

Chapter 1

Introduction

A note on the use of these PowerPoint slides:

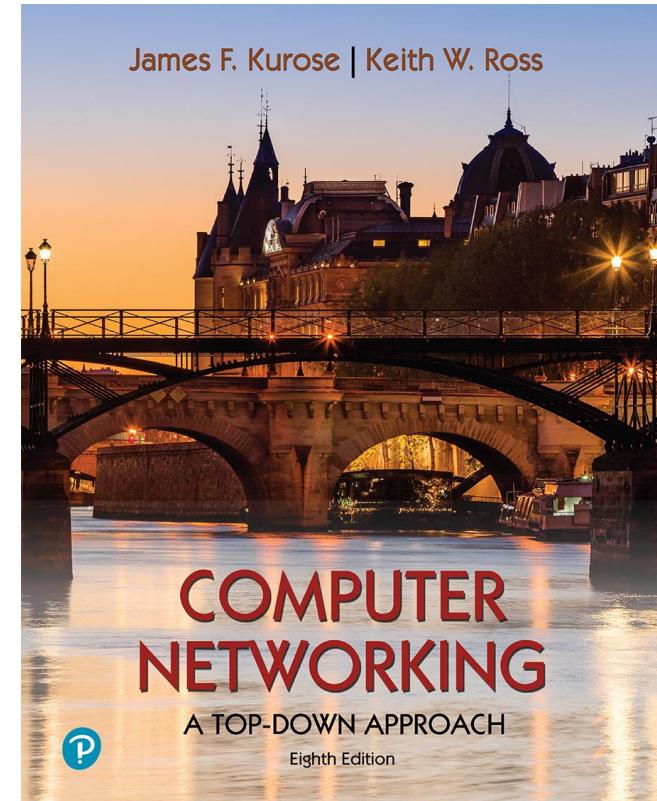
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use,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

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

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A
Top-Down Approach*
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Chapter 1: introduction

Chapter goal:

- Get “feel,” “big picture,” introduction to terminology
 - more depth, detail *later* in course



Overview/roadmap:

- What *is* the Internet? What *is* a protocol?
- **Network edge:** hosts, access network, physical media
- **Network core:** packet/circuit switching, internet structure
- **Performance:** loss, delay, throughput
- Protocol layers, service models
- Security
- History

The Internet: a “nuts and bolts” view



Billions of connected computing *devices*:

- *hosts* = end systems
- running *network apps* at Internet's “edge”

Packet switches: forward packets (chunks of data)

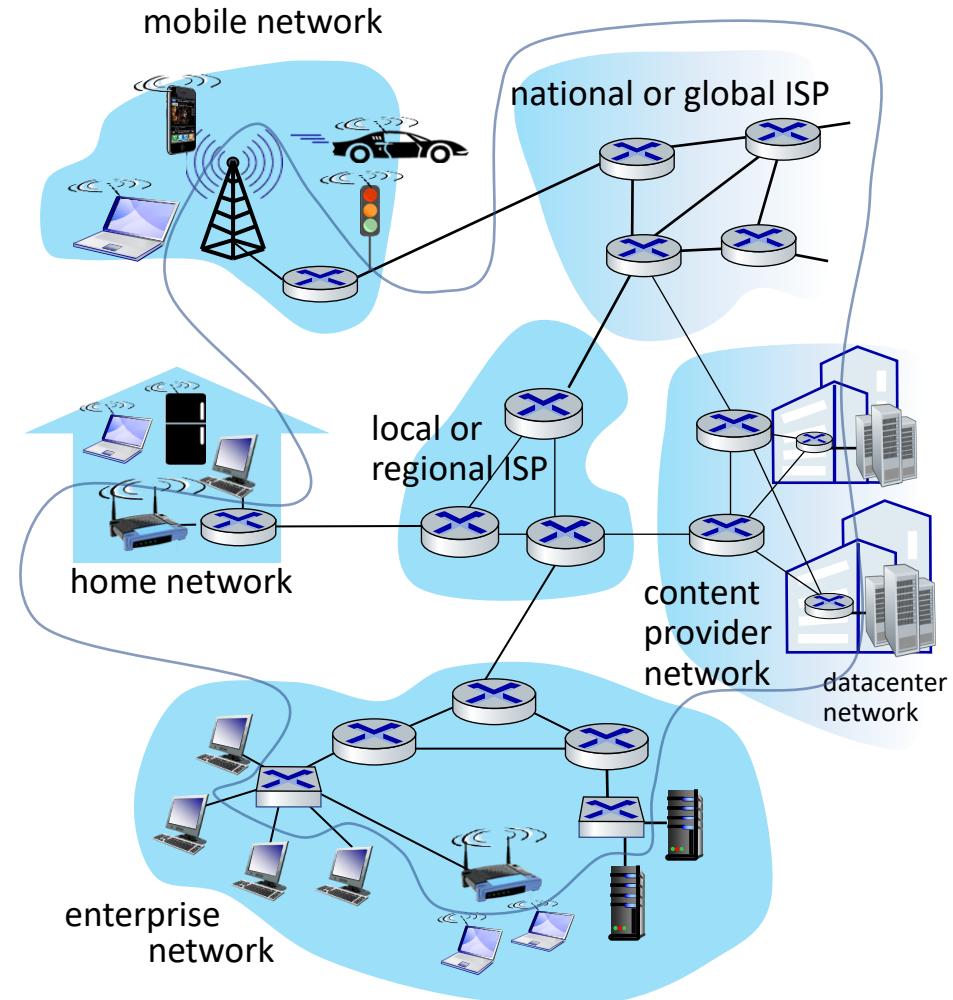
- routers, switches

Communication links

- fiber, copper, radio, satellite
- transmission rate: *bandwidth*

Networks

- collection of devices, routers, links: managed by an organization



“Fun” Internet-connected devices



Amazon Echo



Internet refrigerator



Security Camera



Internet phones



IP picture frame



Slingbox: remote control cable TV



Gaming devices



Pacemaker & Monitor



Web-enabled toaster + weather forecaster



sensorized, bed mattress



Fitbit



Tweet-a-watt:
monitor energy use

bikes



cars

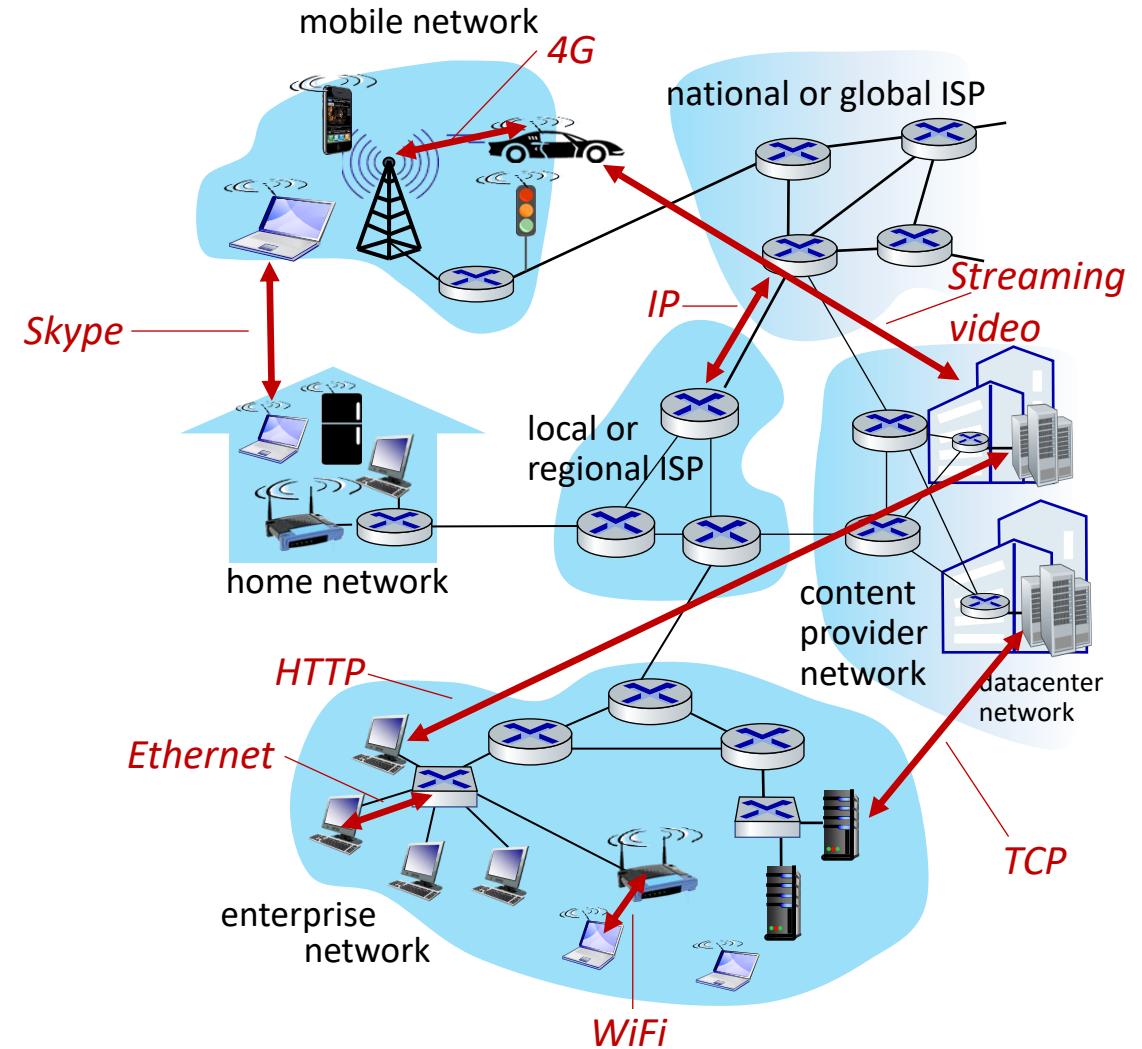


scooters

Others?

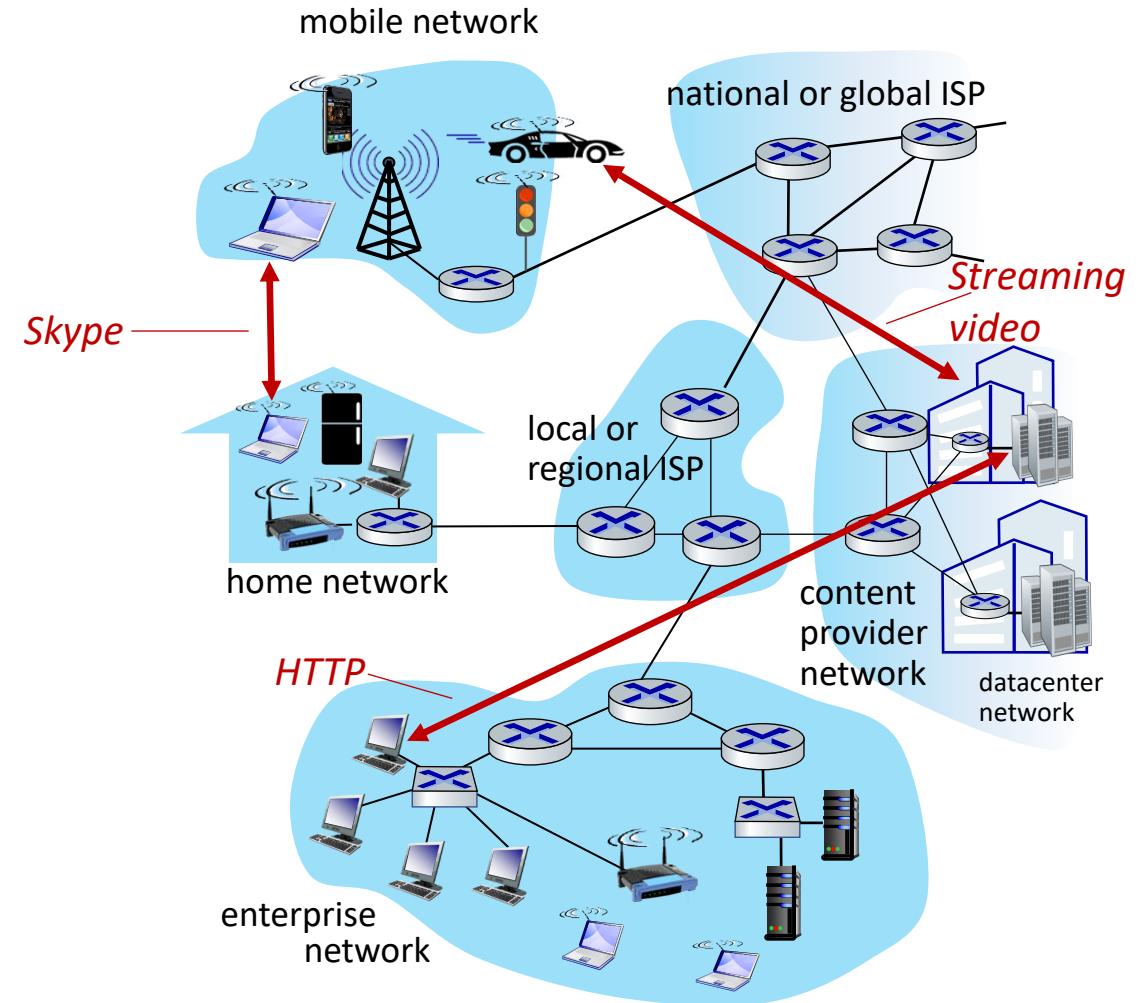
The Internet: a “nuts and bolts” view

- *Internet: “network of networks”*
 - Interconnected ISPs
- *protocols are everywhere*
 - control sending, receiving of messages
 - e.g., HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, Ethernet
- *Internet standards*
 - RFC: Request for Comments
 - IETF: Internet Engineering Task Force



The Internet: a “services” view

- *Infrastructure* that provides services to applications:
 - Web, streaming video, multimedia teleconferencing, email, games, e-commerce, social media, interconnected appliances, ...
- provides *programming interface* to distributed applications:
 - “hooks” allowing sending/receiving apps to “connect” to, use Internet transport service
 - provides service options, analogous to postal service



What's a protocol?

Human protocols:

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

Rules for:

- ... specific messages sent
- ... specific actions taken
when message received,
or other events

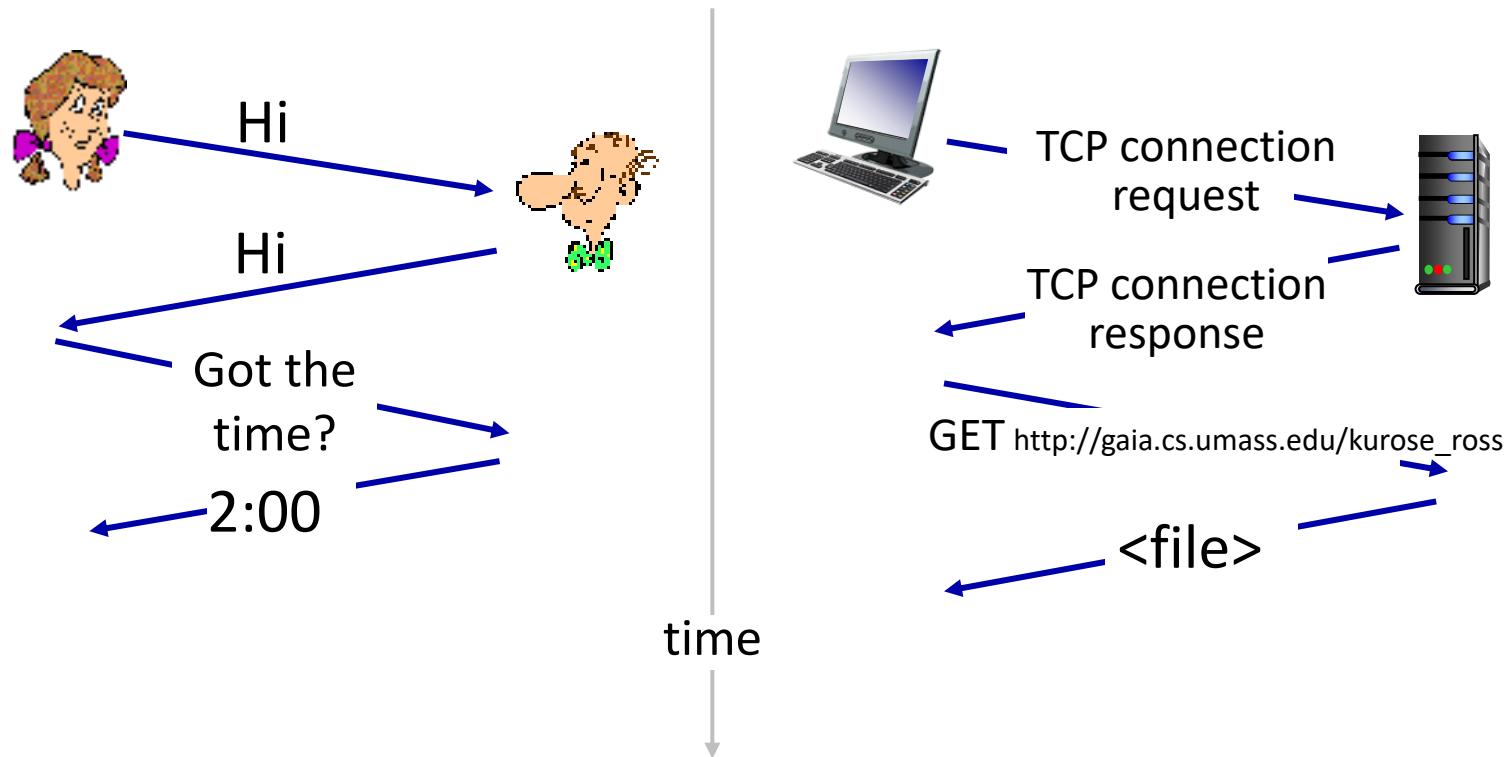
Network protocols:

- computers (devices) rather than humans
- all communication activity in Internet governed by protocols

*Protocols define the **format, order** of messages sent and received among network entities, and **actions taken** on message transmission, receipt*

What's a protocol?

A human protocol and a computer network protocol:



Q: other human protocols?

Chapter 1: roadmap

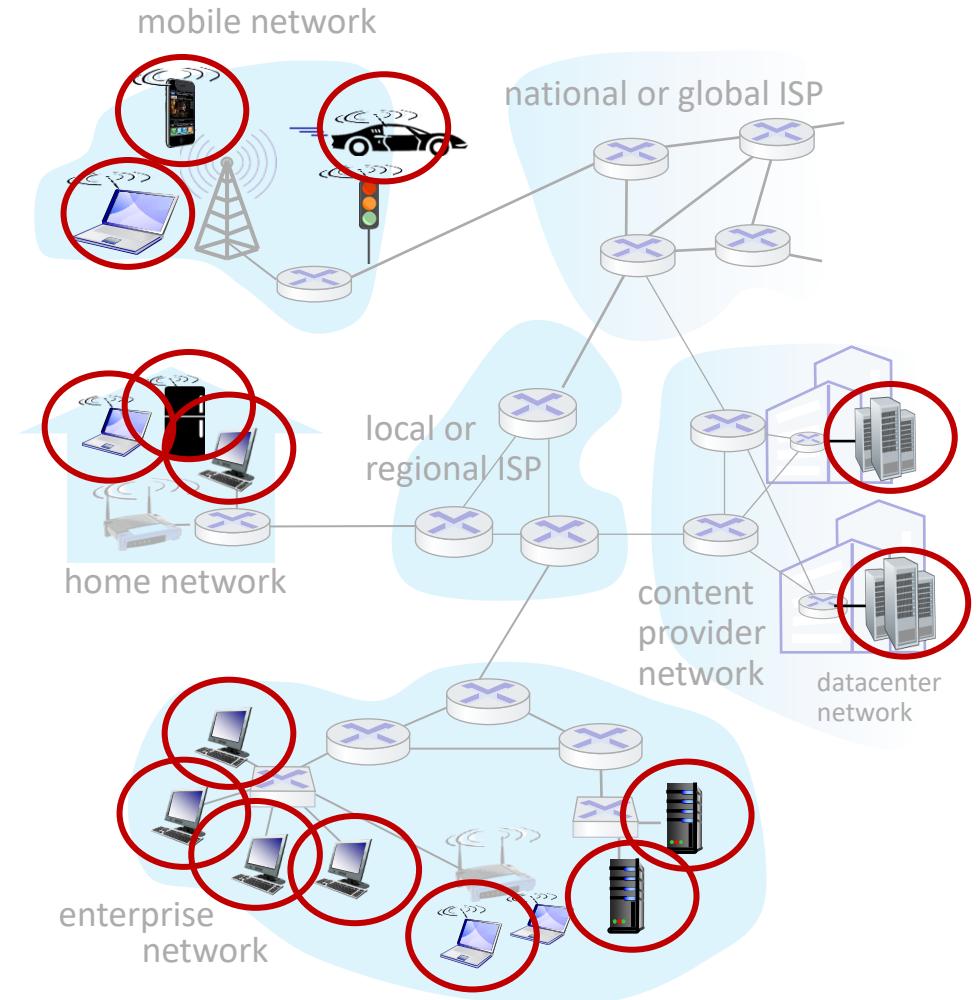
- What *is* the Internet?
- What *is* a protocol?
- **Network edge:** hosts, access network, physical media
- Network core: packet/circuit switching, internet structure
- Performance: loss, delay, throughput
- Security
- Protocol layers, service models
- History



A closer look at Internet structure

Network edge:

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



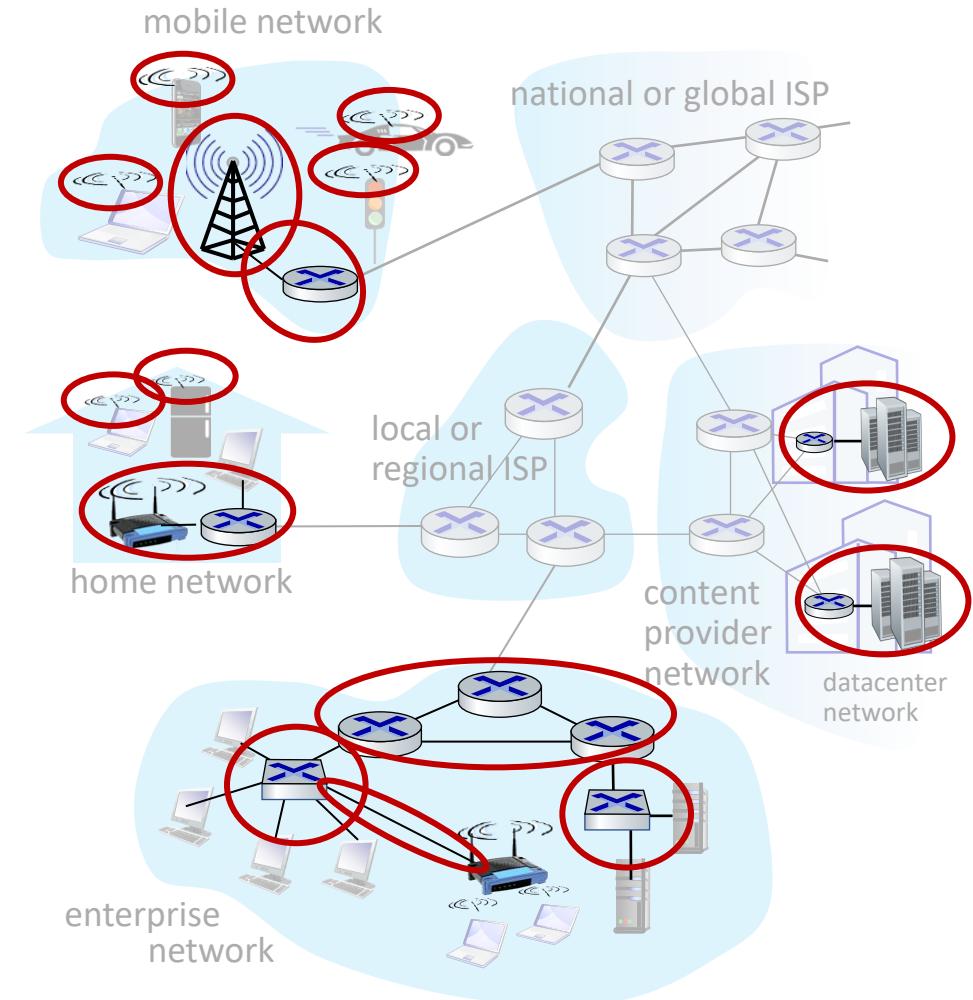
A closer look at Internet structure

Network edge:

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

Access networks, physical media:

- wired, wireless communication links



A closer look at Internet structure

Network edge:

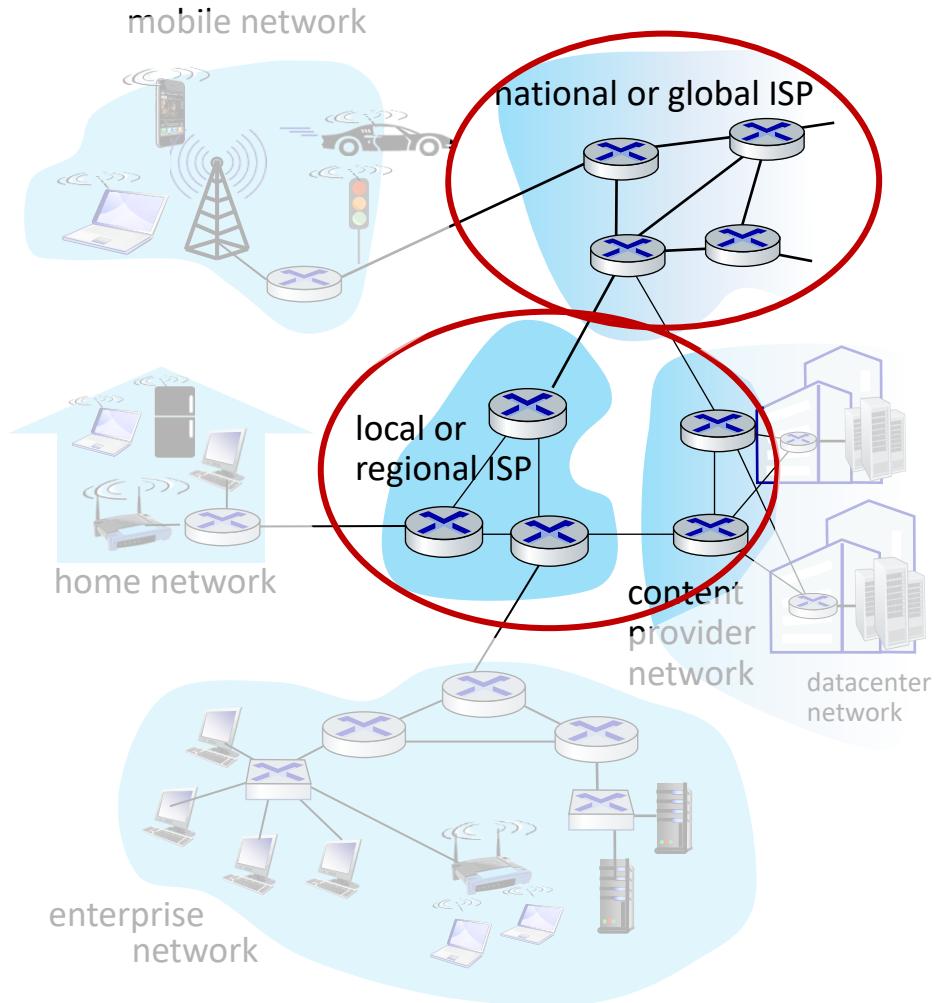
- hosts: clients and servers
- servers often in data centers

Access networks, physical media:

- wired, wireless communication links

Network core:

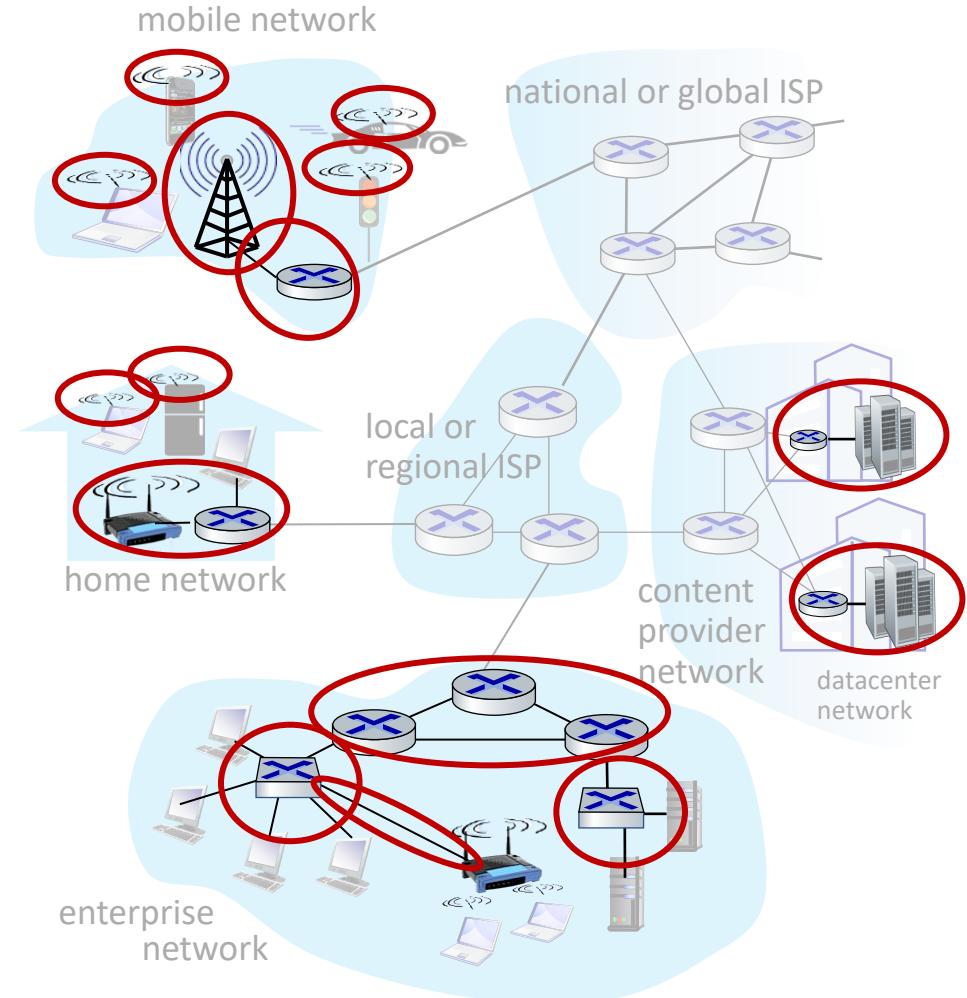
- interconnected routers
- network of networks



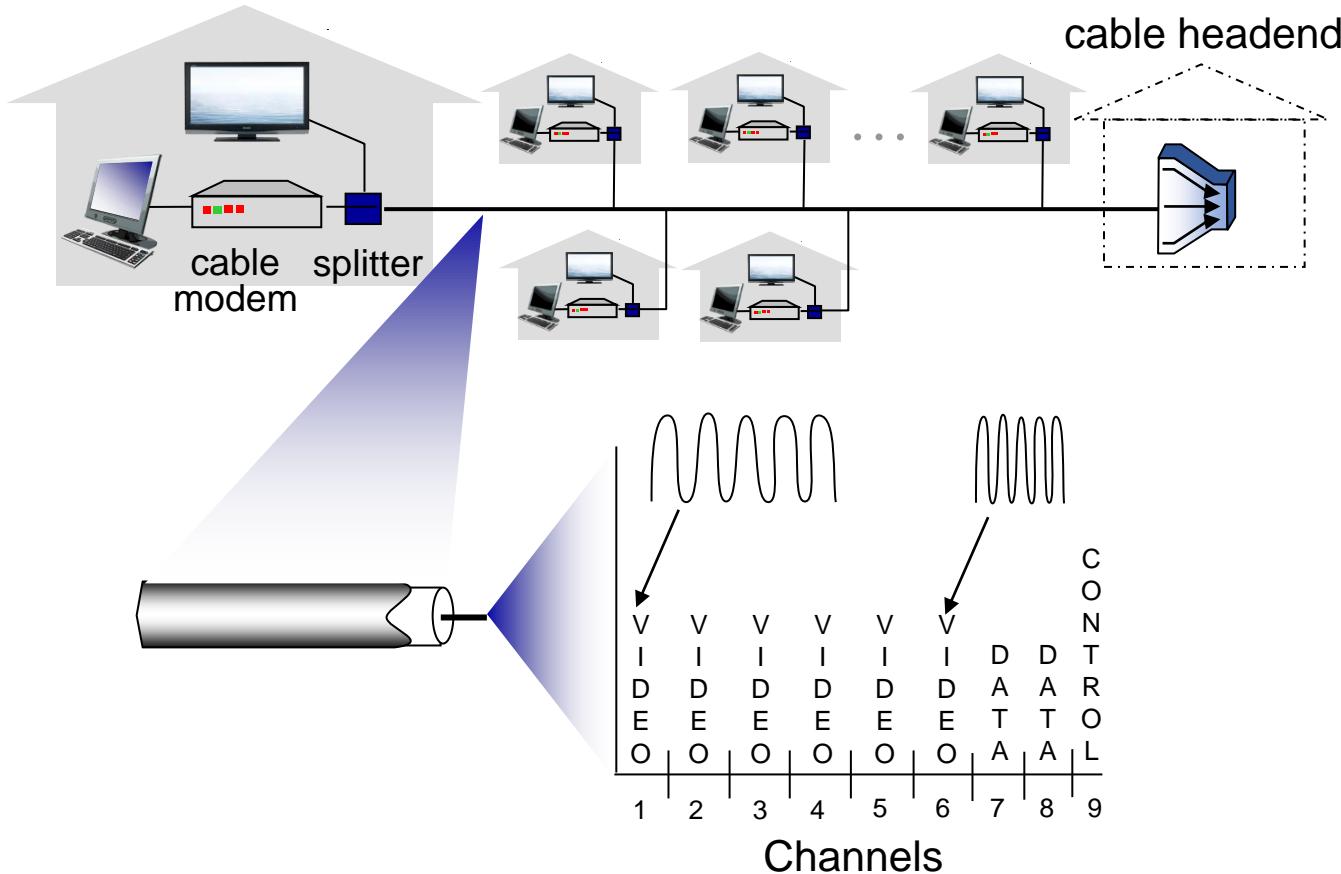
Access networks and physical media

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

- residential access nets
- institutional access networks (school, company)
- mobile access networks (WiFi, 4G/5G)

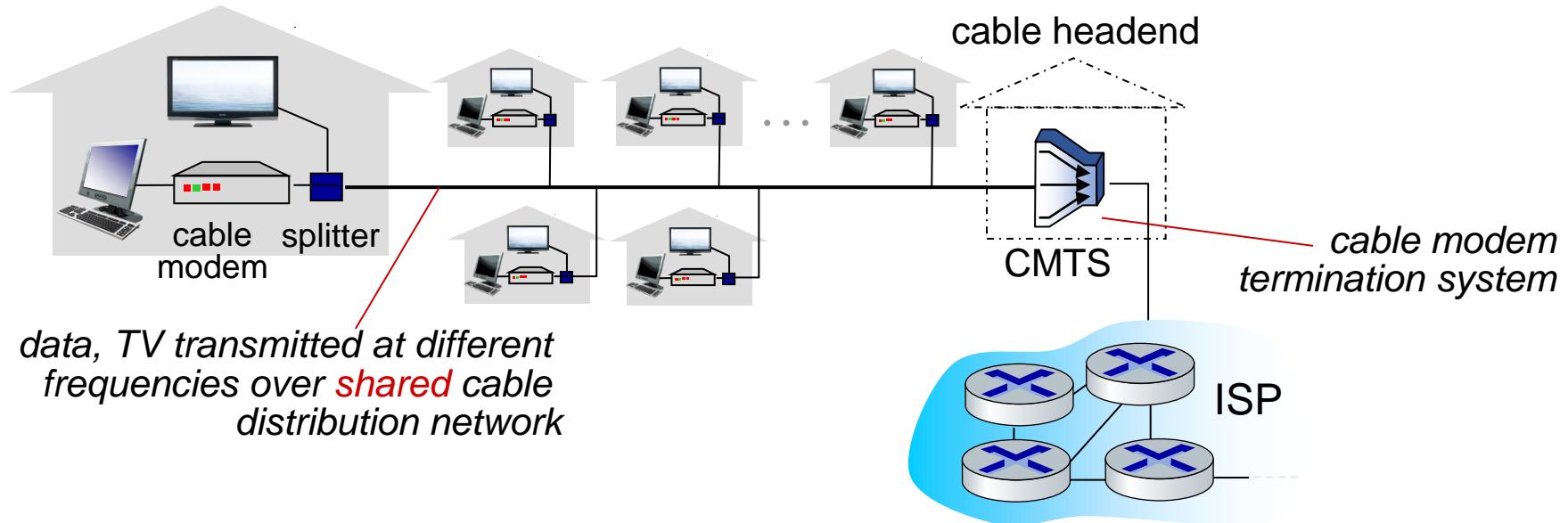


Access networks: cable-based access



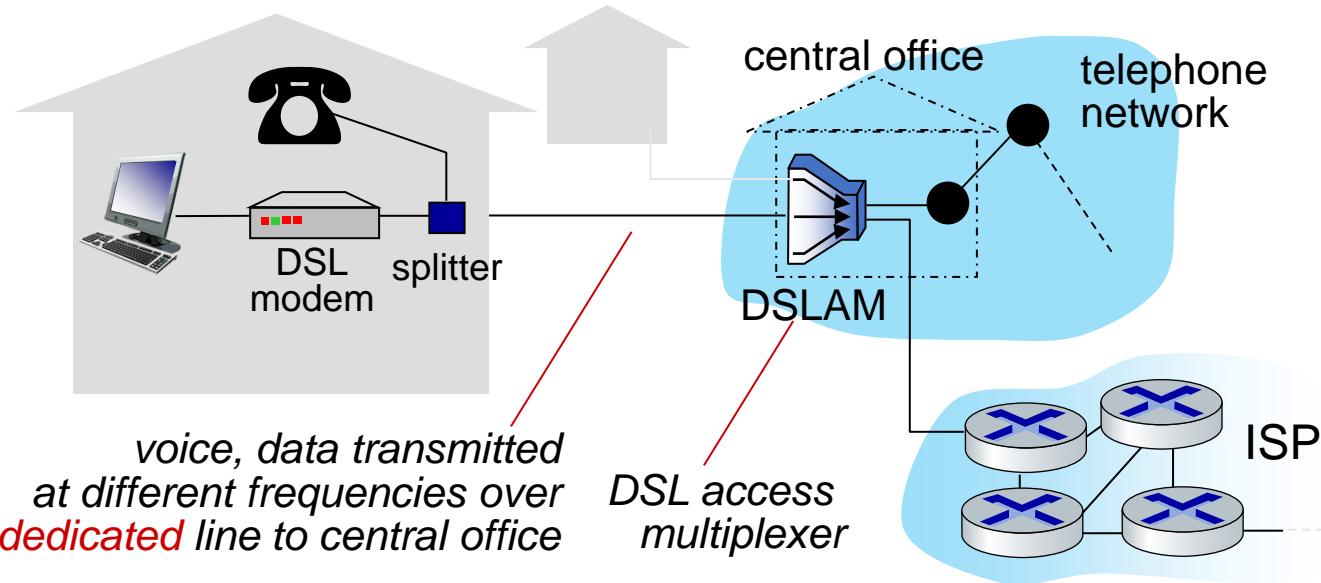
frequency division multiplexing (FDM): different channels transmitted in different frequency bands

Access networks: cable-based access



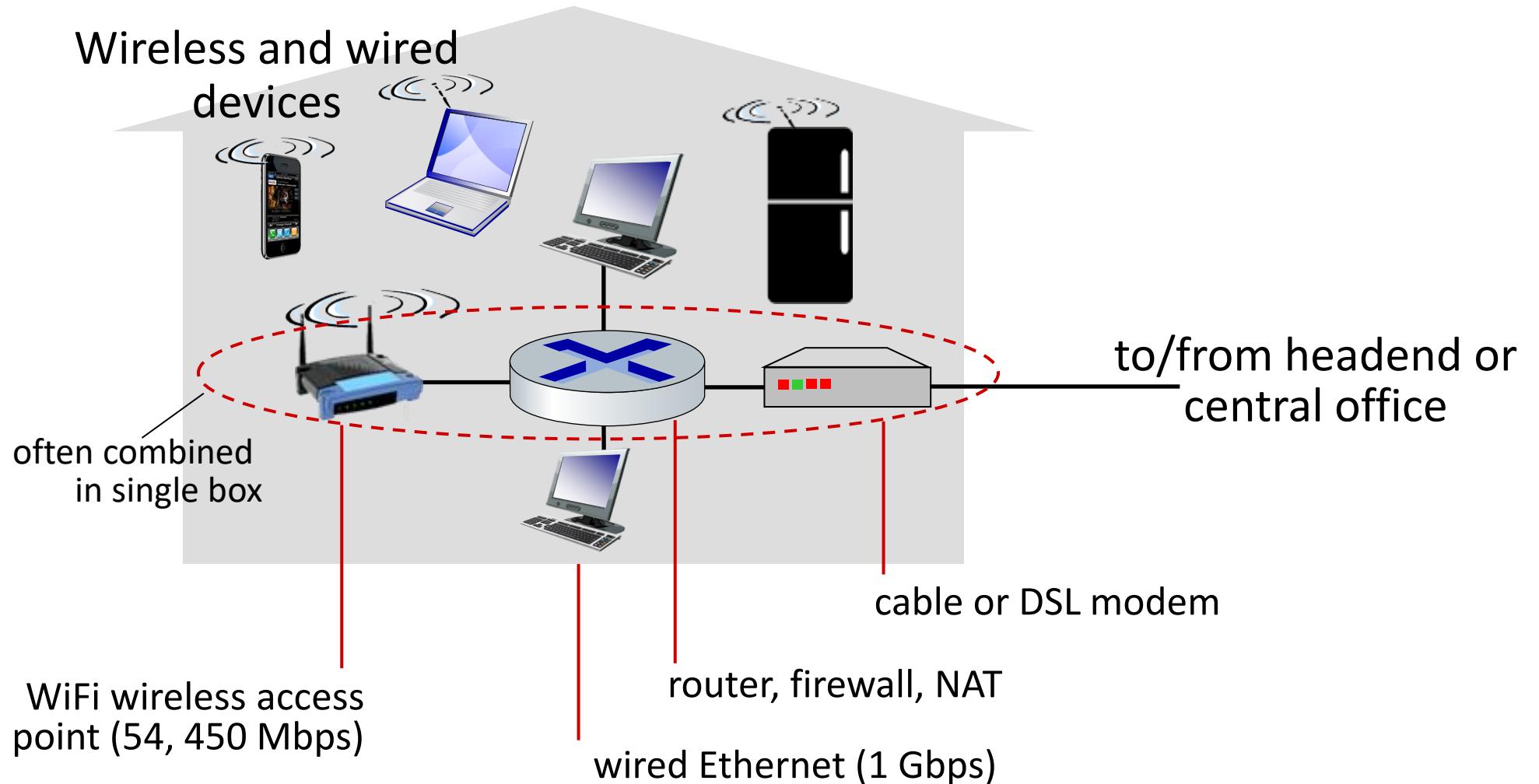
- HFC: hybrid fiber coax
 - asymmetric: up to 40 Mbps – 1.2 Gbps downstream transmission rate, 30-100 Mbps upstream transmission rate
- network of cable, fiber attaches homes to ISP router
 - homes **share access network** to cable headend

Access networks: 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
- 24-52 Mbps dedicated downstream transmission rate
- 3.5-16 Mbps dedicated upstream transmission rate

Access networks: home networks



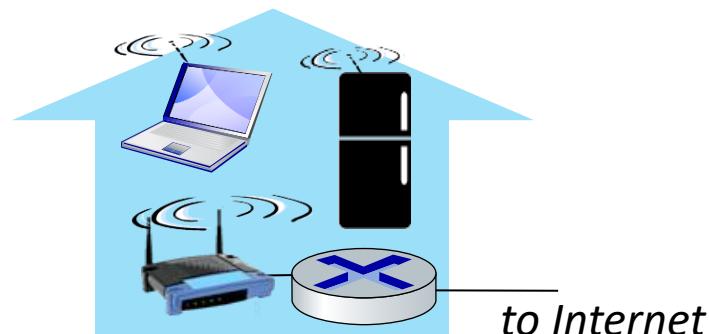
Wireless access networks

Shared *wireless* access network connects end system to router

- via base station aka “access point”

Wireless local area networks (WLANs)

- typically within or around building (~100 ft)
- 802.11b/g/n (WiFi): 11, 54, 450 Mbps transmission rate

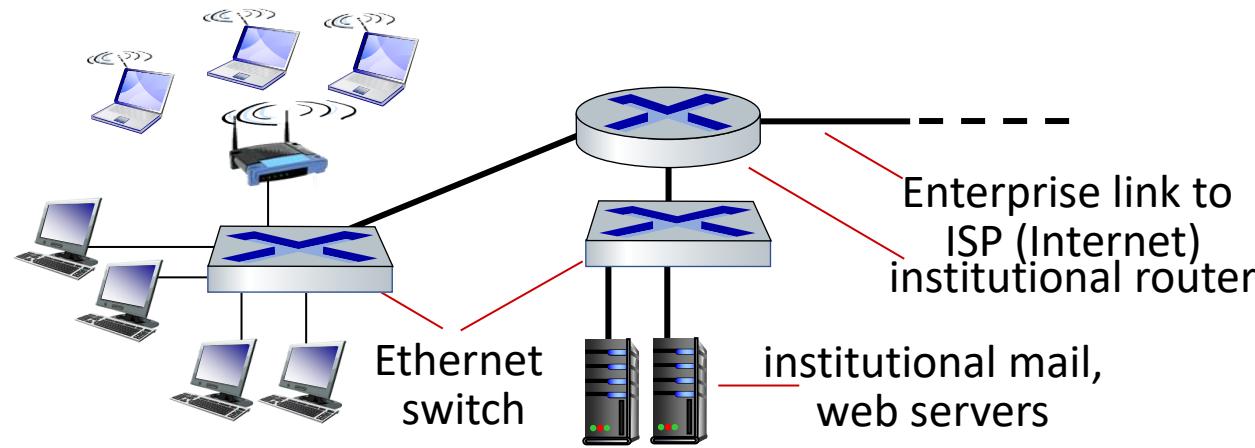


Wide-area cellular access networks

- provided by mobile, cellular network operator (10's km)
- 10's Mbps
- 4G cellular networks (5G coming)



Access networks: enterprise networks



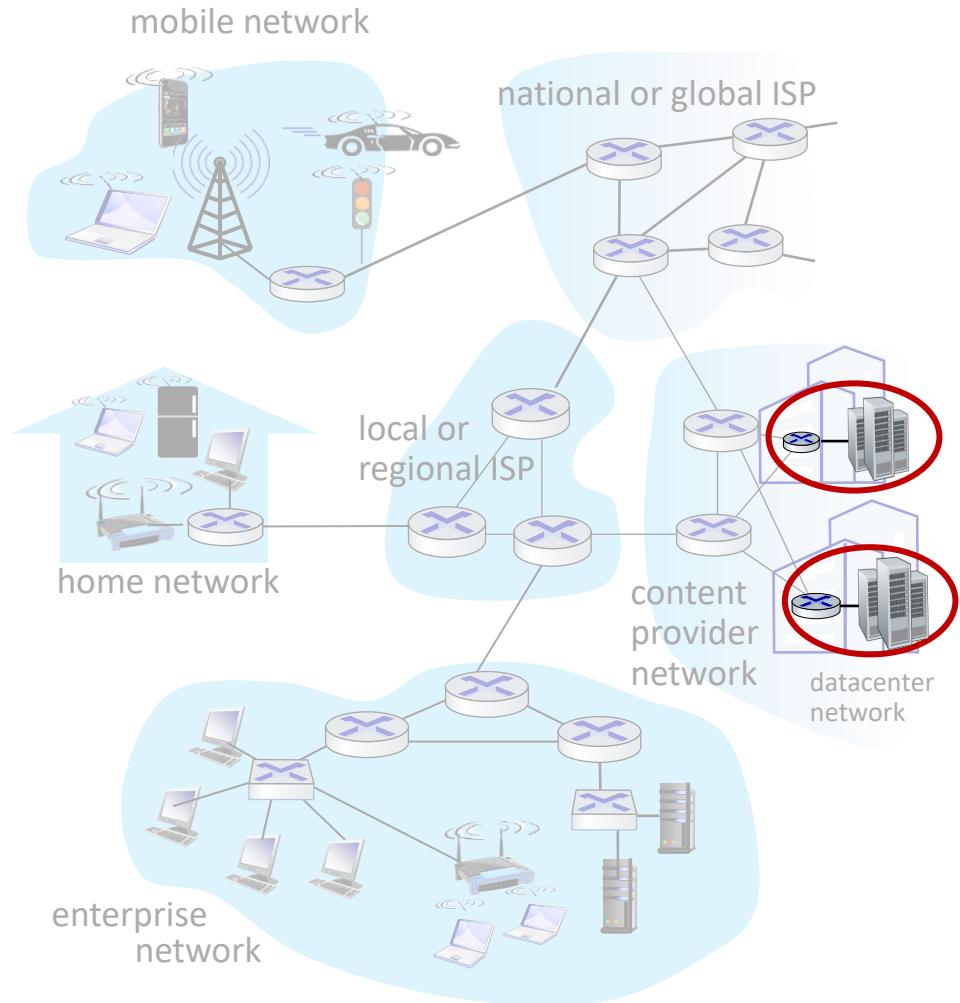
- companies, universities, etc.
- mix of wired, wireless link technologies, connecting a mix of switches and routers (we'll cover differences shortly)
 - Ethernet: wired access at 100Mbps, 1Gbps, 10Gbps
 - WiFi: wireless access points at 11, 54, 450 Mbps

Access networks: data center networks

- high-bandwidth links (10s to 100s Gbps) connect hundreds to thousands of servers together, and to Internet



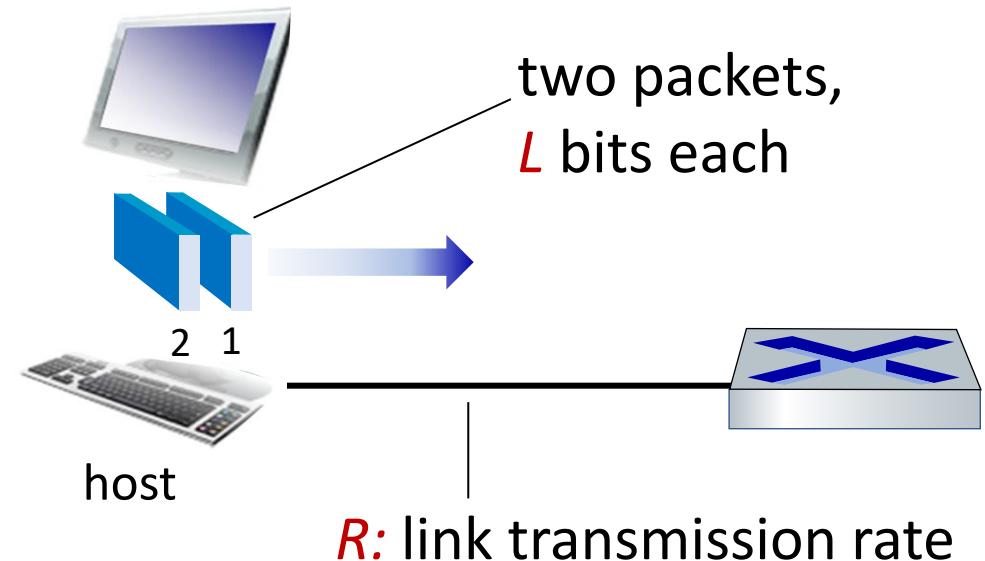
Courtesy: Massachusetts Green High Performance Computing Center (mghpcc.org)



Host: sends *packets* of data

host sending function:

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



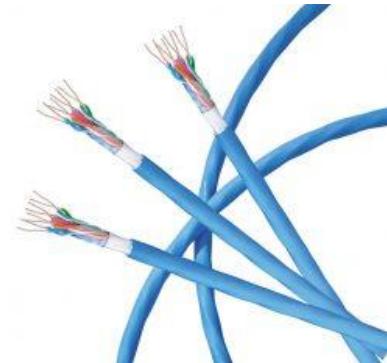
$$\text{packet transmission delay} = \frac{\text{time needed to transmit } L\text{-bit packet into link}}{R \text{ (bits/sec)}} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

Links: physical media

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

Twisted pair (TP)

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



Links: physical media

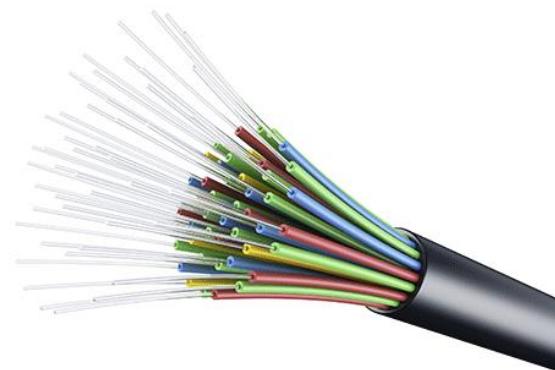
Coaxial cable:

- two concentric copper conductors
- bidirectional
- broadband:
 - multiple frequency channels on cable
 - 100's Mbps per channel



Fiber optic cable:

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



Links: physical media

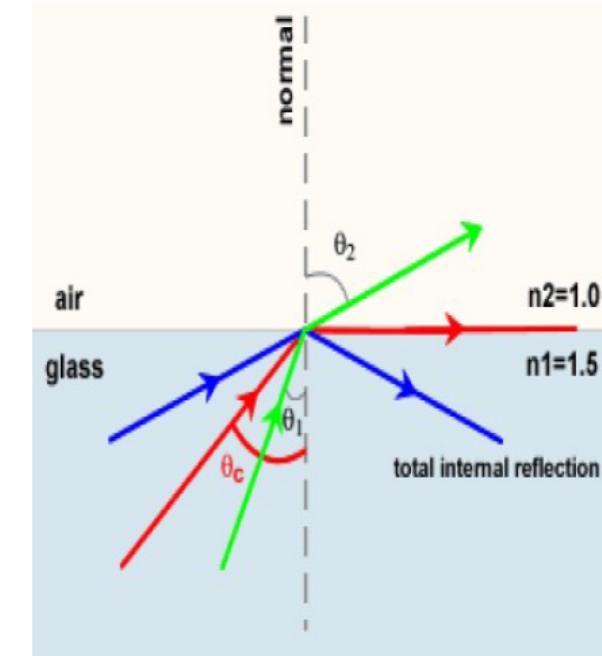
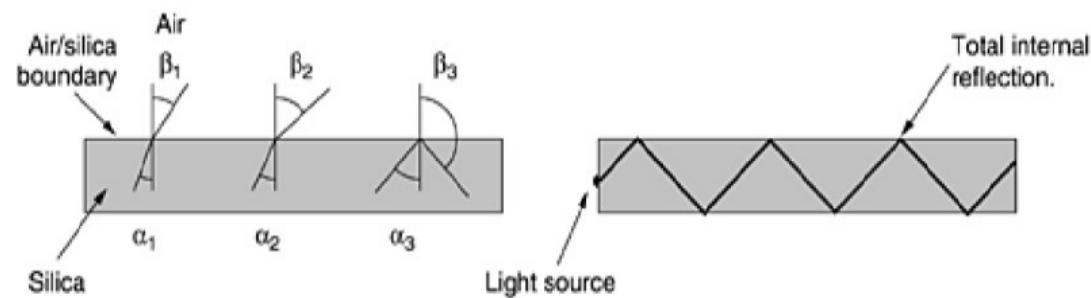
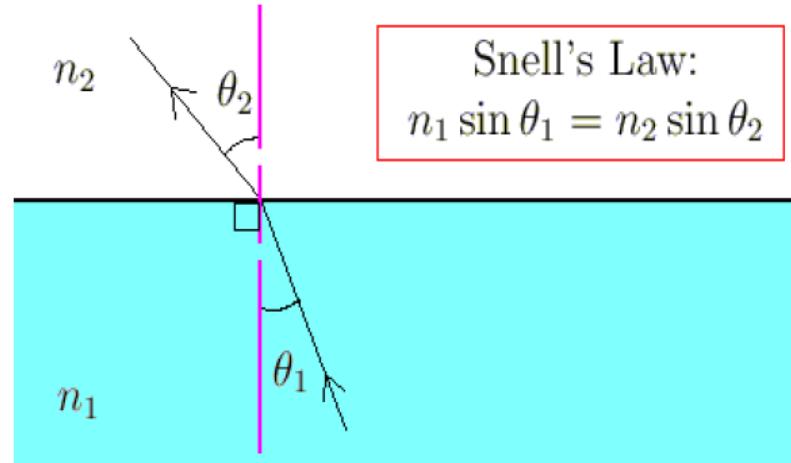
Wireless radio

- signal carried in various “bands” in electromagnetic spectrum
- no physical “wire”
- broadcast, “half-duplex” (sender to receiver)
- propagation environment effects:
 - reflection
 - obstruction by objects
 - Interference/noise

Radio link types:

- **Wireless LAN (WiFi)**
 - 10-100's Mbps; 10's of meters
- **wide-area** (e.g., 4G cellular)
 - 10's Mbps over ~10 Km
- **Bluetooth:** cable replacement
 - short distances, limited rates
- **terrestrial microwave**
 - point-to-point; 45 Mbps channels
- **satellite**
 - up to 45 Mbps per channel
 - 270 msec end-end delay

Snell's Law: Refraction, Critical Angle, TIR



Snell's Law (numerical example)

A beam of light going through a medium with an index of refraction of 1.14 has an angle of incidence of 48° with another medium. If the light wave is at a new angle of 39° , what is the index of refraction for the second medium?

Use Snell's Law. Answer: 1.346

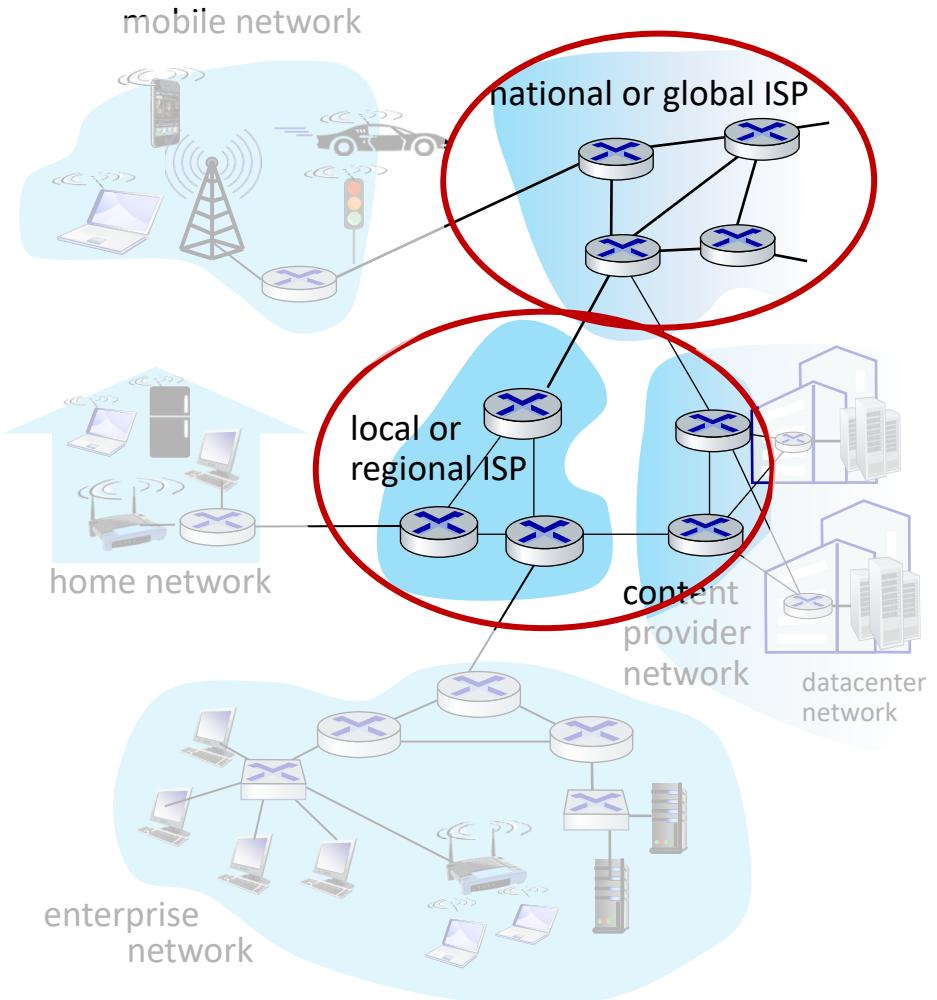
Chapter 1: roadmap

- What *is* the Internet?
- What *is* a protocol?
- Network edge: hosts, access network, physical media
- **Network core:** packet/circuit switching, internet structure
- Performance: loss, delay, throughput
- Security
- Protocol layers, service models
- History



The network core

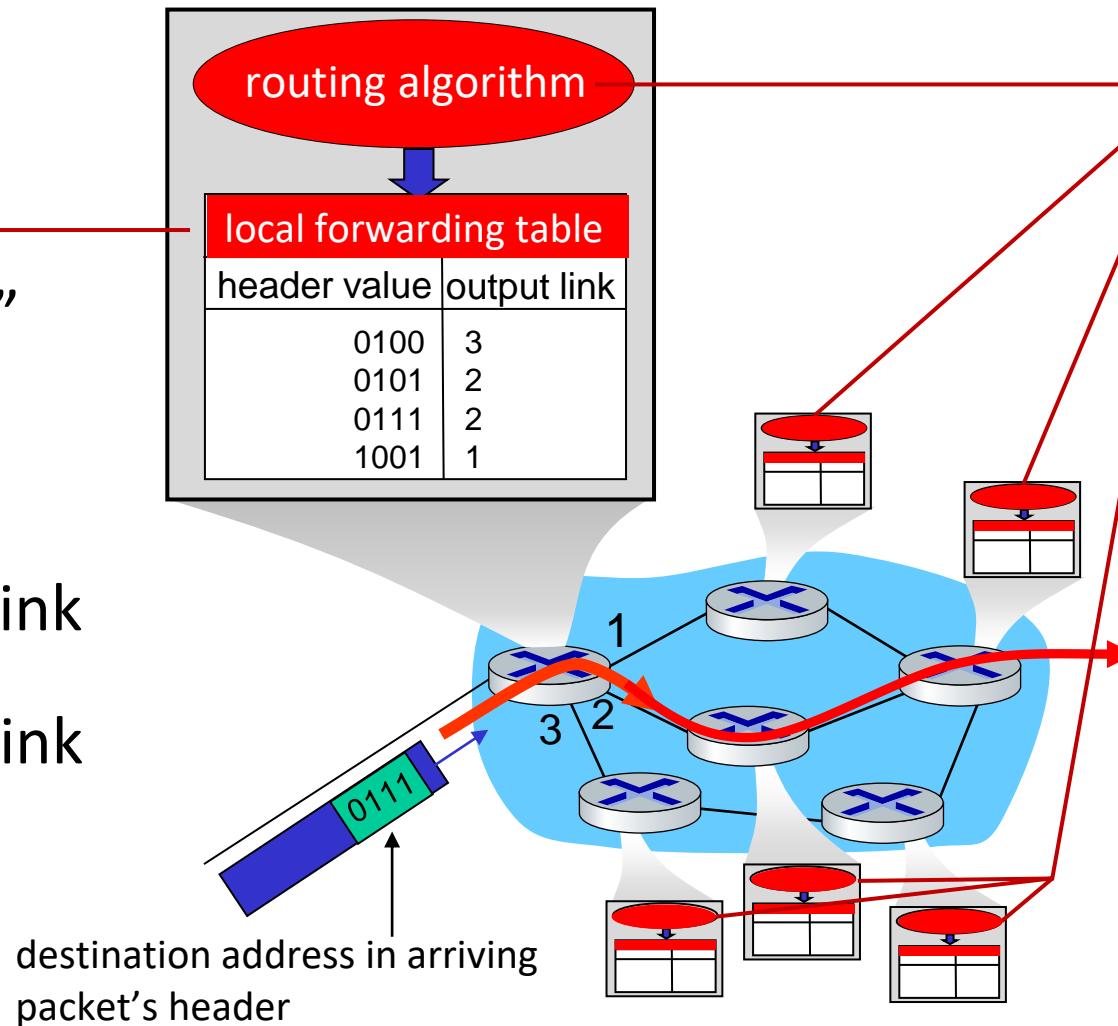
- mesh of interconnected routers
- **packet-switching**: hosts break application-layer messages into *packets*
 - network **forwards** packets from one router to the next, across links on path from **source to destination**



Two key network-core functions

Forwarding:

- aka “switching”
- *local* action:
move arriving
packets from
router’s input link
to appropriate
router output link



Routing:

- *global* action:
determine source-
destination paths
taken by packets
- routing algorithms

Sample routing table

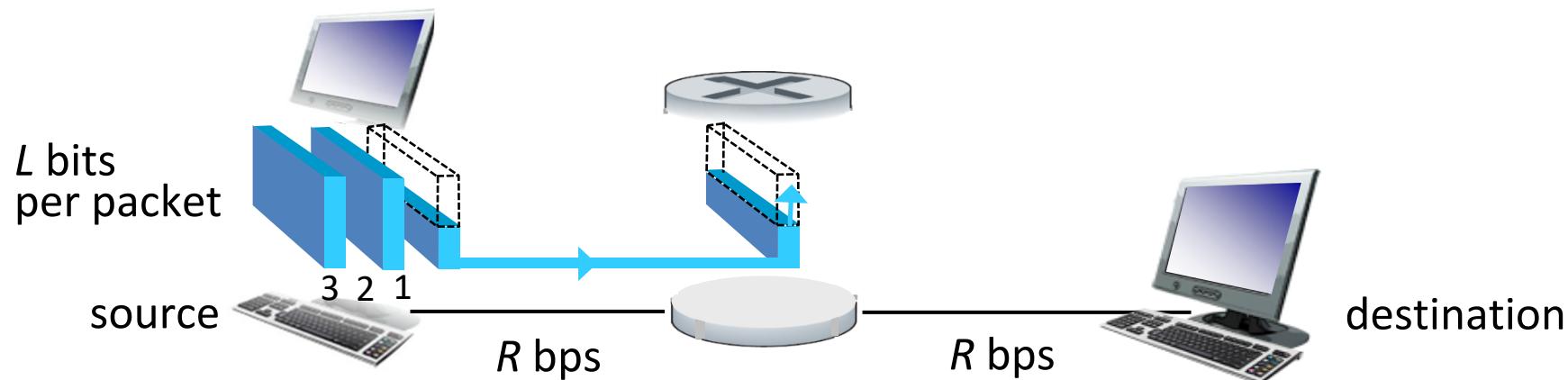
- Gateway & interface column refers to the forwarding table
- All columns combined are known as the routing table

Network Destination	Netmask	Gateway	Interface	Metric
101.25.67.0	255.255.255.0	10.0.0.2	eth3	1
default	0.0.0.0	10.0.0.1	eth0	0
192.25.67.0	255.255.255.0	10.0.0.3	eth5	10





Packet-switching: store-and-forward

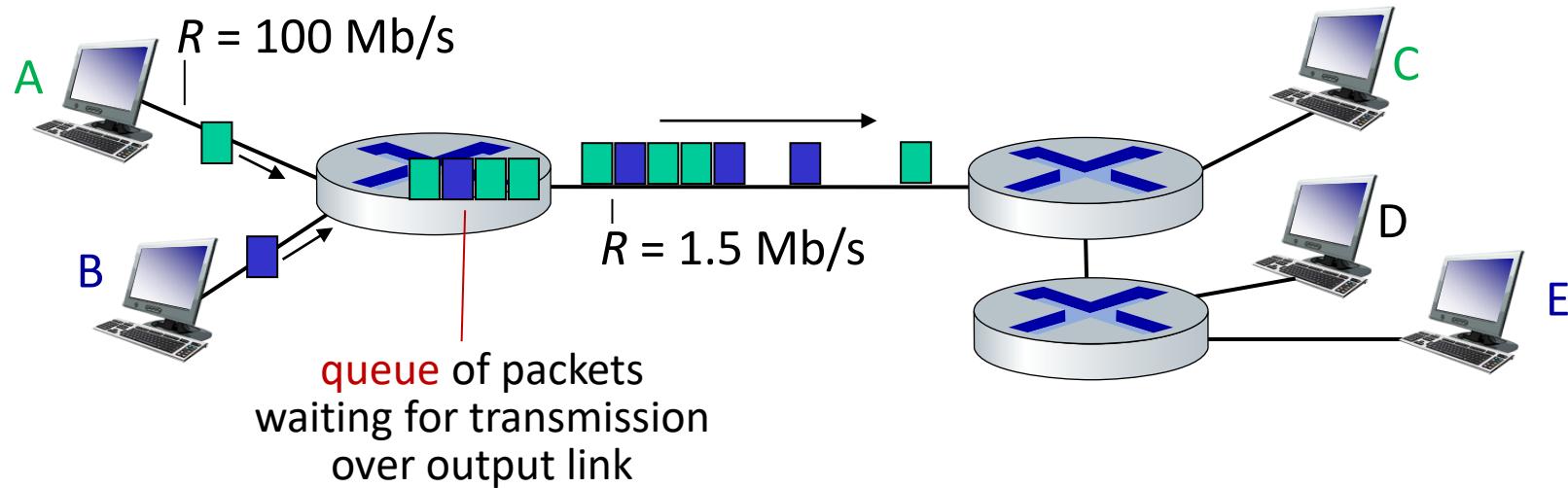


- **packet transmission delay:** takes L/R seconds to transmit (push out) L -bit packet into link at R bps
- ***store and forward:*** entire packet must arrive at router before it can be transmitted on next link
- No resources are reserved. On demand switching.
- Connectionless: No connection state is maintained by the routers in the path
- Best effort delivery and individual packets may follow any path. E.g., Internet

One-hop numerical example:

- $L = 10$ Kbits
- $R = 100$ Mbps
- one-hop transmission delay = 0.1 msec

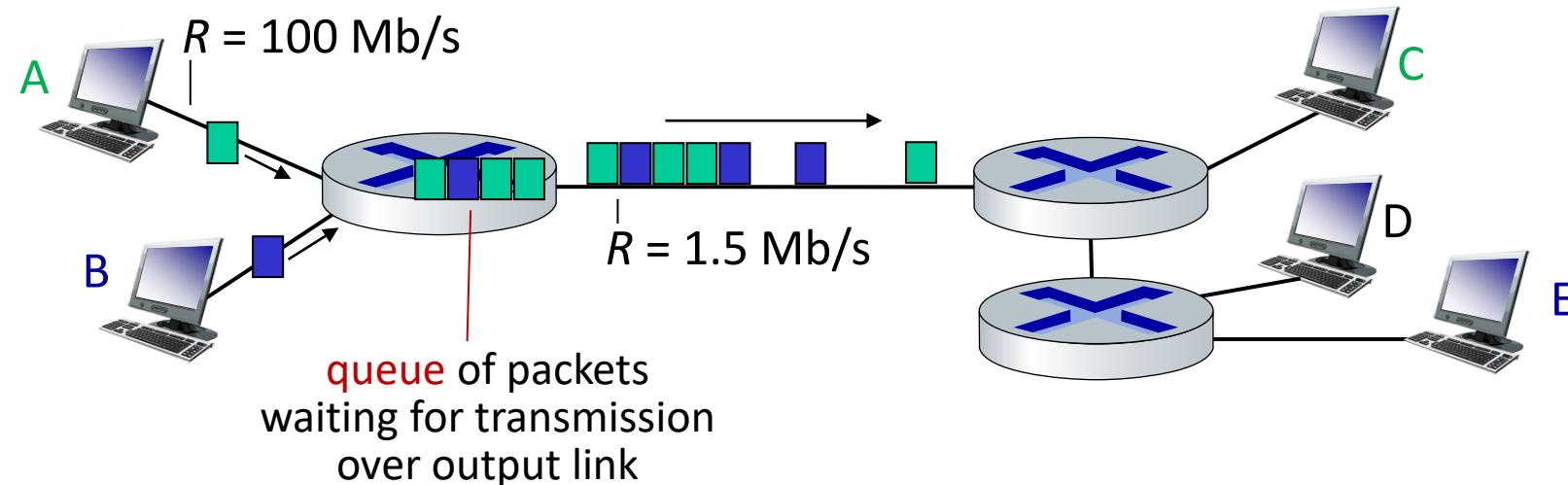
Packet-switching: queueing



Queueing occurs when work arrives faster than it can be serviced:



Packet-switching: queueing



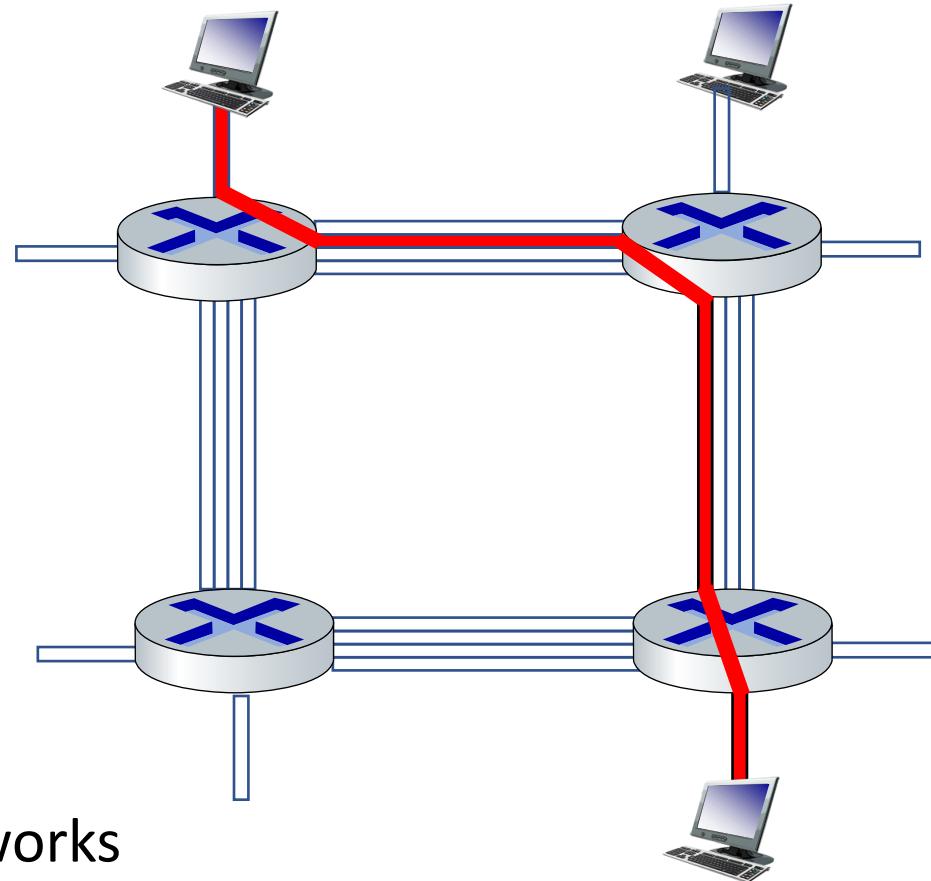
Packet queuing and loss: if arrival rate (in bps) to link exceeds transmission rate (bps) of link for some period of time:

- packets will queue, waiting to be transmitted on output link
- packets can be dropped (lost) if memory (buffer) in router fills up

Alternative to packet switching: circuit switching

end-end resources (buffers, link trans. rate) allocated to, reserved for “call” between source and destination

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



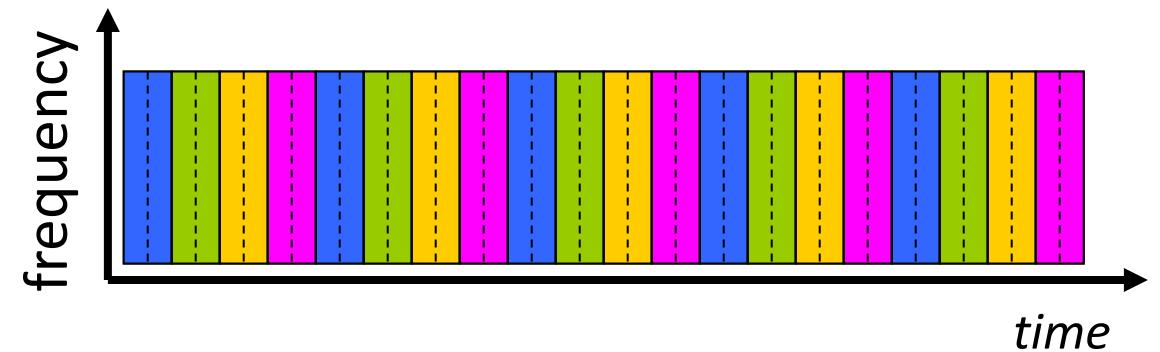
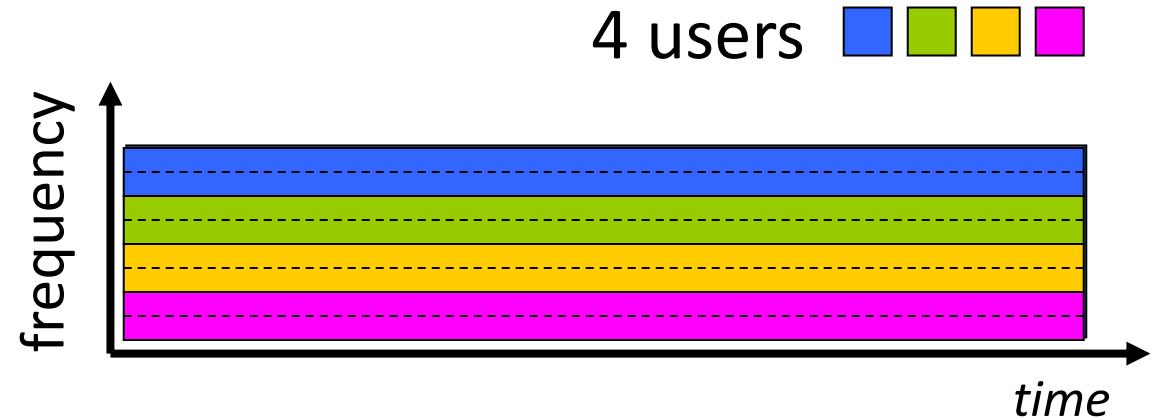
Circuit switching: FDM and TDM

Frequency Division Multiplexing (FDM)

- optical, electromagnetic frequencies divided into (narrow) frequency bands
- each call allocated its own band, can transmit at max rate of that narrow band

Time Division Multiplexing (TDM)

- time divided into slots
- each call allocated periodic slot(s), can transmit at maximum rate of (wider) frequency band (only) during its time slot(s)



Circuit switching (a numerical example)

- For TDM: If the link can transmit/generate 'f' frames per second and each slot consists of 'b' bits, the transmission rate of each circuit is ' $f \cdot b$ ' bits per second.
- How long it takes to send a file of 640,000 bits from Host A to Host B over a circuit-switched network. Suppose that all links in the network use TDM with 24 slots and have a bit rate of 1.536 Mbps. Also suppose that it takes 500 msec to establish an end-to-end circuit before Host A can begin to transmit the file.
- Each circuit has a transmission rate of $(1.536 \text{ Mbps})/24 = 64 \text{ kbps}$,
- so it takes $(640,000 \text{ bits})/(64 \text{ kbps}) = 10 \text{ seconds}$ to transmit the file.
- To this 10 seconds we add the circuit establishment time, giving 10.5 seconds to send the file

Packet switching versus circuit switching

- Advantages of packet switching:
 - Great for ‘bursty’ data – sometimes data to send, but sometimes not.
 - Resource can be shared.
 - Simpler, no call setup.
- Disadvantages:
 - Excessive congestion possible: packet delay and loss due to buffer overflow
 - protocols needed for reliable data transfer (TCP – Unit-3)
 - congestion control (using TCP – Unit-3)
 - No guaranteed service (does not mean its unreliable, it means you may or may not get bandwidth)
 - Out-of-order delivery (reassembly at network layer – Unit-4)
- ***Q: How to provide circuit-like behavior with packet-switching?***
 - “It’s complicated.” We’ll study various techniques that try to make packet switching as “circuit-like” as possible.

Packet switching versus circuit switching

- Advantages of circuit switching:
 - Great for constant traffic
 - Minimal delay and loss
 - Guaranteed service
- Disadvantages:
 - Sharing not possible so limited number of users can have access to link
 - Call setup required which takes some time
 - Resource idle if no data to send. (e.g. caller and callee are not talking and put the phone on hold)

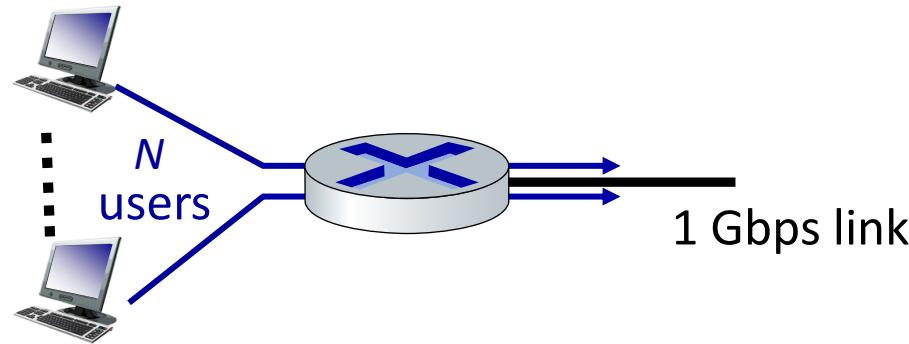
Packet switching versus circuit switching

- Suppose a link capacity is 1Mbps and 10 users are there but only one user generates one thousand 1,000 bits packets, while other users are totally inactive. Under TDM circuit switching, how much time will the active user need to send its data (i.e., one thousand 1,000 bits packets)?
 - Each user will get only 100 Kbps (i.e., the circuit transmission rate).
 - It needs to send 1,000,000 bits so time required = $1,000,000/100,000 = 10$ seconds.
- What about packet switching?
 - Just 1 second

Packet switching versus circuit switching

example:

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



Q: how many users can use this network under circuit-switching and packet switching?

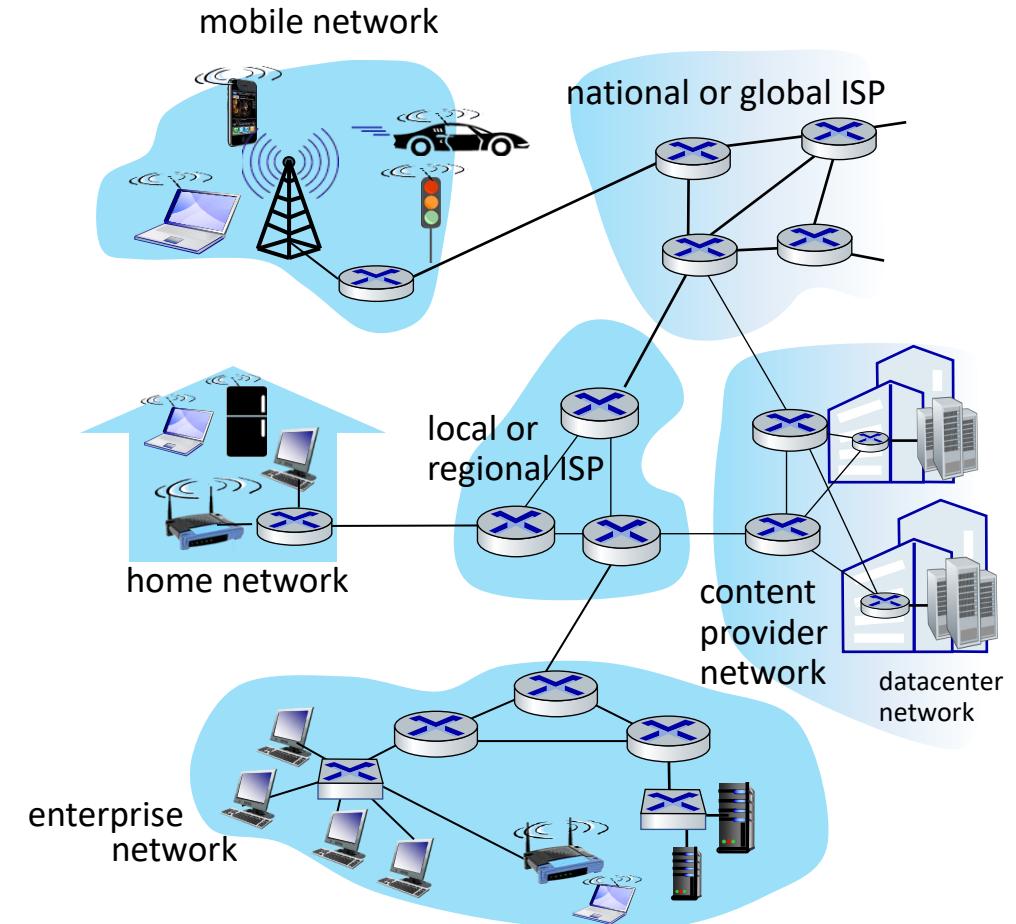
- *circuit-switching:* 10 users
- *packet switching:* with 35 users,
probability > 10 active at same time
is less than .0004 *

Q: how did we get value 0.0004?
A: HW problem (for those with
course in probability only)

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive

Internet structure: a “network of networks”

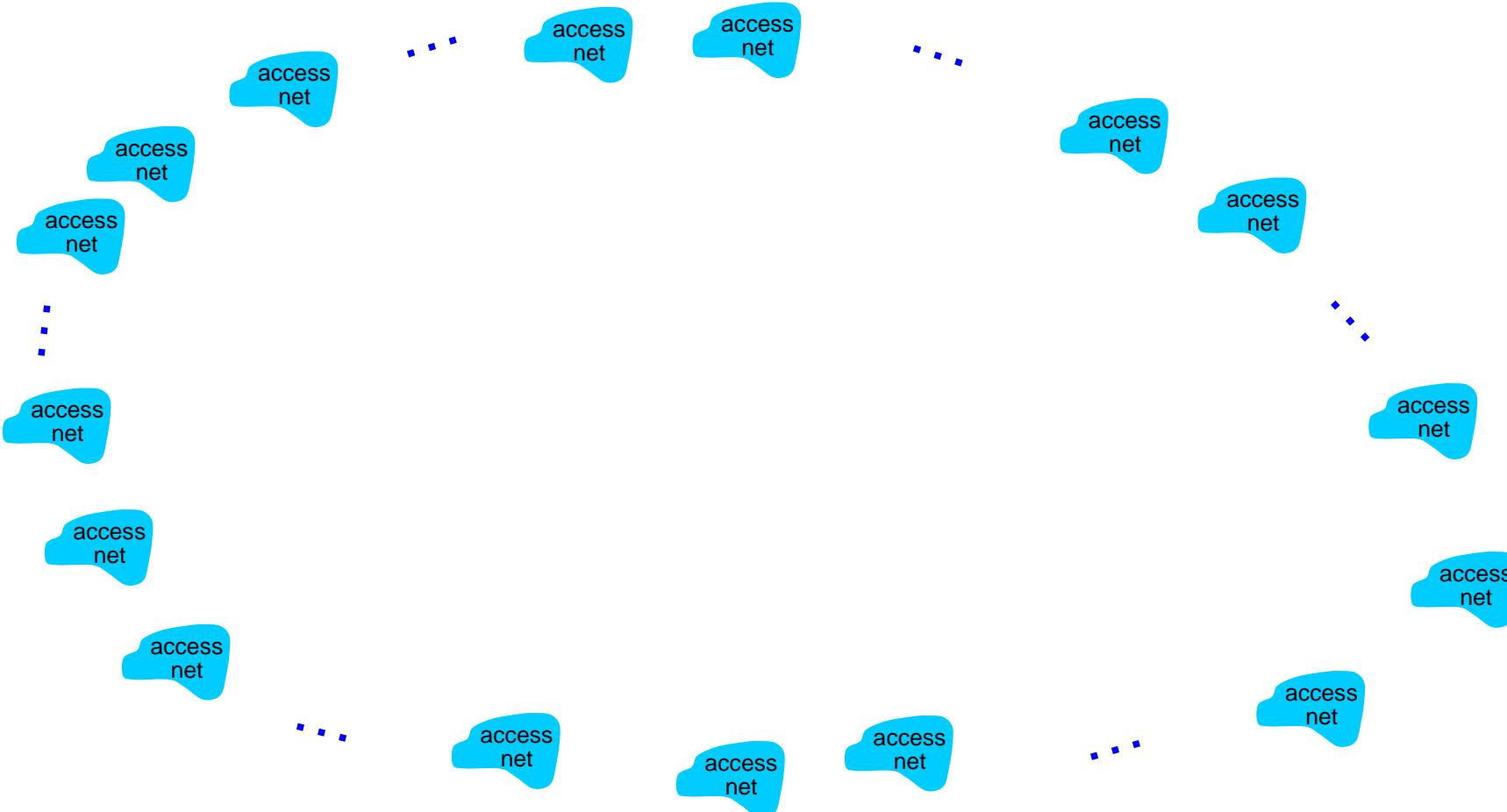
- hosts connect to Internet via **access** Internet Service Providers (ISPs)
- access ISPs in turn must be interconnected
 - so that *any* two hosts (*anywhere!*) can send packets to each other
- resulting network of networks is very complex
 - evolution driven by **economics, national policies**



Let's take a stepwise approach to describe current Internet structure

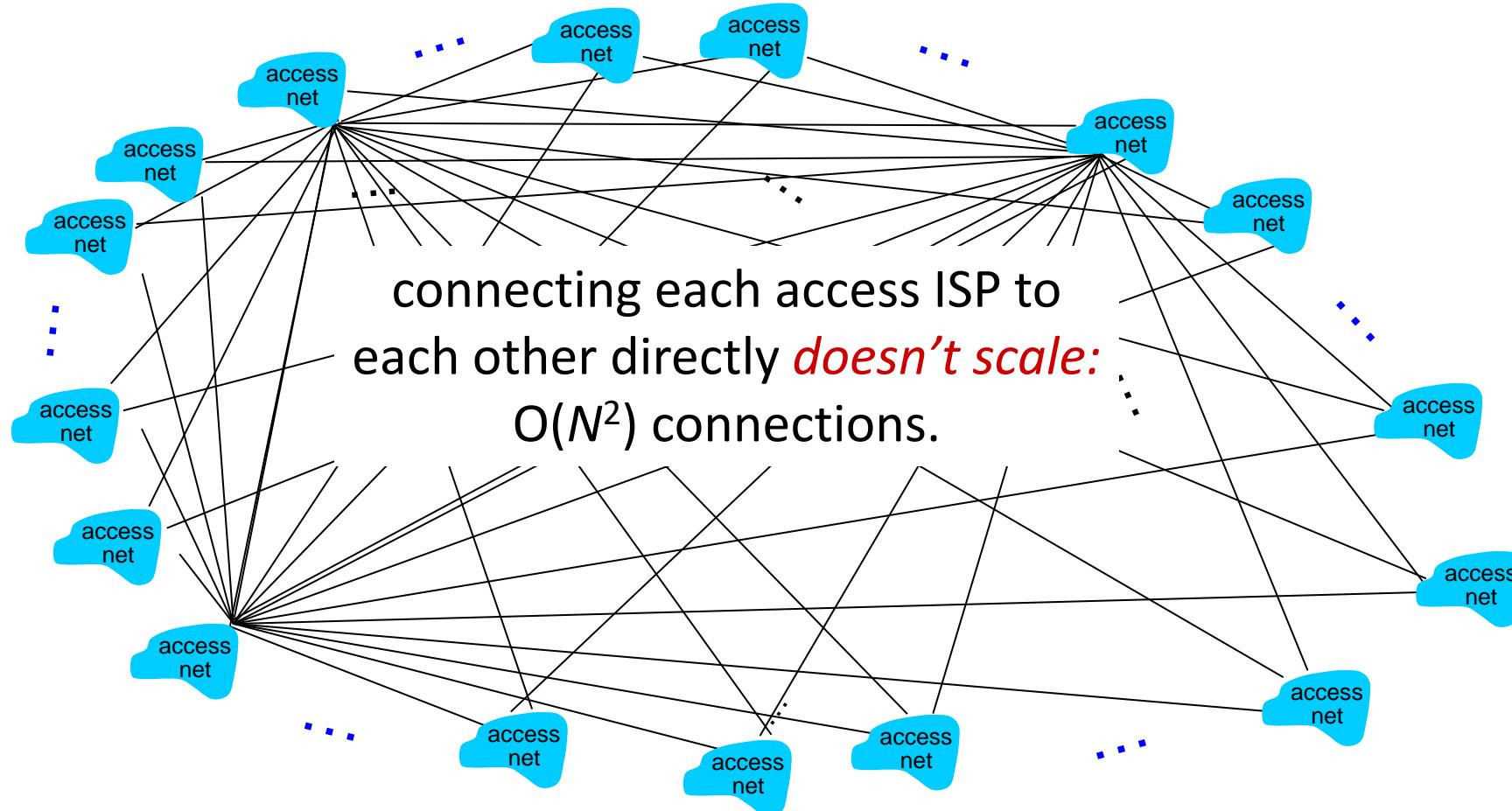
Internet structure: a “network of networks”

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



Internet structure: a “network of networks”

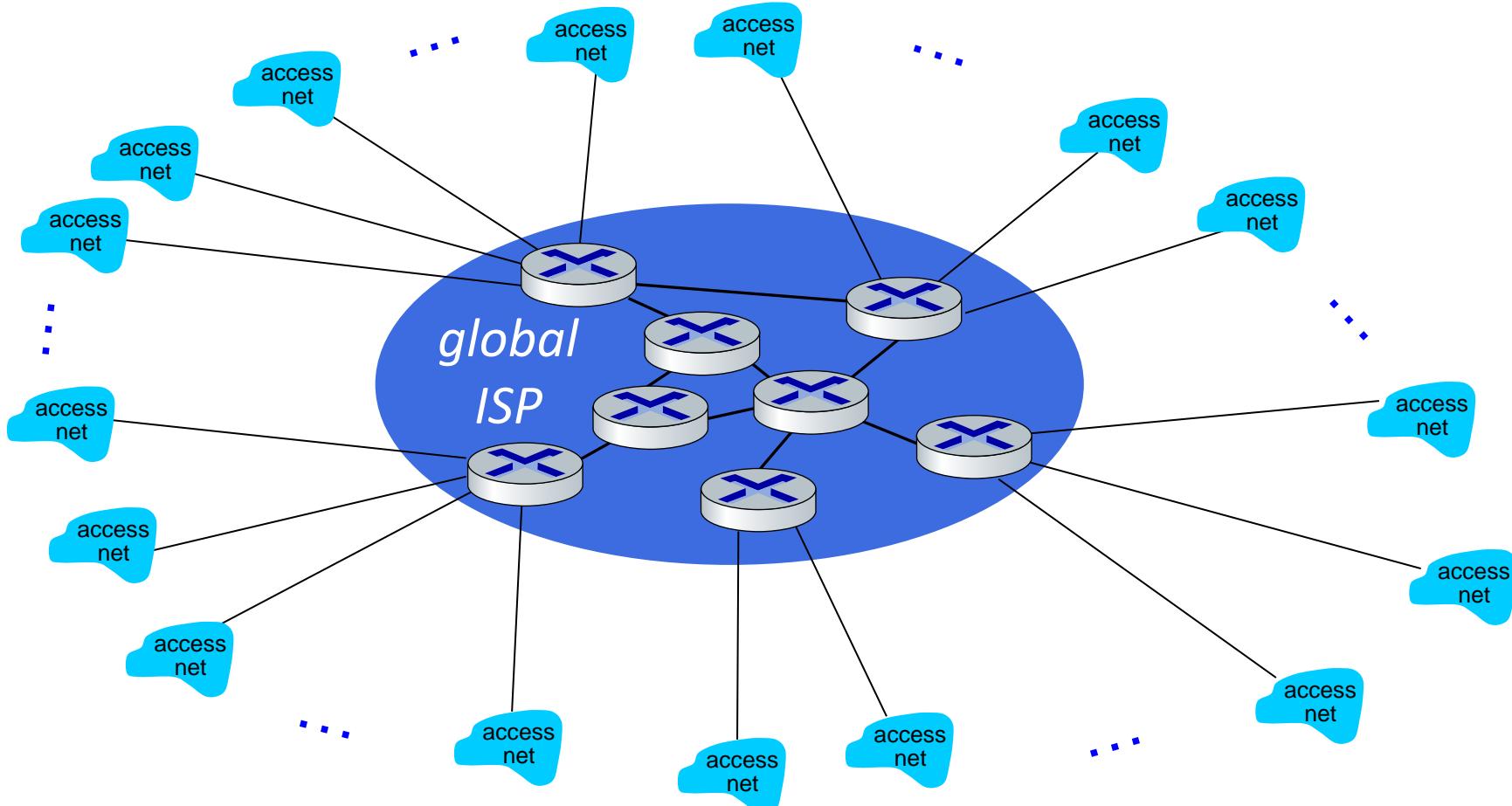
Question: given *millions* of access ISPs, how to connect them together?



Internet structure: a “network of networks”

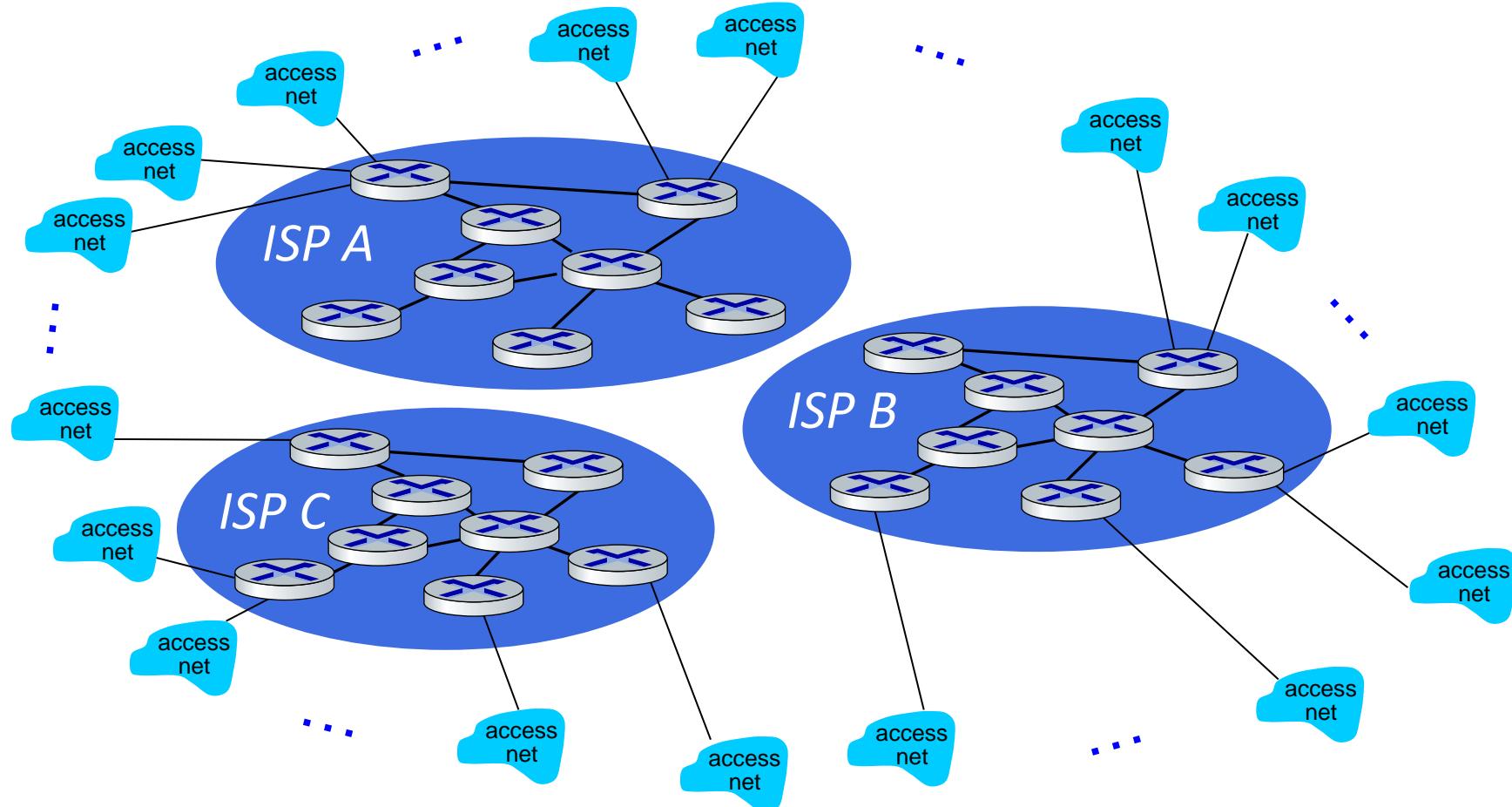
Option: connect each access ISP to one global transit ISP?

Customer and provider ISPs have economic agreement.



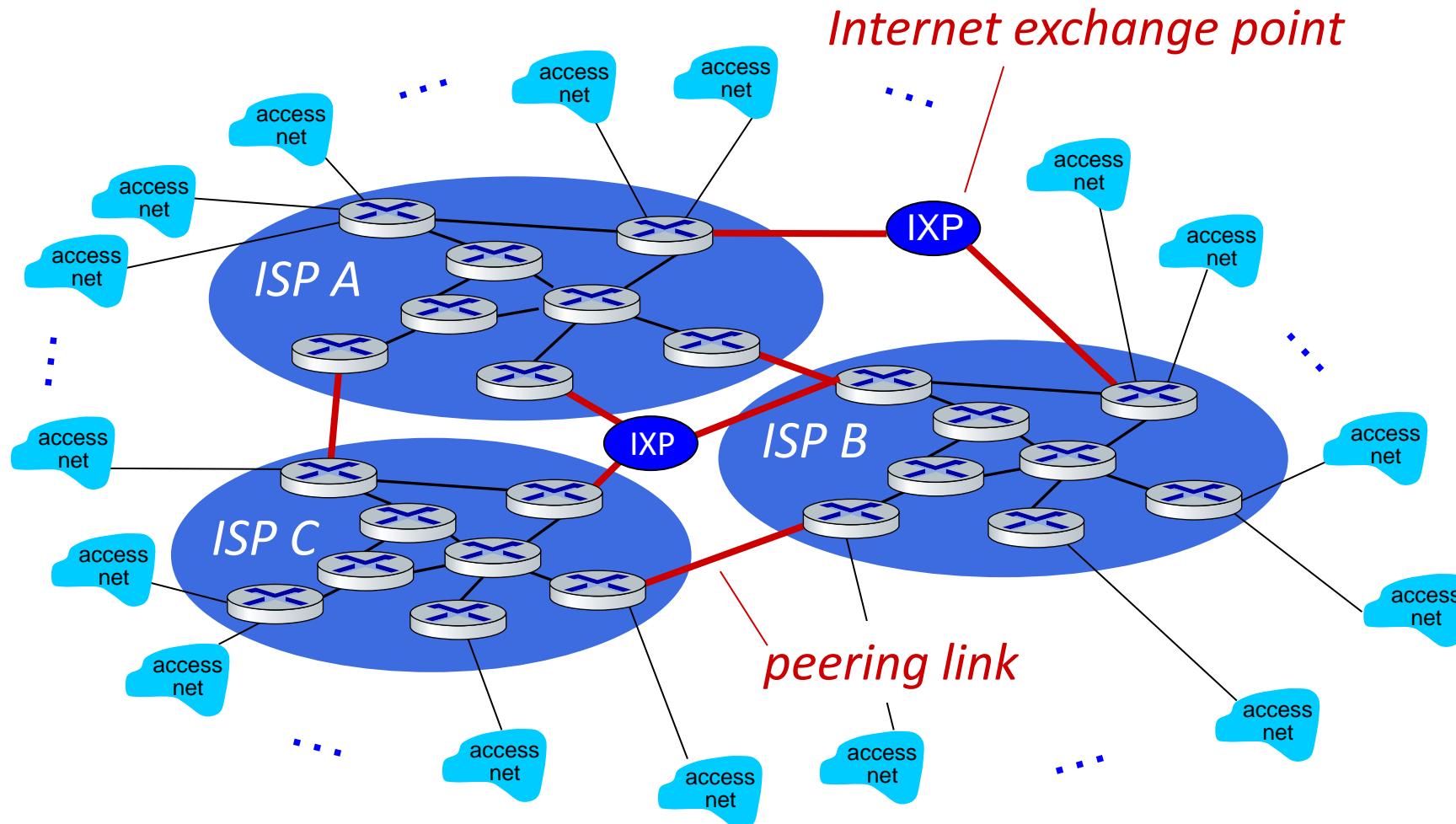
Internet structure: a “network of networks”

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



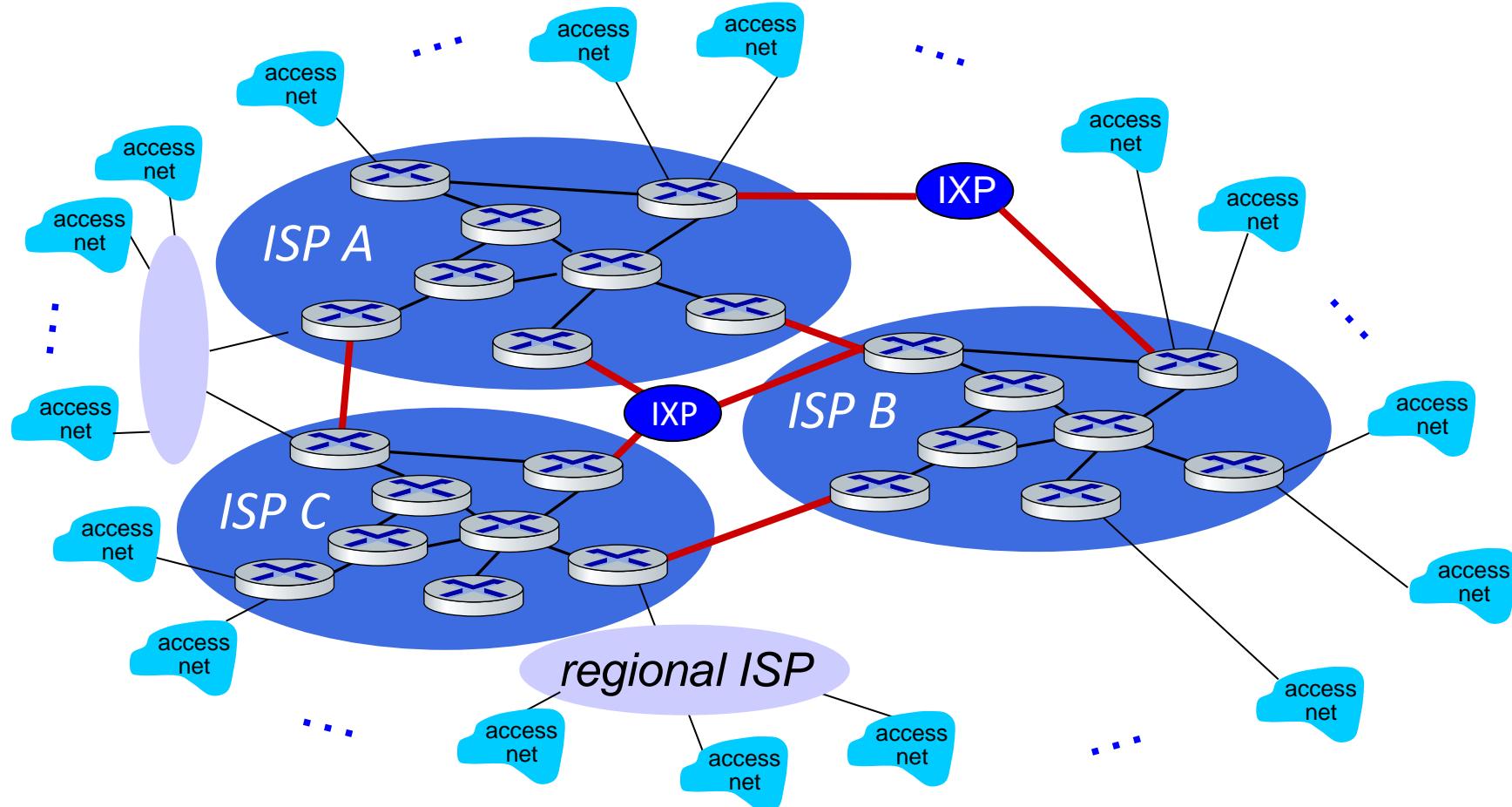
Internet structure: a “network of networks”

But if one global ISP is viable business, there will be competitors ... who will want to be connected



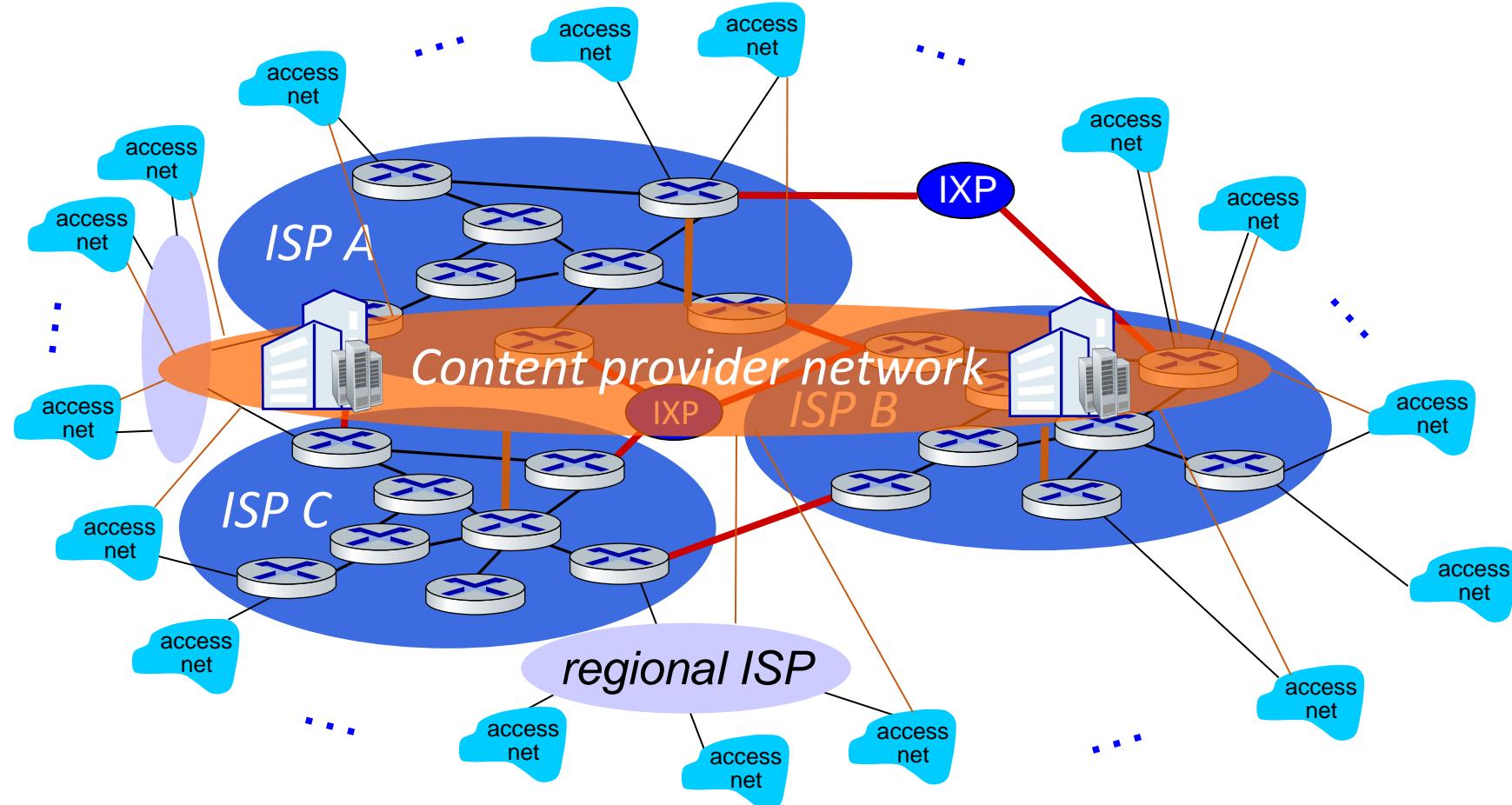
Internet structure: a “network of networks”

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

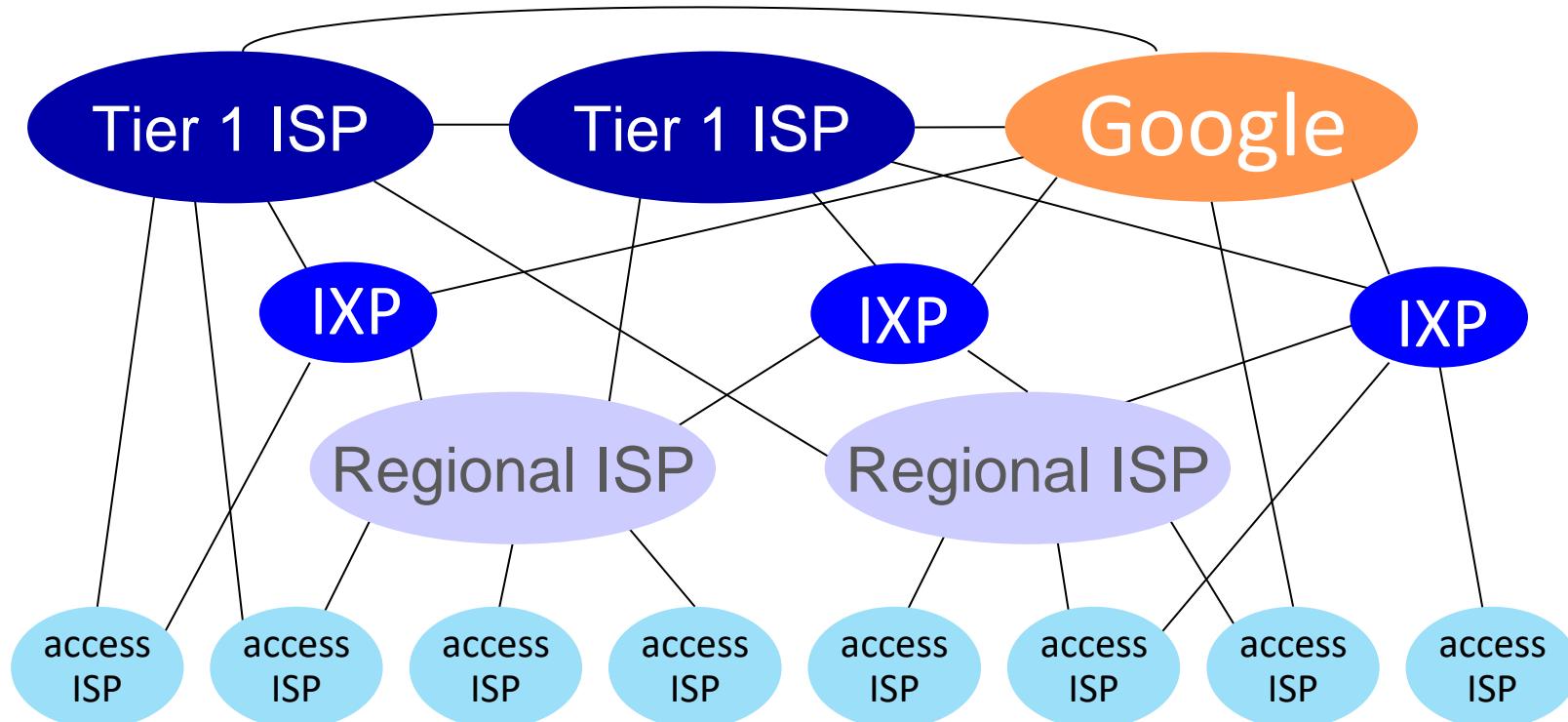


Internet structure: a “network of networks”

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



Internet structure: a “network of networks”



At “center”: small # of well-connected large networks

- **“tier-1” commercial ISPs** (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
- **content provider networks** (e.g., Google, Facebook): private network that connects its data centers to Internet, often bypassing tier-1, regional ISPs

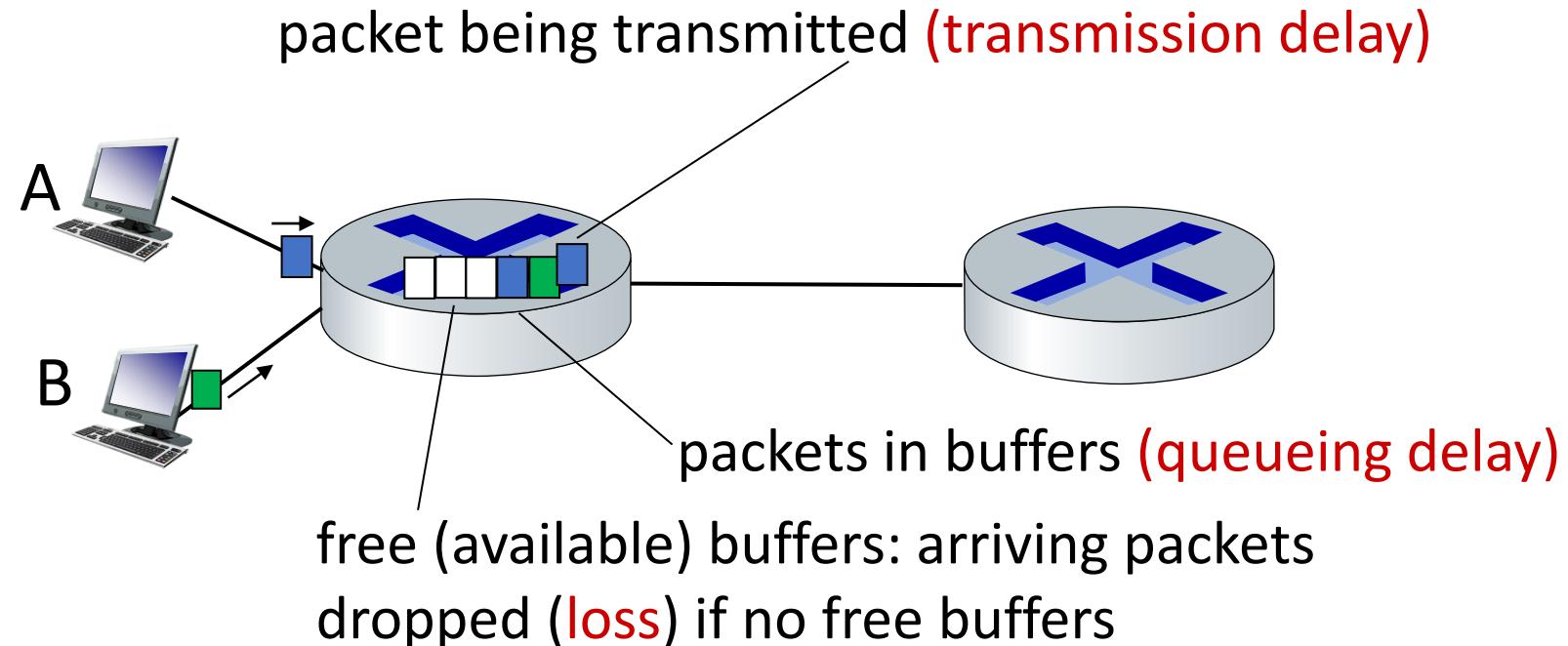
Chapter 1: roadmap

- What *is* the Internet?
- What *is* a protocol?
- Network edge: hosts, access network, physical media
- Network core: packet/circuit switching, internet structure
- **Performance:** loss, delay, throughput
- Security
- Protocol layers, service models
- History

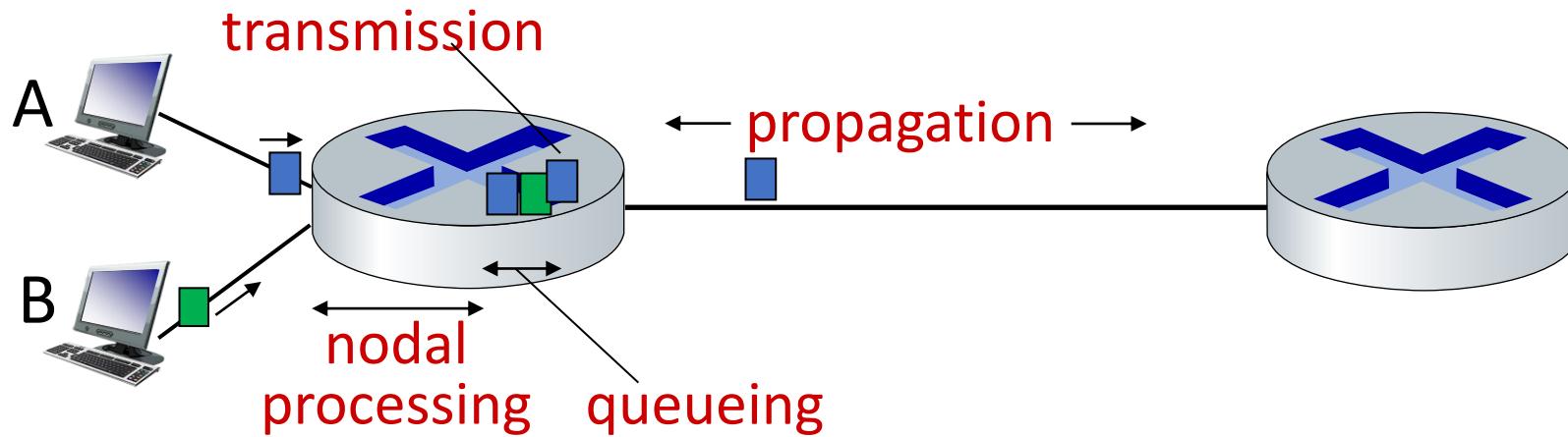


How do packet delay and loss occur?

- packets *queue* in router buffers, waiting for turn for transmission
 - queue length grows when arrival rate to link (temporarily) exceeds output link capacity
- packet *loss* occurs when memory to hold queued packets fills up



Packet delay: four sources



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

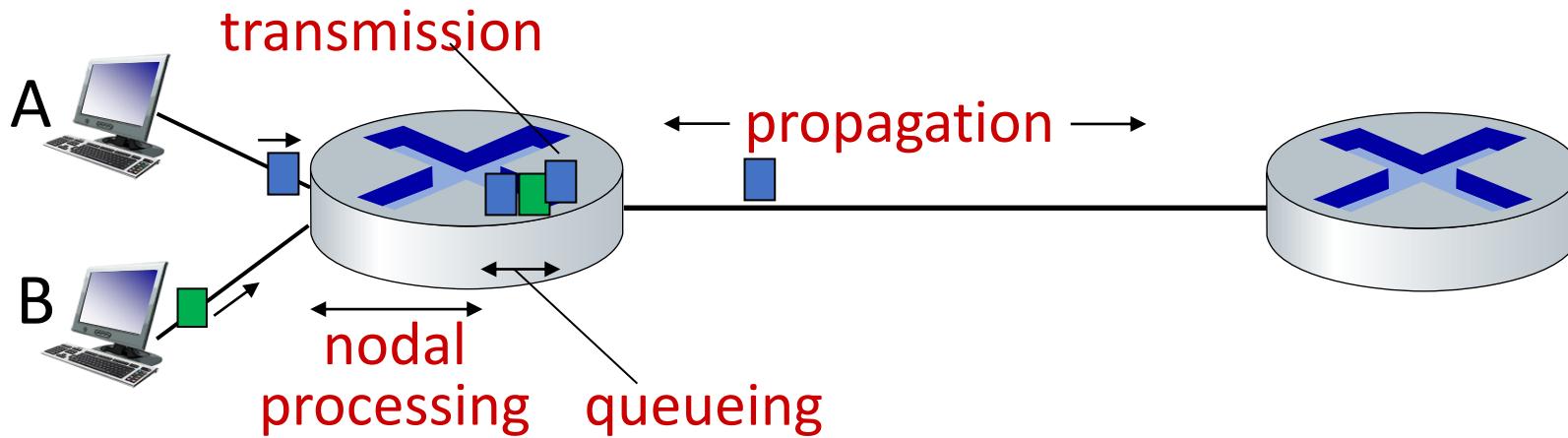
d_{proc} : nodal processing

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

d_{queue} : queueing delay

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

Packet delay: four sources



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

d_{trans} : transmission delay:

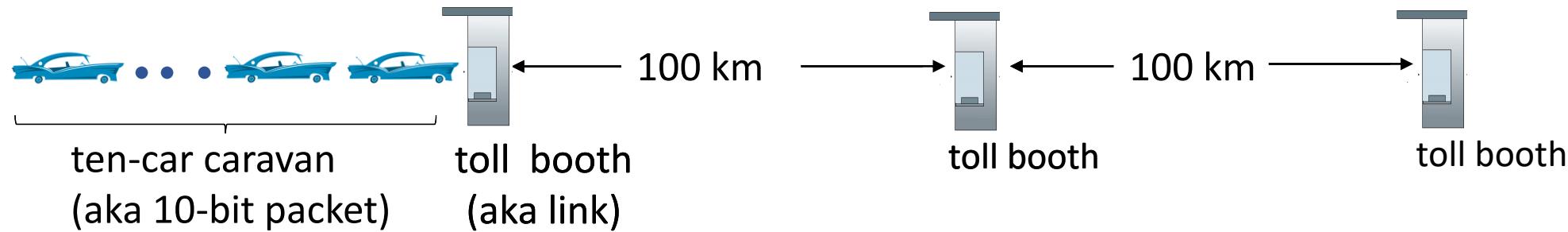
- L : packet length (bits)
- R : link *transmission rate (bps)*
- $d_{\text{trans}} = L/R$

d_{prop} : propagation delay:

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

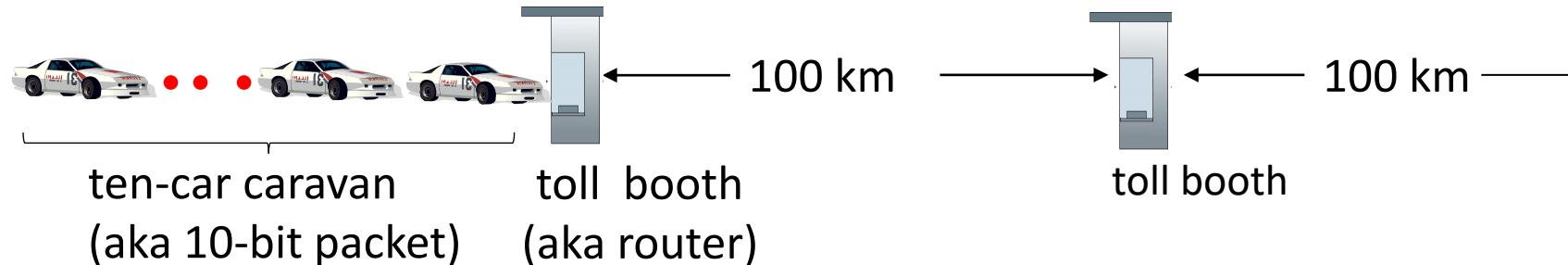
d_{trans} and d_{prop}
very different

Caravan analogy



- car ~ bit; caravan ~ packet; toll service ~ link transmission
- toll booth takes 12 sec to service car (bit transmission time)
- “propagate” at 100 km/hr
- **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: $100\text{km}/(100\text{km/hr}) = 1$ hr
- **A: 62 minutes**

Caravan analogy



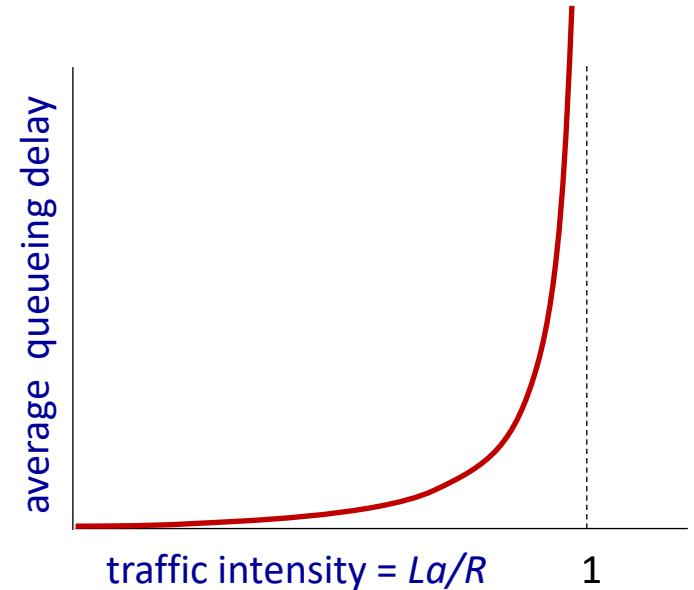
- suppose cars now “propagate” at 1000 km/hr
 - and suppose toll booth now takes one min to service a car
 - ***Q: Will cars arrive to 2nd booth before all cars serviced at first booth?***
- A: Yes!** after 7 min, first car arrives at second booth; three cars still at first booth

Packet queueing delay (revisited)

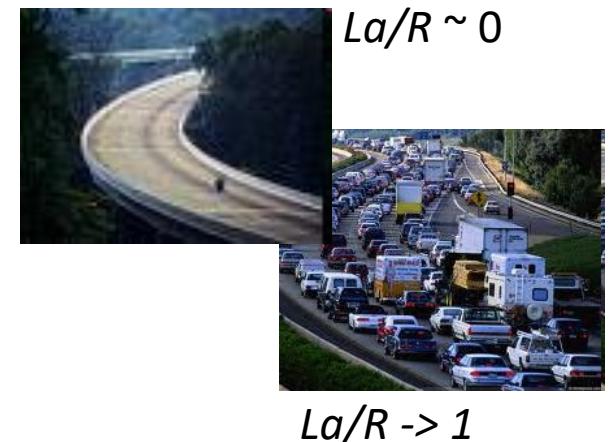
- a : average packet arrival rate
- L : packet length (bits)
- R : link bandwidth (bit transmission rate)

$$\frac{L \cdot a}{R} : \frac{\text{arrival rate of bits}}{\text{service rate of bits}}$$

“traffic intensity”

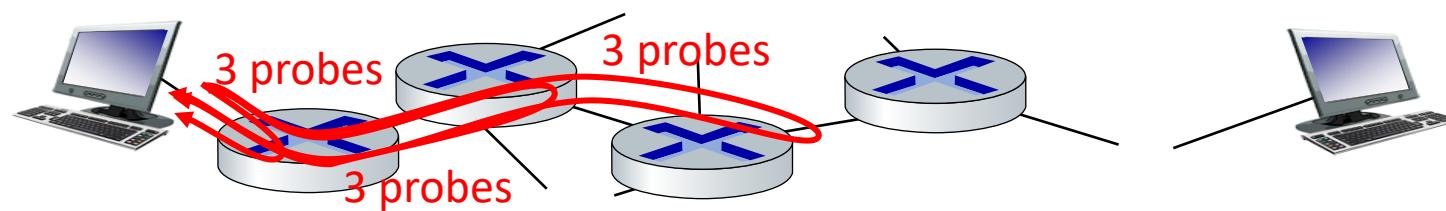


- $La/R \sim 0$: avg. queueing delay small
- $La/R \rightarrow 1$: avg. queueing delay large
- $La/R > 1$: more “work” arriving is more than can be serviced - average delay infinite!



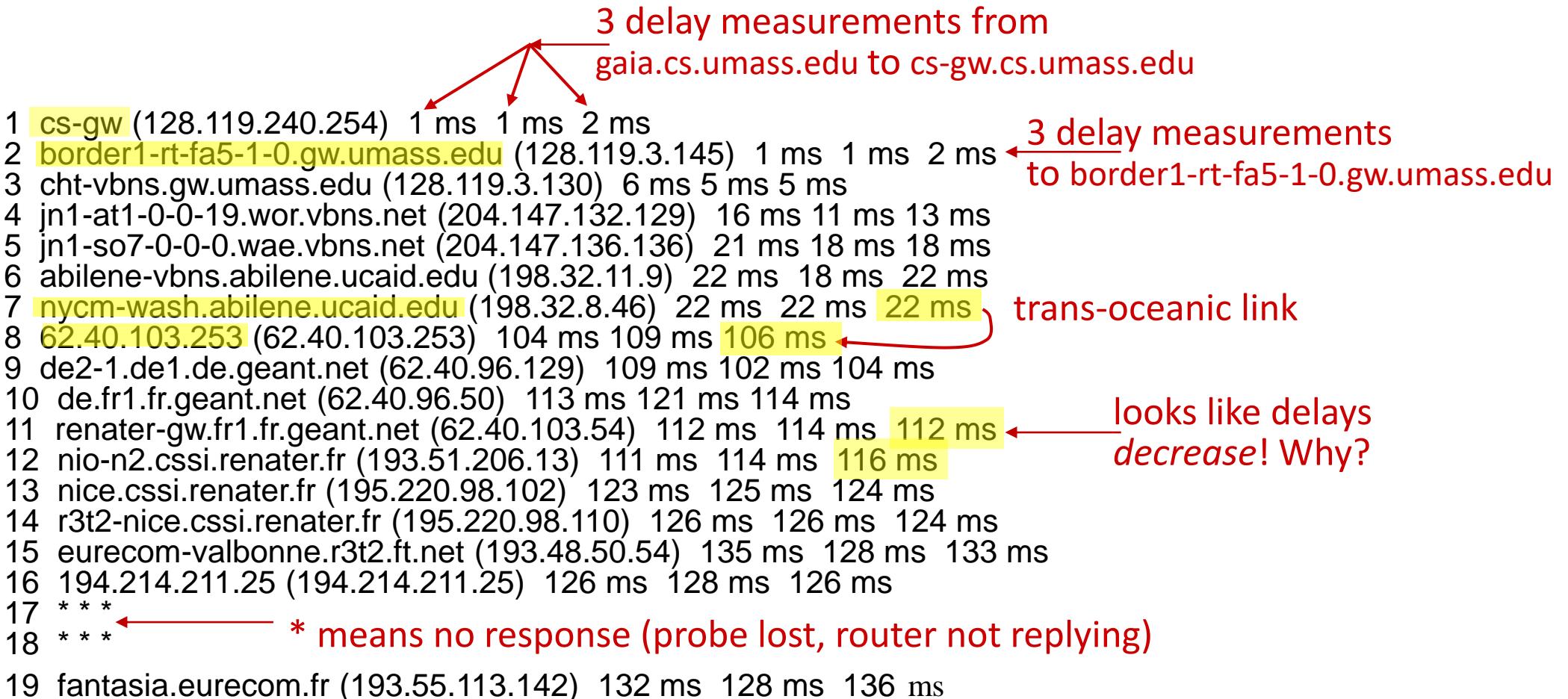
“Real” Internet delays and routes

- 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 (with time-to-live field value of i)
 - router i will return packets to sender
 - sender measures time interval between transmission and reply



Real Internet delays and routes

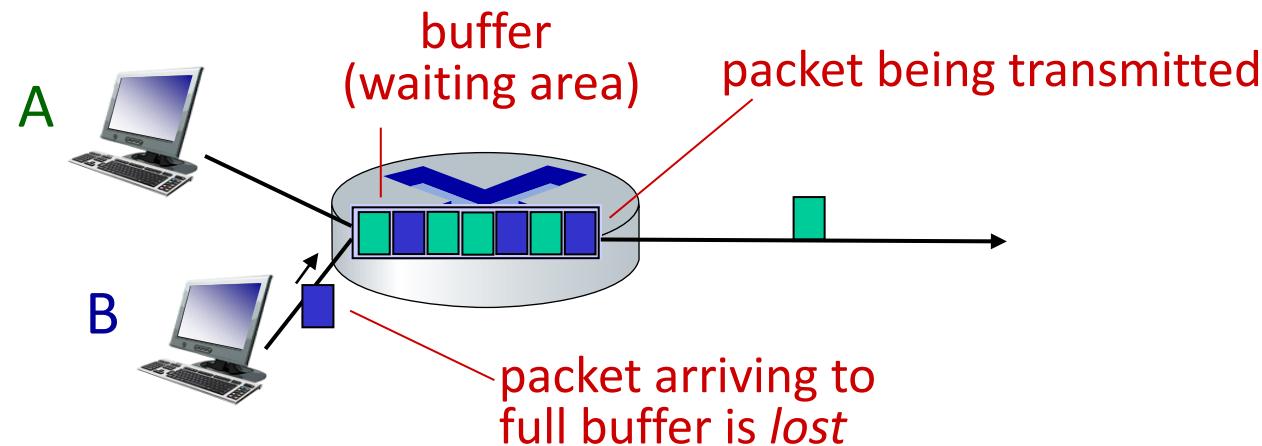
traceroute: gaia.cs.umass.edu to www.eurecom.fr



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

Packet loss

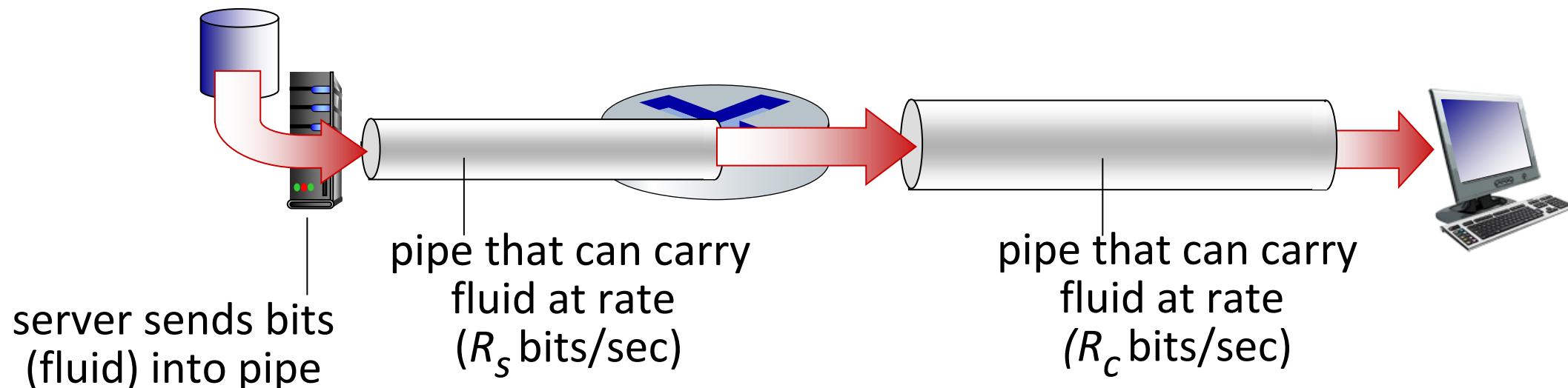
- queue (aka buffer) preceding link in buffer has finite capacity
- packet arriving to full queue dropped (aka lost)
- lost packet may be retransmitted by previous node, by source end system, or not at all



* Check out the Java applet for an interactive animation (on publisher's website) of queuing and loss

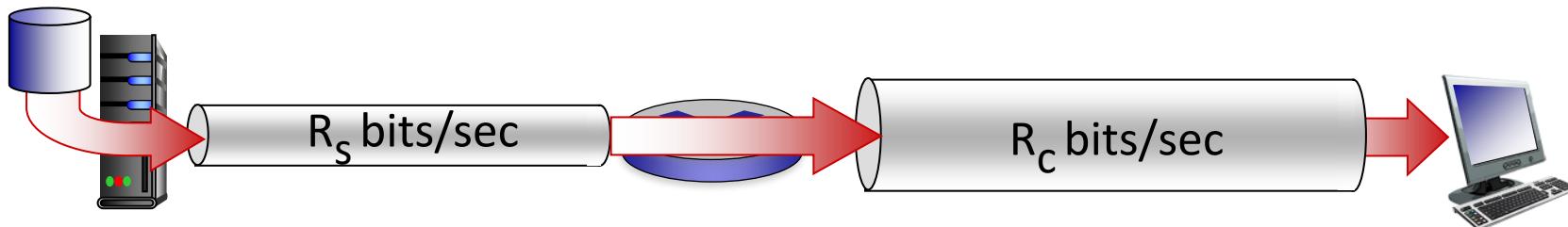
Throughput

- *throughput*: rate (bits/time unit) at which bits are being sent from sender to receiver
 - *instantaneous*: rate at given point in time
 - *average*: rate over longer period of time

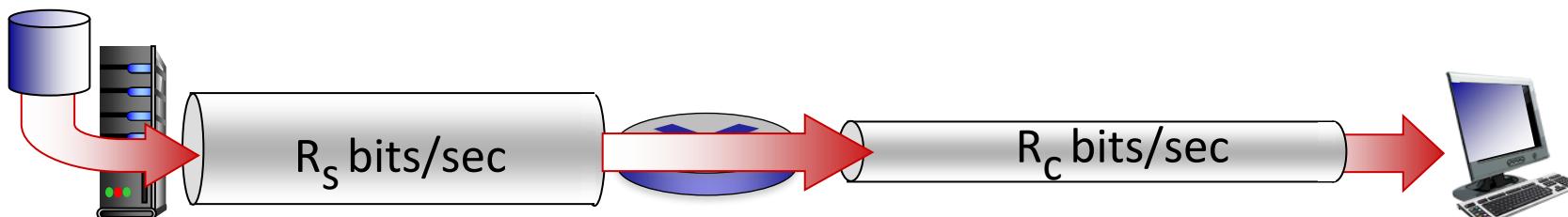


Throughput

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



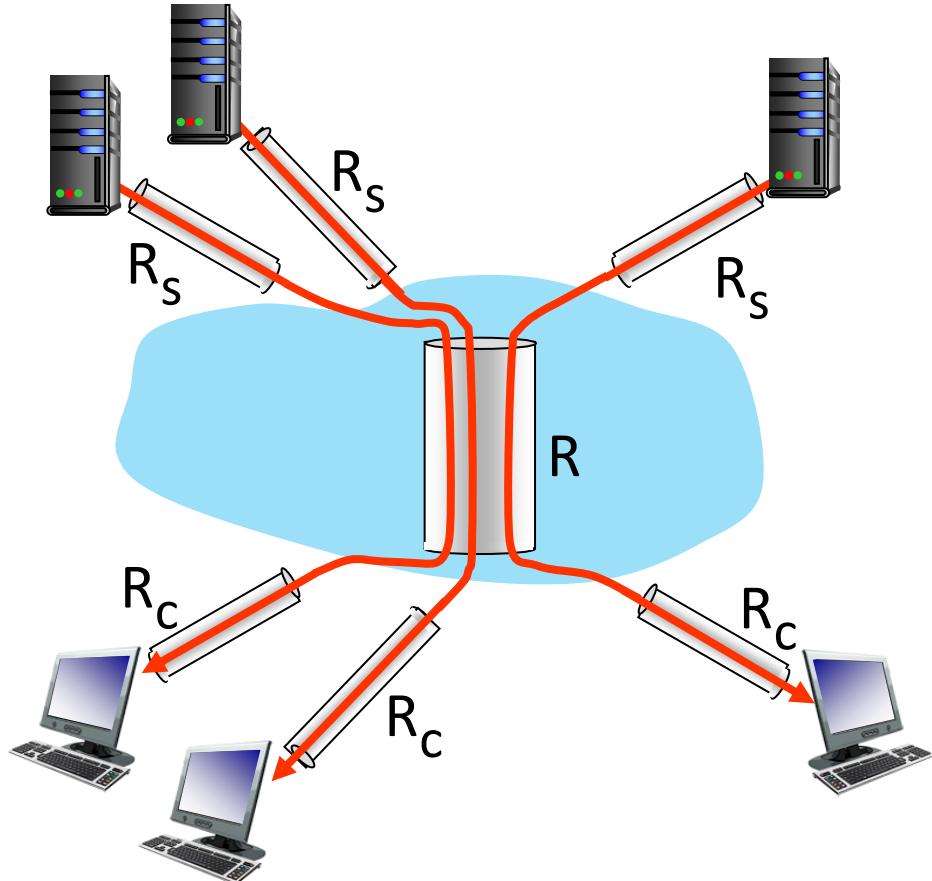
$R_s > R_c$ What is average end-end throughput?



bottleneck link

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

Throughput: network scenario



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

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/

Chapter 1: roadmap

- What *is* the Internet?
- What *is* a protocol?
- Network edge: hosts, access network, physical media
- Network core: packet/circuit switching, internet structure
- Performance: loss, delay, throughput
- Security
- **Protocol layers, service models**
- History



Protocol “layers” and reference models

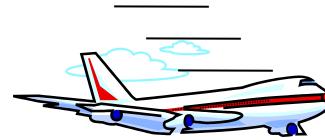
Networks are complex,
with many “pieces”:

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

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

- and/or our *discussion*
of networks?

Example: organization of air travel



— *end-to-end transfer of person plus baggage* —→

ticket (purchase)
baggage (check)
gates (load)
runway takeoff
airplane routing

ticket (complain)
baggage (claim)
gates (unload)
runway landing
airplane routing

airplane routing

How would you *define/discuss* the *system* of airline travel?

- a series of steps, involving many services

Example: organization of air travel



layers: each layer implements a service

- via its own internal-layer actions (e.g., gate layer- loading and unloading)
- relying on services provided by layer below (e.g., takeoff/landing layer - runway-to-runway passenger transfer service)

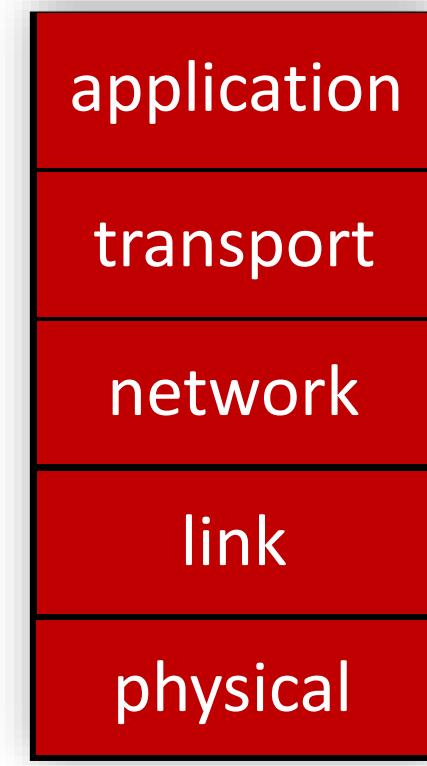
Why layering?

Approach to designing/discussing complex systems:

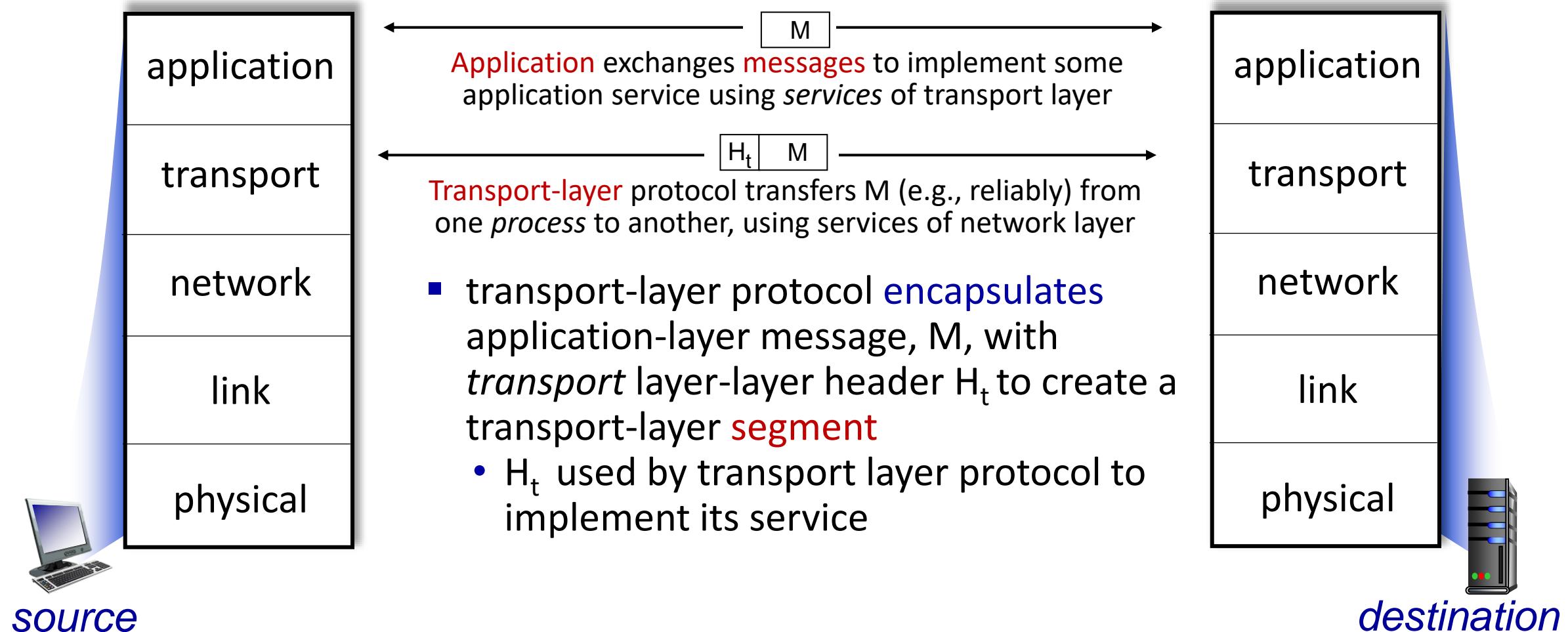
- explicit structure allows identification, relationship of system's pieces
 - layered *reference model* for discussion
- modularization eases maintenance, updating of system
 - change in layer's service *implementation*: transparent to rest of system
 - e.g., change in gate procedure doesn't affect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

Layered Internet protocol stack

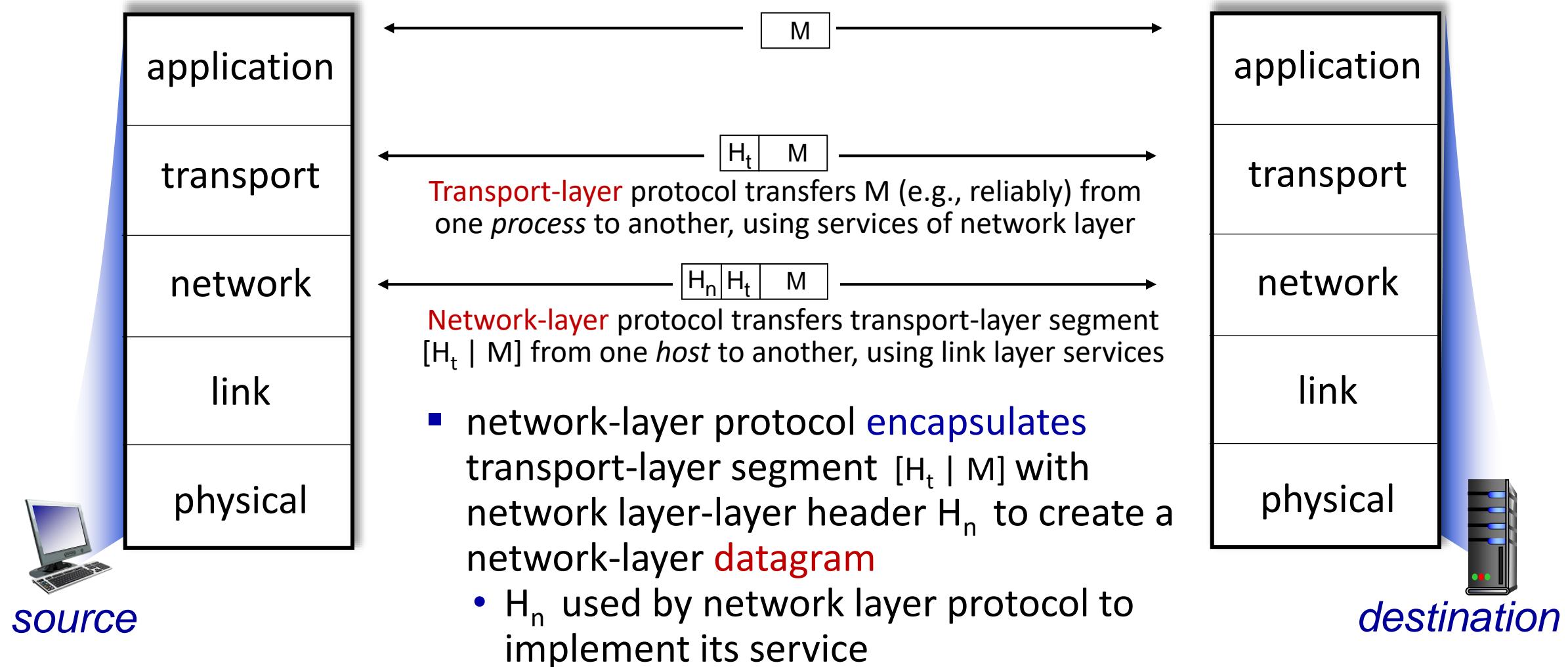
- *application*: supporting network applications
 - HTTP, IMAP, SMTP, DNS
- *transport*: process-process data transfer
 - TCP, UDP
- *network*: routing of datagrams from source to destination
 - IP, routing protocols
- *link*: data transfer between neighboring network elements
 - Ethernet, 802.11 (WiFi),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



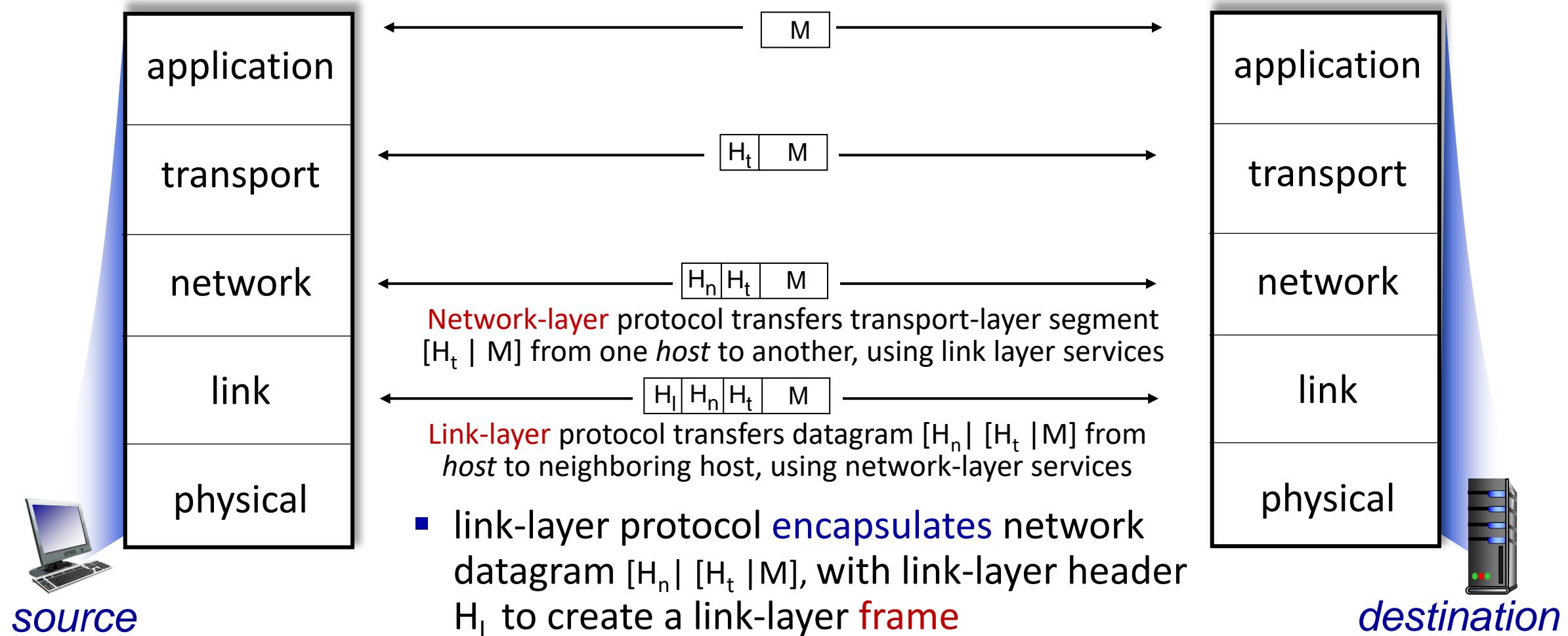
Services, Layering and Encapsulation



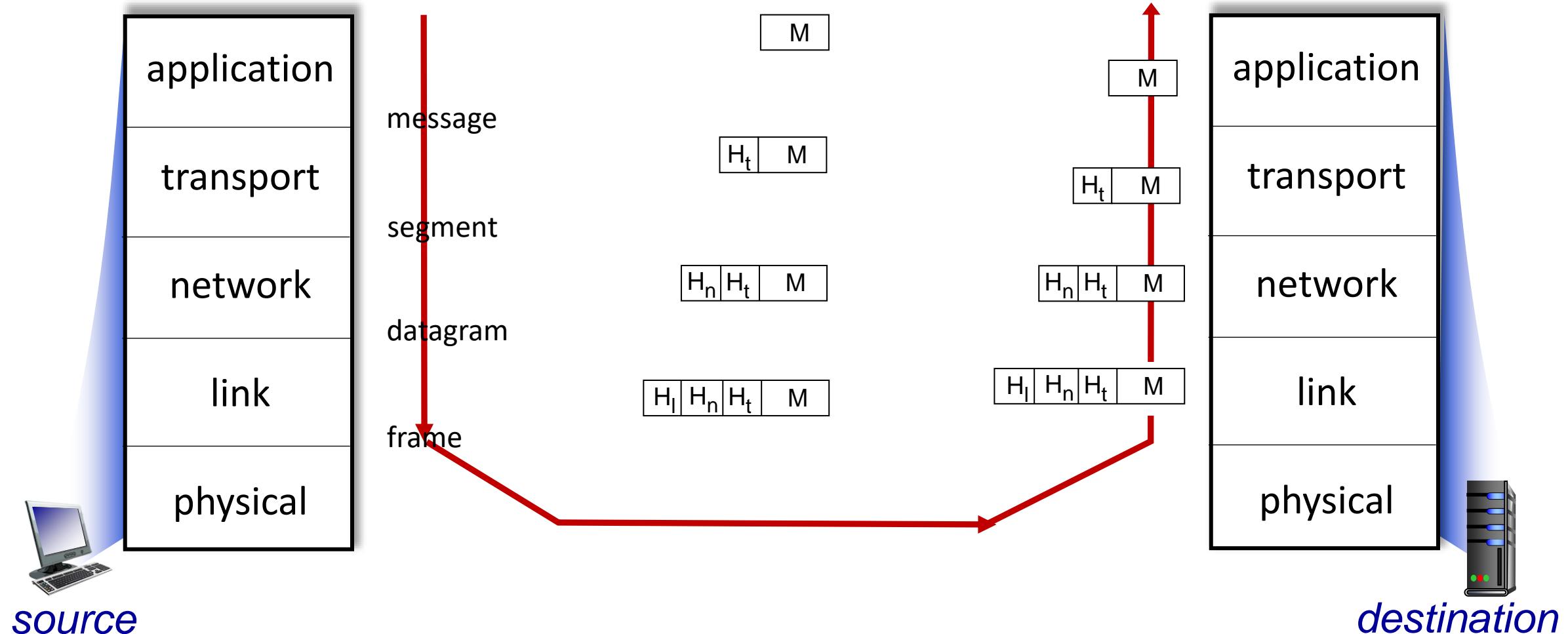
Services, Layering and Encapsulation



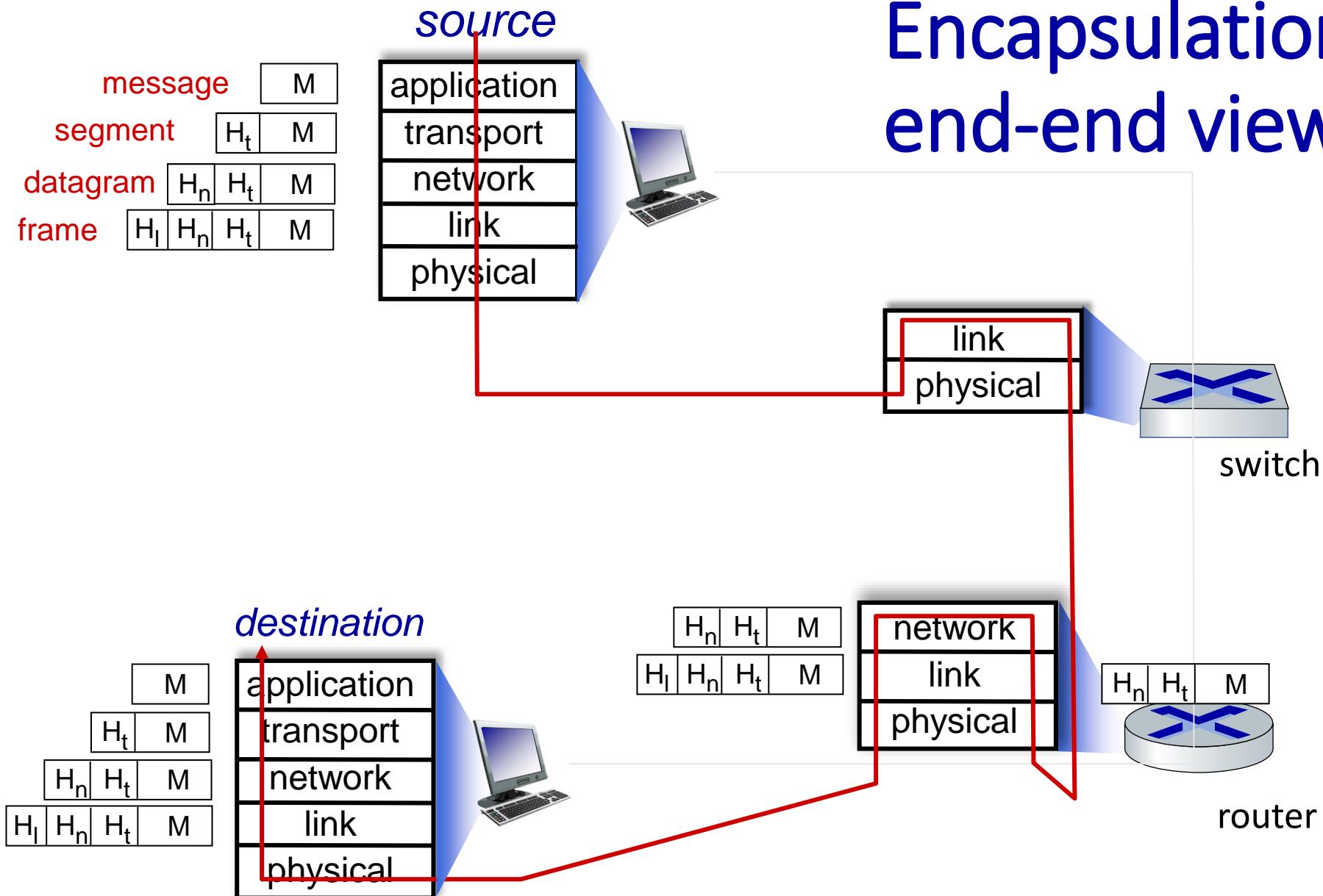
Services, Layering and Encapsulation



Services, Layering and Encapsulation



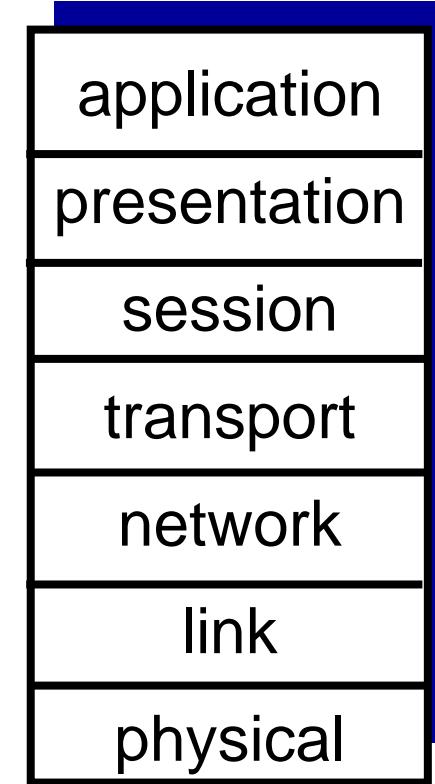
Encapsulation: an end-end view



ISO/OSI reference model

Two layers not found in Internet protocol stack!

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



The seven layer OSI/ISO reference model

Chapter 1: summary

We've covered a "ton" of material!

- Internet overview
- what's a protocol?
- network edge, access network, core
 - packet-switching versus circuit-switching
 - Internet structure
- performance: loss, delay, throughput
- layering, service models
- Security (left for your specialization ☺)

You now have:

- context, overview, vocabulary, "feel" of networking
- more depth, detail, *and fun* to follow!

Some numerical Problems

1. Suppose that a user at a PC at one end of Canada sends a 1-Mbit file to a remote server on the other end over a data link operating at 64 kbps. Assume that we are using a fiber optic link with a propagation rate of the speed of light, approximately $3 * 10^8$ m/sec, and that the distance is 4800 km.
 - (a) What is the overall network delay, i.e. time to transmit the file? (Ignore any processing or queueing delays.)

Some numerical Problems

$$d_{\text{total}} = d_{\text{propagation}} + d_{\text{transmission}}$$

$$d_{\text{propagation}} = \text{distance/speed} = 4800 * 10^3 / 3 * 10^8 = 0.016 \text{ seconds}$$

$$d_{\text{transmission}} = \text{length/rate} = 10^6 / 64 * 10^3 = 15.625 \text{ seconds}$$

Some numerical Problems

2. Suppose you are downloading an MP3 file of 64 MB, the server has a transmission rate of $R_s = 4 \text{ Mbps}$, and you have an access link of $R_c = 2 \text{ Mbps}$. What is the throughput ? What is the overall network delay, i.e. time to transmit the file? (Ignore any processing or queueing delays.)

Throughput = $\min(R_s, R_c) = 2 \text{ Mbps}$

Time to transmit the file = $64 * 8 / 2 = 256 \text{ seconds}$

Some numerical Problems

3. What is the propagation time if the distance between the two points is 12,000 km? Assume the propagation speed to be 2.4×10^8 m/s in cable.

$$\text{Propagation time} = 12000 * 1000 / 2.4 * 10^8 = 50 \text{ ms}$$

Some numerical Problems

4. What are the propagation time and the transmission time for a 2.5-kbyte message (an e-mail) if the bandwidth of the network is 1 Gbps? Assume that the distance between the sender and the receiver is 12,000 km and that light travels at 2.4×10^8 m/s.

$$\text{Propagation time} = 12000 * 1000 / 2.4 * 10^8 = 50 \text{ ms}$$

$$\text{Transmission time} = 2500 * 8 / 10^9 = 0.02 \text{ ms}$$

Message is short and the bandwidth is high, the dominant factor is the propagation time, not the transmission time. The transmission time can be ignored.

Some numerical Problems

5. What are the propagation time and the transmission time for a 5 MB image if the bandwidth of the network is 1 Mbps? Assume that the distance between the sender and the receiver is 12,000 km and that light travels at 2.4×10^8 m/s.

$$\text{Propagation time} = 12000 * 1000 / 2.4 * 10^8 = 50 \text{ ms}$$

$$\text{Transmission time} = 5,000,000 * 8 / 10^6 = 40 \text{ s}$$

Message is long and the bandwidth is less, the dominant factor is the transmission time, not the propagation time. The propagation time can be ignored.

Chapter 2

Application Layer

A note on the use of these PowerPoint slides:

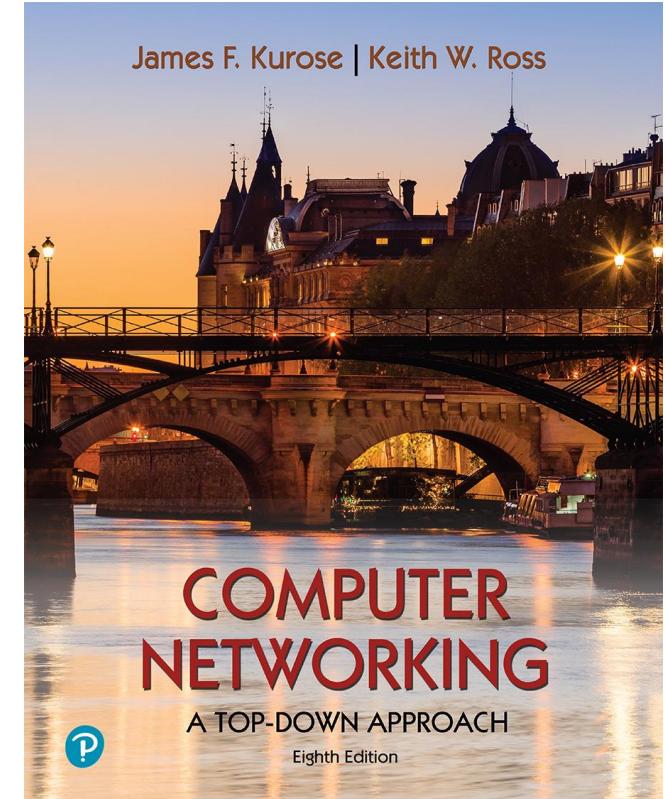
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use,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

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

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A
Top-Down Approach*
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Application layer: overview

Our goals:

- conceptual *and* implementation aspects of application-layer protocols
 - transport-layer service models
 - client-server paradigm
 - peer-to-peer paradigm
- learn about protocols by examining popular application-layer protocols and infrastructure
 - HTTP
 - SMTP, IMAP
 - DNS
 - video streaming systems, CDNs
- programming network applications
 - socket API

Some network apps

- social networking
- Web
- text messaging
- e-mail
- multi-user network games
- streaming stored video
(YouTube, Hulu, Netflix)
- P2P file sharing
- voice over IP (e.g., Skype)
- real-time video conferencing
(e.g., Zoom)
- Internet search
- remote login
- ...

Q: your favorites?

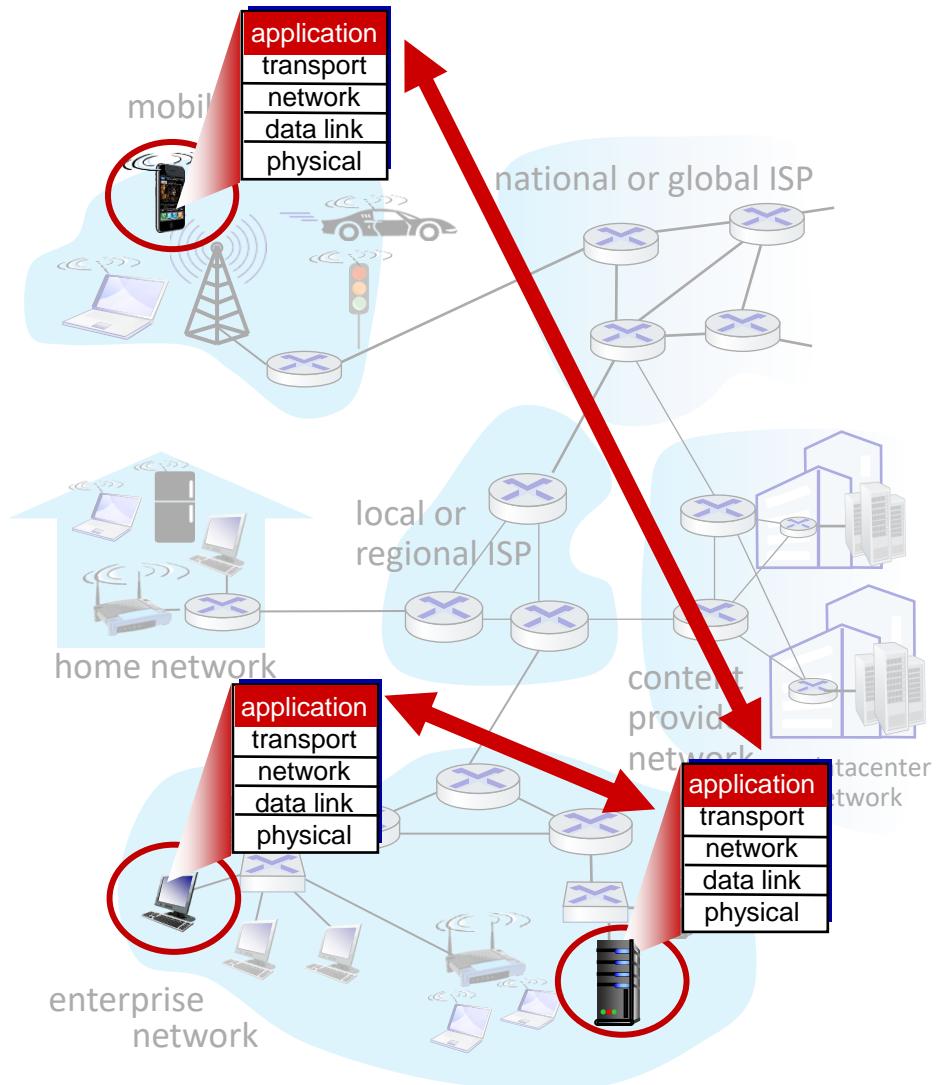
Creating a network app

write programs that:

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

no need to write software for
network-core devices

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



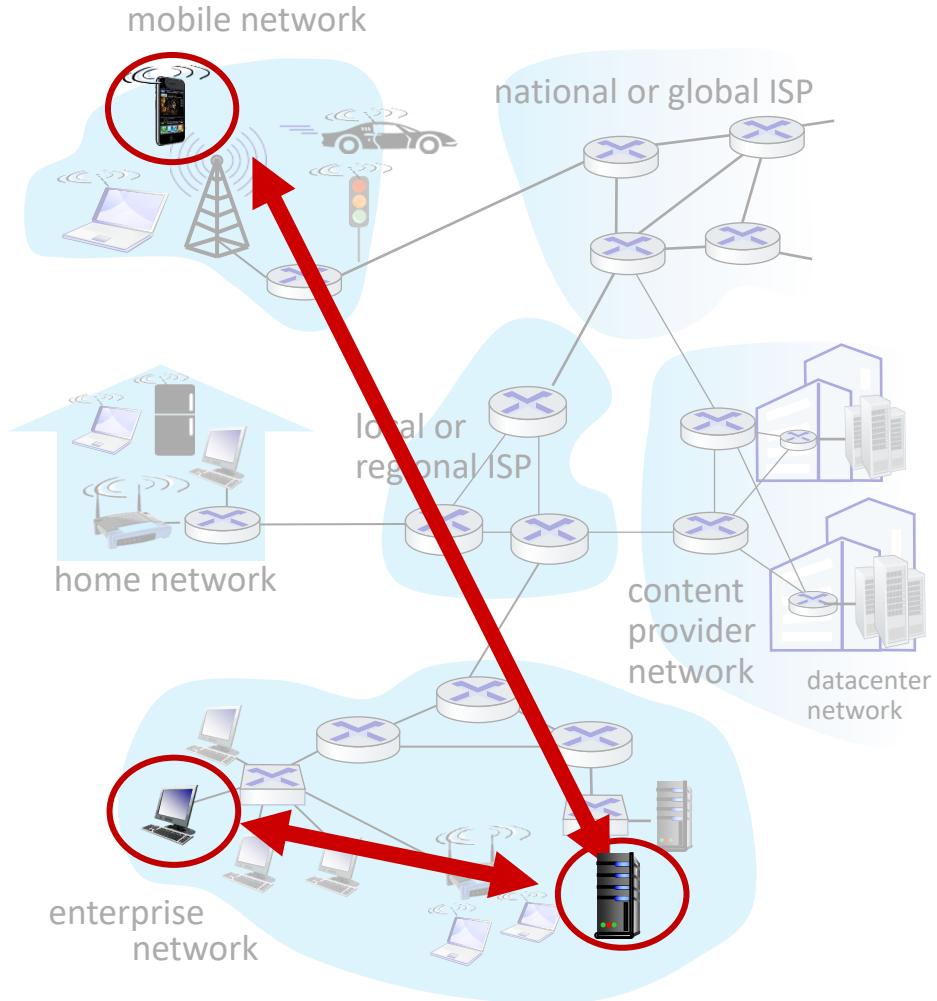
Client-server paradigm

server:

- always-on host
- permanent IP address
- often in data centers, for scaling

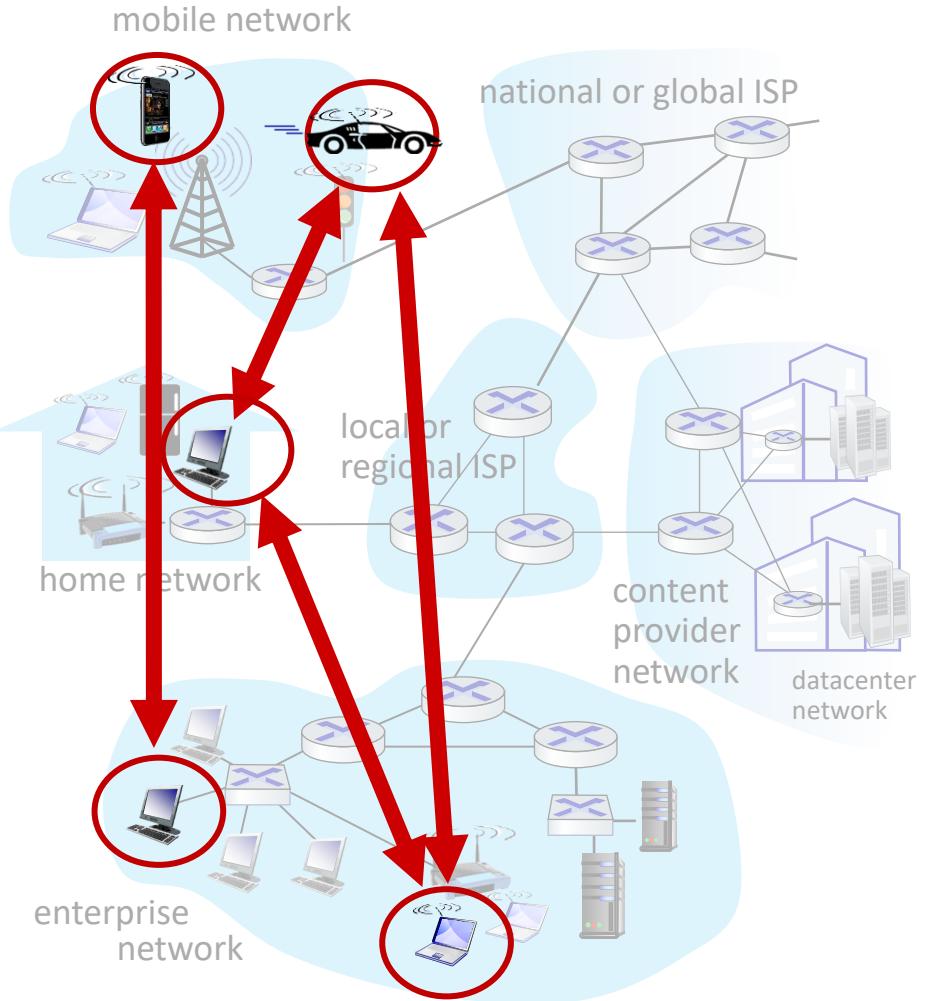
clients:

- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other
- examples: HTTP, IMAP, FTP



Peer-peer architecture

- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
 - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
 - complex management
- example: P2P file sharing



Processes communicating

process: program running within a host

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

clients, servers

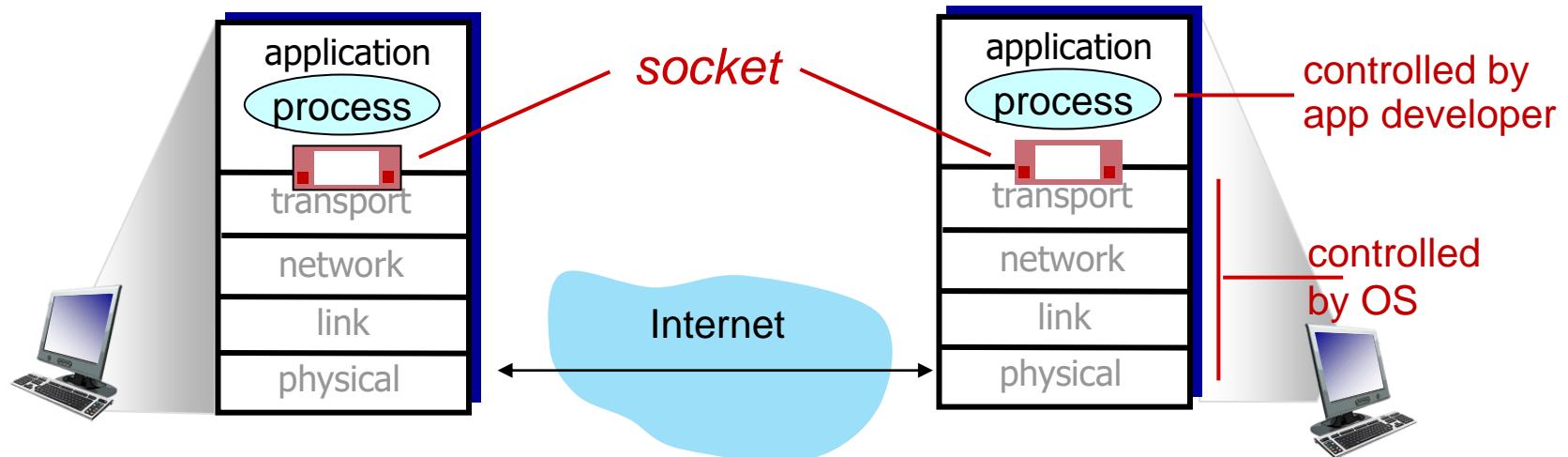
client process: process that initiates communication

server process: process that waits to be contacted

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

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
 - two sockets involved: one on each side



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 www.iiits.ac.in web server:
 - **IP address:** 103.21.58.130
 - **port number:** 443
- more shortly...

An application-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, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype, Zoom

What transport service does an app need?

data integrity

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

throughput

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

timing

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

security

- encryption, data integrity, ...

Transport service requirements: common apps

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? *Why* is there a UDP?

Internet applications, and transport protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Securing TCP

Vanilla TCP & UDP sockets:

- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (TLS)

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

TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn
- cleartext sent into “socket” traverse Internet *encrypted*
- more: Chapter 8

Application layer: overview

- Principles of network applications
- **Web and HTTP**
- E-mail, SMTP, IMAP
- The Domain Name System
DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Web and HTTP

First, a quick review...

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

www . someschool . edu / someDept / pic . gif

host name

path name

- One HTTP request for each object.

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)

HTTP 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 connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-persistent HTTP: example

User enters URL: `www.someSchool.edu/someDepartment/home.index`
(containing text, references to 10 jpeg images)



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



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

time
↓

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

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

Non-persistent HTTP: example (cont.)

User enters URL: `www.someSchool.edu/someDepartment/home.index`
(containing text, references to 10 jpeg images)



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

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



4. HTTP server closes TCP connection.

time

Non-persistent HTTP: response time

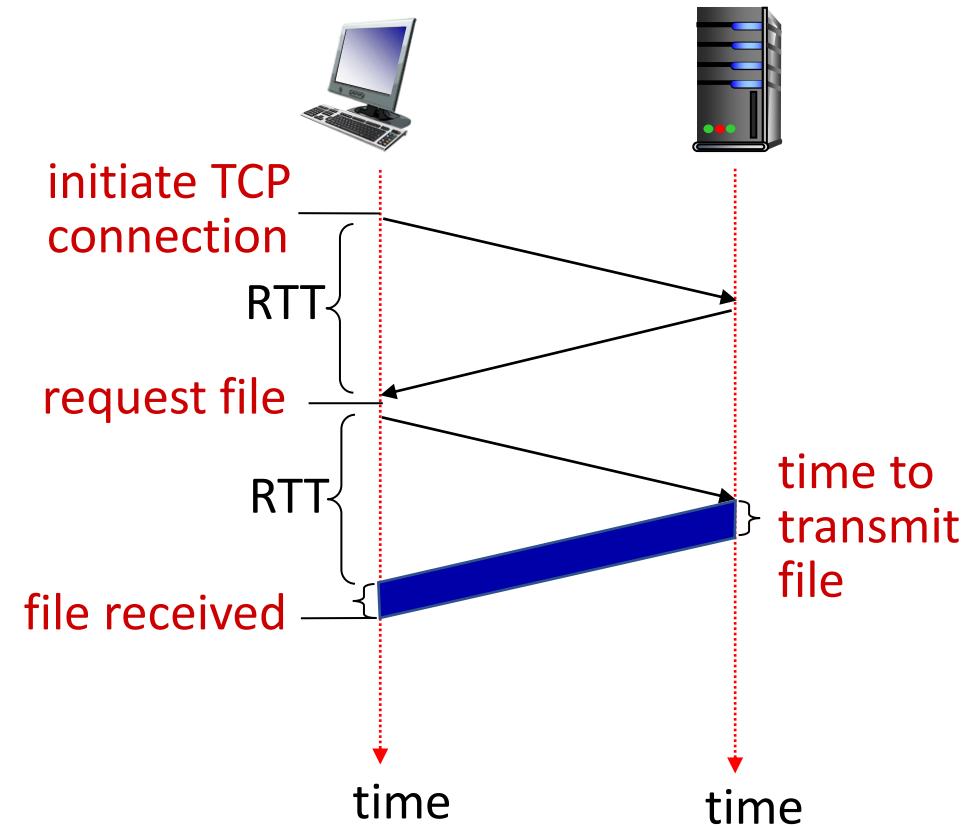
Remember Transmission Delay?

- Larger for larger packets and vice versa.

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

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and file transmission time



$$\text{Non-persistent HTTP response time} = 2\text{RTT} + \text{file transmission time}$$

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

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

Persistent HTTP (HTTP1.1):

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

Persistent v/s Non-persistent HTTP

1. Suppose you want to fetch an HTML page (index.html) from <https://iiits.ac.in> such that it also has references to 8 very small objects on the same server. Let RTT_0 denotes the RTT between your computer and the IIIT's web server. Neglecting transmission times, how much time elapses with

- a) Non-persistent HTTP with no parallel TCP connections?

$$2*RTT_0 + 8*2*RTT_0 = 18RTT_0$$

- b) Non-persistent HTTP with the browser configured for 6 parallel connections?

$$2*RTT_0 + 2*RTT_0 + 2*RTT_0 = 6RTT_0$$

- c) Persistent HTTP?

- i. Without pipelining:

$$2*RTT_0 + 8*RTT_0 = 10RTT_0$$

- ii. With pipelining:

$$2*RTT_0 + 1*RTT_0 = 3RTT_0 \text{ (note that objects are very small and thus, assume that transmission delay is negligible.)}$$

Which one should HTTP use?

Persistent HTTP with pipelining (But due to issues such as Head-of-Line blocking, pipelining is not used by default. So the most common is persistent HTTP w/o pipelining.)

HTTP request message

- two types of HTTP messages: *request, response*
- **HTTP request message:**

- ASCII (human-readable format)

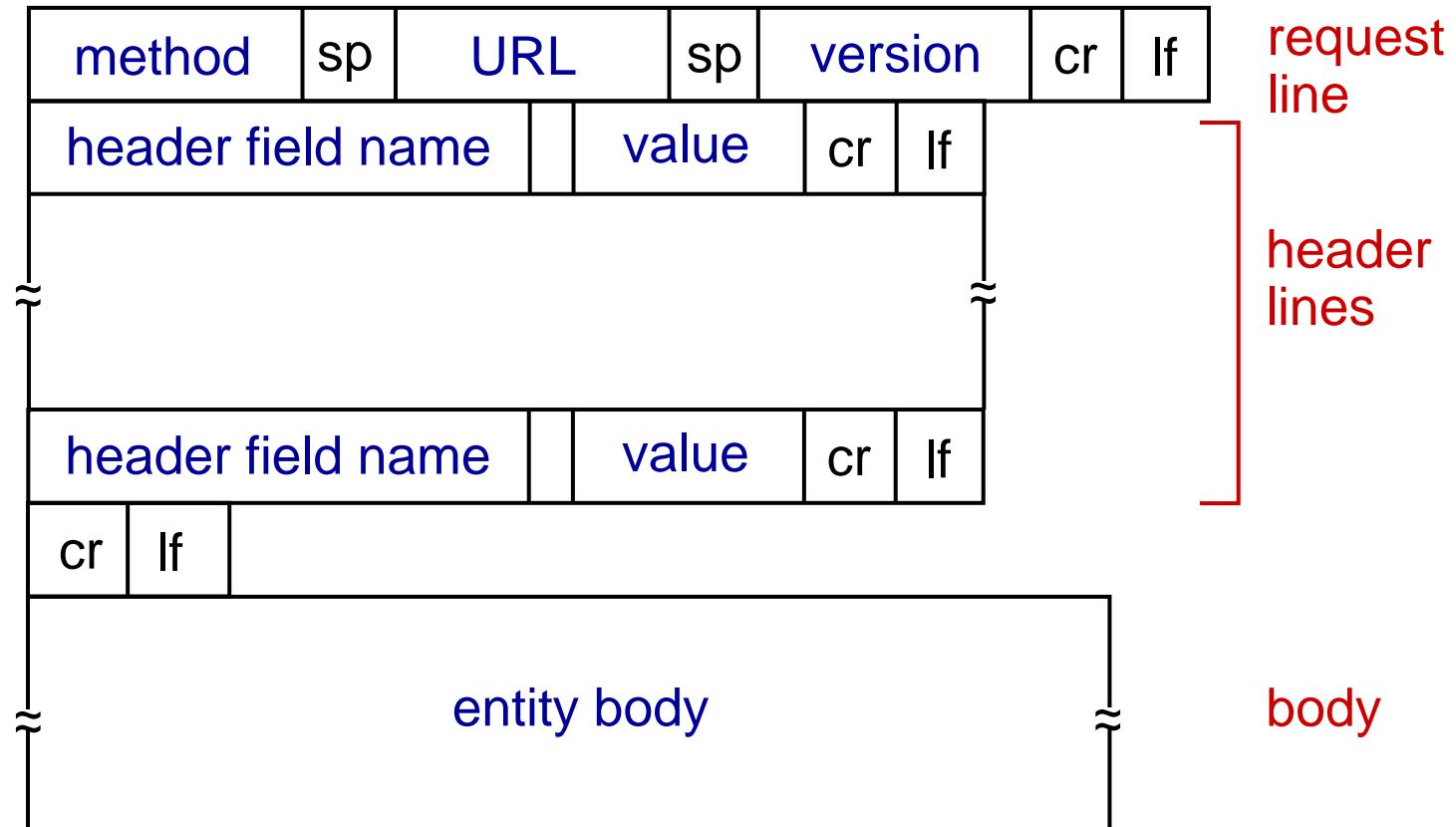
request line (GET, POST,
HEAD commands) →

carriage return character
line-feed character

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

* Check out the online interactive exercises for more
examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP request message: general format



Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

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

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

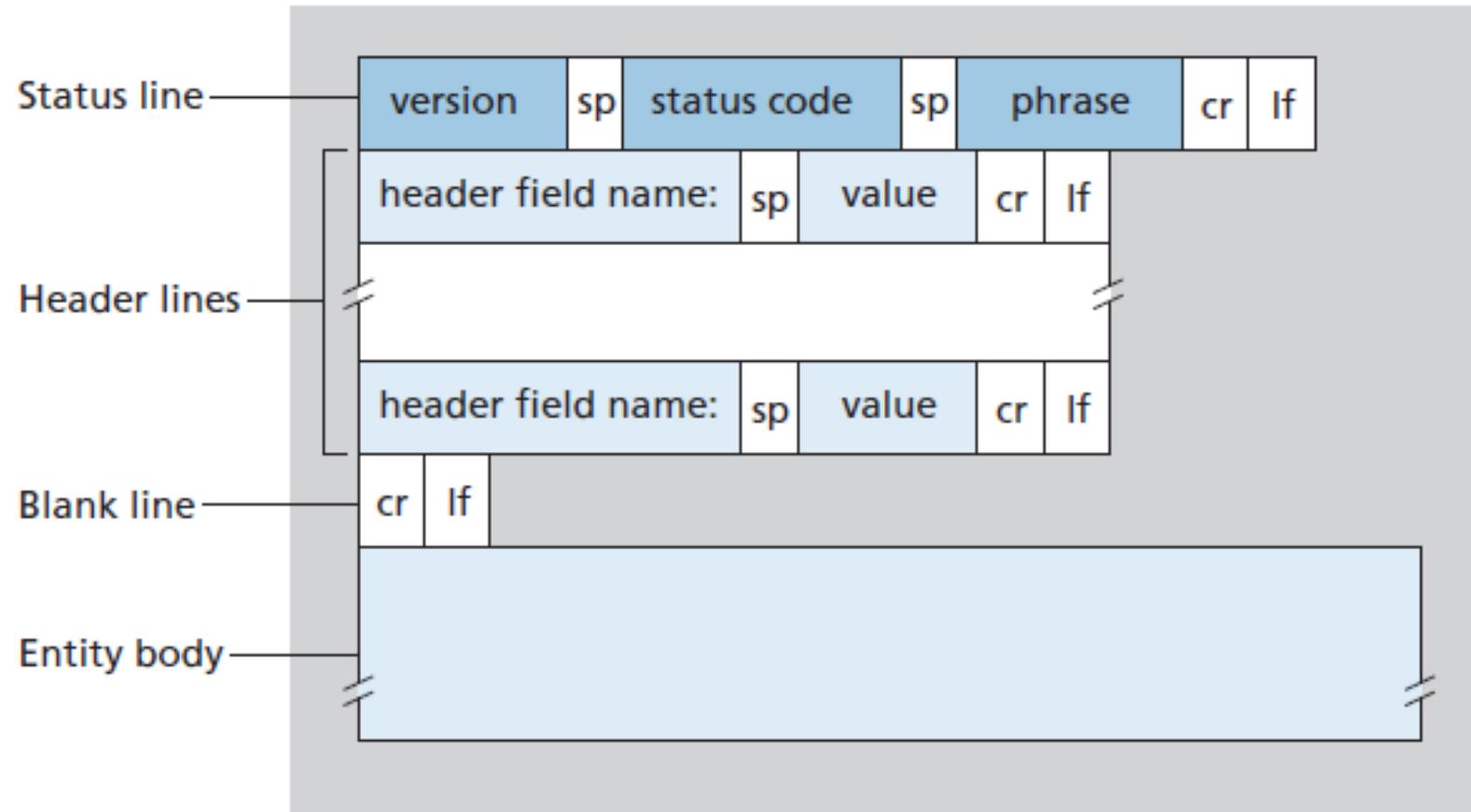
PUT method:

- uploads new file (object) to server

HTTP response message

status line (protocol → **HTTP/1.1 200 OK**
status code status phrase)

HTTP request message: general format



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 message

301 Moved Permanently

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

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. netcat to your favorite Web server:

```
% nc -c -v gaia.cs.umass.edu 80
```

- opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass.edu.
- anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1  
Host: gaia.cs.umass.edu
```

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

3. look at response message sent by HTTP server!

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

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

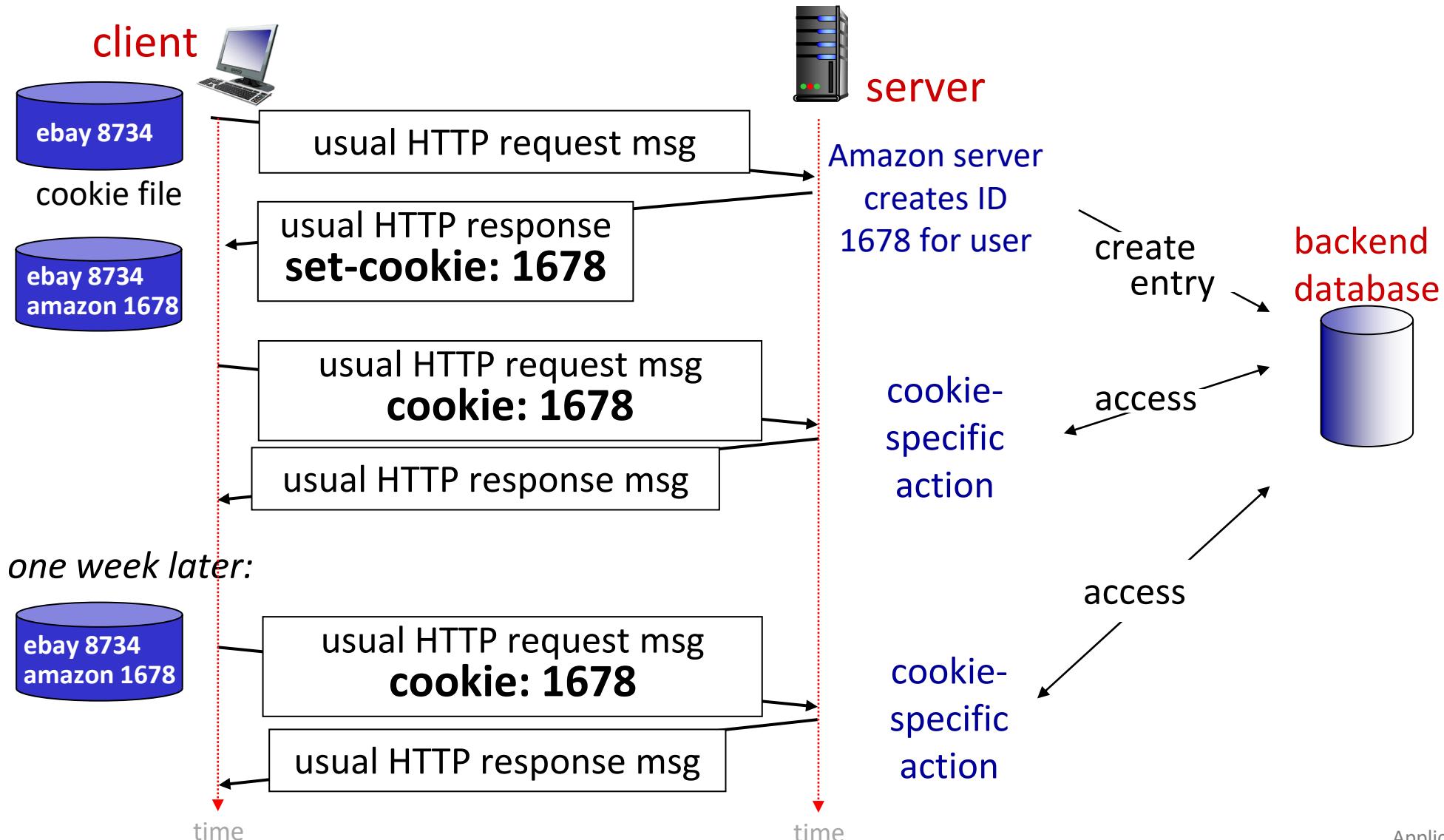
four components:

- 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 uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
 - unique ID (aka "cookie")
 - entry in backend database for ID
 - subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

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

*aside
cookies and privacy:*

- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



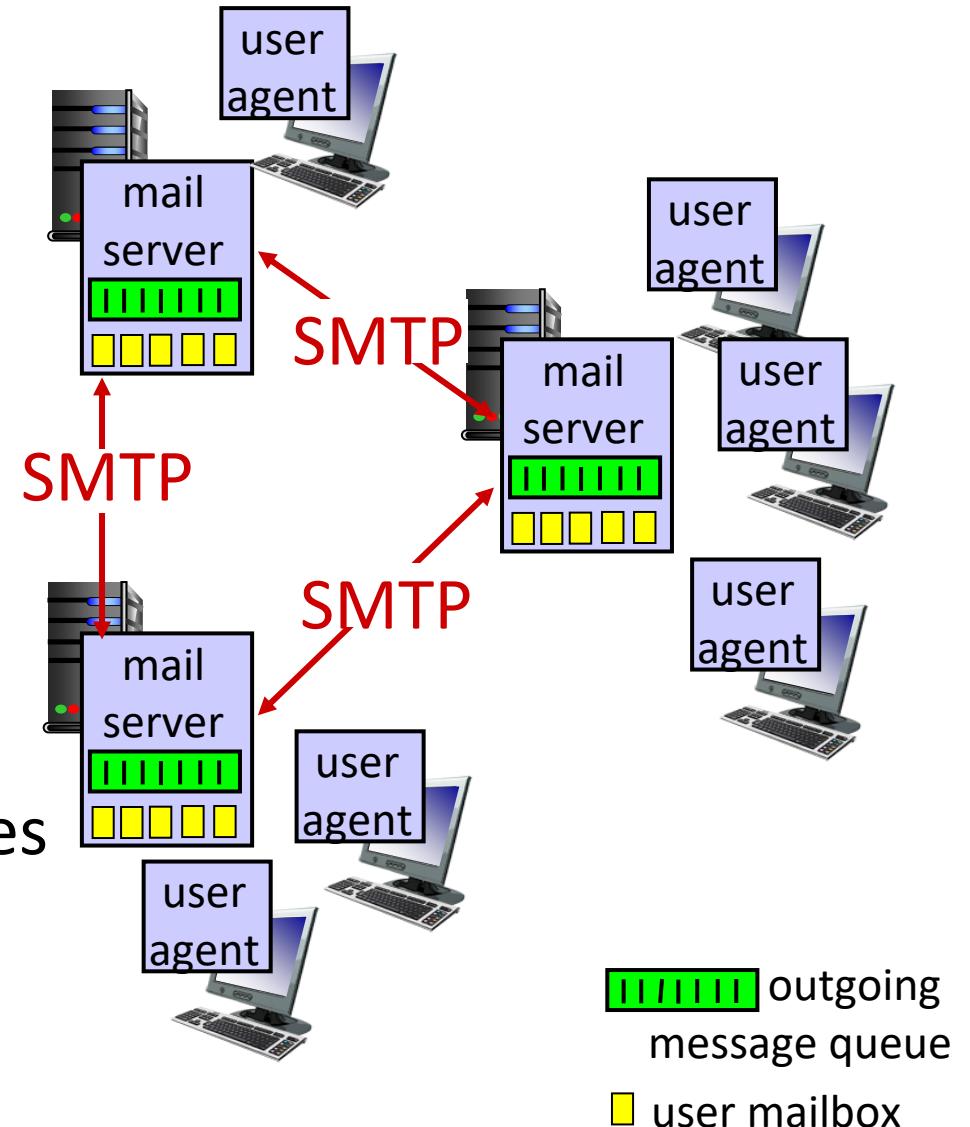
E-mail

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, iPhone mail client
- outgoing, incoming messages stored on server



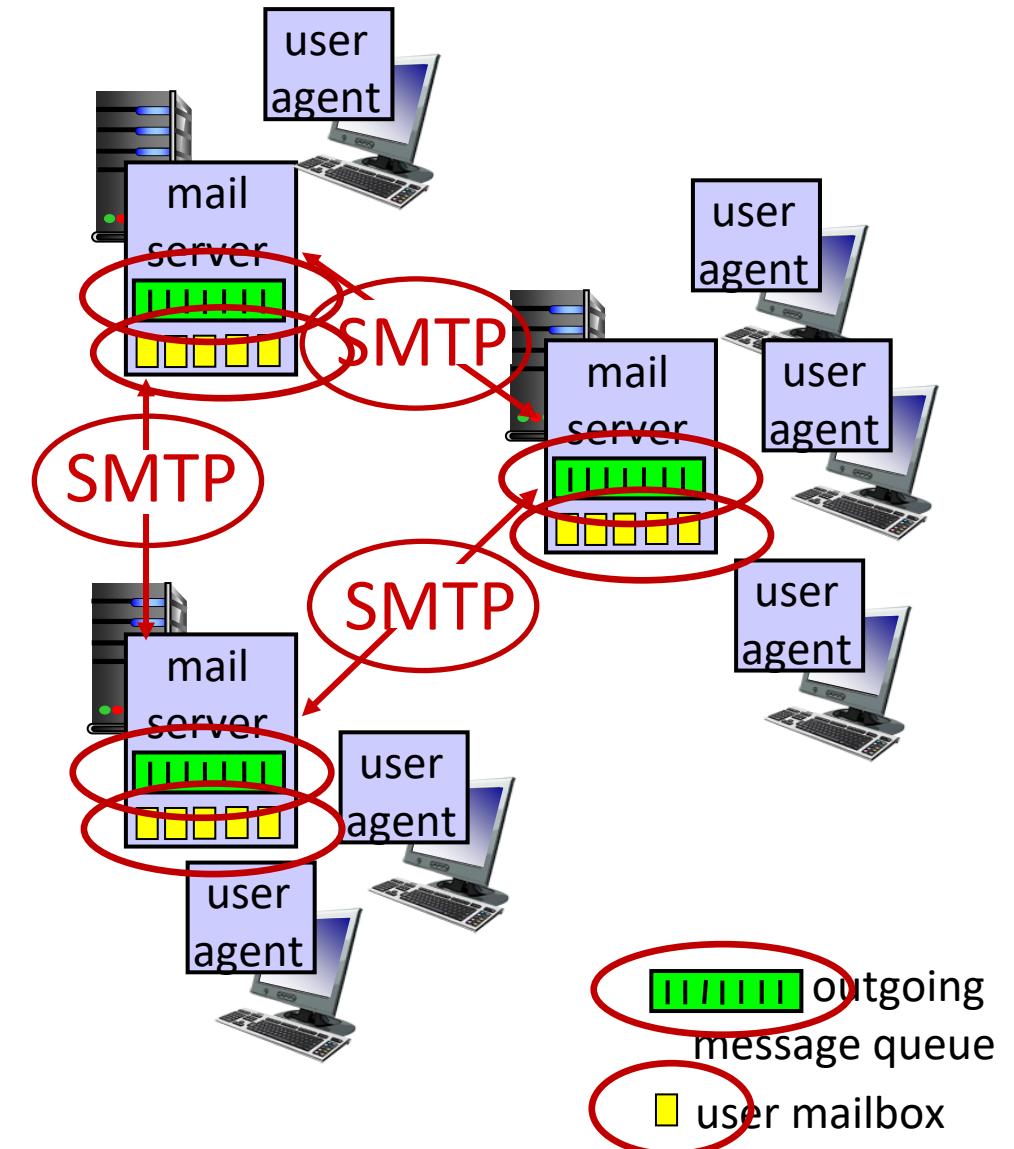
E-mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages

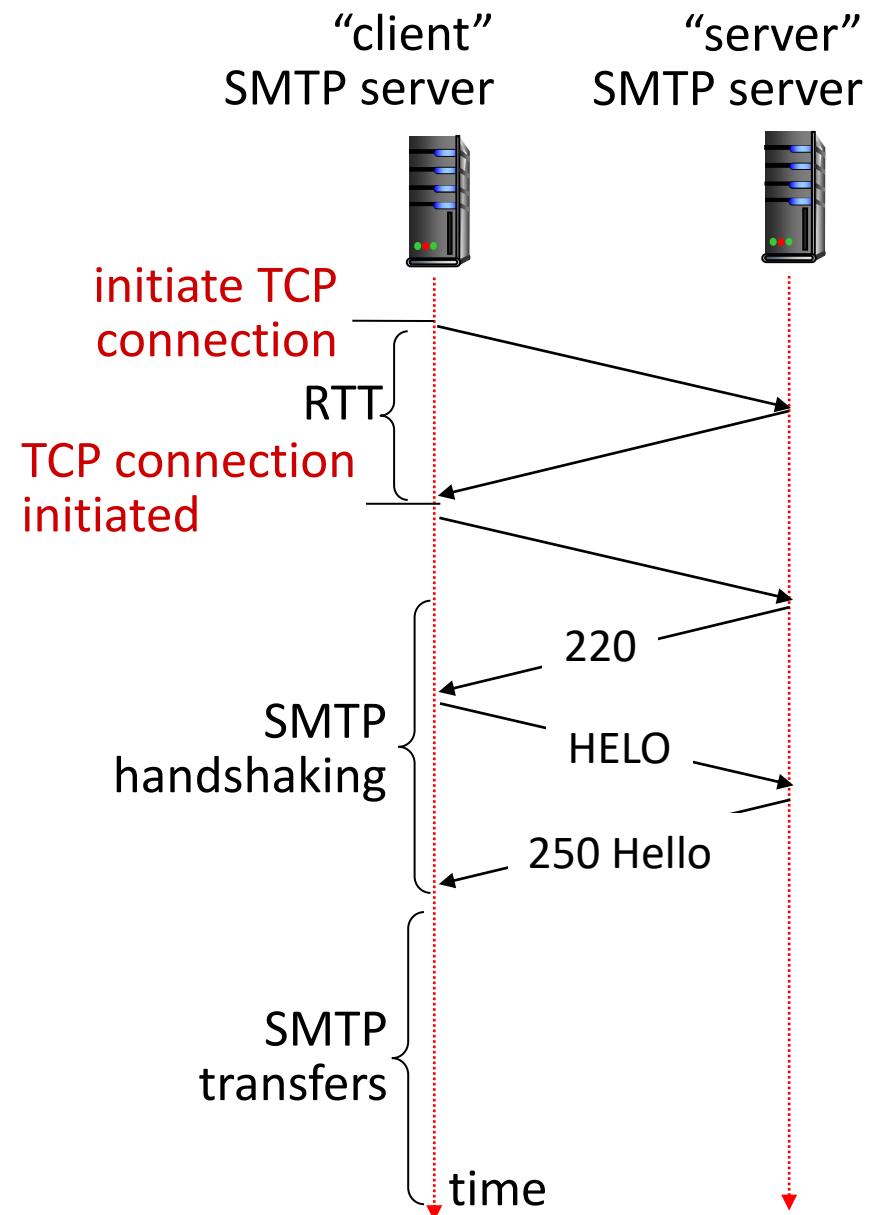
SMTP protocol between mail servers to send email messages

- client: sending mail server
- “server”: receiving mail server



SMTP RFC (5321)

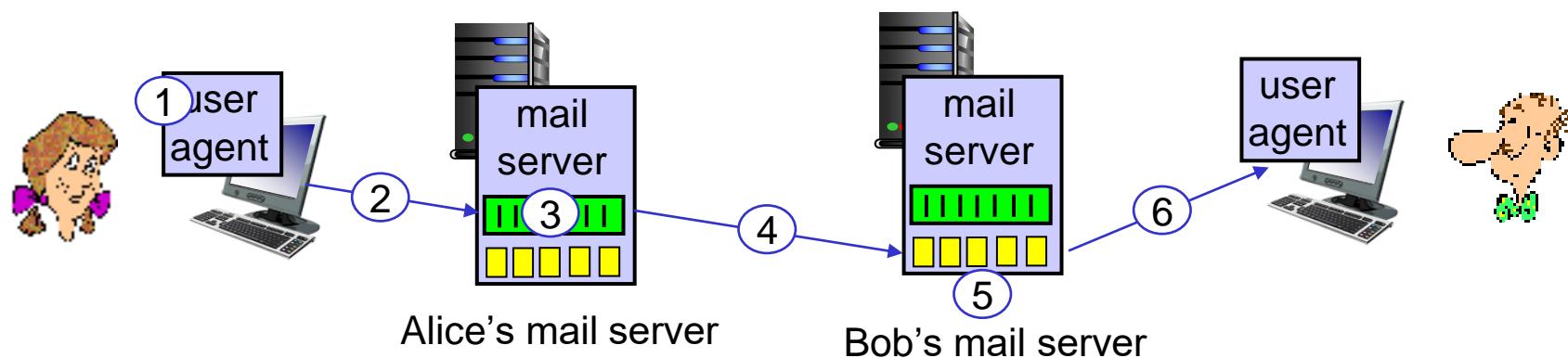
- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
 - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
 - SMTP handshaking (greeting)
 - SMTP transfer of messages
 - SMTP closure
- command/response interaction (like HTTP)
 - commands: ASCII text
 - response: status code and phrase



Scenario: Alice sends e-mail to Bob

- 1) Alice uses UA to compose e-mail message “to” bob@someschool.edu
- 2) Alice’s UA sends message to her mail server using SMTP; message placed in message queue
- 3) client side of SMTP at mail server 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

SMTP: observations

comparison with HTTP:

- HTTP: client pull
- SMTP: client push
- both have ASCII command/response interaction, status codes

- 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

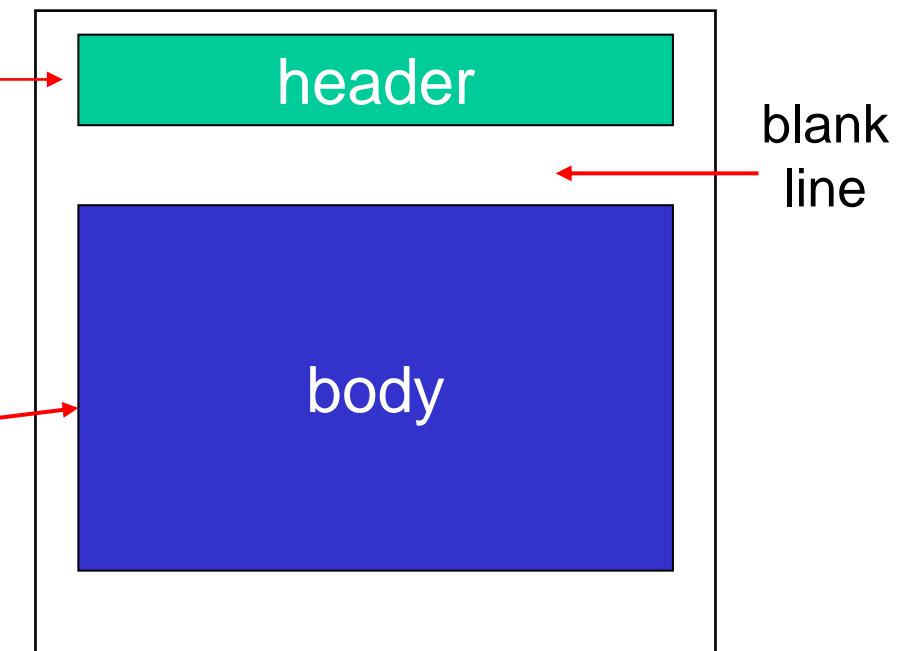
Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 5321
(like RFC 7231 defines HTTP)

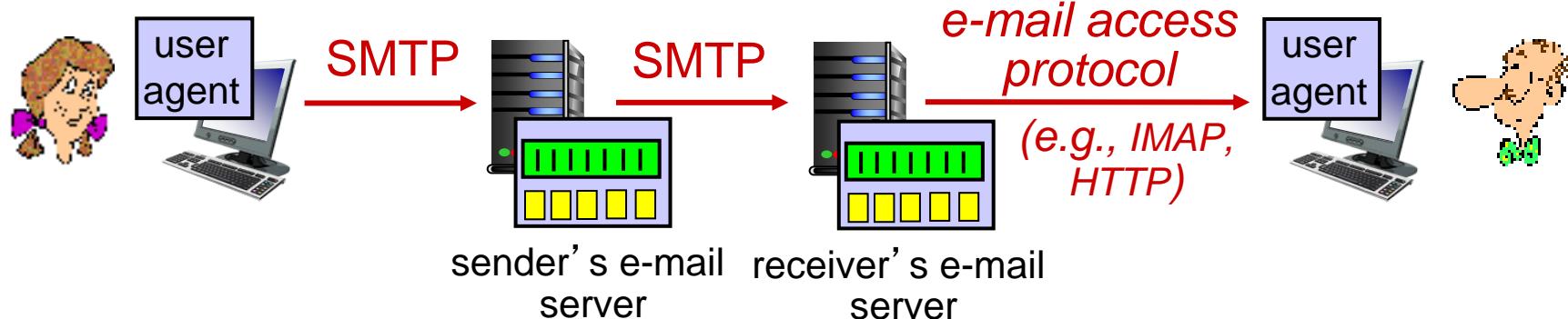
RFC 2822 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

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

these lines, within the body of the email message area different from SMTP MAIL FROM:,
~~RCPT TO: commands!~~
- Body: the “message”, ASCII characters only



Retrieving email: mail access protocols



- **SMTP:** delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
 - **IMAP:** Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- **HTTP:** gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages

POP3 and IMAP

- POP3 have 2 modes:
 - Download and delete
 - Can't reread if client changed
 - Download and keep
 - Copies of mails on different clients
- Keeps all messages at one place: at server
 - Allows users to organize mails in folders
 - Messages will not be deleted from the server if downloaded on local computer

Application Layer: Overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



DNS: Domain Name System

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., cs.umass.edu - used by humans

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

Domain Name System (DNS):

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

DNS: services, structure

DNS services:

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

Q: Why not centralize DNS?

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

A: doesn't scale!

- Comcast DNS servers alone: 600B DNS queries/day
- Akamai DNS servers alone: 2.2T DNS queries/day

Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

- *many* more reads than writes
- *performance matters*: almost every Internet transaction interacts with DNS - msecs count!

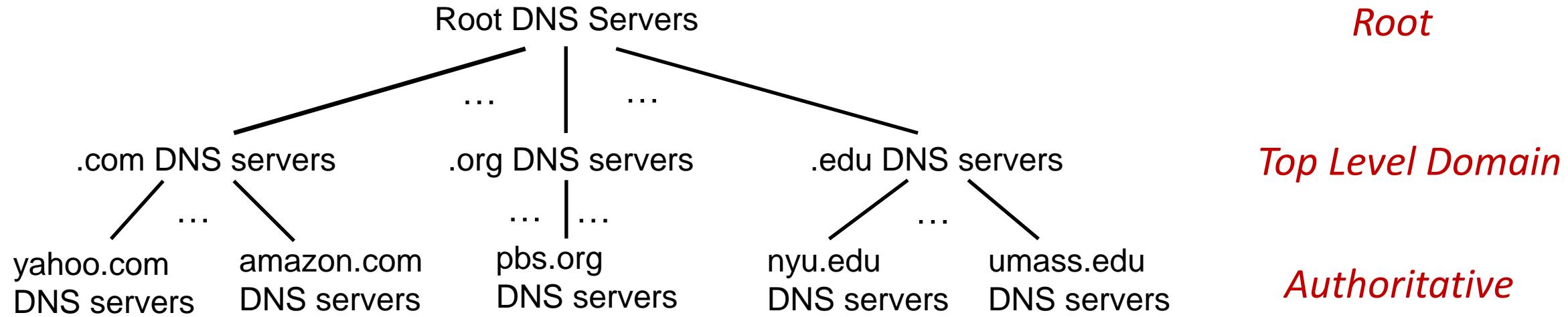
organizationally, physically decentralized:

- millions of different organizations responsible for their records

“bulletproof”: reliability, security



DNS: a distributed, hierarchical database

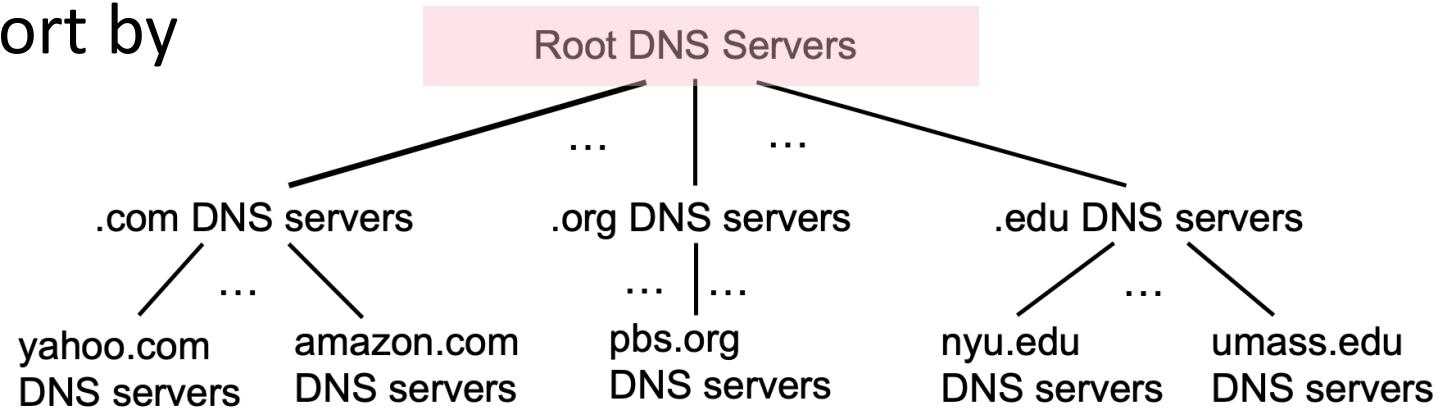


Client wants IP address for www.amazon.com; 1st approximation:

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

DNS: root name servers

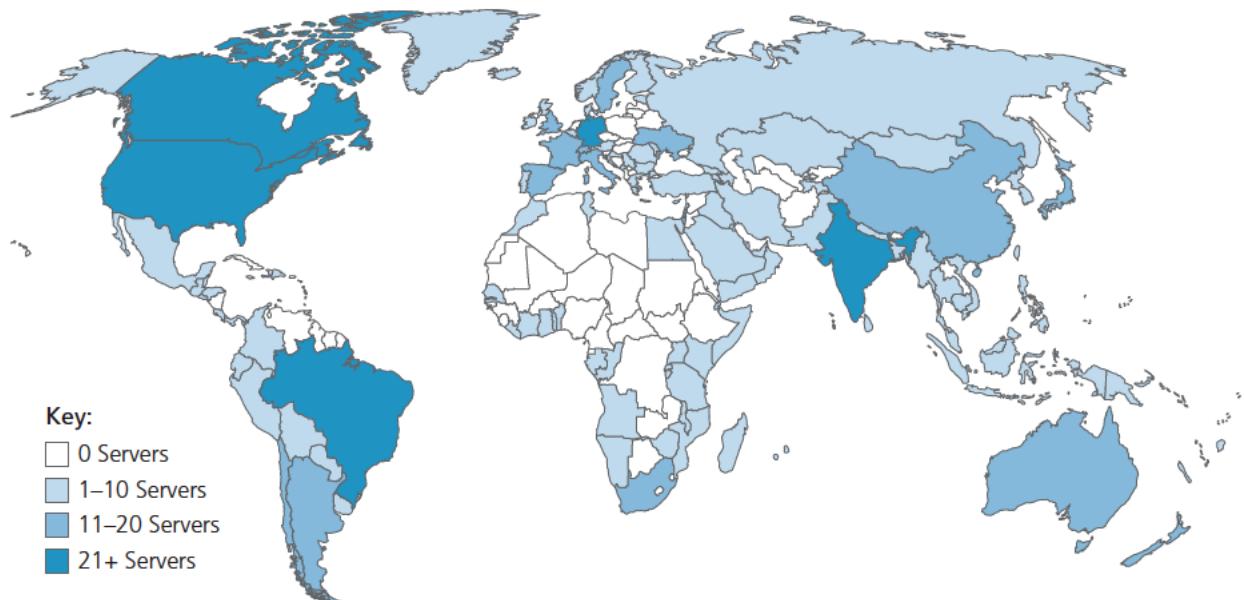
- official, contact-of-last-resort by name servers that can not resolve name



DNS: root name servers

- official, contact-of-last-resort by name servers that can not resolve name
- *incredibly important* Internet function
 - Internet couldn't function without it!
 - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

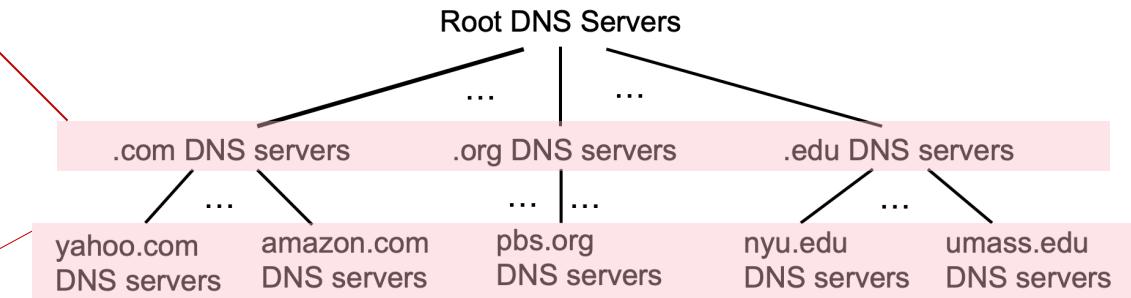
13 logical root name “servers” worldwide each “server” replicated many times (~200 servers in US)



Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

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

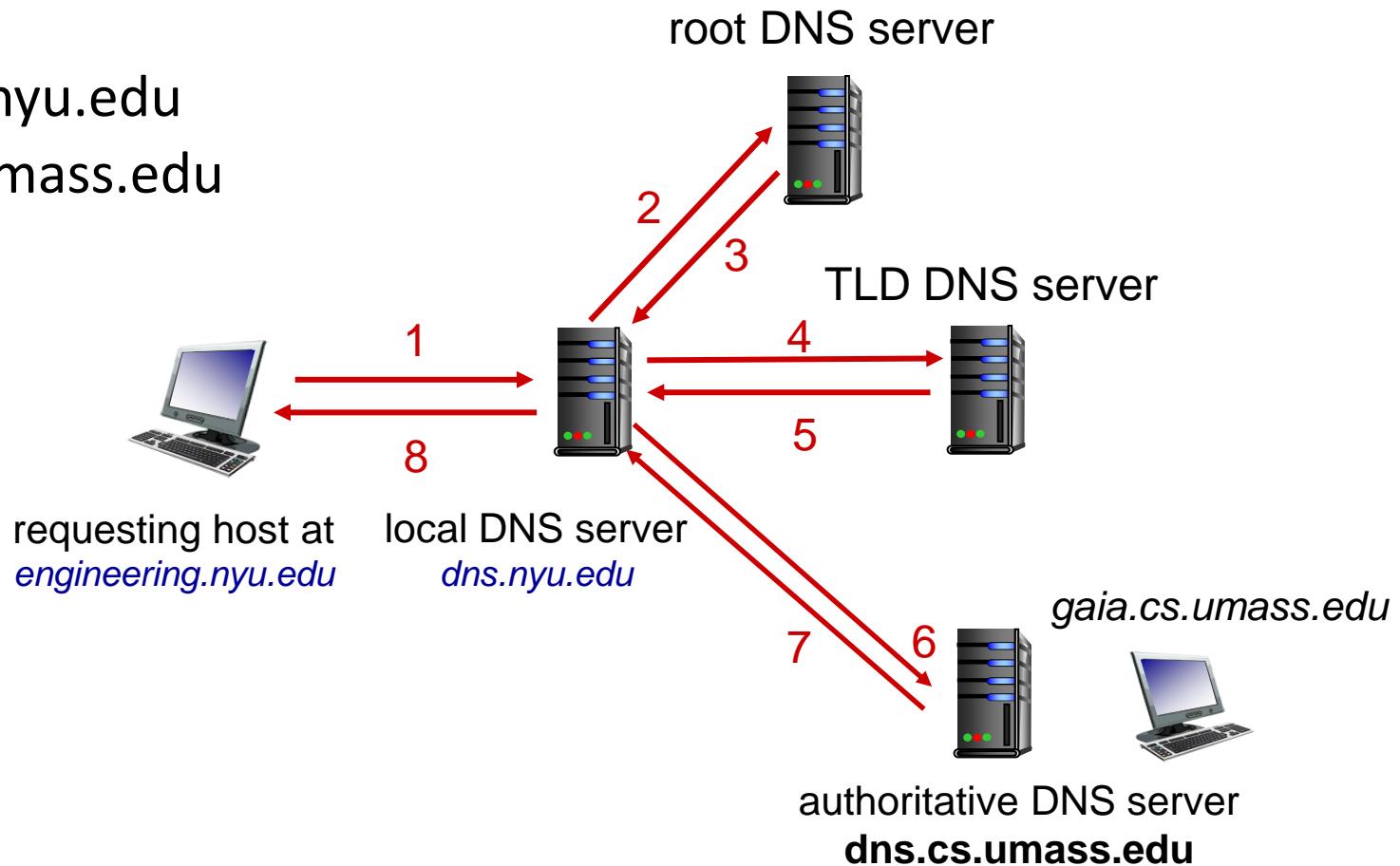
- when host makes DNS query, it is sent to its *local* DNS server
 - Local DNS server returns reply, answering:
 - from its local cache of recent name-to-address translation pairs (possibly out of date!)
 - forwarding request into DNS hierarchy for resolution
 - each ISP has local DNS name server; to find yours:
 - MacOS: % scutil --dns
 - Windows: >ipconfig /all
- local DNS server doesn't strictly belong to hierarchy

DNS name resolution: iterated query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Iterated query:

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

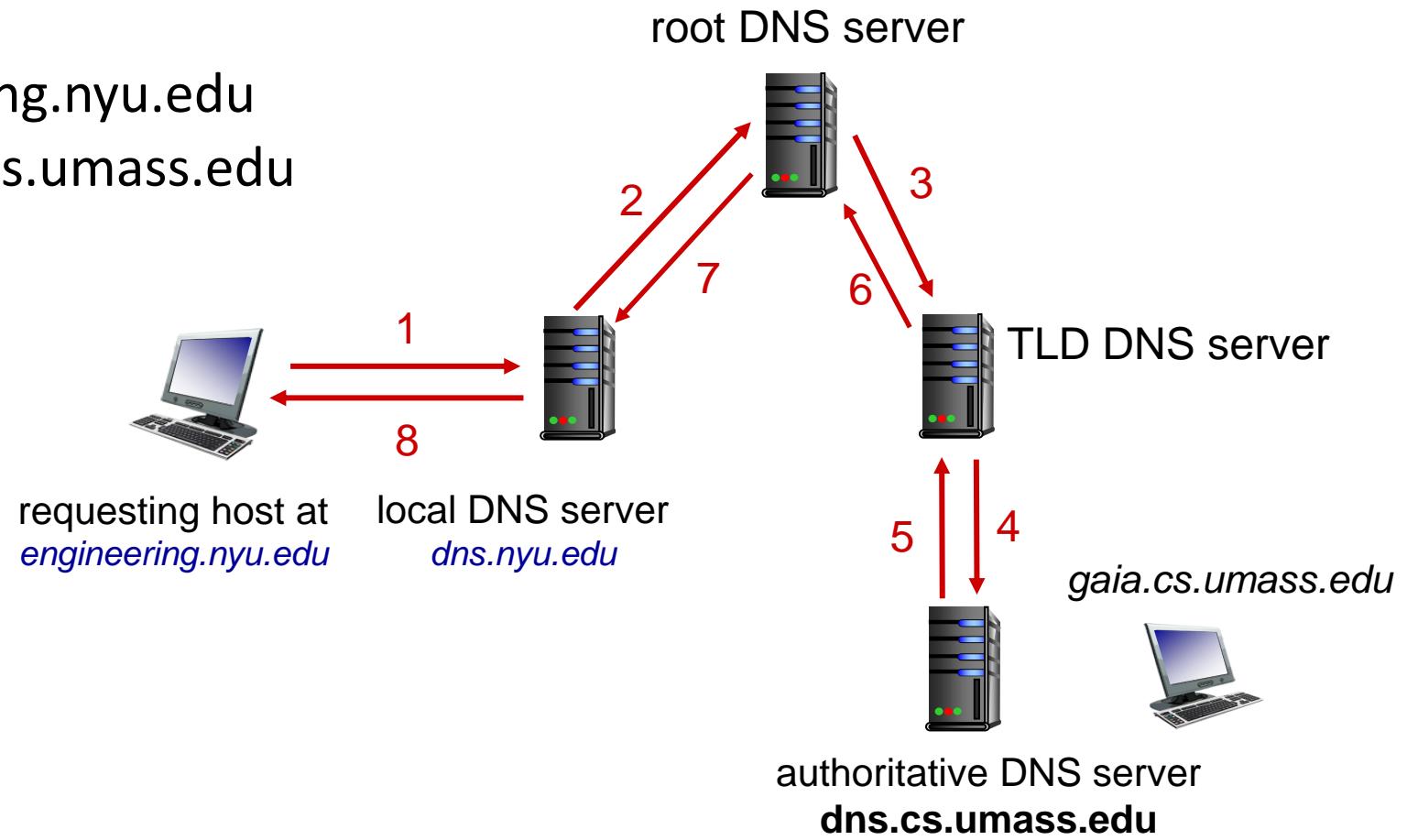


DNS name resolution: recursive query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:

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



Caching DNS Information

- once (any) name server learns mapping, it *caches* mapping, and *immediately* returns a cached mapping in response to a query
 - caching improves response time
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers
- cached entries may be *out-of-date*
 - if named host changes IP address, may not be known Internet-wide until all TTLs expire!
 - *best-effort name-to-address translation!*

DNS records

DNS: distributed database 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

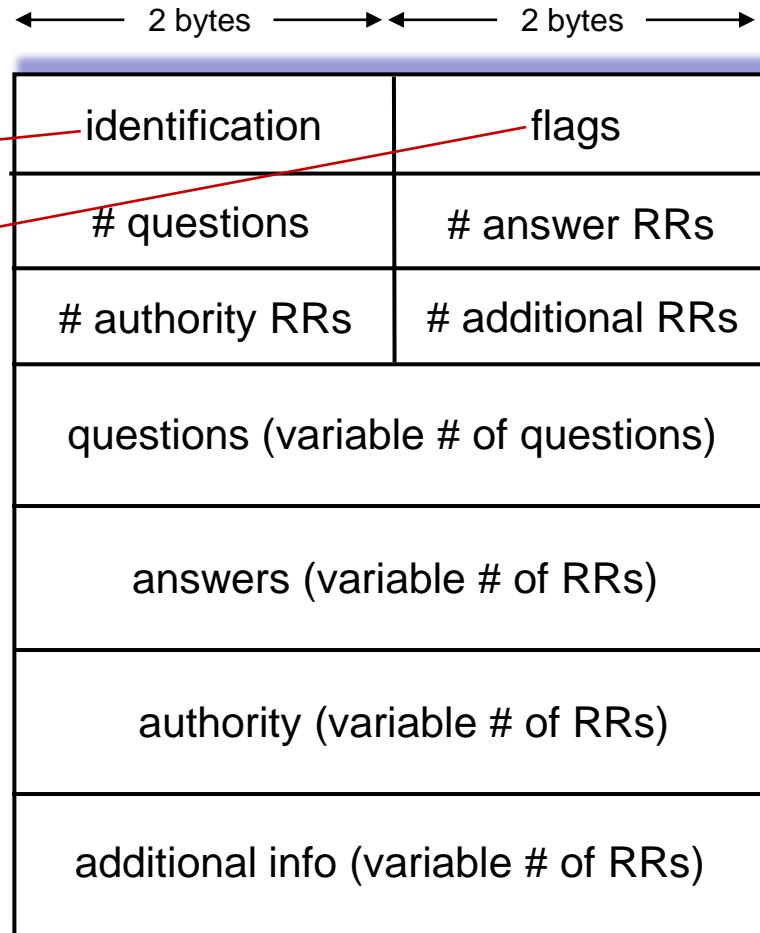
- value is name of SMTP mail server associated with name

DNS protocol messages

DNS *query* and *reply* messages, both have same *format*:

message 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

DNS *query* and *reply* messages, both have same *format*:

↔ 2 bytes → ← 2 bytes →

identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

name, type fields for a query

questions (variable # of questions)

RRs in response to query

answers (variable # of RRs)

records for authoritative servers

authority (variable # of RRs)

additional “helpful” info that may
be used

additional info (variable # of RRs)

Getting your info into the DNS

example: new startup “Network Utopia”

- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts NS, A RRs into .com TLD server:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server locally with IP address 212.212.212.1
 - type A record for www.networkuptopia.com
 - type MX record for networkutopia.com

DNS security

DDoS attacks

- bombard root servers with traffic
 - not successful to date
 - traffic filtering
 - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
 - potentially more dangerous

Spoofing attacks

- intercept DNS queries, returning bogus replies
 - DNS cache poisoning
 - RFC 4033: DNSSEC authentication services

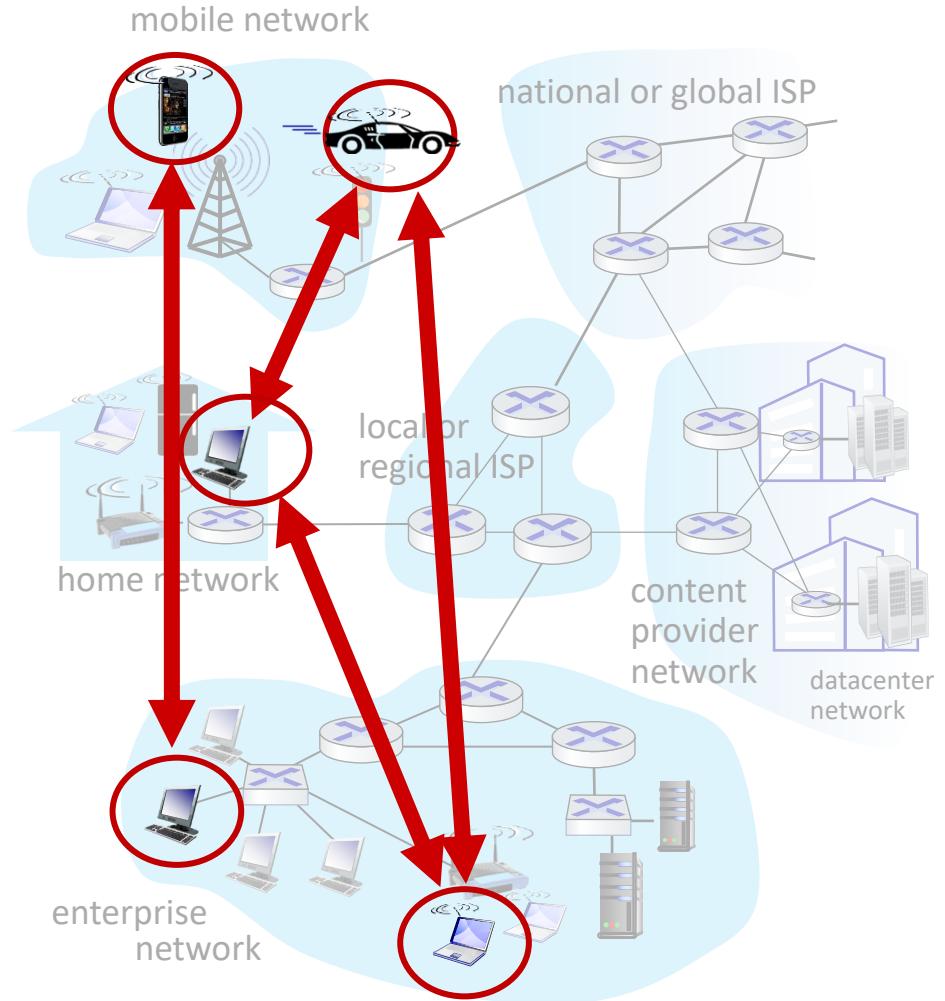
Application Layer: Overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP



Peer-to-peer (P2P) architecture

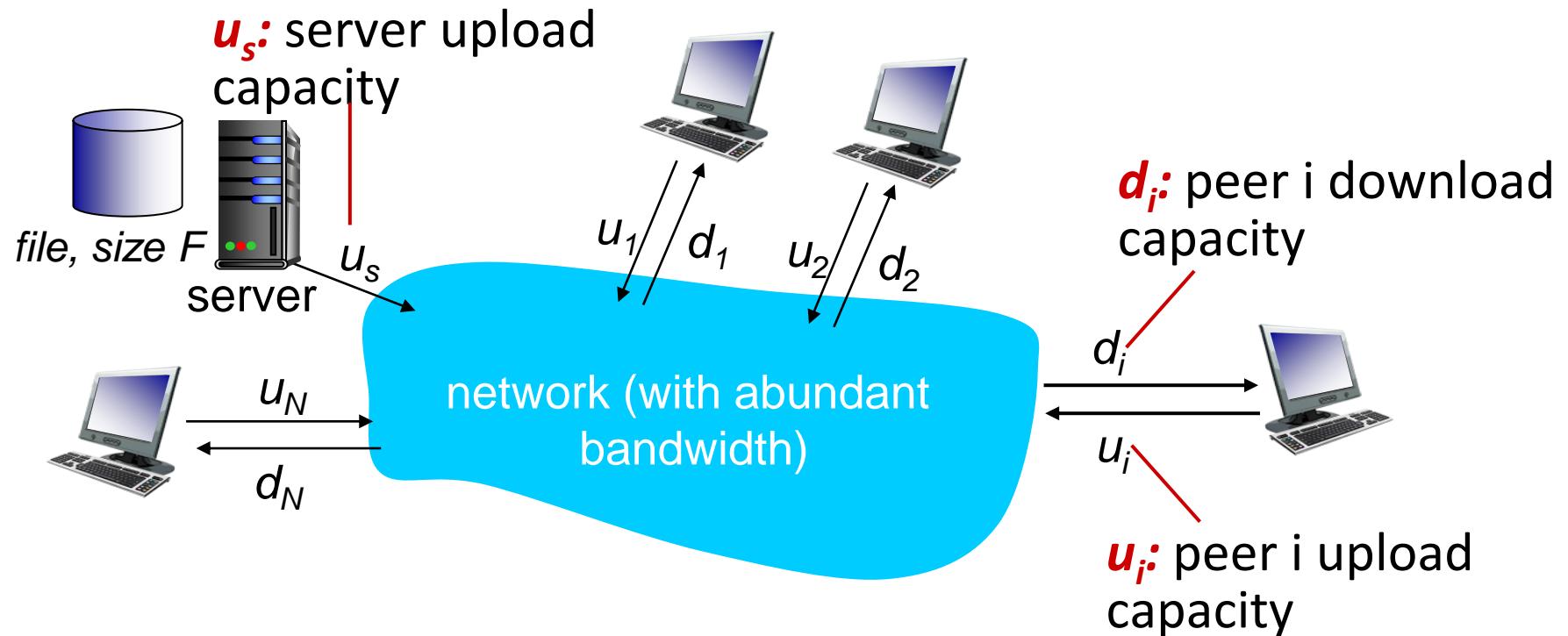
- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
 - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
 - complex management
- examples: P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)



File distribution: client-server vs P2P

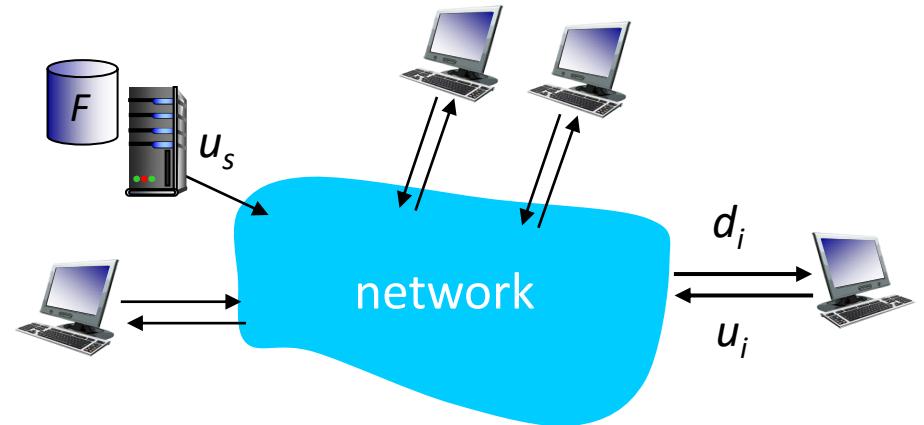
Q: how much time to distribute file (size F) from one server to N peers?

- peer upload/download capacity is limited resource



File distribution time: client-server

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



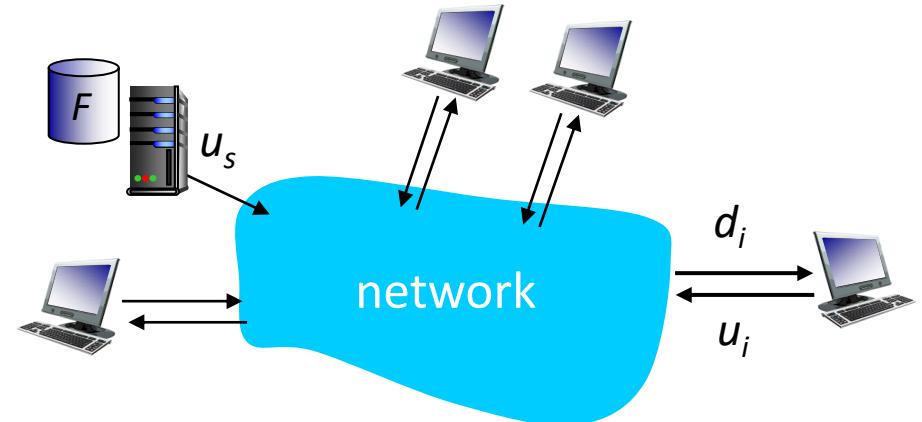
*time to distribute F
to N clients using
client-server approach*

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

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



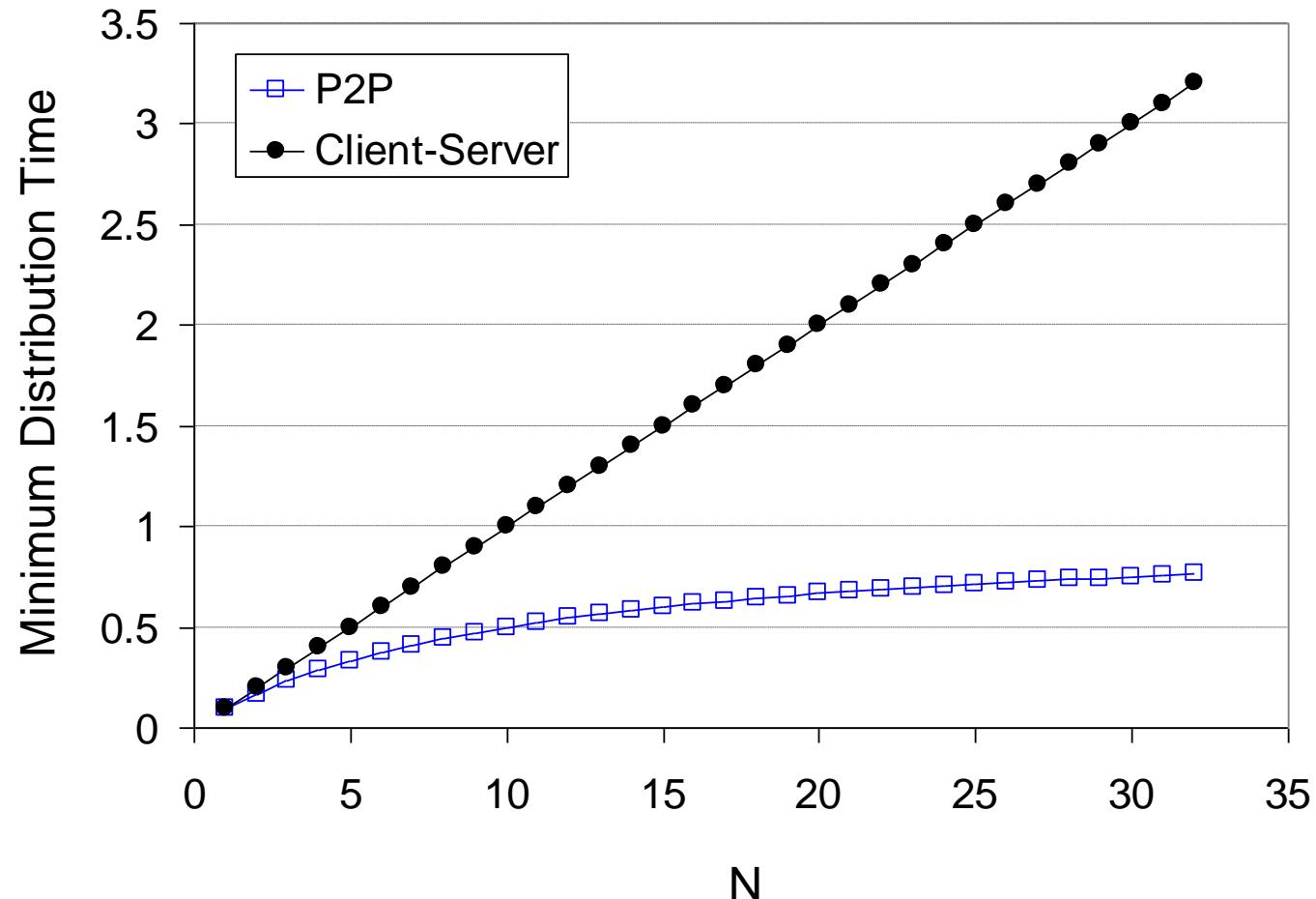
time to distribute F
to N clients using
P2P approach

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

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

Client-server vs. P2P: example

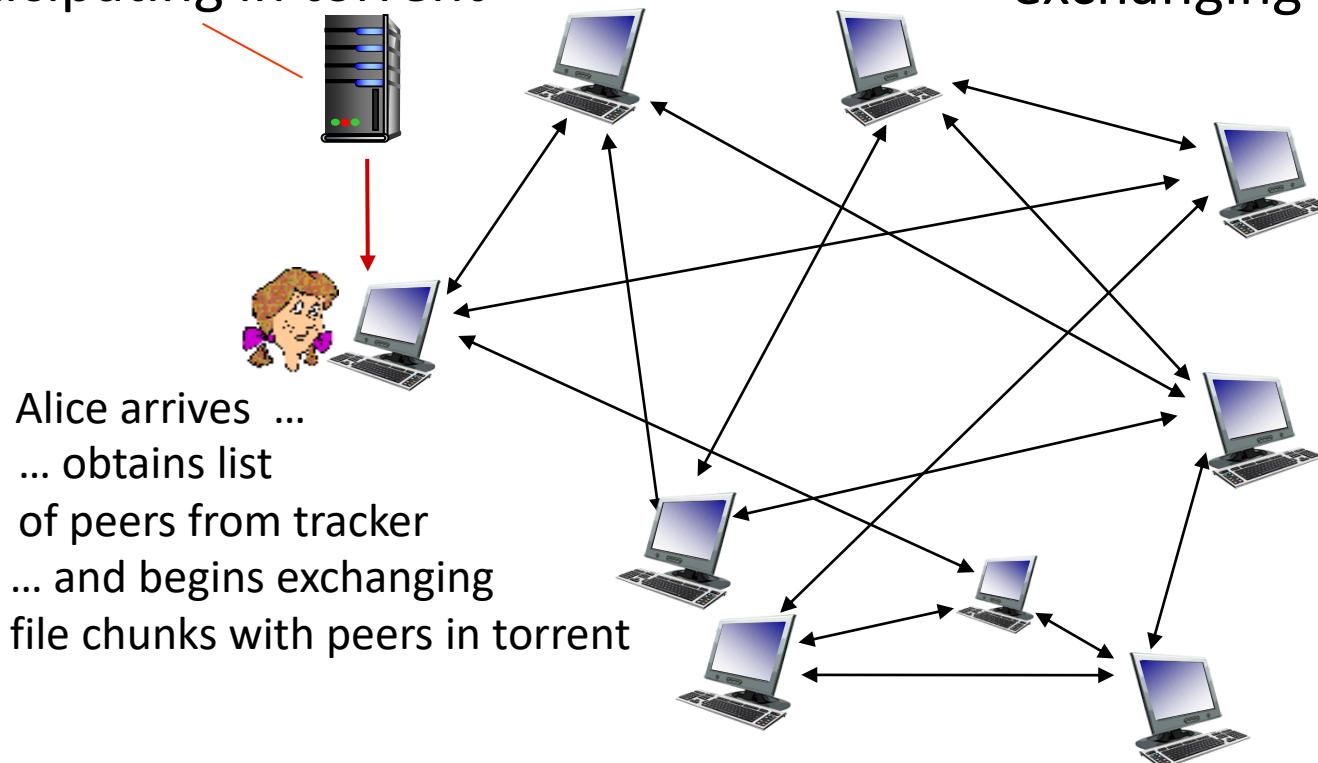
client upload rate = u , $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$



P2P file distribution: BitTorrent

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

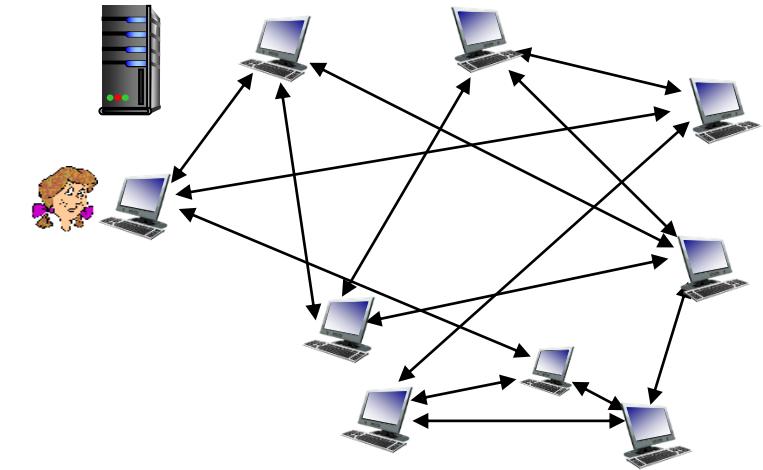
tracker: tracks peers
participating in torrent



torrent: group of peers
exchanging chunks of a file

P2P file distribution: BitTorrent

- peer joining torrent:
 - has no chunks, but will accumulate them over time from other peers
 - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn*: peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent



BitTorrent: requesting, sending file chunks

Requesting chunks:

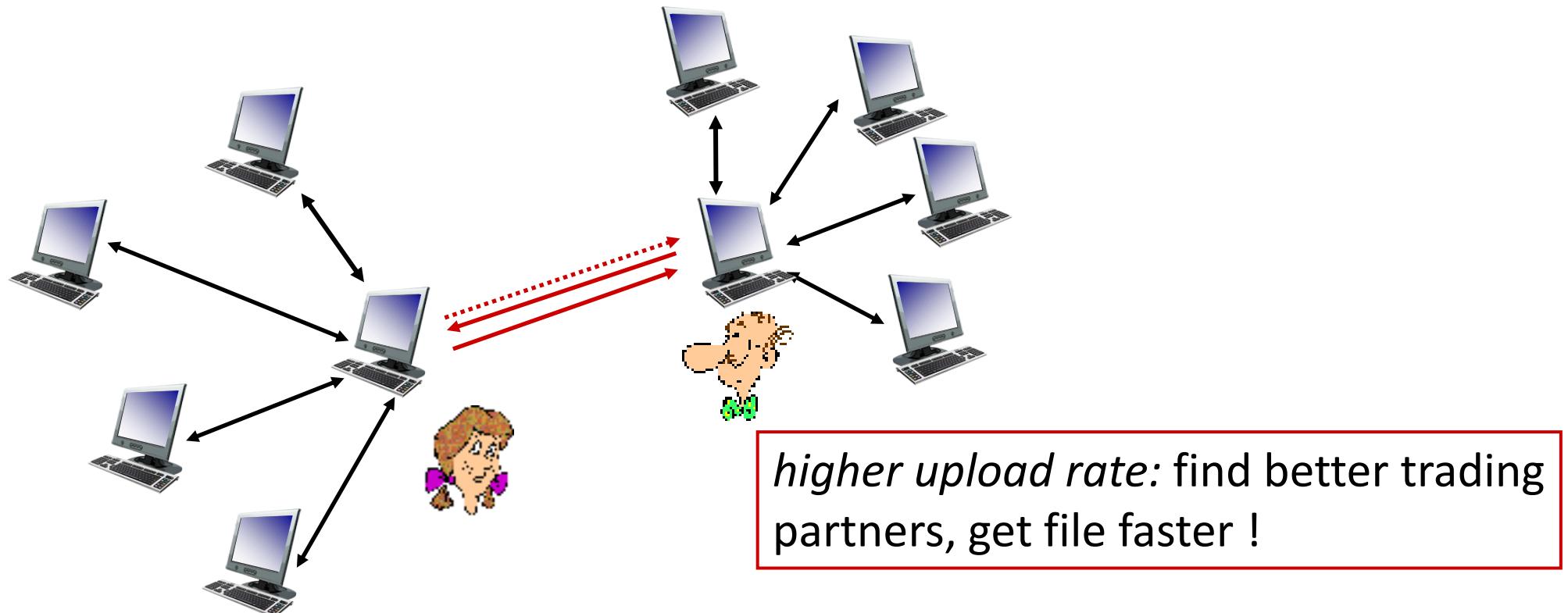
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

Sending chunks: tit-for-tat

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

BitTorrent: tit-for-tat

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



Chapter 2: Summary

our study of network application layer is 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, IMAP
 - DNS
 - P2P: BitTorrent

Chapter 2: Summary

Most importantly: learned about *protocols*!

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

important themes:

- centralized vs. decentralized
- stateless vs. stateful
- scalability
- reliable vs. unreliable message transfer
- “complexity at network edge”

Additional Chapter 2 slides

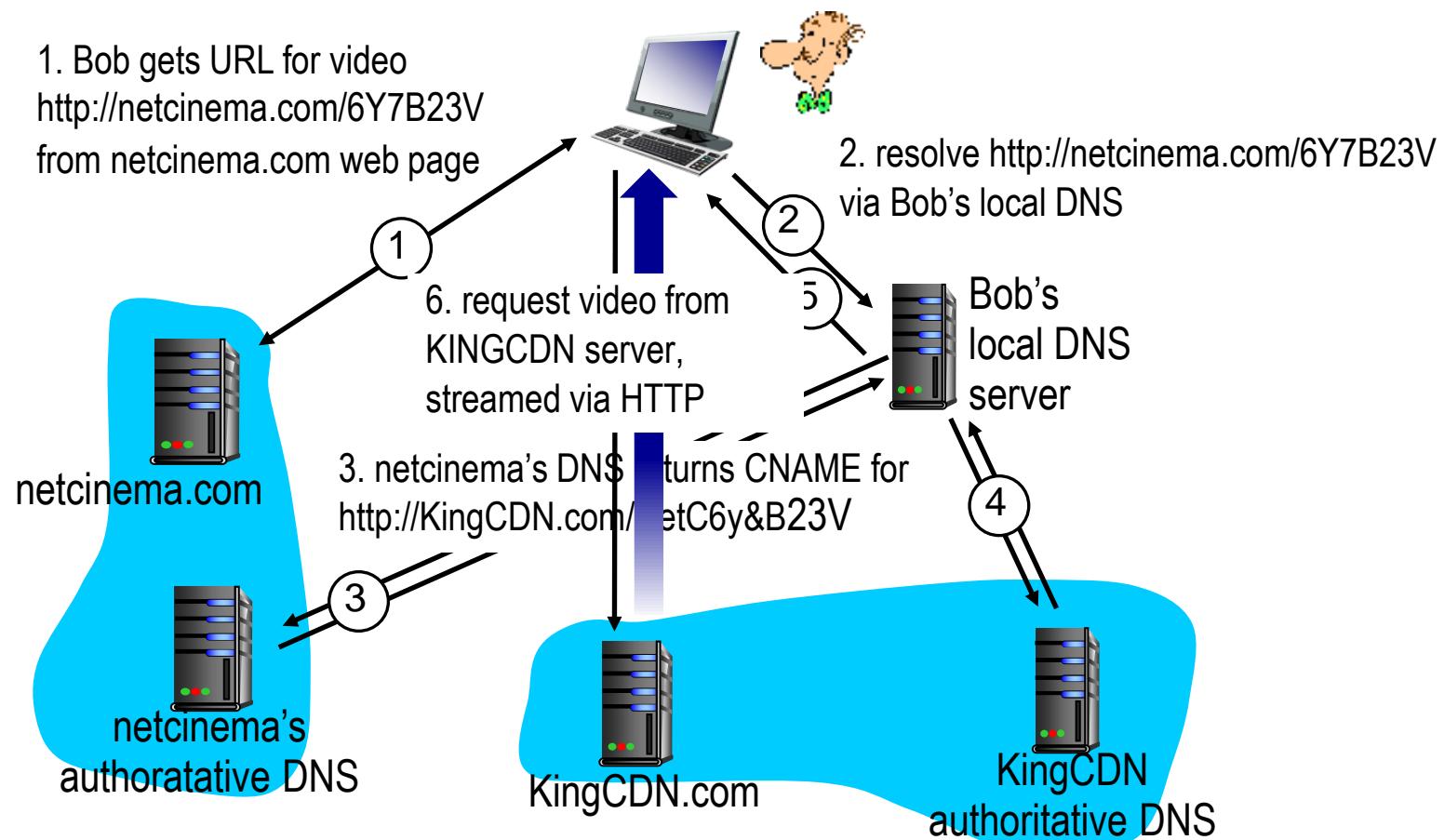
Sample SMTP interaction

```
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
```

CDN content access: a closer look

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

- video stored in CDN at <http://KingCDN.com/NetC6y&B23V>



Case study: Netflix

