

# 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/>

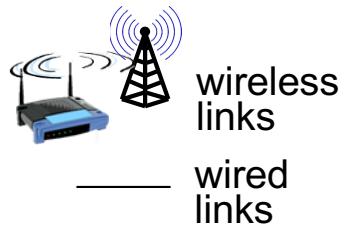
# **Quiz: What is the Internet?**



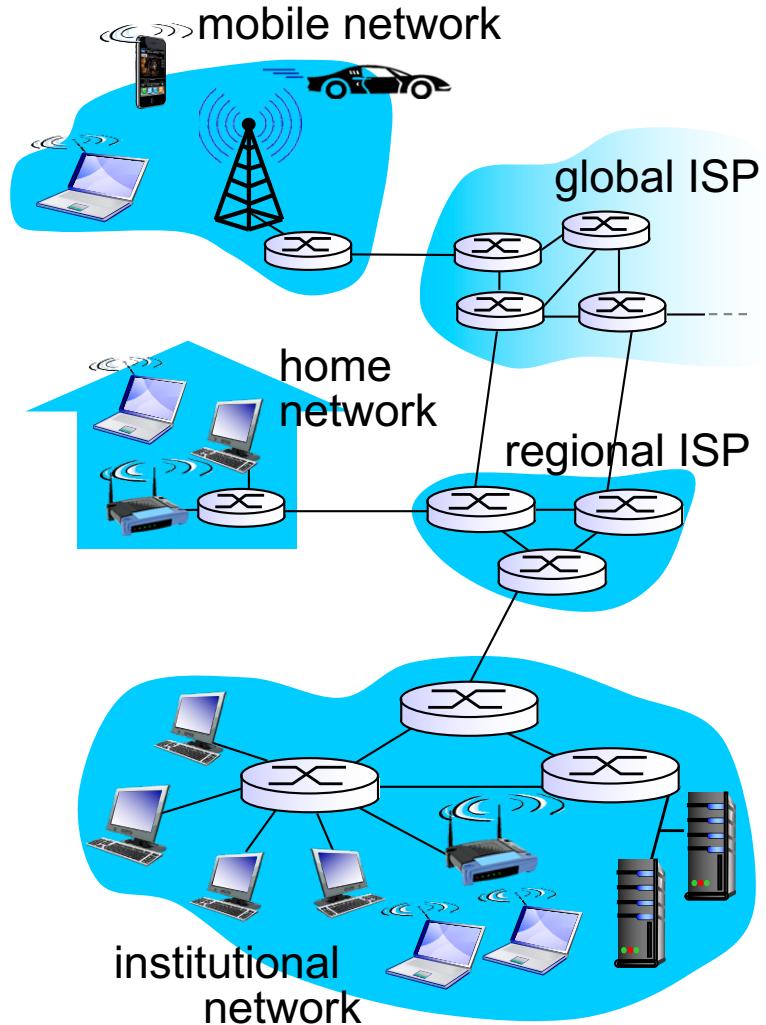
- A. One single homogenous network**
- B. An interconnection of different computer networks**
- C. An infrastructure that provides services to networked applications**
- D. Something else (be prepared to discuss)**

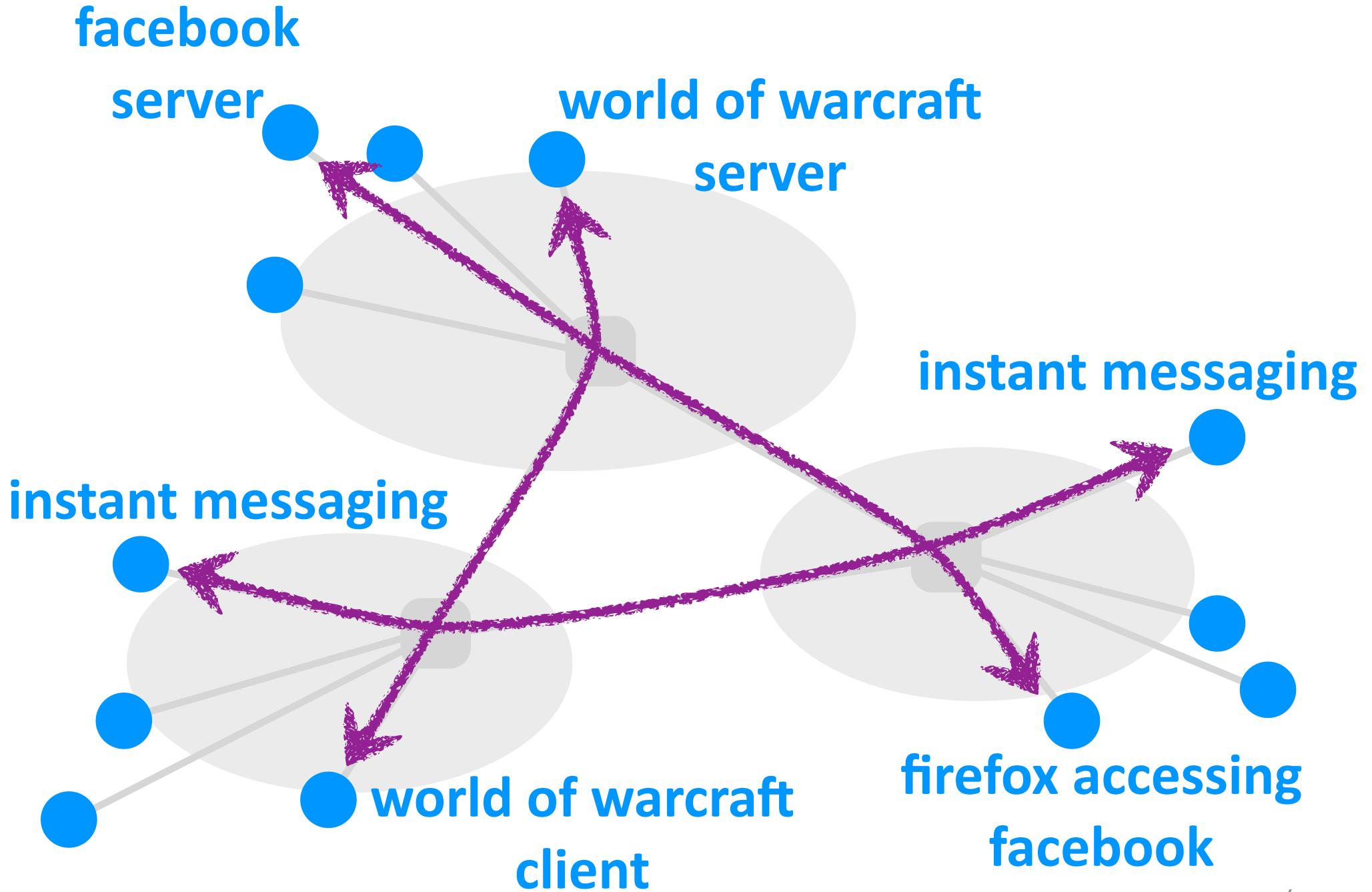
Open a browser and type: [www.zetings.com/salil](http://www.zetings.com/salil)

# 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 switches*





# “Fun” Internet appliances



Internet refrigerator



Picture frame



sensorized,  
bed  
mattress



Networked TV Set top Boxes



Web-enabled toaster +  
weather forecaster

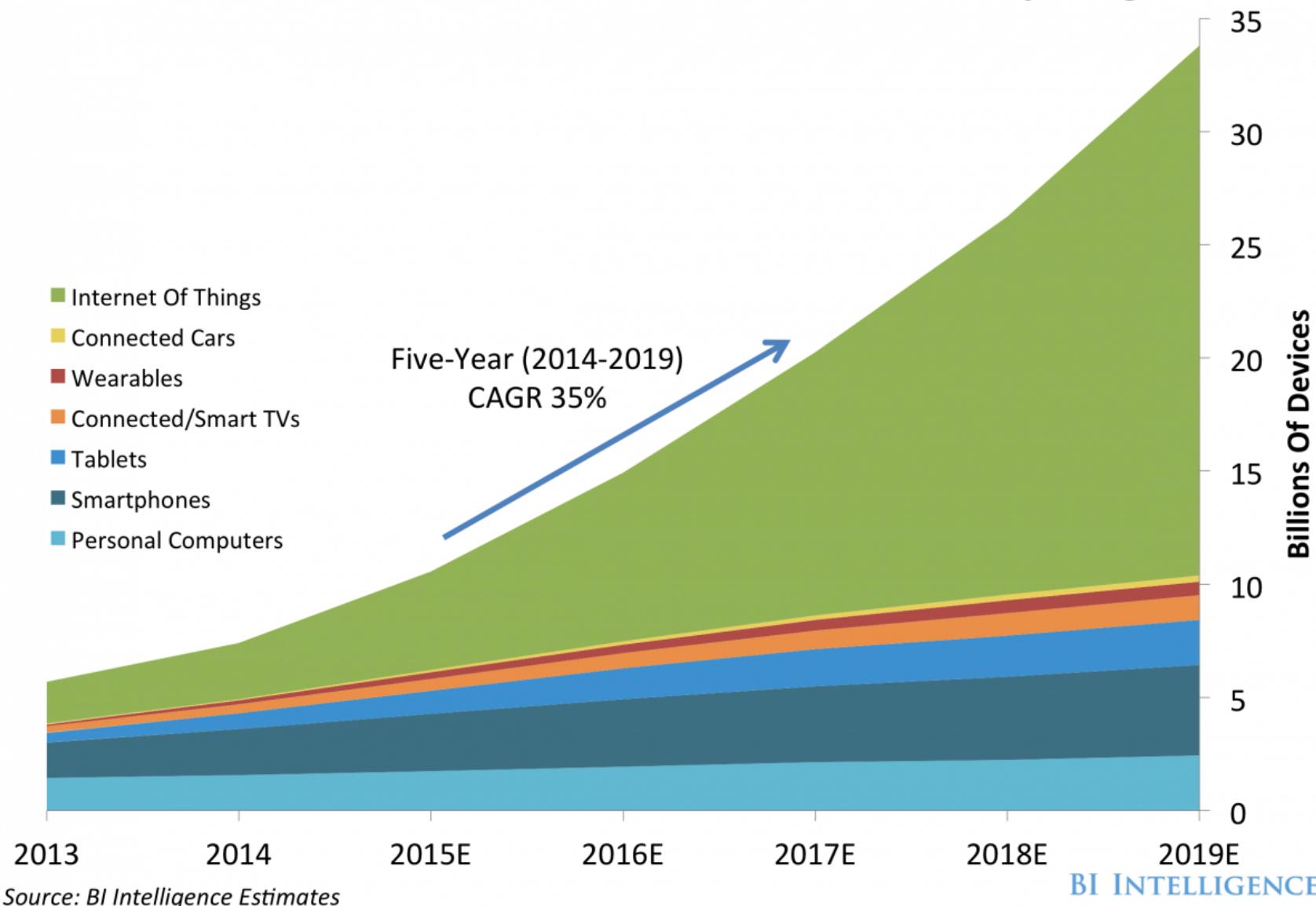


Tweet-a-watt:  
monitor energy use



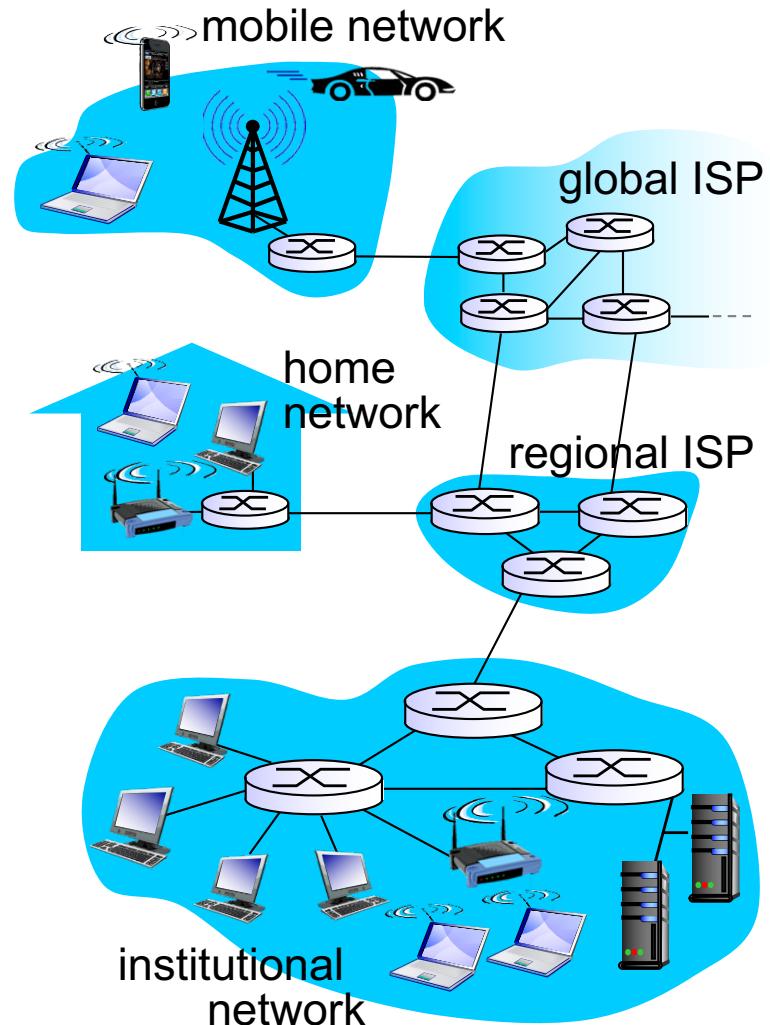
Smart Lightbulbs

# Number Of Devices In The Internet Of Everything



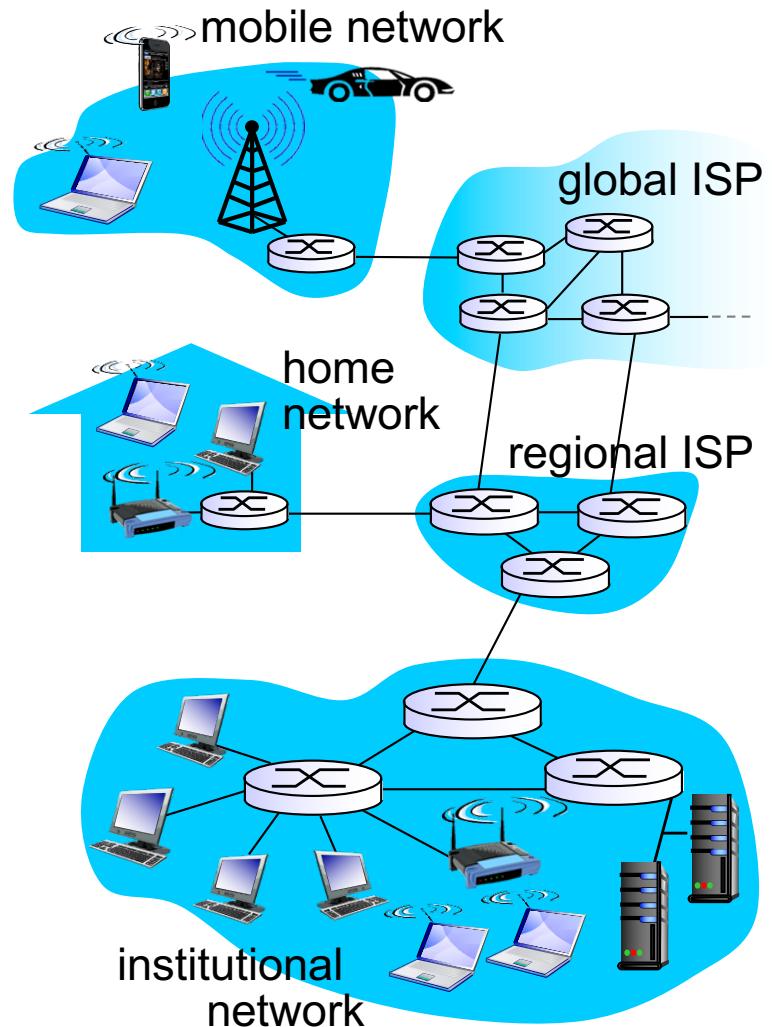
# What's the Internet: “nuts and bolts” view

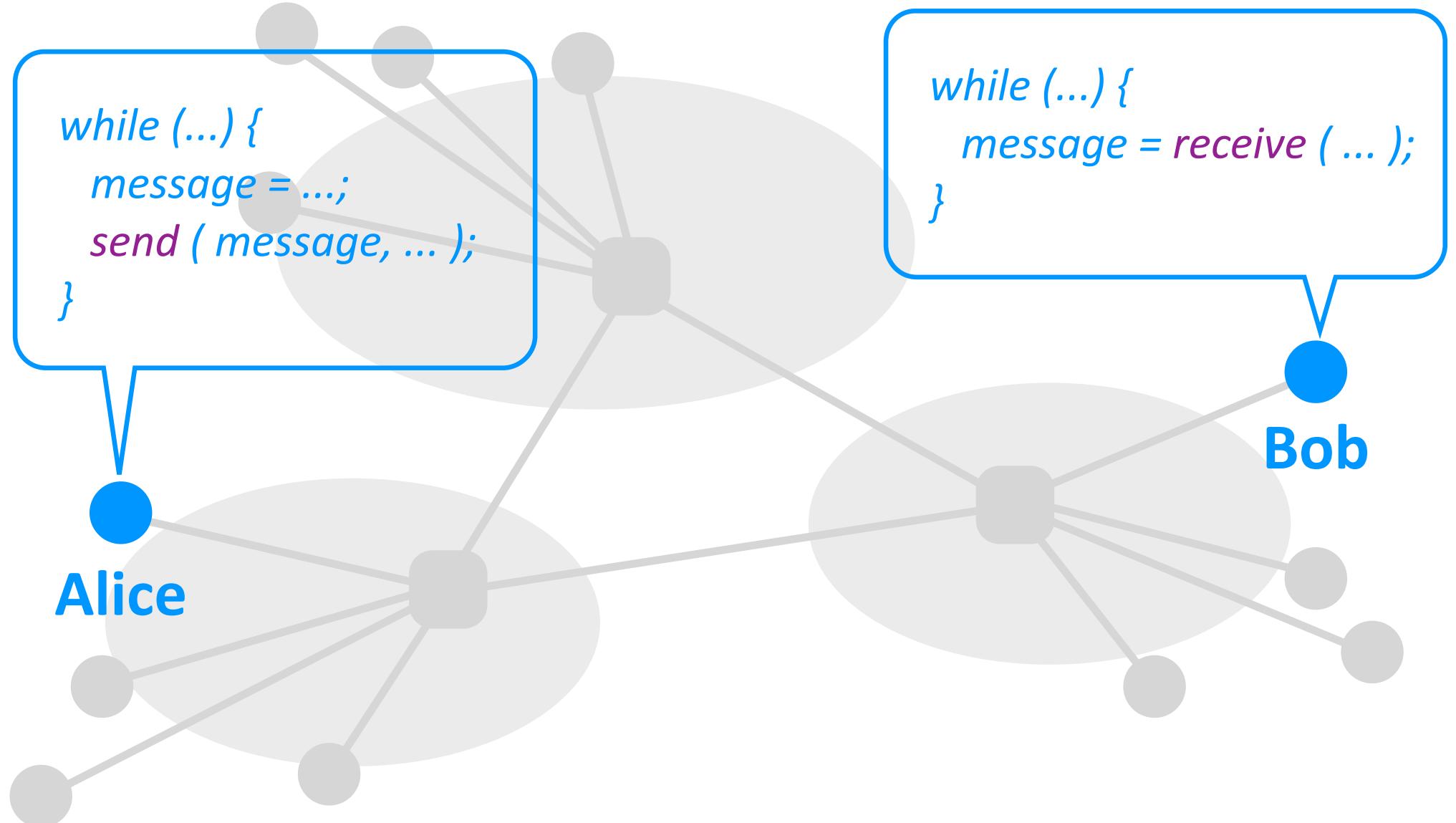
- ❖ *Internet: “network of networks”*
  - Interconnected ISPs
- ❖ *protocols* control sending, receiving of msgs
  - e.g., TCP, IP, HTTP, Skype, 802.11
- ❖ *Internet standards*
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force



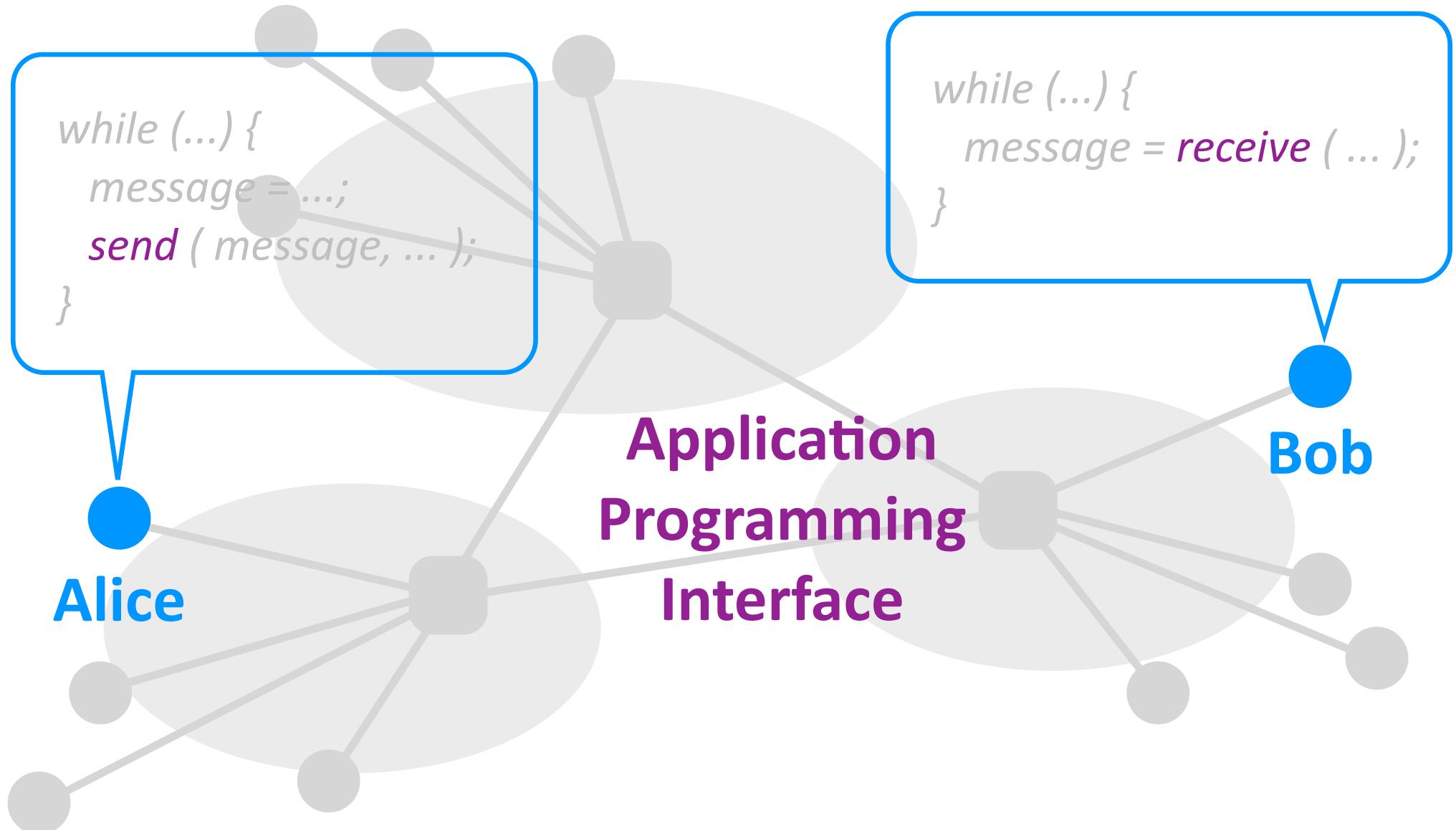
# What's the Internet: a service view

- ❖ *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

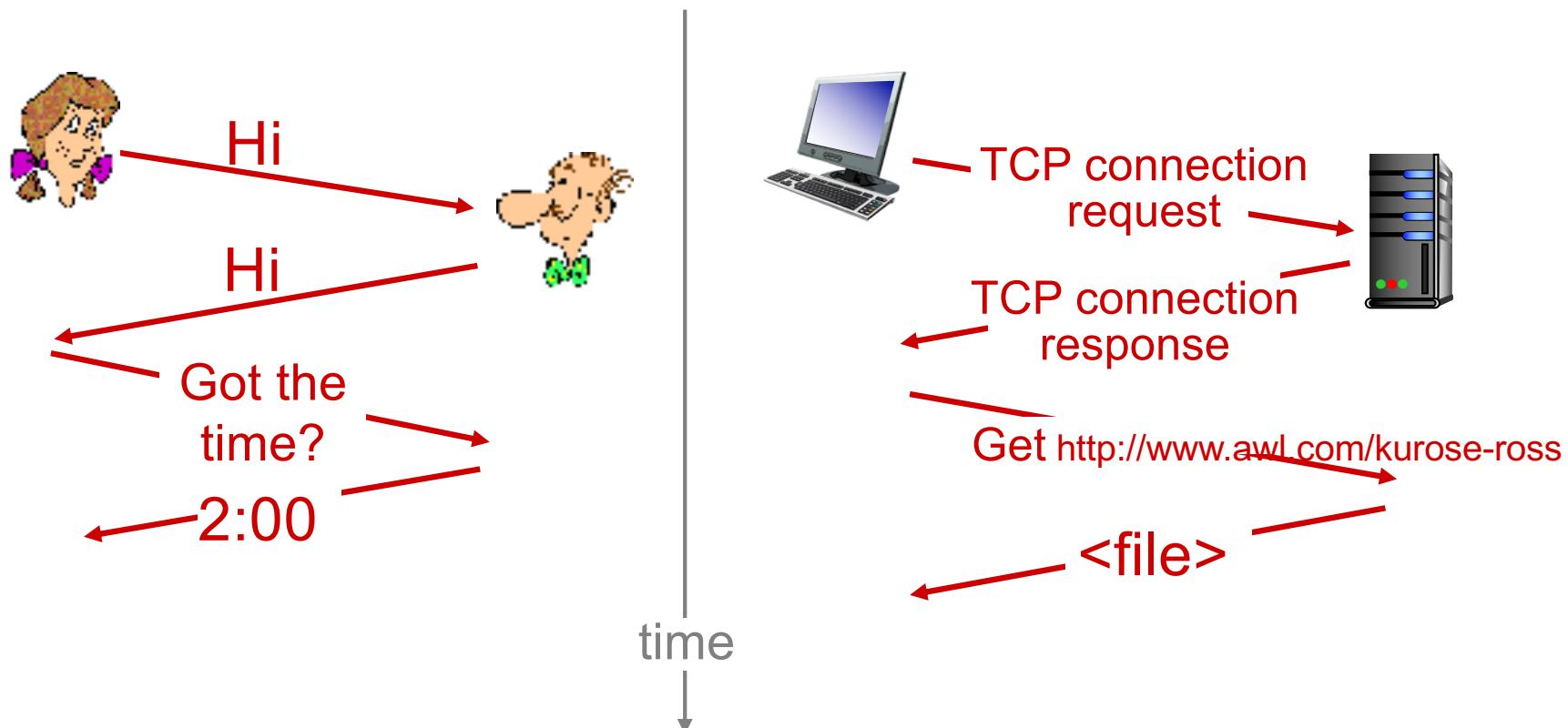
## *network protocols:*

- ❖ 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?

# 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

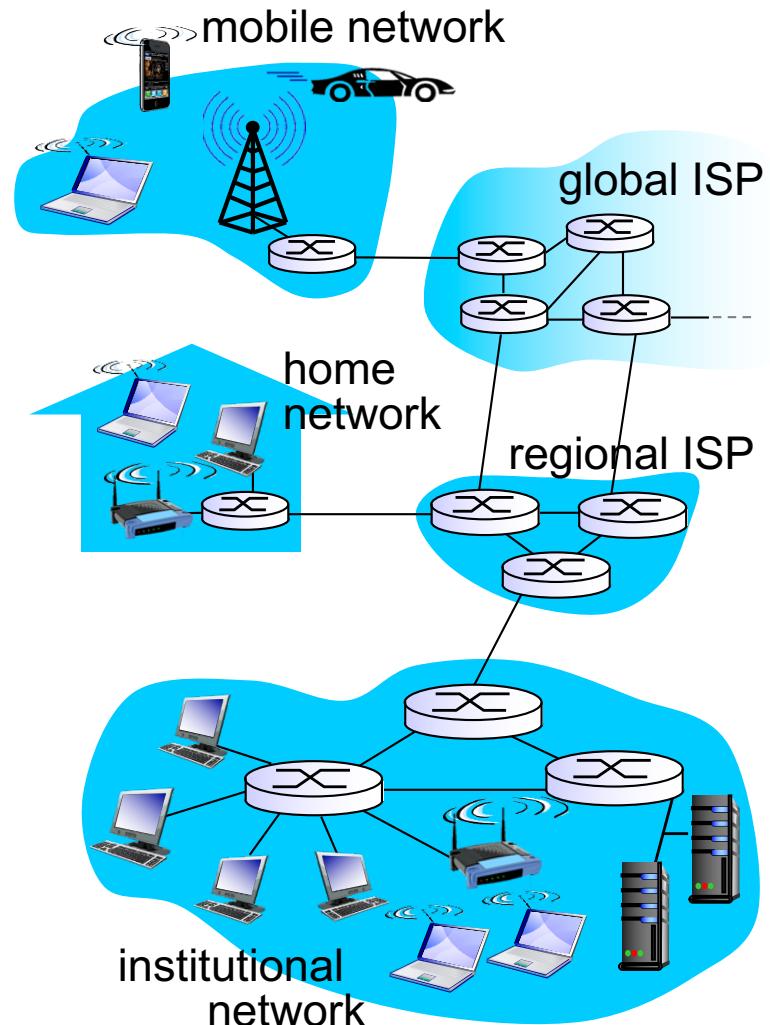
I.5 protocol layers, service models

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



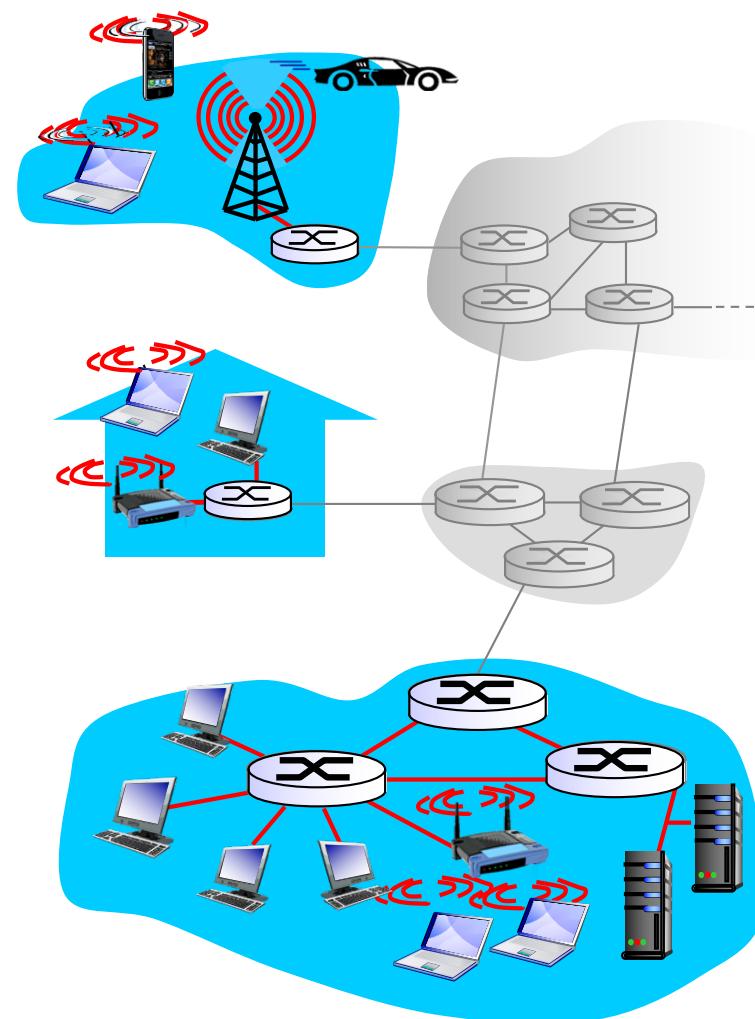
# Access networks and physical media

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

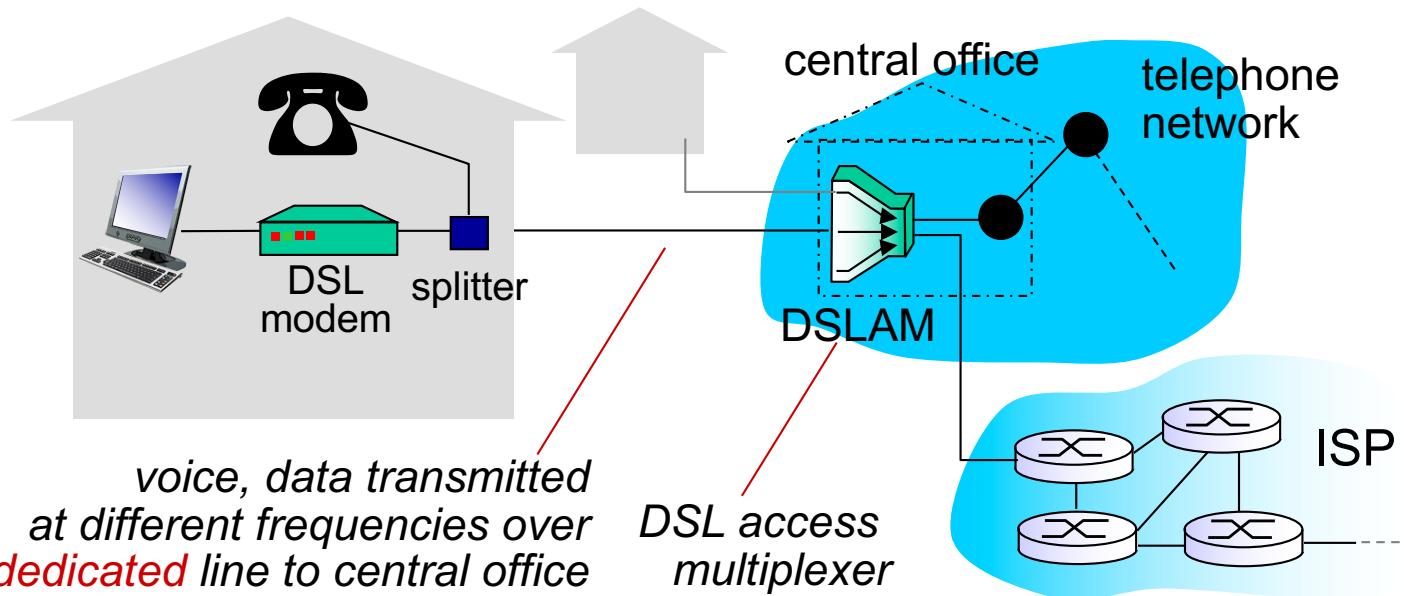
- ❖ residential access nets
- ❖ institutional access networks (school, company)
- ❖ mobile access networks

*keep in mind:*

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

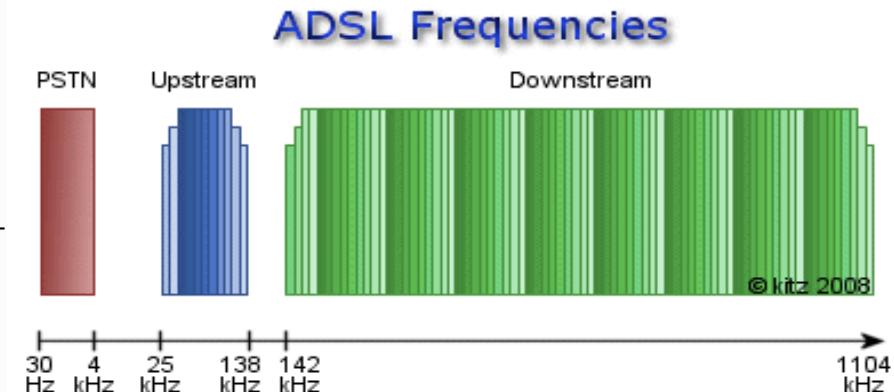
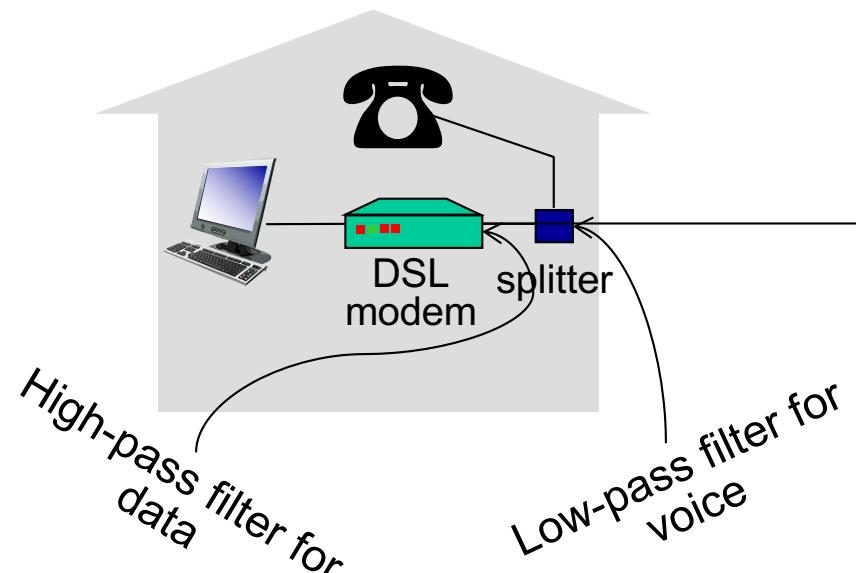


# Access net: digital subscriber line (DSL)



- ❖ 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)

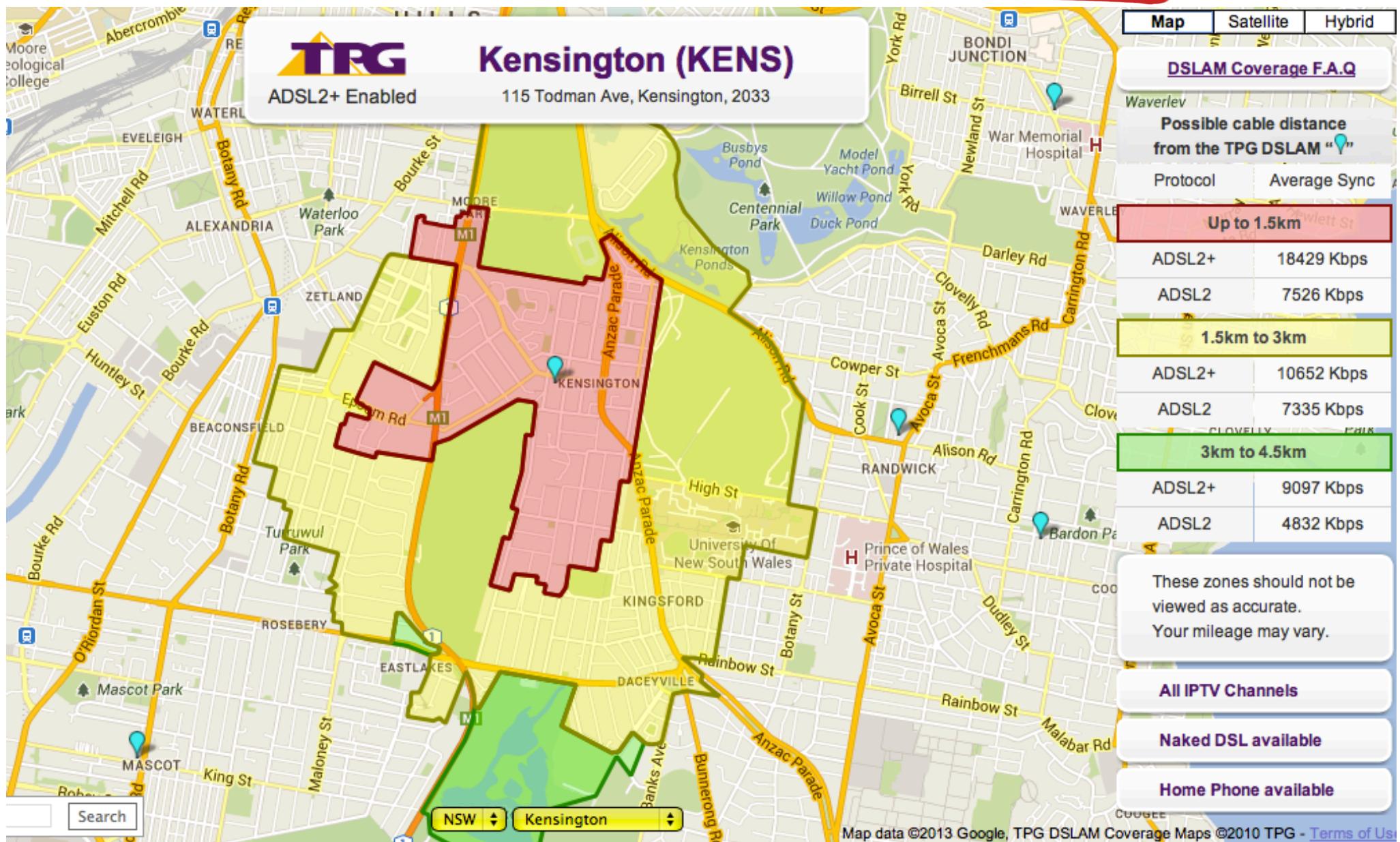


ADSL over POTS

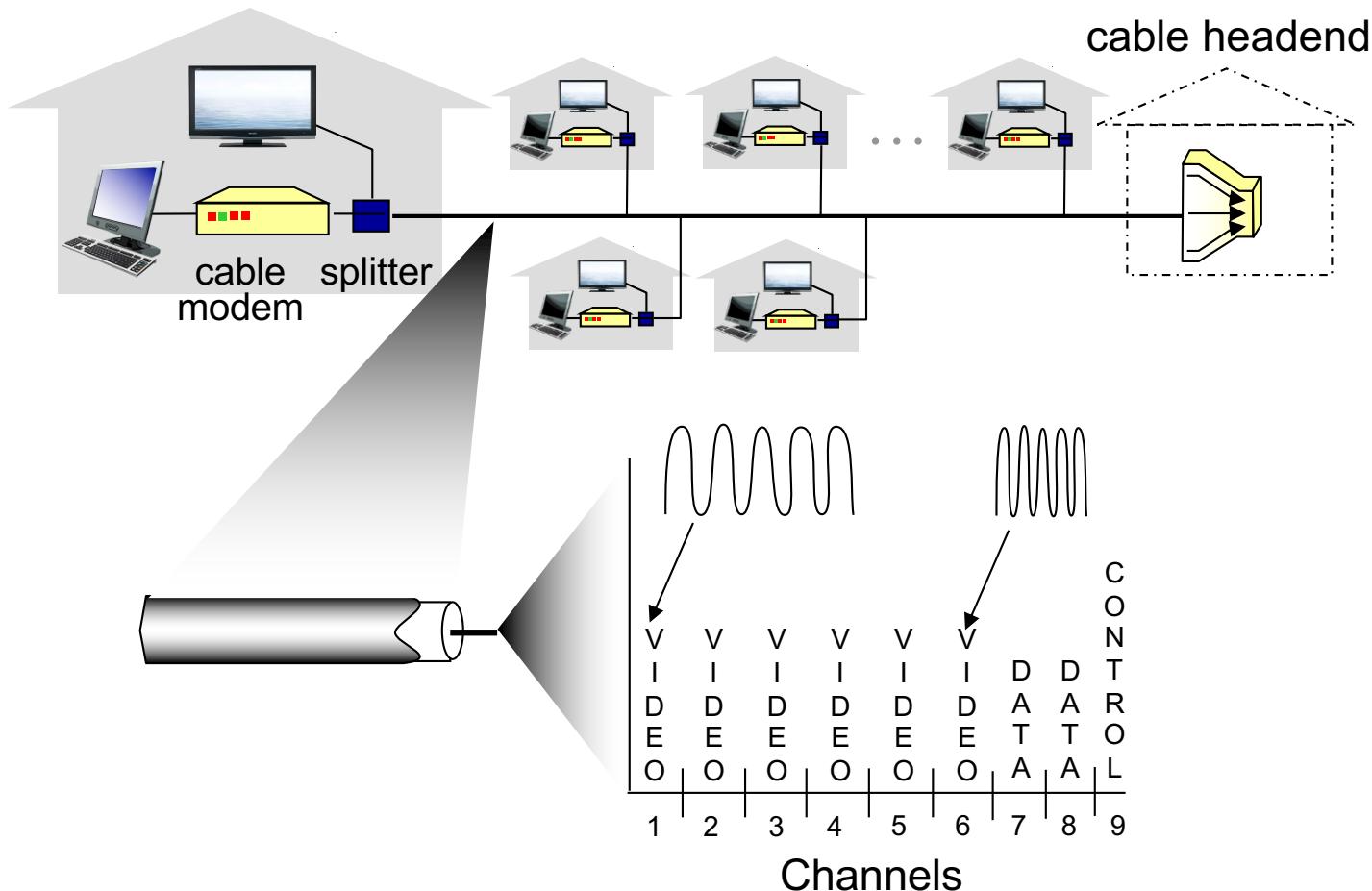
*voice, data transmitted  
at different frequencies over  
dedicated line to central office*

- 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)

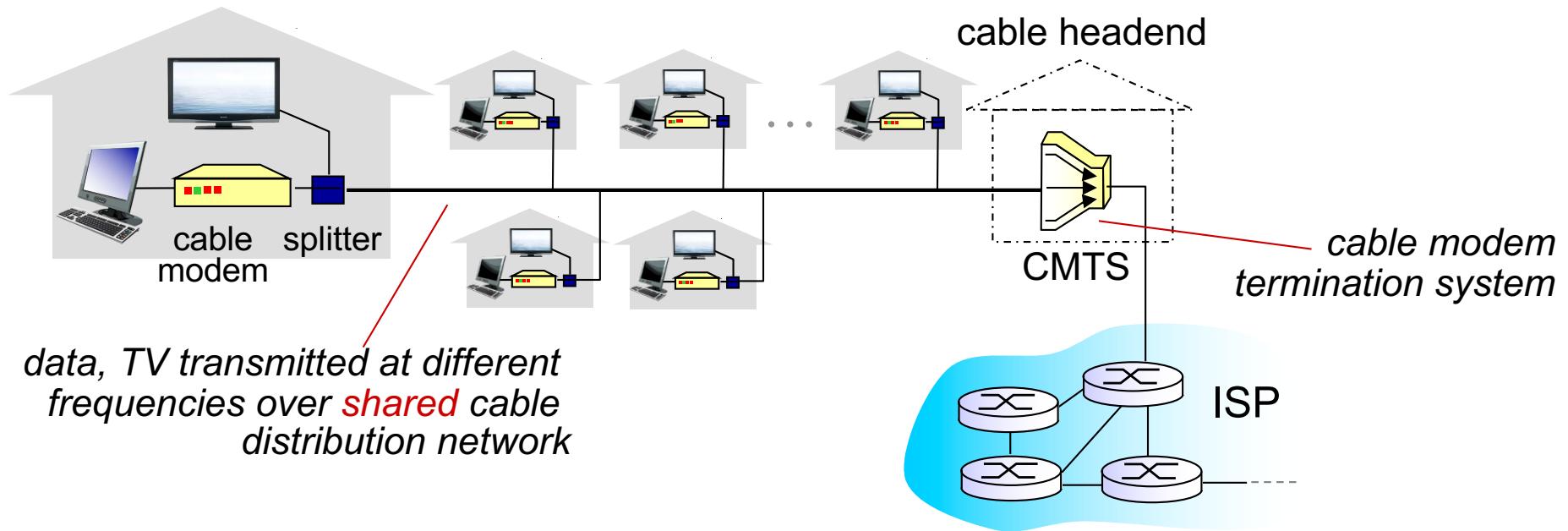


# Access net: cable network



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

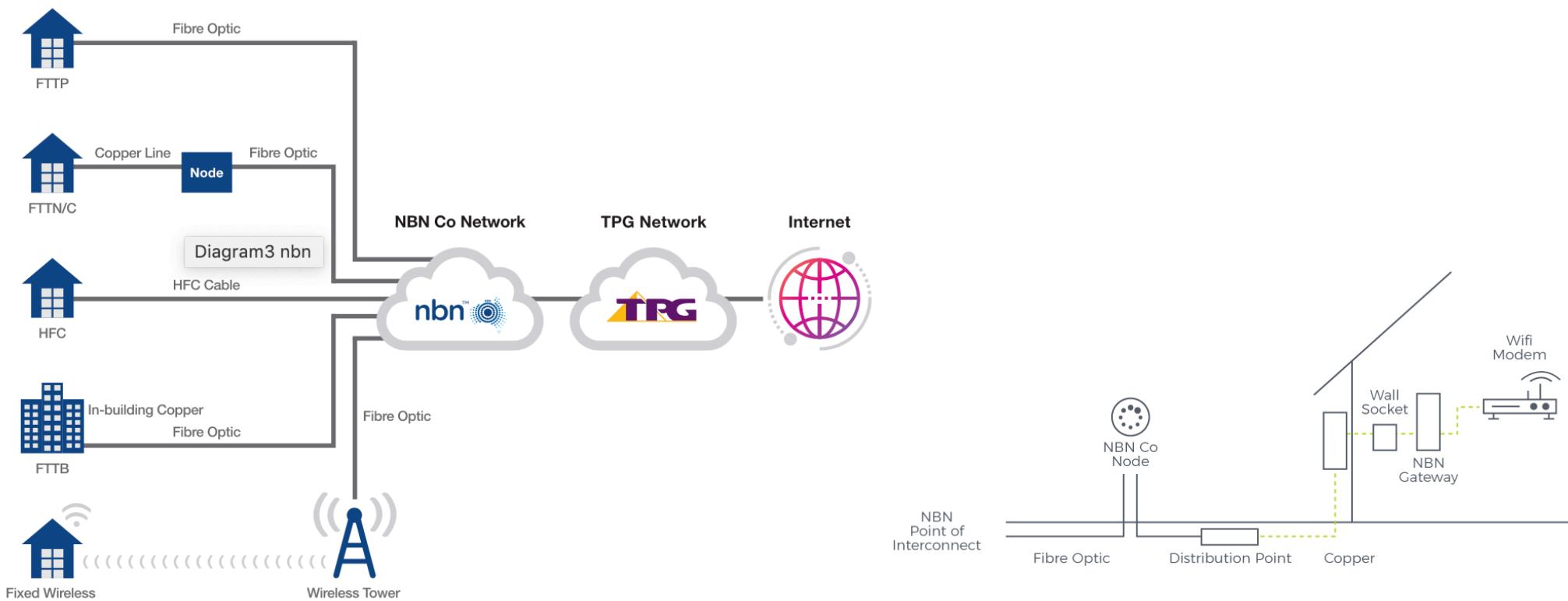
# Access net: cable network



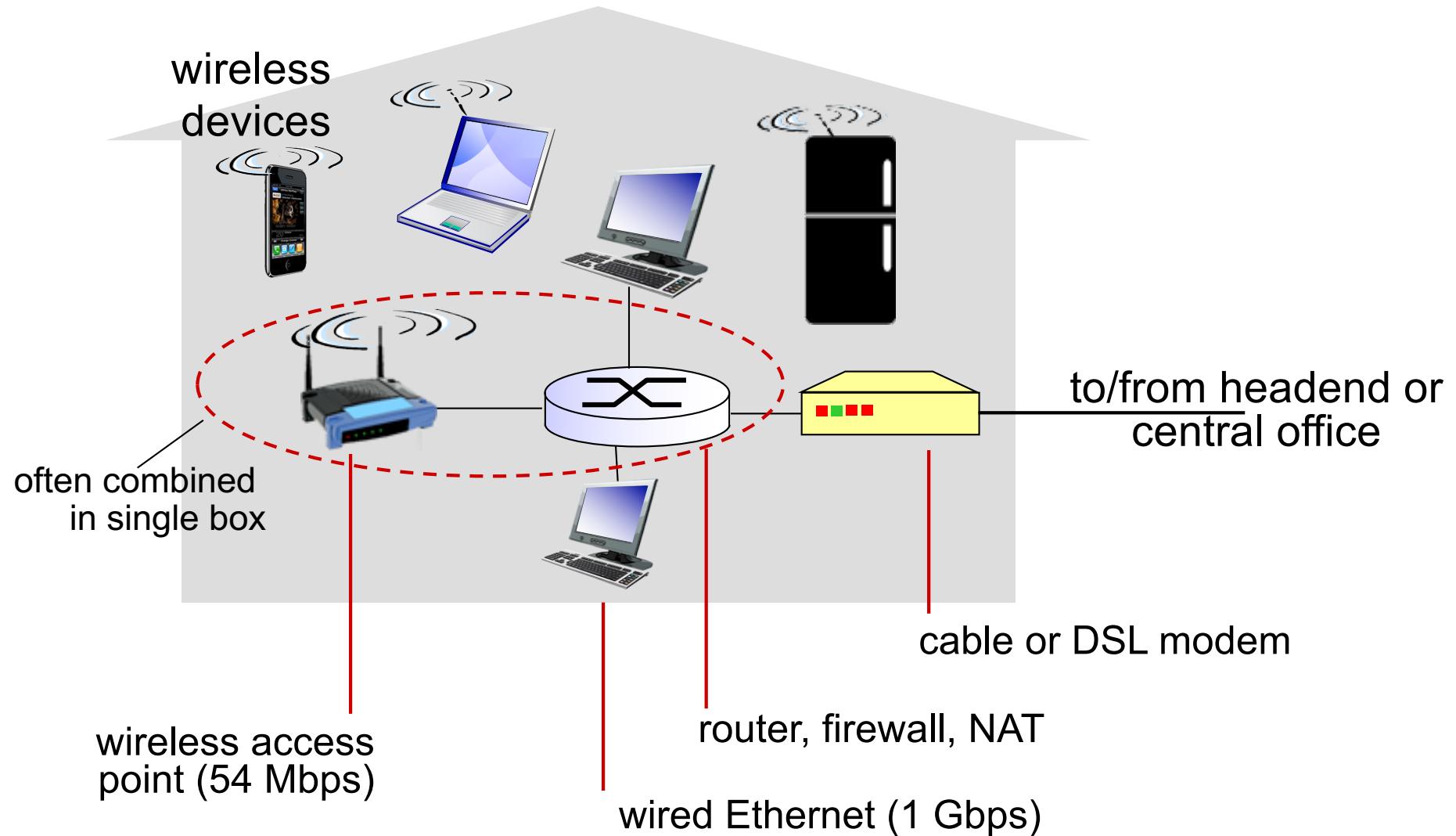
- ❖ HFC: hybrid fiber coax
  - asymmetric: up to 30Mbps downstream transmission rate, 2 Mbps upstream transmission rate
- ❖ network of cable, fiber attaches homes to ISP router
  - homes **share access network** to cable headend
  - unlike DSL, which has dedicated access to central office

# Fiber to the home/premise/curb

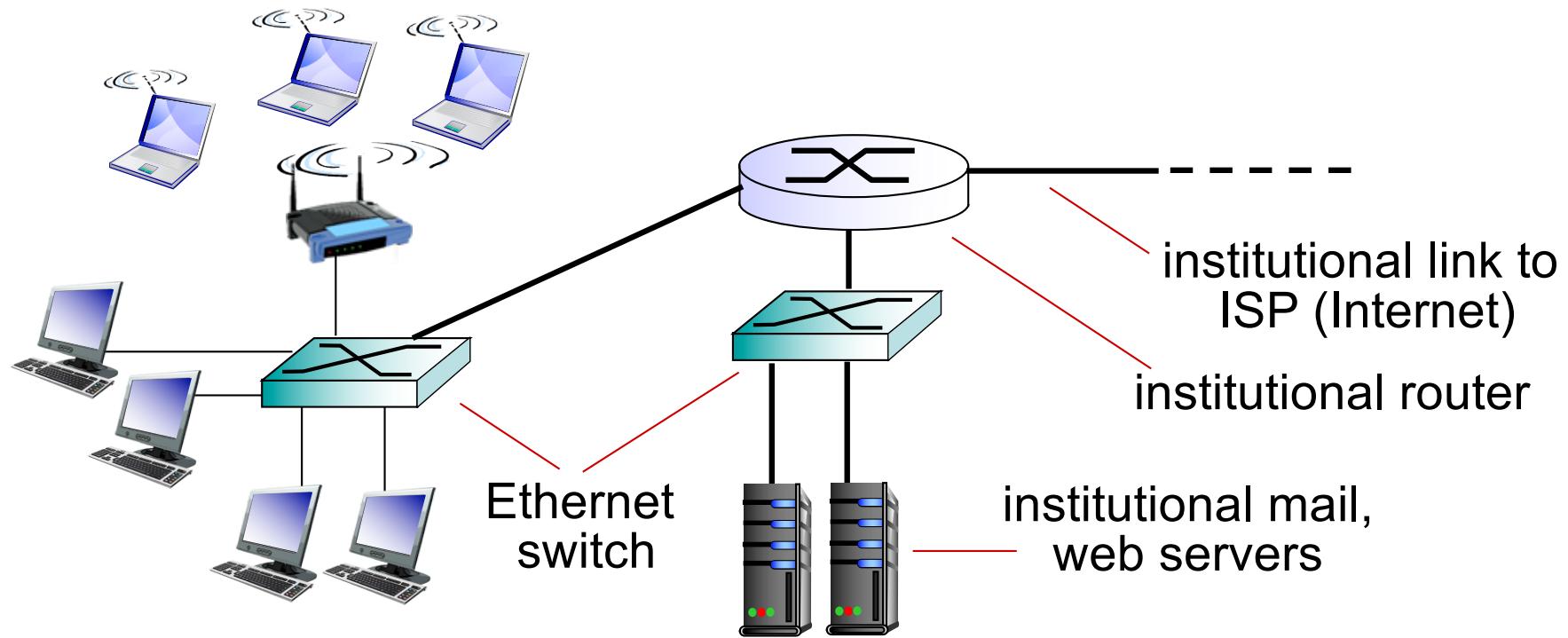
- ❖ Fully optical fiber path all the way to the home
  - e.g., NBN, Google, Verizon FIOS
  - ~30 Mbps to 1 Gbps



# 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”

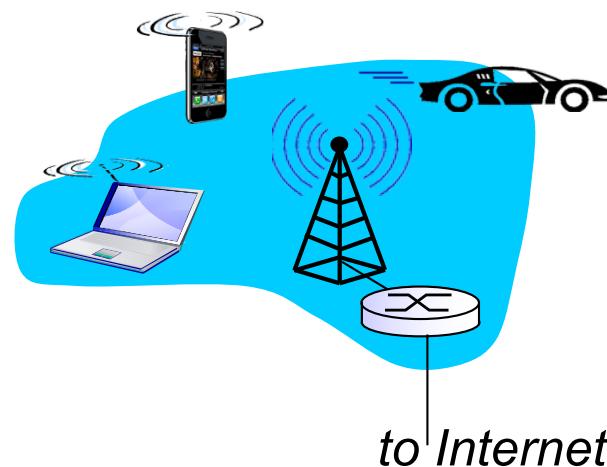
## wireless LANs:

- 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



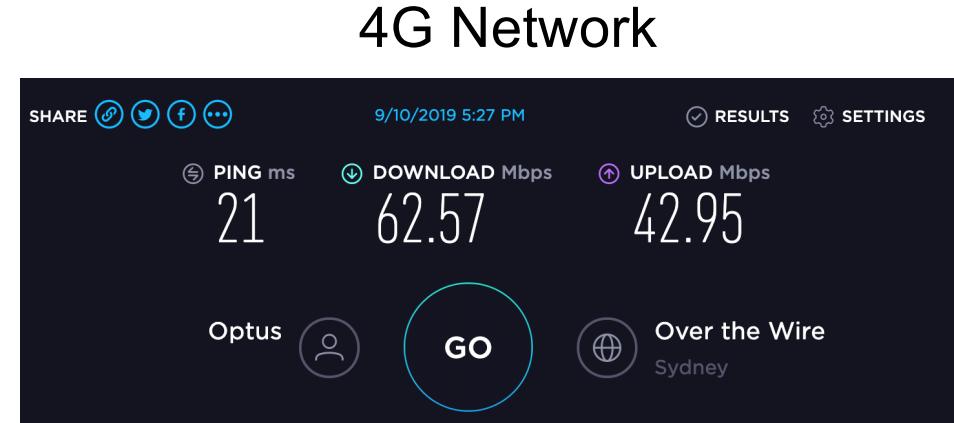
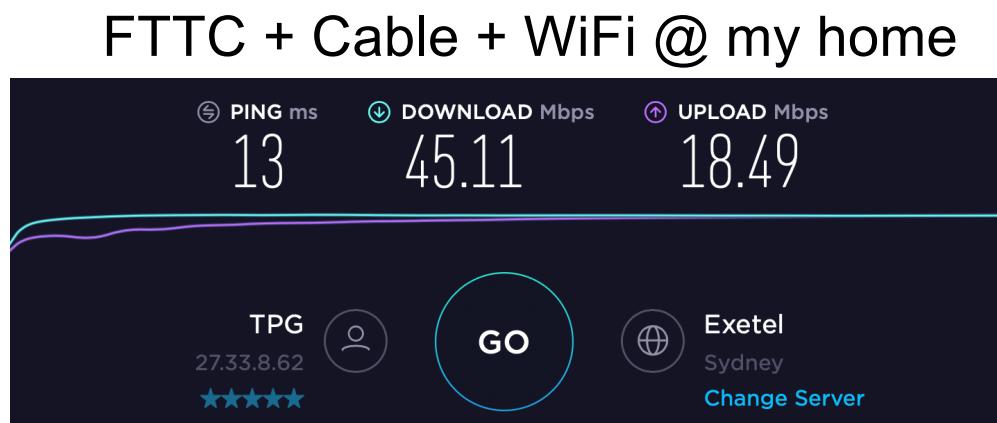
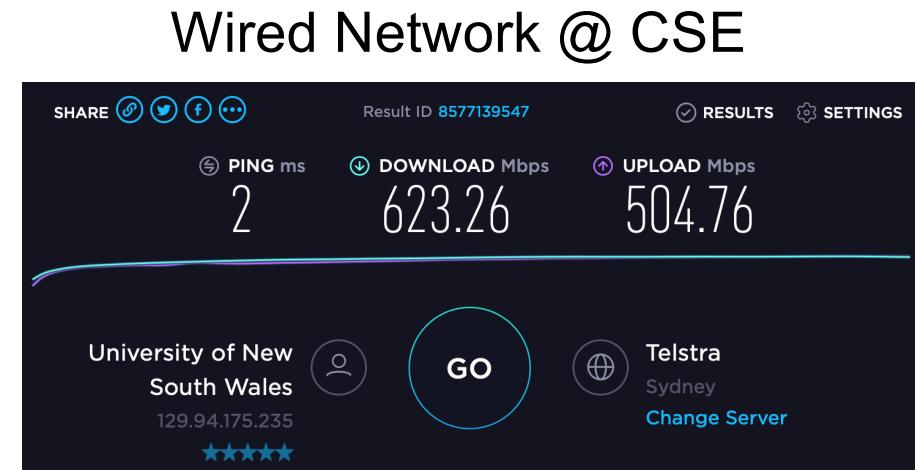
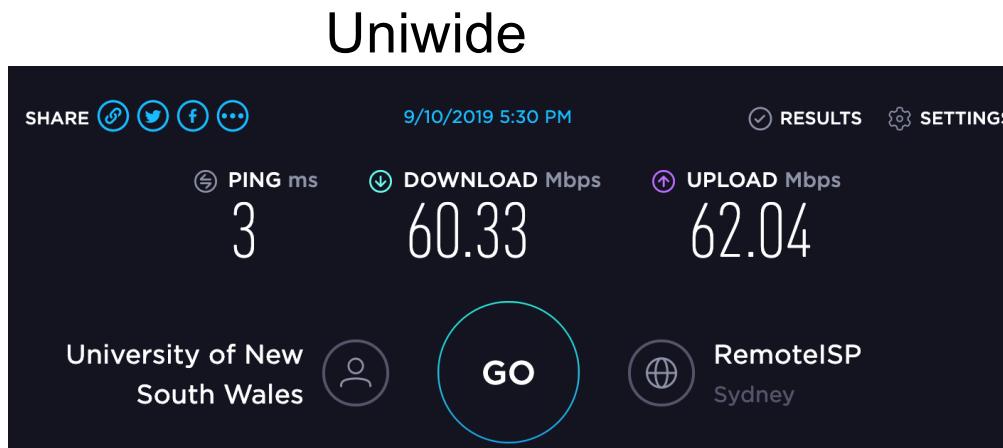
## wide-area wireless access

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



# Sample results

Can you explain the differences?



# 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

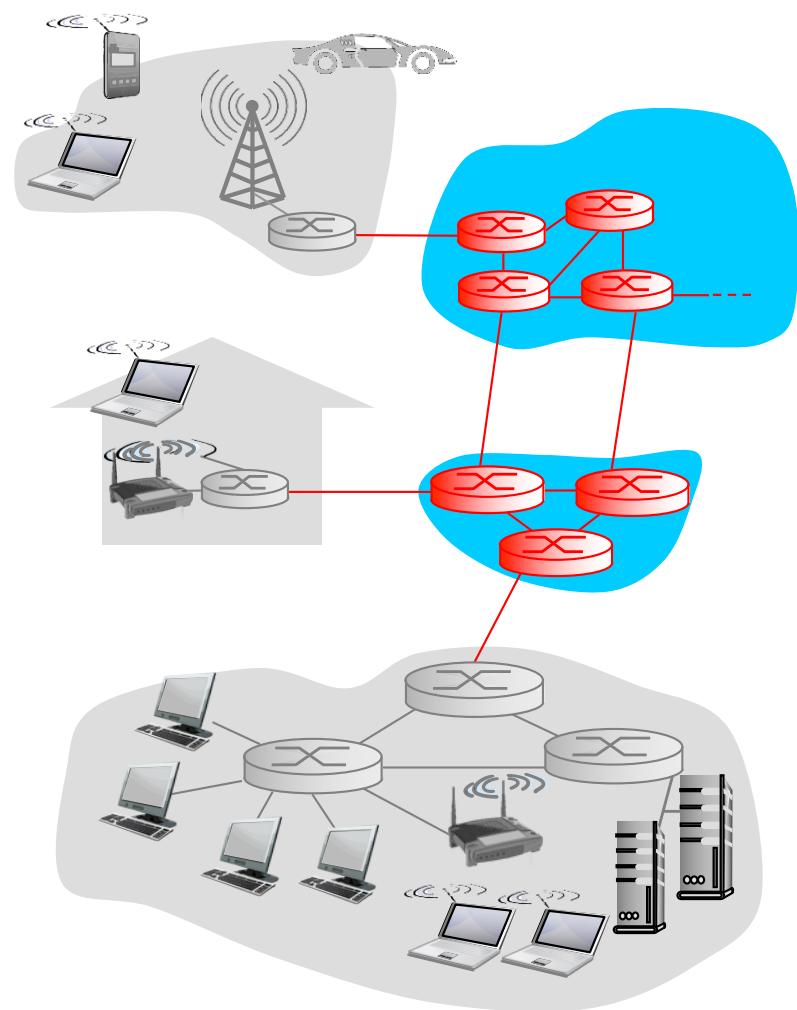
I.5 protocol layers, service models

I.6 networks under attack: security

I.7 history

# The network core

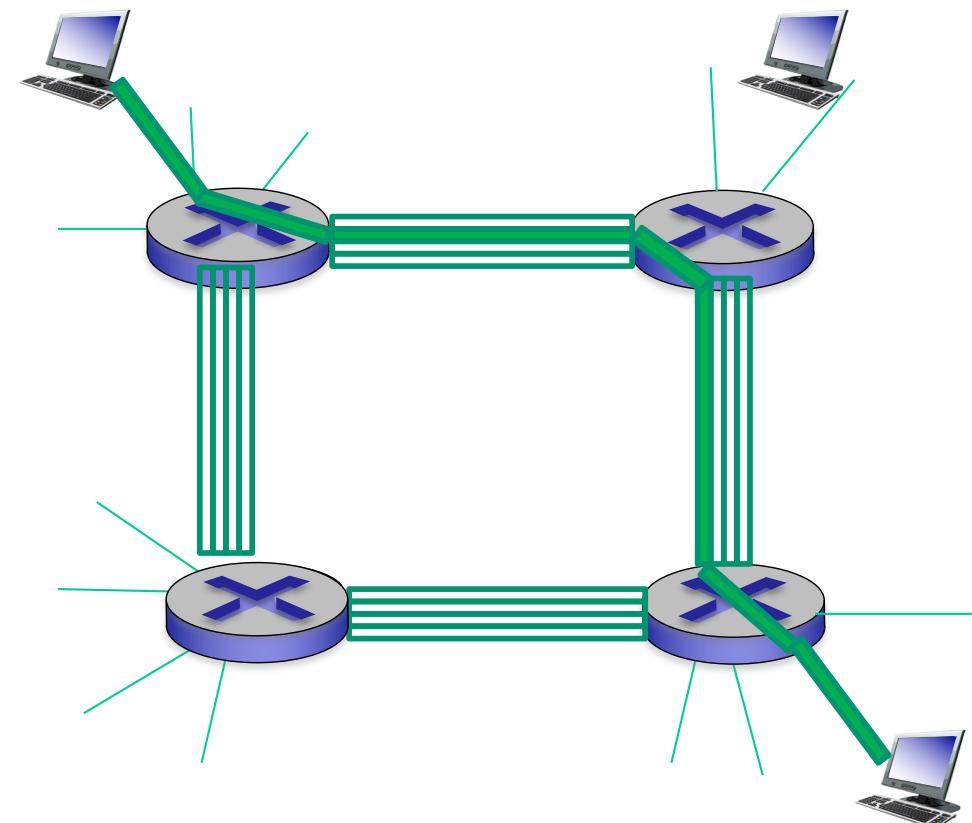
- ❖ mesh of interconnected routers/switches
- ❖ Two forms of switched networks:
  - Circuit switching: used in the legacy telephone networks
  - Packet switching: used in the Internet



# Circuit Switching

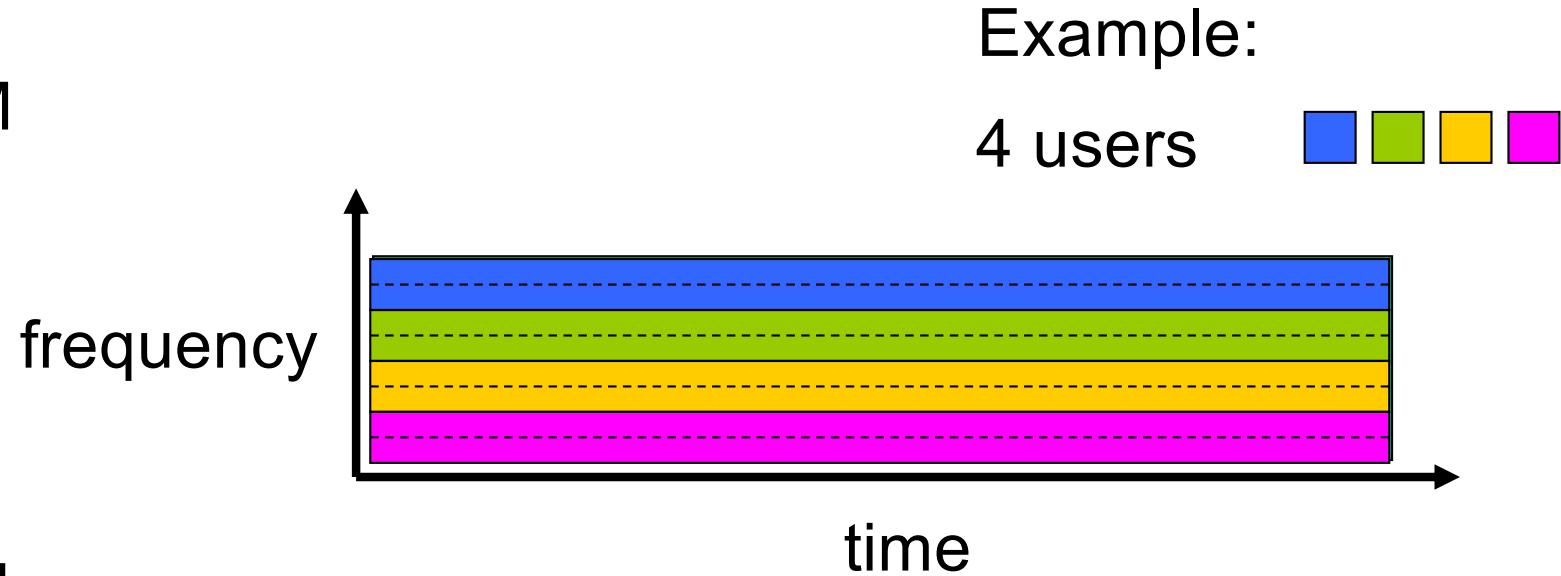
end-end resources allocated to, reserved for “call” between source & dest:

- 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

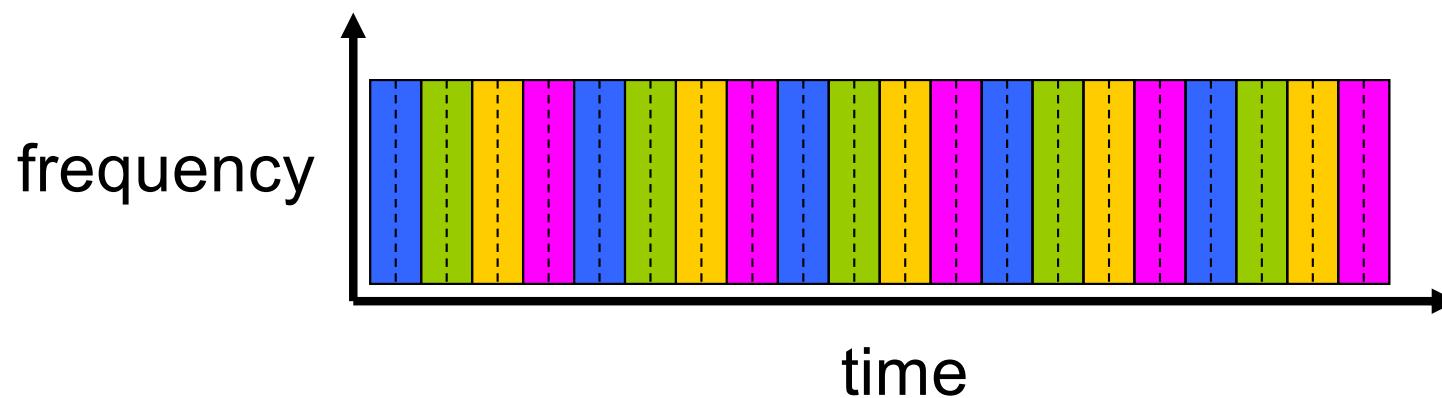


# Circuit switching: FDM versus TDM

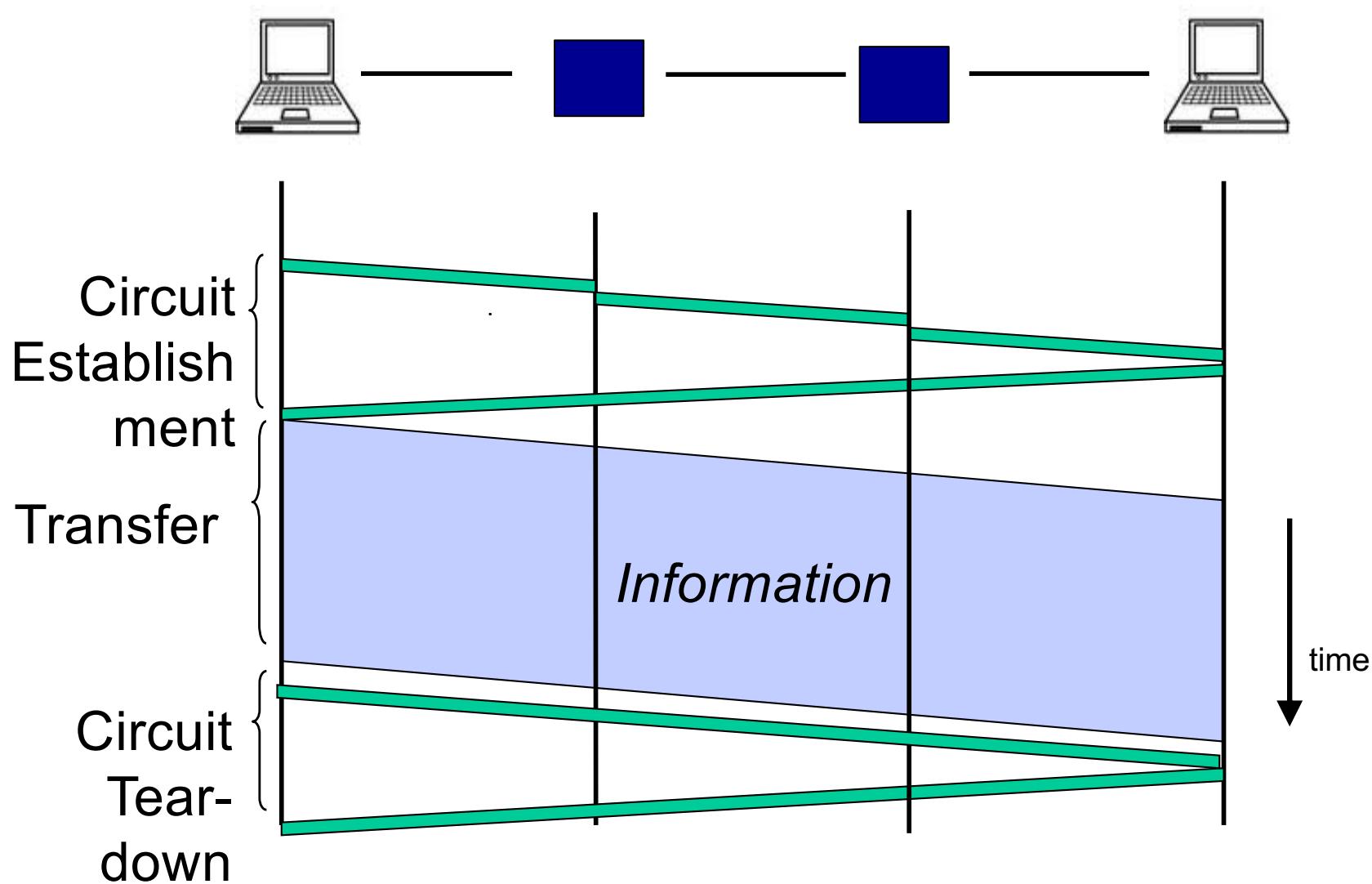
FDM



TDM



# Timing in Circuit Switching



## **Quiz: What are the pros and cons of circuit switching? Let's discuss ..**



- ❖ Pros:

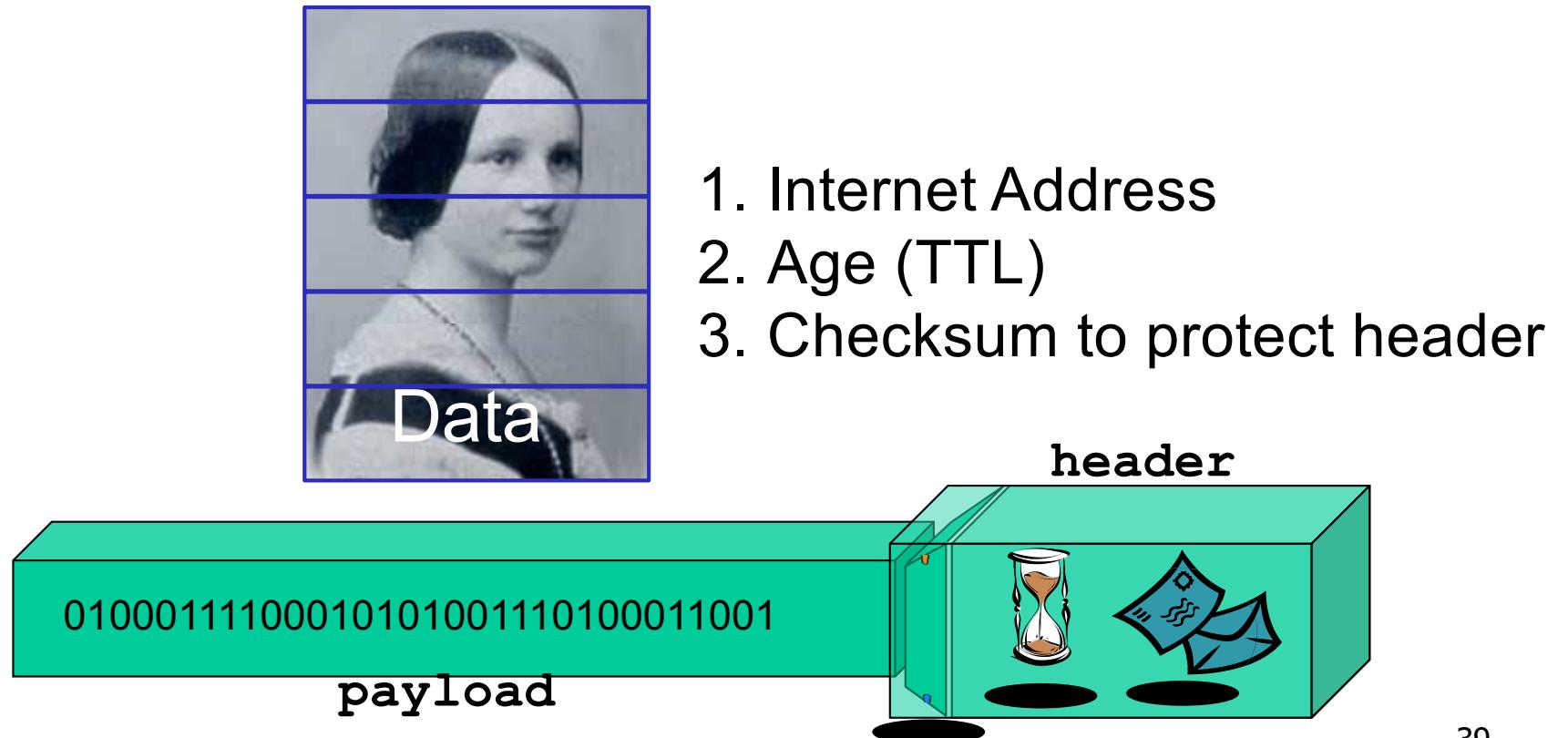
- ❖ Cons:

# 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”



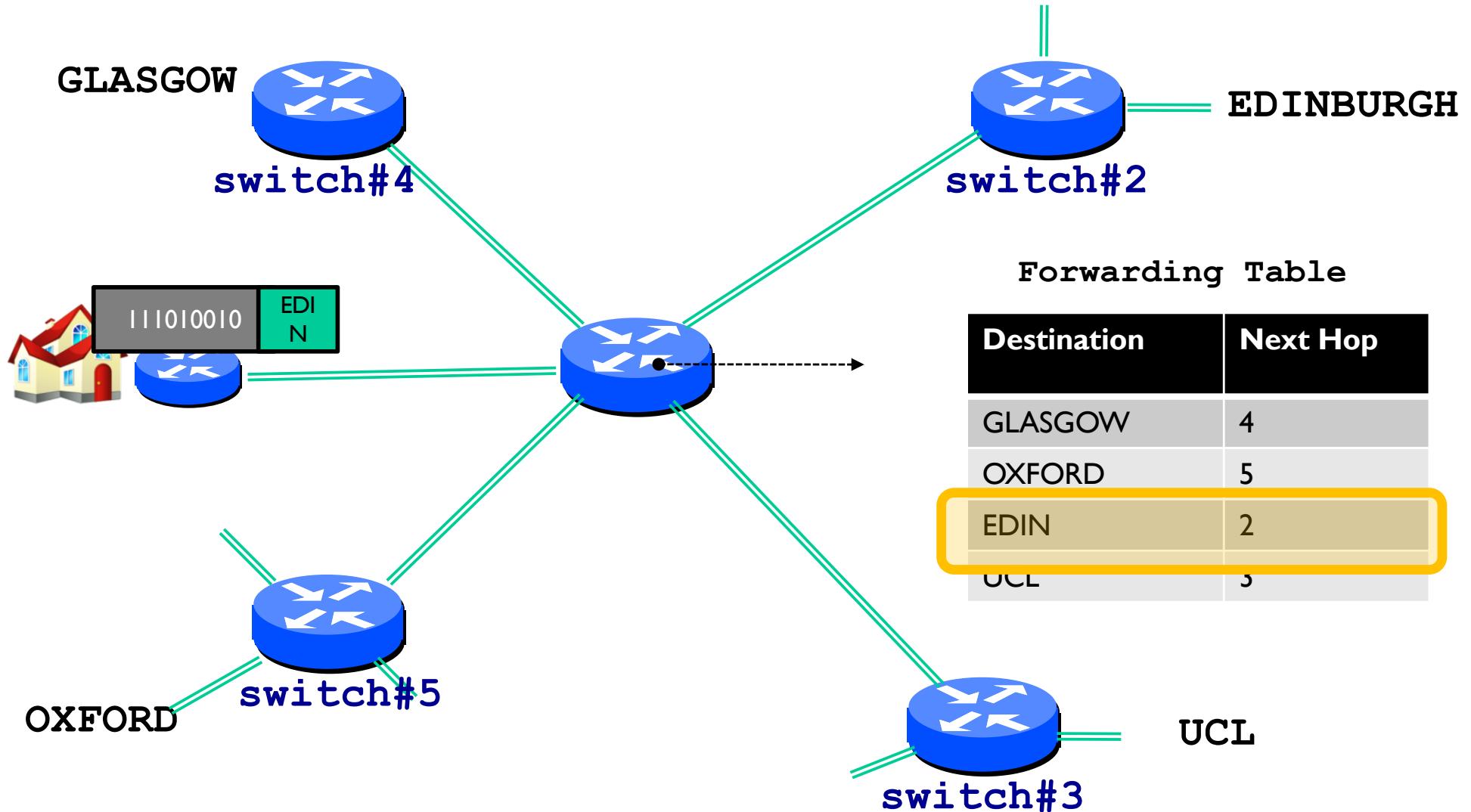
# Packet Switching

- ❖ 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)

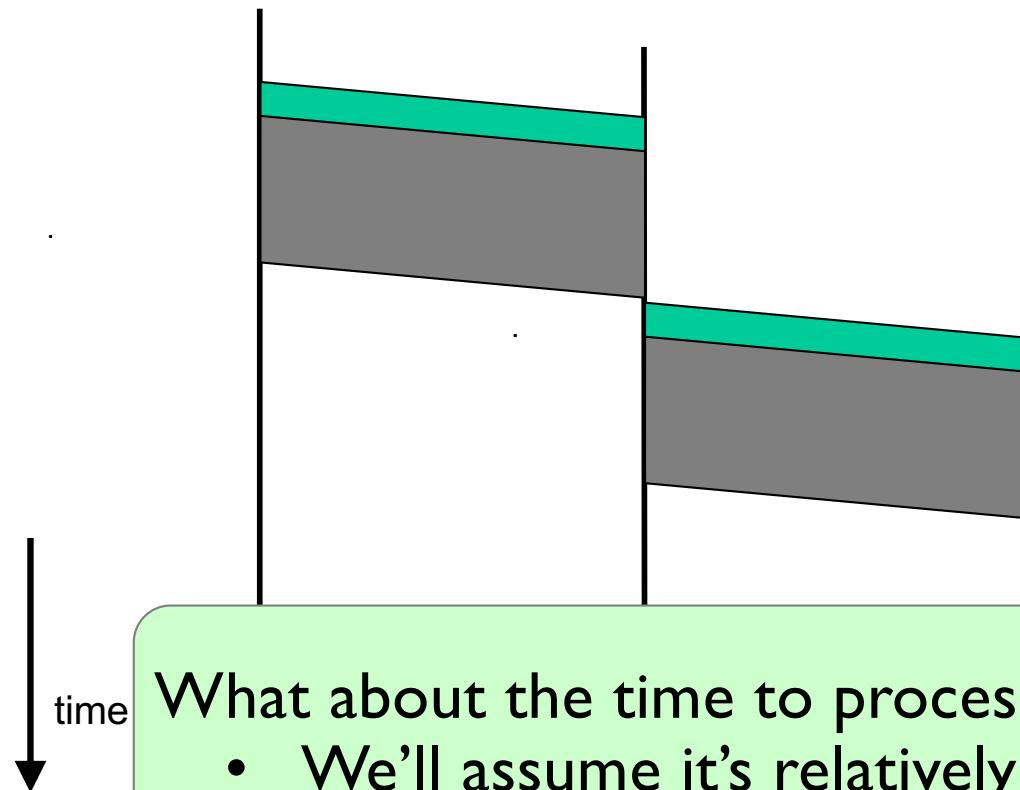
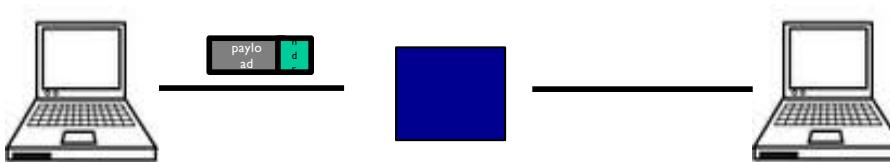
# Packet Switching

- ❖ 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

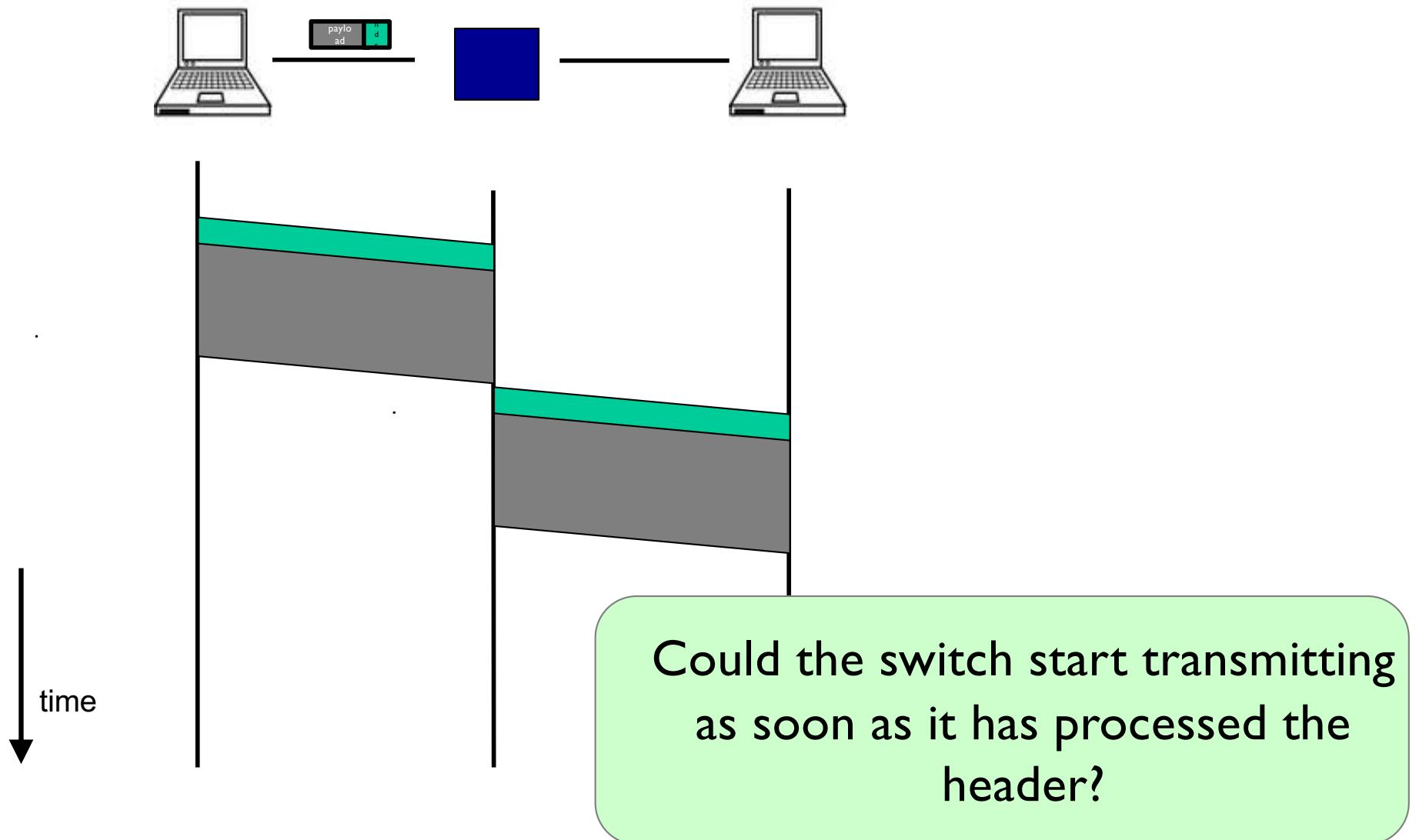


# Timing in Packet Switching

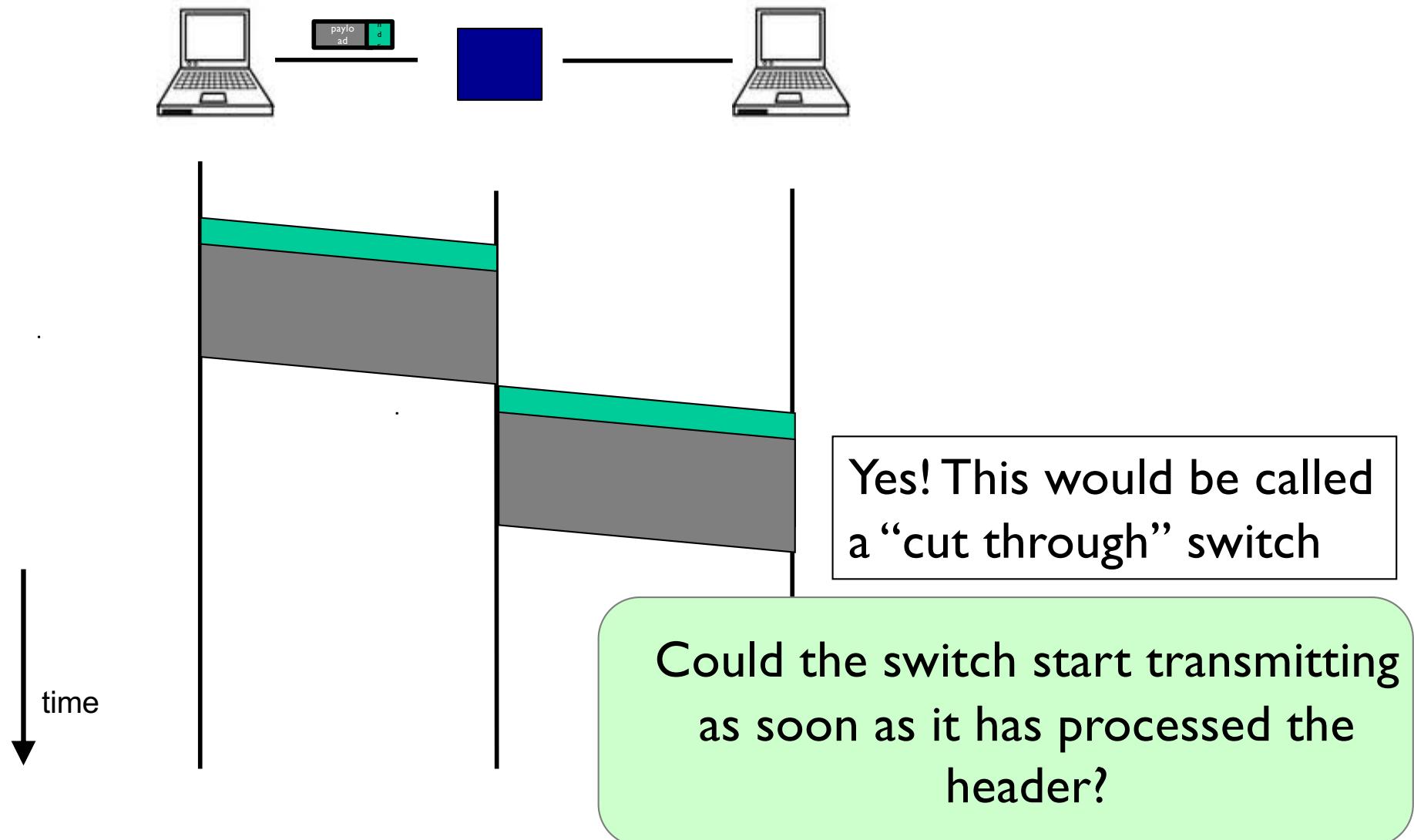


What about the time to process the packet at the switch?  
• We'll assume it's relatively negligible (mostly true)

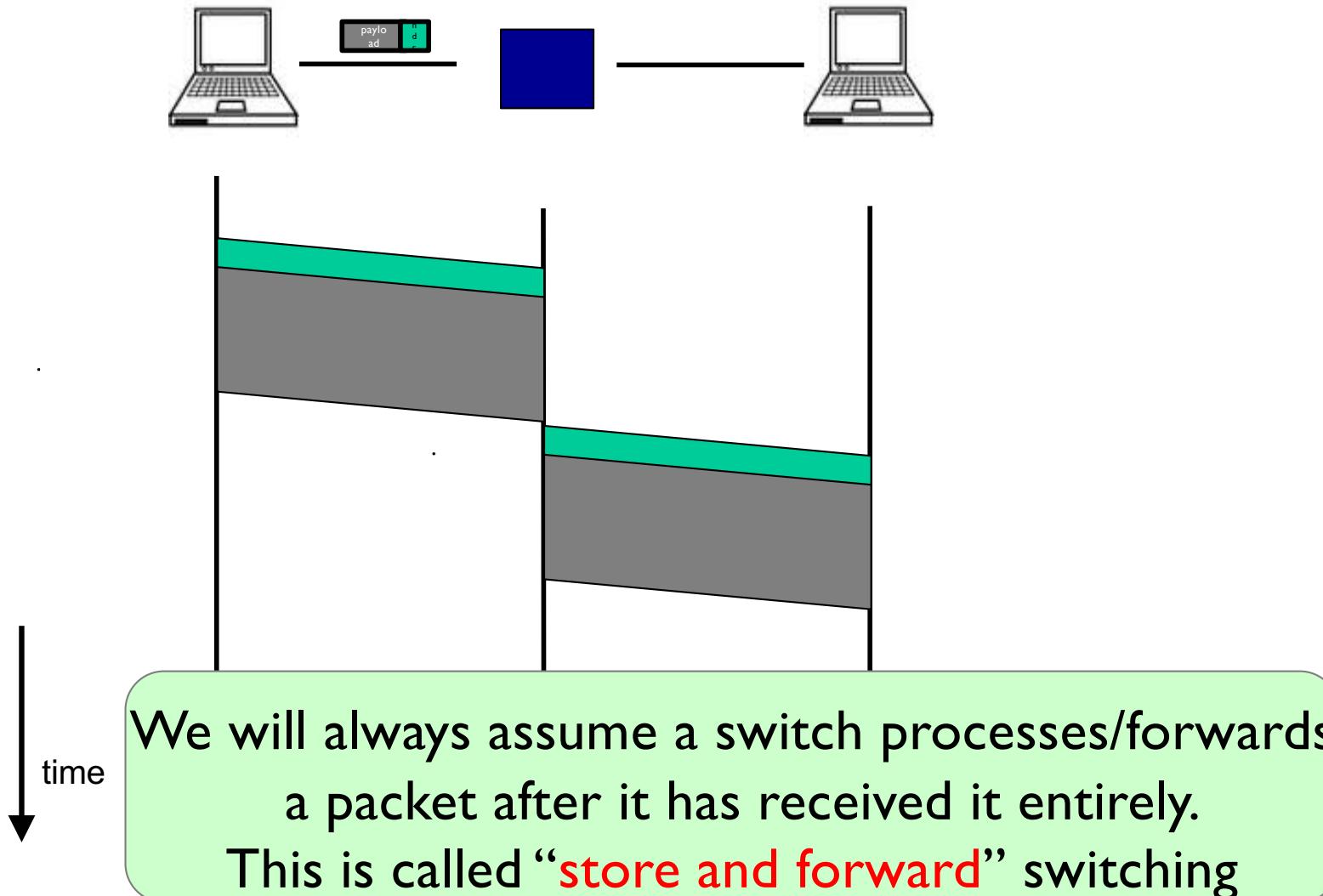
# Timing in Packet Switching



# 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

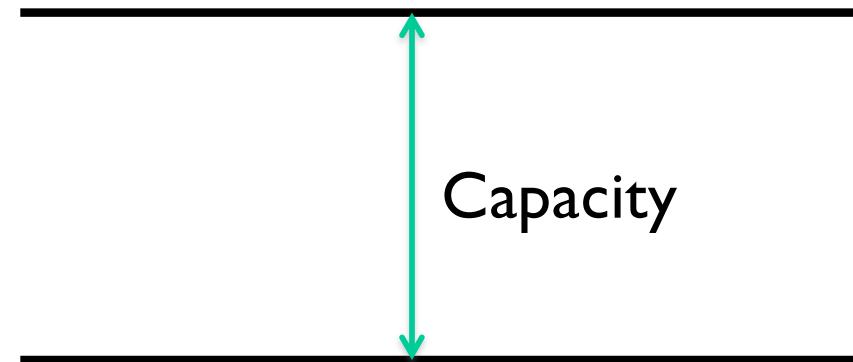
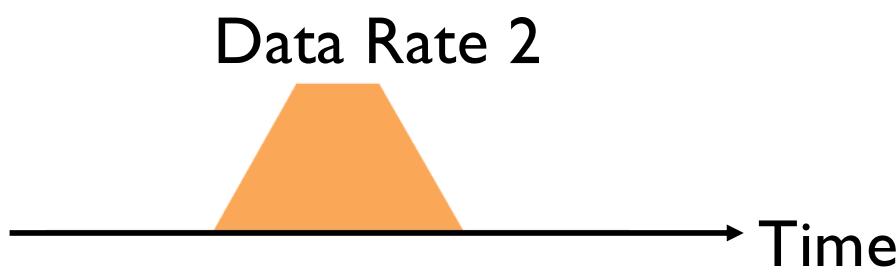
# Packet Switching

- ❖ 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”

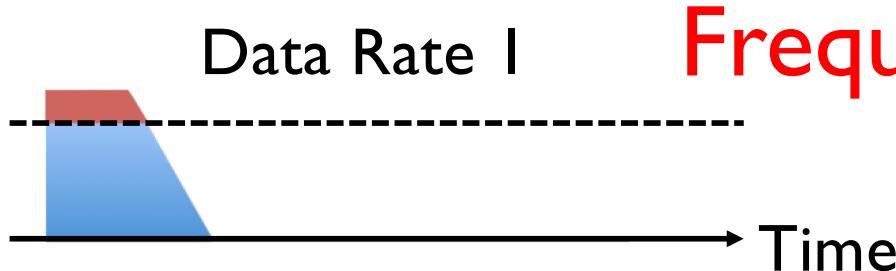
# Packet Switching

- ❖ 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**

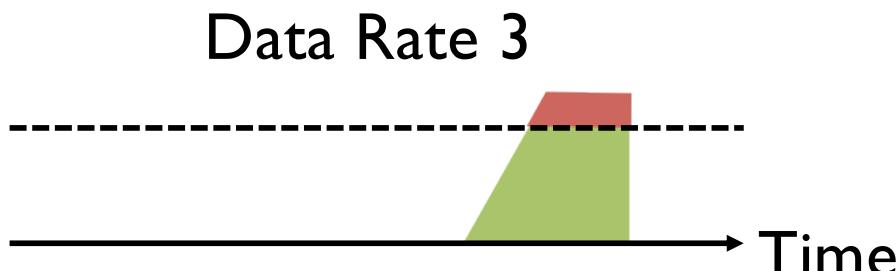
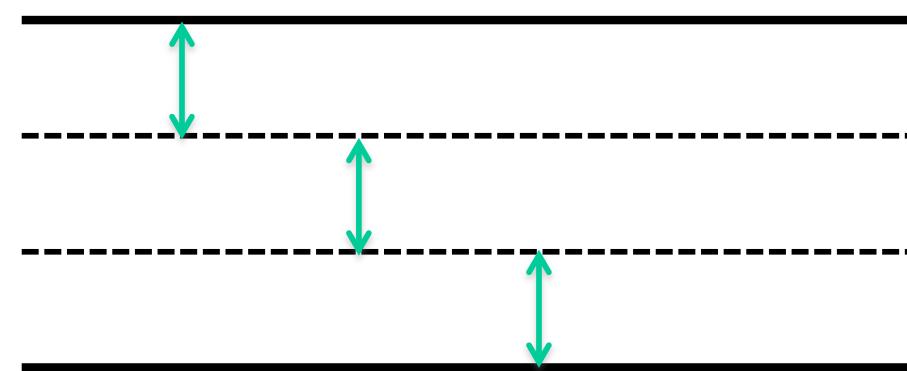
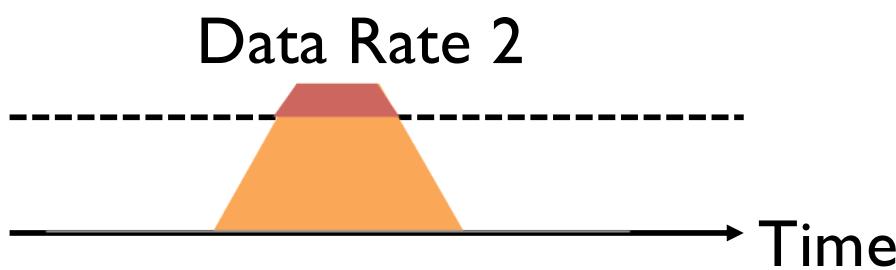
# Three Flows with Bursty Traffic



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



Frequent Overloading



# When Flows Share Total Capacity

---



No Overloading



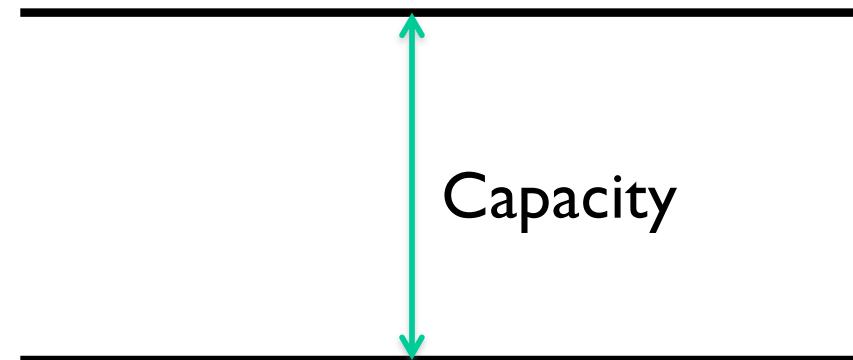
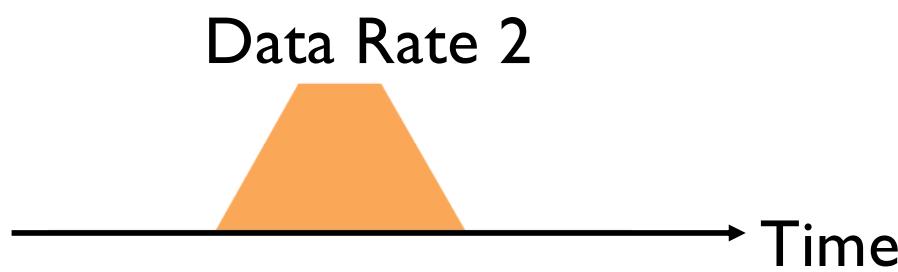
Statistical multiplexing relies on the assumption  
that not all flows burst at the same time.

Very similar to insurance, and has same failure case

A diagram illustrating a single flow's usage over time. A horizontal arrow points to the right and is labeled "Time". Above the arrow, a green shaded area represents the flow's usage. The usage starts at zero, rises to a peak, and then falls back to zero. A dashed horizontal line extends from the peak of the usage area.

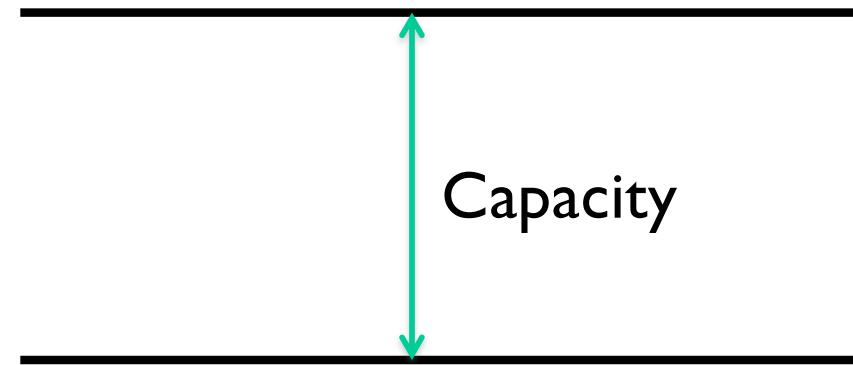
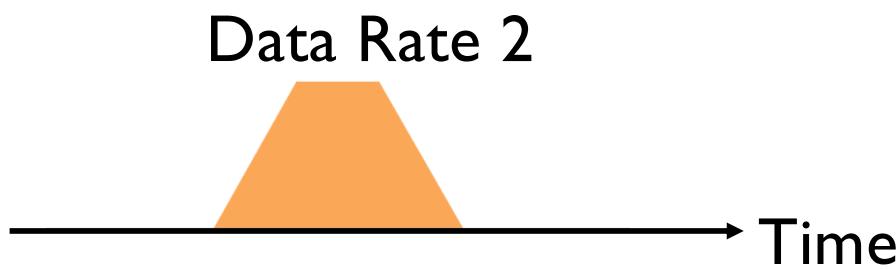
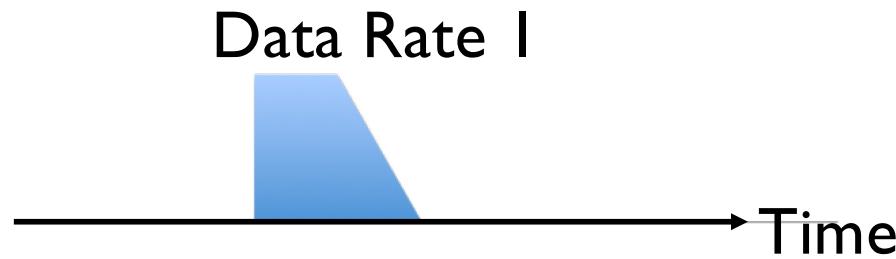
# Three Flows with Bursty Traffic

---



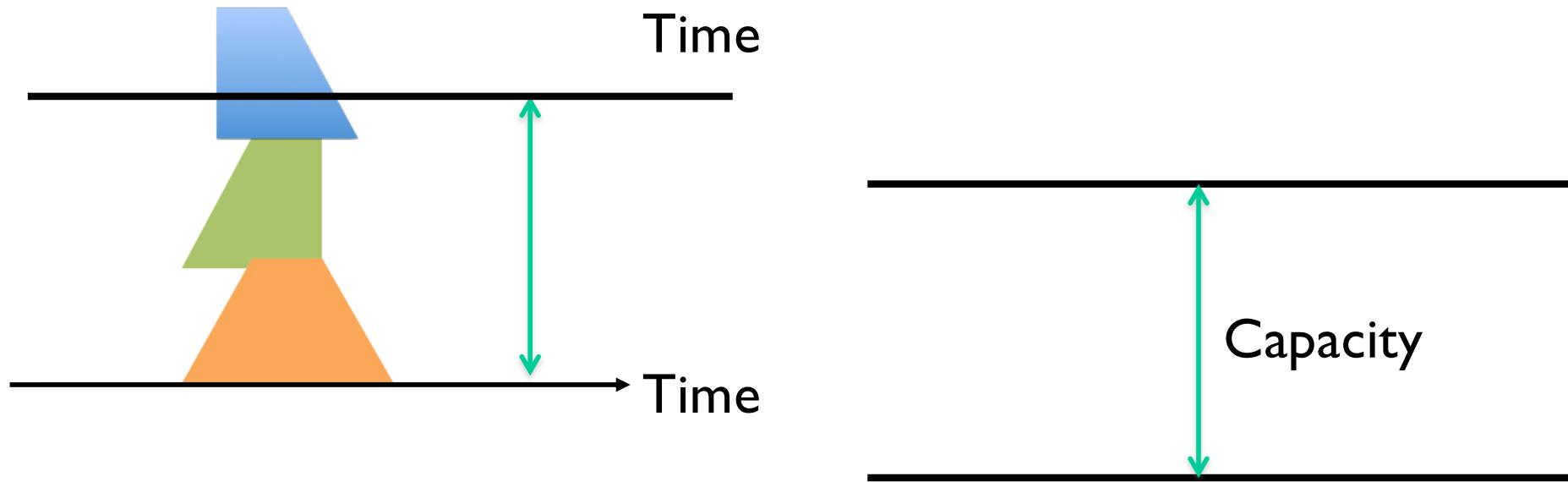
# Three Flows with Bursty Traffic

---



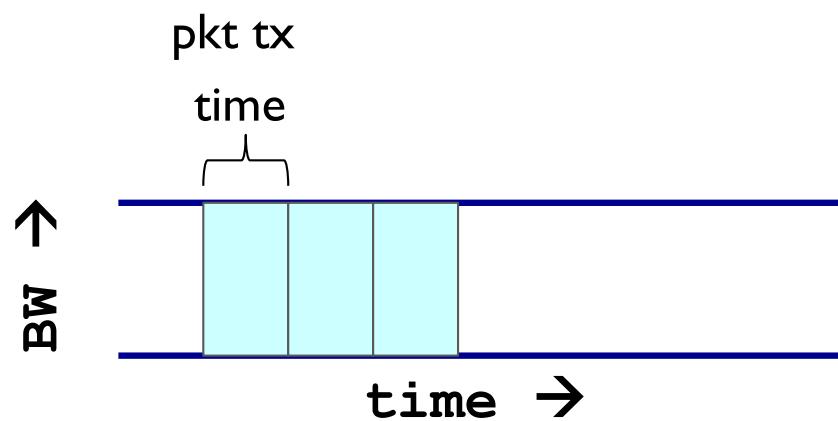
# Three Flows with Bursty Traffic

Data Rate 1+2+3 >> Capacity



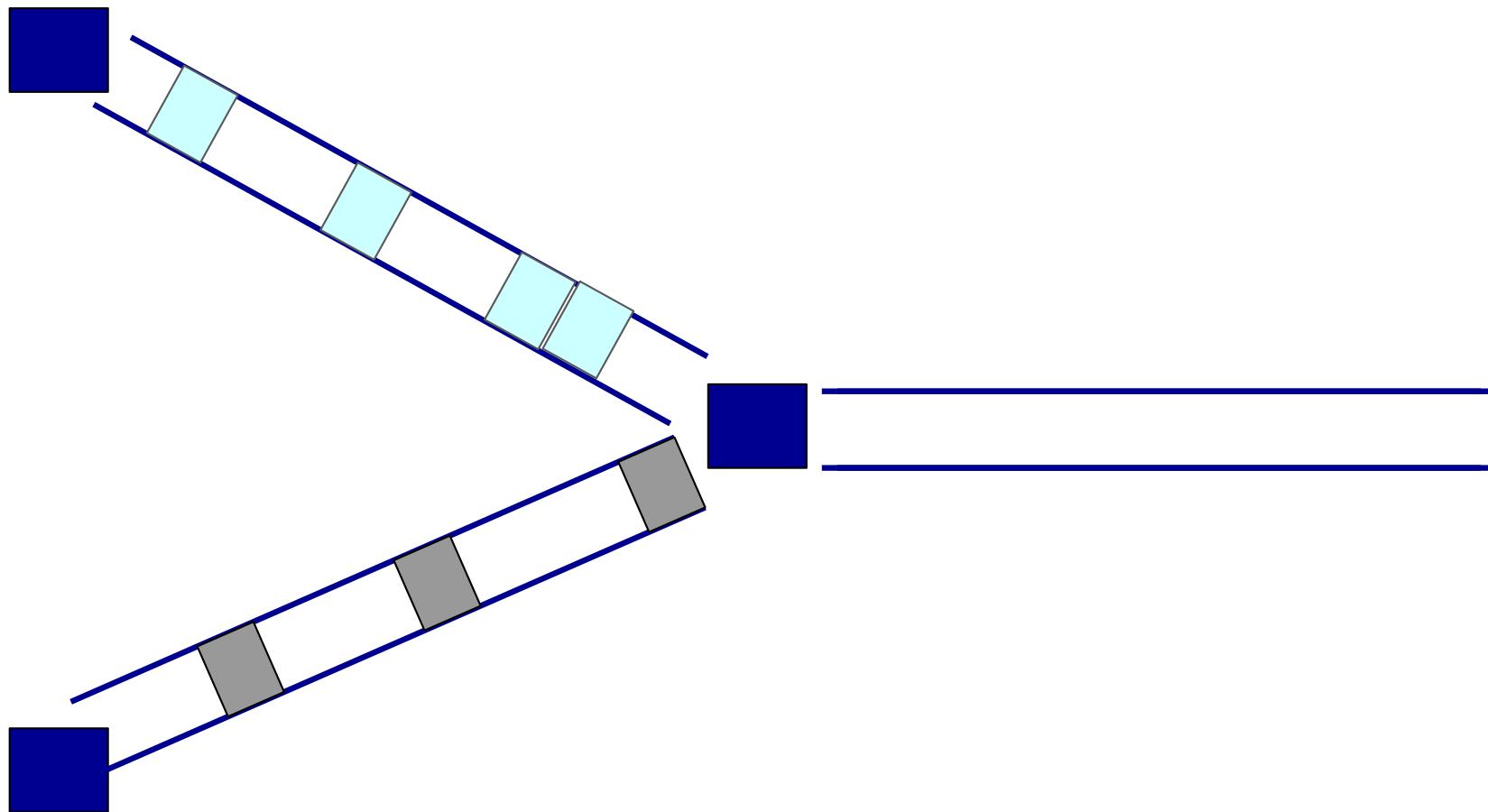
What do we do under overload?

# Statistical multiplexing: pipe view

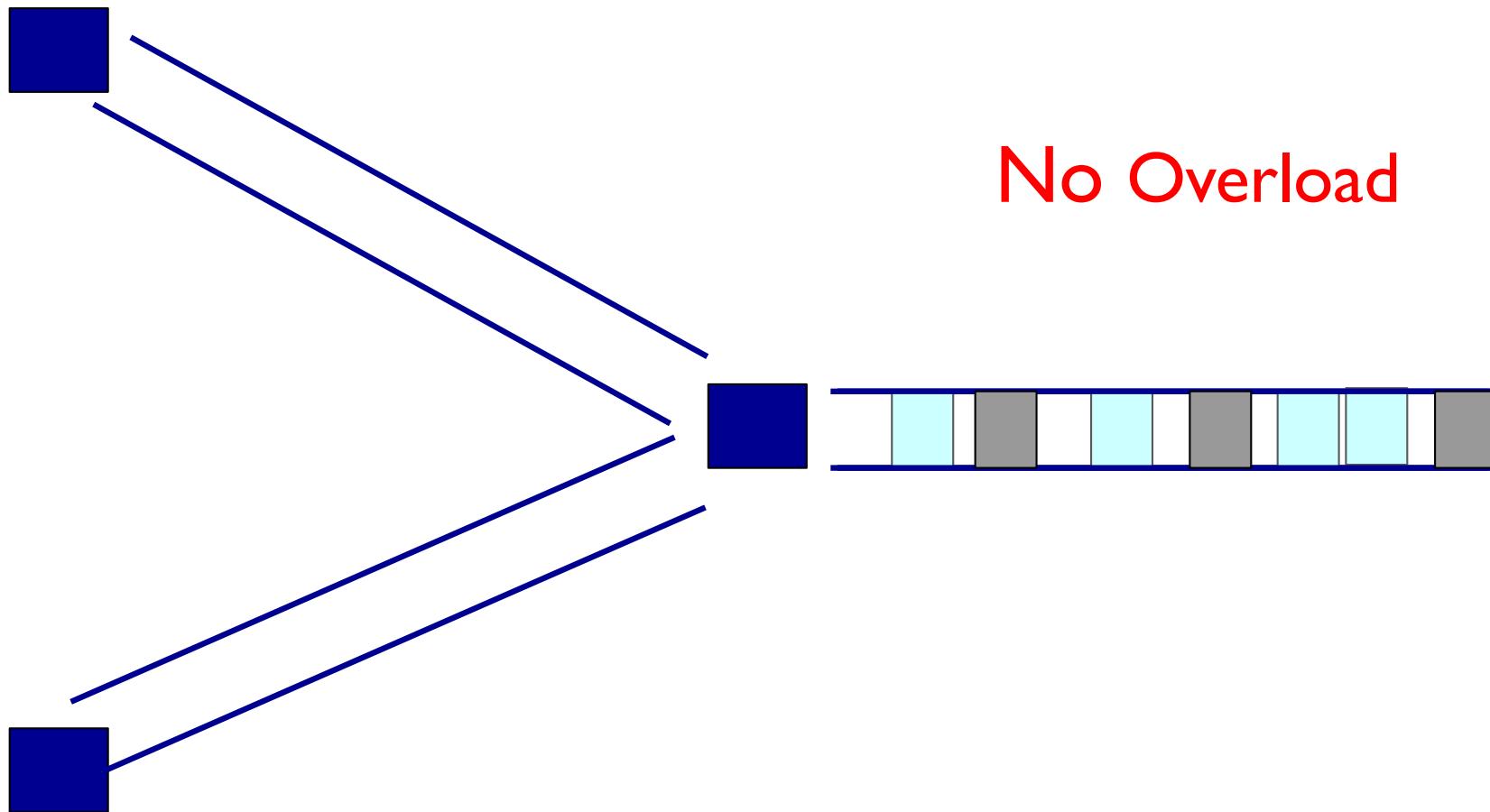


# Statistical multiplexing: pipe view

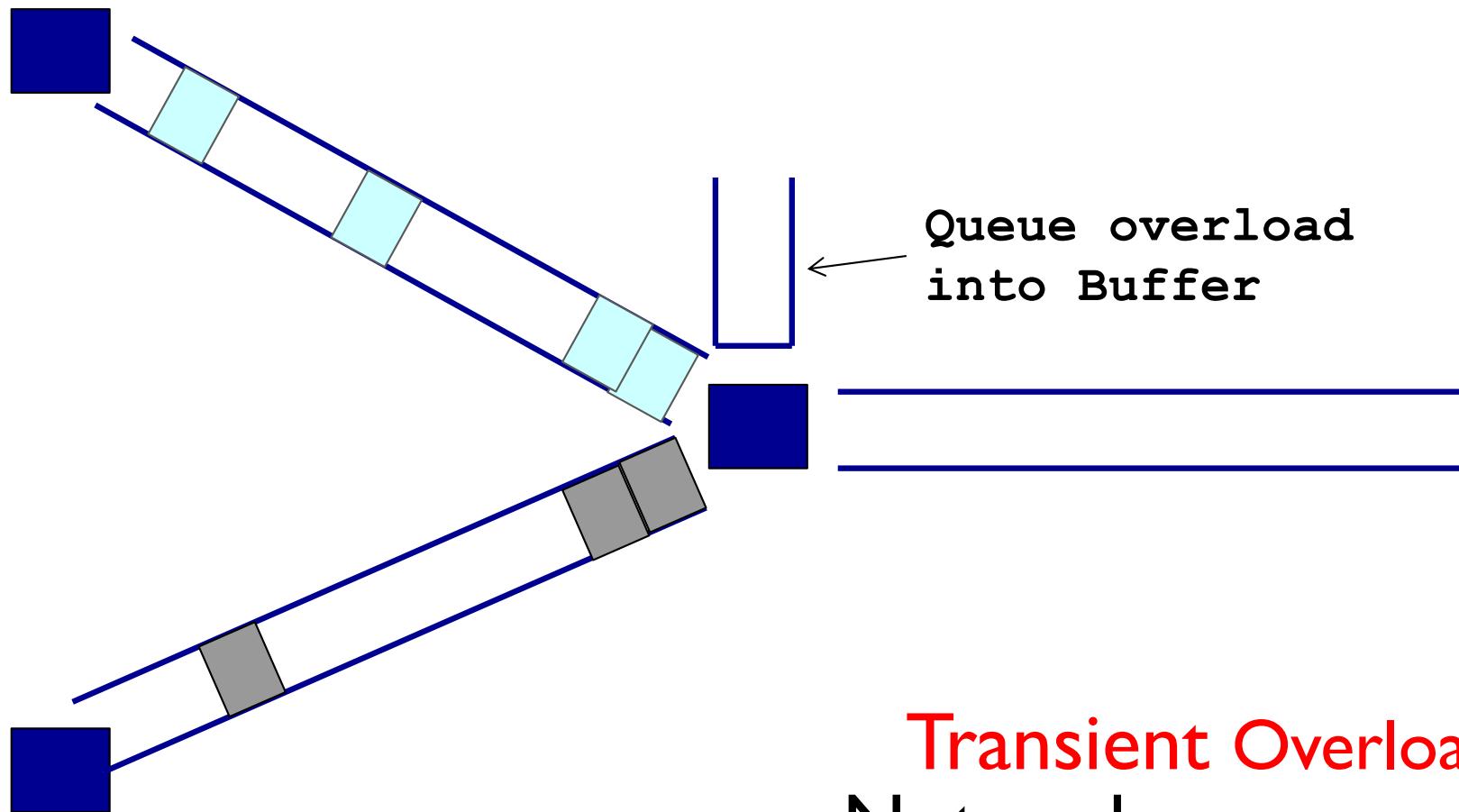
---



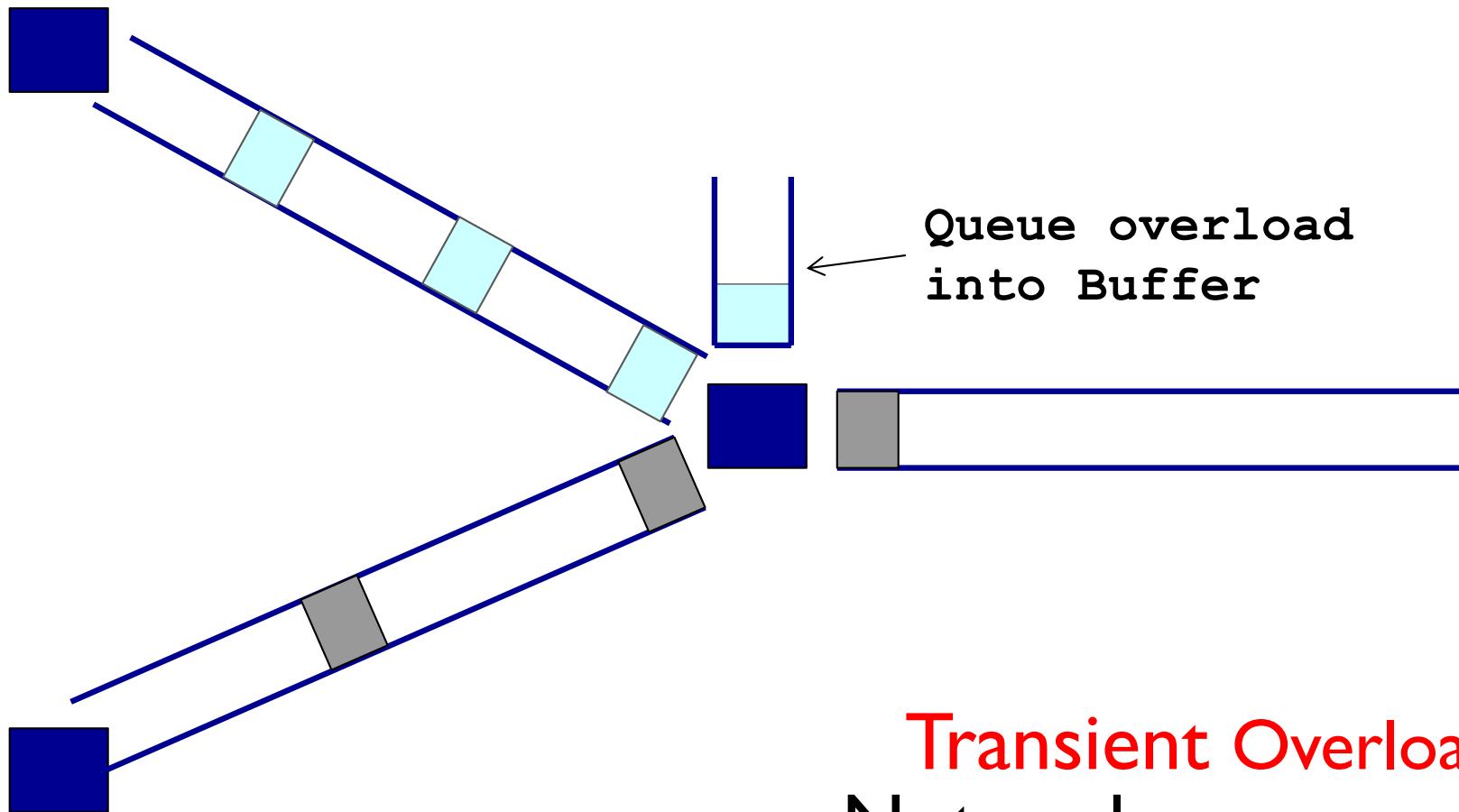
# Statistical multiplexing: pipe view



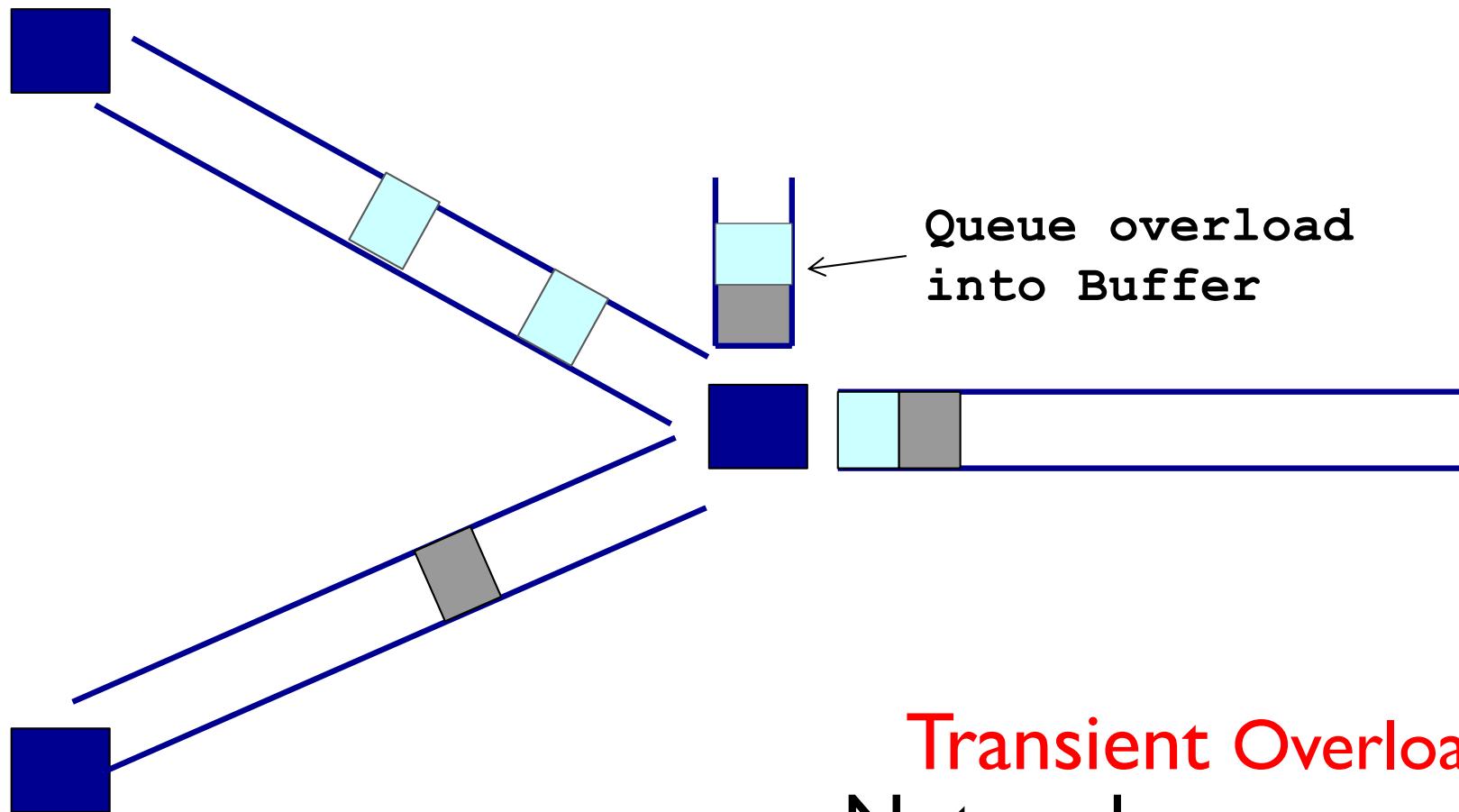
# Statistical multiplexing: pipe view



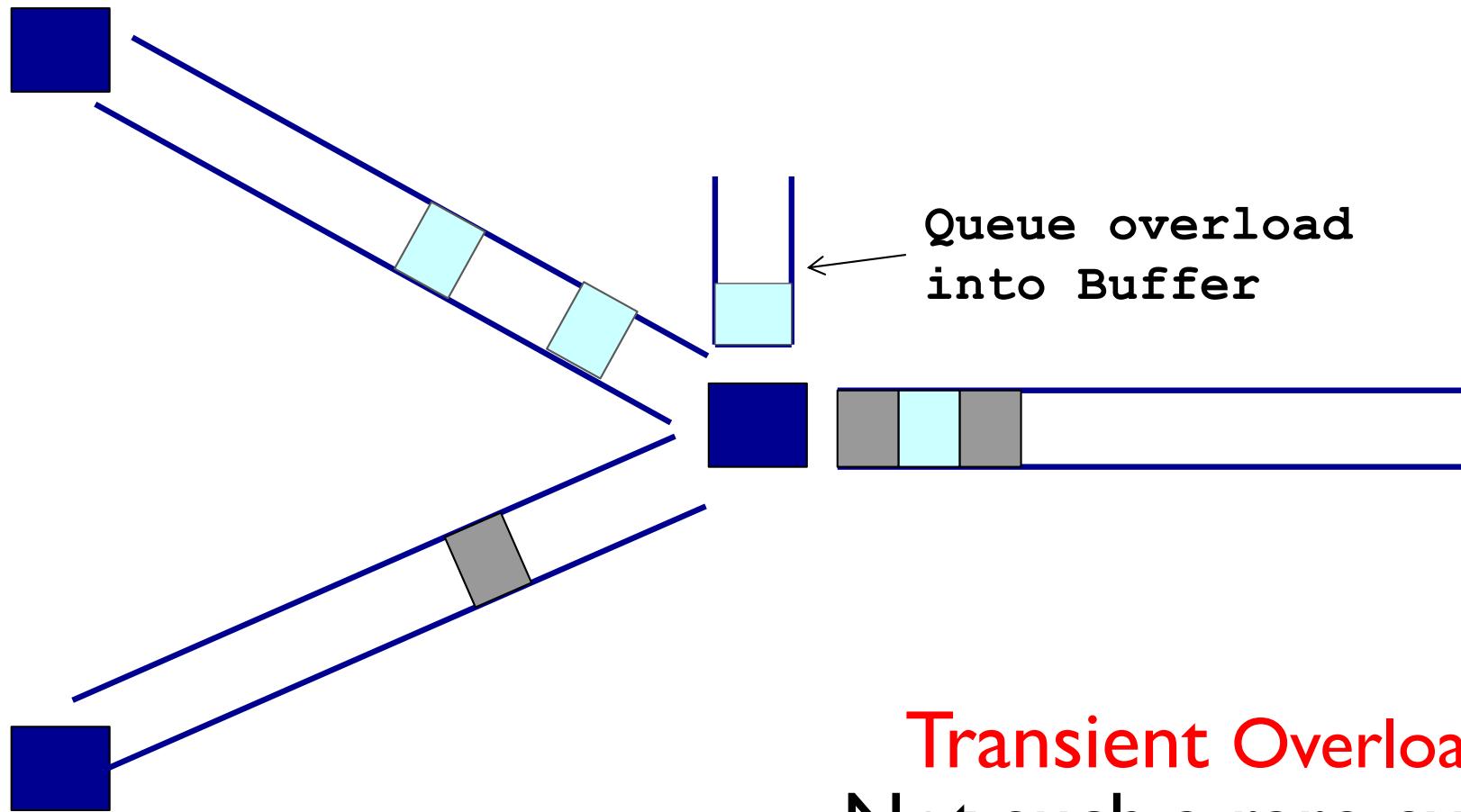
# Statistical multiplexing: pipe view



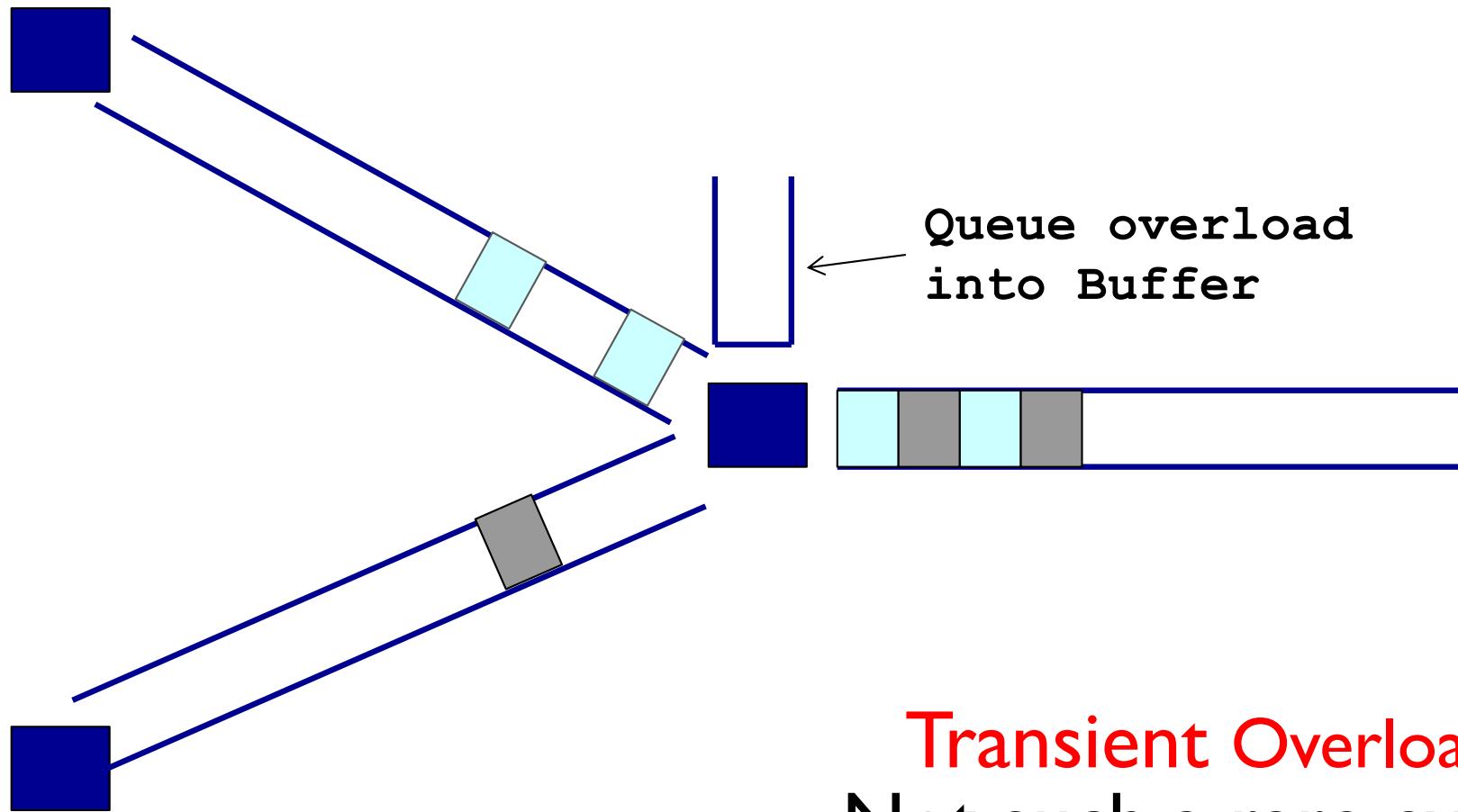
# Statistical multiplexing: pipe view



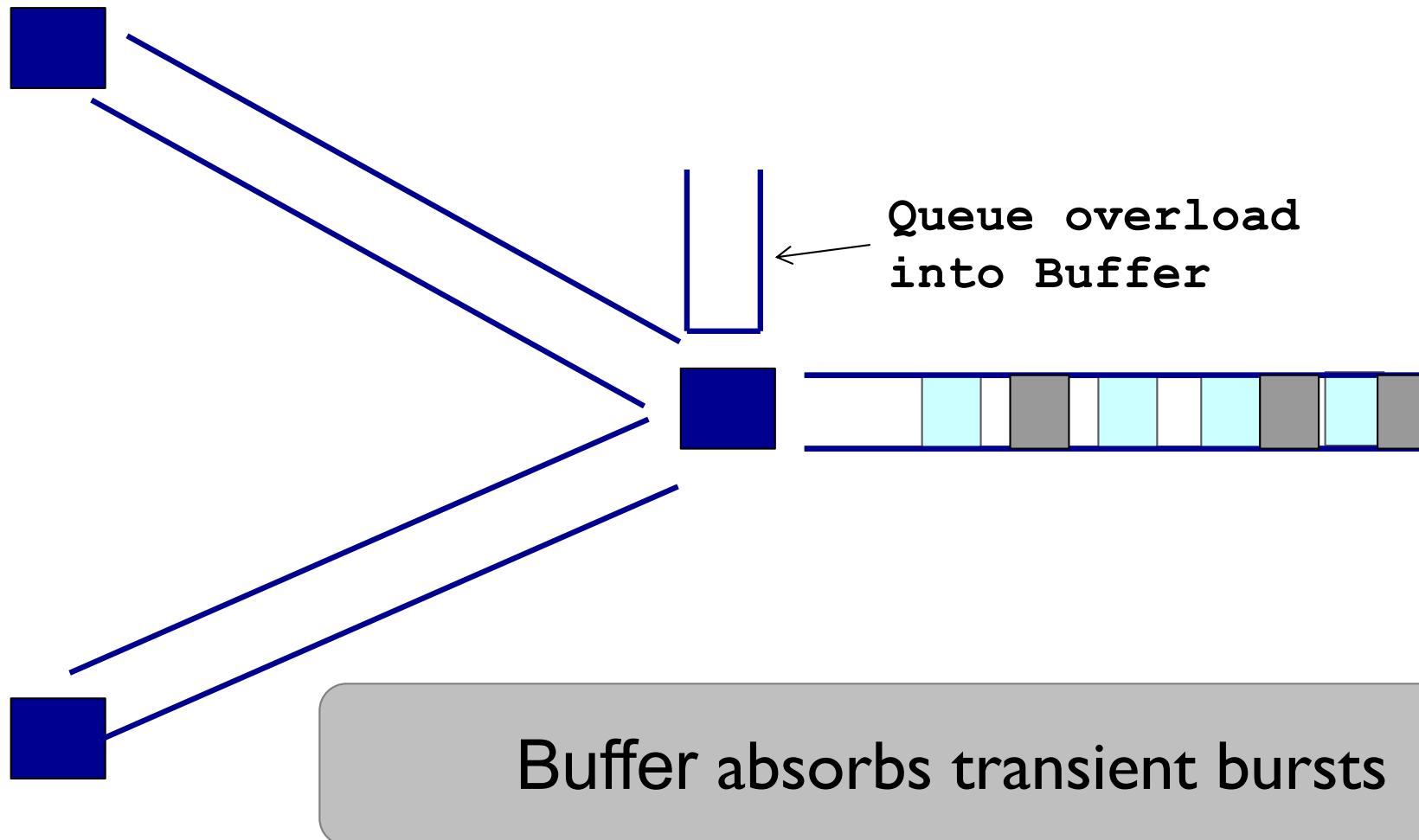
# Statistical multiplexing: pipe view



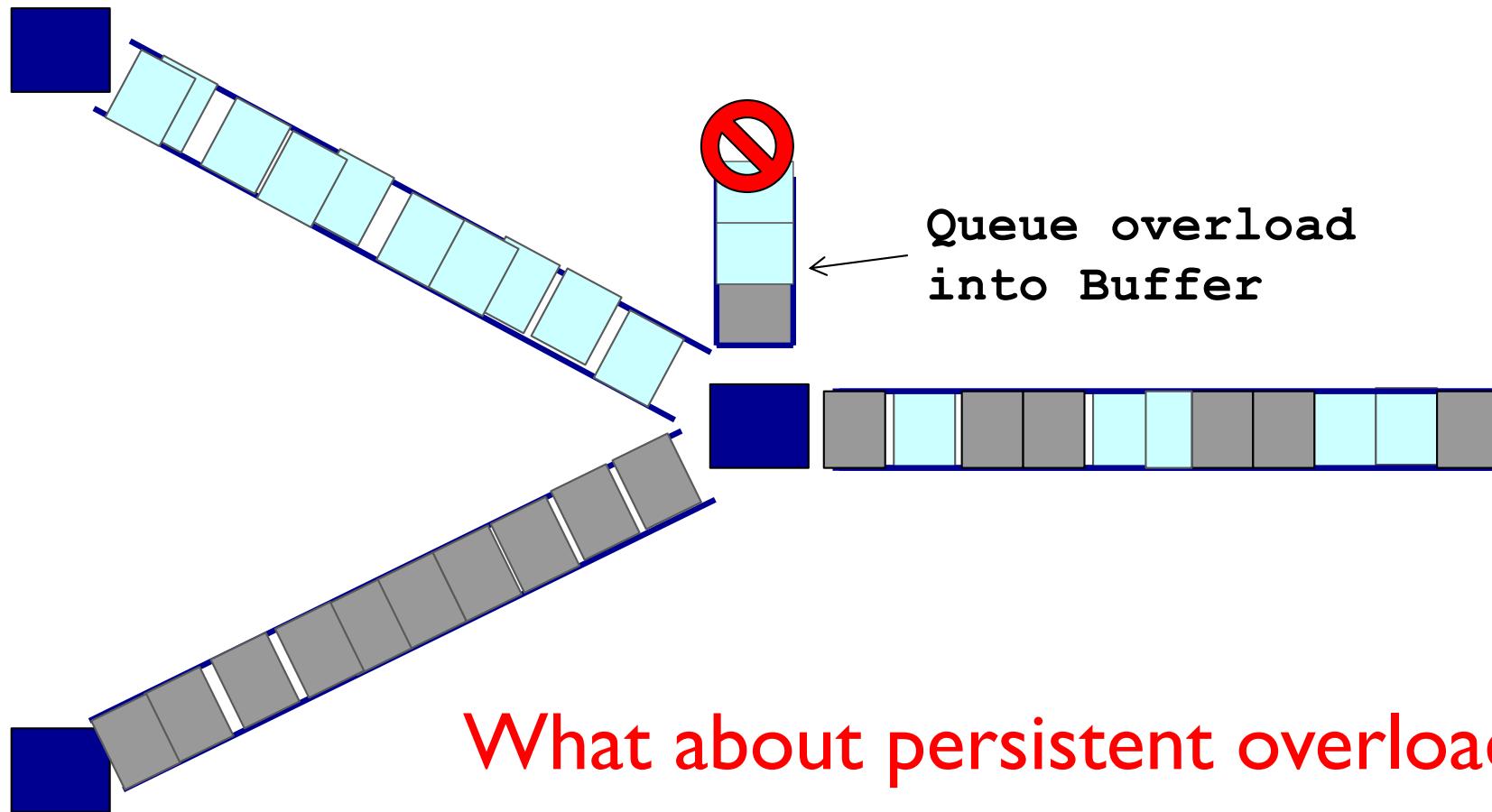
# Statistical multiplexing: pipe view



# Statistical multiplexing: pipe view



# Statistical multiplexing: pipe view





## **Quiz: What are the pros and cons of packet switching? Let's discuss ..**

- ❖ Pros:

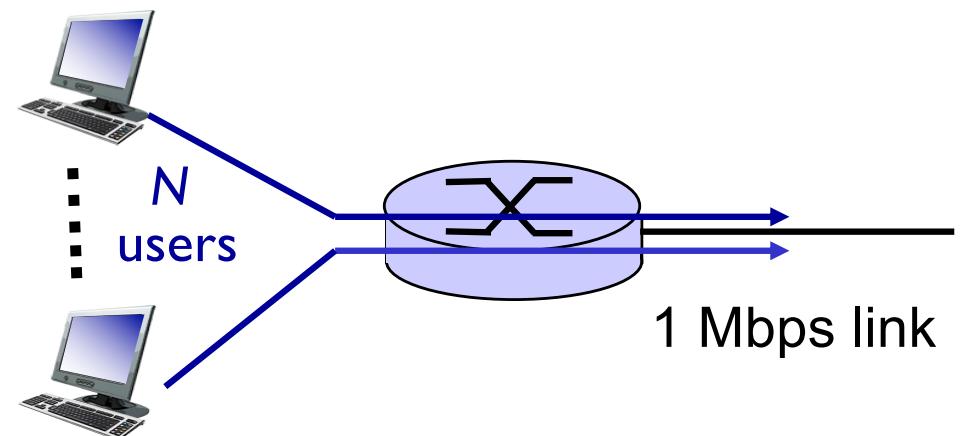
- ❖ Cons:

# 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**

# Probability Basics

In general, if the random variable  $X$  follows the binomial distribution with parameters  $n \in \mathbb{N}$  and  $p \in [0,1]$ , we write  $X \sim B(n, p)$ . The probability of getting exactly  $k$  successes in  $n$  trials is given by the **probability mass function**:

$$f(k, n, p) = \Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

for  $k = 0, 1, 2, \dots, n$ , where

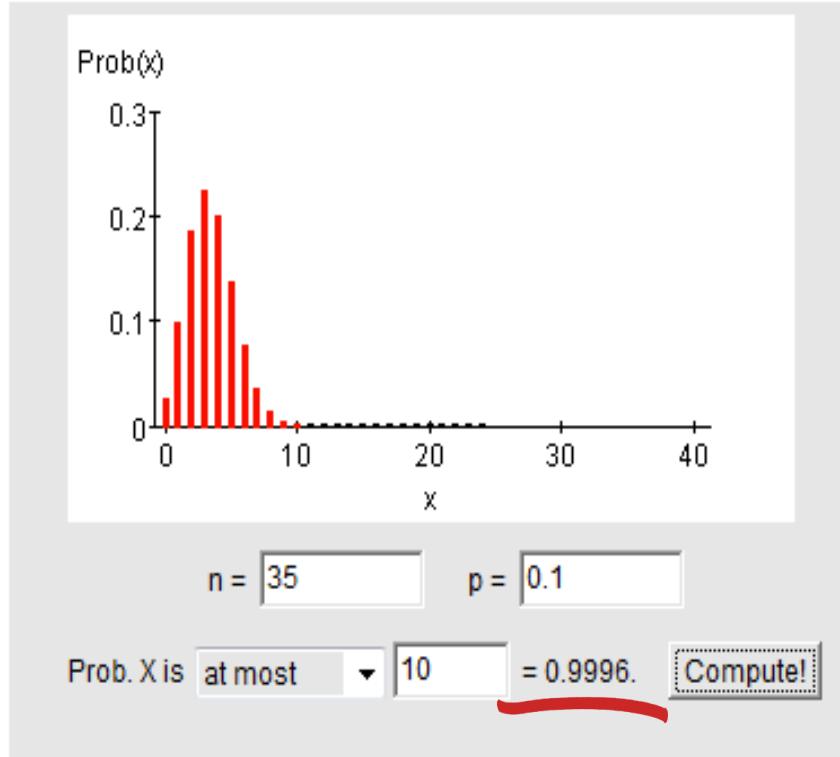
$$\binom{n}{k} = \frac{n!}{k!(n - k)!}$$

The **cumulative distribution function** can be expressed as:

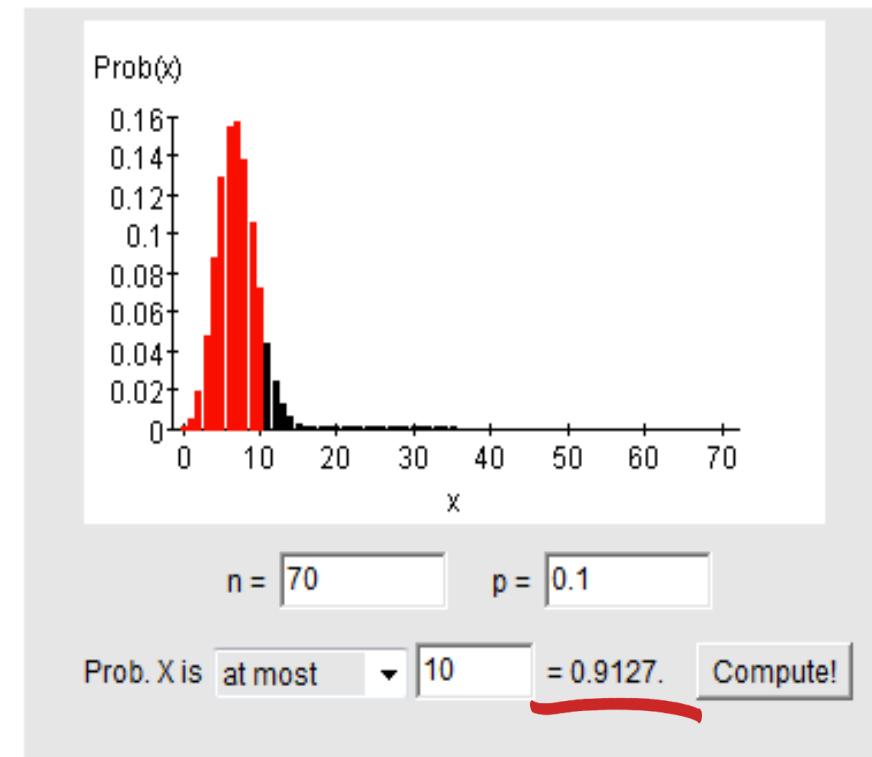
$$F(k; n, p) = \Pr(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1 - p)^{n-i}$$

# Statistical Multiplexing Gain (SMG)

Binomial Calculator



Binomial Calculator



$$\text{SMG: } 35/10 = 3.5$$

$$\text{SMG: } 70/10 = 7$$

# 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



## 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

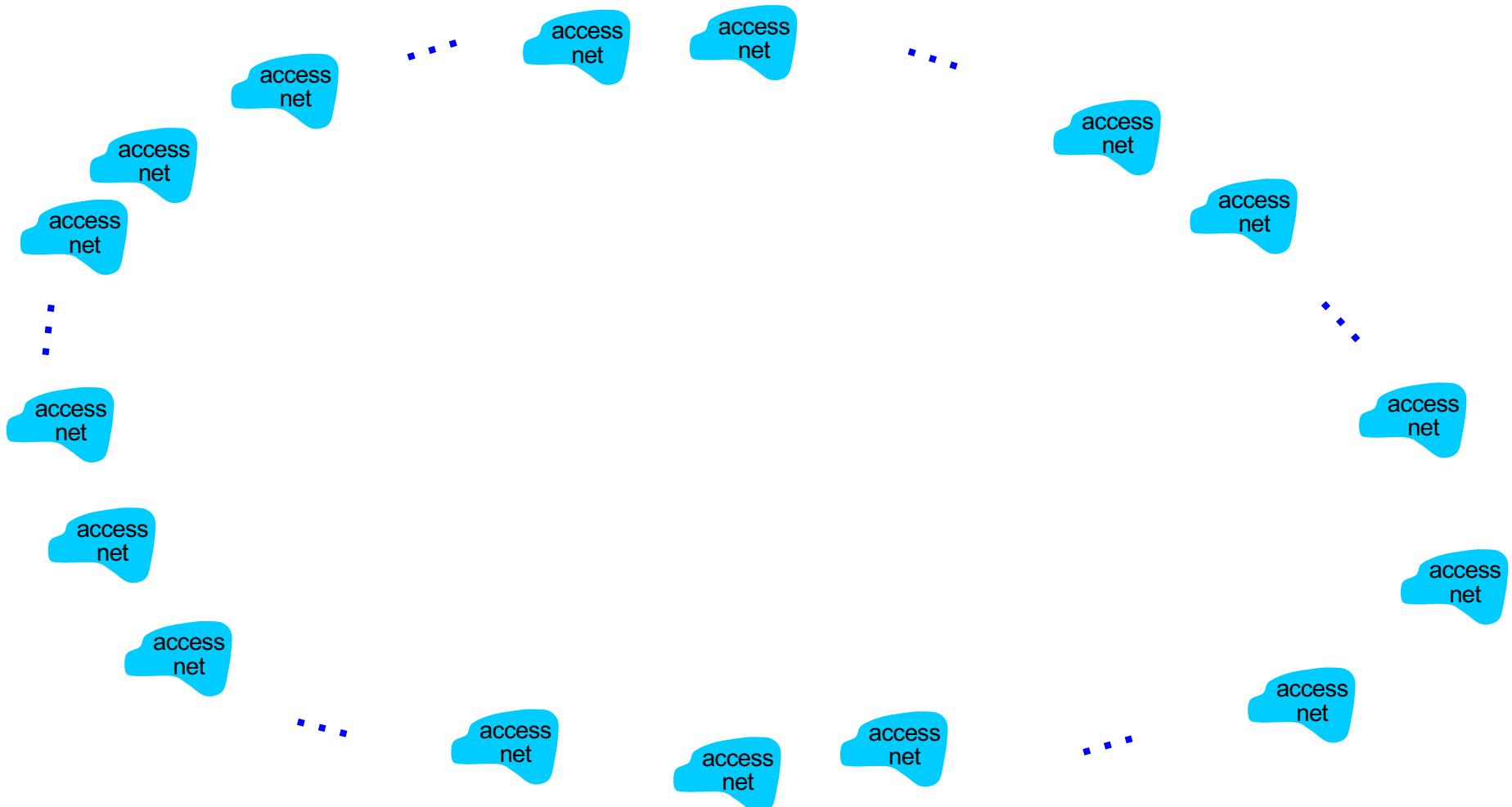
- A. is the same
- B. is independent
- C. is always different from

# 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

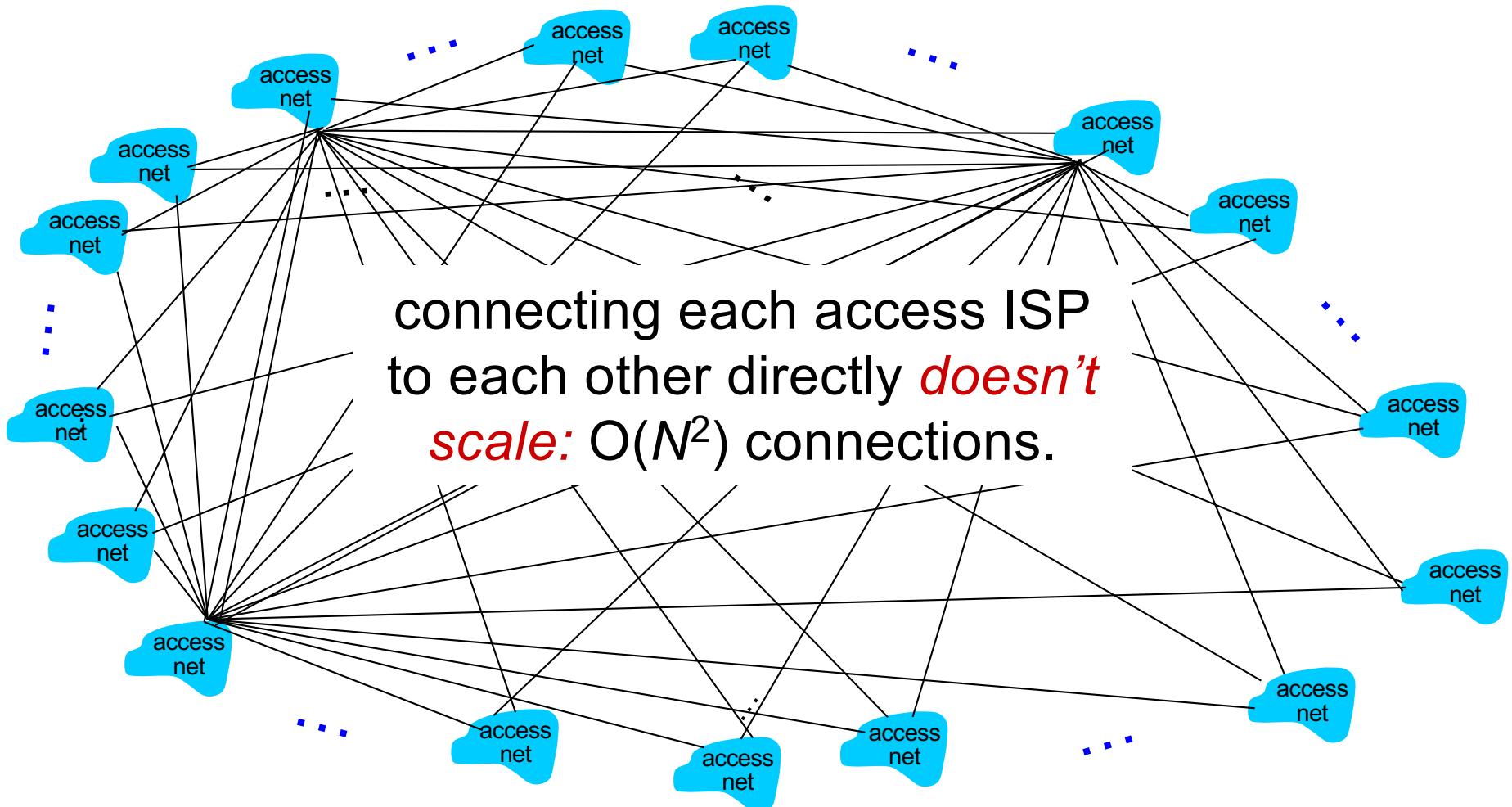
# Internet structure: network of networks

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



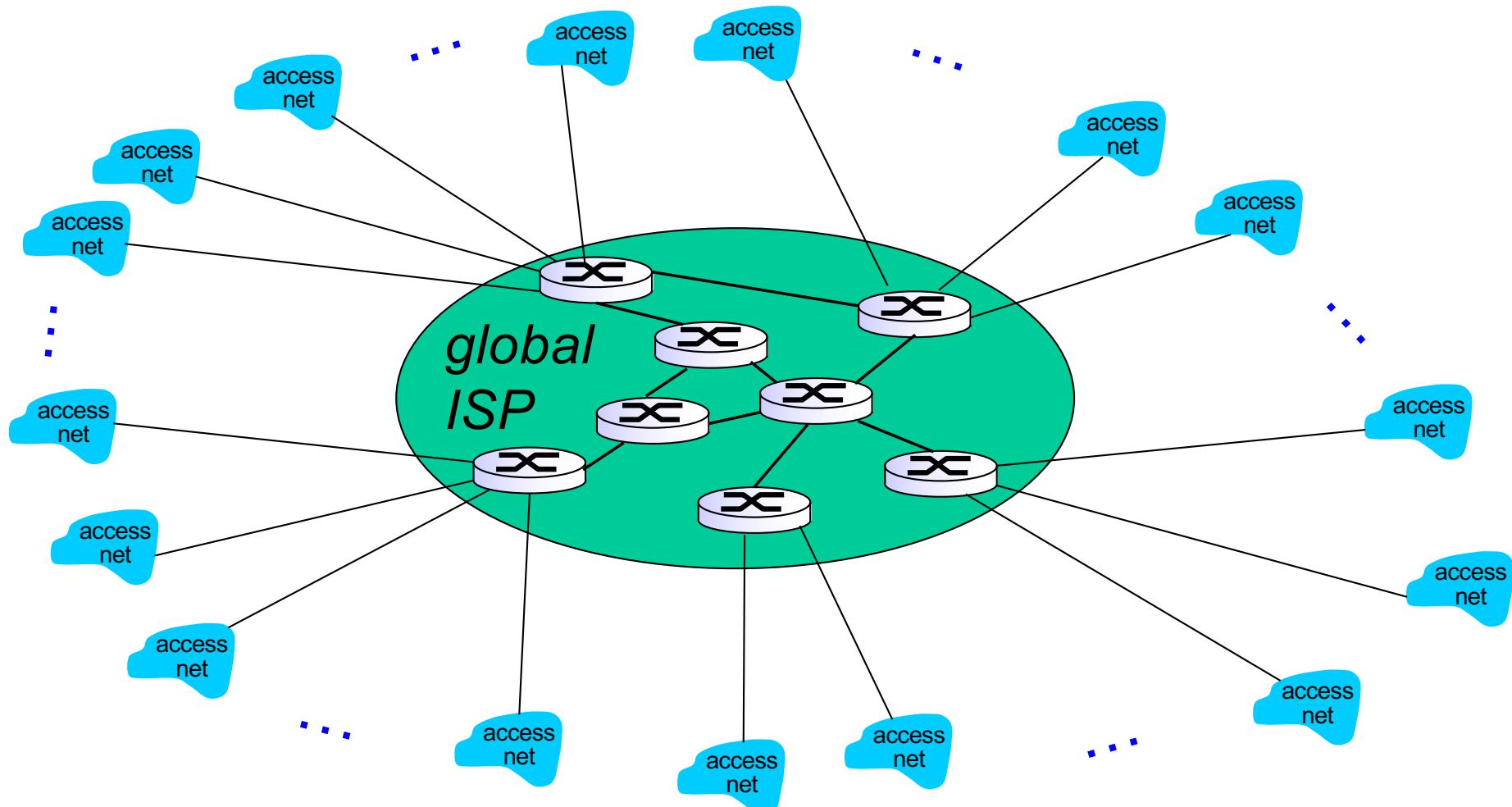
# Internet structure: network of networks

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



# Internet structure: network of networks

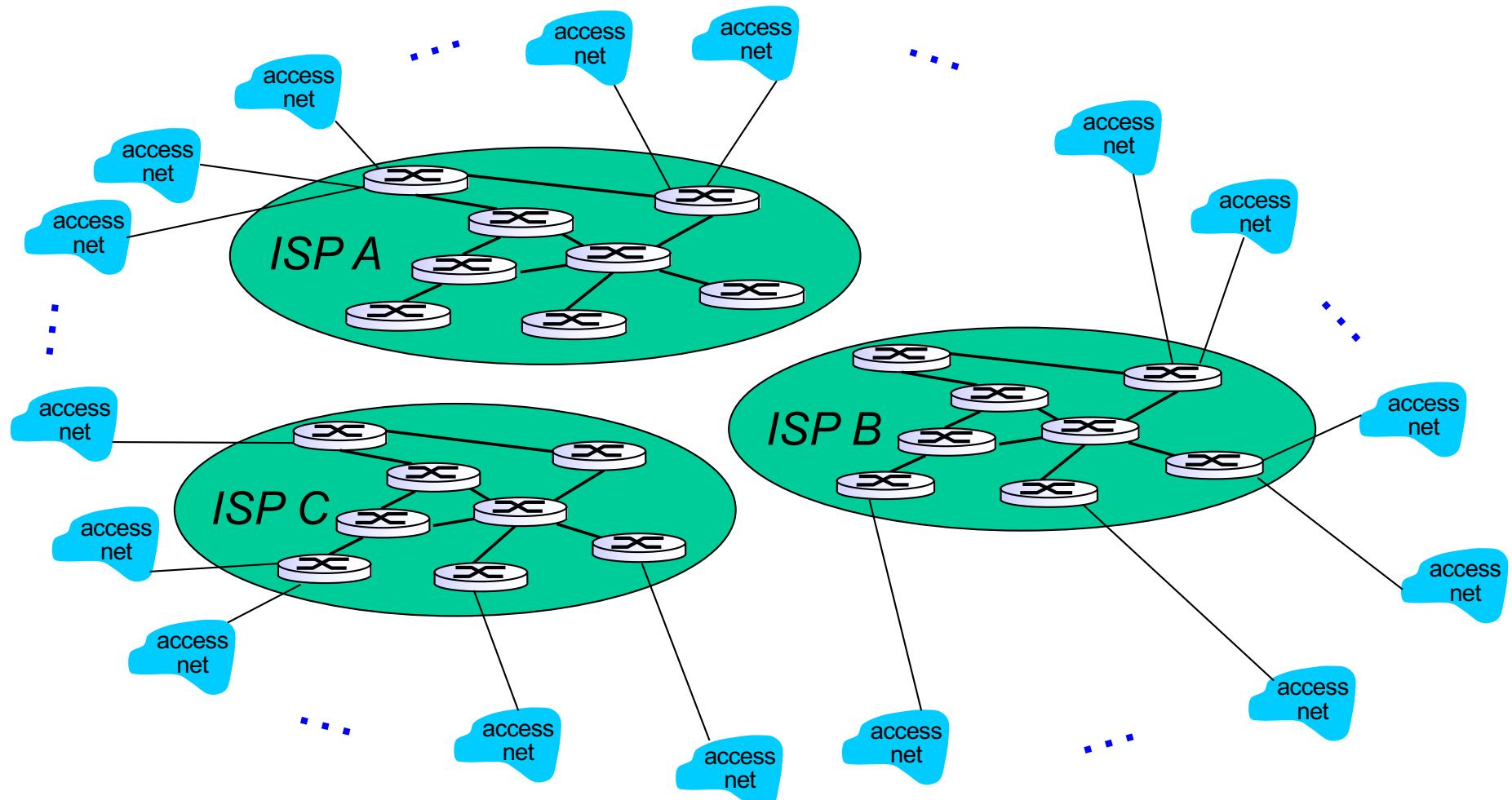
*Option: connect each access ISP to a global transit ISP? Customer and provider ISPs have economic agreement.*



# Internet structure: network of networks

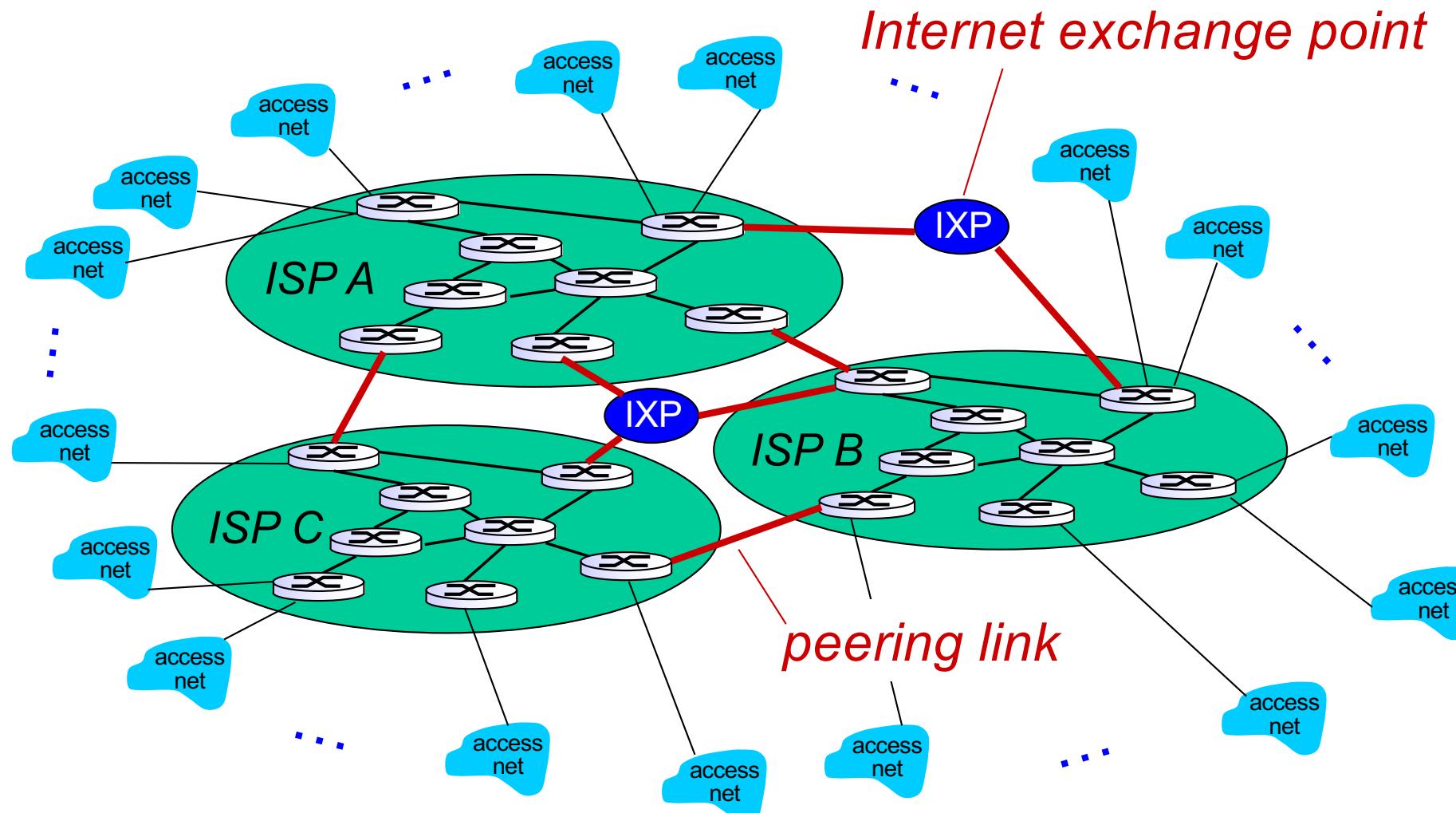
But if one global ISP is viable business, there will be competitors

....



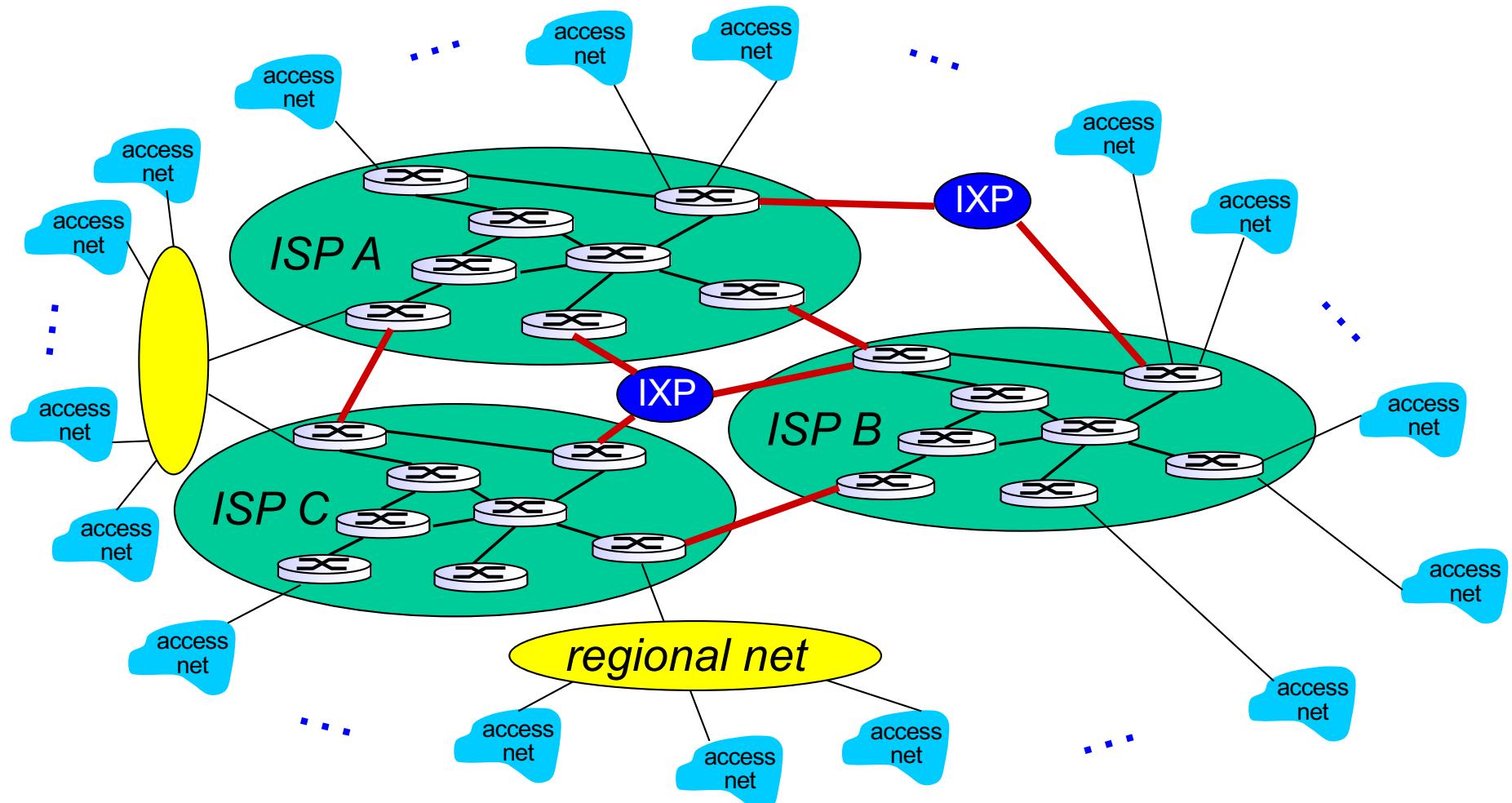
# Internet structure: network of networks

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



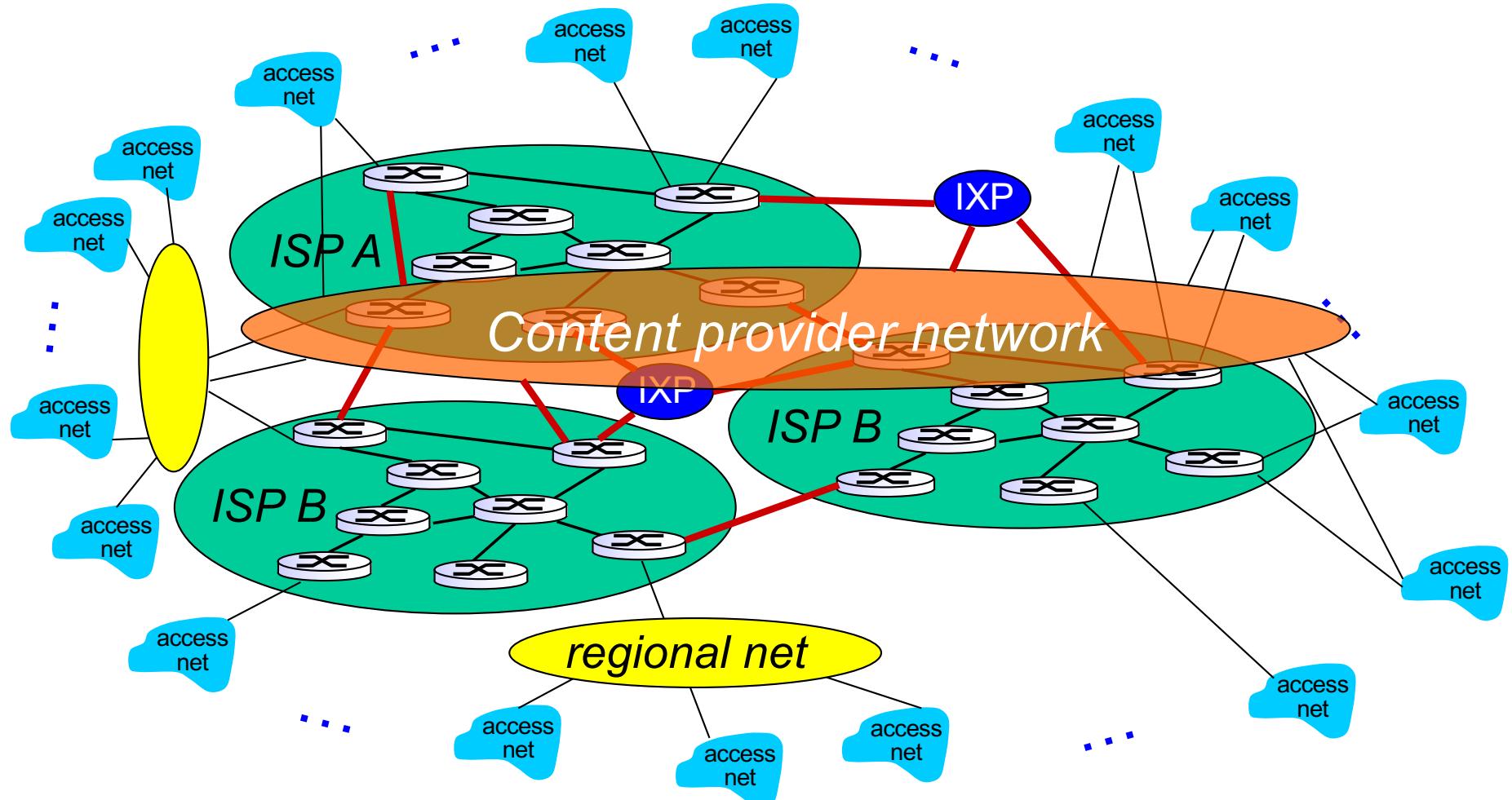
# Internet structure: network of networks

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

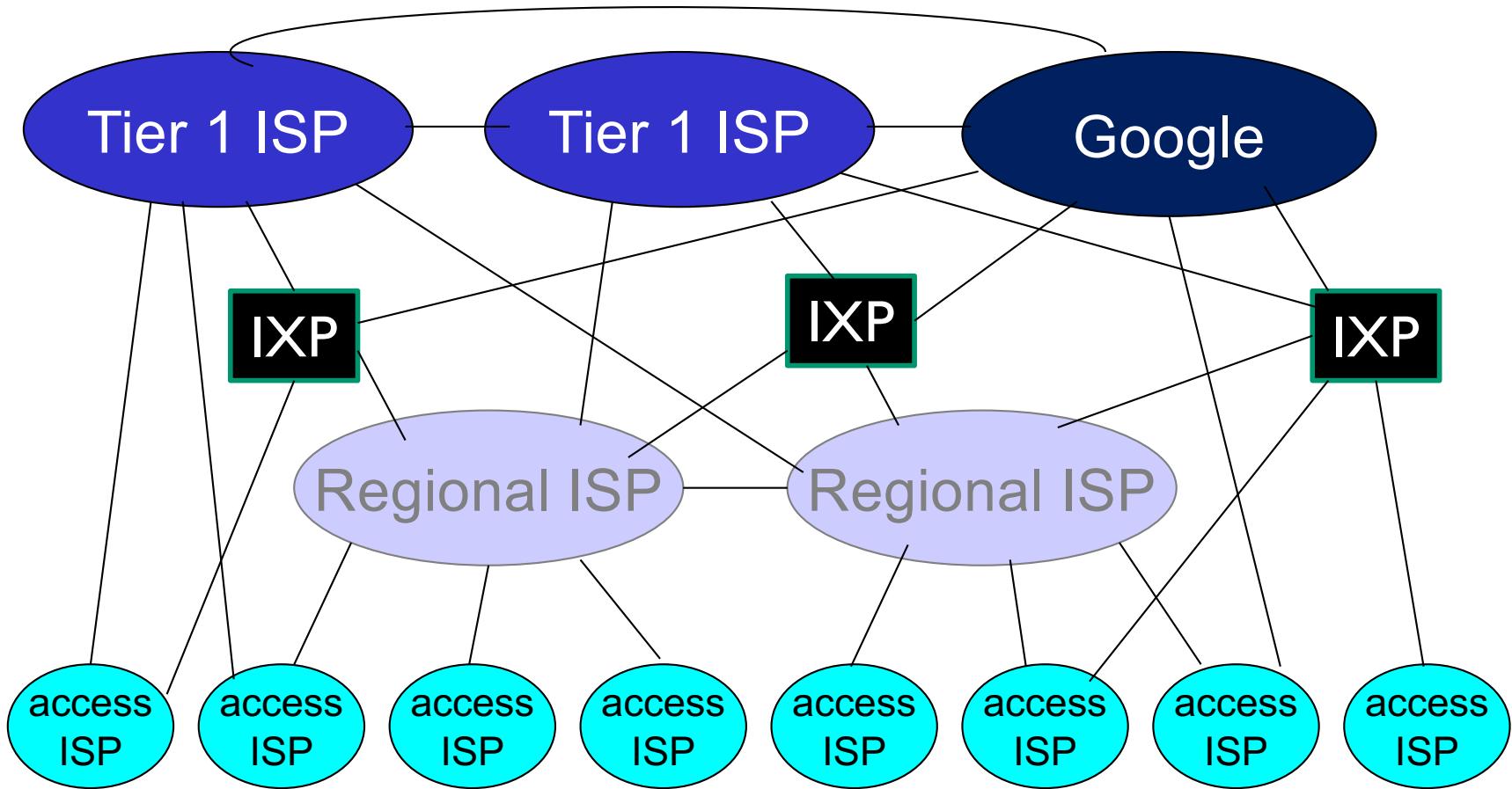


# Internet structure: network of networks

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



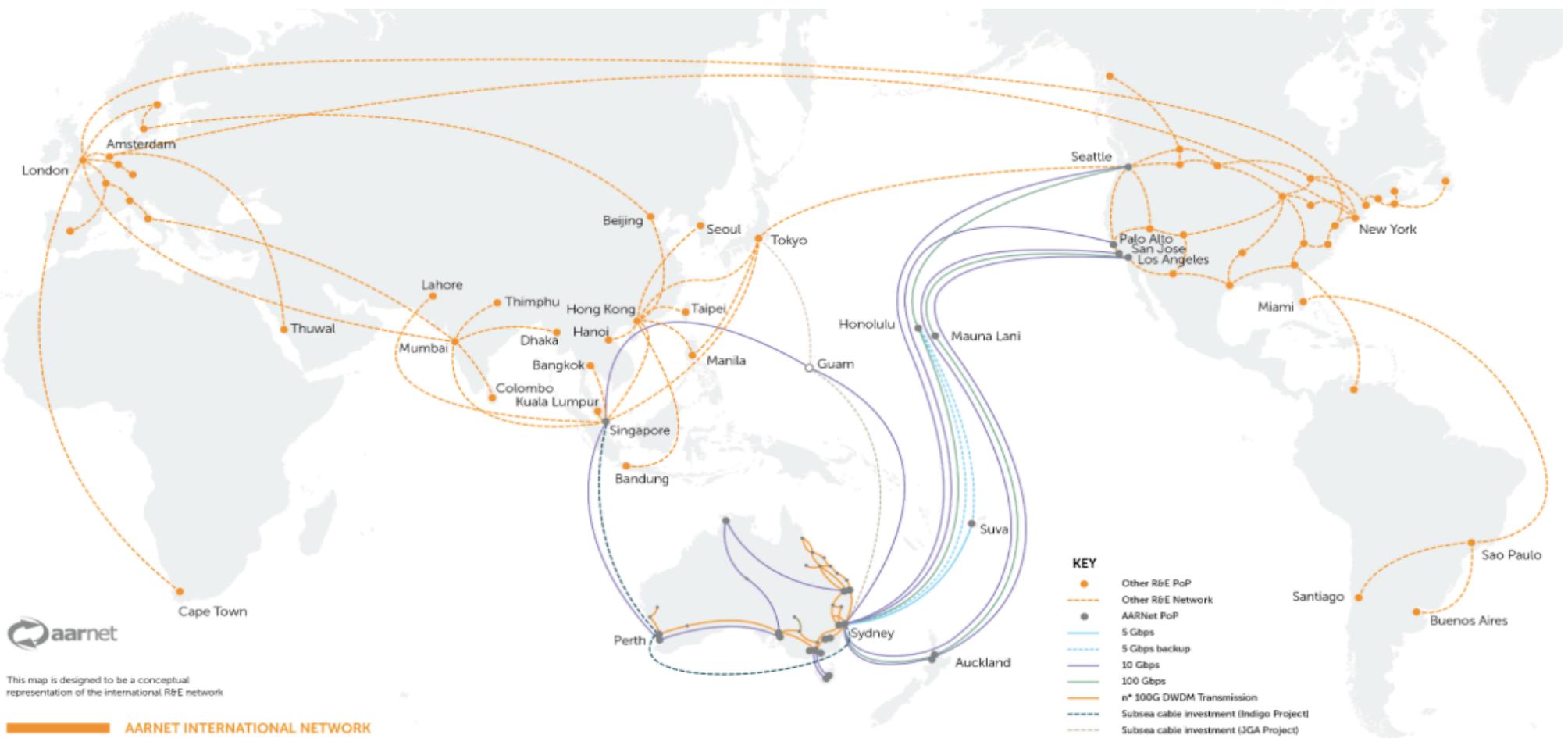
# Internet structure: network of networks



- ❖ 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

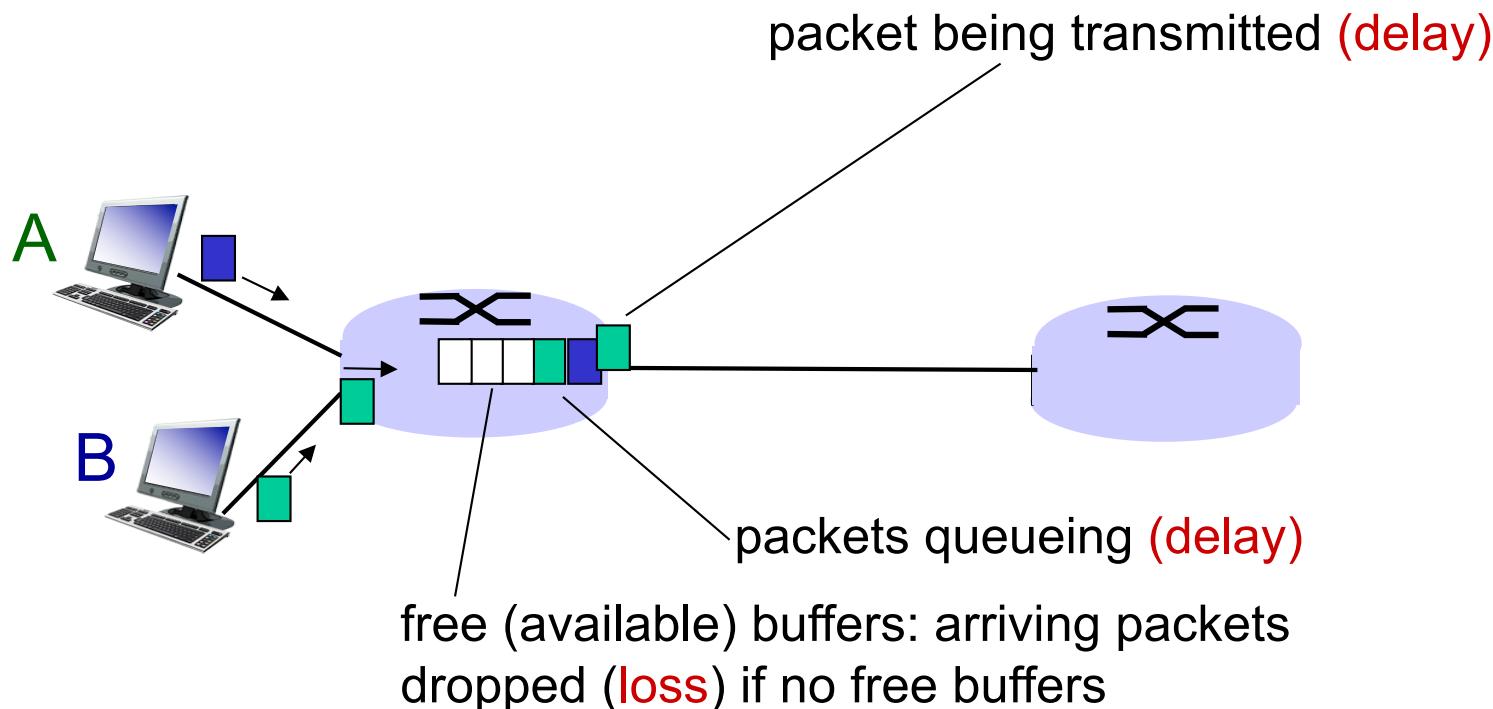
I.6 networks under attack: security

I.7 history

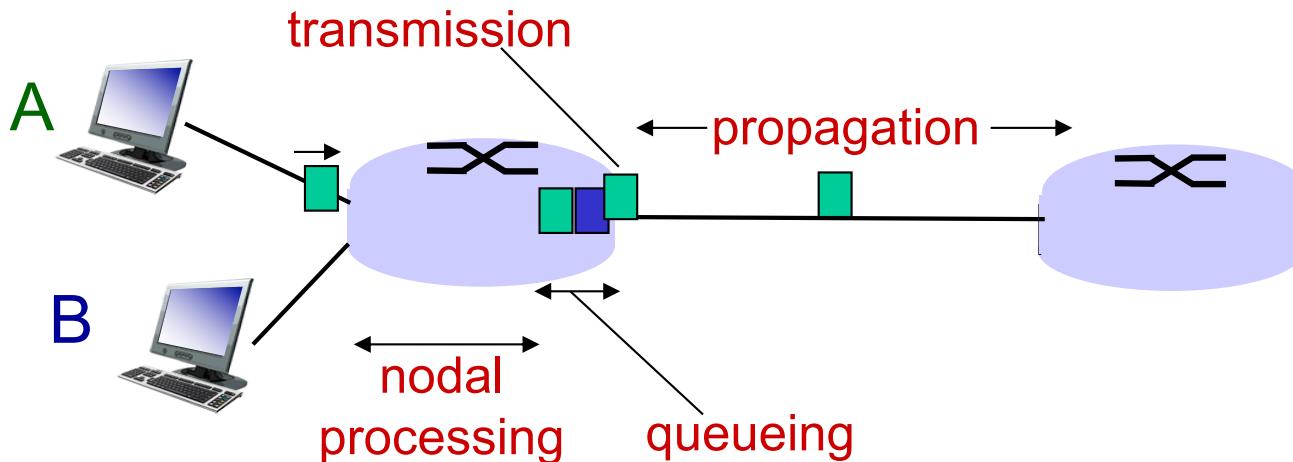
# How do loss and delay occur?

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



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

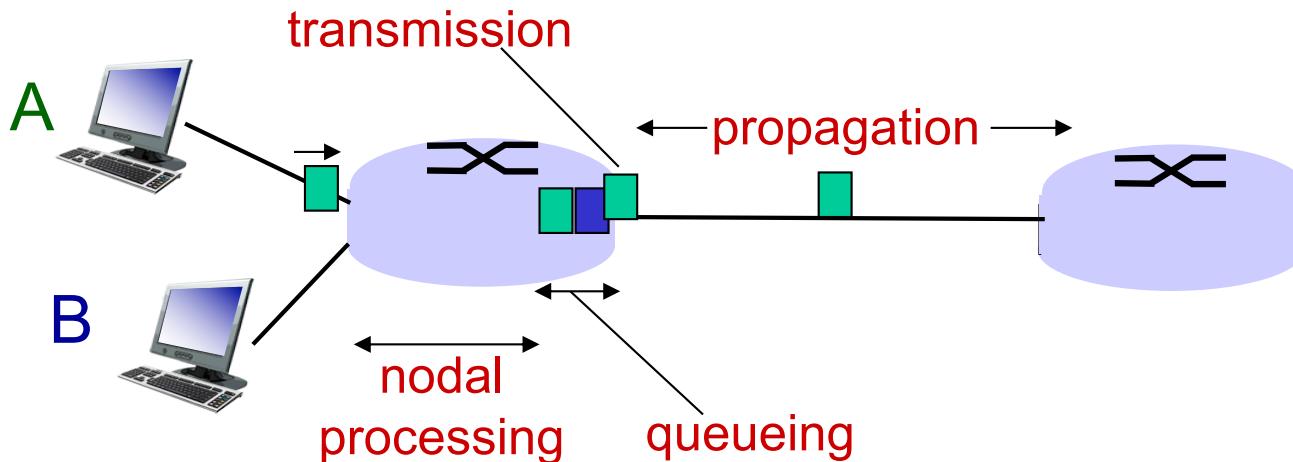
$d_{\text{proc}}$ : nodal processing

- 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:

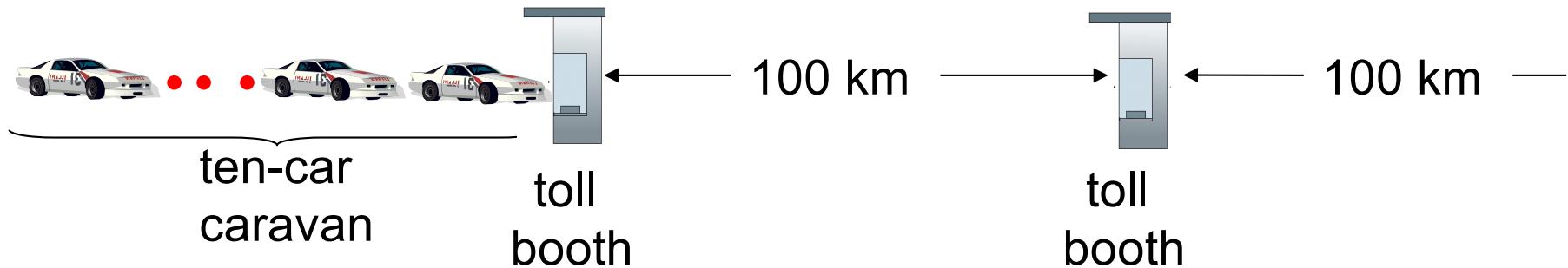
- $L$ : packet length (bits)
- $R$ : link bandwidth ( $\text{bps}$ )
- $d_{\text{trans}} = L/R$

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

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

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

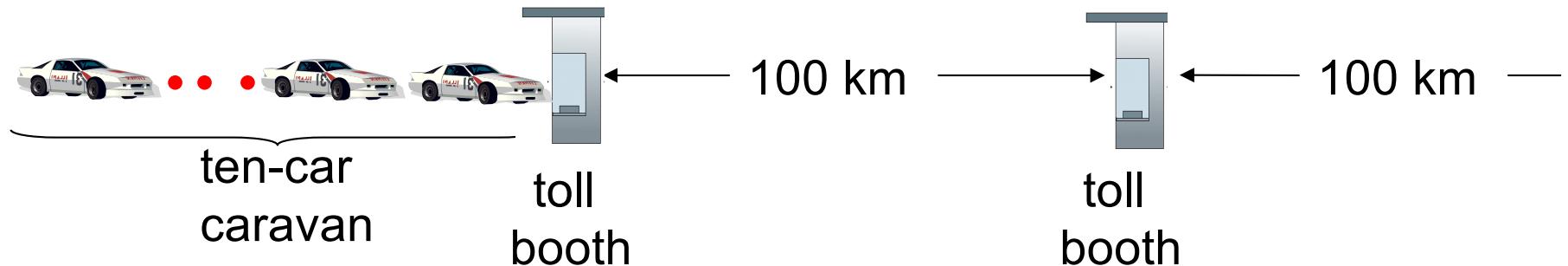
# Caravan analogy



- 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

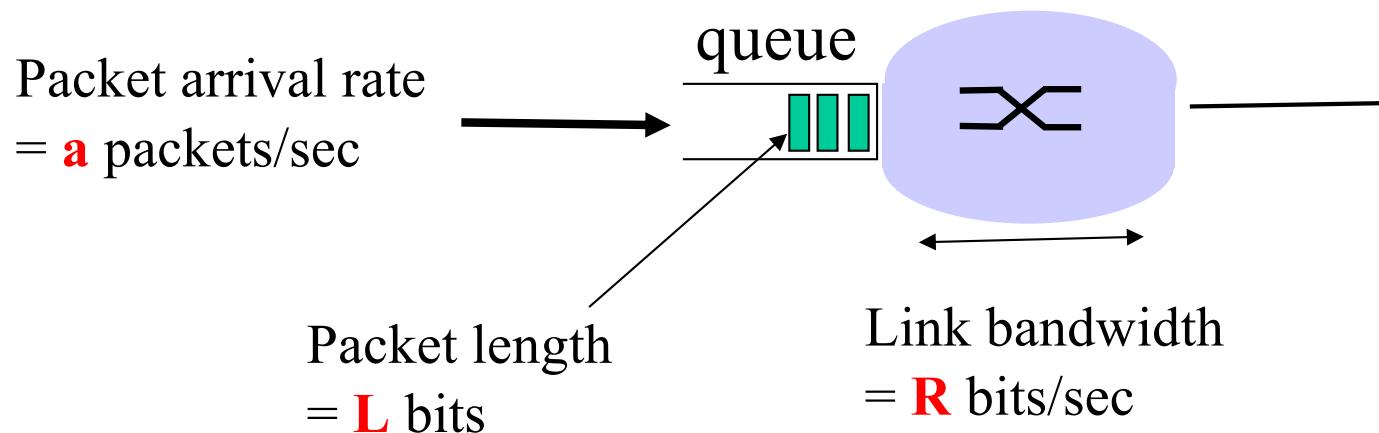
# Caravan analogy (more)



- 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.

Animation comparing  $d_{\text{trans}}$  and  $d_{\text{prop}}$ : <https://www2.tkn.tu-berlin.de/teaching/rn/animations/propagation/>

# Queueing delay (more insight)

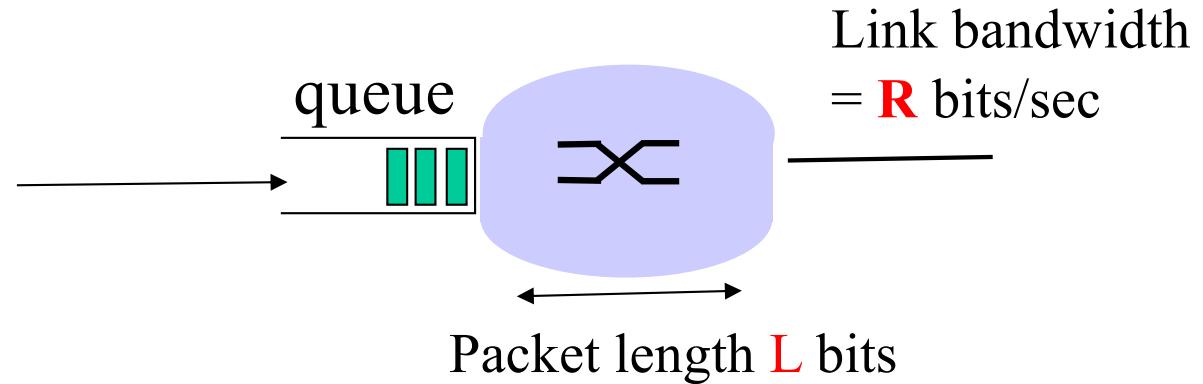


- ❖ Every second:  $aL$  bits arrive to queue
- ❖ Every second:  $R$  bits leave the router
- ❖ Question: what happens if  $aL > R$  ?
- ❖ Answer: queue will fill up, and packets will get dropped!!

$aL/R$  is called traffic intensity

# Queueing delay: illustration

1 packet arrives  
every  $L/R$  seconds



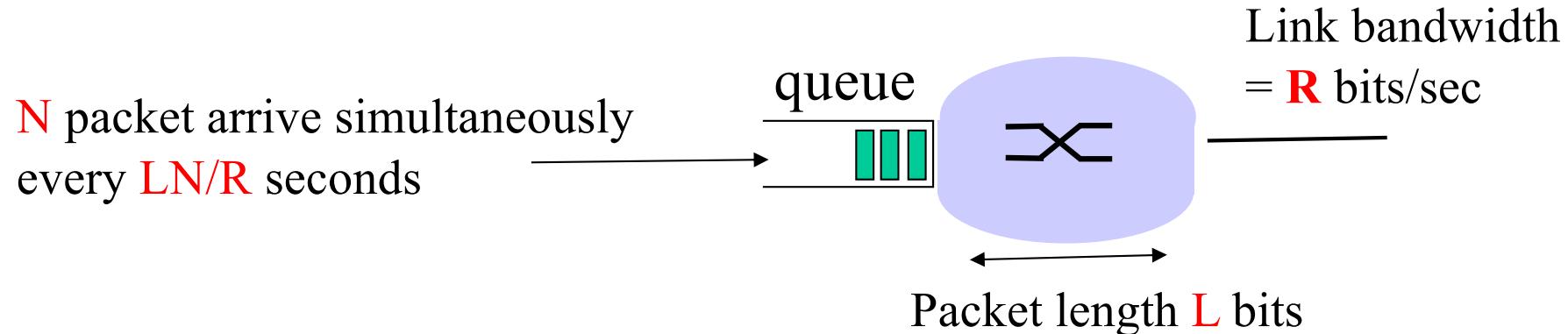
**Arrival rate:**  $a = 1/(L/R) = R/L$  (packet/second)



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

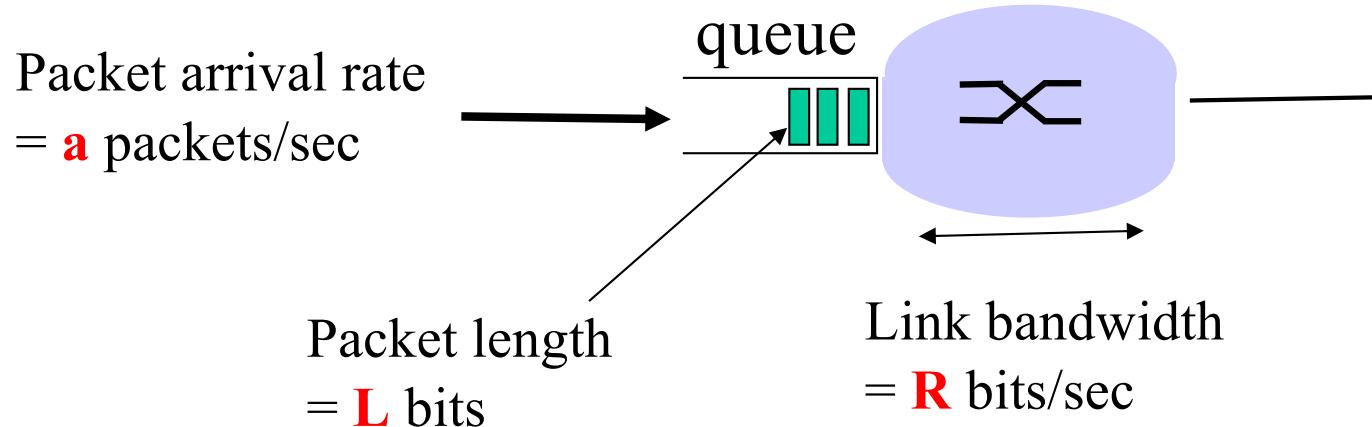


Average queueing delay (queue is empty at time 0) ?

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

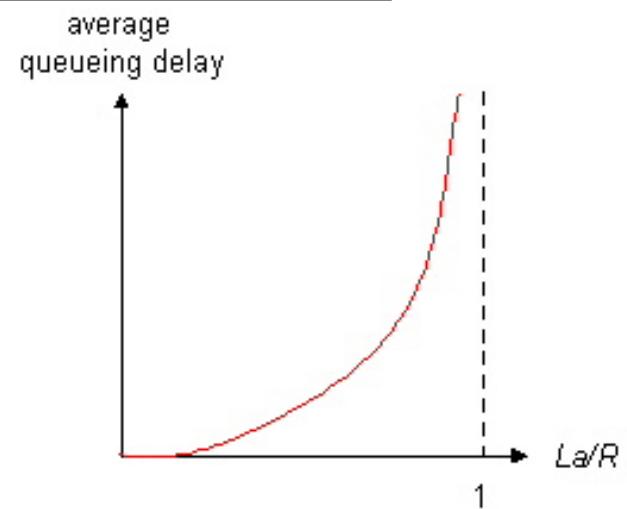
Note: traffic intensity is same as previous scenario, but queueing delay is different

# Queueing delay: behaviour

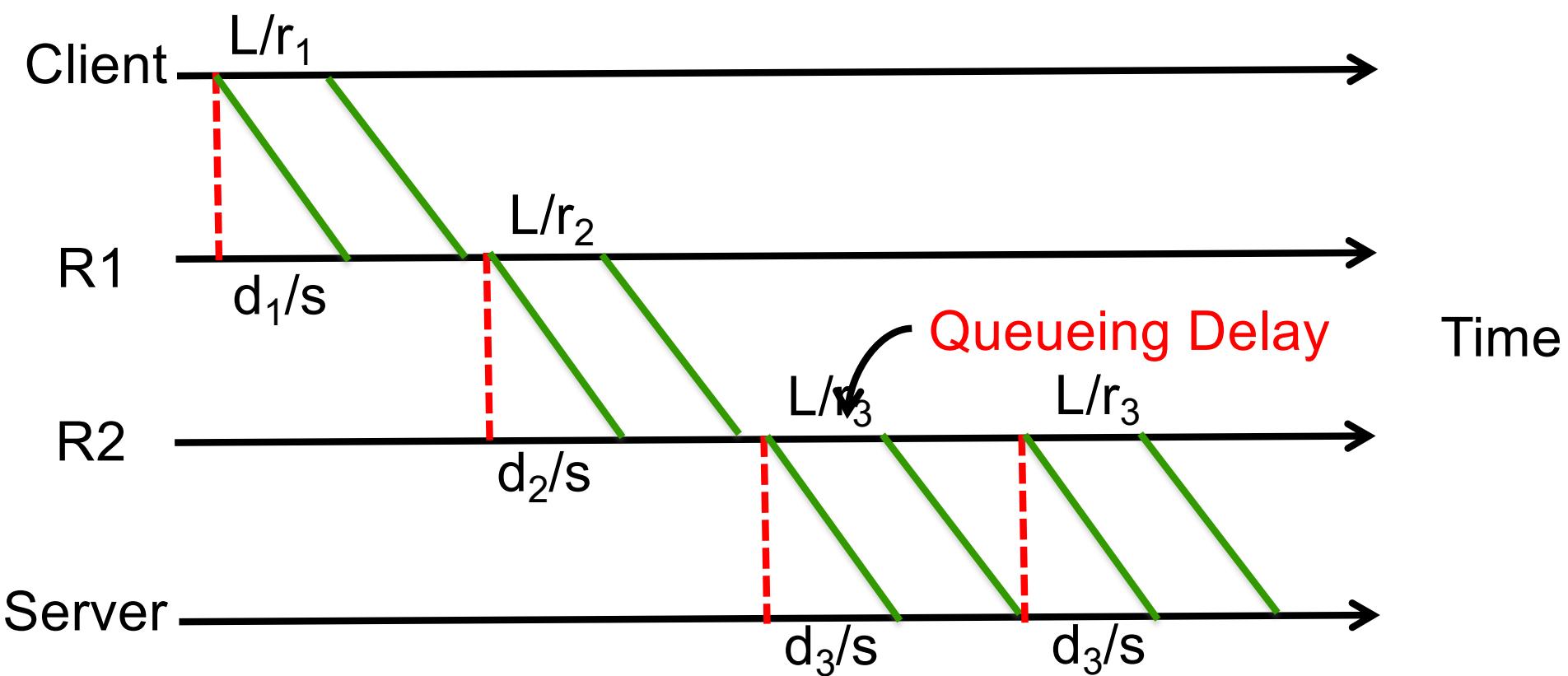
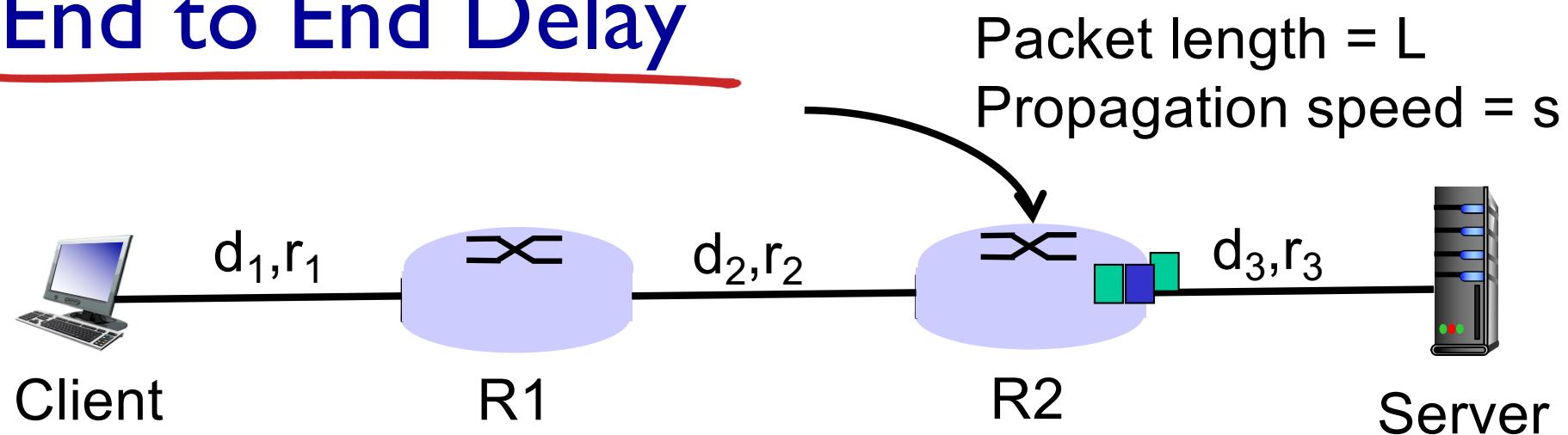


Interactive Java Applet:  
<http://computerscience.unicam.it/marcantoni/reti/applet/QueuingAndLossInteractive/1.html>

- $La/R \sim 0$ : 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!)

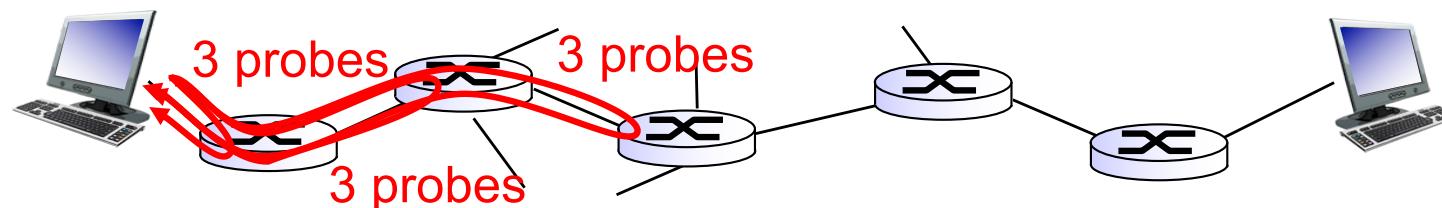


# End to End Delay



# “Real” Internet delays and routes

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



# “Real” Internet delays, routes

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

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

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

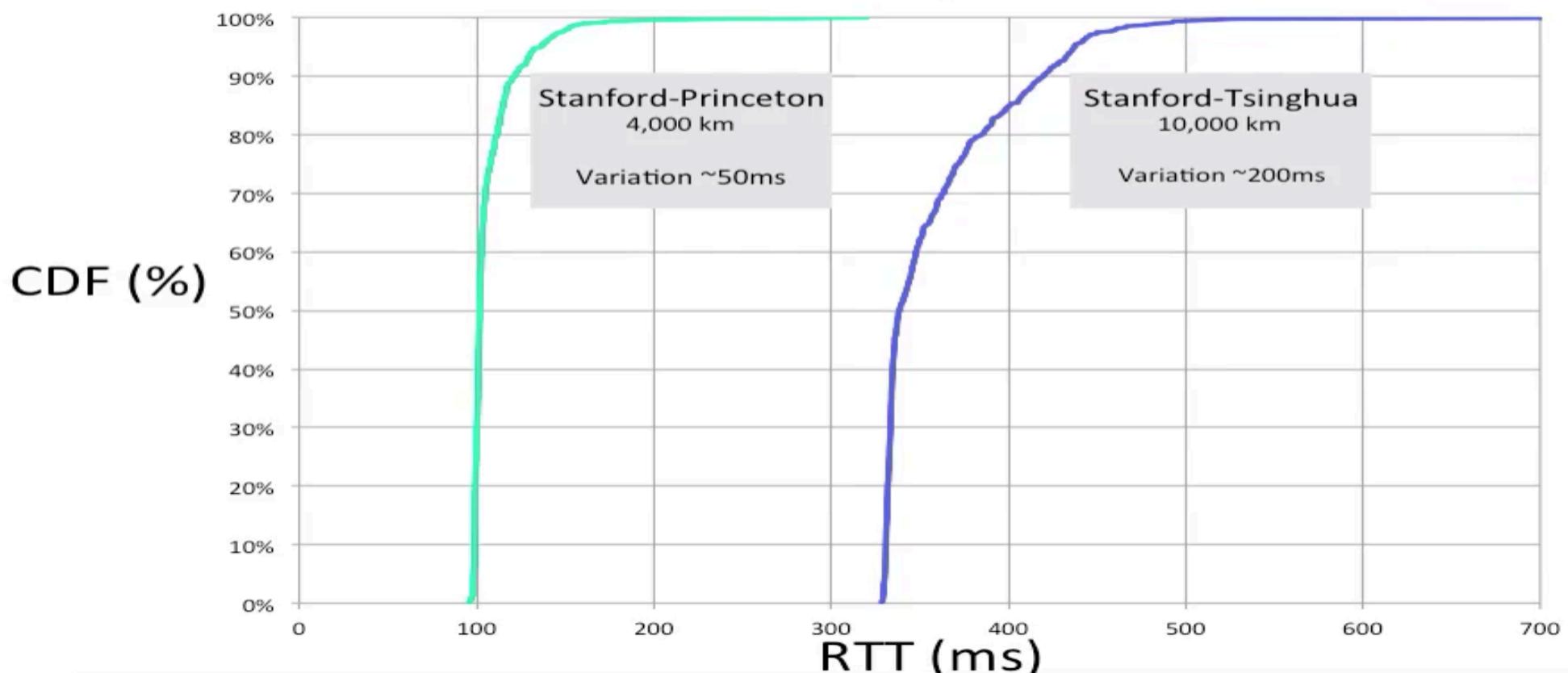
trans-oceanic link

\* Do some traceroutes from 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

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



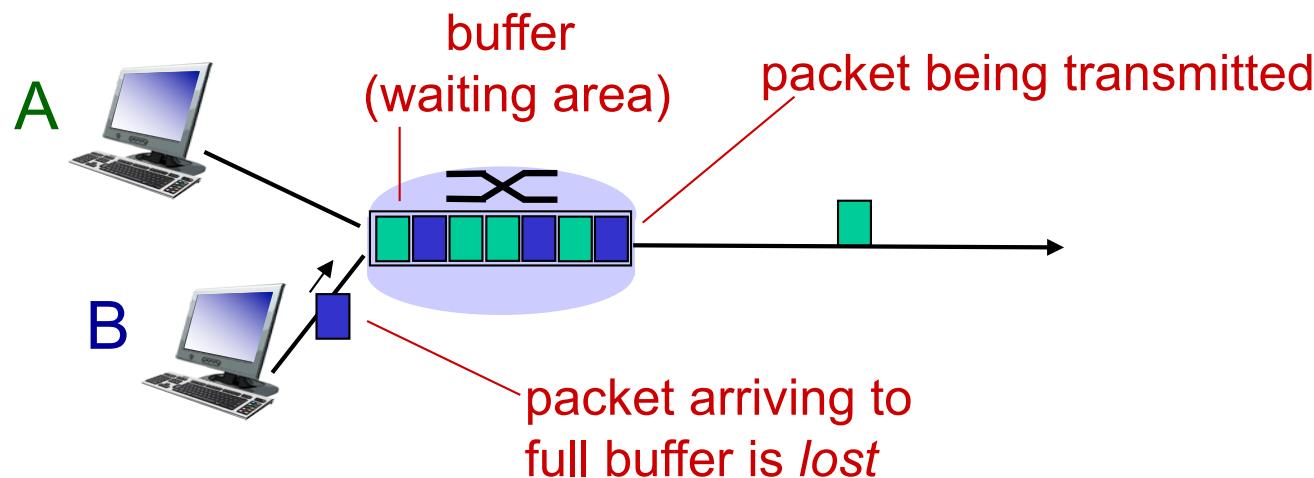
## Quiz: Oh these delays

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

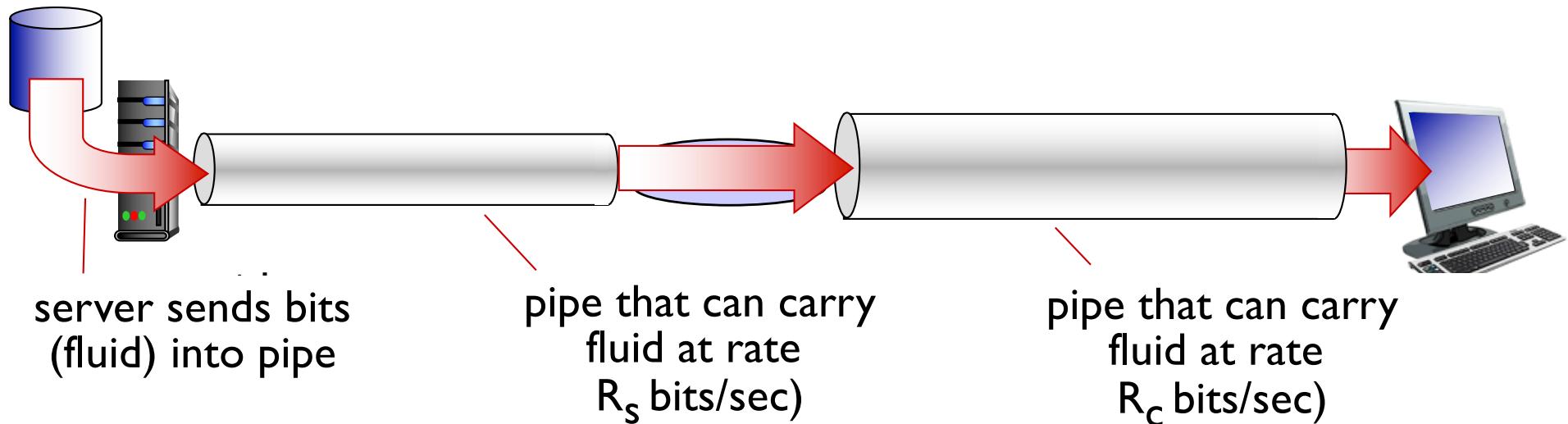
# Packet loss

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



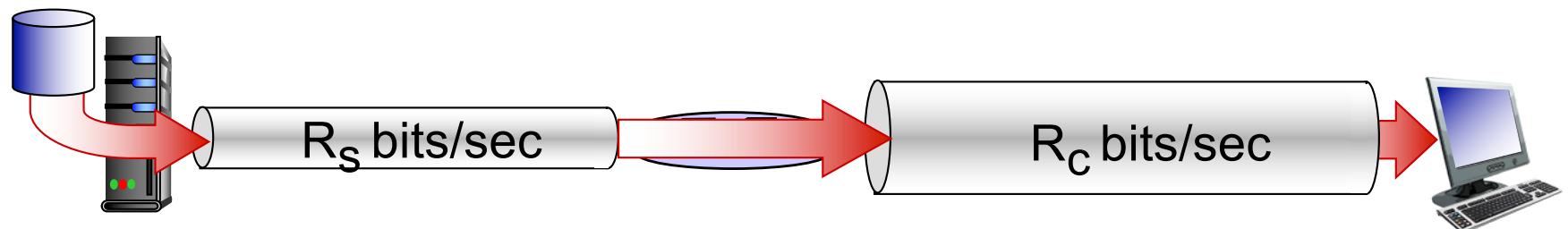
# Throughput

- ❖ *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

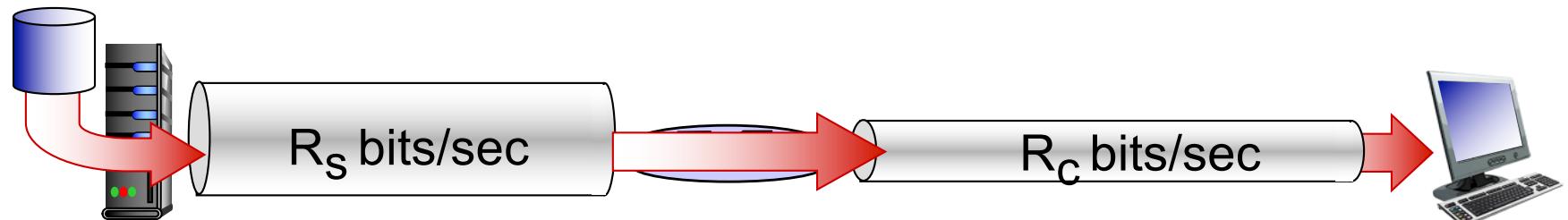


# Throughput (more)

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



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

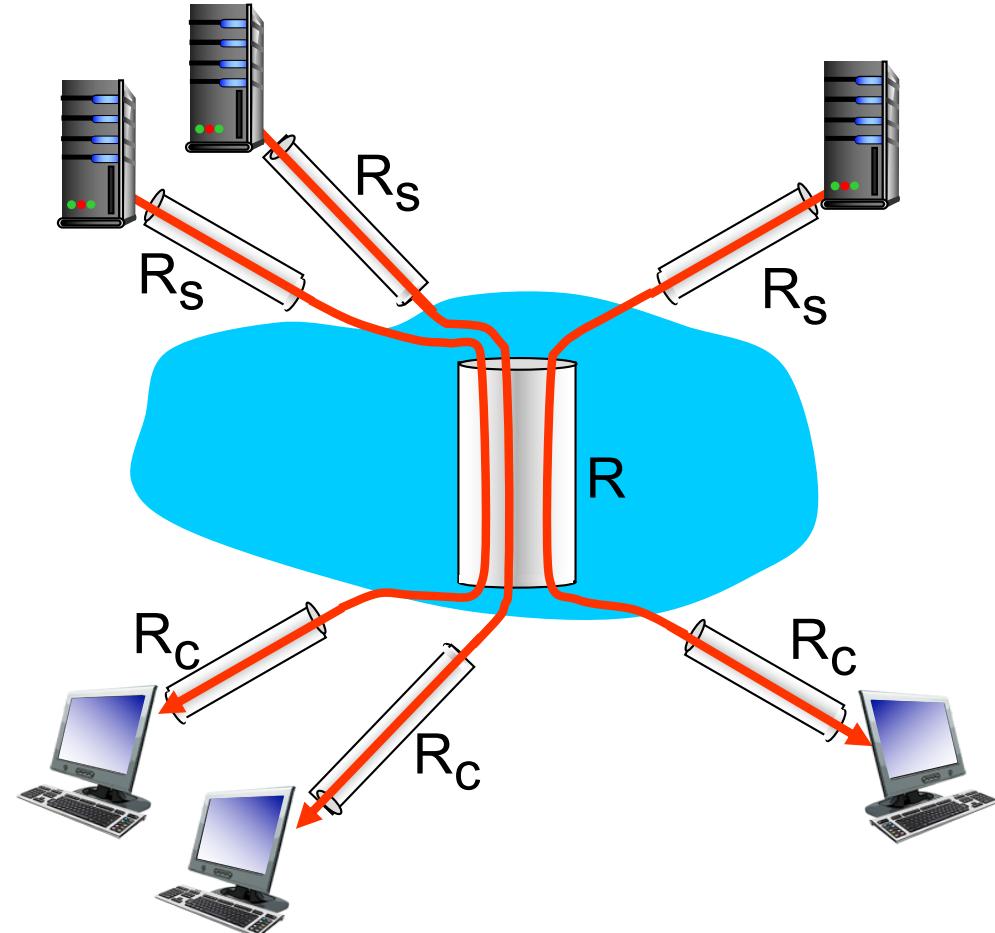


*bottleneck link*

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

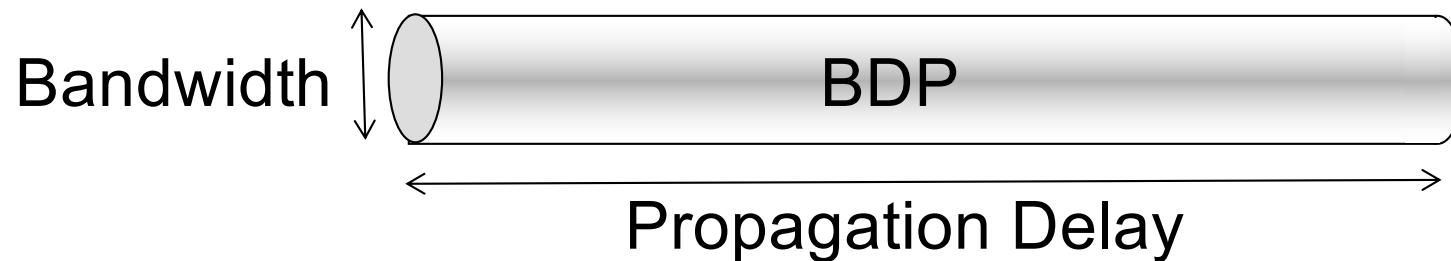
# Introduction: summary



*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

# Key Properties of Links



- Bandwidth: “width” of the link
  - Number of bits sent per unit of time (bps)
- Propagation delay: “length” of the link
  - Propagation time to travel along the link (seconds)
- Bandwidth-Delay Product: “Volume” of the link
  - Amount of data in flight ( $\text{bps} \times \text{s} = \text{bits}$ )

# 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

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 the users?

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

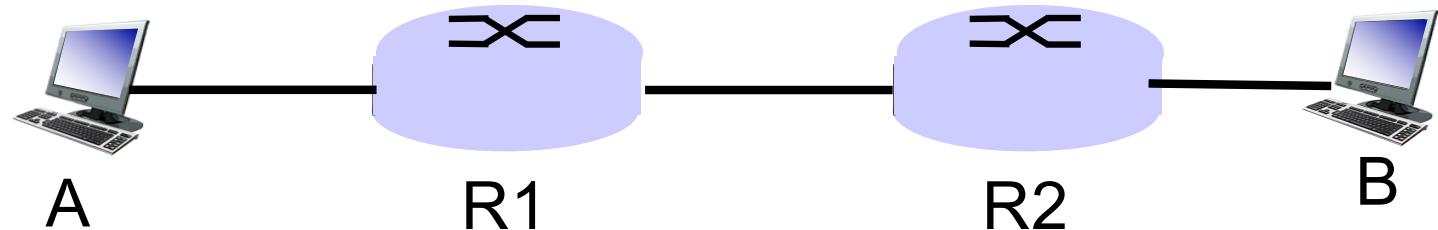
Open a browser and type: **[www.zeetings.com/salil](http://www.zeetings.com/salil)**



## 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



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 takes 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 administrative aide

**Dear John,**

» Aide:

**Your days are numbered.**

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

--Pat

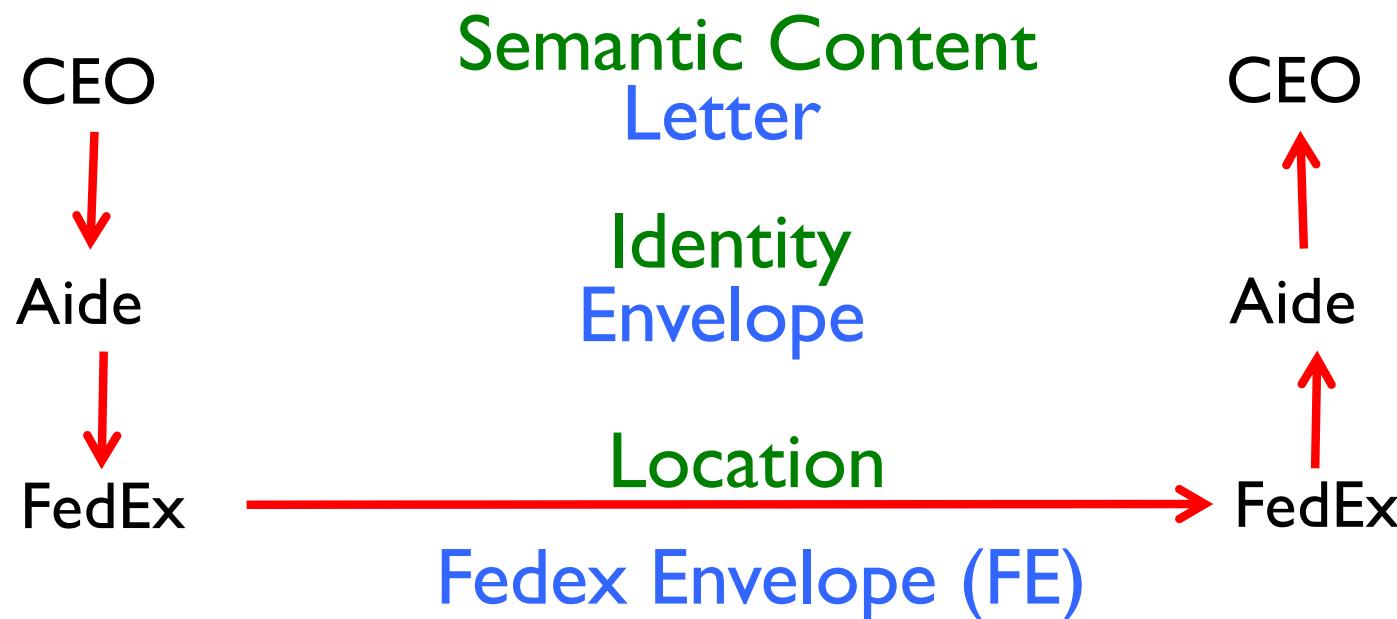
- ❖ FedEx Office
  - Puts letter in larger envelope
  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck
- ❖ FedEx delivers to other company

# The Path of the Letter

“Peers” on each side understand the same things

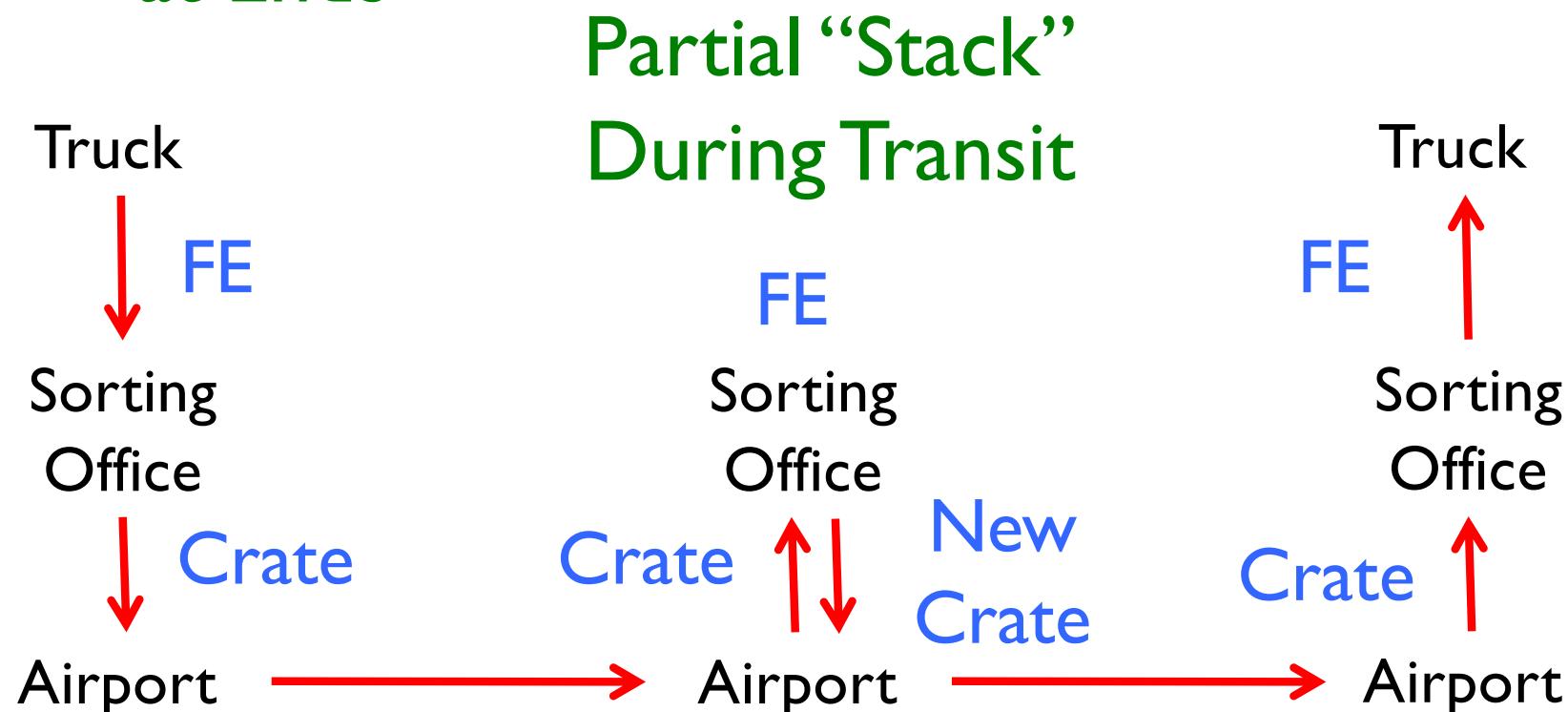
No one else needs to (abstraction)

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...

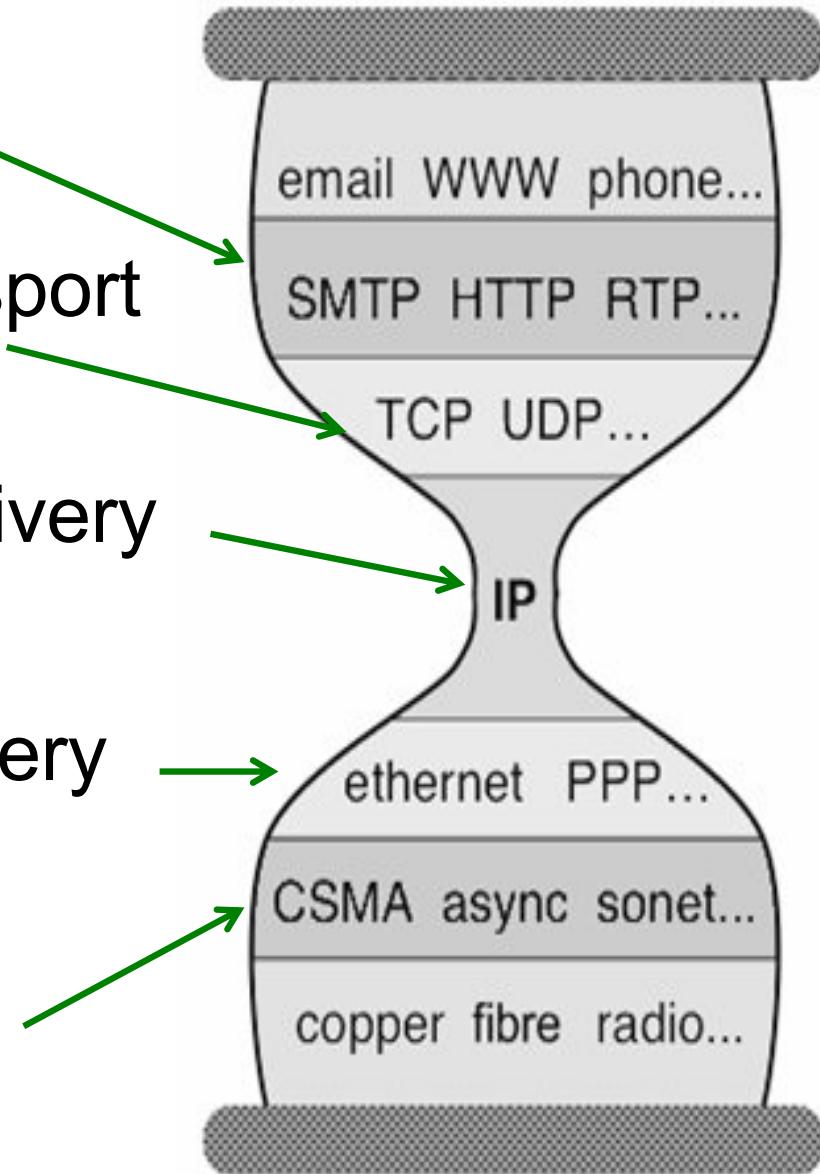
Best-effort global packet delivery

...built on...

Best-effort local packet delivery

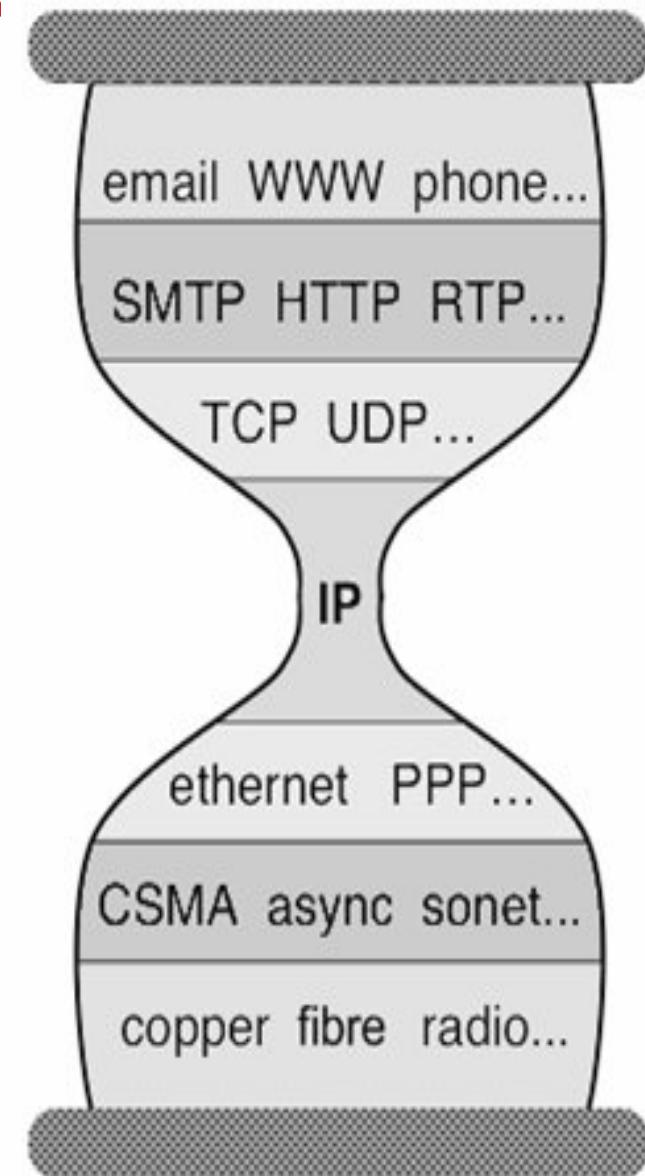
...built on...

Physical transfer of bits



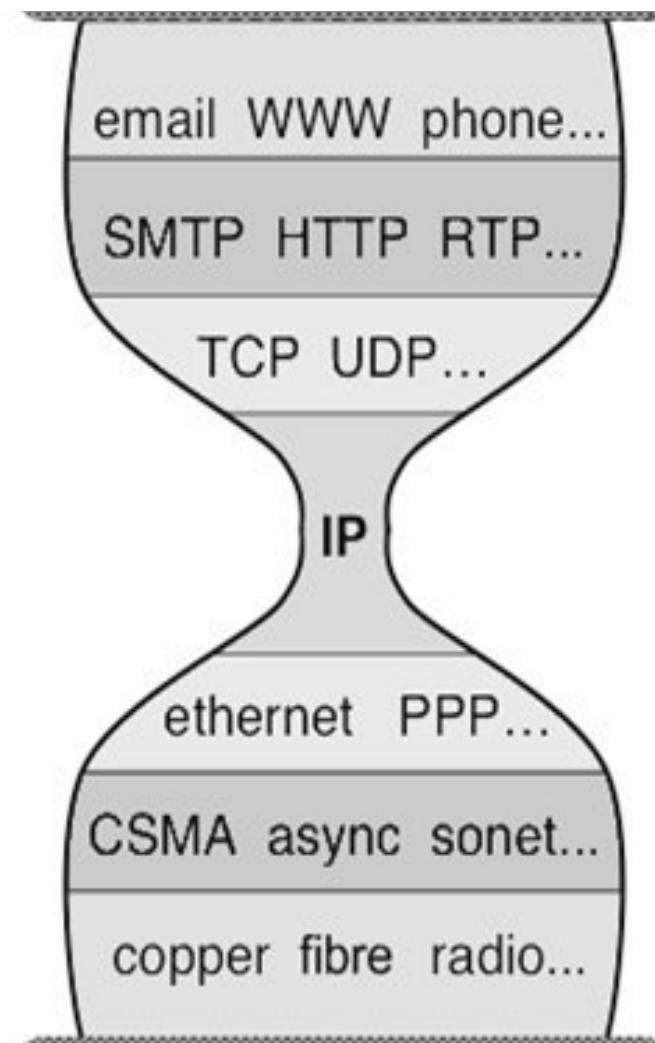
# Internet protocol stack

- ❖ *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”

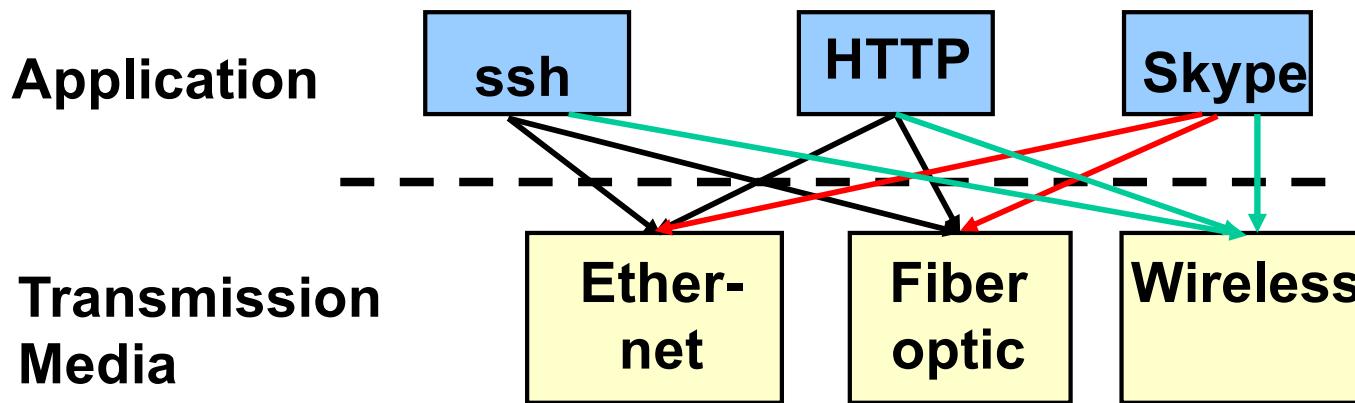


# Three Observations

- ❖ 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



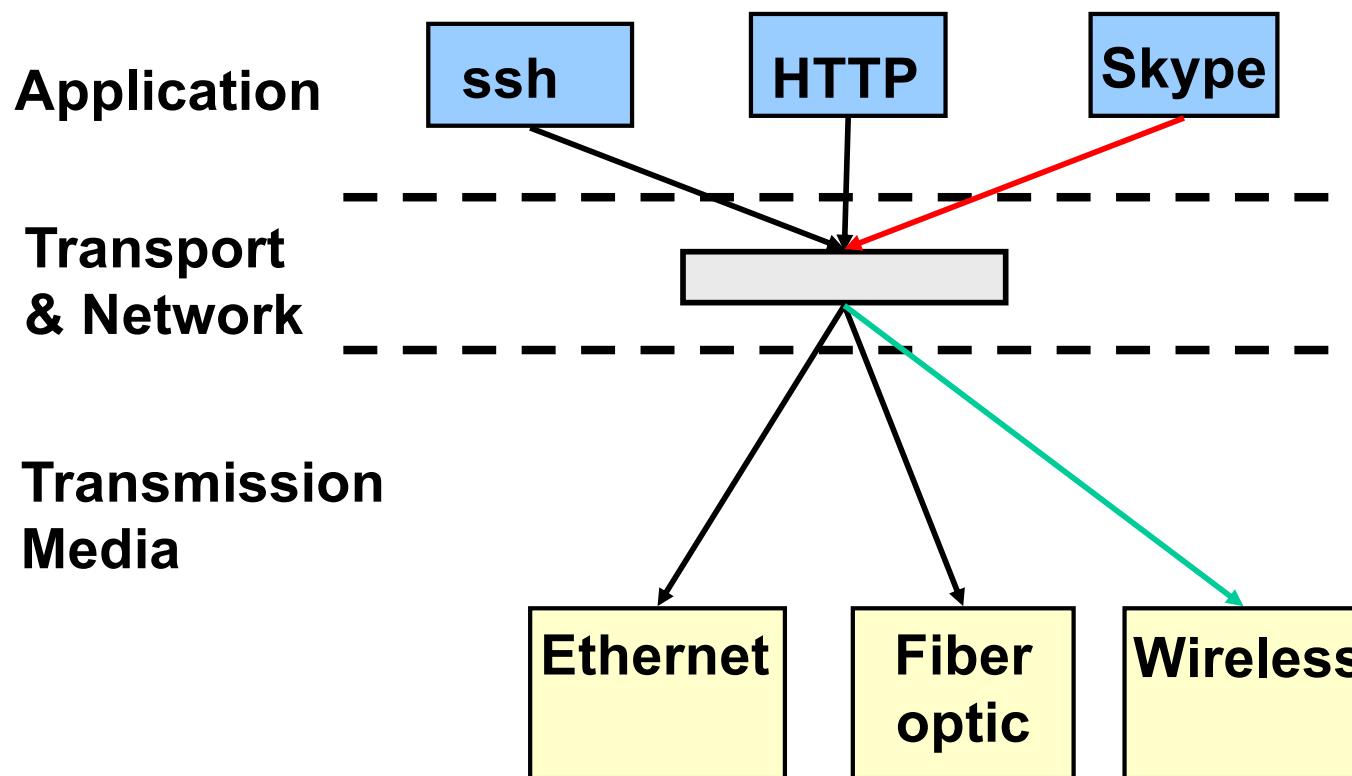
# An Example: No Layering



- ❖ 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 really big
  - 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?

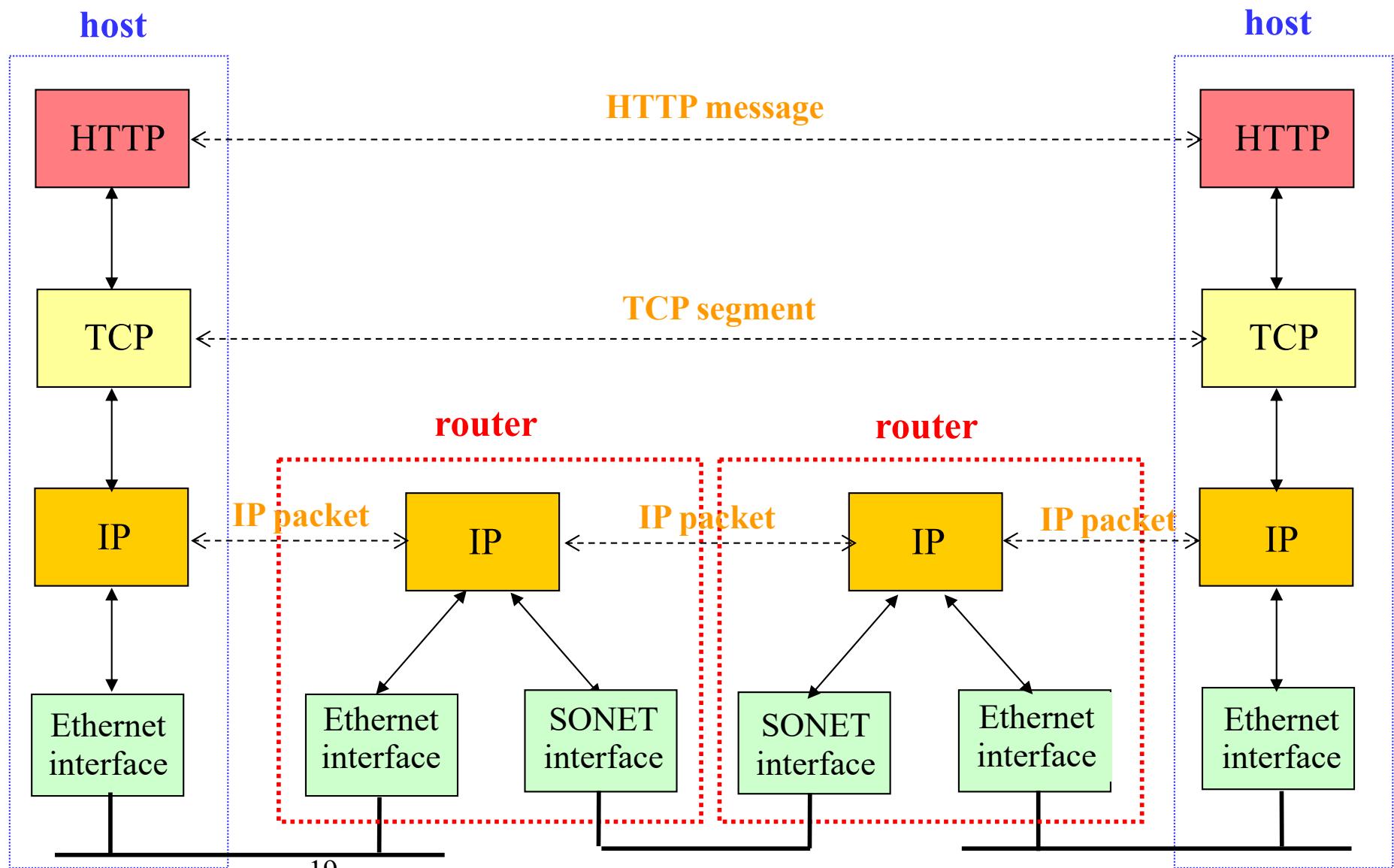
# What Gets Implemented on Host?

- ❖ 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?

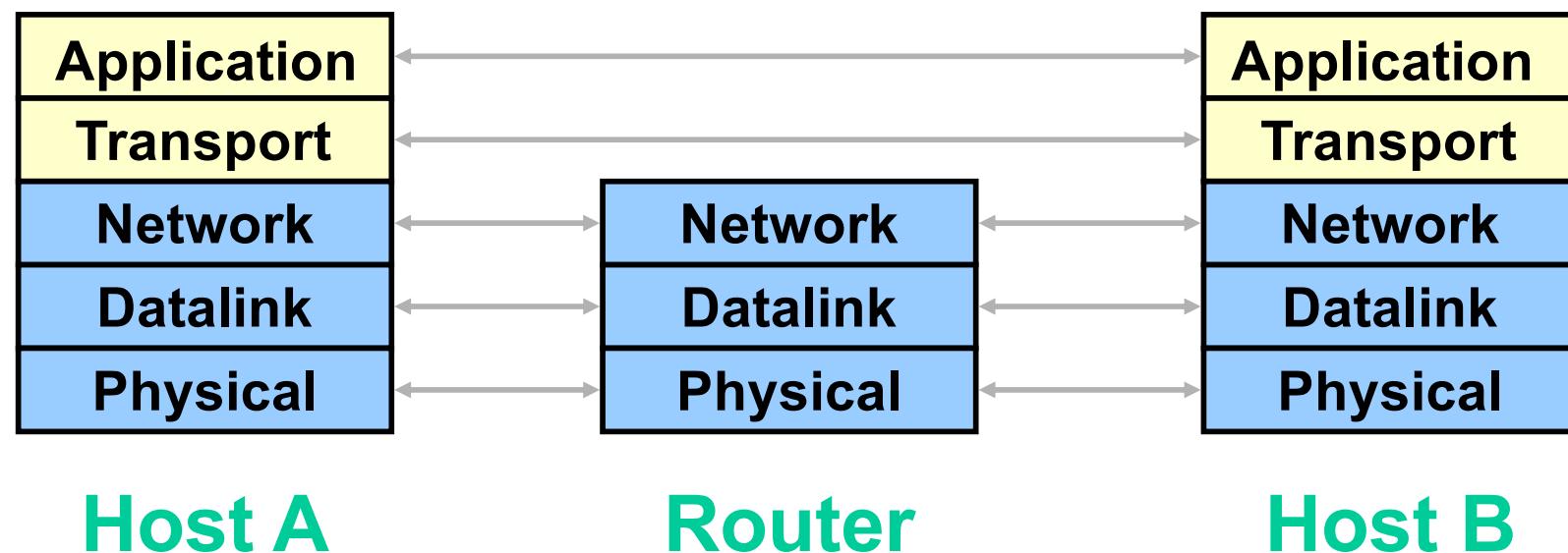
- ❖ Bits arrive on wire
  - Physical layer necessary
- ❖ Packets must be delivered to next-hop
  - datalink layer necessary
- ❖ Routers participate in global delivery
  - Network layer necessary
- ❖ Routers don't support reliable delivery
  - Transport layer (and above) **not** supported

# Internet Layered Architecture



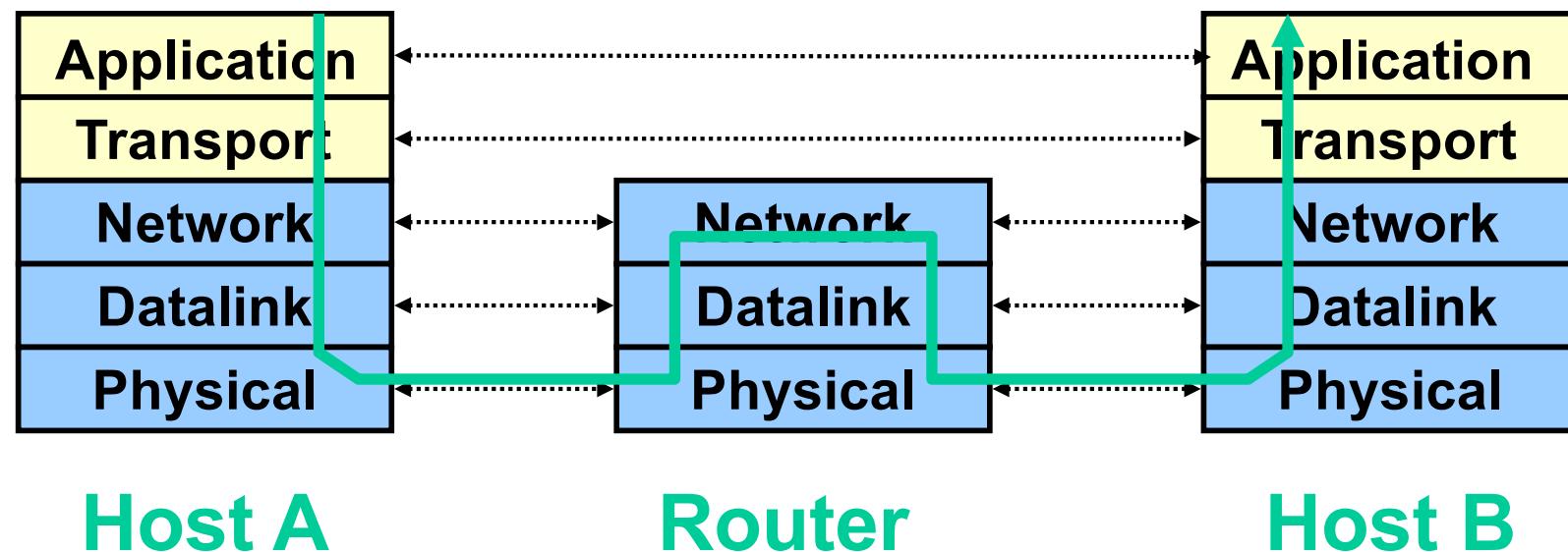
# Logical Communication

- ❖ Layers interacts with peer's corresponding layer



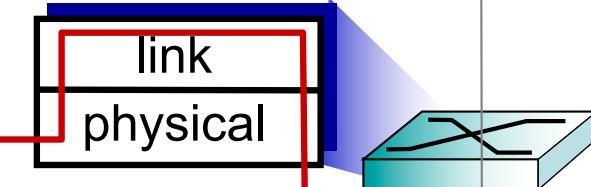
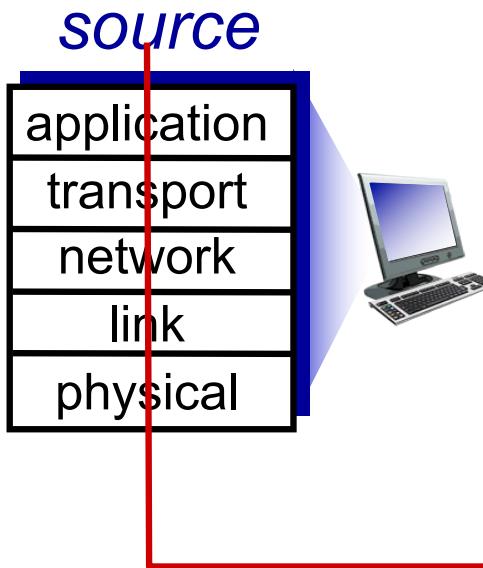
# Physical Communication

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

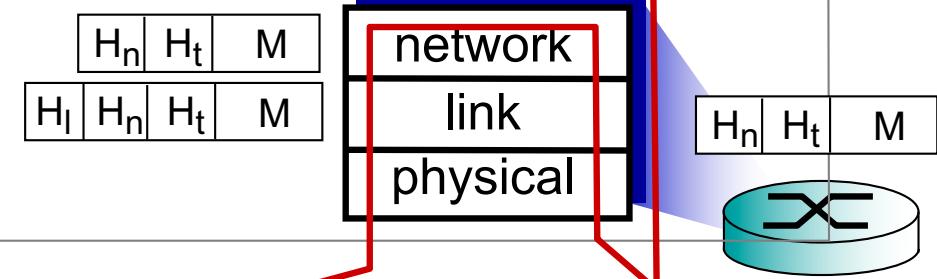
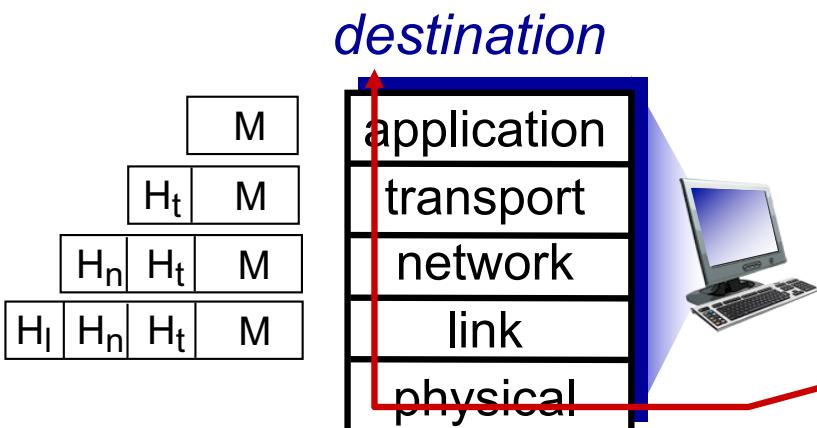


# Encapsulation

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

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 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, 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

## 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

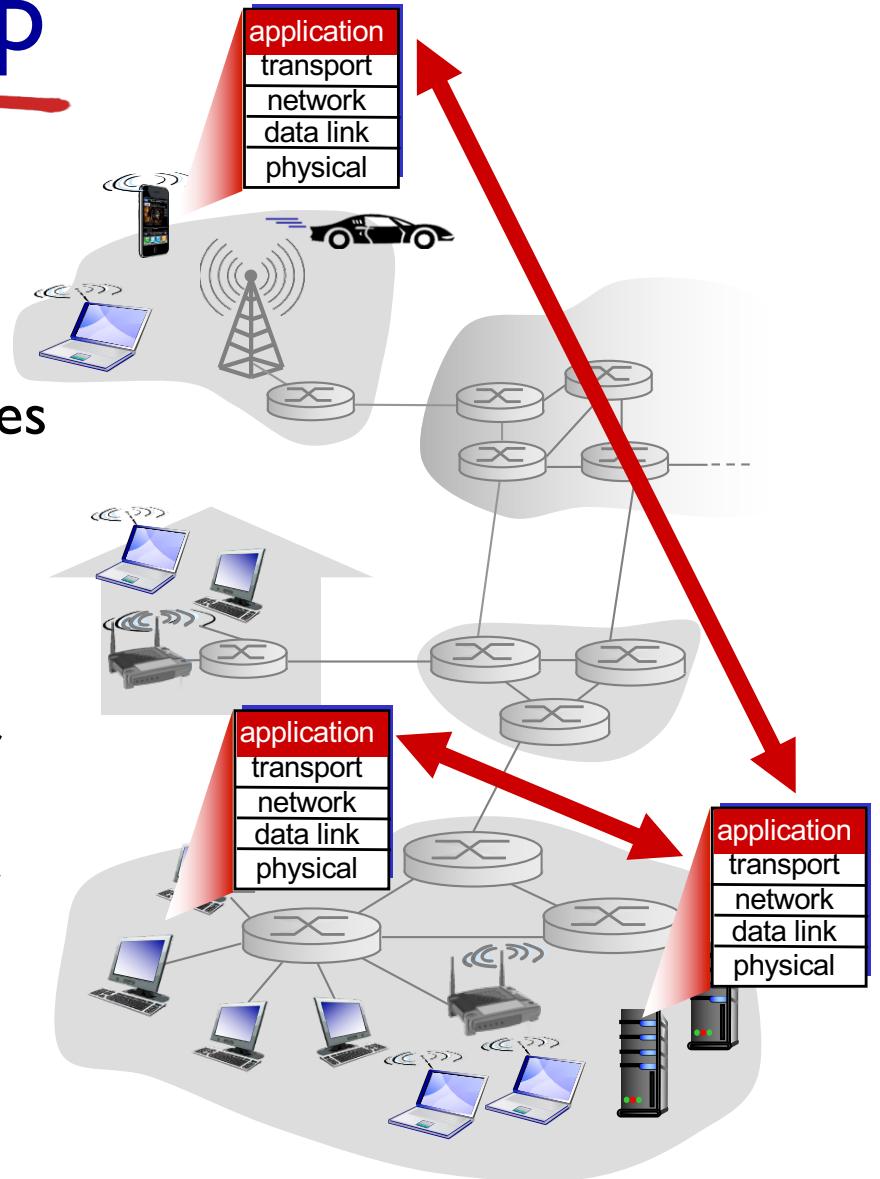
# Creating a network app

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

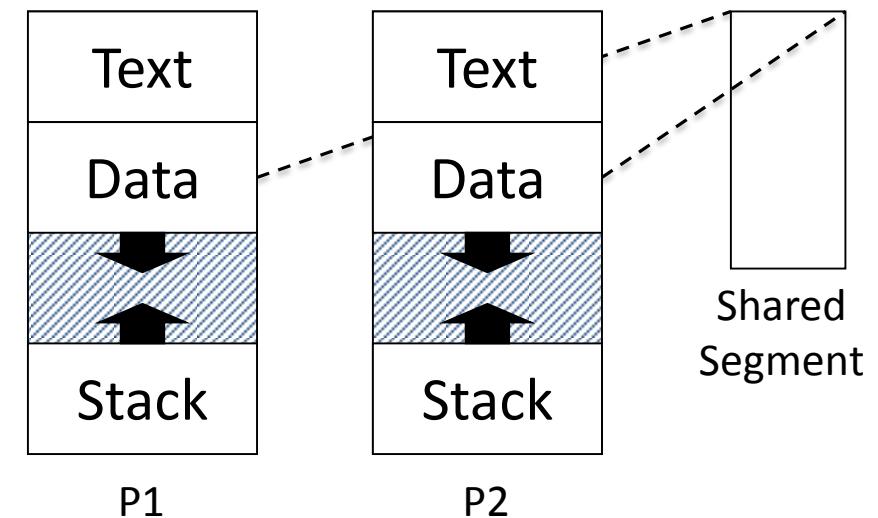
- ❖ network-core devices do not run user applications
- ❖ applications on end systems allows for rapid app development



# Interprocess Communication (IPC)

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

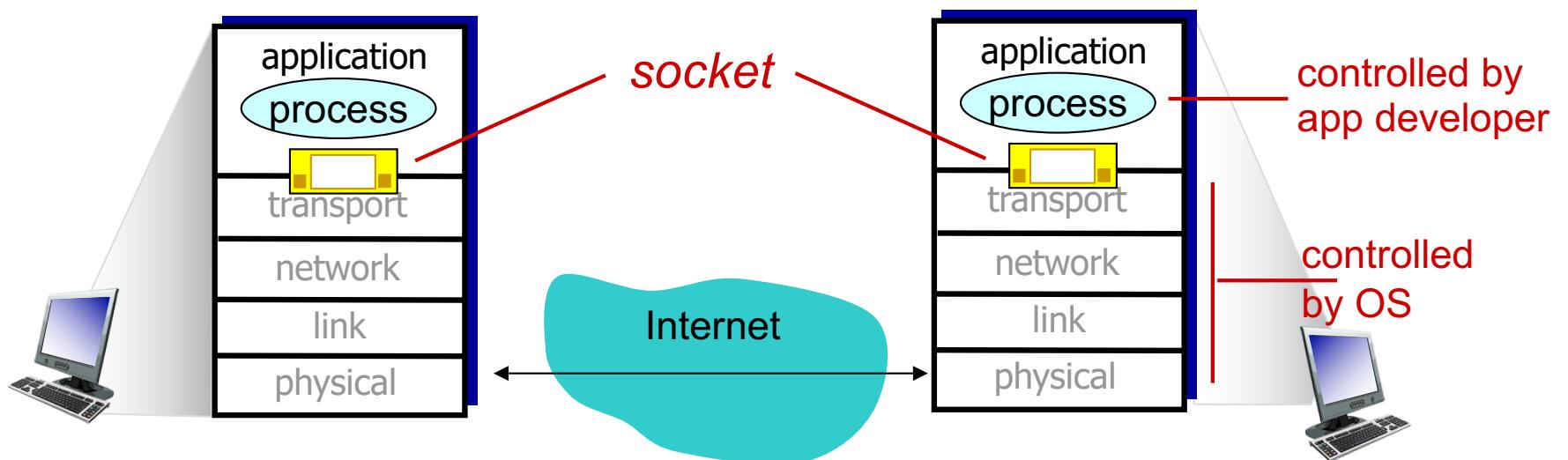
- ❖ On a single machine:
  - Shared memory



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

# Sockets

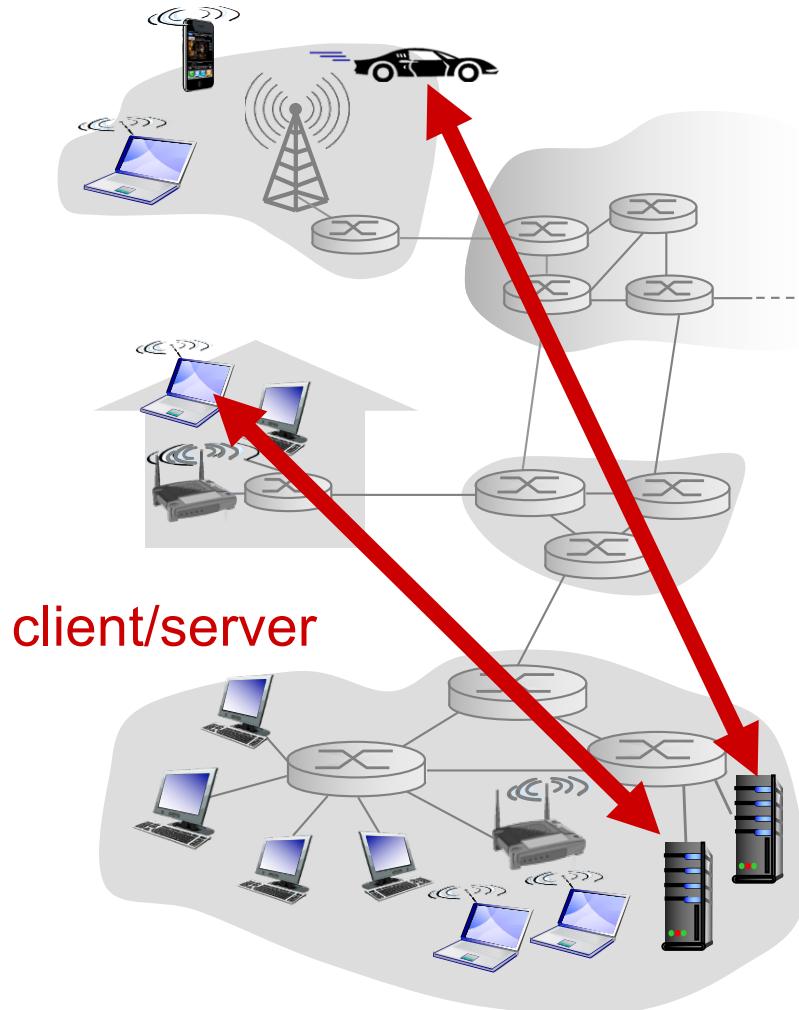
- ❖ process sends/receives messages to/from its **socket**
- ❖ socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
- ❖ Application has a few options, OS handles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

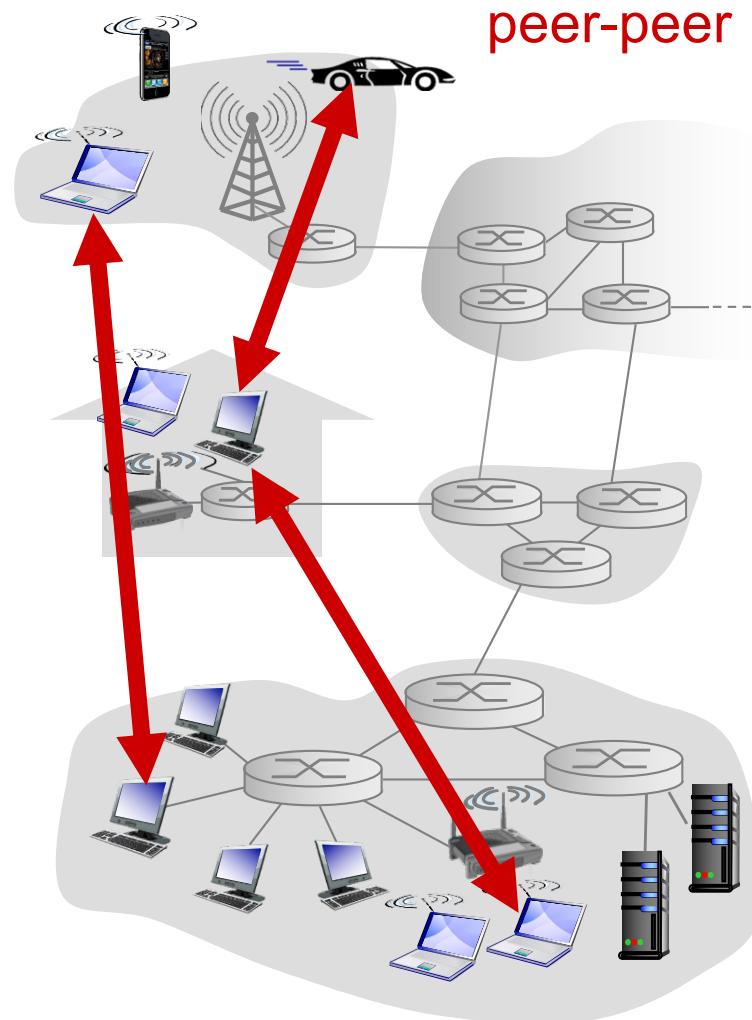
- Always-on host
- Permanent IP address (rendezvous location)
- Static port conventions (http: 80, email: 25, ssh:22)
- Data centres for scaling
- May communicate with other servers to respond

## ❖ 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

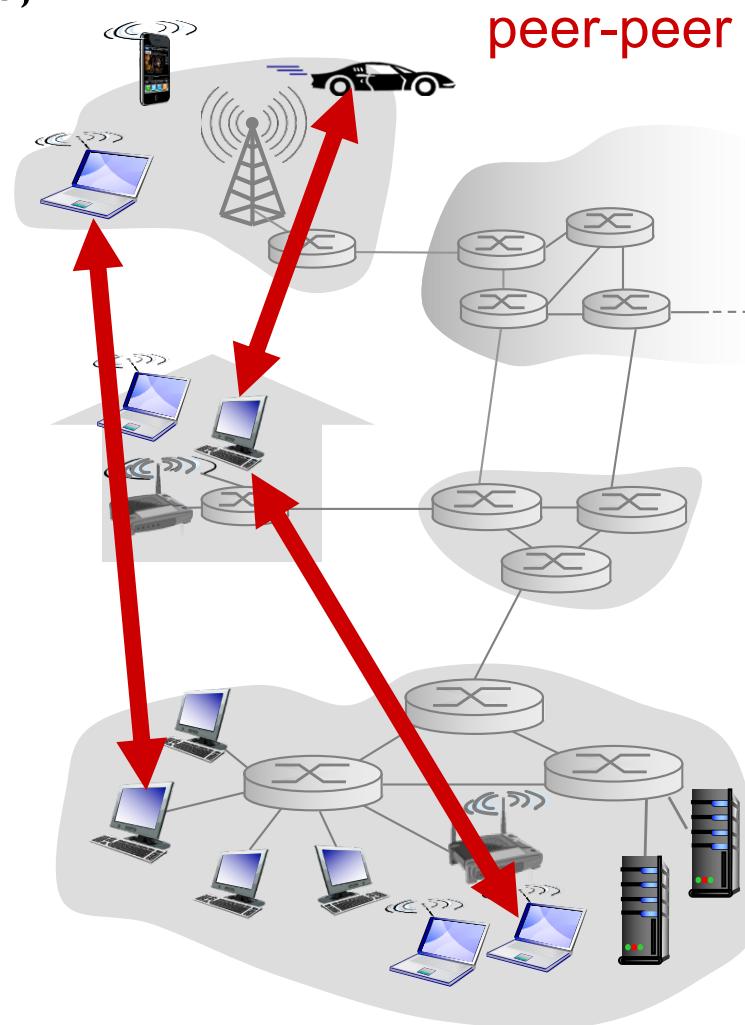
+ Reliability: redundancy, fault tolerance

+ 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

**proprietary protocols:**

- ❖ e.g., Skype

# What transport service does an app need?

## data integrity

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

## timing

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

## throughput

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

## security

- ❖ encryption, data integrity,

...

# Transport service requirements: common apps

<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

	<b>application layer protocol</b>	<b>underlying transport protocol</b>
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

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, 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

# 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>

# 2019 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)

---

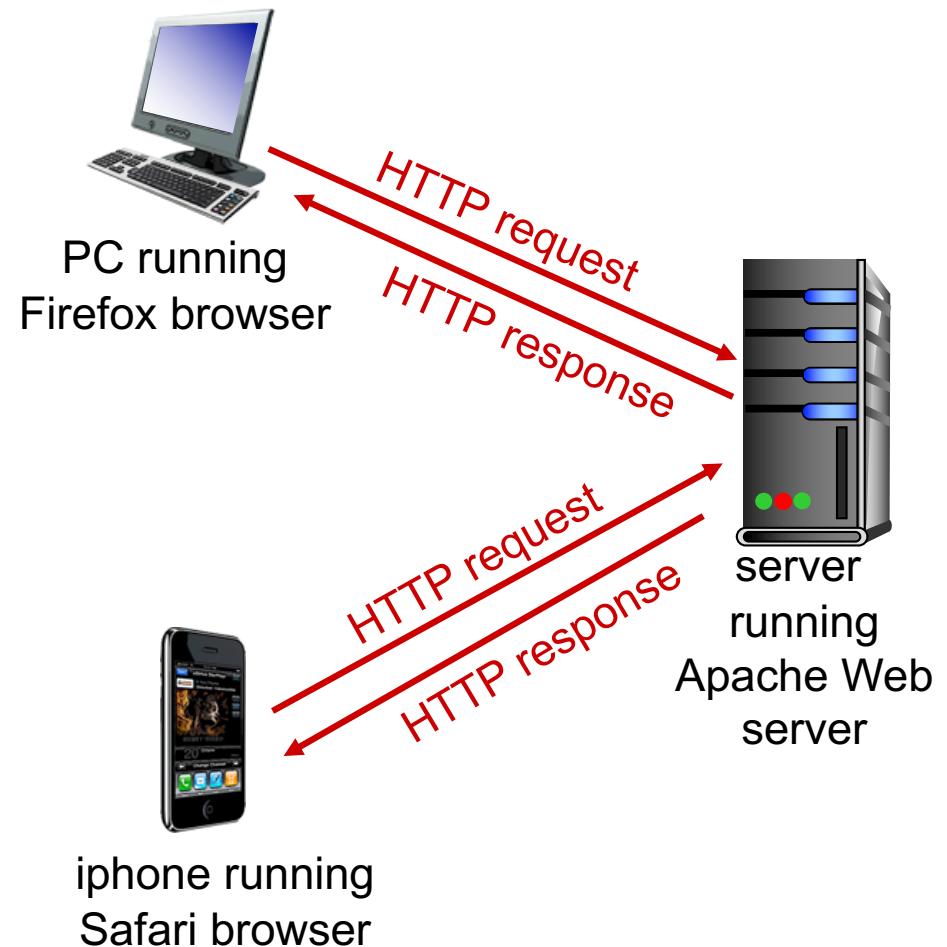
**protocol://host-name[:port]/directory-path/resource**

- ❖ *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

*aside*  
protocols that maintain “state” are complex!

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

# 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

# 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

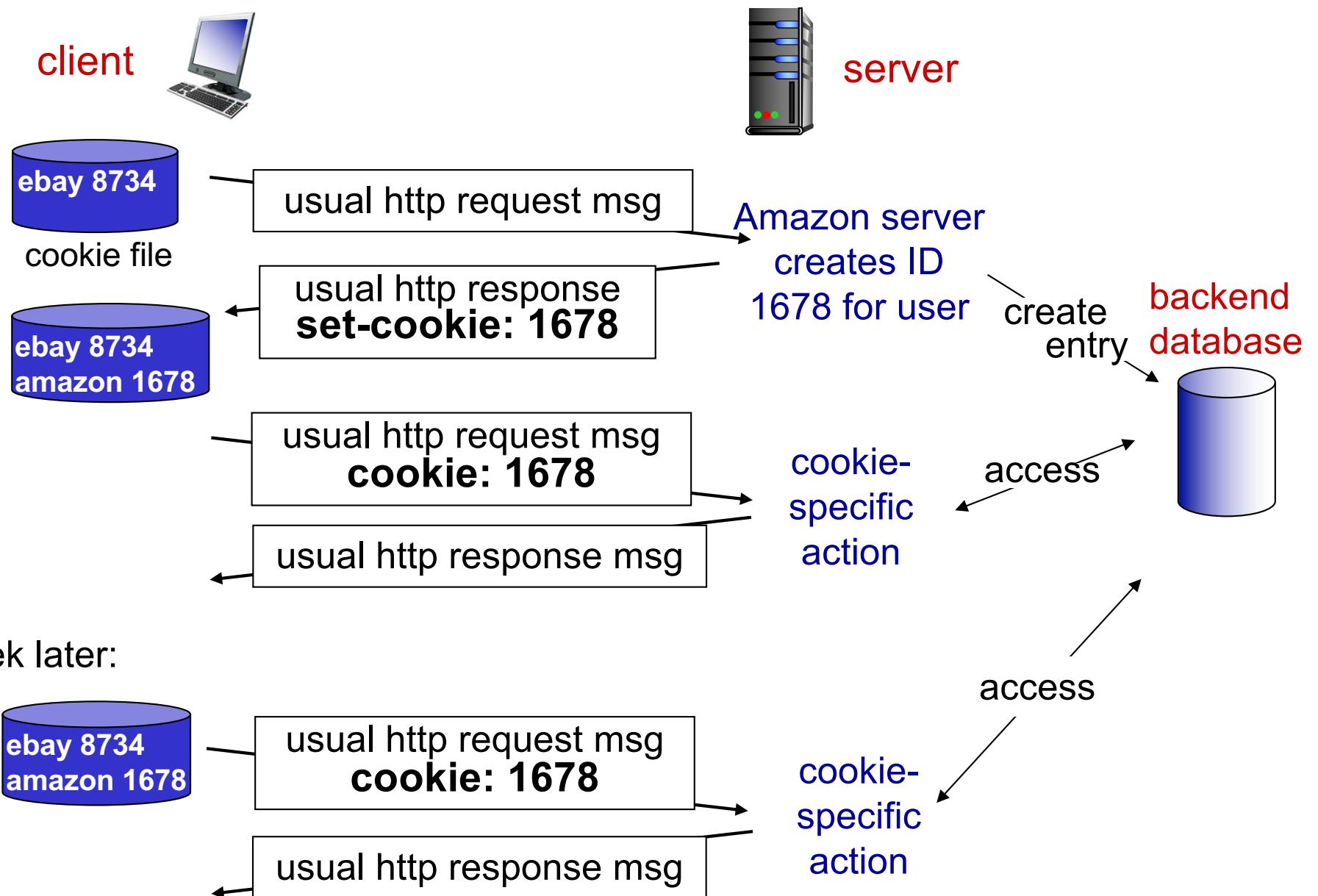
*four components:*

- 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

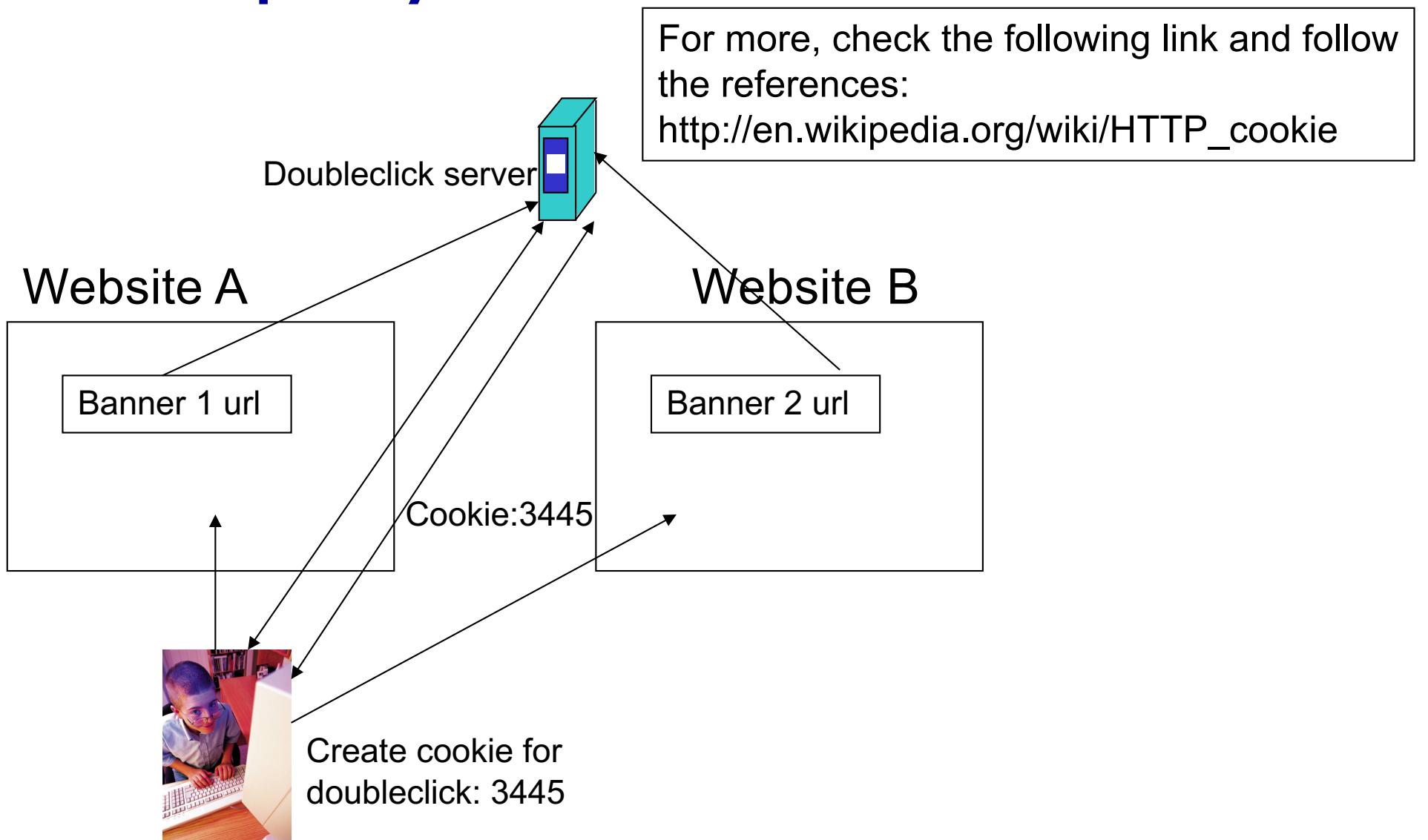
# Cookies: keeping “state” (cont.)



# 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



# 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

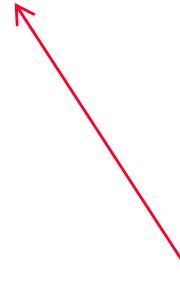
- happy users (hence, above)
- cost-effective delivery infrastructure

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 Page Load Time (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*

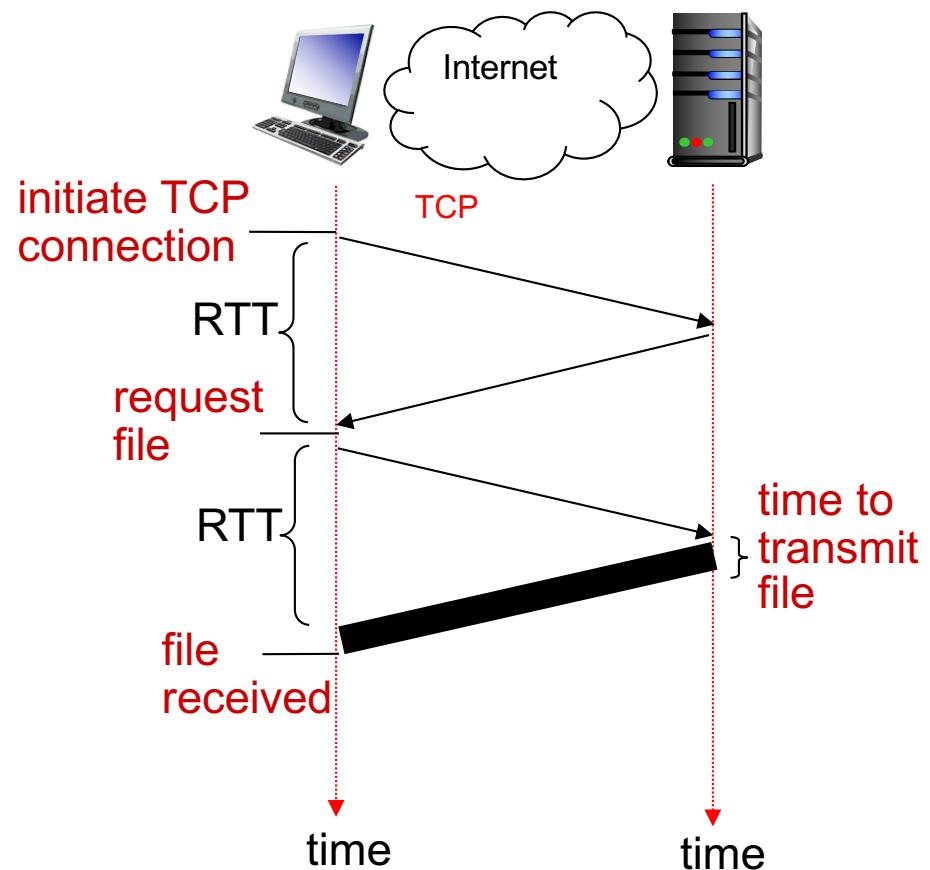
- ❖ at most one object sent over TCP connection
  - connection then closed
- ❖ downloading multiple objects required multiple connections

# Non-persistent HTTP: response time

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

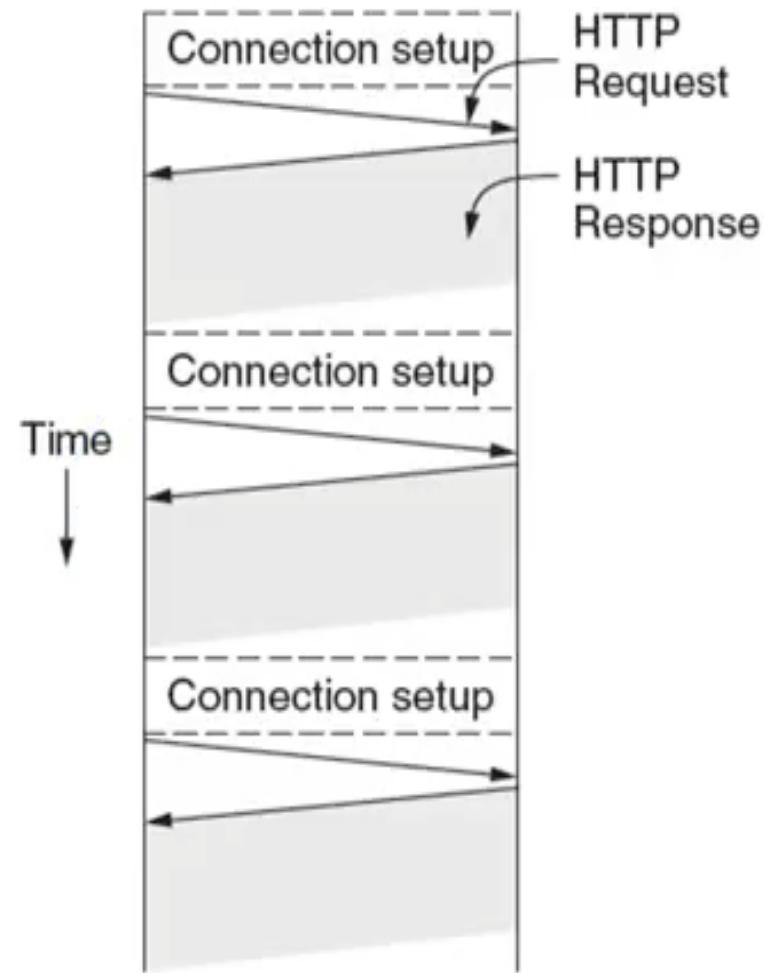
**HTTP response time:**

- ❖ 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

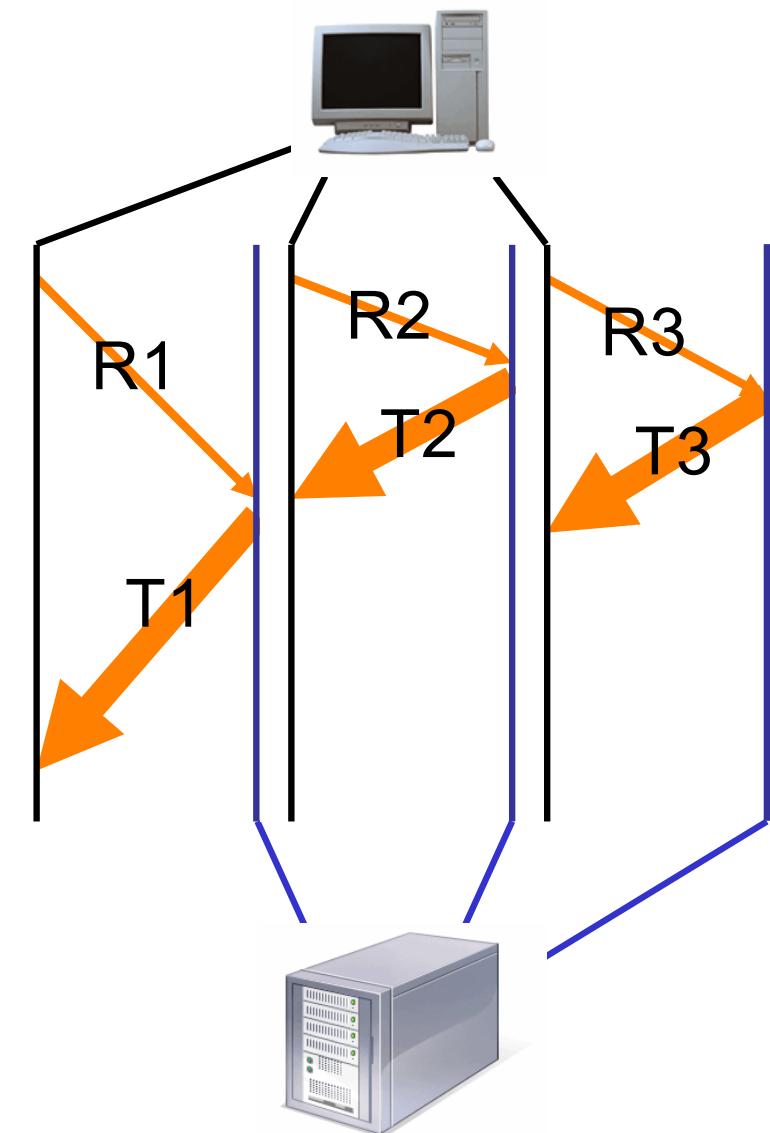
- Non-Persistent: One TCP connection to fetch one web resource
- Fairly poor PLT
- 2 Scenarios
  - Multiple TCP connections setups to the **same server**
  - Sequential request/responses even when resources are located on **different servers**
- Multiple TCP slow-start phases (more in lecture on TCP)



# 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?

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

# Persistent HTTP

## Persistent HTTP

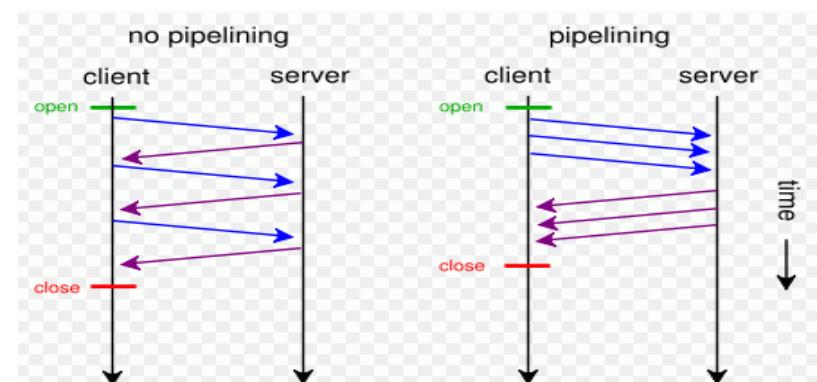
- ❖ server leaves TCP connection open after sending response
- ❖ subsequent HTTP messages between same client/server are sent over the same TCP connection
- ❖ Allow TCP to learn more accurate RTT estimate (APPARENT LATER IN THE COURSE)
- ❖ Allow TCP congestion window to increase (APPARENT LATER)
- ❖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

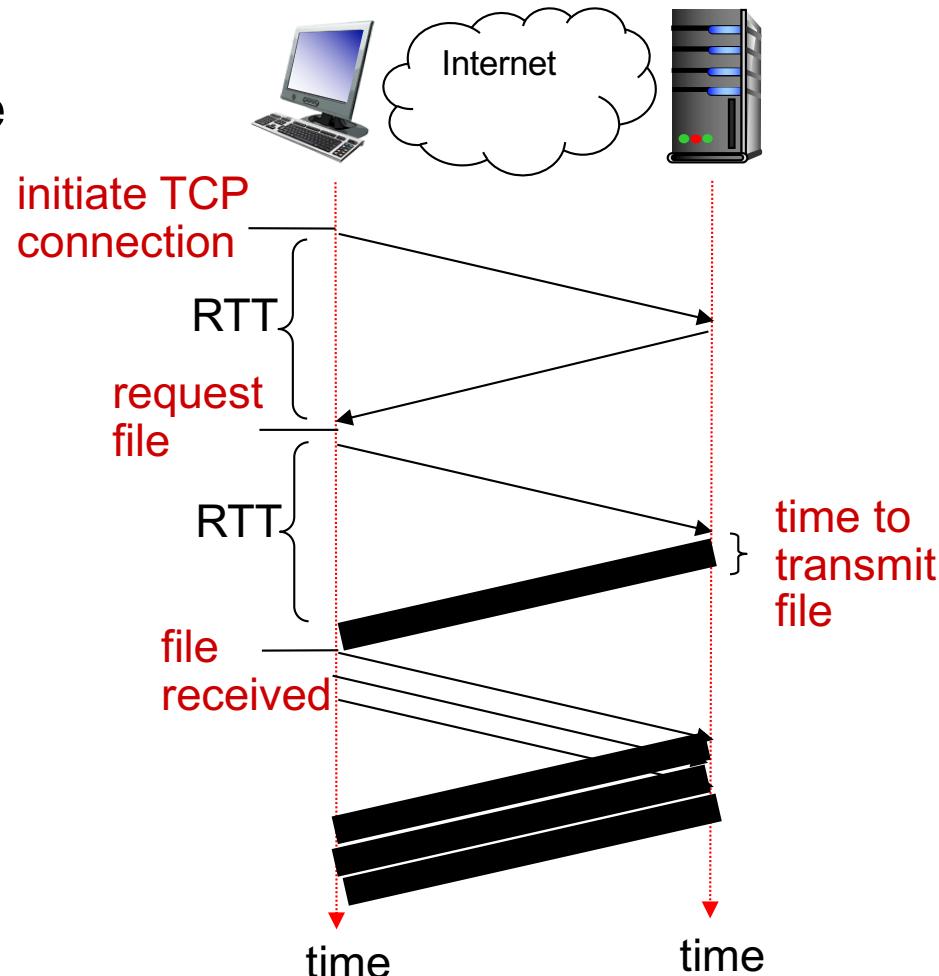
## Persistent with pipelining:

- ❖ 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

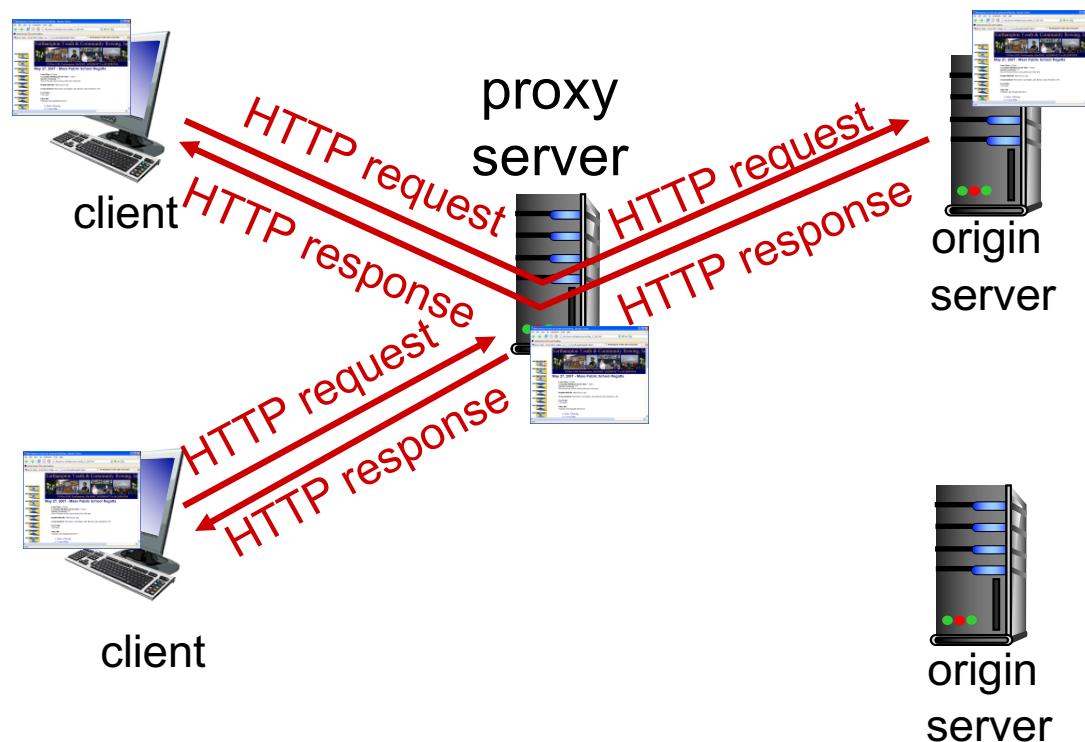
---

- Why does caching work?
  - Exploits *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

# 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

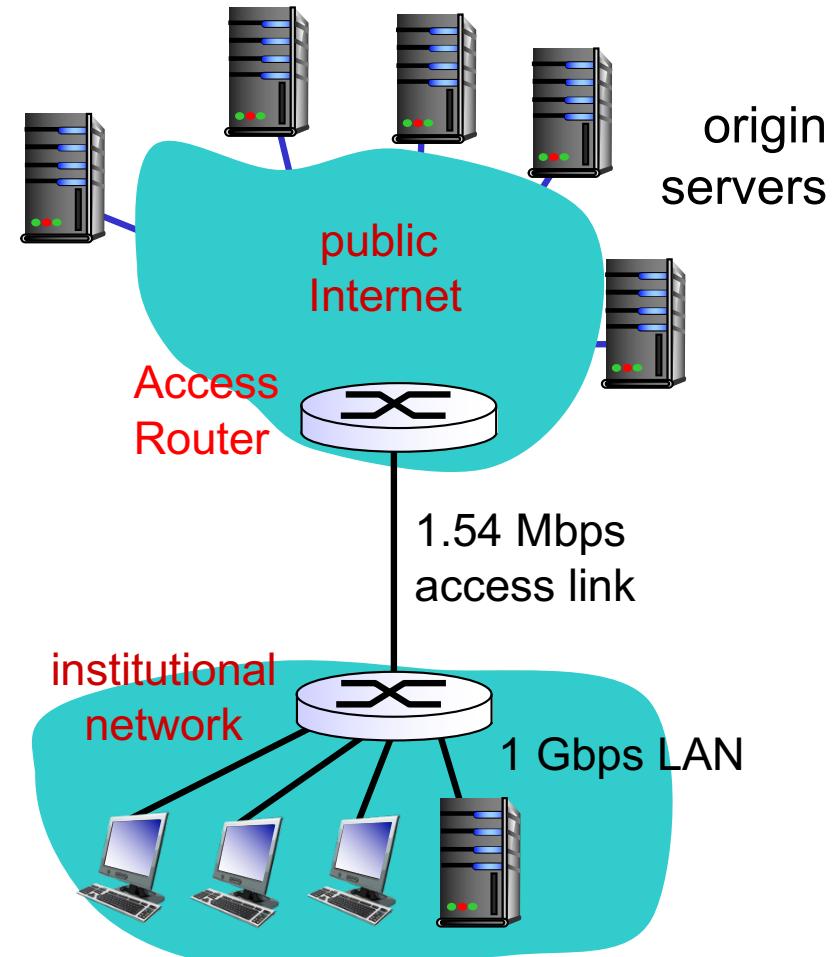
# Caching example:

## *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!*

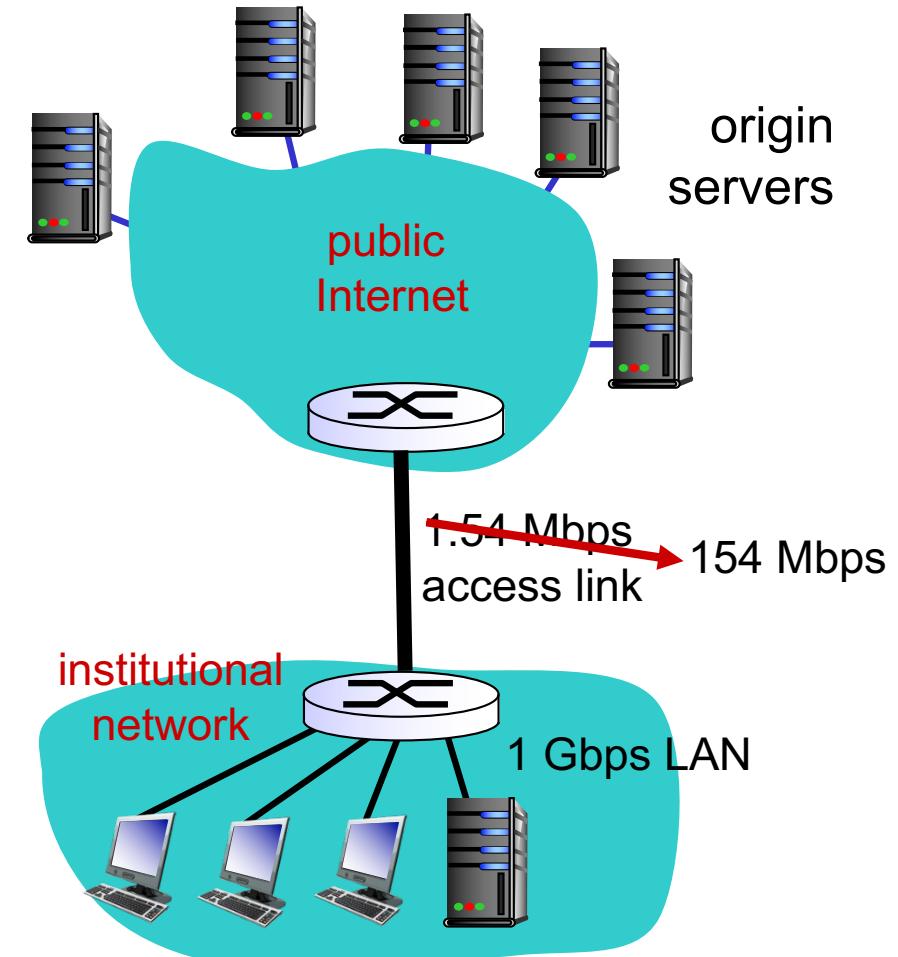
# Caching example: fatter access link

## 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:

- ❖ 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!)

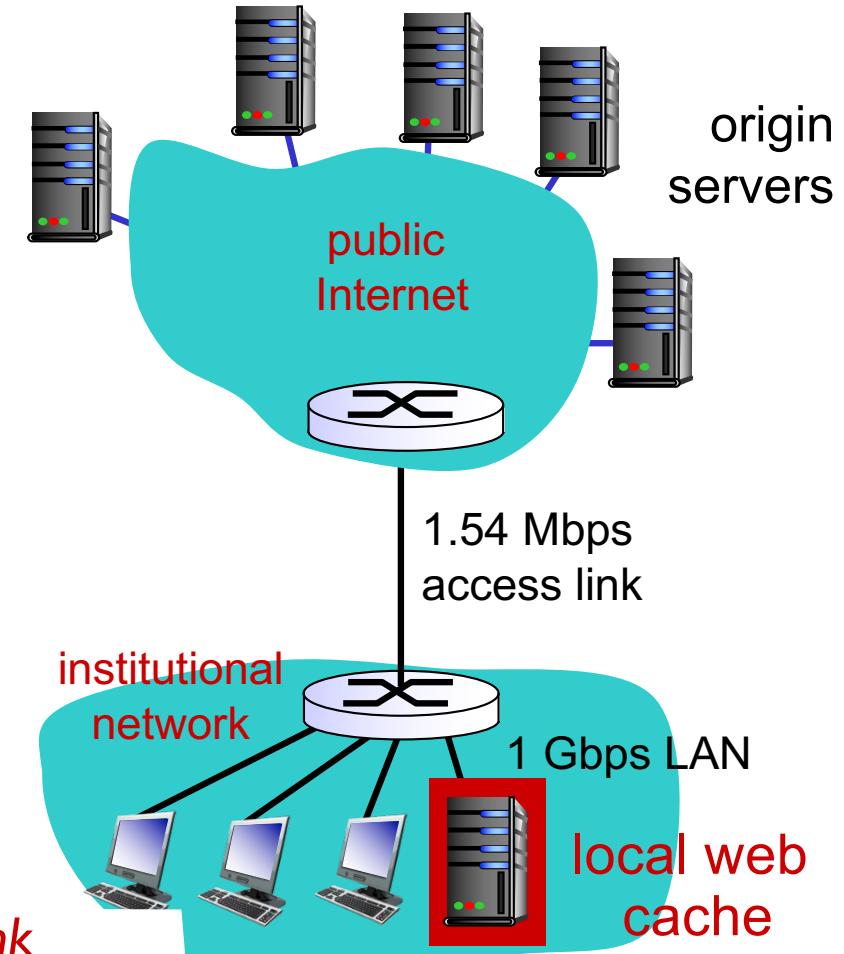
# Caching example: install local cache

## *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?*

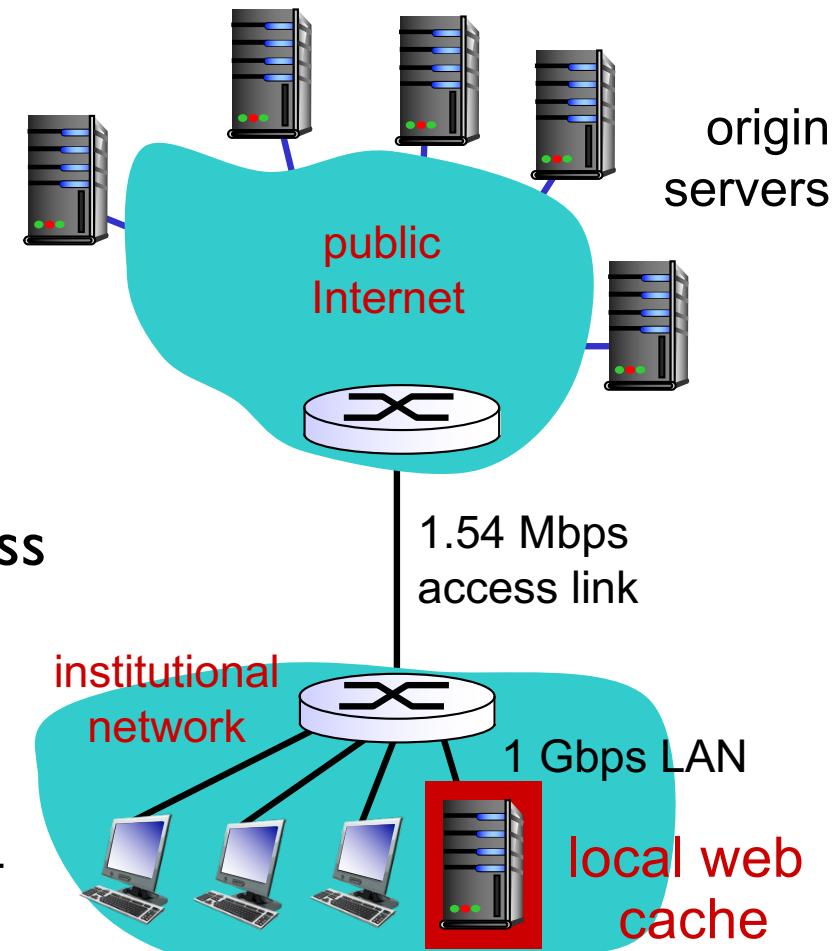


*Cost:* web cache (cheap!)

# 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>**

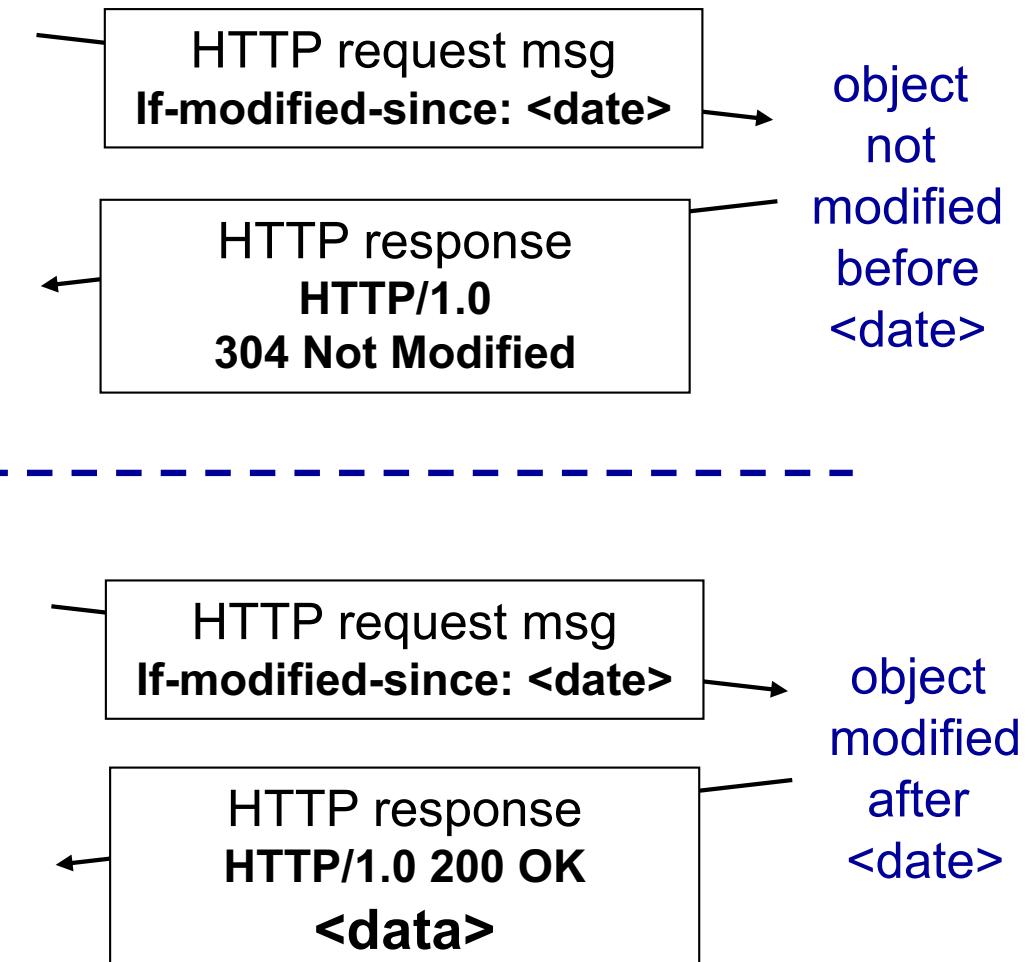
- ❖ **server:** response contains no object if cached copy is up-to-date:

**HTTP/1.0 304 Not Modified**



client

server



# 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

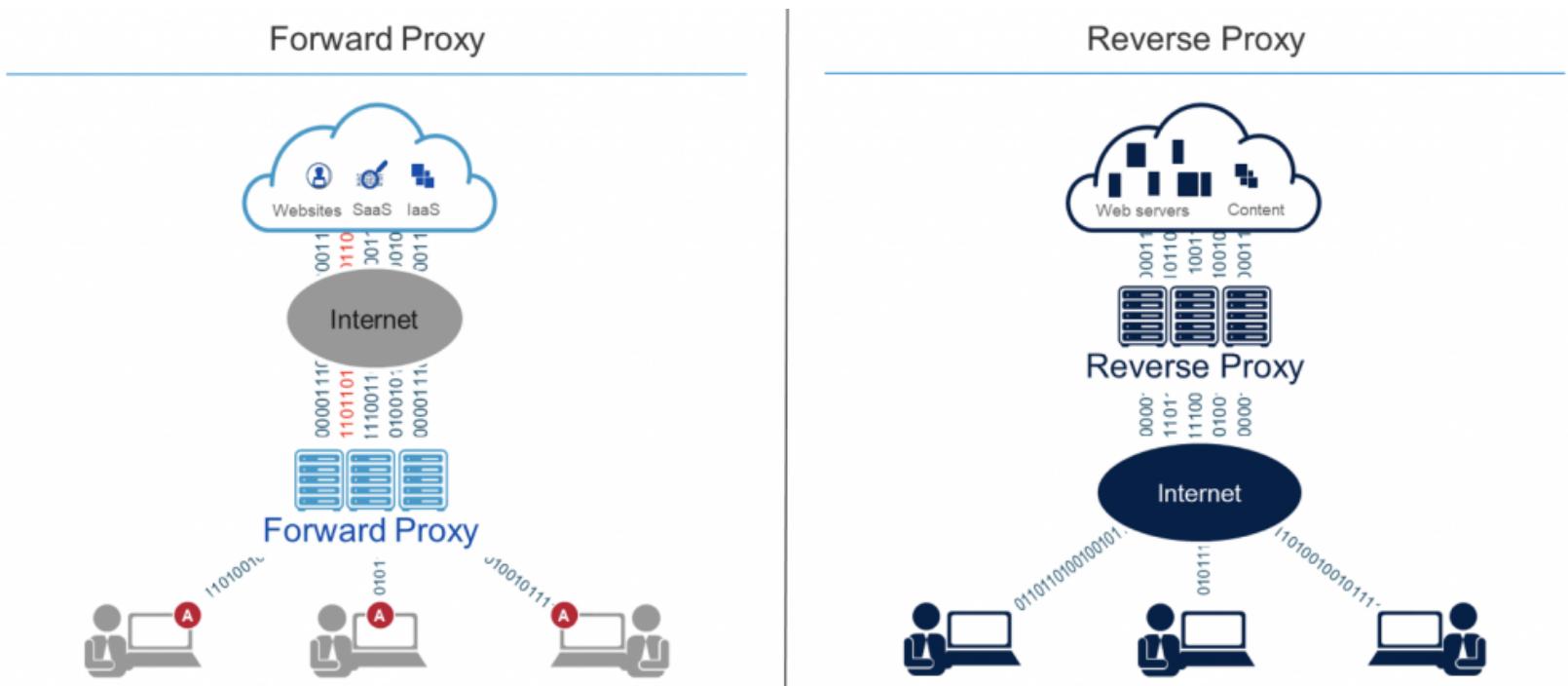
Keep-Alive: timeout=15, max=100

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

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

# Caching: Where?

- Everywhere
  - Client
  - Forward Proxies
  - Reverse Proxies
  - CDN



# 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 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
- Integrate forward and reverse caching functionality
- 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?

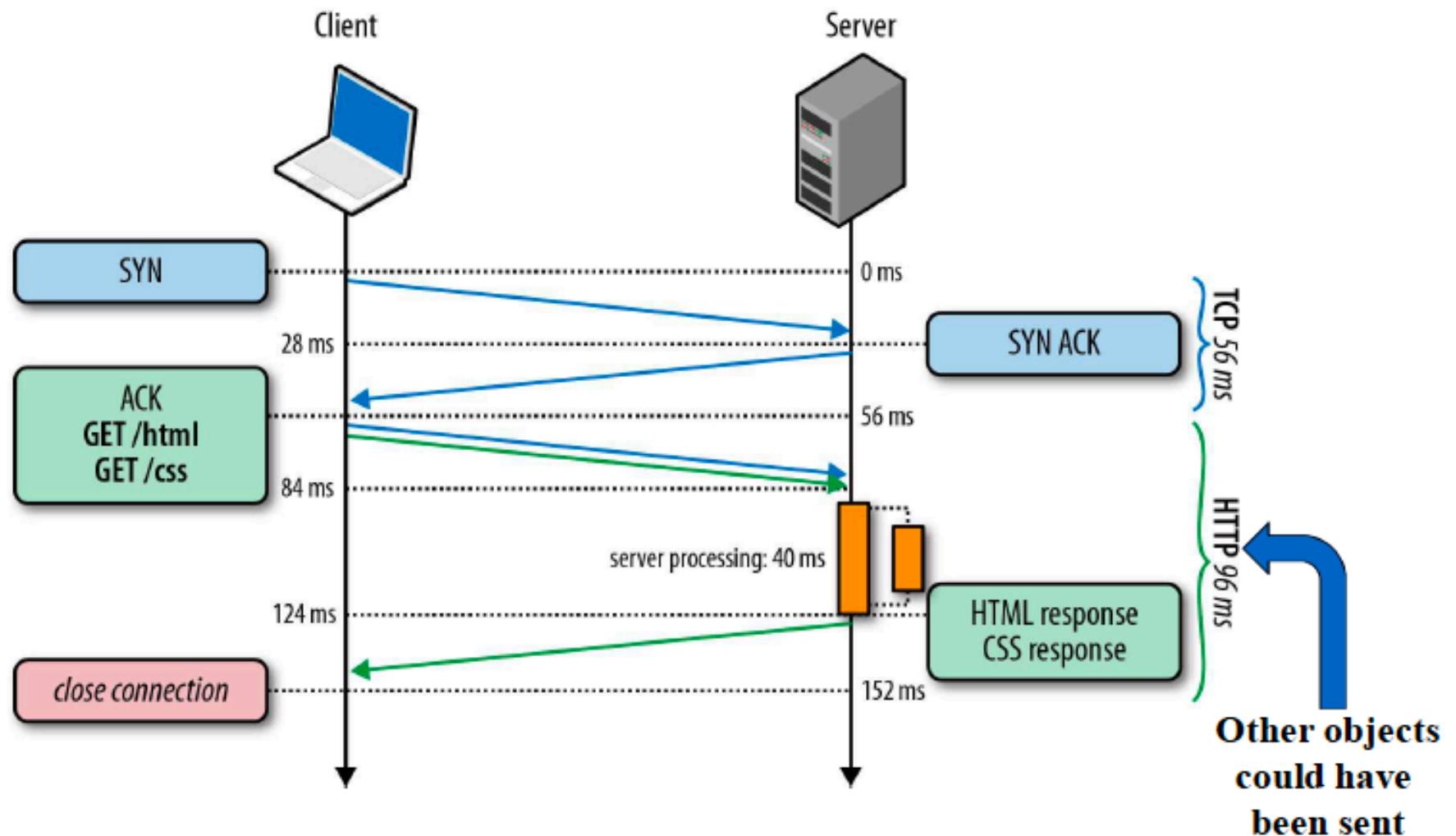
- HTTP is insecure
- HTTP basic authentication: password sent using base64 encoding (can be readily converted to plaintext)
- HTTPS: HTTP over a connection encrypted by Transport Layer Security (TLS)
- Provides:
  - Authentication
  - Bidirectional encryption
- Widely used in place of plain vanilla HTTP



# 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>

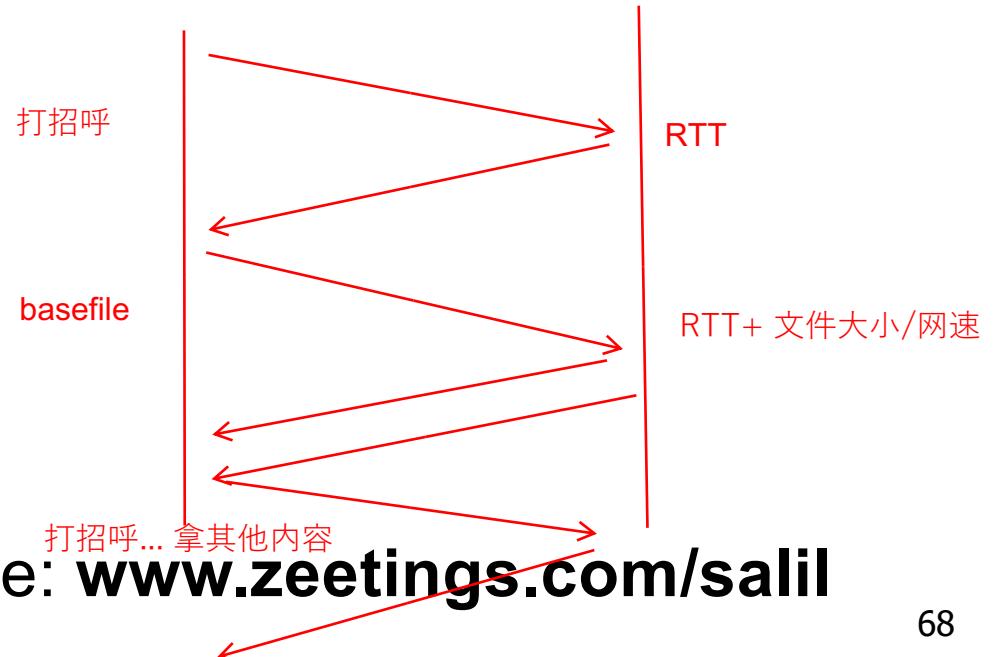


# Quiz: HTTP (1)

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.  $2D + S_0/C + N(D + S/C)$

$$\begin{aligned} \text{Basefile} &= S_0 \\ \text{N inline files} &= N * S \\ \text{Speed} &= C \\ \text{RTT} &= D \\ 2D + S_0/C + N * (2D + S/C) \end{aligned}$$

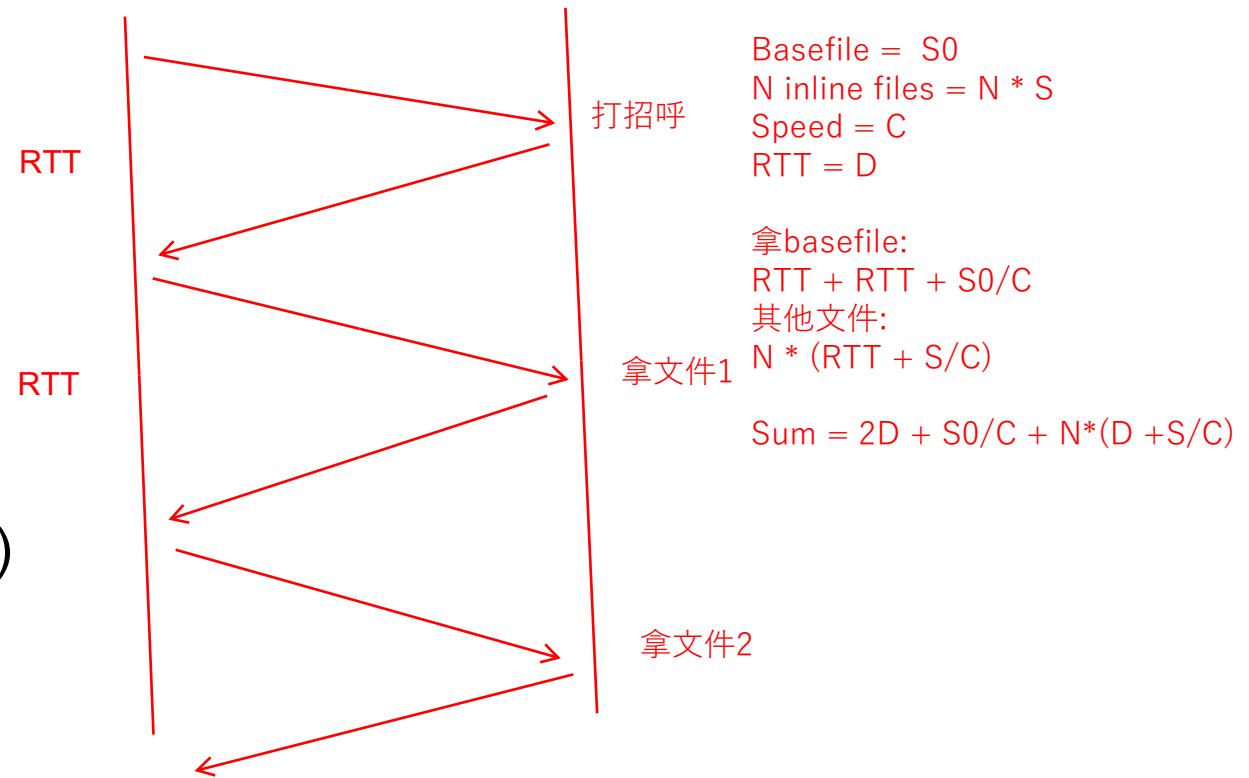




## Quiz: HTTP (2)

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.  $2D + S_0/C + N(D + S/C)$



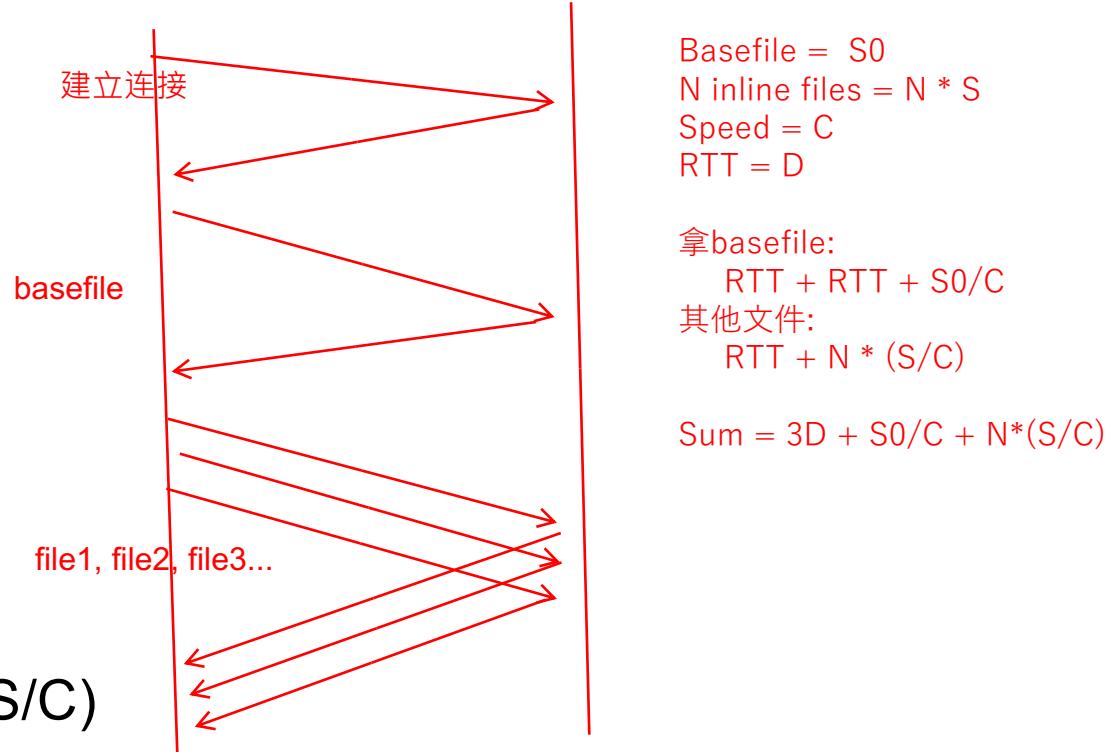
Open a browser and type: [www.zeetings.com/salil](http://www.zeetings.com/salil)



## 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.  $2D + S_0/C + N(D + S/C)$



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, 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

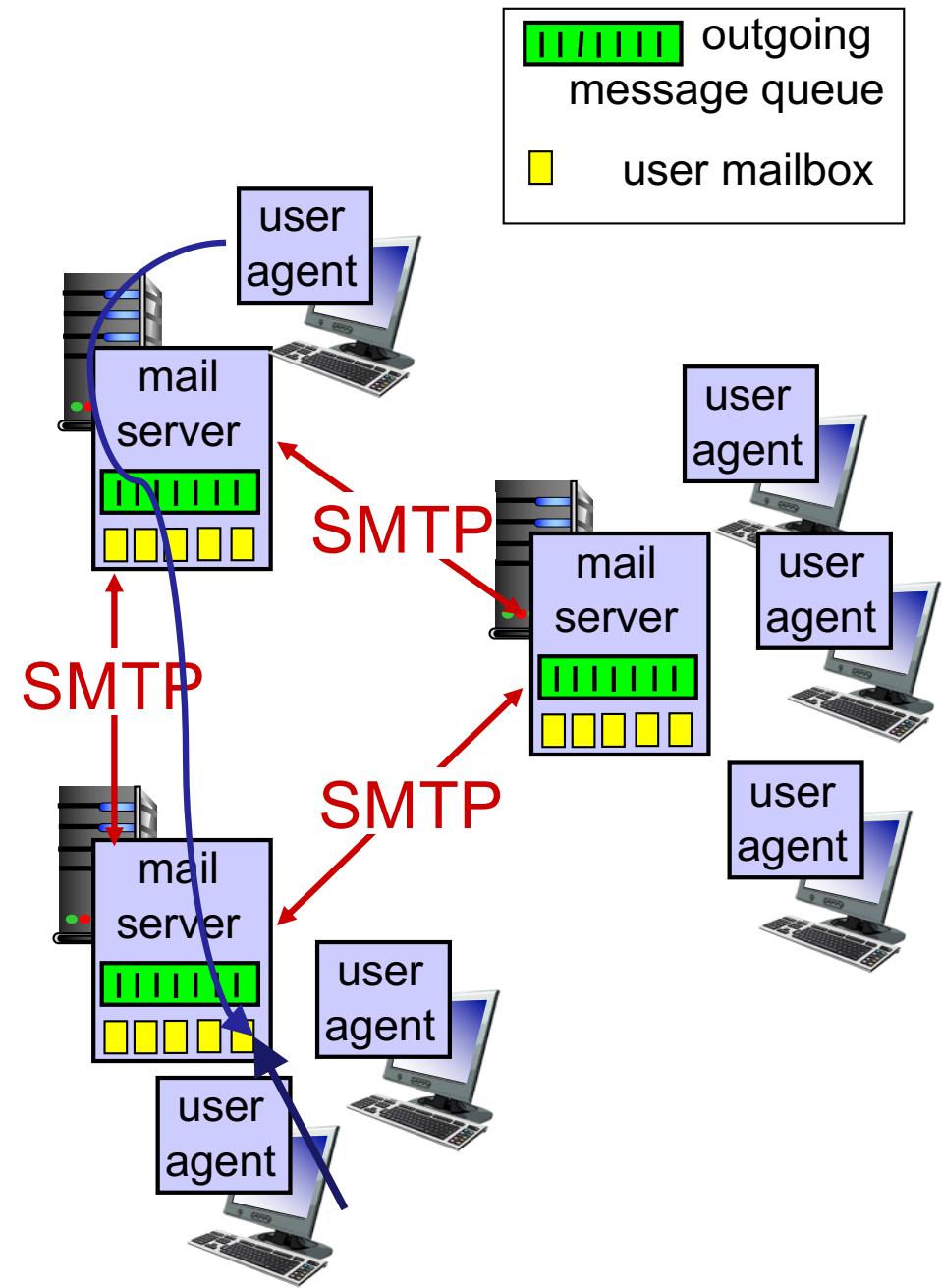
# Electronic mail

*Three major components:*

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

## User Agent

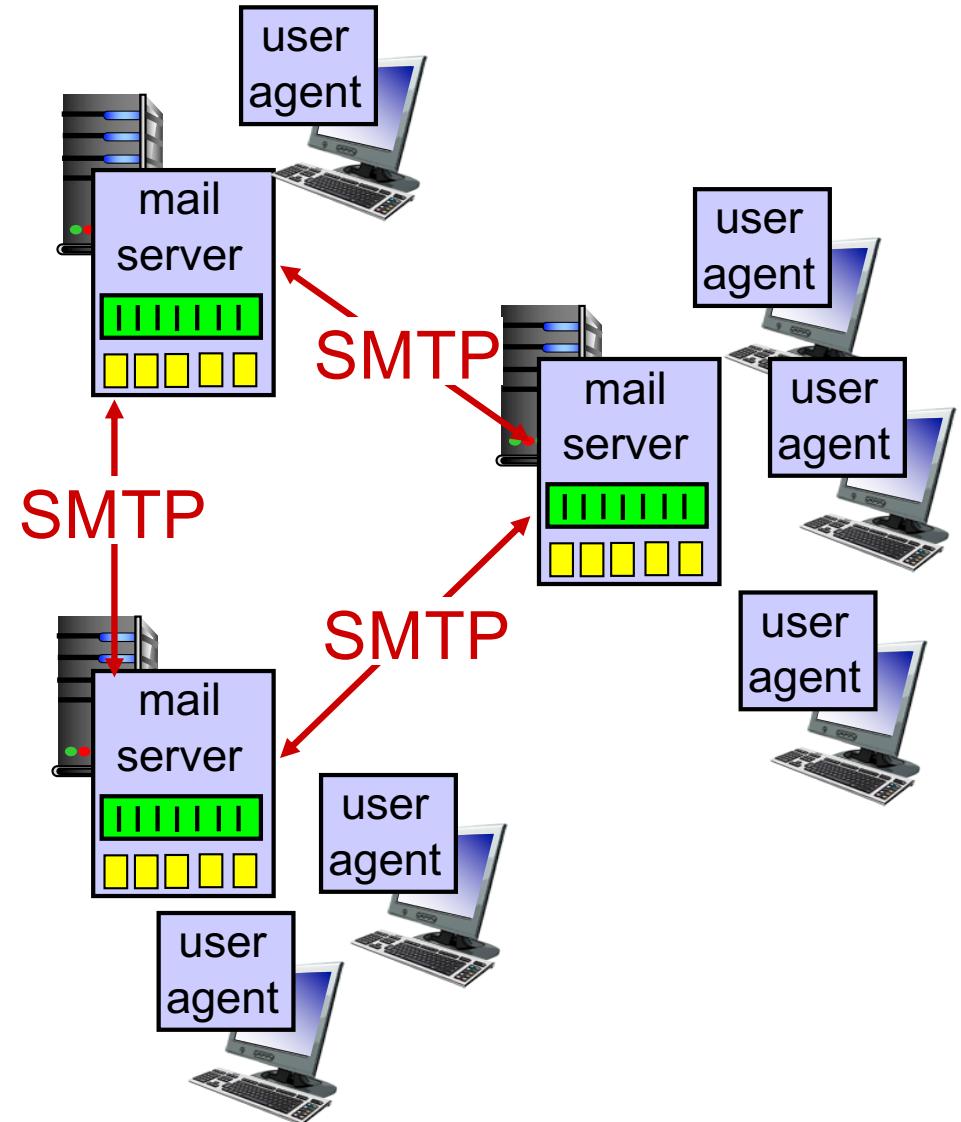
- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server



# 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

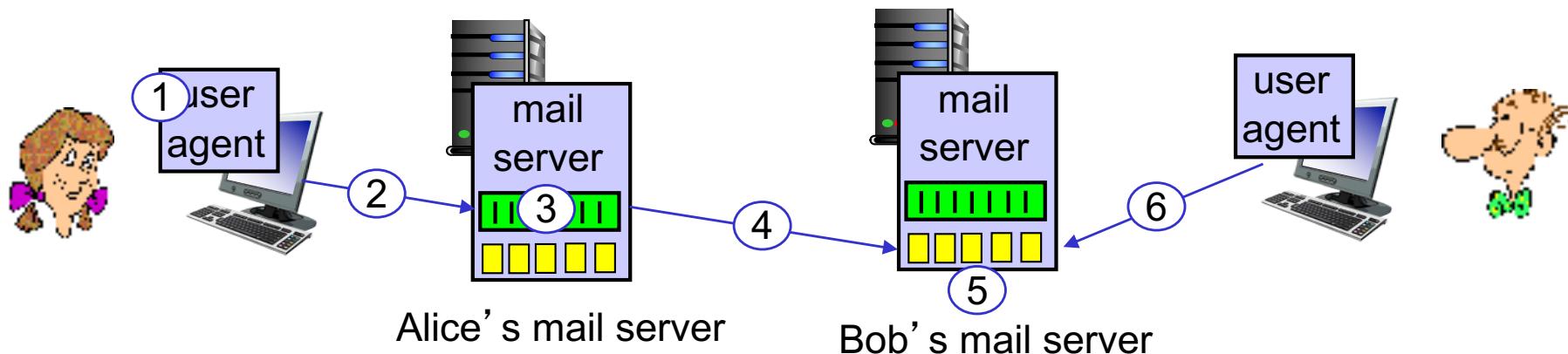


# Electronic Mail: SMTP [RFC 2821]

- ❖ 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

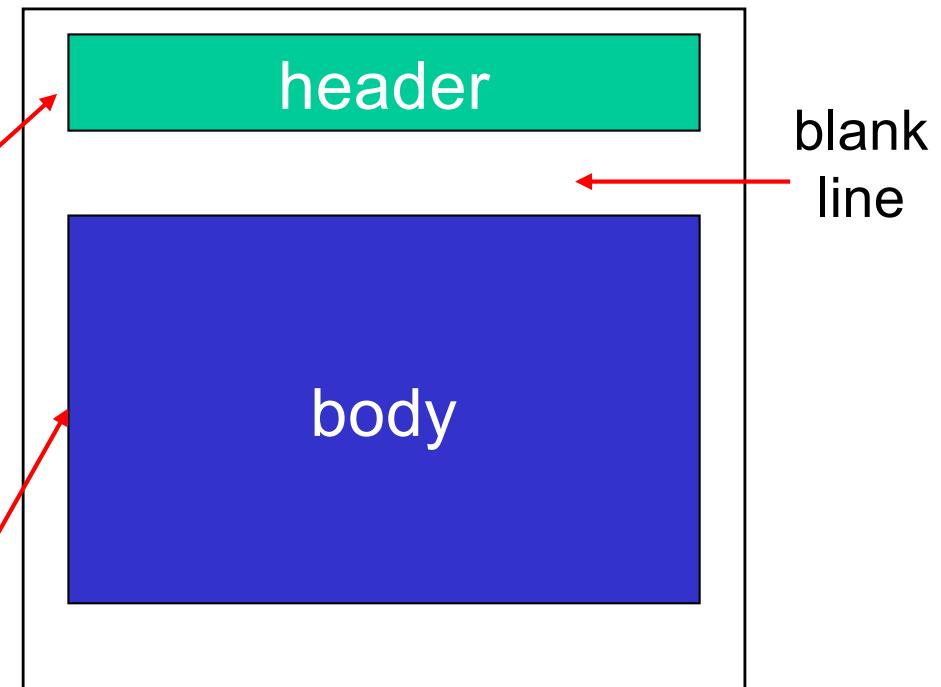
RFC 5322 (822,2822):  
standard for text message  
format (Internet Message  
Format, IMF):

- ❖ header lines, e.g.

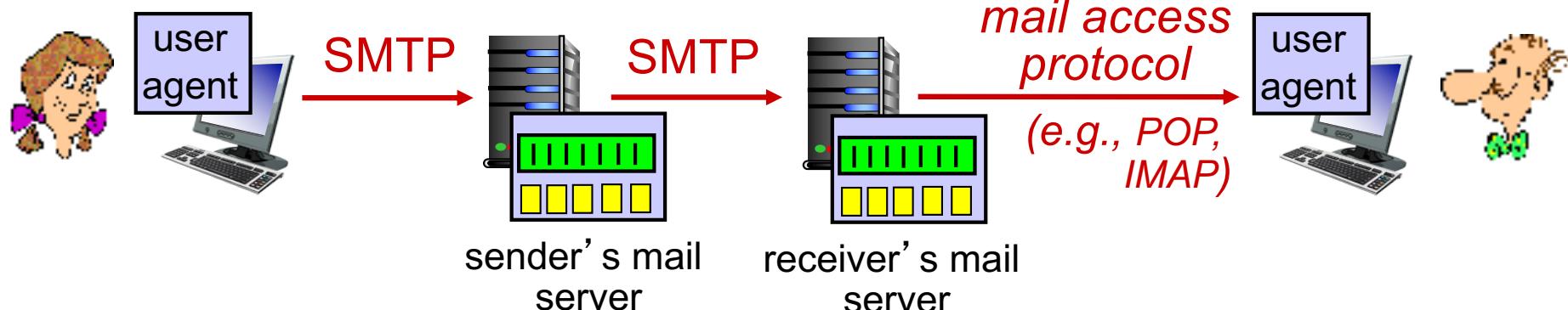
- To:
  - From:
  - Subject:

*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?**

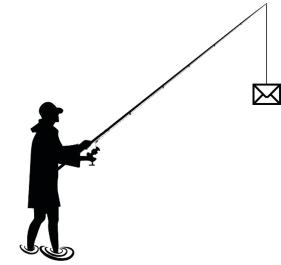
- User agent can directly connect with recipient mail server without the need of sender's mail server? What's the catch?

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

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

Open a browser and type: **www.zeetings.com/salil**

# Phishing

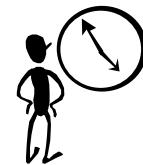


- ❖ Spear phishing
  - Phishing attempts directed at specific individuals or companies
  - Attackers may gather personal information (social engineering) about their targets to increase their probability of success
  - Most popular and accounts for over 90% of attacks
- ❖ Clone phishing
  - A type of phishing attack whereby a legitimate, and previously delivered email containing an attachment or link has had its content and recipient address(es) taken and used to create an almost identical or cloned email.
  - The attachment or link within the email is replaced with a malicious version and then sent from an email address spoofed to appear to come from the original sender.



# 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**

# 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**

## 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

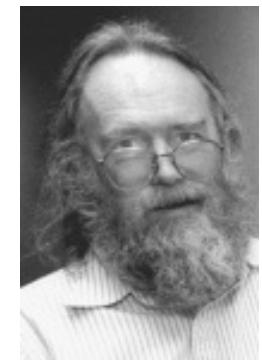
*Q:* how to map between IP address and name, and vice versa ?

*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

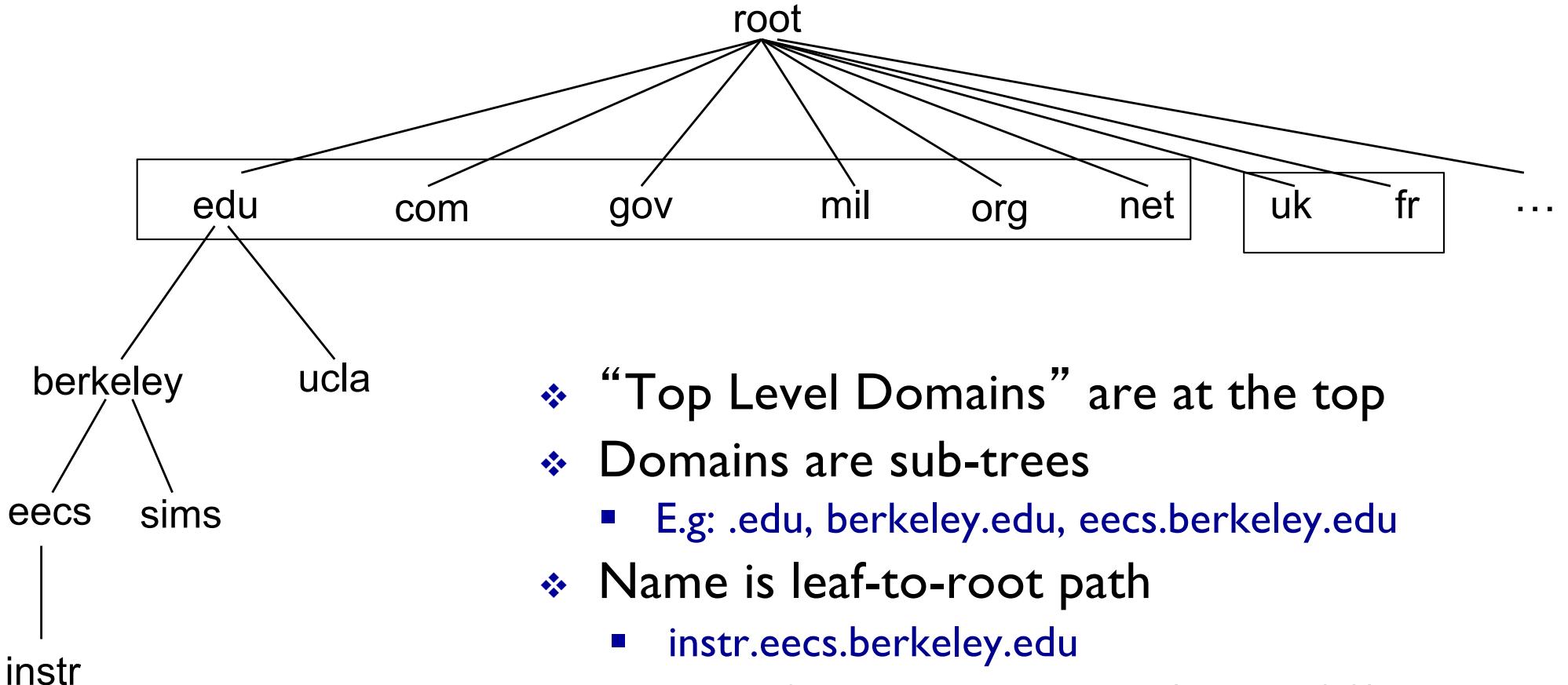
- ❖ No naming conflicts (uniqueness)
- ❖ Scalable
  - many names
  - (secondary) frequent updates
- ❖ Distributed, autonomous administration
  - Ability to update my own (domains') names
  - Don't have to track everybody's updates
- ❖ Highly available
- ❖ Lookups should be fast

# Key idea: Hierarchy

Three intertwined hierarchies

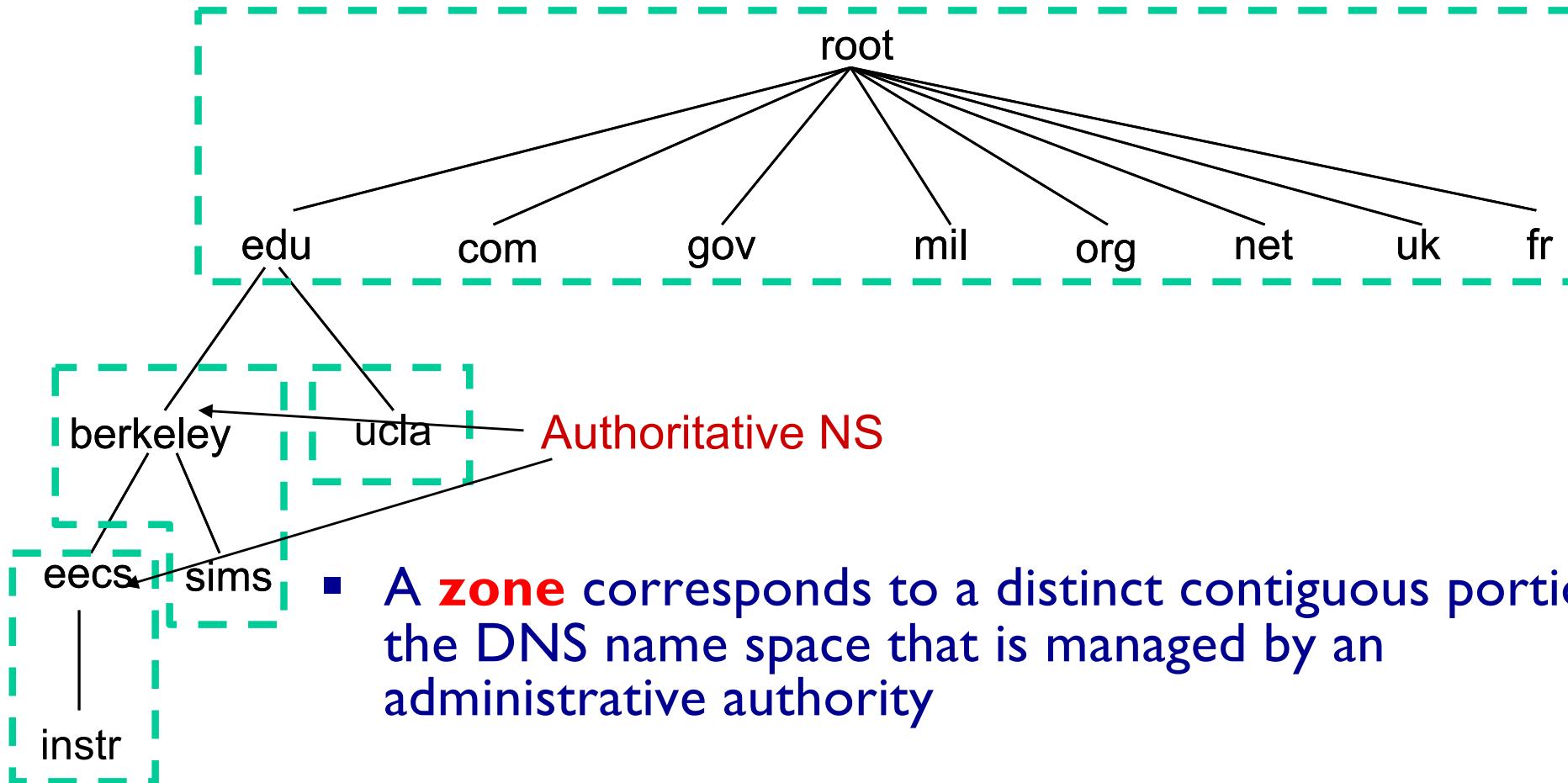
- Hierarchical namespace
  - As opposed to original flat namespace
- Hierarchically administered
  - As opposed to centralised
- (Distributed) hierarchy of servers
  - As opposed to centralised storage

# 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

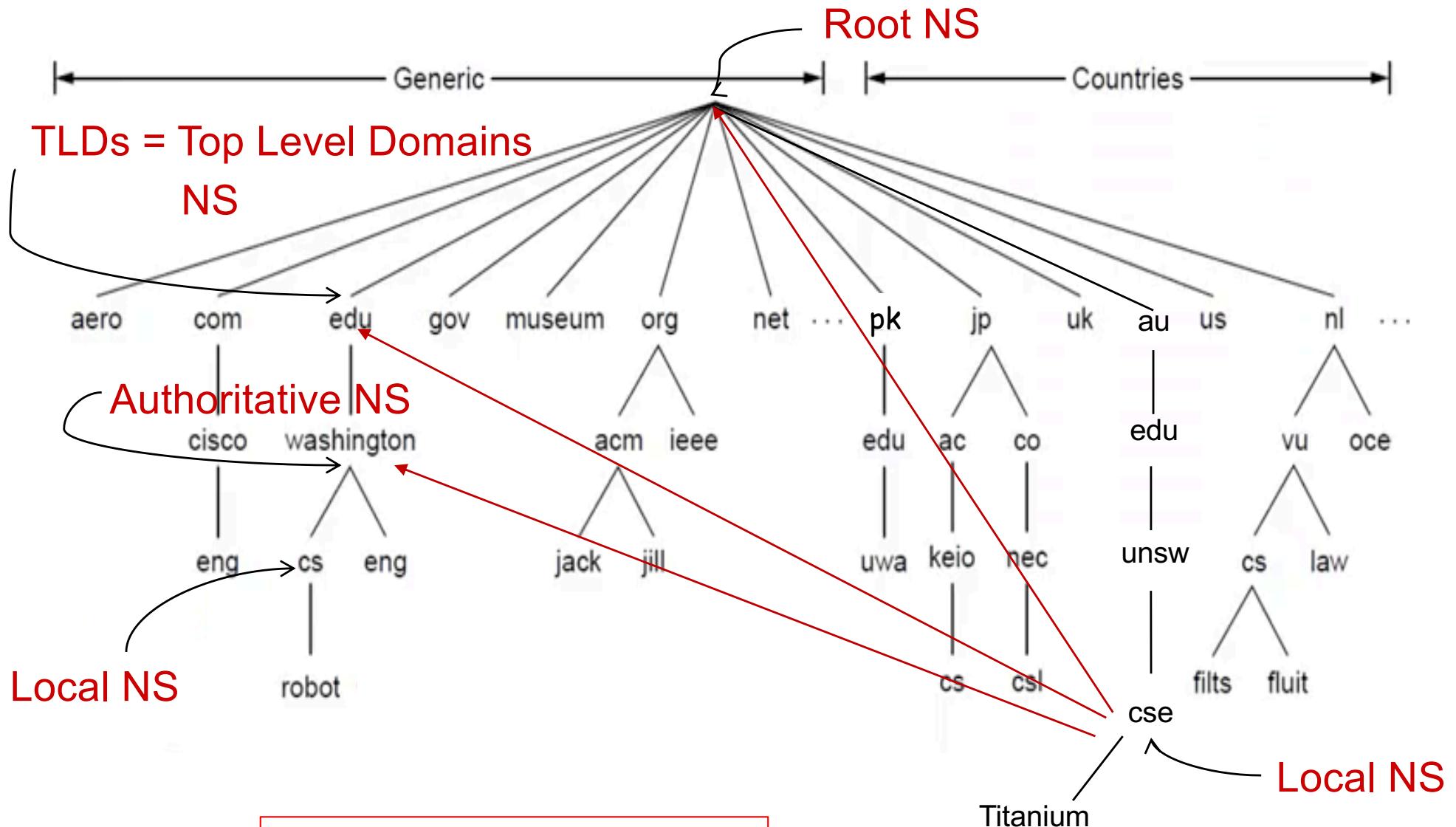
# Server Hierarchy

- ❖ 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
  - Actually store the name-to-address mapping
  - Maintained by the corresponding administrative authority

# Server Hierarchy

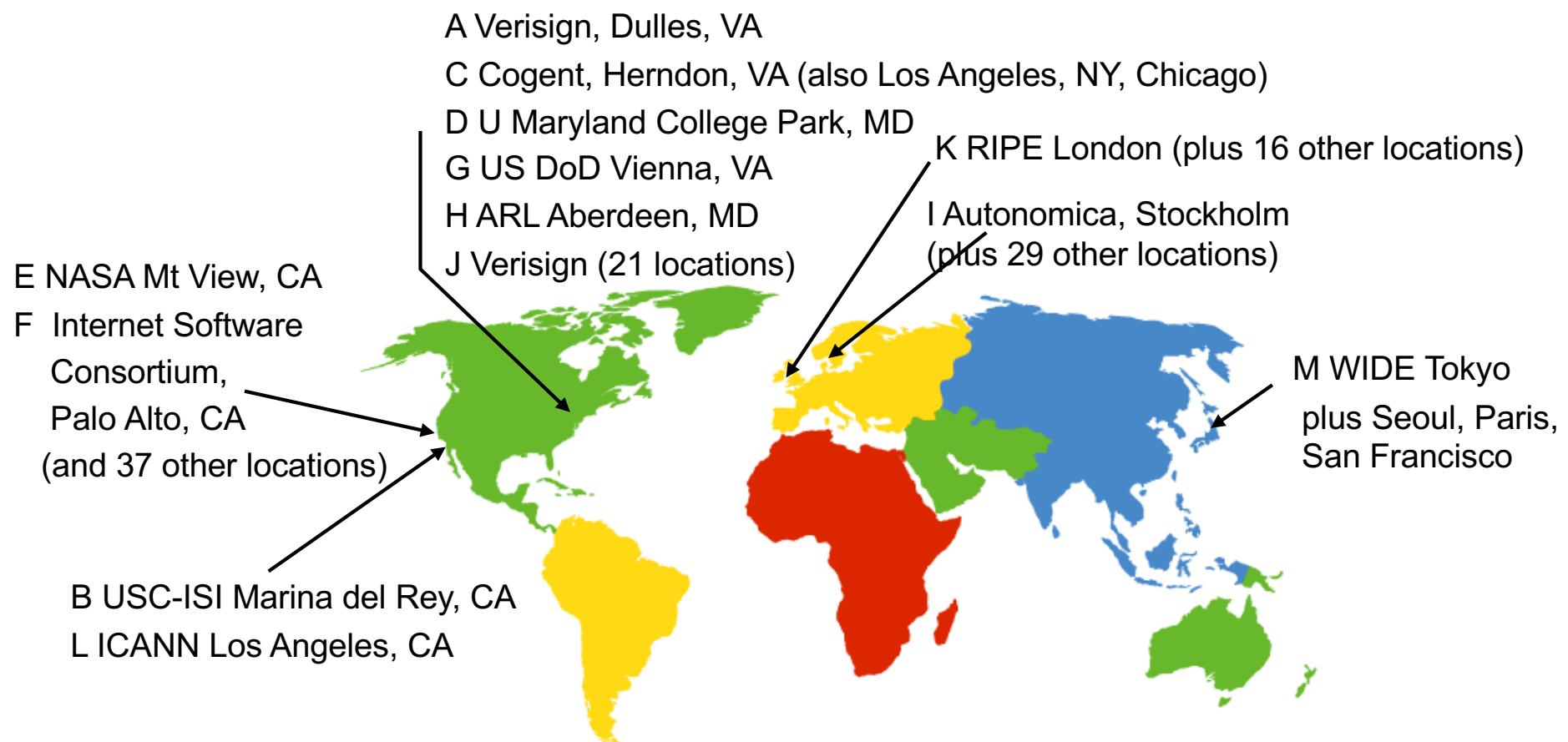
- ❖ Each server stores a (small!) subset of the total DNS database
- ❖ An authoritative DNS server stores “resource records” for all DNS names in the domain that it has authority for
- ❖ Each server can discover the server(s) that are responsible for the other portions of the hierarchy
  - Every server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

# DNS: a distributed, hierarchical database



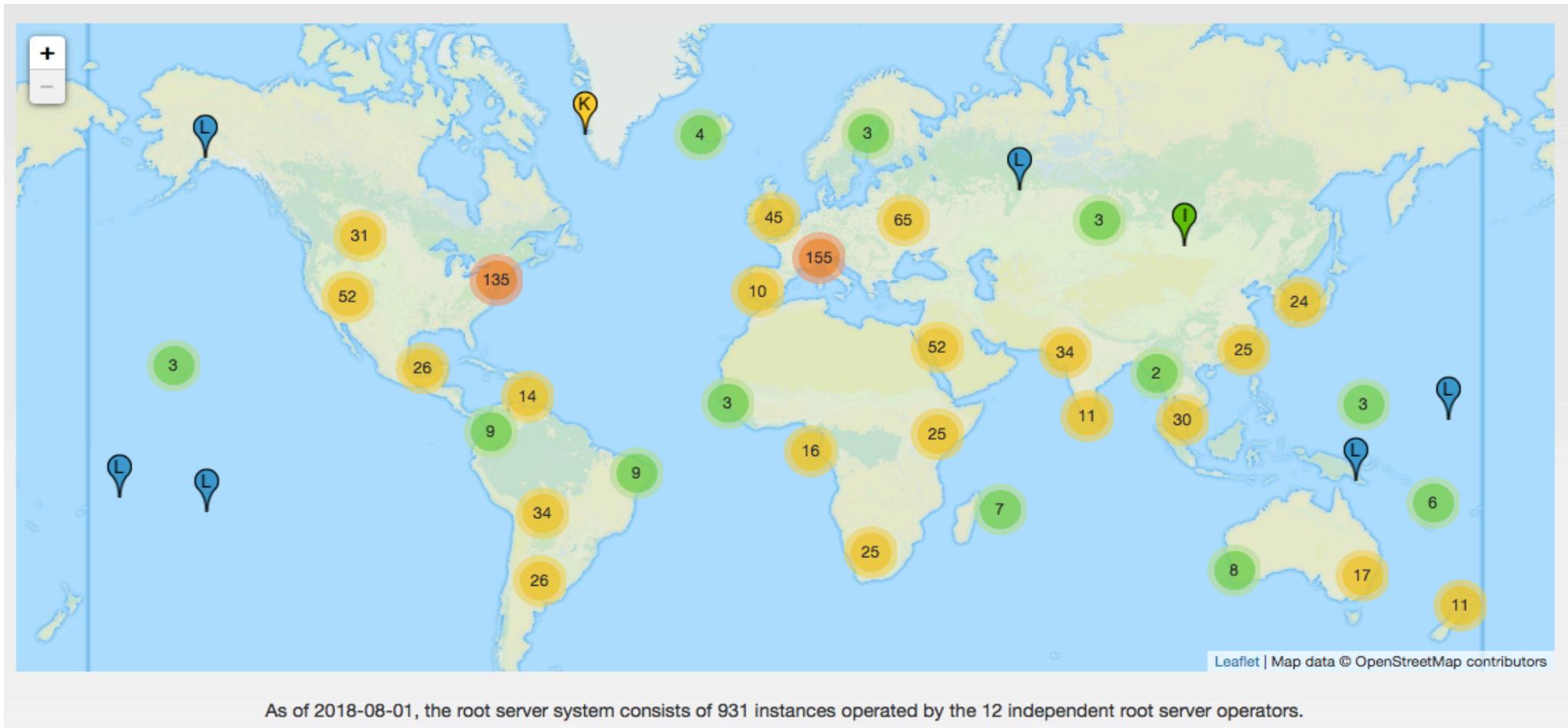
# DNS Root Servers

- 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)



Root Server health: <https://www.ultratools.com/tools/dnsRootServerSpeed>

# 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

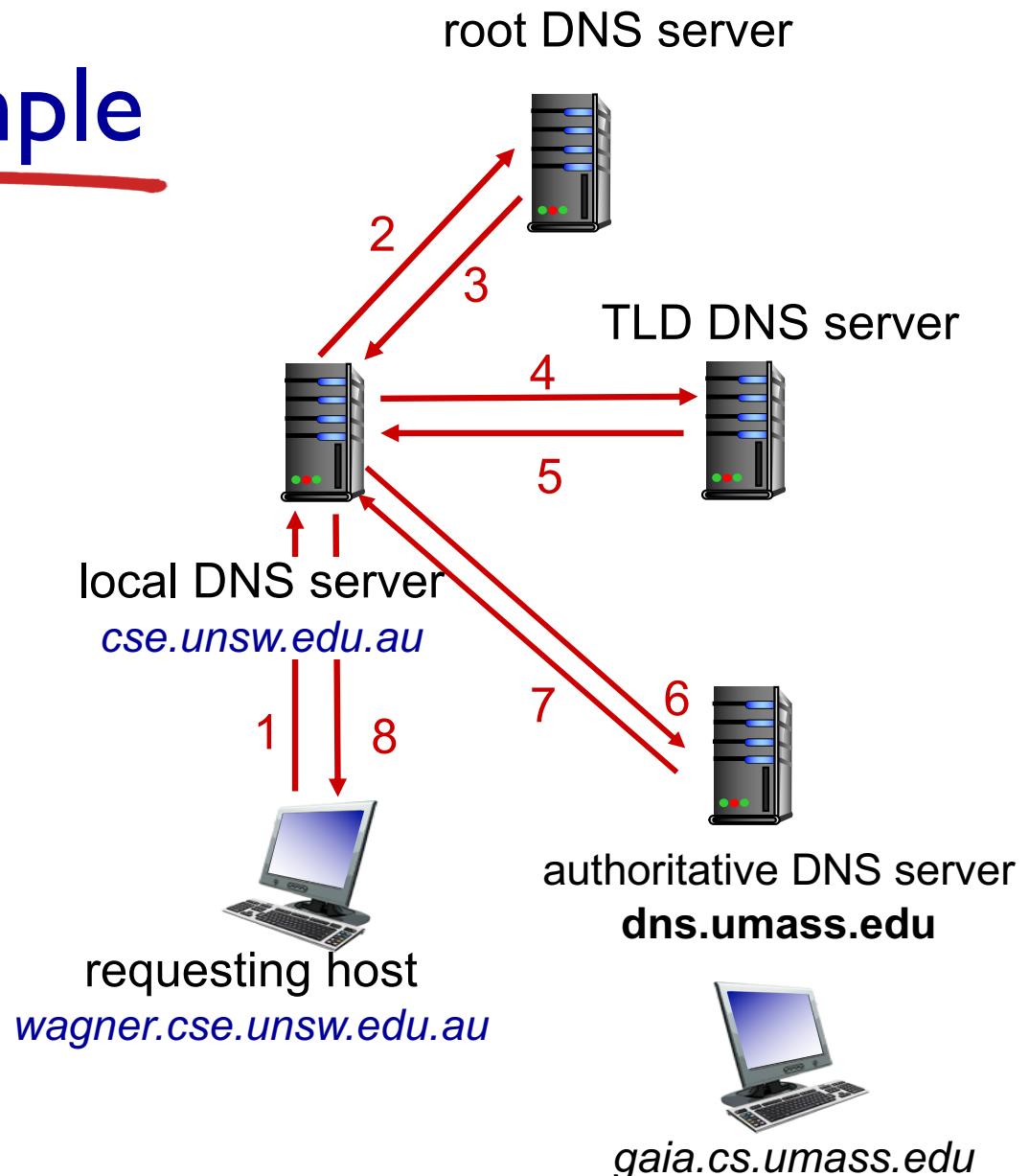
- ❖ does not strictly belong to hierarchy
- ❖ each ISP (residential ISP, company, university) has one
  - also called “default name server”
- ❖ Hosts configured with local DNS server address (e.g.,  
`/etc/resolv.conf`) or learn server via a host configuration protocol (e.g., DHCP)
- ❖ Client application
  - Obtain DNS name (e.g., from URL)
  - Do `getaddrinfo()` 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:*

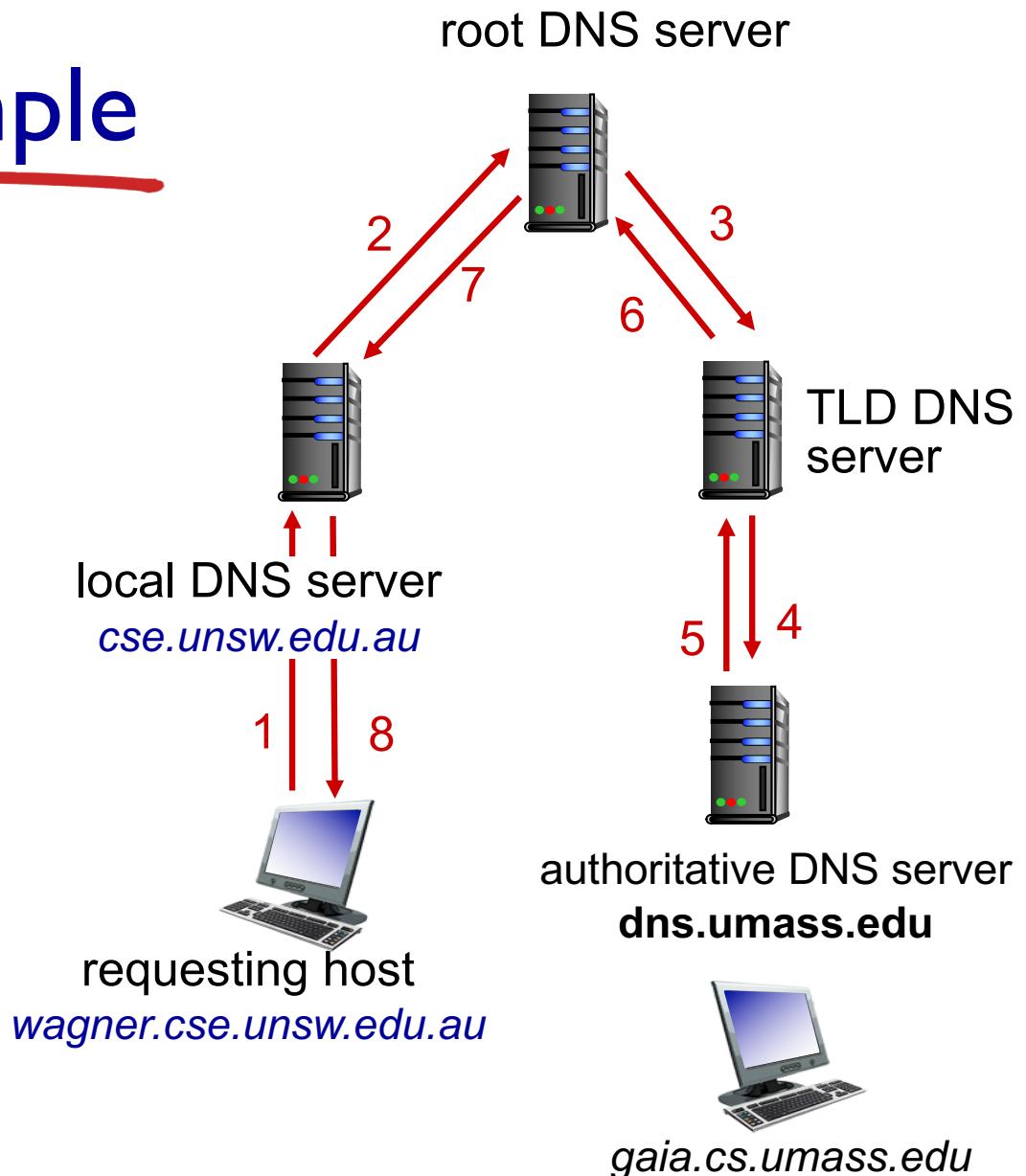
- ❖ contacted server replies with name of server to contact
- ❖ “I don’t know this name, but ask this server”



# 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

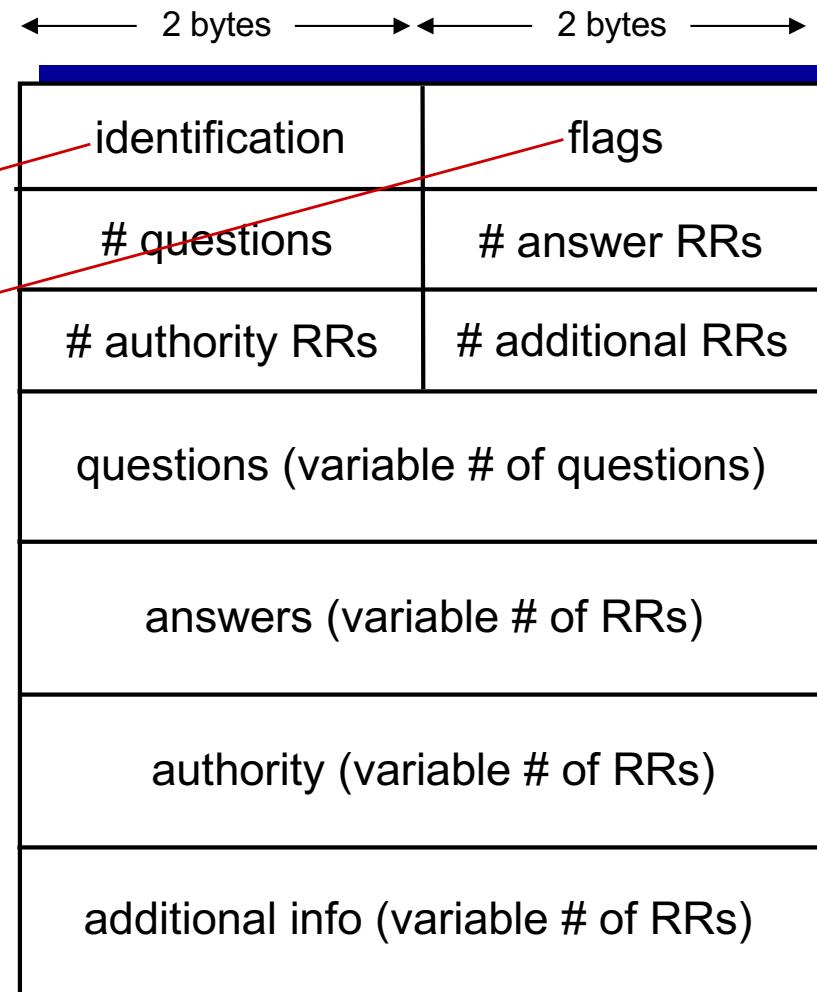
- **value** is name of mailserver  
associated with **name**

# DNS protocol, messages

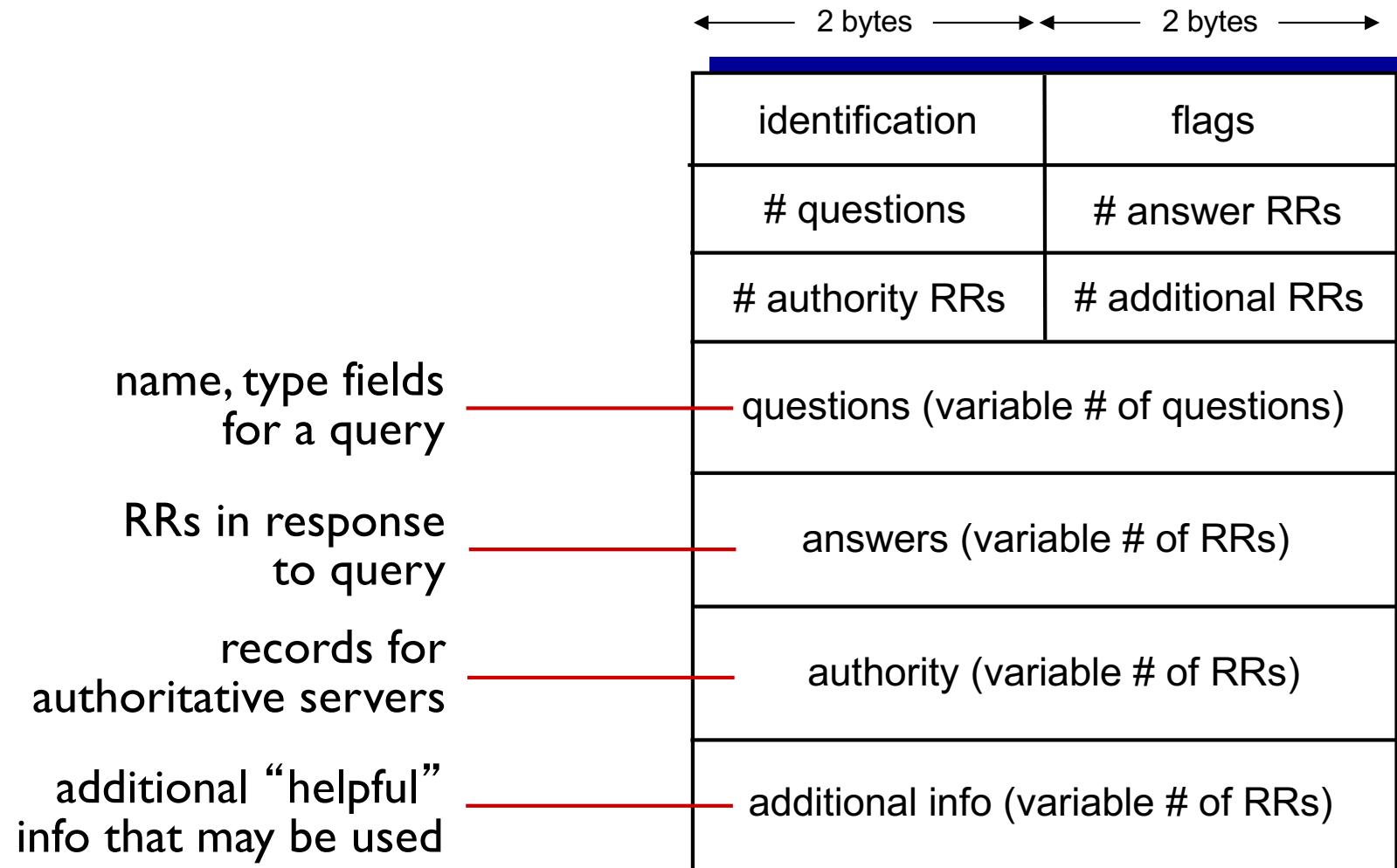
- ❖ *query* and *reply* messages, both with same *message format*

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  
one of the lab

# 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: ??

# Reliability

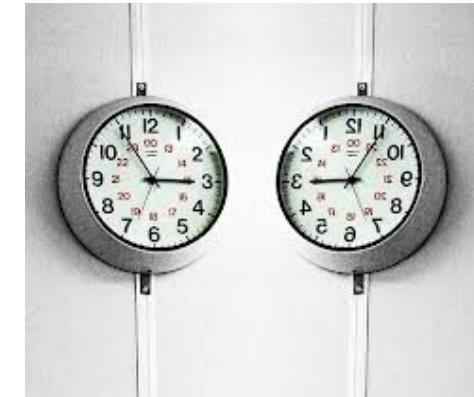
- ❖ DNS servers are **replicated** (primary/secondary)
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- ❖ Usually, UDP used for queries
  - Need reliability: must implement this on top of UDP
  - Spec supports TCP too, but not always implemented
- ❖ 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!

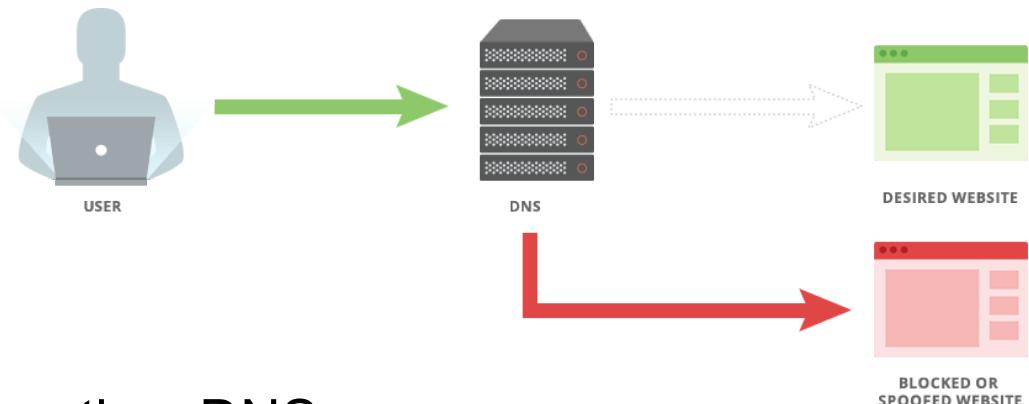
# Reverse DNS

- ❖ 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>



# Schneier on Security

**Blog**

Newsletter

Books

Essays

News

Talks

Academic

About Me

[Blog >](#)

## IoT Attack Against a University Network

Verizon's *Data Brief Digest 2017* describes [an attack](#) against an unnamed university by attackers who hacked a variety of IoT devices and had them spam network targets and slow them down:

Analysis of the university firewall identified over 5,000 devices making hundreds of Domain Name Service (DNS) look-ups every 15 minutes, slowing the institution's entire network and restricting access to the majority of internet services.

In this instance, all of the DNS requests were attempting to look up seafood restaurants -- and it wasn't because thousands of students all had an overwhelming urge to eat fish -- but because devices on the network had been instructed to repeatedly carry out this request.

"We identified that this was coming from their IoT network, their vending machines and their light sensors were actually looking for seafood domains; 5,000 discreet systems and they were nearly all in the IoT infrastructure," says Laurance Dine, managing principal of investigative response at Verizon.

The actual Verizon document doesn't appear to be available online yet, but there is an advance version that only discusses the incident above, available [here](#).

Detailed Report at - [http://www.verizonenterprise.com/resources/reports/rp\\_data-breach-digest-2017-sneak-peek\\_xg\\_en.pdf](http://www.verizonenterprise.com/resources/reports/rp_data-breach-digest-2017-sneak-peek_xg_en.pdf)

# 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

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

;; ADDITIONAL SECTION:

google.com 600 IN A 129.45.212.222

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

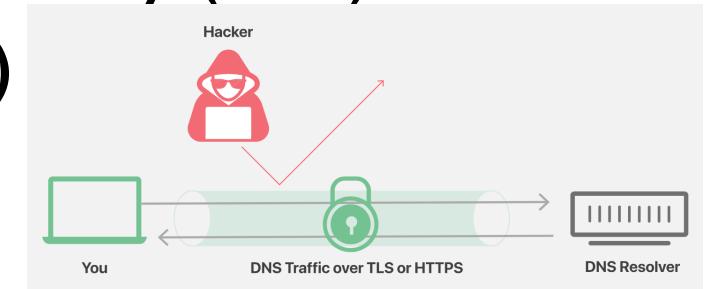
# DNSSEC

---

- ❖ 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

# 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

# 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

# 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

# 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)

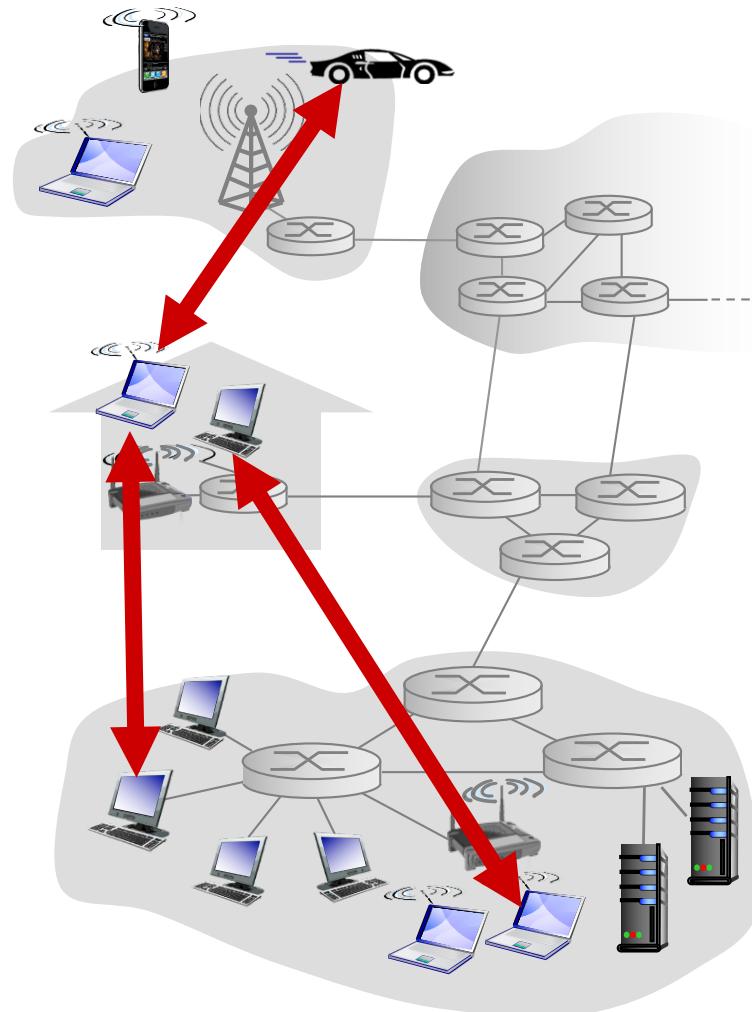
2.7 socket programming with UDP and TCP

# Pure P2P architecture

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

## *examples:*

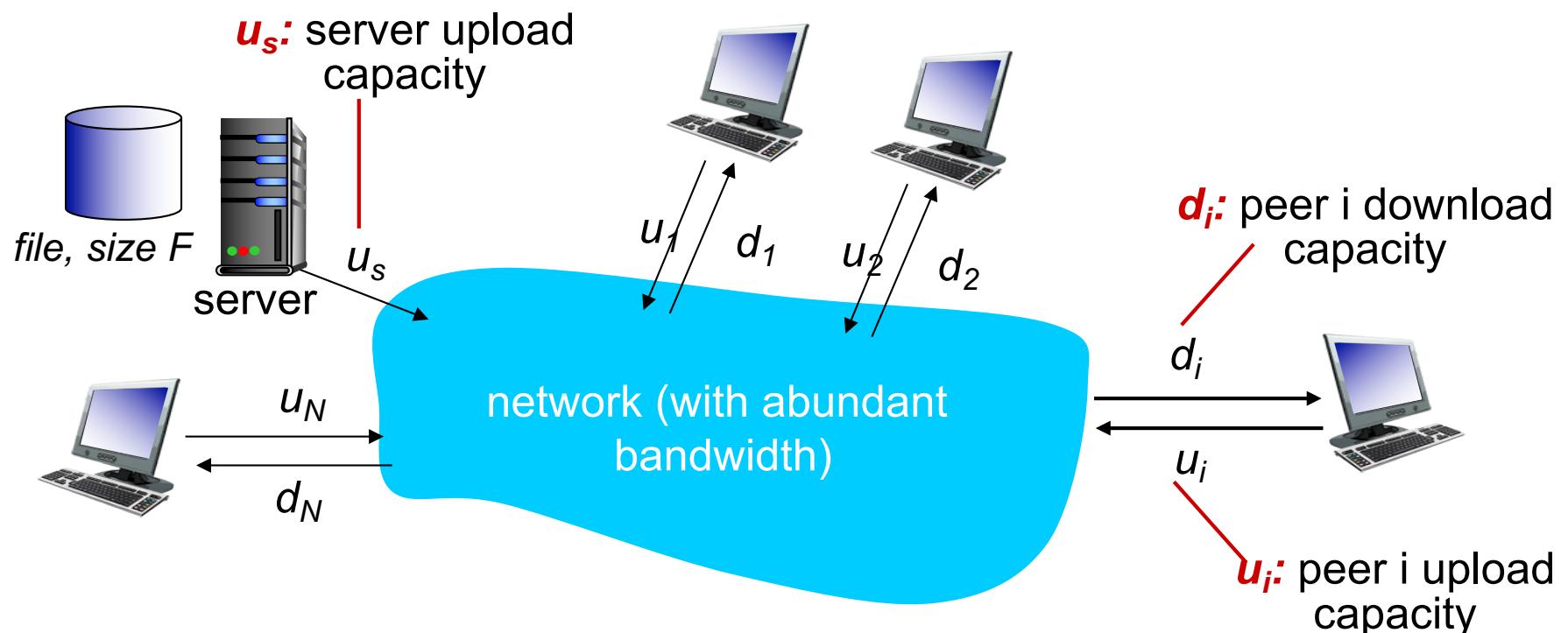
- file distribution  
(BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)
- Cryptocurrency  
(BitCoin)



# File distribution: client-server vs P2P

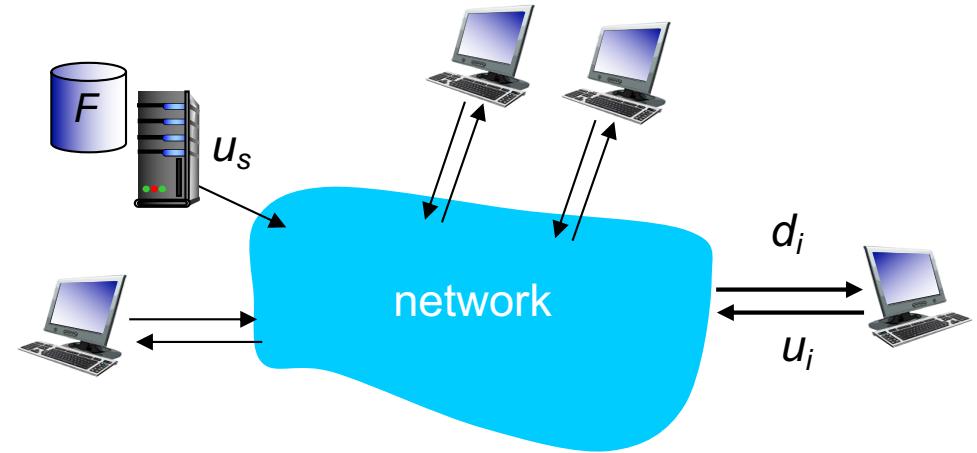
Question: how much time to distribute file (size  $F$ ) from one server to  $N$  peers?

- 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:  $F/d_{\min}$

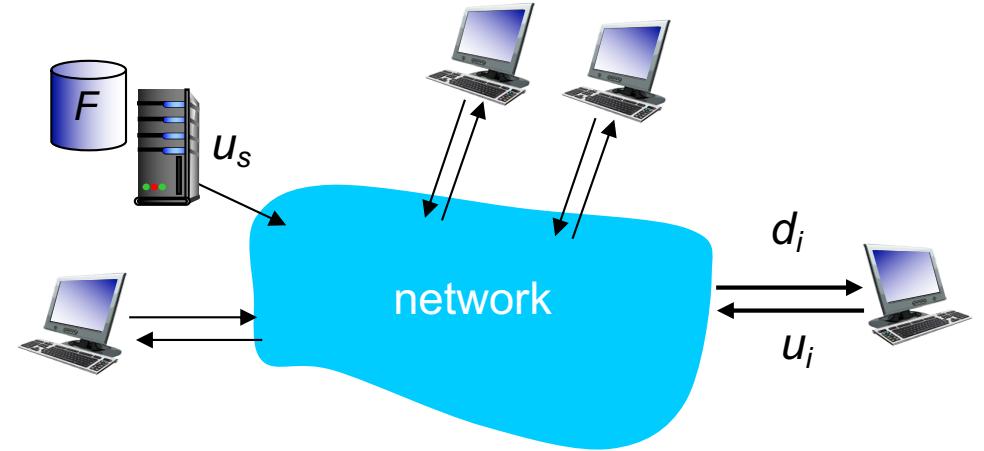


*time to distribute  $F$   
to  $N$  clients using  
client-server approach*       $D_{c-s} \geq \max\{NF/u_s, F/d_{\min}\}$

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  $u_s + \sum u_i$



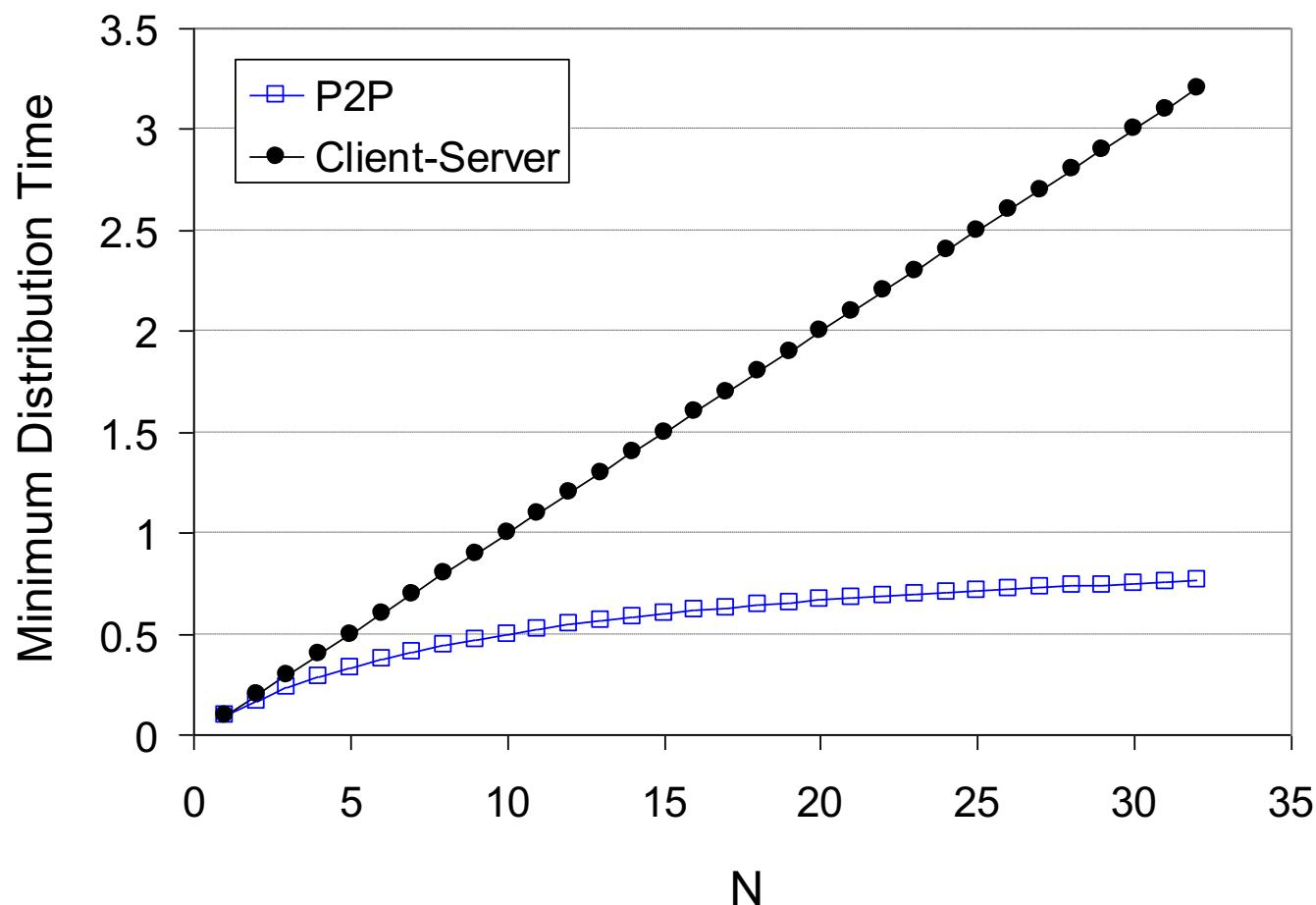
time to distribute  $F$   
to  $N$  clients using  
P2P approach

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

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,  $u_s = 10u$

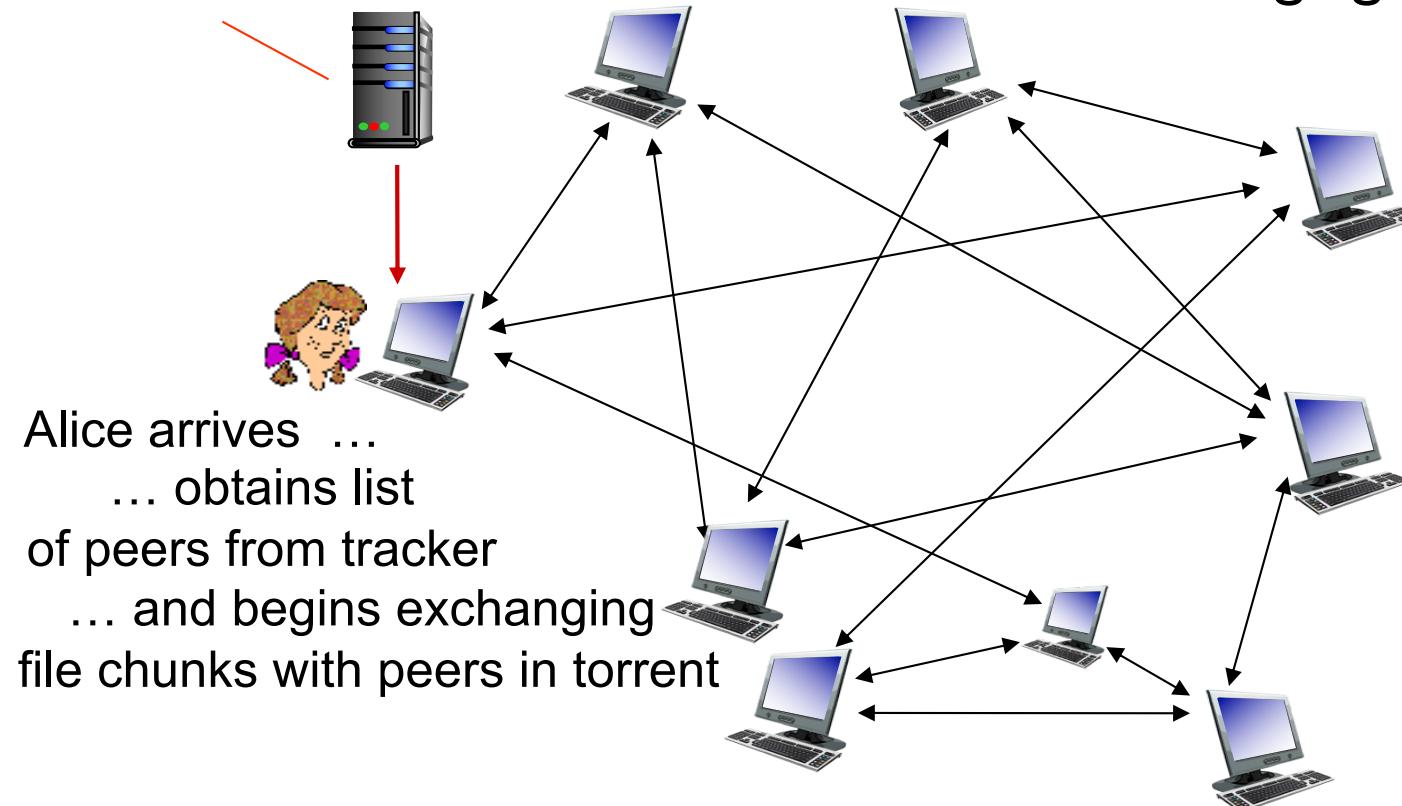


# P2P file distribution: BitTorrent

- ❖ 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

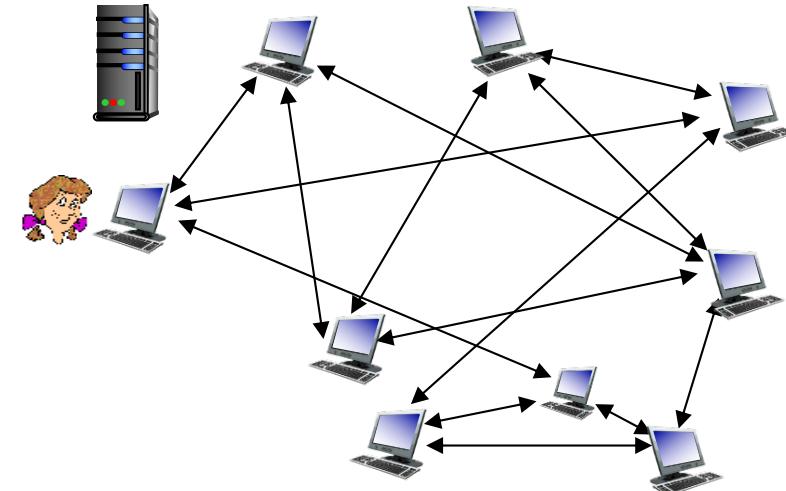
Tracker2-url

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:*

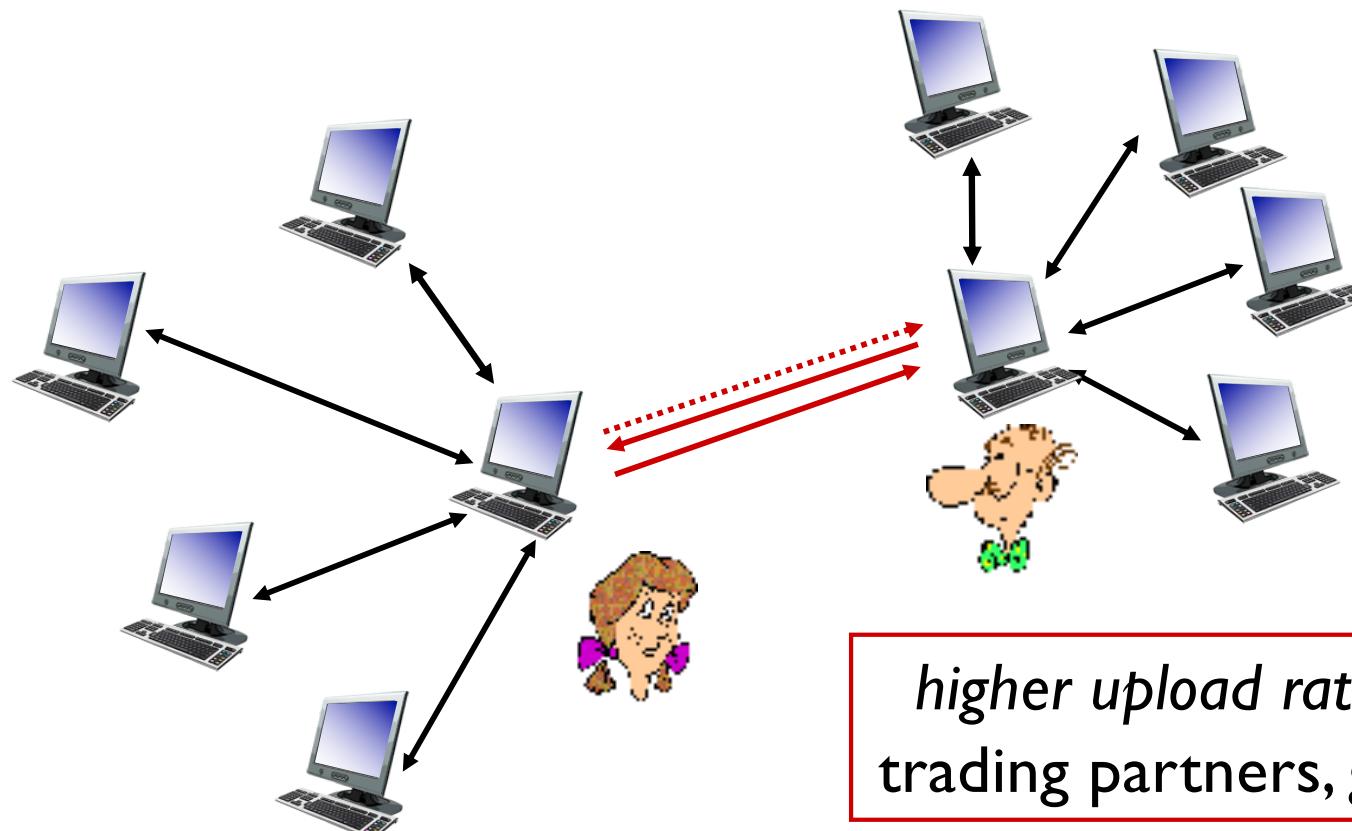
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first
- ❖ **Q:** Why rarest first?

## *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



# Getting rid of the server/tracker

---

- ❖ Distribute the tracker information using a Distributed Hash Table (DHT)
- ❖ A DHT is a lookup structure
  - Maps keys to an arbitrary value
  - Works a lot like, well .... hash table

Content available in 6<sup>th</sup> Edition of the textbook Section 2.6.2

# Hash table - review

- ❖ (key,value) pairs
- ❖ Centralised hash table – all (key, value) pairs on one node
- ❖ Distributed hash tables – each node has a “section” of (key, value) pairs

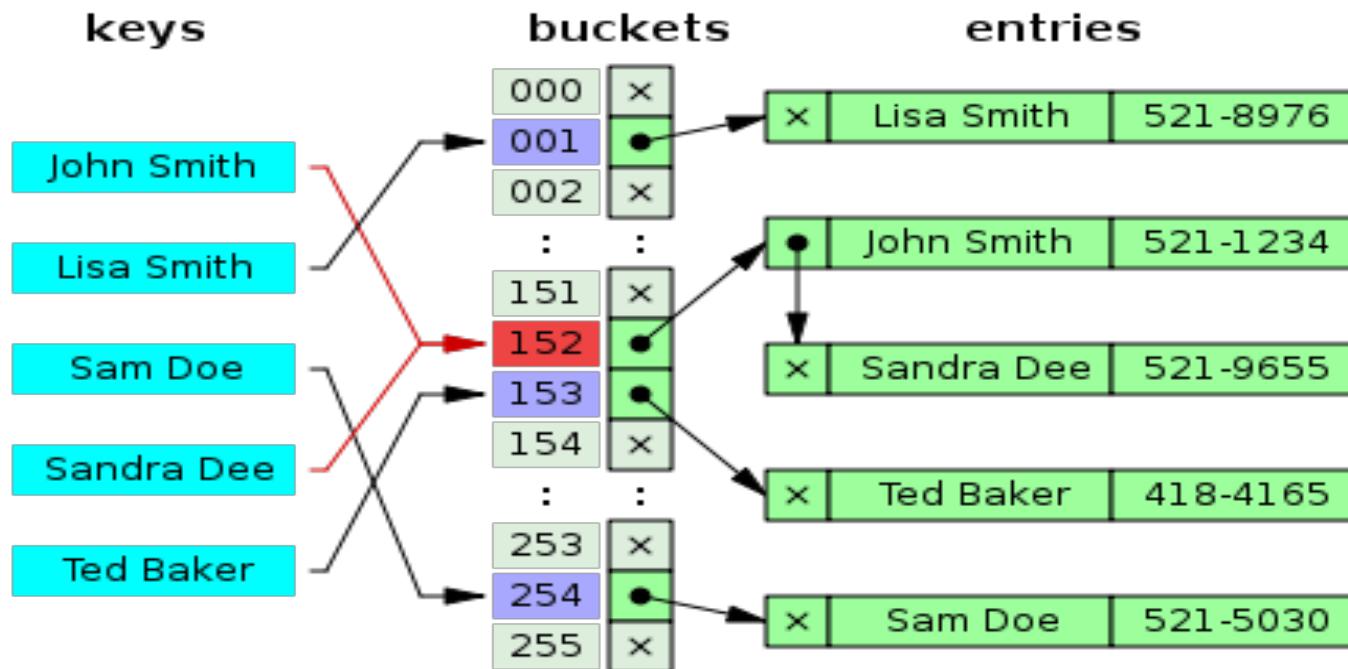


Figure src: [http://en.wikipedia.org/wiki/Hash\\_table](http://en.wikipedia.org/wiki/Hash_table)

# 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: BT tracker(s)
- ❖ 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

# 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

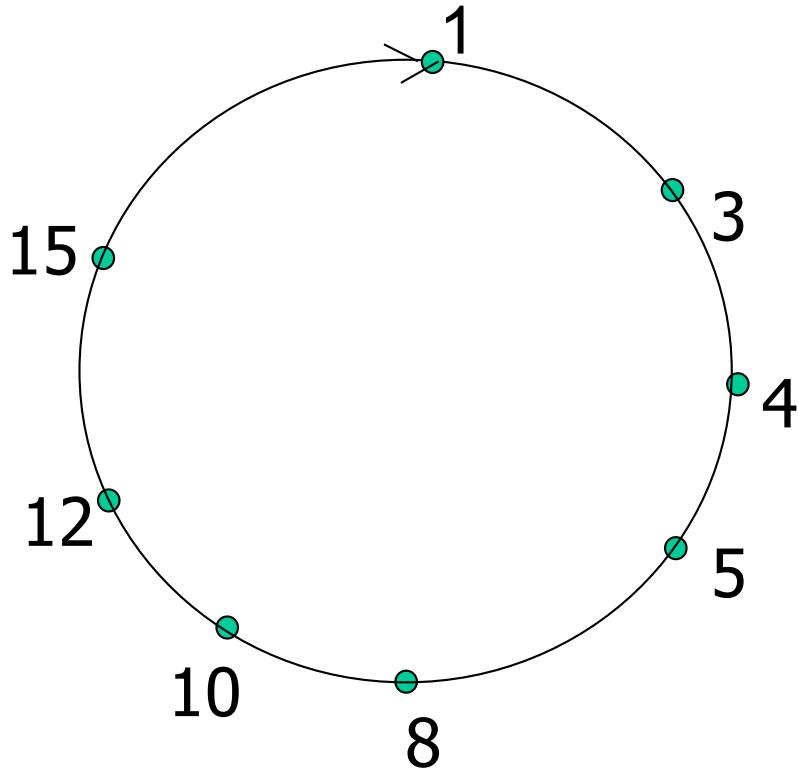
---

- ❖ 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")
  - this is why it's 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

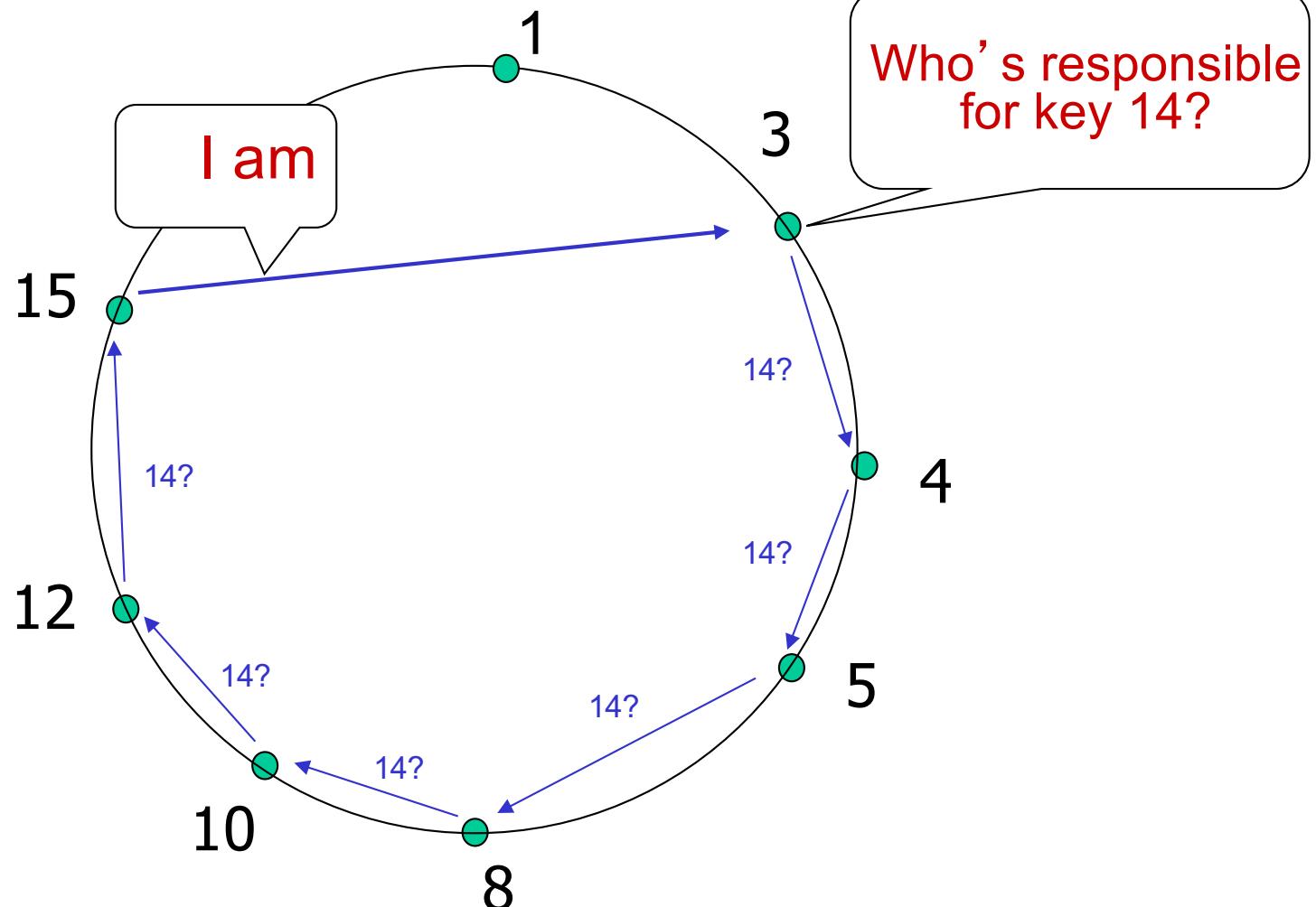
# Circular DHT (I)



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

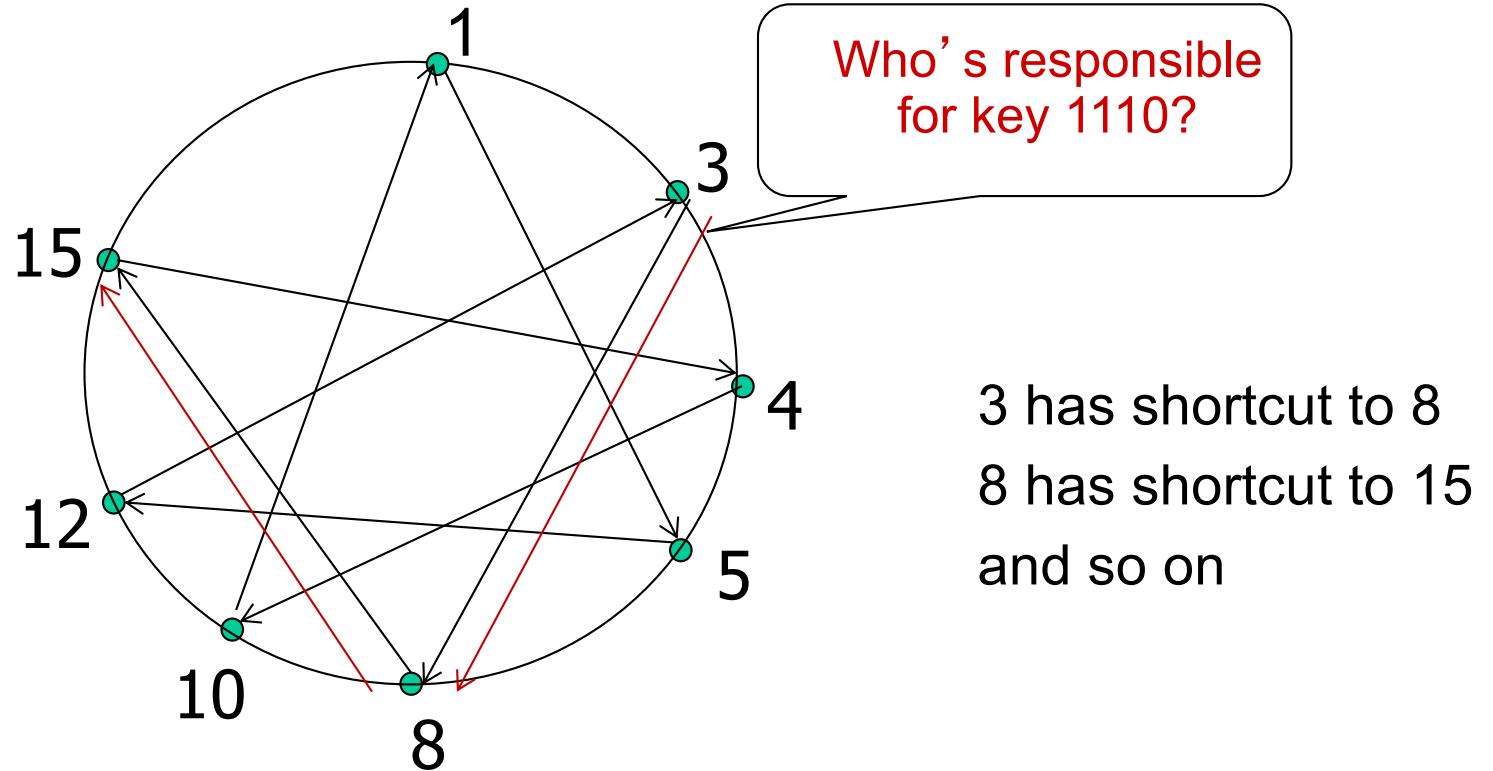
# Circular DHT (2)

Define closest as closest successor



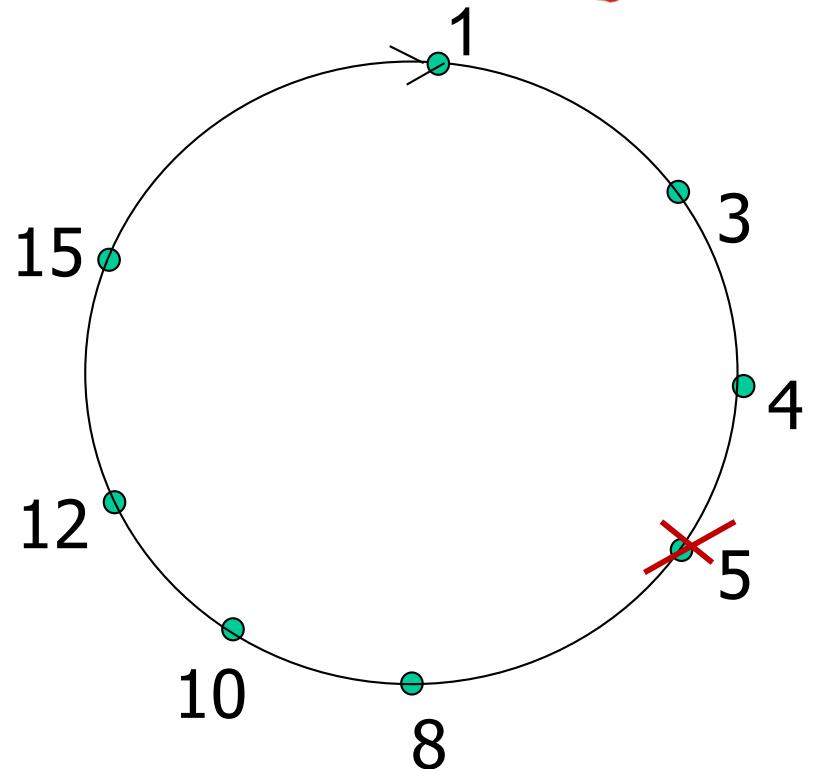
- ❖ Each peer maintains 2 neighbours
- ❖ In this example, 6 query messages are sent
- ❖ Worst case:  $N$  messages, Average:  $N/2$  messages<sup>56</sup>

# Circular DHT with shortcuts



- ❖ 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



# 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

# 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)?

# 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

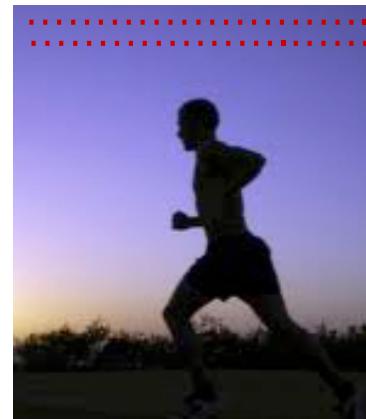
- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1.8B YouTube users, ~140M Netflix users
- challenge: scale - how to reach ~2B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different 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  $i$

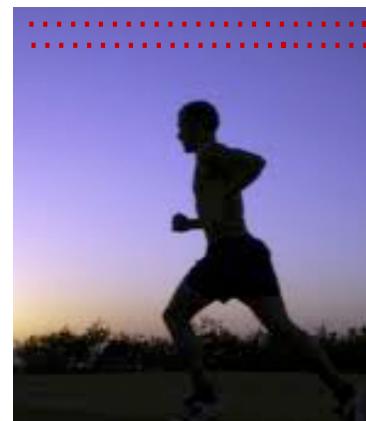


frame  $i+1$

# 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  $i$

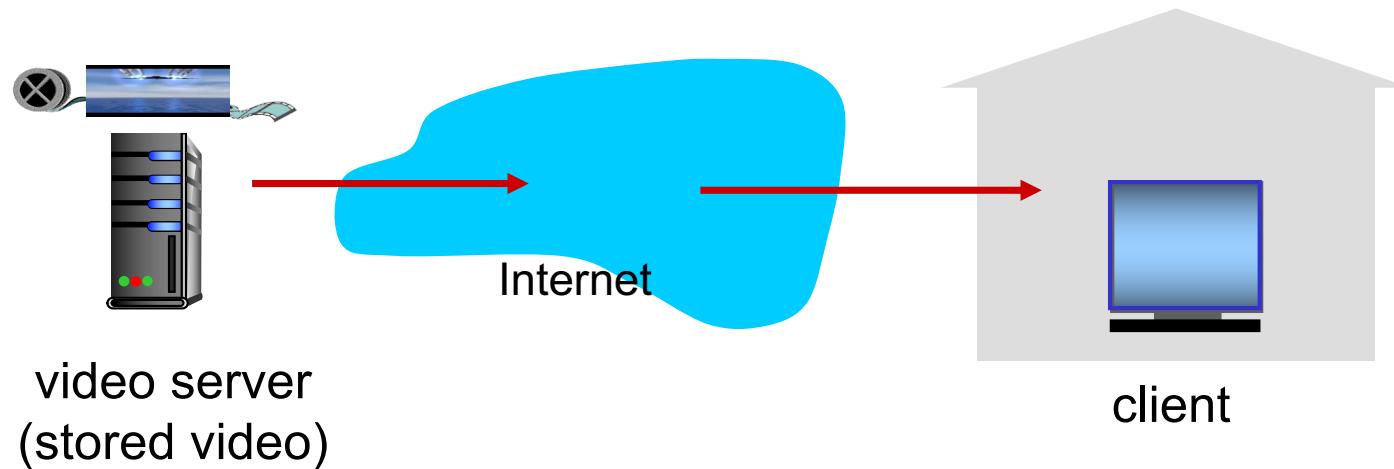
*temporal coding example:*  
instead of sending complete frame at  $i+1$ ,  
send only differences from frame  $i$



frame  $i+1$

# Streaming stored video:

simple scenario:

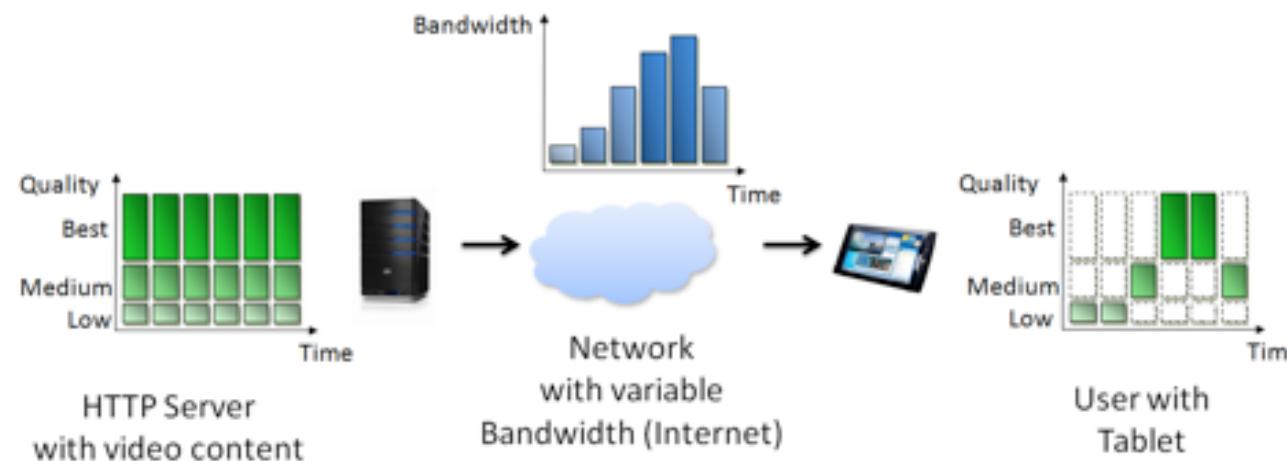


# Streaming multimedia: DASH

- ❖ *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 (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 POPs 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.

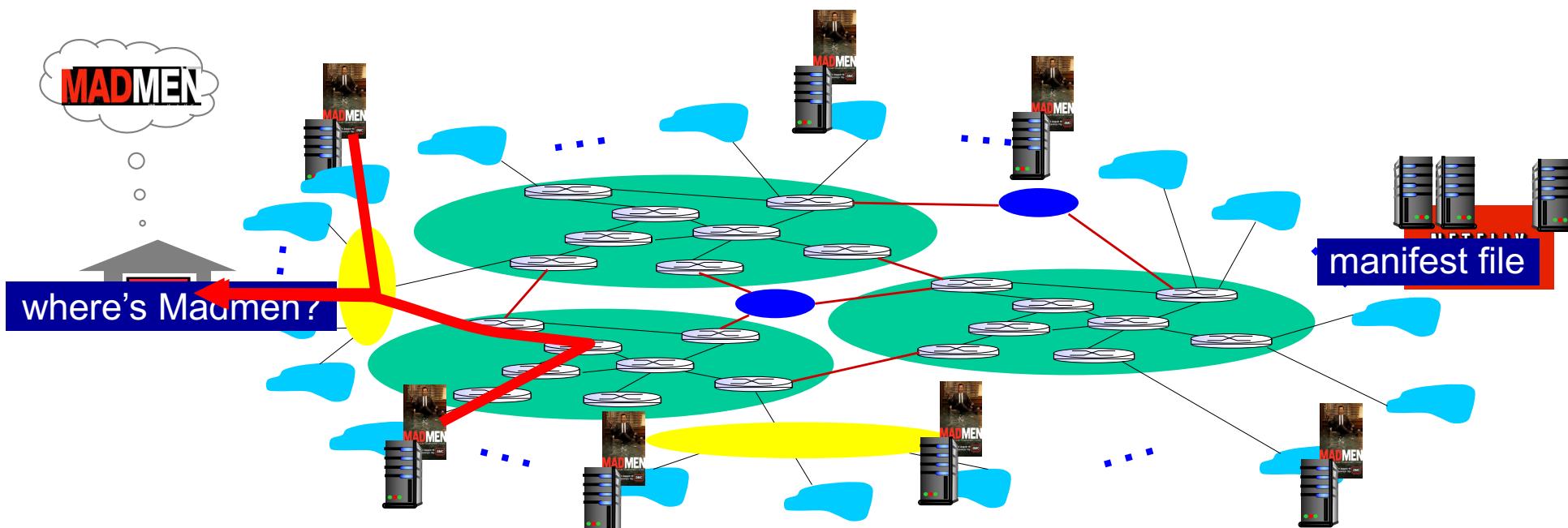
;; ADDITIONAL SECTION:
n0dscb.akamaiedge.net. 1241   IN      A       88.221.81.192
n0dscb.akamaiedge.net. 1124   IN      AAAA   2600:1480:e800::c0
n1dscb.akamaiedge.net. 842    IN      A       23.32.5.76
n2dscb.akamaiedge.net. 749    IN      A       23.32.5.84
n4dscb.akamaiedge.net. 1399   IN      A       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)

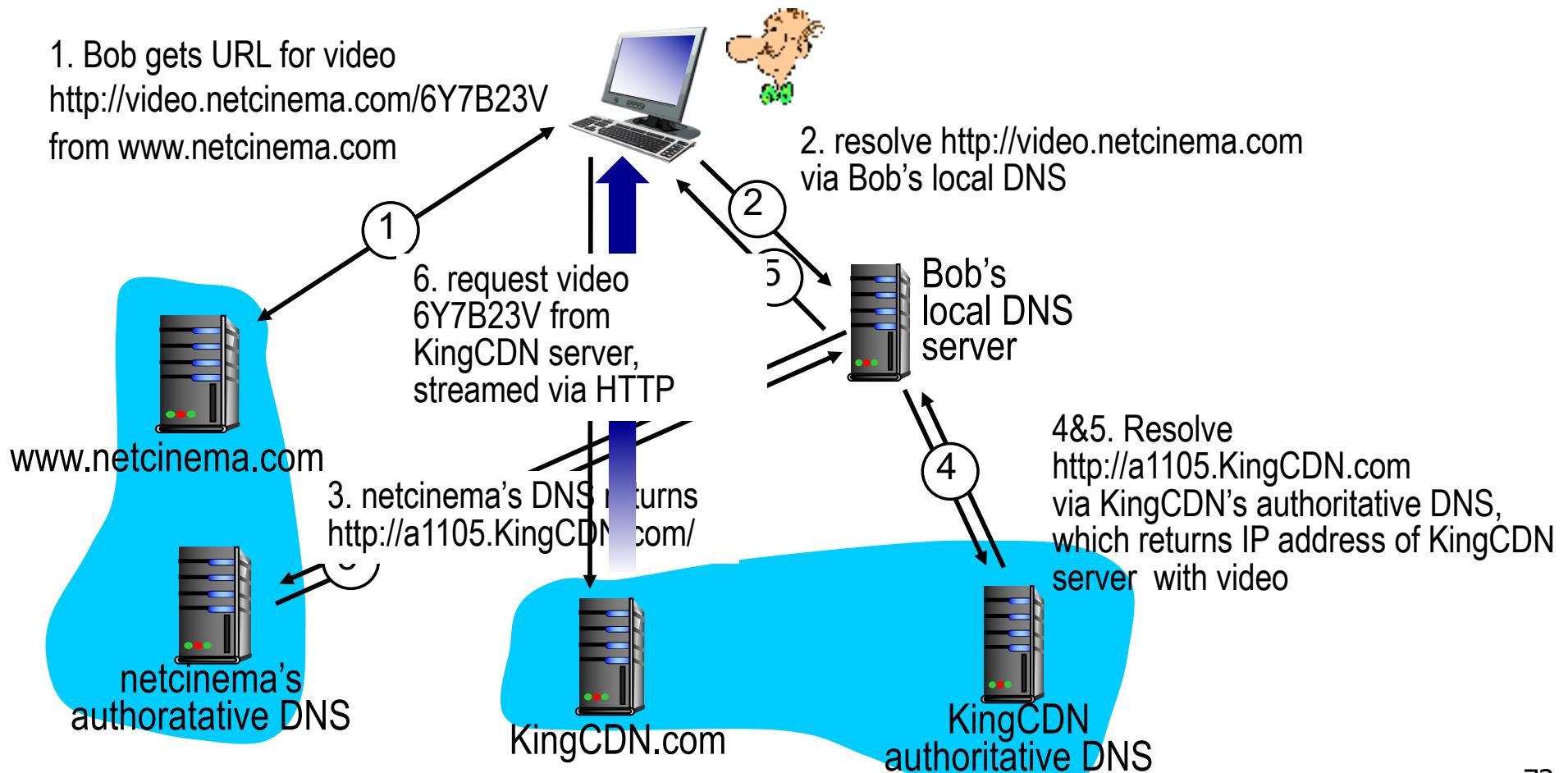
- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



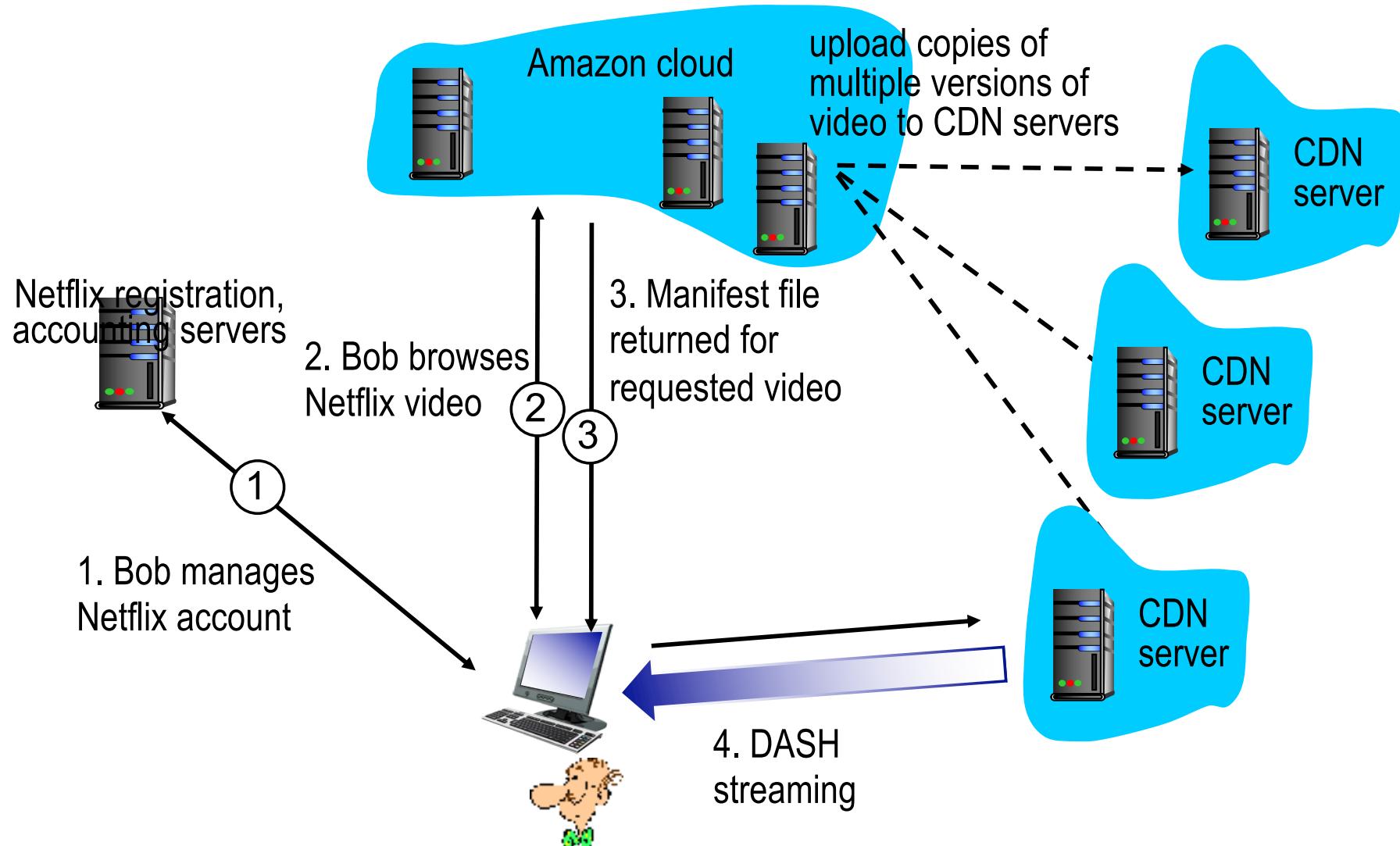
# CDN content access: a closer look

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

- video stored in CDN at managed by KingCDN.com

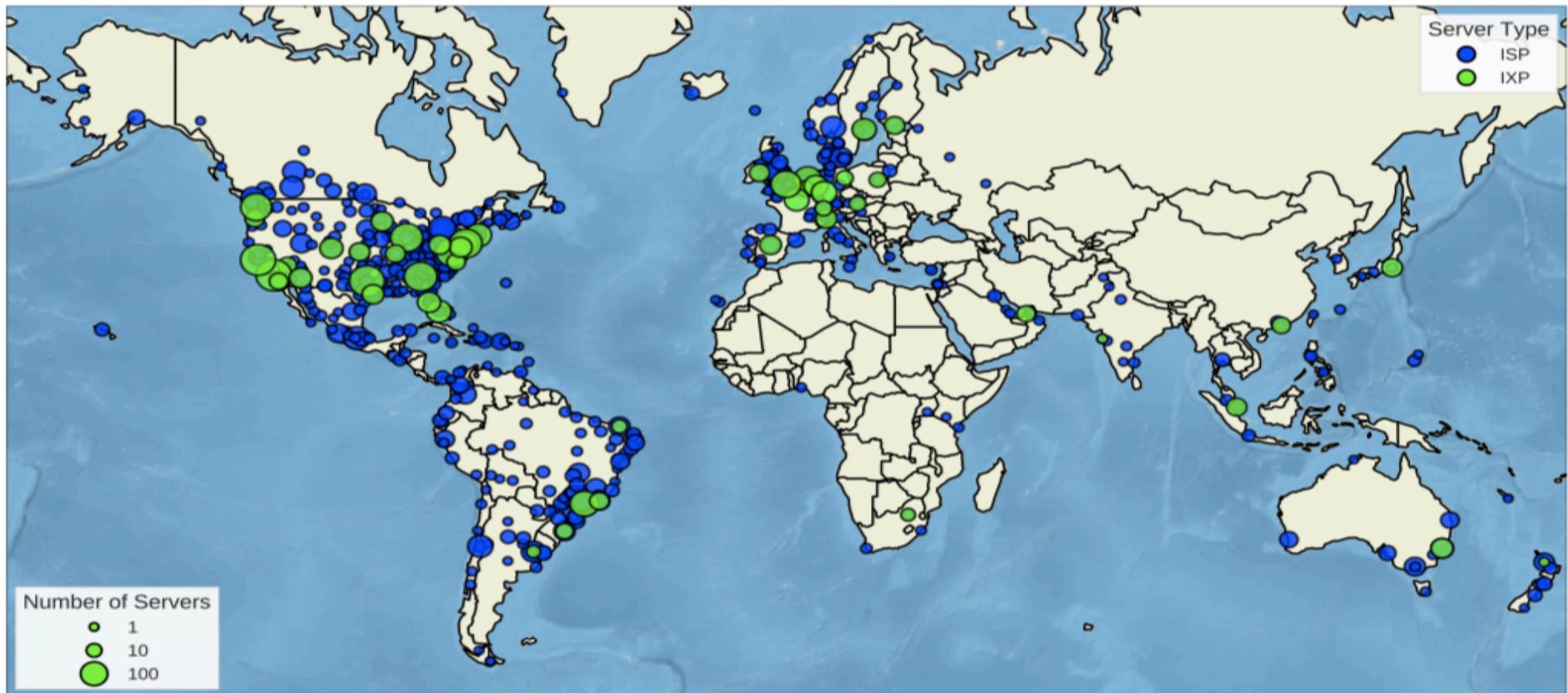


# Case study: Netflix



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)

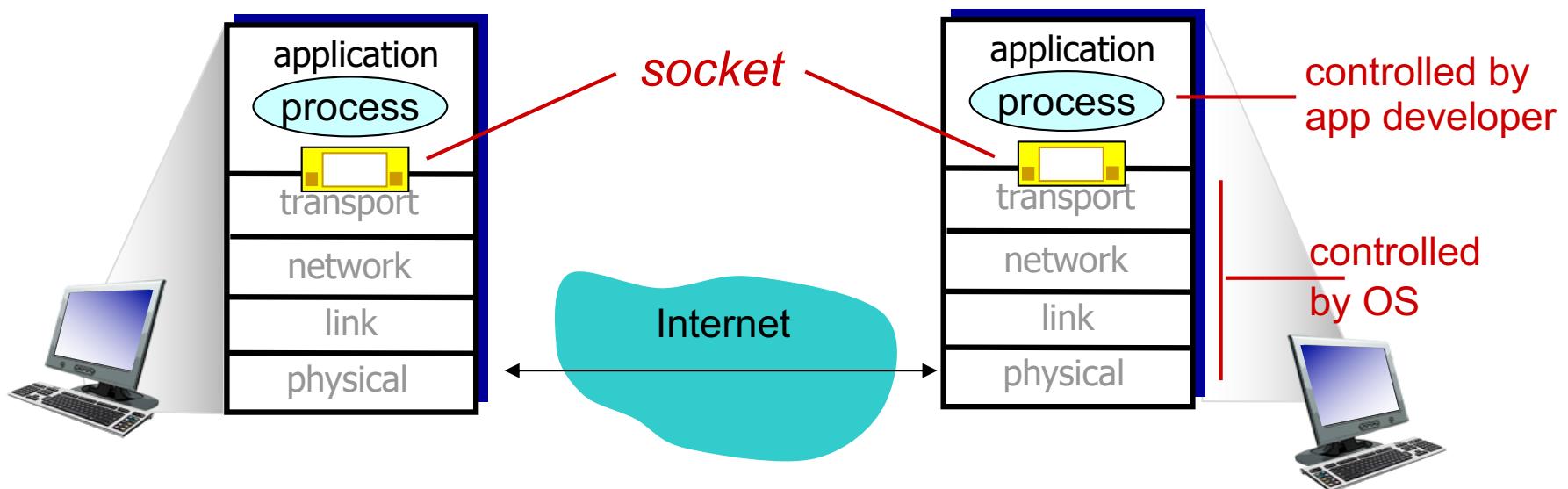
### 2.7 socket programming with UDP and TCP

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 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

# Socket programming with TCP

## client must contact server

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

## client contacts server by:

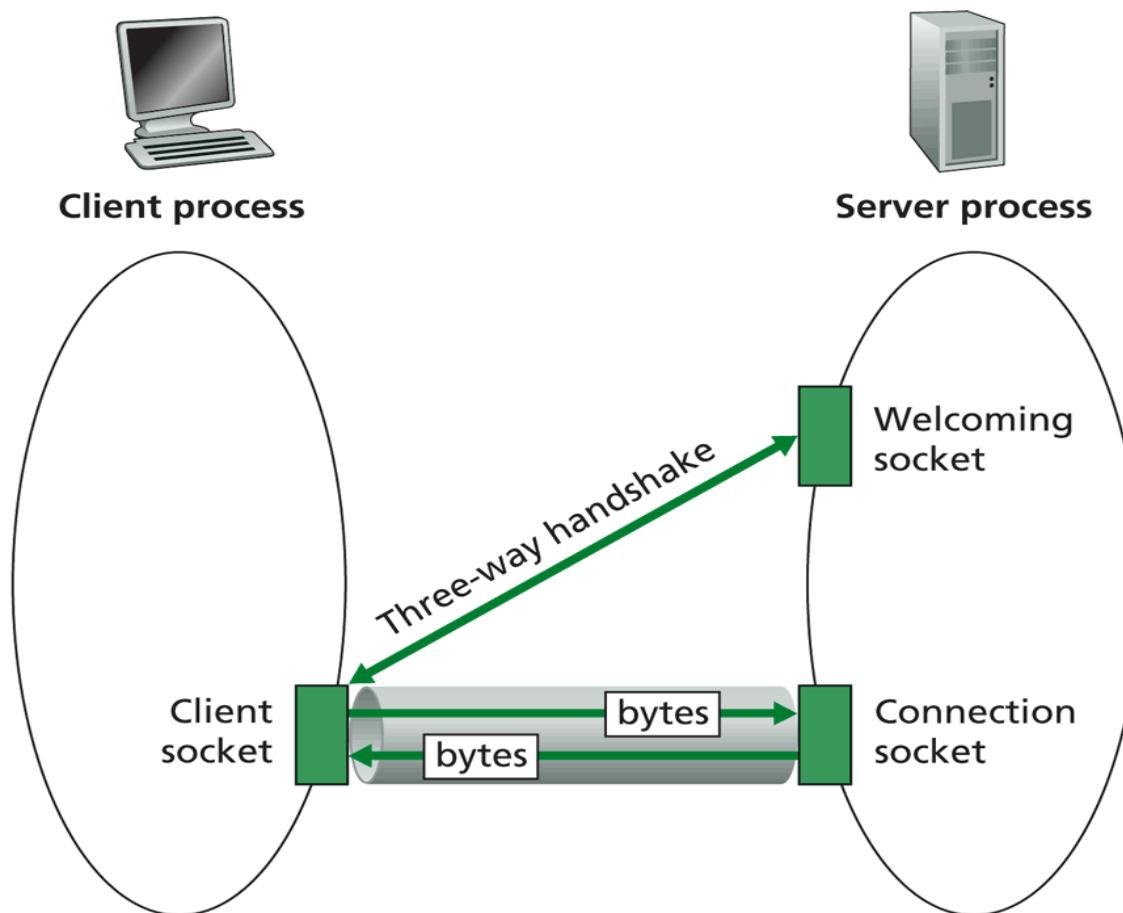
- ❖ Creating TCP socket, specifying IP address, port number of server process
- ❖ *when client creates socket:* client TCP establishes connection to server TCP

- ❖ when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more 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

# 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

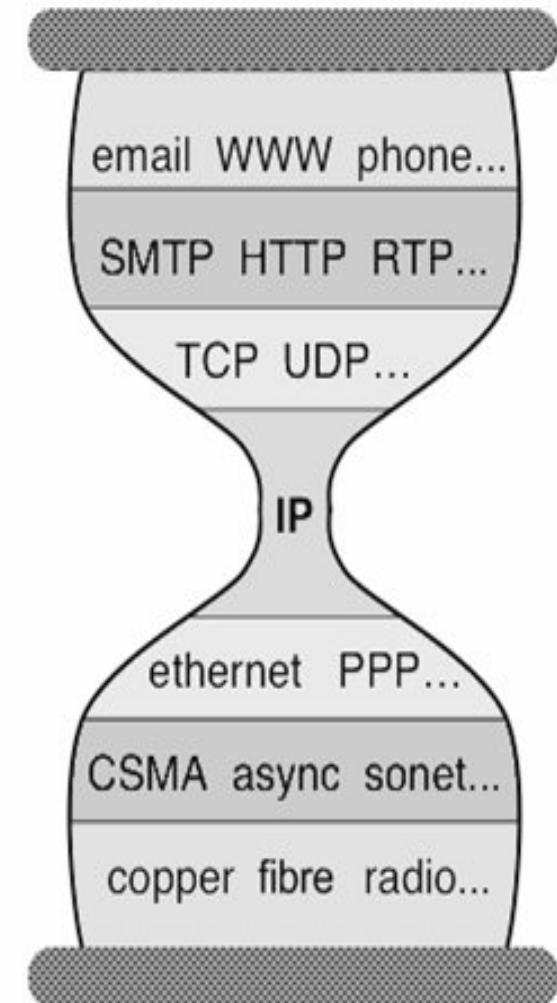
- segment structure
- reliable data transfer
- flow control
- connection management

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 !!

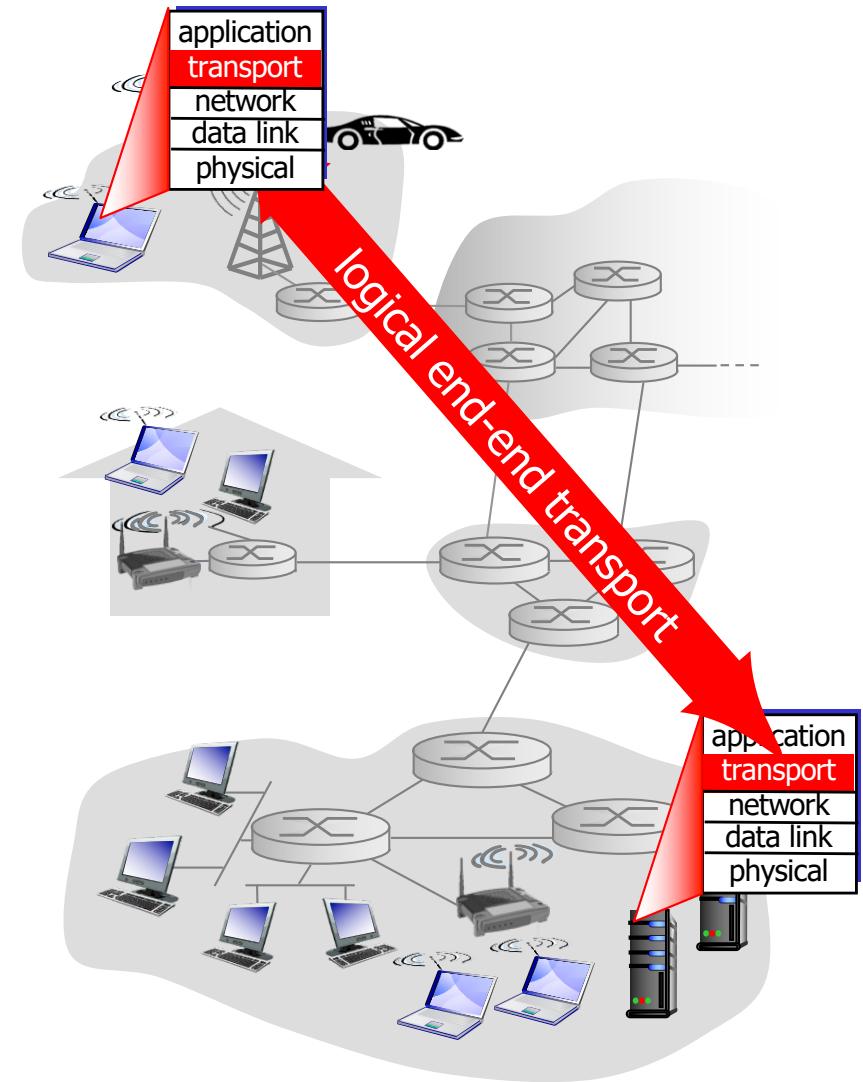


# Network layer (some context)

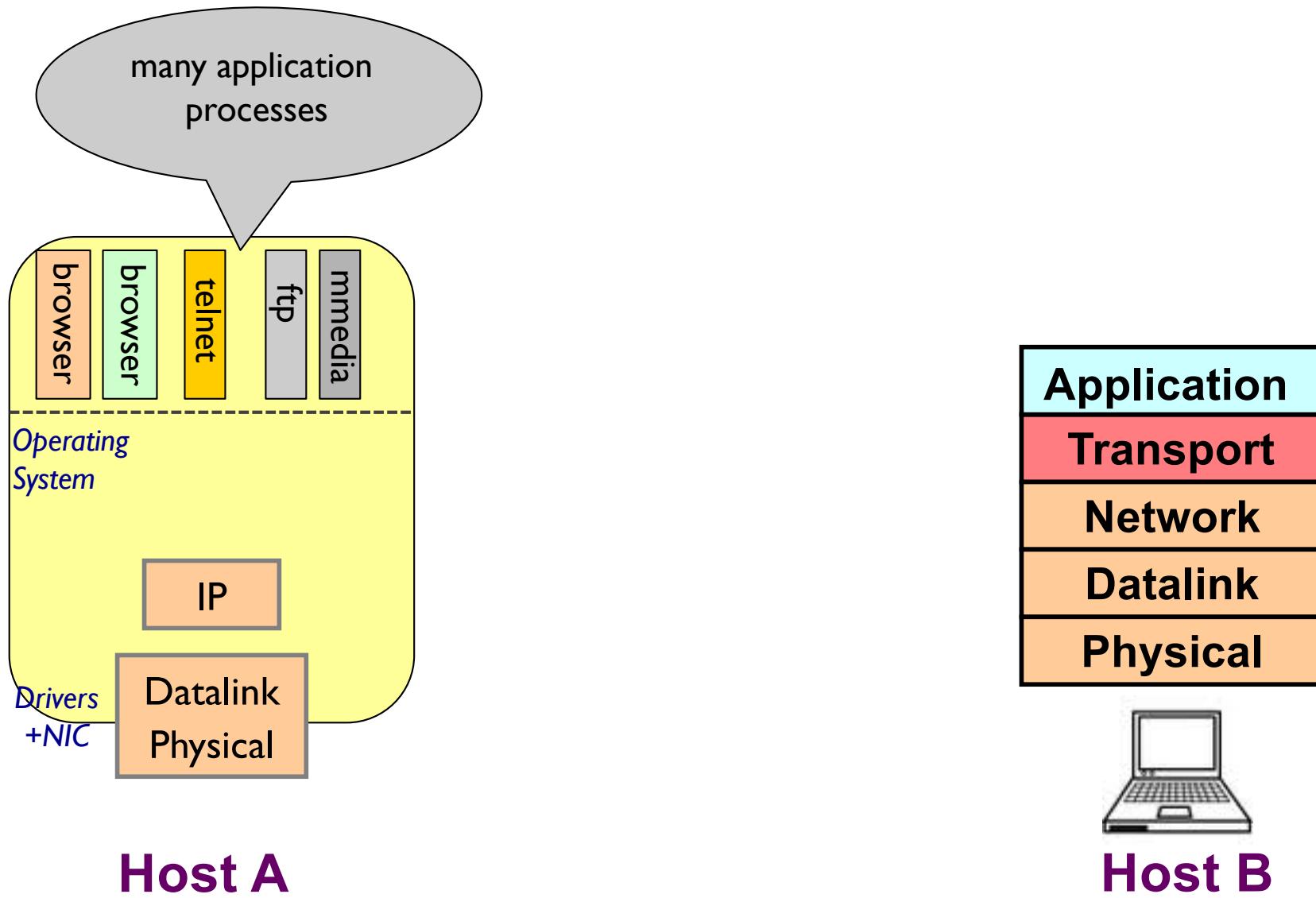
- ❖ What it does: finds paths through network
  - Routing from one end host to another
- ❖ What it doesn't:
  - Reliable transfer: “best effort delivery”
  - Guarantee paths
  - Arbitrate transfer rates
- ❖ For now, think of the network layer as giving us an “API” with one function:  
*sendtohost(data,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

# Transport services and protocols

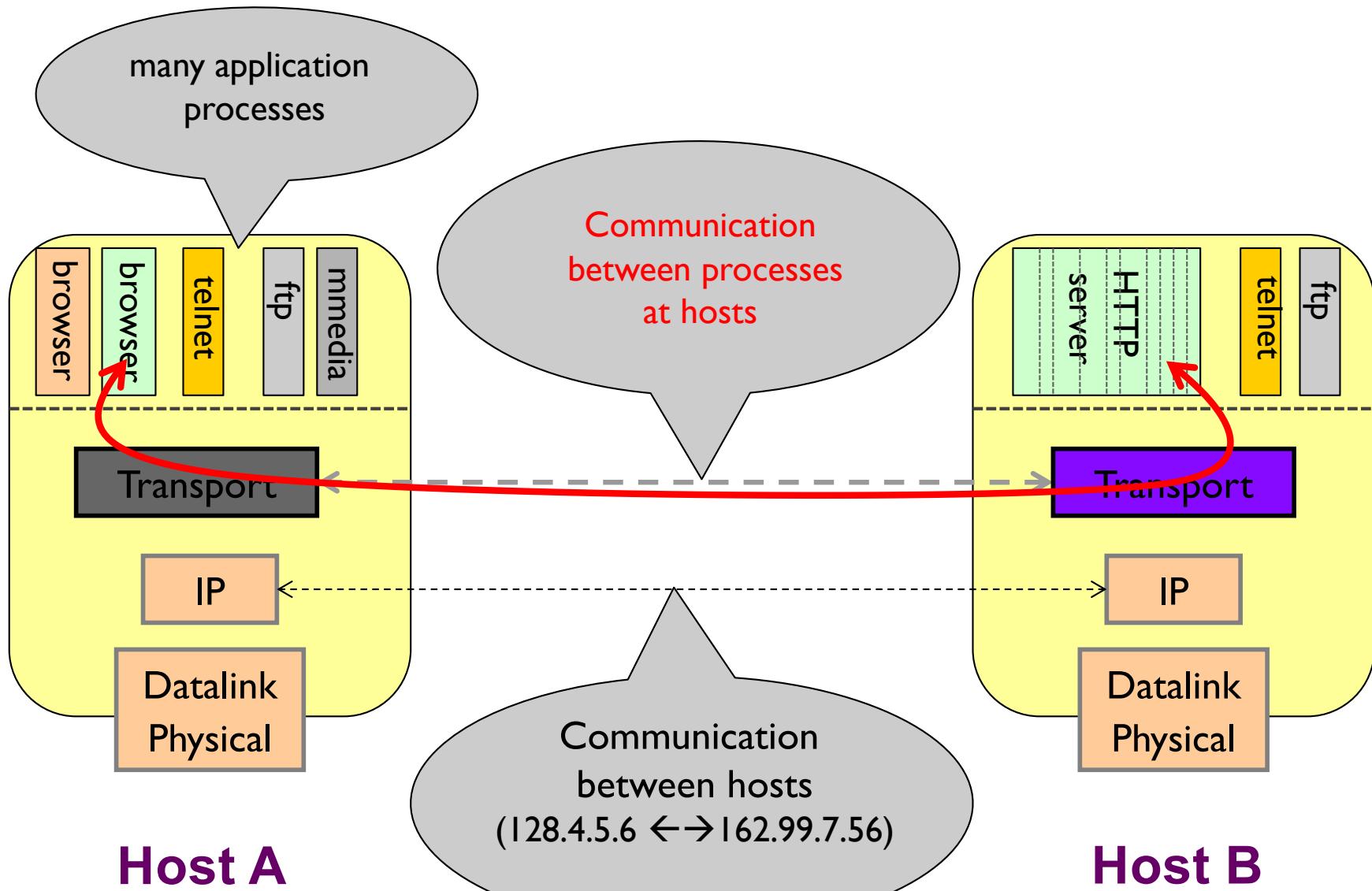
- ❖ provide *logical communication* between app processes running on different hosts
- ❖ transport protocols run in end systems
  - sender side: breaks app messages into *segments*, passes to network layer
  - receiver side: reassembles segments into messages, passes to app layer
  - Exports services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

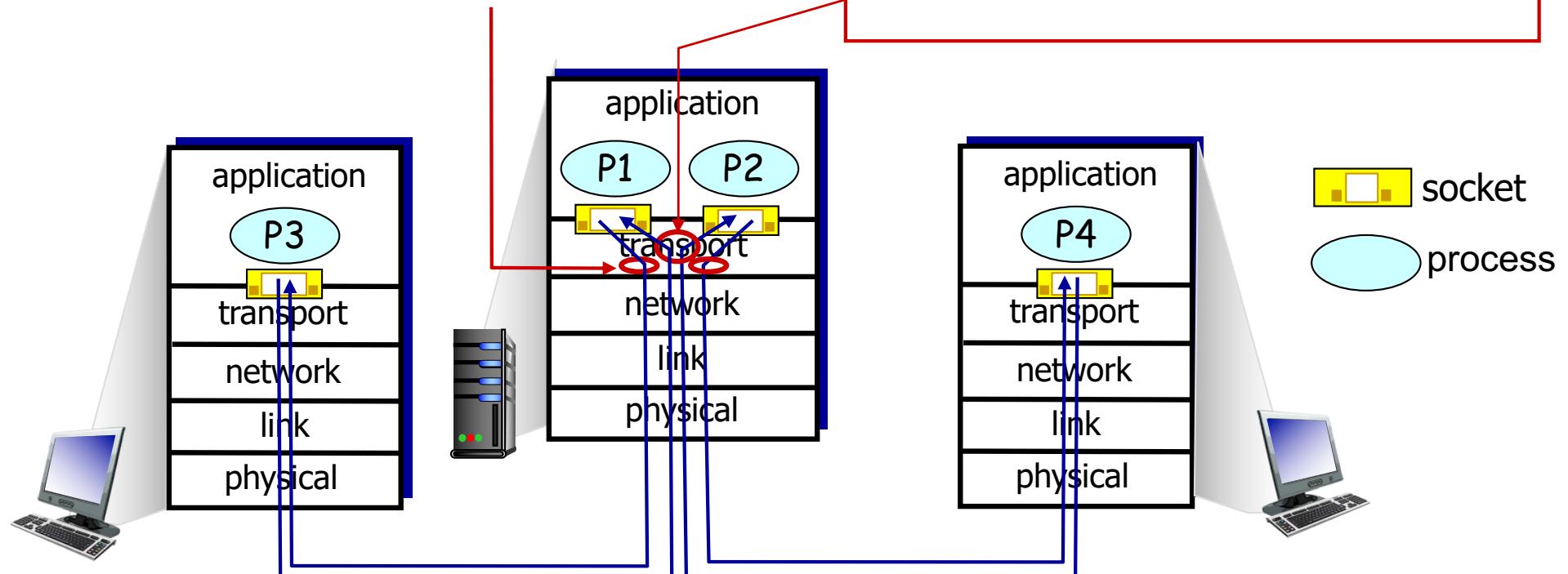
# Multiplexing/demultiplexing

*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:

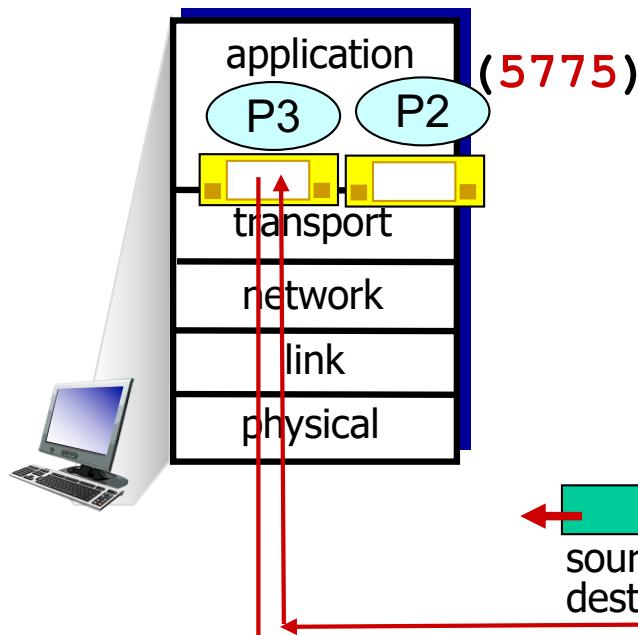
- checks destination port # in segment
- directs UDP segment to socket with that port #



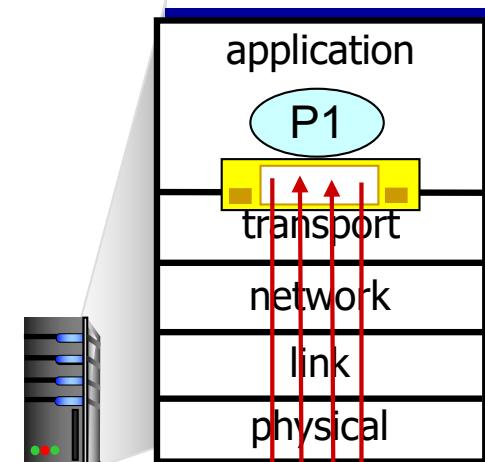
IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

# Connectionless demux: example

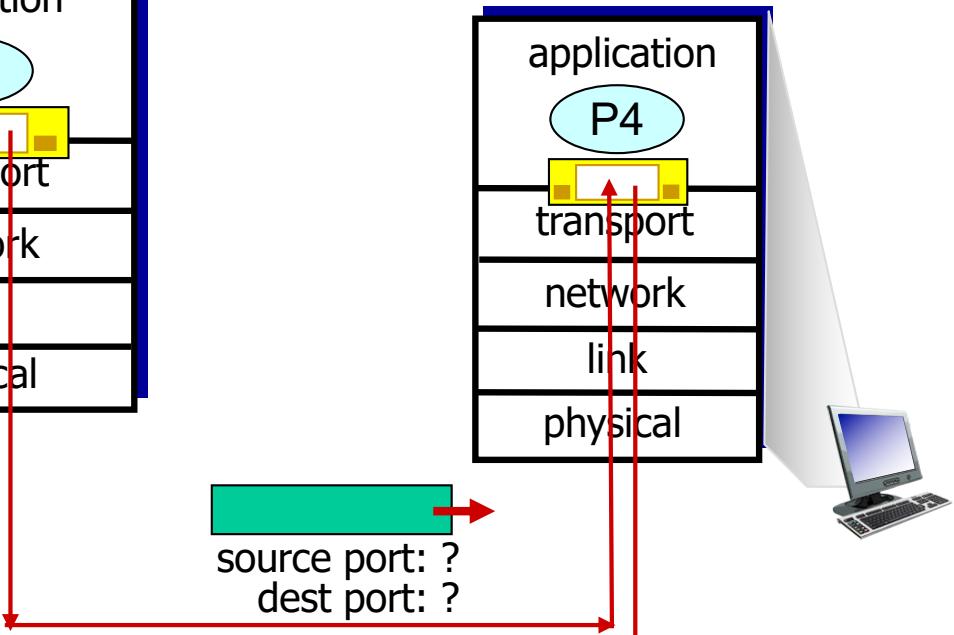
```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```



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



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



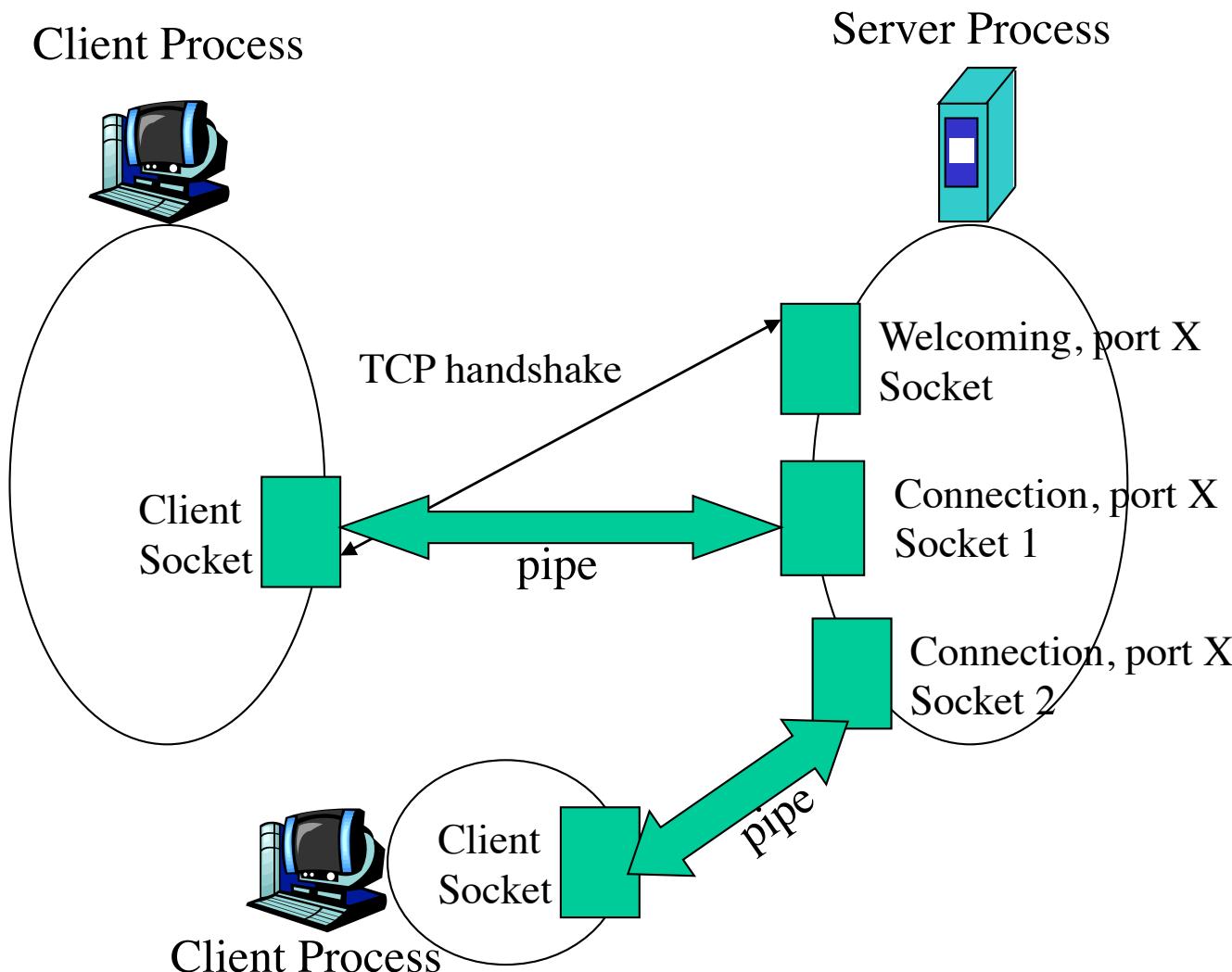
source port: 9157  
dest port: 6428

source port: ?  
dest port: ?

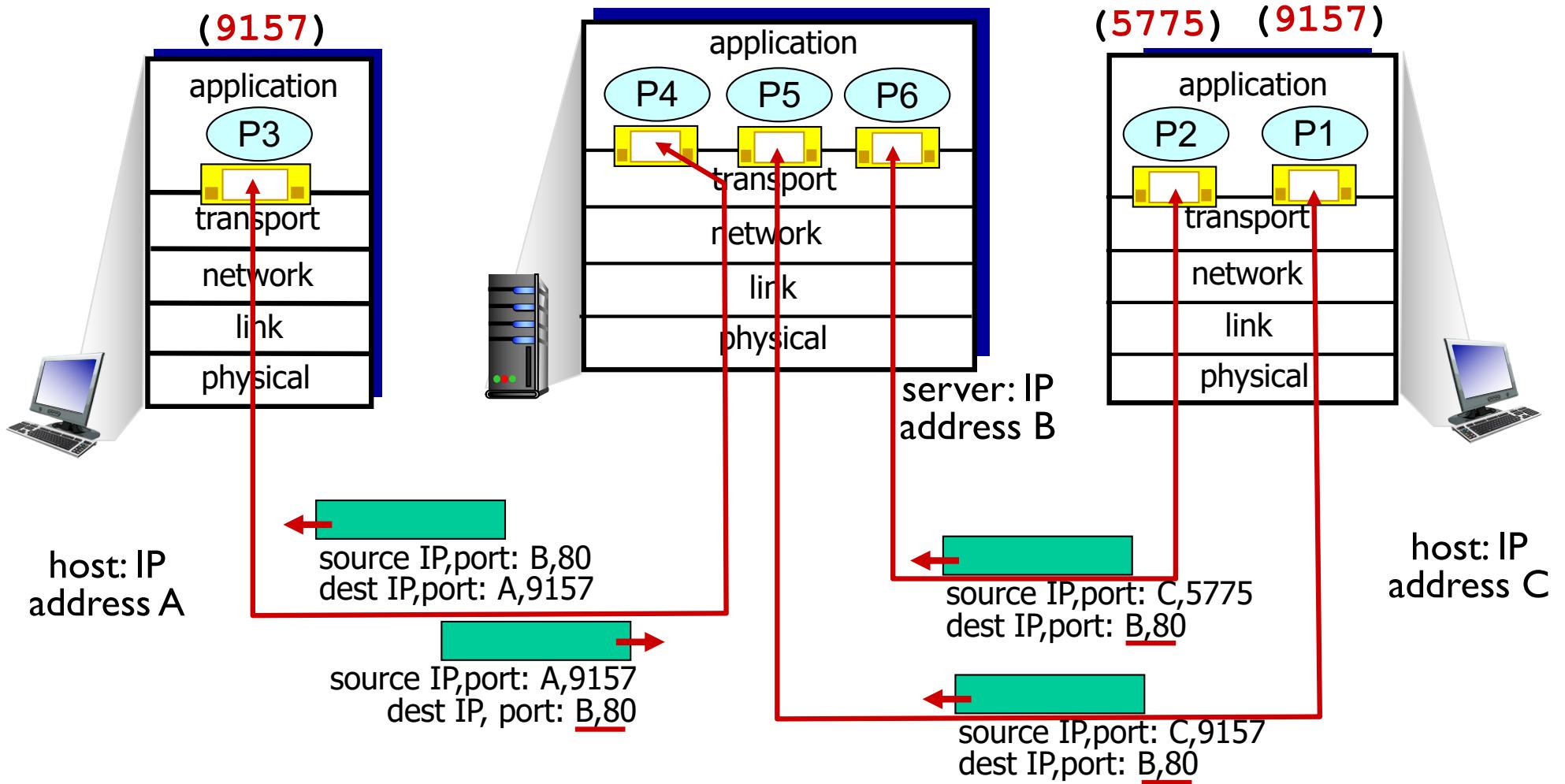
# Connection-oriented demux

- ❖ 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

# 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

# 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

# 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

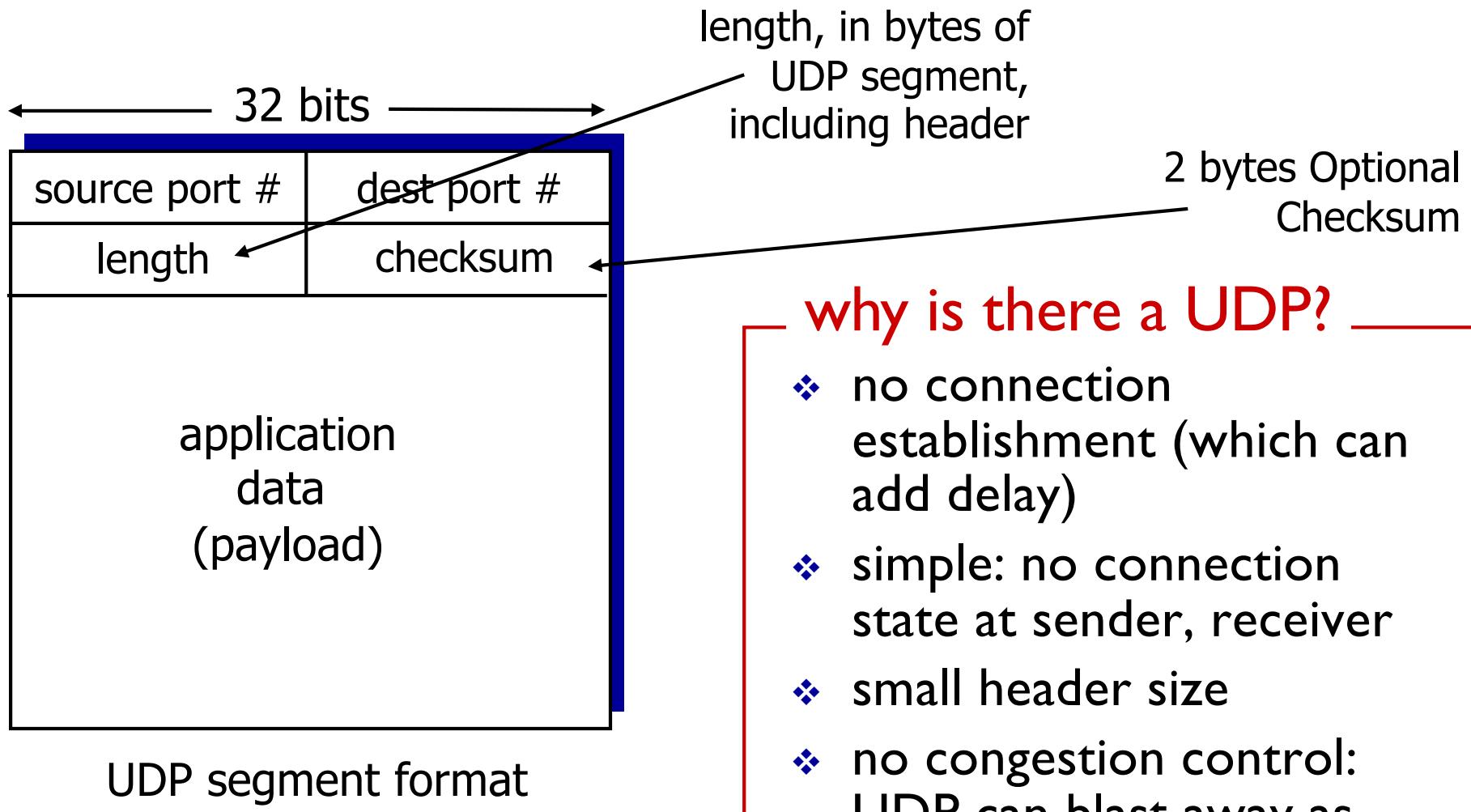
3.6 principles of congestion control

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

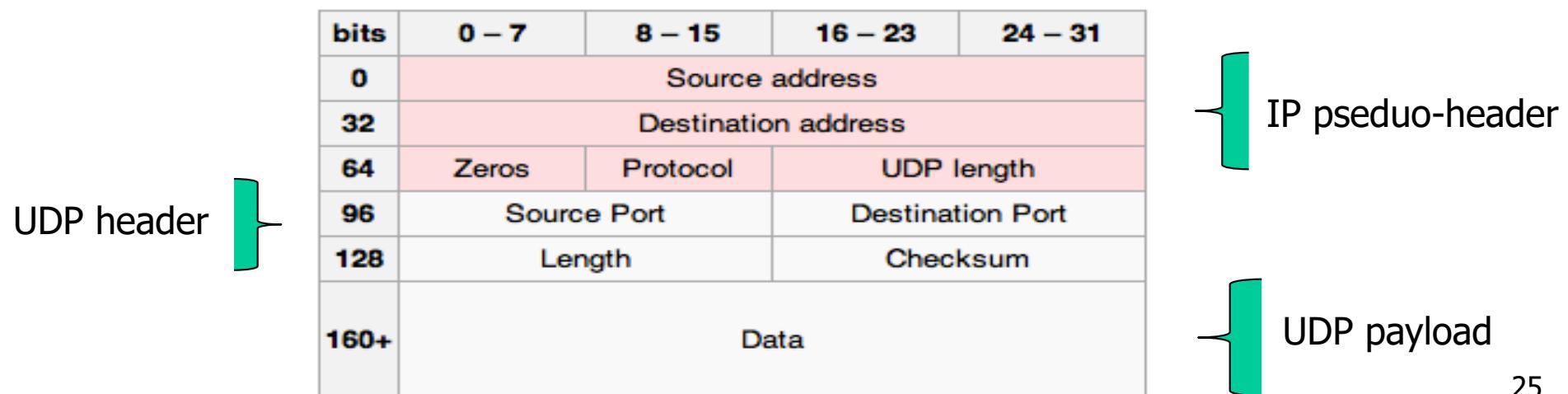
---

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

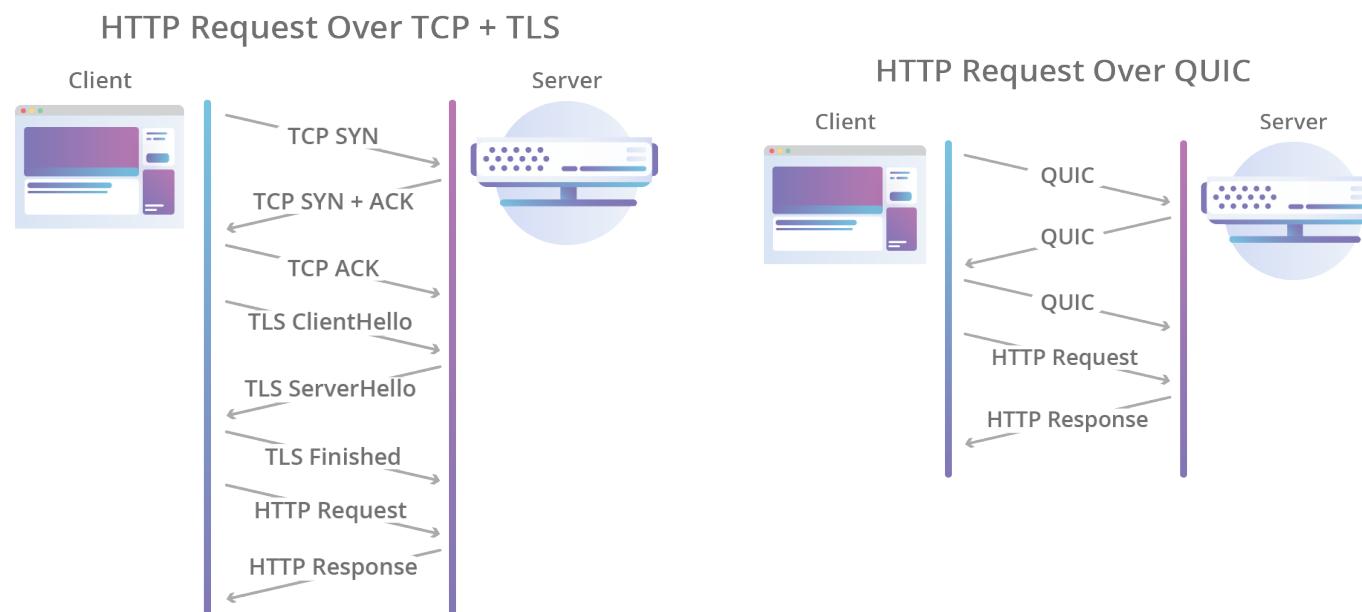
Note: TCP Checksum computation is exactly similar

# 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 unnecessary (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

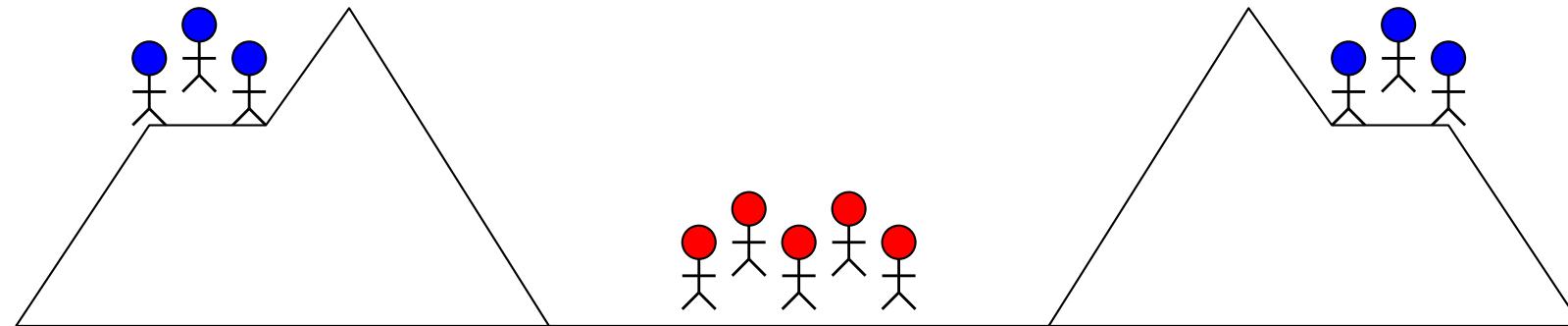
3.6 principles of congestion control

3.7 TCP congestion control

# Reliable Transport

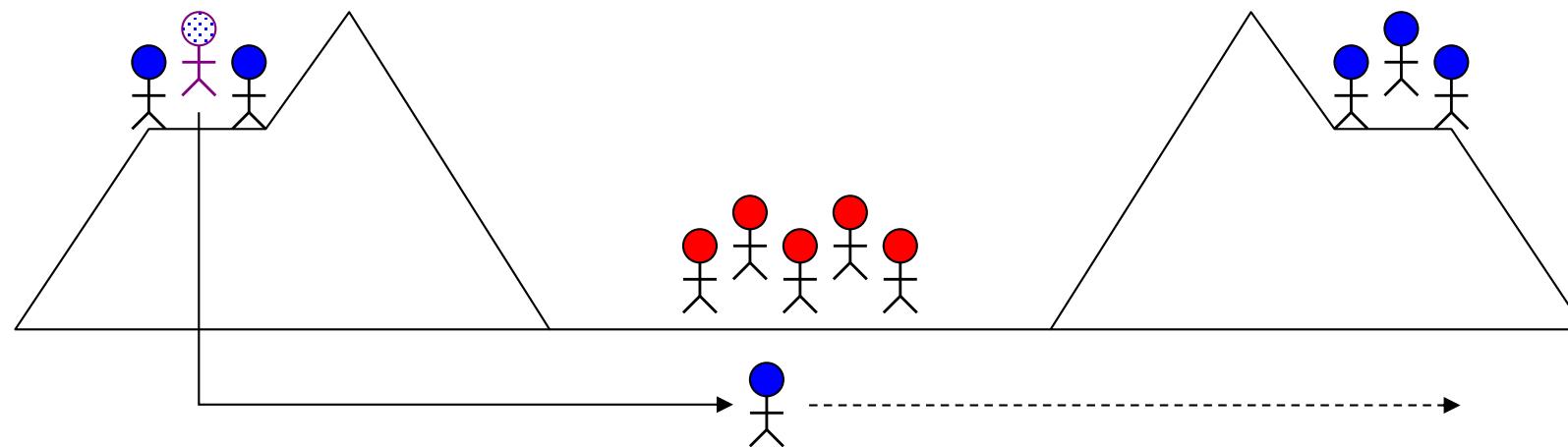
- In a perfect world, reliable transport is easy
- All the bad things best-effort can do
  - a packet is corrupted (bit errors)
  - a packet is lost (*why?*)
  - a packet is delayed (*why?*)
  - packets are reordered (*why?*)
  - a packet is duplicated (*why?*)

# The Two Generals Problem



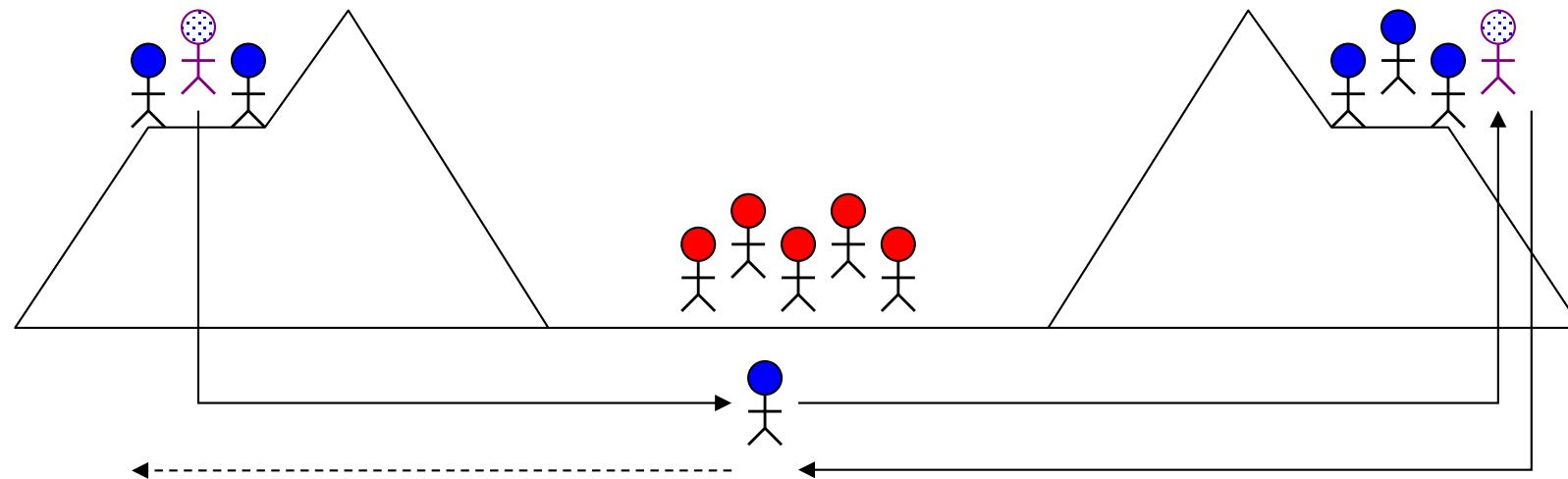
- ❖ Two army divisions (blue) surround enemy (red)
  - Each division led by a general
  - Both must agree when to simultaneously attack
  - If either side attacks alone, defeat
- ❖ Generals can only communicate via messengers
  - Messengers may get captured (unreliable channel)

# The Two Generals Problem



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

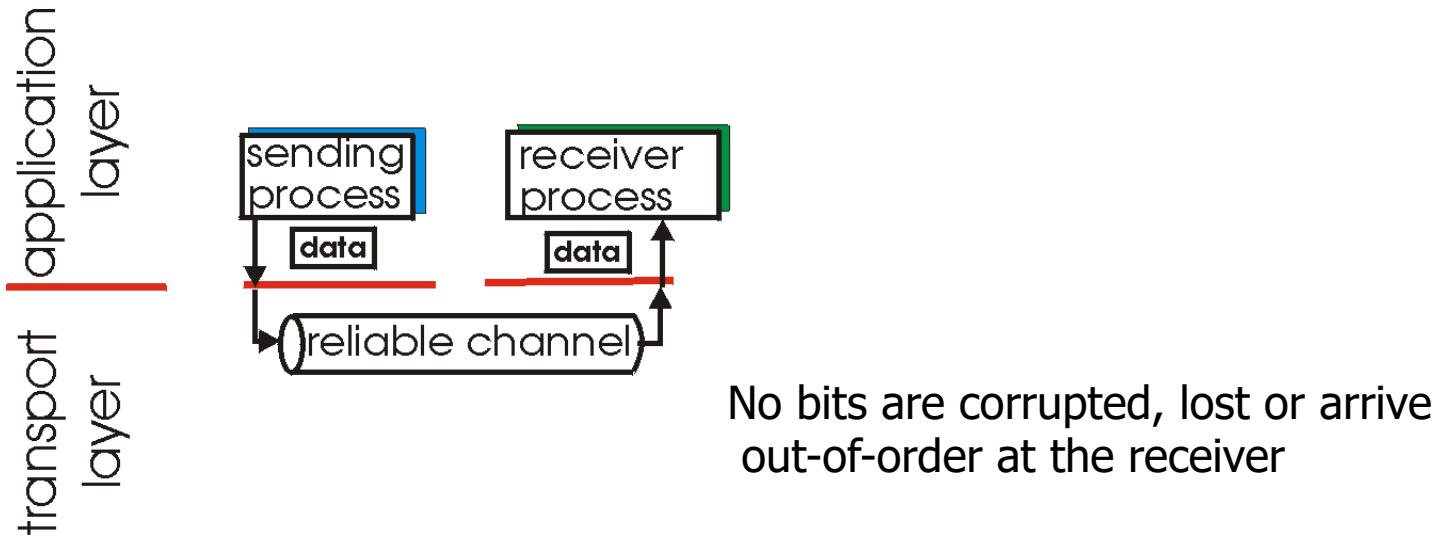
# The Two Generals Problem



- ❖ 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!

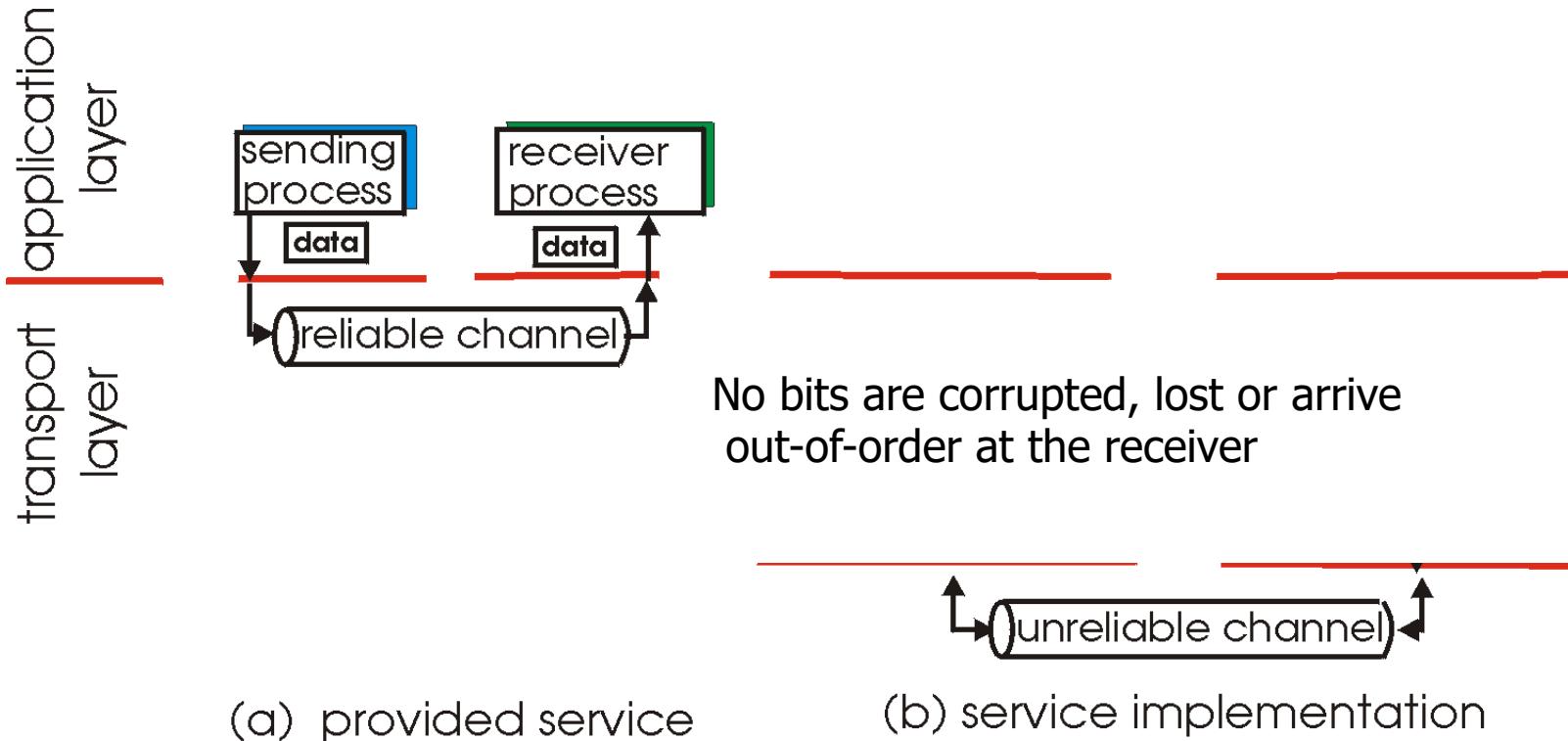


(a) provided service

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

# Principles of reliable data transfer

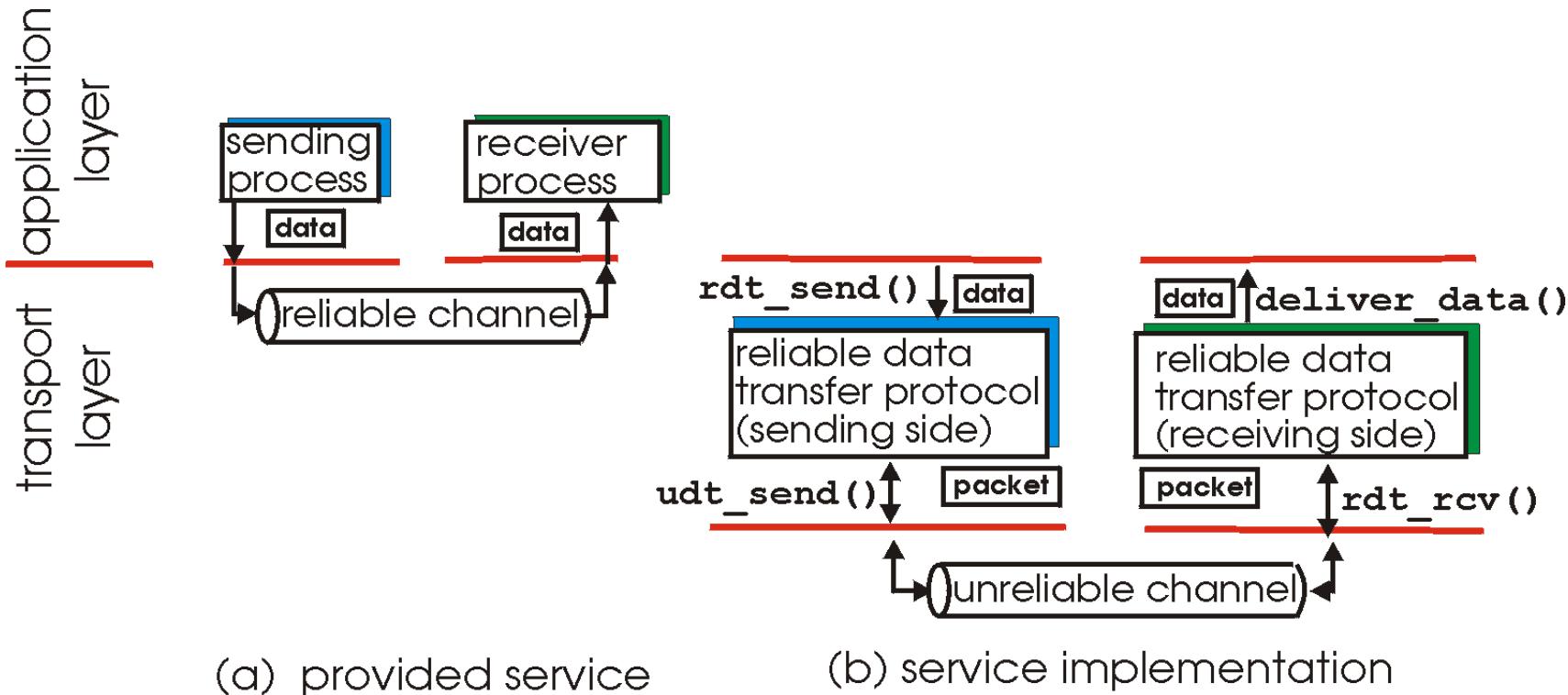
- ❖ important in application, transport, link layers
  - top-10 list of important networking topics!



- ❖ 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:

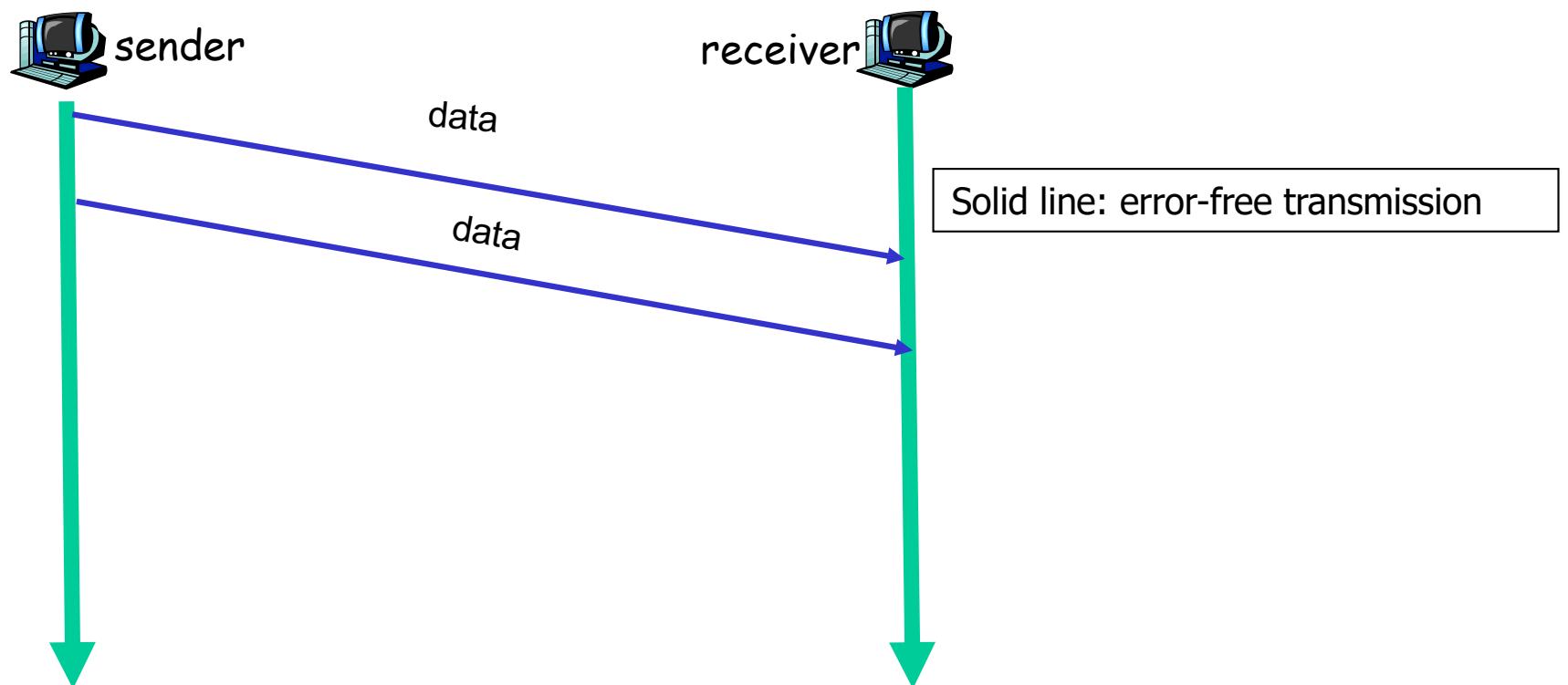
- Incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- Consider only unidirectional data transfer
  - but control info will flow on both directions!
- Channel will not re-order packets

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

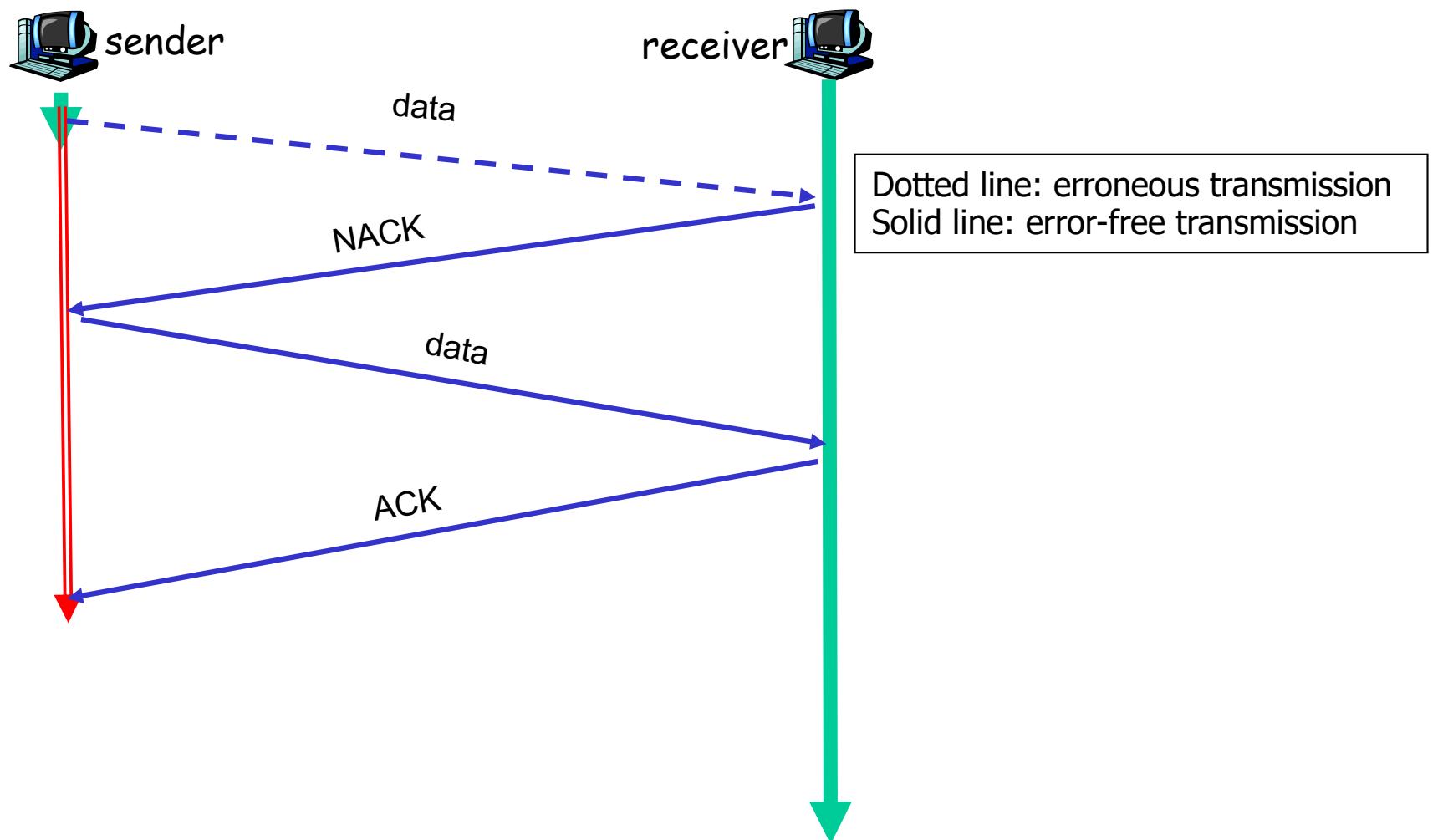
- ❖ underlying channel may flip bits in packet
  - checksum to detect bit errors
- ❖ *the question: how to recover from errors:*

*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 many states
  - state must “remember” whether “expected” pkt should have seq # of 0 or 1

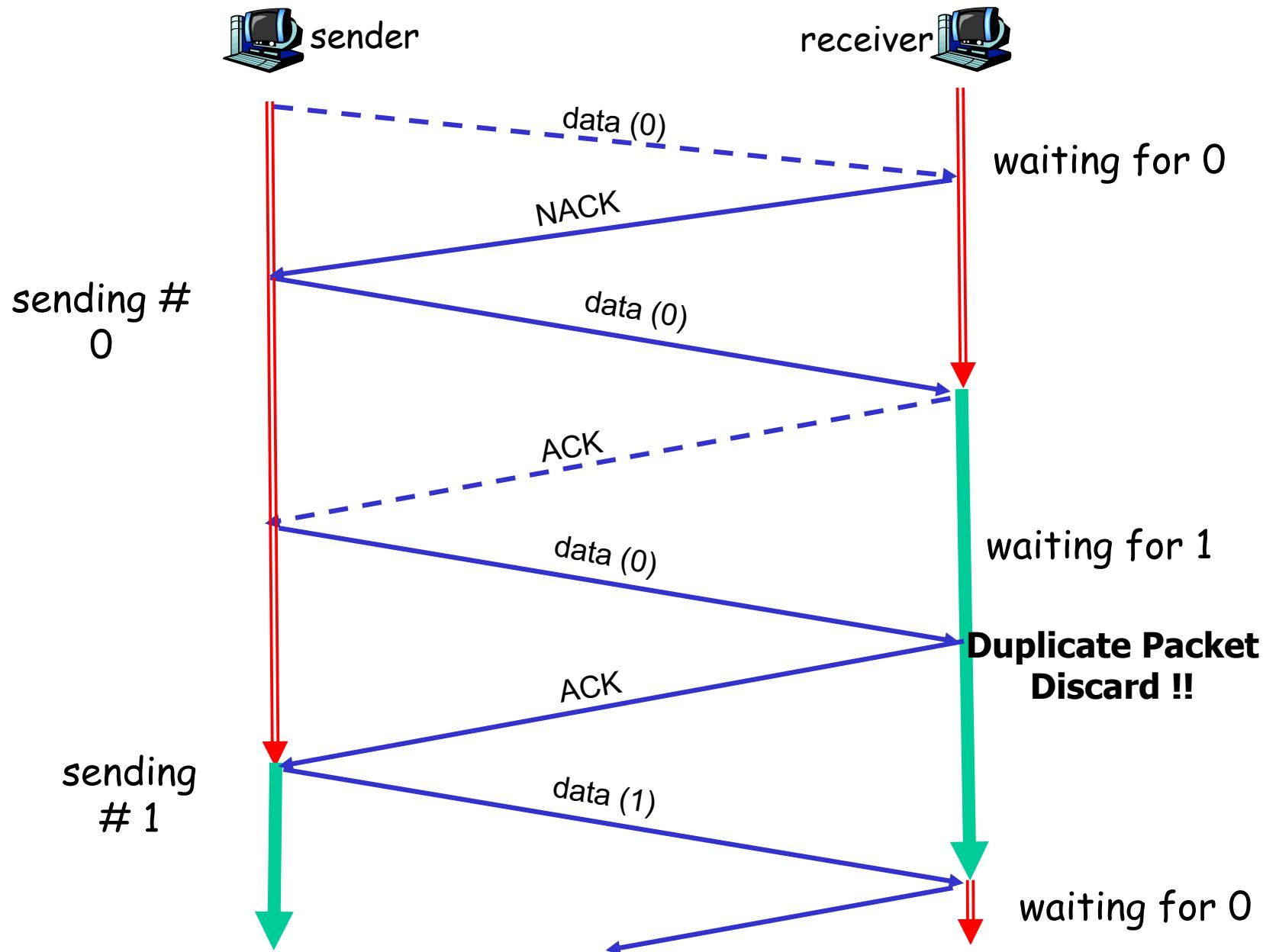
## receiver:

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

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

# Another Look at rdt2.1

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

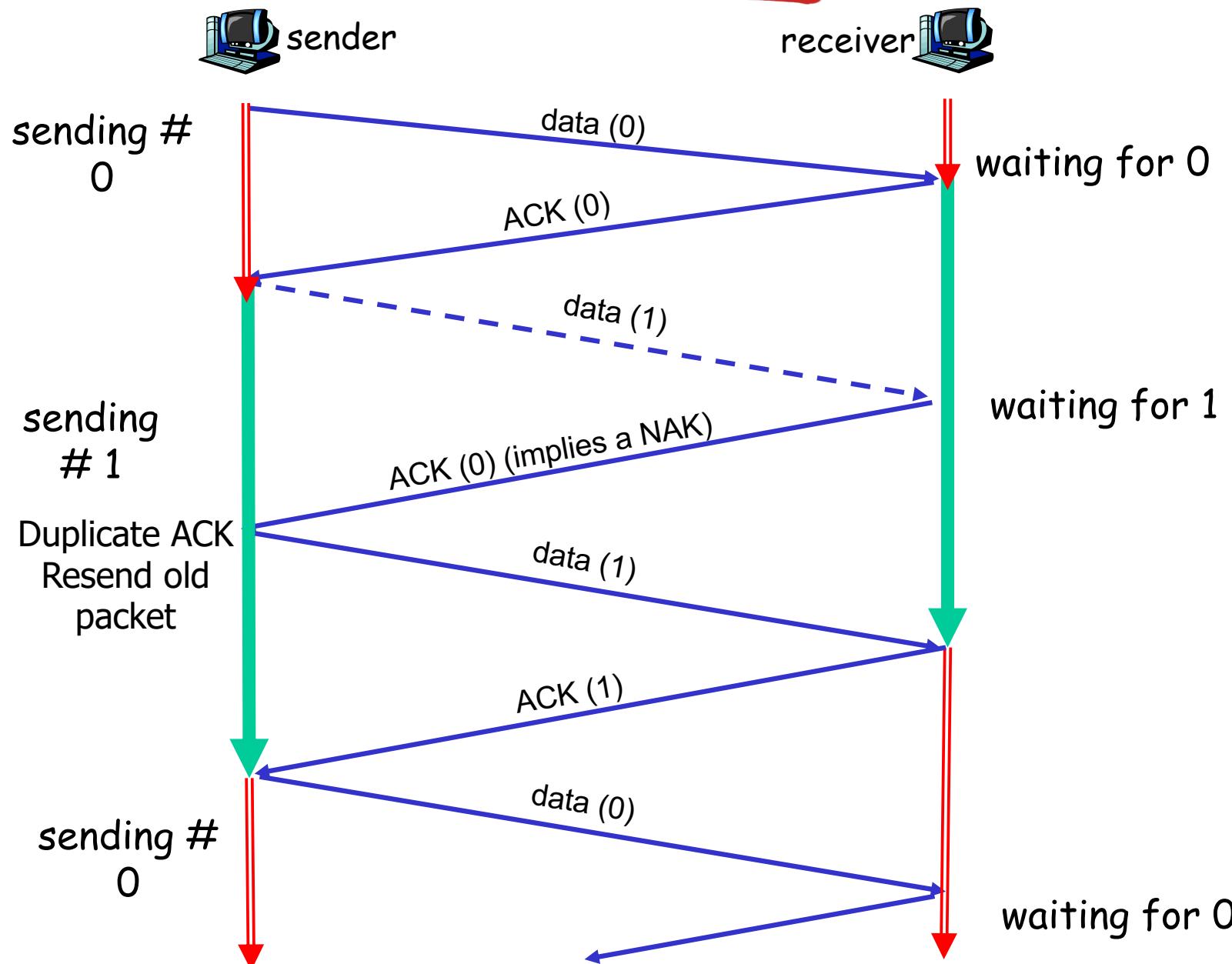


## rdt2.2: a NAK-free protocol

- ❖ 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)

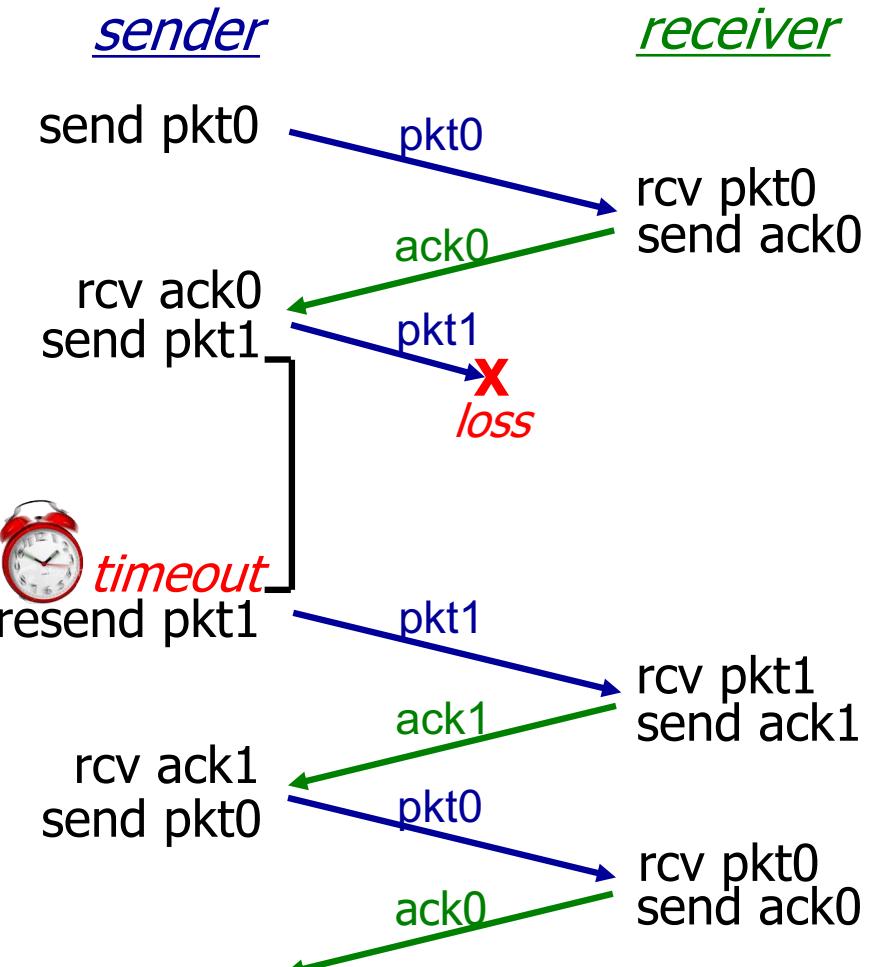
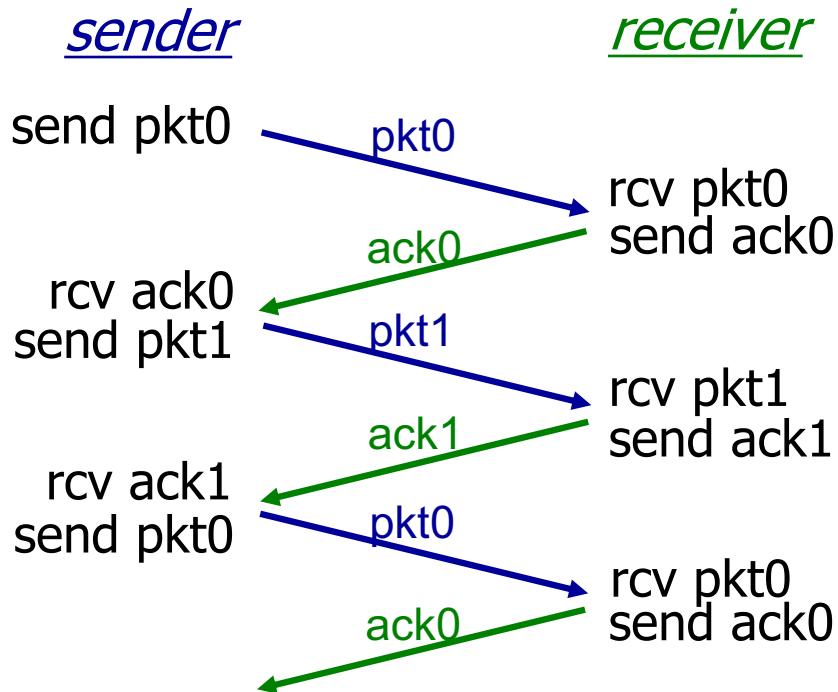
- checksum, seq. #, ACKs, retransmissions will be of help ... but not enough

## approach: sender waits

“reasonable” amount of time for ACK

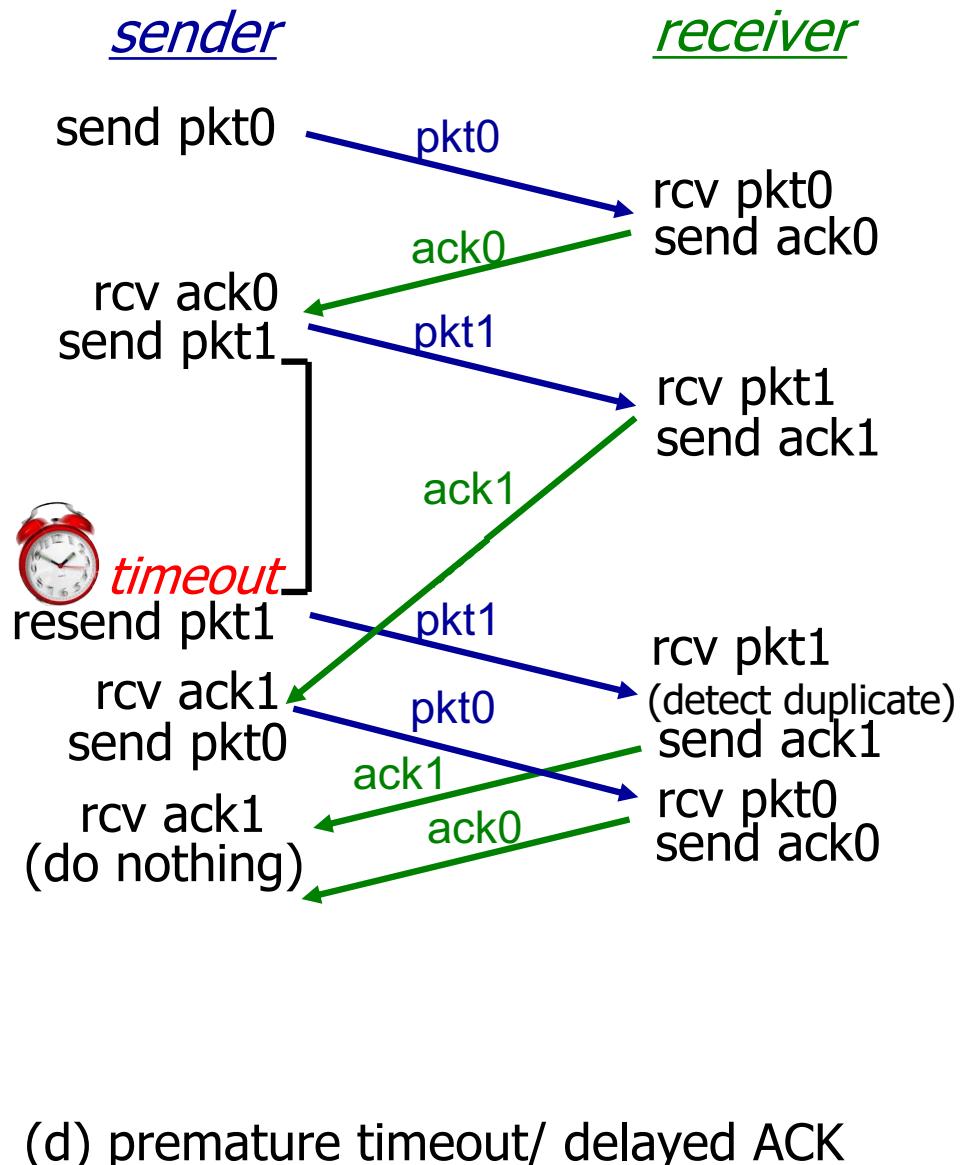
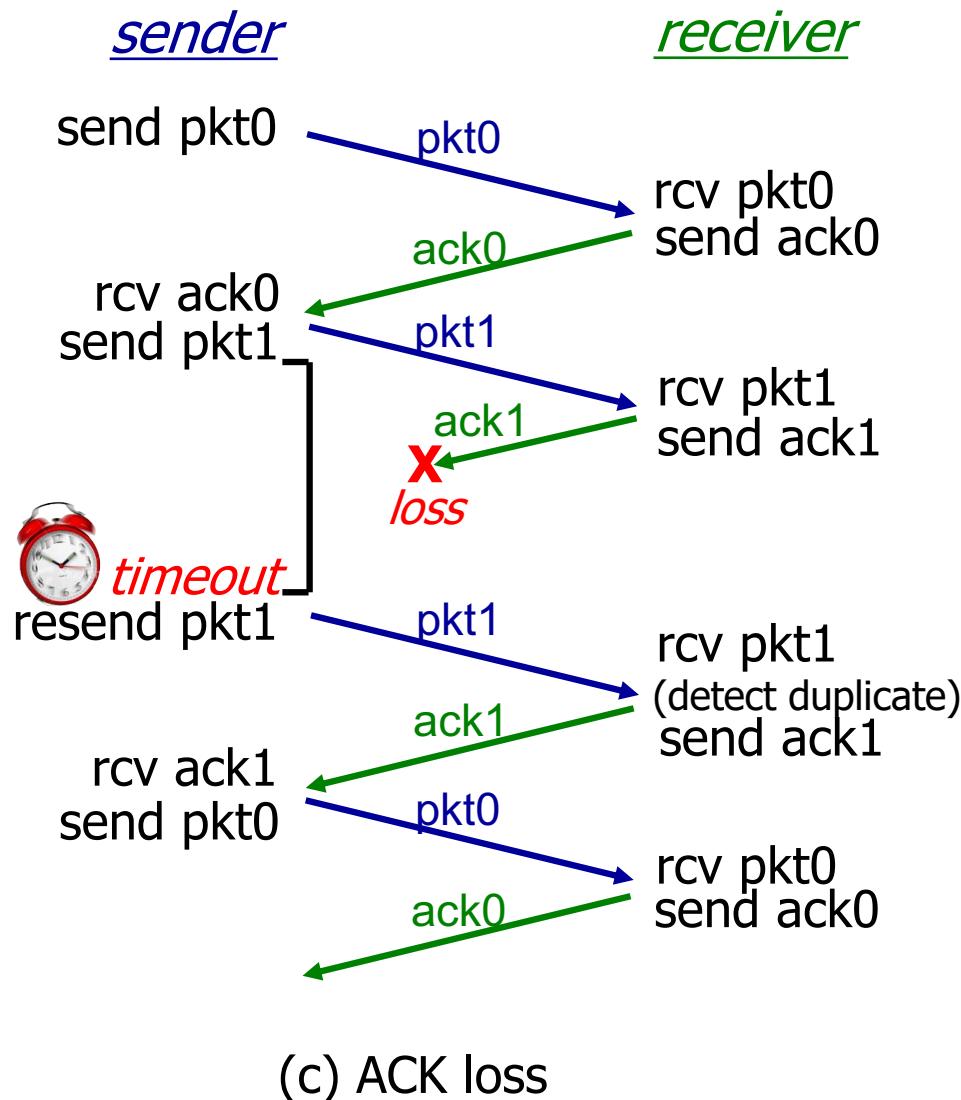
- ❖ retransmits if no ACK received in this time
- ❖ if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- ❖ requires countdown timer

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

# Quiz: Reliable Data Transfer



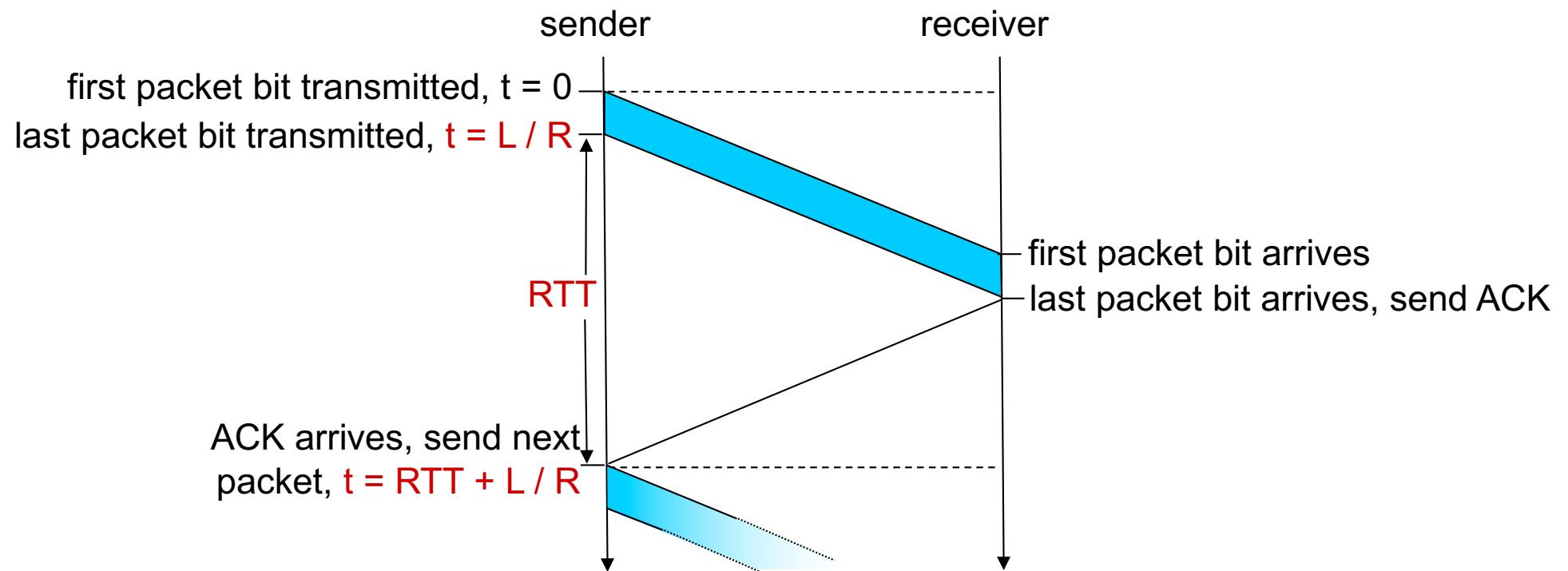
- ❖ If packets (and ACKs and NACKs) could be lost which of the following is true of RDT 2.1 (or 2.2)?
  - a) Reliable in-order delivery is still achieved
  - b) The protocol will get stuck
  - c) The protocol will continue making progress but may skip delivering some messages

# 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

## 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

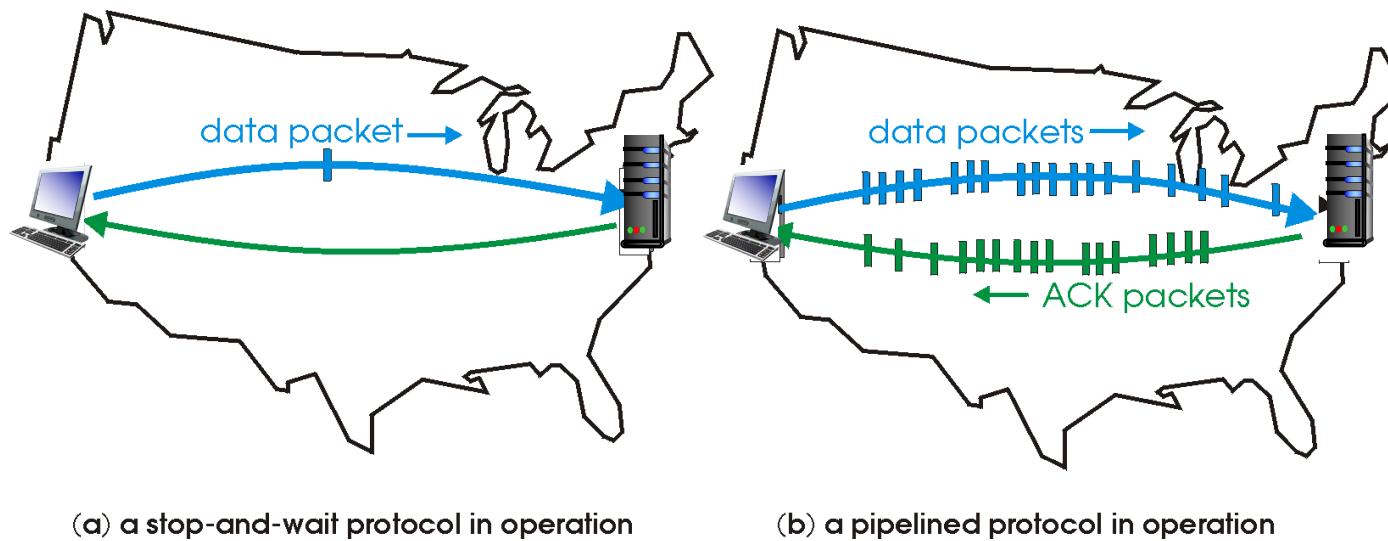
$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

- 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

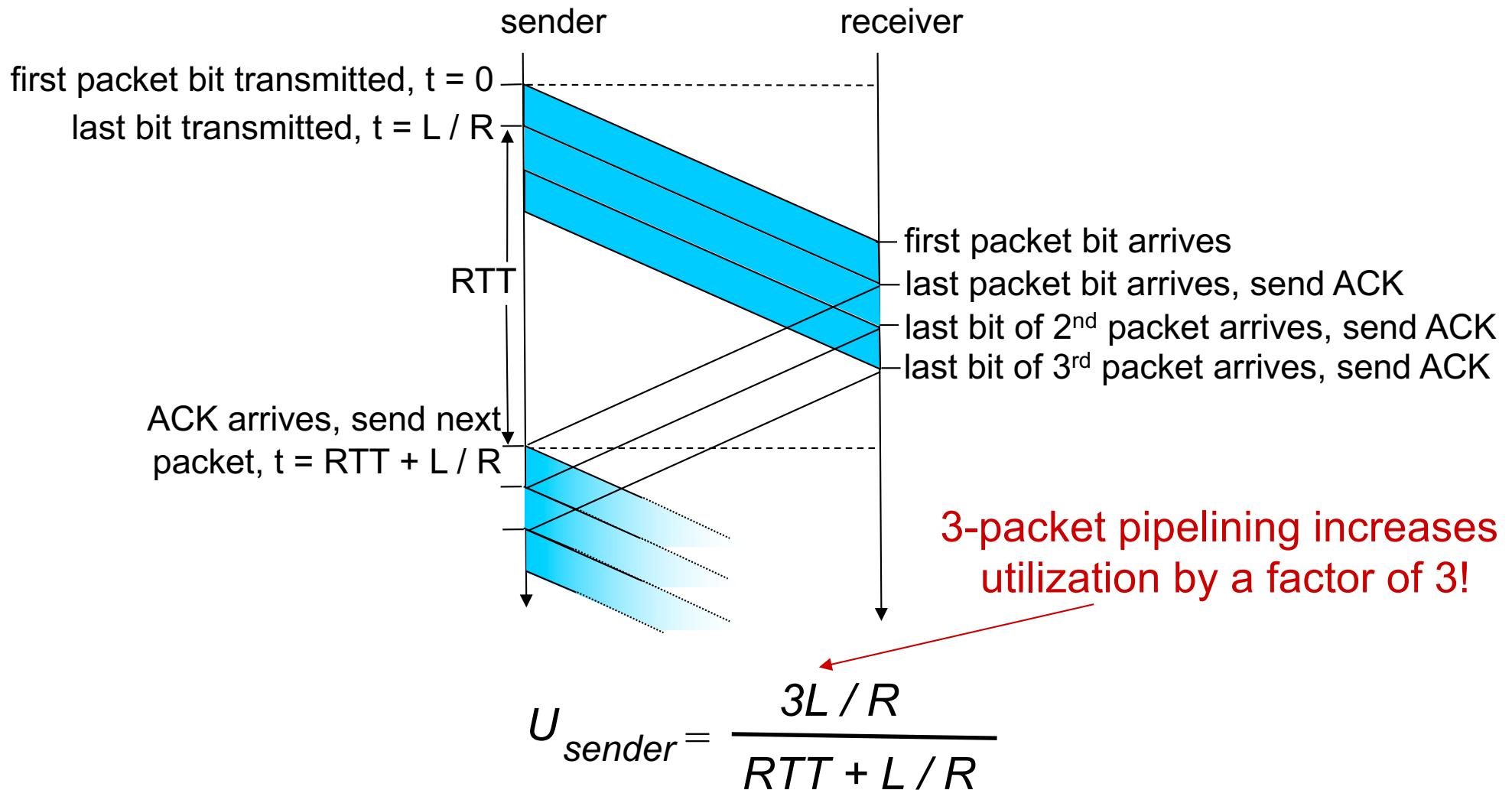
**pipelining:** sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- 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:

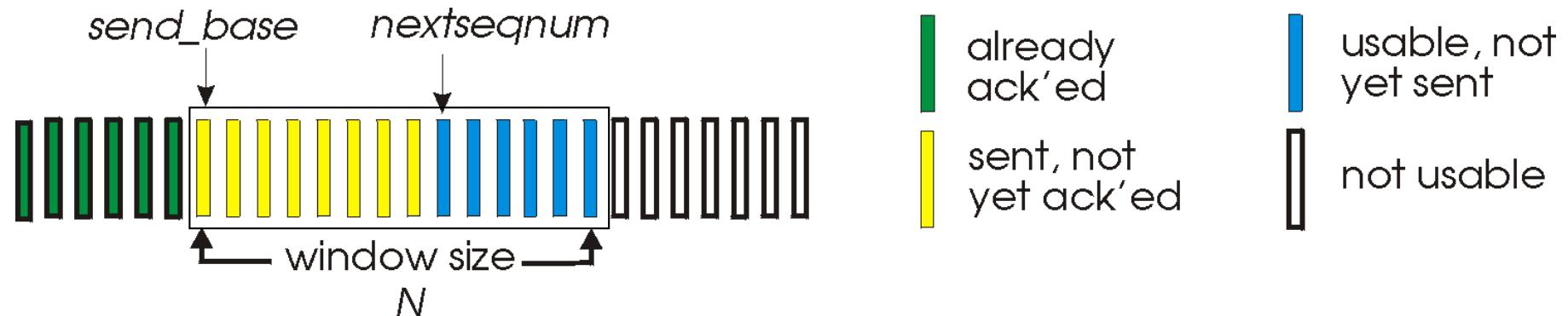
- Sender can have up to N unacked packets in pipeline
- Sender has **single timer** for oldest unacked packet, when timer expires, retransmit *all* unacked packets
- There is no buffer available at Receiver, out of order packets are discarded
- Receiver only sends **cumulative ack**, doesn't ack new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



*pkt 2 timeout*

*receiver*

receive pkt0, send ack0  
receive pkt1, send ack1

receive pkt3, discard,  
(re)send ack1

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

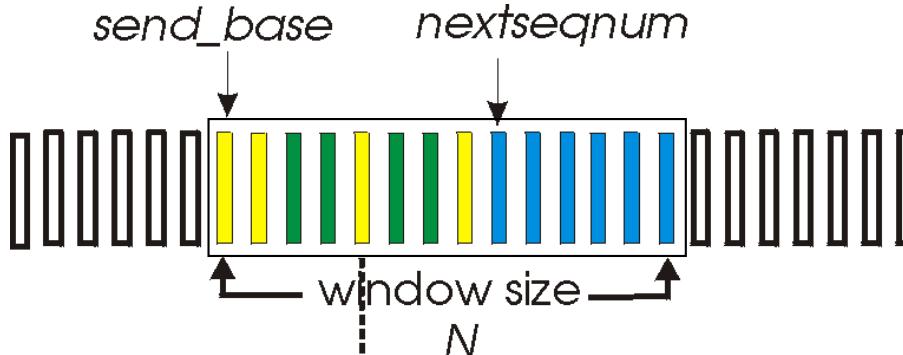
rcv pkt2, deliver, send ack2  
rcv pkt3, deliver, send ack3  
rcv pkt4, deliver, send ack4  
rcv pkt5, deliver, send ack5

# Selective repeat

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

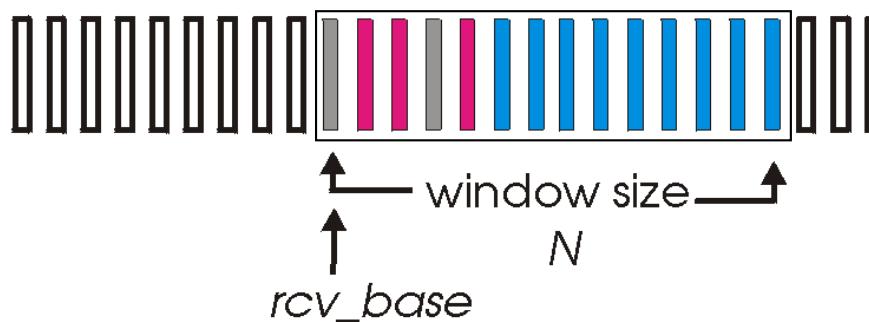
Applet: [http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_3/applets/SelectRepeat/SR.html](http://media.pearsoncmg.com/aw/aw_kurose_network_3/applets/SelectRepeat/SR.html)

# 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]:**

- ❖ 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

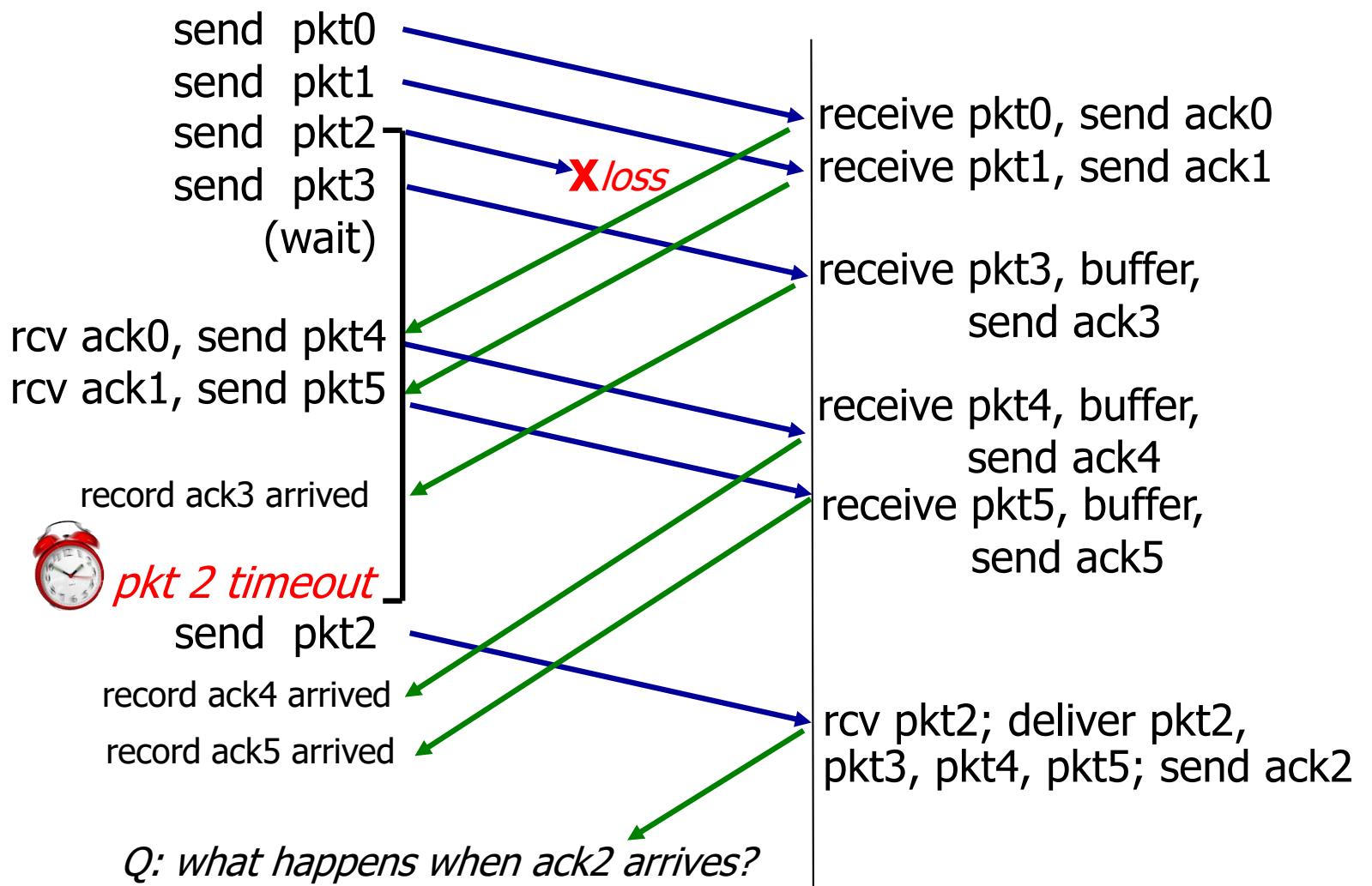
sender window ( $N=4$ )

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



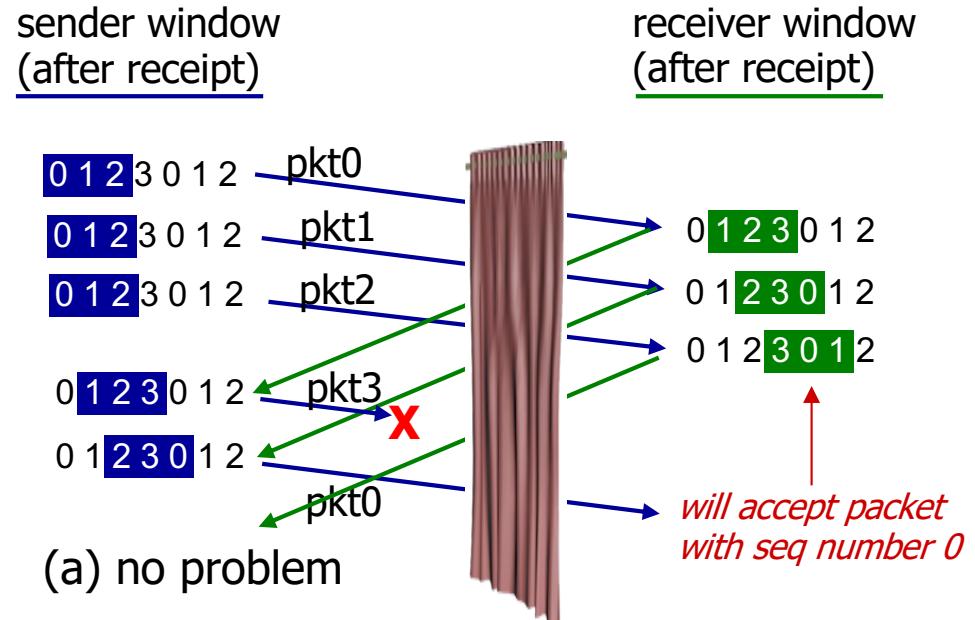
# Selective repeat: dilemma

example:

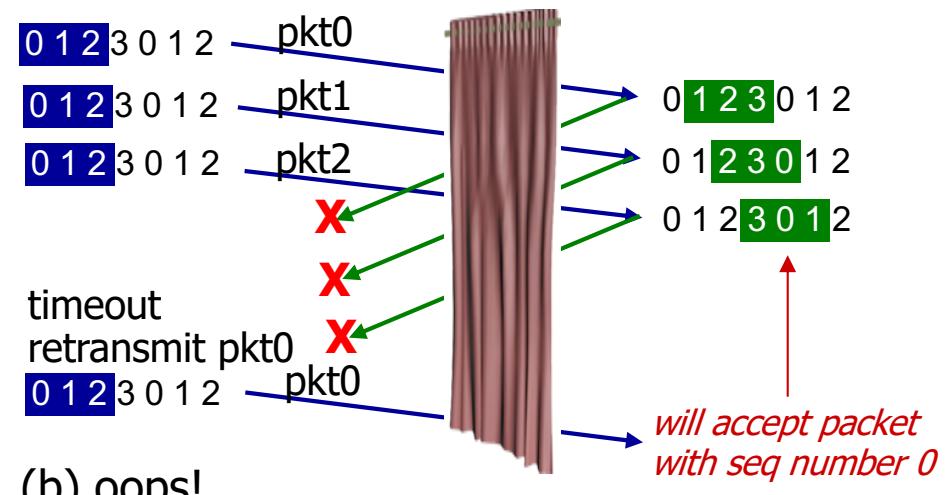
- ❖ seq #'s: 0, 1, 2, 3
- ❖ window size=3
- ❖ receiver sees no difference in two scenarios!
- ❖ duplicate data accepted as new in (b)

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



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



# 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]
  - 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

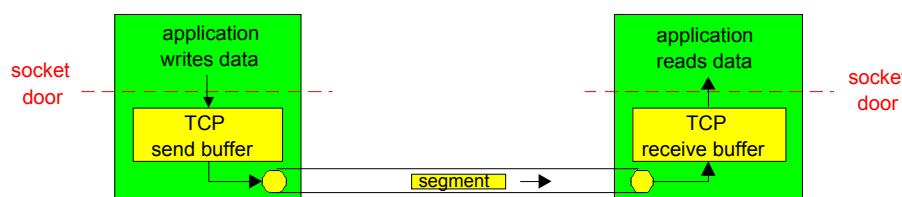
- ❖ How do the sender and receiver keep track of outstanding pipelined segments?
- ❖ How many segments should be pipelined?
- ❖ How do we choose sequence numbers?
- ❖ What does connection establishment and teardown look like?
- ❖ How should we choose timeout values?

# TCP: Overview

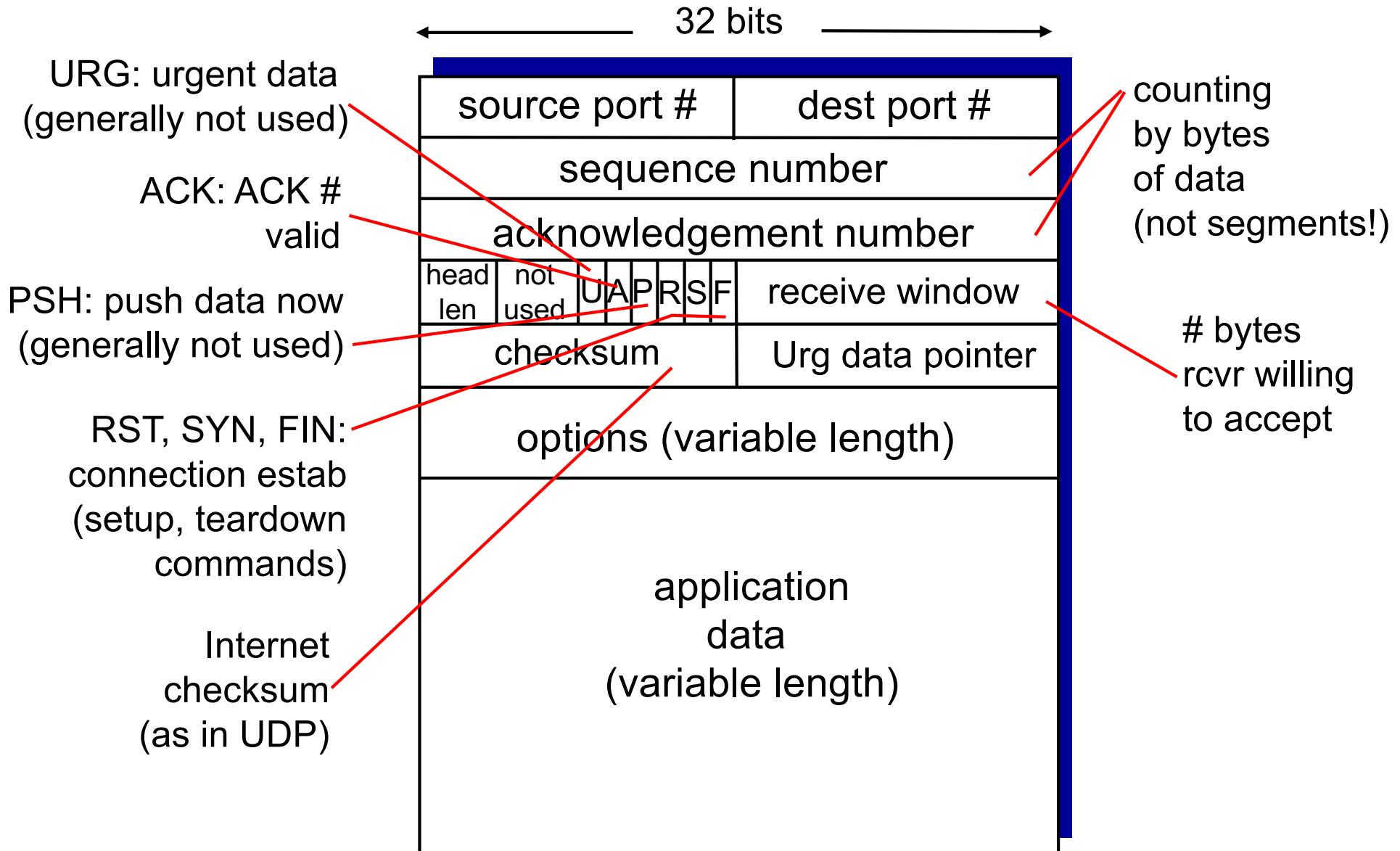
RFCs: 793, 1122, 1323, 2018, 2581

- ❖ **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**

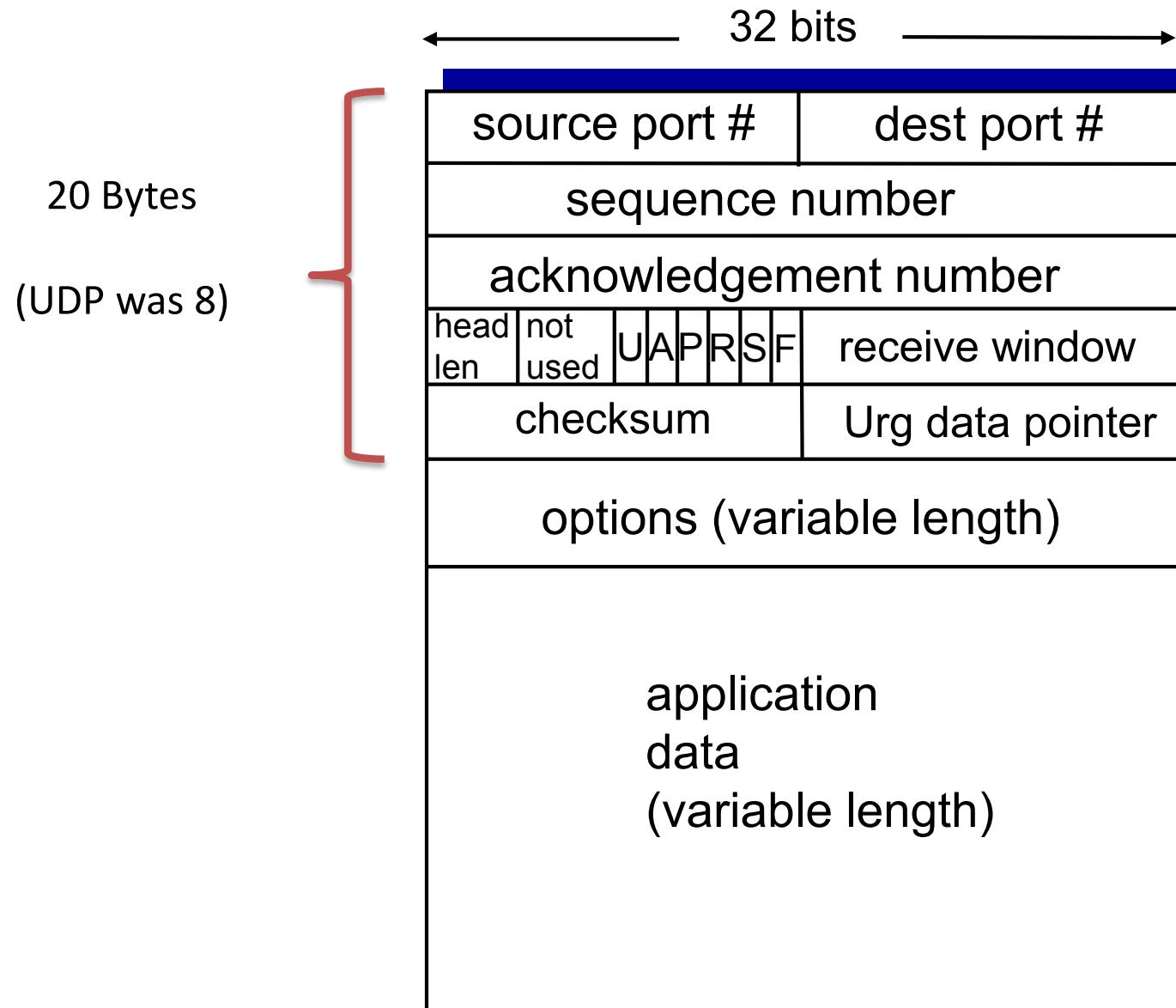
- ❖ **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- ❖ **connection-oriented:**
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- ❖ **flow controlled:**
  - sender will not overwhelm receiver



# 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)

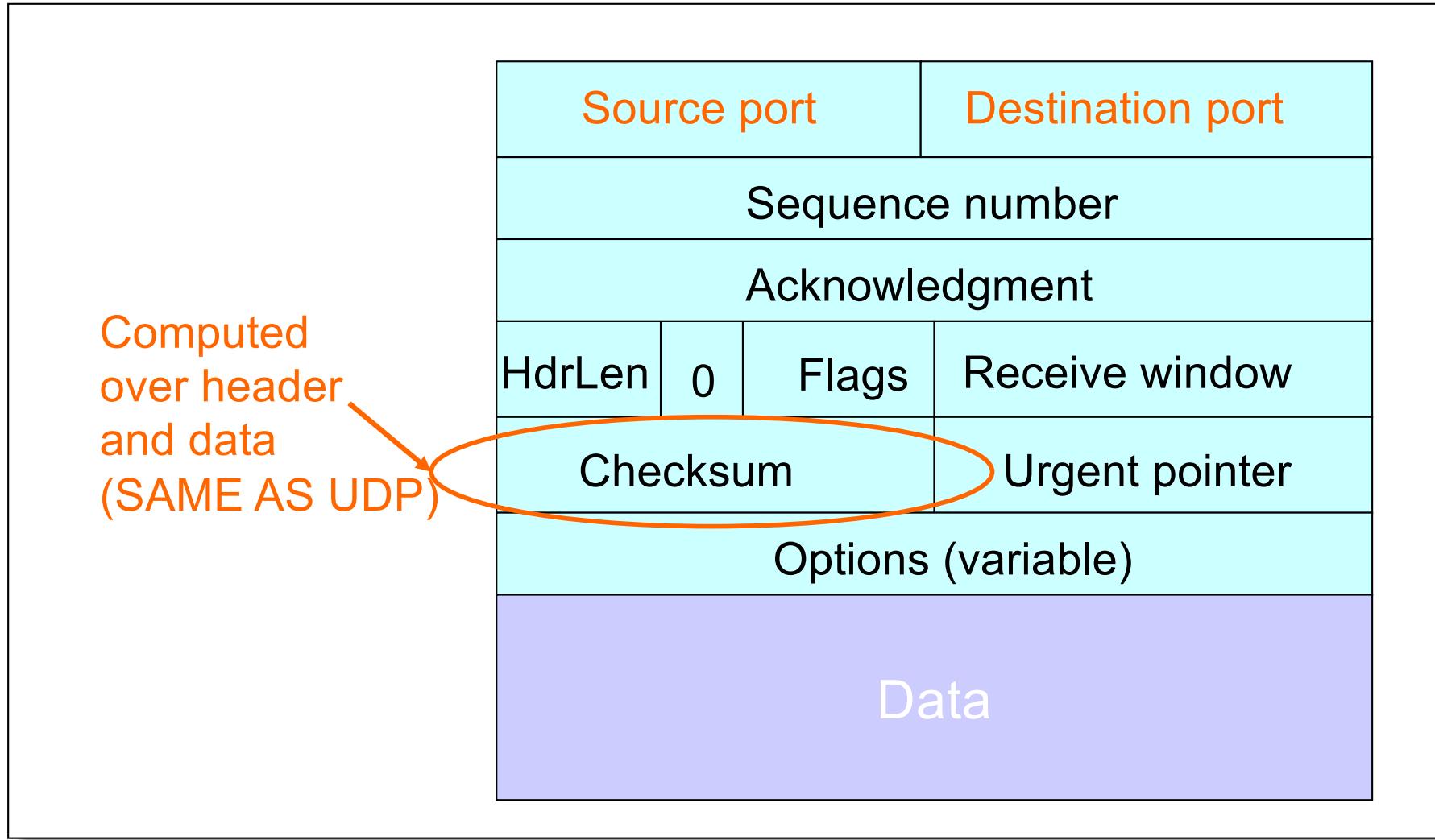
# What does TCP do?

Many of our previous ideas, but some key differences

- ❖ Checksum

# TCP Header

---



# What does TCP do?

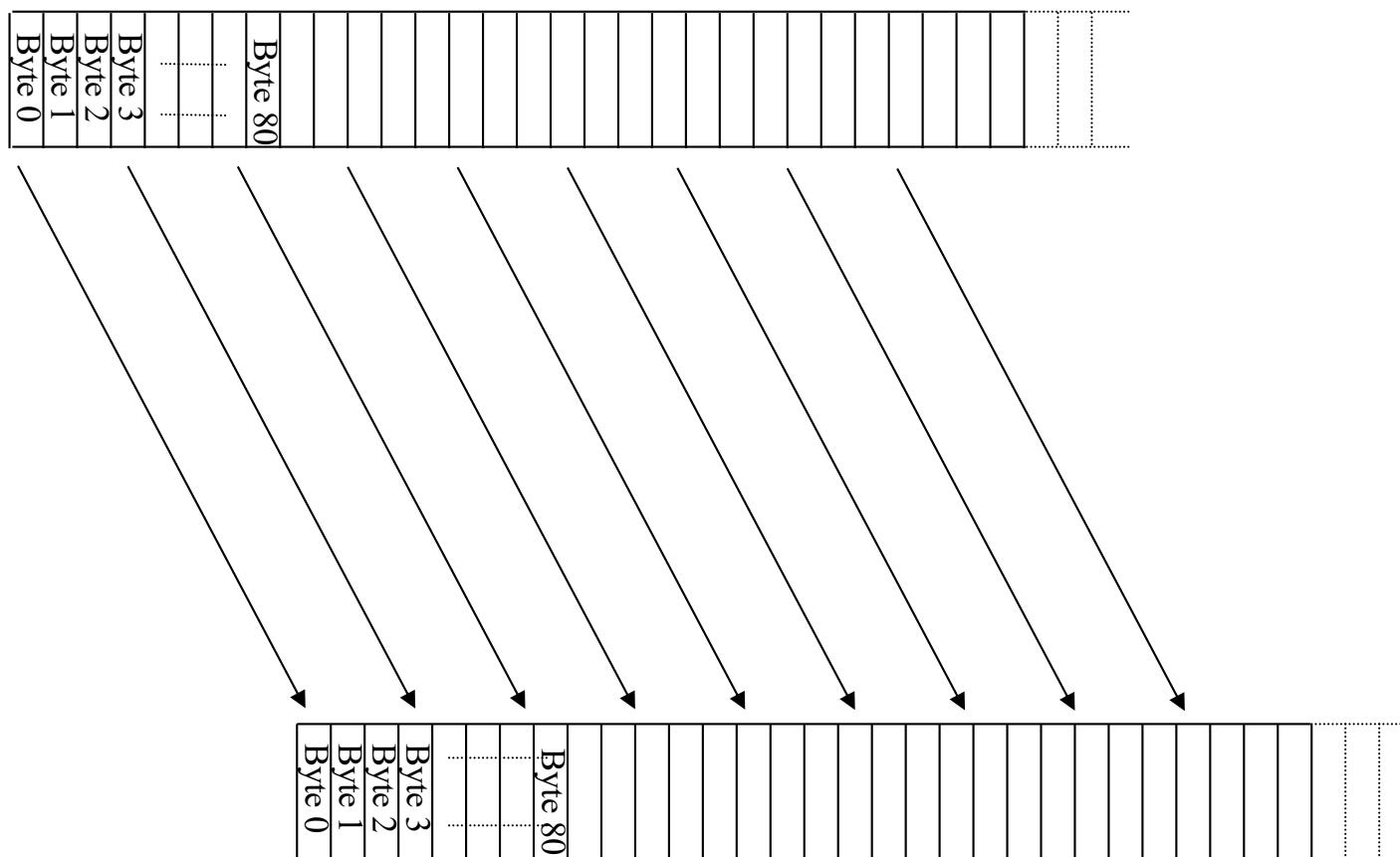
---

Many of our previous ideas, but some key differences

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

# TCP “Stream of Bytes” Service ..

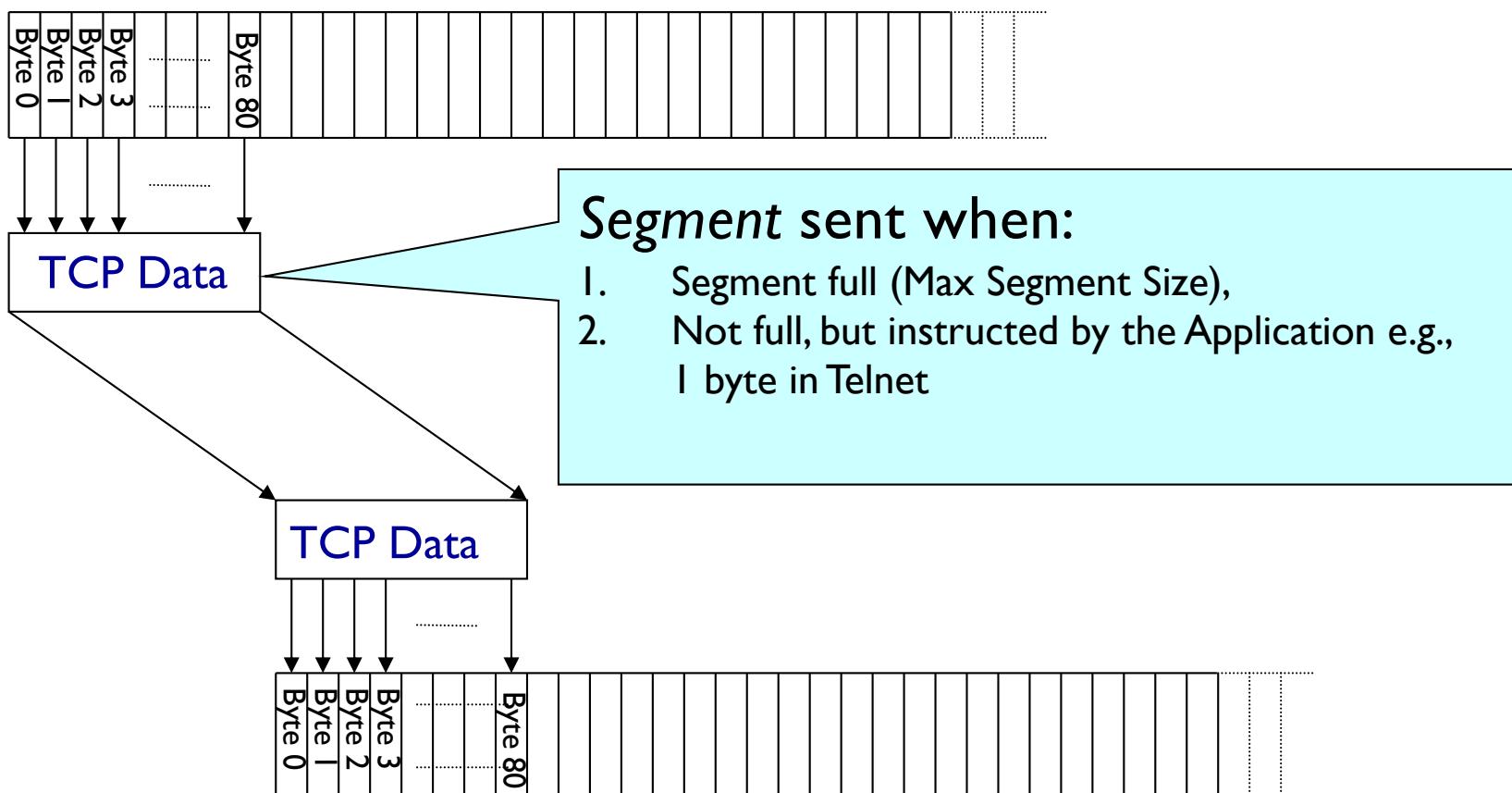
Application @ Host A



Application @ Host B

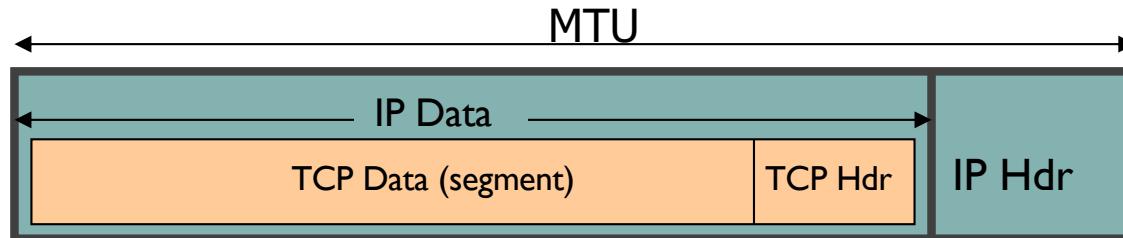
# .. Provided Using TCP “Segments”

Host A



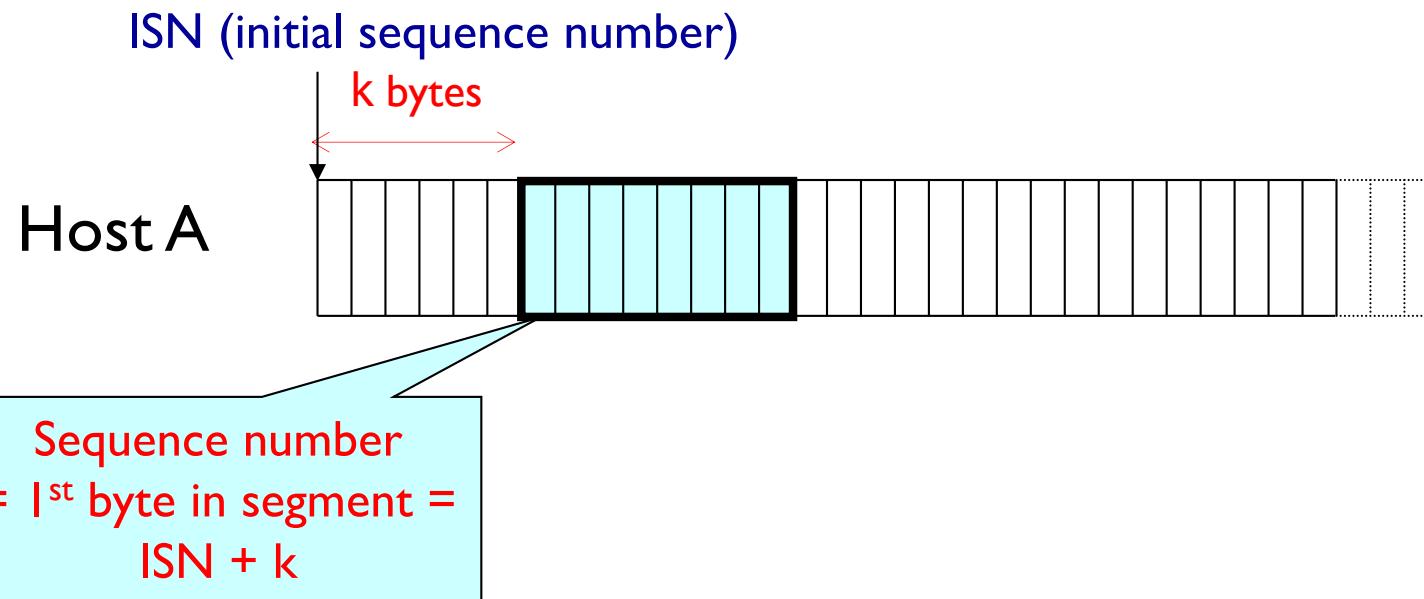
Host B

# 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
  - $MSS = MTU - 20 \text{ (min IP header)} - 20 \text{ ( min TCP header )}$

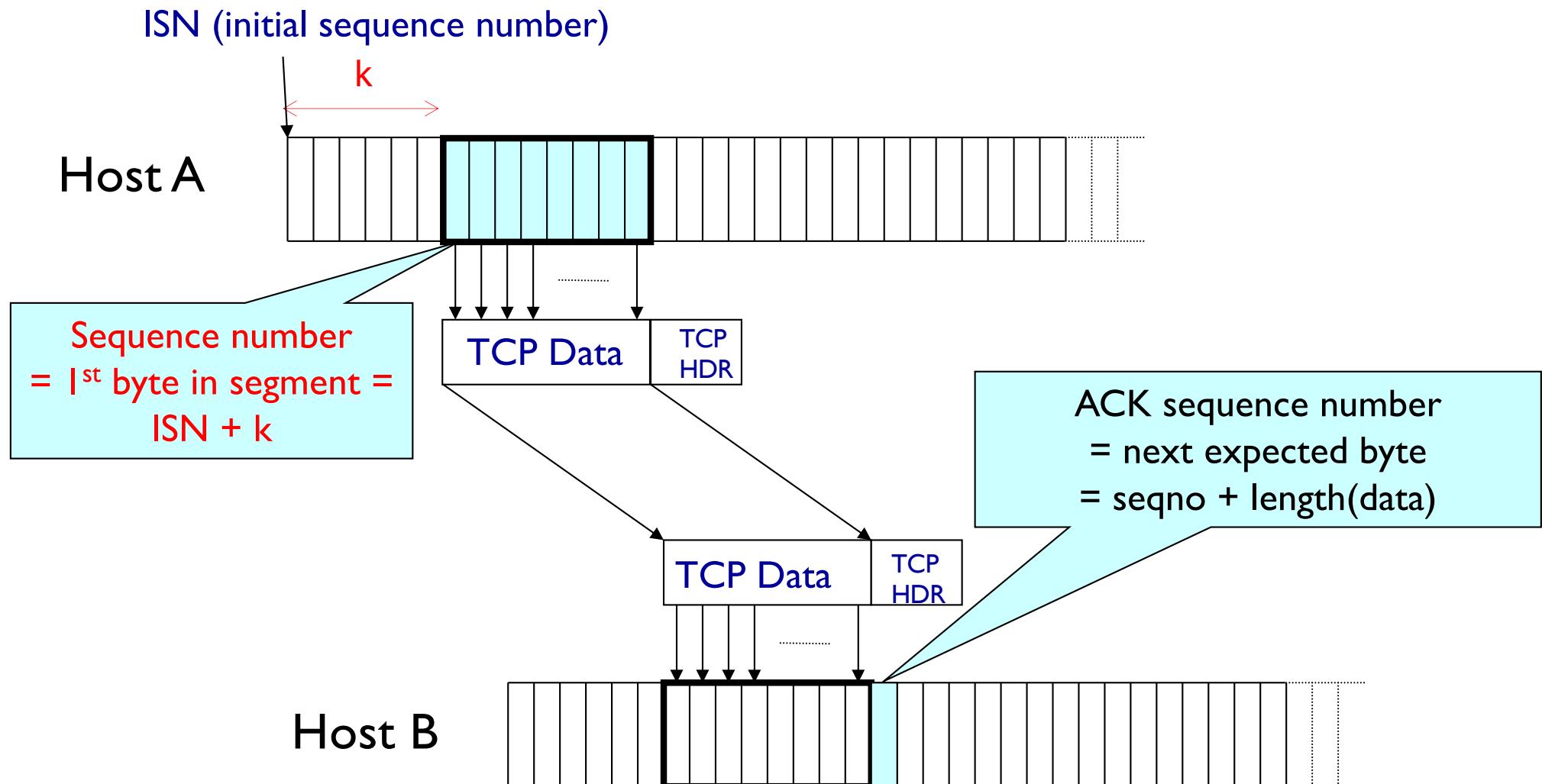
# Sequence Numbers



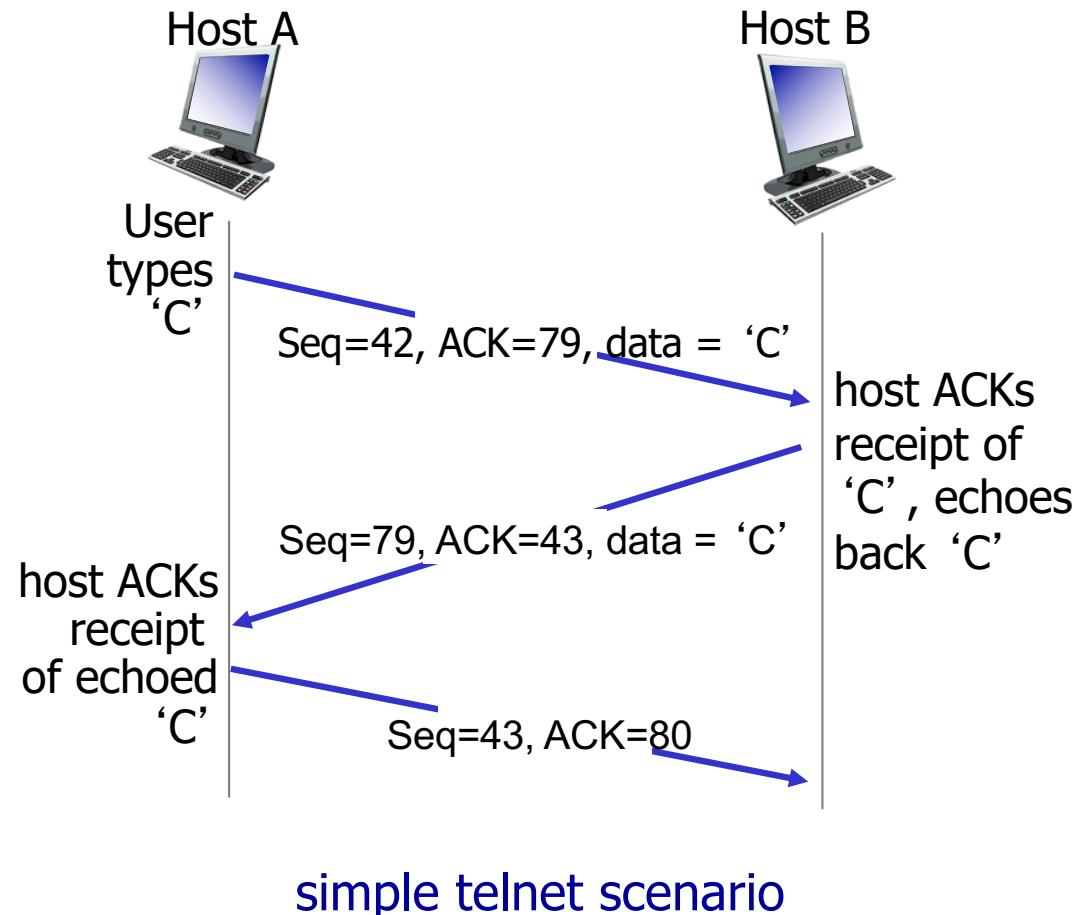
## Sequence numbers:

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

# Sequence & Ack Numbers



# TCP seq. numbers, ACKs



# What does TCP do?

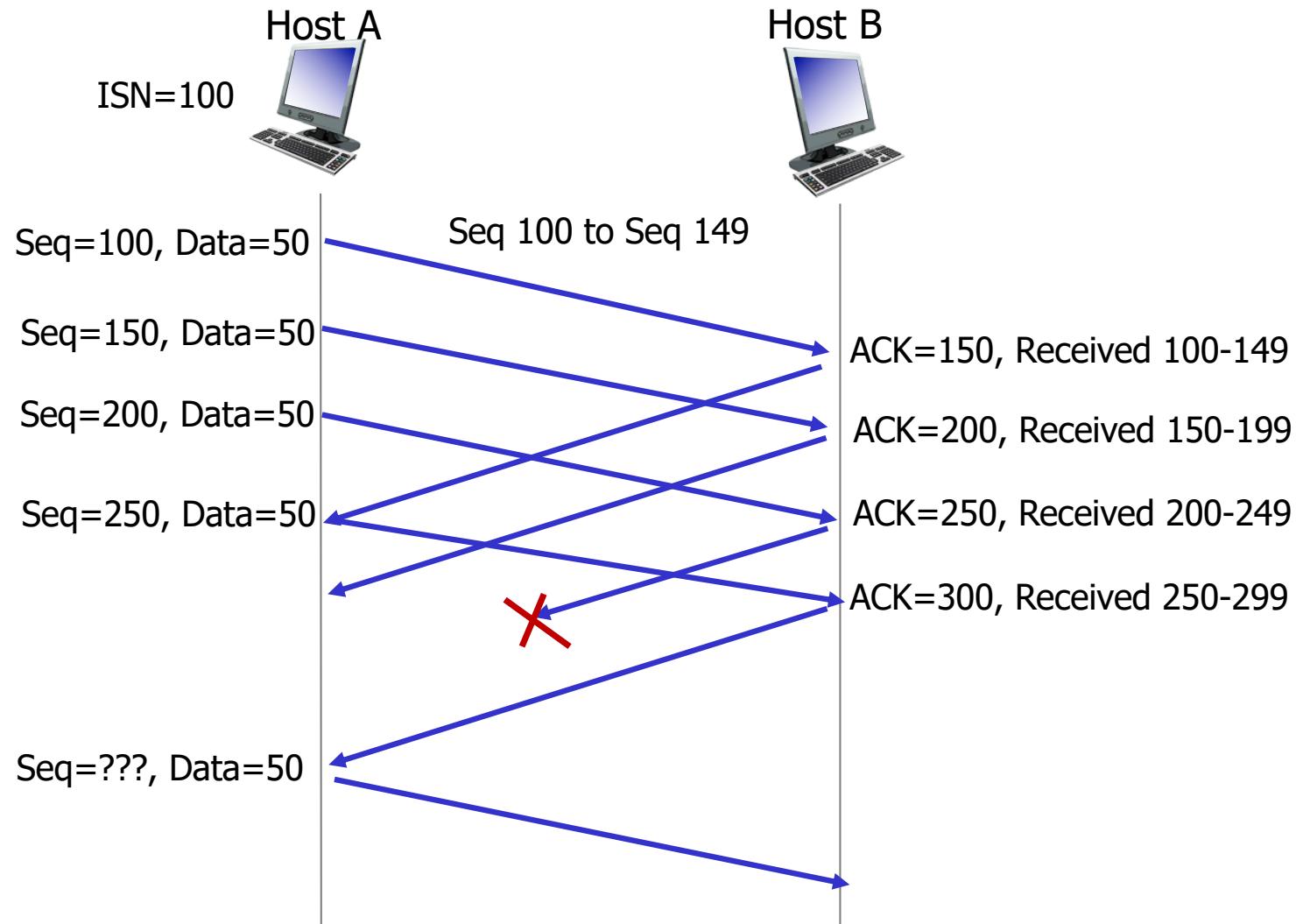
Most of our previous tricks, but a few differences

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

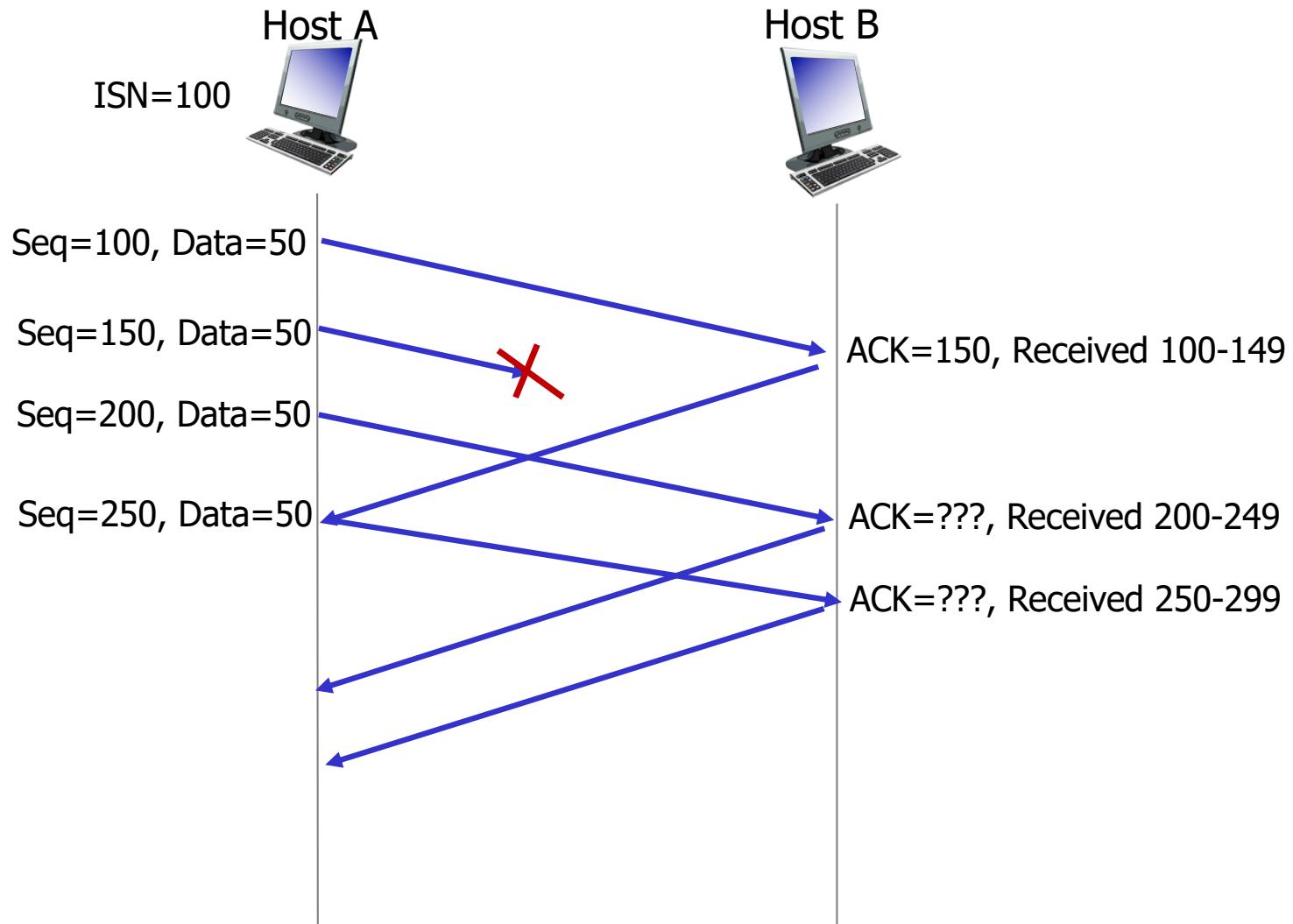
# ACKing and Sequence Numbers

- ❖ 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

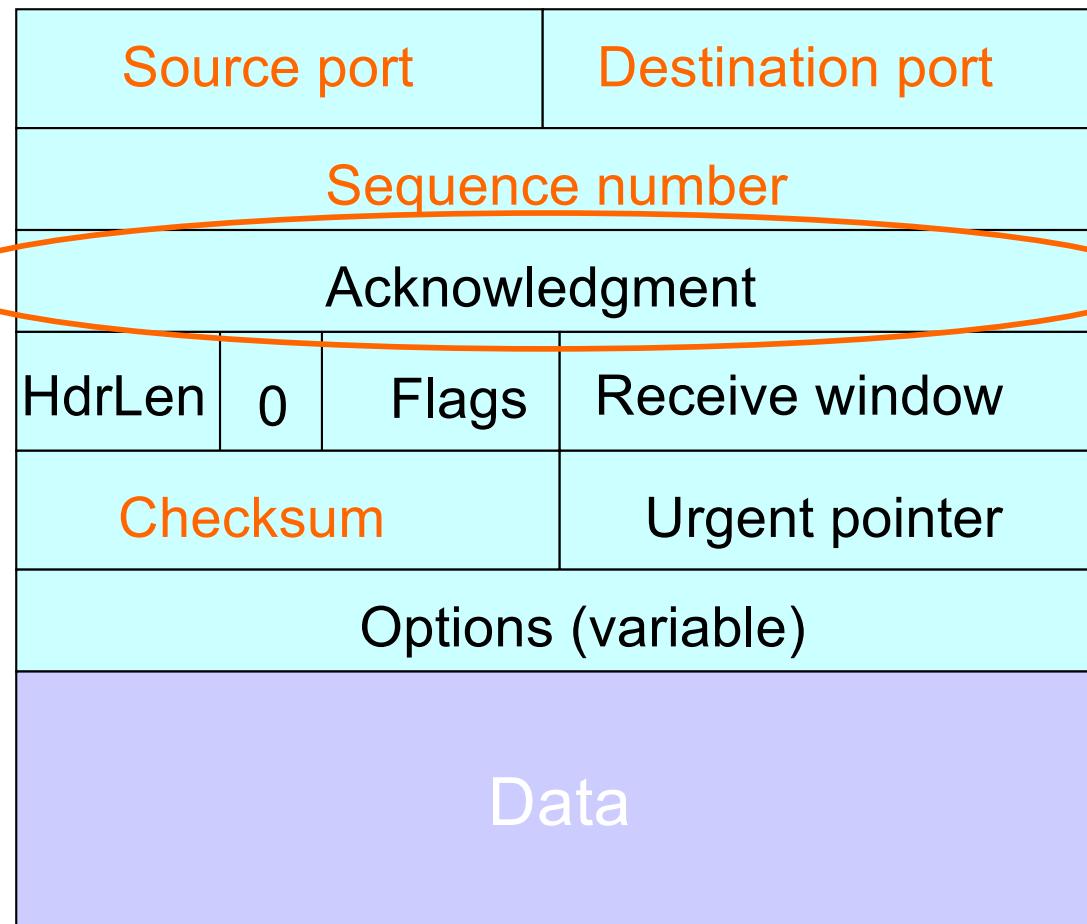
- ❖ Sender: seqno=X, length=B
- ❖ Receiver: ACK=X+B
- ❖ Sender: seqno=X+B, length=B
- ❖ Receiver: ACK=X+2B
- ❖ Sender: seqno=X+2B, length=B
  
- ❖ Seqno of next packet is same as last ACK field

# 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

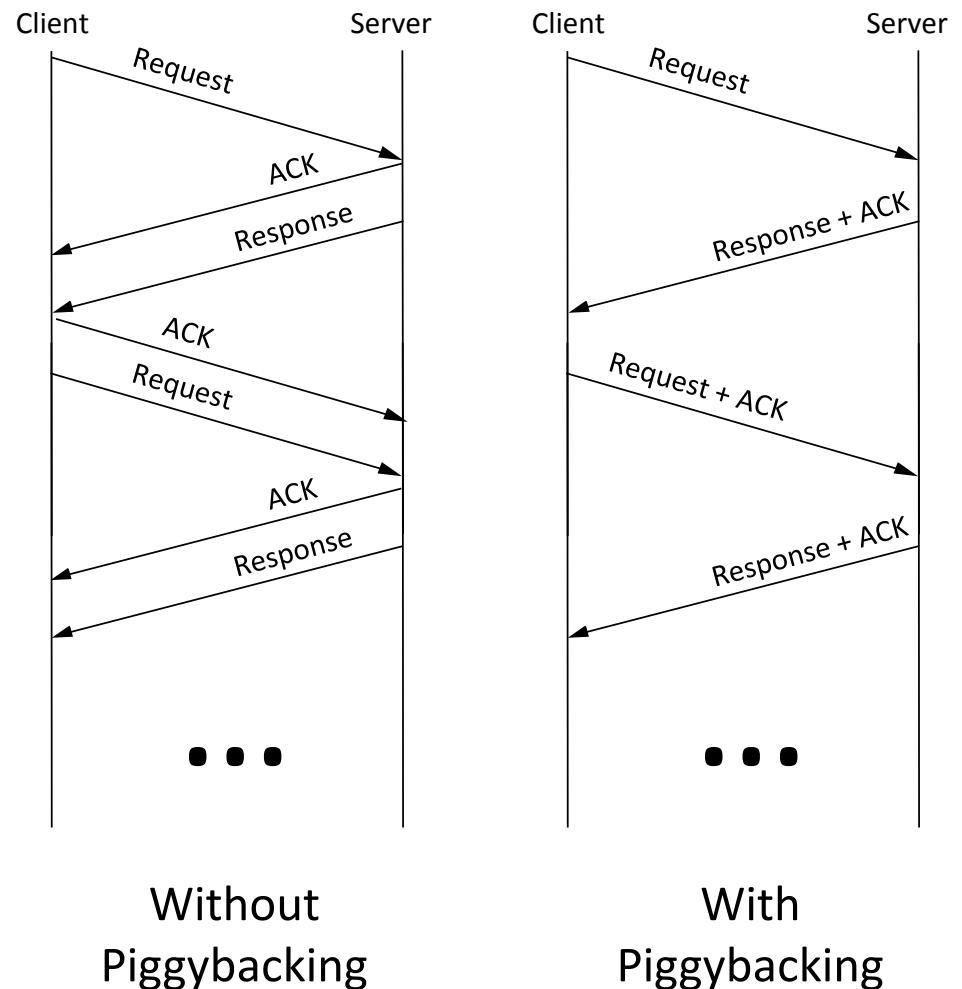
# TCP Header

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



# Piggybacking

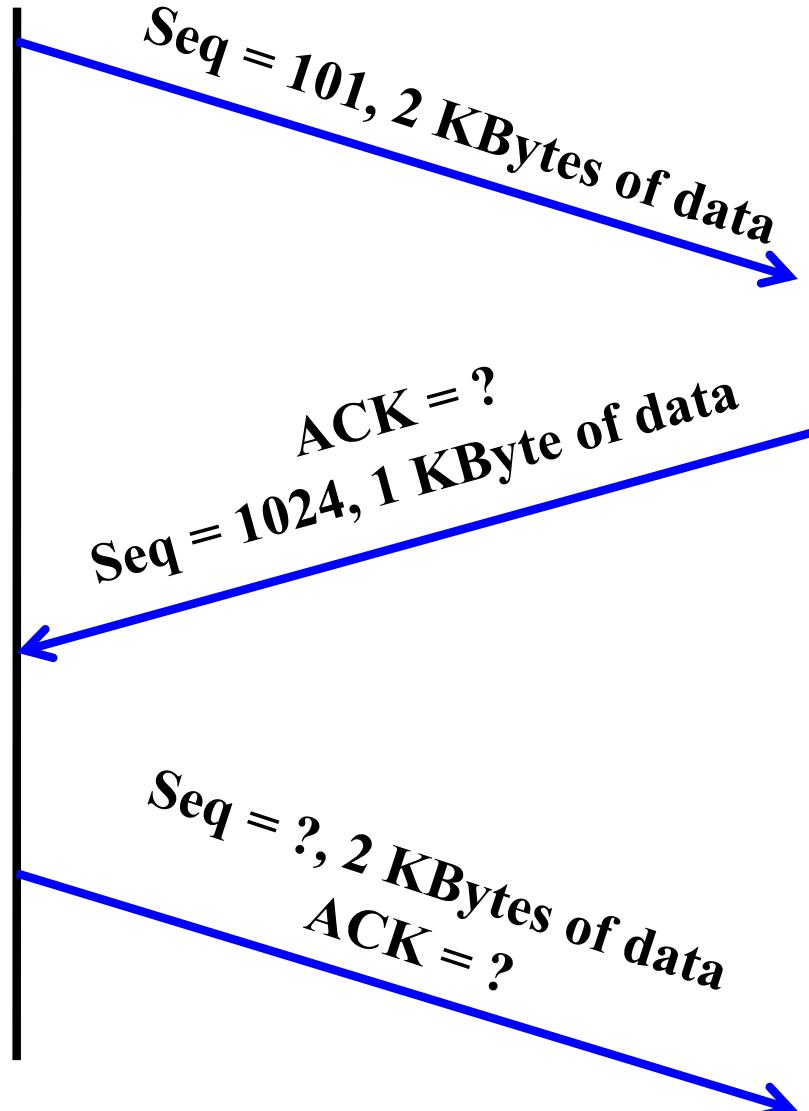
- ❖ So far, we've assumed distinct “sender” and “receiver” roles
- ❖ In reality, usually both sides of a connection send some data



Without  
Piggybacking

With  
Piggybacking

# Quiz



# 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

---

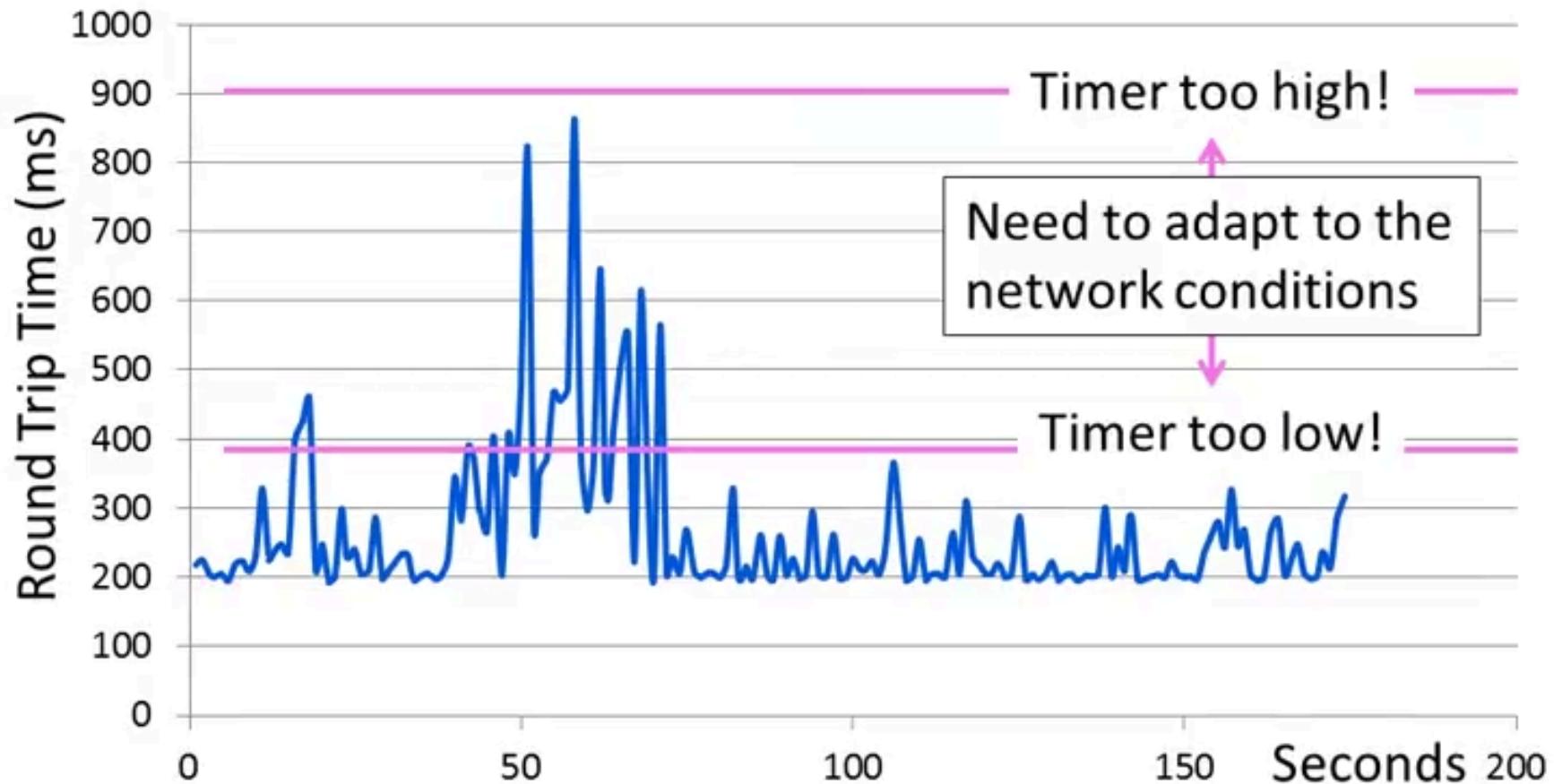
- ❖ Sender sends packets with 100Bytes and sequence numbers:
  - 100, 200, 300, 400, 500, 600, 700, 800, 900, ...
- ❖ Assume the fifth packet (seq. no. 500) is lost, but no others
- ❖ Stream of ACKs will be:
  - 200, 300, 400, 500, 500, 500, ...

# 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

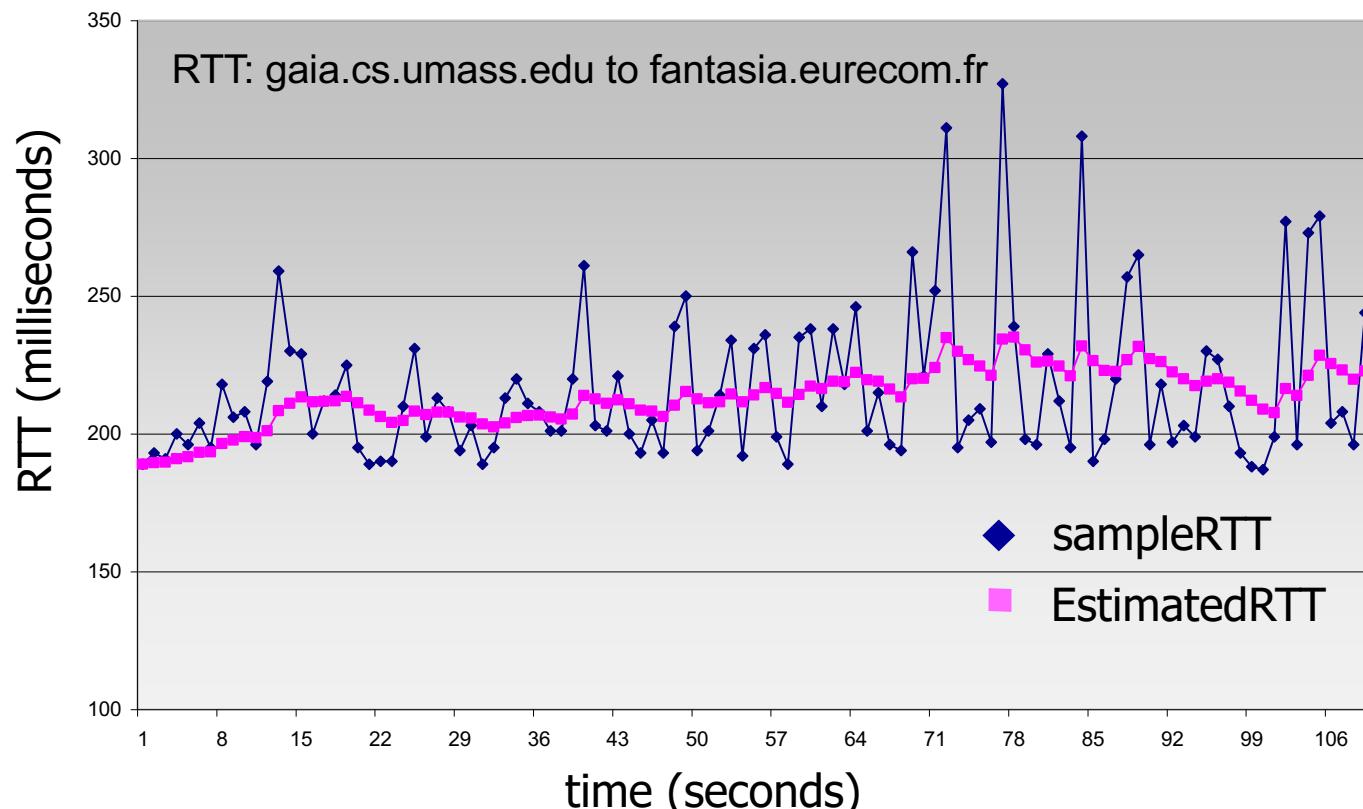
**Q:** how to estimate RTT?

- ❖ **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,  $\beta = 0.25$ )

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



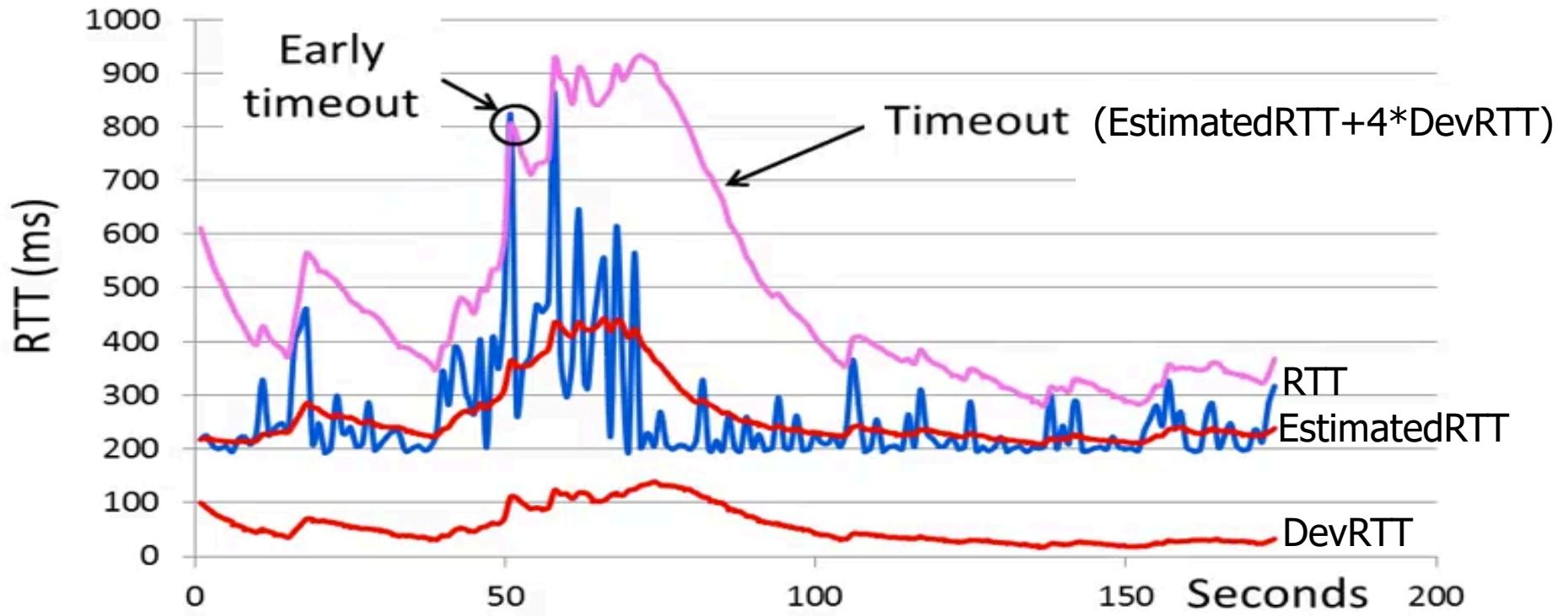
↑  
estimated RTT

↑  
“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



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



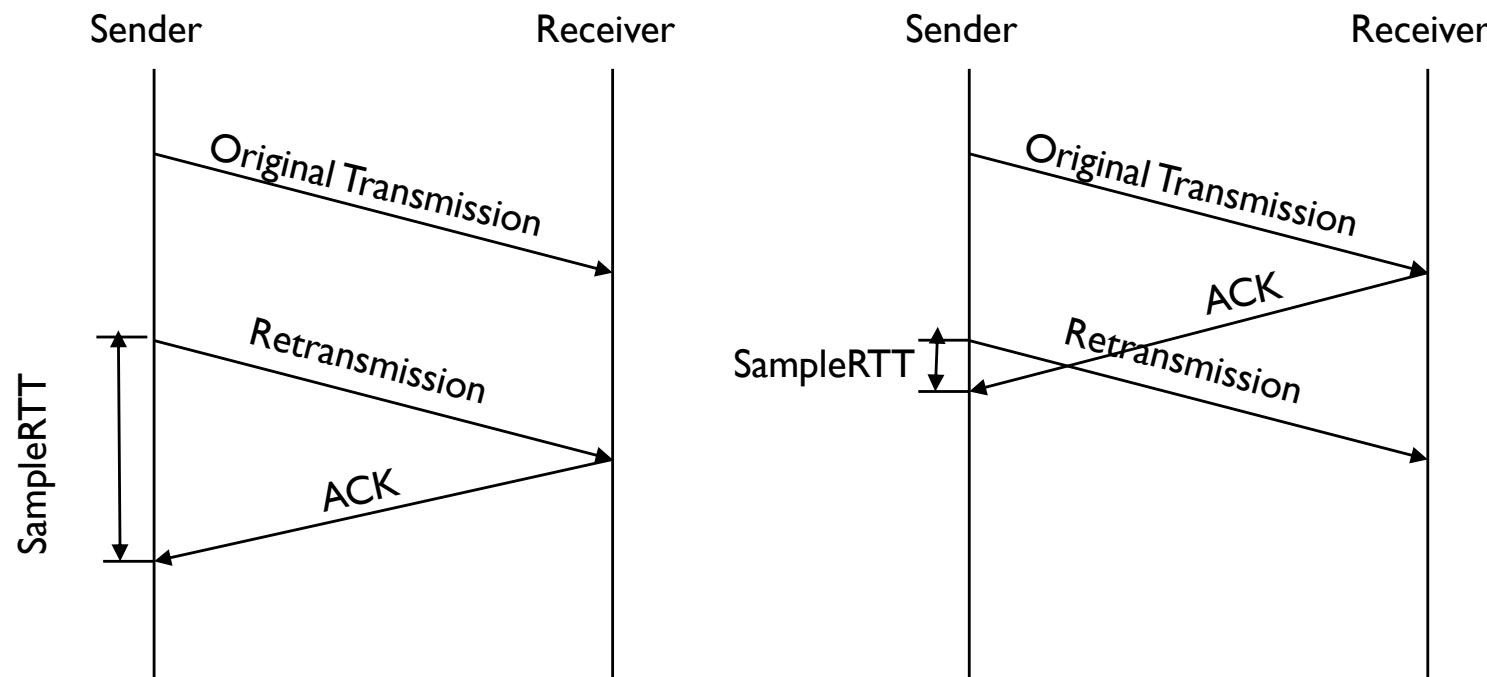
↑  
estimated RTT

↑  
“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

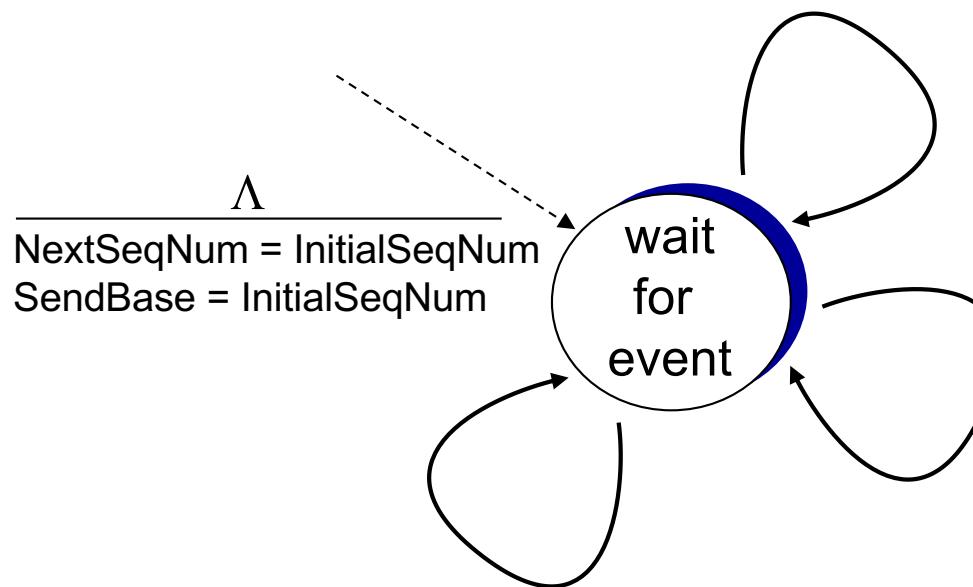
- ❖ restart timer

## *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



$\Lambda$   
NextSeqNum = InitialSeqNum  
SendBase = InitialSeqNum

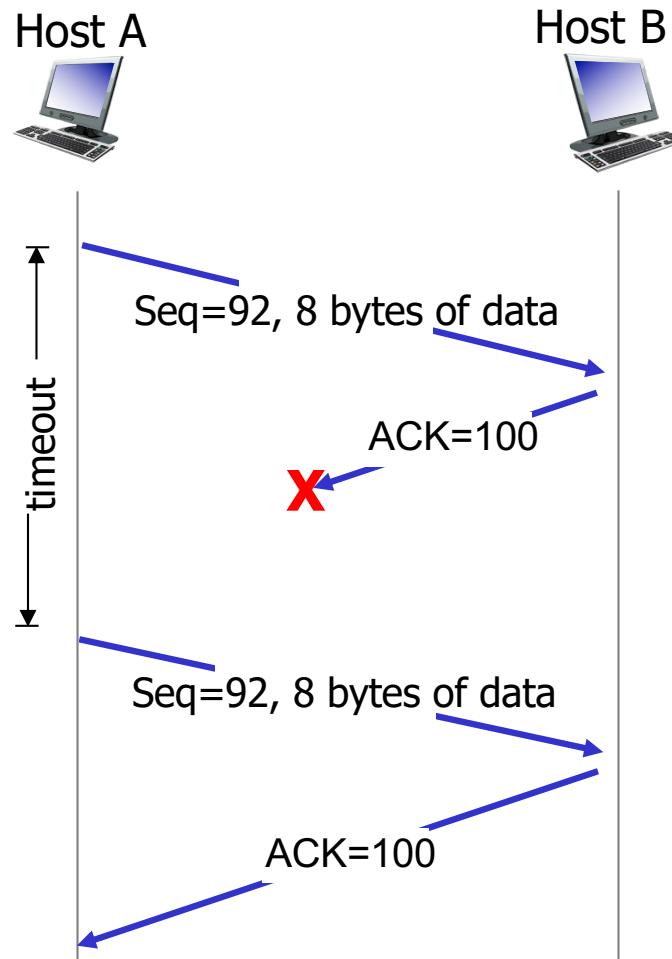
ACK received, with ACK field value  $y$

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

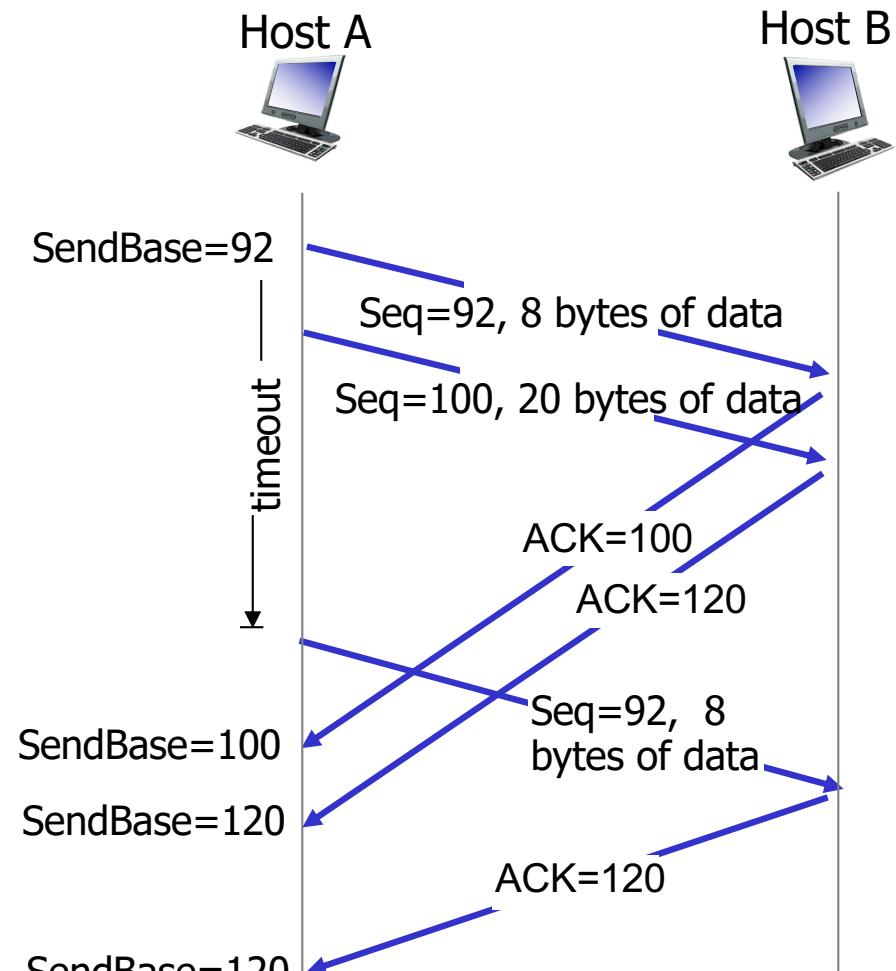
data received from application above  
create segment, seq. #: NextSeqNum  
pass segment to IP (i.e., “send”)  
 $\text{NextSeqNum} = \text{NextSeqNum} + \text{length(data)}$   
if (timer currently not running)  
    start timer

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

# TCP: retransmission scenarios

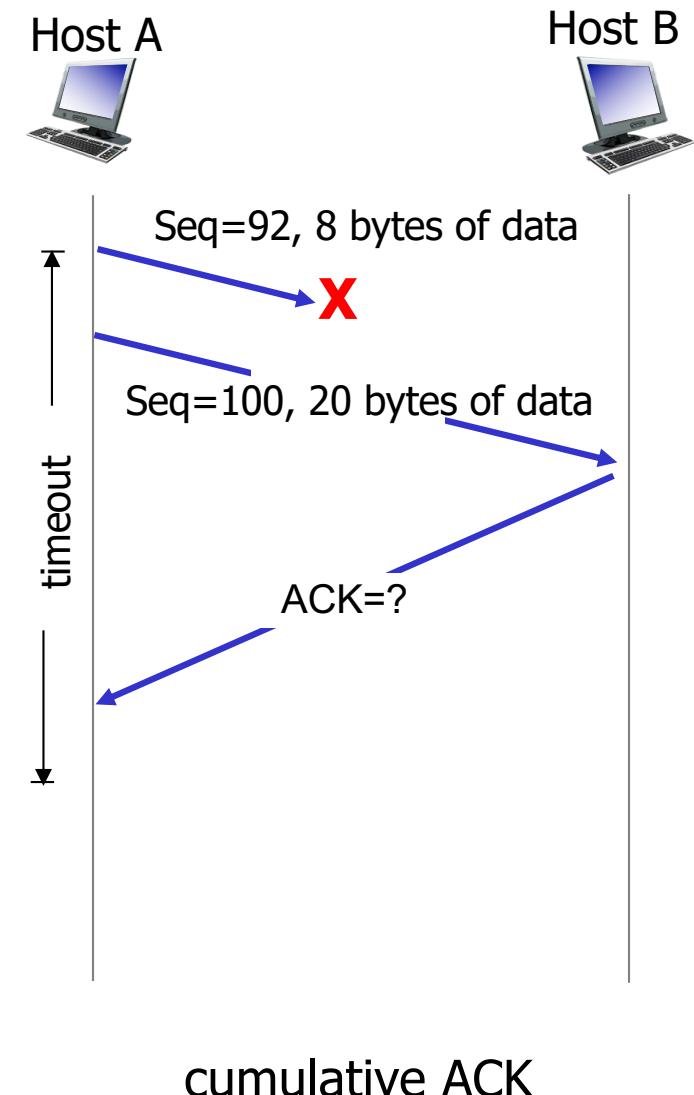
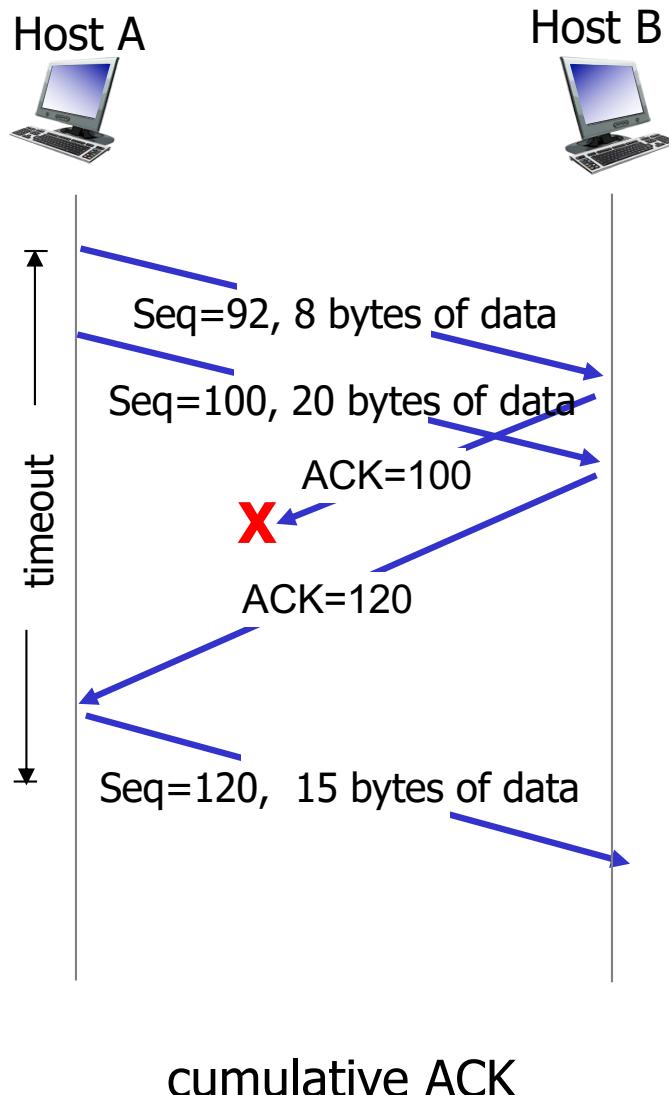


lost ACK scenario



premature timeout

# TCP: retransmission scenarios



# TCP ACK generation [RFC 1122, RFC 2581]

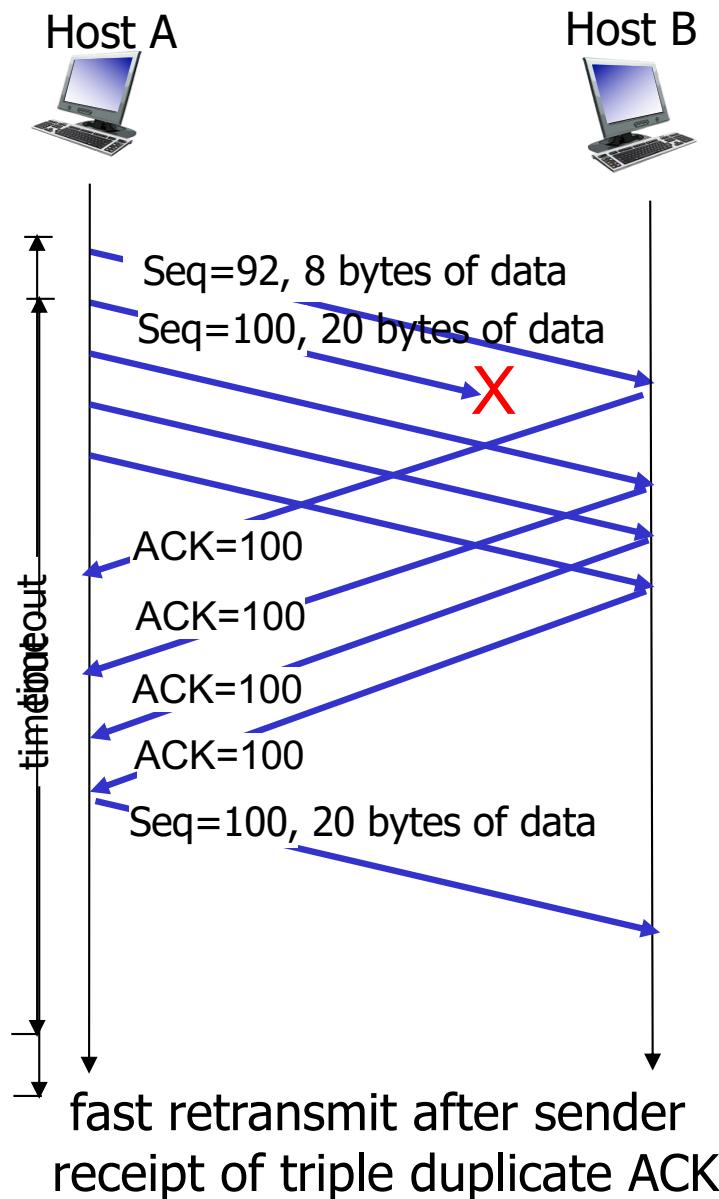
<i>event at receiver</i>	<i>TCP receiver action</i>
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	<b>delayed ACK.</b> Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK 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

# 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

# Quiz: TCP Timeout?



A TCP Sender maintains an EstimatedRTT of 100ms. Suppose the next SampleRTT is 108. Which of the following is true about the sender?

- A. It will increase EstimatedRTT but leave timeout unchanged
- B. It will increase the timeout
- C. Whether it increases EstimatedRTT will depend on the deviation
- D. Whether it increases the timeout will depend on the deviation

# 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

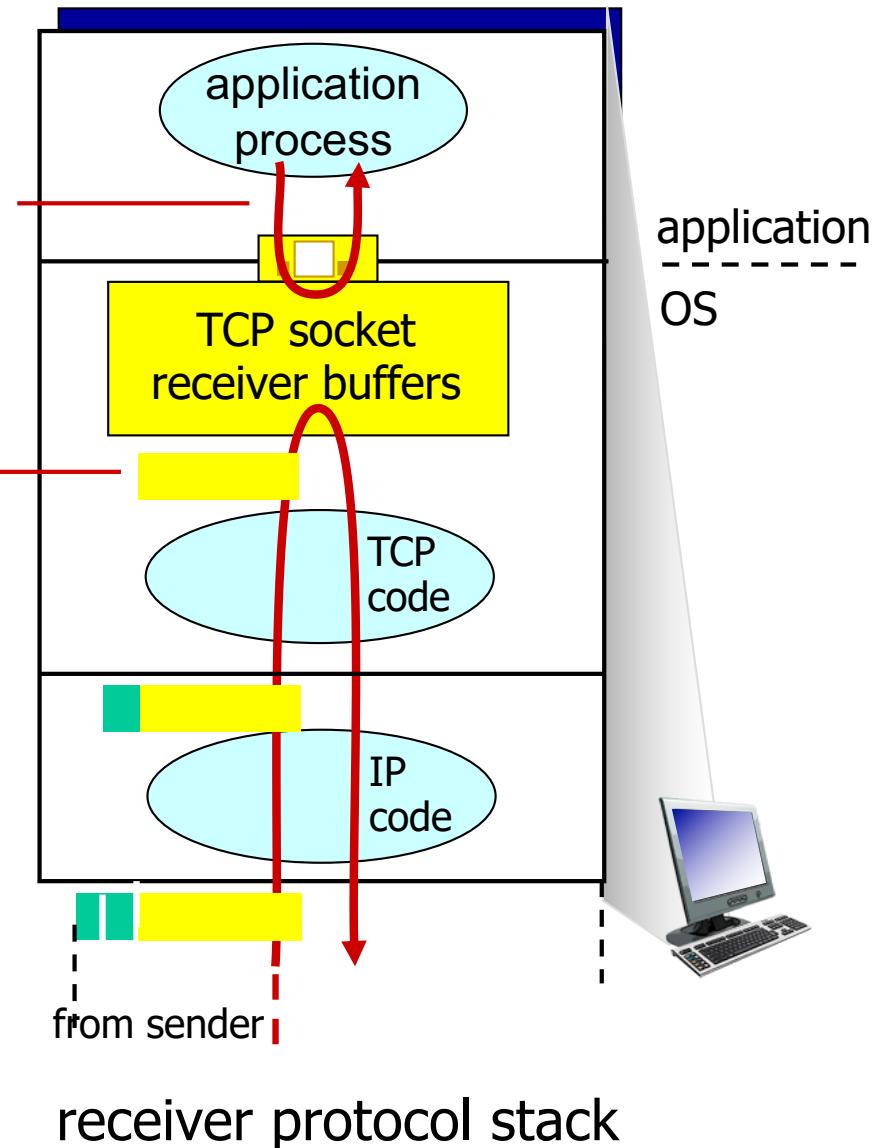
3.7 TCP congestion control

# TCP flow control

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

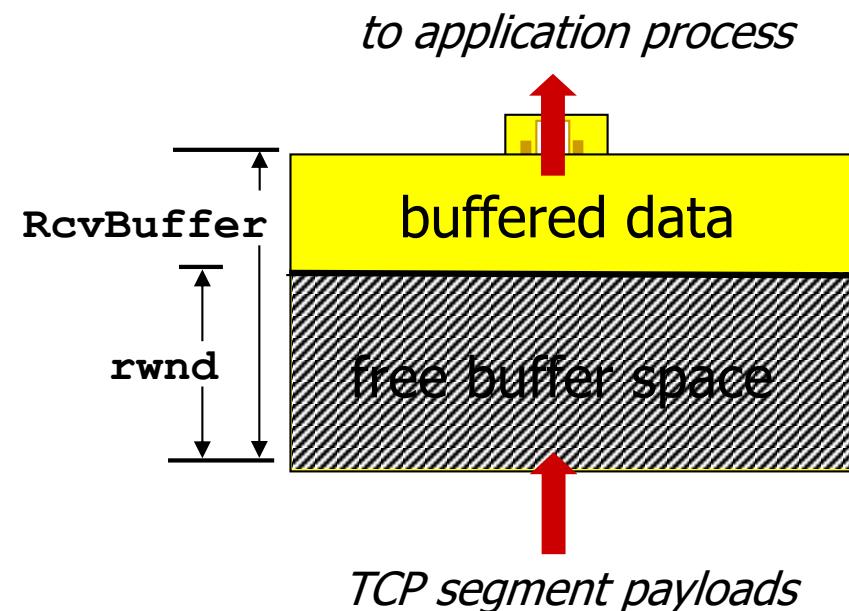
... slower than TCP  
receiver is delivering  
(sender is sending)

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



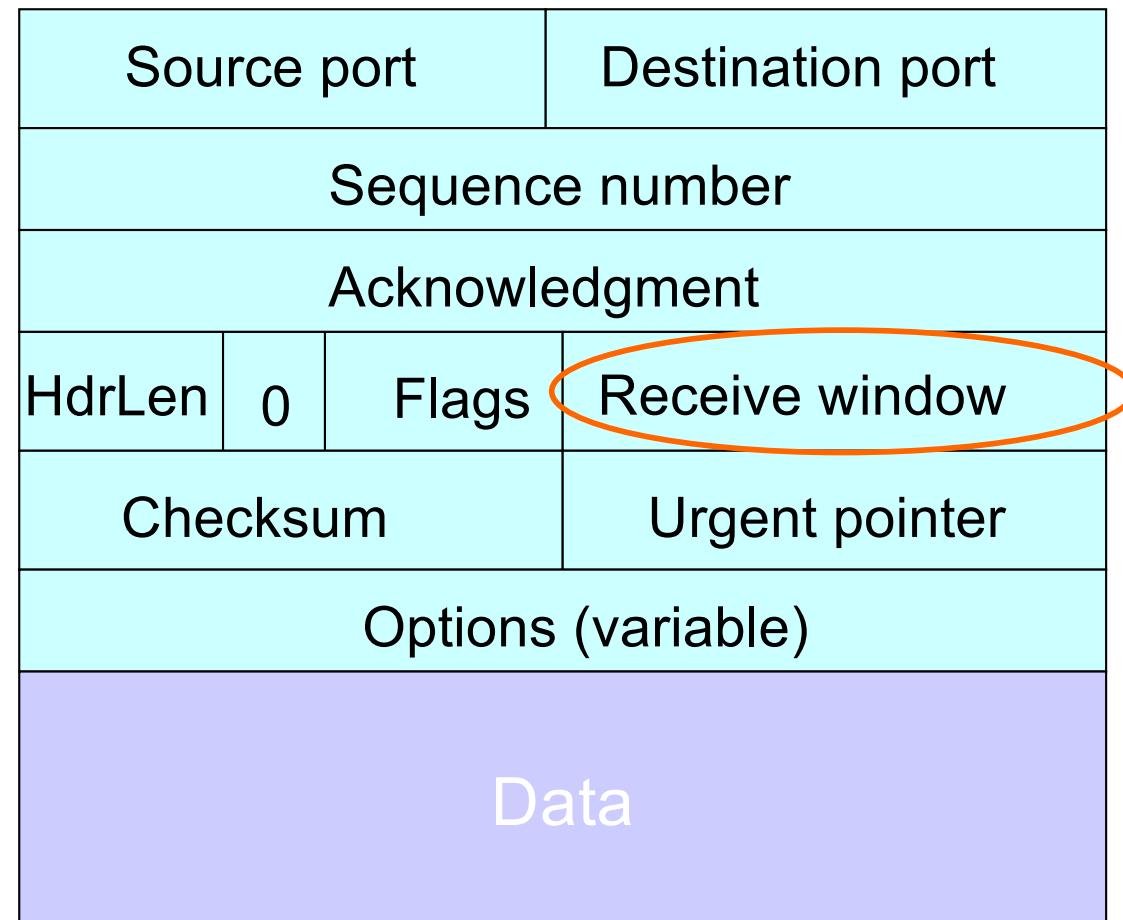
# TCP flow control

- ❖ 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



*receiver-side buffering*

# TCP Header



# 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

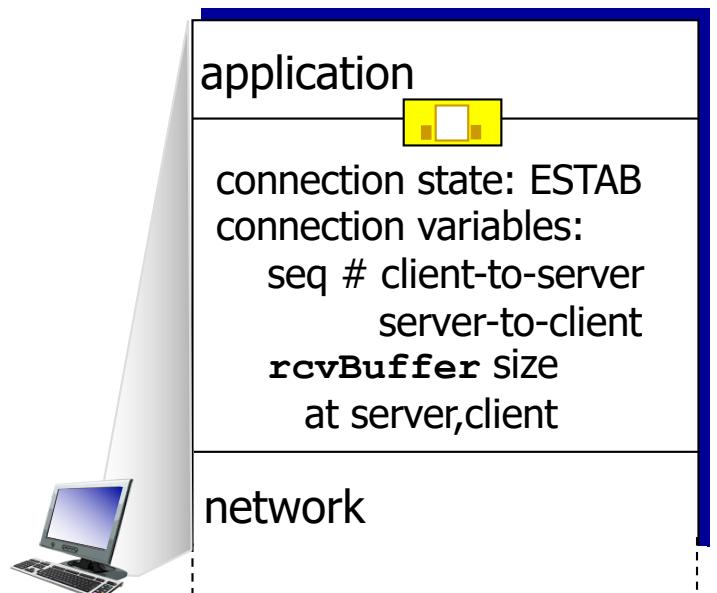
3.6 principles of congestion control

3.7 TCP congestion control

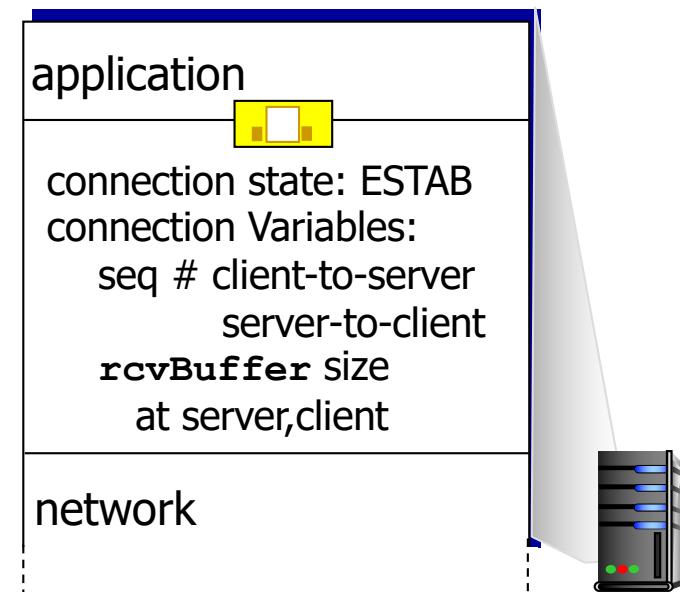
# Connection Management

before exchanging data, sender/receiver “handshake”:

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

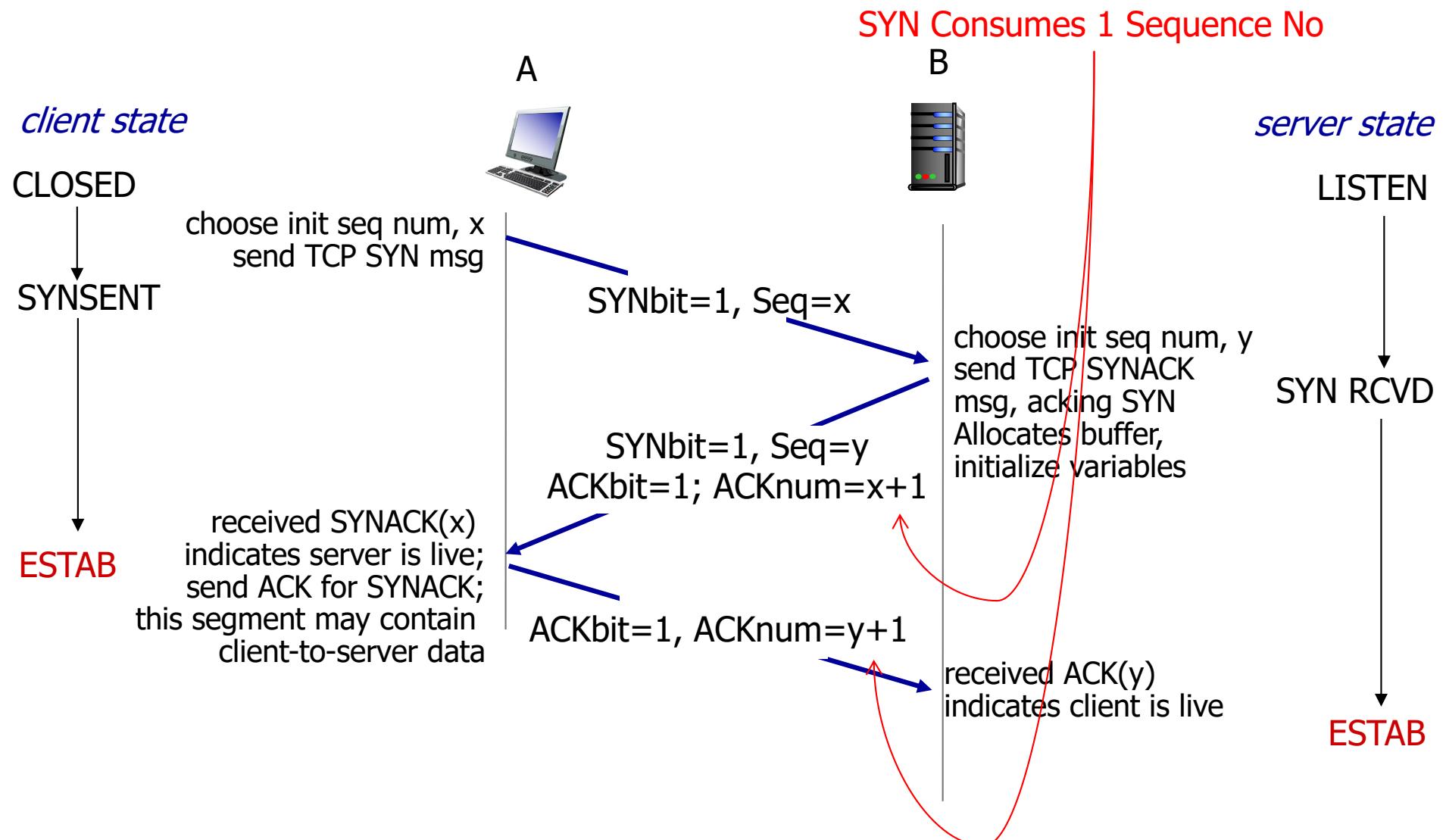


```
Socket clientSocket =  
    newSocket("hostname", "port  
    number");
```

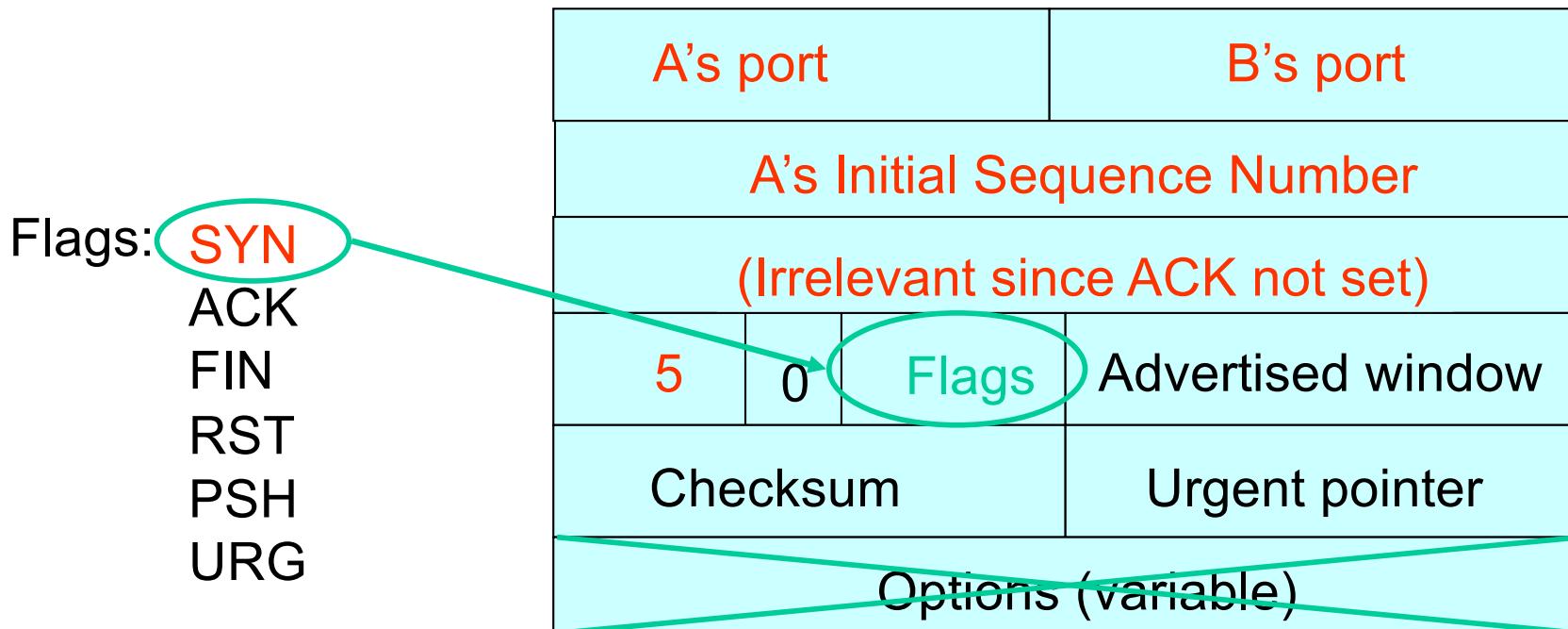


```
Socket connectionSocket =  
    welcomeSocket.accept();
```

# TCP 3-way handshake

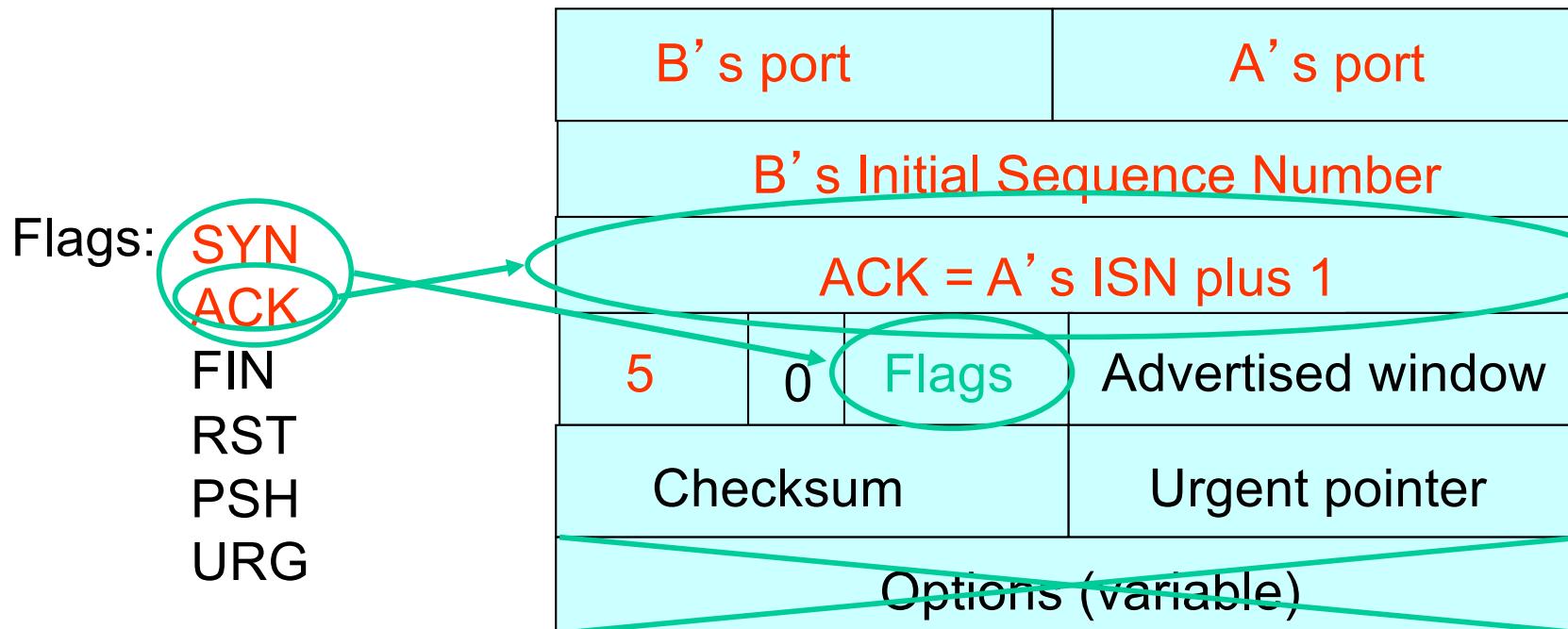


# Step 1: A's Initial SYN Packet



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

# Step 2: B's SYN-ACK Packet

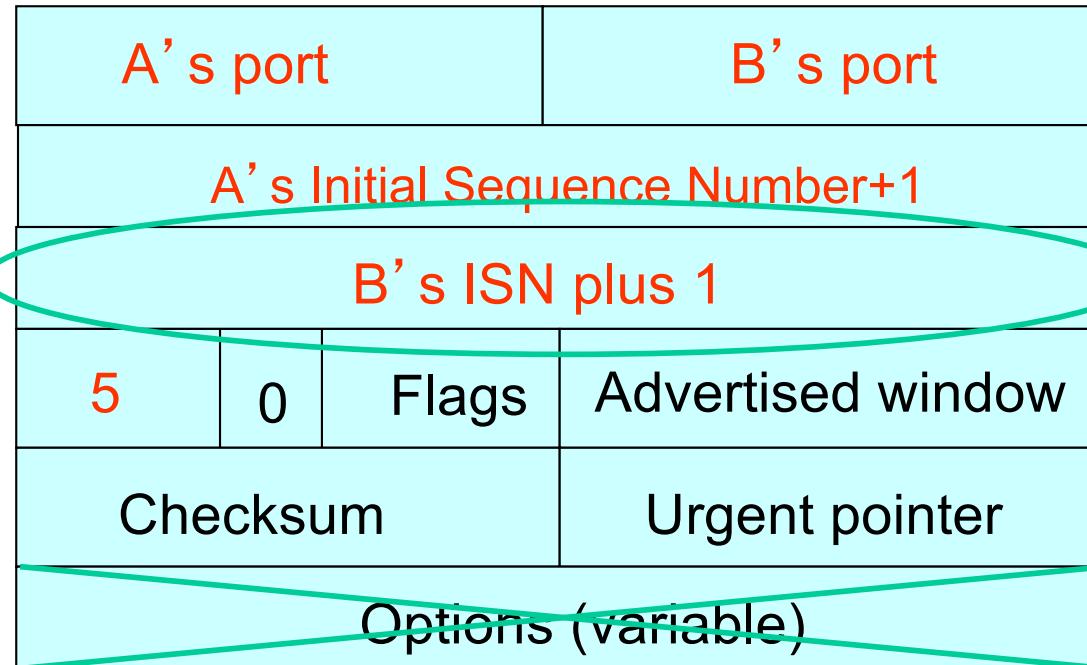


**B tells A it accepts, and is ready to hear the next byte...**

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

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

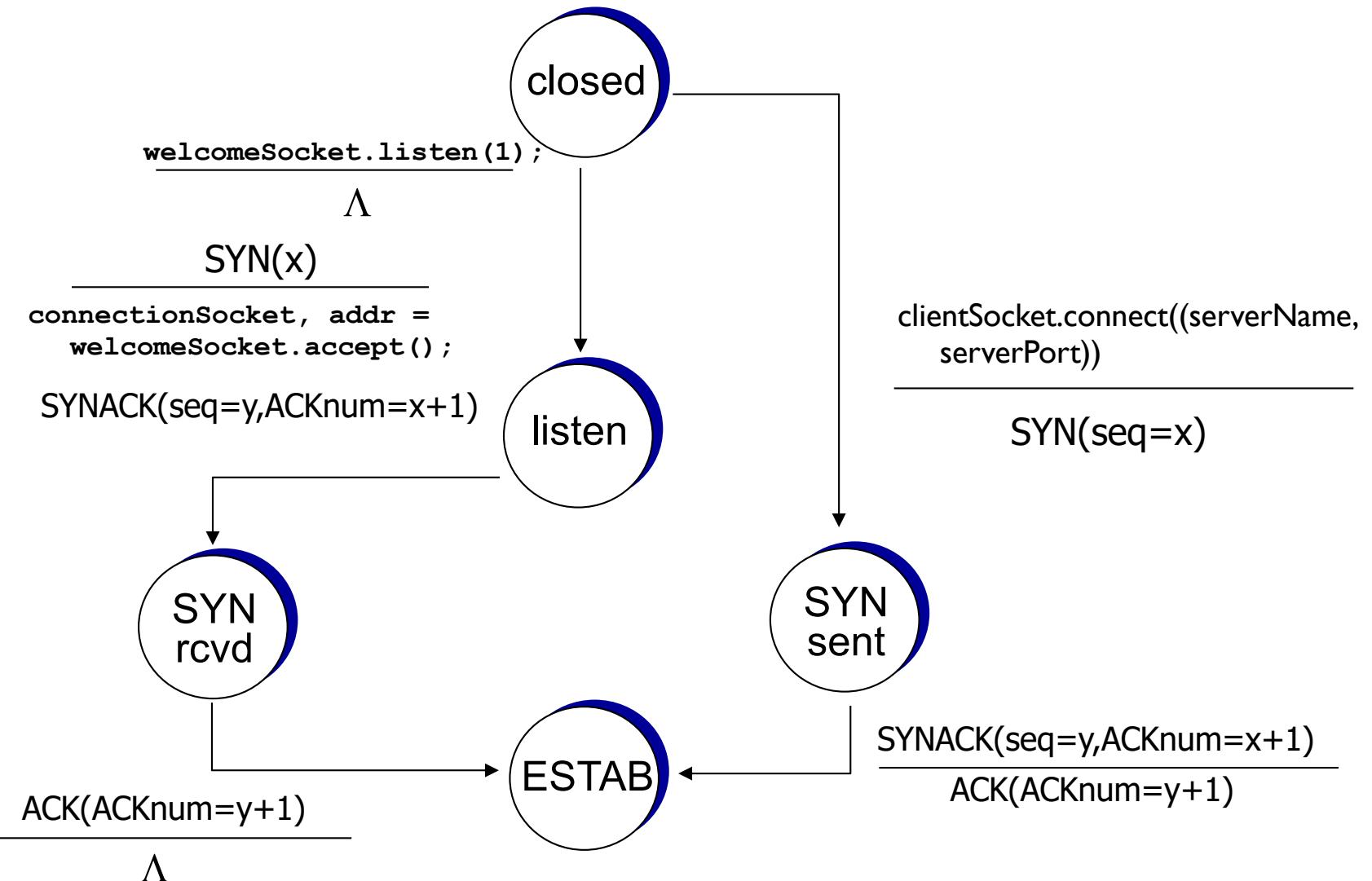
Flags: **SYN**  
**ACK**  
FIN  
RST  
PSH  
URG



**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**

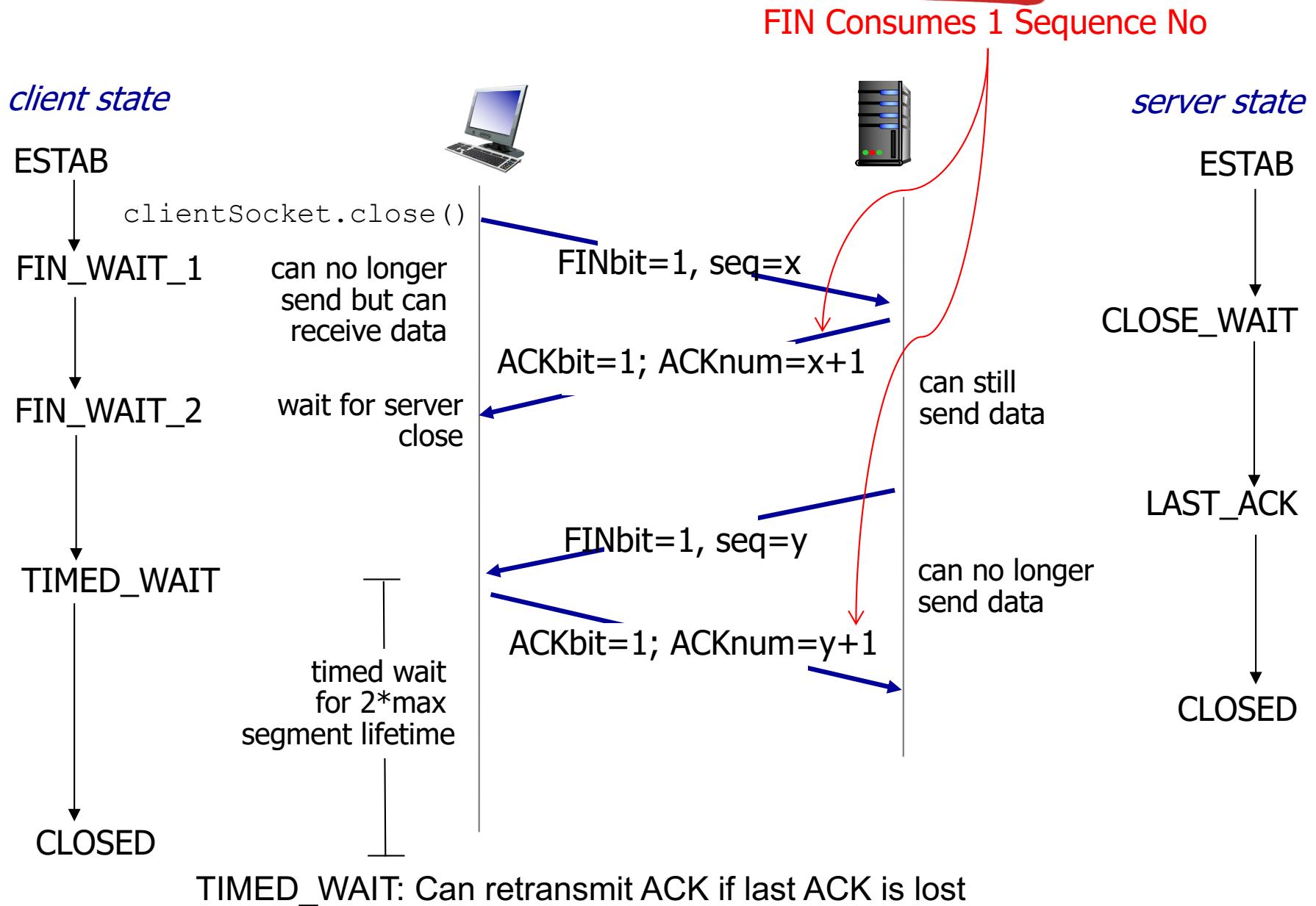
# SYN Loss and Web Downloads

- ❖ 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

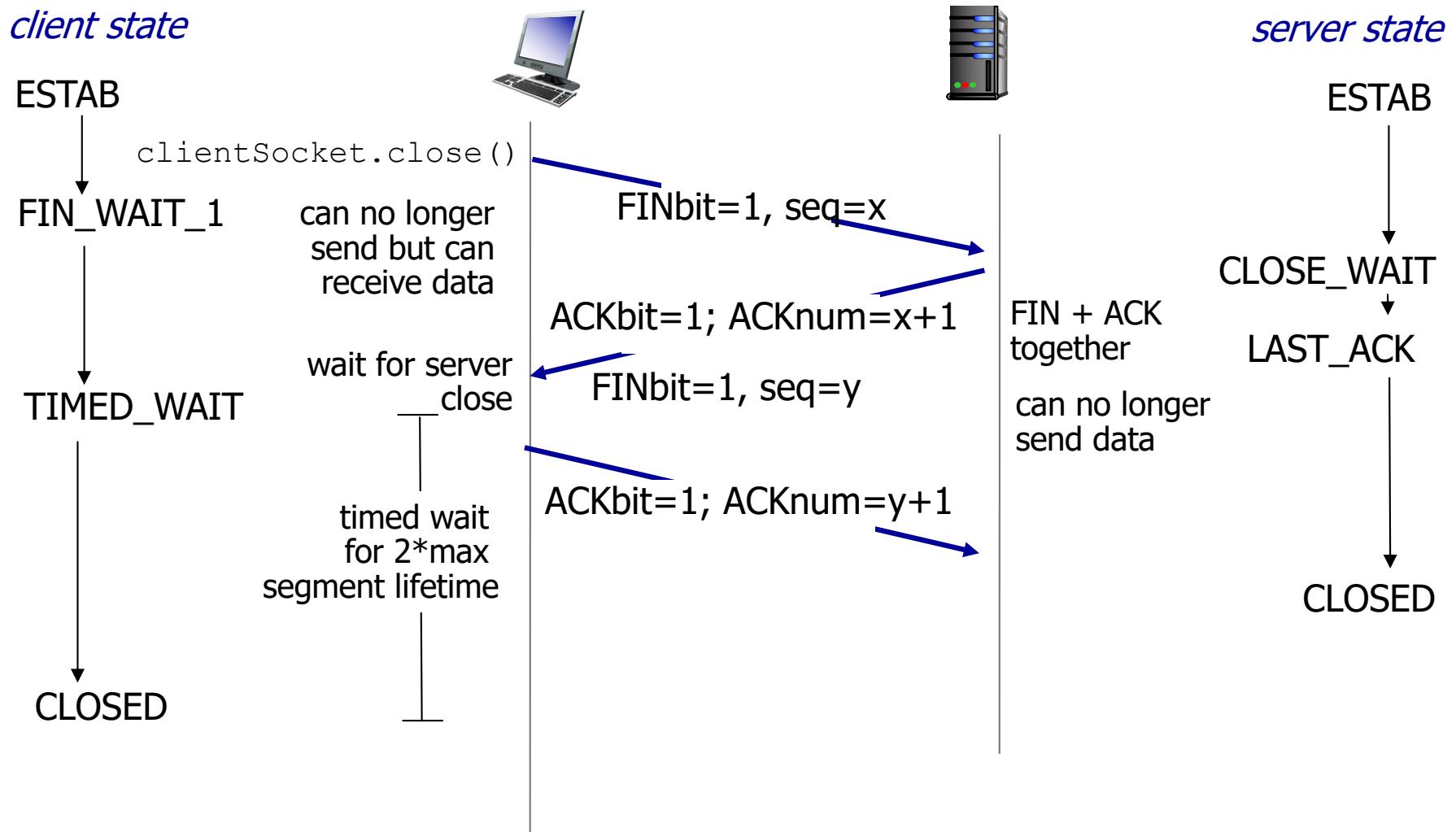
# TCP: closing a connection

- ❖ 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

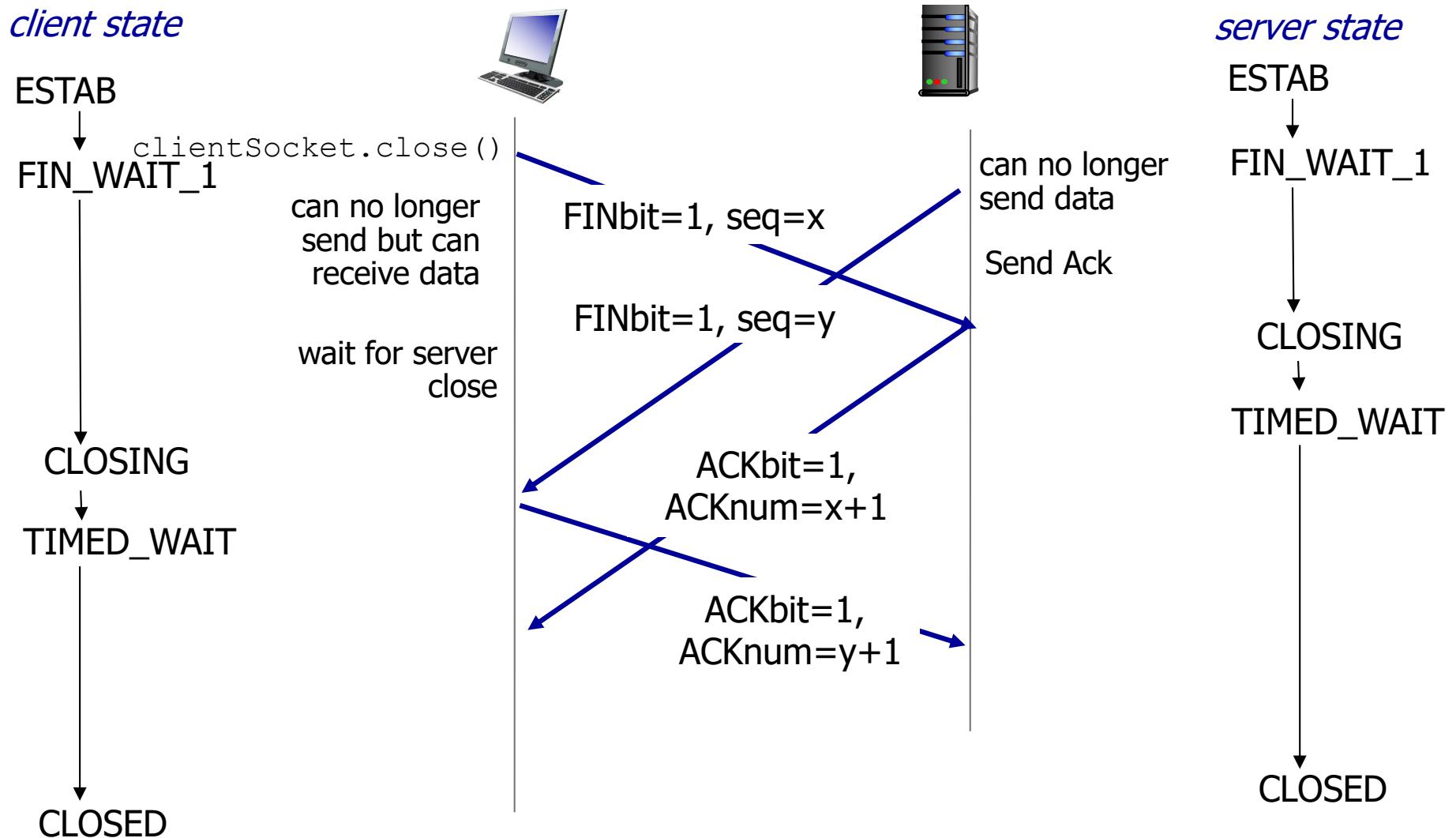
# Normal Termination, One at a Time



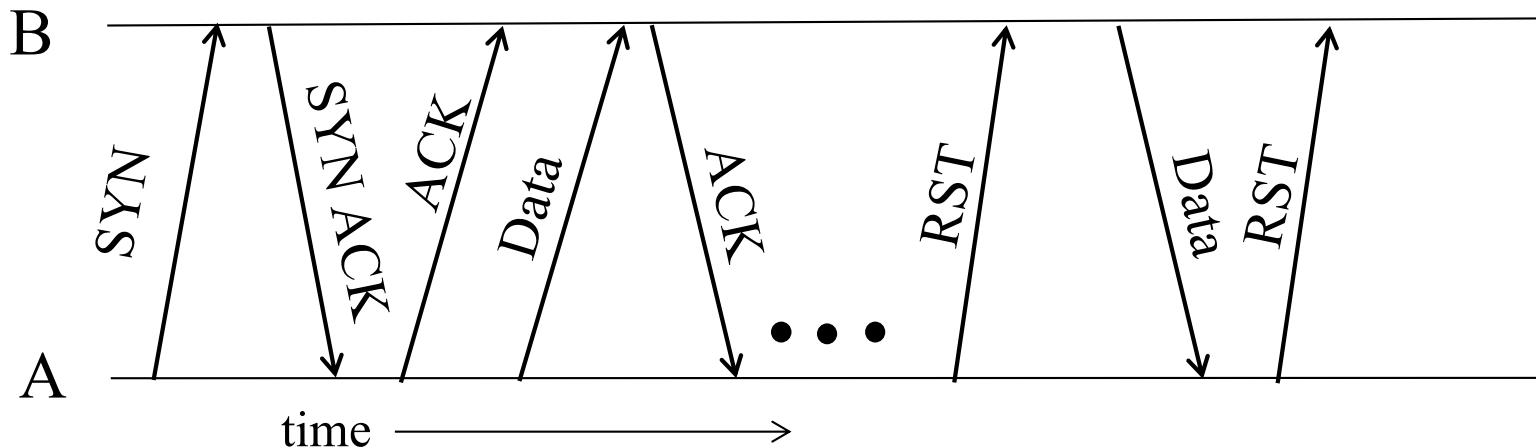
# Normal Termination, Both Together



# Simultaneous Closure

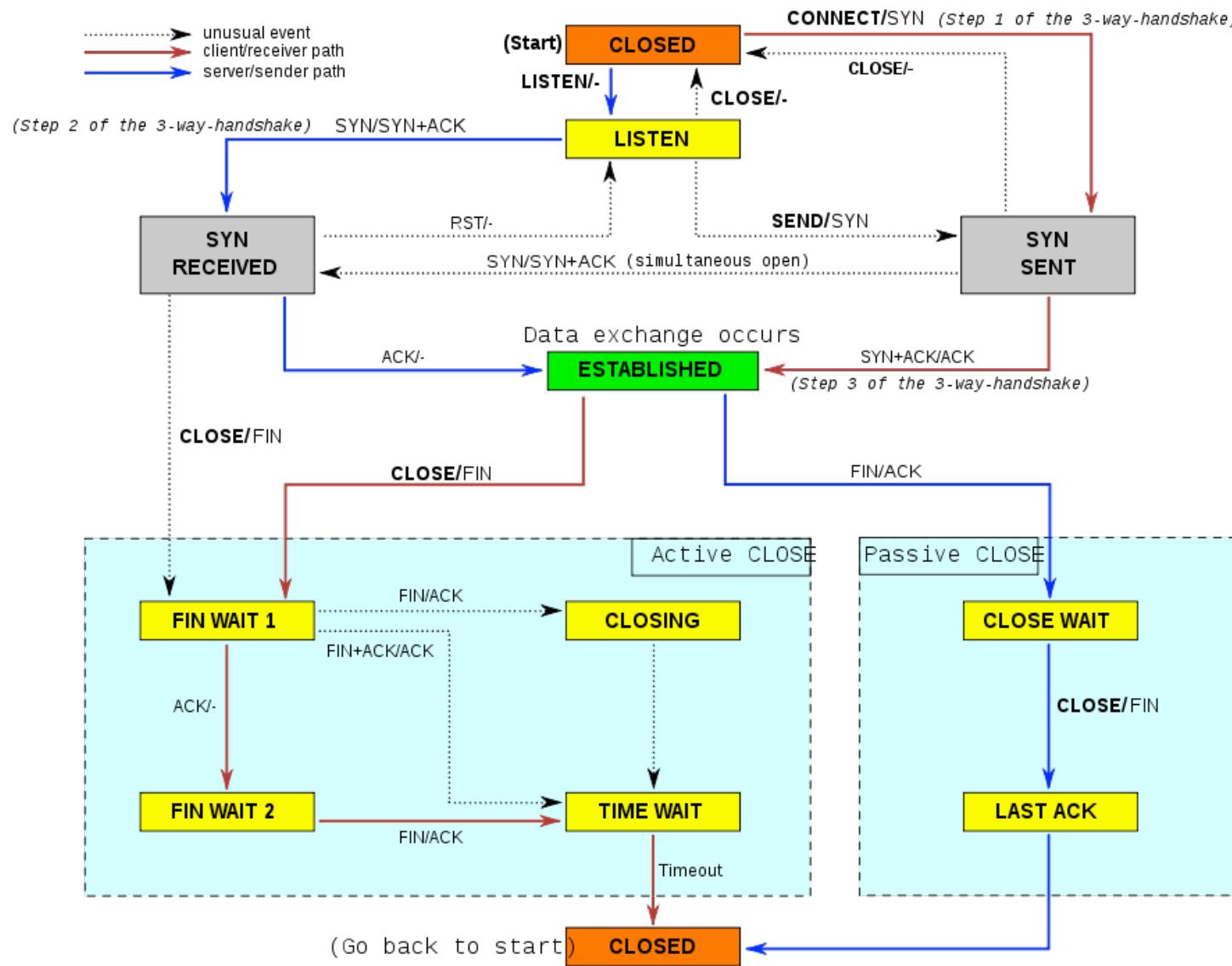


# Abrupt Termination



- ❖ 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

# Quiz: TCP Reliability?



TCP uses cumulative ACKs like Go-Back-N but does not retransmit the entire window of outstanding packets upon a timeout. What mechanism lets TCP get away with this?

- A. Per-byte sequence and acknowledgement numbers
- B. Triple Duplicate ACKs
- C. Receiver window-based flow control
- D. Timeout estimation algorithm

# 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

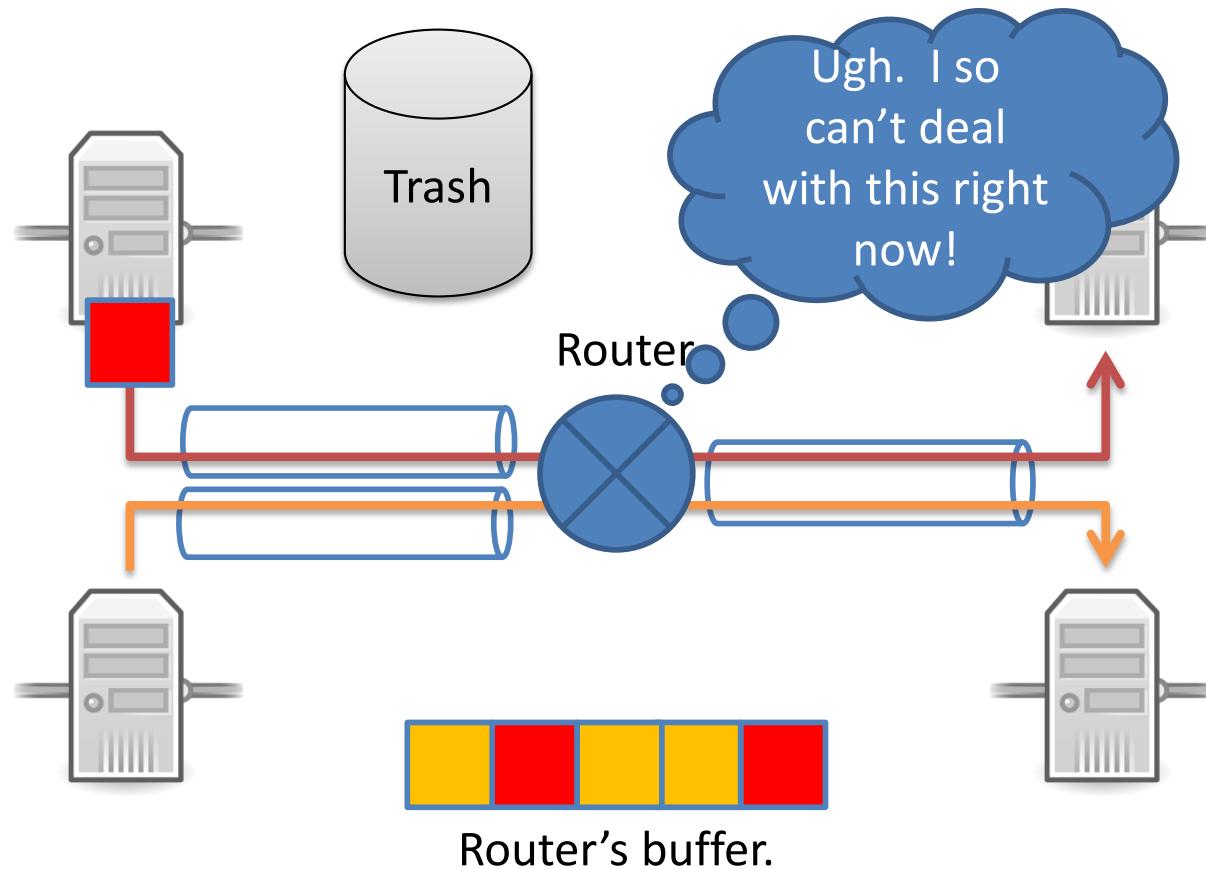
3.7 TCP congestion control

# 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



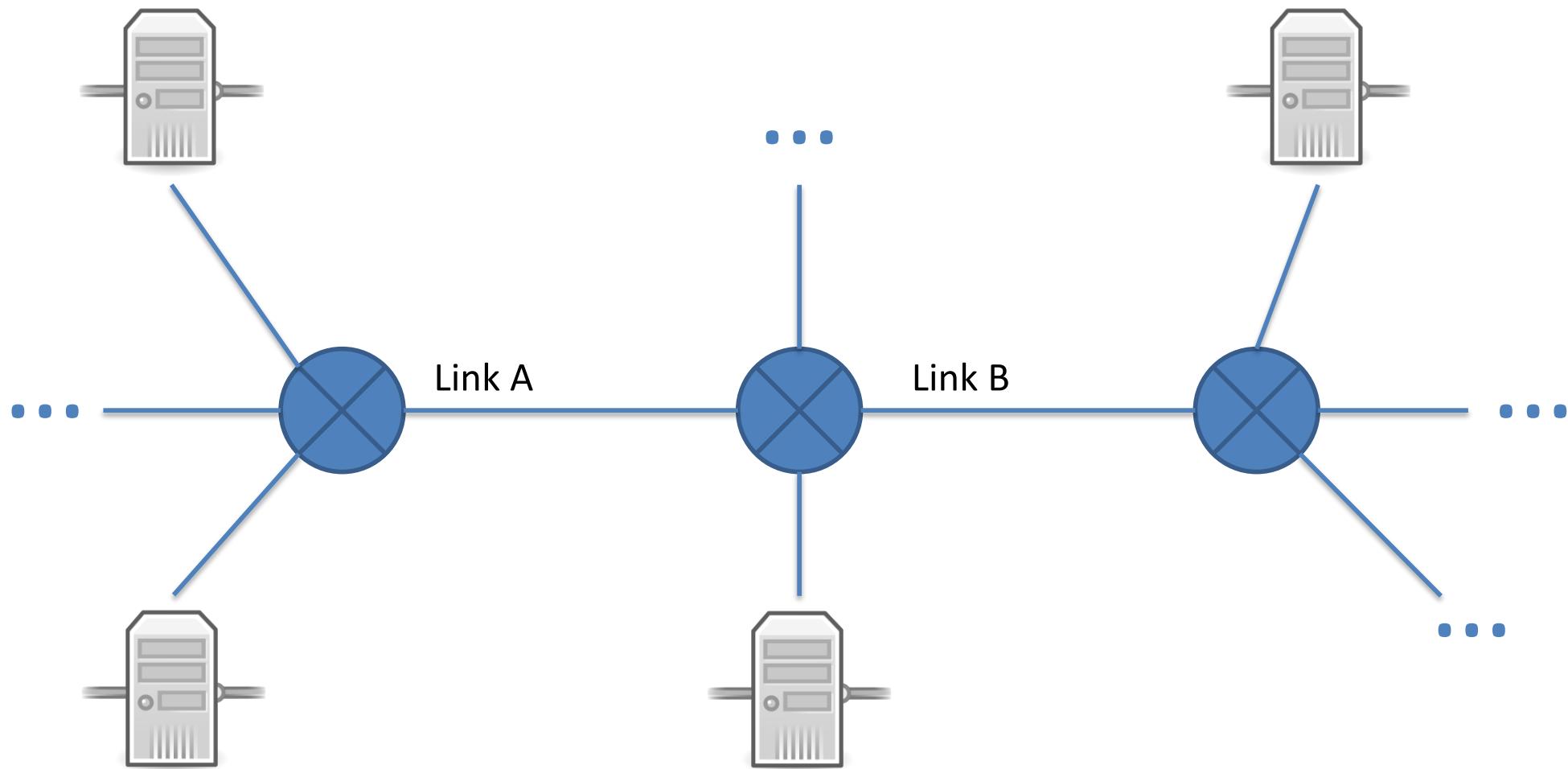
Incoming rate is faster than outgoing link can support.

# Quiz: What's the worst that can happen?

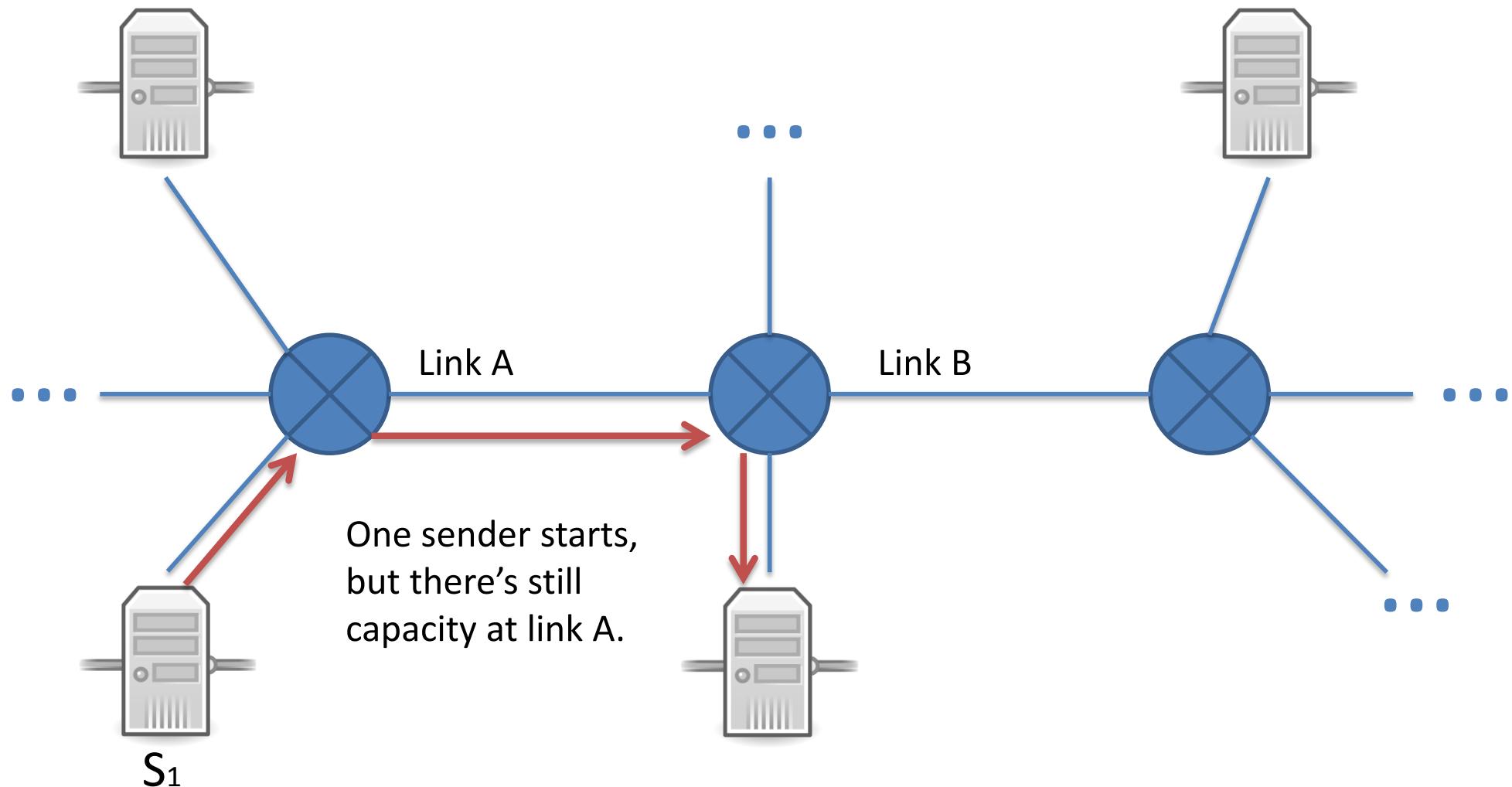


- A: This is no problem. Senders just keep transmitting, and it'll all work out.
- B: There will be retransmissions, but the network will still perform without much trouble.
- C: Retransmissions will become very frequent, causing a serious loss of efficiency
- D: The network will become completely unusable

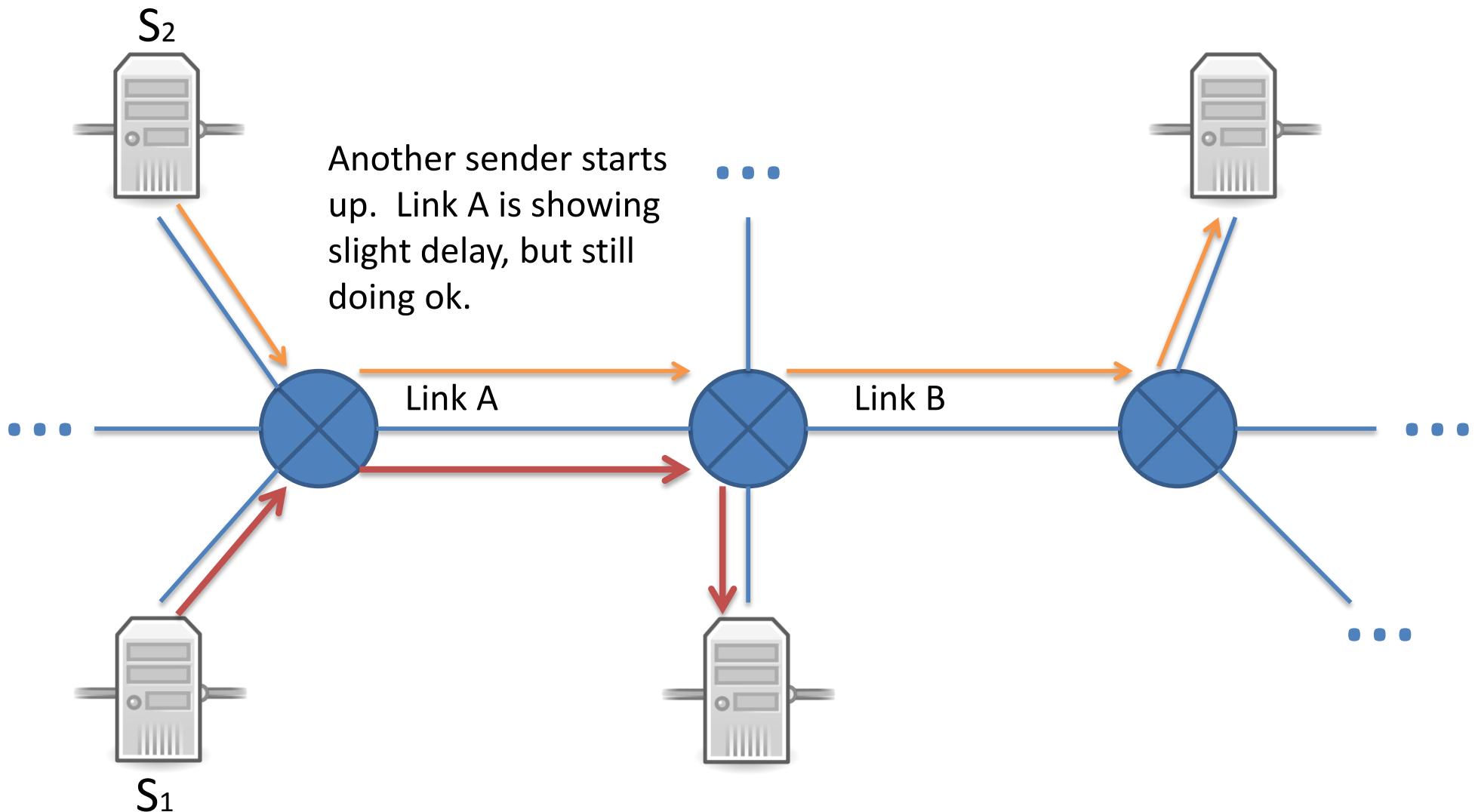
# Congestion Collapse



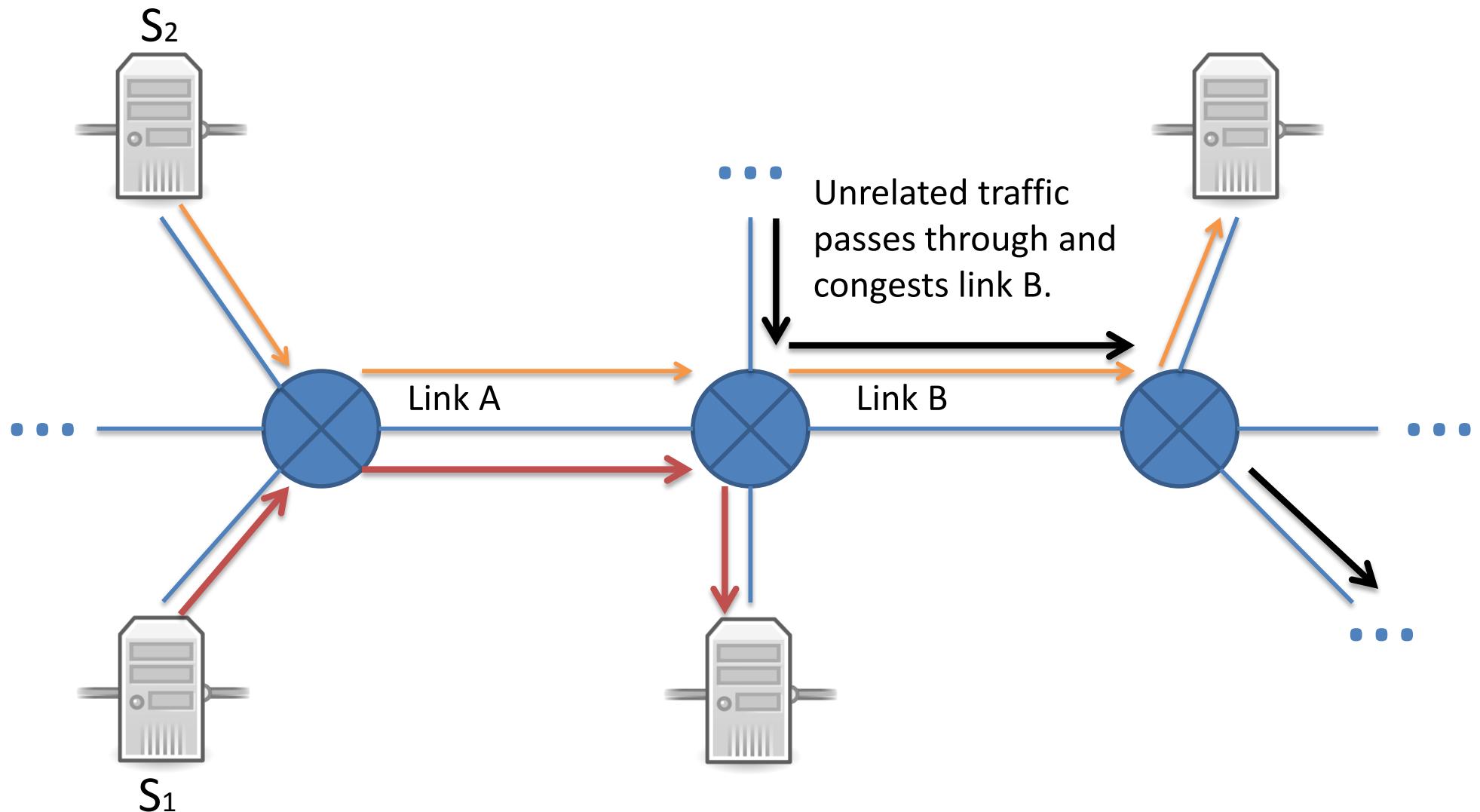
# Congestion Collapse



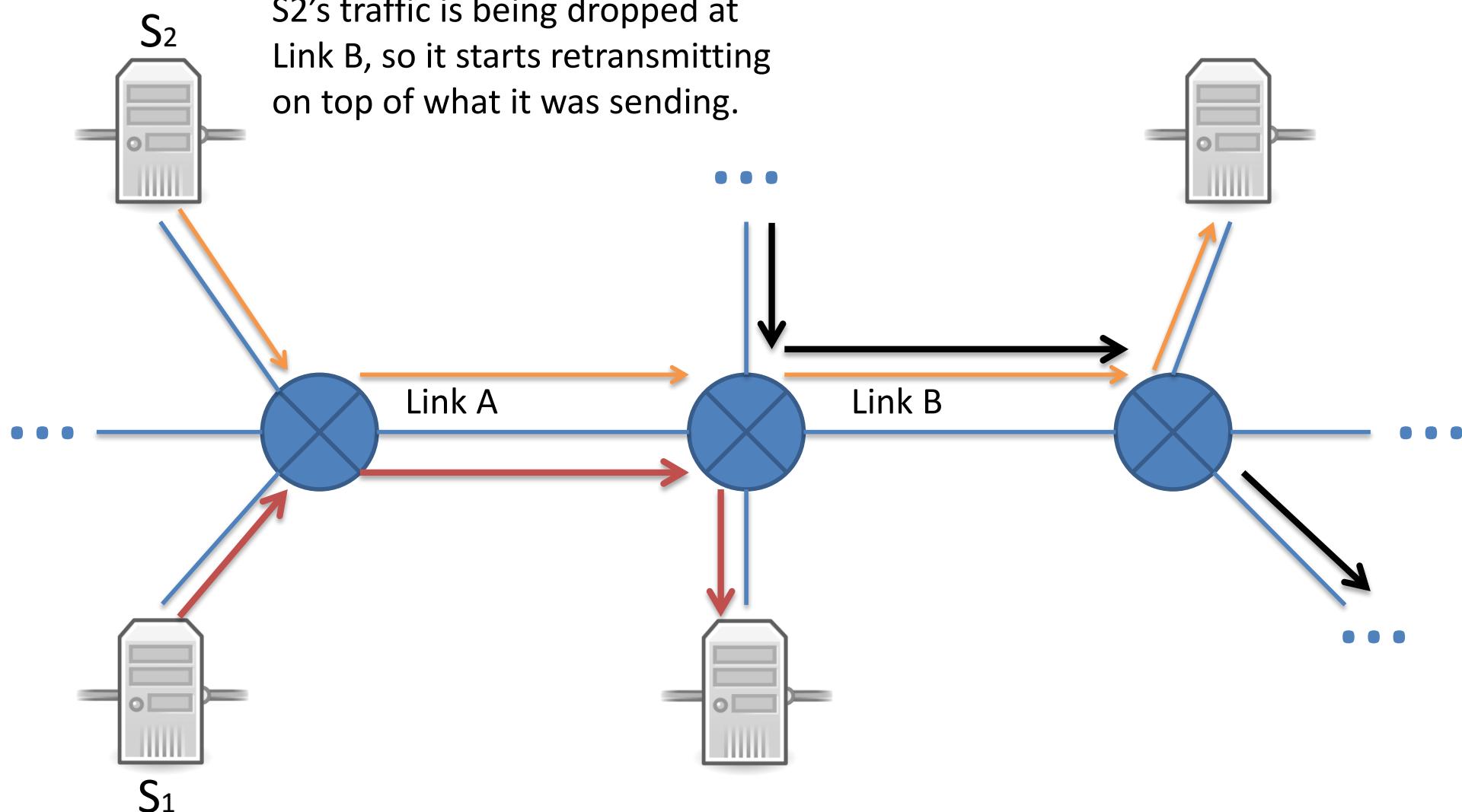
# Congestion Collapse



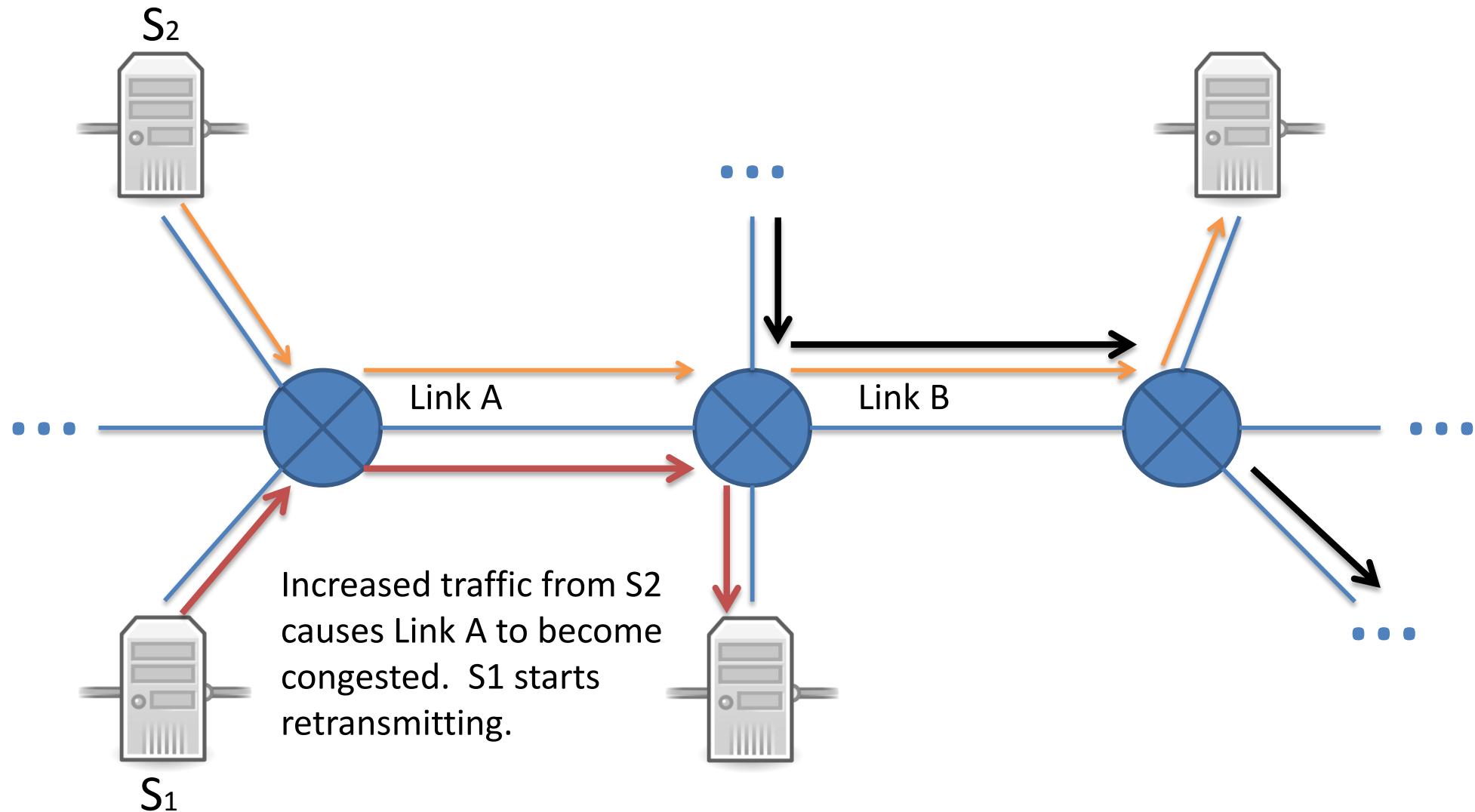
# Congestion Collapse



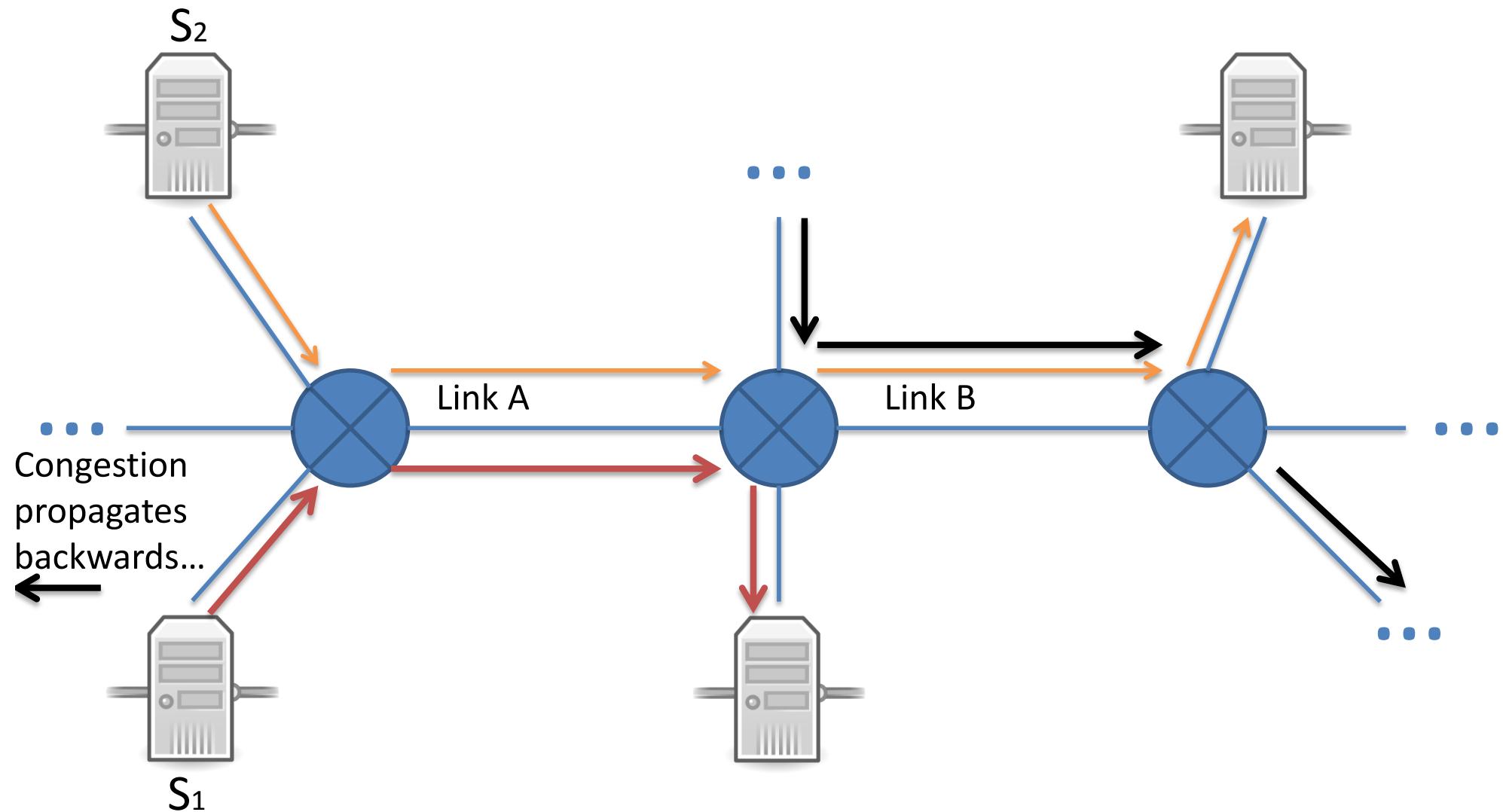
# Congestion Collapse



# Congestion Collapse



# Congestion Collapse



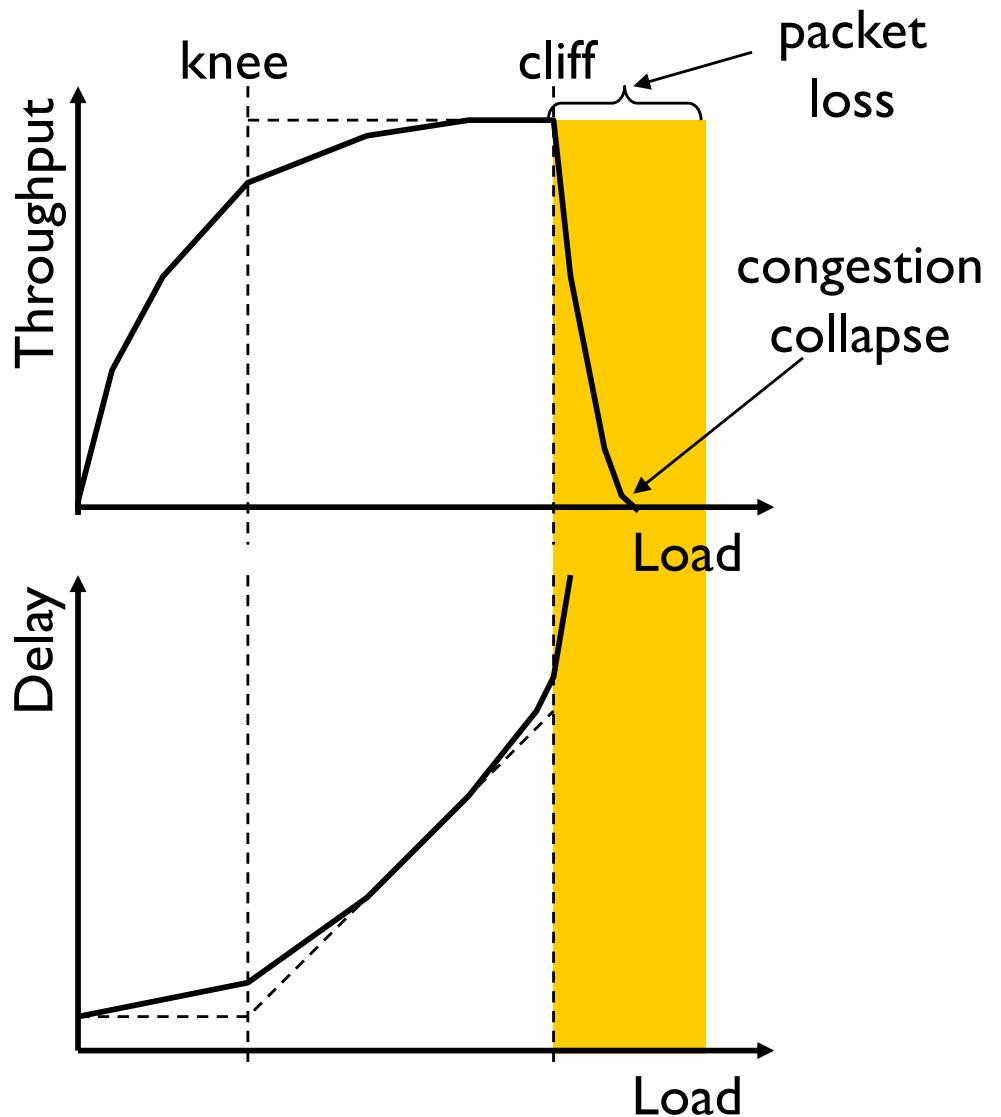
# Without congestion control

*congestion:*

- ❖ Increases delivery latency
  - Variable 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

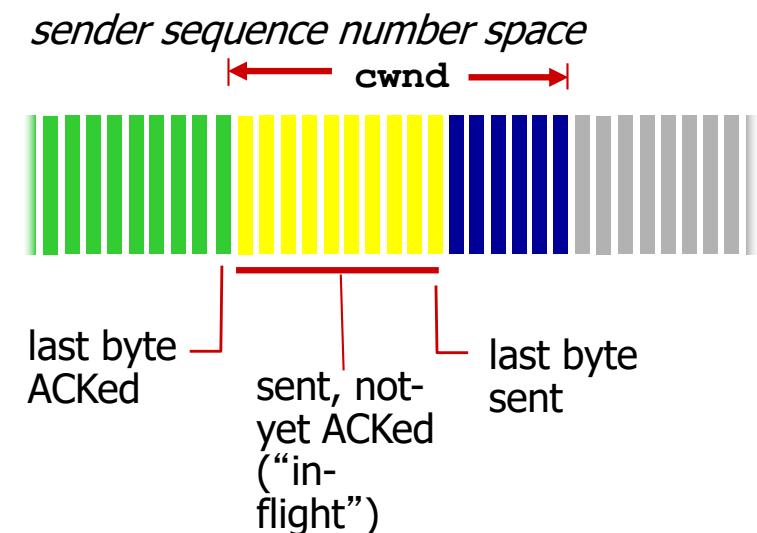
3.6 principles of congestion control

3.7 TCP congestion control

# TCP's Approach in a Nutshell

- ❖ TCP connection has 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 lecture that RWND >> CWN

# 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?

# Quiz: What is a “congestion event”



- A: A segment loss (but how can the sender be sure of this?)
- B: Increased delays
- C: Receiving duplicate acknowledgement(s)
- D: A retransmission timeout firing
- E: Some subset of A, B, C & D (what is the subset?)

# Quiz: How should we set CWND?



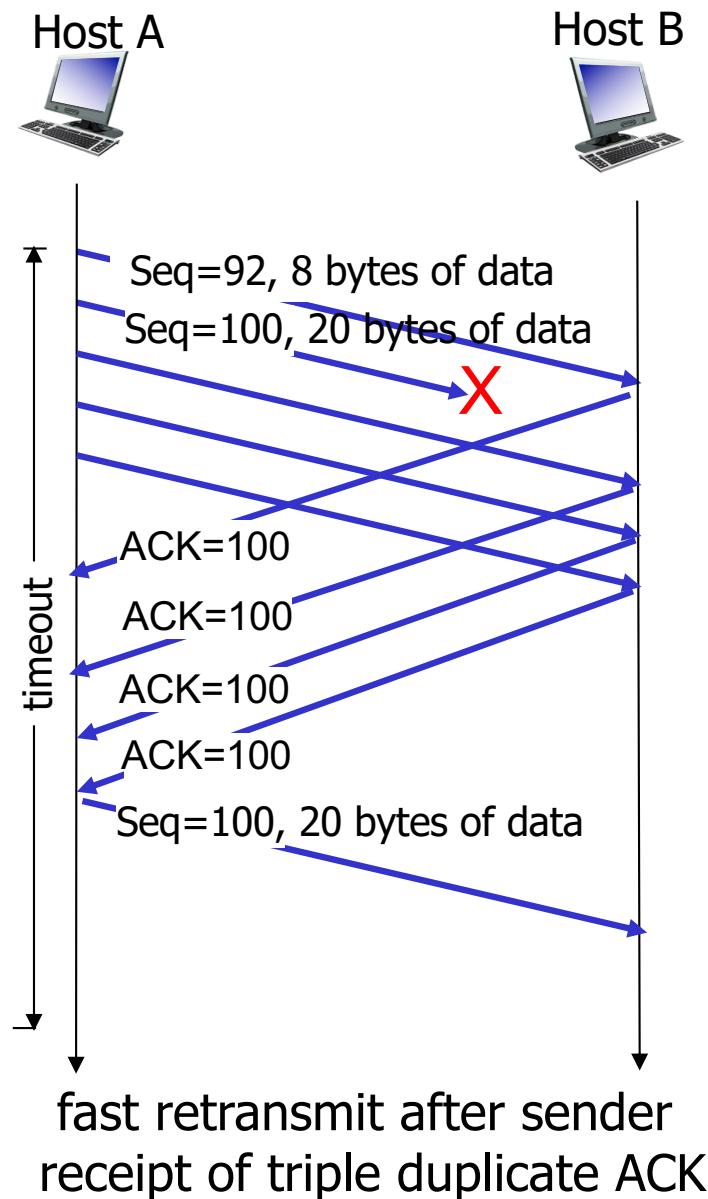
- A: We should keep raising it until a “congestion event” then back off slightly until we notice no more events
- B: We should raise it until a “congestion event”, then go back to 1 and start raising it again
- C: We should raise it until a “congestion event”, then go back to median value and start raising it again
- D: We should sent as fast as possible at all times

# Not All Losses the Same

---

- ❖ 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



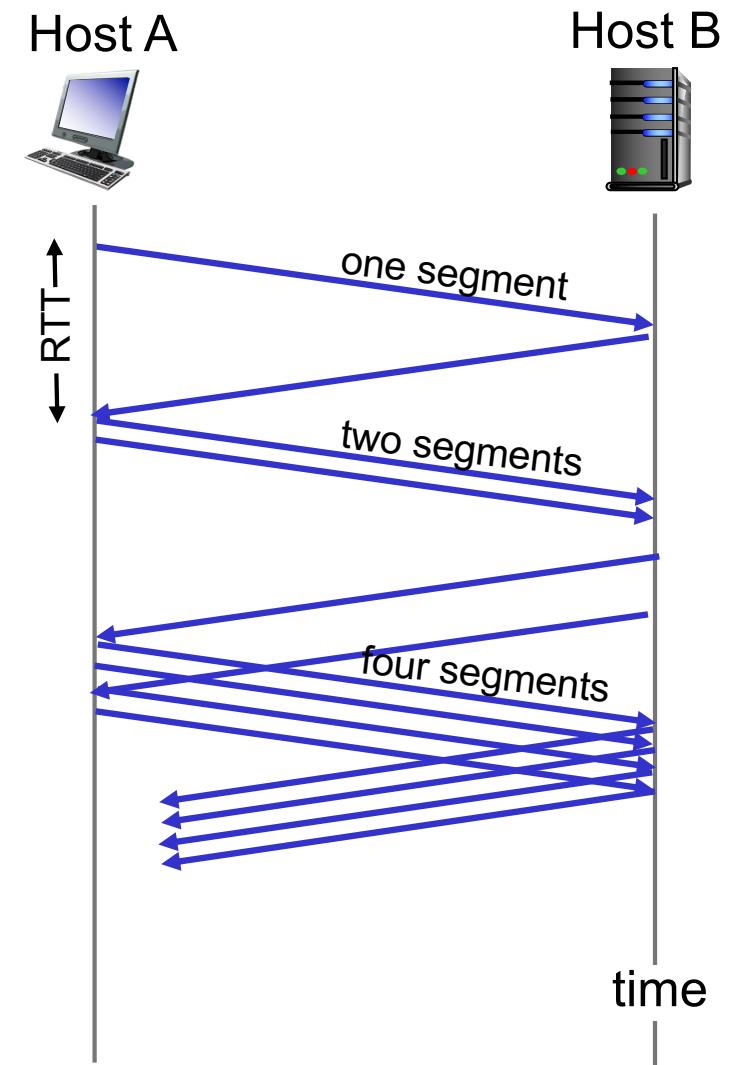
# Rate Adjustment

---

- ❖ 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



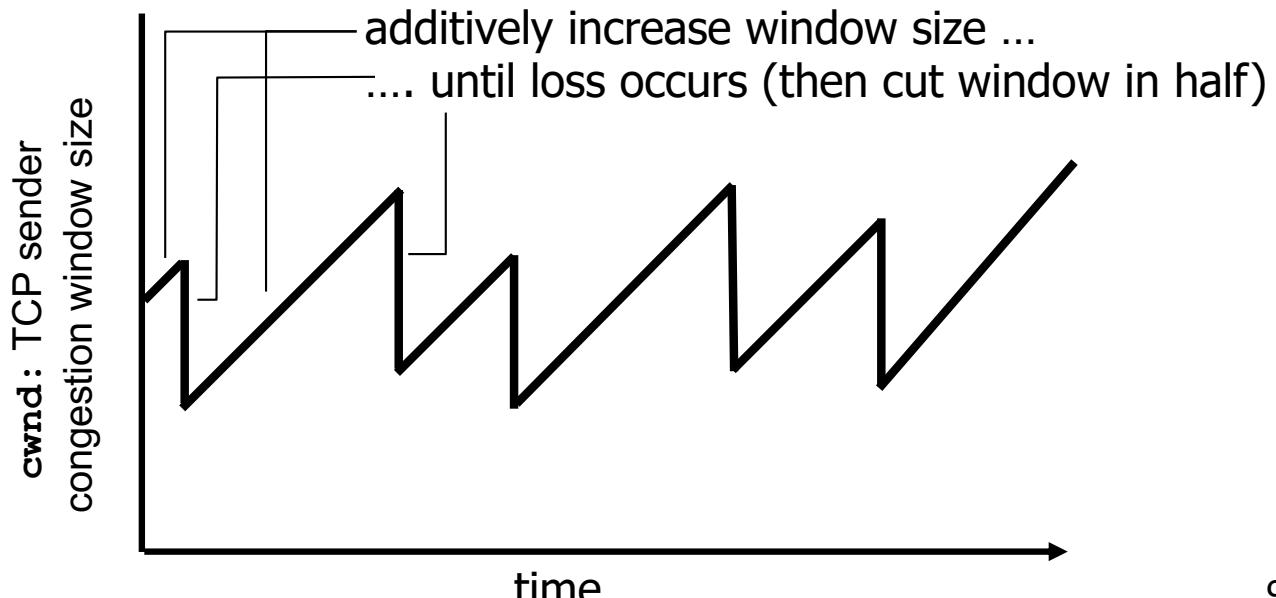
# Adjusting to Varying Bandwidth

- ❖ 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)
  - We’ll see why shortly...

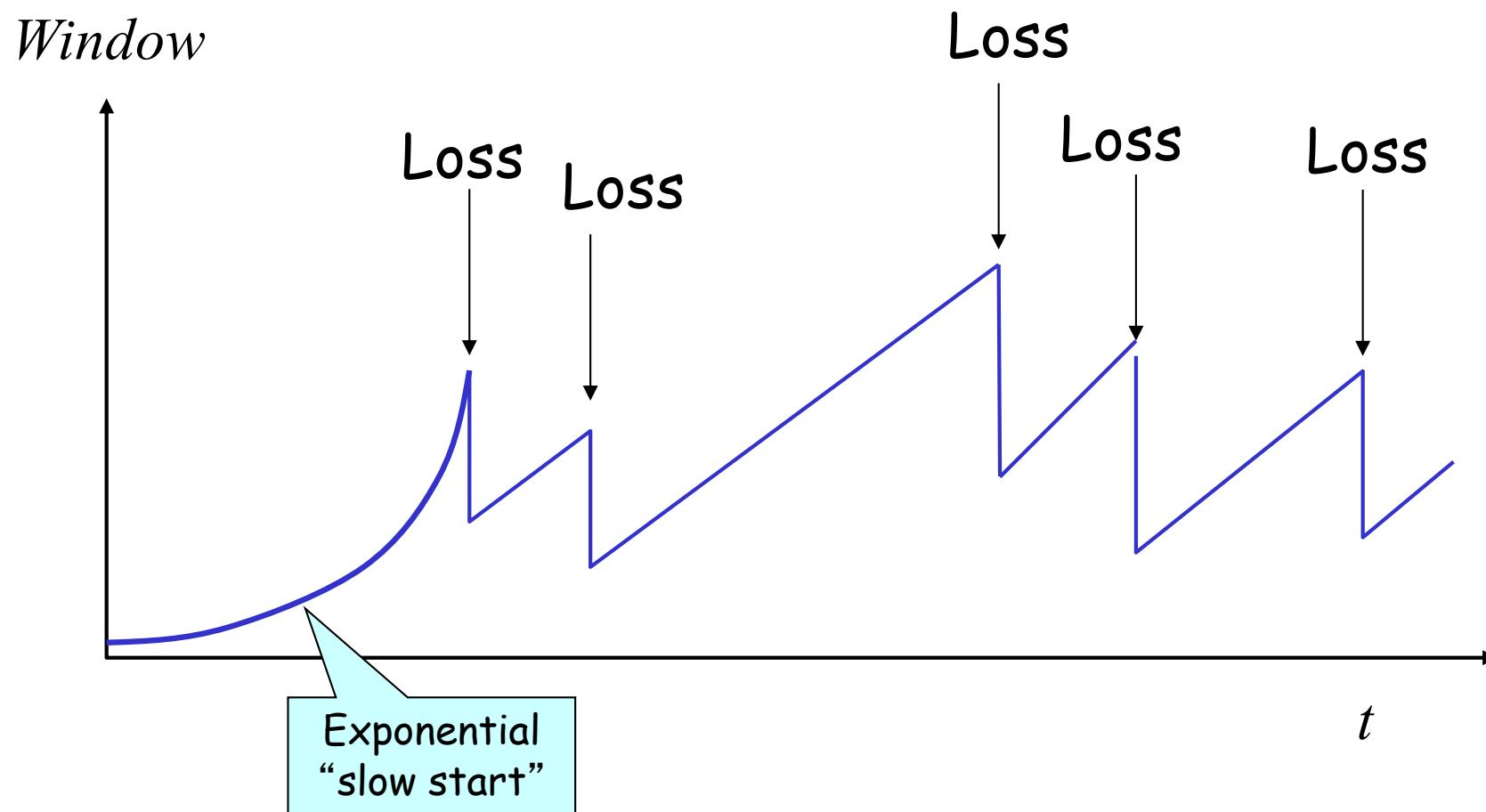
# AIMD

- ❖ **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,  $cwnd = cwnd + 1/cwnd$
  - **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)

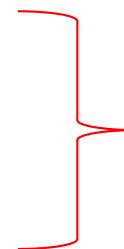
- ❖ If  $CWND < ssthresh$

- $CWND += +$

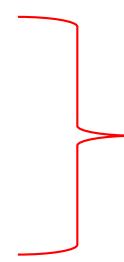
- $2 \times MSS$  packets per ACK
- Hence after one RTT (All ACKs with no drops):  
 $CWND = 2 \times CWND$

# 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

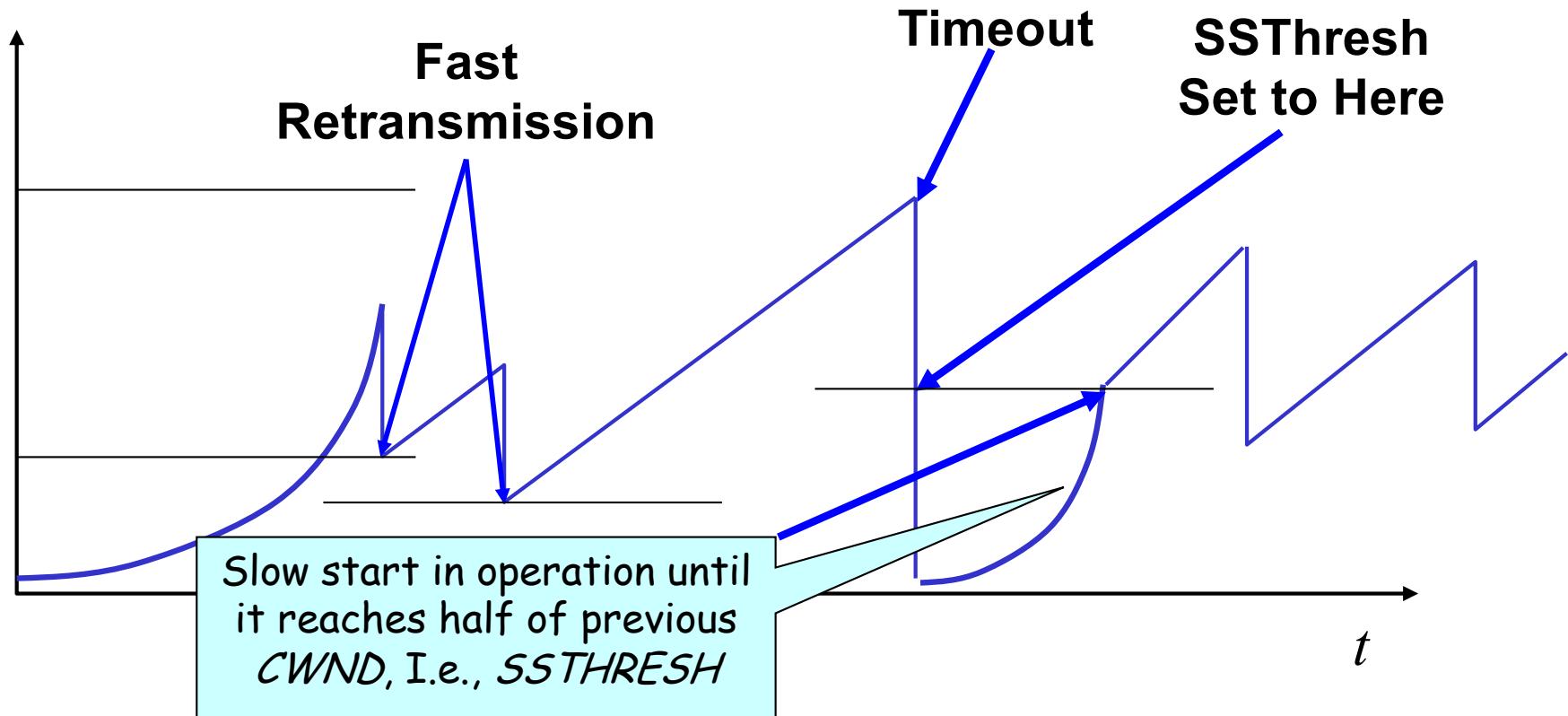
- ❖ dupACKcount ++
- ❖ If dupACKcount = 3 /\* fast retransmit \*/
  - ssthresh = CWND/2
  - **CWND = CWND/2**

# Event: TimeOut

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

# Example

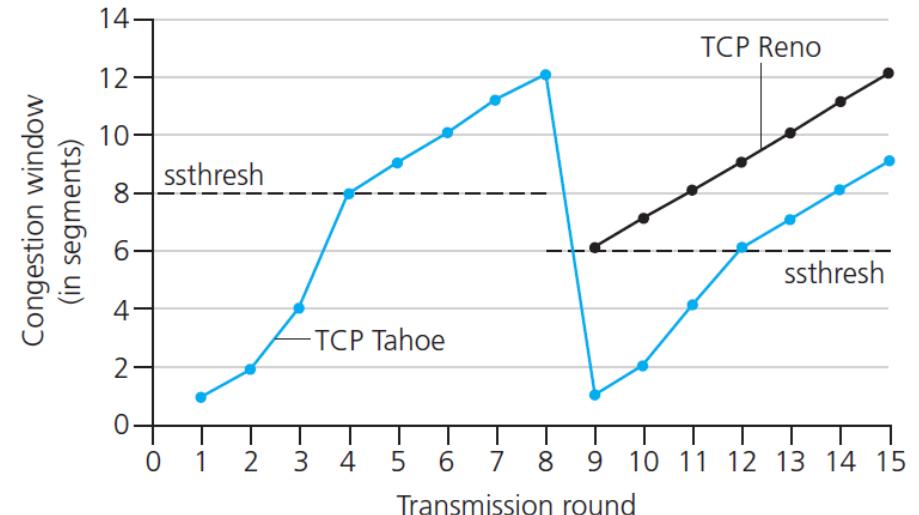
Window



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

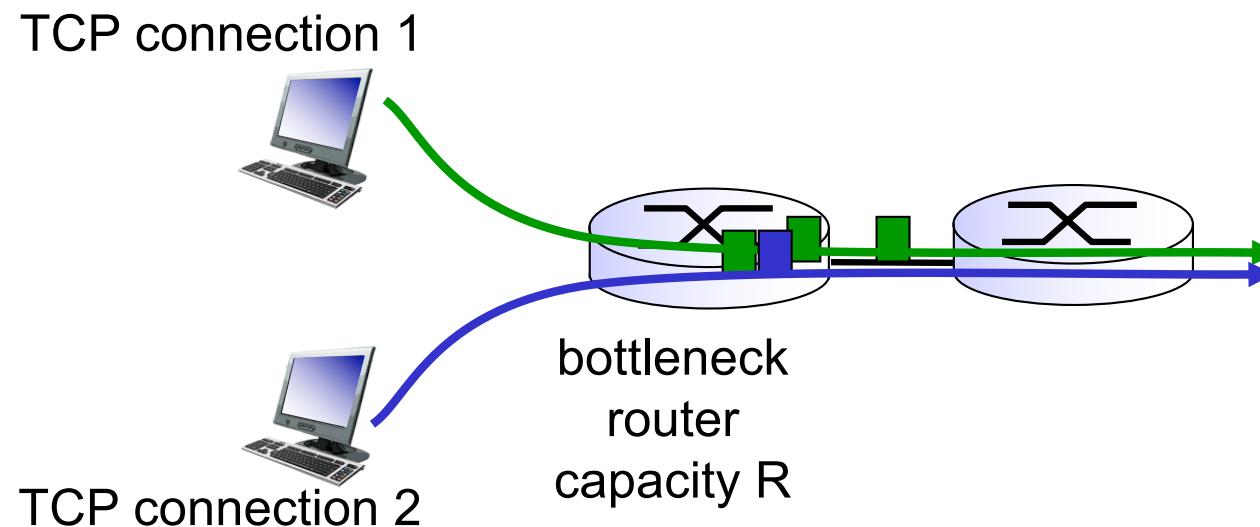
# TCP Flavours

- ❖ 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



# TCP Fairness

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



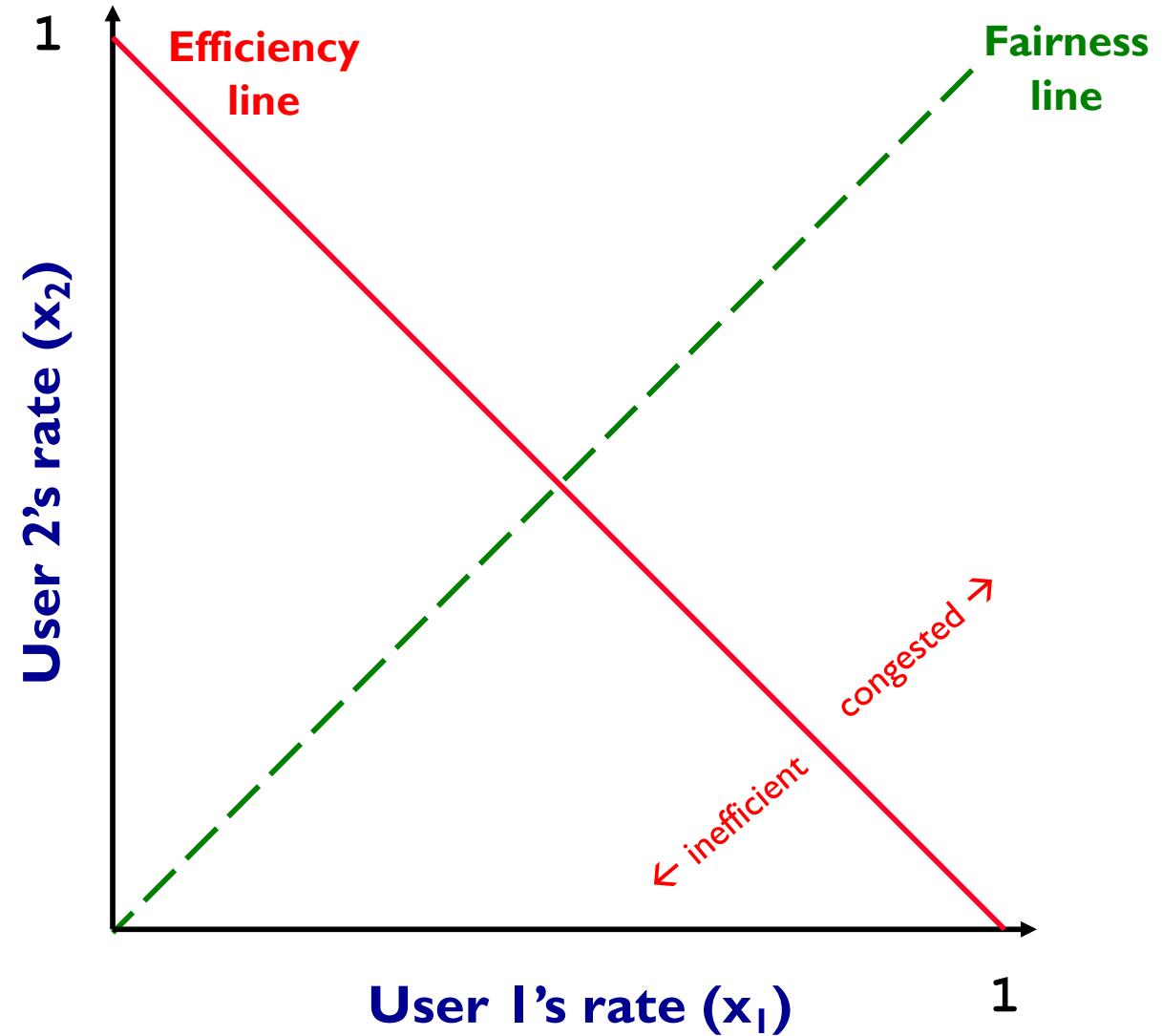
# Why AIMD?

- ❖ Some rate adjustment options: Every RTT, we can
  - Multiplicative increase or decrease:  $\text{WND} \rightarrow a * \text{WND}$
  - Additive increase or decrease:  $\text{WND} \rightarrow \text{WND} + b$
- ❖ Four alternatives:
  - AIAD: gentle increase, gentle decrease
  - AIMD: gentle increase, drastic decrease
  - MIAD: drastic increase, gentle decrease
  - MIMD: drastic increase and decrease

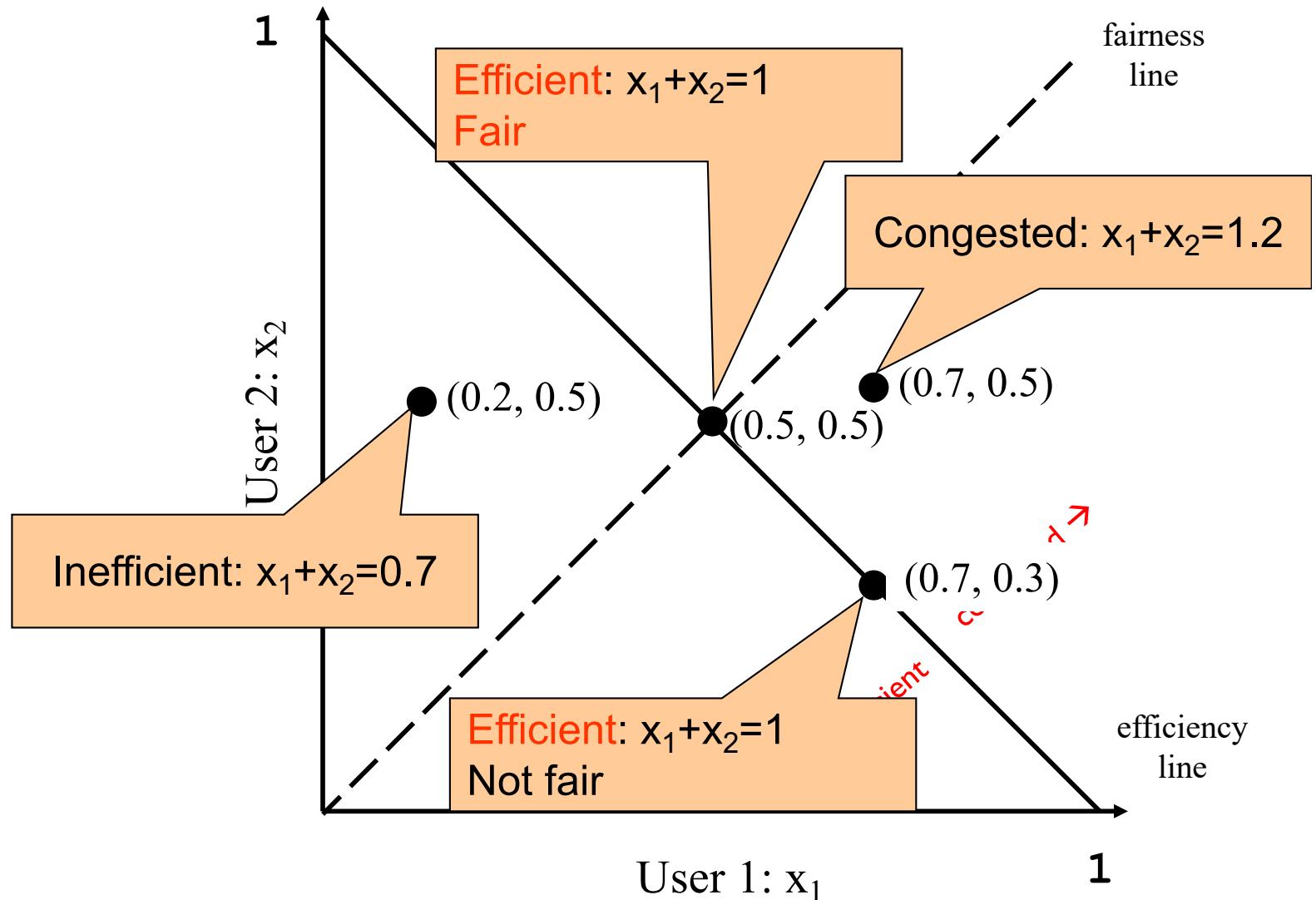
# Simple Model of Congestion Control

---

- ❖ Two users
  - rates  $x_1$  and  $x_2$
- ❖ Congestion when  $x_1+x_2 > 1$
- ❖ Unused capacity when  $x_1+x_2 < 1$
- ❖ Fair when  $x_1 = x_2$

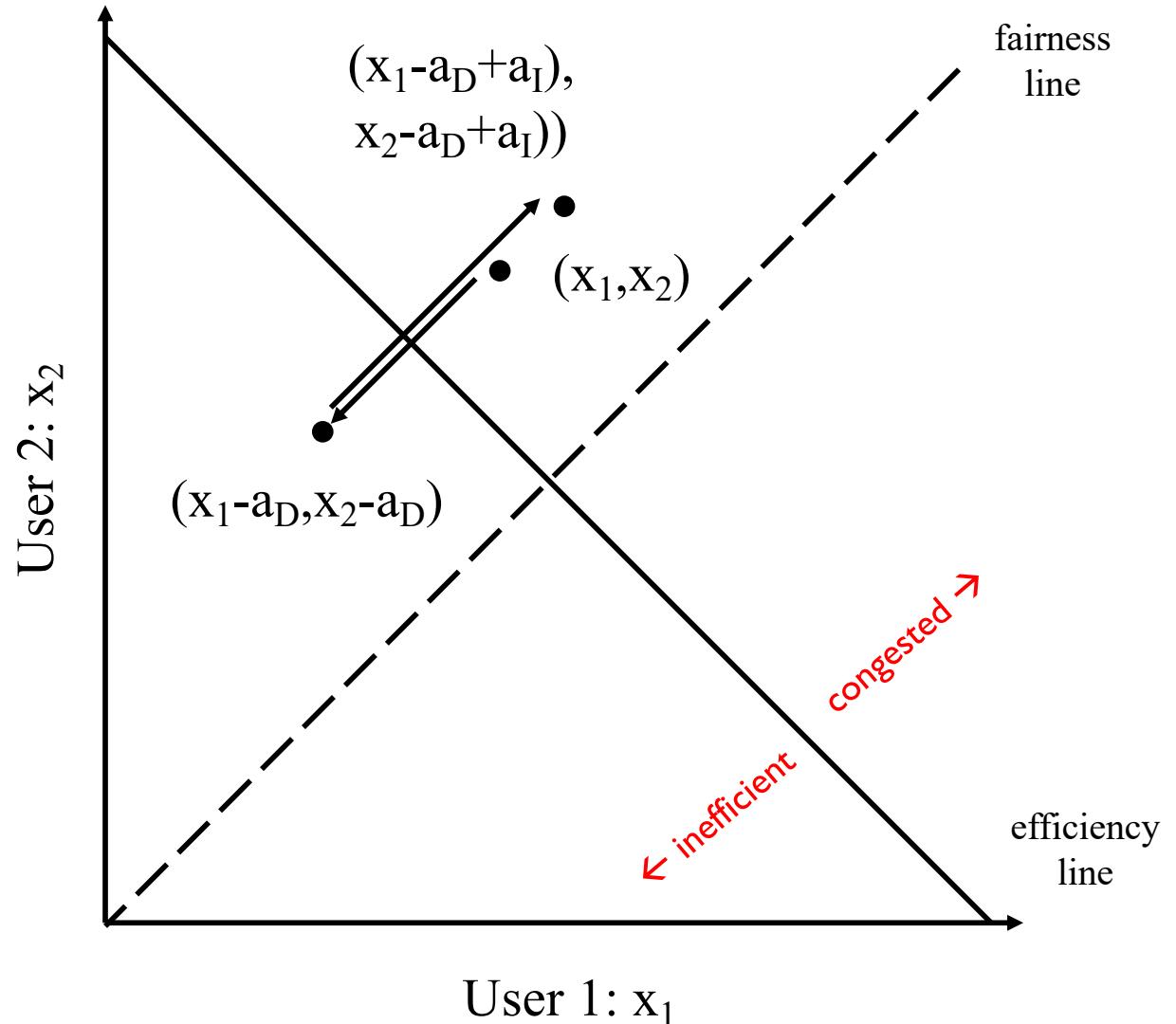


# Example



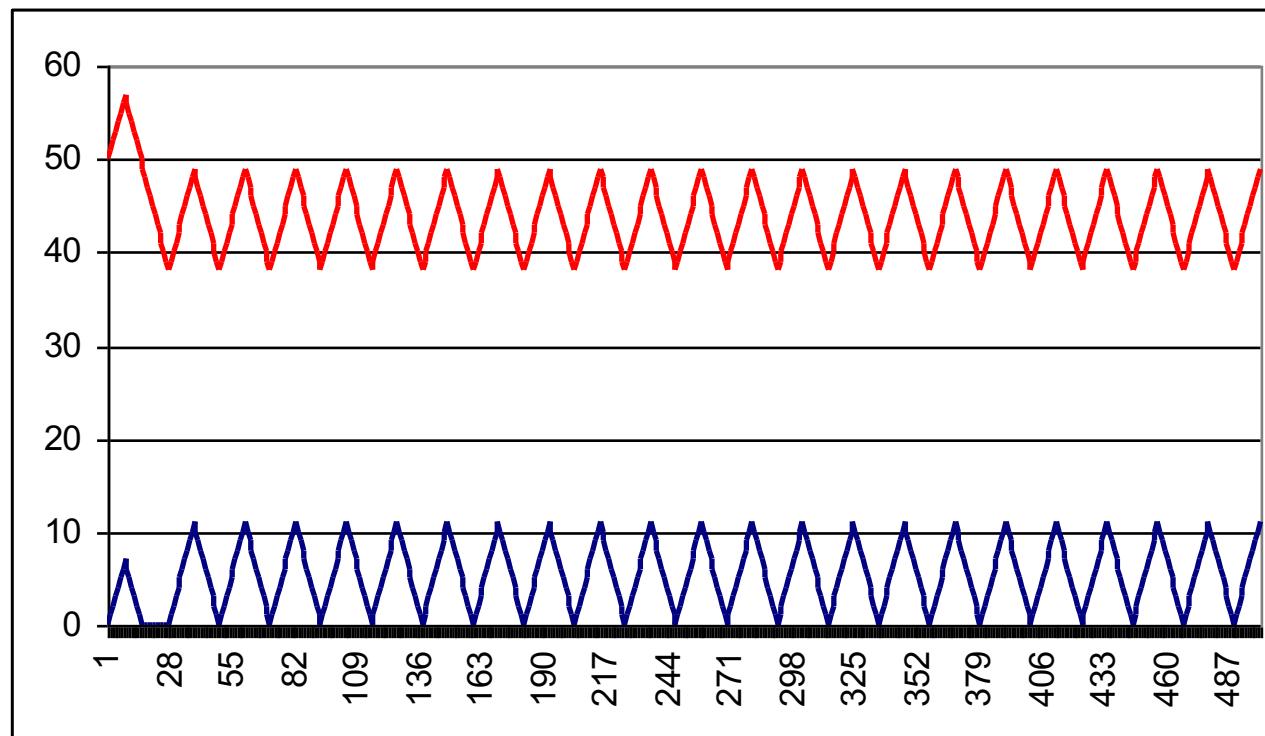
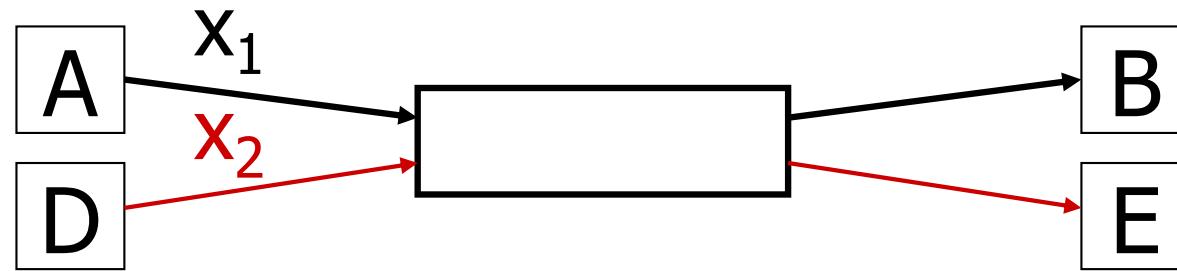
# AIAD

- ❖ Increase:  $x + a_I$
- ❖ Decrease:  $x - a_D$
- ❖ Does not converge to fairness



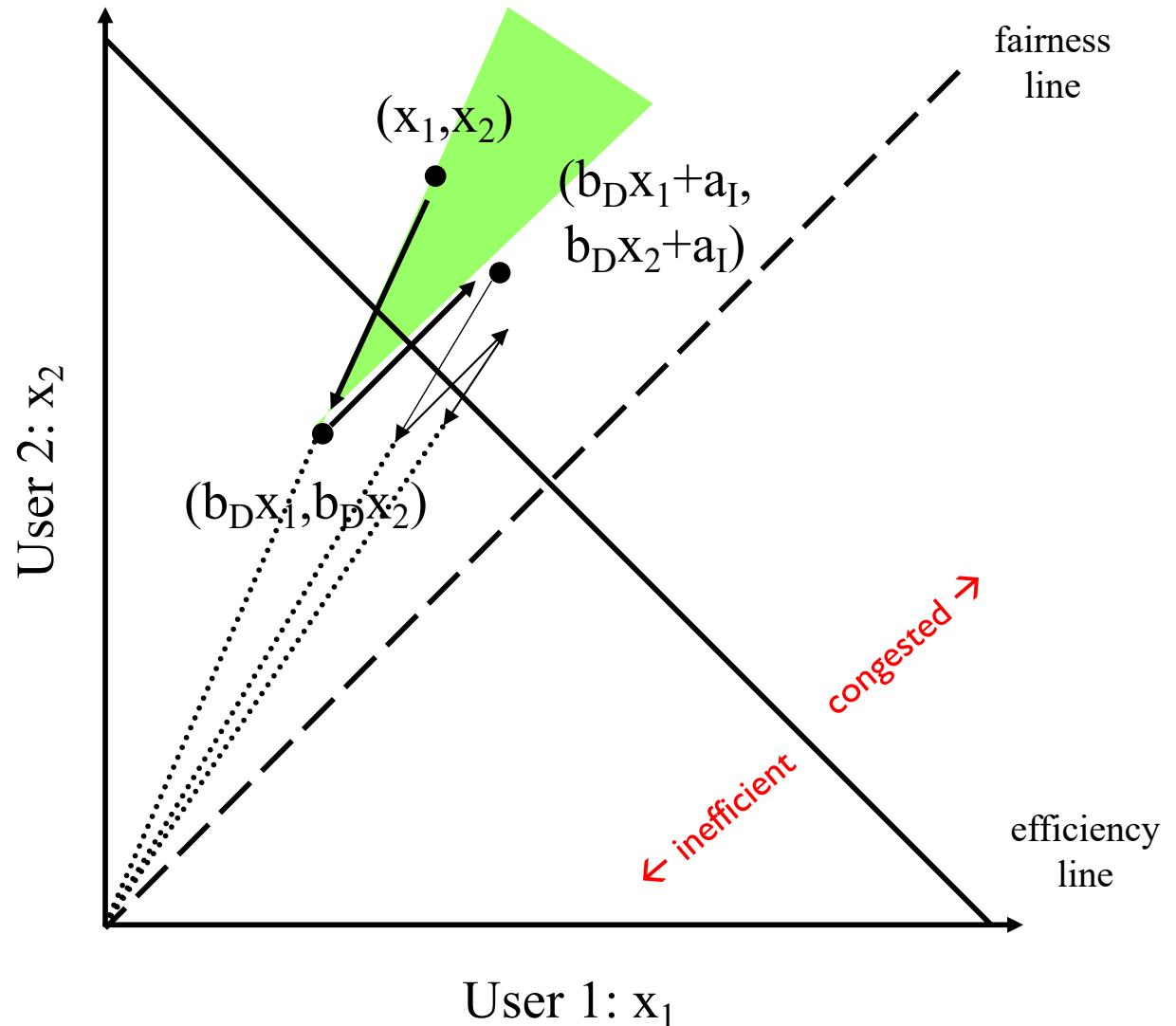
# AIAD Sharing Dynamics

---

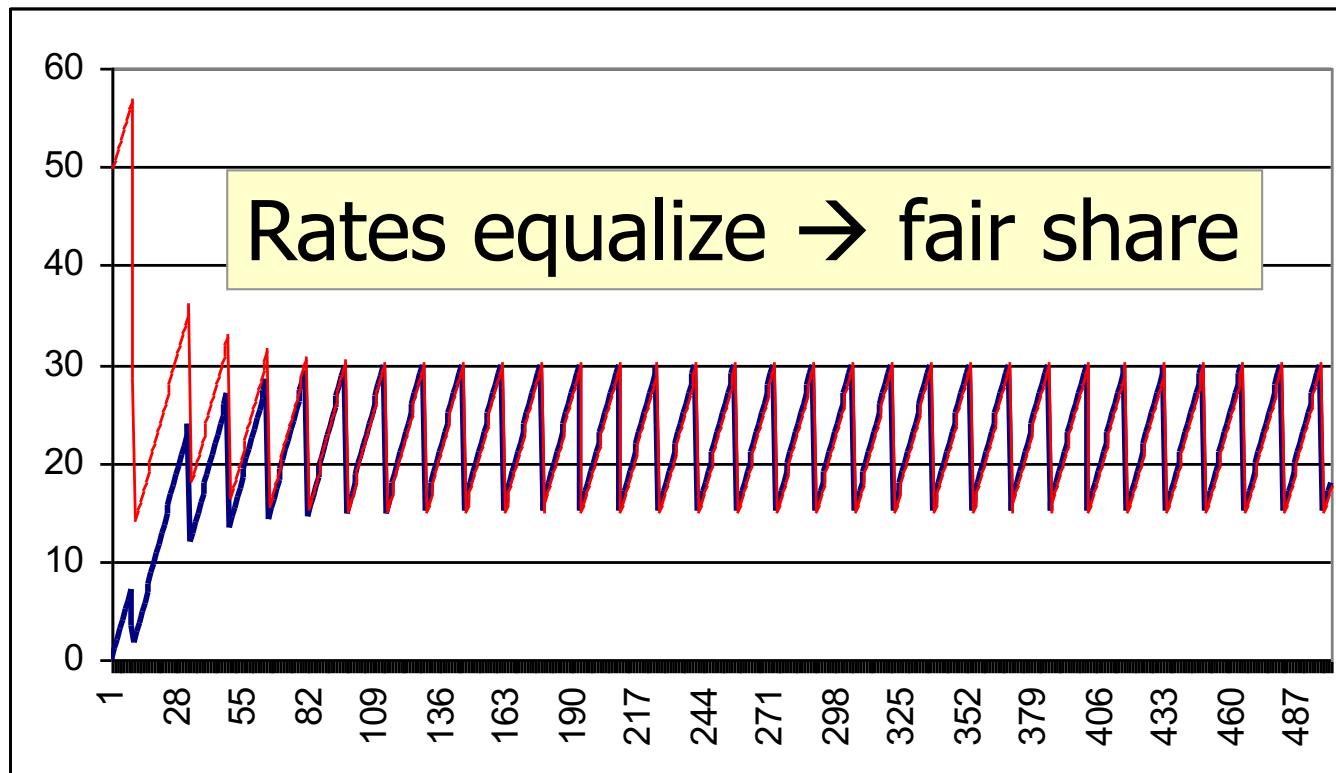


# AIMD

- ❖ Increase:  $x+a_I$
- ❖ Decrease:  $x*b_D$
- ❖ Converges to fairness



# AIMD Sharing Dynamics



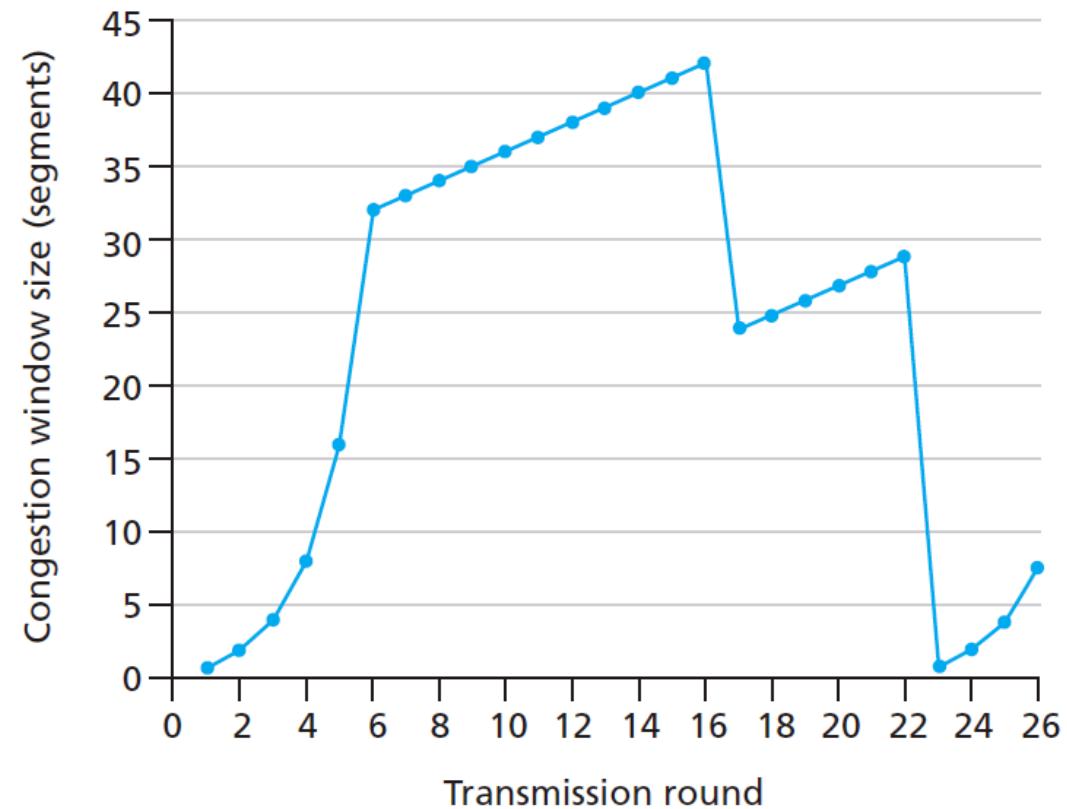
# Quiz: TCP Congestion Control?

---



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

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

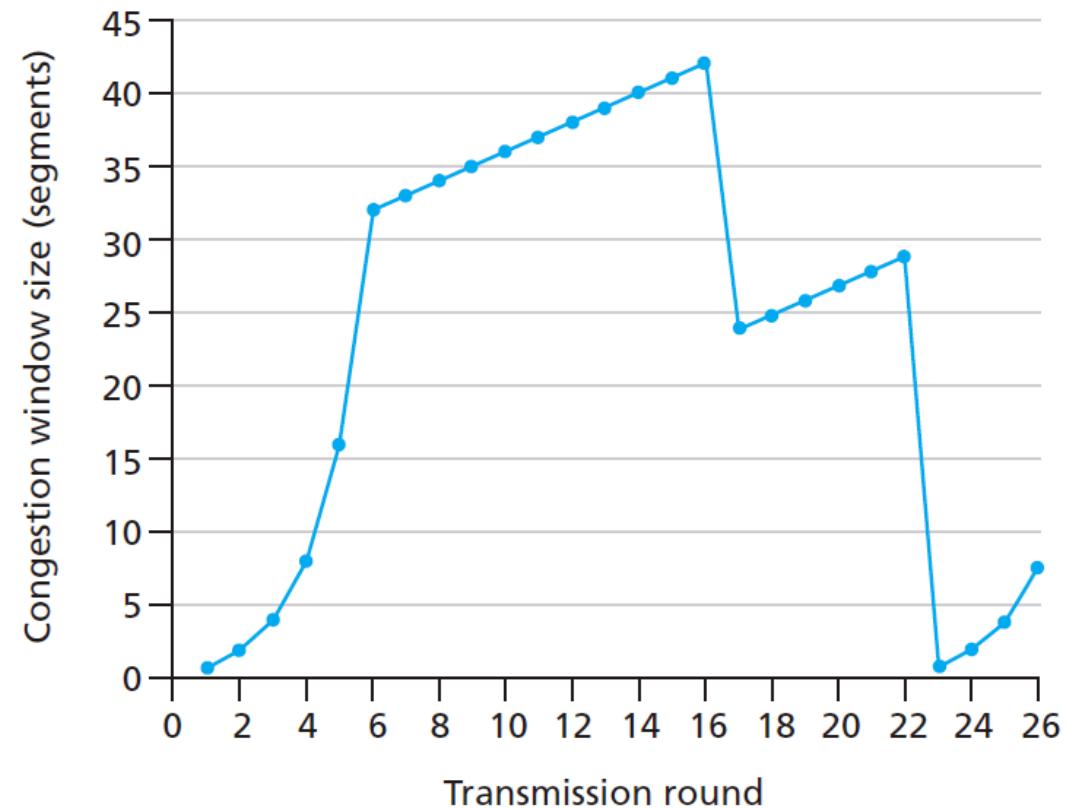


# Quiz: TCP Congestion Control?



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

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

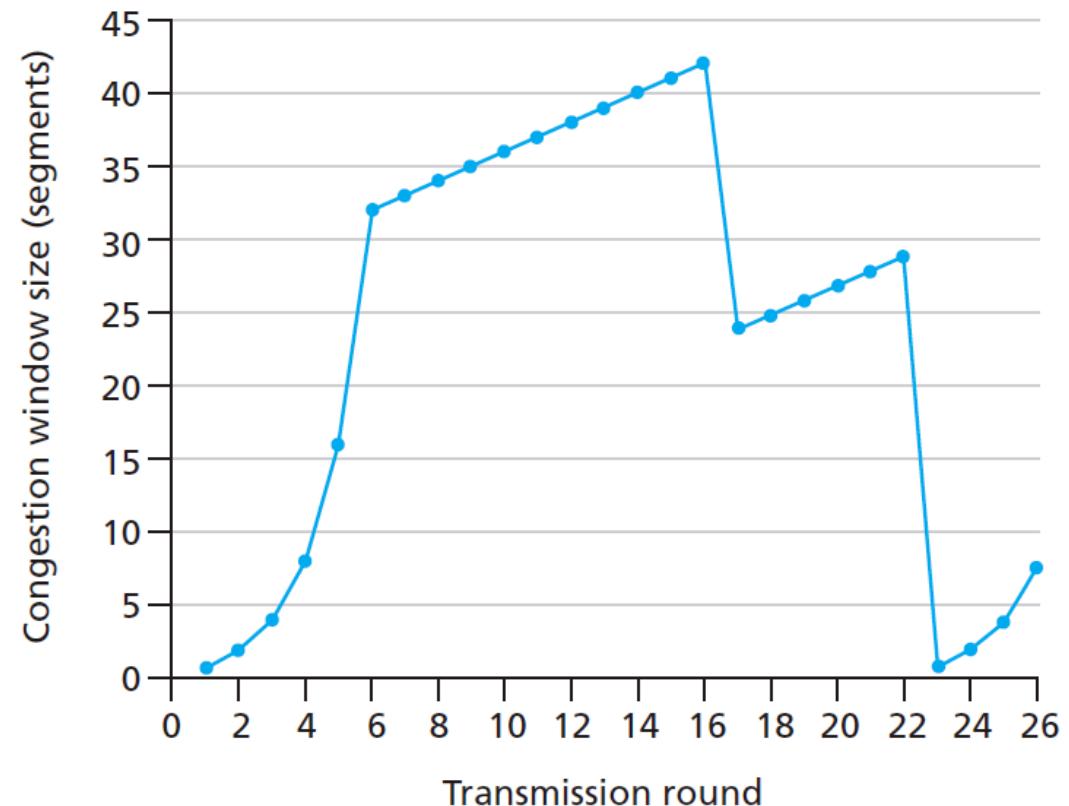


# Quiz: TCP Congestion Control?



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

- A. Triple Dup Ack
- B. Timeout

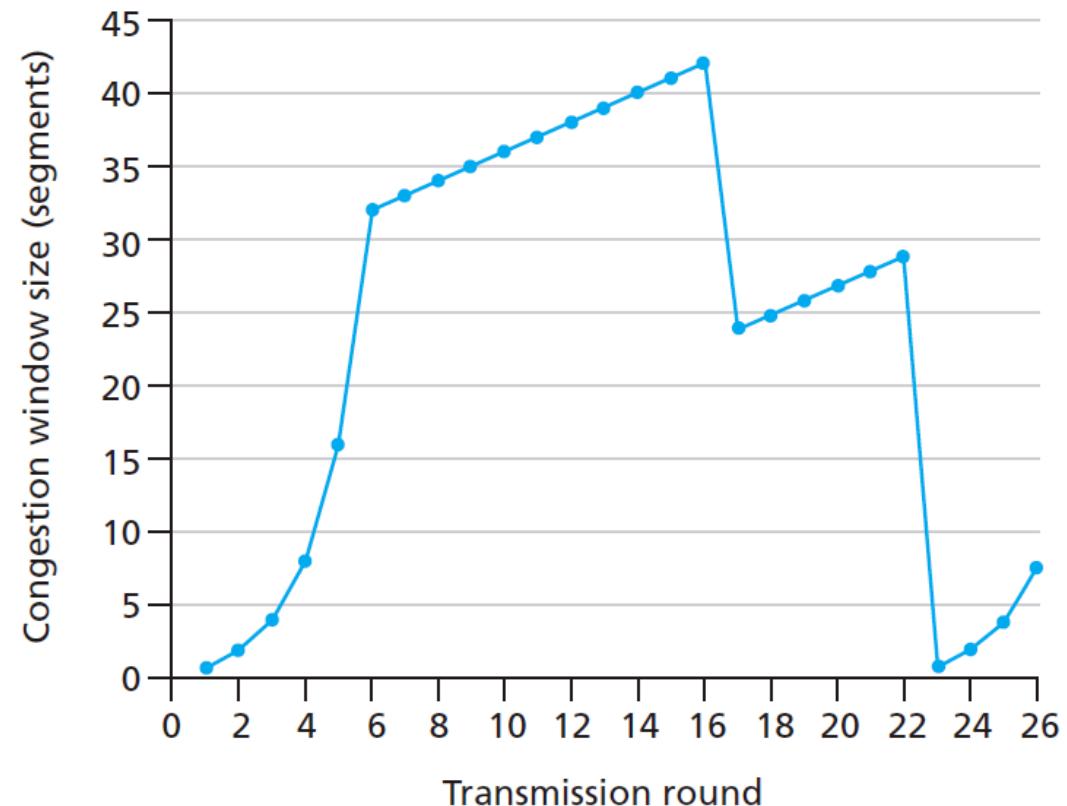


# Quiz: TCP Congestion Control?



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

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

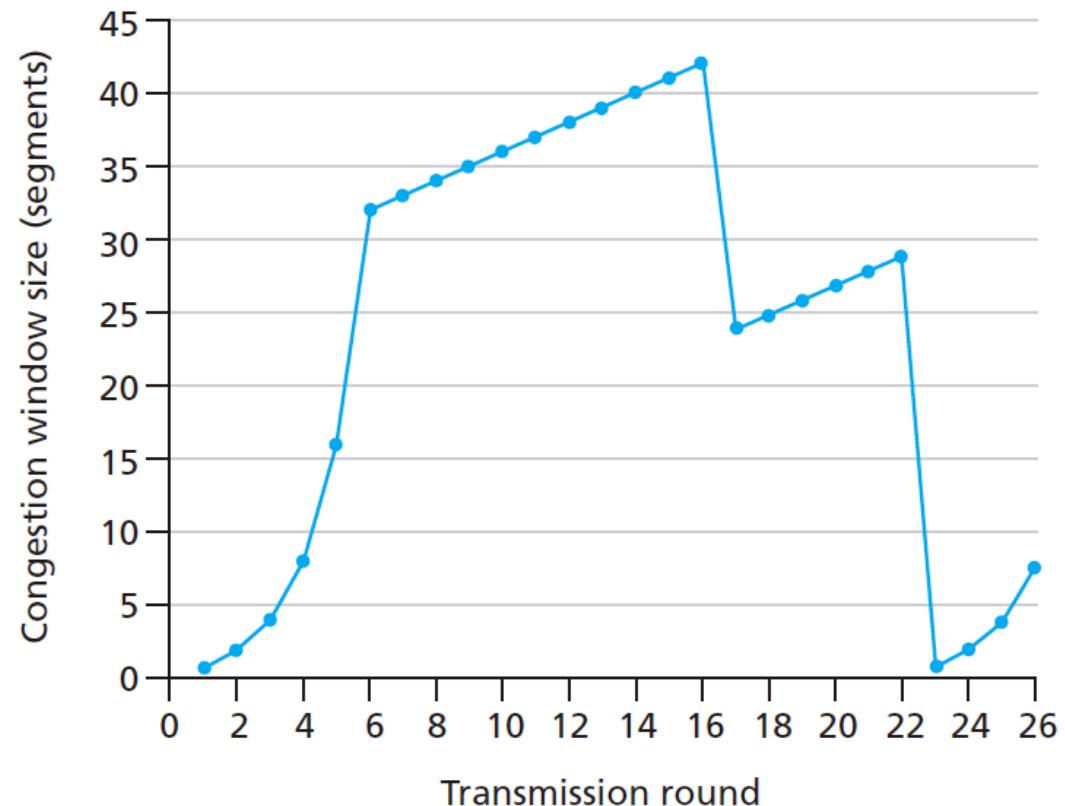


# Quiz: TCP Congestion Control?



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



# 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”