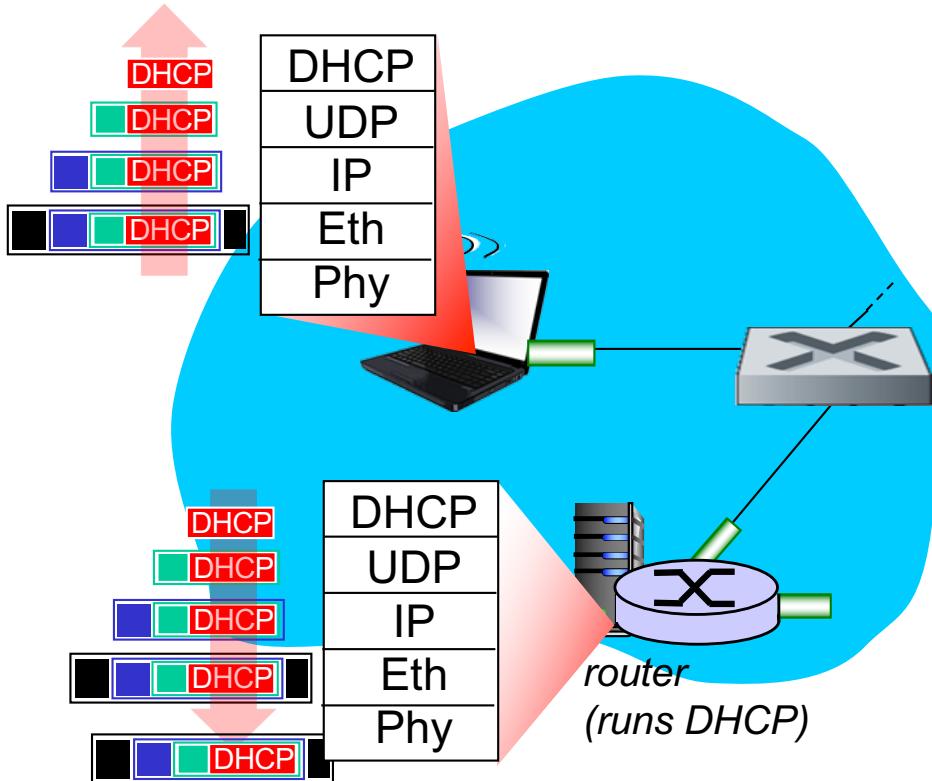


# A day in the life... connecting to the Internet



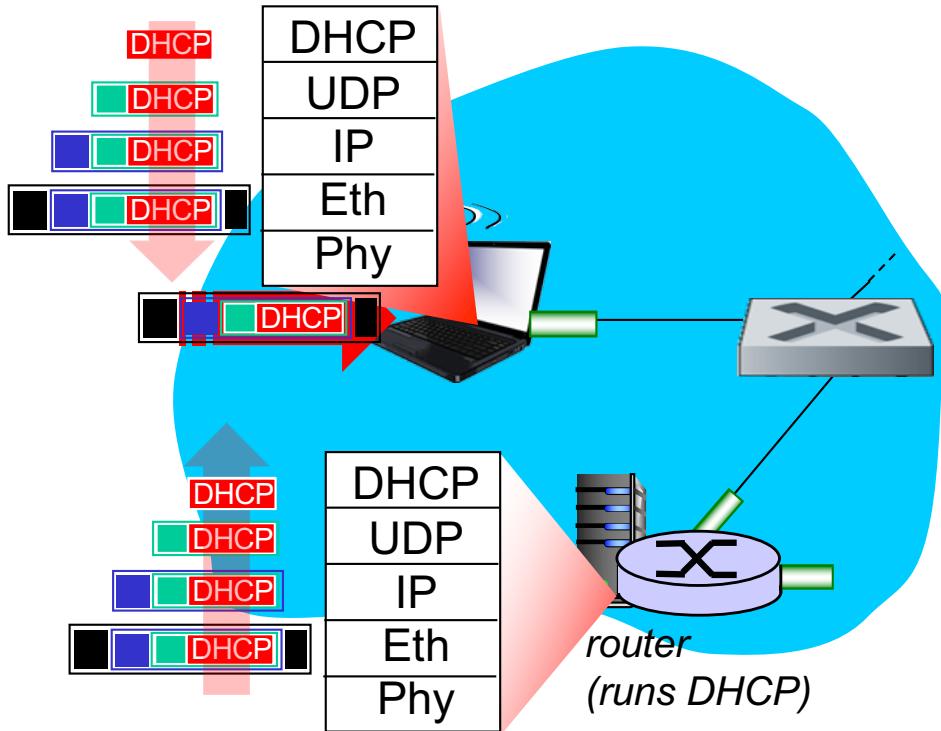
- ❖ connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use **DHCP**
- ❖ DHCP **Discover** Message *encapsulated* in **UDP**, *encapsulated* in **IP**, *encapsulated* in **802.3** Ethernet
- ❖ Ethernet frame *broadcast* (dest: FFFFFFFFFFFF) on LAN, received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

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



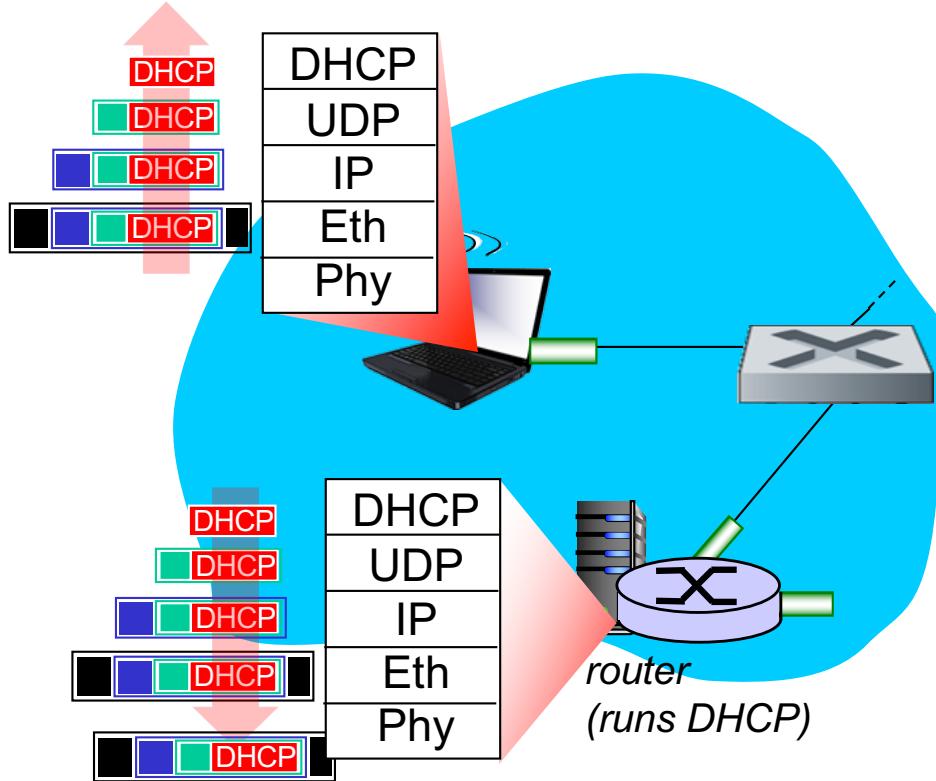
- ❖ DHCP server formulates *DHCP Offer* message containing client's IP address
- ❖ encapsulation at DHCP server; frame again broadcasted on LAN
- ❖ DHCP client receives DHCP Offer message

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



- ❖ The client initiates **DHCP Request** message
- ❖ DHCP Request *encapsulated* in **UDP**, encapsulated in **IP**, encapsulated in **802.3 Ethernet**
- ❖ Ethernet frame *broadcast* (dest: FFFFFFFFFFFF) on LAN, received at router running **DHCP server**
- ❖ Ethernet *demuxed* to IP demuxed, UDP demuxed to DHCP

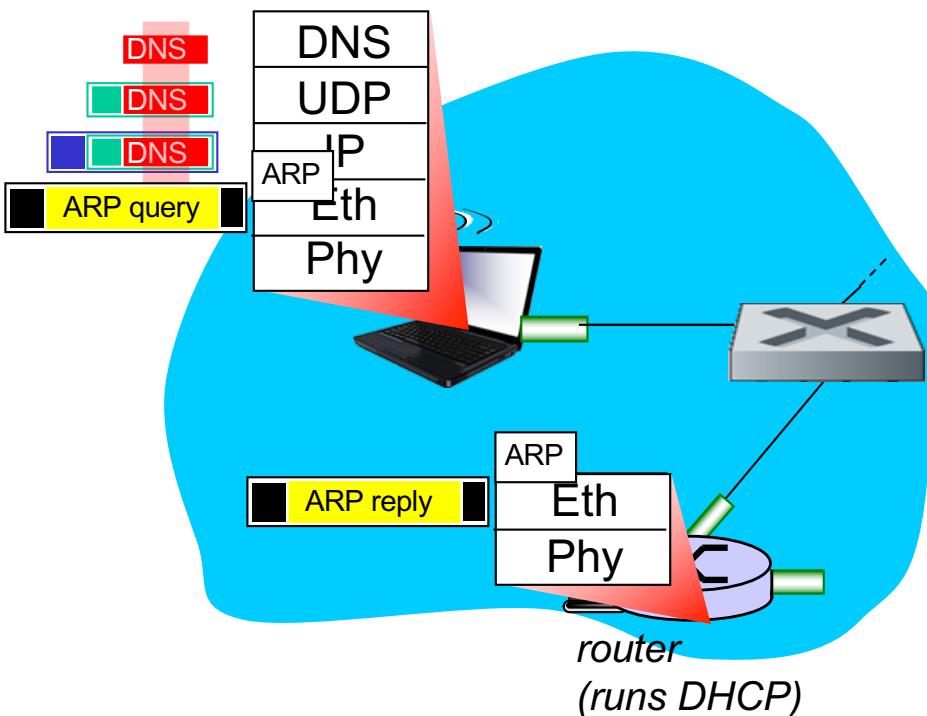
# A day in the life... connecting to the Internet



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

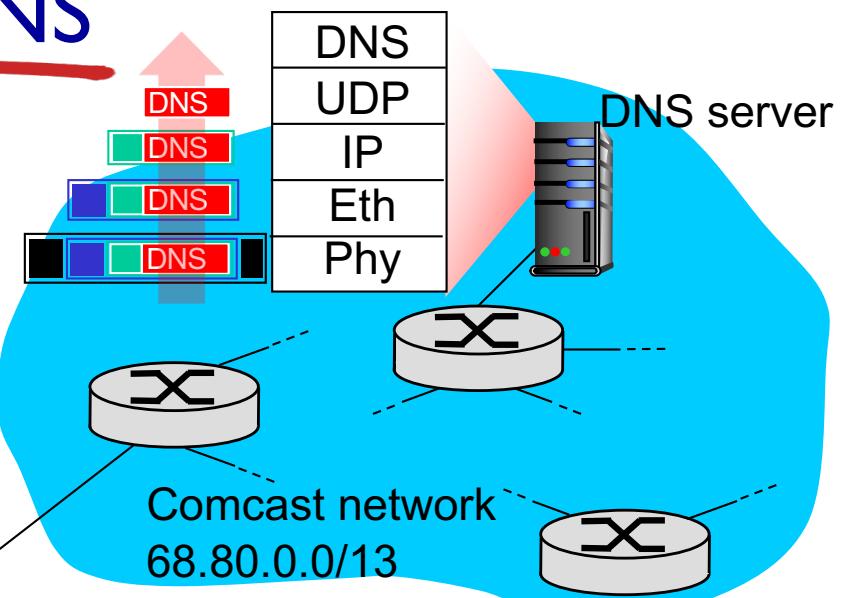
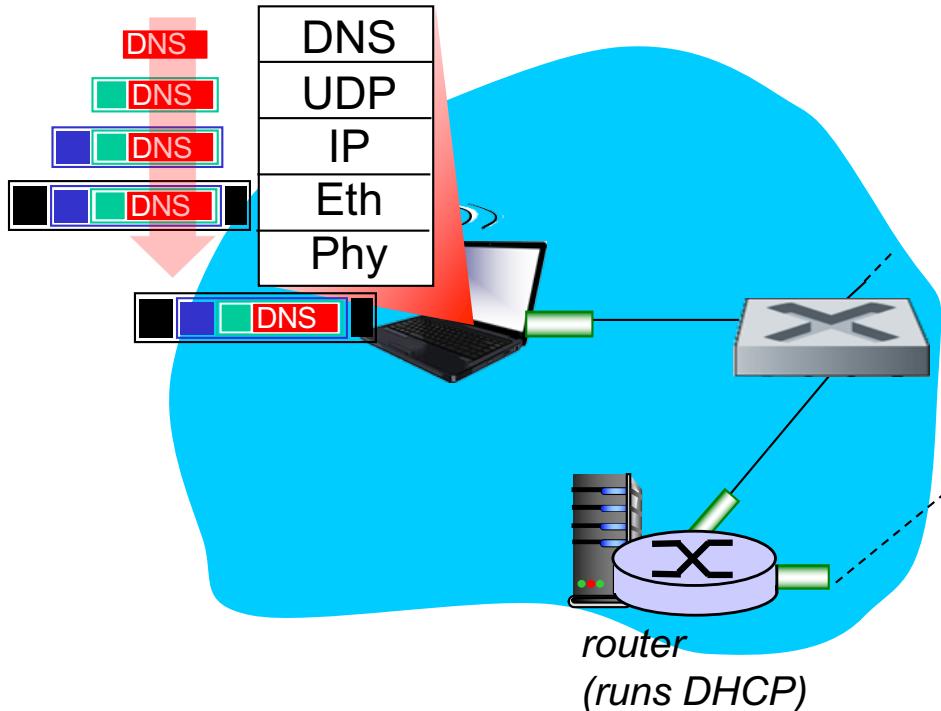
*Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router*

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



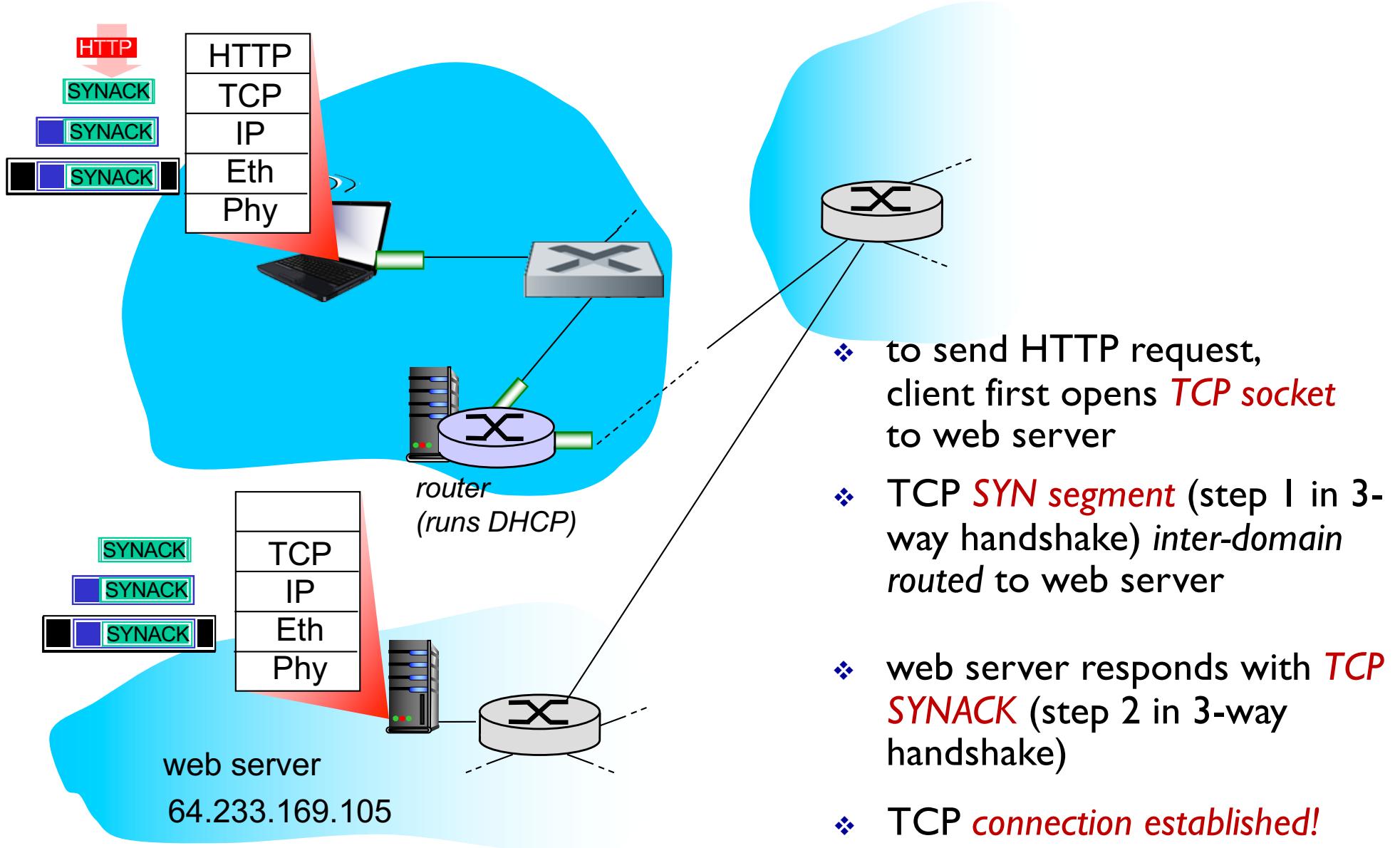
- ❖ before sending **HTTP** request, need IP address of [www.google.com](http://www.google.com): **DNS**
- ❖ DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to DNS server, need MAC address of first hop router: **ARP**
- ❖ **ARP query** broadcast, received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

# A day in the life... using DNS

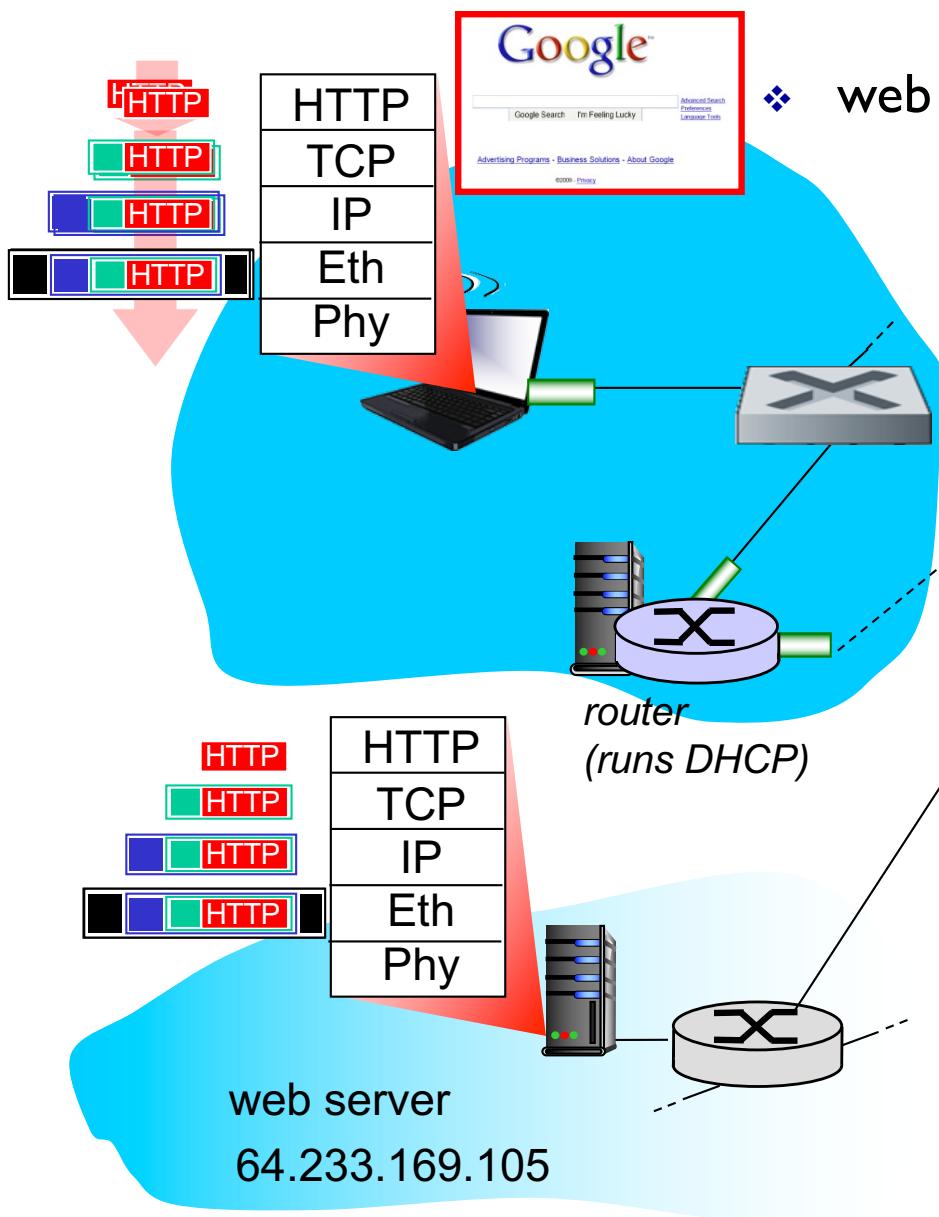


- ❖ IP datagram containing DNS query forwarded via LAN switch from client to first hop router
- ❖ IP datagram forwarded from first hop router in campus network into Comcast network, routed (tables created by *RIP, OSPF, IS-IS* and/or *BGP* routing protocols) to DNS server
- ❖ demux'ed to DNS server
- ❖ DNS server replies to client with IP address of [www.google.com](http://www.google.com)

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



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



❖ web page *finally (!!)* displayed

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

# Link Layer: Summary

- ❖ principles behind data link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
- ❖ instantiation and implementation of various link layer technologies
  - Ethernet
  - switched LANS

# Link Layer: let's take a breath

- ❖ journey down protocol stack *complete* (except PHY)
- ❖ solid understanding of networking principles, practice
- ❖ ..... could stop here .... but *lots* of interesting topics!
  - **Wireless**
  - **Security**

# COMP 3331/9331: Computer Networks and Applications

Week 10  
Network Security

**Reading Guide: Chapter 8: 8.1 – 8.5**



Complete your myExperience and shape  
the future of education at UNSW.

Click the  my Experience link in Moodle

or login to [myExperience.unsw.edu.au](https://myexperience.unsw.edu.au)

(use z1234567@ad.unsw.edu.au to login)

The survey is confidential, your identity will never be released  
Survey results are not released to teaching staff until after your results are published

# Network Security: Overview

*Our goals:*

- ❖ understand principles of network security:
  - cryptography and its *many* uses beyond “confidentiality”
  - authentication
  - message integrity

# Network Security: roadmap

8.1 *What is network security?*

8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

8.6 - 8.9 SSL, IPSec, Firewall/IDS - **not covered.**

**There are several security electives offered**

# What is network security?

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

- sender encrypts message
- receiver decrypts message

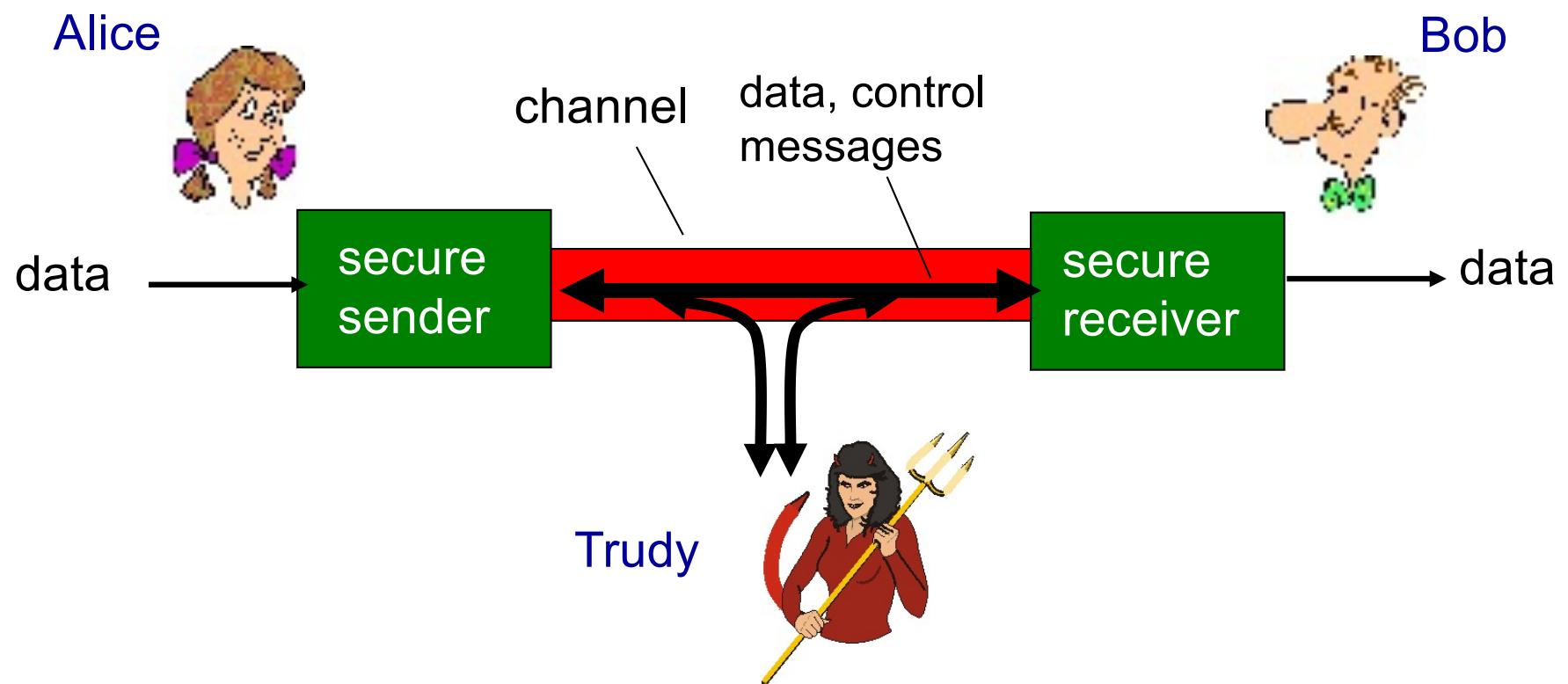
***authentication:*** sender, receiver want to confirm identity of each other

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

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

# Friends and enemies: Alice, Bob, Trudy

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



# Who might Bob, Alice be?

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

# There are bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: A lot!

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

# Network Security: roadmap

8.1 What is network security?

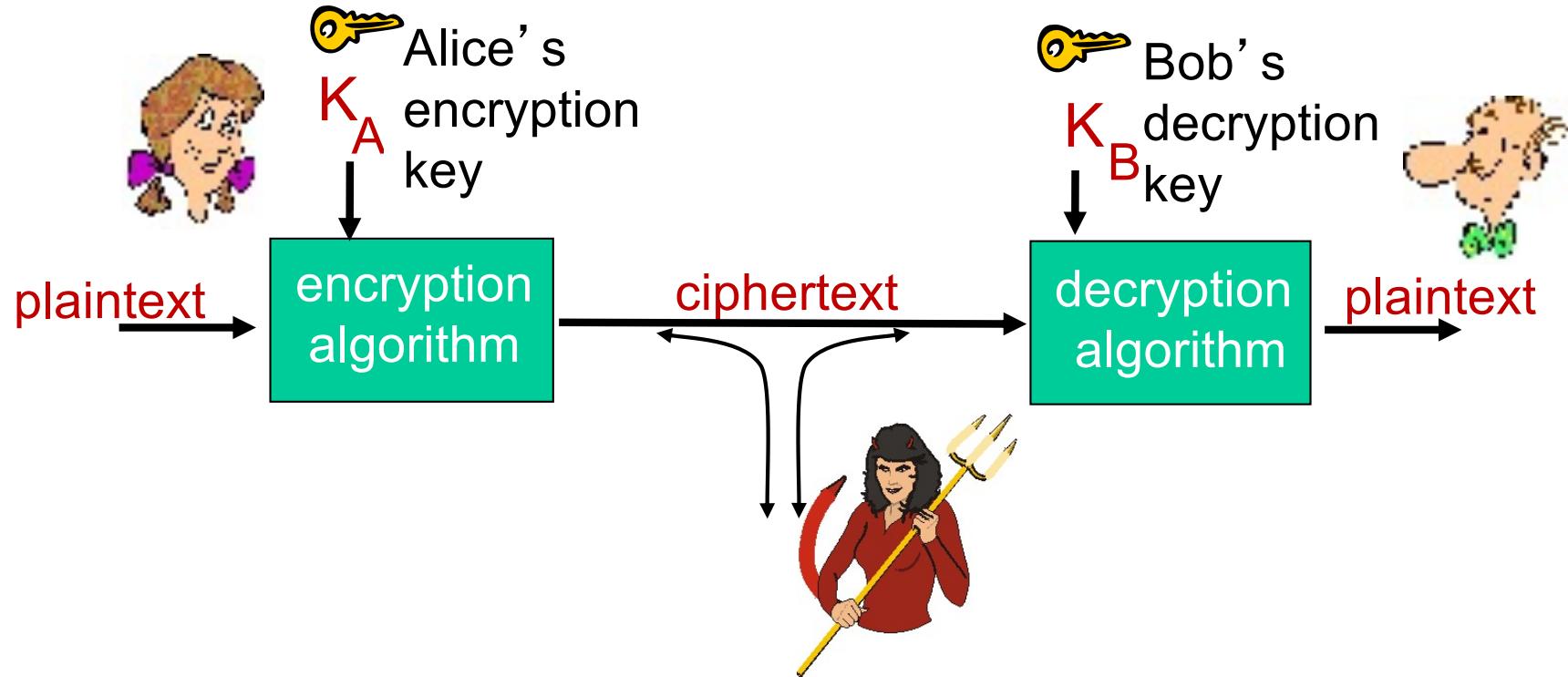
8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

# The language of cryptography

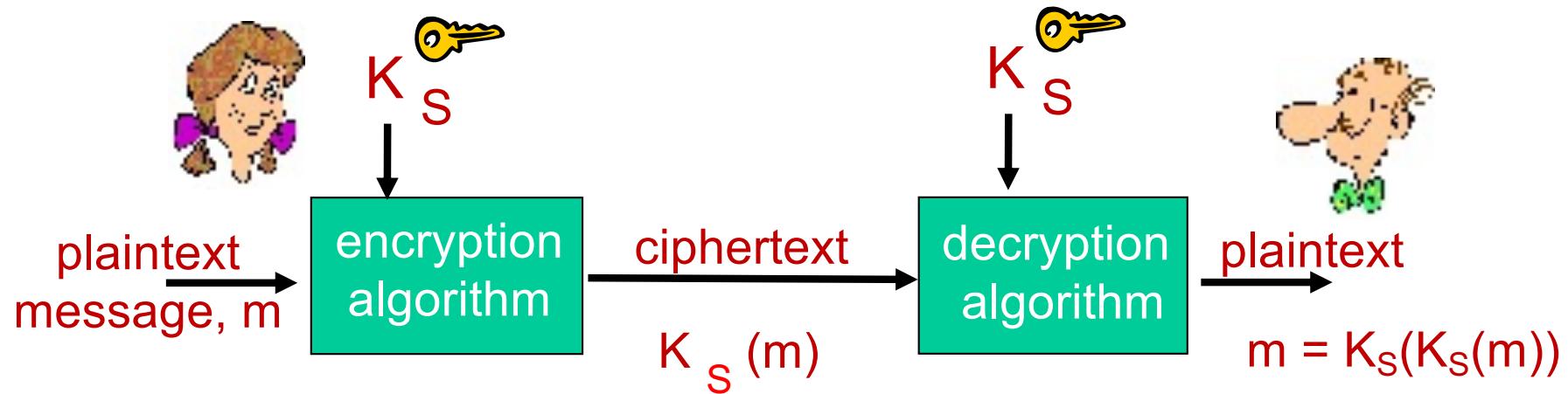


$m$  plaintext message

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

$m = K_B(K_A(m))$

# Symmetric key cryptography



**symmetric key crypto:** Bob and Alice share same (symmetric) key:  $K_S$

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

# Simple encryption scheme

**substitution cipher:** substituting one thing for another

- ❖ monoalphabetic cipher: substitute one letter for another
- ❖ Ceaser Cipher: replace each letter of the alphabet with the letter standing three places further down the alphabet.

Plain : a b c d e f g h i j k l m n o p q r s t u v w x y z  
cipher: d e f g h i j k l m n o p q r s t u v w x y z a b c

e.g.:      **Plaintext:** meet me after the party

**ciphertext:** phhw ph diwhu wkh sduwb



**Encryption key:**  $c = (p+3) \text{ mod } 26$

Each plaintext letter  $p$  substituted by the ciphertext letter  $c$

In general, we have  $c = (p+k) \text{ mode } 26$

where  $k$  is in range 1 to 25

# Simple encryption scheme

- ❖ With only 25 possible keys, the Caeser cipher is vulnerable to brute force cryptanalysis
- ❖ Cipher can be any permutation of the 26 alphabet characters

**plaintext:** abcdefghijklmnopqrstuvwxyz  
**ciphertext:** mnbvcxzasdfghjklpoiuytrewq

e.g.: **Plaintext:** bob. i love you. alice  
**ciphertext:** nkn. s gktc wky. mgsbc

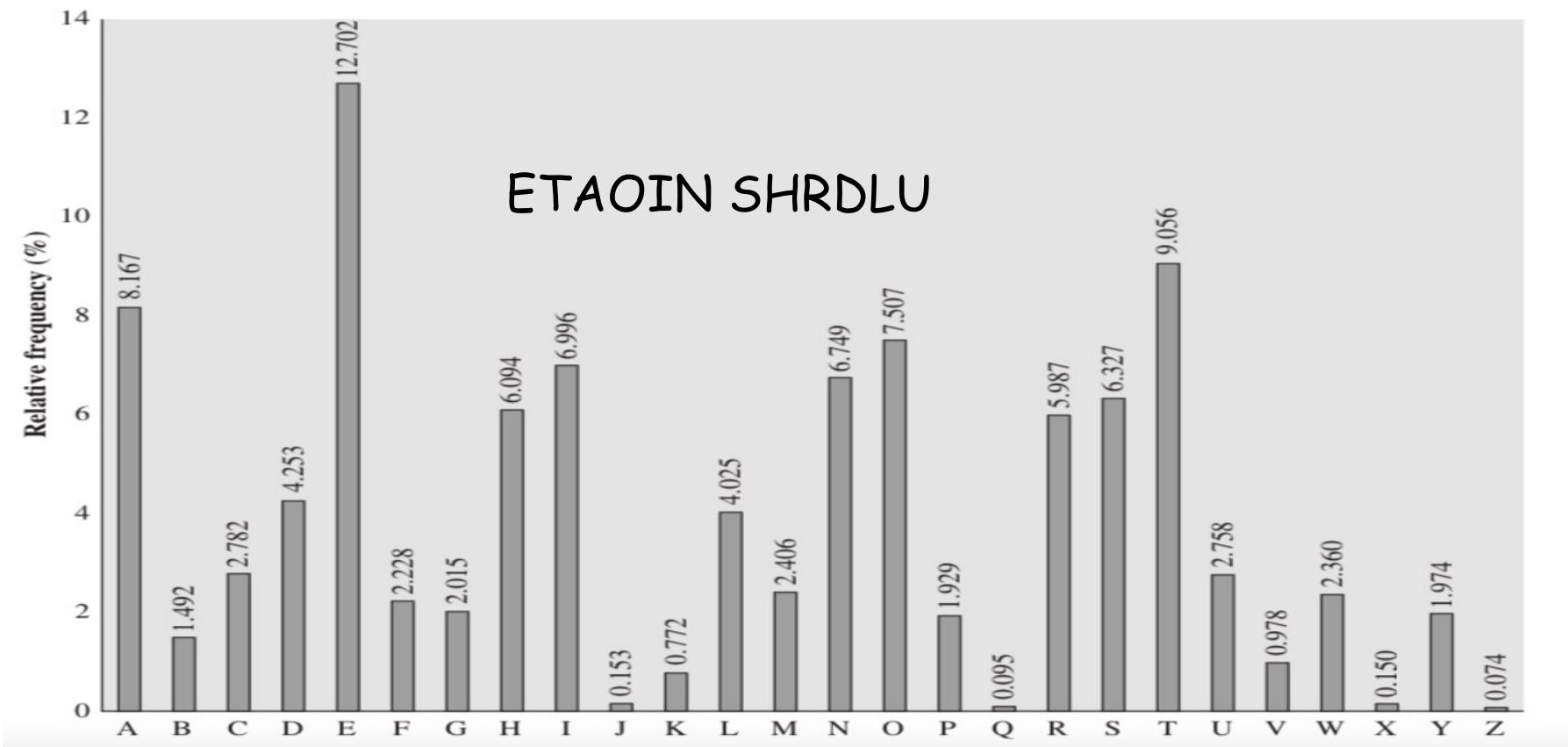
 **Encryption key:** mapping from set of 26 letters  
to set of 26 letters

We have  $26!$  ( $> 4 \times 10^{26}$ ) possible keys

# Breaking an encryption scheme

- ❖ **cipher-text only attack:**  
Trudy has ciphertext she can analyze
- ❖ **two approaches:**
  - brute force: search through all keys
  - statistical analysis
- ❖ **known-plaintext attack:**  
Trudy has (part of) plaintext corresponding to ciphertext
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,b
- ❖ **chosen-plaintext attack:**  
Trudy can get ciphertext for chosen plaintext

# Breaking an encryption scheme



Frequency Histogram Analysis for letters in English language

Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet

## A more sophisticated encryption approach

- ❖ Polyalphabet ciphers
- ❖ n substitution ciphers,  $M_1, M_2, \dots, M_n$
- ❖ cycling pattern:
  - e.g.,  $n=4$ : and key is  $M_1, M_3, M_4, M_3, M_2; M_1, M_3, M_4, M_3, M_2; \dots$
- ❖ for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  - dog: d from  $M_1$ , o from  $M_3$ , g from  $M_4$



*Encryption key:* n substitution ciphers, and cyclic pattern

# Two types of symmetric ciphers

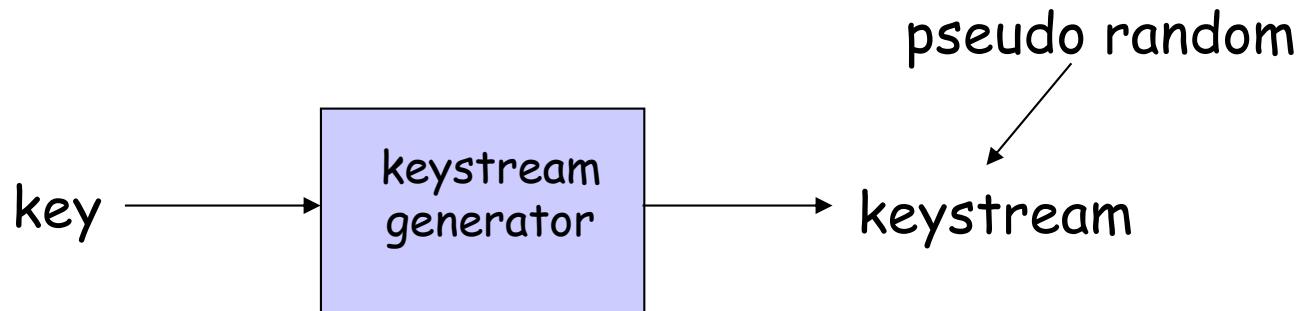
- ❖ **Stream ciphers**

- encrypt one bit at time

- ❖ **Block ciphers**

- Break plaintext message in equal-size blocks
  - Encrypt each block as a unit

# Stream Ciphers



- ❖ Combine each bit of keystream with bit of plaintext to get bit of ciphertext
- ❖  $m(i)$  = ith bit of message
- ❖  $ks(i)$  = ith bit of keystream
- ❖  $c(i)$  = ith bit of ciphertext
- ❖  $c(i) = ks(i) \oplus m(i)$  ( $\oplus$  = exclusive or)
- ❖  $m(i) = ks(i) \oplus c(i)$

# RC4 Stream Cipher

- ❖ RC4 is a popular stream cipher
  - Extensively analyzed and considered good
  - Key can be from 1 to 256 bytes
  - Used in WEP for 802.11
  - Known to have vulnerabilities
  - Many other alternatives: ChaCha, SOBER, SEAL, ...

# Block Cipher

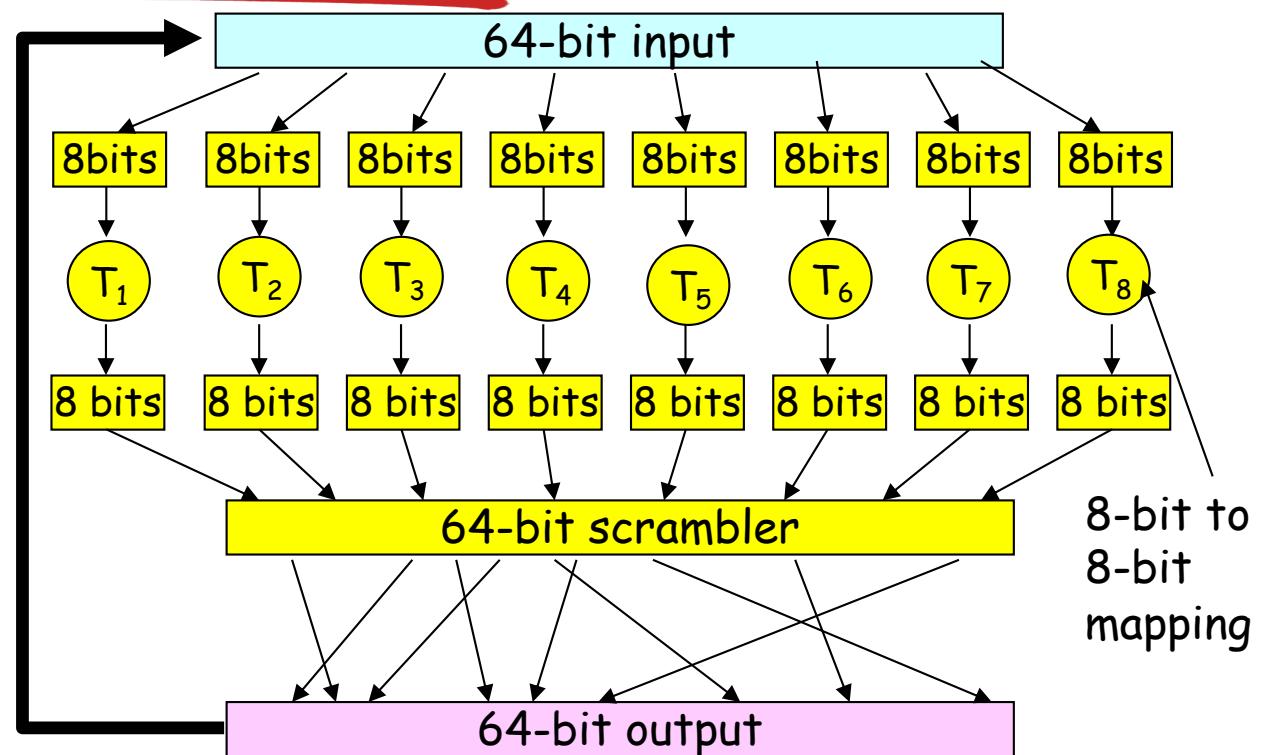
- ❖ Ciphertext processed as  $k$  bit blocks
- ❖ 1-to-1 mapping is used to map  $k$ -bit block of plaintext to  $k$ -bit block of ciphertext
- ❖ E.g:  $k=3$  (see table)
  - $010110001111 \Rightarrow 101000111001$
- ❖ Possible permutations =  $8!$  (40,320)
- ❖ To prevent brute force attacks
  - Choose large  $K$  (64, 128, etc)
- ❖ Full-table block ciphers not scalable
  - E.g., for  $k = 64$ , a table with  $2^{64}$  entries required
  - instead use function that simulates a randomly permuted table

Input	Output
000	110
111	001
001	111
010	101
011	100
100	011
101	010
110	000

# Block Cipher (contd.)

loop for  
n rounds

- ❖ If only a single round, then one bit of input affects at most 8 bits of output
- ❖ In 2<sup>nd</sup> round, the 8 affected bits get scattered and inputted into multiple substitution boxes
- ❖ How many rounds?
  - How many times do you need to shuffle cards
  - Becomes less efficient as n increases



# Symmetric key crypto: DES

## DES: Data Encryption Standard

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

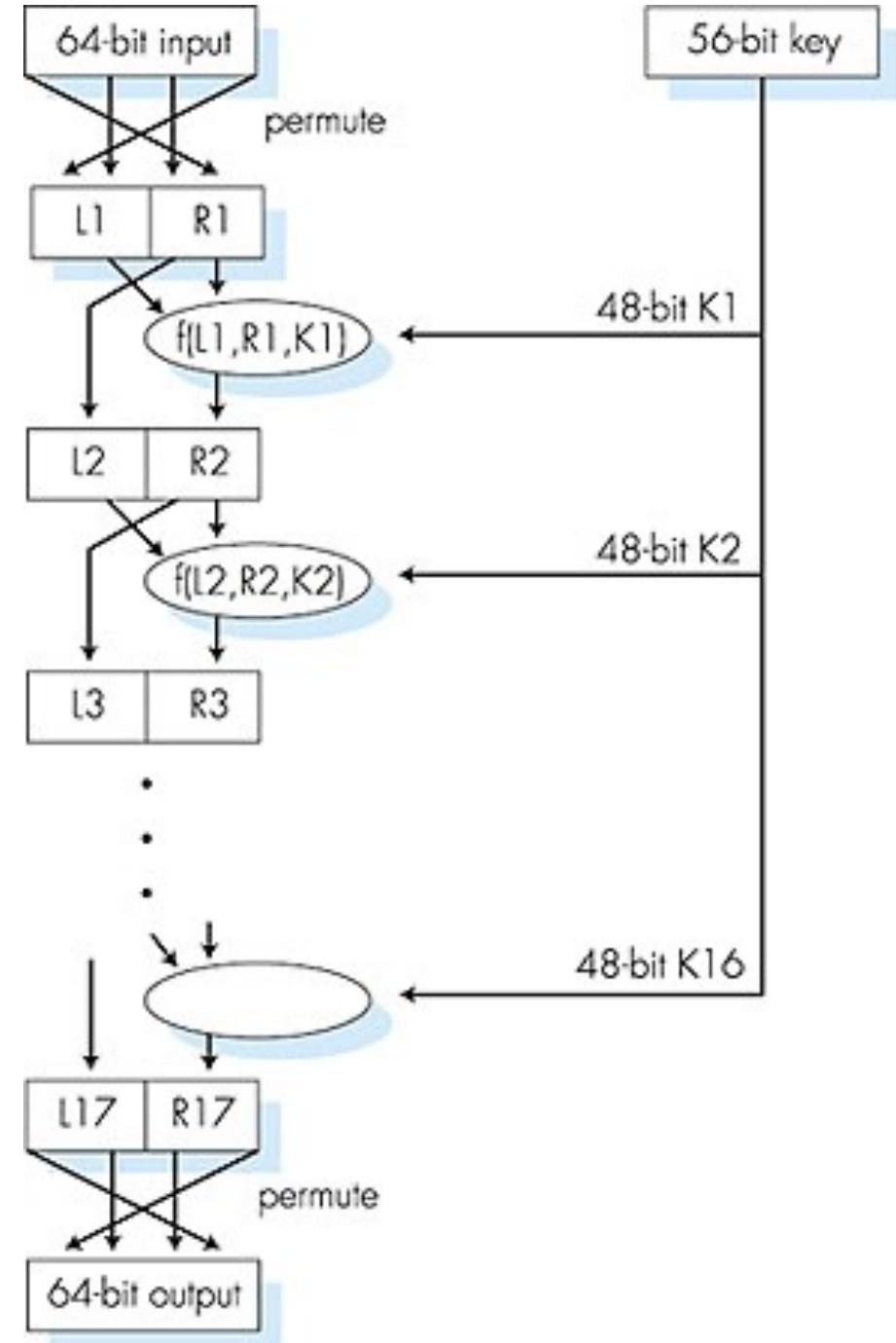
# Symmetric key crypto: DES

## *DES operation*

initial permutation

16 identical “rounds” of function application,  
each using different 48 bits of key

final permutation

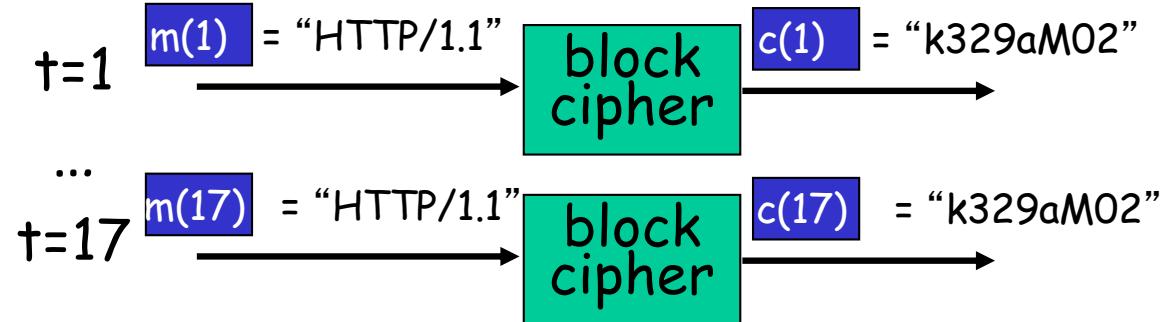


## AES: Advanced Encryption Standard

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

# Cipher Block Chaining

- ❖ cipher block: if input block repeated, will produce same cipher text:



- *Use random numbers:* XOR ith input block,  $m(i)$  and random number  $r(i)$  and apply block-cipher encryption algorithm

- $C(i) = K_s(m(i) \oplus r(i))$
- Send across  $c(i)$  and  $r(i)$

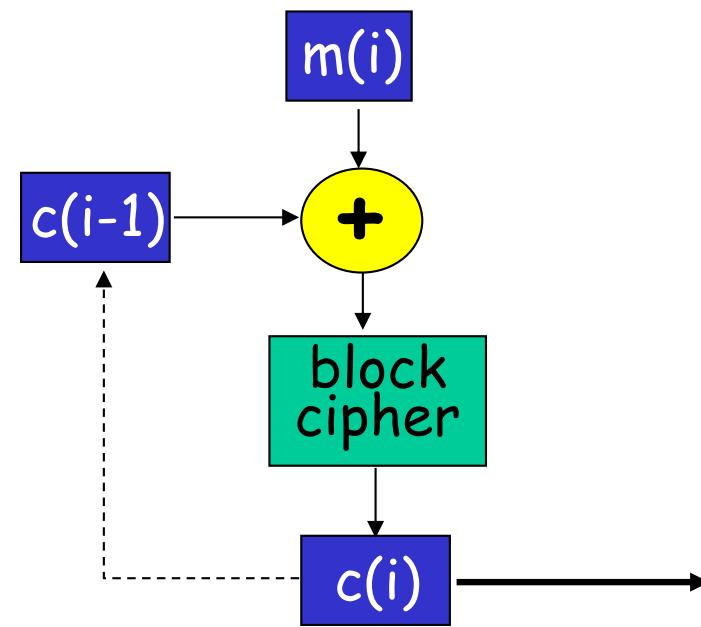
# CBC Example

- ❖ Plaintext: 010 010 010
- ❖ If no CBC, sent txt : 101 101 101
  - 1-to-1 mapping table used
- ❖ Lets use the following random bits
  - r1: 001, r2: 111, r3: 100
  - XoR the plaintext with these random bits
  - 010 XoR 001 = 011
  - Now do table lookup for 011 -> 100
- ❖ We get  $c(1)=100$ ,  $c(2)=010$  and  $c(3)=000$ , although plaintext is the same (010)
- ❖ Need to transmit twice as many bits ( $c(i)$  as well as  $r(i)$ )

Input	Output
000	110
111	001
001	111
010	101
011	100
100	011
101	010
110	000

# Cipher Block Chaining

- *cipher block chaining*: send only one random value alongwith the very first message block, and then have the sender and receiver use the computed cipher block in place of the subsequent random number
- XOR ith input block,  $m(i)$ , with previous block of cipher text,  $c(i-1)$ 
  - $c(0)$  is an initialisation vector (random) transmitted to receiver in clear

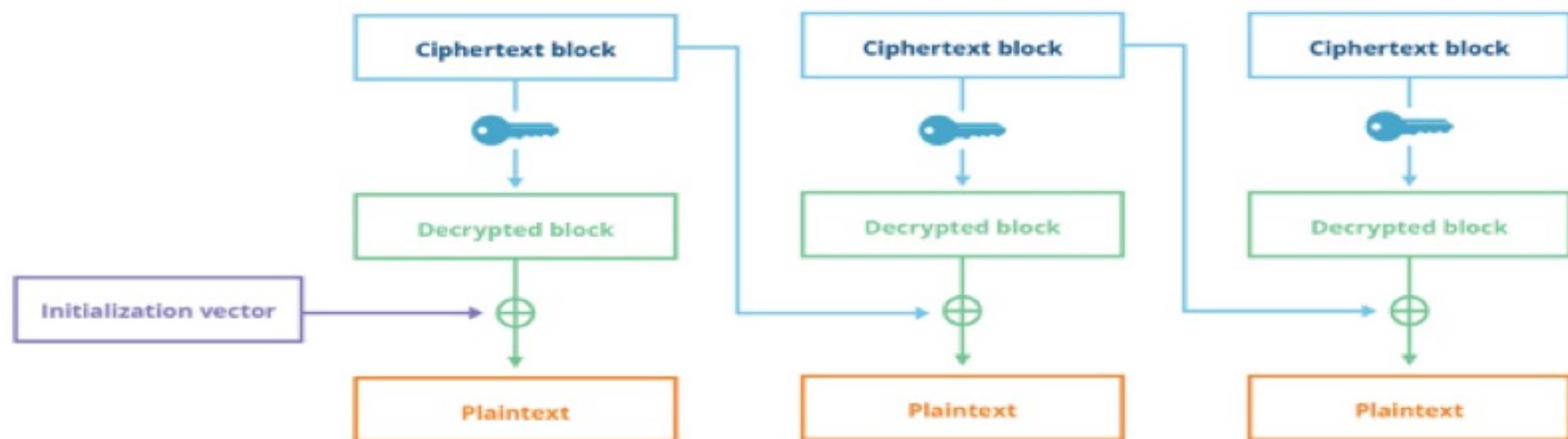
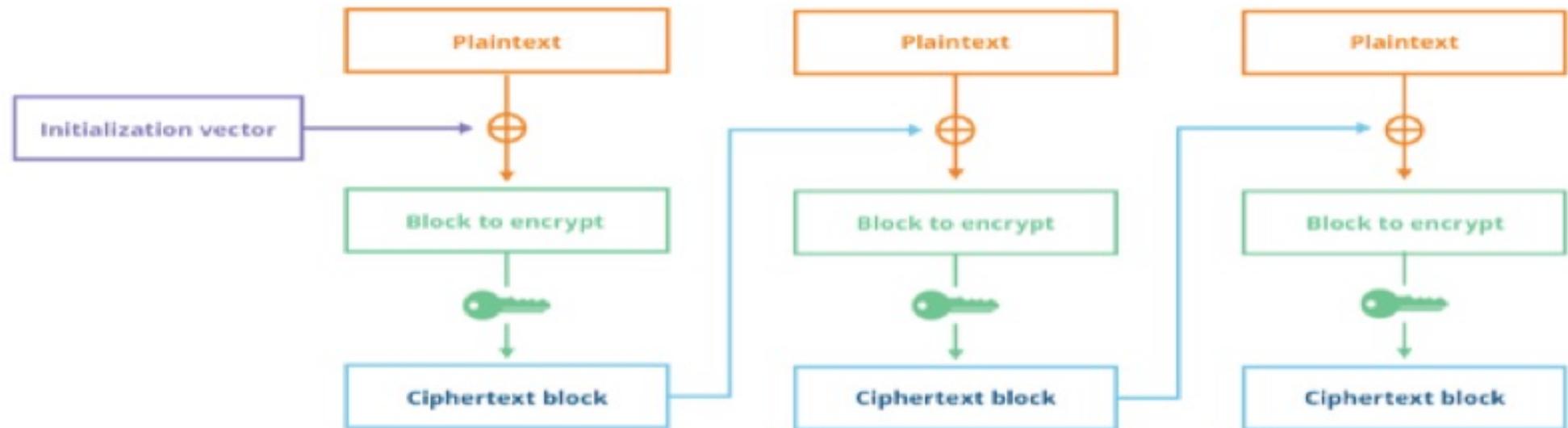


# Cipher Block Chaining (CBC)

---

- ❖ CBC generates its own random numbers
  - Have encryption of current block depend on result of previous block
  - $c(i) = K_S( m(i) \oplus c(i-1) )$
  - $m(i) = K_S( c(i) ) \oplus c(i-1)$
- ❖ How do we encrypt first block?
  - Initialization vector (IV): random block =  $c(0)$
  - IV does not have to be secret
- ❖ Change IV for each message (or session)
  - Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time

# Cipher Block Chaining (CBC)



# Public Key Cryptography

## *symmetric key crypto*

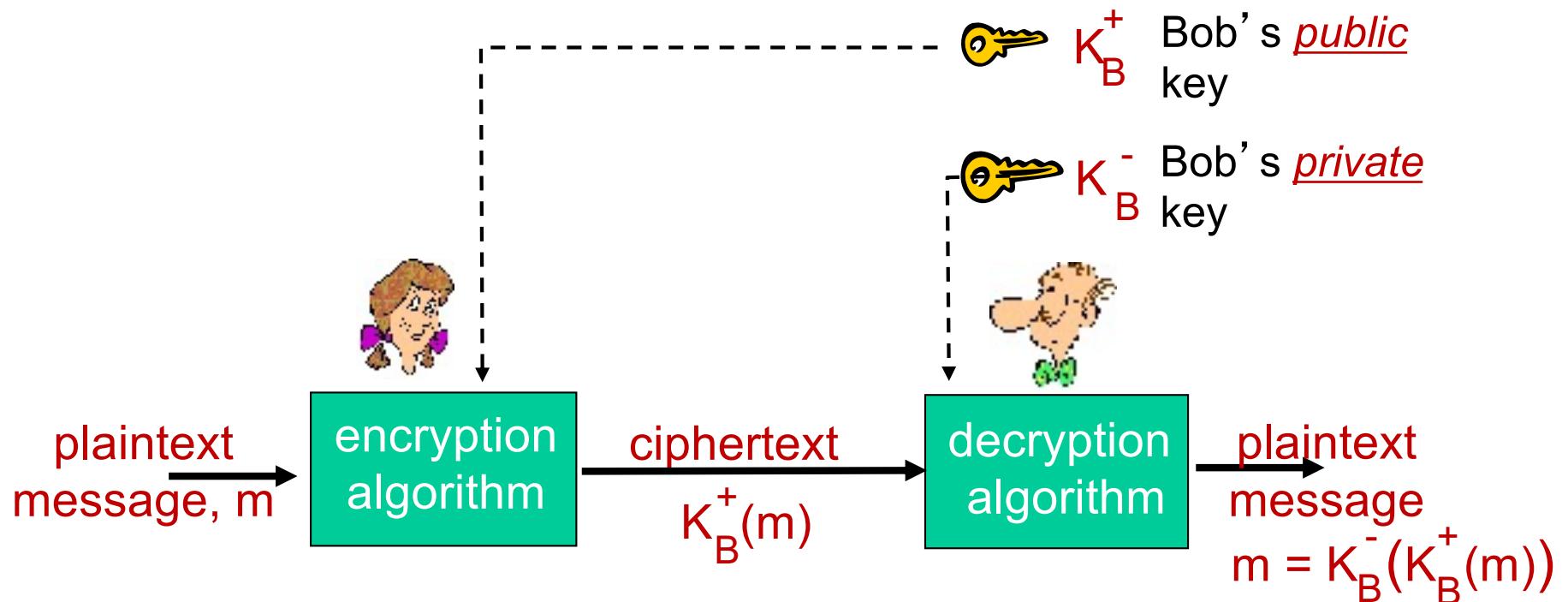
- ❖ requires sender, receiver know shared secret key
- ❖ Q: how to agree on key in first place (particularly if never “met”)?

## *public key crypto*

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



# Public key cryptography



# Public key encryption algorithms

requirements:

- ① need  $K_B^+(\cdot)$  and  $K_B^-(\cdot)$  such that  
$$K_B^-(K_B^+(m)) = m$$
- ② given public key  $K_B^+$ , it should be impossible to compute private key  $K_B^-$

**RSA:** Rivest, Shamir, Adelson algorithm

# Prerequisite: modular arithmetic

- ❖  $x \bmod n = \text{remainder of } x \text{ when divide by } n$
- ❖ facts:
  - $[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$
  - $[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$
  - $[(a \bmod n) * (b \bmod n)] \bmod n = (a*b) \bmod n$
- ❖ thus
  - $(a \bmod n)^d \bmod n = a^d \bmod n$
- ❖ example:  $x=14, n=10, d=2:$   
 $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$   
 $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

Note: You don't need to know this for the exam

# RSA: getting ready

- ❖ message: just a bit pattern
- ❖ bit pattern can be uniquely represented by an integer number
- ❖ thus, encrypting a message is equivalent to encrypting a number.

*example:*

- ❖  $m = 10010001$ . This message is uniquely represented by the decimal number 145.
- ❖ to encrypt  $m$ , we encrypt the corresponding number, which gives a new number (the ciphertext).

## RSA: Creating public/private key pair

1. choose two large prime numbers  $p, q$ .  
(e.g., 1024 bits each)
2. compute  $n = pq$ ,  $z = (p-1)(q-1)$
3. choose  $e$  (with  $e < n$ ) that has no common factors with  $z$  ( $e, z$  are “relatively prime”).
4. choose  $d$  such that  $ed - 1$  is exactly divisible by  $z$ .  
(in other words:  $ed \bmod z = 1$  ).
5. public key is  $\underbrace{(n,e)}_{K_B^+}$ . private key is  $\underbrace{(n,d)}_{K_B^-}$ .

# RSA: encryption, decryption

0. given  $(n,e)$  and  $(n,d)$  as computed above

I. to encrypt message  $m (< n)$ , compute

$$c = m^e \bmod n$$

2. to decrypt received bit pattern,  $c$ , compute

$$m = c^d \bmod n$$

*magic happens!*  $m = \underbrace{(m^e \bmod n)^d}_{c} \bmod n$

Proof of Correctness: Fermat's Little Theorem or Euler's Theorem (not on exam)

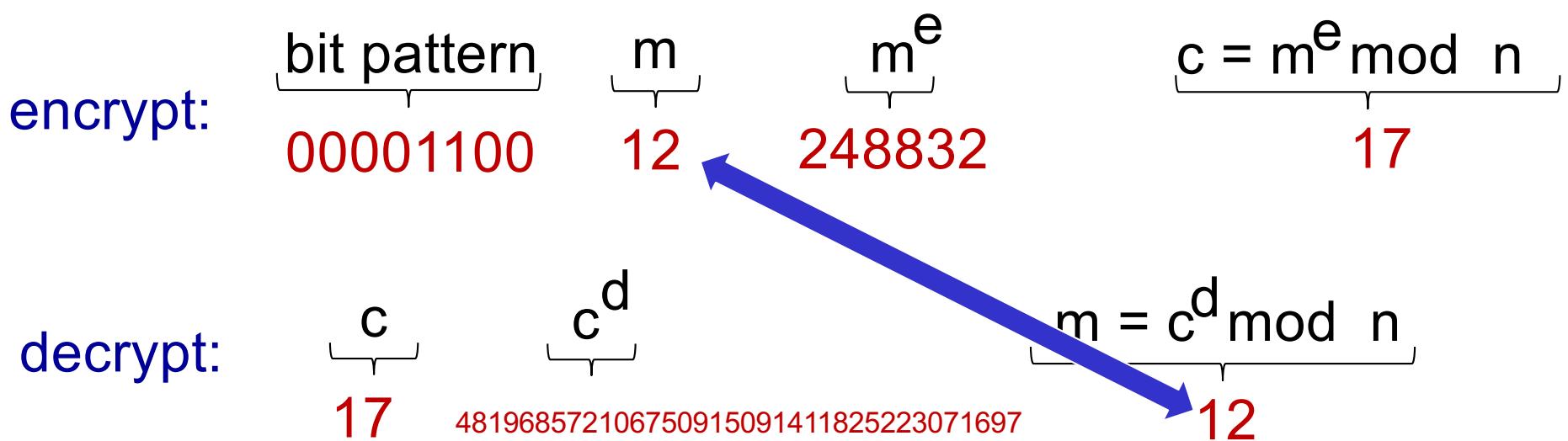
# RSA example:

Bob chooses  $p=5$ ,  $q=7$ . Then  $n=35$ ,  $z=24$ .

$e=5$  (so  $e$ ,  $z$  relatively prime).

$d=29$  (so  $ed-1$  exactly divisible by  $z$ ).

encrypting 8-bit messages.



# RSA: another important property

The following property will be *very* useful later:

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

use public key first,  
followed by  
private key

use private key  
first, followed by  
public key

*result is the same!*

# Why is RSA secure?

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

# RSA in practice: session keys

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

*session key,  $K_s$*

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

# Network Security: roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

# Authentication

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

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



Failure scenario??

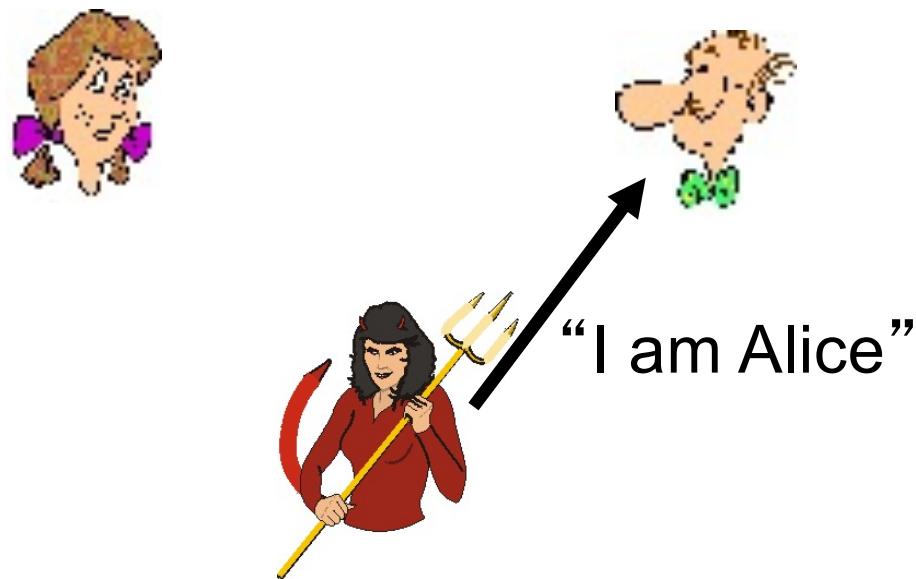


Note: In some applications, both end points may need to authenticate each other

# Authentication

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

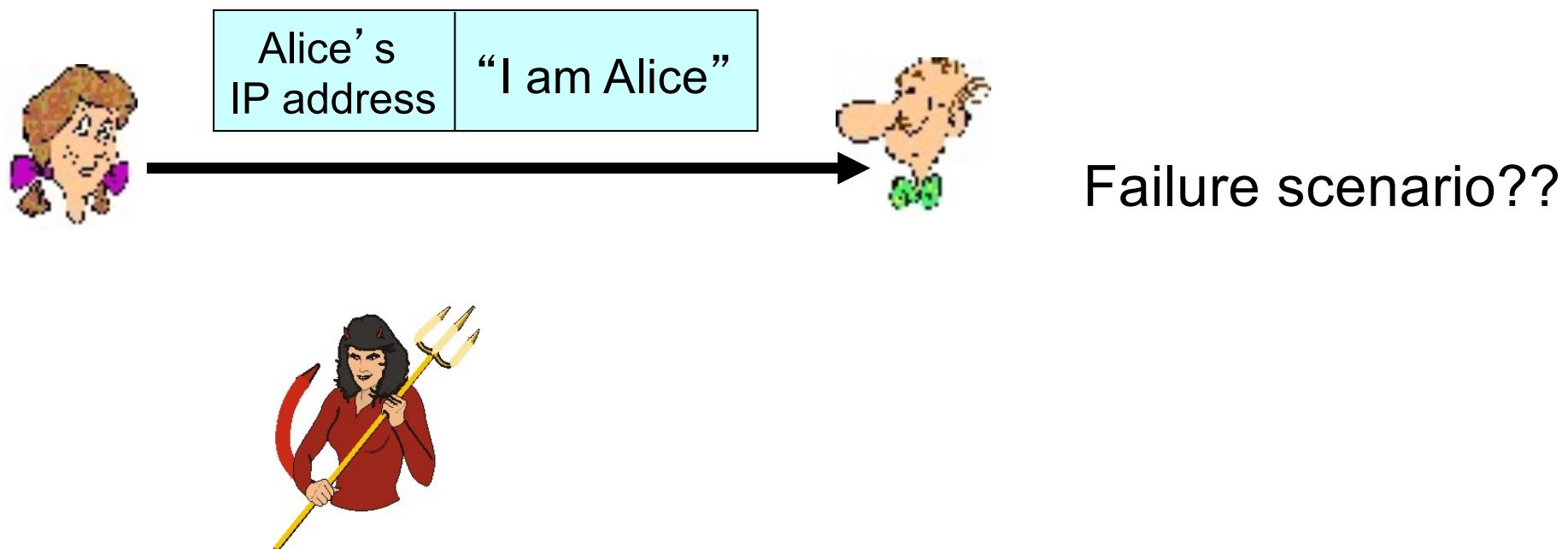
**Protocol 1.0:** Alice says “I am Alice”



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

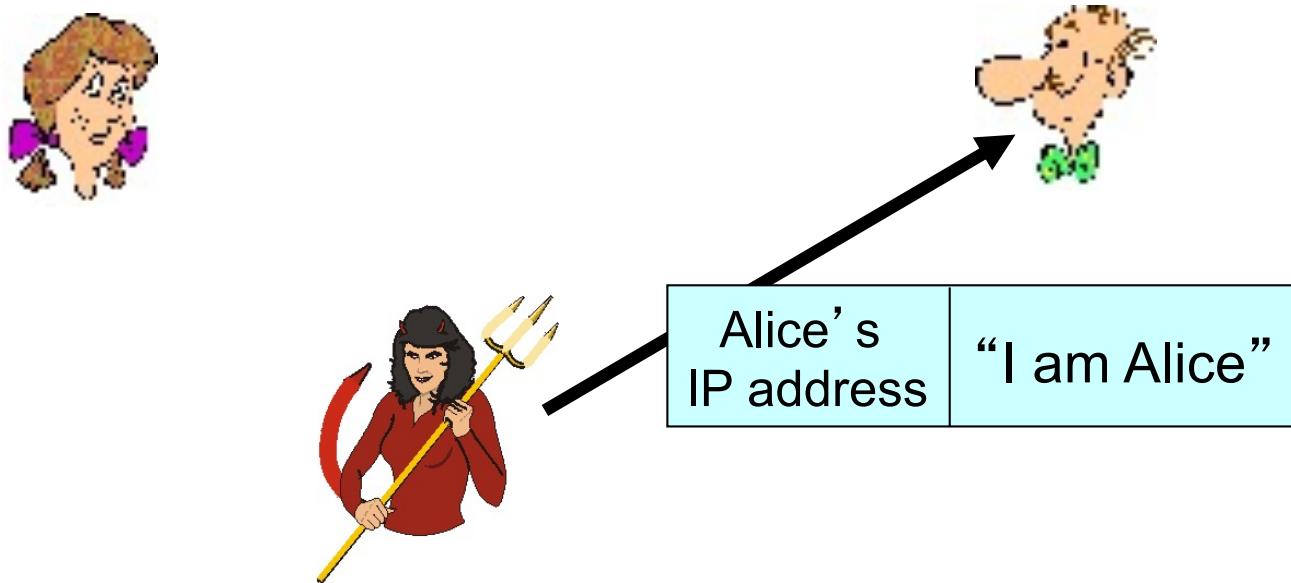
# Authentication: another try

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



# Authentication: another try

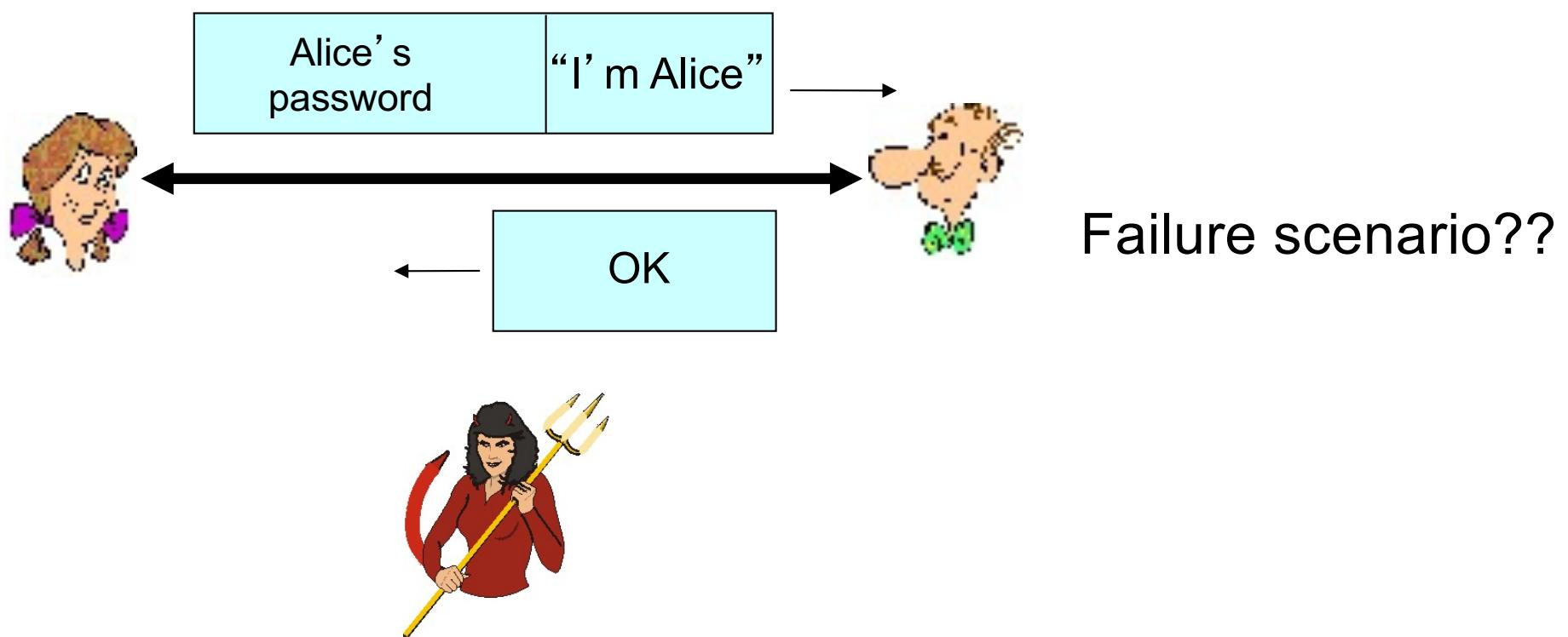
*Protocol ap2.0:* Alice says “I am Alice” in an IP packet containing her source IP address



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

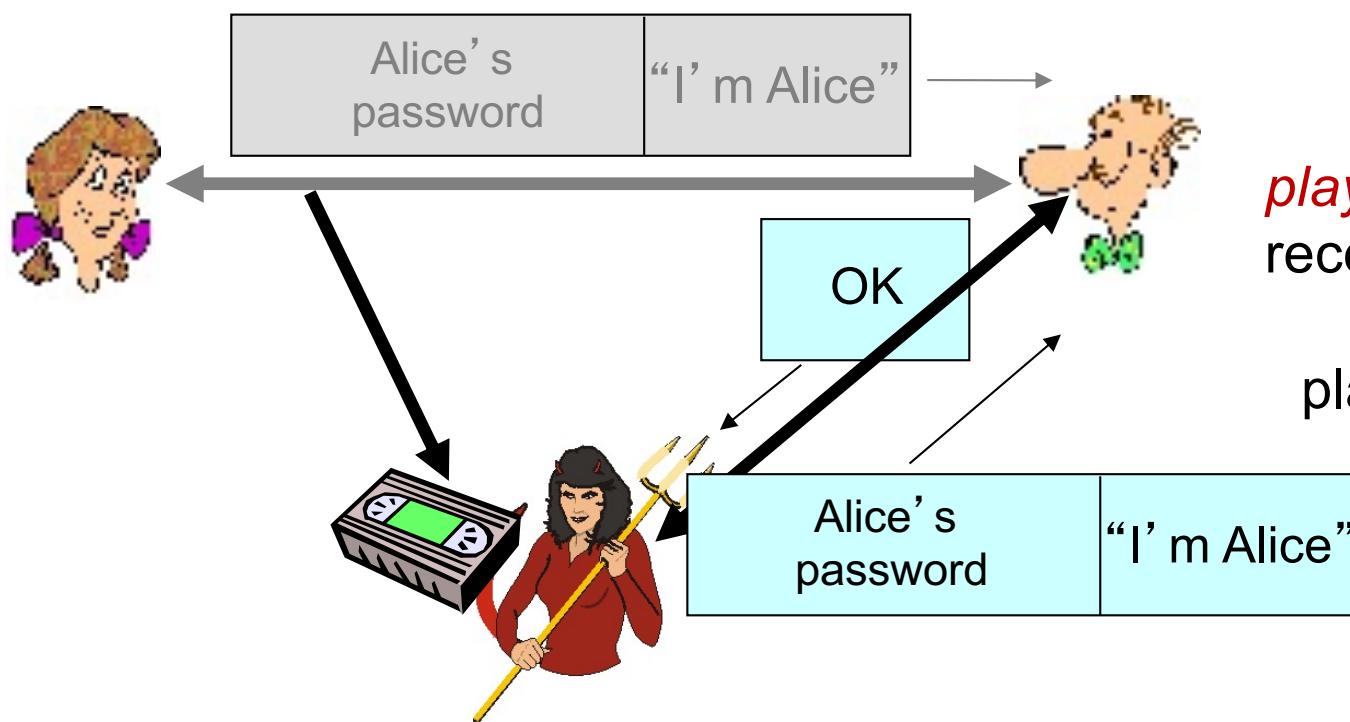
# Authentication: another try

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



# Authentication: another try

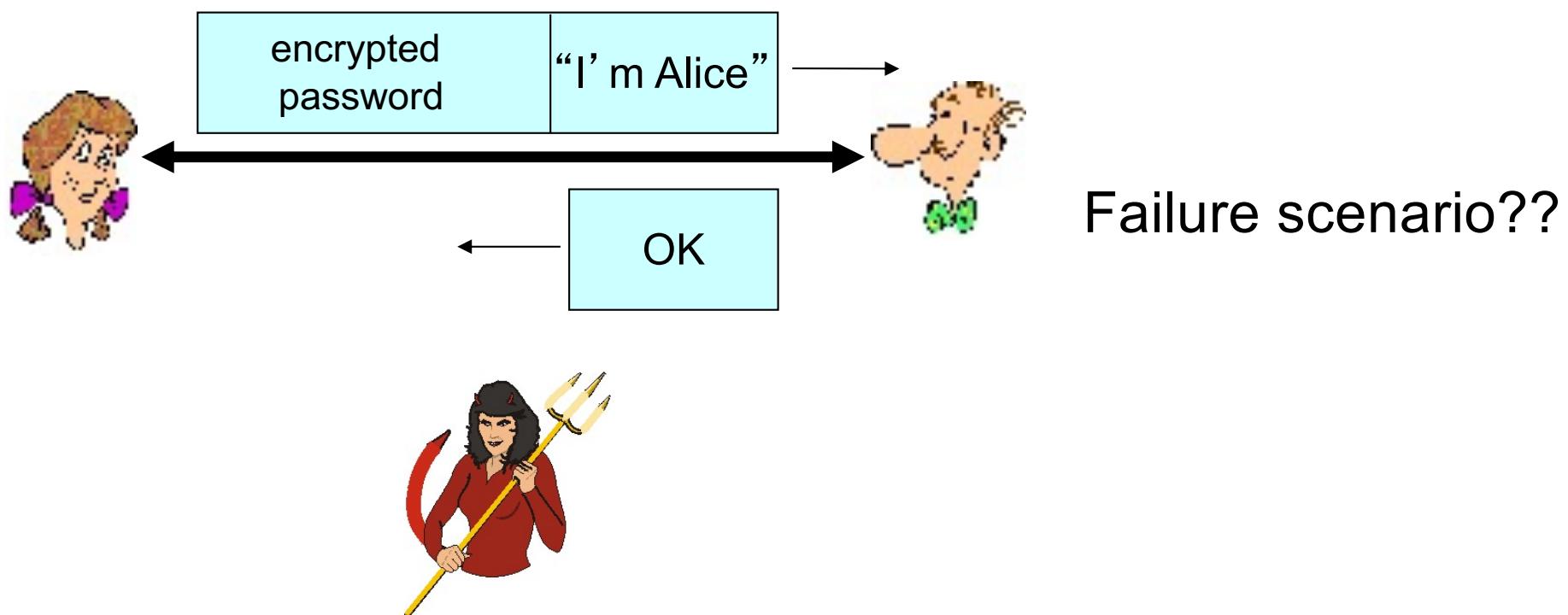
*Protocol ap3.0:* Alice says “I am Alice” and sends her secret password to “prove” it.



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

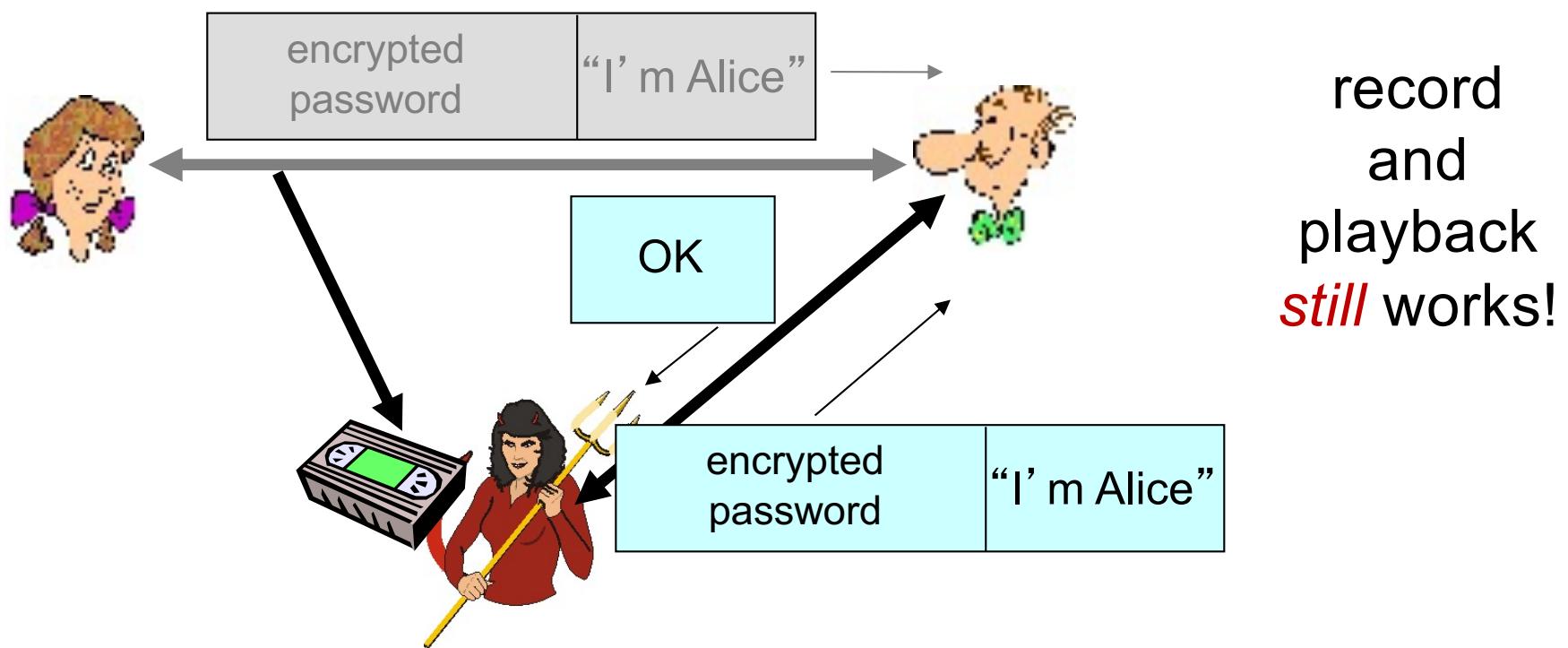
# Authentication: yet another try

*Protocol ap3.1:* Alice says “I am Alice” and sends her **encrypted** secret password to “prove” it.



# Authentication: yet another try

*Protocol ap3.1:* Alice says “I am Alice” and sends her **encrypted** secret password to “prove” it.

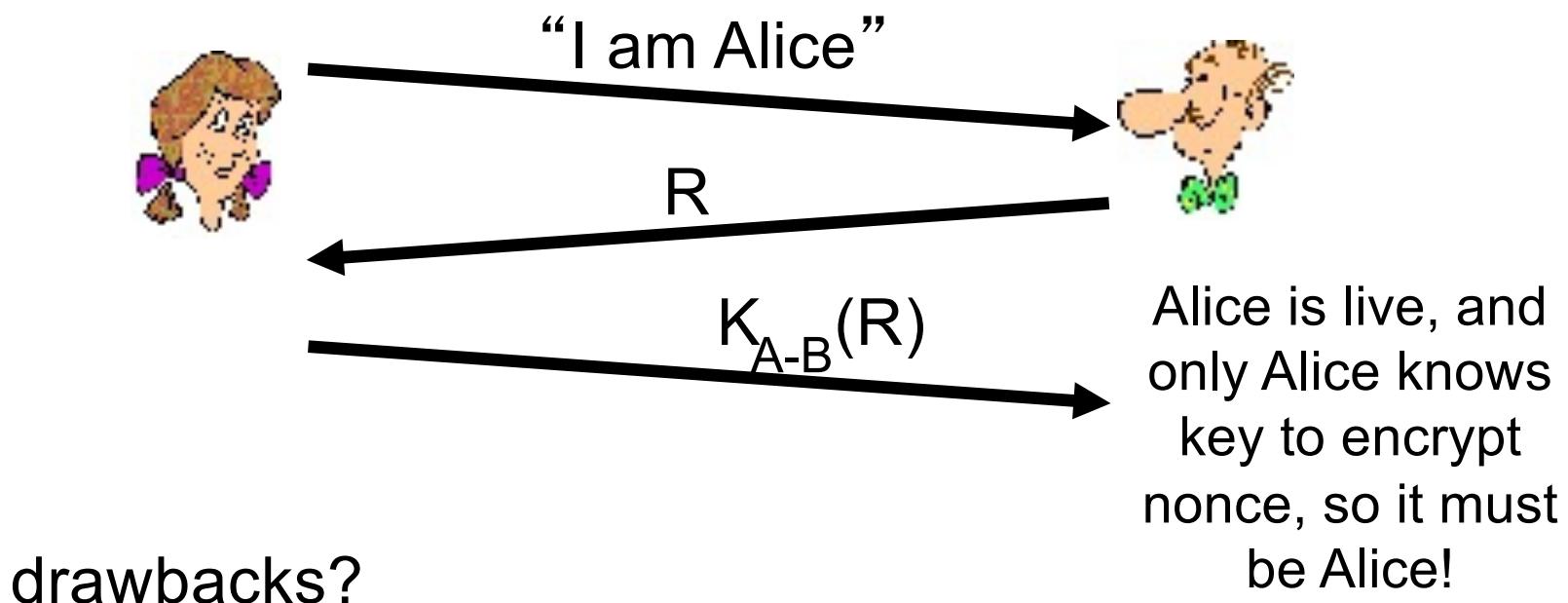


# Authentication: yet another try

**Goal:** avoid playback attack

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

**ap4.0:** to prove Alice “live”, Bob sends Alice **nonce**, R. Alice must return R, encrypted with shared secret key



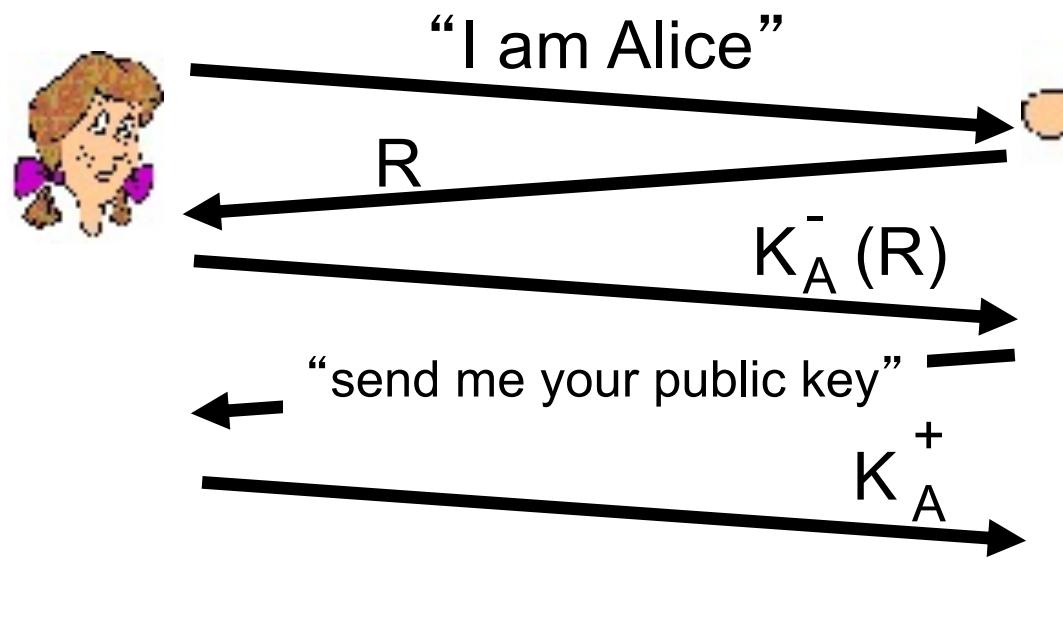
drawbacks?

# Authentication: ap5.0

ap4.0 requires shared symmetric key

- ❖ can we authenticate using public key techniques?

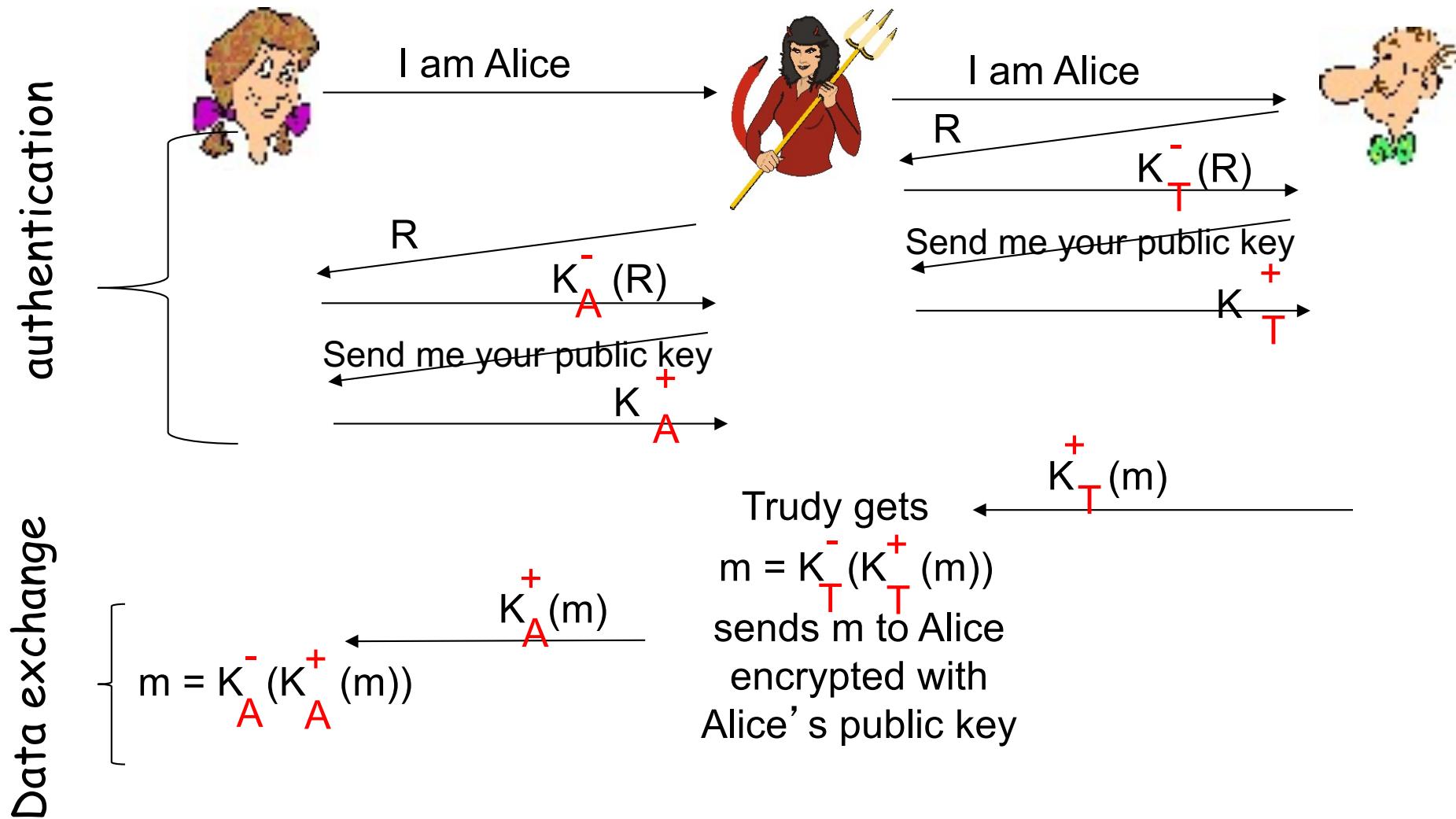
*ap5.0:* use nonce, public key cryptography



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

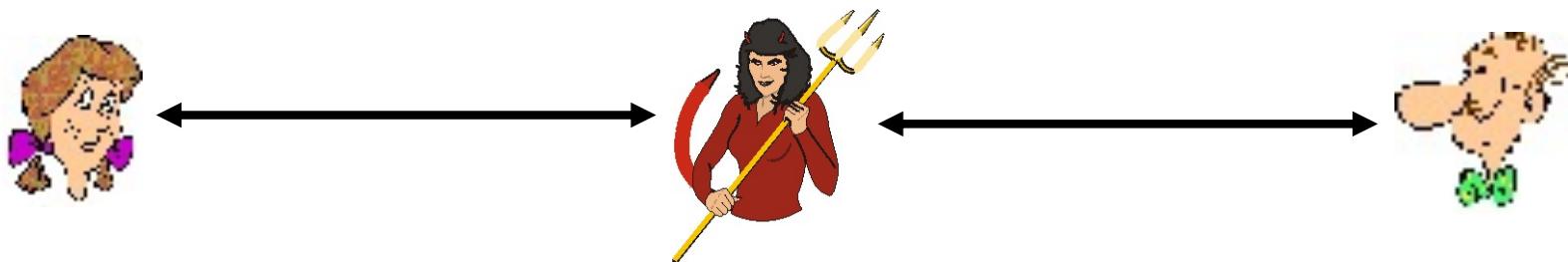
# ap5.0: security hole

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



## ap5.0: security hole

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



difficult to detect:

- ❖ Bob receives everything that Alice sends, and vice versa.  
(e.g., so Bob, Alice can meet one week later and recall conversation!)
- ❖ problem is that Trudy receives all messages as well!

# Network Security: roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

8.4 Authentication

8.5 Securing email

# Confidentiality vs Integrity

- ❖ Confidentiality: message private and secret
- ❖ Integrity: protection against message tampering
- ❖ Encryption alone may not guarantee integrity
  - Attacker can modify message under encryption without learning what it is
- ❖ Public Key Crypto Standard (PKCS)
  - “RSA encryption is intended primarily to provide confidentiality .... It is not intended to provide integrity”
- ❖ Both confidentiality and integrity are needed for security

# Digital signatures

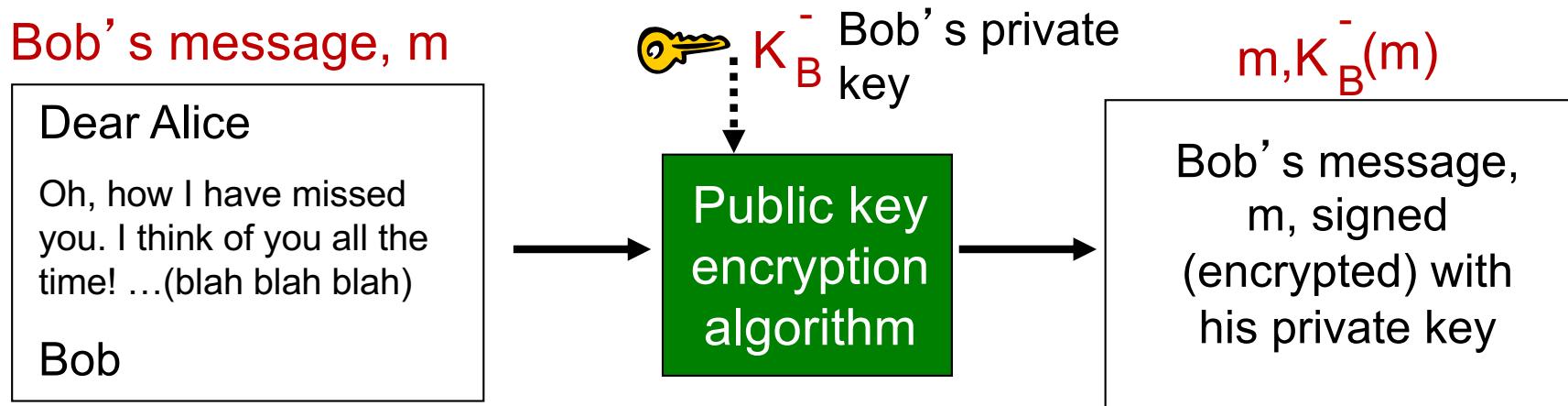
cryptographic technique analogous to hand-written signatures:

- ❖ sender (Bob) digitally signs document, establishing he is document owner/creator.
- ❖ *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

# Digital signatures

simple digital signature for message  $m$ :

- ❖ Bob signs  $m$  by encrypting with his private key  $K_B^-$ , creating “signed” message,  $K_B^-(m)$



# Digital signatures

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

Alice thus verifies that:

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

non-repudiation:

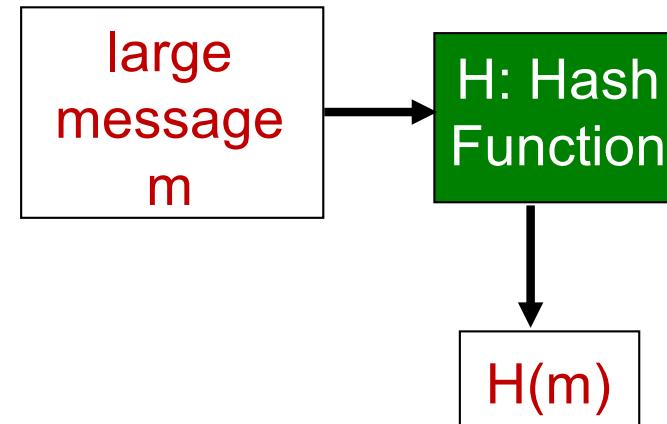
- ✓ Alice can take  $m$ , and signature  $K_B^-(m)$  to court and prove that Bob signed  $m$

# Message digests

computationally expensive to public-key-encrypt long messages

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

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



**Hash function properties:**

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

# Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- ✓ produces fixed length digest (16-bit sum) of message
- ✓ is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

<u>message</u>	<u>ASCII format</u>
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42

**B2 C1 D2 AC**

different messages  
but identical checksums!

<u>message</u>	<u>ASCII format</u>
I O U <b>9</b>	49 4F 55 <b>39</b>
0 0 . <b>1</b>	30 30 2E <b>31</b>
9 B O B	39 42 D2 42

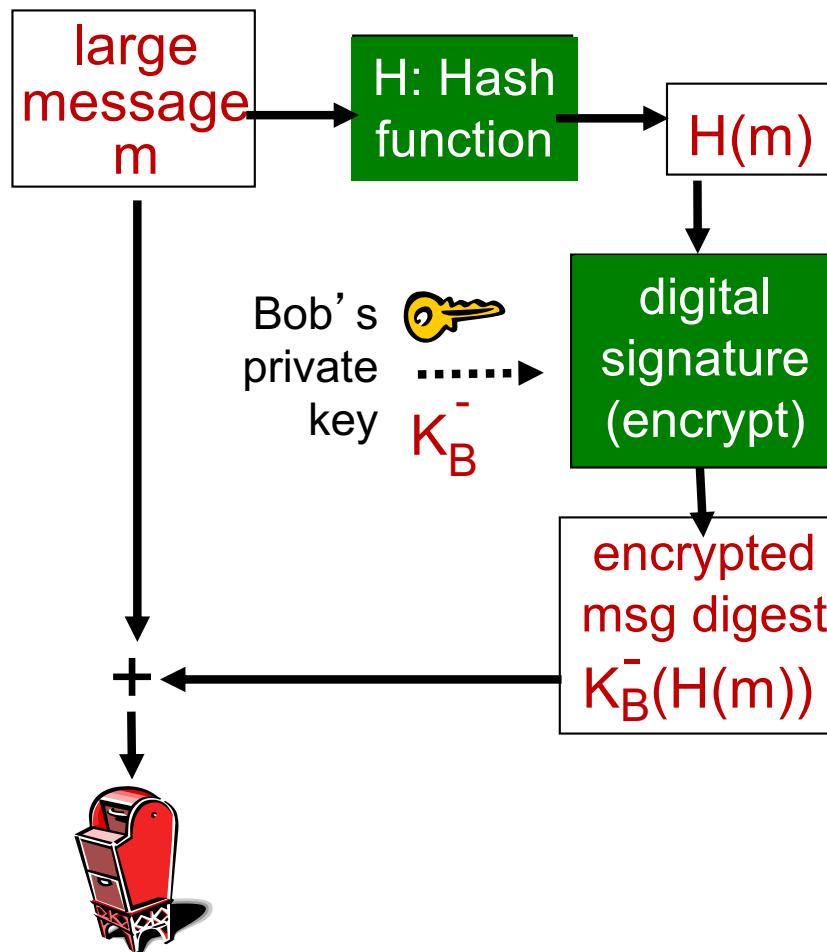
**B2 C1 D2 AC**

# Hash function algorithms

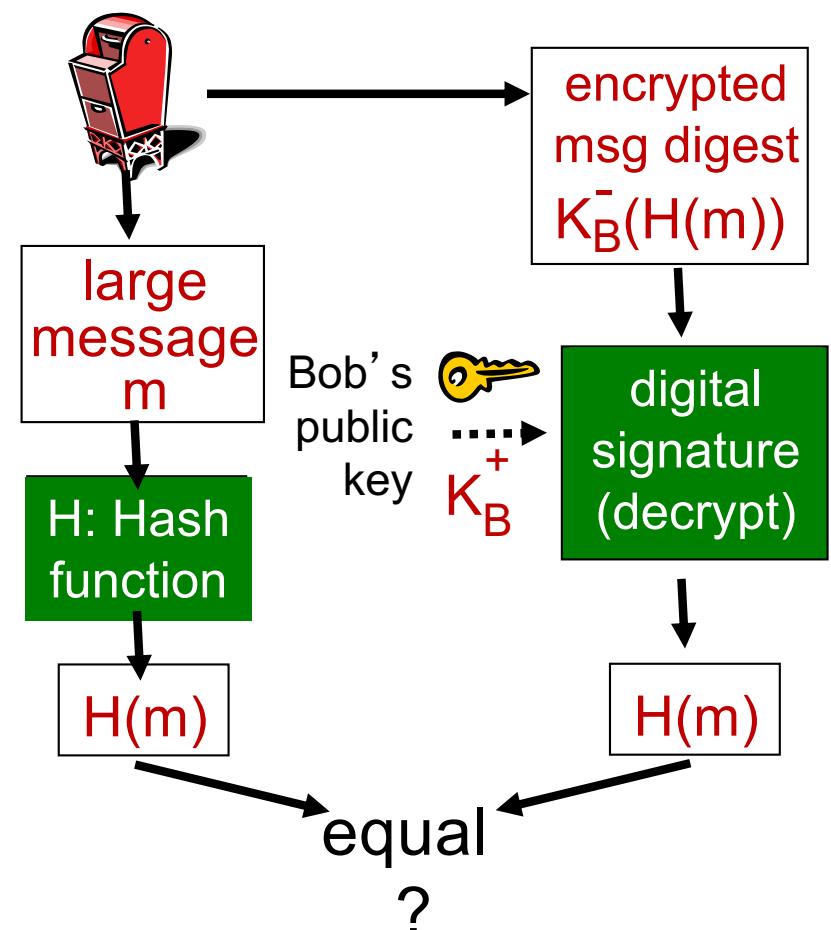
- ❖ MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string  $x$ , appears difficult to construct msg  $m$  whose MD5 hash is equal to  $x$
- ❖ SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
- ❖ SHA-2 and SHA-3 (recent standard) are better - security

# Digital signature = signed message digest

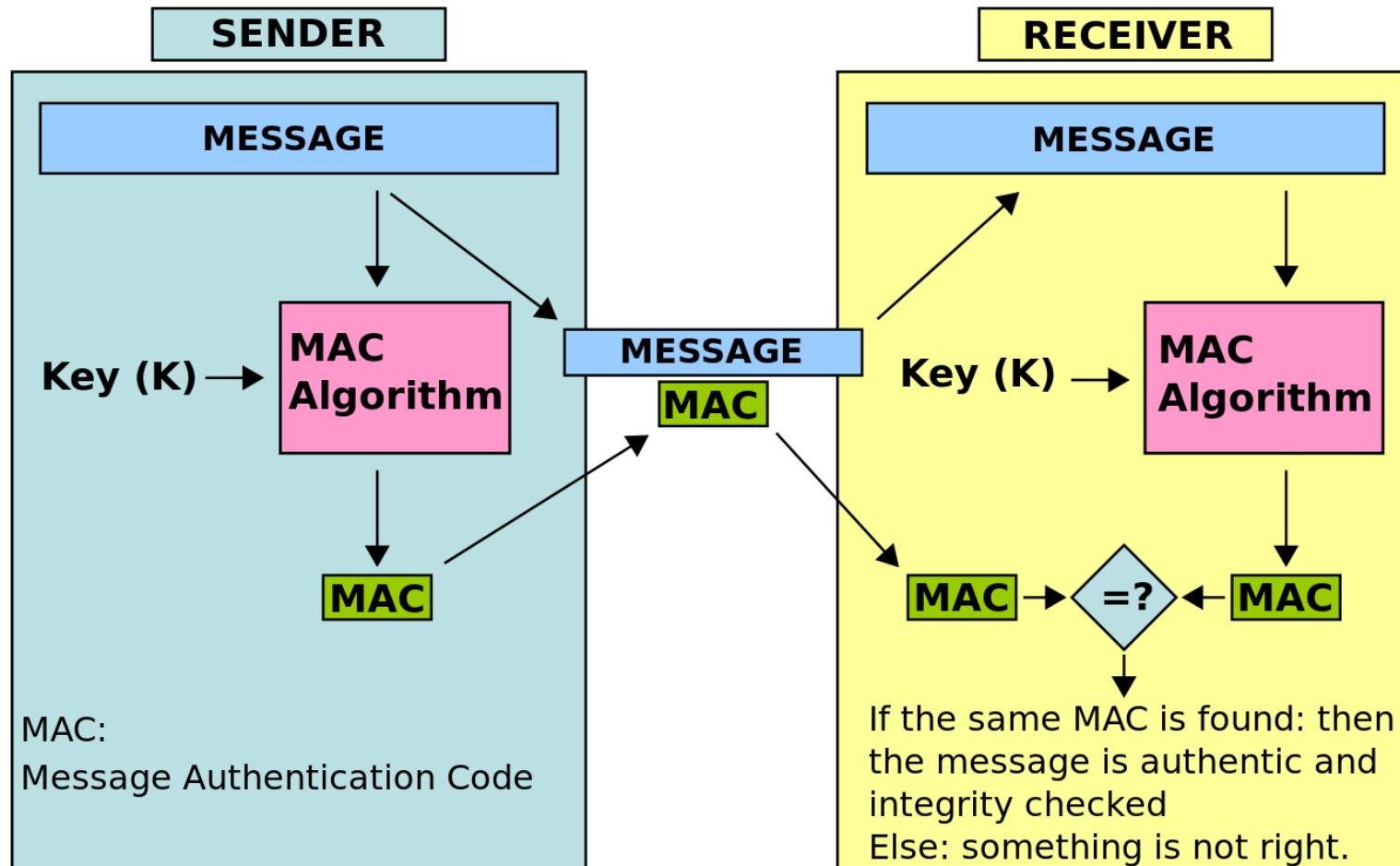
Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:



# Message Authentication Code (MAC)



**Digital signatures use asymmetric key crypto**

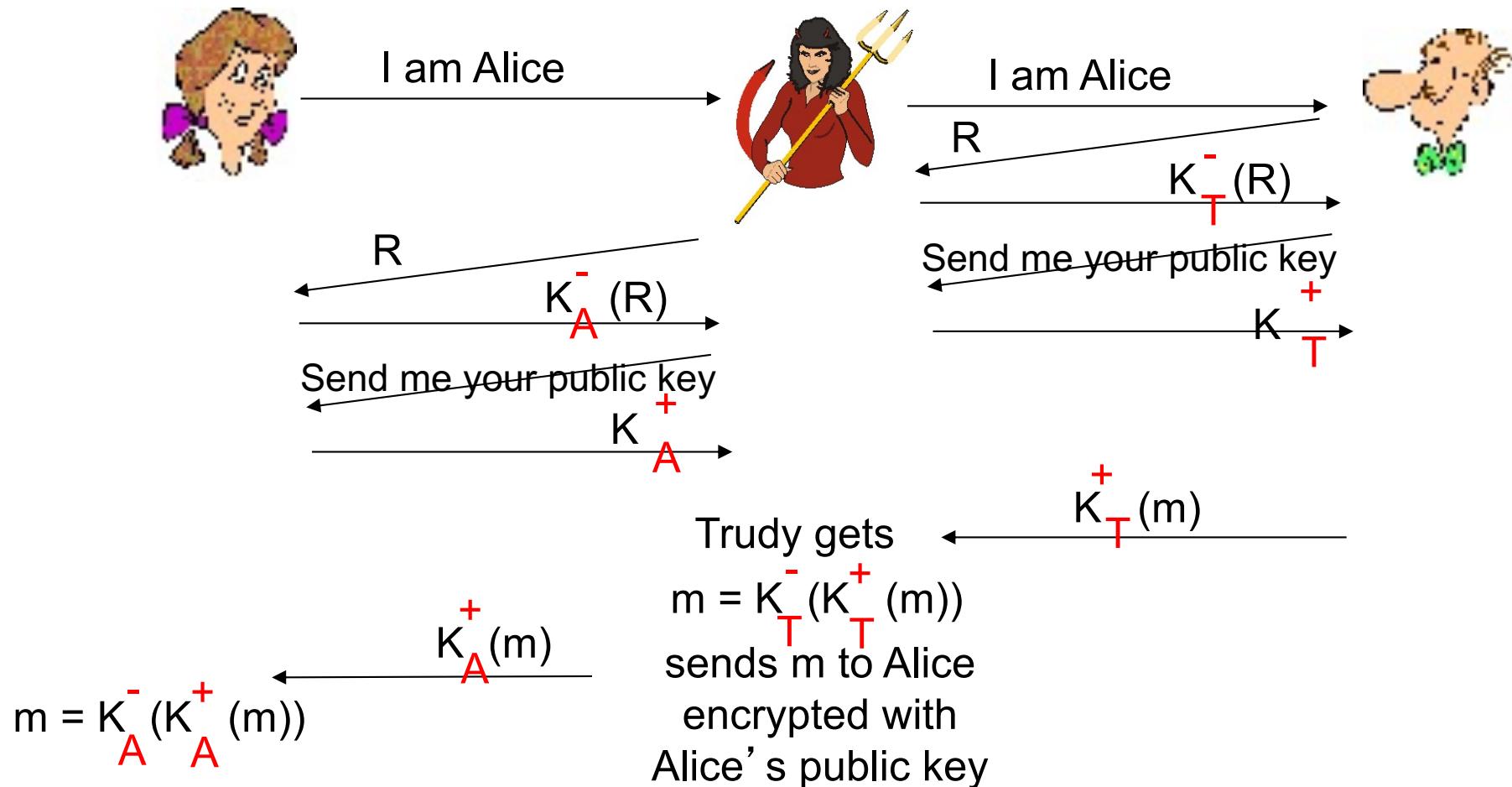
**MAC allows a way to sign a message but using symmetric key**

**Requires a shared secret key**

**Examples: UMAC-VMAC, SipHash, Poly1305-AES**

# Recall: ap5.0 security hole

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



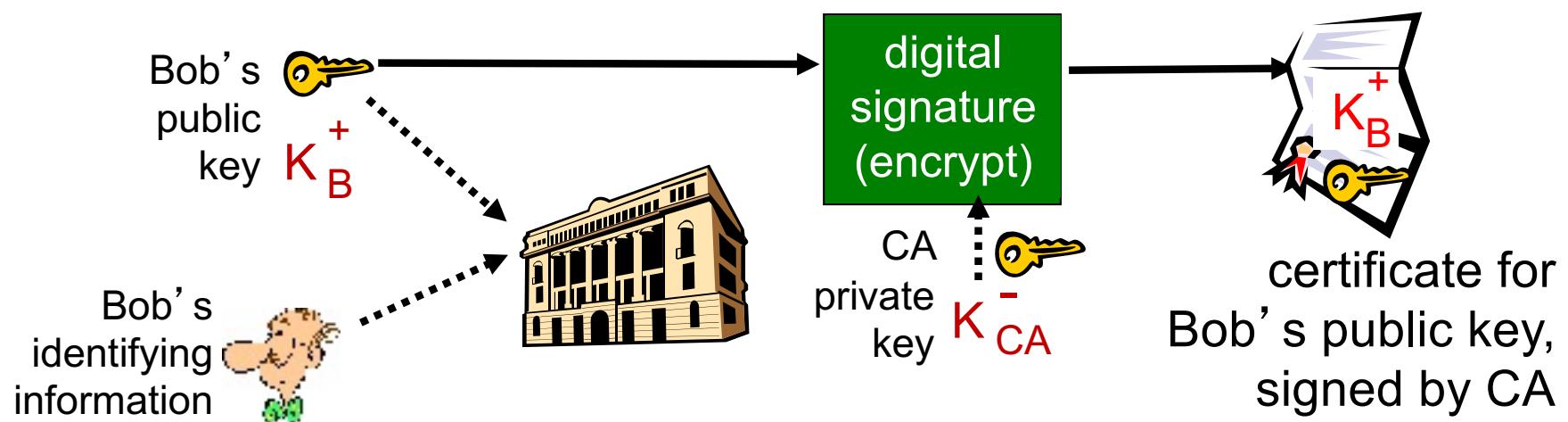
# Public-key certification

- ❖ motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:  
*Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob*
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni



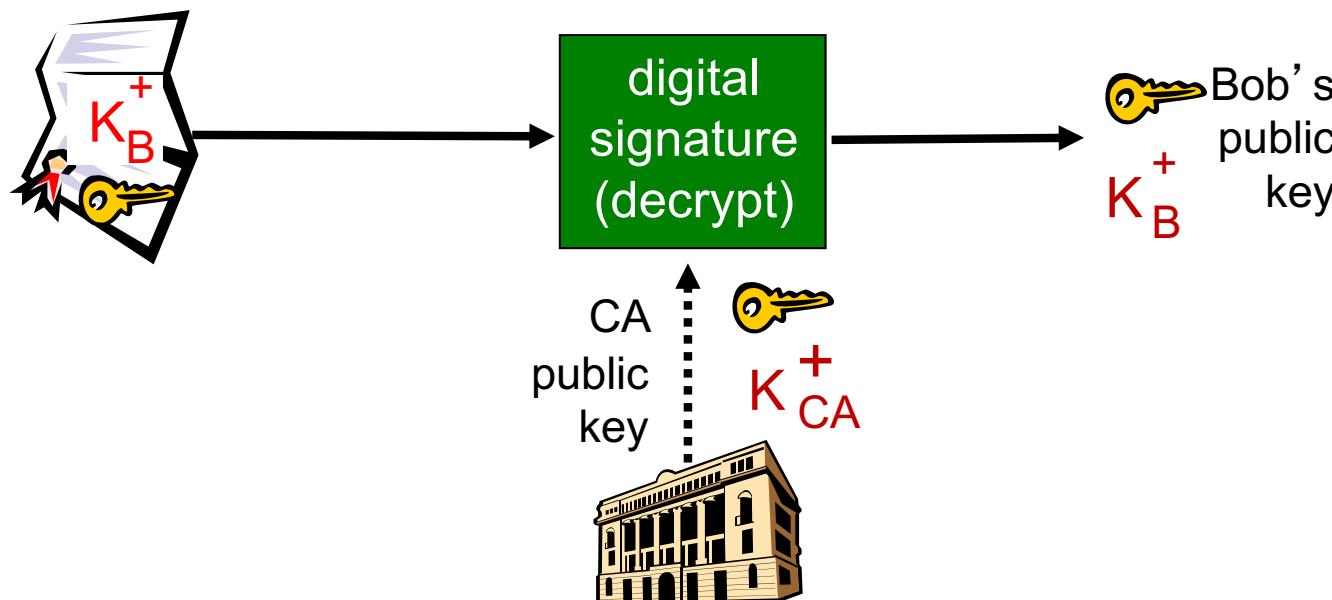
# Certification authorities

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



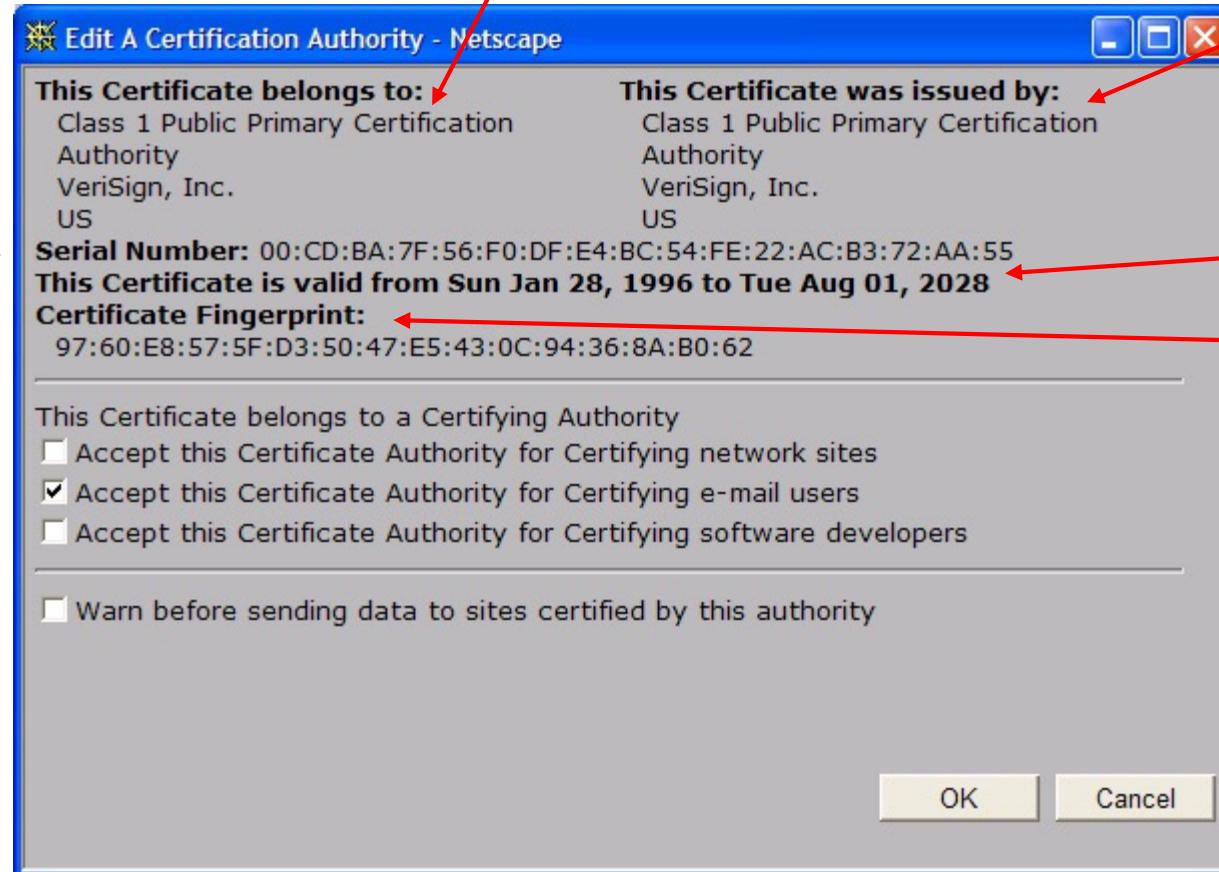
# Certification authorities

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



# A certificate contains:

- ❖ Serial number (unique to issuer)
- ❖ info about certificate owner, including algorithm and key value itself  
(not shown)



- info about certificate issuer
- valid dates
- digital signature by issuer

# Certificates: summary

---

- ❖ Primary standard X.509 (RFC 2459)
- ❖ Certificate contains:
  - Issuer name
  - Entity name, address, domain name, etc.
  - Entity's public key
  - Digital signature (signed with issuer's private key)
- ❖ Public-Key Infrastructure (PKI)
  - Certificates and certification authorities
  - Often considered “heavy”

# Quiz



- ❖ Suppose Bob wants to send Alice a digital signature for the message  $m$ . To create the digital signature
  - a) Bob applies a hash function to  $m$  and encrypts the result with his private key
  - b) Bob applies a hash function to  $m$  and encrypts the result with Alice's public key
  - c) Bob encrypts  $m$  with his private key and then applies a hash function to the result
  - d) Bob applies a hash function to  $m$  and encrypts the result with his public key

Answer: A

# Quiz



❖ Suppose a CA creates Bob's certificate, which binds Bob's public key to Bob. This certificate is signed with

Answer: C

- a) Bob's private key
- b) Bob's public key
- c) The CA's private key
- d) The CA's public key
- e) Donald Trump's key



# Network Security: roadmap

8.1 What is network security?

8.2 Principles of cryptography

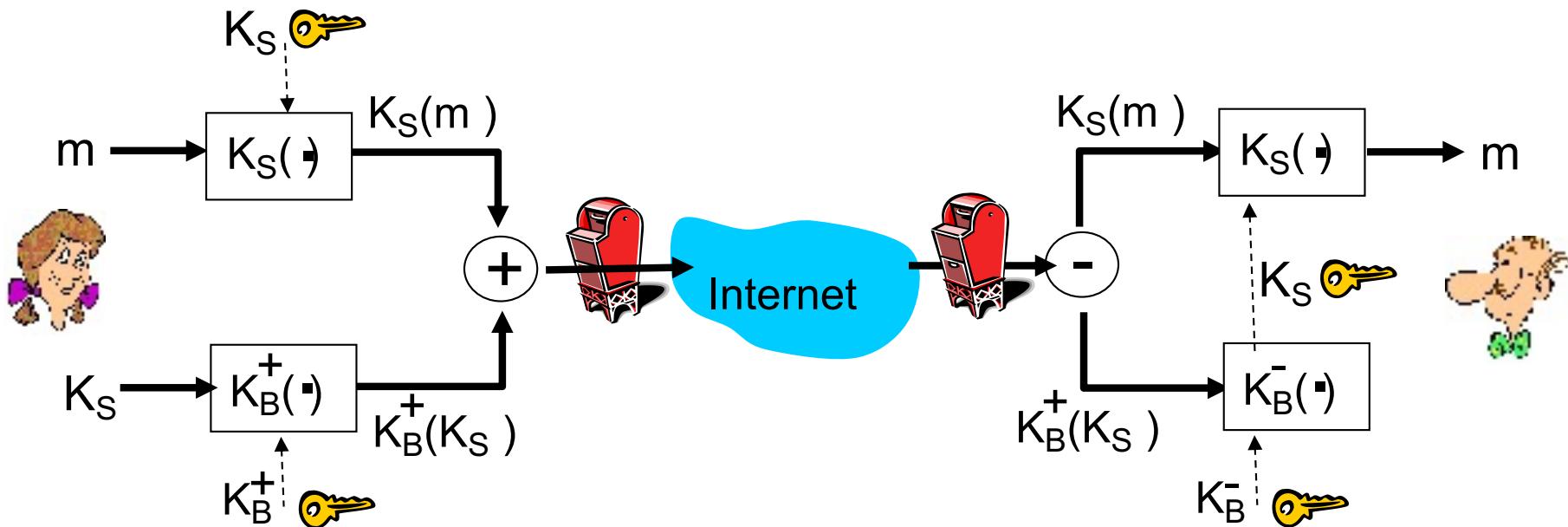
8.3 Message integrity

8.4 Authentication

8.5 Securing e-mail

# Secure e-mail

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

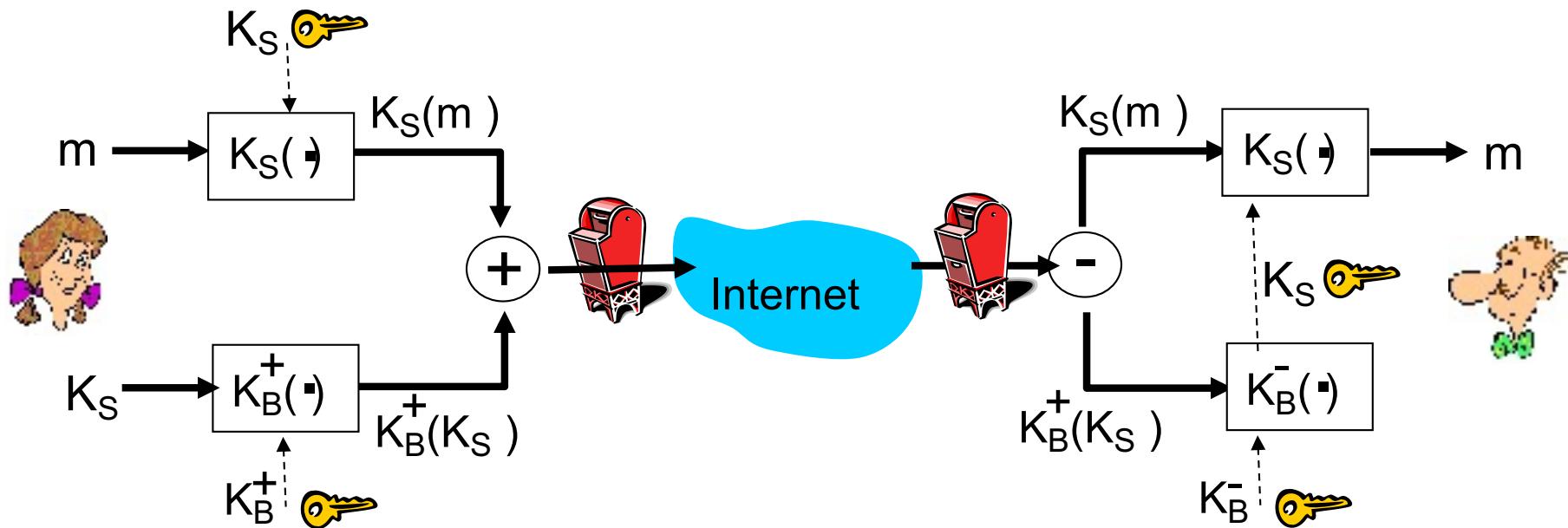


*Alice:*

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

# Secure e-mail

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

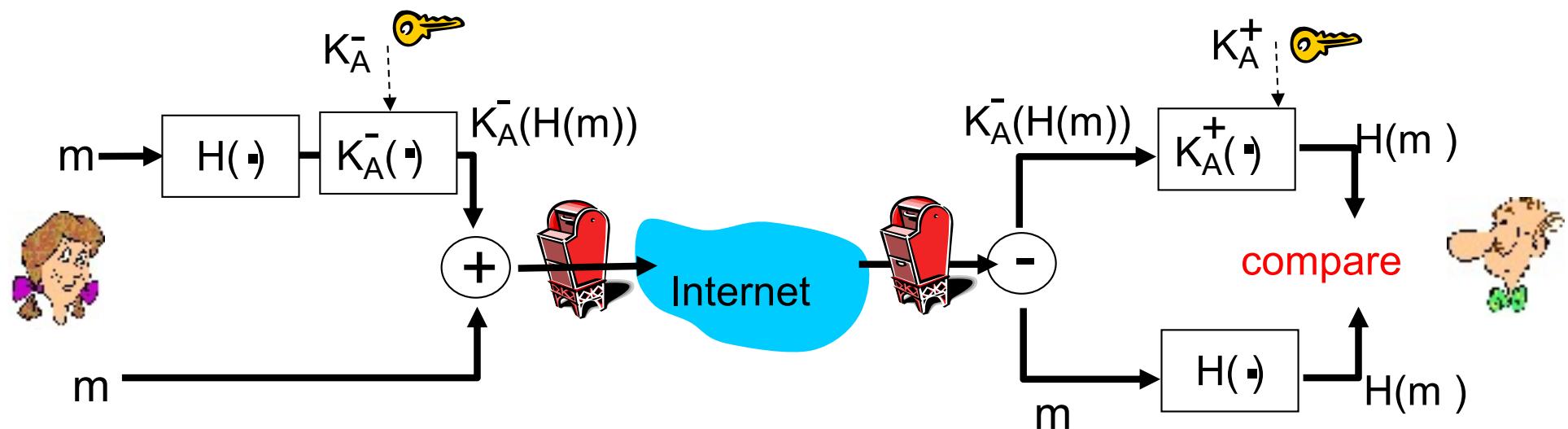


*Bob:*

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

# Secure e-mail (continued)

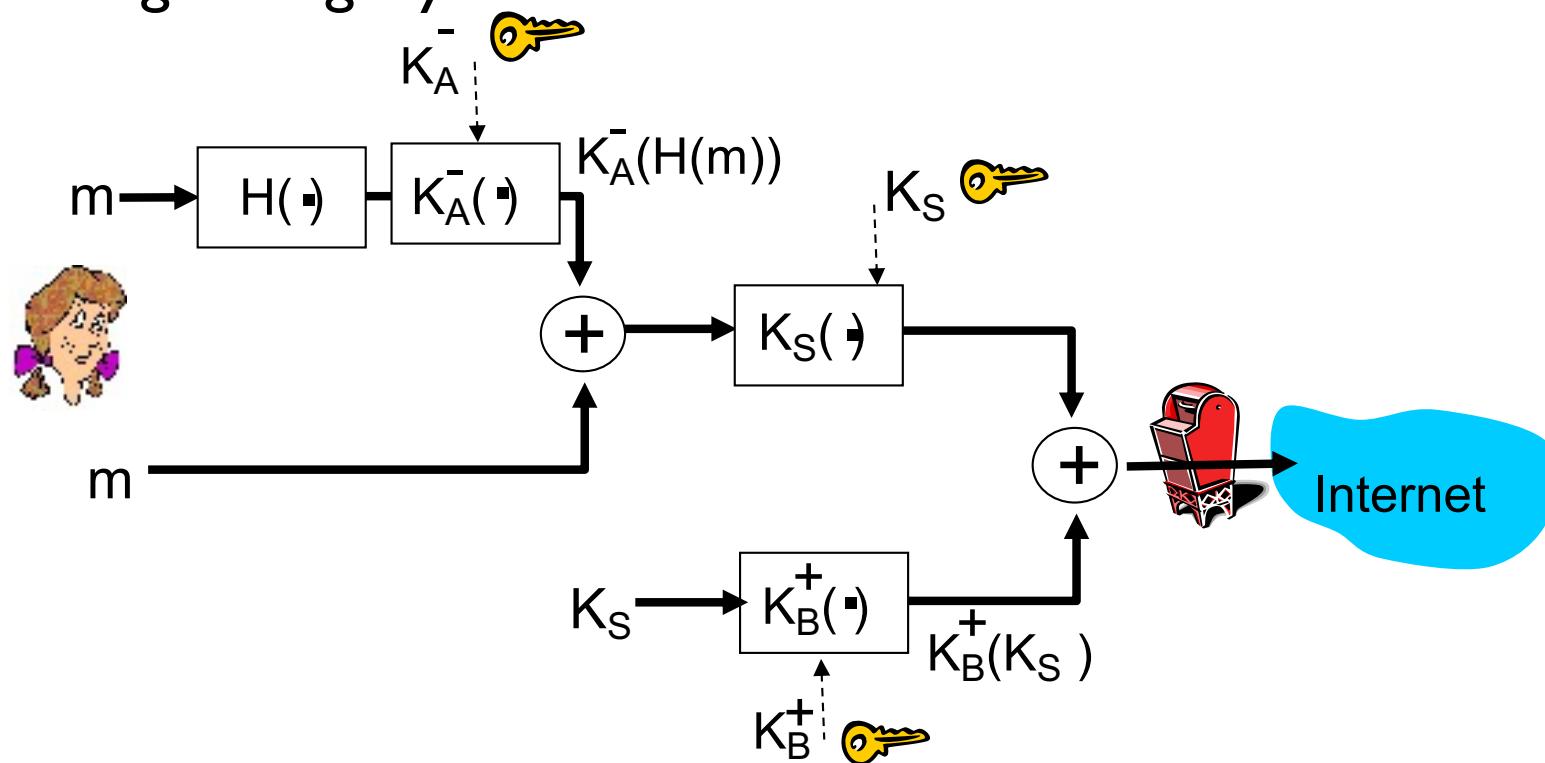
- ❖ Alice wants to provide sender authentication, message integrity



- ❖ Alice digitally signs message
- ❖ sends both message (in the clear) and digital signature

# Secure e-mail (continued)

- ❖ Alice wants to provide confidentiality, sender authentication, message integrity.



*Alice uses three keys:* her private key, Bob's public key, newly created symmetric key

# Secure E-mail: PGP

- ❖ De-factor standard for email encryption
- ❖ On installation PGP creates public, private key pair
  - Public key posted on user's webpage or placed in a public key server
  - Private key protected by password
- ❖ Option to digitally sign the message, encrypt the message or both
- ❖ MD5 or SHA for message digest
- ❖ CAST, triple-DES or DEA for symmetric key encryption
- ❖ RSA for public key encryption

# Secure E-mail: PGP

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv
yHJRhhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

**Figure 8.22** ♦ A PGP signed message

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=1KhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE-----
```

**Figure 8.23** ♦ A secret PGP message

# Network Security: Conclusion

- ❖ What is security?
- ❖ Symmetric and Asymmetric cryptography
- ❖ Encryption
- ❖ Authentication
- ❖ Message Integrity
  - Digital Signatures
  - MAC
- ❖ Secure E-mail
  - Putting it all together