

COMP3331/9331

# Computer Networks and Applications

2024 Term 2

Lecturer in Charge: **Mahbub Hassan**

Course Outline & Logistics

1<sup>st</sup> Course on  
Computer Networks



© Scott Adams, Inc./Dist. by UFS, Inc.

# Who cares about computer networking?

	2023 Revenue (US\$)
Google	305 Billion
Facebook (Meta)	134 Billion
Cisco	57 Billion

## Today's Agenda

- Course (non-technical) details
- Logistics: How we will roll
- What is this course about?
- Introduction to Computer Networks (course content begins)

# Course Staff

- Lecturer-in-Charge:  
Mahbub Hassan
- Course Admin: Isura  
Nirmal
- Tutors (tentative):
  - Yihe Yan
  - Ayda Valinezhad Orang
  - Gary (Jiawei) Hu
  - Isura Nirmal
  - Ravin Gunawardena
  - Jueming Chen
  - Varun Agarwal
  - Tim Arney
  - Mahshid Gohari

Yifan Wang  
Cheng Jiang  
Wei Song  
Thirasara Pathiranage  
Manmeet Dhaliwal  
Pooja Gupta  
Yanxiang Wang  
Navodika Gedara  
Muhammad Akhtar  
Daijao Liu  
Mark Cardamis



# Resources

Very important

- <https://webcms3.cse.unsw.edu.au/COMP3331/24T2/>
- Everything is posted on the course website
  - **Course Outline (PLEASE READ THIS THOROUGHLY)**
  - Lecture Notes
  - Video Recordings
  - Lab Schedules, Allocations and Locations
  - Assignment and Lab Exercises
  - Homework Problems
  - Exam Information
  - Consultation hours
  - **Announcement:** Your responsibility to check the announcement forum (in Ed Forums) on regular basis for important updates/changes to schedule, etc.
  - **Your active participation and interaction is crucial to ensure that all of us get the most out of this course**
  - Note: You will need to login using your **zID/zPass**

# Me

- Professor of Computer Networking at UNSW
- PhD in Computer Networking (Monash Uni)
- 30 yrs teaching and research experience in Computer Networking
- Computer Networking books authored/co-authored:
  - Wireless and Mobile Networking, CRC Press, 2022
  - High Performance TCP/IP Networking, Prentice Hall, 2004
  - Engineering Internet Quality of Service, Artech House, 2002
  - Performance of TCP/IP over ATM Networks, Artech House, 2000
- Winner of Teaching Excellence Awards (Monash Uni and UNSW)
- More details available from personal pages:
  - <https://www.cse.unsw.edu.au/~mahbub/>
  - <https://www.linkedin.com/in/mahbub-hassan-b11259187/>

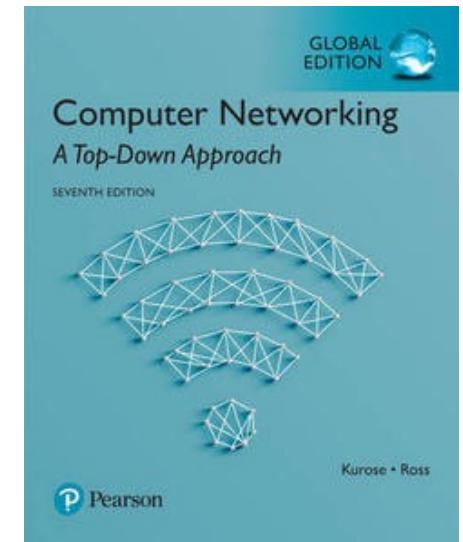


# You

- Mix of UG (mostly 2<sup>nd</sup>/3<sup>rd</sup> year) and PG (mostly 1<sup>st</sup> year)
- Mostly CSE students but a few from other Engineering schools (Mech, EET) and Faculties (Business, Science, Law)
- Assumed Knowledge:
  - COMP1927/COMP2521/MTRN3500
  - Good understanding of data structures, algorithms, basic probability theory
  - Proficient in one of the following programming languages: C, Java or Python
  - We DO NOT assume that you know anything about computer networks

# Course Material

- **Textbook:** *Computer Networking: A Top Down Approach*, Jim Kurose, Keith Ross, Addison-Wesley (Pearson), 8<sup>th</sup> Edition, 2020
  - UNSW Book Shop Links: [Physical](#)
- Lecture Notes (on WebCMS)
- Links/articles on additional material
- Reference Books:
  - *Computer Networks: A Systems Approach*, Larry Peterson and Bruce Davie, Morgan Kaufmann, Fourth Edition, 2007.
  - *Unix Network Programming Volume 1 - Networking APIs: Sockets and XTI*, W. Richard Stevens, Prentice Hall, Second Edition, 1998 (Third edition also available)
  - *Java Network Programming*, E. R. Harold, O'Reilly, Third Edition, 2004.
  - *Wireless and Mobile networking*, M. Hassan, CRC Press 2022 (more details about wireless networking)
- Links to programming help



## Course Aims

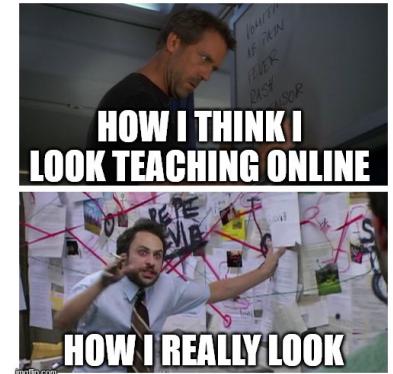
- To gain in-depth introduction to the key topics in the field of computer networks, including the Internet
- To obtain hands-on understanding of networking protocols
- To gain skills in network programming, designing and implementing network protocols, evaluating network performance and problem solving
- To build necessary foundational knowledge required in more advanced networking courses

## Teaching/Learning Strategies

- Lectures (9 weeks, 4-hr per week)
- Labs (five 2-hr labs)
  - Hands-on learning (enforcing theory taught in lectures)
- Programming Assignment
  - Network programming and protocol design
- Weekly Homework/Quiz (Self-assessed)
  - Problem solving skills (enforcing theory taught in lectures)

# Lectures

- In-person face-to-face: Mon & Thu 14:00 - 16:00
- Weeks 1-5 and 7-10 (2 x 2-hour lectures x 9 weeks)
- Lecture Recordings
  - Echo360 accessed via Moodle
- We will focus on most important concepts and theories reinforced with
  - Practical labs
  - Problem solving exercises/quizzes
  - Ed forum discussions
- Certain material will be left for self study
  - These will be indicated on the lecture notes
- In-lecture polls and quizzes
  - For you to reinforce concepts
  - For me to get an indication of your understanding



## Labs

- 2-hour lab sessions starting **Week 2** (Weeks 2-5, 7-10)
- **Mix of in-person and online labs (as per your enrolment)**
  - In-person labs: CSE labs, online: Teams/Zoom – meeting links will be posted on course webpage
- Hands-on experiments related to concepts covered in lectures
  - Wireshark packet sniffer, ns-2 network simulator, other network measurement tools, socket programming practice
- 8 lab sessions:
  - 5 Lab Exercises (guided by tutors)
    - Lab report to be submitted (no demos)
    - Highly encouraged to attempt lab tasks before attending labs
  - 2 Problem-based learning sessions (Tutorials in Week 5 & 10)
    - No marks
  - 1 programming tutorial (Week 7)
    - No marks



## Online Labs: VLAB

- Access CSE lab environment on your own machine remotely
- Uses VNC
- Recommended client: TigerVNC (<https://tigervnc.org>)
- [https://taggi.cse.unsw.edu.au/FAQ/Really\\_quick\\_guide\\_to\\_VLAB/](https://taggi.cse.unsw.edu.au/FAQ/Really_quick_guide_to_VLAB/)
- UNSW VPN: <https://www.myit.unsw.edu.au/services/students/remote-access-vpn>
- You will need to know basic command line Linux commands:  
<http://www.unixguide.net/linux/linuxshortcuts.shtml>



# Getting help

- Use online discussion forums on Ed (WebCMS forum disabled)
  - Must join Ed first: <https://edstem.org/au/join/PjBmH4>
  - Fellow students benefit from your questions
  - Fellow students can answer your questions
  - Develop a community
- Use **cs3331@cse.unsw.edu.au** for communication with us.
  - DO NOT email LiC/admin on personal email address
- Consultation hours
  - LiC for lecture-related help – 1 hour each week
  - Tutor consultations for assignment help - C/Python/Java
- Tutors
  - Establish an agreeable mode of communicating with your tutor

Scan to join  
Ed Forum



## Revisions based on myExperience Feedback

- Student feedback from Term 1 was generally positive
- Request for more programming support for assignment
  - We have replaced 1 lab session with a programming tutorial
  - We are allocating extra tutor consultations for assignment help

## Code of Conduct

- CSE offers an inclusive learning environment for all students. In anything connected to UNSW, including social media, these things are student misconduct and will not be tolerated:
  - racist/sexist/offensive language or images
  - sexually inappropriate behaviour
  - bullying, harassing or aggressive behaviour
  - invasion of privacy
- Show respect to your fellow students and course staff
- Staff are also reminded to show respect to students

# Plagiarism



What is plagiarism?

Presenting the (thoughts or) work of another as your own. Cheating of any kind constitutes academic misconduct and carries a range of penalties. Please read course intro for details.

Examples of inappropriate conduct:

- groupwork on assignments/labs (discussion OK)
- allowing another student to copy your work
- getting your hacker cousin to code for you
- purchasing a solution to the assignment

**Remember:** You are only cheating yourself and chances are you will get caught!

# Plagiarism



- Labs, assignments, exams must be entirely your own work
- You **can not** work on assignment as a pair (or group)
- Plagiarism will be checked for and penalized
- Plagiarism may result in suspension from UNSW
- Scholarship students may lose scholarship
- International students may lose visa
- Supplying your work to any another person may result in loss of all your marks for the lab/assignment
- If you store your code in online repositories DO NOT MAKE IT PUBLICLY ACCESSIBLE (THIS IS ASSUMED TO BE PLAGIARISM)

# Assessment

- **Hands-on – 40%**
  - Labs 20%
  - Assignment 20%
    - Assignment due in Week 9
    - Implement a networked application or protocol
    - We assume you are proficient in one of C/Java/Python (coding skills are must in most practical networking jobs!)
- **Concepts and theory – 60%**
  - Mid-term exam (20%):
    - **Week 7**
    - Open-book online exam (**Inspera**)
  - Final Exam (40%)
    - End of term
    - Open-book exam (**Inspera**)
    - Hurdle – **must score at least 40% to pass the course**
  - **Inspera** platform <https://www.student.unsw.edu.au/exams/inspera>

# Assessment

**NOTE:** To pass the course, a student MUST receive at least **40% marks on the final exam**

```
lab = marks for lab exercises (20 marks)
assign = mark for the programming assignment (20 marks)
midTerm = mark for the mid-semester exam (20 marks)
scaledfinalExam = scaled mark for the final exam (out of
40 marks)
mark = lab + assign + midTerm + scaledfinalExam
Grade:
= HD|DN|CR|PS if mark >= 50 && scaledfinalExam >= 16
= FL           if mark < 50 || scaledfinalExam < 16
```

**NOTE: If you cannot clear the final exam hurdle (after scaling), reported grade would  
be 'UF' with maximum marks reported as 40**

## How to do well in this course

- Keep up with and absorb all the content
  - Clear weekly tasks; do not accumulate
  - **This is an intense course requiring full attention**
- A critical/analytical viewpoint will help
- Solve all homework/practice problems
- Do the lab exercises *yourself*
- Do the assignment *yourself*
- Practice, practice, practice



# Let's Work Together

- Course Staff
  - Regular communication about upcoming deadlines (weekly notices)
  - Timely response to questions
  - Timely feedback on assessments
- Students
  - Take responsibility
  - Be aware of deadlines/deliverables and how to access resources
    - Links for lectures/labs/consults/exams
    - VLAB for labs and assignments
  - Check course notices regularly
  - Ask questions through the appropriate channels (online forum is preferred)
  - Participate in lectures and forum (community building)



## What is this course about ?

- Introductory course in computer network
- Learn *principles* and *practice* of computer networking
- We use the **Internet** as a vehicle to understand the core concepts of networking

# What is this course about ?

1. To learn how the Internet works
  - Internet is a complex global infrastructure
  - What are the organising principles behind the Internet?
  - What really happens when you “browse the Web”?
  - What are TCP/IP, DNS, HTTP, NAT, VPNs, 802.11,... anyway?



## What is this course about ?

1. To learn how the Internet works
  - Internet is a complex global infrastructure
  - What are the organising principles behind the Internet?
  - What really happens when you “browse the Web”?
  - What are TCP/IP, DNS, HTTP, NAT, VPNs, 802.11,... anyway?
  
2. To learn the fundamentals of computer networks
  - What issue you need to take into consideration to make a computer network work well?
  - What design strategies have proven valuable?
  - How do we evaluate network performance?

## Where could I go from here?

- COMP 4336/9336: Mobile Data Networking
- COMP6733: Internet of Things Design Studio
- COMP 9334: System Capacity and Planning
- COMP 3441/9441: Security Engineering
- COMP 4337/9337: Securing Wireless Networks
  
- Thesis/Coursework Projects
- Research Degree (MPhil, PhD)



# Computer Networks and Applications

COMP 3331/COMP 9331

## Key Topics

- Internet as a network of networks
- The protocol stack and layering principle
- Edge vs. Core
- Loss, delay and throughput
- Packet switching vs. Circuit switching

Week 1

## Introduction to Computer Networks

Reading Guide: Chapter 1, Sections 1.1 - 1.4

# Acknowledgment

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

# Introduction

## *Our goal:*

- ❖ Get “feel,” “big picture,” introduction to terminology
  - more depth, detail *later* in course
- ❖ Approach:
  - use Internet as example

## *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 (self study, not on exam)**
- ❖ **History (self study, not on exam)**

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

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



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

# The Internet: a “nuts and bolts” view



Billions of connected computing **devices**:

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

**Packet switches:** forward packets (chunks of data)

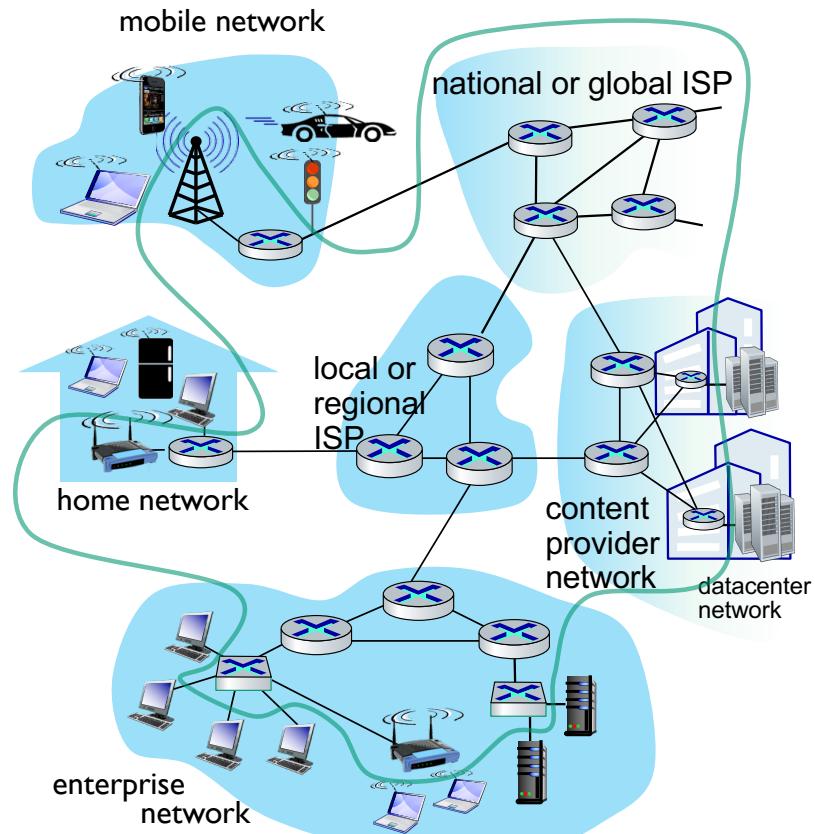
- routers, switches

**Communication links**

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

**Networks**

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



# “Fun” Internet appliances



Security Camera



Picture frame



Web-enabled toaster +  
weather forecaster



car



Amazon Echo



Internet  
refrigerator



Networked TV Set top Boxes



sensorized,  
bed  
mattress



pacemaker



Tweet-a-watt:  
monitor energy use



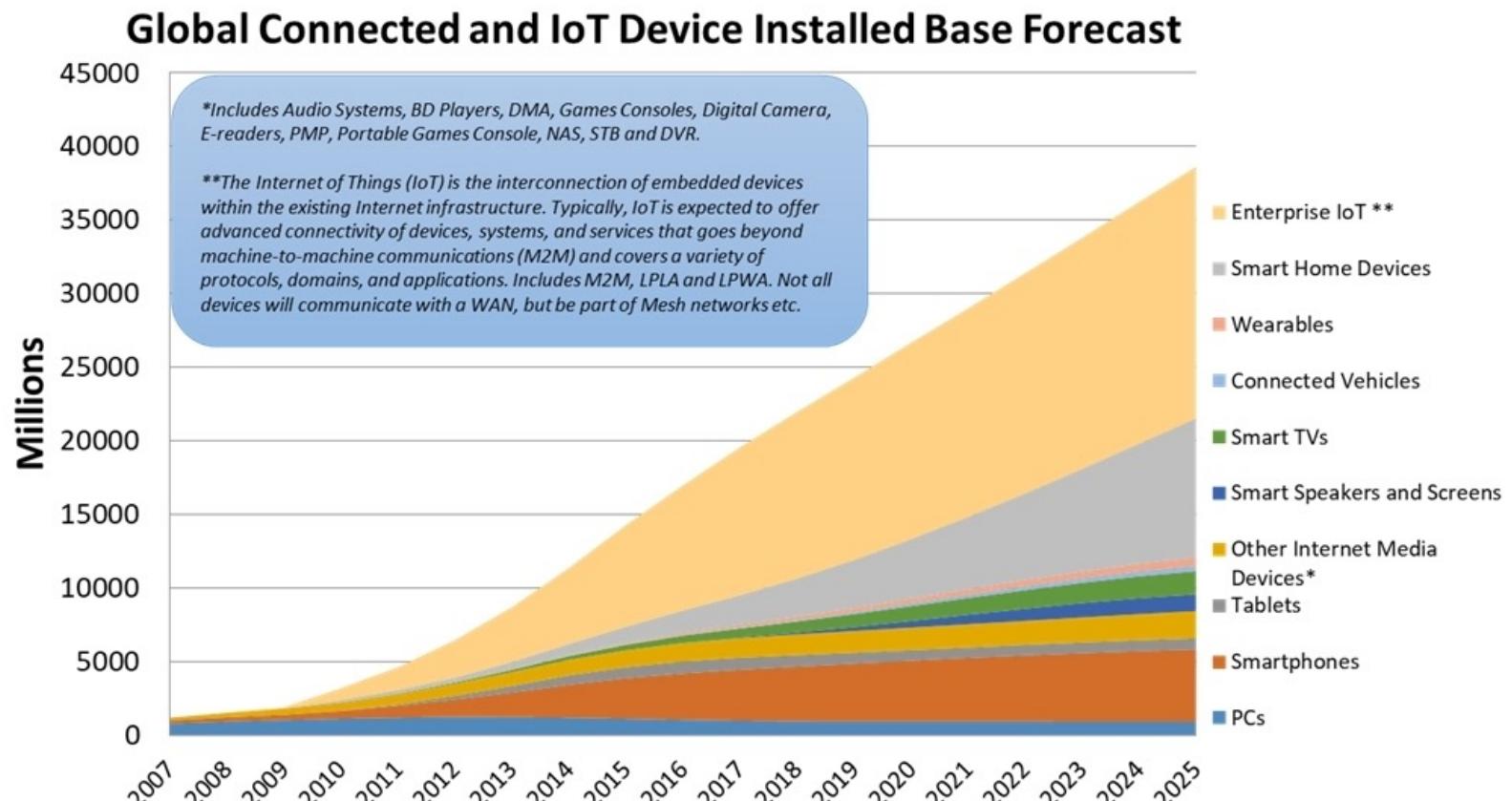
Fitbit



Smart Lightbulbs



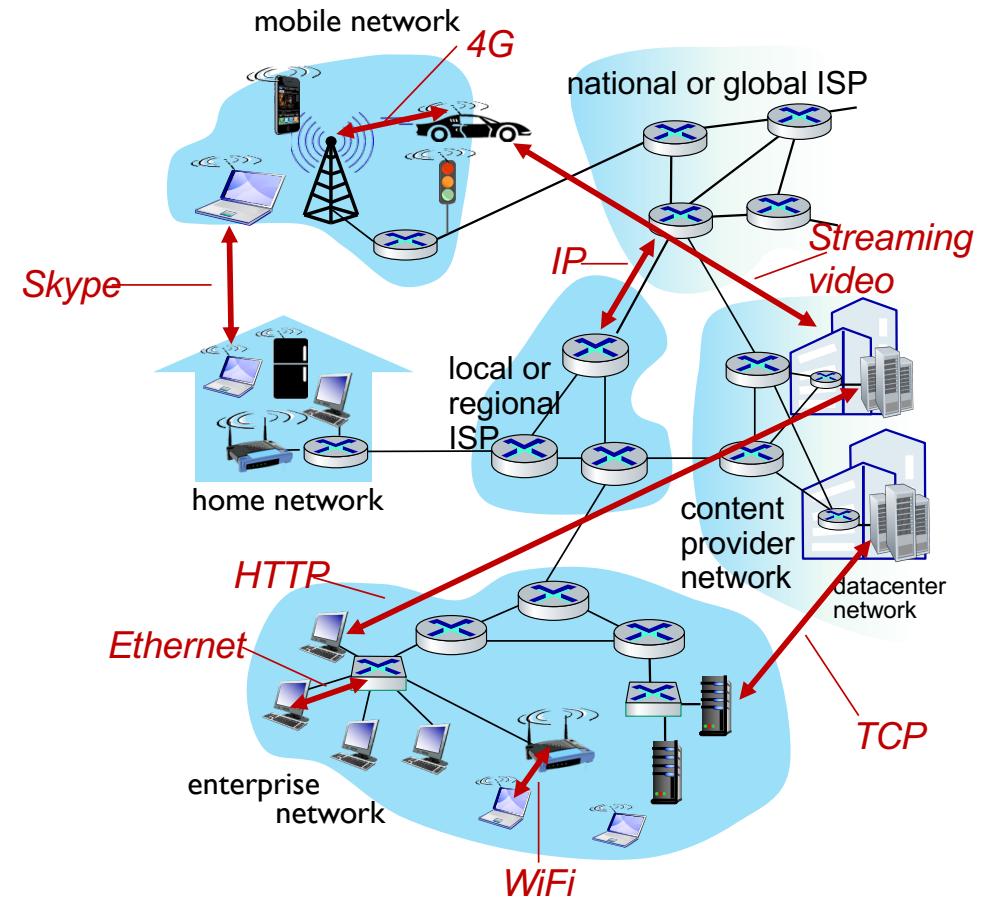
AR devices



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

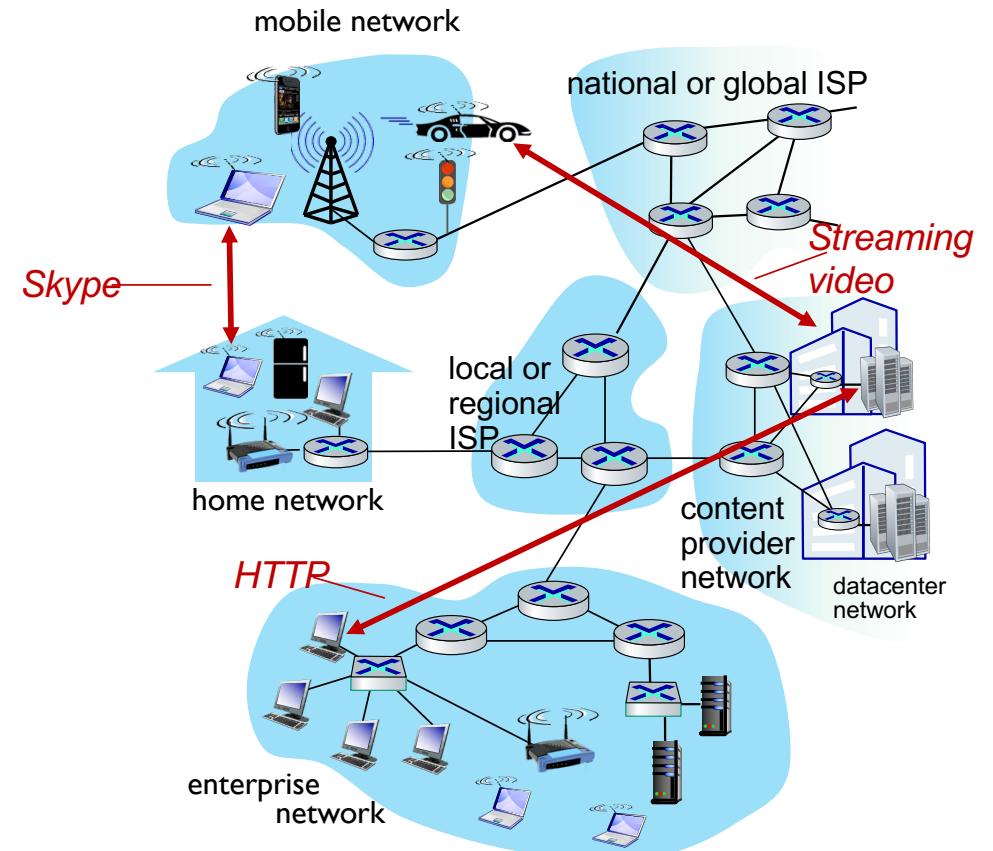
# The Internet: a “nuts and bolts” view

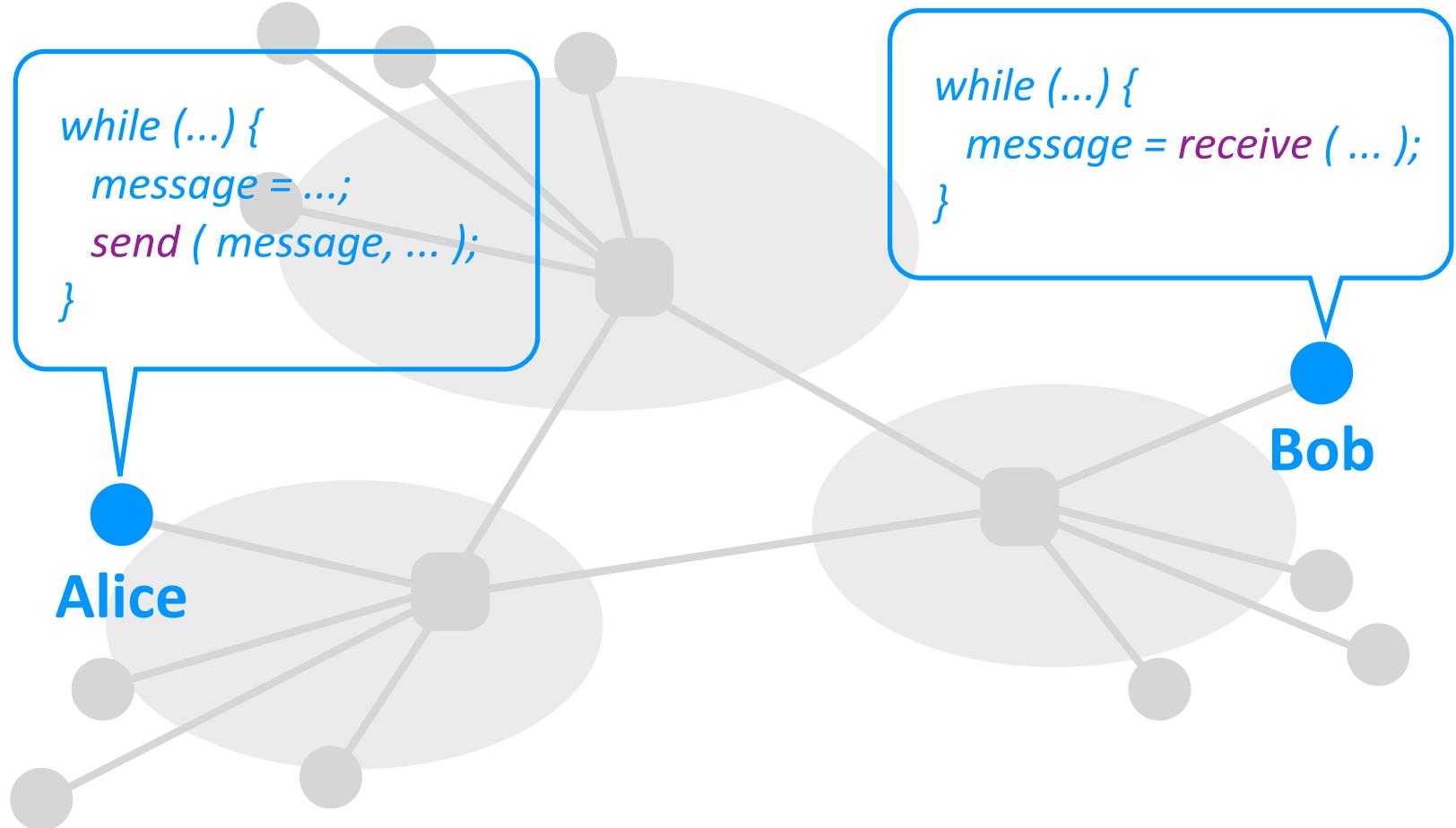
- ❖ *Internet: “network of networks”*
  - Interconnected ISPs
  - *protocols are everywhere*
    - control sending, receiving of messages
    - e.g., HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, Ethernet
  - *Internet standards*
    - RFC: Request for Comments
    - IETF: Internet Engineering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

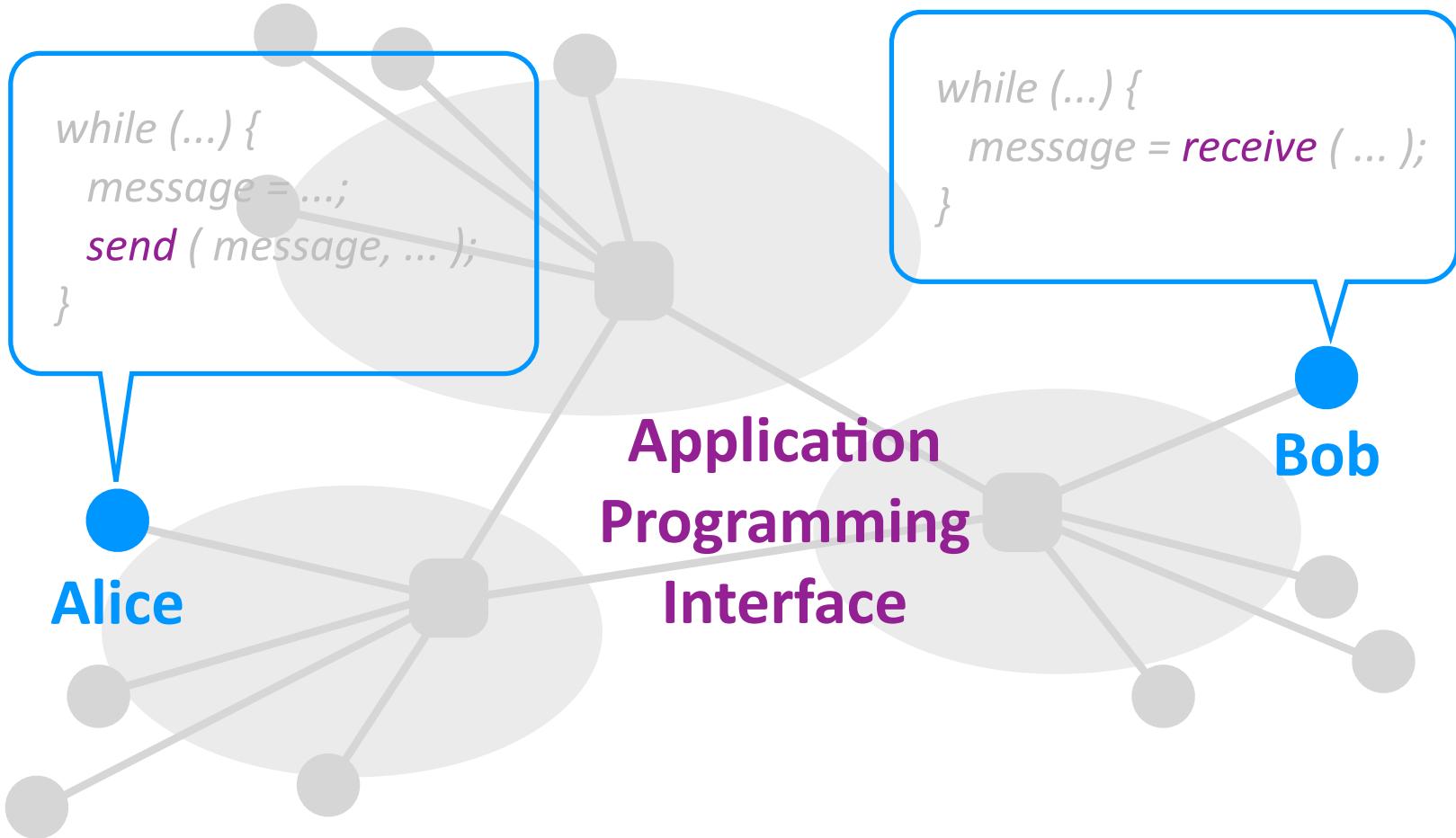


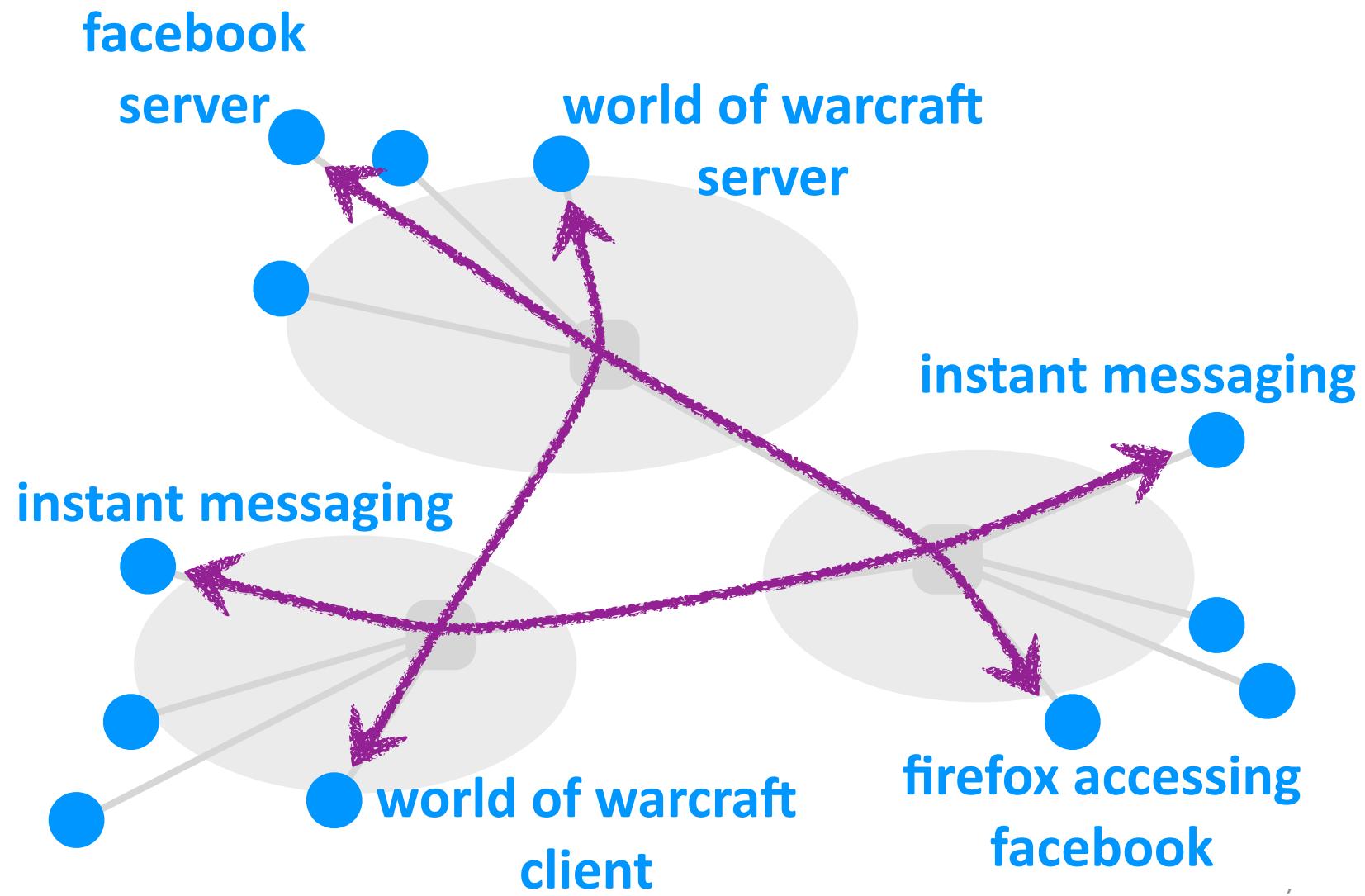
# The Internet: a “service” view

- ❖ **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

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

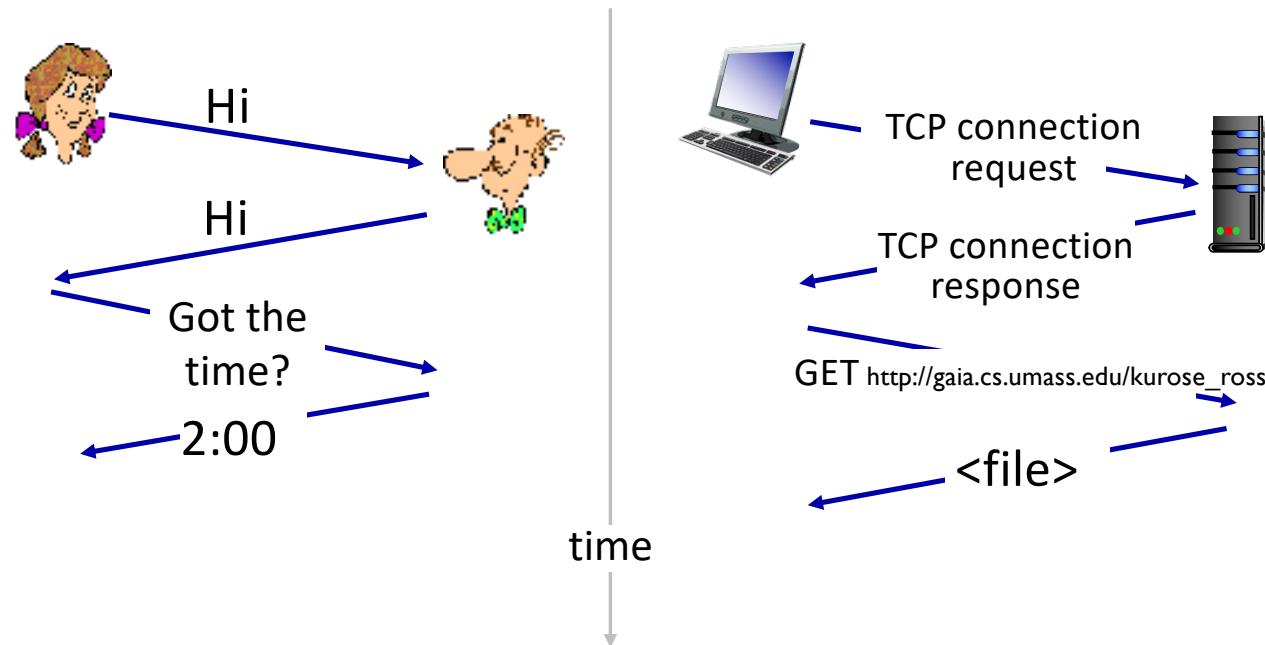
## *Network protocols:*

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

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

# What's a protocol?

A human protocol and a computer network protocol:



*Q:* other human protocols?



## Quiz: Internet of Things

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

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

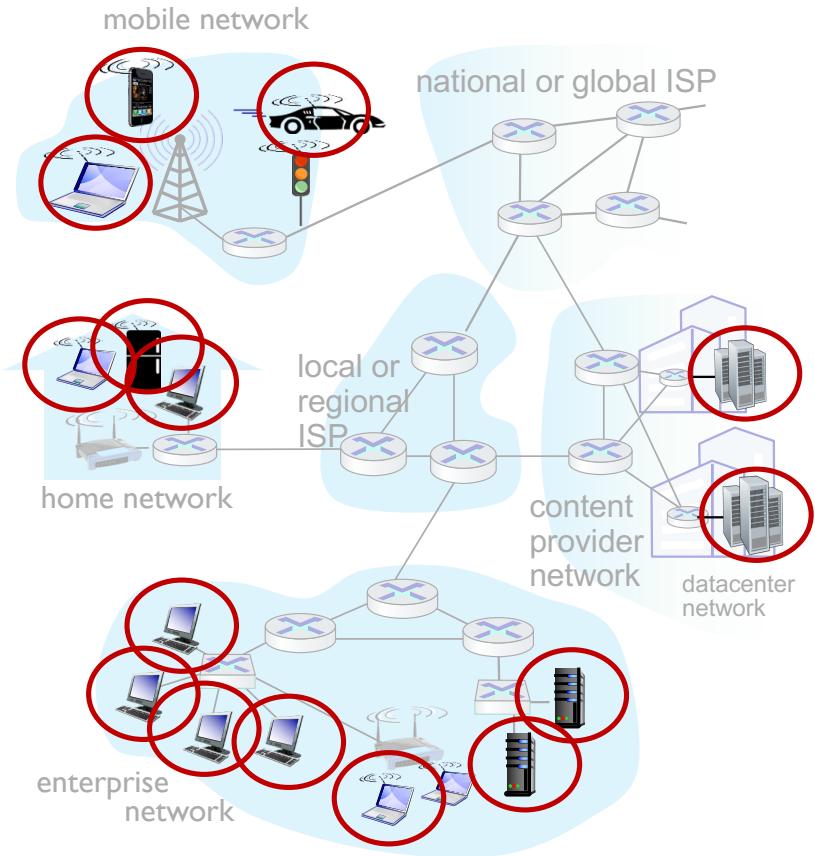
# Introduction: roadmap

- ❖ 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



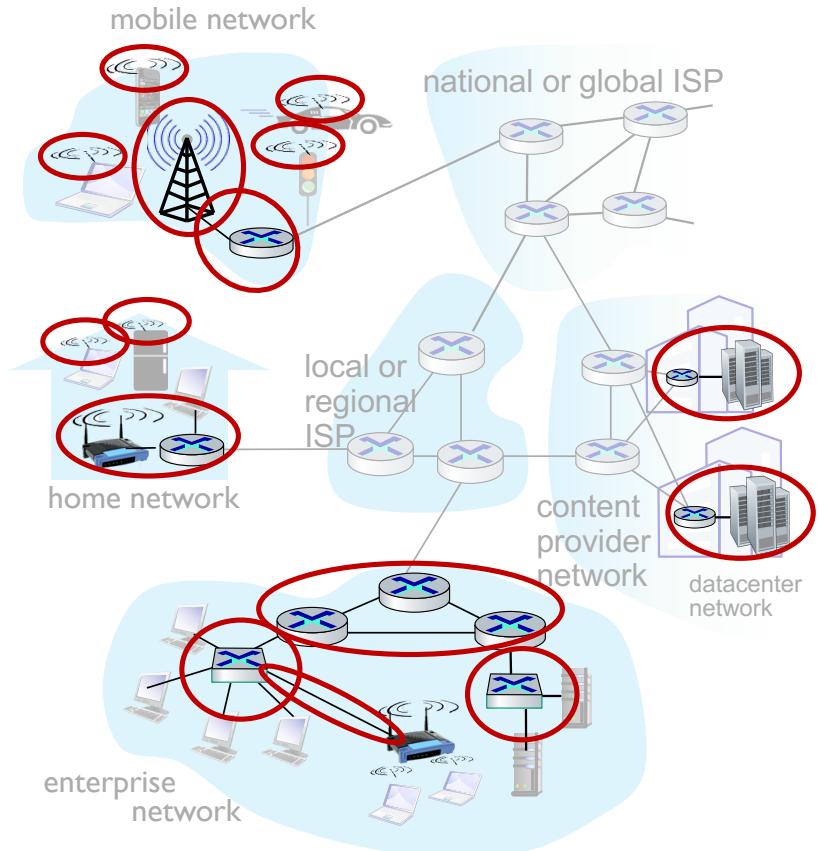
# A closer look at Internet structure

## 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:

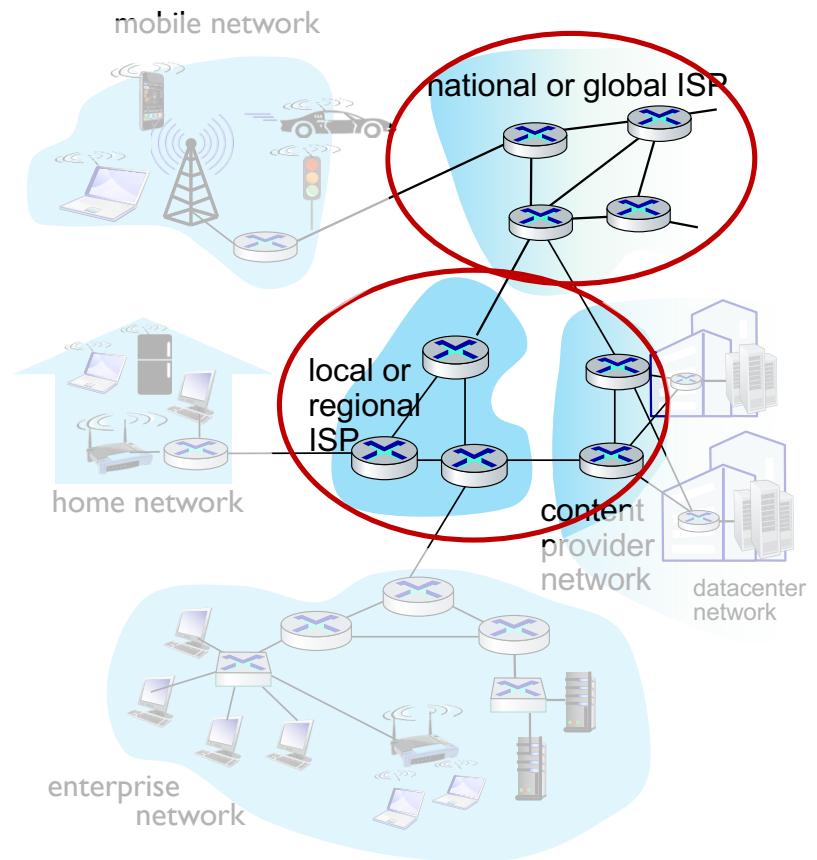
- ❖ hosts: clients and servers
- ❖ servers often in data centers

## Access networks, physical media:

- ❖ wired, wireless communication links

## Network core:

- interconnected routers
- network of networks



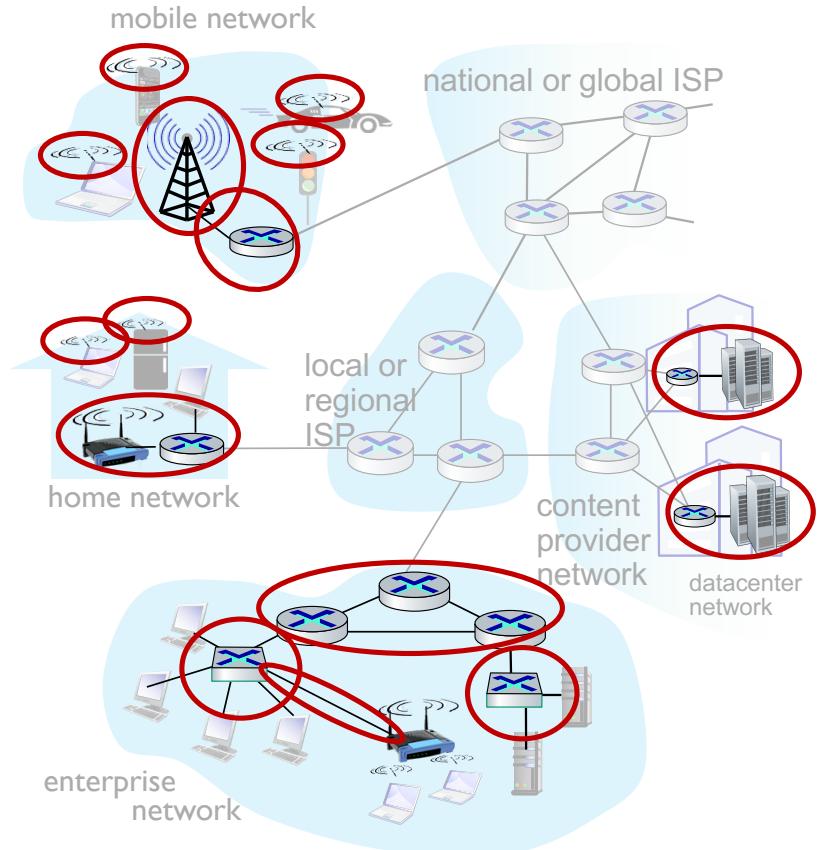
# Access networks and physical media

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

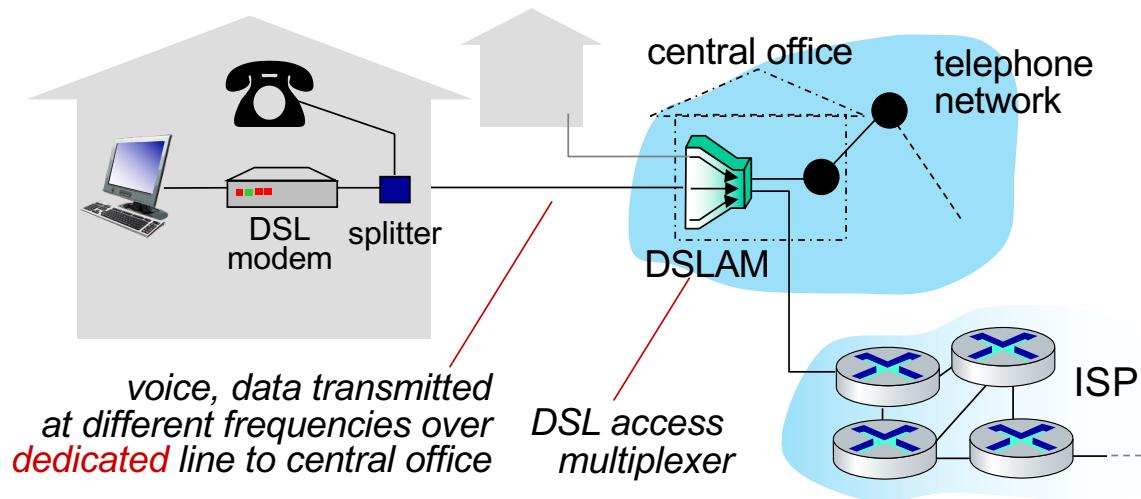
- ❖ residential access nets
- ❖ institutional access networks (school, company)
- ❖ mobile access networks (WiFi, 4G/5G)

*What to look for:*

- transmission rate (bits per second) of access network?
- shared or dedicated access among users?

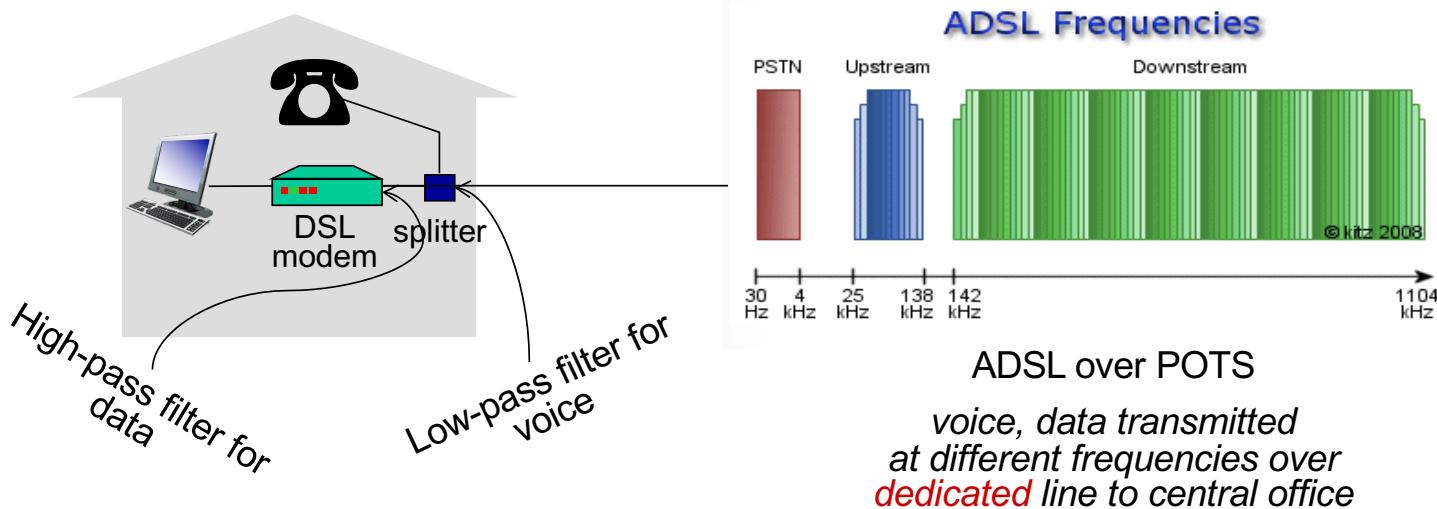


# Access networks: digital subscriber line (DSL)



- 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

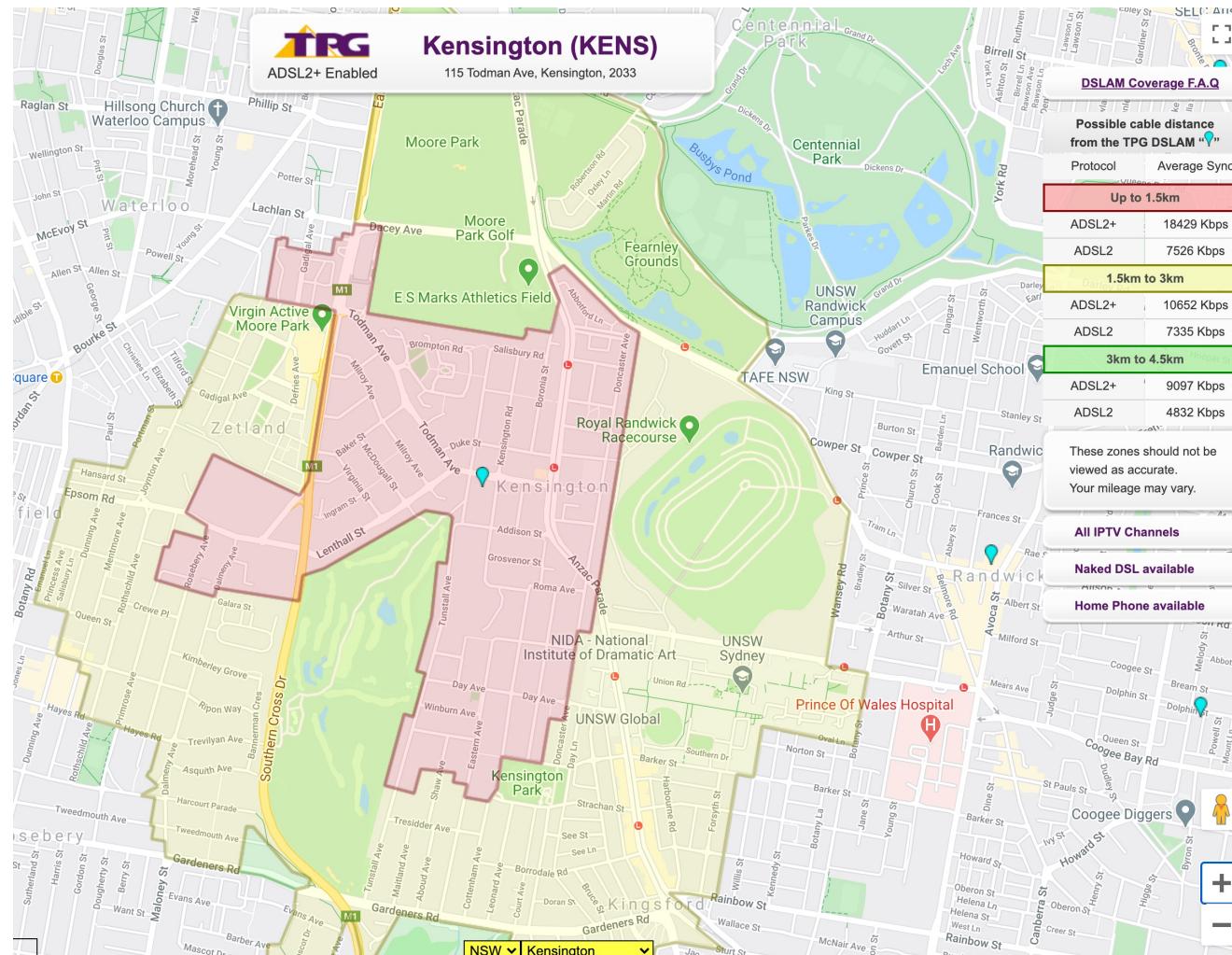
# Access net: digital subscriber line (DSL)



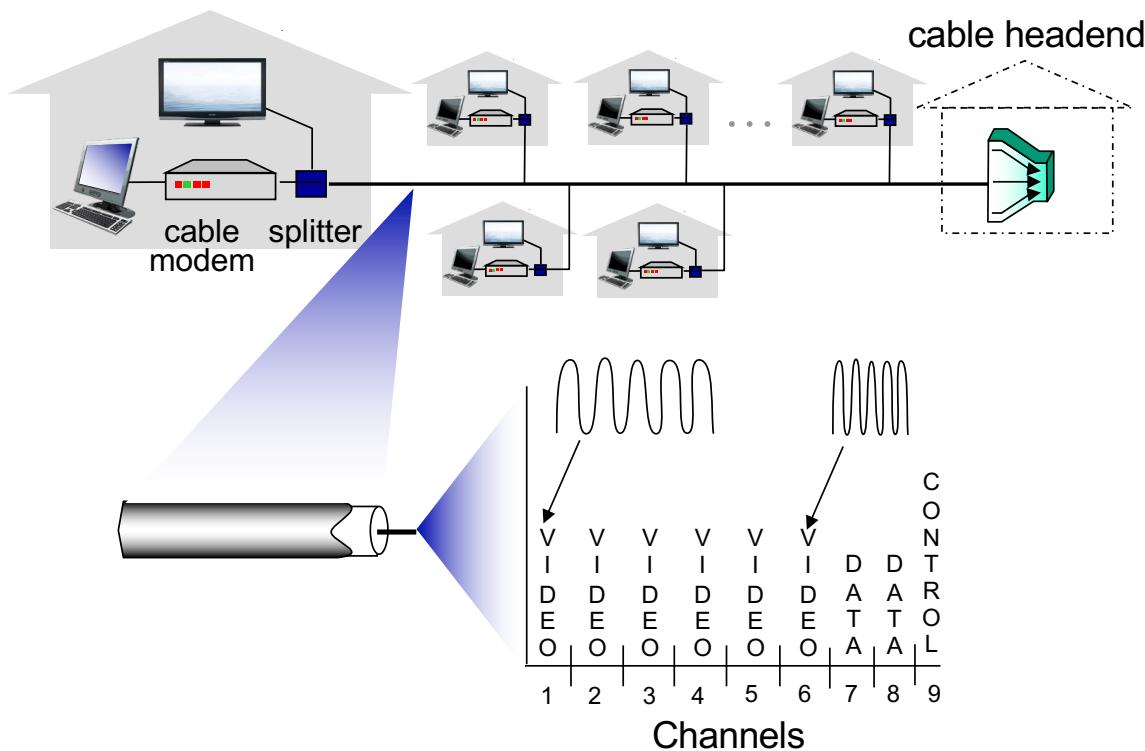
Different data rates for upload and download (ADSL)

- 24-52 Mbps dedicated downstream transmission rate
- 3.5-16 Mbps dedicated upstream transmission rate

# Access net: digital subscriber line (DSL)

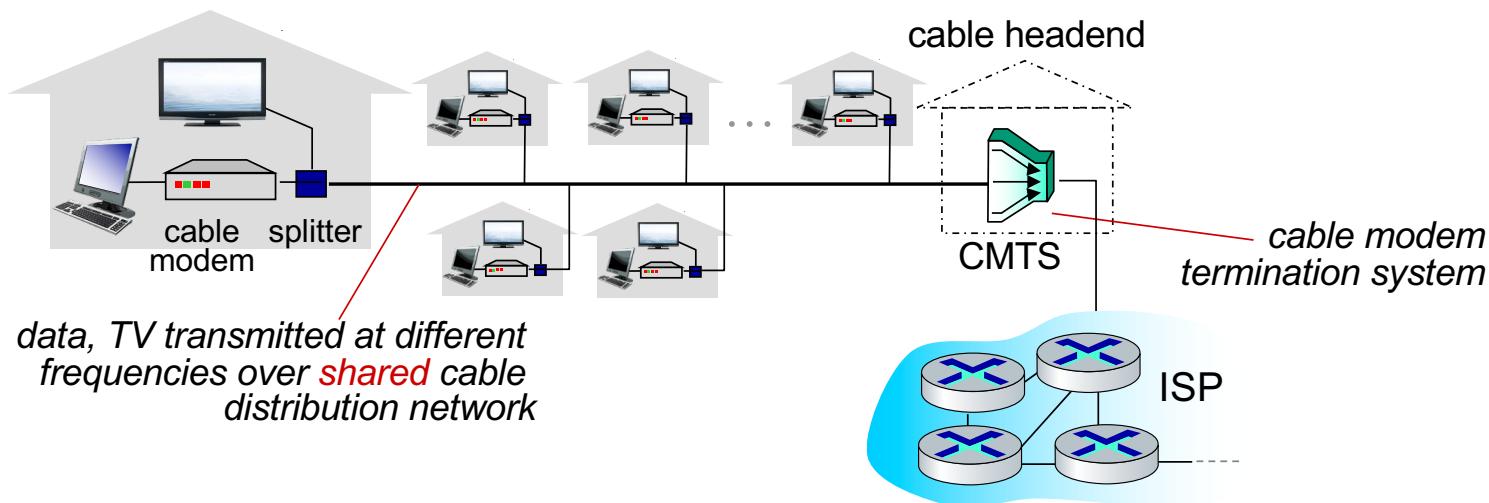


# Access networks: cable-based access



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

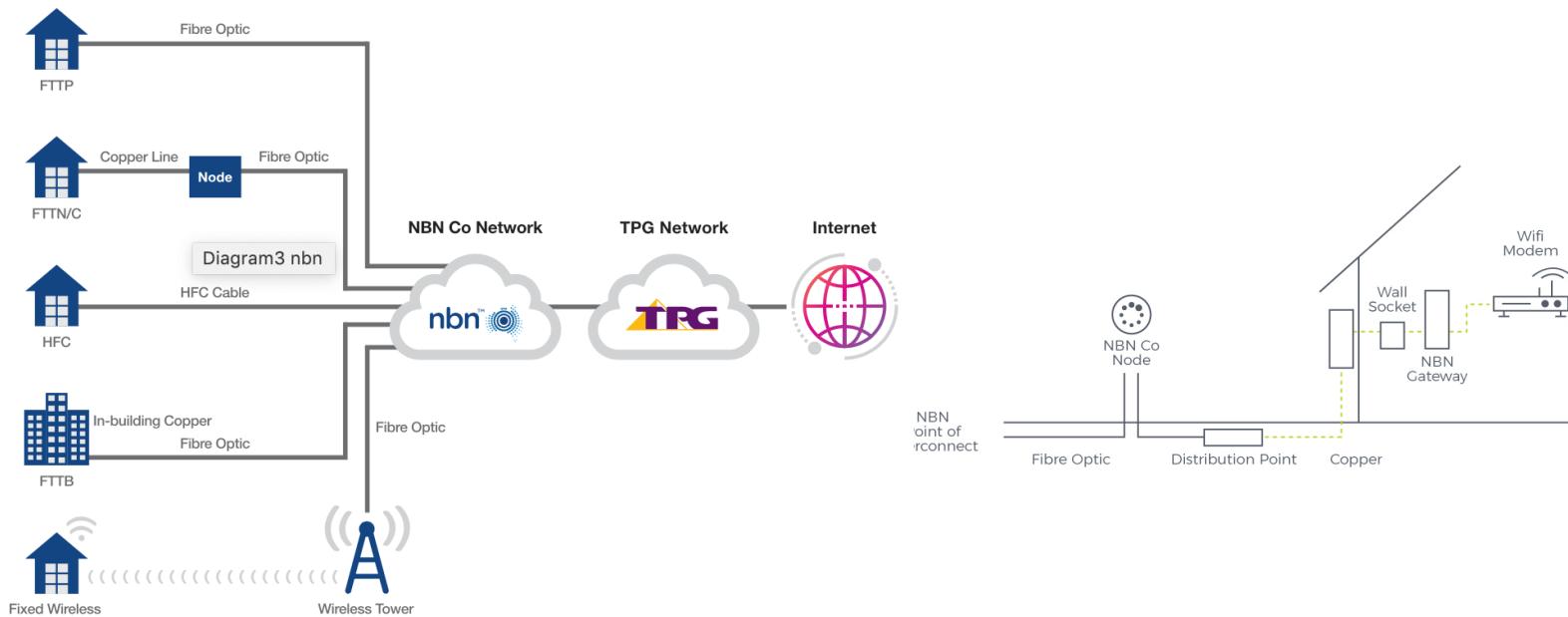
# Access networks: cable-based access



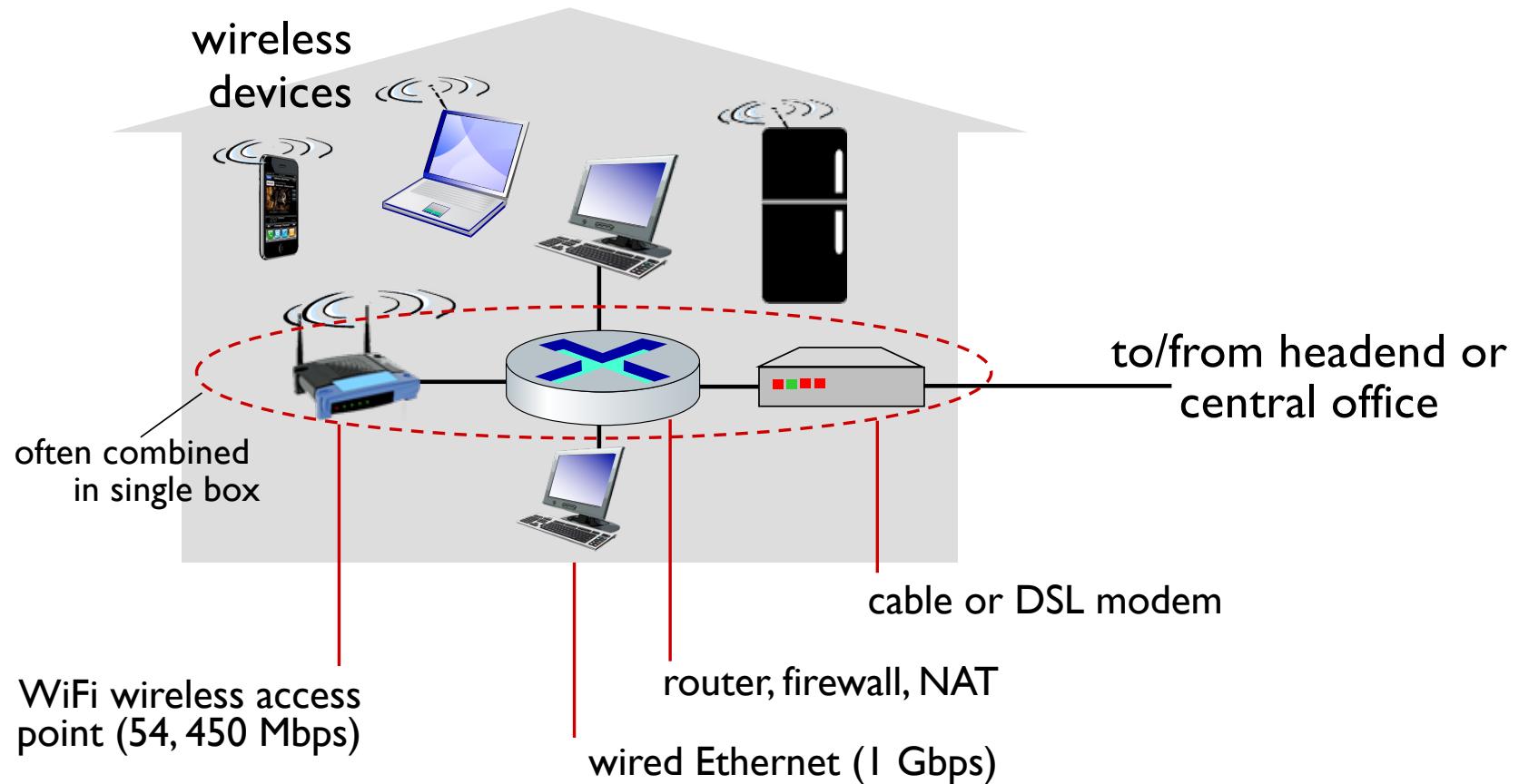
- HFC: hybrid fiber coax
  - asymmetric: up to 40 Mbps – 1.2 Gbs downstream transmission rate, 30-100 Mbps upstream transmission rate
- network of cable, fiber attaches homes to ISP router
  - homes **share access network** to cable headend
  - Unlike DSL, which has dedicated access to central office

# Fiber to the home/premise/curb

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



# Access networks: home networks



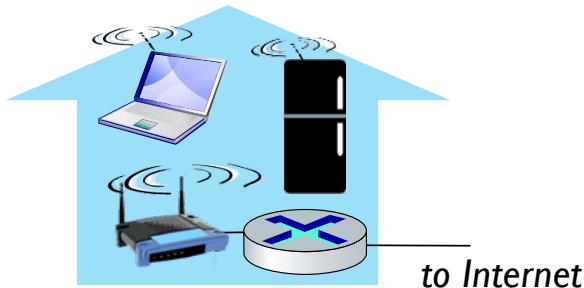
# Wireless access networks

Shared wireless access network connects end system to router

- via base station aka “access point”

## Wireless local area networks (WLANs)

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

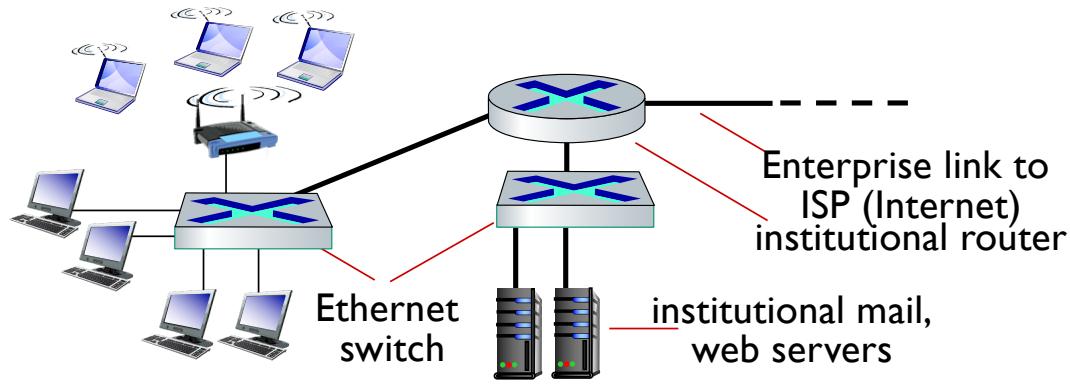


## Wide-area cellular access networks

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



# Access networks: enterprise networks

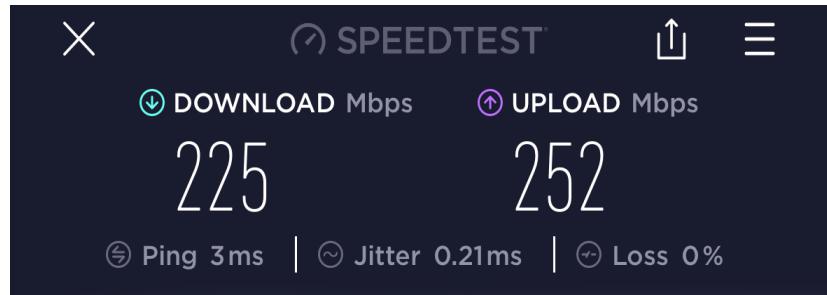


- companies, universities, etc.
- mix of wired, wireless link technologies, connecting a mix of switches and routers (we'll cover differences shortly)
  - Ethernet: wired access at 100Mbps, 1Gbps, 10Gbps
  - WiFi: wireless access points at 11, 54, 450 Mbps

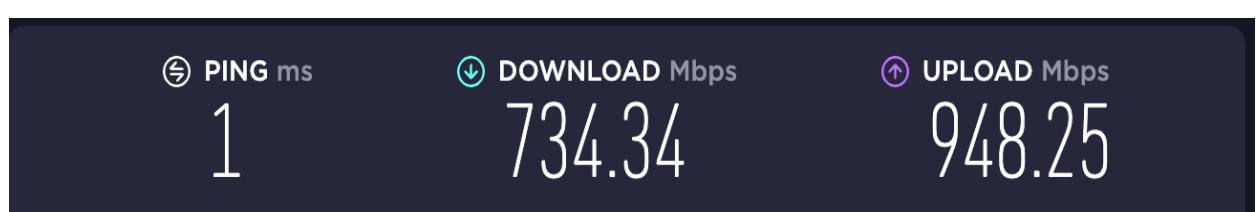
# Sample results

Can you explain the differences?

Uniwide



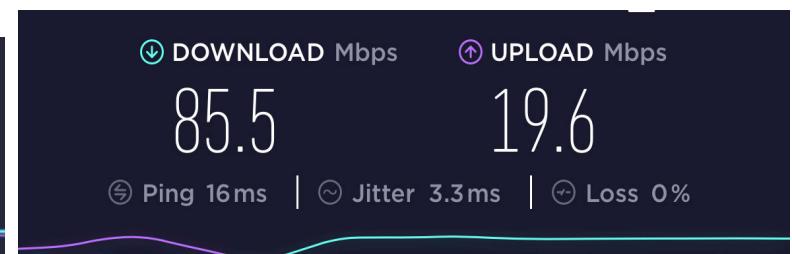
Wired Network @ CSE



FTTC + Cable + WiFi @ my home



4G Network





## Quiz: Your access network

Your residential ISP provides connectivity using the following technology:

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

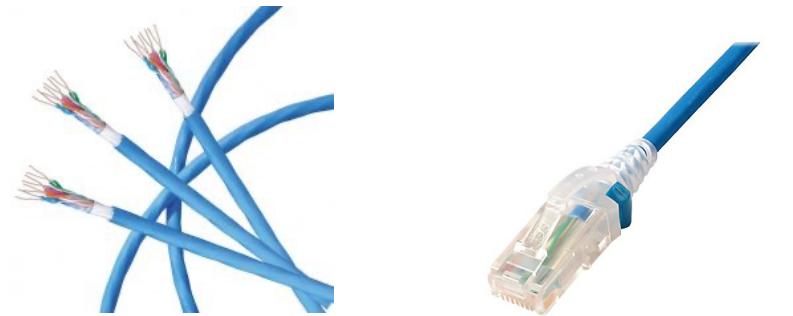
# Links: physical media

SELF STUDY  
NOT ON EXAM

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

## Twisted pair (TP)

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



# Links: physical media

## Coaxial cable:

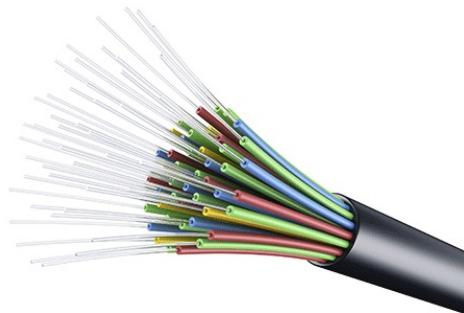
- two concentric copper conductors
- bidirectional
- broadband:
  - multiple frequency channels on cable
  - 100's Mbps per channel



SELF STUDY  
NOT ON EXAM

## 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

SELF STUDY  
NOT ON EXAM

## Wireless radio

- signal carried in electromagnetic spectrum
- no physical “wire”
- broadcast and “half-duplex” (sender to receiver)
- propagation environment effects:
  - reflection
  - obstruction by objects
  - interference

## Radio link types:

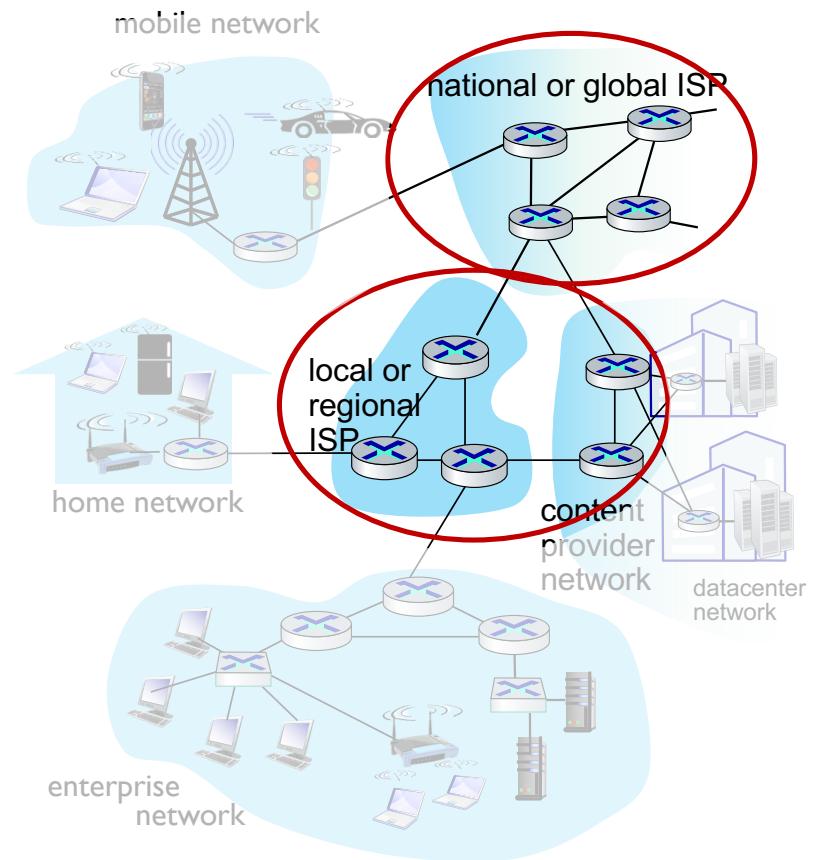
- terrestrial microwave
  - up to 45 Mbps channels
- Wireless LAN (WiFi)
  - Up to 100's Mbps
- wide-area (e.g., cellular)
  - 4G cellular: ~ 10's Mbps
- satellite
  - up to 45 Mbps per channel
  - 270 msec end-end delay
  - geosynchronous versus low-earth-orbit

# Introduction: 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

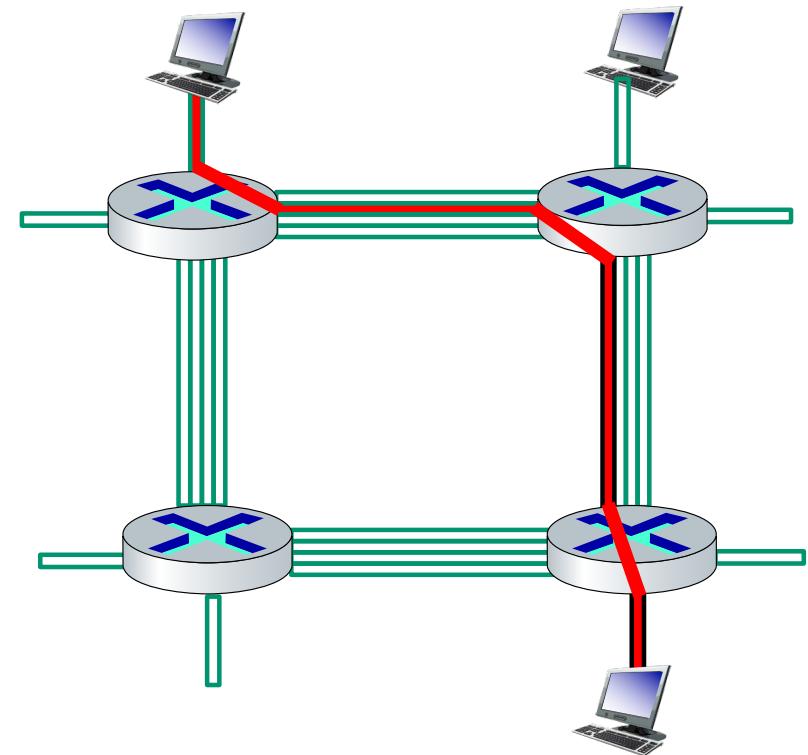
- ❖ mesh of interconnected routers
- ❖ **packet-switching:** hosts break application-layer messages into **packets**
  - forward packets from one router to the next, across links on path from source to destination
  - each packet transmitted at full link capacity
  - **Is used in the Internet**
- ❖ **circuit-switching:** an alternative used in legacy telephone networks which was considered during the design of the Internet



# 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 2<sup>nd</sup> circuit in top link and 1<sup>st</sup> circuit in right link.
- ❖ dedicated resources: no sharing
  - circuit-like (guaranteed) performance
- ❖ circuit segment idle if not used by call (no sharing)
- ❖ commonly used in traditional telephone networks



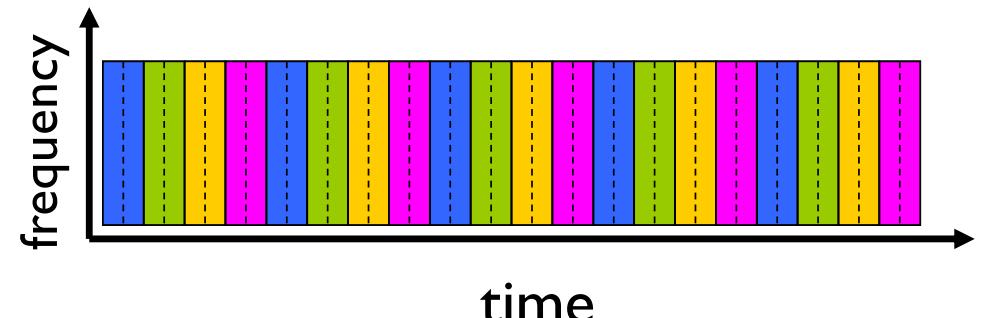
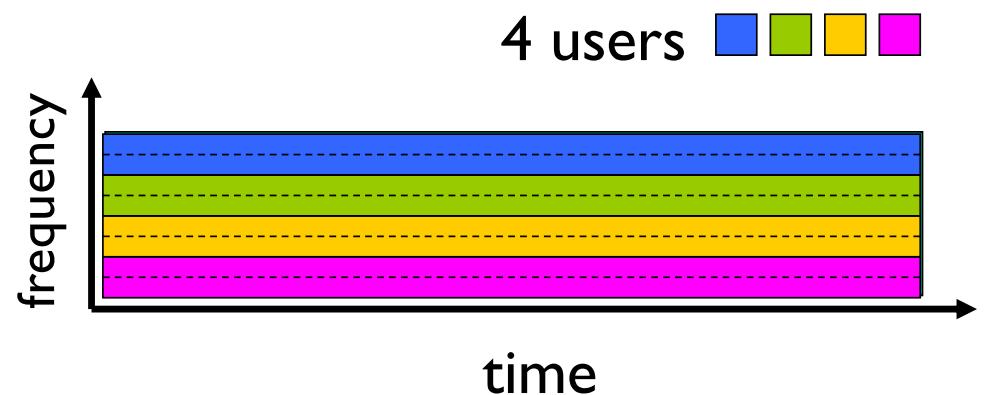
# Circuit switching: FDM and TDM

## Frequency Division Multiplexing (FDM)

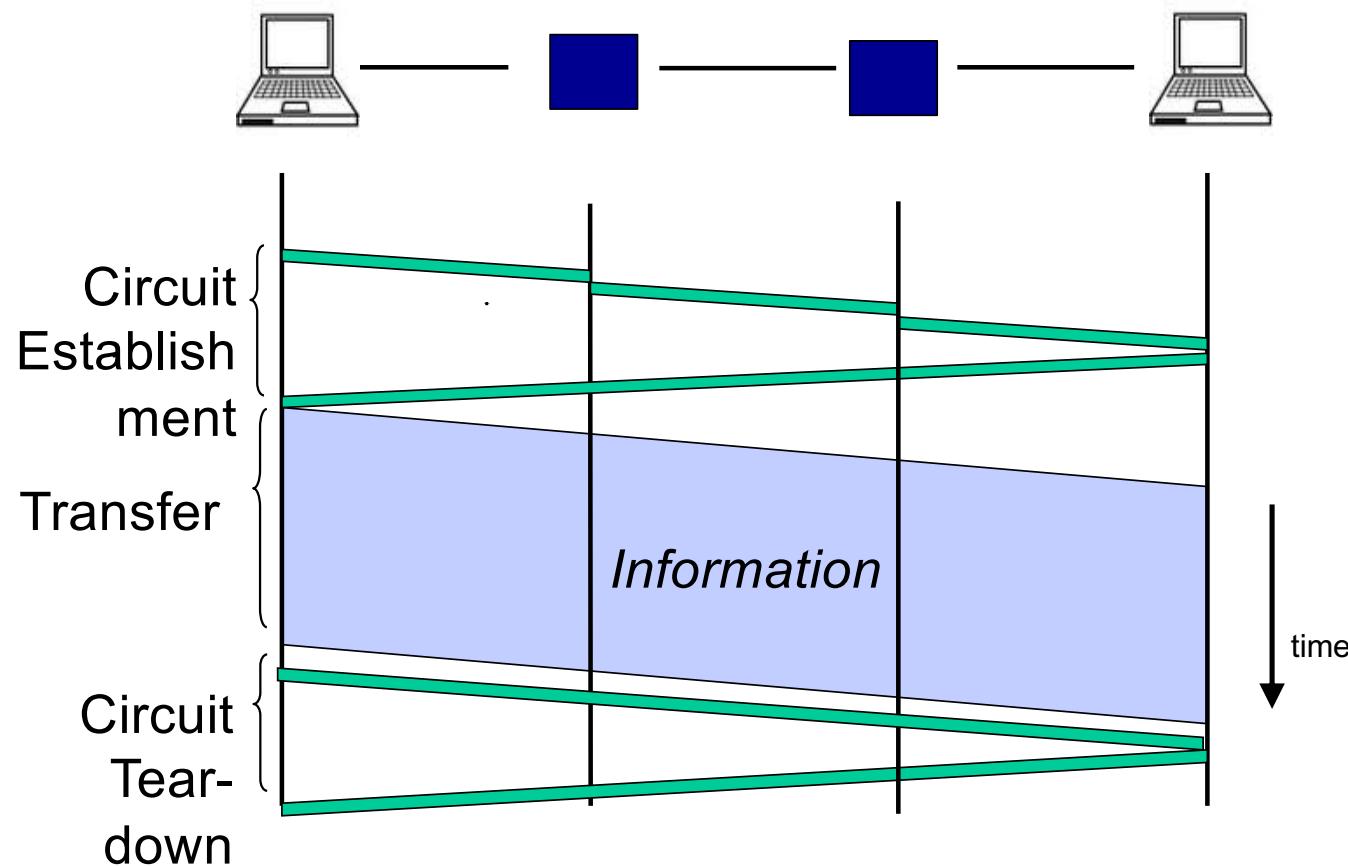
- ❖ optical, electromagnetic frequencies divided into (narrow) frequency bands
- ❖ each call allocated its own band, can transmit at max rate of that narrow band

## Time Division Multiplexing (TDM)

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



# Timing in Circuit Switching



# Why circuit switching is not feasible?

## ➤ Inefficient

- Computer communications tends to be very bursty. For example, viewing a sequence of web pages
- Dedicated circuit cannot be used or shared in periods of silence
- Cannot adapt to network dynamics

## ➤ Fixed data rate

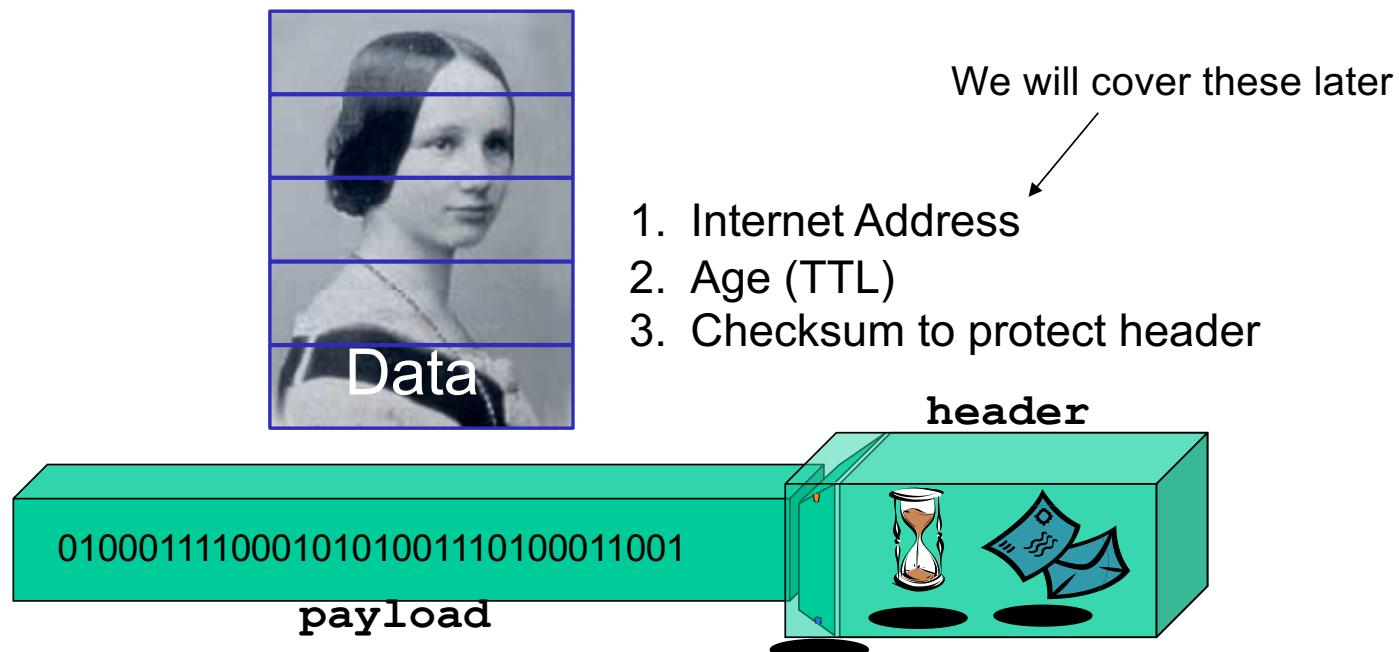
- Computers communicate at very diverse rates. For example, viewing a video vs using telnet or web browsing
- Fixed data rate is not useful

## ➤ Connection state maintenance

- Requires per communication state to be maintained that is a considerable overhead
- Not scalable

# Packet Switching

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



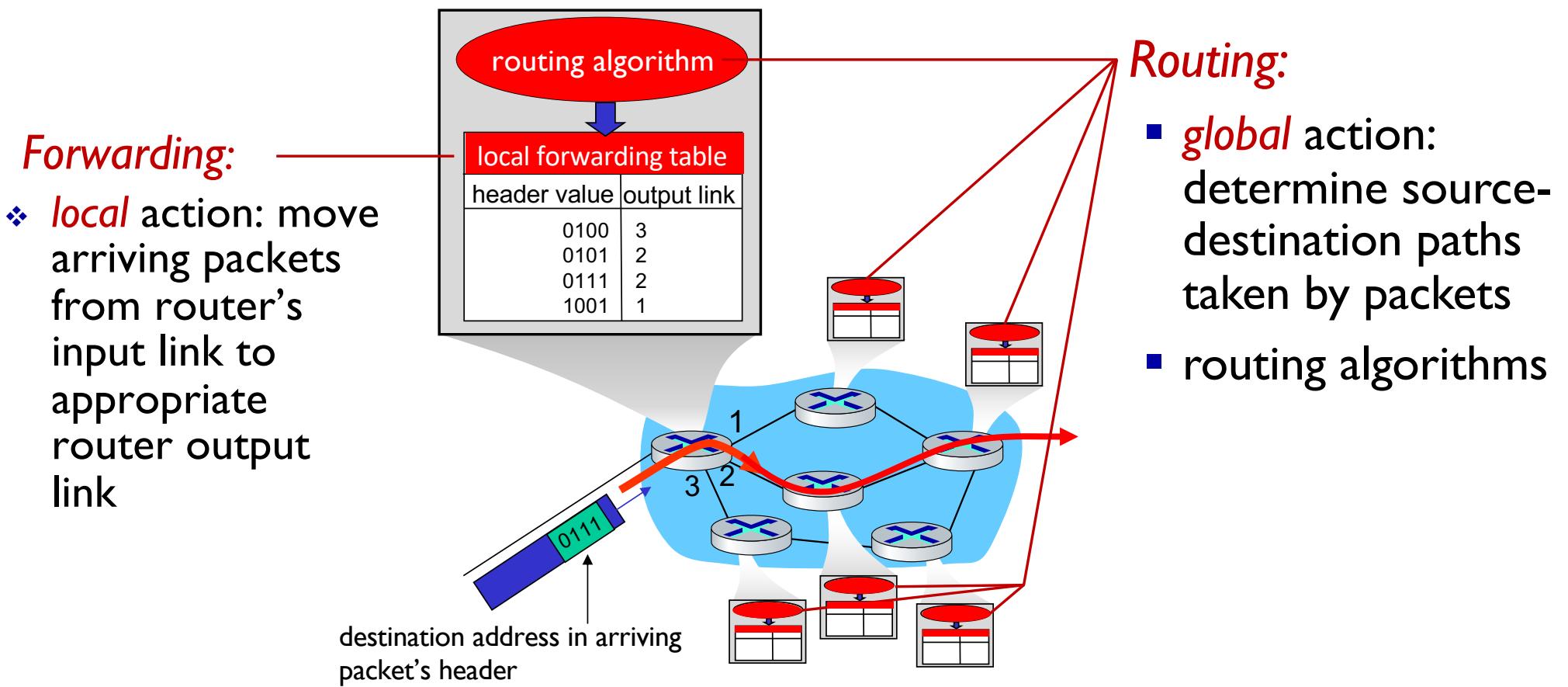
# Packet Switching

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

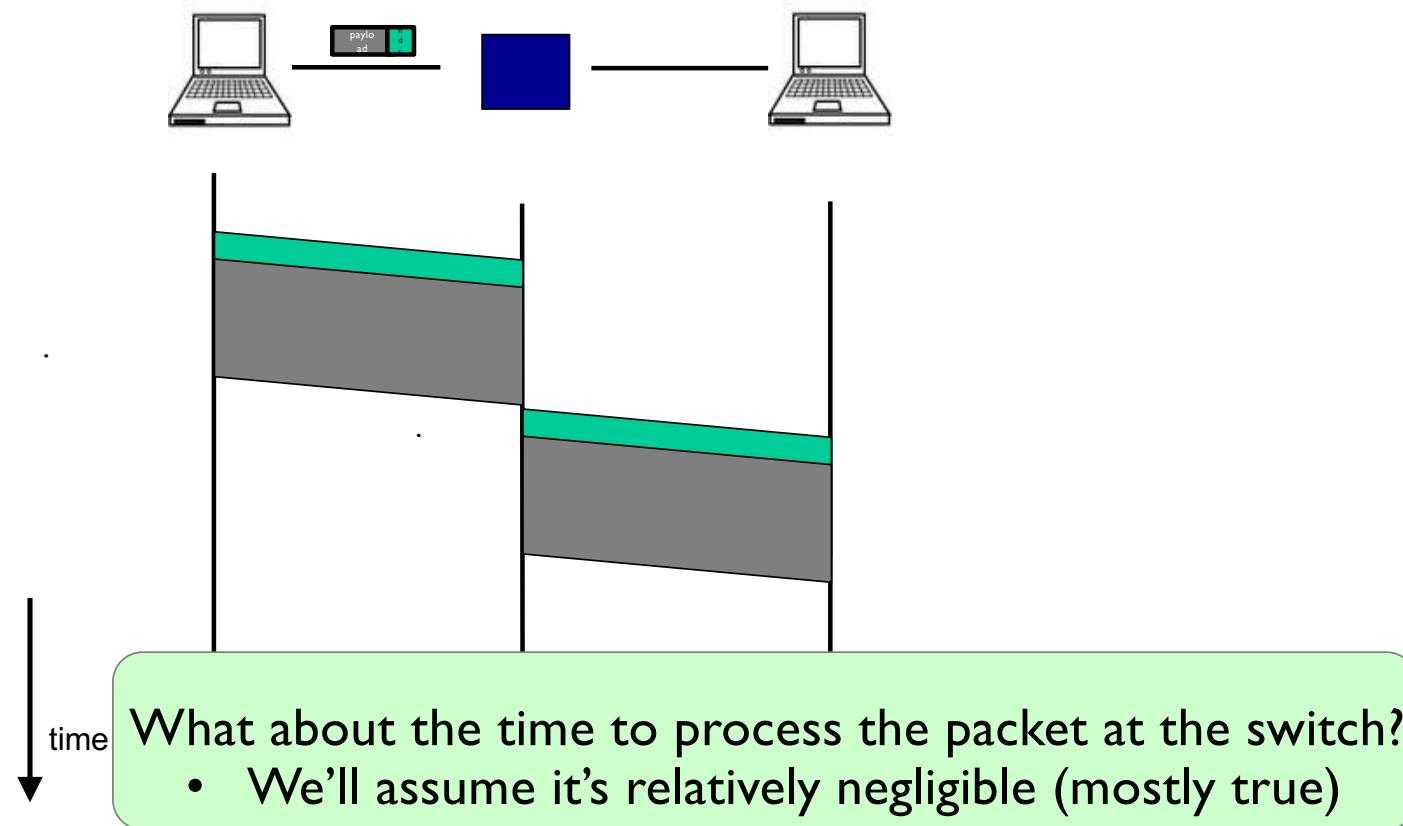
# Packet Switching

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

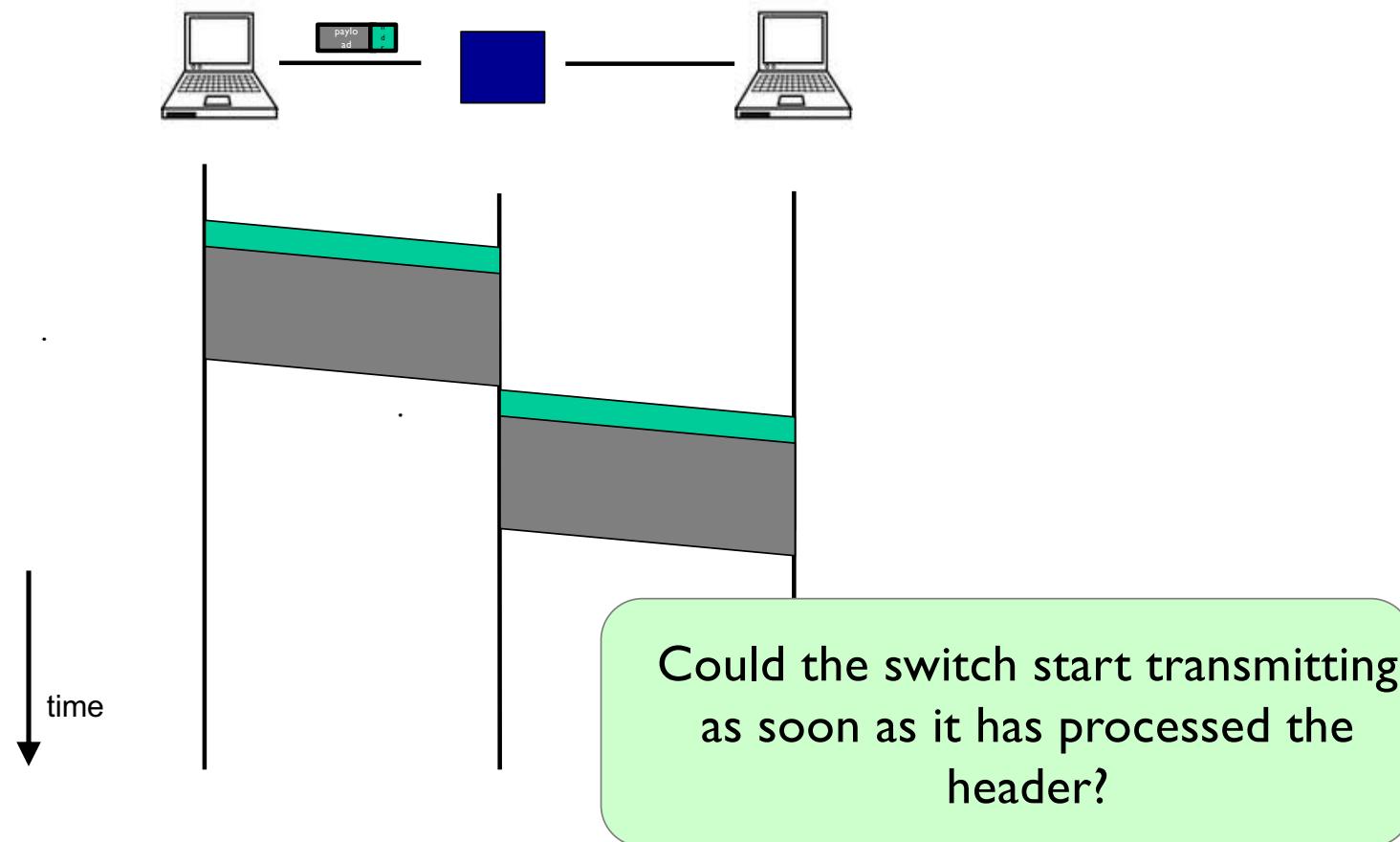
# Peek ahead: Two key network-core functions



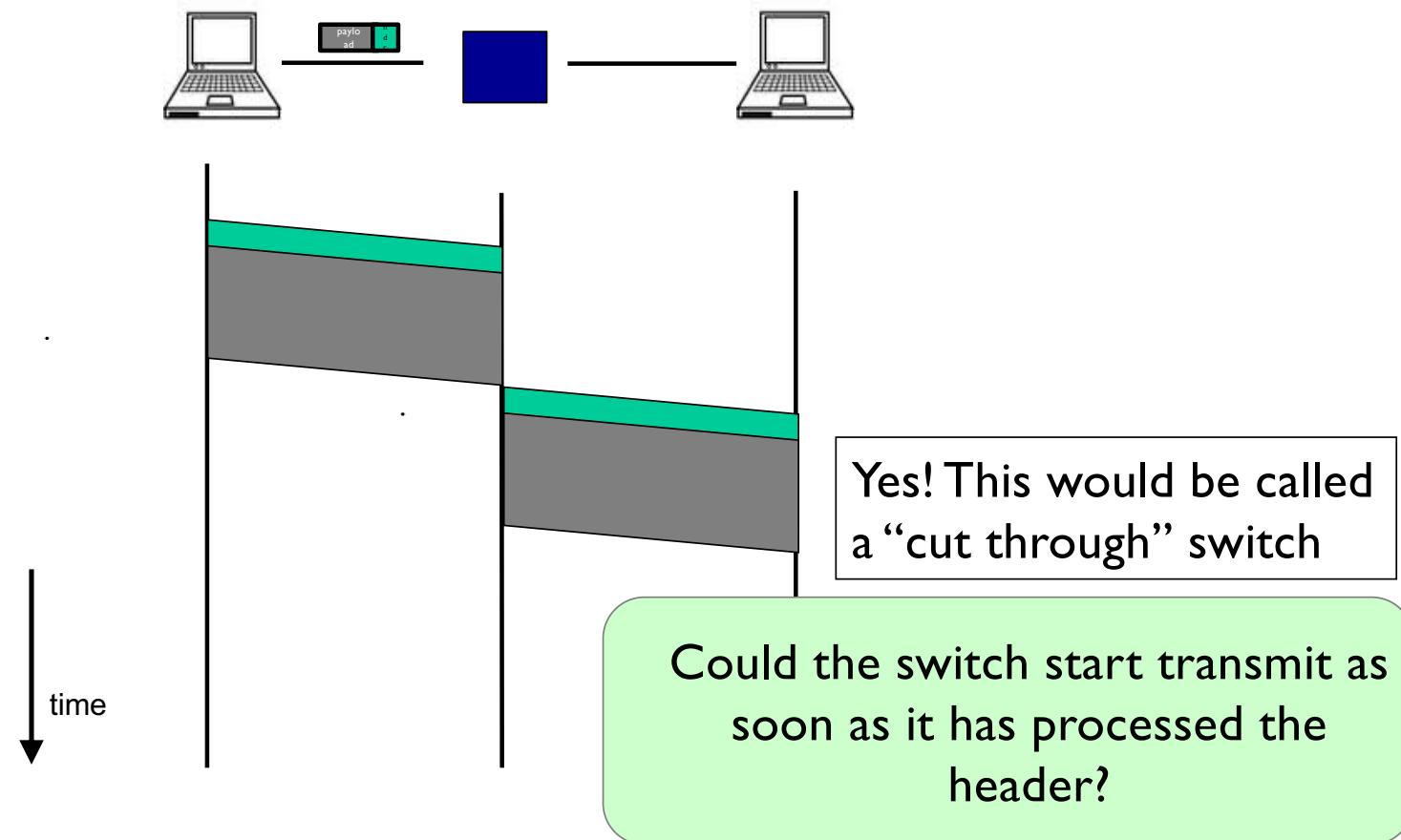
# Timing in Packet Switching



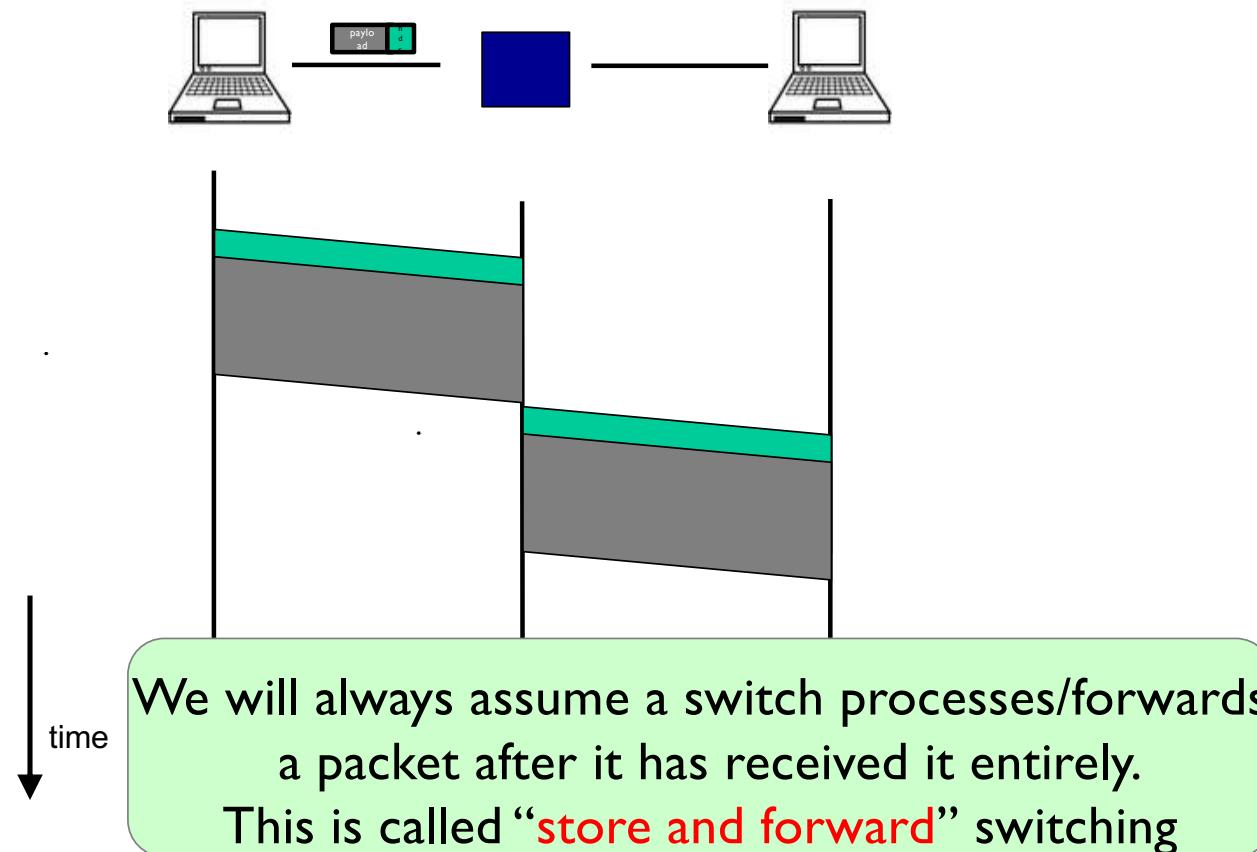
# Timing in Packet Switching



# Timing in Packet Switching



# Timing in Packet Switching



# Packet Switching

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

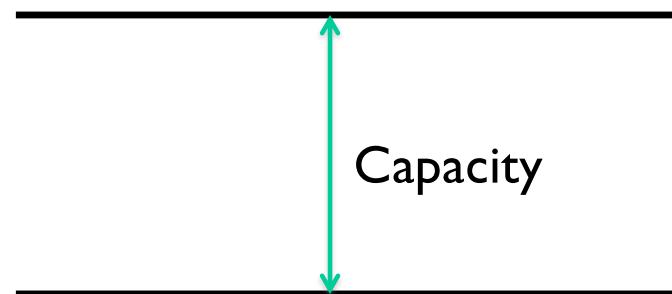
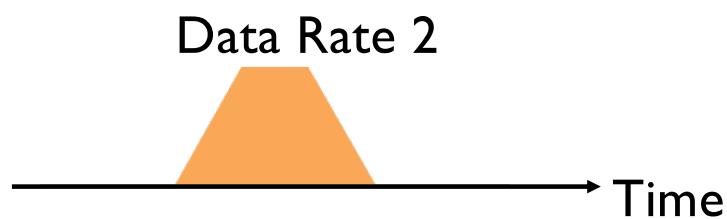
# Packet Switching

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

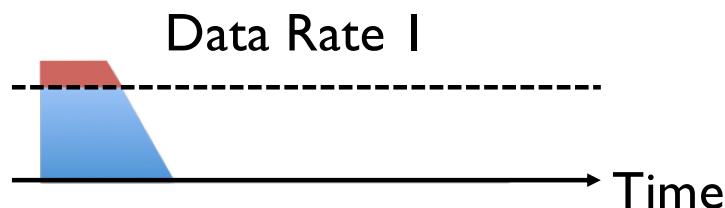
# Packet Switching

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

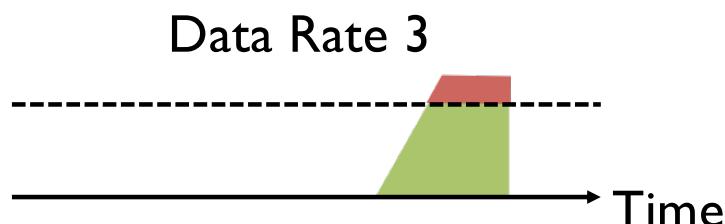
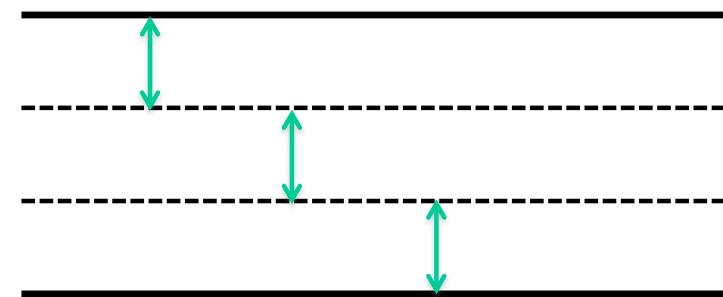
# Three Flows with Bursty Traffic



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



like circuit switching



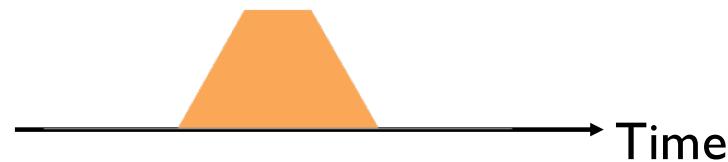
Overloaded

# When Flows Share Total Capacity

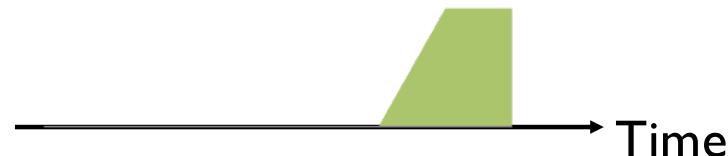


**packet switching**

No Overloading

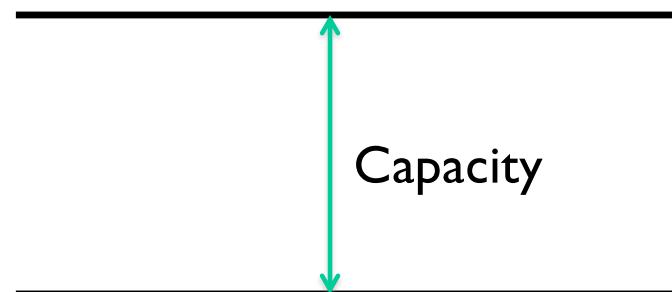
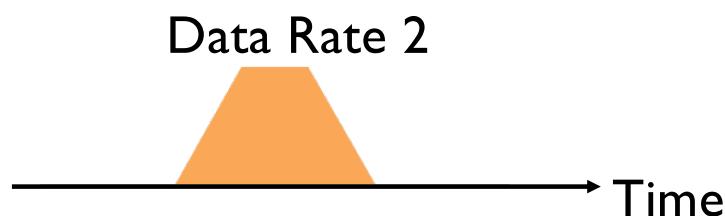
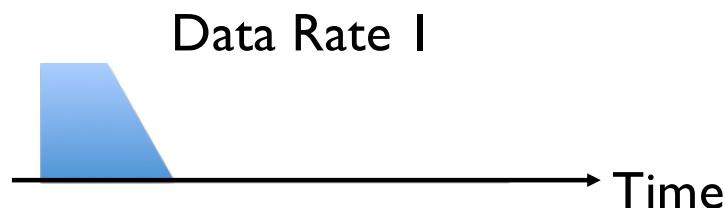


Statistical multiplexing relies on the assumption that not all flows burst at the same time

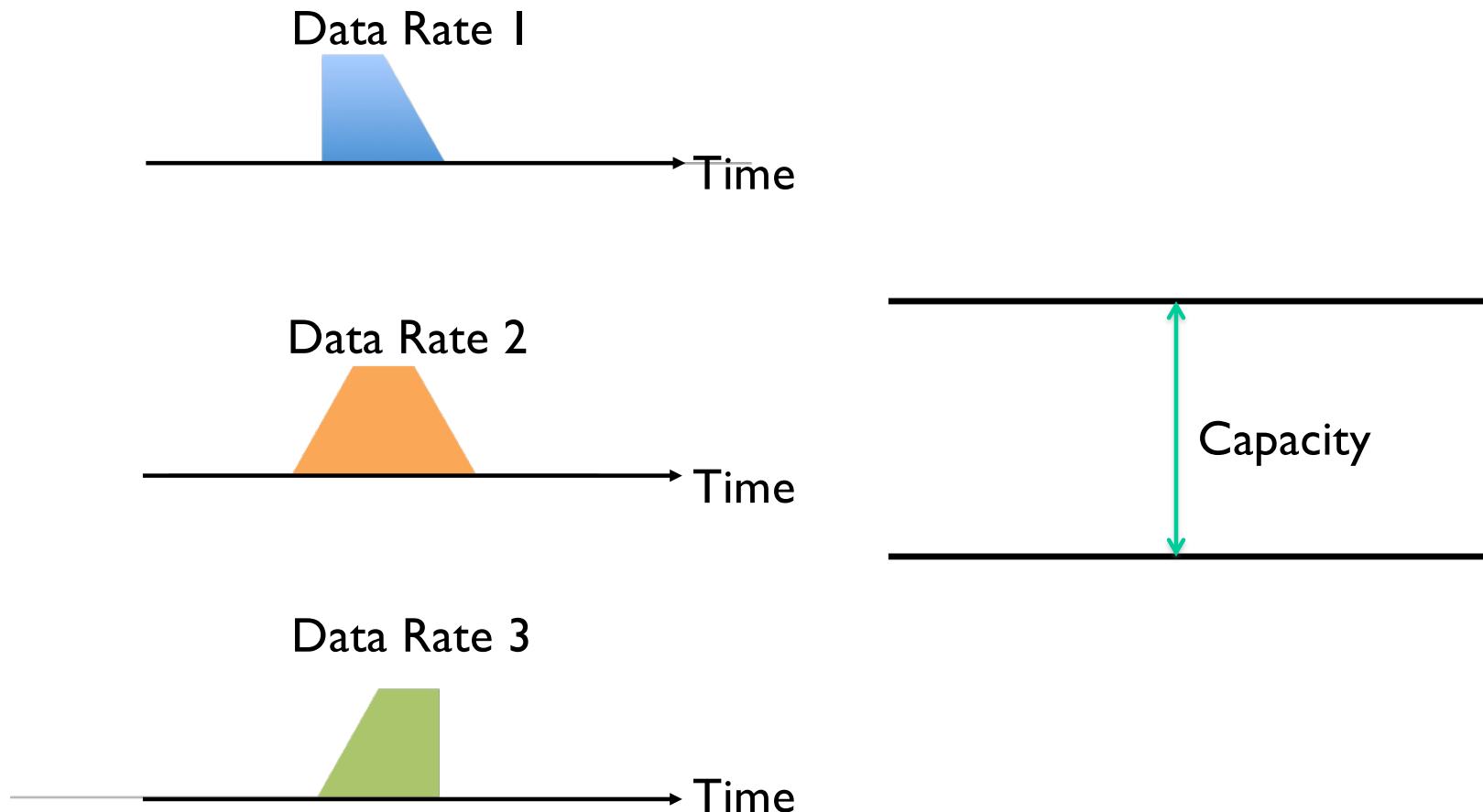


Very similar to insurance, and has same failure case

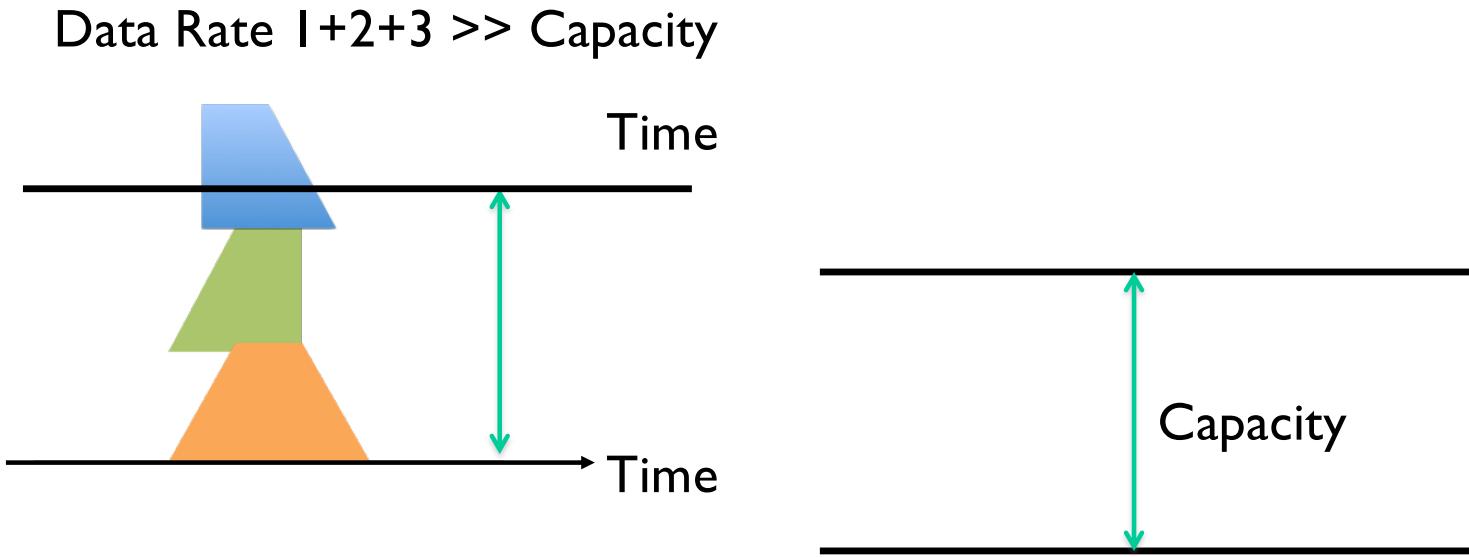
# Three Flows with Bursty Traffic



# Three Flows with Bursty Traffic

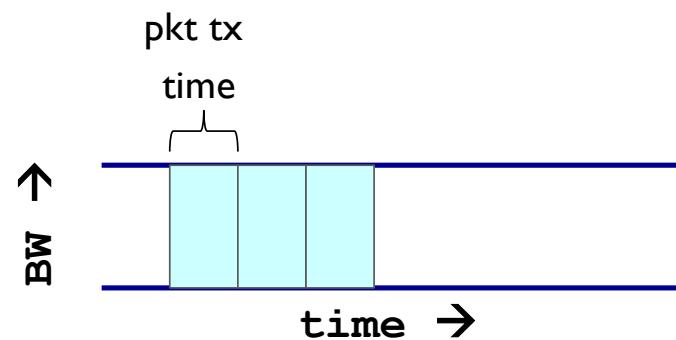


# Three Flows with Bursty Traffic

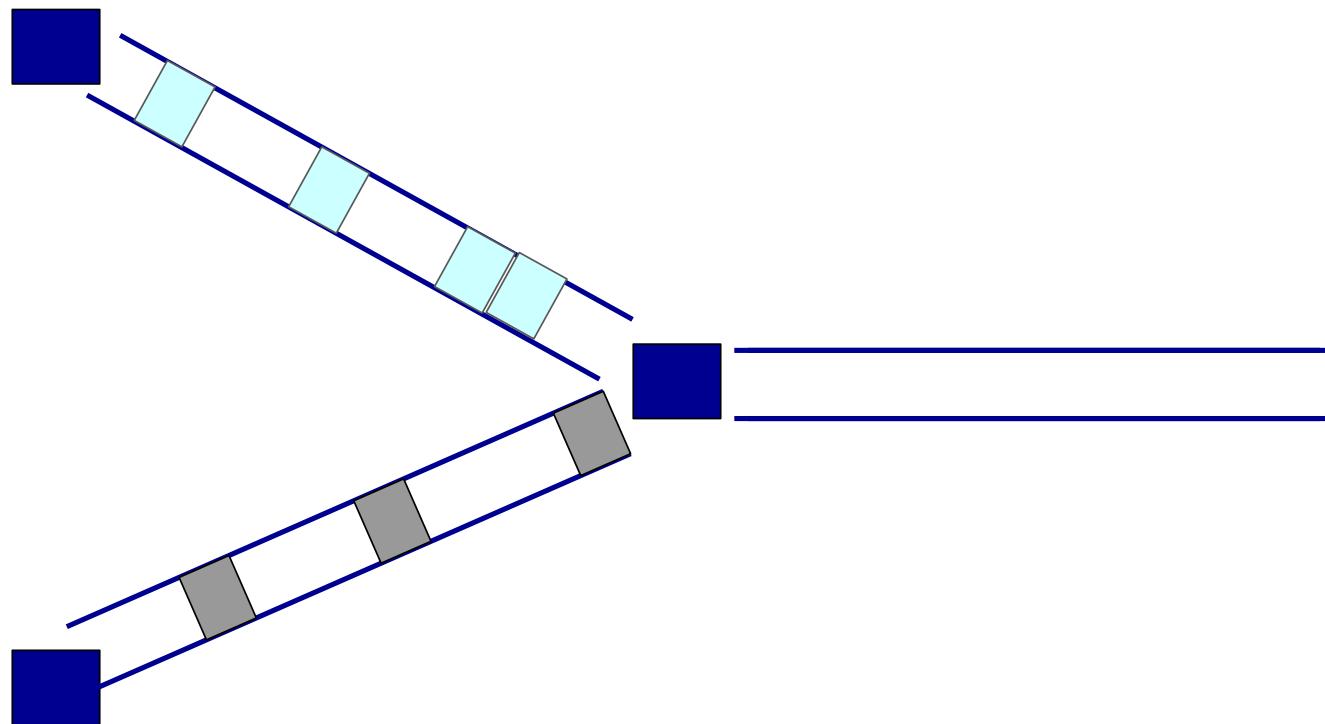


What do we do under overload?

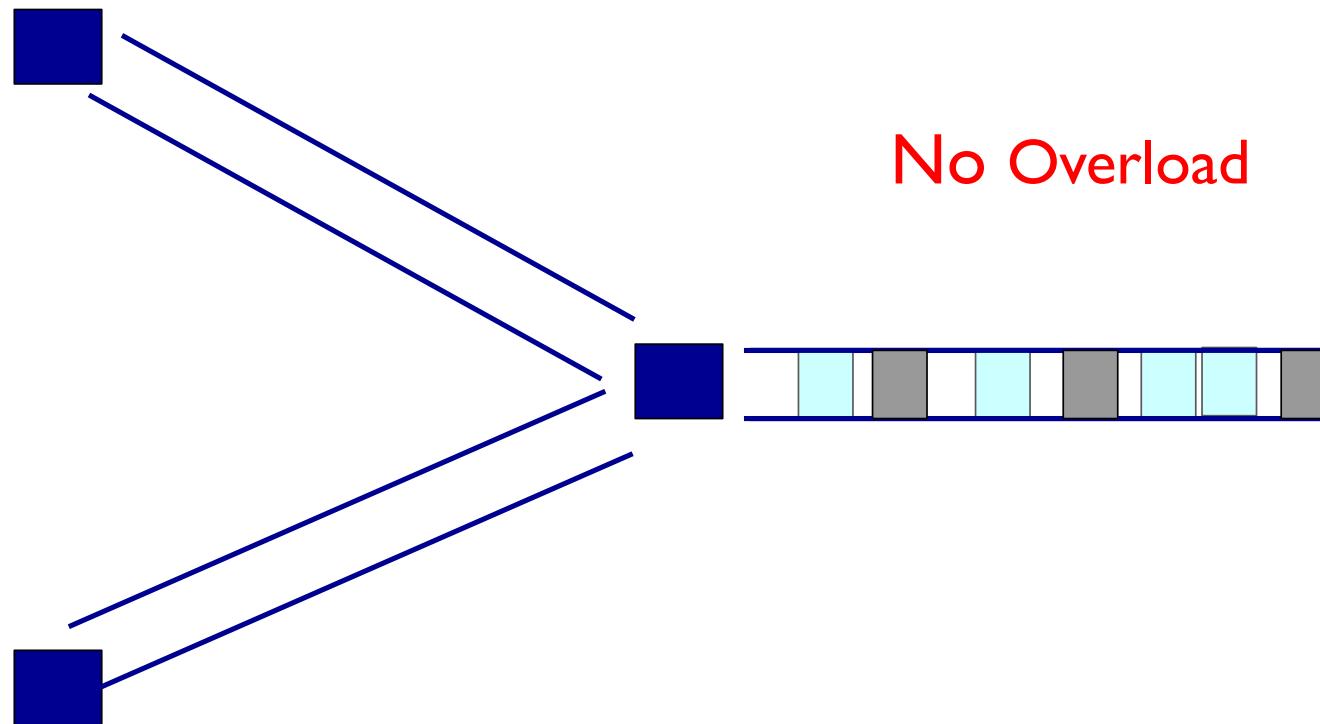
## Statistical multiplexing: pipe view



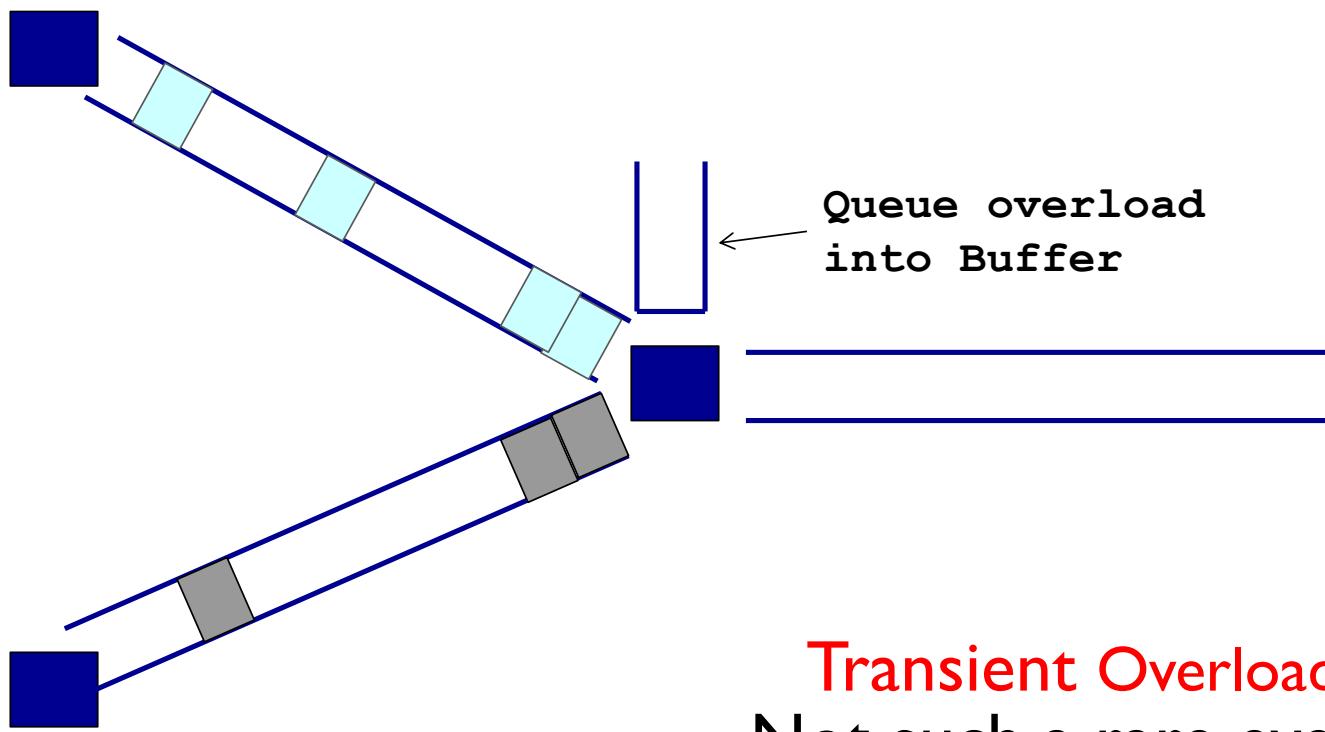
## Statistical multiplexing: pipe view



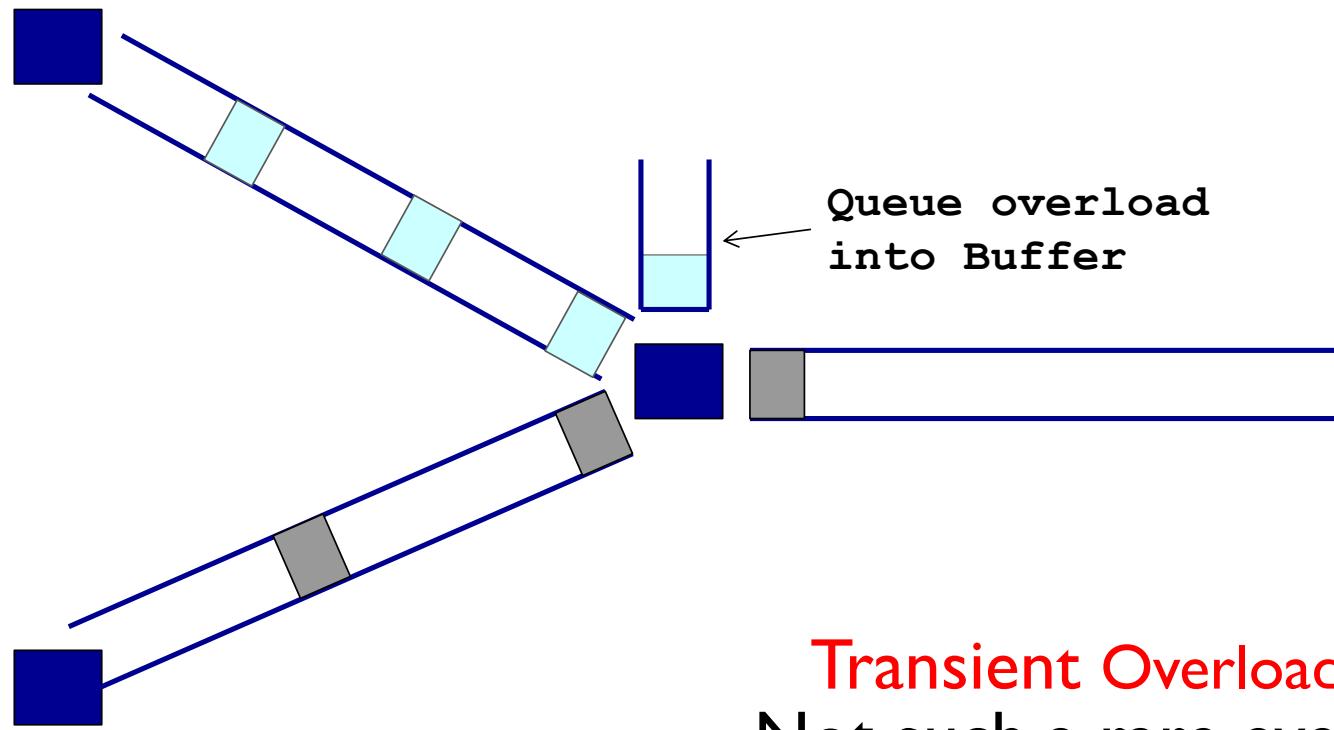
## Statistical multiplexing: pipe view



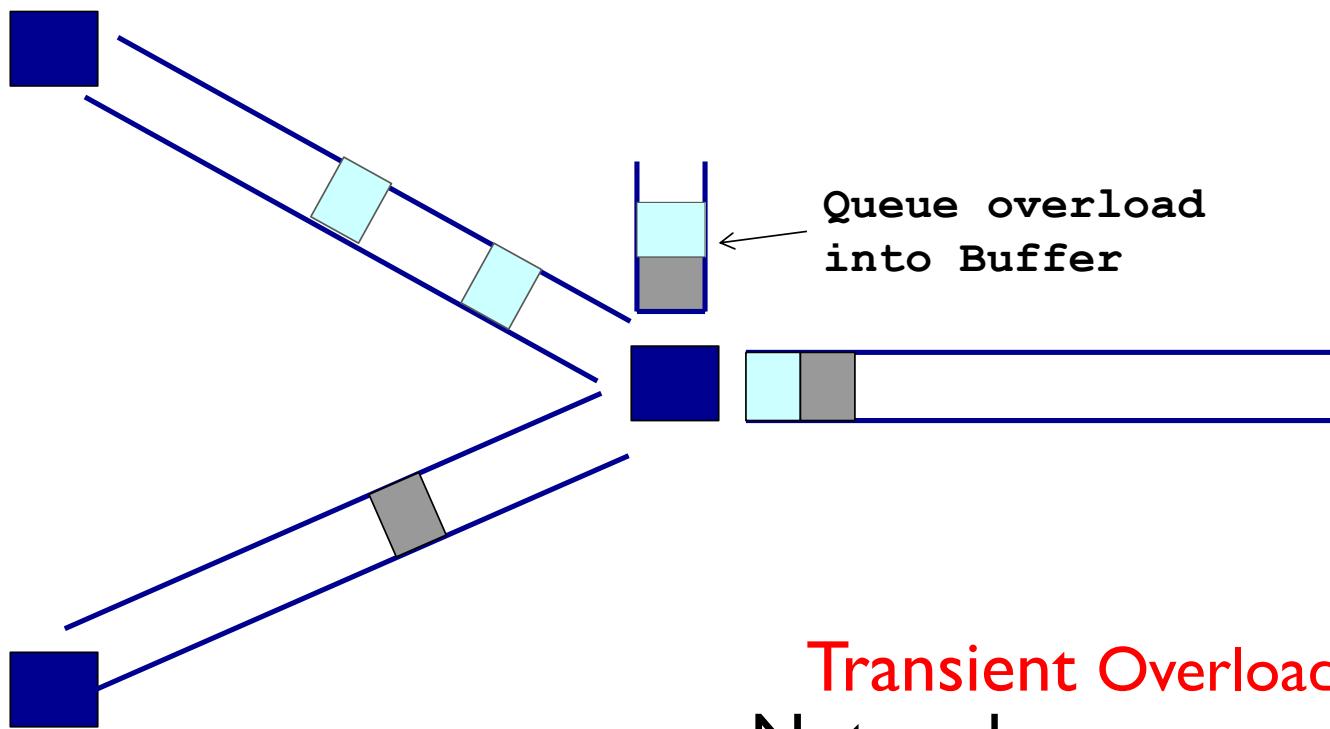
## Statistical multiplexing: pipe view



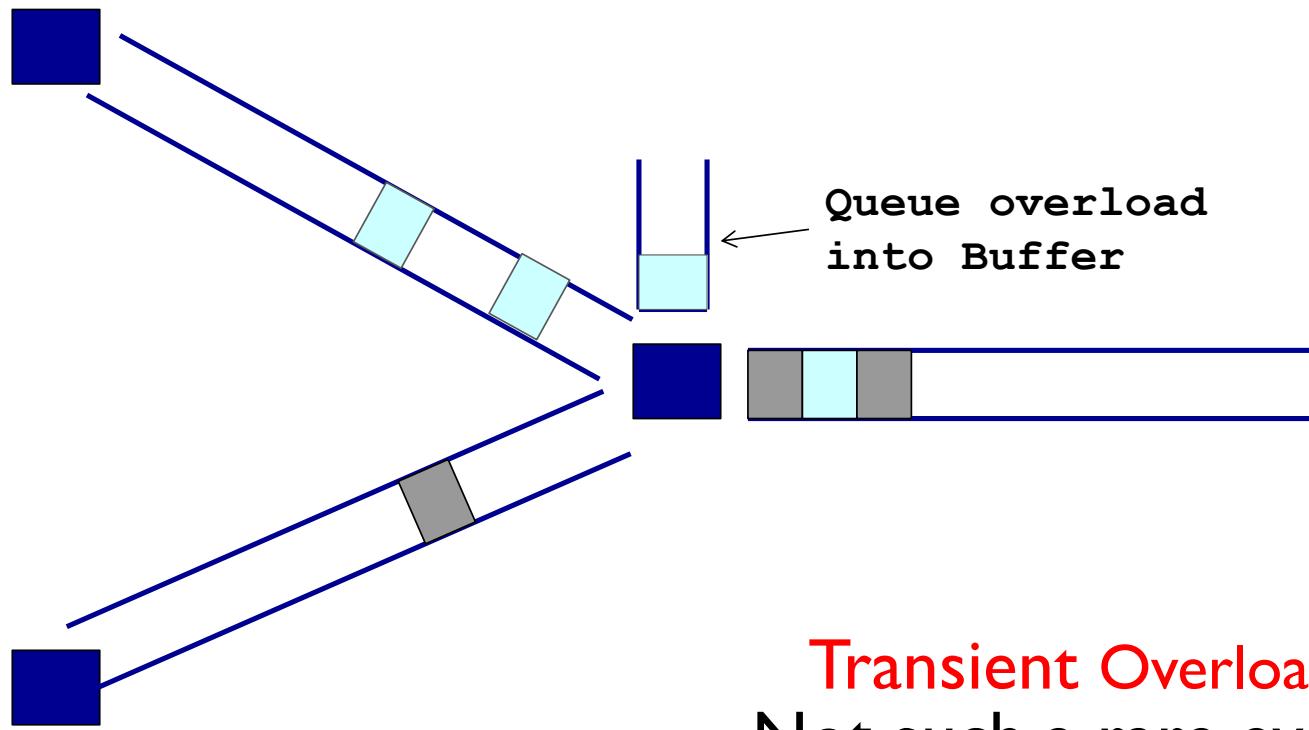
# Statistical multiplexing: pipe view



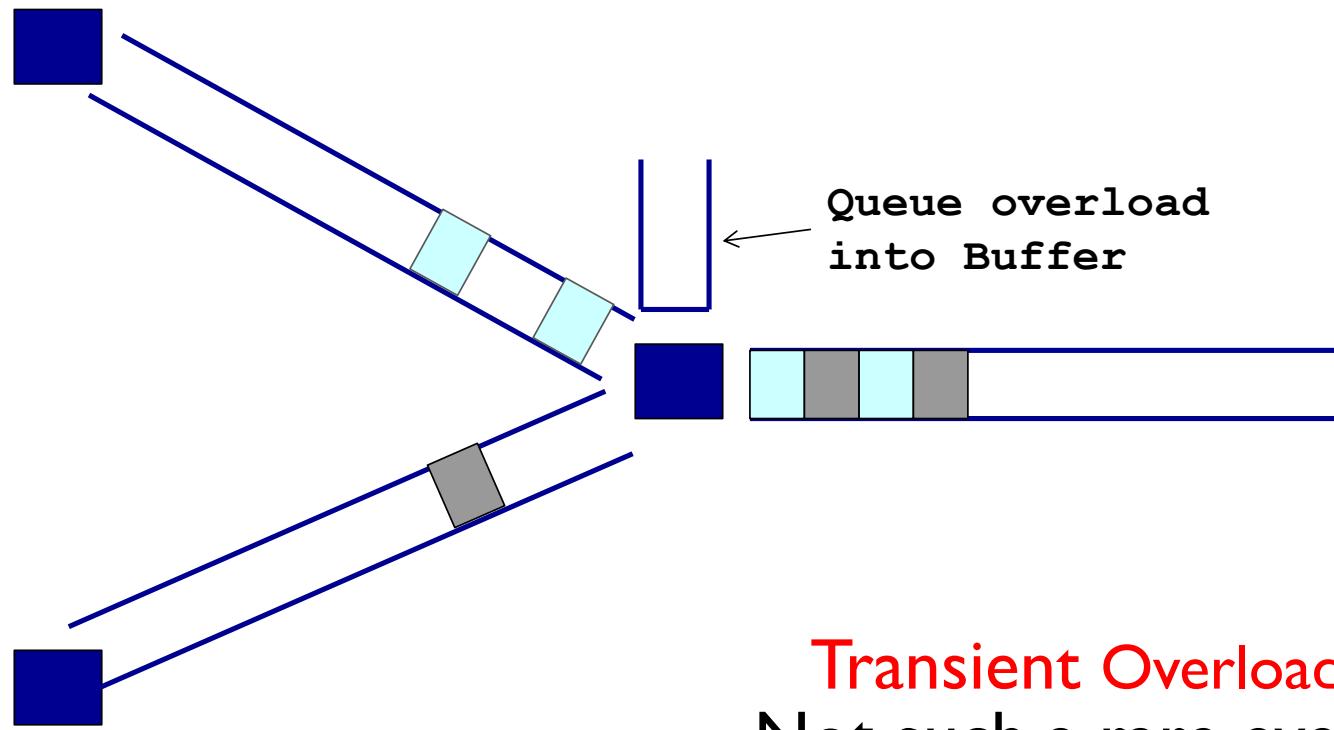
# Statistical multiplexing: pipe view



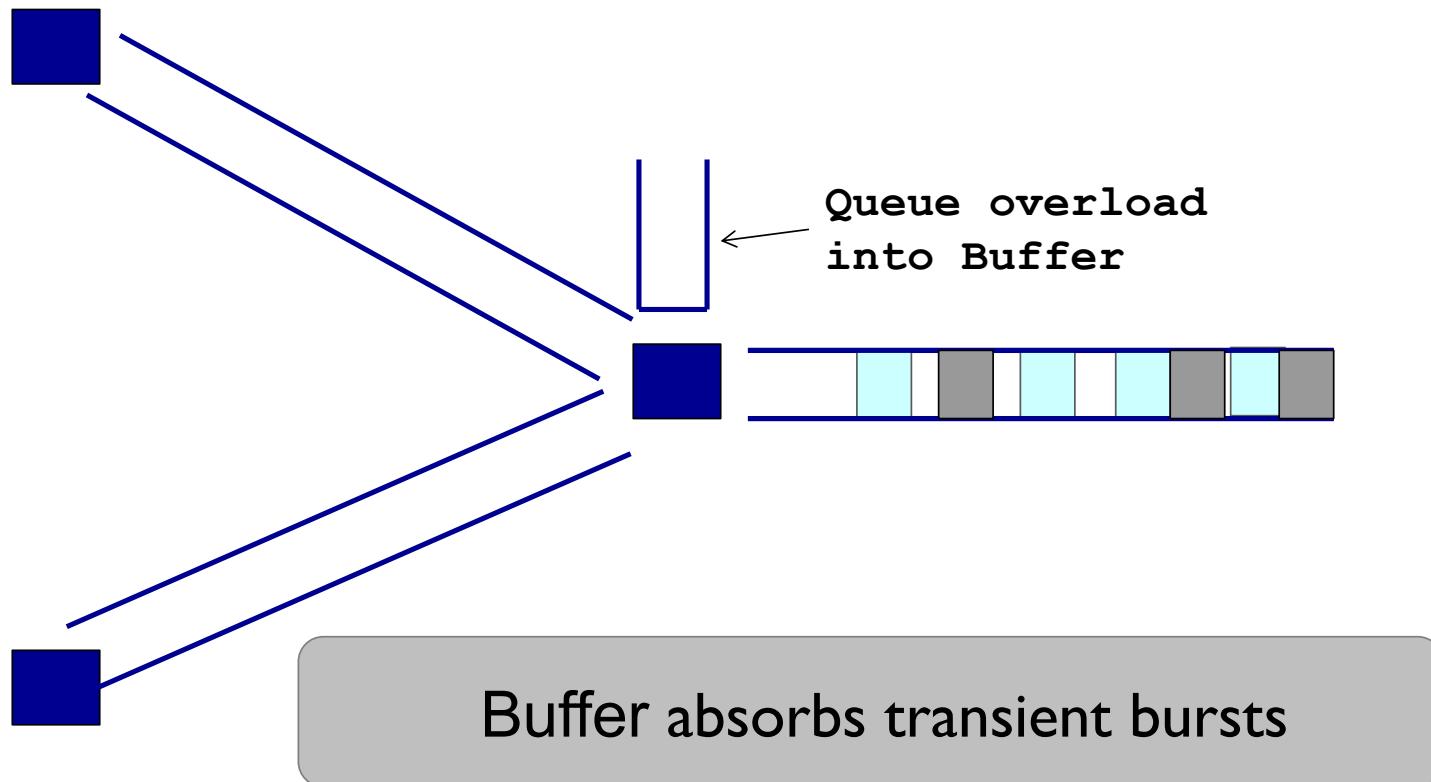
# Statistical multiplexing: pipe view



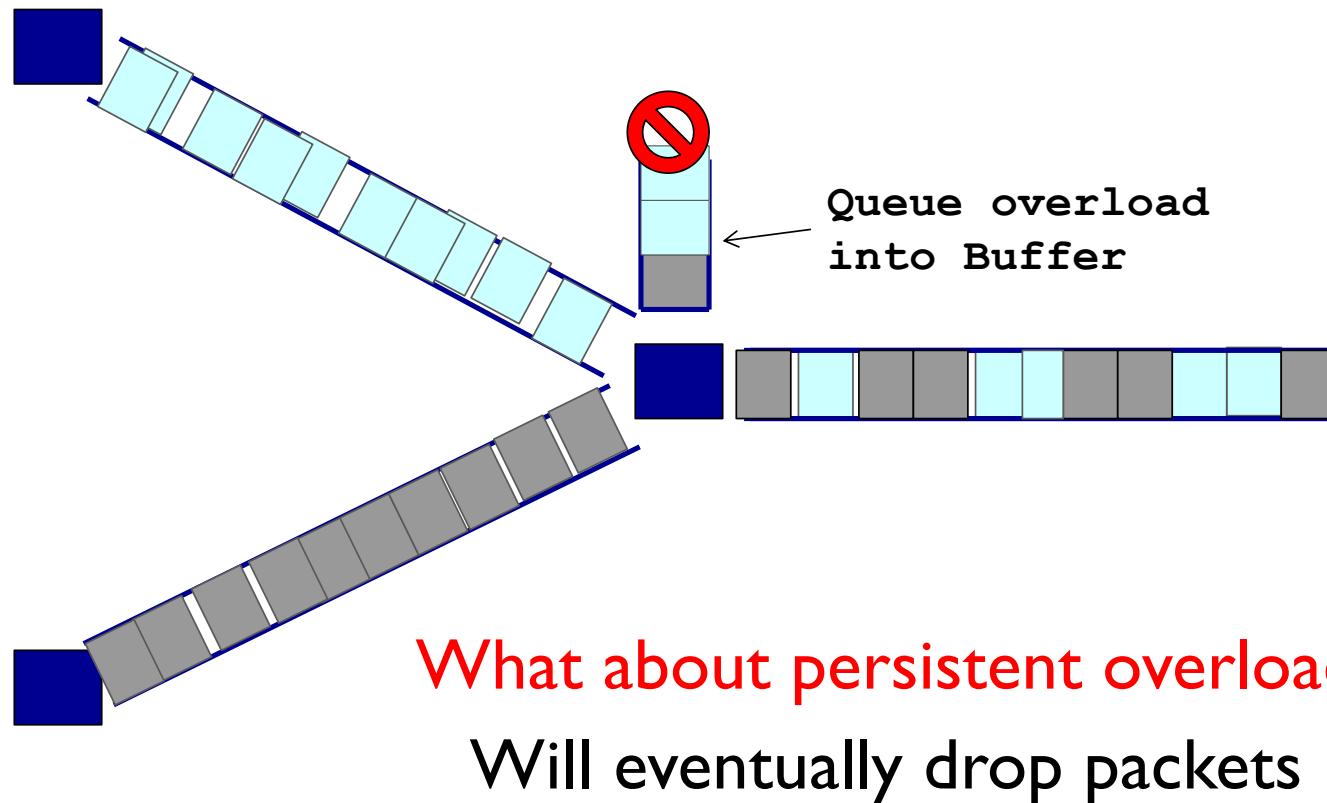
# Statistical multiplexing: pipe view



## Statistical multiplexing: pipe view



# Statistical multiplexing: pipe view



# Packet switching versus circuit switching

*packet switching allows more users to use network!*

example:

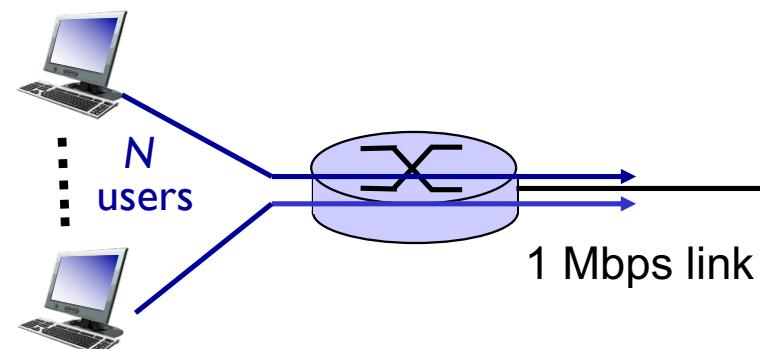
- 1 Mb/s link
- each user:
  - 100 kb/s when “active”
  - active 10% of time

❖ *circuit-switching:*

- 10 users

❖ *packet switching:*

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



*Q:* how did we get value 0.0004?

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

**Hint: Bernoulli Trials and Binomial Distribution**

# Binomial Probability Distribution

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

# Binomial Distribution: Example

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

# Binomial Distribution: Example

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

$\binom{5}{3}$  ways to arrange 3 heads in 5 trials

${}^5C_3 = 5!/3!2! = 10$

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

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

# Binomial Distribution

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

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

$n$  = number of trials  
 $X$  = # successes out of  $n$  trials  
 $p$  = probability of success  
 $1-p$  = probability of failure

## Packet switching versus circuit switching

- ❖ Let's revisit the earlier problem
- ❖  $N = 35$  users
- ❖  $\text{Prob } (\# \text{ active users} > 10) = 1 - \underbrace{\text{Prob } (\# \text{ active} = 10)}_{\begin{aligned} &- \text{Prob } (\# \text{ active} = 9) \\ &- \text{Prob } (\# \text{ active} = 8) \\ &\dots \\ &- \text{Prob } (\# \text{ active} = 0) \end{aligned}}$

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

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

# Packet switching versus circuit switching

Is packet switching a “slam dunk winner”?

- great for “bursty” data – 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
- **Q: How to provide circuit-like behavior?**
  - bandwidth guarantees traditionally used for audio/video applications

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



## Quiz: Switching-1

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

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



## Quiz: Switching-2

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

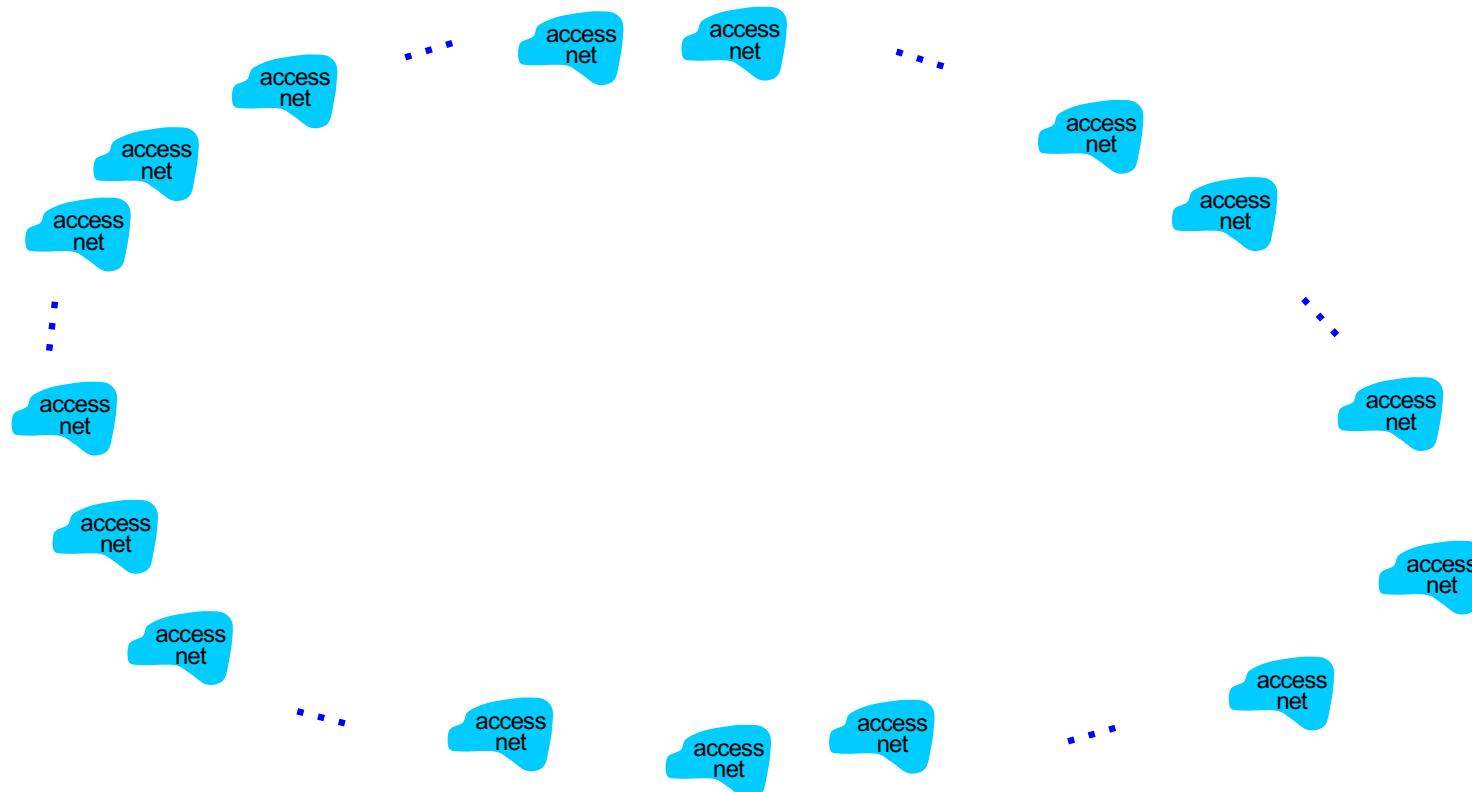
- \_\_\_\_\_
- A. is the same
- B. is independent
- C. is always different from

# Internet structure: a “network of networks”

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

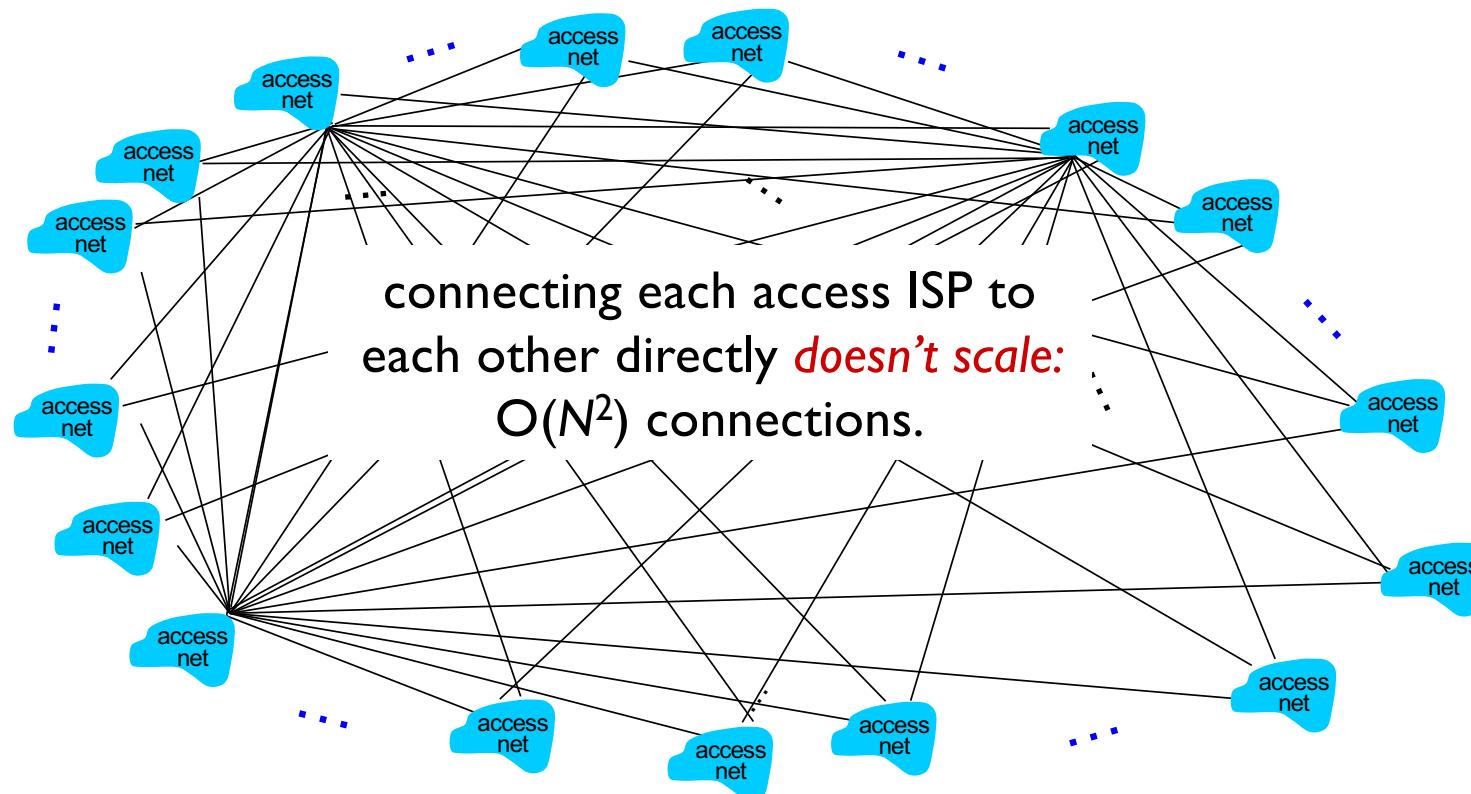
# Internet structure: a “network of networks”

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



# Internet structure: a “network of networks”

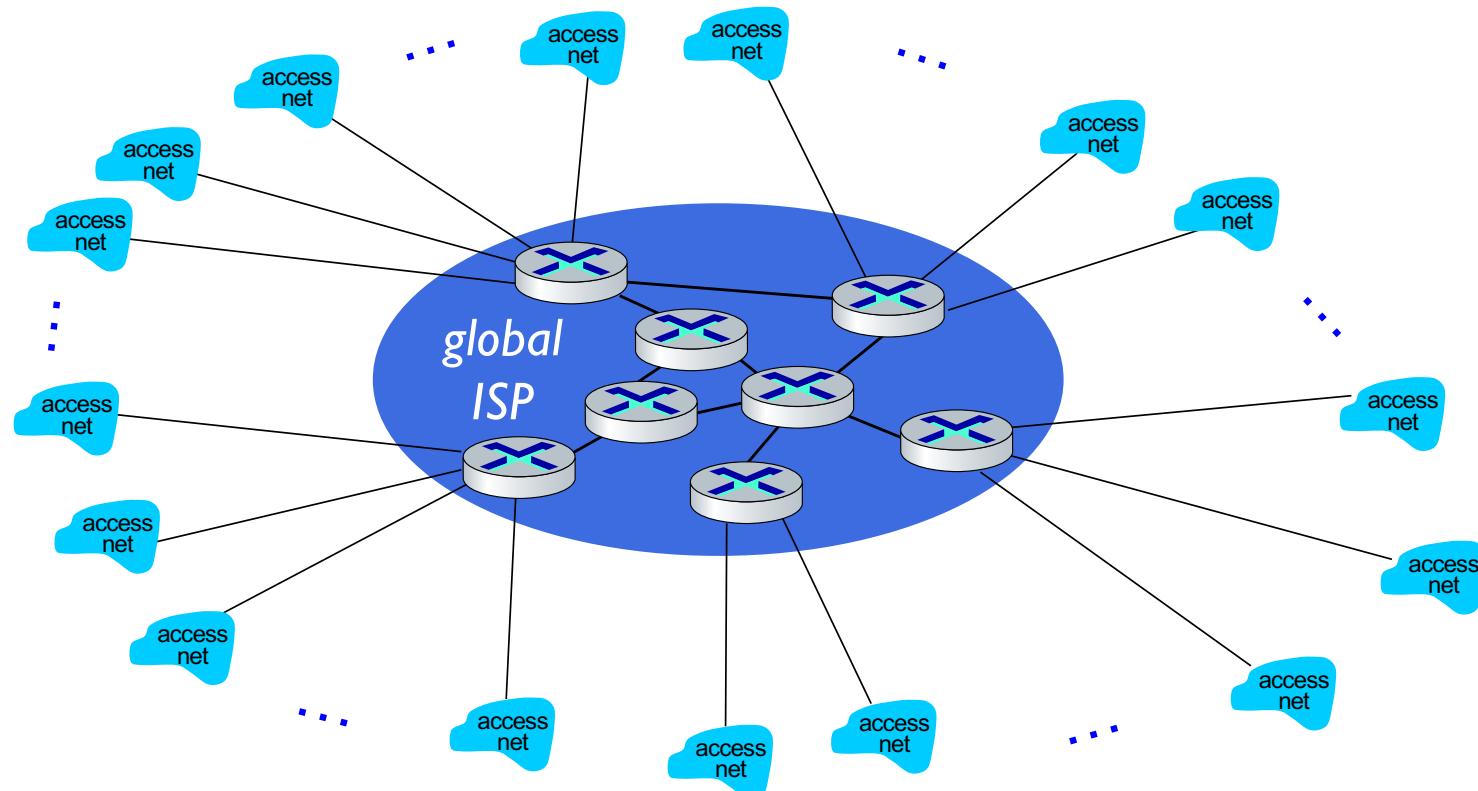
**Question:** given *millions* of access ISPs, how to connect them together?



# Internet structure: a “network of networks”

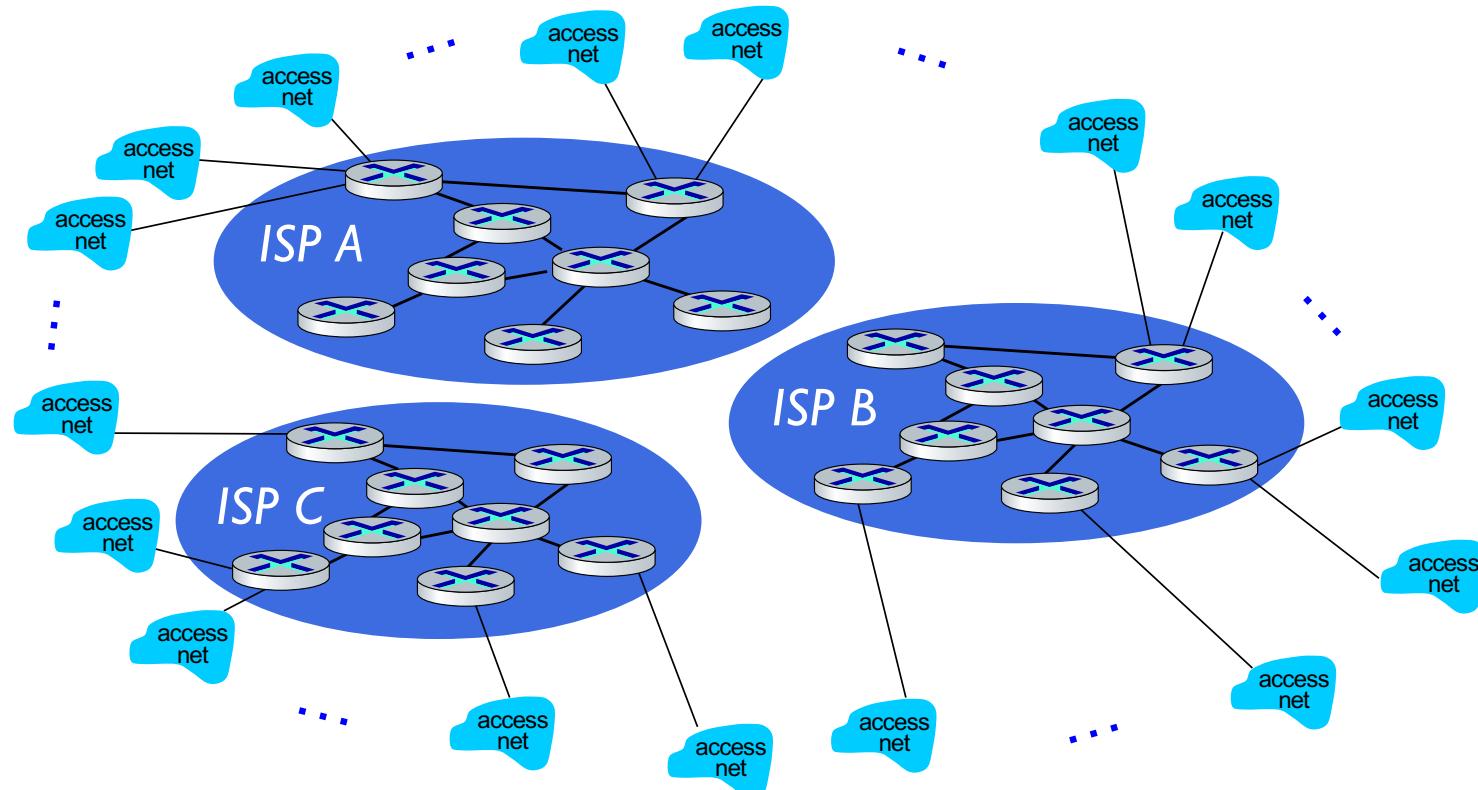
*Option:* connect each access ISP to one global transit ISP?

*Customer and provider ISPs have economic agreement.*



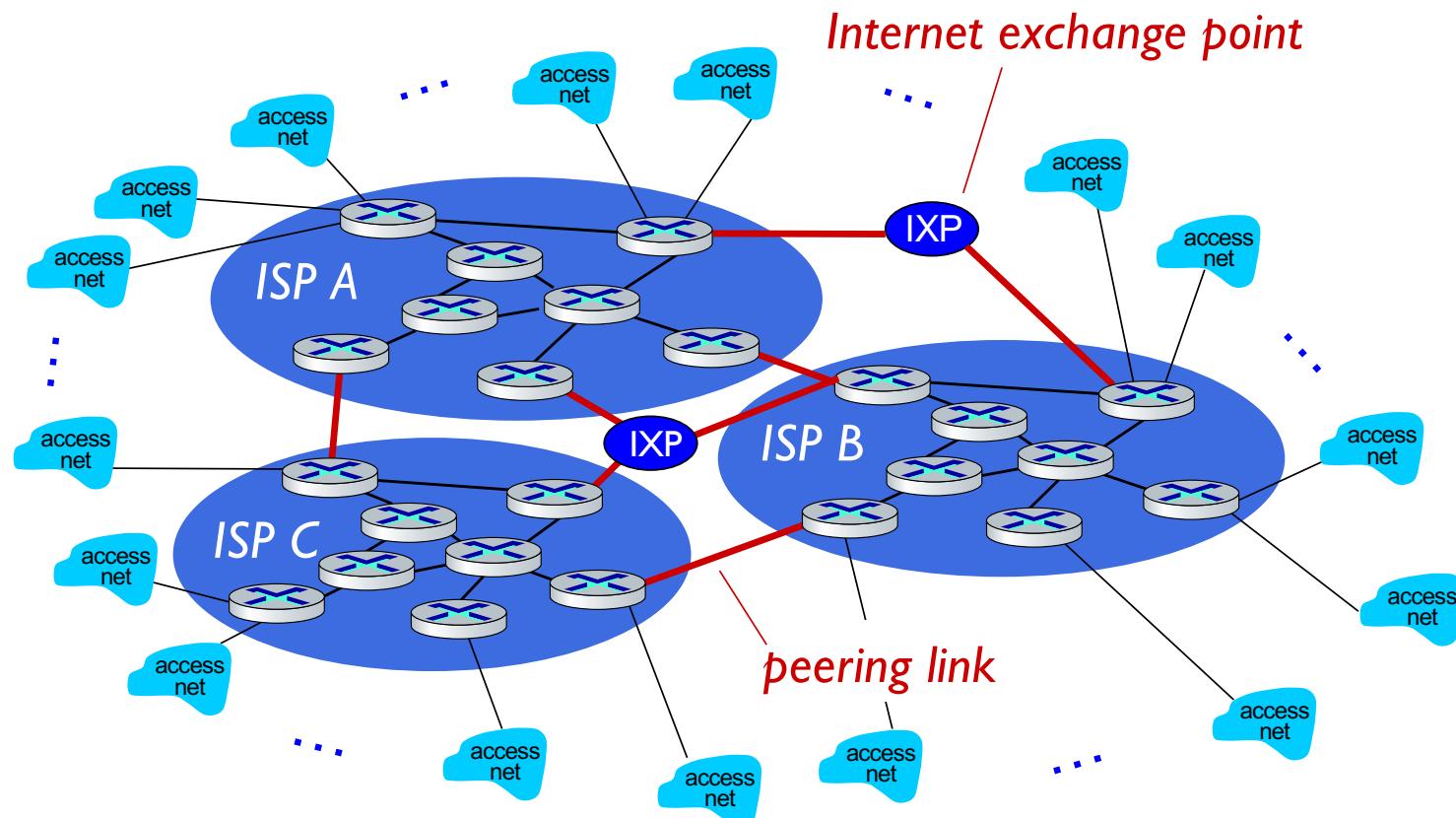
# Internet structure: a “network of networks”

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



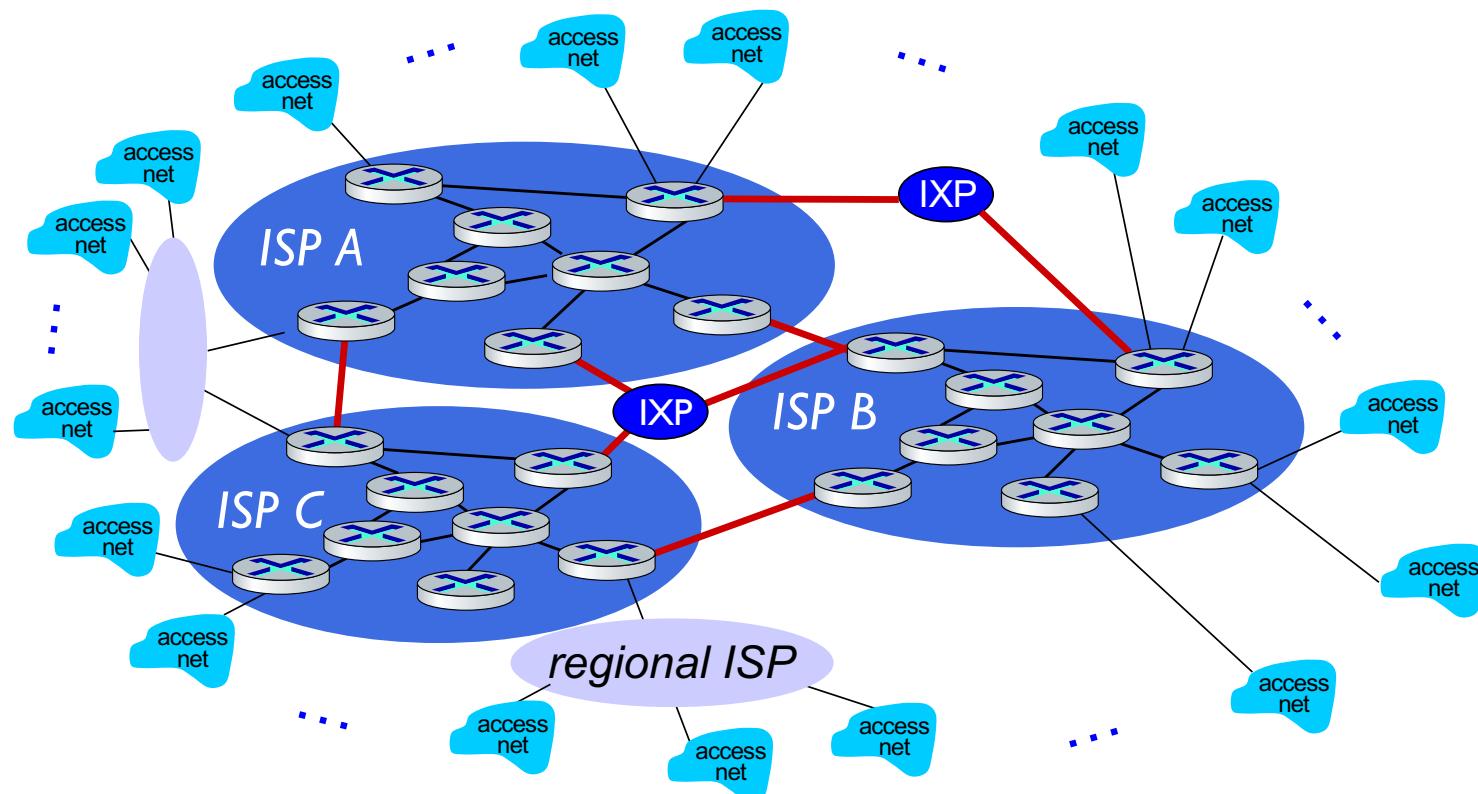
# Internet structure: a “network of networks”

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



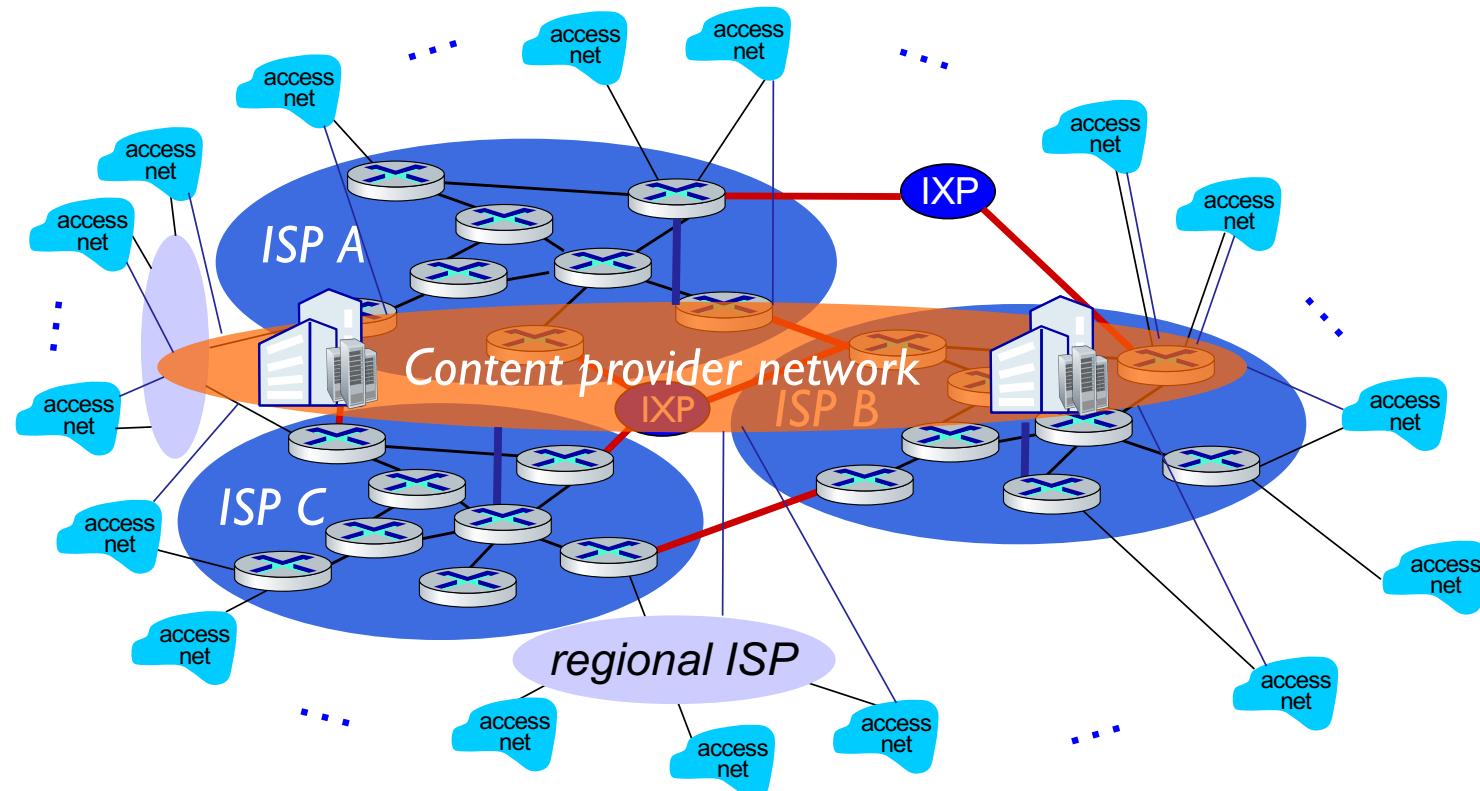
# Internet structure: a “network of networks”

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

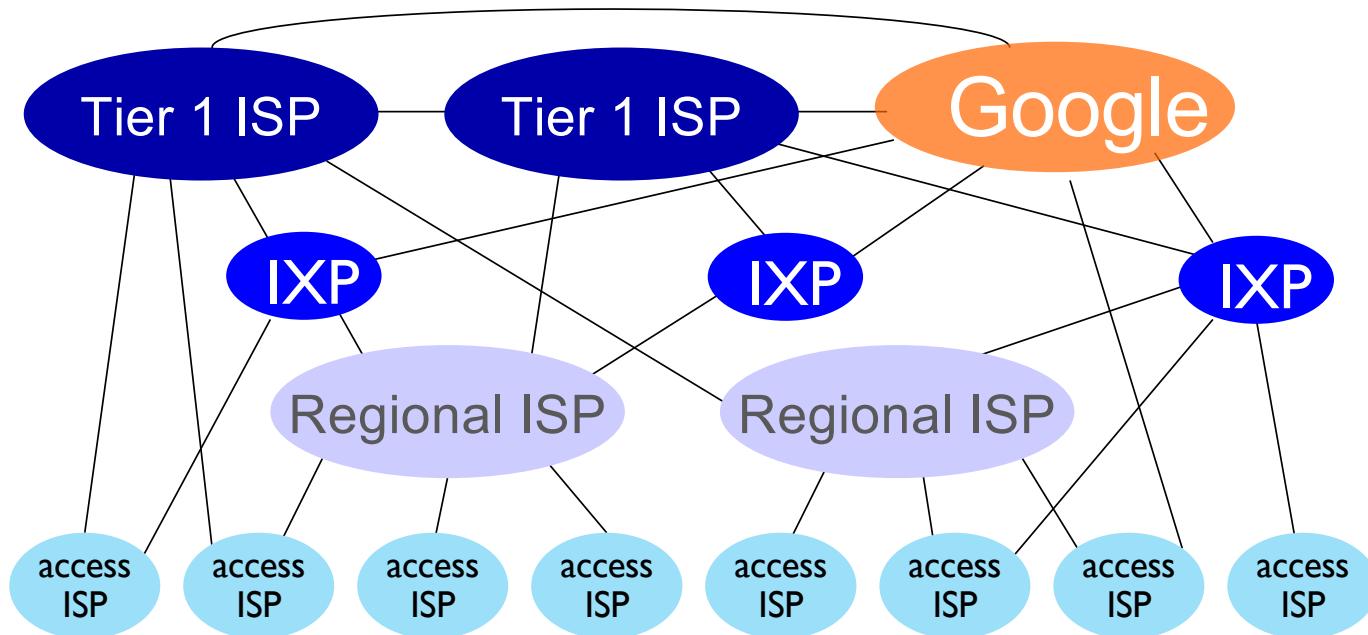


# Internet structure: a “network of networks”

... 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

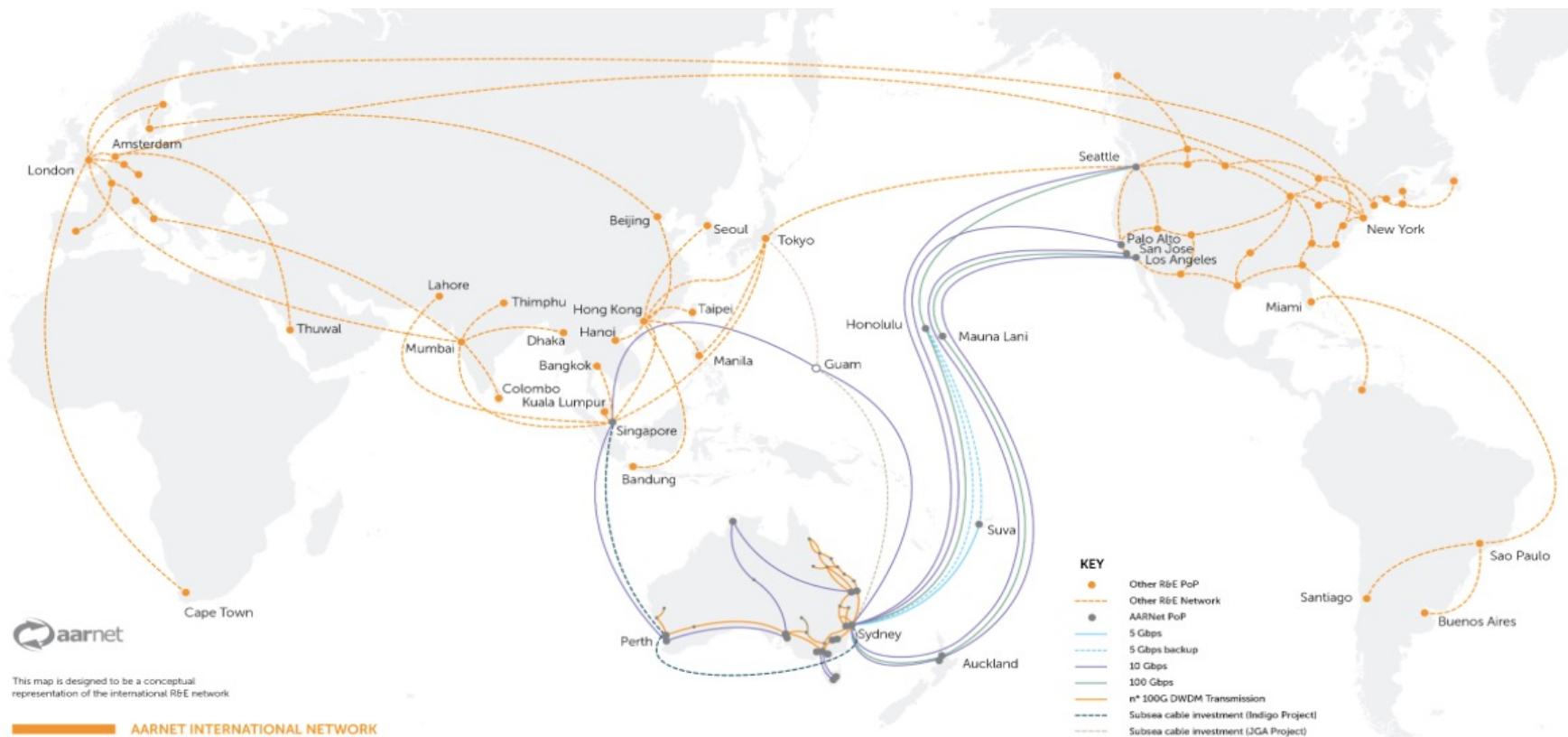
# Tier-1 ISP Network map: Sprint (2019)



# AARNET: Australia's Academic and Research Network

<https://www.aarnet.edu.au/>

<https://www.submarinecablemap.com>



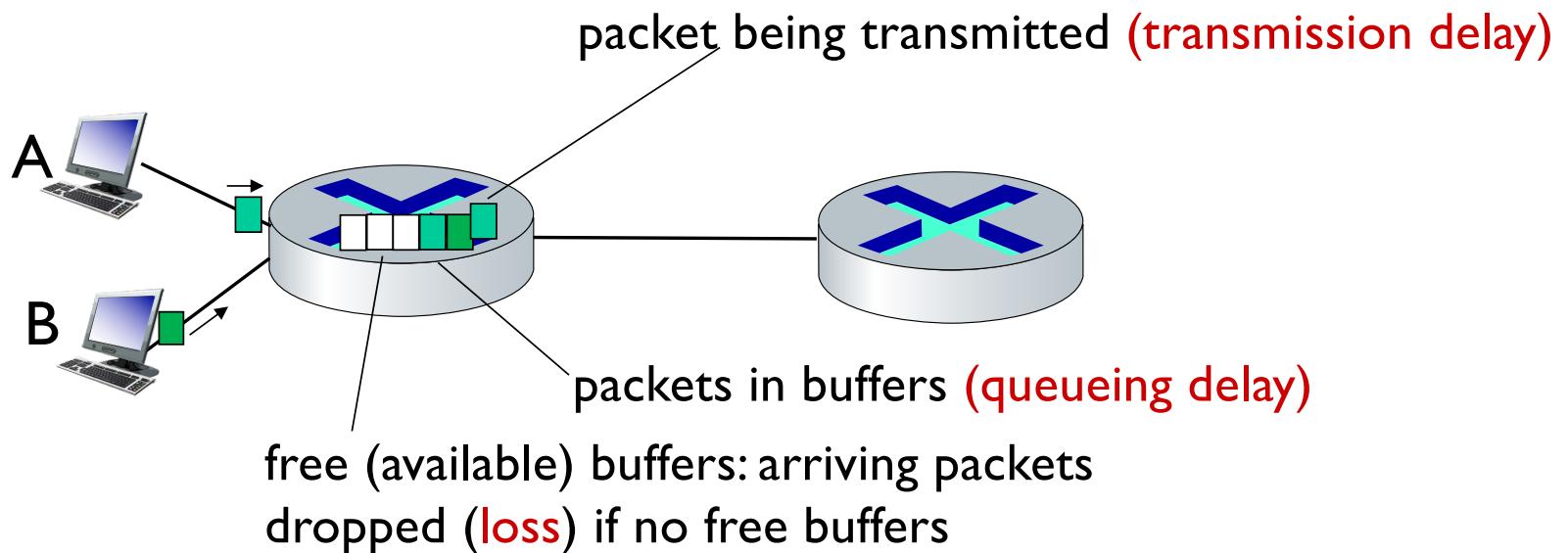
# Introduction: roadmap

- ❖ 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

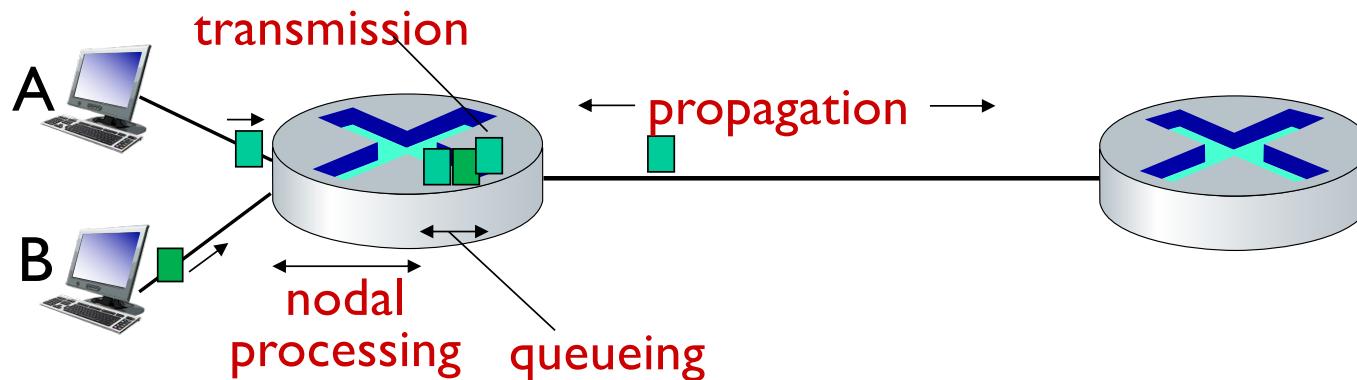
# How do packet loss and delay occur?

packets queue in router buffers

- packets queue, wait for turn
- arrival rate to link (temporarily) exceeds output link capacity: packet loss



# Packet delay: four sources



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

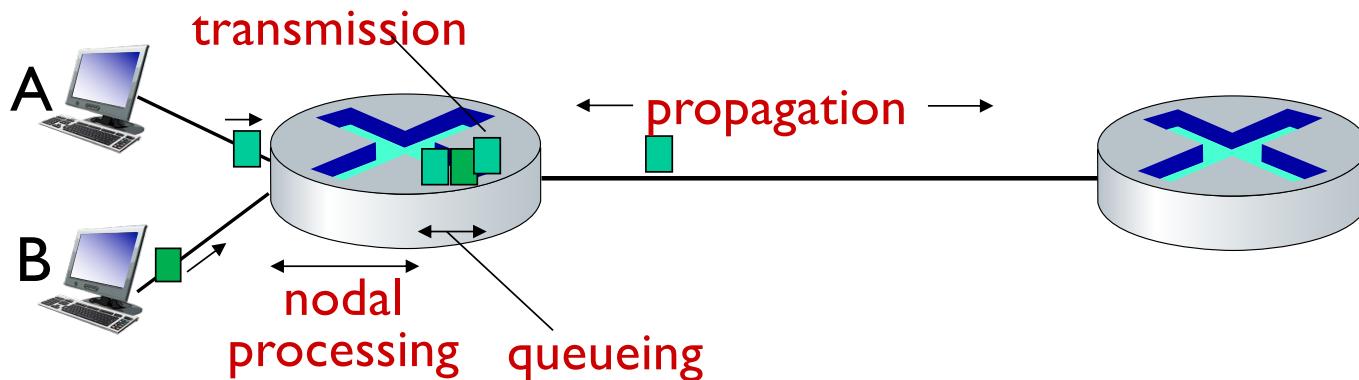
$d_{\text{proc}}$ : nodal processing

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

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

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

# Packet delay: four sources



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

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

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

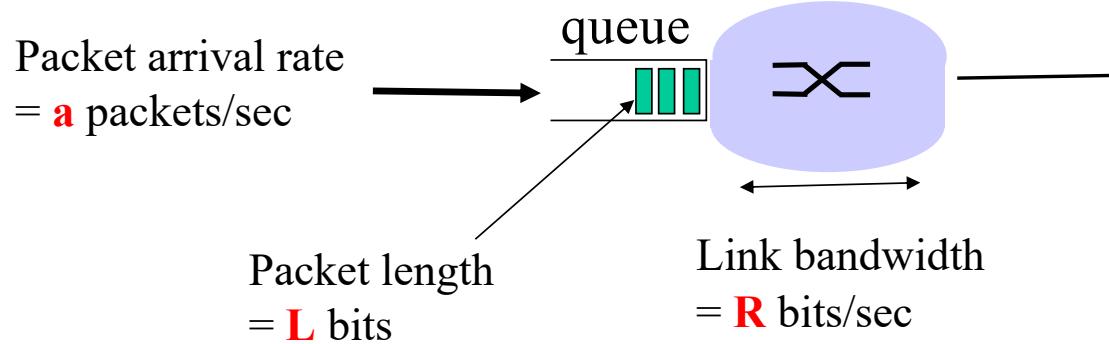
$d_{\text{prop}}$ : propagation delay:

- $d$ : length of physical link
- $s$ : propagation speed ( $\sim 2 \times 10^8$  m/sec)

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

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

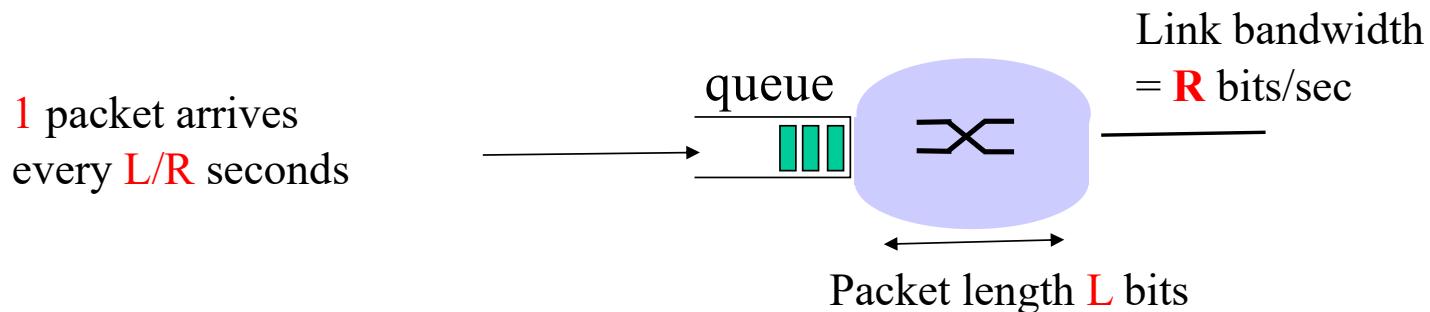
# Queueing delay (more insight)



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

aL/R is called **traffic intensity**

# Queueing delay: One Scenario



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

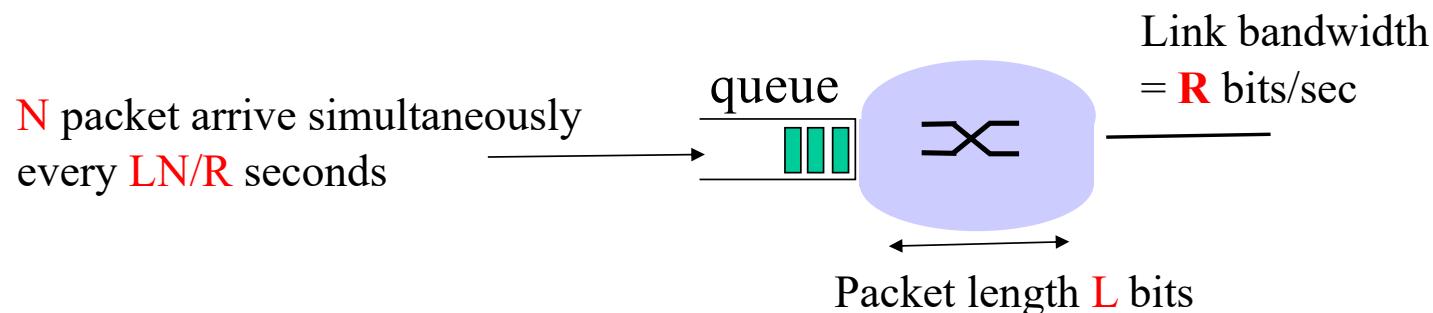


Traffic intensity =  $aL/R = (R/L)(L/R) = 1$

Average queueing delay = 0

(queue is initially empty)

# Queueing delay: Another Scenario



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



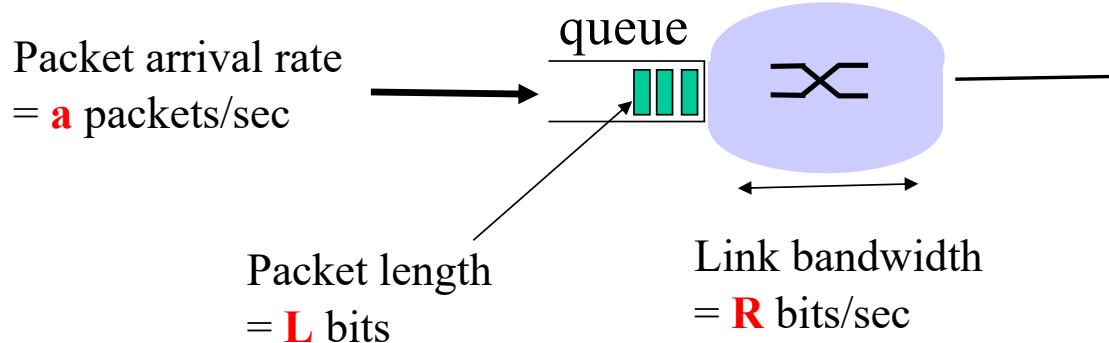
Traffic intensity =  $aL/R = (R/L)(L/R) = 1$

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

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

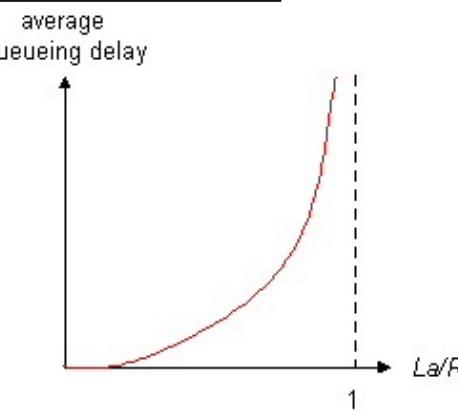
Note: traffic intensity is same as previous scenario, but queueing delay is different

# Queueing delay: typical behaviour

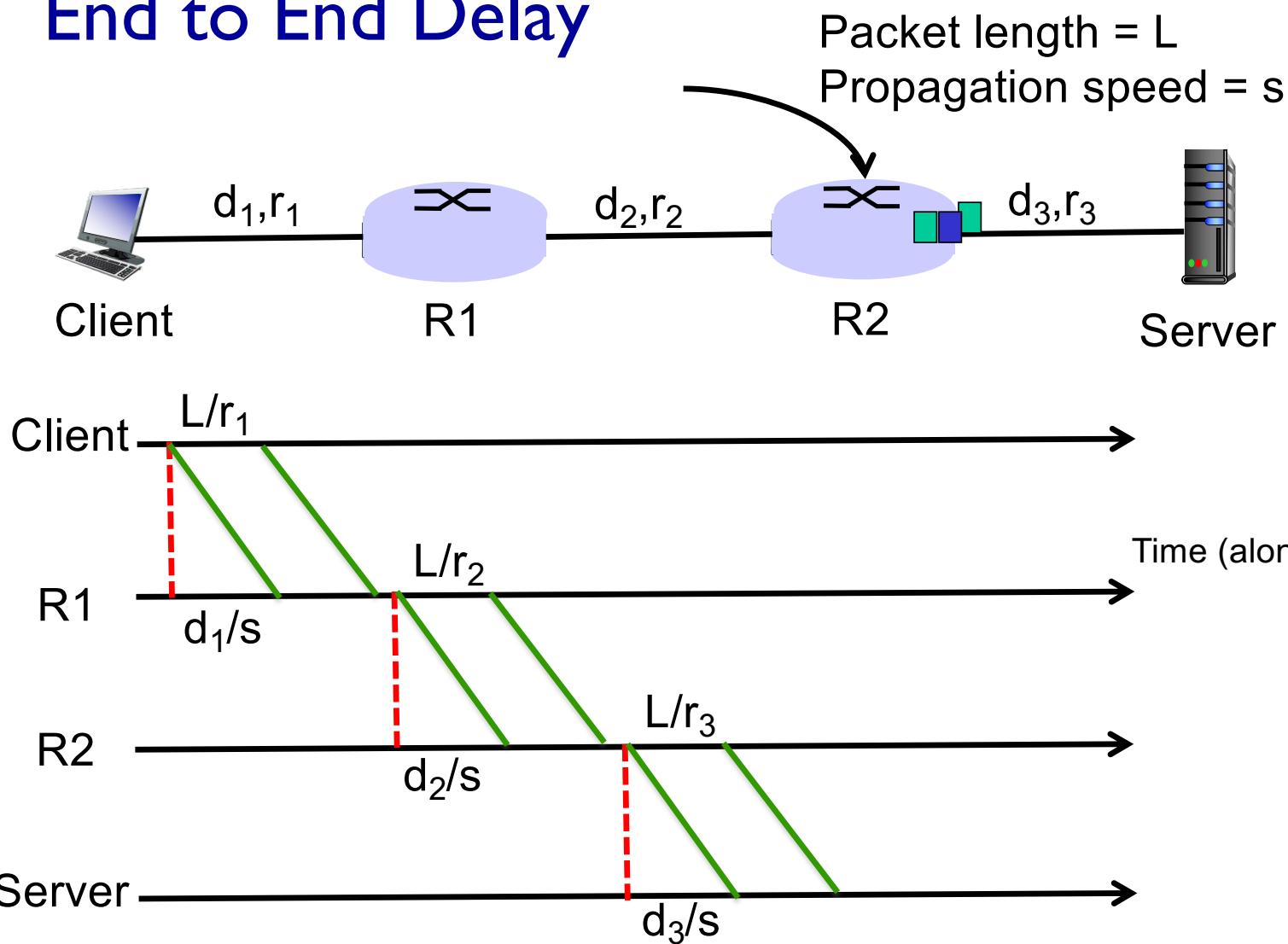


Interactive Java Applet:  
<http://computerscience.unicam.it/marcantoni/reti/applet/QueuingAndLossInteractive/1.html>

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

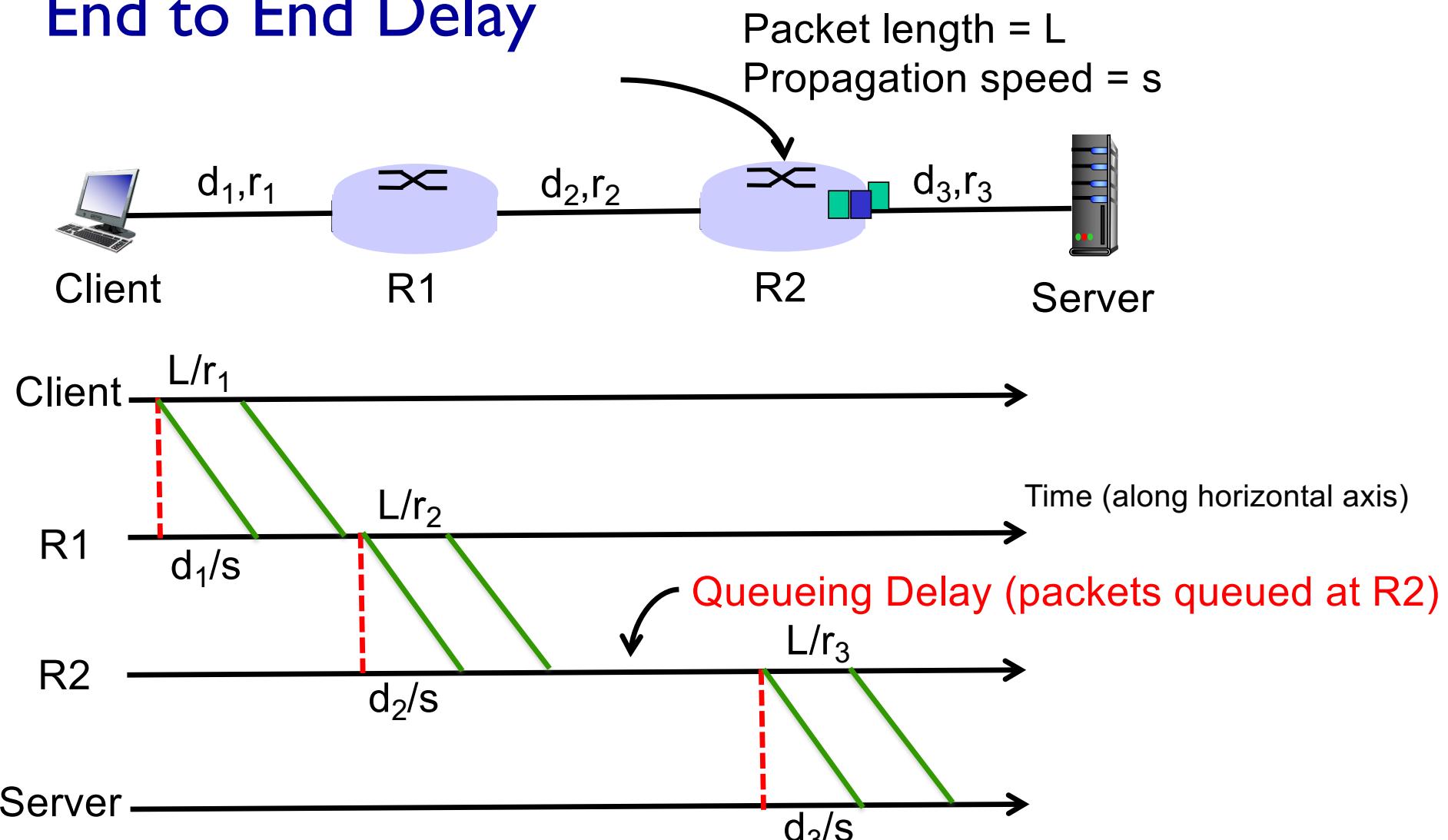


# End to End Delay



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

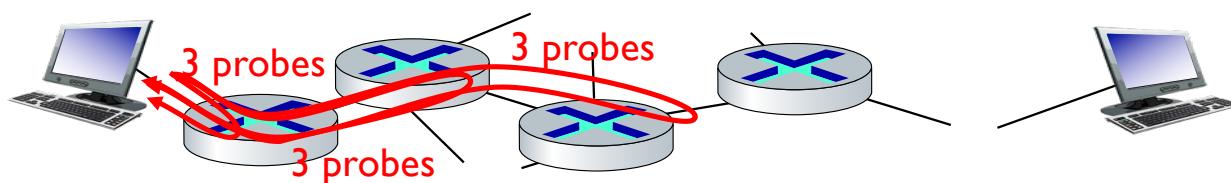
# End to End Delay



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

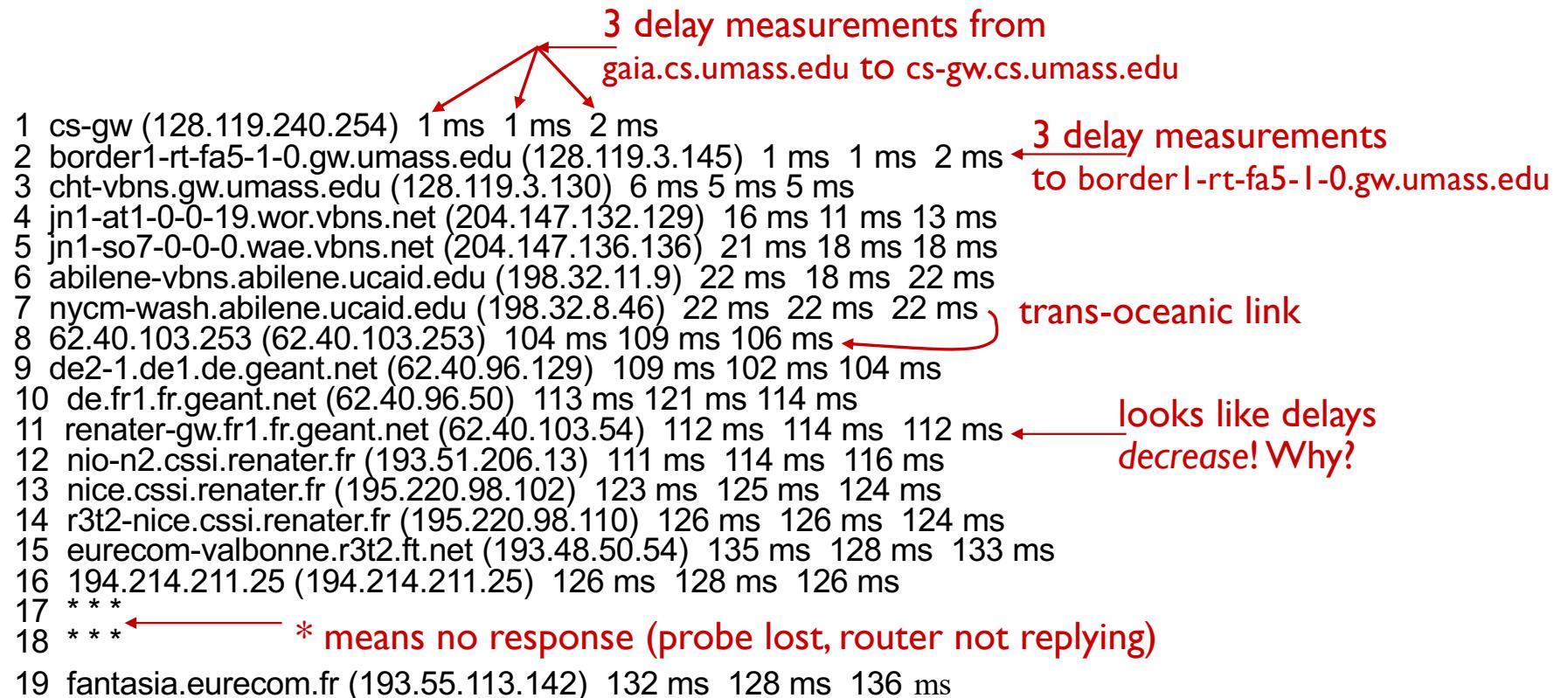
# “Real” Internet delays and routes

- what do “real” Internet delay & loss look like?
- **traceroute** program: provides delay measurement from source to router along end-end Internet path towards destination. For all  $i$ :
  - sends three packets that will reach router  $i$  on path towards destination (with time-to-live field value of  $i$ )
  - router  $i$  will return packets to sender
  - sender measures time interval between transmission and reply



# Real Internet delays and routes

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

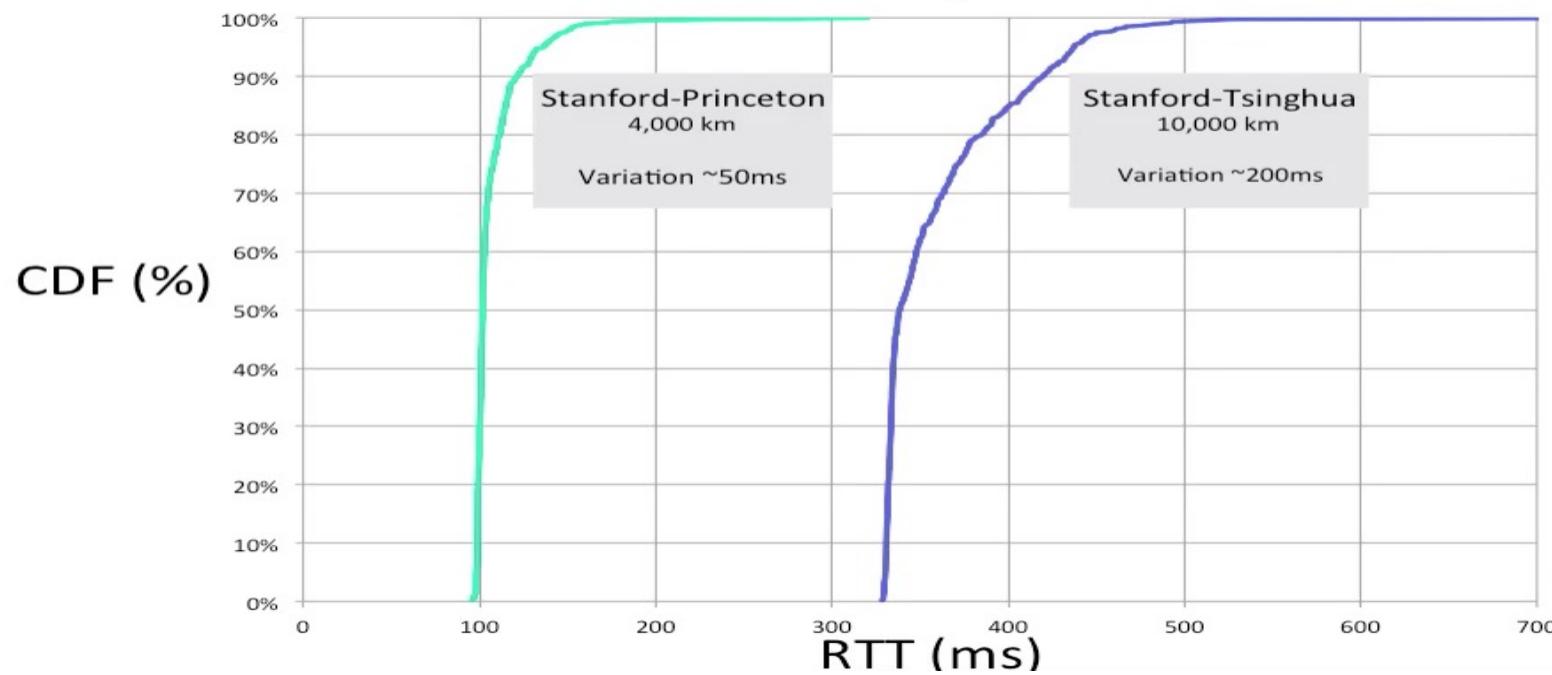


\* Do some traceroutes from exotic countries at [www.traceroute.org](http://www.traceroute.org)

# “Real” delay variations

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

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





## Quiz: Propagation Delay

Propagation delay depends on the size of the packet

- A. True
- B. False



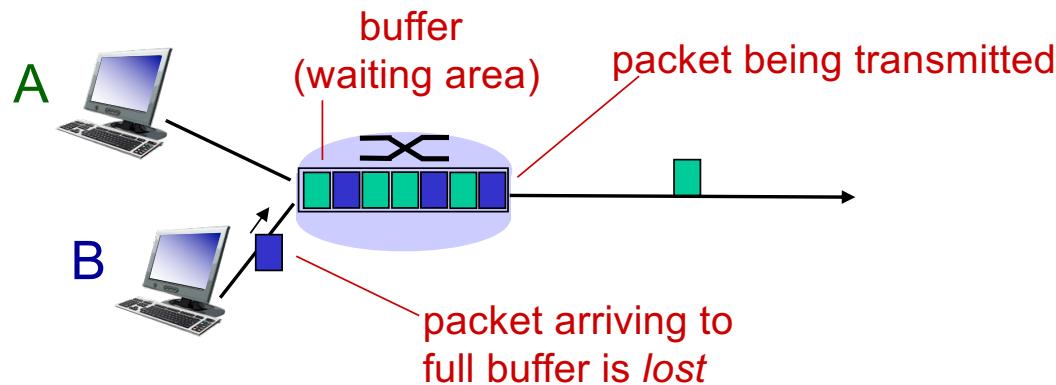
## Quiz: Oh these delays

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

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

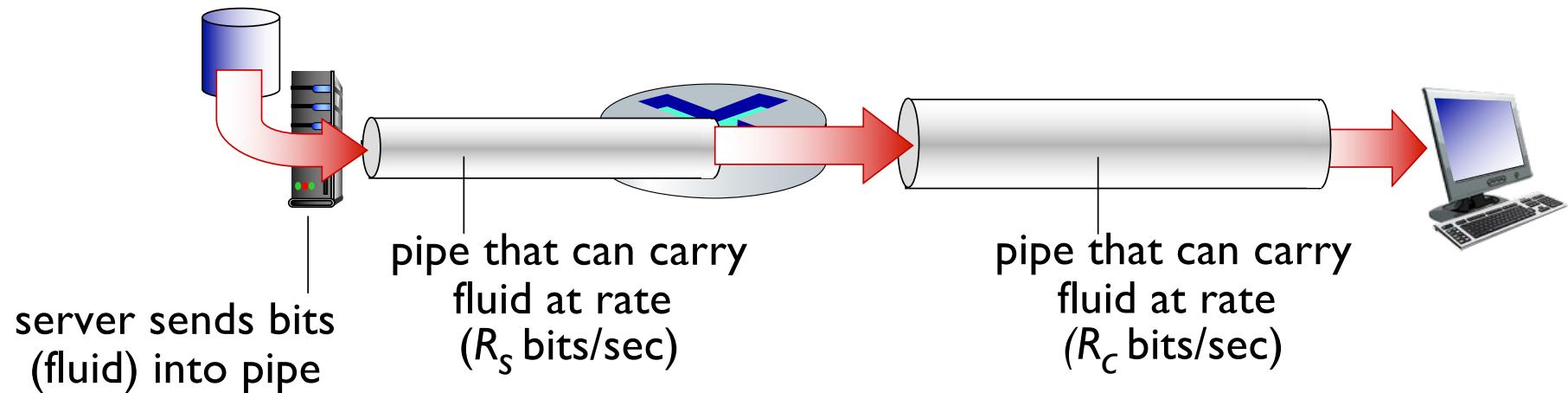
# Packet loss

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



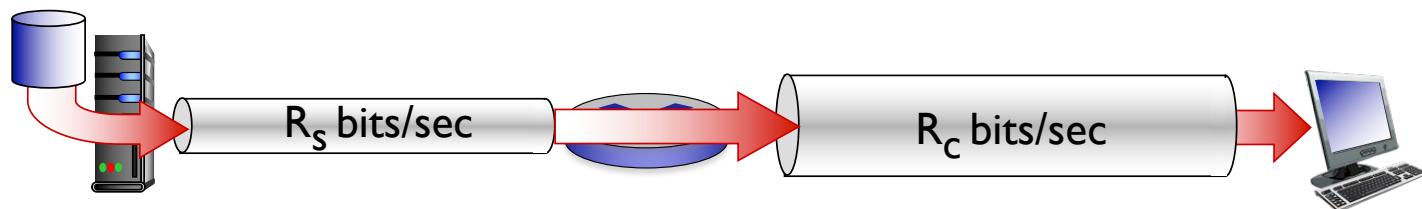
# Throughput

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

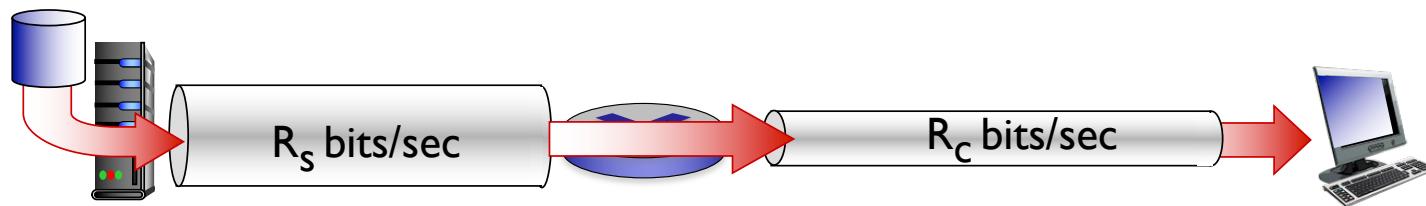


# Throughput

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



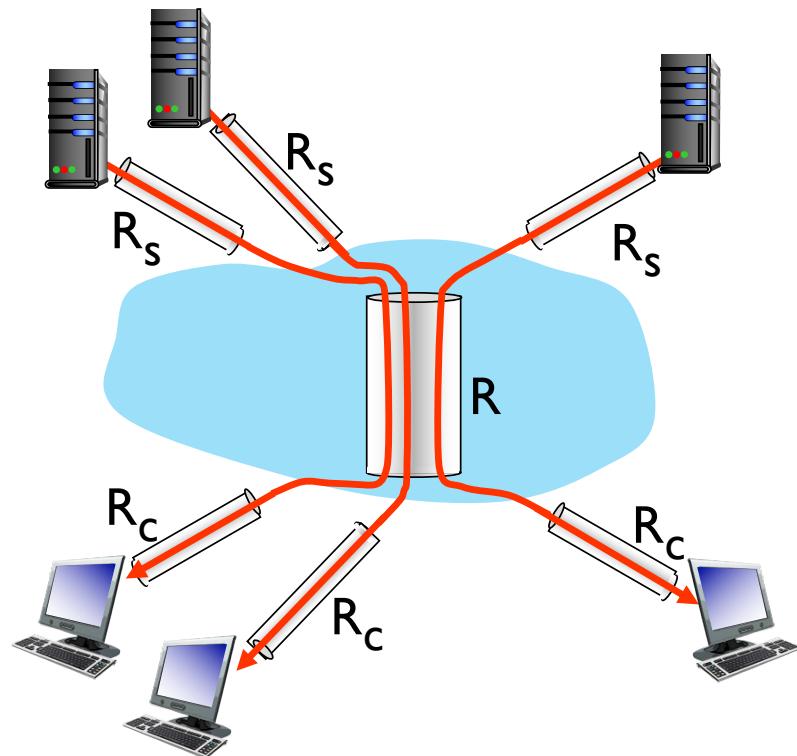
$R_s > R_c$  What is average end-end throughput?



*bottleneck link*

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

# Throughput: network scenario



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

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

# Introduction: summary



*covered a “ton” of material!*

- ❖ Internet overview
- ❖ what's a protocol?
- ❖ network edge, core, access network
  - packet-switching versus circuit-switching
  - Internet structure
- ❖ performance: loss, delay, throughput
- ❖ Next Week
  - Protocol layers, service models
  - Application Layer

## End of Week 1

# I. Introduction: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

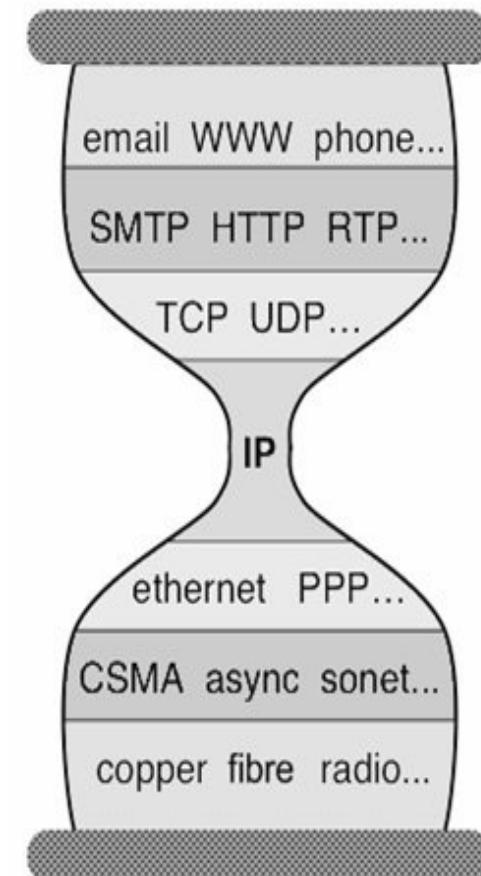
I.6 networks under attack: security

I.7 history

Self study

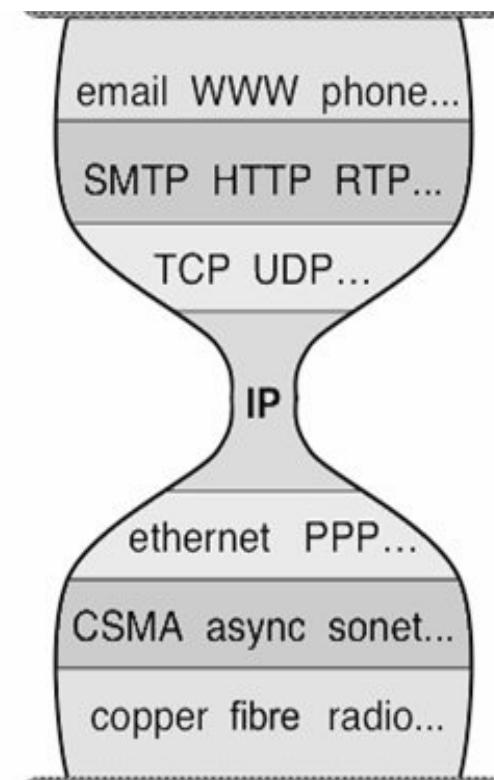
# Internet protocol stack

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

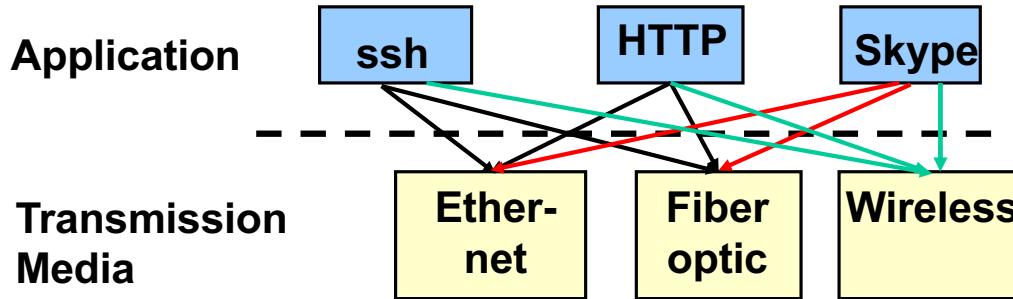


# Three Observations

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



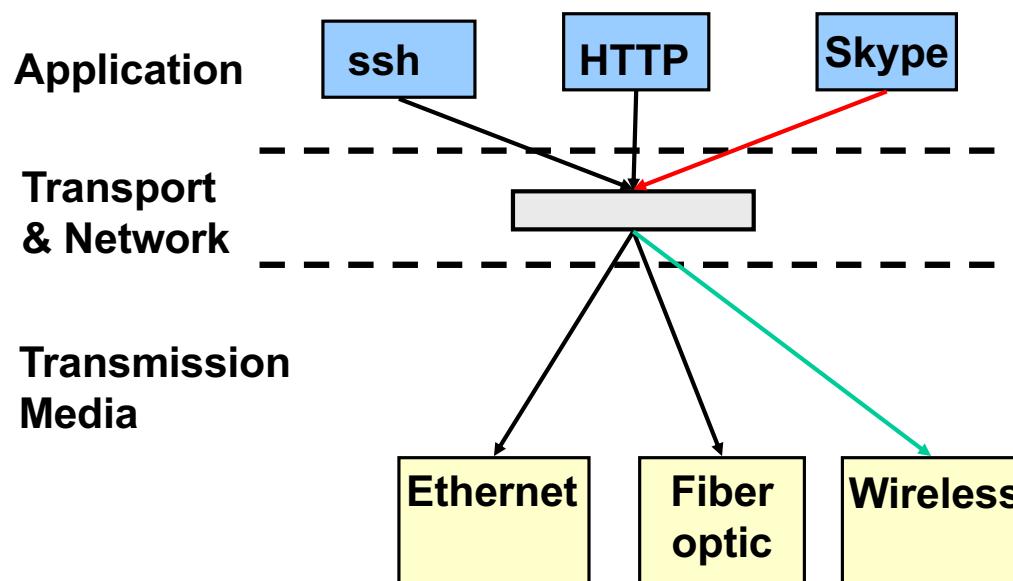
## An Example: No Layering



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

# An Example: Benefit of Layering

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



# Is Layering Harmful?

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

# Distributing Layers Across Network

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

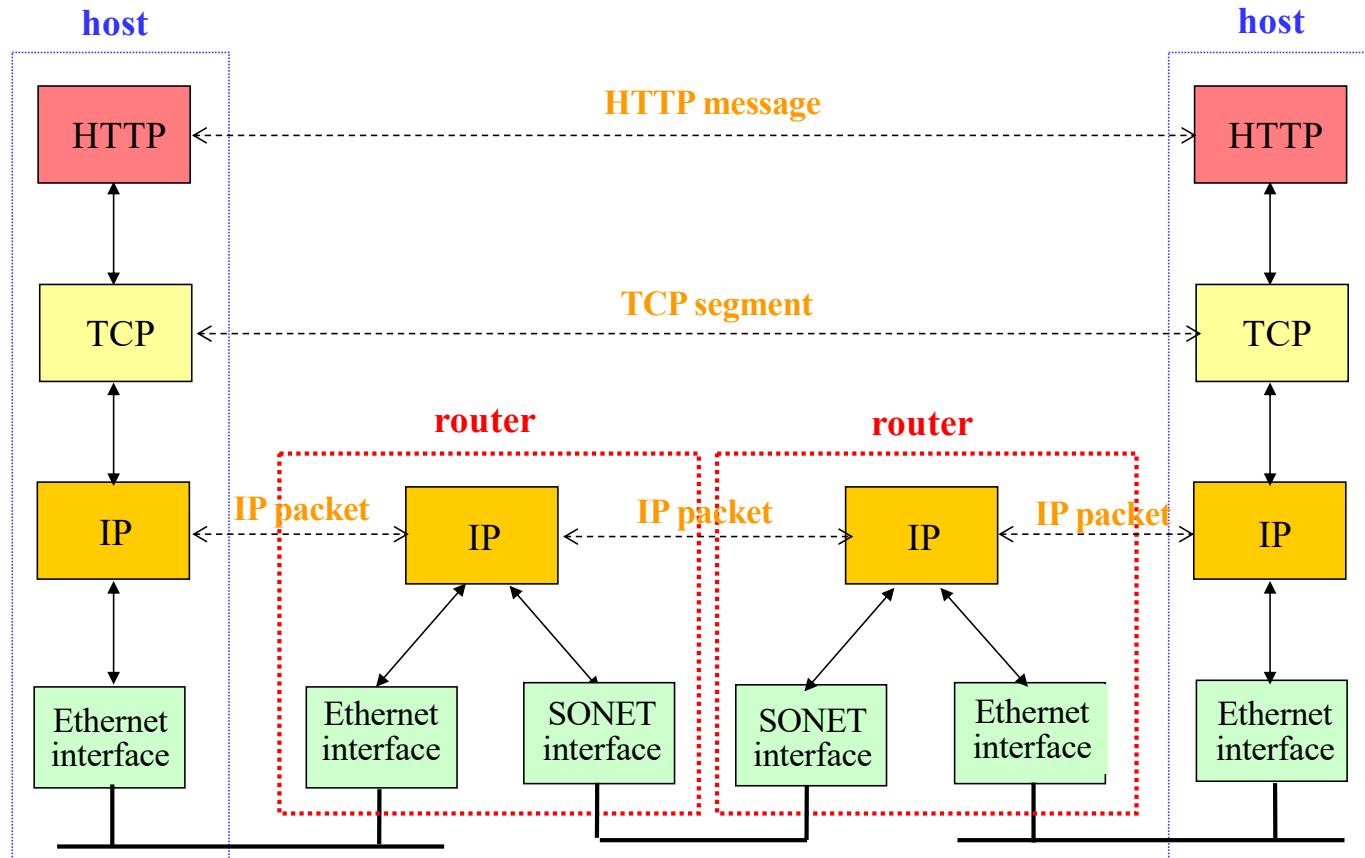
## What Gets Implemented on Host?

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

# What Gets Implemented on Router?

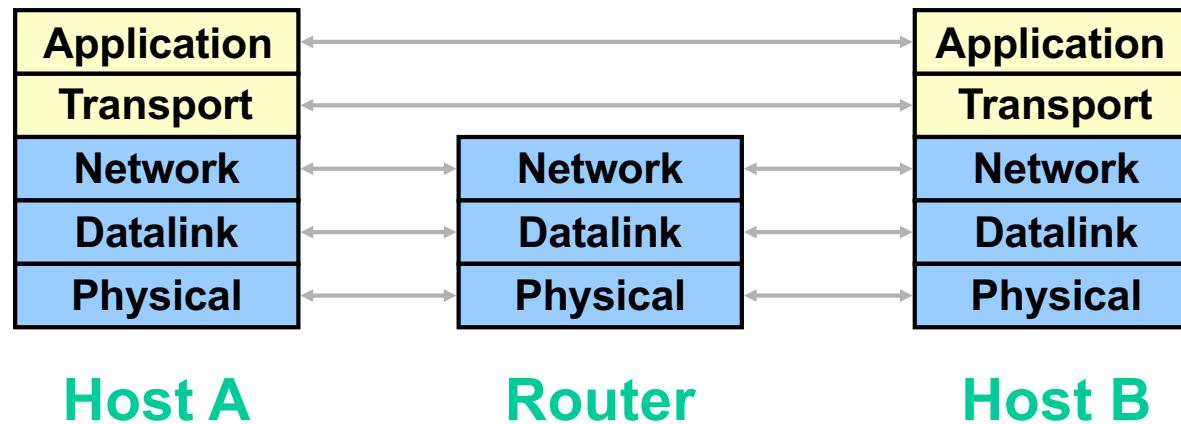
- ❖ Bits arrive on wire
  - Physical layer necessary
- ❖ Packets must be delivered to next-hop
  - datalink layer necessary
- ❖ Routers participate in global delivery
  - Network layer necessary
- ❖ Routers don't support reliable delivery
  - Transport layer (and above) not supported

# Internet Layered Architecture



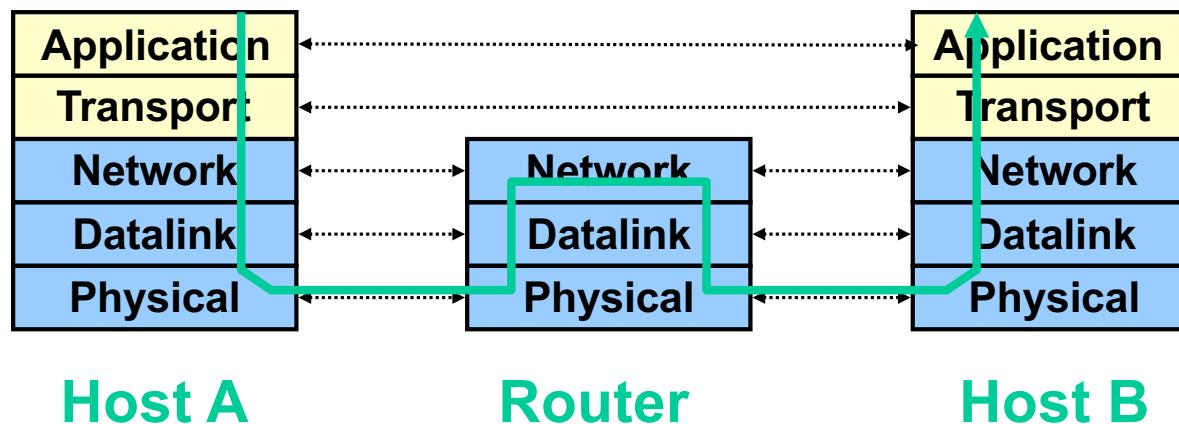
# Logical Communication

- ❖ Layers interact with peer's corresponding layer

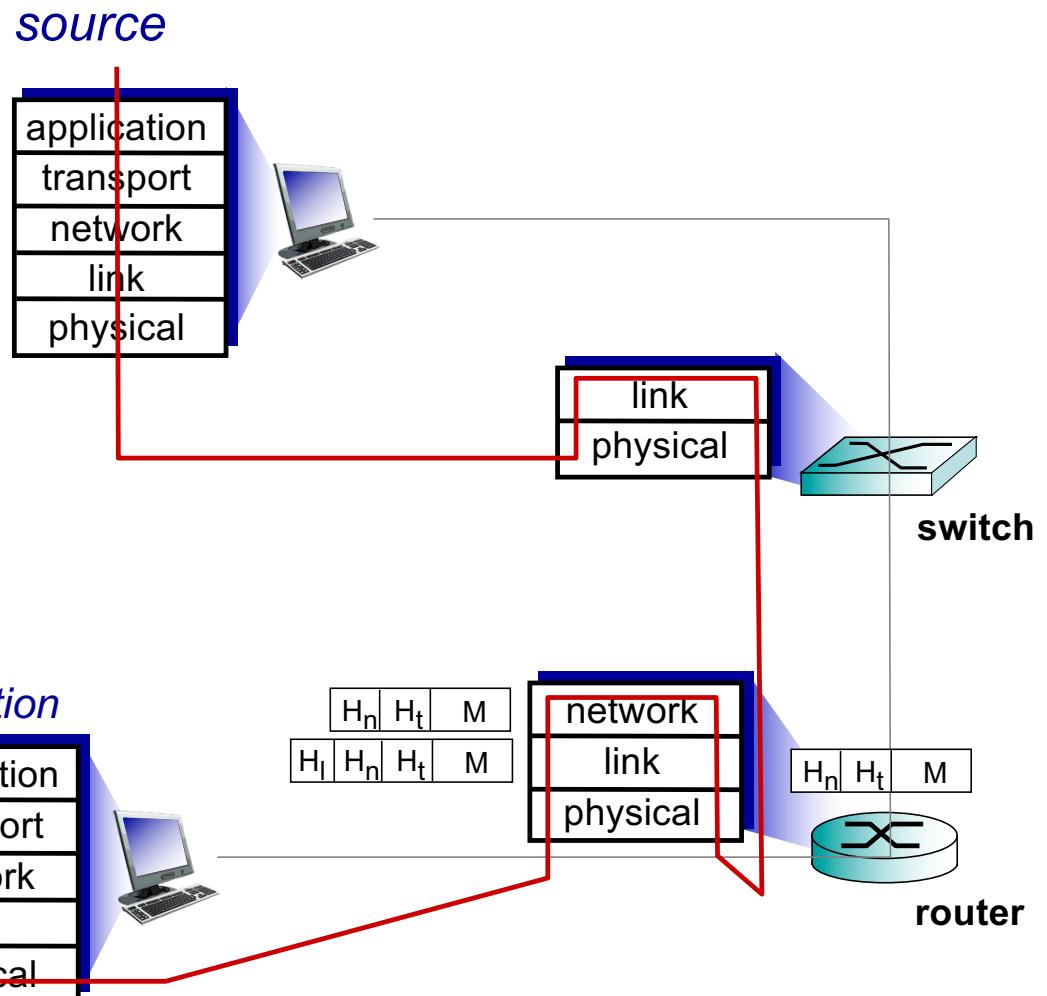
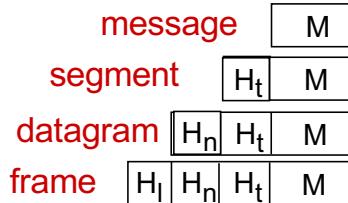


# Physical Communication

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



# Encapsulation





## Quiz: Layering

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

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

# I. Introduction: roadmap

I.1 what *is* the Internet?

I.2 network edge

- end systems, access networks, links

I.3 network core

- packet switching, circuit switching, network structure

I.4 delay, loss, throughput in networks

I.5 protocol layers, service models

I.6 networks under attack: security

Self study

I.7 history

We have now completed Chapter 1 from the textbook

# Computer Networks and Applications

COMP 3331/COMP 9331

Week 2

Application Layer (Principles, Web, Email)

**Chapter 2: Sections 2.1, 2.2, 2.4**

## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

## 2. Application layer

### our goals:

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

# Some network apps

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

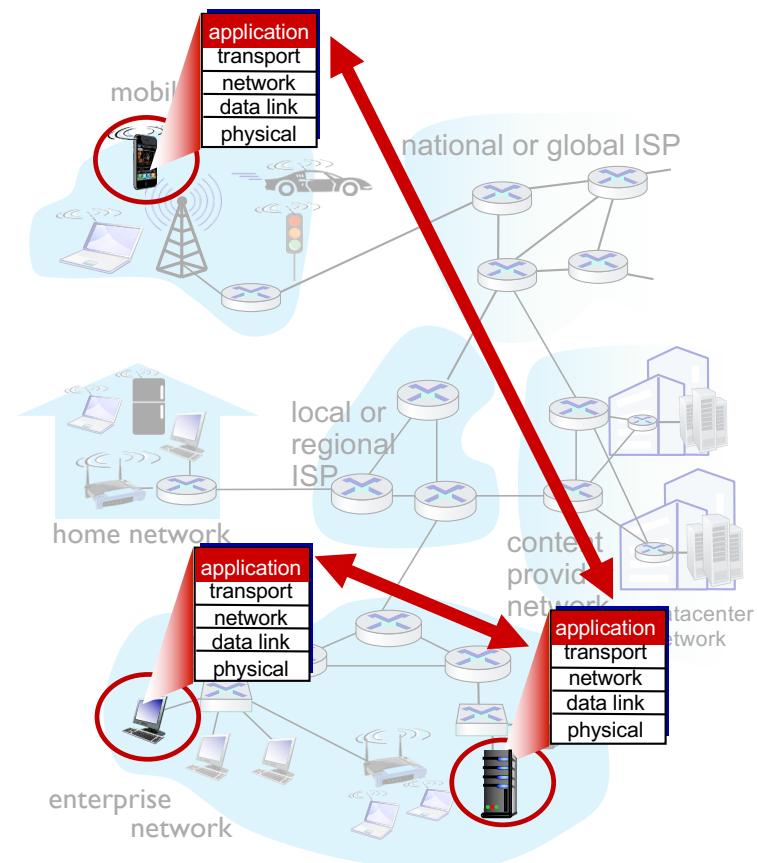
# Creating a network app

write programs that:

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

no need to write software for  
network-core devices

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



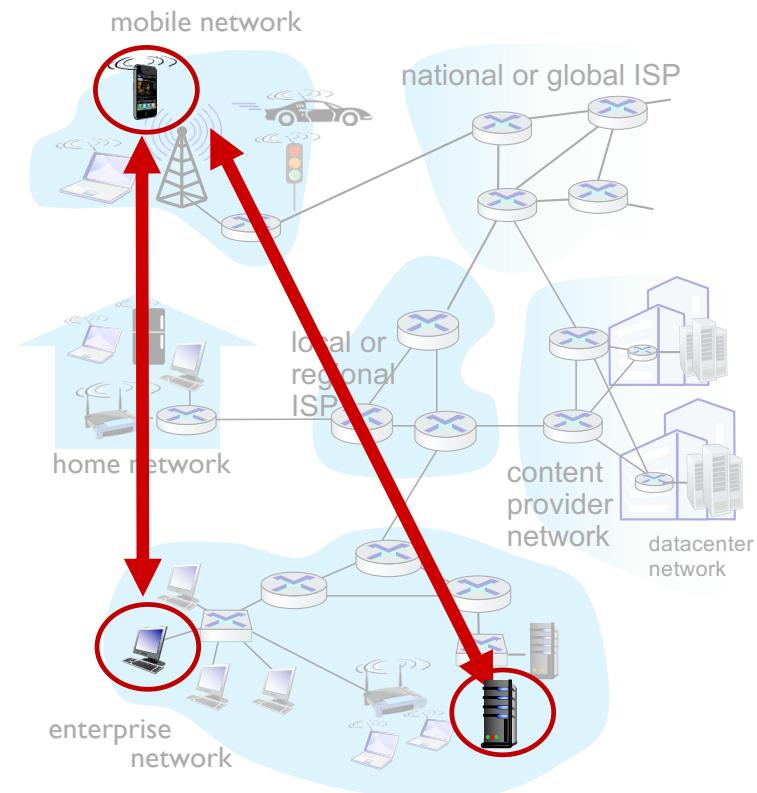
# Client-server paradigm

## server:

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

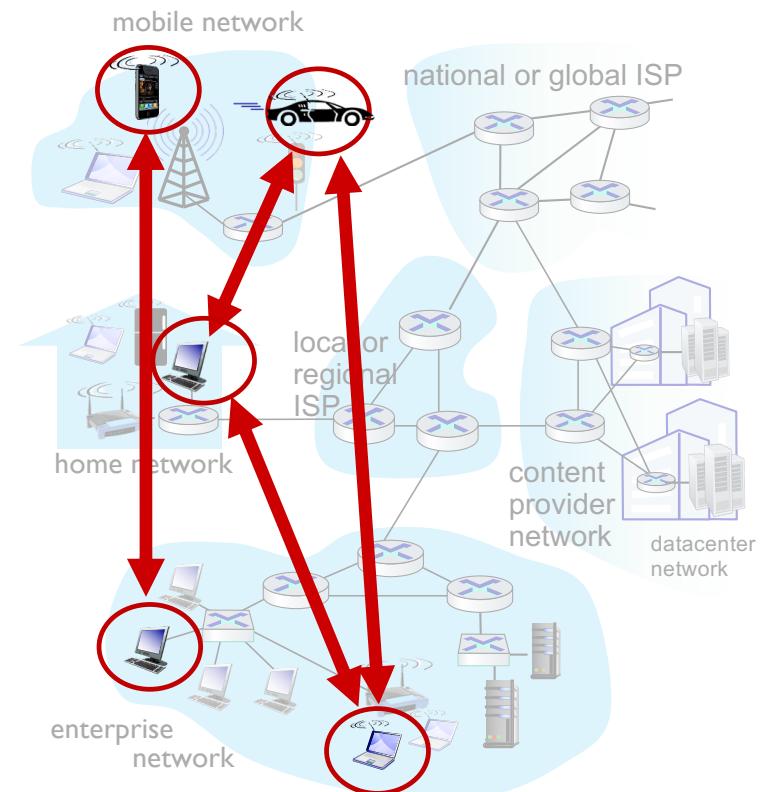
## clients:

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



# Peer-peer architecture

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



# Processes communicating

*process*: program running within a host

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

clients, servers

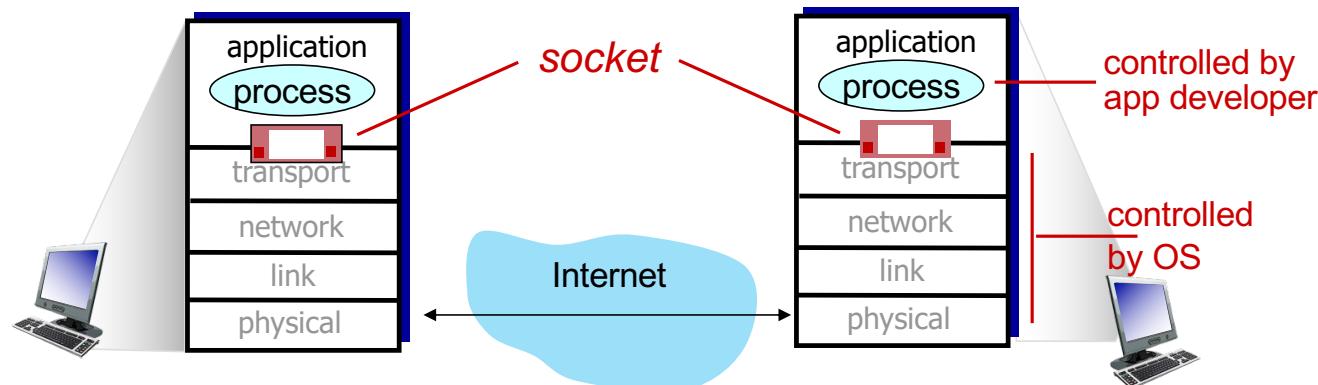
*client process*: process that initiates communication

*server process*: process that waits to be contacted

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

# Sockets

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



# Addressing processes

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

# An application-layer protocol defines:

- types of messages exchanged,
    - e.g., request, response
  - message syntax:
    - what fields in messages & how fields are delineated
  - message semantics
    - meaning of information in fields
  - rules for when and how processes send & respond to messages
- open protocols:**
- defined in RFCs, everyone has access to protocol definition
  - allows for interoperability
  - e.g., HTTP, SMTP, WebRTC
- proprietary protocols:**
- e.g., Skype, Zoom, Teams

# What transport service does an app need?

## data integrity

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

## timing

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

## throughput

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

## security

- encryption, data integrity, ...

# Transport service requirements: common apps

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

# Internet transport protocols services

## TCP service:

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

## UDP service:

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

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

**NOTE:** More on transport layer later

# Internet transport protocols services

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

# Securing TCP

## Vanilla TCP & UDP sockets:

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

## Transport Layer Security (TLS)

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

## TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn

## TLS socket API

- cleartext sent into socket traverse Internet encrypted
- see Chapter 8



## Quiz: Transport

Pick the true statement

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

## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# The Web – History



Tim Berners-Lee

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

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

# 2021 This Is What Happens In An Internet Minute



# Web and HTTP

*First, a quick review...*

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

www.someschool.edu/someDept/pic.gif

host name

path name

# Uniform Resource Locator (URL)

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

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

# HTTP overview

## HTTP: hypertext transfer protocol

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



# HTTP overview (continued)

## *HTTP uses TCP:*

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

## *HTTP is “stateless”*

- server maintains *no* information about past client requests

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

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

# HTTP request message

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

- **HTTP request message:**

- ASCII (human-readable format)

request line (GET,  
POST,  
HEAD commands)

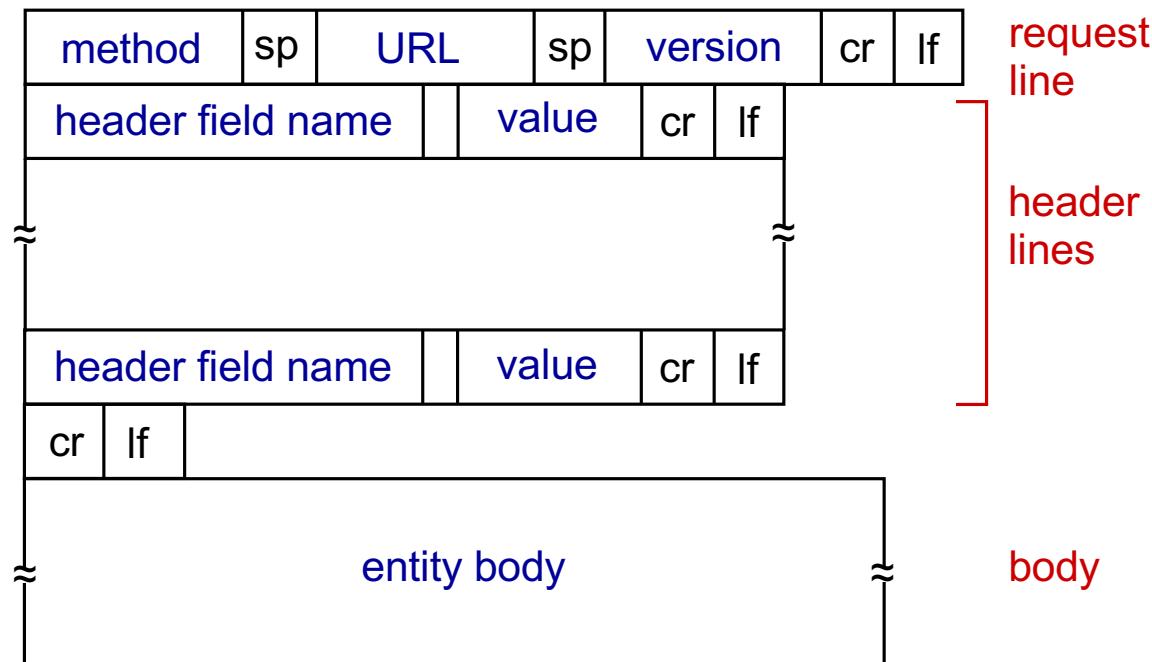
header  
lines

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

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

carriage return character  
line-feed character

# HTTP request message: general format



# Other HTTP request messages

## POST method:

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

## GET method (for sending data to server):

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

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

## HEAD method:

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

## PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of PUT HTTP request message

# HTTP response message

status line (protocol → HTTP/1.1 200 OK\r\nstatus code status phrase)

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

data, e.g., requested → HTML file

# 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

# HTTP is all text

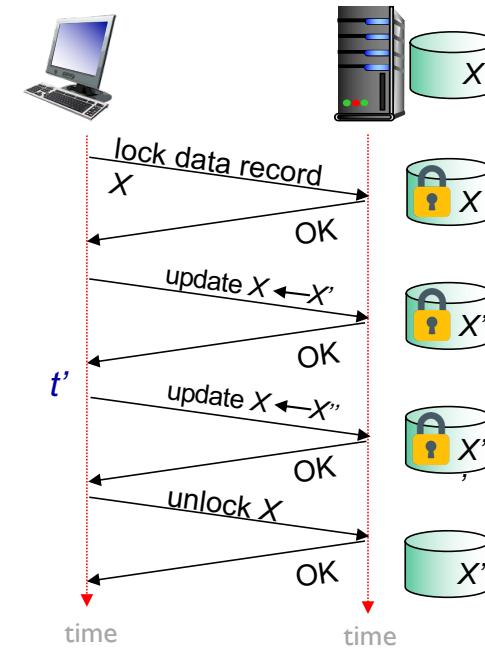
- ❖ Makes the protocol simple
  - Easy to delineate messages (\r\n)
  - (relatively) human-readable
  - No issues about encoding or formatting data
  - Variable length data
- ❖ Not the most efficient
  - Many protocols use binary fields
    - Sending "12345678" as a string is 8 bytes
    - As an integer,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

# Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web “transaction”
  - no need for client/server to track “state” of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to “recover” from a partially-completed-but-never-completely-completed transaction

a **stateful protocol**: client makes two changes to X, or none



Q: what happens if network connection or client crashes at  $t'$ ?

# Maintaining user/server state: cookies

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

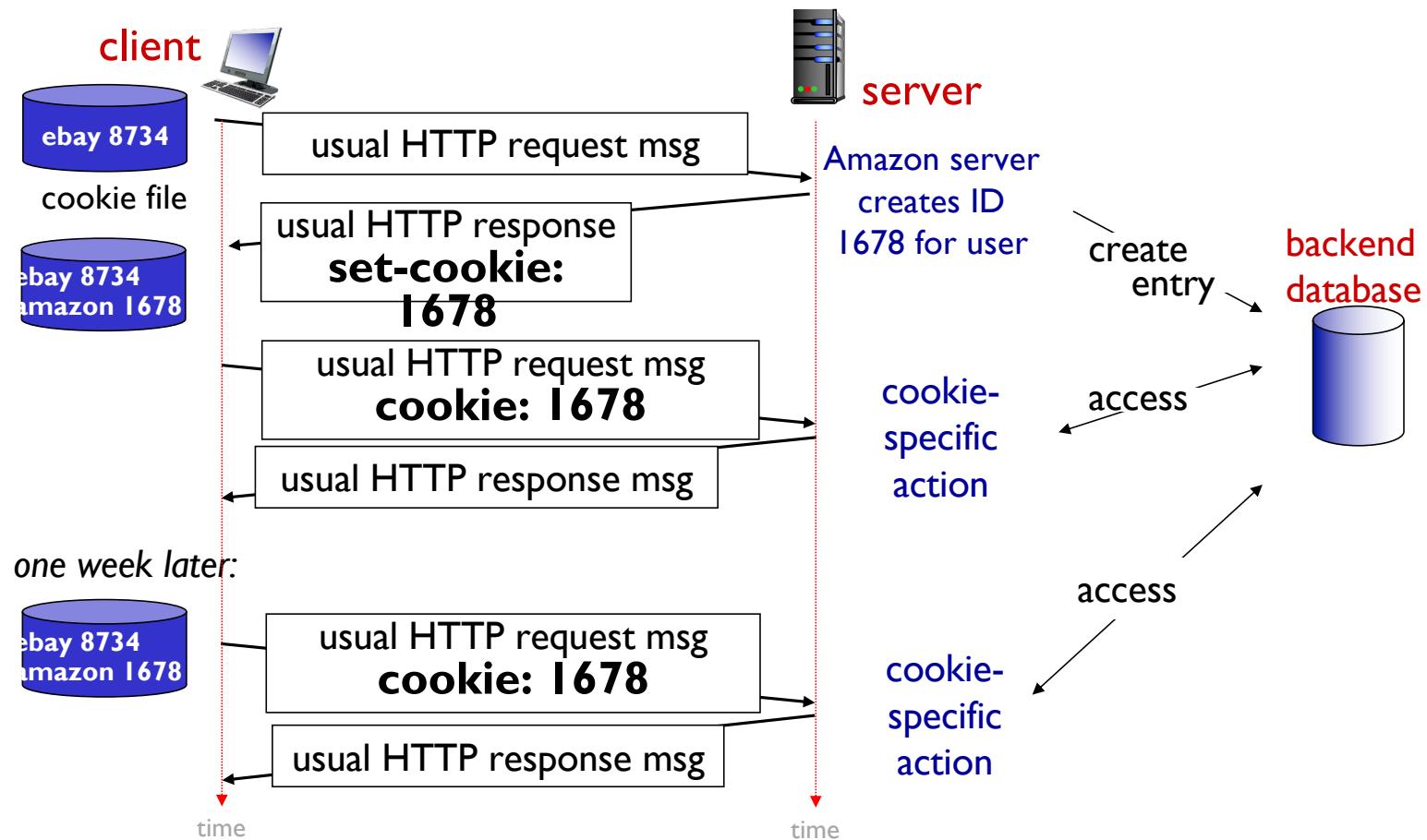
*four components:*

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

**Example:**

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

# Maintaining user/server state: cookies



# HTTP cookies: comments

*What cookies can be used for:*

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

*Challenge: How to keep state:*

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: HTTP messages carry state

*aside  
cookies and privacy:*

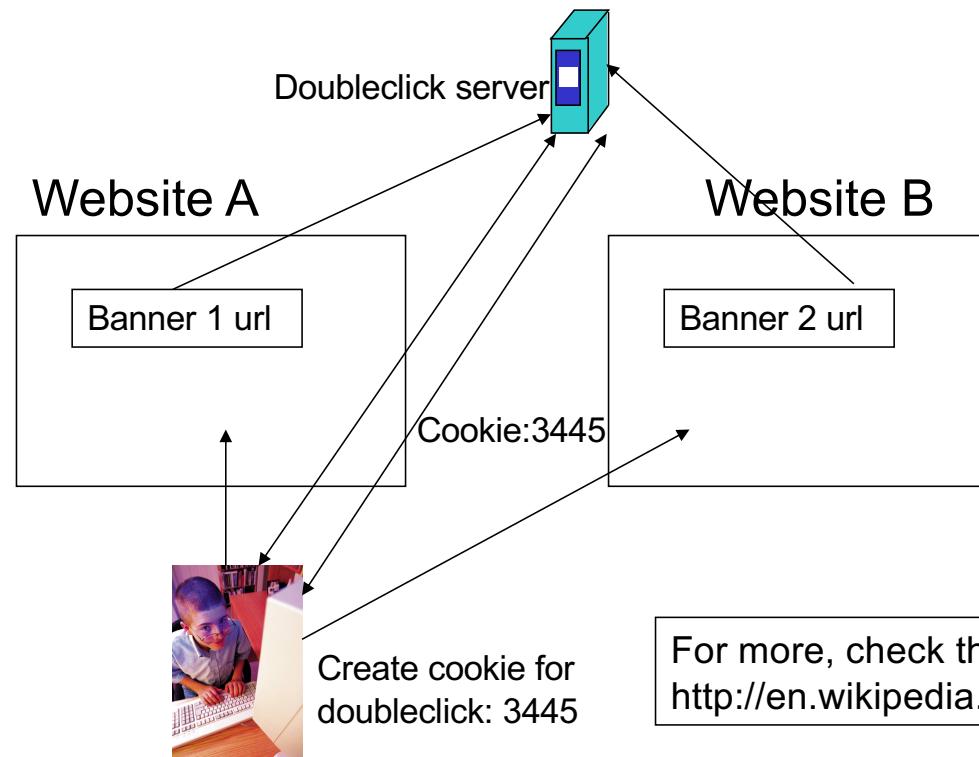
- cookies permit sites to learn a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

# The Dark Side of Cookies



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

# Third party cookies



For more, check the following link and follow the references:  
[http://en.wikipedia.org/wiki/HTTP\\_cookie](http://en.wikipedia.org/wiki/HTTP_cookie)

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



# Performance of HTTP

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

# Performance Goals

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

# Solutions?

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

Improve HTTP to  
achieve faster  
downloads



# Solutions?

- ❖ User

- fast downloads
- high availability

Improve HTTP to  
achieve faster  
downloads

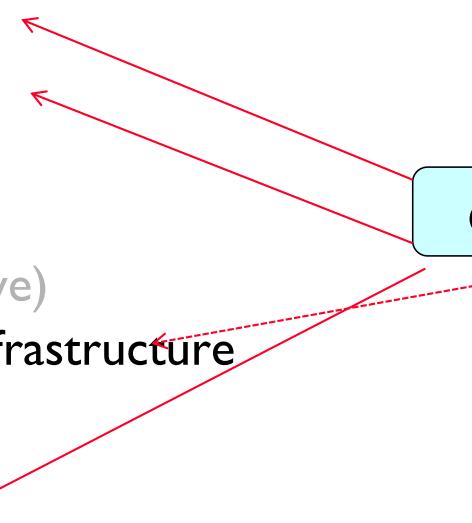
- ❖ Content provider

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

Caching and Replication

- ❖ Network (secondary)

- avoid overload



# Solutions?

- ❖ User

- fast downloads
- high availability

Improve HTTP to  
achieve faster  
downloads

- ❖ Content provider

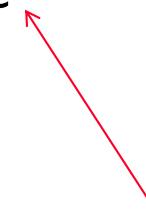
- happy users (hence, above)
- cost-effective delivery infrastructure

Caching and Replication

- ❖ Network (secondary)

- avoid overload

Exploit economies of scale  
(Webhosting, CDNs, datacenters)



# How to improve Page Load Time (PLT)

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

# HTTP Performance

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

## *non-persistent HTTP*

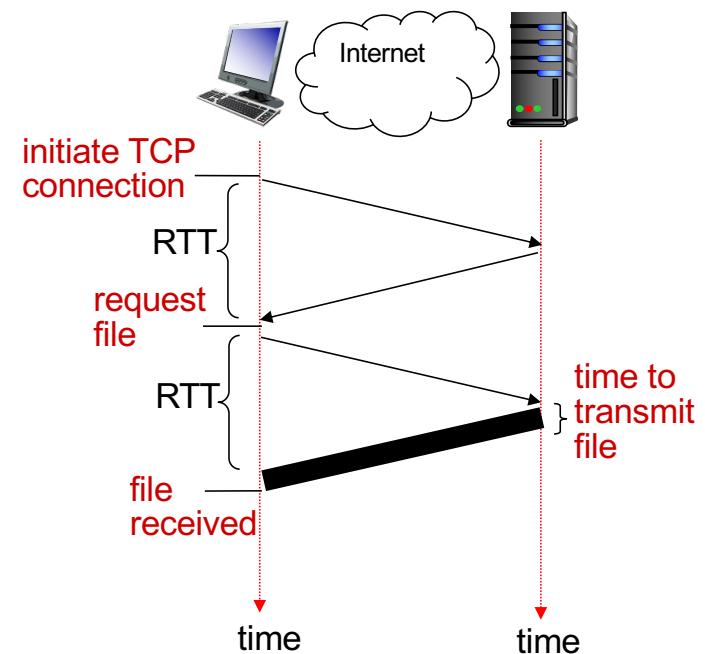
- ❖ at most one object sent over TCP connection
  - connection then closed
- ❖ downloading multiple objects required multiple connections

# Non-persistent HTTP: response time

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

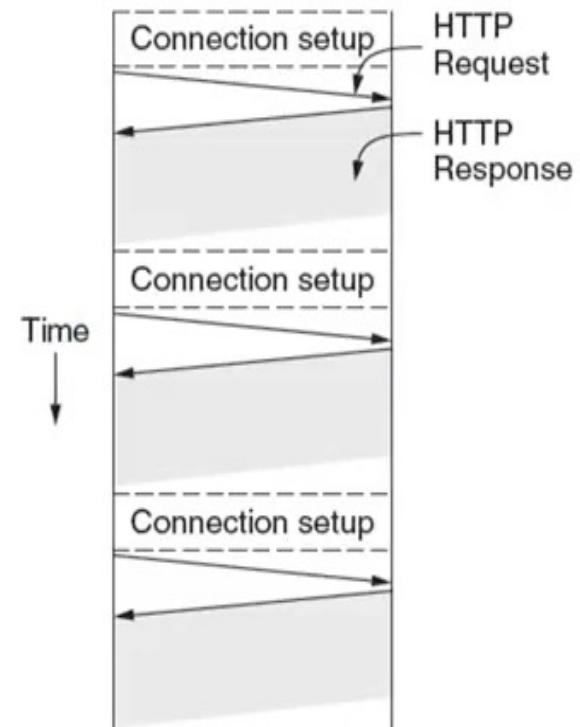
**HTTP response time:**

- ❖ one RTT to initiate TCP connection (approximate 3-way handshake)
- ❖ one RTT for HTTP request and first few bytes of HTTP response to return
- ❖ file transmission time
- ❖ non-persistent HTTP response time =  $2\text{RTT} + \text{file transmission time}$



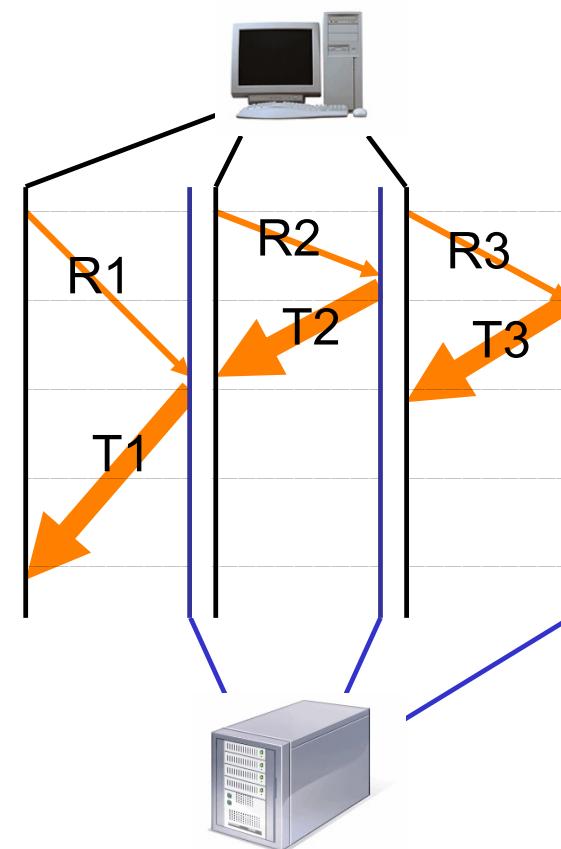
# HTTP/1.0

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



Improving HTTP Performance:  
**Concurrent Requests & Responses**

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



## Quiz: Parallel HTTP Connections



What are potential downsides of parallel HTTP connections, i.e., can opening too many parallel connections be harmful and if so in what way?

Answer:

# Persistent HTTP (HTTP/1.1)

## Persistent HTTP

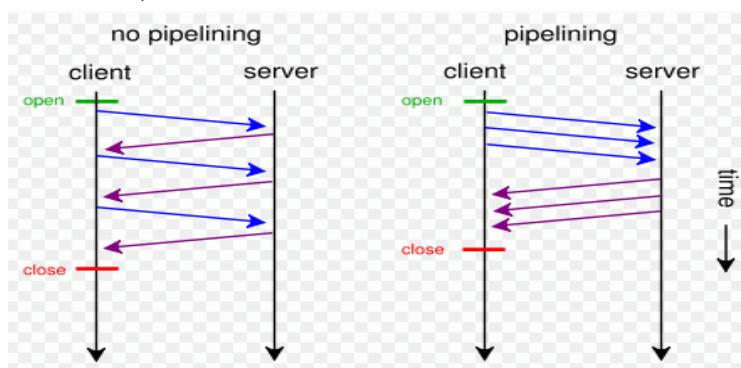
- ❖ server leaves TCP connection open after sending response
- ❖ subsequent HTTP messages between same client/server are sent over the same TCP connection
- ❖ Allow TCP to learn more accurate RTT estimate (APPARENT LATER IN THE COURSE)
- ❖ Allow TCP congestion window to increase (APPARENT LATER)
- ❖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

## Persistent without pipelining:

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

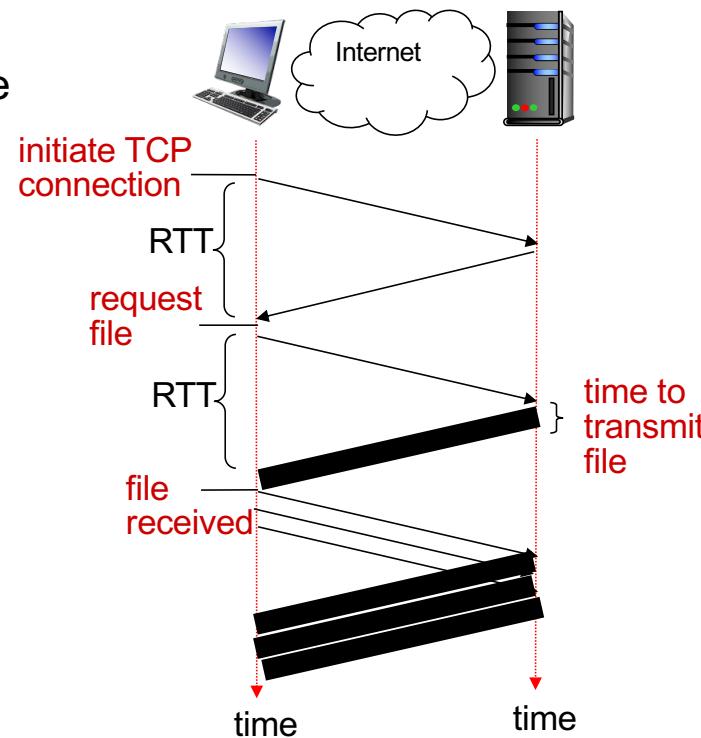
## Persistent with pipelining:

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



# HTTP 1.1: response time with pipelining

Website with one index page and three embedded objects



# How to improve PLT

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

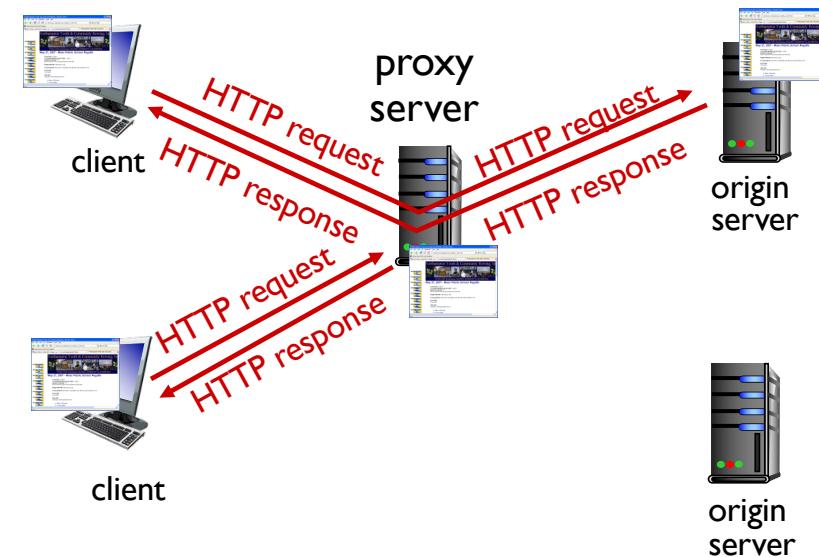
# Improving HTTP Performance: Caching

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

# Web caches (proxy servers)

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

- user configures browser to point to a *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



# Web caches (proxy servers)

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

## *Why* Web caching?

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

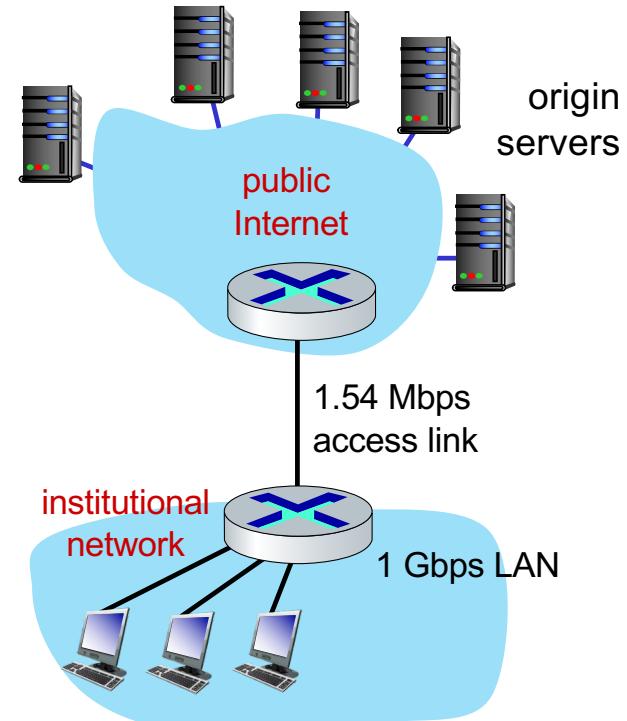
# Caching example

## Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Average request rate from browsers to origin servers: 15/sec
  - average data rate to browsers: 1.50 Mbps

## Performance:

- LAN utilization: .0015
- access link utilization = **.97**  
*problem: large delays at high utilization!*
- end-end delay = Internet delay +  
access link delay + LAN delay  
= 2 sec + minutes + usecs



# Caching example: buy a faster access link

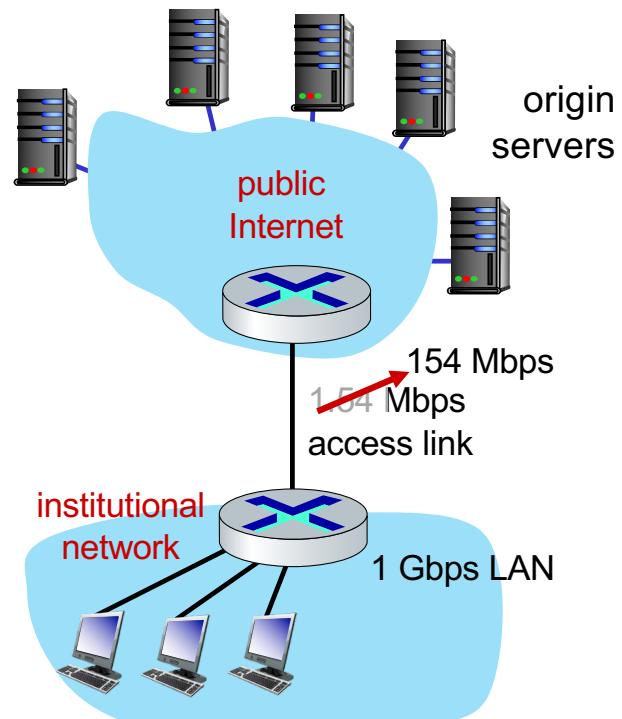
## Scenario:

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

## Performance:

- LAN utilization: .0015
- access link utilization = .07 → .0097
- end-end delay = Internet delay +  
access link delay + LAN delay  
= 2 sec + minutes + usecs

Cost: faster access link (expensive!) → msecs



# Caching example: install a web cache

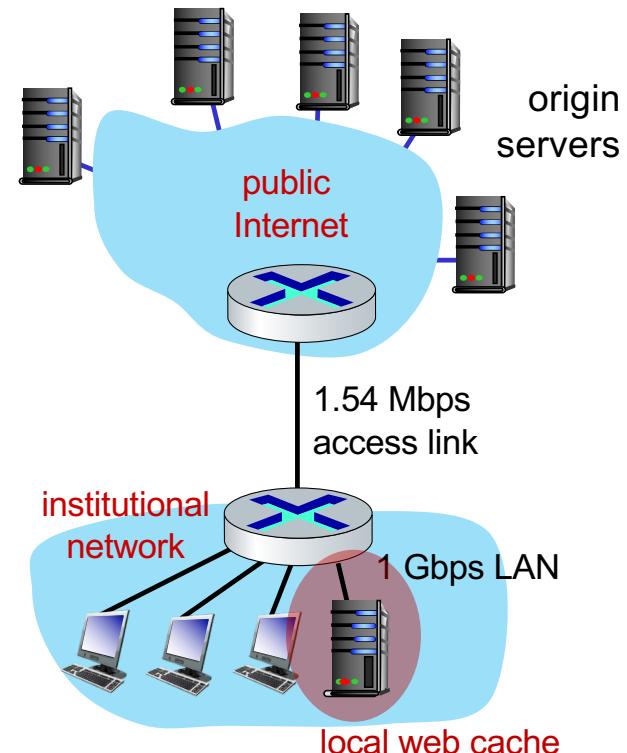
## Scenario:

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

## Performance:

- LAN utilization: .?
- access link utilization = ? *How to compute link utilization, delay?*
- average end-end delay = ?

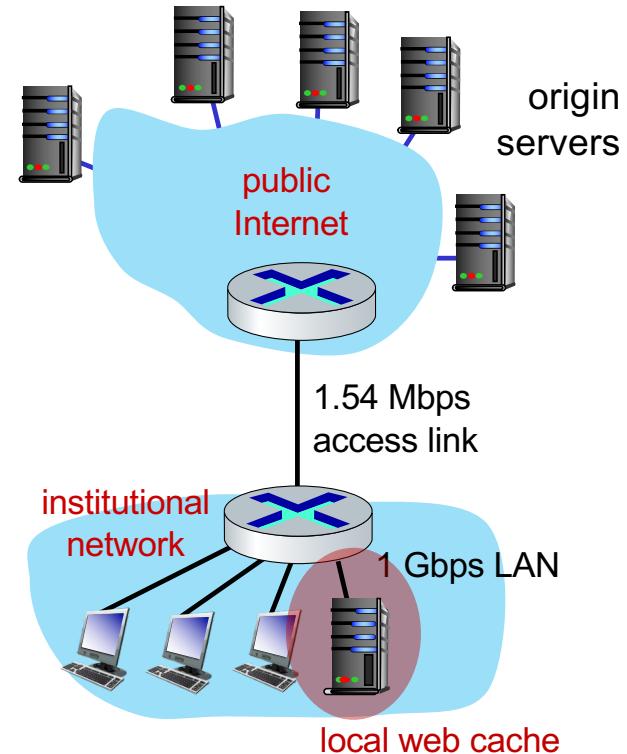
*Cost:* web cache (cheap!)



# Caching example: install a web cache

Calculating access link utilization, end-end delay with cache:

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

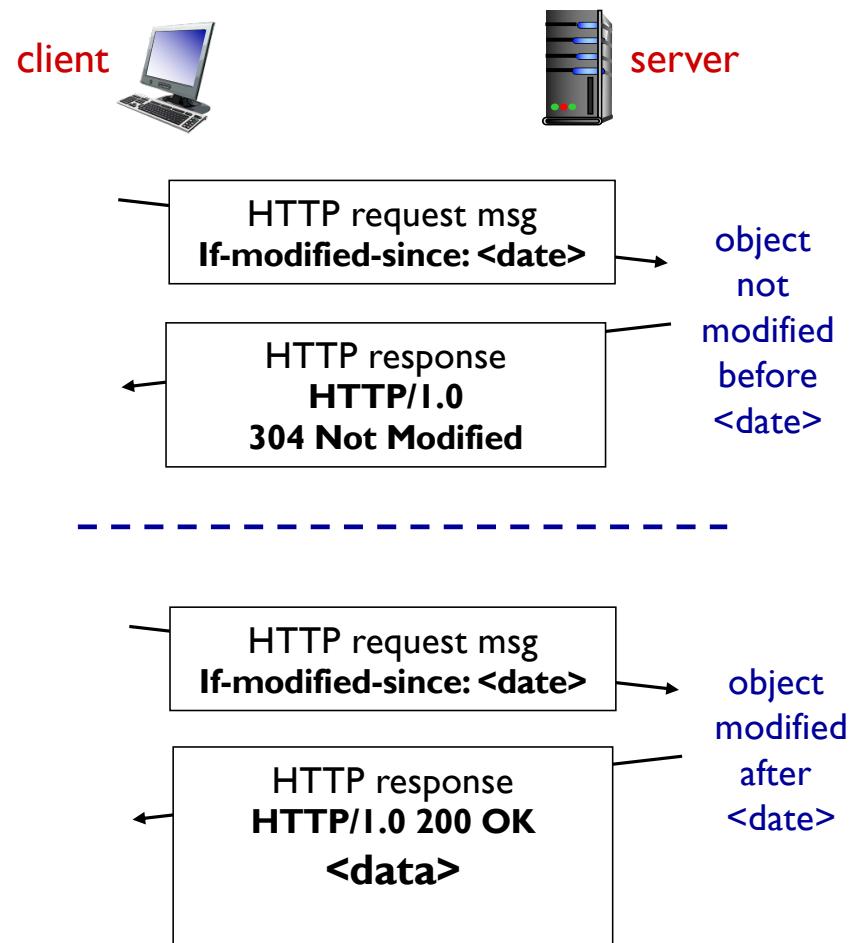


*lower average end-end delay than with 154 Mbps link (and cheaper too!)*

# Conditional GET

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

- no object transmission delay
  - lower link utilization
- **cache:** specify date of cached copy in HTTP request  
**If-modified-since: <date>**
  - **server:** response contains no object if cached copy is up-to-date:  
**HTTP/1.0 304 Not Modified**



# Example Cache Check Request

GET / HTTP/1.1

Accept: \*/\*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

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

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

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

Host: www.intel-iris.net

Connection: Keep-Alive

# Example Cache Check Response

HTTP/1.1 304 Not Modified

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

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

Connection: Keep-Alive

Keep-Alive: timeout=15, max=100

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

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

# Improving HTTP Performance: Replication

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

More on this later

More on this later

## Improving HTTP Performance: CDN

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

# What about HTTPS?



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

# HTTP/2

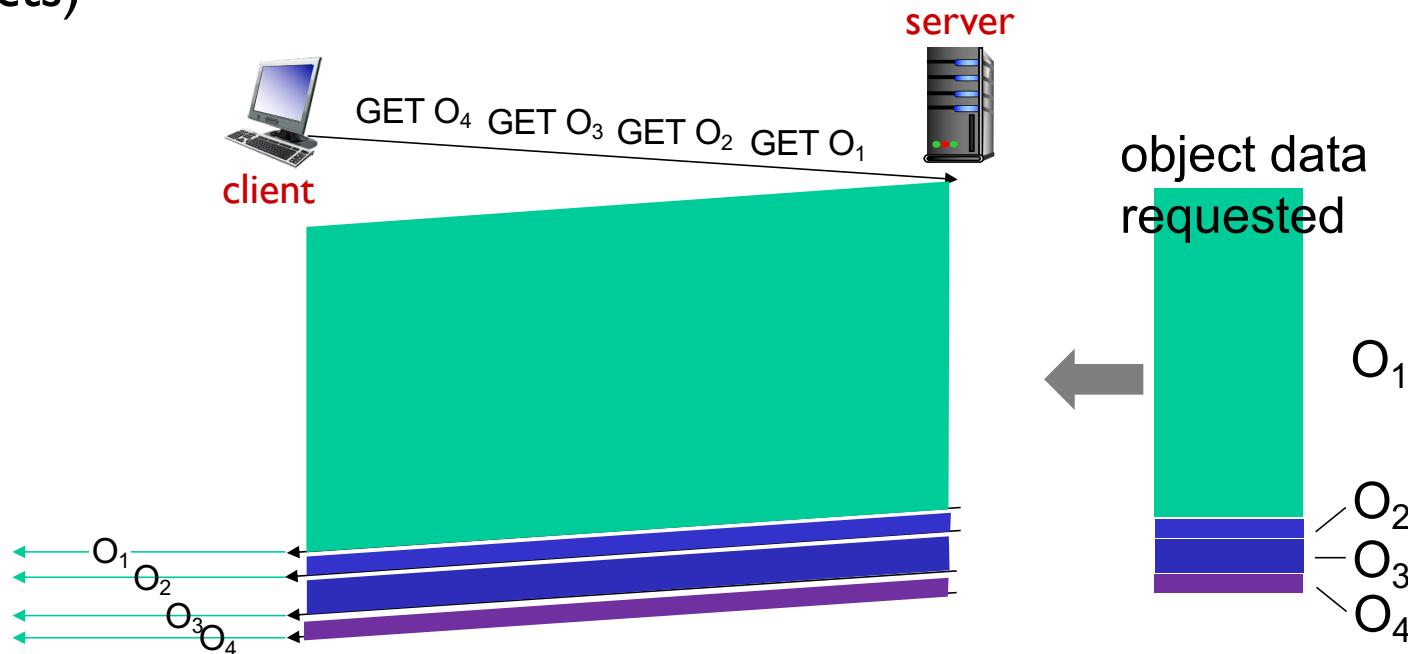
*Key goal:* decreased delay in multi-object HTTP requests

HTTP 1.1: 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)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

# HTTP/2: mitigating HOL blocking

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



*objects delivered in order requested: O<sub>2</sub>, O<sub>3</sub>, O<sub>4</sub> wait behind O<sub>1</sub>*

# HTTP/2

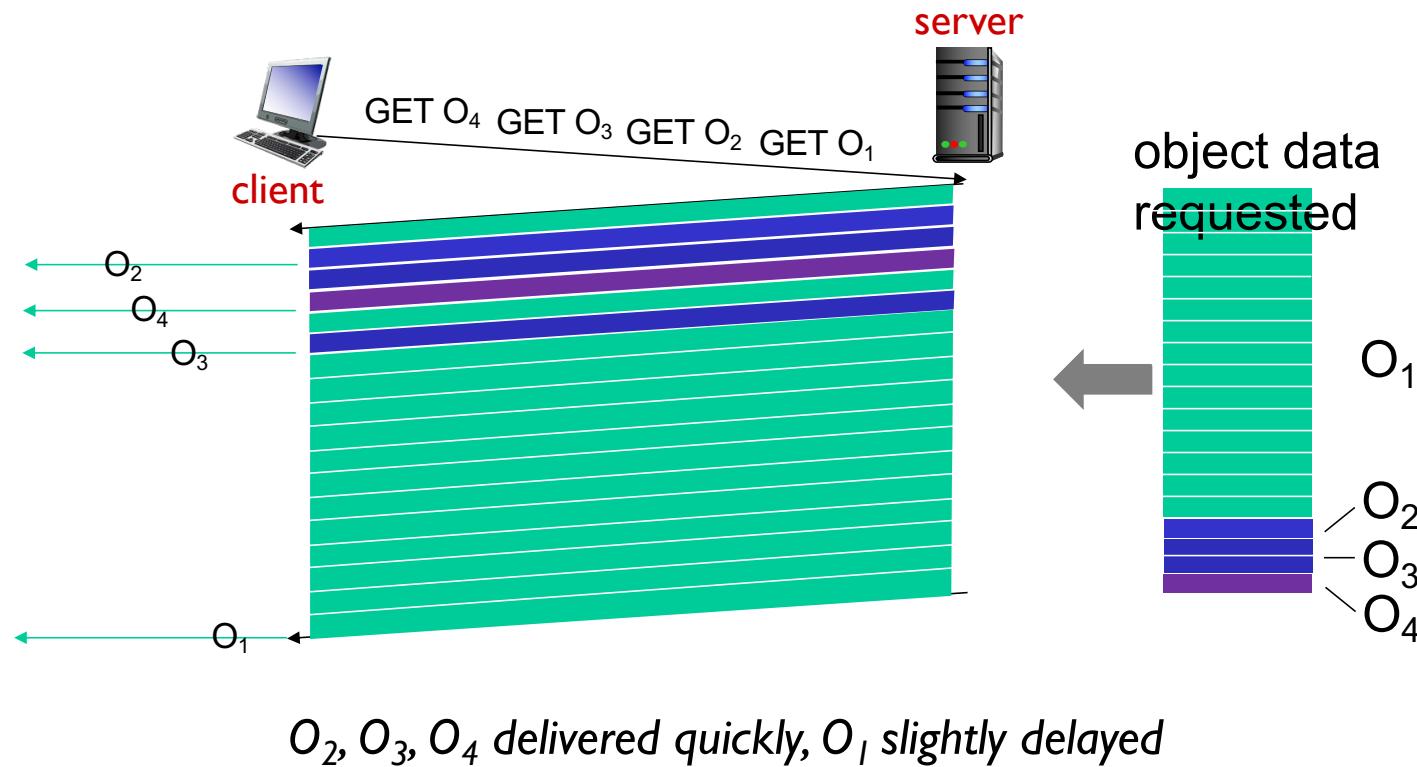
*Key goal:* decreased delay in multi-object HTTP requests

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
- transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
- *push* unrequested objects to client
- divide objects into frames, schedule frames to mitigate HOL blocking

# HTTP/2: mitigating HOL blocking

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

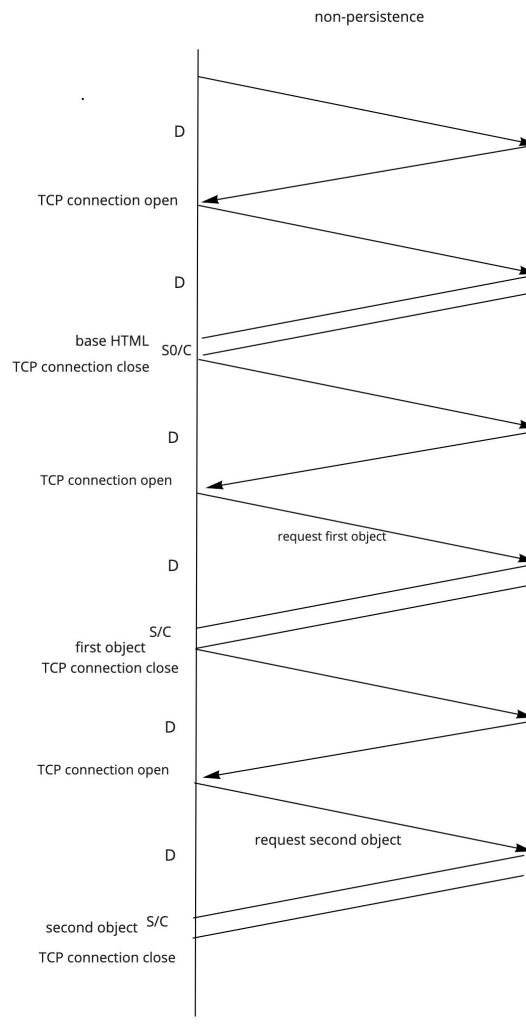




## Quiz: HTTP (1)

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

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



N=2

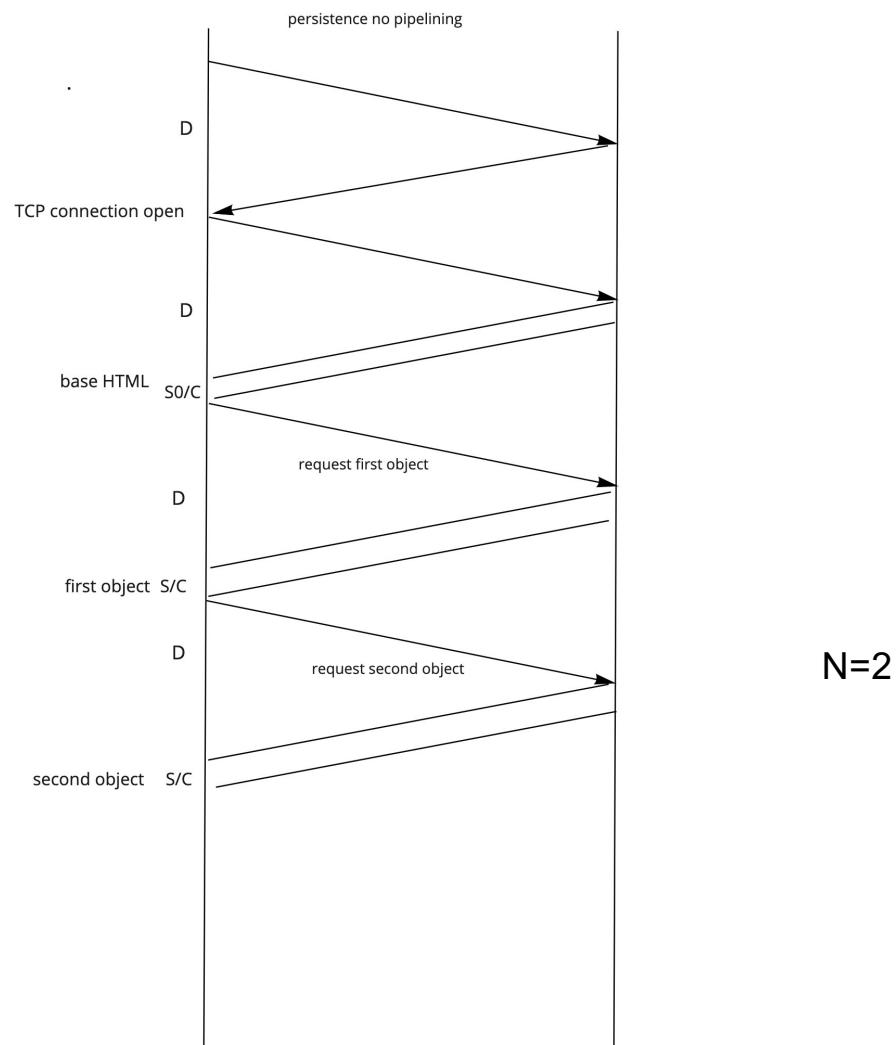
miro



## Quiz: HTTP (2)

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

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



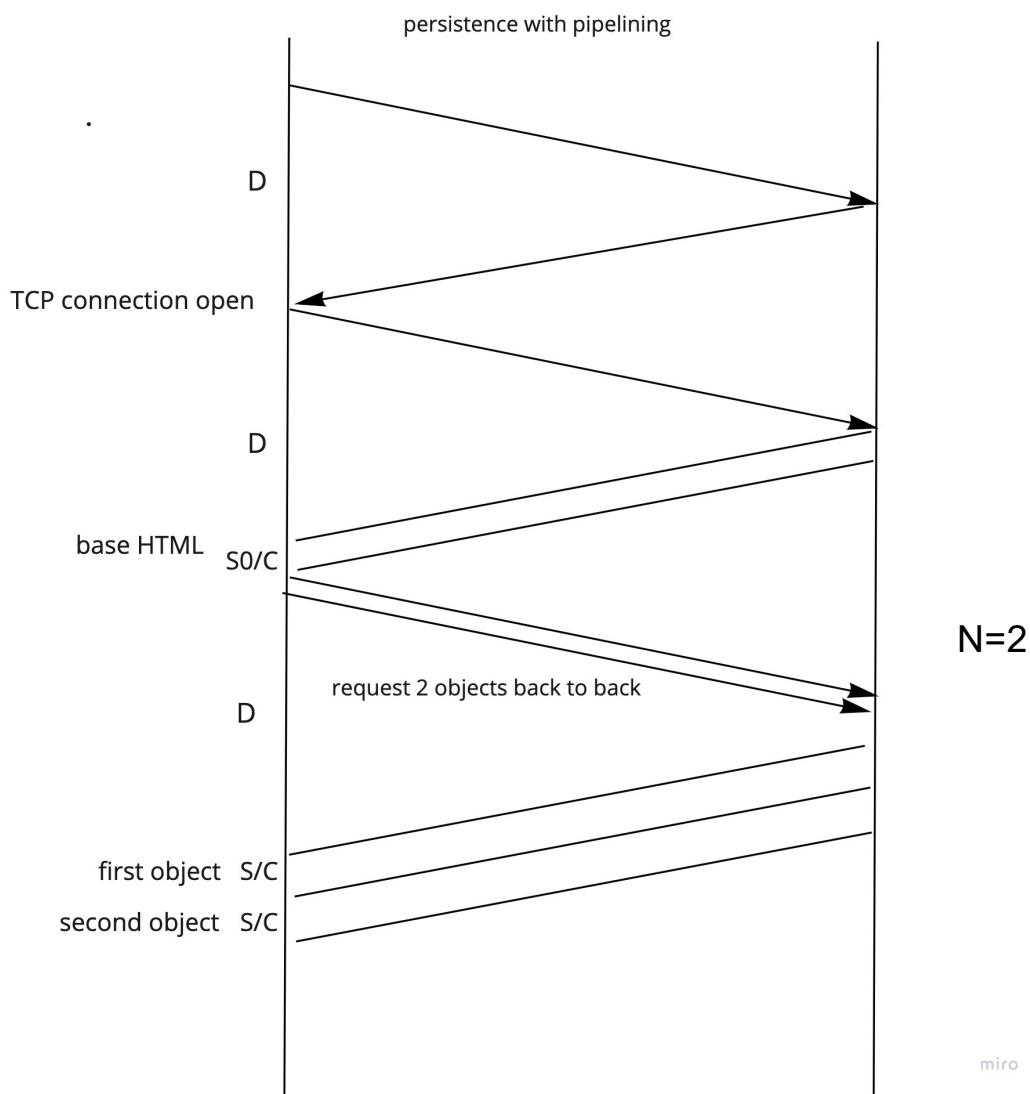
micro



## Quiz: HTTP (3)

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

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



## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

**2.3 electronic mail**

- SMTP, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

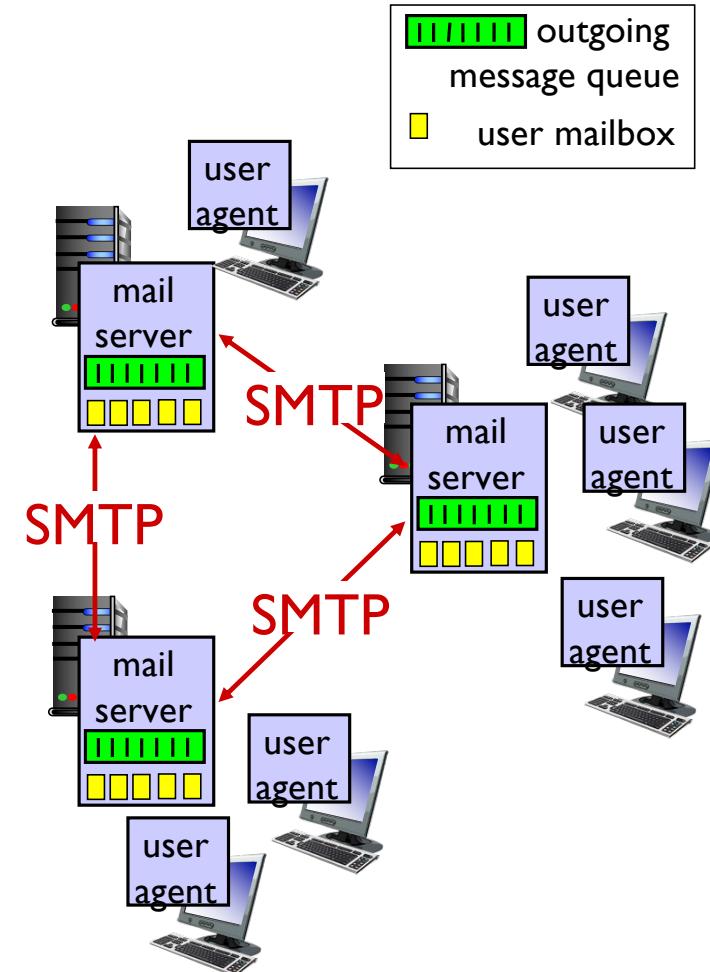
# E-mail

Three major components:

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

## User Agent

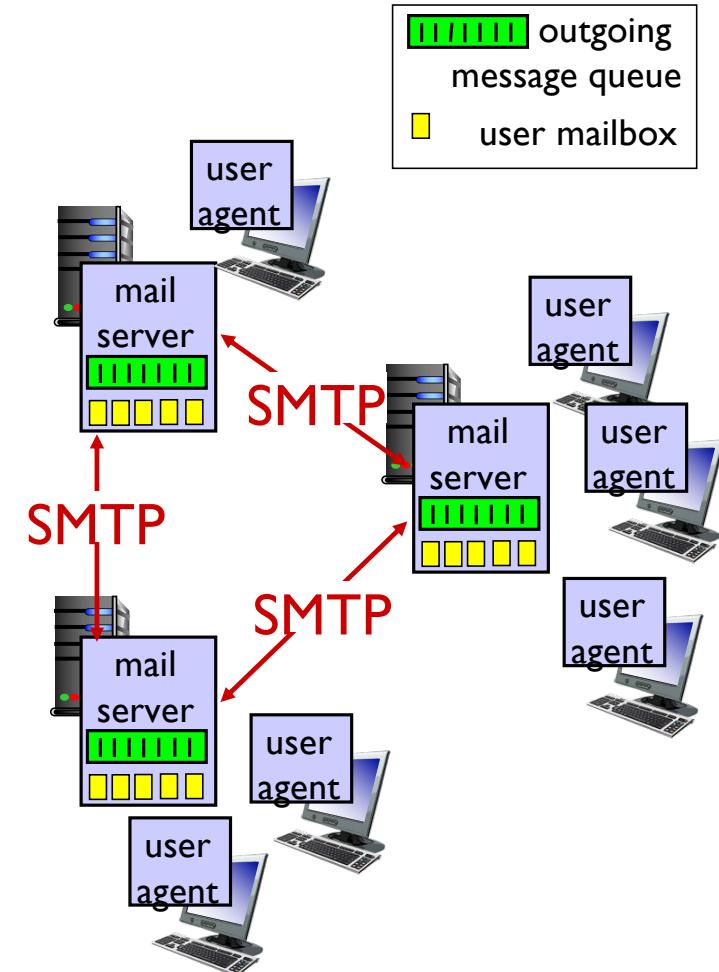
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



# E-mail: mail servers

## mail servers:

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

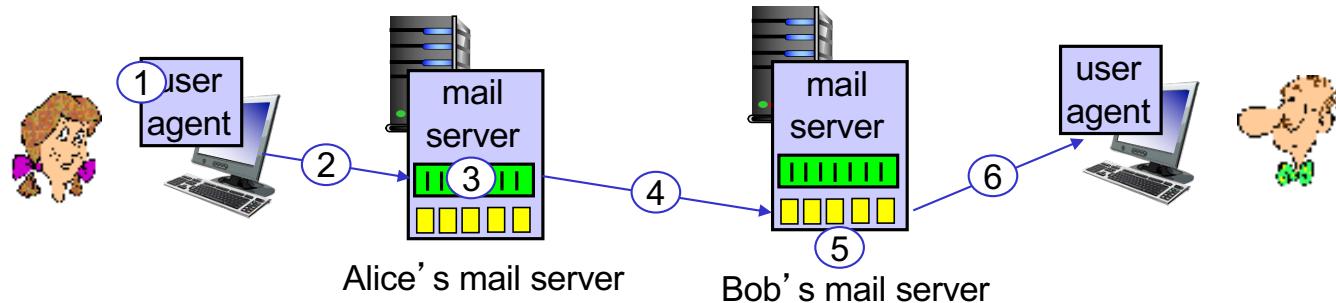


## E-mail: the RFC (5321)

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

# Scenario: Alice sends e-mail to Bob

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



# Sample SMTP interaction

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

# SMTP: closing observations

## *comparison with HTTP:*

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message
- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine end of message

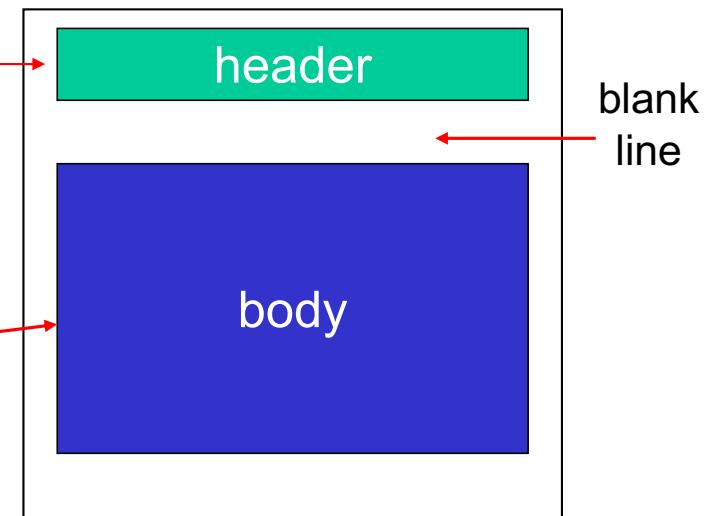
# Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 531 (like HTTP)

RFC 822 defines syntax for e-mail message itself (like HTML)

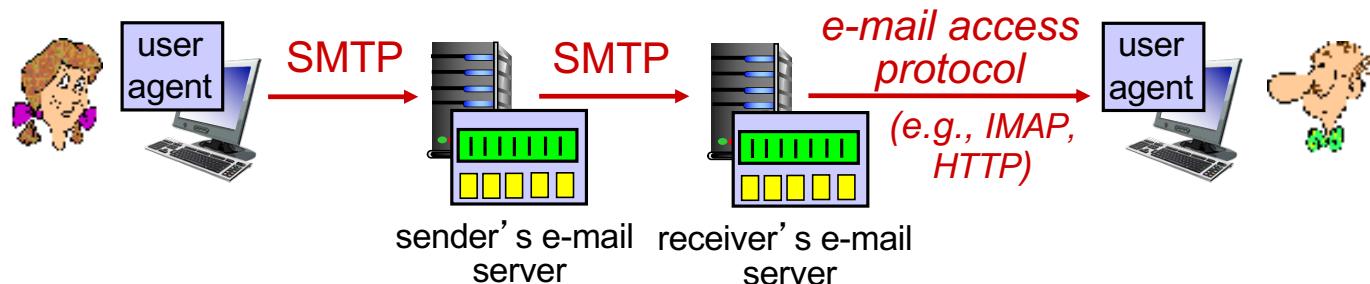
- header lines, e.g.,
  - To:
  - From:
  - Subject:

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



POP/IMAP Not on exam

# Mail access protocols



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

## Quiz: SMTP

Why do we have Sender's mail server?

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

## Quiz: SMTP

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

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

# Summary

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



# Computer Networks and Applications

COMP 3331/COMP 9331

Week 3

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

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

## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

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

# DNS: Domain Name System

*people:* many identifiers:

- TFN, name, passport #

*Internet hosts, routers:*

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

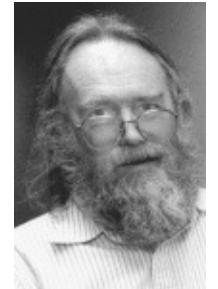
Q: how to map between IP address and name, and vice versa ?

**Domain Name System:**

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

# DNS: History

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



Jon Postel

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

# DNS: services, structure

## DNS services

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

## *Q: Why not centralize DNS?*

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

## *A: doesn't scale!*

- Comcast DNS servers alone: 600B DNS queries per day

# Goals

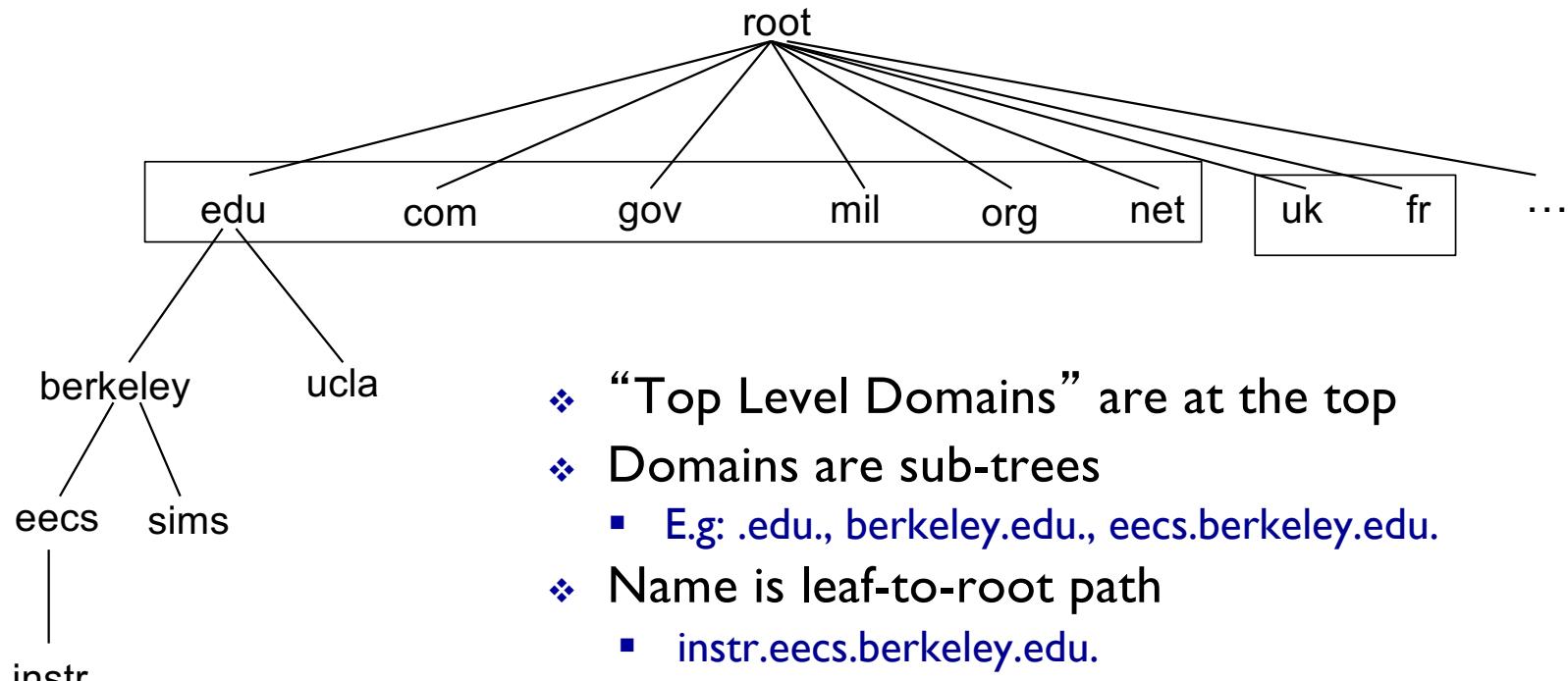
- ❖ No naming conflicts (uniqueness)
- ❖ Scalable
  - many names
  - (secondary) frequent updates
- ❖ Distributed, autonomous administration
  - Ability to update my own (domains') names
  - Don't have to track everybody's updates
- ❖ Highly available
- ❖ Lookups should be fast

# Key idea: Hierarchy

Three intertwined hierarchies

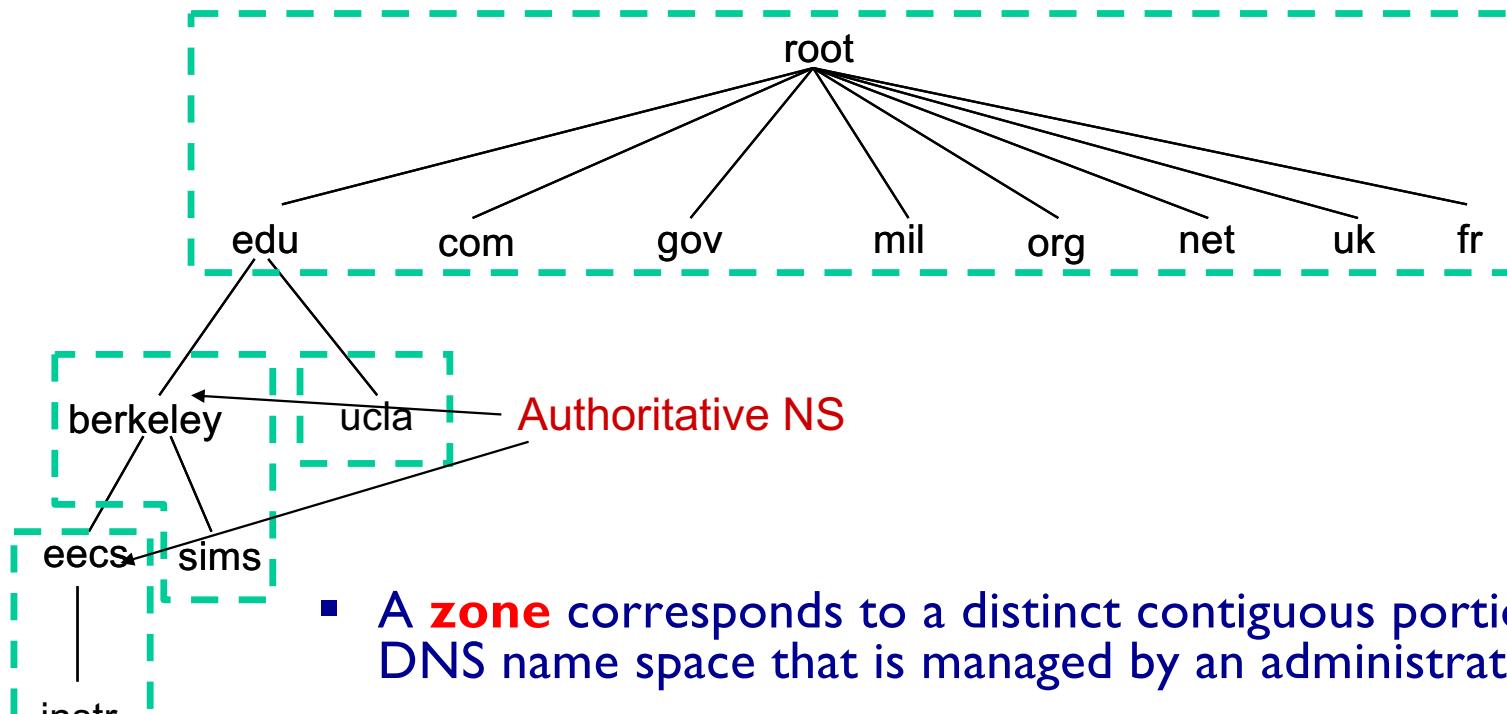
- Hierarchical namespace
  - As opposed to original flat namespace
- Hierarchically administered
  - As opposed to centralised
- (Distributed) hierarchy of servers
  - As opposed to centralised storage

# Hierarchical Namespace



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

# Hierarchical Administration



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

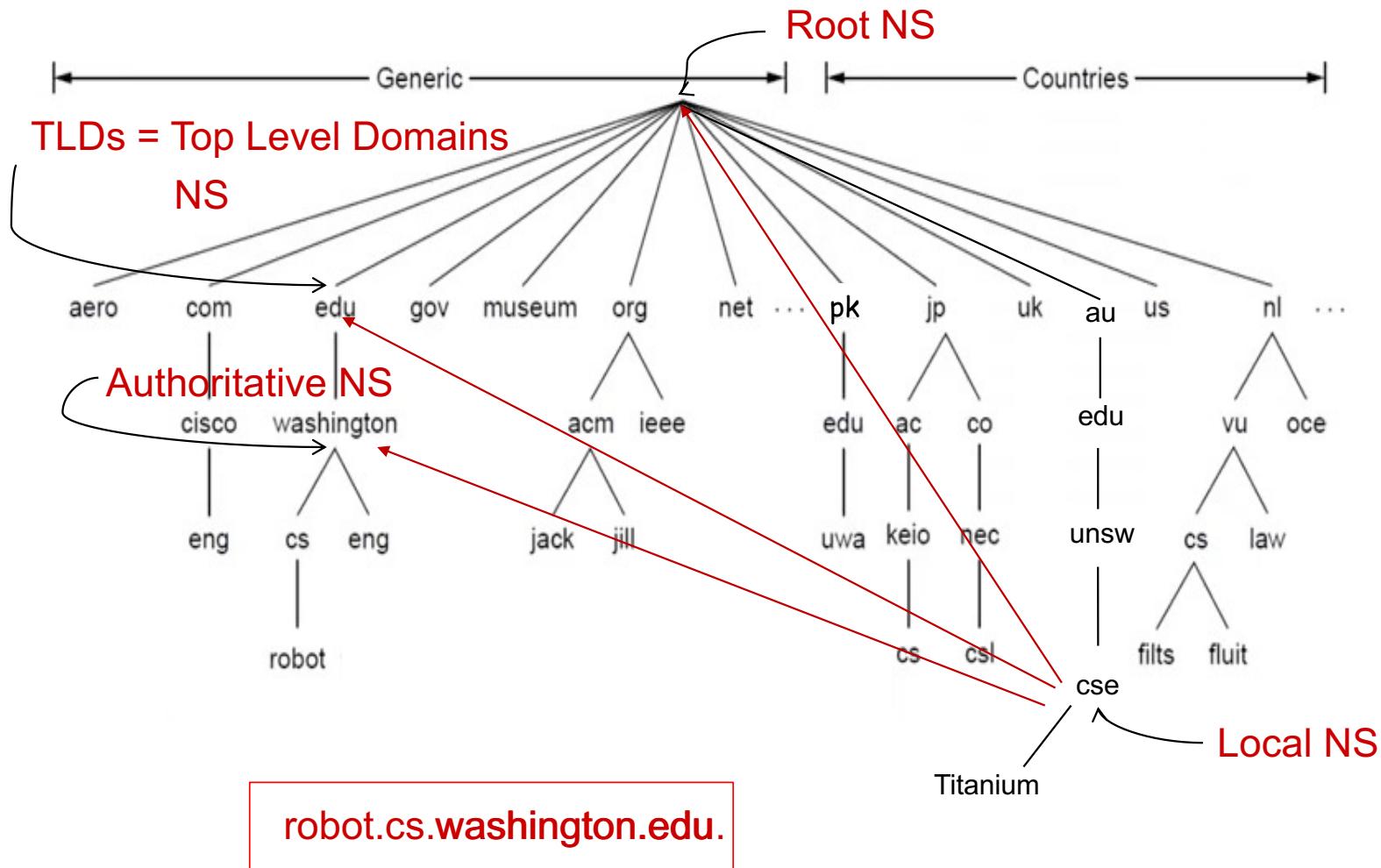
# Server Hierarchy

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

# Server Hierarchy

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

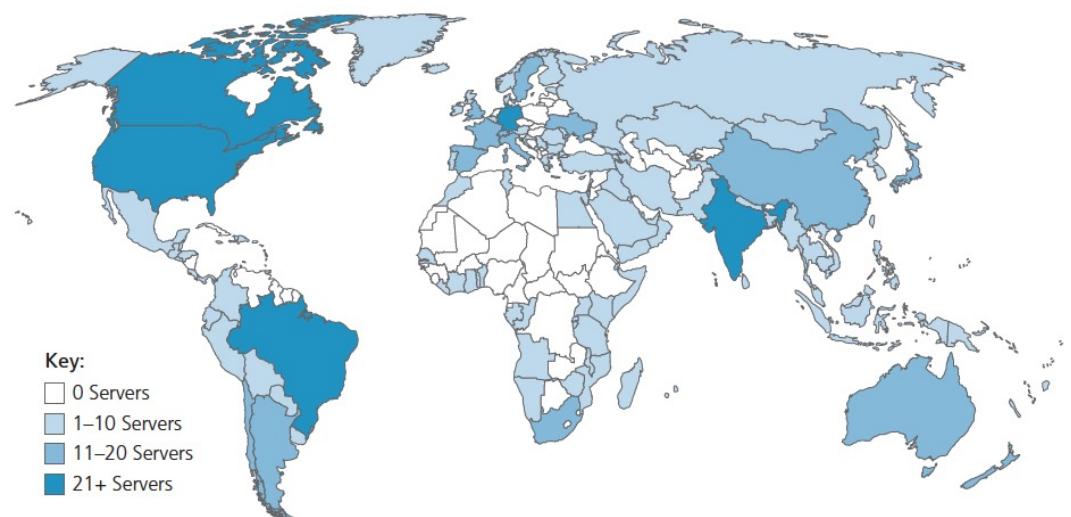
# DNS: a distributed, hierarchical database



# DNS: root name servers

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

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



# DNS: root name servers



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



# TLD: authoritative servers

## Top-Level Domain (TLD) servers:

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

## Authoritative DNS servers:

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

# Local DNS name servers

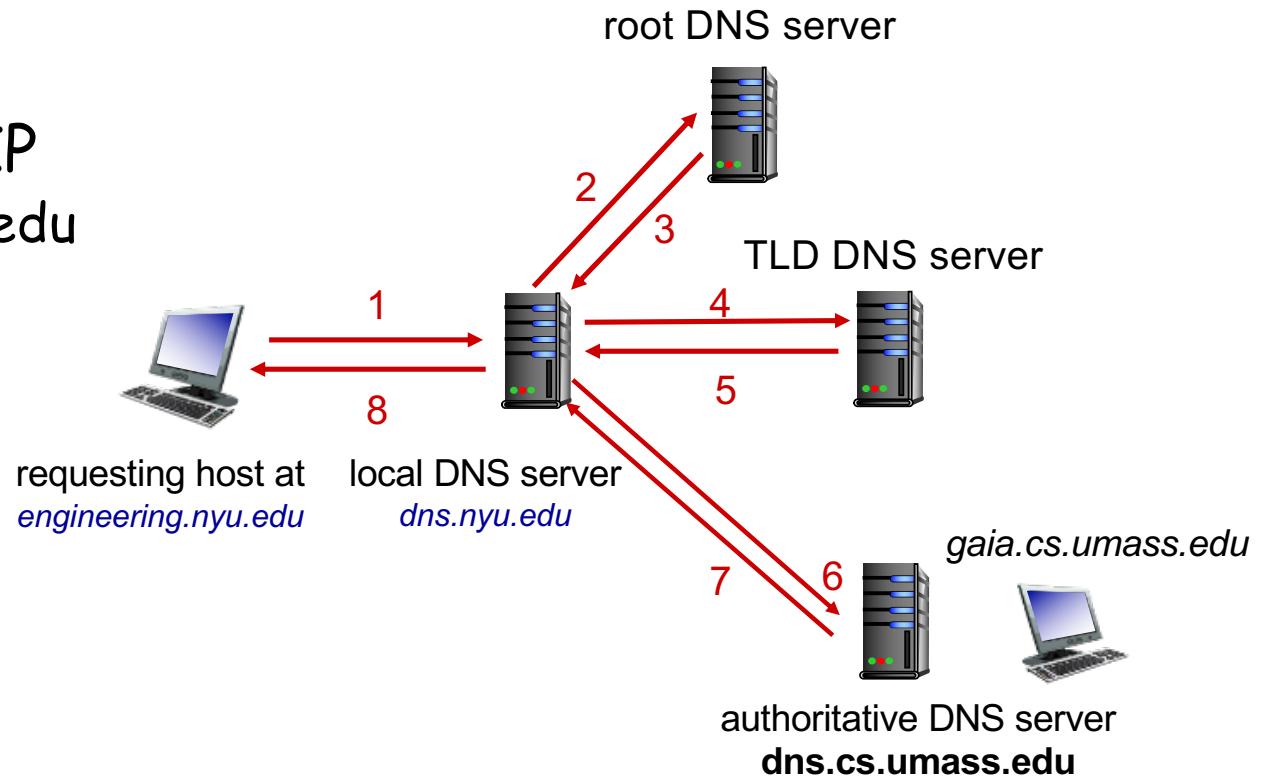
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- Hosts learn about the local DNS server via a host configuration protocol (e.g., DHCP)
- Client application
  - Obtain hostname (e.g., from URL)
  - Do `gethostname()`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

# DNS name resolution: iterated query

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

## Iterated query:

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

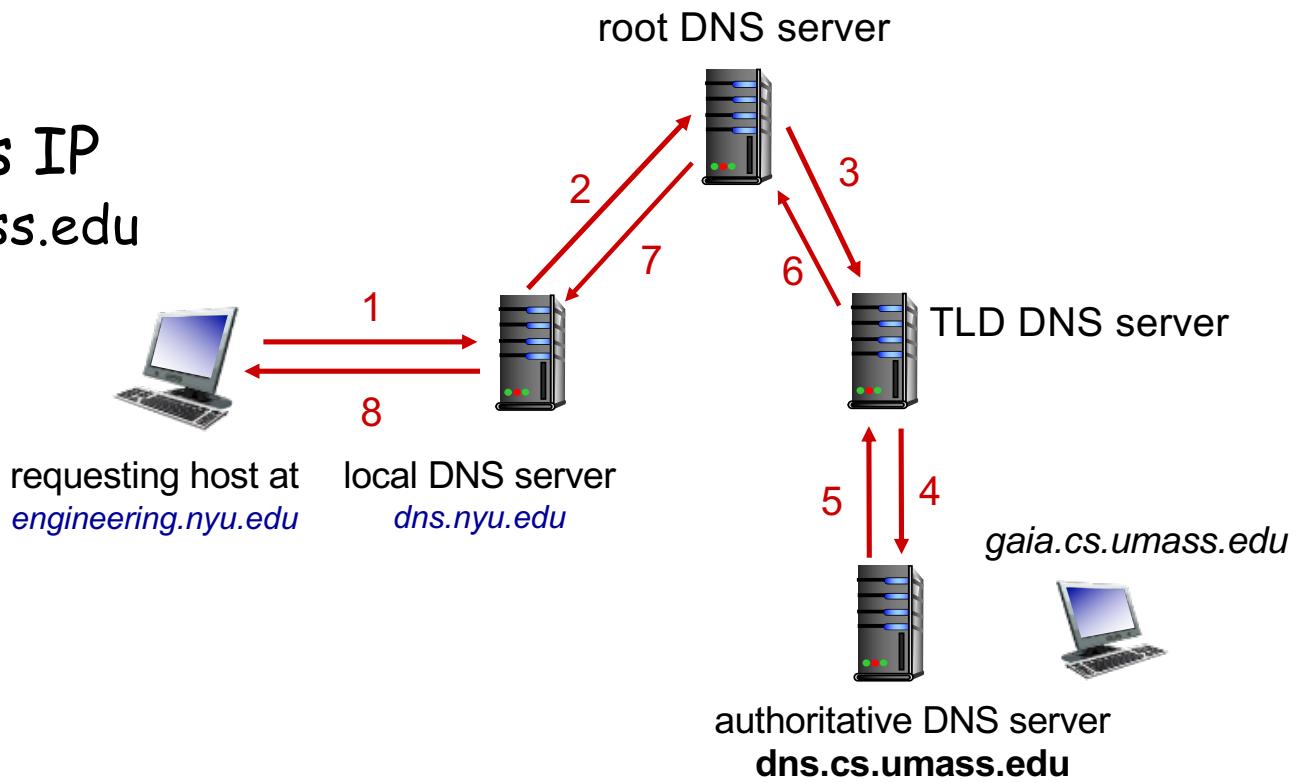


# DNS name resolution: recursive query

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

## Recursive query:

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



# Caching, Updating DNS Records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best-effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire!
- update/notify mechanisms proposed IETF standard
  - RFC 2136
- Negative caching (optional)
  - Remember things that don't work
  - E.g., misspellings like www.cnn.comm and www.cnnn.com

# DNS records

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

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

## type=A

- name is hostname
- value is IP address

## type=NS

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

## type=CNAME

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

## type=MX

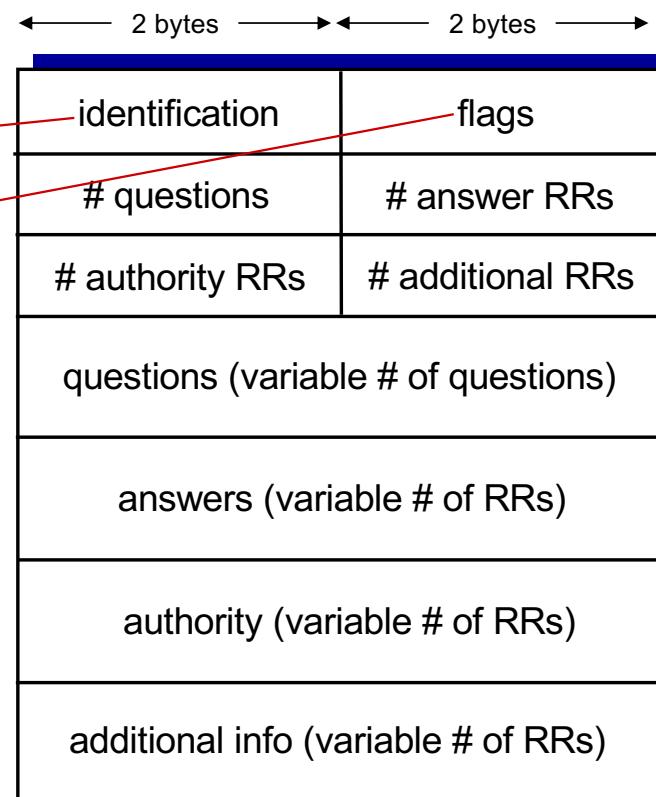
- value is name of mailserver associated with name

# DNS protocol messages

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

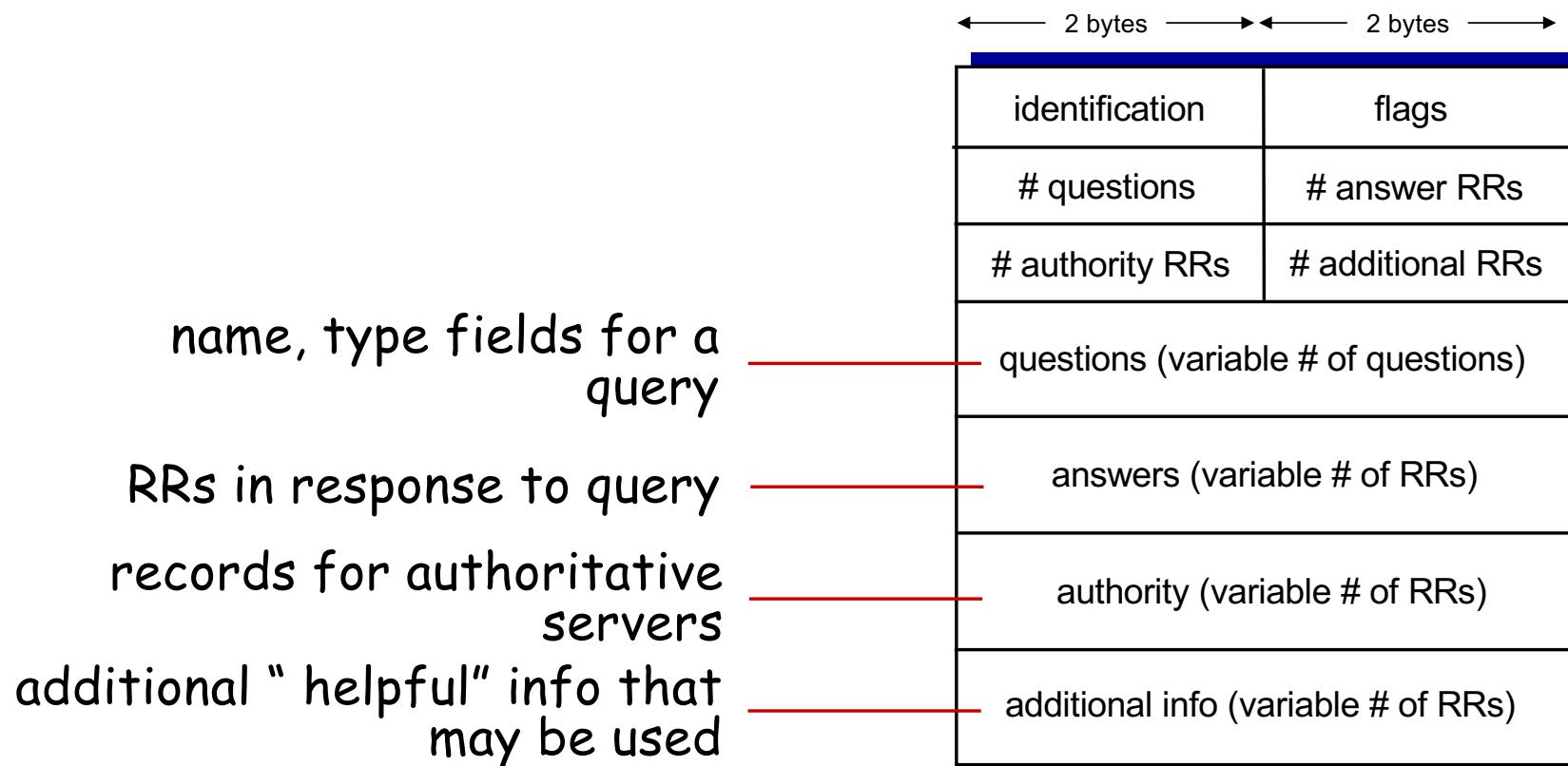
message header:

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



# DNS protocol messages

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



name, type fields for a query

RRs in response to query

records for authoritative servers

additional "helpful" info that may be used

# An Example

Try this out yourself. Part of Lab 3

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

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

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

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

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

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

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

# Inserting records into DNS

Example: new startup “Network Utopia”

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

# Updating DNS records

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

# Reliability

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

# CDN example (more later)

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

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

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

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

;; AUTHORITY SECTION:
dsrb.akamaiedge.net. 623    IN      NS      n0dsrb.akamaiedge.net.
dsrb.akamaiedge.net. 623    IN      NS      n2dsrb.akamaiedge.net.
dsrb.akamaiedge.net. 623    IN      NS      n7dsrb.akamaiedge.net.
dsrb.akamaiedge.net. 623    IN      NS      n6dsrb.akamaiedge.net.
dsrb.akamaiedge.net. 623    IN      NS      n1dsrb.akamaiedge.net.
dsrb.akamaiedge.net. 623    IN      NS      n3dsrb.akamaiedge.net.
dsrb.akamaiedge.net. 623    IN      NS      n5dsrb.akamaiedge.net.
dsrb.akamaiedge.net. 623    IN      NS      n4dsrb.akamaiedge.net.

;; ADDITIONAL SECTION:
n0dsrb.akamaiedge.net. 1241   IN      A       88.221.81.192
n0dsrb.akamaiedge.net. 1124   IN      AAAA   2600:1480:e800::c0
n1dsrb.akamaiedge.net. 842    IN      A       23.32.5.76
n2dsrb.akamaiedge.net. 749    IN      A       23.32.5.84
n4dsrb.akamaiedge.net. 1399   IN      A       23.32.5.177
n6dsrb.akamaiedge.net. 702    IN      A       23.32.5.98
n7dsrb.akamaiedge.net. 1208   IN      A       23.206.243.54

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

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

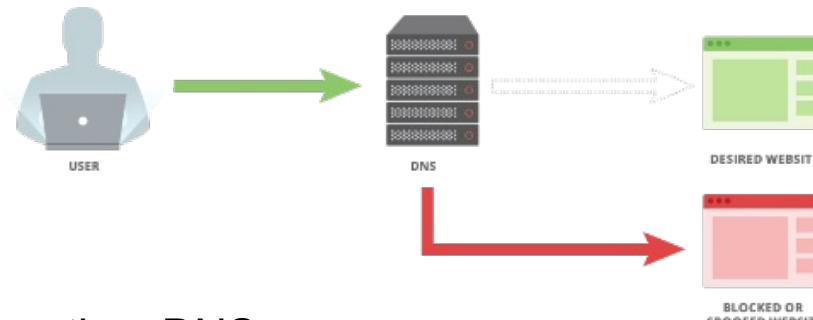
# WWW vs non-WWW domains

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

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

# Do you trust your DNS server?

- ❖ Censorship



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

- ❖ Logging

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

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

# 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

## Redirect attacks

- man-in-middle
  - intercept DNS queries
- DNS poisoning
  - send bogus replies to DNS server, which caches

## Exploit DNS for DDoS

- send queries with spoofed source address: target IP
- requires amplification

DNSSEC  
[RFC 4033]



# DNS Cache Poisoning



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

;; QUESTION SECTION:

;www.drevil.com. IN A

;; ANSWER SECTION:

www.drevil.com 300 IN A 129.45.212.42

;; AUTHORITY SECTION:

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

drevil.com 86400 IN NS google.com

;; ADDITIONAL SECTION:

google.com 600 IN A 129.45.212.222

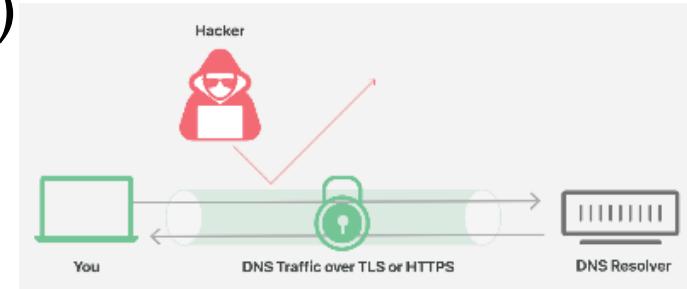
A drevil.com machine, **not** google.com

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

NOT ON EXAM

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

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



# Quiz: DNS (I)



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

## Quiz: DNS (2)



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



## Quiz: DNS (3)

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



## Quiz: DNS (4)

- ❖ You open your browser and type [www.pollev.com](http://www.pollev.com). The minimum number of DNS requests sent by your local DNS server to obtain the corresponding IP address is:
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 42

## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP

2.4 DNS

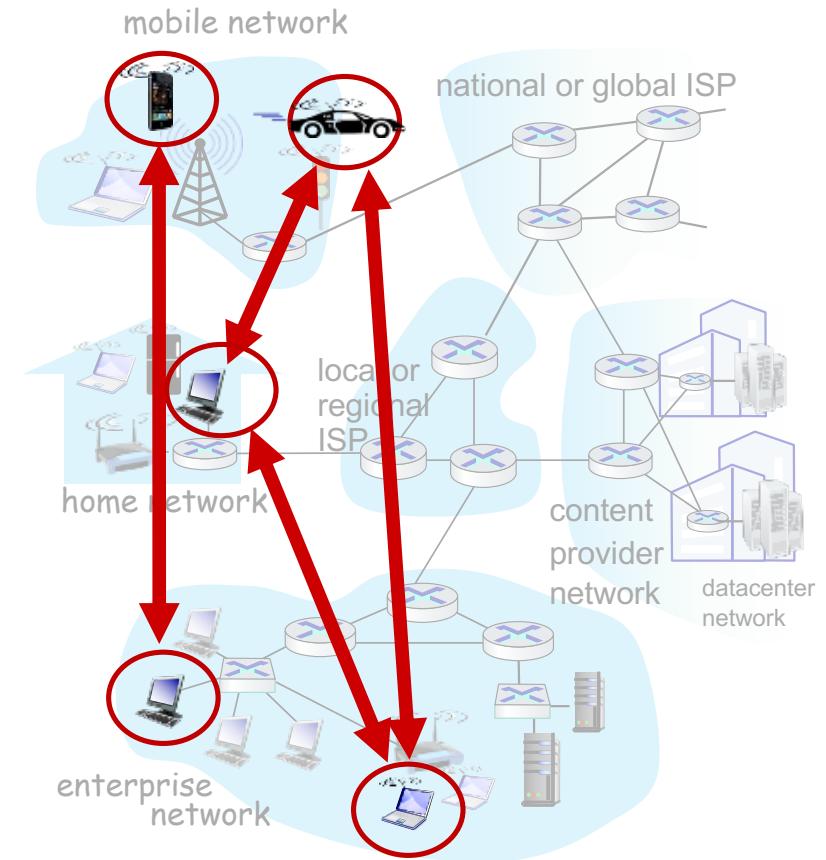
2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# Peer-to-peer (P2P) architecture

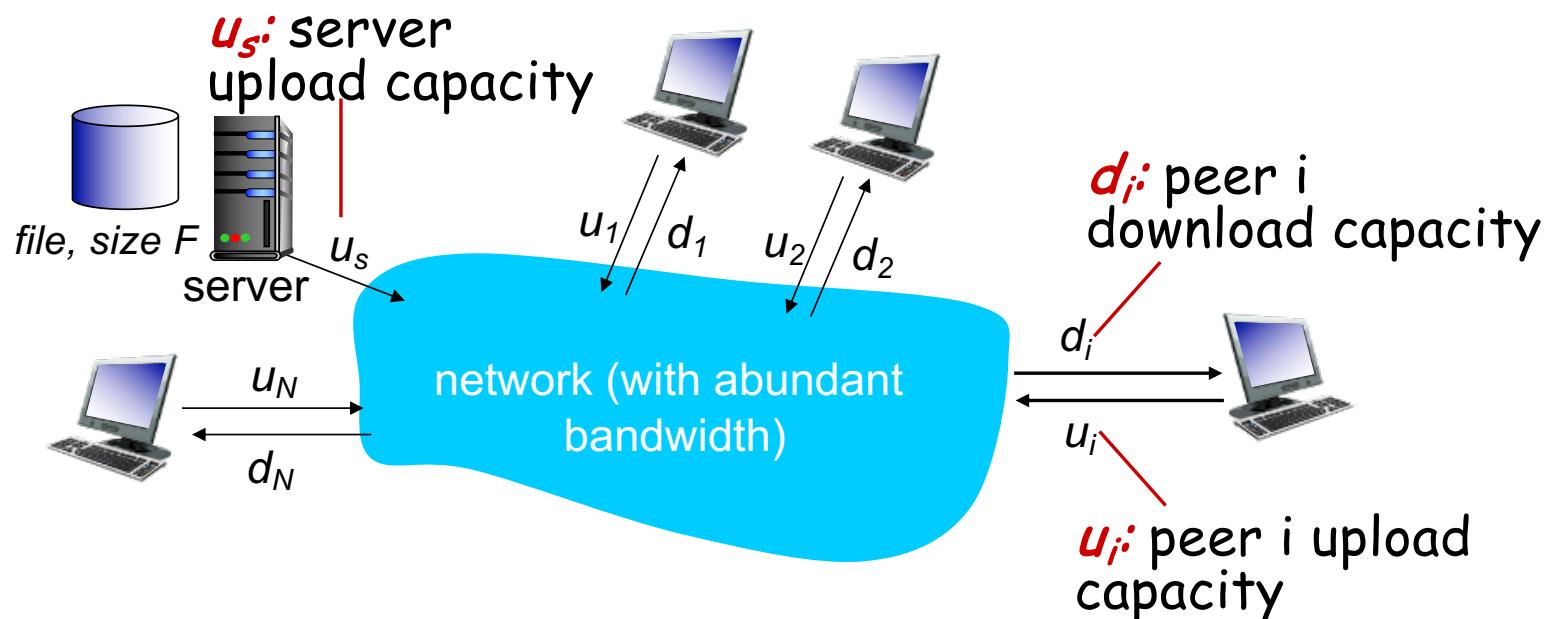
- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* - new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
  -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



# File distribution: client-server vs P2P

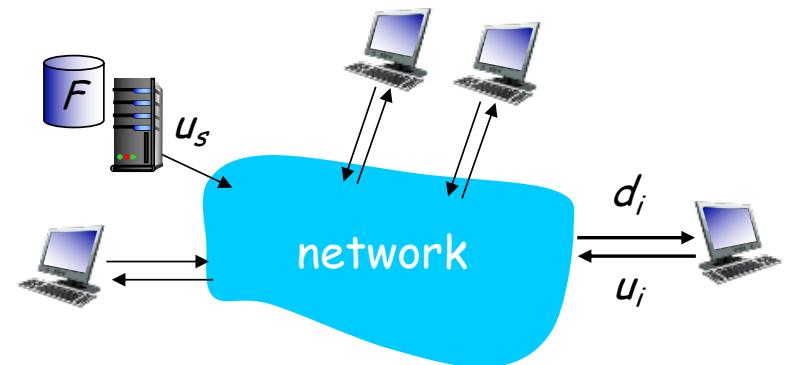
**Q:** how much time to distribute file (size  $F$ ) from one server to  $N$  peers?

- peer upload/download capacity is limited resource



# File distribution time: client-server

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

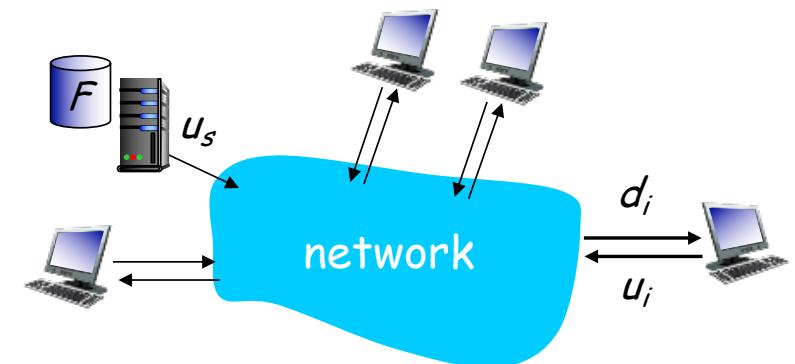


time to distribute  $F$   
to  $N$  clients using  $D_{c-s} \rightarrow \max\{NF/u_s, F/d_{min}\}$   
client-server approach

increases linearly in  $N$

# File distribution time: P2P

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



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

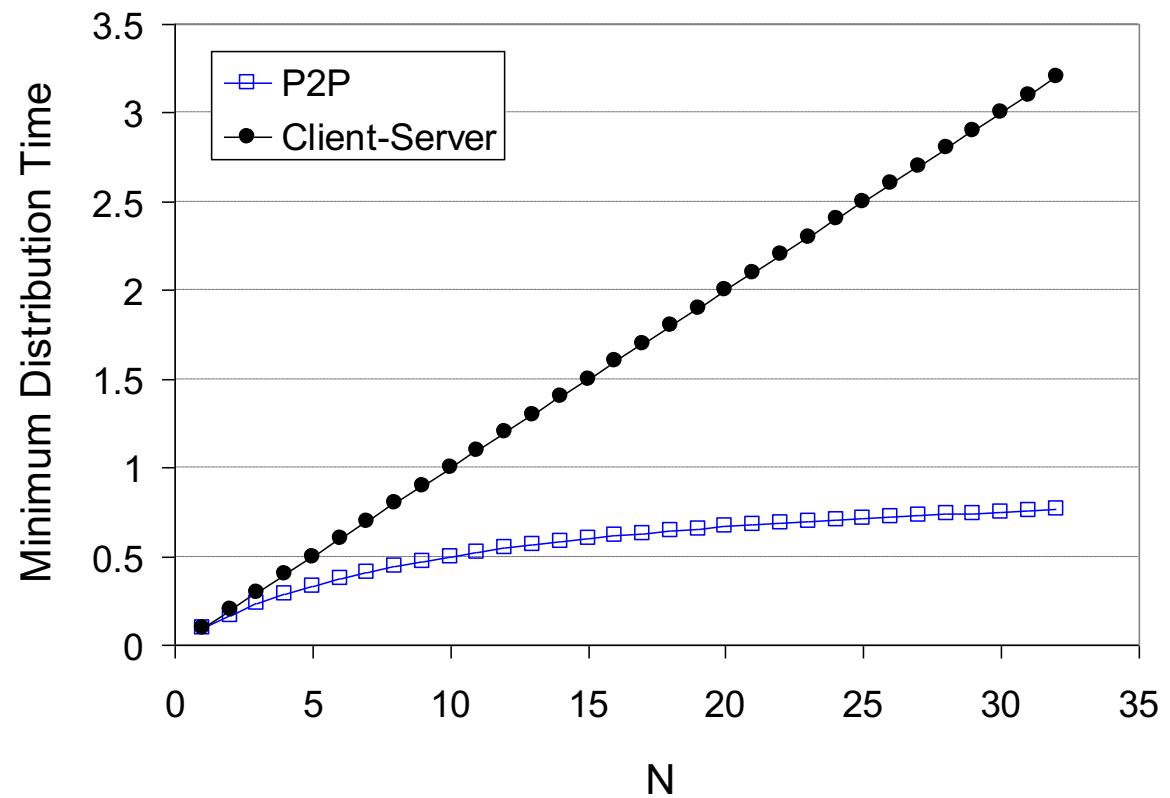
$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

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

increases linearly in  $N$  ...

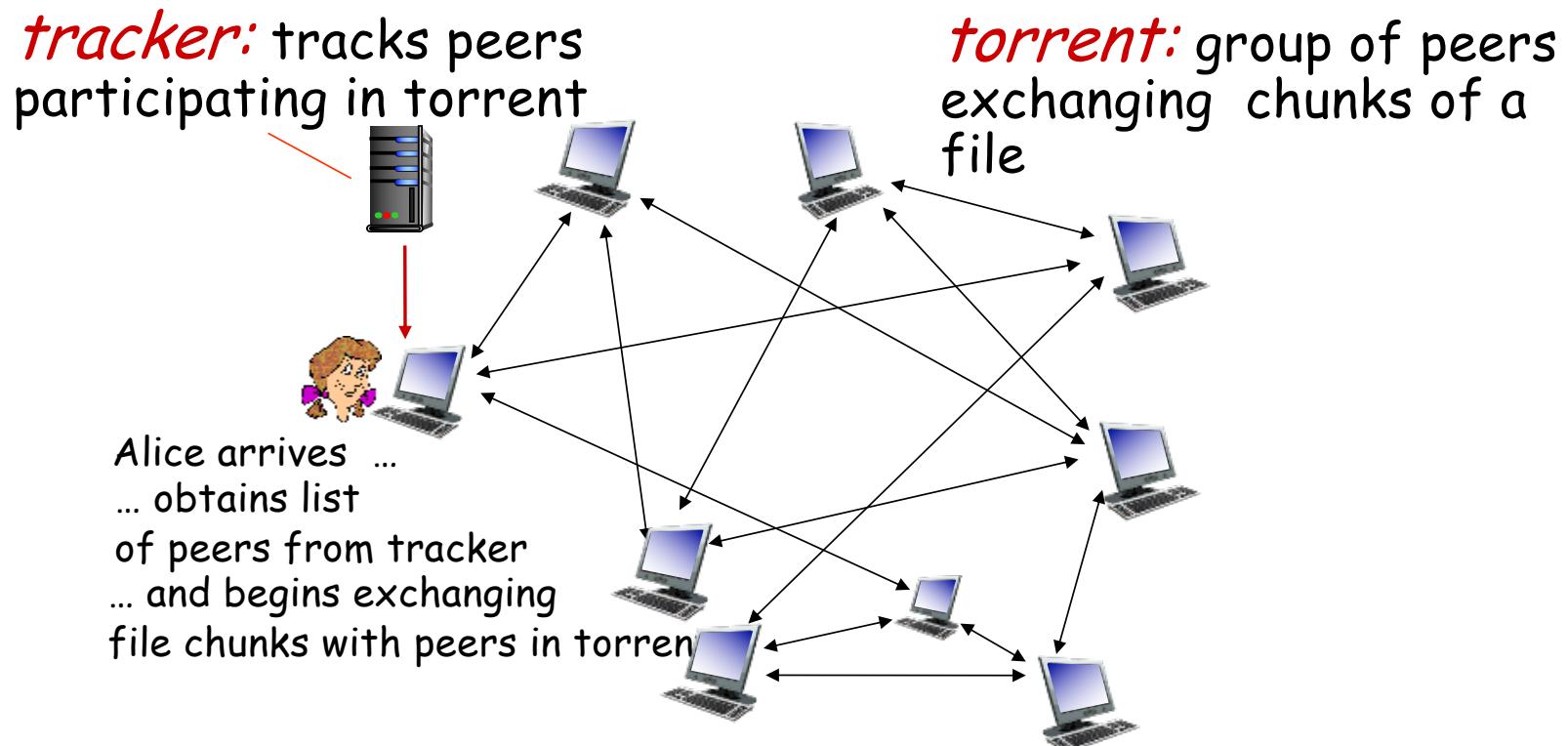
# Client-server vs. P2P: example

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



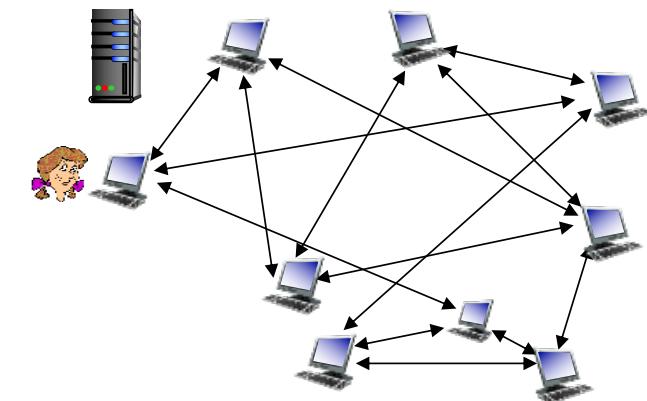
# P2P file distribution: BitTorrent

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



# P2P file distribution: BitTorrent

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



# BitTorrent: requesting, sending file chunks

## Requesting chunks:

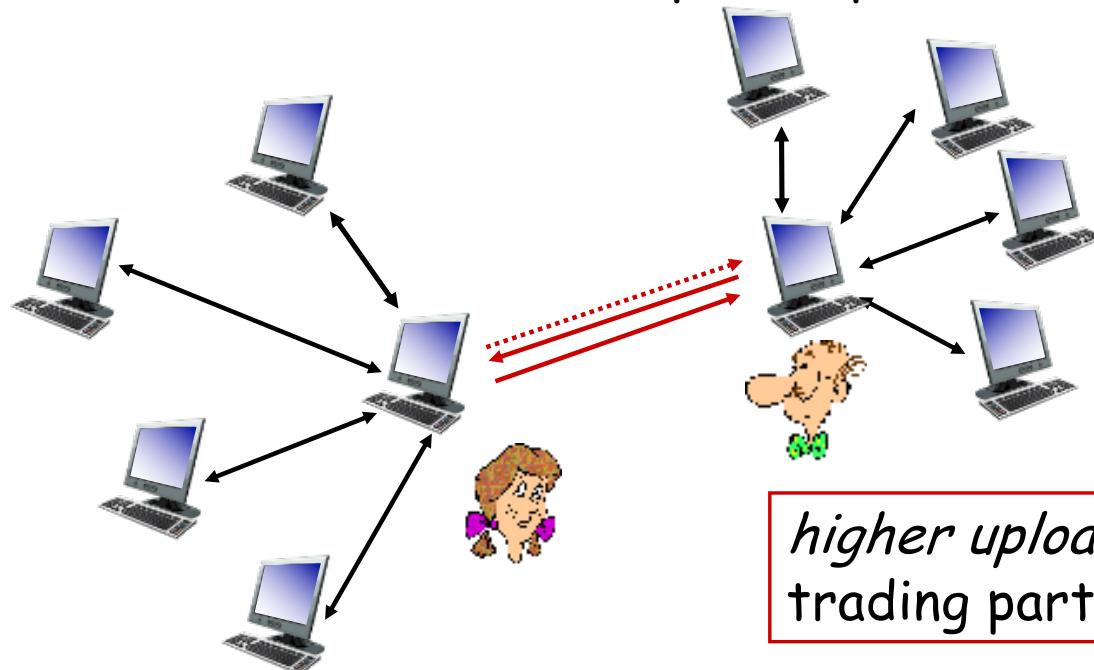
- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first (why?)

## Sending chunks: tit-for-tat

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

# BitTorrent: tit-for-tat

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



## Quiz: BitTorrent

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

## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

# Video Streaming and CDNs: context

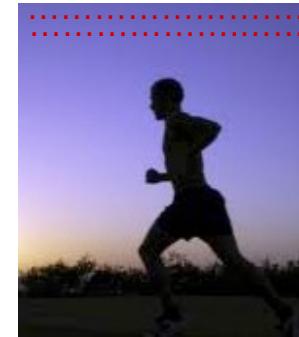
- stream video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- challenge: scale - how to reach ~1B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution: distributed, application-level infrastructure*



# Multimedia: video

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

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



frame  $i$



frame  $i+1$

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

# Multimedia: video

- **CBR: (constant bit rate):** video encoding rate fixed
- **VBR: (variable bit rate):** video encoding rate changes as amount of spatial, temporal coding changes
- **examples:**
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, 64Kbps - 12 Mbps)

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



frame  $i$

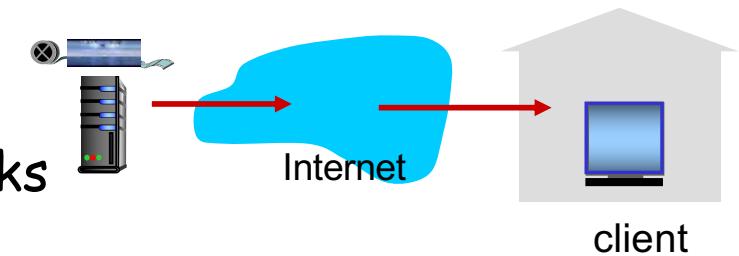
*temporal coding example:* instead of sending complete frame at  $i+1$ , send only differences from frame  $i$



frame  $i+1$

# Streaming multimedia: DASH

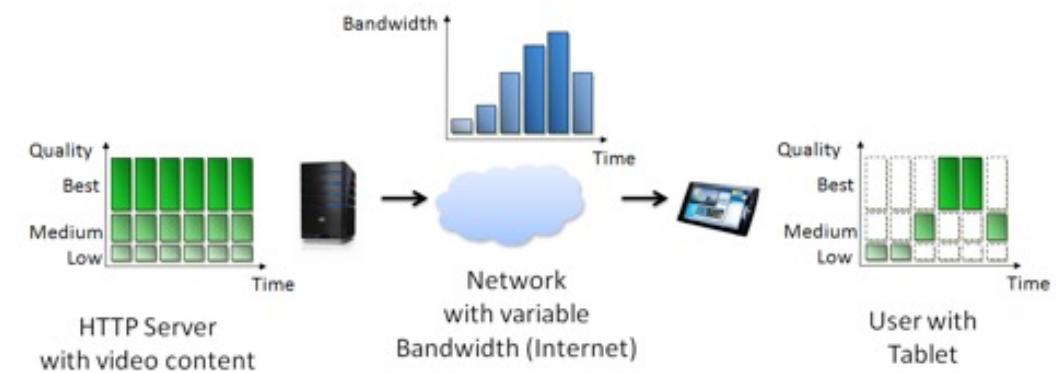
- **DASH: Dynamic, Adaptive Streaming over HTTP**
- **server:**
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file*: provides URLs for different chunks
- **client:**
  - periodically measures server-to-client bandwidth
  - consulting manifest,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



# Streaming multimedia: DASH

- “*intelligence*” at client: client determines

- *when* to request chunk (so that buffer starvation, or overflow does not occur)
- *what encoding rate* to request (higher quality when more bandwidth available)
- *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)



Streaming video = encoding + DASH + playout buffering

# Content distribution networks (CDNs)

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

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

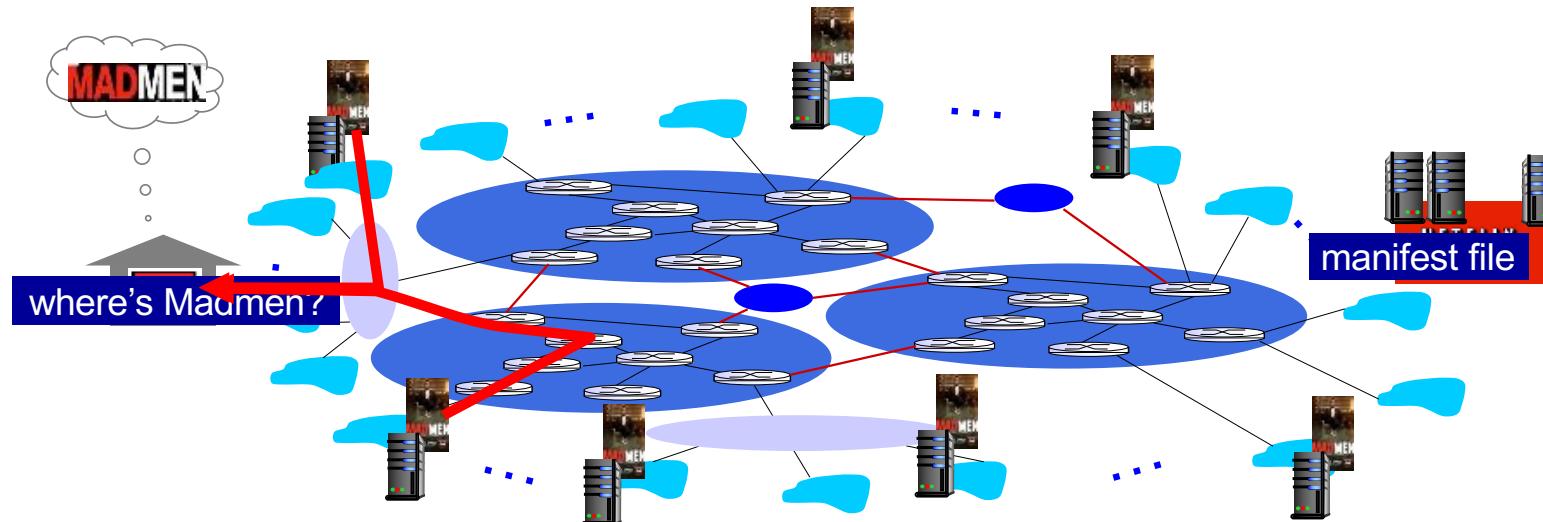
# Content distribution networks (CDNs)

- **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- **option 2:** store/serve multiple copies of videos at multiple geographically distributed sites (*CDN*)
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - Akamai: 240,000 servers deployed in more than 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight



# Content distribution networks (CDNs)

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



# Content distribution networks (CDNs)



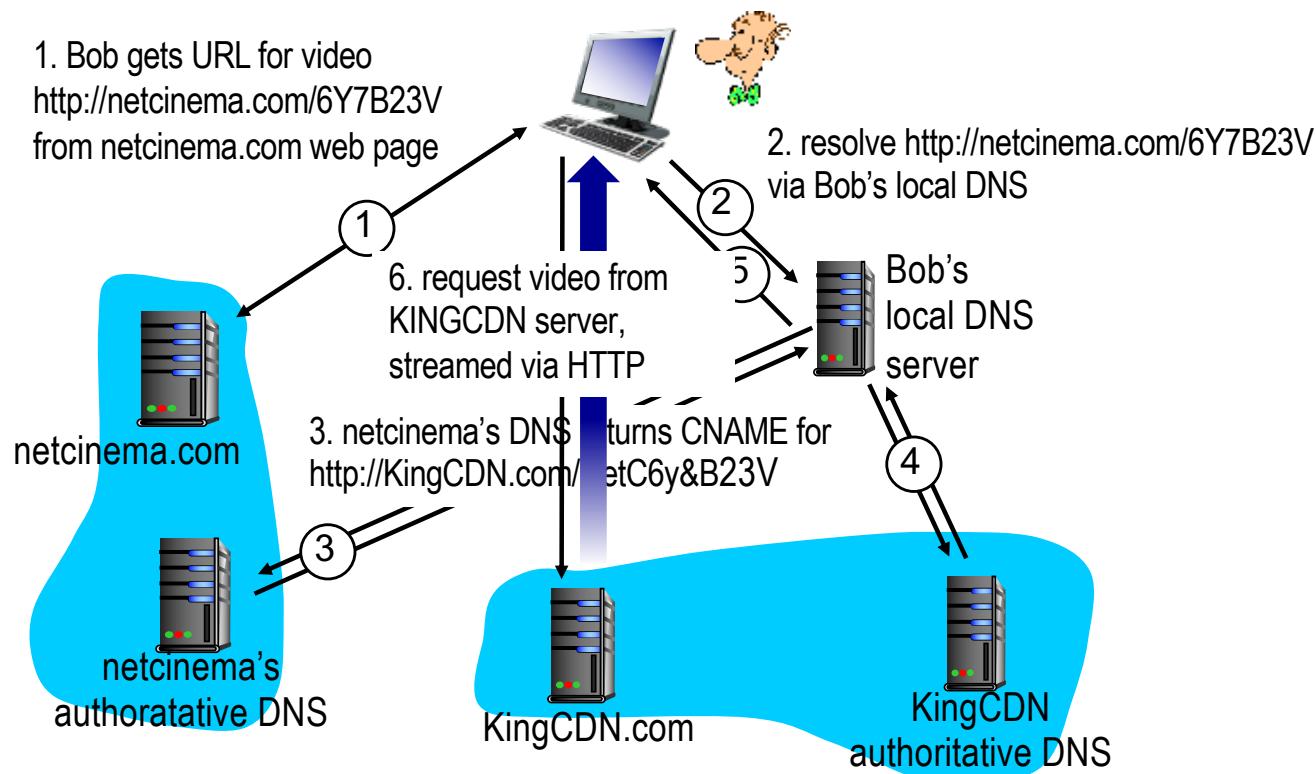
*OTT challenges:* coping with a congested Internet

- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

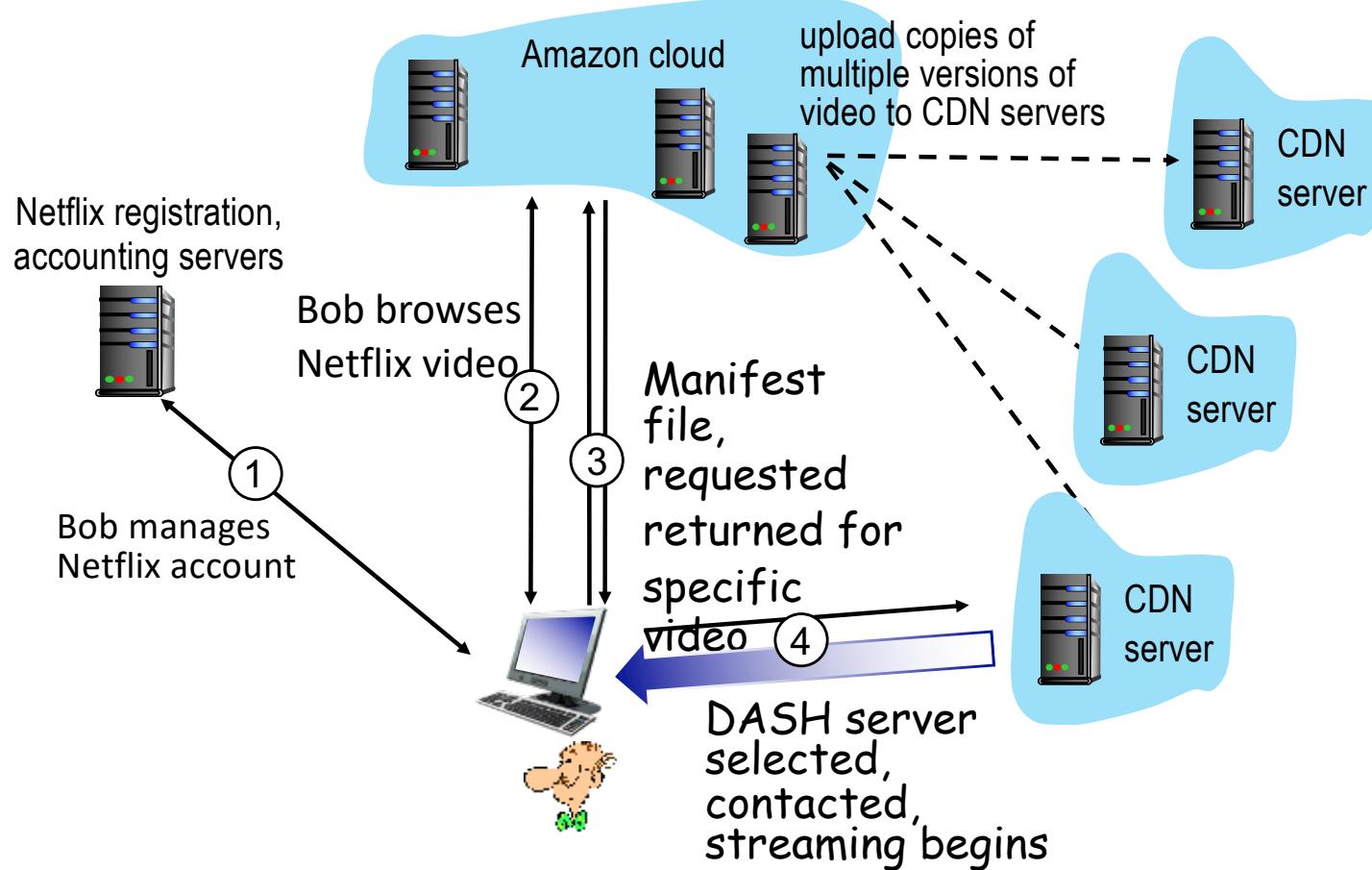
# CDN content access: a closer look

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

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



# Case study: Netflix





## Quiz: CDN

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

## 2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

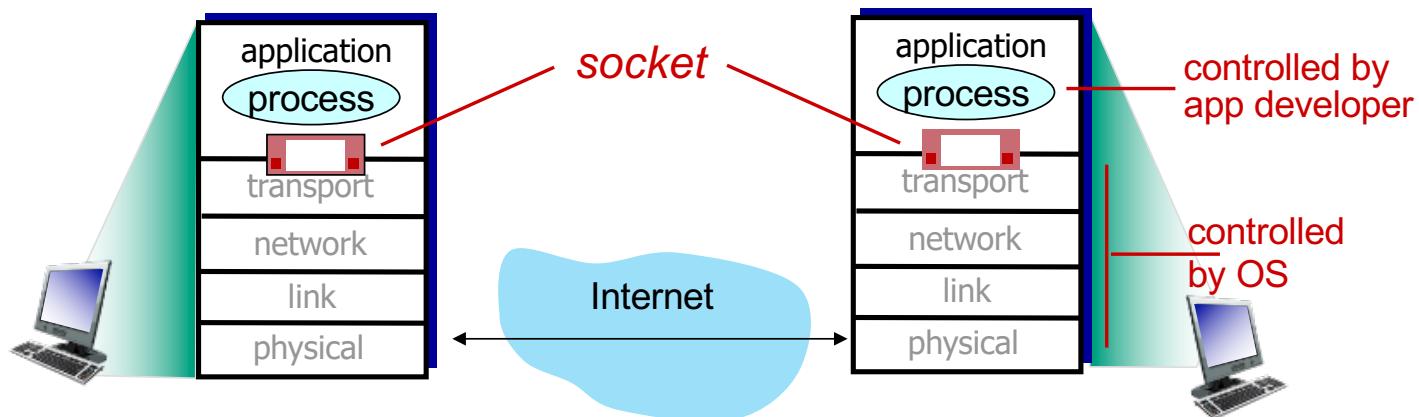
2.7 socket programming with UDP and TCP

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

# Socket programming

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

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



# Socket programming with UDP

**UDP: no “connection” between client & server**

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

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

**Application viewpoint:**

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

# Pseudo code UDP client

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

# Pseudo code UDP server

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

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

# Socket programming with TCP

## Client must contact server

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

## Client contacts server by:

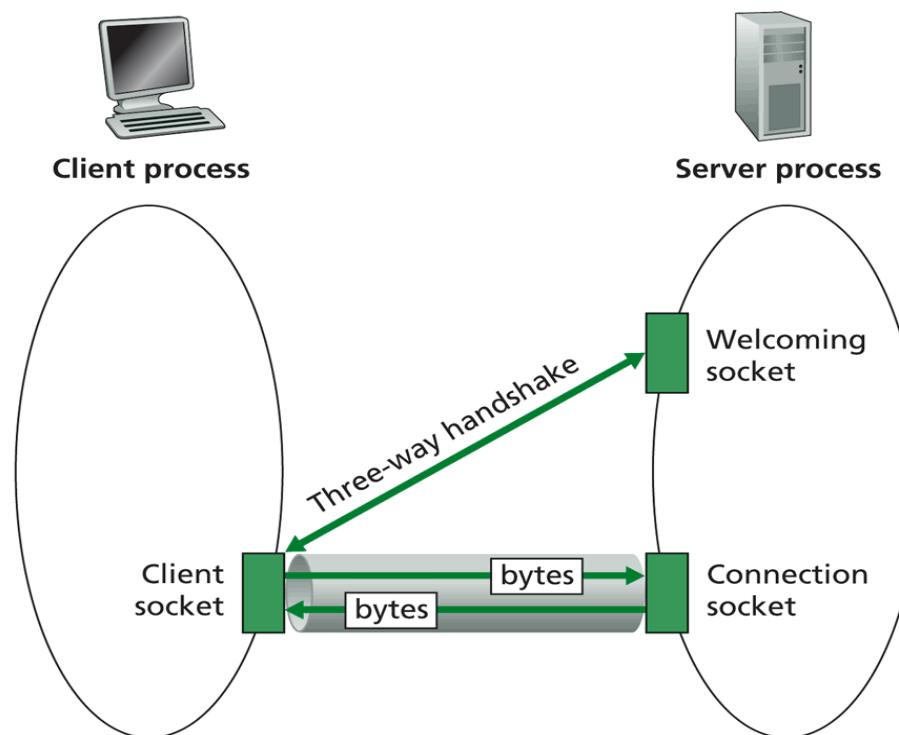
- ❖ Creating TCP socket, specifying IP address, port number of server process
- ❖ *when client creates socket*: client TCP establishes connection to server TCP

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

## Application viewpoint

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

# TCP Sockets



# Pseudo code TCP client

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

# Pseudo code TCP server

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

# Summary: Completed Application Layer

*our study of network apps now complete!*

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

# Computer Networks and Applications

COMP 3331/COMP 9331

Week 4

Major Concepts:

1. Multiplexing-demultiplexing
2. Checksum
3. Reliable data transfer

## Transport Layer Part 1

Reading Guide:  
Chapter 3, Sections 3.1 – 3.5.2

# Transport layer: overview

*Our goal:*

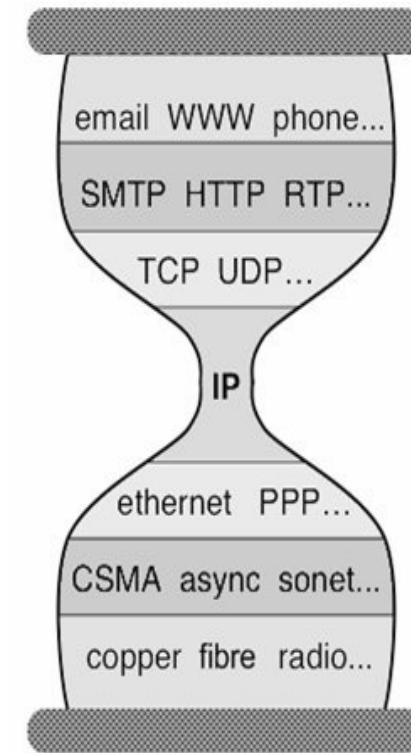
- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport

# Transport layer: roadmap

- ❖ Transport-layer services
- ❖ Multiplexing and demultiplexing
- ❖ Connectionless transport: UDP
- ❖ Principles of reliable data transfer
- ❖ Connection-oriented transport: TCP
- ❖ Principles of congestion control
- ❖ TCP congestion control
- ❖ Evolution of transport-layer functionality

# Transport layer

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

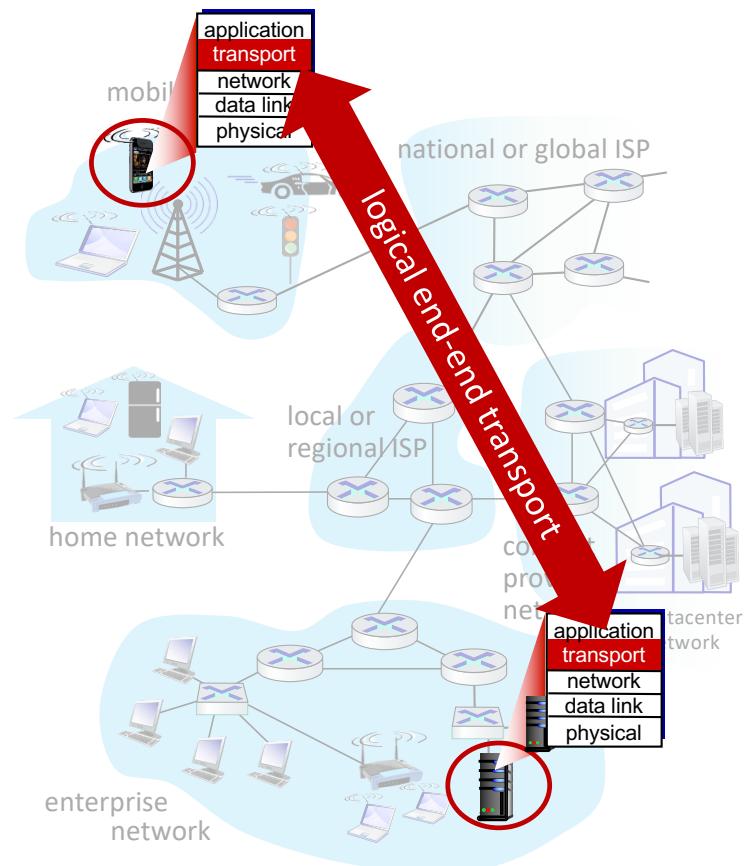


# Network layer (some context)

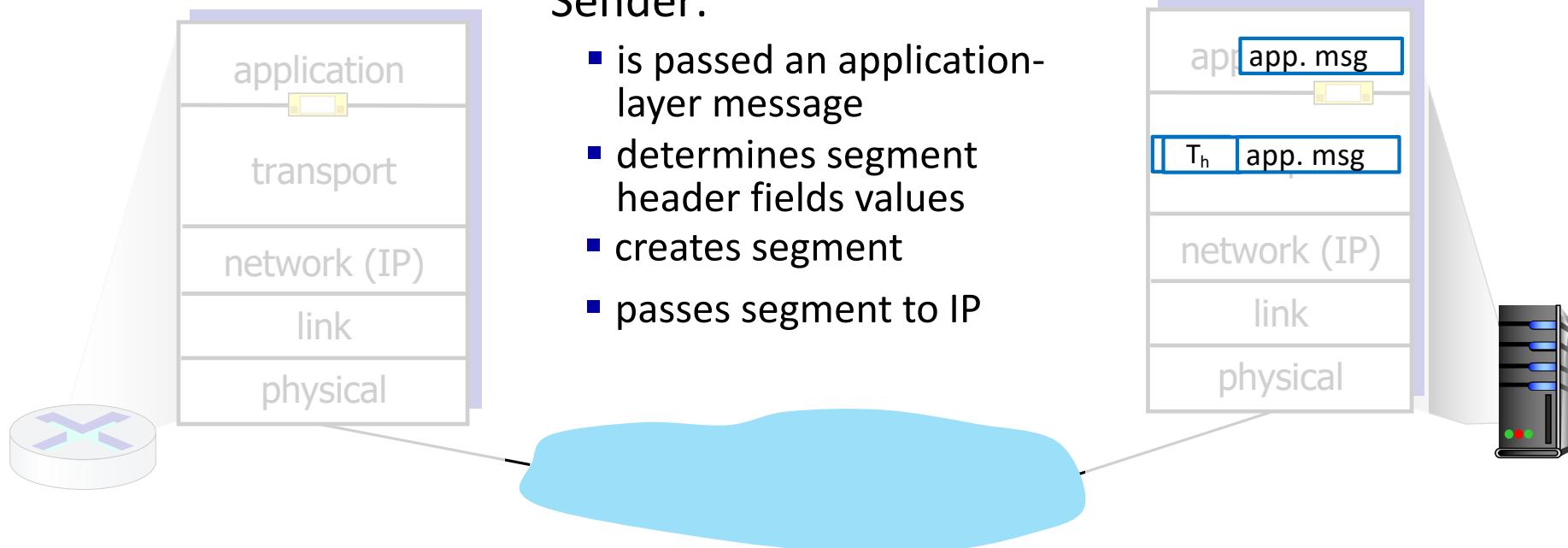
- ❖ What it does: finds paths through network
  - Routing from one end host to another
- ❖ What it doesn't:
  - Reliable transfer: “best effort delivery”
  - Guarantee paths
  - Arbitrate transmission rates
- ❖ For now, think of the network layer as giving us an “API” with one function: `sendtohost(data,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

# Transport services and protocols

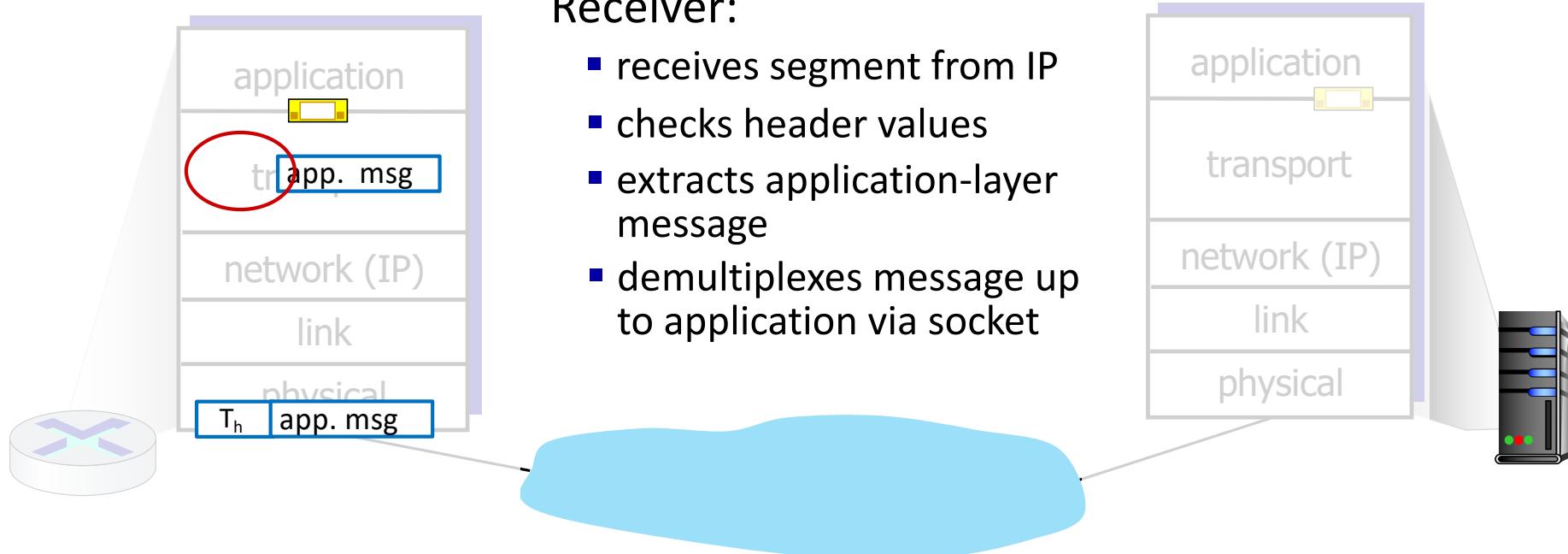
- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
  - sender: breaks application messages into *segments*, passes to network layer
  - receiver: reassembles segments into messages, passes to application layer
- two transport protocols available to Internet applications
  - TCP, UDP



# Transport Layer Actions

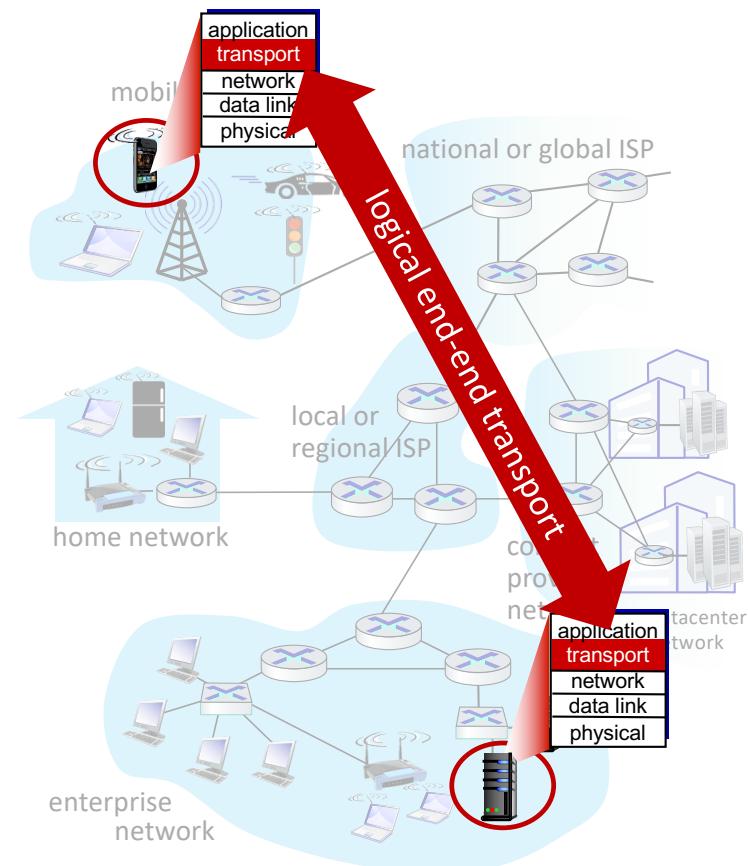


# Transport Layer Actions



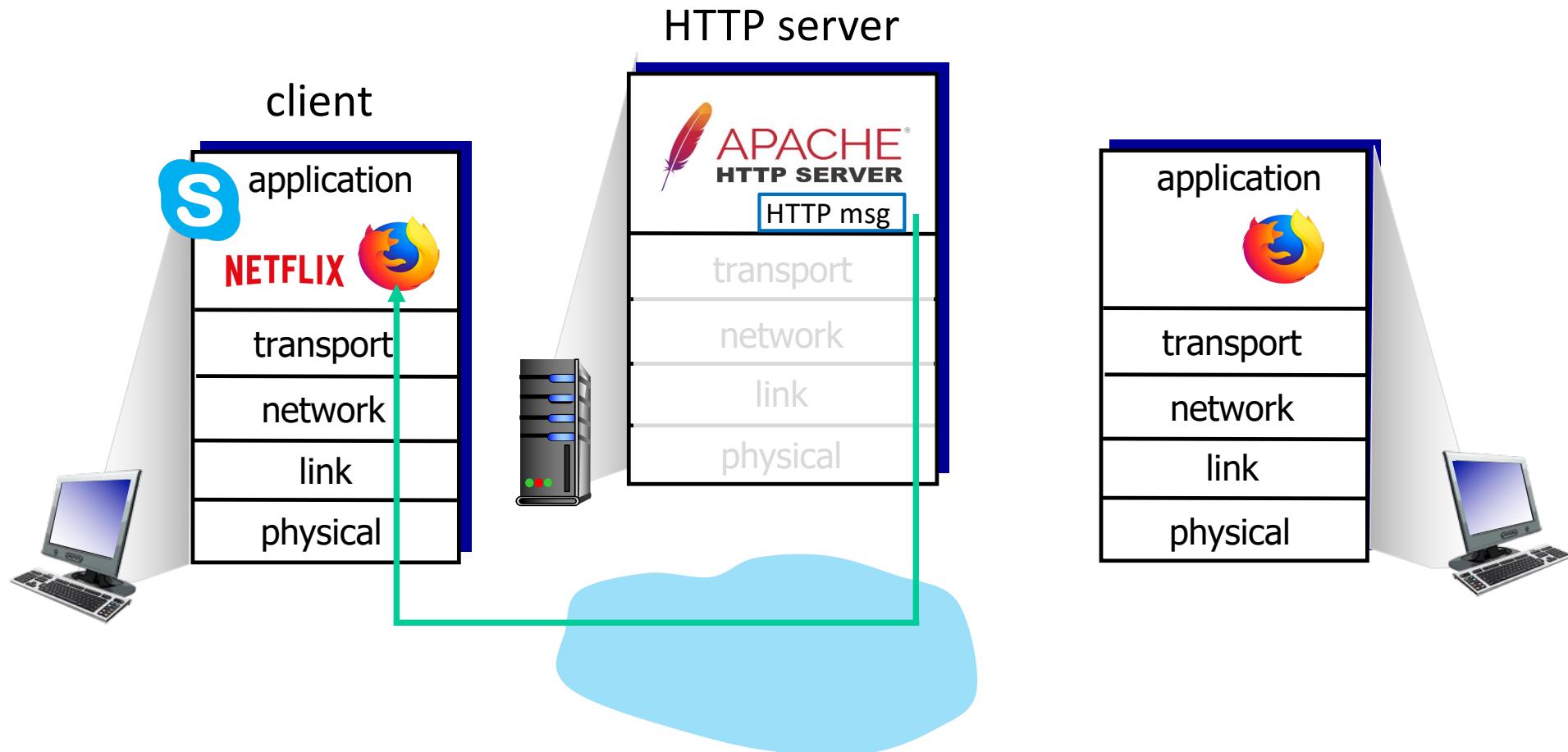
# Two principal Internet transport protocols

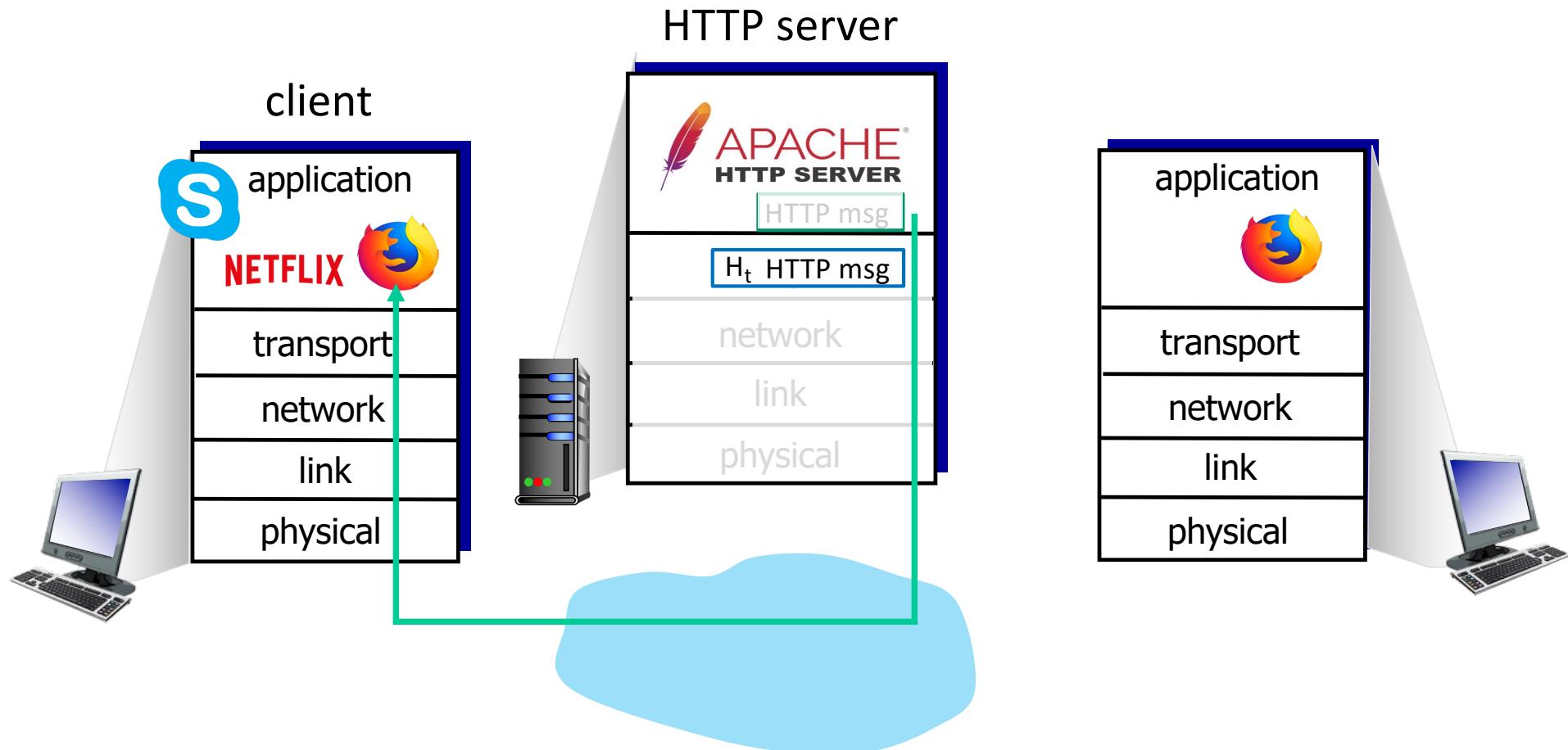
- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

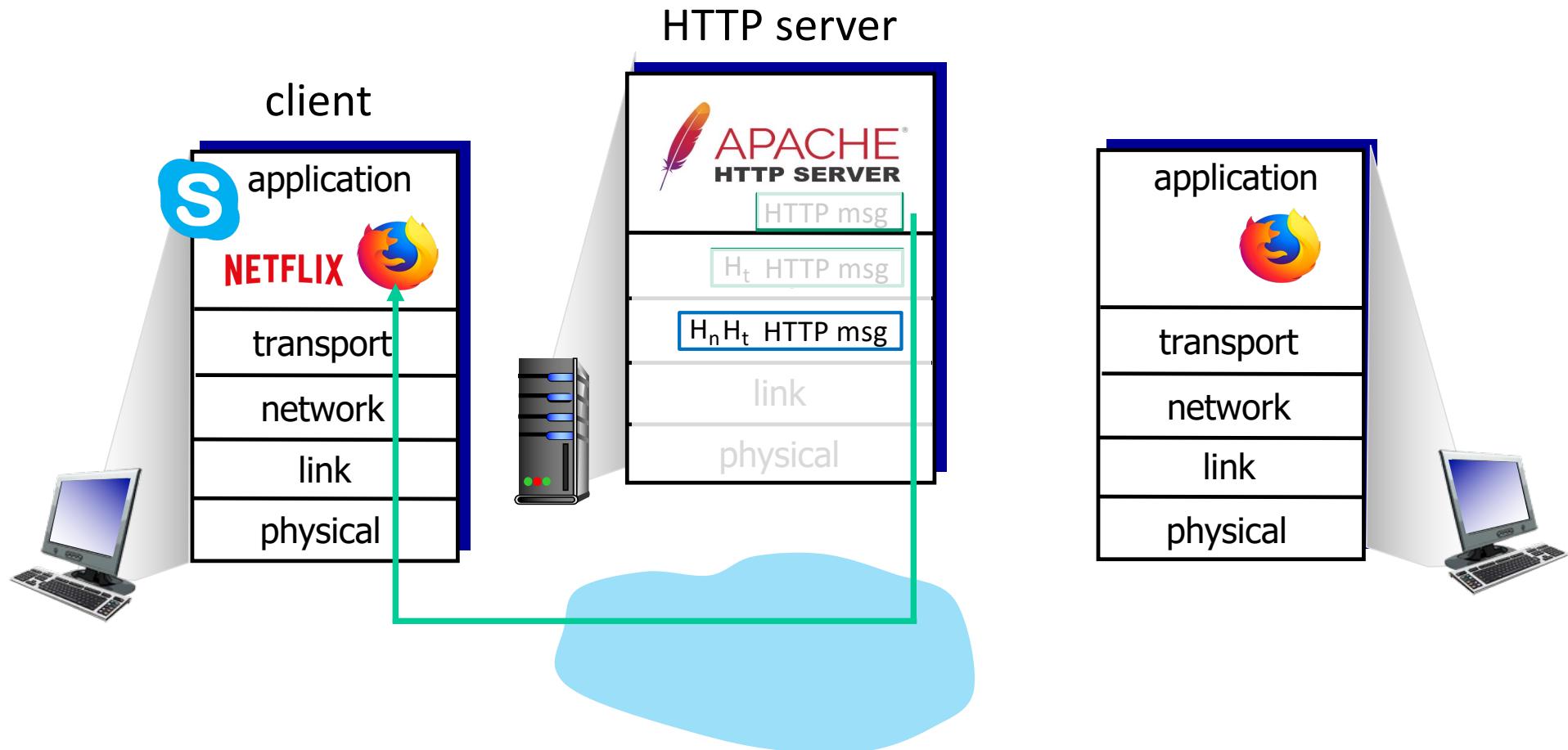


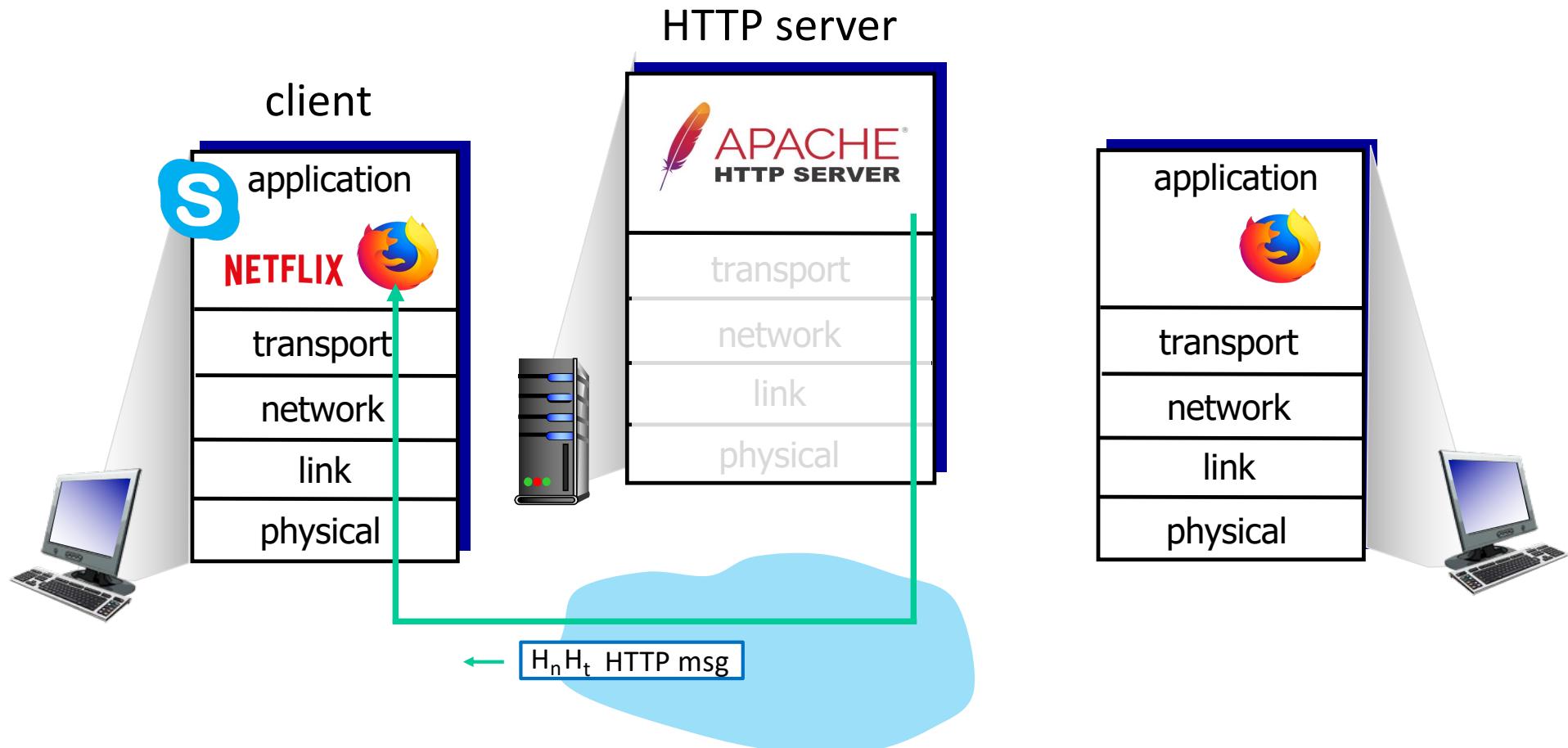
# Transport layer: roadmap

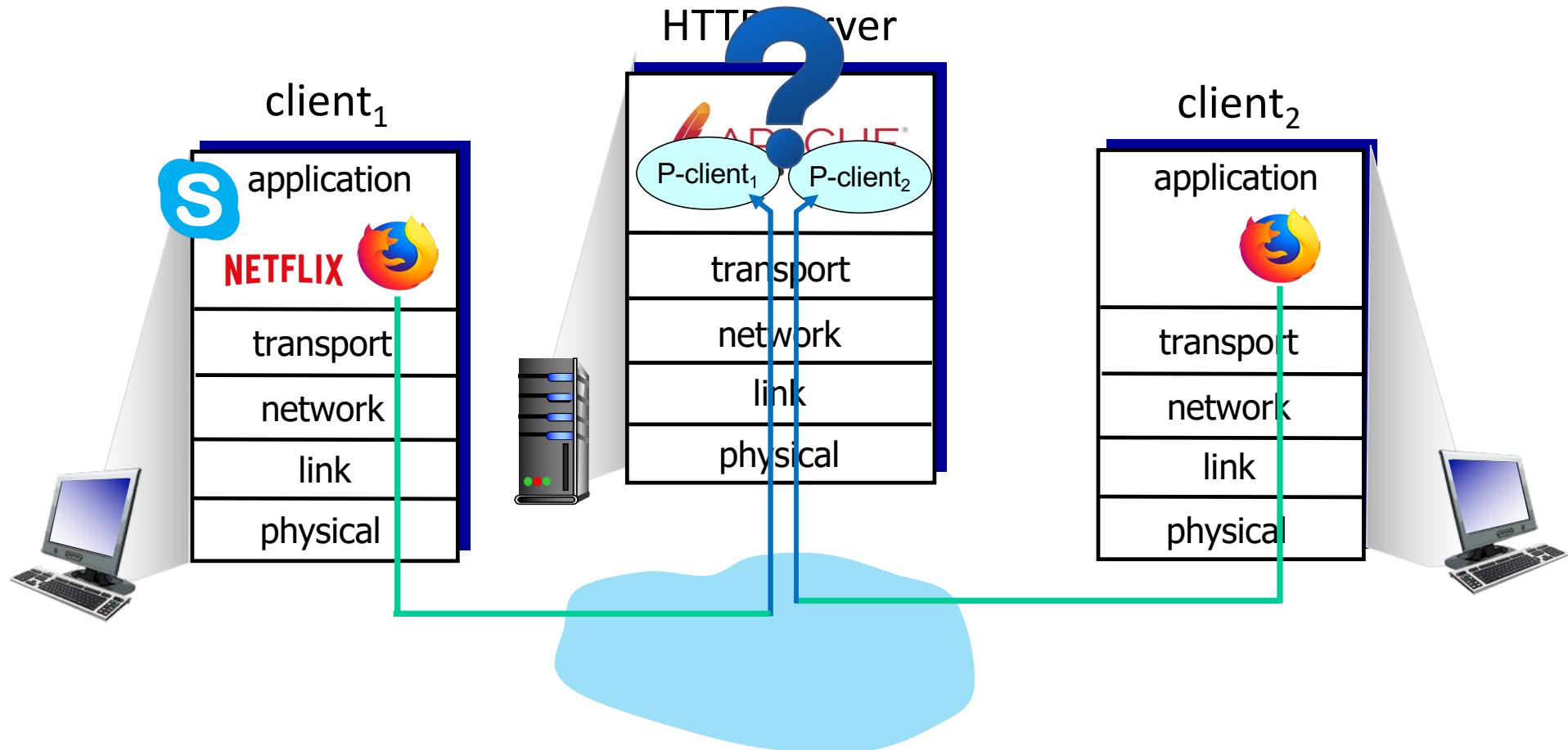
- ❖ Transport-layer services
- ❖ Multiplexing and demultiplexing
- ❖ Connectionless transport: UDP
- ❖ Principles of reliable data transfer
- ❖ Connection-oriented transport: TCP
- ❖ Principles of congestion control
- ❖ TCP congestion control
- ❖ Evolution of transport-layer functionality











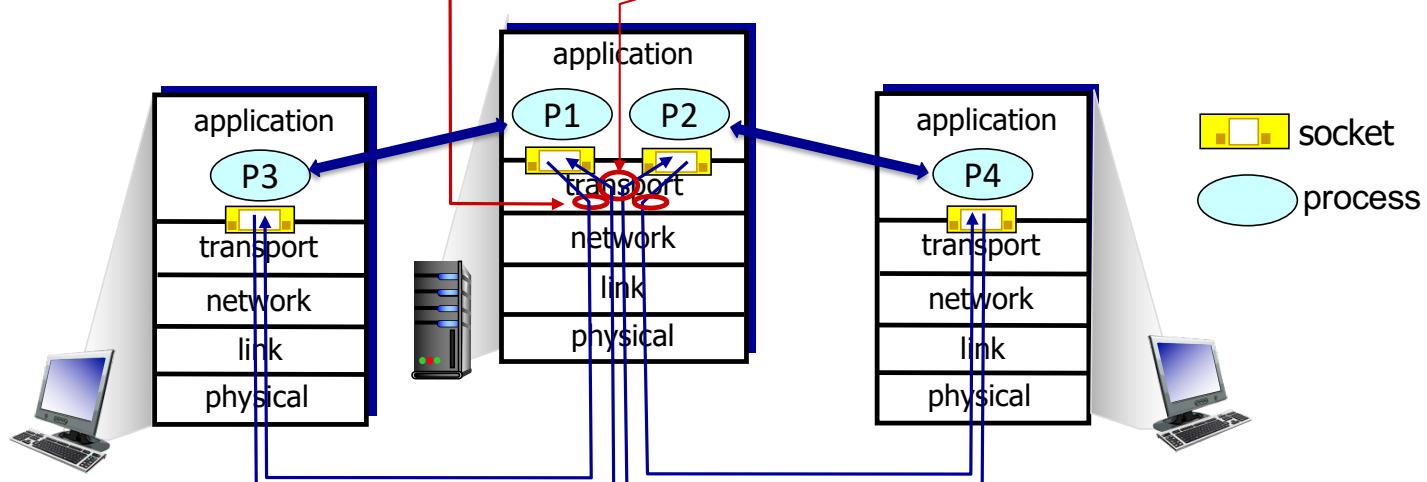
# Multiplexing/demultiplexing

*multiplexing at sender:*

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

*demultiplexing at receiver:*

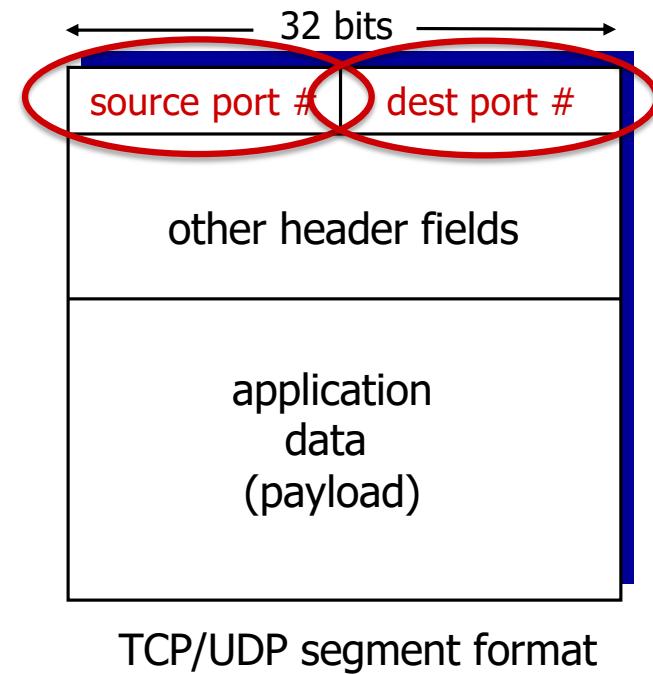
use header info to deliver received segments to correct socket



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

# How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address (not shown in the pic)
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

# Connectionless demultiplexing (UDP)

Recall:

- when creating socket, either specify *host-local* port # (or let OS pick random available port):

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

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

when receiving host receives UDP segment:

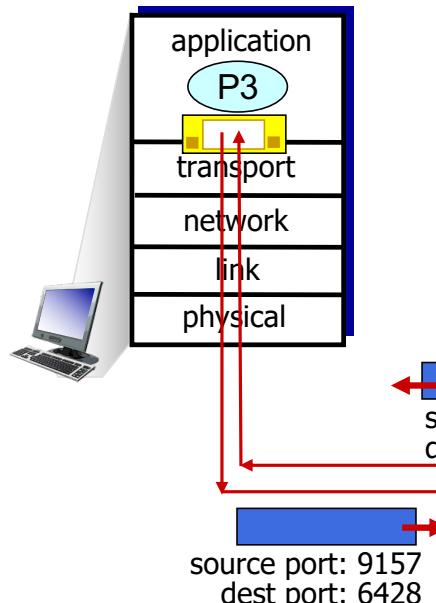
- checks destination port # in segment
- directs UDP segment to socket with that port #



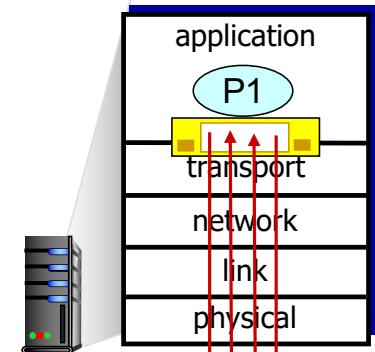
IP/UDP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

# (UDP) Connectionless demultiplexing: an example

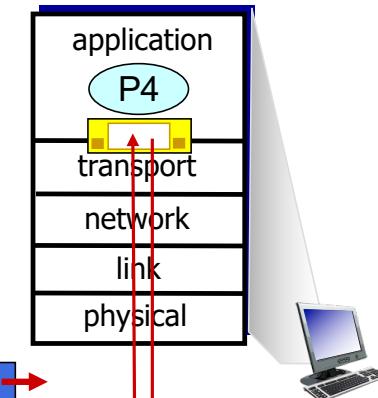
```
DatagramSocket mySocket2 =  
    new DatagramSocket  
(9157);
```



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



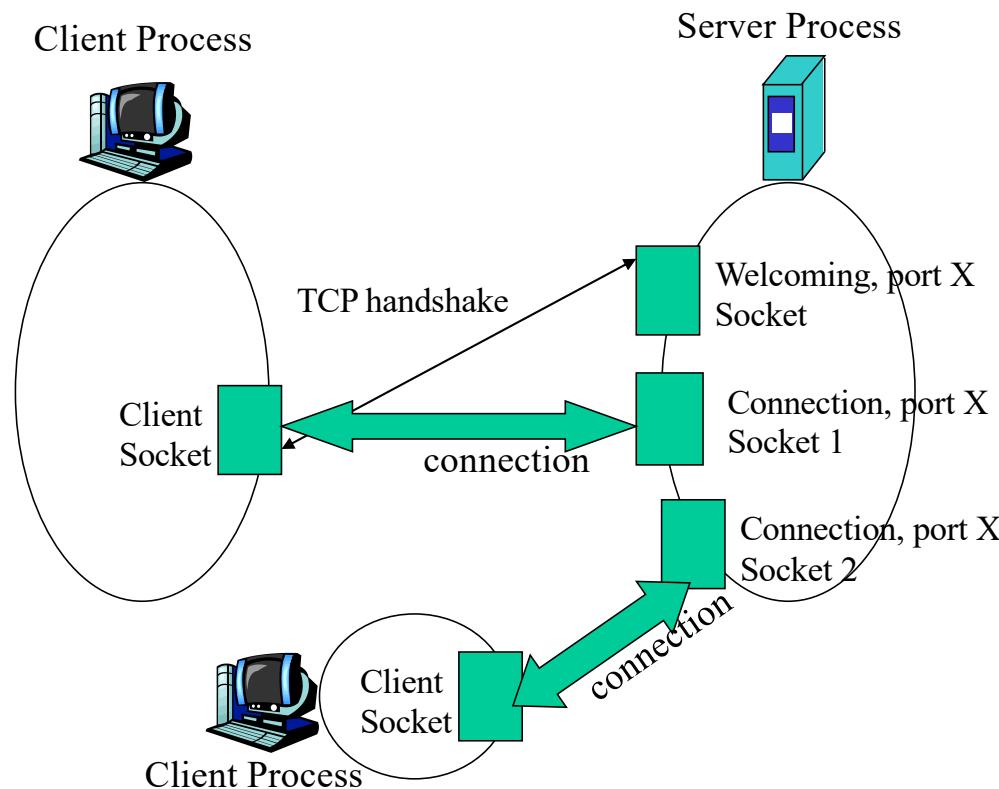
```
DatagramSocket mySocket1 =  
    new DatagramSocket (5775);
```



# Connection-oriented demultiplexing (TCP)

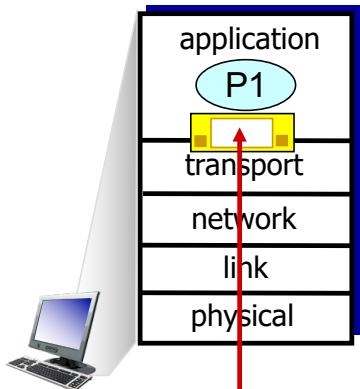
- TCP socket identified by **4-tuple**:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values* (4-tuple) to direct segment to appropriate socket
- server may support many simultaneous TCP sockets:
  - each socket identified by its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

# Revisiting TCP Sockets

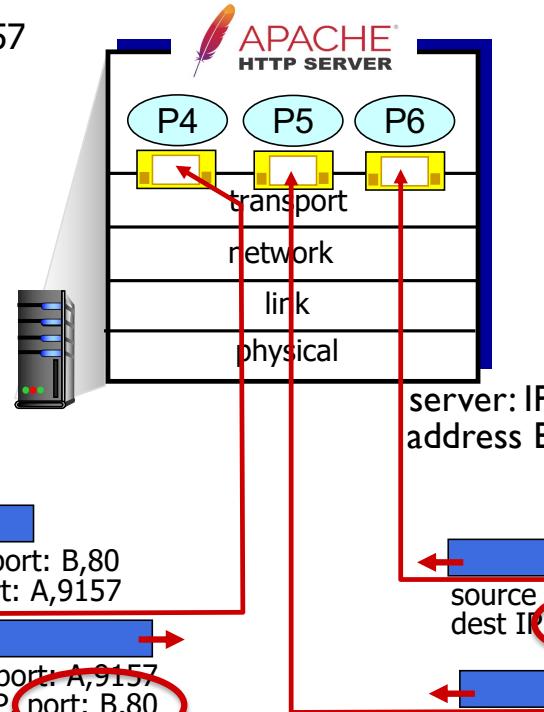


# Connection-oriented demultiplexing: example

Browser process p1 uses port 9157

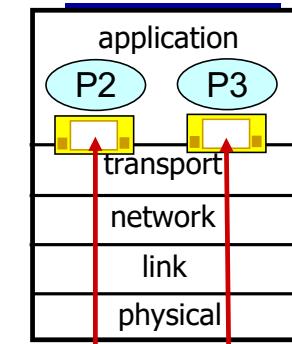


host: IP address A

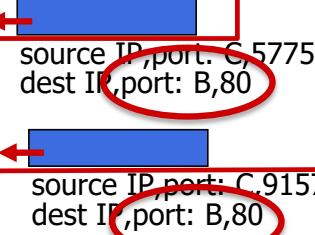


server: IP address B

Browser process p2 uses port 5775, and p3 uses port 9157



host: IP address C



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

# Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination IP and port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at *all* layers (more later in the course)

# May I scan your ports?

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

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

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

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

## Quiz: UDP Sockets



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

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

## Quiz: TCP Sockets



Suppose 100 clients are simultaneously communicating with a traditional HTTP/TCP web server. How many sockets are active respectively at the server and each client?

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



## Quiz: TCP Sockets

Suppose 100 clients are simultaneously communicating with a traditional HTTP/TCP web server. Do all the TCP sockets at the server have the same server-side port number?

- a) Yes
- b) No

# Transport layer: roadmap

- ❖ Transport-layer services
- ❖ Multiplexing and demultiplexing
- ❖ **Connectionless transport: UDP**
- ❖ Principles of reliable data transfer
- ❖ Connection-oriented transport: TCP
- ❖ Principles of congestion control
- ❖ TCP congestion control
- ❖ Evolution of transport-layer functionality

# UDP: User Datagram Protocol

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

## 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

- Applications that use UDP:
  - 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: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768                    J. Postel  
                                  ISI  
                                  28 August 1980

**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.

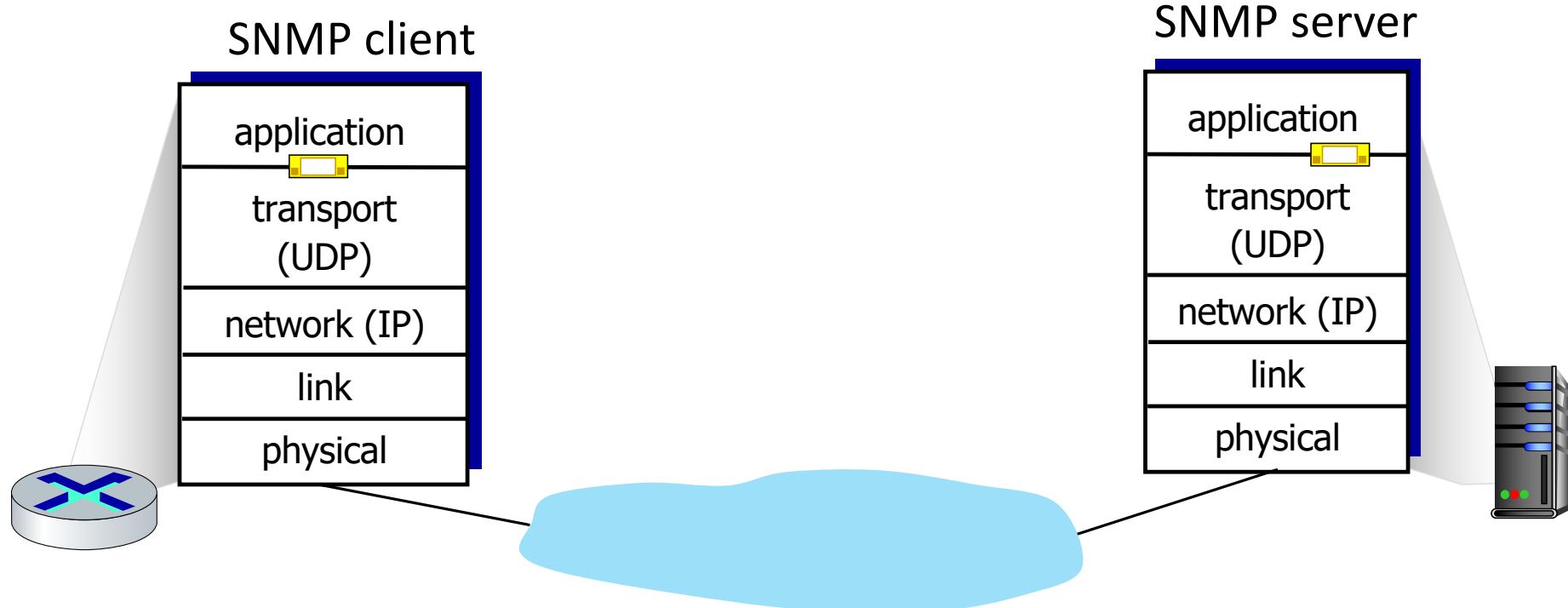
This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should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

**Format**

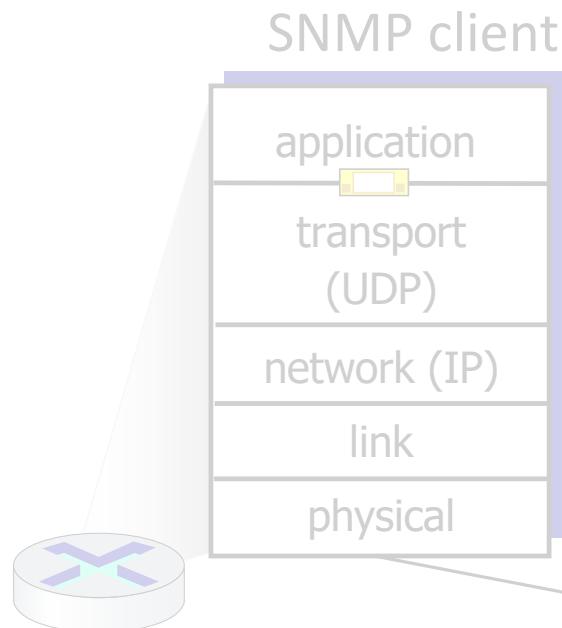
-----

	0	7 8	15 16	23 24	31	
	Source Port		Destination Port			
	Length		Checksum			
	data octets ...					

# UDP: Transport Layer Actions



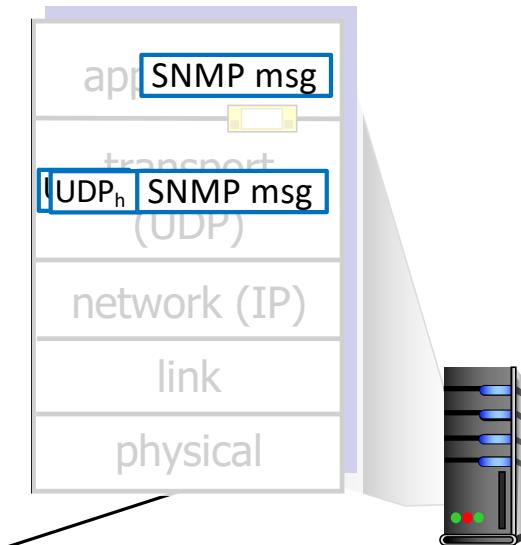
# UDP: Transport Layer Actions



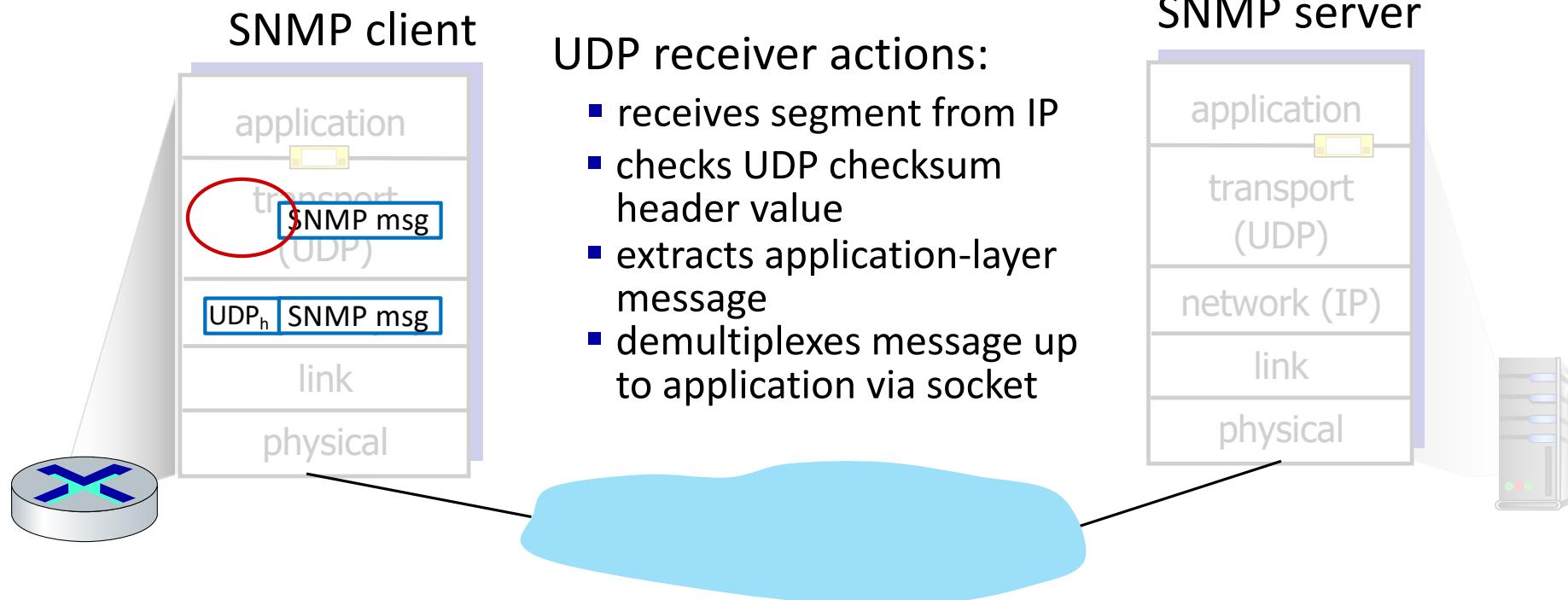
## UDP sender actions:

- is passed an application-layer message
- determines UDP segment header fields values
- creates UDP segment
- passes segment to IP

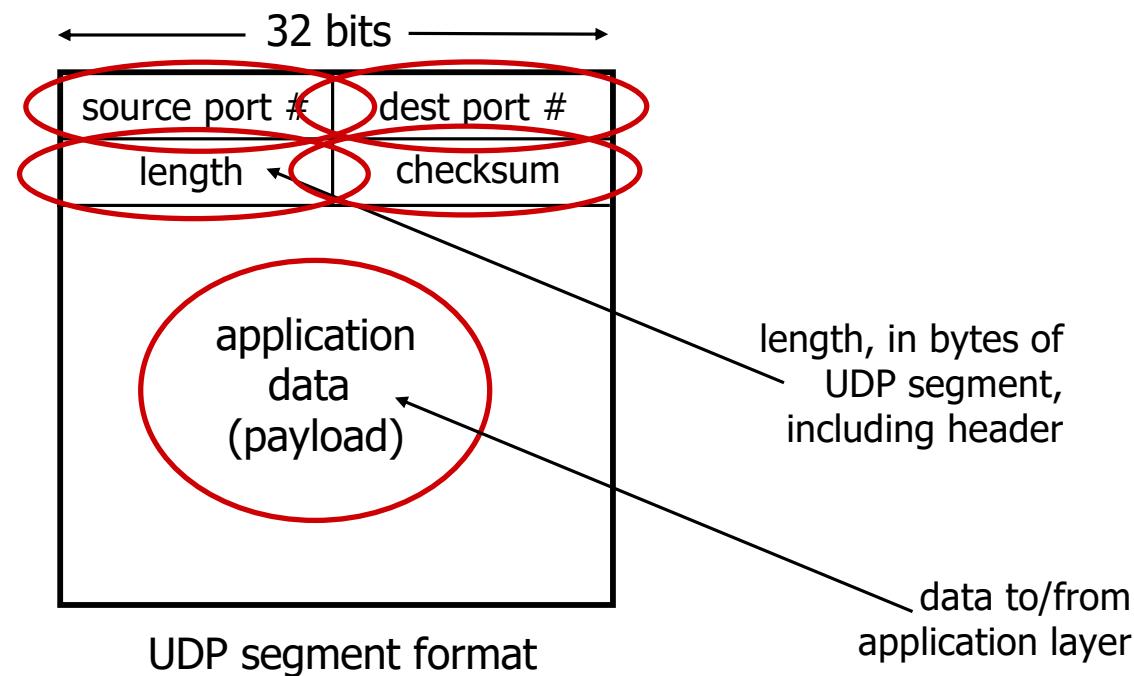
## SNMP server



# UDP: Transport Layer Actions

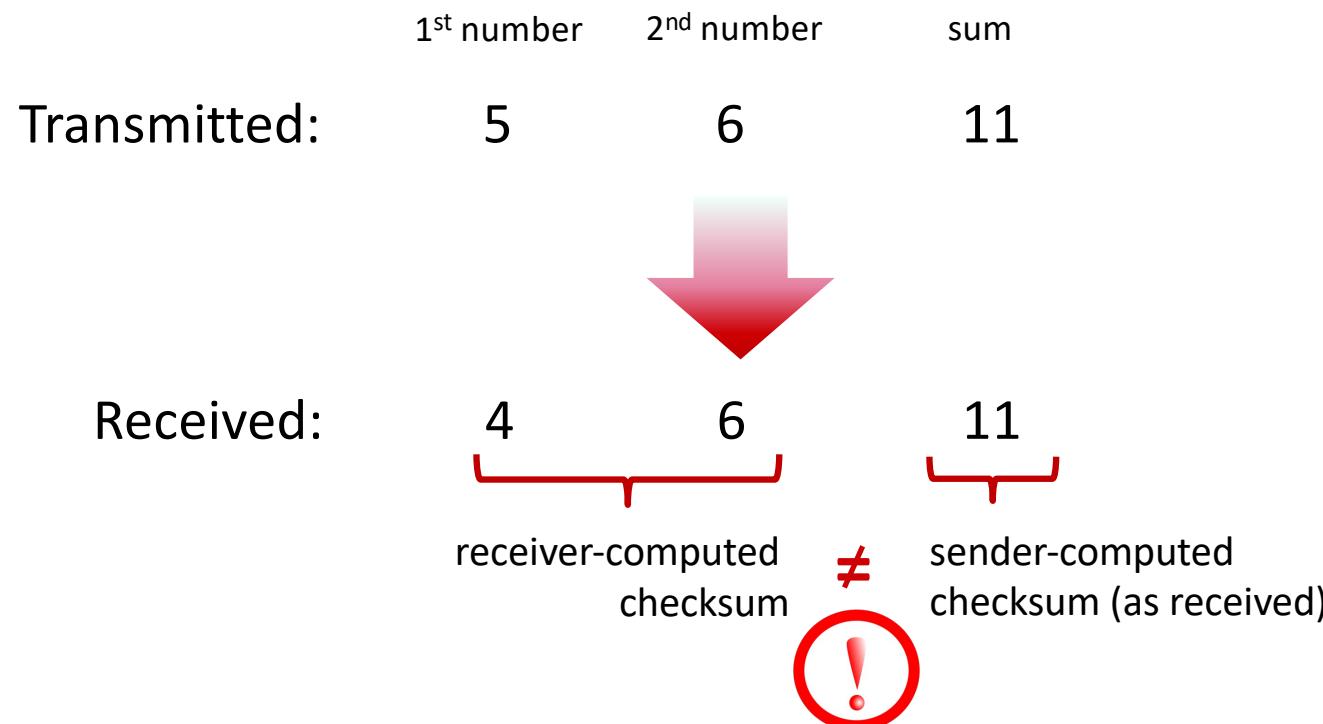


# UDP segment header



# UDP checksum

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



# Internet checksum

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

## sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) 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

# Internet checksum: weak protection!

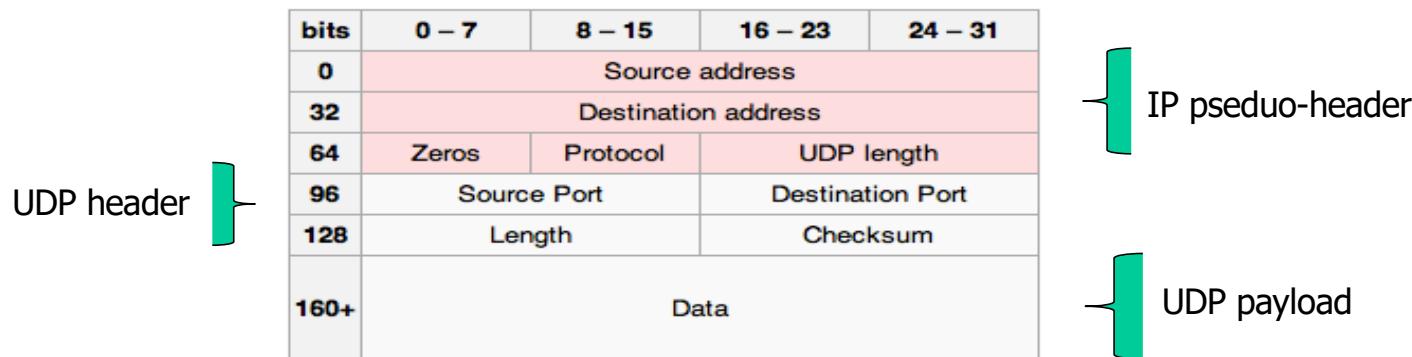
example: add two 16-bit integers

	1 1 1 0 0 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 0 1 0 1	1 0
wraparound	<u>1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1</u>	
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!

# UDP Checksum in Practice

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



# Checksum: example

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

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

**Note: TCP Checksum computation is exactly similar**

# UDP Applications

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

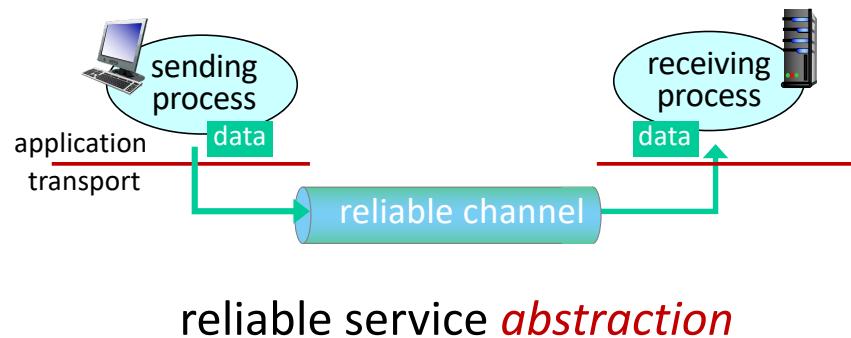
# Summary: UDP

- “no frills” protocol:
  - segments may be lost, delivered out of order
  - best effort service: “send and hope for the best”
- UDP has its plusses:
  - no setup/handshaking needed (no RTT incurred)
  - can function when network service is compromised
  - helps with reliability (checksum)
- build additional functionality on top of UDP in application layer (e.g., HTTP/3)

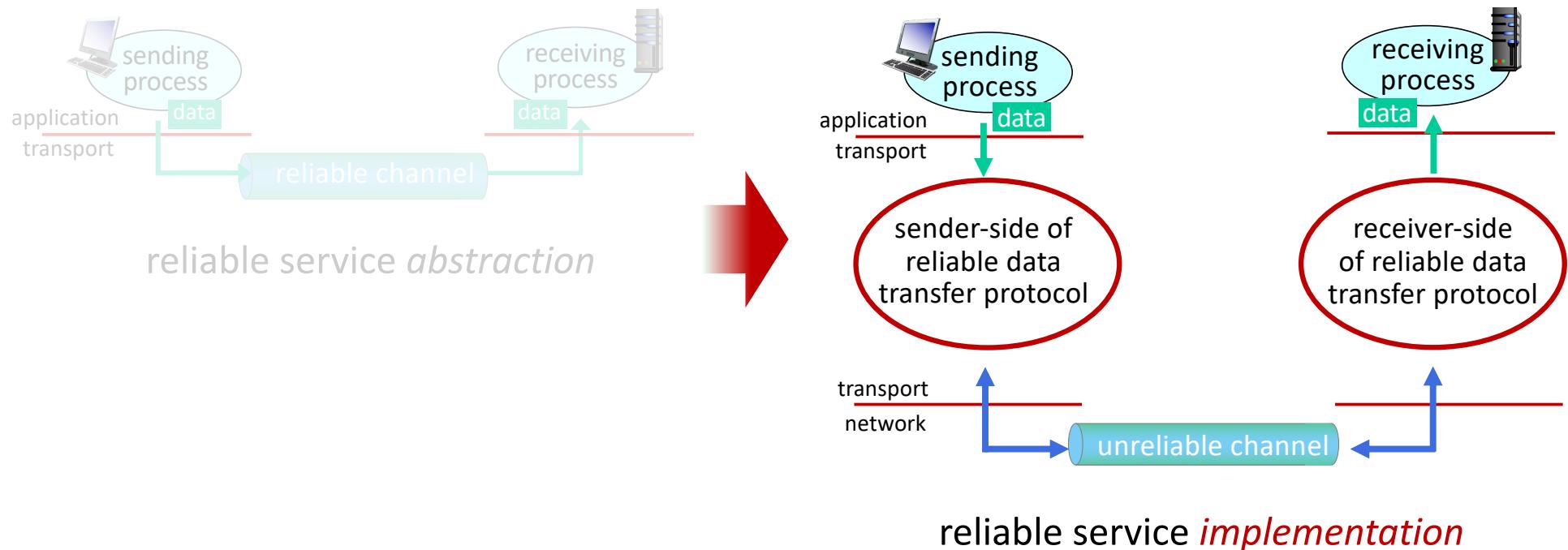
# Transport layer: roadmap

- ❖ Transport-layer services
- ❖ Multiplexing and demultiplexing
- ❖ Connectionless transport: UDP
- ❖ **Principles of reliable data transfer**
- ❖ Connection-oriented transport: TCP
- ❖ Principles of congestion control
- ❖ TCP congestion control
- ❖ Evolution of transport-layer functionality

# Principles of reliable data transfer

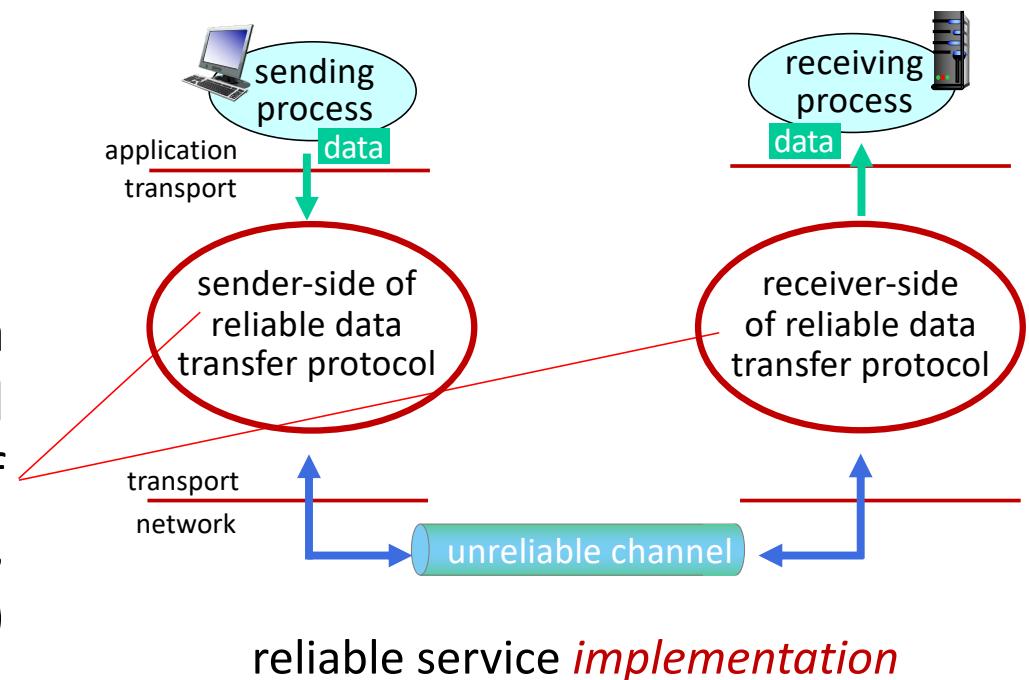


# Principles of reliable data transfer



# Principles of reliable data transfer

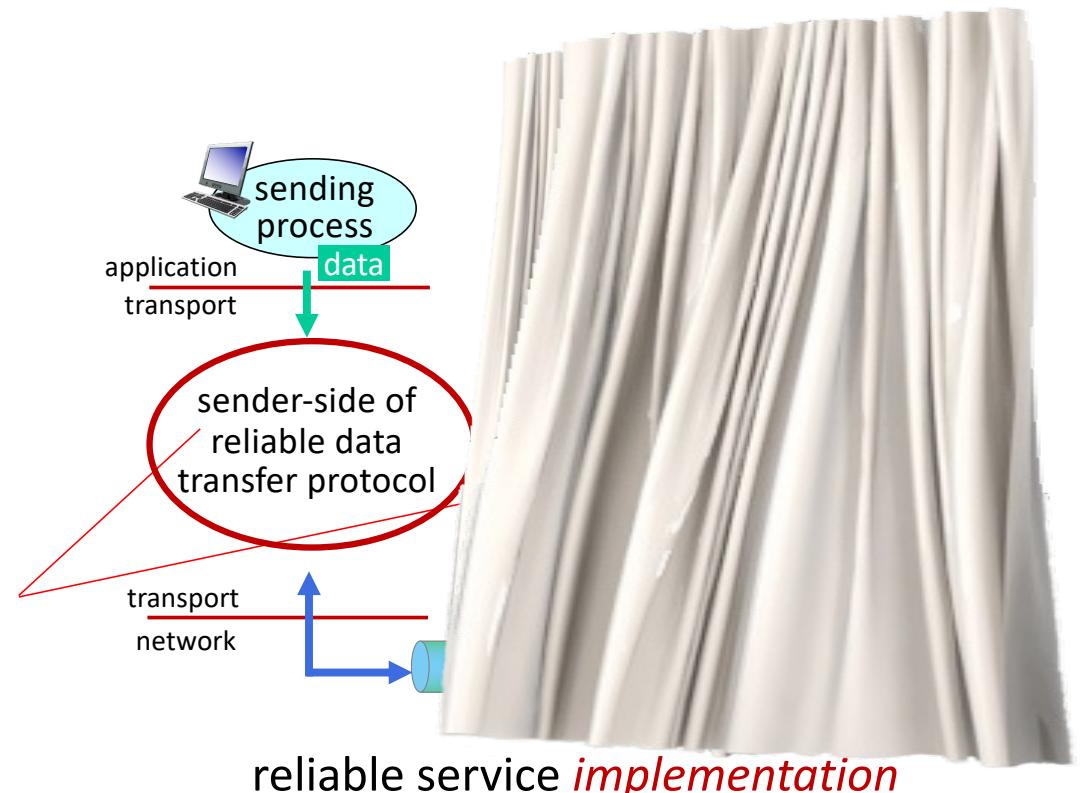
Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)



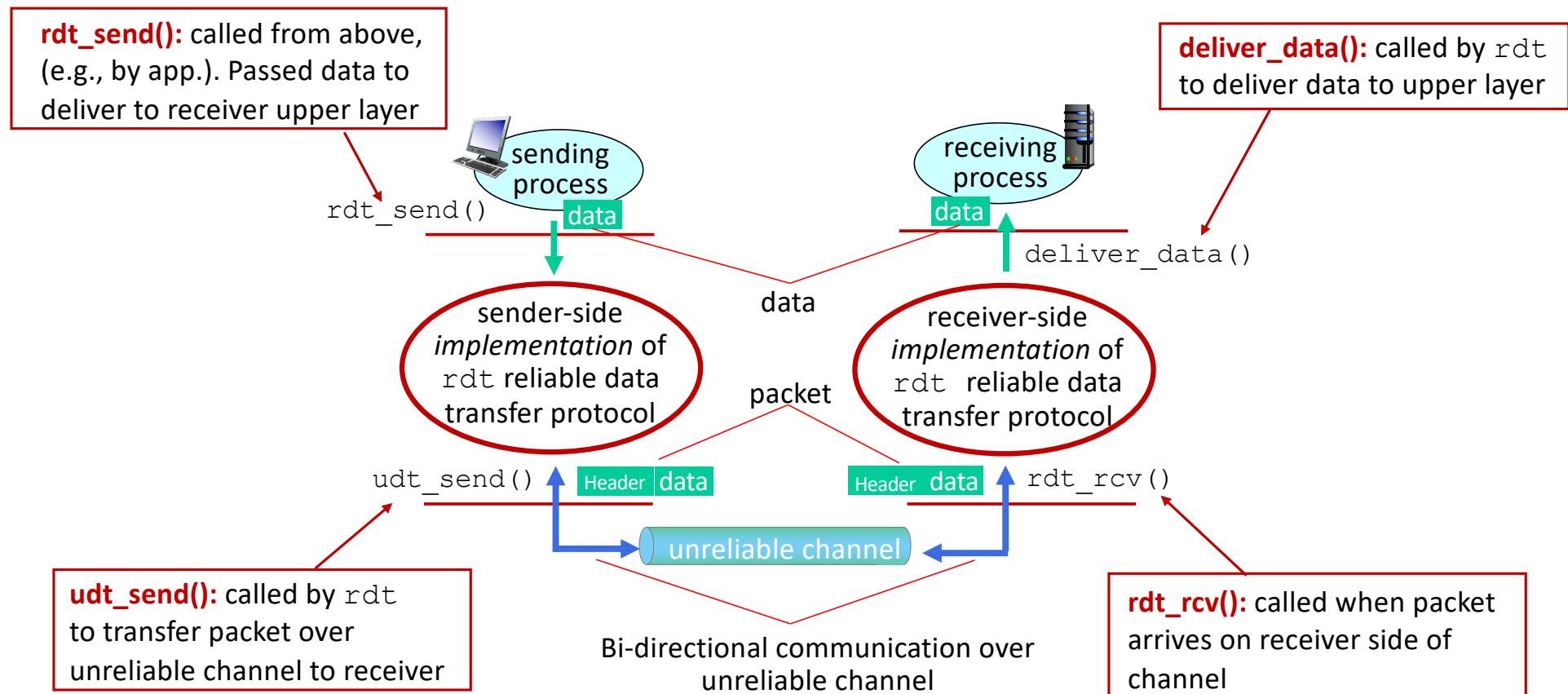
# Principles of reliable data transfer

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

- unless communicated via a message



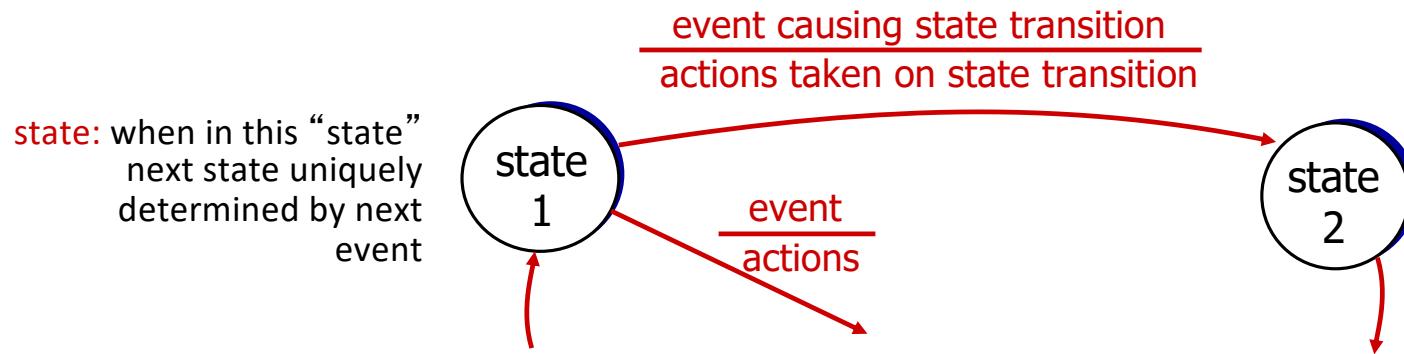
# Reliable data transfer protocol (rdt): interfaces



# Reliable data transfer: getting started

We will:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow in both directions!
- Book uses finite state machines (FSM) to specify sender, receiver
  - We won't use them in the lecture, and you won't be asked exam questions on them

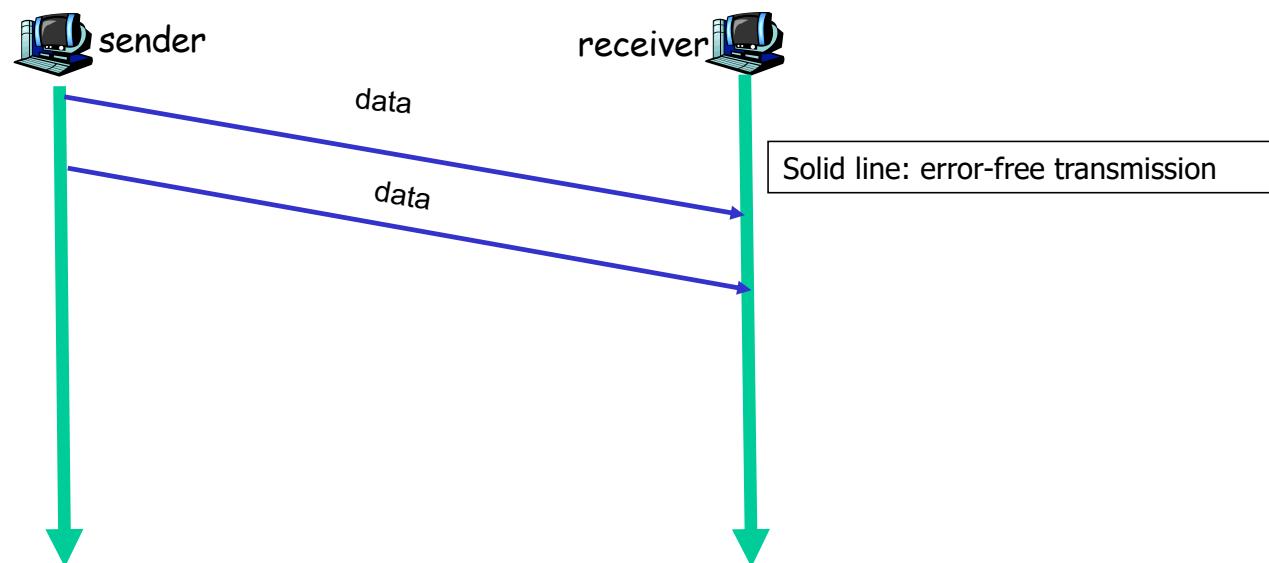


# rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
  - Nothing to do



# Global Picture of rdt1.0



## rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum (e.g., Internet checksum) to detect bit errors
- *the question:* how to recover from errors?

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

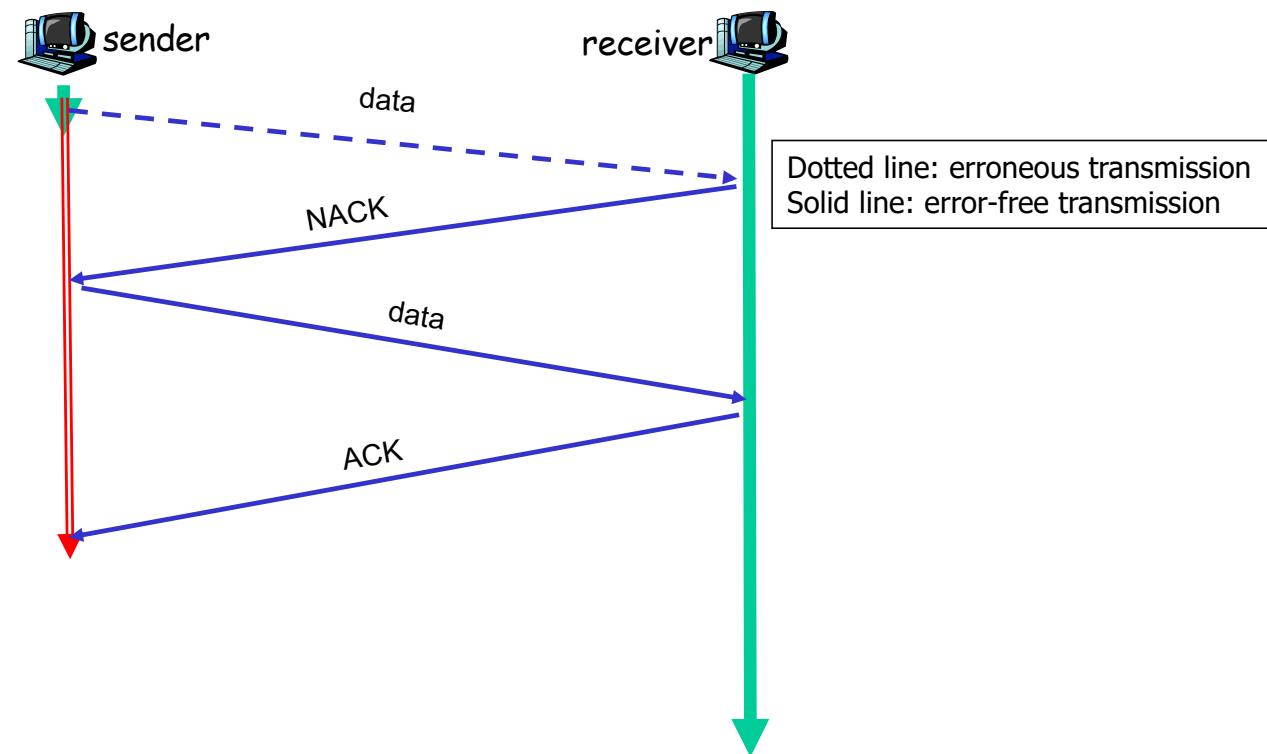
## rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the question:* how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender *retransmits* pkt on receipt of NAK
- new mechanisms in rdt2.0 (beyond rdt1.0):
  - error detection
  - feedback: control msgs (ACK,NAK) from receiver to sender
  - retransmission

**stop and wait**

sender sends one packet, then waits for receiver response

# Global Picture of rdt2.0



# rdt2.0 has a fatal flaw!

what happens if ACK/NAK corrupted?

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

handling duplicates:

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

stop and wait

sender sends one packet, then waits for receiver response

# rdt2.1: discussion

## sender:

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

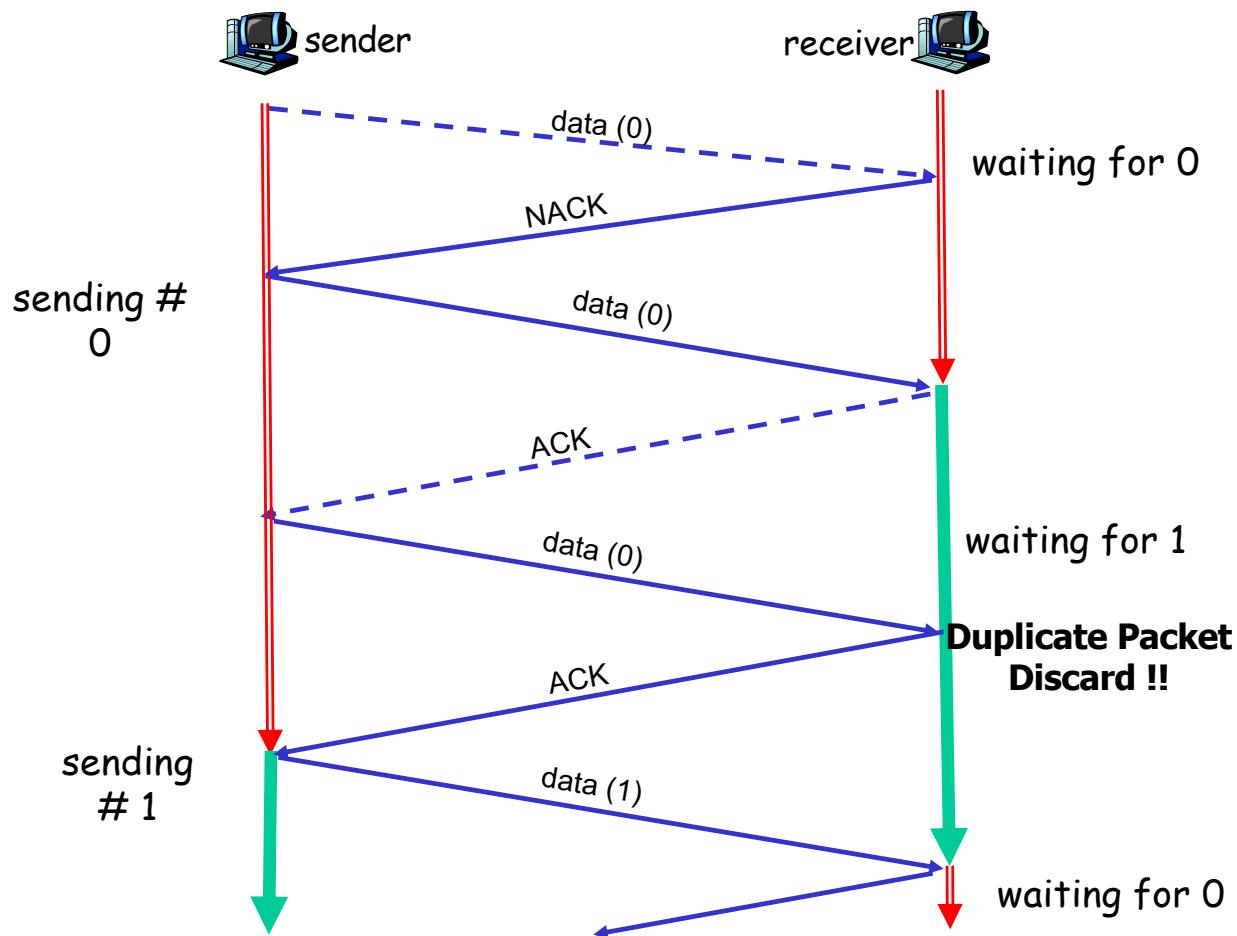
## receiver:

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

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

# Another Look at rdt2.1

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



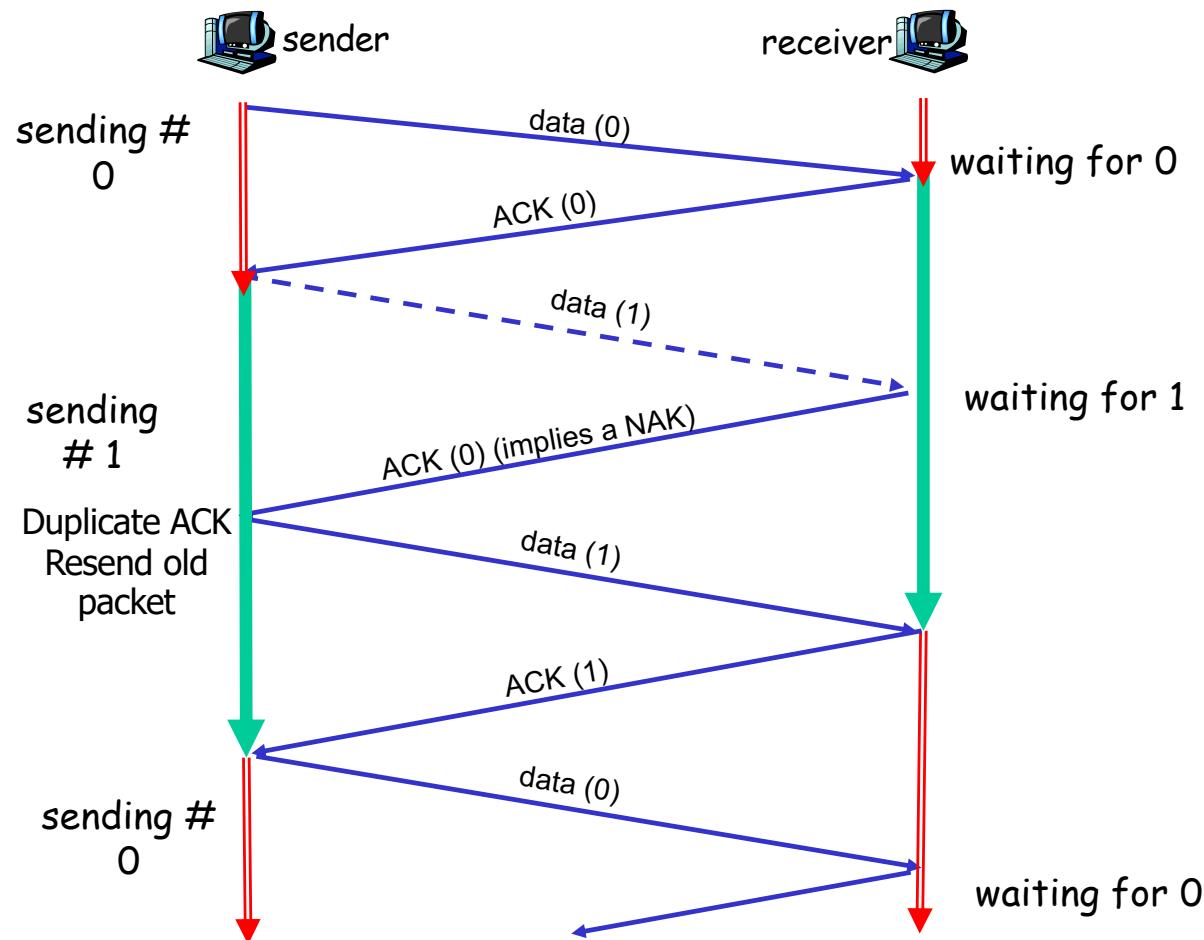
## rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK:  
*retransmit current pkt*

As we will see, TCP uses this approach to be NAK-free

## rdt2.2: Example

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



## rdt3.0: channels with errors and loss

*New channel assumption:* underlying channel can also *lose* packets (data, ACKs)

- checksum, sequence #s, ACKs, retransmissions will be of help ... but not quite enough

*Q:* How do *humans* handle lost sender-to-receiver words in conversation?

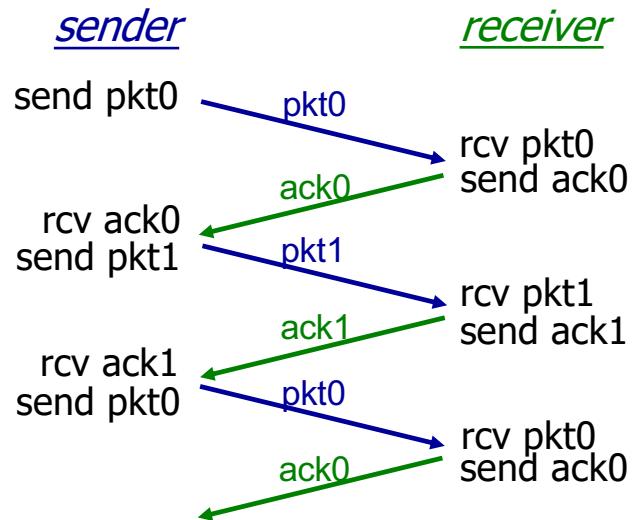
## rdt3.0: channels with errors and loss

*Approach:* sender waits “reasonable” amount of time for ACK

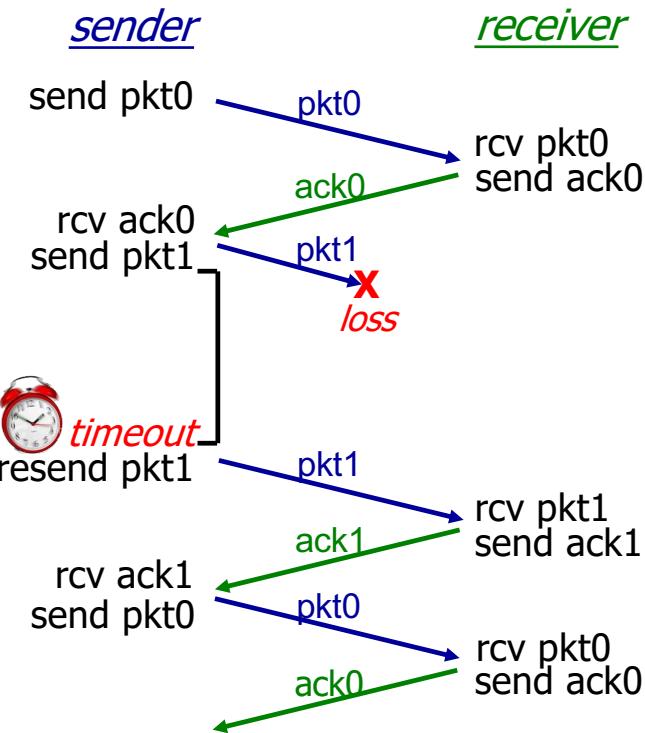
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq #s already handles this!
  - receiver must specify seq # of packet being ACKed
- use countdown timer to interrupt after “reasonable” amount of time
- No retransmission on duplicate ACKs



# rdt3.0 in action

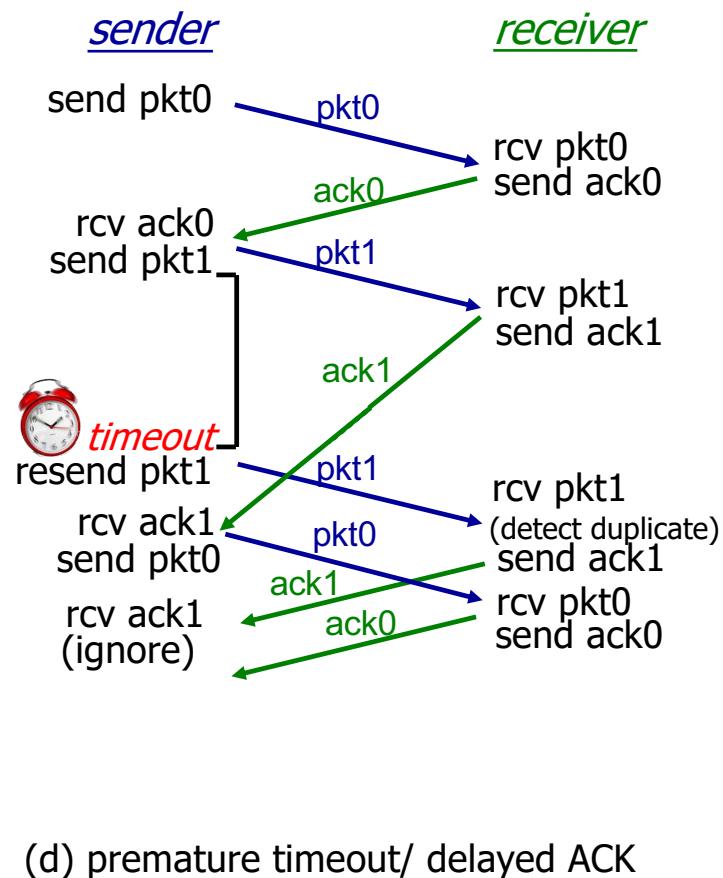
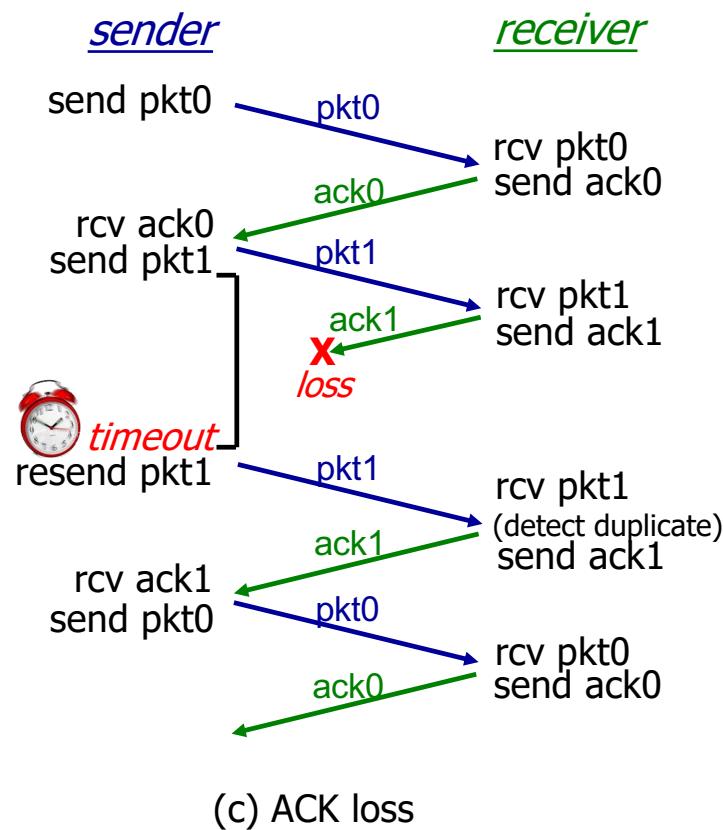


(a) no loss



(b) packet loss

## rdt3.0 in action



## Quiz: Reliable Data Transfer



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

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

## Quiz: Reliable Data Transfer



If packets (and ACKs and NACKs) could be lost which of the following is true of RDT 2.1 (or 2.2)?

- a) Reliable in-order delivery is still achieved
- b) The protocol will get stuck
- c) The protocol will continue making progress but may skip delivering some messages



## Quiz: Reliable Data Transfer

Which of the following are needed for reliable data transfer to handle packet corruption and loss? Use only as much as is strictly needed.

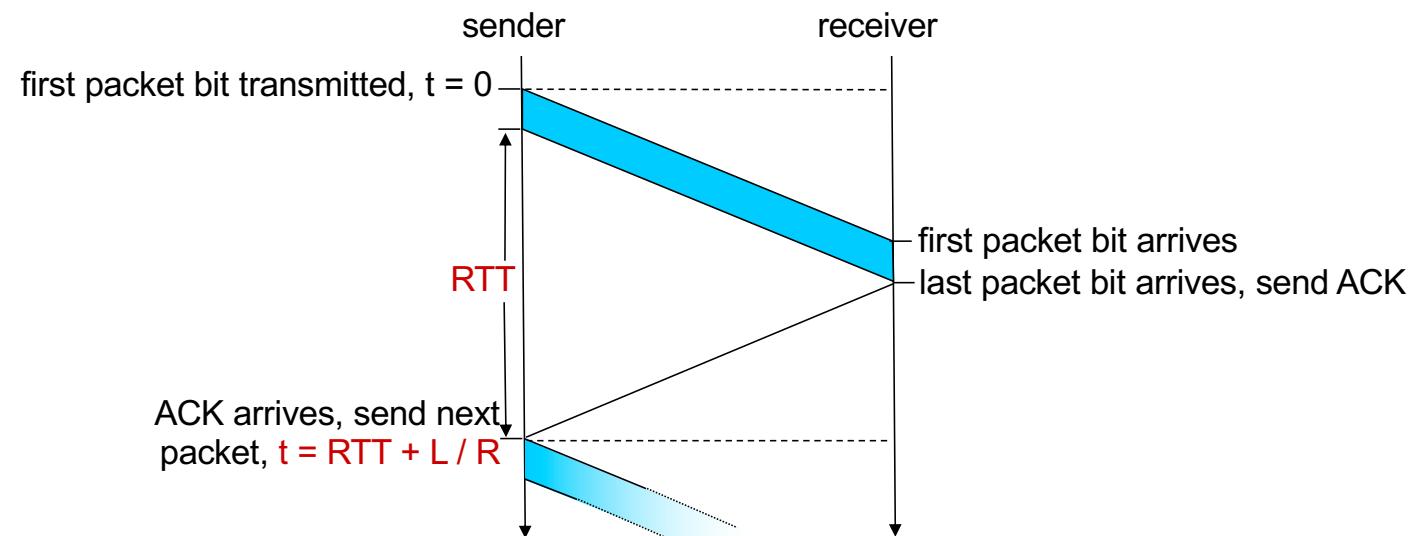
- a) Checksums, timeouts
- b) Checksums, ACKs, sequence numbers
- c) Checksums, ACKs, timeouts
- d) Checksums, ACKs, timeouts, sequence numbers
- e) Checksums, ACKs, NACKs, timeouts, sequence numbers

# Performance of rdt3.0 (stop-and-wait)

- $U_{sender}$ : *utilization* – fraction of time sender busy sending
- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
  - time to transmit packet into channel:

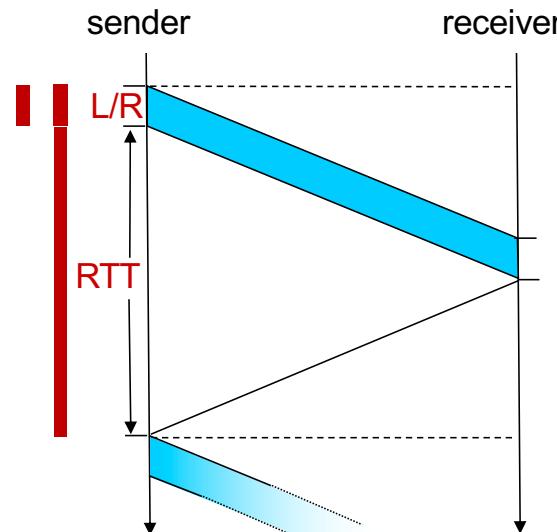
$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

# rdt3.0: stop-and-wait operation



# rdt3.0: stop-and-wait operation

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

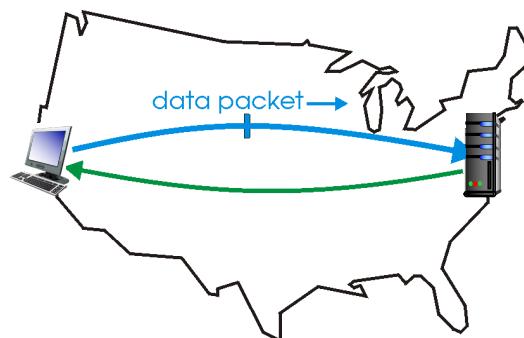


- rdt 3.0 protocol performance is very poor!
- Protocol limits performance of underlying infrastructure (channel)

# rdt3.0: pipelined protocols operation

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

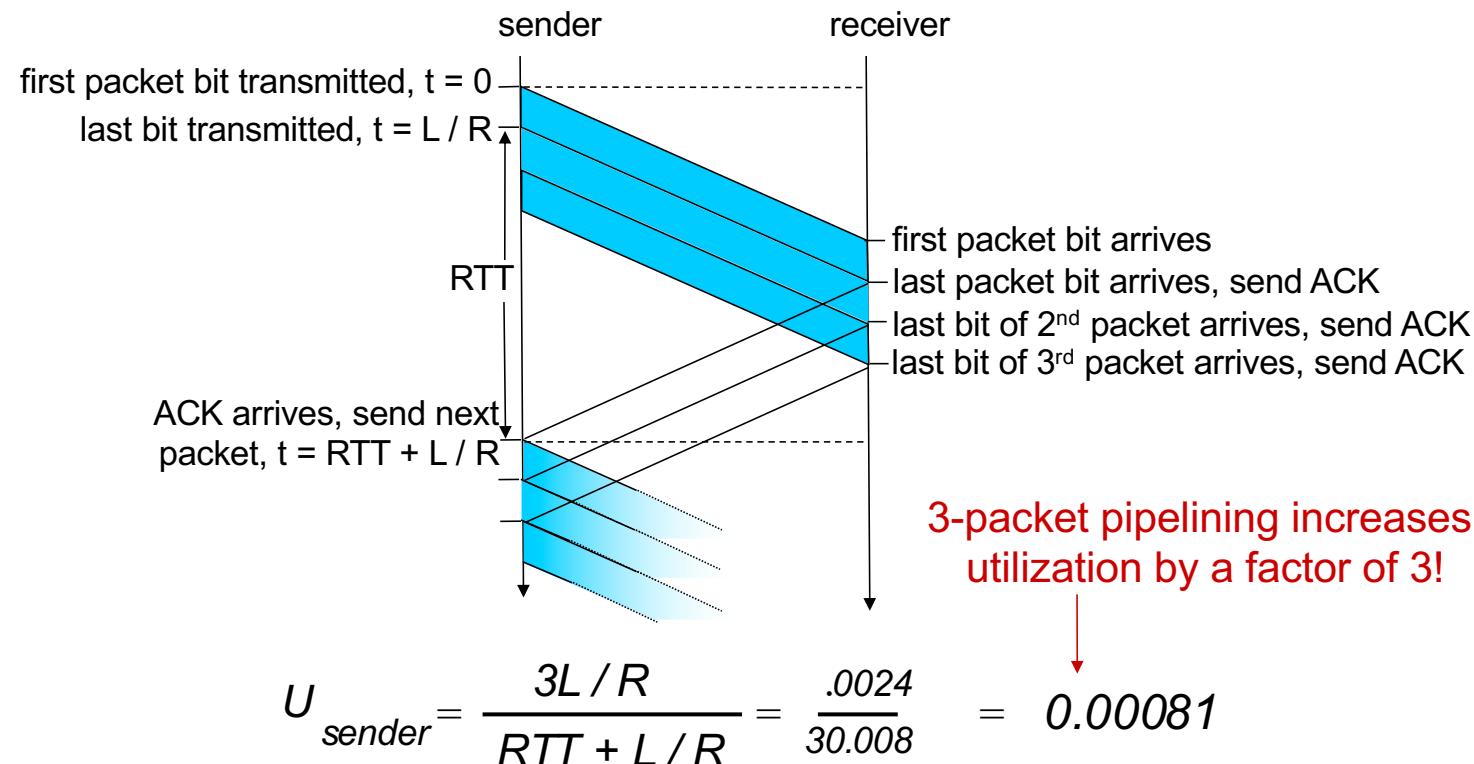
- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

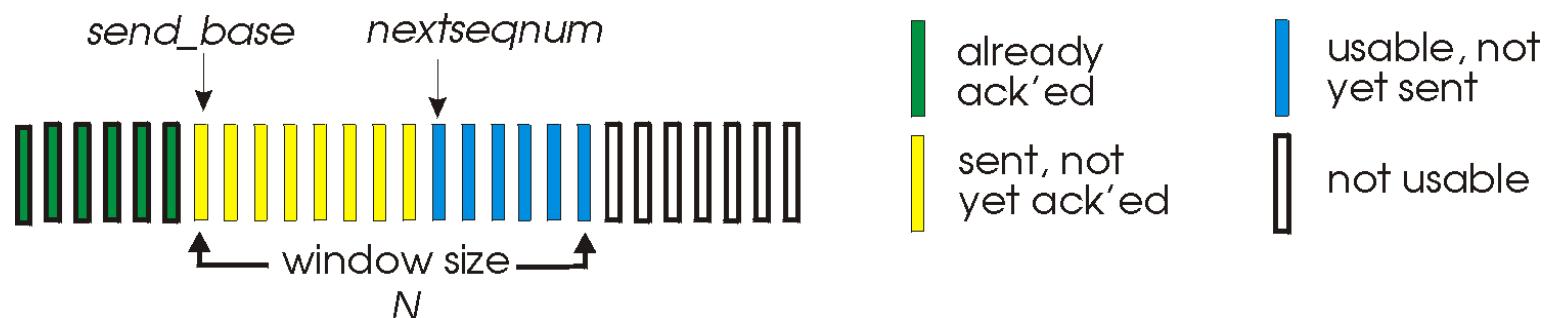
- Go Back N, Selective Repeat

# Pipelining: increased utilization



# Go-Back-N: sender

- sender: “window” of up to  $N$ , consecutive transmitted but unACKed pkts
  - $k$ -bit seq # in pkt header



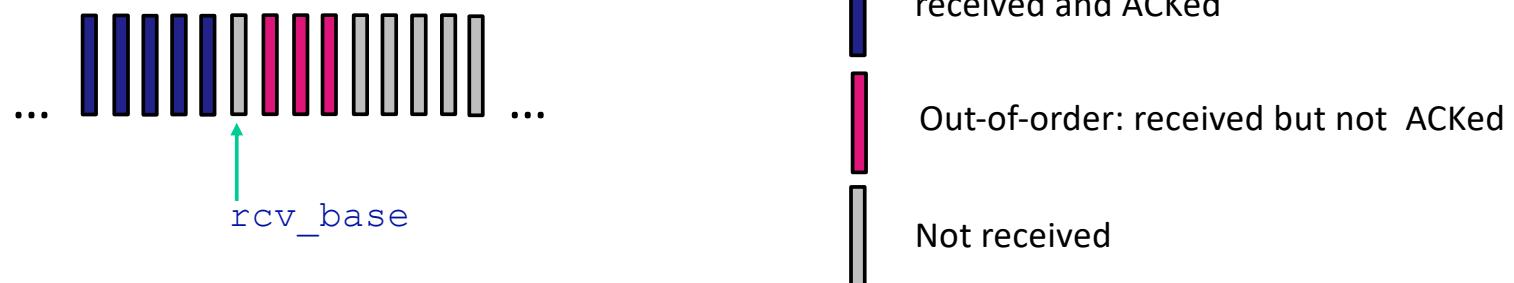
- *cumulative ACK*:  $\text{ACK}(n)$ : ACKs all packets up to, including seq #  $n$ 
  - on receiving  $\text{ACK}(n)$ : move window forward to begin at  $n+1$
- timer for oldest in-flight packet
- $\text{timeout}(n)$ : retransmit packet  $n$  and all higher seq # packets in window

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

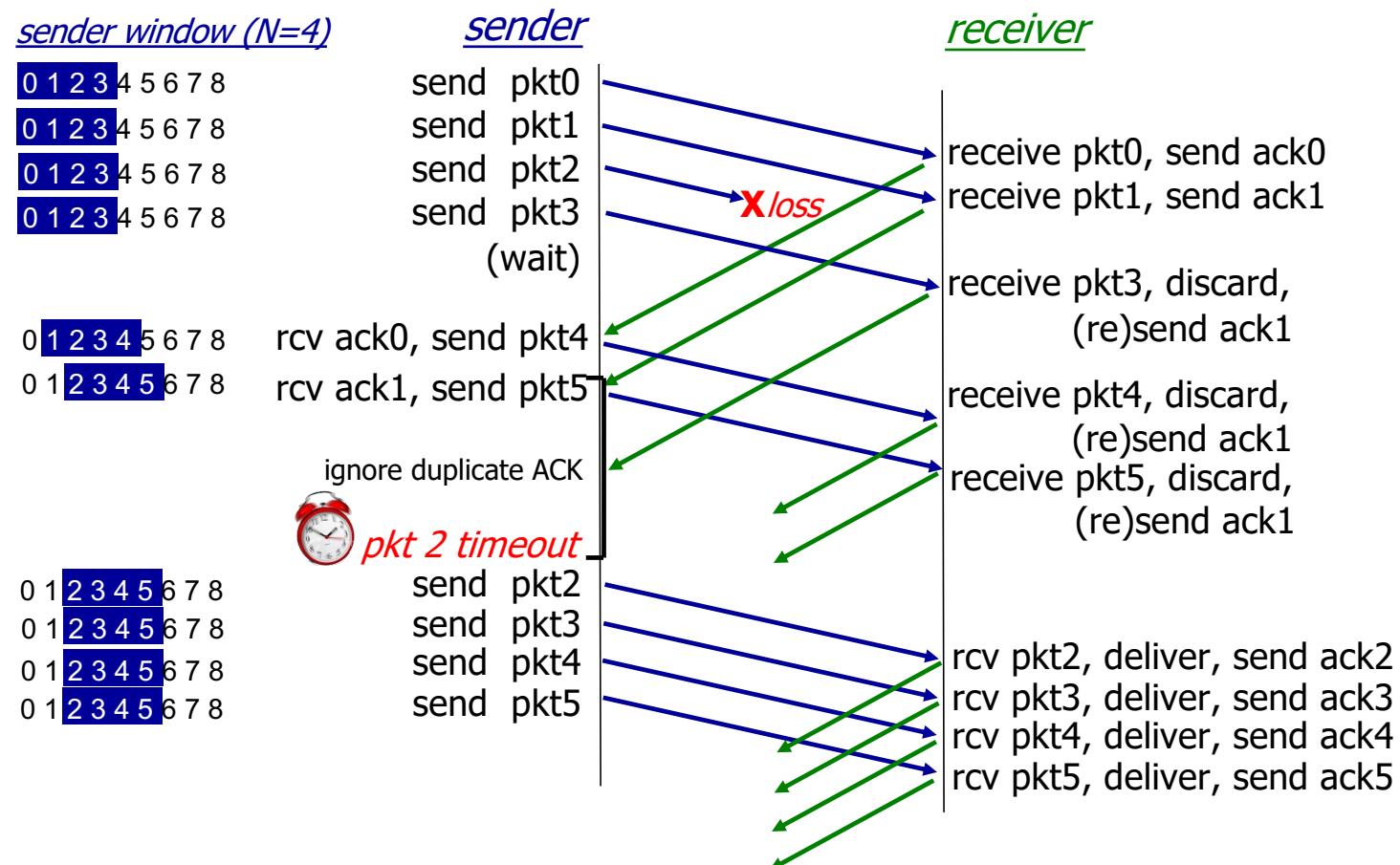
# Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:



# Go-Back-N in action



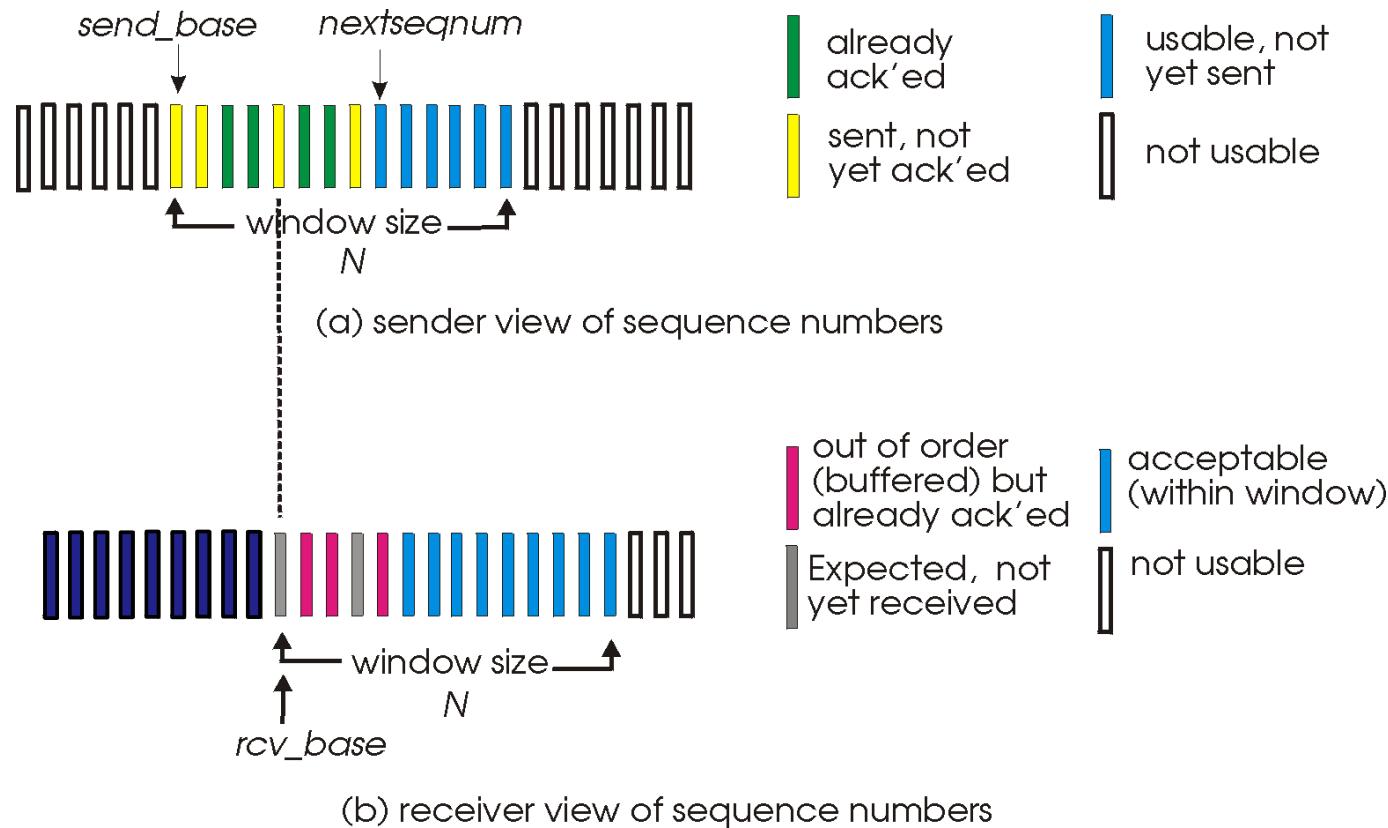
# Selective repeat

- receiver *individually* acknowledges all correctly received packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
  - sender maintains timer for each unACKed pkt
- sender window
  - $N$  consecutive seq #s
  - limits seq #s of sent, unACKed packets

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

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

# Selective repeat: sender, receiver windows



# Selective repeat: sender and receiver

## sender

**data from above:**

- if next available seq # in window, send packet

**timeout( $n$ ):**

- resend packet  $n$ , restart timer

**ACK( $n$ ) in [sendbase, sendbase+N]:**

- mark packet  $n$  as received
- if  $n$  smallest unACKed packet, advance window base to next unACKed seq #

## receiver

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

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

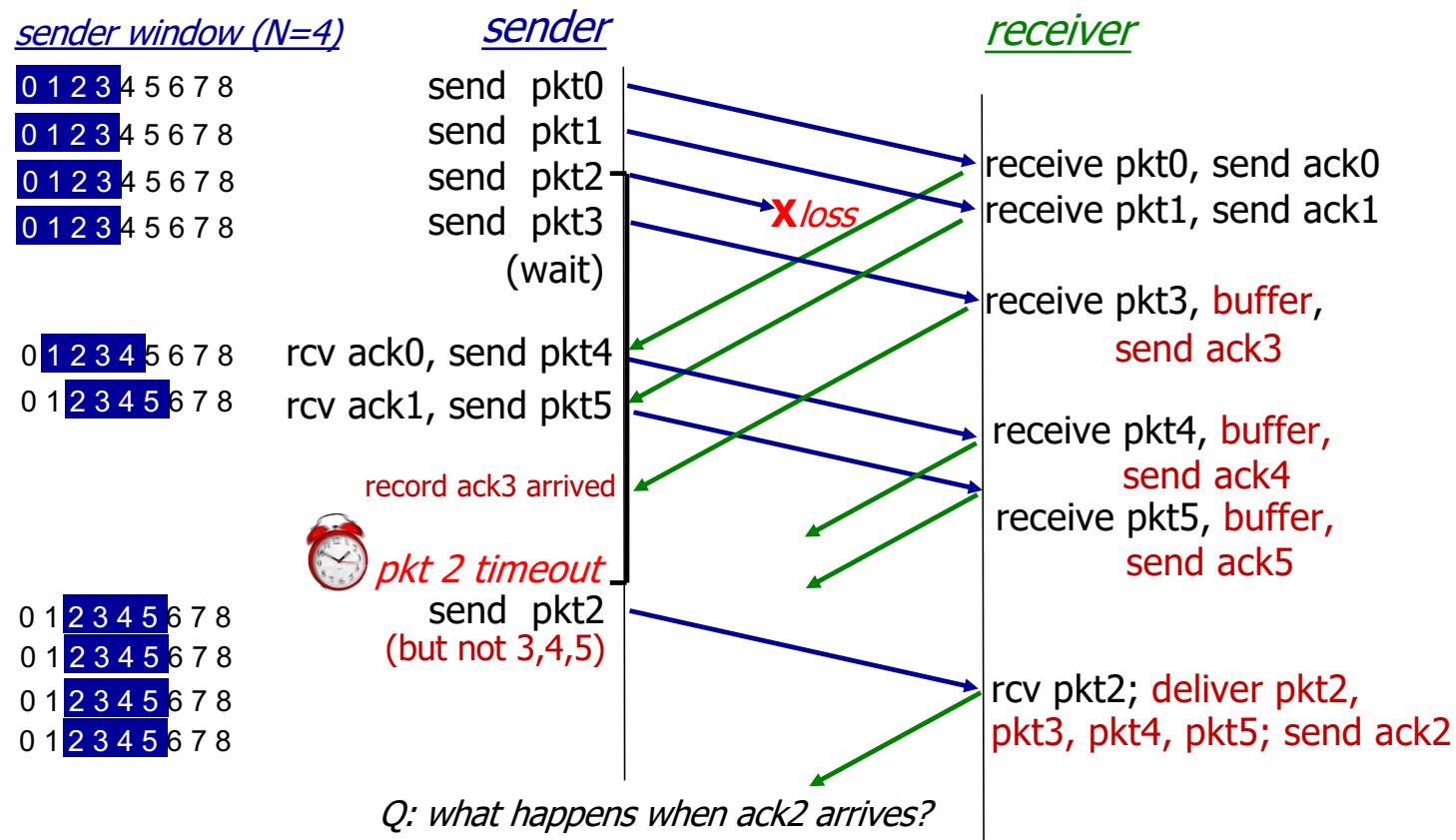
**packet  $n$  in [rcvbase-N, rcvbase-1]**

- ACK( $n$ )

**otherwise:**

- ignore

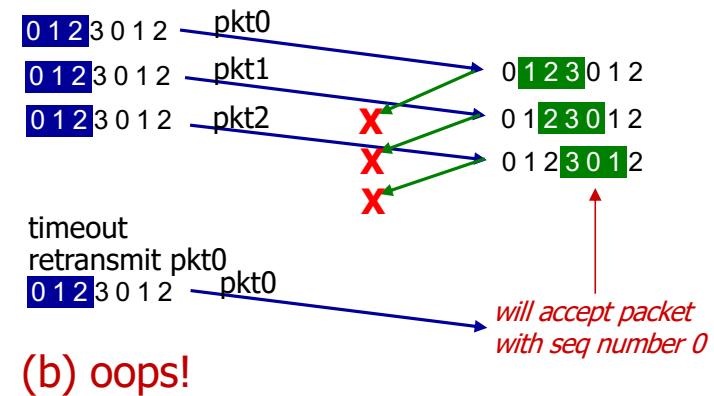
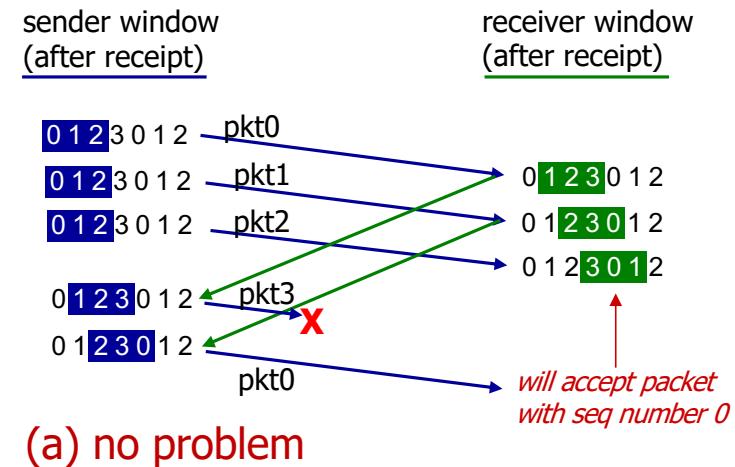
# Selective Repeat in action



# Selective repeat: a dilemma!

example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3



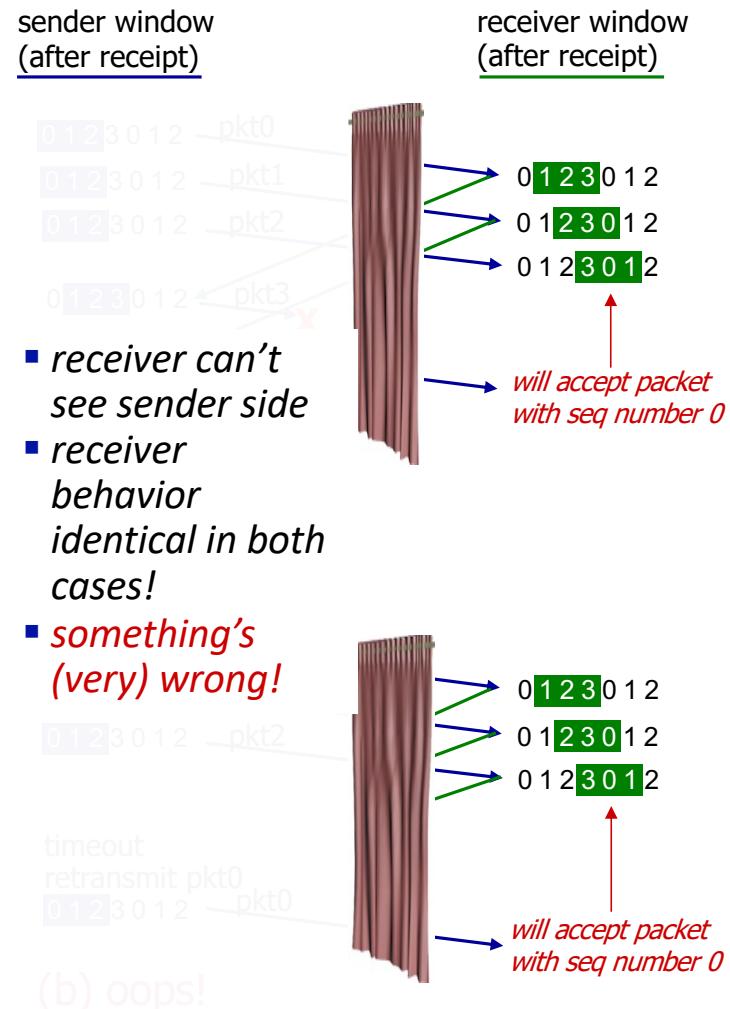
# Selective repeat: a dilemma!

example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3

**Q:** what relationship is needed between sequence # size and window size to avoid problem in scenario (b)?

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



# Recap: components of a solution

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

# Practical Reliability Questions

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

**We will answer these questions when we cover TCP**



## Quiz: GBN, SR

Which of the following is **not** true?

- a) GBN uses cumulative ACKs, SR uses individual ACKs
- b) Both GBN and SR use timeouts to address packet loss
- c) GBN maintains a separate timer for each outstanding packet
- d) SR maintains a separate timer for each outstanding packet
- e) Neither GBN nor SR use NACKs

## Quiz: GBN, SR



Suppose a receiver that has received all packets up to and including sequence number 24 and next receives packet 27 and 28. In response, what are the sequence numbers in the ACK(s) sent out by the GBN and SR receiver, respectively?

- a) [27, 28], [28, 28]
- b) [24, 24], [27, 28]
- c) [27, 28], [27, 28]
- d) [25, 25], [25, 25]
- e) [nothing], [27, 28]

# Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and  
demultiplexing

3.3 connectionless transport:  
UDP

3.4 principles of reliable data  
transfer

3.5 connection-oriented  
transport: TCP

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

3.6 principles of congestion  
control

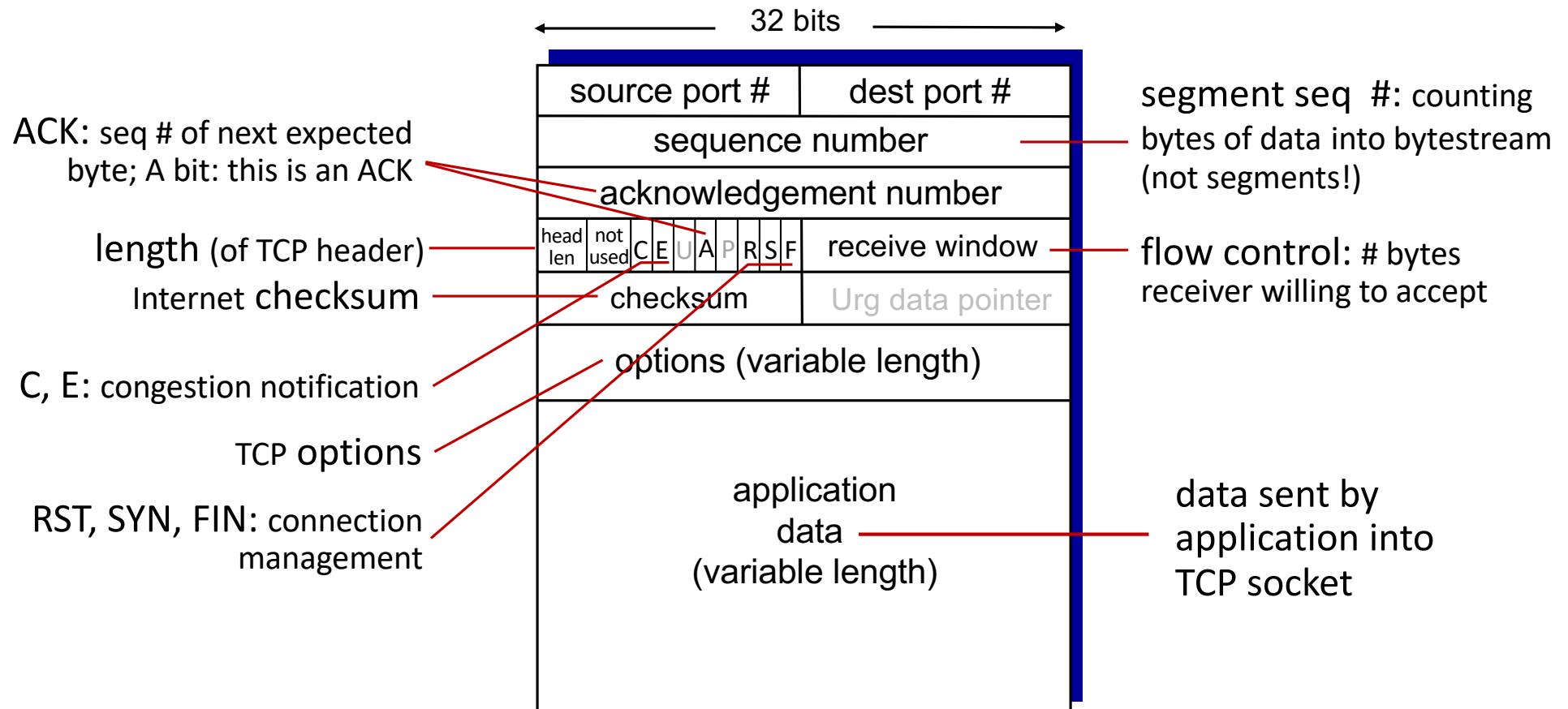
3.7 TCP congestion control

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

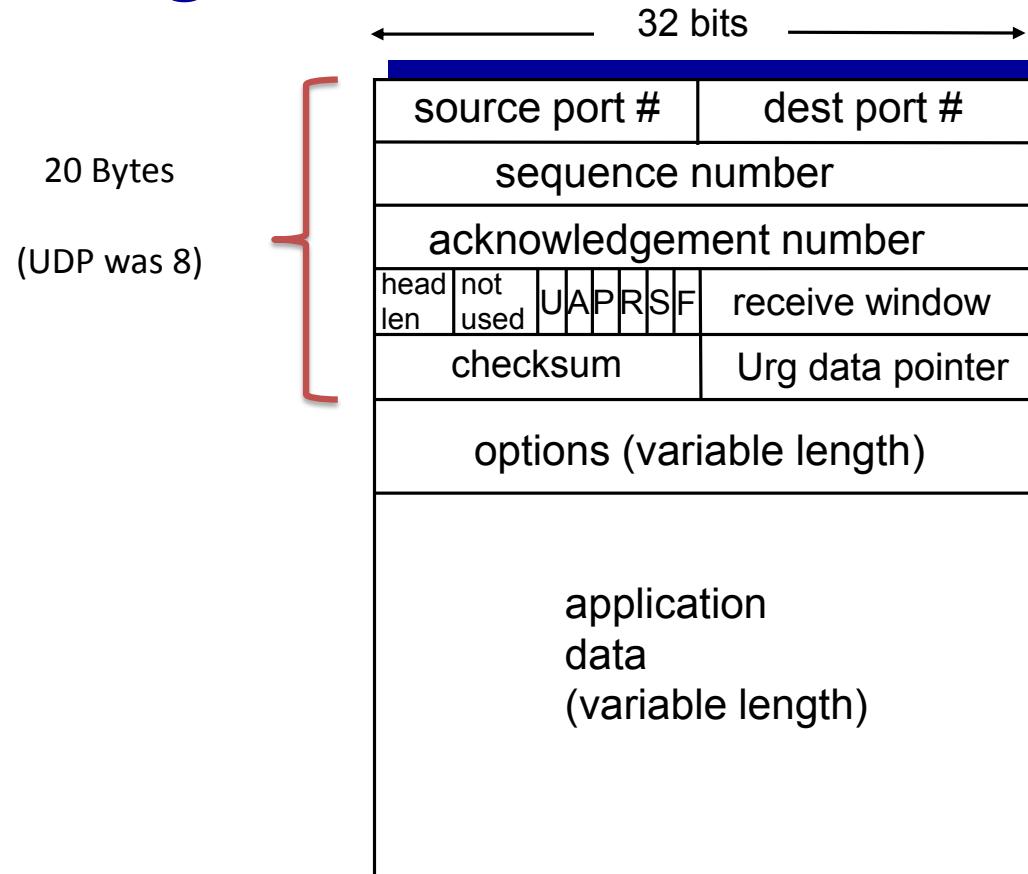
- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream*:
  - no “message boundaries”
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- cumulative ACKs
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver



# TCP segment structure



# TCP segment structure



# Summary

- ❖ Multiplexing/Demultiplexing
- ❖ UDP
- ❖ Reliable Data Transfer
  - Stop-and-wait protocols
  - Sliding window protocols
- ❖ TCP - intro
- ❖ Up Next:
  - TCP – continued in more detail
  - Congestion Control

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

**Week 5**

**Transport Layer (Continued)**

1. TCP Reliability
2. TCP Flow Control
3. TCP Connection management
4. TCP Congestion Control

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

# Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and  
demultiplexing

3.3 connectionless transport:  
UDP

3.4 principles of reliable data  
transfer

3.5 connection-oriented  
transport: TCP

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

3.6 principles of congestion  
control

3.7 TCP congestion control

# Recall: Components of a solution for reliable transport

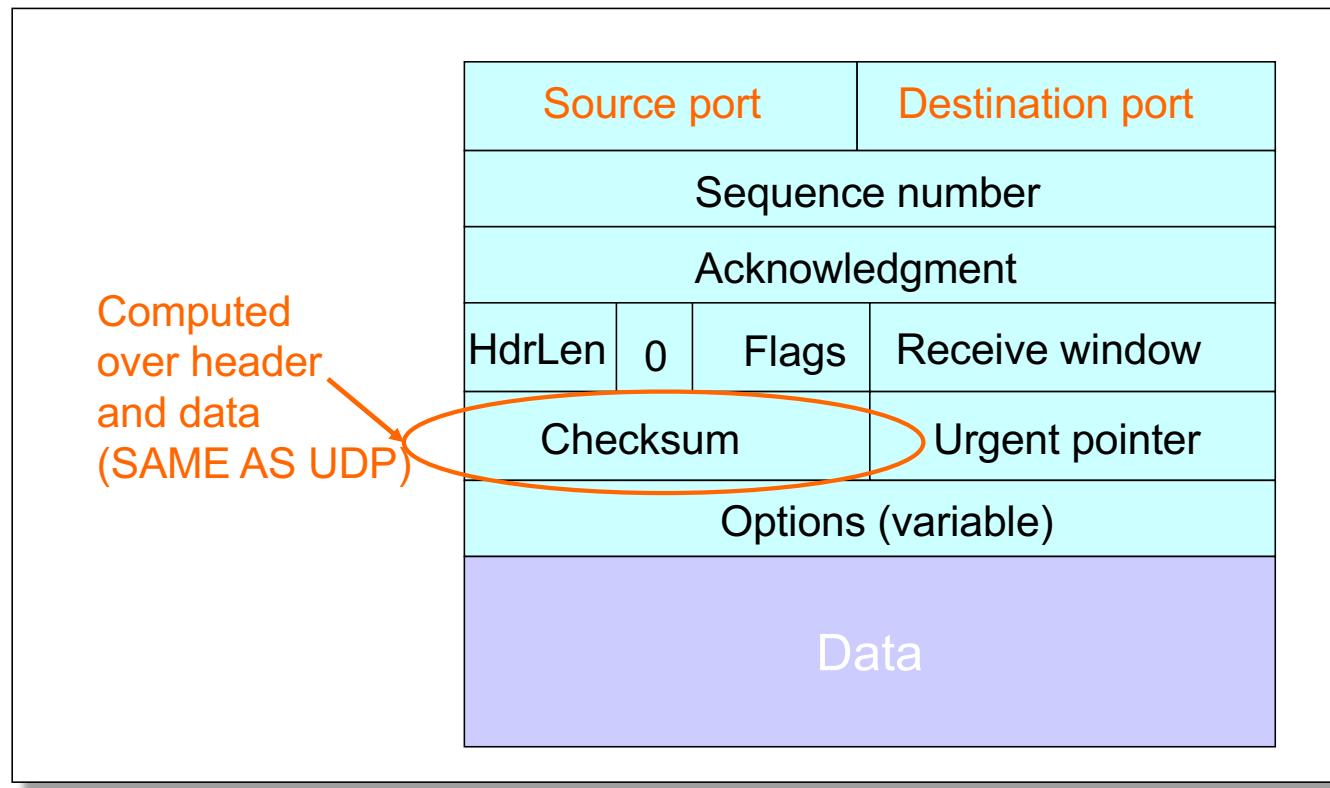
- ❖ Checksums (for error detection)
- ❖ Timers (for loss detection)
- ❖ Acknowledgments
  - Cumulative
  - Selective
- ❖ Sequence numbers (duplicates, windows)
- ❖ Sliding Windows (for efficiency)
  - Go-Back-N (GBN)
  - Selective Repeat (SR)

# What does TCP do?

Many of our previous ideas, but some key differences

- ❖ Checksum

# TCP Header



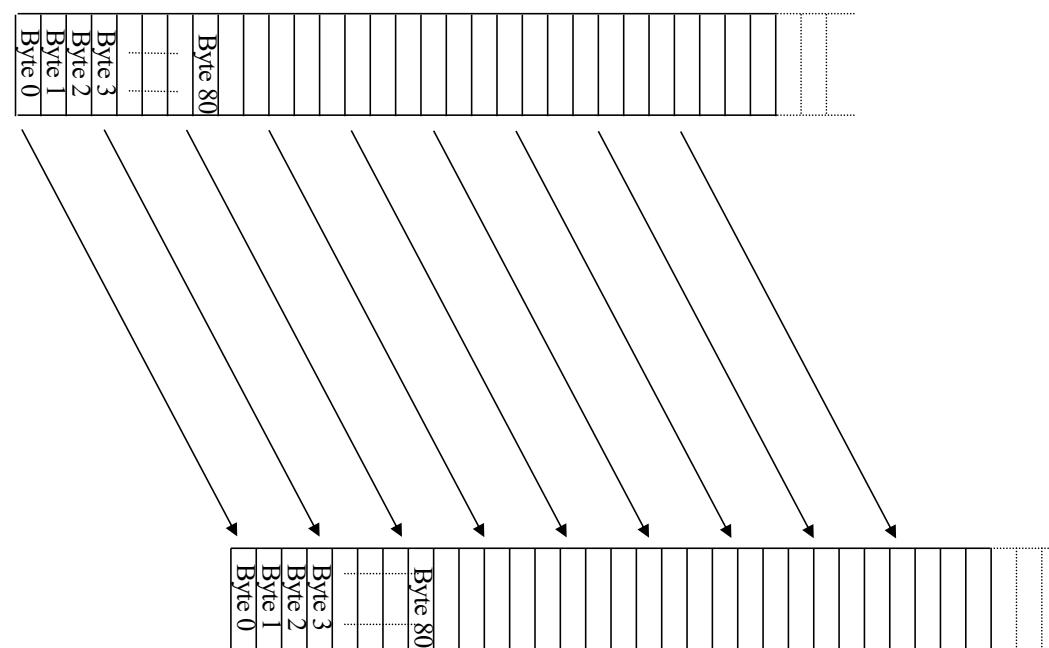
# What does TCP do?

Many of our previous ideas, but some key differences

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

# TCP “Stream of Bytes” Service ..

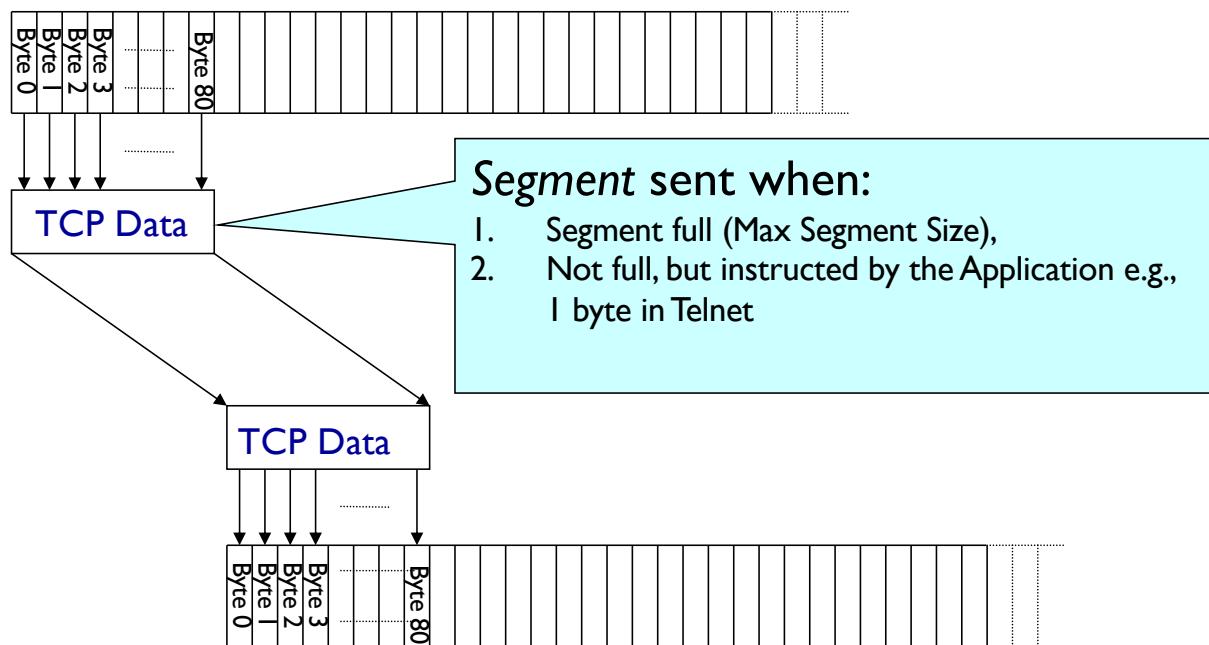
Application @ Host A



Application @ Host B

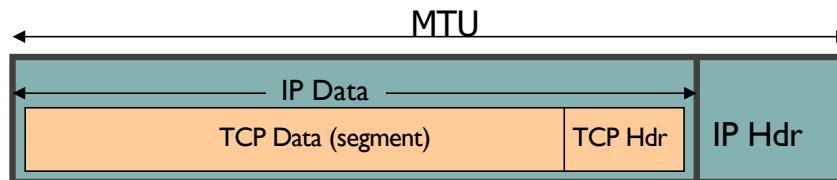
# .. Provided Using TCP “Segments”

Host A



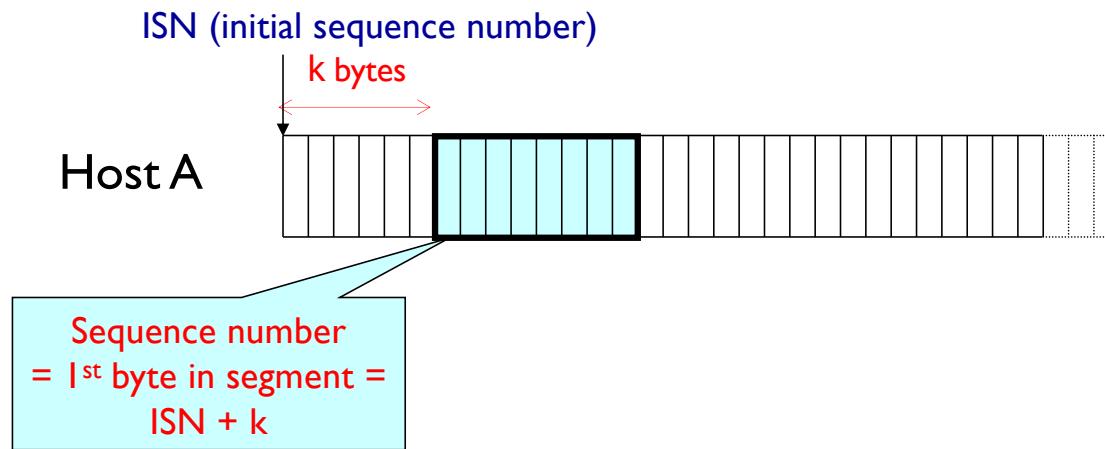
Host B

# TCP Maximum Segment Size



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

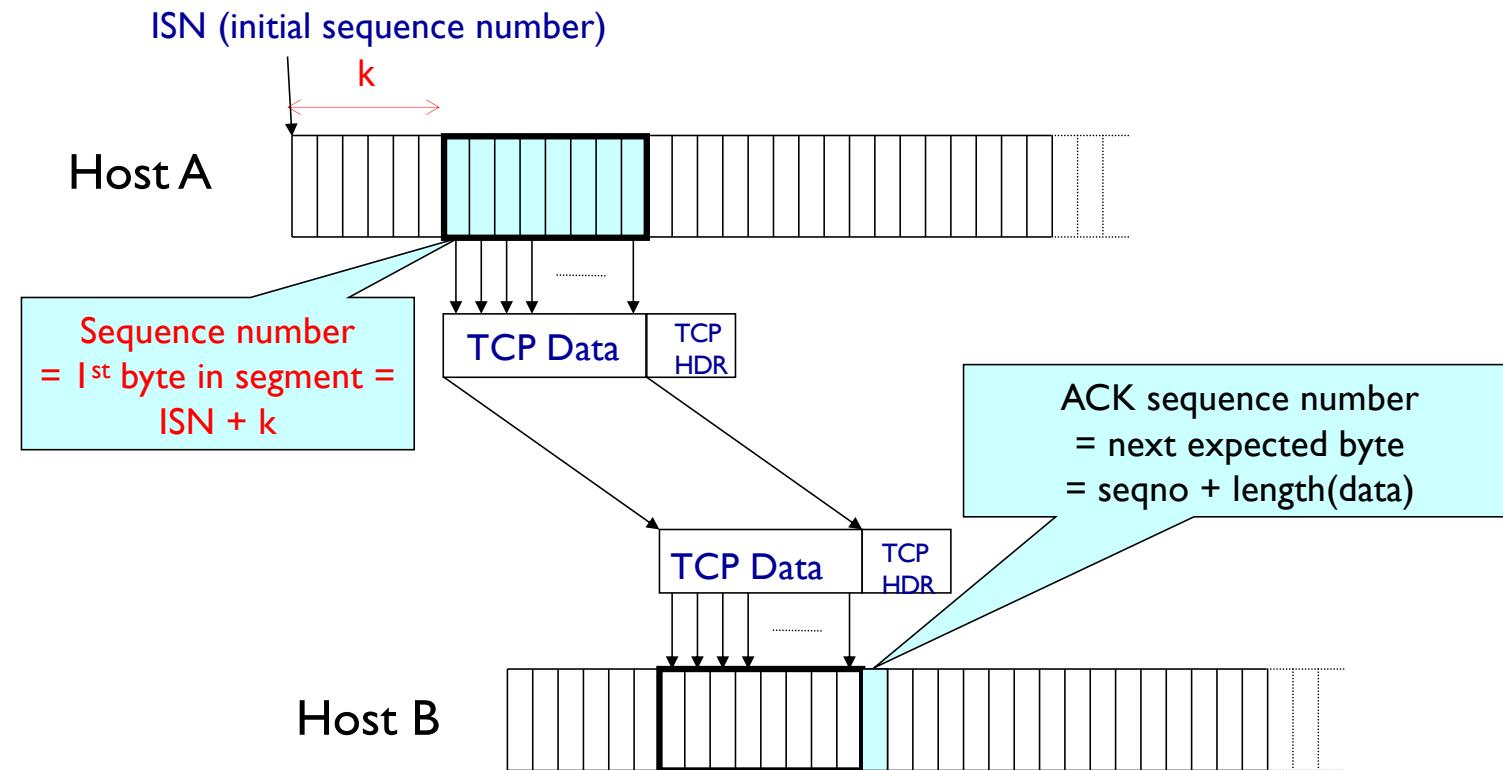
# Sequence Numbers



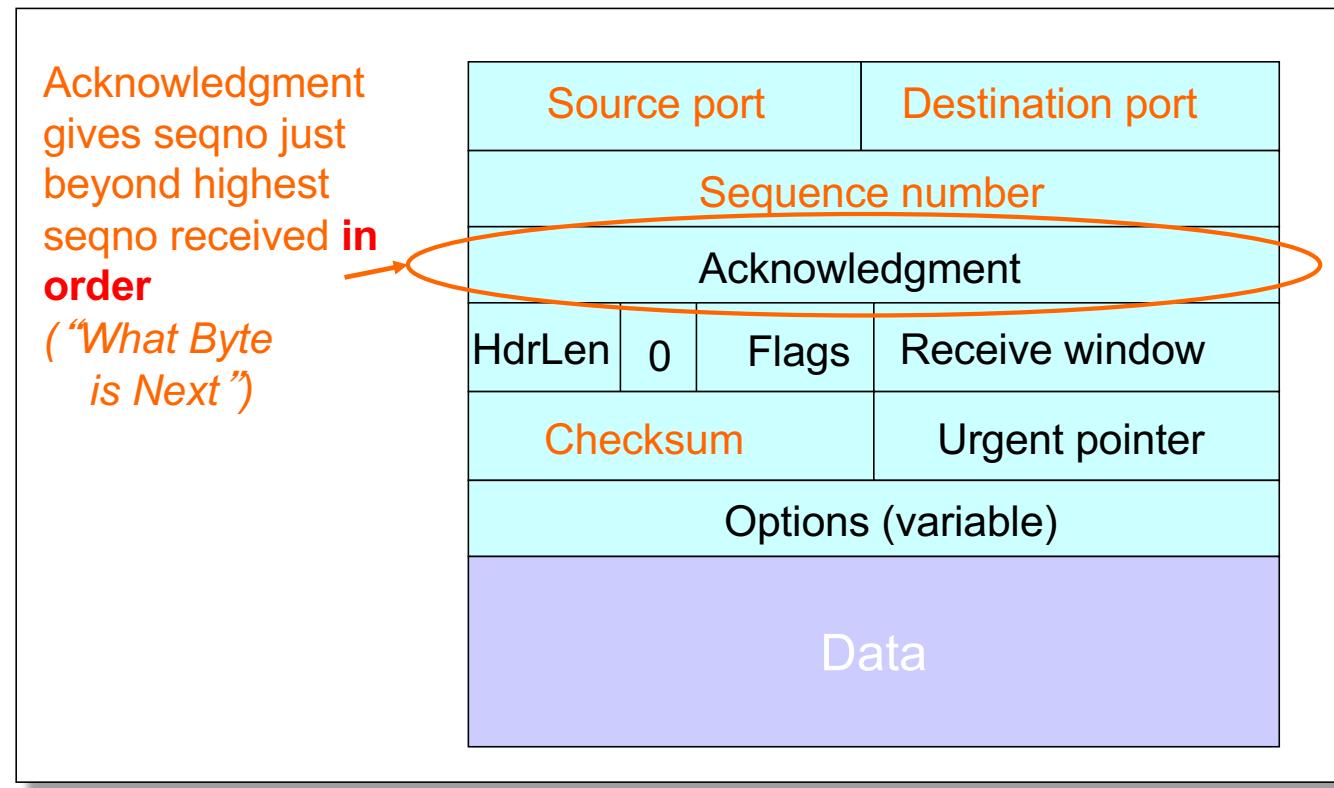
## Sequence numbers:

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

# Sequence & Ack Numbers

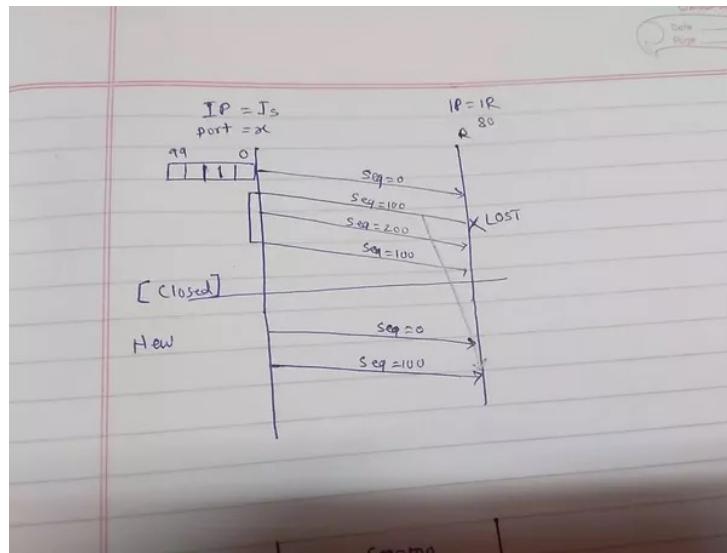


# TCP Header

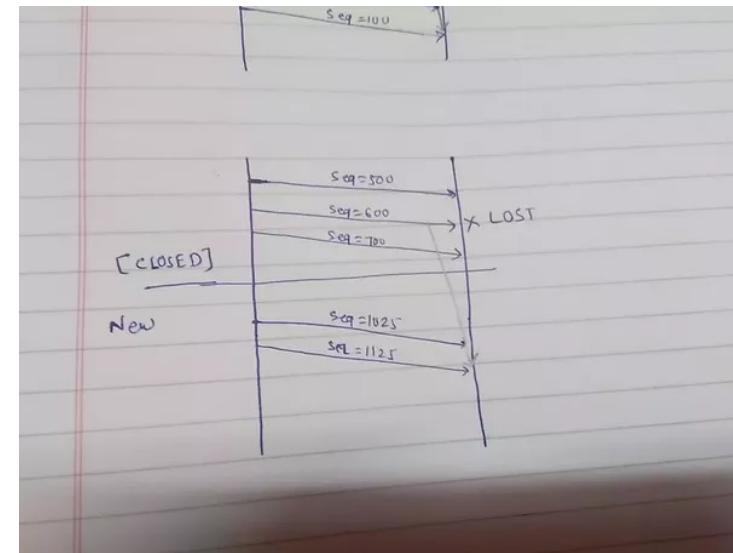


# Why choose random ISN?

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



(a) When ISN=0



(b) When ISN is random

- ❖ Potential security issue if the ISN is known

# What does TCP do?

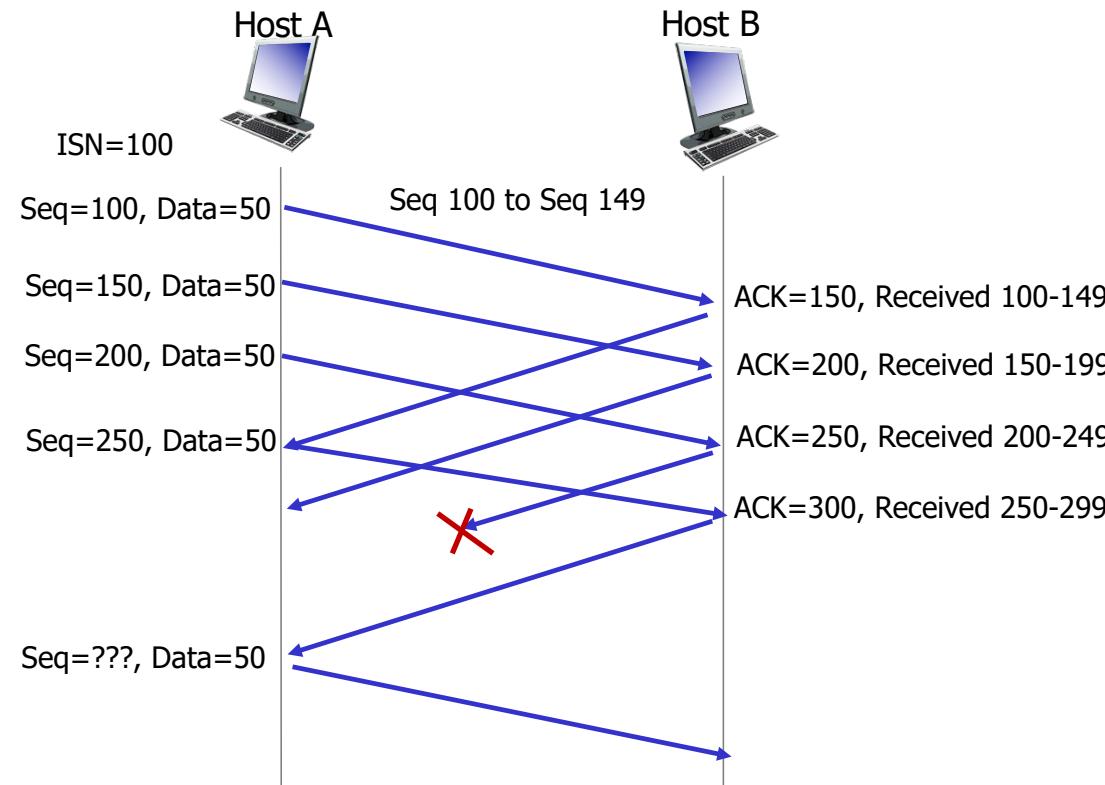
Most of our previous tricks, but a few differences

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

# ACKing and Sequence Numbers

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

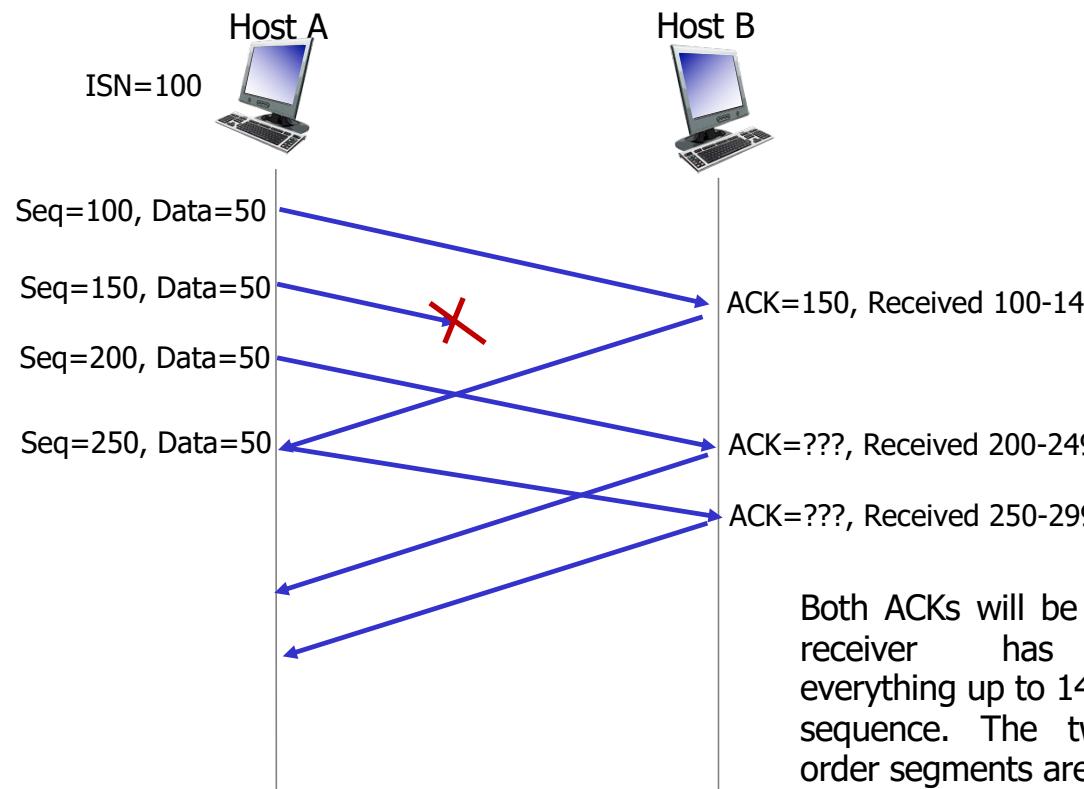
# An Example



Seq = 300 (new segment)

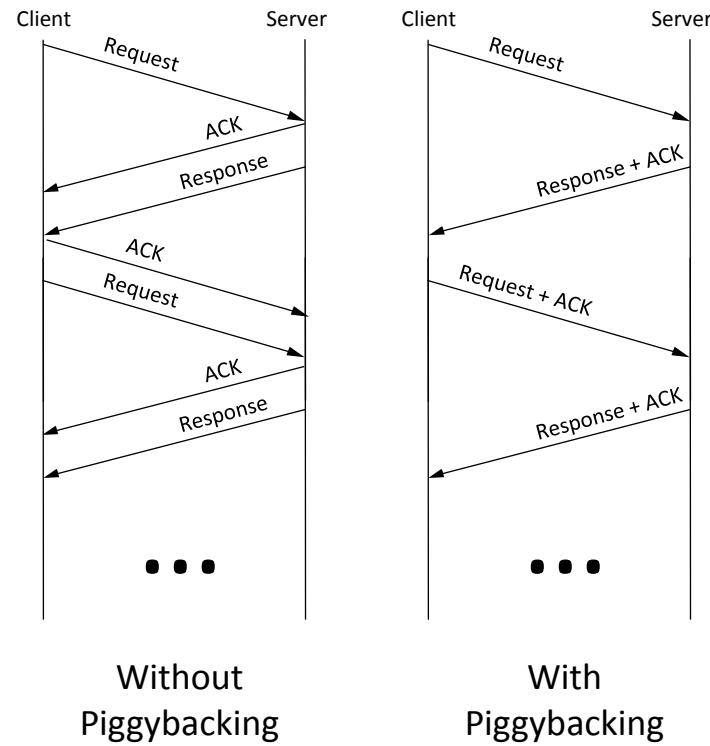
Since TCP uses cumulative ACKs, the receipt of ACK 300 before a timeout (for seg with sequence number 200) implies the receiver has received all 4 segments sent above

# Another Example

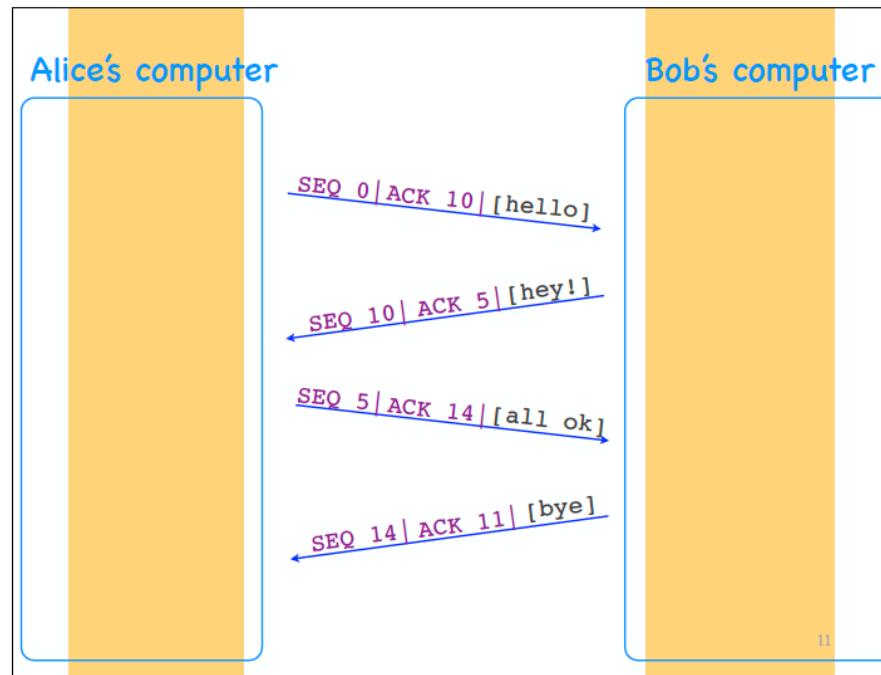


# Piggybacking

- ❖ So far, we've assumed distinct “sender” and “receiver” roles
- ❖ Usually both sides of a connection (i.e. the application processes) send some data

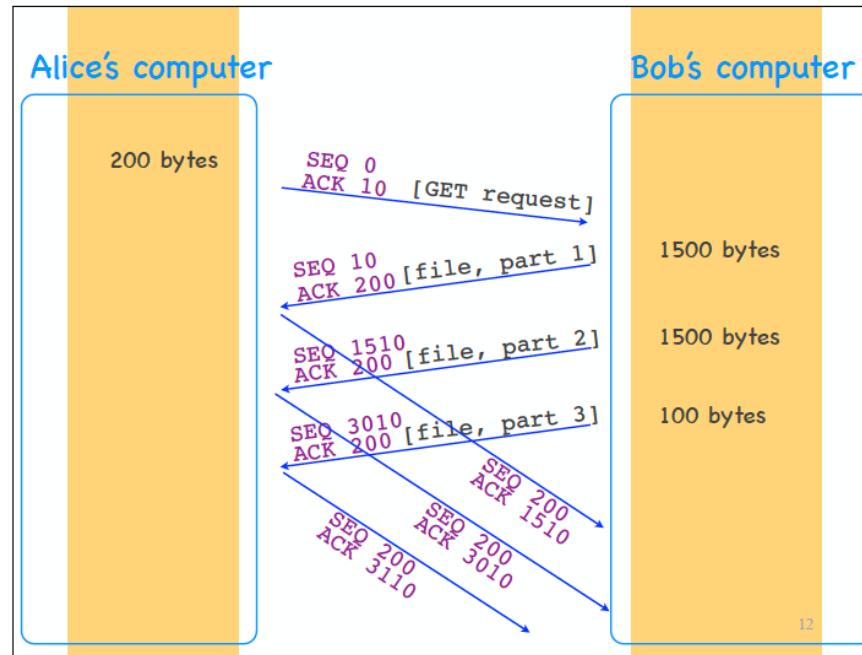


# Example



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

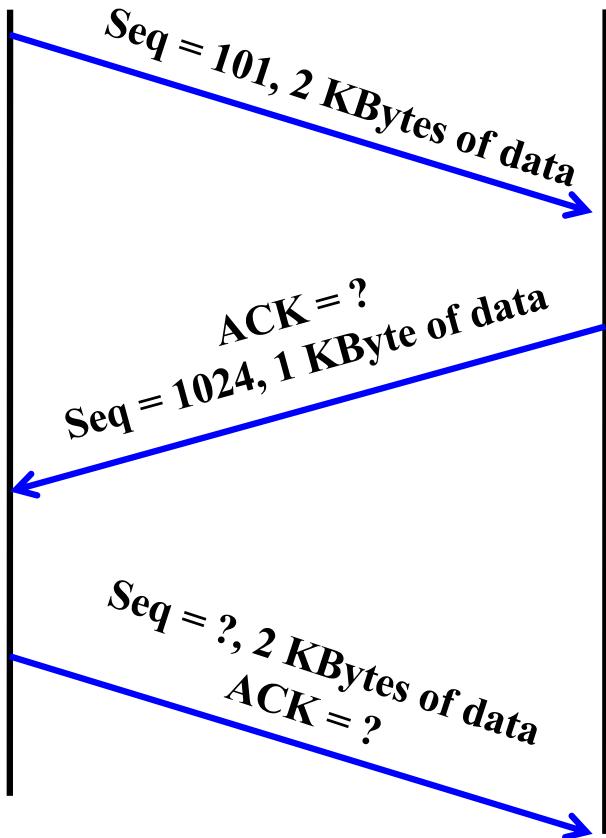
# Another Example



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

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

# Quiz



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

$$\text{Seq} = 2149$$

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

# What does TCP do?

Most of our previous tricks, but a few differences

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

## Loss with cumulative ACKs

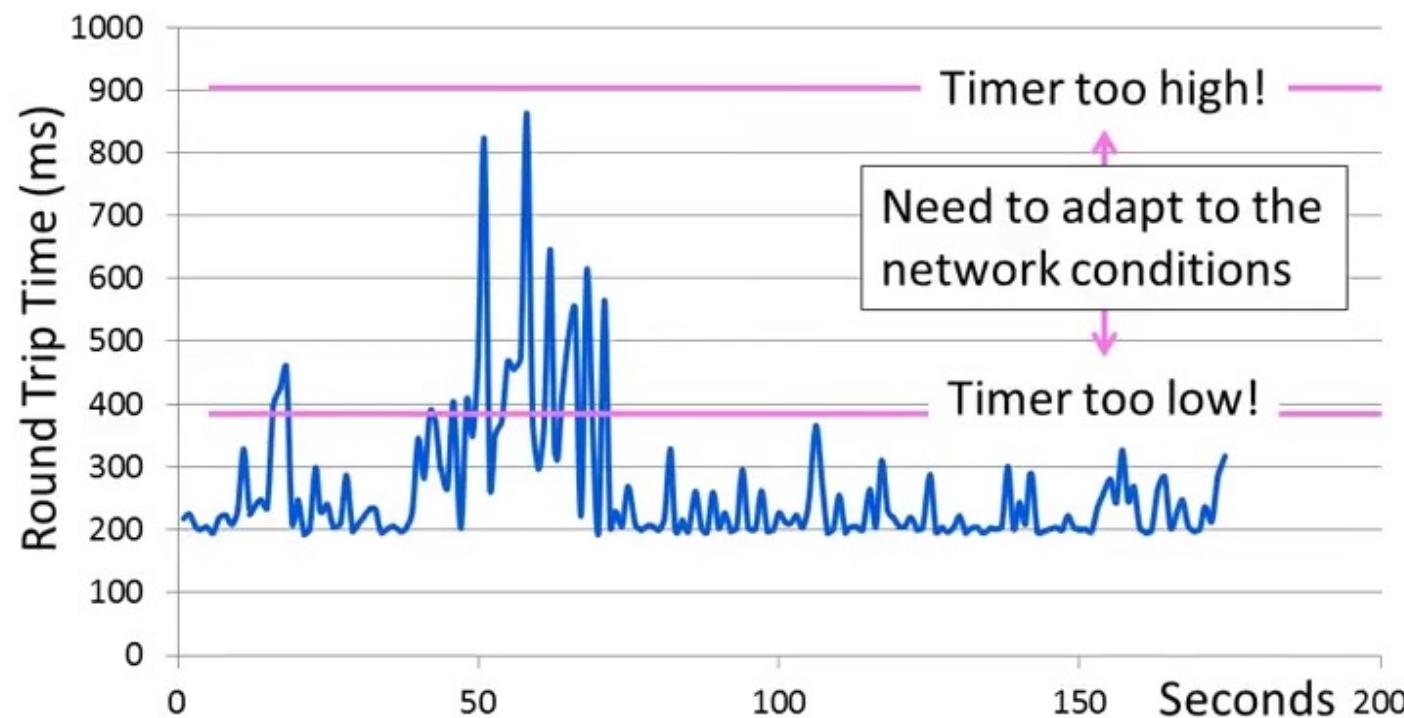
- ❖ Sender sends packets with 100 bytes and sequence numbers:
  - 100, 200, 300, 400, 500, 600, 700, 800, 900, ...
- ❖ Assume the fifth packet (seq. no. 500) is lost, but no others
- ❖ 6th packet onwards are buffered
- ❖ Stream of ACKs will be:
  - 200, 300, 400, 500, 500, 500, ...

# What does TCP do?

Most of our previous tricks, but a few differences

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

# TCP round trip time, timeout



# TCP round trip time, timeout

Q: how to set TCP timeout value?

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

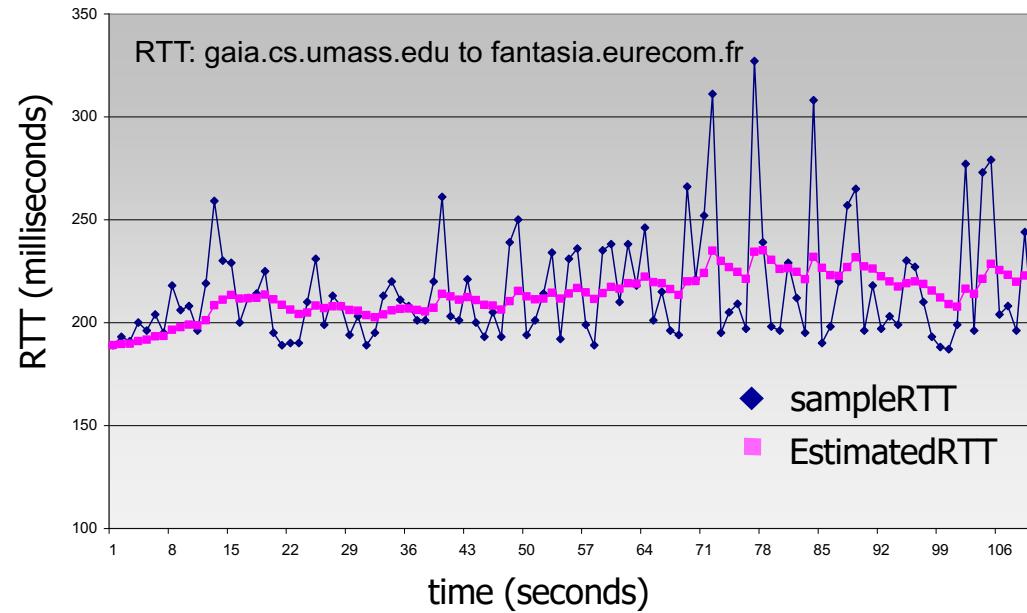
Q: how to estimate RTT?

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

# TCP round trip time, timeout

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

- exponential weighted moving average (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**:

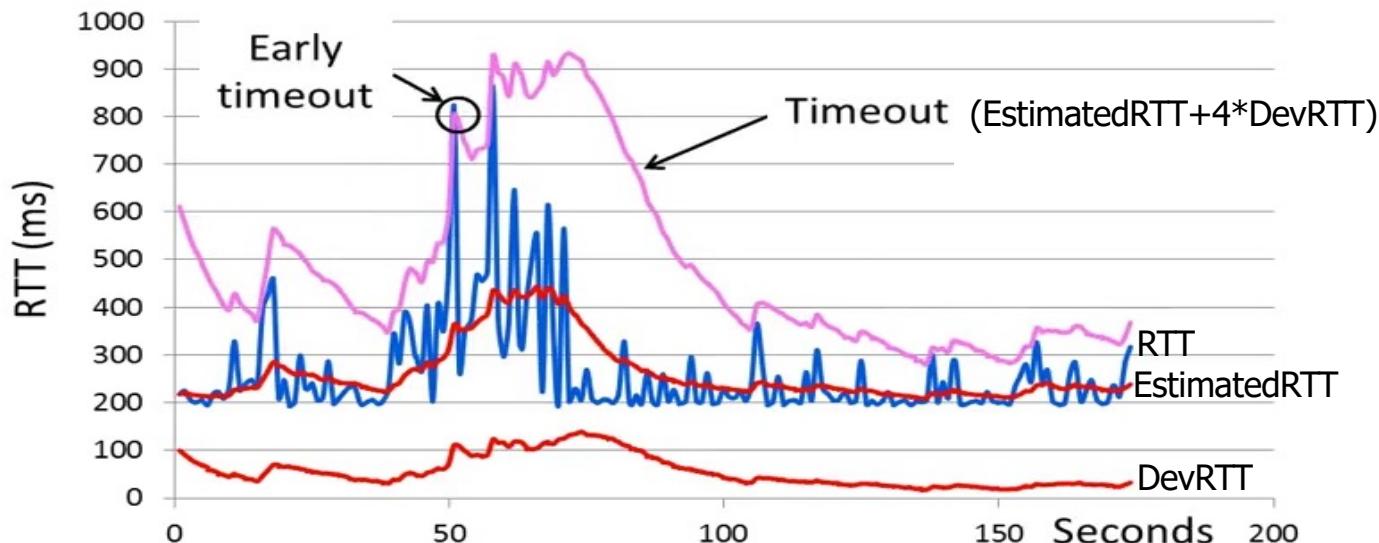
$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically,  $\beta = 0.25$ )

Practice Problem:

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

# TCP round trip time, timeout



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



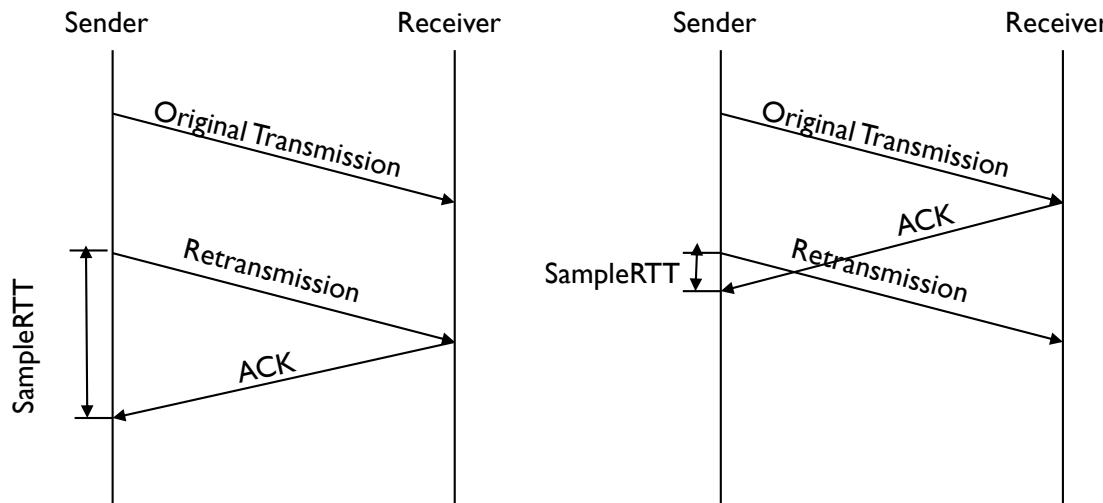
estimated RTT

“safety margin”

Figure: Credits Prof David Wetherall UoW

# Why exclude retransmissions in RTT computation?

- ❖ How do we differentiate between the real ACK, and ACK of the retransmitted packet?
- ❖ Sender cannot differentiate between the two scenarios shown below



# TCP Sender (simplified)

*event: data received from application*

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

*event: timeout*

- retransmit segment that caused timeout
- restart timer

*event: ACK received*

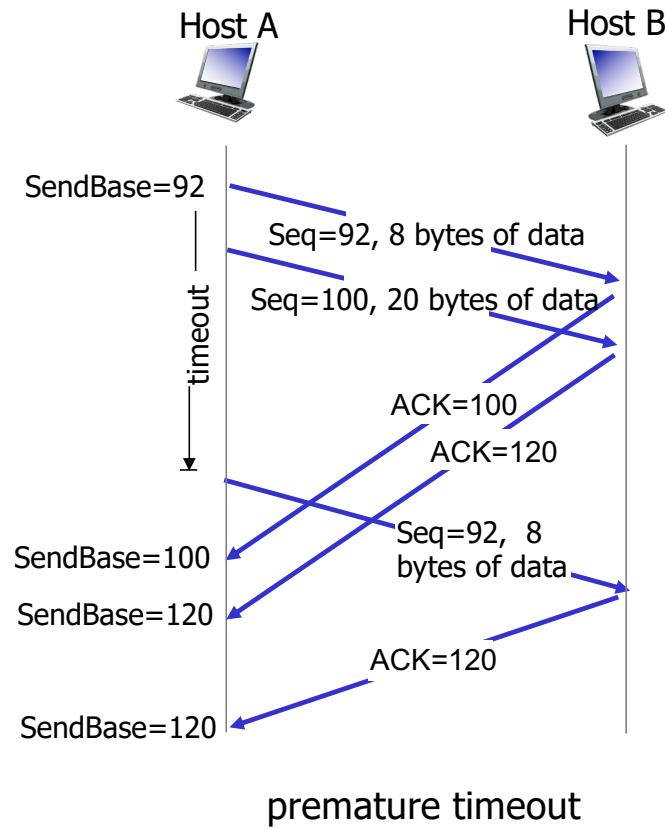
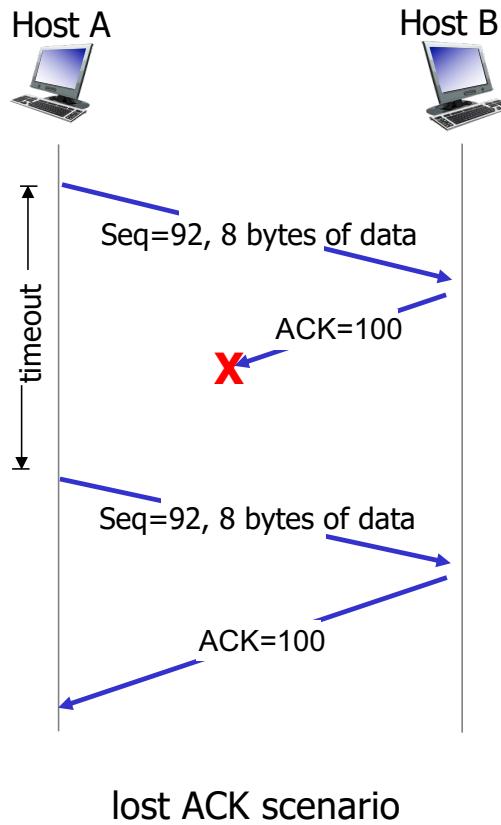
- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

# TCP ACK generation [RFC 1122, RFC 2581]

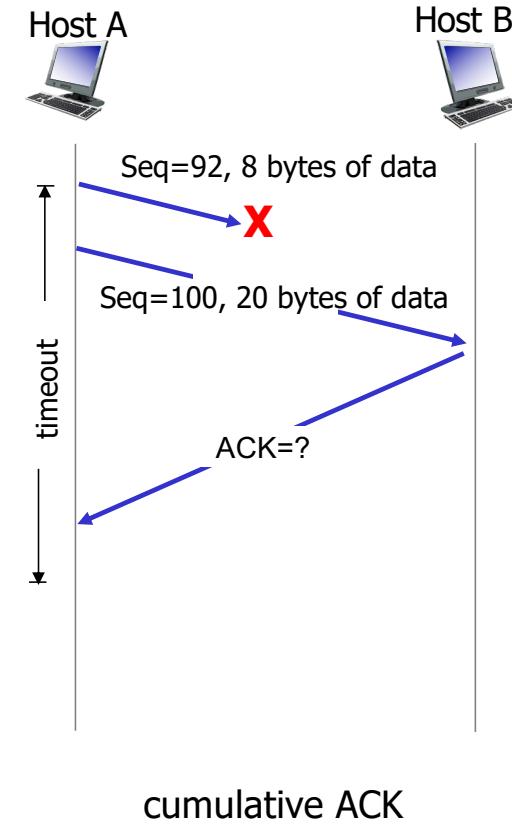
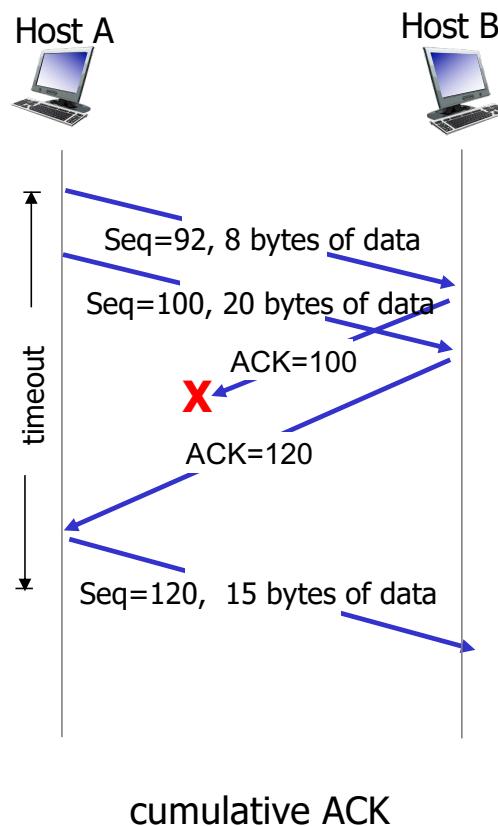
**Note: You may neglect delayed ACKs in the exam unless explicitly told to consider it**

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

# TCP: retransmission scenarios



# TCP: retransmission scenarios



# What does TCP do?

Most of our previous tricks, but a few differences

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

# TCP fast retransmit

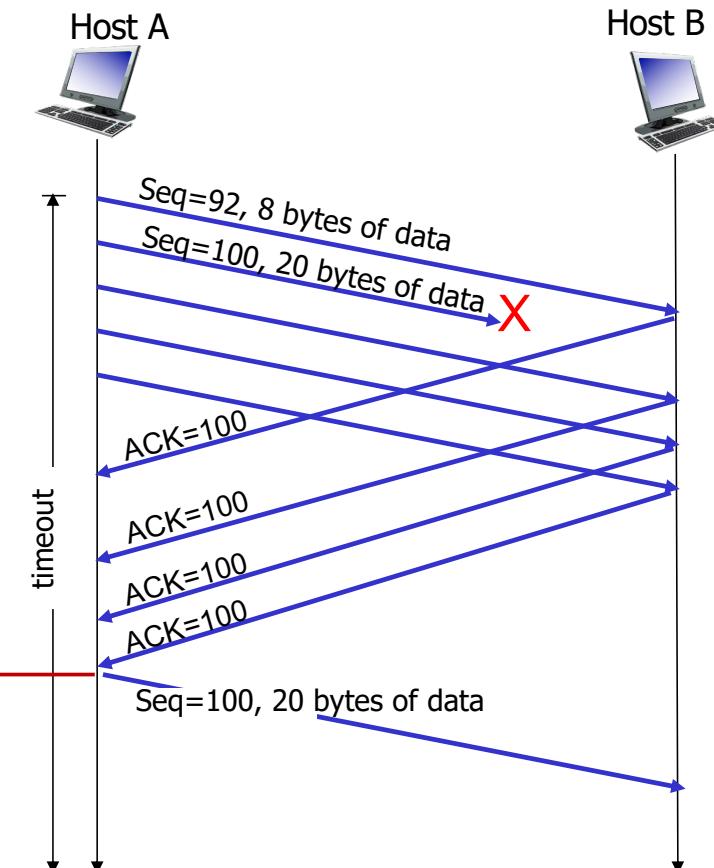
## *TCP fast retransmit*

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

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



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





## Quiz: TCP Sequence Numbers?

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

- A. 123
- B. 436
- C. 435
- D. 1334
- E. 536



## Quiz: TCP Sequence Numbers?

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

- A. Yes
- B. No



## Quiz: TCP Sequence Numbers?

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

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

# Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and  
demultiplexing

3.3 connectionless transport:  
UDP

3.4 principles of reliable data  
transfer

3.5 connection-oriented  
transport: TCP

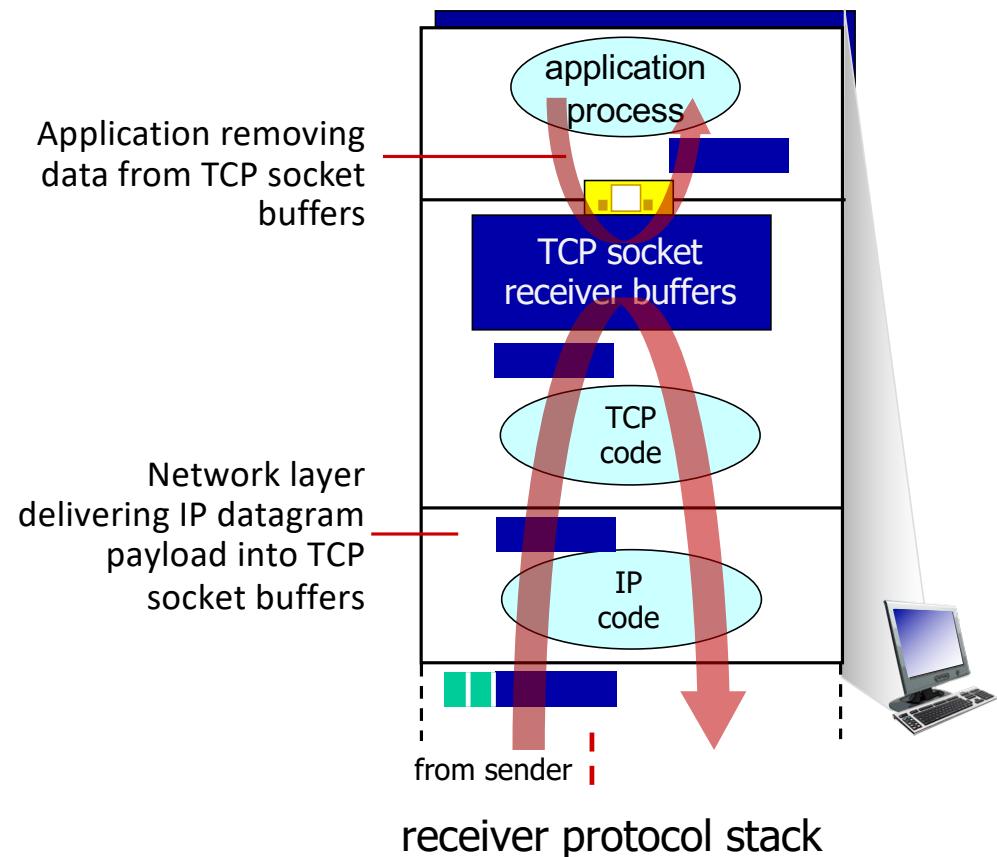
- segment structure
- reliable data transfer
- **flow control**
- connection management

3.6 principles of congestion  
control

3.7 TCP congestion control

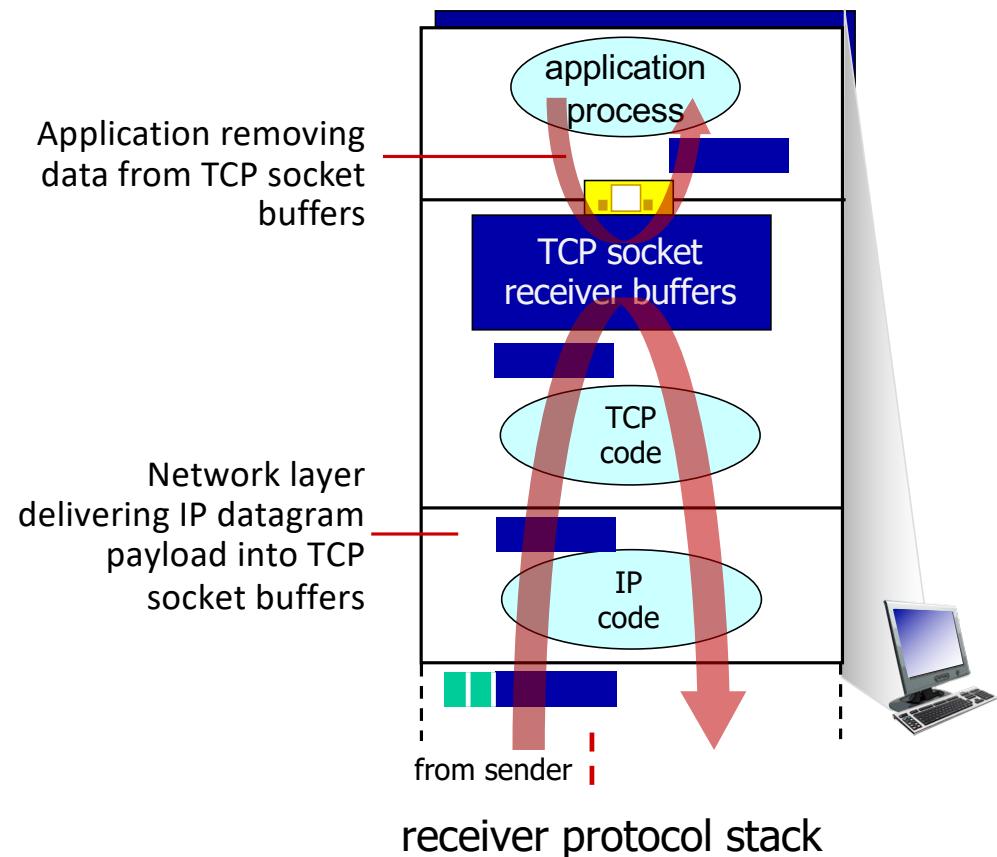
# TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



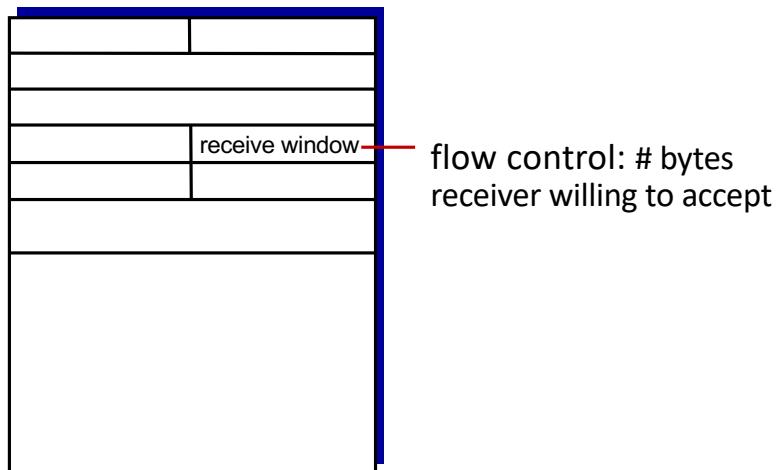
# TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

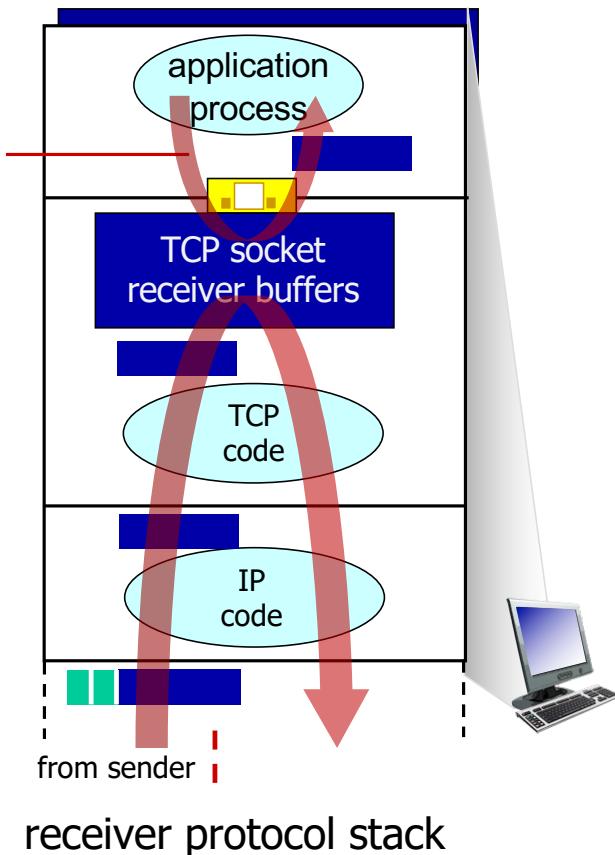


# TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



Application removing  
data from TCP socket  
buffers



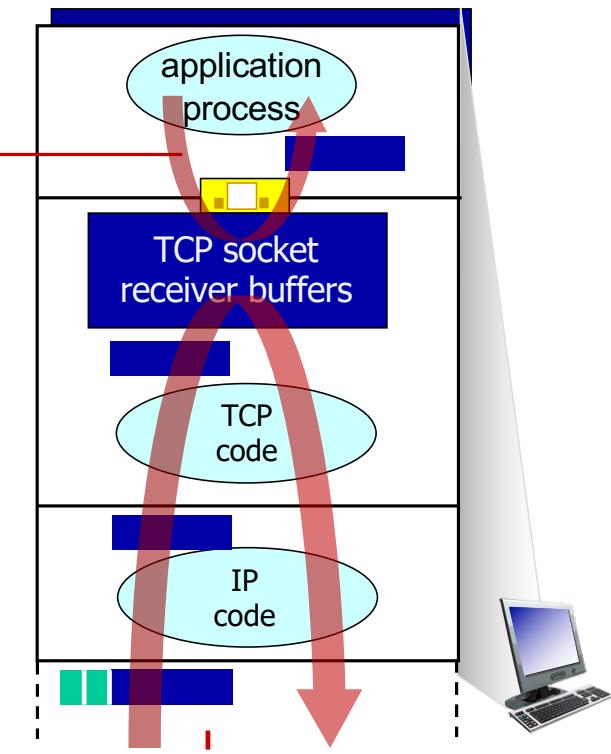
# TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

## flow control

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

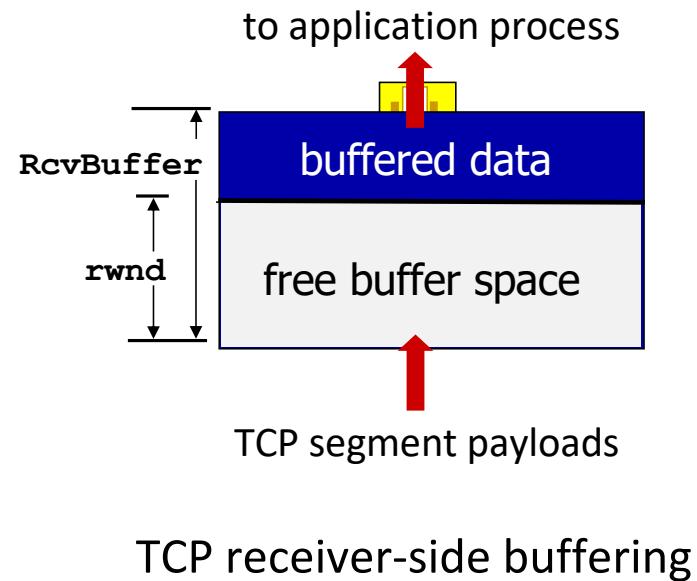
Application removing data from TCP socket buffers



receiver protocol stack

# TCP flow control

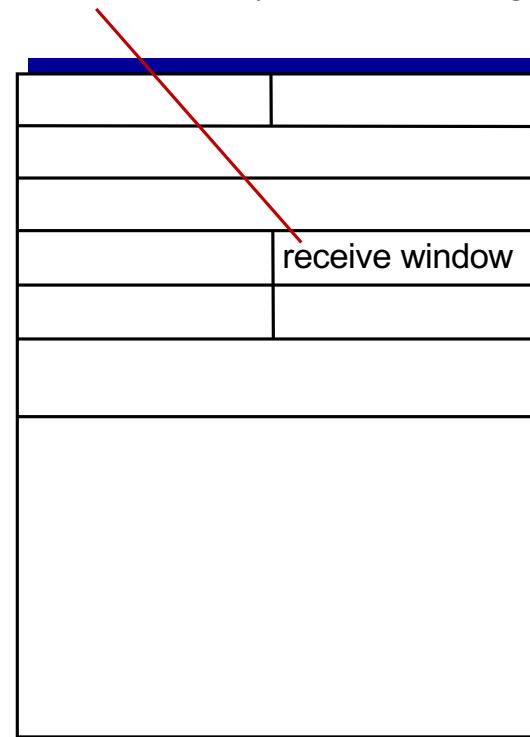
- TCP receiver “advertises” free buffer space in **rwnd** field in TCP header
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **RcvBuffer**
- sender limits amount of unACKed (“in-flight”) data to received **rwnd**
- guarantees receive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



# TCP flow control

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

flow control: # bytes receiver willing to accept



TCP segment format

# TCP flow control

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

# Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and  
demultiplexing

3.3 connectionless transport:  
UDP

3.4 principles of reliable data  
transfer

3.5 connection-oriented  
transport: TCP

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

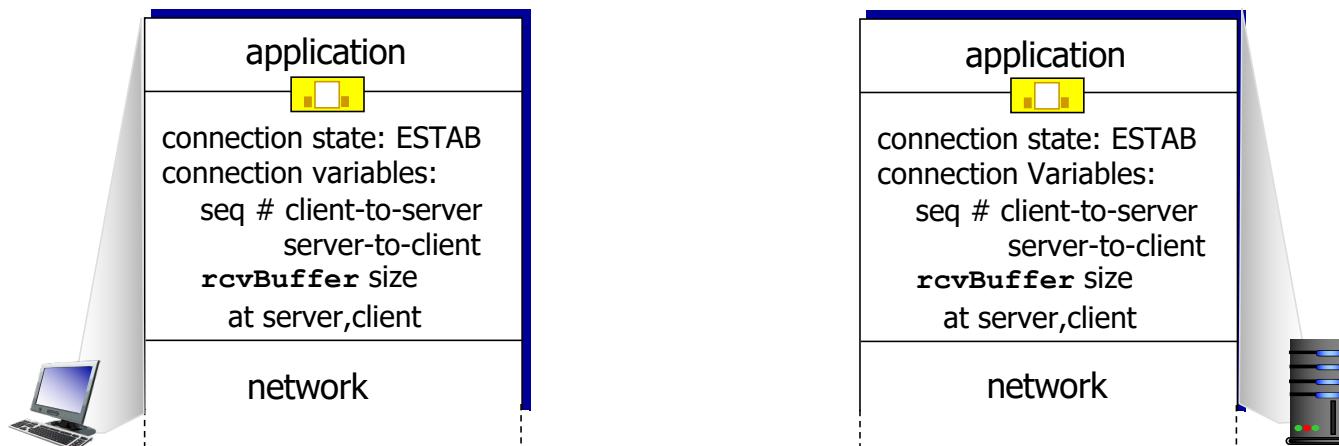
3.6 principles of congestion  
control

3.7 TCP congestion control

# 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)

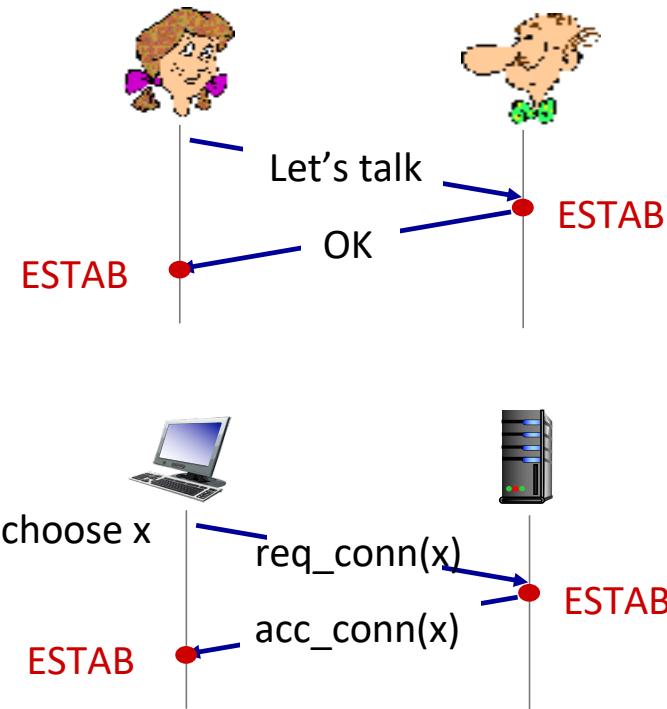


```
Socket clientSocket =  
    newSocket("hostname", "port number");
```

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

# Agreeing to establish a connection

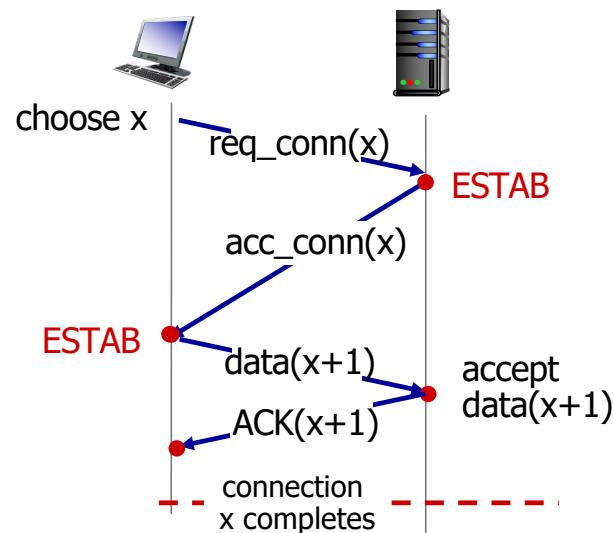
2-way handshake:



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

- variable delays
- retransmitted messages (e.g.  $\text{req\_conn}(x)$ ) due to message loss
- message reordering
- can't “see” other side

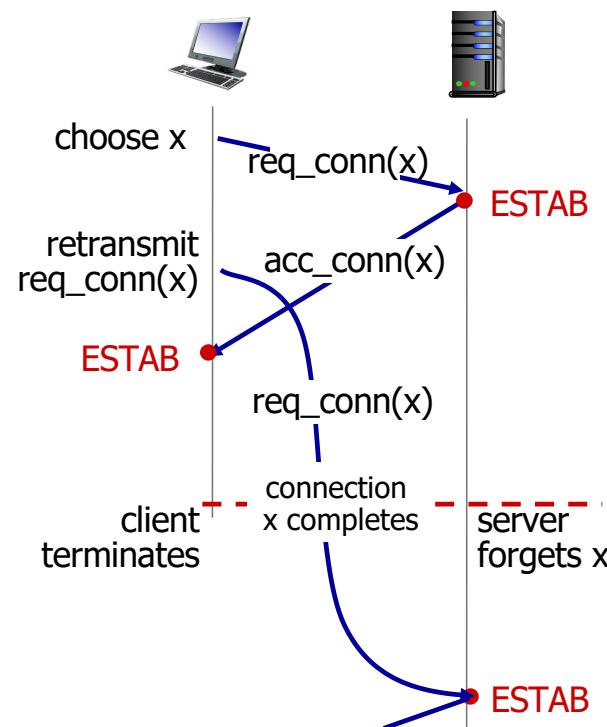
## 2-way handshake scenarios



No problem!

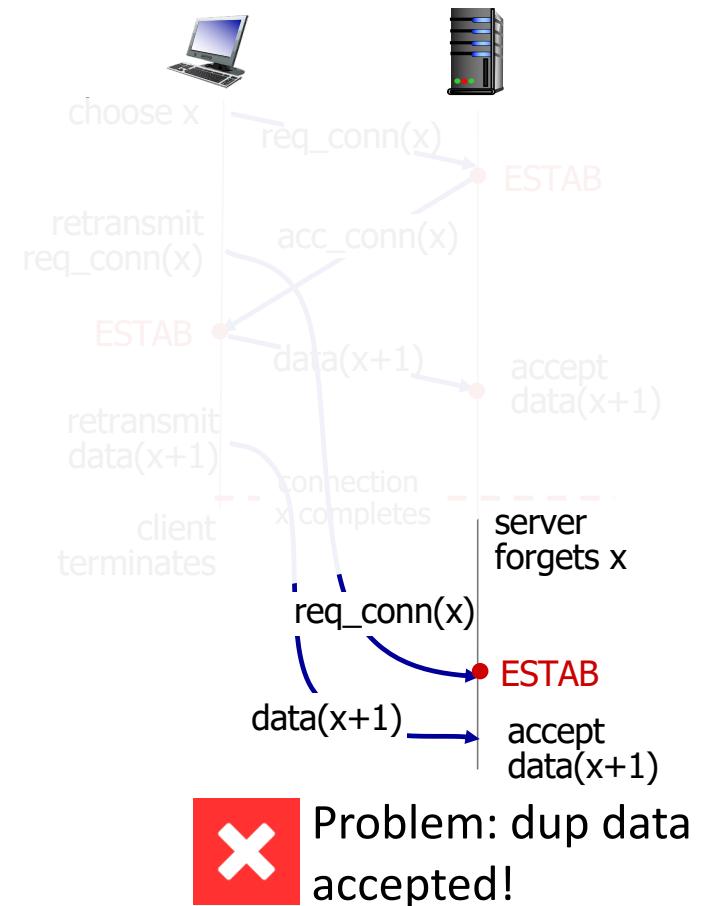


## 2-way handshake scenarios

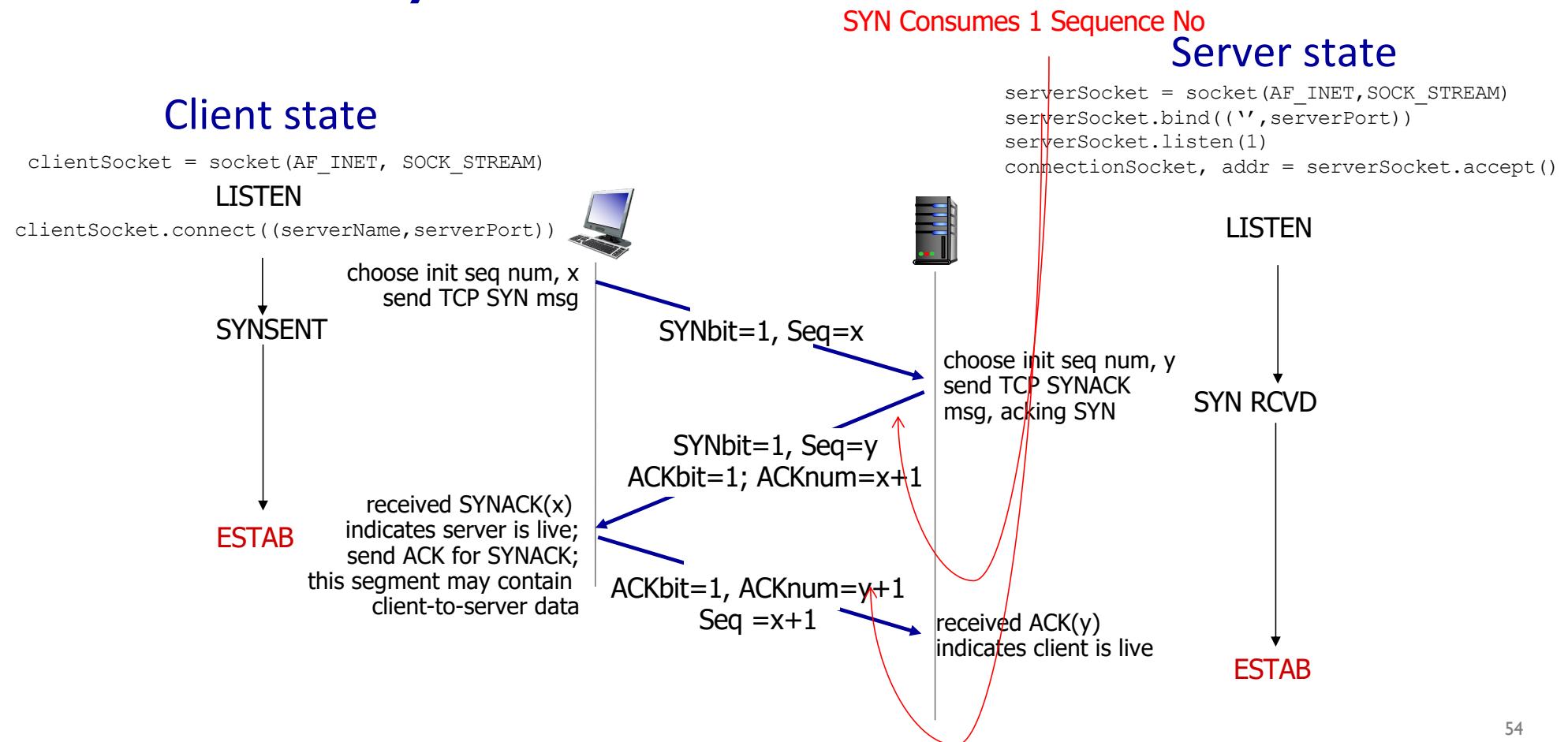


**X** Problem: half open connection! (no client)

# 2-way handshake scenarios



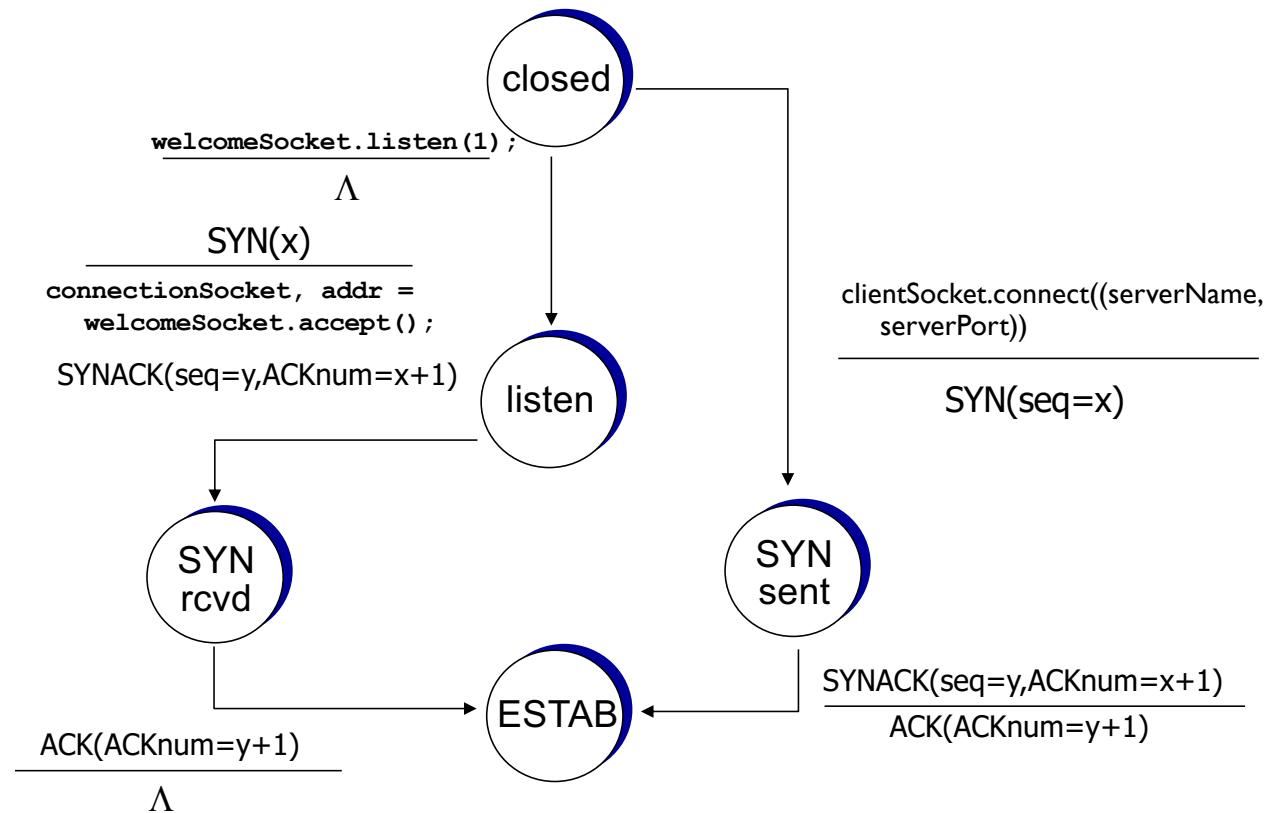
# TCP 3-way handshake



# A human 3-way handshake protocol



# TCP 3-way handshake: Partial state machine



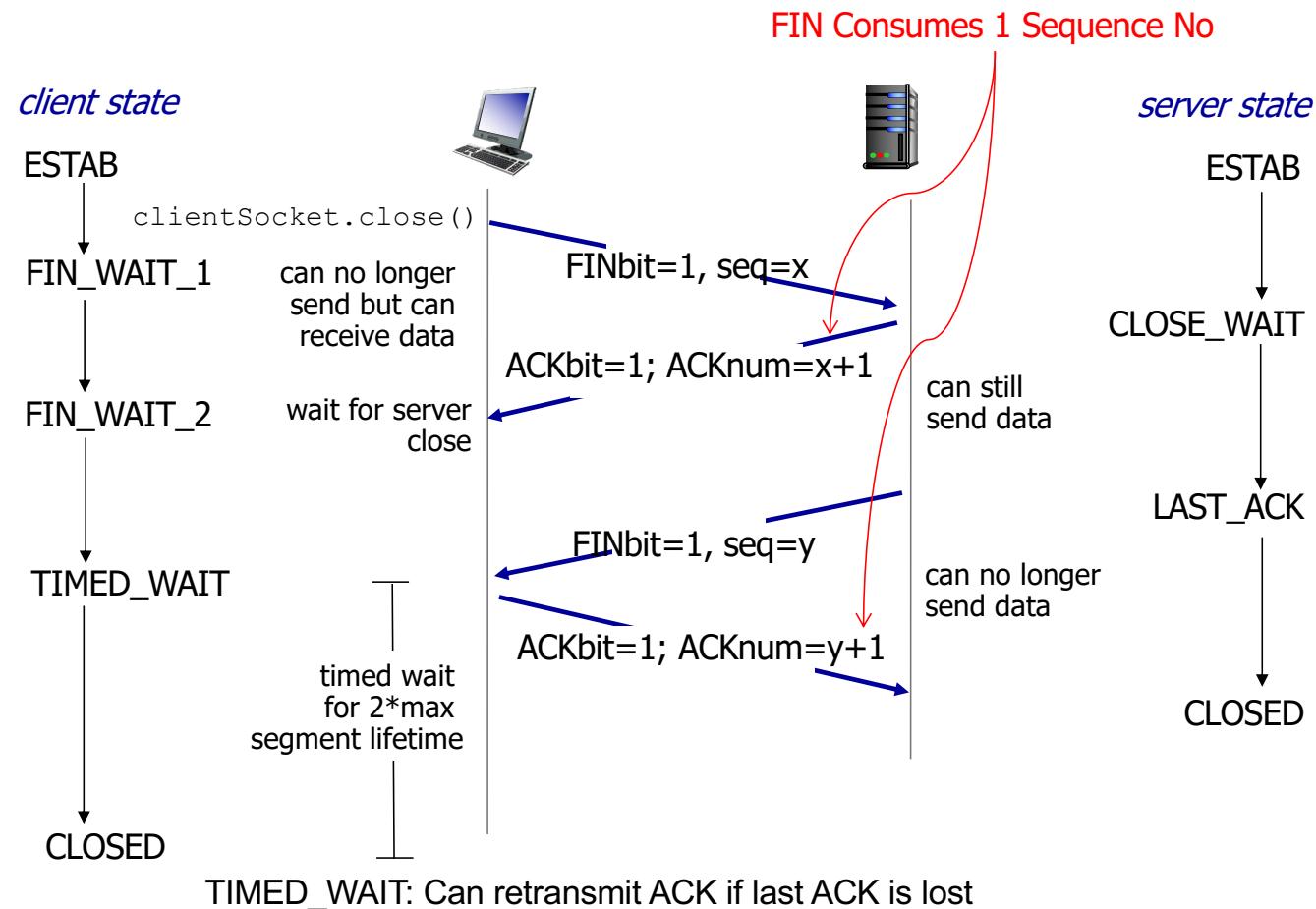
# What if the SYN Packet Gets Lost?

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

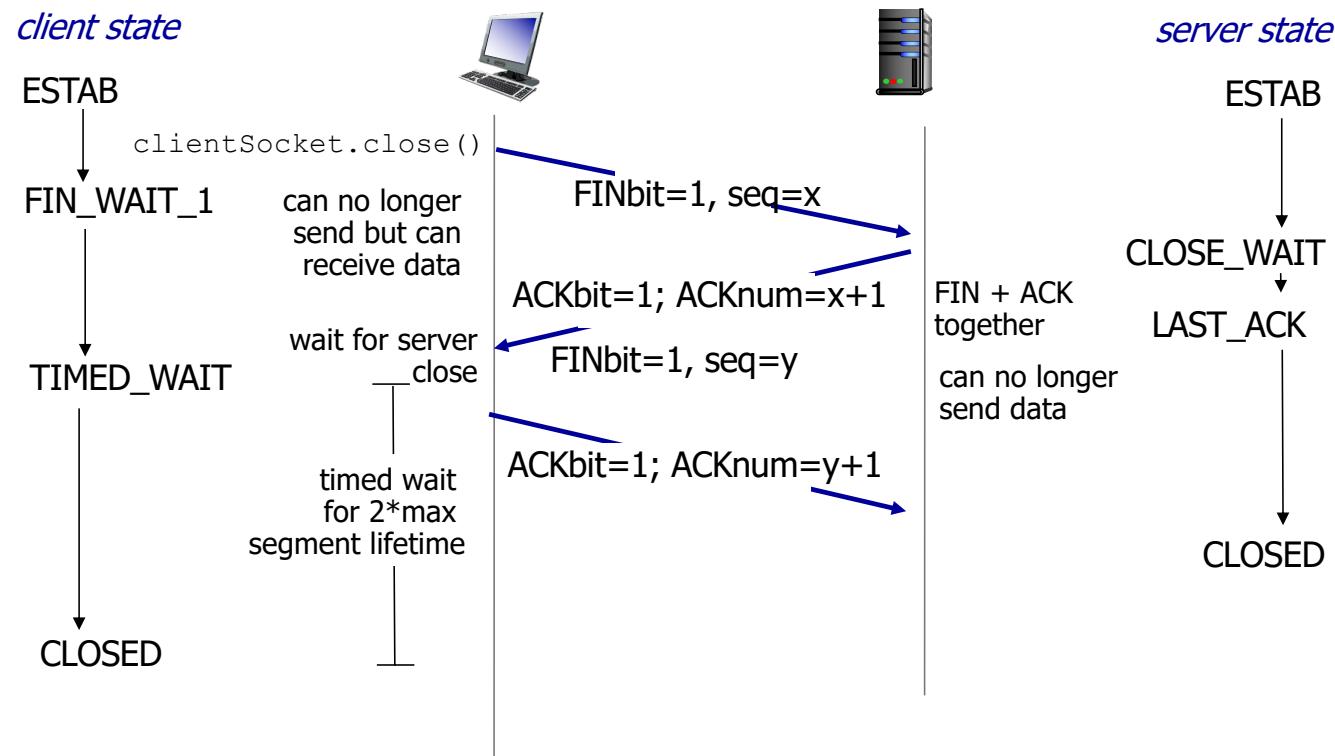
# TCP: closing a connection

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

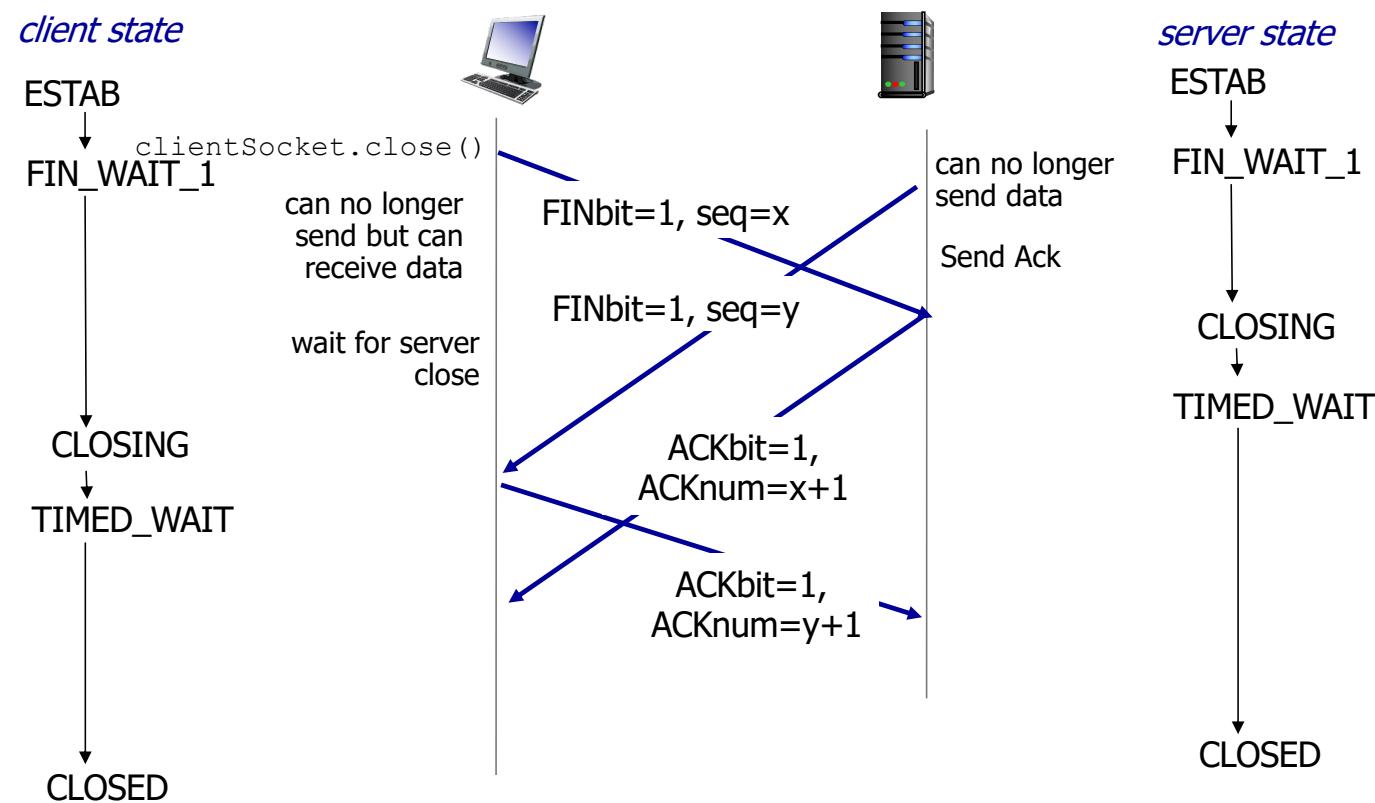
# Normal Termination, One at a Time



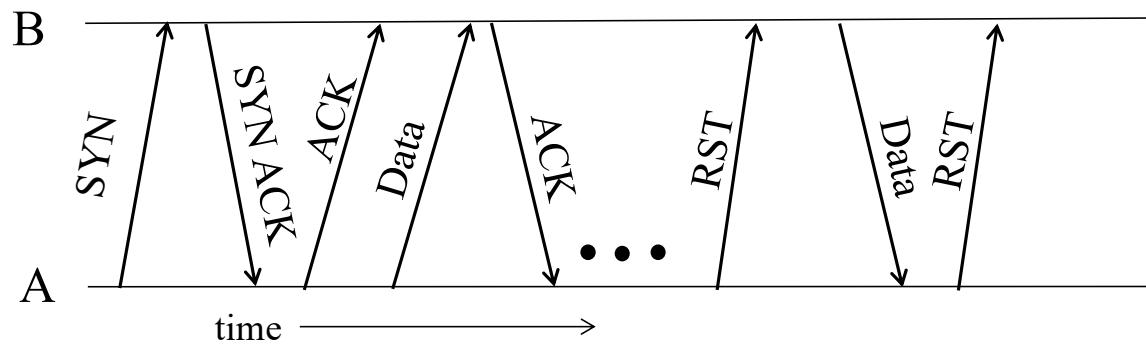
# Normal Termination, Both Together



# Simultaneous Closure



# Abrupt Termination



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

# TCP SYN Attack (SYN flooding)

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

# TCP SYN Cookie

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

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

## Quiz: TCP Connection Management?



Assume that one end of a TCP connection selects an initial sequence number 120. The first TCP segment containing data sent by this end point will have a sequence number of \_\_\_\_\_

- A. 120
- B. 121
- C. 122
- D. 128
- E. 0

## Quiz: TCP Connection Management



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

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

# Transport Layer: Outline

3.1 transport-layer services

3.2 multiplexing and  
demultiplexing

3.3 connectionless transport:  
UDP

3.4 principles of reliable data  
transfer

3.5 connection-oriented  
transport: TCP

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

3.6 principles of congestion  
control

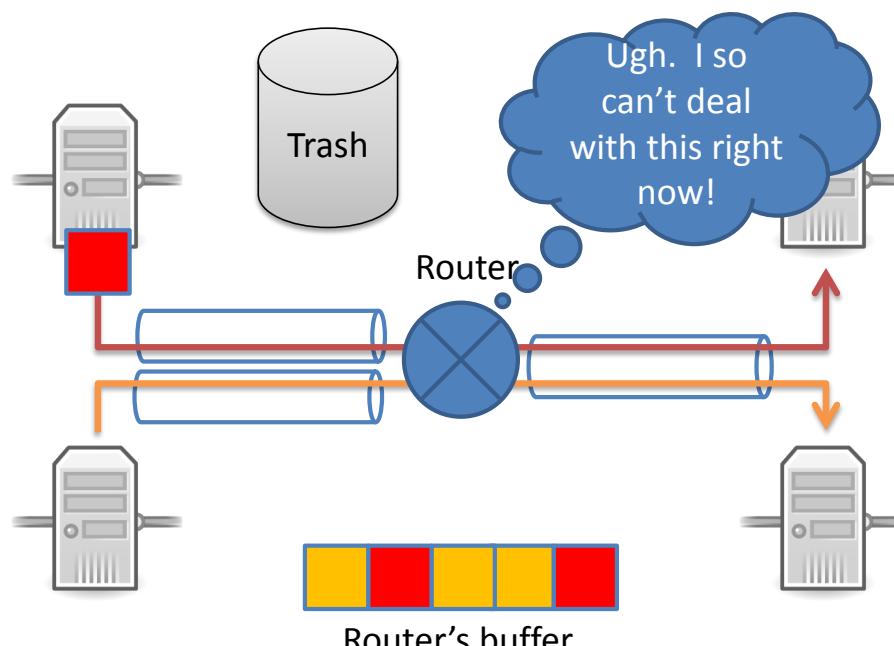
3.7 TCP congestion control

# Principles of congestion control

## *congestion:*

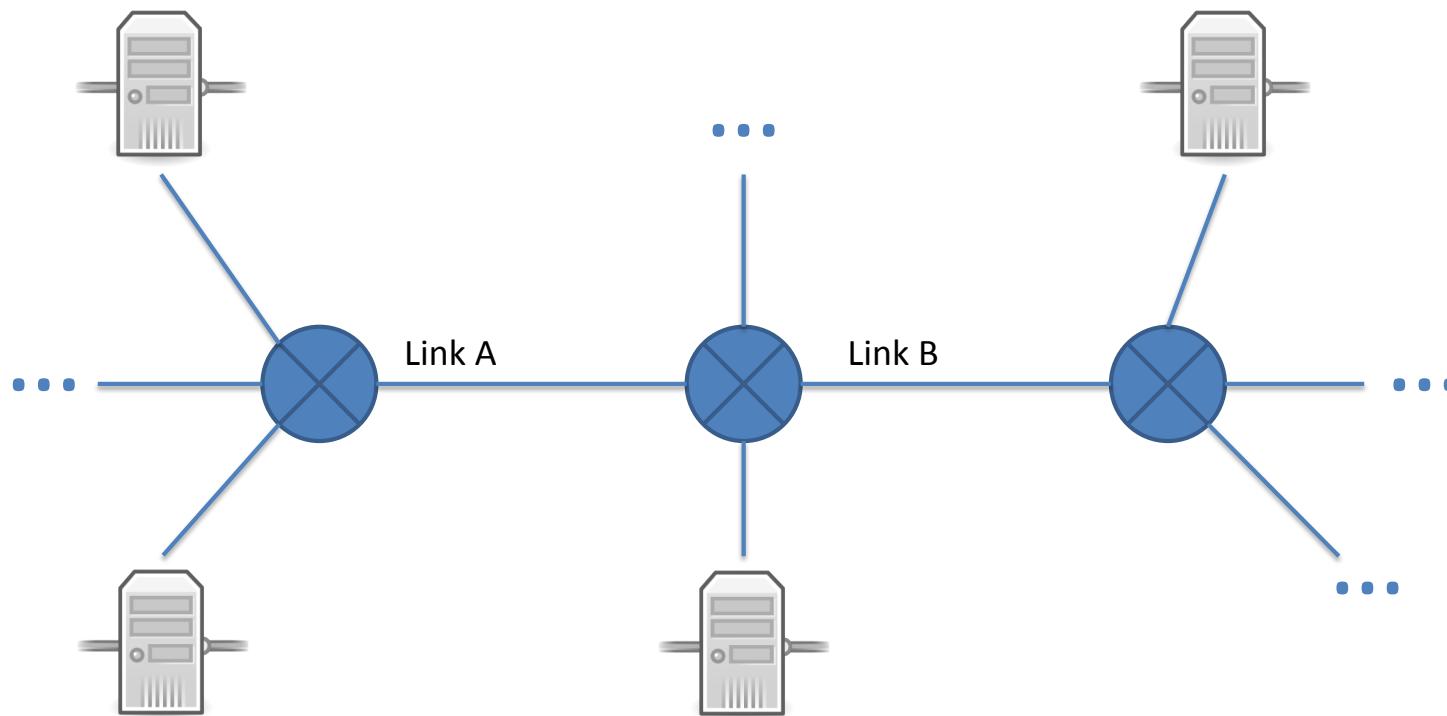
- ❖ informally: “too many sources sending too much data too fast for *network* to handle”
- ❖ different from flow control!
- ❖ manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- ❖ a top-10 problem!

# Congestion

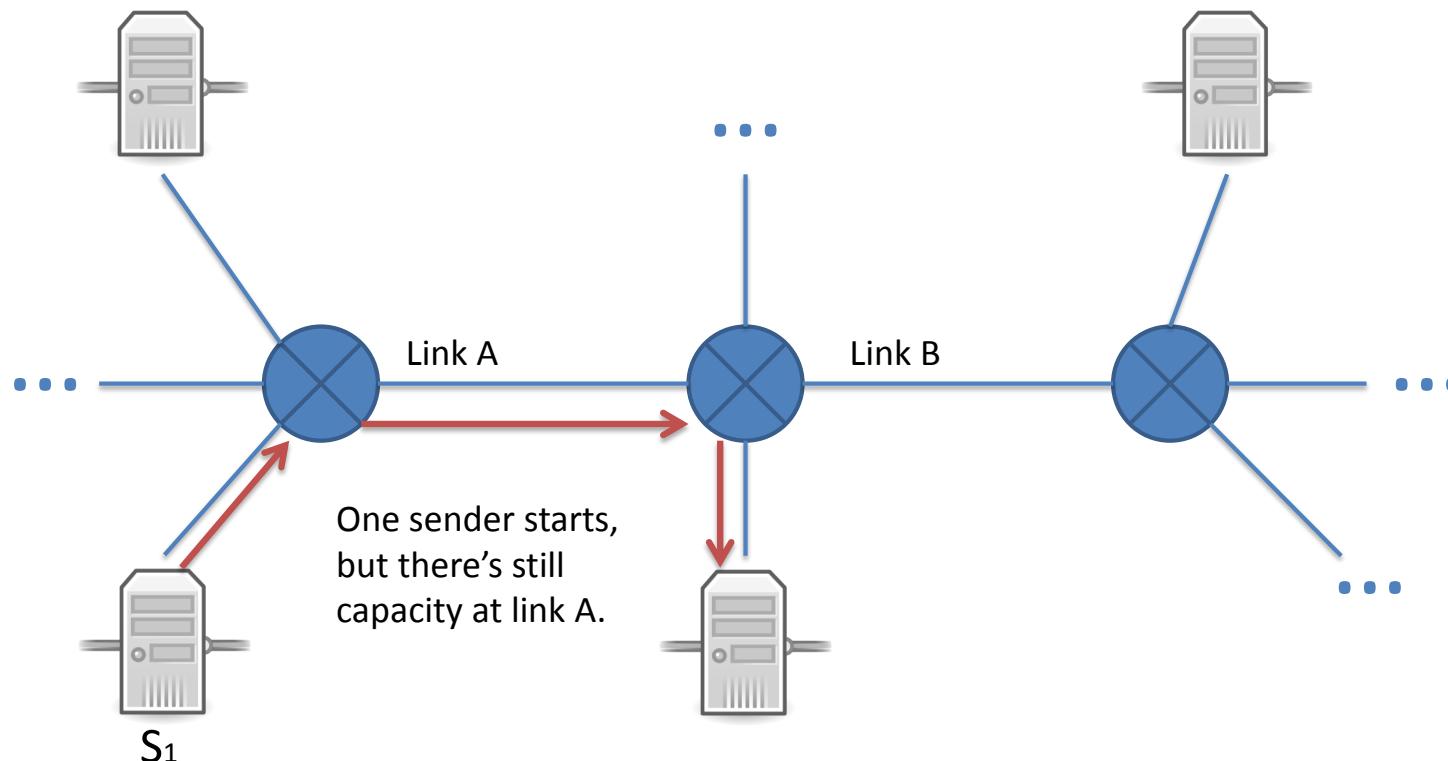


Incoming rate is faster than  
outgoing link can support.

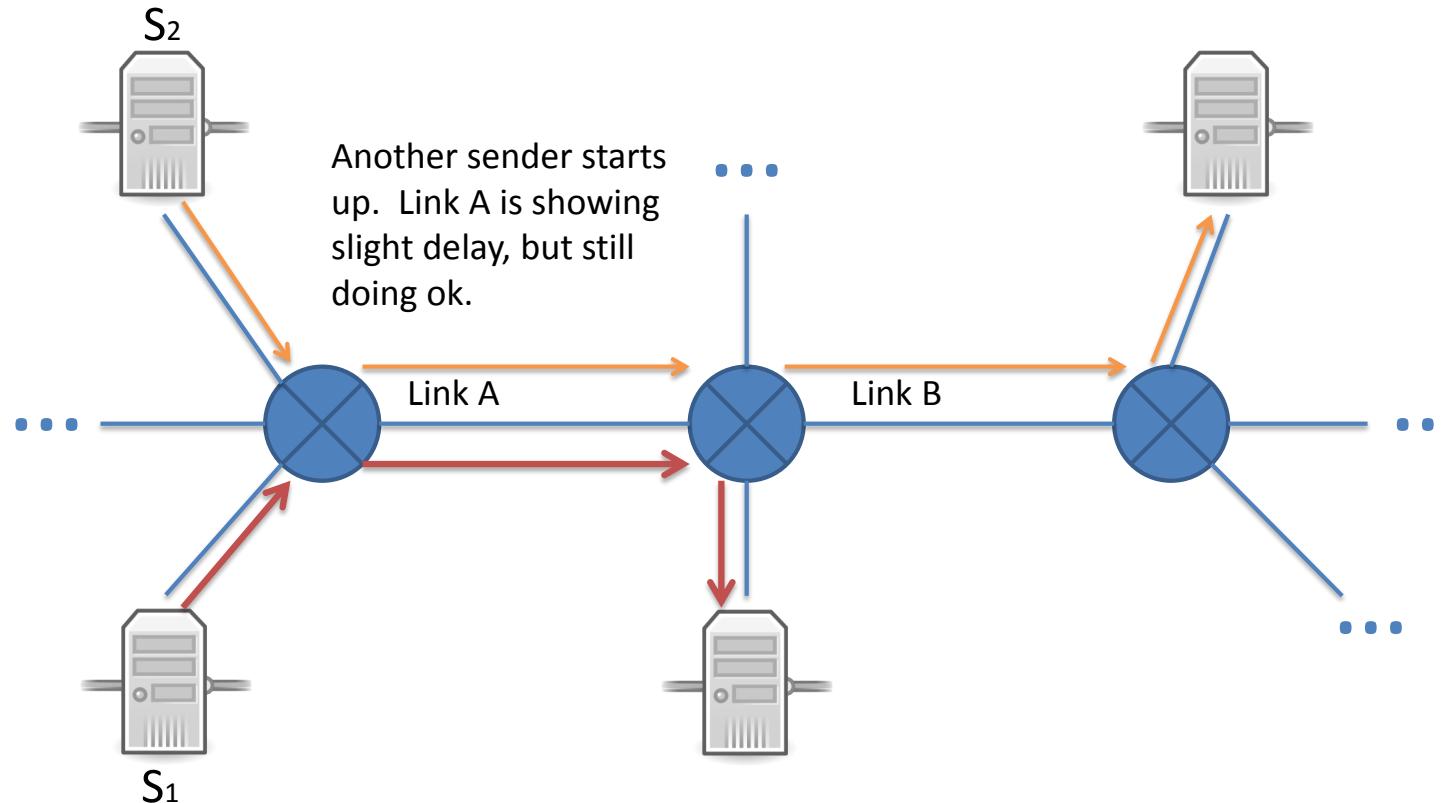
# Congestion Collapse



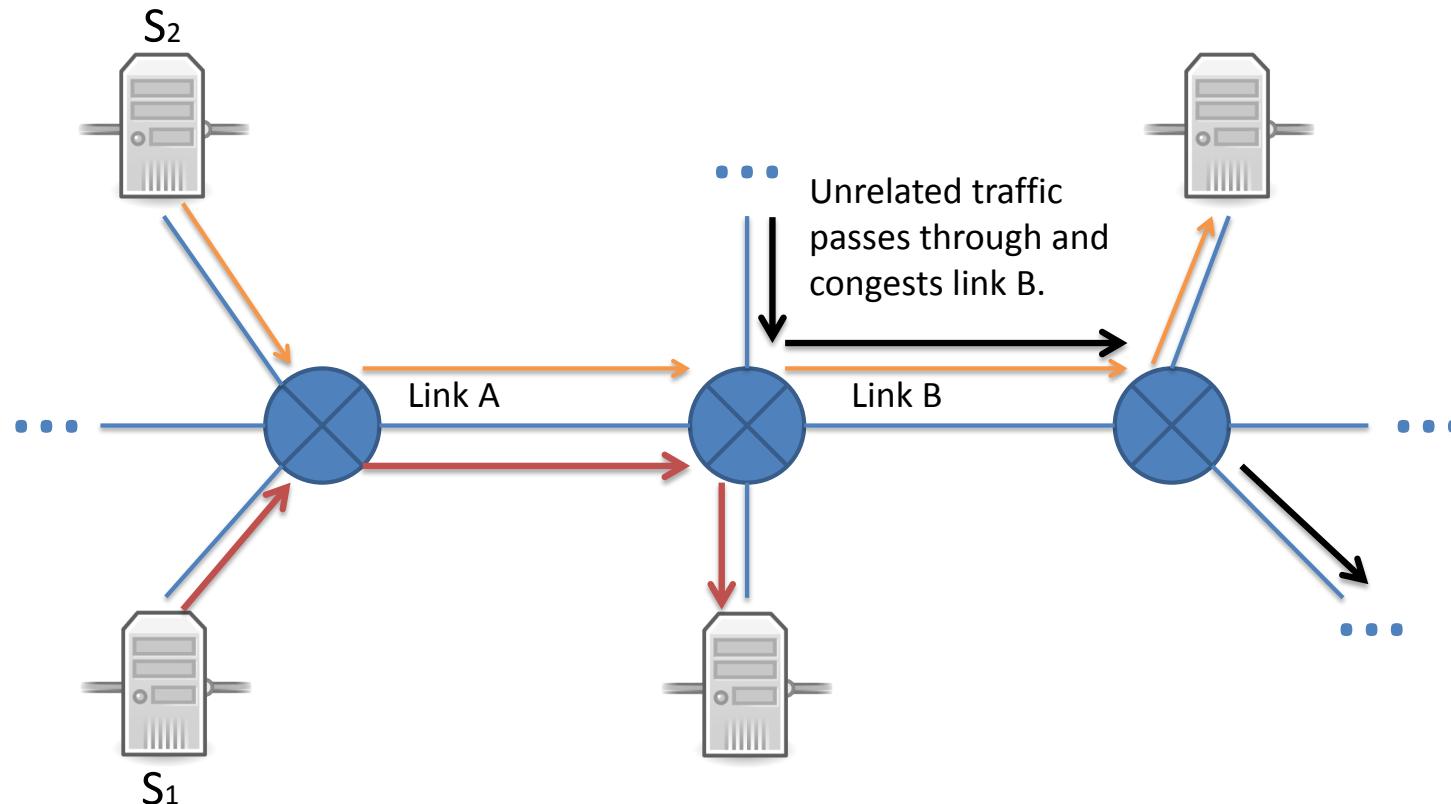
# Congestion Collapse



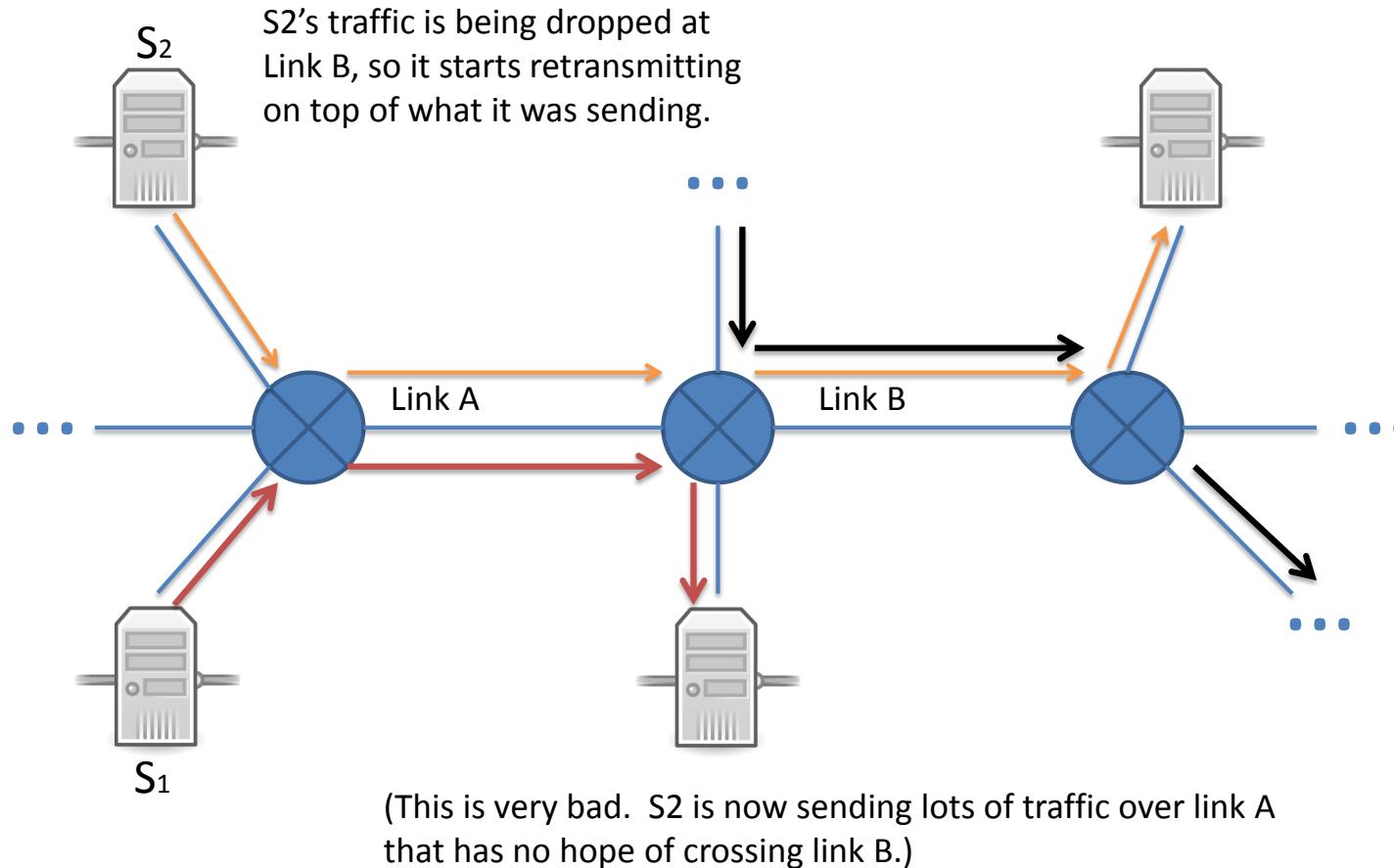
# Congestion Collapse



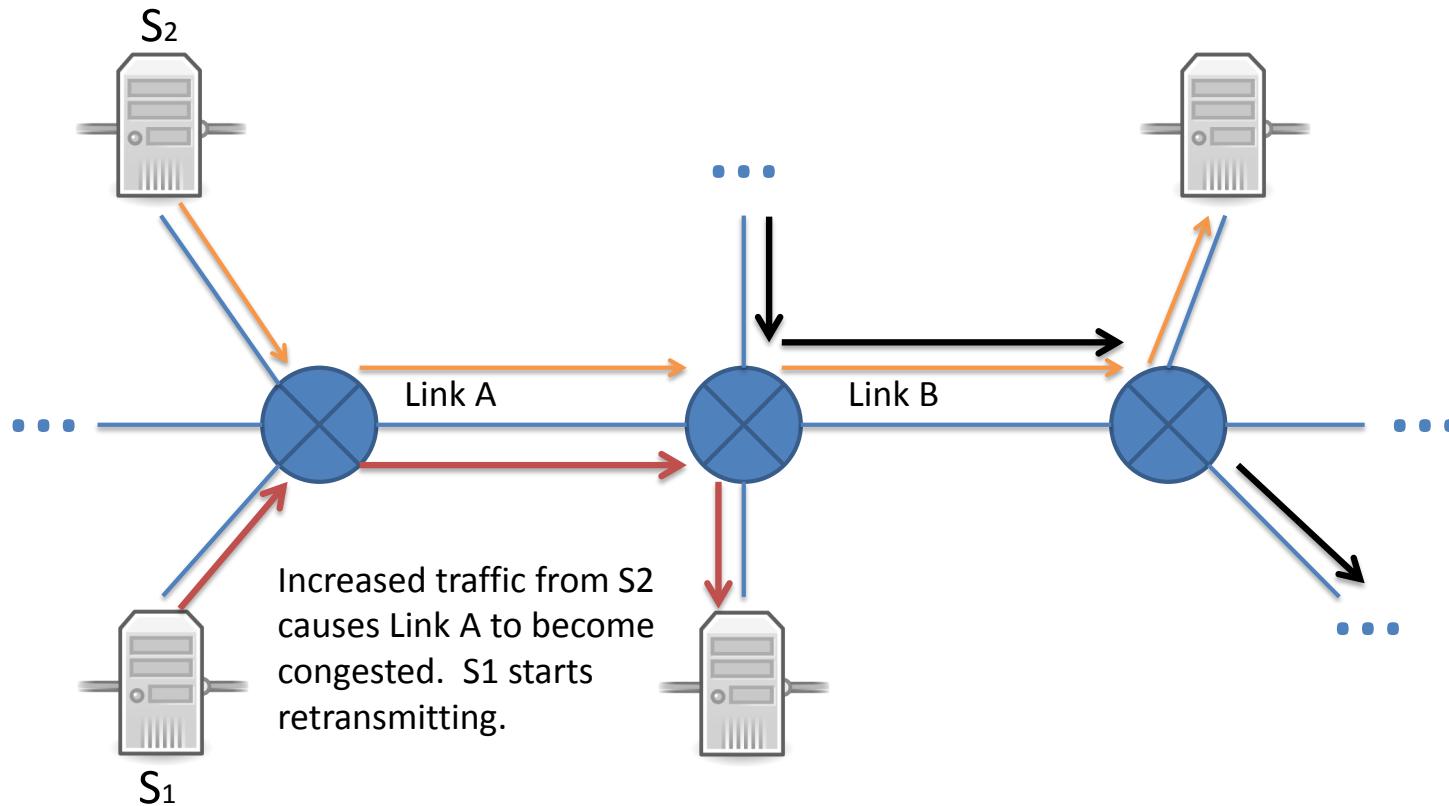
# Congestion Collapse



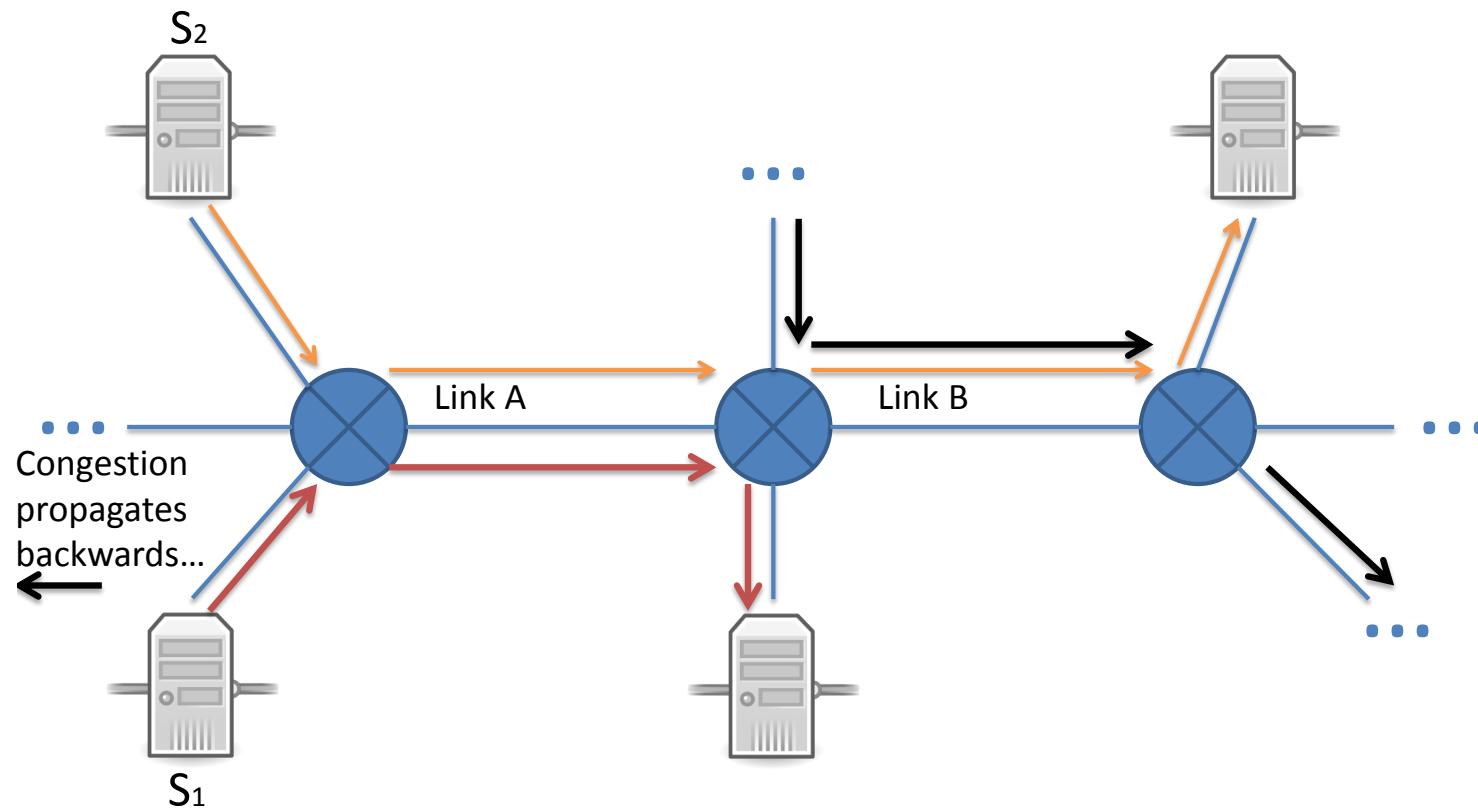
# Congestion Collapse



# Congestion Collapse



# Congestion Collapse



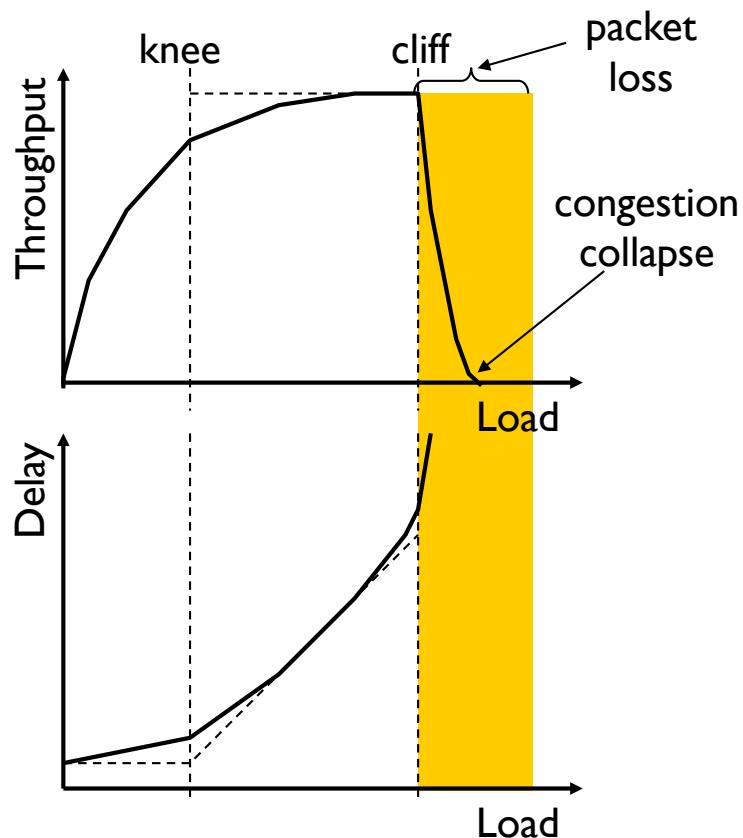
# Without congestion control

*congestion:*

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

# Cost of Congestion

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



# Congestion Collapse

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

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

# Approaches towards congestion control

two broad approaches towards congestion control:

## end-end congestion control:

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

## network-assisted congestion control:

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

# Transport Layer: Outline

3.1 transport-layer services

3.2 multiplexing and  
demultiplexing

3.3 connectionless transport:  
UDP

3.4 principles of reliable data  
transfer

3.5 connection-oriented  
transport: TCP

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

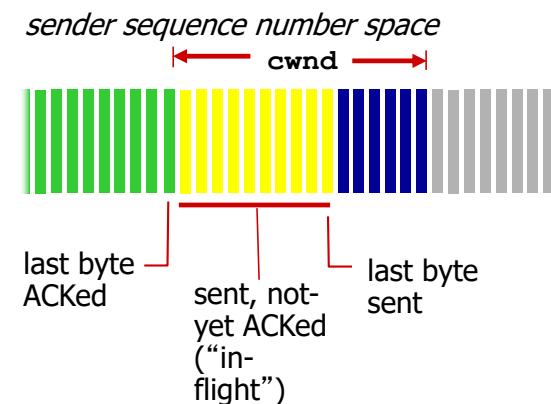
3.6 principles of congestion  
control

3.7 TCP congestion control

# TCP's Approach in a Nutshell

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

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



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

# All These Windows...

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

# CWND

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

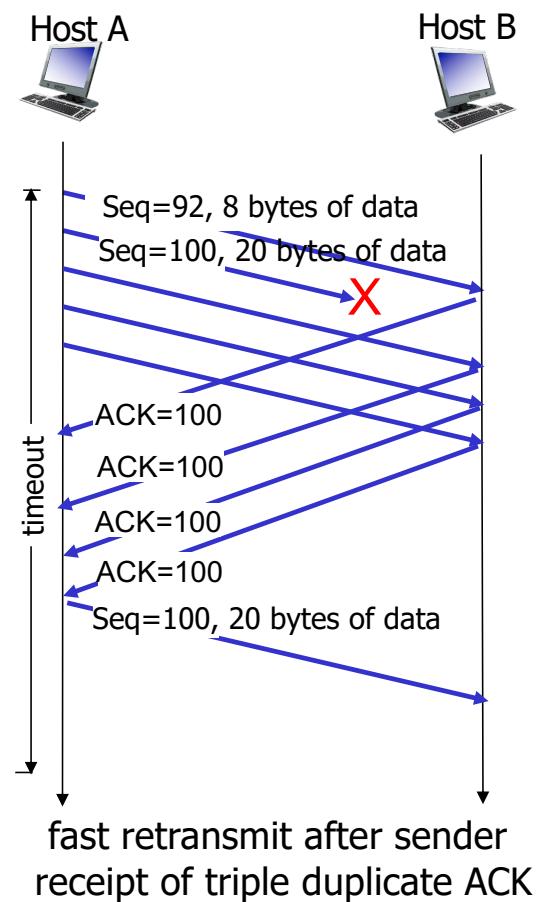
## Two Basic Questions

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

# Detection Congestion: Infer Loss

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

## RECAP: TCP fast retransmit (dup acks)

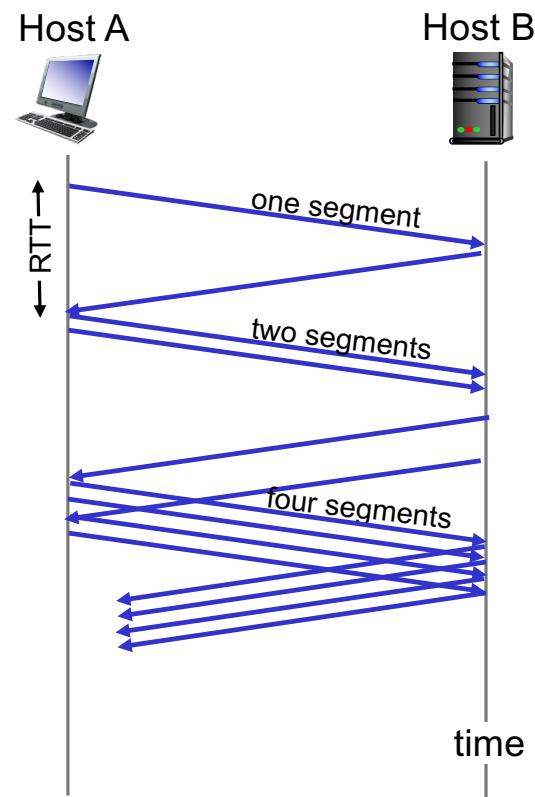


# Rate Adjustment

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

## TCP Slow Start (Bandwidth discovery)

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



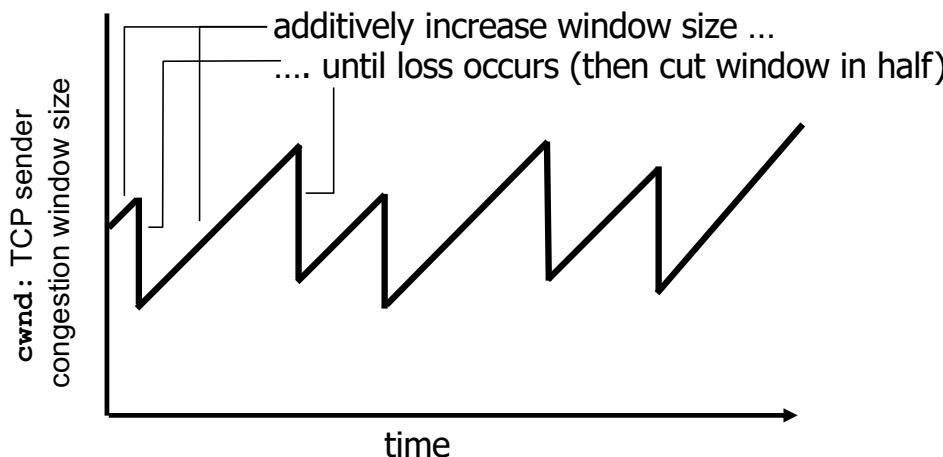
# Adjusting to Varying Bandwidth

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

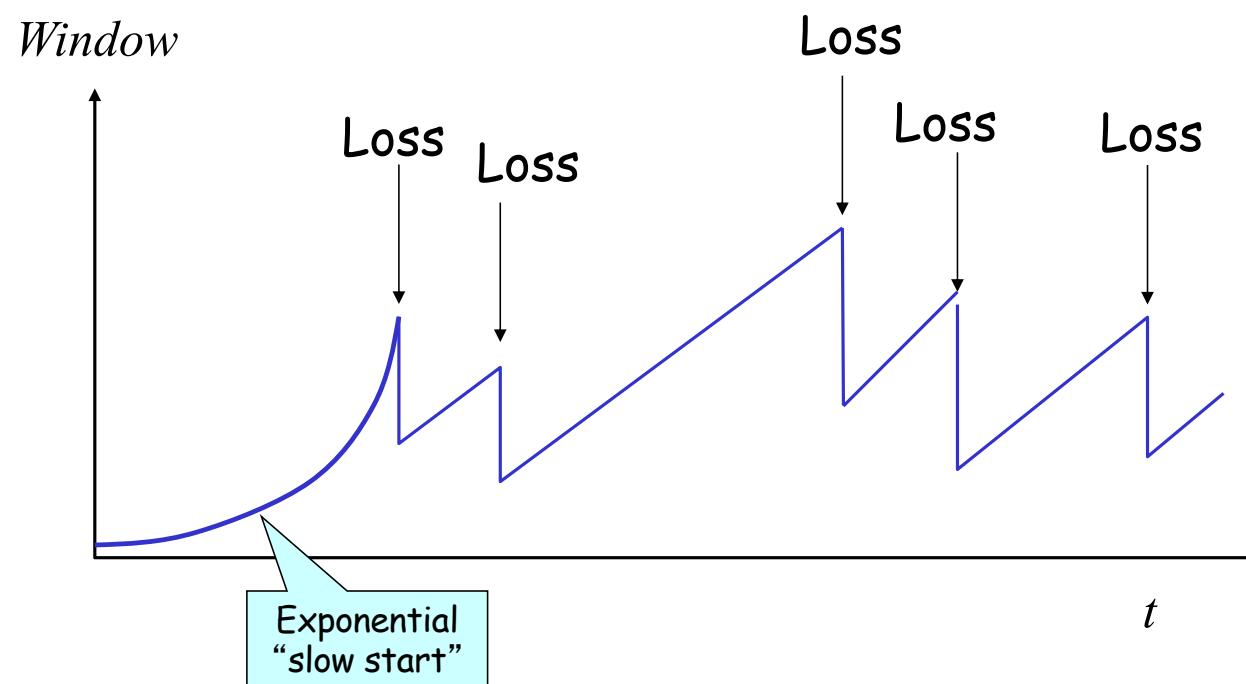
# AIMD

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

AIMD saw tooth behavior: probing for bandwidth



# Leads to the TCP “Sawtooth”



## Slow-Start vs. AIMD

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

# Implementation

## ❖ State at sender

- CWND (initialized to a small constant)
  - the slides use multiple of MSS
- ssthresh (initialized to a large constant)
- [Also dupACKcount and timer, as before]

## ❖ Events

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

## Event: ACK (new data)

- ❖ If  $CWND < ssthresh$ 
  - $CWND += 1$

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

## Event: ACK (new data)

- ❖ If  $CWND < ssthresh$ 
  - $CWND += 1$



*Slow start phase*

- ❖ Else
  - $CWND = CWND + 1/CWND$



*“Congestion  
Avoidance” phase  
(additive increase)*

- Hence after one RTT (All ACKs with no drops):  
 $CWND = CWND + 1$

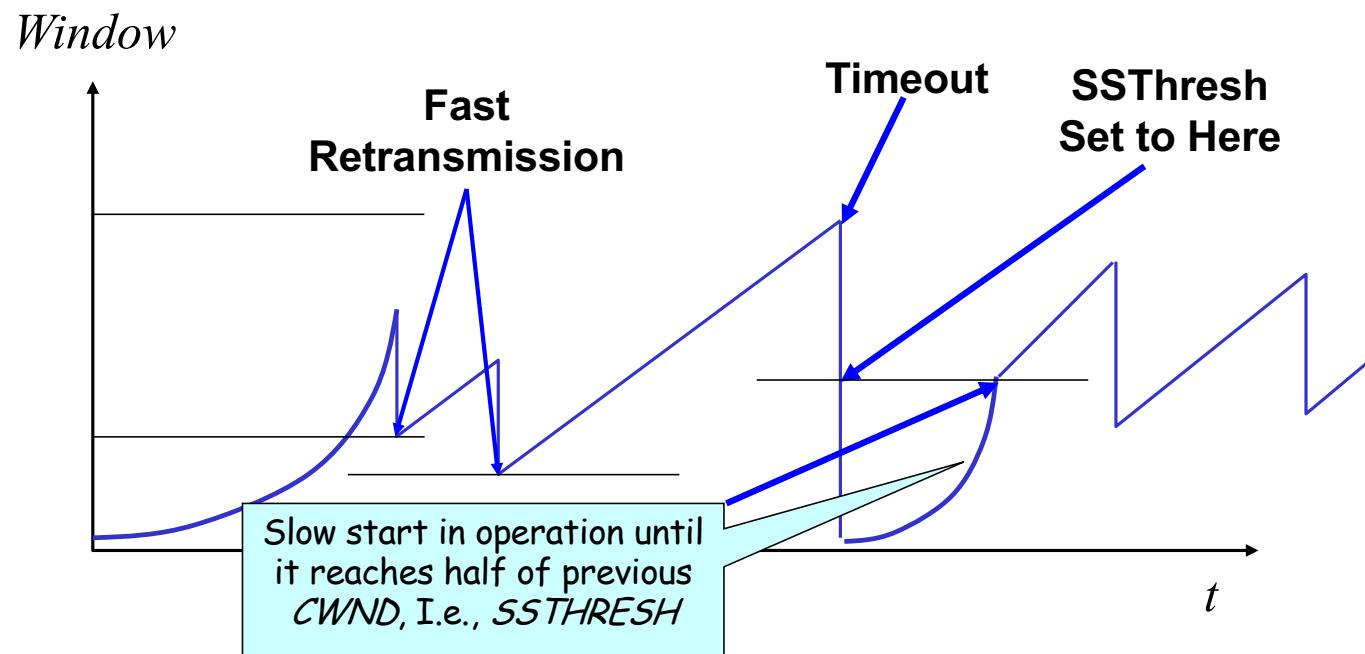
## Event: dupACK

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

## Event: TimeOut

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

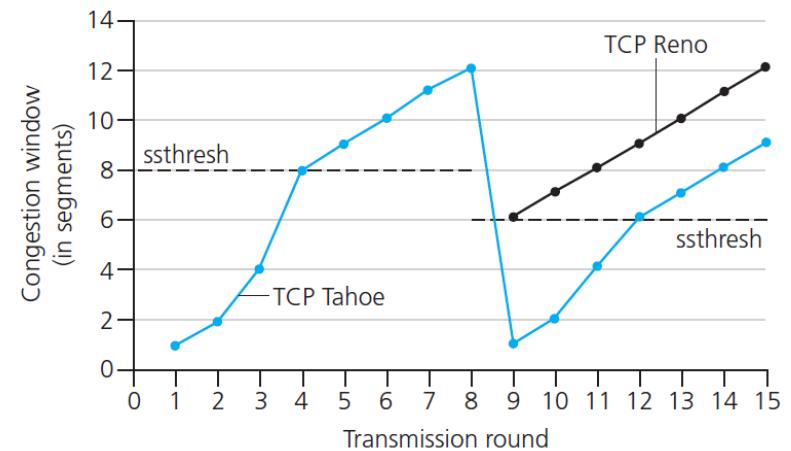
# Example



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

# TCP Flavours

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

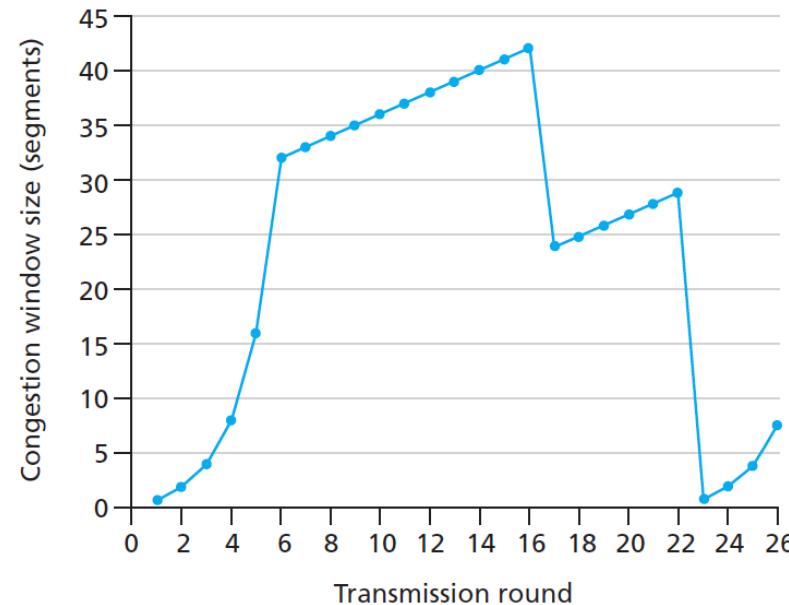




## Quiz: TCP Congestion Control?

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

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



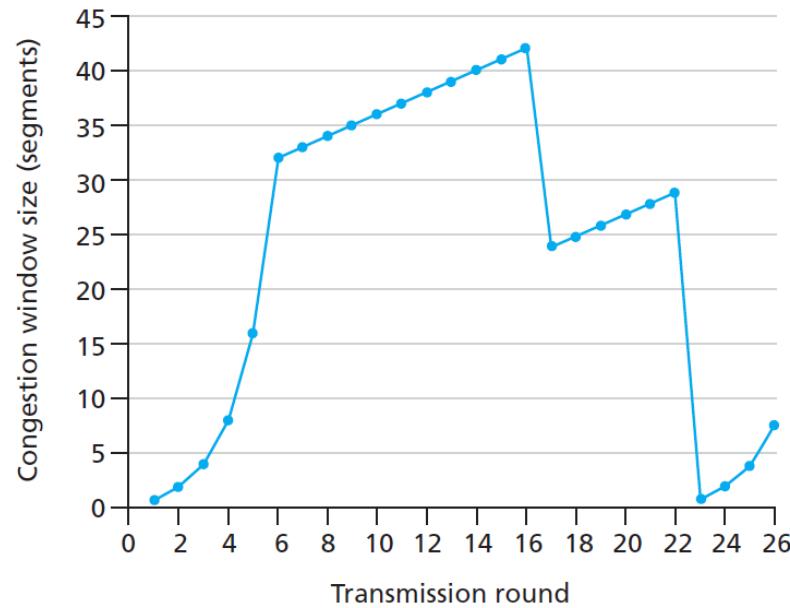
Note: the transition at round 17 is not entirely accurate, the window should reduce to 21 (currently 24)



## Quiz: TCP Congestion Control?

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

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



Note: the transition at round 17 is not entirely accurate, the window should reduce to 21 (currently 24)

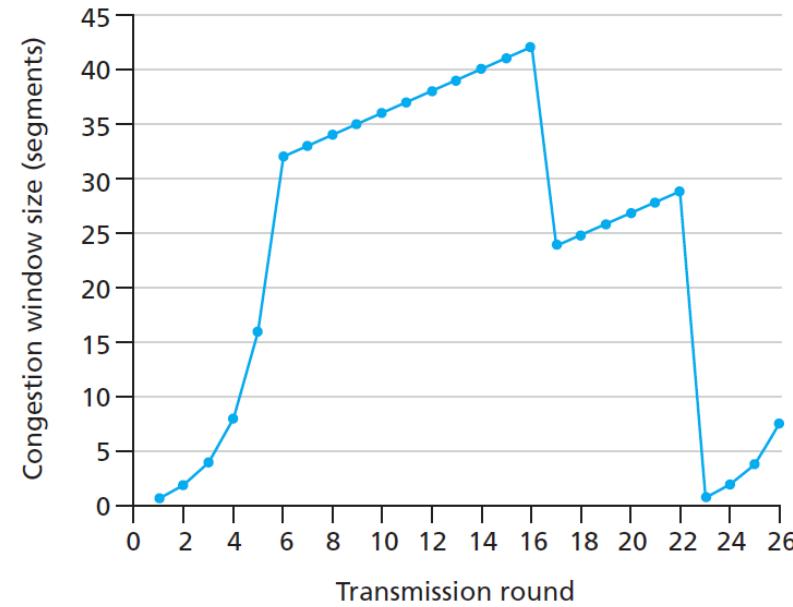


## Quiz: TCP Congestion Control

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

- A. Triple Dup Ack
- B. Timeout

Note: the transition at round 17 is not entirely accurate, the window should reduce to 21 (currently 24)

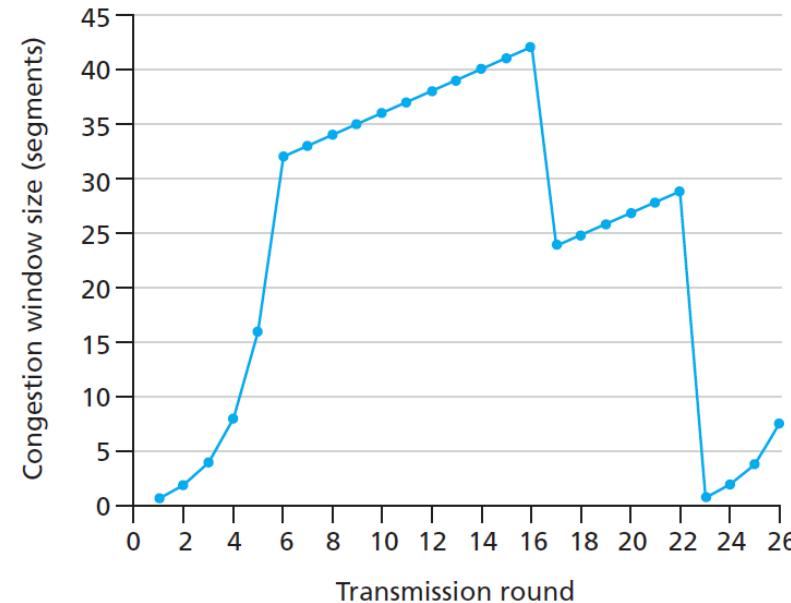




## Quiz: TCP Congestion Control

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

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



Note: the transition at round 17 is not entirely accurate, the window should reduce to 21 (currently 24)

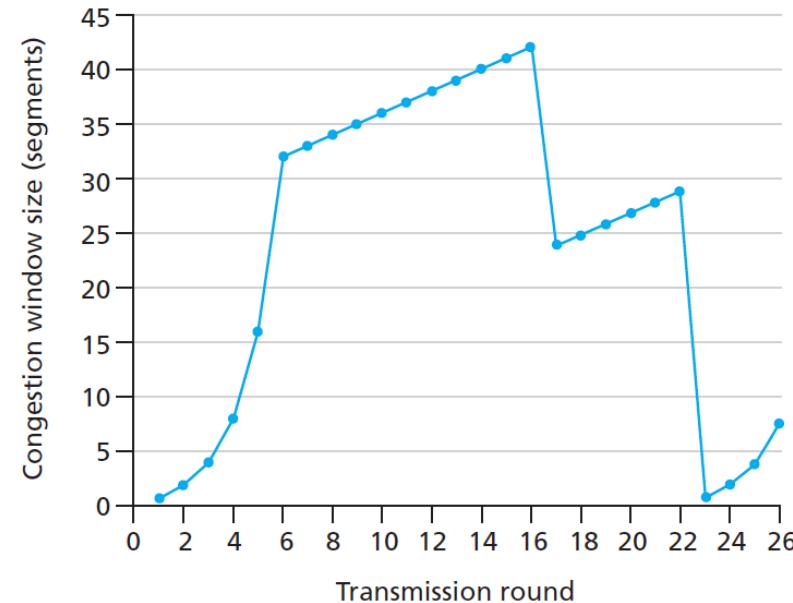


## Quiz: TCP Congestion Control

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

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

Note: the transition at round 17 is not entirely accurate, the window should reduce to 21 (currently 24)



## Quiz: TCP Reliability



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

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

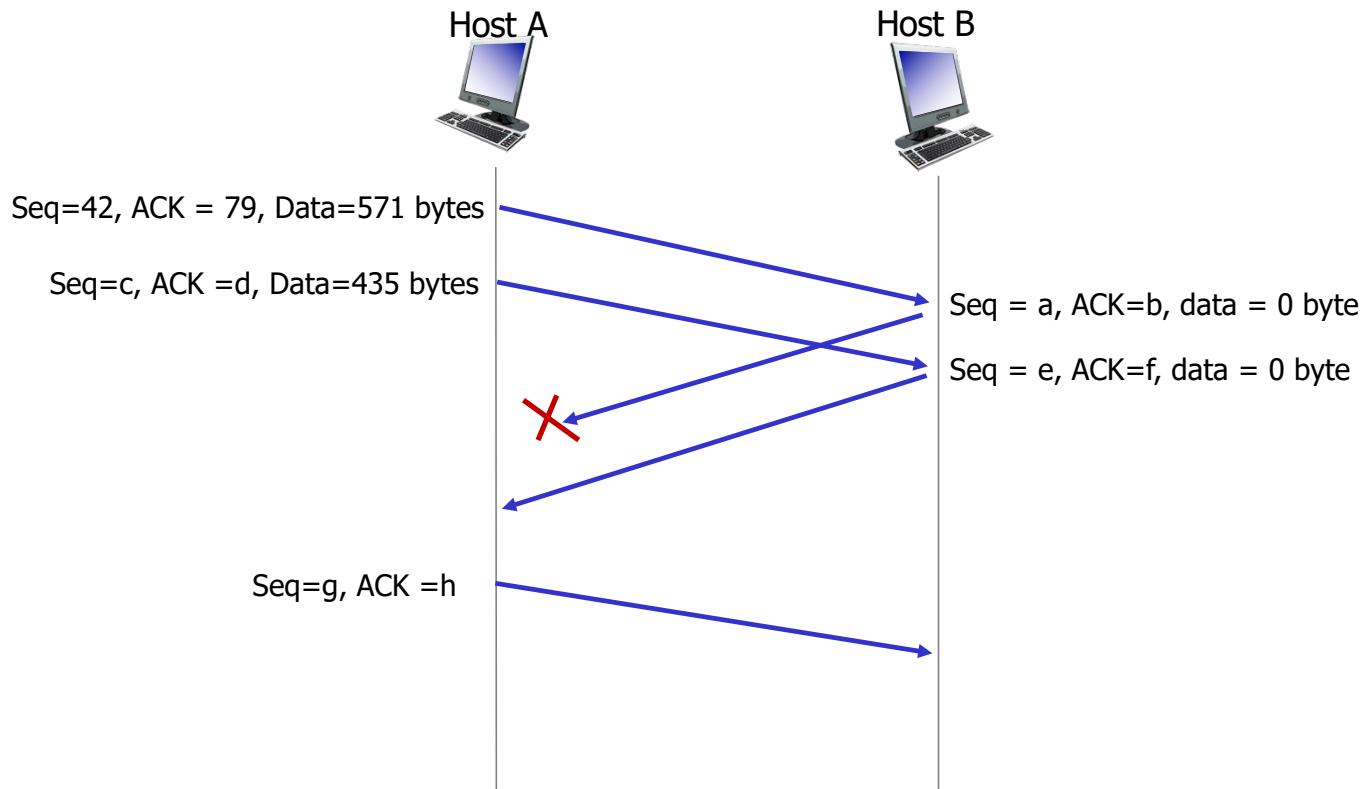
## Quiz: TCP Timeout



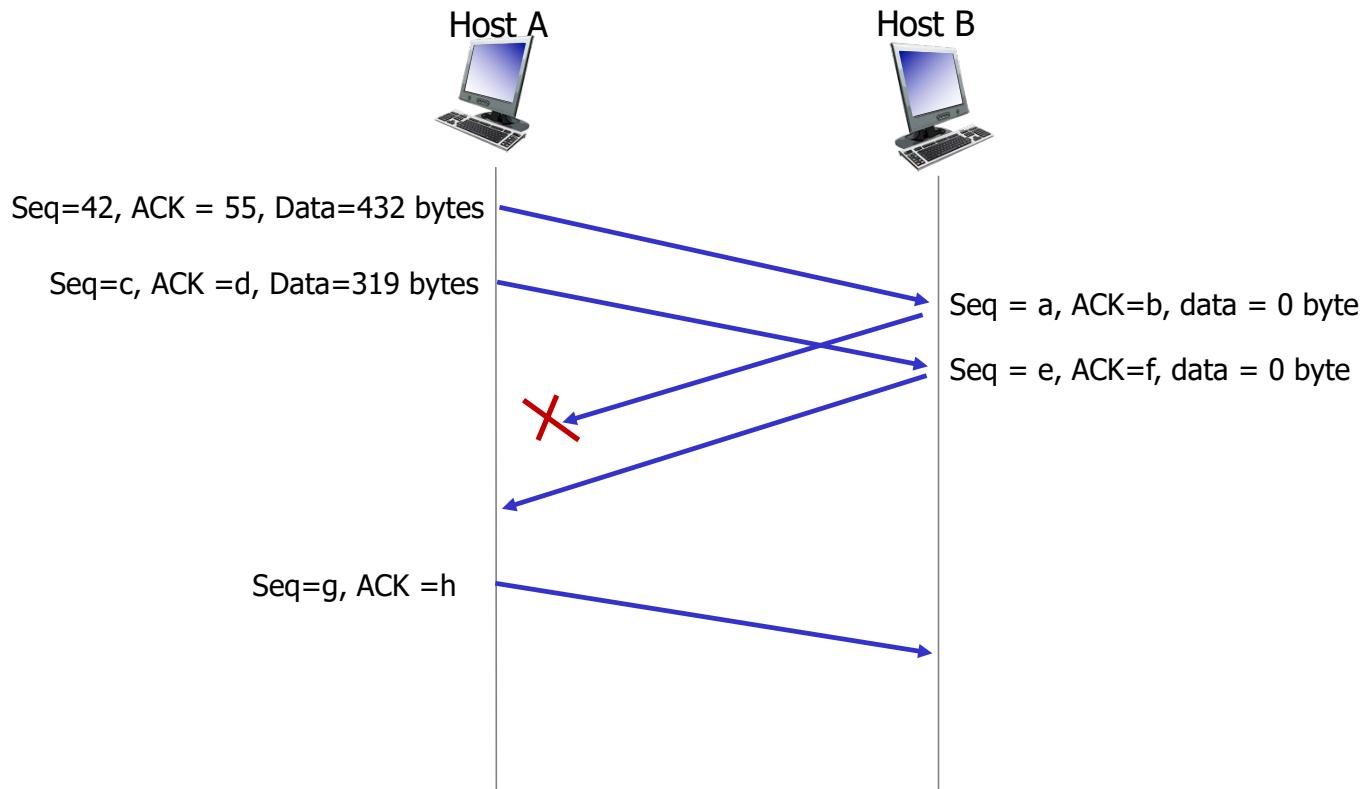
A TCP Sender maintains an EstimatedRTT of 100ms. Suppose the next SampleRTT is 108. Which of the following is true about the sender?

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

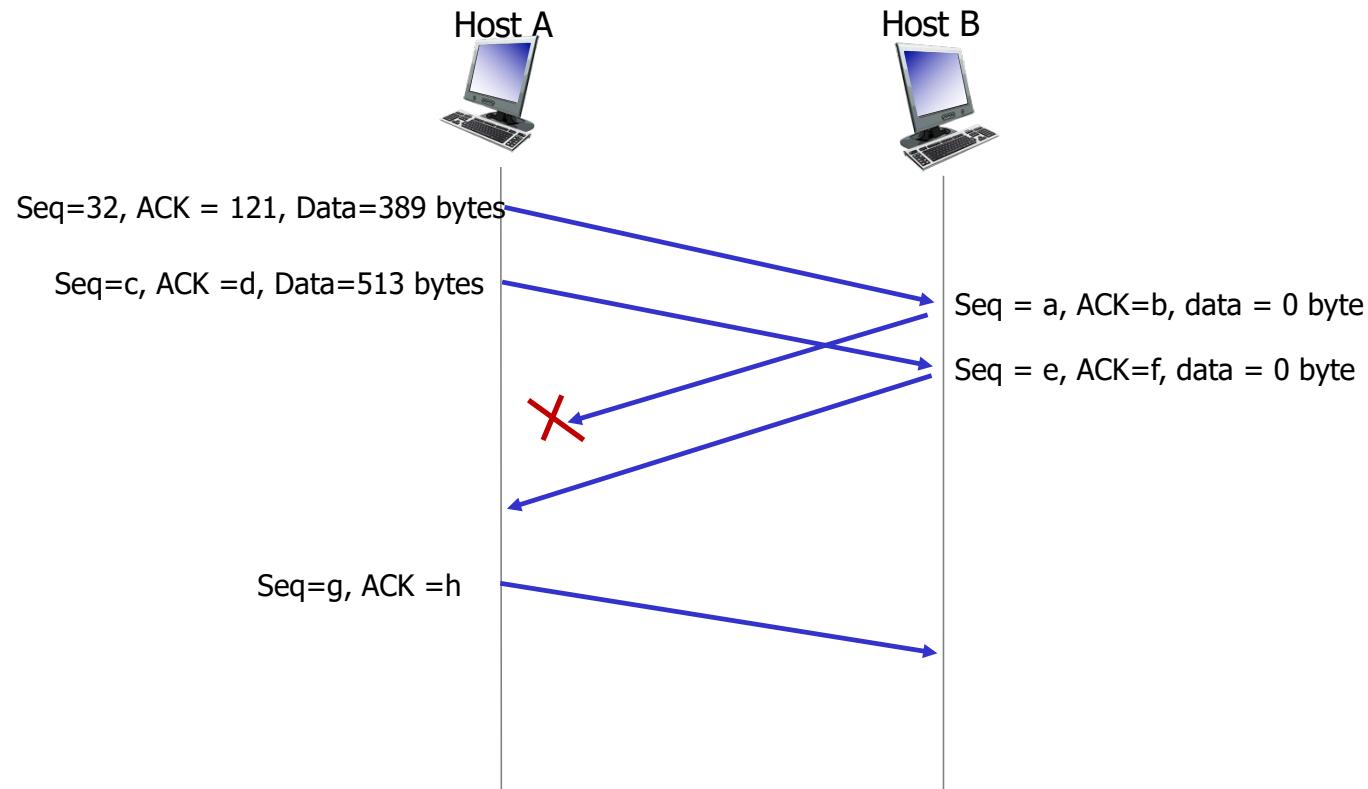
## TCP seq. numbers, ACKs



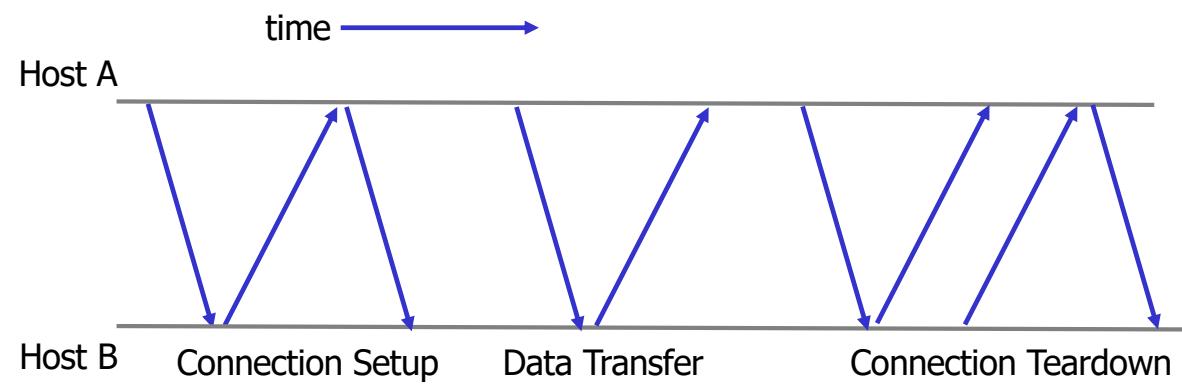
## TCP seq. numbers, ACKs

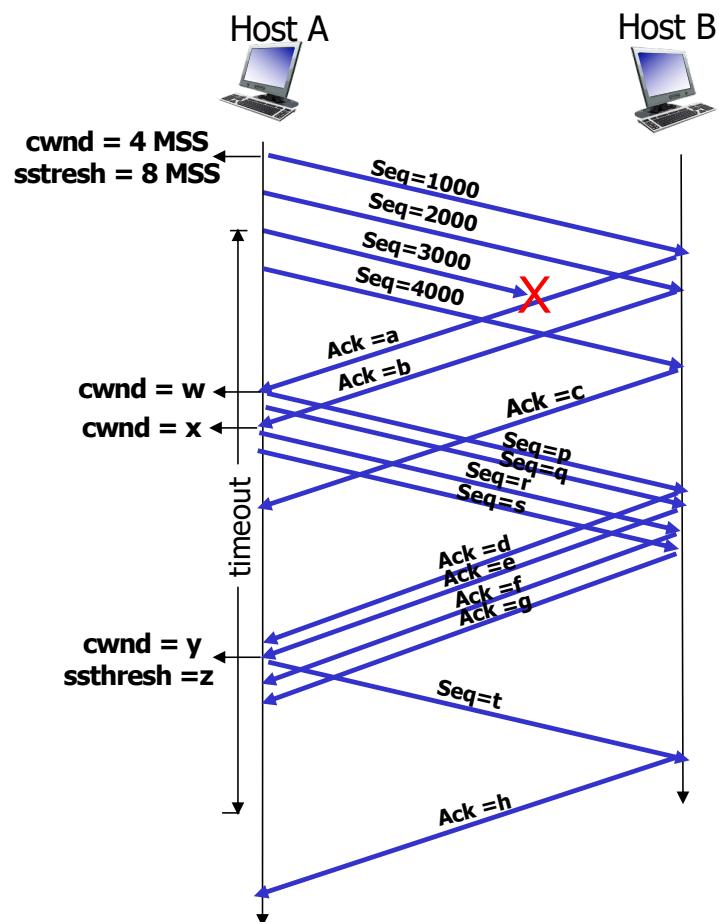


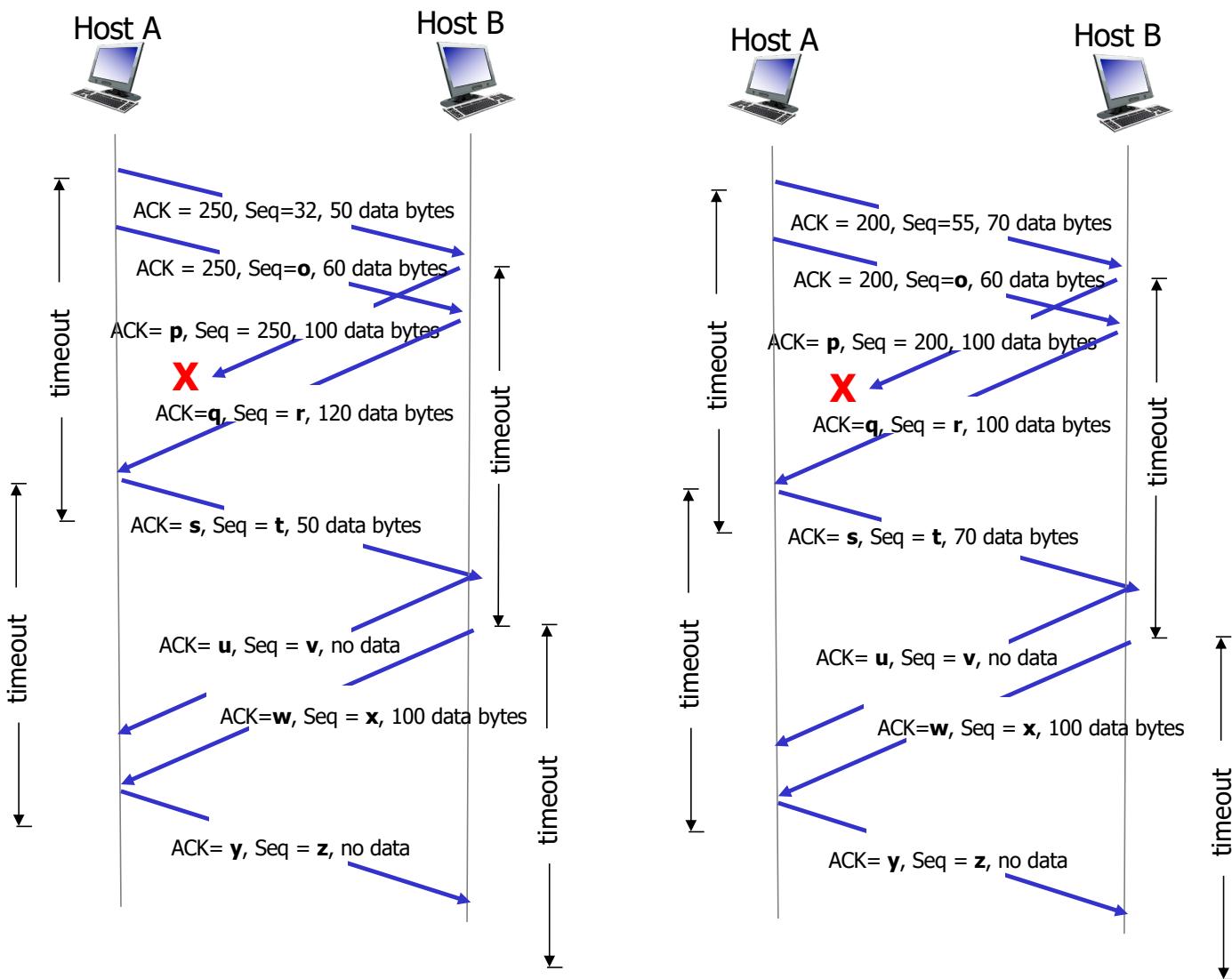
## TCP seq. numbers, ACKs

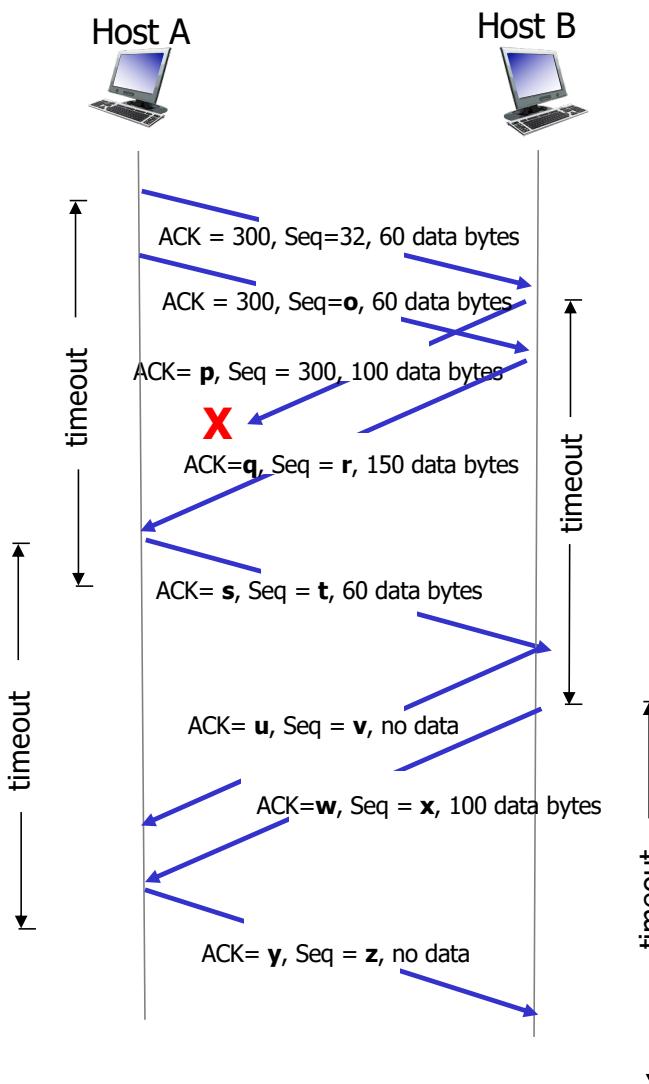


Seq=32, ACK = 121, Data=389 bytes      Seq=c, ACK =d, Data=513 bytes









# COMP 3331/9331: Computer Networks and Applications

1. IP datagram header format
2. IP fragmentation
3. IP addressing, subnets, network mask, longest prefix match, DHCP
4. Network address translation (NAT)

Week 7

Network Layer: Data Plane

Reading Guide: Chapter 4: Sections 4.1, 4.3

# Network Layer: outline

*Our goals:*

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

# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

-- **Not Covered**

## 4.3 IP: Internet Protocol

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

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

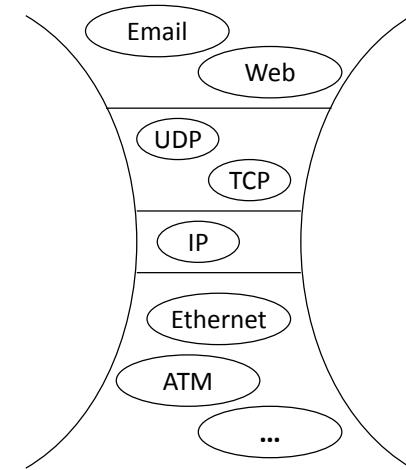
– **Not Covered**

# Some Background

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

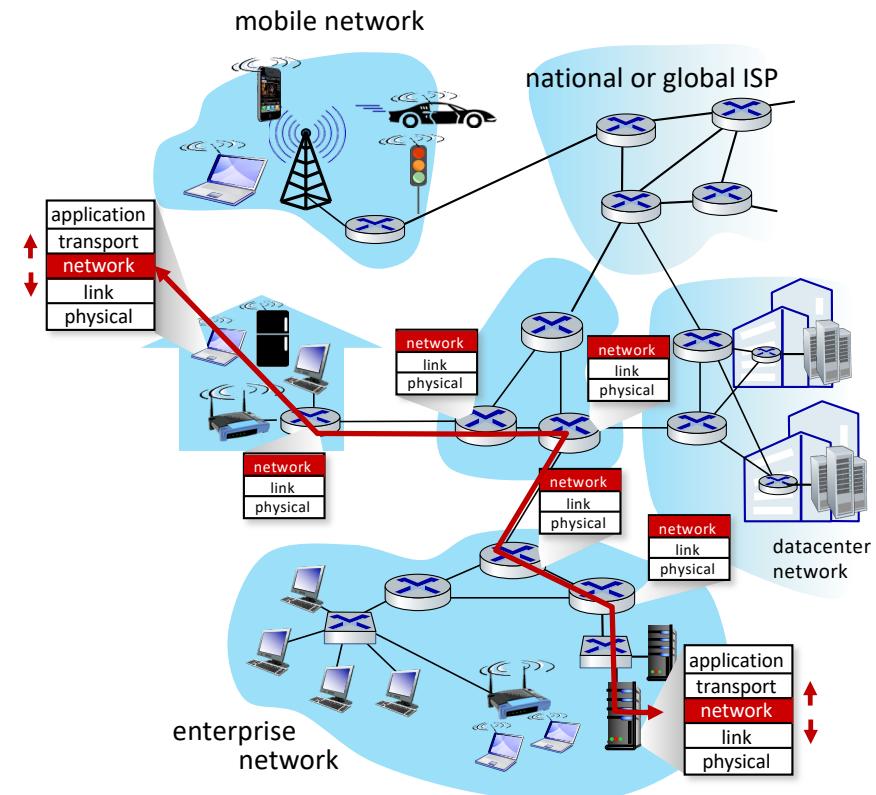
# Internetworking

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



# Network-layer 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:

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

## analogy: taking a trip

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

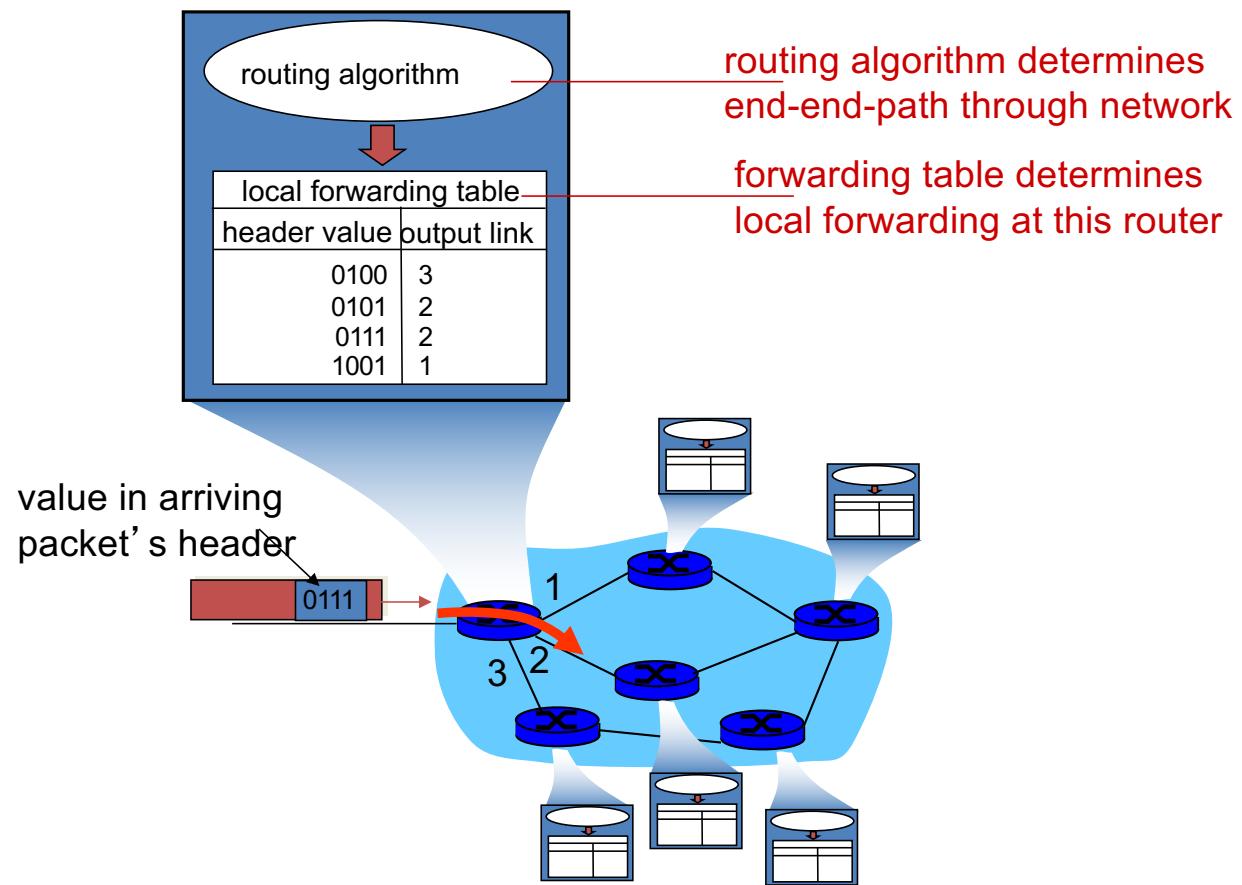


forwarding



routing

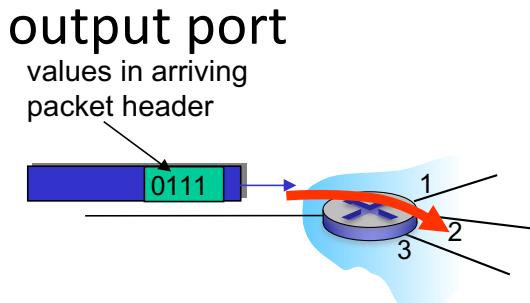
# Interplay between routing and forwarding



# Network layer: data plane, control plane

## Data plane:

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

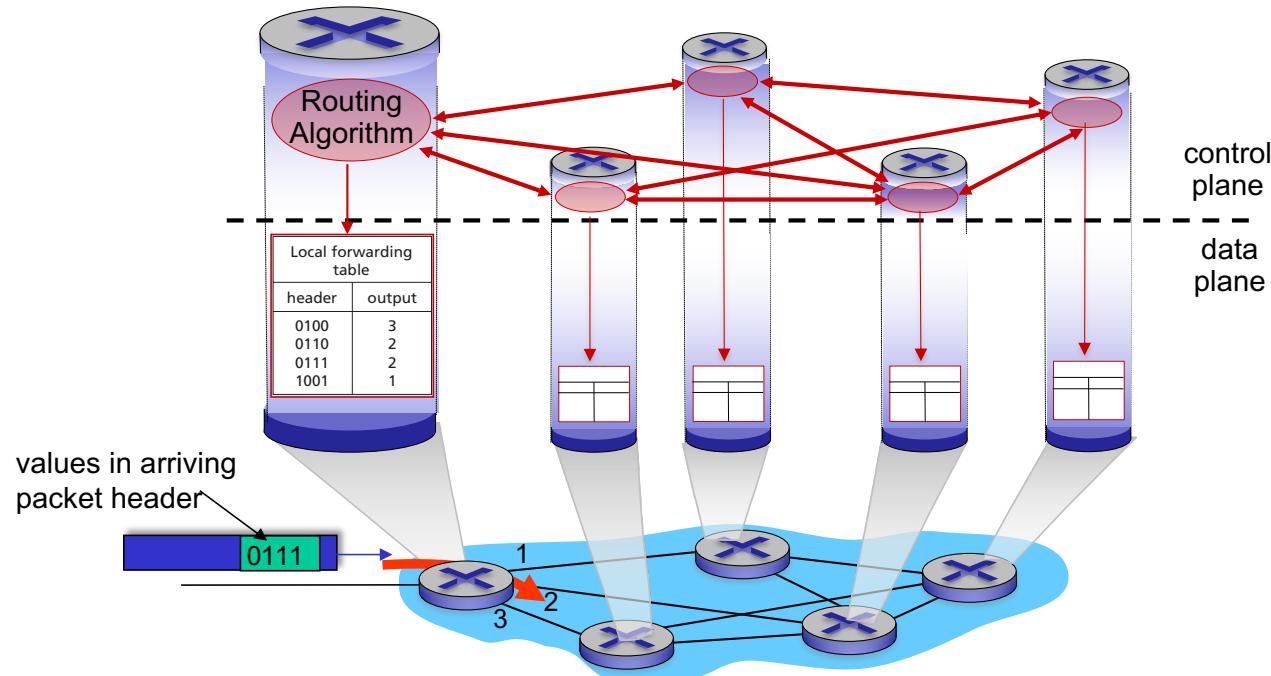


## Control plane

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

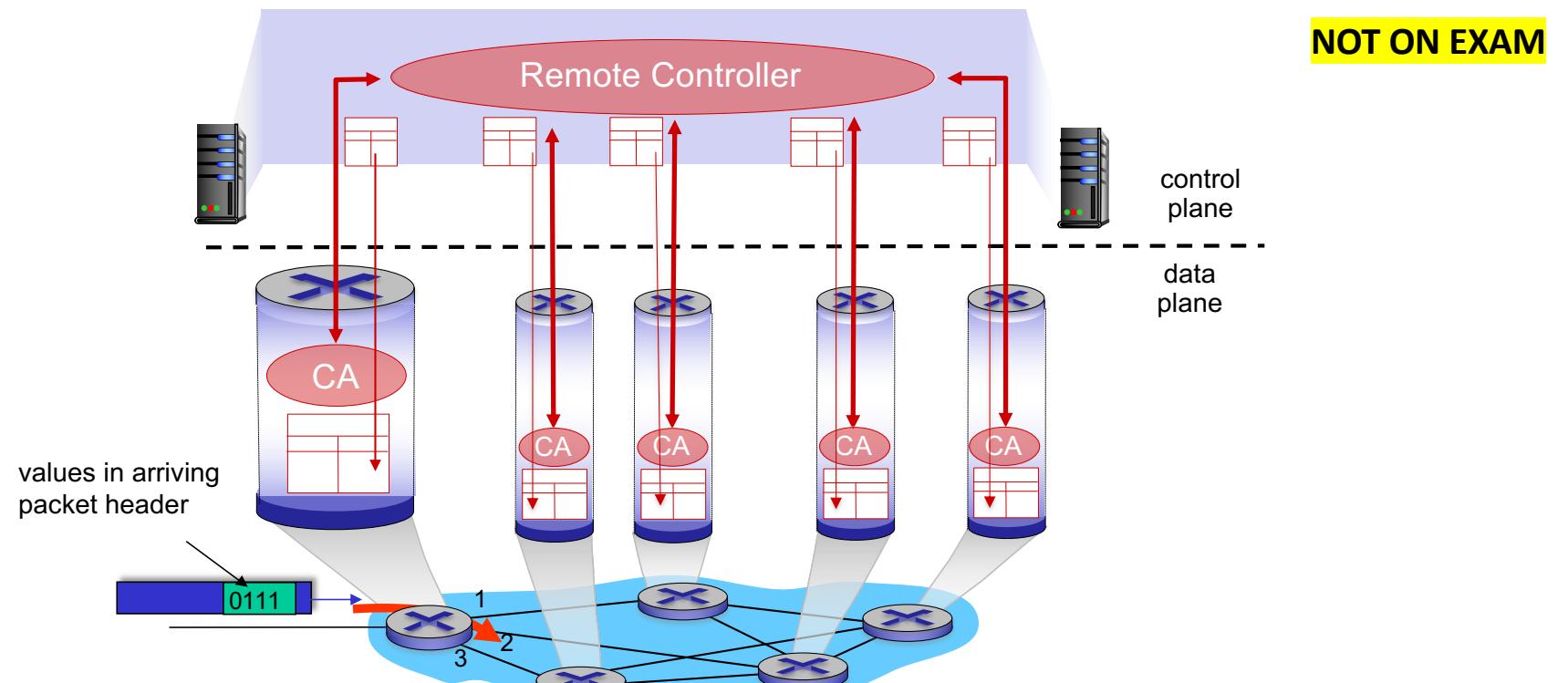
# Per-router control plane

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



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Network Layer: service model

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

Internet “best effort” service model

*No* guarantees on:

- i. successful datagram delivery to destination
- ii. timing or order of delivery
- iii. bandwidth available to end-end flow

## Reflections on best-effort service:

- simplicity of mechanism has allowed Internet to be widely deployed adopted
- sufficient provisioning of bandwidth allows performance of real-time applications (e.g., interactive voice, video) to be “good enough” for “most of the time”
- replicated, application-layer distributed services (datacenters, content distribution networks) connecting close to clients’ networks, allow services to be provided from multiple locations
- congestion control of “elastic” services helps

*It's hard to argue with success of best-effort service model*

# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

-- **Not Covered**

## 4.3 IP: Internet Protocol

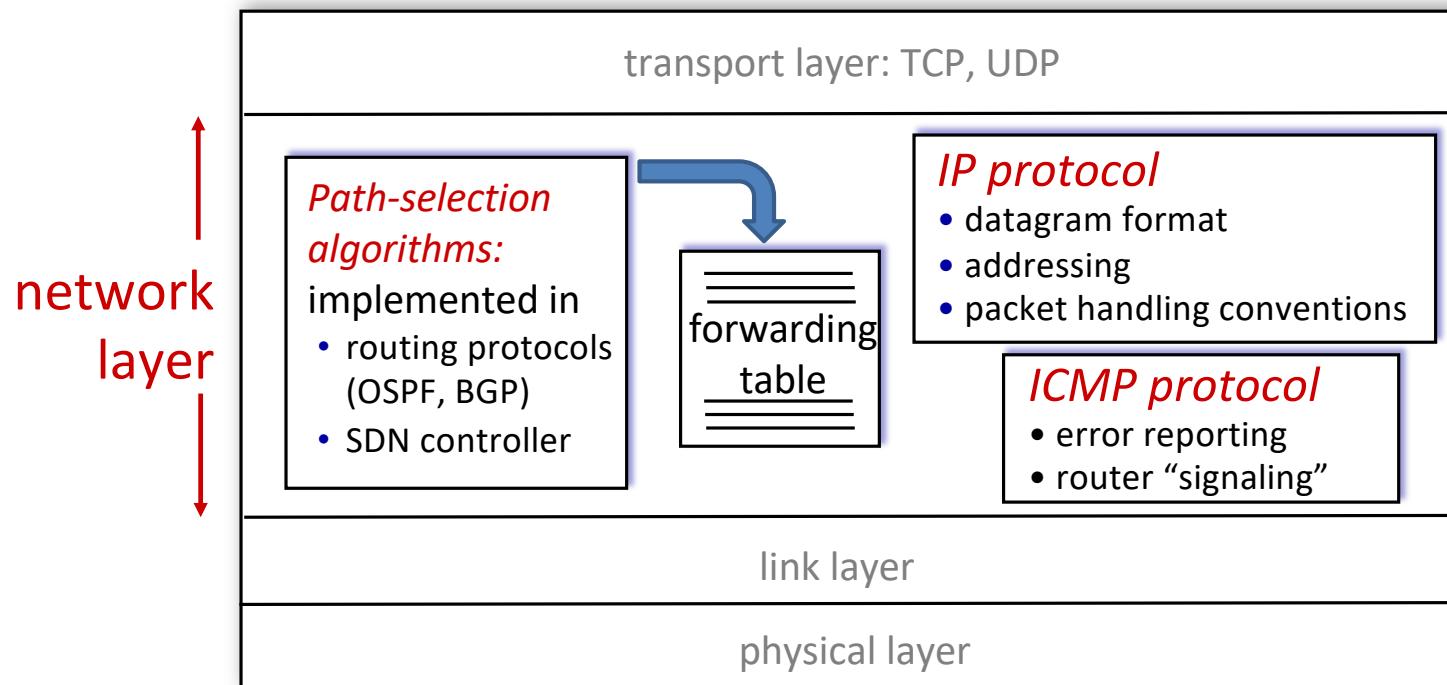
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

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

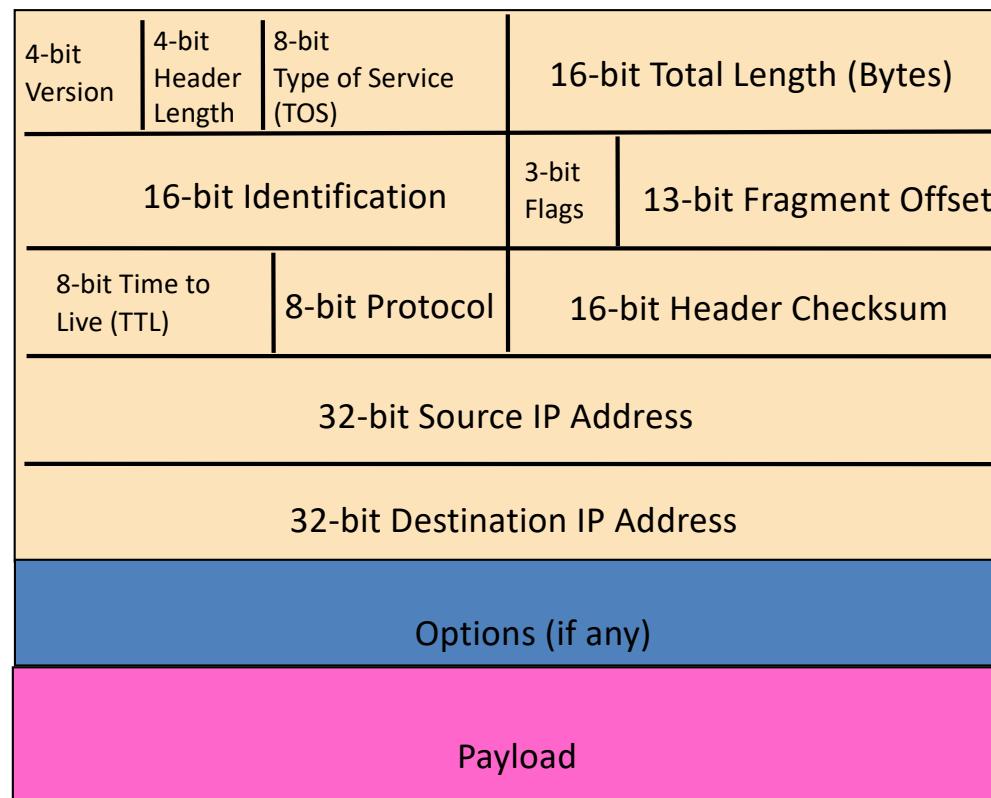
– **Not Covered**

# Network Layer: Internet

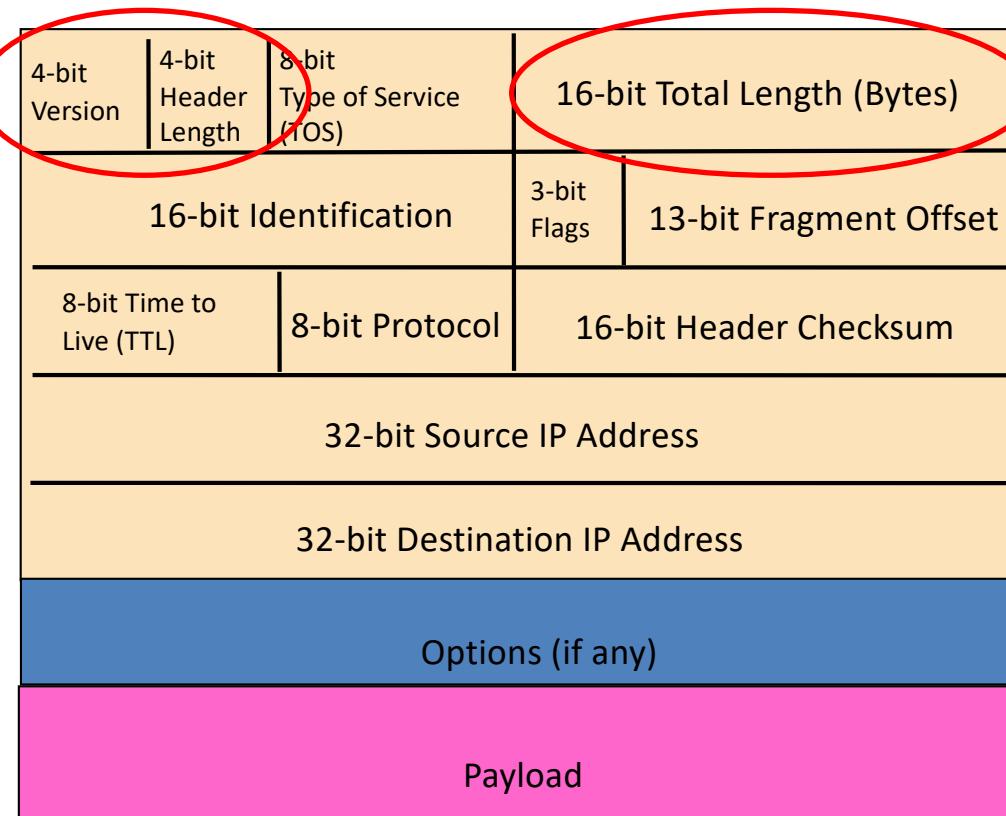
host, router network layer functions:



# IP Datagram Format



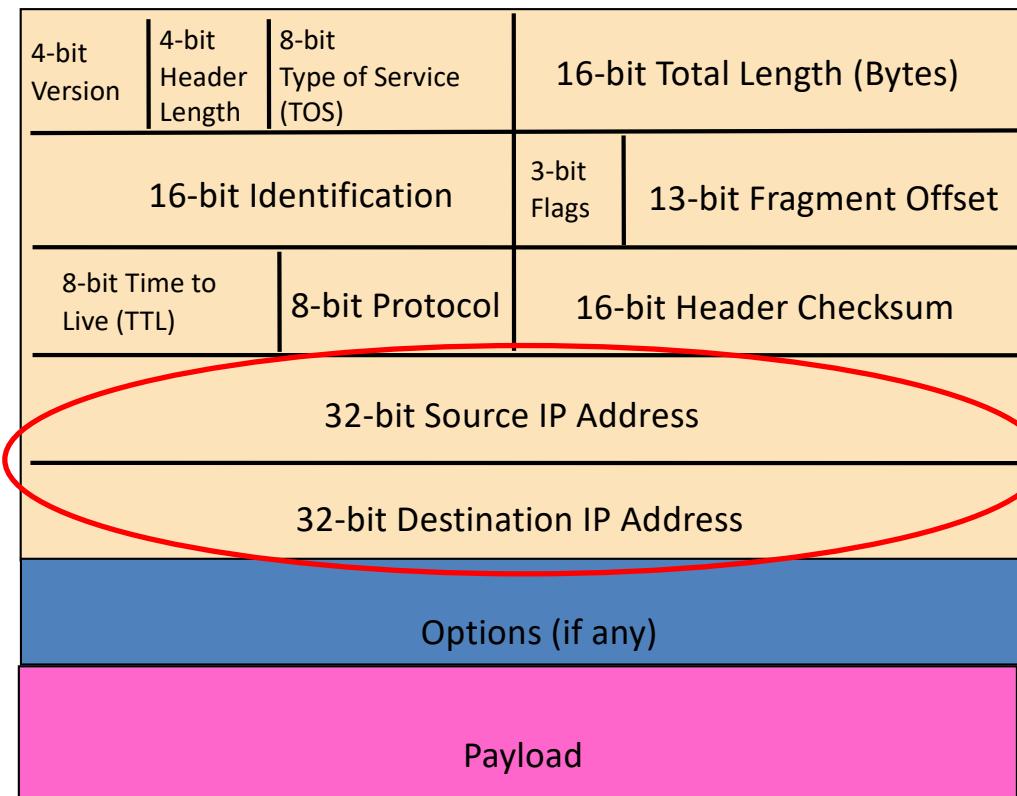
# Fields for Reading Packet Correctly



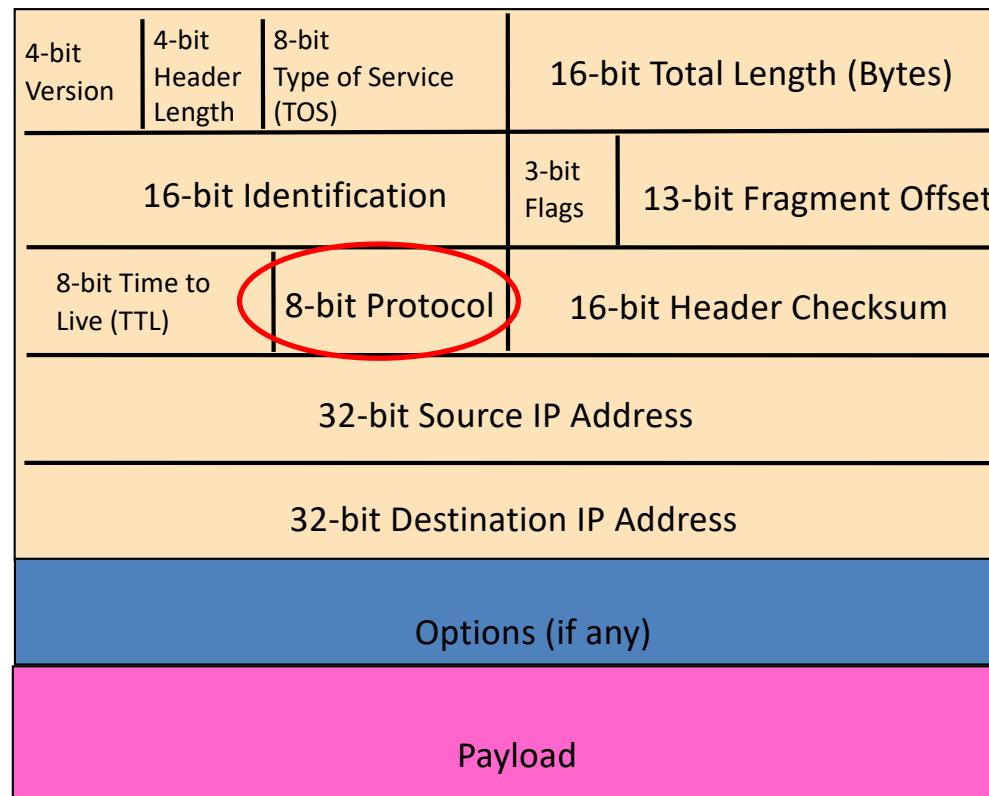
# Reading Packet Correctly

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

# Fields for Reaching Destination and Back

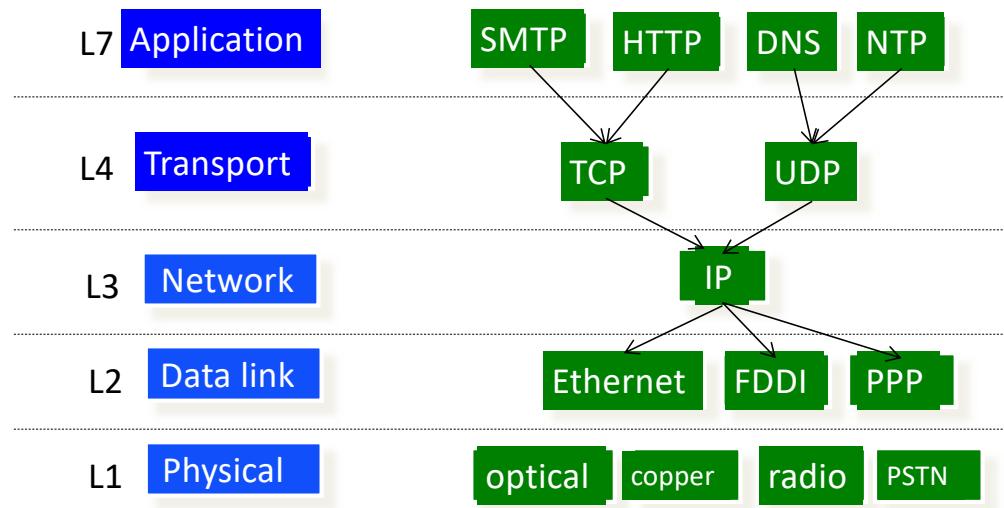


# Telling End-Host How to Handle Packet

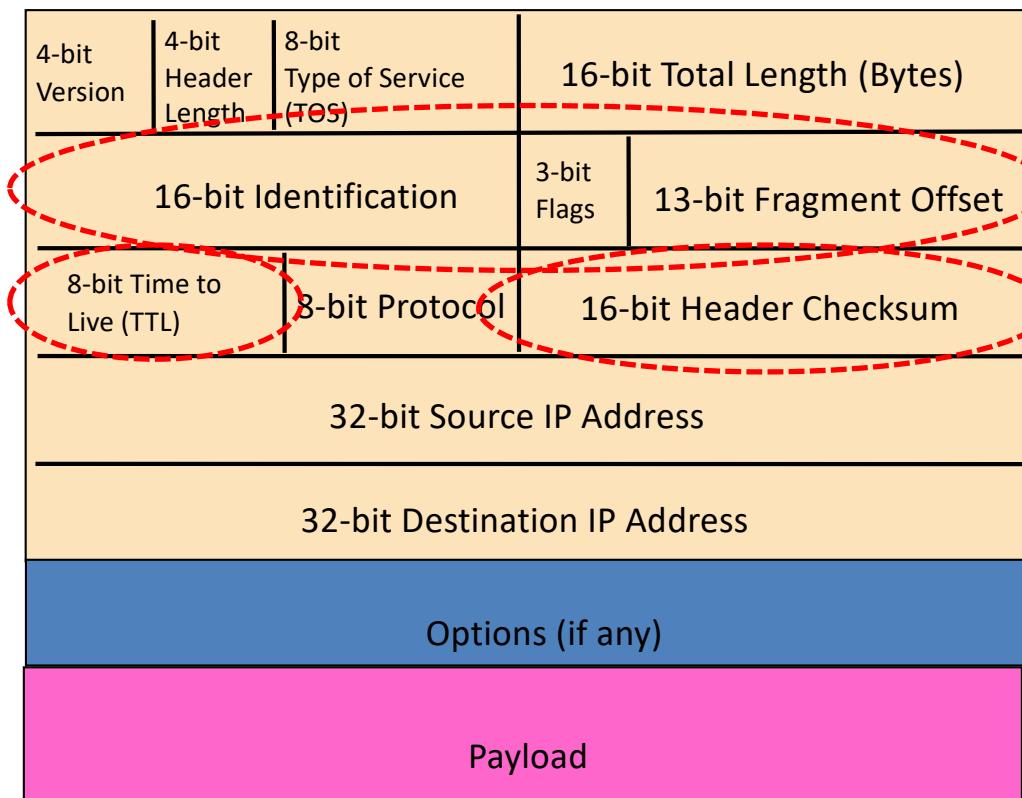


# Telling End-Host How to Handle Packet

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



# Checksum, TTL and Fragmentation Fields

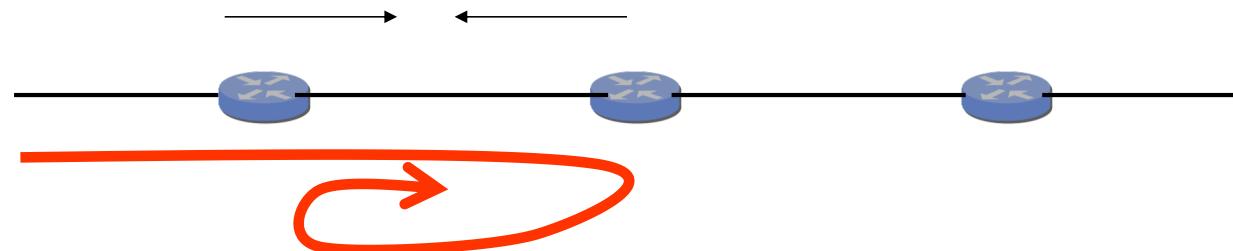


# Potential Problems

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

# Preventing Loops (TTL)

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



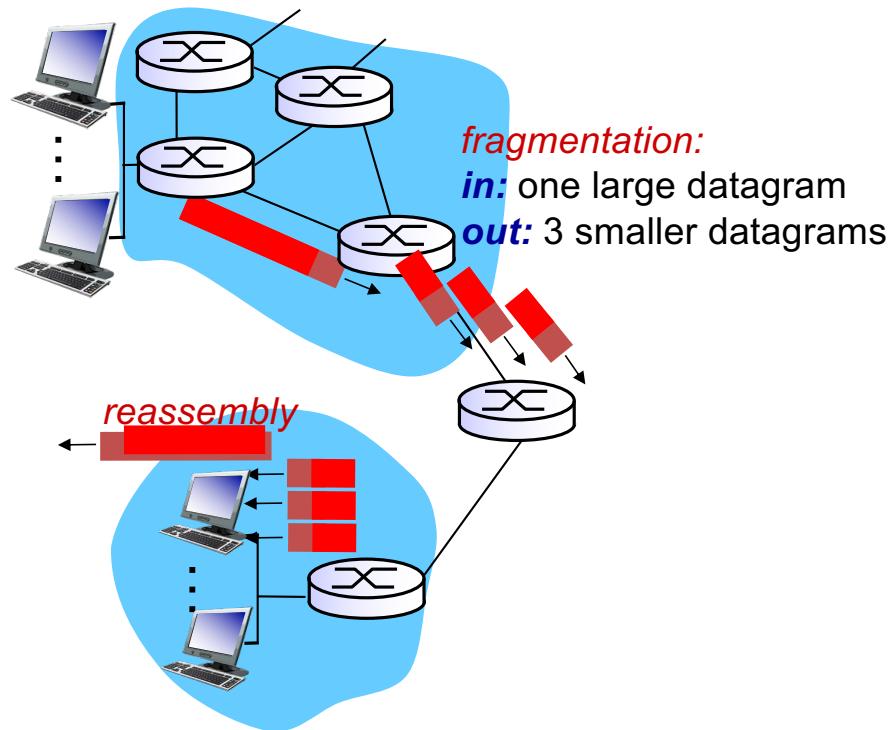
- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0
  - ...and “time exceeded” message is sent to the source
  - Recommended default value is 64

# Header Corruption (Checksum)

- Checksum (16 bits)
  - Only computed over packet header, method is same as UDP/TCP checksum
- If not correct, router discards packets
  - So, it doesn't act on bogus information
- Checksum recalculated at every router
  - Why?
  - Why include TTL?
  - Why only header?

# IP fragmentation, reassembly

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



# IP fragmentation, reassembly

Note: Offset is expressed as multiple of 8 bytes

## example:

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

$$\begin{aligned} \text{Data} &= 4000 - 20 \text{ (IP header)} \\ &= 3980 \end{aligned}$$

$$\begin{aligned} F1 &= 1480 \\ F2 &= 1480 \\ F3 &= 1020 \end{aligned}$$

1480 bytes in  
data field

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

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

*one large datagram becomes  
several smaller datagrams*

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

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

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

# IPv4 fragmentation procedure

## ➤ Fragmentation

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

## ➤ Re-assembly

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

## ➤ Fragmentation of fragments also supported

## Fragmentation of a fragment

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

			4,020
14,567	0	000	

Bytes 0000–3,999

Original datagram

			1,420
14,567	1	000	

Bytes 0000–1,399

Fragment 1

			1,420
14,567	1	175	

Bytes 1,400–2,799

Fragment 2

			1,220
14,567	0	350	

Bytes 2,800–3,999

Fragment 3

MTU=1420

			820
14,567	1	175	

Bytes 1,400–2,199

Fragment 2.1

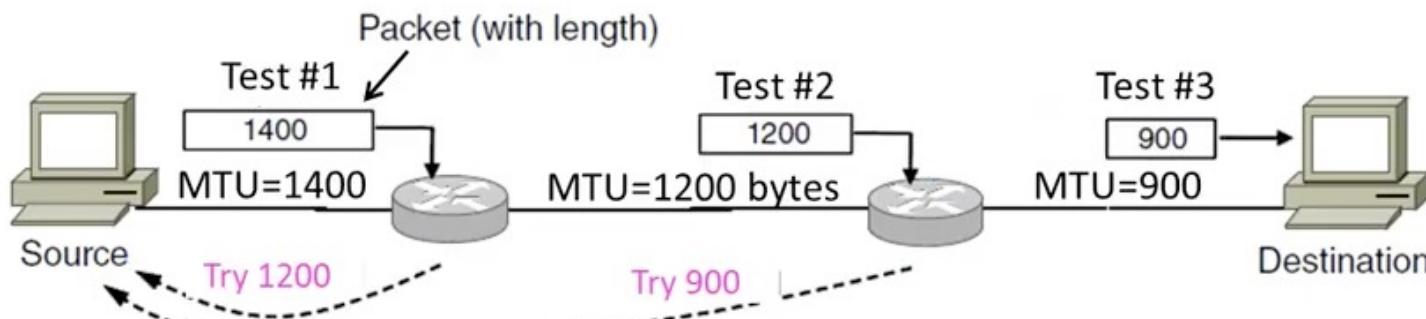
			620
14,567	1	275	

Bytes 2,200–2,799

Fragment 2.2

MTU=820

# Path MTU Discovery procedure



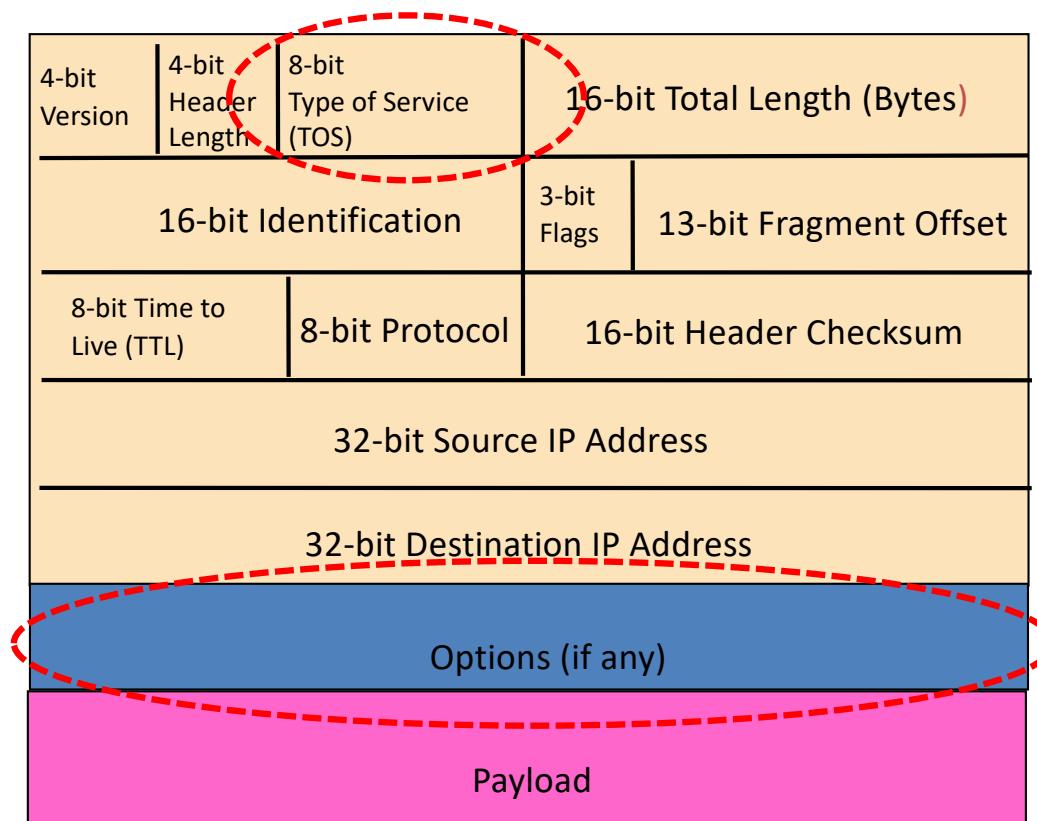
## ➤ Host

- Sends a big packet to test whether all routers in path to the destination can support or not
- Set DF (Do not fragment) flag

## ➤ Routers

- Drops the packet if it is too large (as DF is set)
- Provides feedback to Host with ICMP message telling the maximum supported size

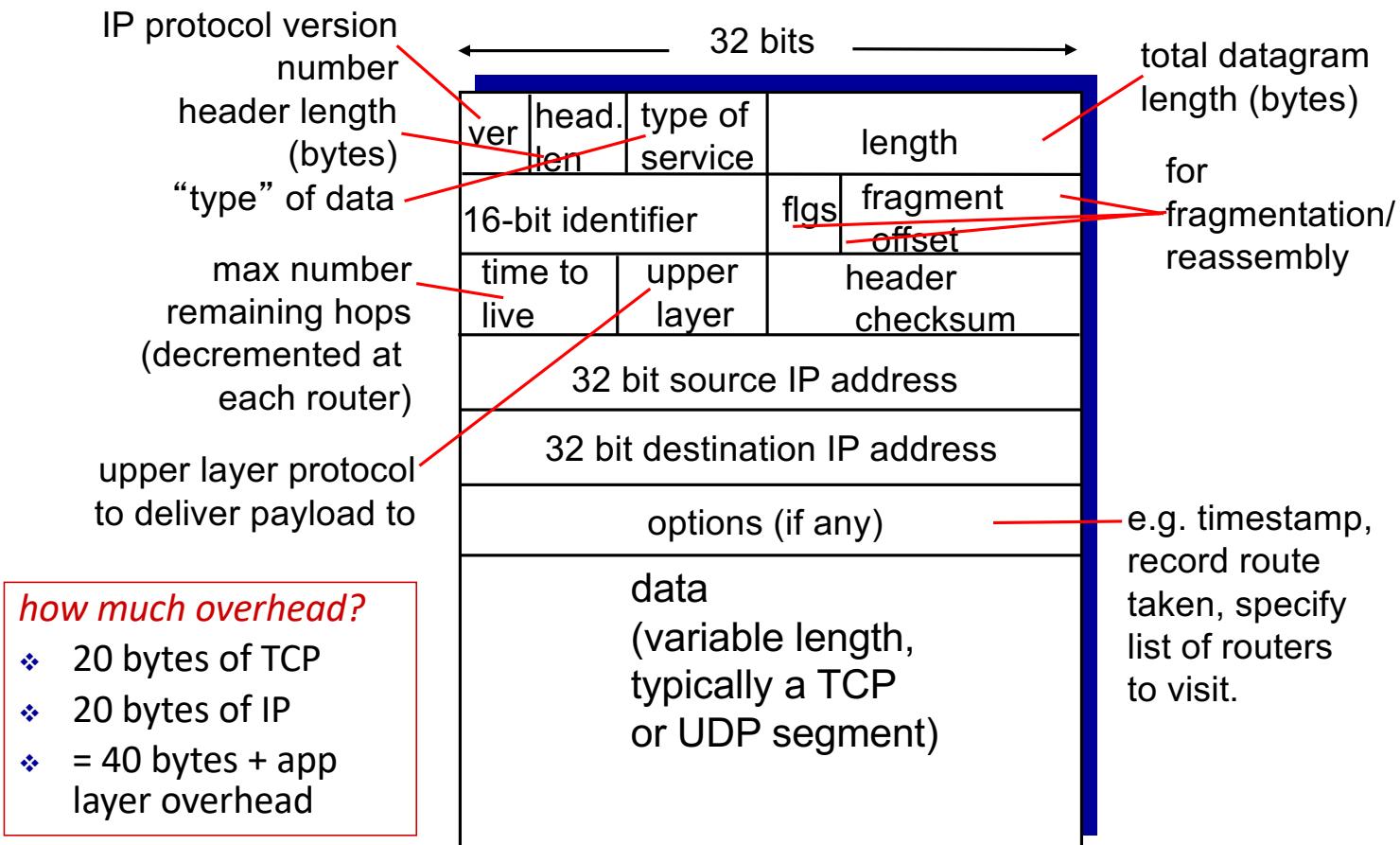
# Fields for Special Handling



# Special Handling

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

# RECAP: IP datagram format



# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

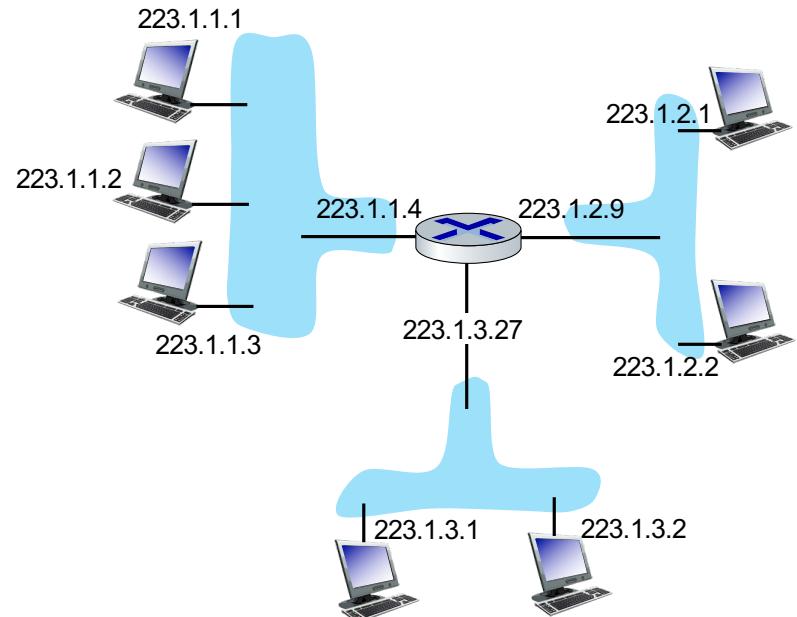
## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

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

# IP addressing: introduction

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



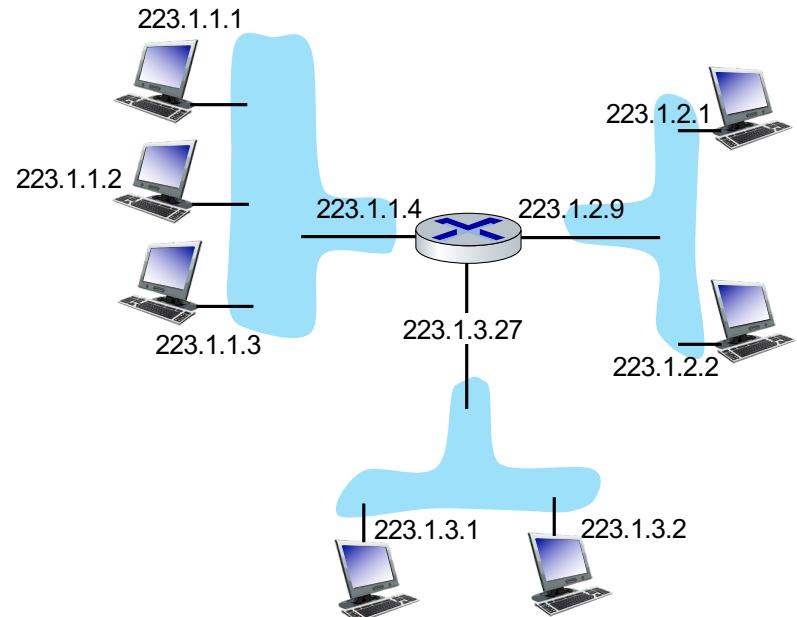
dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

# IP addressing: introduction

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



dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001

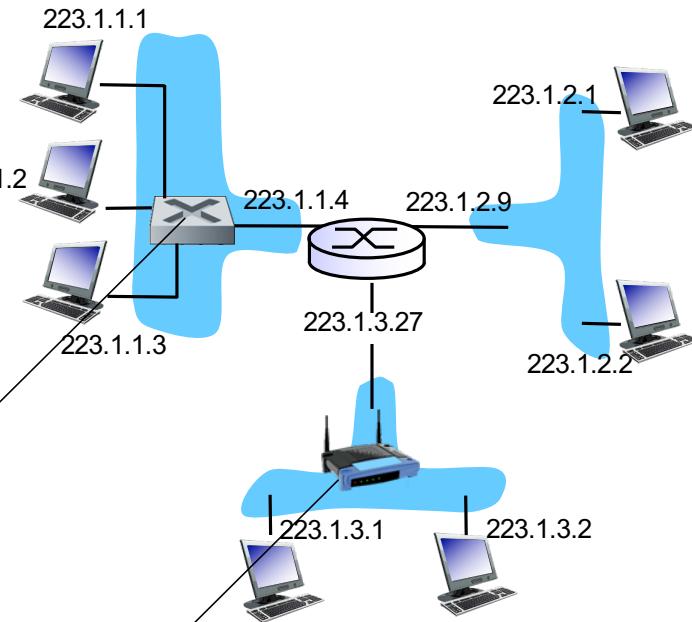
223      1      1      1

# IP addressing: introduction

*Q: how are interfaces  
actually connected?*

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

*A: wired Ethernet interfaces  
connected by Ethernet switches*



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

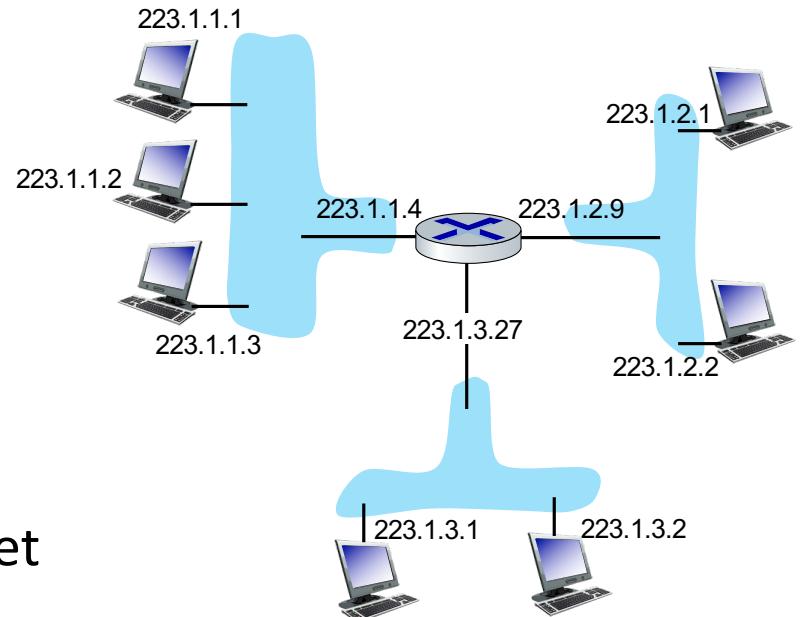
# Subnets

- *What's a subnet ?*

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

- *IP addresses have structure:*

- **subnet part:** devices in same subnet have common high order bits
- **host part:** remaining low order bits

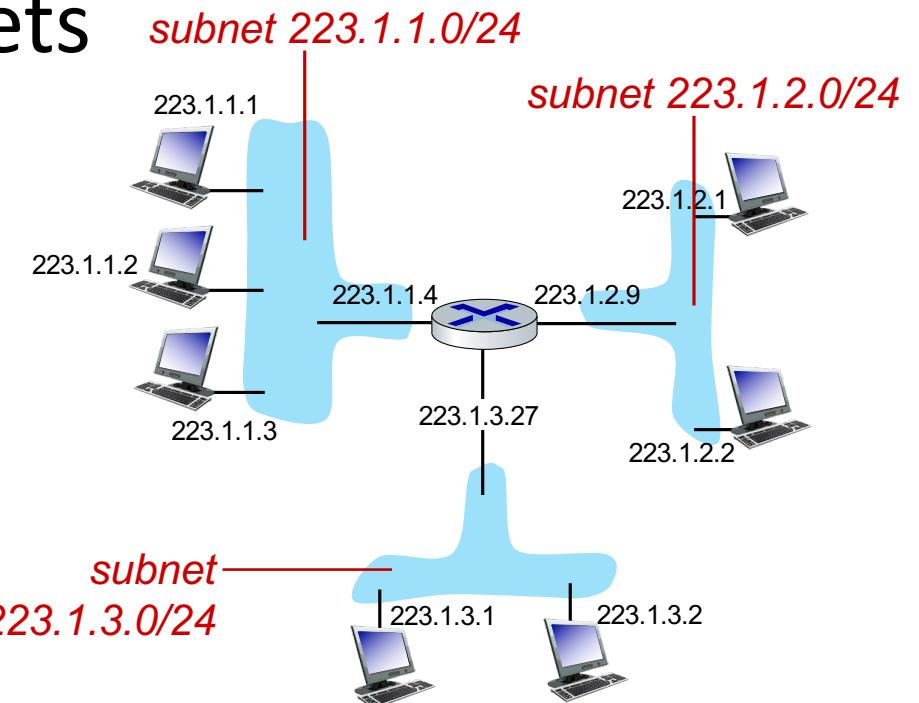


network consisting of 3 subnets

# Subnets

*Recipe for defining subnets:*

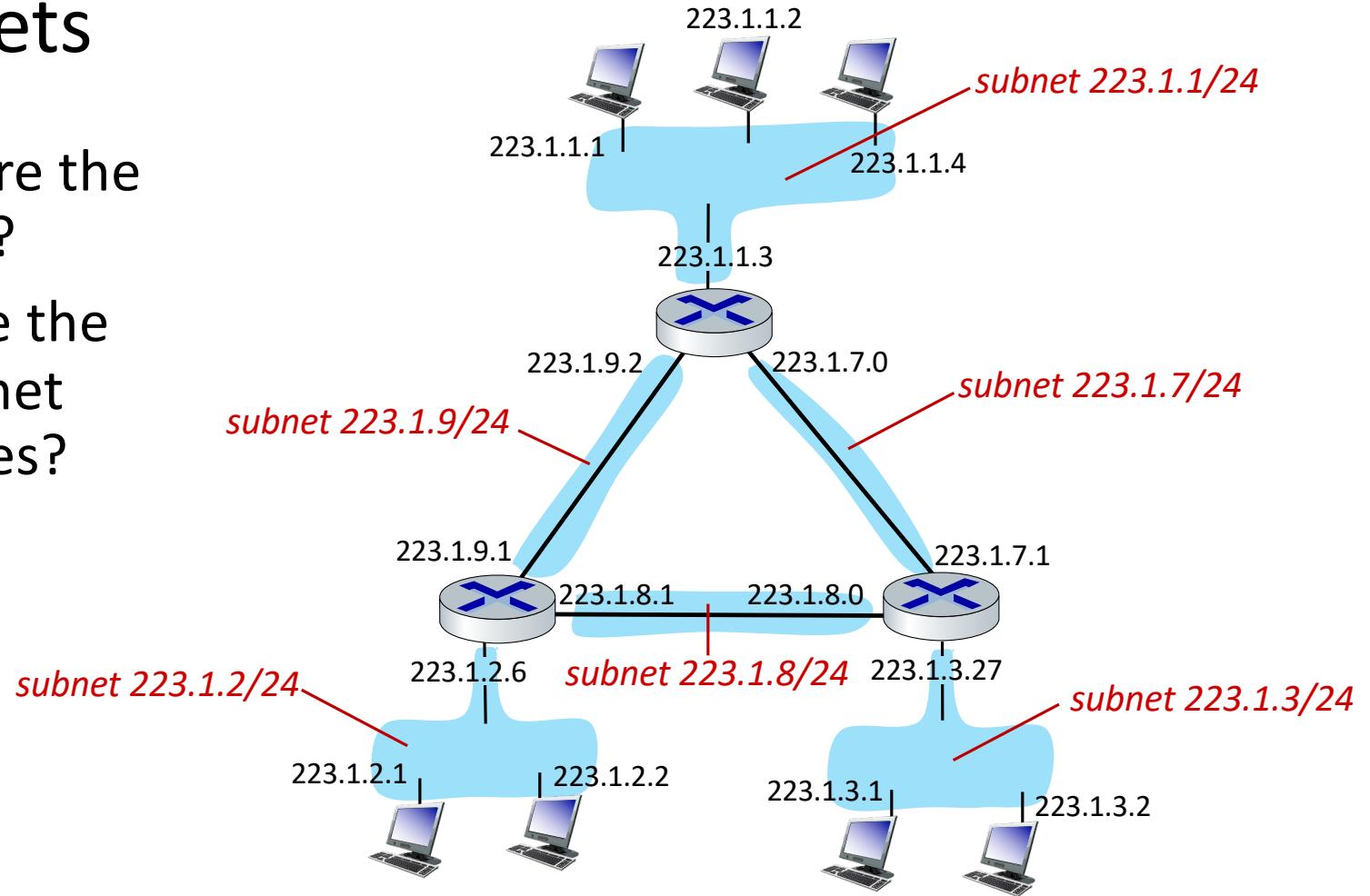
- detach each interface from its host or router, creating “islands” of isolated networks
- each isolated network is called a *subnet*



subnet mask: /24  
(high-order 24 bits: subnet part of IP address)

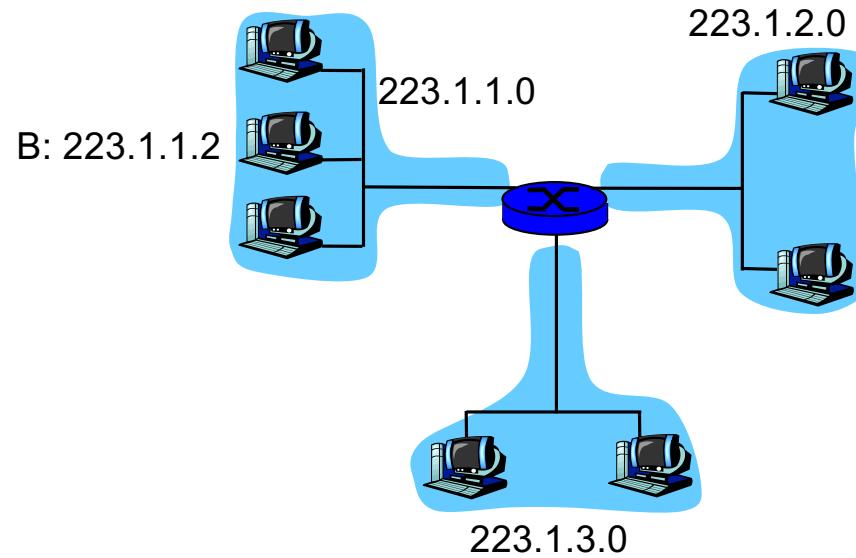
# Subnets

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



# Network Mask

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



Host B	Dot-decimal address	Binary
IP address	223.1.1.2	11011111.00000001.00000001.00000010
Mask	255.255.255.0	11111111.11111111.11111111.00000000
Network Part	223.1.1.0	11011111.00000001.00000001.00000000
Host Part	0.0.0.2	00000000.00000000.00000000.00000010

# Original Internet Addresses

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

*Assumed 256 networks were more than enough!*

# Next design: Class-ful Addresses

Used till the introduction of CIDR 1993

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

Problem: Networks only come in three sizes!

## What are the issues?

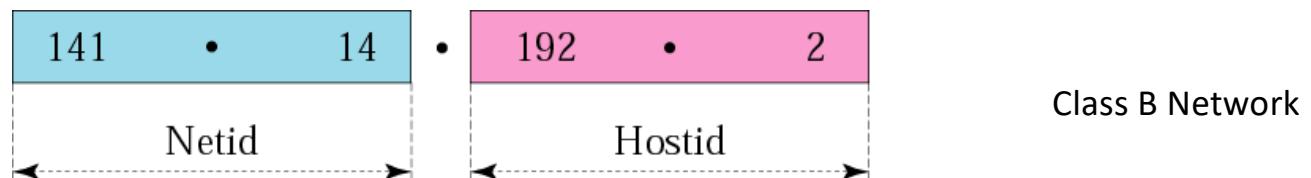
- An organization requires 6 nets each of size 30.  
Does it have to buy 6 class C address blocks?
  
- An organization requires 512 addresses? How  
many IP addresses should it buy?

# Subnetting

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

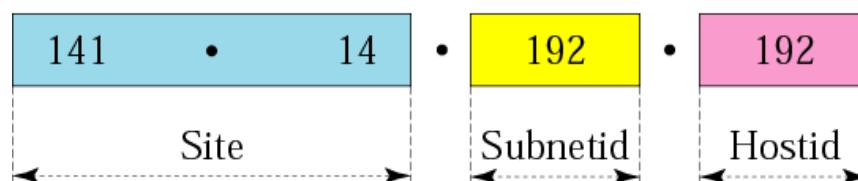
# Subnetting

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



a. Without subnetting

Class B Network



b. With subnetting

## Quiz?

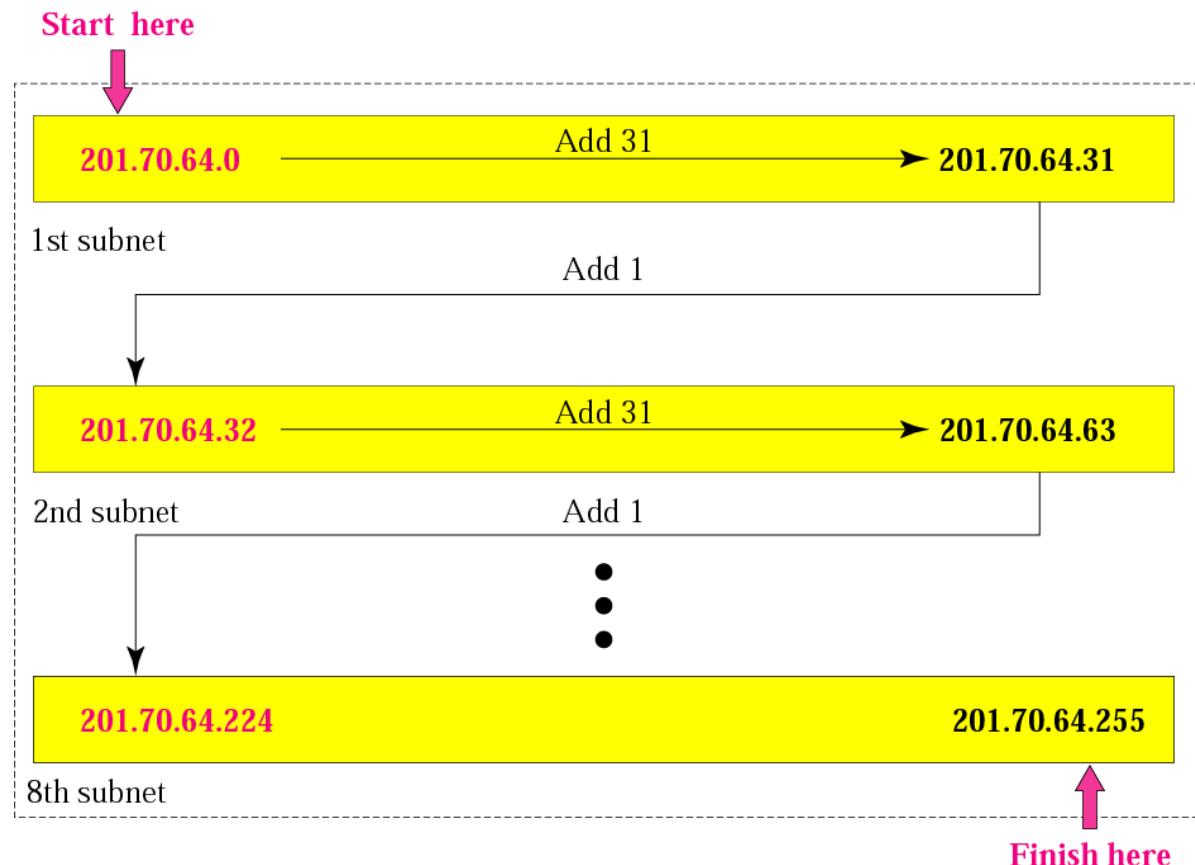
A company is granted the site address 201.70.64.0 (class C). The company needs six subnets. Design the subnets.

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

11111111 11111111 11111111 11100000  
or 255.255.255.224

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

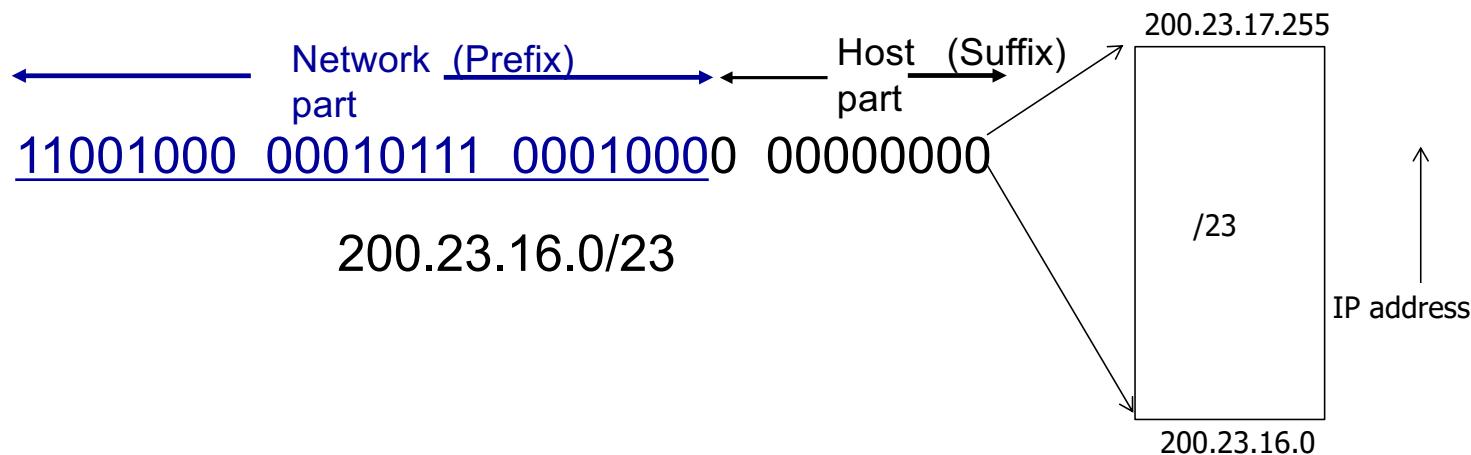
The number of addresses in each subnet is  $2^5$  or 32.



# Today's addressing: CIDR

## CIDR: Classless InterDomain Routing

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

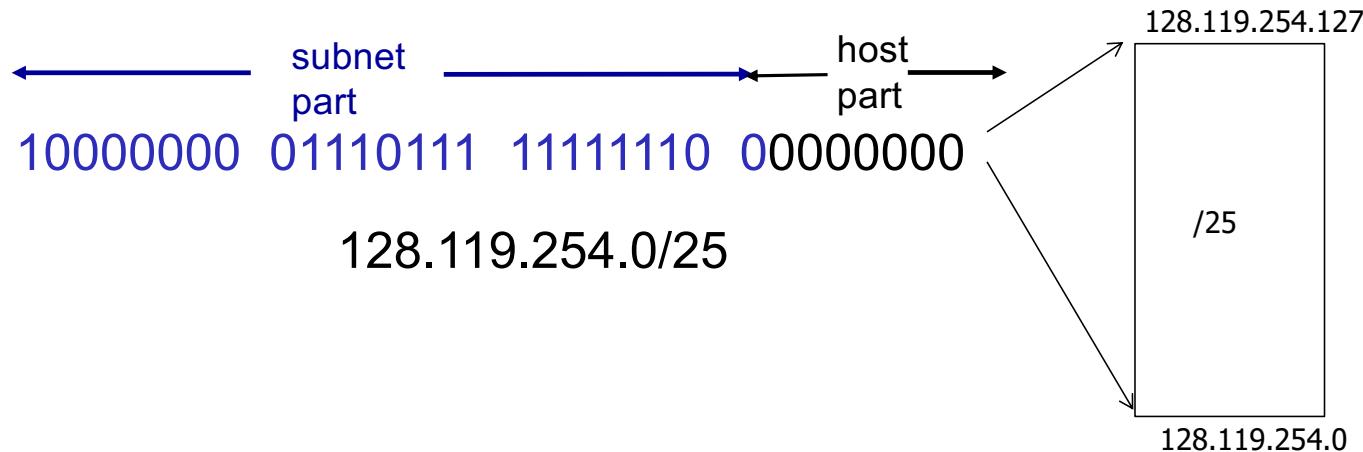


# Quiz: IP Addressing



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

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



# Quiz: IP Addressing



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

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



## Quiz: IP Addressing

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

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

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

## Group 1

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

01: 190.100.0.0/24 → 190.100.0.255/24

02: 190.100.1.0/24 → 190.100.1.255/24

.....

64: 190.100.63.0/24 → 190.100.63.255/24

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

## Group 2

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

001: 190.100.64.0/25  $\rightarrow$  190.100.64.127/25

002: 190.100.64.128/25  $\rightarrow$  190.100.64.255/25

.....

128: 190.100.127.128/25  $\rightarrow$  190.100.127.255/25

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

## Group 3

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

001:190.100.128.0/26 → 190.100.128.63/26

002:190.100.128.64/26 → 190.100.128.127/26

.....

128:190.100.159.192/26 → 190.100.159.255/26

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

Number of granted addresses: 65, 536

Number of allocated addresses: 40, 960

Number of available addresses: 24, 576

# IP addresses: how to get one?

That's actually **two** questions:

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

How does *host* get IP address?

- hard-coded by sysadmin in config file (e.g., `/etc/rc.config` in UNIX)
- **DHCP: Dynamic Host Configuration Protocol:** dynamically get address from server
  - “plug-and-play”

# DHCP: Dynamic Host Configuration Protocol

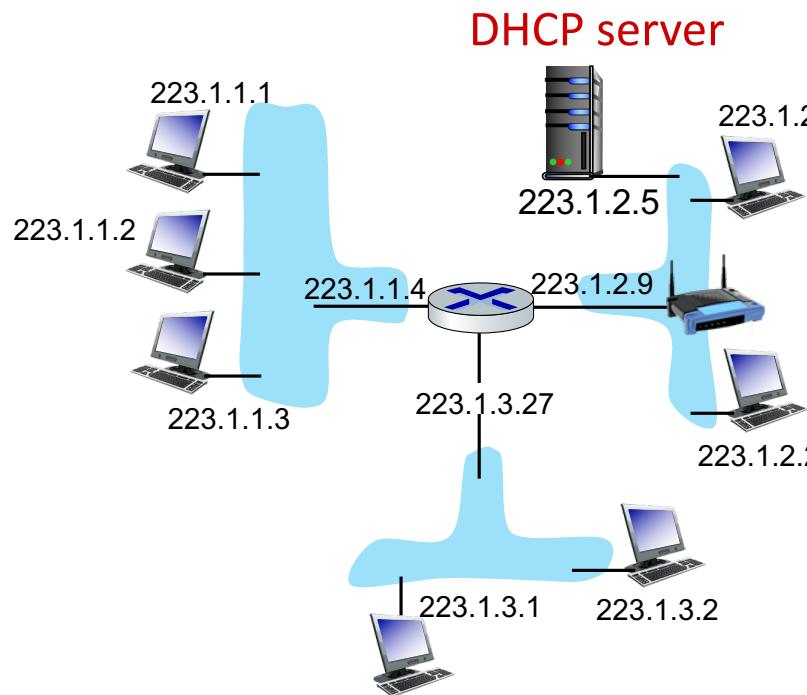
**goal:** host *dynamically* obtains IP address from network server when it “joins” network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

## DHCP overview:

- host broadcasts **DHCP discover** msg [optional]
- DHCP server responds with **DHCP offer** msg [optional]
- host requests IP address: **DHCP request** msg
- DHCP server sends address: **DHCP ack** msg

# DHCP client-server scenario



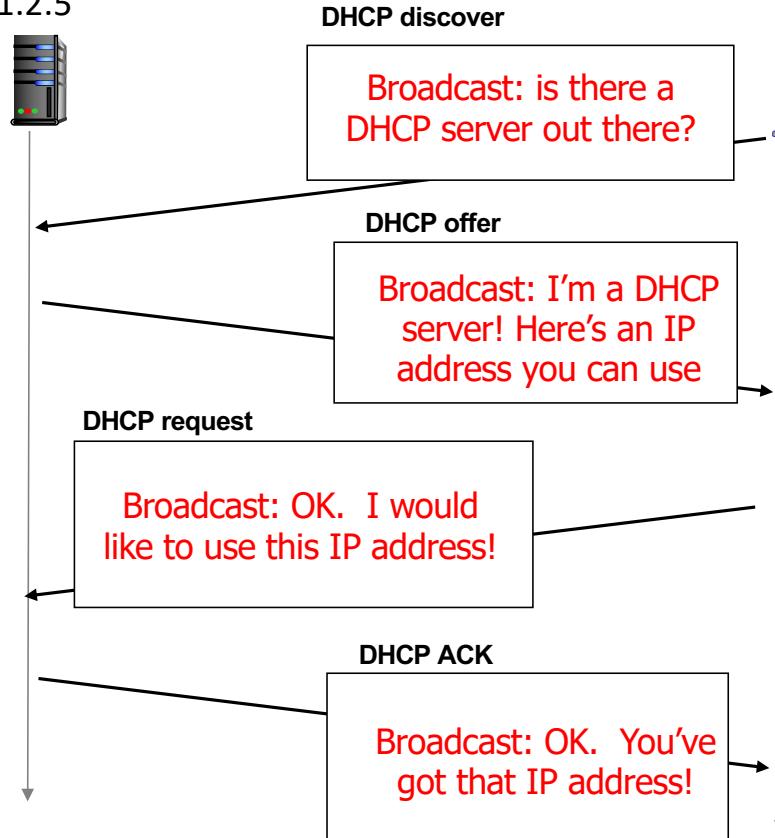
Typically, DHCP server will be co-located in router, serving all subnets to which router is attached



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

# DHCP client-server scenario

DHCP server: 223.1.2.5



Arriving client

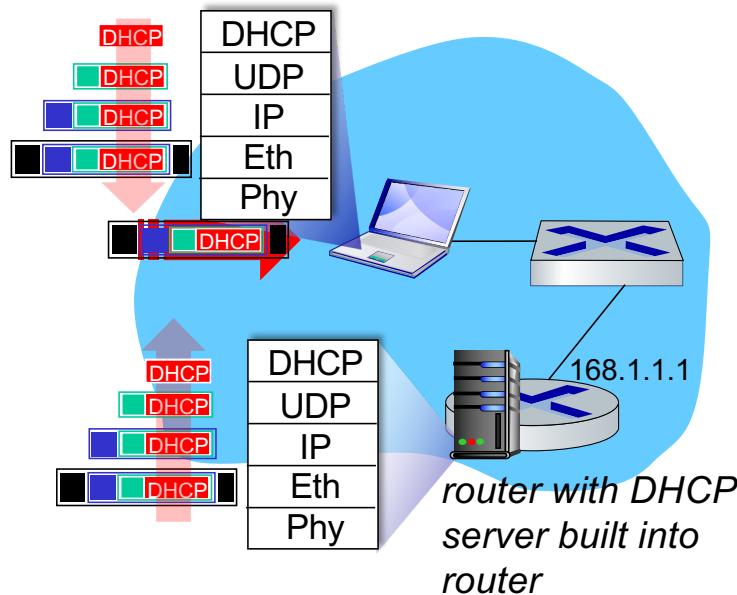
The two steps above can be skipped "if a client remembers and wishes to reuse a previously allocated network address"  
[RFC 2131]

# DHCP: more than IP addresses

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

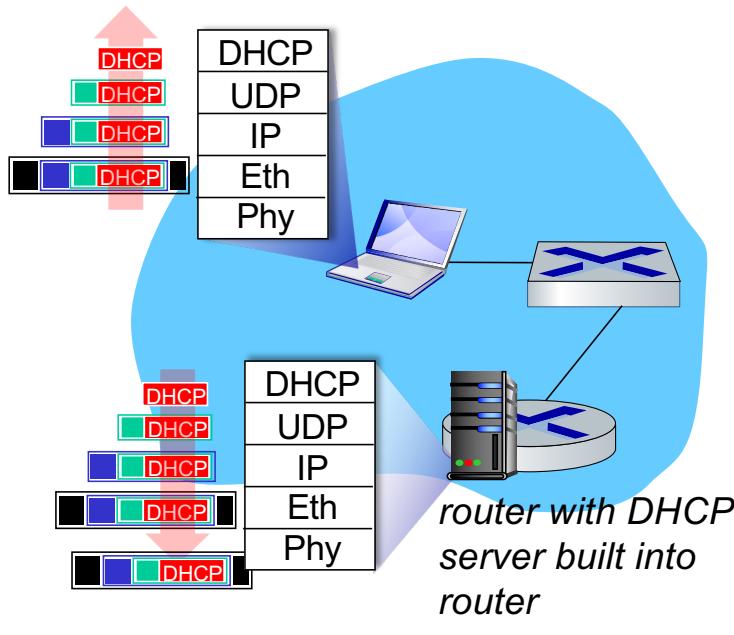
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# DHCP: example



- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP

# DHCP: example



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

Note: Only last 2 steps of the 4-way message exchange are shown

# IP addresses: how to get one?

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

**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

...

.....

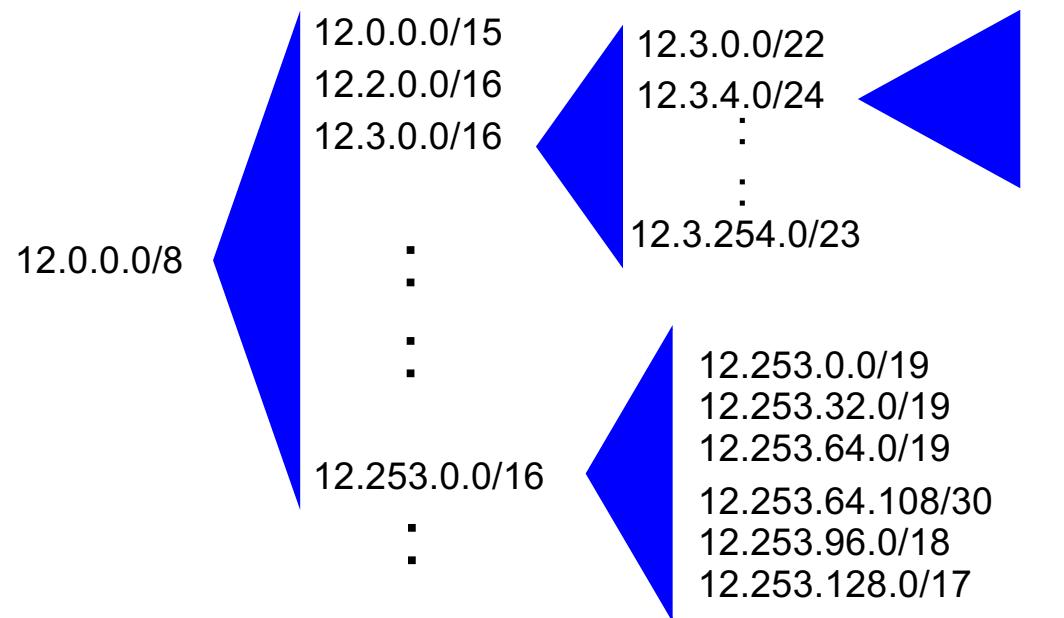
....

....

Organization 7    11001000 00010111 00011110 00000000    200.23.30.0/23

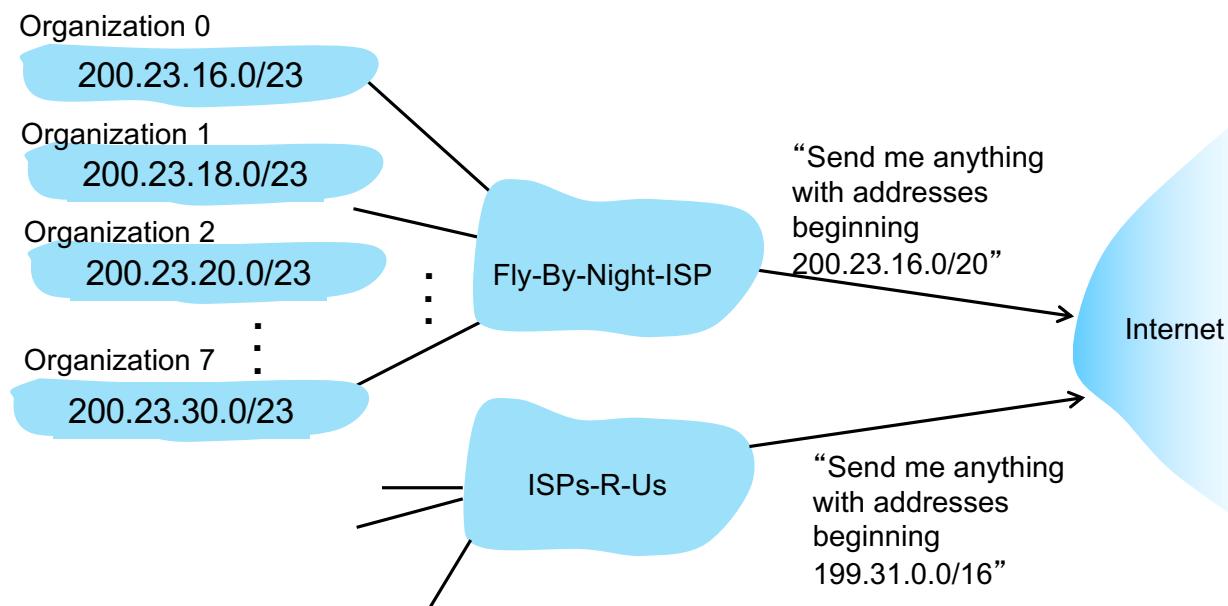
# CIDR: Addresses allocated in contiguous prefix chunks

Recursively break down chunks as get closer to host



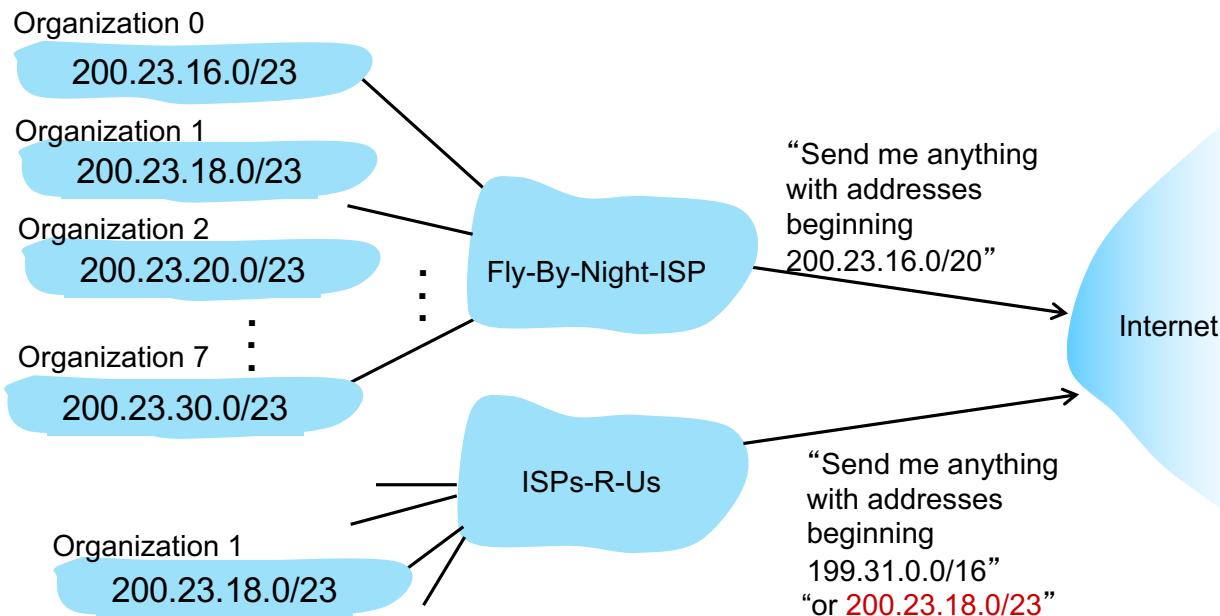
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



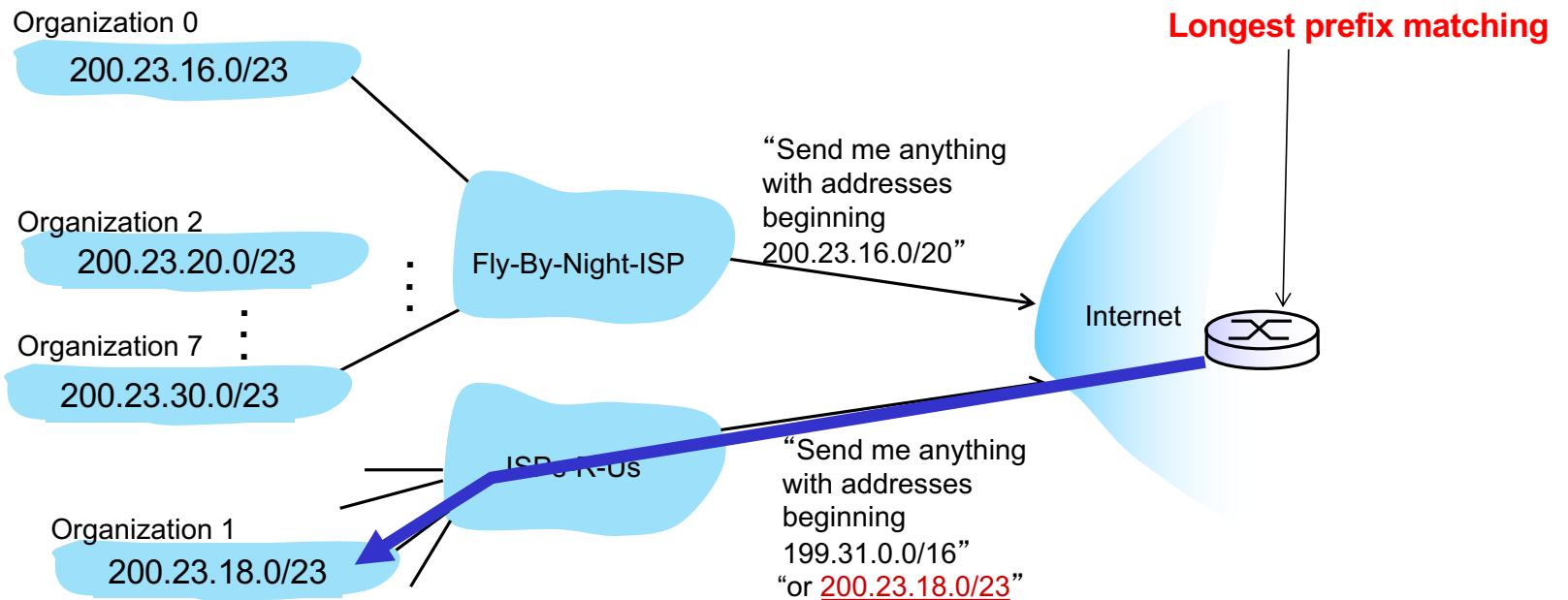
# Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



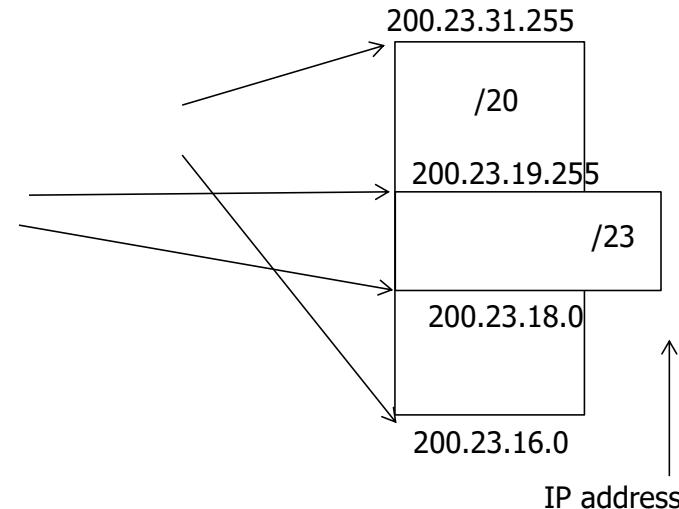
# Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



## Example: continued

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



# Longest prefix matching

*longest prefix matching* —

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

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

examples:

DA: 11001000 00010111 00010110 10100001      which interface?

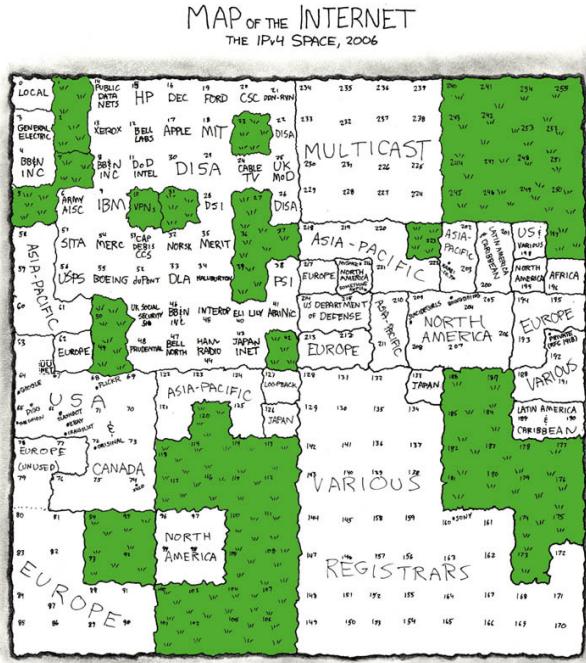
DA: 11001000 00010111 00011000 10101010      which interface?

# More on IP addresses

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

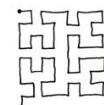
<http://www.geobytes.com/IpLocator.htm>

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



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

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



= UNALLOCATED BLOCK

# IP Addressing: the last word...

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

**A:** ICANN: Internet Corporation for Assigned

Names and Numbers <http://www.icann.org/>



IANA works through Regional Internet Registries (RIRs):



IRéseaux IP Européens Network Coordination Centre



American Registry for Internet Numbers



Asia-Pacific Network Information Center

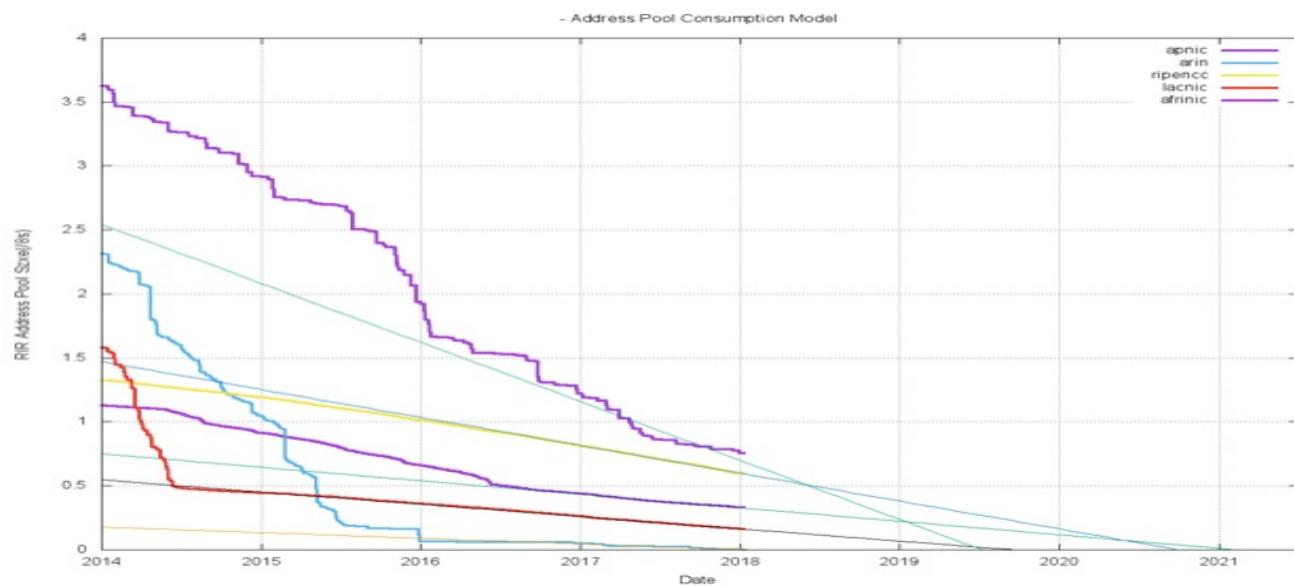


Latin America and Caribbean Network Information Centre



African Network Information Centre

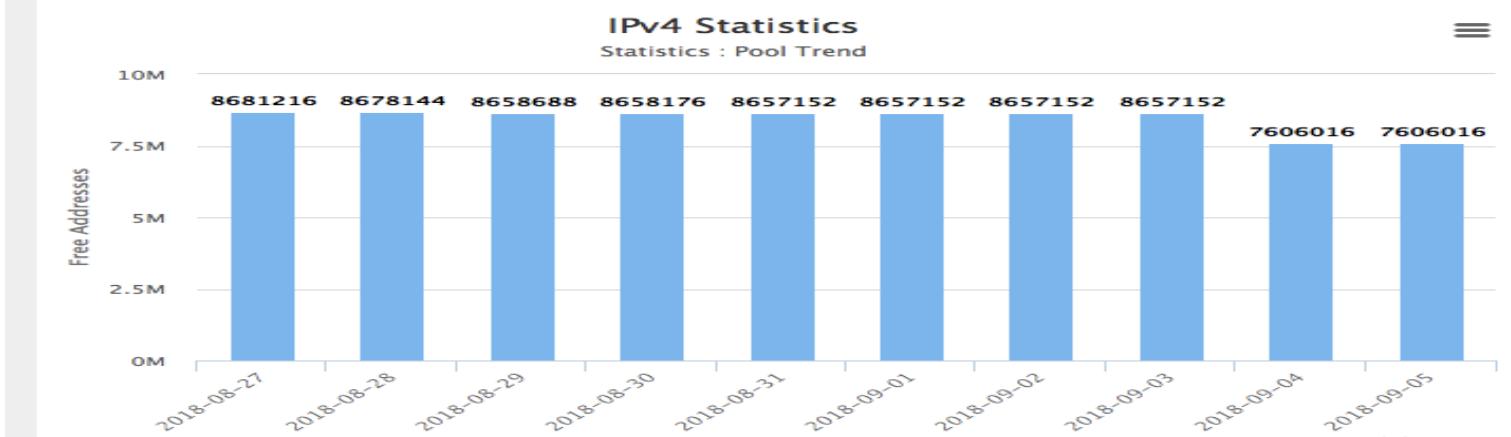




## IPv4 Exhaustion

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

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



# Made-up Example

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

# COMP 3331/9331: Computer Networks and Applications

Week 8

Network Layer: Data Plane (contd.)

Reading Guide: Chapter 4: 4.3

# Network Layer, data plane: outline

## 4.1 Overview of Network layer

- data plane
- control plane

## 4.2 What's inside a router

## 4.3 IP: Internet Protocol

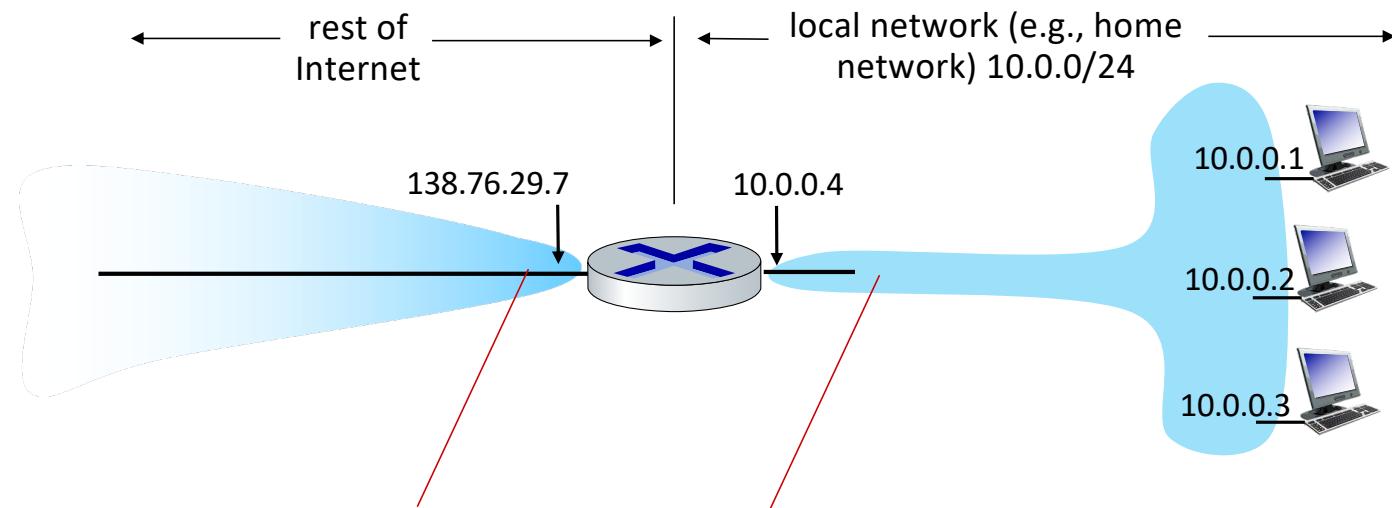
- datagram format
- fragmentation
- IPv4 addressing
- **network address translation**
- IPv6

# Private Addresses

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

# 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

- all devices in local network have 32-bit addresses in a “private” IP address space (10/8, 172.16/12, 192.168/16 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

# NAT: network address translation

**implementation:** NAT router must (transparently):

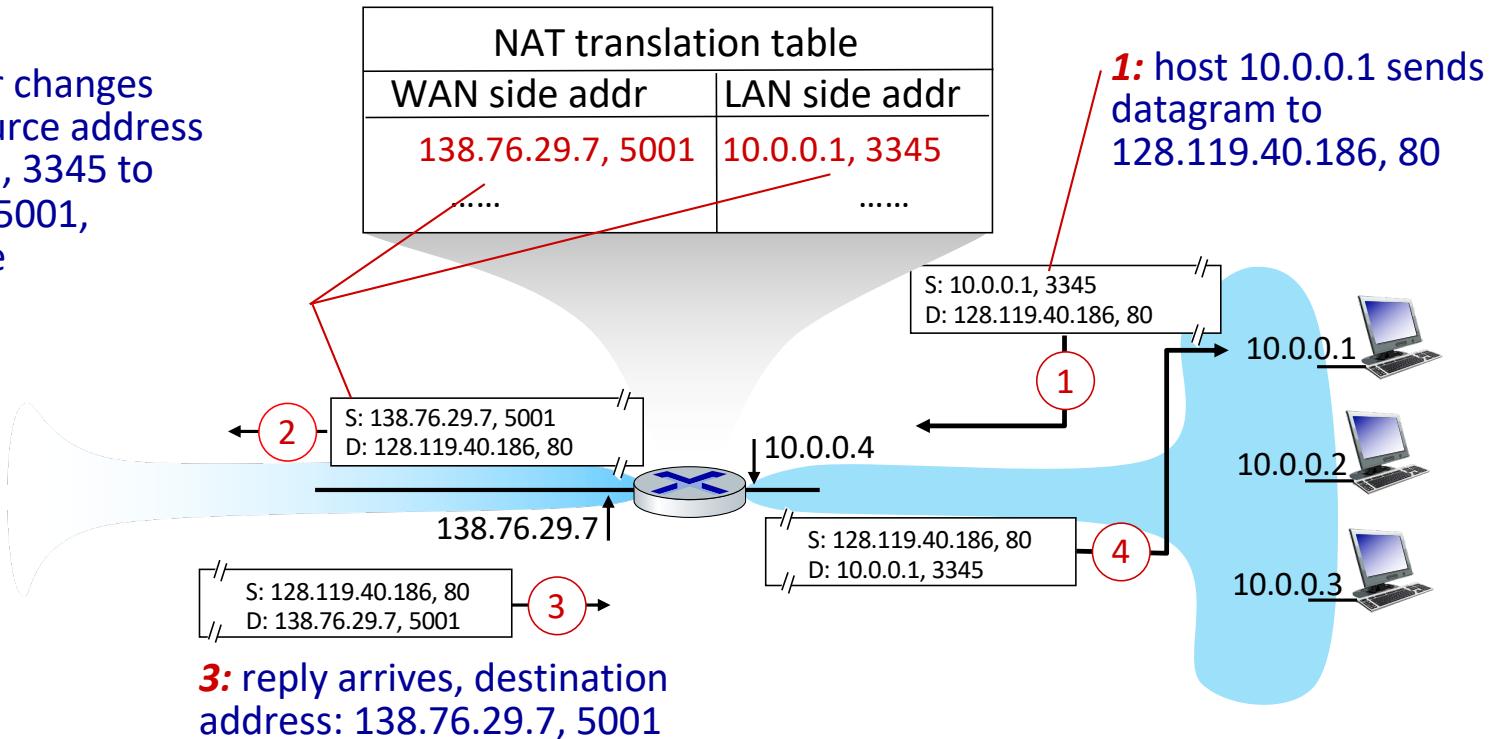
- **outgoing datagrams:** replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - remote clients/servers will respond using (NAT IP address, new port #) as destination address
- **remember (in NAT translation table)** every (source IP address, port #) to (NAT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

# NAT: network address translation

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

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

**1:** host 10.0.0.1 sends datagram to 128.119.40.186, 80



**3:** reply arrives, destination address: 138.76.29.7, 5001

# NAT: network address translation

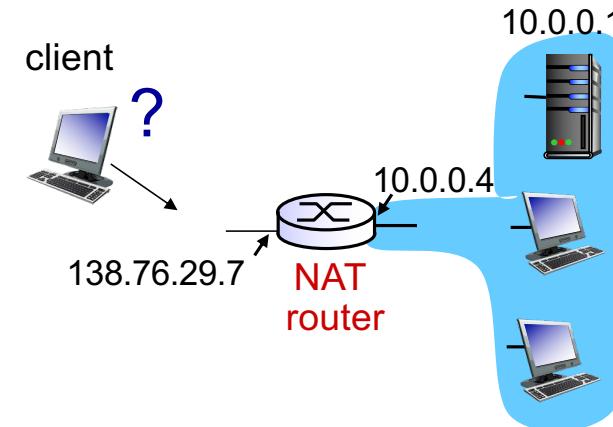
- NAT has been controversial:
  - routers “should” only process up to layer 3
  - address “shortage” should be solved by IPv6
  - violates end-to-end argument (port # manipulation by network-layer device)
  - NAT traversal: what if client wants to connect to server behind NAT?
- but NAT is here to stay:
  - extensively used in home and institutional nets, 4G/5G cellular nets

# NAT: Practical Issues

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

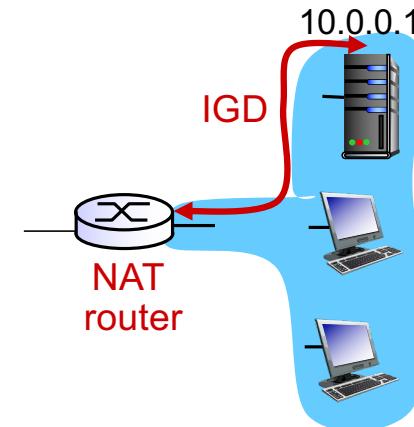
# NAT traversal problem

- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7
- *Solution1:* Inbound-NAT Statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



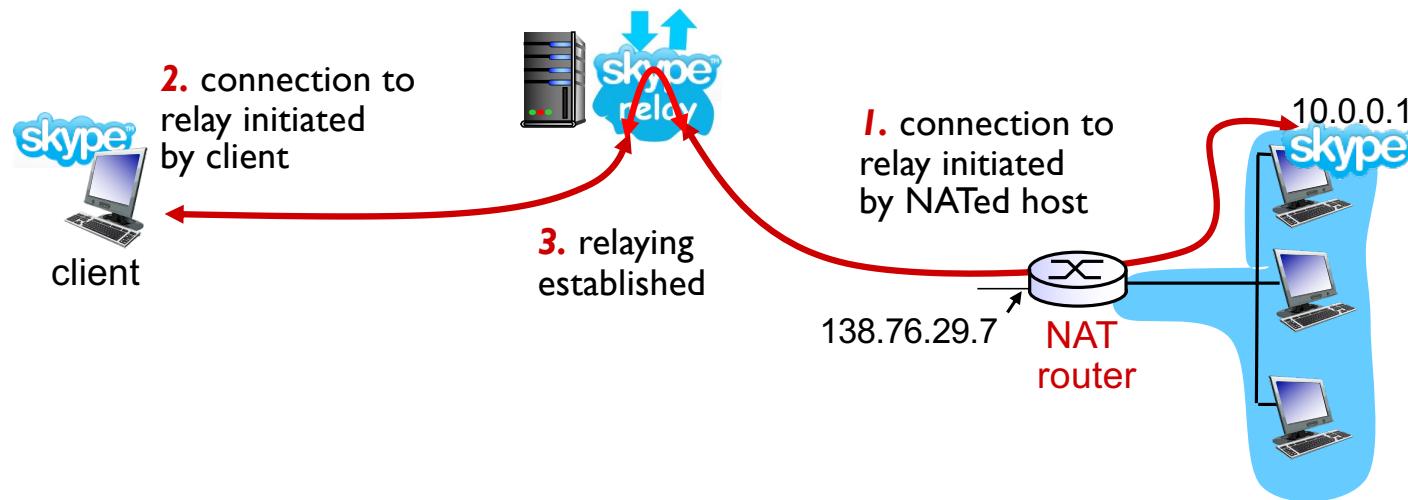
# NAT traversal problem

- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - ❖ learn public IP address (138.76.29.7)
  - ❖ add/remove port mappings (with lease times)i.e., automate static NAT port map configuration



# NAT traversal problem

- *solution 3:* relaying (used in Skype)
  - NATed client establishes connection to relay
  - external client connects to relay
  - relay bridges packets between connections



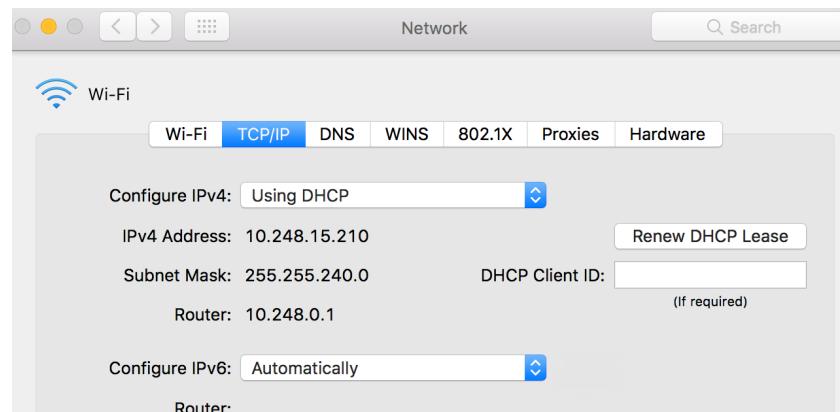
# NAT: Devil in the details

- Despite the problems, NAT has been widely deployed
- Most protocols can be successfully passed through a NAT, including VPN
- Modern hardware can easily perform NAT functions at > 100 Mbps
- IPv6 is still not widely deployed commercially, so the need for NAT is real
- After years of refusing to work on NAT, the IETF has been developing “NAT control protocols” for hosts
- Lot of practical variations
  - Full-cone NAT, Restricted Cone NAT, Port Restricted Cone NAT, Symmetric NAT, .....
  - The devil is in the detail (NOT COVERED IN THE COURSE)

# Quiz



- The picture below shows you the IP address of my machine connected to the uniwide wireless network.



- However when I ask Google it says my IP address is as noted below. Can you explain the discrepancy?

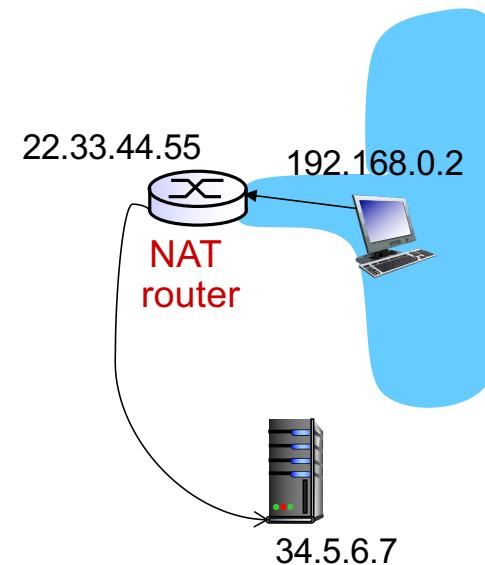
129.94.8.210  
Your public IP address



## Quiz: NAT

A host with a private IP address 192.168.0.2 opens a TCP socket on its local port 4567 and connects to a web server at 34.5.6.7. The NAT's public IP address is 22.33.44.55. Which of the following mapping entries *could* the NAT create as a result?

- A.[22.33.44.55, 4567] → [192.168.0.2, 80]
- B.[34.5.6.7, 80] → [22.33.44.55, 4567]
- C.[192.168.0.2, 80] → [34.5.6.7, 4567]
- D.[22.33.44.55, 3967] → [192.168.0.2, 4567]

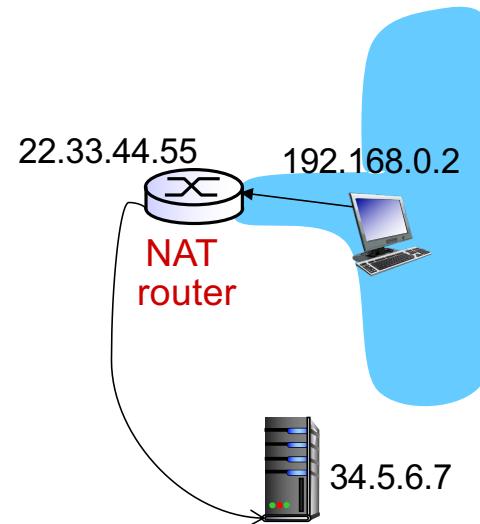




# Quiz: NAT

A host with a private IP address 192.168.0.2 opens a TCP socket on its local port 4567 and connects to a web server at 34.5.6.7. The NAT's public IP address is 22.33.44.55. Suppose the NAT created the mapping [22.33.44.55, 3967]→[192.168.0.2, 4567] as a result. What are the source and destination port numbers in the SYN-ACK response from the server?

- A.80, 3967
- B.4567, 80
- C.3967, 80
- D.3967, 4567
- E.80, 4567



COMP 3331/9331:  
Computer Networks and  
Applications

- 1. Intra- vs. inter-domain routing
- 2. Link state routing
- 3. Distance vector routing

Week 8  
Control Plane (Routing)

**Chapter 5: Section 5.1 – 5.2, 5.6**

# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- ❖ link state
- ❖ distance vector
- ❖ Hierarchical routing (NOT ON EXAM)

5.6 ICMP: The 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*

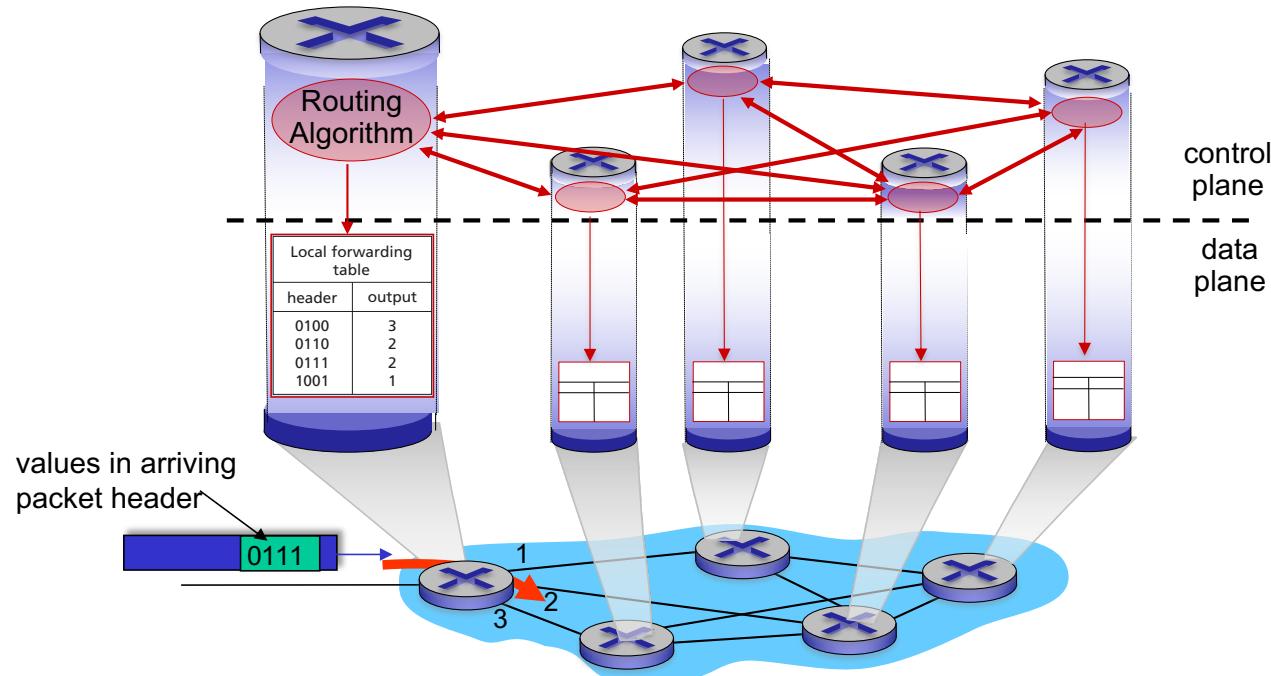
*control plane*

**Two approaches to structuring network control plane:**

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

# Per-router control plane

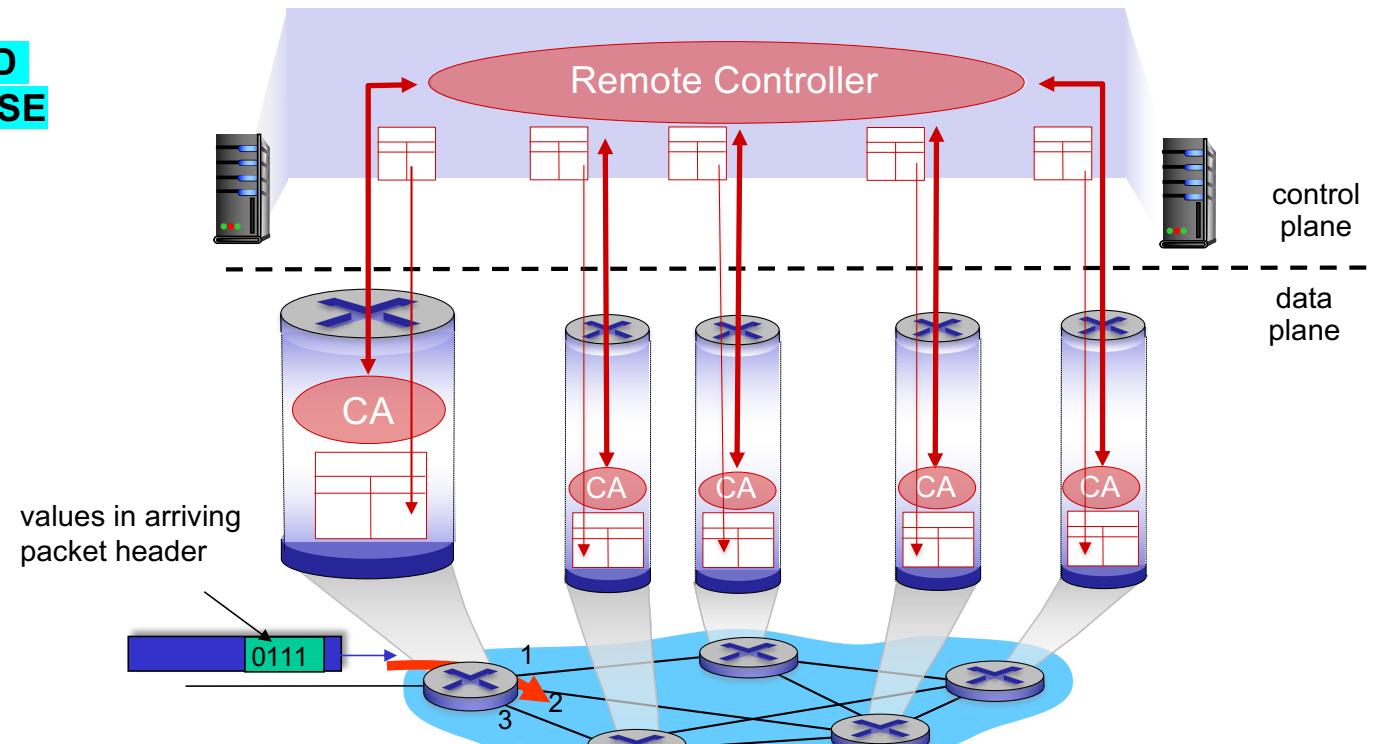
Individual routing algorithm components *in each and every router* interact in the control plane



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers

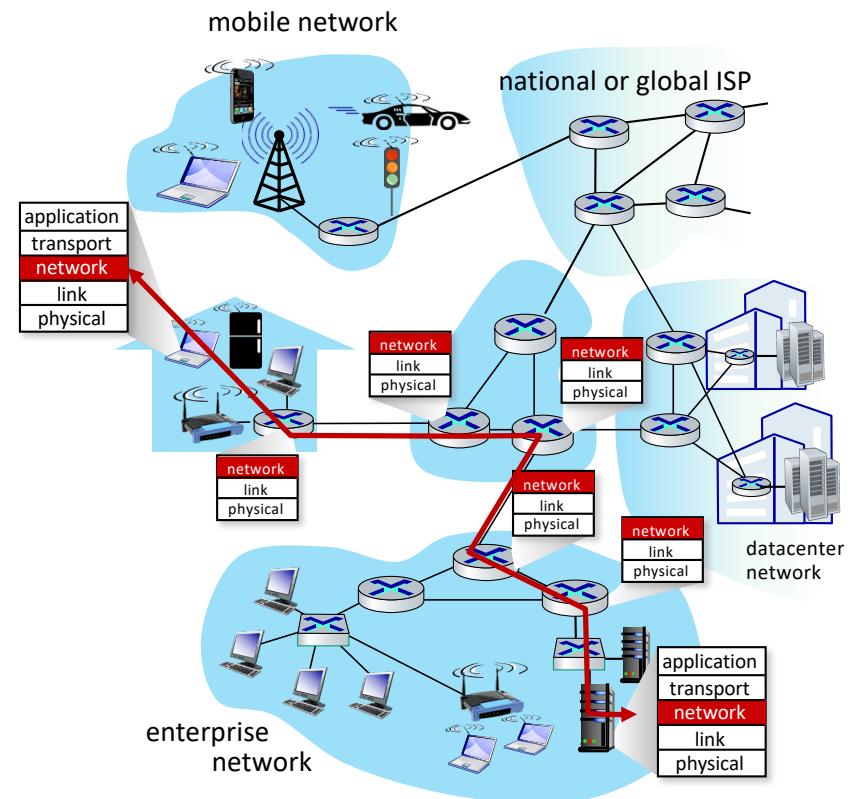
**NOT COVERED  
IN THIS COURSE**



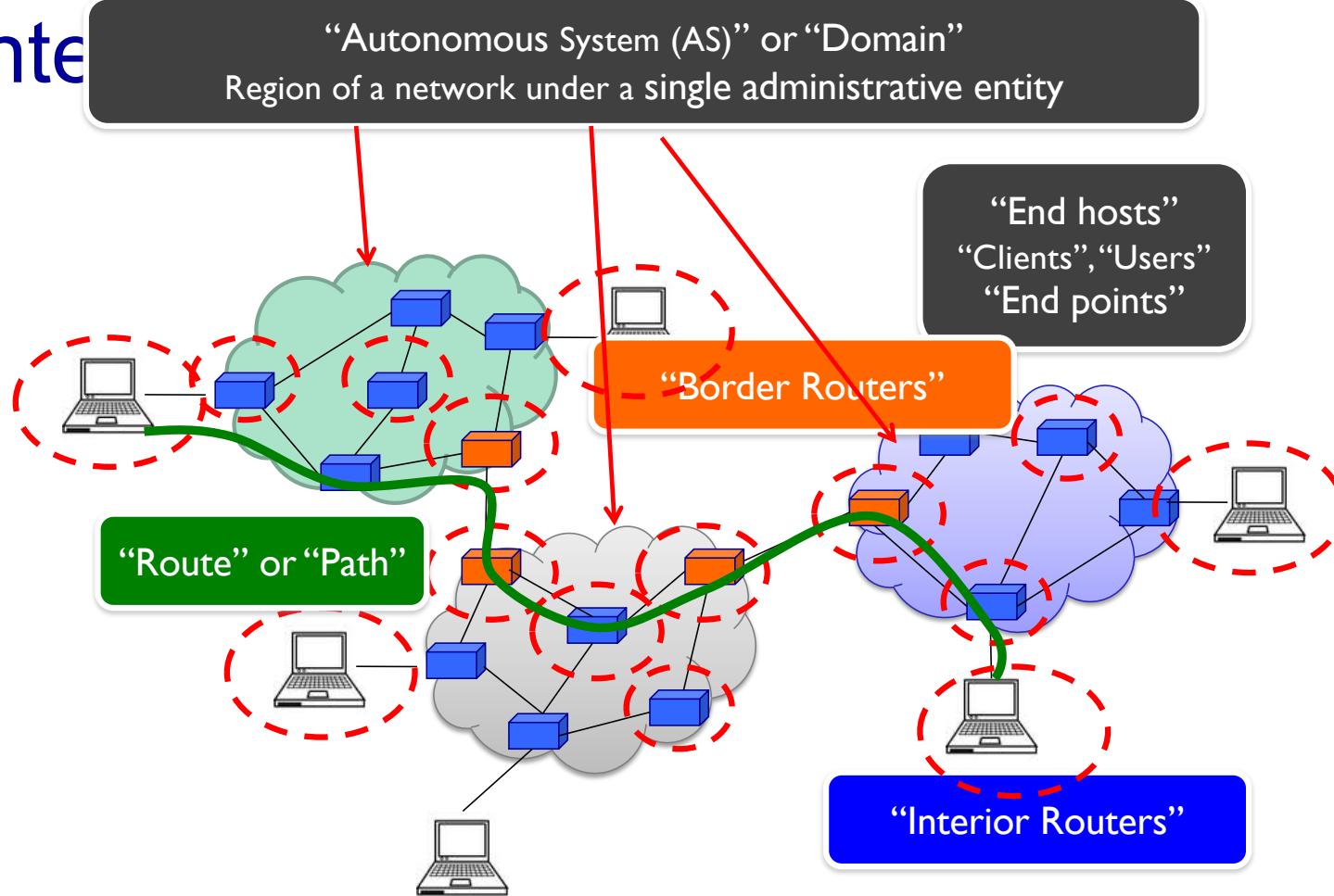
# Routing protocols

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

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



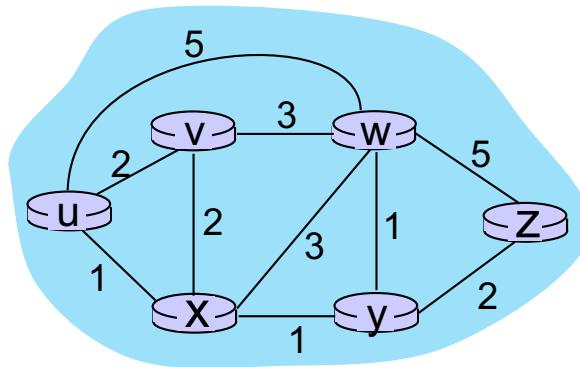
# Content



# Internet Routing

- ❖ Internet Routing works at two levels
- ❖ Each AS runs an **intra-domain** routing protocol that establishes routes within its domain
  - AS -- region of network under a single administrative entity
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)
- ❖ ASes participate in an **inter-domain** routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)

# Graph abstraction: link costs



graph:  $G = (N, E)$

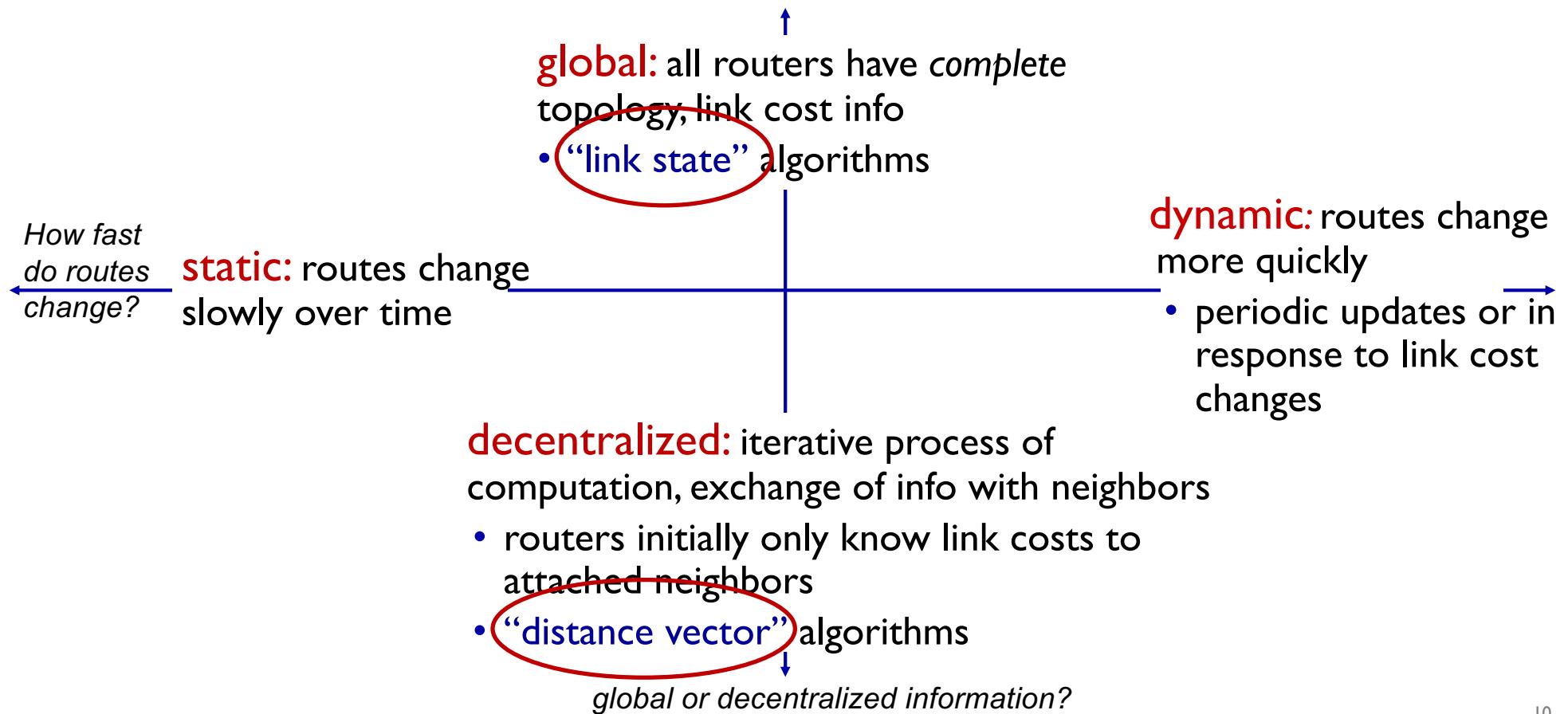
$N$ : set of routers = {  $u, v, w, x, y, z$  }

$E$ : 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



# Network layer, control plane: outline

5.1 introduction

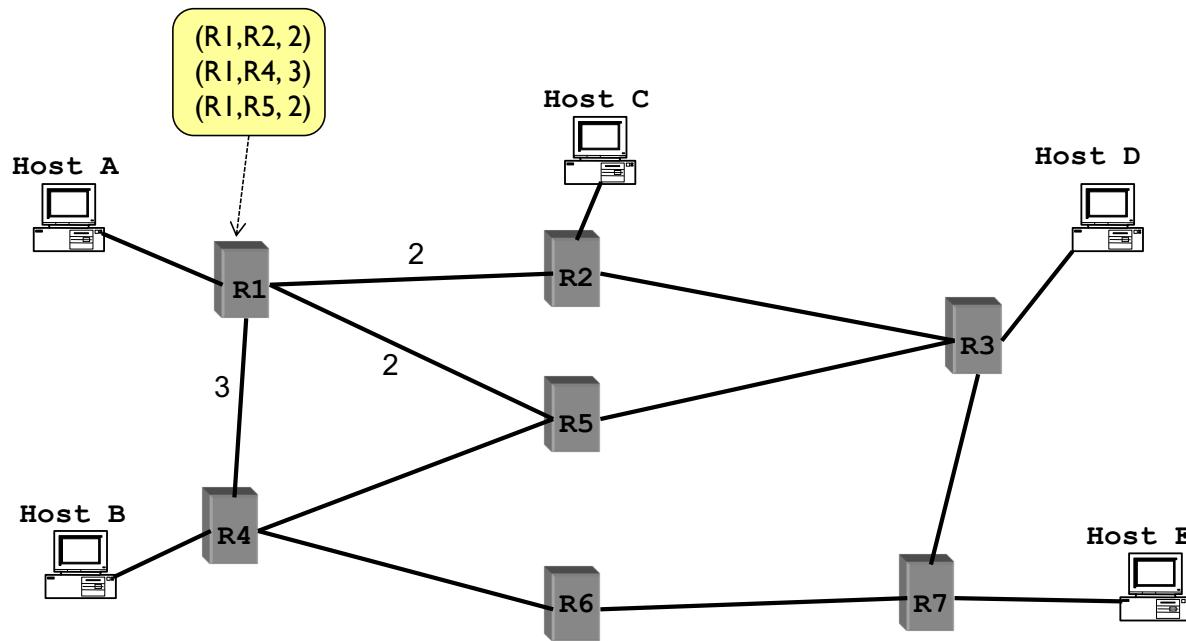
5.2 routing protocols

- ❖ link state
- ❖ distance vector
- ❖ hierarchical routing

5.6 ICMP: The Internet Control  
Message Protocol

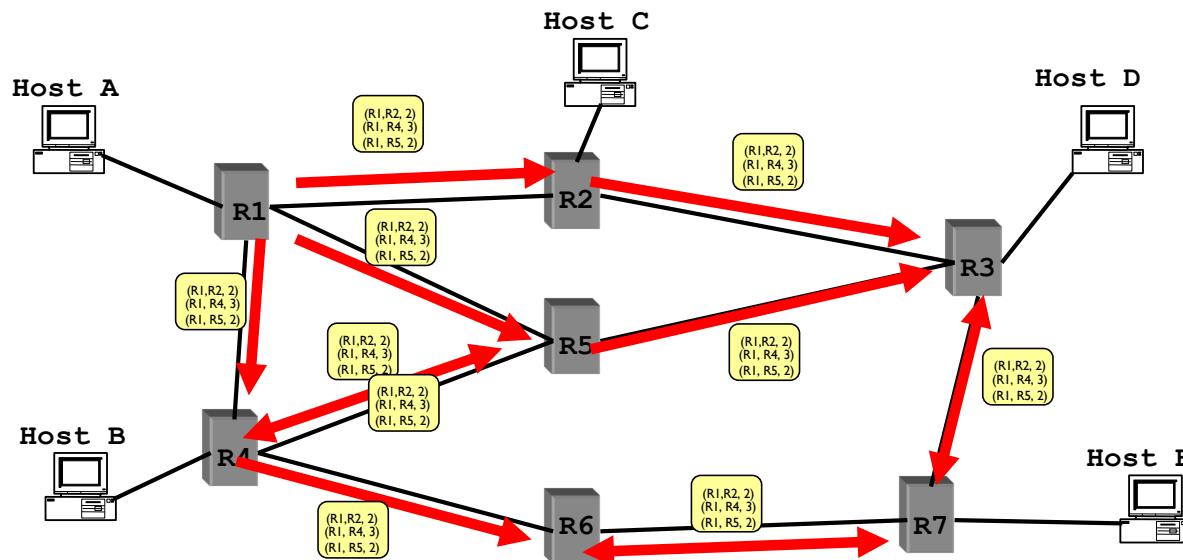
# Link State Routing

- ❖ Each node maintains its **local** “link state” (LS)
  - i.e., a list of its directly attached links and their costs



# Link State Routing

- ❖ Each node maintains its local “link state” (LS)
- ❖ Each node floods its local link state
  - on receiving a **new** LS message, a router forwards the message to all its neighbors other than the one it received the message from

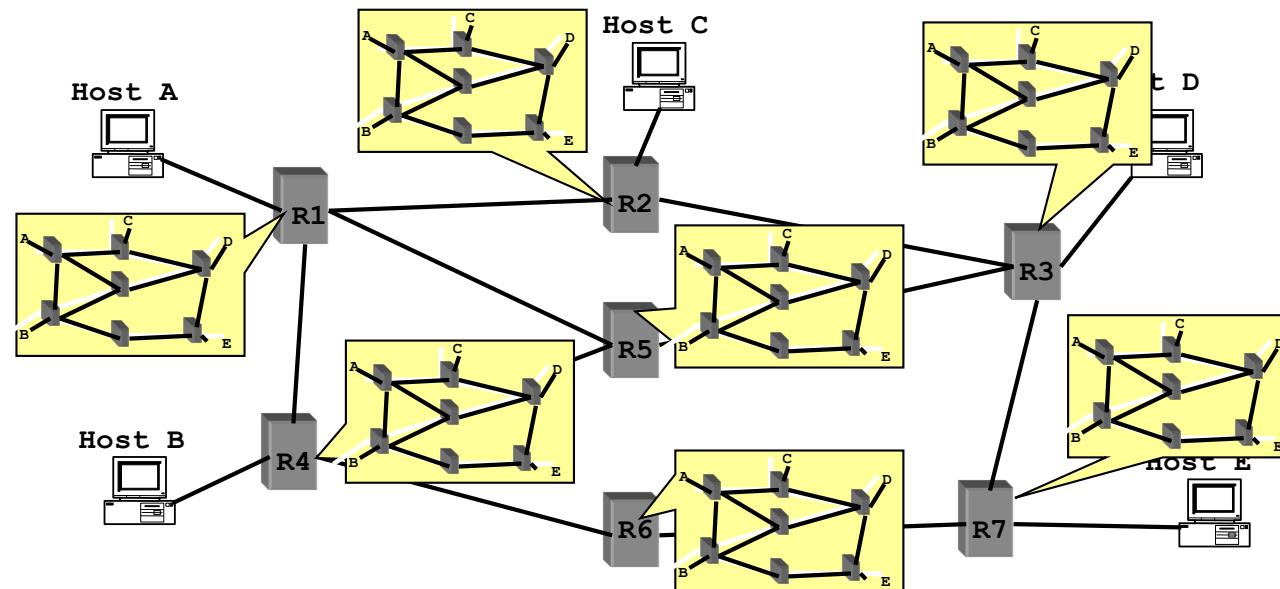


# Flooding LSAs

- ❖ Routers transmit **Link State Advertisement (LSA)** on links
  - A neighbouring router forwards out on all links except incoming
  - Keep a copy locally; don't forward previously-seen LSAs
- ❖ Challenges
  - Packet loss
  - Out of order arrival
- ❖ Solutions
  - Acknowledgements and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

# Link State Routing

- ❖ Each node maintains its local “link state” (LS)
- ❖ Each node floods its local link state
- ❖ Eventually, each node learns the entire network topology
  - Can use Dijkstra’s to compute the shortest paths between nodes



# Dijkstra's link-state routing algorithm

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

## notation

- $c_{x,y}$ : direct link cost from node  $x$  to  $y$ ;  $= \infty$  if not direct neighbors
- $D(v)$ : current estimate of cost of least-cost-path from source to destination  $v$
- $p(v)$ : predecessor node along path from source to  $v$
- $N'$ : set of nodes whose least-cost-path *definitively* known

# Dijkstra's link-state routing algorithm

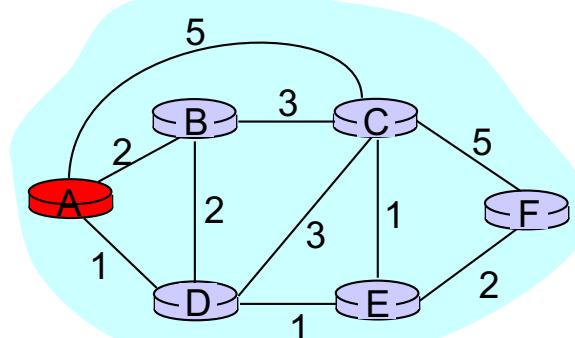
```
1 Initialization:
2    $N' = \{u\}$                                 /* compute least cost path from u to all other nodes */
3   for all nodes v
4     if v adjacent to u                      /* u initially knows direct-path-cost only to direct neighbors
*/
5       then  $D(v) = c_{u,v}$                   /* but may not be minimum cost!
*/
6     else  $D(v) = \infty$ 
7
8 Loop
9   find w not in  $N'$  such that  $D(w)$  is a minimum
10  add w to  $N'$ 
11  update  $D(v)$  for all v adjacent to w and not in  $N'$ :
    
$$D(v) = \min(D(v), D(w) + c_{w,v})$$

13  /* new least-path-cost to v is either old least-cost-path to v or known
14  least-cost-path to w plus direct-cost from w to v */
15 until all nodes in  $N'$ 
```



## Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1						
2						
3						
4						
5						



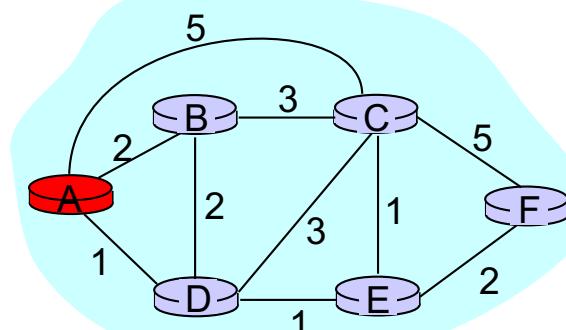
```

1 Initialization:
2    $N' = \{A\};$ 
3   for all nodes  $v$ 
4     if  $v$  adjacent to  $A$ 
5       then  $D(v) = c(A,v);$ 
6     else  $D(v) = \infty;$ 
...

```

## Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1						
2						
3						
4						
5						



...

8 **Loop**

9 find  $w$  not in  $N'$  s.t.  $D(w)$  is a minimum;

10 add  $w$  to  $N'$ ;

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$ :

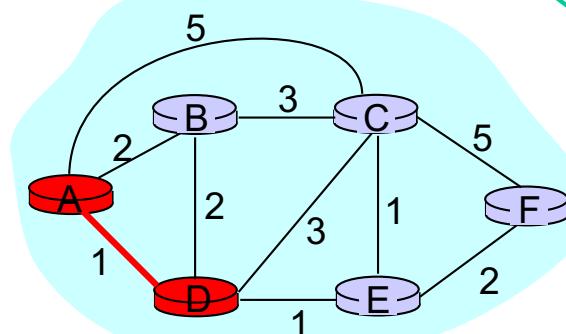
12 If  $D(w) + c(w,v) < D(v)$  then

13      $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;

14 until all nodes in  $N'$ ;

## Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD					
2						
3						
4						
5						



...

8 **Loop**

9 find  $w$  not in  $N'$  s.t.  $D(w)$  is a minimum;

10 add  $w$  to  $N'$ ,

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$ :

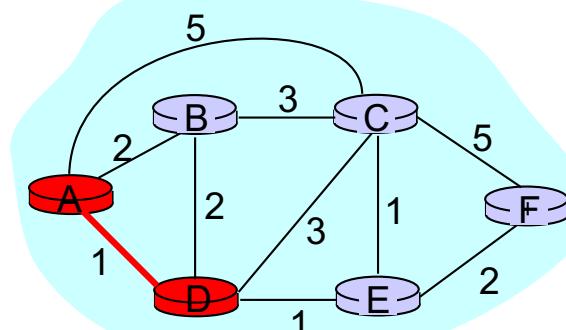
12 If  $D(w) + c(w,v) < D(v)$  then

13      $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;

14 until all nodes in  $N'$ ;

## Example: Dijkstra's Algorithm

Step	Set $N'$	$D(B), p(B)$	$D(C), p(C)$	$D(D), p(D)$	$D(E), p(E)$	$D(F), p(F)$
0	A	2, A	5, A	1, A	$\infty$	$\infty$
1	AD	2, A	4, D		2, D	$\infty$
2						
3						
4						
5						



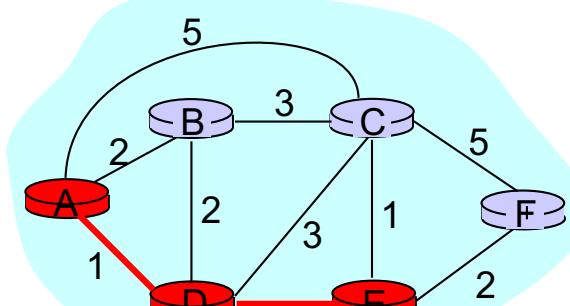
```

...
8  Loop
9  find w not in N' s.t. D(w) is a minimum;
10 add w to N';
11 update D(v) for all v adjacent
    to w and not in N':
12 If D(w) + c(w,v) < D(v) then
13   D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in N';

```

## Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2, A	4,D		2,D	$\infty$
2	ADE	2, A	3,E			4,E
3						
4						
5						



...

8 **Loop**

9 find  $w$  not in  $N'$  s.t.  $D(w)$  is a minimum;

10 add  $w$  to  $N'$ ;

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$ :

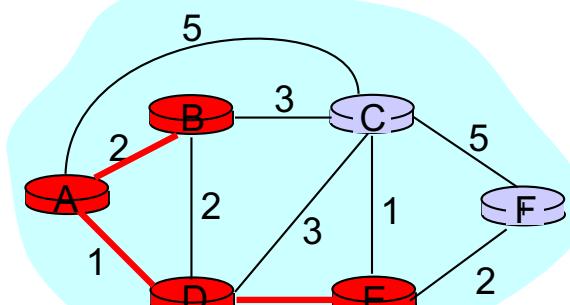
12 If  $D(w) + c(w,v) < D(v)$  then

13      $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;

14 **until all nodes in  $N'$ ;**

## Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4						
5						



...

8 **Loop**

9 find  $w$  not in  $N'$  s.t.  $D(w)$  is a minimum;

10 add  $w$  to  $N'$ ;

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$ :

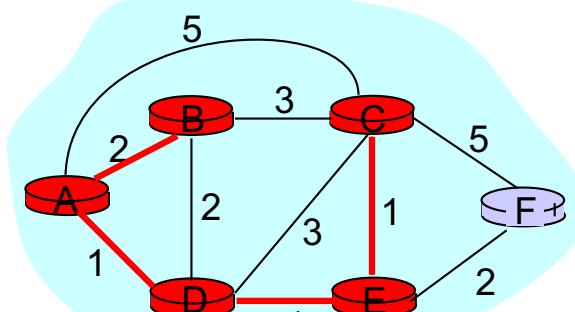
12 If  $D(w) + c(w,v) < D(v)$  then

13      $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;

14 **until all nodes in  $N'$ ;**

# Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E		4,E	
3	ADEB		3,E		4,E	
4	ADEBC				4,E	
5						



...

8 **Loop**

9 find  $w$  not in  $N'$  s.t.  $D(w)$  is a minimum;

10 add  $w$  to  $N'$ ;

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$ :

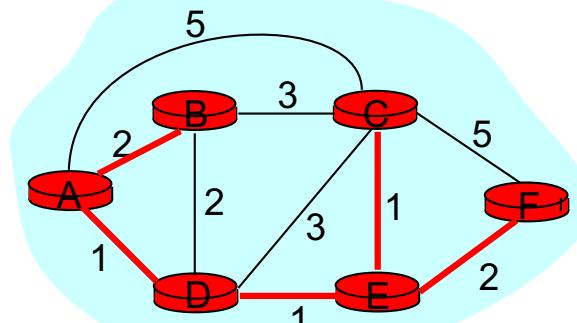
12 If  $D(w) + c(w,v) < D(v)$  then

13      $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;

14 **until all nodes in  $N'$** ;

## Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
→5	ADEBCF					



...

8 **Loop**

9 find  $w$  not in  $N'$  s.t.  $D(w)$  is a minimum;

10 add  $w$  to  $N'$ ;

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$ :

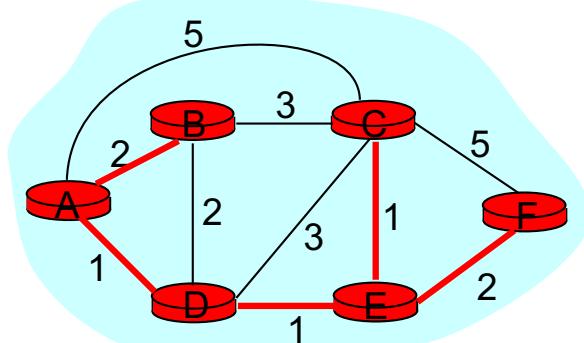
12 If  $D(w) + c(w,v) < D(v)$  then

13      $D(v) = D(w) + c(w,v)$ ;  $p(v) = w$ ;

14 until all nodes in  $N'$ ;

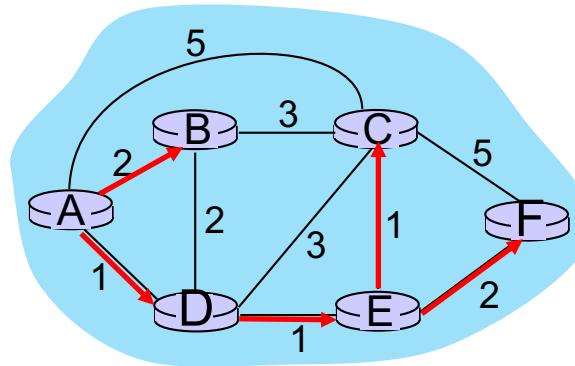
## Example: Dijkstra's Algorithm

Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	$\infty$	$\infty$
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					

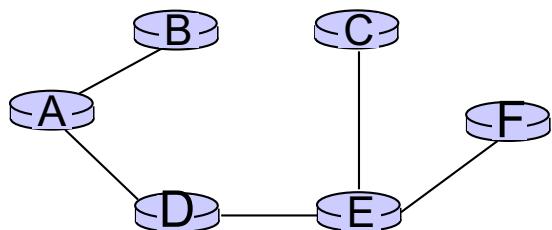


To determine path  $A \rightarrow C$  (say), work backward from C via  $p(v)$

## Example: Dijkstra's Algorithm



resulting least-cost-path tree from A:



resulting forwarding table in A:

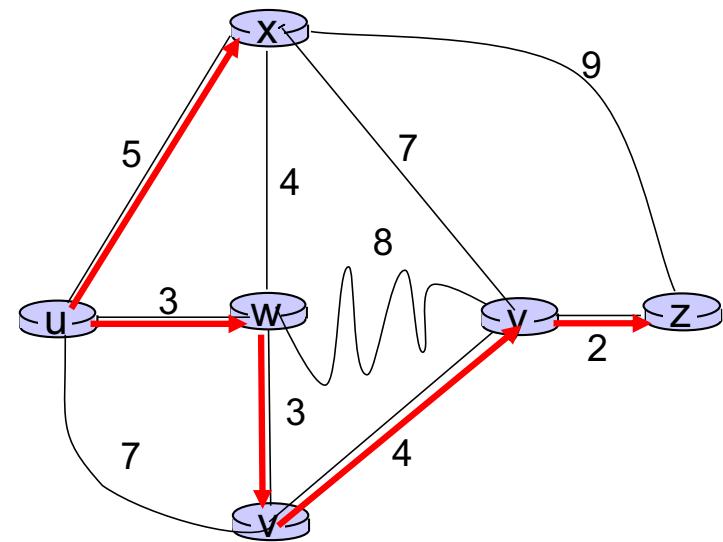
destination	outgoing link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

route from A to B directly

route from A to all other destinations via D

# Dijkstra's algorithm: another example

Step	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	7,u	3,u	5,u	$\infty$	$\infty$
1	uw	6,w	5,u	11,w	$\infty$	
2	uwx	6,w		11,w	14,x	
3	uwxv		10,v	14,x		
4	uwxvy			12,y		
5	uwxvyz					



notes:

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

# Dijkstra's algorithm: discussion

**algorithm complexity:**  $n$  nodes

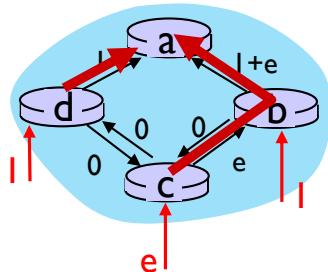
- each of  $n$  iteration: need to check all nodes,  $w$ , not in  $N$
- $n(n+1)/2$  comparisons:  $O(n^2)$  complexity
- more efficient implementations possible:  $O(n \log n)$

**message complexity:**

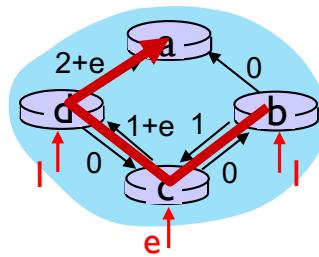
- each router must *broadcast* its link state information to other  $n$  routers
- efficient (and interesting!) broadcast algorithms:  $O(n)$  link crossings to disseminate a broadcast message from one source
- each router's message crosses  $O(n)$  links: overall message complexity:  $O(n^2)$

# Dijkstra's algorithm: oscillations possible

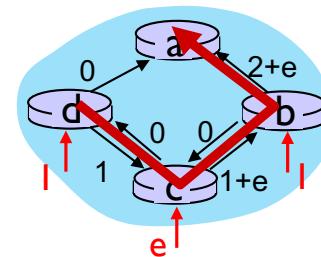
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates  $l, e (< l), l$
  - link costs are directional, and volume-dependent



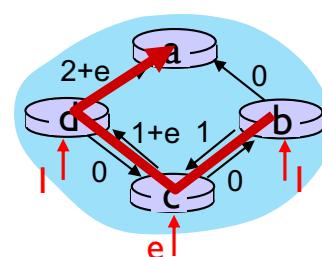
initially



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: outline

5.1 introduction

## 5.2 routing protocols

- ❖ link state
- ❖ **distance vector**
- ❖ hierarchical routing

5.6 ICMP: The Internet Control  
Message Protocol

# Distance vector algorithm

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:

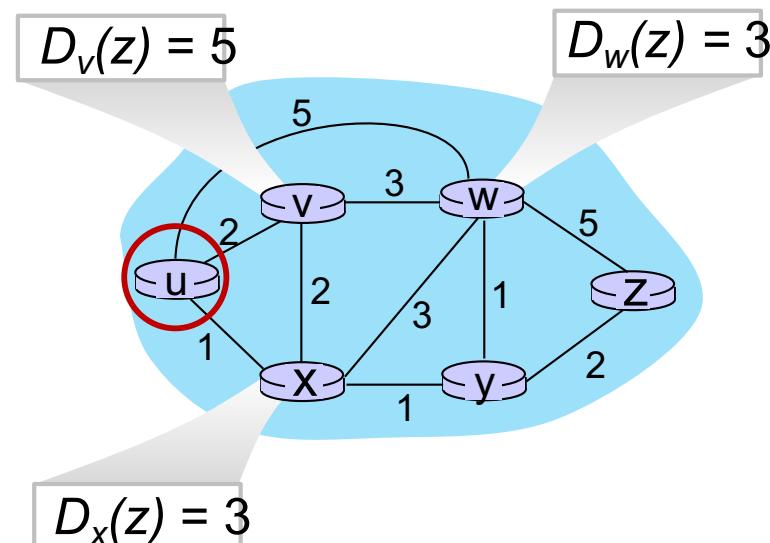
$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

$\min$  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 achieving minimum (x) is next hop on estimated 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[each node:] --> B[wait for (change in local link cost or DV from neighbor)]; B --> C[recompute DV estimates using DV received from neighbor]; C --> D;if DV to any destination has changed, notify neighbors
```
- wait for (change in local link cost or DV 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

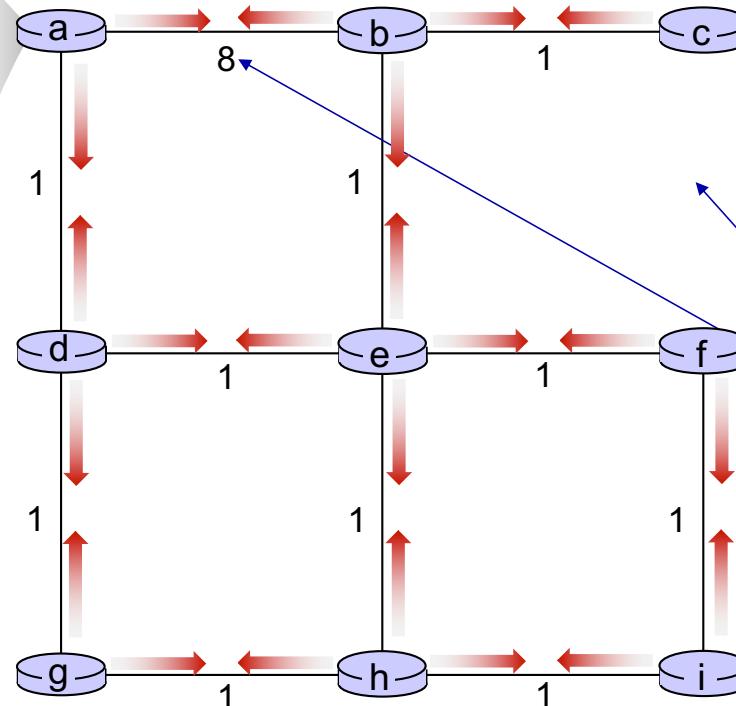
- neighbors then notify their neighbors – *only if necessary*
- no notification received; no actions taken!

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



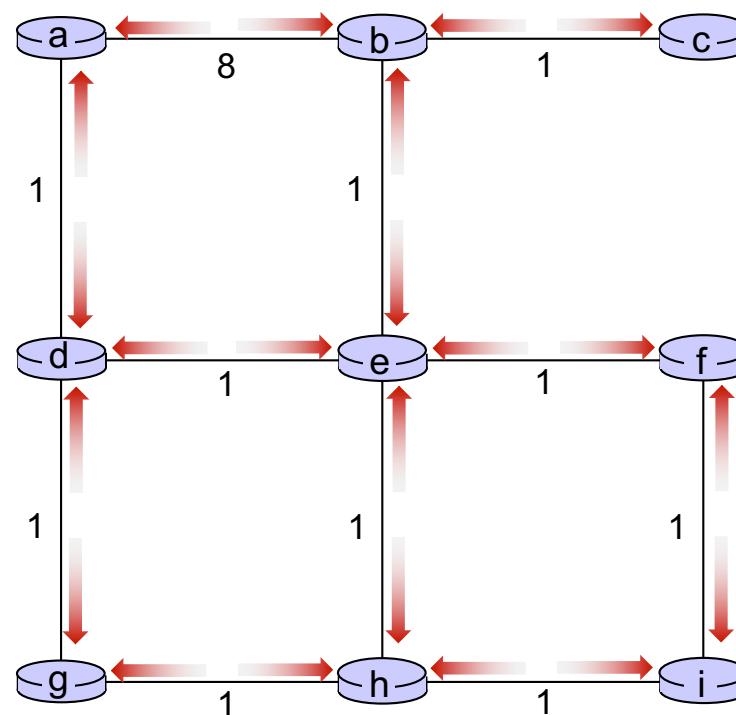
# Distance vector example: iteration



$t=1$

All nodes:

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



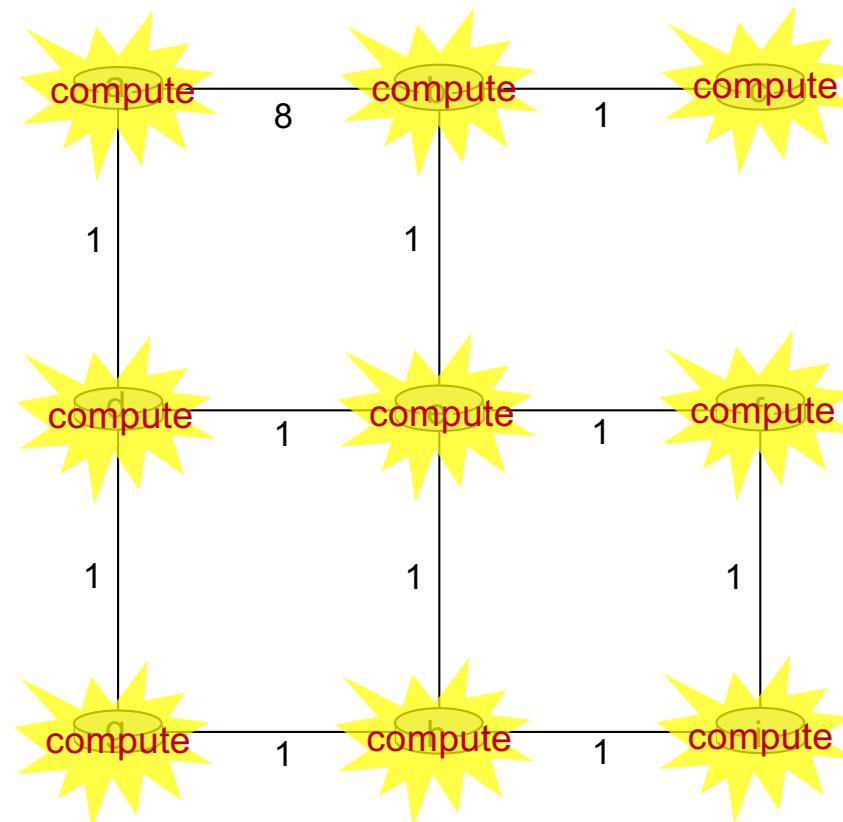
# Distance vector example: iteration



$t=1$

All nodes:

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



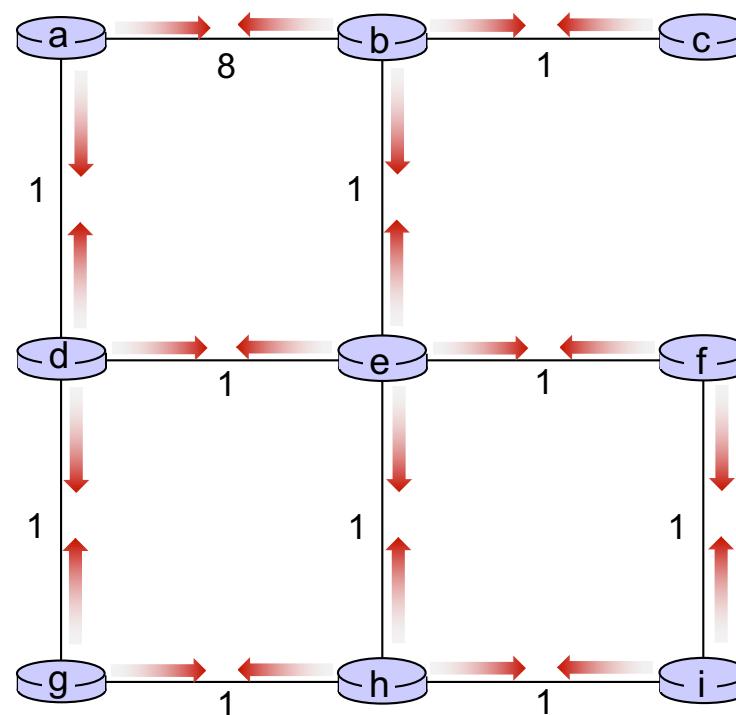
# Distance vector example: iteration



$t=1$

All nodes:

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



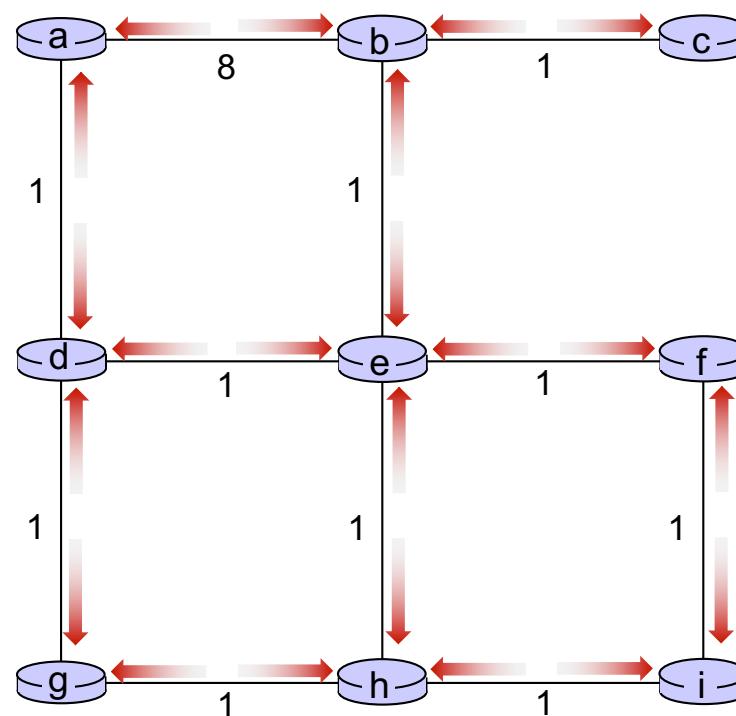
# Distance vector example: iteration



t=2

All nodes:

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



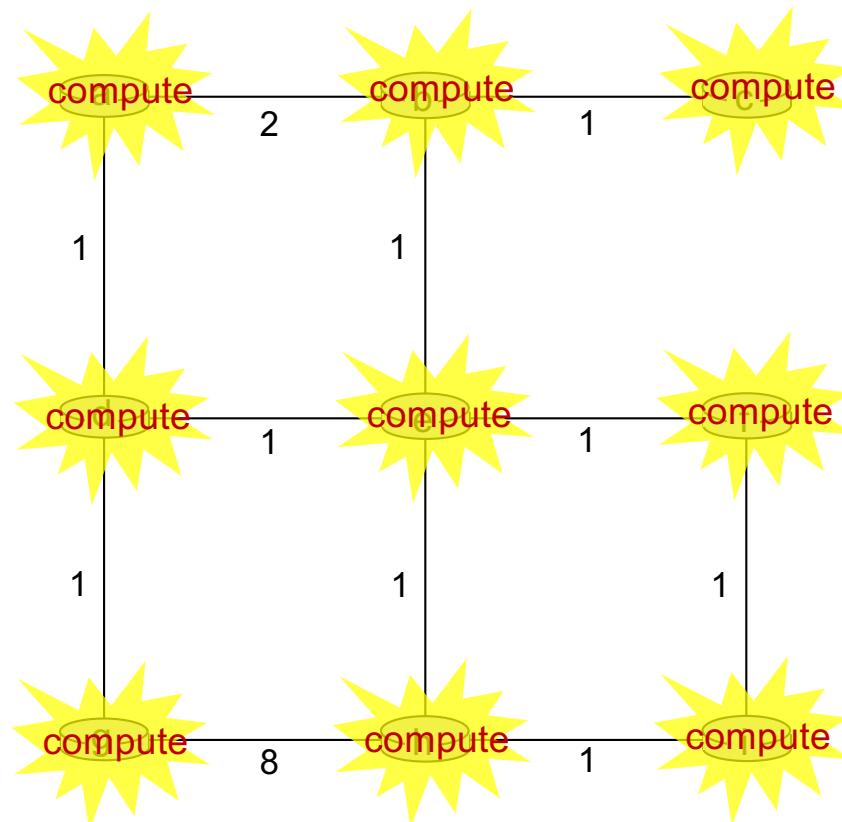
# Distance vector example: iteration



$t=2$

All nodes:

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



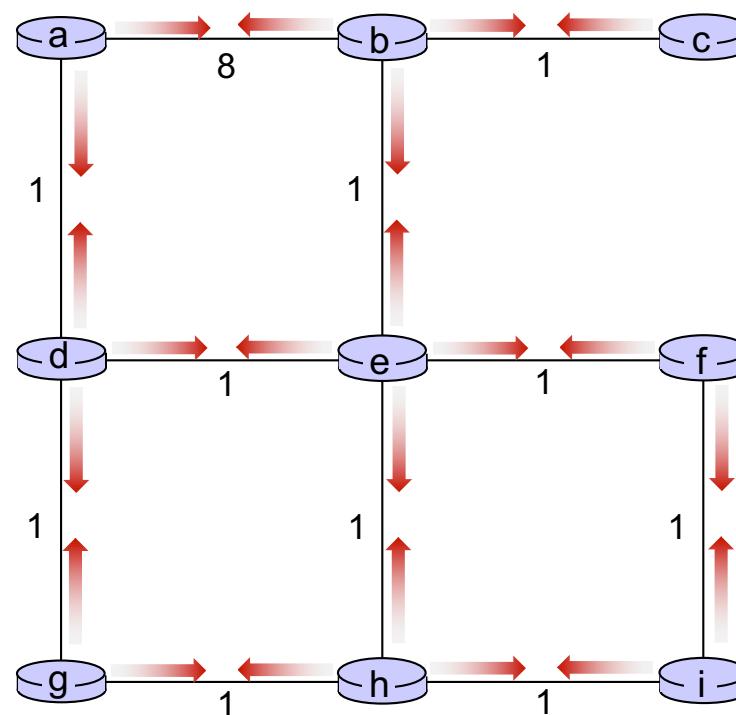
# Distance vector example: iteration



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

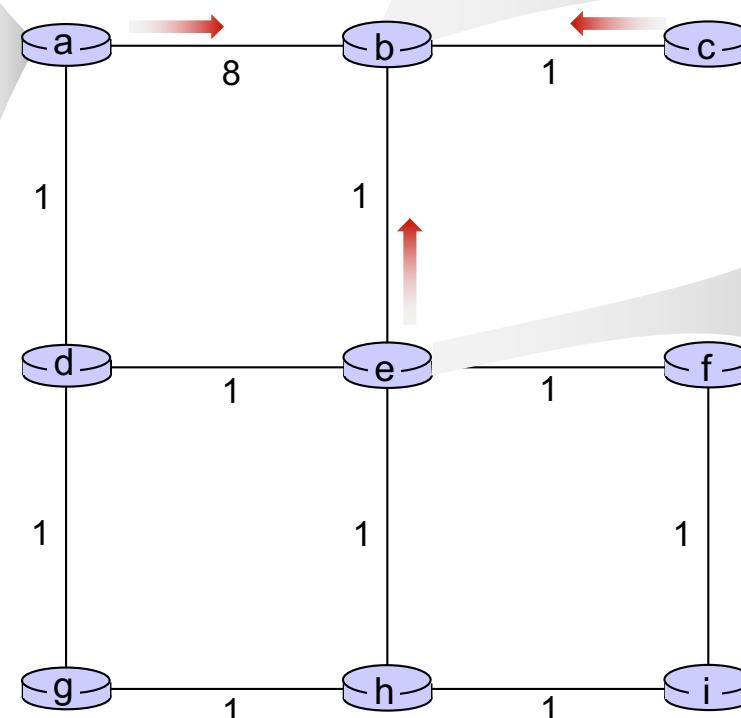
Let's next take a look at the iterative *computations* at nodes

# Distance vector example:



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

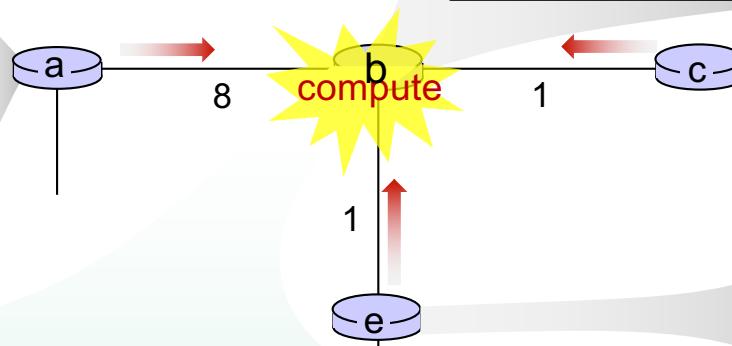
# Distance vector example: convergence

  
 $t=1$

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

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

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

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

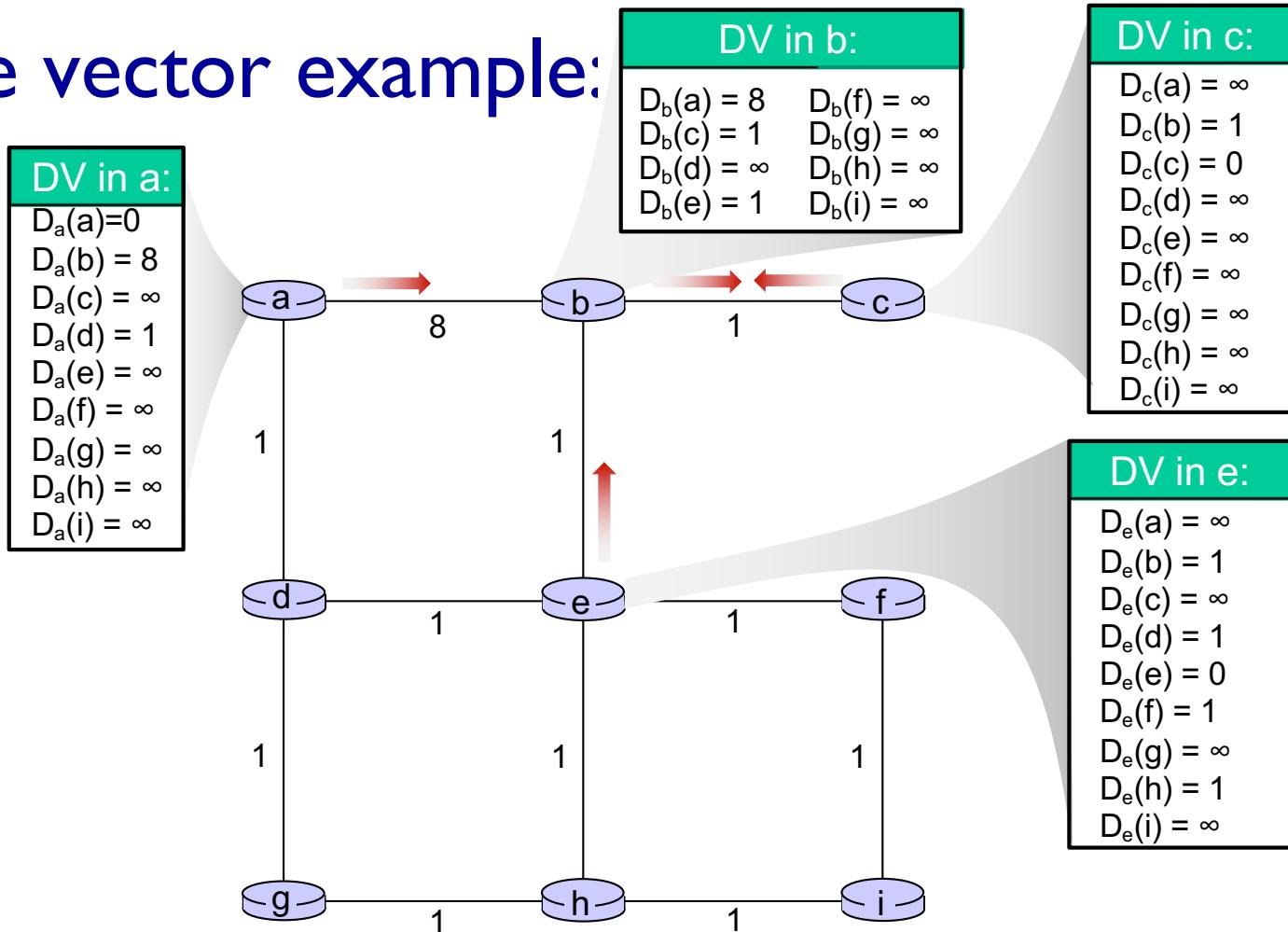
| DV in e:          |
|-------------------|
| $D_e(a) = \infty$ |
| $D_e(b) = 1$      |
| $D_e(c) = \infty$ |
| $D_e(d) = 1$      |
| $D_e(e) = 0$      |
| $D_e(f) = 1$      |
| $D_e(g) = \infty$ |
| $D_e(h) = 1$      |
| $D_e(i) = \infty$ |

| 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

- c receives DVs from b



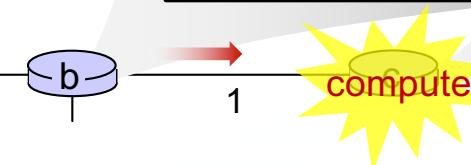
# Distance vector example:



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

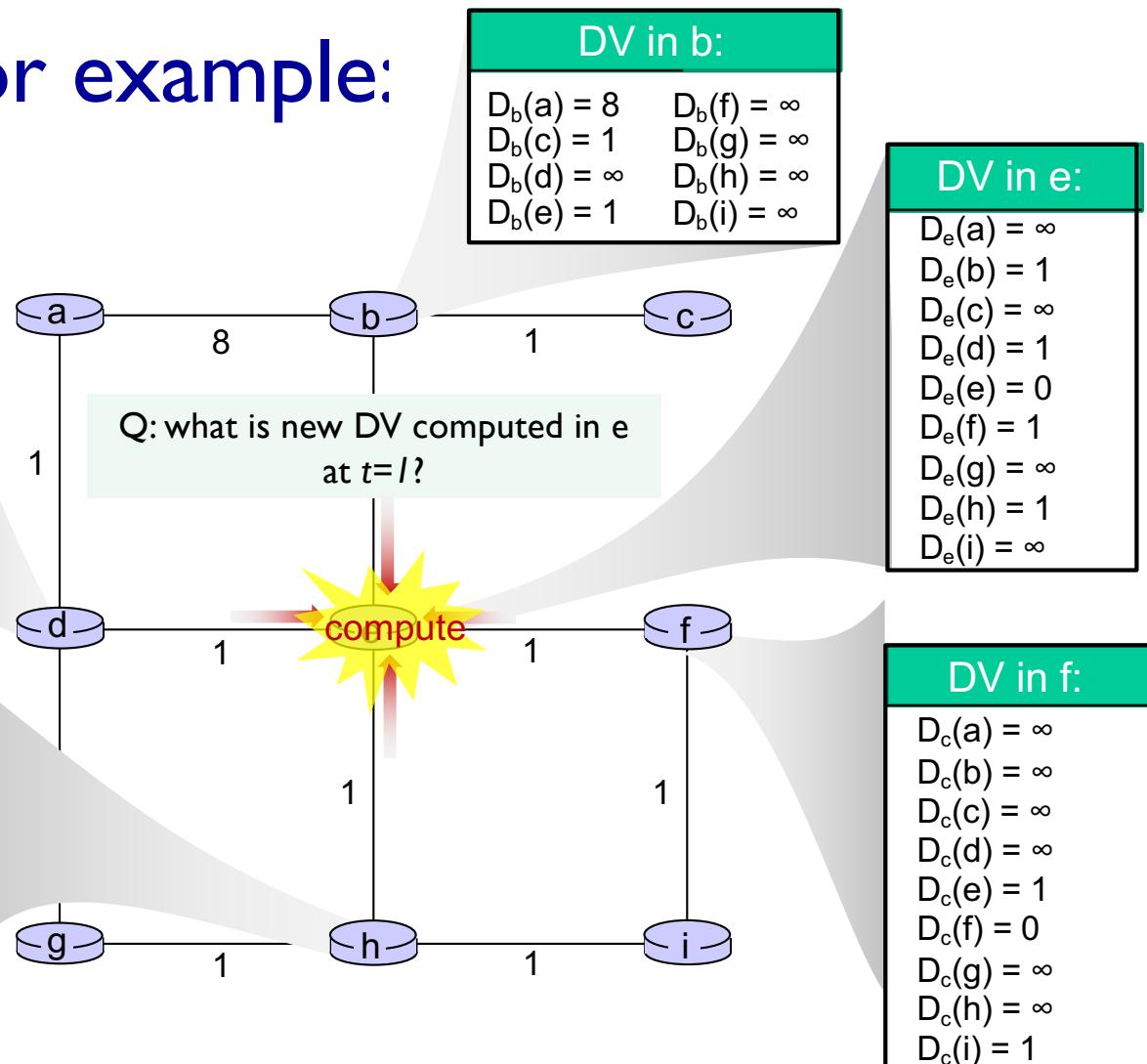
# Distance vector example:

 t=1

- e receives DVs from b, d, f, h

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

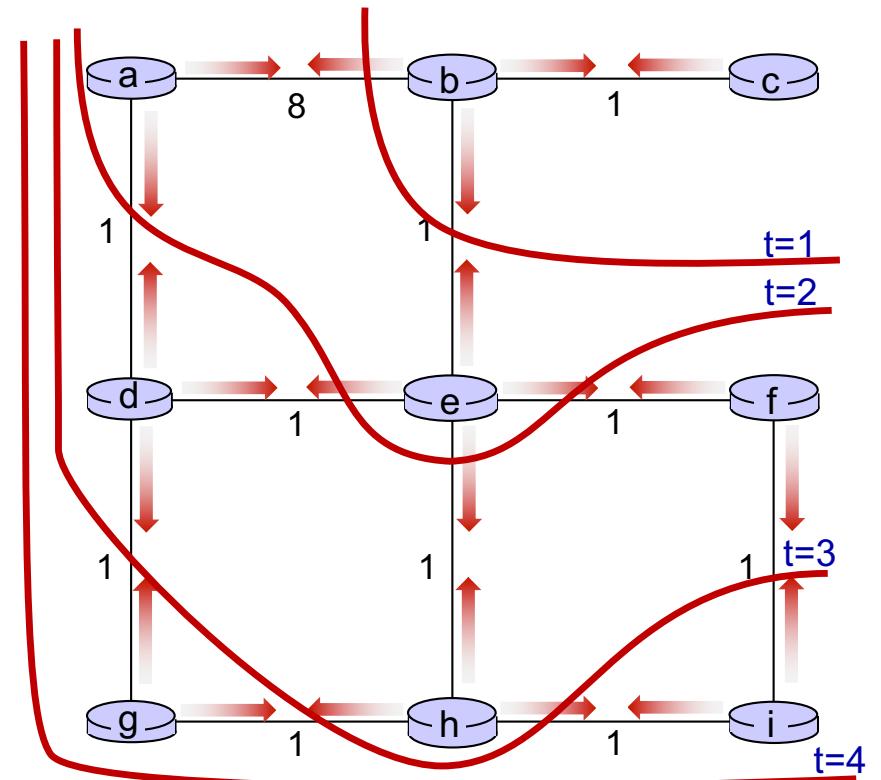
| DV in h:          |
|-------------------|
| $D_c(a) = \infty$ |
| $D_c(b) = \infty$ |
| $D_c(c) = \infty$ |
| $D_c(d) = \infty$ |
| $D_c(e) = 1$      |
| $D_c(f) = \infty$ |
| $D_c(g) = 1$      |
| $D_c(h) = 0$      |
| $D_c(i) = 1$      |



# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

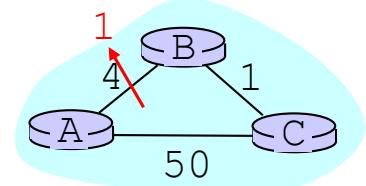
- ⌚ t=0 c's state at t=0 is at c only
- ⌚ t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
- ⌚ t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
- ⌚ t=3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



# Problems with Distance Vector

- A number of problems can occur in a network using distance vector algorithm
- Most of these problems are caused by slow convergence or routers converging on incorrect information
- **Convergence** is the time during which all routers come to an agreement about the best paths through the internetwork
  - whenever topology changes there is a period of instability in the network as the routers converge
- Reacts rapidly to good news, but leisurely to bad news

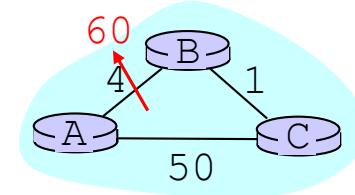
# DV: Link Cost Changes



NOTE: DIFFERENT REPRESENTATION FROM BEFORE. YELLOW ENTRIES ARE THE DV

|                                                         | Stable state<br>via                                                                                                                                  | A-B changed | A sends its DV to B, C | B sends its DV to A, C | C sends its DV to A, B |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|---------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|------------------------|------------------------|------------------------|----|----|---|----|----|------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|----|----|------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|----|---|------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|----|---|------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|----|----|
| Node A                                                  | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>4</td><td>51</td></tr> <tr><td>C</td><td>5</td><td>50</td></tr> </table> |             | B                      | C                      | B                      | 4  | 51 | C | 5  | 50 | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>1</td><td>51</td></tr> <tr><td>C</td><td>2</td><td>50</td></tr> </table> |  | B | C | B | 1  | 51 | C | 2  | 50 | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>1</td><td>51</td></tr> <tr><td></td><td></td><td></td></tr> </table>     |  | B | C | B | 1  | 51 |   |    |   | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>1</td><td>51</td></tr> <tr><td></td><td></td><td></td></tr> </table>     |  | B | C | B | 1  | 51 |   |    |   | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>1</td><td>51</td></tr> <tr><td></td><td></td><td>50</td></tr> </table>   |  | B | C | B | 1  | 51 |   |    | 50 |
|                                                         | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 4                                                                                                                                                    | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| C                                                       | 5                                                                                                                                                    | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 1                                                                                                                                                    | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| C                                                       | 2                                                                                                                                                    | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 1                                                                                                                                                    | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         |                                                                                                                                                      |             |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 1                                                                                                                                                    | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         |                                                                                                                                                      |             |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 1                                                                                                                                                    | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         |                                                                                                                                                      | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| Node B                                                  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>4</td><td>6</td></tr> <tr><td>C</td><td>9</td><td>1</td></tr> </table>   |             | A                      | C                      | A                      | 4  | 6  | C | 9  | 1  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>1</td><td>6</td></tr> <tr><td>C</td><td>6</td><td>1</td></tr> </table>   |  | A | C | A | 1  | 6  | C | 6  | 1  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>3</td><td>6</td></tr> <tr><td>C</td><td>3</td><td>1</td></tr> </table>   |  | A | C | A | 3  | 6  | C | 3  | 1 | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>3</td><td>6</td></tr> <tr><td>C</td><td>3</td><td>1</td></tr> </table>   |  | A | C | A | 3  | 6  | C | 3  | 1 | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>3</td><td>6</td></tr> <tr><td>C</td><td>3</td><td>1</td></tr> </table>   |  | A | C | A | 3  | 6  | C | 3  | 1  |
|                                                         | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 4                                                                                                                                                    | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| C                                                       | 9                                                                                                                                                    | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 1                                                                                                                                                    | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| C                                                       | 6                                                                                                                                                    | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 3                                                                                                                                                    | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| C                                                       | 3                                                                                                                                                    | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 3                                                                                                                                                    | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| C                                                       | 3                                                                                                                                                    | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 3                                                                                                                                                    | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| C                                                       | 3                                                                                                                                                    | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| Node C                                                  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table> |             | A                      | B                      | A                      | 50 | 5  | B | 54 | 1  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table> |  | A | B | A | 50 | 5  | B | 54 | 1  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>51</td><td>1</td></tr> </table> |  | A | B | A | 50 | 5  | B | 51 | 1 | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>2</td></tr> <tr><td>B</td><td>51</td><td>1</td></tr> </table> |  | A | B | A | 50 | 2  | B | 51 | 1 | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>2</td></tr> <tr><td>B</td><td>51</td><td>1</td></tr> </table> |  | A | B | A | 50 | 2  | B | 51 | 1  |
|                                                         | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 50                                                                                                                                                   | 5           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 54                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 50                                                                                                                                                   | 5           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 54                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 50                                                                                                                                                   | 5           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 51                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 50                                                                                                                                                   | 2           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 51                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
|                                                         | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| A                                                       | 50                                                                                                                                                   | 2           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| B                                                       | 51                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| <span style="color: red;">Link cost changes here</span> |                                                                                                                                                      |             |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |
| <b>“good news travels fast”</b>                         |                                                                                                                                                      |             |                        |                        |                        |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |   |                                                                                                                                                      |  |   |   |   |    |    |   |    |    |

# DV: Link Cost Changes



Stable state      A-B changed

| via       |     |     |
|-----------|-----|-----|
| Node A    |     | B C |
| <b>to</b> |     |     |
|           | B 4 | 51  |
|           | C 5 | 50  |

| Node B |     | A C |
|--------|-----|-----|
|        | A 4 | 6   |
|        | C 9 | 1   |

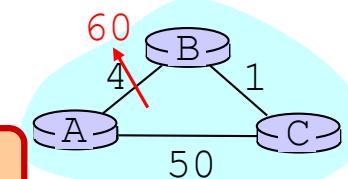
| Node C | A B    |  |
|--------|--------|--|
|        | A 50 5 |  |
|        | B 54 1 |  |

Link cost changes here ↑

add 56 to distances  
 $\text{dist}_B(A,*)$  and  $\text{dist}_A(B,*)$

# DV: Link Cost Changes

This is the “Counting to Infinity” Problem



|               | Stable state                                                                                                                                         | A-B changed | A sends its DV to B, C | B sends its DV to A, C | C sends its DV to A, B |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|------------------------|------------------------|------------------------|----|----|---|----|----|--------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|----|----|--------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|-----|----|--------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|-----|----|--------------------------------------------------------------------------------------------------------------------------------------------------------|--|---|---|---|----|----|---|-----|----|
|               | via                                                                                                                                                  |             |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| <b>Node A</b> | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>4</td><td>51</td></tr> <tr><td>C</td><td>5</td><td>50</td></tr> </table> |             | B                      | C                      | B                      | 4  | 51 | C | 5  | 50 | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table> |  | B | C | B | 60 | 51 | C | 61 | 50 | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table> |  | B | C | B | 60 | 51 | C | 61  | 50 | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table> |  | B | C | B | 60 | 51 | C | 61  | 50 | <table border="1"> <tr><td></td><td>B</td><td>C</td></tr> <tr><td>B</td><td>60</td><td>51</td></tr> <tr><td>C</td><td>61</td><td>50</td></tr> </table> |  | B | C | B | 60 | 51 | C | 61  | 50 |
|               | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 4                                                                                                                                                    | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 5                                                                                                                                                    | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 60                                                                                                                                                   | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 61                                                                                                                                                   | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 60                                                                                                                                                   | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 61                                                                                                                                                   | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 60                                                                                                                                                   | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 61                                                                                                                                                   | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | B                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 60                                                                                                                                                   | 51          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 61                                                                                                                                                   | 50          |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| <b>Node B</b> | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>4</td><td>6</td></tr> <tr><td>C</td><td>9</td><td>1</td></tr> </table>   |             | A                      | C                      | A                      | 4  | 6  | C | 9  | 1  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>6</td></tr> <tr><td>C</td><td>65</td><td>1</td></tr> </table>   |  | A | C | A | 60 | 6  | C | 65 | 1  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>6</td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>  |  | A | C | A | 60 | 6  | C | 110 | 1  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>6</td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>  |  | A | C | A | 60 | 6  | C | 110 | 1  | <table border="1"> <tr><td></td><td>A</td><td>C</td></tr> <tr><td>A</td><td>60</td><td>8</td></tr> <tr><td>C</td><td>110</td><td>1</td></tr> </table>  |  | A | C | A | 60 | 8  | C | 110 | 1  |
|               | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 4                                                                                                                                                    | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 9                                                                                                                                                    | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 60                                                                                                                                                   | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 65                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 60                                                                                                                                                   | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 110                                                                                                                                                  | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 60                                                                                                                                                   | 6           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 110                                                                                                                                                  | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | C           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 60                                                                                                                                                   | 8           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| C             | 110                                                                                                                                                  | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| <b>Node C</b> | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table> |             | A                      | B                      | A                      | 50 | 5  | B | 54 | 1  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>54</td><td>1</td></tr> </table>   |  | A | B | A | 50 | 5  | B | 54 | 1  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>5</td></tr> <tr><td>B</td><td>101</td><td>1</td></tr> </table>  |  | A | B | A | 50 | 5  | B | 101 | 1  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>7</td></tr> <tr><td>B</td><td>101</td><td>1</td></tr> </table>  |  | A | B | A | 50 | 7  | B | 101 | 1  | <table border="1"> <tr><td></td><td>A</td><td>B</td></tr> <tr><td>A</td><td>50</td><td>7</td></tr> <tr><td>B</td><td>101</td><td>1</td></tr> </table>  |  | A | B | A | 50 | 7  | B | 101 | 1  |
|               | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 50                                                                                                                                                   | 5           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 54                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 50                                                                                                                                                   | 5           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 54                                                                                                                                                   | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 50                                                                                                                                                   | 5           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 101                                                                                                                                                  | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 50                                                                                                                                                   | 7           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 101                                                                                                                                                  | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
|               | A                                                                                                                                                    | B           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| A             | 50                                                                                                                                                   | 7           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |
| B             | 101                                                                                                                                                  | 1           |                        |                        |                        |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |    |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |                                                                                                                                                        |  |   |   |   |    |    |   |     |    |

↑  
Link cost changes here

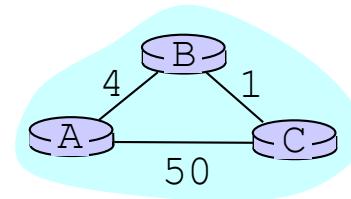
“bad news travels slowly”  
(not yet converged)

## The “Poisoned Reverse” Rule

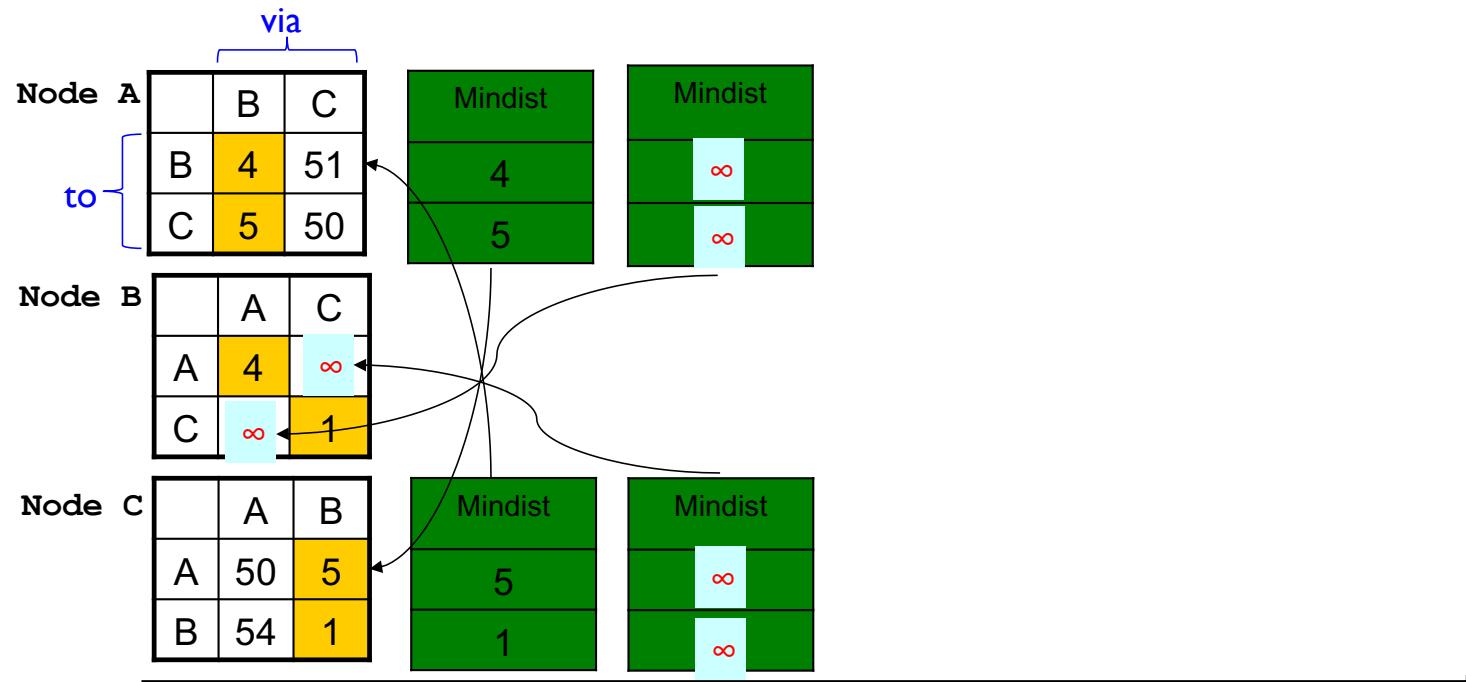
- ❖ Heuristic to avoid count-to-infinity
- ❖ If B routes via C to get to A:
  - B tells C its (B's) distance to A is infinite  
(so C won't route to A via B)

# DV: Poisoned Reverse

If B routes through C to get to A:  
 B tells C its (B's) distance to A is infinite

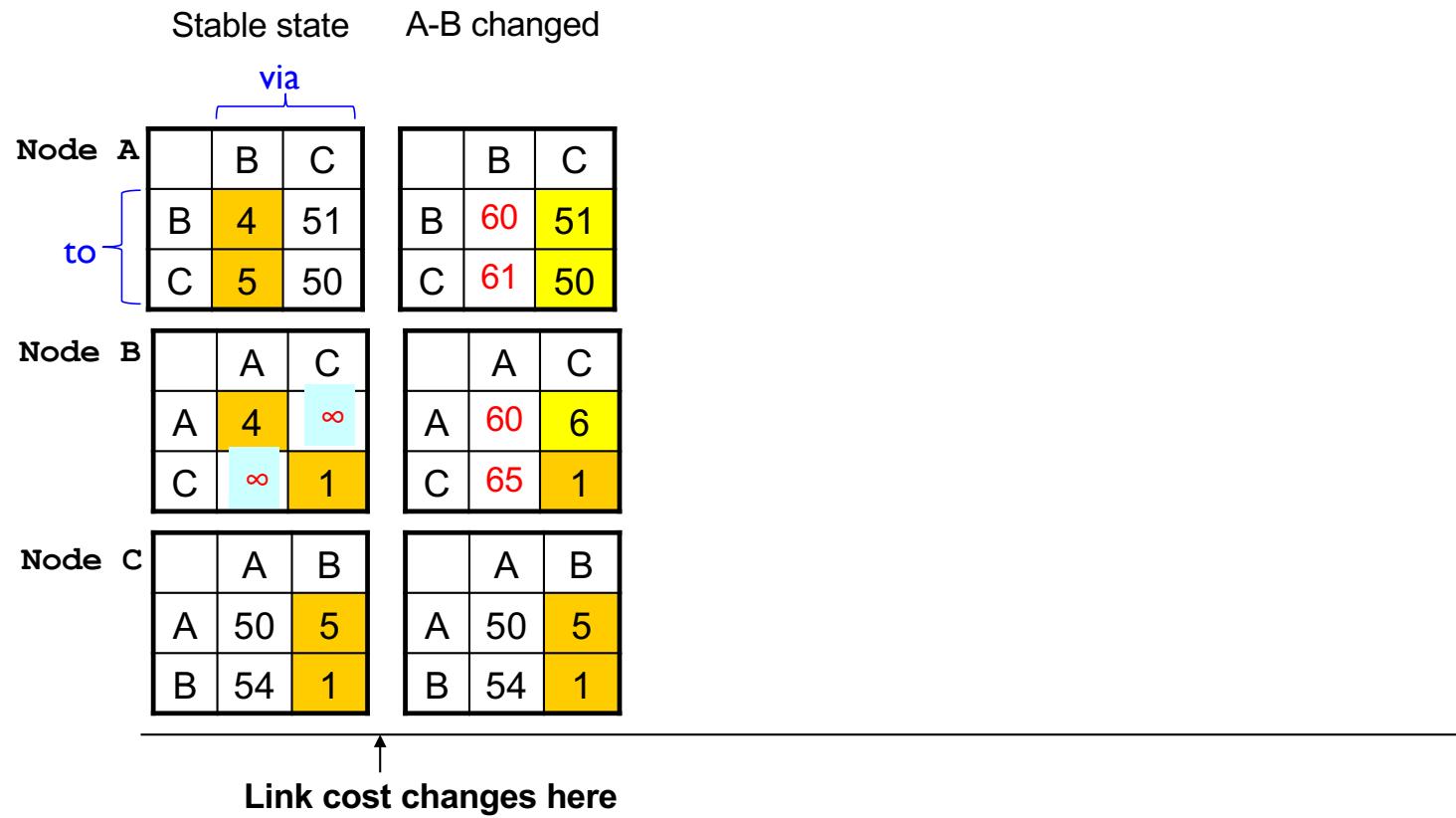
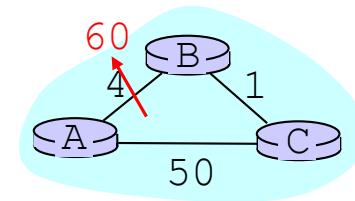


Stable state



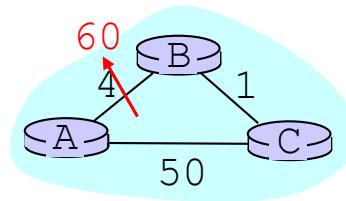
# DV: Poisoned Reverse

If B routes through C to get to A:  
B tells C its (B's) distance to A is infinite



# DV: Poisoned Reverse

*If B routes through C to get to A:  
B tells C its (B's) distance to A is infinite*



Stable state      A-B changed      A sends its DV to B, C      B sends its DV to A, C

*via*

*to*

Link cost changes here

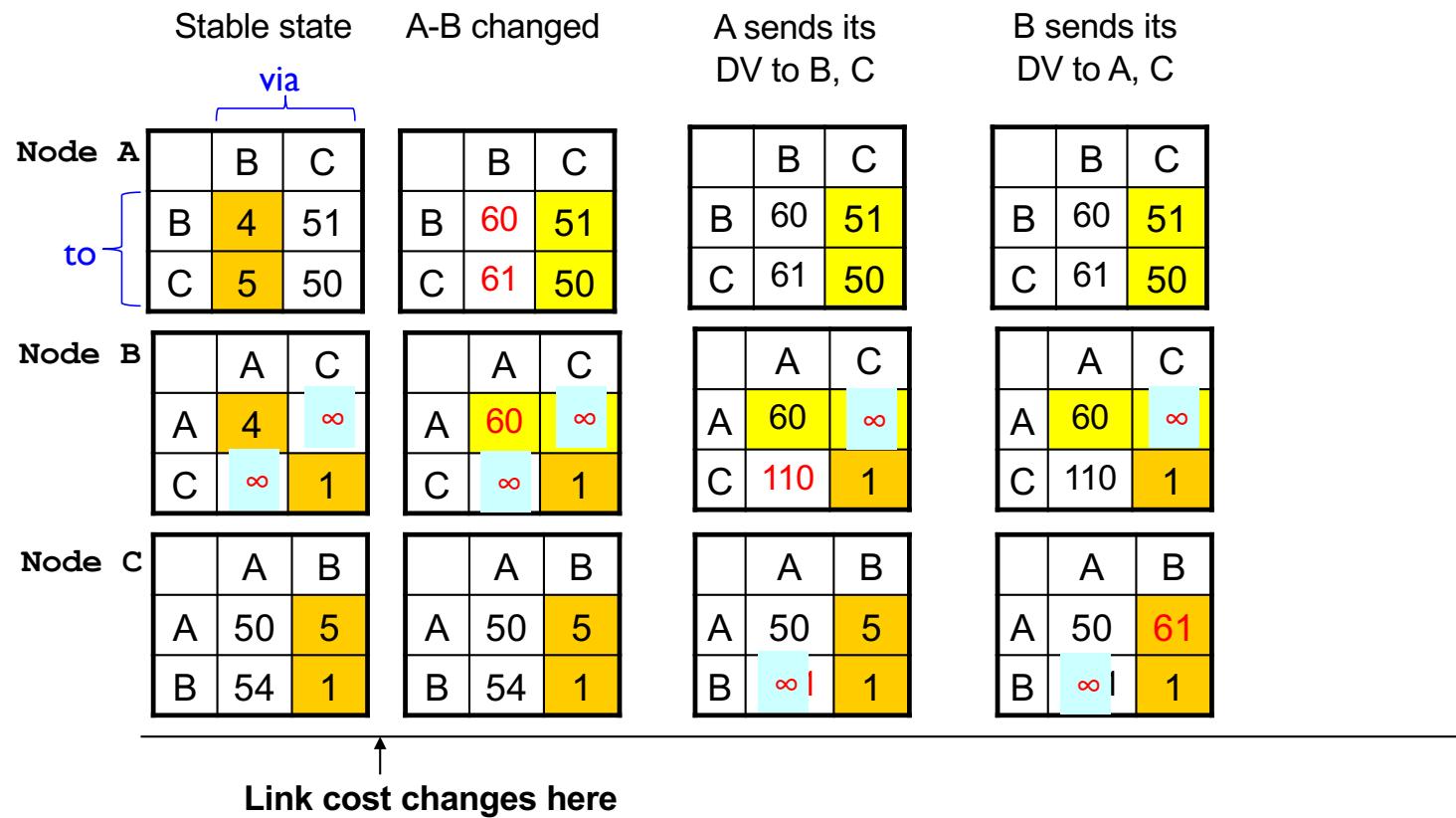
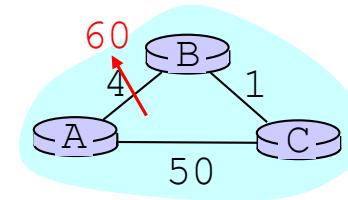
|        |   |    |    |    |    |    |    |    |
|--------|---|----|----|----|----|----|----|----|
| Node A | B | C  | B  | C  | B  | C  | B  | C  |
|        | 4 | 51 | 60 | 51 | 60 | 51 | 60 | 51 |
|        | 5 | 50 | 61 | 50 | 61 | 50 | 61 | 50 |

|        |          |          |          |          |     |          |     |          |
|--------|----------|----------|----------|----------|-----|----------|-----|----------|
| Node B | A        | C        | A        | C        | A   | C        | A   | C        |
|        | 4        | $\infty$ | 60       | $\infty$ | 60  | $\infty$ | 60  | $\infty$ |
|        | $\infty$ | 1        | $\infty$ | 1        | 110 | 1        | 110 | 1        |

|        |    |   |    |   |          |   |          |   |
|--------|----|---|----|---|----------|---|----------|---|
| Node C | A  | B | A  | B | A        | B | A        | B |
|        | 50 | 5 | 50 | 5 | 50       | 5 | 50       | 7 |
|        | 54 | 1 | 54 | 1 | $\infty$ | 1 | $\infty$ | 1 |

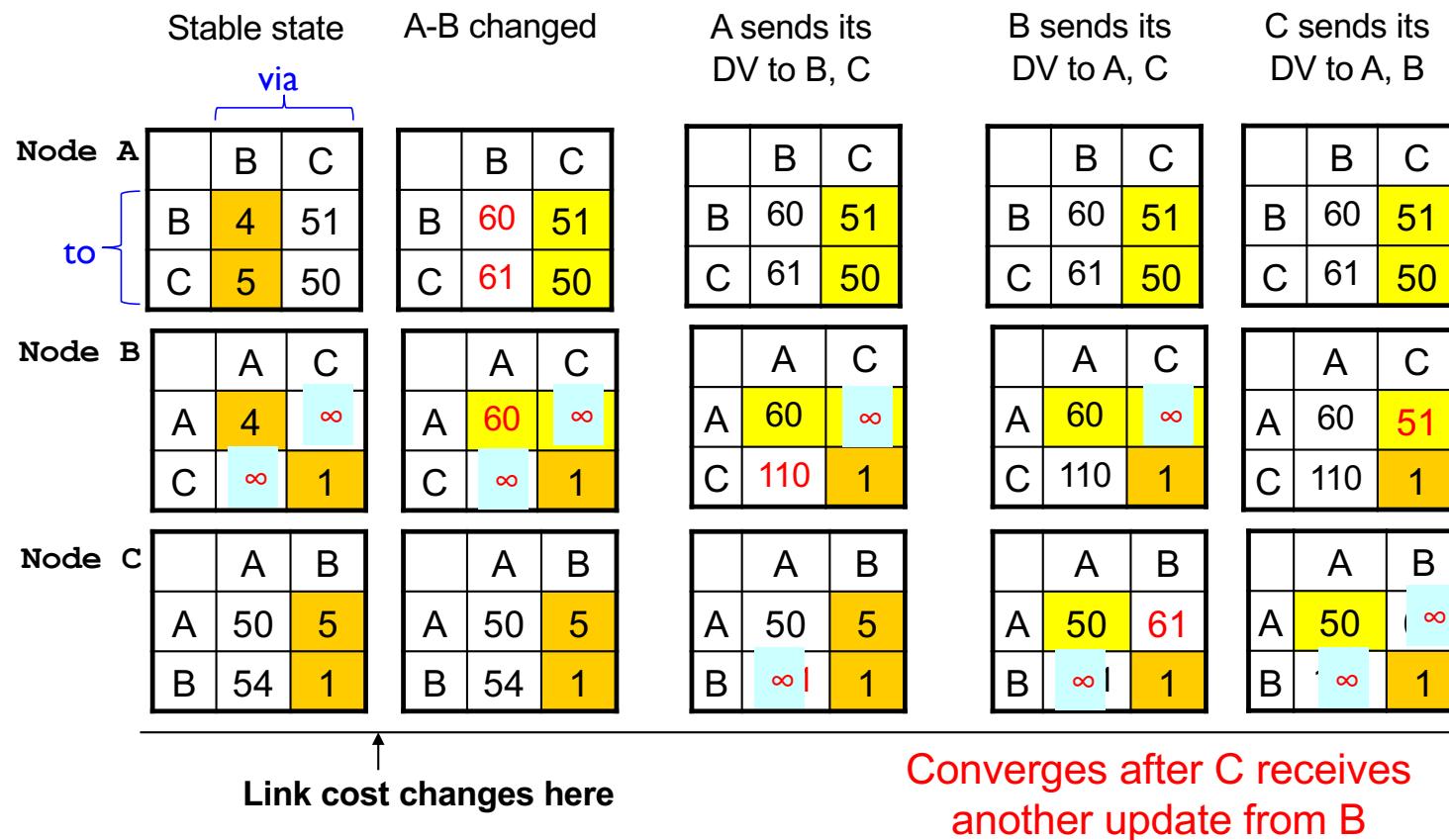
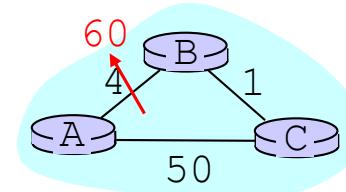
# DV: Poisoned Reverse

If B routes through C to get to A:  
 B tells C its (B's) distance to A is infinite

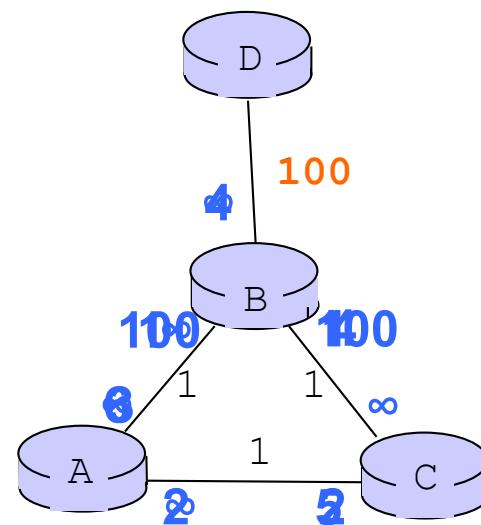


# DV: Poisoned Reverse

If B routes through C to get to A:  
 B tells C its (B's) distance to A is infinite



# Will Poison-Reverse Completely Solve the Count-to-Infinity Problem?



Numbers in blue denote the best cost  
to destination D advertised along the link

# Comparison of LS and DV algorithms

## message complexity

LS:  $n$  routers,  $O(n^2)$  messages sent

DV: exchange between neighbors;  
convergence time varies

## speed of convergence

LS:  $O(n^2)$  algorithm,  $O(n^2)$  messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

LS:

- router can advertise incorrect *link* cost
- each router computes only its own table

DV:

- DV router can advertise incorrect *path* cost (“I have a *really* low cost path to everywhere”): black-holing
- each router’s table used by others: error propagate thru network

# Real Protocols

*Link State*

Open Shortest Path First  
(OSPF)

Intermediate system to  
intermediate system (IS-IS)

*Distance Vector*

Routing Information  
Protocol (RIP)

Interior Gateway Routing  
Protocol (IGRP-Cisco)

Border Gateway Protocol  
(BGP) - variant

# Quiz: Link-state routing

- ❖ In link state routing, each node sends information of its direct links (i.e., link state) to \_\_\_\_\_?
  - A. Immediate neighbours
  - B. All nodes in the network
  - C. Any one neighbor
  - D. No one

# Quiz: Distance-vector routing

- ❖ In distance vector routing, each node shares its distance table with \_\_\_\_\_?
  - A. All Immediate neighbours
  - B. All nodes in the network
  - C. Any one neighbor
  - D. No one

# Quiz: Distance-vector routing

- ❖ Which of the following is true of distance vector routing?
  - A. Convergence delay depends on the topology (nodes and links) and link weights
  - B. Convergence delay depends on the number of nodes and links
  - C. Each node knows the entire topology
  - D. A and C
  - E. B and C

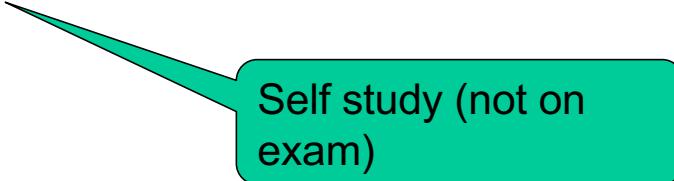
# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- ❖ link state
- ❖ distance vector
- ❖ hierarchical routing

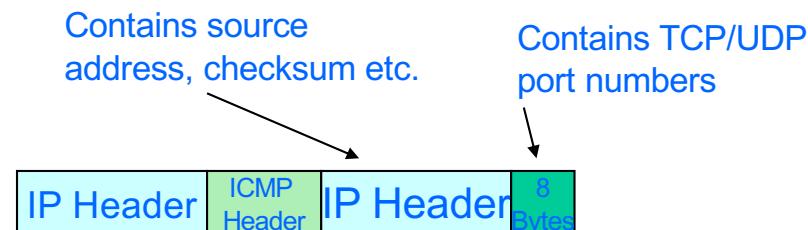
5.6 ICMP: The Internet Control  
Message Protocol



Self study (not on  
exam)

# ICMP: Internet Control Message Protocol

- ❖ Used by hosts & routers to communicate network level information
  - Error reporting: unreachable host, network, port
  - Echo request/reply (used by ping)
- ❖ Works above IP layer
  - ICMP messages carried in IP datagrams
- ❖ ICMP message: type, code plus IP header and first 8 bytes of IP datagram payload causing error



## ICMP: Internet Control Message Protocol

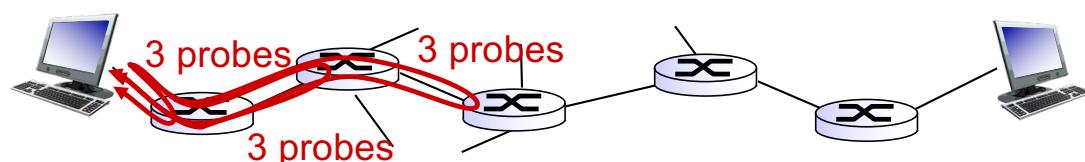
| Type | Code | Description                   |
|------|------|-------------------------------|
| 0    | 0    | echo reply(ping)              |
| 3    | 0    | dest. network unreachable     |
| 3    | 1    | dest host unreachable         |
| 3    | 3    | dest port unreachable         |
| 3    | 4    | frag needed; DF set           |
| 8    | 0    | echo request(ping)            |
| 11   | 0    | TTL expired                   |
| 11   | 1    | frag reassembly time exceeded |
| 12   | 0    | bad IP header                 |

# Traceroute and ICMP

- Source sends series of UDP segments to dest
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- When  $n$ th set of datagrams arrives to  $n$ th router:
  - router discards datagrams
  - and sends source ICMP messages (type 11, code 0)
  - ICMP messages includes IP address of router
- when ICMP messages arrives, source records RTTs

*stopping criteria:*

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



# Summary

- ❖ Network Layer: Data Plane
  - Overview
  - IP
- ❖ Network Layer: Control Plane
  - Routing Protocols
    - Link-state
    - Distance Vector
  - ICMP

# COMP 3331/9331: Computer Networks and

1. *Error detection/correction: Parity, CRC*
2. *Medium access control (MAC)*
3. *Switch, ARP*
4. *Ethernet*

## Applications Week 9 Data link Layer

Reading Guide: Chapter 6, Sections 6.1 – 6.4, 6.7



**“Having the lecture  
recordings up – absolute  
lifesaver when it comes  
to revision”**

**Shape  
our  
Future**

Tell us about your experience and  
shape the future of education at UNSW.



Click the link in Moodle now



*Complete your myExperience and shape  
the future of education at UNSW.*

*Click the  Experience link in Moodle*

*or login to [myExperience.unsw.edu.au](https://myexperience.unsw.edu.au)*

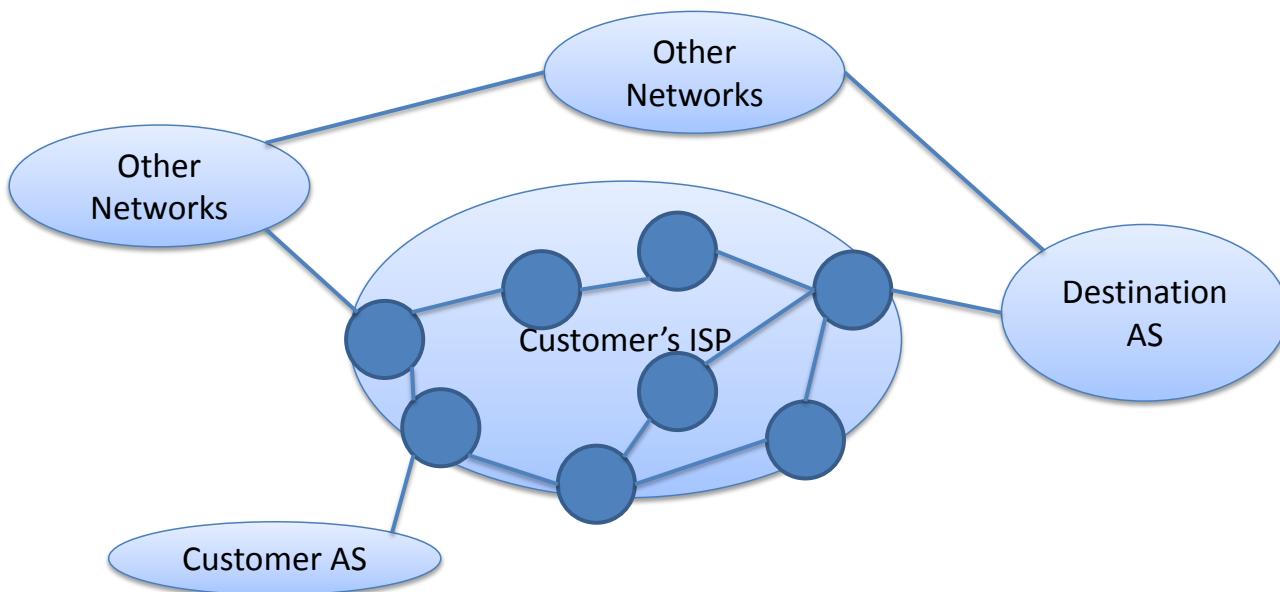
*(use z1234567@ad.unsw.edu.au to login)*

*The survey is confidential, your identity will never be released*

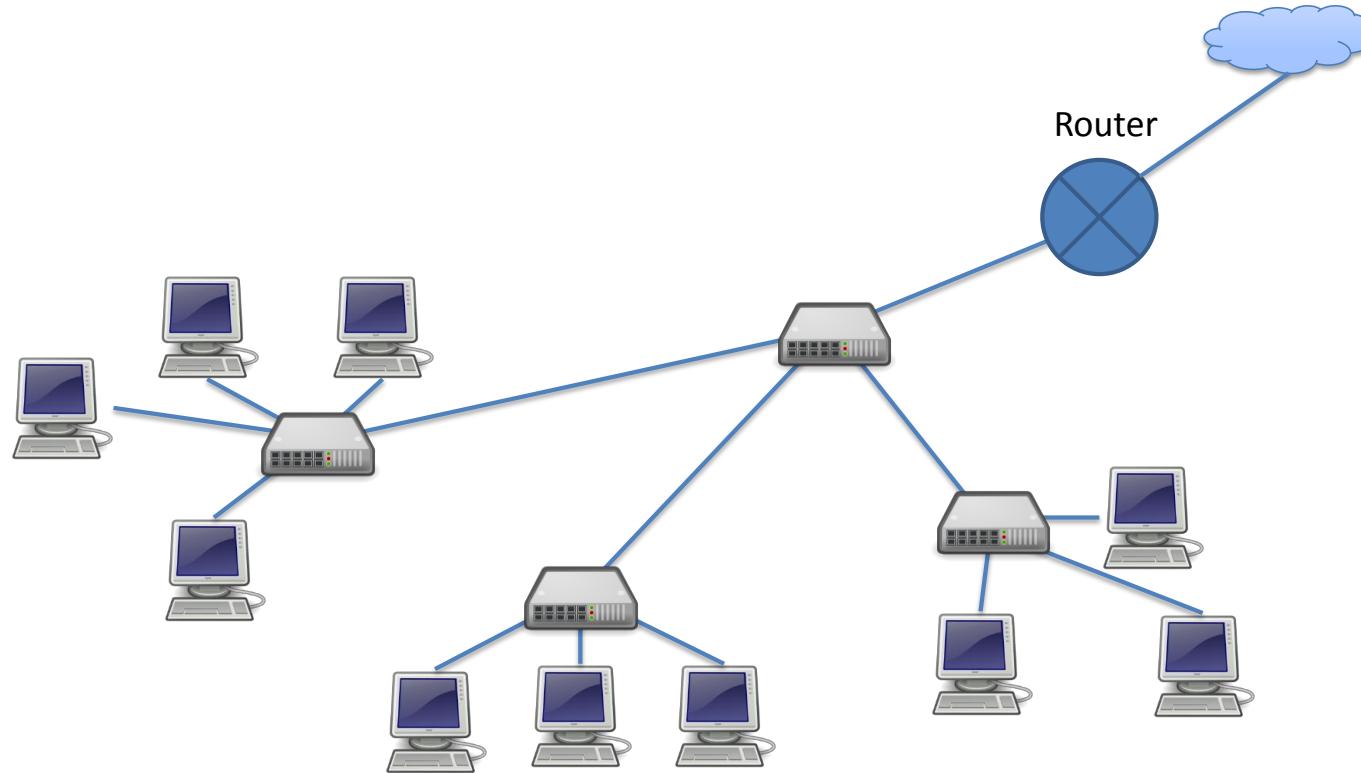
*Survey results are not released to teaching staff until after your results are published*

# From Macro- to Micro-

- Previously, we looked at Internet scale...



## Link layer focus: Within a Subnet



# 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

# Link layer, LANs: roadmap

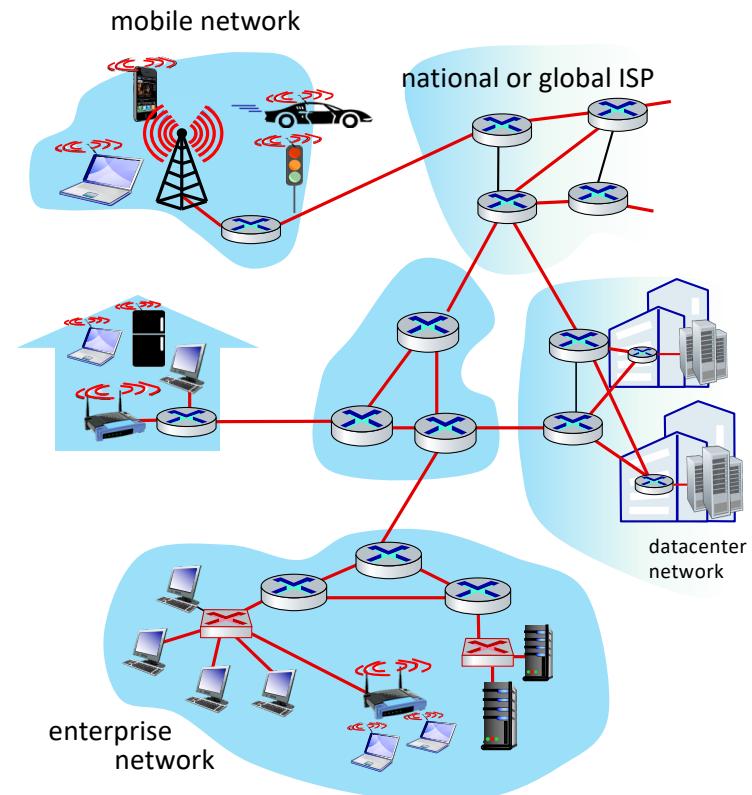
- *introduction*
- *error detection, correction*
- *multiple access protocols*
- *LANs*
  - *addressing, ARP*
  - *Ethernet*
  - *switches*
  - *VLANs (NOT COVERED)*
- *link virtualization: MPLS (NOT COVERED)*
- *data center networking (NOT COVERED)*
- *a day in the life of a web request*

# 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*

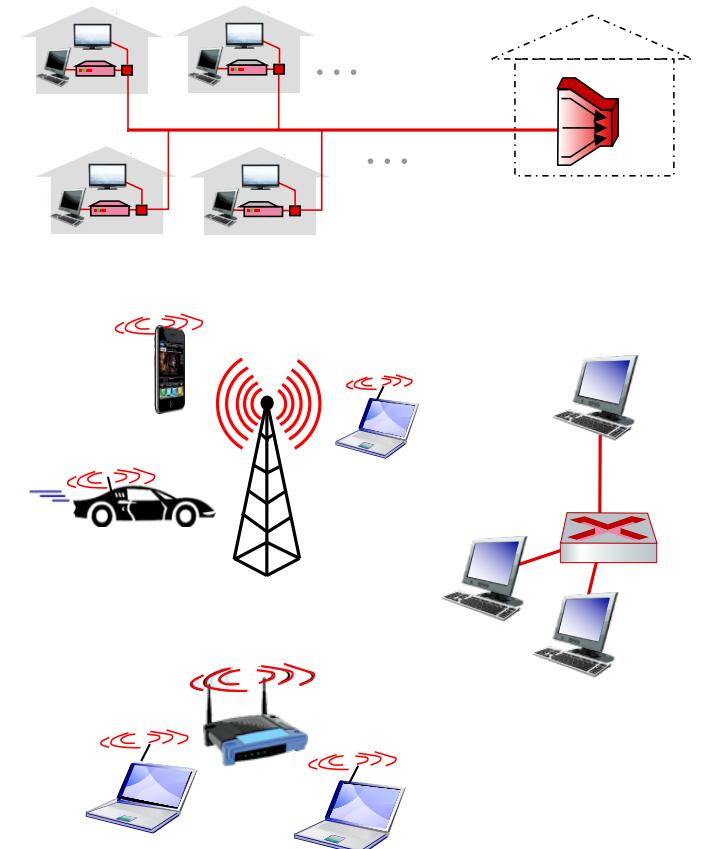


# Link layer: context

- *datagram transferred by different link protocols over different links:*
    - e.g., WiFi on *first link*, Ethernet on *next link*
  - *each link protocol provides different services*
    - e.g., *may or may not provide reliable data transfer over link*
- transportation analogy:*
- *trip from Princeton to Lausanne*
    - *limo: Princeton to JFK*
    - *plane: JFK to Geneva*
    - *train: Geneva to Lausanne*
  - *tourist = datagram*
  - *transport segment = communication link*
  - *transportation mode = link-layer protocol*
  - *travel agent = routing algorithm*

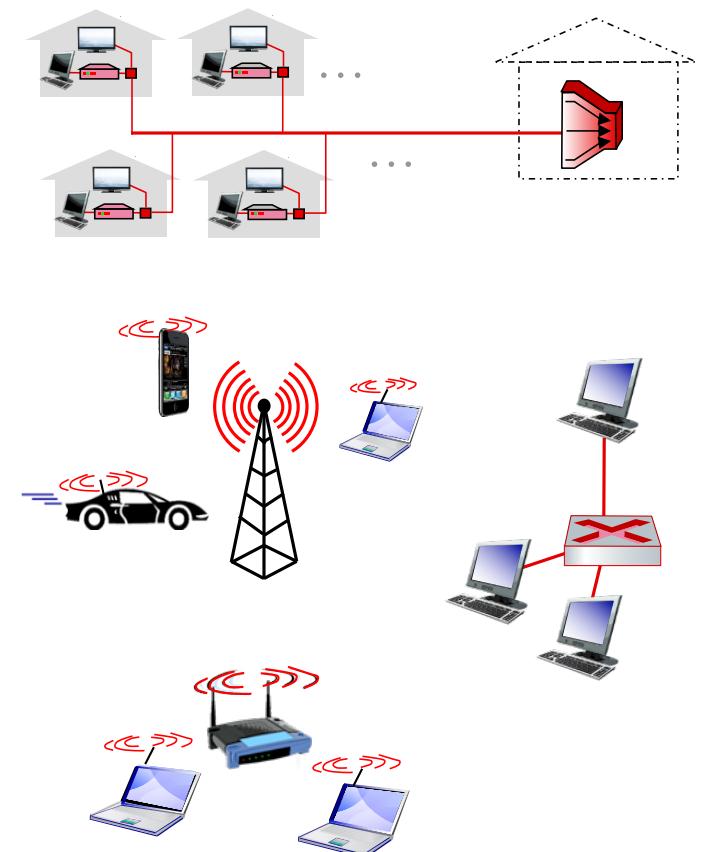
# Link layer: services

- **framing, link access:**
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - “MAC” addresses 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?



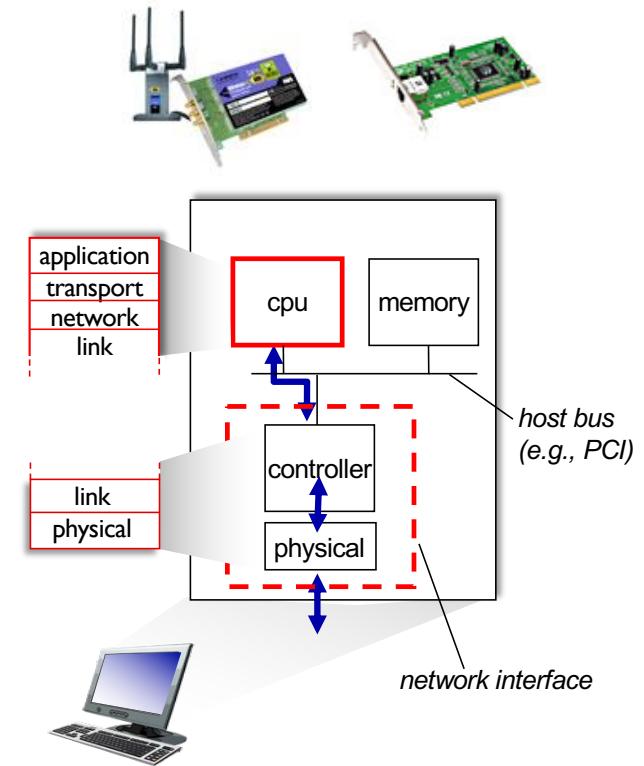
# Link layer: services (more)

- **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*
- **half-duplex and full-duplex:**
  - *with half duplex, nodes at both ends of link can transmit, but not at same time*

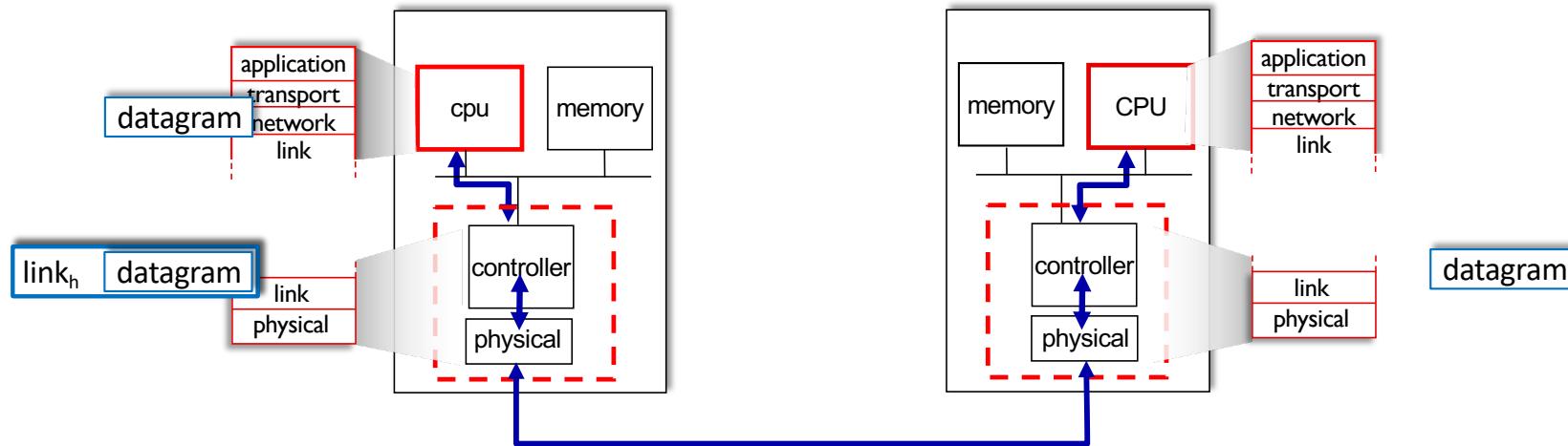


# Where is the link layer implemented?

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



# Interfaces communicating



*sending side:*

- *encapsulates datagram in frame*
- *adds error checking bits, reliable data transfer, flow control, etc.*

*receiving side:*

- *looks for errors, reliable data transfer, flow control, etc.*
- *extracts datagram, passes to upper layer at receiving side*

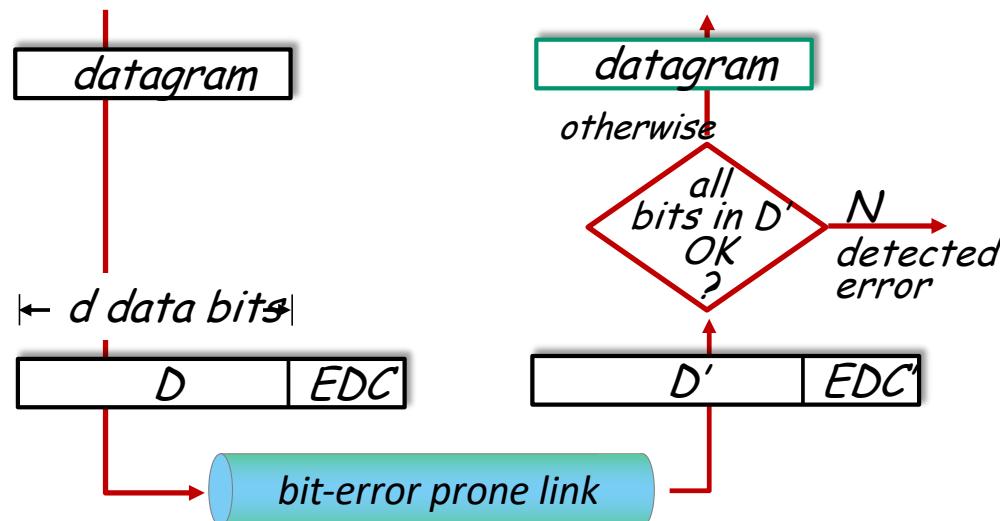
# Link layer, LANs: roadmap

- *introduction*
- ***error detection, correction***
- *multiple access protocols*
- *LANs*
  - *addressing, ARP*
  - *Ethernet*
  - *switches*
  - *VLANs (NOT COVERED)*
- *link virtualization: MPLS (NOT COVERED)*
- *data center networking (NOT COVERED)*
- *a day in the life of a web request*

# Error detection

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

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



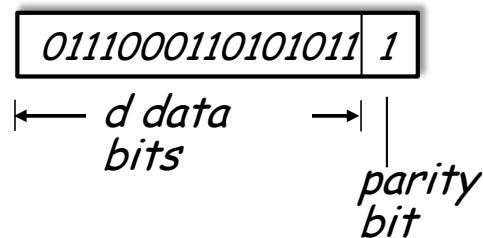
Error detection not 100% reliable!

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

# Parity checking

## single bit parity:

- detect single bit errors



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

## two-dimensional bit parity:

- detect *and* correct single bit errors

|             |     |             |               | row parity |
|-------------|-----|-------------|---------------|------------|
| $d_{1,1}$   | ... | $d_{1,j}$   | $d_{1,j+1}$   |            |
| $d_{2,1}$   | ... | $d_{2,j}$   | $d_{2,j+1}$   |            |
| ...         | ... | ...         | 1 ..          |            |
| $d_{i,1}$   | ... | $d_{i,j}$   | $d_{i,j+1}$   |            |
| <hr/>       |     |             |               |            |
| $d_{i+1,1}$ | ... | $d_{i+1,j}$ | $d_{i+1,j+1}$ |            |
|             |     |             |               | 1          |

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

parity error

parity error

# Internet checksum (review)

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

## sender:

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

## receiver:

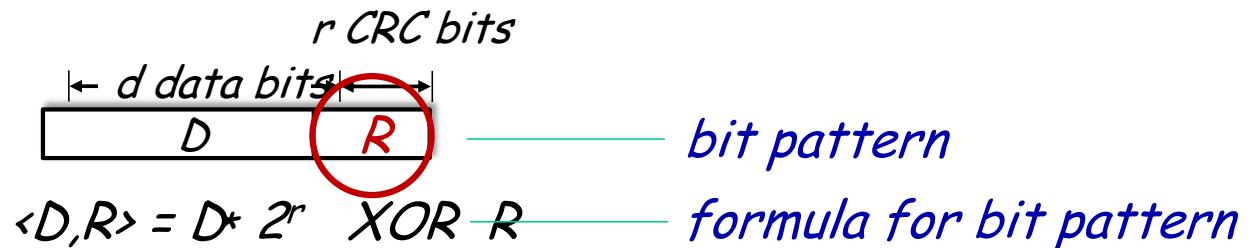
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected. *But maybe errors nonetheless?* More later ....

## In practice

- Bit errors occur in bursts.
- We're willing to trade computational complexity for space efficiency.
  - Make the detection routine more complex, to detect error bursts, without tons of extra data
- Insight: We need hardware to interface with the network, do the computation there!

# 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)

# A Note on Modulo-2 Arithmetic

- All calculations are modulo-2 arithmetic
- No carries or borrows in subtraction
- Addition and subtraction are identical and both are equivalent to XOR
  - $1011 \text{ XOR } 0101 = 1110$
  - $1011 - 0101 = 1110$
  - $1011 + 0101 = 1110$
- Multiplication by  $2^k$  is essentially a left shift by  $k$  bits
  - $1011 \times 2^2 = 101100$

# CRC example

want:

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

equivalently:

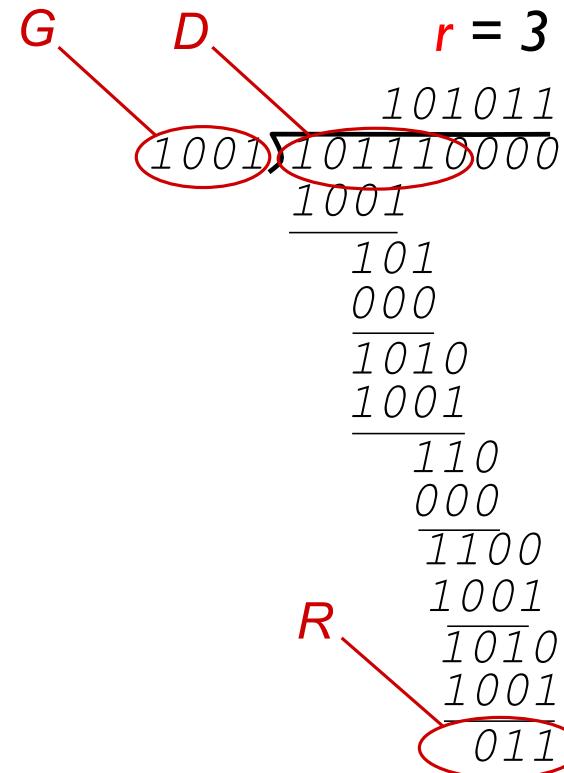
$$D \cdot 2^r = nG \text{ XOR } R$$

equivalently:

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

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

*At the sender*



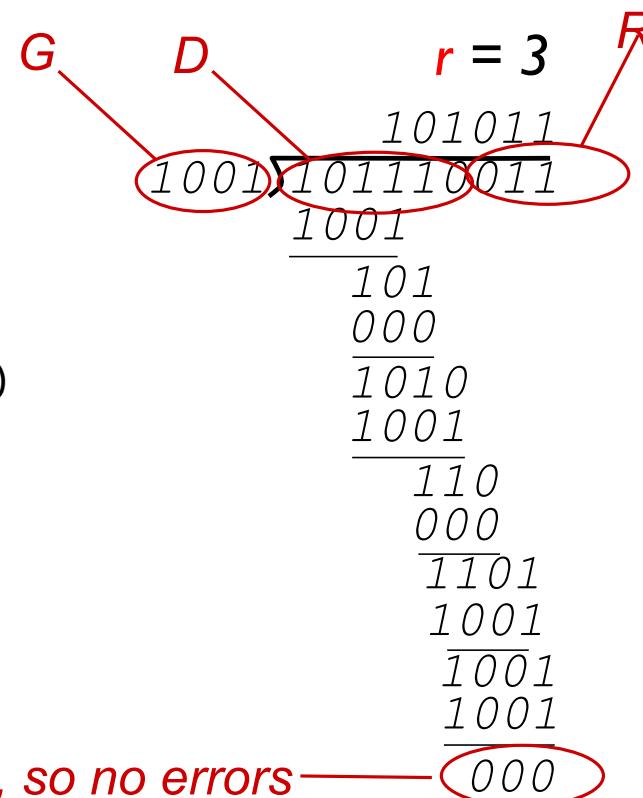
Sender sends  $\langle D, R \rangle$  into the channel

# CRC example

Receiver divides the received frame and divides by  $G$  and checks if the remainder is zero.

In this example, there are no errors, so the receiver receives  $\langle D, R \rangle$  (from previous slide)

*At the receiver*





## Quiz: Error Detection/Correction

- ❖ Can these schemes respectively correct any bit errors: Internet checksums, two-dimensional parity, cyclic redundancy check (CRC)
  - a) Yes, No, No
  - b) No, Yes, Yes
  - c) No, Yes, No
  - d) No, No, Yes
  - e) No, No, No

# Link layer, LANs: roadmap

- *introduction*
- *error detection, correction*
- ***multiple access protocols***
- *LANs*
  - *addressing, ARP*
  - *Ethernet*
  - *switches*
  - *VLANs*
- *link virtualization: MPLS (NOT COVERED)*
- *data center networking (NOT COVERED)*
- *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/4G satellite



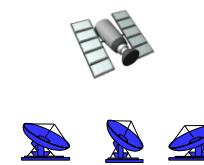
shared wire (e.g., cabled Ethernet)



shared radio: 4G/5G



shared radio: WiFi



shared radio: satellite



humans at a cocktail party (shared air, acoustical)

# Multiple access protocols

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

## *multiple access protocol*

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

# An ideal multiple access protocol

*given: multiple access channel (MAC) of rate  $R$  bps*

*desired properties:*

1. *when one node wants to transmit, it can send at rate  $R$ .*
2. *when  $M$  nodes want to transmit, each can send at average rate  $R/M$*
3. *fully decentralized:*
  - *no special node to coordinate transmissions*
  - *no synchronization of clocks, slots*
4. *simple*

# MAC protocols: taxonomy

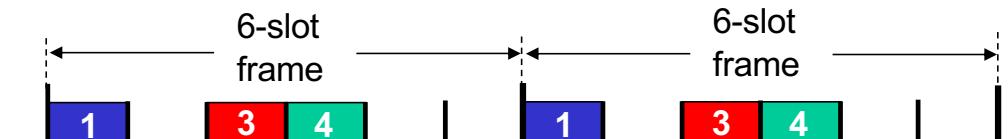
*three broad classes:*

- ***channel partitioning***
  - divide channel into smaller “pieces” (time slots, frequency, code)
  - allocate piece to node for exclusive use
- ***random access***
  - channel not divided, allow collisions
  - “recover” from collisions
- ***“taking turns”***
  - nodes take turns, but nodes with more to send can take longer turns

# Channel partitioning MAC protocols: TDMA

*TDMA: time division multiple access*

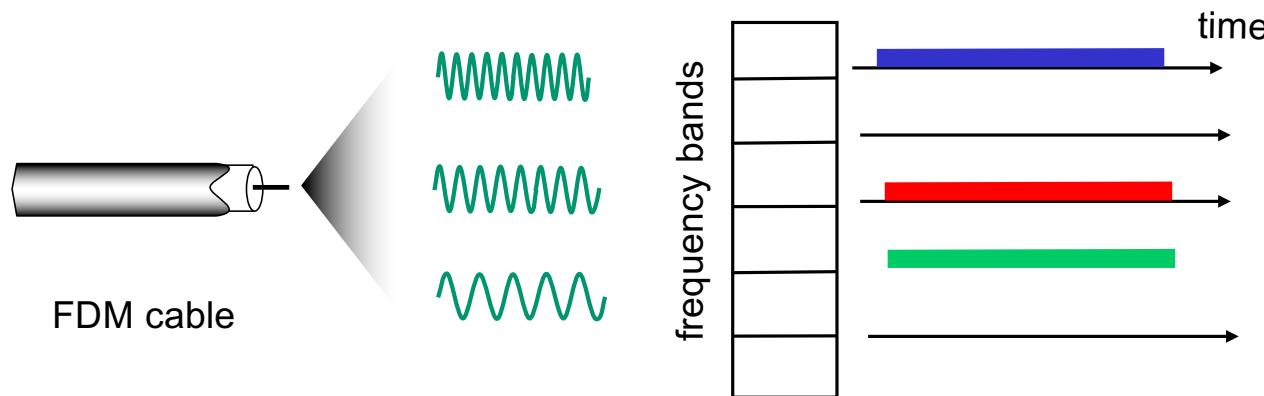
- access to channel in “rounds”
- each node gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-node LAN, 1,3,4 have packets to send, slots 2,5,6 idle



# Channel partitioning MAC protocols: FDMA

## *FDMA: frequency division multiple access*

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





## Quiz: Does channel partitioning satisfy ideal properties ?

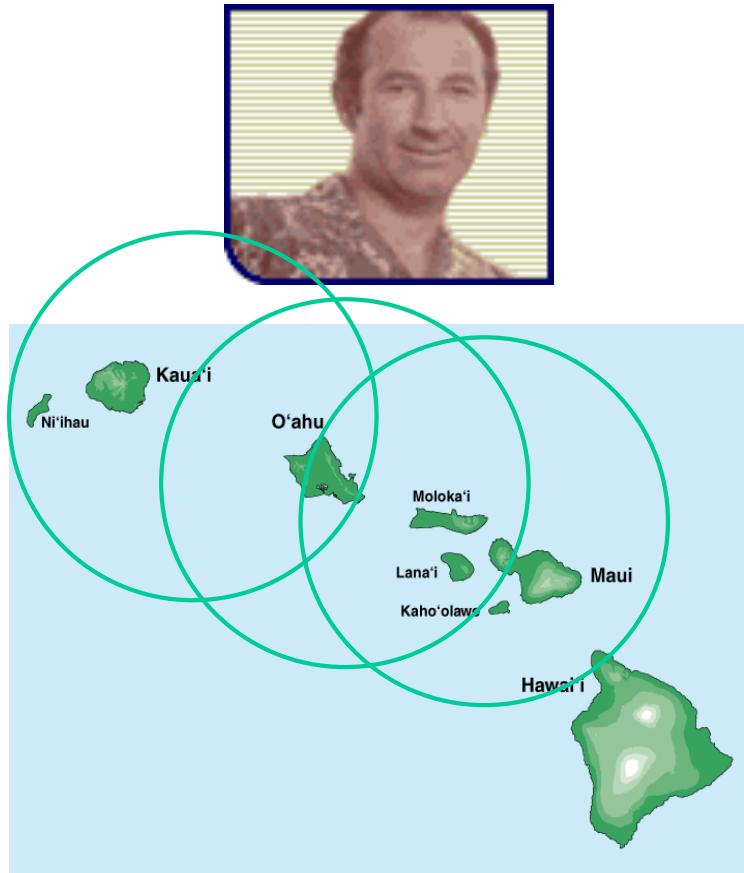
1. if only one node wants to transmit, it can send at rate R.
  2. when M nodes want to transmit, each can send at average rate  $R/M$  (fairness)
  3. fully decentralized:
    - no synchronization of clocks, slots
    - no special node to coordinate transmissions
  4. simple
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4

(Which ones?)

# Random access protocols

- *when node has packet to send*
  - *transmit at full channel data rate  $R$ .*
  - *no a priori coordination among nodes*
- *two or more transmitting nodes: “collision”*
- *random access MAC protocol specifies:*
  - *how to detect collisions*
  - *how to recover from collisions (e.g., via delayed retransmissions)*
- *examples of random-access MAC protocols:*
  - ALOHA, slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA

# Where it all Started: AlohaNet



- ❖ Norm Abramson left Stanford in 1970 (***so he could surf!***)
- ❖ Set up first data communication system for Hawaiian islands
- ❖ Central hub at U. Hawaii, Oahu

# Slotted ALOHA

## *assumptions:*

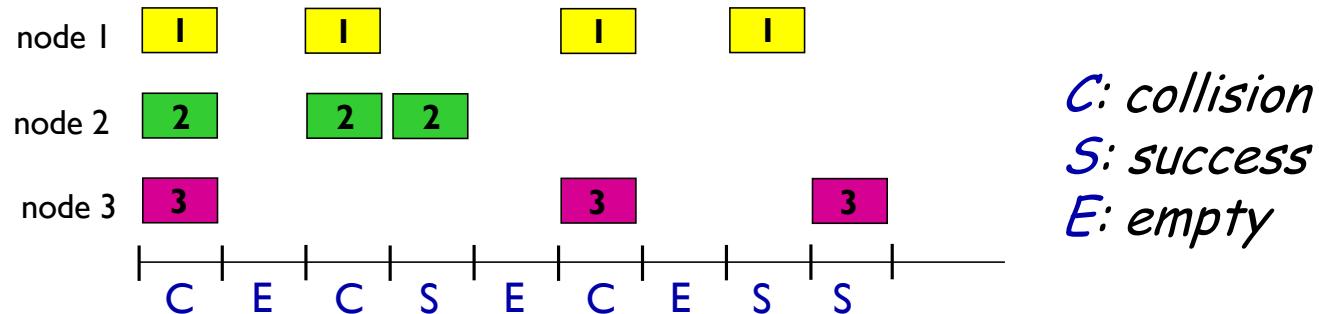
- *all frames same size*
- *time divided into equal size slots  
(time to transmit 1 frame)*
- *nodes start to transmit only slot  
beginning*
- *nodes are synchronized*
- *if 2 or more nodes transmit in  
slot, all nodes detect collision*

## *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

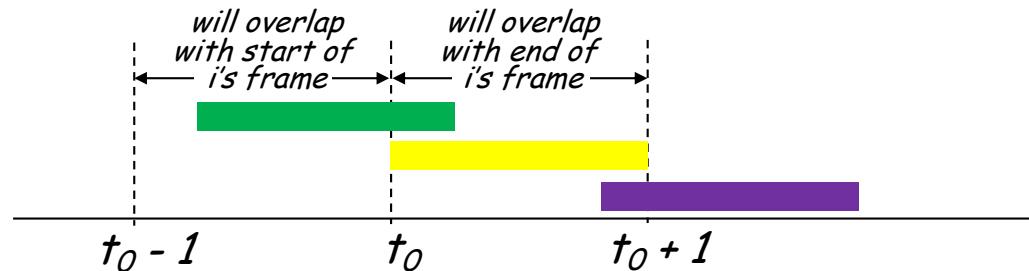
## Slotted ALOHA: efficiency

**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!

# Pure ALOHA

- *unslotted Aloha: simpler, no synchronization*
  - *when frame first arrives: transmit immediately*
- *collision probability increases with no synchronization:*
  - *frame sent at  $t_0$  collides with other frames sent in  $[t_0 - l, t_0 + l]$*



- *pure Aloha efficiency: 18% !*

# CSMA (carrier sense multiple access)

*simple CSMA: listen before transmit:*

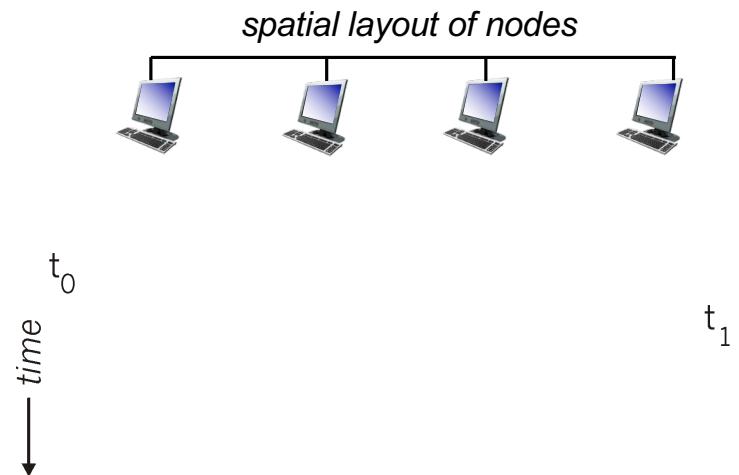
- *if channel sensed idle: transmit entire frame*
- *if channel sensed busy: defer transmission*
- *human analogy: don 't interrupt others!*

*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*

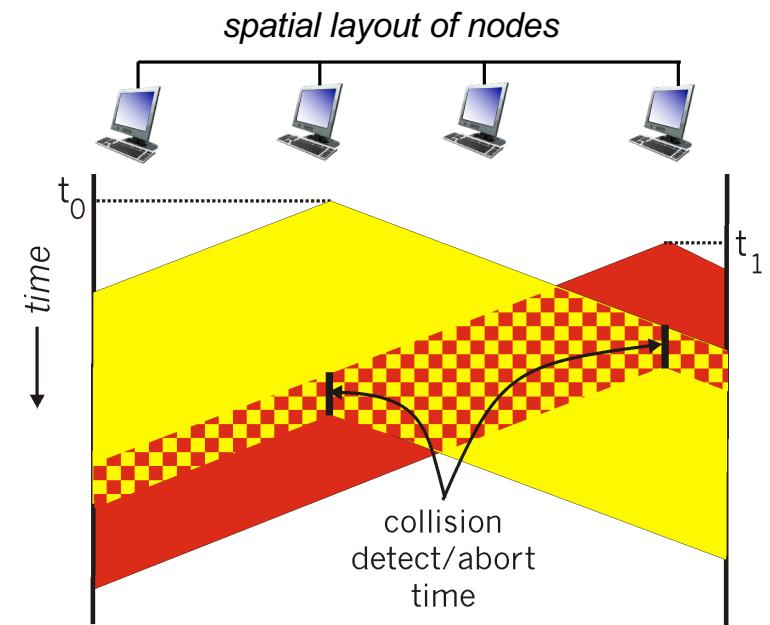
# CSMA: collisions

- *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



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

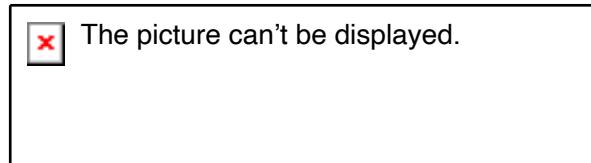
# Ethernet CSMA/CD algorithm

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

NIC = Network Interface Card

# CSMA/CD efficiency

- $T_{prop}$  = max prop delay between 2 nodes in LAN
- $t_{trans}$  = time to transmit max-size frame



- efficiency goes to 1
  - as  $t_{prop}$  goes to 0
  - as  $t_{trans}$  goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!



## Quiz: Does CSMA/CD satisfy ideal properties ?

1. if only one node wants to transmit, it can send at rate R.
  2. when M nodes want to transmit, each can send at average rate  $R/M$  (fairness)
  3. fully decentralized:
    - no synchronization of clocks, slots
    - no special node to coordinate transmissions
  4. simple
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4

(Which ones?)

# “Taking turns” MAC protocols

*channel partitioning MAC protocols:*

- share channel efficiently and fairly at high load
- inefficient at low load: delay in channel access, 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

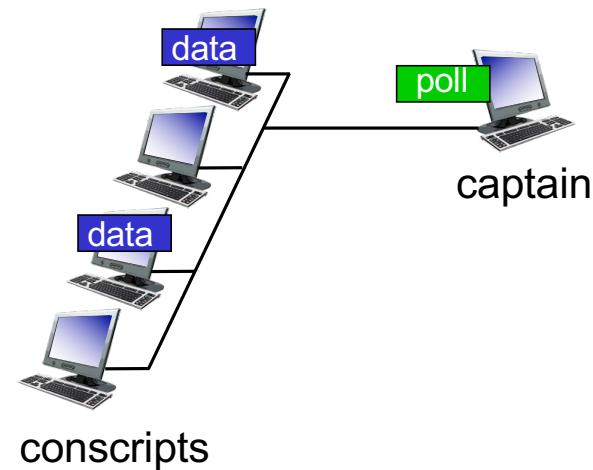
*“taking turns” protocols*

- look for best of both worlds!

# “Taking turns” MAC protocols

## *polling:*

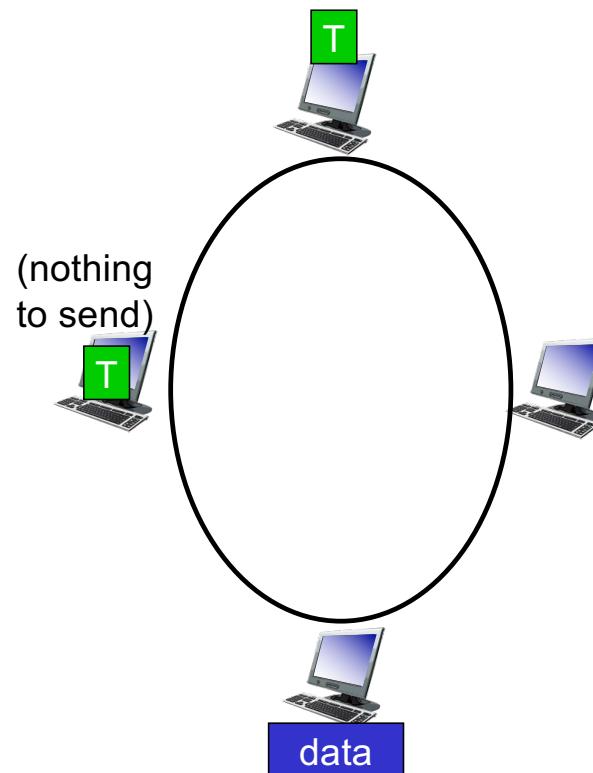
- *captain node “invites” other nodes to transmit in turn*
- *typically used with “dumb” devices*
- *concerns:*
  - *polling overhead*
  - *latency*
  - *single point of failure (captain)*



# “Taking turns” MAC protocols

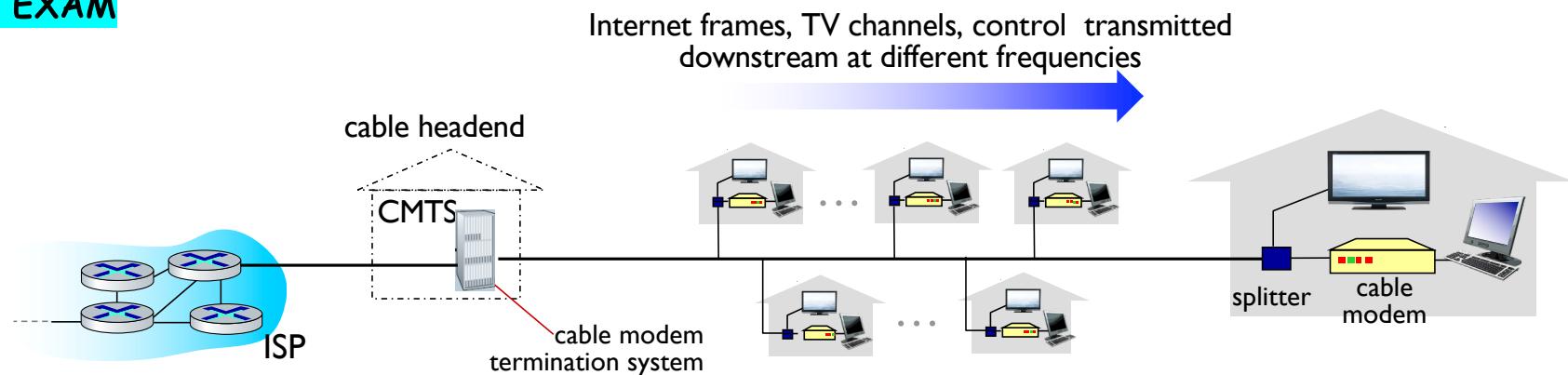
*token passing:*

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



# Cable access network: FDM, TDM and random access!

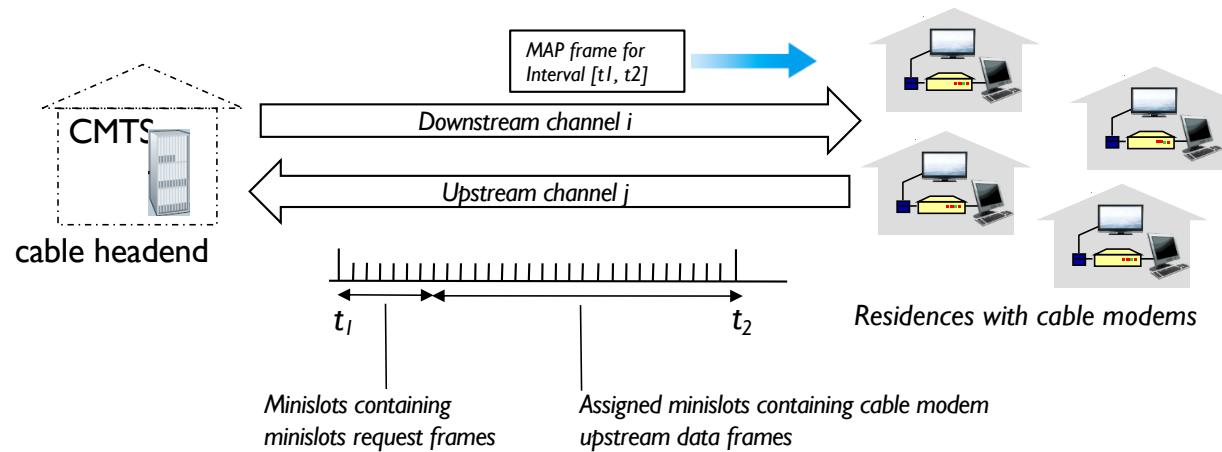
NOT ON EXAM



- **multiple** downstream (broadcast) *FDM channels: up to 1.6 Gbps/channel*
  - single CMTS transmits into channels
- **multiple** upstream channels (up to 1 Gbps/channel)
  - **multiple access:** all users contend (random access) for certain upstream channel time slots; others assigned TDM

# Cable access network:

NOT ON EXAM



**DOCSIS: data over cable service interface specification**

- FDM over upstream, downstream frequency channels
- TDM upstream: some slots assigned, some have contention
  - downstream MAP frame: assigns upstream slots
  - request for upstream slots (and data) transmitted random access (binary backoff) in selected slots

# Quiz: Does taking turns satisfy ideal properties ?



1. if only one node wants to transmit, it can send at rate R.
  2. when M nodes want to transmit, each can send at average rate  $R/M$  (fairness)
  3. fully decentralized:
    - no synchronization of clocks, slots
    - no special node to coordinate transmissions
  4. simple
- A. 0  
B. 1  
C. 2  
D. 3  
E. 4

(Which ones?)

# Summary of MAC protocols

- *channel partitioning*, by time, frequency or code
  - Time Division, Frequency Division
- *random access (dynamic)*,
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- *taking turns*
  - polling from central site, token passing
  - Bluetooth, FDDI, token ring

# Link layer, LANs: roadmap

- *introduction*
- *error detection, correction*
- *multiple access protocols*
- **LANs**
  - **addressing, ARP**
  - *Ethernet*
  - *switches*
  - *VLANs*
- *link virtualization: MPLS (NOT COVERED)*
- *data center networking (NOT COVERED)*
- *a day in the life of a web request*

# MAC addresses

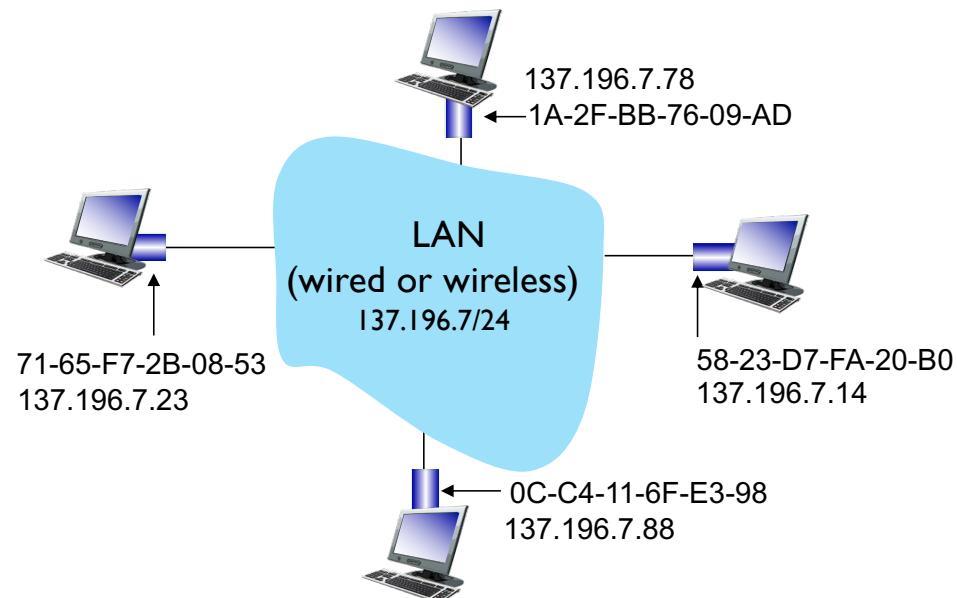
- 32-bit IP address:
  - network-layer address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
  - function: used “locally” to get frame from one interface to another physically-connected interface (same subnet, in 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)



# MAC addresses

- *MAC address allocation administered by IEEE*
- *manufacturer buys portion of MAC address space (to assure uniqueness)*
- *analogy:*
  - *MAC address: like TFN (SSN)*
  - *IP address: like postal address*
- *MAC flat address: portability*
  - *can move interface from one LAN to another*
  - *recall IP address not portable: depends on IP subnet to which node is attached*

# MAC Address vs. IP Address

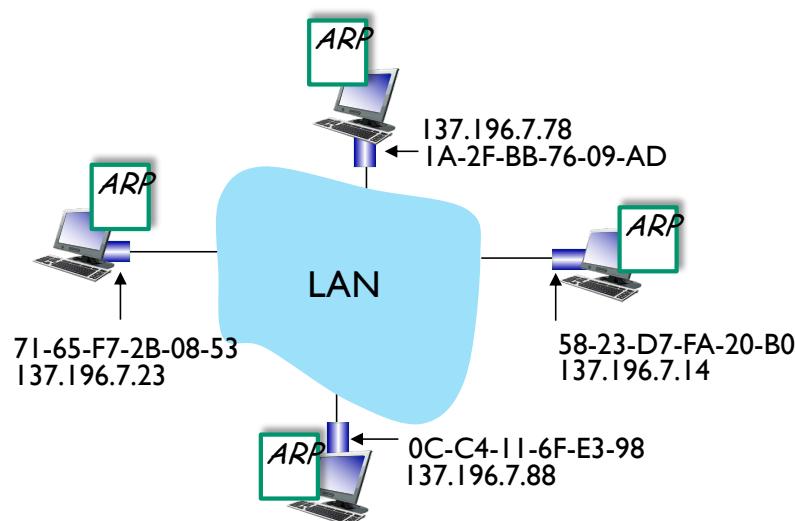
- ❖ MAC addresses (used in link-layer)
  - Hard-coded in read-only memory when adapter is built
  - Flat name space of 48 bits (e.g., 00-0E-9B-6E-49-76)
  - Portable, and can stay the same as the host moves
  - Used to get packet between interfaces on same network
- ❖ IP addresses
  - learned dynamically
  - Hierarchical name space of 32 bits (e.g., 12.178.66.9)
  - Not portable, and depends on where the host is attached
  - Used to get a packet to destination IP subnet

# Taking Stock: Naming

| Layer         | Examples            | Structure                | Configuration | Resolution Service |
|---------------|---------------------|--------------------------|---------------|--------------------|
| App. Layer    | www.cse.unsw.edu.au | organizational hierarchy | ~ manual      | DNS                |
| Network Layer | 129.94.242.51       | topological hierarchy    | DHCP          |                    |
| Link layer    | 45-CC-4E-12-F0-97   | vendor (flat)            | hard-coded    | ARP                |

# 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

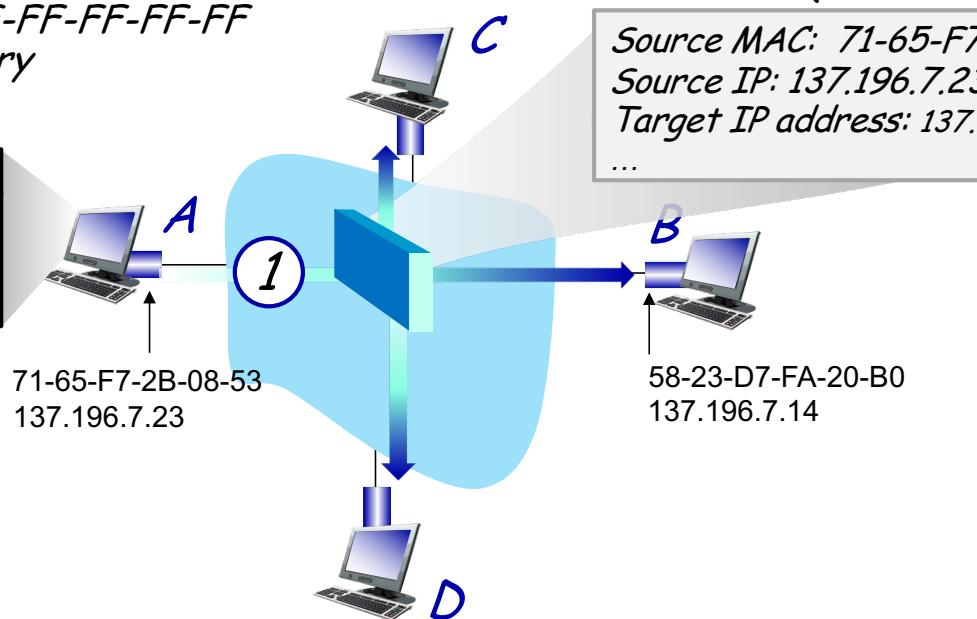
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

A broadcasts ARP query, containing B's IP addr

- destination MAC address = FF-FF-FF-FF-FF-FF
- all nodes on LAN receive ARP query

1

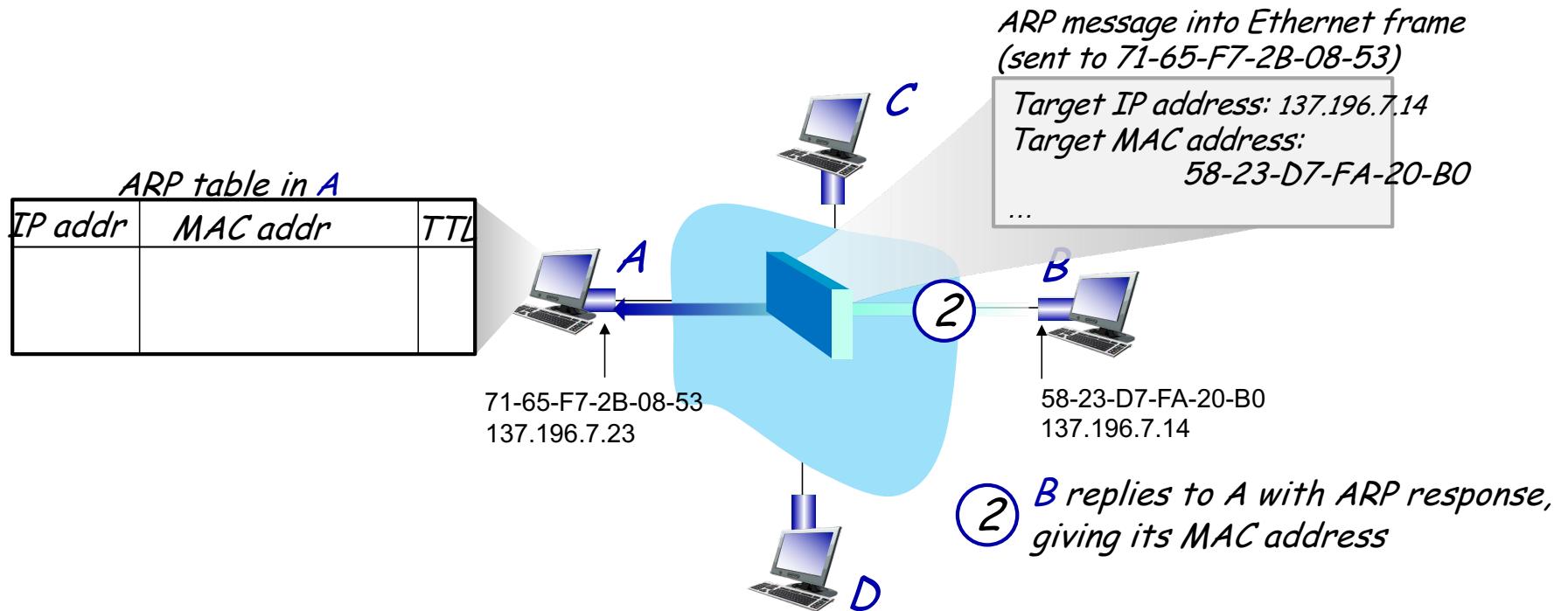
| IP addr | MAC addr | TTL |
|---------|----------|-----|
|         |          |     |



# ARP protocol in action

example: A wants to send datagram to B

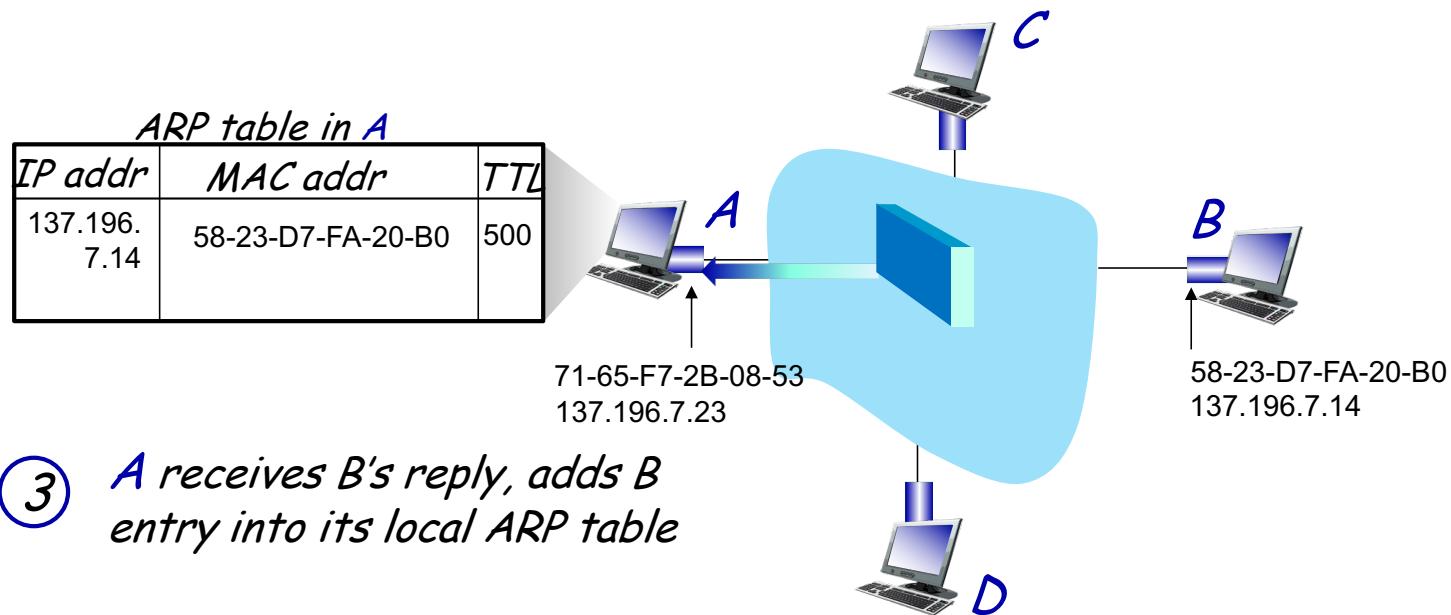
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



# 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

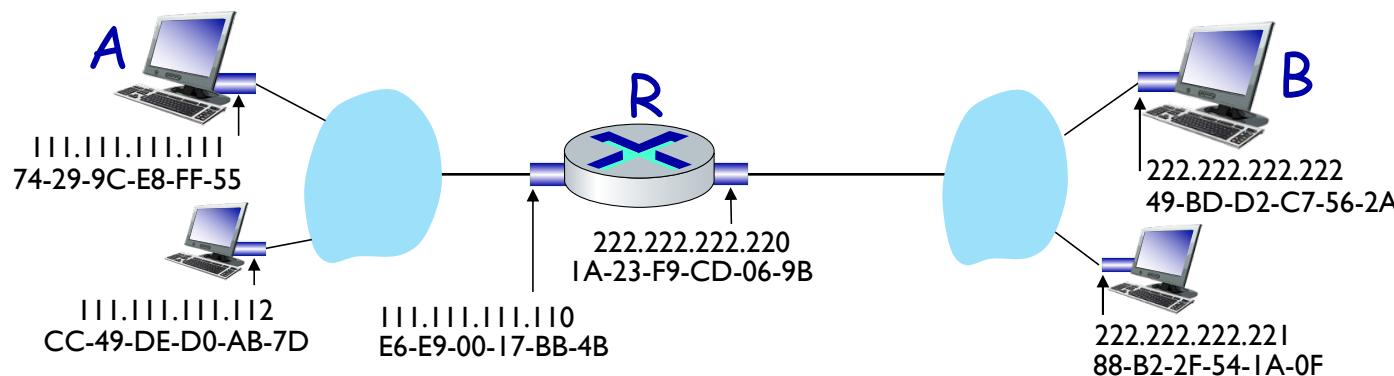


- ③ A receives B's reply, adds B entry into its local ARP table

# Routing to another subnet: addressing

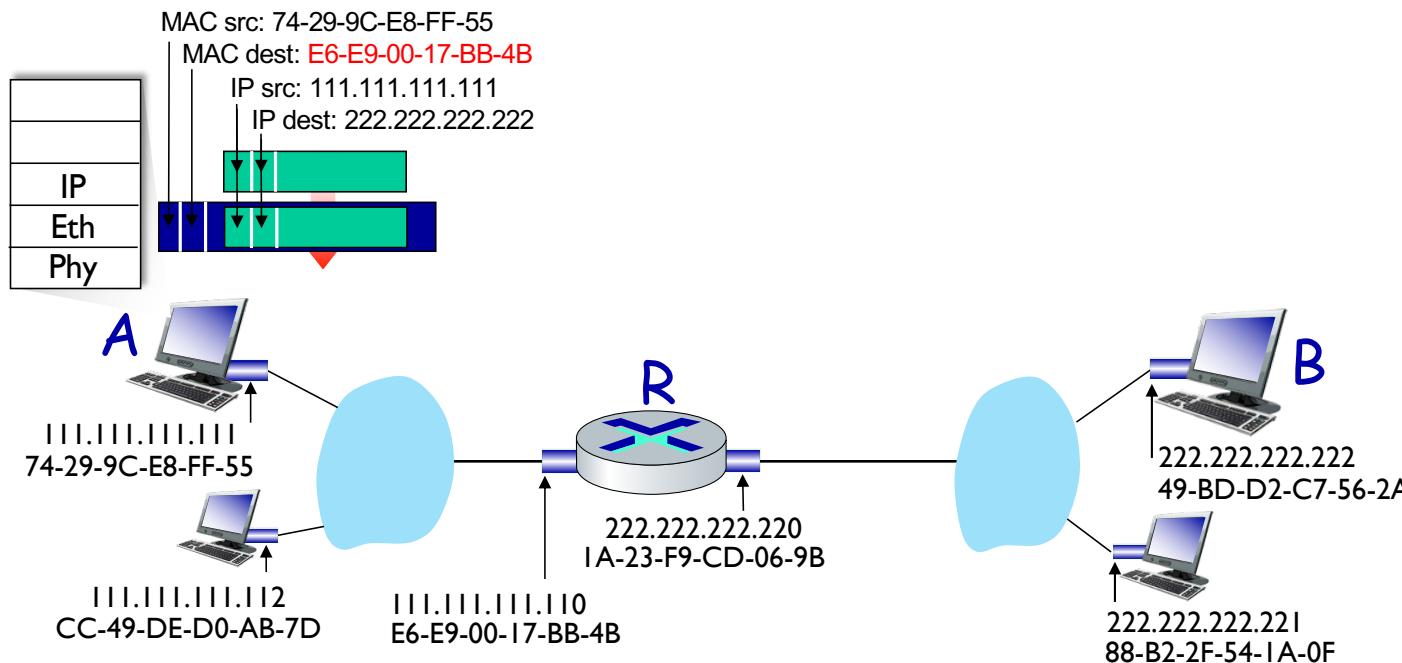
walkthrough: sending a datagram from A to B via R

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



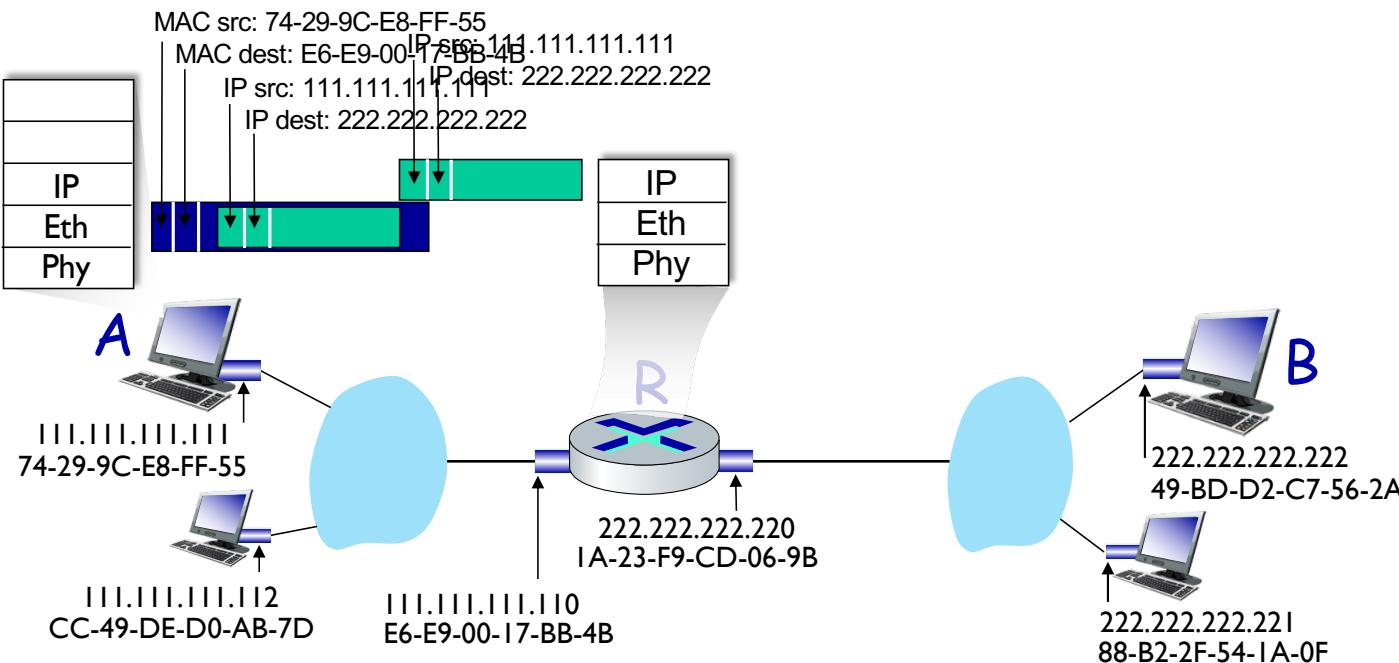
# Routing to another subnet: addressing

- 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



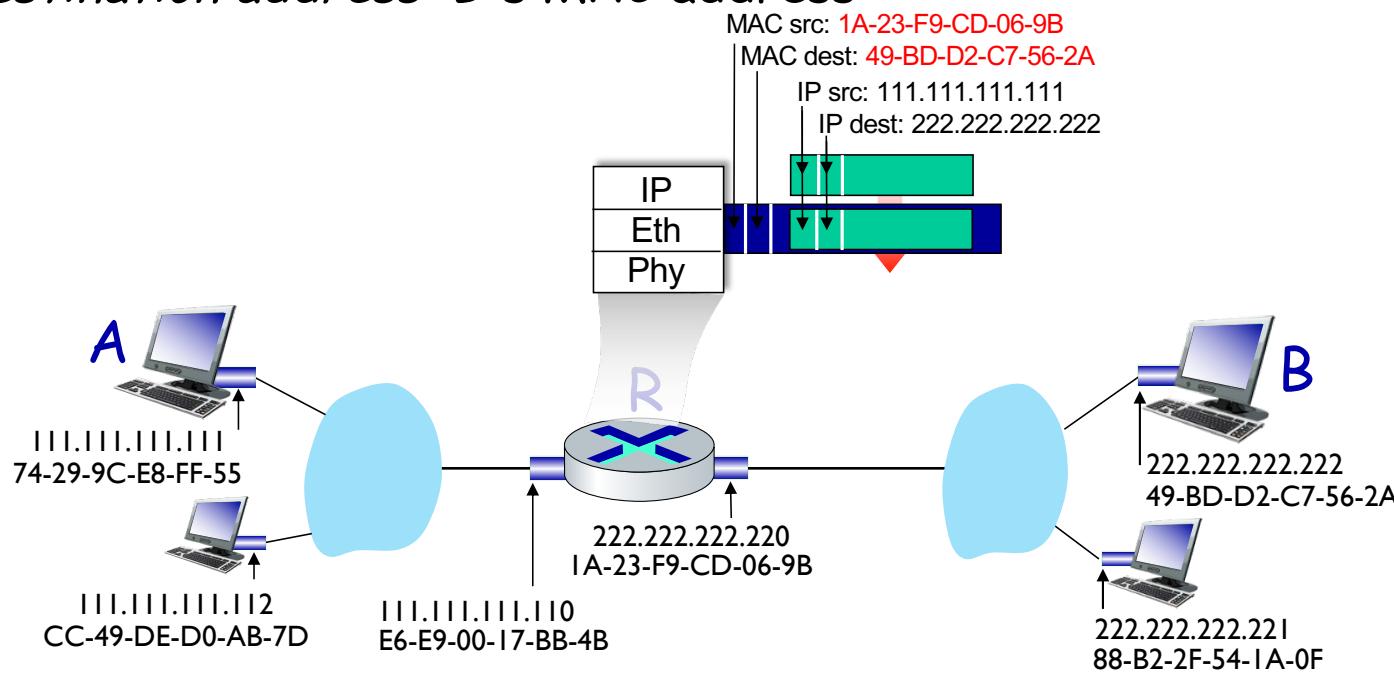
# Routing to another subnet: addressing

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



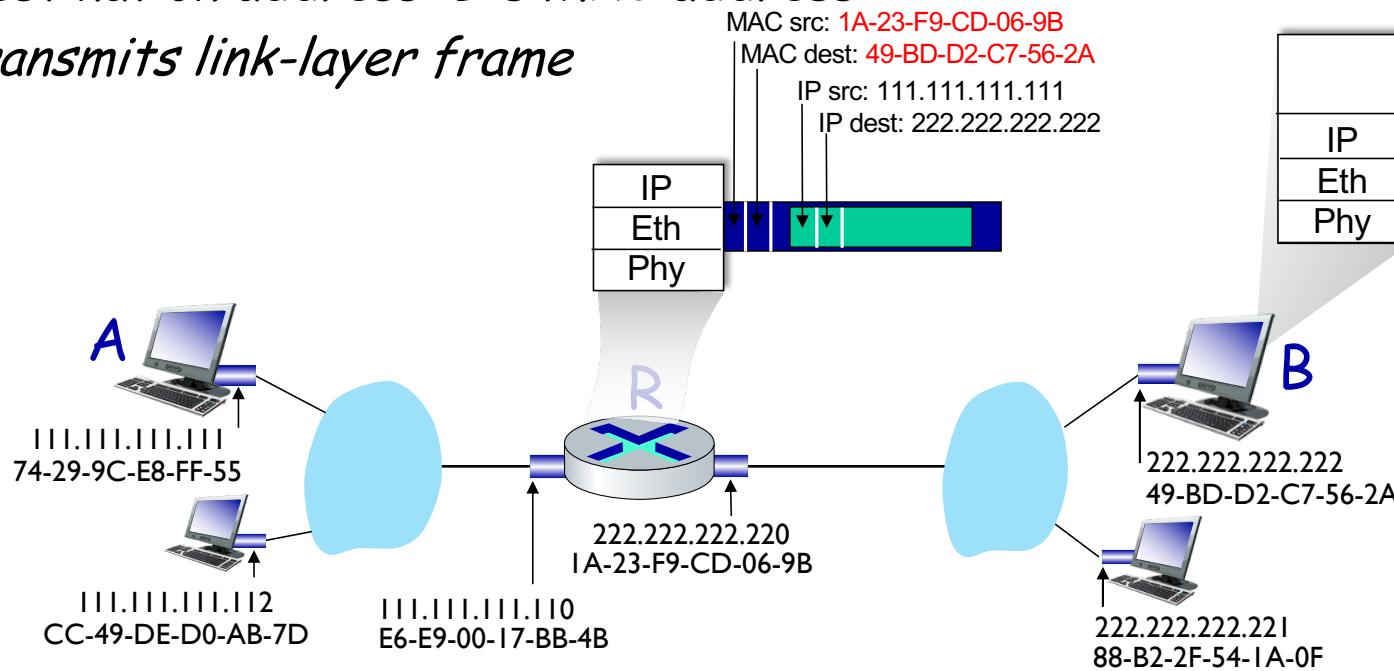
# Routing to another subnet: addressing

- 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



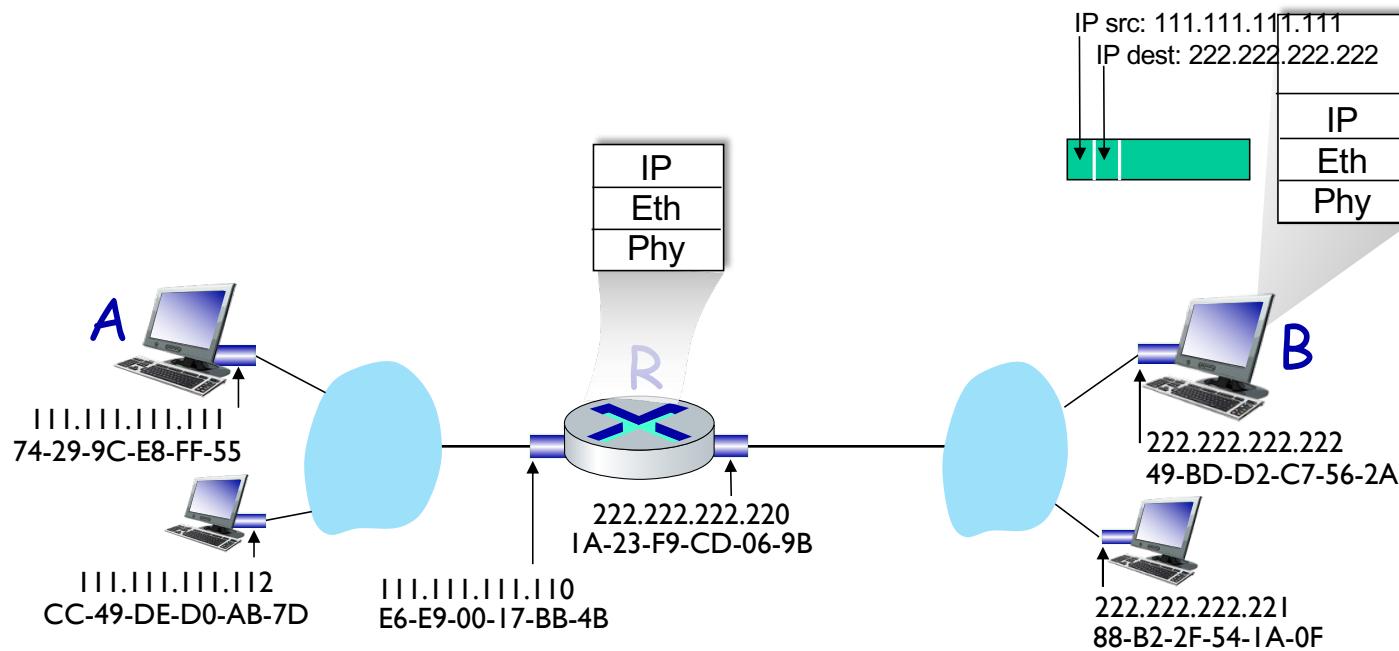
# Routing to another subnet: addressing

- 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 passes datagram up protocol stack to IP



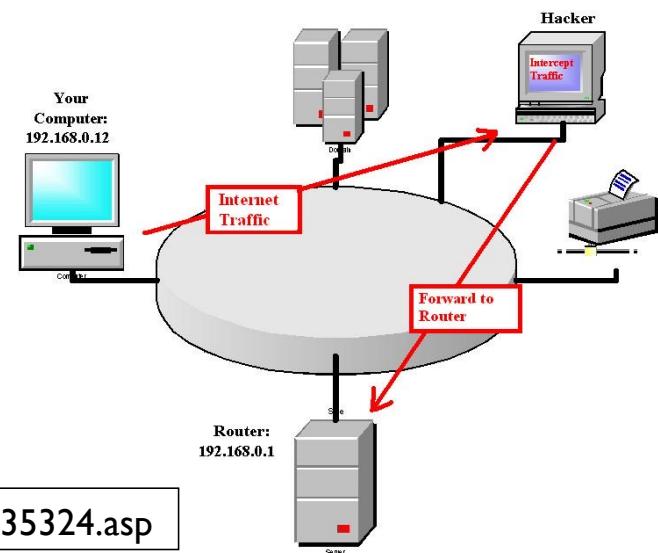
# Security Issues: ARP Cache Poisoning



- ❖ Denial of Service - Hacker replies back to an ARP query for a router NIC with a fake MAC address
- ❖ Man-in-the-middle attack - Hacker can insert his/her machine along the path between victim machine and gateway router
- ❖ Such attacks are generally hard to launch as hacker needs physical access to the network

## Solutions -

- Use Switched Ethernet with port security enabled (i.e., one host MAC address per switch port)
- Adopt static ARP configuration for small size networks
- Use ARP monitoring tools such as ARPWatch



<http://www.watchguard.com/infocenter/editorial/135324.asp>

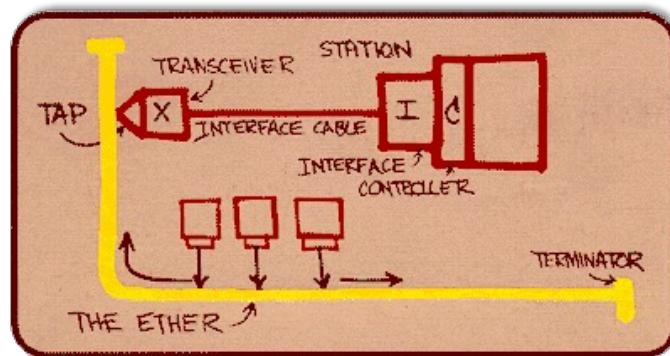
# Link layer, LANs: roadmap

- *introduction*
- *error detection, correction*
- *multiple access protocols*
- *LANs*
  - *addressing, ARP*
  - **Ethernet**
  - *switches*
  - *VLANs (NOT COVERED)*
- *link virtualization: MPLS (NOT COVERED)*
- *data center networking (NOT COVERED)*
- *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., Broadcom BCM5761)



Metcalfe's Ethernet sketch

<https://www.uspto.gov/learning-and-resources/journeys-innovation/audio-stories/defying-doubters>

# 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)



# Ethernet frame structure

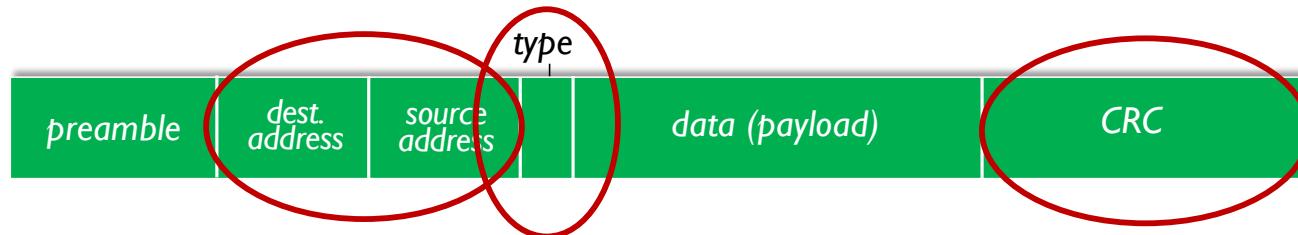
*sending interface encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame***



***preamble:***

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

## Ethernet frame structure (more)



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

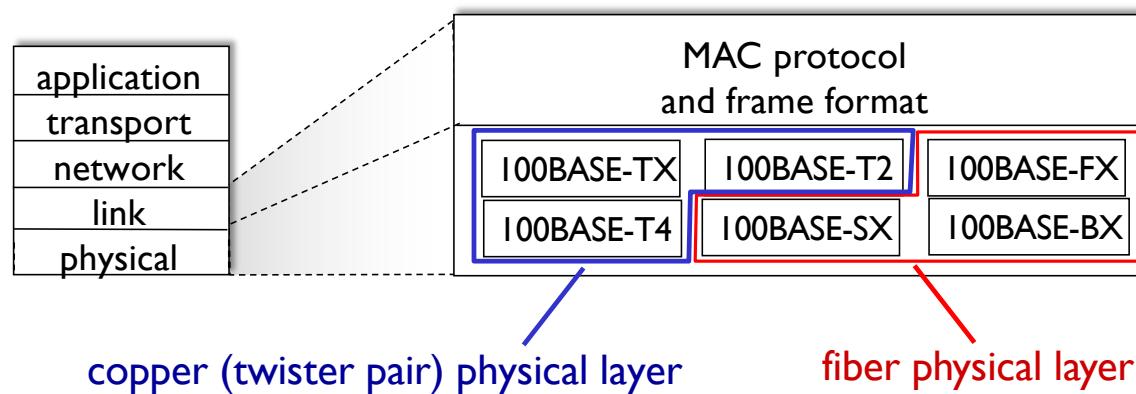
# Ethernet: unreliable, connectionless

- **connectionless:** no handshaking between sending and receiving NICs
- **unreliable:** receiving NIC doesn't send ACKs or NAKs to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted CSMA/CD with binary backoff

## 802.3 Ethernet standards: link & physical layers

NOT ON EXAM

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



# Link layer, LANs: roadmap

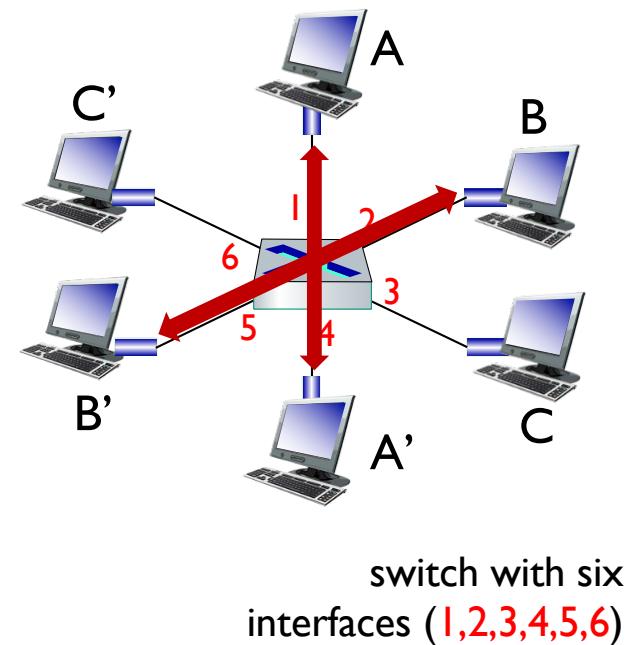
- *introduction*
- *error detection, correction*
- *multiple access protocols*
- *LANs*
  - *addressing, ARP*
  - *Ethernet*
  - **switches**
  - *VLANs (NOT COVERED)*
- *link virtualization: MPLS (NOT COVERED)*
- *data center networking (NOT COVERED)*
- *a day in the life of a web request*

# Ethernet switch

- Switch is a *link-layer* device: it takes an *active role*
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, *selectively* forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- transparent: hosts unaware of presence of switches
- plug-and-play, self-learning
  - switches do not need to be configured

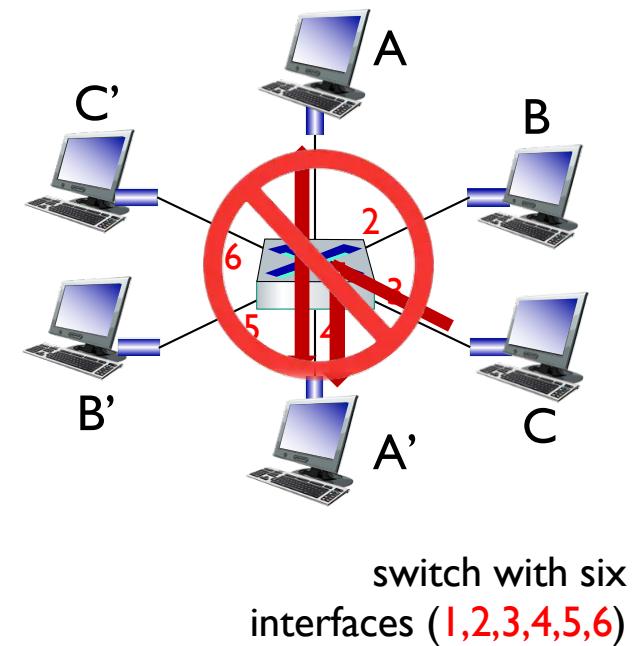
# Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on each incoming link, 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: 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
  - but A-to-A' and C to A' can not happen simultaneously



# Switch forwarding table

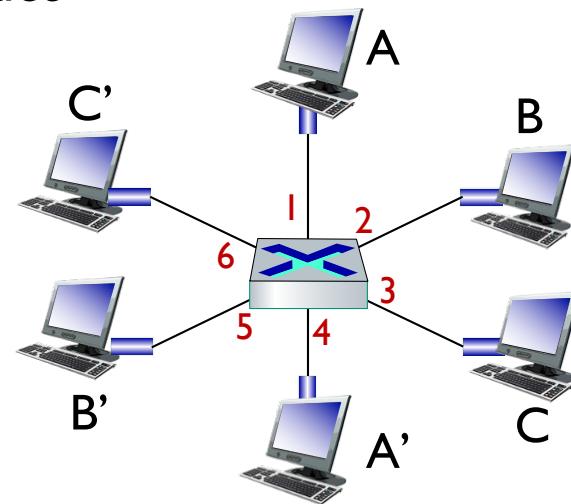
Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

A: each switch has a *switch table*, each entry:

- (MAC address of host, interface to reach host, time stamp)
- looks like a *routing table*!

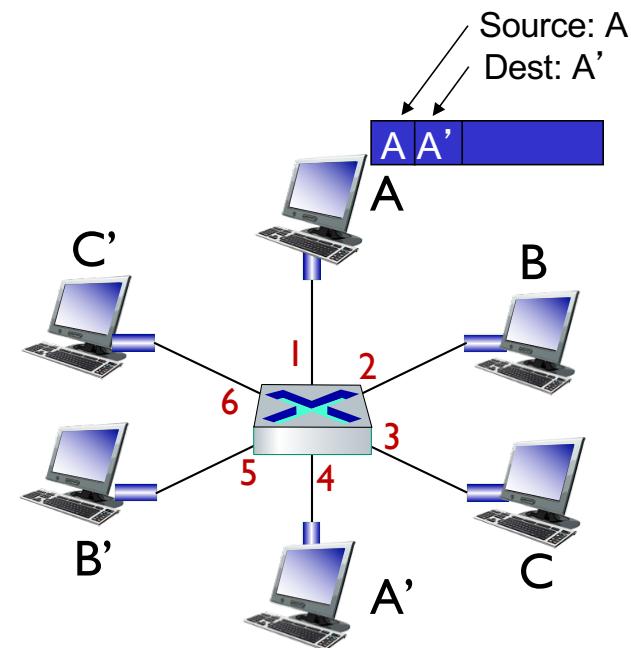
Q: how are entries created, maintained in switch table?

- something like a *routing protocol*?



# Switch: self-learning

- switch **learns** which hosts can be reached through which interfaces
  - when frame received, switch “learns” location of sender: incoming LAN segment
  - records sender/location pair in switch table



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

Switch table  
(initially empty)

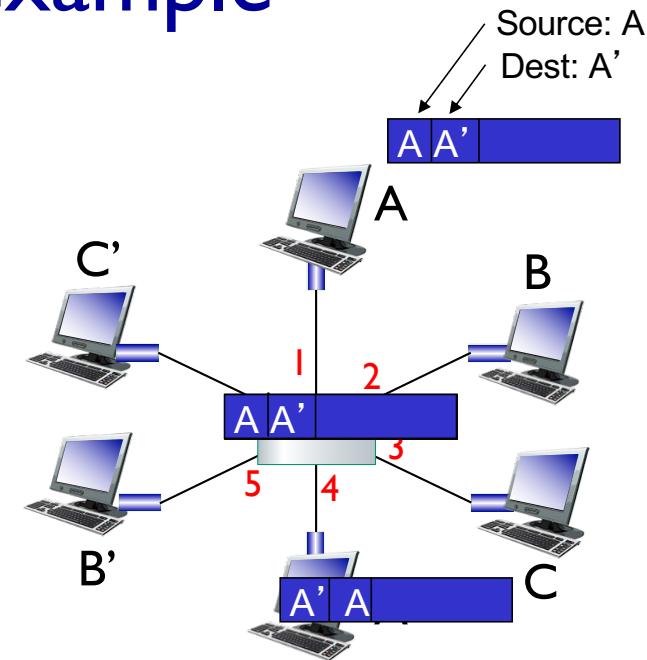
# Switch: frame filtering/forwarding

*when frame received at switch:*

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

## Self-learning, forwarding: example

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

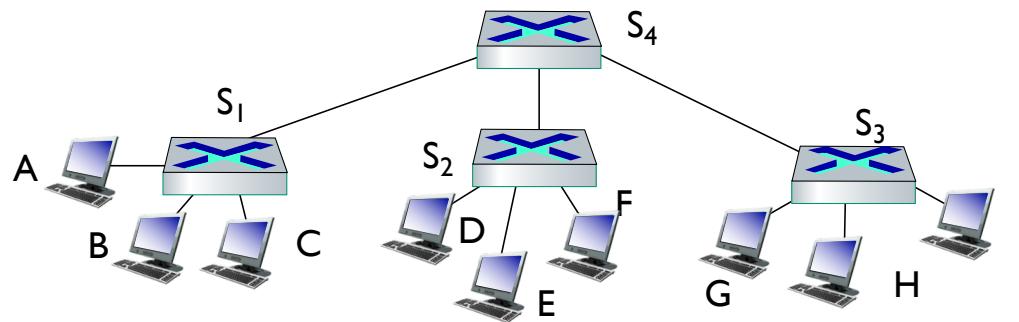


| MAC addr | interface | TTL |
|----------|-----------|-----|
| A        | 1         | 60  |
| A'       | 4         | 60  |

*switch table  
(initially empty)*

# Interconnecting switches

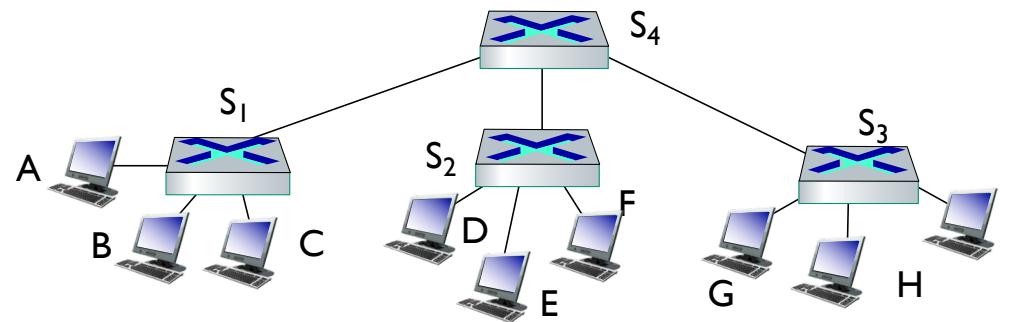
*self-learning switches can be connected together:*



- 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!)

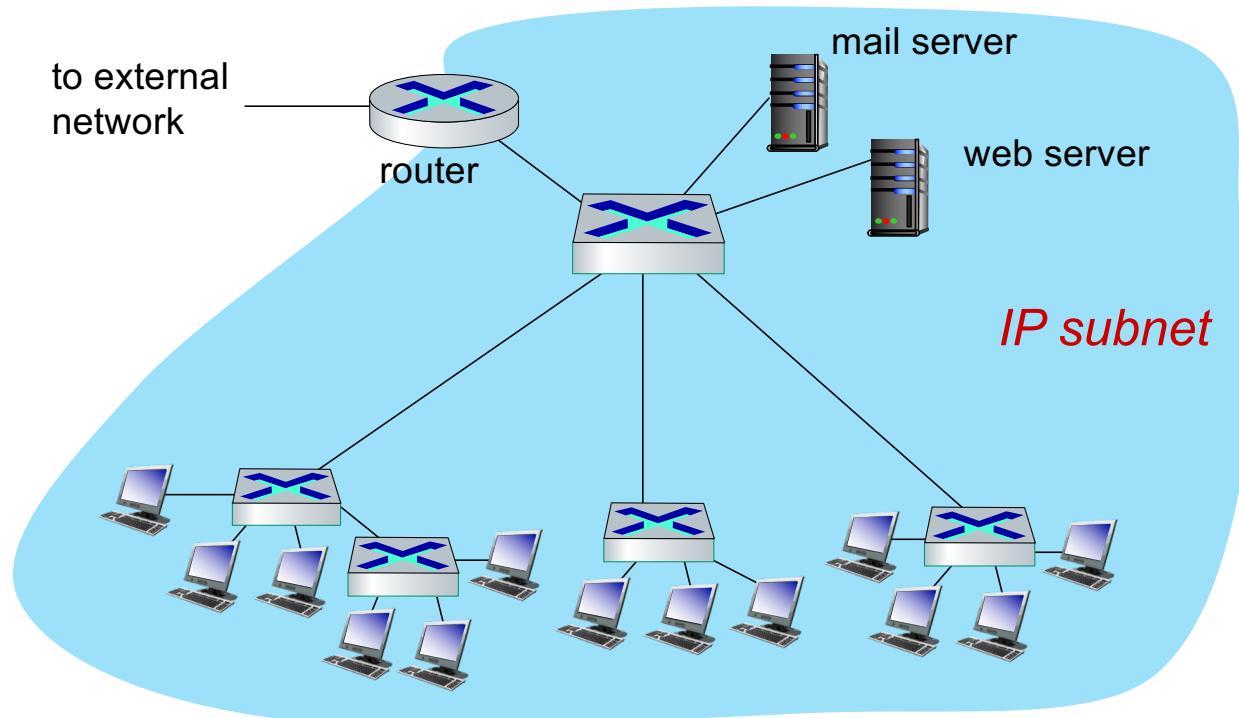
## Self-learning multi-switch example

Suppose C sends frame to I, I responds to C



Q: show switch tables and packet forwarding in  $S_1, S_2, S_3, S_4$

# Small institutional network



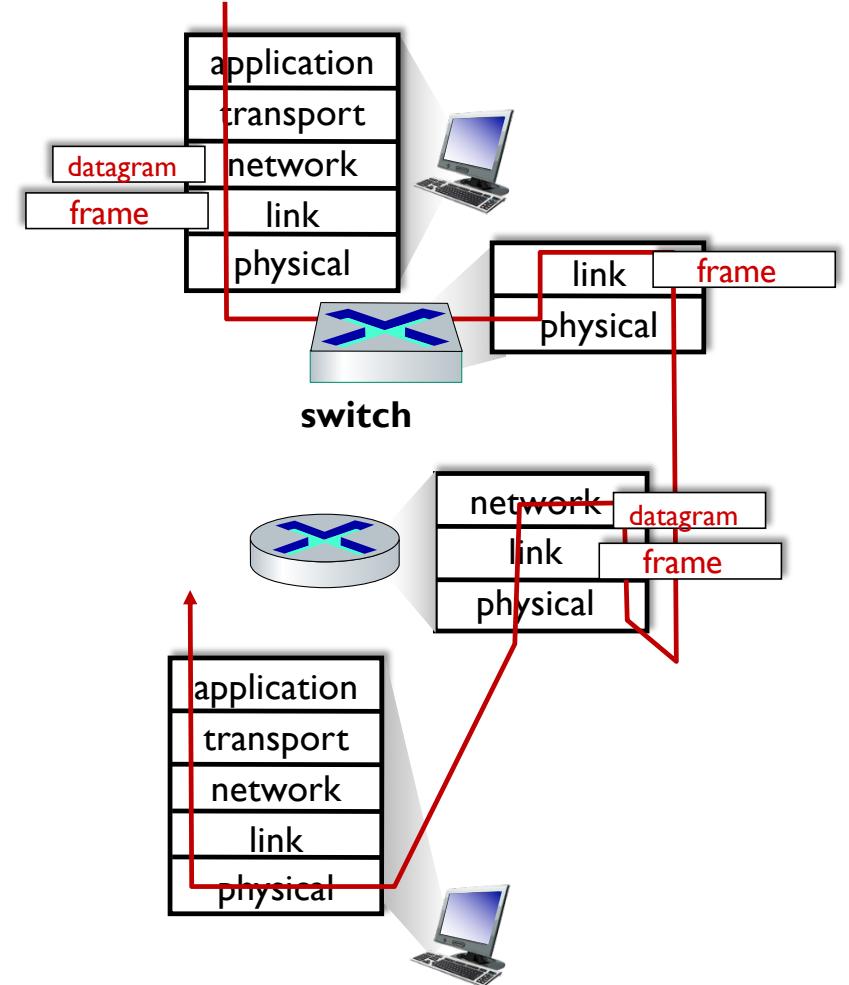
# Switches vs. routers

*both are store-and-forward:*

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

*both have forwarding tables:*

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



# Security Issues

- ❖ In a switched LAN once the switch table entries are established frames are not broadcast
  - Sniffing frames is harder than pure broadcast LANs
  - Note: attacker can still sniff broadcast frames and frames for which there are no entries (as they are broadcast)
- ❖ Switch Poisoning: Attacker fills up switch table with bogus entries by sending large # of frames with bogus source MAC addresses
- ❖ Since switch table is full, genuine packets frequently need to be broadcast as previous entries have been wiped out

# Quiz



- ❖ A switch can
  - A. Filter a frame
  - B. Forward a frame
  - C. Extend a LAN
  - D. All of the above

# Quiz



- ❖ The \_\_\_\_\_ will typically change from link to link,  
but the \_\_\_\_\_ will typically remain the same
  - A. Source MAC address, destination MAC address
  - B. Source IP address, destination IP address
  - C. Source & destination IP addresses, source & destination  
MAC addresses
  - D. Source & destination MAC addresses, source & destination  
IP addresses

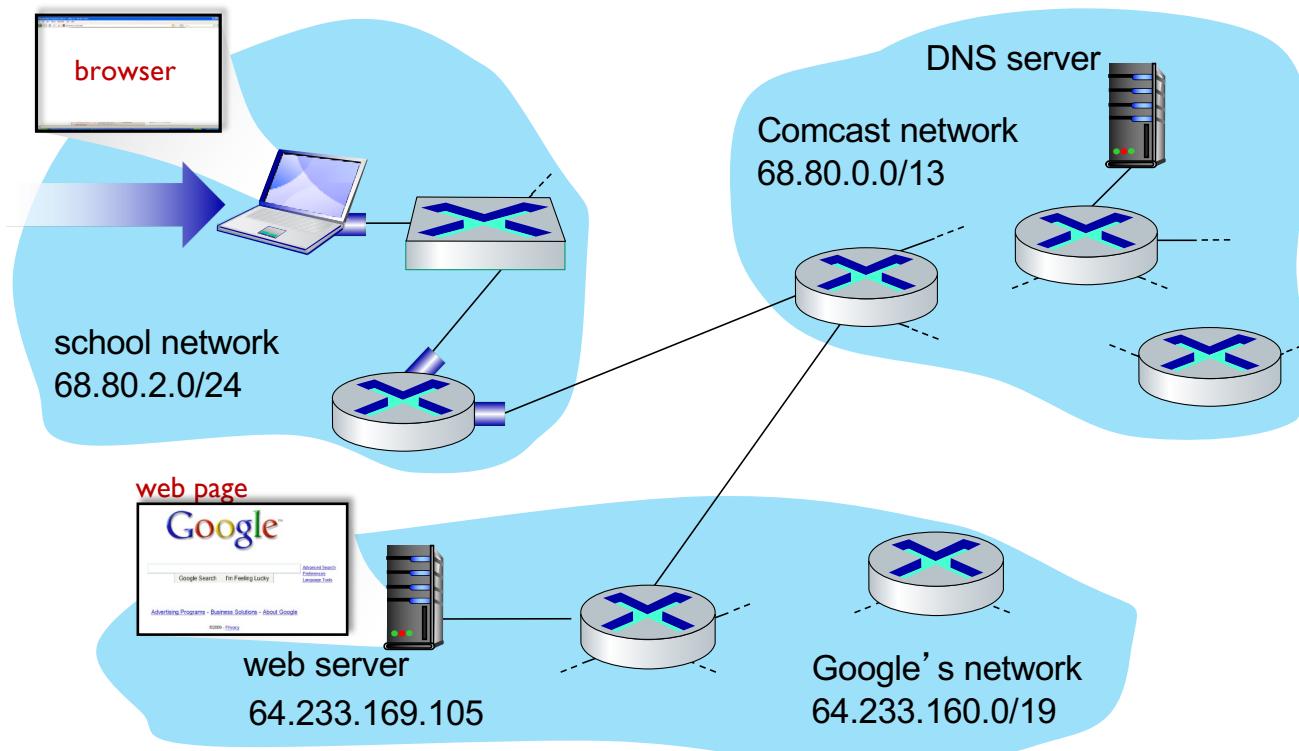
# Link layer, LANs: roadmap

- *introduction*
  - *error detection, correction*
  - *multiple access protocols*
  - *LANs*
    - *addressing, ARP*
    - *Ethernet*
    - *switches*
    - *VLANs (NOT COVERED)*
  - *link virtualization: MPLS (NOT COVERED)*
  - *data center networking (NOT COVERED)*
- ***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

# A day in the life: scenario

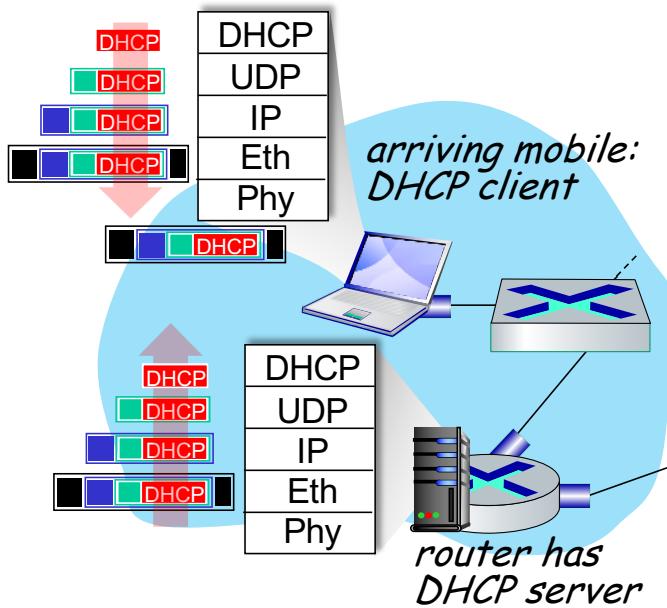


*scenario:*

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

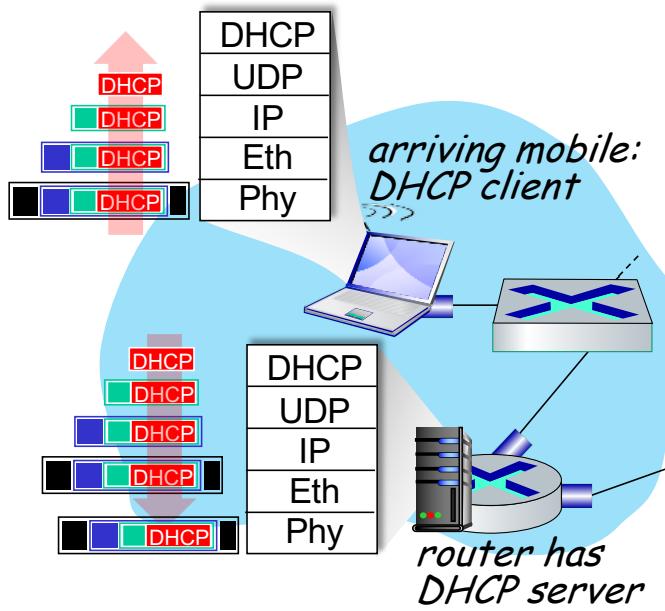
*Sounds simple!* !

# 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: FFFFFFFFFFFF) on LAN, received at router running **DHCP** server*
- *Ethernet demuxed to IP demuxed, UDP demuxed to **DHCP***

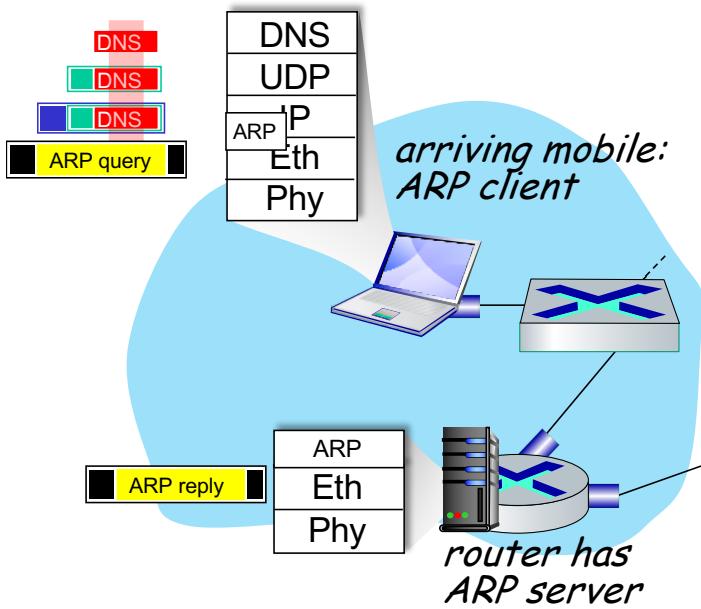
# A day in the life: connecting to the Internet



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

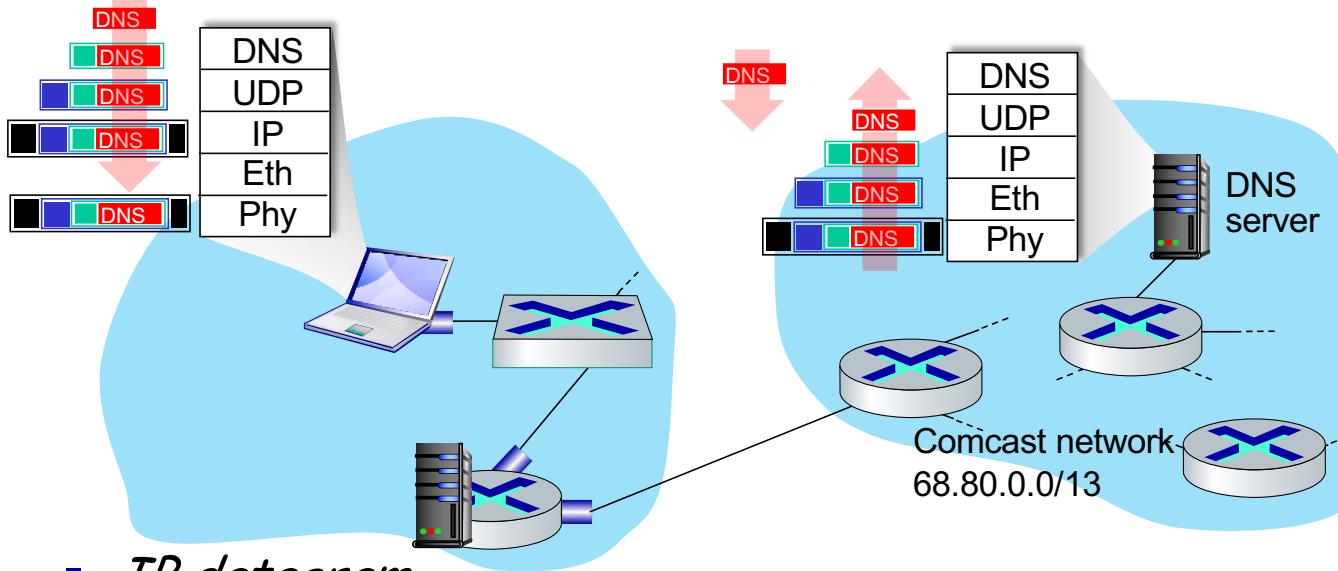
*Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router*

# 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*

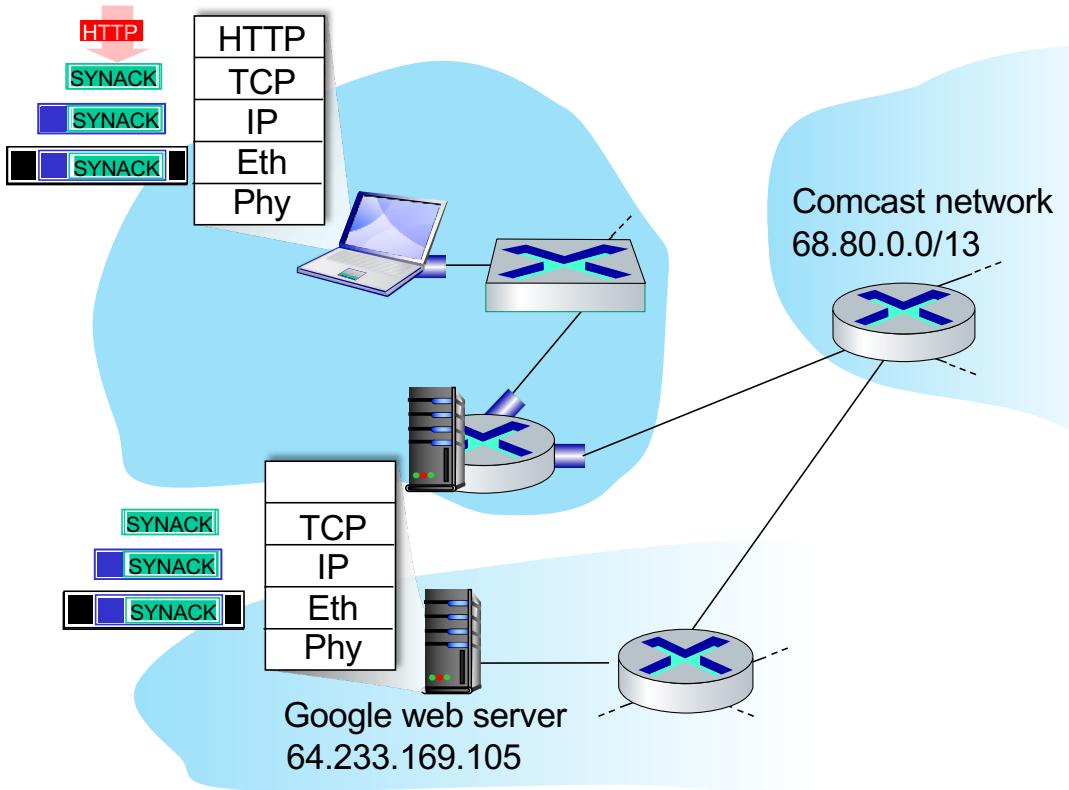
# A day in the life... using DNS



- *demuxed to DNS*
- *DNS replies to client with IP address of www.google.com*

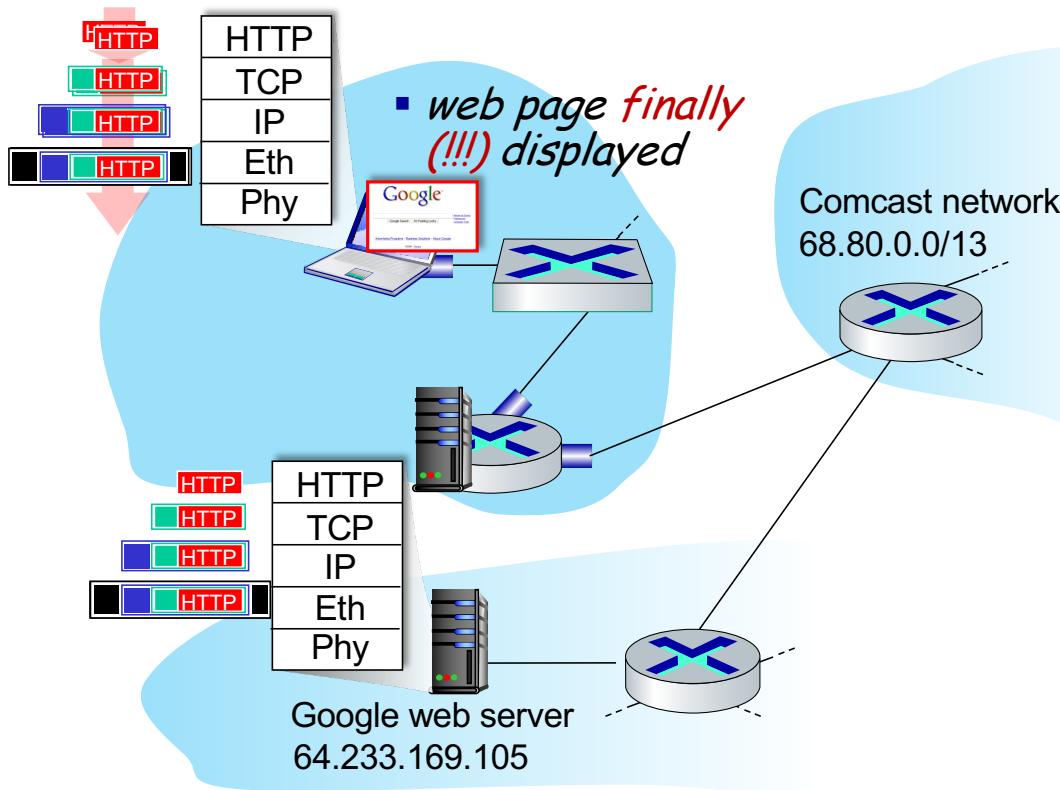
- *IP datagram containing DNS query forwarded via LAN switch from client to 1<sup>st</sup> hop router*
- *IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server*

# 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... HTTP request/reply



- *HTTP request sent into TCP socket*
- *IP datagram containing HTTP request routed to www.google.com*
- *web server responds with HTTP reply (containing web page)*
- *IP datagram containing HTTP reply routed back to client*

# Link Layer: Summary

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

# COMP 3331/9331: Computer Networks and Applications

1. BER, SNR, Data Rate
2. Hidden/Exposed Terminals
3. Channel association
4. CSMA/CA: RTS-CTS
5. Addressing

## Week 10 Wireless Networks

Reading Guide: Chapter 7, Sections 7.1 – 7.3



*Complete your myExperience and shape  
the future of education at UNSW.*

*Click the  Experience link in Moodle*

*or login to [myExperience.unsw.edu.au](https://myexperience.unsw.edu.au)*

*(use z1234567@ad.unsw.edu.au to login)*

*The survey is confidential, your identity will never be released*

*Survey results are not released to teaching staff until after your results are published*

# Wireless and Mobile Networks: context

- *more wireless (mobile) phone subscribers than fixed (wired) phone subscribers (10-to-1 in 2019)!*
- *more mobile-broadband-connected devices than fixed-broadband-connected devices (5-1 in 2019)!*
  - *4G/5G cellular networks now embracing Internet protocol stack, including SDN*
- *two important (but different) challenges*
  - **wireless:** communication over wireless link
  - **mobility:** handling the mobile user who changes point of attachment to network

*We will only focus on wireless challenges*

# Outline

## 7.1 Introduction

### Wireless

## 7.2 Wireless links, characteristics

## 7.3 IEEE 802.11 wireless LANs (“Wi-Fi”)

# Wireless 101

- **Frequency/Wave-Length –**

C is the speed of light

f is frequency

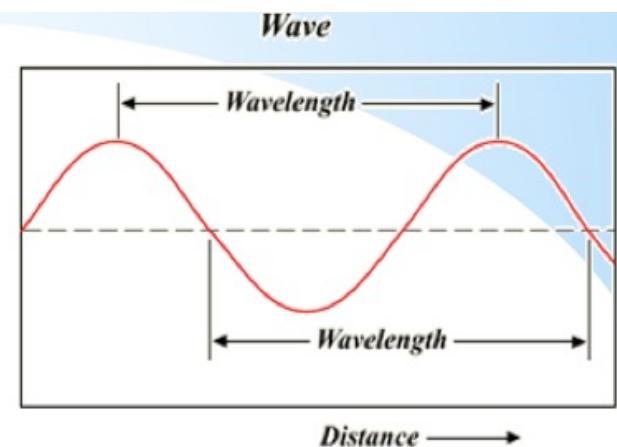
$\lambda$  (lambda) is wavelength

Wavelength

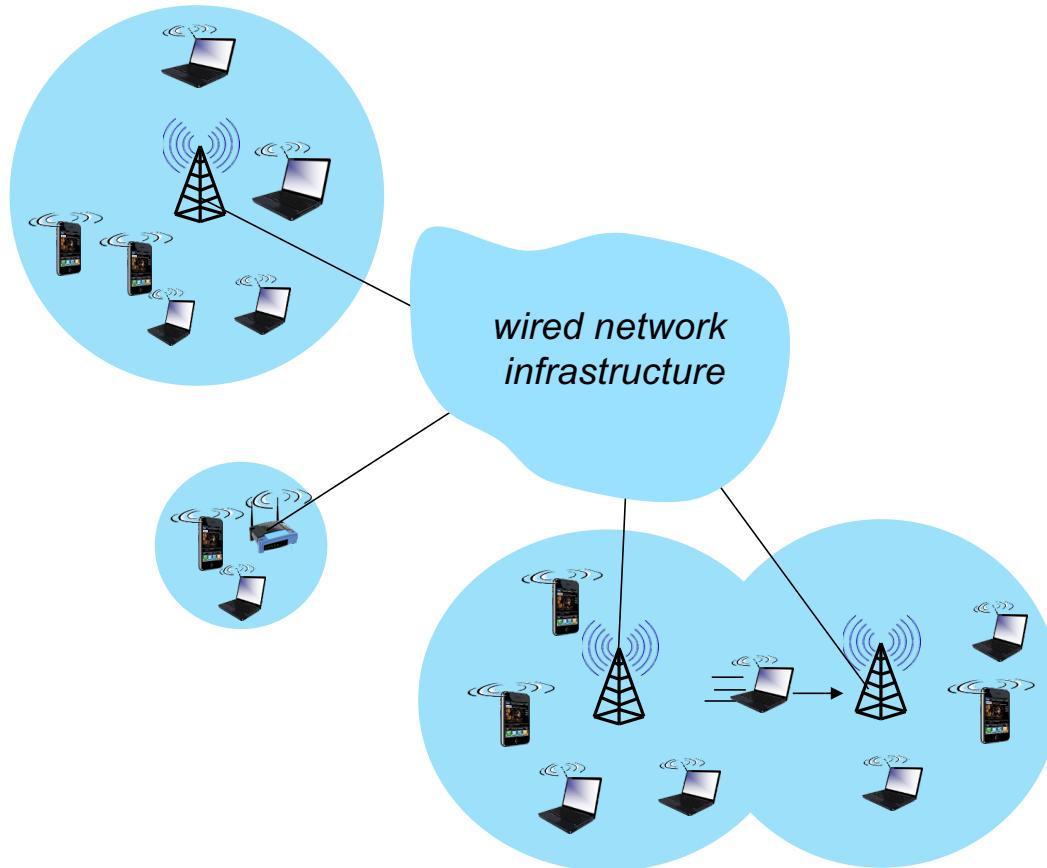
$$\lambda = \frac{C}{f}$$

Frequency

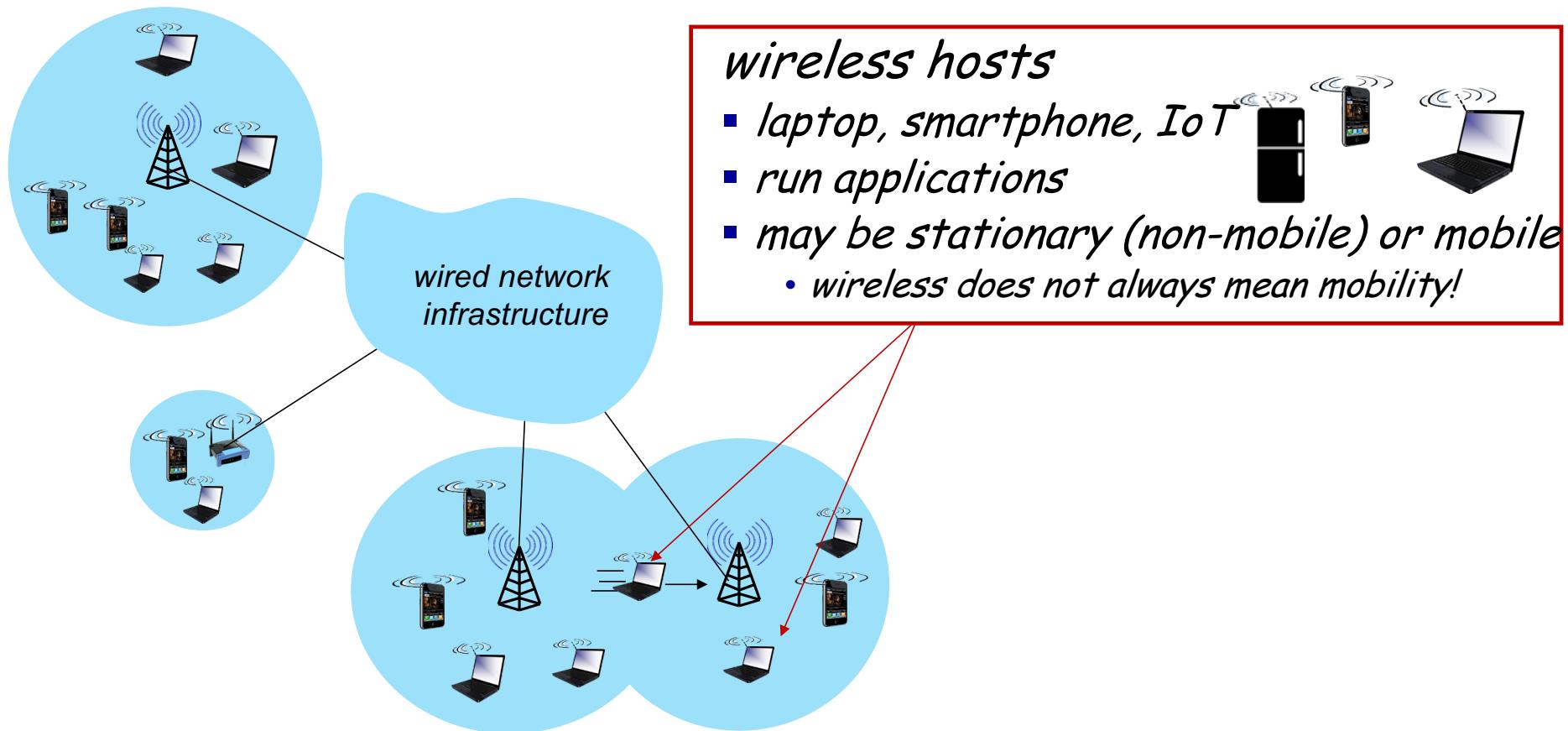
$$f = \frac{C}{\lambda}$$



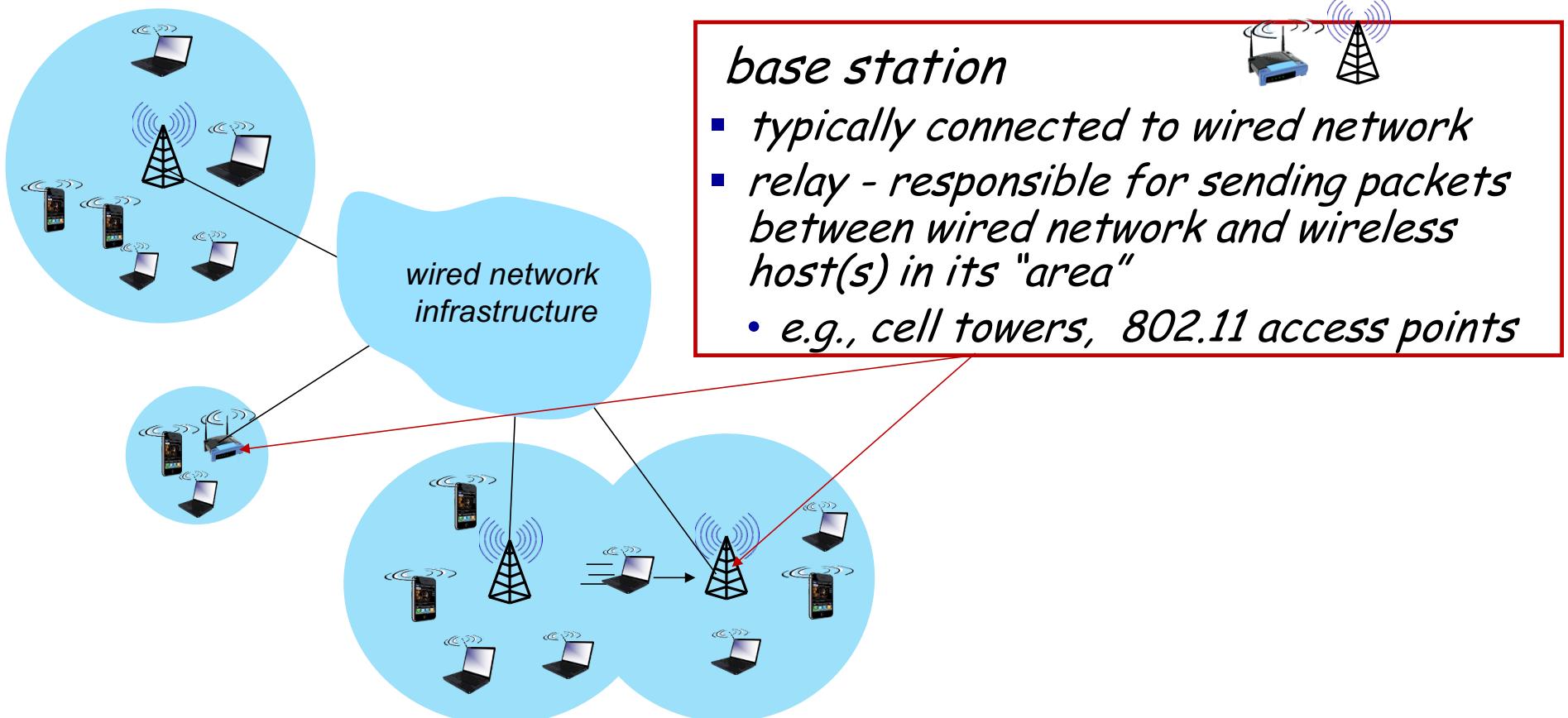
# Elements of a wireless network



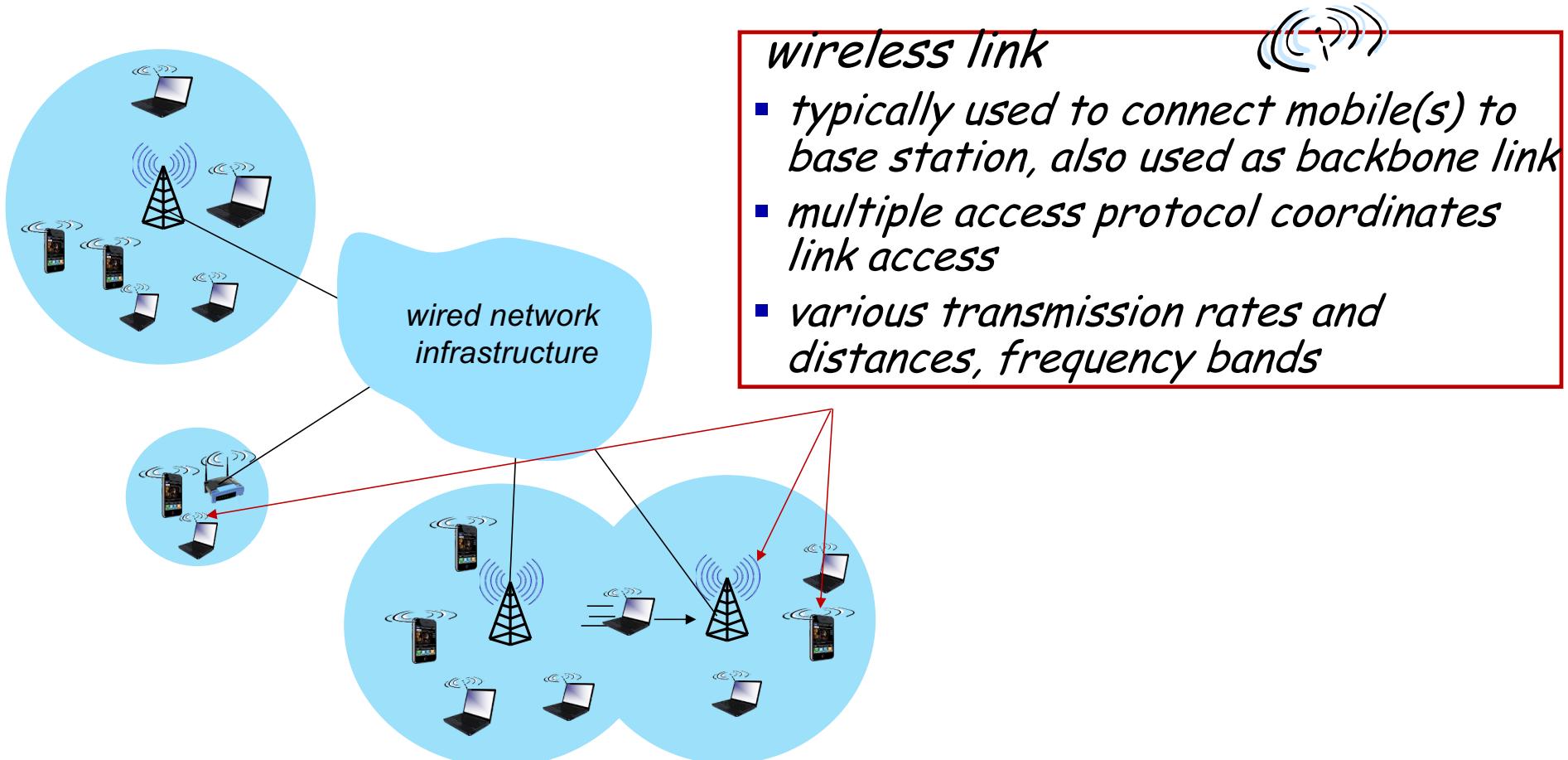
# Elements of a wireless network



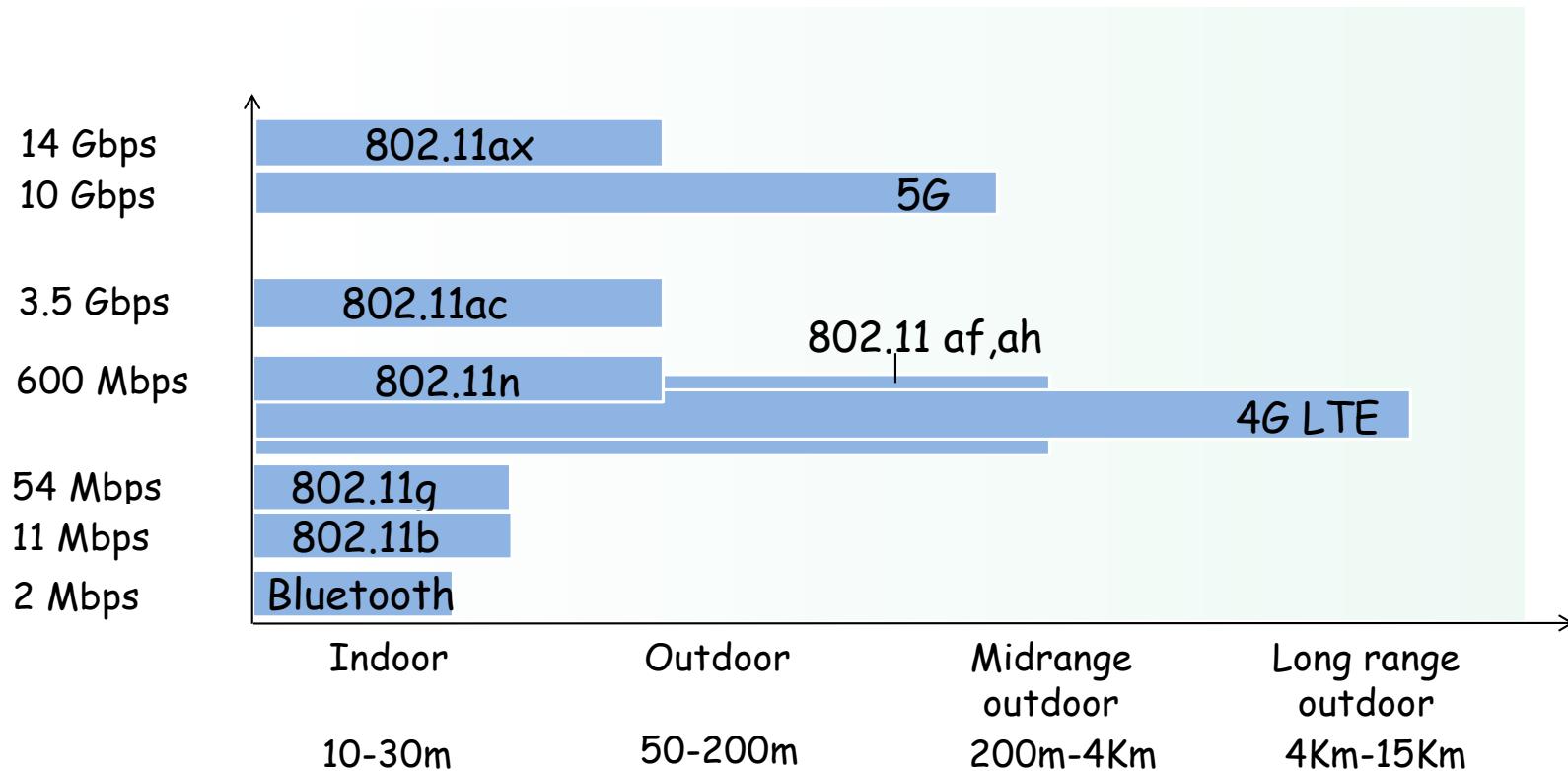
# Elements of a wireless network



# Elements of a wireless network

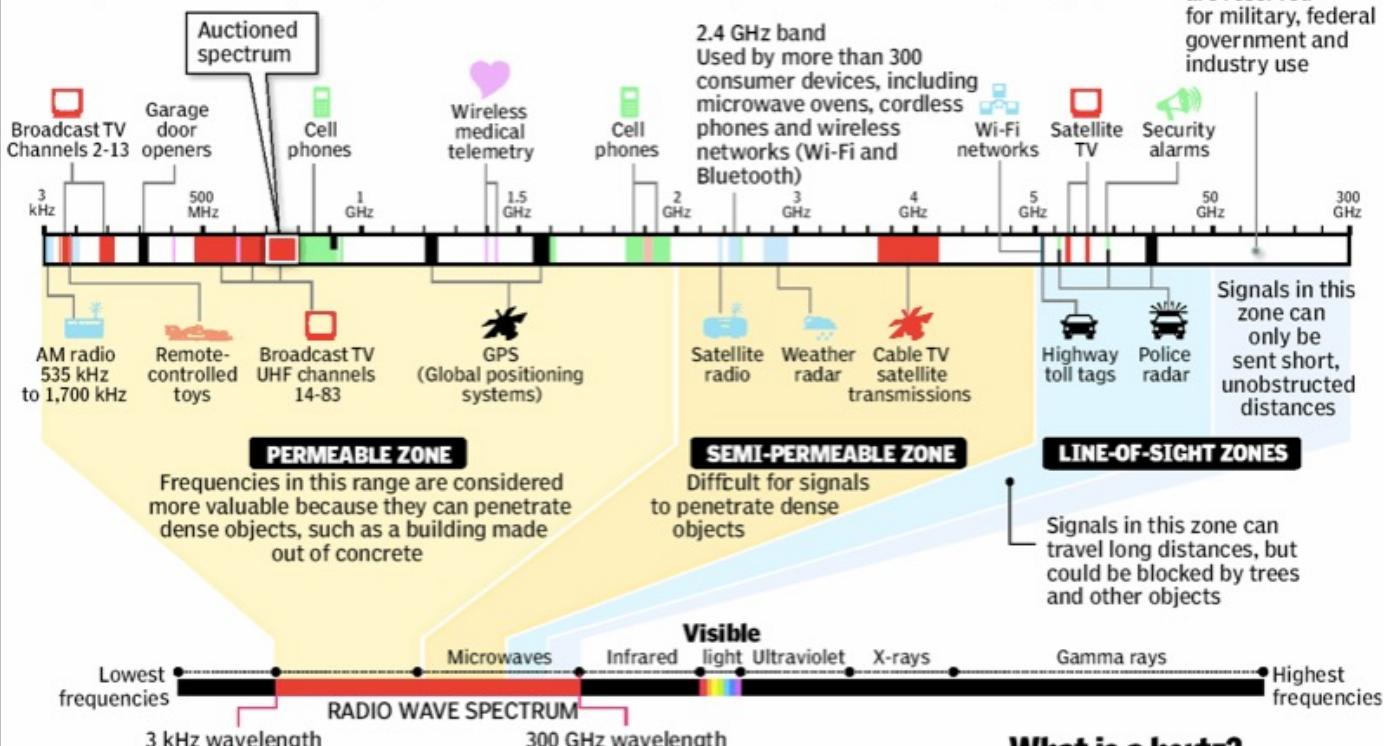


# Characteristics of selected wireless links



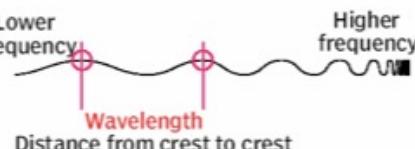
# Inside the radio wave spectrum

Almost every wireless technology – from cell phones to garage door openers – uses radio waves to communicate. Some services, such as TV and radio broadcasts, have exclusive use of their frequency within a geographic area. But many devices share frequencies, which can cause interference. Examples of radio waves used by everyday devices:



## The electromagnetic spectrum

Radio waves occupy part of the electromagnetic spectrum, a range of electric and magnetic waves of different lengths that travel at the speed of light; other parts of the spectrum include visible light and x-rays; the shortest wavelengths have the highest frequency, measured in hertz



## What is a hertz?

One hertz is one cycle per second. For radio waves, a cycle is the distance from wave crest to crest

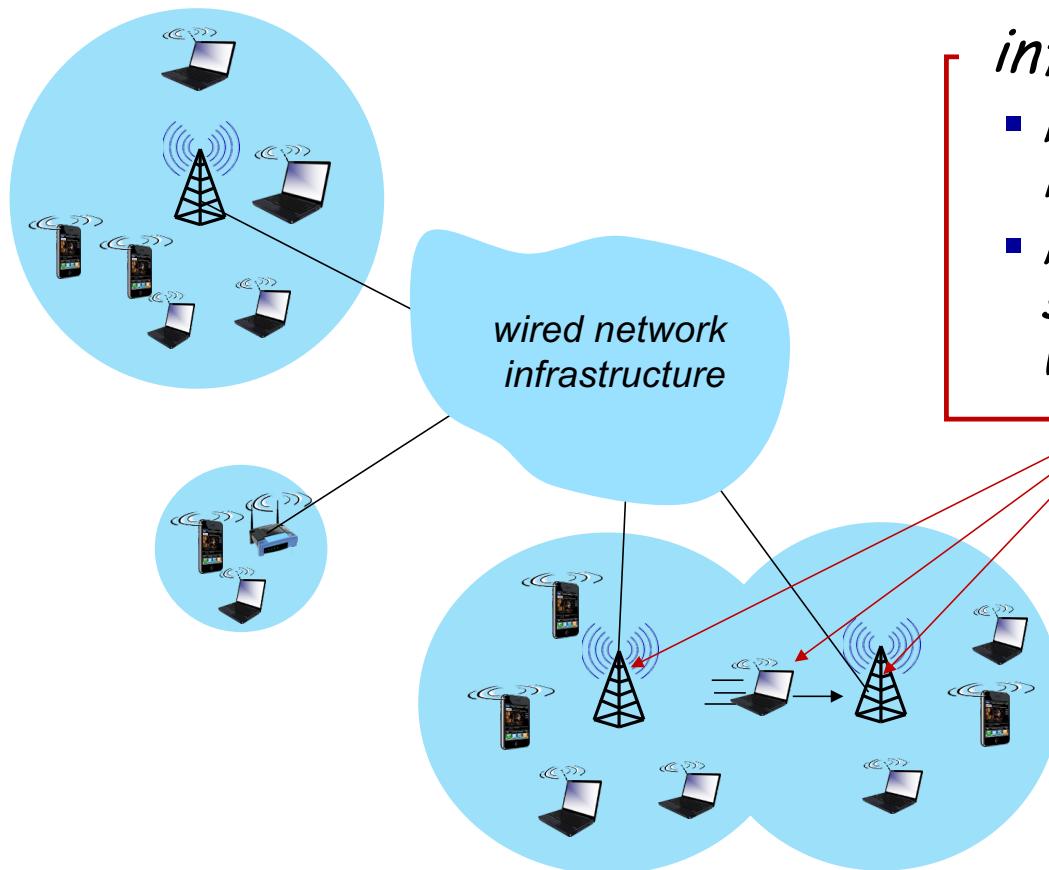
1 kilohertz (kHz) = 1,000 hertz

1 megahertz (MHz) = 1 million hertz

1 gigahertz (GHz) = 1 billion hertz

Source: New America Foundation, MCT, Howstuffworks.com  
Graphic: Nathaniel Levine, Sacramento Bee

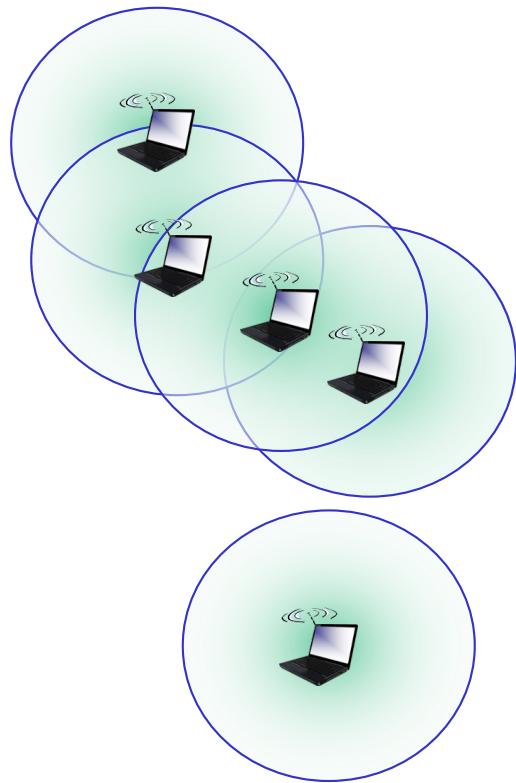
# Elements of a wireless network



## *infrastructure mode*

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

# Elements of a wireless network

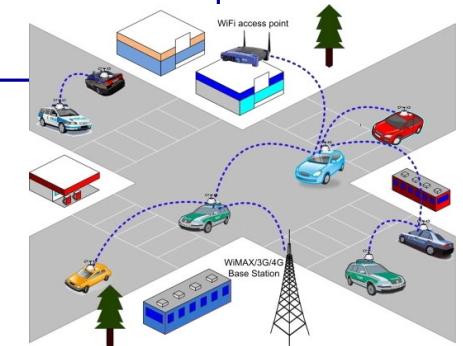


*ad hoc mode*

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

# Wireless network taxonomy

|                                      | <i>single hop</i>                                                                       | <i>multiple hops</i>                                                                                                    |
|--------------------------------------|-----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>infrastructure</i><br>(e.g., APs) | <i>host connects to base station (WiFi, cellular) which connects to larger Internet</i> | <i>host may have to relay through several wireless nodes to connect to larger Internet: mesh net</i>                    |
| <i>no infrastructure</i>             | <i>no base station, no connection to larger Internet (Bluetooth, ad hoc nets)</i>       | <i>no base station, no connection to larger Internet. May have to relay to reach a given wireless node; MANET,VANET</i> |



# Outline

## 7.1 Introduction

### Wireless

## 7.2 Wireless links, characteristics

## 7.3 IEEE 802.11 wireless LANs (“Wi-Fi”)

# Wireless link characteristics (I)

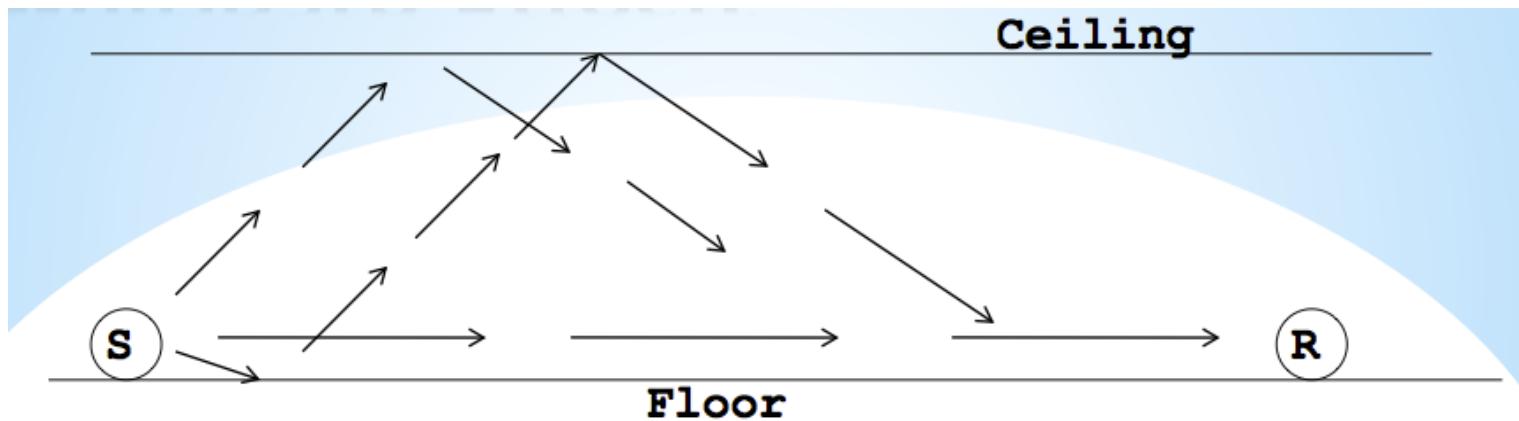
*important differences from wired link ....*

- **decreased signal strength:** radio signal attenuates as it propagates through matter (path loss)
- **interference from other sources:** wireless network frequencies (e.g., 2.4 GHz) shared by many devices (e.g., WiFi, cellular, motors): interference
- **multipath propagation:** radio signal reflects off objects ground, arriving at destination at slightly different times



*.... make communication across (even a point to point) wireless link much more “difficult”*

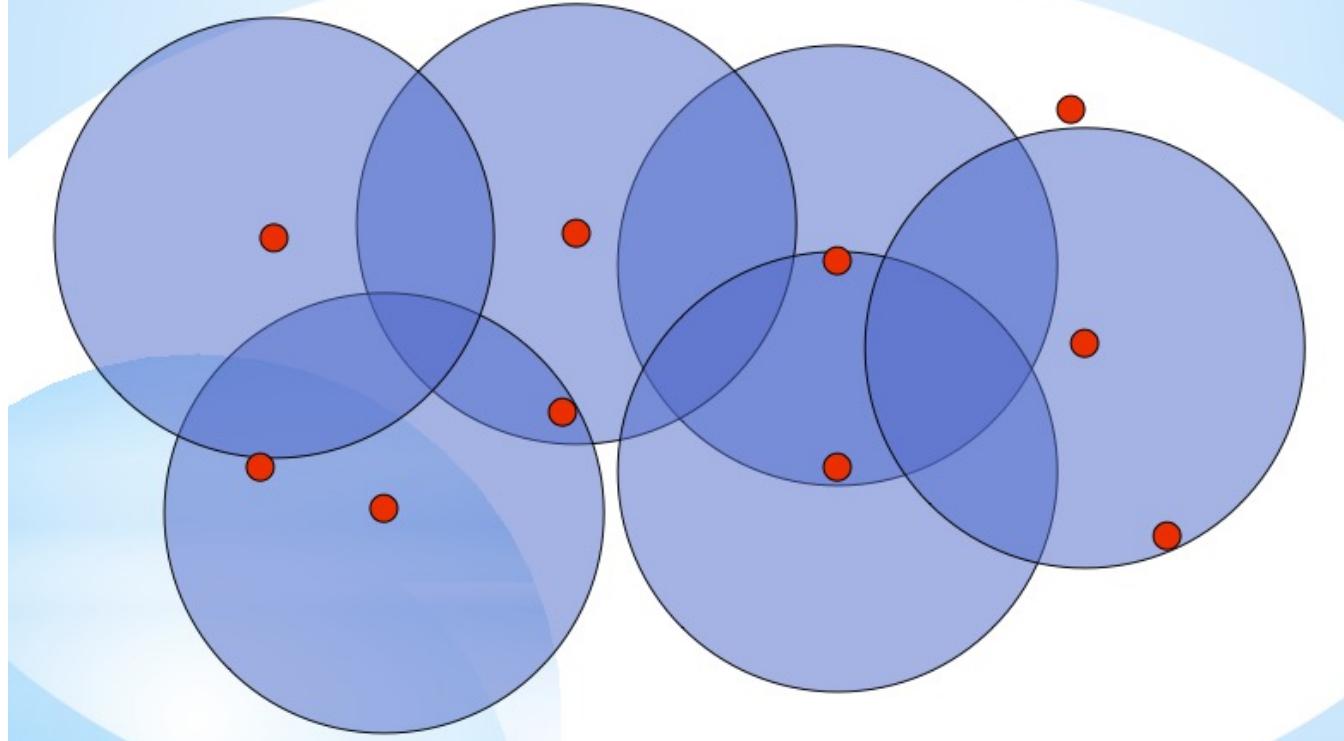
# Multipath Effects



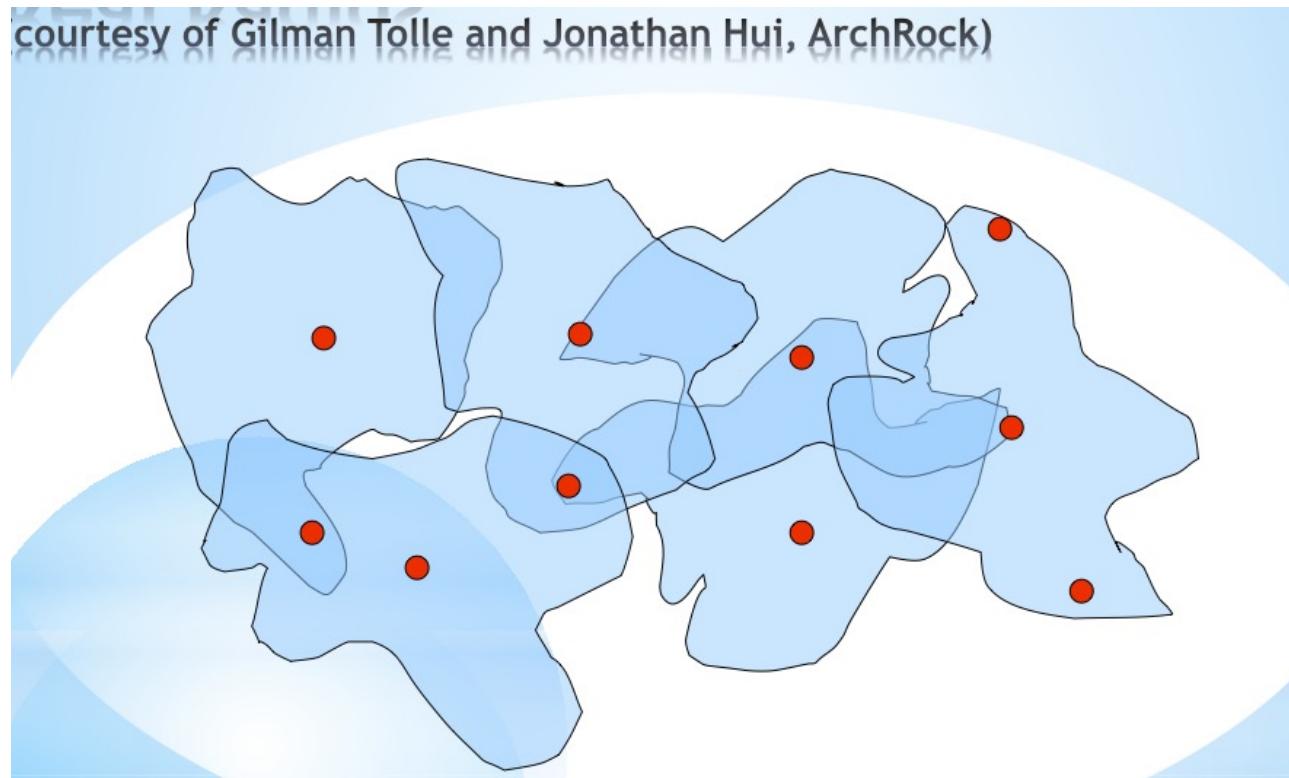
- ❖ Signals bounce off surface and interfere (constructive or destructive) with one another
- ❖ Self-interference

# Ideal Radios

(courtesy of Gilman Tolle and Jonathan Hui, ArchRock)



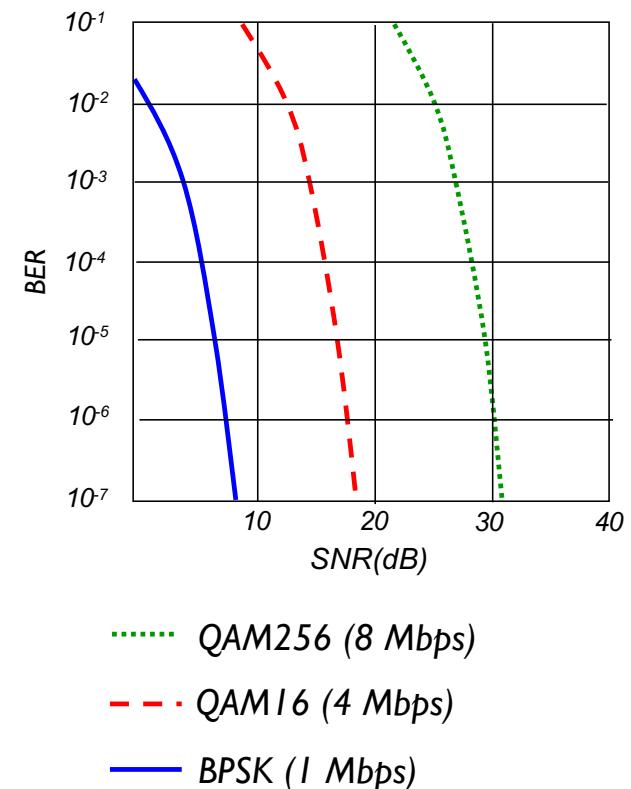
# Real Radios



# Wireless link characteristics (2)

- 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, transmission rate)

*BER: Bit Error Rate*

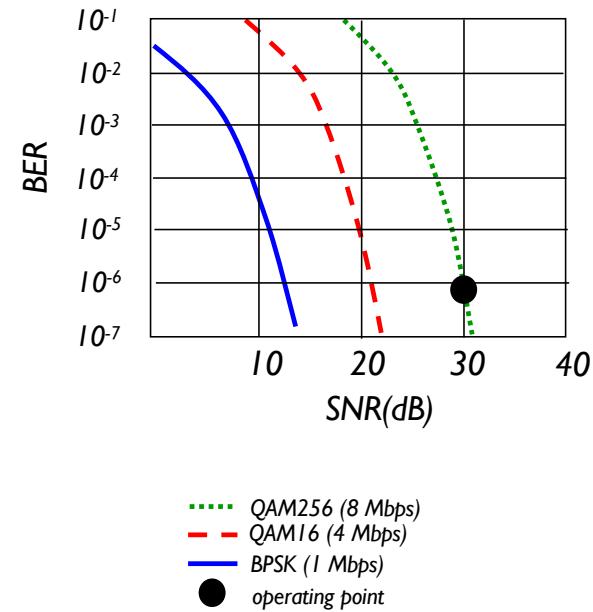


# 802.11: advanced capabilities

## Rate adaptation

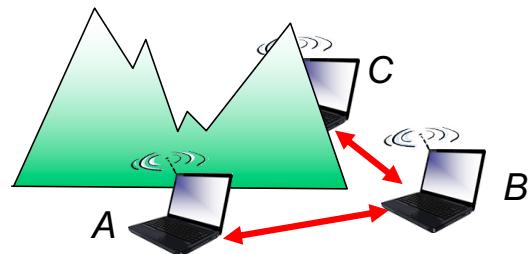
- base station, mobile dynamically change transmission rate (physical layer modulation technique) as mobile moves, SNR varies

- SNR decreases, BER increase as node moves away from base station
- When BER becomes too high, switch to lower transmission rate but with lower BER



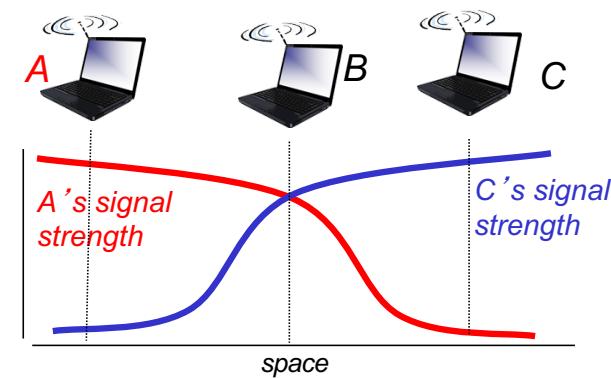
## Wireless link characteristics (3)

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

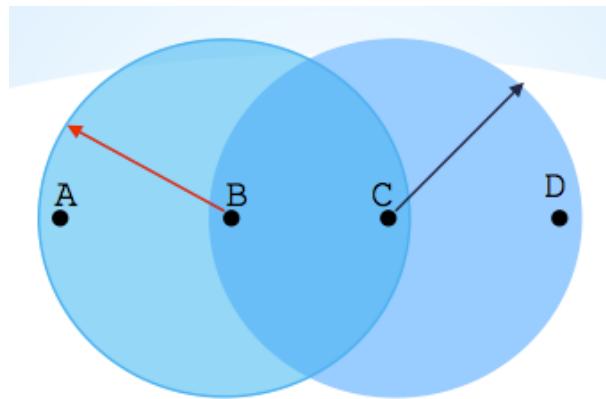


### Signal attenuation:

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

## Wireless network characteristics

- ❖ Exposed Terminals



- ❖ Node B sends a packet to A; C hears this and decides not to send a packet to D (despite the fact that this will not cause interference) !!
- ❖ Carrier sense would prevent a successful transmission

# Outline

7.1 Introduction

Wireless

7.2 Wireless links,  
characteristics

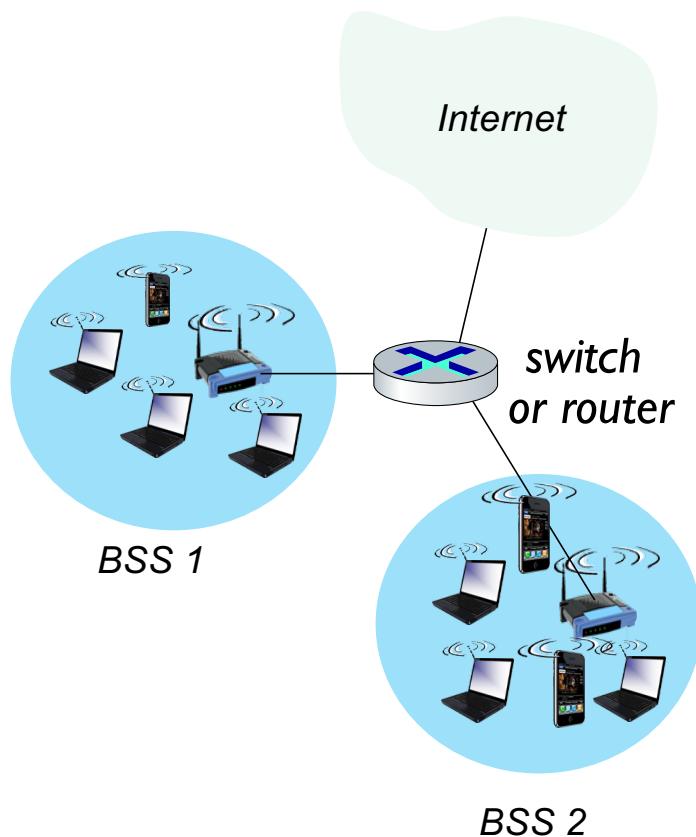
7.3 IEEE 802.11 wireless  
LANs (“Wi-Fi”)

# 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           | 70m   | 2.4, 5 Ghz                   |
| 802.11ac (WiFi 5)     | 2013 | 3.47Gbps      | 70m   | 5 Ghz                        |
| 802.11ax (WiFi 6)     | 2021 | 14 Gbps       | 70m   | 2.4, 5 Ghz                   |
| 802.11af              | 2014 | 35 – 560 Mbps | 1 Km  | unused TV bands (54-790 MHz) |
| 802.11ah (WiFi Halow) | 2017 | 347Mbps       | 1 Km  | 900 Mhz                      |

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

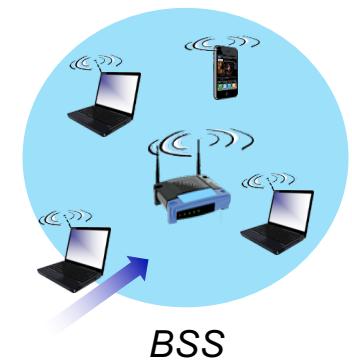
# 802.11 LAN architecture



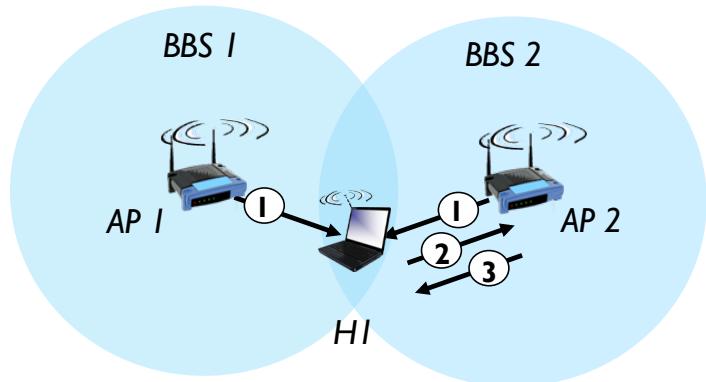
- *wireless host communicates with base station*
  - *base station = access point (AP)*
- ***Basic Service Set (BSS)*** (aka "cell") in infrastructure mode contains:
  - *wireless hosts*
  - *access point (AP): base station*
  - *ad hoc mode: hosts only*

# 802.11: Channels, association

- 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!
- 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 [Security]
  - then typically run DHCP to get IP address in AP's subnet

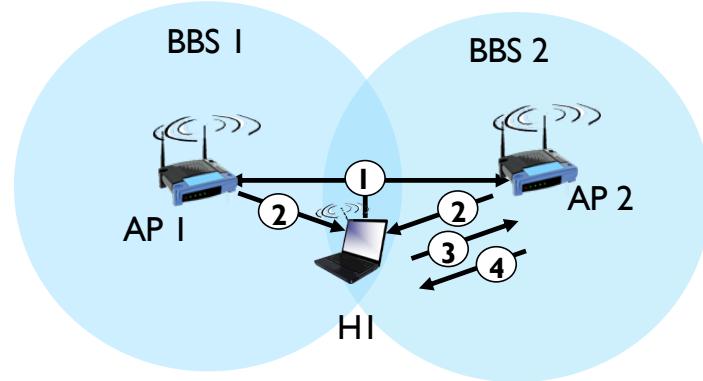


# 802.11: passive/active scanning



## passive scanning:

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

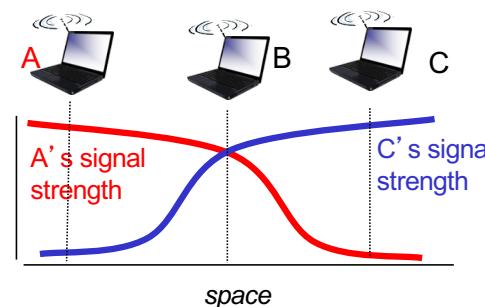
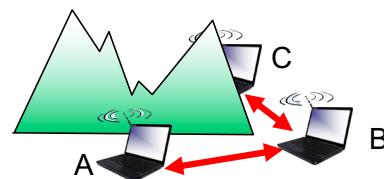


## active scanning:

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

# 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/Collision Avoidance



# Multiple access: Key Points

- ❖ No concept of a global collision
  - Different receivers hear different signals
  - Different senders reach different receivers
- ❖ Collisions are at receiver, not sender
  - Only care if receiver can hear the sender clearly
  - It does not matter if sender can hear someone else
  - As long as that signal does not interfere with receiver
- ❖ Goal of protocol
  - Detect if receiver can hear sender
  - Tell senders who might interfere with receiver to shut up

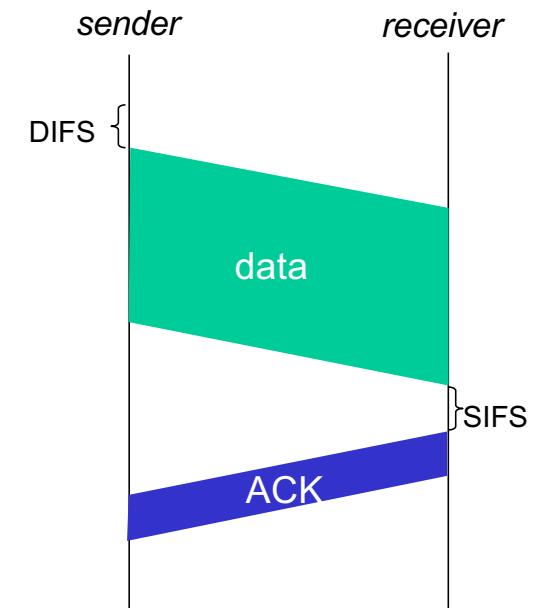
# IEEE 802.11 MAC Protocol: CSMA/CA

## *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  
    transmit when timer expires  
    if no ACK, double random backoff interval, repeat 2

## *802.11 receiver*

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

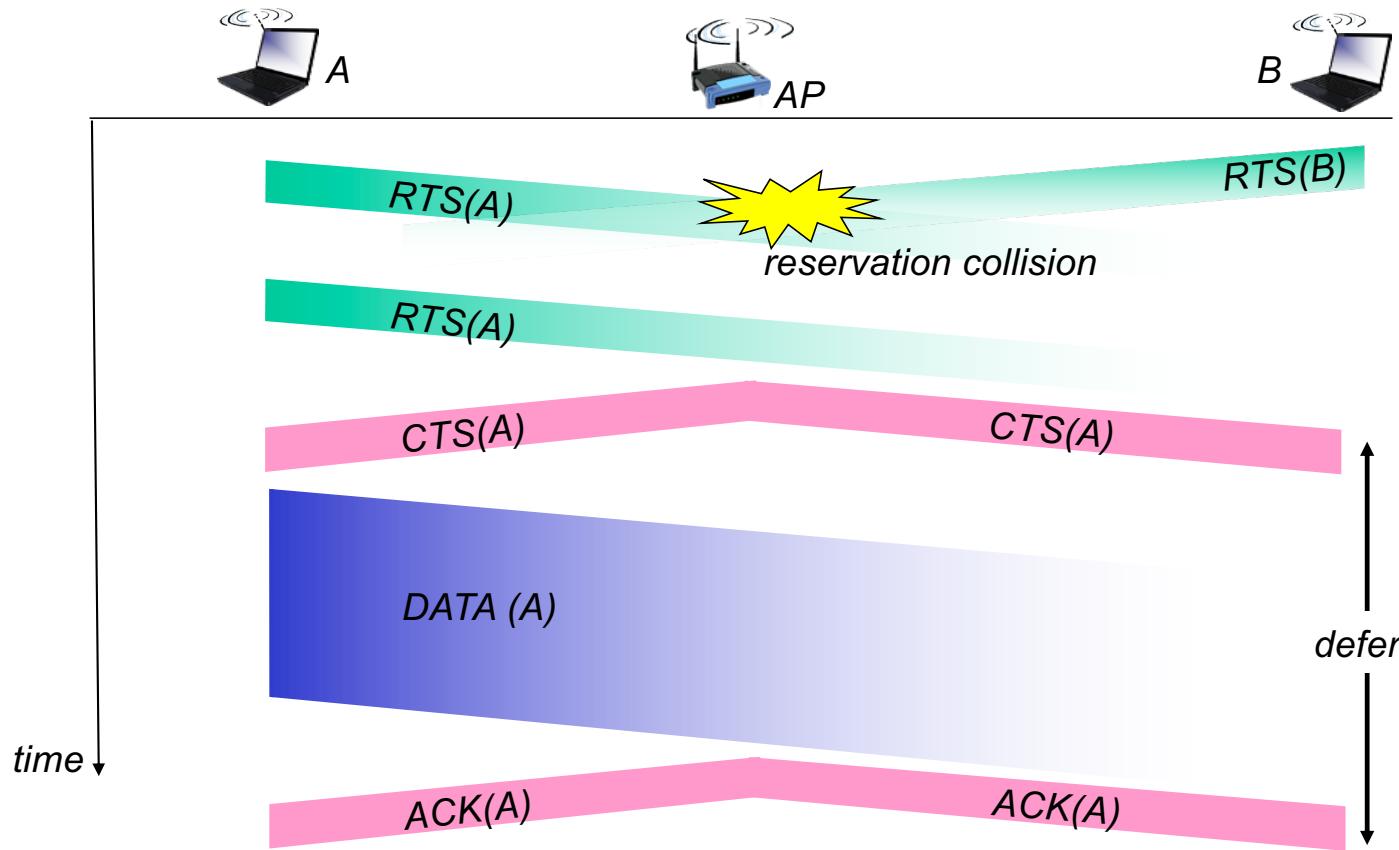


# Avoiding collisions (more)

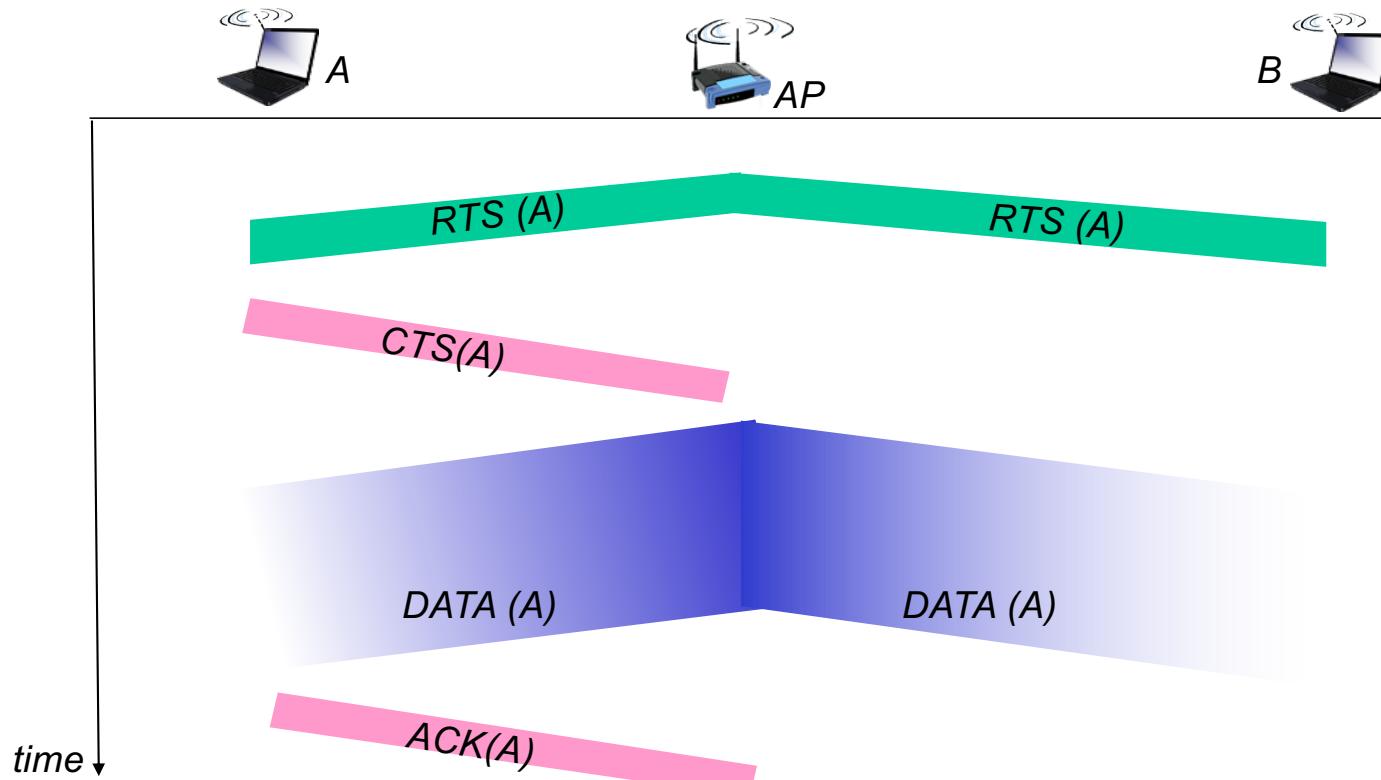
*idea:* sender “reserves” channel use for data frames using small reservation packets

- sender first transmits small request-to-send (RTS) packet to BS using CSMA
  - RTSs may still collide with each other (but they’re short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

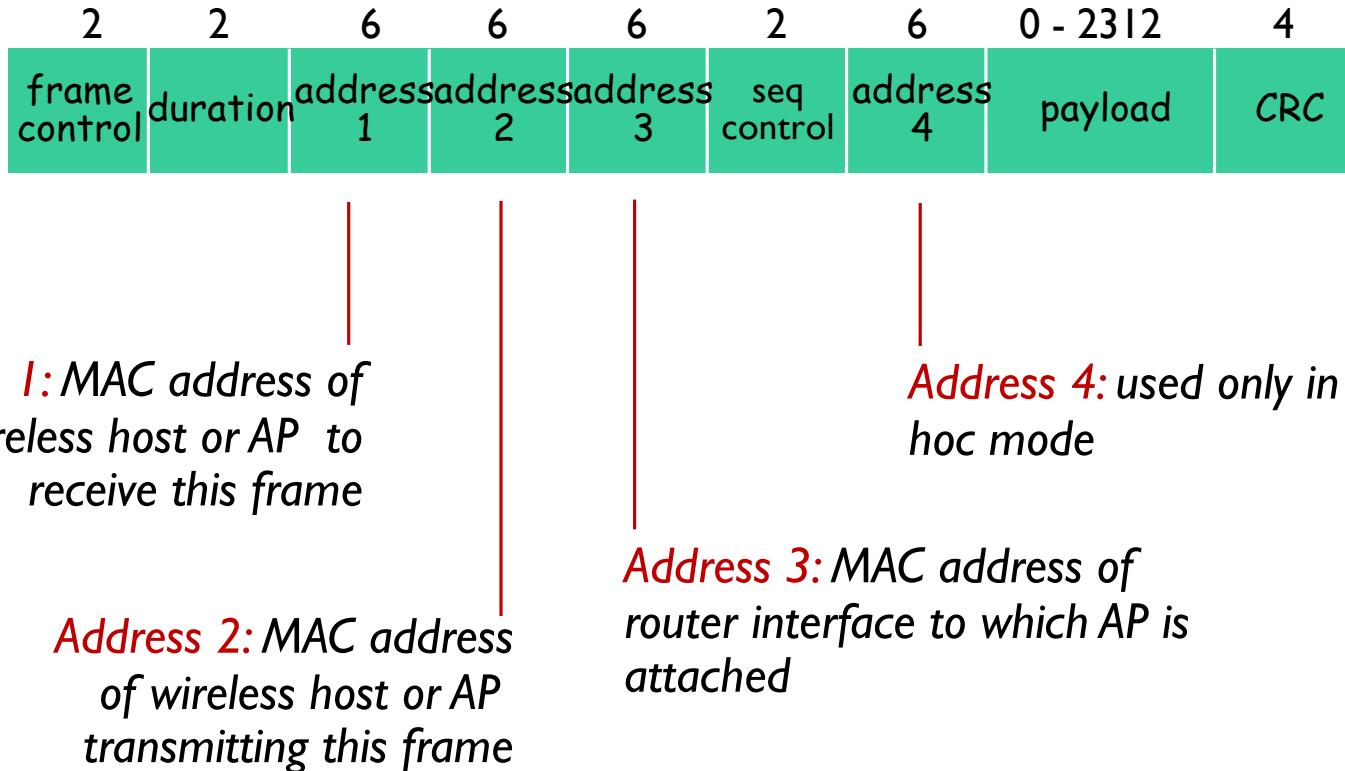
## Collision Avoidance: RTS-CTS exchange



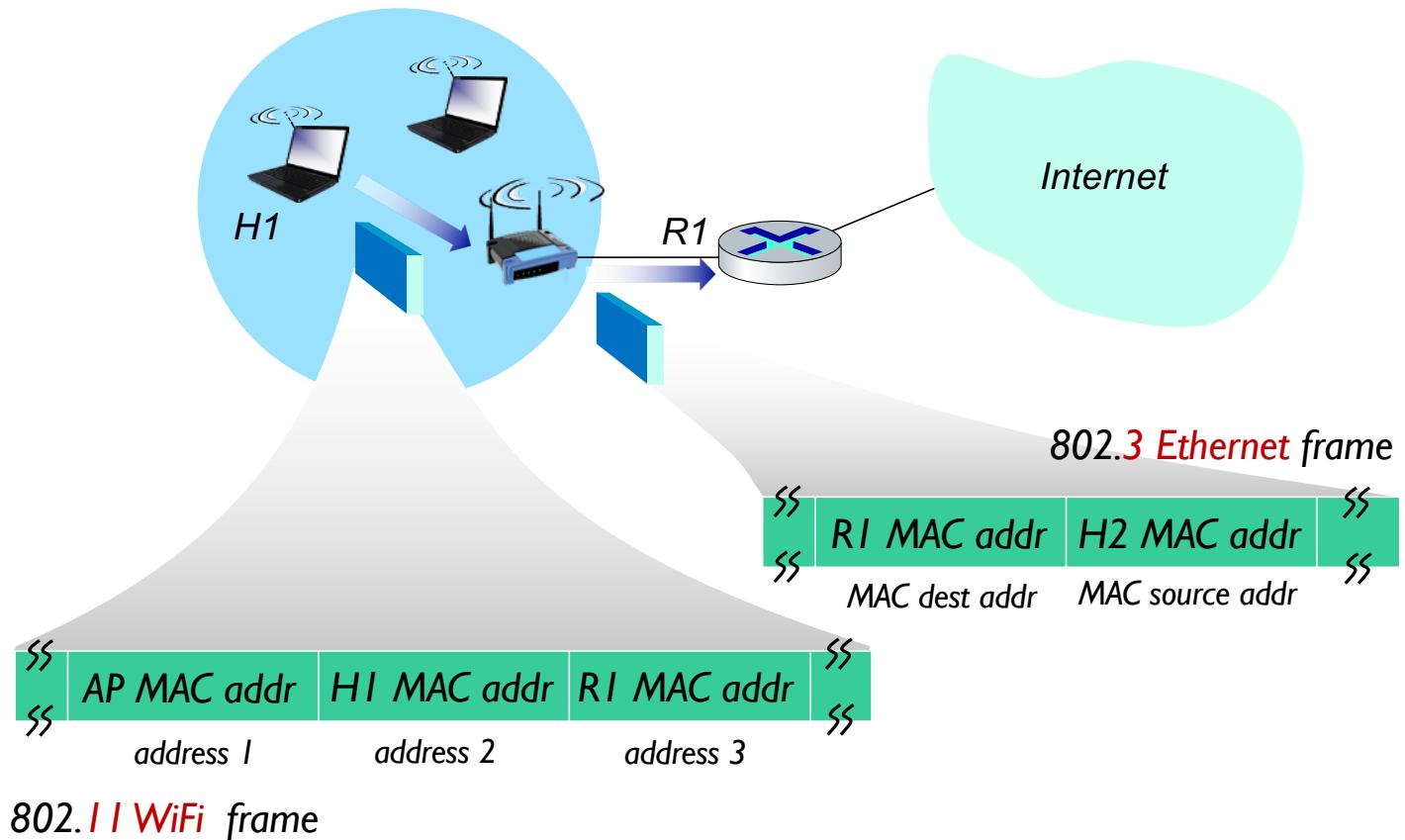
## Collision Avoidance: RTS-CTS exchange



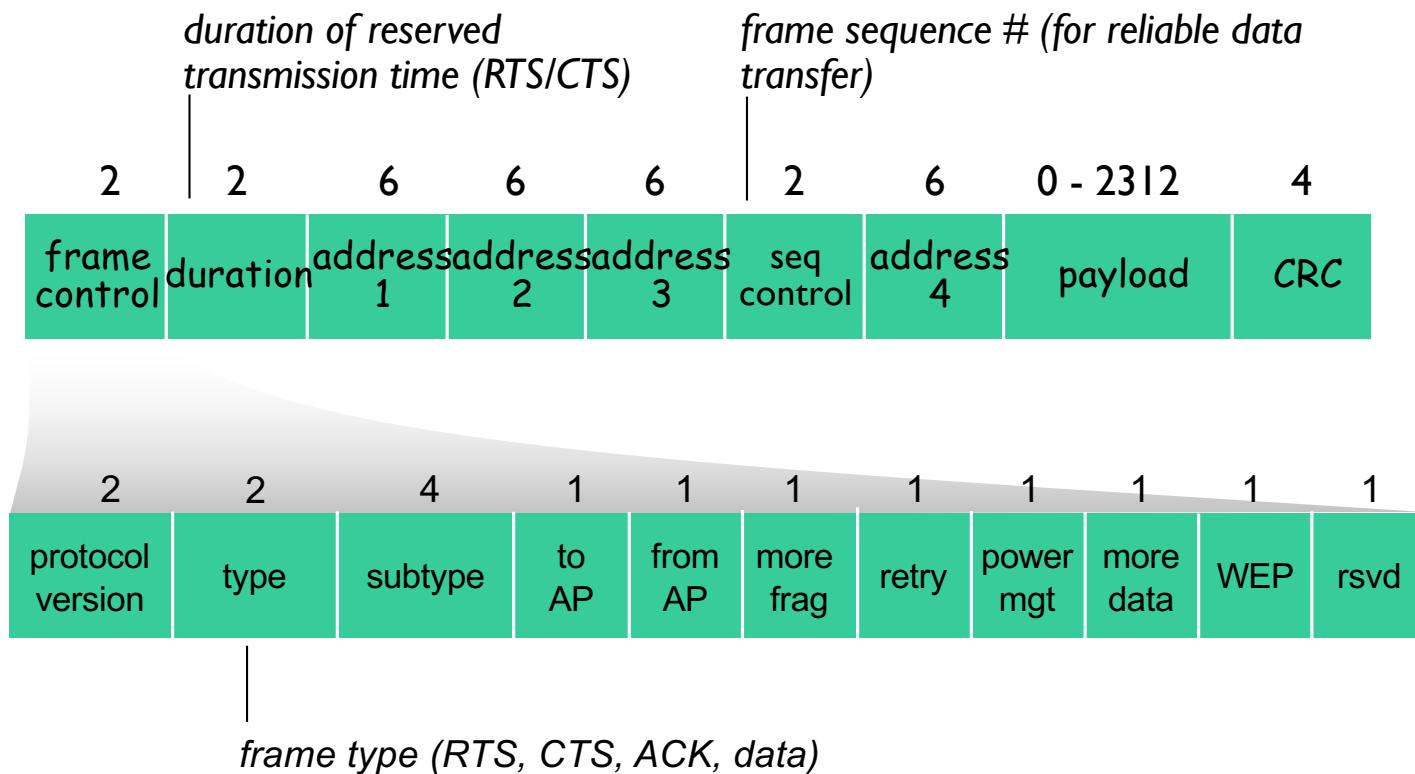
# 802.11 frame: addressing



# 802.11 frame: addressing



# 802.11 frame: addressing



# Quiz

The following is the correct sequence of message exchanges as per the reservation process of 802.11 CSMA/CA

- A. RTS->CTS->DATA->CTS
- B. CTS->RTS->DATA->ACK
- C. RTS->CTS->DATA->ACK
- D. RTS->ACK->DATA->CTS

# Quiz

- ❖ Which multiple access technique is used by IEEE 802.11?
  - A. CSMA/CD
  - B. Slotted ALOHA
  - C. CSMA/CA
  - D. TDMA
  - E. FDMA

# Summary

## Wireless

- ❖ wireless links:
  - capacity, distance
  - channel impairments
- ❖ IEEE 802.11 (“Wi-Fi”)
  - CSMA/CA reflects wireless channel characteristics