

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.

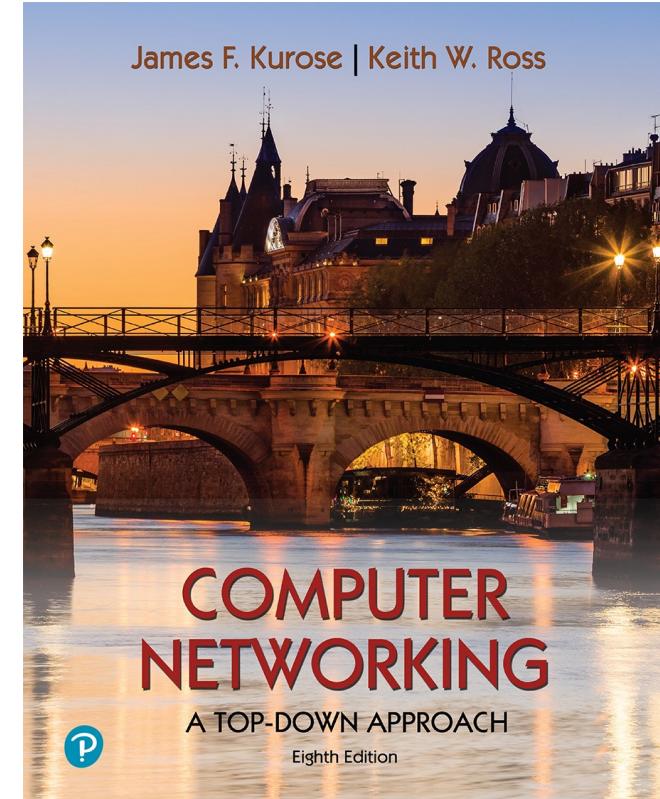
In return for use, we only ask the following:

- 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)

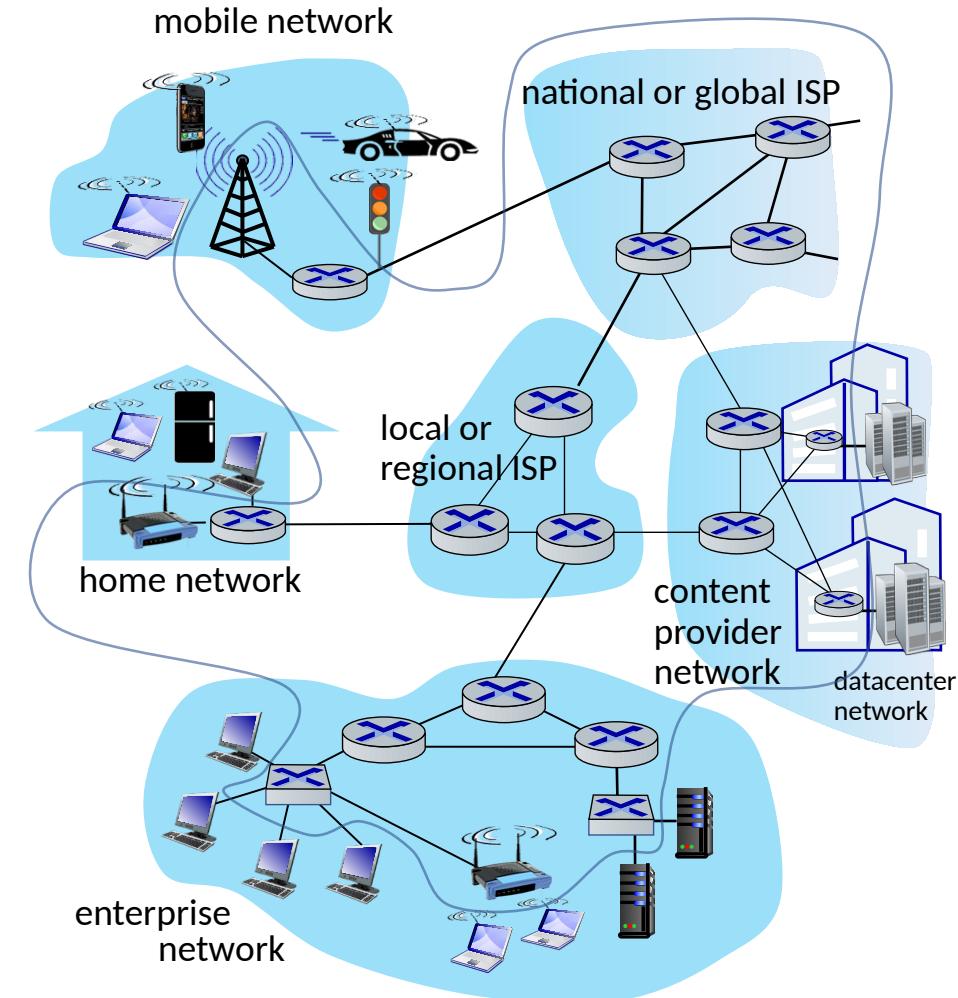
- routers, switches

Communication links

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

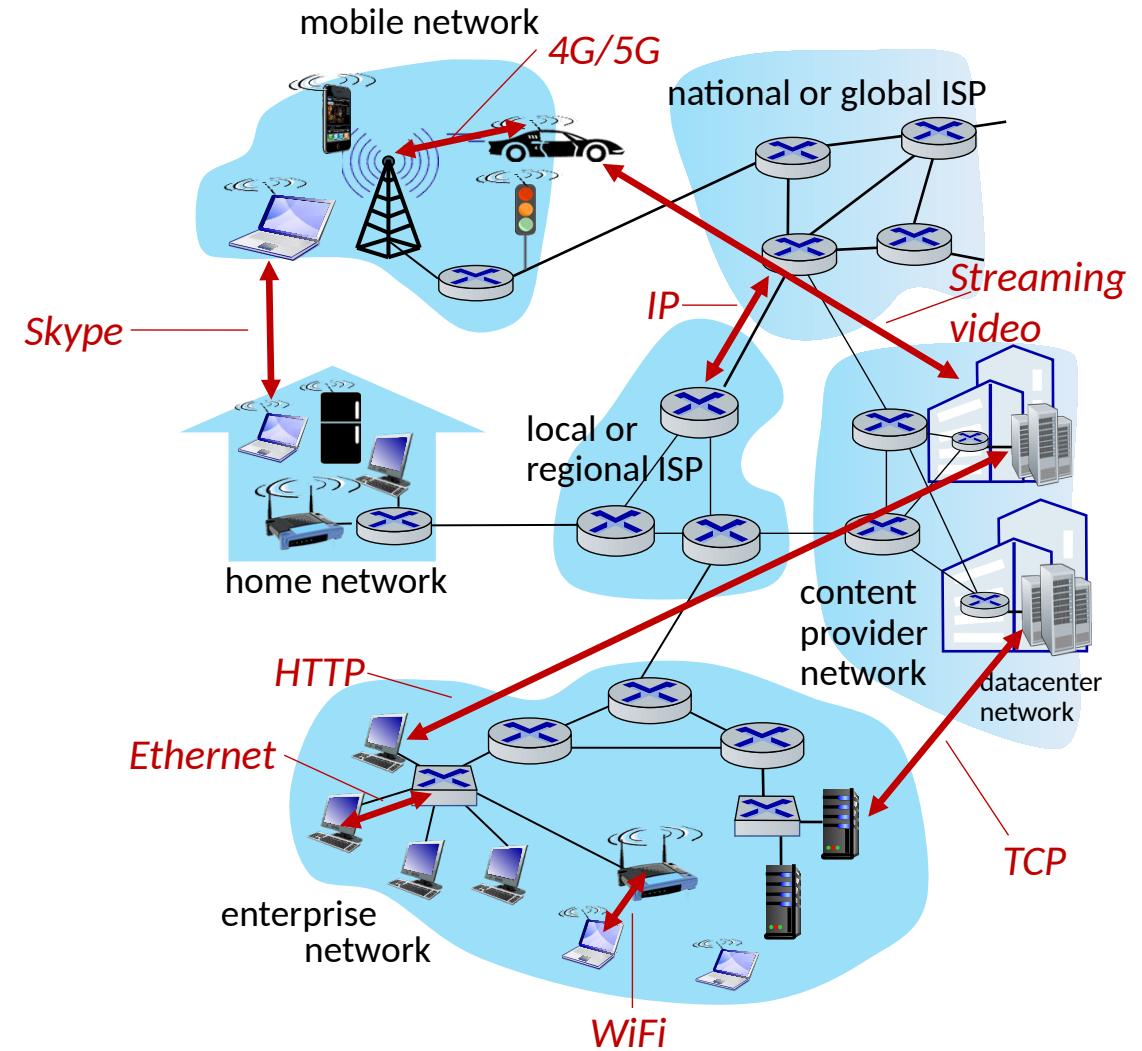
Networks

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



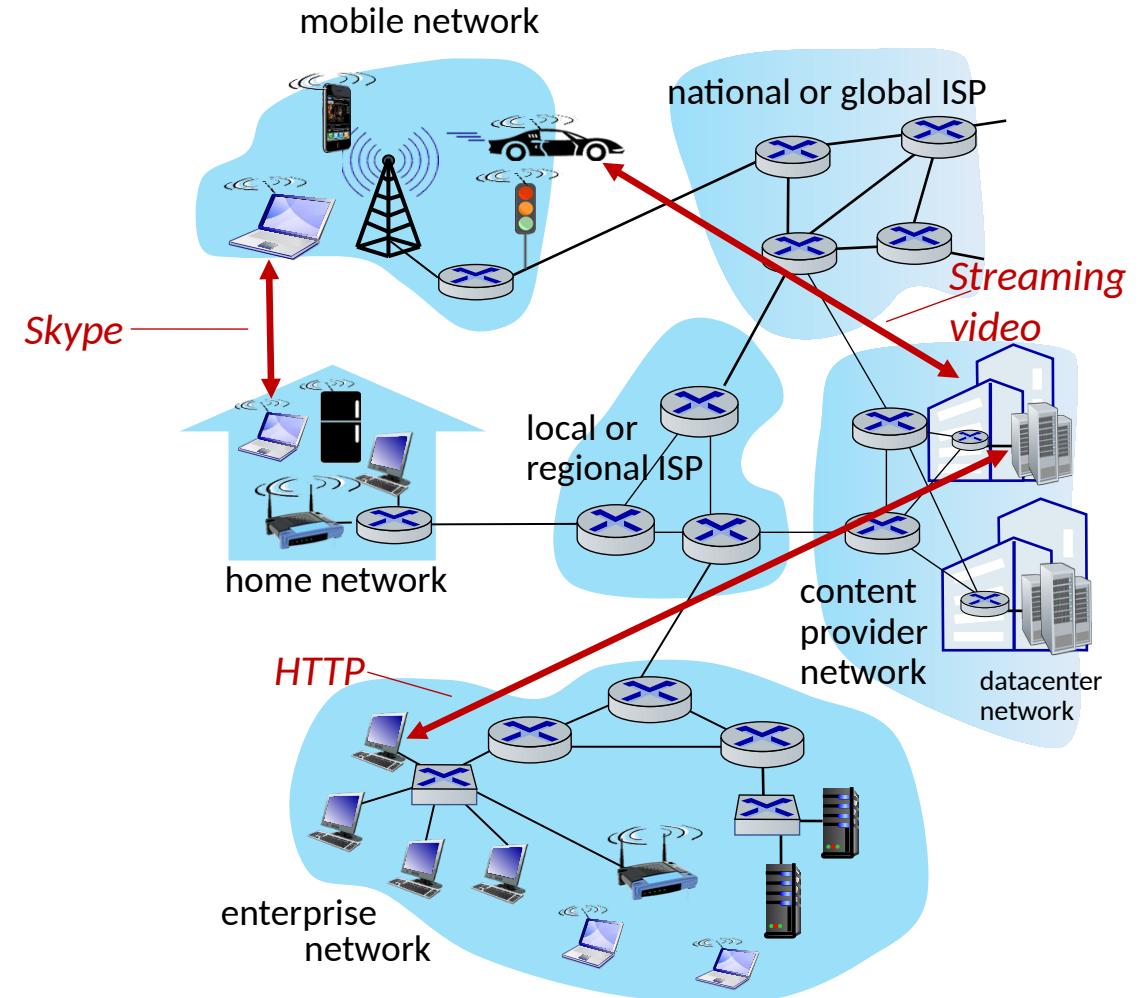
The Internet: a “nuts and bolts” view

- *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

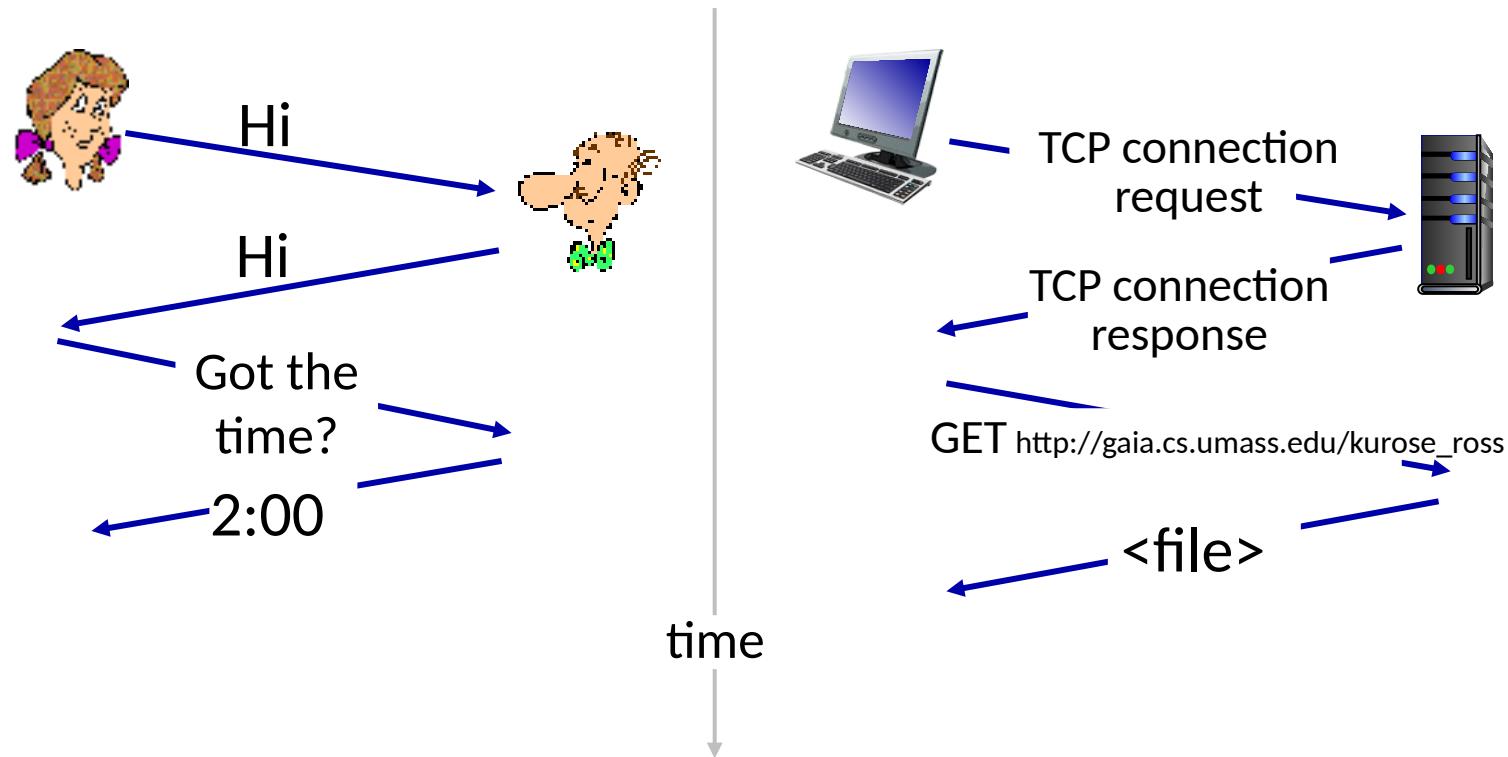
Network protocols:

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

*Protocols define the **format** and the **order of messages exchanged** between two or more communicating entities, as well as the **actions taken** on the transmission and/or receipt of a message or other event*

What's a protocol?

A human protocol and a computer network protocol:



Q: other human protocols?

Chapter 1: roadmap

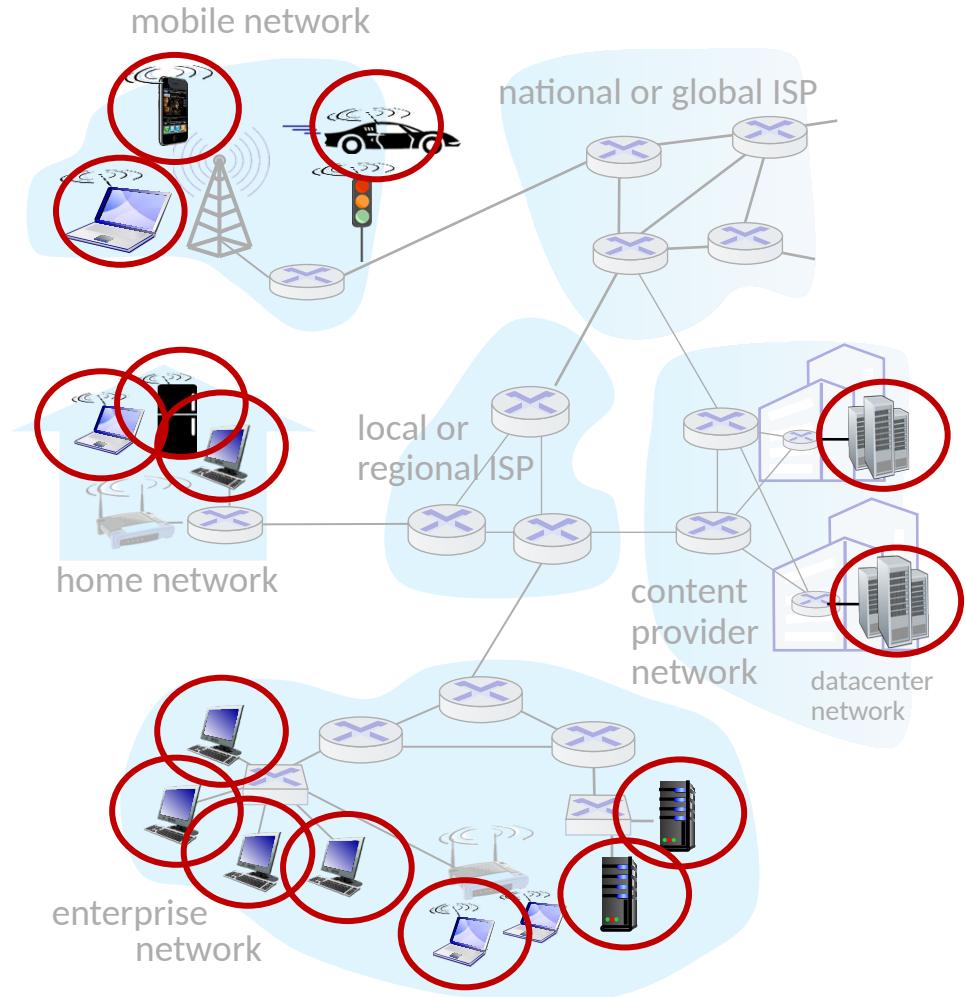
- What *is* the Internet?
- What *is* a protocol?
- **Network edge:** hosts, access network, physical media
- Network core: packet/circuit switching, internet structure
- Performance: loss, delay, throughput
- Security
- Protocol layers, service models
- History



A closer look at Internet structure

Network edge:

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



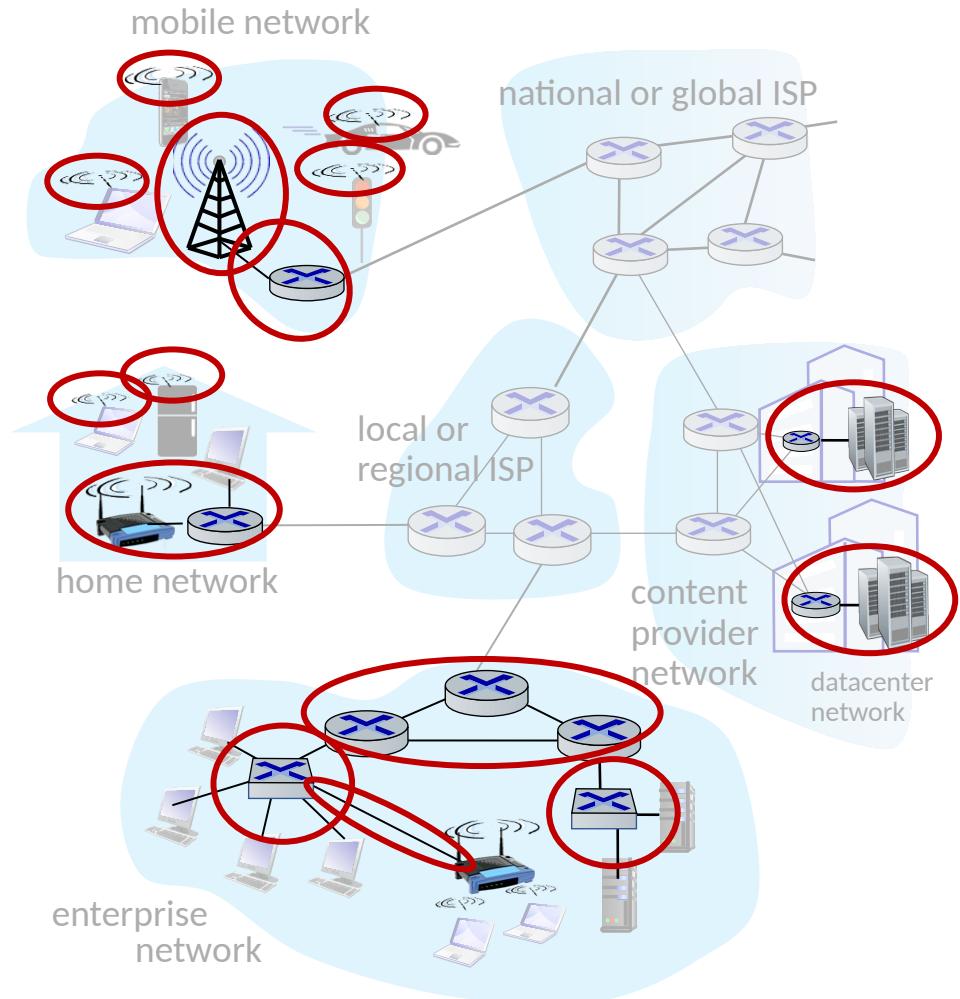
A closer look at Internet structure

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:

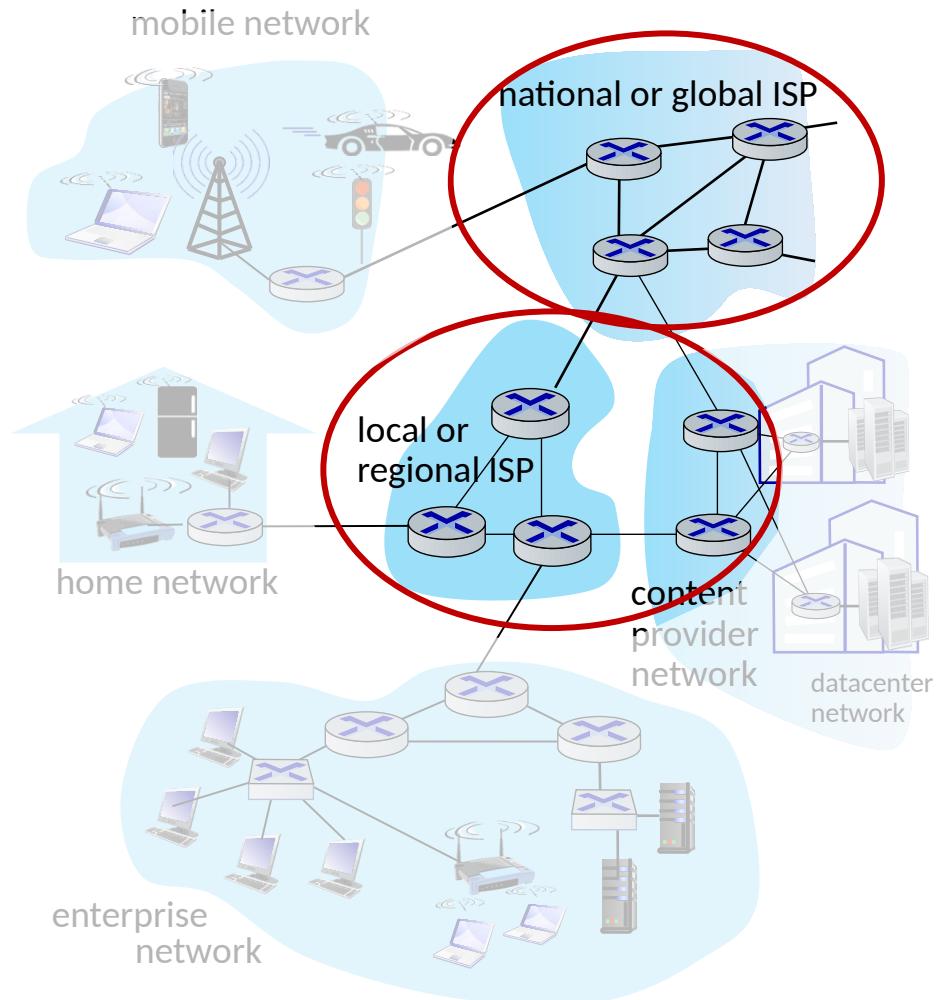
- hosts: clients and servers
- servers often in data centers

Access networks, physical media:

- wired, wireless communication links

Network core:

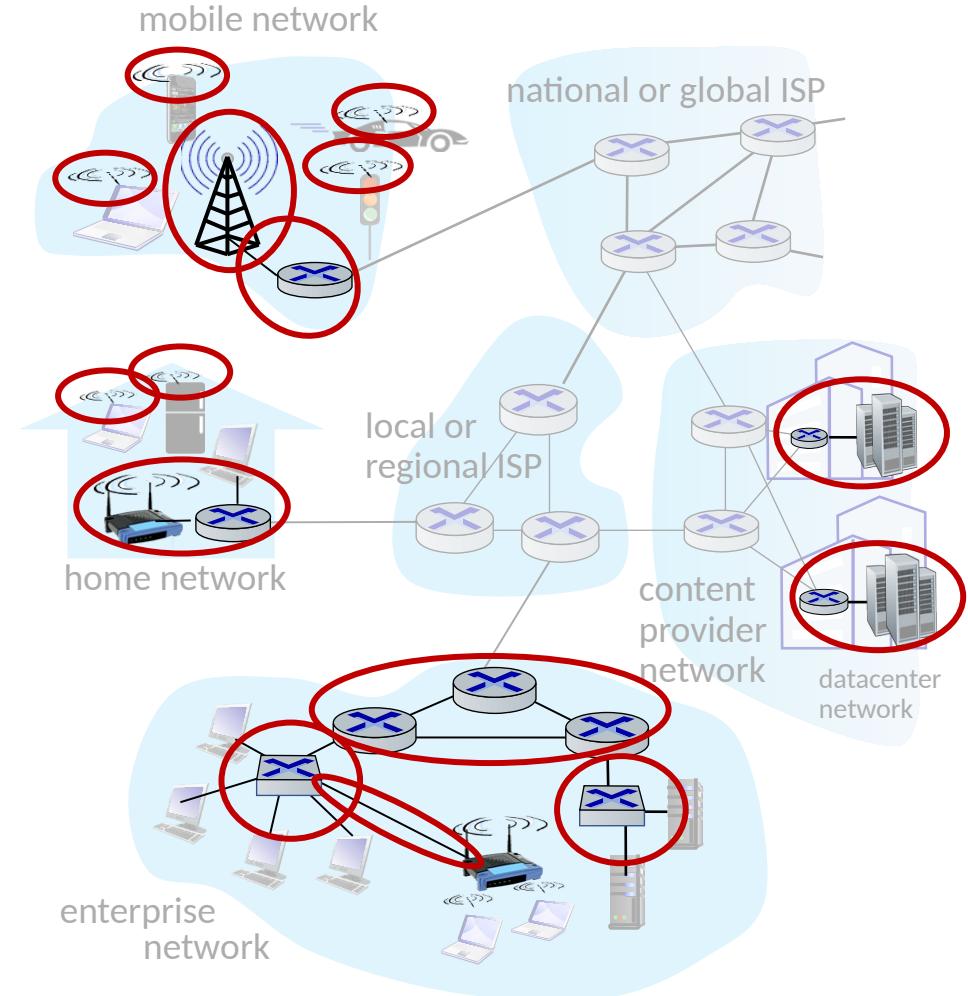
- interconnected routers
- network of networks



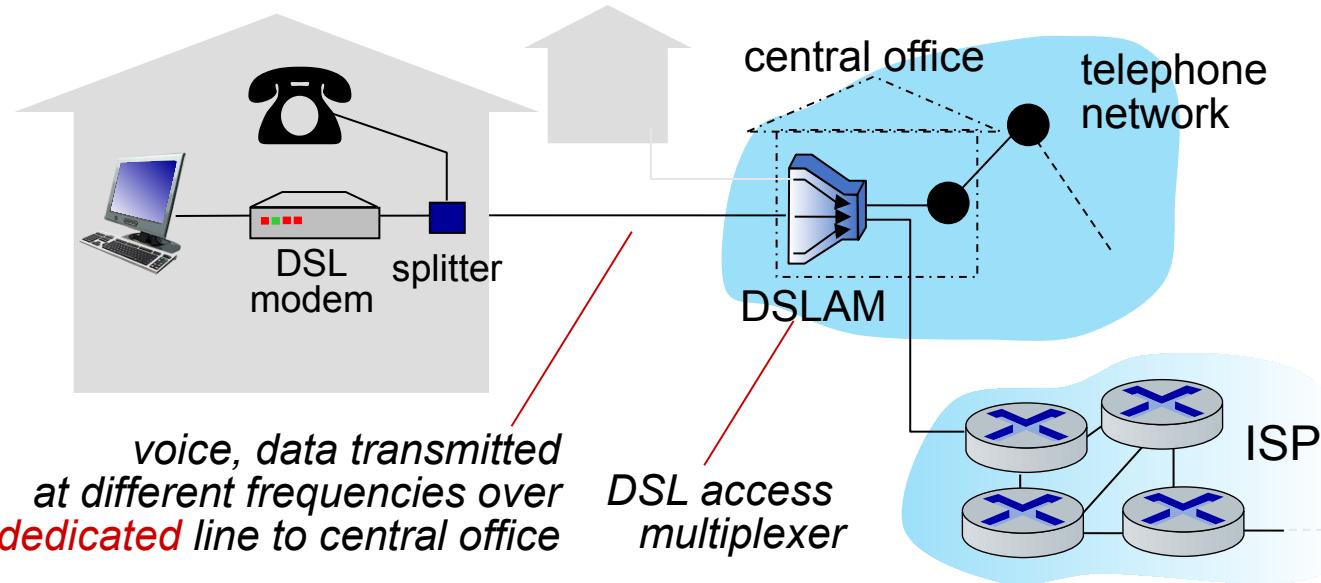
Access networks and physical media

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

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

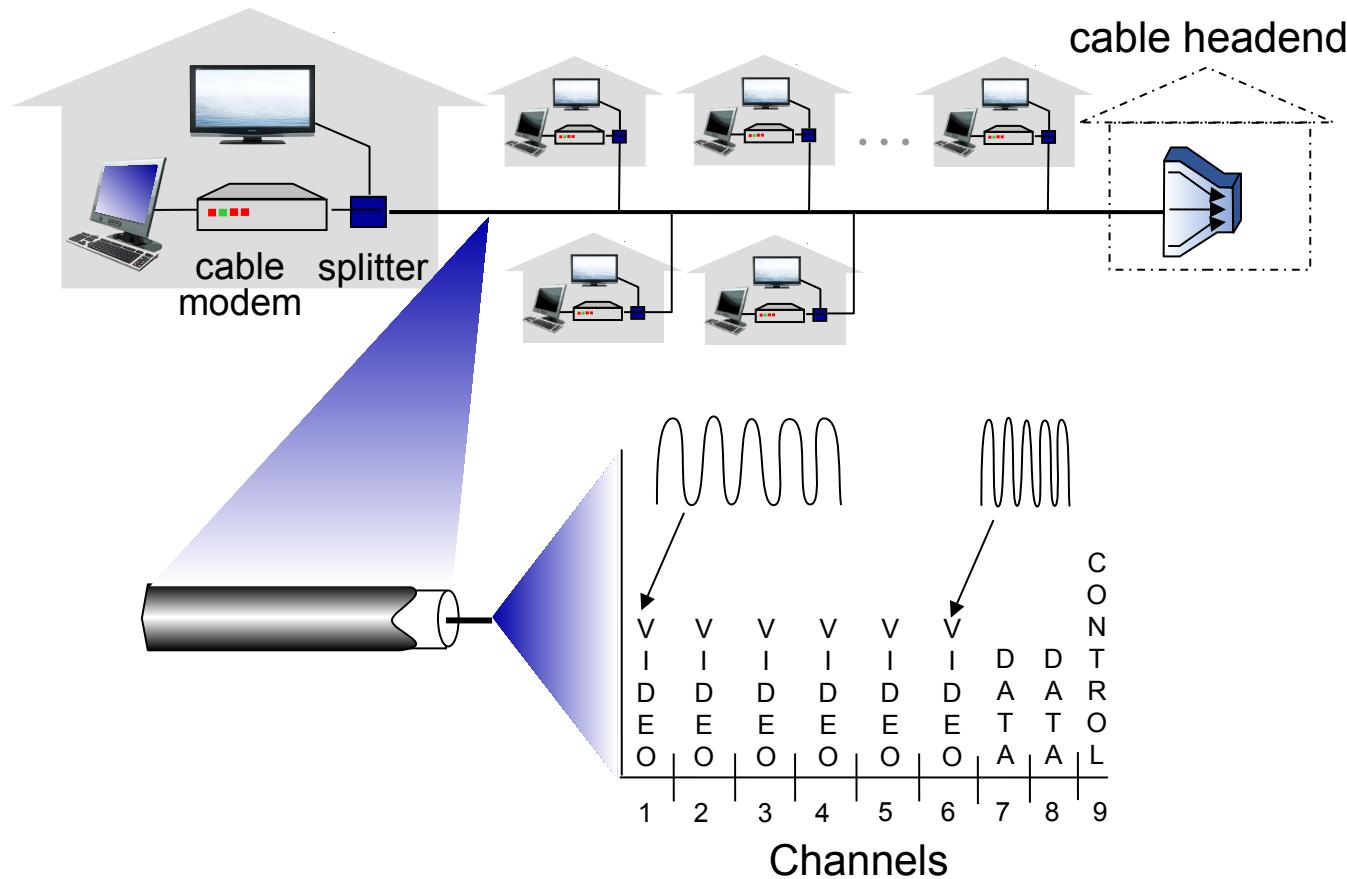


Access networks: digital subscriber line (DSL)



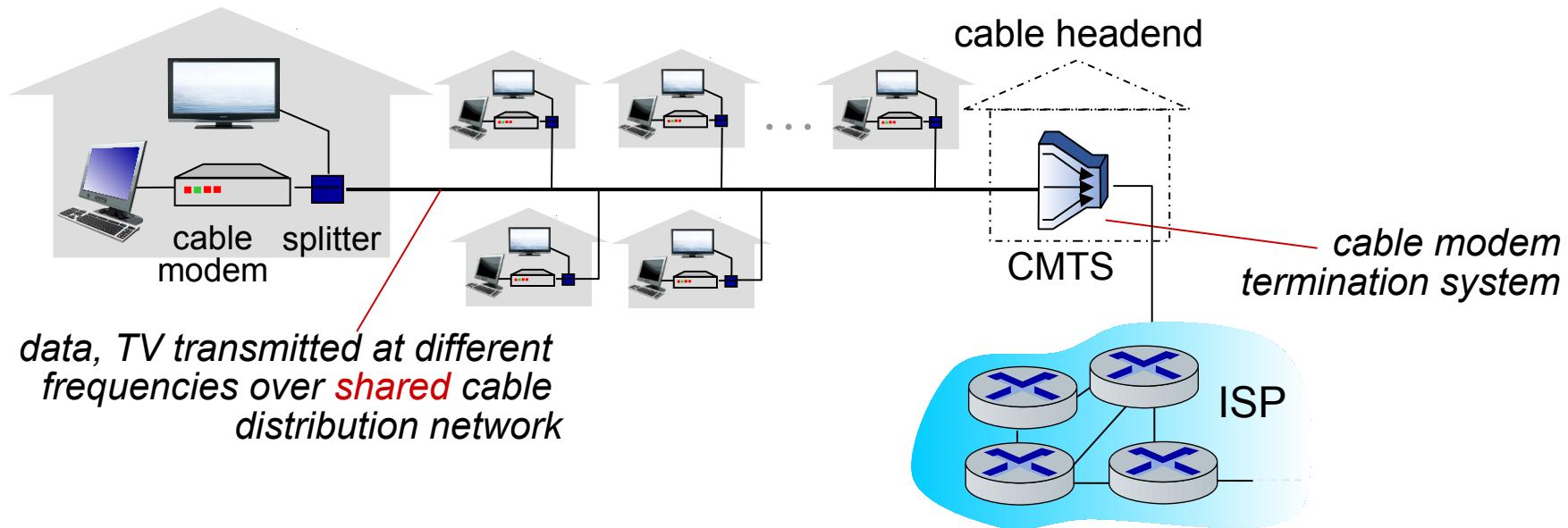
- use **existing** telephone line to central office DSLAM
 - data over DSL phone line goes to Internet
 - voice over DSL phone line goes to telephone net
- 24-52 Mbps dedicated downstream transmission rate
- 3.5-16 Mbps dedicated upstream transmission rate

Access networks: cable-based access



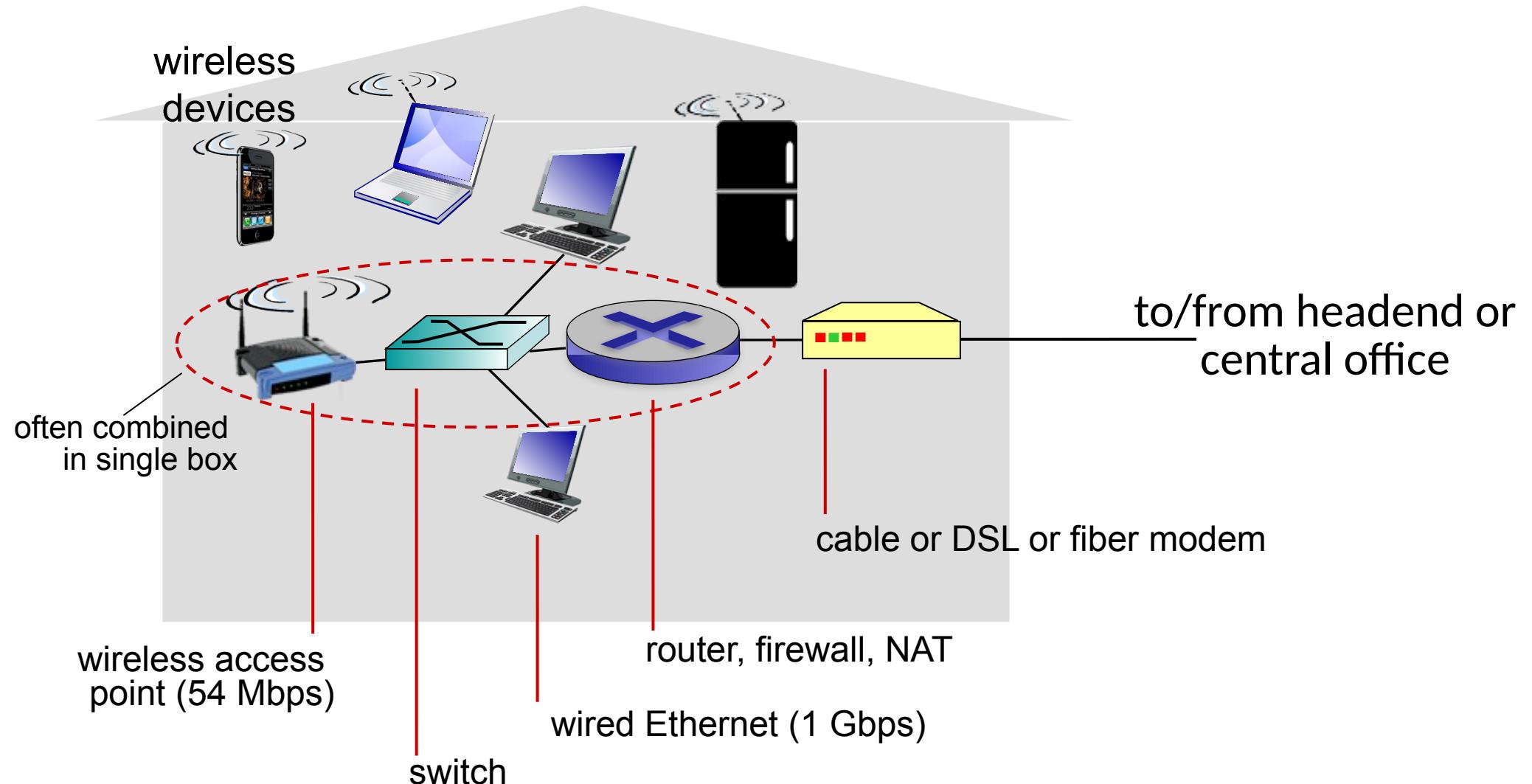
frequency division multiplexing (FDM): different channels transmitted in different frequency bands

Access networks: cable-based access



- HFC: hybrid fiber coax
 - asymmetric: up to 40 Mbps - 1.6 Gbps downstream transmission rate, 30 Mbps - 1 Gbps upstream transmission rate, per channel
- network of cable, fiber attaches homes to ISP router
 - homes **share access network** to cable headend

Access networks: home networks



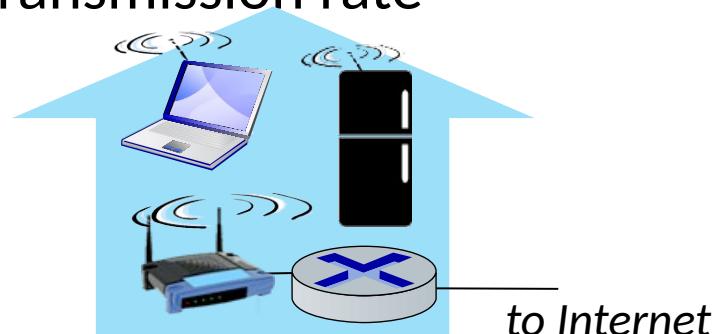
Wireless access networks

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 (~30 m)
- IEEE 802.11b/g/n/ac/ax (Wi-Fi): 11, 54, 450, 1730, 2400 Mbps transmission rate

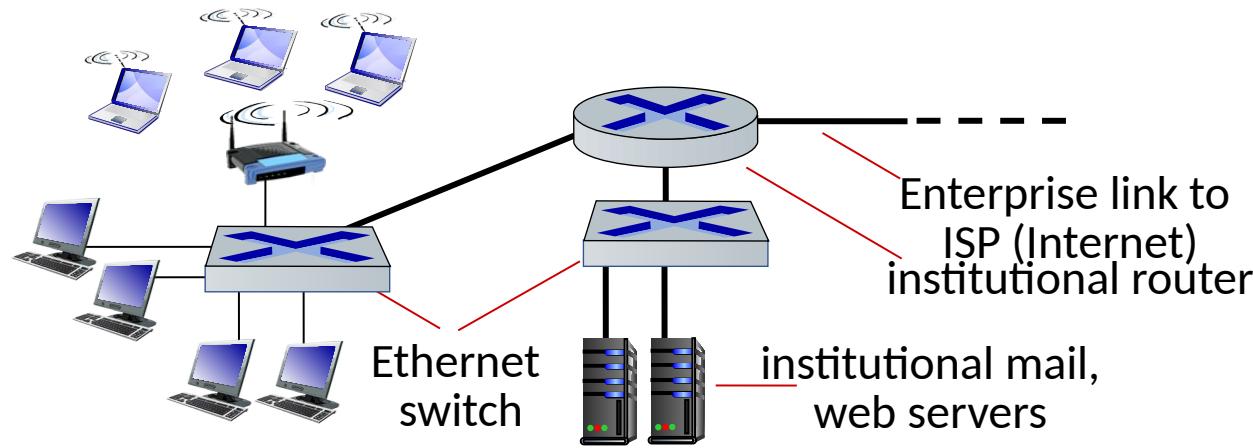


Wide-area cellular access networks

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



Access networks: enterprise networks



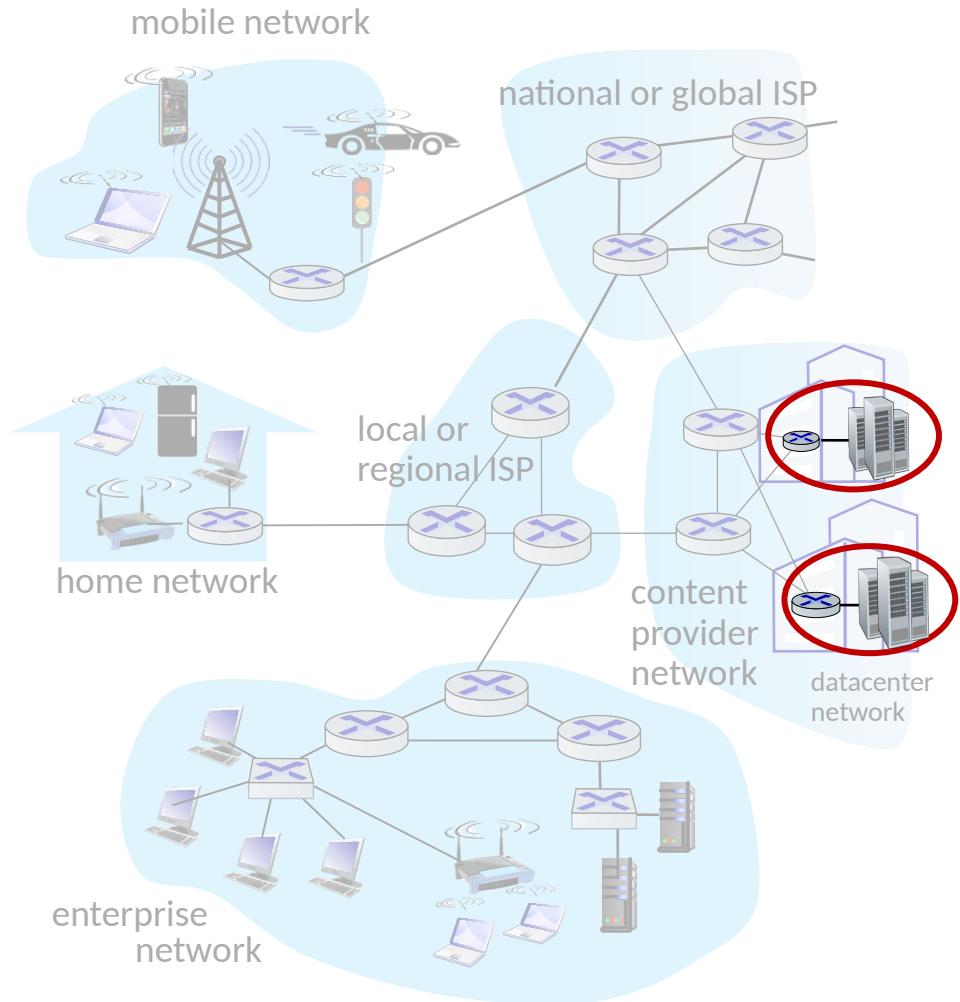
- companies, universities, etc.
- mix of wired, wireless link technologies, connecting a mix of switches and routers (we'll cover differences shortly)
 - Ethernet: wired access at 100Mbps, 1Gbps, 10Gbps
 - WiFi: wireless access points at 11, 54, 450 Mbps

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)



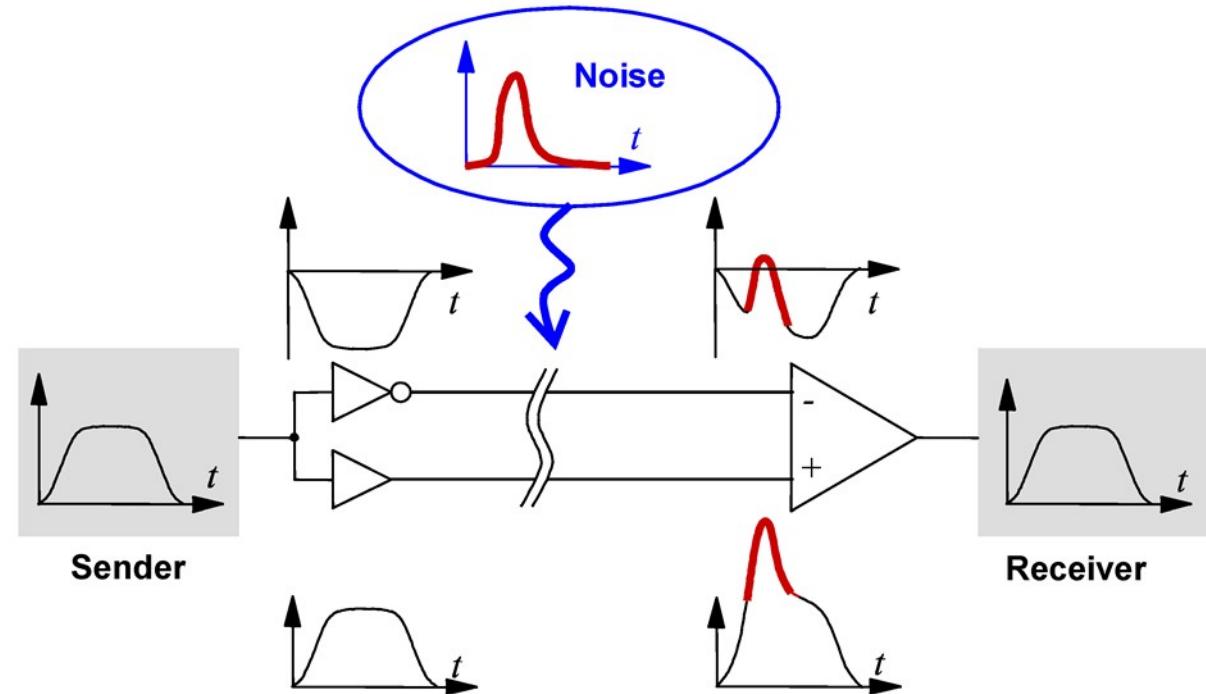
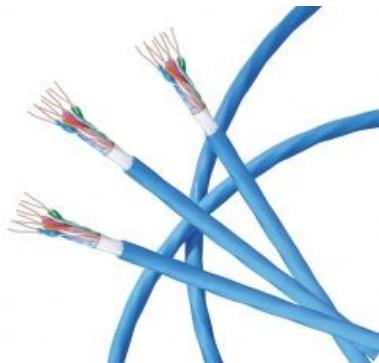
Links: physical media

- **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

Links: physical media (twisted pair)

Twisted pair (TP)

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



<https://resourcespcb.cadence.com/blog/differential-signaling-demystified>

Links: physical media (coax. fiber)

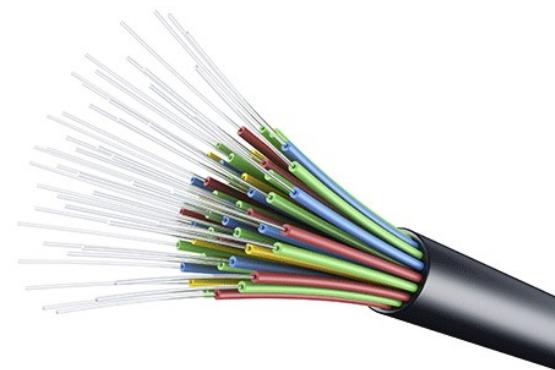
Coaxial cable:

- 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 (radio)

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 (Wi-Fi)**
 - 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

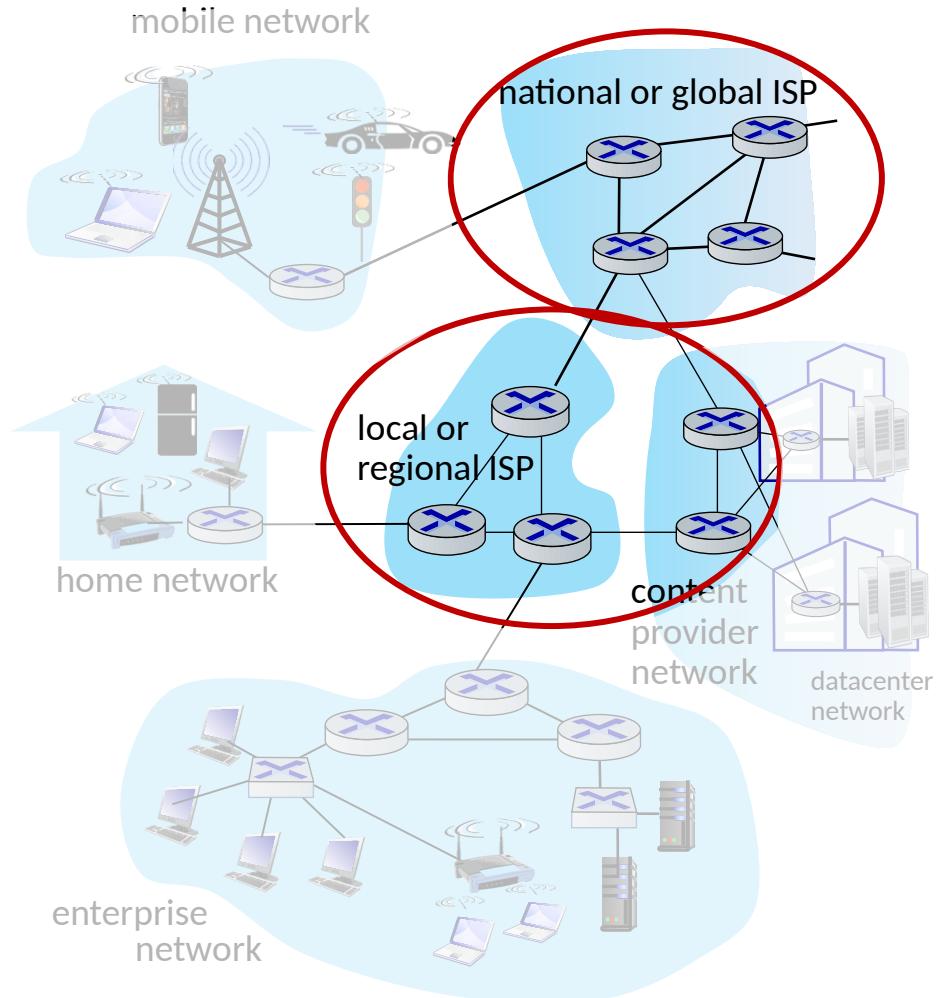
Chapter 1: roadmap

- 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

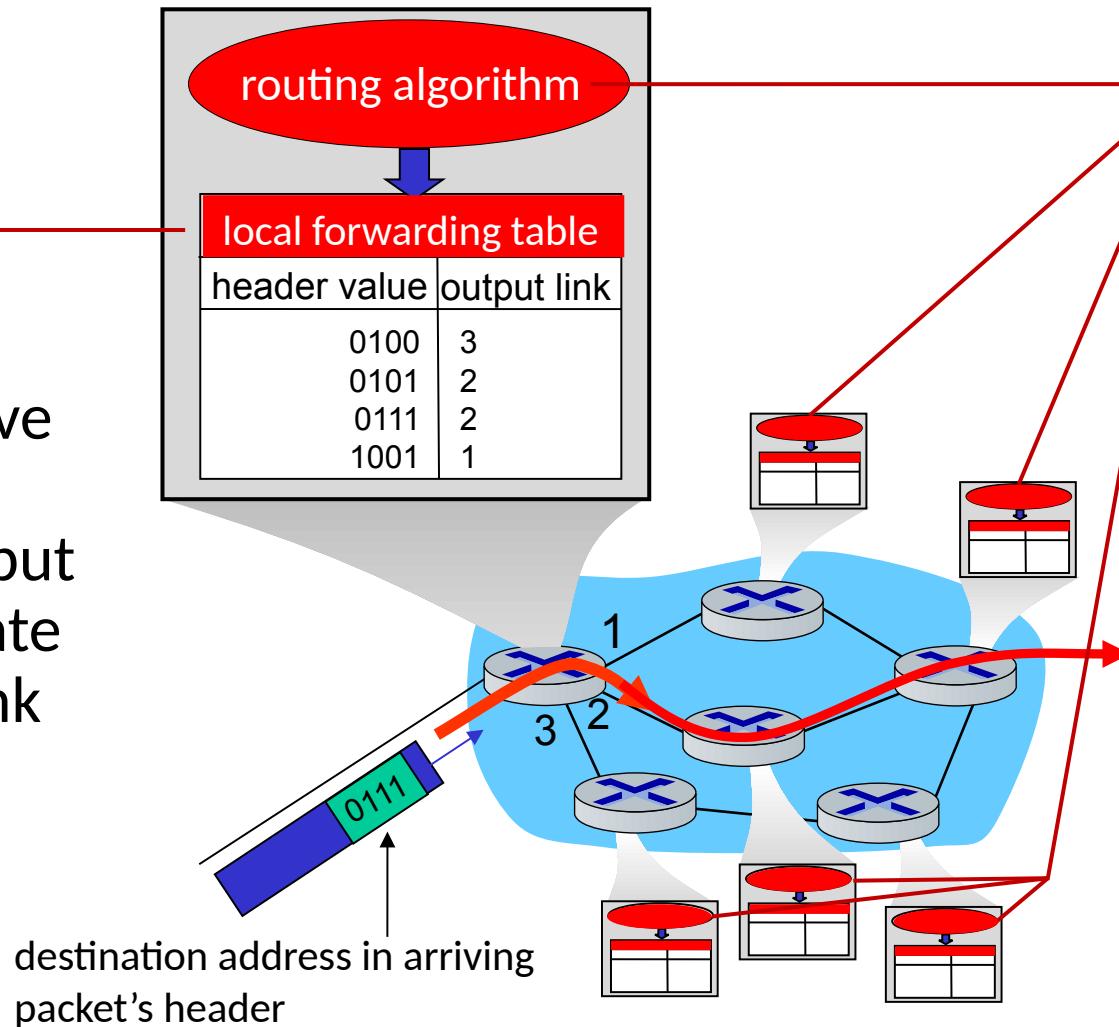
- mesh of interconnected routers
- **packet-switching**: hosts break application-layer messages into *packets*
 - network **forwards** packets from one router to the next, across links on path from **source to destination**



Two key network-core functions

Forwarding:

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



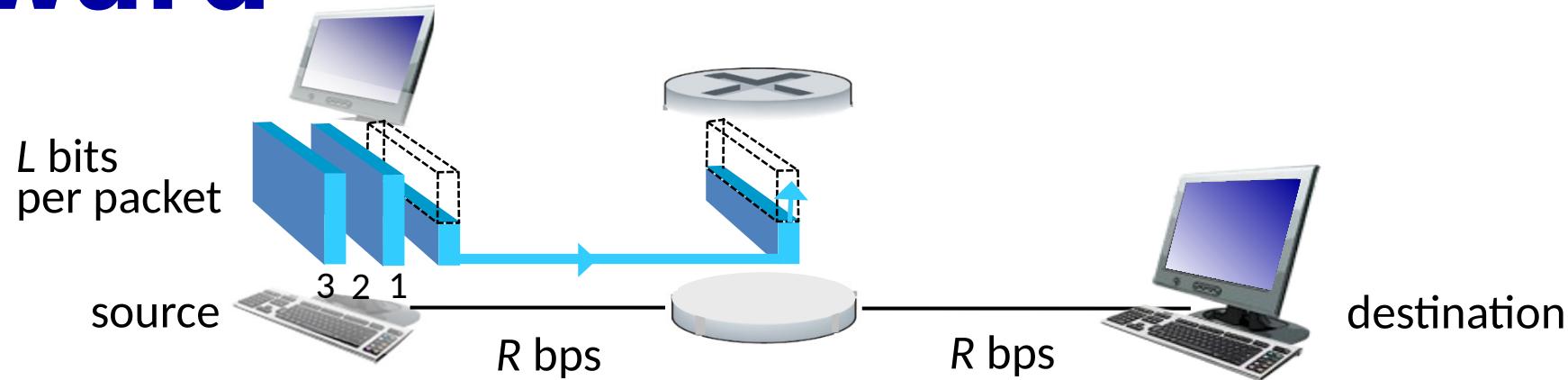
Routing:

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





Packet-switching: store-and-forward

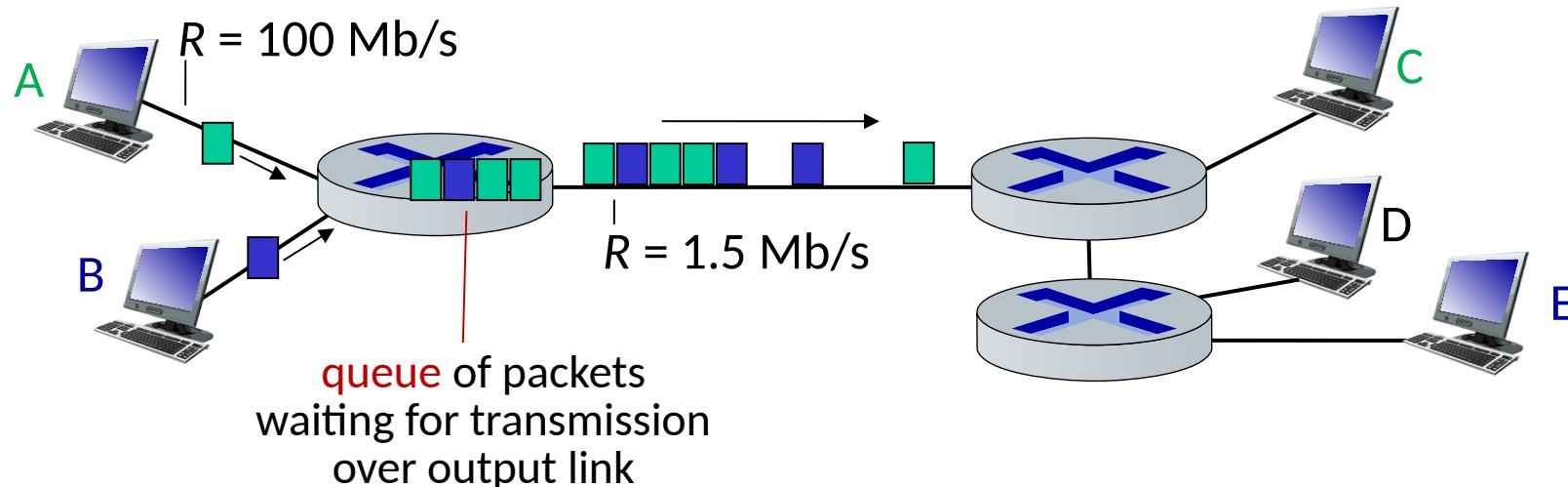


- **packet transmission delay:** takes L/R seconds to transmit (push out) L -bit packet into link at R bps
- **store and forward:** entire packet must arrive at router before it can be transmitted on next link

One-hop numerical example:

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

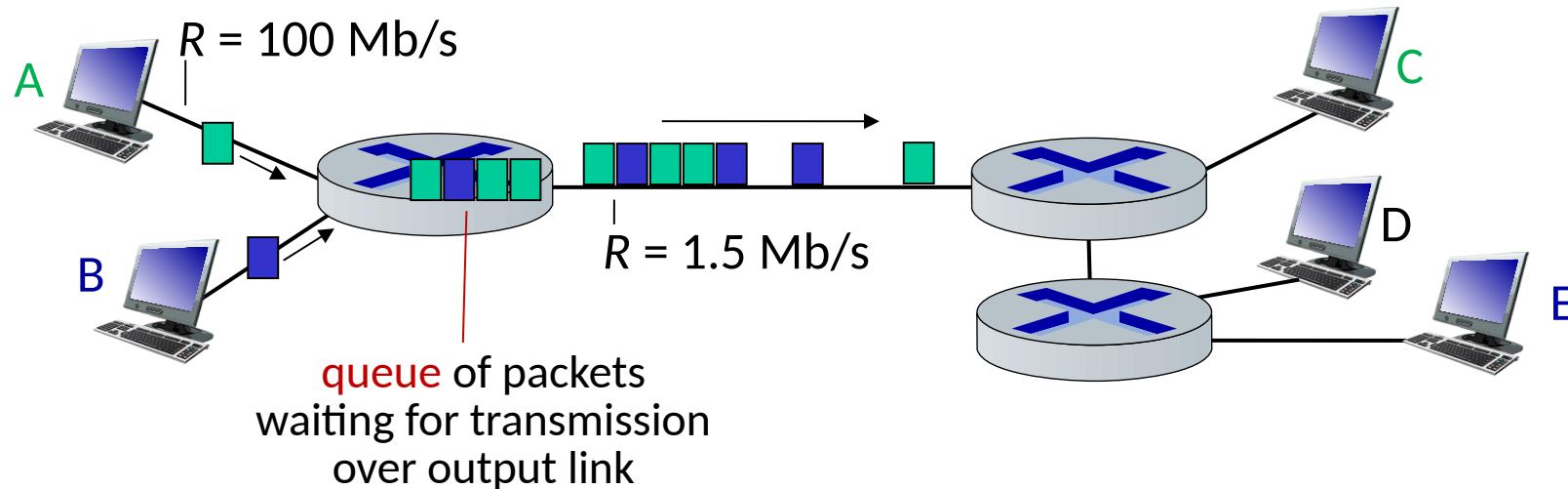
Packet-switching: queueing



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



Packet-switching: queueing



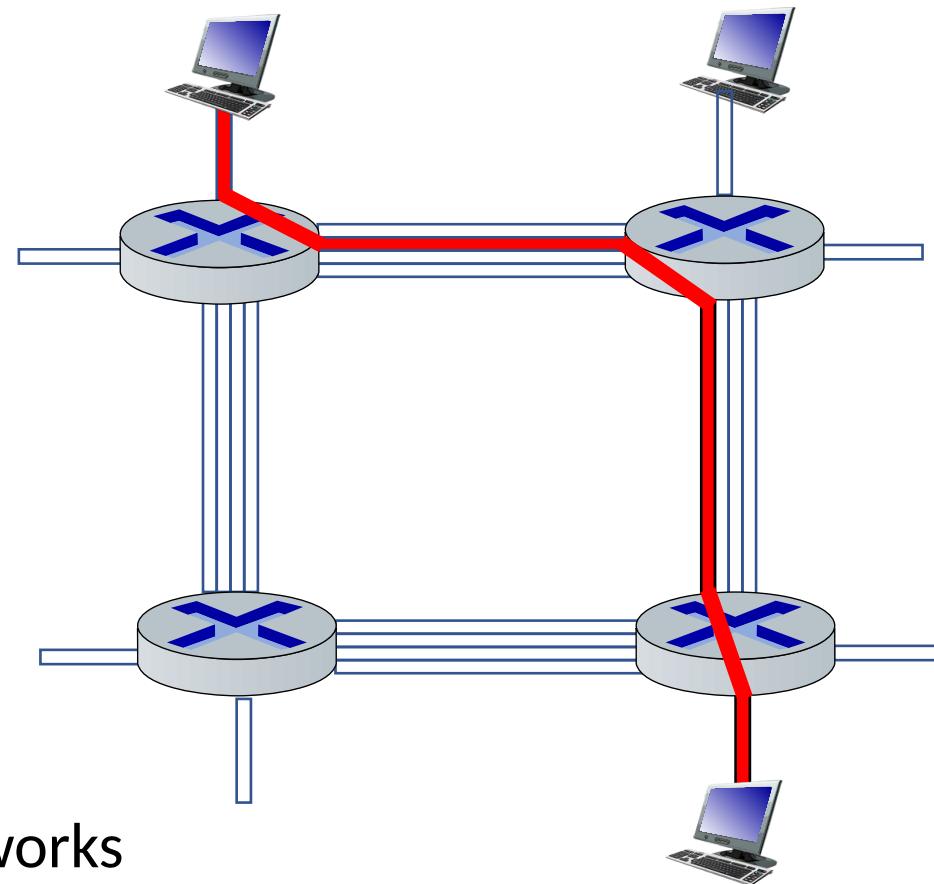
Packet queuing and loss: if arrival rate (in bps) to link exceeds transmission rate (bps) of link for some period of time:

- 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 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



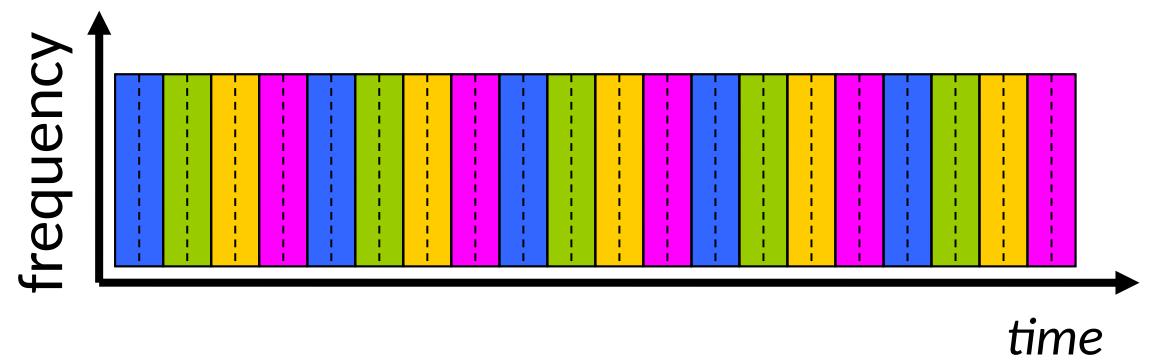
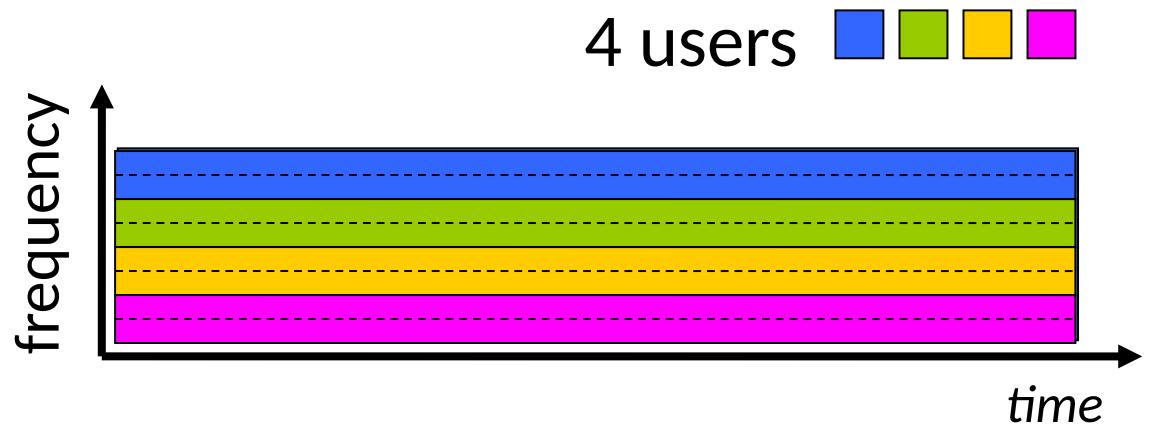
Circuit switching: FDM and TDM

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)

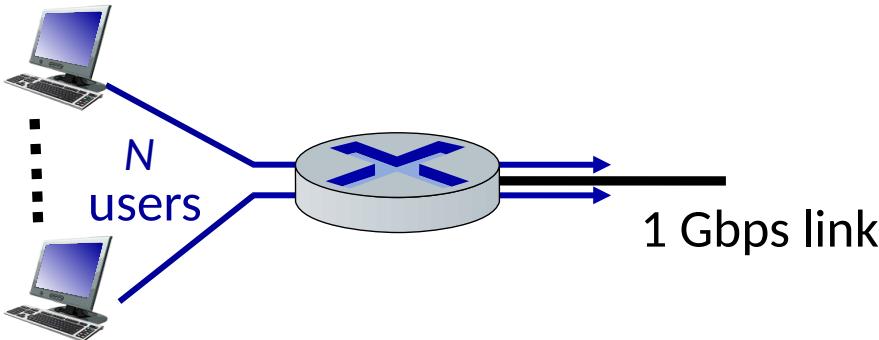
- time divided into slots
- each call allocated periodic slot(s), can transmit at maximum rate of (wider) frequency band (only) during its time slot(s)



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)

Packet switching versus circuit switching

En binomisk fordeling eller binomialfordeling er en diskret fordeling (et begrep innen sannsynlighetsteori og matematisk statistikk) som håndterer hyppige (diskrete) forsøk med fast sannsynlighet.

Dersom en stokastisk variabel X er binomisk fordelt, med n =totale antall forsøk, k =antall lykkede forsøk og p =sannsynligheten for å lykkes i hvert forsøk, skriver man :

$$X \in Bin(n, p)$$

X har sannsynlighetsfunksjonen

$$p_X(k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

der p er sannsynligheten for at hendelsen skal inntreffe og $1 - p = q$ således sannsynligheten for at hendelsen ikke skal inntreffe. Slik dukker binomialkoeffisientene opp i fordelingen.

p	0,1	Hver bruker aktiv 10 prosent av tiden
n	35	Antall brukere totalt
k		Antall aktive brukere

k	P(k)	P(1 <= X <= k)	P(X >= k)
0	0,02503	0,02503	1,00000
1	0,09734	0,12238	0,97497
2	0,18387	0,30625	0,87762
3	0,22473	0,53098	0,69375
4	0,19976	0,73075	0,46902
5	0,13762	0,86836	0,26925
6	0,07645	0,94482	0,13164
7	0,03519	0,98001	0,05518
8	0,01369	0,99370	0,01999
9	0,00456	0,99826	0,00630
10	0,00132	0,99958	0,00174
11	0,00033	0,99991	0,00042 0,042 % sannsynlighet for 11 eller flere aktive bruker
12	0,00007	0,99998	0,00009
13	0,00001	1,00000	0,00002

Packet switching versus circuit switching

En binomisk fordeling eller binomialfordeling er en **diskret fordeling** (et begrep innen sannsynlighetsteori og matematisk statistikk) som håndterer hyppige (diskrete) forsøk med fast **sannsynlighet**.

Dersom en stokastisk variabel X er binomisk fordelt, med n =totale antall forsøk, k =antall lykkede forsøk og p =sannsynligheten for å lykkes i hvert forsøk, skriver man :

$$X \in Bin(n, p)$$

X har **sannsynlighetsfunksjonen**

$$p_X(k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

der p er sannsynligheten for at **hendelsen** skal inntreffe og $1 - p = q$ således sannsynligheten for at hendelsen ikke skal inntreffe. Slik dukker **binomialkoeffisientene** opp i fordelingen.

p	0,1	Hver bruker aktiv 10 prosent av tiden
n	70	Antall brukere totalt
k		Antall aktive brukere

k	$P(k)$	$P(1 \leq X \leq k)$	$P(X \geq k)$
0	0,00063	0,00063	1,00000
1	0,00487	0,00550	0,99937
2	0,01868	0,02418	0,99450
3	0,04705	0,07123	0,97582
4	0,08756	0,15879	0,92877
5	0,12843	0,28722	0,84121
6	0,15459	0,44181	0,71278
7	0,15704	0,59885	0,55819
8	0,13741	0,73626	0,40115
9	0,10518	0,84144	0,26374
10	0,07129	0,91273	0,15856
11	0,04320	0,95594	0,08727 8,727 % sannsynlighet for 11 eller flere aktive bruker
12	0,02360	0,97954	0,04406
13	0,01170	0,99124	0,02046

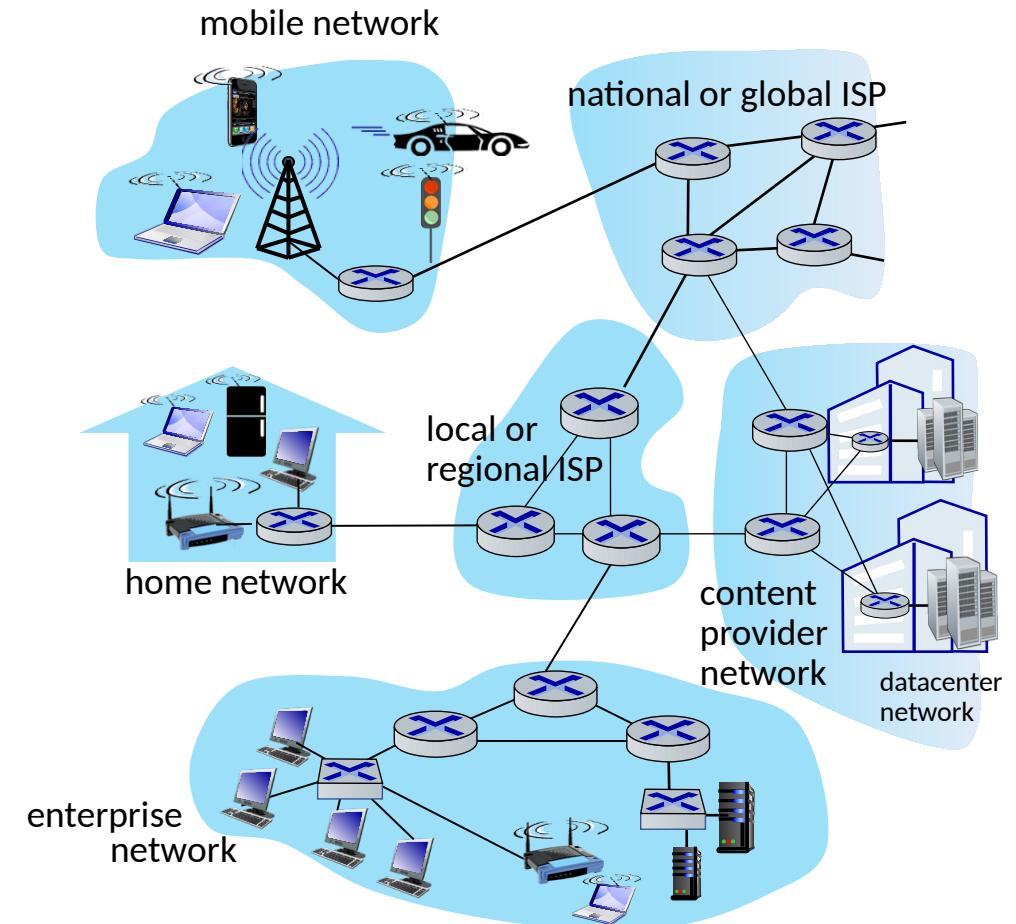
Packet switching versus circuit switching

Is packet switching a “slam dunk winner”?

- great for “bursty” data - sometimes has data to send, but at other times not
 - resource sharing
 - simpler, no call setup
- **excessive congestion possible:** packet delay and loss due to buffer overflow
 - protocols needed for reliable data transfer, congestion control

Internet structure: a “network of networks”

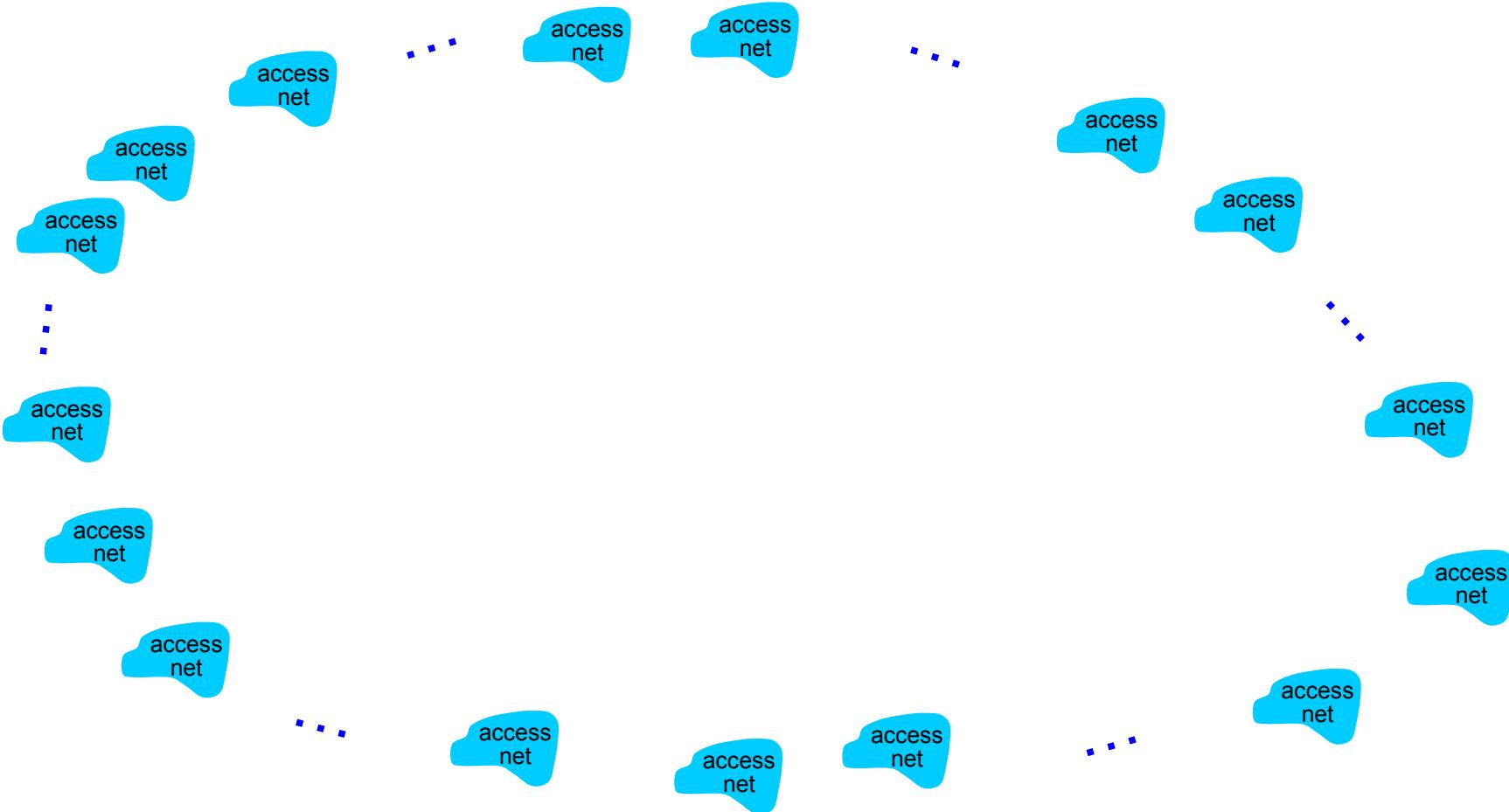
- hosts connect to Internet via **access** Internet Service Providers (ISPs)
- access ISPs in turn must be interconnected
 - so that *any two hosts (anywhere!)* can send packets to each other
- resulting network of networks is very complex
 - evolution driven by **economics, national policies**



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

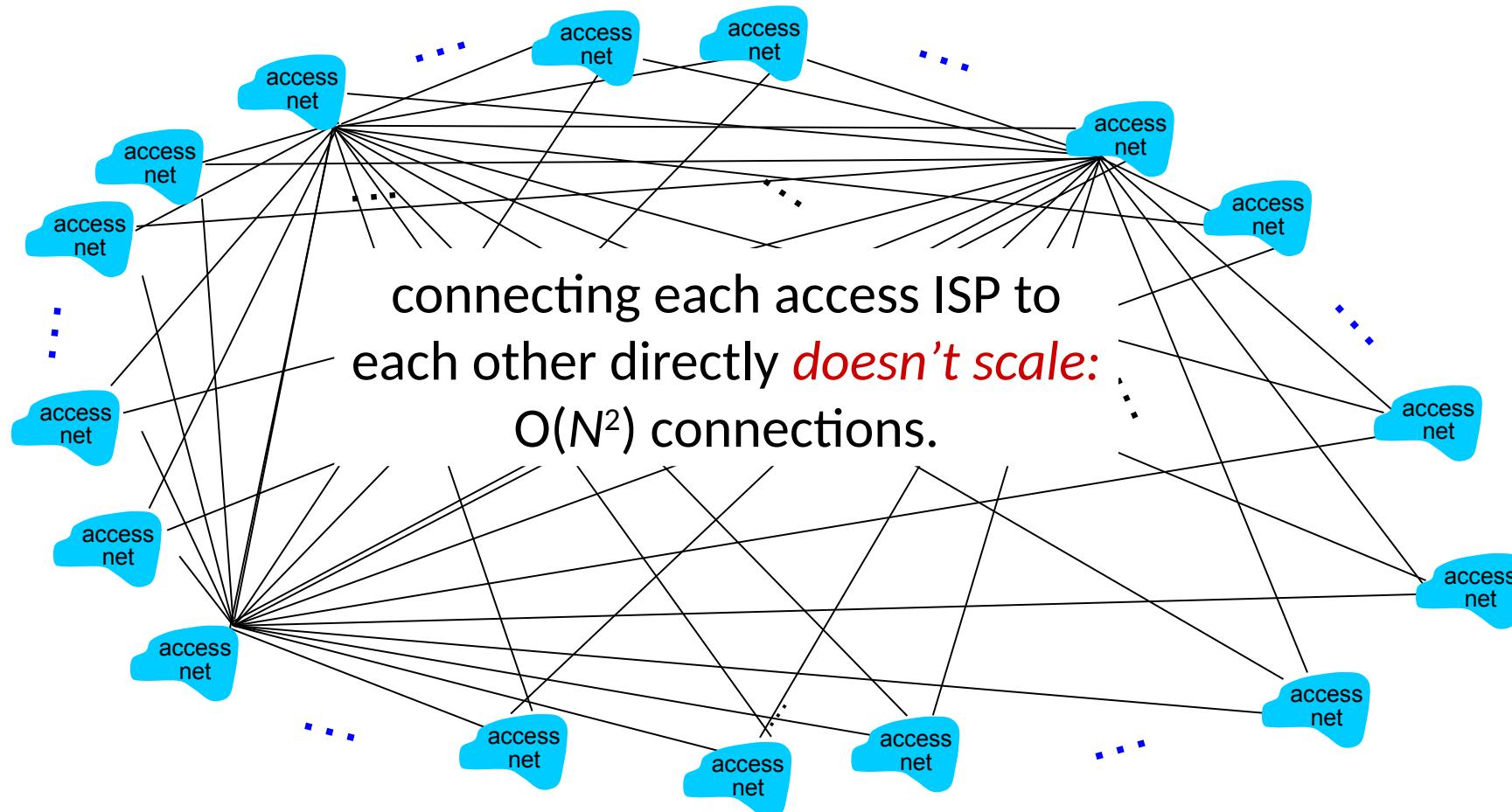
Internet structure: a “network of networks”

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



Internet structure: a “network of networks”

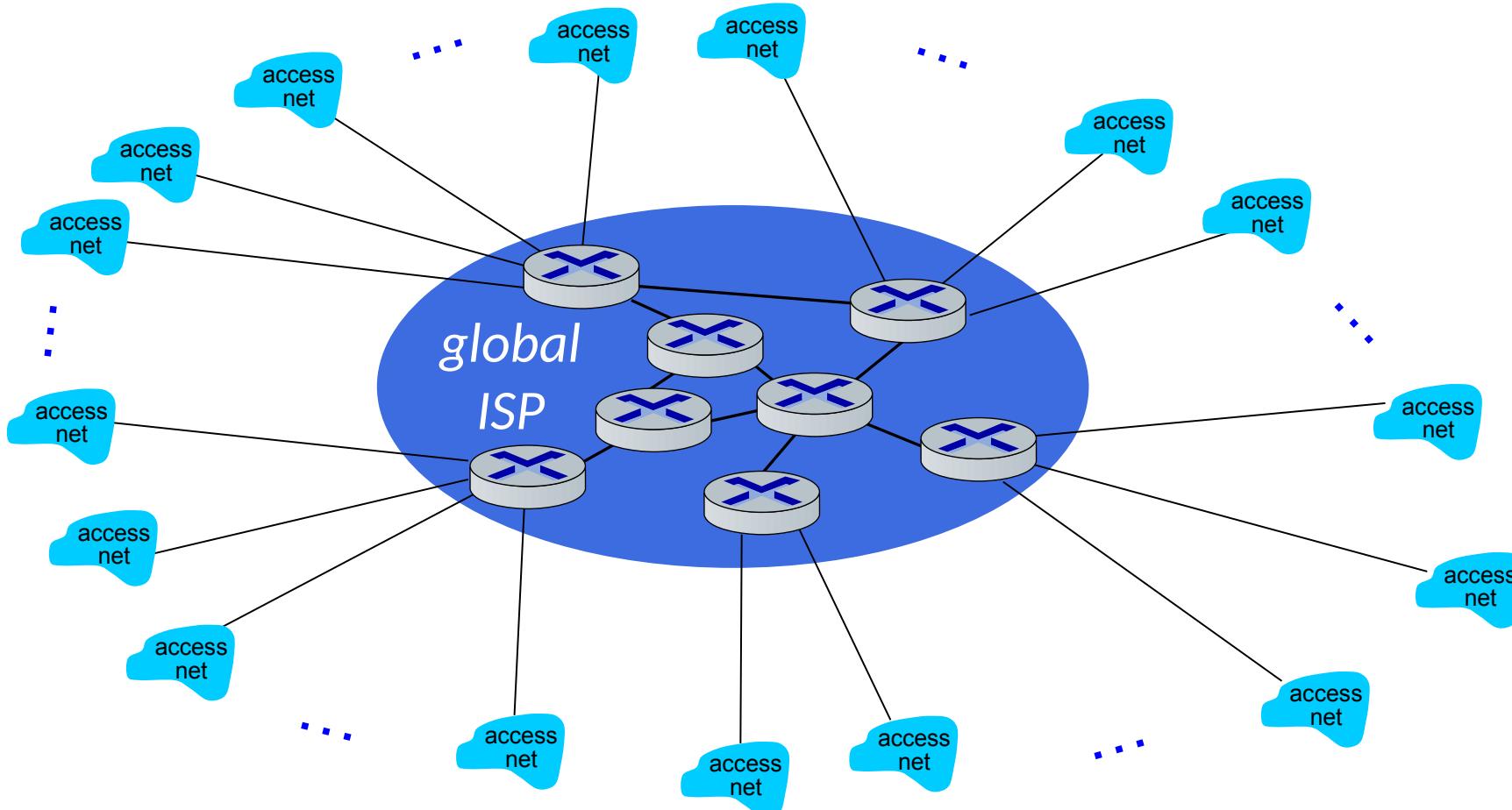
Question: given millions of access ISPs, how to connect them together?



Internet structure: a “network of networks”

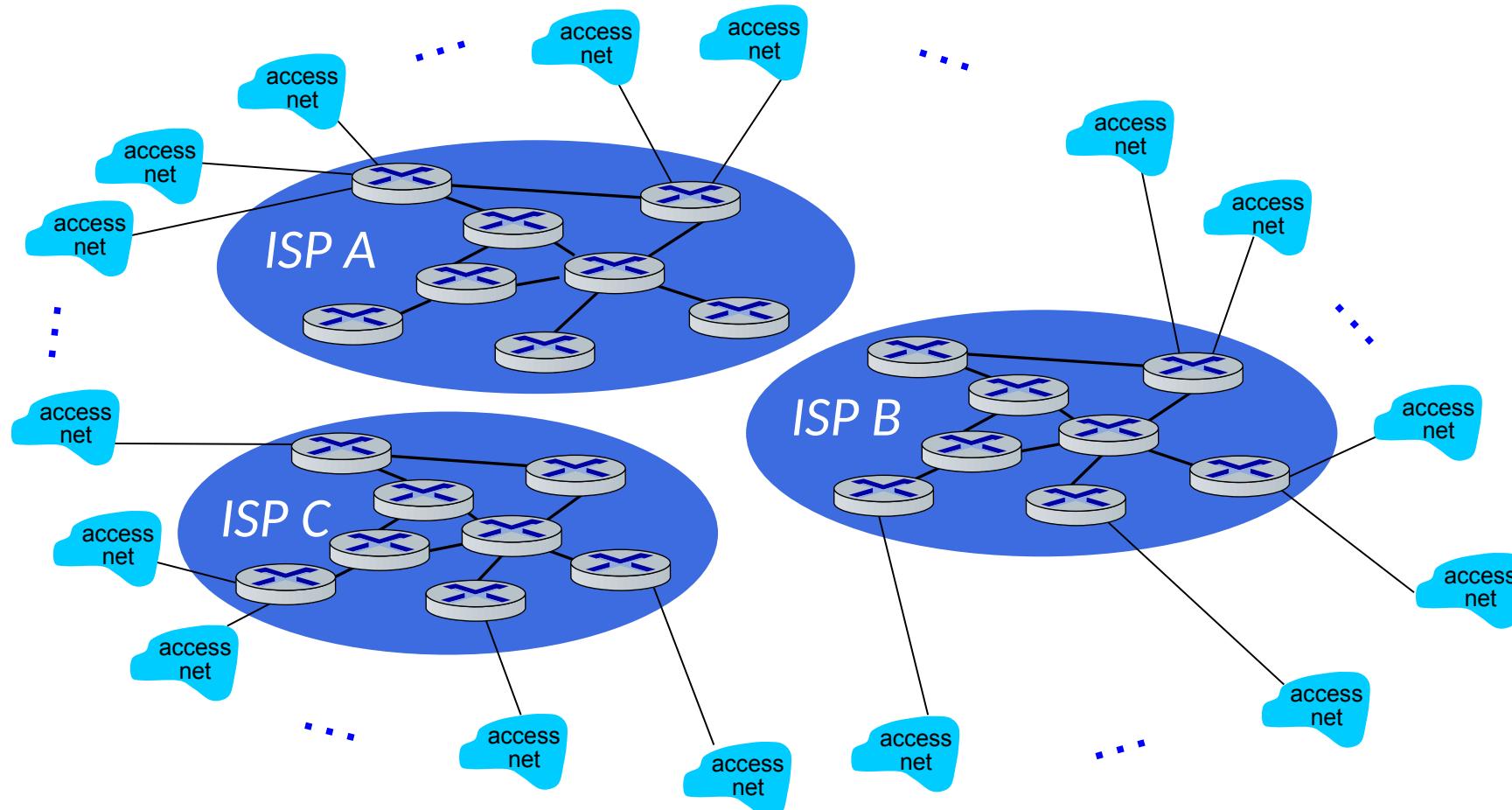
Option: connect each access ISP to one global transit ISP?

Customer and provider ISPs have economic agreement.



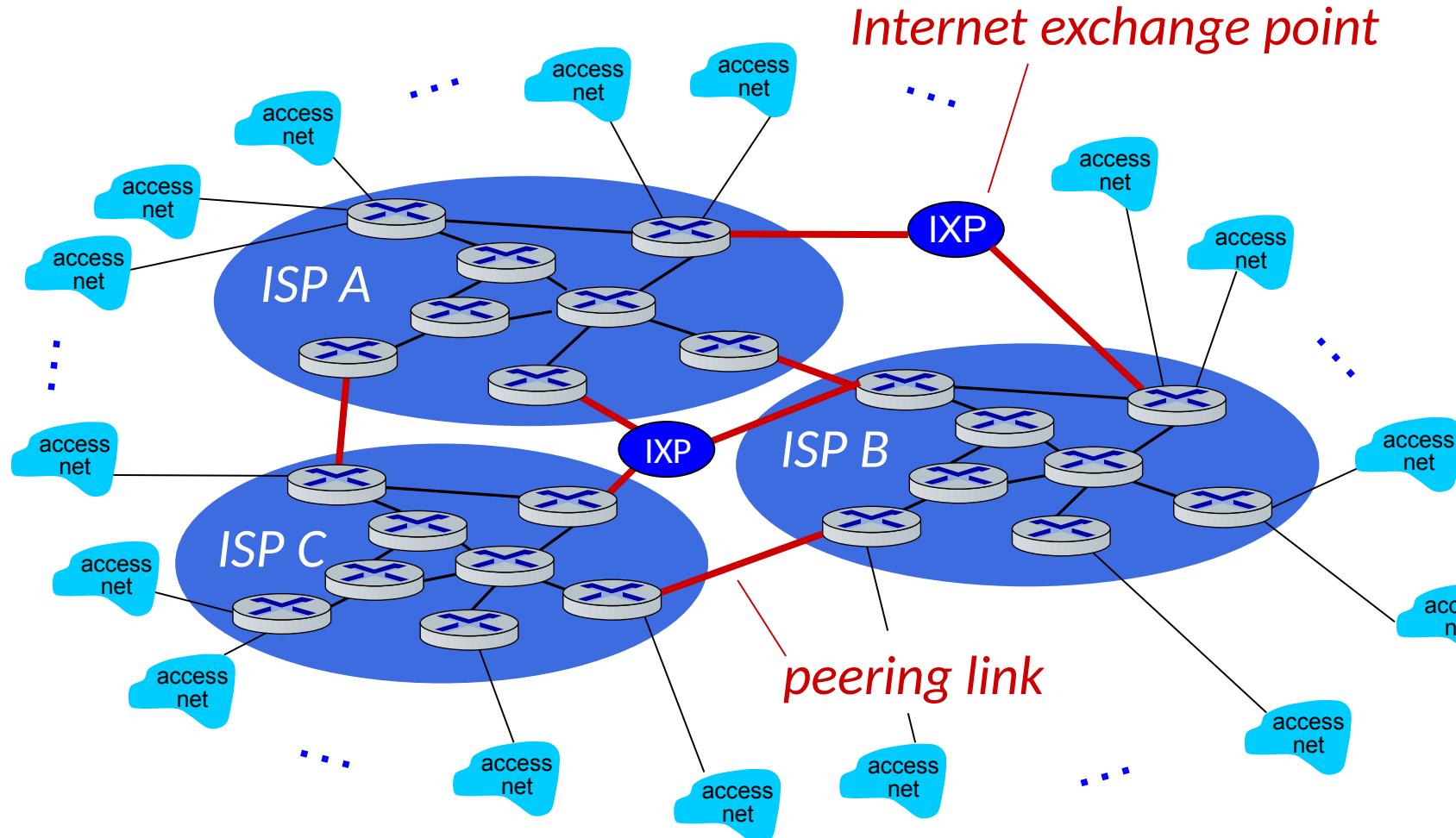
Internet structure: a “network of networks”

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



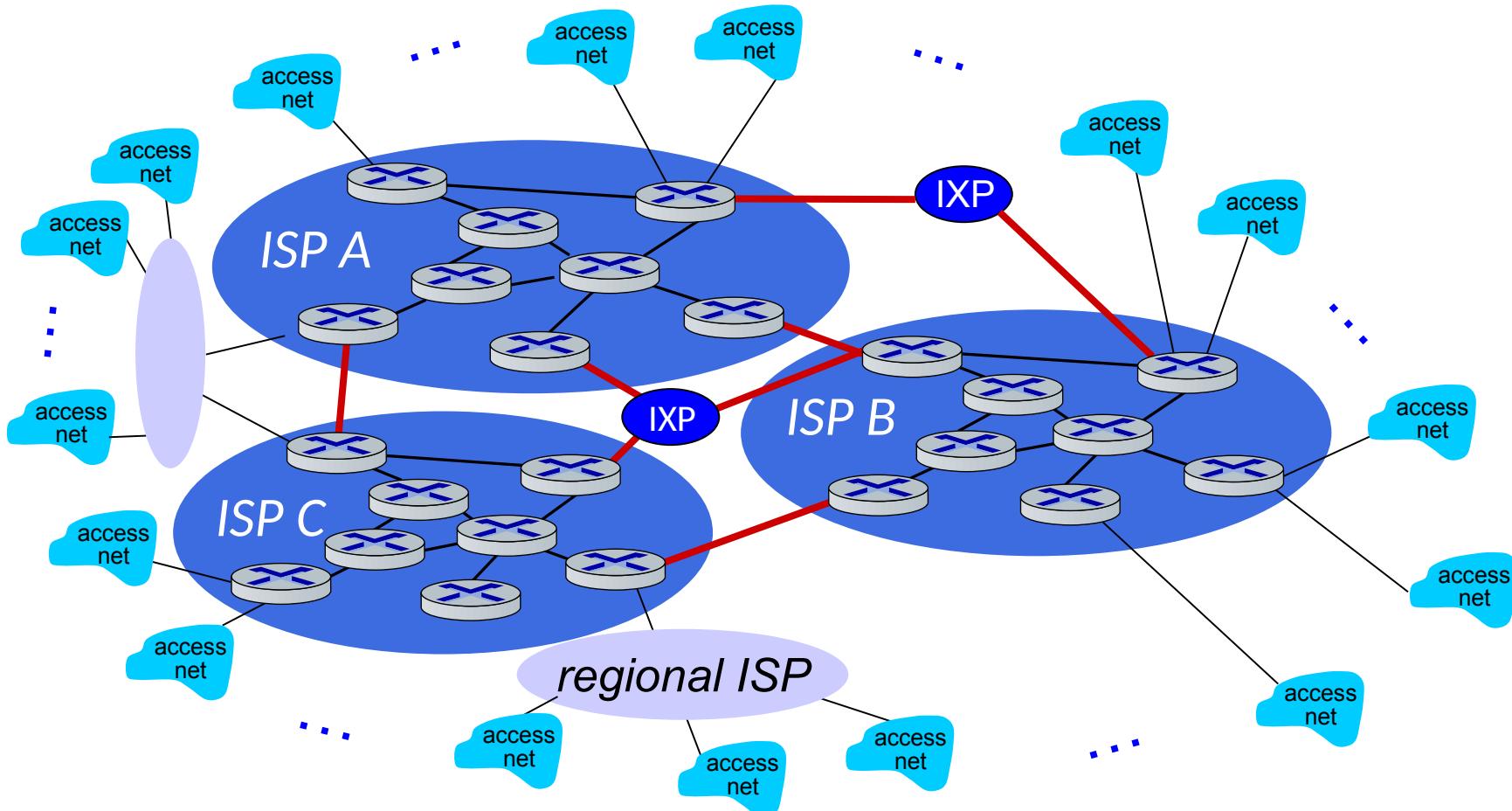
Internet structure: a “network of networks”

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



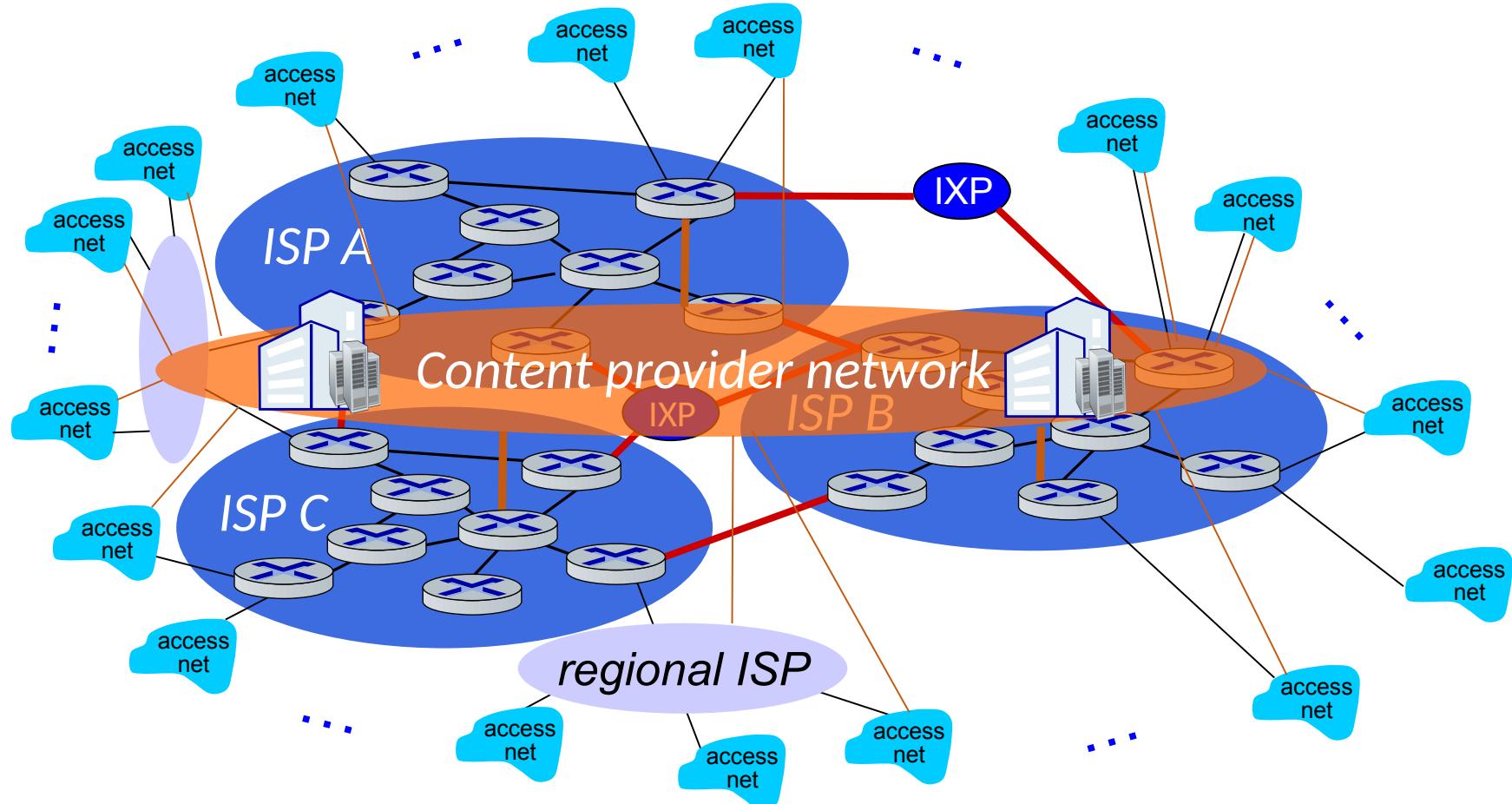
Internet structure: a “network of networks”

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

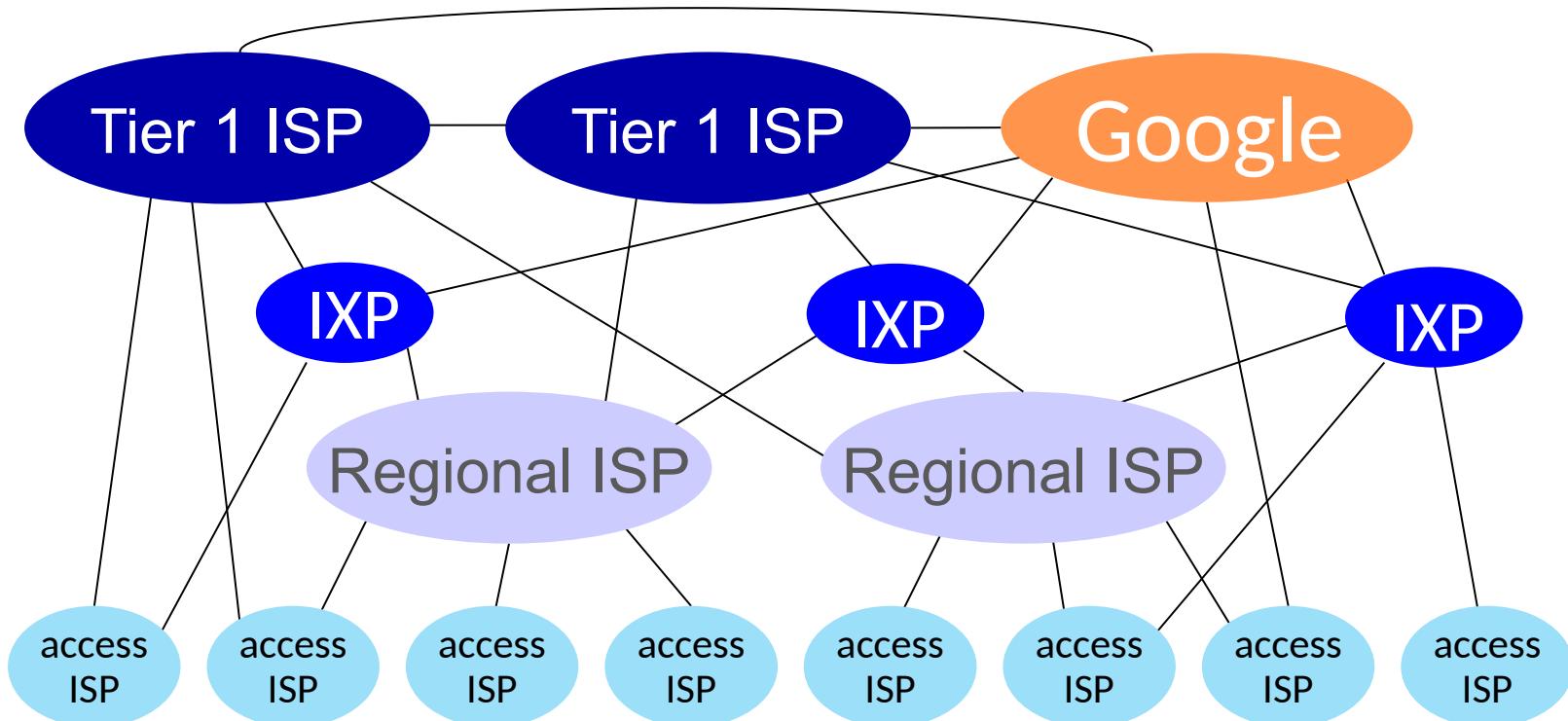


Internet structure: a “network of networks”

... 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

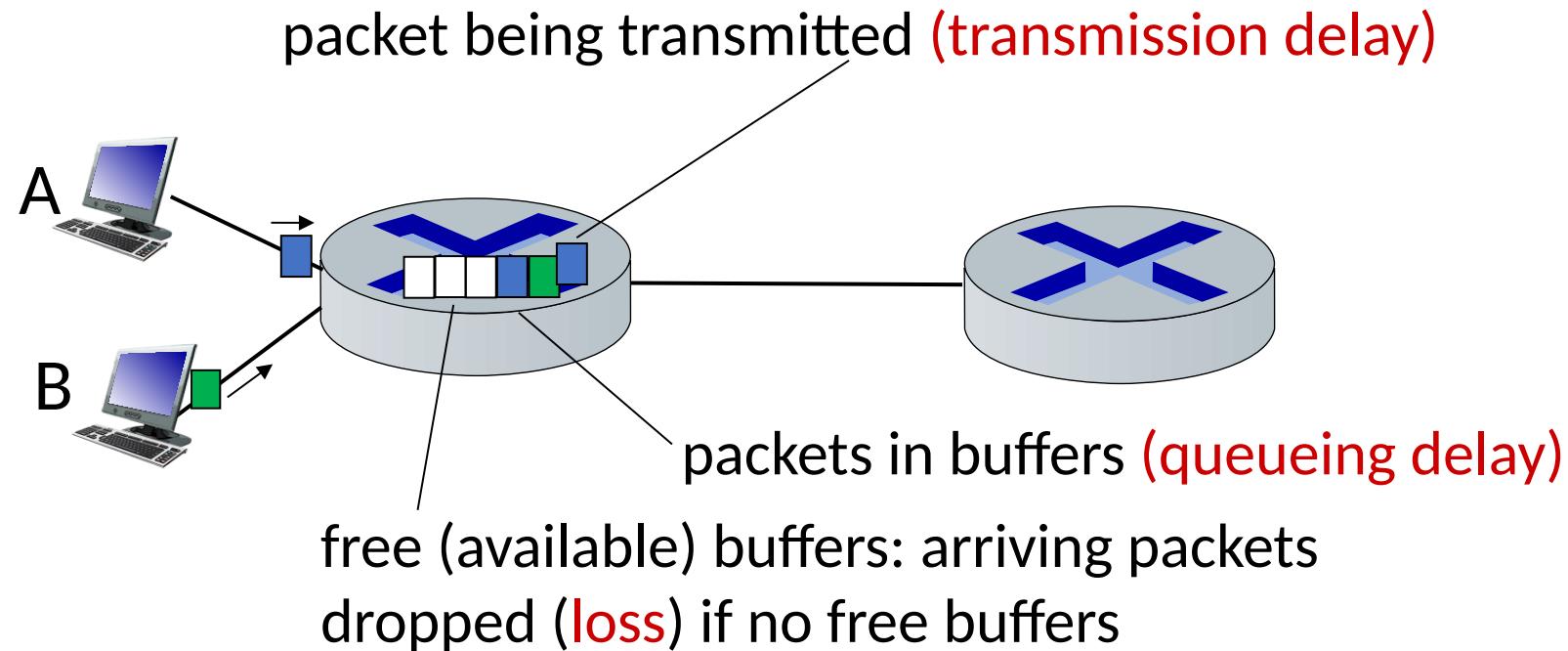
Chapter 1: roadmap

- 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

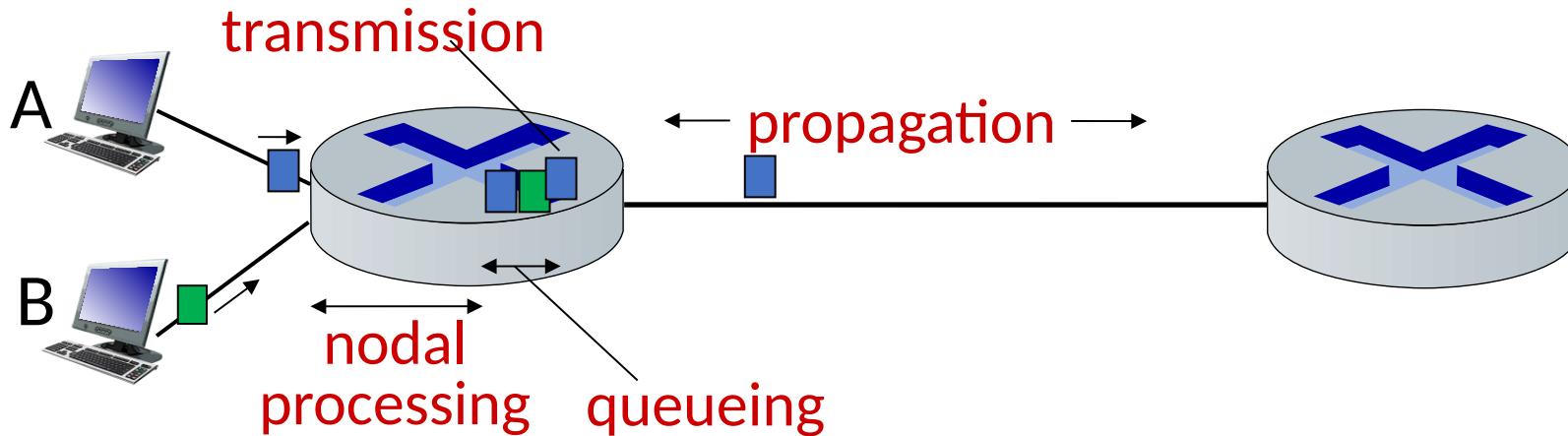


How do packet delay and loss occur?

- 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



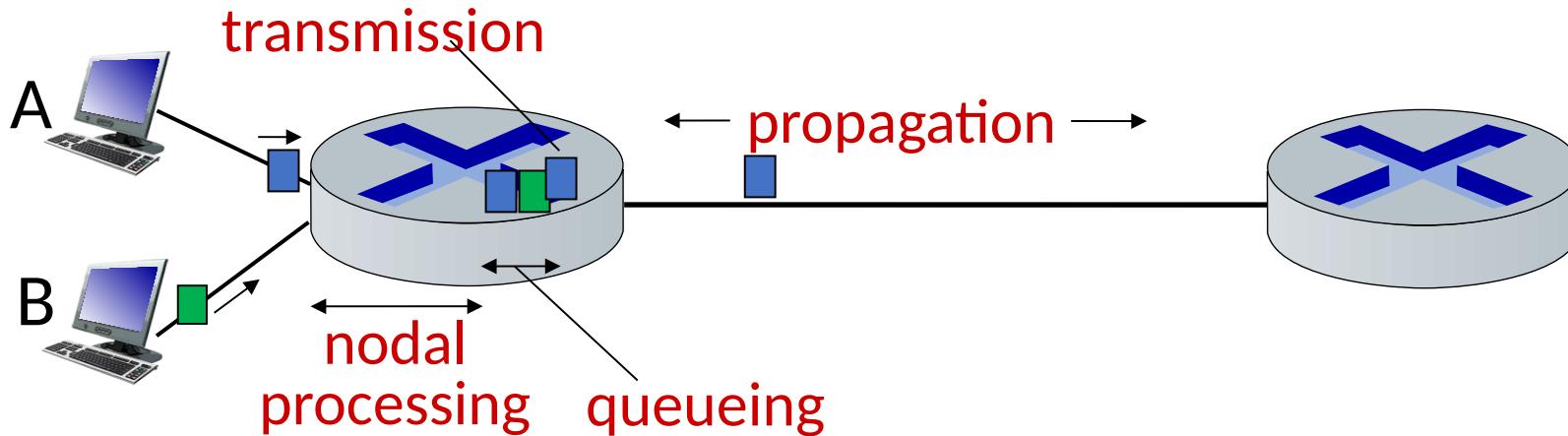
d_{proc} : nodal processing

- 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)

$$d_{\text{prop}} = d/s$$

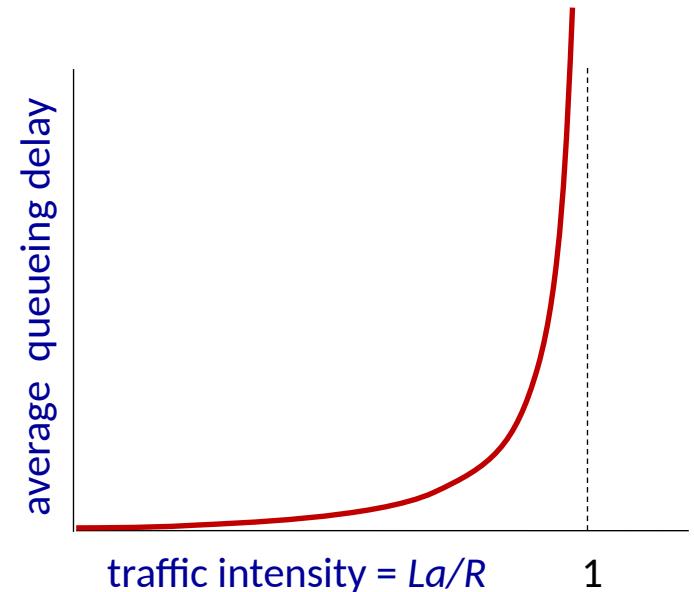
d_{trans} and d_{prop}
very different

Packet queueing delay (revisited)

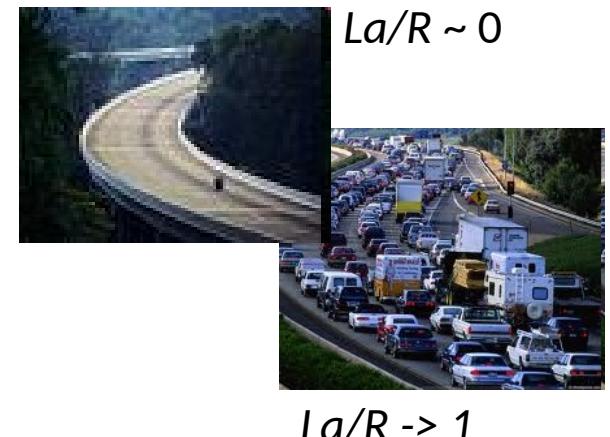
- a : average packet arrival rate
- L : packet length (bits)
- R : link bandwidth (bit transmission rate)

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

“traffic
intensity”

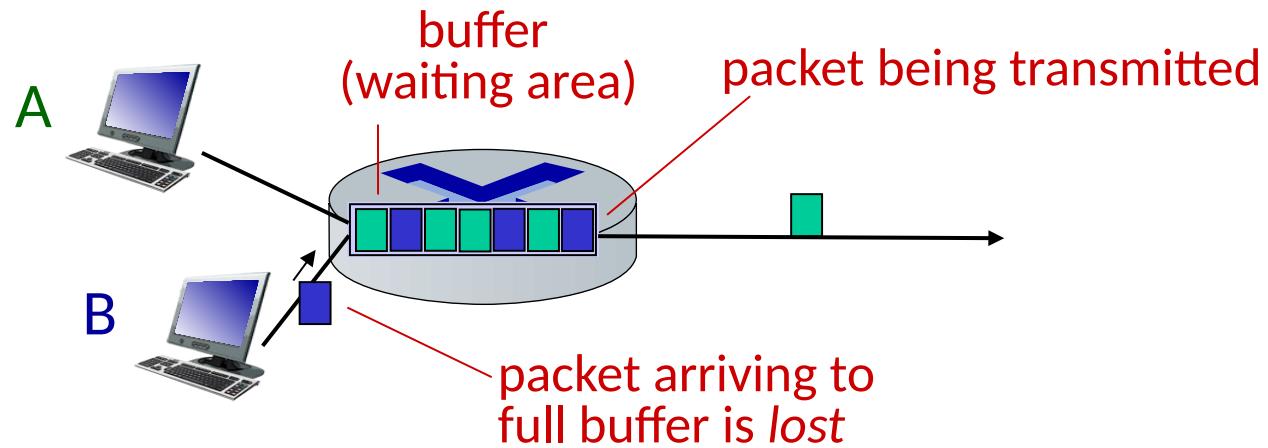


- $La/R \sim 0$: avg. queueing delay small
- $La/R \rightarrow 1$: avg. queueing delay large
- $La/R > 1$: more “work” arriving is more than can be serviced - average delay infinite!



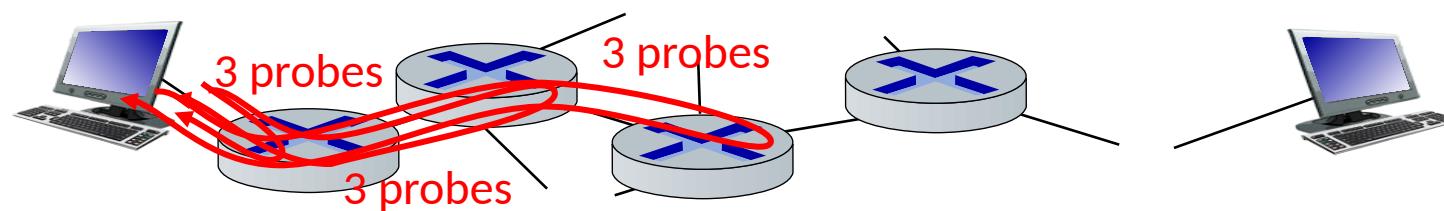
Packet loss

- 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



“Real” Internet delays and routes

- what do “real” Internet delay & loss look like?
- **traceroute** (**tracert** in Windows) 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 (TTL) field value of i)
 - router i will return packets to sender
 - sender measures time interval between transmission and reply



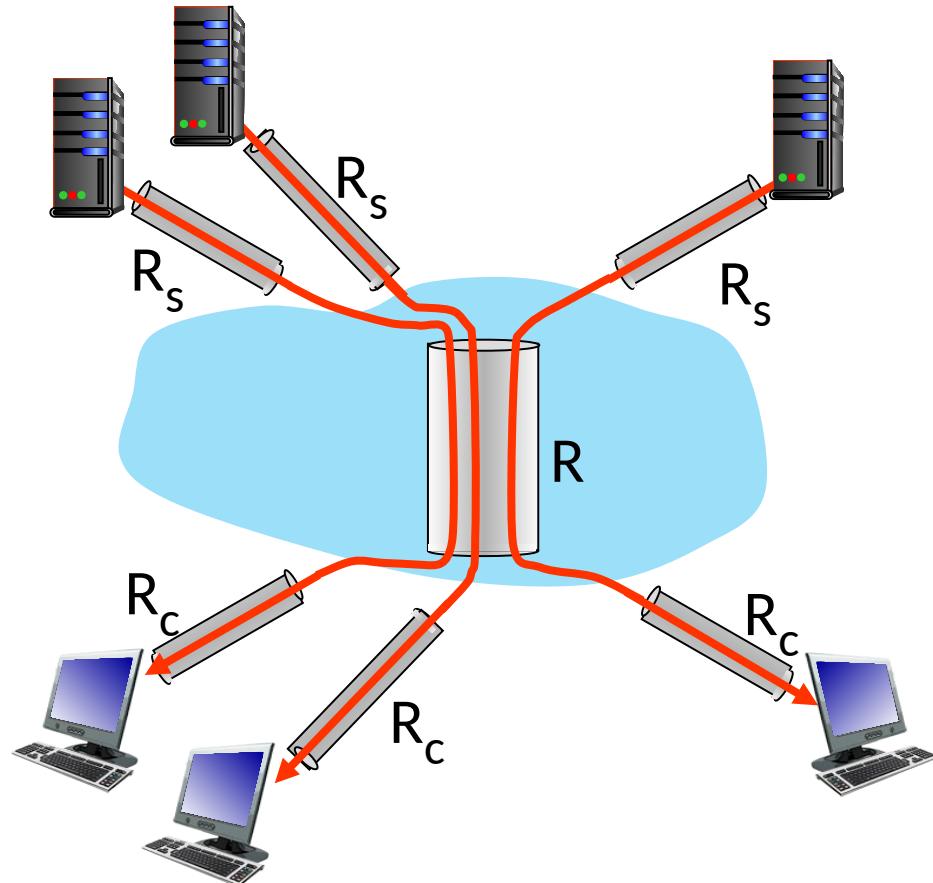
Real Internet delays and routes

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

		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 delay measurements to border1-rt-fa5-1-0.gw.umass.edu
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	trans-oceanic link
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	looks like delays decrease! Why?
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	

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

Throughput: network scenario



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

- per-connection end-end throughput:
 $\min(R_c, R_s, R/3)$
- 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



Network security

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

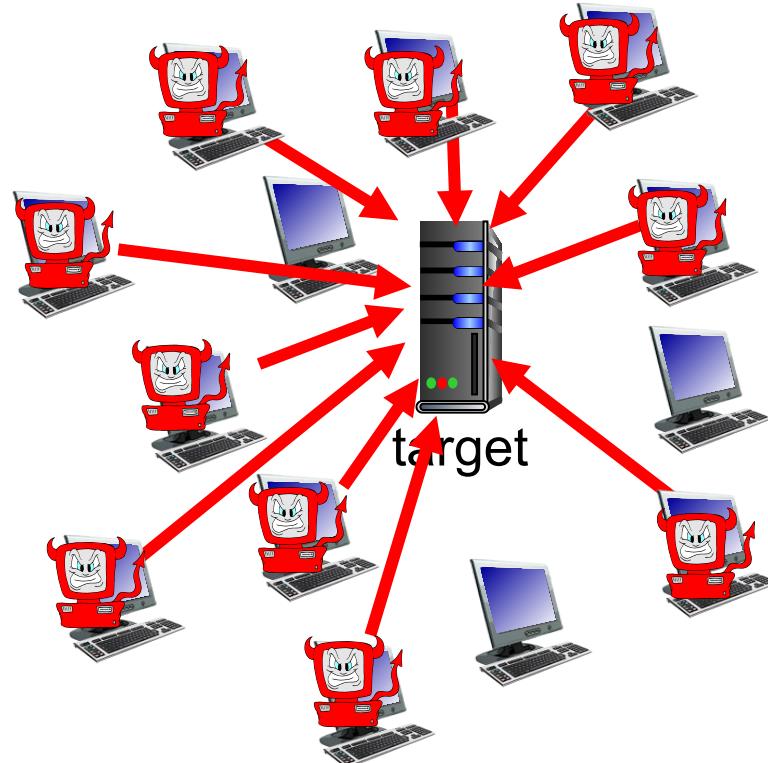
Bad guys: put malware into hosts via Internet

- **malware** can get in host from:
 - *virus*: self-replicating infection targeting a software or system vulnerability
 - *worm*: self-replicating infection by passively receiving object that gets itself executed
 - *trojan*: tricks a user into running malicious software by receiving/executing object (e.g., e-mail attachment)
- **keyloggers** can record keystrokes, **spyware** logs web sites visited, upload info to collection site
- infected host can be enrolled in **botnet**, used for spam, DDoS attacks and others...

Bad guys: denial of service

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

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



Lines of defense:

- **authentication:** proving you are who you say you are
 - cellular networks provides hardware identity via SIM card; no such hardware assist in traditional Internet
- **confidentiality:** via encryption
- **integrity checks:** digital signatures prevent/detect tampering
- **access restrictions:** password-protected VPNs
- **firewalls:** specialized “middleboxes” in access and core networks:
 - off-by-default: filter incoming packets to restrict senders, receivers, applications
 - detecting/reacting to DOS attacks

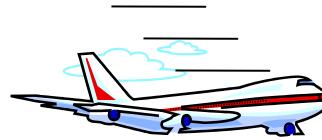
... lots more on security (throughout, Chapter 8)

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



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
- relying on services provided by layer below

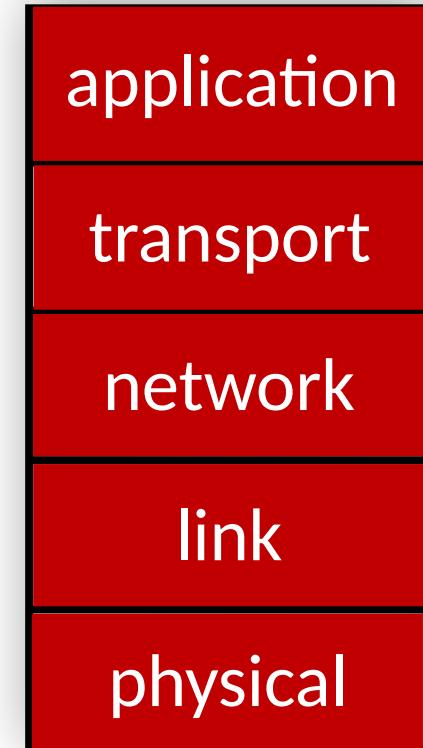
Why layering?

Approach to designing/discussing complex systems:

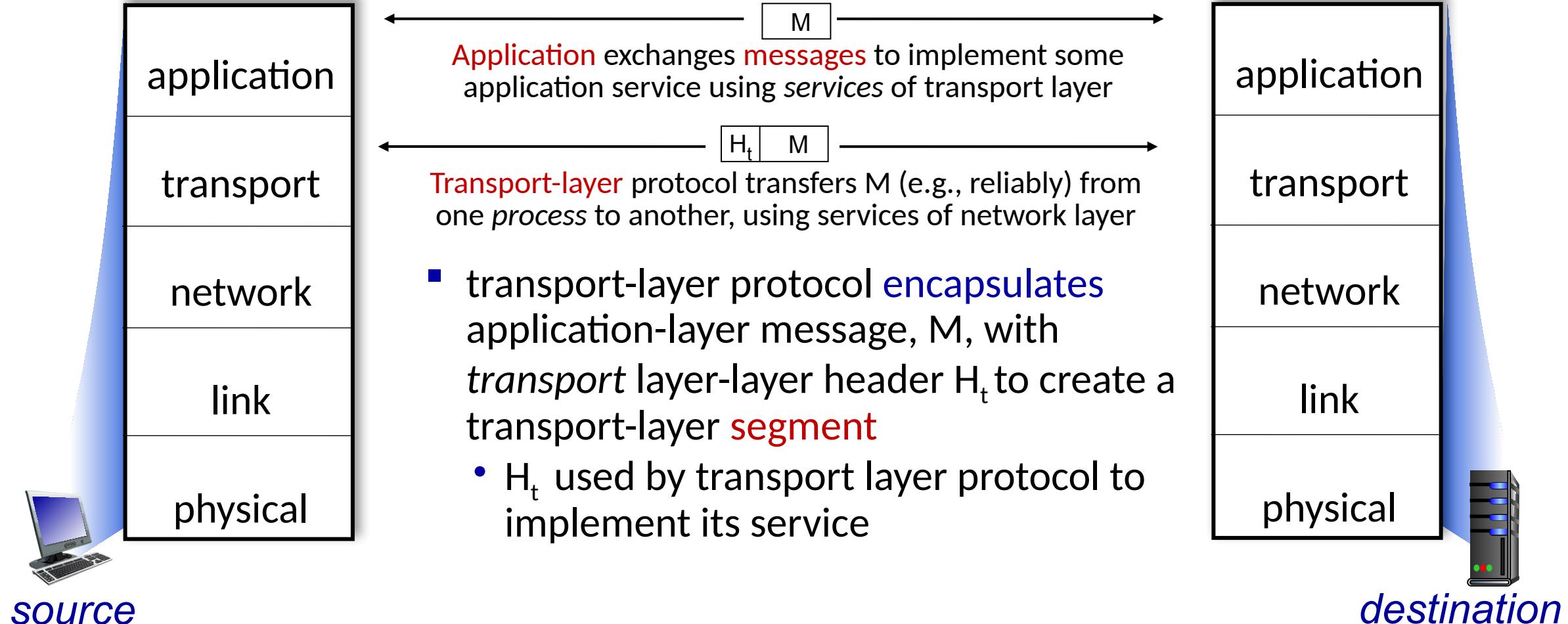
- explicit structure allows identification, relationship of system's pieces
 - layered *reference model* for discussion
- modularization eases maintenance, updating of system
 - change in layer's service *implementation*: transparent to rest of system
 - e.g., change in gate procedure doesn't affect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

Layered Internet protocol stack

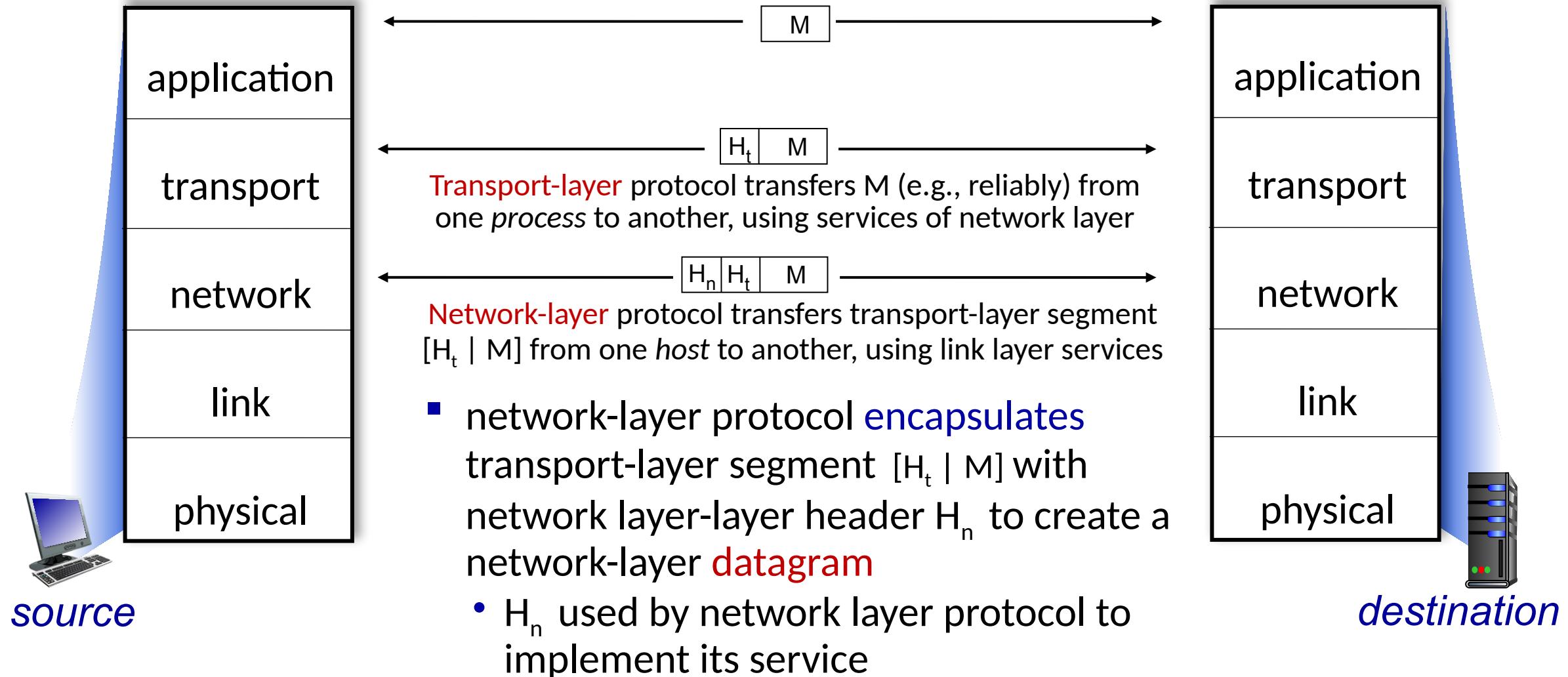
- *application*: supporting network applications
 - HTTP, IMAP, SMTP, DNS
- *transport*: process-process data transfer
 - TCP, UDP
- *network*: routing of datagrams from source to destination
 - IP, routing protocols
- *link*: data transfer between neighboring network elements
 - Ethernet, 802.11 (WiFi),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



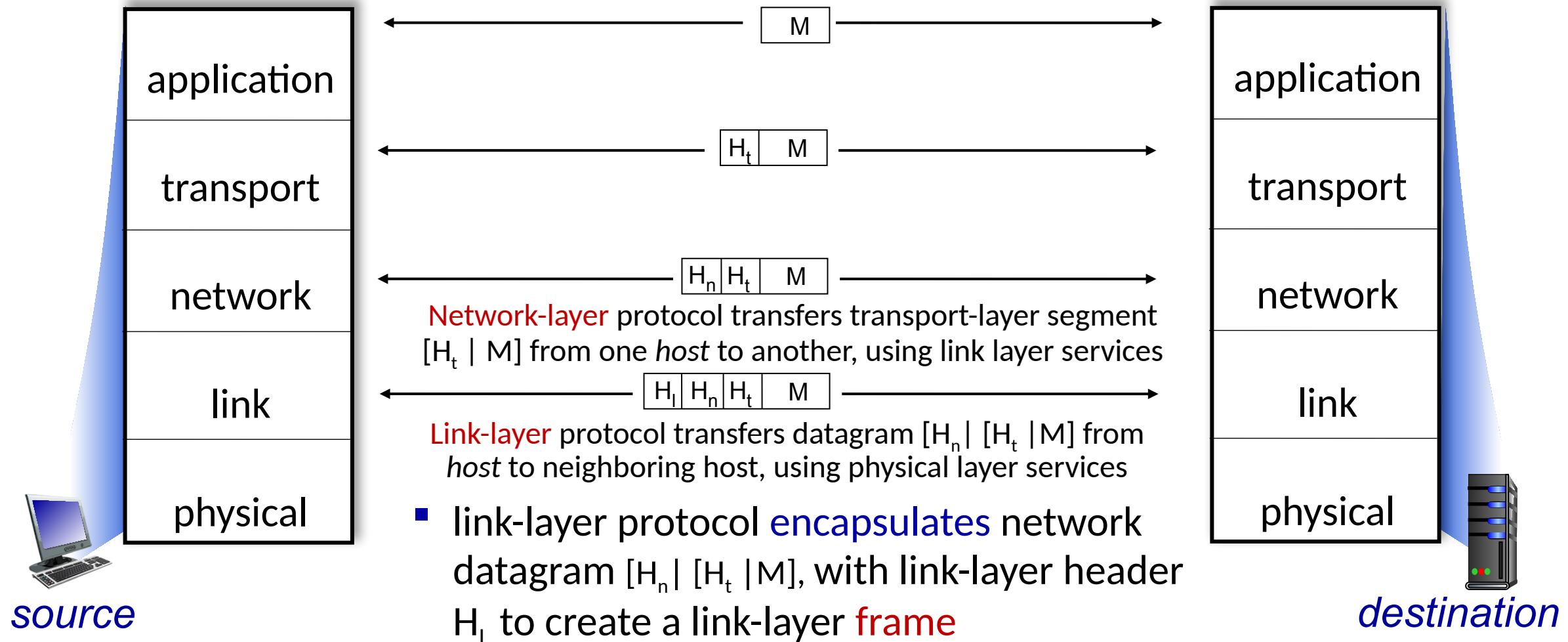
Services, Layering and Encapsulation



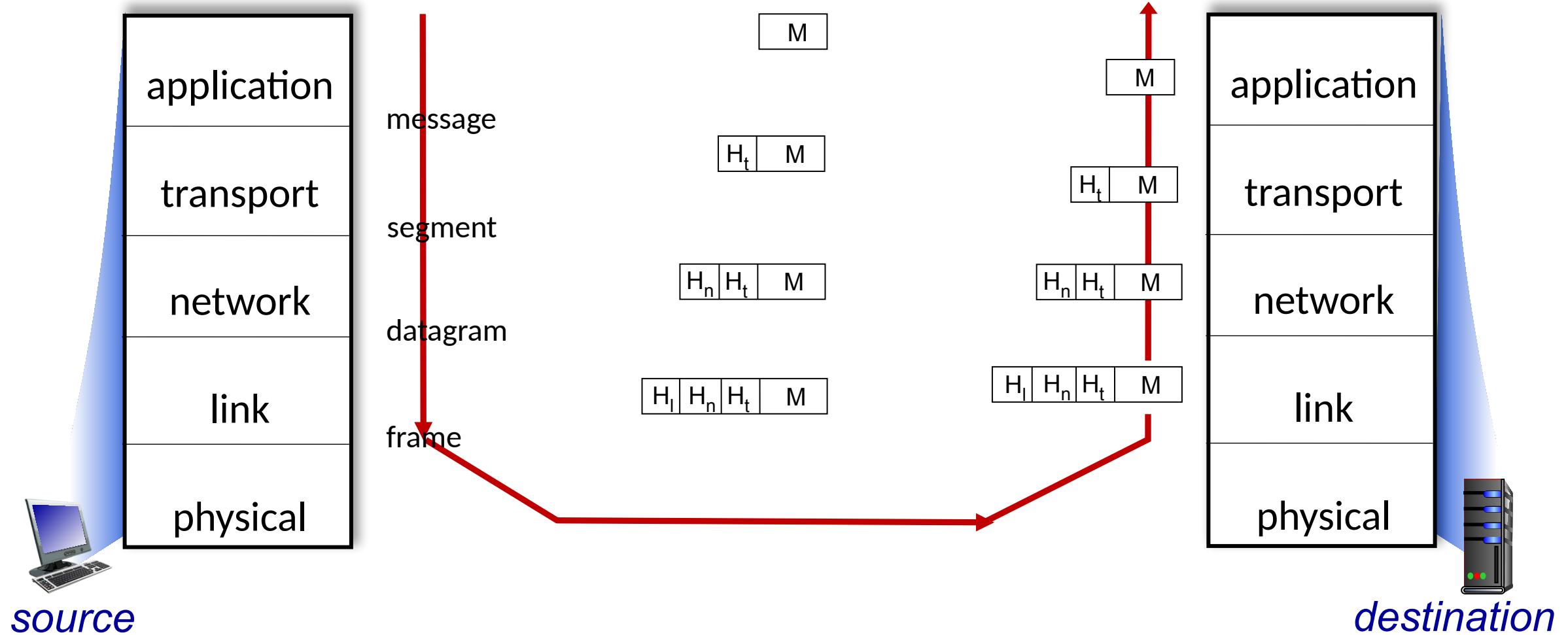
Services, Layering and Encapsulation



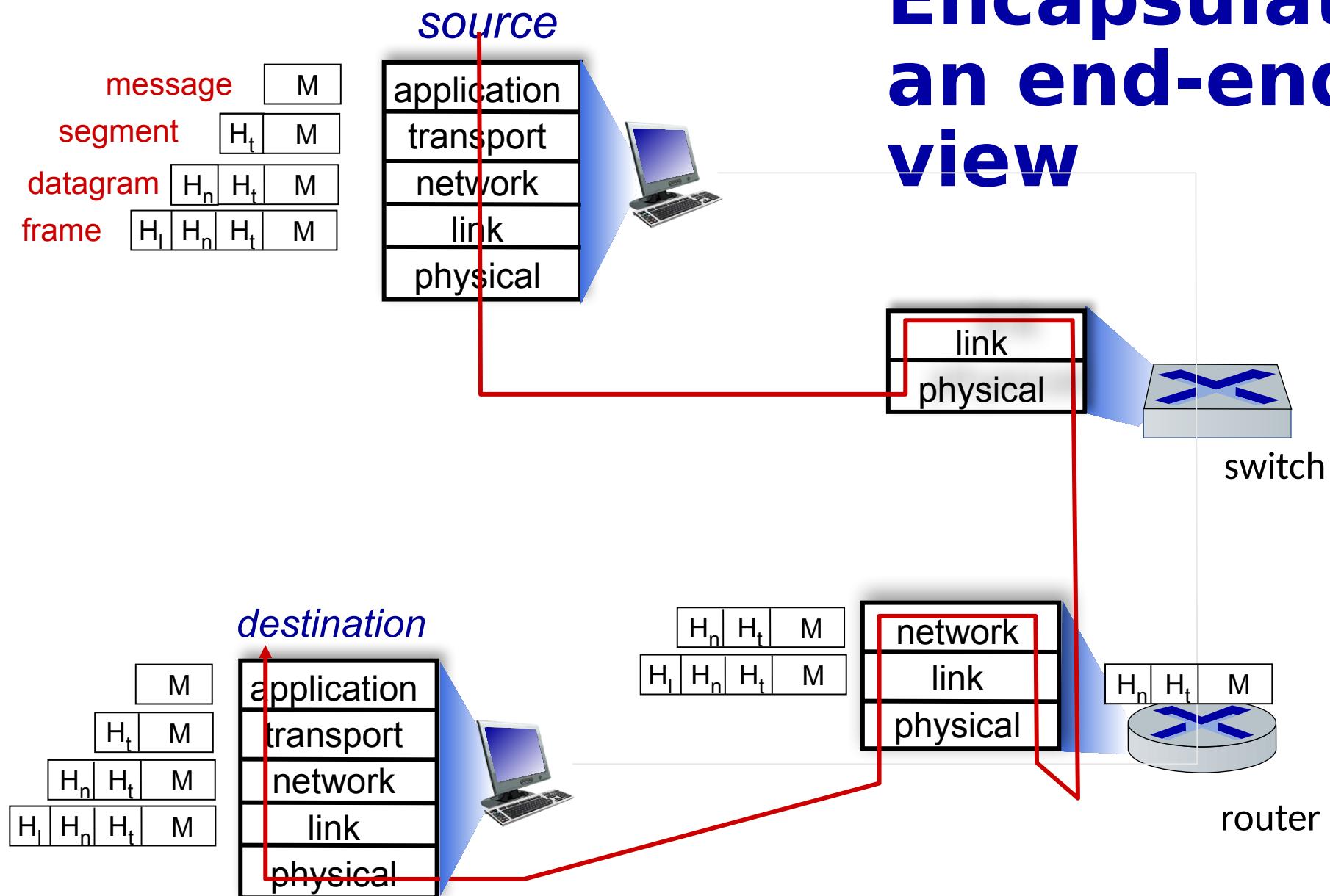
Services, Layering and Encapsulation



Services, Layering and Encapsulation



Encapsulation: an end-end view



Chapter 1: roadmap

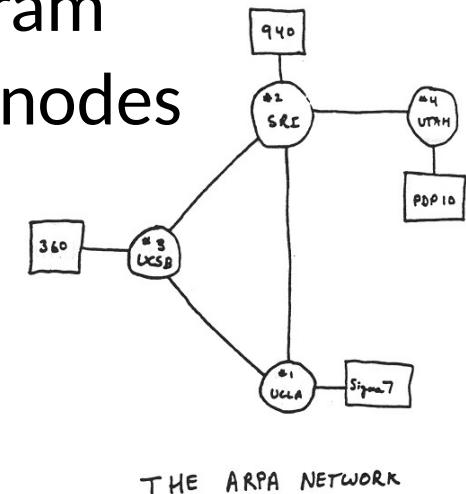
- What *is* the Internet?
- What *is* a protocol?
- Network edge: hosts, access network, physical media
- Network core: packet/circuit switching, internet structure
- Performance: loss, delay, throughput
- Security
- Protocol layers, service models
- History



Internet history

1961-1972: Early packet-switching principles

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



Internet history

1972-1980: Internetworking, new and proprietary networks

- 1970: ALOHAnet satellite network in Hawaii
- 1974: Cerf and Kahn - architecture for interconnecting networks
- 1976: Ethernet at Xerox PARC
- late70's: proprietary architectures: DECnet, SNA, XNA
- 1979: ARPAnet has 200 nodes

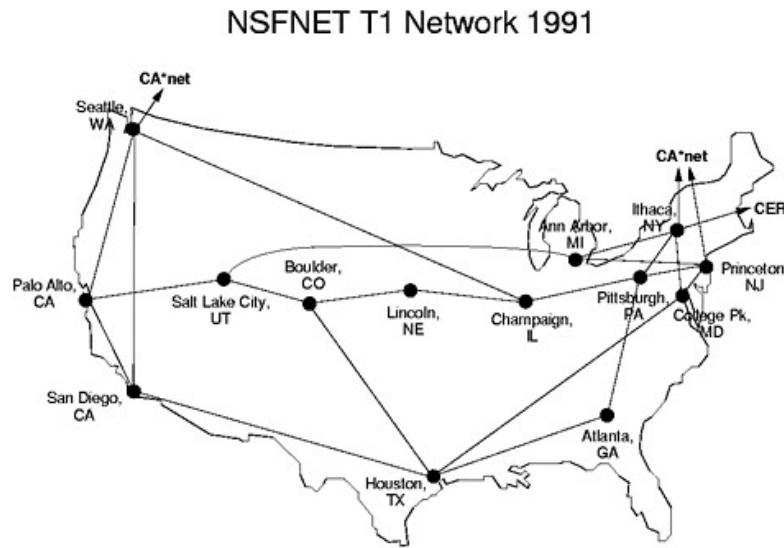
Cerf and Kahn's internetworking principles:

- minimalism, autonomy - no internal changes required to interconnect networks
 - best-effort service model
 - stateless routing
 - decentralized control
- define today's Internet architecture

Internet history

1980-1990: new protocols, a proliferation of networks

- Jan 1, 1983: birth of the Internet as ARPANET switched to TCP/IP protocols
- 1983: deployment of TCP/IP
- 1982: SMTP e-mail protocol defined
- 1983: DNS defined for name-to-IP-address translation
- 1985: FTP protocol defined
- 1988: TCP congestion control
- new national networks: CSnet, BITnet, NSFnet, Minitel
- 100,000 hosts connected to confederation of networks



Internet history

1990, 2000s: commercialization, the Web, new applications

- early 1990s: ARPAnet decommissioned
 - 1991: NSF lifts restrictions on commercial use of NSFnet (decommissioned, 1995)
 - early 1990s: Web
 - hypertext [Bush 1945, Nelson 1960's]
 - HTML, HTTP: Timothy Berners-Lee
 - 1994: Mosaic, later Netscape
 - late 1990s: commercialization of the Web
- late 1990s – 2000s:
- more killer apps: instant messaging, P2P file sharing
 - network security to forefront
 - est. 50 million host, 100 million+ users
 - backbone links running at Gbps

Internet history

2005-present: scale, SDN, mobility, cloud

- aggressive deployment of broadband home access (10-100's Mbps)
- 2008: software-defined networking (SDN)
- increasing ubiquity of high-speed wireless access: 4G/5G, WiFi
- service providers (Google, FB, Microsoft) create their own networks
 - bypass commercial Internet to connect “close” to end user, providing “instantaneous” access to social media, search, video content, ...
- enterprises run their services in “cloud” (e.g., Amazon Web Services, Microsoft Azure)
- rise of smartphones: more mobile than fixed devices on Internet (2017)
- ~18B devices attached to Internet (2017)

Chapter 1: summary

We've covered a “ton” of material!

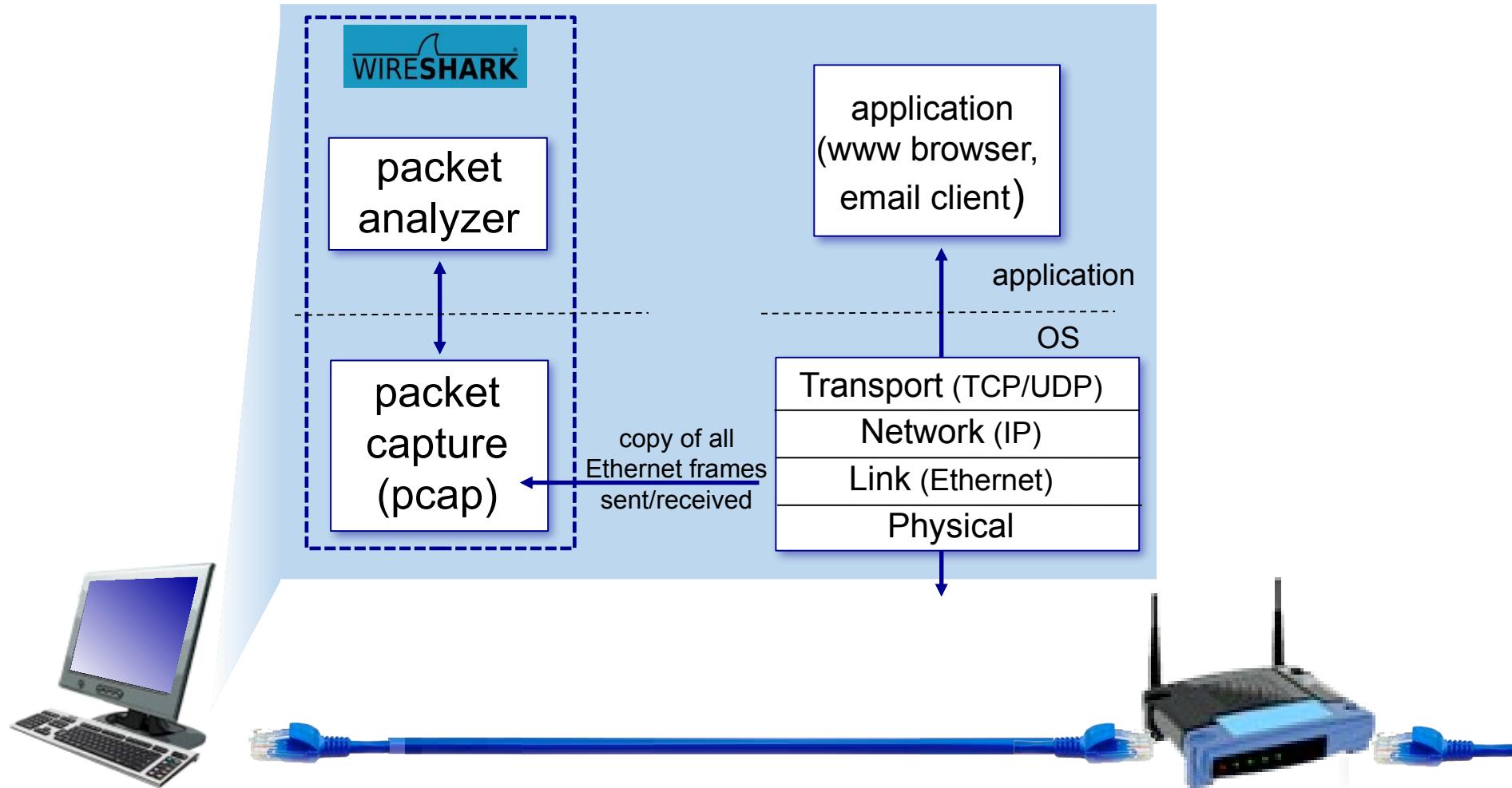
- Internet overview
- what's a protocol?
- network edge, access network, core
 - packet-switching versus circuit-switching
 - Internet structure
- performance: loss, delay, throughput
- layering, service models
- security
- history

You now have:

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

Additional Chapter 1 slides

Wireshark



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

EIGHTH EDITION

James F. Kurose • Keith W. Ross



Computer Networking: A Top-Down Approach

8th edition

Jim Kurose, Keith Ross
Pearson, 2020

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

Application layer: overview

Our goals:

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

Some network apps

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

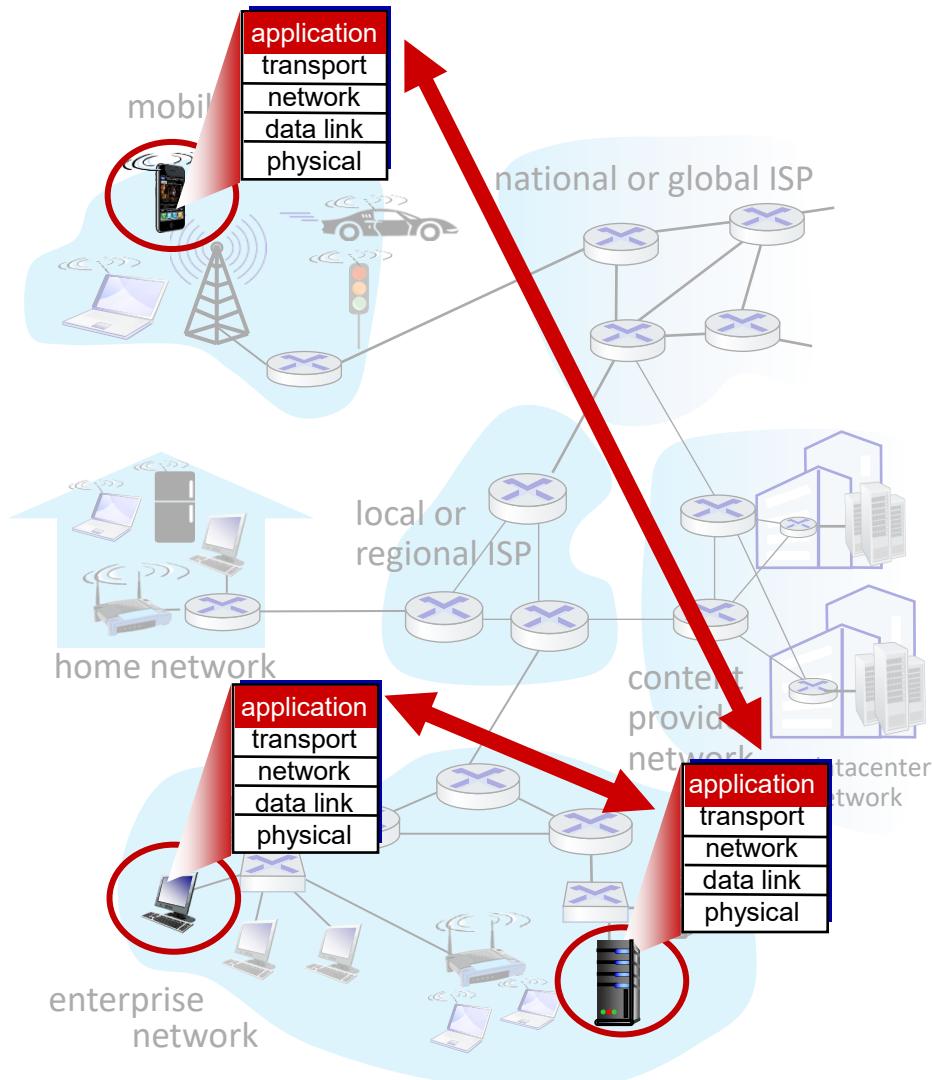
Creating a network app

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



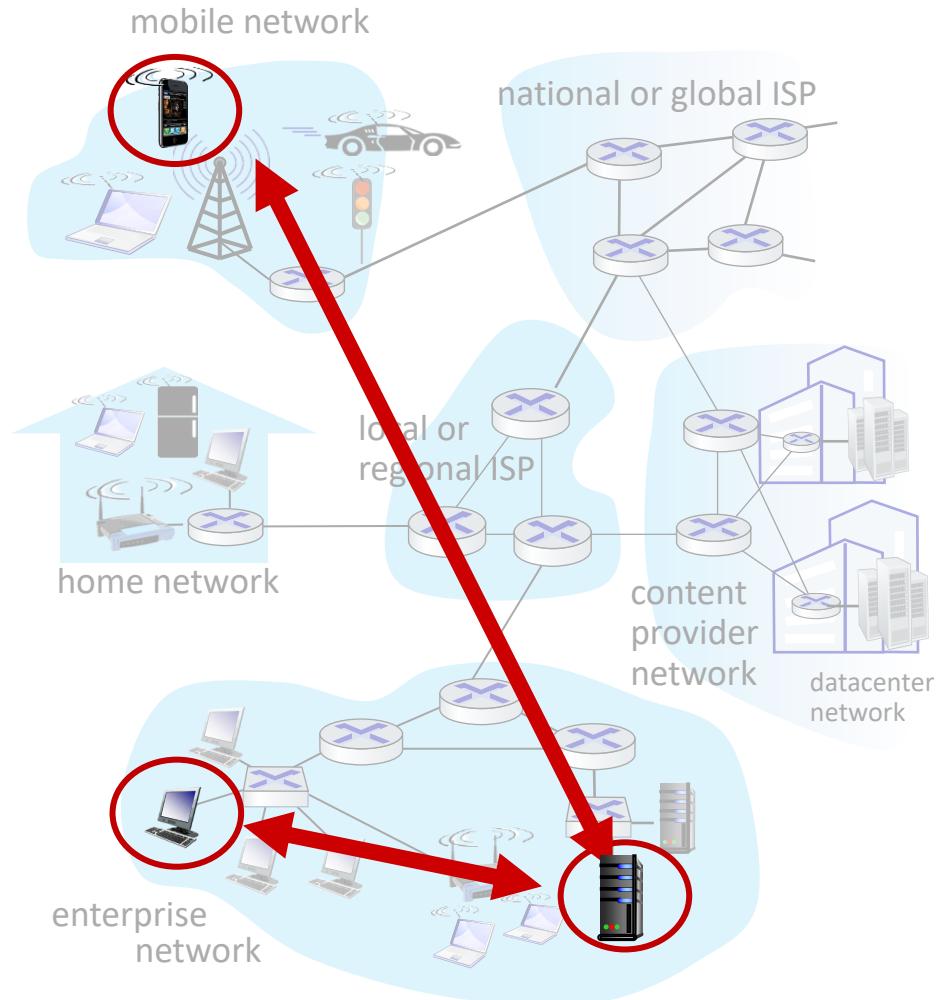
Client-server model

server:

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

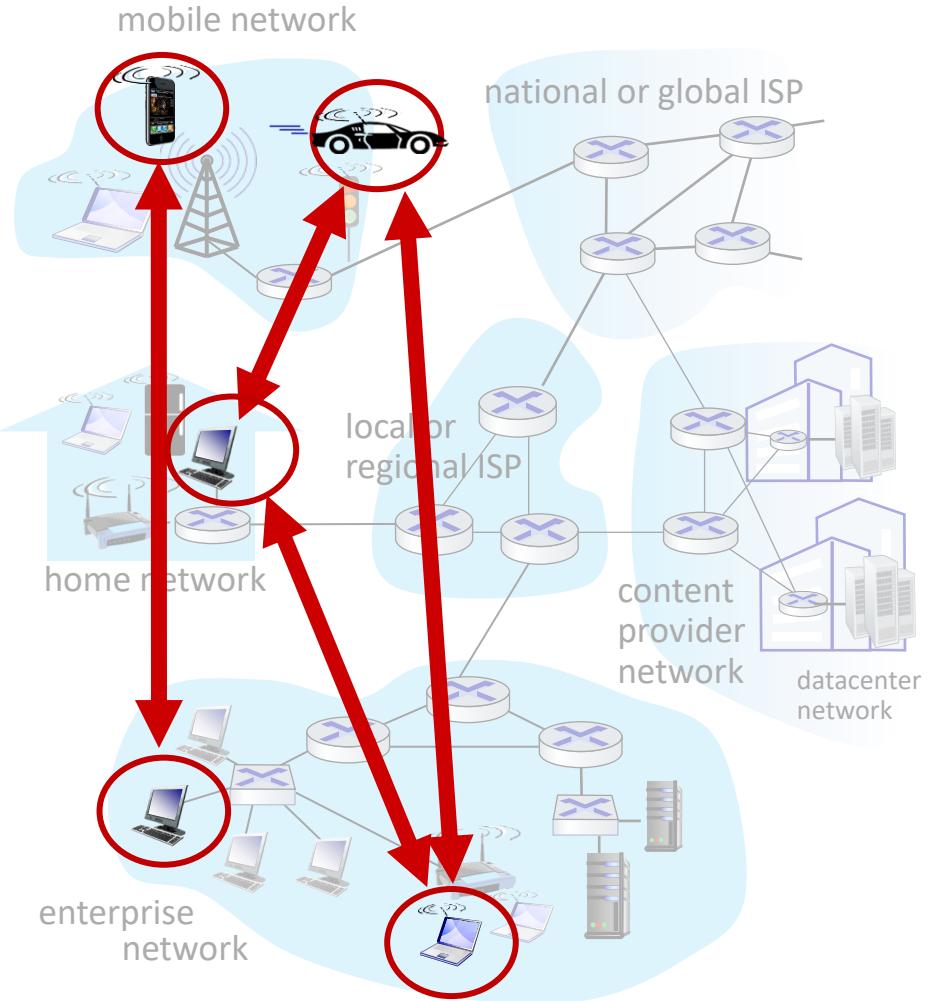
clients:

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



Peer-to-peer (P2P) model

- 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**

Request-response

client process: process that initiates communication using a request

server process:

- process that waits to be contacted
- sends a response to the client

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

Addressing processes

- to receive messages, a process must have *identifier*
- host device has unique 32-bit IP address
- Q: is the IP address of the host that the process runs on enough for identifying the process?
- A: no, *many* processes can be running on same host
- *identifier* includes both **IP address** and **port numbers** associated with process on host.
- example port numbers:
 - HTTP server: 80
 - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
 - **IP address:** 128.119.245.12
 - **port number:** 80
- more shortly...

What transport service does an app need?

reliability

- 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 minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput they get

timing, delay

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

security

- encryption, data integrity, ...

Internet transport protocols services

TCP (Transmission Control Protocol):

- ***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/delay, minimum throughput guarantee

UDP (User Datagram Protocol):

- ***fast packet transmission***
- ***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
	HTTP/3 [RFC 9114]	UDP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	UDP or TCP

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

2.2 Web and HTTP

- 2.2.1 Overview of HTTP
- 2.2.2 Non-persistent and persistent connections
- 2.2.3 HTTP message format
- 2.2.4 Cookies
- ~~2.2.5 Web caching~~
- 2.2.6 HTTP/2

2.2 Web and HTTP

HTTP: transfer protocol
HTML: markup language

A quick HTML review...

- web page consists of *base HTML-file* which *references several objects, each* addressable by a *URL*
 - objects can be HTML files, JPEG images, Java applets, audio files, etc.
 - Each object has an address and can stored on different Web servers

www.someschool.edu/someDept/pic.gif

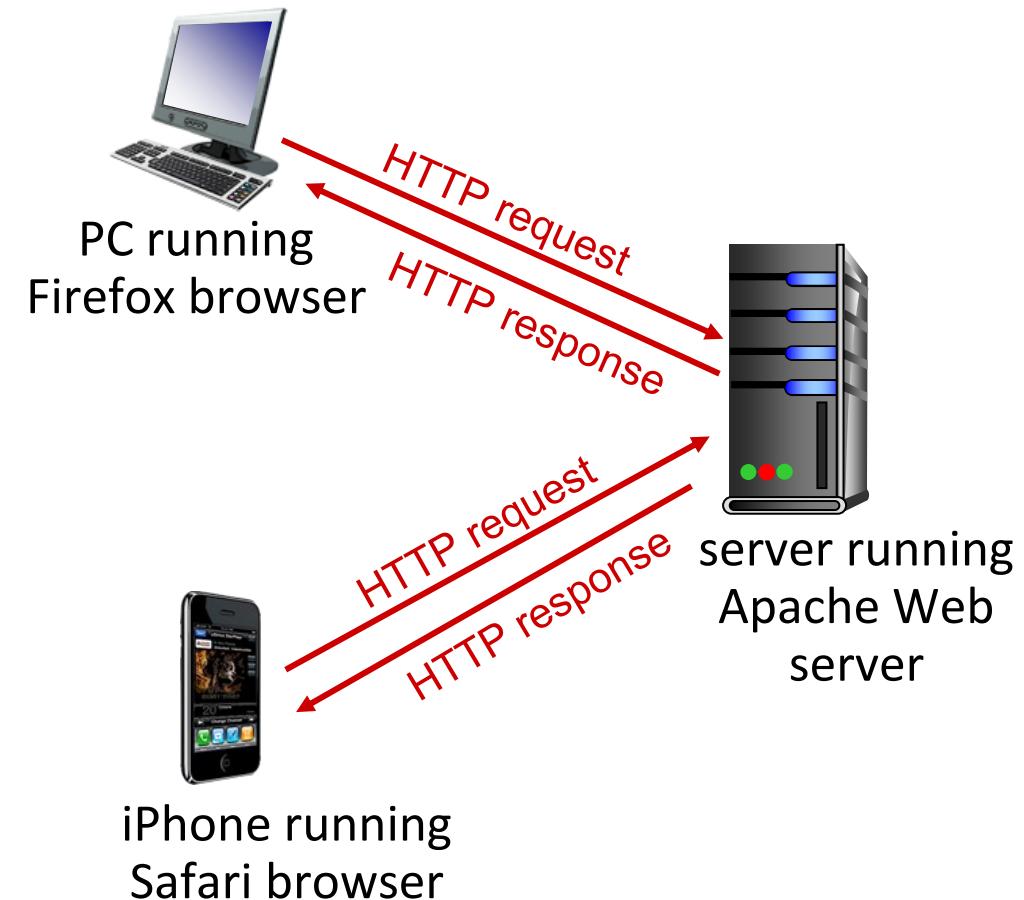
host name

path name

2.2.1 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

2.2.2 HTTP connections: two types

Non-persistent HTTP/1.0 [1996]

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/1.1 [1997]

- 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)



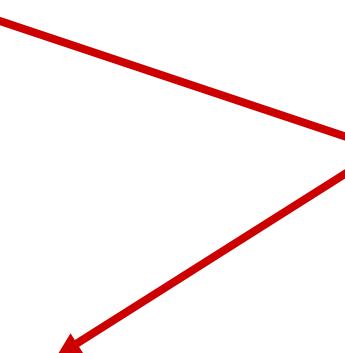
1a. HTTP client initiates TCP connection to HTTP server (process) at `www.someSchool.edu` on port 80



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.

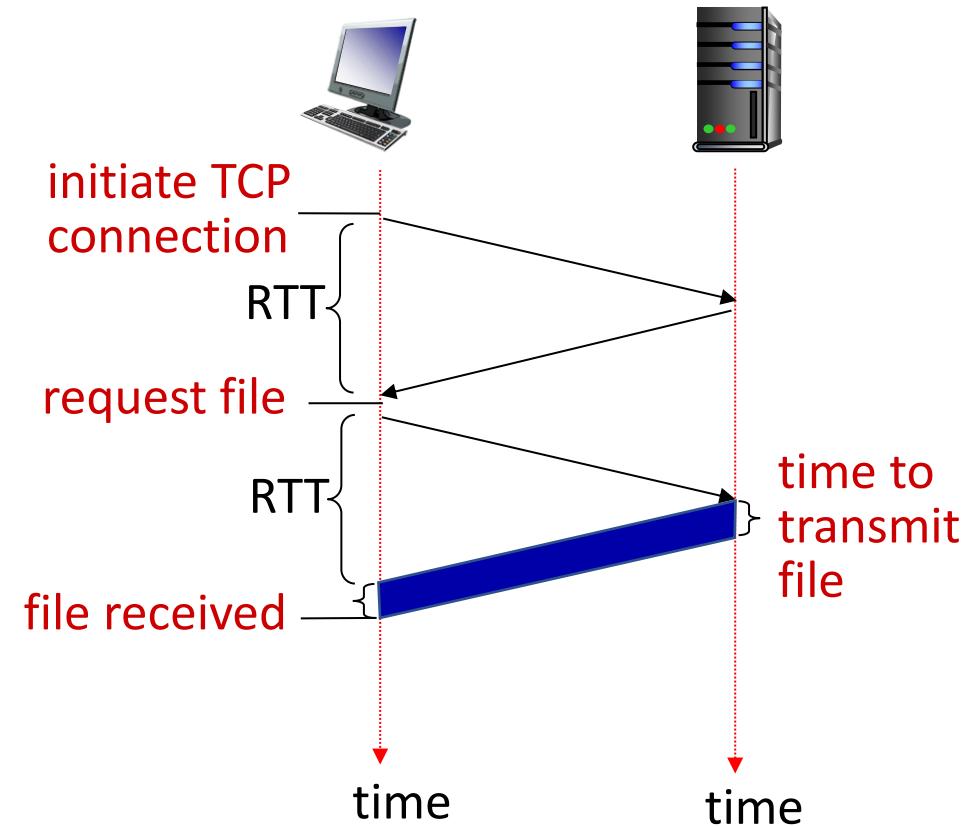
time

Non-persistent HTTP: response time

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 HTTP response to return
- object/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 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)

HTTP messages

- Two types of HTTP messages:
 - **requests** sent by the client to trigger an action on the server
 - **responses** are the answer returned by the server in response to a request
- HTTP messages are normally machine generated
- HTTP messages are text-based (ASCII), and easy to read and understand

2.2.3 HTTP/1.1 request message example

Start line → GET /index.html HTTP/1.1

Header lines (optional) {

- Host: www-net.cs.umass.edu
- User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:80.0) Gecko/20100101 Firefox/80.0
- Accept: text/html,application/xhtml+xml
- Accept-Language: en-us,en;q=0.5
- Accept-Encoding: gzip,deflate
- Connection: keep-alive

Empty line → \r\n

carriage return character (0x0D)

newline character (0x0A)

HTTP request methods

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
- completely replaces file that exists at specified URL with content in entity body of PUT HTTP request message

HTTP/1.1 response message example

Start line → HTTP/1.1 200 OK

Header lines (optional) {

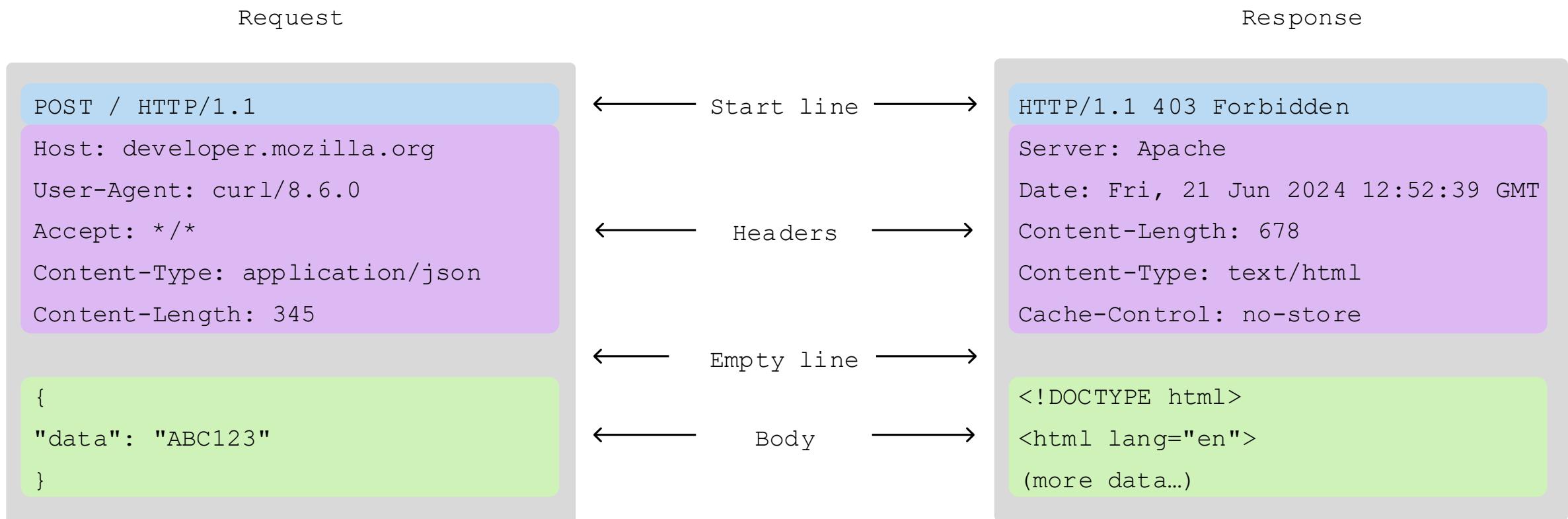
- Date: Tue, 08 Sep 2020 00:53:20 GMT
- Server: Apache/2.4.6 (CentOS)
- OpenSSL/1.0.2k-fips PHP/7.4.9
- mod_perl/2.0.11 Perl/v5.16.3
- Last-Modified: Tue, 01 Mar 2016 18:57:50 GMT
- ETag: "a5b-52d015789ee9e"
- Accept-Ranges: bytes
- Content-Length: 2651
- Content-Type: text/html; charset=UTF-8

Empty line → \r\n

Body (requested object) {

- data data data data data ...

Comparison of HTTP request and response



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

Web-pages and HTML

- A very simple webpage that shows how its HTML code and the layout corresponds

URL:

http://pracnet.net/simple.html

host name

path name

Default HTTP server port 80. No encryption

Trying out HTTP using netcat

1. start command prompt, and type

```
C:\temp>ncat pracnet.net 80
```

- opens TCP connection to pracnet.net at port 80 (default HTTP server port)
- anything typed in will be sent to port 80

2. type in a GET HTTP request:

```
GET /simple.html HTTP/1.1
```

```
Host: pracnet.net
```

- hit return twice

3. look at response message sent by HTTP server!

```
C:\temp>ncat pracnet.net 80
GET /simple.html HTTP/1.1
Host: pracnet.net

HTTP/1.1 200 OK
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
Content-Type: text/html
Last-Modified: Tue, 12 Sep 2023 18:12:09 GMT
Accept-Ranges: bytes
Content-Length: 408
Date: Fri, 17 Jan 2025 23:47:48 GMT
Server: LiteSpeed
```

```
<HTML>
```

```
<HEAD>
```

```
    <TITLE>This is a Simple HTML Page</TITLE>
    <link rel="icon" href="favicon.png" />
```

```
</HEAD>
```

```
<BODY>
```

```
    <H1>This is a Title</H1>
    <hr>
```

```
    <H2>This is a Subtitle</H2>
    <p>This is some text.</p>
```

```
<HTML>

<HEAD>
    <TITLE>This is a Simple HTML Page</TITLE>
    <link rel="icon" href="favicon.png" />
</HEAD>

<BODY>

    <H1>This is a Title</H1>
    <hr>

    <H2>This is a Subtitle</H2>
    <p>This is some text.</p>
    <p>This is some <strong>bold</strong> text.</p>
    <p>This is some <em>italic</em> text.</p>
    <p>This is an image: <BR />
        
    </p>
    <hr>

</BODY>

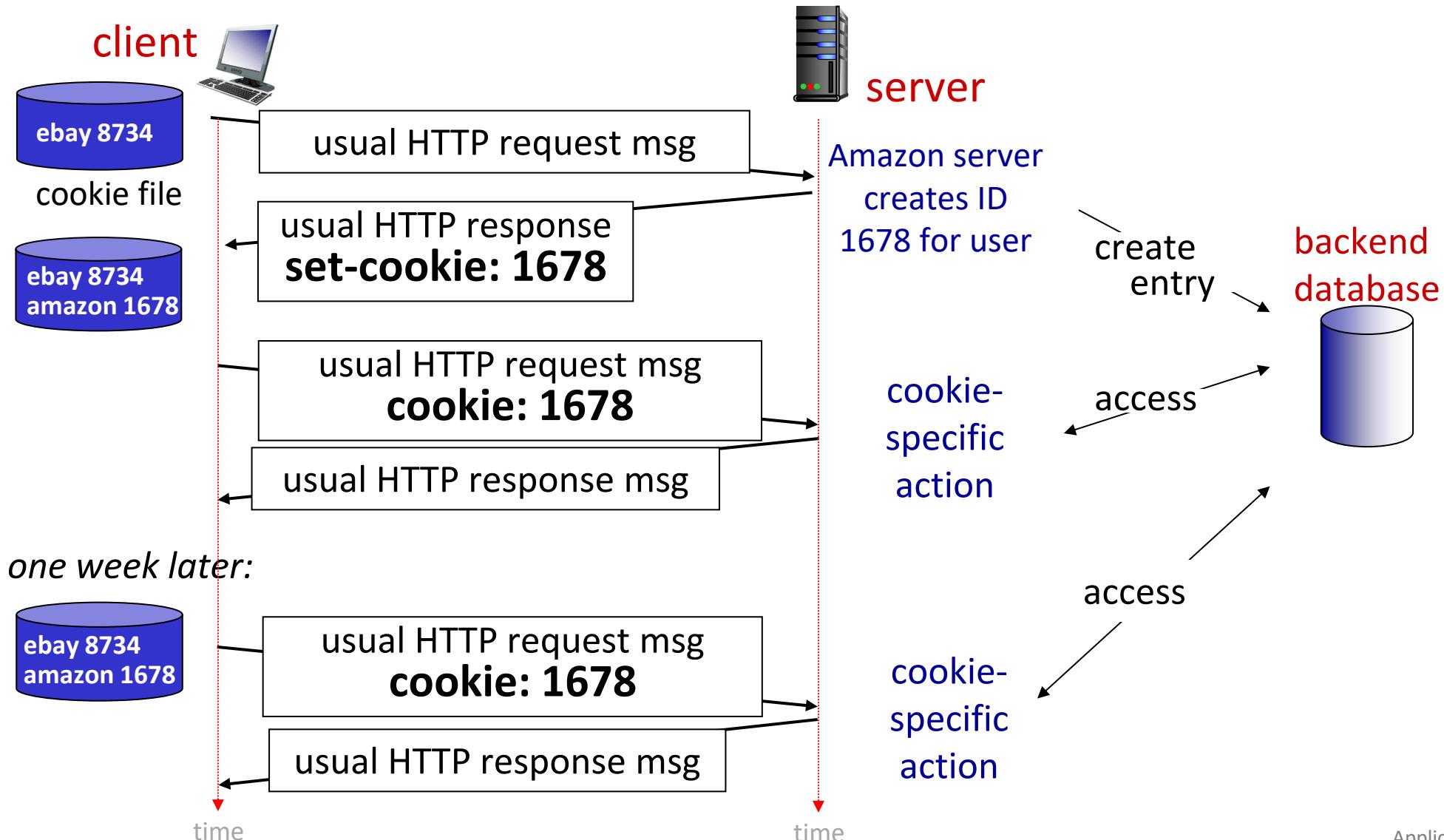
</HTML>
```

2.2.4 Maintaining user/server state: cookies

Web sites and browsers use *cookies* to maintain state

- Cookies are unique long-term identifiers
 - allows to identify web browsers
 - keep track of user activities
-
1. A cookie is selected randomly first time a user visits a webpage
 - The server includes the cookie in the response
 2. The cookie is stored locally by the browser and in the Web site's database
 3. subsequent HTTP requests from the same browser to this site will contain cookie ID value, allowing site to “identify” the browser

Maintaining user/server state: cookies



Cookies: tracking a user's browsing behavior

Cookies can be used to:

- track user behavior on a given website (**first party cookies**)
- track user behavior across multiple websites (**third party cookies**) without user ever choosing to visit tracker site (!)
- tracking may be *invisible* to user:
 - rather than displayed ad triggering HTTP GET to tracker, could be an invisible link

third party tracking via cookies:

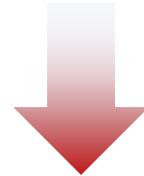
- disabled by default in Firefox, Safari browsers
- to be disabled in Chrome browser in 2023

GDPR (EU General Data Protection Regulation) and cookies

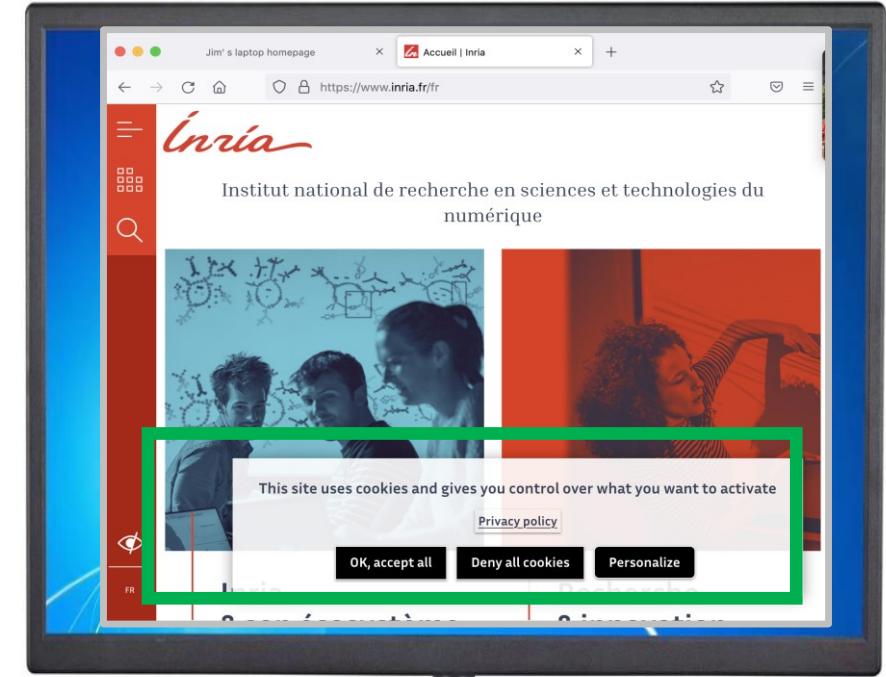
“Natural persons may be associated with online identifiers [...] such as internet protocol addresses, cookie identifiers or other identifiers [...].

This may leave traces which, in particular when combined with unique identifiers and other information received by the servers, may be used to create profiles of the natural persons and identify them.”

GDPR, recital 30 (May 2018)



when cookies can identify an individual, cookies are considered personal data, subject to GDPR personal data regulations

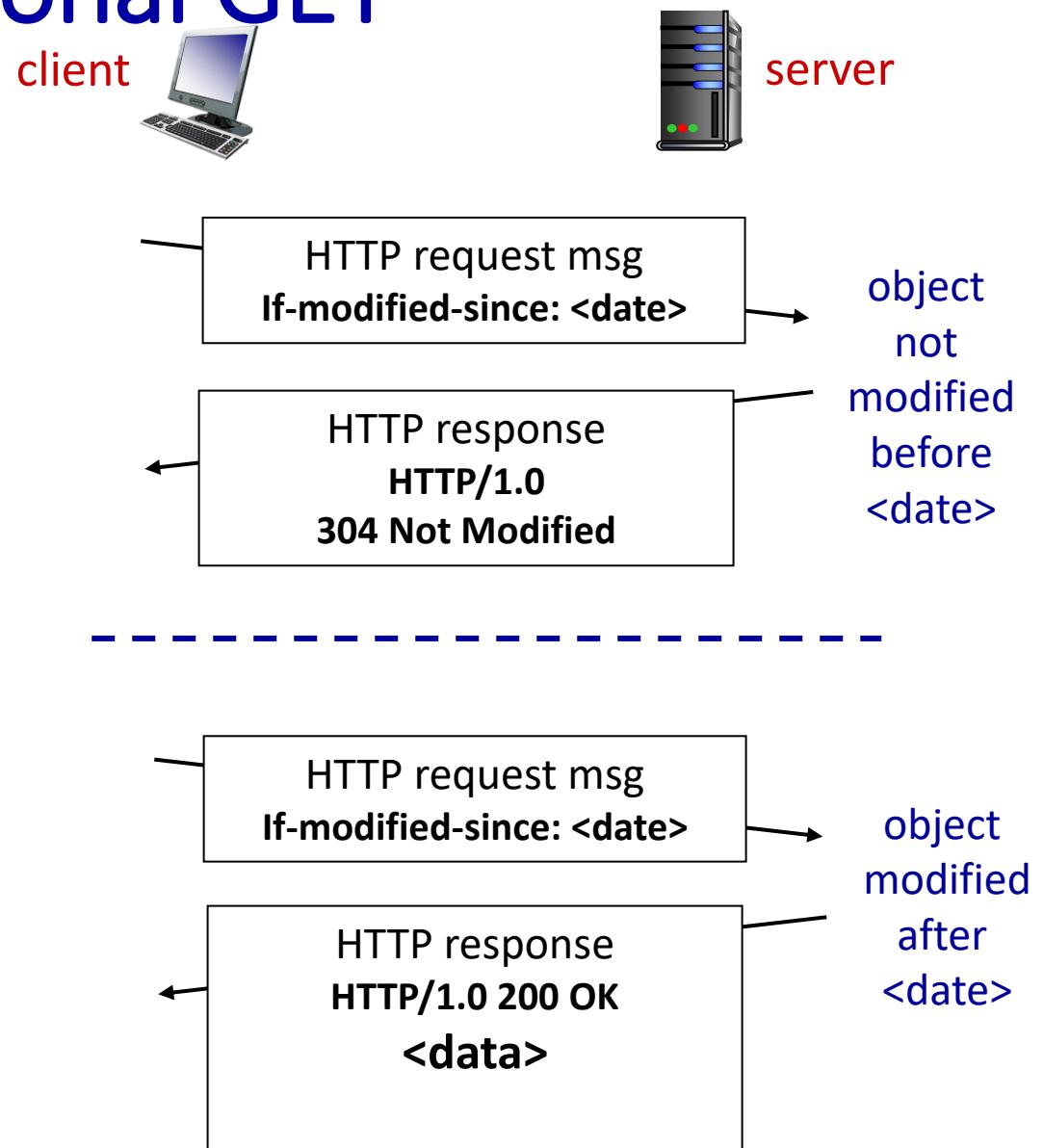


User has explicit control over whether or not cookies are allowed

Browser caching: Conditional GET

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

- no object transmission delay (or use of network resources)
- **client:** 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



2.2.6 HTTP/2

Key goal: reduce delay in multi-object HTTP requests,
avoid parallel TCP connections

HTTP1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (**head-of-line (HOL) blocking**) behind large object(s)

HTTP/2

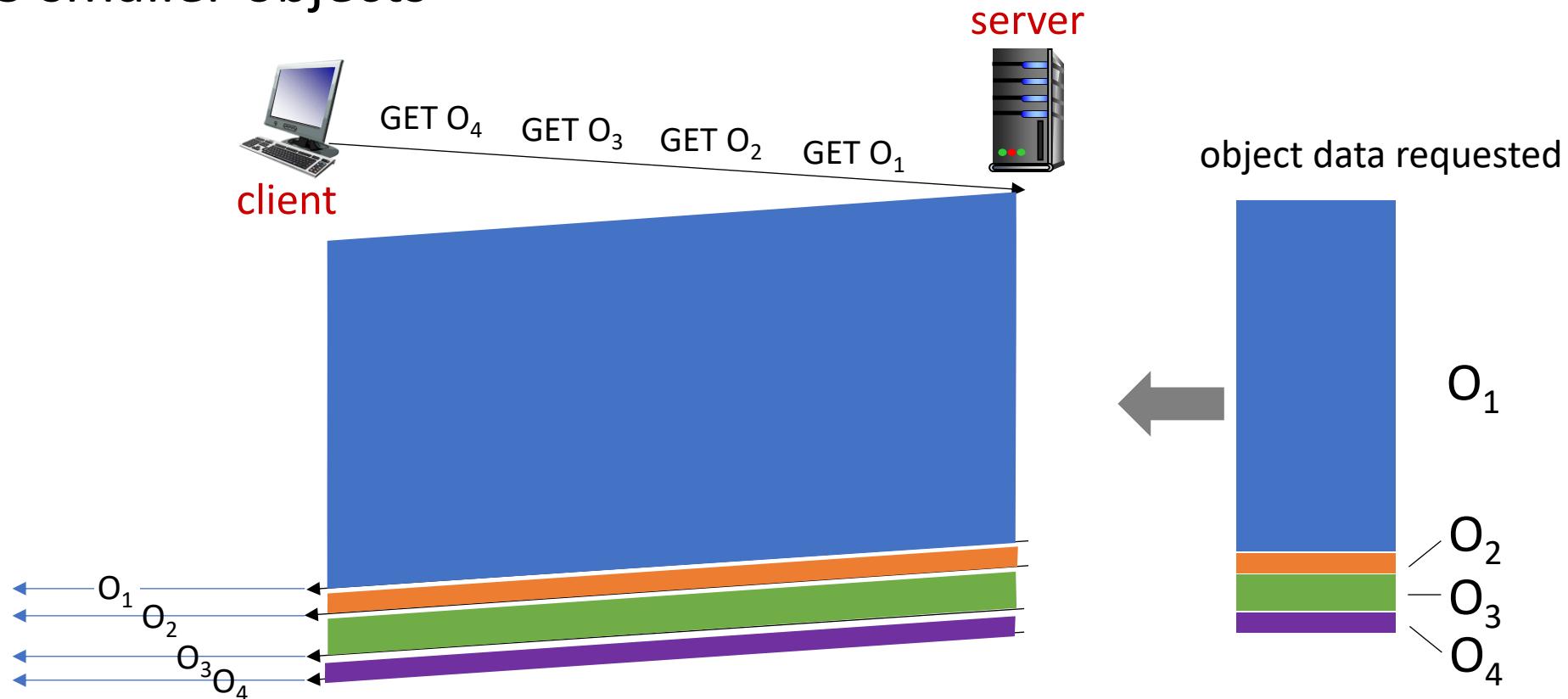
Key goal: reduce delay in multi-object HTTP requests, avoid parallel TCP connections

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields **unchanged** from HTTP/1.1, but represented in compressed binary frames
- transmission order of requested objects based on client-specified object priority
- **push** unrequested objects to client
- **mitigate HOL blocking** by dividing objects into frames, schedule frames using **interleaving**

HTTP/2: mitigating HOL blocking

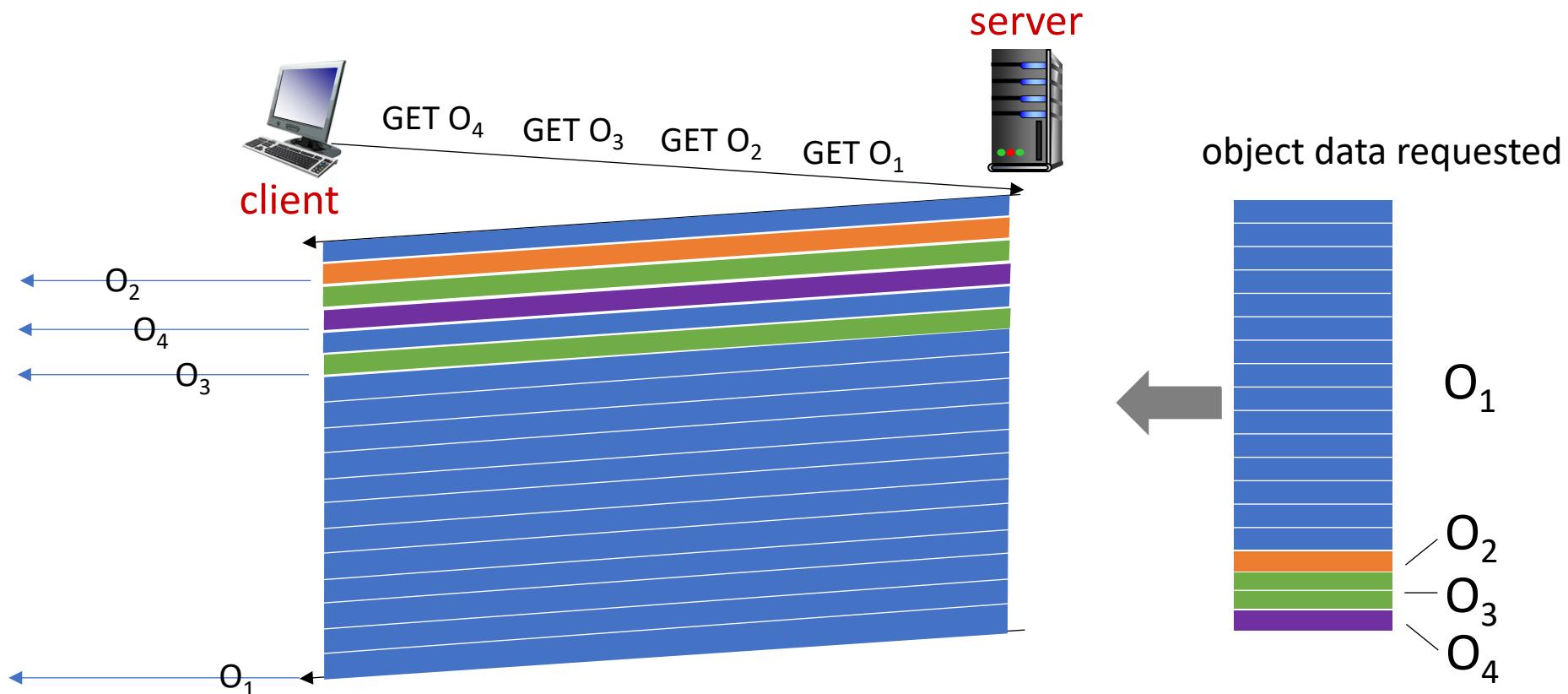
HTTP 1.1: client requests 1 large object (e.g., a big chunk of a video file) and 3 smaller objects



objects delivered in order requested: O_2 , O_3 , O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames; frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 video streaming and content distribution networks

2.3 E-mail, SMTP, IMAP

- 2.3.1 SMTP
- 2.3.2 Mail message formats
- 2.3.3 Mail access protocols

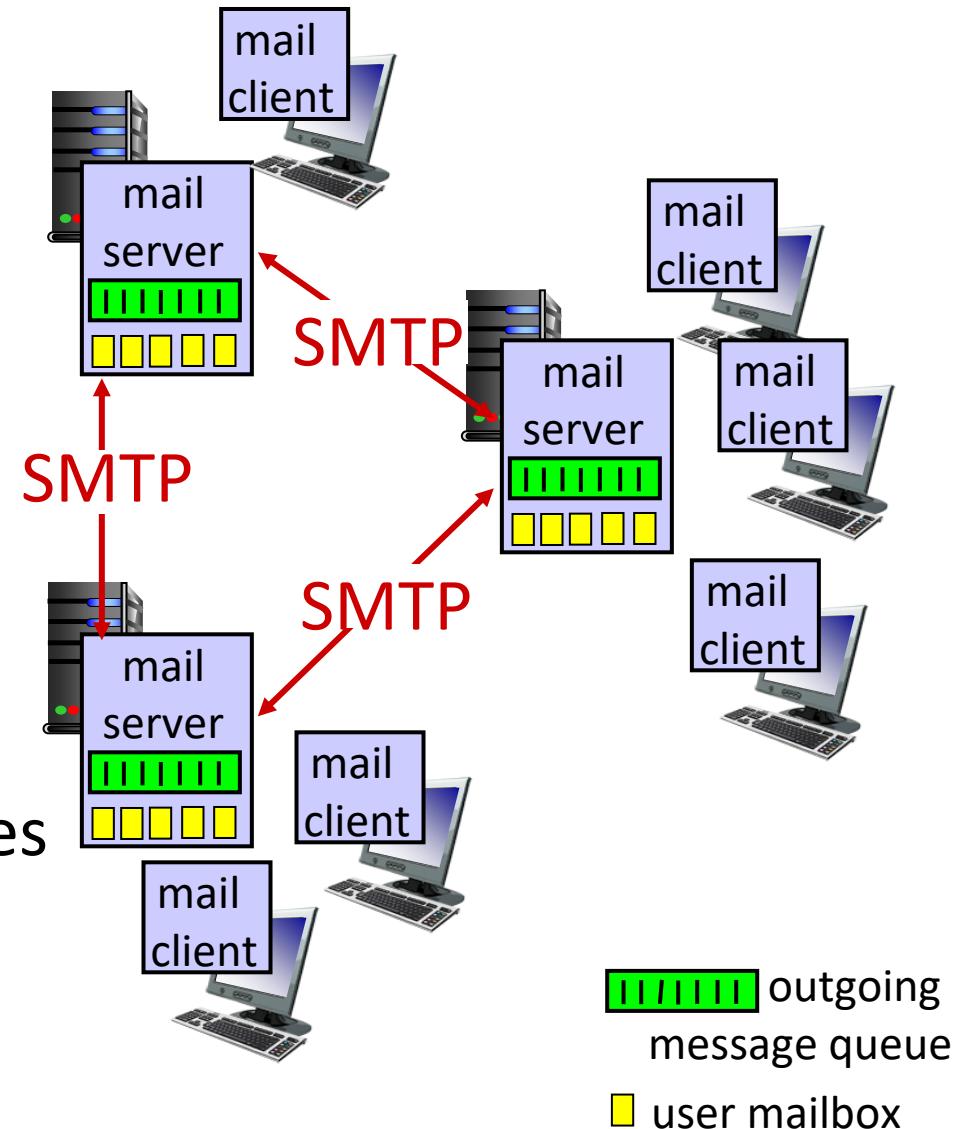
2.3 E-mail

Three major components:

1. mail client
2. mail servers
3. simple mail transfer protocol: SMTP

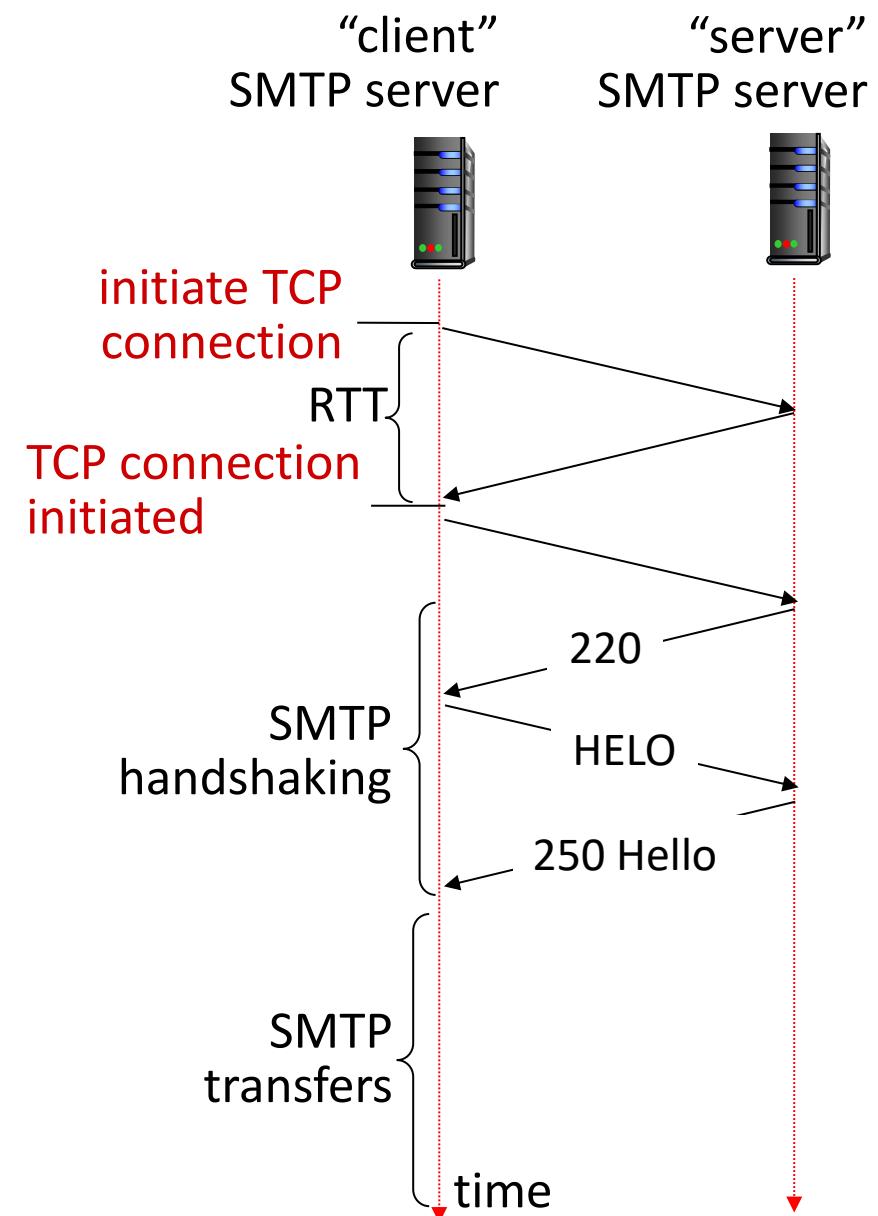
Mail client

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



2.3.1 SMTP RFC (821, 2811, 5321)

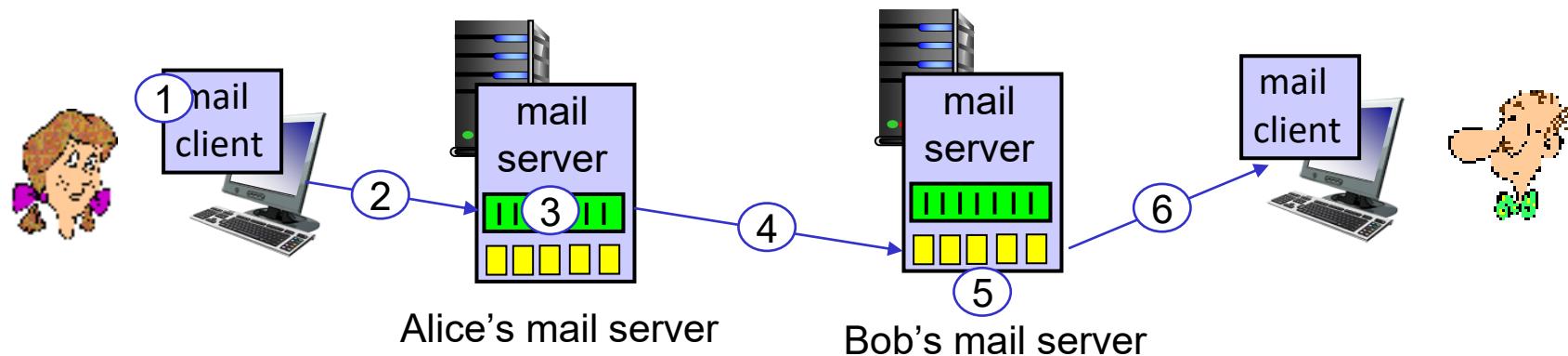
- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25 (or TLS port 587)
 - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
 - SMTP handshaking (greeting)
 - SMTP transfer of messages
 - SMTP closure
- command/response interaction (like HTTP)
 - commands: ASCII text
 - response: status code and phrase



Scenario: Alice sends e-mail to Bob

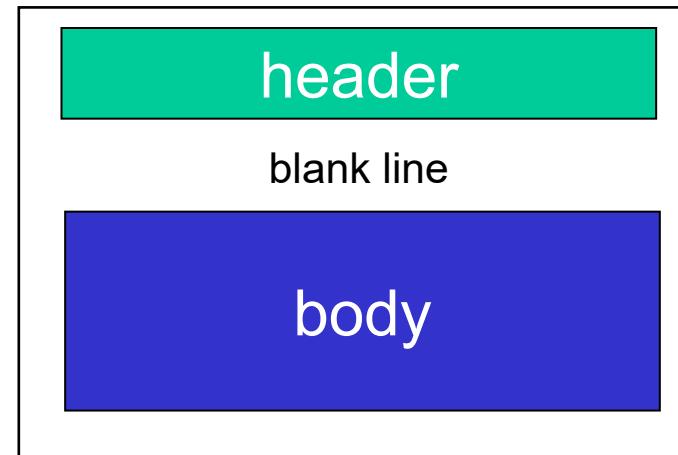
- 1) Alice uses MC to compose e-mail message “to” bob@someschool.edu
- 2) Alice’s MC sends message to her mail server using SMTP
- 3) client side of SMTP at mail server opens **TCP connection** with Bob’s mail server

- 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 mail client to read message



Sample SMTP interaction

```
S: 220 smtp.example.com
C: HELO relay.example.org
S: 250 Hello relay.example.org
C: MAIL FROM:<alice@example.org>
S: 250 Ok
C: RCPT TO:<bob@example.com>
S: 250 Ok
C: DATA
S: 354 Ready to receive
C: From: "Alice" <alice@example.org>
C: To: "Bob" <bob@example.com>
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```



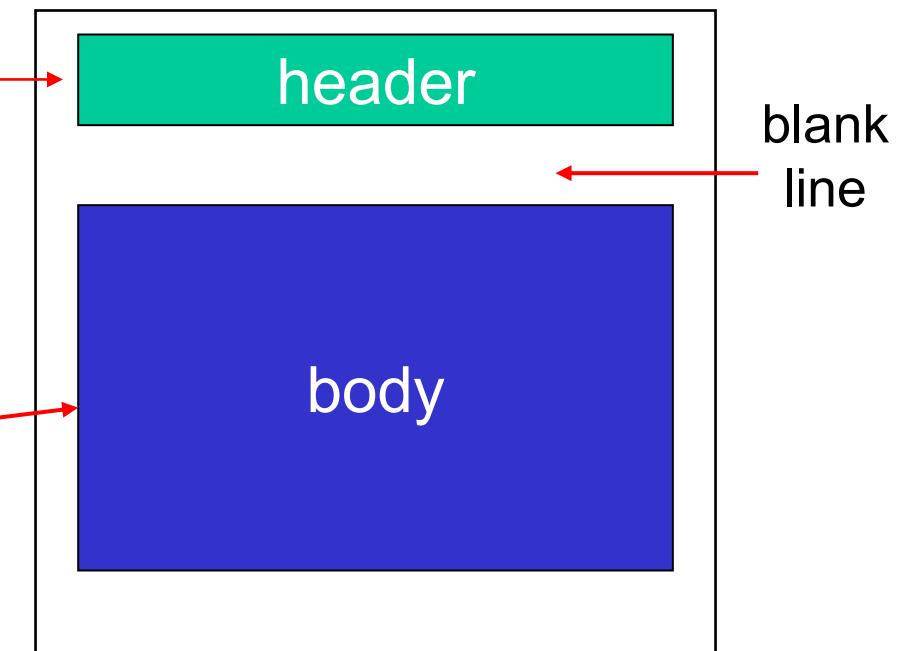
2.3.2 Mail message format

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

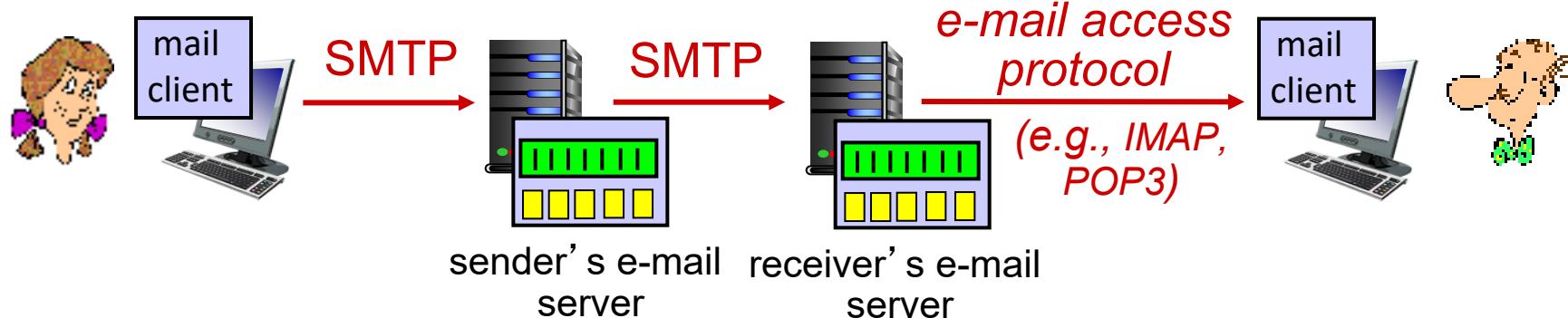
RFC 5322 defines **syntax** for e-mail message itself (like HTML defines syntax for web documents)

- 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



2.3.3 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]: “views” messages stored on server, IMAP provides retrieval, deletion
 - **POP3:** Post Office Protocol version 3. The email client downloads emails to client
- **HTTP:** gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP3) to retrieve e-mail messages

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- **2.4 The Domain Name System DNS**
- 2.5 Peer-to-peer file distribution
- 2.6 video streaming and content distribution networks

DNS: Domain Name System

Domain names:

- uia.no, youtube.com

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., vg.no - used by humans

■ Hostname-to-IP-address translation

- Internet “phone book”
- Domains vs. hosts

Domain Name System (DNS):

1. *distributed database* implemented in hierarchy of many *name servers*
2. *DNS protocol*
 - *application-layer protocol*: hosts, DNS servers communicate to *resolve* names (address/name translation)
 - Uses the **UDP** transport layer protocol using port 53
 - complexity at network’s “edge”

uia.no → 158.37.242.21, 158.37.218.21, 129.240.118.130, 158.37.242.20, 158.37.218.20

DNS: services, structure

DNS services:

- hostname-to-IP-address translation
- host aliasing
 - canonical, alias names
- mail server aliasing
 - e.g.: uia.no → mx.uhpost.no
- load distribution
 - replicated Web servers: many IP addresses correspond to one name

Q: Why not centralize DNS?

- single point of failure
- no redundancy
- 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

Very large distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

- *performance matters*: almost every Internet transaction interacts with DNS - msecs count!

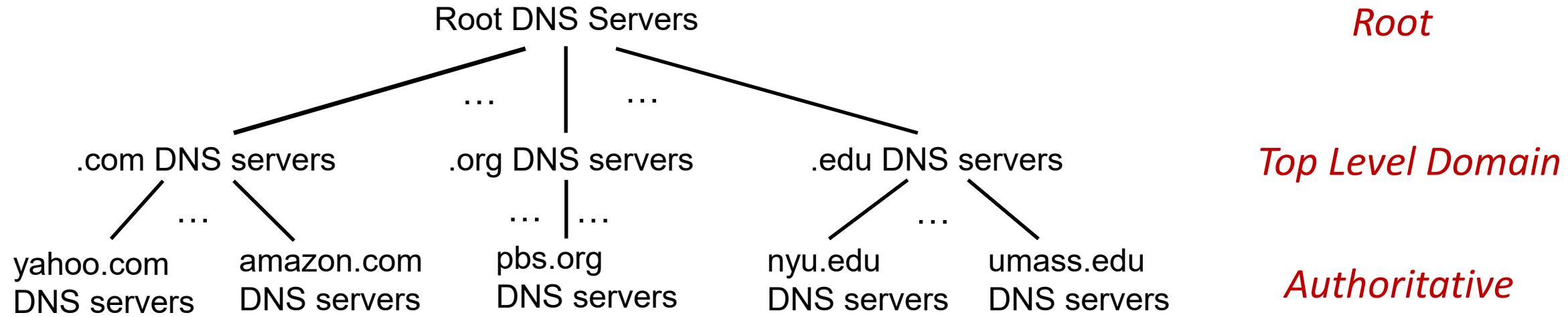
organizationally, physically decentralized:

- 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:

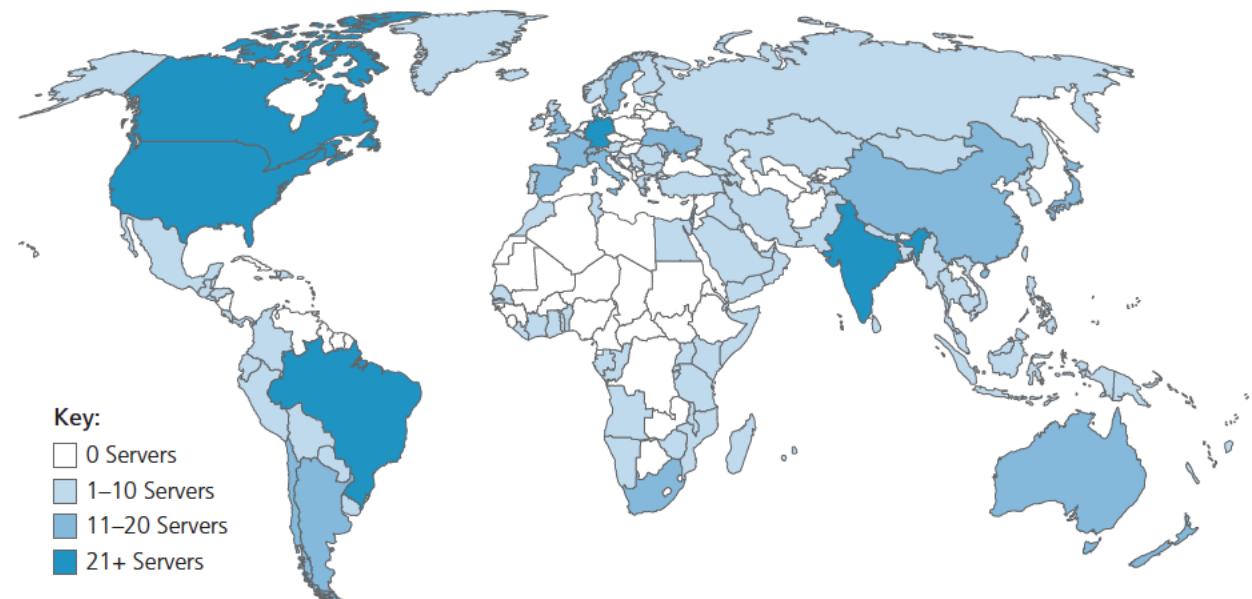
- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- 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)

e.g., A.ROOT-SERVERS.NET fixed IP 198.41.0.4



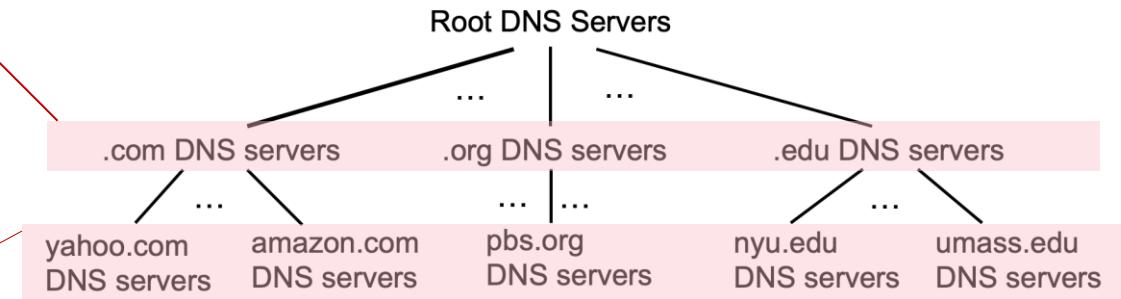
Top-Level Domain, and authoritative servers

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
- Answers requests by returning a list of the authoritative name servers

1058 TLDs :

- 730 generic TLD
- 301 country code 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

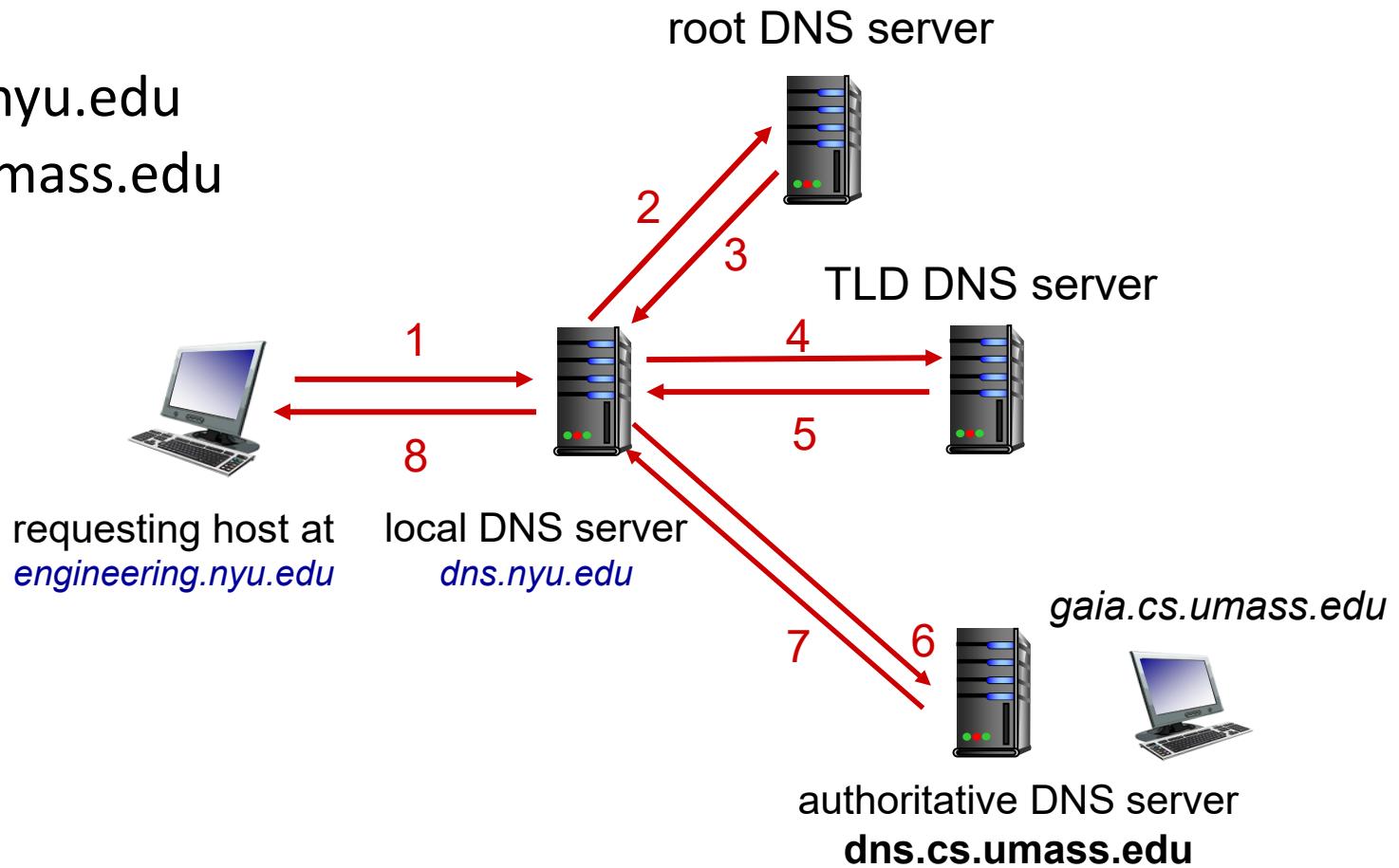
- Local DNS servers don't strictly belong to hierarchy
- When a host makes DNS query, it is sent to its *local* DNS server:
 - checks its local cache of recent name-to-address translation pairs (possibly out of date), or
 - forwarding request into DNS hierarchy for resolution
- Each ISP has a local DNS name server; to find yours:
 - C:\> ipconfig /all

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”



Looking up a host's IP address

```
C:\Users\sigurde>nslookup uia.no
Server: nv2.ti.telenor.net
Address: 2001:4600:4:1fff::253

Non-authoritative answer:
Name: uia.no
Addresses: 2001:700:100:118::130
           129.240.118.130
```

```
C:\Users\sigurde>nslookup uia.no
Server: adgrm01.uia.no
Address: 158.37.218.20 →

Name: uia.no
Addresses: 2001:700:100:118::130
           2001:700:1500:7250::242:20
           2001:700:1500:7250::242:21
           2001:700:1501:7250::218:20
           2001:700:1501:7250::218:21
           158.37.218.21
           129.240.118.130
           158.37.242.20 ←
           158.37.218.20 ←
           158.37.242.21
```

Looking up IP address of an authoritative DNS server

```
C:\Users\sigurde>nslookup -query=soa uia.no
Server: nv2.ti.telenor.net
Address: 2001:4600:4:1fff::253

Non-authoritative answer:
uia.no
    primary name server = ns1.uia.no
    responsible mail addr = hostmaster.uia.no
    serial = 2025012300
    refresh = 28800 (8 hours)
    retry = 3600 (1 hour)
    expire = 86400 (1 day)
    default TTL = 3600 (1 hour)
```

```
C:\Users\sigurde>nslookup uia.no ns1.uia.no
Server: ns1.uia.no
Address: 2001:700:1500:d270::245:200

Name: uia.no
Addresses: 2001:700:100:118::130
           129.240.118.130
```

DNS records

DNS: distributed database storing resource records

format: (name, ttl, type, value)

type=A

- name is hostname
- value is IPv4 address

www.demosite.com. 3600 A 207.124.120.25

type=CNAME

- name refers to a “canonical” name specified in value

shop.example.com. 3600 CNAME shops.myshopify.com.

type=NS

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

demosite.com. 3600 NS ns1.demosite.net.

type=MX

- name is domain (e.g., uia.no)
- value is name of SMTP mail server for this domain

demosite.com. 3600 MX 10 mail1.demosite.com.

DNS records

DNS: distributed database storing resource records

type=SOA

- A start of authority (SOA) is a DNS record with information about authoritative name-server for a domain name

```
C:\Users\sigurde>nslookup -query=a uia.no
Server: adgrm01.uia.no
Address: 158.37.218.20

Name: uia.no
Addresses: 158.37.218.21
          129.240.118.130
          158.37.242.20
          158.37.218.20
          158.37.242.21
```

```
Address: 2001:4600:4:1fff::253
```

```
Non-authoritative answer:
```

```
uia.no nameserver = ns2.uia.no
uia.no nameserver = nn.uninett.no
uia.no nameserver = ns1.uia.no
```

```
nn.uninett.no    AAAA IPv6 address = 2001:700:0:503::aa:5302
ns1.uia.no      AAAA IPv6 address = 2001:700:1500:d270::245:200
ns2.uia.no      AAAA IPv6 address = 2001:700:1501:d270::221:200
nn.uninett.no    internet address = 158.38.0.181
ns1.uia.no      internet address = 158.37.245.200
ns2.uia.no      internet address = 158.37.221.200
```

```
Non-authoritative answer:
```

```
uia.no MX preference = 10, mail exchanger = mx.uhpost.no
```

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!*

Locally cached DNS information

```
C:\Users\sigurde>ipconfig /displaydns

Windows IP Configuration

finn.no
-----
Record Name . . . . : finn.no
Record Type . . . . : 1
Time To Live . . . . : 297
Data Length . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 35.228.105.46

finn.no
-----
No records of type AAAA
```

Registering domain info into the DNS

example: new startup “Network Utopia”

- register name networkuptopia.com at *DNS registrar*
(I Norge: www.uniweb.no, www.norid.no)
 - provide names, IP addresses of authoritative name server (primary and secondary)
 - registrar inserts **NS** and **A** resource records into .com **TLD server**:
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
 - type A record mapping www.networkuptopia.com to an IP address
 - 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

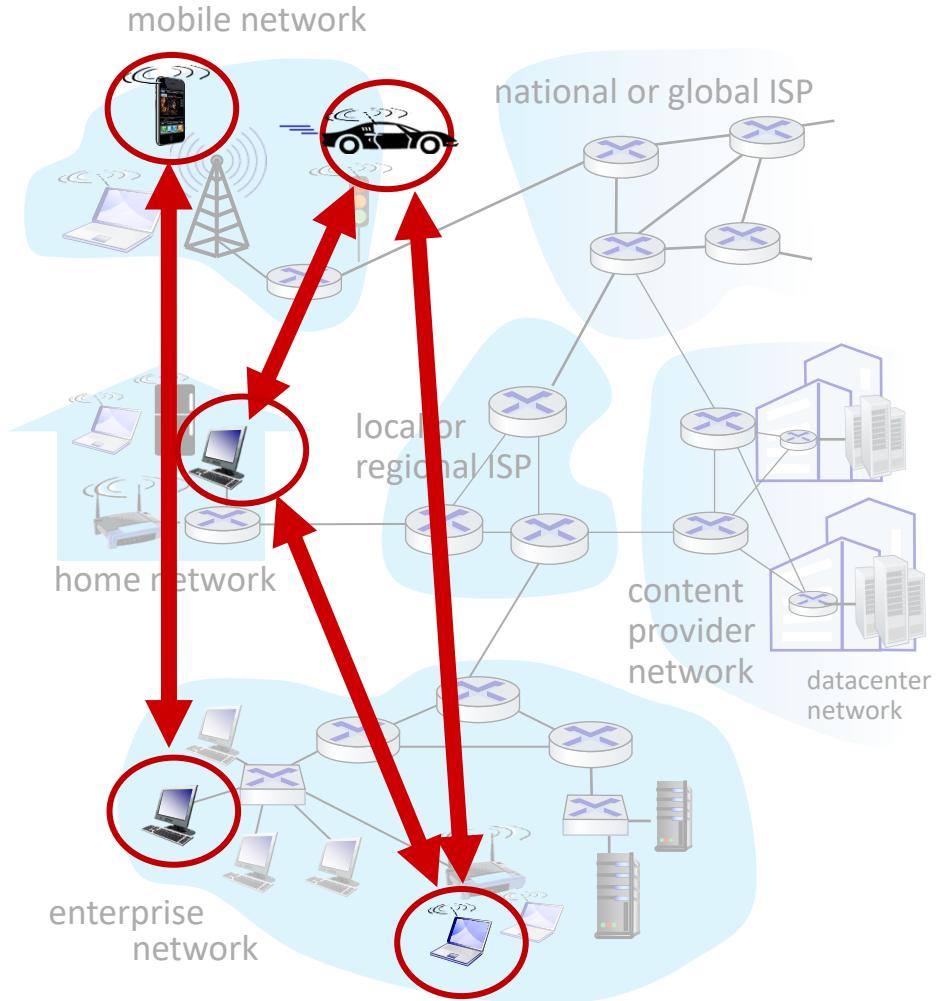
- intercept DNS queries, returning bogus replies
 - DNS cache poisoning
 - RFC 4033: DNSSEC authentication services

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- **2.5 Peer-to-peer file distribution**
- 2.6 video streaming and content distribution networks

Peer-to-peer (P2P) model

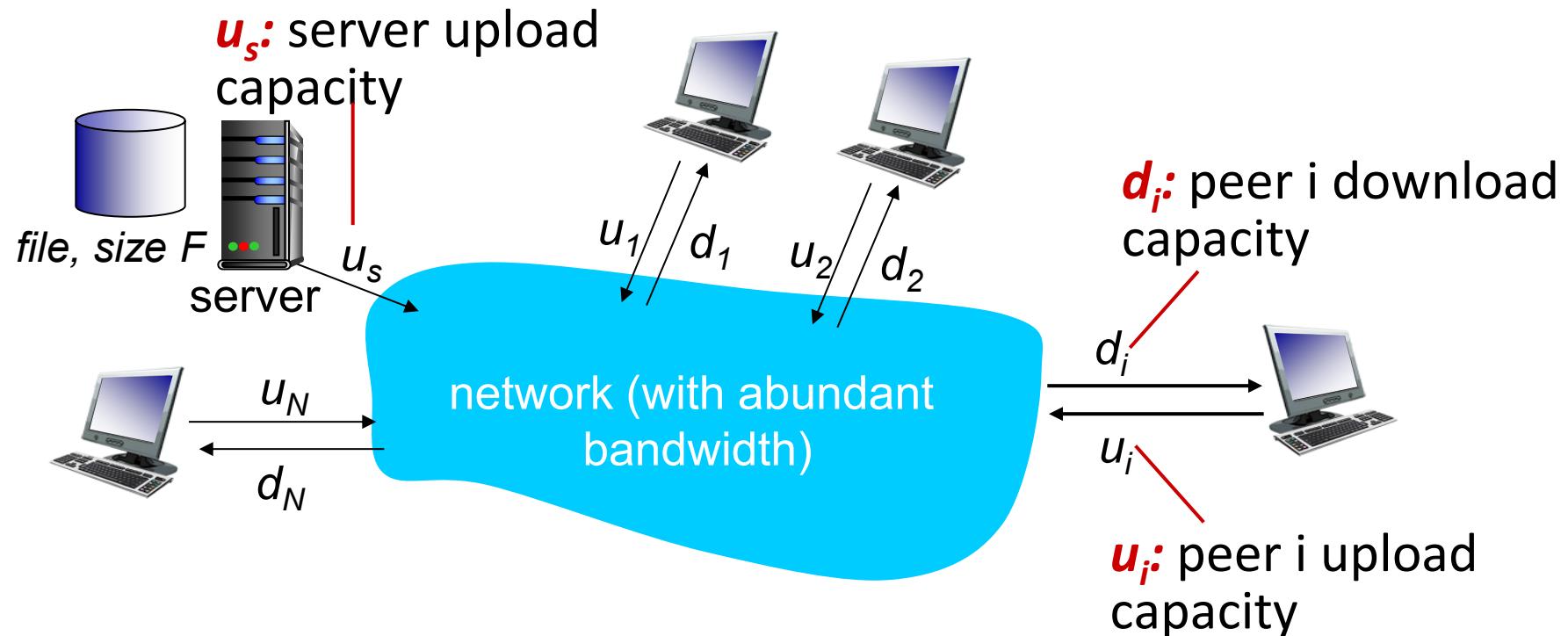
- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
 - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
 - complex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



File distribution: client-server vs P2P

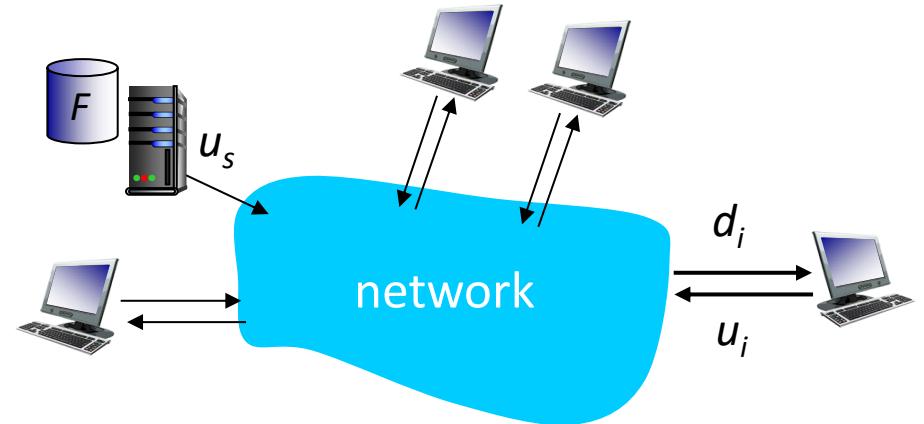
Q: how much time to distribute file (size F) from one server to N peers?

- peer upload/download capacity is limited resource



File distribution time: client-server

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



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

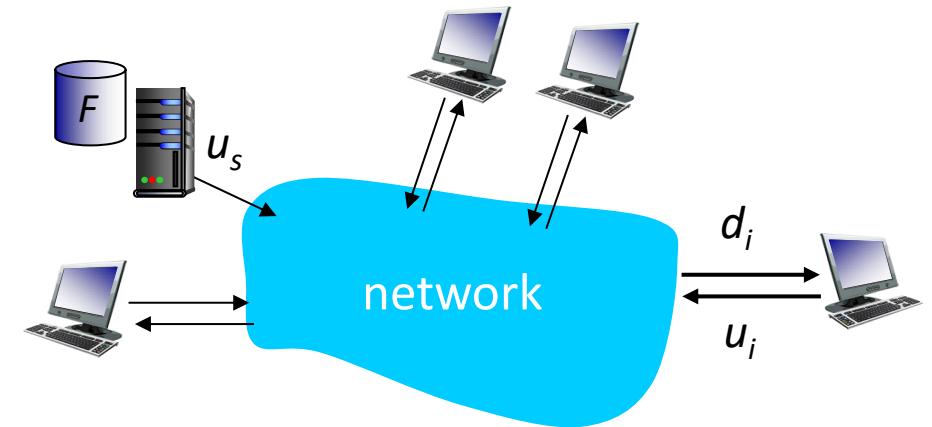
$$D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

- *server transmission*: must upload at least one copy:

- time to upload one copy: $\frac{F}{u_s}$



- *client*: each client must download file copy

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

- *clients*: upload NF bits

- total upload time: $\frac{N \cdot F}{u_s + \sum u_i}$

minimum time to distribute F to N clients using P2P approach

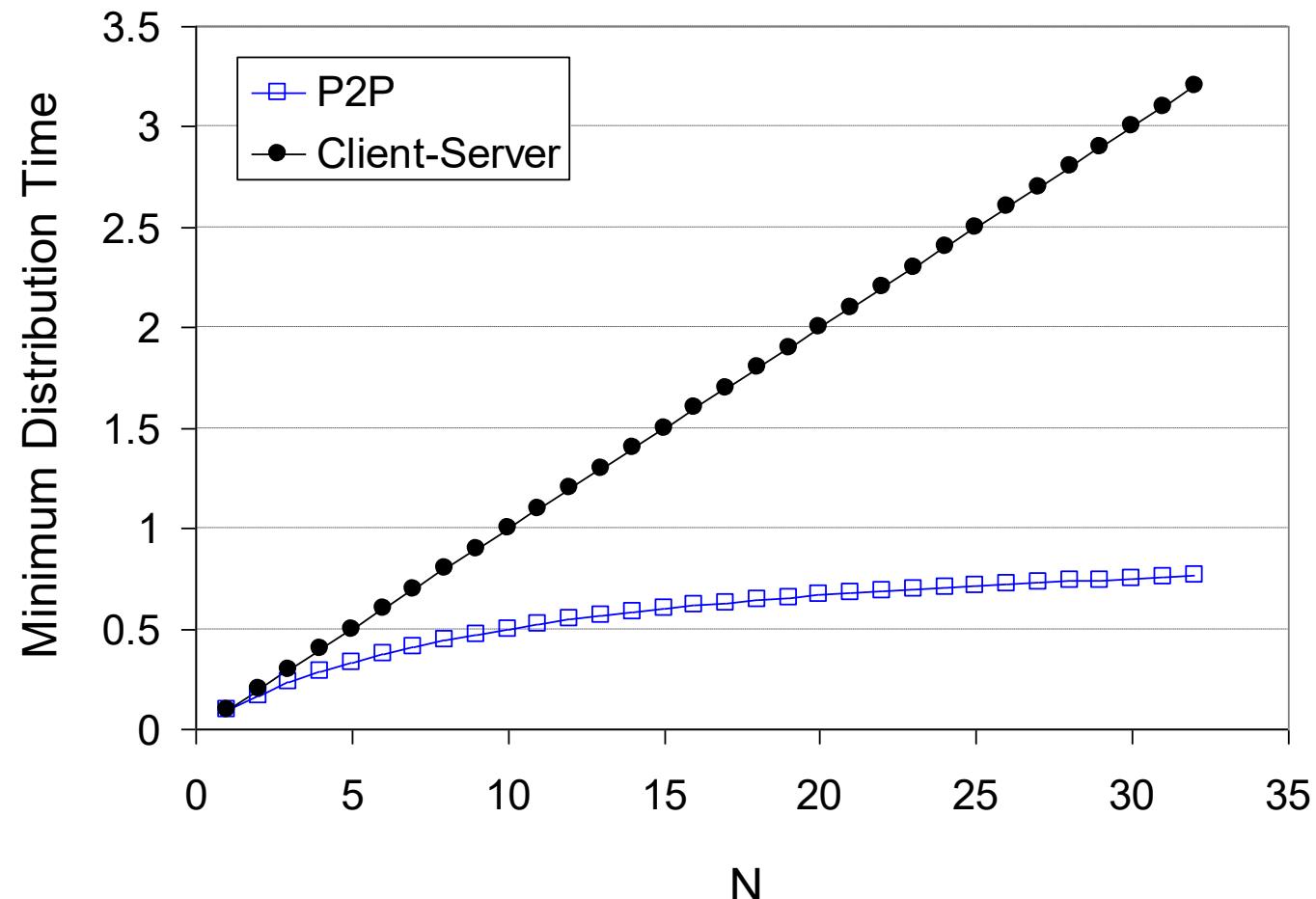
$$D_{P2P} > \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{N \cdot F}{u_s + \sum u_i}\right\}$$

increases linearly in N ...

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

Client-server vs. P2P: example

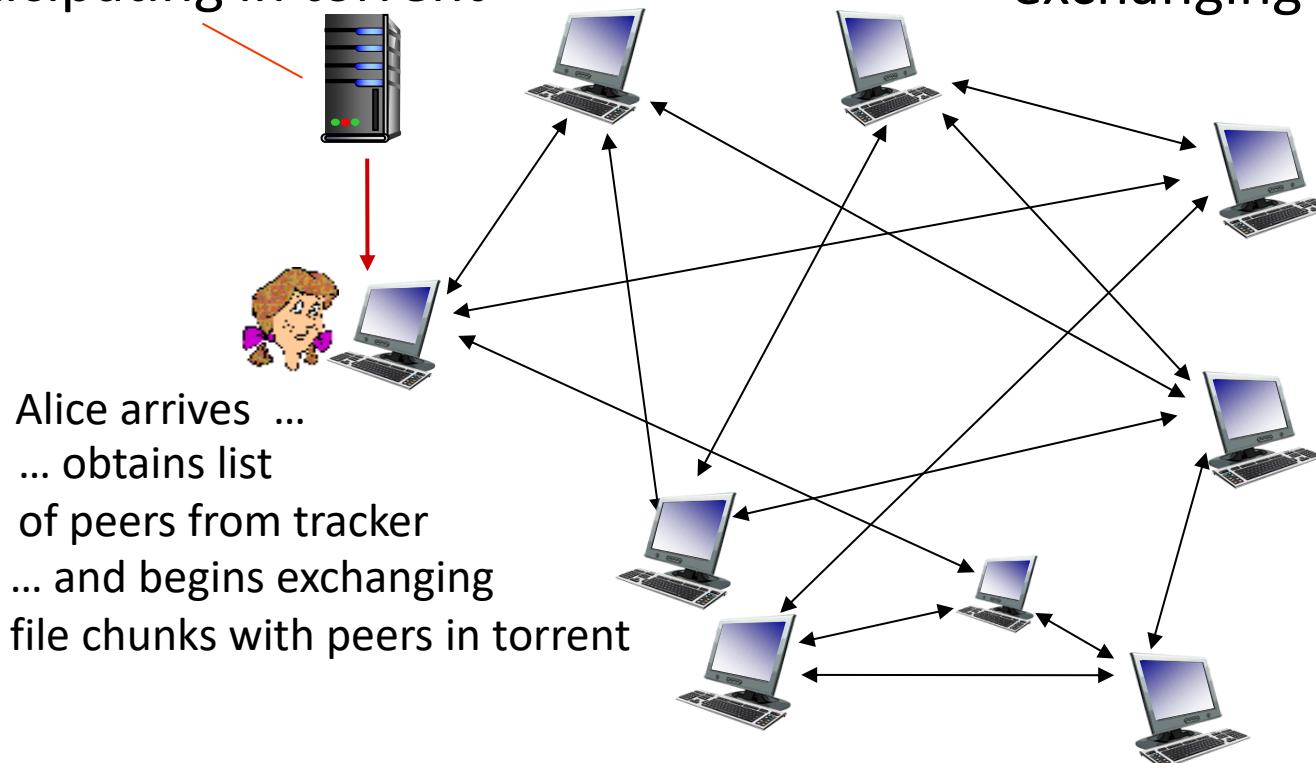
client upload rate = u , $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$



P2P file distribution: BitTorrent

- file divided into 512KB (or less) chunks called pieces
- peers in torrent send/receive file piece by piece

tracker: tracks peers
participating in torrent



torrent: group of peers
exchanging chunks of a file

Application layer: overview

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 E-mail, SMTP, IMAP
- 2.4 The Domain Name System DNS
- 2.5 Peer-to-peer file distribution
- 2.6 Video streaming and content distribution networks

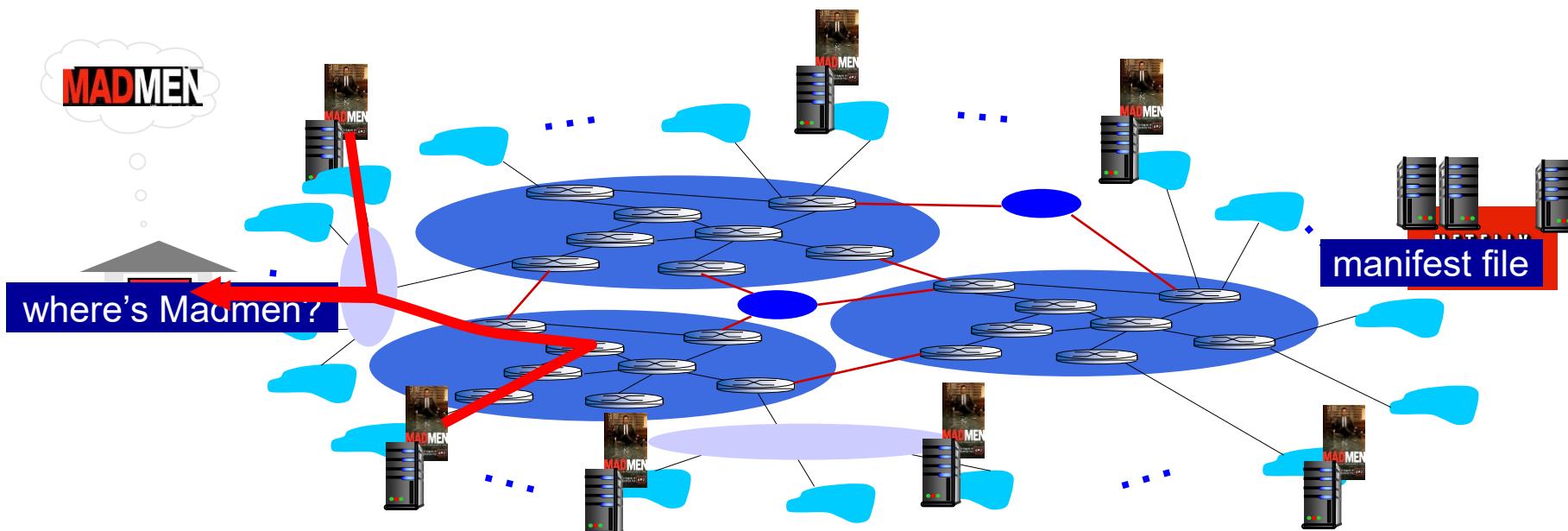
2.6 Video Streaming and CDNs: context

- stream video traffic: major consumer of Internet bandwidth
 - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge:* scale - how to reach ~1 billion users?
- *challenge:* heterogeneity
 - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure



Content distribution networks (CDNs)

- CDN: stores copies of video content at CDN nodes
- subscriber requests content, service provider returns manifest
 - using manifest, client retrieves content at highest supportable rate
 - may choose different rate or copy if network path congested



Chapter 2: Summary

our study of network application layer is now complete!

- application models
 - 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
- video streaming, CDNs
- UDP og TCP-socket
programmering
i neste kapittel

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”

Chapter 3

Transport

Layer

A note on the use of these PowerPoint slides:

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

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any -lides 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

EIGHTH EDITION

James F. Kurose • Keith W. Ross



Computer Networking: A
Top-Down Approach

8th edition

Jim Kurose, Keith Ross
Pearson, 2020

Transport layer: roadm

- 3.1 Transport-layer services
- 3.2 Port numbers
- 3.3 Connectionless transport: UDP
 - UDP socket programming
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
- 3.6 Principles of congestion control
- 3.7 TCP congestion control
- 3.8 QUIC: Quick UDP Internet Connections

Computer Networking

A Top-Down Approach

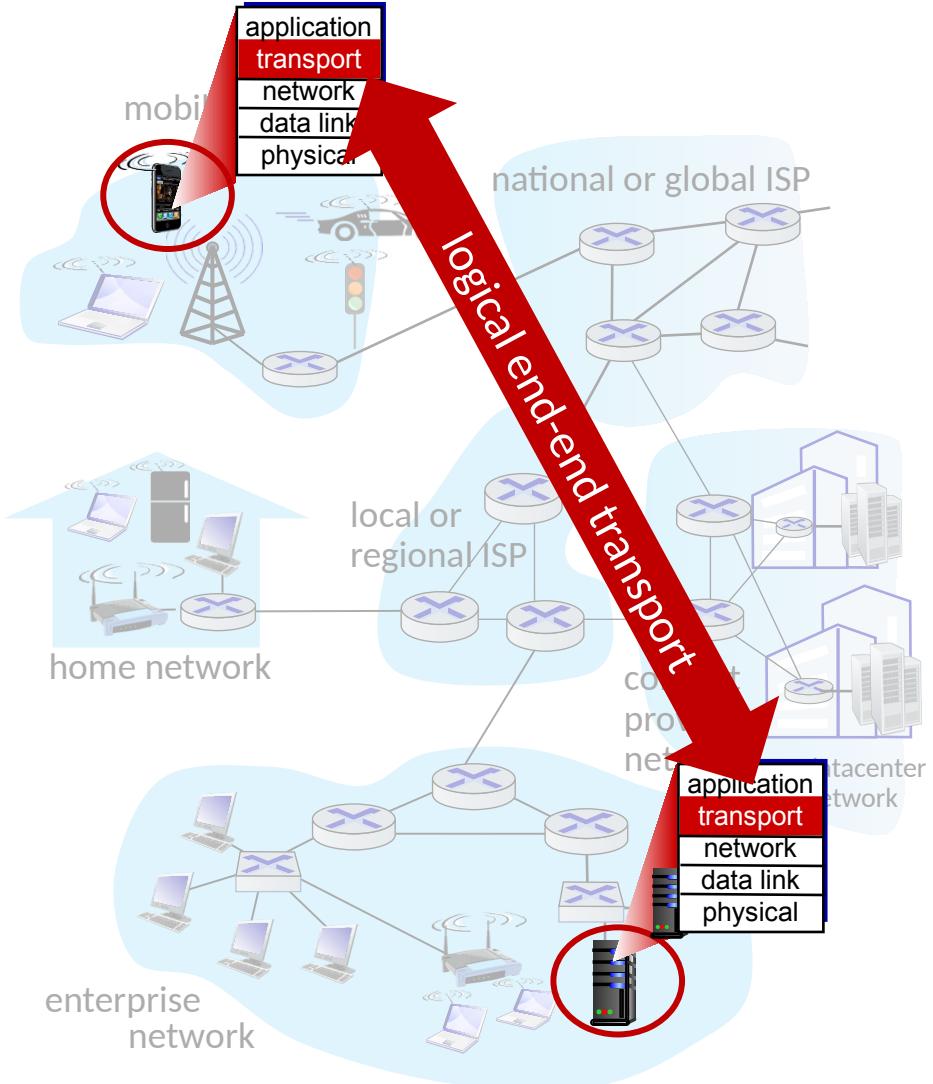
EIGHTH EDITION

James F. Kurose • Keith W. Ross



3.1 Transport layer

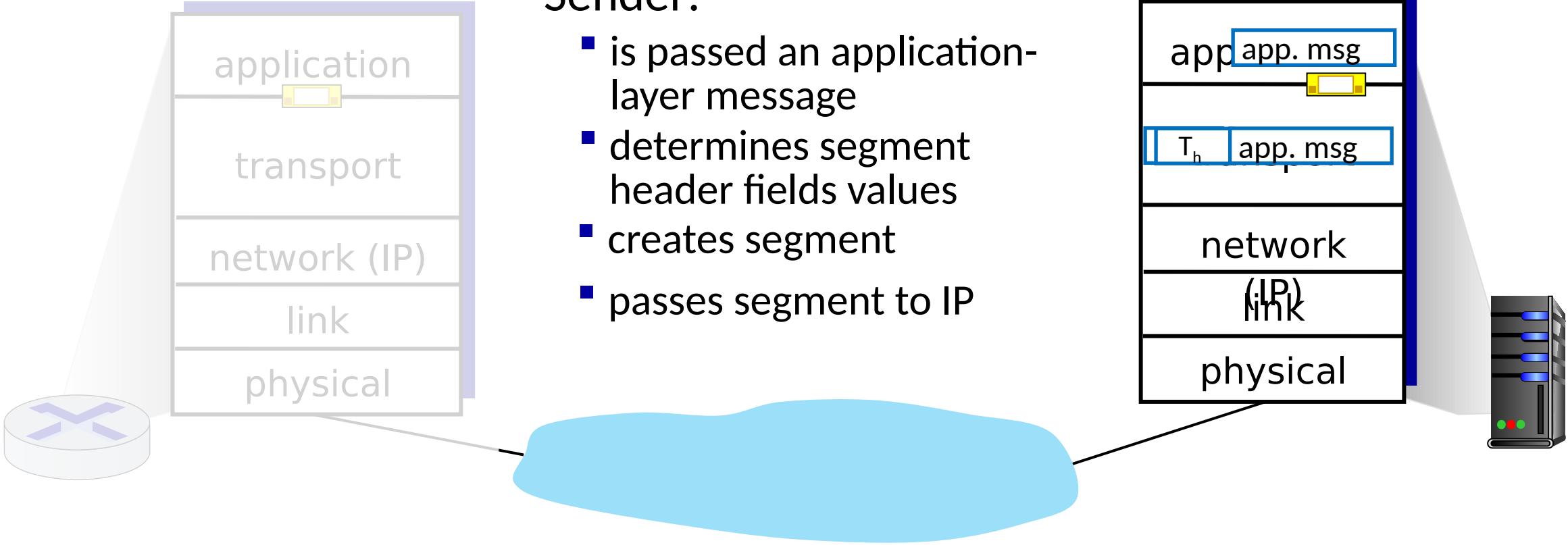
- Transport layer protocols
 - End-to-end protocols
 - Implemented in the communicating end nodes, *not* in the network itself
- provide *logical communication* between *application processes* running on different hosts
- two transport protocols
 - TCP, UDP



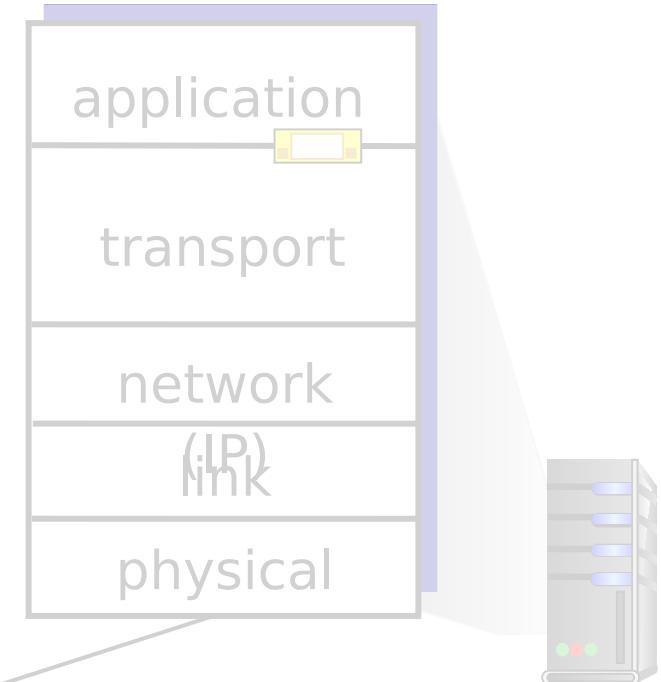
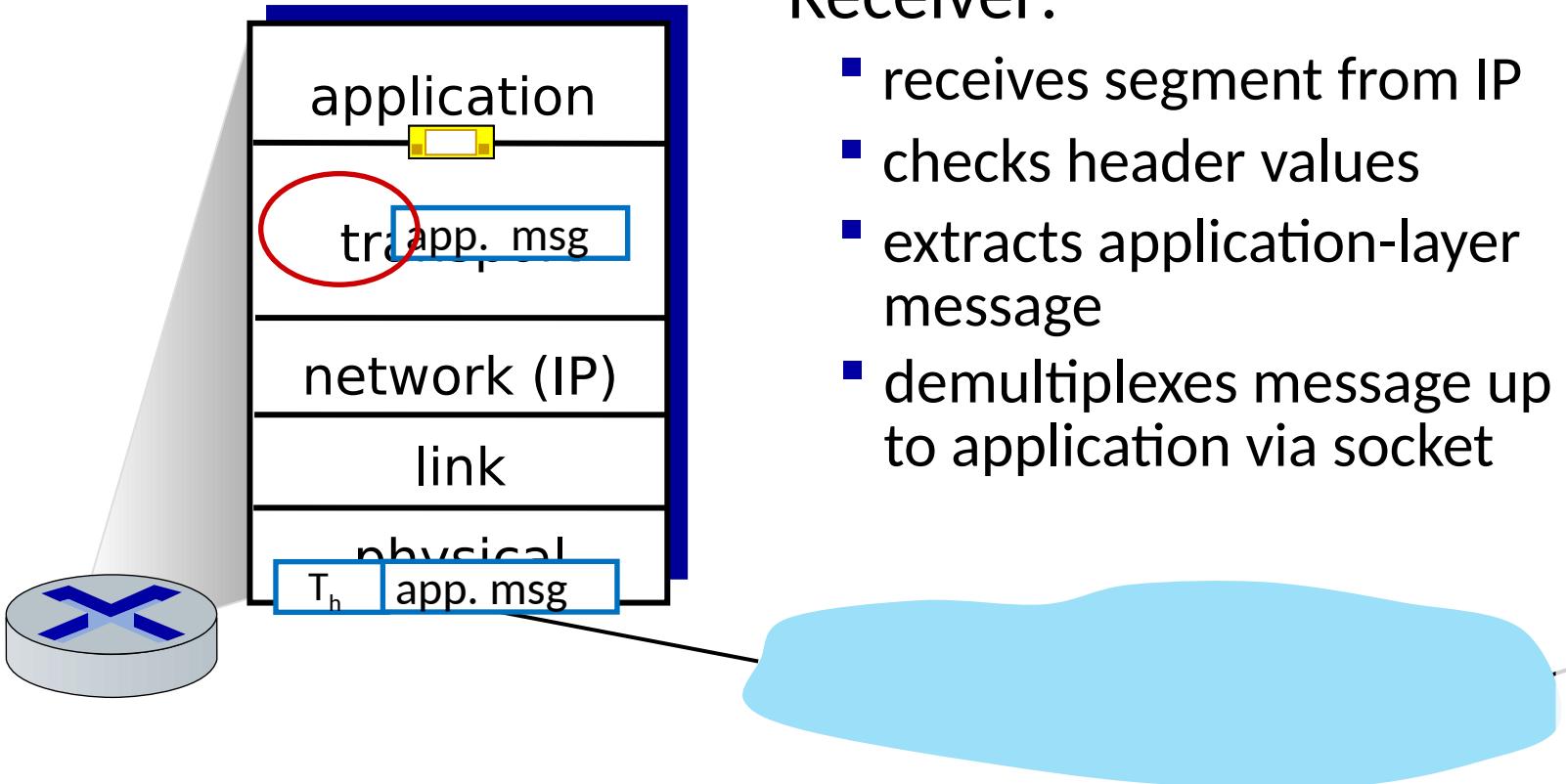
Transport vs. network layer services and protocols

- network layer: logical communication between *hosts*
 - IP addressing
- transport layer: logical communication between *processes*
 - relies on the network layer

Transport Layer Actions



Transport Layer Actions



Comparison of UDP and TCP

UDP:

- Port numbers
- Integrity check
- Connectionless data transmission
- No data segmentation
- Not reliable data transfer

TCP:

- Port numbers
- Integrity check
- Connection-oriented data transmission
- Data segmentation
- Reliable data transfer
 - 1. flow control
 - 2. congestion control

services not available:

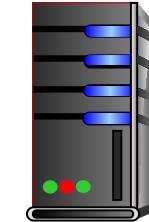
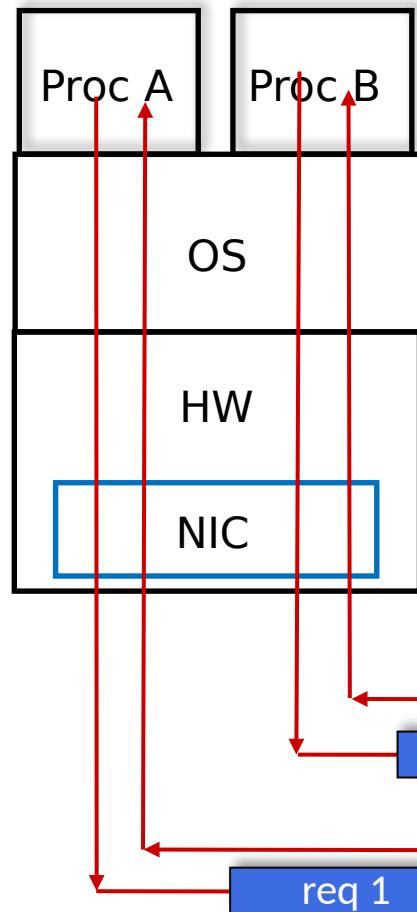
delay guarantees and bandwidth guarantees

Chapter 3: roadmap

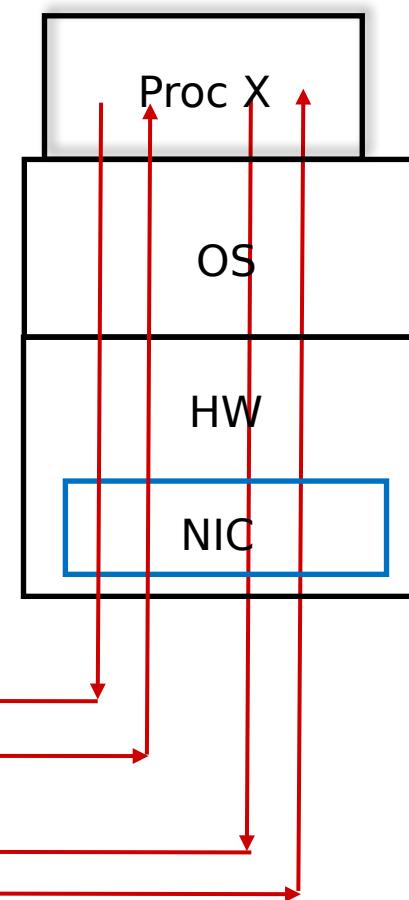
1. Transport-layer services
- 2. Port numbers**
3. Connectionless transport: UDP
 - UDP socket programming
4. Principles of reliable data transfer
5. Connection-oriented transport: TCP
6. ~~Principles of congestion control~~
7. TCP congestion control
8. QUIC: Quick UDP Internet Connections



IP address: 1.2.3.4

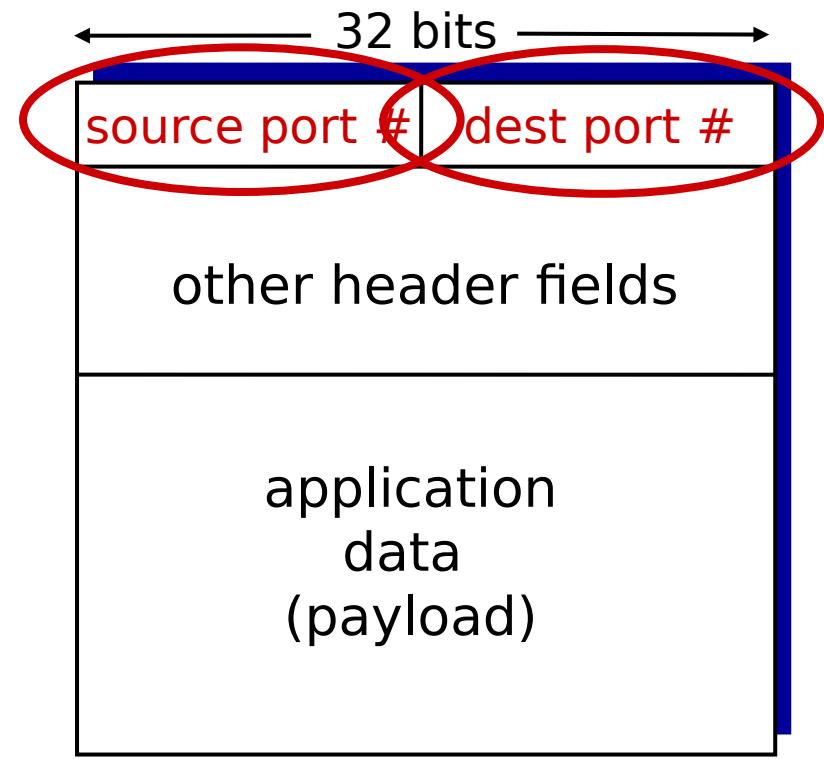


IP address: 3.3.3.3



Port numbers

1. IP addresses **identify hosts** in the network
2. Port numbers is a method to **identify** network applications processes
3. Implemented as 16-bit fields in **TCP and UDP segments**
 - Source port number
 - Destination port number
4. **IP packets** contain
 - source and target IP addresses
 - an **encapsulated** UDP or TCP packet (with port numbers)
5. ☺ IP addresses and port numbers are associated
 - **Source IP address + Source port number**
 - **Destination IP address + Destination port number**



TCP/UDP segment format

Port numbers

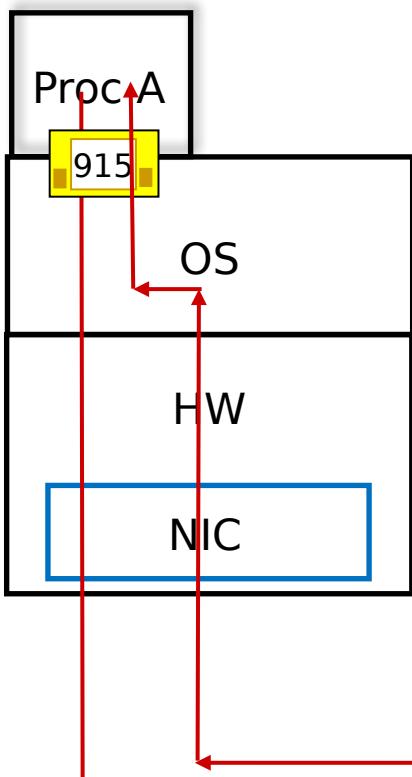
Three categories:

1. Well-known port numbers: **0 – 1023**
 - Reserved for widely used protocols and services, e.g. HTTP, HTTPS, SMTP, etc.
2. Registered port numbers: **1024 – 49151**
 - Used by specific applications and services.
3. Dynamic/private ports: **49152 – 65535**
 - Temporarily assigned for communication between clients and servers.

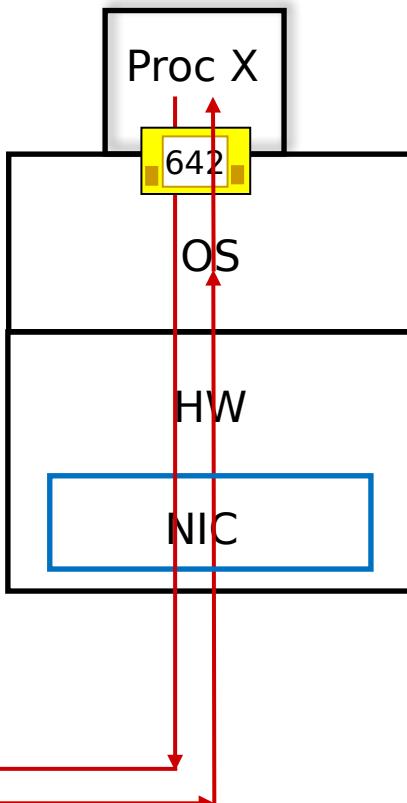
Some well-known port numbers

Application protocol	Port number	Protocol
DNS	53	UDP
HTTP (plaintext)	80	TCP
HTTPS (TLS/SSL)	443	TCP
SMTP (plaintext)	25	TCP
SMTP (TLS/SSL)	587, 465	TCP
POP3 (plaintext)	110	TCP
POP3 (TLS/SSL)	995	TCP
IMAP (plaintext)	143	TCP
IMAP (TLS/SSL)	993	TCP

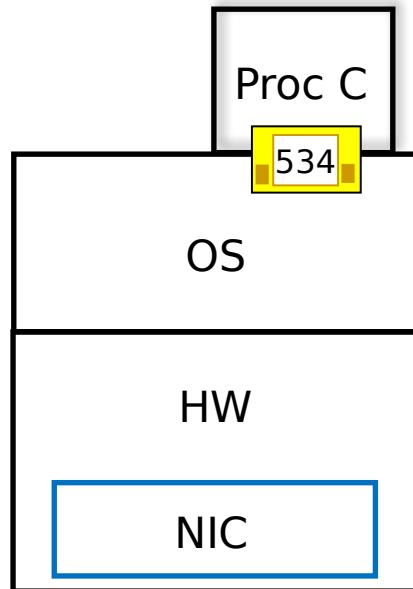

IP address: 1.2.3.4

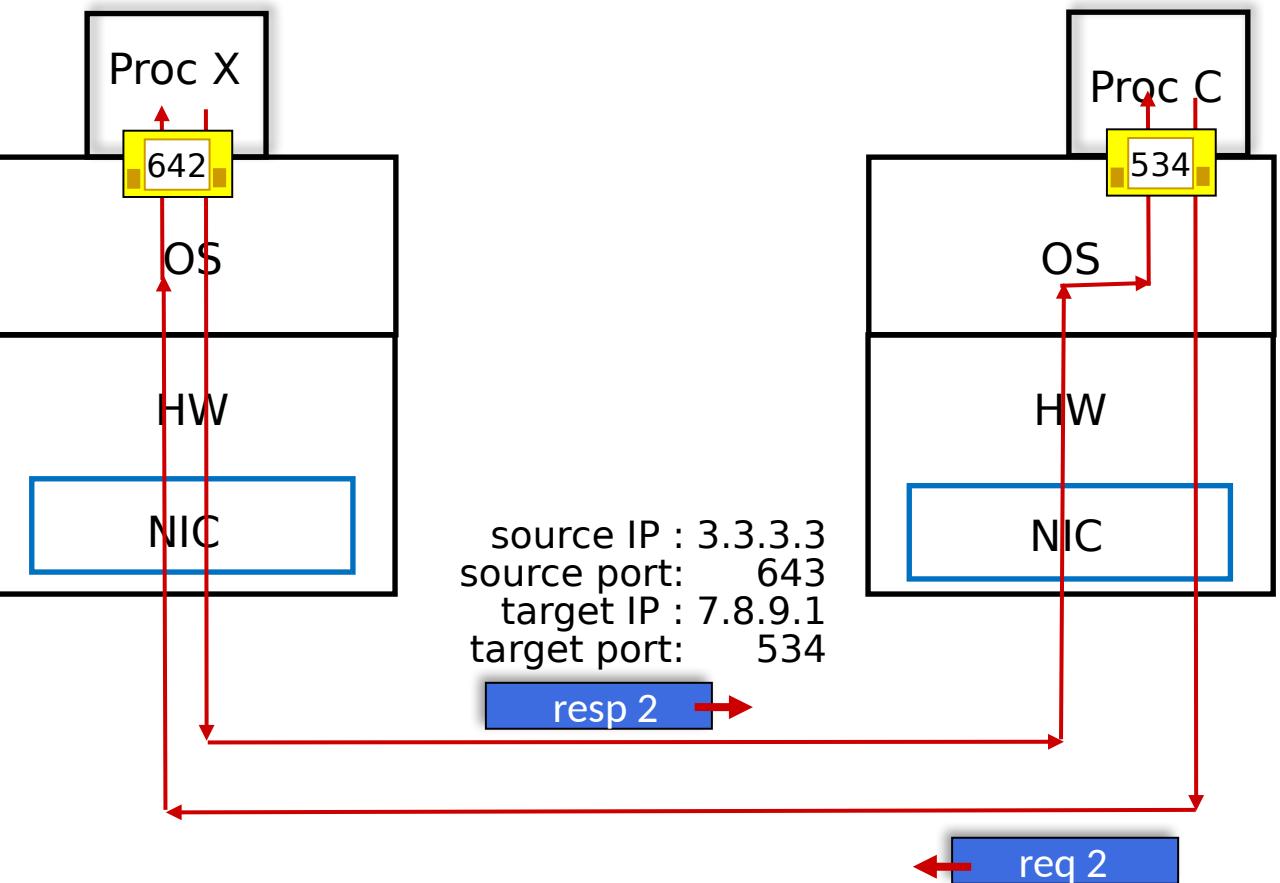
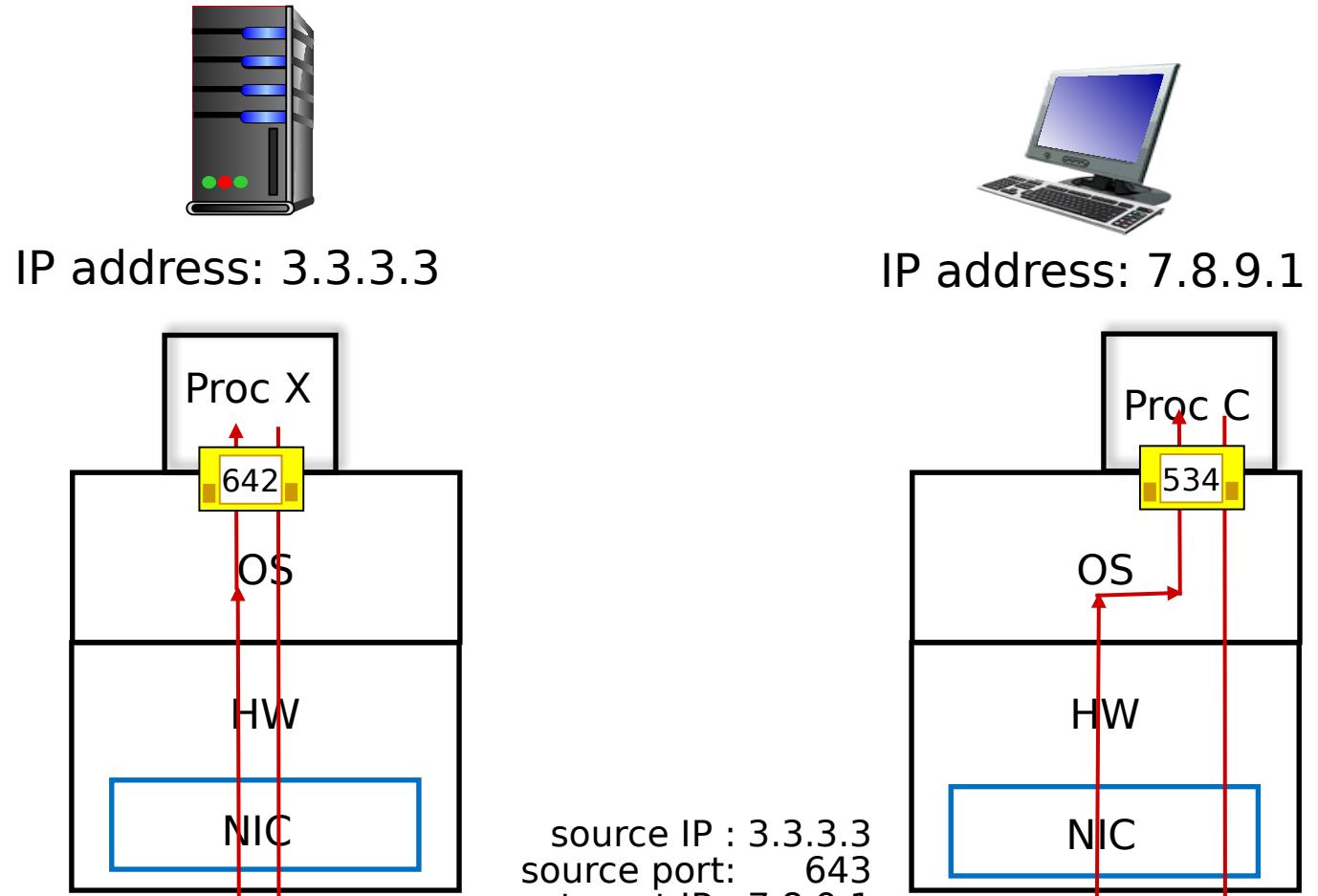
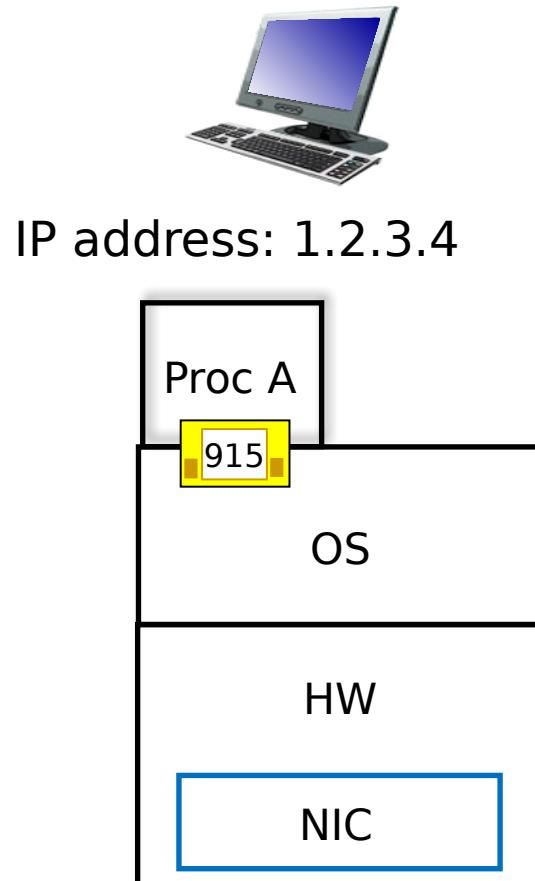



IP address: 3.3.3.3




IP address: 7.8.9.1





source IP : 3.3.3.3
source port: 643
target IP : 7.8.9.1
target port: 534

resp 2 →

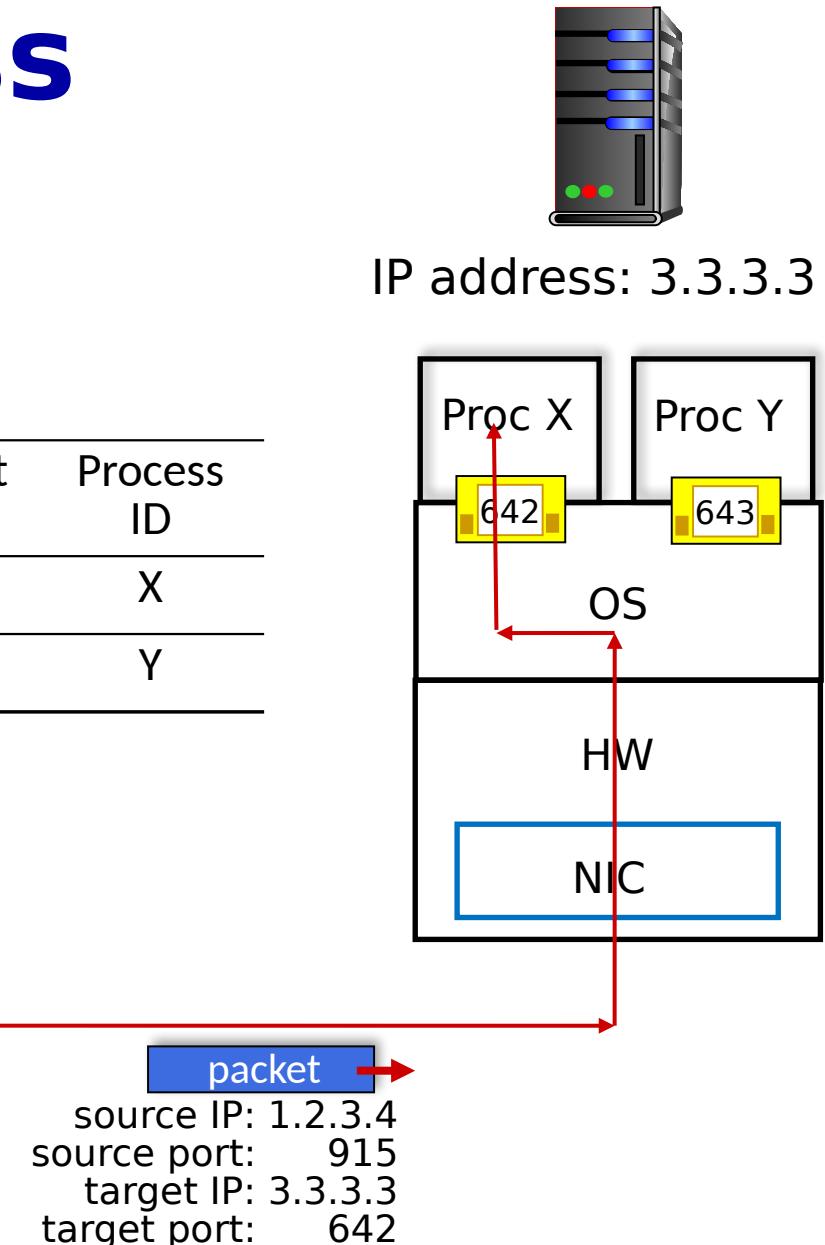
← req 2

source IP: 7.8.9.1
source port: 534
target IP: 3.3.3.3
target port: 643

Selecting right process

- A host can have more than one network application processes
- Received segments must be directed to the correct process
- The OS determines the target process looking at the target port number

Target port number	Process ID
642	X
643	Y

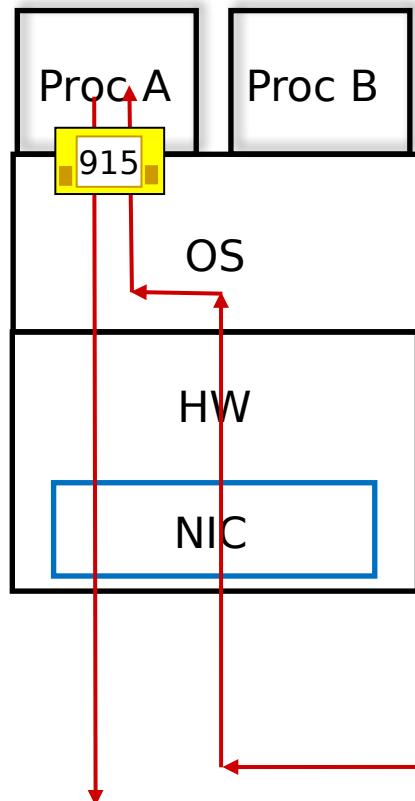




Selecting the right process



IP address: 1.2.3.4



Target port number	Process ID
915	A

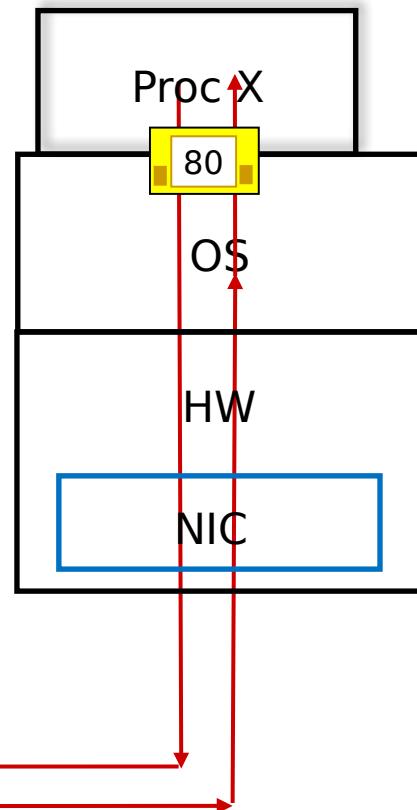
source IP: 3.3.3.3
source port: 80
target IP: 1.2.3.4
target port: 915

resp 1

req 1

source IP: 1.2.3.4
source port: 915
target IP: 3.3.3.3
target port: 80

IP address: 3.3.3.3

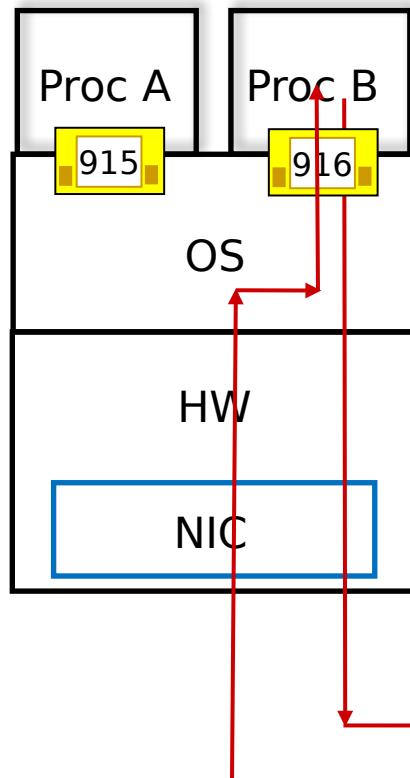




Selecting the right process



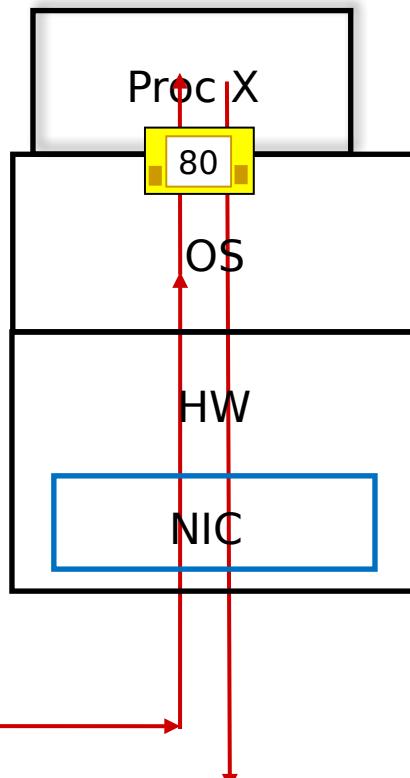
IP address: 1.2.3.4



Target port number	Process ID
915	A
916	B

Target port number	Process ID
80	X

IP address: 3.3.3.3



source IP : 1.2.3.4
source port: 916
target IP : 3.3.3.3
target port: 80

req 2 →

source IP address: 3.3.3.3
source port: 80
target IP address: 1.2.3.4
target port: 916

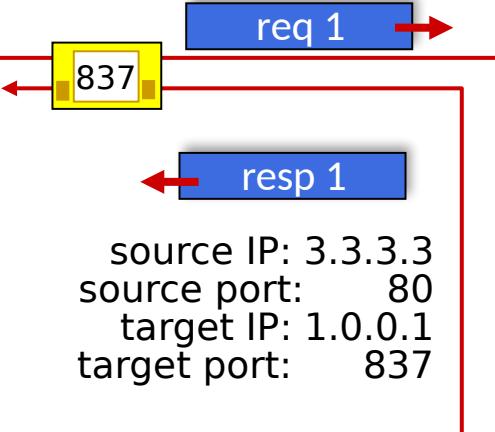
← resp 2

CLIENTS
in a local network (LAN)

IP address:
1.0.0.1



source IP: 1.0.0.1
source port: 837
target IP: 3.3.3.3
target port: 80



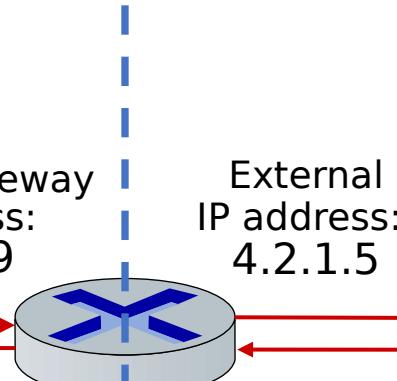
IP address:
1.0.0.2

Network address translation (NAT)

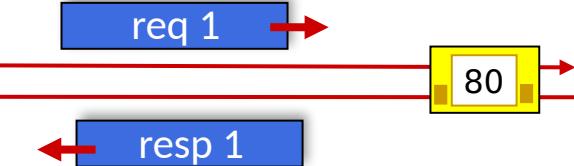
Private network side		Public network side	
Source IP	Source Port	Target IP	NAT Port
1.0.0.1	837	3.3.3.3	267

Default gateway
IP address:
1.0.0.9

“Proxy”, NAT



source IP: 4.2.1.5
source port: 267
target IP: 3.3.3.3
target port: 80



SERVER

Target IP address:
3.3.3.3



source IP: 3.3.3.3
source port: 80
target IP: 4.2.1.5
target port: 267

CLIENTS
in a local network (LAN)

IP address:
1.0.0.1



IP address:
1.0.0.2

source IP: 1.0.0.2
source port: 932
target IP: 3.3.3.3
target port: 80



source IP: 3.3.3.3
source port: 80
target IP: 1.0.0.2
target port: 932

Network address translation (NAT)			
Private network side		Public network side	
Source IP	Source Port	Target IP	NAT Port
1.0.0.1	837	3.3.3.3	267
1.0.0.2	932	3.3.3.3	413

Default gateway
IP address:
1.0.0.9



“Proxy”, NAT

External
IP address:
4.2.1.5

source IP: 4.2.1.5
source port: 413
target IP: 3.3.3.3
target port: 80



source IP: 3.3.3.3
source port: 80
target IP: 4.2.1.5
target port: 413

SERVER

Target

IP address:
3.3.3.3

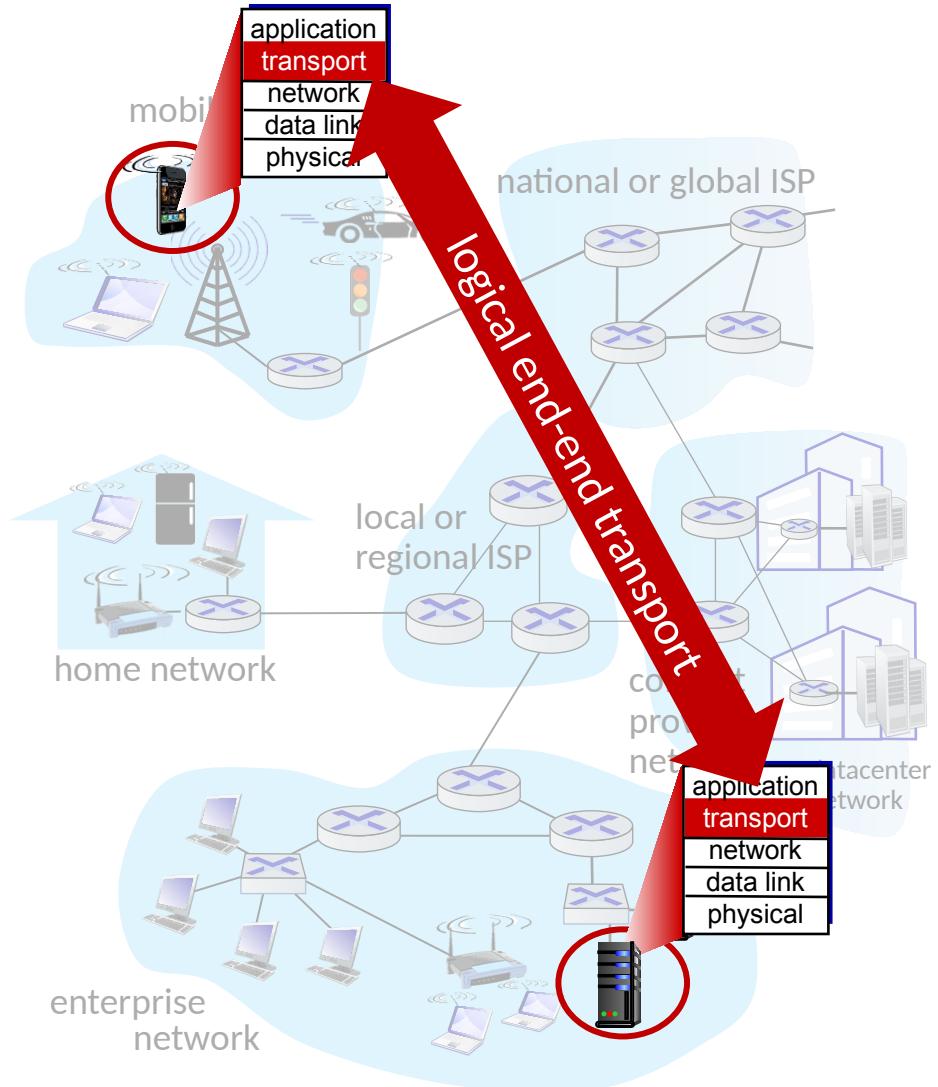


Chapter 3: roadmap

1. Transport-layer services
2. Port numbers
- 3. Connectionless transport: UDP**
 - UDP socket programming
4. Principles of reliable data transfer
5. Connection-oriented transport: TCP
- ~~6. Principles of congestion control~~
7. TCP congestion control
8. QUIC: Quick UDP Internet Connections

UDP: User Datagram Protocol

- Port numbers
- Integrity check
- Connectionless data transmission
- No data segmentation
- Not reliable data transfer
 - “Best-effort” IP
- services **not** available:
 - delay guarantees
 - bandwidth guarantees



UDP: User Datagram Protocol

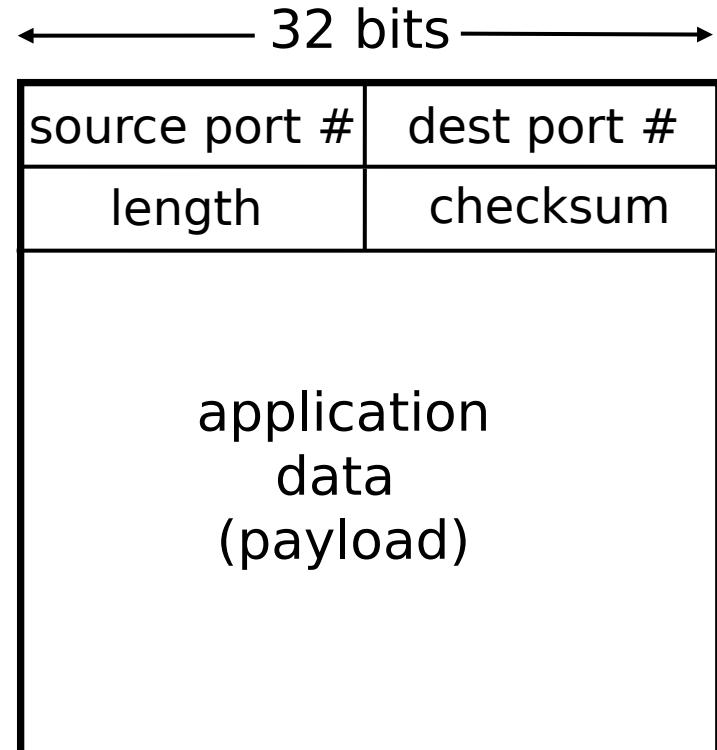
Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
 - UDP can blast away as fast as desired!
 - can function in the face of congestion

UDP: User Datagram Protocol

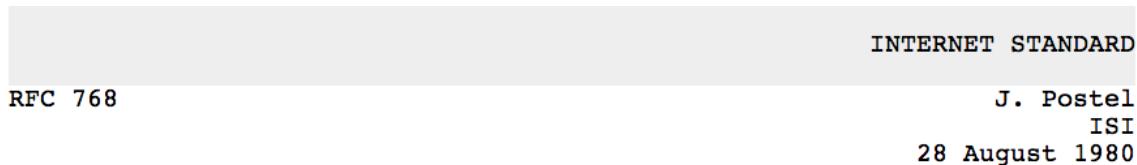
- UDP use:
 - streaming multimedia apps (loss tolerant, rate sensitive)
 - DNS
 - SNMP
 - HTTP/3
- if reliable transfer needed over UDP (e.g., HTTP/3):
 - add needed reliability at application layer
 - add congestion control at application layer

UDP segment header



UDP segment format

UDP: User Datagram Protocol [RFC 768]



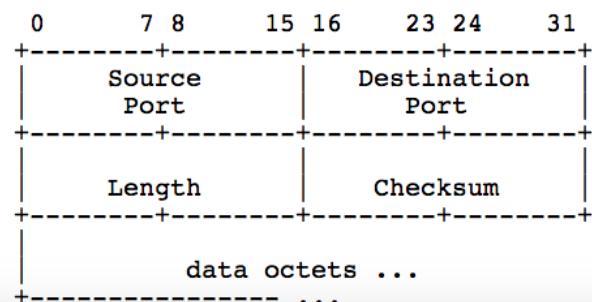
User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

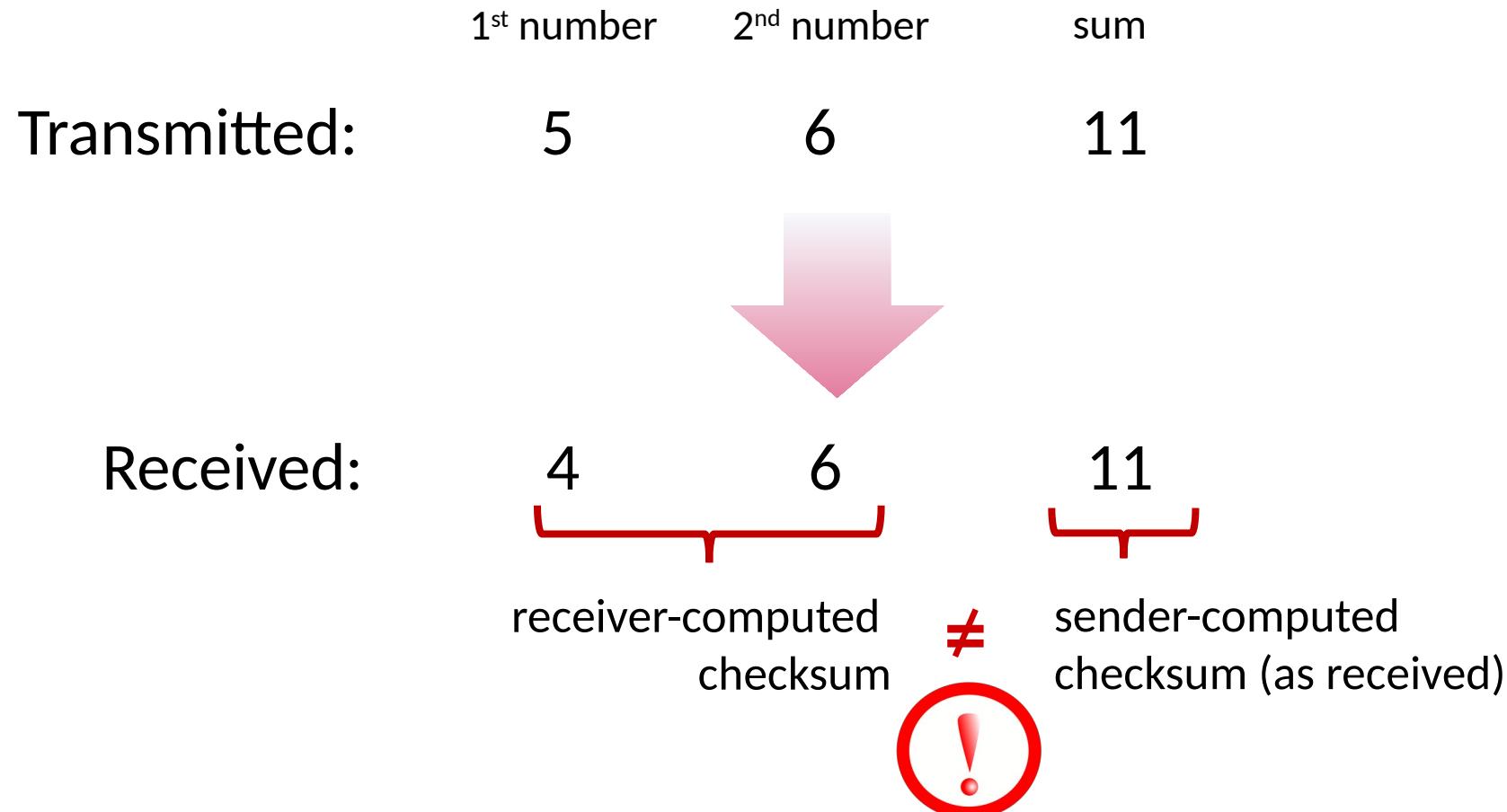
This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



UDP checksum

Goal: detect errors (i.e., flipped bits) in transmitted segment



UDP checksum

Goal: detect errors (i.e., flipped bits) in transmitted segment

sender:

- treat contents of UDP segment (including UDP header fields and a pseudo header from IP header) as sequence of 16-bit integers
- **checksum:** addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
 - Not equal - error detected
 - Equal - no error detected. *But maybe errors, nonetheless? More later*

Internet checksum: an example

example: add two 16-bit integers

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

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

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

Internet checksum: weak protection!

example: add two 16-bit integers

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

Even though numbers have changed (bit flips), **no** change in checksum!

IP loopback address

- The special network address, 127.0.0.1, is defined as a **local loopback address**
- Hosts use local loopback addresses to send messages to themselves
- In Windows, the name **localhost** is an alias for 127.0.0.1

```
C:\Users\sigurde>tracert 127.0.0.1  
Tracing route to UIA5CG4081L51 [127.0.0.1]  
over a maximum of 30 hops:  
    1    <1 ms    <1 ms    <1 ms  UIA5CG4081L51  
  
Trace complete.  
  
C:\Users\sigurde>tracert localhost  
Tracing route to UIA5CG4081L51 [::1]  
over a maximum of 30 hops:  
    1    <1 ms    <1 ms    <1 ms  UIA5CG4081L51  
  
Trace complete.
```

Sending and receiving UDP packets using ncat

- Two command prompt windows

SENDER

```
C:\> ncat -u 127.0.0.1 8888
```

RECEIVER

```
C:\> ncat -l -u 8888
```

Active connections - TCP ports

- A TCP or UDP process *listens* on a local port

```
C:\> netstat -a
```

Active Connections

Proto	Local Address	Foreign Address	State
TCP	10.0.0.9:54747	20.238.236.234:https	ESTABLISHED
TCP	10.0.0.9:54909	104.18.35.23:https	ESTABLISHED
TCP	127.0.0.1:49201	UIA5CG4081L51:0	LISTENING

- netstat lists the names of the processes that are listening using **-b**
 - Requires elevated command prompt

Active connections - UDP-ports

- A network process *listens* on a local port

```
C:\> netstat -a p udp
```

Active Connections

Proto	Local Address	Foreign Address	State
UDP	0.0.0.0:5355	* : *	
UDP	0.0.0.0:8888	* : *	
UDP	0.0.0.0:50080	* : *	
UDP	0.0.0.0:52068	127.0.0.1:52067	

Scanning for open UDP ports

```
C:\Users\sigurde>nmap -sU -p 8888 localhost
Starting Nmap 7.95 ( https://nmap.org ) at 2025-01-31 10:56 W. Europe S
e
Nmap scan report for localhost (127.0.0.1)
Host is up.
Other addresses for localhost (not scanned): ::1

PORT      STATE            SERVICE
8888/udp  open|filtered  ddi-udp-1

Nmap done: 1 IP address (1 host up) scanned in 2.34 seconds
```

UDP socket programming

UDP socket programming

Sigurd Eskeland

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**

Request-response

client process: process that initiates communication using a request

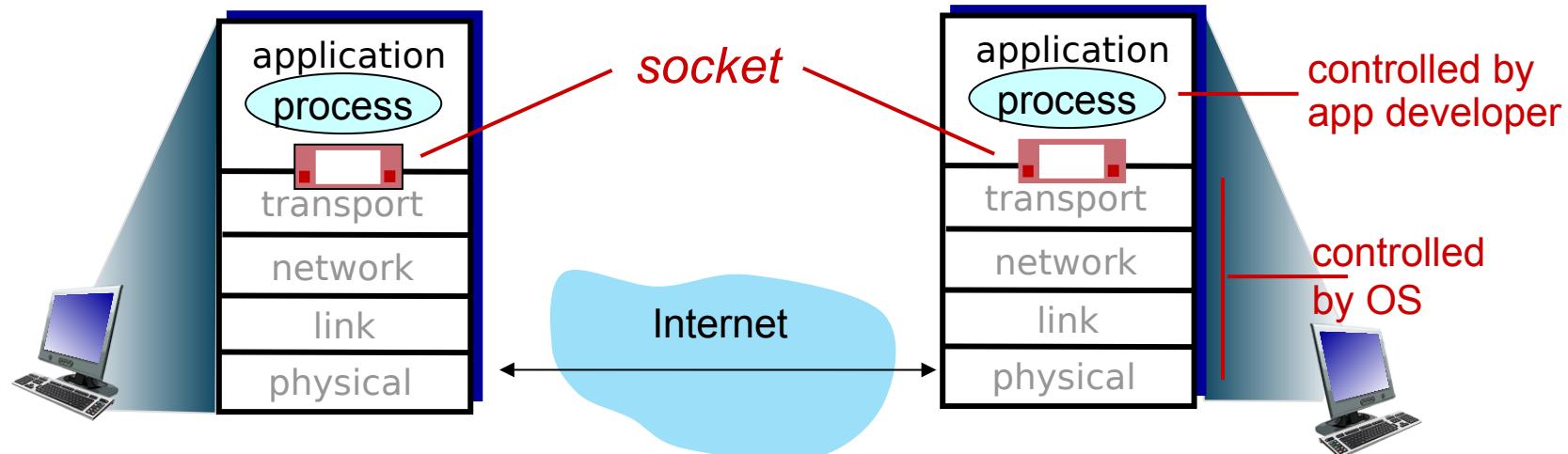
server process:

- process that waits to be contacted
- sends a response to the client

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

Sockets

1. Process sends/receives messages to/from its **socket**
 - Two sockets involved: one on each side
2. A socket is analogous to a **door** between network application process and the transport protocol
 - The sending process shoves the message out the “door”
 - **Not the same as port number**; sockets **use** port numbers



What is a socket?

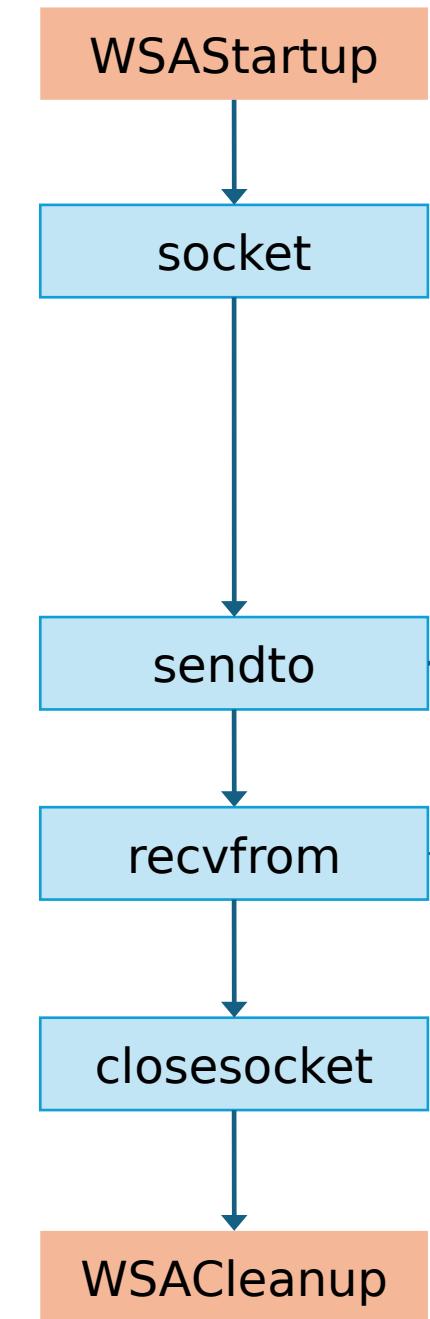
1. A network socket is a *data structure* that serves as an **endpoint for sending and receiving data** across a network
2. Sockets are defined by an application programming interface (API)
3. A socket is accessed by a “handle” ☰ **socket file descriptor**
4. When creating a socket instance, the transport layer protocol (TCP or UDP) is specified

What is a socket?

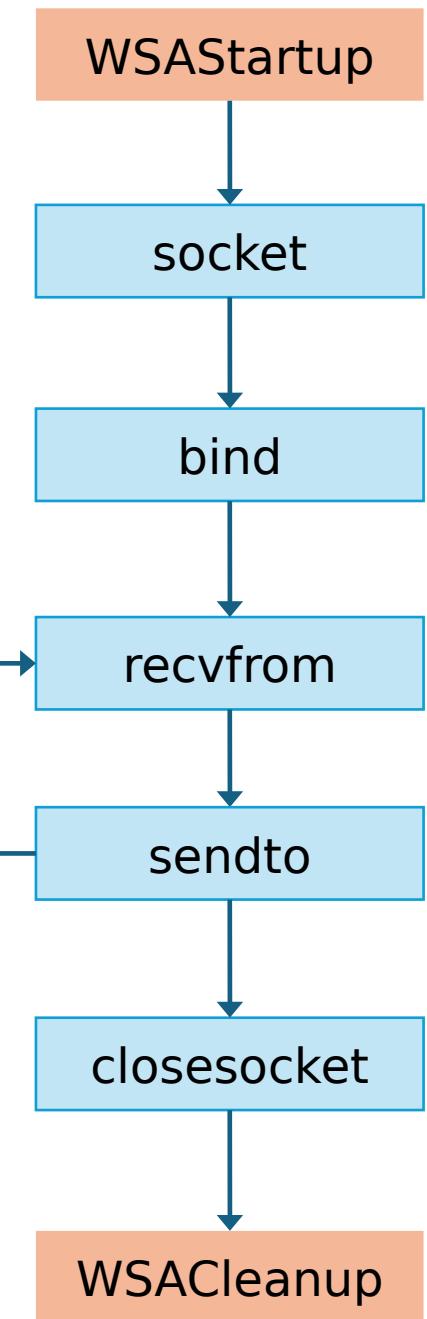
5. If a program shall **receive packets**, the program must **bind** the socket to a specific **port number**
 - So that the OS can deliver incoming data packets to the right application process
6. A socket that has received a packet, then **knows** the sender's IP address and port number
 - Needed for sending a response to the right **sending process**

UDP sockets

Client



Server



UDP server

```
#include <winsock2.h>
```

	WSADATA wsa; SOCKET socketServer;
Holds address of a received message	struct sockaddr_in addressServer; struct sockaddr_in addressReceived;
Start use of Winsock DLL (Ws2_32.dll)	WSAStartup(MAKEWORD(2,2), &wsa);
Create UDP-socket (IPv4)	socketServer = socket(AF_INET, SOCK_DGRAM, 0);
Any address	// Initialize address structure
Address is	addressServer.sin_addr.s_addr = INADDR_ANY;
Specify port number	addressServer.sin_family = AF_INET; addressServer.sin_port = htons(8888); // LSB -> MSB
assign addressServer to the socket	bind(socketServer, (struct sockaddr*)& addressServer, sizeof(addressServer));
get size of address structure	int nFromlen = sizeof(addressReceived);
Wait for a message: capture string and address	recvfrom(socketServer, sReceivedString, STR_SIZE, 0, (struct sockaddr*)& addressReceived, &

UDP server

Any address → `WSAStartup(MAKEWORD(2,2), &wsa);`

Address is → `socketServer = socket(AF_INET , SOCK_DGRAM , 0);`

Specify port number → `// Initialize address structure
addressServer.sin_addr.s_addr = INADDR_ANY;`

IPv4 → `addressServer.sin_family = AF_INET;`

IPv6 → `addressServer.sin_port = htons(8888); // LSB -> MSB`

assign `addressServer` to the → `bind(socketServer, (struct sockaddr*)& addressServer, sizeof(addressServer));`

socket → `get size of address structure`

structure → `int nFromlen = sizeof(addressReceived);`

Wait for a message: → `recvfrom(socketServer, sReceivedString, STR_SIZE, 0,`

capture string and → `(struct sockaddr*)& addressReceived, &`

address → `nFromlen);`

`sendto(...);`

`closesocket(socketServer);`

`WSACleanup();`

Endianness - htons()

Big-endian system

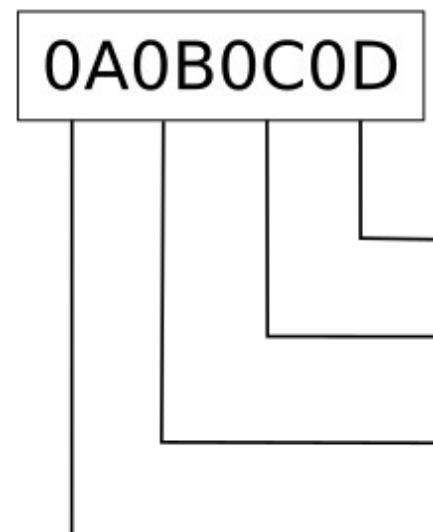
- stores the most significant byte (**MSB**) of a word at the smallest memory address

Little-endian system

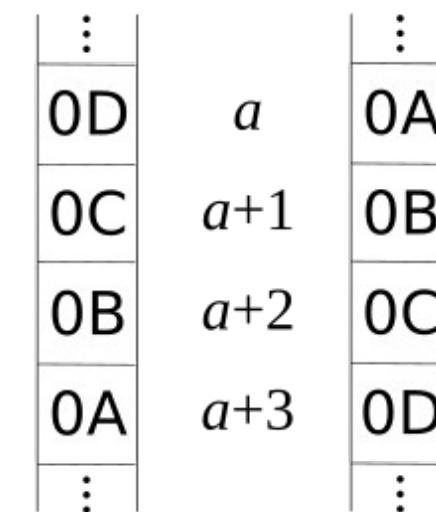
- stores the least-significant byte (**LSB**) at the smallest address

Little-endian

32-bit integer

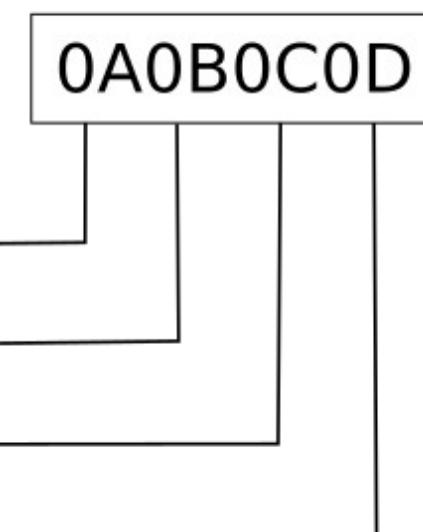


Memory



Big-endian

32-bit integer

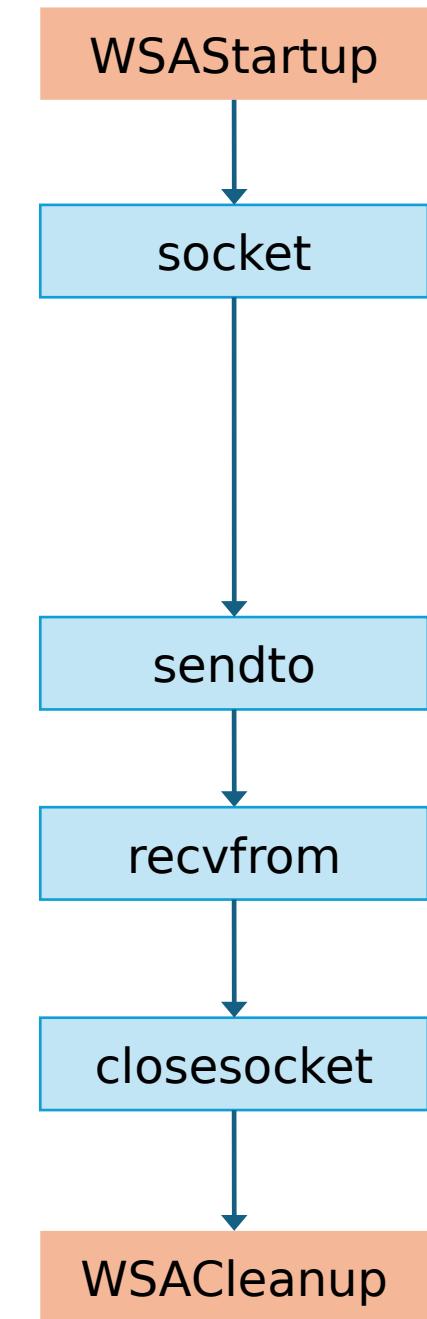


Endianness

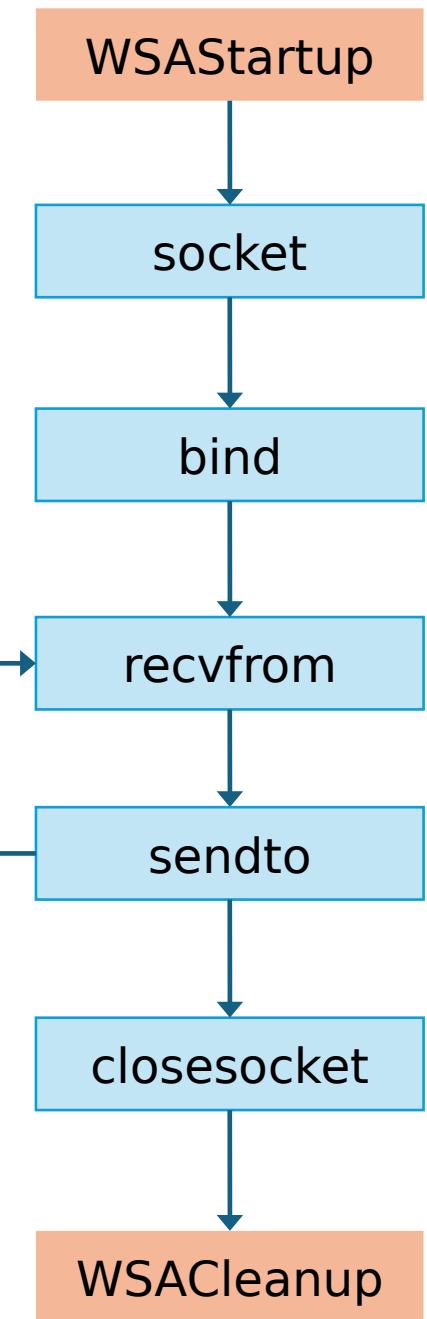
A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest. A little-endian system, in contrast, stores the least-significant byte at the smallest address

UDP sockets

Client



Server



UDP client

```
#include <winsock2.h>
#include <ws2tcpip.h> // InetPton

WSADATA wsa;
SOCKET socketClient;

Holds target address → struct sockaddr_in addressServer;
char sTarget[] = "127.0.0.1"; // loopback address

Start use of Winsock DLL → WSAStartup(MAKEWORD(2,2), &wsa);

Create UDP-socket → socketClient = socket(AF_INET, SOCK_DGRAM, 0 );

Receiver's IP address - → // Initialize address structure wrt. the target
convert text-address to binary → InetPton(AF_INET, _TEXT(sTarget), &
Address forms → addressServer.sin_addr);
→ addressServer.sin_family = AF_INET;
→ addressServer.sin_port = htons(8888); // LSB -> MSB

Specify target's port →
number → sendto(socketClient, sMessage, sizeof(message), 0,
Send message to target → (struct sockaddr*)& addressServer,
address using socketClient → sizeof(addressServer) );
```

UDP client

Receiver's IP address -
convert text-address to binary

Address forms

Specify target's IP4
number

Send message to address
using socketClient

Receive message from
client

Close the socket

```
struct sockaddr_in addressServer;
```

```
WSAStartup(MAKEWORD(2,2), &wsa);
```

```
socketClient = socket(AF_INET , SOCK_DGRAM , 0 );
```

```
// Initialize address structure wrt. the target  
InetPton(AF_INET, _TEXT("127.0.0.1"), &  
addressServer.sin_addr);  
addressServer.sin_family = AF_INET;  
addressServer.sin_port = htons(8888); // LSB -> MSB
```

```
sendto(socketClient, sSendText, strlen(sSendText), 0,  
(struct sockaddr*)& addressServer,  
sizeof(addressServer) );
```

```
recvfrom( ... );
```

```
closesocket(socketClient);
```

```
WSACleanup();
```

Linking in CLion

- In Windows, the WinSock library is implemented by **WS2_32.dll**
- In CLion, this DLL is specified in CMakeLists.txt by the line

```
target_link_libraries(${CMAKE_PROJECT_NAME} Ws2_32)
```

Programming exercise

1. Client reads a line of characters (data) from its keyboard and sends data to server
2. Server receives the data and converts characters to uppercase
3. Server sends modified data to client
4. Client receives modified data and displays line on its screen

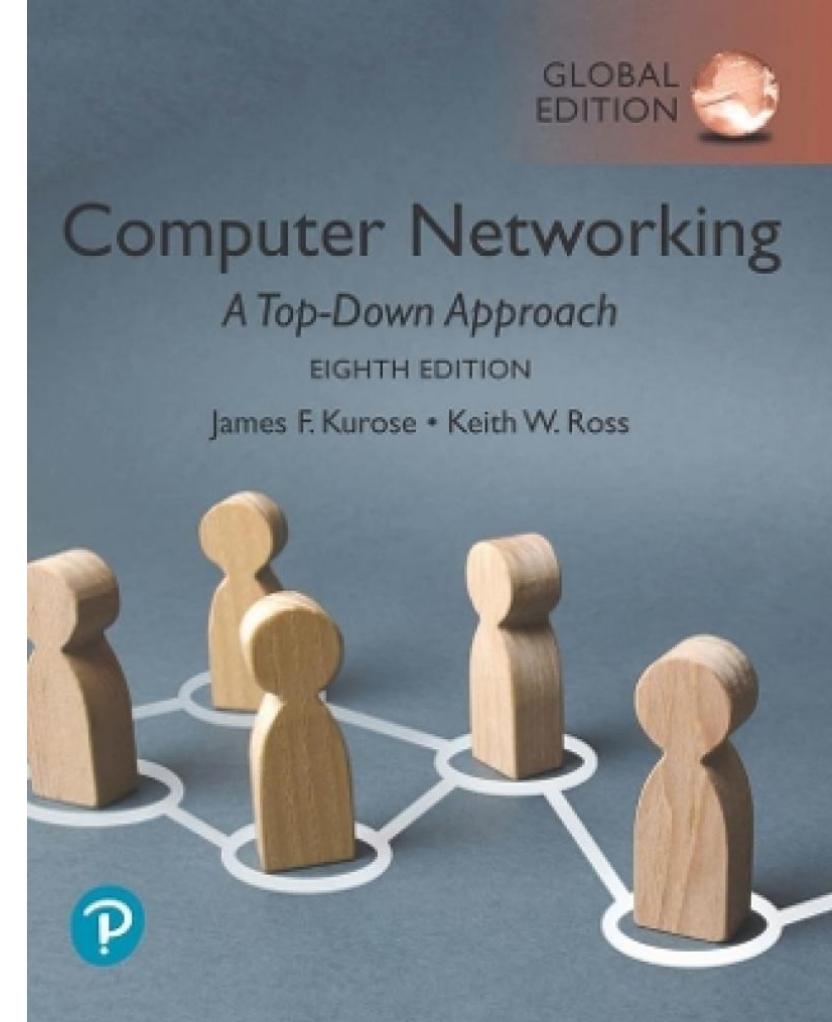
IP loopback address

- The special network address, 127.0.0.1, is defined as a **local loopback address**
- Hosts use local loopback addresses to send messages to themselves
- In Windows, the name **localhost** is an alias for 127.0.0.1

```
C:\Users\sigurde>tracert 127.0.0.1  
Tracing route to UIA5CG4081L51 [127.0.0.1]  
over a maximum of 30 hops:  
    1    <1 ms    <1 ms    <1 ms  UIA5CG4081L51  
  
Trace complete.  
  
C:\Users\sigurde>tracert localhost  
Tracing route to UIA5CG4081L51 [::1]  
over a maximum of 30 hops:  
    1    <1 ms    <1 ms    <1 ms  UIA5CG4081L51  
  
Trace complete.
```

Chapter 3: roadmap

- 3.1 Transport-layer services
- 3.2 Port numbers
- 3.3 Connectionless transport: UDP
 - UDP socket programming
- 3.4 Principles of reliable data transfer**
- 3.5 Connection-oriented transport: TCP
- ~~3.6 Principles of congestion control~~
- 3.7 TCP congestion control
- 3.8 QUIC: Quick UDP Internet Connections

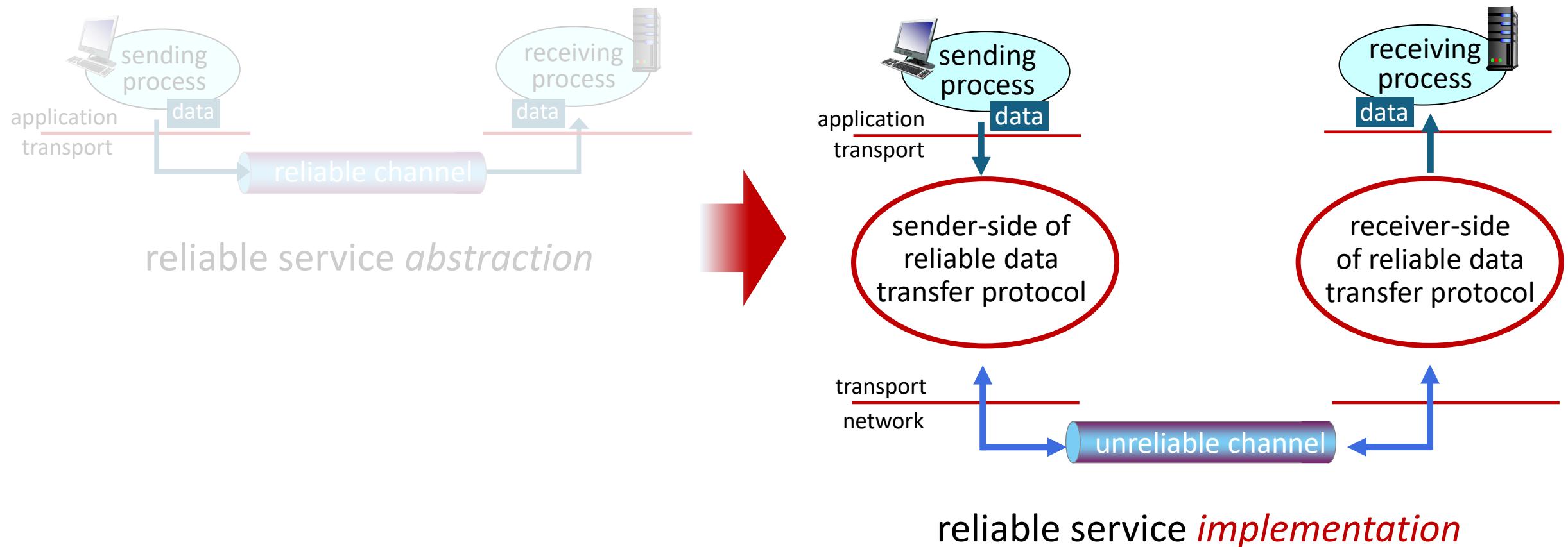


Principles of reliable data transfer



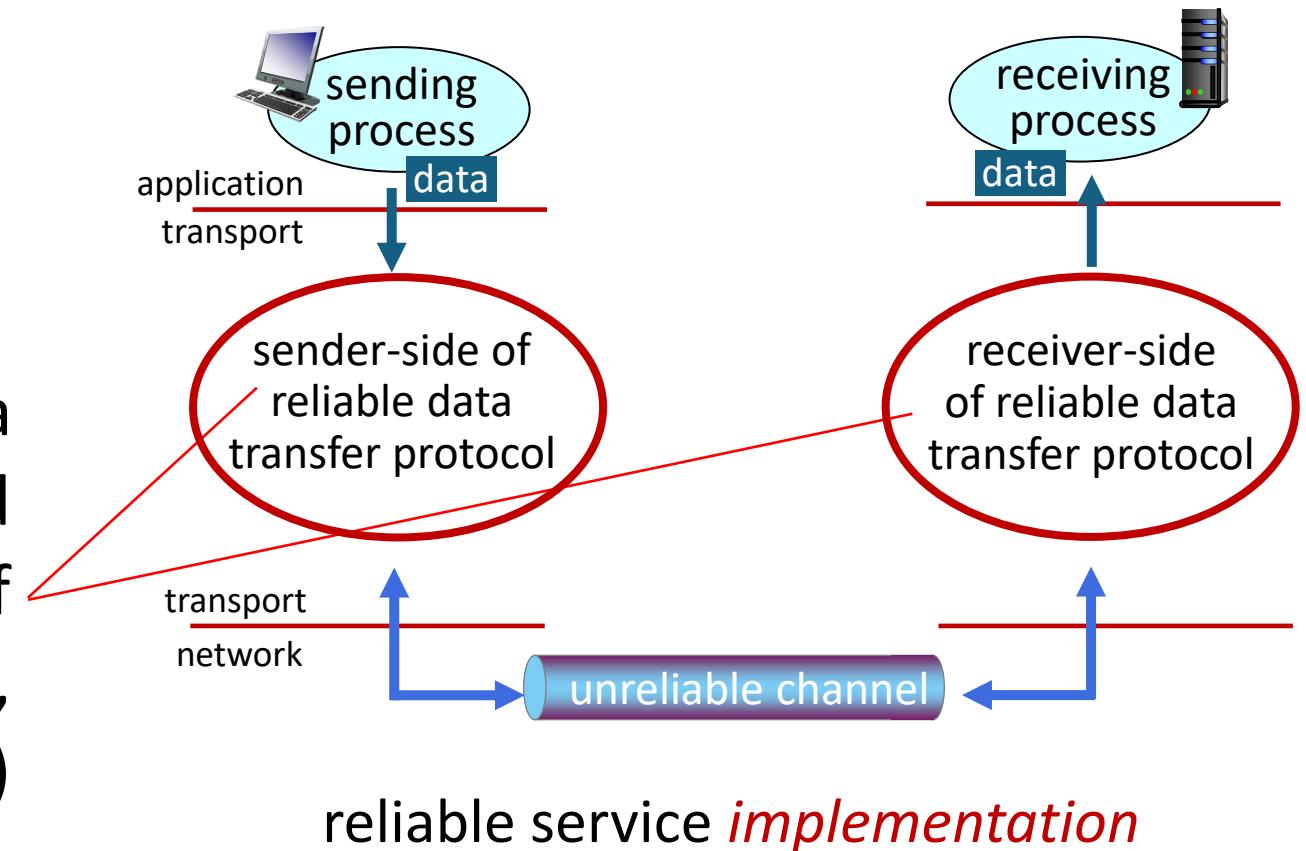
reliable service *abstraction*

Principles of reliable data transfer



Principles of reliable data transfer

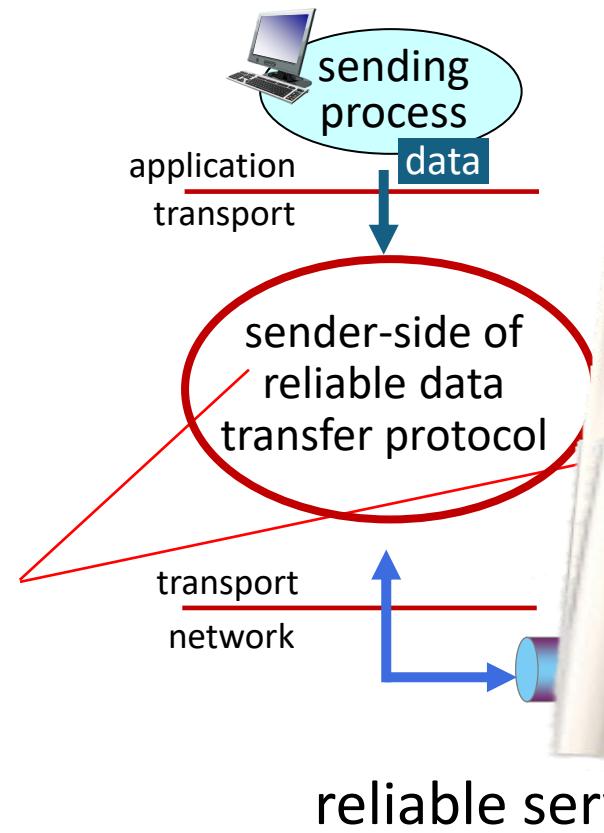
Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)



Principles of reliable data transfer

Sender, receiver do *not* know the “state” of each other, e.g., was a message received?

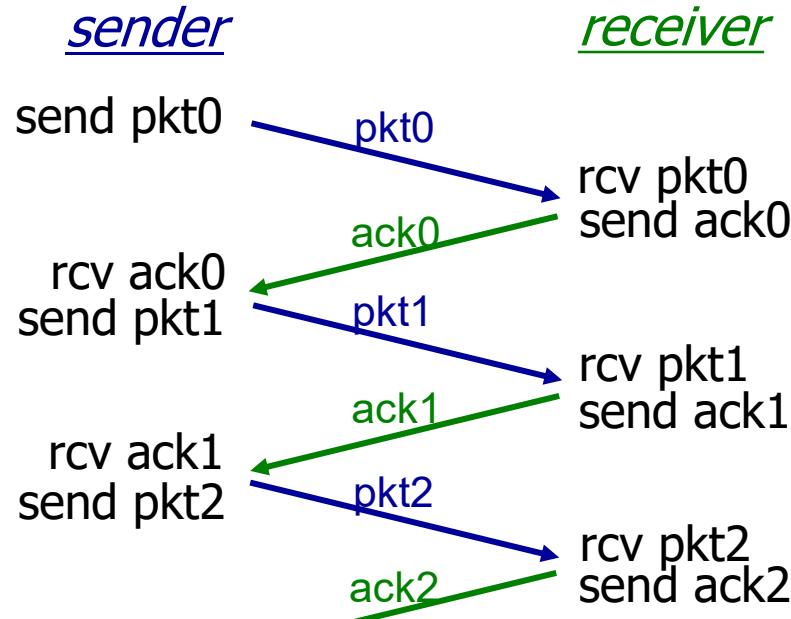
- unless communicated via a message



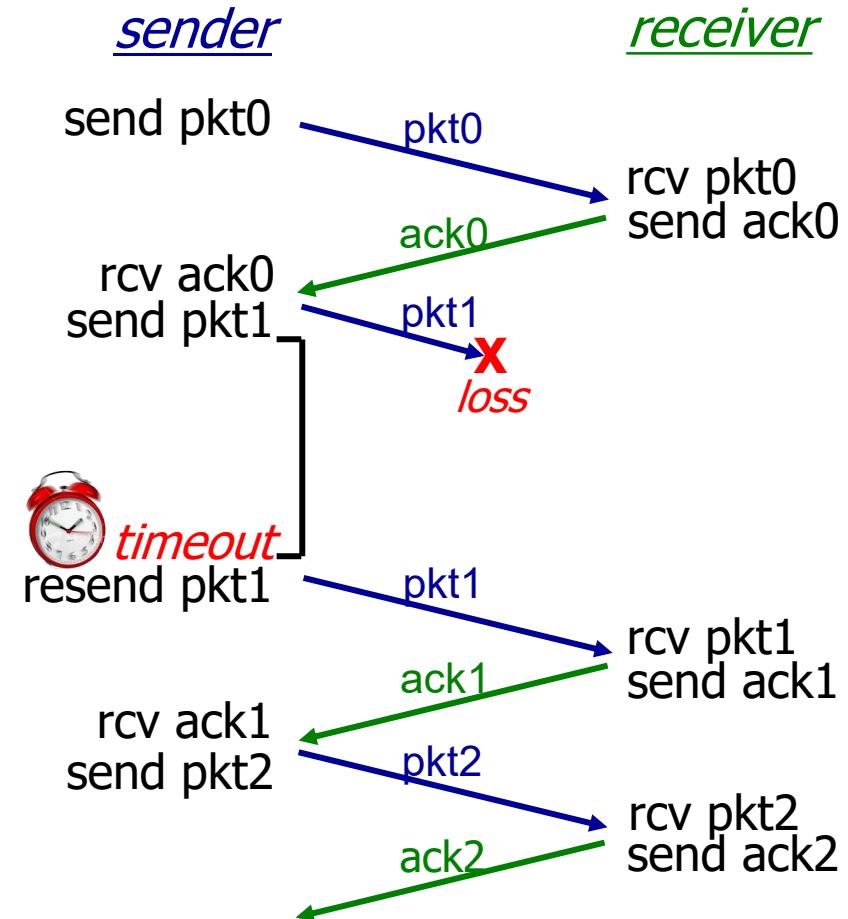
Mechanisms for reliable data transfer (rdt)

1. Acknowledgement: ACK
2. Timer (sender-side)
3. Sequence numbers

Principles of reliable data transfer (rdt3.0)

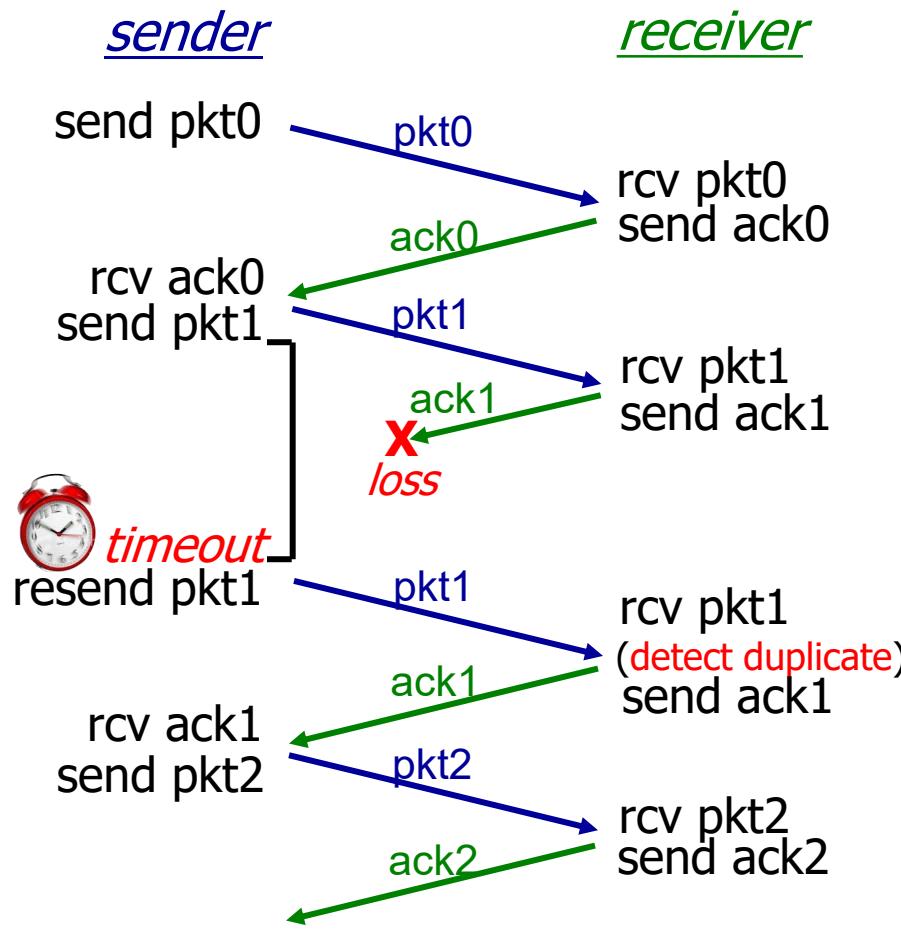


(a) no loss

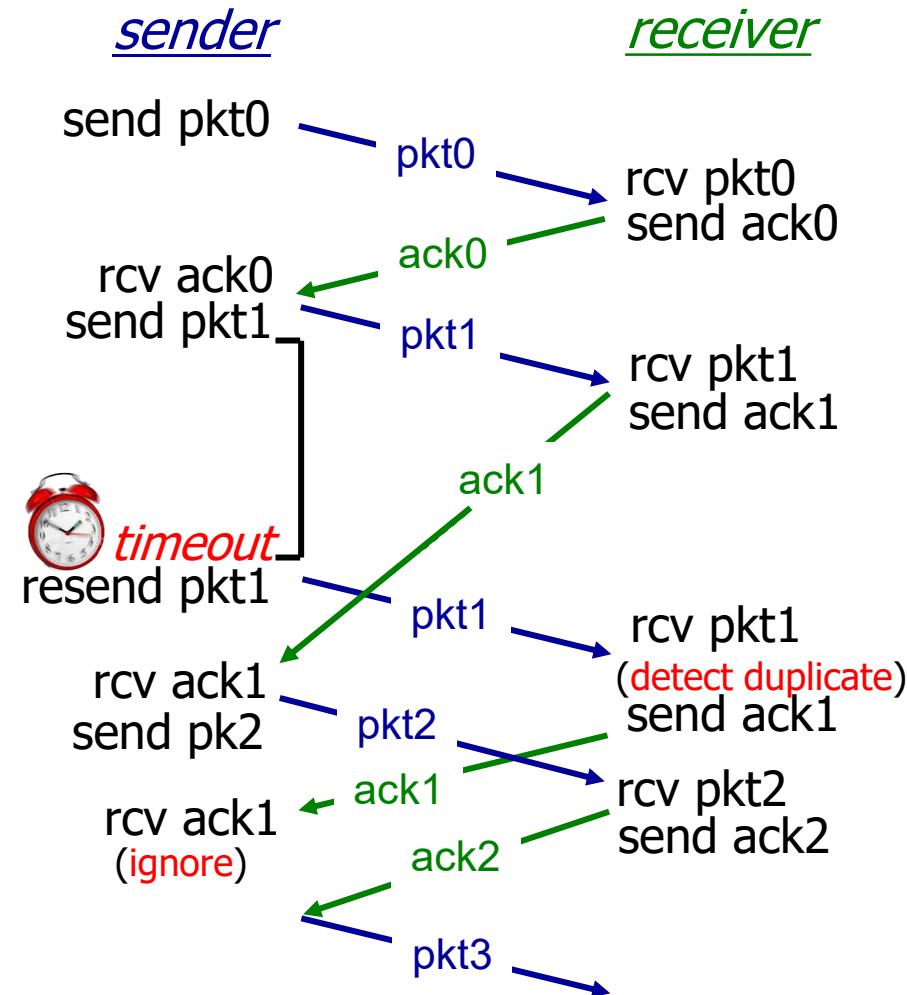


(b) packet loss

Principles of reliable data transfer (rdt3.0)



(c) ACK loss



(d) premature timeout/ delayed ACK

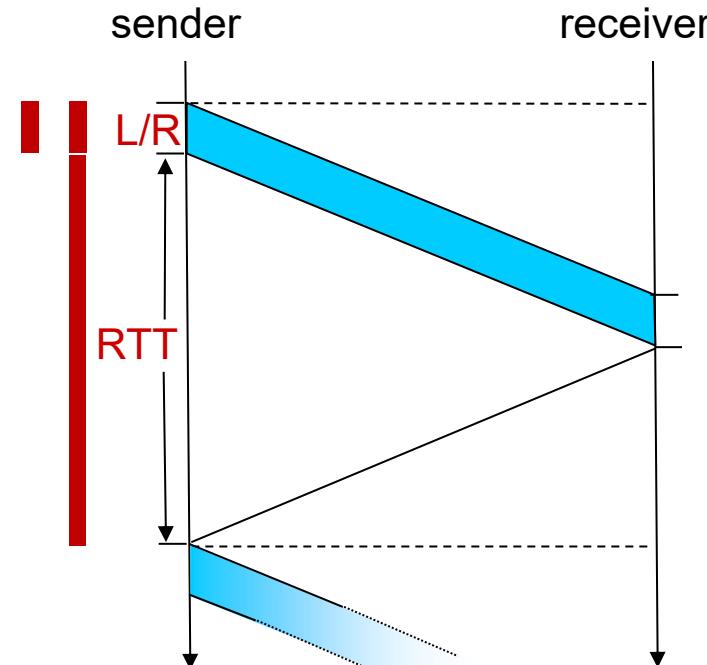
Performance - stop-and-wait

- U_{sender} : *utilization* – fraction of time sender busy sending
- example: 1 Gbps link, 15 ms *propagation delay*, 1000 bytes packet
 - time to transmit packet into channel:

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

- example: 1 Gbps link, 15 ms propagation delay, 1000 bytes packet

$$\begin{aligned}
 \text{Utilization } U_{\text{sender}} &= \frac{L / R}{RTT + L / R} \\
 &= \frac{0.008}{30.008} \\
 &\approx 0.00027 \\
 &\approx 0.027\%
 \end{aligned}$$

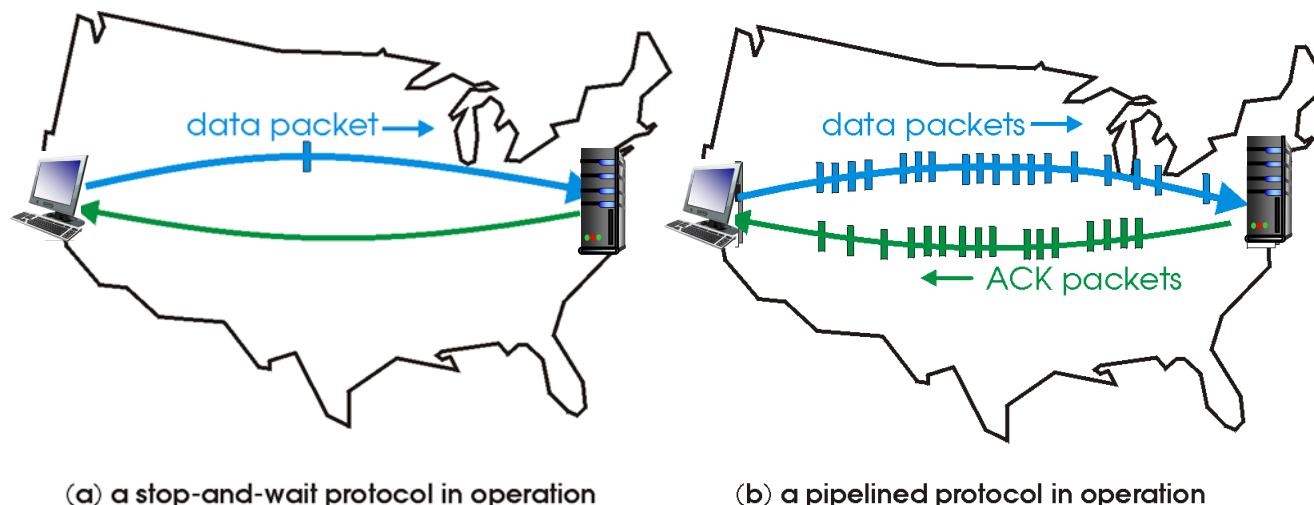


- rdt 3.0 protocol performance stinks!
- Protocol limits performance of underlying infrastructure (channel)

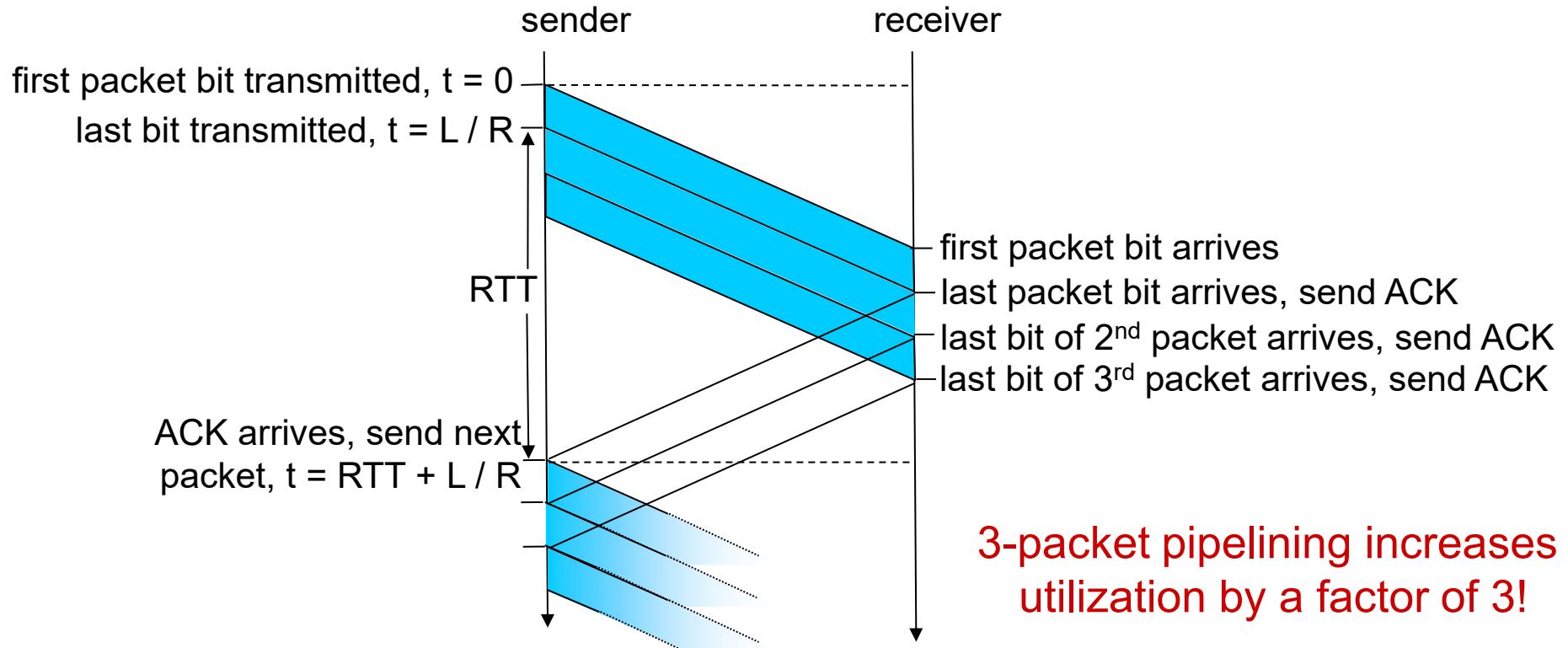
rdt3.0: pipelined protocols operation

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

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



Pipelining: increased utilization



$$U_{\text{sender}} = \frac{3 \cdot L/R}{RTT + L/R} = \frac{0.0024}{30.008} \approx 0.00081 \approx 0.081\%$$

Pipelined protocols: overview

Go-back-N:

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

Selective Repeat:

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

Chapter 3: roadmap

3.1 Transport-layer services

3.2 Port numbers

3.3 Connectionless transport: UDP

 UDP socket programming

3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP

~~3.6 Principles of congestion control~~

3.7 TCP congestion control

3.8 QUIC: Quick UDP Internet Connections

3.5 Connection-oriented transport: TCP

1. TCP overview
2. Segment structure
3. The TCP connection
4. Opening and closing TCP connections
5. Reliable data transfer
6. TCP round trip time, timeout
7. Flow control

3.5.1 TCP: overview RFCs: 793, 1122, 2018, 5681, 7323

- **Connection-orientation**
 - handshaking establishes a TCP connection
 - **point-to-point**: one sender, one receiver
 - **full duplex** : bi-directional data flow in same connection
- **Data segmentation**
 - sender: breaks application messages into **segments**, passes to network layer
 - receiver: **reassembles** segments into messages, passes to application layer
- **Reliable, in-order *byte steam***
- **Pipelining**
 - Cumulative ACKs
- **Flow control**
 - sender will not overwhelm receiver
- **Congestion control**

Comparison of UDP and TCP

UDP:

- Port numbers
- Integrity check
- Connection**less** data transmission
- **No** data segmentation
- **Not** reliable data transfer

TCP:

- Port numbers
- Integrity check
- Connection-**oriented** data transmission
- Data segmentation
- Reliable data transfer
 - flow control and congestion control

services **not** available:

delay guarantees and bandwidth guarantees

3.5 Connection-oriented transport: TCP

1. TCP overview
2. Segment structure
3. The TCP connection
4. Opening and closing TCP connections
5. Reliable data transfer
6. TCP round trip time, timeout
7. Flow control

3.5.2 TCP segment structure

ACK: seq # of next expected byte; A bit: this is an ACK

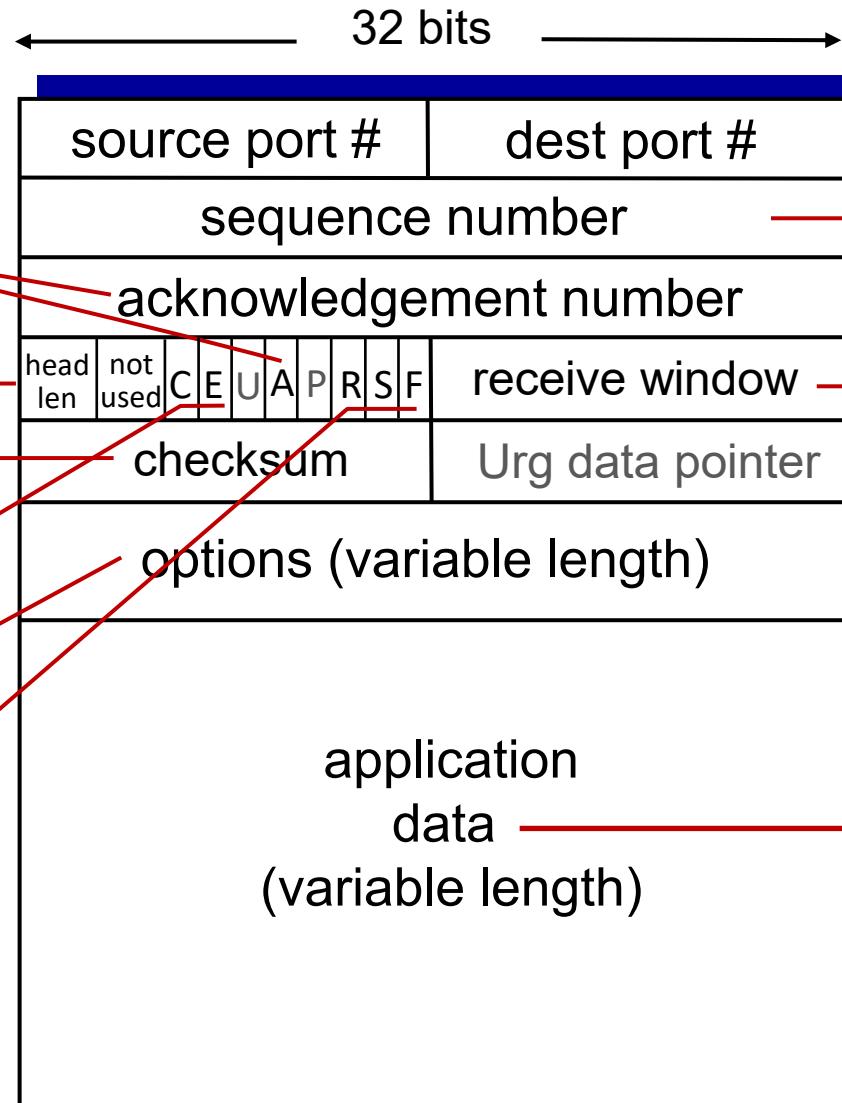
length (of TCP header)

Internet checksum

C, E: congestion notification

TCP options

RST, SYN, FIN: connection management



segment seq #: counting bytes of data into bytestream (not segments!)

flow control: # bytes receiver willing to accept

data sent by application into TCP socket

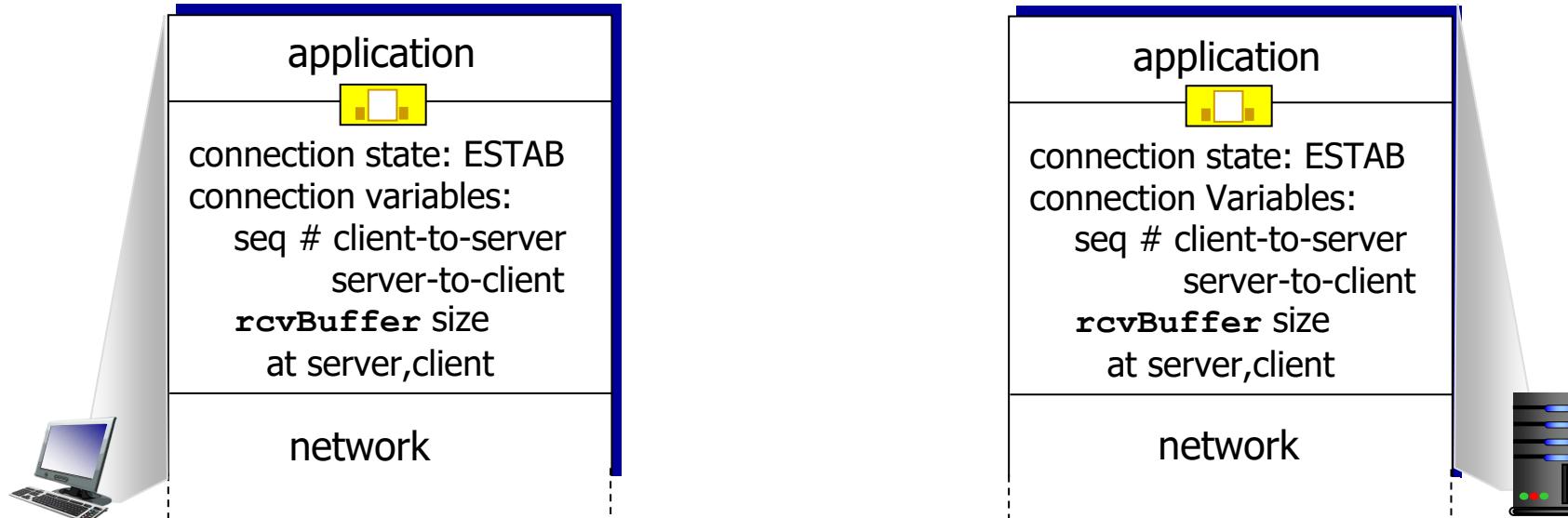
3.5 Connection-oriented transport: TCP

1. TCP overview
2. Segment structure
- 3. The TCP connection**
4. Opening and closing TCP connections
5. Reliable data transfer
6. TCP round trip time, timeout
7. Flow control

3.5.3 TCP connection management

before exchanging data, sender/receiver “handshake”:

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters (e.g., starting seq #s)



```
clientSocket = socket(AF_INET, SOCK_STREAM)  
  
connect(clientSocket, hostName, portNumber)
```

```
serverSocket = socket(AF_INET, SOCK_STREAM)  
bind(serverSocket, serverPort)  
serverSocket.listen(1)  
connectionSocket = accept(serverSocket)
```

TCP connection management

- TCP socket identified by **4-tuple**:
 - source IP address
 - source port number
 - destination IP address
 - destination port number
- receiver uses *all four values* (*4-tuple*) to direct segment to appropriate socket
- server may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple
 - each socket associated with a different connecting client

Comparison of connection management

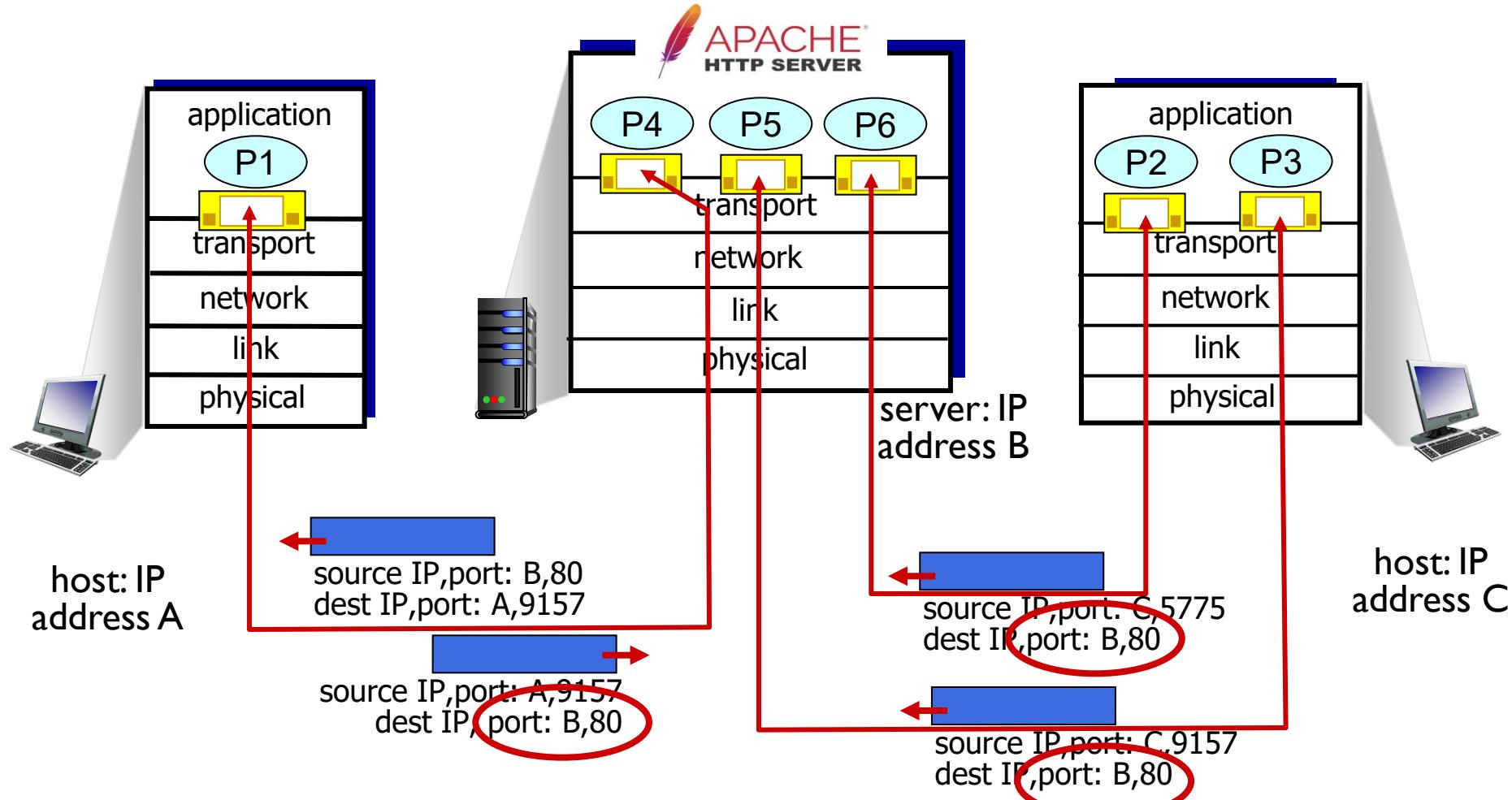
UDP: target socket identified using 2-tuple:

- destination IP and destination port number

TCP: target socket identified using 4-tuple:

- source and destination IP addresses
- source and destination port numbers

TCP connections and sockets: example

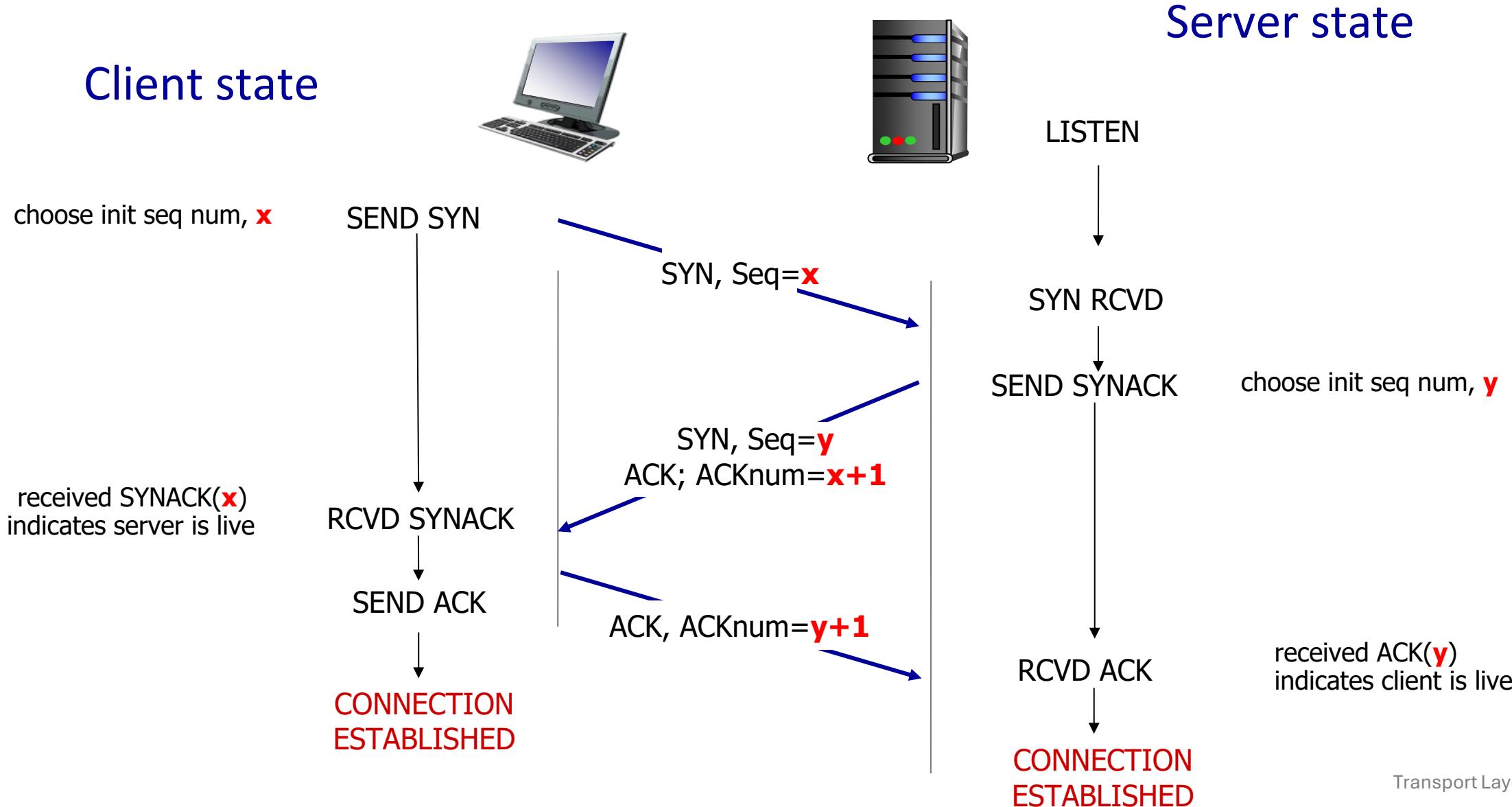


Three segments, all destined to IP address: B,
destination port: 80 are demultiplexed to *different* sockets

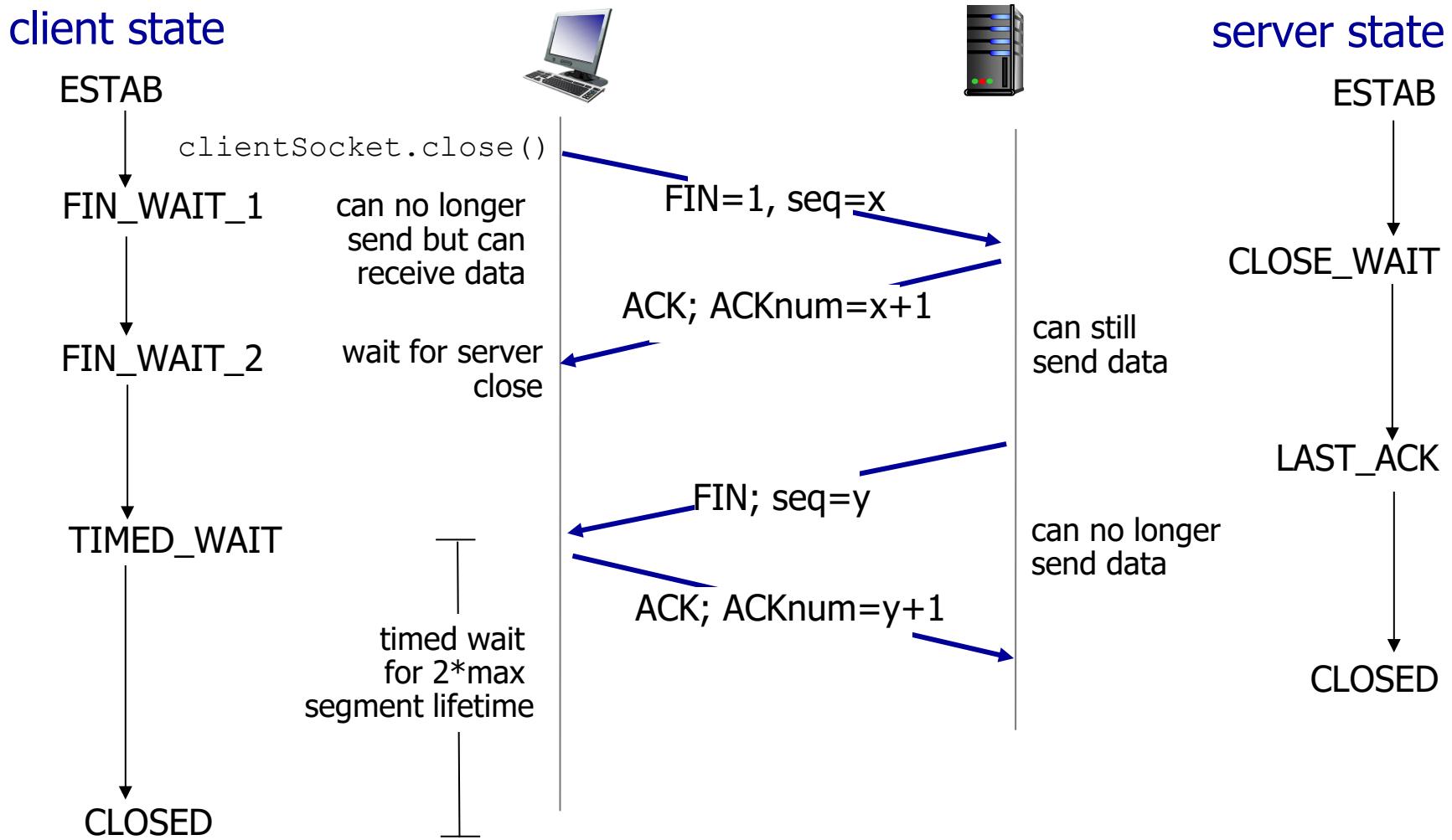
3.5 Connection-oriented transport: TCP

1. TCP overview
2. Segment structure
3. The TCP connection
- 4. Opening and closing TCP connections**
5. Reliable data transfer
6. TCP round trip time, timeout
7. Flow control

New TCP connection: 3-way handshake



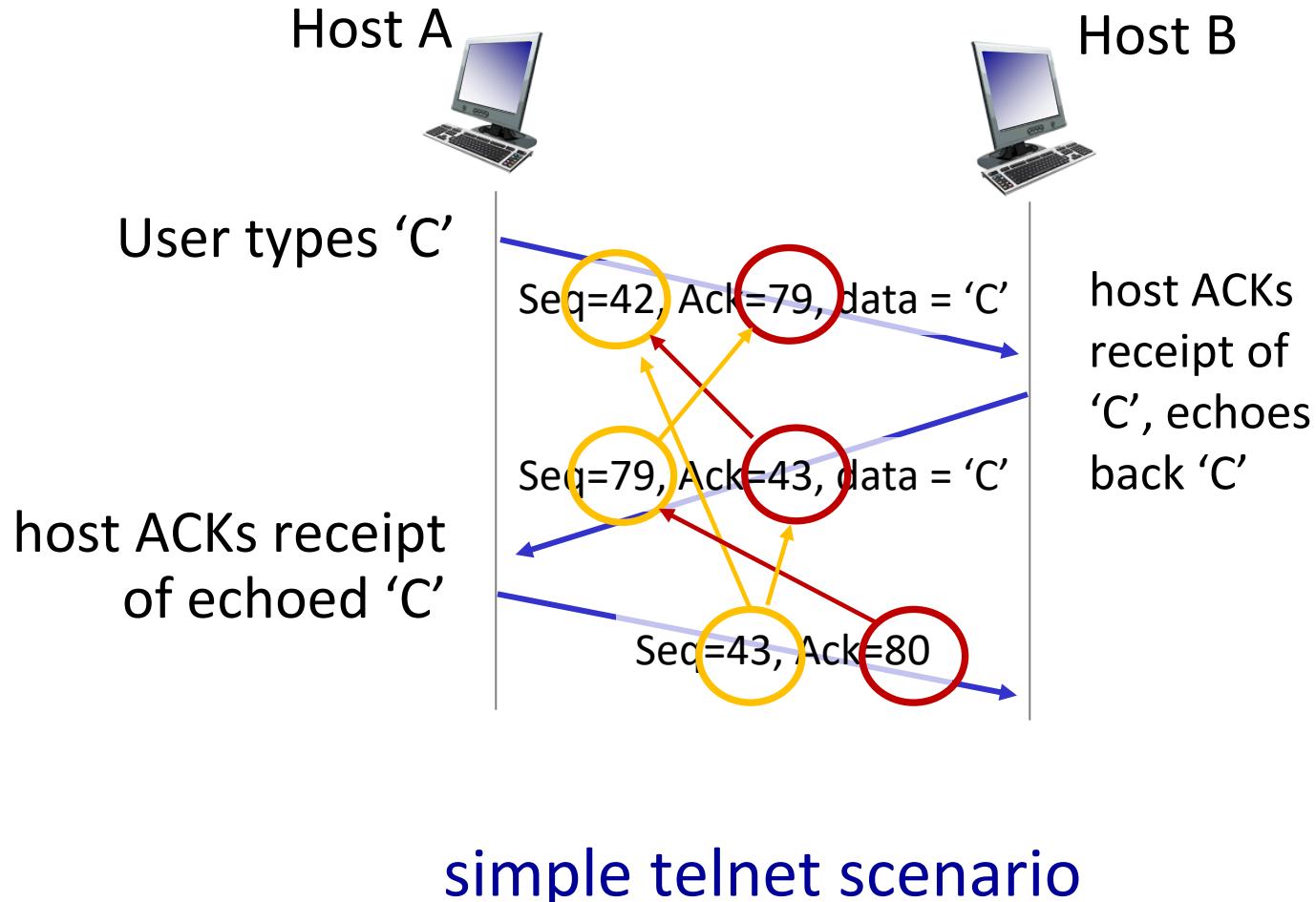
Closing a TCP connection



3.5 Connection-oriented transport: TCP

1. TCP overview
2. Segment structure
3. The TCP connection
4. Opening and closing TCP connections
- 5. Reliable data transfer**
6. TCP round trip time, timeout
7. Flow control

TCP sequence numbers, ACKs



SEQ-number:

- ACK-number of received segment
- incremented by payload size of previously sent segment

ACK-number:

- seq. number + payload size of received segment
- the expected seq. number of next received segment

Why random sequence numbers?

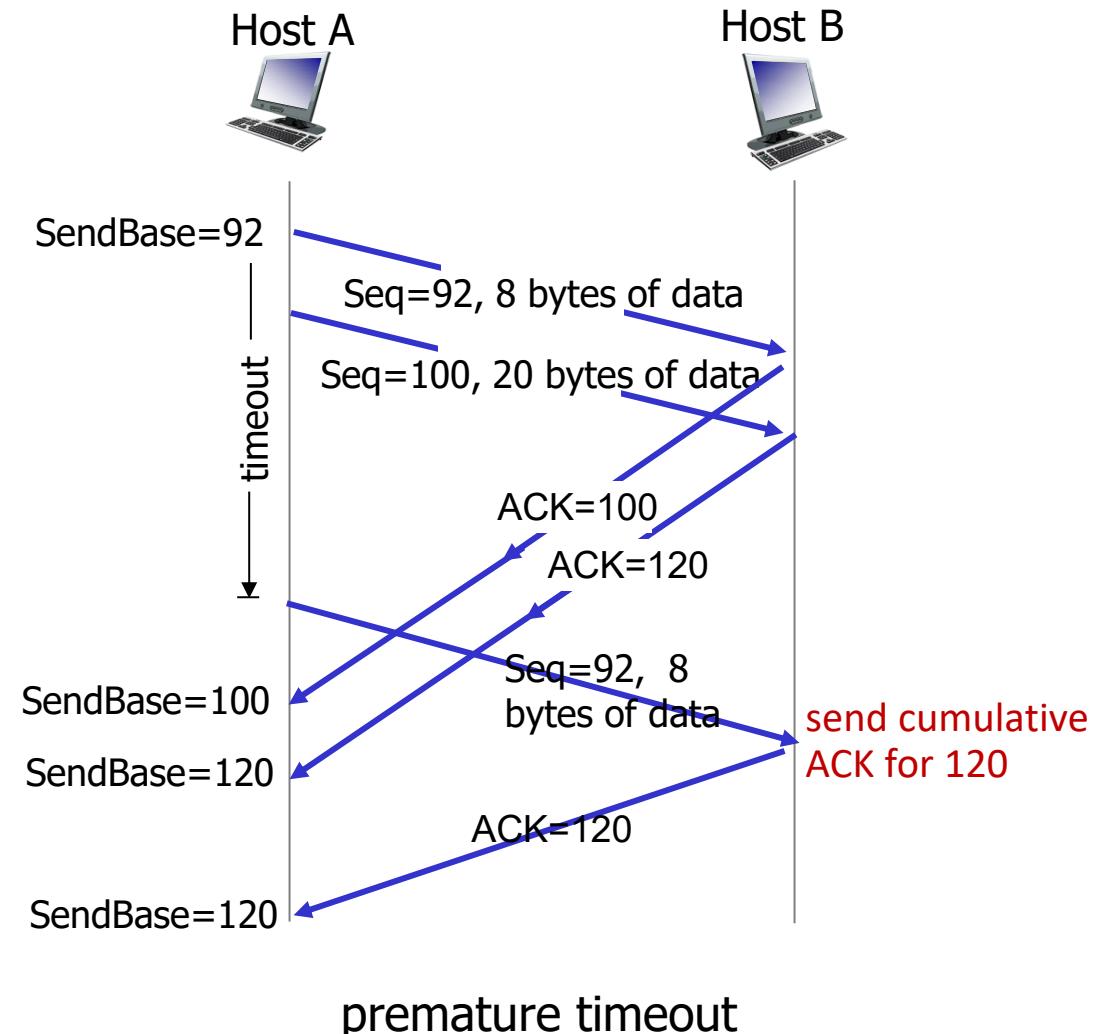
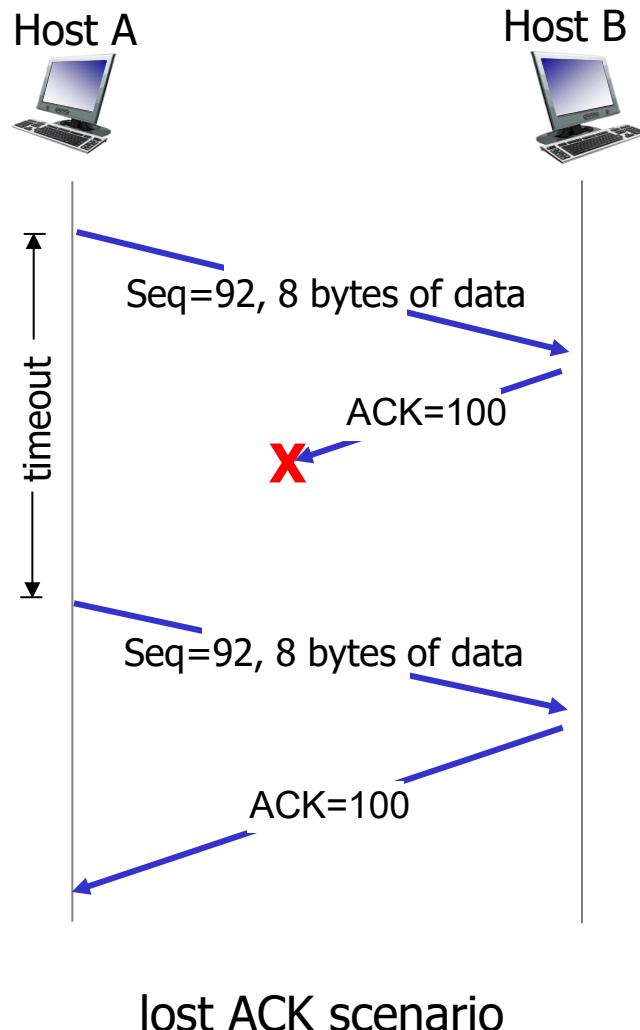
Multi-session interference.

- If all sessions started their sequence numbers at 1, then it would be much easier to end up in situations where you mix up packets from various sessions between two hosts (though there are other measures in place to avoid this, like randomizing the source port).

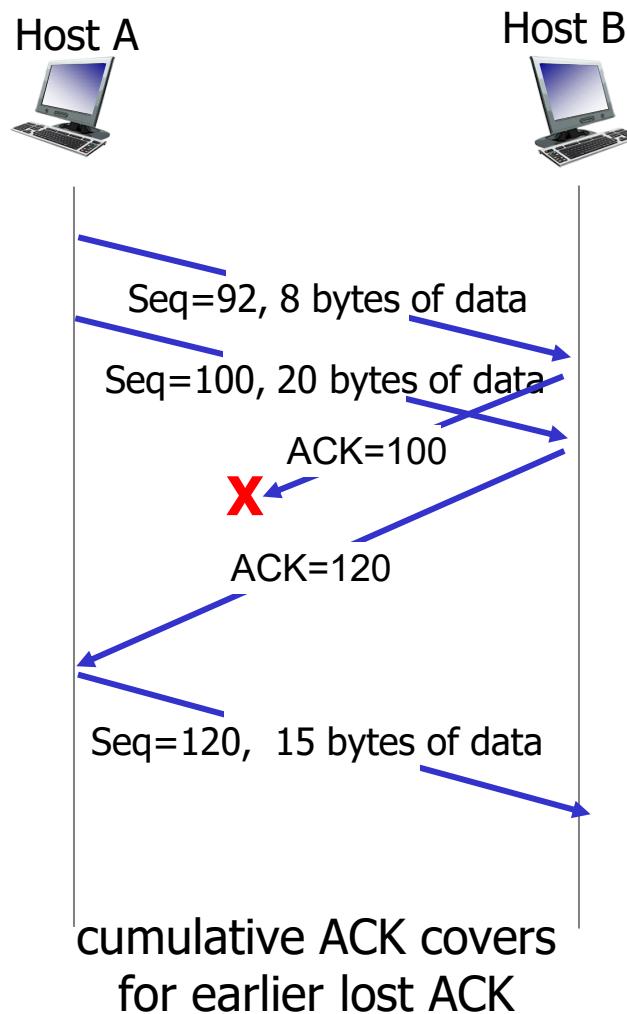
Security

- TCP sequence number randomization is a technique that aims to make TCP sequence numbers unpredictable and hard to guess by attackers. This can improve network security by reducing the chances of TCP spoofing attacks

TCP: retransmission scenarios



TCP: retransmission scenarios



TCP fast retransmit

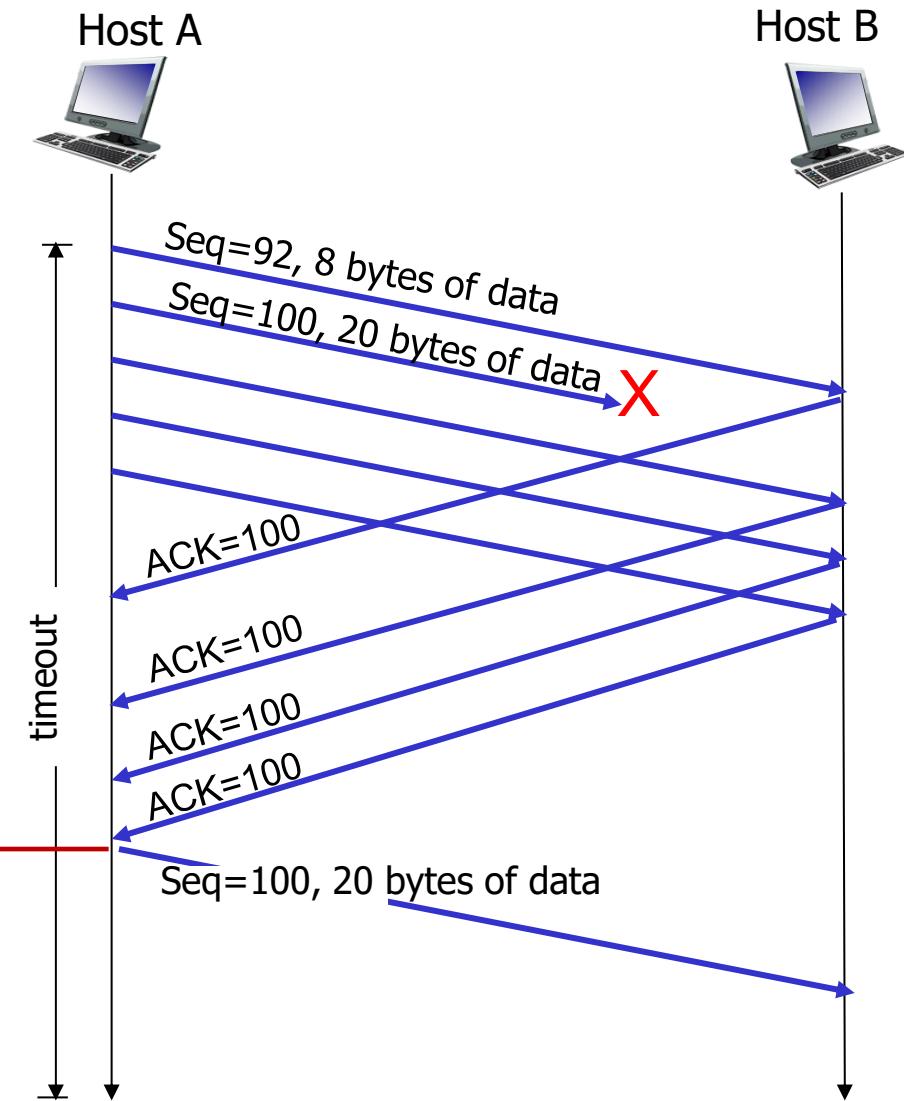
TCP fast retransmit

if sender receives 3 additional ACK's for same data ("triple duplicate ACK's"), resend unACK'ed segment with smallest seq #

- likely that unACK'ed segment lost, so don't wait for timeout



Receipt of three duplicate ACK's indicates 3 segments received after a missing segment – lost segment is likely. So, retransmit!



TCP Sender (simplified)

event: data received from application

- create segment with seq number
- seq number is byte-stream number of first data byte in segment
- start timer if not already running
 - for oldest unACKed segment
 - expiration interval:
TimeOutInterval

event: timeout

- retransmit segment that caused timeout
- restart timer

event: ACK received

- update what is known to be ACKed segments
- start timer if there are still unACKed segments

TCP Receiver: ACK generation [RFC 5681]

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

3.5 Connection-oriented transport: TCP

1. TCP overview
2. Segment structure
3. The TCP connection
4. Opening and closing TCP connections
5. Reliable data transfer
6. TCP round trip time, timeout
7. Flow control

3.5.6 TCP round trip time, timeout

Q: how to set TCP timeout value?

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

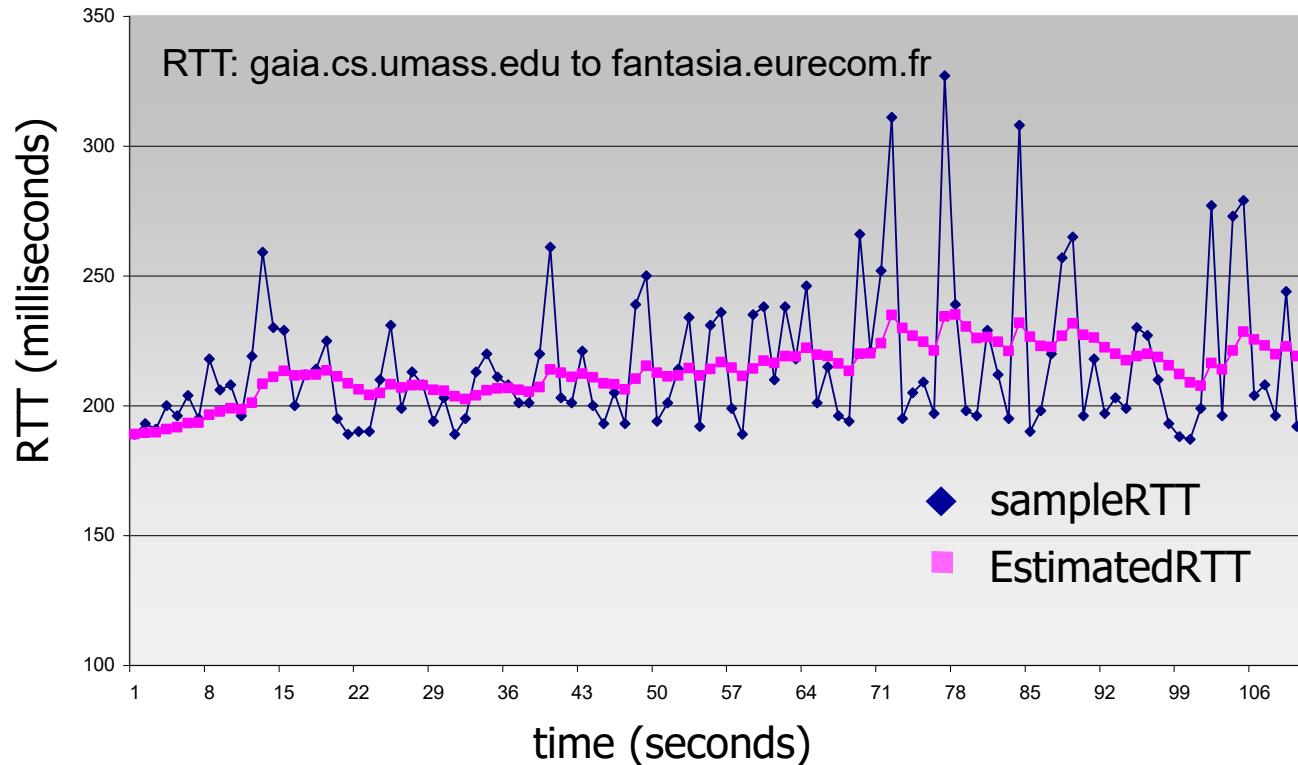
Q: how to estimate RTT?

- **SampleRTT**: measured time from segment transmission until ACK receipt
- **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 (EWMA)
- 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**: want a larger safety margin

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



estimated RTT

“safety margin”

- DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

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

(typically, $\beta = 0.25$)

deviation, difference

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

3.5 Connection-oriented transport: TCP

1. TCP overview
2. Segment structure
3. The TCP connection
4. Opening and closing TCP connections
5. Reliable data transfer
6. TCP round trip time, timeout
7. Flow control

3.5.7 TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from receiver buffer?

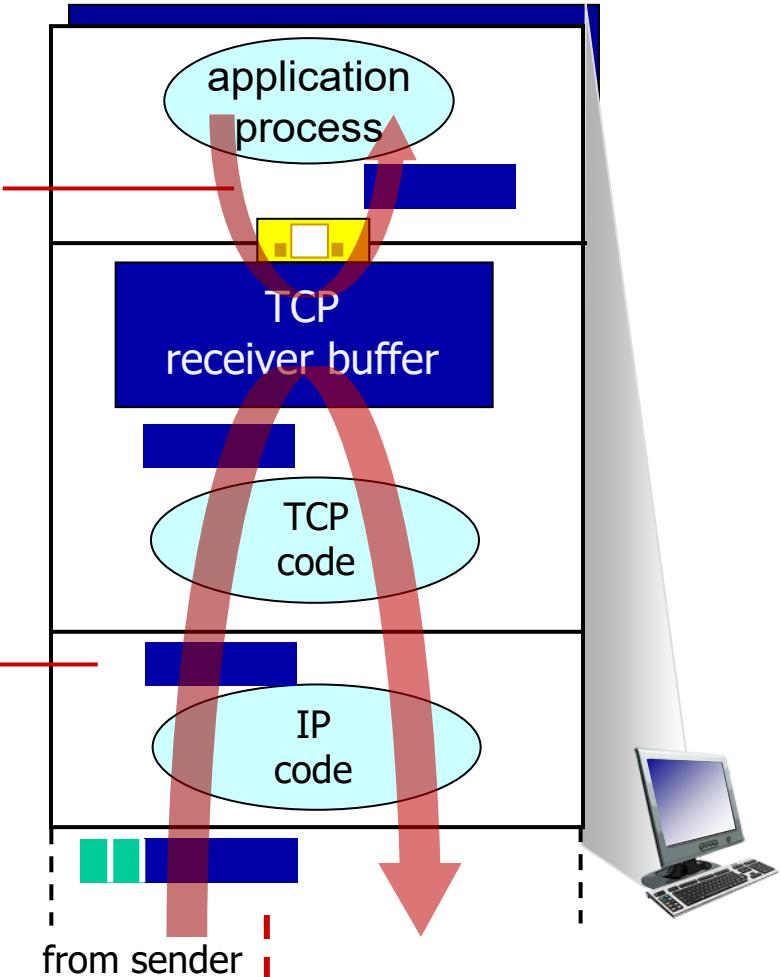
flow control

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

Application fetching data from TCP buffer

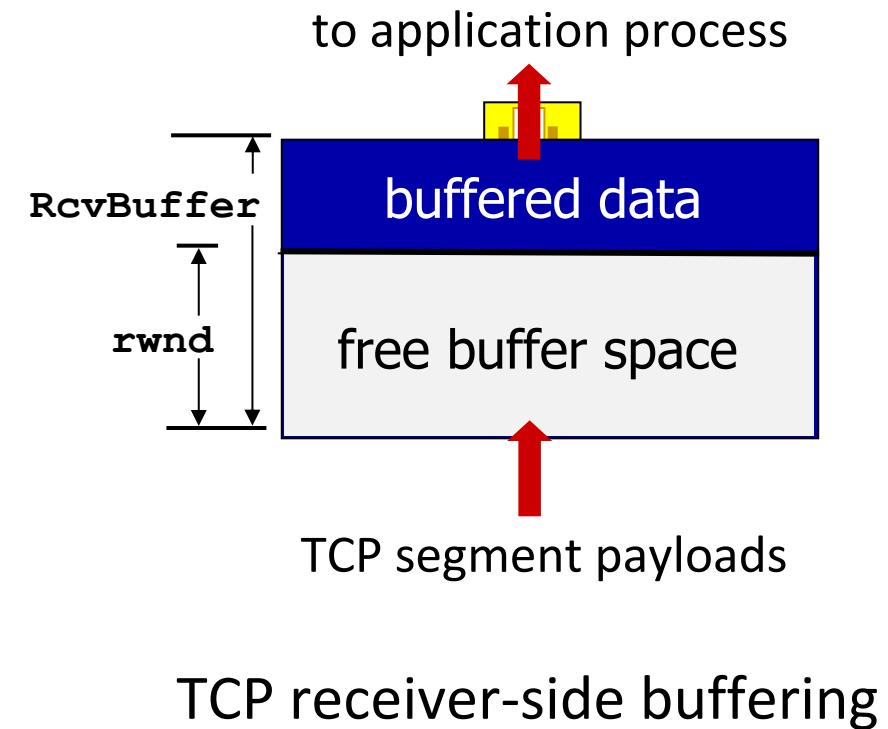
Network layer delivering IP datagram payload into TCP socket buffers

receiver protocol stack



TCP flow control

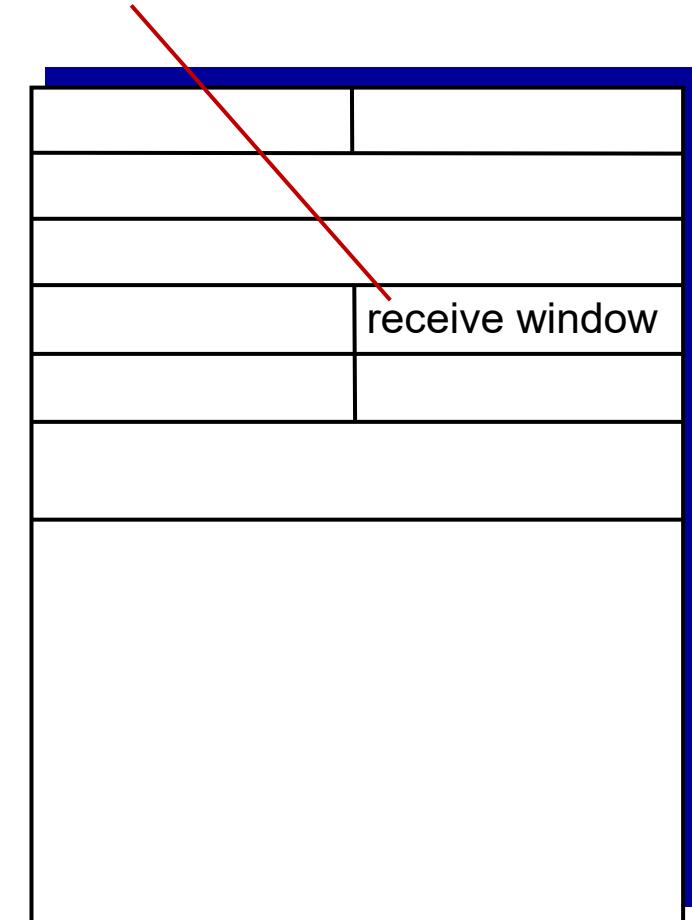
- TCP receiver “**advertises**” free buffer space in *receive window* (**rwnd**) field in TCP header of ACK packets
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - many operating systems auto adjust **RcvBuffer**
- sender **limits** amount of unACKed (“in-flight”) data to received **rwnd**
- guarantees receive buffer will not overflow
- *receive window* is 16 bit --> 65 535 bytes



TCP flow control

- TCP receiver “**advertises**” free buffer space in *receive window* (**rwnd**) field in TCP header of ACK packets
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - many operating systems auto adjust **RcvBuffer**
- sender **limits** amount of unACKed (“in-flight”) data to received **rwnd**
- guarantees receive buffer will not overflow
- *receive window* is 16 bit --> 65 535 bytes

flow control: # bytes receiver willing to accept



TCP segment format

Chapter 3: roadmap

3.1 Transport-layer services

3.2 Port numbers

3.3 Connectionless transport: UDP

 UDP socket programming

3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP

~~3.6 Principles of congestion control~~

3.7 TCP congestion control

3.8 QUIC: Quick UDP Internet Connections

Principles of congestion control

Congestion:

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



congestion control:
too many senders,
sending too fast

flow control: one sender
too fast for one receiver

Chapter 3: roadmap

3.1 Transport-layer services

3.2 Port numbers

3.3 Connectionless transport: UDP

 UDP socket programming

3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP

3.6 ~~Principles of congestion control~~

3.7 TCP congestion control

3.8 QUIC: Quick UDP Internet Connections

TCP congestion control: AIMD

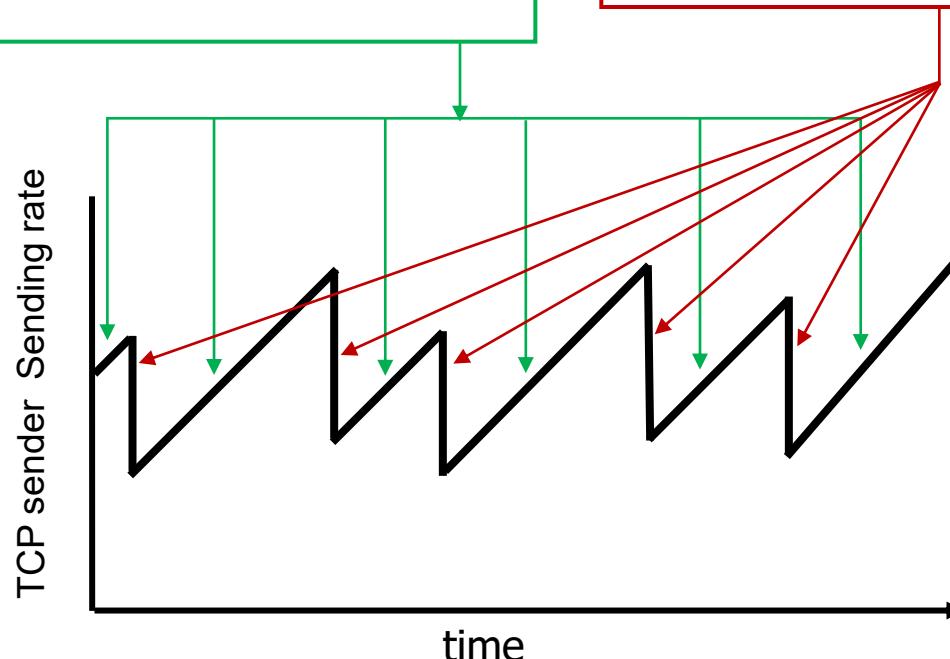
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP AIMD: more

Multiplicative decrease detail: sending rate is

- Cut in half on loss detected by **triple duplicate ACK** (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by **timeout** (TCP Tahoe)

Why AIMD?

- AIMD – a distributed, asynchronous algorithm – has been shown to:
 - optimize congested flow rates network wide!
 - have desirable stability properties

TCP fast retransmit

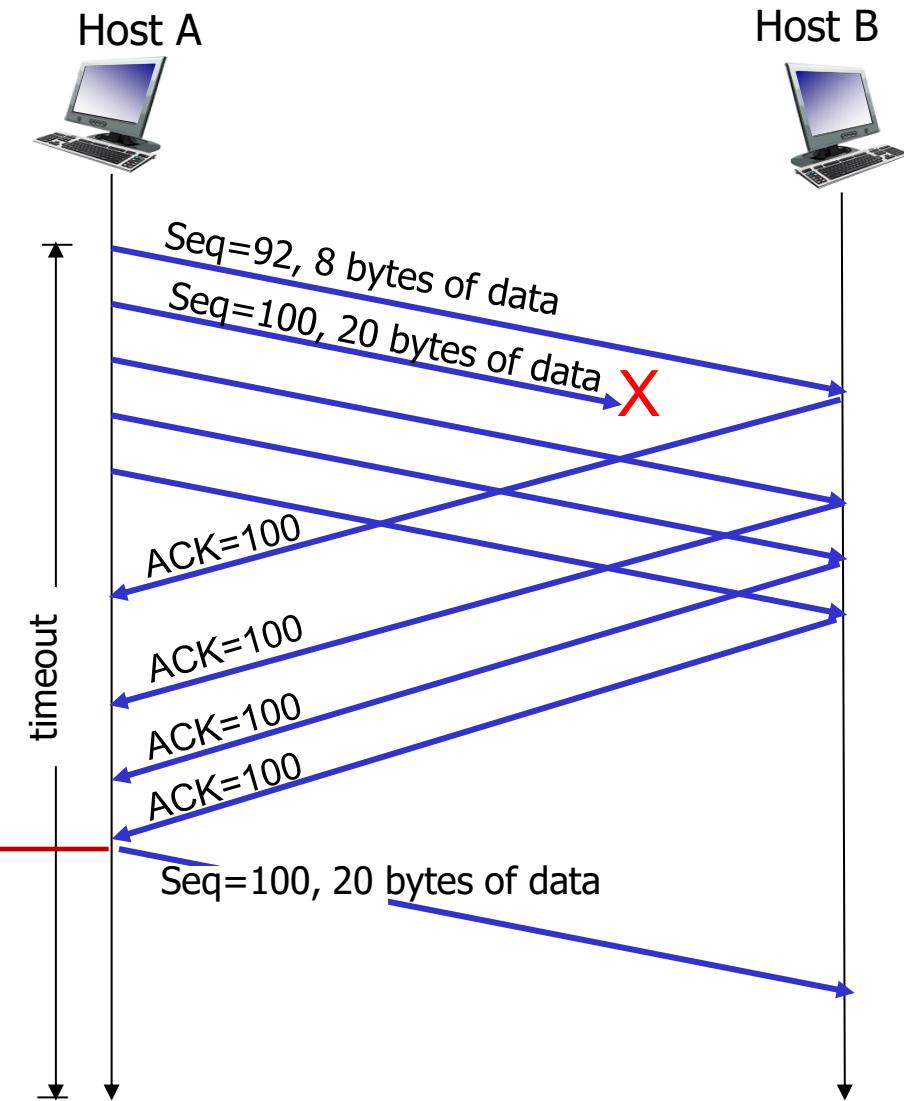
TCP fast retransmit

if sender receives 3 additional ACK's for same data ("triple duplicate ACK's"), resend unACK'ed segment with smallest seq #

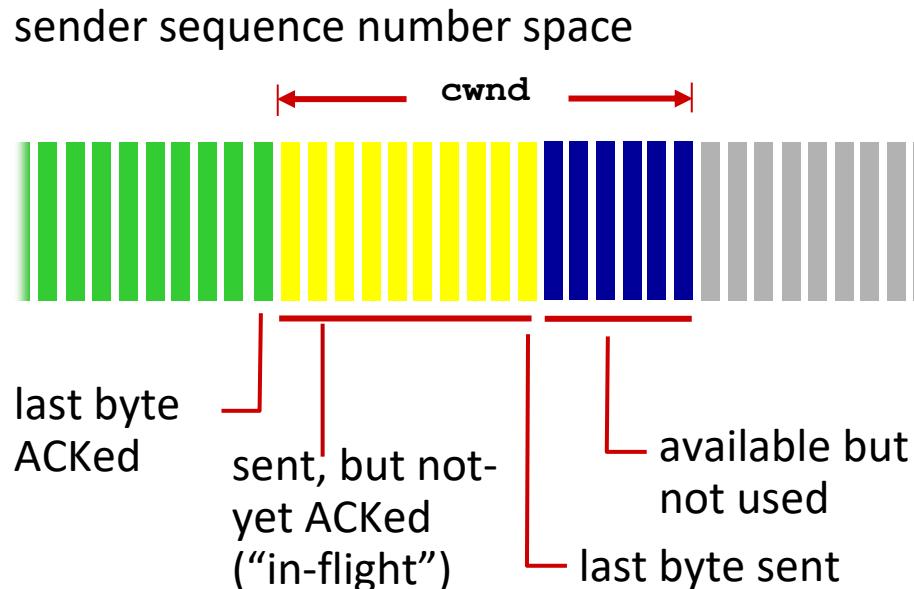
- likely that unACK'ed segment lost, so don't wait for timeout



Receipt of three duplicate ACK's indicates 3 segments received after a missing segment – lost segment is likely. So, retransmit!



TCP congestion control: details



TCP sending behavior:

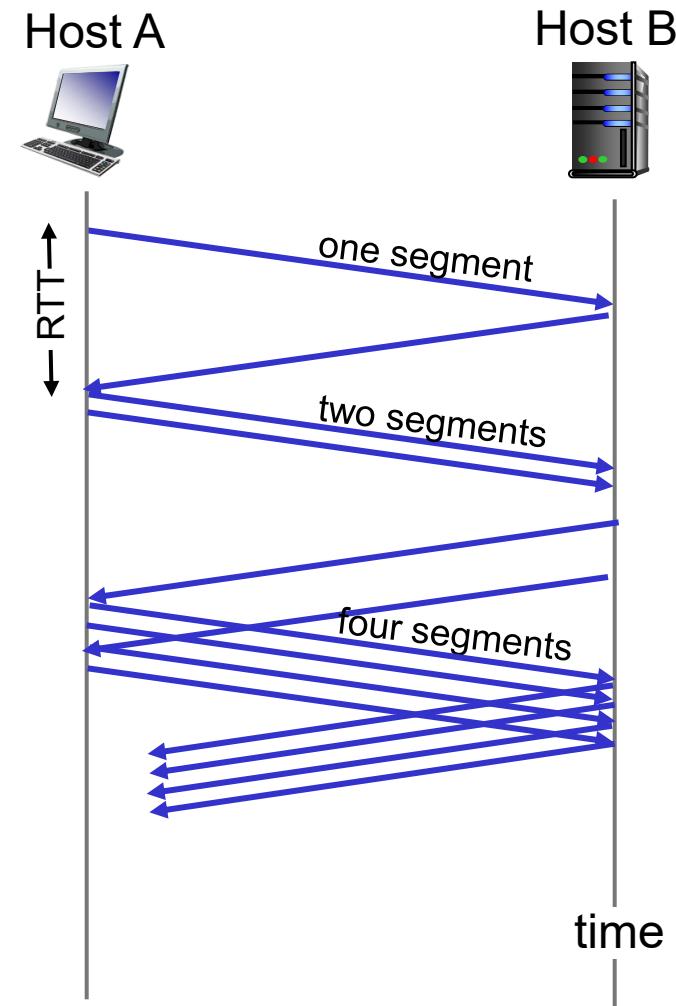
- *roughly*: send $cwnd$ bytes, wait RTT for ACKS, then send more bytes

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

- TCP sender limits transmission: $\text{LastByteSent} - \text{LastByteAcked} \leq cwnd$
- $cwnd$ is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

TCP slow start

- when connection begins, increase rate **exponentially** until first loss event:
 - initially **cwnd** = 1 MSS
 - double **cwnd** every RTT
 - done by incrementing **cwnd** for every ACK received
- **summary:** initial rate is slow, but ramps up exponentially fast



TCP: from slow start to congestion avoidance

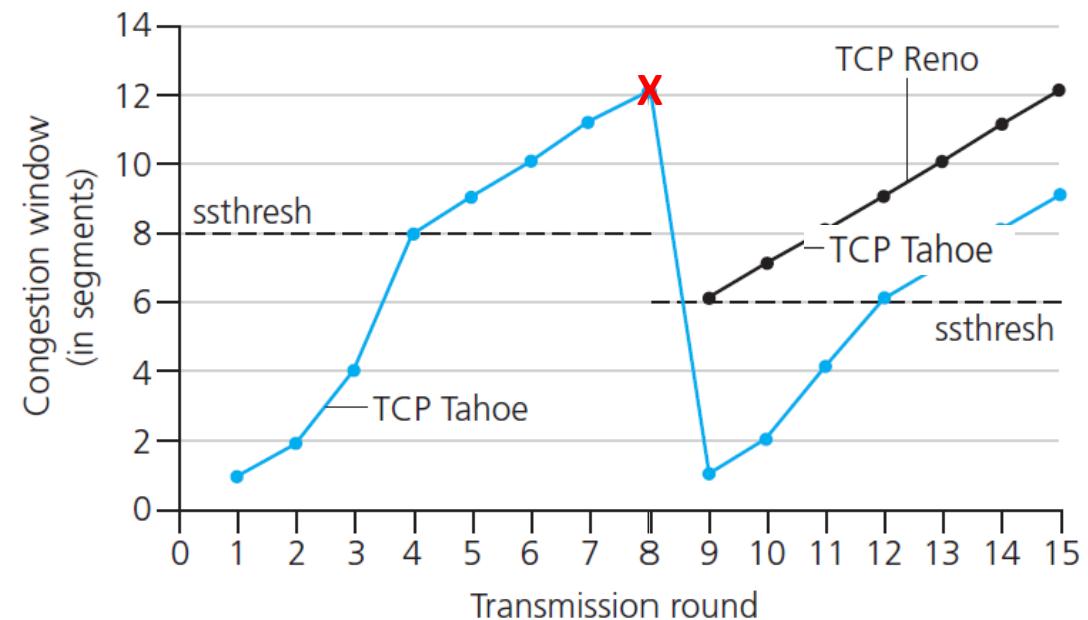
Q: when should the exponential increase switch to linear?

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

Implementation:

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

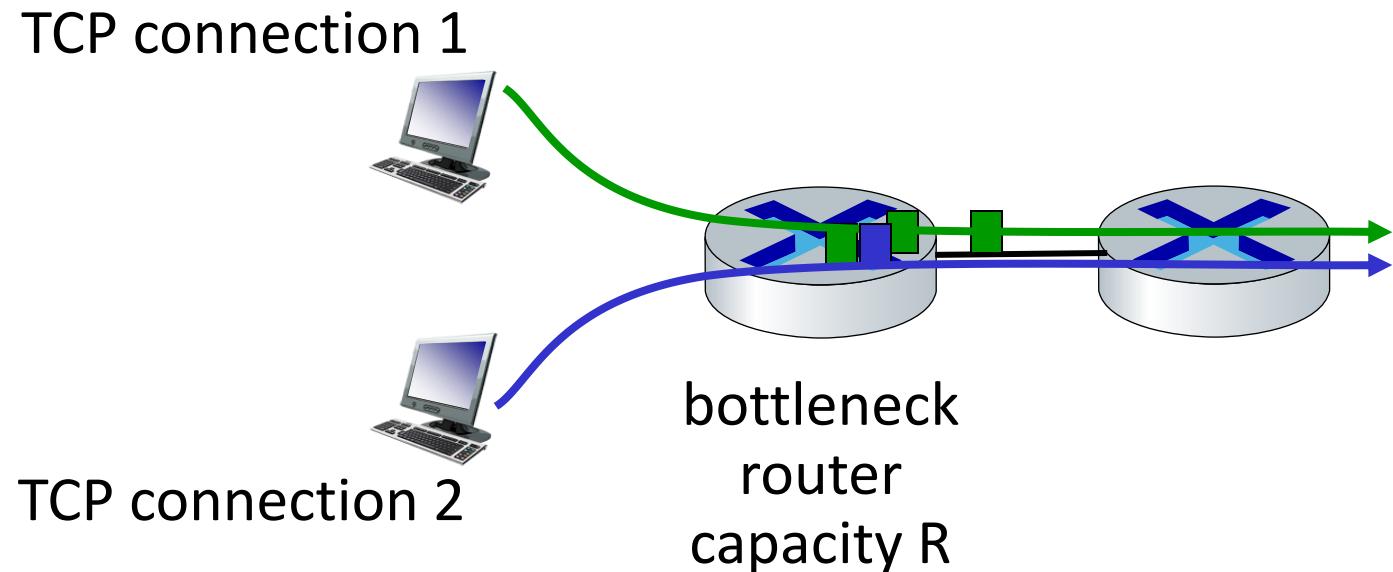
- TCP Tahoe: Cut to 1 MSS (maximum segment size) when loss detected by **timeout**
- TCP Reno: Cut in half on loss detected by **triple duplicate ACK**



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

TCP fairness

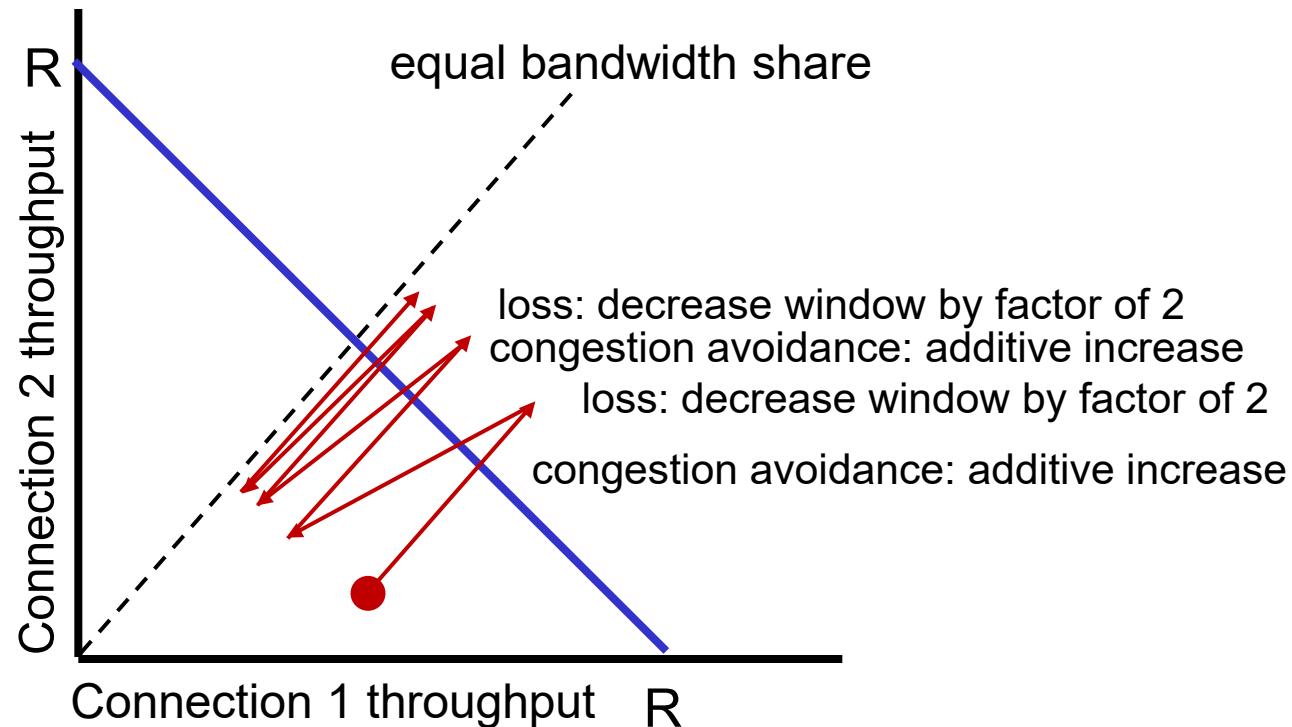
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R , each should have average rate of R/K



Q: is TCP Fair?

Example: two competing TCP sessions:

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



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

Fairness: must all network apps be “fair”?

Fairness and UDP

- multimedia apps often do not use TCP
 - do not want rate throttled by congestion control
- instead use UDP:
 - send audio/video at constant rate, tolerate packet loss
- there is no “Internet police” policing use of congestion control

Fairness, parallel TCP connections

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

Transport layer: roadmap

3.1 Transport-layer services

3.2 Port numbers

3.3 Connectionless transport: UDP

 UDP socket programming

3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP

3.6 ~~Principles of congestion control~~

3.7 TCP congestion control

3.8 QUIC: Quick UDP Internet Connections

QUIC: Quick UDP Internet Connections

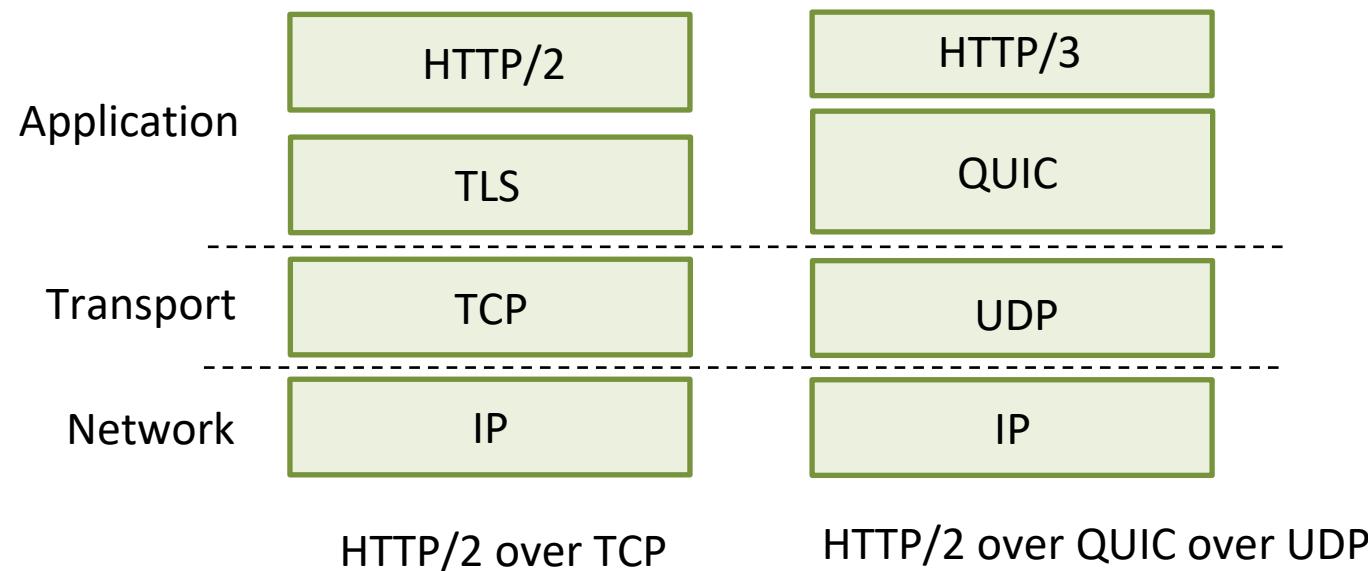
- TCP, UDP: principal transport protocols for 40 years
- different “flavors” of TCP developed, for specific scenarios:

Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets “in flight”; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, “background” TCP flows

- moving transport–layer functions to application layer, on top of UDP
 - HTTP/3: QUIC

QUIC: Quick UDP Internet Connections

- QUIC is an **application-layer protocol**, on top of UDP
 - HTTP/3, increase performance of HTTP
 - deployed on many Google servers, apps (Chrome, mobile YouTube app)

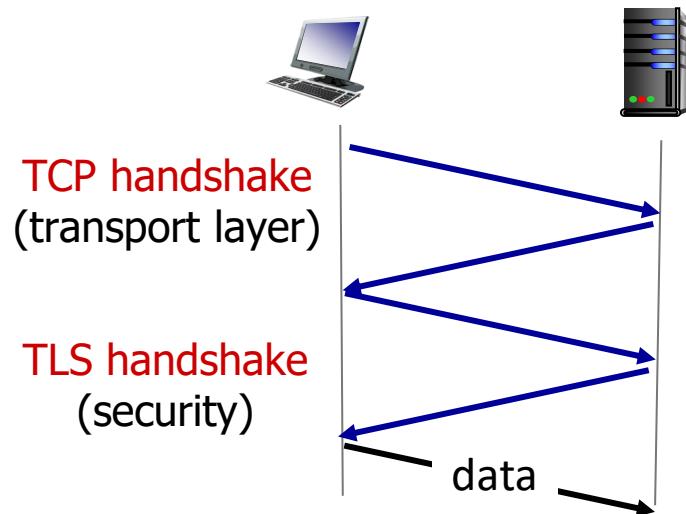


QUIC: Quick UDP Internet Connections

adopts approaches we've studied in this chapter for connection establishment, error control, congestion control

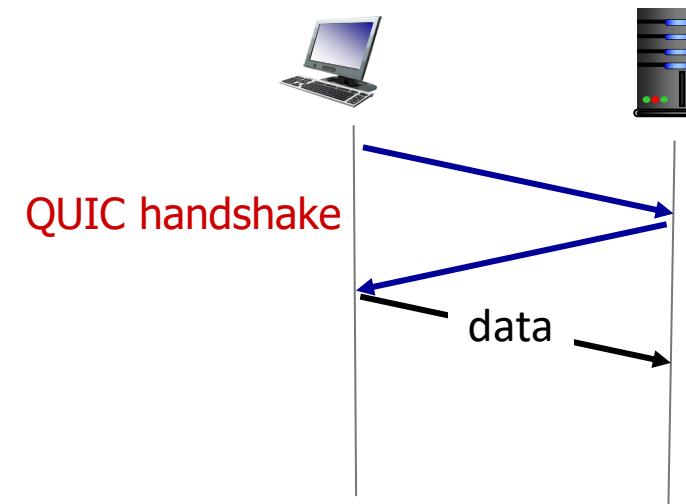
- Connection-oriented: Two endpoints
- Reliable data transfer: In-order packet delivery, retransmissions of lost packets (on application-level)
- Security: authentication, encryption
- Congestion control
- HTTP/3 multiple application-level “streams” of multiplexed over single QUIC connection
 - separate reliable data transfer, security
 - common congestion control

QUIC: Connection establishment



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

- 2 or 3 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

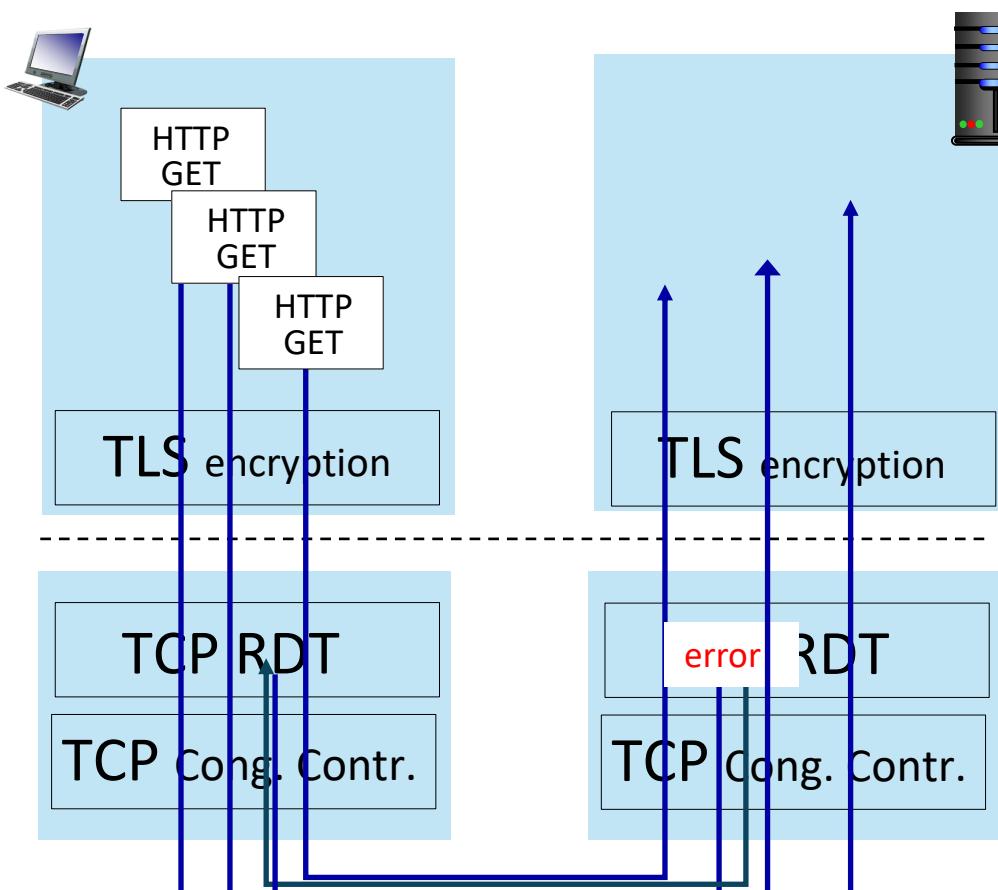
- 1 handshake

HOL blocking problem

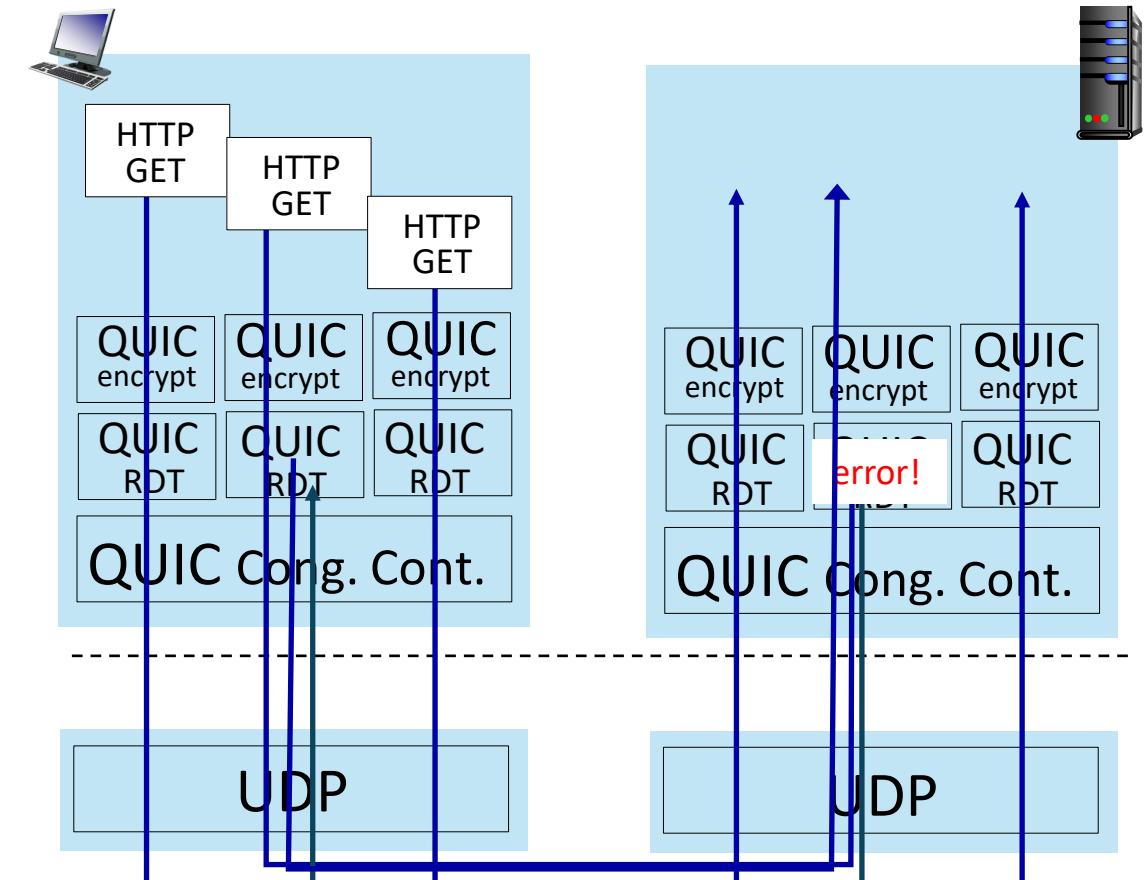
- HTTP/1.1 sending multiple HTTP requests, all over a single TCP connection
- Since TCP provides reliable, in-order byte delivery, this means that the multiple HTTP requests must be delivered in-order at the destination HTTP server.
- If a packet of one HTTP request are lost, TCP at the HTTP server cannot restore the remaining HTTP requests until the lost packet is retransmitted and correctly received
- A variant of the HOL blocking problem (Section 2.2.5)

QUIC: streams: parallelism, no HOL blocking

application



(a) HTTP 1.1



(b) HTTP/2 with QUIC: no HOL blocking

Chapter 3: summary

- principles behind transport layer services:
 - port numbers
 - reliable data transfer
 - flow control
 - congestion control
- instantiation, implementation in the Internet
 - UDP
 - TCP

Up next:

- leaving the network “edge” (application, transport layers)
- into the network “core”
- two network-layer chapters:
 - data plane
 - control plane

Chapter 4

Network Layer:

Data Plane

A note on the use of these PowerPoint slides:

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

In return for use, we only ask the following:

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

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

EIGHTH EDITION

James F. Kurose • Keith W. Ross



Computer Networking: A Top-Down Approach

8th edition

Jim Kurose, Keith Ross
Pearson, 2020

Network layer: our goals

- understand principles behind network layer services, focusing on **data plane**:
 - network layer service models
 - forwarding versus routing
 - how a router works
 - addressing
 - generalized forwarding
- IP protocol
- Network address translation (NAT)

Network layer: “data plane” roadmap

4.1 Network layer: overview

- forwarding plane (data plane)
- routing (control plane)

4.2 What's inside a router

- input ports, forwarding,
- switching, output ports, scheduling

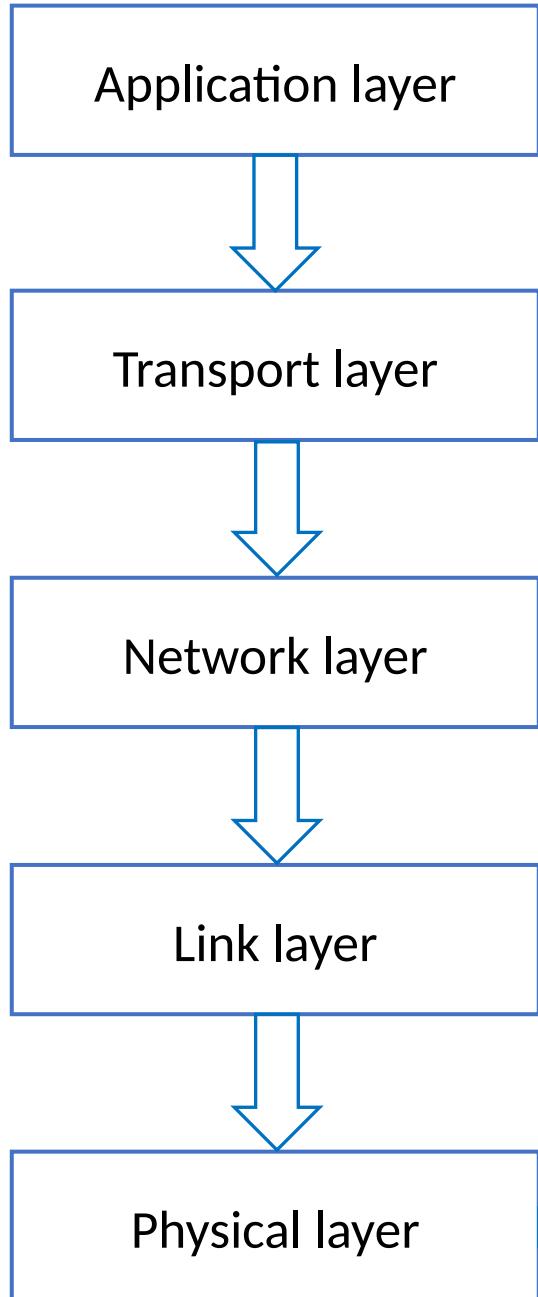
4.3 IP: the Internet Protocol

- datagram format
- addressing
- network address translation
- IPv6

4.4 Generalized Forwarding, SDN

- match + action
- OpenFlow: match + action in action

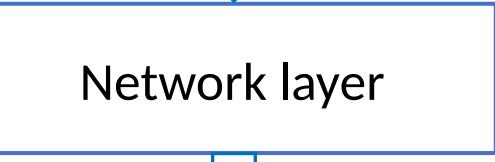
Sender



Message

M

Segment



Application layer

M

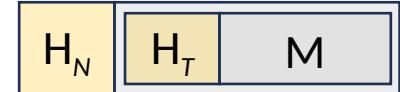
Transport layer

Network layer

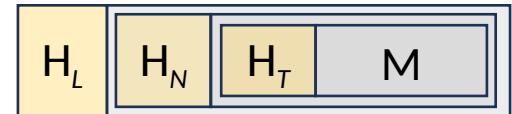
Link layer

Physical layer

Datagram



Frame



H_T M

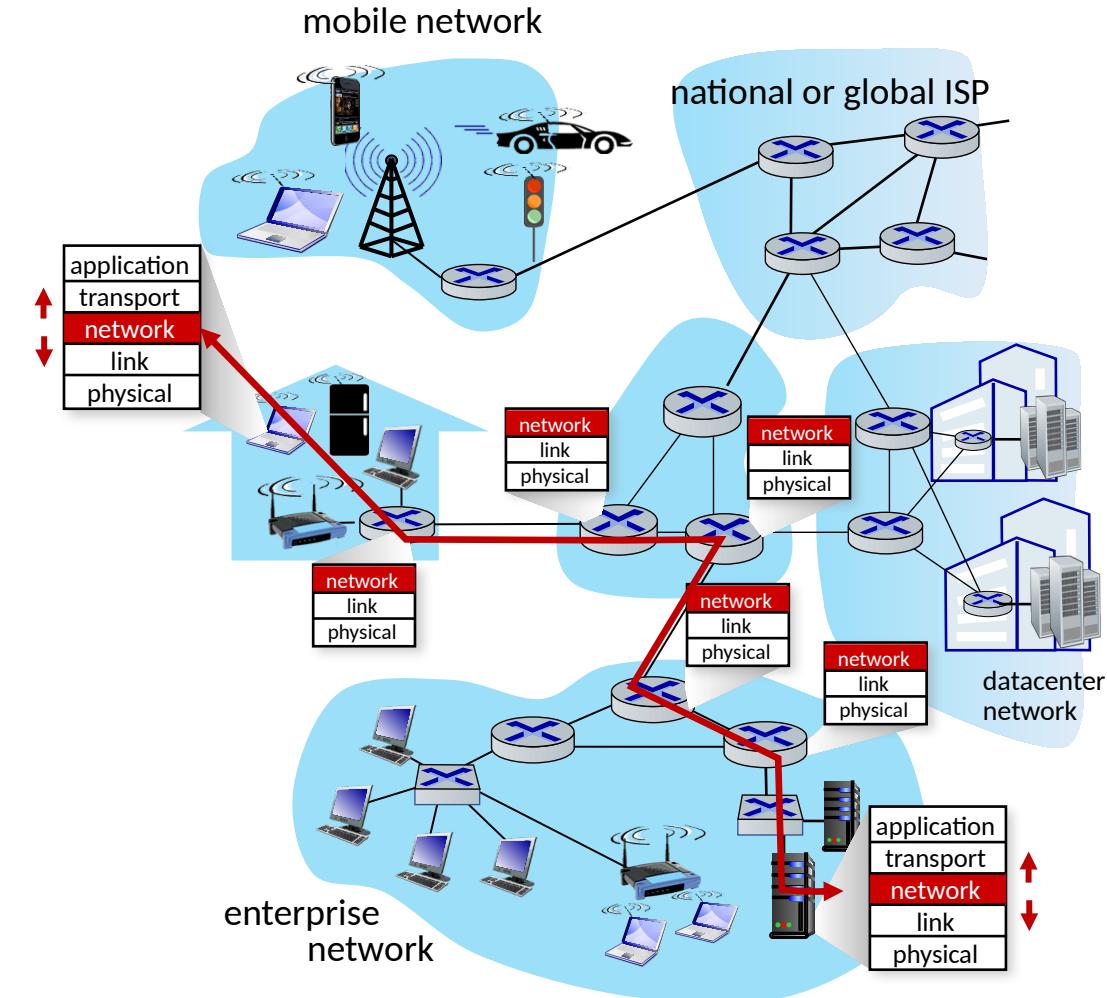
H_N H_T M

Physical layer

Physical layer

Network-layer services and protocols

- transport segment from sending to receiving host
 - **sender:** encapsulates segments into datagrams, passes to link layer
 - **receiver:** delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- **routers:**
 - examines header fields in all IP datagrams passing through it
 - moves datagrams from input ports to output ports to transfer datagrams along end-end path



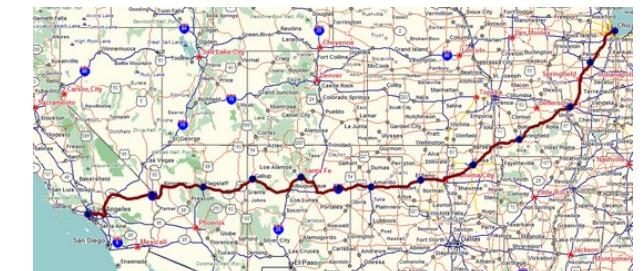
Two key network-layer functions

network-layer functions:

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

analogy: taking a trip

- **routing**: process of planning trip from source to destination



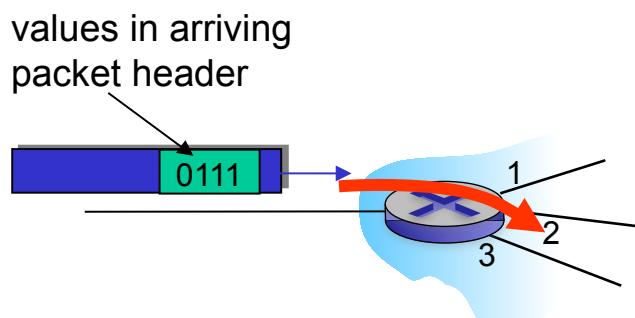
routing

- **forwarding**: process of getting through single interchange

Network layer: forwarding plane, control plane

Forwarding (“data plane”):

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

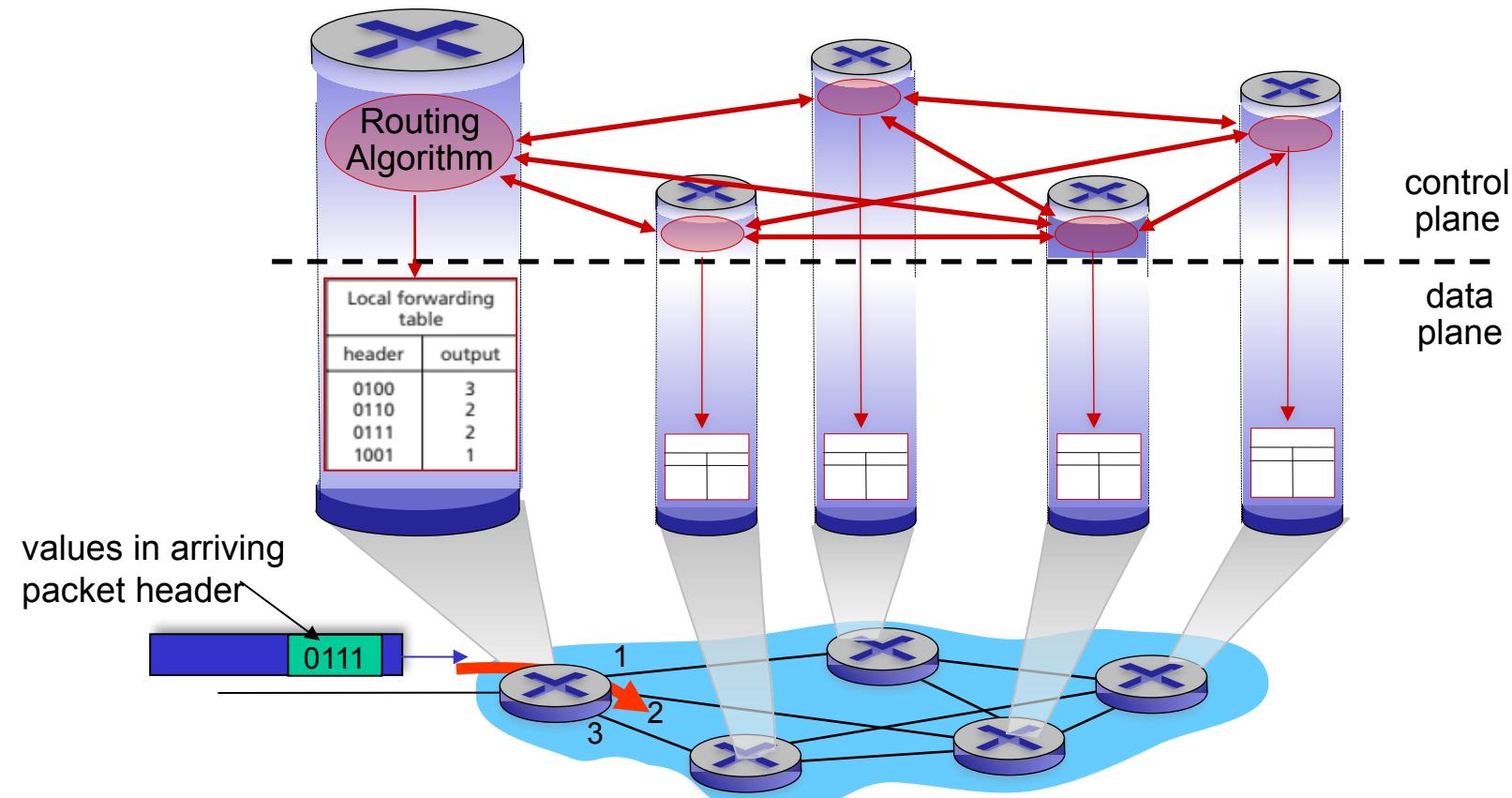


Routing (“control plane”):

- *network-wide logic*
- determines how datagram is *routed* among routers along end-end path from source host to destination host
- two control-plane approaches:
 - *Distributed routing*: implemented in routers
 - *Centralized routing (aka. software-defined networking)*: implemented in remote servers

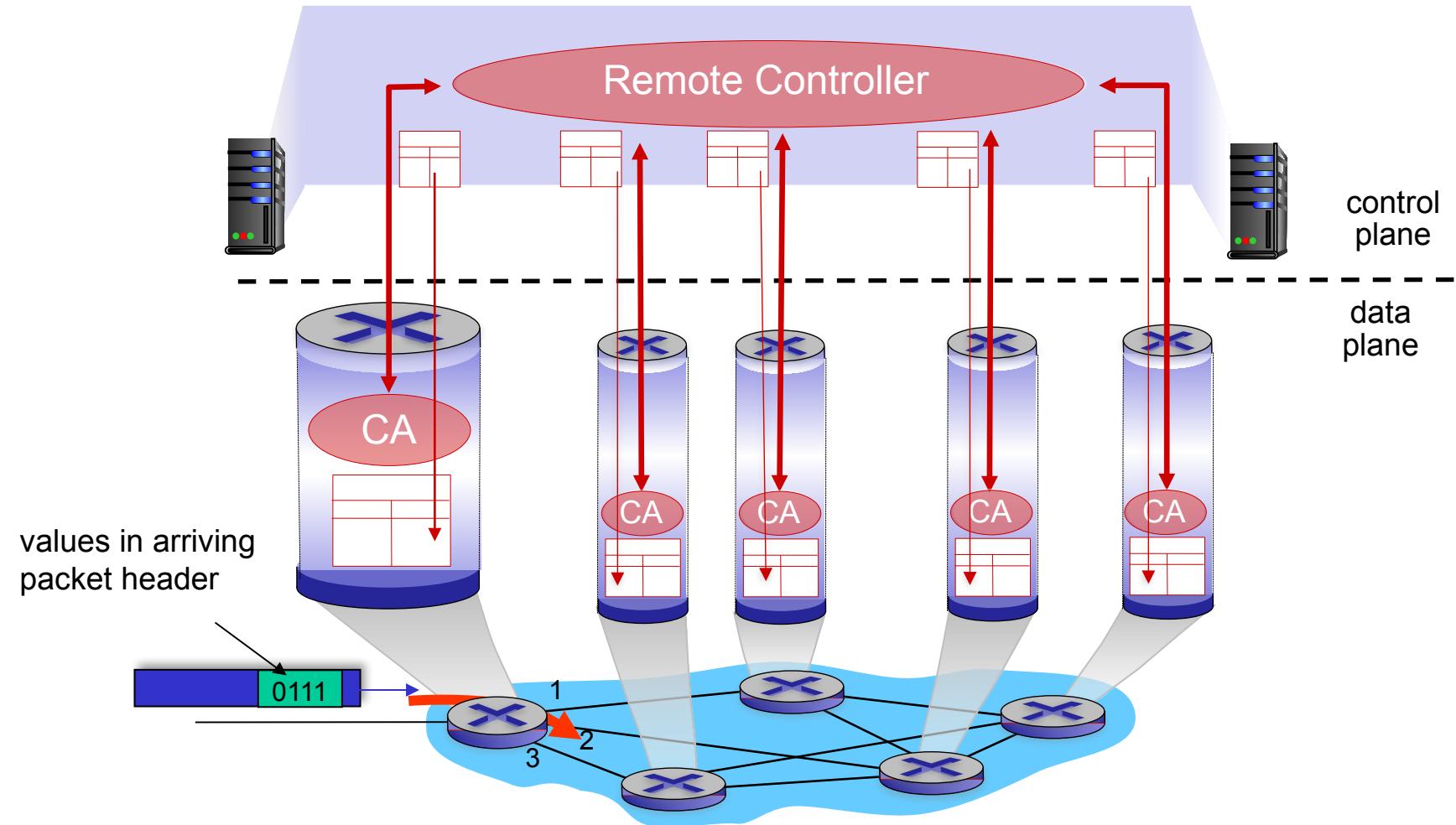
Distributed routing

Individual routing algorithms *in each and every router*



Centralized routing: Software-Defined Networking (SDN)

Remote controller computes forwarding tables in routers



IP network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no

Internet “best effort” service model

No guarantees on:

- i. successful datagram delivery (r.d.t.)
- ii. order of delivery (r.d.t.)
- iii. bandwidth available to end-end flow

Network layer: “forwarding plane” roadmap

4.1 Network layer: overview

- forwarding plane
- control plane

4.2 What's inside a router

- input ports, forwarding,
- switching, output ports, scheduling

4.3 IP: the Internet Protocol

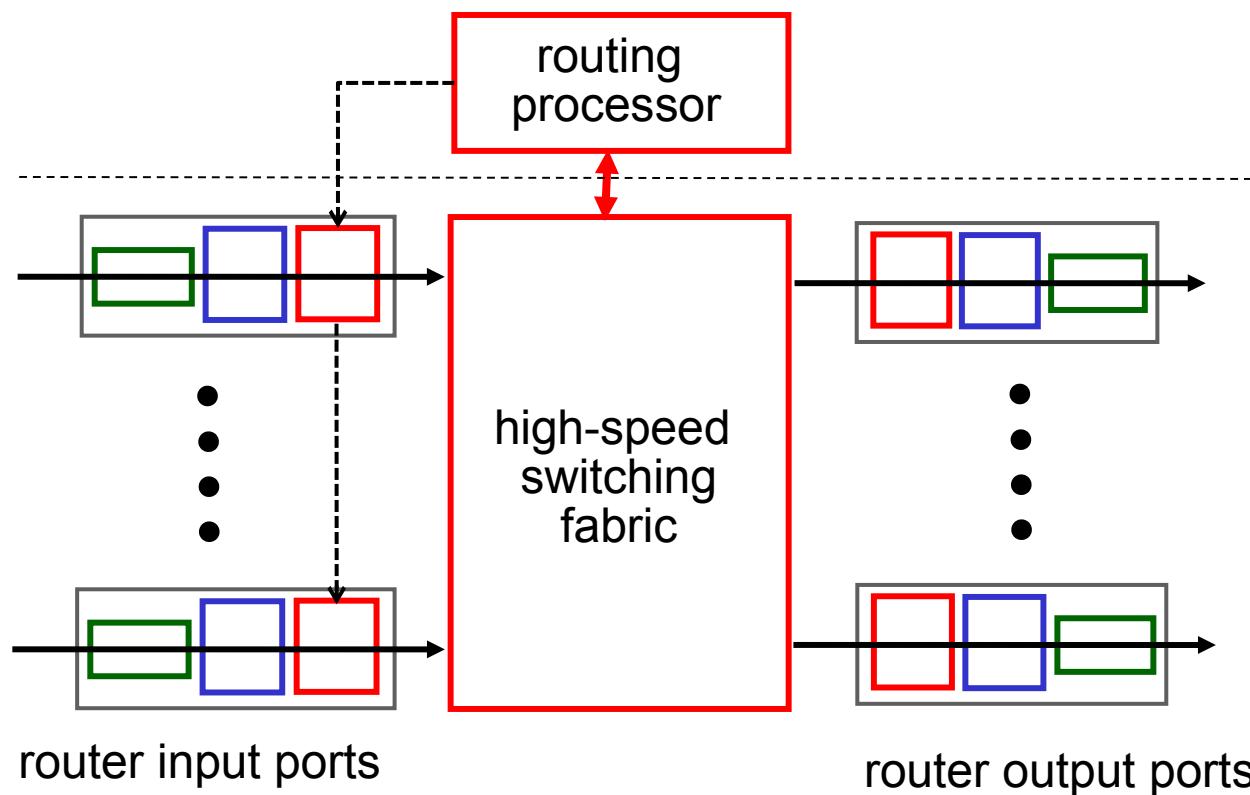
- datagram format
- addressing
- network address translation
- IPv6

4.4 Generalized Forwarding, SDN

- match + action
- OpenFlow: match + action in action

Router architecture overview

high-level view of generic router architecture:



routing, management
- **control plane** (software)
operates in millisecond
time frame

forwarding - data plane
(hardware) operates in
nanosecond timeframe

Destination-based forwarding

<i>forwarding table</i>	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through	0
11001000 00010111 00010000 00000100 through	3
11001000 00010111 00010000 00000111	
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000 through	2
11001000 00010111 00011111 11111111	
otherwise	3

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

Longest prefix matching

longest prefix match

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

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

- examples:
- 11001000 00010111 00010110 10100001 which interface?
 - 11001000 00010111 00011000 10101010 which interface?

Longest prefix matching

longest prefix match

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

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

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

Longest prefix matching

longest prefix match

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

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

match!

examples:

11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

Longest prefix matching

longest prefix match

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

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

match!

examples:

11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

Longest prefix matching

longest prefix match

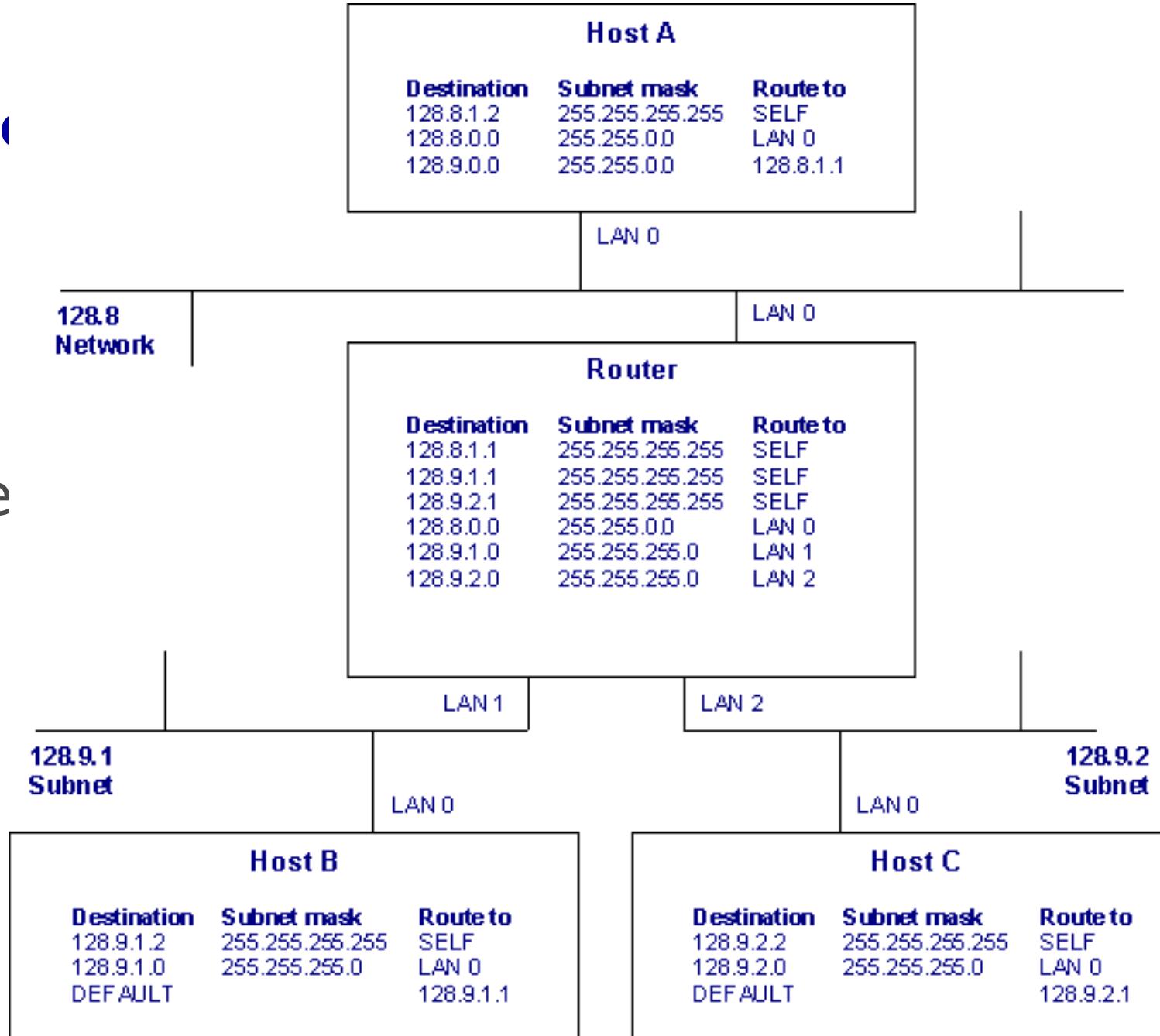
when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

new entry

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
11001000 00010111 00010000 000001**	3
otherwise	3

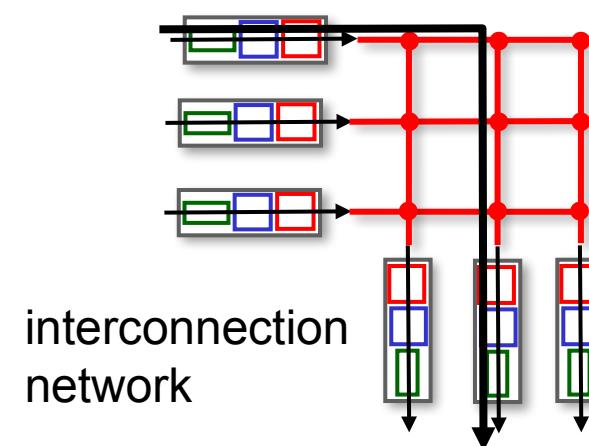
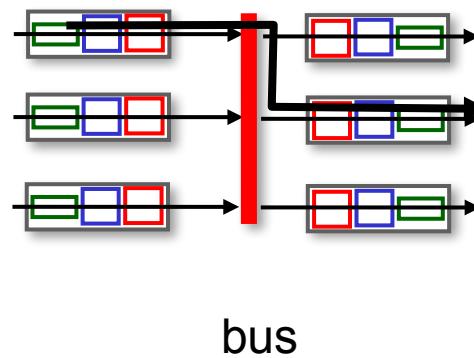
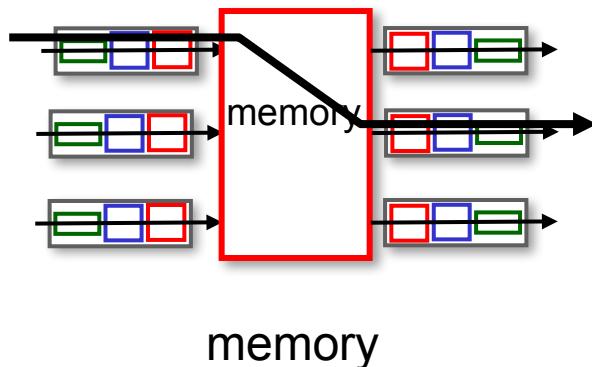
Destination-based forwarding example

- Host A sends a packet to **128.9.2.2** (Host C)
- The router has three addresses
 - one IP address per interface



Switching fabrics

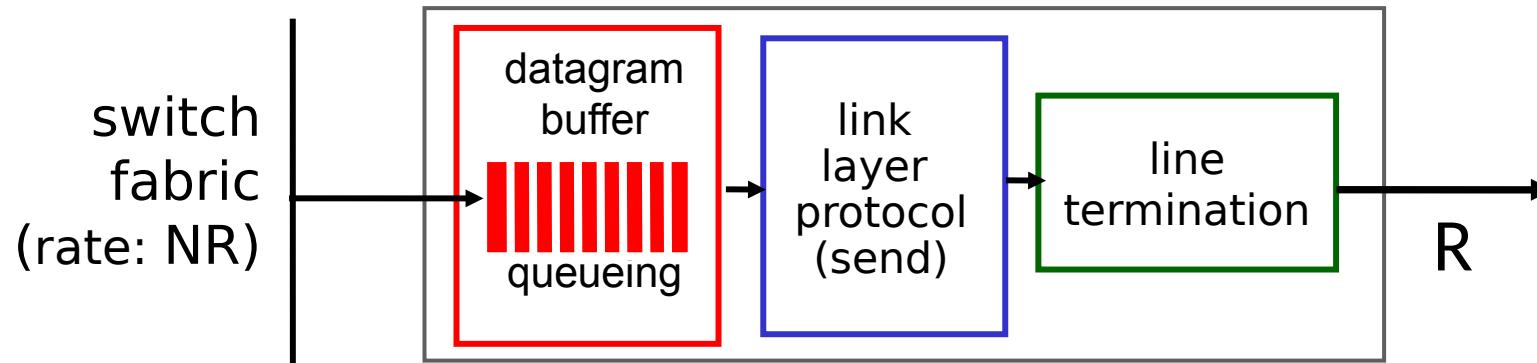
- transfer packet from input link to appropriate output link
- **switching rate:** rate at which packets can be transferred from inputs to outputs
 - often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate desirable
- three major types of switching fabrics:



Output port queuing



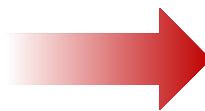
This is a really important slide



- *Buffering* required when datagrams arrive from fabric faster than link transmission rate. *Drop policy*: which datagrams to drop if no free buffers?
- *Scheduling discipline* chooses among queued datagrams for transmission



Datagrams can be lost due to congestion, lack of buffers

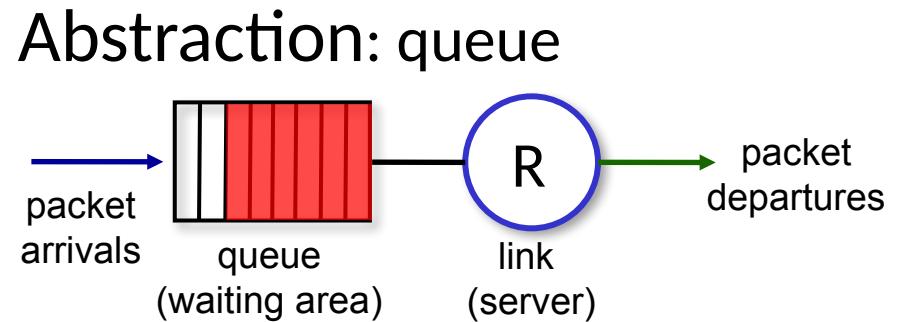


Priority scheduling – who gets best performance, network neutrality

Packet scheduling

packet scheduling: deciding which packet to send next on link:

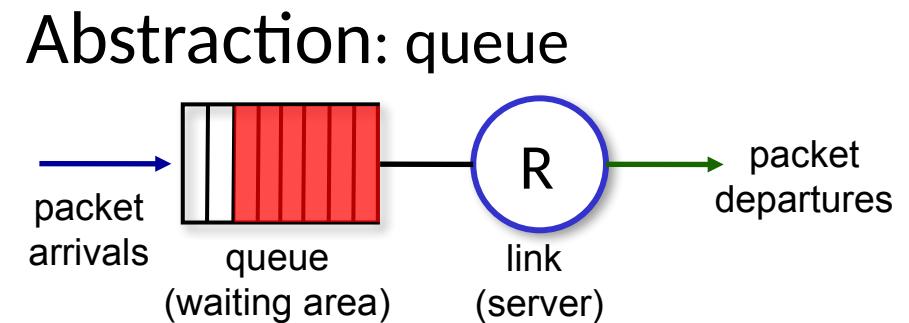
1. First come, first served (FCFS)
2. Priority
3. Round robin
4. Weighted fair queueing



Packet Scheduling: First-come-first-served (FCFS)

FCFS:

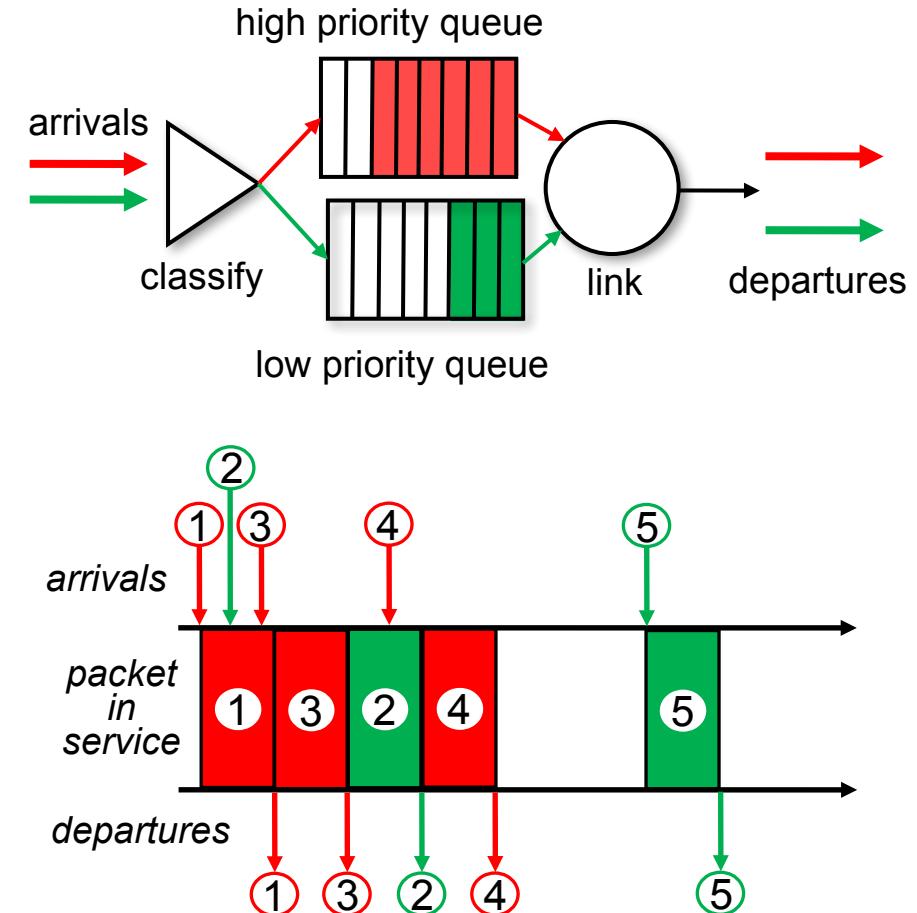
- packets transmitted in the same order they arrive the queue
- also known as: First-in-first-out (FIFO)
- real world examples?



Scheduling policies: Priority scheduling

Priority scheduling:

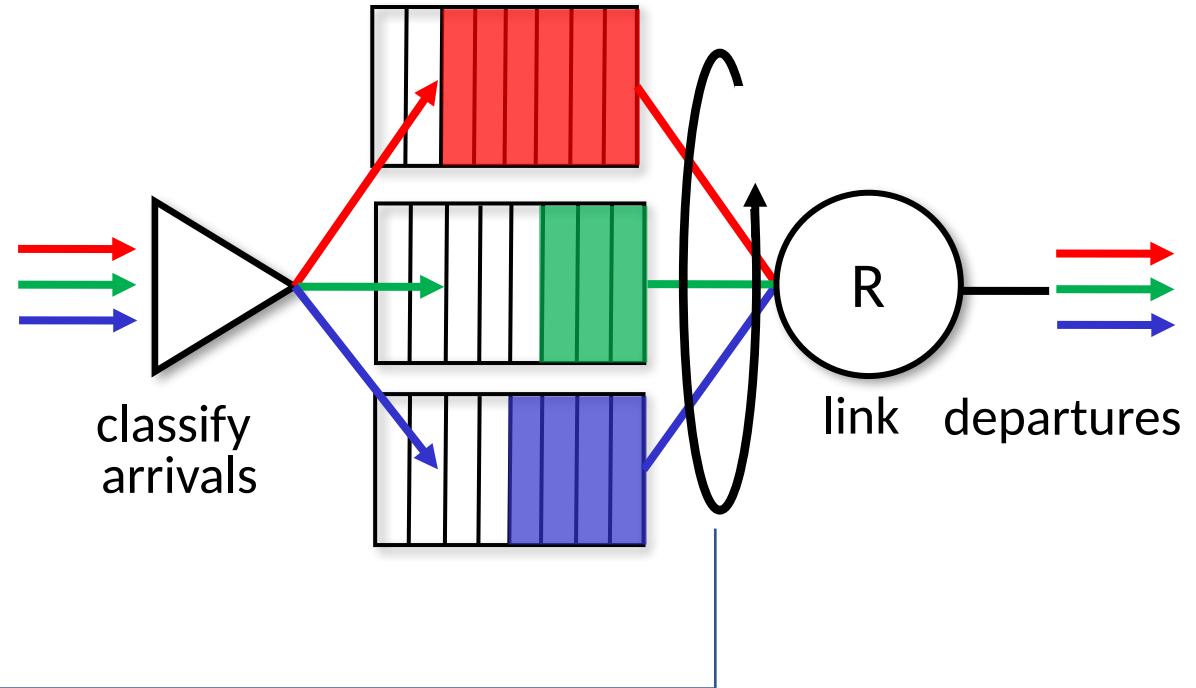
- arriving traffic classified, queued by class
 - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
 - FCFS within priority class



Scheduling policies: Round-robin scheduling

Round Robin (RR) scheduling:

- arriving traffic classified, queued by class
 - any header fields can be used for classification
- server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn



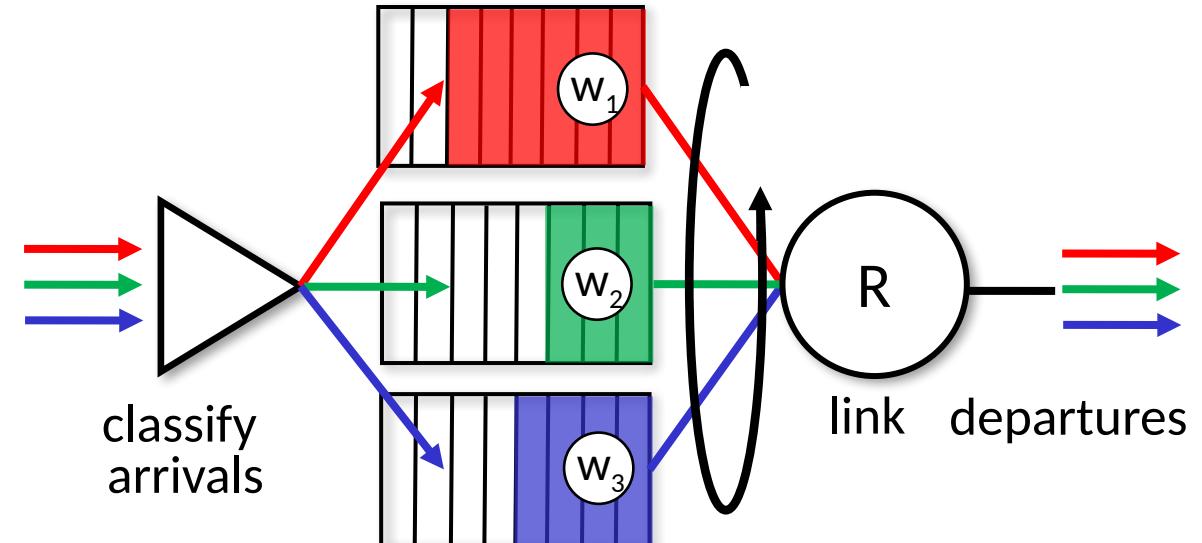
Scheduling policies: weighted fair queueing

Weighted Fair Queueing (WFQ):

- generalized Round Robin
- each class, i , has weight, w_i , and gets weighted amount of service in each cycle:

$$\frac{w_i}{\sum_j w_j}$$

- minimum bandwidth guarantee (per-traffic-class)



Network layer: “data plane” roadmap

4.1 Network layer: overview

- data plane
- control plane

4.2 What's inside a router

- input ports, forwarding,
- switching, output ports, scheduling

4.3 The Internet Protocol (IP)

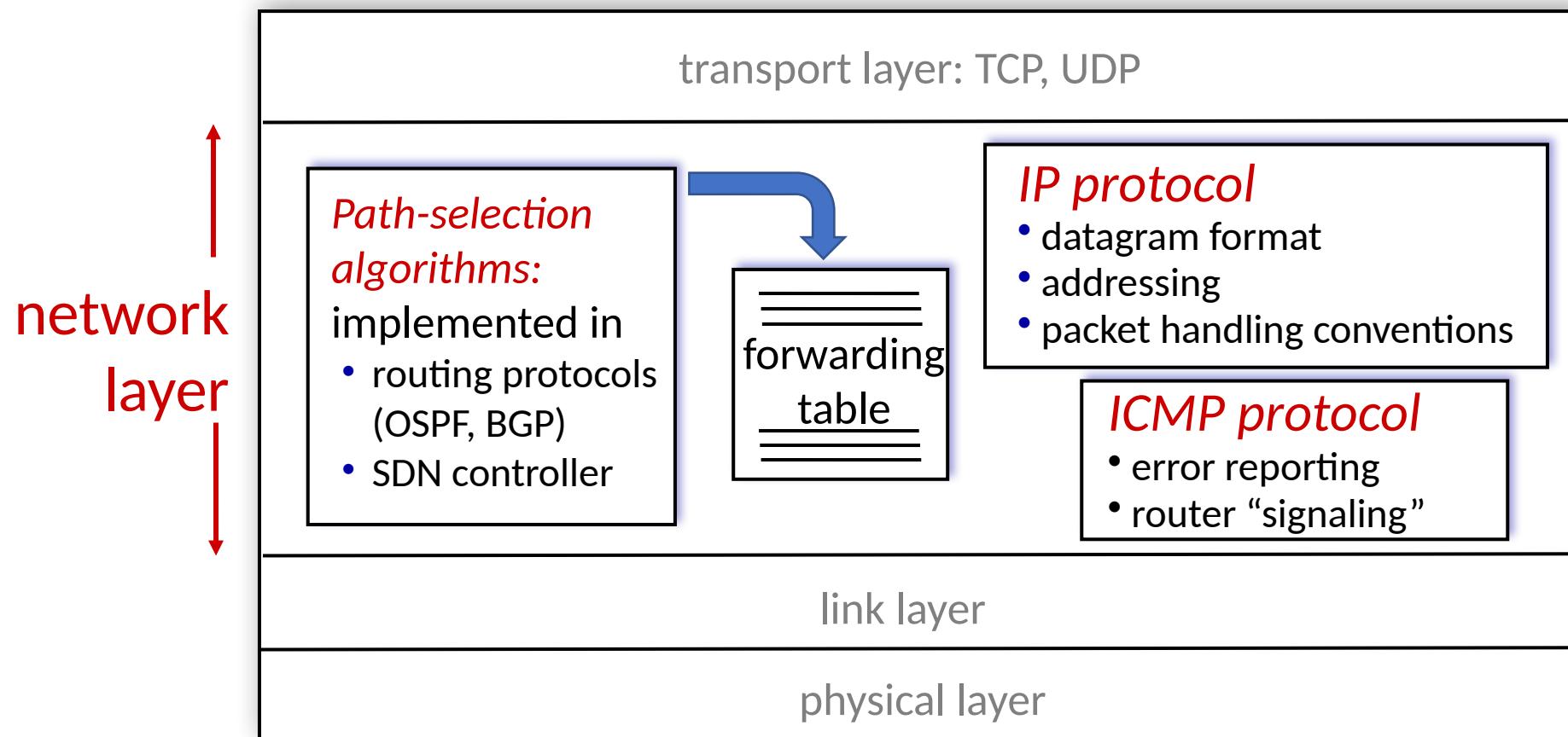
- datagram format
- addressing
- network address translation
- IPv6

4.4 Generalized Forwarding, SDN

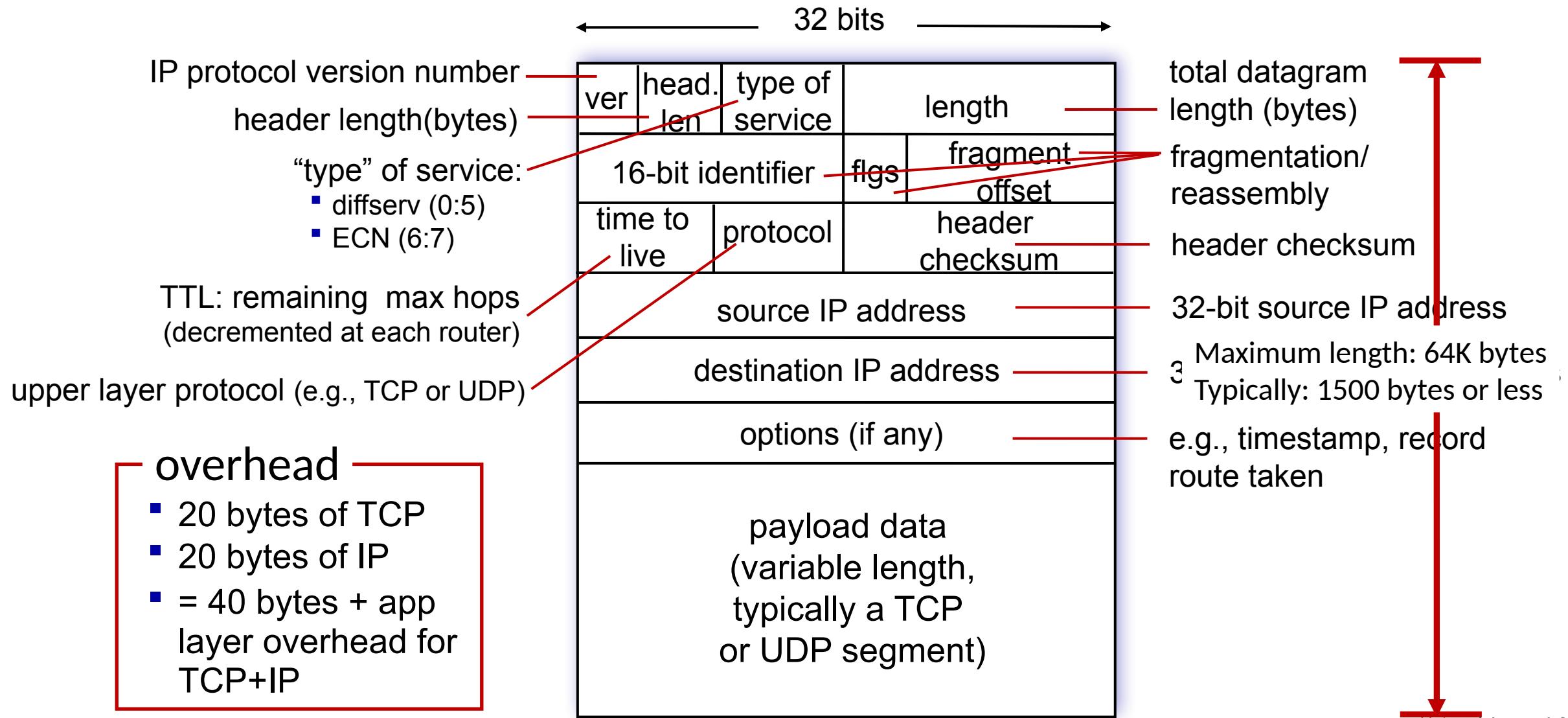
- match + action
- OpenFlow: match + action in action

Network Layer: Internet

host, router network layer functions:



IPv4 Datagram format



Maximum payload size: MTU vs. MSS

Maximum payload size:

- **Frames:**

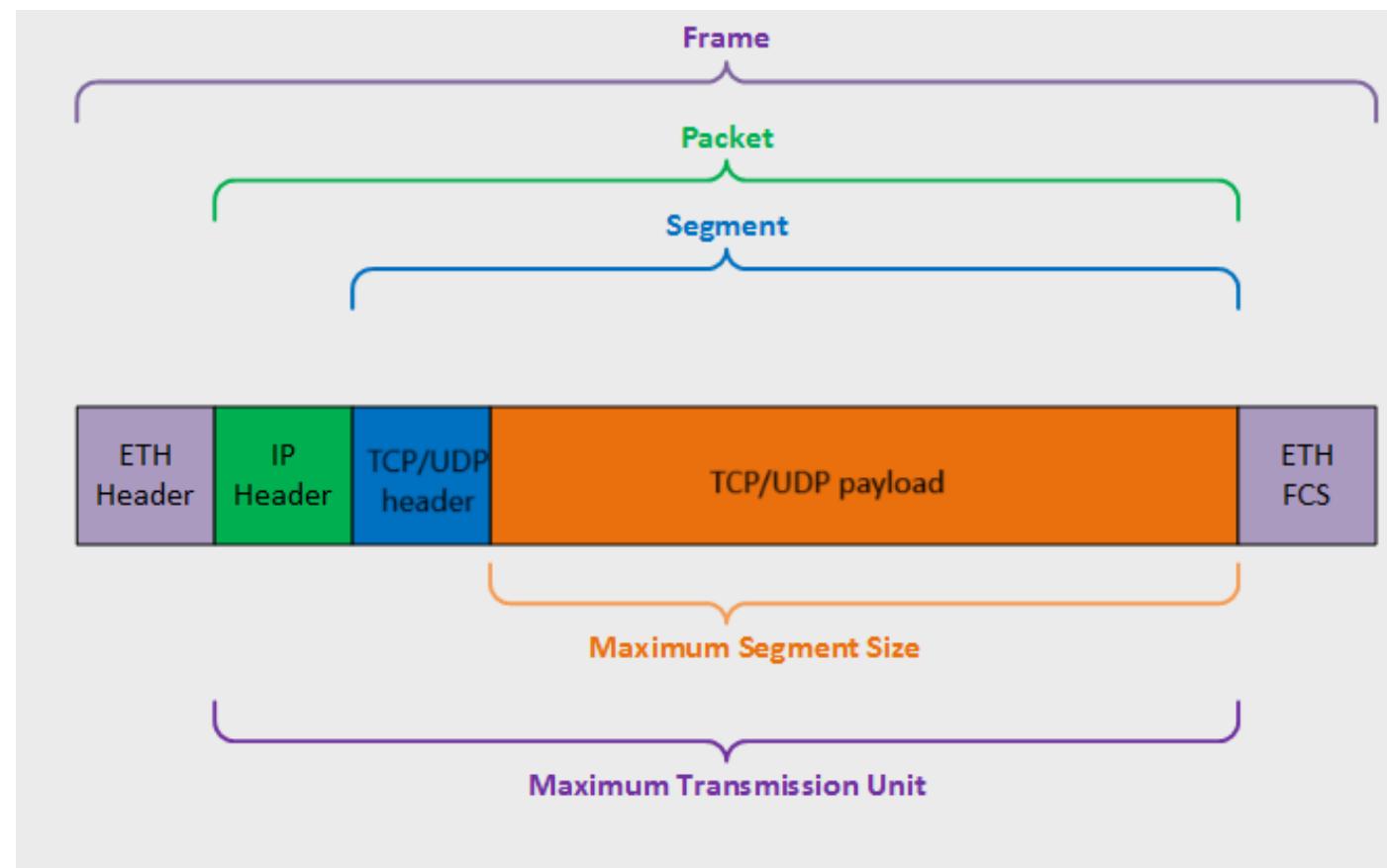
- Maximum transfer unit (MTU)

- e.g. Ethernet: 1500 bytes

- **Segments:**

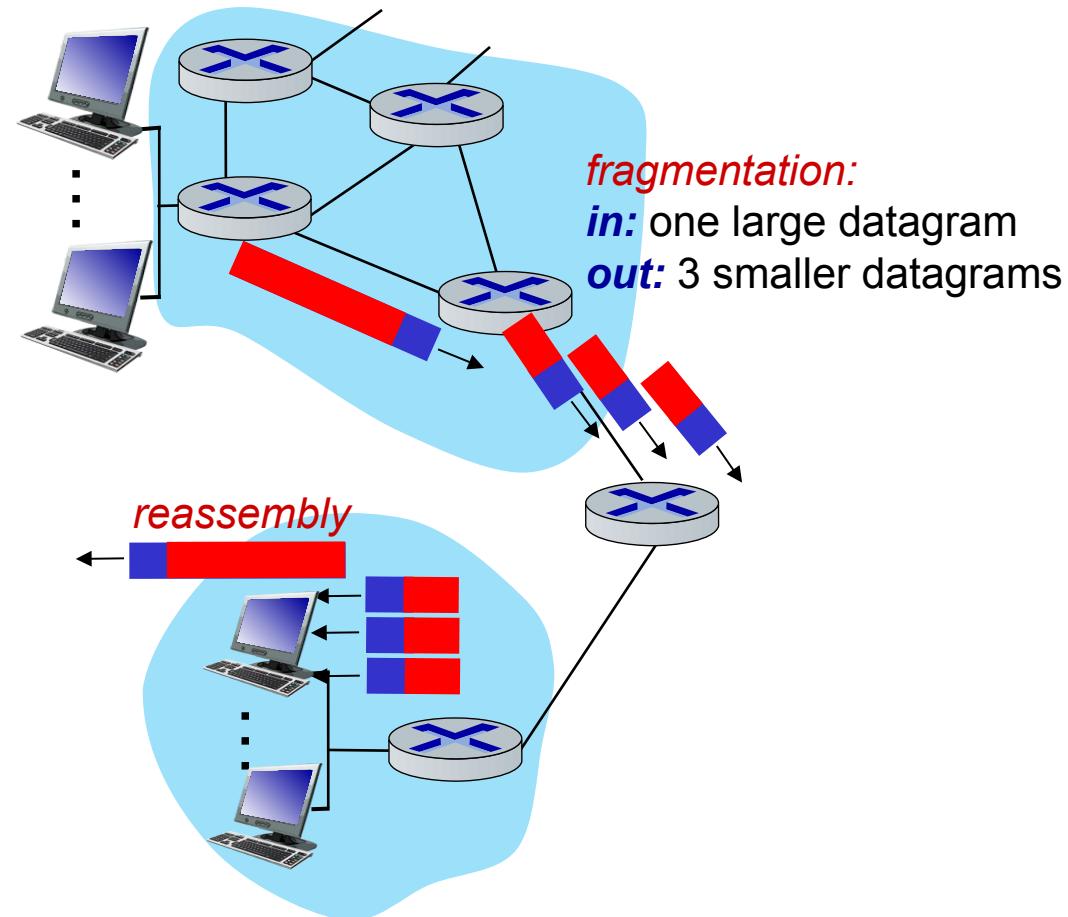
- Maximum segment size (MSS)

- $MSS = MTU - 40$
 $= 1500 - 40 = 1460$
 - $40 = IP \text{ header size} + TCP \text{ header size}$

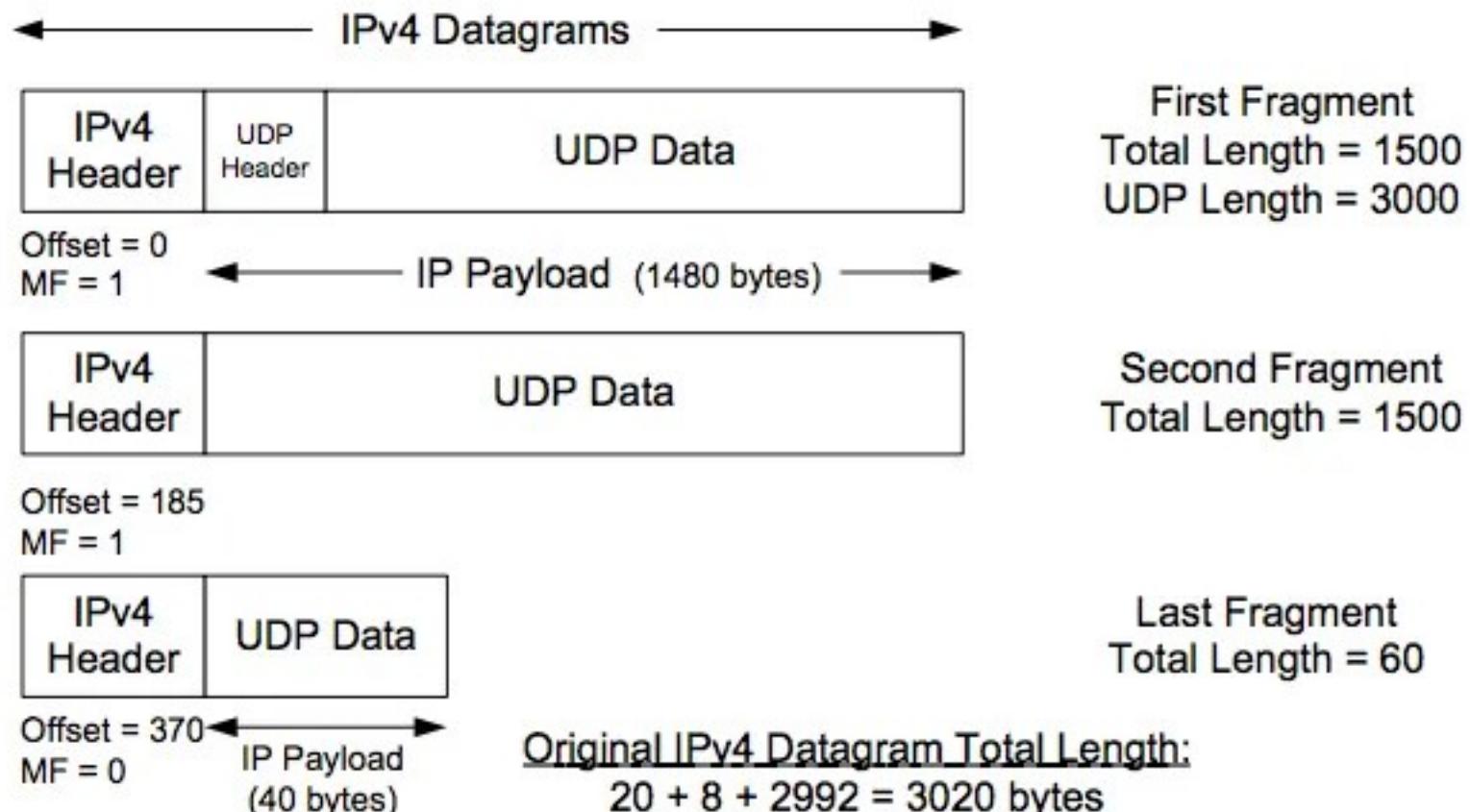


IP fragmentation

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



IP fragmentation



Network layer: “data plane” roadmap

4.1 Network layer: overview

- data plane
- control plane

4.2 What's inside a router

- input ports, forwarding,
- switching, output ports, scheduling

4.3 The Internet Protocol (IP)

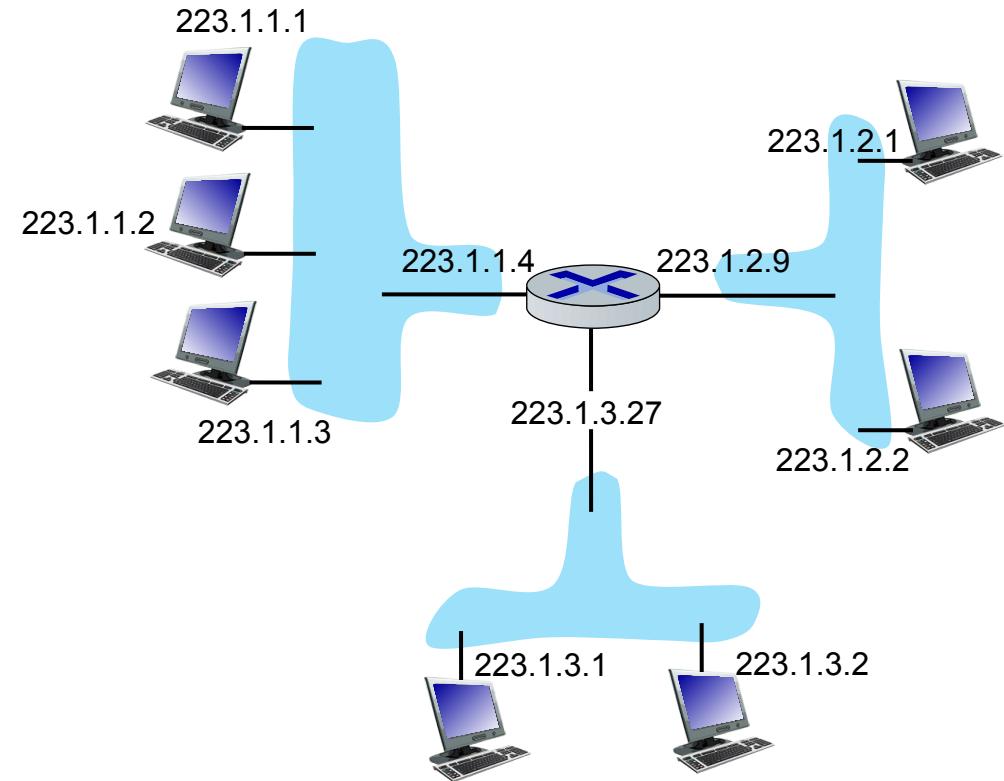
- datagram format
- addressing and subnets
- network address translation
- IPv6

4.4 Generalized Forwarding, SDN

- match + action
- OpenFlow: match + action in action

IP addressing: introduction

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



dotted-decimal IP address notation:

223.1.1.1 = $\begin{array}{cccc} 11011111 & 00000001 & 00000001 & 00000001 \end{array}$

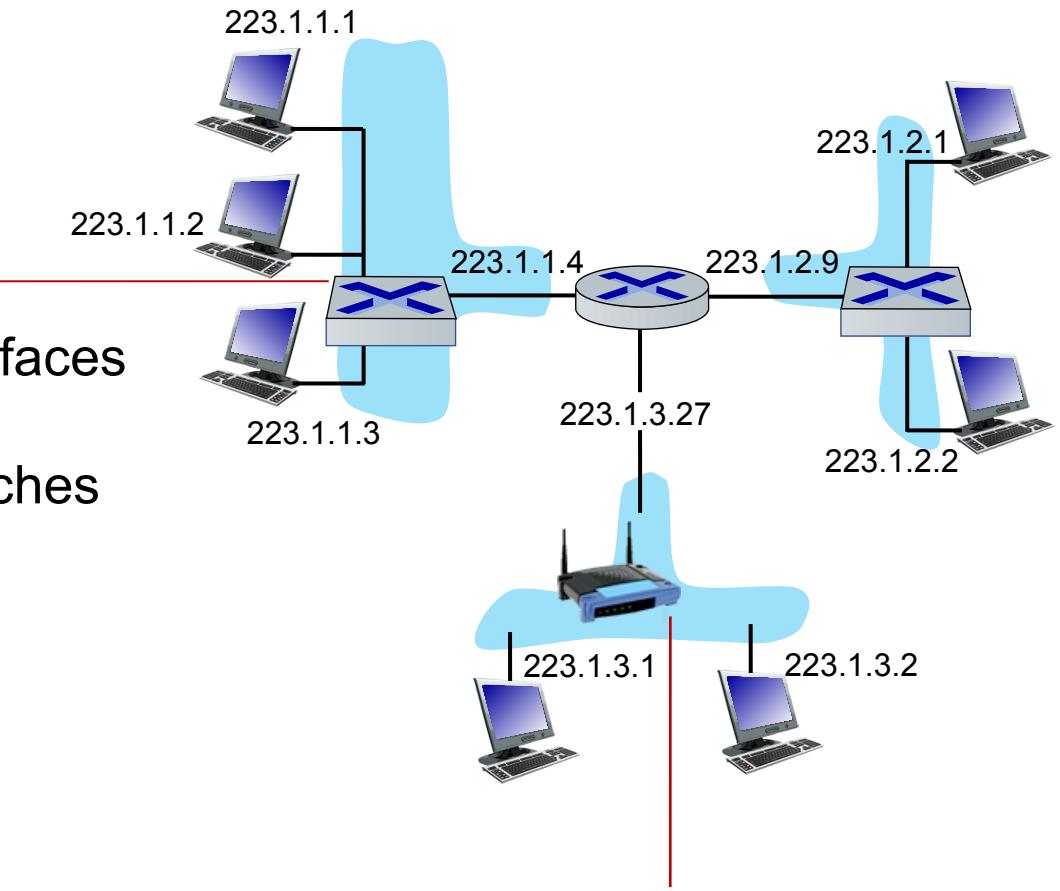
IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapters 6, 7

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

A: wired Ethernet interfaces connected by Ethernet switches

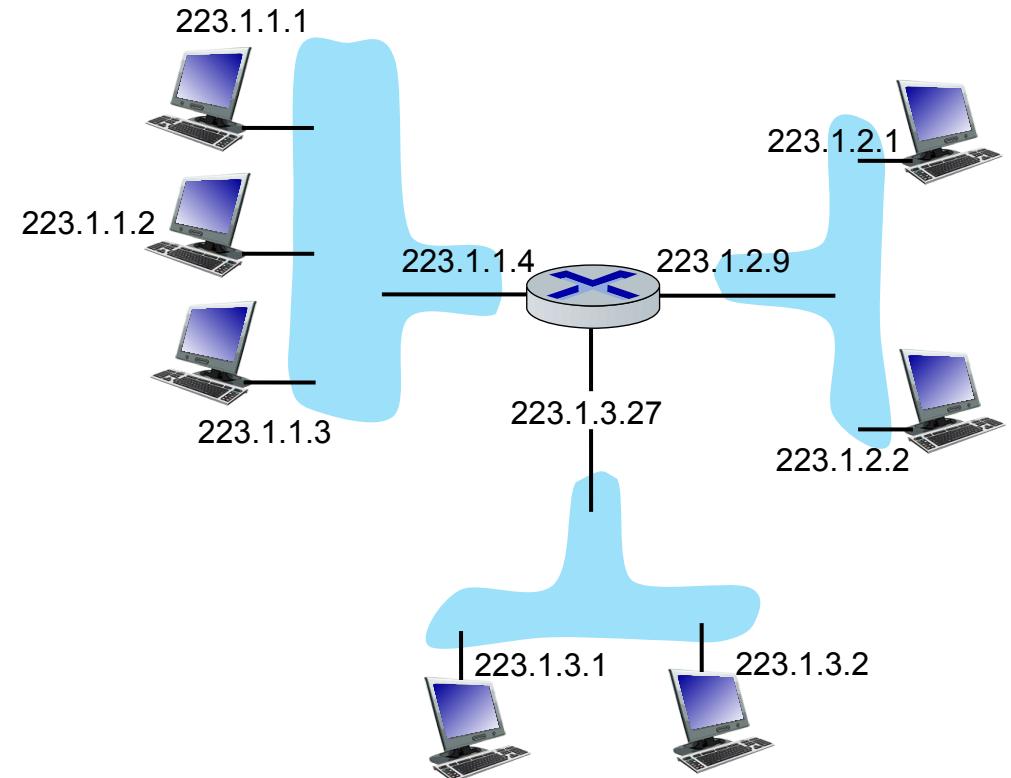


A: wireless WiFi interfaces connected by WiFi base station

Subnets

■ What's a subnet ?

- device interfaces that can physically reach each other **without passing through an intervening router**



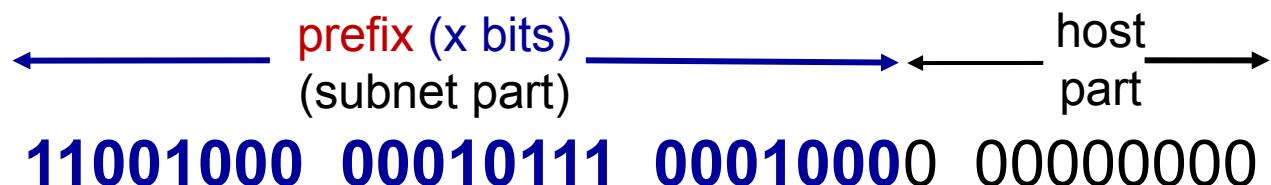
network consisting of 3 subnets

Subnets

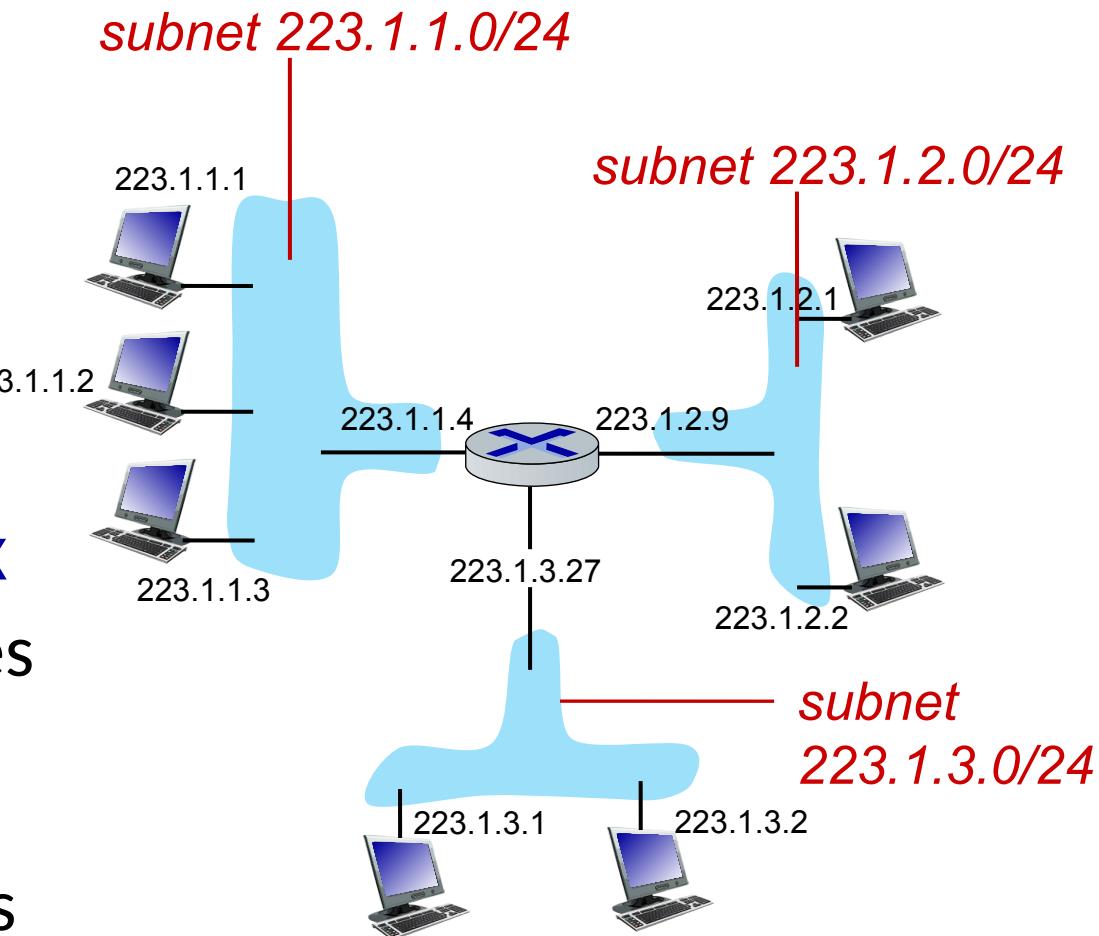
- CIDR: Classless InterDomain Routing

- IP addresses have structure: a.b.c.d/x

- prefix: **subnet part/network part**: devices in subnet have same **x** most significant bits (msb.)
- **host part**: remaining least significant bits



200.23.16.0/23



network consisting of 3 subnets

Subnet mask vs. prefix a.b.c.d/x

- A subnet mask is a **bitmask** when applied by a **bitwise AND** operation to any **IP address** in the network, yields the routing **prefix** / subnet address

prefix = IP address && bitmask

- 198.51.100.20/24 has the subnet mask 255.255.255.0 (24 msb. are 1's)

11000110 00110011 01100100 00010100

⌚ 198.51.100.20 (IP address)

11111111 11111111 11111111 00000000

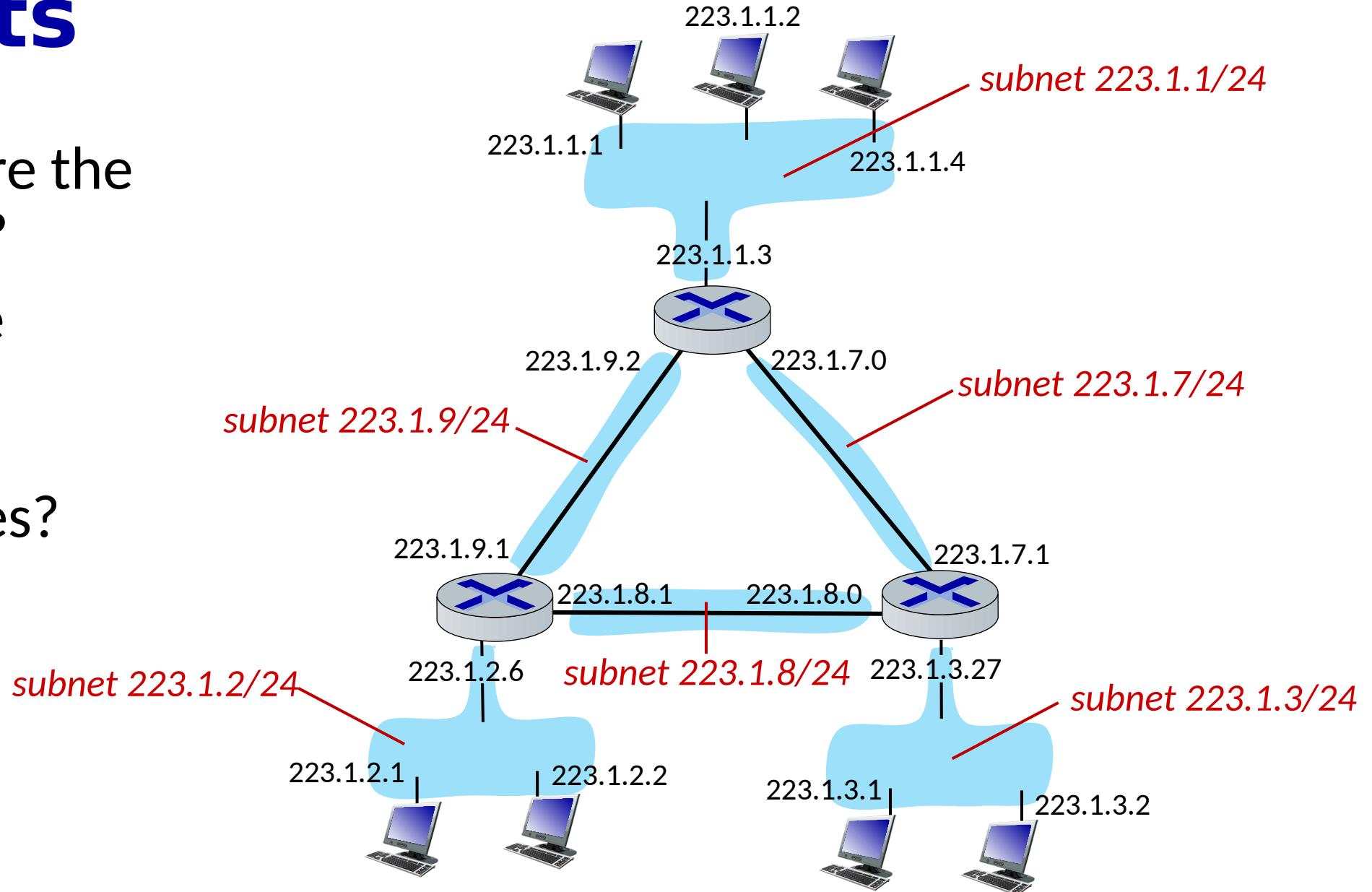
⌚ 255.255.255.0 (subnet mask)

11000110 00110011 01100100 00000000

⌚ 198.51.100.20 (subnet address)

Subnets

- where are the subnets?
- what are the /24 subnet addresses?



How to get IP addresses?

That's **two** questions:

1. How does an *organization* get IP addresses for itself (network part of address)
2. How does a *host* get an IP address within its network (host part of address)?

1. How gets an organization IP addresses?

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

ISP's block 11001000 00010111 00010000 00000000 200.23.16.0/20

ISP can then allocate out its address space in 8 blocks:

Organization 0 11001000 00010111 00010000 00000000 200.23.16.0/23

Organization 1 11001000 00010111 00010010 00000000 200.23.18.0/23

Organization 2 11001000 00010111 00010100 00000000 200.23.20.0/23

...

.....

.....

.....

Organization 7 11001000 00010111 00011110 00000000 200.23.30.0/23

2. How gets a host an IP address?

1. Manually by system admin in config file (e.g., /etc/rc.config in UNIX)
2. Dynamically from a server: **DHCP: Dynamic Host Configuration Protocol**
 - “plug-and-play”

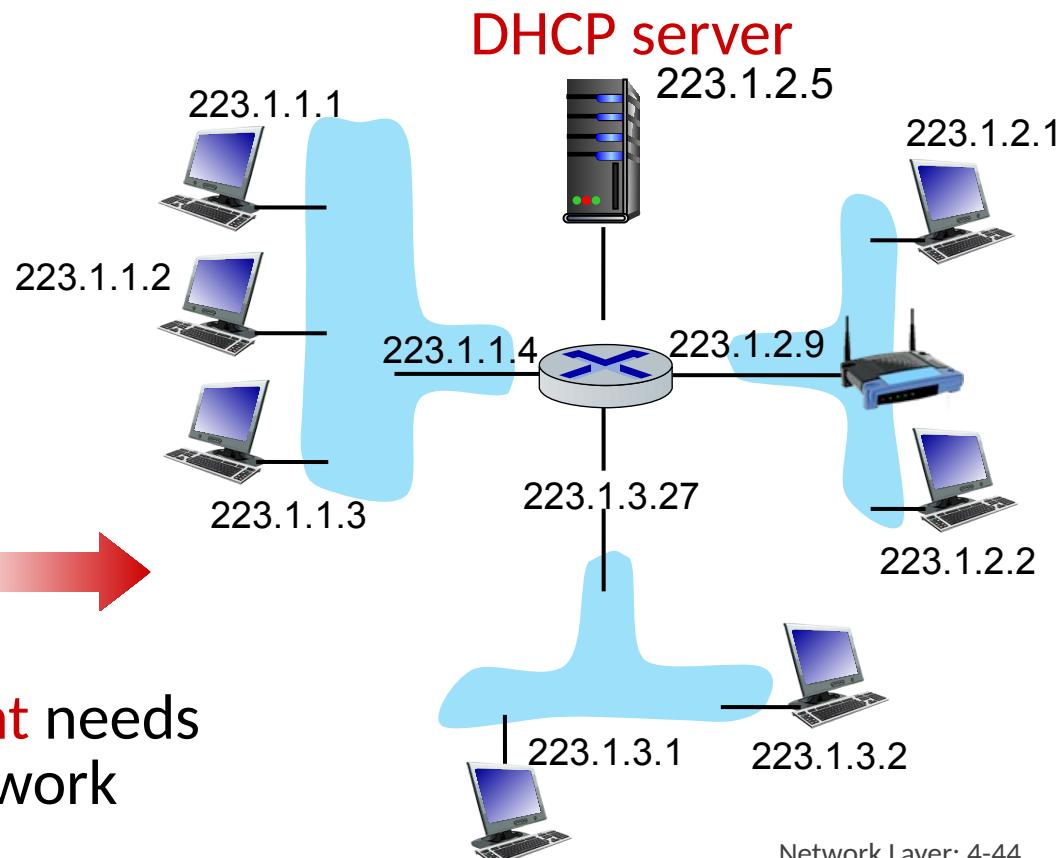
DHCP client-server scenario

goal: host *dynamically* obtains IP address from network server when it “joins” network

- allows reuse of addresses (only hold address while connected/on)

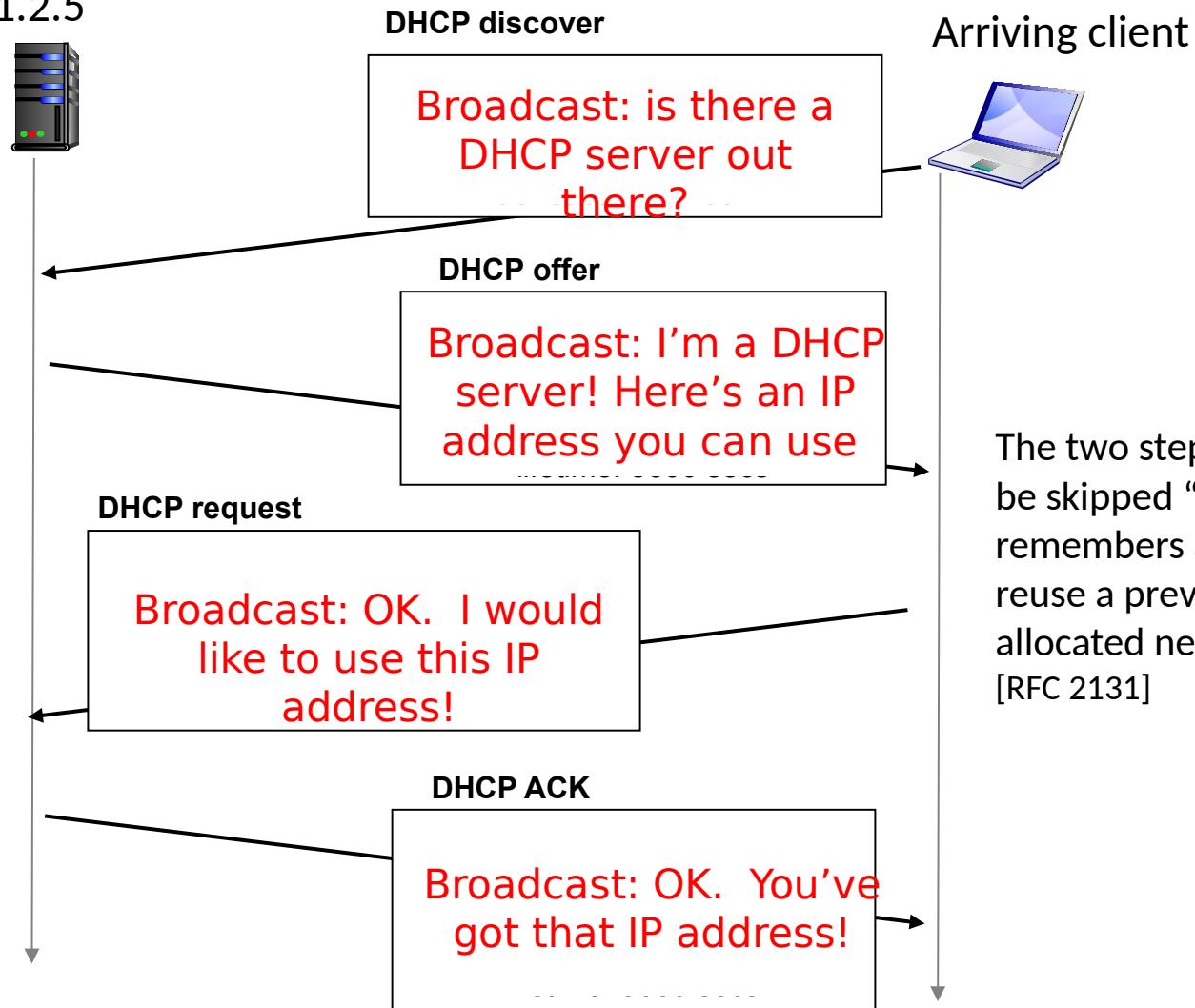


arriving **DHCP client** needs address in this network

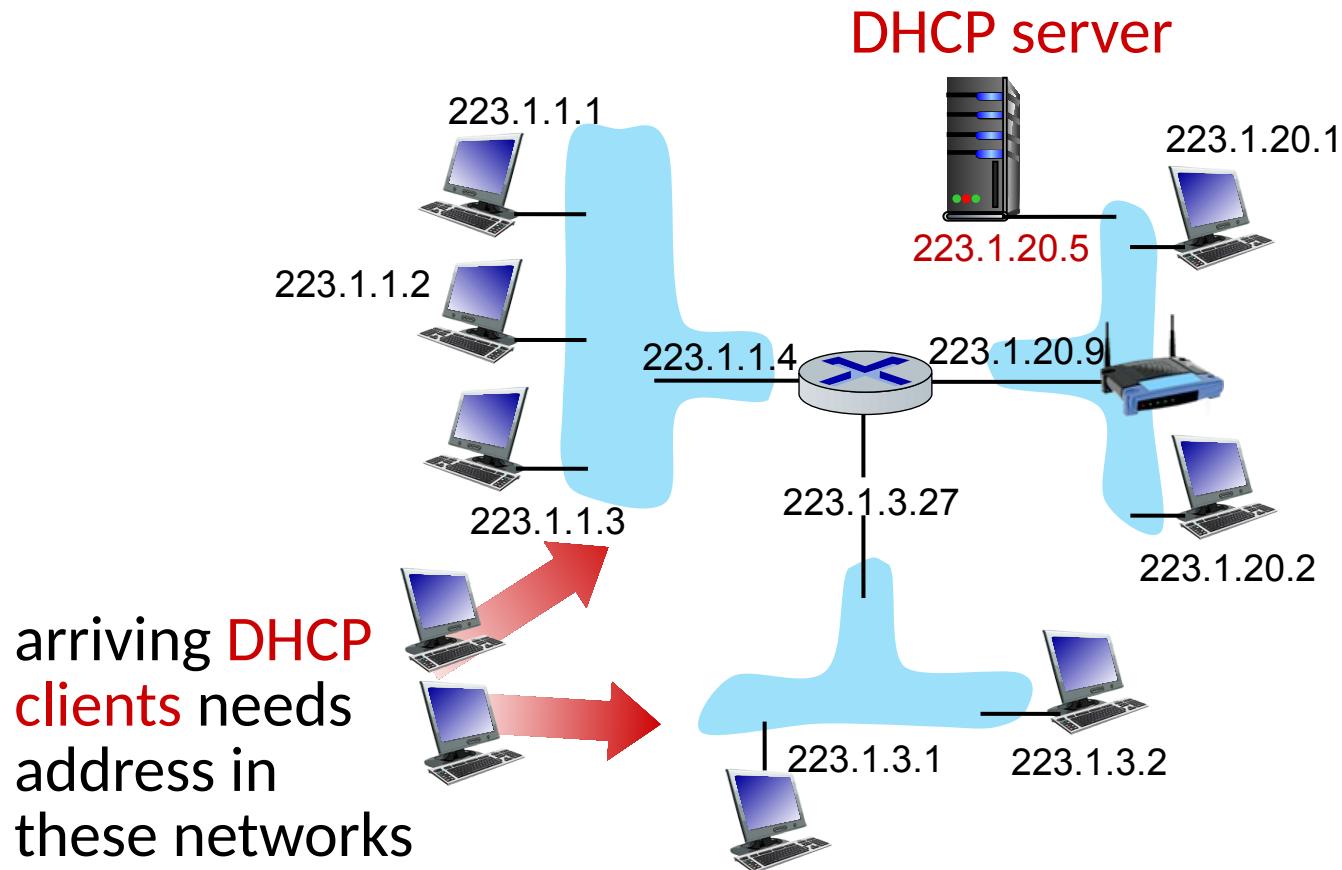


DHCP client-server scenario

DHCP server: 223.1.2.5



DHCP router relay agent



DHCP server located outside router serving all subnets to which router is attached:

- Router configured as a DHCP relay agent (ip helper-address 223.1.20.5)
- Router forward DHCP requests and replies between client and DHCP server
- Router sets the gateway IP address (223.1.1.4) in **giaddr** field of the DHCP packet
- This allows DHCP server to identify which subnet the request originated

DHCP: more than IP addresses

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

- address of gateway router for client
- name and IP address of DNS sever
- subnet mask (indicating network versus host portion of address)
- IP address lease time

IP addressing: last words ...

Q: how does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

<http://www.icann.org/>

- allocates IP addresses, through 5 regional registries (RRs) (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management

Q: are there enough 32-bit IP addresses?

- ICANN allocated last chunk of IPv4 addresses to RRs in 2011
- NAT (next) helps IPv4 address space exhaustion

"Who the hell knew how much address space we needed?" Vint Cerf (reflecting on decision to make IPv4 address 32 bits long)

Network layer: “data plane” roadmap

4.1 Network layer: overview

- data plane
- control plane

4.2 What's inside a router

- input ports, forwarding,
- switching, output ports, scheduling

4.3 The Internet Protocol (IP)

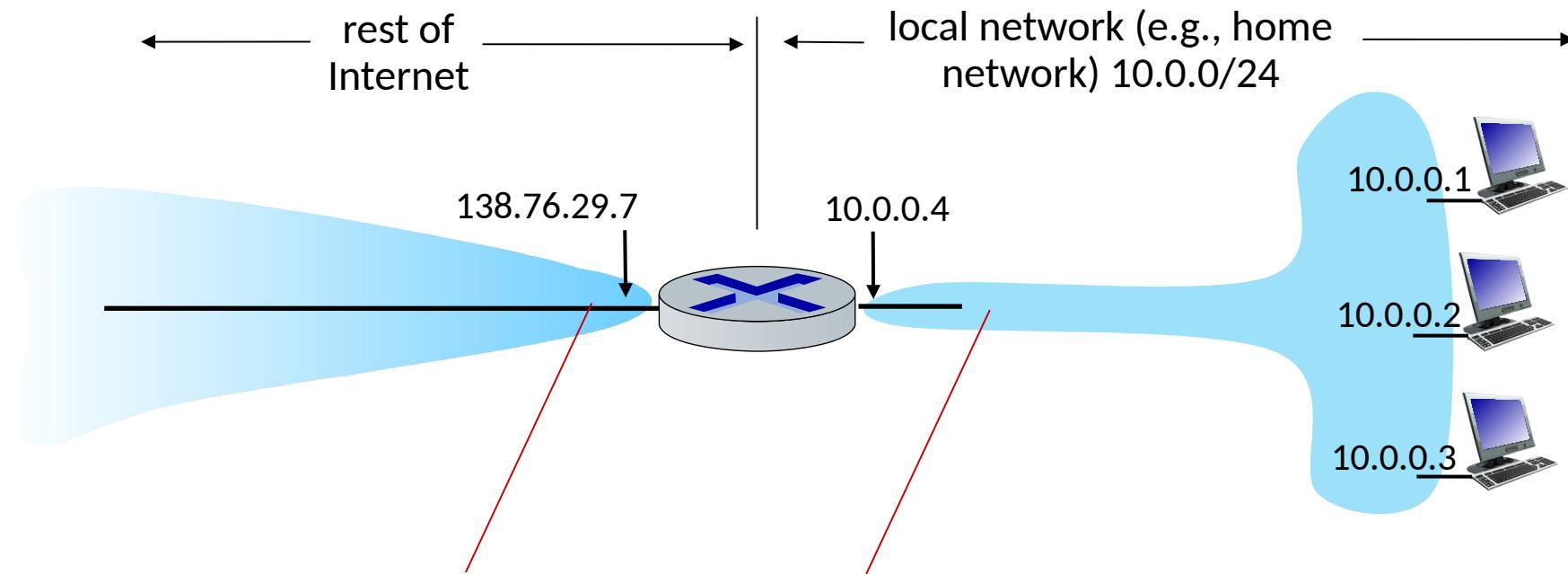
- datagram format
- addressing
- NAT: network address translation
- IPv6

4.4 Generalized Forwarding, SDN

- match + action
- OpenFlow: match + action in action

NAT: network address translation

NAT: All devices in local network share just **one** IPv4 address as far as outside world is concerned



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

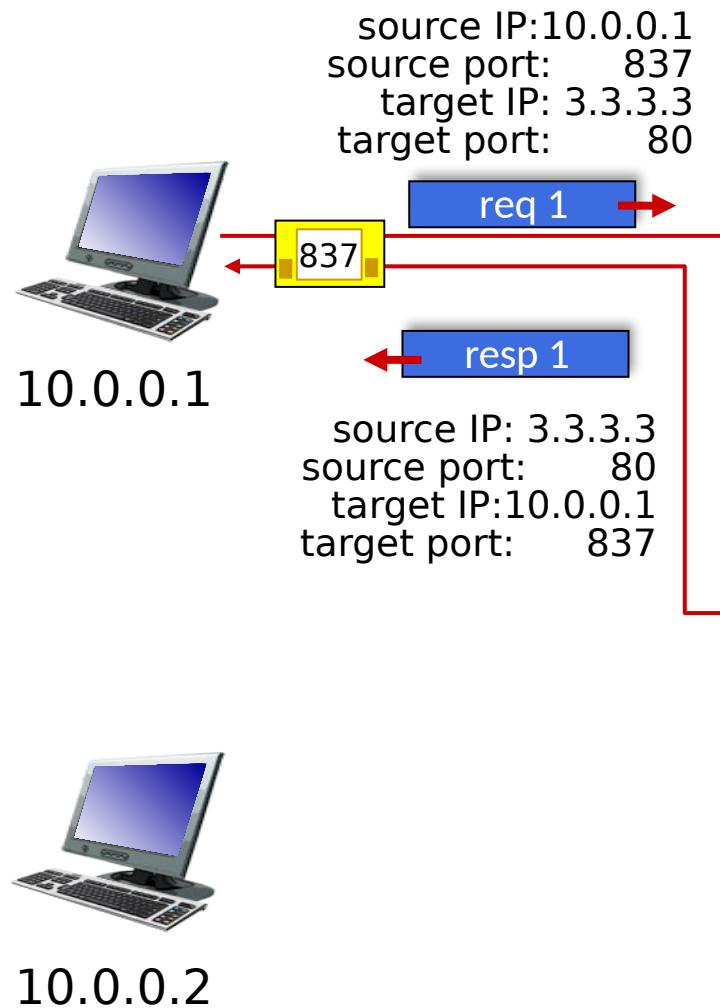
Datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

NAT: network address translation

implementation: NAT router must (transparently):

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

Clients in a local network (LAN)



Network address translation (NAT)

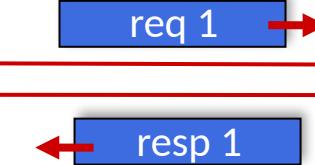
Private network side		Public network side	
Source IP	Source Port	Target IP	NAT Port
10.0.0.1	837	3.3.3.3	267

Remote server

NAT router

1. changes packet's source address from 10.0.0.1:837 to 4.2.1.5:267
2. updates table

source IP: 4.2.1.5
source port: 267
target IP: 3.3.3.3
target port: 80



source IP: 3.3.3.3
source port: 80
target IP: 4.2.1.5
target port: 267

3.3.3.3

Clients in a local network (LAN)



10.0.0.1



10.0.0.2

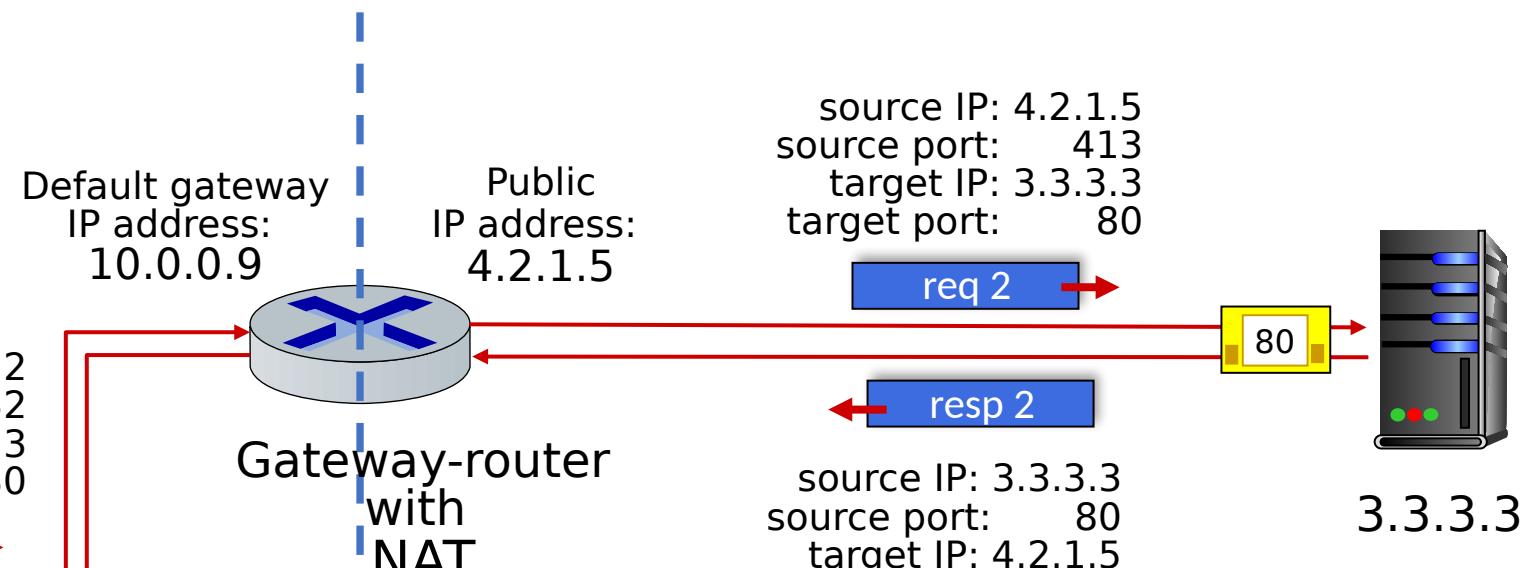
Network address translation (NAT)

Private network side		Public network side	
Source IP	Source Port	Target IP	NAT Port
10.0.0.1	837	3.3.3.3	267
10.0.0.2	932	3.3.3.3	413

Remote server

NAT router

1. changes packet's source address from 10.0.0.2:932 to 4.2.1.5:413
2. updates table



NAT: network address translation

- all devices in local network have 32-bit addresses in a “private” IP address space (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 prefixes) that can only be used in local network
- advantages:
 - just **one** IP address needed from provider ISP for ***all*** devices
 - can change addresses of host in local network without notifying outside world
 - can change ISP without changing addresses of devices in local network
 - security: devices inside local net not directly addressable, visible by outside world

Network layer: “data plane” roadmap

4.1 Network layer: overview

- data plane
- control plane

4.2 What's inside a router

- input ports, forwarding,
- switching, output ports, scheduling

4.3 The Internet Protocol (IP)

- datagram format
- addressing
- NAT: network address translation
- IPv6

4.4 Generalized Forwarding, SDN

- match + action
- OpenFlow: match + action in action

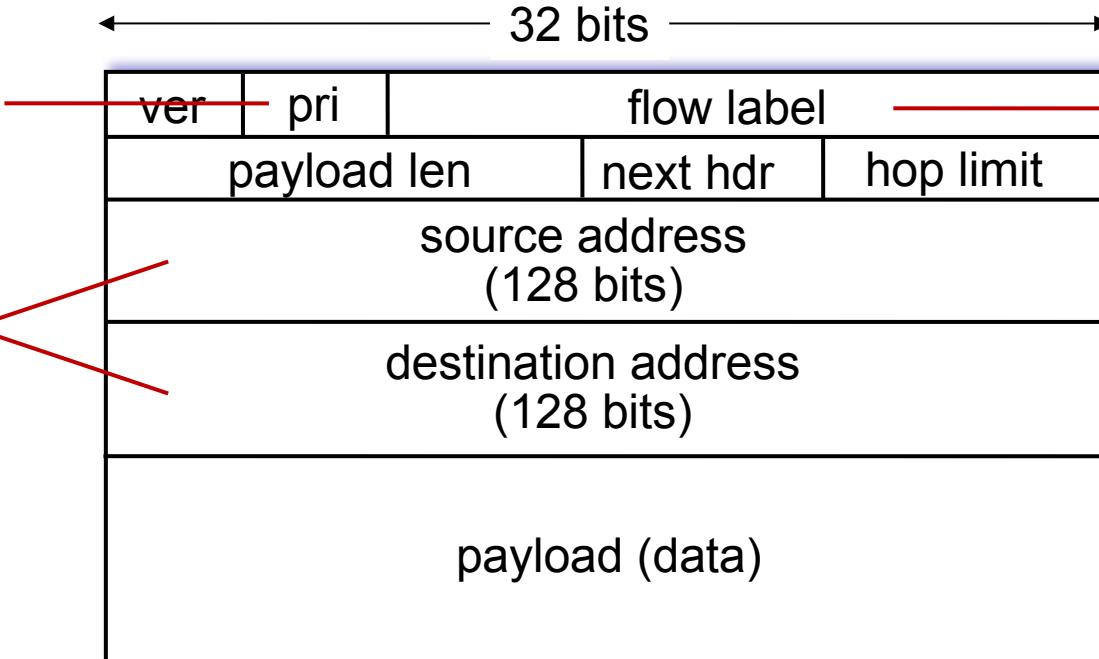
IPv6: motivation

- **initial motivation:** 32-bit IPv4 address space would be completely allocated
- additional motivation:
 - speed processing/forwarding: 40-byte fixed length header
 - enable different network-layer treatment of “flows”

IPv6 datagram format

priority: identify priority among datagrams in flow

128-bit
IPv6 addresses



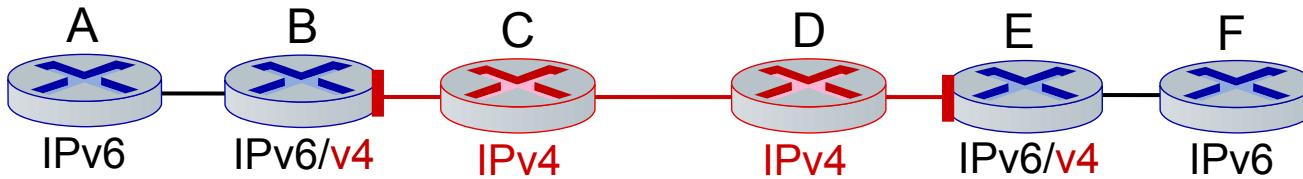
flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4):

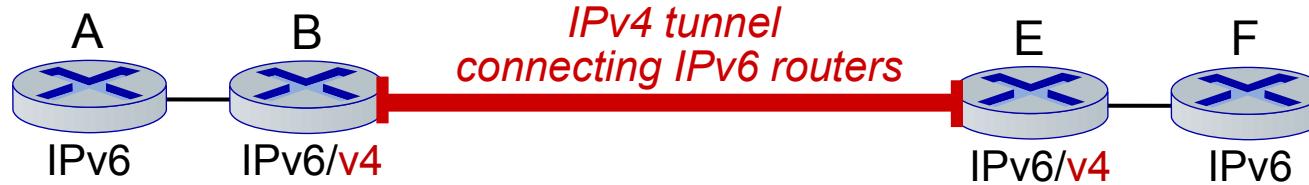
- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

Tunneling

physical view:

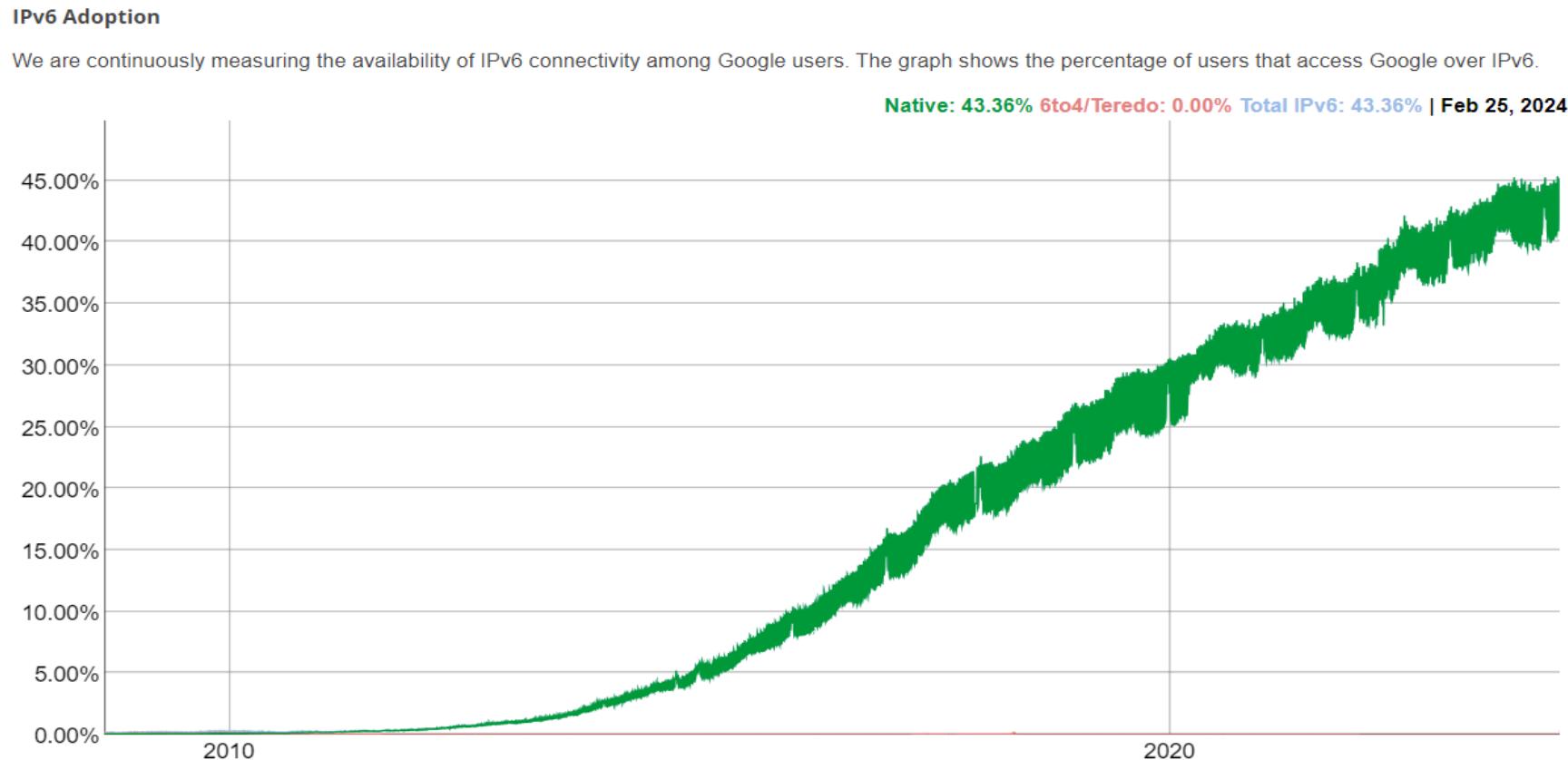


logical view:



IPv6: adoption

- Google¹: ~ 40% of clients access services via IPv6 (2023)
- NIST: 1/3 of all US government domains are IPv6 capable



Network layer: “data plane” roadmap

4.1 Network layer: overview

- data plane
- control plane

4.2 What's inside a router

- input ports, forwarding,
- switching, output ports, scheduling

4.3 IP: the Internet Protocol

- datagram format
- addressing
- network address translation
- IPv6



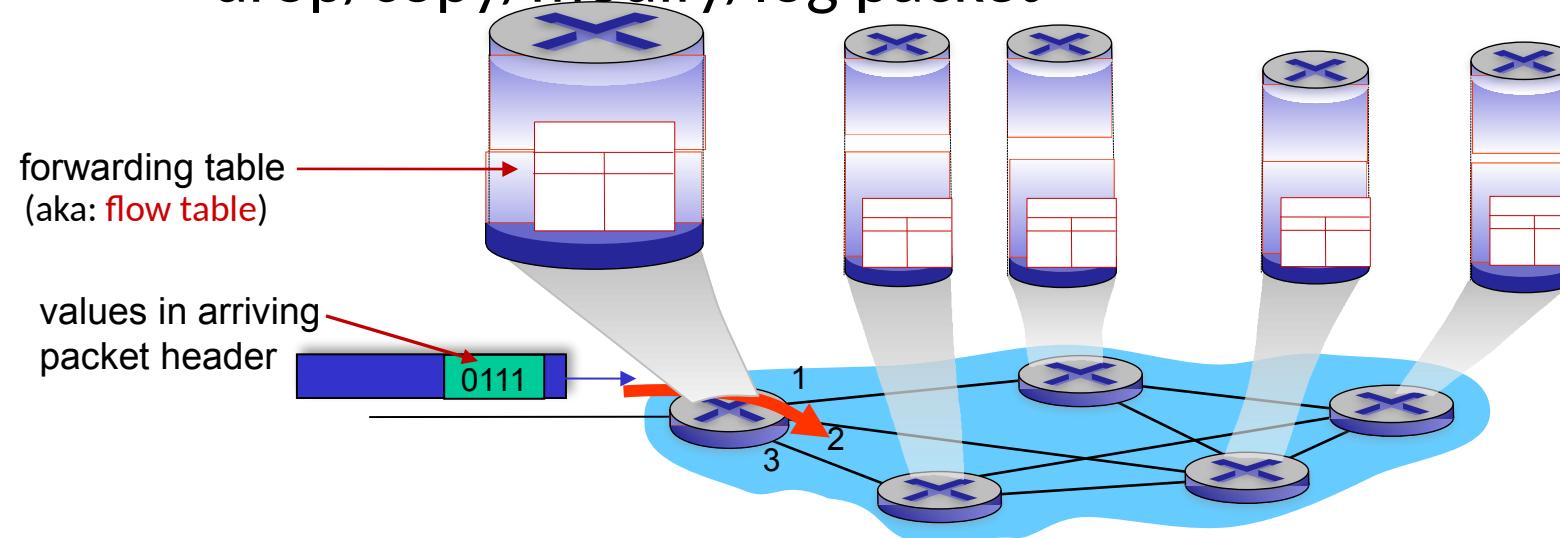
4.4 Generalized Forwarding, SDN

- match + action
- OpenFlow: match + action in action

Generalized forwarding: match plus action

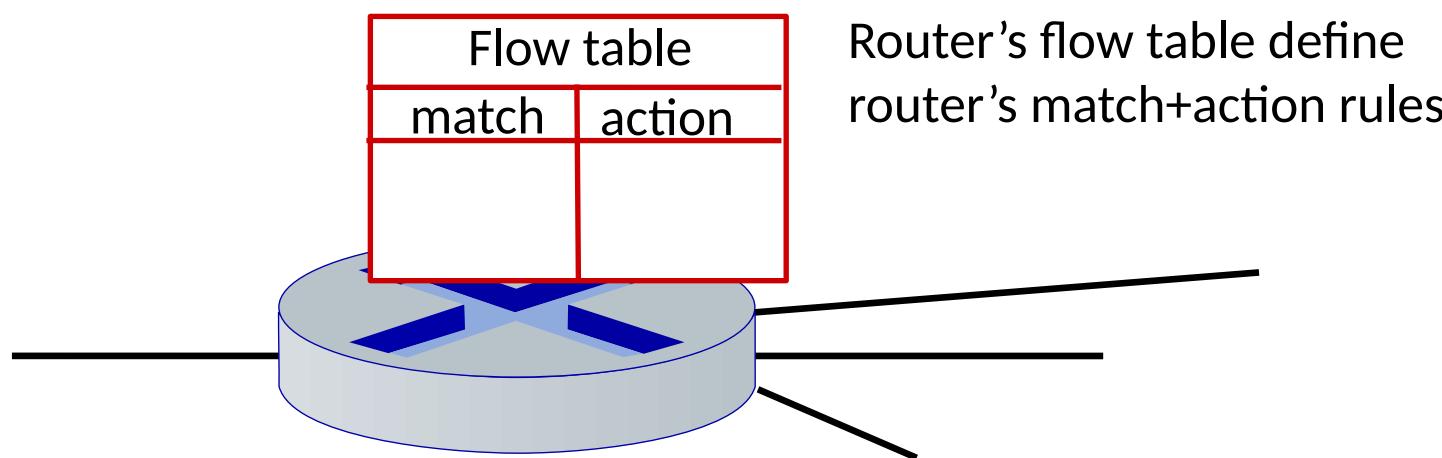
Review: each router contains a **forwarding table** (aka: **flow table**)

- “**match plus action**” abstraction: match bits in arriving packet, take action
 - *destination-based forwarding*: forward based on dest. IP address
 - *generalized forwarding*:
 - many header fields can determine action
 - many action possible:
drop/copy/modify/log packet

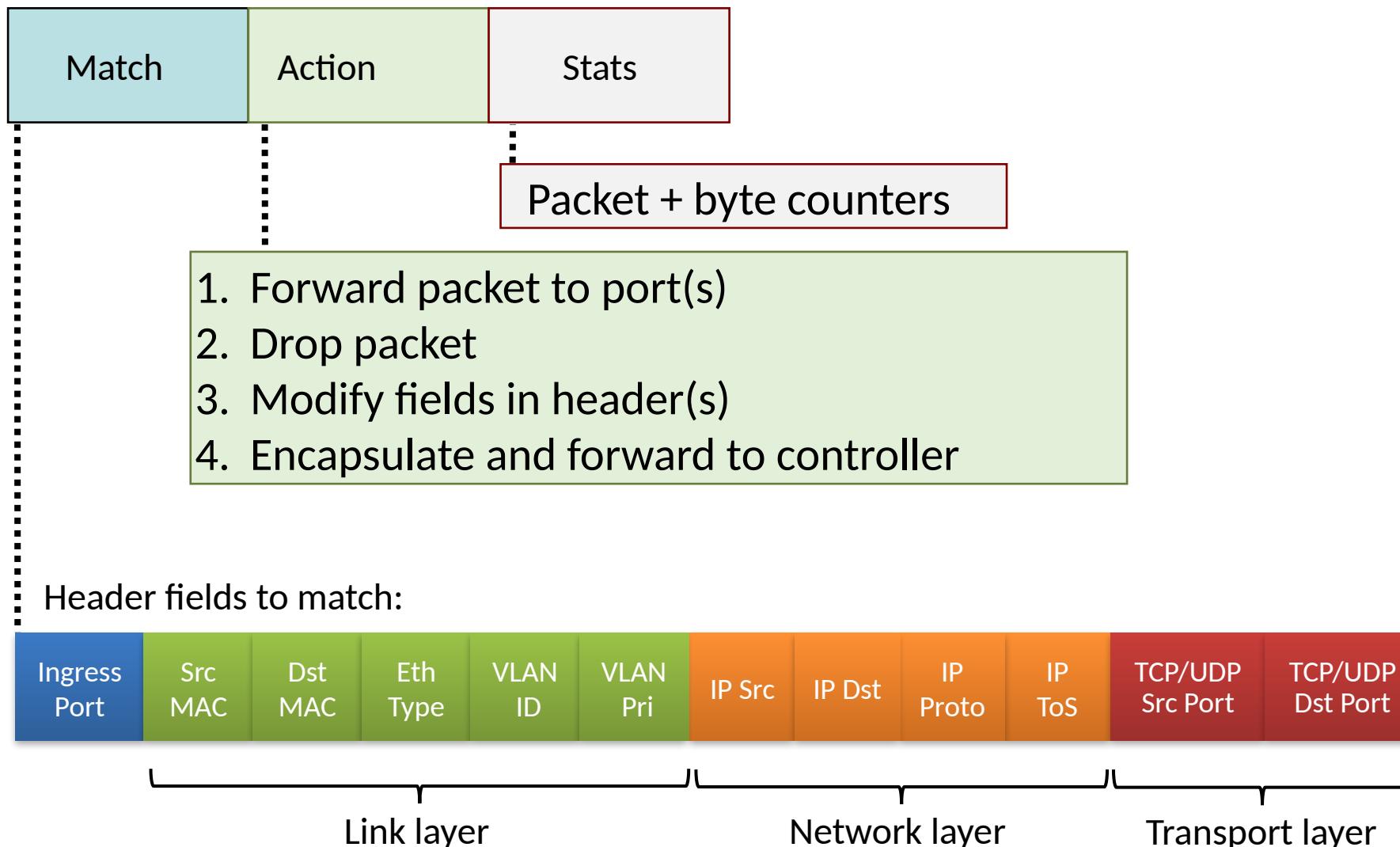


Flow table abstraction

- **flow**: defined by header field values (in link-, network-, transport-layer fields)
- **generalized forwarding**: simple packet-handling rules
 - **match**: pattern values in packet header fields
 - **actions**: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
 - **priority**: disambiguate overlapping patterns
 - **counters**: #bytes and #packets



OpenFlow: flow table entries



OpenFlow: examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

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

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

OpenFlow abstraction

- **match+action:** abstraction unifies different kinds of devices

Router

- *match:* longest destination IP prefix
- *action:* forward out a link

Switch

- *match:* destination MAC address
- *action:* forward or flood

Firewall

- *match:* IP addresses and TCP/UDP port numbers
- *action:* permit or deny

NAT

- *match:* IP address and port
- *action:* rewrite address and port

Chapter 4: done!

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding, SDN



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

Answer: by the control plane (next chapter)

Chapter 5

Network Layer:

Control Plane

A note on the use of these PowerPoint slides:

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

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

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

EIGHTH EDITION

James F. Kurose • Keith W. Ross



*Computer Networking: A
Top-Down Approach*

8th edition

Jim Kurose, Keith Ross
Pearson, 2020

Network layer control plane: our goals

- understand principles behind network control plane:
 - traditional routing algorithms
 - SDN controllers
- instantiation, implementation in the Internet:
 - OSPF, BGP
 - OpenFlow, ODL and ONOS controllers
 - Internet Control Message Protocol: ICMP

5 Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

5.2.1 Link state

5.2.2 Distance vector

5.3 Intra-ISP routing: OSPF

5.4 Routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Network-layer functions

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

data plane

control plane

Two approaches to routing:

- Distributed routing (traditional)
- Logically centralized routing (software defined networking)

5 Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

 5.2.1 Link state

 5.2.2 Distance vector

5.3 Intra-ISP routing: OSPF

5.4 routing among ISPs: BGP

5.5 SDN control plane

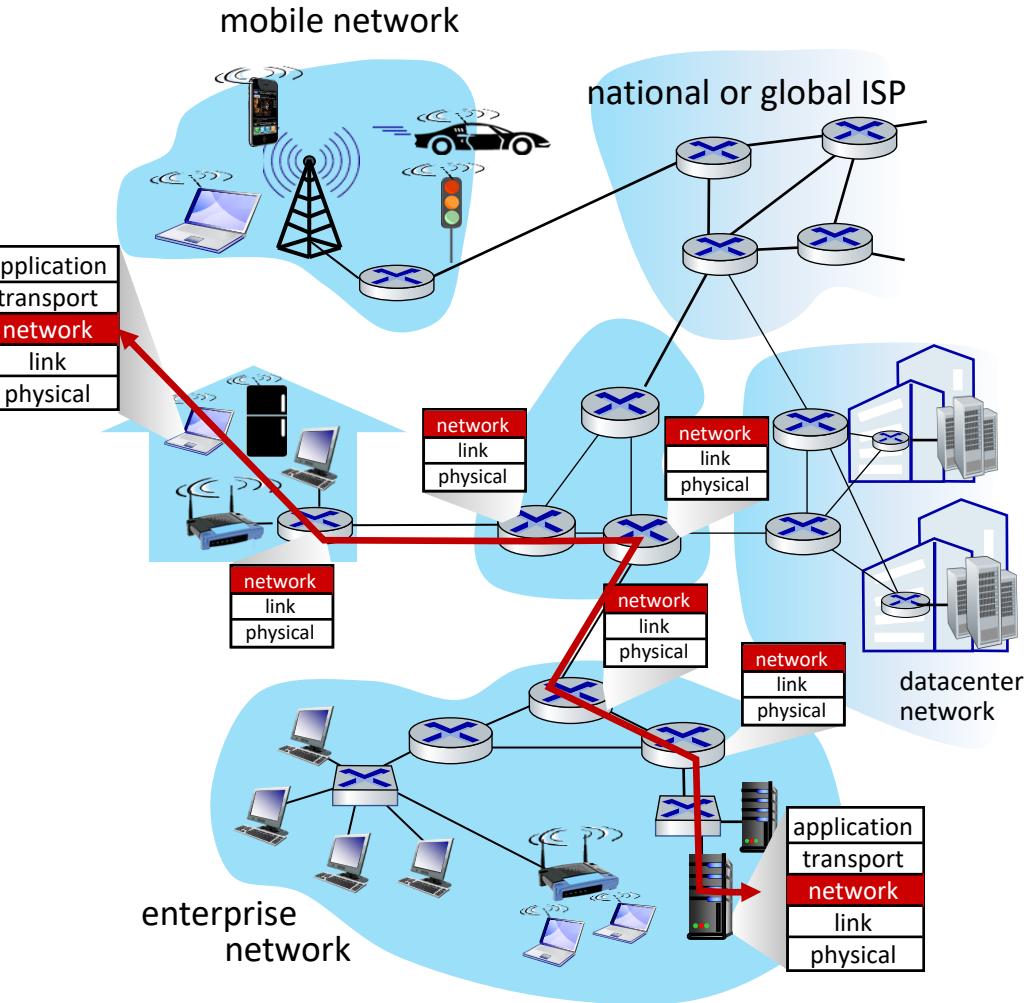
5.6 Internet Control Message Protocol

Routing algorithms

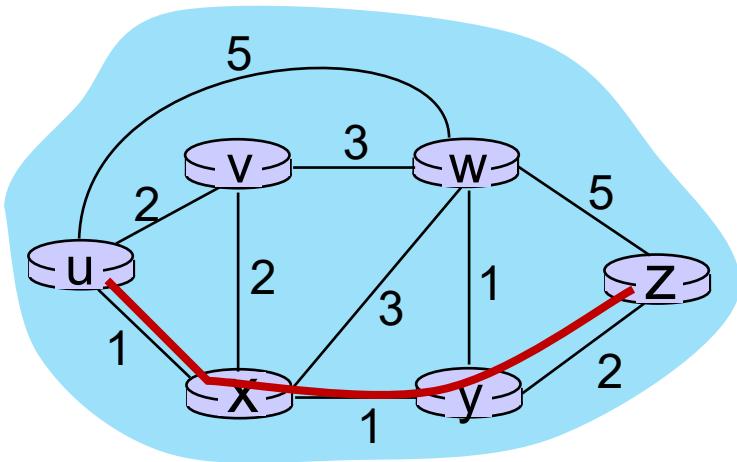
compute routing tables

Routing goal: determine “good” paths from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets traverse from given initial source host to final destination host
- **“good”:** least “cost”, “fastest”, “least congested”



Graph abstraction: link costs



graph: $G = (N, E)$

N : set of routers = { u, v, w, x, y, z }

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

$c_{a,b}$: cost of *direct* link connecting a and b

e.g., $c_{w,z} = 5, c_{u,z} = \infty$

cost defined by network operator:
could always be 1, or inversely related
to bandwidth, or inversely related to
congestion

Routing algorithm classification

“Global” algorithms: complete network topology

all routers have *complete* topology, link cost info

- “link state” algorithms

“Distributed” algorithms: partial network topology

iterative process of computation, exchange of info with neighbors

- routers initially only know link costs to attached neighbors
- “distance vector” algorithms

global or decentralized information?

5 Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

 5.2.1 Link state

 5.2.2 Distance vector

5.3 Intra-ISP routing: OSPF

5.4 routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Dijkstra's link-state routing algorithm

- **centralized:** global network topology, link costs known to *all* nodes
 - accomplished via “link state broadcast”
 - all nodes have same info
- Each node/router runs this algorithm
 - computes least cost paths from one node (“source”) to all other nodes
 - gives *routing table* for that router
- **iterative:** after k iterations, know least cost path to k destinations

notation

- $c_{x,y}$: direct distance (cost) from node x to y ; initially ∞ if not direct neighbors
- $d(v)$: distance (cost) from source to a node v
- $p(v)$: path vector - points to the adjacent node with the shortest distance to node v
- N : set of nodes

Dijkstra's link-state routing algorithm

Initialization:

```
N = {u}                                /* compute least cost path from u to all other nodes */  
for all nodes v  
    if v adjacent to u                /* u initially knows distance/costs only to direct neighbors */  
        then d[v] = cu,v  
    else d[v] = ∞
```

current node u = source node; add u to N

while N is not complete

for each neighbor *v* of *u*, not in N:

// shortest distance to *v* is unchanged or goes via *u* to *v*:

d[*v*] = min (d[*v*], d[*u*] + c_{*u,v*})

p[*v*] = p[*u*]

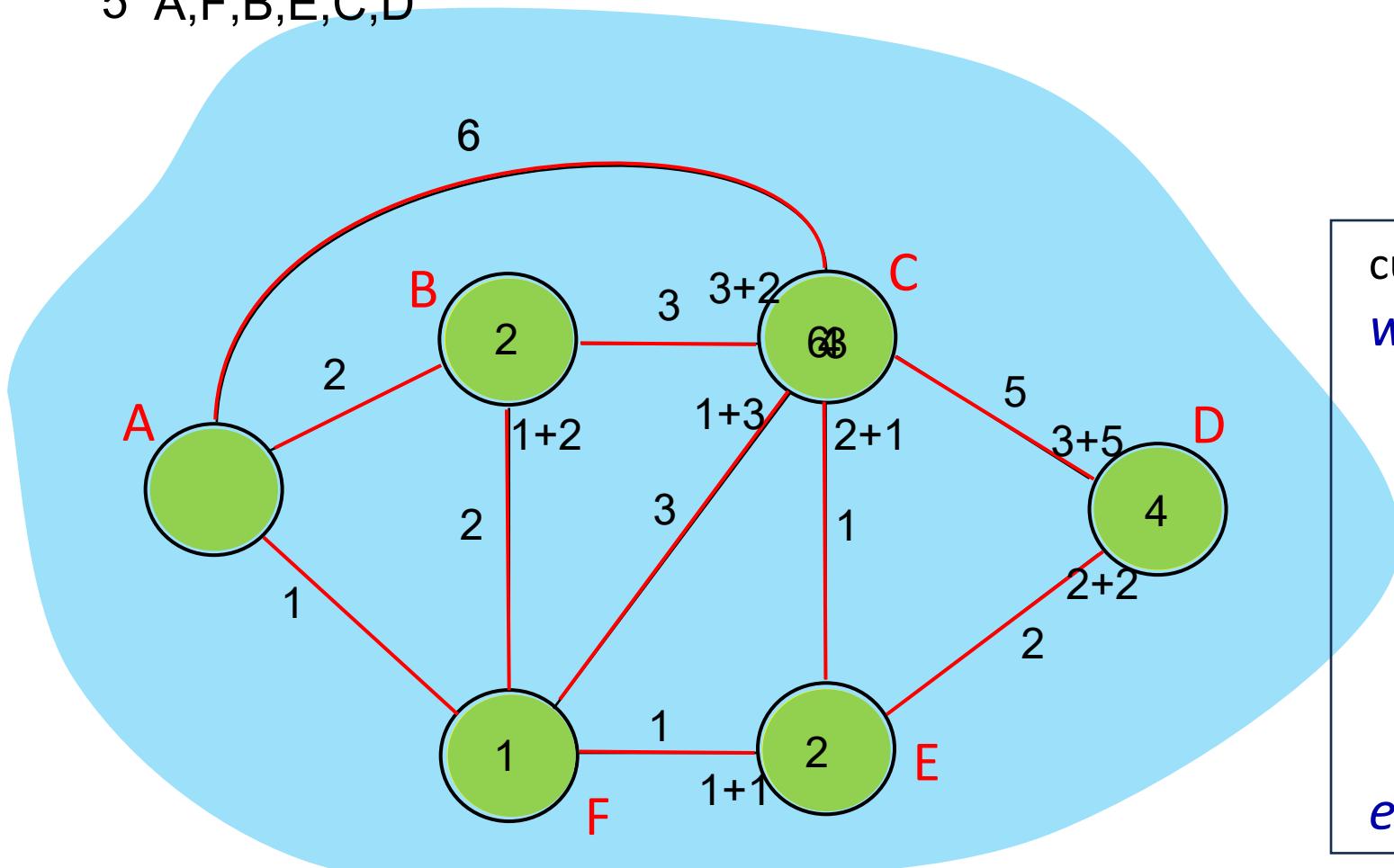
next

new current node u = unvisited node in N with shortest distance to current node u

add u to N

Step	N	d(B),p(B)	d(C),p(C)	d(D),p(D)	d(E),p(E)	d(F),p(F)
0	A	2, A	6, A			1, A
1	A,F	2, A	4, F		2, F	
2	A,F,B		4, F			
3	A,F,B,E		3, E	4, E		
4	A,F,B,E,C			4, E		
5	A,F,B,E,C,D					

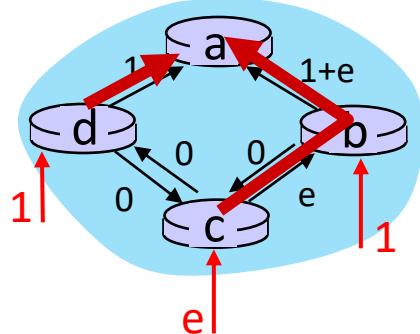
n	d(n)	p(n)
A	-	-
B	2	A
C	3	E
D	4	E
E	2	F
F	1	A



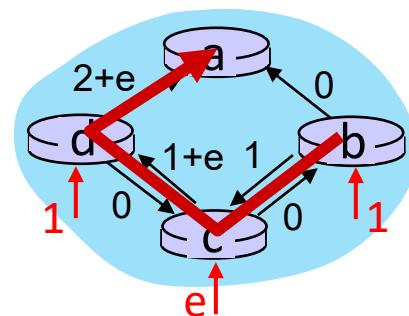
current node u = source node, add u to N
while N is not complete
 for each neighbor v to u , not in N
 $d[v] = \min (d[v], d[u] + c_{u,v})$
 $p[v] = p[u]$
 next
 let u = unvisited node in N with min.
 distance to current node u
 add u to N
end while

Dijkstra's algorithm: oscillations possible

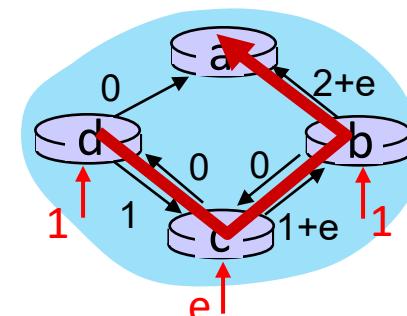
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
 - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
 - link costs are directional, and volume-dependent



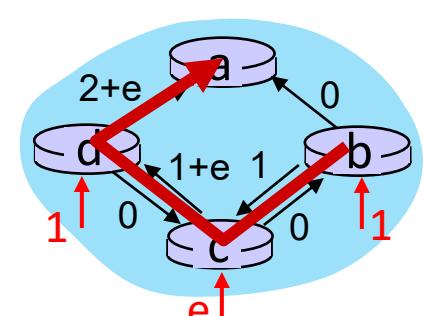
initially



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



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



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

Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

 5.2.1 Link state

 5.2.2 Distance vector

5.3 Intra-ISP routing: OSPF

5.4 Routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y .

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

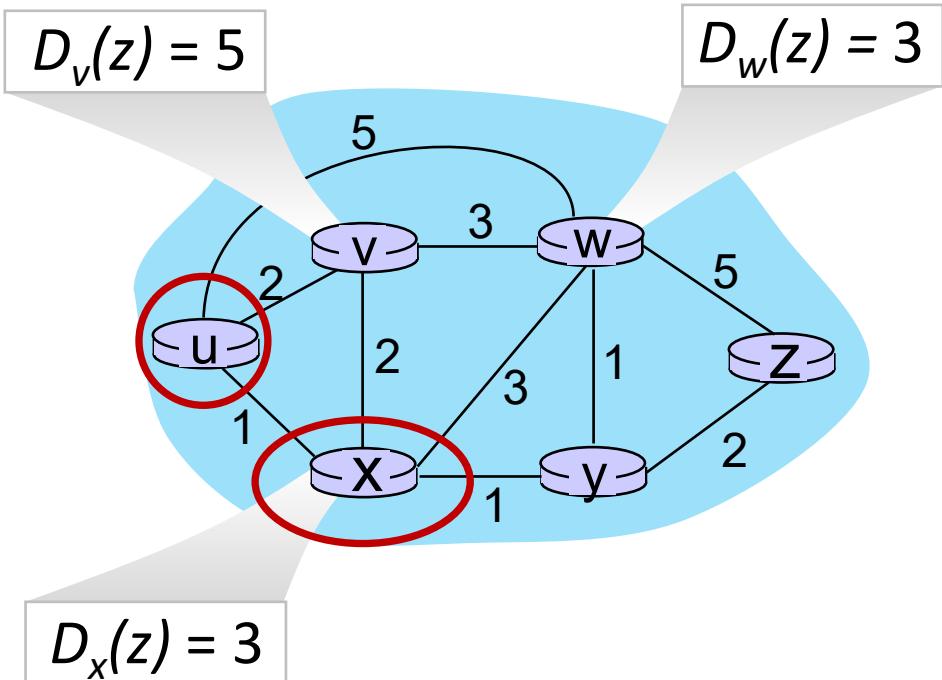
\min taken over all neighbors v of x

v 's estimated least-cost-path cost to y

direct cost of link from x to v

Bellman-Ford Example

Suppose that u 's neighboring nodes, x, v, w , know that for destination z :



Bellman-Ford equation says:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,x} + D_x(z), \\ &\quad c_{u,w} + D_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

node x is next hop on least-cost path to destination z

Distance vector algorithm

key idea:

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

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

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

Distance vector algorithm:

each node:

-
- ```
graph TD; A["wait for (change in local link cost or msg from neighbor)"] --> B["recompute DV estimates using DV received from neighbor"]; B --> C["if DV to any destination has changed, notify neighbors"]
```
- wait* for (change in local link cost or msg from neighbor)
  - recompute* DV estimates using DV received from neighbor
  - if DV to any destination has changed, *notify* neighbors

**iterative, asynchronous:** each local iteration caused by:

- local link cost change
- DV update message from neighbor

**distributed, self-stopping:** each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

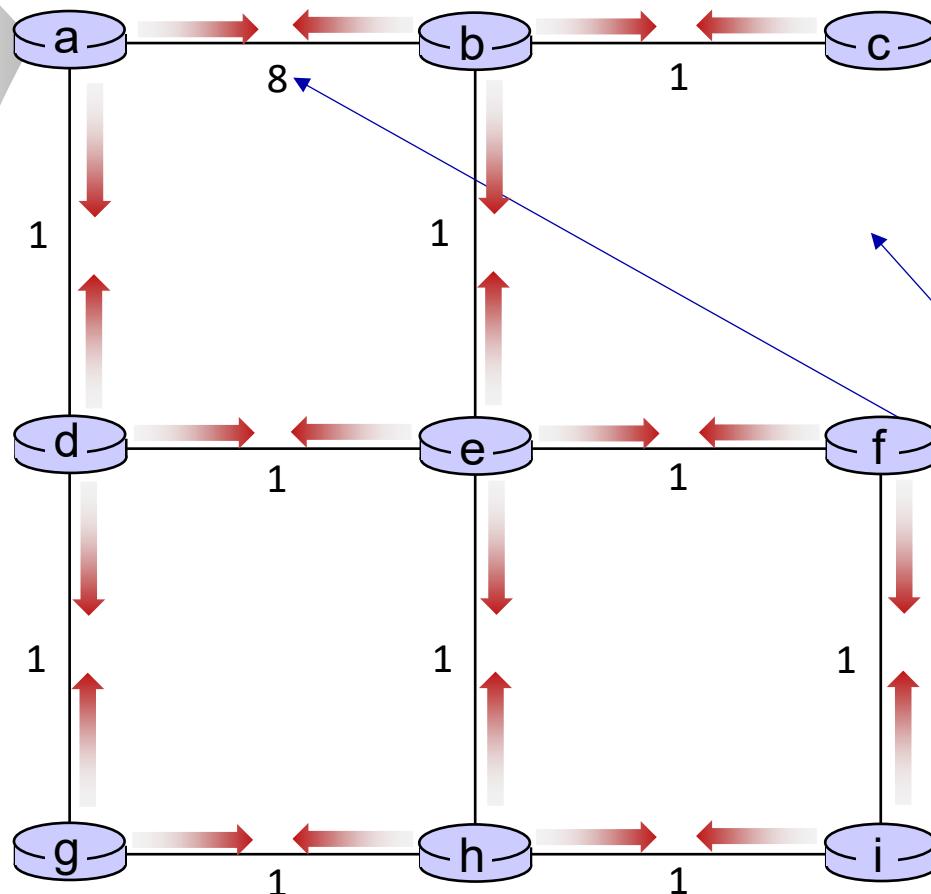
# Distance vector: example



$t=0$

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

| DV in a:          |
|-------------------|
| $D_a(a)=0$        |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



- A few asymmetries:
- missing link
  - larger cost

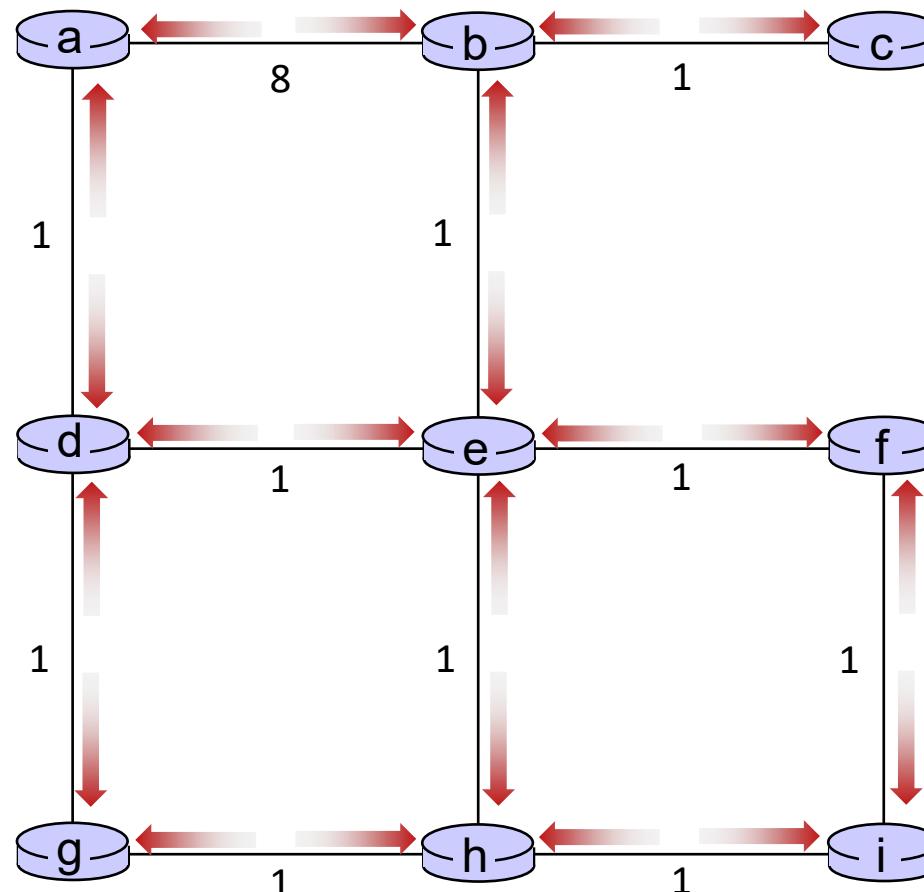
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



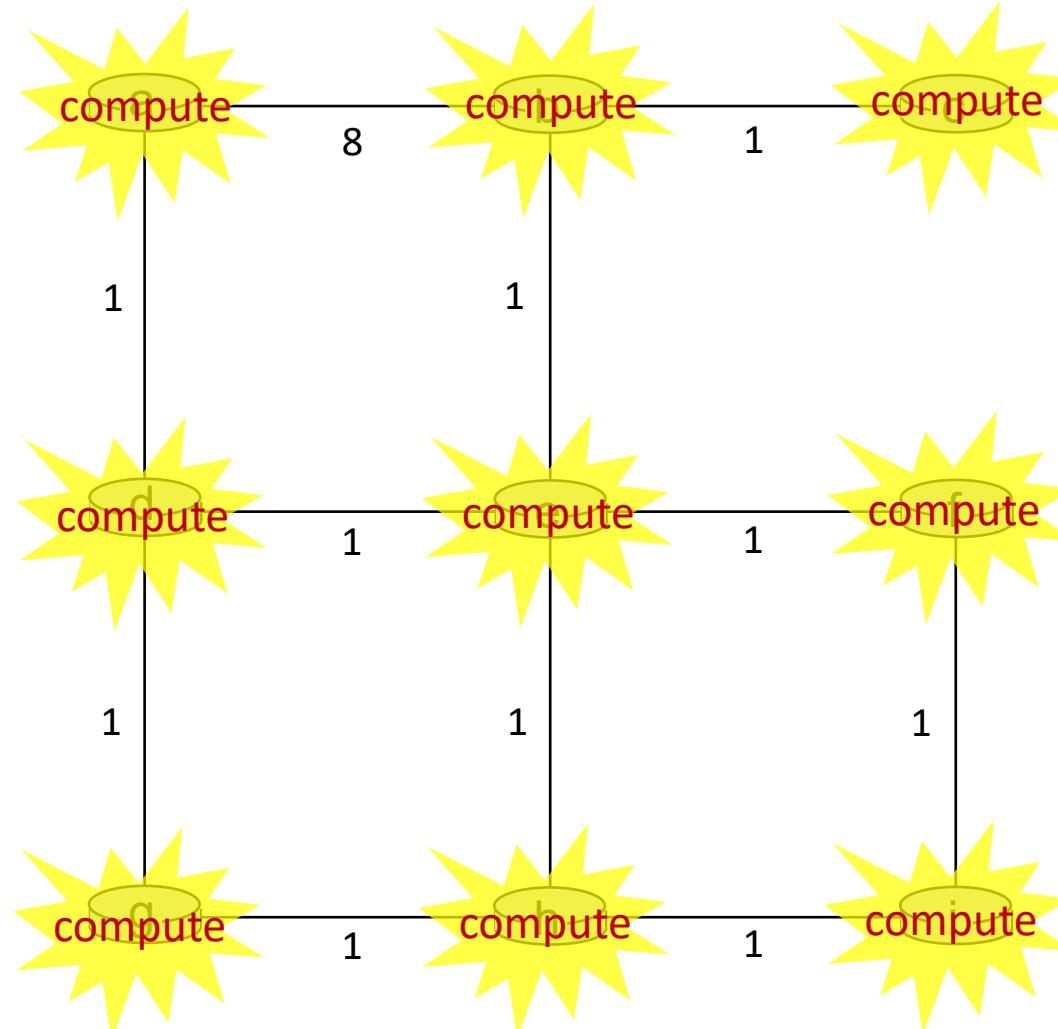
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



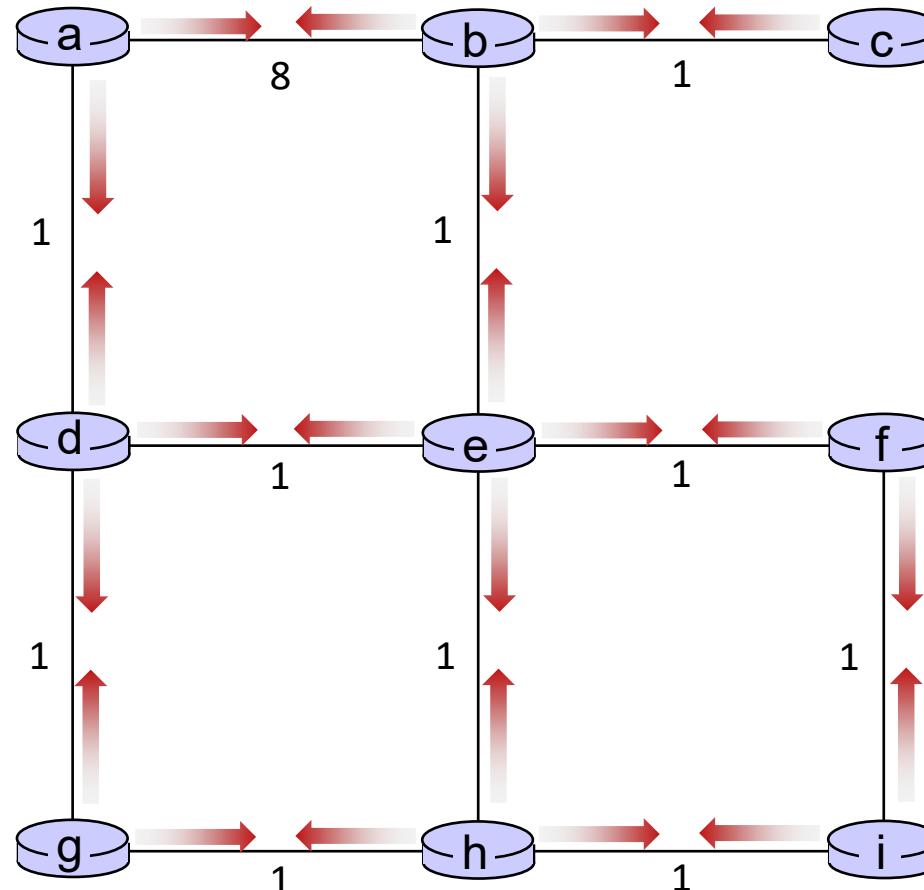
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



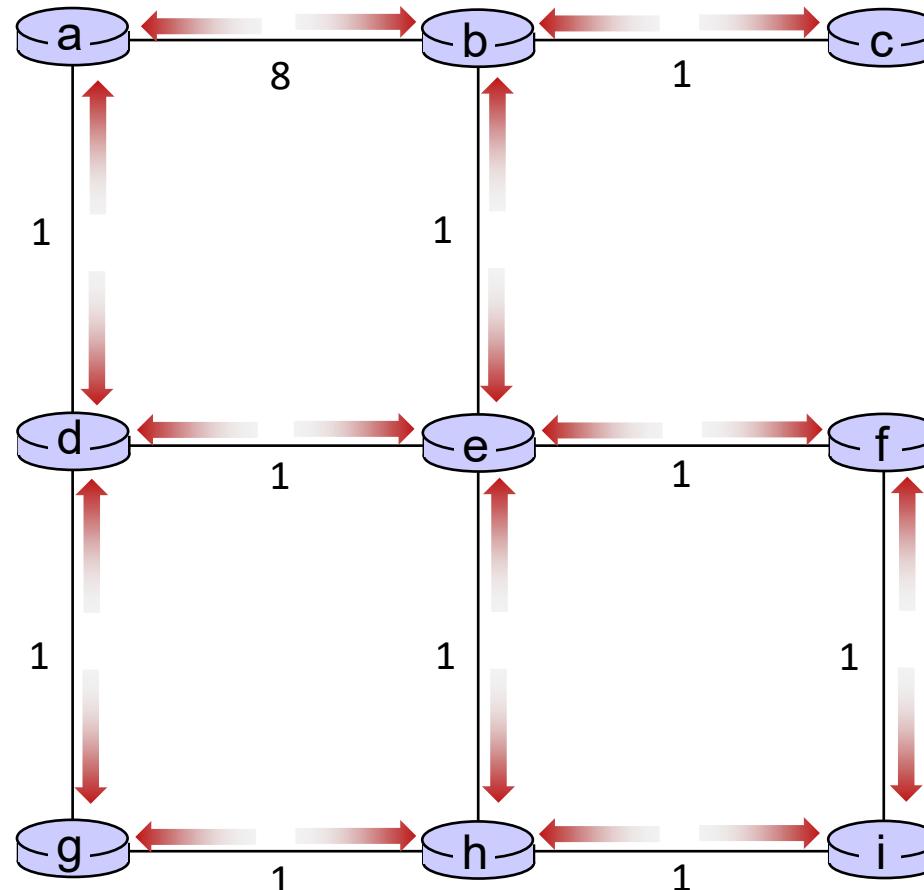
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



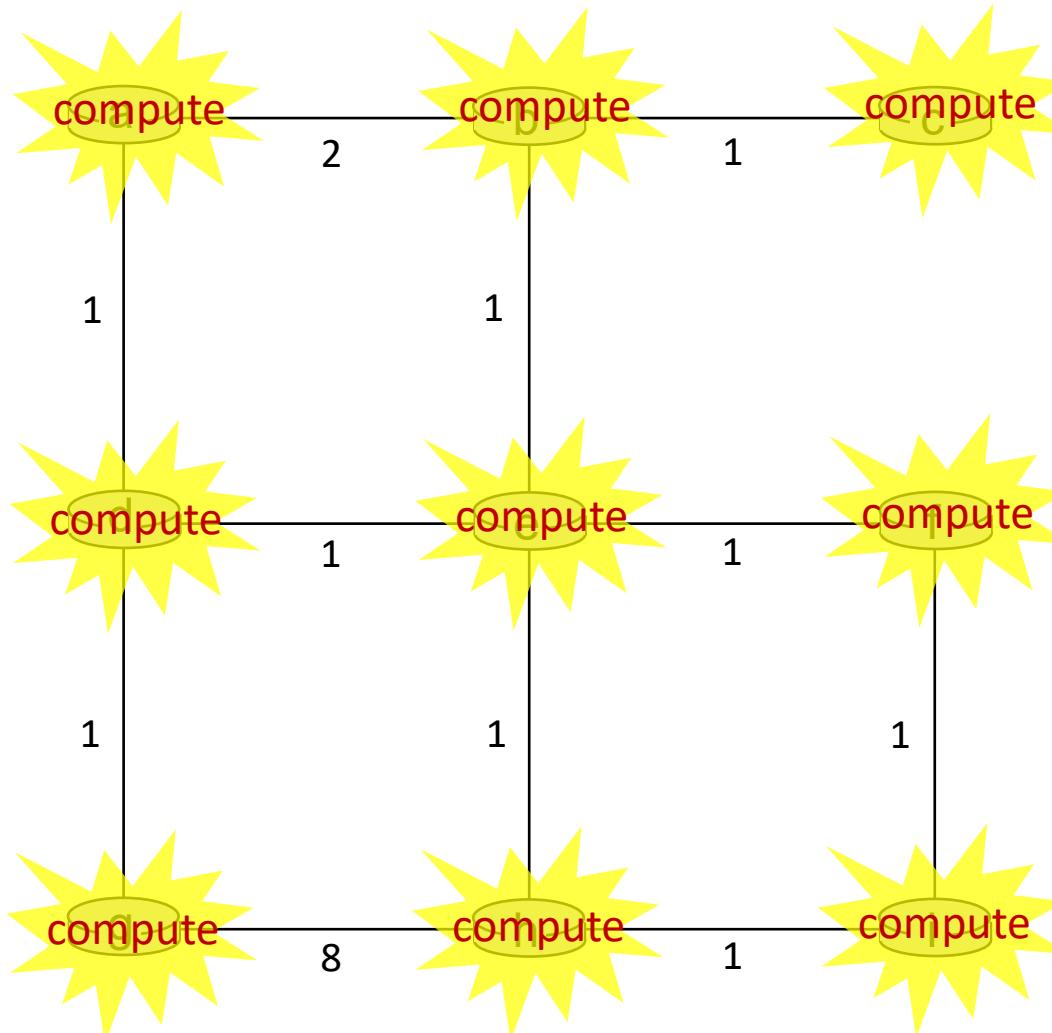
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



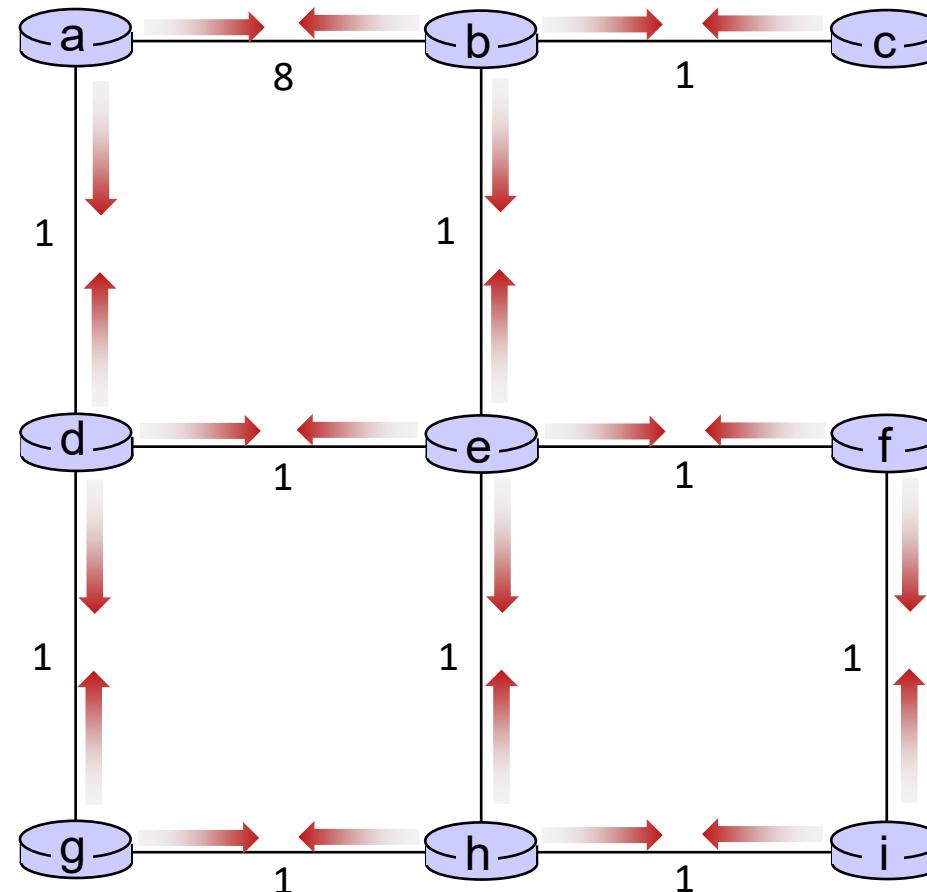
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



# Distance vector example: iteration

.... and so on

Let's next take a look at the iterative *computations* at nodes

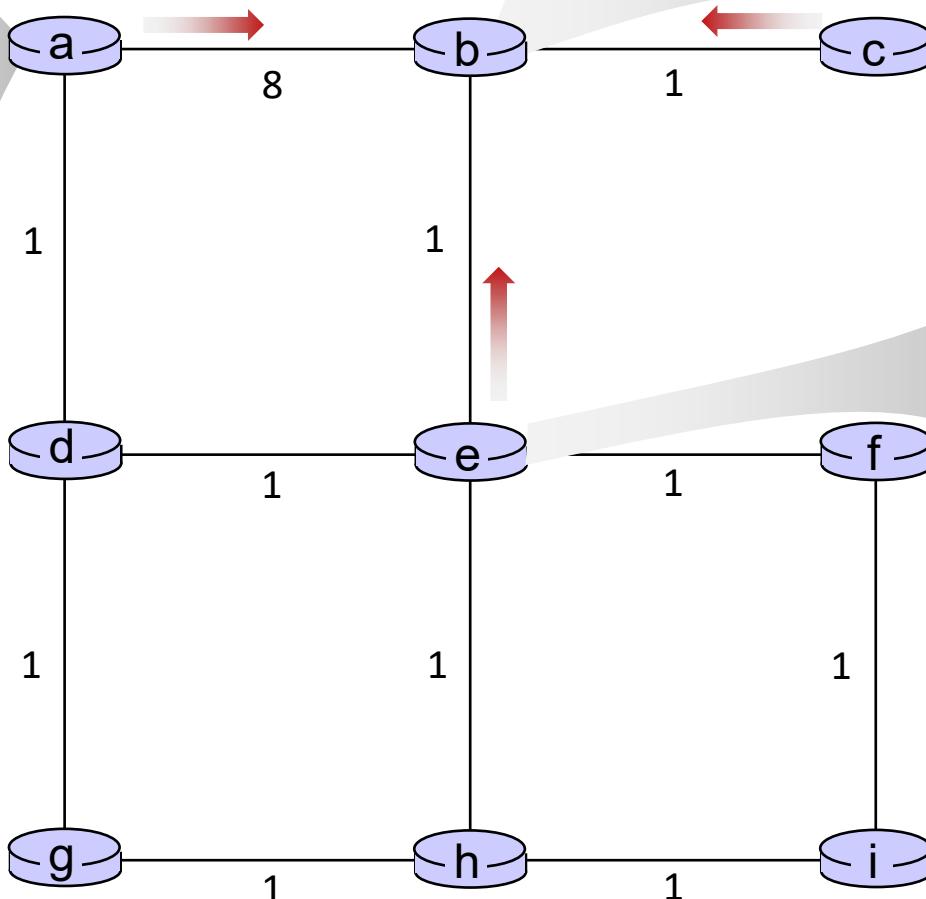
# Distance vector example: t=1



$t=1$

- b receives DVs from a, c, e

| DV in a:          |
|-------------------|
| $D_a(a) = 0$      |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



| DV in b:          |
|-------------------|
| $D_b(a) = 8$      |
| $D_b(f) = \infty$ |
| $D_b(c) = 1$      |
| $D_b(g) = \infty$ |
| $D_b(d) = \infty$ |
| $D_b(h) = \infty$ |
| $D_b(e) = 1$      |
| $D_b(i) = \infty$ |

| DV in c:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = 1$      |
| $D_c(c) = 0$      |
| $D_c(d) = \infty$ |
| $D_c(e) = \infty$ |
| $D_c(f) = \infty$ |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

# Distance vector example: t=1



**t=1**

- b receives DVs from a, c, e, computes:

$$D_b(a) = \min\{c_{b,a} + D_a(a), c_{b,c} + D_c(a), c_{b,e} + D_e(a)\} = \min\{8, \infty, \infty\} = 8$$

$$D_b(c) = \min\{c_{b,a} + D_a(c), c_{b,c} + D_c(c), c_{b,e} + D_e(c)\} = \min\{\infty, 1, \infty\} = 1$$

$$D_b(d) = \min\{c_{b,a} + D_a(d), c_{b,c} + D_c(d), c_{b,e} + D_e(d)\} = \min\{8+1, \infty, 1+1\} = 2$$

$$D_b(e) = \min\{c_{b,a} + D_a(e), c_{b,c} + D_c(e), c_{b,e} + D_e(e)\} = \min\{\infty, \infty, 1\} = 1$$

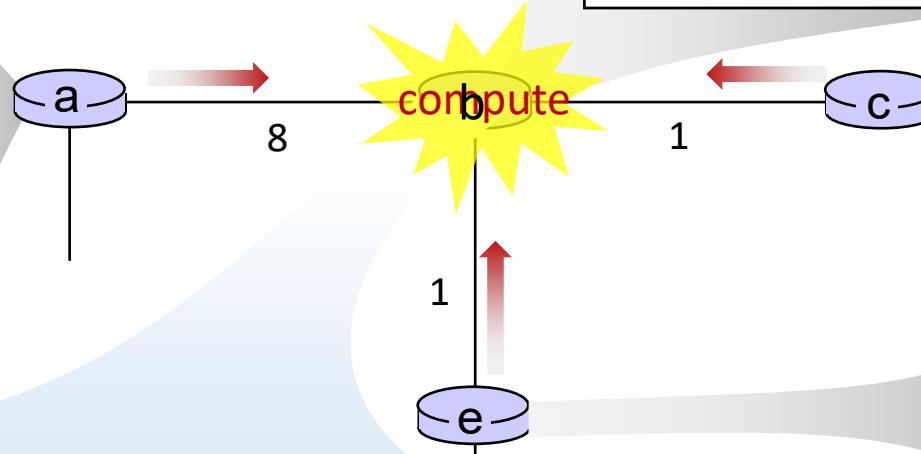
$$D_b(f) = \min\{c_{b,a} + D_a(f), c_{b,c} + D_c(f), c_{b,e} + D_e(f)\} = \min\{\infty, \infty, 1+1\} = 2$$

$$D_b(g) = \min\{c_{b,a} + D_a(g), c_{b,c} + D_c(g), c_{b,e} + D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty$$

$$D_b(h) = \min\{c_{b,a} + D_a(h), c_{b,c} + D_c(h), c_{b,e} + D_e(h)\} = \min\{\infty, \infty, 1+1\} = 2$$

$$D_b(i) = \min\{c_{b,a} + D_a(i), c_{b,c} + D_c(i), c_{b,e} + D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty$$

| DV in a:          |
|-------------------|
| $D_a(a)=0$        |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



| DV in b:          |
|-------------------|
| $D_b(a) = 8$      |
| $D_b(f) = \infty$ |
| $D_b(c) = 1$      |
| $D_b(g) = \infty$ |
| $D_b(d) = \infty$ |
| $D_b(h) = \infty$ |
| $D_b(e) = 1$      |
| $D_b(i) = \infty$ |

| DV in c:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = 1$      |
| $D_c(c) = 0$      |
| $D_c(d) = \infty$ |
| $D_c(e) = \infty$ |
| $D_c(f) = \infty$ |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

| DV in b:          |
|-------------------|
| $D_b(a) = 8$      |
| $D_b(f) = 2$      |
| $D_b(c) = 1$      |
| $D_b(g) = \infty$ |
| $D_b(d) = 2$      |
| $D_b(h) = 2$      |
| $D_b(e) = 1$      |
| $D_b(i) = \infty$ |

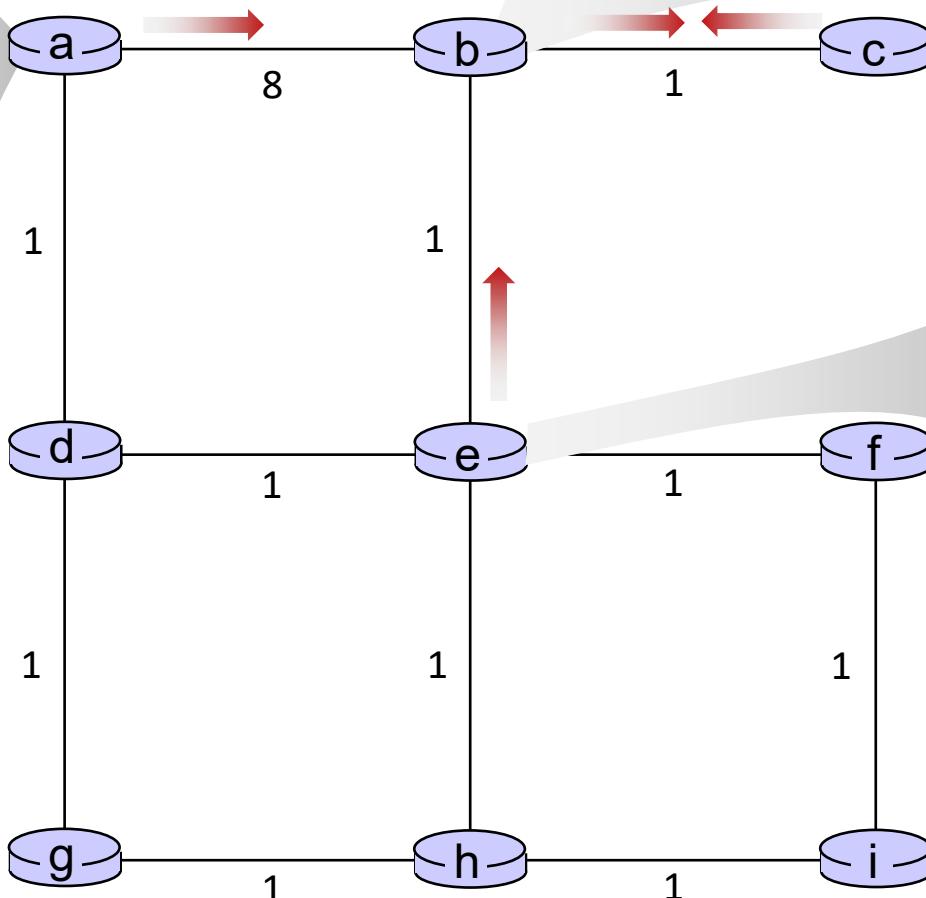
# Distance vector example: t=1



**t=1**

- c receives DVs from b

| DV in a:          |
|-------------------|
| $D_a(a) = 0$      |
| $D_a(b) = 8$      |
| $D_a(c) = \infty$ |
| $D_a(d) = 1$      |
| $D_a(e) = \infty$ |
| $D_a(f) = \infty$ |
| $D_a(g) = \infty$ |
| $D_a(h) = \infty$ |
| $D_a(i) = \infty$ |



| DV in b:          |
|-------------------|
| $D_b(a) = 8$      |
| $D_b(f) = \infty$ |
| $D_b(c) = 1$      |
| $D_b(g) = \infty$ |
| $D_b(d) = \infty$ |
| $D_b(h) = \infty$ |
| $D_b(e) = 1$      |
| $D_b(i) = \infty$ |

| DV in c:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = 1$      |
| $D_c(c) = 0$      |
| $D_c(d) = \infty$ |
| $D_c(e) = \infty$ |
| $D_c(f) = \infty$ |
| $D_c(g) = \infty$ |
| $D_c(h) = \infty$ |
| $D_c(i) = \infty$ |

| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

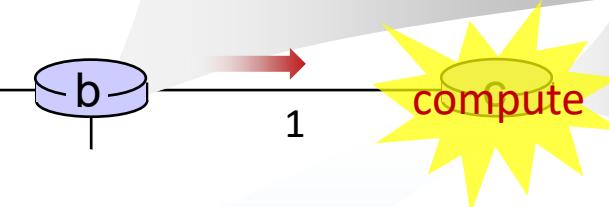
# Distance vector example: (t=1)



t=1

- c receives DVs from b computes:

$$\begin{aligned}D_c(a) &= \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9 \\D_c(b) &= \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1 \\D_c(d) &= \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty \\D_c(e) &= \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2 \\D_c(f) &= \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty \\D_c(g) &= \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty \\D_c(h) &= \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty \\D_c(i) &= \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty\end{aligned}$$



| DV in b:          |                   |
|-------------------|-------------------|
| $D_b(a) = 8$      | $D_b(f) = \infty$ |
| $D_b(c) = 1$      | $D_b(g) = \infty$ |
| $D_b(d) = \infty$ | $D_b(h) = \infty$ |
| $D_b(e) = 1$      | $D_b(i) = \infty$ |

| DV in c:          |  |
|-------------------|--|
| $D_c(a) = \infty$ |  |
| $D_c(b) = 1$      |  |
| $D_c(c) = 0$      |  |
| $D_c(d) = \infty$ |  |
| $D_c(e) = \infty$ |  |
| $D_c(f) = \infty$ |  |
| $D_c(g) = \infty$ |  |
| $D_c(h) = \infty$ |  |
| $D_c(i) = \infty$ |  |

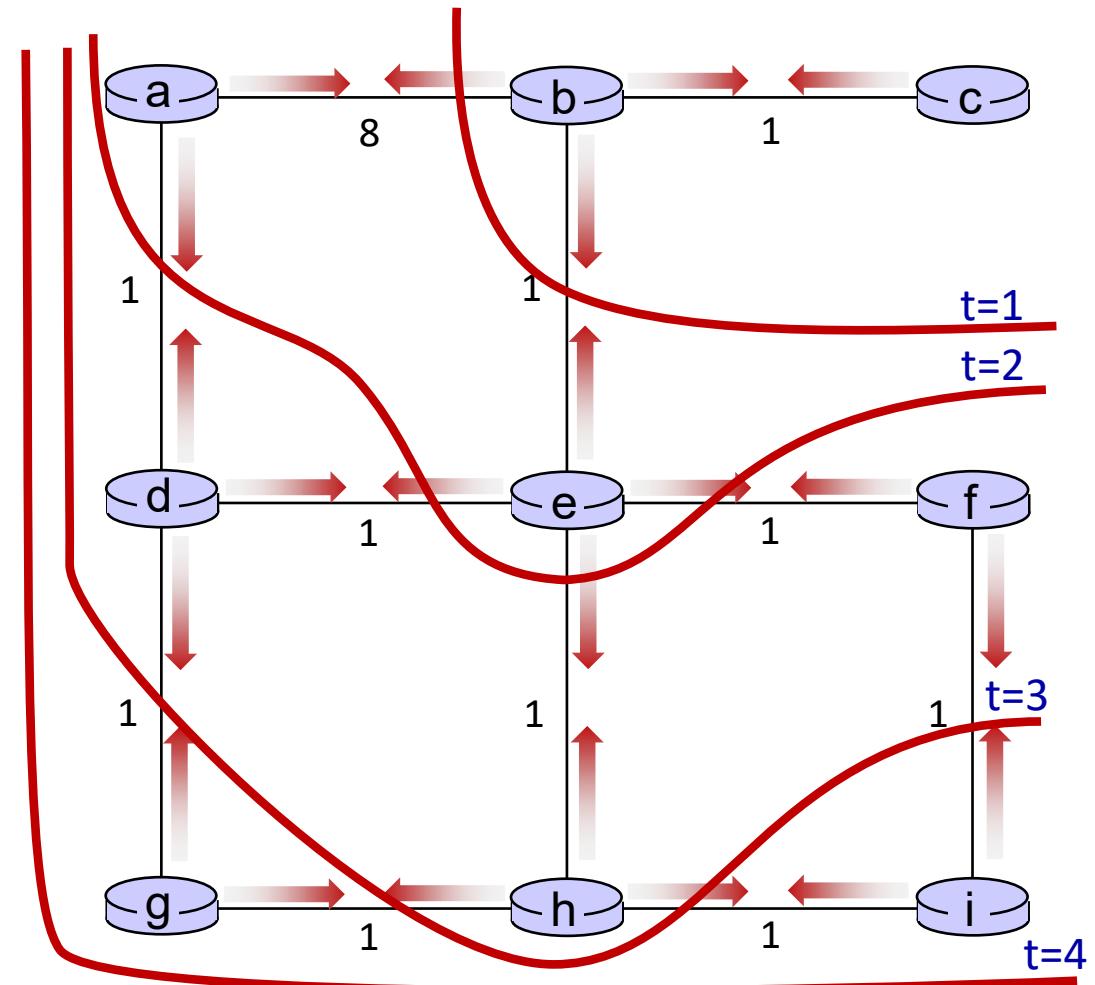
| DV in c:          |  |
|-------------------|--|
| $D_c(a) = 9$      |  |
| $D_c(b) = 1$      |  |
| $D_c(c) = 0$      |  |
| $D_c(d) = 2$      |  |
| $D_c(e) = \infty$ |  |
| $D_c(f) = \infty$ |  |
| $D_c(g) = \infty$ |  |
| $D_c(h) = \infty$ |  |
| $D_c(i) = \infty$ |  |

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

# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

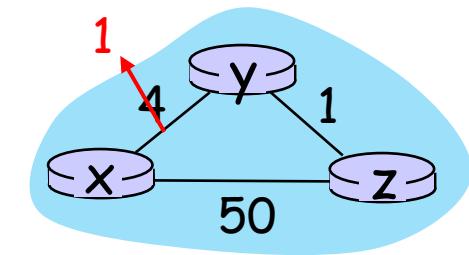
-  t=0 c's state at t=0 is at c only
-  t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away i.e., at b
-  t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away i.e., at b and now at a, e as well
-  t=3 c's state at t=0 may influence distance vector computations up to **3** hops away i.e., at d, f, h
-  t=4 c's state at t=0 may influence distance vector computations up to **4** hops away i.e., at g, i



# Distance vector: link cost changes

## link cost changes:

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



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

“good news travels fast”

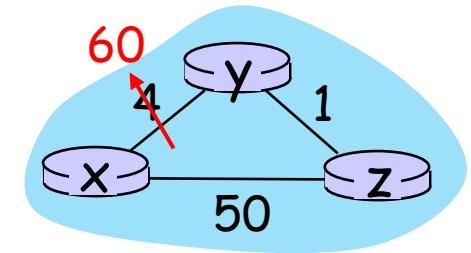
$t_1$ : z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV.

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

# Distance vector: link cost changes

## link cost changes:

- node detects local link cost change
- “bad news travels slow” – count-to-infinity problem:
  - y sees direct link to x has new cost 60, but z has said it has a path at cost of 5. So y computes “my new cost to x will be 6, via z); notifies z of new cost of 6 to x.
  - z learns that path to x via y has new cost 6, so z computes “my new cost to x will be 7 via y), notifies y of new cost of 7 to x.
  - y learns that path to x via z has new cost 7, so y computes “my new cost to x will be 8 via y), notifies z of new cost of 8 to x.
  - z learns that path to x via y has new cost 8, so z computes “my new cost to x will be 9 via y), notifies y of new cost of 9 to x.
  - ...
- see text for solutions. *Distributed algorithms are tricky!*



# Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

**5.3 Intra-ISP routing: OSPF**

5.4 Routing among ISPs: BGP

5.5 SDN control plane

5.6 Internet Control Message Protocol

# Making routing scalable

our routing study thus far - idealized

- all routers identical
- network “flat”

... not true in practice

**scale:** billions of destinations:

- can't store all destinations in routing tables!
- routing table exchange would swamp links!

**administrative autonomy:**

- Internet: a network of networks
- each network admin may want to control routing in its own network

# Internet approach to scalable routing

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

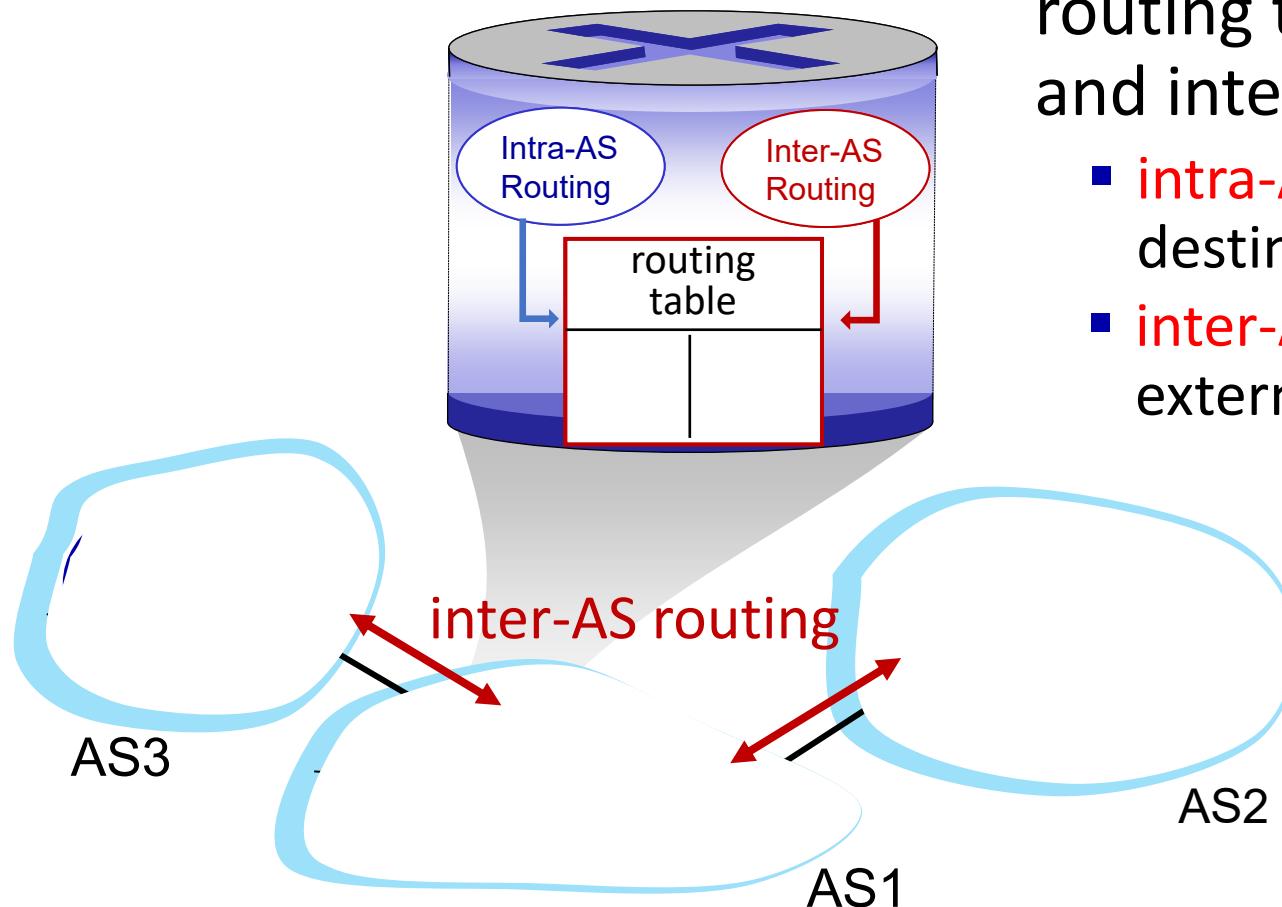
**intra-AS (aka “intra-domain”):**  
routing among *within same AS (“network”)*

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

**inter-AS (aka “inter-domain”):**  
routing *among* AS'es

- gateways perform inter-domain routing (as well as intra-domain routing)

# Interconnected ASes



routing tables configured by intra-  
and inter-AS routing algorithms

- **intra-AS** routing determine entries for destinations within AS
- **inter-AS** routing determine entries for external destinations

# Intra-AS routing: Routing within an AS

Most common intra-AS routing protocols:

- **RIP: Routing Information Protocol [RFC 1723]**
  - DVs exchanged every 30 secs
  - no longer widely used
- **EIGRP: Enhanced Interior Gateway Routing Protocol**
  - DV based
  - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
- **OSPF: Open Shortest Path First [RFC 2328]**
  - link-state routing
  - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

# OSPF (Open Shortest Path First) routing

- centralized routing algorithm
  - each router has full topology, uses Dijkstra's algorithm to compute routing table
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
- *security*: all OSPF messages authenticated (to prevent malicious intrusion)

# Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

5.3 Intra-ISP routing: OSPF

**5.4 Routing among ISPs: BGP**

5.5 SDN control plane

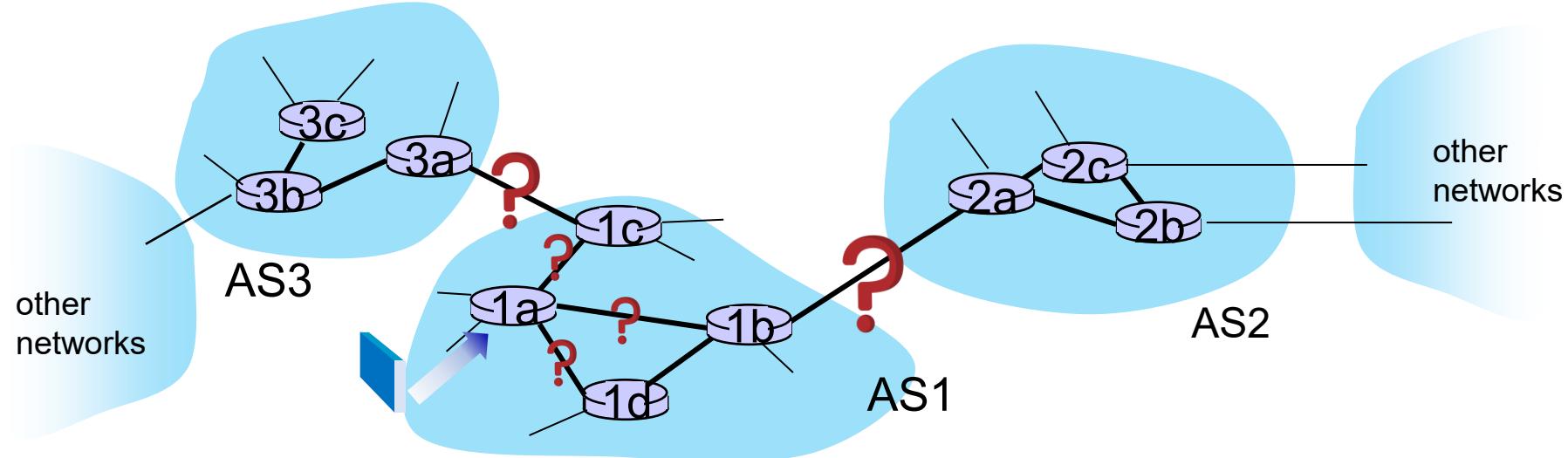
5.6 Internet Control Message Protocol

# Inter-AS routing: a role in intradomain forwarding

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

**AS1 inter-domain routing must:**

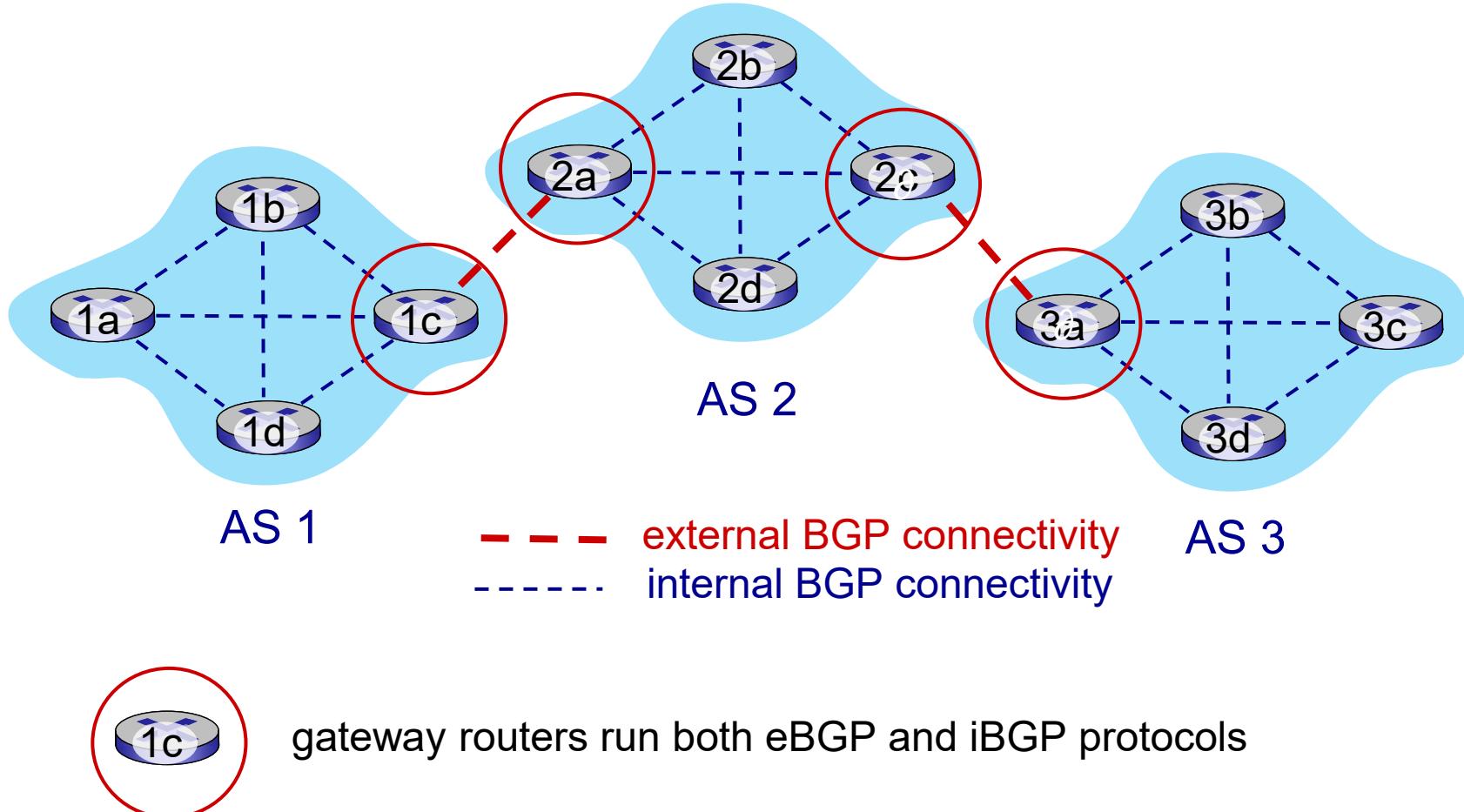
1. learn which destinations are reachable through AS2 and AS3
2. propagate this reachability info to all routers in AS1



# Internet inter-AS routing: BGP

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

# eBGP, iBGP connections

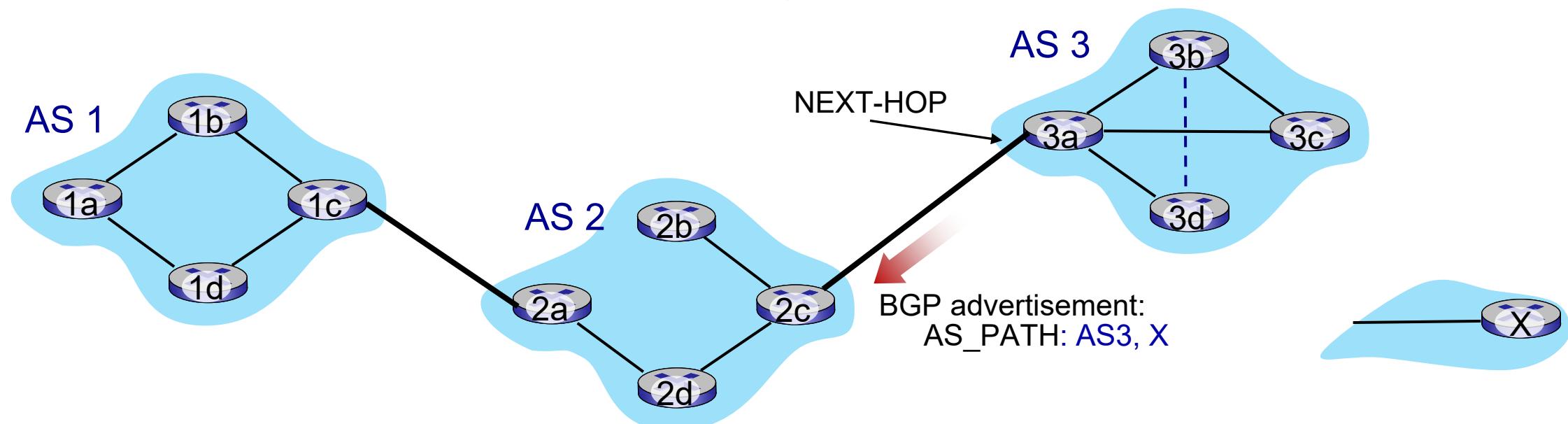


# Path attributes and BGP routes

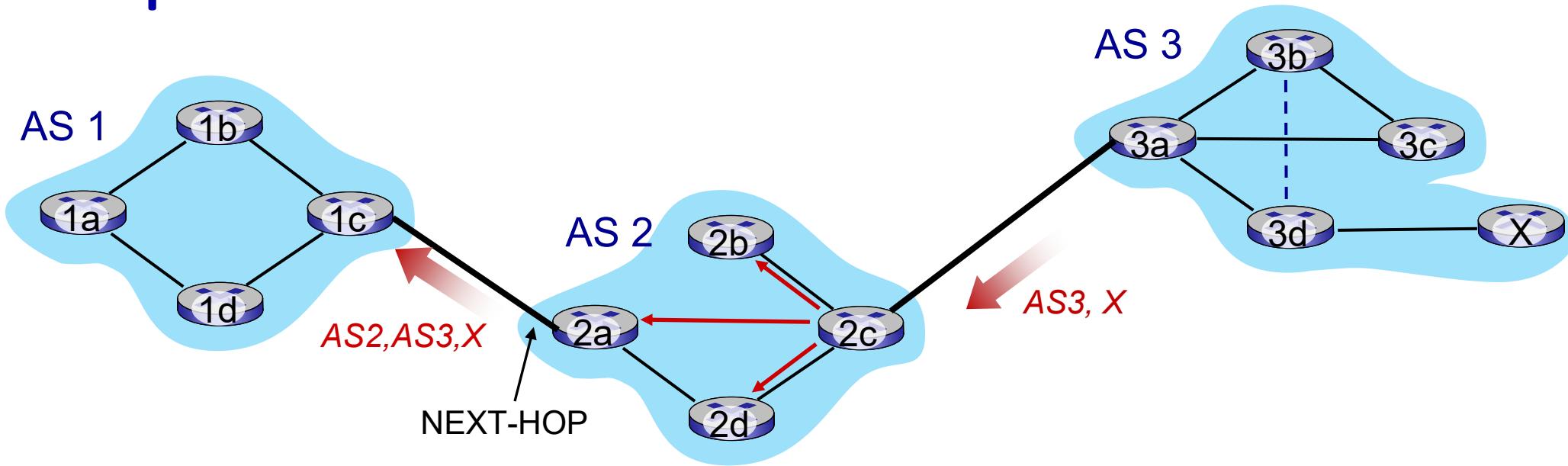
- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of AS's through which prefix advertisement has passed
    - NEXT-HOP: indicates IP address of the router interface that begins the AS-PATH
- **policy-based routing:**
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

# BGP basics

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

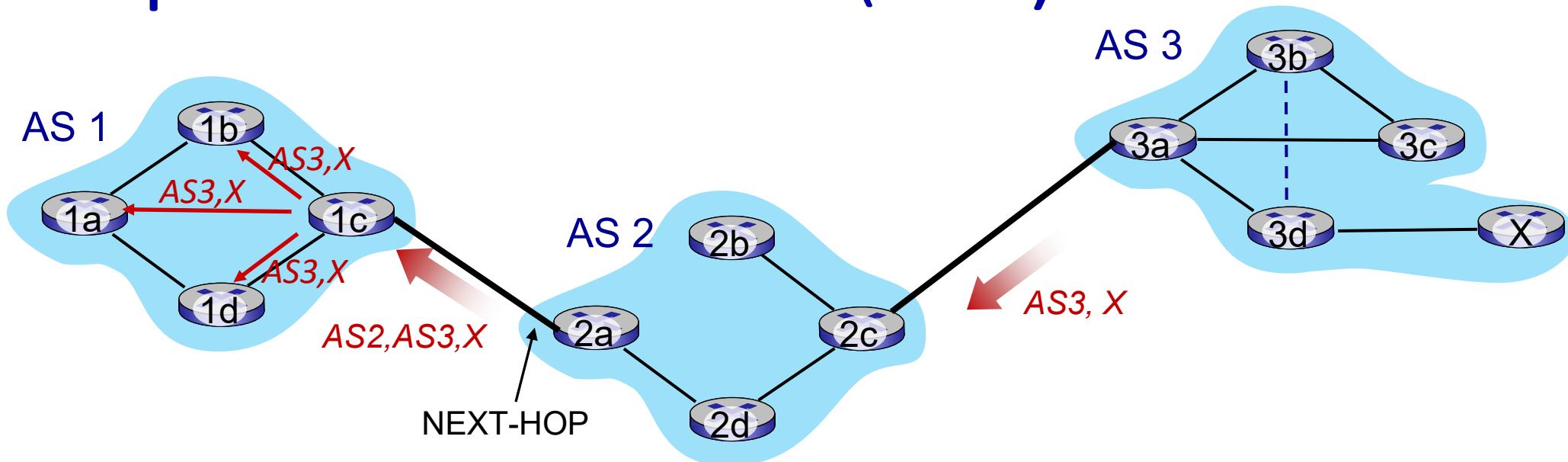


# BGP path advertisement



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

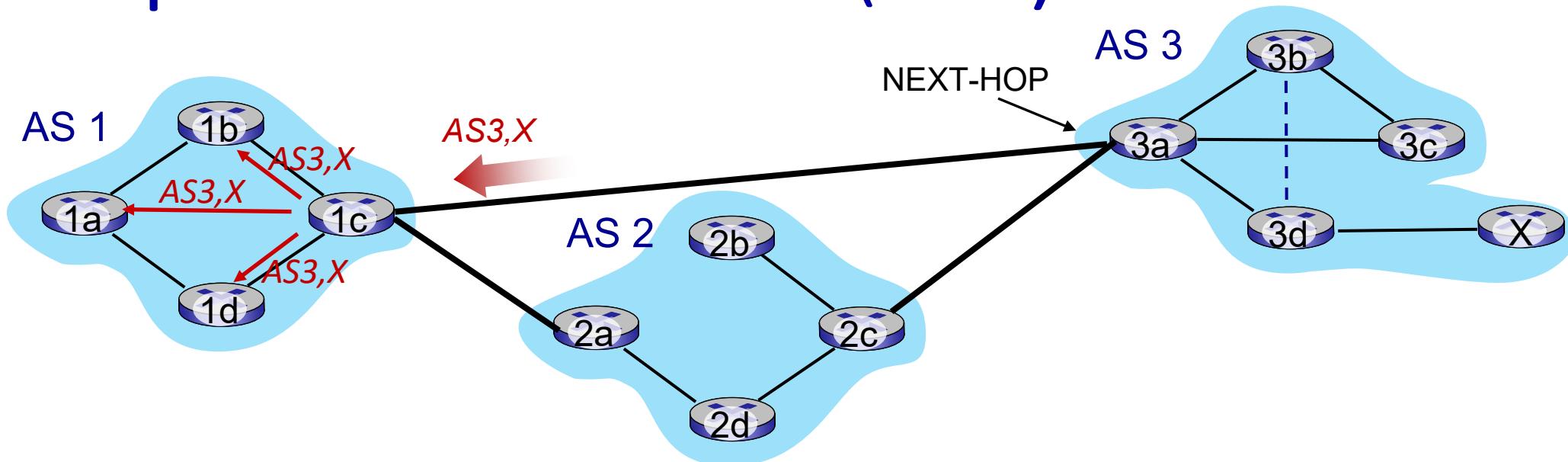
# BGP path advertisement (more)



gateway router may learn about multiple paths to destination:

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

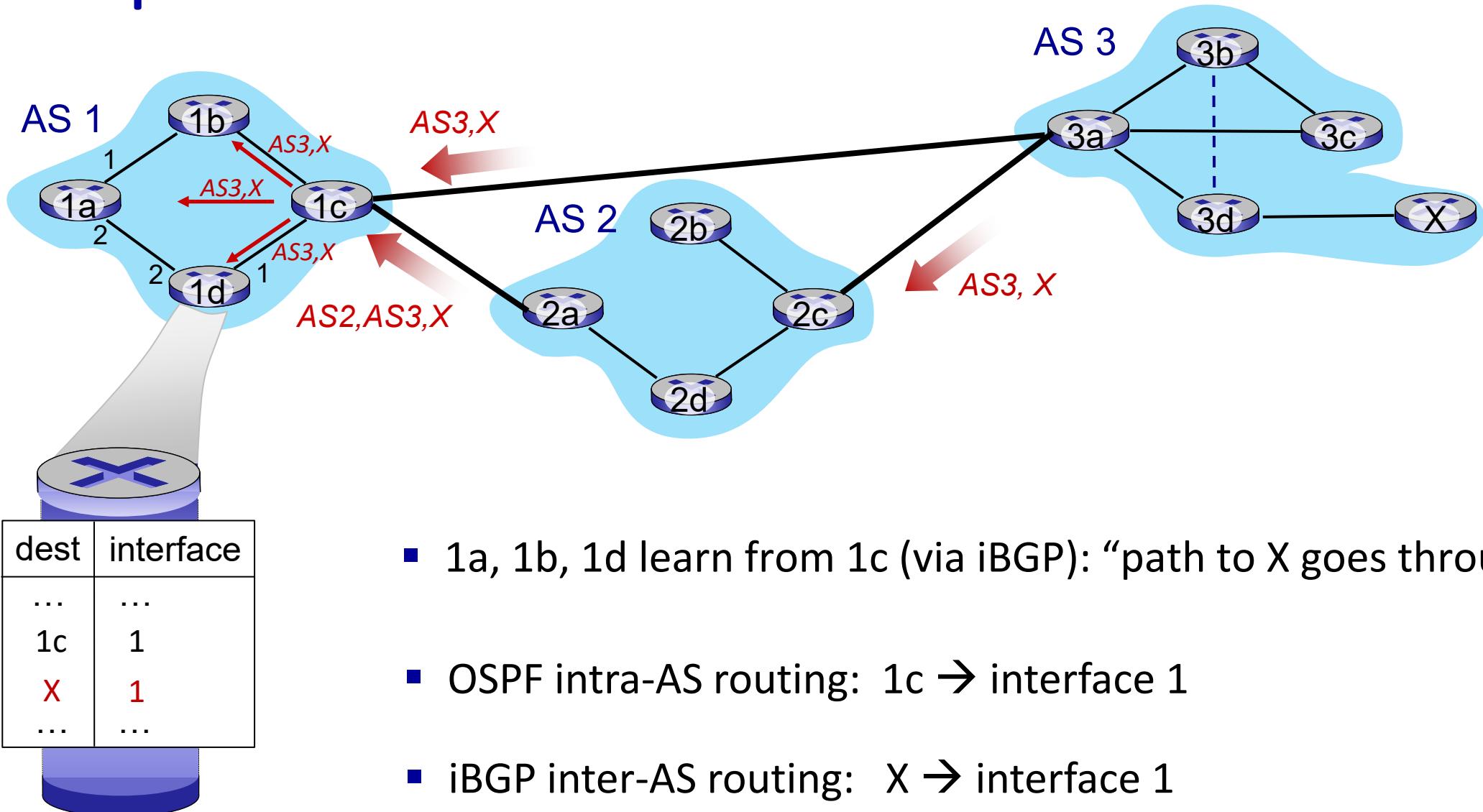
# BGP path advertisement (more)



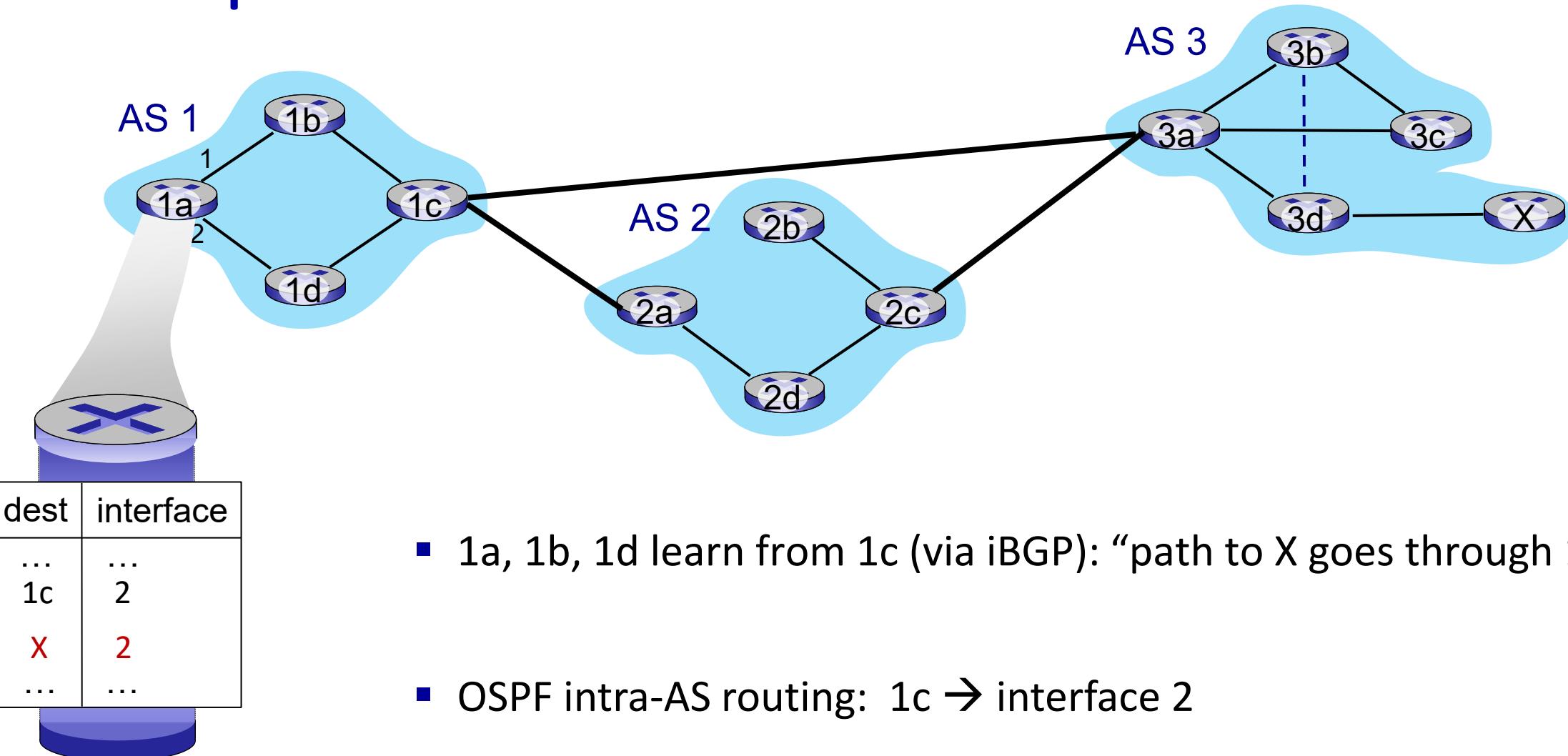
gateway router may learn about multiple paths to destination:

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

# BGP path advertisement



# BGP path advertisement



- 1a, 1b, 1d learn from 1c (via iBGP): “path to X goes through 1c”
- OSPF intra-AS routing: 1c → interface 2
- iBGP inter-AS routing: X → interface 2

# Network layer: “control plane” roadmap

5.1 Introduction

5.2 Routing algorithms

5.3 Intra-ISP routing: OSPF

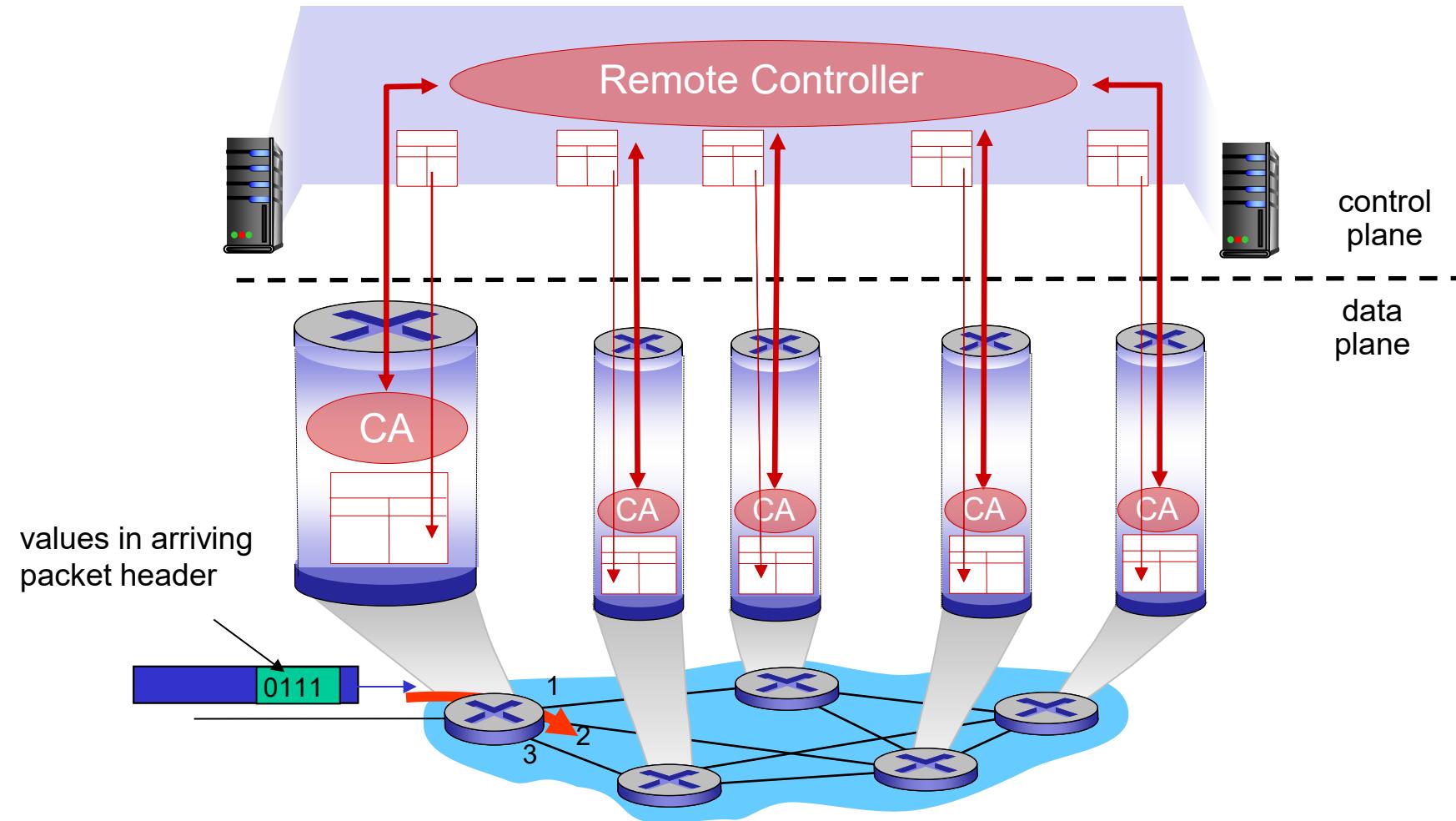
5.4 Routing among ISPs: BGP

**5.5 SDN control plane**

5.6 Internet Control Message Protocol

# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Software defined networking (SDN)

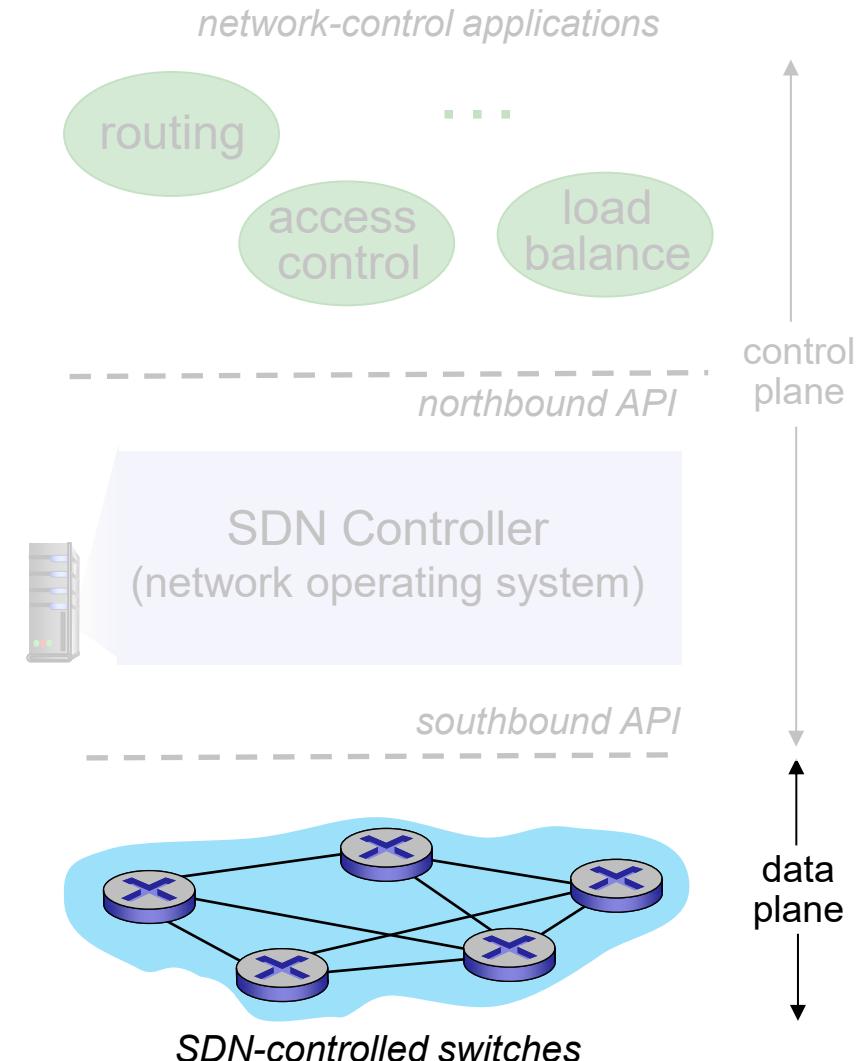
*Why* a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- “programming” routers
  - centralized “programming” easier: compute tables centrally and distribute
  - distributed “programming” more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

# Software defined networking (SDN)

## Data-plane switches:

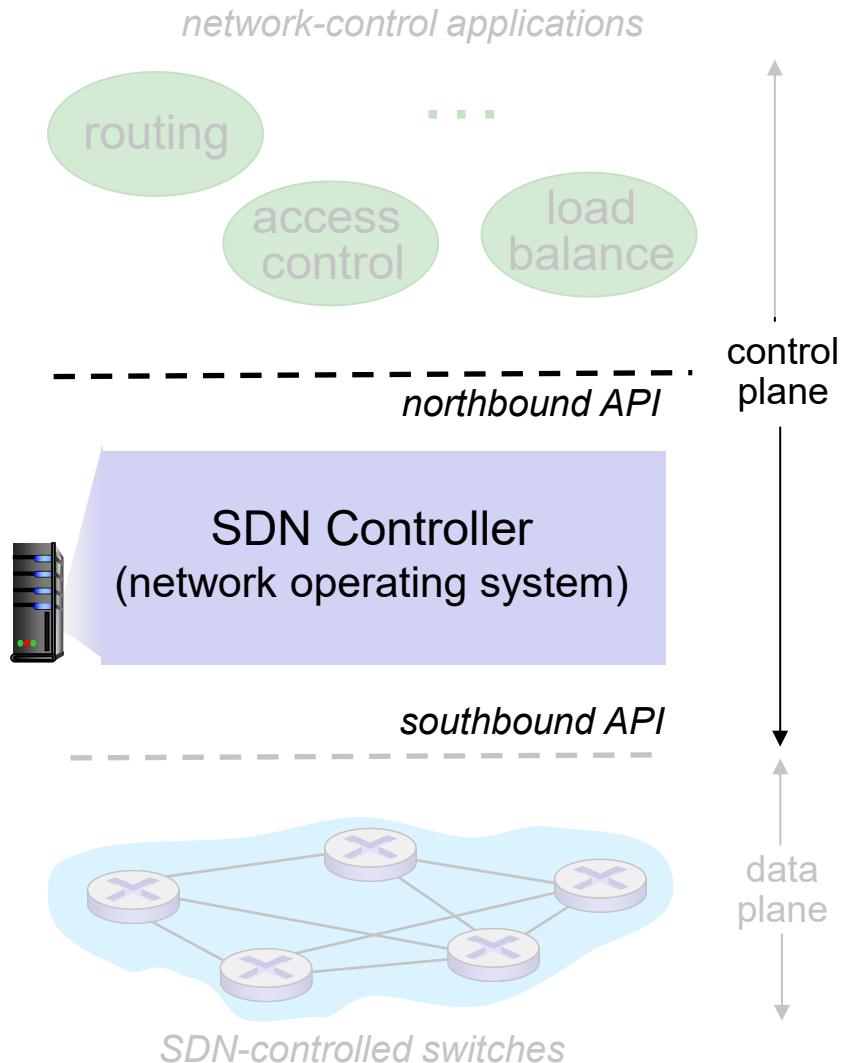
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating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



# Software defined networking (SDN)

## SDN controller (network OS):

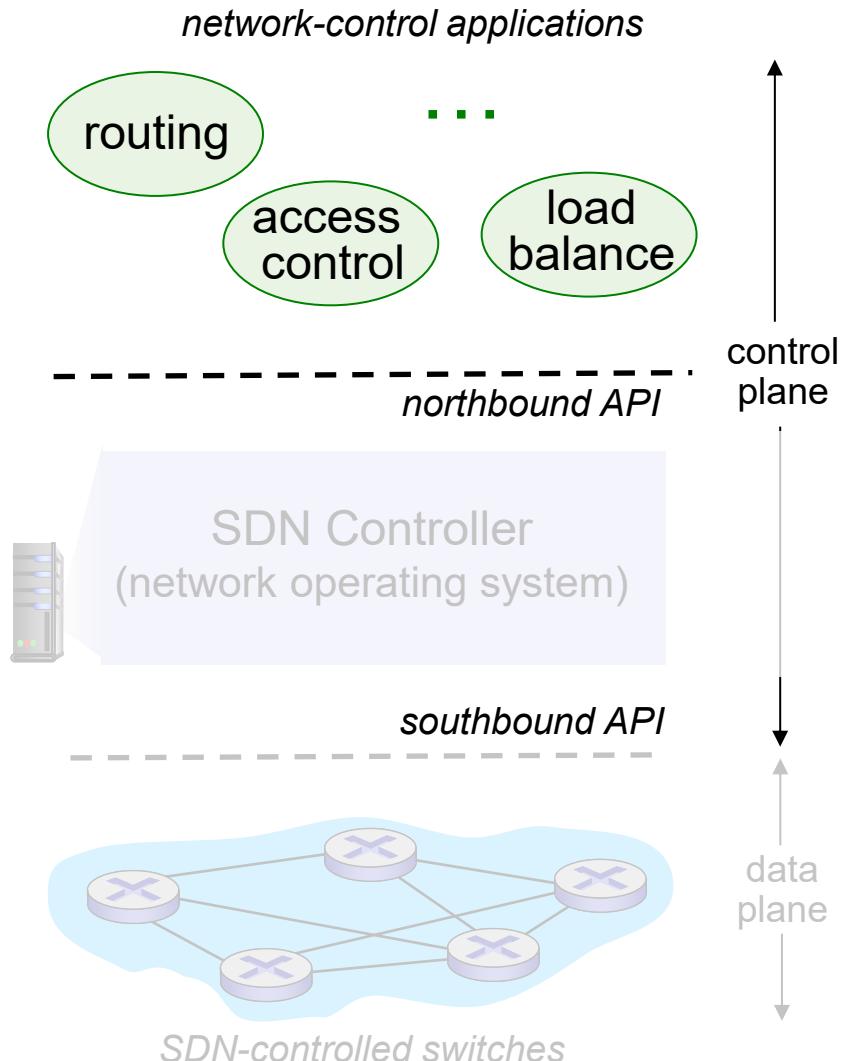
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



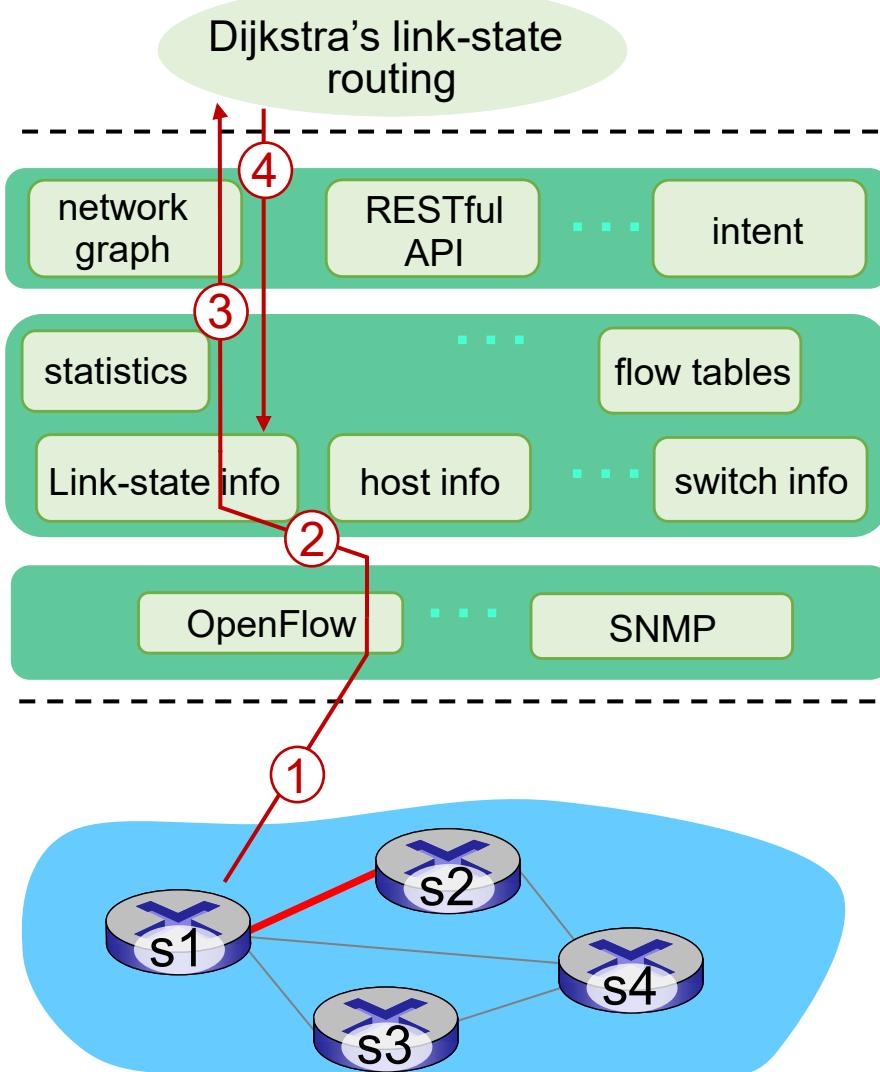
# Software defined networking (SDN)

## network-control apps:

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

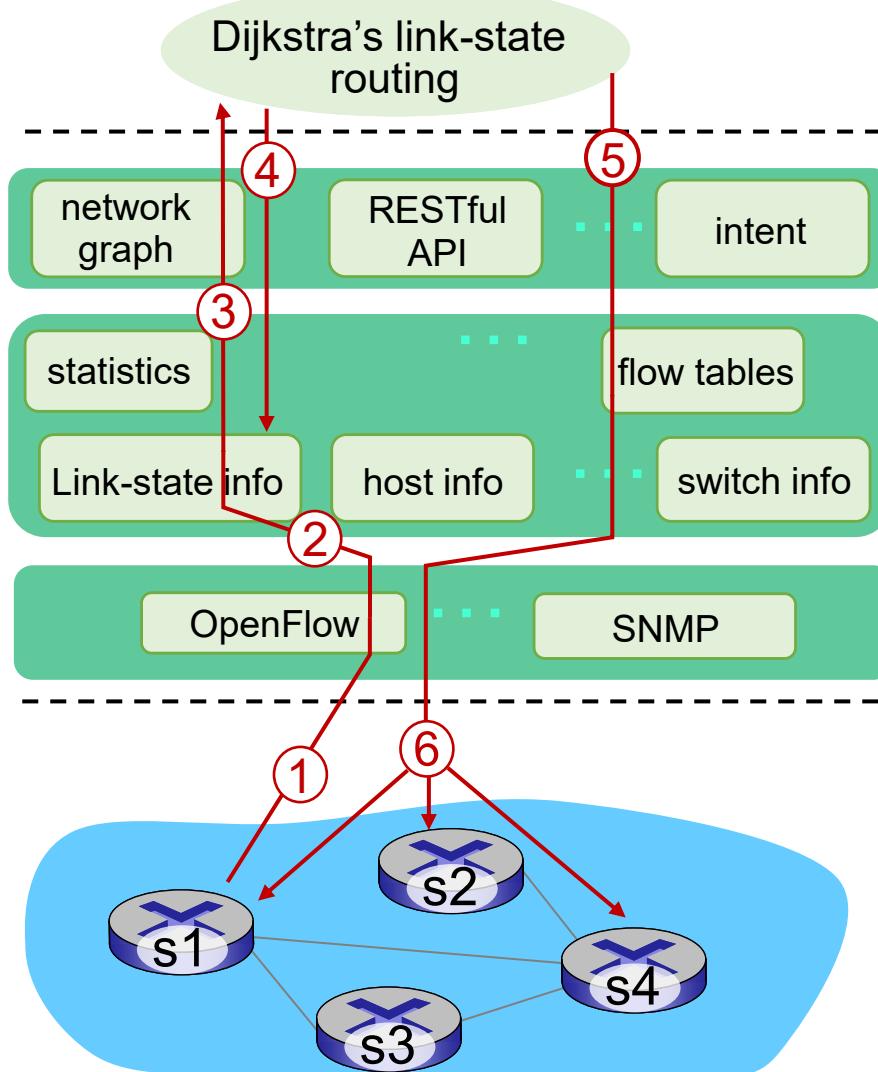


# SDN: control/data plane interaction example



- ① S1, experiencing link failure uses OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It is called.
- ④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

# SDN: control/data plane interaction example



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

# NoviSwitch™ 2150 High Performance OpenFlow Switch

**NoviSwitch 2150** is an OpenFlow switch offering genuine wire-speed performance using the OpenFlow 1.3 to 1.5 standards and has been specifically designed for use in high bandwidth / flow-intensive network deployments. Includes the *NoviWare™ 400.4* OpenFlow Switch Software for use with the Mellanox high performance NP-5 network processor.



Today's major network operators demand flexible, scalable networking solutions that deliver wire-speed performance. **NoviFlow Inc.™** is changing the traditional approach to networking by making switching smarter. The company delivers upon the promise of OpenFlow and Software Defined Networking (SDN) by combining the benefits of virtualization and programmability with network processors that can handle complex flows to make it possible for carriers, cloud providers and hyperscale data centers to keep up with today's exponentially growing networking demand.

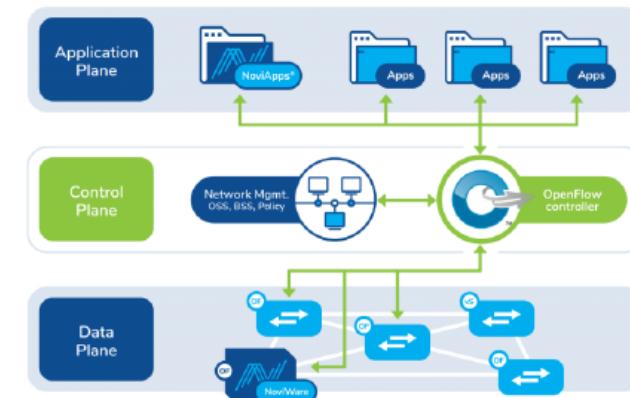
**NoviSwitch 2150** was specifically designed for deployment in Access Networks and data centers looking to leverage the benefits of SDN to improve the cost/performance, security, scalability and flexibility of networks. It is a forwarding plane platform delivering maximum OpenFlow capability in a compact form factor. The system is provided in a stand-alone, self-contained, 1U rack-mountable enclosure box that can be configured to support a wide variety of networking applications to deliver unmatched performance levels.

## Key Features:

Features the NoviWare 400.4 OpenFlow switching software, supporting all required and optional OpenFlow 1.3 and 1.4 match fields, instructions, actions and counters, as well as key OpenFlow 1.5 features, including group chaining.

- 256 Gbps and 190 Mpps of switching capacity powered by an Mellanox NP-5 NPU
- 50 data plane ports:
  - 2 QSFP+ transceiver cages for 40GE connectivity
  - 48 SFP transceiver cages for 10/100/1000BASE-T/100BASE-FX/1000BASE-X connectivity
- Up to 1 Million wildcard match flow (with optional external TCAM), otherwise 16,000 flow entries in up to 60 tables
- Up to 40,000 flow-mods/second
- Up to 1 Million meters and 10,000 entries in Groups table

NoviFlow's Flexible Platform



# Network layer: “control plane” roadmap

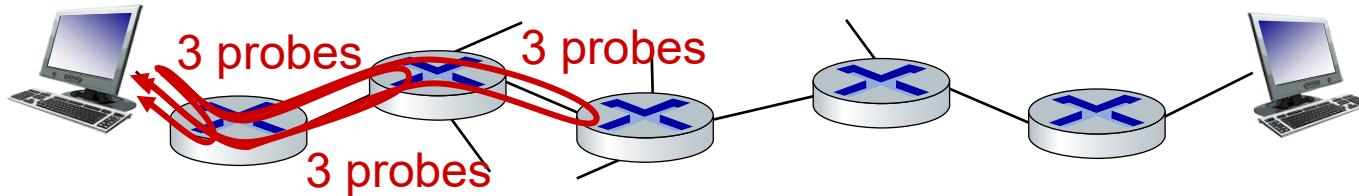
- 5.1 Introduction
- 5.2 Routing algorithms
- 5.3 Intra-ISP routing: OSPF
- 5.4 Routing among ISPs: BGP
- 5.5 SDN control plane
- **5.6 Internet Control Message Protocol**

# ICMP: internet control message protocol

- used by hosts and routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer “above” IP:
  - ICMP messages carried in IP datagrams
- *ICMP message*: type, code plus first 8 bytes of IP datagram causing error

| Type | Code | description                                   |
|------|------|-----------------------------------------------|
| 0    | 0    | echo reply (ping)                             |
| 3    | 0    | dest network unreachable                      |
| 3    | 1    | dest host unreachable                         |
| 3    | 2    | dest protocol unreachable                     |
| 3    | 3    | dest port unreachable                         |
| 3    | 6    | dest network unknown                          |
| 3    | 7    | dest host unknown                             |
| 4    | 0    | source quench (congestion control - not used) |
| 8    | 0    | echo request (ping)                           |
| 9    | 0    | route advertisement                           |
| 10   | 0    | router discovery                              |
| 11   | 0    | TTL expired                                   |
| 12   | 0    | bad IP header                                 |

# Traceroute and ICMP



- source sends sets of UDP segments to destination
  - 1<sup>st</sup> set has TTL =1, 2<sup>nd</sup> set has TTL=2, etc.
- datagram in *n*th set arrives to *n*th router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

## stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP “port unreachable” message (type 3, code 3)
- source stops

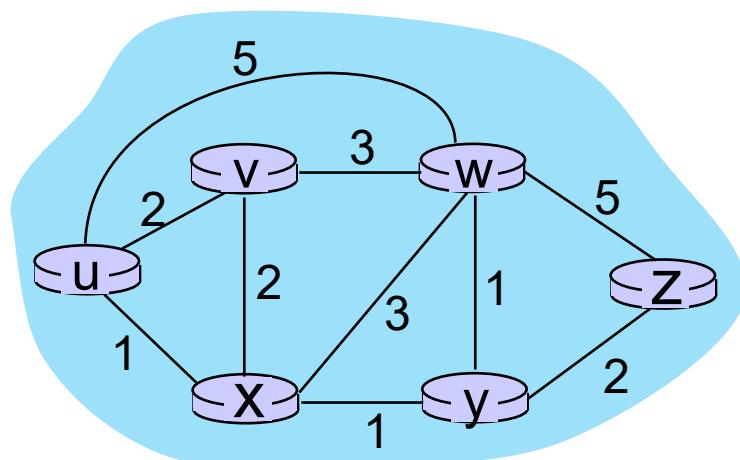
# Network layer, control plane: Done!

- Introduction
- Routing algorithms
  - link state
  - distance vector
- Intra-ISP routing: OSPF
- Routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

# Original slides on Dijkstra example

# Dijkstra's algorithm: an example

| Step | N' | V<br>D(v),p(v) | W<br>D(w),p(w) | X<br>D(x),p(x) | Y<br>D(y),p(y) | Z<br>D(z),p(z) |
|------|----|----------------|----------------|----------------|----------------|----------------|
| 0    | u  | 2,u            | 5,u            | 1,u            | $\infty$       | $\infty$       |
| 1    |    |                |                |                |                |                |
| 2    |    |                |                |                |                |                |
| 3    |    |                |                |                |                |                |
| 4    |    |                |                |                |                |                |
| 5    |    |                |                |                |                |                |

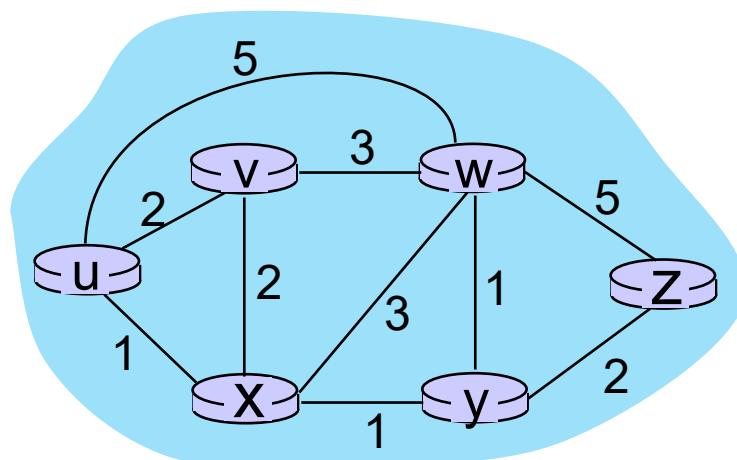


Initialization (step 0):

For all  $a$ : if  $a$  adjacent to  $u$  then  $D(a) = c_{u,a}$

# Dijkstra's algorithm: an example

| Step | N | t<br>d(t), p(t) | u<br>d(u), p(u) | v<br>d(v), p(v) | w<br>d(w), p(w) | y<br>d(y), p(y) |
|------|---|-----------------|-----------------|-----------------|-----------------|-----------------|
| 0    |   |                 |                 |                 |                 |                 |
| 1    |   |                 |                 |                 |                 |                 |
| 2    |   |                 |                 |                 |                 |                 |
| 3    |   |                 |                 |                 |                 |                 |
| 4    |   |                 |                 |                 |                 |                 |
| 5    |   |                 |                 |                 |                 |                 |

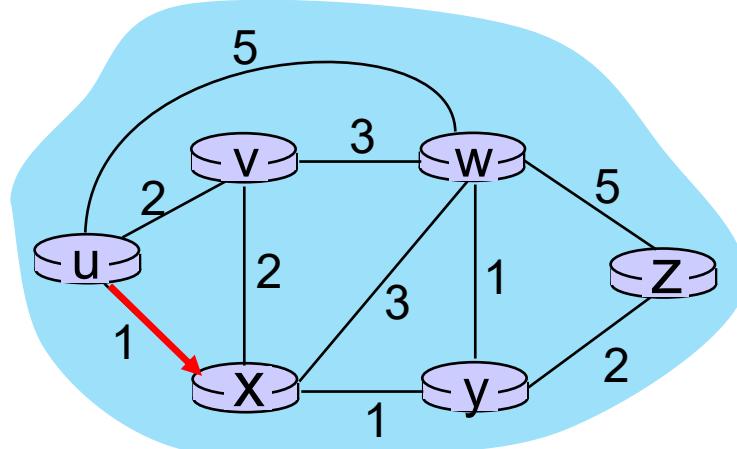


Initialization (step 0):

For all  $a$ : if  $a$  adjacent to  $u$  then  $D(a) = c_{u,a}$

# Dijkstra's algorithm: an example

| Step | $N'$ | $v$<br>$D(v), p(v)$ | $w$<br>$D(w), p(w)$ | $x$<br>$D(x), p(x)$ | $y$<br>$D(y), p(y)$ | $z$<br>$D(z), p(z)$ |
|------|------|---------------------|---------------------|---------------------|---------------------|---------------------|
| 0    | u    | 2,u                 | 5,u                 | 1,u                 | $\infty$            | $\infty$            |
| 1    | ux   |                     |                     |                     |                     |                     |
| 2    |      |                     |                     |                     |                     |                     |
| 3    |      |                     |                     |                     |                     |                     |
| 4    |      |                     |                     |                     |                     |                     |
| 5    |      |                     |                     |                     |                     |                     |



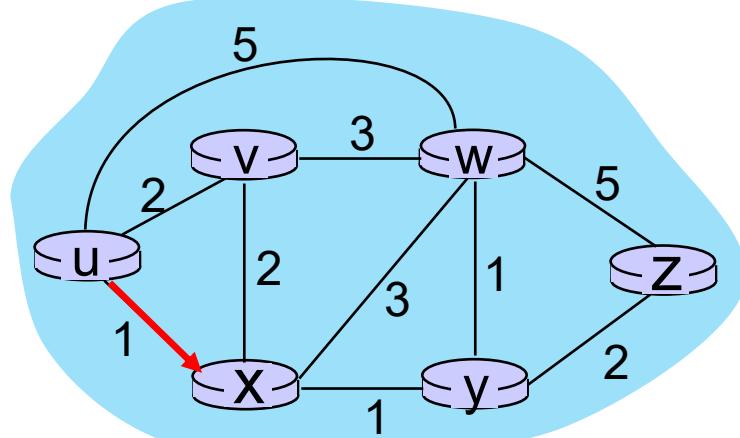
8 Loop

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

10 add  $a$  to  $N'$

# Dijkstra's algorithm: an example

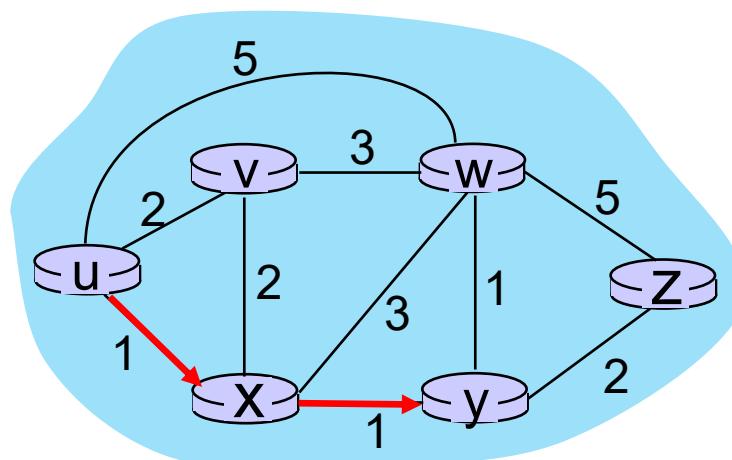
| Step | $N'$ | $D(v), p(v)$ | $D(w), p(w)$ | $D(x), p(x)$ | $D(y), p(y)$ | $D(z), p(z)$ |
|------|------|--------------|--------------|--------------|--------------|--------------|
| 0    | u    | 2,u          | 5,u          | 1,u          | $\infty$     | $\infty$     |
| 1    | ux   | 2,u        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



- 8 Loop
- 9 find  $a$  not in  $N'$  such that  $D(a)$  is a minimum
- 10 add  $a$  to  $N'$
- 11 update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$ :
- $$D(b) = \min ( D(b), D(a) + c_{a,b} )$$
- $D(v) = \min ( D(v), D(x) + c_{x,v} ) = \min(2, 1+2) = 2$
- $D(w) = \min ( D(w), D(x) + c_{x,w} ) = \min (5, 1+3) = 4$
- $D(y) = \min ( D(y), D(x) + c_{x,y} ) = \min(\infty, 1+1) = 2$

# Dijkstra's algorithm: an example

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



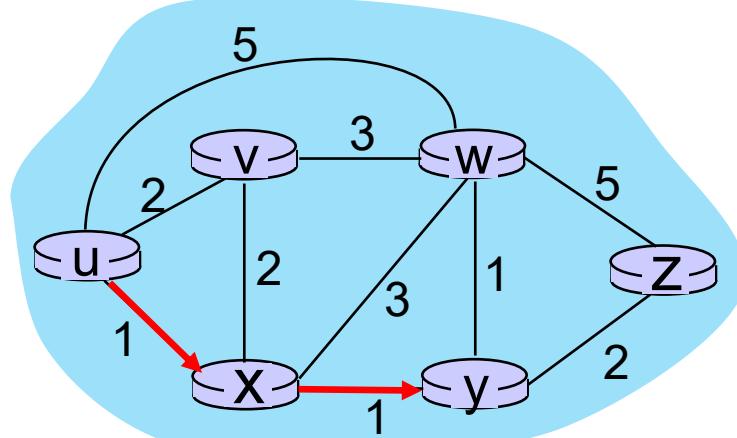
8 Loop

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

10 add  $a$  to  $N'$

# Dijkstra's algorithm: an example

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



8 Loop

- 9 find  $a$  not in  $N'$  such that  $D(a)$  is a minimum  
 10 add  $a$  to  $N'$   
 11 update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$ :

$$D(b) = \min ( D(b), D(a) + c_{a,b} )$$

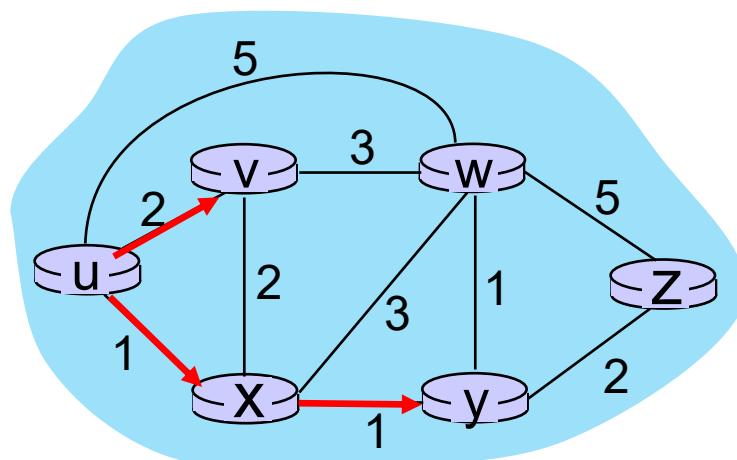
$$D(w) = \min ( D(w), D(y) + c_{y,w} ) = \min ( 4, 2+1 ) = 3$$

$$D(z) = \min ( D(z), D(y) + c_{y,z} ) = \min(\inf, 2+2) = 4$$



# Dijkstra's algorithm: an example

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



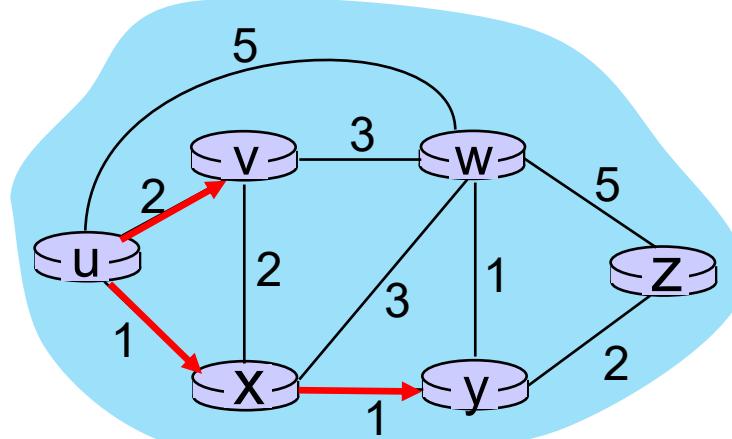
8 Loop

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

10 add  $a$  to  $N'$

# Dijkstra's algorithm: an example

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



8 Loop

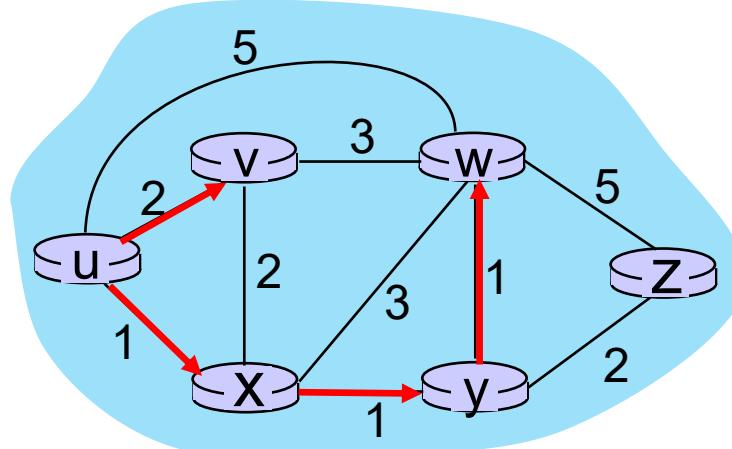
- 9 find  $a$  not in  $N'$  such that  $D(a)$  is a minimum  
 10 add  $a$  to  $N'$   
 11 update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$ :

$$D(b) = \min ( D(b), D(a) + c_{a,b} )$$

$$D(w) = \min ( D(w), D(v) + c_{v,w} ) = \min ( 3, 2+3 ) = 3$$

# Dijkstra's algorithm: an example

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



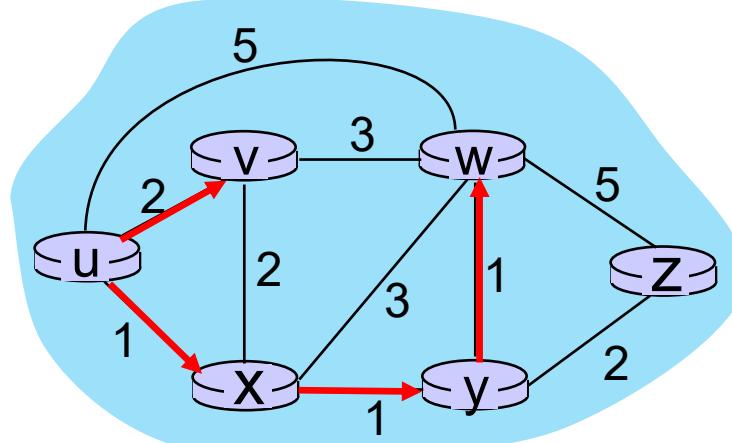
8 Loop

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

10 add  $a$  to  $N'$

# Dijkstra's algorithm: an example

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



8 Loop

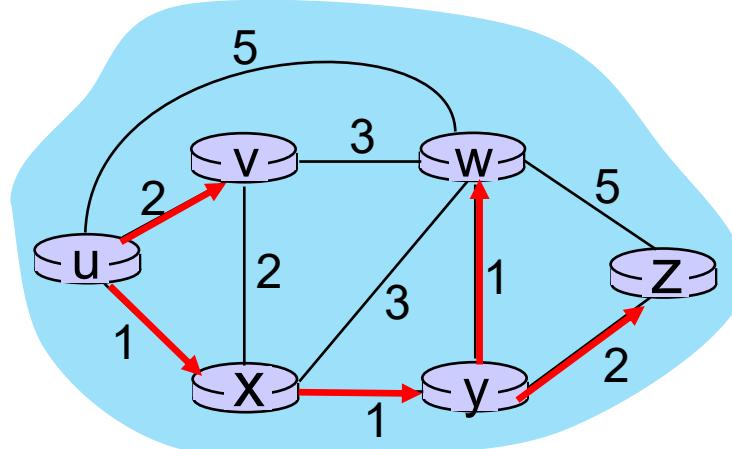
- 9 find  $a$  not in  $N'$  such that  $D(a)$  is a minimum  
 10 add  $a$  to  $N'$   
 11 update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$ :

$$D(b) = \min ( D(b), D(a) + c_{a,b} )$$

$$D(z) = \min ( D(z), D(w) + c_{w,z} ) = \min ( 4, 3+5 ) = 4$$

# Dijkstra's algorithm: an example

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



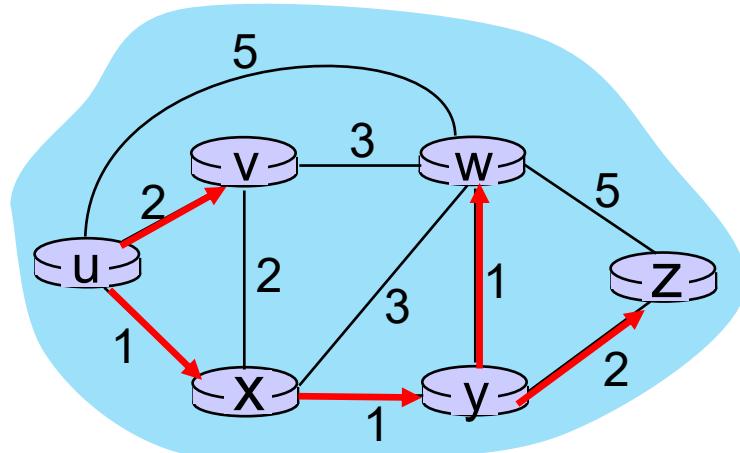
8 Loop

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

10 add  $a$  to  $N'$

# Dijkstra's algorithm: an example

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

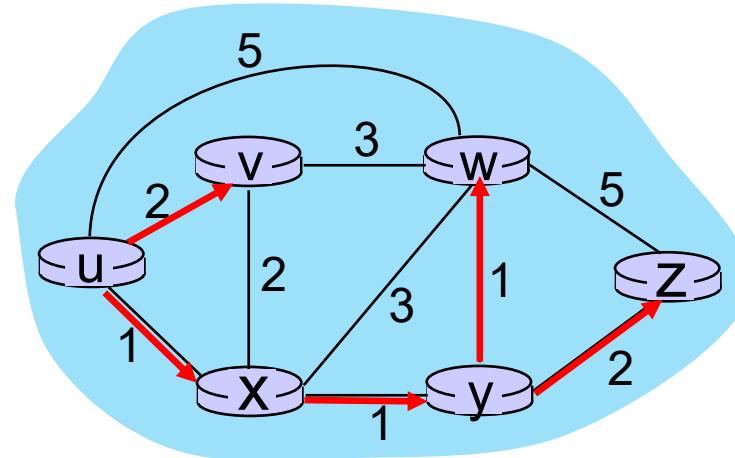


## 8 Loop

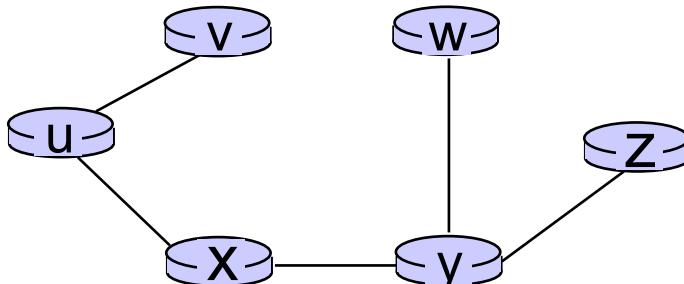
- 9 find  $a$  not in  $N'$  such that  $D(a)$  is a minimum
- 10 add  $a$  to  $N'$
- 11 update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$ :  

$$D(b) = \min ( D(b), D(a) + c_{a,b} )$$

# Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



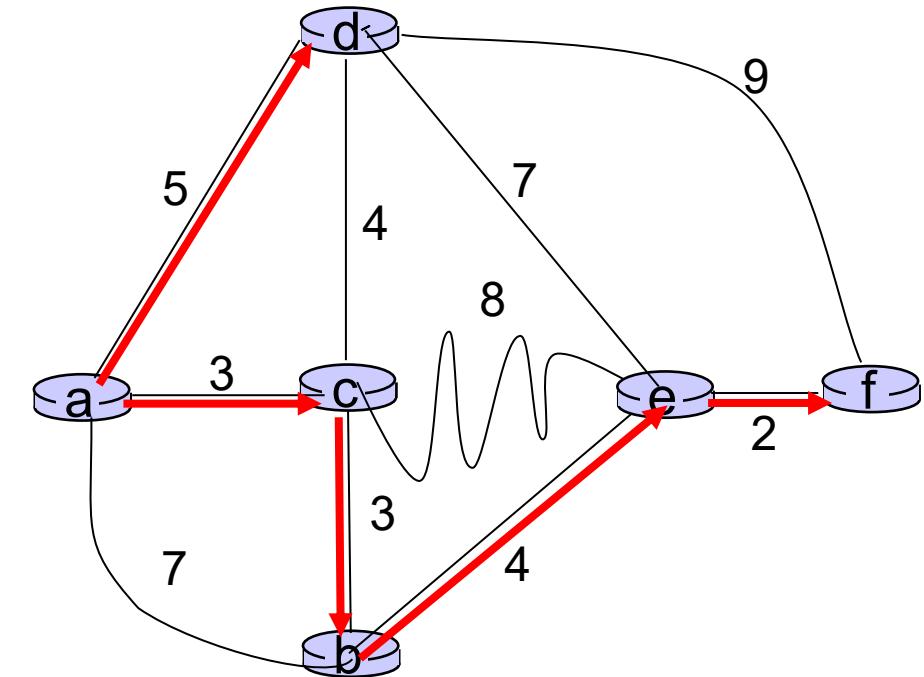
resulting forwarding table in u:

| destination | outgoing link |
|-------------|---------------|
| v           | (u,v)         |
| x           | (u,x)         |
| y           | (u,x)         |
| w           | (u,x)         |
| x           | (u,x)         |

route from  $u$  to  $v$  directly  
route from  $u$  to all other destinations via  $x$

# Dijkstra's algorithm: another example

| Step | $N'$   | $b$          | $c$          | $d$          | $e$          | $f$          |
|------|--------|--------------|--------------|--------------|--------------|--------------|
| 0    | a      | $D(b), p(b)$ | $D(c), p(c)$ | $D(d), p(d)$ | $D(e), p(e)$ | $D(f), p(f)$ |
| 1    | ac     | 7,a          | 3,a          | 5,a          | $\infty$     | $\infty$     |
| 2    | acd    | 6,c          | 5,a          | 11,c         | $\infty$     |              |
| 3    | acdb   | 6,c          |              | 11,c         | 14,d         |              |
| 4    | acdbe  |              | 10,b         | 14,d         |              |              |
| 5    | acdbef |              |              | 12,e         |              |              |



notes:

- construct least-cost-path tree by tracing predecessor nodes

# Chapter 6

# The Link Layer and LANs

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 1996-2023  
J.F Kurose and K.W. Ross, All Rights Reserved

## Computer Networking

*A Top-Down Approach*

EIGHTH EDITION

James F. Kurose • Keith W. Ross



## Computer Networking: A Top-Down Approach

8<sup>th</sup> edition

Jim Kurose, Keith Ross  
Pearson, 2020

# Link layer and LANs: our goals

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

# Link layer, LANs: roadmap

## 6.1 Introduction

6.2 Error detection and correction

6.3 Multiple access protocols

6.4 LANs

    Addressing, ARP

    Ethernet

    Switches

    VLANs

6.6 Data center networking

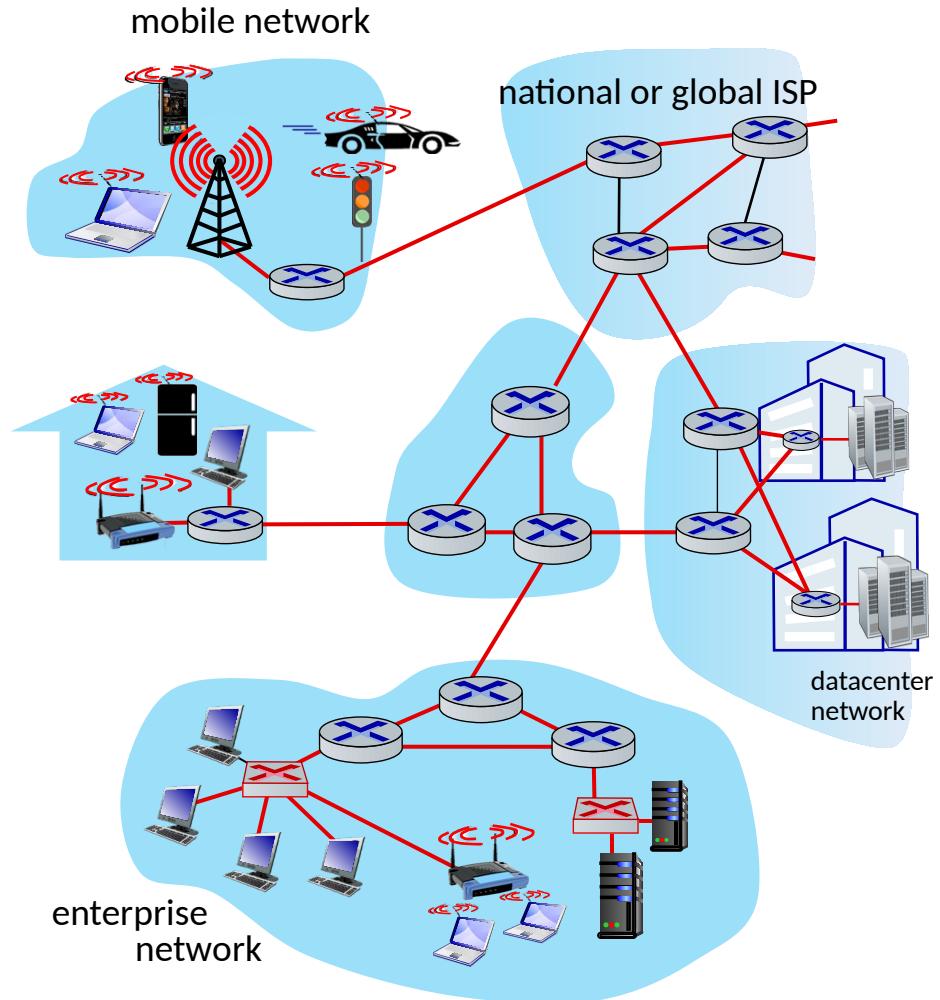
6.7 A day in the life of a web request

# Link layer: introduction

terminology:

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

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



# Link layer: services

- **framing:**

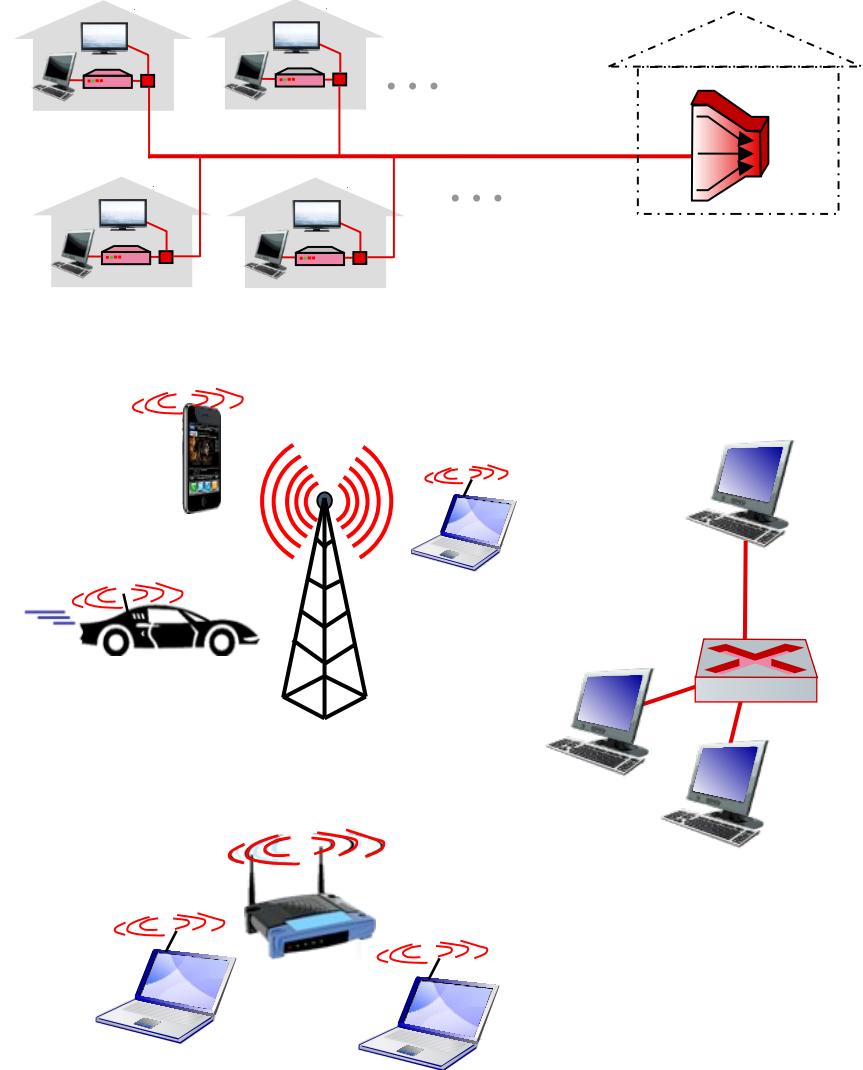
- encapsulate datagram into frame, adding header, trailer

- **link access**

- Medium access control (MAC) protocol for transmitting frames on a shared medium
- MAC addresses in frame headers identify source, destination (different from IP address!)

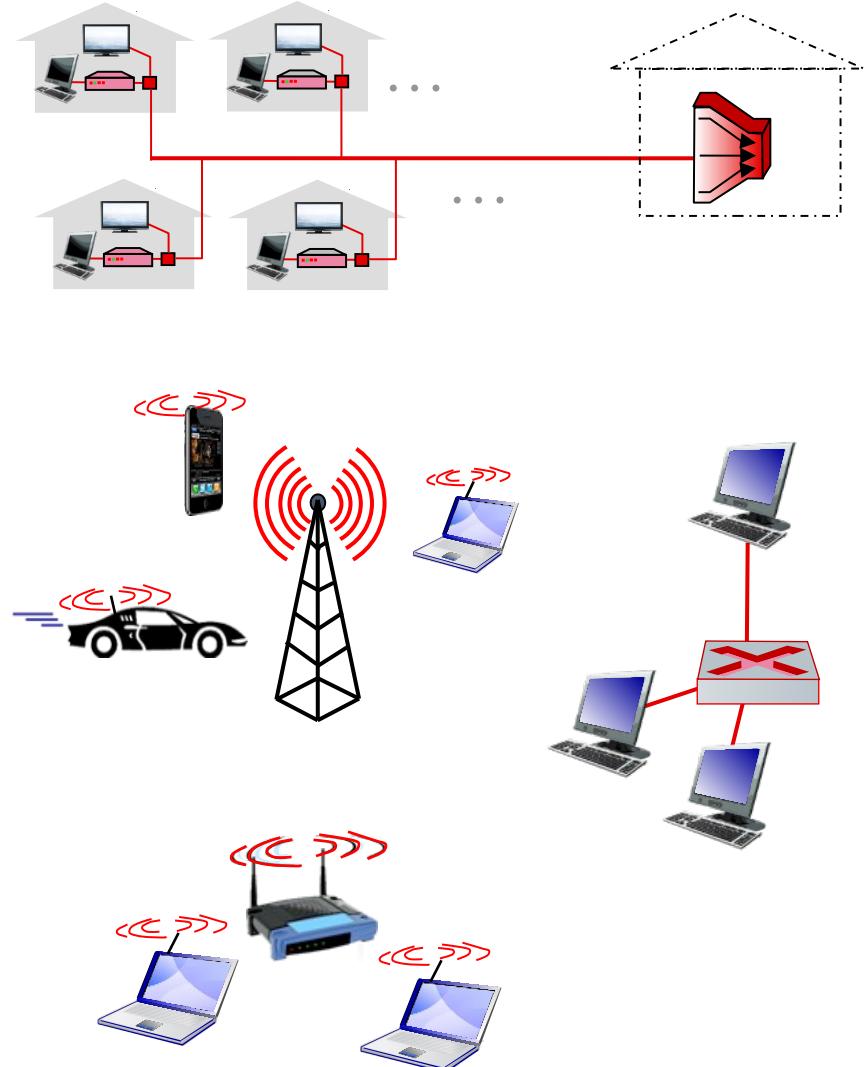
- **reliable delivery between adjacent nodes**

- we already know how to do this!
- seldom used on low bit-error links
- wireless links: high error rates
  - Q: why both link-level and end-end reliability?



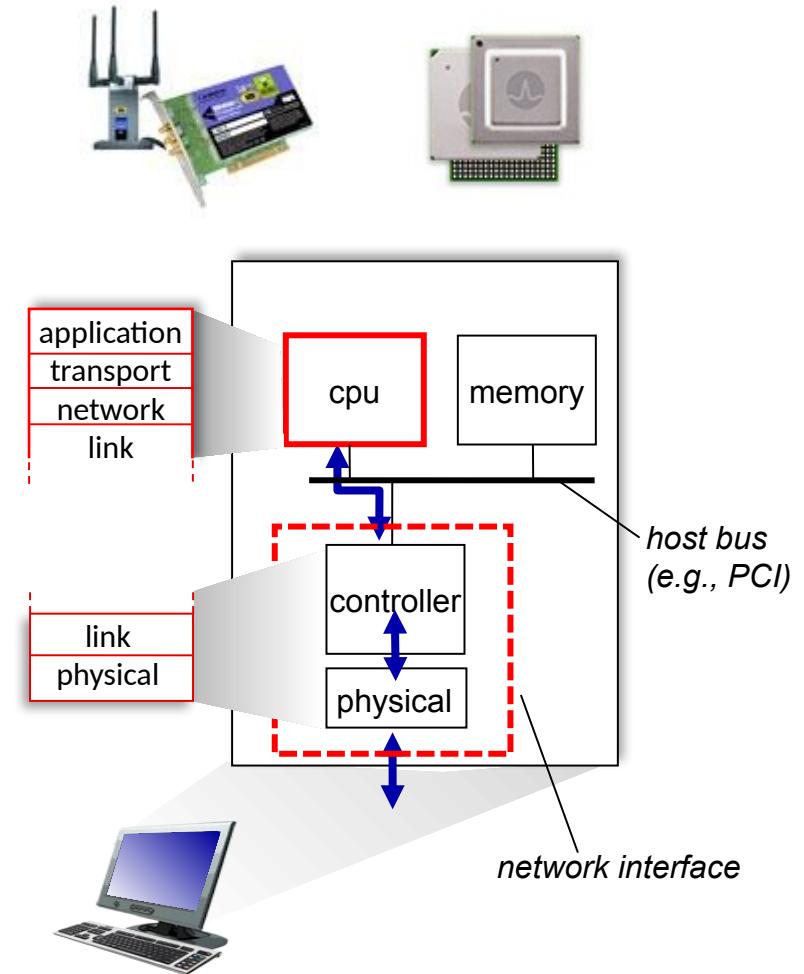
# Link layer: services (more)

- **flow control:**
  - pacing between adjacent sending and receiving nodes
- **error detection:**
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- **error correction:**
  - receiver identifies *and corrects* bit error(s) without retransmission



# Host link-layer implementation

- in each-and-every host
- link layer implemented on-chip or in network interface card (NIC)
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



# **Link layer, LANs: roadmap**

6.1 Introduction

**6.2 Error detection and correction**

6.3 Multiple access protocols

6.4 LANs

addressing, ARP

Ethernet

switches

VLANs

6.6 Data center networking

6.7 A day in the life of a web request

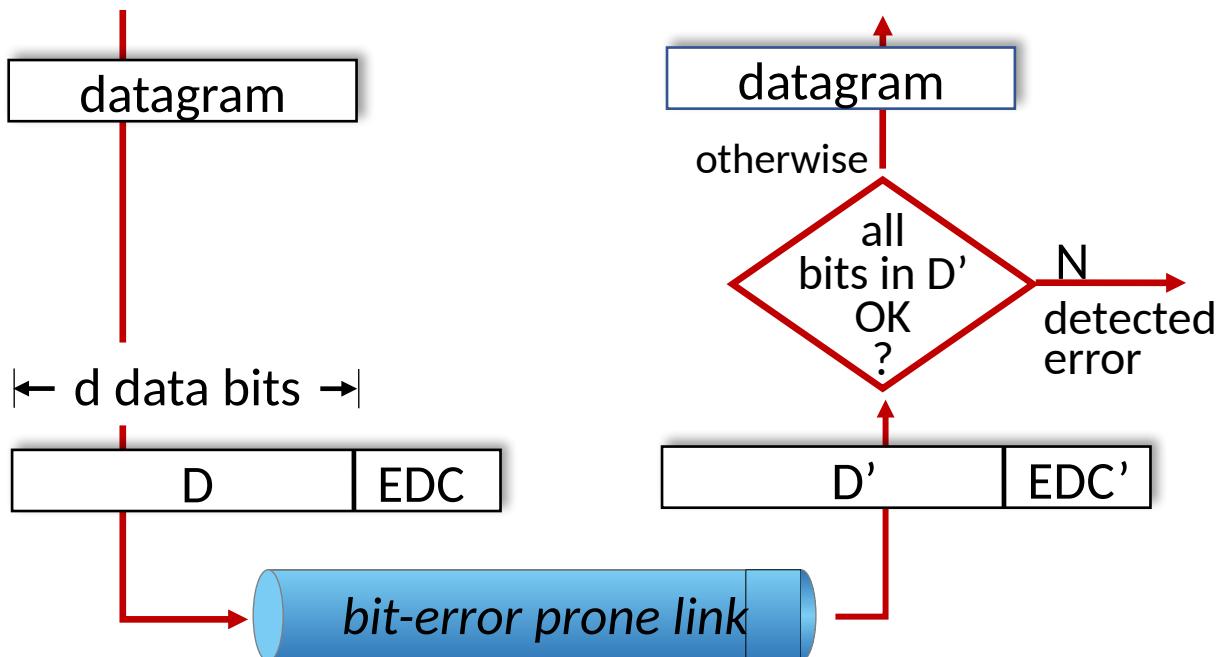
# Error detection principles

- Checksum
  - UDP checksum 16 bits (chapter 3.3.2)
  - TCP checksum 16 bits (chapter 3.5.2)
  - IP *header* checksum 16 bits (chapter 4.3.1)
  - ICMP checksum 16 bits (chapter 5.6)
  - and other protocols
- Parity checking
- Cyclic redundancy check
  - Ethernet

# Error detection

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



Error detection not 100% reliable!

- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

# Parity checking

## single bit parity:

- detect single bit errors

|                  |   |
|------------------|---|
| 0111000110101011 | 1 |
|------------------|---|

←  $d$  data bits → |  
                          parity bit

Even/odd parity: set parity bit so there is an even/odd number of 1's

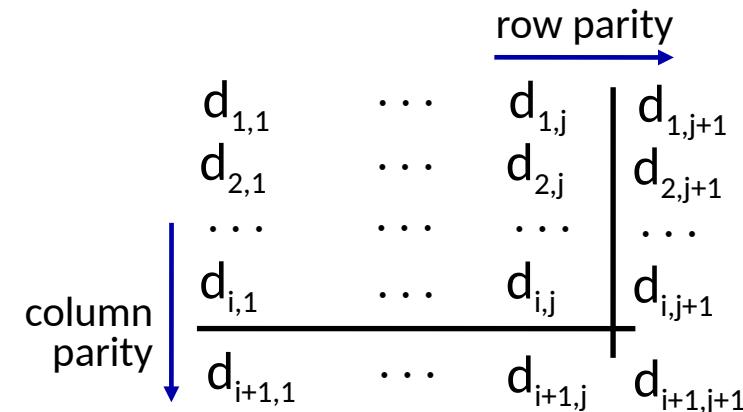
## At receiver:

- compute parity of  $d$  received bits
- compare with received parity bit
  - if different than error detected



Can detect *and* correct errors (without retransmission!)

- two-dimensional parity: detect *and correct* single bit errors



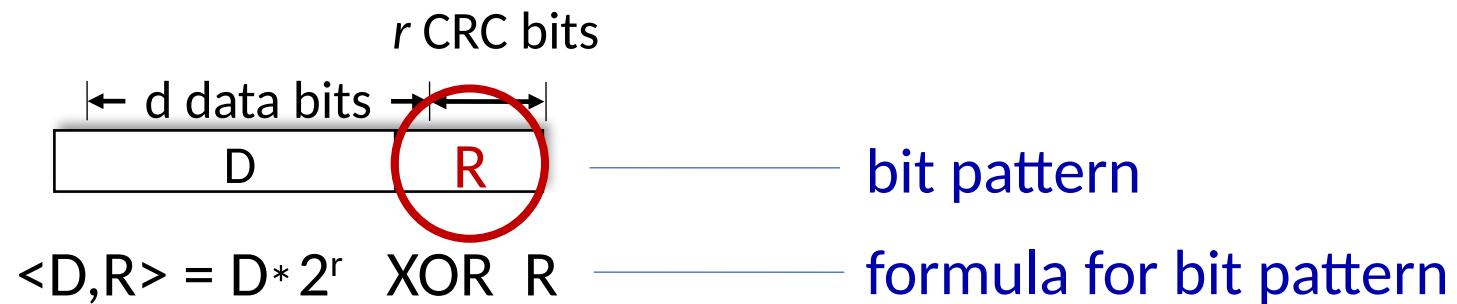
|            |               |
|------------|---------------|
| no errors: | 1 0 1 0 1   1 |
|            | 1 1 1 1 0   0 |
|            | 0 1 1 1 0   1 |
|            | 1 0 1 0 1   0 |

detected  
and  
correctable  
single-bit  
error:

|               |
|---------------|
| 1 0 1 0 1   1 |
| 1 0 1 1 0   0 |
| 0 1 1 1 0   1 |
| 1 0 1 0 1   0 |

# Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- **D**: data bits (given, think of these as a binary number)
- **G**: bit pattern (generator), of  $r+1$  bits (given)



goal: choose  $r$  CRC bits, **R**, such that  $\langle D, R \rangle$  exactly divisible by **G** ( $\text{mod } 2$ )

- receiver knows **G**, divides  $\langle D, R \rangle$  by **G**. If non-zero remainder: error detected!
- can detect all burst errors less than  $r+1$  bits
- widely used in practice (Ethernet, 802.11 WiFi)

# Cyclic Redundancy Check (CRC): example

Sender wants to compute R such that:

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

... or equivalently (XOR R both sides):

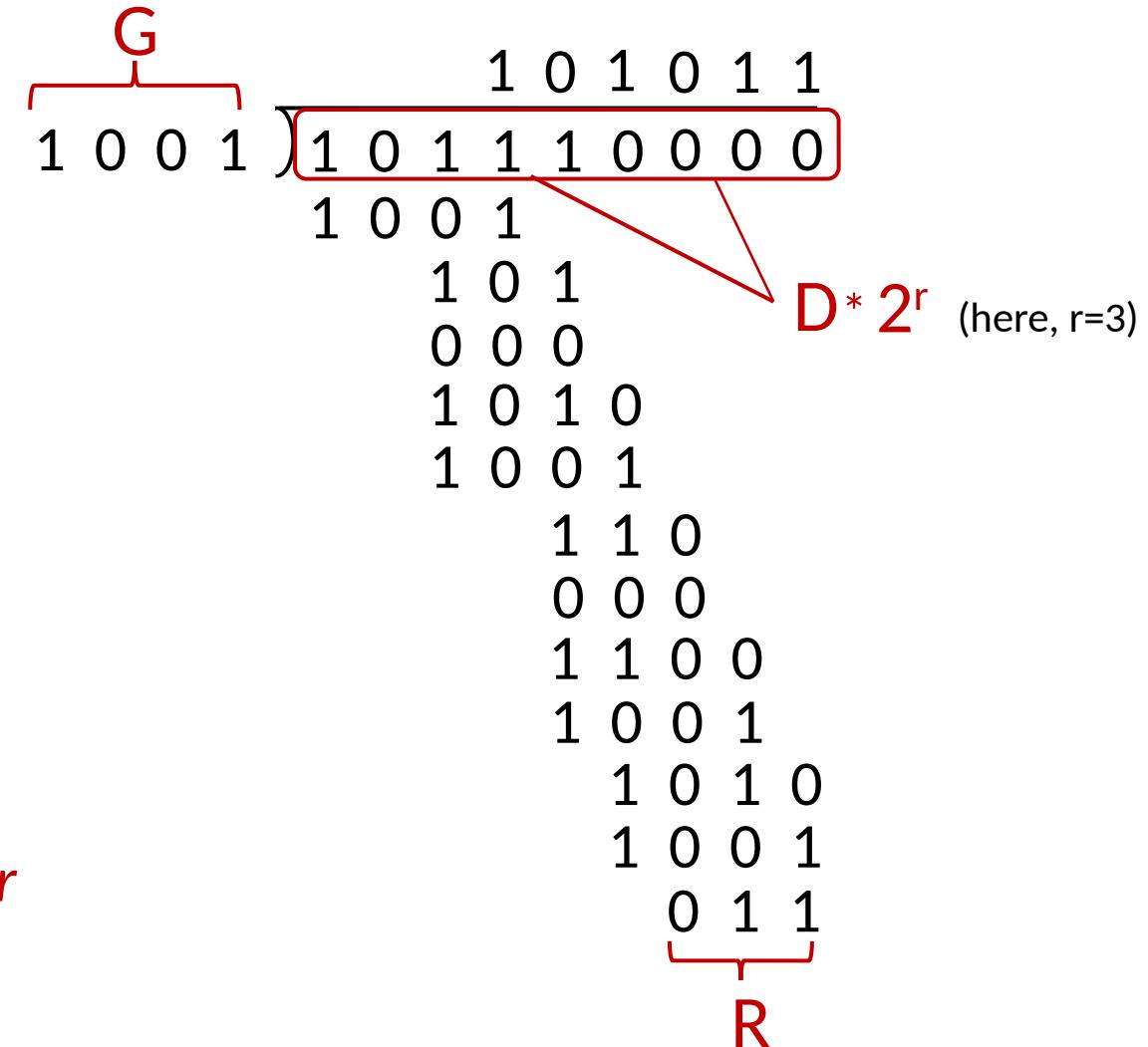
$$D \cdot 2^r = nG \text{ XOR } R$$

... which says:

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

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

algorithm for computing R



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

# **Link layer, LANs: roadmap**

6.1 Introduction

6.2 Error detection, correction

## **6.3 Multiple access protocols**

6.4 LANs

addressing, ARP

Ethernet

switches

VLANs

6.6 Data center networking

6.7 A day in the life of a web request

# Multiple access links, protocols

two types of “links”:

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-fashioned Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/5G, satellite



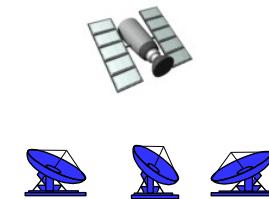
shared wire (e.g.,  
cabled Ethernet)



shared radio: 4G/5G



shared radio: WiFi



shared radio: satellite



humans at a cocktail party  
(shared air, acoustical)

# Multiple access channel protocols

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

## multiple access protocol

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

# MAC protocols: taxonomy

three broad classes:

## 1. Channel partitioning protocols

- divide channel into smaller “pieces” (time slots, frequency, code)
- allocate piece to node for exclusive use

## 2. Random access protocols

- channel not divided, allow collisions
- “recover” from collisions

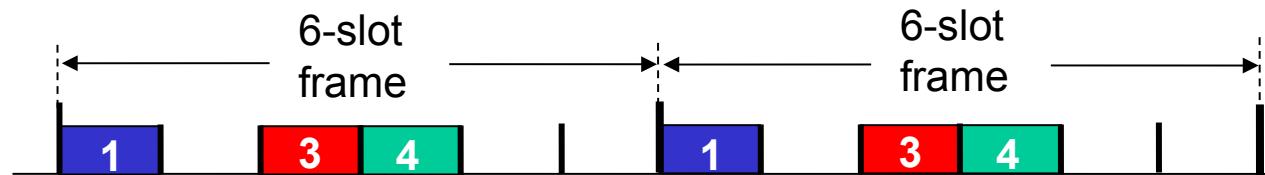
## 3. “Taking turns” protocols

- nodes take turns, but nodes with more to send can take longer turns

# Channel partitioning MAC protocols: TDMA

## TDMA: time division multiple access

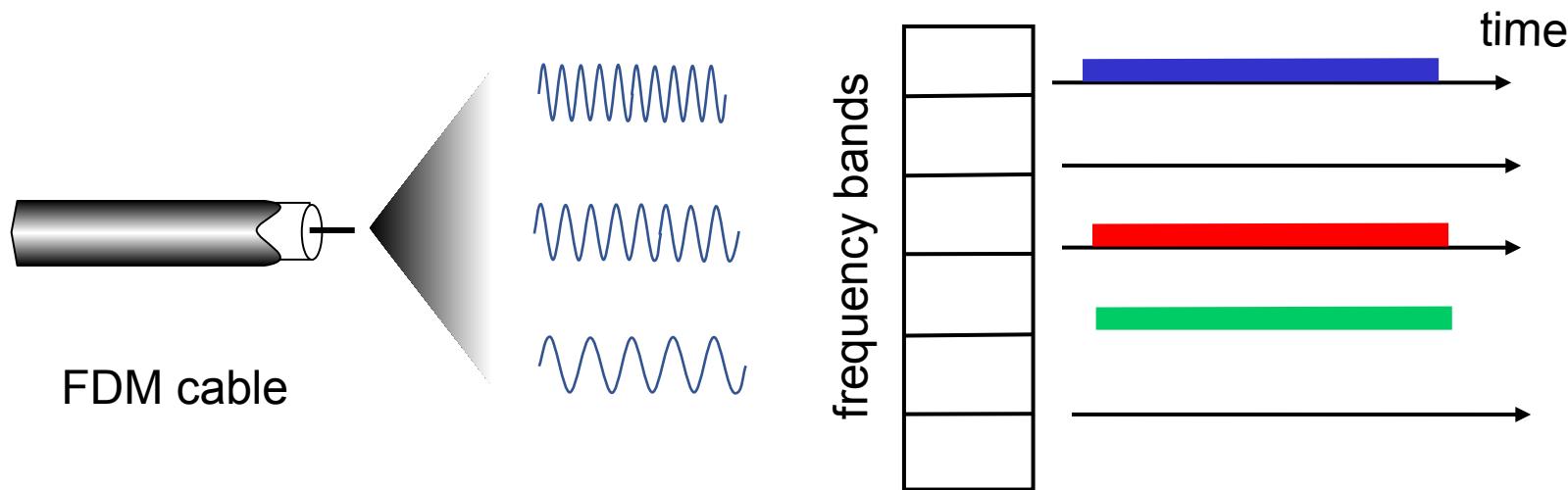
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



# Channel partitioning MAC protocols: FDMA

## FDMA: frequency division multiple access

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



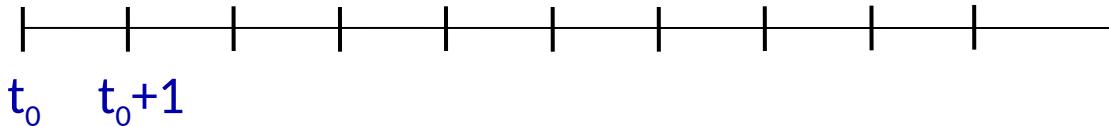
# MAC protocols: taxonomy

1. Channel partitioning protocols
2. Random access protocols
3. “Taking turns” protocols

# Random access protocols

- When node has packet to send
  - transmit at full channel data rate
  - no *a priori* coordination among nodes
- Two or more transmitting nodes: “collision”
- **Random access MAC protocol** specifies:
  - how to recover from collisions (e.g., via delayed retransmissions)
- Examples of random-access MAC protocols:
  - Slotted ALOHA
  - Carrier sense multiple access

# Slotted ALOHA



## assumptions:

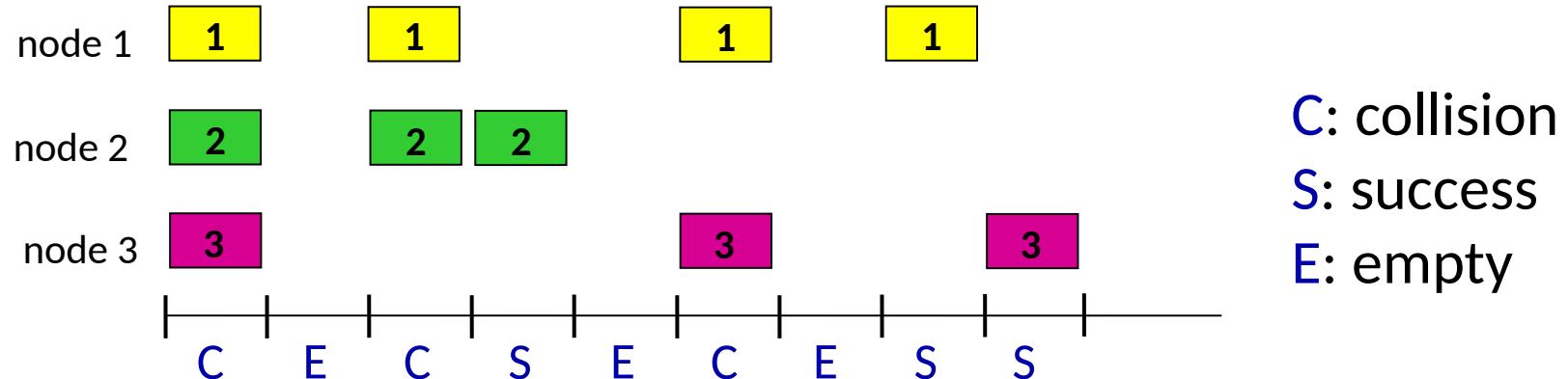
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

## operation:

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

randomization – why?

# Slotted ALOHA



## Pros:

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

## Cons:

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

# Slotted ALOHA: efficiency

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

- suppose:  $N$  nodes with many frames to send, each transmits in slot with probability  $p$ 
  - prob that given node has success in a slot =  $p(1-p)^{N-1}$
  - prob that *any* node has a success =  $Np(1-p)^{N-1}$
  - max efficiency: find  $p^*$  that maximizes  $Np(1-p)^{N-1}$
  - for many nodes, take limit of  $Np^*(1-p^*)^{N-1}$  as  $N$  goes to infinity, gives:  
*max efficiency =  $1/e = .37$*
- *at best:* channel used for useful transmissions 37% of time!



# CSMA (carrier sense multiple access)

simple **CSMA**: listen before transmit:

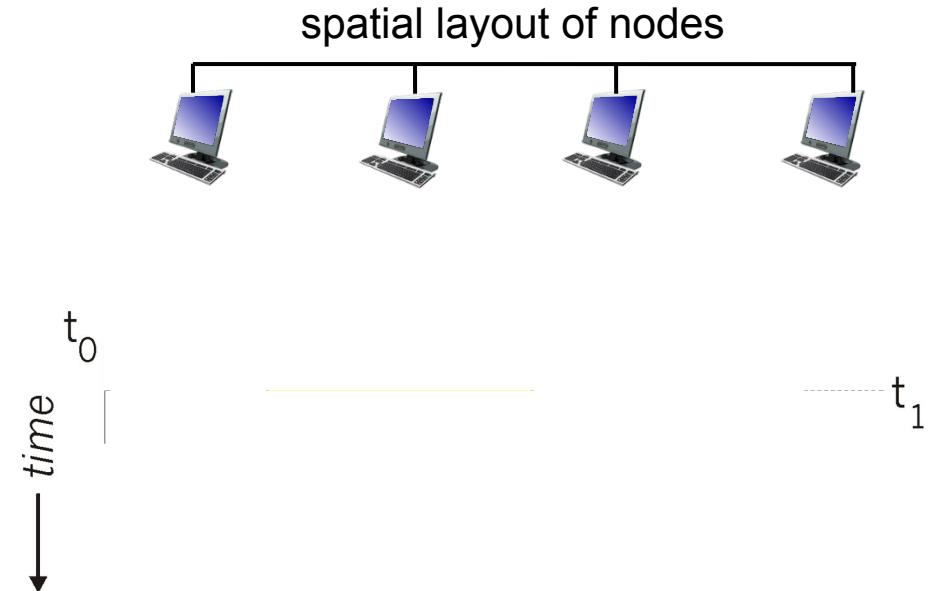
- if channel sensed idle: transmit entire frame
- if channel sensed busy: defer transmission
- human analogy: don't interrupt others!

**CSMA/CD**: CSMA with *collision detection*

- collisions detected within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless
- human analogy: the polite conversationalist

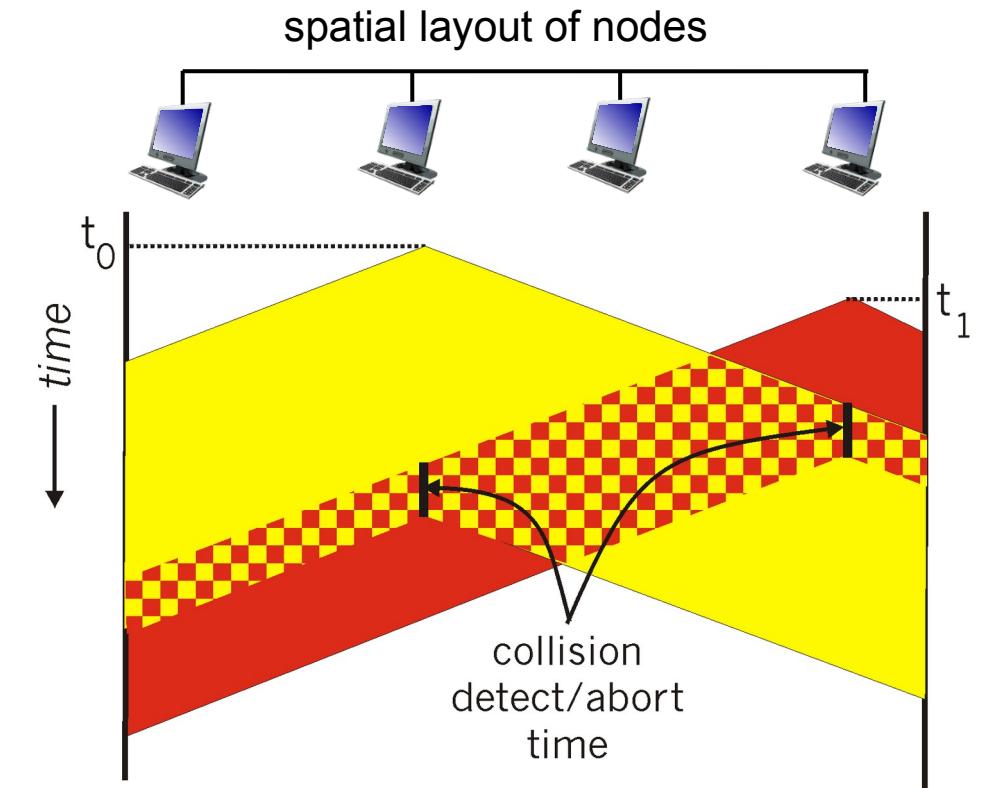
# CSMA: collisions

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



# CSMA/CD:

- CSMA/CD reduces the amount of time wasted in collisions
  - transmission aborted on collision detection



# Ethernet CSMA/CD algorithm

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

*Ethernet CSMA/CD was used in now-obsolete shared media Ethernet variants*

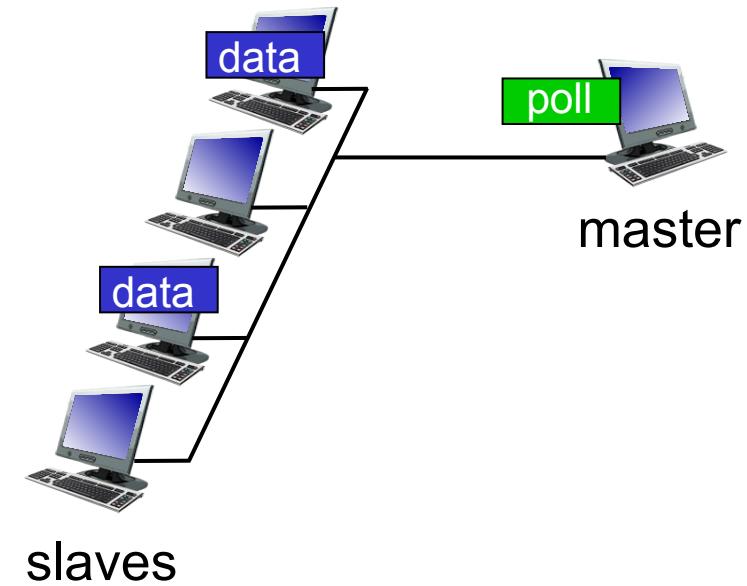
# MAC protocols: taxonomy

1. Channel partitioning protocols
2. Random access protocols
3. “Taking turns” protocols

# “Taking turns” MAC protocols

## polling:

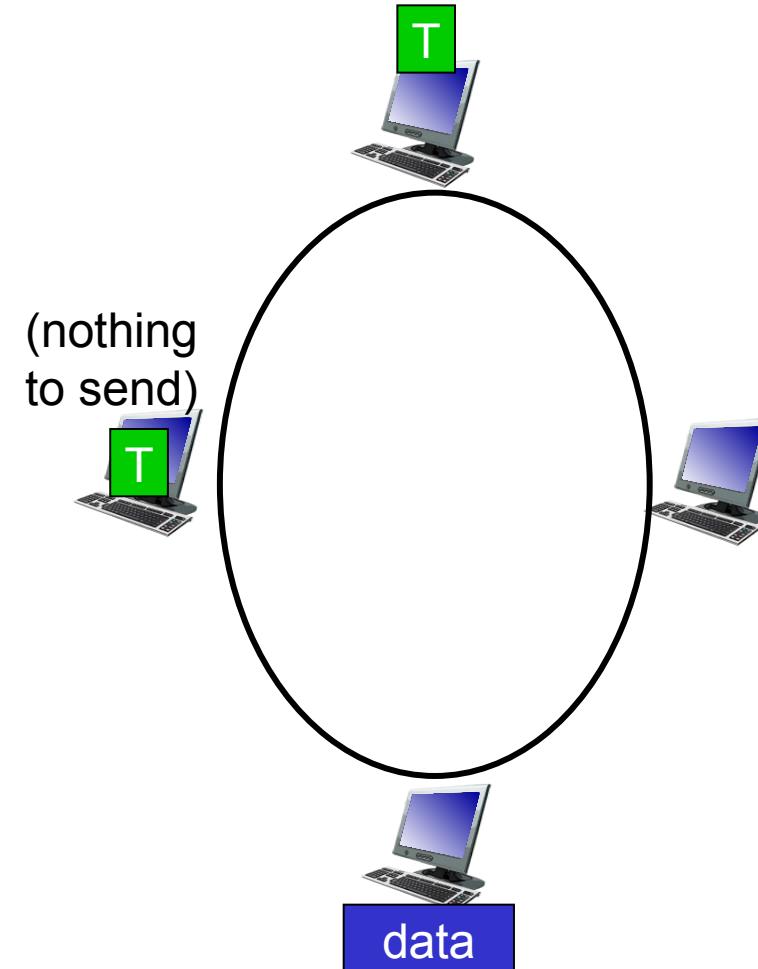
- master node “invites” other nodes to transmit in turn
- typically used with “dumb” devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



# “Taking turns” MAC protocols

## token ring:

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



# “Taking turns” MAC protocols

## channel partitioning MAC protocols:

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

## random access MAC protocols

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

## “taking turns” protocols

- look for best of both worlds!

# Summary of MAC protocols

- **Channel partitioning**, by time, frequency or code
  - Time Division, Frequency Division
- **Random access (dynamic)**,
  - CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in obsolete Ethernet-variant
  - CSMA/CA used in 802.11
- **Taking turns**
  - polling from central site, token passing
  - Bluetooth, token ring

# Link layer, LANs: roadmap

6.1 Introduction

6.2 Error detection and correction

6.3 Multiple access protocols

## 6.4 LANs

6.4.1 Addressing, ARP

6.4.2 Ethernet

6.4.3 Switches

6.4.4 VLANs

6.6 Data center networking

6.7 A day in the life of a web request

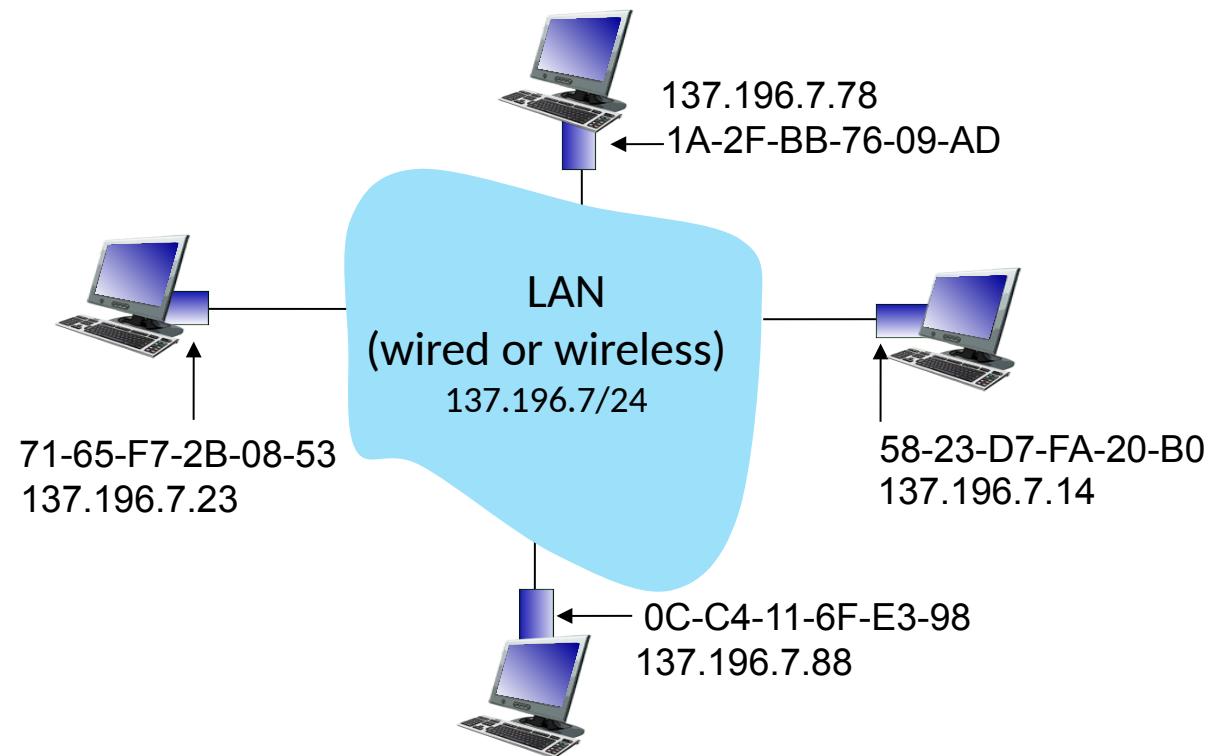
# MAC addresses

- 32-bit IP address:
    - *network-layer* address for interface
    - used for layer 3 (network layer) forwarding
    - e.g.: 128.119.40.136
  - MAC (or LAN or physical or Ethernet) address:
    - function: used “locally” to get frame from one interface to another physically-connected interface (same subnet, in IP-addressing sense)
    - 48-bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
    - e.g.: 1A-2F-BB-76-09-AD
- hexadecimal (base 16) notation  
(each “numeral” represents 4 bits)

# MAC addresses

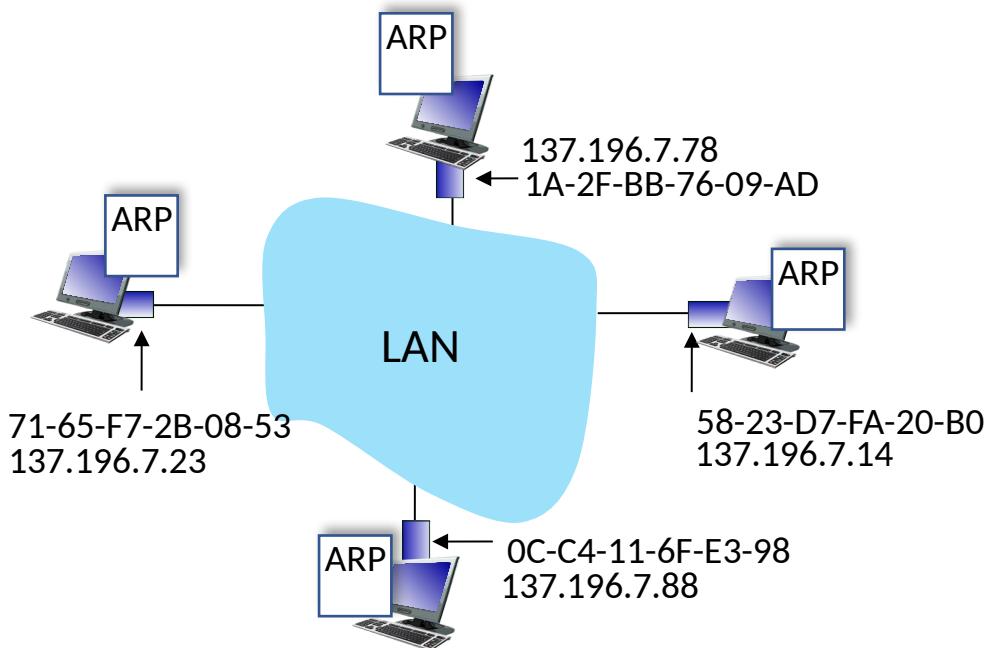
each interface on LAN

- has unique 48-bit **MAC** address
- has a locally unique 32-bit IP address (as we've seen)



# ARP: address resolution protocol

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



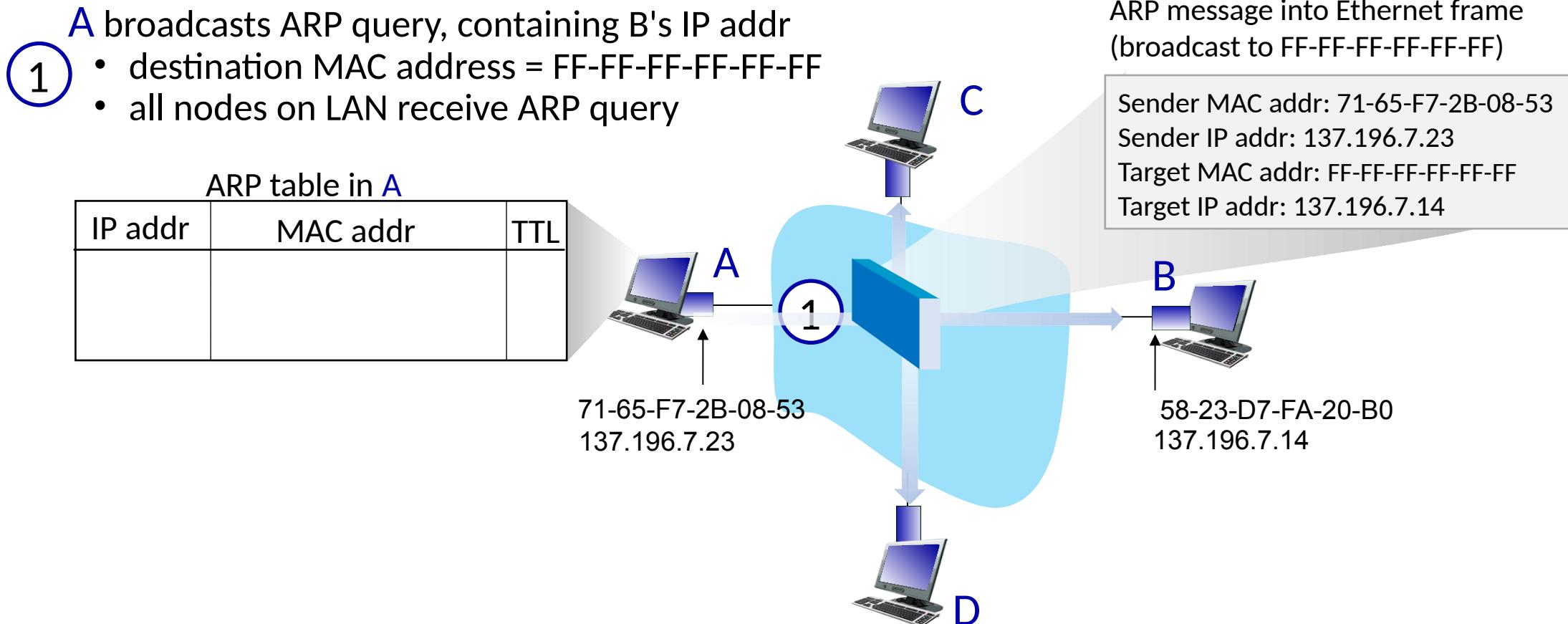
**ARP table:** each IP node (host, router) on LAN has table

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

# ARP protocol in action

example: A wants to send datagram to B

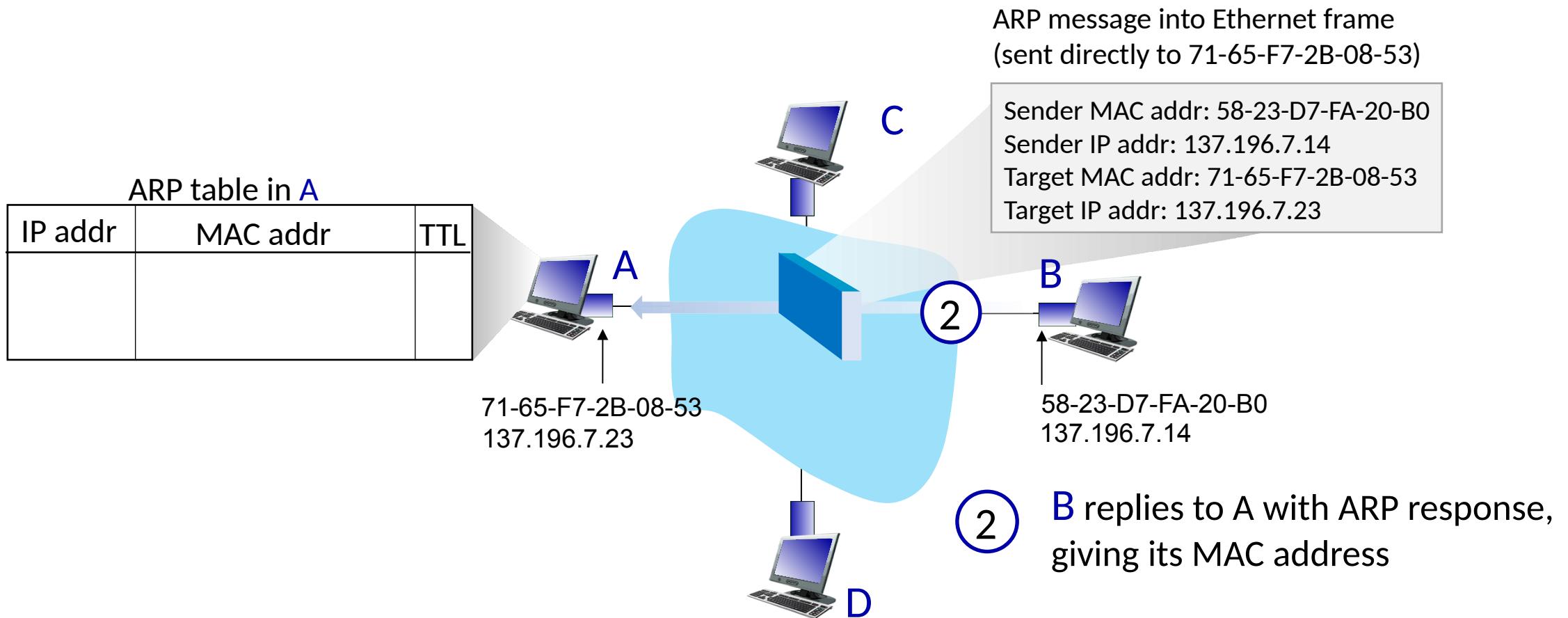
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# ARP protocol in action

example: A wants to send datagram to B

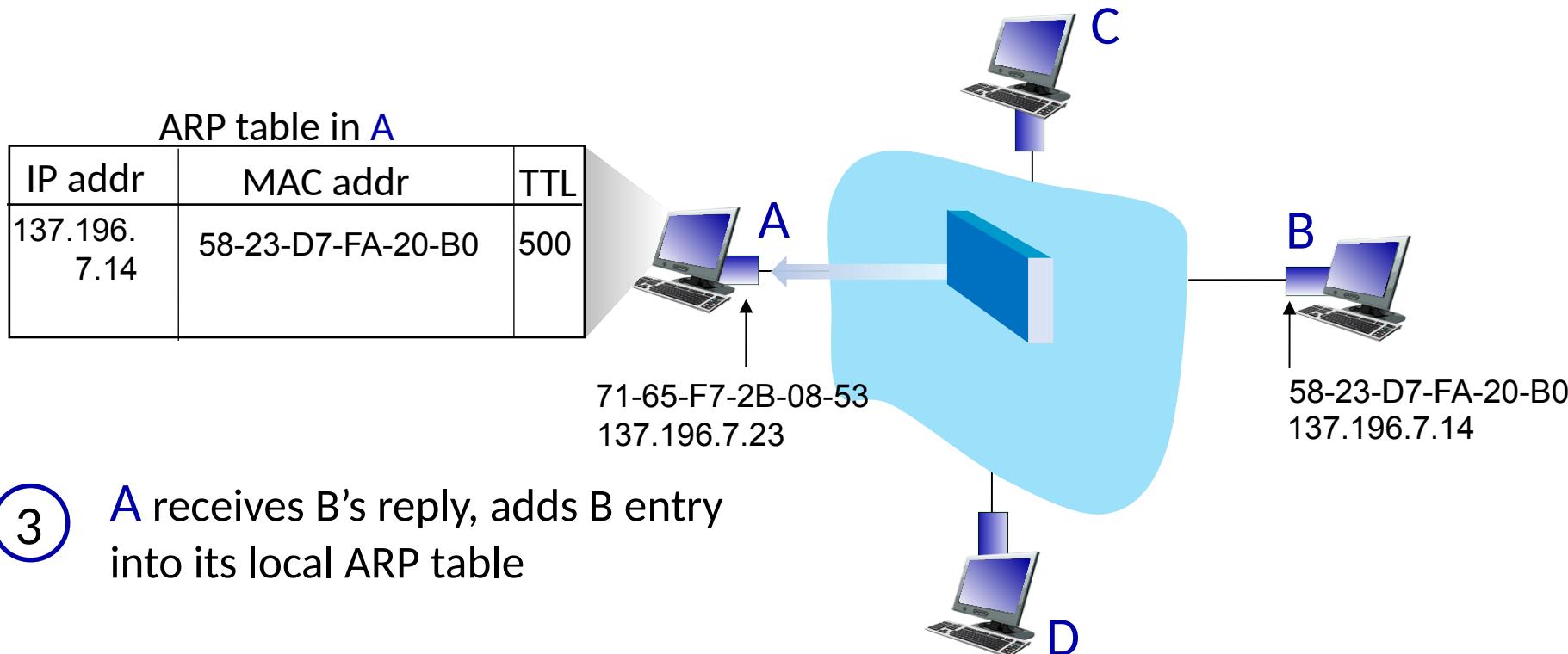
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# ARP protocol in action

example: A wants to send datagram to B

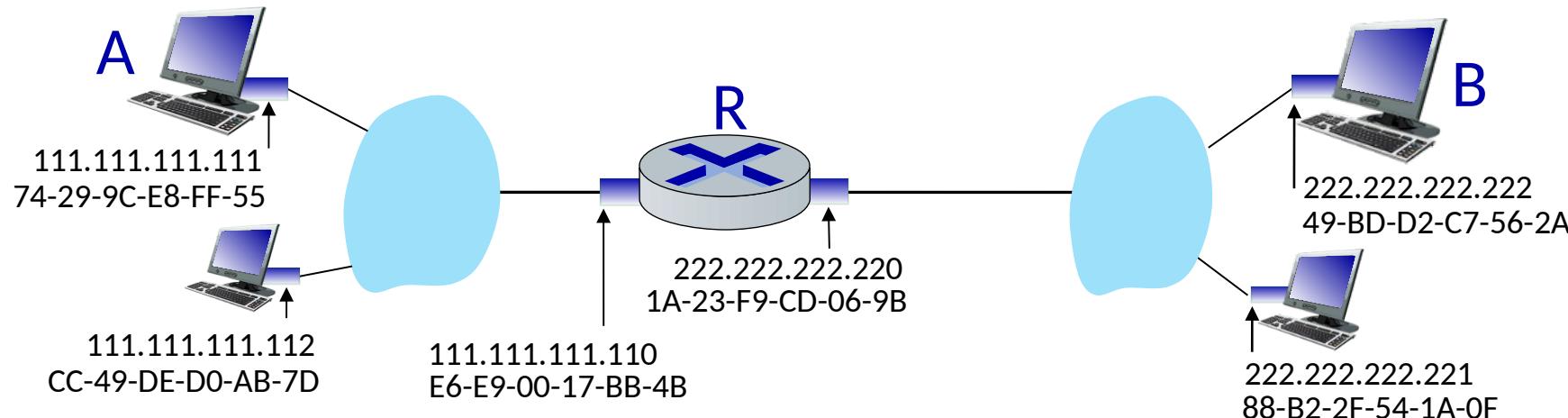
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# Routing to another subnet: addressing

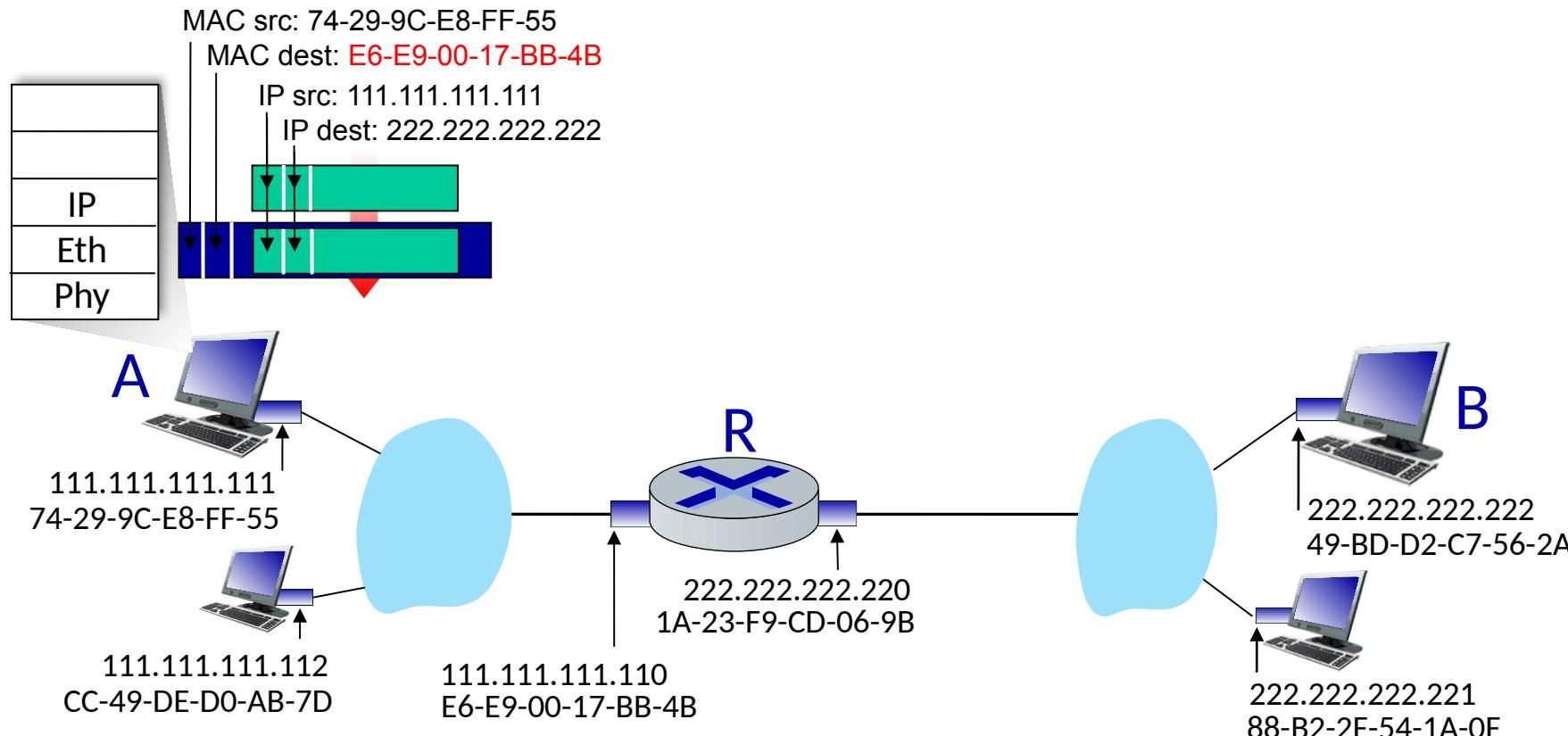
walkthrough: sending a datagram from A to B via R

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



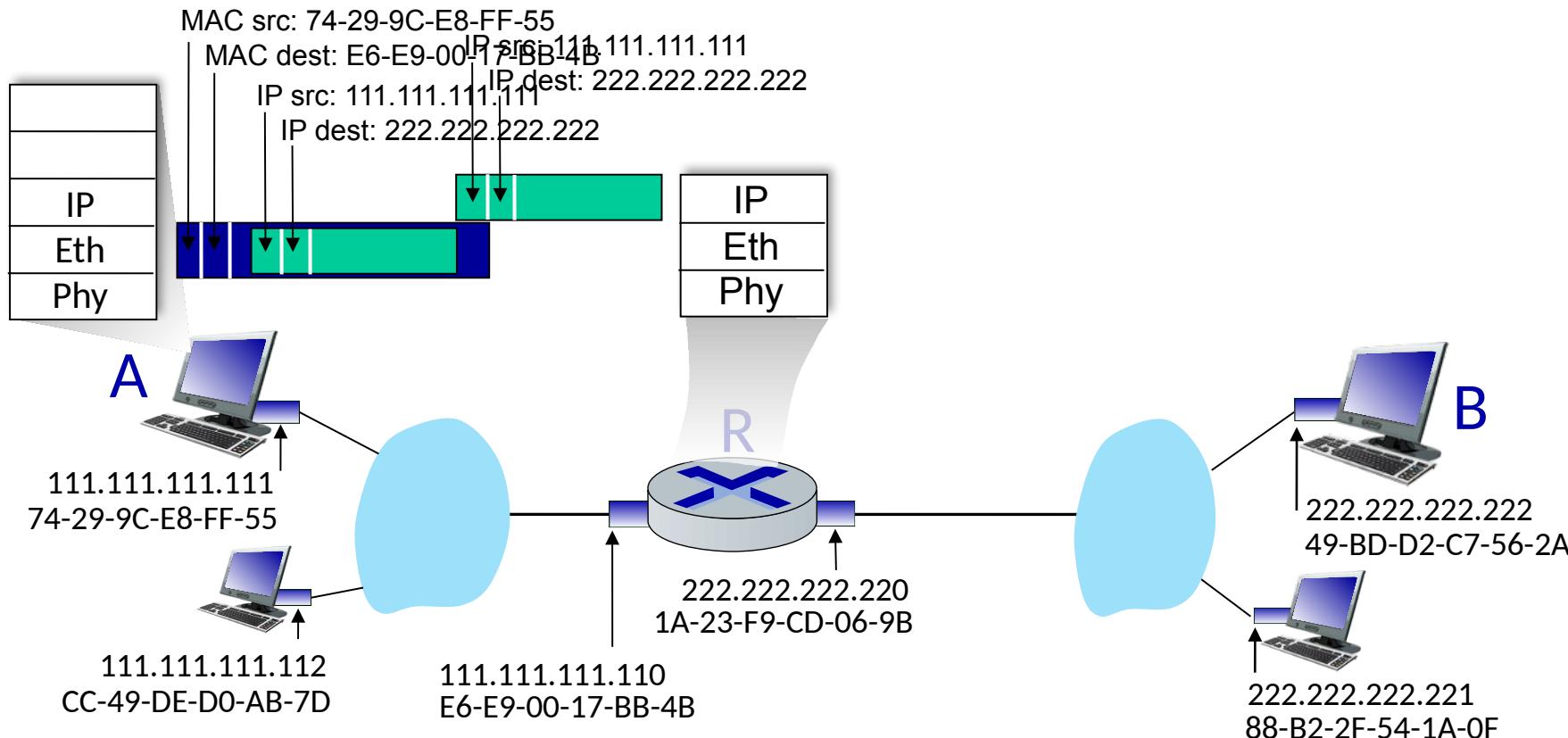
# Routing to another subnet: addressing

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame containing A-to-B IP datagram
  - R's MAC address is frame's destination



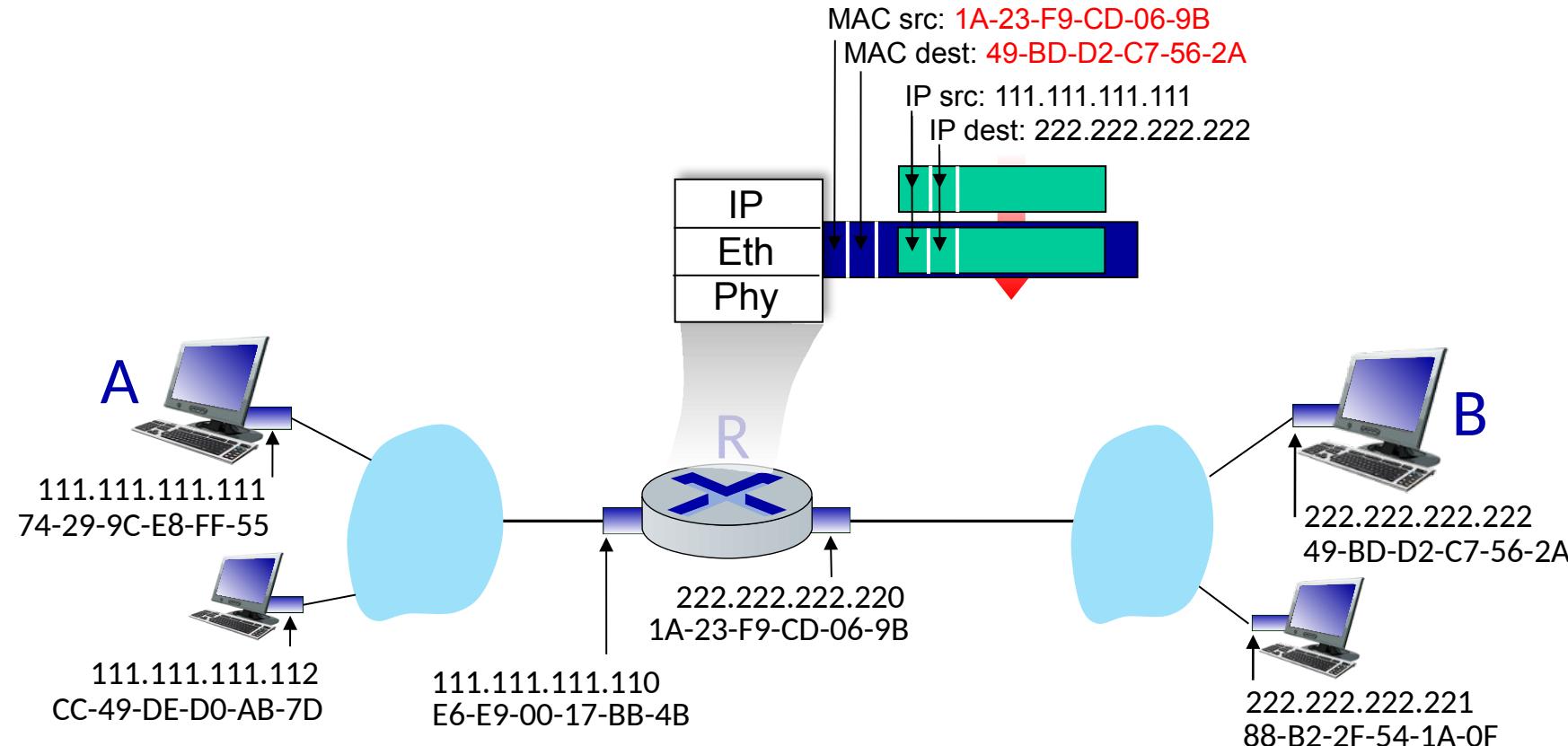
# Routing to another subnet: addressing

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



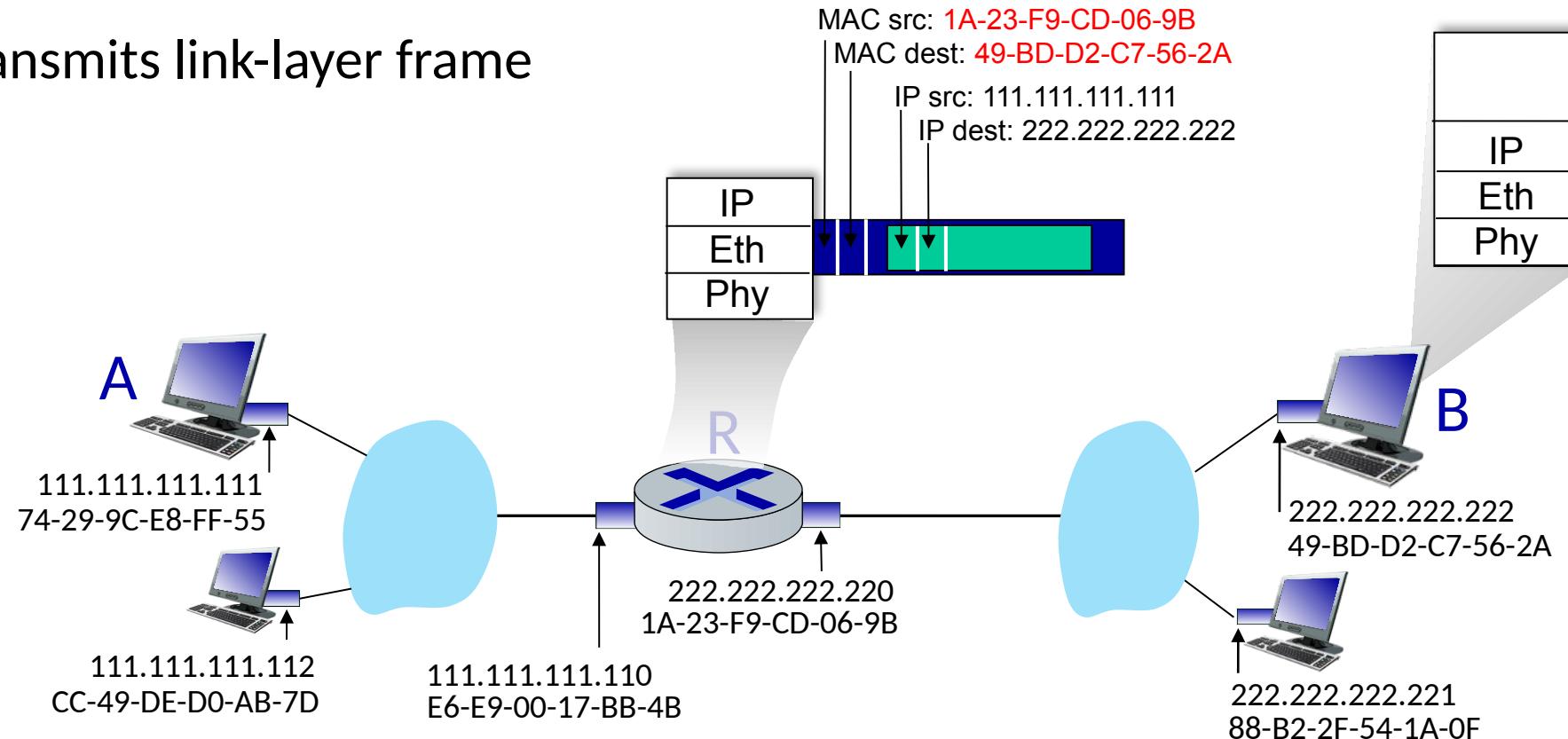
# Routing to another subnet: addressing

- Router determines outgoing interface, passes datagram with IP source A, destination B to link layer
- Router creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address



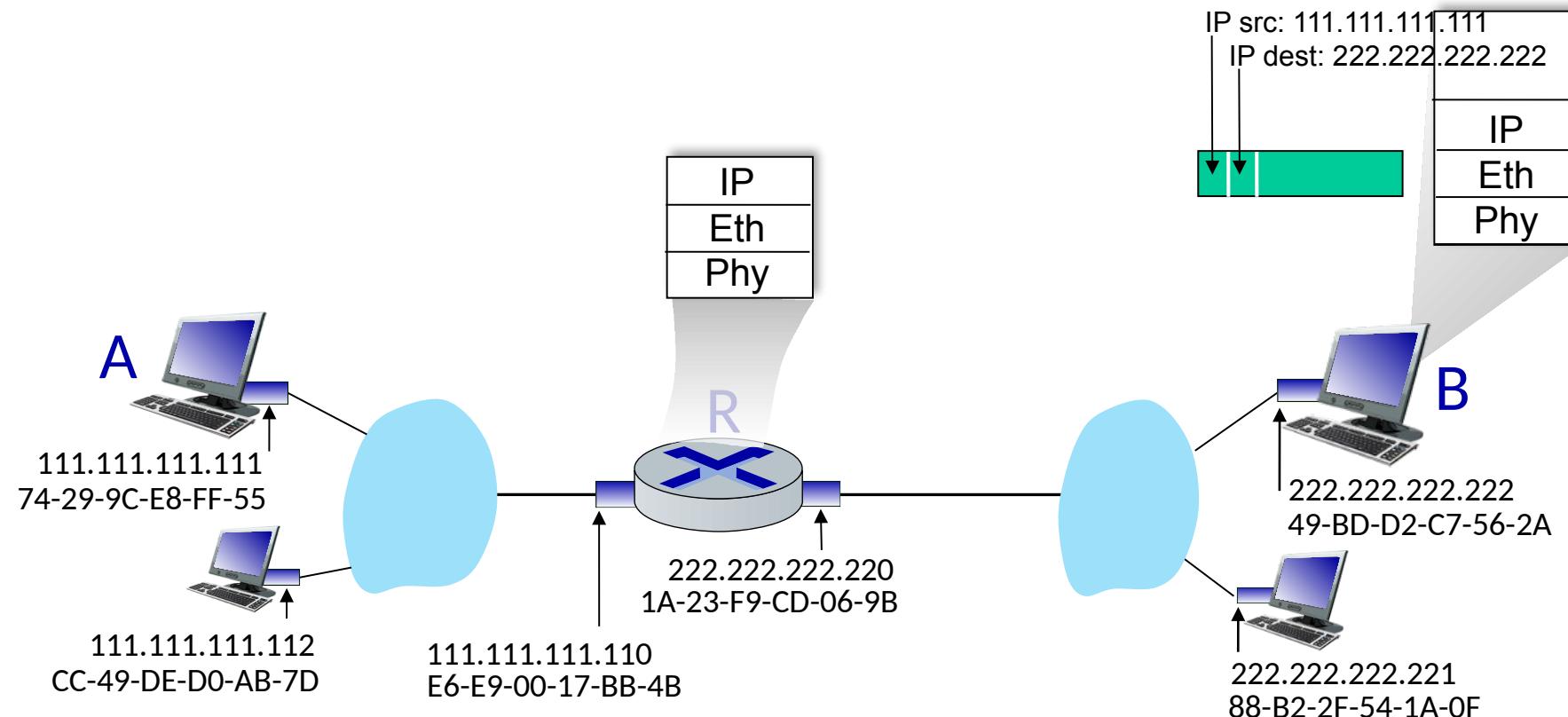
# Routing to another subnet: addressing

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address
- transmits link-layer frame



# Routing to another subnet: addressing

- B receives frame, extracts IP datagram destination B
- B passes datagram up protocol stack to IP



# Link layer, LANs: roadmap

6.1 Introduction

6.2 Error detection and correction

6.3 Multiple access protocols

## 6.4 LANs

6.4.1 Addressing, ARP

### 6.4.2 Ethernet

6.4.3 switches

6.4.4 VLANs

6.6 Data center networking

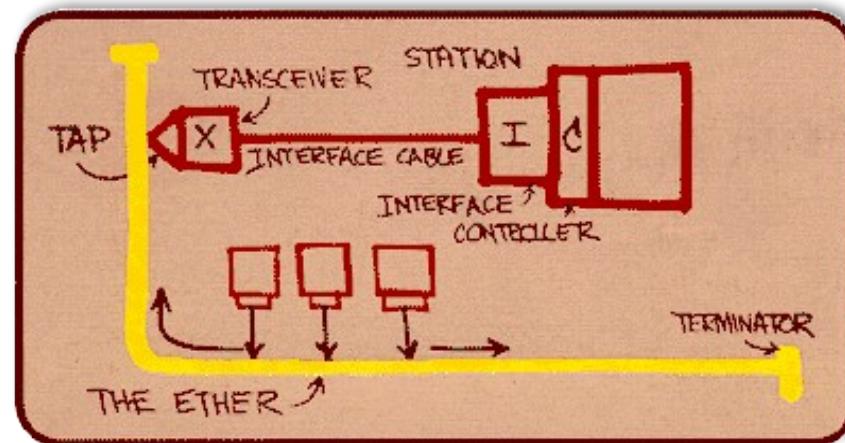
6.7 A day in the life of a web request

# Ethernet

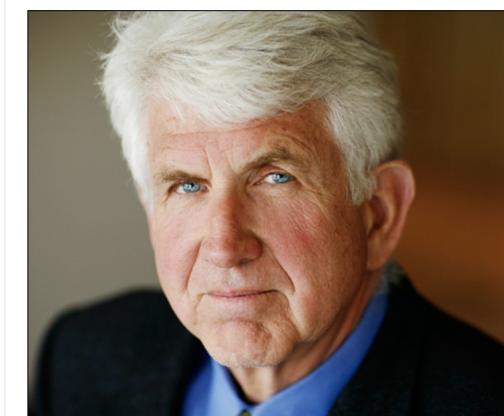
“dominant” wired LAN technology:

- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 400 Gbps
- single chip, multiple speeds (e.g., Broadband Ethernet)

Metcalfe's Ethernet sketch



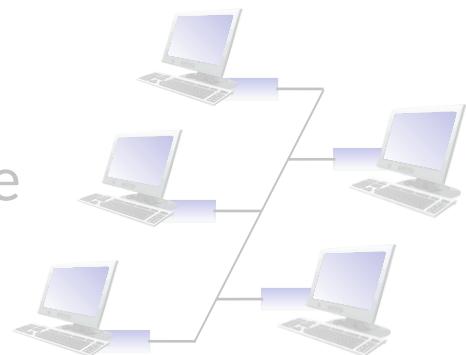
BOB METCALFE  
Bob Metcalfe: Ethernet co-inventor,  
2022 ACM Turing Award recipient



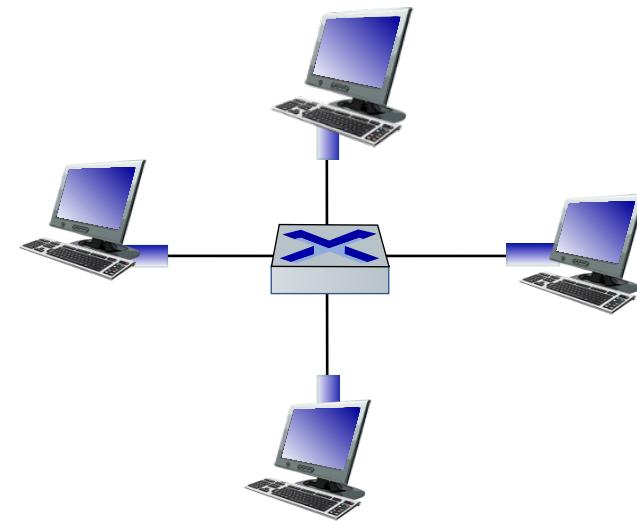
# Ethernet: physical topology

- **bus:** popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- **switched:** prevails today
  - active link-layer 2 *switch* in center
  - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)

bus: coaxial cable

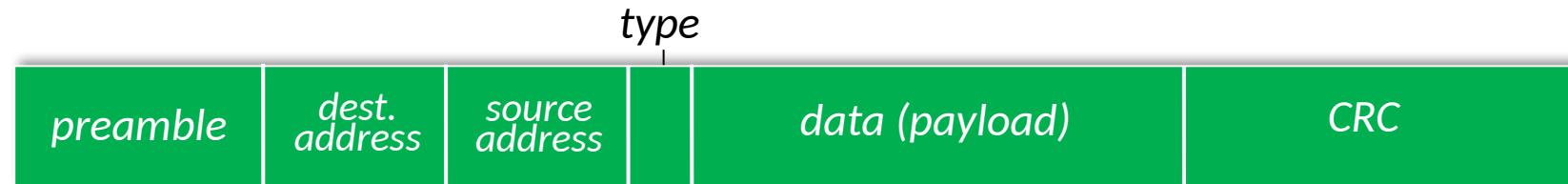


switched



# Ethernet frame structure

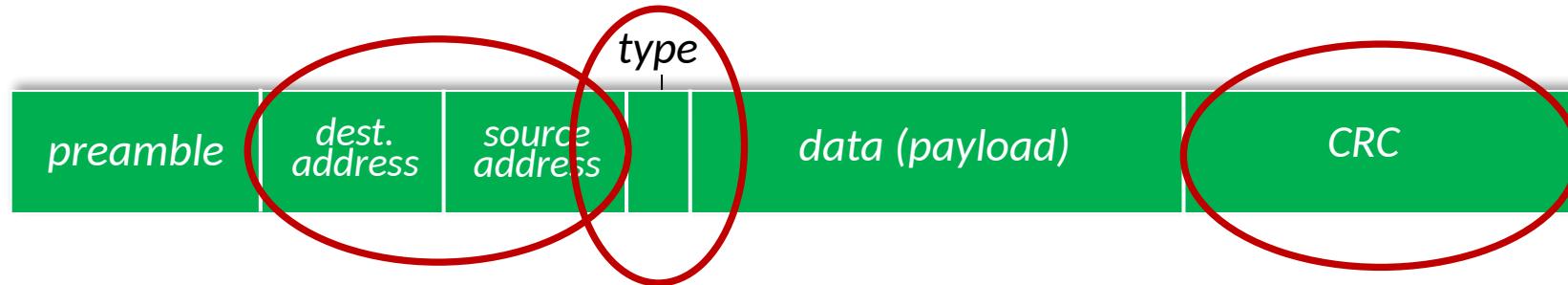
sending interface encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**



## *preamble:*

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

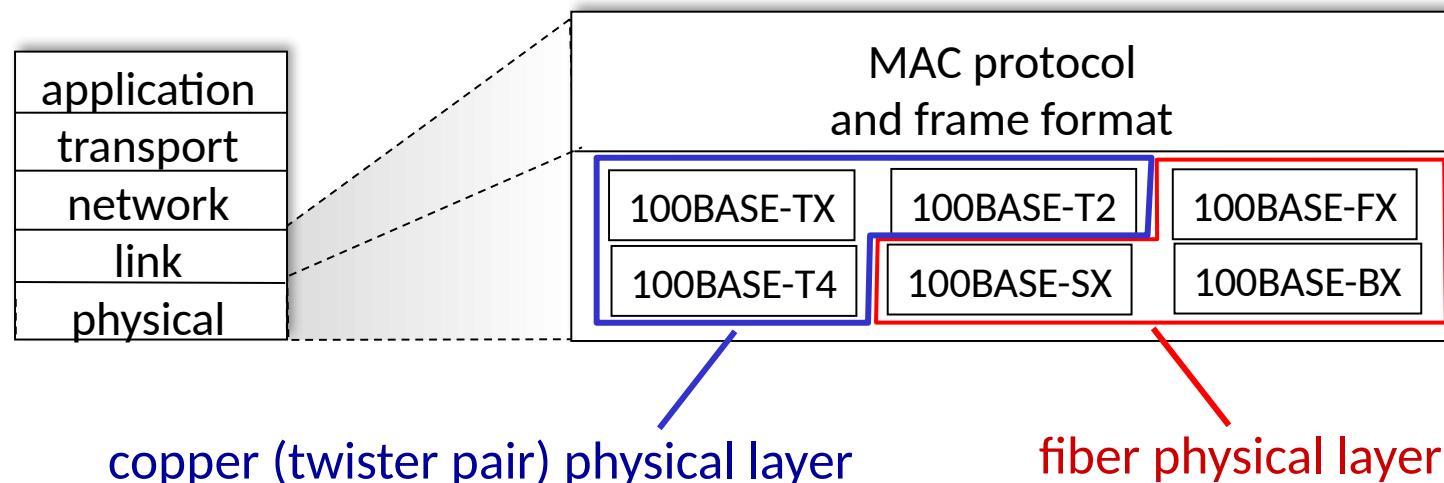
# Ethernet frame structure (more)



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

# 802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, ... 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps, 80 Gbps
    - different physical layer media: fiber, cable



# Link layer, LANs: roadmap

6.1 Introduction

6.2 Error detection and correction

6.3 Multiple access protocols

## 6.4 LANs

6.4.1 Addressing, ARP

6.4.2 Ethernet

### 6.4.3 Switches

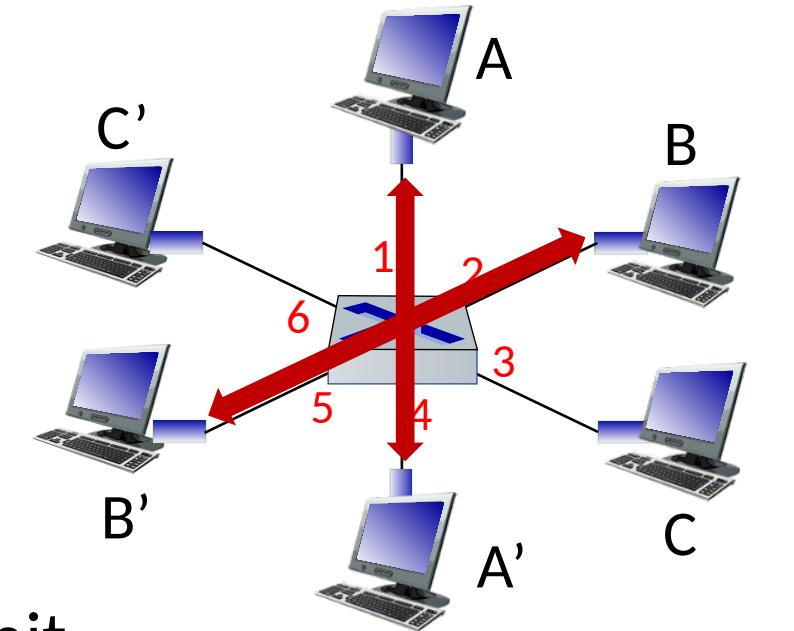
6.4.4 VLANs

6.6 Data center networking

6.7 A day in the life of a web request

# Switch: multiple simultaneous transmissions

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



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

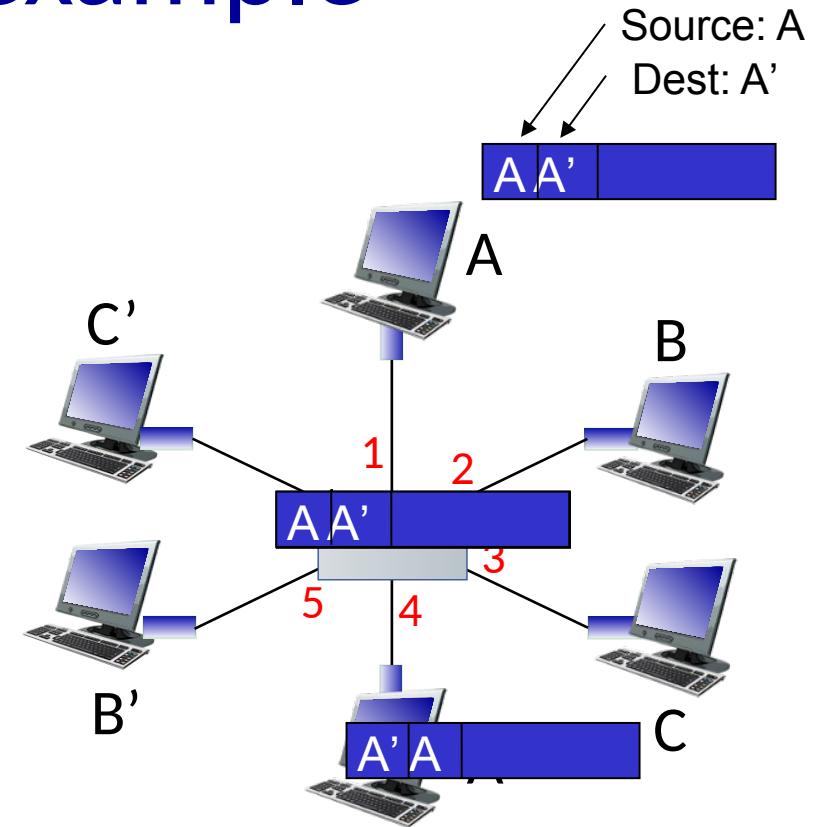
# Switch: frame filtering/forwarding

when frame received at switch:

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

# Self-learning, forwarding: example

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

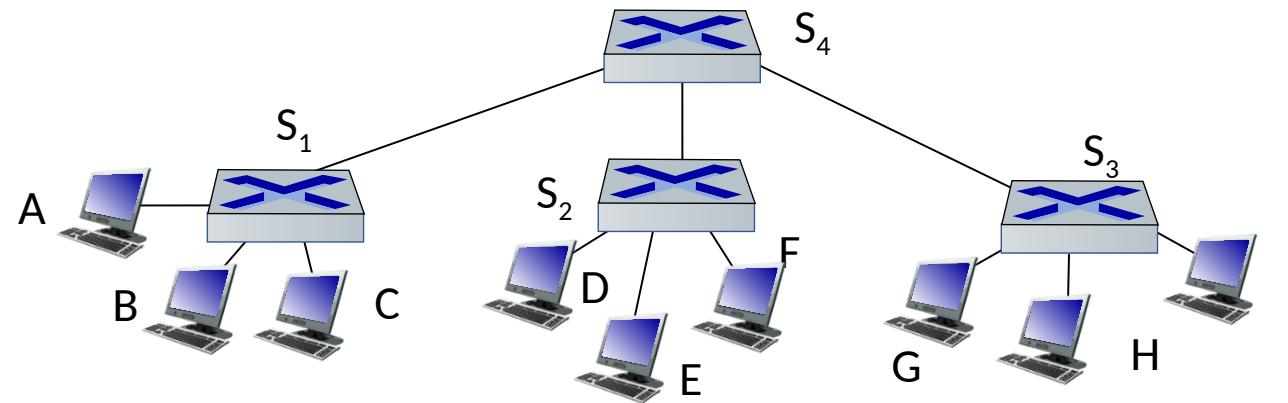


| MAC addr | interface | TTL |
|----------|-----------|-----|
| A        | 1         | 60  |
| A'       | 4         | 60  |

*switch table  
(initially empty)*

# Interconnecting switches

self-learning switches can be connected together:



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

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

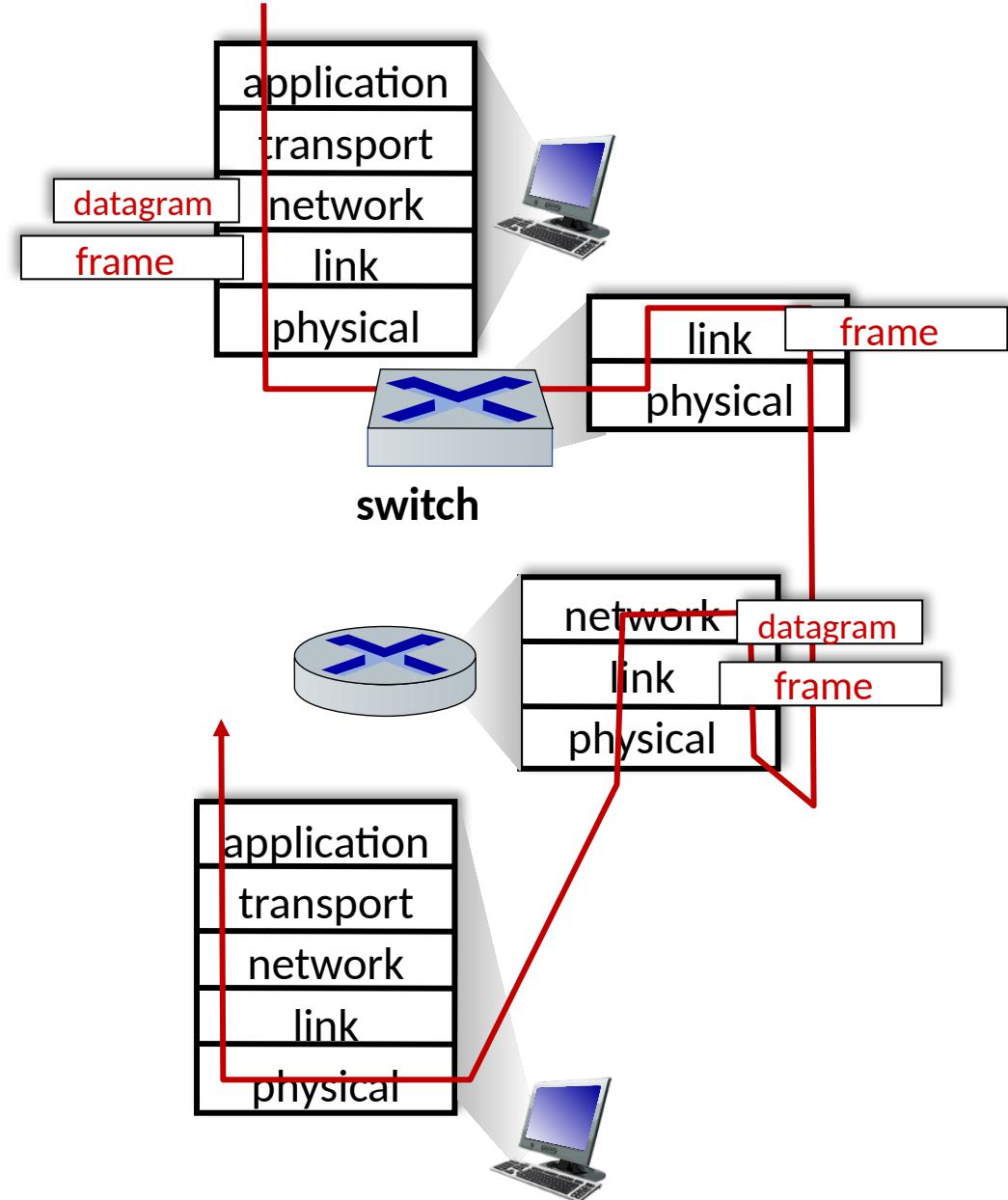
# Switches vs. routers

both are store-and-forward:

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

both have forwarding tables:

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



# Link layer, LANs: roadmap

6.1 Introduction

6.2 Error detection and correction

6.3 Multiple access protocols

## 6.4 LANs

6.4.1 Addressing, ARP

6.4.2 Ethernet

6.4.3 Switches

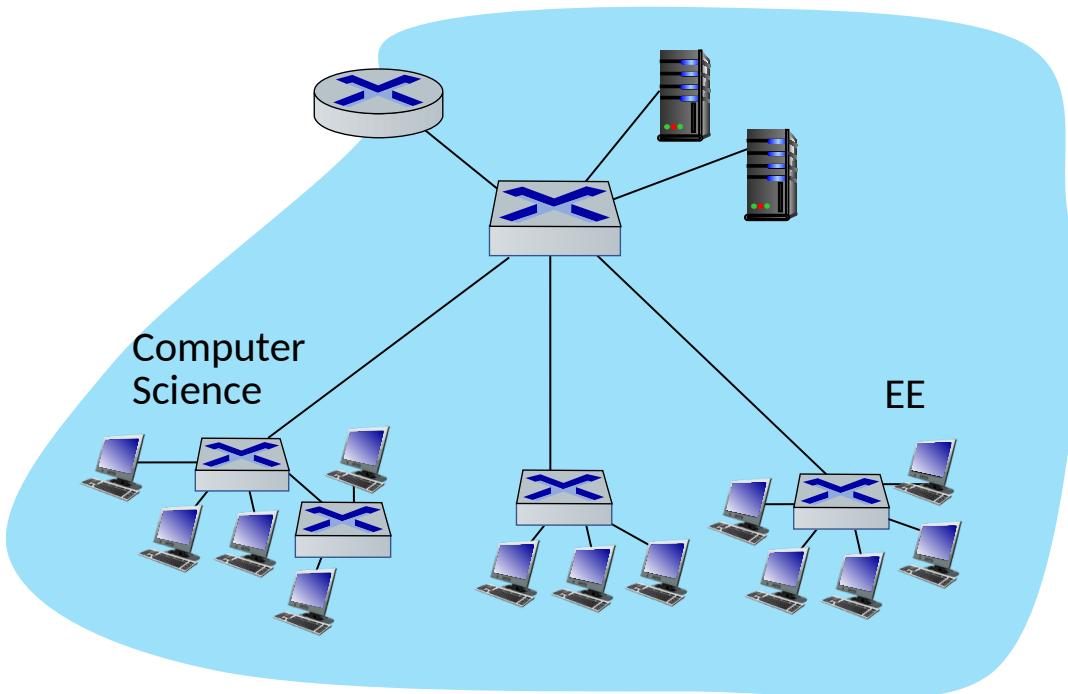
### 6.4.4 VLANs

6.6 Data center networking

6.7 A day in the life of a web request

# Virtual LANs (VLANs): motivation

Q: what happens as LAN sizes scale, users change point of attachment?

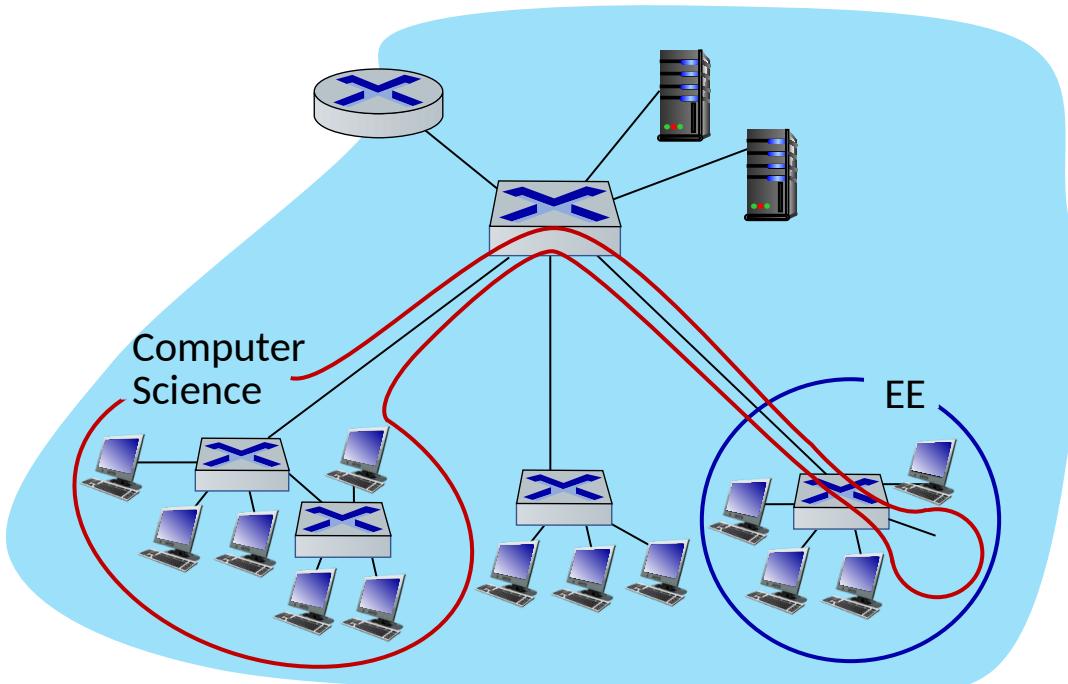


**single broadcast domain:**

- *scaling*: all layer-2 broadcast traffic (ARP, DHCP, unknown MAC) must cross entire LAN
- efficiency, security, privacy issues

# Virtual LANs (VLANs): motivation

Q: what happens as LAN sizes scale, users change point of attachment?



## single broadcast domain:

- *scaling*: all layer-2 broadcast traffic (ARP, DHCP, unknown MAC) must cross entire LAN
- efficiency, security, privacy, efficiency issues

## administrative issues:

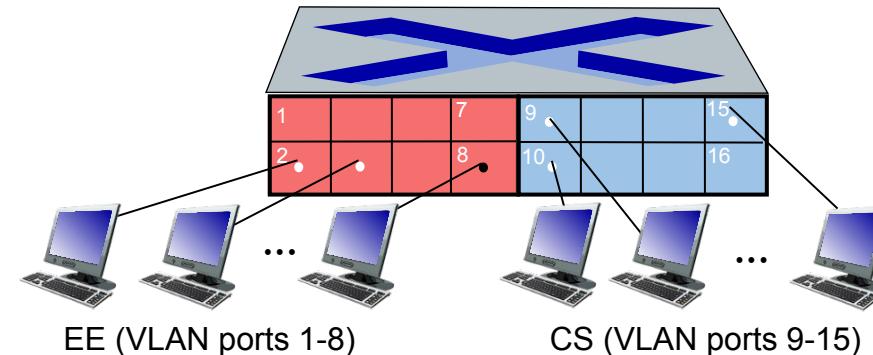
- CS user moves office to EE - *physically* attached to EE switch, but wants to remain *logically* attached to CS switch

# Port-based VLANs

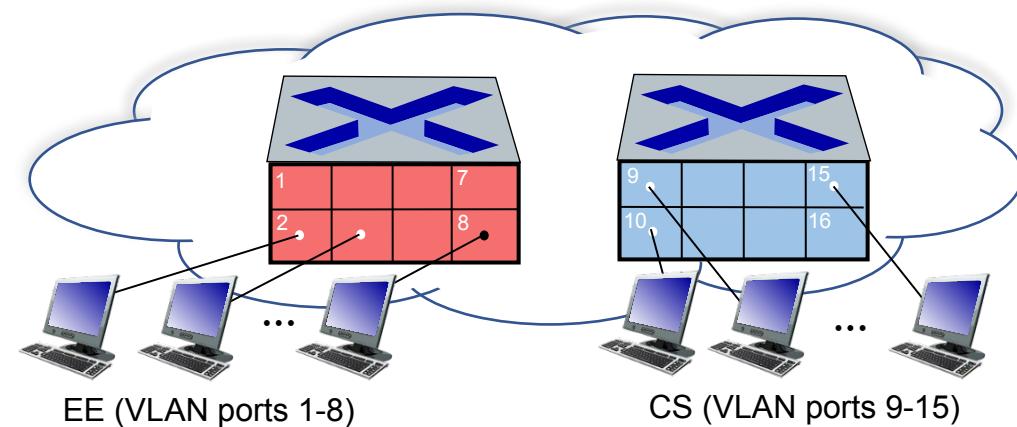
## Virtual Local Area Network (VLAN)

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

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

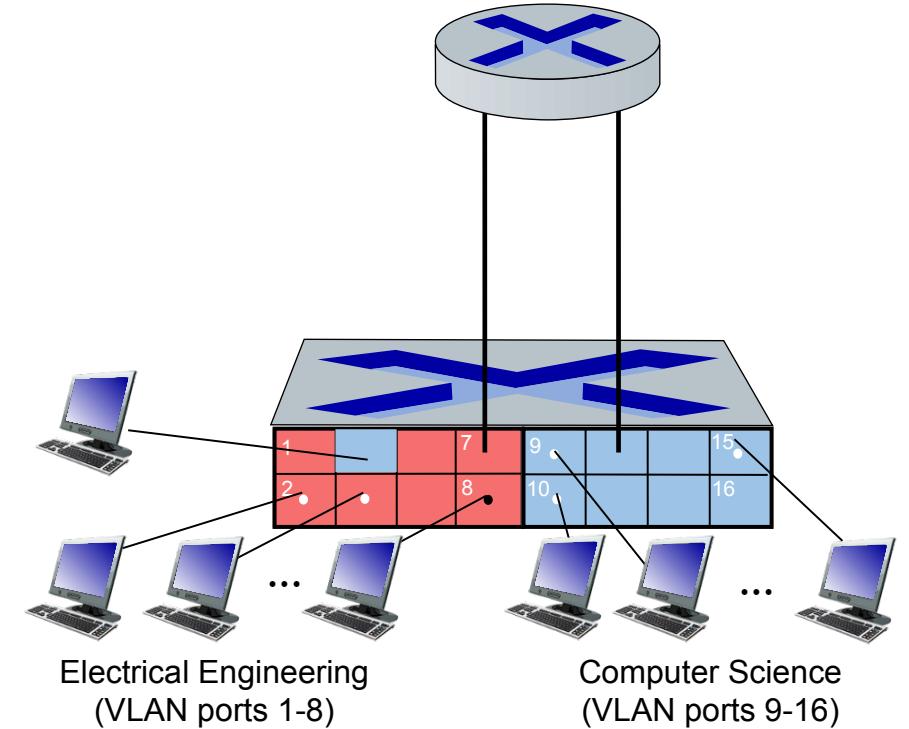


... operates as **multiple virtual switches**



# Port-based VLANs

- **traffic isolation:** frames to/from ports 1-8 can only reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- **dynamic membership:** ports can be dynamically assigned among VLANs
- **forwarding between VLANs:** done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers



# **Link layer, LANs: roadmap**

6.1 Introduction

6.2 Error detection and correction

6.3 Multiple access protocols

6.4 LANs

Addressing, ARP

Ethernet

Switches

VLANs

**6.6 Data center networking**

6.7 A day in the life of a web request

# Datacenter networks

10's to 100's of thousands of hosts, often closely coupled, in close proximity:

- e-business (e.g. Amazon)
- content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
- search engines, data mining (e.g., Google)

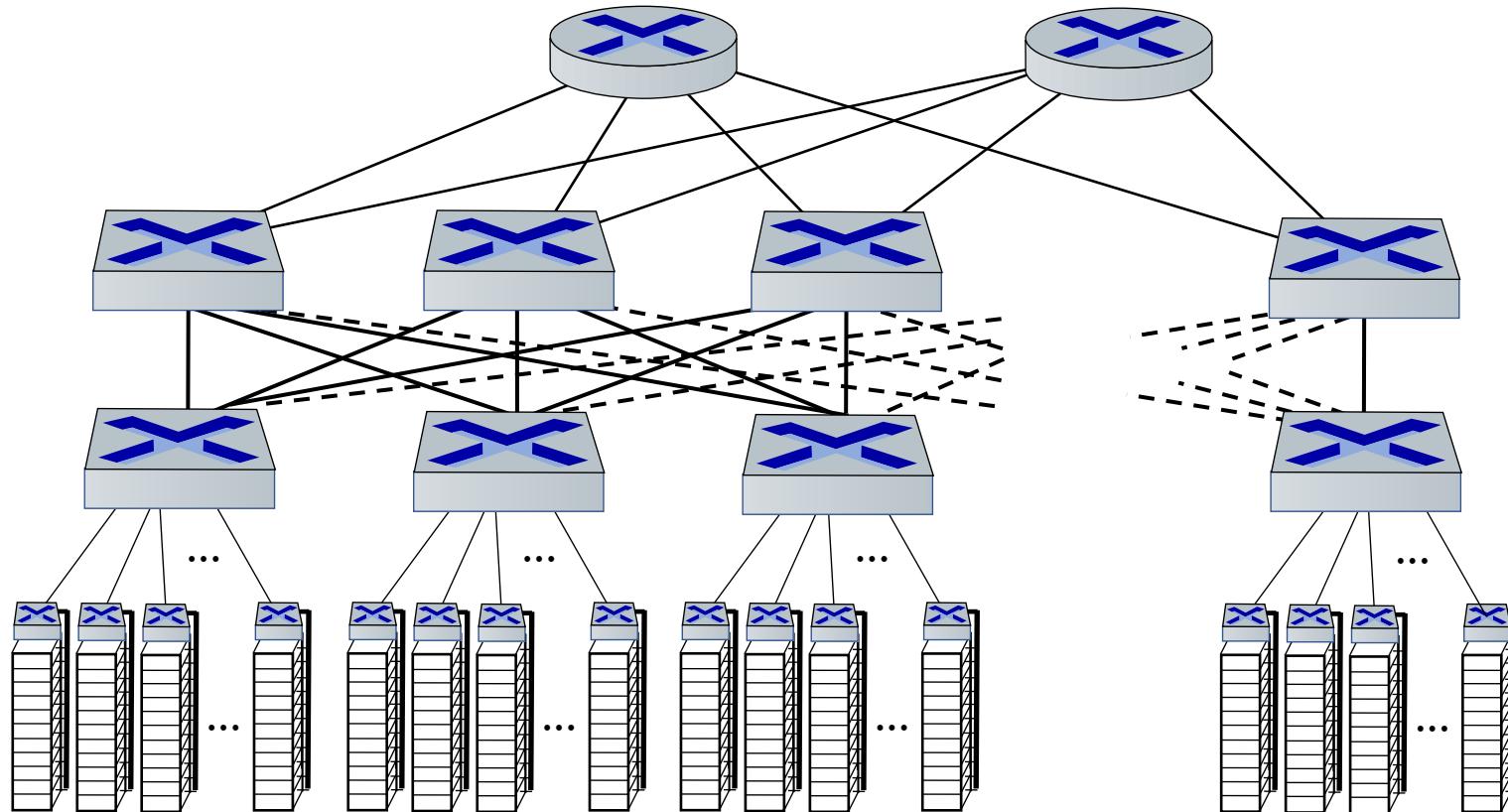
challenges:

- multiple applications, each serving massive numbers of clients
- reliability
- managing/balancing load, avoiding processing, networking, data bottlenecks



Inside a 40-ft Microsoft container, Chicago data center

# Datacenter networks: network elements



## Border routers

- connections outside datacenter

## Tier-1 switches

- connecting to ~16 Tier-2s below

## Tier-2 switches

- connecting to ~16 TORs below

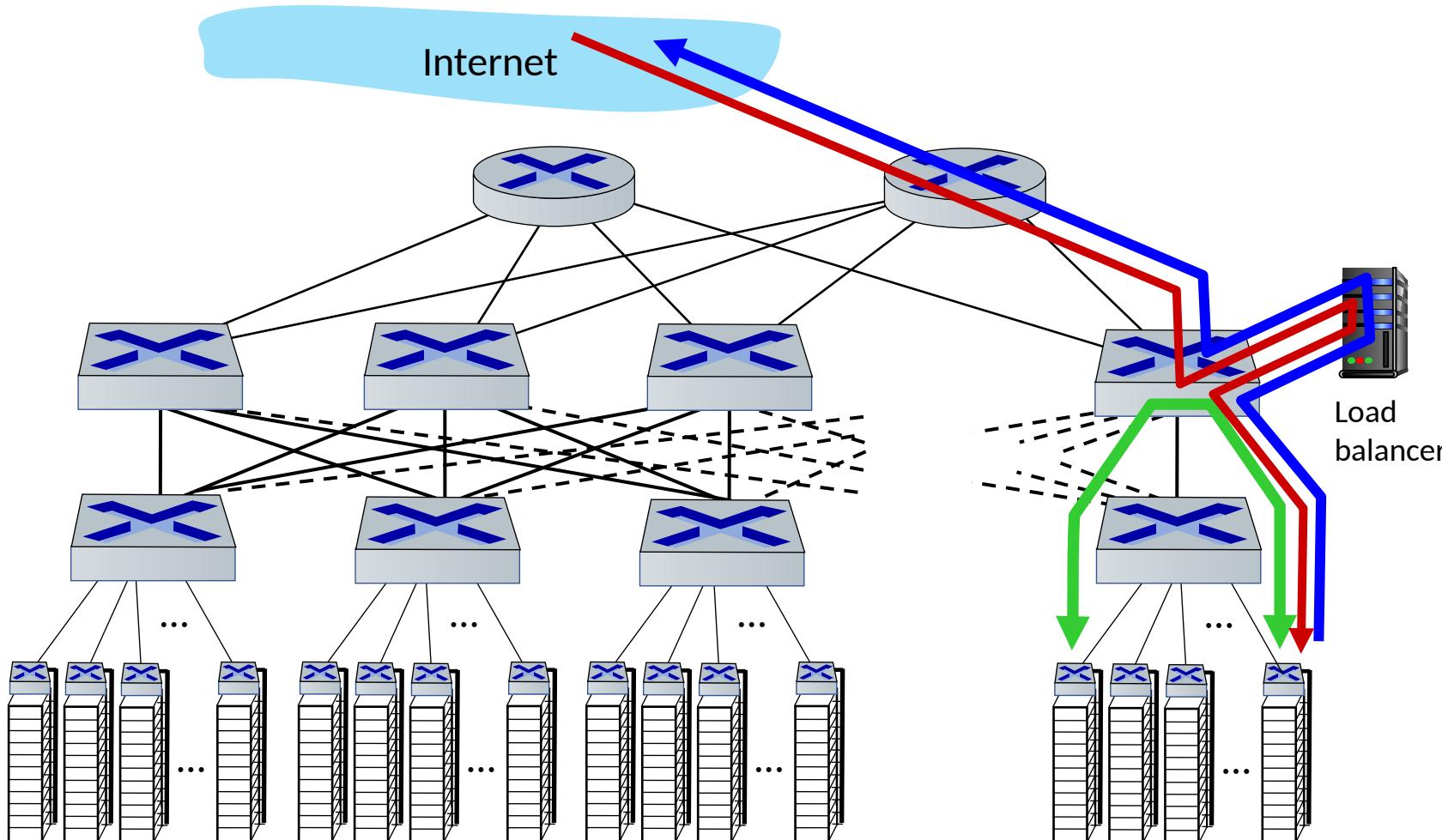
## Top of Rack (TOR) switch

- one per rack
- 40-100Gbps Ethernet to blades

## Server racks

- 20- 40 server blades: hosts

# Datacenter networks: application-layer routing



load balancer:  
application-layer  
routing

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

# **Link layer, LANs: roadmap**

6.1 Introduction

6.2 Error detection, correction

6.3 Multiple access protocols

6.4 LANs

Addressing, ARP

Ethernet

Switches

VLANs

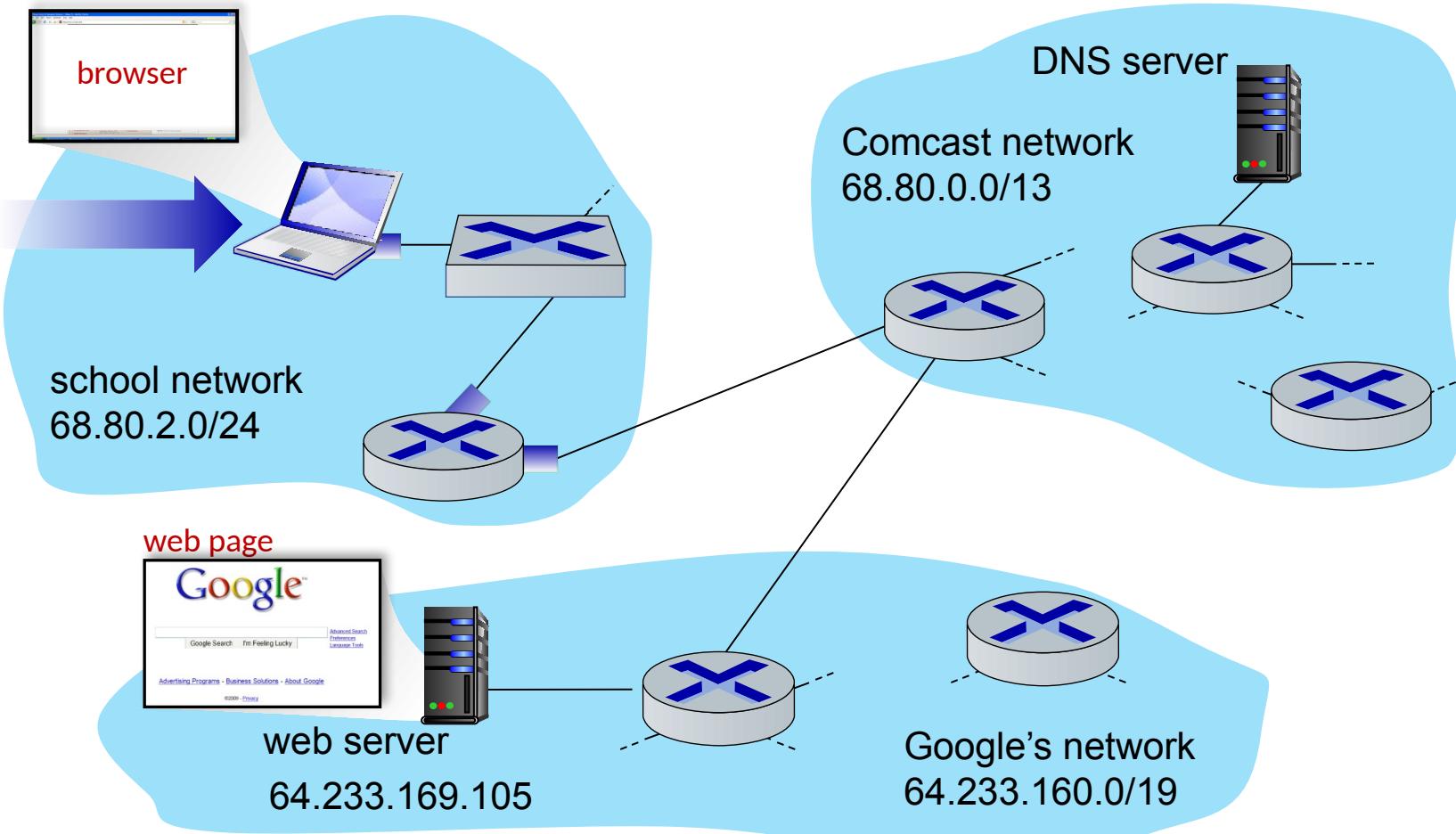
6.6 Data center networking

**6.7 A day in the life of a web request**

# Synthesis: a day in the life of a web request

- our journey down the protocol stack is now complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal*: identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario*: student attaches laptop to campus network, requests/receives [www.google.com](http://www.google.com)

# A day in the life: scenario

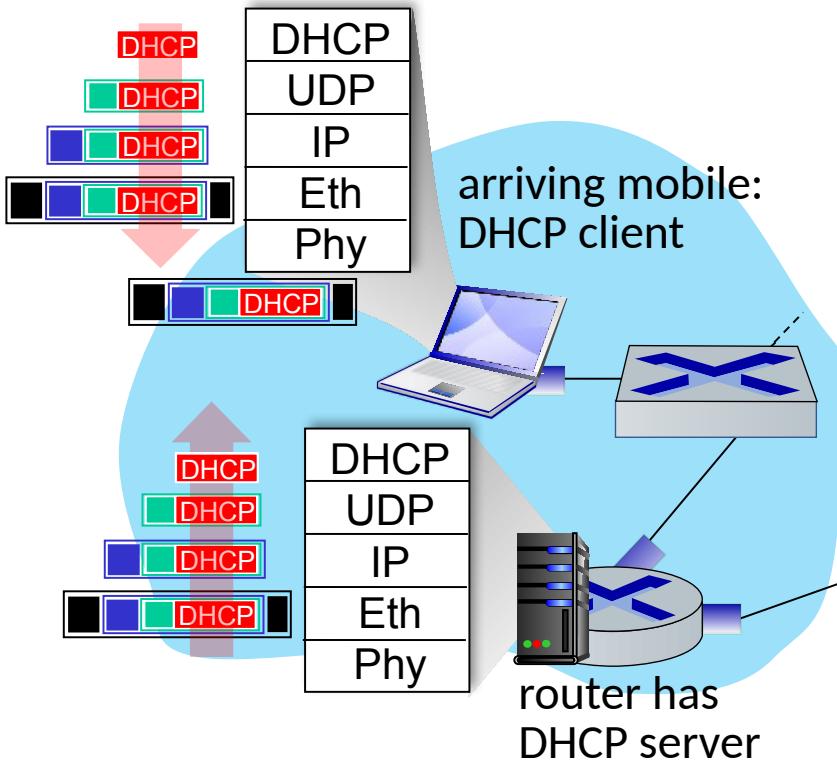


scenario:

- arriving mobile client attaches to network ...
- requests web page:  
[www.google.com](http://www.google.com)

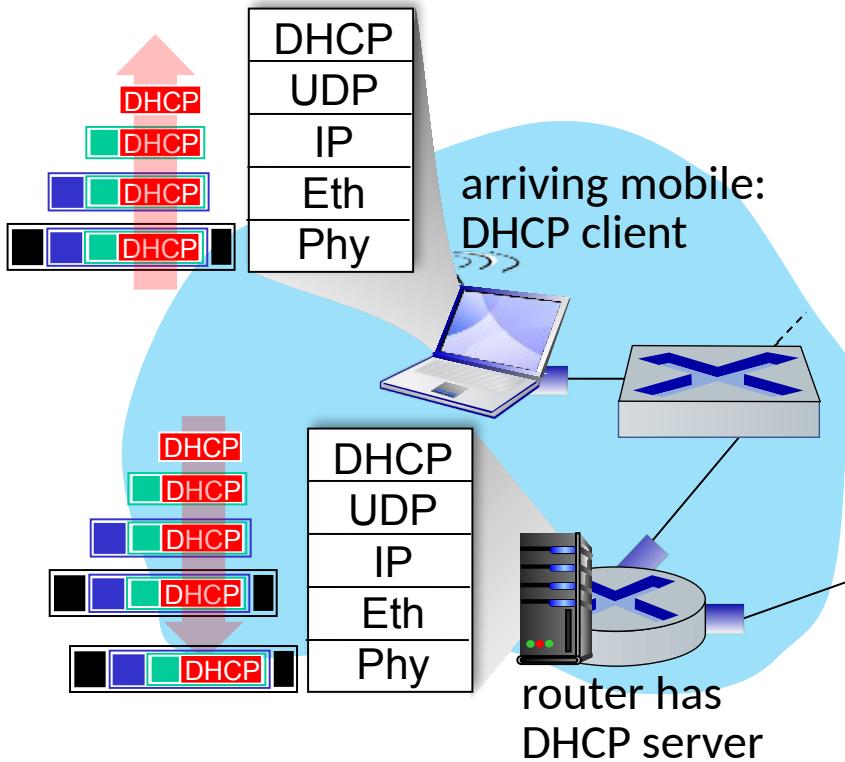
*Sounds simple!* !

# A day in the life: connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use **DHCP**
- DHCP request encapsulated in **UDP**, encapsulated in **IP**, encapsulated in **802.3 Ethernet**
- Ethernet frame **broadcast** (dest: 0xFFFFFFFFFFFF) on LAN, received at router running **DHCP server**
- Ethernet **demuxed** to IP demuxed, UDP demuxed to DHCP

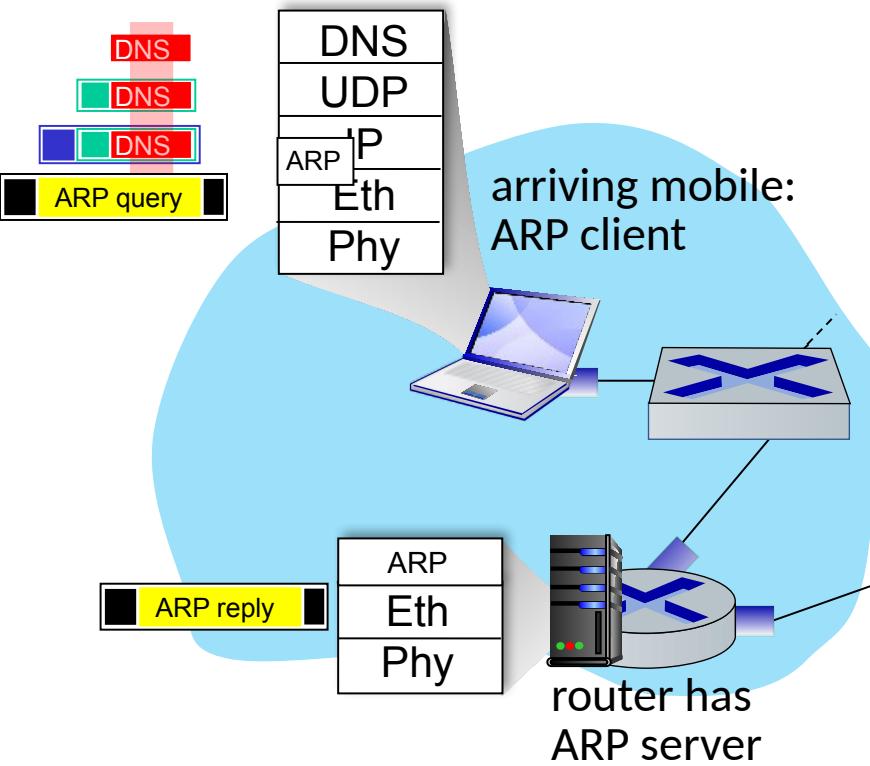
# A day in the life: connecting to the Internet



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

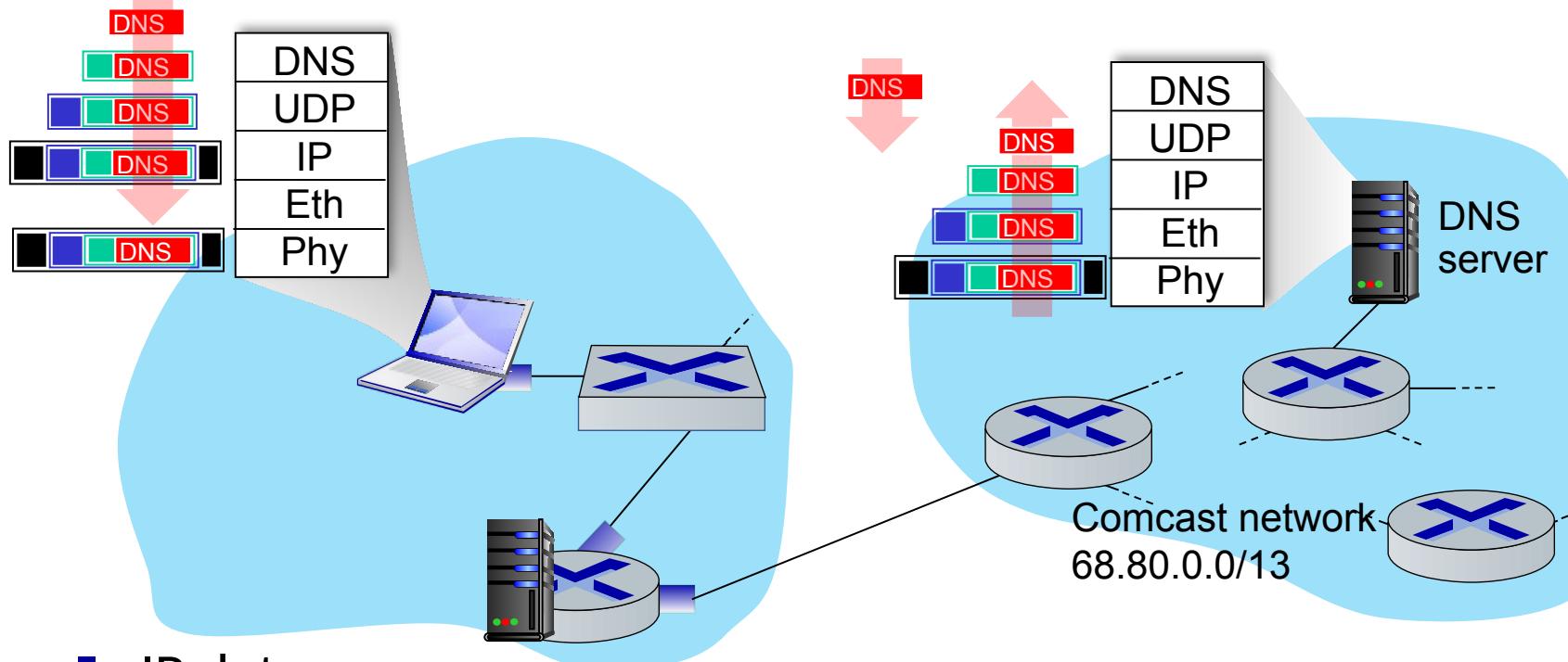
*Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router*

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



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

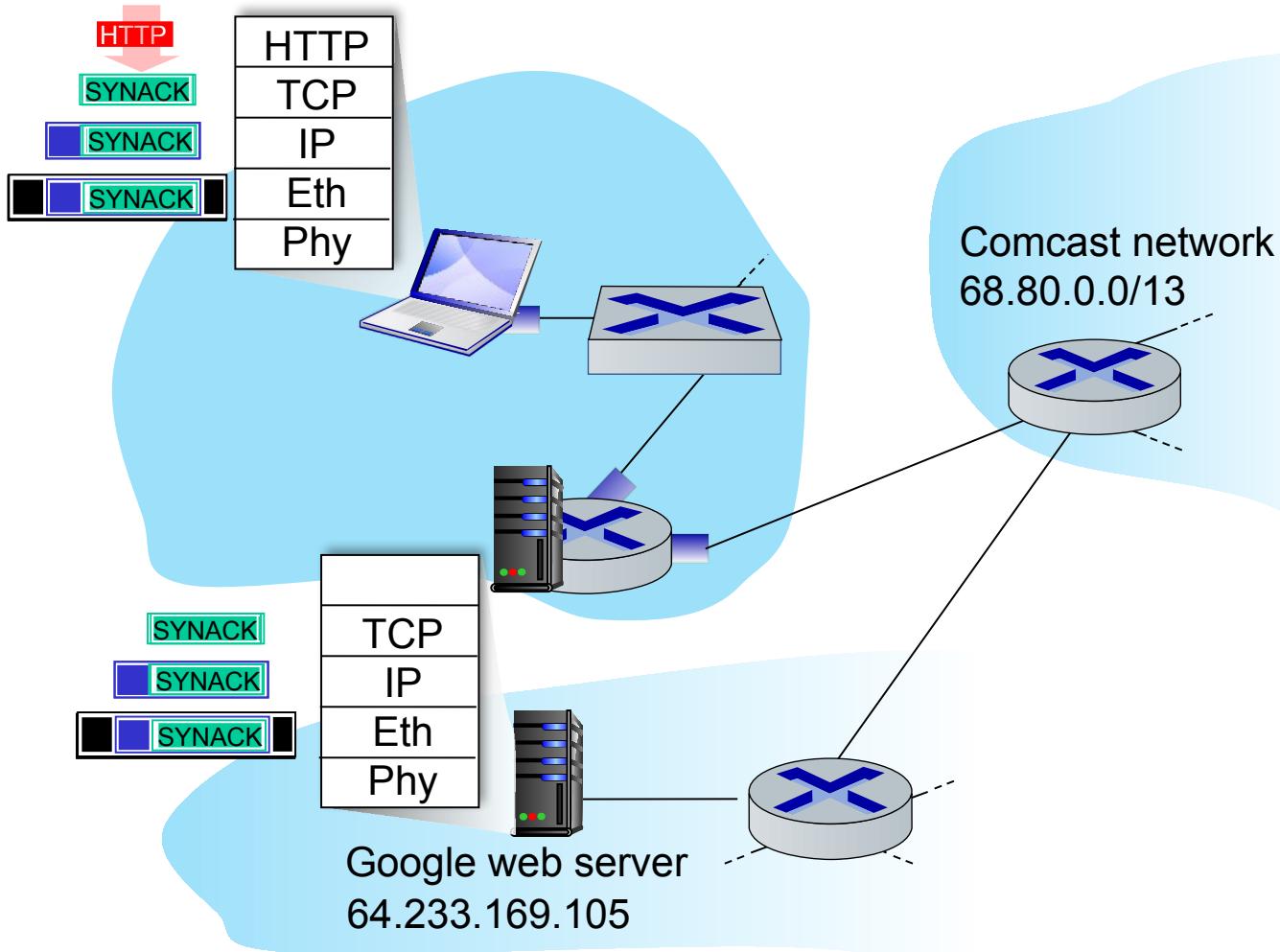
# A day in the life... using DNS



- IP datagram containing DNS query forwarded via LAN switch from client to 1<sup>st</sup> hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by **RIP**, **OSPF** and/or **BGP** routing protocols) to DNS server

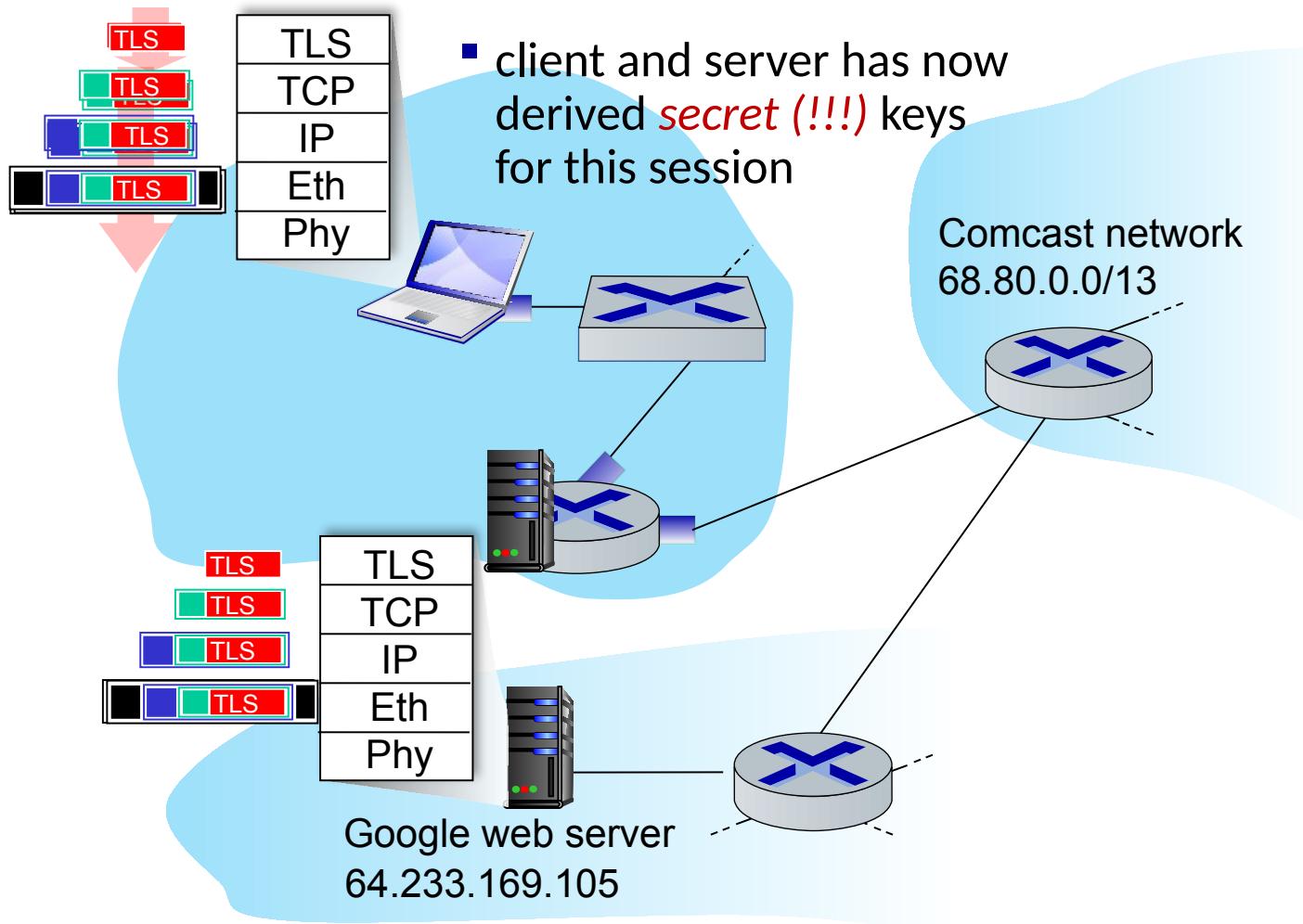
- demuxed to DNS
- DNS replies to client with IP address of [www.google.com](http://www.google.com)

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



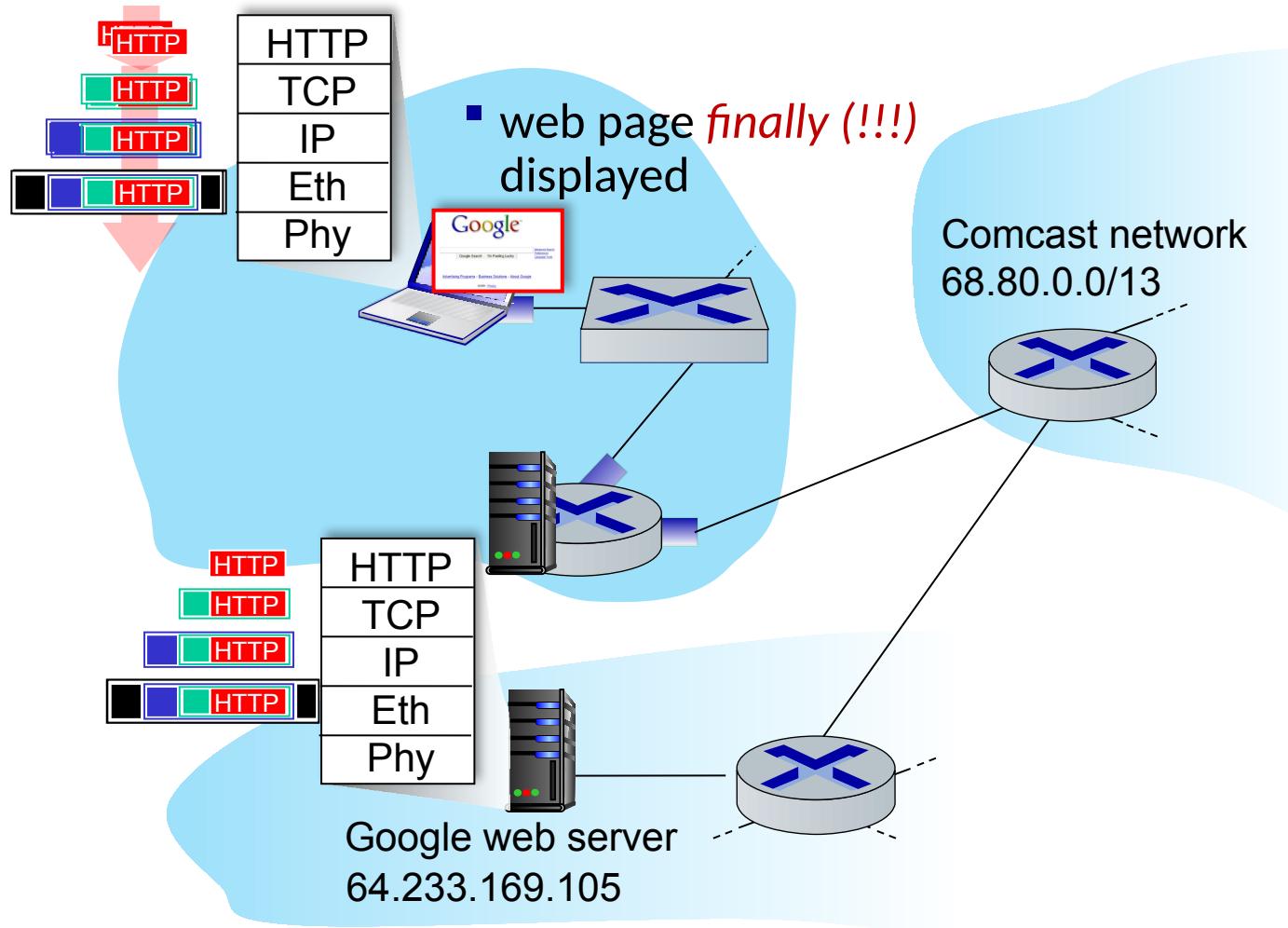
- to send HTTP request, client first opens **TCP socket** to web server
- **TCP SYN segment** (step 1 in TCP 3-way handshake) inter-domain routed to web server
- web server responds with **TCP SYNACK** (step 2 in TCP 3-way handshake)
- **TCP connection established!**

# A day in the life... TLS securing TCP connections



- TLS Client Hello, other records (cipher suite, nonce, DH params, public point) sent into TCP socket
- IP datagram containing TLS records routed to [www.google.com](http://www.google.com)
- web server responds with TLS Server Hello, other records (certificate, cipher suite, nonce, DH params, public key, finished)
- IP datagram containing TLS records) routed back to client

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



- **HTTP request sent into SSL socket**
- IP datagram containing TLS Finished record and encrypted HTTP request routed to [www.google.com](http://www.google.com)
- web server responds with **HTTP reply** (containing web page)
- IP datagram containing encrypted HTTP reply in TCP segment routed back to client

# Chapter 6: Summary

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

# Chapter 7

# Wireless and Mobile Networks

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

GLOBAL  
EDITION 

Computer Networking

*A Top-Down Approach*

EIGHTH EDITION

James F. Kurose • Keith W. Ross



**Computer Networking: A  
Top-Down Approach**

8<sup>th</sup> edition

Jim Kurose, Keith Ross  
Pearson, 2020

# Chapter 7 outline

## 7.1 Introduction

7.2 Wireless links and network characteristics

7.3 WiFi: 802.11 wireless LANs

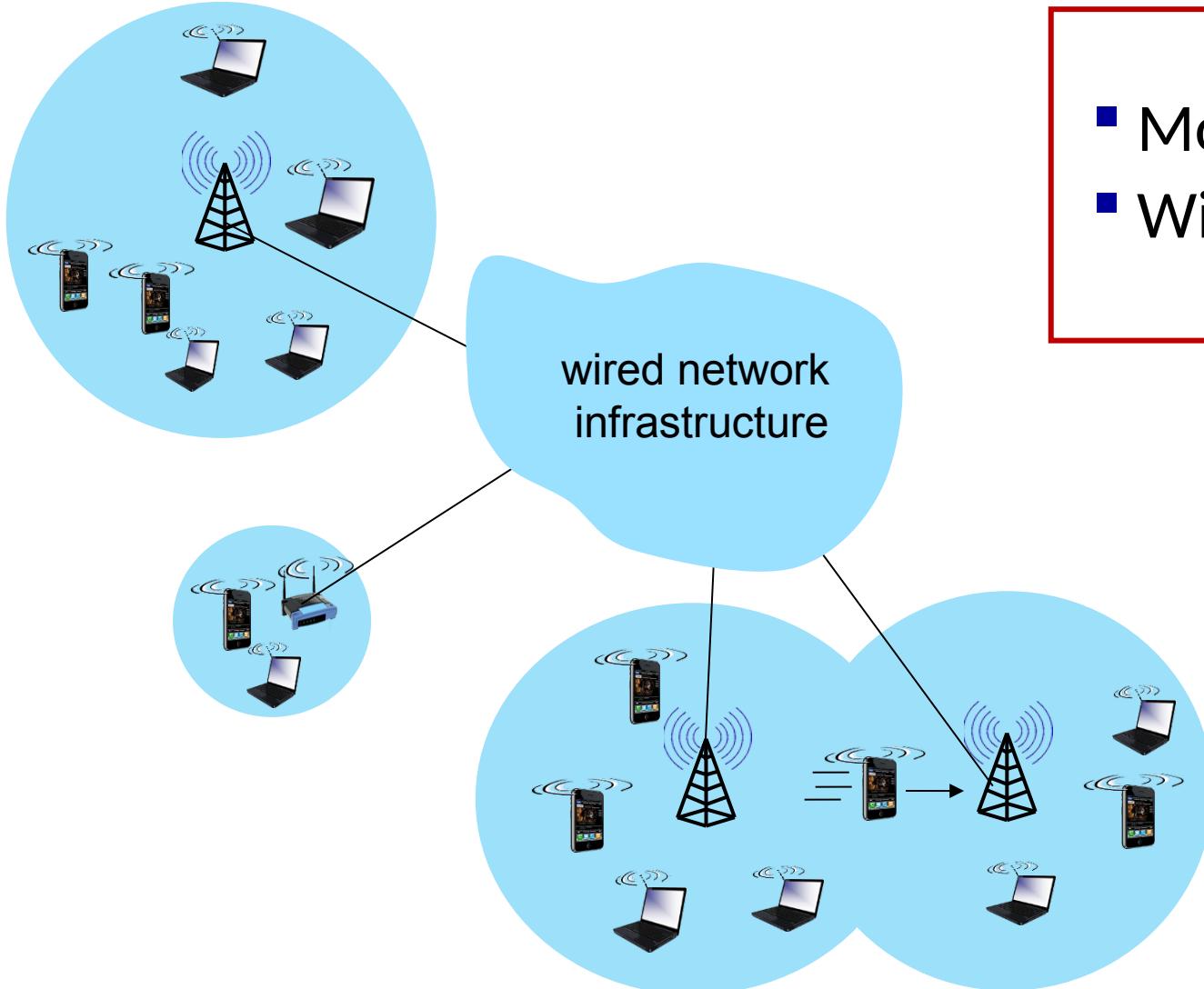
~~7.4 Cellular networks: 4G and 5G~~

~~7.5 Mobility management: principles~~

~~7.6 Mobility management: practice~~

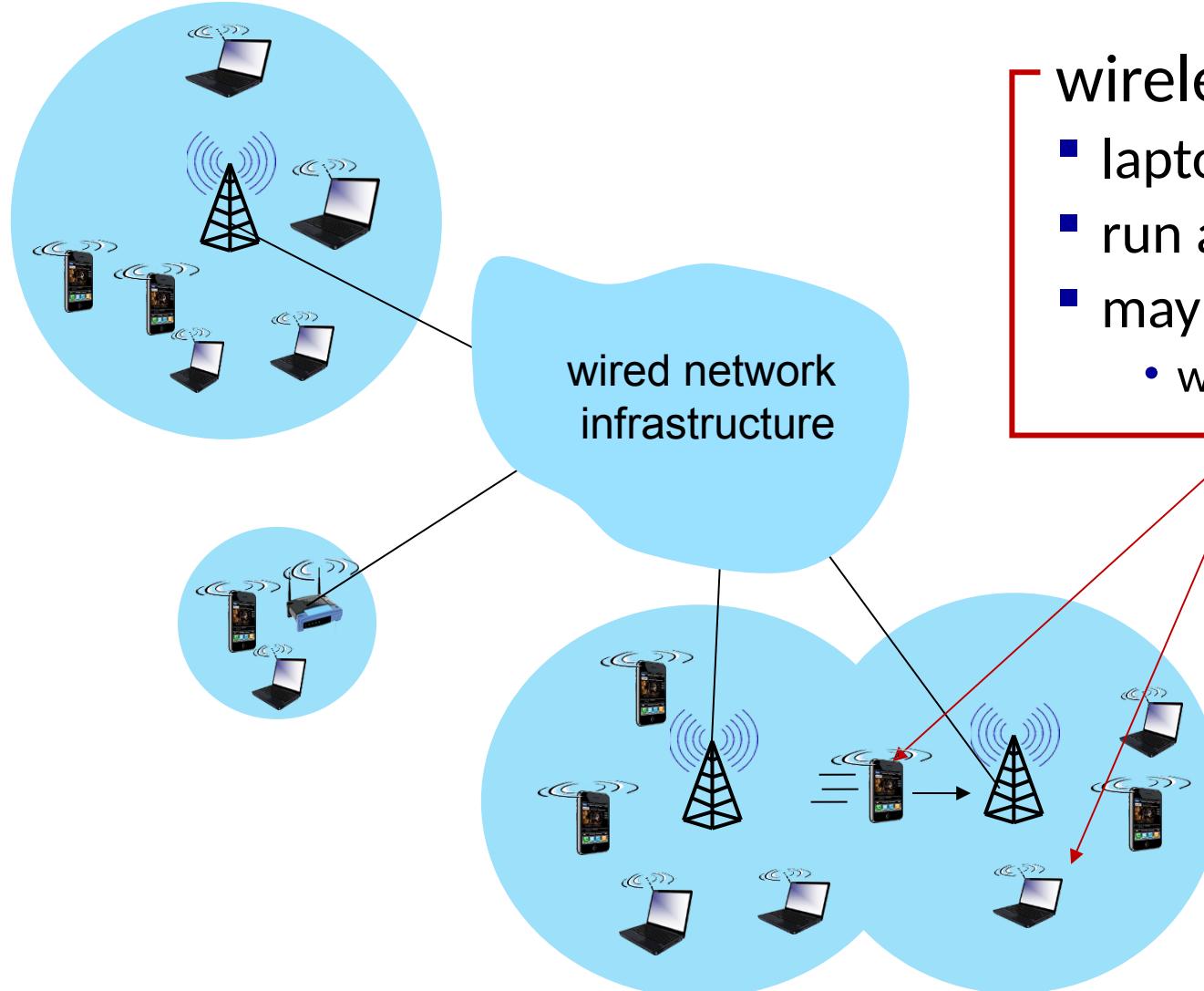
~~7.7 Mobility: impact on higher layer protocols~~

# Elements of a wireless network



- Mobile networks, cellular networks
- Wireless LANs, Wi-Fi

# Elements of a wireless network

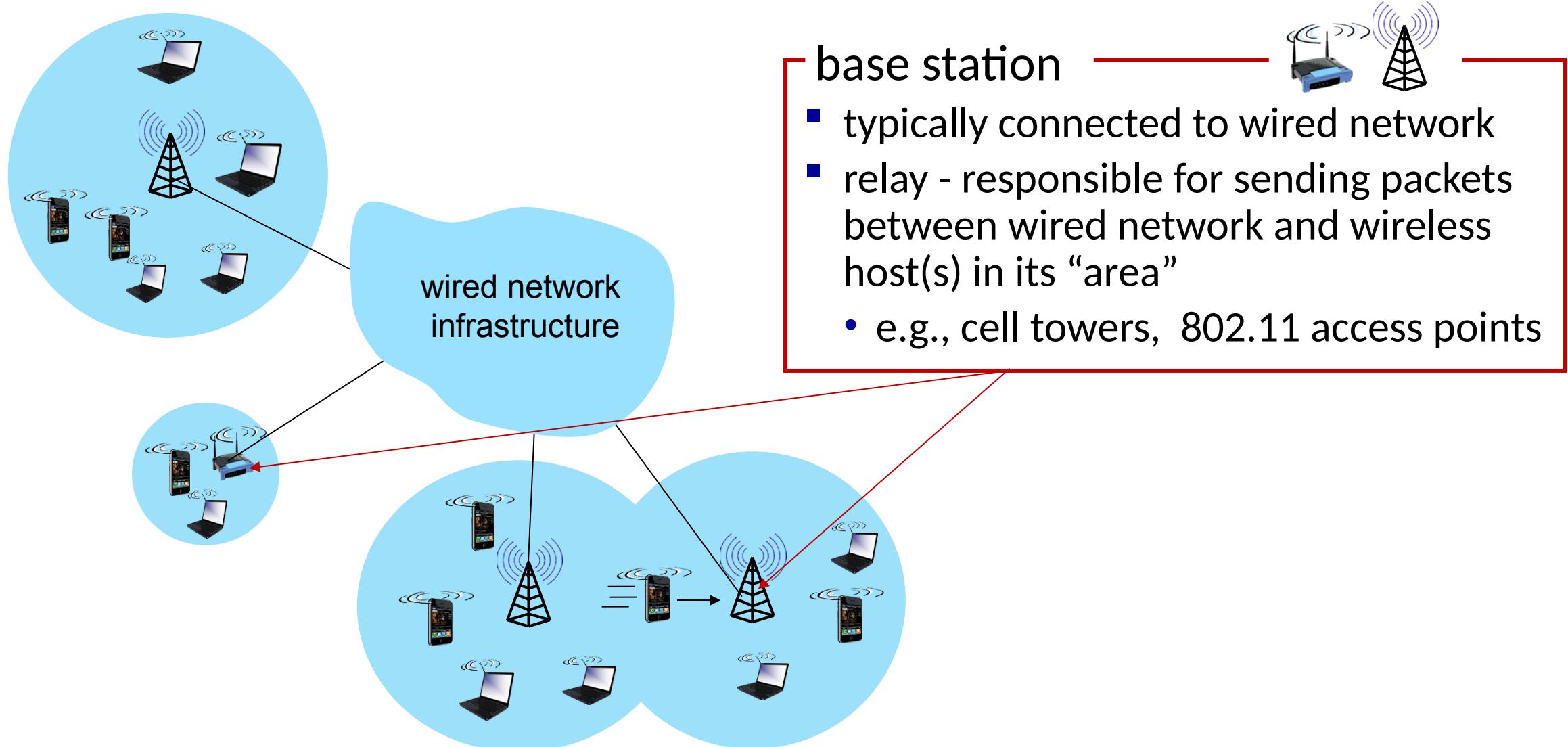


## wireless hosts

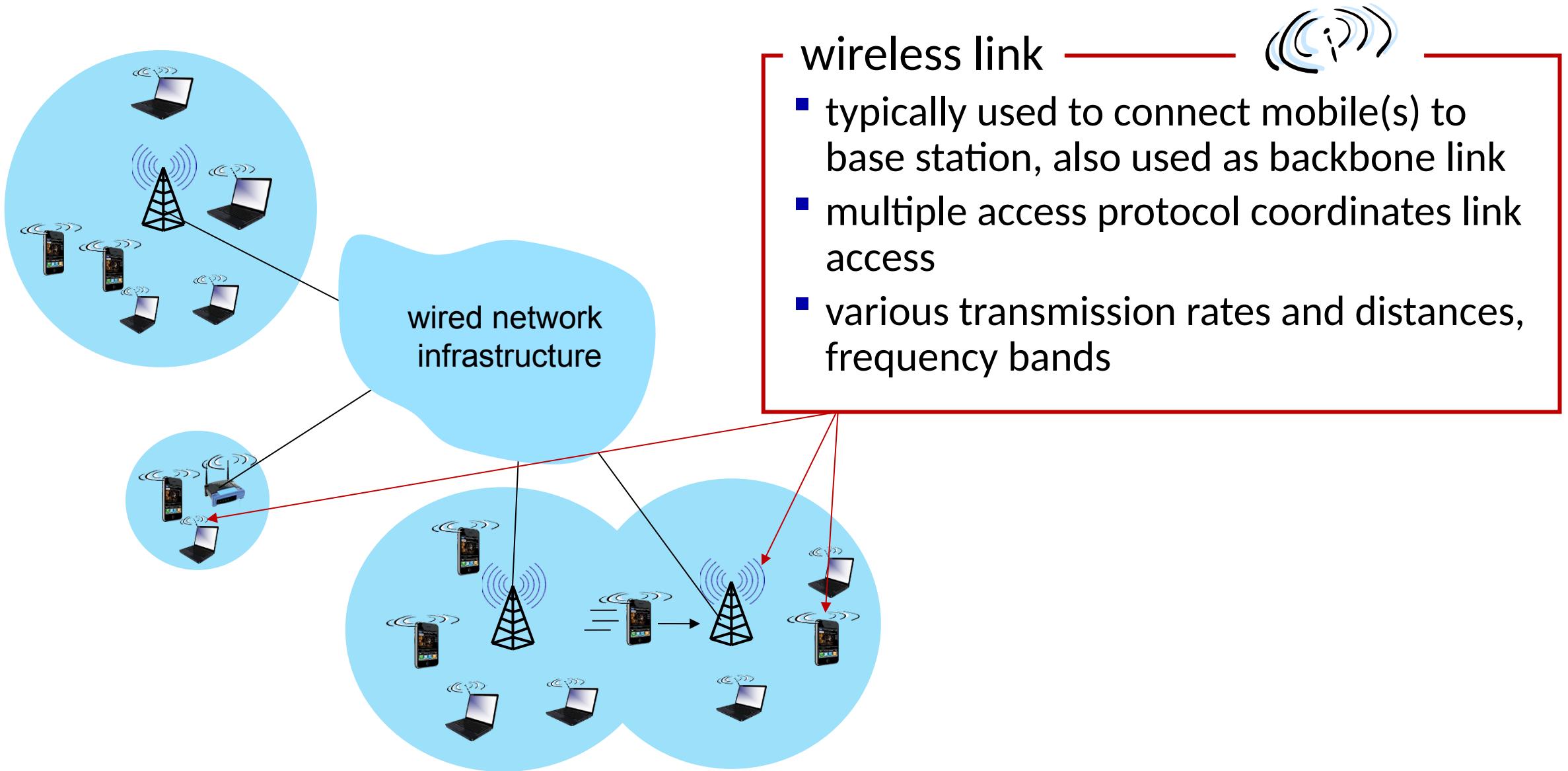
- laptop, smartphone, IoT
- run applications
- may be stationary (non-mobile) or mobile
  - wireless does not always mean mobility!



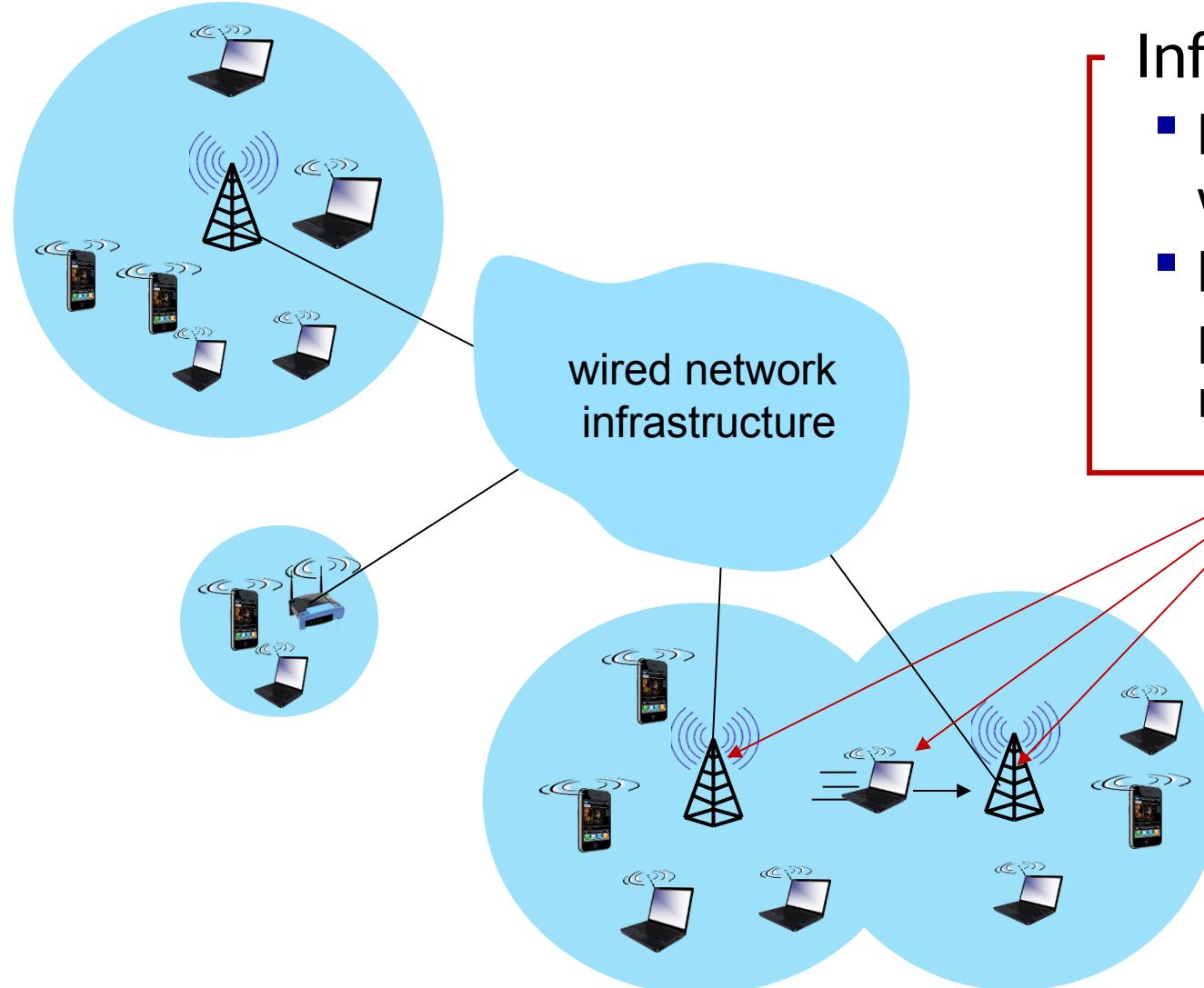
# Elements of a wireless network



# Elements of a wireless network



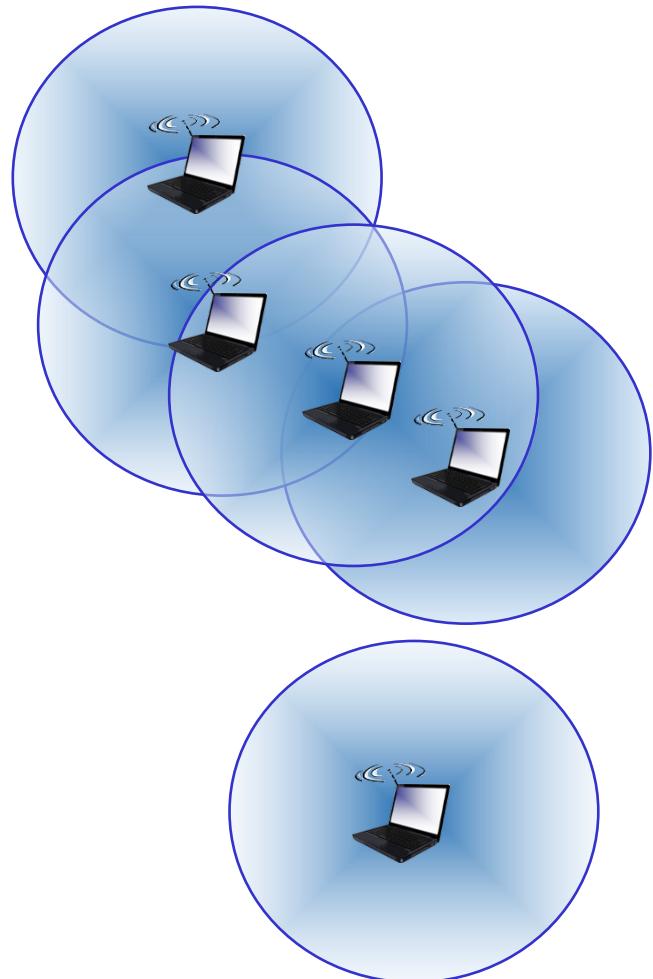
# Infrastructure networks



## Infrastructure mode

- base station connects mobiles into wired network
- handoff: mobile changes base station providing connection into wired network

# Ad hoc networks



ad hoc mode

- no base stations
- nodes can only transmit to other nodes within link coverage
- nodes organize themselves into a network: route among themselves

# Chapter 7 outline

7.1 Introduction

## 7.2 Wireless links and network characteristics

7.3 WiFi: 802.11 wireless LANs

~~7.4 Cellular networks: 4G and 5G~~

~~7.5 Mobility management: principles~~

~~7.6 Mobility management: practice~~

~~7.7 Mobility: impact on higher layer protocols~~

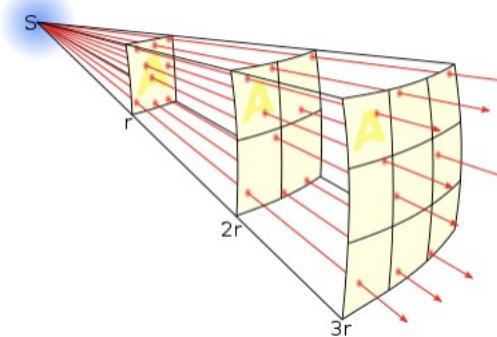
# Wireless link characteristics: fading (attenuation)

Wireless radio signal attenuates (loses power) as it propagates (free space “path loss”)

$$\text{Free space path loss} \sim (fd)^2$$

$f$ : frequency

$d$ : distance



higher frequency or  
longer distance



larger free space  
path loss

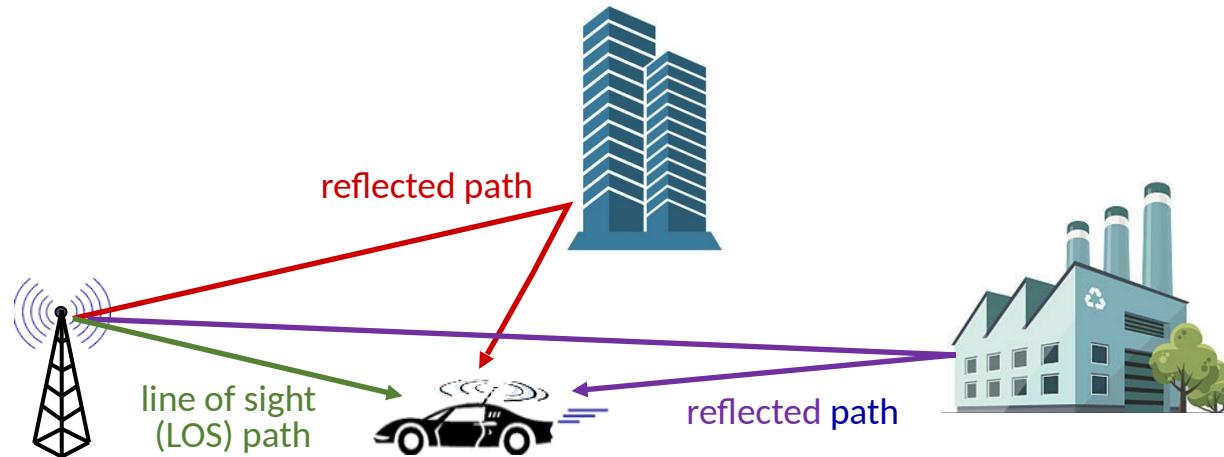
# Wireless link characteristics: interference

- **interference from other sources:** wireless network frequencies (e.g., 2.4 GHz) shared by many devices (e.g., WiFi, cellular, motors): interference



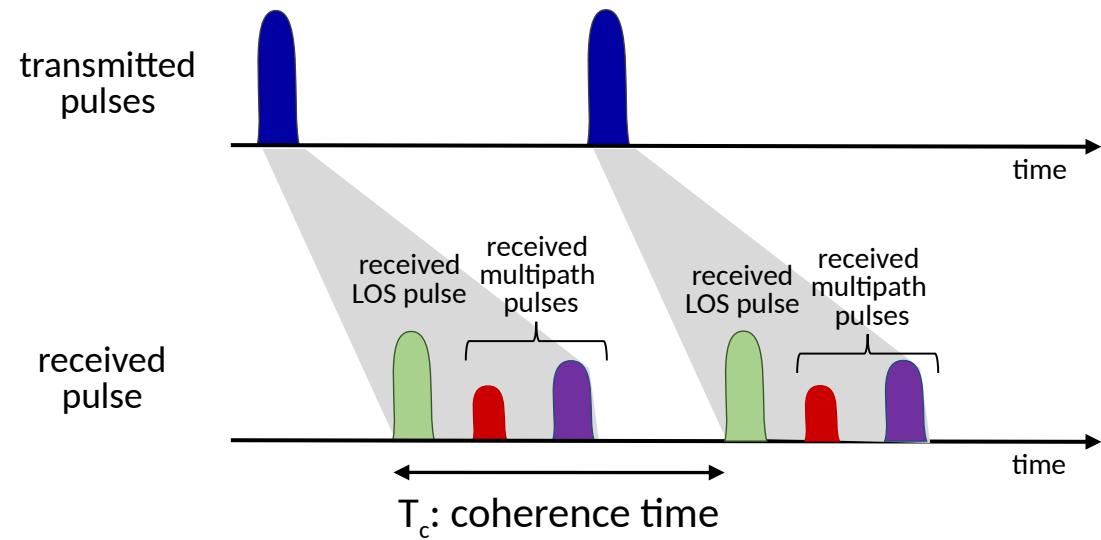
# Wireless link characteristics: multipath

**multipath propagation:** radio signal reflects off objects ground, built environment, arriving at destination at slightly different times



# Wireless link characteristics: multipath

**multipath propagation:** radio signal reflects off objects ground, built environment, arriving at destination at slightly different times

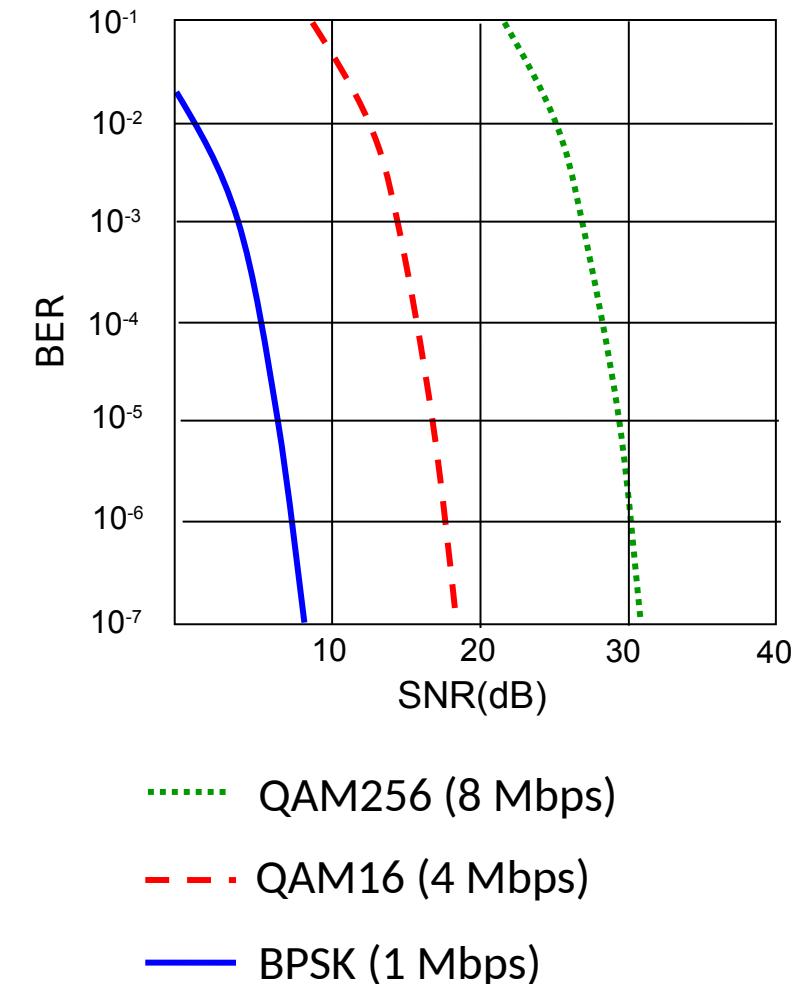


## Coherence time:

- amount of time bit is present in channel to be received
- influences maximum possible transmission rate, since coherence times can not overlap
- inversely proportional to
  - frequency
  - receiver velocity

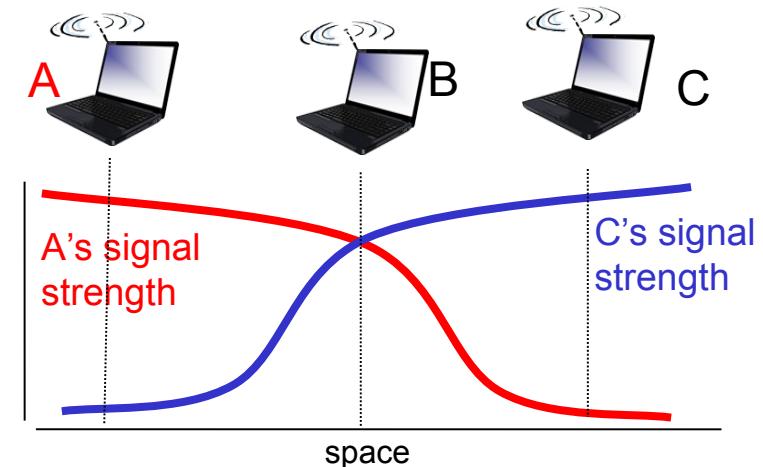
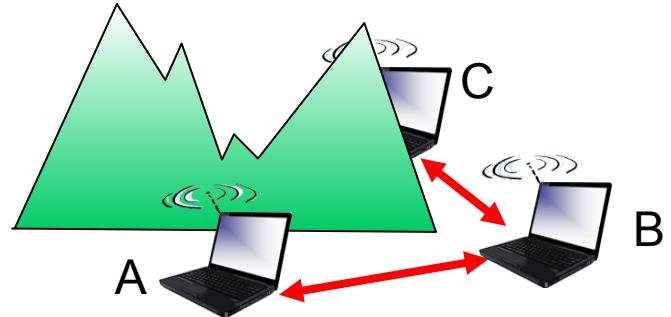
# Wireless link characteristics: noise

- SNR: signal-to-noise ratio
  - larger SNR – easier to extract signal from noise (a “good thing”)
- SNR versus BER tradeoffs
  - *given physical layer*: increase power -> increase SNR->decrease BER
  - *given SNR*: choose physical layer that meets BER requirement, giving highest throughput
    - SNR may change with mobility: dynamically adapt physical layer (modulation technique, rate)



# Wireless link characteristics (3)

Multiple wireless senders, receivers create additional problems (beyond multiple access):



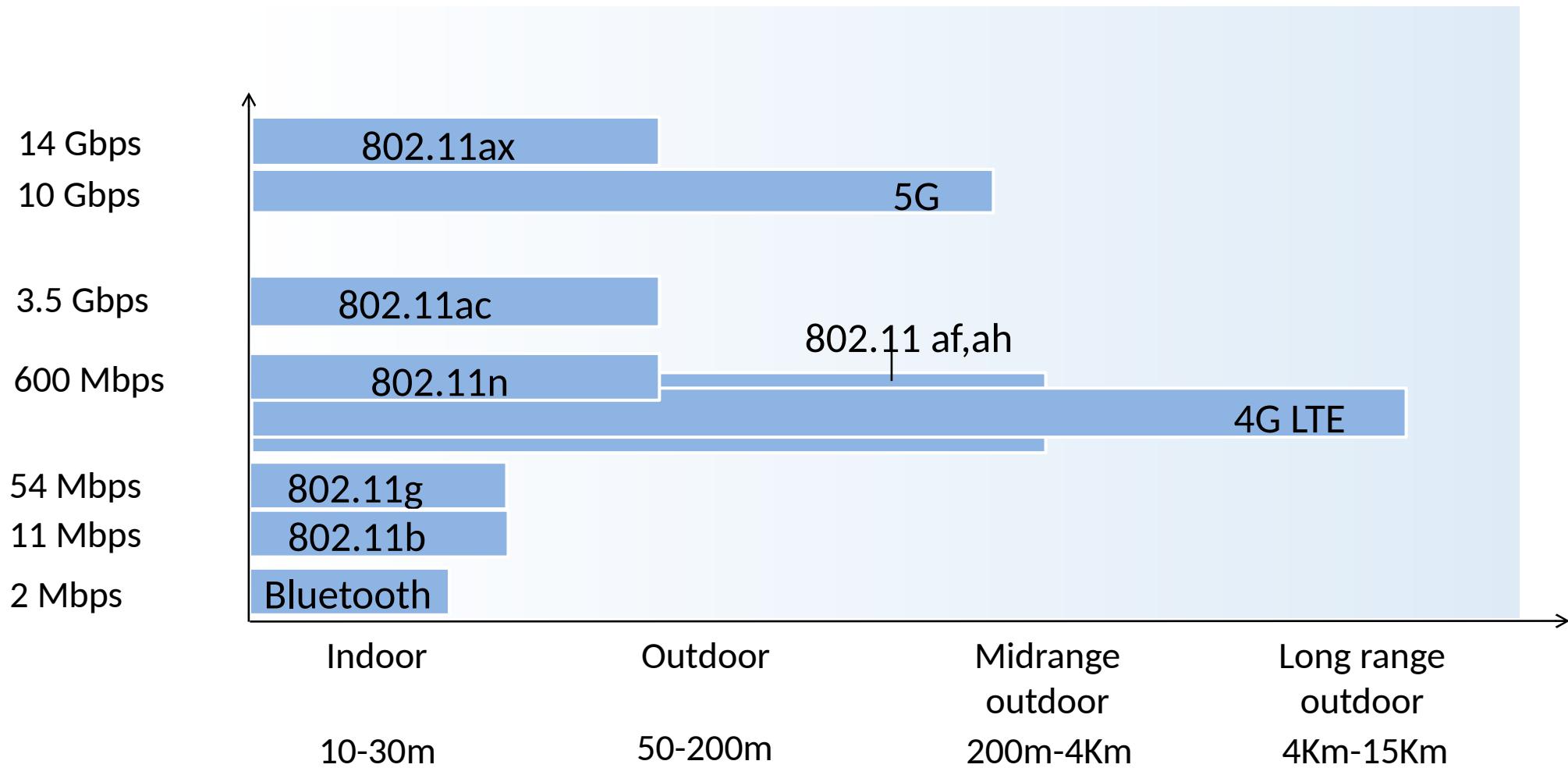
## Hidden terminal problem

- B, A hear each other
- B, C hear each other
- A, C can not hear each other means A, C unaware of their interference at B

## Signal attenuation:

- B, A hear each other
- B, C hear each other
- A, C can not hear each other interfering at B

# Characteristics of selected wireless links



# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

## 7.2.1 CDMA: code division multiple access

7.3 WiFi: 802.11 wireless LANs

~~7.4 Cellular networks: 4G and 5G~~

~~7.5 Mobility management: principles~~

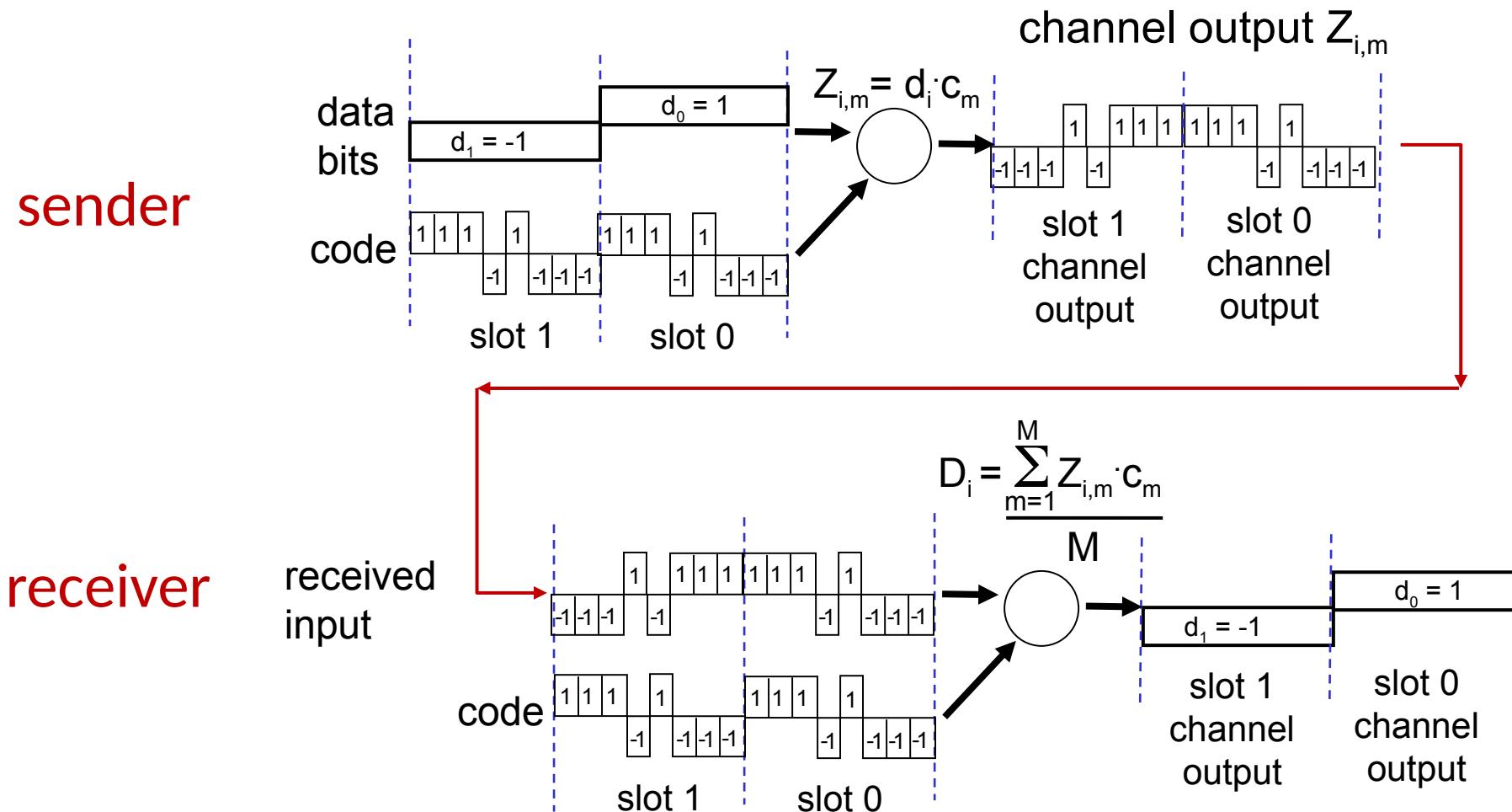
~~7.6 Mobility management: practice~~

~~7.7 Mobility: impact on higher layer protocols~~

# Code Division Multiple Access (CDMA)

- unique “code” assigned to each user; i.e., code set partitioning
  - all users share same frequency, but each user has own “chipping” sequence (i.e., code) to encode data
  - allows multiple users to “coexist” and transmit simultaneously with minimal interference (if codes are “orthogonal”)
- **encoding:** inner product: (original data)  $\times$  (chipping sequence)
- **decoding:** summed inner-product: (encoded data)  $\times$  (chipping sequence)

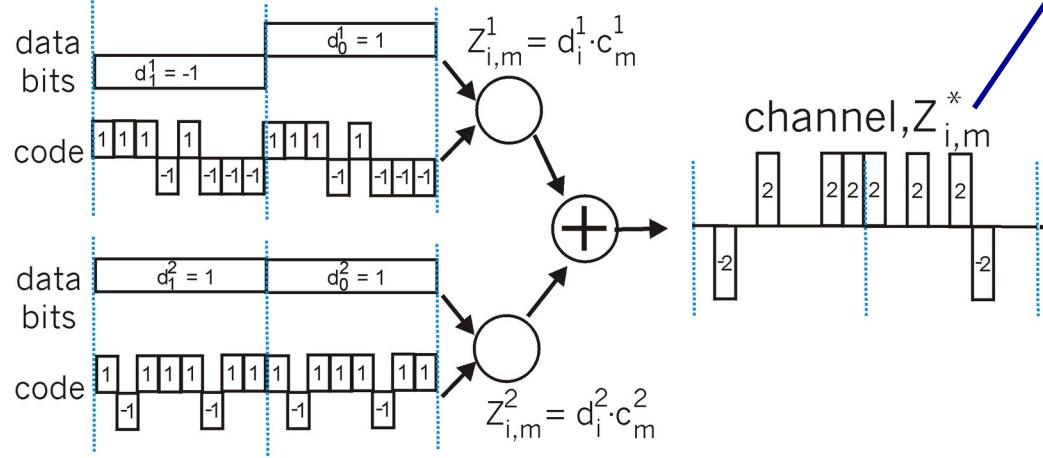
# CDMA encode/decode



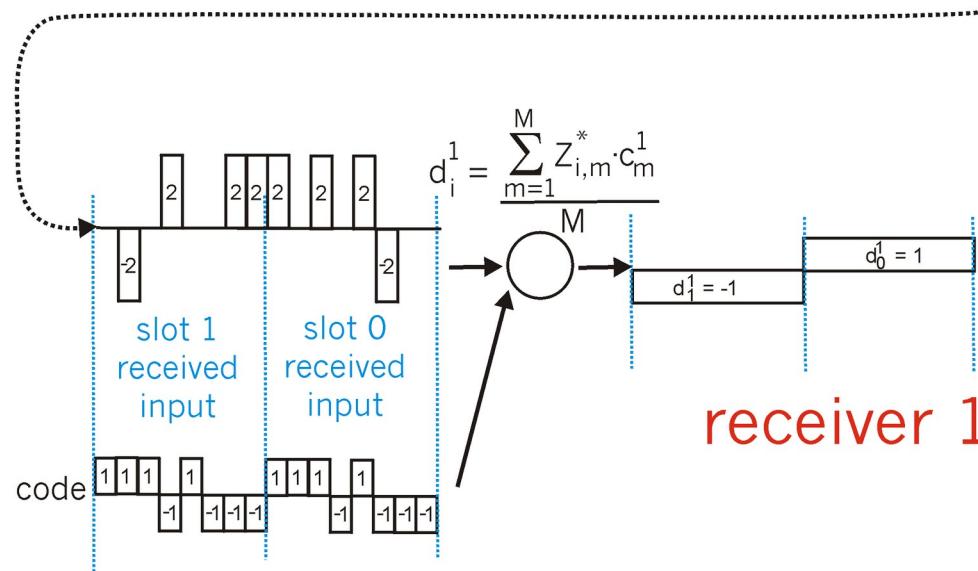
... but this isn't really useful yet!

# CDMA: two-sender interference

Sender  
1  
  
Sender  
2



channel sums together  
transmissions by sender  
1 and 2



receiver 1

using same code as sender  
1, receiver recovers sender  
1's original data from  
summed channel data!

... now *that's* useful!

# **Orthogonal Frequency Division Multiplexing (OFDM)**

# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

**7.3 WiFi: 802.11 wireless LANs**

**7.3.1 The 802.11 wireless LAN architecture**

7.3.2 The 802.11 MAC protocol

7.3.3 The 802.11 frame format

7.3.4 Mobility in the same IP subnet

7.3.5 Advanced features in 802.11

7.3.6 Personal area networks: Bluetooth

~~7.4 Cellular networks: 4G and 5G~~

~~7.5 Mobility management: principles~~

~~7.6 Mobility management: practice~~

~~7.7 Mobility: impact on higher-layer protocols~~

# IEEE 802.11 Wireless LAN

| IEEE 802.11 standard | Year | Max data rate | Range | Frequency                       |
|----------------------|------|---------------|-------|---------------------------------|
| 802.11b              | 1999 | 11 Mbps       | 30 m  | 2.4 Ghz                         |
| 802.11g              | 2003 | 54 Mbps       | 30m   | 2.4 Ghz                         |
| 802.11n (WiFi 4)     | 2009 | 600 Mbps      | 70m   | 2.4, 5 Ghz                      |
| 802.11ac (WiFi 5)    | 2013 | 3.47Gpbs      | 70m   | 5 Ghz                           |
| 802.11ax (WiFi 6)    | 2020 | 14 Gbps       | 70m   | 2.4, 5 Ghz                      |
| 802.11af             | 2014 | 35 – 560 Mbps | 1 Km  | unused TV bands<br>(54-790 MHz) |
| 802.11ah             | 2017 | 347Mbps       | 1 Km  | 900 Mhz                         |

- all use CSMA/CA for multiple access, and have base-station and ad-hoc network versions

# Unlicensed ISM 2.4 GHz frequency band

- ISM radio bands: Reserved for Industrial, Scientific and Medical purposes
- Used without a government license
- Short-range, low-power wireless communications systems
- Used by low-power transmitters not considered to be ISM devices
  - Cordless phones, Bluetooth devices, near-field communication (NFC) devices, garage door openers, baby monitors, and wireless computer networks (Wi-Fi)

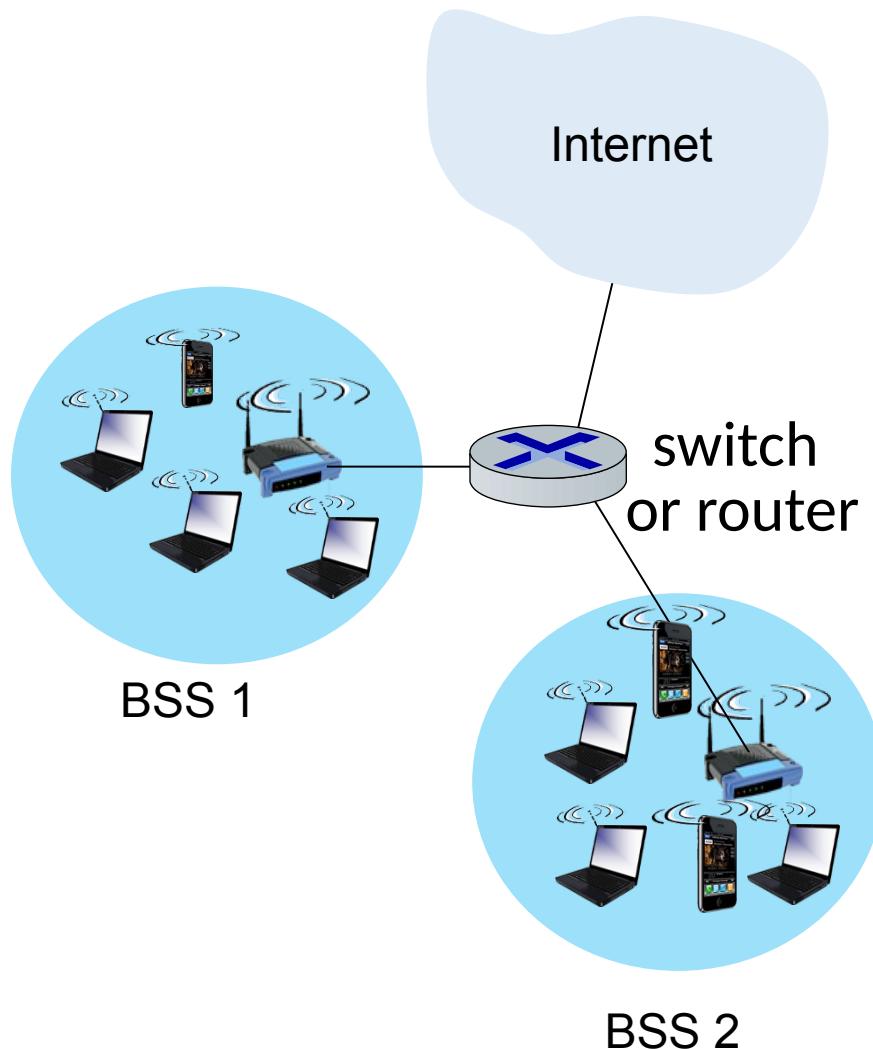
# Unlicensed 5 GHz frequency band

As of March 2021, U-NII consists of eight ranges. U-NII 1 through 4 are for 5 GHz WLAN ([802.11a](#) and newer), and 5 through 8 are for 6 GHz WLAN ([802.11ax](#)) use. U-NII 2 is further divided into three subsections.

**U-NII bands and FCC regs**

| Name     | Aliases                                       | Freq. Range (GHz) | Bandwidth (MHz) | Max Power (mW) | Max EIRP (mW) |
|----------|-----------------------------------------------|-------------------|-----------------|----------------|---------------|
| U-NII-1  | U-NII Low / U-NII Indoor                      | 5.150–5.250       | 100             | 50             | 200           |
| U-NII-2A | U-NII Mid                                     | 5.250–5.350       | 100             | 250            | 1,000         |
| U-NII-2B |                                               | 5.350–5.470       | 120             | —              | —             |
| U-NII-2C | U-NII Worldwide / U-NII-2-Extended / U-NII-2e | 5.470–5.725       | 255             | 250            | 1,000         |
| U-NII-3  | U-NII Upper                                   | 5.725–5.850       | 125             | 1,000          | 4,000         |
| U-NII-4  | DSRC/ITS                                      | 5.850–5.925       | 75              | —              | —             |
| U-NII-5  |                                               | 5.925–6.425       | 500             | —              | —             |
| U-NII-6  |                                               | 6.425–6.525       | 100             | —              | —             |
| U-NII-7  |                                               | 6.525–6.875       | 350             | —              | —             |
| U-NII-8  |                                               | 6.875–7.125       | 250             | —              | —             |

# 802.11 LAN architecture

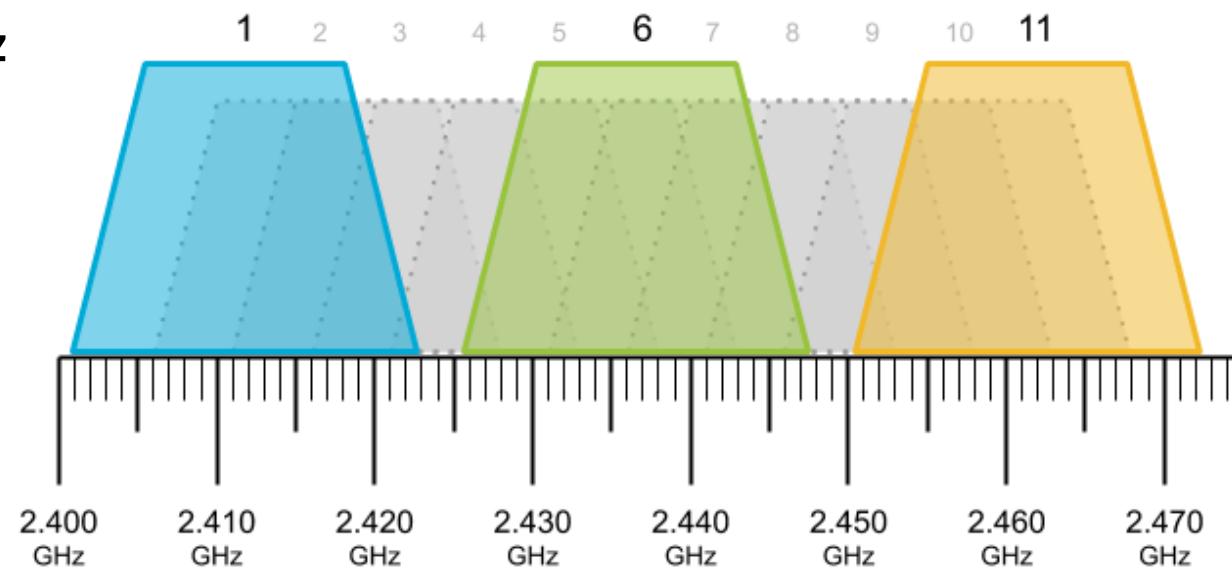


- **Basic Service Set (BSS)**  
a single **access point (AP)** interconnecting with wireless hosts
- **Basic Service Area (BSA) (aka. cell)**  
is the area that is covered by the **access point 's signal**

# 802.11: Channels

- spectrum **divided into channels** at different frequencies
  - AP admin chooses frequency for AP
  - interference possible: channel can be same as that chosen by neighboring AP!

Example: 2.4 GHz

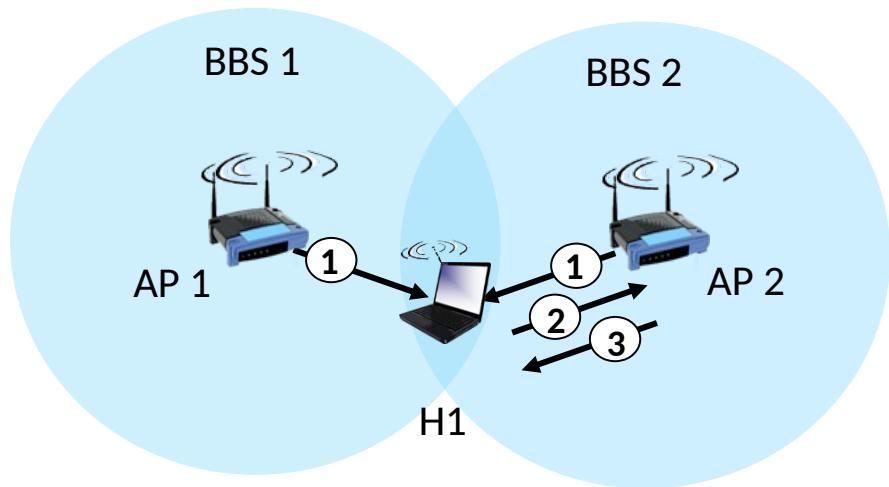


# 802.11: Channels, association

- arriving host: must **associate** with an AP
  - scans channels, listening for *beacon frames* containing AP's name (SSID) and MAC address
  - selects AP to associate with
  - then may perform authentication [Chapter 8]
  - then typically run DHCP to get IP address in AP's subnet

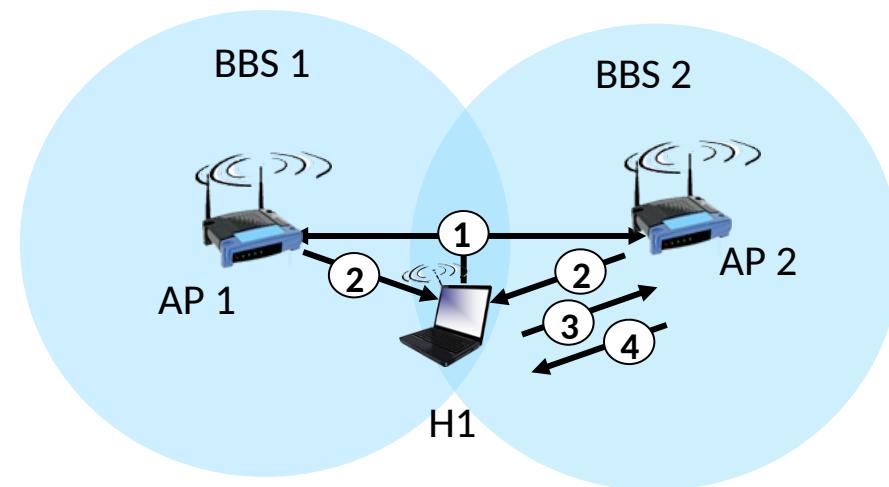


# 802.11: passive/active scanning



## passive scanning:

- (1) beacon frames sent from APs
- (2) association Request frame sent: H1 to selected AP
- (3) association Response frame sent from selected AP to H1



## active scanning:

- (1) Probe Request frame broadcast from H1
- (2) Probe Response frames sent from APs
- (3) Association Request frame sent: H1 to selected AP
- (4) Association Response frame sent from selected AP to H1

# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

## 7.3 WiFi: 802.11 wireless LANs

7.3.1 The 802.11 wireless LAN architecture

### 7.3.2 The 802.11 MAC protocol

7.3.3 The 802.11 frame format

7.3.4 Mobility in the same IP subnet

7.3.5 Advanced features in 802.11

7.3.6 Personal area networks: Bluetooth

~~7.4 Cellular networks: 4G and 5G~~

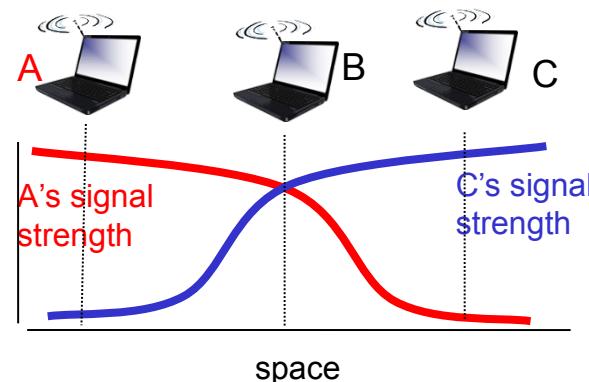
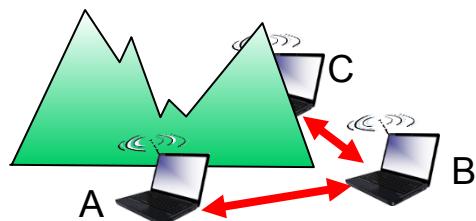
~~7.5 Mobility management: principles~~

~~7.6 Mobility management: practice~~

~~7.7 Mobility: impact on higher-layer protocols~~

# IEEE 802.11: multiple access

- avoid collisions: 2+ nodes transmitting at same time
- 802.11: CSMA - sense before transmitting
  - don't collide with detected ongoing transmission by another node
- 802.11: no collision detection!
  - difficult to sense collisions: high transmitting signal, weak received signal due to fading
  - can't sense all collisions in any case: hidden terminal, fading
  - goal: *avoid collisions*: CSMA/CollisionAvoidance



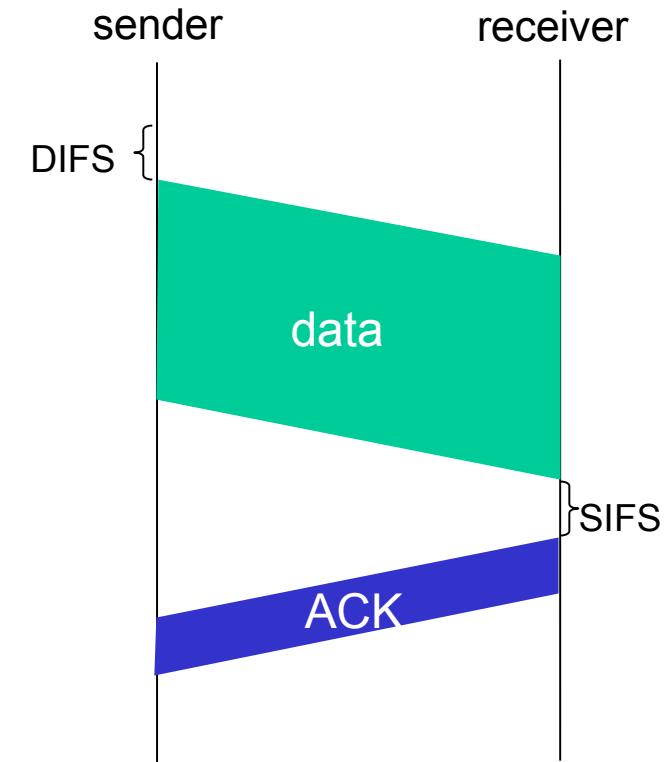
# IEEE 802.11 MAC Protocol: CSMA/CA

## 802.11 sender

- 1 if sense channel idle for **DIFS** then  
    transmit entire frame (no CD)
- 2 if sense channel busy then  
    start random backoff time  
    timer counts down while channel idle
- 3 transmit when timer expires  
    if no ACK, increase random backoff interval, repeat 2

## 802.11 receiver

if frame received OK  
    return ACK after **SIFS** (ACK needed due to hidden  
    terminal problem)

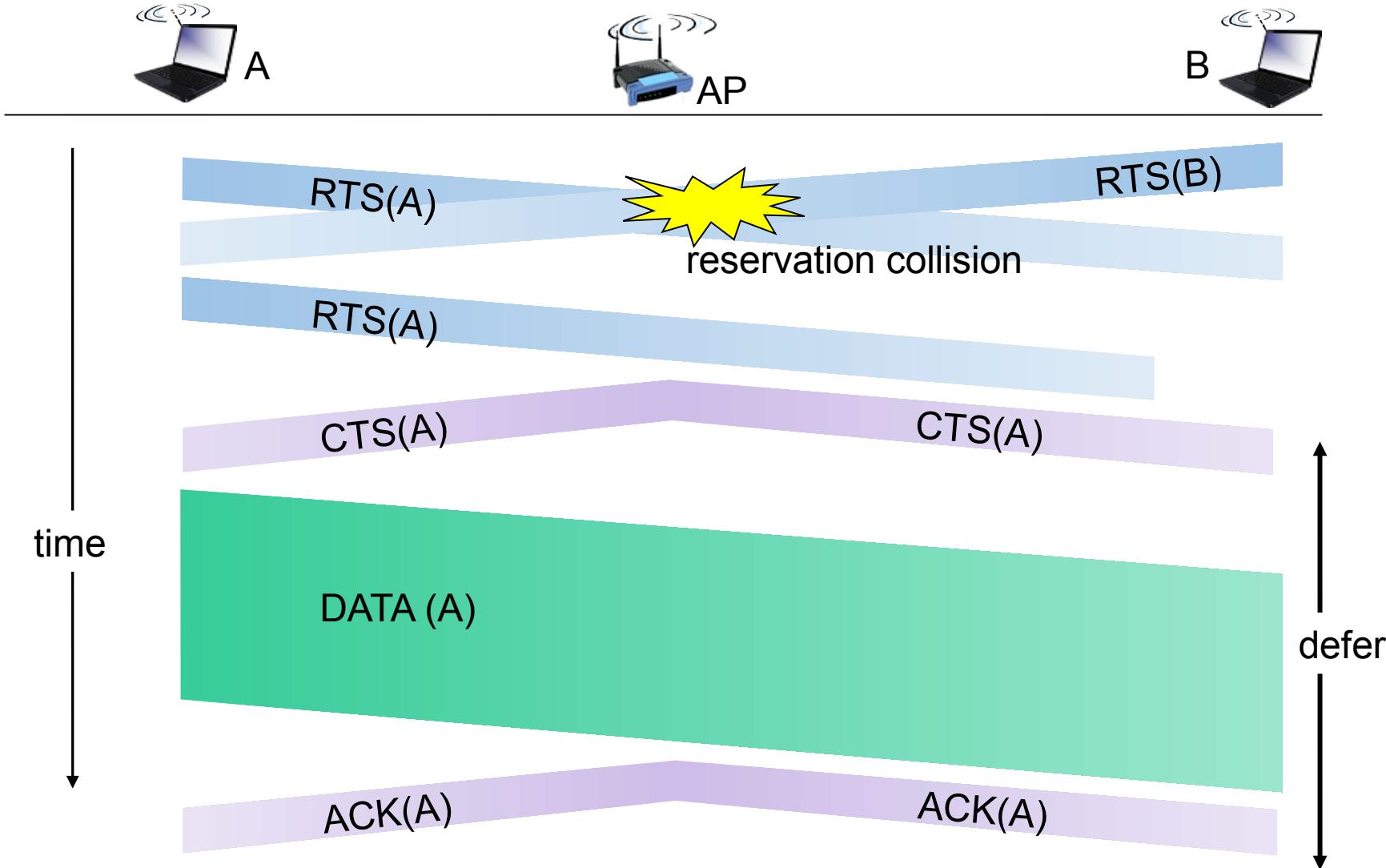


# Avoiding collisions: RTS-CTS exchange

**idea:** sender “reserves” channel use for data frames using small reservation packets

- sender first transmits *small request-to-send* (RTS) packet to AP using CSMA
  - RTSs may still collide with each other (but they are short)
- AP broadcasts **clear-to-send** (CTS) in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

# Collision Avoidance: RTS-CTS exchange



# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

## 7.3 WiFi: 802.11 wireless LANs

7.3.1 The 802.11 wireless LAN architecture

7.3.2 The 802.11 MAC protocol

### 7.3.3 The 802.11 frame format

7.3.4 Mobility in the same IP subnet

7.3.5 Advanced features in 802.11

7.3.6 Personal area networks: Bluetooth

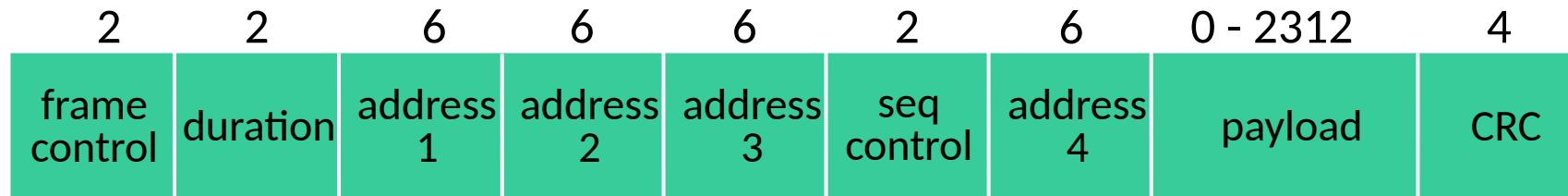
7.4 Cellular networks: 4G and 5G

7.5 Mobility management: principles

7.6 Mobility management: practice

7.7 Mobility: impact on higher-layer protocols

# 802.11 frame: addressing



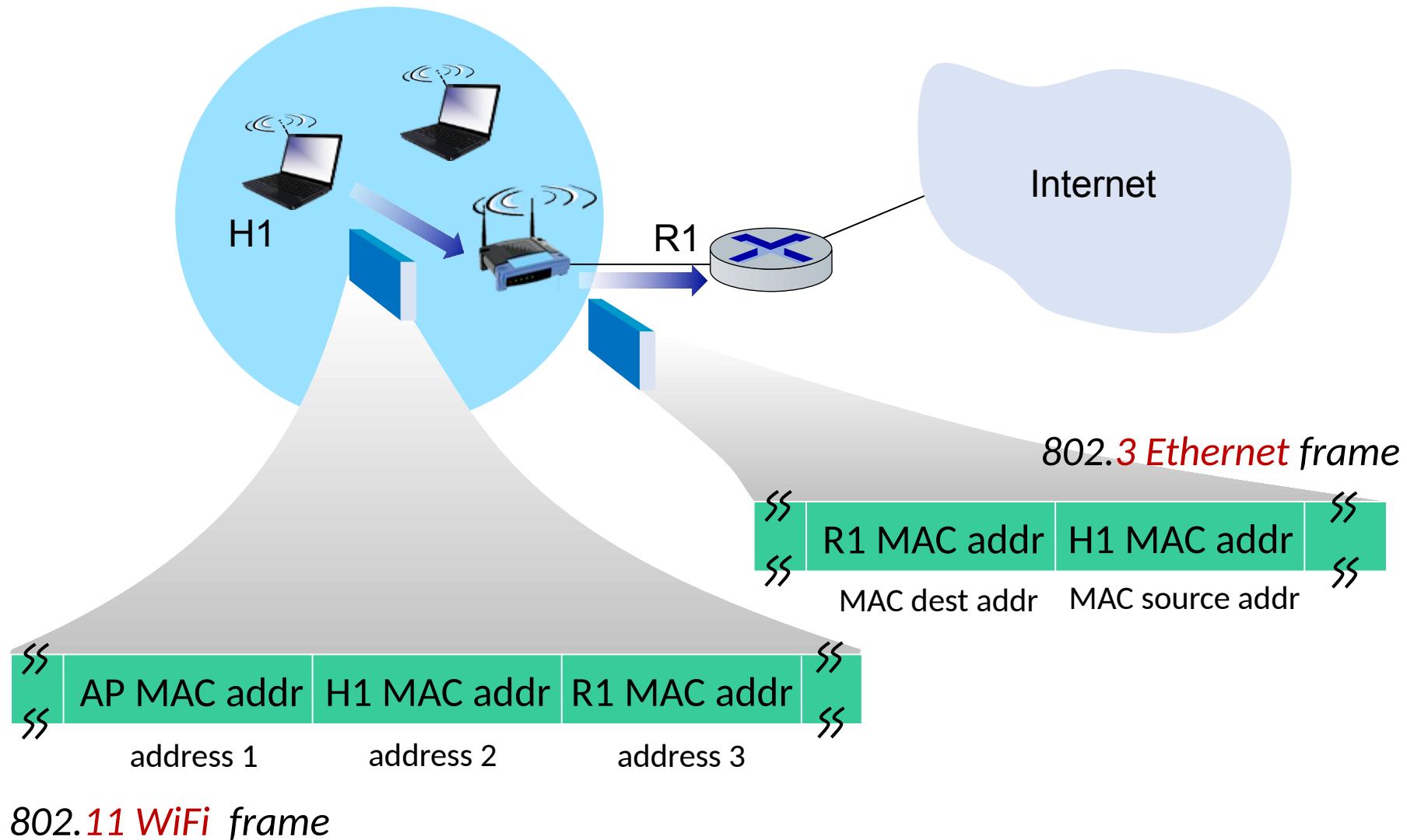
**Address 1:** MAC address of wireless host or AP to receive this frame

**Address 2:** MAC address of wireless host or AP transmitting this frame

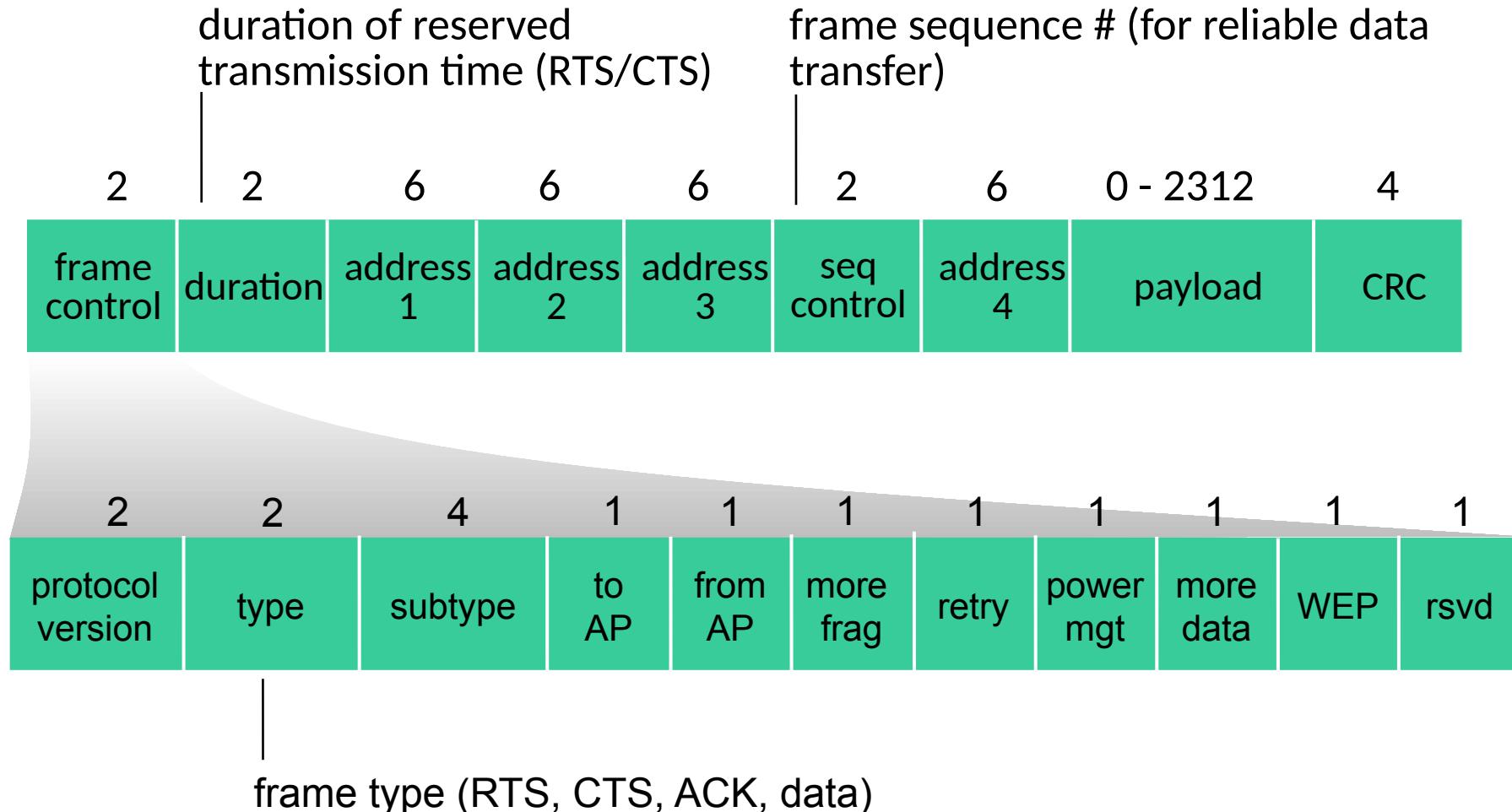
**Address 4:** used only in ad hoc mode

**Address 3:** MAC address of frame destination or source (in infrastructure mode)

# 802.11 frame: addressing - host to router



# 802.11 frame: addressing



# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

## 7.3 WiFi: 802.11 wireless LANs

7.3.1 The 802.11 wireless LAN architecture

7.3.2 The 802.11 MAC protocol

7.3.3 The 802.11 frame format

### 7.3.4 Mobility in the same IP subnet

7.3.5 Advanced features in 802.11

7.3.6 Personal area networks: Bluetooth

7.4 Cellular networks: 4G and 5G

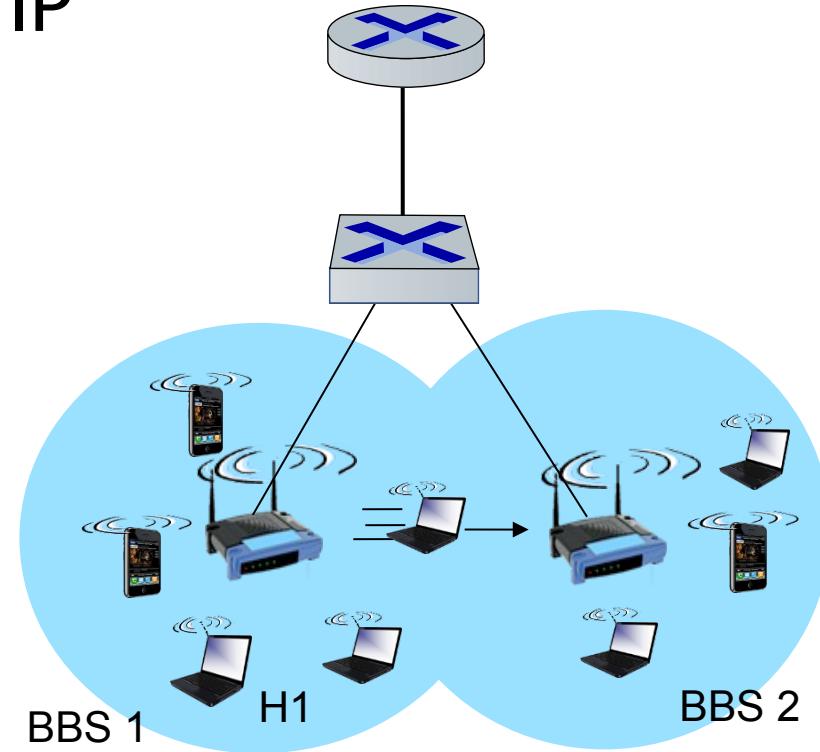
7.5 Mobility management: principles

7.6 Mobility management: practice

7.7 Mobility: impact on higher-layer protocols

# 802.11: mobility within same subnet

- H1 remains in same IP subnet: IP address can remain same
- switch: which AP is associated with H1?
  - self-learning (Ch. 6): switch will see frame from H1 and “remember” which switch port can be used to reach H1



# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

## 7.3 WiFi: 802.11 wireless LANs

7.3.1 The 802.11 wireless LAN architecture

7.3.2 The 802.11 MAC protocol

7.3.3 The 802.11 frame format

7.3.4 Mobility in the same IP subnet

### 7.3.5 Advanced features in 802.11

7.3.6 Personal area networks: Bluetooth

7.4 Cellular networks: 4G and 5G

7.5 Mobility management: principles

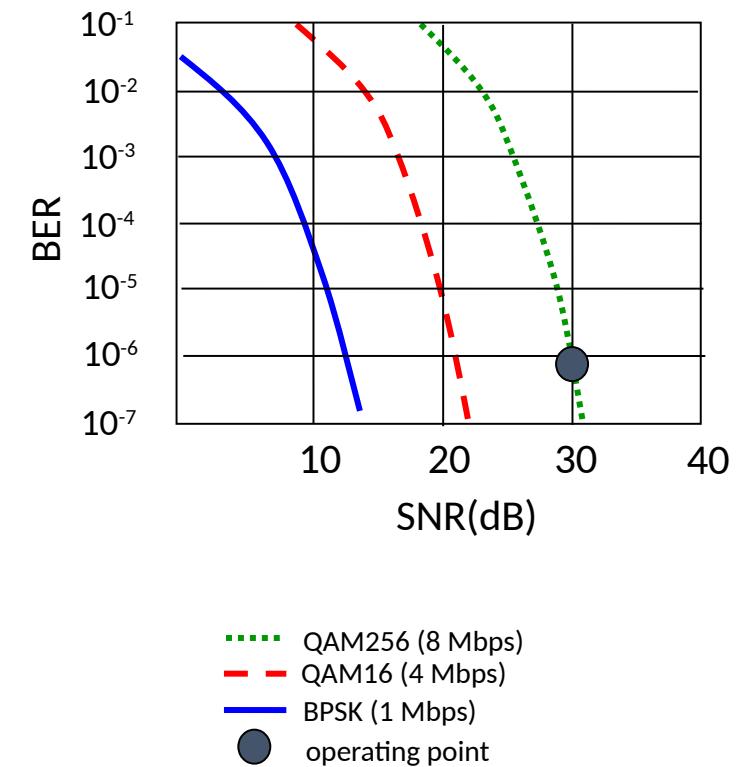
7.6 Mobility management: practice

7.7 Mobility: impact on higher-layer protocols

# 802.11: advanced capabilities

## Rate adaptation

- base station, mobile dynamically change transmission rate (physical layer modulation technique) as mobile moves, SNR varies
  1. SNR decreases, BER increase as node moves away from base station
  2. When BER becomes too high, switch to lower transmission rate but with lower BER



# 802.11: advanced capabilities

## power management

- node-to-AP: “I am going to sleep until next beacon frame”
  - AP knows not to transmit frames to this node
  - node wakes up before next beacon frame
- beacon frame: contains list of mobiles with AP-to-mobile frames waiting to be sent
  - node will stay awake if AP-to-mobile frames to be sent; otherwise sleep again until next beacon frame

# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

## 7.3 WiFi: 802.11 wireless LANs

7.3.1 The 802.11 wireless LAN architecture

7.3.2 The 802.11 MAC protocol

7.3.3 The 802.11 frame format

7.3.4 Mobility in the same IP subnet

7.3.5 Advanced features in 802.11

## 7.3.6 Personal area networks: Bluetooth

7.4 Cellular networks: 4G and 5G

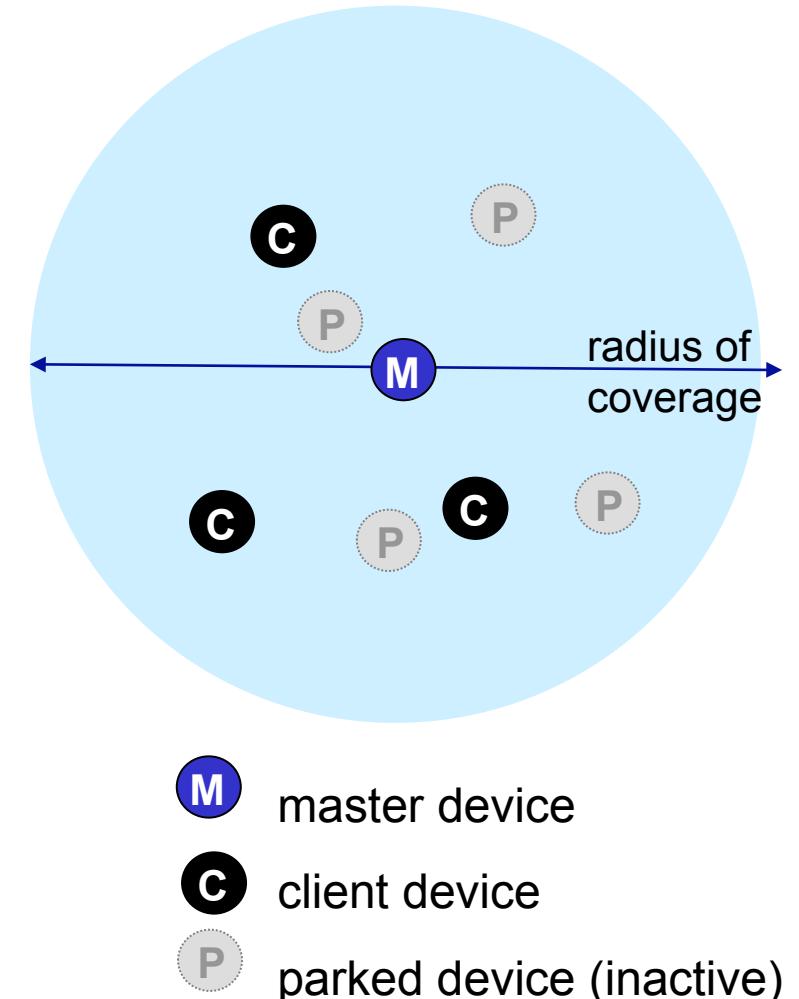
7.5 Mobility management: principles

7.6 Mobility management: practice

7.7 Mobility: impact on higher-layer protocols

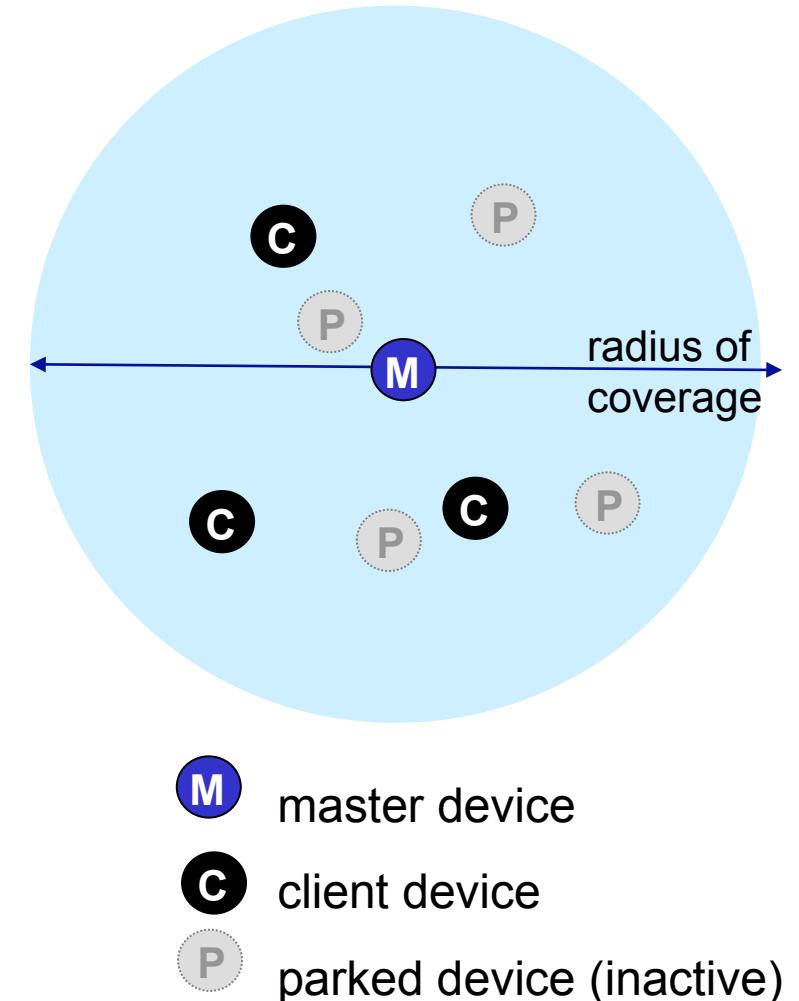
# Personal area networks: Bluetooth

- less than 10 m diameter
- replacement for cables (mouse, keyboard, headphones)
- ad hoc: no infrastructure
- master controller / clients devices:
  - master polls clients, grants requests for client transmissions



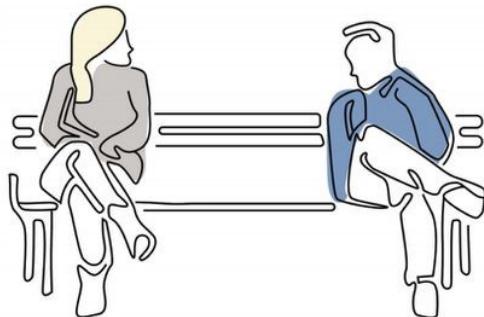
# Personal area networks: Bluetooth

- 2.4-2.5 GHz ISM radio band, up to 2 Mbps
- TDM, 625  $\mu$ sec sec. slot
- FDM: sender uses 79 frequency channels in known, pseudo-random order slot-to-slot (spread spectrum)
  - other devices/equipment not in piconet only interfere in some slots
- **parked mode:** clients can “go to sleep” (park) and later wakeup (to preserve battery)
- **bootstrapping:** nodes self-assemble (plug and play) into piconet

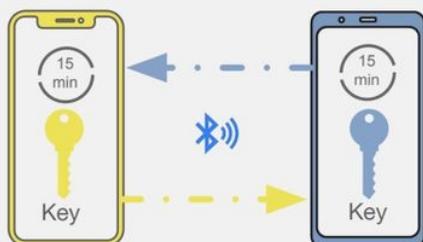


# Pandemic + Bluetooth

Alice and Bob meet each other for the first time and have a 10-minute conversation.



Their phones exchange anonymous identifier beacons (which change frequently).



A few days later...



Bob is positively diagnosed for COVID-19 and enters the test result in an app from a public health authority.



With Bob's consent, his phone uploads the last 14 days of keys for his broadcast beacons to the cloud.

Apps can only get more information via user consent



Apple | Google



# Chapter 7 outline

7.1 Introduction

7.2 Wireless Links and network characteristics

7.3 WiFi: 802.11 wireless LANs

~~7.4 Cellular networks: 4G and 5G~~

~~7.5 Mobility management: principles~~

~~7.6 Mobility management: practice~~

~~7.7 Mobility: impact on higher layer protocols~~

# Chapter 7 summary

## Wireless

- Wireless Links and network characteristics
- Wi-Fi: 802.11 wireless LANs



# Chapter 7 summary

## Wireless

- Wireless Links and network characteristics
- Wi-Fi: 802.11 wireless LANs



# Chapter 8

# Security in

# Computer

# Networks

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.

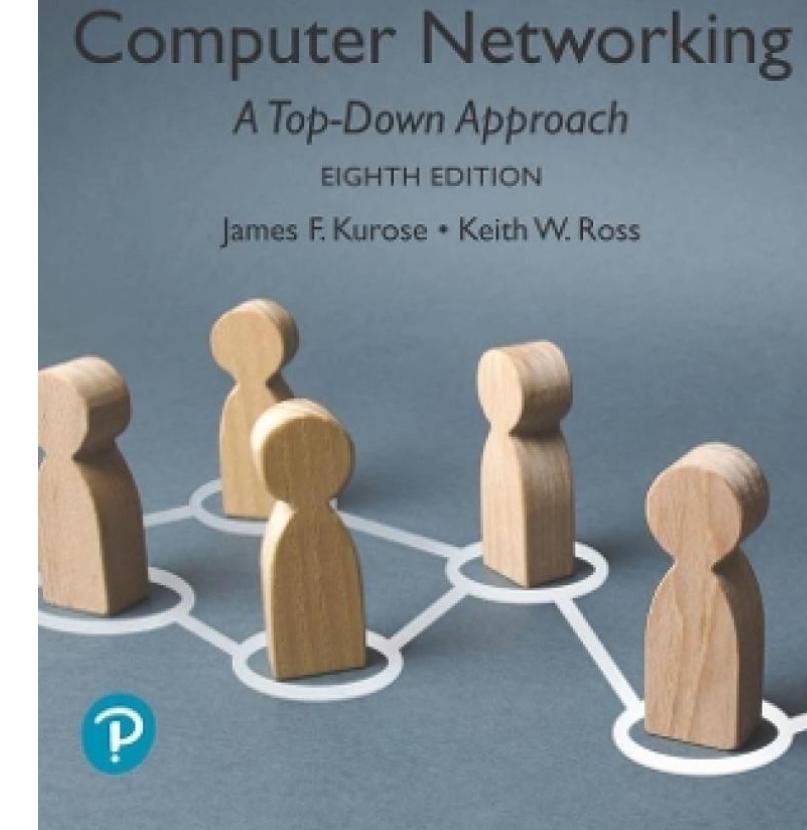
In return for use, we only ask the following:

- 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*

EIGHTH EDITION

James F. Kurose • Keith W. Ross



Computer Networking: A  
*Top-Down Approach*

8<sup>th</sup> edition

Jim Kurose, Keith Ross  
Pearson, 2020

# Security: overview

## Chapter goals:

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

# Chapter 8 outline

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and digital signatures

~~8.4 Authentication~~

8.5 Securing e-mail

8.6 Securing TCP connections: TLS

~~8.7 Network layer security: IPsec~~

~~8.8 Security in wireless and mobile networks~~

8.9 Operational security: firewalls and IDS

# Network security: Security properties

**confidentiality:** only the intended receiver should read messages

**data integrity:** the receiver wants to ensure that data have not been altered

**message authenticity, data origin authentication:**

1. confirm that the message came from the stated sender (**sender authenticity**)
2. confirm that the message has not been changed (**integrity**)

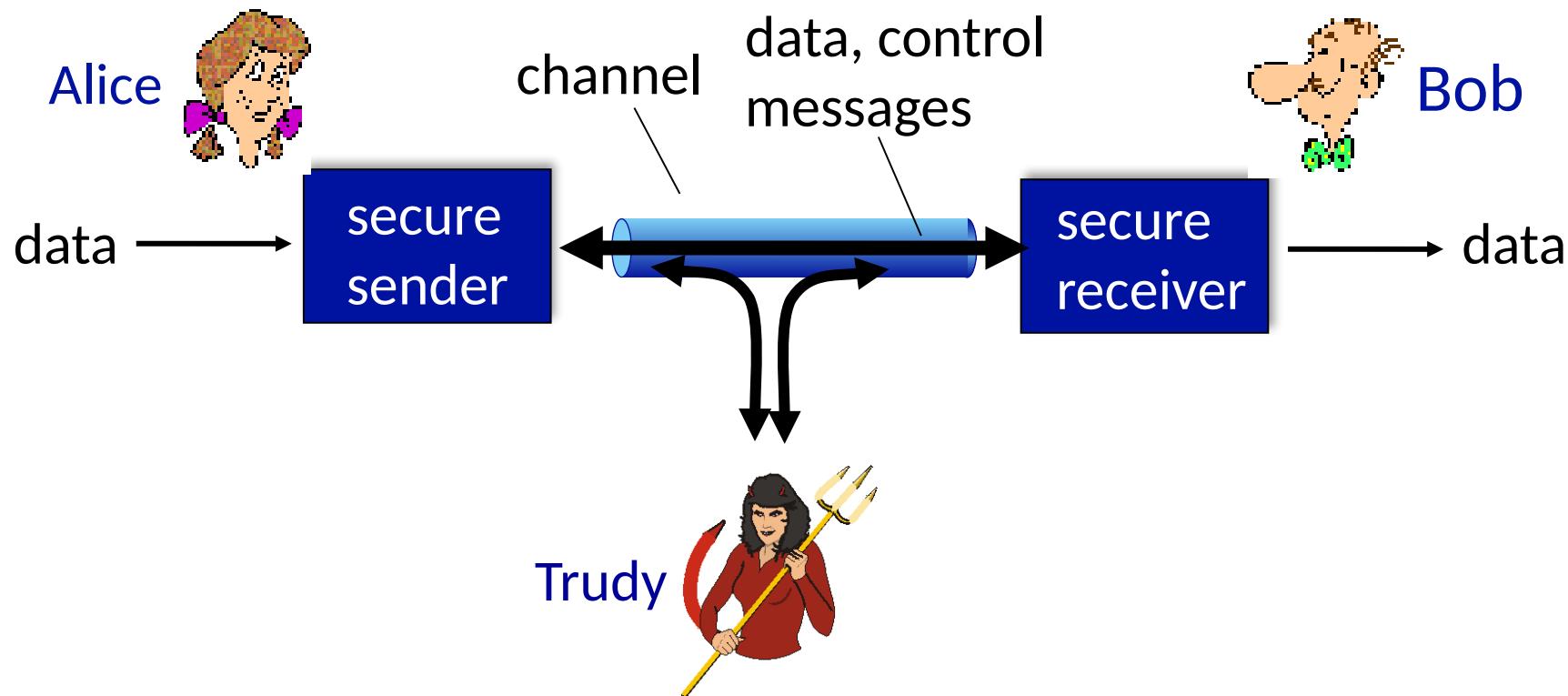
**entity authentication:**

- Alice and Bob want to confirm the identities of each other during a session

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

# Friends and enemies: Alice, Bob, Trudy

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



# Types of attacks (1)

- eavesdropping: intercept messages
  - ⌚ confidentiality breach
- data modification
  - ⌚ integrity breach
  - ⌚ authenticity breach
- impersonation, spoofing
  - ⌚ authenticity breach
- replay attacks
  - ⌚ “sort of” authenticity breach

# Types of attacks (2)

## System-oriented attacks:

- **denial of service:** prevent service from being used by others (e.g., by overloading resources) ☽ availability breach
- **hijacking:** “take over” ongoing connection by removing sender or receiver, inserting himself in place

# **Chapter 8 outline**

8.1 What is network security?

## **8.2 Principles of cryptography**

8.3 Message integrity and digital signatures

8.4 Authentication

8.5 Securing e-mail

8.6 Securing TCP connections: TLS

~~8.7 Network layer security: IPsec~~

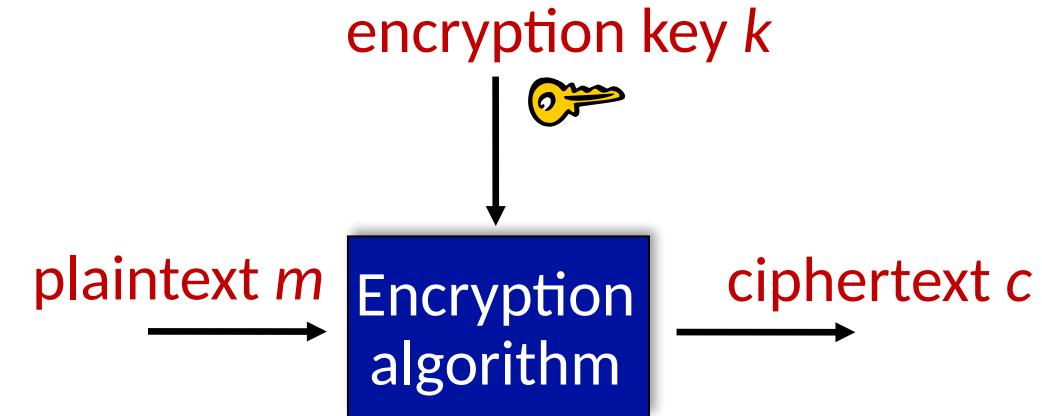
~~8.8 Security in wireless and mobile networks~~

8.9 Operational security: firewalls and IDS

# The language of cryptography

- Plaintext  $m$
- Encryption key  $k$
- Ciphertext  $c$
- Cryptographic algorithms
  - Encryption algorithm  $E$
  - Decryption algorithm  $D$

$$c = E_k(m)$$



plaintext  
ciphertext

encryption key

# The language of cryptography

- Symmetric key cryptography
  - Aka. secret key cryptography
- Asymmetric key cryptography
  - Aka. public key cryptography

# Symmetric key cryptography

Encryption key = decryption key

- Symmetric key, secret key

- Plaintext  $m$

- Ciphertext  $c$

- Cryptographic algorithms

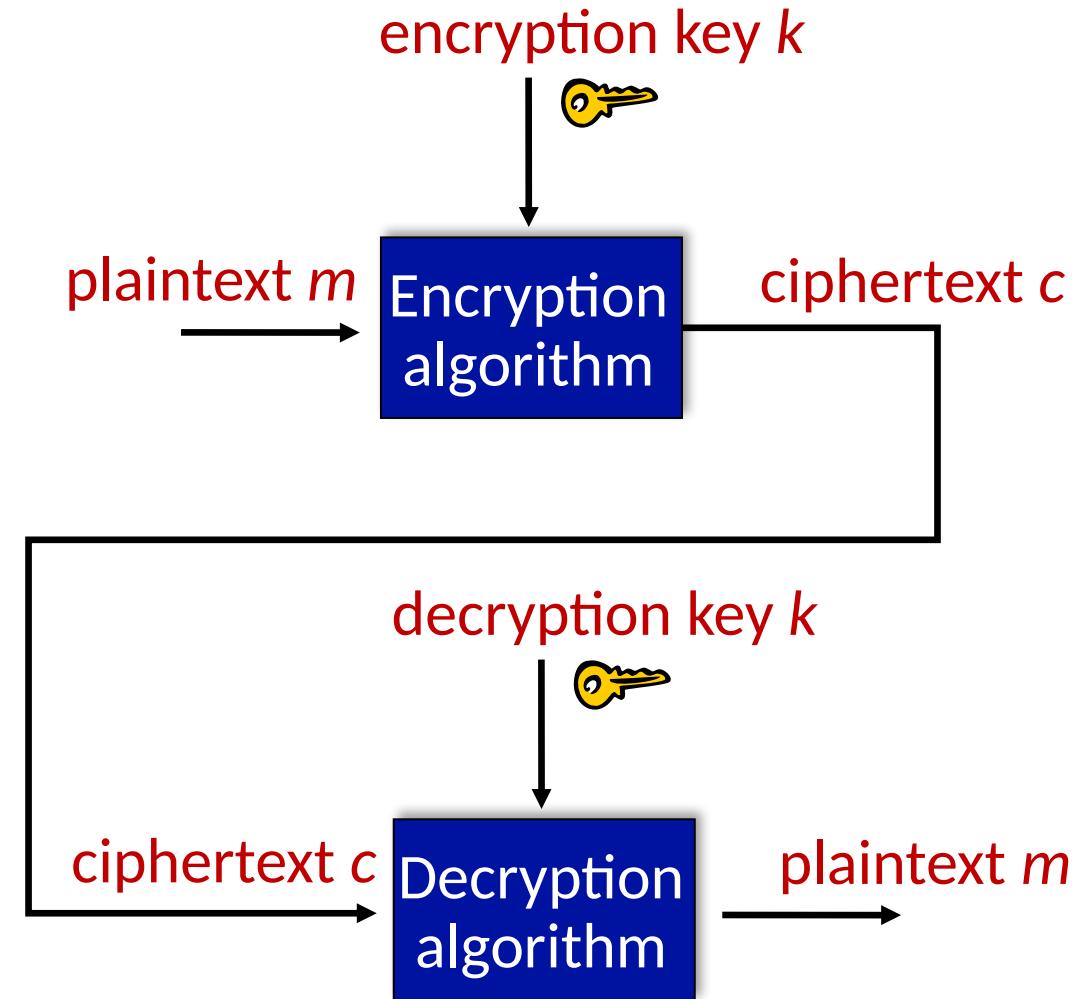
  - Encryption algorithm  $E$

  - Decryption algorithm  $D$

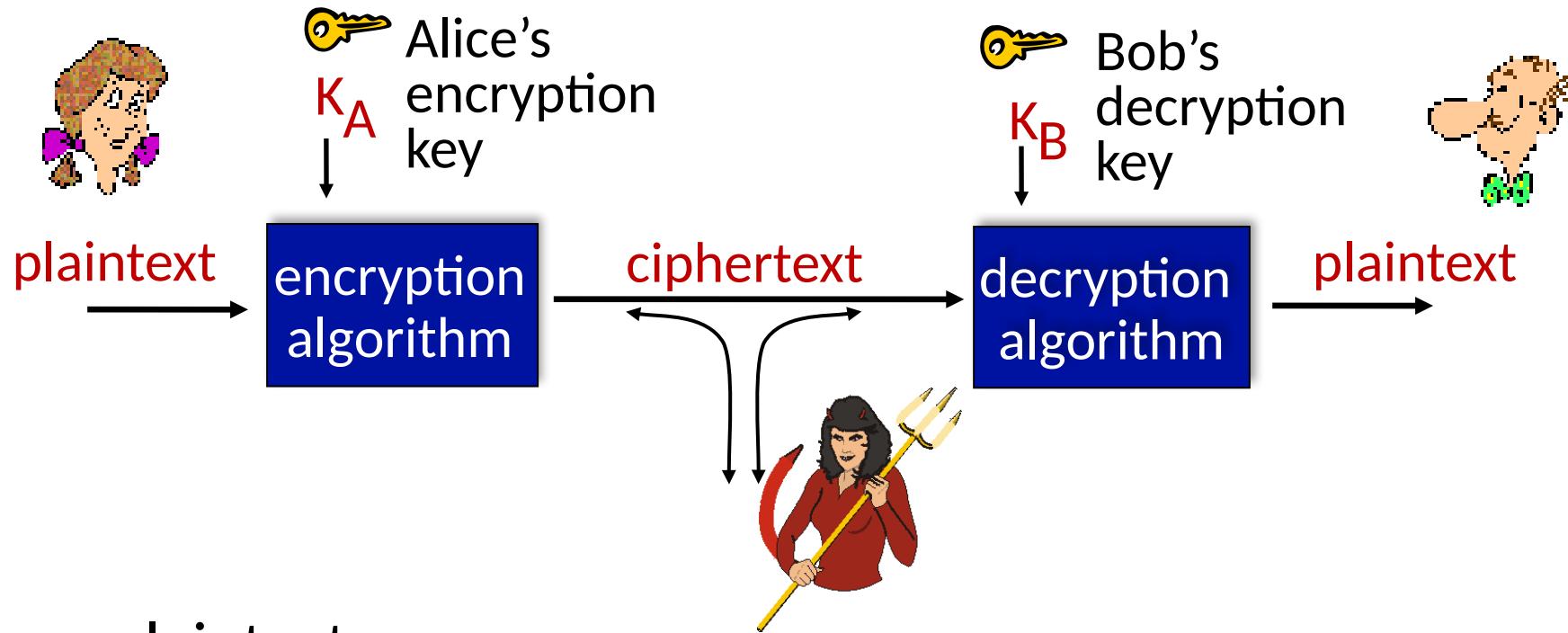
$$c = E_k(m)$$

$$m = D_k(E_k(m))$$

$$m = D_k(c)$$



# Adversary, attacker, cryptanalysis



$m$ : plaintext message

$c = E_K(m)$ : ciphertext, encrypted with key  $K_A$

$m = D_K(E_K(m))$

# Breaking an encryption scheme

- **brute force:** search through all keys
  - key space
- **known-plaintext attack:** Trudy has plaintext corresponding to ciphertext
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- **chosen-plaintext attack:** Trudy can get ciphertext for chosen plaintext

# Caesar cipher

- The rotation equivalent to the modulus operation:

- Encryption:  $C = P + K \bmod 26$
- Decryption:  $P = C - K \bmod 26$

Plaintext: bob. i love you. al:

Ciphertext: dqd. k nqxg aqw. cnk

- Trivial complexity

- With only 26 possible keys to try out, an exhaustive key search is easy
- Thus, no security

# Monoalphabetic ciphers

- Each letter in the plaintext is mapped to another letter in the ciphertext
- The mapping is always the same

*Encryption key:* 

plaintext letter: abcdefghijklmnopqrstuvwxyz

ciphertext letter: mnbvcxzasdfghjklpoiuytrewq

*Encryption:*      Plaintext: bob. i love you. alice

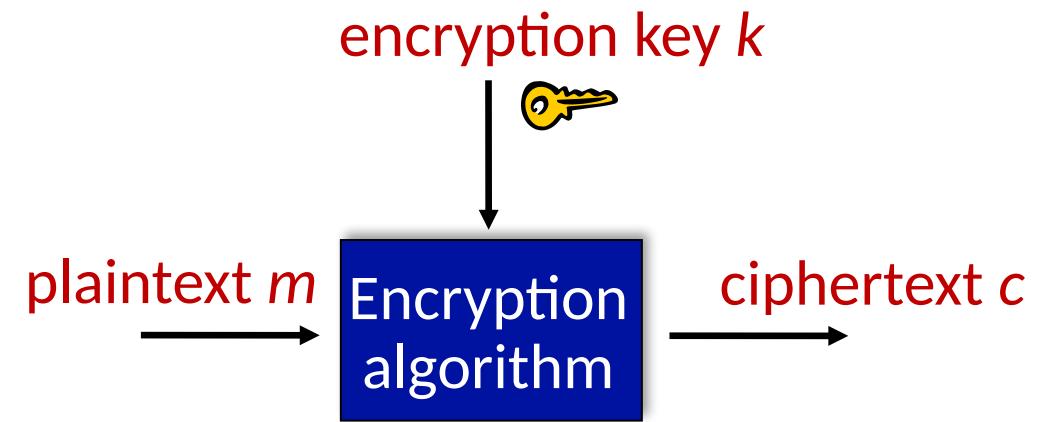
                          Ciphertext: nkn. s gktc wky. mgsbc

Mapping 26 letters to 26 letters:

$26! = 403\ 291\ 461\ 126\ 605\ 635\ 584\ 000\ 000$  possible keys

# Symmetric key crypto: Block ciphers

- Two **paired** algorithms, one for encryption and the other for decryption
- Deterministic
- **Fixed block sizes**
  - plaintext /ciphertext block of size  $n$  bits (e.g. 128 bits)
  - key size  $k$  bits (e.g. 128, 256 bits)
- Important properties:
  - Cannot tell the difference between ciphertext block and **random bits**
  - **One-way-property**
  - **Avalance effect**



# The avalanche effect

- Randomization property
- A desirable property of cryptographic algorithms, typically block ciphers and cryptographic hash functions
- If **flipping a single input bit, the output changes significantly** (e.g., half the output bits flip)
  - Both encryption (plaintext, key) and decryption (ciphertext, key)
- Very important property for block ciphers and hash functions
- A poor avalanche property can make a cipher vulnerable to cryptanalysis

# Symmetric key crypto

## DES: Data Encryption Standard

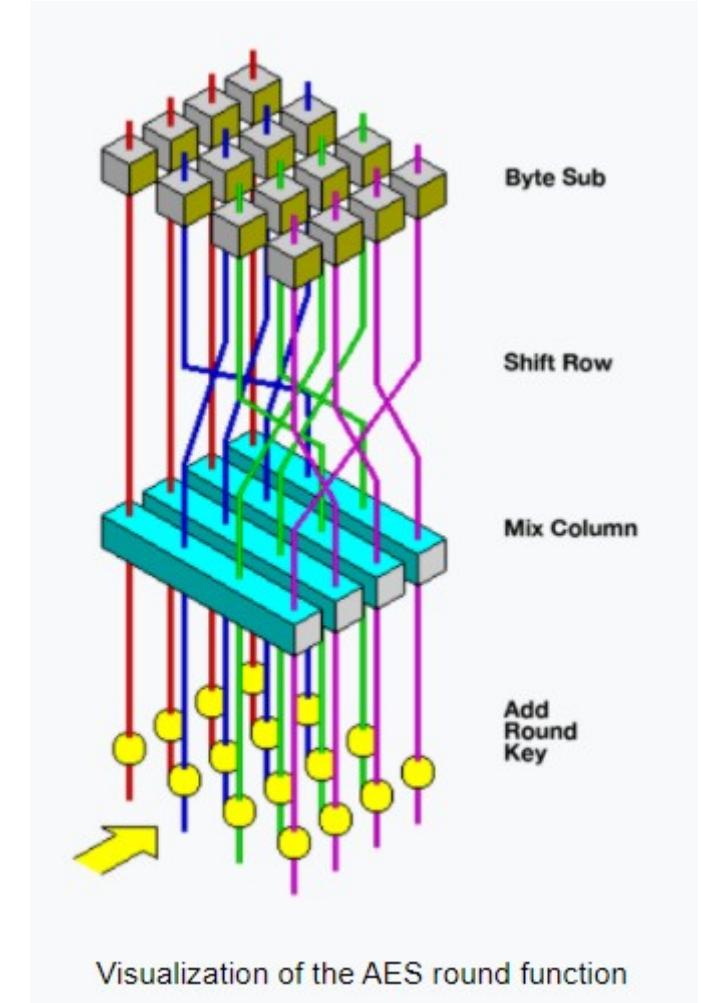
- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit block size

$$2^{56} = 72,057,594,037,927,936$$

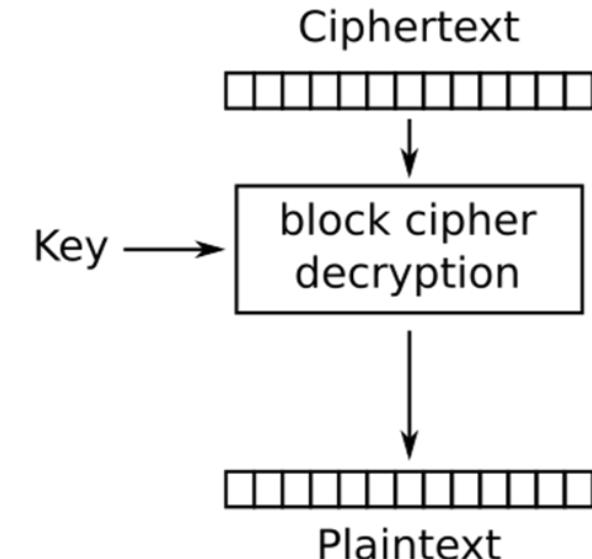
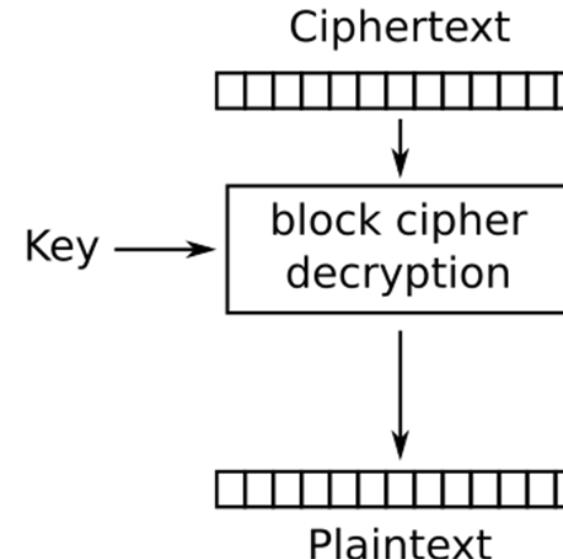
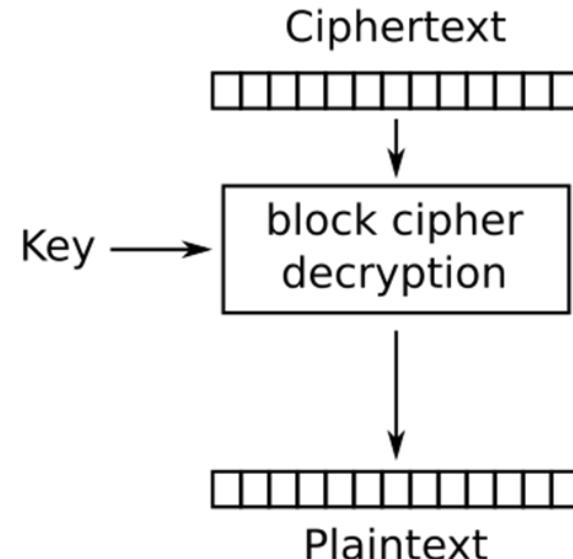
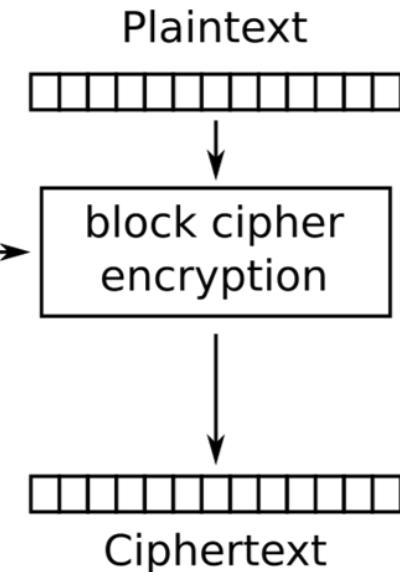
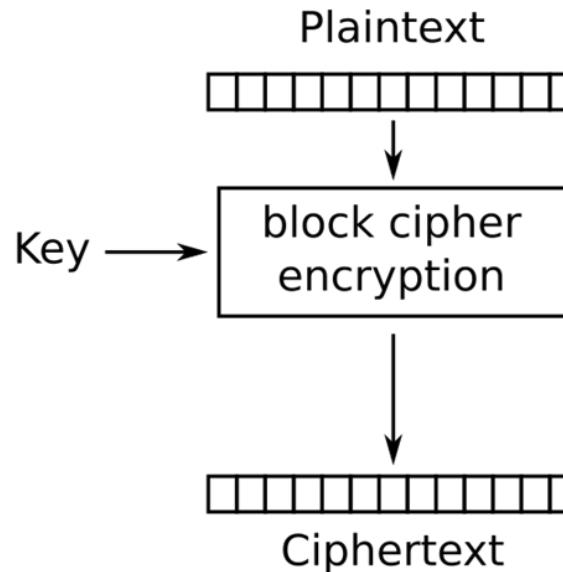
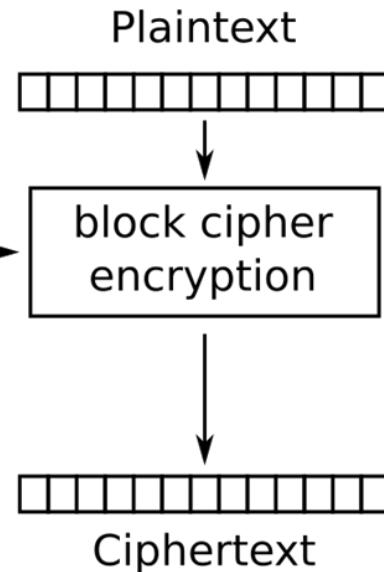
- how secure is DES?
  - DES Challenge: 56-bit-key-encrypted, phrase decrypted (brute force) in less than a day
  - no known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

# AES: Advanced Encryption Standard

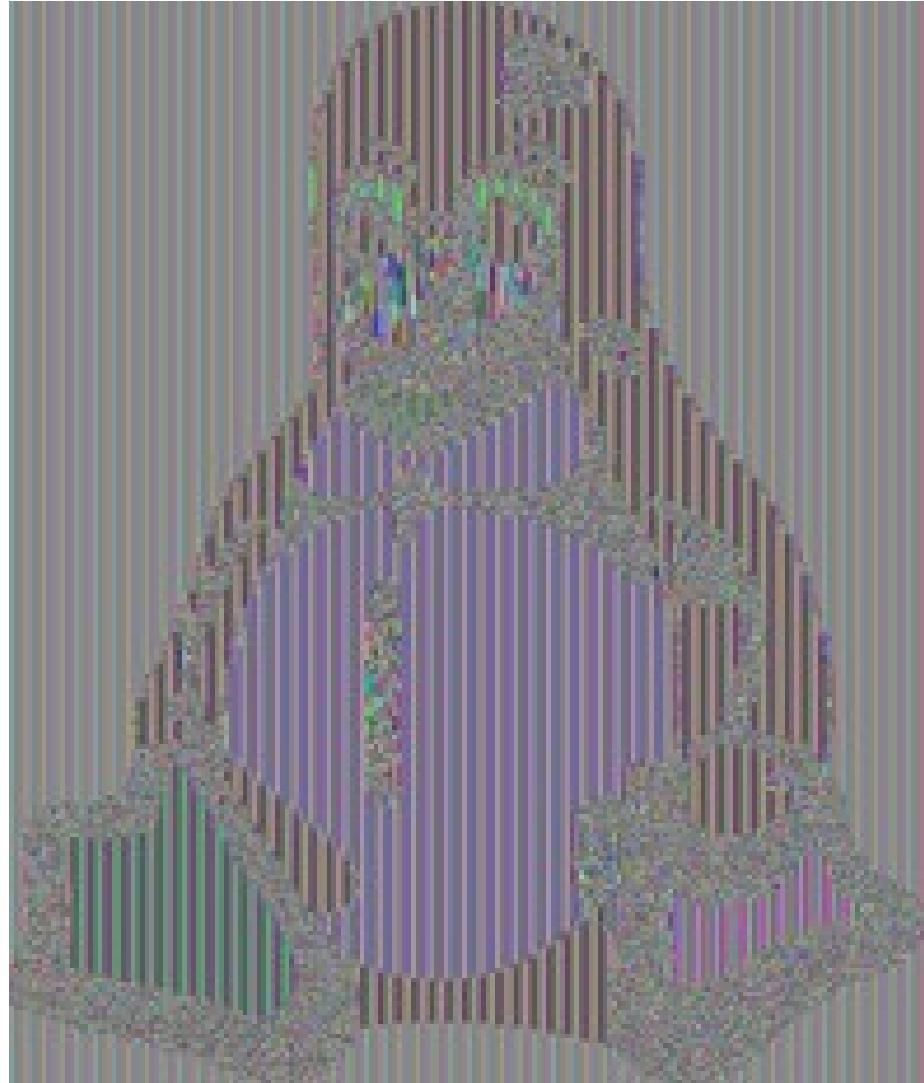
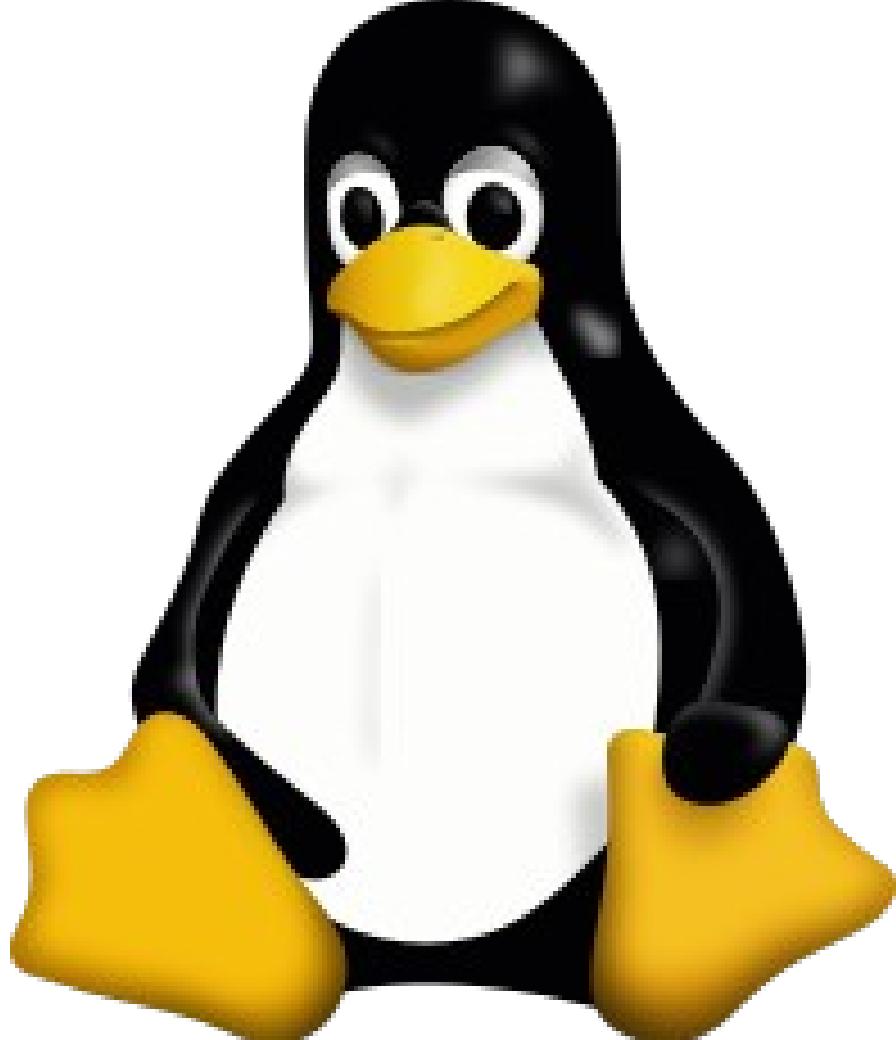
- symmetric-key NIST standard, replaced DES (Nov 2001)
- also known by its original name **Rijndael**
- processes data in 128-bit blocks
- 128-, 192-, or 256-bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES



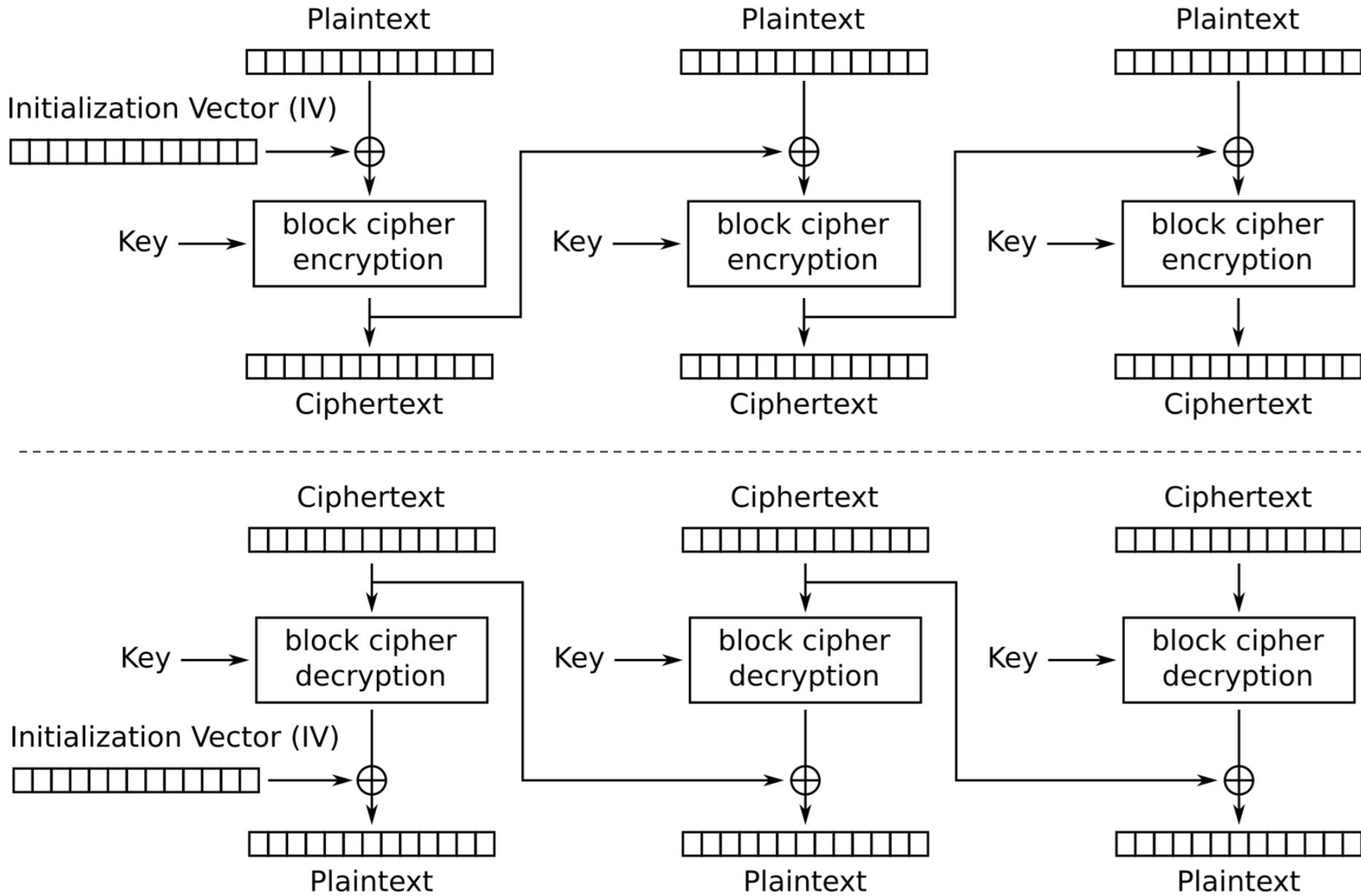
# Electronic codebook (ECB)



# ECB can reveal pattern information

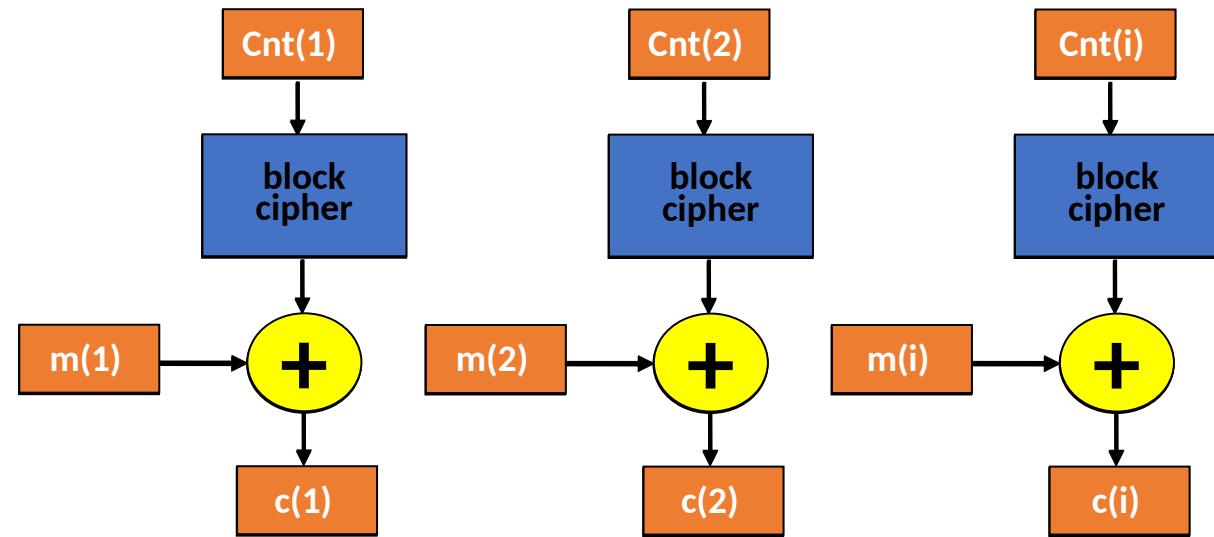


# Cipher block chaini ng

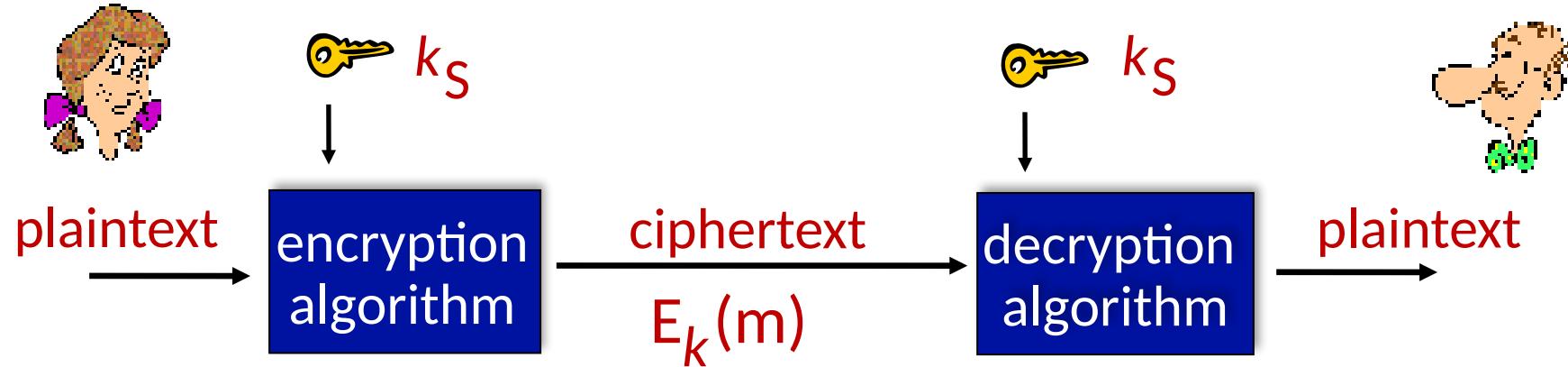


# Counter mode (CTR or CM)

- **counter mode:** a counter is incremented by one for each block and run through the cipher and XOR with message to produce the cipher text
  - receiver has got the counter start value, an Initialization Vector (IV), and encrypts the counter the same way and XOR with cipher text to produce original message



# Symmetric key cryptography



**Symmetric-key cryptography:** Bob and Alice share same secret (symmetric) key:  $k$

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

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

# Public Key Cryptography

## symmetric key crypto:

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

## public key crypto

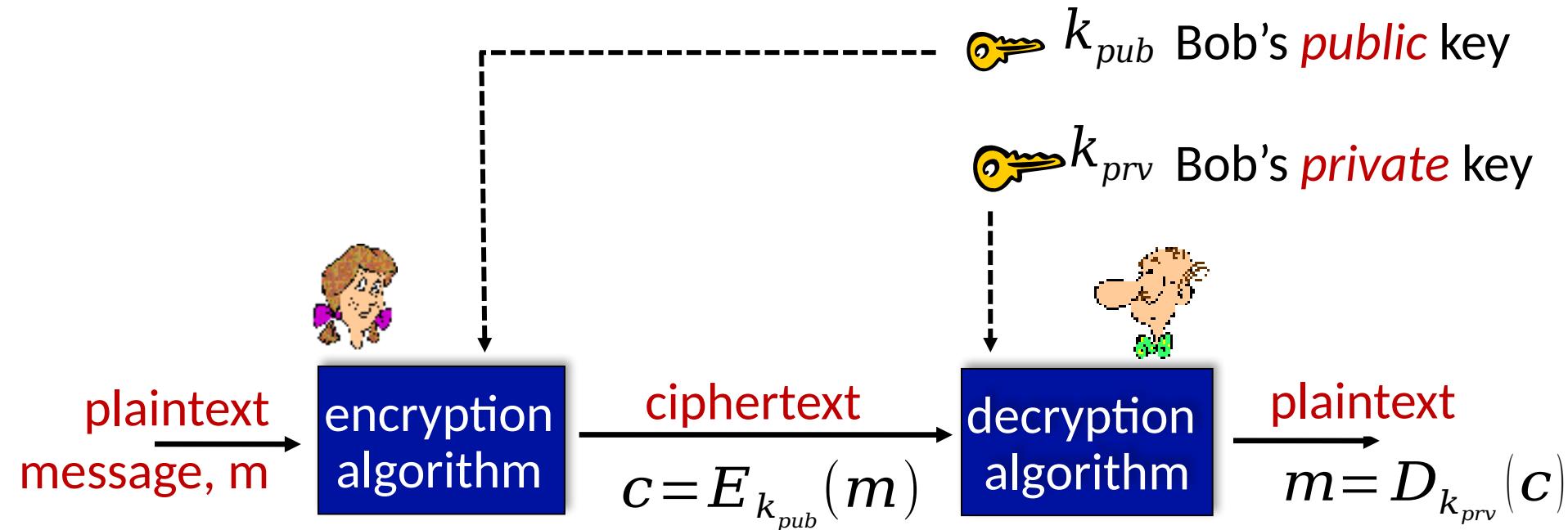
- *radically* different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver



# Public Key Cryptography

- Key pair :*
- encryption: **public key** - known by anybody
  - decryption: **private key** - known **only** by the owner

Security property: Confidentiality protection



# Public key encryption algorithms

requirements:

- ① need and such that

$$m = D_{k_{prv}}(E_{k_{pub}}(m))$$

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

# **Chapter 8 outline**

8.1 What is network security?

8.2 Principles of cryptography

## **8.3 Message integrity and digital signatures**

8.4 Authentication

8.5 Securing e-mail

8.6 Securing TCP connections: TLS

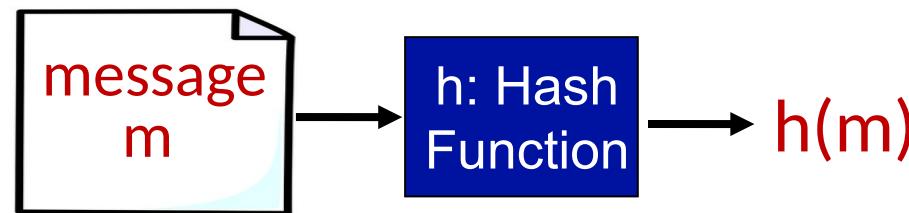
~~8.7 Network layer security: IPsec~~

~~8.8 Security in wireless and mobile networks~~

8.9 Operational security: firewalls and IDS

# Hash functions

- A hash function  $h(m)$  converts any input  $m$  into a **hash** (or **message digest**) of fixed length ☺ “fingerprint” of  $m$
- Secure one-way function



## Hash function properties:

- **Onewayness**: given message digest  $x$ , computationally infeasible to find  $m$  such that  $x = h(m)$
- Difficult to find two messages, so that  $h(m_1) = h(m_2)$

# Hash function algorithms

- **Message Digest (MD5)**
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string  $x$ , appears difficult to construct msg  $m$  whose MD5 hash is equal to  $x$
- **Secure Hash Algorithm (SHA-1)**
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
- **Secure Hash Algorithm (SHA-2)**
  - US standard [NIST, FIPS PUB 180-2]
  - SHA-256, SHA-384 and SHA-512 named after their digest length

# Message authentication

If Bob receives a message from Alice, how can Bob be sure that:

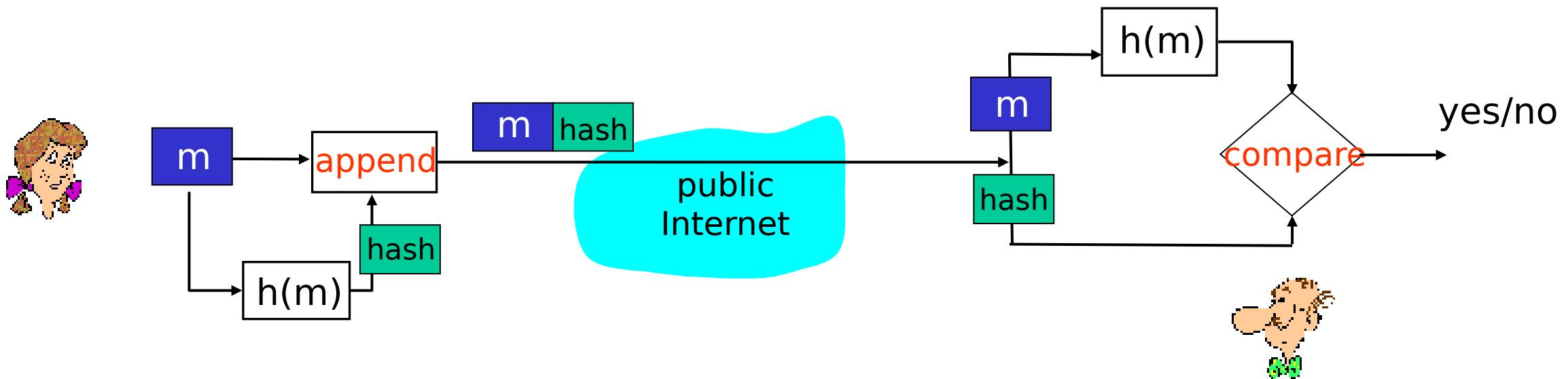
1. Alice sent it?              impersonation, spoofing attacks
2. the message has not been intentionally modified by an adversary?              integrity breech

**Message authentication** aka. **data origin authentication**:

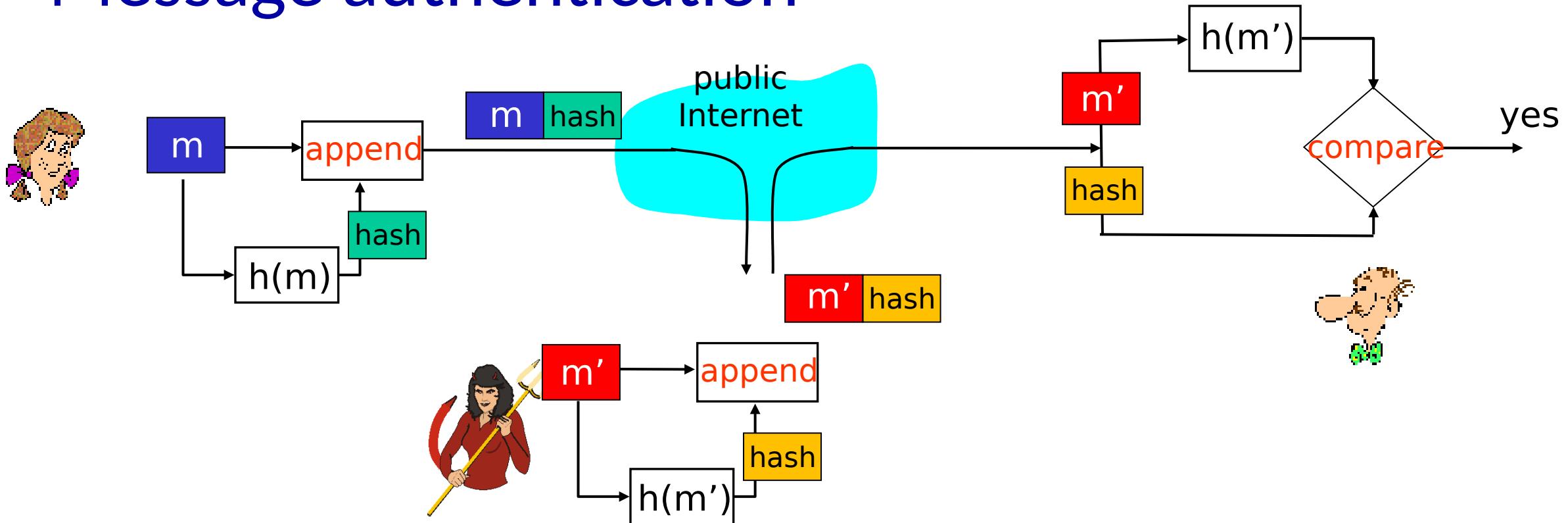
3. confirm that the message originated from the stated sender  
**(sender authenticity)**
4. confirm that the message has not been changed (integrity)

# Message authentication

What if Alice appends a hash?



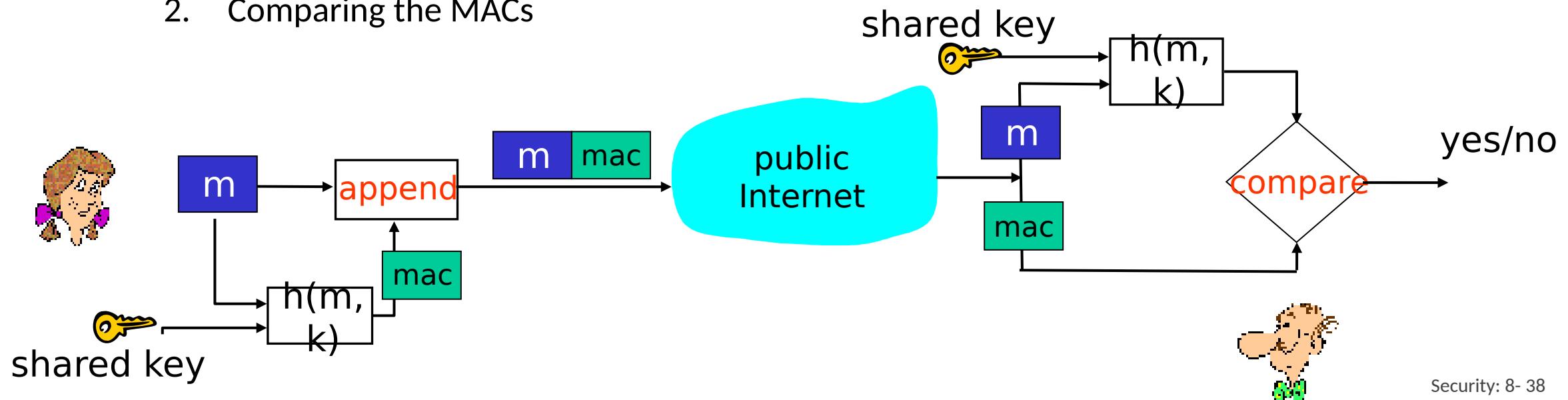
# Message authentication



- **Important:** Hashes do **not** protect against an adversary that
  1. modifies message contents
  2. pretends to be a somebody else

# Message authentication

- Message authentication codes (MAC) are a short piece of information used for authenticating and integrity-checking a message
- Alice and Bob share a secret key
  1. Alice computes a hash of the key and the message
  2. Bob verifies the message by:
    1. Computing a hash of his key and the received message
    2. Comparing the MACs

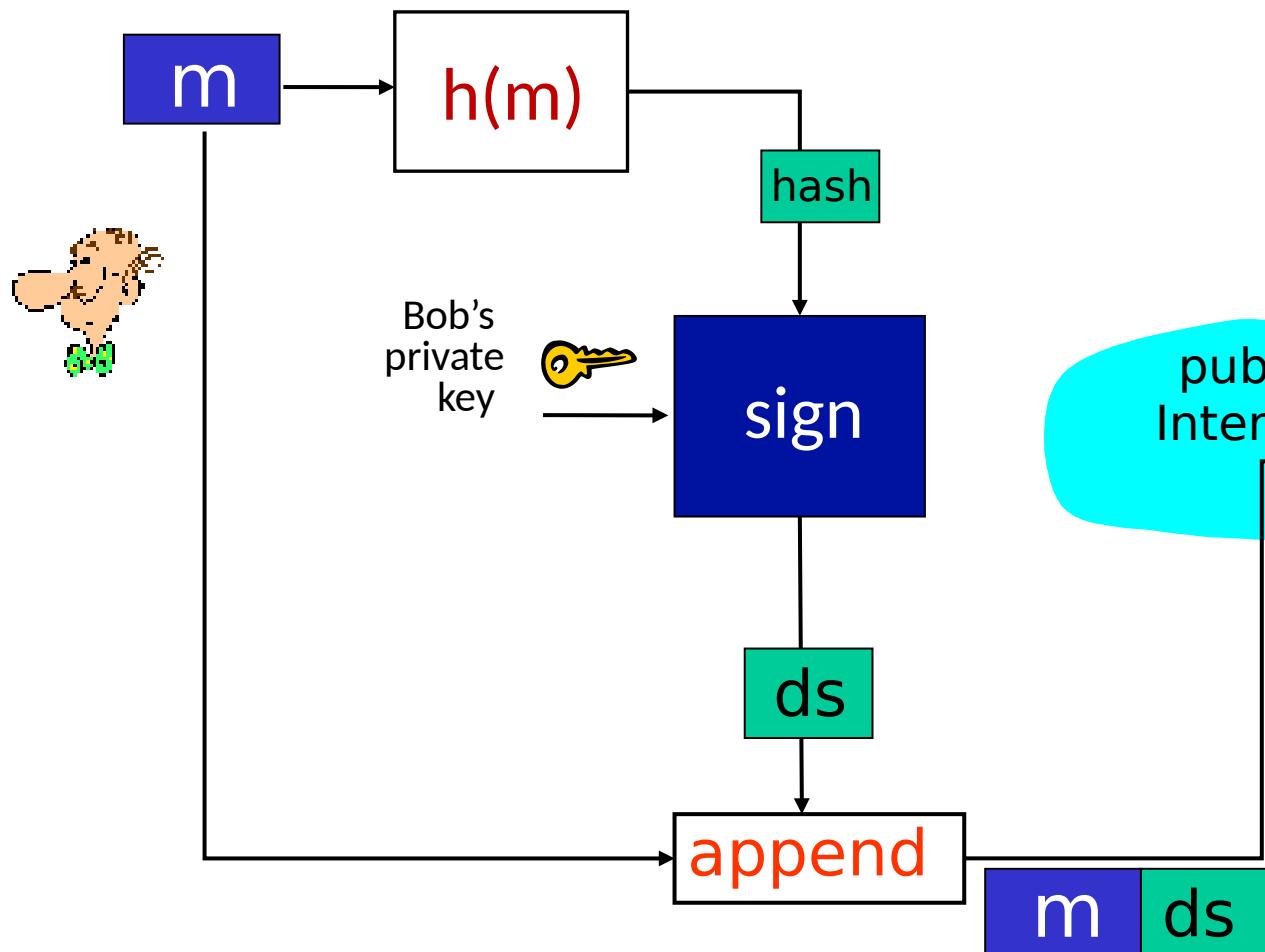


# Digital signatures

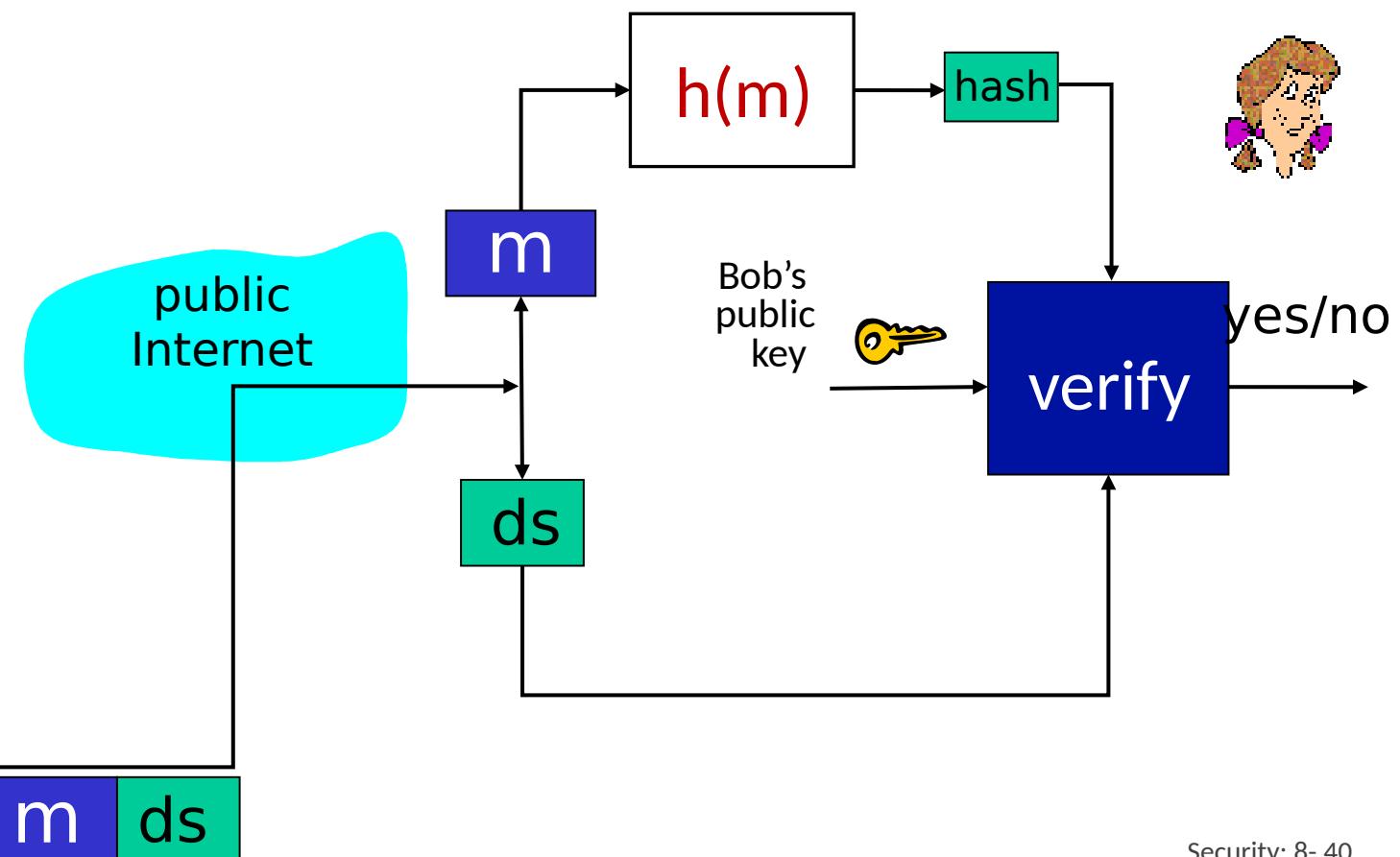
- Message authentication works for two parties sharing a secret key
- The receiver can prove the identity of the sender
- How can it be proven to **anybody** who
  1. is the sender of a message?
  2. is the originator of a file (e.g. document)?
- Digital signatures use **asymmetric cryptography**
- Security property: **Non-repudiation**
  - Not forgeable
    - In contrast, **handwritten signatures** are forgeable

# Digital signatures

Bob signs his message:

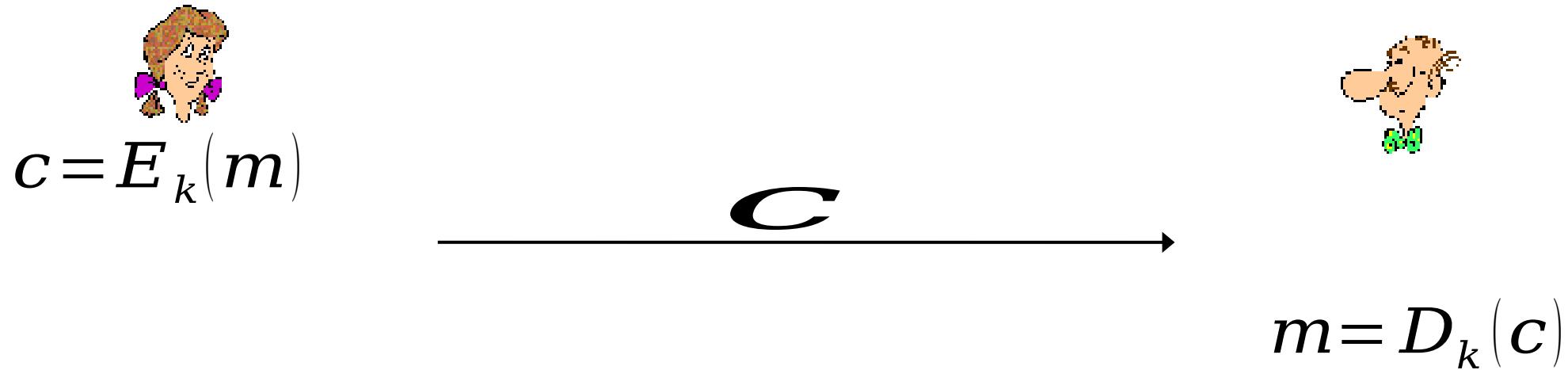


Alice verifies the signature:



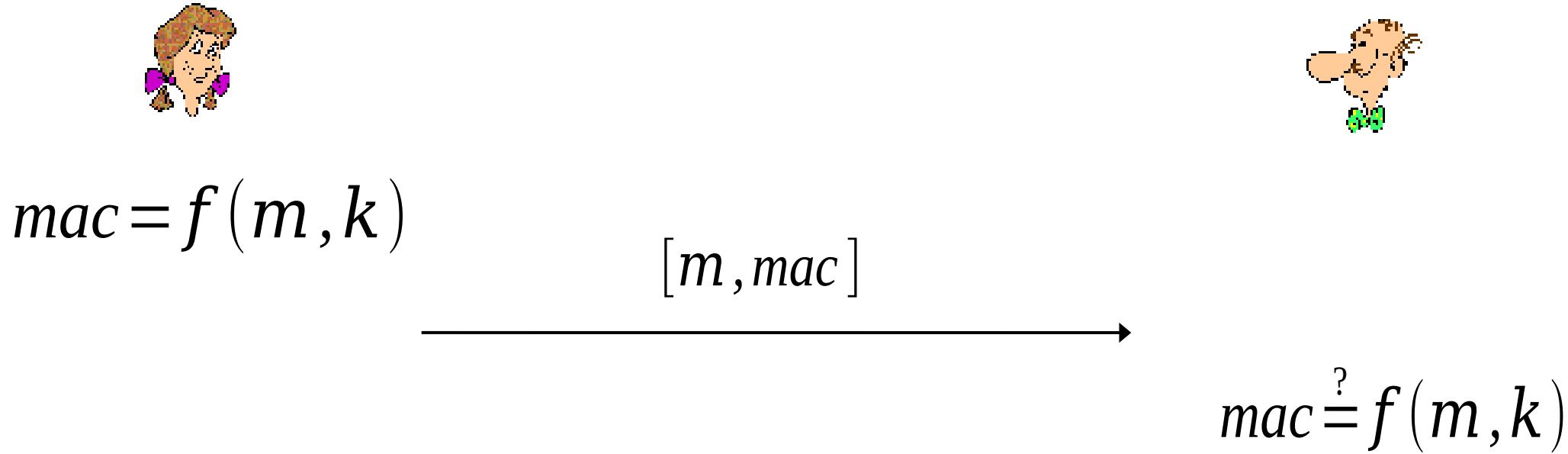
# Symmetric key cryptography

- Confidentiality protection



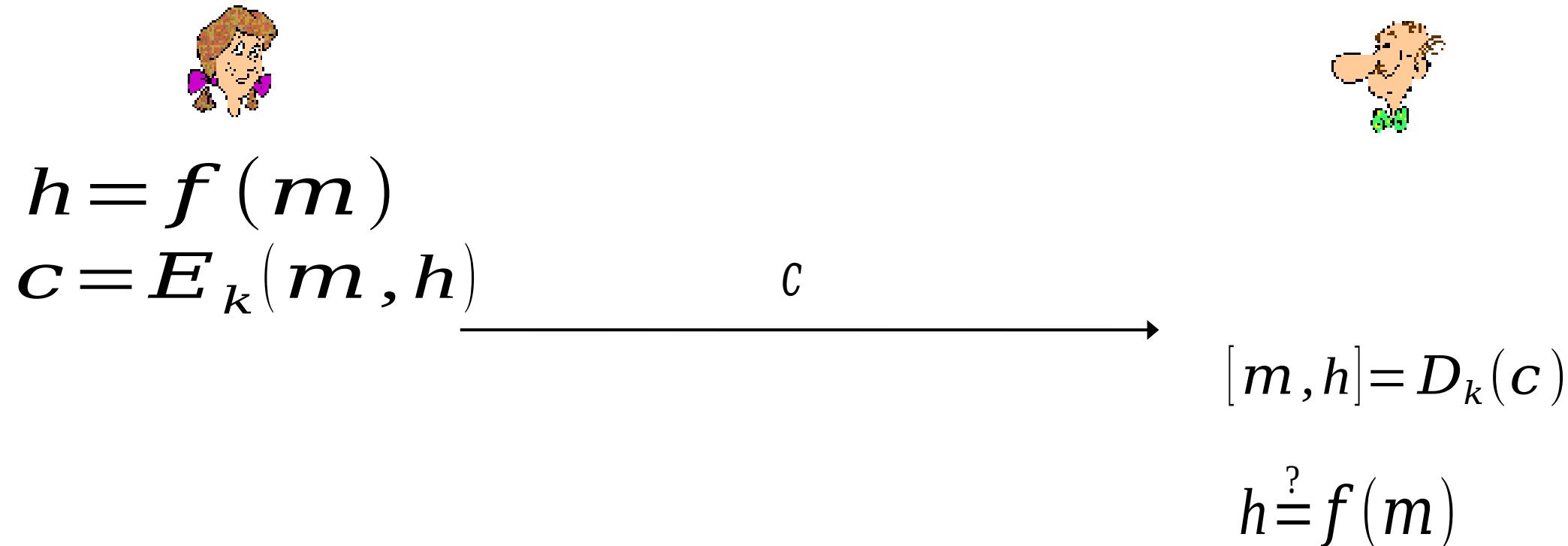
# Symmetric key cryptography

- Message authentication



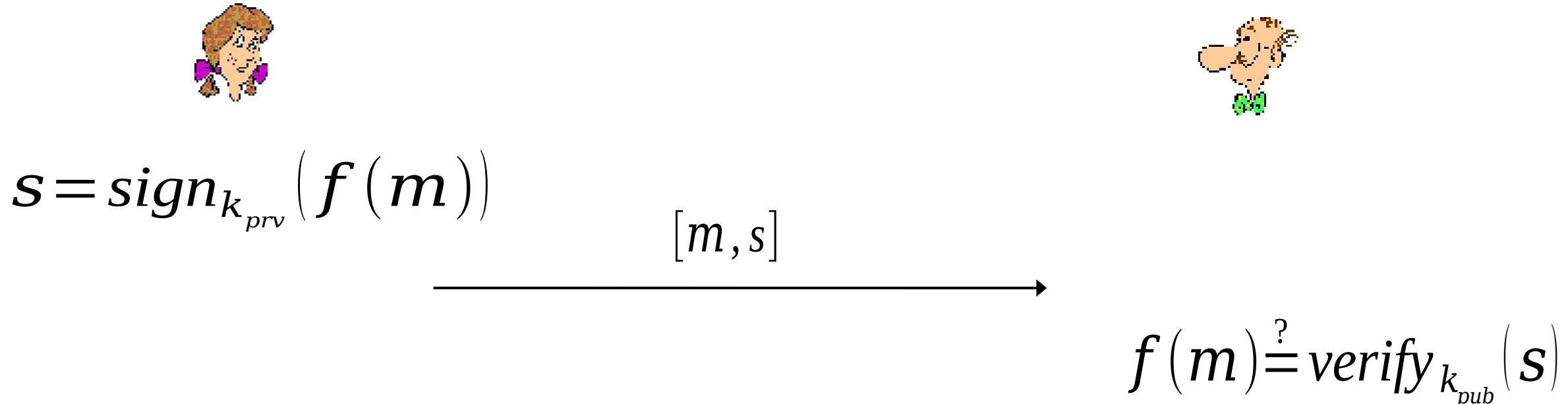
# Symmetric key cryptography

- Authenticated encryption:  
Confidentiality protection + message authentication



# Asymmetric key cryptography

- Digital signature



# Security properties and cryptographic primitives

|                              | Confidentiality | Data integrity | Message authentication | Non-repudiation |
|------------------------------|-----------------|----------------|------------------------|-----------------|
| Hash functions               | No              | No             | No                     | No              |
| Symmetric key encryption     | Yes             | No*            | No*                    | No              |
| Public key encryption        | Yes             | No             | No                     | No              |
| Message authentication codes | No              | Yes            | Yes                    | No              |
| Digital signatures           | No              | Yes            | Yes                    | Yes             |

\* The encryption contains an appended hash

# Replay attacks

- A passive adversary can **eavesdrop** and record
  - **encrypted messages** (or plaintext messages)
  - plaintext messages with MACs
  - signed messages
- Then later **replayed** (sent) to the same receiver
  - The messages appear authentic
- Thus, the adversary can successfully deceive the receiver
- Encryption, MACs and/or digital signatures alone do not prevent replay attacks

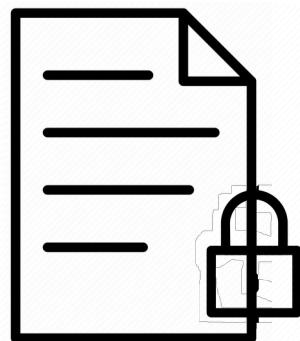
# Digital certificates

# Authenticity of public keys

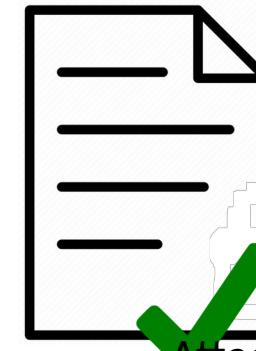
public key problem:

- when Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

# Threat scenario 1



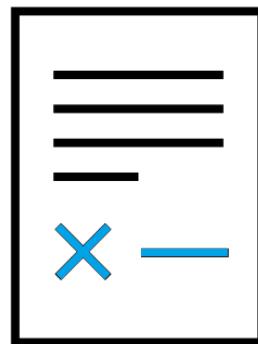
Encrypt



Attacker's  
Decry**pt** public key



# Threat scenario 2



Attacker's  
signature key



Sign

Attacker's  
verification key

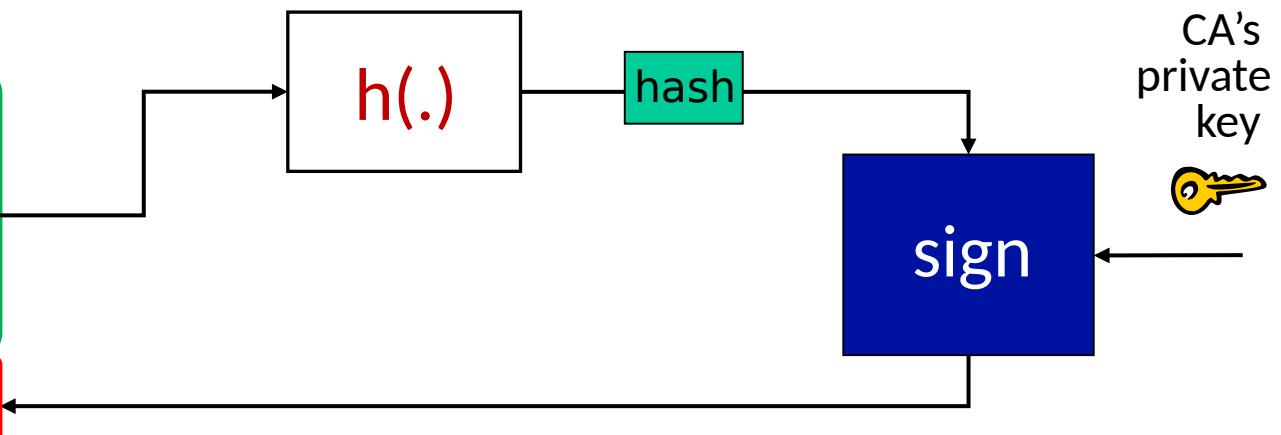


Signature  
verification



# Digital certificates

- A document to **prove the authenticity** of a public key/verification key
- Digital certificate is a binds a public key to the owner's name
- Assumes a **point of trust**: issued by a **certificate authority (CA)**
- The certificate contains
  1. Owner's name
  2. Owner's **public key**
  3. CA's name
  4. CA's signature
- To prove the validity of a public key, the certificate must be validated



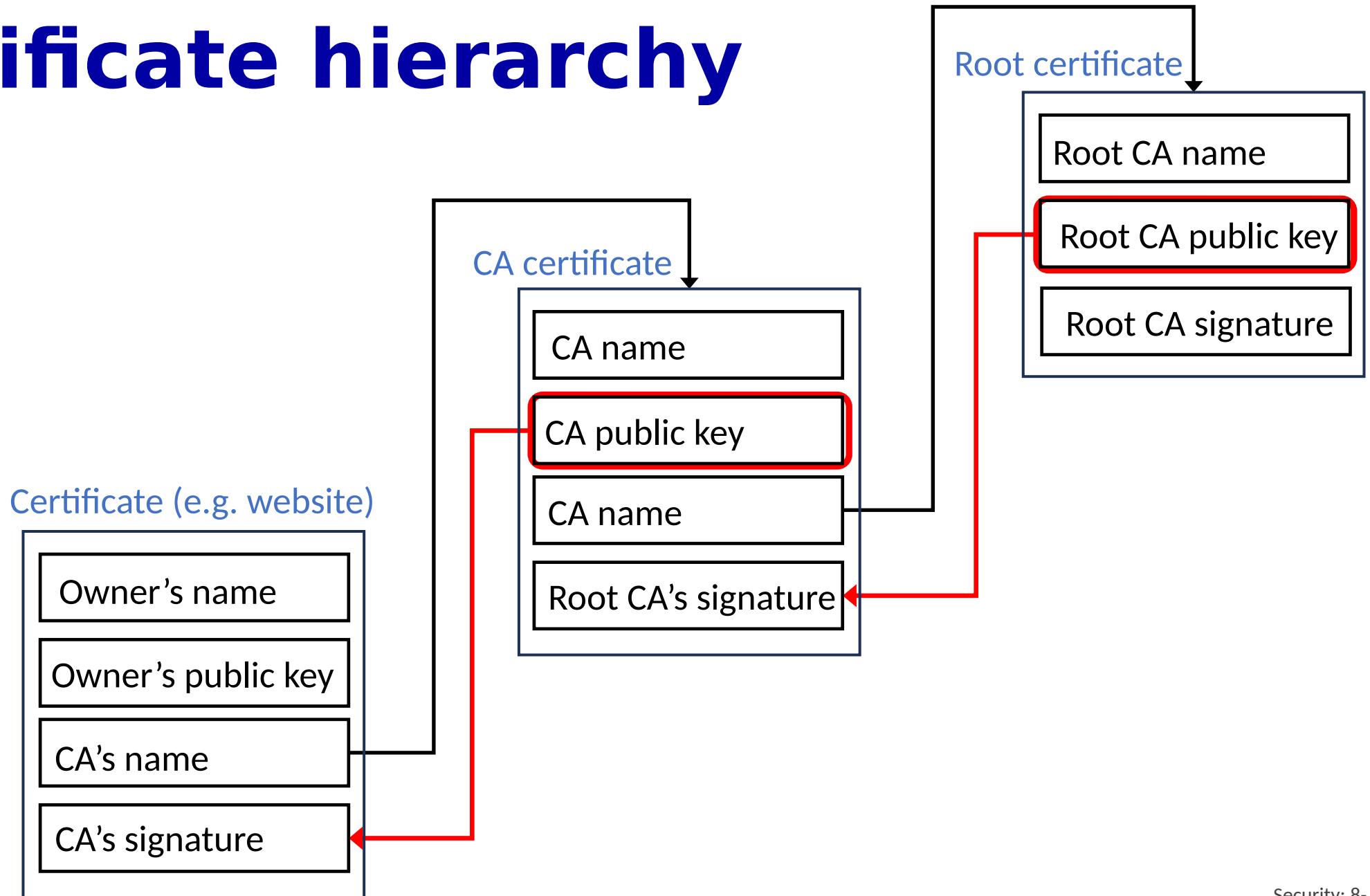
# X.509 v.1 certificate format

|                         |                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------|
| Version                 | Version number of the certificate.                                                     |
| Serial Number           | Unique number for each certificate issued by a certificate authority (CA).             |
| Signature               | The identifier for the cryptographic algorithm used by the CA to sign the certificate. |
| Issuer                  | The distinguished name (DN) of the certificate's issuing CA.                           |
| Validity                | The time period for which the certificate is valid.                                    |
| Subject                 | The distinguished name (DN) of the certificate owner.                                  |
| Subject Public Key Info | The public key owned by the certificate subject.                                       |

# What it is not

- A public key certificate is not a proof of identity
- A public-key certificate is a statement that the public key contained in it belongs to the named owner and has the properties specified in the certificate
- Once the certificate has been checked, the public key can be extracted from the certificate and then used for its specified purpose

# Certificate hierarchy



# **Browser certificates**

# Browser root certificates

- Web-browsers have prestored root certificates
- Mozilla Firefox
  - 47 root certificates, representing 52 organizations
  - Actual count gives 176 root certificates, representing 75 organizations

Certificate Manager

X

Your Certificates    Authentication Decisions    People    Servers    Authorities

You have certificates on file that identify these certificate authorities

| Certificate Name        | Security Device      |
|-------------------------|----------------------|
| BJCA Global Root CA1    | Builtin Object Token |
| ▼ Buypass AS-983163327  |                      |
| Buypass Class 2 Root CA | Builtin Object Token |
| Buypass Class 3 Root CA | Builtin Object Token |
| ▼ Certainly             |                      |
| Certainly Root E1       | Builtin Object Token |
|                         |                      |
|                         |                      |

View...    Edit Trust...    Import...    Export...    Delete or Distrust...

OK

# Browser root certificates

(The browser hides the signatures)

|                                |                                                                                   |
|--------------------------------|-----------------------------------------------------------------------------------|
| <b>Buypass Class 2 Root CA</b> |                                                                                   |
| <b>Subject Name</b>            |                                                                                   |
| Country                        | NO                                                                                |
| Organization                   | Buypass AS-983163327                                                              |
| Common Name                    | Buypass Class 2 Root CA                                                           |
| <hr/>                          |                                                                                   |
| <b>Issuer Name</b>             |                                                                                   |
| Country                        | NO                                                                                |
| Organization                   | Buypass AS-983163327                                                              |
| Common Name                    | Buypass Class 2 Root CA                                                           |
| <hr/>                          |                                                                                   |
| <b>Validity</b>                |                                                                                   |
| Not Before                     | Tue, 26 Oct 2010 08:38:03 GMT                                                     |
| Not After                      | Fri, 26 Oct 2040 08:38:03 GMT                                                     |
| <hr/>                          |                                                                                   |
| <b>Public Key Info</b>         |                                                                                   |
| Algorithm                      | RSA                                                                               |
| Key Size                       | 4096                                                                              |
| Exponent                       | 65537                                                                             |
| Modulus                        | D7:C7:5E:F7:C1:07:D4:77:FB:43:21:F4:F4:F5:69:E4:EE:32:01:DB:A3:86:1F:E4:59:0D:... |
| <hr/>                          |                                                                                   |
| <b>Miscellaneous</b>           |                                                                                   |
| Serial Number                  | 02                                                                                |
| Signature Algorithm            | SHA-256 with RSA Encryption                                                       |
| Version                        | 3                                                                                 |
| Download                       | <a href="#">PEM (cert)</a> <a href="#">PEM (chain)</a>                            |

# HTTPS

- HTTPS is indicated by padlock symbol
  - downloads the certificate chain of a visited website
  - Parts of certs. can be viewed



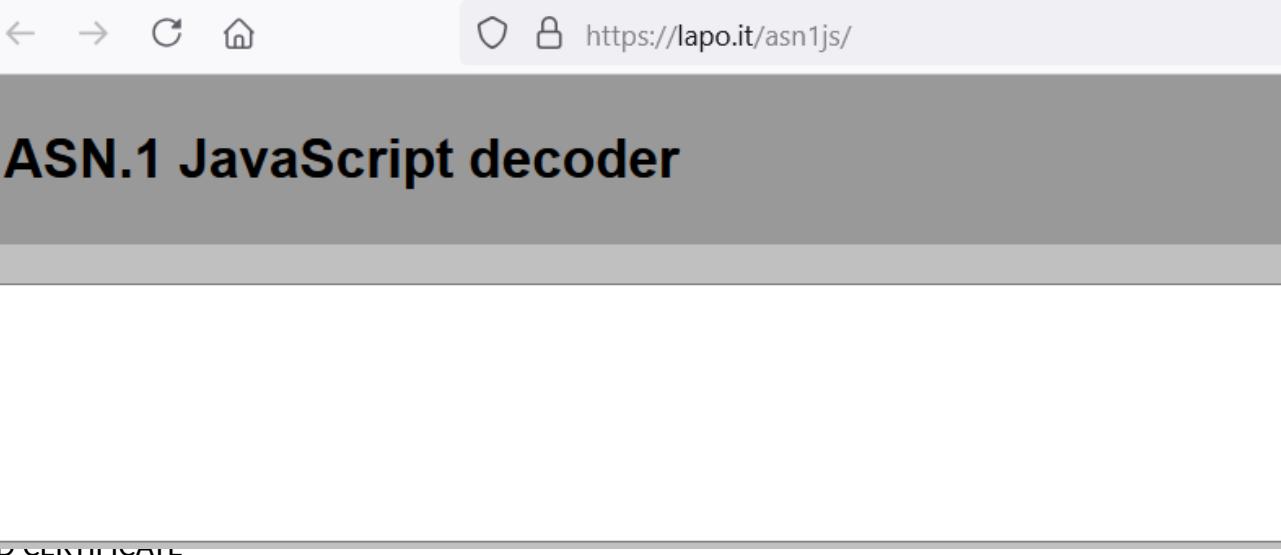
## Certificate

| vg.no                    | ZeroSSL RSA Domain Secure Site CA                                             | USERTrust RSA Certification Authority |
|--------------------------|-------------------------------------------------------------------------------|---------------------------------------|
| <b>Subject Name</b>      |                                                                               |                                       |
| Common Name              | vg.no                                                                         |                                       |
| <b>Issuer Name</b>       |                                                                               |                                       |
| Country                  | AT                                                                            |                                       |
| Organization             | ZeroSSL                                                                       |                                       |
| Common Name              | <a href="#">ZeroSSL RSA Domain Secure Site CA</a>                             |                                       |
| <b>Validity</b>          |                                                                               |                                       |
| Not Before               | Tue, 24 Sep 2024 00:00:00 GMT                                                 |                                       |
| Not After                | Mon, 23 Dec 2024 23:59:59 GMT                                                 |                                       |
| <b>Subject Alt Names</b> |                                                                               |                                       |
| DNS Name                 | vg.no                                                                         |                                       |
| DNS Name                 | *.vg.no                                                                       |                                       |
| <b>Public Key Info</b>   |                                                                               |                                       |
| Algorithm                | RSA                                                                           |                                       |
| Key Size                 | 2048                                                                          |                                       |
| Exponent                 | 65537                                                                         |                                       |
| Modulus                  | B2:23:C0:09:93:AF:84:EE:09:30:84:C9:29:3A:93:47:A9:35:B7:CF:52:A1:A1:76:34... |                                       |
| <b>Miscellaneous</b>     |                                                                               |                                       |
| Serial Number            | 4D:AF:05:D4:80:D5:32:73:4D:7A:19:F5:D0:35:6E:98                               |                                       |
| Signature Algorithm      | SHA-384 with RSA Encryption                                                   |                                       |
| Version                  | 3                                                                             |                                       |
| Download                 | <a href="#">PEM (cert)</a> <a href="#">PEM (chain)</a>                        |                                       |

# PEM-format

- The certs are stored in PEM-files
  - PEM: Privacy Enhanced Mail
- Can be viewed by a PEM-“decoders”
  - Some does not view signatures

-----BEGIN CERTIFICATE-----  
MIIFWTCCA0GgAwIBAgIBAjANBgkqhkiG9w0BAQsFADBOMQswCQYDVQQGEwJOTzEd  
MBsGA1UECgwUQnV5cGFzcyBBUy05ODMxNjMzMjcxIDAeBgNVBAMMF0J1eXBhc3Mg  
Q2xhc3MgMiBSb290IENBMB4XDTEwMTAyNjA4MzgwM1oXDTQwMTAyNjA4MzgwM1ow  
TjELMAkGA1UEBhMCTk8xHTAbBgNVBAoMFEJ1eXBhc3MgQVMtOTgzMTYzMzI3MSAw  
HgYDVQQDDBdCdXIwYXNzIENsYXNzIDigUm9vdCBDQTCAiIwDQYJKoZIhvcNAQEB  
BQADggIPADCCAgcggIBANfHXvfBB9R3+0Mh9PT1aeTuMgHbo4Yf5FkNuud1g1Lr  
6hxhFUi7HQfKjK6w3Jad6sNgkoakHOCvgb/S2TwDCo3SbXIzwx87vFKu3MwZfPV  
L4O2fuPn9Z6rYPnT8Z2SdlrkHJasW4DptfQxh6NR/Md+oW+OU3fUI8FVM5I+GC91  
1K2GScuVr1QGbNgGE41b/+EmGVnAJLqBcXmQRFB0JJRfuLMR8SIBYaNByyM21cHx  
MIAQTn/0hpPshNOOvEu/XAFOBz3cFlqUCqTqc/sLUegTBxj6DvEr0VQVfTzh97QZ  
QmdiXnfgolXsttlpF9U6r0TtSsWe5HonfOV116rLJeffawrbD02TTqigzXsu8IkB  
arcNuAeBfos4GzjmCleZPe4h6KP1DBbdi+w0jpwqHAADV41og9JwnxglzRFo1clr  
Us3ERo/ctfPYV3Me6ZQ5BL/T3jjetFPsaRyifsSP5BtwrfKi+fv3FmRmaZ9JUaLi  
FRhnBkp/1Wy1TbMz4GHRx7pmA8y1x1LPC5aAVKRCflf6o3YBkBjqhHk/sM3nhRS  
P/TizPJhk9H9Z2vXUq6/aKtAQ6BXNVN48FP4YUIHZMbXb5tMOA1jrGKvNouicwoN  
9SG9c...  
AgMB...  
uvdM...  
9bqw...  
A20uc...  
Oluwl...  
+fsicd...  
KcZ7X...  
DISCL...  
H8SIU...  
I+uUV...  
5t98b...  
3PFaT...  
Y11aV...  
-----END CERTIFICATE-----



The screenshot shows a browser window with a certificate decoded into ASN.1 JavaScript. The URL is https://lapo.it/asn1js/. The page title is "ASN.1 JavaScript decoder". The certificate content is displayed as a large block of JSON-like ASN.1 code.

# ASN.1 JavaScript decoder PEM viewers

```

Certificate SEQUENCE (3 elem)
└─ tbsCertificate TBSCertificate SEQUENCE (8 elem)
 └─ version [0] (1 elem)
 Version INTEGER 2
 └─ serialNumber CertificateSerialNumber INTEGER (127 bit) 103259325873972409032970270068247457432
 └─ signature AlgorithmIdentifier SEQUENCE (2 elem)
 └─ algorithm OBJECT IDENTIFIER 1.2.840.113549.1.1.12 sha384WithRSAEncryption (PKCS #1)
 └─ parameters ANY NULL
 └─ issuer Name SEQUENCE (3 elem)
 └─ RelativeDistinguishedName SET (1 elem)
 └─ RelativeDistinguishedName SET (1 elem)
 └─ AttributeTypeAndValue SEQUENCE (2 elem)
 └─ type AttributeType OBJECT IDENTIFIER 2.5.4.10 organizationName (X.520 DN component)
 └─ value AttributeValue [?] PrintableString ZeroSSL
 └─ RelativeDistinguishedName SET (1 elem)
 └─ AttributeTypeAndValue SEQUENCE (2 elem)
 └─ type AttributeType OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
 └─ value AttributeValue [?] PrintableString ZeroSSL RSA Domain Secure Site CA
 └─ validity Validity SEQUENCE (2 elem)
 └─ notBefore Time UTCTime 2024-09-24 00:00:00 UTC
 └─ notAfter Time UTCTime 2024-12-23 23:59:59 UTC
 └─ subject Name SEQUENCE (1 elem)
 └─ RelativeDistinguishedName SET (1 elem)
 └─ AttributeTypeAndValue SEQUENCE (2 elem)
 └─ type AttributeType OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
 └─ value AttributeValue [?] PrintableString vg.no
 └─ subjectPublicKeyInfo SubjectPublicKeyInfo SEQUENCE (2 elem)
 └─ algorithm AlgorithmIdentifier SEQUENCE (2 elem)
 └─ algorithm OBJECT IDENTIFIER 1.2.840.113549.1.1.1 rsaEncryption (PKCS #1)
 └─ parameters ANY NULL
 └─ subjectPublicKey BIT STRING (2160 bit) 001100001000001000000001000010100000001010000001000000001000000010000...
 └─ SEQUENCE (2 elem)
 └─ INTEGER (2048 bit) 224880473380598945989851281155271575189853335392743311739870072286391...
 └─ INTEGER 65537

```

|     |          |     |       |     |    |    |    |    |   |
|-----|----------|-----|-------|-----|----|----|----|----|---|
| 30  | 82       | 06  | 61    | 30  | 82 | 04 | 49 | A0 | 0 |
| AF  | 05       | D4  | 80    | D5  | 32 | 73 | 4D | 7A | 1 |
| 0D  | 06       | 09  | 2A    | 86  | 48 | 86 | F7 | 0D | 0 |
| 31  | 0B       | 30  | 09    | 06  | 03 | 55 | 04 | 06 | 1 |
| 0E  | 06       | 03  | 55    | 04  | 0A | 13 | 07 | 5A | 6 |
| 2A  | 30       | 28  | 06    | 03  | 55 | 04 | 03 | 13 | 2 |
| 4C  | 20       | 52  | 53    | 41  | 20 | 44 | 6F | 6D | 6 |
| 75  | 72       | 65  | 20    | 53  | 69 | 74 | 65 | 20 | 4 |
| 34  | 30       | 39  | 32    | 34  | 30 | 30 | 30 | 30 | 3 |
| 31  | 32       | 32  | 33    | 32  | 33 | 35 | 39 | 35 | 3 |
| 0C  | 06       | 03  | 55    | 04  | 03 | 13 | 05 | 76 | 6 |
| 22  | 30       | 0D  | 06    | 09  | 2A | 86 | 48 | 86 | F |
| 03  | 82       | 01  | 0F    | 00  | 30 | 82 | 01 | 0A | 0 |
| C0  | 09       | 93  | AF    | 84  | EE | 09 | 30 | 84 | C |
| B7  | CF       | 52  | A1    | A1  | 76 | 34 | 2A | 9C | 4 |
| A7  | 28       | E8  | 7A    | 93  | F8 | 6D | C8 | FF | F |
| BF  | 4C       | 13  | 9D    | 4A  | 74 | 43 | 9C | 54 | 4 |
| ... | skipping | 160 | bytes | ... |    |    |    |    |   |
| 98  | 32       | 19  | BA    | 5A  | 06 | 38 | 50 | D5 | F |
| 1A  | 77       | 7B  | C6    | C7  | 86 | 7A | 61 | 0F | 7 |
| 01  | 00       | 01  | A3    | 82  | 02 | 7A | 30 | 82 | 0 |
| 1D  | 23       | 04  | 18    | 30  | 16 | 80 | 14 | C8 | D |
| D5  | 3D       | 72  | DE    | 5F  | 0A | 3E | DC | B5 | 8 |
| 55  | 1D       | 0E  | 04    | 16  | 04 | 14 | 88 | 81 | 4 |
| FC  | 59       | C9  | C1    | 40  | 1E | 4B | AA | 18 | 7 |
| 1D  | 0F       | 01  | 01    | FF  | 04 | 04 | 03 | 02 | 0 |
| 1D  | 13       | 01  | 01    | FF  | 04 | 02 | 30 | 00 | 3 |
| 04  | 16       | 30  | 14    | 06  | 08 | 2B | 06 | 01 | 0 |
| 2B  | 06       | 01  | 05    | 05  | 07 | 03 | 02 | 30 | 4 |
| 42  | 30       | 40  | 30    | 34  | 06 | 0B | 2B | 06 | 0 |
| 02  | 4E       | 30  | 25    | 30  | 23 | 06 | 08 | 2B | 0 |
| 16  | 17       | 68  | 74    | 74  | 70 | 73 | 3A | 2F | 2 |
| 6F  | 2E       | 63  | 6F    | 6D  | 2F | 43 | 50 | 53 | 3 |
| 01  | 02       | 01  | 30    | 81  | 88 | 06 | 08 | 2B | 0 |

c

```

Extension [1] (31 byte) https://www.googleapis.com
 Extension SEQUENCE (2 elem)
 extnID OBJECT IDENTIFIER 1.3.6.1.4.1.11129.2.4.2 googleSignedCertificateTimestamp (Google Certificate Tra...
 extnValue OCTET STRING (245 byte) 0481F200F000760076FF883F0AB6FB9551C261CCF587BA34B4A4CDBB29DC68420A9FE...
 OCTET STRING (242 byte) 00F000760076FF883F0AB6FB9551C261CCF587BA34B4A4CDBB29DC68420A9FE6674C5...
Extension SEQUENCE (2 elem)
 extnID OBJECT IDENTIFIER 2.5.29.17 subjectAltName (X.509 extension)
 extnValue OCTET STRING (18 byte) 3010820576672E6E6F82072A2E76672E6E6F
 SEQUENCE (2 elem)
 [2] (5 byte) vg.no
 [2] (7 byte) *.vg.no
signatureAlgorithm AlgorithmIdentifier SEQUENCE (2 elem)
 algorithm OBJECT IDENTIFIER 1.2.840.113549.1.1.12 sha384WithRSAEncryption (PKCS #1)
 parameters ANY NULL
signature BIT STRING (4096 bit) 0000010001010100110100010101100110000000110101010001011111000011011...
Offset: 1120
Length: 4+513
INi
Value:
.Gx
(4096 bit)
.yY
000001000101010011010001010110011000000011010101000101111100001101110000100000
.i4h
1111100100011010101110011101001110010011011011001001011111001101110100100110010
.sTi
00110100100111011111000111111101111001010100001001011001010000111001001110011001
.W4:
11001001111101111110100000110100101011010001001100101101100111110001001011010
.kNl
110010111111011101001110011001011001110101100111110000101010001011000010101
.nQ
1000000101011100010010100000001100101001010000100111010111000110110000101011
1110001000000100101000111011110110101010111000011110001001110001101101000010
with
1011100110100111001100010110100110111110110010001001001110101000011011000101011

```

# **Chapter 8 outline**

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and digital signatures

## **8.4 Authentication**

8.5 Securing e-mail

8.6 Securing TCP connections: TLS

~~8.7 Network layer security: IPsec~~

~~8.8 Security in wireless and mobile networks~~

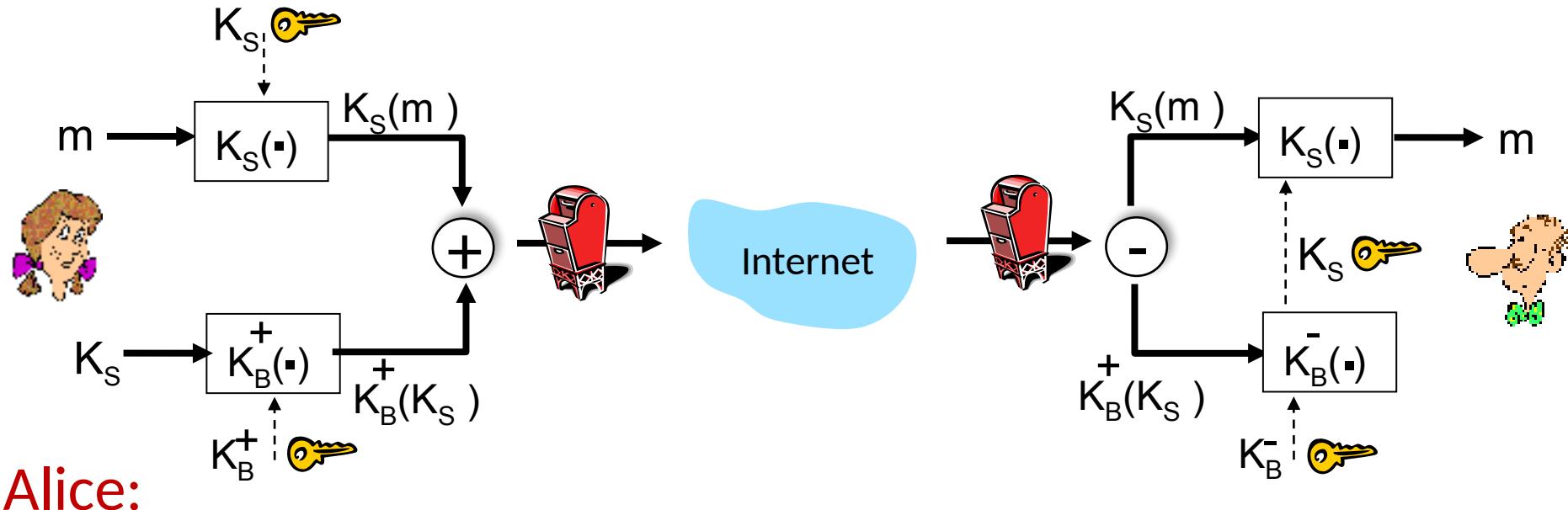
8.9 Operational security: firewalls and IDS

# **Chapter 8 outline**

- 8.1 What is network security?
- 8.2 Principles of cryptography
- 8.3 Message integrity and digital signatures
- 8.4 Authentication
- 8.5 Securing e-mail**
- 8.6 Securing TCP connections: TLS
- ~~8.7 Network layer security: IPsec~~
- ~~8.8 Security in wireless and mobile networks~~
- 8.9 Operational security: firewalls and IDS

# Secure e-mail: confidentiality

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

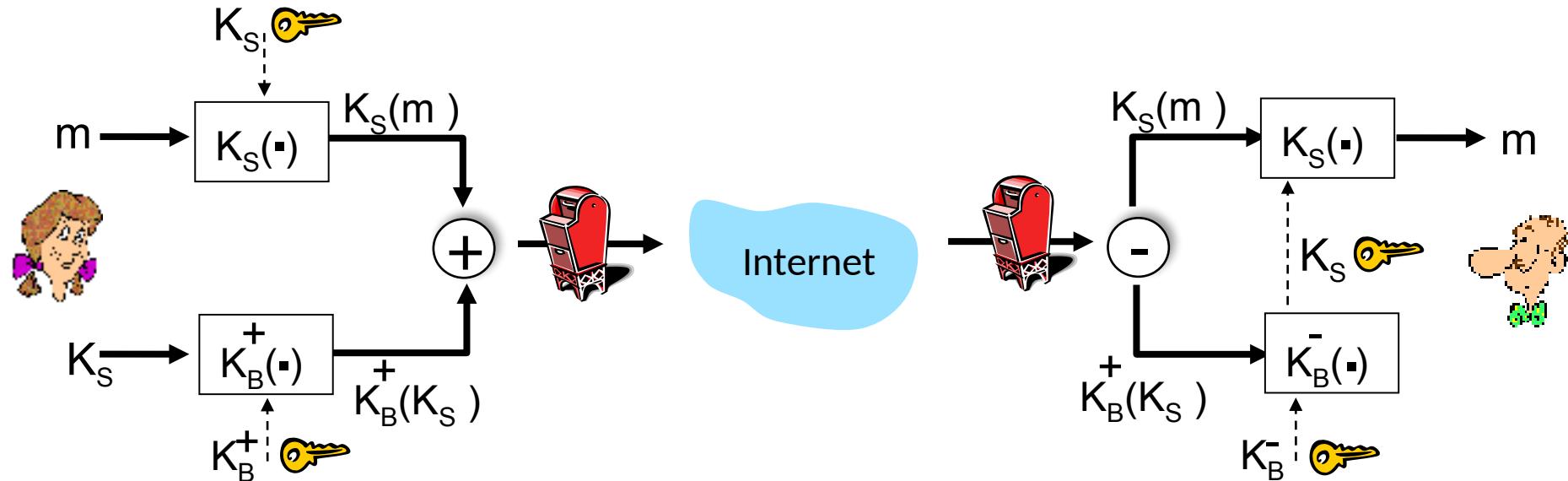


Alice:

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

# Secure e-mail: confidentiality (more)

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

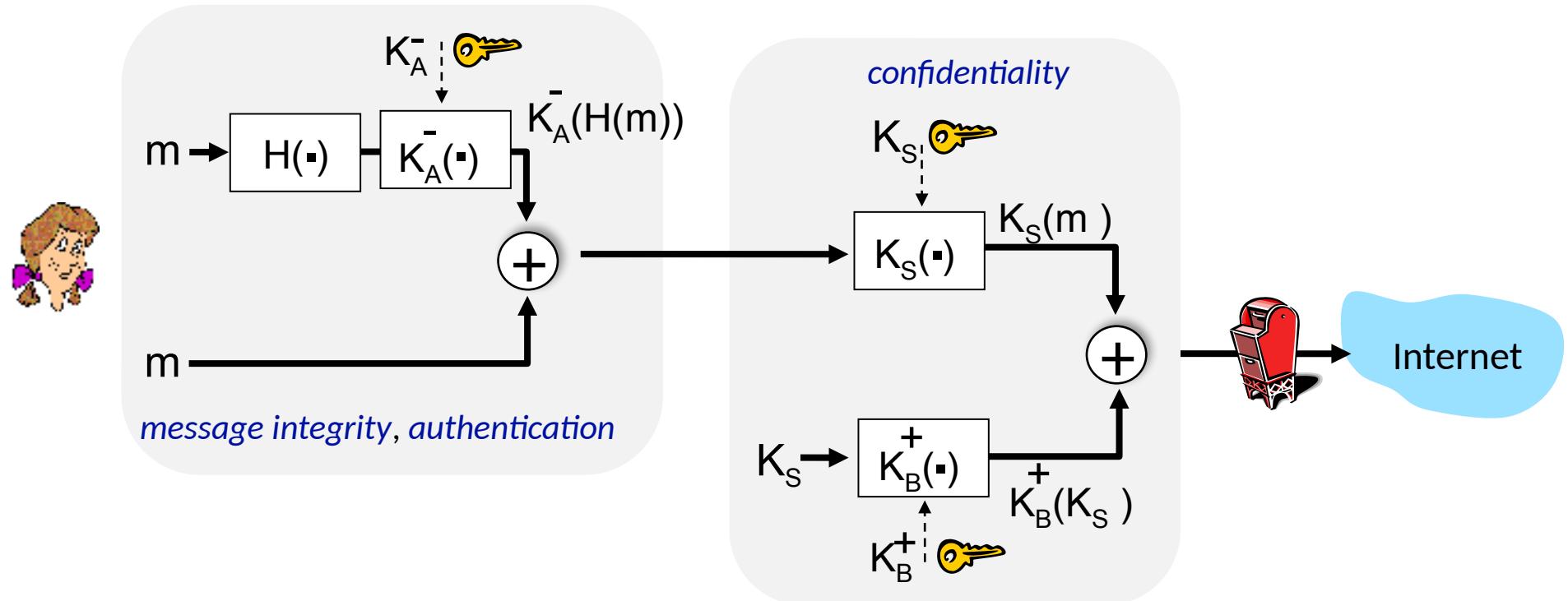


**Bob:**

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

# Secure e-mail: integrity, authentication

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



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

*What are Bob's complementary actions?*

# Pretty Good Privacy (PGP)

- internet e-mail encryption scheme, de-facto standard
- uses symmetric key cryptography, public key cryptography, hash function, and digital signature as described
- provides secrecy, sender authentication, integrity
- inventor, Phil Zimmerman, was target of 3-year federal investigation

## A PGP signed message:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Bob: Can I see you tonight?
Passionately yours, Alice

-----BEGIN PGP SIGNATURE-----
Version: PGP 5.0
Charset: noconv
yHJRHHgJGhgg/
12EpJ+lo8gE4vB3mqhFEvZP9t6n7G6m5Gw
2
-----END PGP SIGNATURE-----
```

## A secret PGP message:

```
-----BEGIN PGP MESSAGE-----
Version: PGP 5.0
u2R4d+/
jKmn8Bc5+hgDsqAewsDfrGdszX681iKm5F
6Gc4sDfcXytRfdS10juHgbcfDssWe7/
K=1KhnMikLo0+1/
BvcX4t==Ujk9Pbcd4Thdf2awQfgHbnmKlo
k8iy6gThlp
-----END PGP MESSAGE-----
```

# **Chapter 8 outline**

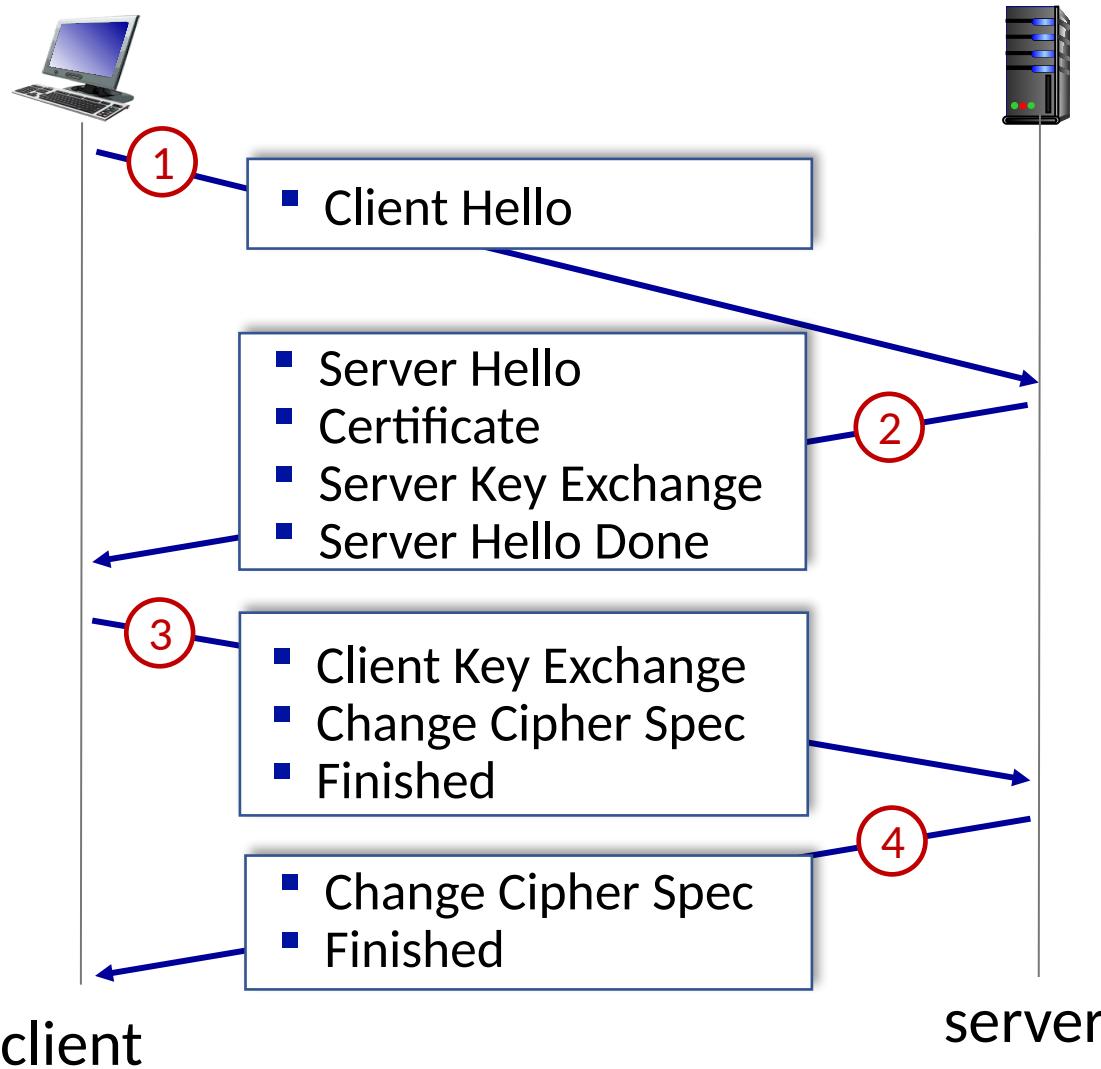
- 8.1 What is network security?
- 8.2 Principles of cryptography
- 8.3 Message integrity and digital signatures
- 8.4 Authentication
- 8.5 Securing e-mail
- 8.6 Securing TCP connections: TLS**
- ~~8.7 Network layer security: IPsec~~
- ~~8.8 Security in wireless and mobile networks~~
- 8.9 Operational security: firewalls and IDS

# Transport-layer security (TLS)

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

} *all techniques we have studied!*

# TLS 1.2 handshake – DH key exchange: 2 RTT



- ① Client Hello: List of supported cipher suites and nonce
- ② Server Hello: Selected cipher suite and nonce
  - Certificate: RSA or ECDSA
  - Server Key Exchange: signed DHE or ECDHE parameters and key share
  - Server Hello Done: Done
- ③ Client Key Exchange: DHE or ECDHE key share
  - Change Cipher Spec: Have generated session keys and switch to encrypted communication
  - Finished: MAC of sent/received handshake messages
- ④ Change Cipher Spec: Have generated session keys and switch to encrypted communication
  - Finished: MAC of sent/received handshake messages

# Diffie-Hellman key exchange Ephemeral (DHE)

1. Alice and Bob publicly agree to use a modulus  $p = 23$  and base  $g = 5$  (which is a primitive root modulo 23).
2. Alice chooses a secret integer  $a = 4$ , then sends Bob  $A = g^a \text{ mod } p$ 
  - $A = 5^4 \text{ mod } 23 = 4$
3. Bob chooses a secret integer  $b = 3$ , then sends Alice  $B = g^b \text{ mod } p$ 
  - $B = 5^3 \text{ mod } 23 = 10$
4. Alice computes  $s = B^a \text{ mod } p$ 
  - $s = 10^4 \text{ mod } 23 = 18$
5. Bob computes  $s = A^b \text{ mod } p$ 
  - $s = 4^3 \text{ mod } 23 = 18$
6. Alice and Bob now share a secret (the number 18). [4]

Both Alice and Bob have arrived at the same values because under mod p,

$$A^b \text{ mod } p = g^{ab} \text{ mod } p = g^{ba} \text{ mod } p = B^a \text{ mod } p$$

# Diffie-Hellman key exchange Ephemeral (DHE)

- Prime **p** is huge (e.g., 6144-bit MODP Group (Group 17) from RFC 3526):

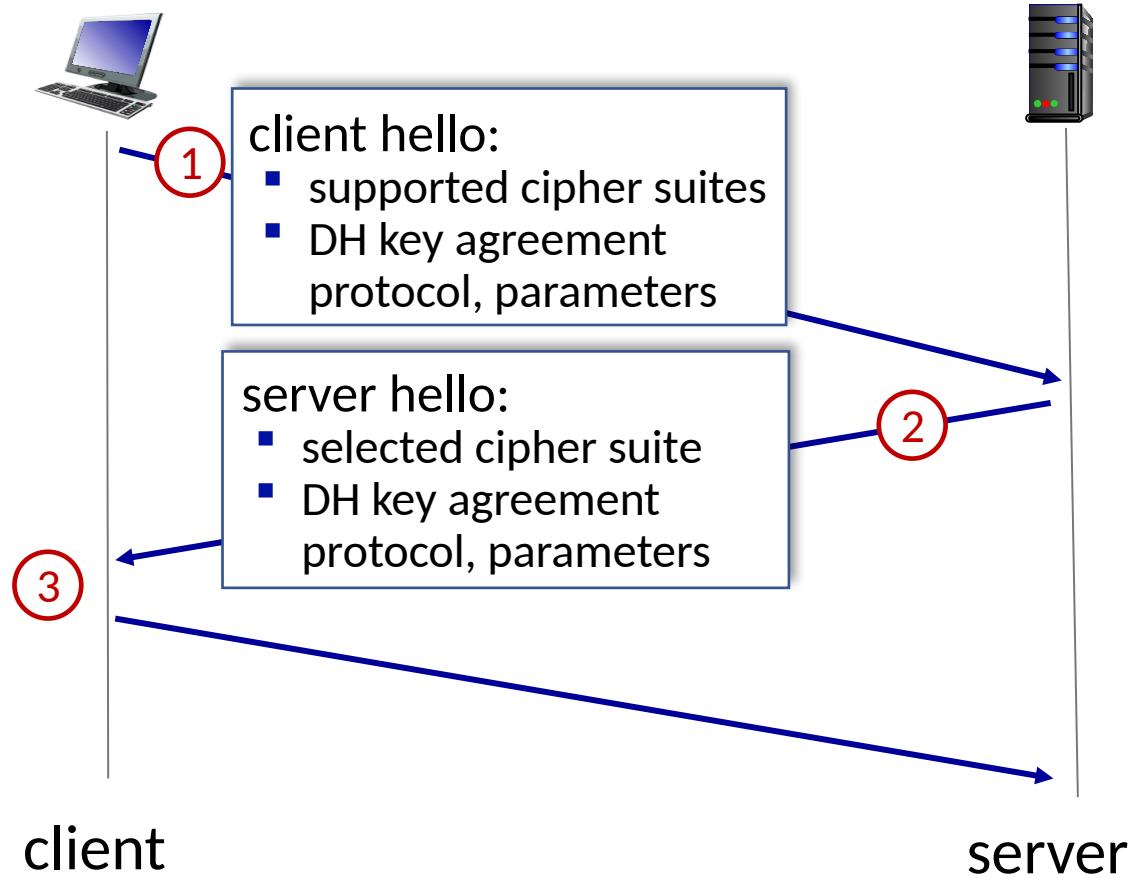
337515218214385611845185231599674123300648978057418465481738904744294299013266724452032351019191654  
839641943594609948810620893878937628140442574382044325739410830148270060902589258751610180963277323  
358005958319159760142088223040073278481327349332978858032136752615649626033404572207768263225000580  
913109672539766199739880336636663851881552126562680795017262233696934279998041344678101207723564985  
969455323665274005175754719693358549052745041195095923660137119541482588848792245999152034563158810  
347765530836769957183355985863955911699995708245150350175435333526975252877533325005271765695768949  
267349504692935961340950866037168600863020515445396526890912990997845889190523834630577894405654606  
814419024423999564190605216296046973478790246543138001860783165269645292880627408790110351759200591  
921785614731990062058967194350147653455184908823666071109053034491525562211632321274264406919211346  
487666356958502392313045917442156109850296368954067188807663082492273159842675422662594896843722239  
164454110159005062394192679097163203312089889781808689874316237103476179923562014490238922032301330  
09421463914291201346063125219636964261683591541014344239275340735690997732220697587739633908763605  
465157552805170421605254873028981223116697996794475304536003993426970327144585495912859394539490349  
812481143223223672386450425159844478907889178235763300191516965686543141530585475920913660145501438  
196851700683437001046776090411663697600809334136054989623820777788455998349074759534307874462013845  
673285306752757929623548837708069008271836857183534695747316805206219445409477346190351771800579730  
226525710321965982292591948757099947097217931541586865157485072742241813169487971046010682120152329  
216914824963468544136987197501906011027052744810505432398151306860736010763045122845492184598460460  
82253596762433827419060089029417044871218316020923109988915707117567

- Generator **g** is generally small (for OpenSSL always 2)
- The secret choices **s** are usually huge (256-bits and above)

# TLS: 1.3 cipher suite

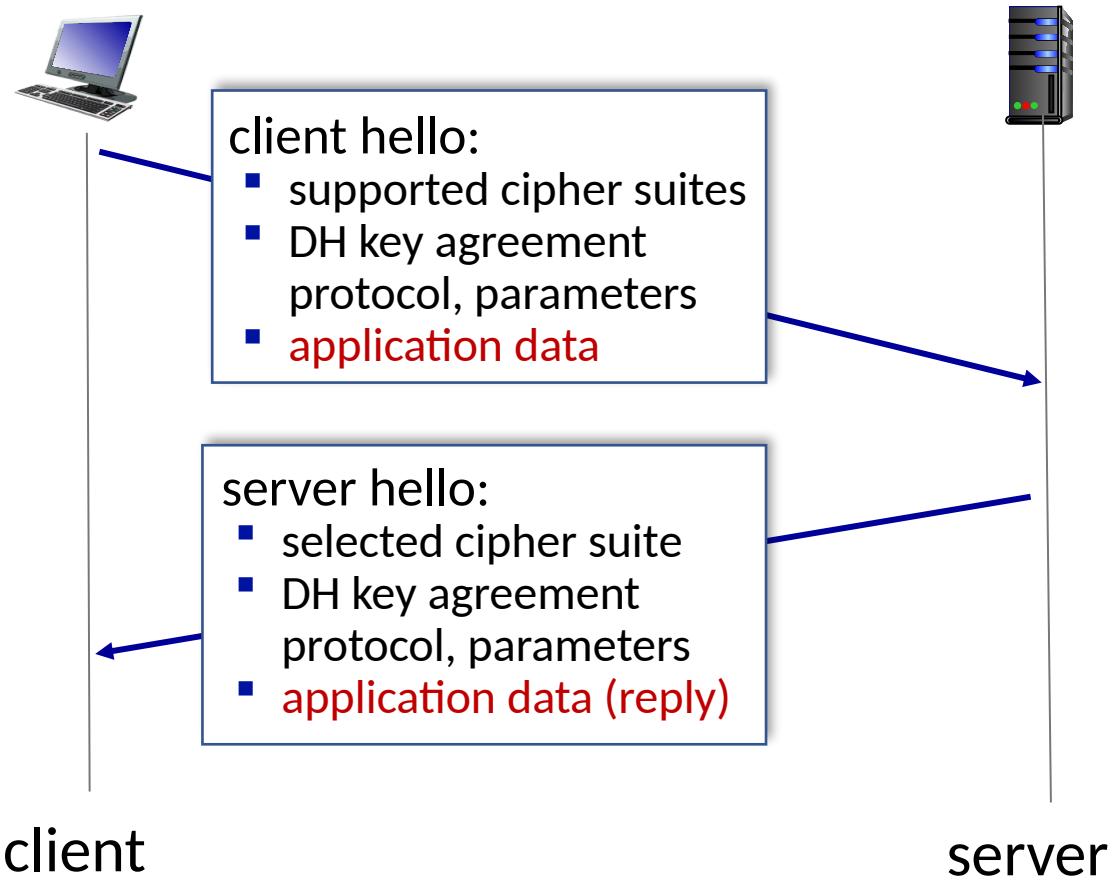
- “cipher suite”: algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
  - only 5 choices, rather than 37 choices
  - *requires* Diffie-Hellman (DHE and ECDHE) for key exchange, rather than DH and RSA
  - combined encryption and authentication algorithm (“authenticated encryption”) for data rather than serial encryption, authentication
    - 4 based on AES
  - HMAC uses SHA (256 or 384) cryptographic hash function

# TLS 1.3 handshake: 1 RTT



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

# TLS 1.3 handshake: 0 RTT



- initial hello message contains encrypted application data!
  - “resuming” earlier connection between client and server
  - application data encrypted using “resumption master secret” from earlier connection
- vulnerable to replay attacks!
  - maybe OK for get HTTP GET or client requests not modifying server state

# **Chapter 8 outline**

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and digital signatures

8.4 Authentication

8.5 Securing e-mail

8.6 Securing TCP connections: TLS

8.7 ~~Network layer security: IPsec~~

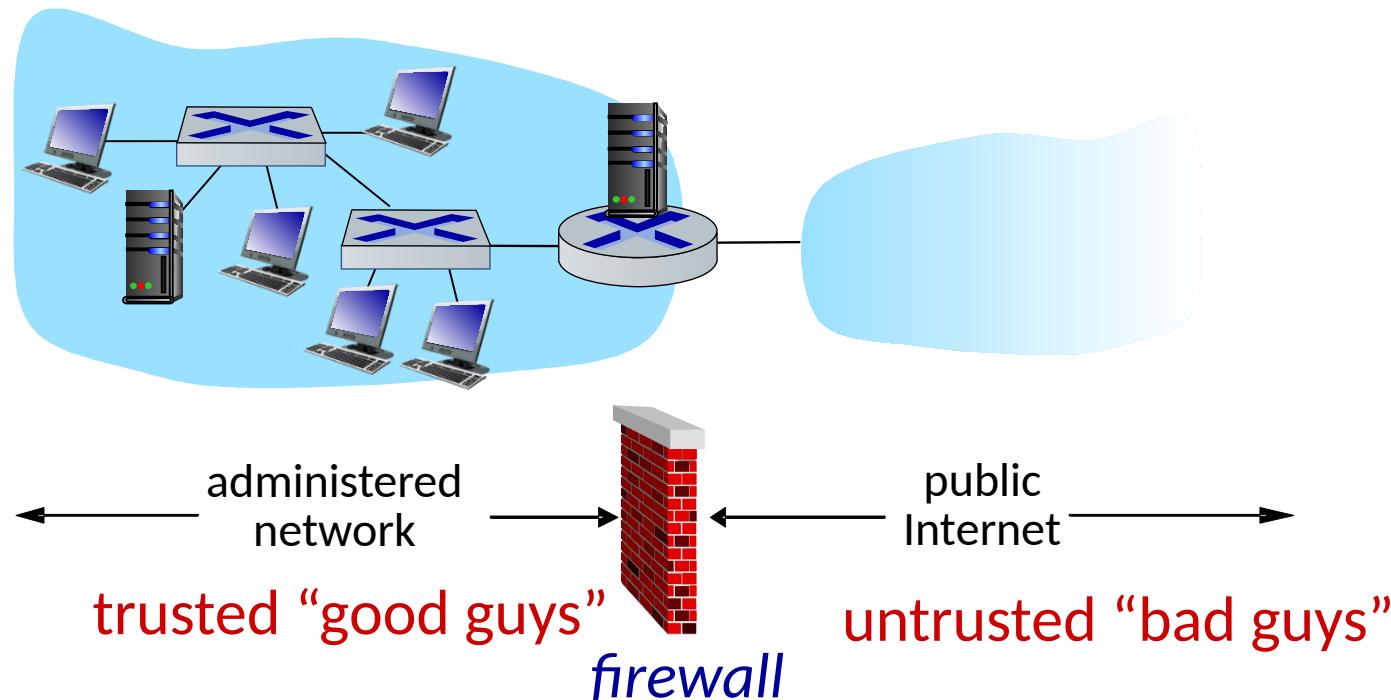
8.8 ~~Security in wireless and mobile networks~~

**8.9 Operational security: firewalls and IDS**

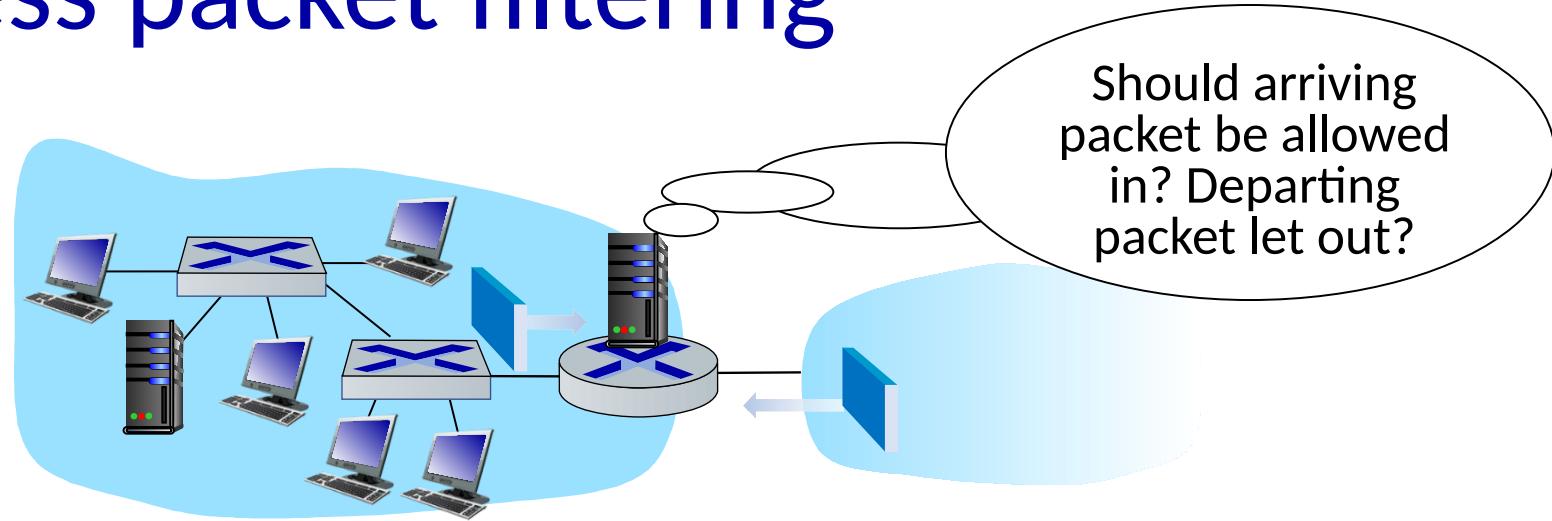
# Firewalls

firewall

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



# Stateless packet filtering



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

# Access Control Lists

**ACL:** table of rules, applied top to bottom to incoming packets: (action, condition) pairs looks like OpenFlow forwarding (Ch. 4)!

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

# Stateful packet filtering

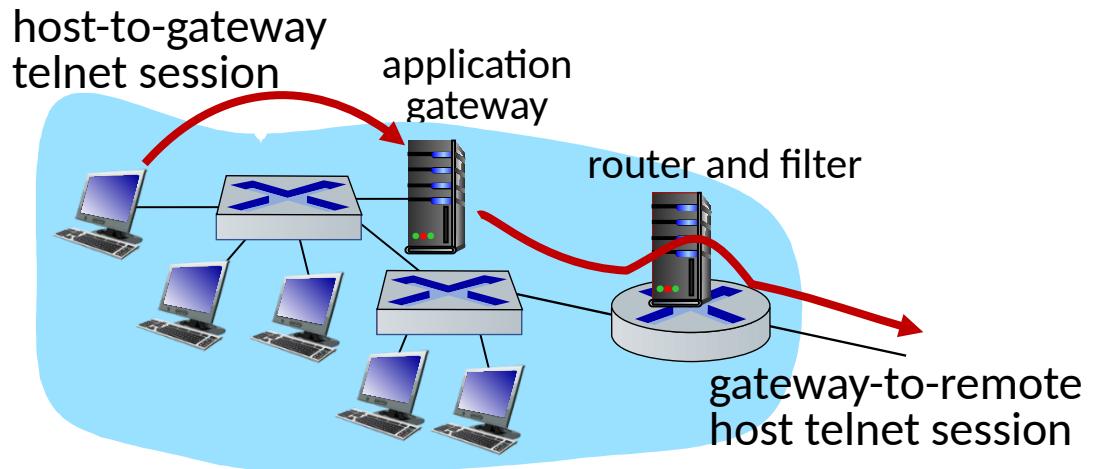
- *stateless packet filter*: heavy handed tool
  - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

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

- *stateful packet filter*: track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets

# Application gateways

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



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

# Intrusion detection systems

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

# Network Security (summary)

basic techniques.....

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

.... used in many different security scenarios

- secure email
- secure transport (TLS)
- ~~IPsec~~
- ~~802.11, 4G/5G~~

operational security: firewalls and IDS



# TCP socket programming

Sigurd Eskeland

# Socket programming with TCP

## Server

- server process must first be running
- server has created a TCP **socket** that waits for a contacting client

## Client contacts server by:

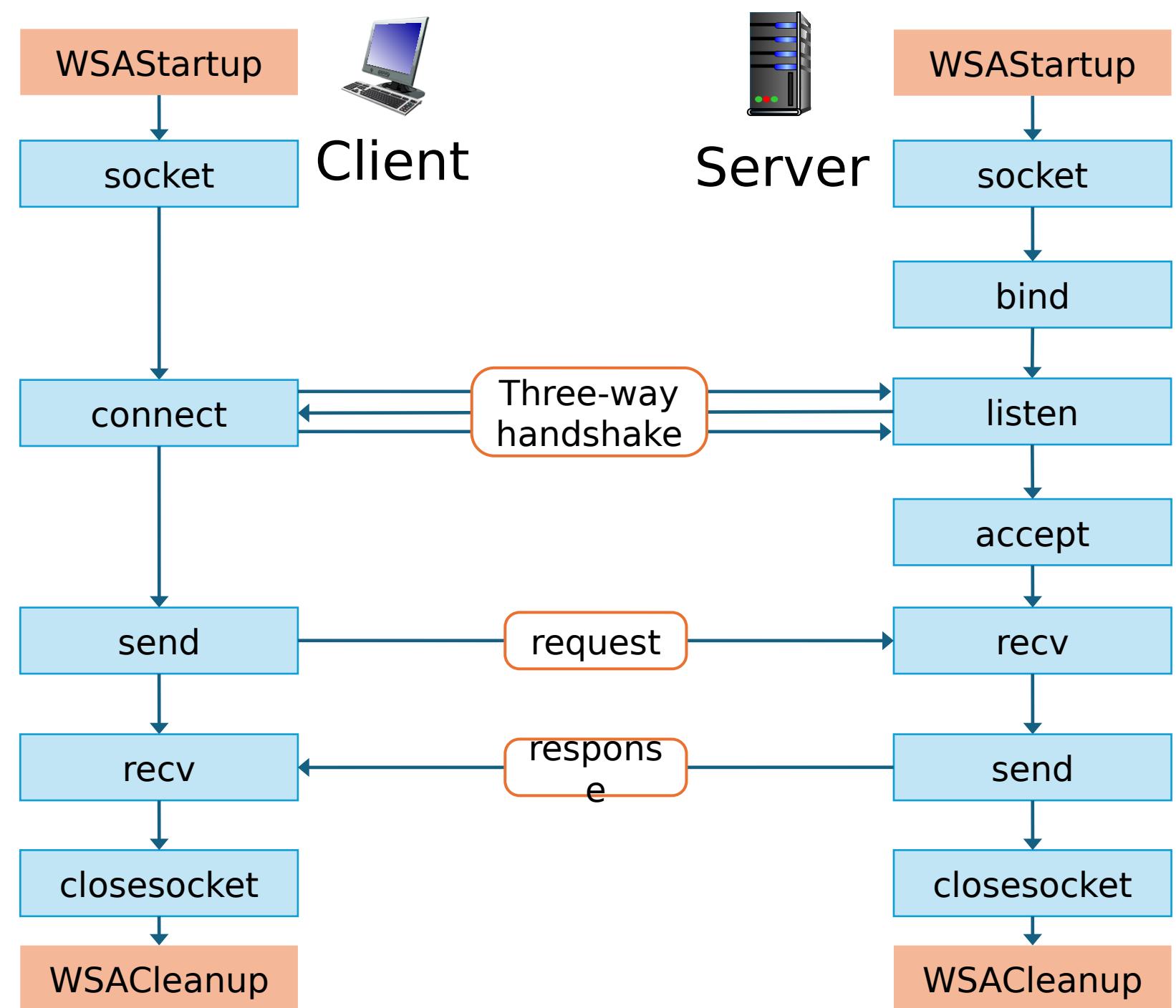
- Creating TCP socket, specifying IP address, port number of server process
- *When the server accepts a request*, a TCP connection is established

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

### Application viewpoint

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

# TCP client/server socket interaction



# TCP server

```
#include <winsock2.h>
```

Holds address of a received message

```
WSADATA wsa;
SOCKET socketListen;
SOCKET socketTCPconnection;
struct sockaddr_in addressServer;
struct sockaddr_in addressClient;
```

Start use of Winsock DLL  
*(Ws2\_32.dll)*

```
WSAStartup(MAKEWORD(2,2), &wsa);
```

Create TCP-socket  
(IPv4)

```
socketListen = socket(AF_INET, SOCK_STREAM, 0);
```

Any address

```
// Initialize address structure
```

Address is

```
addressServer.sin_addr.s_addr = INADDR_ANY;
```

Specify port number

```
addressServer.sin_family = AF_INET;
```

```
addressServer.sin_port = htons(nPortNumber); // LSB ->
```

MSB

assign **addressServer** to the socket

```
bind(socketListen, (struct sockaddr*)& addressServer,
 sizeof(addressServer));
```

wait for a TCP connection request

```
listen(socketListen, 5);
```

create socket for TCP connection:

```
socketTCPconnection = accept(socketListen,
```

# TCP server

Any address  
Address is  
Specify port number  
IPv4  
assign `addressServer` to the  
socket  
wait for a TCP connection  
request  
create socket for TCP connection  
+  
-----capture address-----

```
WSAStartup(MAKEWORD(2,2), &wsa);

socketListen = socket(AF_INET, SOCK_STREAM, 0);

// Initialize address structure
→ addressServer.sin_addr.s_addr = INADDR_ANY;
→ addressServer.sin_family = AF_INET;
→ addressServer.sin_port = htons(nPortNumber); // LSB -> MSB

→ bind(socketListen, (struct sockaddr*)& addressServer,
 sizeof(addressServer));
→ listen(socketListen, 5);

→ socketTCPconnection = accept(socketListen,
 (struct
 sockaddr*)&addressClient, &nSize);
-----recv(socketTCPconnection, sReceivedString, BUF_SIZE, 0);

send(...);

closesocket(socketTCPconnection); closesocket(socketListen);

WSACleanup();
```

# TCP client

```
SOCKET socketClient;
```

```
struct sockaddr_in addressServer;
char sTargetAddress[] = "127.0.0.1"; // loopback address
```

```
WSAStartup(MAKEWORD(2,2), &wsa);
```

Create TCP-socket  
(IPv4) → `socketClient = socket(AF_INET, SOCK_STREAM, 0 );`

Server's IP address -  
convert text-address to binary → `// Initialize target address structure`  
`InetPton(AF_INET, _TEXT(sTargetAddress), &`

Address form → `addressServer.sin_addr);`

Server's port number → `addressServer.sin_family = AF_INET;`  
`addressServer.sin_port = htons(nPortNumber); // LSB ->`

connect → MSB

```
connect(socketClient, (struct sockaddr *)&addressServer ,
```

Send message to server  
address → `sizeof(addressServer) );`

Receive message from  
server → `send(socketClient, sMessage, strlen(sMessage), 0);`

Close the socket → `recv( ... );`

```
closesocket(socketClient);
```

# Firewall blocking TCP port numbers

- Scanning for open TCP port numbers

```
C:\Users\Sigurd.PCL342>nmap 10.0.0.9 -p 54555
Starting Nmap 7.95 (https://nmap.org) at 2025-02-12 14:04 W. Europe Standard Time
Nmap scan report for 10.0.0.9
Host is up (0.0020s latency).

PORT STATE SERVICE
54555/tcp filtered unknown

MAC Address: E8:C8:29:85:A3:3B (Intel Corporate)

Nmap done: 1 IP address (1 host up) scanned in 0.53 seconds
```

# Active TCP connections

```
C:\Users\sigurde>netstat -a
```

## Active Connections

| Proto | Local Address       | Foreign Address     | State       |
|-------|---------------------|---------------------|-------------|
| TCP   | 128.39.201.58:12293 | 20.50.201.204:https | ESTABLISHED |
| TCP   | 128.39.201.58:12294 | 20.189.173.23:https | ESTABLISHED |
| TCP   | [::]:135            | UIA5CG4081L51:0     | LISTENING   |
| TCP   | [::]:445            | UIA5CG4081L51:0     | LISTENING   |

- netstat **-b** lists the process names
  - Requires elevated command prompt

# Packet analysis using Wireshark

Sigurd Eskeland

# ping command

- *ping* sends four **ICMP** Echo-requests (encapsulated in an IP datagram) to the destination address
- If the specified address is a URL, *ping* will need to do a DNS lookup to get the IP address
  - If the DNS entry does not exist in the local DNS cache, then the host will send a DNS request to the local DNS server (hosted by the IPS)
  - The command *nslookup* checks the local DNS cache first.
  - If no proper entry, then the host will send a DNS request to the local DNS server and et mappings from domain names (URL) to IP addresses

# ping output example

```
C:\Users\sigurde>ping uia.no

Pinging uia.no [2001:700:100:118::130] with 32 bytes of data:
Reply from 2001:700:100:118::130: time=7ms
Reply from 2001:700:100:118::130: time=9ms
Reply from 2001:700:100:118::130: time=9ms
Reply from 2001:700:100:118::130: time=9ms

Ping statistics for 2001:700:100:118::130:
 Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
 Minimum = 7ms, Maximum = 9ms, Average = 8ms
```

# Checking DNS server IP addresses

```
C:\Users\sigurde>ipconfig /all
```

```
Connection-specific DNS Suffix . : home
Description : Intel(R) Wi-Fi 6E AX211 160MHz
Physical Address. : E8-C8-29-85-A3-3B
DHCP Enabled. : Yes
Autoconfiguration Enabled : Yes
IPv6 Address. : 2001:464d:9098:0:20ff:d76b:c97c:cb26(Preferred)
IPv6 Address. : 2001:464d:9098:0:c9d8:cd01:e4df:3442(Preferred)
Lease Obtained. : Tuesday, 11 March, 2025 10:44:39
Lease Expires : Tuesday, 11 March, 2025 13:04:39
Temporary IPv6 Address. : 2001:464d:9098:0:28c7:bac9:e4b7:f48f(Preferred)
Link-local IPv6 Address : fe80::a214:30ce:213f:5b4b%19(Preferred)

IPv4 Address. : 10.0.0.9(Preferred)
Subnet Mask : 255.255.255.0
Lease Obtained. : Tuesday, 11 March, 2025 03:01:39
Lease Expires : Tuesday, 11 March, 2025 13:31:18
Default Gateway : fe80::1633:75ff:fece:6040%19
 10.0.0.138
DHCP Server : 10.0.0.138
DHCPv6 IAID : 183027753
DHCPv6 Client DUID. : 00-01-00-01-2D-67-19-A3-2C-58-B9-B5-8F-42
DNS Servers : 2001:4600:4:1fff::253
 2001:4600:4:fff::253
 148.122.164.253
 148.122.16.253
```



Apply a display filter ... &lt;Ctrl-/&gt;



| No. | Time     | Source                               | Destination                          | Protocol | Length | Info                                                                    |
|-----|----------|--------------------------------------|--------------------------------------|----------|--------|-------------------------------------------------------------------------|
| 1   | 0.000000 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1026:2402:1::7                  | TLSv1.2  | 102    | Application Data                                                        |
| 2   | 0.061562 | 2603:1026:2402:1::7                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TCP      | 74     | 443 → 52678 [ACK] Seq=1 Ack=29 Win=49151 Len=0                          |
| 3   | 0.911287 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2001:4600:4:1fff::253                | DNS      | 86     | Standard query 0xad84 A uia.no                                          |
| 4   | 0.911388 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2001:4600:4:1fff::253                | DNS      | 86     | Standard query 0x3129 AAAA uia.no                                       |
| 5   | 0.929604 | 2001:4600:4:1fff::253                | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | DNS      | 114    | Standard query response 0x3129 AAAA uia.no AAAA 2001:700:100:118::130   |
| 6   | 0.929604 | 2001:4600:4:1fff::253                | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | DNS      | 102    | Standard query response 0xad84 A uia.no A 129.240.118.130               |
| 7   | 0.946011 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2001:700:100:118::130                | ICMPv6   | 94     | Echo (ping) request id=0x0001, seq=1, hop limit=128 (reply in progress) |
| 8   | 0.953676 | 2001:700:100:118::130                | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | ICMPv6   | 94     | Echo (ping) reply id=0x0001, seq=1, hop limit=55 (request in progress)  |
| 9   | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 113    | Application Data                                                        |
| 10  | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 937    | Application Data                                                        |
| 11  | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 334    | Application Data                                                        |
| 12  | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 113    | Application Data                                                        |
| 13  | 1.446657 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1026:c0f:15::2                  | TCP      | 74     | 52286 → 443 [ACK] Seq=1 Ack=903 Win=1025 Len=0                          |
| 14  | 1.446834 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1026:c0f:15::2                  | TCP      | 74     | 52286 → 443 [ACK] Seq=1 Ack=1202 Win=1024 Len=0                         |
| 15  | 1.602851 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1020:705:8::402                 | TLSv1.2  | 118    | Application Data                                                        |

> Frame 3: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF\_{...}  
 > Ethernet II, Src: Intel\_85:a3:3b (e8:c8:29:85:a3:3b), Dst: ZyxelCommuni\_ce:60:40 (14:  
 > Internet Protocol Version 6, Src: 2001:464d:9098:0:28c7:bac9:e4b7:f48f, Dst: 2001:4600:4:1fff::253  
 > User Datagram Protocol, Src Port: 63926, Dst Port: 53  
 ✓ Domain Name System (query)

Transaction ID: 0xad84

&gt; Flags: 0x0100 Standard query

Questions: 1

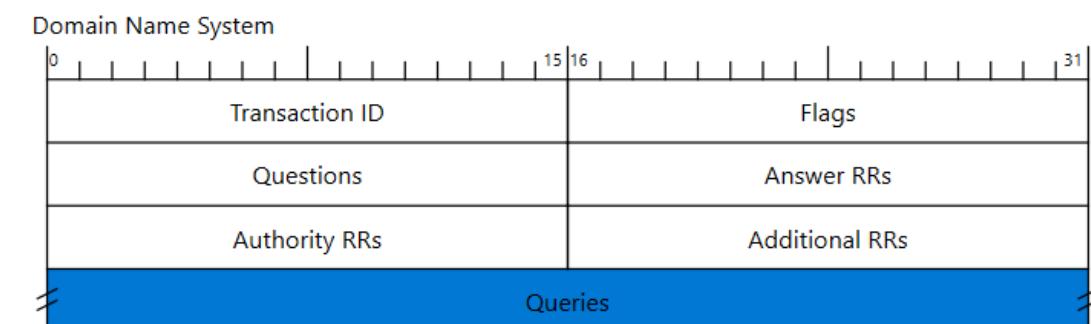
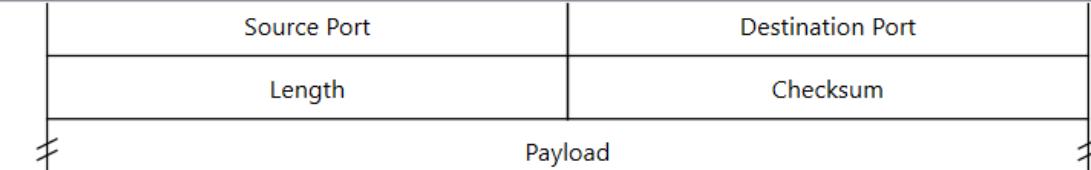
Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

✓ Queries

&gt; uia.no: type A, class IN

[\[Response In: 6\]](#)



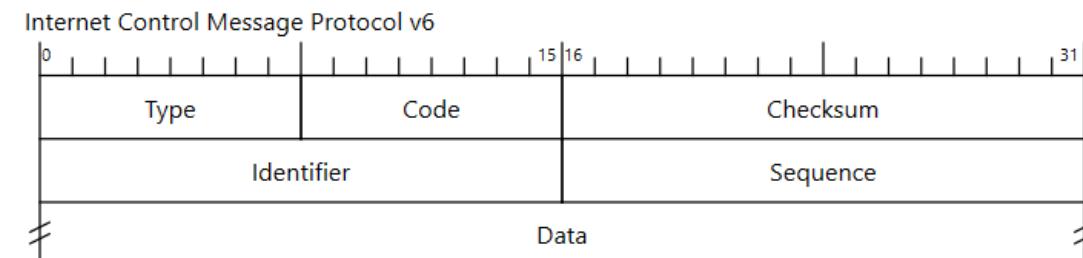
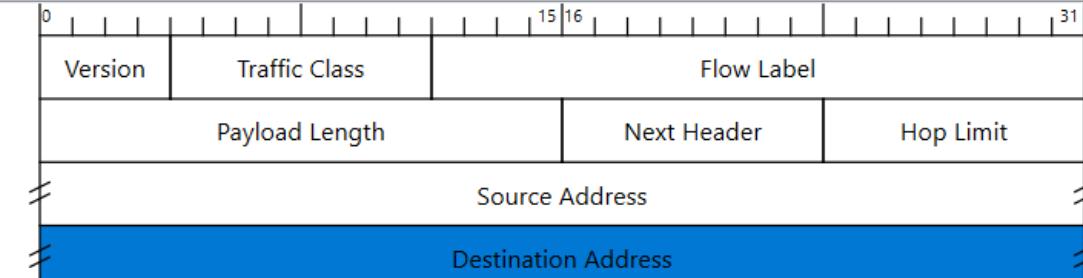
Apply a display filter ... &lt;Ctrl-/&gt;



| No. | Time     | Source                               | Destination                          | Protocol | Length | Info                                                             |
|-----|----------|--------------------------------------|--------------------------------------|----------|--------|------------------------------------------------------------------|
| 1   | 0.000000 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1026:2402:1::7                  | TLSv1.2  | 102    | Application Data                                                 |
| 2   | 0.061562 | 2603:1026:2402:1::7                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TCP      | 74     | 443 → 52678 [ACK] Seq=1 Ack=29 Win=49151 Len=0                   |
| 3   | 0.911287 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2001:4600:4:1ffff::253               | DNS      | 86     | Standard query 0xad84 A uia.no                                   |
| 4   | 0.911388 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2001:4600:4:1ffff::253               | DNS      | 86     | Standard query 0x3129 AAAA uia.no                                |
| 5   | 0.929604 | 2001:4600:4:1ffff::253               | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | DNS      | 114    | Standard query response 0x3129 AAAA uia.no AAAA 2001:700:100:11  |
| 6   | 0.929604 | 2001:4600:4:1ffff::253               | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | DNS      | 102    | Standard query response 0xad84 A uia.no A 129.240.118.130        |
| 7   | 0.946011 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2001:700:100:118::130                | ICMPv6   | 94     | Echo (ping) request id=0x0001, seq=1, hop limit=128 (reply in 8) |
| 8   | 0.953676 | 2001:700:100:118::130                | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | ICMPv6   | 94     | Echo (ping) reply id=0x0001, seq=1, hop limit=55 (request in 7)  |
| 9   | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 113    | Application Data                                                 |
| 10  | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 937    | Application Data                                                 |
| 11  | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 334    | Application Data                                                 |
| 12  | 1.446175 | 2603:1026:c0f:15::2                  | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | TLSv1.2  | 113    | Application Data                                                 |
| 13  | 1.446657 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1026:c0f:15::2                  | TCP      | 74     | 52286 → 443 [ACK] Seq=1 Ack=903 Win=1025 Len=0                   |
| 14  | 1.446834 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1026:c0f:15::2                  | TCP      | 74     | 52286 → 443 [ACK] Seq=1 Ack=1202 Win=1024 Len=0                  |
| 15  | 1.602851 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1020:705:8::402                 | TLSv1.2  | 118    | Application Data                                                 |

Next Header: ICMPv6 (58)  
 Hop Limit: 128  
 > Source Address: 2001:464d:9098:0:28c7:bac9:e4b7:f48f  
 > Destination Address: 2001:700:100:118::130  
 [Stream index: 2]

Internet Control Message Protocol v6  
 Type: Echo (ping) request (128)  
 Code: 0  
 Checksum: 0xf6ee [correct]  
 [Checksum Status: Good]  
 Identifier: 0x0001  
 Sequence: 1  
[Response In: 8]  
 Data (32 bytes)  
 Data: 6162636465666768696a6b6c6d6e6f7071727374757677616263646566676869  
 [Length: 32]



# Show devices connected to the local network

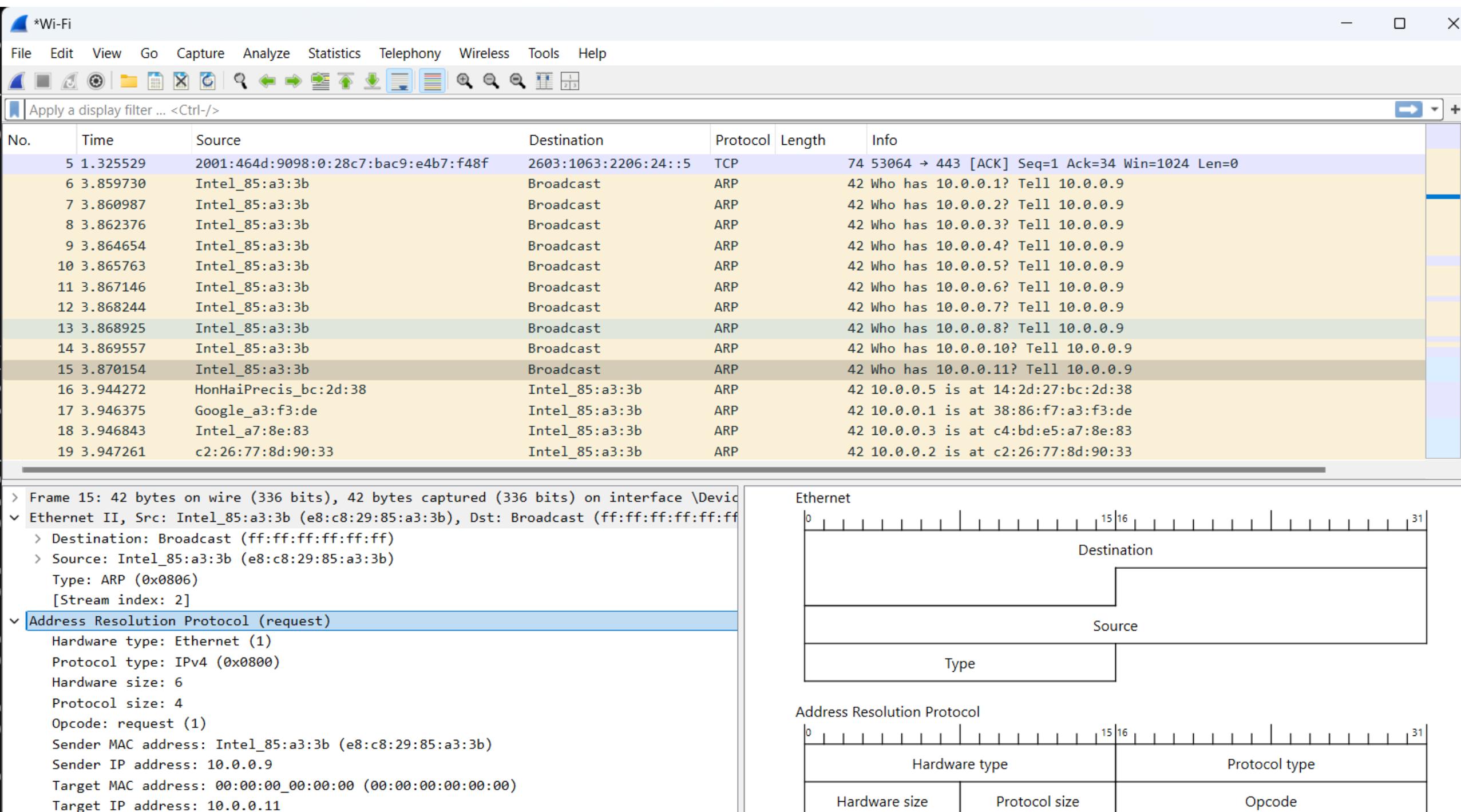
- nmap -sn 192.168.0.0/24
- nmap -sn 10.0.0.0/28

(replacing the subnet with the appropriate one for your LAN)

- nmap broadcast an **ARP request** on the local subset for each IP address in the specified subnet address

# nmap output example

```
C:\Users\sigurde>nmap -sn 10.0.0.0/28
Starting Nmap 7.95 (https://nmap.org) at 2025-03-18 09:37 W. Europe Standard Time
Nmap scan report for 10.0.0.1
Host is up (0.065s latency).
MAC Address: 38:86:F7:A3:F3:DE (Google)
Nmap scan report for 10.0.0.2
Host is up (0.053s latency).
MAC Address: C2:26:77:8D:90:33 (Unknown)
Nmap scan report for 10.0.0.5
Host is up (0.083s latency).
MAC Address: 14:2D:27:BC:2D:38 (Hon Hai Precision Ind.)
Nmap scan report for 10.0.0.9
Host is up.
Nmap done: 16 IP addresses (4 hosts up) scanned in 1.79 seconds
```





Apply a display filter ... &lt;Ctrl-/&gt;



| No. | Time     | Source                               | Destination          | Protocol | Length | Info                                          |
|-----|----------|--------------------------------------|----------------------|----------|--------|-----------------------------------------------|
| 5   | 1.325529 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2603:1063:2206:24::5 | TCP      | 74     | 53064 → 443 [ACK] Seq=1 Ack=34 Win=1024 Len=0 |
| 6   | 3.859730 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.1? Tell 10.0.0.9               |
| 7   | 3.860987 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.2? Tell 10.0.0.9               |
| 8   | 3.862376 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.3? Tell 10.0.0.9               |
| 9   | 3.864654 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.4? Tell 10.0.0.9               |
| 10  | 3.865763 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.5? Tell 10.0.0.9               |
| 11  | 3.867146 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.6? Tell 10.0.0.9               |
| 12  | 3.868244 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.7? Tell 10.0.0.9               |
| 13  | 3.868925 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.8? Tell 10.0.0.9               |
| 14  | 3.869557 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.10? Tell 10.0.0.9              |
| 15  | 3.870154 | Intel_85:a3:3b                       | Broadcast            | ARP      | 42     | Who has 10.0.0.11? Tell 10.0.0.9              |
| 16  | 3.944272 | HonHaiPrecis_bc:2d:38                | Intel_85:a3:3b       | ARP      | 42     | 10.0.0.5 is at 14:2d:27:bc:2d:38              |
| 17  | 3.946375 | Google_a3:f3:de                      | Intel_85:a3:3b       | ARP      | 42     | 10.0.0.1 is at 38:86:f7:a3:f3:de              |
| 18  | 3.946843 | Intel_a7:8e:83                       | Intel_85:a3:3b       | ARP      | 42     | 10.0.0.3 is at c4:bd:e5:a7:8e:83              |
| 19  | 3.947261 | c2:26:77:8d:90:33                    | Intel_85:a3:3b       | ARP      | 42     | 10.0.0.2 is at c2:26:77:8d:90:33              |

> Frame 16: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF\_{...}

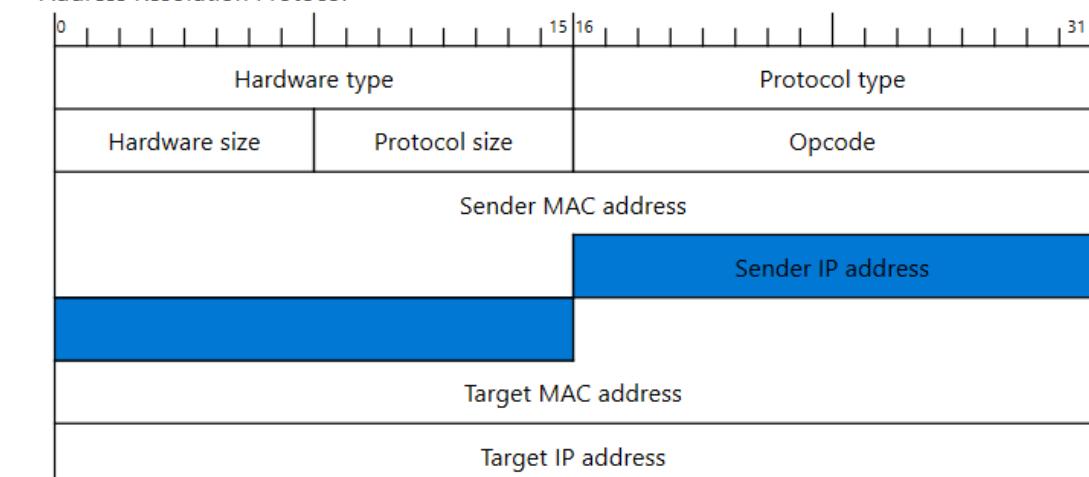
▼ Ethernet II, Src: HonHaiPrecis\_bc:2d:38 (14:2d:27:bc:2d:38), Dst: Intel\_85:a3:3b (e8:c8:29:85:a3:3b)

- > Destination: Intel\_85:a3:3b (e8:c8:29:85:a3:3b)
- > Source: HonHaiPrecis\_bc:2d:38 (14:2d:27:bc:2d:38)
- Type: ARP (0x0806)
- [Stream index: 3]

▼ Address Resolution Protocol (reply)

- Hardware type: Ethernet (1)
- Protocol type: IPv4 (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: reply (2)
- Sender MAC address: HonHaiPrecis\_bc:2d:38 (14:2d:27:bc:2d:38)
- Sender IP address: 10.0.0.5
- Target MAC address: Intel\_85:a3:3b (e8:c8:29:85:a3:3b)
- Target IP address: 10.0.0.9

## Address Resolution Protocol



# Another nmap example

- When looking up an *external* server, nmap sends an ICMP request

```
C:\Users\sigurde>nmap -sn uia.no
Starting Nmap 7.95 (https://nmap.org) at 2025-03-11 14:12 W. Europe Standard Time
Nmap scan report for uia.no (129.240.118.130)
Host is up (0.013s latency).
Other addresses for uia.no (not scanned): 2001:700:100:118::130
rDNS record for 129.240.118.130: lb-w3d-prod-vip-vortex-www.uio.no
Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds
```



Apply a display filter ... &lt;Ctrl-/&gt;



Return the main window text to its normal size

| Time    | Source                               | Destination             | Protocol  | Length | Info                                                         |
|---------|--------------------------------------|-------------------------|-----------|--------|--------------------------------------------------------------|
| .973992 | 2001:464d:9098:0:28c7:bac9:e4b7:f48f | 2001:2030:0:4e::d59b... | TCP       | 75     | 53801 → 443 [ACK] Seq=1 Ack=1 Win=1025 Len=1                 |
| .990278 | 2001:2030:0:4e::d59b:9d58            | 2001:464d:9098:0:28c... | TCP       | 86     | 443 → 53801 [ACK] Seq=1 Ack=2 Win=501 Len=0 SRE=1 SRE=2      |
| .807233 | 10.0.0.9                             | 10.0.0.1                | TCP       | 164    | 51436 → 8009 [PSH, ACK] Seq=1 Ack=1 Win=1023 Len=110         |
| .916541 | 10.0.0.1                             | 10.0.0.9                | TCP       | 164    | 8009 → 51436 [PSH, ACK] Seq=1 Ack=111 Win=400 Len=110        |
| .964947 | 10.0.0.9                             | 10.0.0.1                | TCP       | 54     | 51436 → 8009 [ACK] Seq=111 Ack=111 Win=1022 Len=0            |
| .022915 | ZyxelCommuni_ce:60:40                | Broadcast               | HomePl... | 21     | MAC Management                                               |
| .615379 | 10.0.0.9                             | 129.240.118.130         | ICMP      | 42     | Echo (ping) request id=0xd6ce, seq=0/0, ttl=59 (reply in 19) |
| .624506 | 129.240.118.130                      | 10.0.0.9                | ICMP      | 42     | Echo (ping) reply id=0xd6ce, seq=0/0, ttl=55 (request in 18) |
| .626011 | 10.0.0.9                             | 129.240.118.130         | TCP       | 58     | 51115 → 443 [SYN] Seq=0 Win=1024 Len=0 MSS=1460              |
| .626652 | 10.0.0.9                             | 129.240.118.130         | TCP       | 54     | 51115 → 80 [ACK] Seq=1 Ack=1 Win=1024 Len=0                  |
| .627252 | 10.0.0.9                             | 129.240.118.130         | ICMP      | 54     | Timestamp request id=0x9deb, seq=0/0, ttl=41                 |
| .637030 | 129.240.118.130                      | 10.0.0.9                | TCP       | 58     | 443 → 51115 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460  |
| .637030 | 129.240.118.130                      | 10.0.0.9                | TCP       | 54     | 80 → 51115 [RST] Seq=1 Win=0 Len=0                           |
| .637030 | 129.240.118.130                      | 10.0.0.9                | ICMP      | 54     | Timestamp reply id=0x9deb, seq=0/0, ttl=55                   |
| .639561 | 10.0.0.9                             | 148.122.16.253          | DNS       | 88     | Standard query 0x5ecd PTR 130.118.240.129.in-addr.arpa       |

Protocol: ICMP (1)  
 Header Checksum: 0x11f0 [validation disabled]  
 [Header checksum status: Unverified]  
 Source Address: 10.0.0.9  
 Destination Address: 129.240.118.130  
 [Stream index: 4]

Internet Control Message Protocol  
 Type: 8 (Echo (ping) request)  
 Code: 0  
 Checksum: 0x2131 [correct]  
 [Checksum Status: Good]  
 Identifier (BE): 54990 (0xd6ce)  
 Identifier (LE): 52950 (0xed6c)  
 Sequence Number (BE): 0 (0x0000)  
 Sequence Number (LE): 0 (0x0000)  
 [Response frame: 19]

