

CS2105

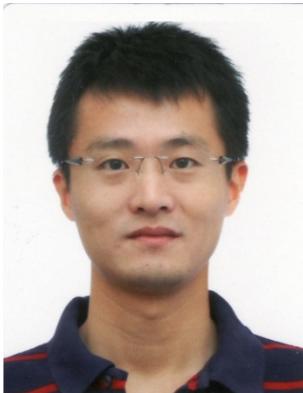
An *Awesome* Introduction to Computer Networks

Lecture 1: Overview



Department of Computer Science
School of Computing

Lecturers



Dr. Zhou Lifeng

Office: COM2 #02-56

Email: zhoulifeng@nus.edu.sg



Prof. Roger Zimmermann

Office: AS6, #05-05

Email: rogerz@comp.nus.edu.sg

What is CS2105 About?

- ❖ Discussion of fundamental **concepts** and **principles** behind computer networking
 - Using the **Internet** as a case study
- ❖ Introduction to networking tools and networked application programming
 - Choice of programming language: **Python, Java, or C**

What you will NOT learn in CS2105

- ❖ How to configure hardware, e.g. router
 - This is covered in [CS3103 Computer Networks Practice](#) - perform hands-on experiments in subnetting, DHCP, DNS, RIP, OSPF, TCP handshaking and congestion mechanism
- ❖ Mobile and wireless networks
 - This is covered in [CS4222 Wireless Networking](#)

Textbook

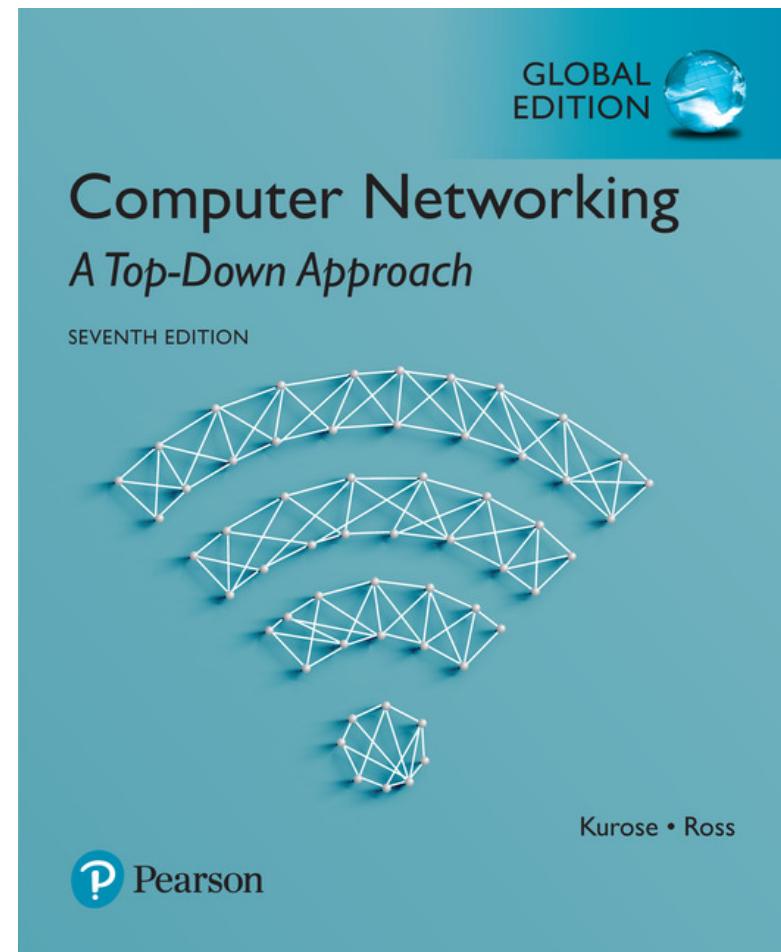
**Computer Networking:
A Top-Down Approach:
Global Edition, 7/E**

Authors : Kurose
Ross

Publisher : Pearson

ISBN : 9781292153599

Acknowledgement:
Most of the lecture slides are
adopted from slides of this textbook.



**Available at NUS
Campus bookstore**

Contact Hours

- ❖ Lectures
 - Video recording will be uploaded every Monday
 - Consultation will be provided online
 - **No lecture in week 8** (reserved for midterm test)
- ❖ Tutorials
 - To be conducted lively online using zoom
 - Start from week 3.
 - 1 hour per session
- ❖ Email me, Roger or your tutor, if you have questions.

Assessments



❖ CA (50%)

- Individual programming assignments - 23%
- Midterm test (week 8 lecture time: **Mon, 4 Oct 2021, 2-4pm**) - 25%
 - E-assessment
- Mock midterm test - 2%
 - Conducted in week 7 tutorial

❖ Final Exam (50%)

- E-assessment
- **Mon, 29 Nov 2021, 9-11am**

Notes and Tips

❖ Why CS2105 can be **easy**

- You use and interact with the Internet constantly
- Many of the concepts are intuitive and based on very practical design considerations
- There are very few equations!

❖ Why CS2105 can be **tough**

- Many concepts are covered
- Programming assignments
 - “Best-effort” technical support from a team of tutors

Lecture 1: Introduction

After this class, you are expected to:

- ❖ understand the basic terms, including host, packet, protocol, throughput, store-and-forward, and autonomous system.
- ❖ know about the **logical** (five protocol layers) and **physical** (a network of ASes) architecture of the Internet.
- ❖ understand the different components of end-to-end delay and their relations to bandwidth, packet size, distance, propagation speed, and queue size.

Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

1.3 Network Core

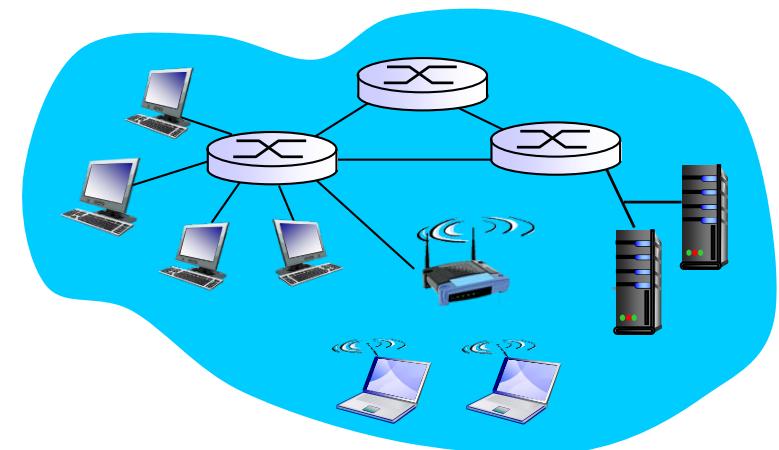
1.4 Delay, Loss and Throughput in Networks

1.5 Protocol Layers and Service Models

Kurose Textbook, Chapter 1
(Some slides are taken from the book)

Internet: “nuts and bolts” View

- ❖ The Internet is a network of connected computing devices (e.g. PC, server, laptop, smartphone)
 - Such devices are known as *hosts* or *end systems*.
 - **Hosts** run network applications (e.g. WhatsApp, browser, Zoom).
 - communicate over links.



Growth of Internet Hosts

number of hosts in Internet

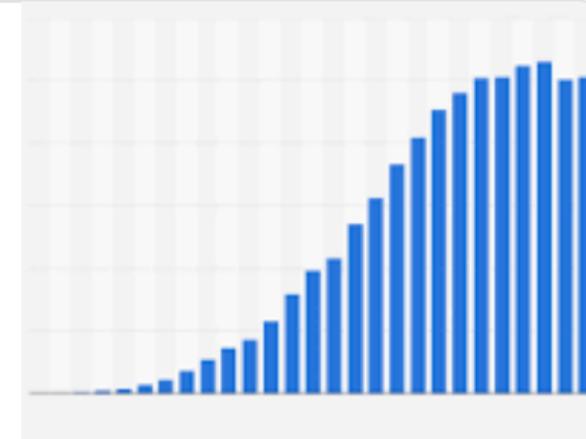
X |  

All Images Videos News Maps More Settings Tools

About 414,000,000 results (0.58 seconds)

1.01 billion

The statistic shows the trend in the global number of internet hosts in the domain name system from 1993 to 2019. In January 2019, approximately **1.01 billion** internet hosts were available on the DNS. May 15, 2020



www.statista.com › Internet › Demographics & Use ▾

- Global internet hosts in the domain name system 2019 ...

“Fun” Internet-connected Devices



IP picture frame
<http://www.ceiva.com/>



Web-enabled toaster +
weather forecaster



Internet
refrigerator



Slingbox: watch,
control cable TV remotely



sensorized,
bed
mattress



Tweet-a-watt:
monitor energy use



Internet phones

Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

- hosts, access networks, links

1.3 Network Core

- packet switching, circuit switching, network structure

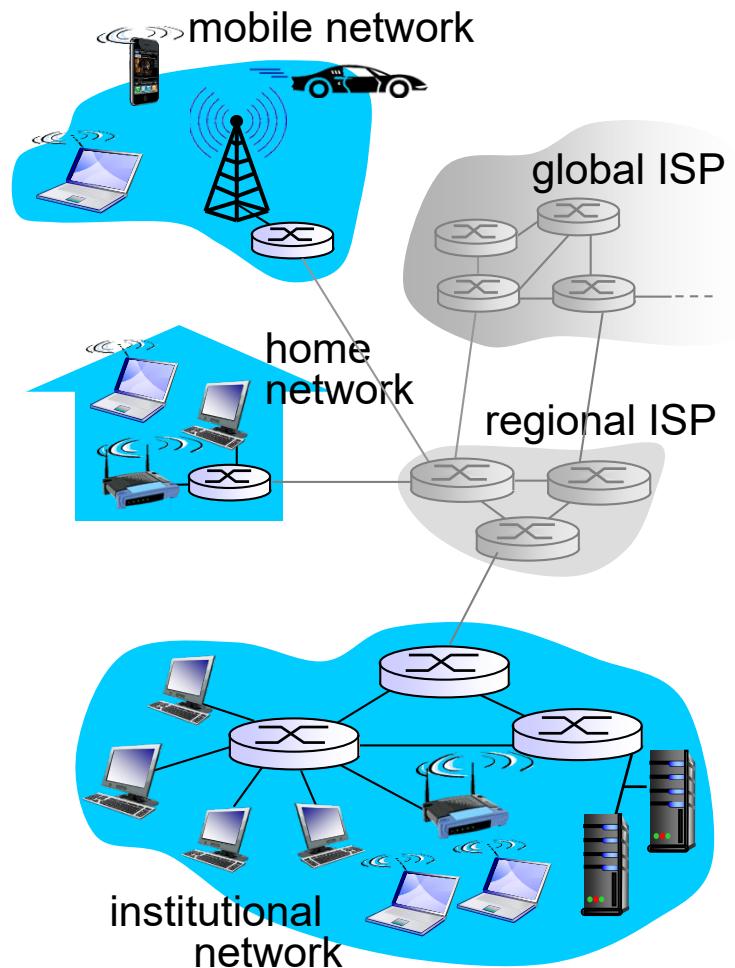
1.4 Delay, Loss and Throughput in Networks

1.5 Protocol Layers and Service Models

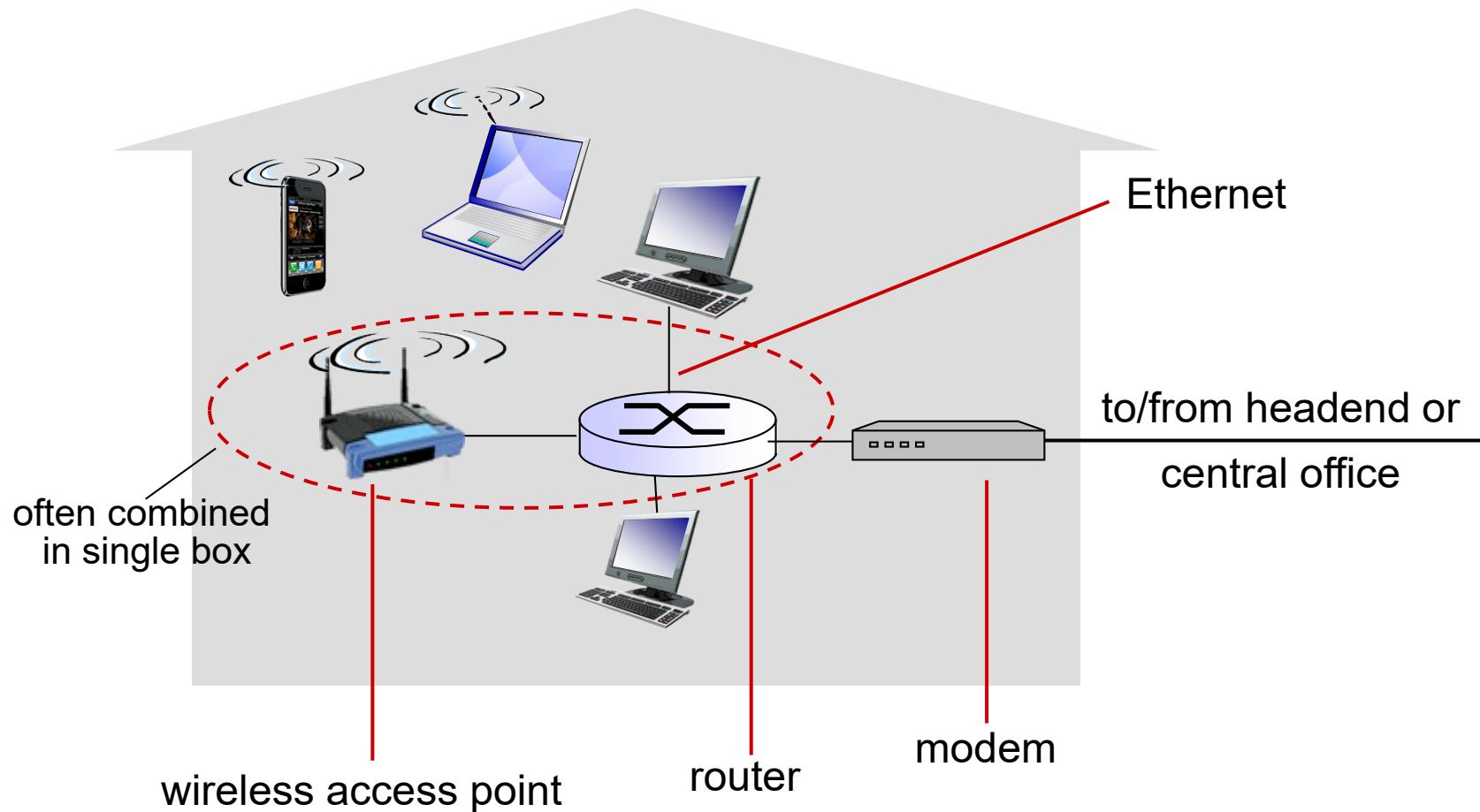
Network Edge (Access Network)

- ❖ Hosts access the Internet through *access network*.
 - Residential access networks
 - Institutional access networks (school, company)
 - Mobile access networks

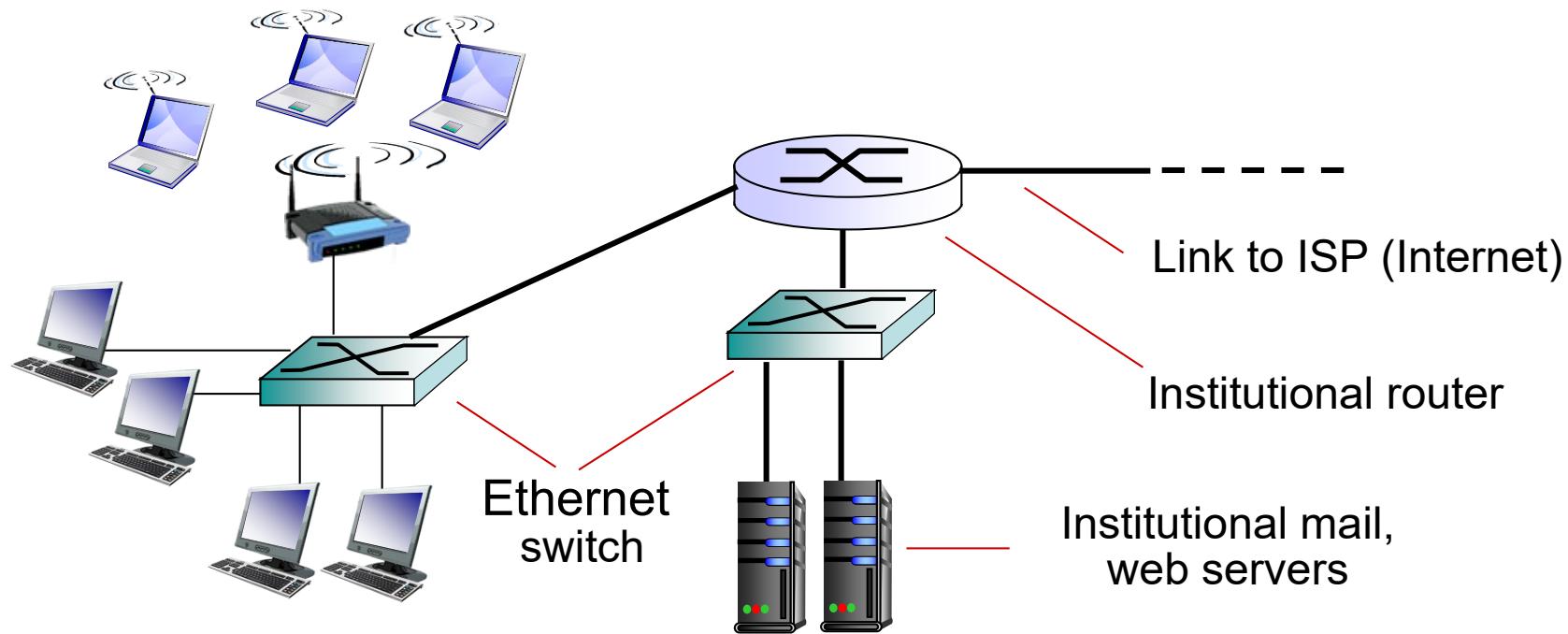
Users/Hosts are here
They access the internet here



Home Networks



Enterprise Access Networks (Ethernet)



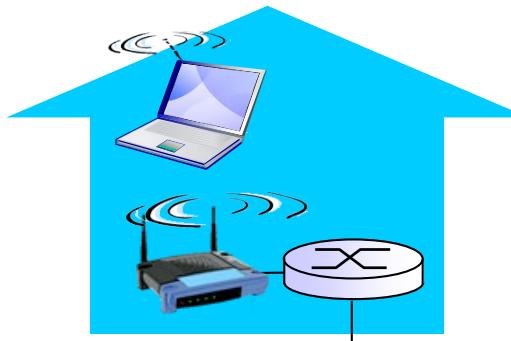
- ❖ Typically used in companies, universities, etc.
- ❖ 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- ❖ Today, hosts typically connect to Ethernet switch

Wireless Access Networks

- ❖ Wireless access network connects hosts to router
 - via base station aka “access point”

Wireless LANs:

- within building (100 ft)
- 802.11b/g/n/ac (Wi-Fi)



to Internet

Wide-area wireless access

- 3G, 4G
- provided by telco (cellular) operator, 10's km



Physical Media

- ❖ Hosts connect to the access network over different physical media.
 - Guided media:
 - signals propagate in solid media



Twisted pair cable



Fiber optic cable

- Unguided media:
 - signals propagate freely, e.g., radio

Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

- hosts, access networks, links

1.3 Network Core

- **packet switching, circuit switching, network structure**

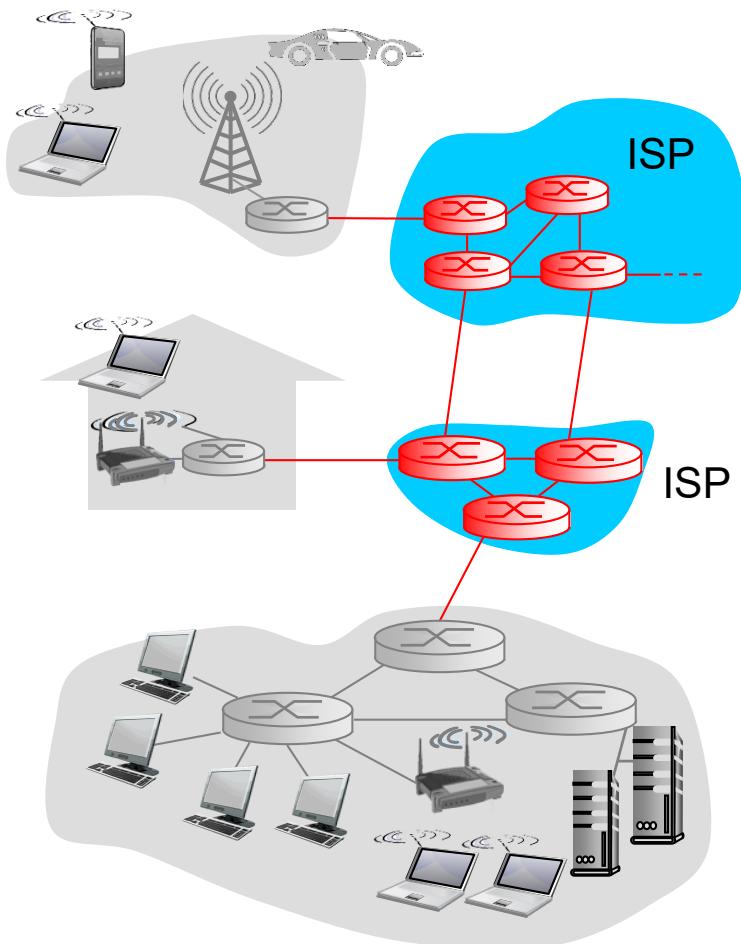
1.4 Delay, Loss and Throughput in Networks

1.5 Protocol Layers and Service Models

The Network Core

- ❖ A mesh of interconnected routers

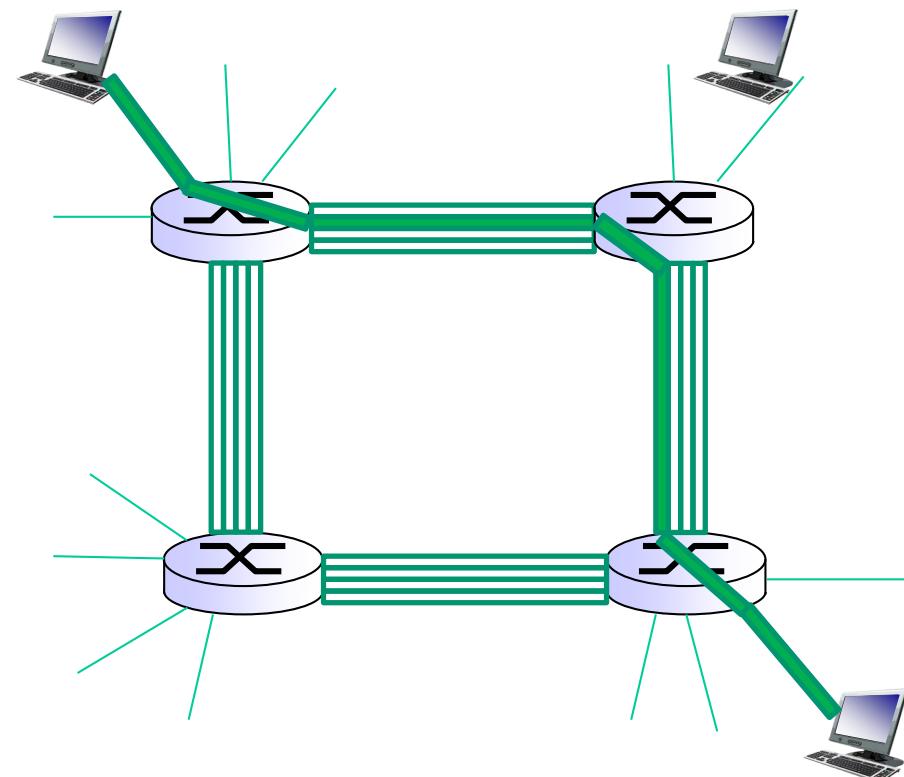
- ❖ How is data transmitted through network?
 - Circuit switching: dedicated circuit per call
 - Packet switching: data sent thru net in discrete “chunks”



Circuit Switching

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

- ❖ call setup required
- ❖ circuit-like (guaranteed) performance
- ❖ circuit segment idle if not used by call (*no sharing*)
- ❖ commonly used in traditional telephone networks

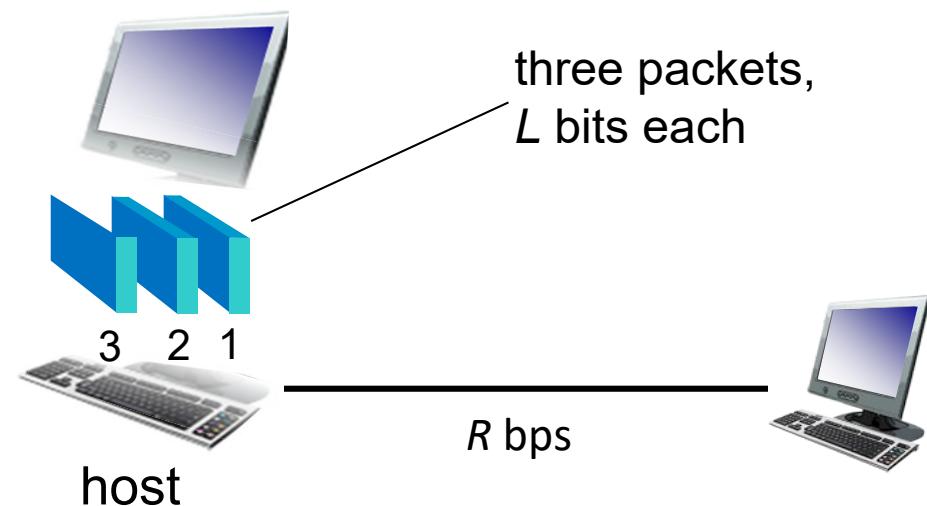


In above diagram, each link has four circuits. A “call” gets 2nd circuit in top link and 1st circuit in right link.

Packet Switching

Host sending function:

- ❖ breaks application message into smaller chunks, known as *packets*, of length L bits
- ❖ transmits packets onto the link at *transmission rate R*
 - link transmission rate is aka *link capacity* or *link bandwidth*

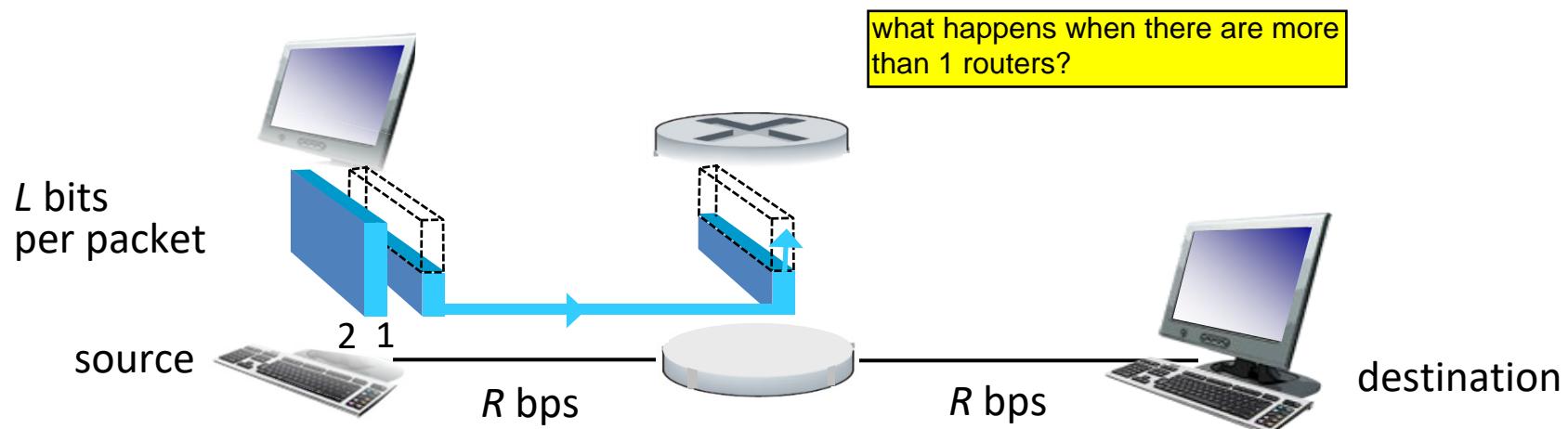


signals are waves, and thus there will be propagation delay

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

Packet-switching: store-and-forward

- ❖ Packets are passed from one **router** to the next, across links on path from source to destination.
- ❖ *Store and forward:* entire packet must arrive at a router before it can be transmitted on the next link.

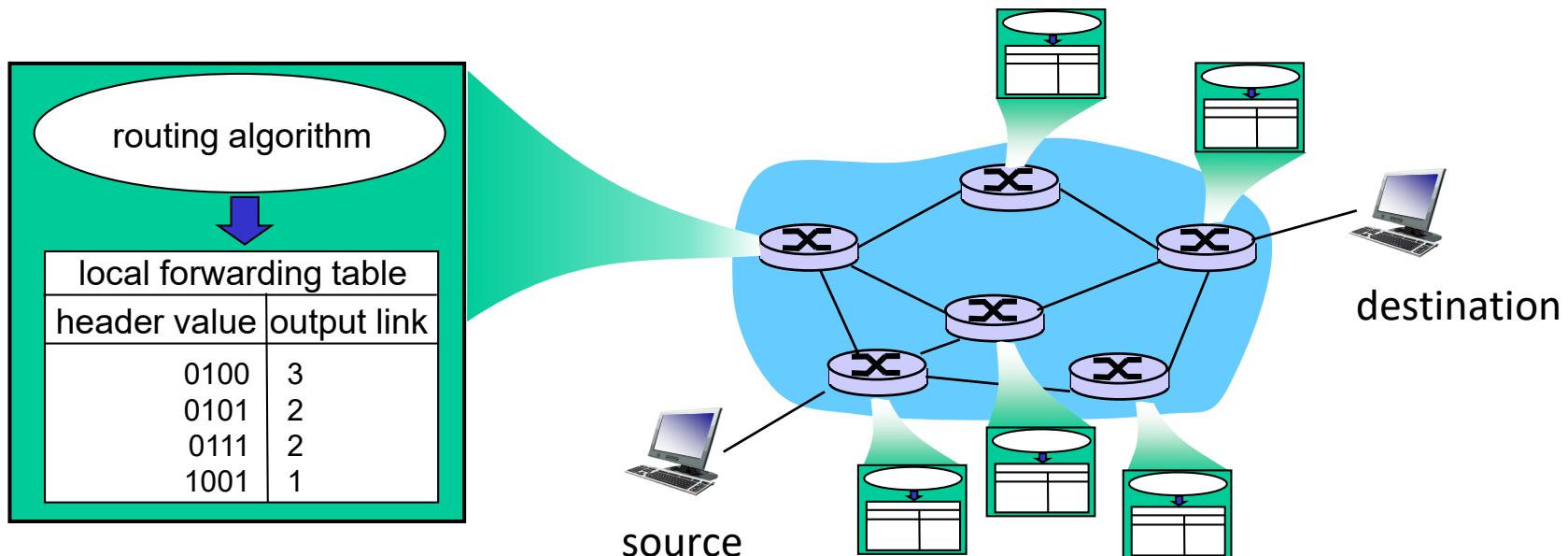


$$\text{End-to-end delay} = 2*L/R \text{ (assuming no other delay)}$$

if there is no router - connected directly - then
delay is only L/R

Routing and Addressing

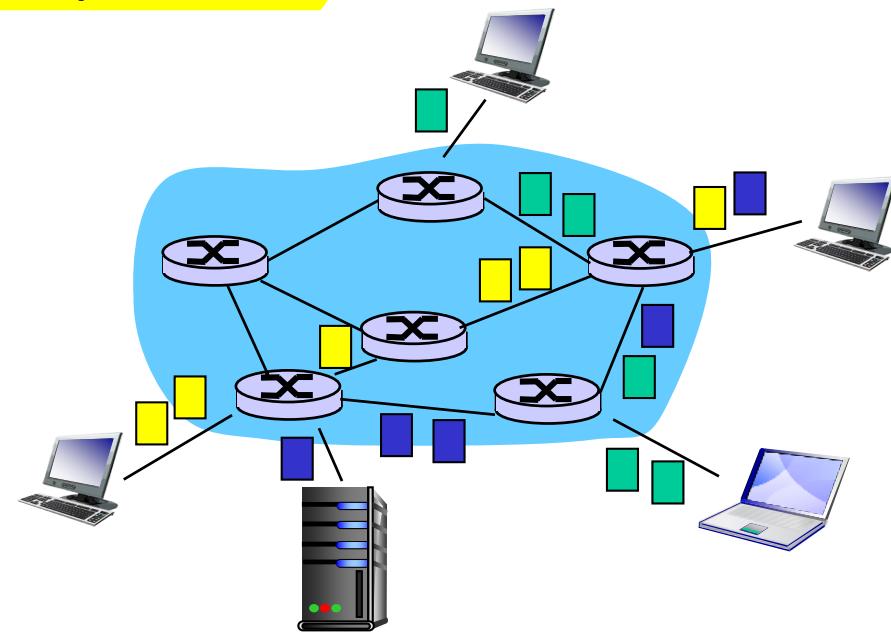
- ❖ Routers determine source-destination route taken by packets.
 - Routing algorithms
- ❖ **Addressing:** each packet needs to carry source and destination information



Summary: Packet Switching

- ❖ The Internet is a packet switching network
- ❖ User A, B ... 's packets *share* network resources
- ❖ Resources are used on demand
- ❖ Excessive congestion is possible

Bandwidth division into
“pieces”
Dedicated allocation
Resource reservation

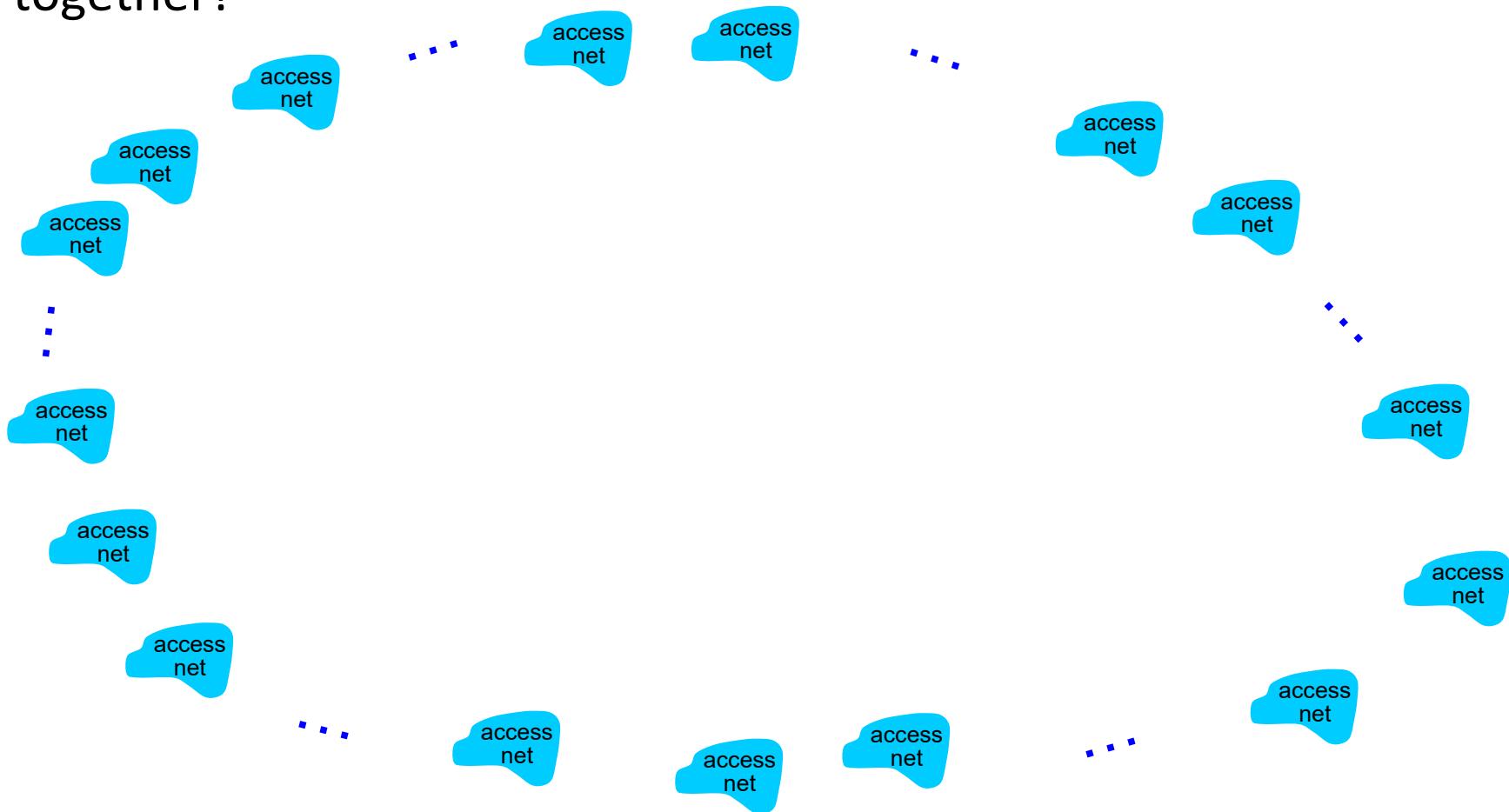


Internet Structure: Network of Networks

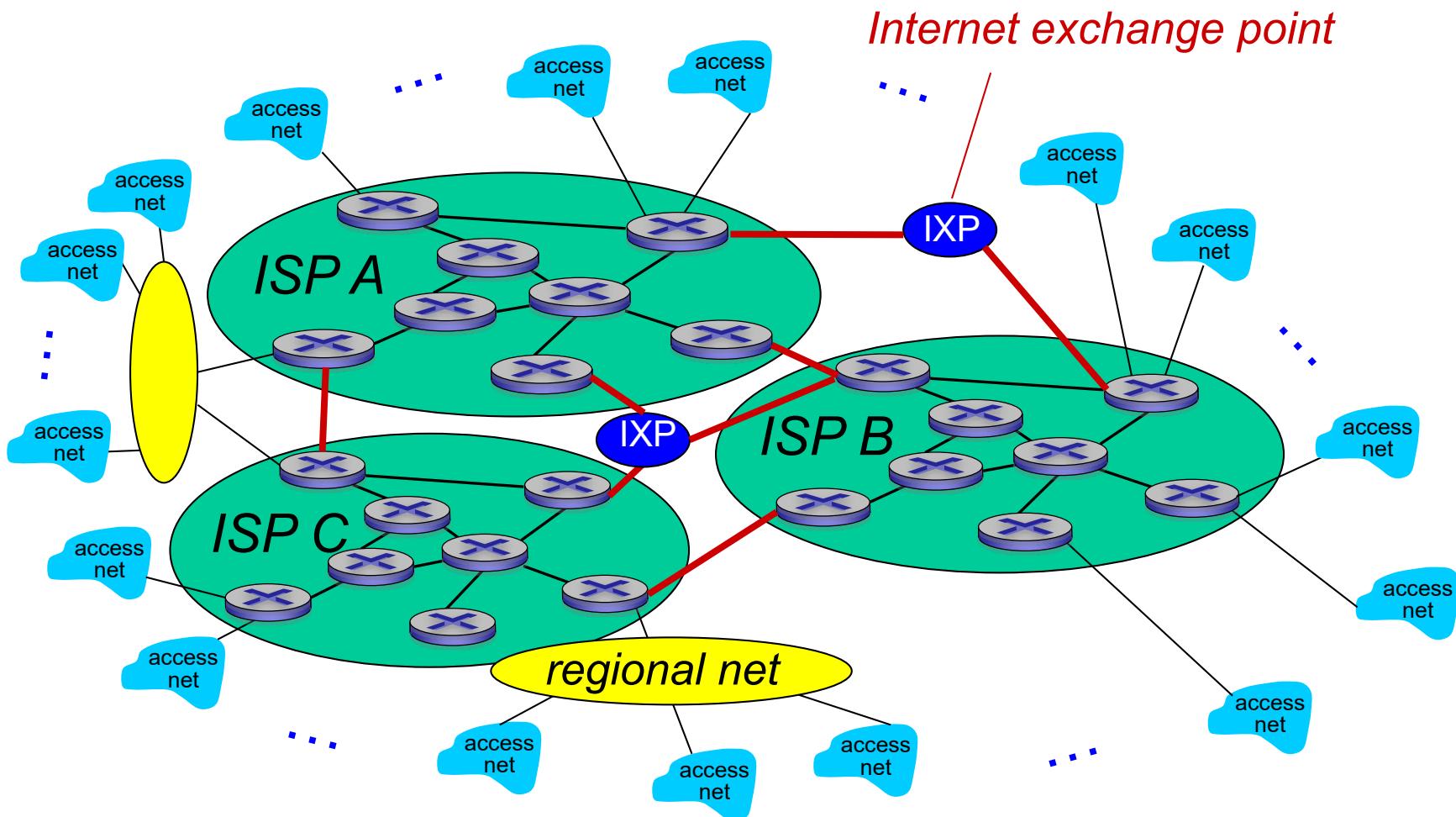
- ❖ Hosts connect to Internet via access **ISPs** (Internet Service Providers)
 - Residential, company and university ISPs
- ❖ Access ISPs in turn must be interconnected.
- ❖ Resulting network of networks is very complex
 - Evolution was driven by **economics** and **national policies**
- ❖ Therefore, the Internet is a “network-of-networks”, organized into autonomous systems (AS), each is owned by an organization.

Internet Structure: Network of Networks

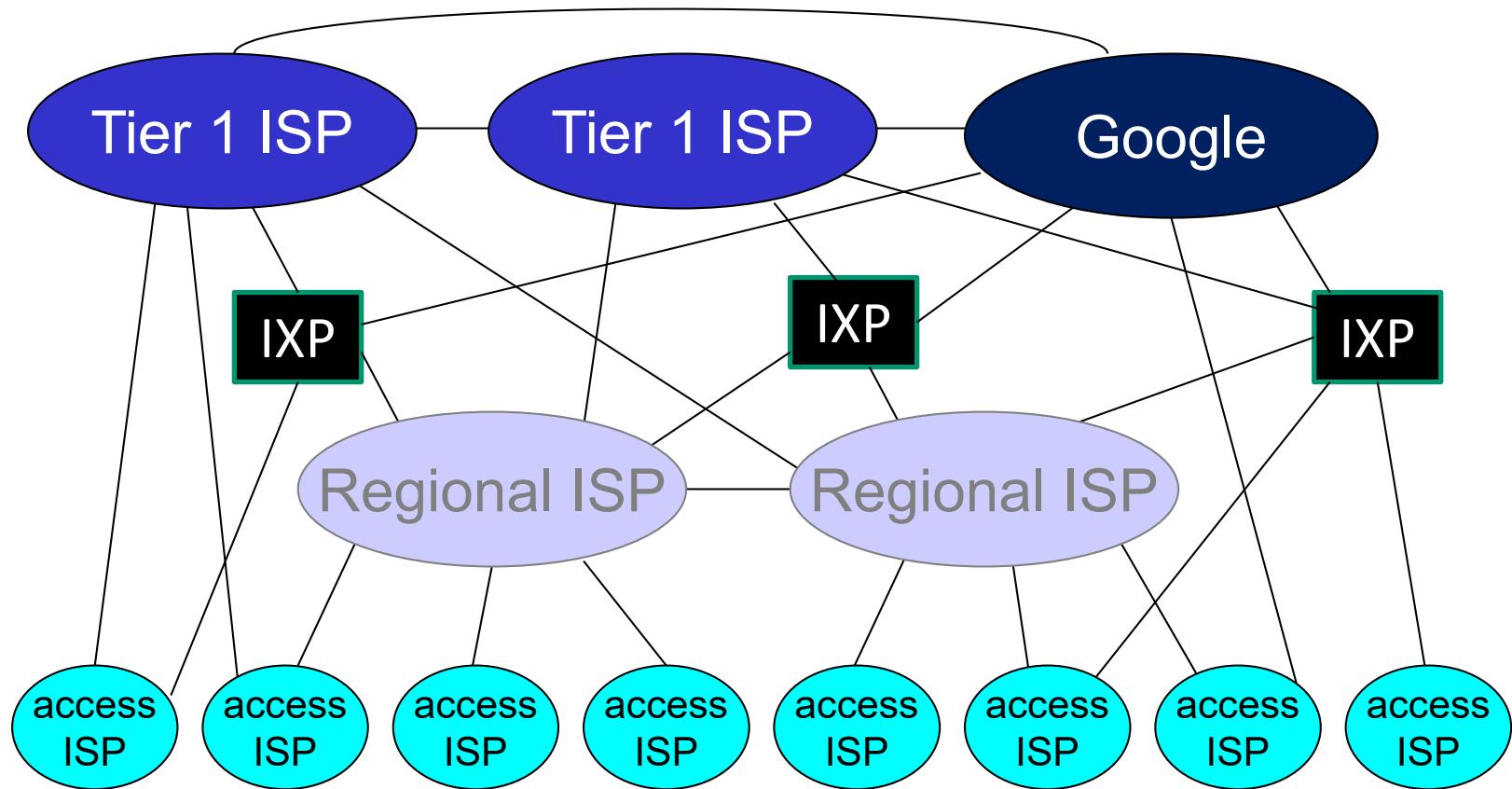
Question: given *millions* of access nets, how to connect them together?



Internet Structure: Network of Networks



Internet Structure: Network of Networks



Who Runs the Internet?

- ❖ IP address & Internet Naming administered by Network Information Centre (NIC)
 - Refer to: www.sgnic.net.sg; www.apnic.org
- ❖ The Internet Society (ISOC) - Provides leadership in Internet related standards, education, and policy around the world.
- ❖ The Internet Architecture Board (IAB) - Authority to issue and update technical standards regarding Internet protocols.
- ❖ Internet Engineering Task Force (IETF) - Protocol engineering, development and standardization arm of the IAB.
 - Internet standards are published as RFCs (Request For Comments)
 - Refer to: www.ietf.org; for RFCs: <http://www.ietf.org/rfc.html>

Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

- hosts, access networks, links

1.3 Network Core

- packet switching, circuit switching, network structure

1.4 Delay, Loss and Throughput in Networks

1.5 Protocol Layers and Service Models

Recall: Packet Switching Network

- ❖ To send a packet in a packet switching network,
 - 1. Sender transmit a packet onto the link as a sequence of bits.
 - 2. Bits are propagated to the next node (e.g. a router) on the link.
 - 3. Router stores, processes and forwards the packet to the next link.
 - 4. Steps 2 & 3 repeat till the packet arrives at the receiver.

signal will propagate over the cable as waves

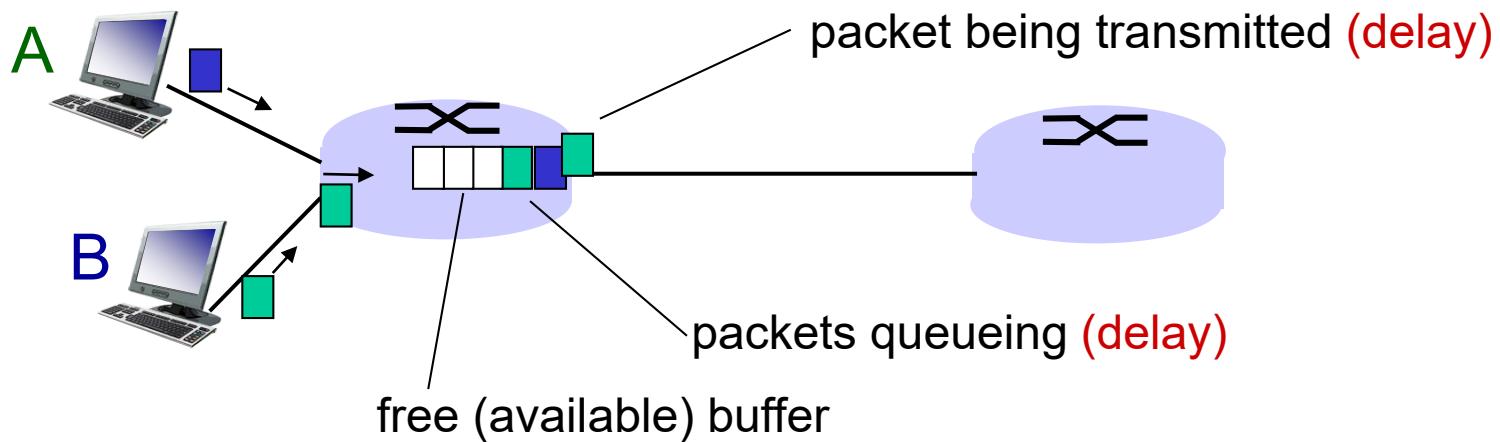
usually a router to check
bits



How do Delay and Loss Occur?

- ❖ Packets *queue* in router buffers
 - wait for turn to be sent out one by one

FIFO

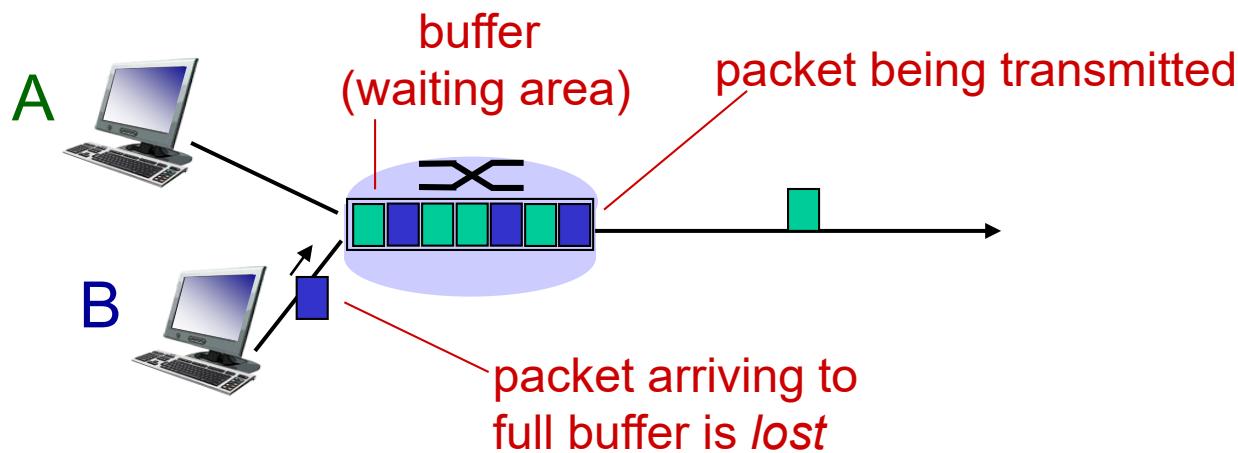


Q: What if packet arrival rate exceeds departure rate?

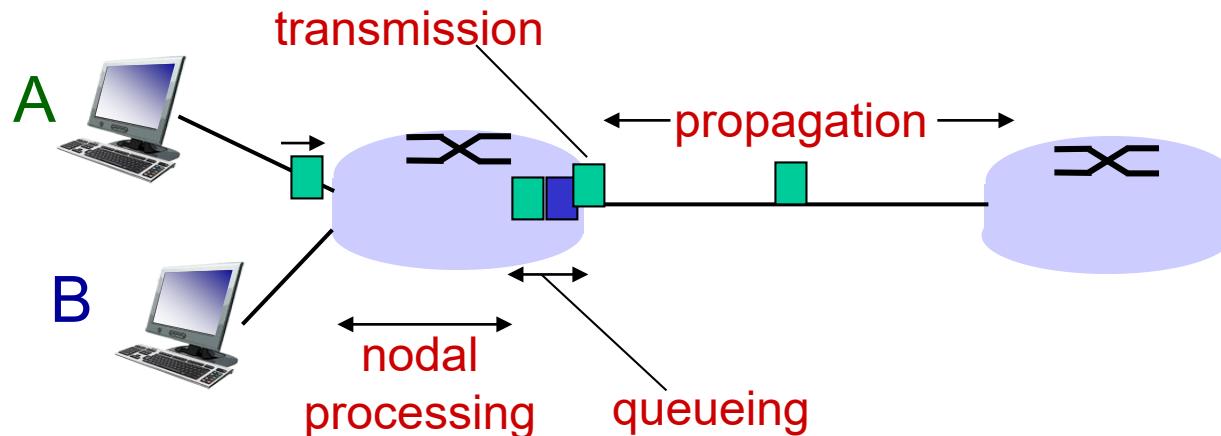
If the queue is full, all incoming packets will be dropped

Packet Loss

- ❖ Queue (aka **buffer**) of a router has finite capacity.
- ❖ Packet arriving to full queue will be dropped (aka lost).



Four Sources of Packet Delay



d_{proc} : nodal processing

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

packet is checked

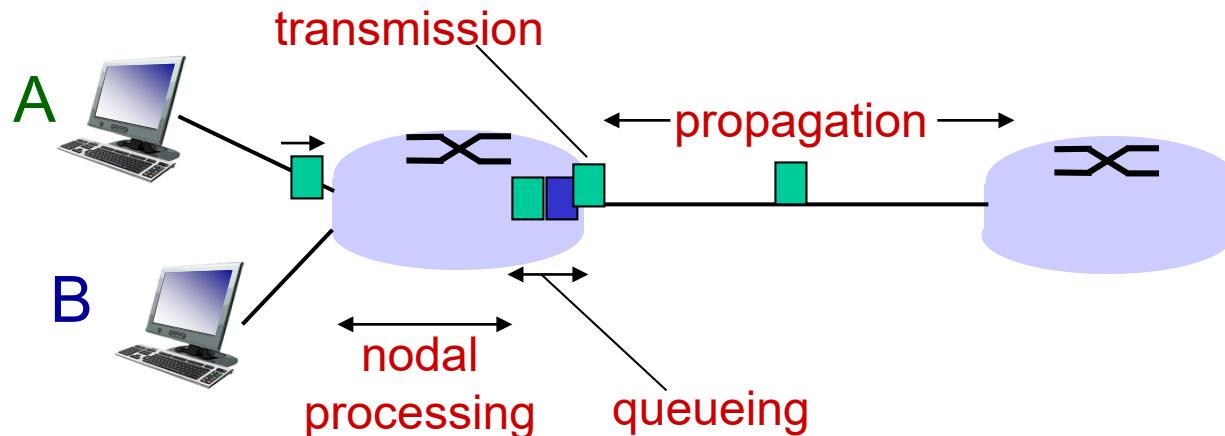
packet is dropped if error

d_{queue} : queuing delay

- time waiting in the queue for transmission
- depends on congestion level of router

if packet is right
sent to queue to wait for
departure

Four Sources of Packet Delay



d_{trans} : transmission delay

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

transmission delay is the L/R as normal

d_{prop} : propagation delay

- d : length of physical link
- s : propagation speed in medium ($\sim 2 \times 10^8$ m/sec)
- $d_{prop} = d/s$
time spent in the cable - moving as wave

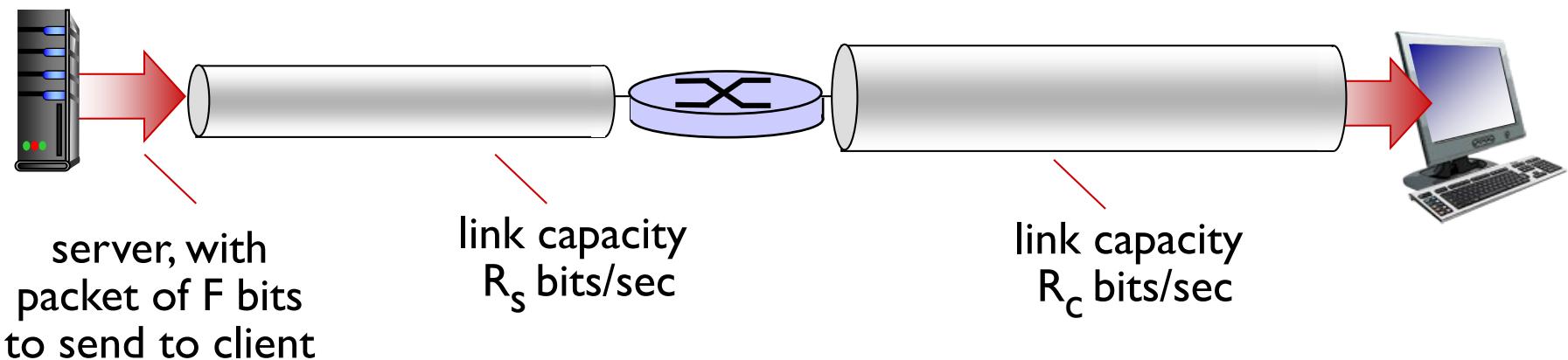
End-to-end Packet Delay

- ❖ End-to-end packet delay is the time taken for a packet to travel from source to destination. It consists of:
 - transmission delay
 - propagation delay
 - processing delay
 - queueing delay

means the start time of 0ms to the end time when the receiver gets all the packets

Throughput

- ❖ Throughput: how many bits can be transmitted per unit time.
 - Throughput is measured for end-to-end communication.
 - Link capacity (bandwidth) is meant for a specific link.



Metric Units

- ❖ 1 byte = 8 bits

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^3	1,000	Kilo
10^{-6}	0.000001	micro	10^6	1,000,000	Mega
10^{-9}	0.000000001	nano	10^9	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

The principal metric prefixes

Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

- hosts, access networks, links

1.3 Network Core

- packet switching, circuit switching, network structure

1.4 Delay, Loss and Throughput in Networks

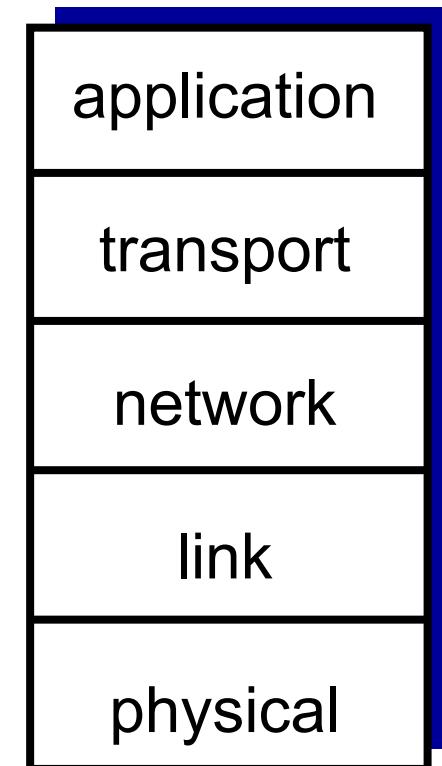
1.5 Protocol Layers and Service Models

Network Protocols

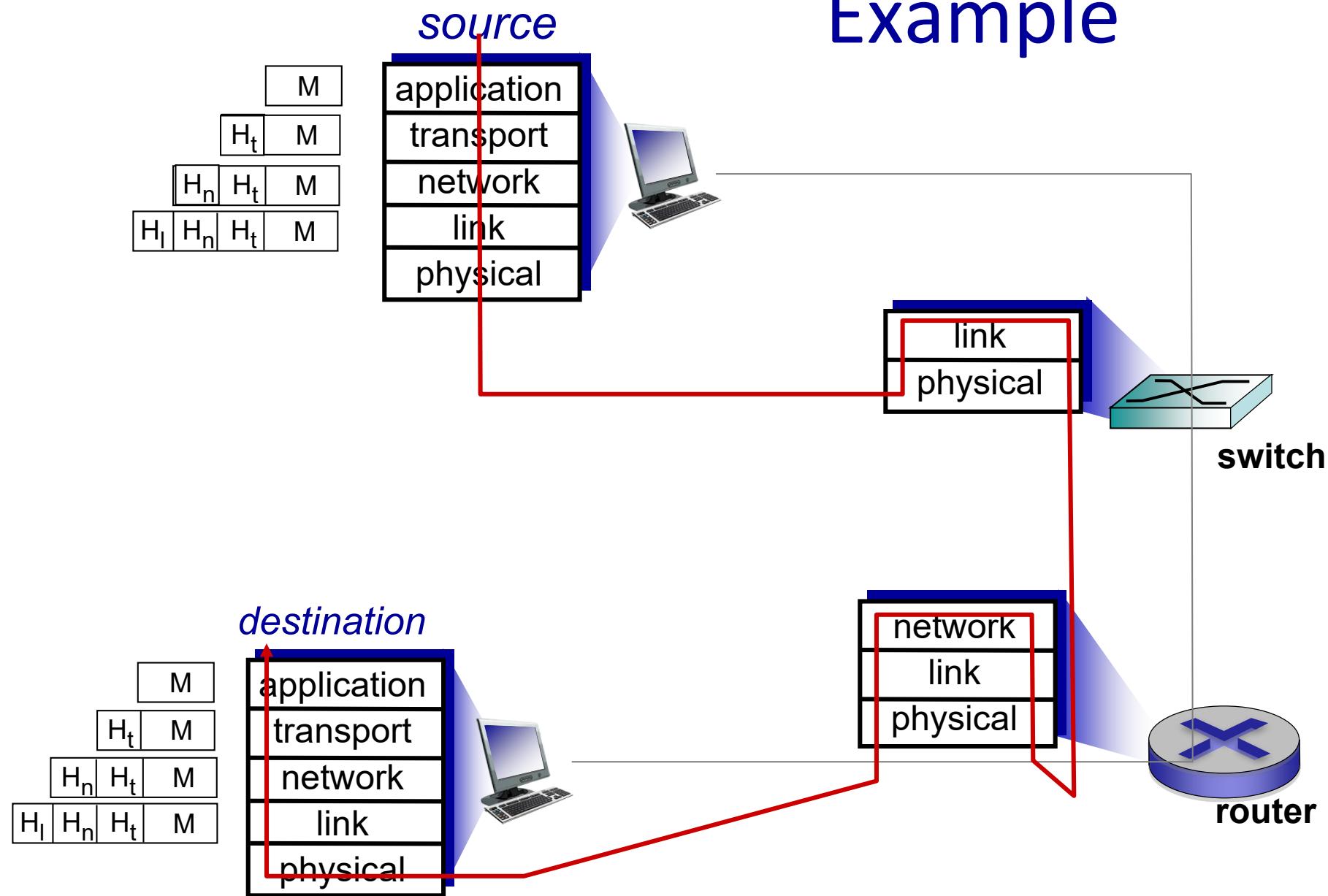
- ❖ The Internet supports various kinds of network applications:
 - Web, VoIP, email, games, e-commerce, social nets, ...
- ❖ Network applications exchange messages and communicate among peers according to **protocols**.
 - A **protocol** defines **format** and **order** of messages exchanged and the **actions** taken after messages are sent or received.

Internet Protocol Stack

- ❖ Protocols in the Internet are logically organized into 5 “layers” according to their purposes.
 - *application*: supporting network applications
 - FTP, SMTP, HTTP
 - *transport*: process-to-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”

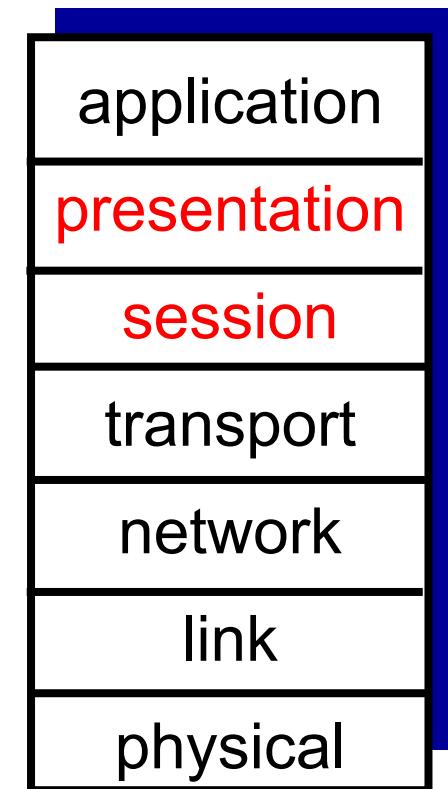


Example



ISO/OSI reference model (FYI)

- ❖ Theoretical model – not in use
- ❖ Two additional layers not present in Internet Protocol Stack
 - *presentation*: allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
 - *session*: synchronization, checkpointing, recovery of data exchange



Lecture 1: Summary

covered a “ton” of material!

- ❖ Internet overview
- ❖ Network edge, core, access network
 - packet-switching versus circuit-switching
 - Internet structure
- ❖ Performance: loss, delay, throughput
- ❖ What’s a protocol?
- ❖ Layering, service models

you now have:

- ❖ Context, overview, “feel” of networking
- ❖ More depth, detail *to follow!*

CS2105

An *Awesome* Introduction to Computer Networks

Lectures 2&3: The Application Layer



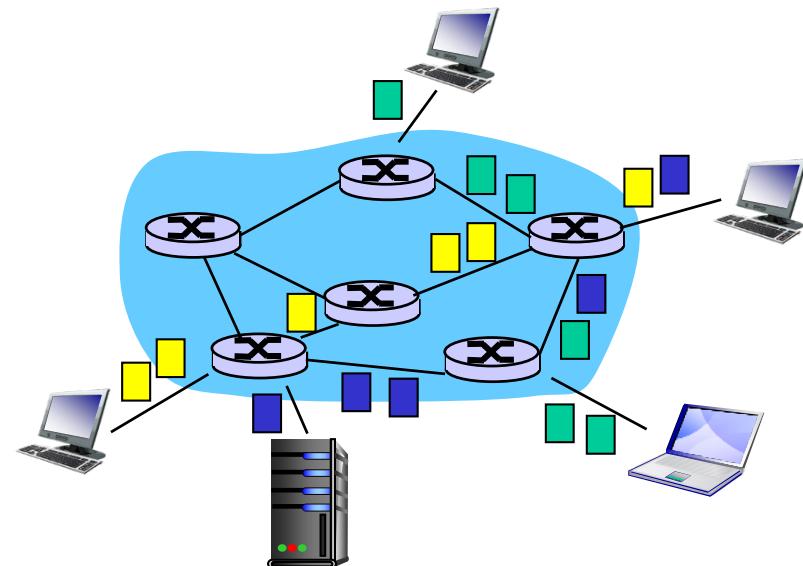
Department of Computer Science
School of Computing

Packet Switching

- ❖ The Internet is a **packet switching** network
 - Hosts share and contend network resources.
 - **Application message** is broken into a bunch of packets and sent onto the link one by one.
 - A router **stores and forwards** packets.
 - Receiver assembles all the packets to restore the **application message**.

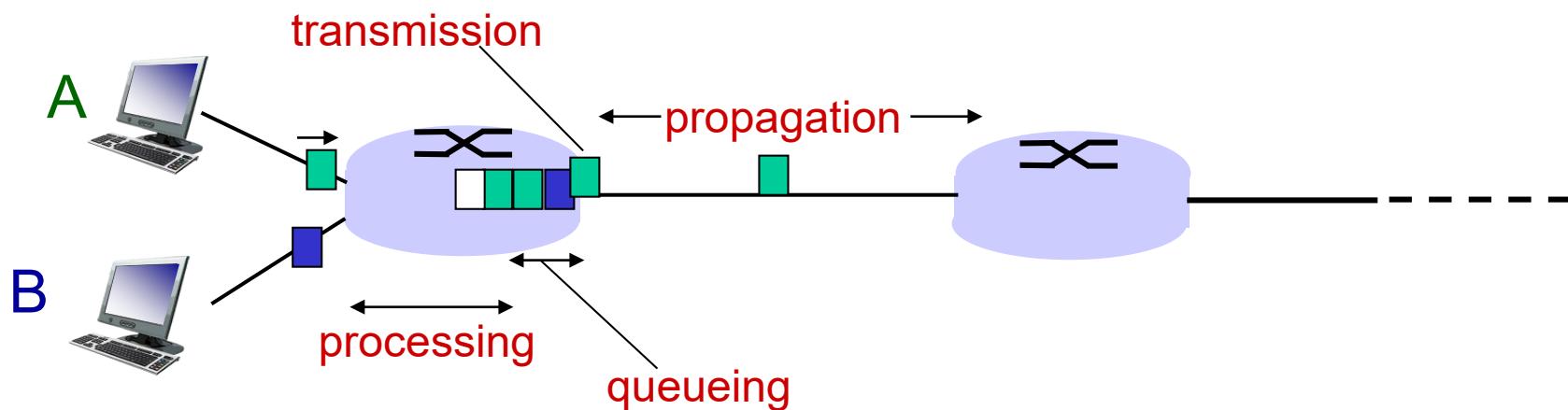
Anyone can use the network at any time

Bandwidth division into “pieces”
Dedicated allocation
Resource reservation



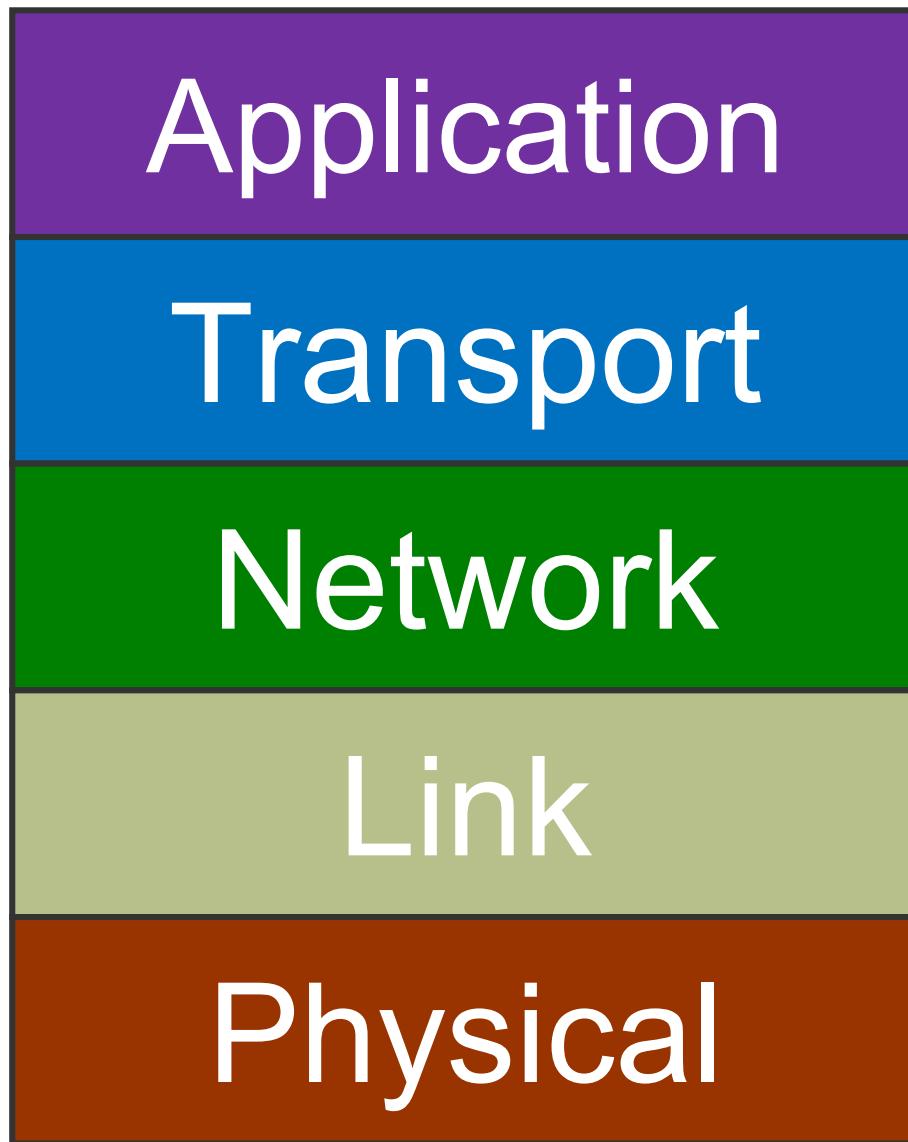
Packet Delay

- ❖ **End-to-end delay** is the time taken for a packet to travel from source to destination. It consists of:
 - processing delay
 - queueing delay
 - transmission delay
 - propagation delay



Network Protocols

- ❖ Networks are complex. There are many issues to consider, to support different applications running on large number of hosts through different access technology and physical media.
- ❖ **Protocols** regulate communication activities in a network.
 - Define the *format* and *order* of messages exchanged between hosts for a specific purpose.



Upper layer protocols will use the service provided by lower layers

Lectures 2&3: Application Layer

After this class, you are expected to:

- ❖ understand the basic HTTP interactions between the client and the server, the concepts of persistent and non-persistent connections.
- ❖ understand the services provided by DNS and how a DNS query is resolved.
- ❖ understand the concept of socket.
- ❖ be able to write simple client/server programs through socket programming.

Lectures 2&3: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.4 DNS

2.7 Socket programming



To discuss
next week

Kurose Textbook, Chapter 2
(Some slides are taken from the book)

Evolution of Network Applications

- ❖ Early days of Internet
 - Remote access (e.g. telnet, now ssh)
- ❖ 70s – 80s
 - Email, FTP
- ❖ 90s
 - Web
- ❖ 2000s – now
 - P2P file sharing
 - Online games
 - Instant Messaging, Skype
 - YouTube, Facebook

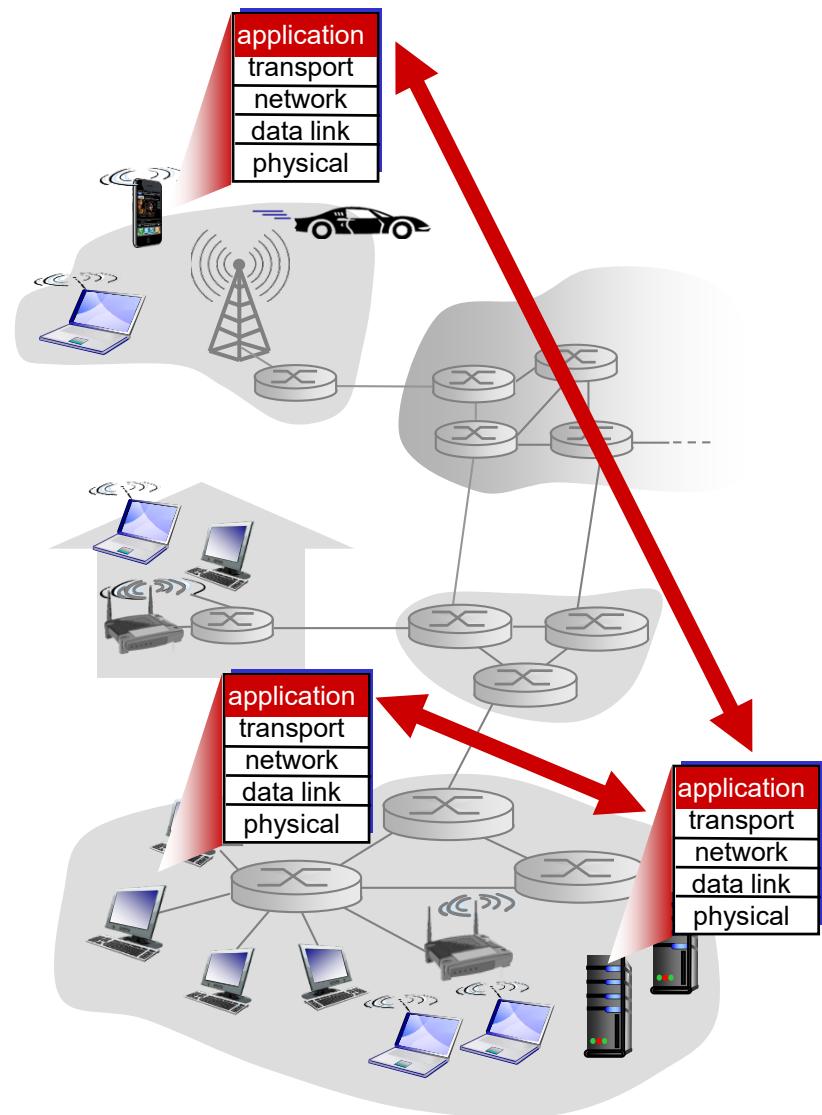
Creating Network Applications

write programs that:

- ❖ run on (different) *hosts*
- ❖ communicate over network
- ❖ e.g., web server software ↔ browser software

classic structure of network applications:

- ❖ Client-server
- ❖ Peer-to-peer (P2P)



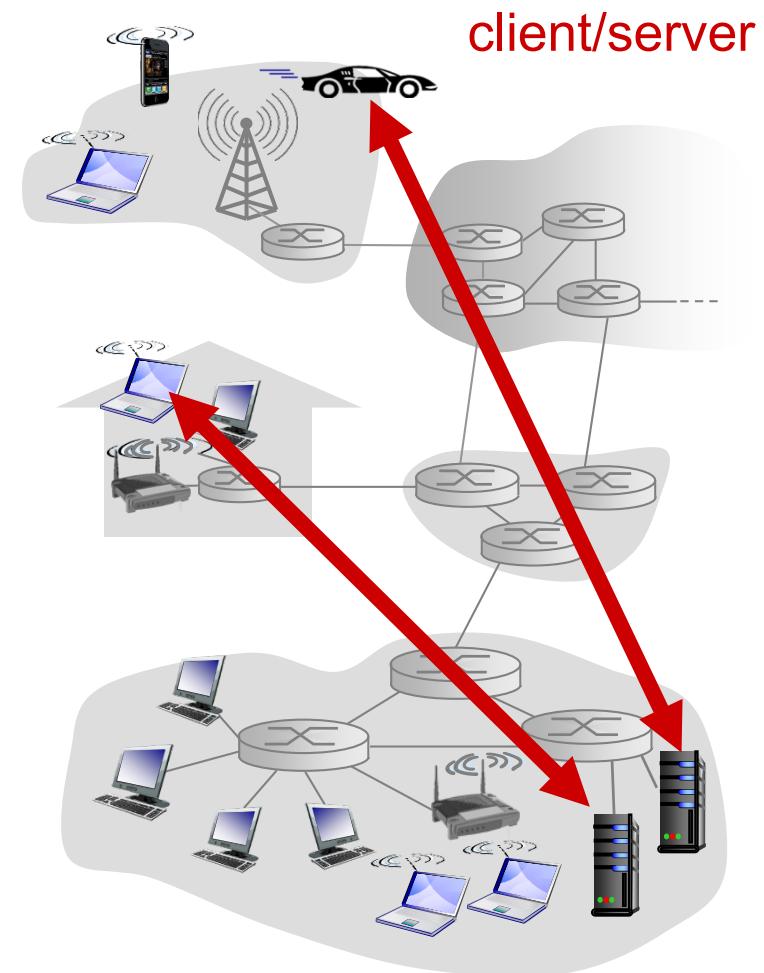
Client-Server Architecture

Server:

- ❖ Waits for incoming requests
- ❖ Provides requested service to client
- ❖ data centers for scaling

Client:

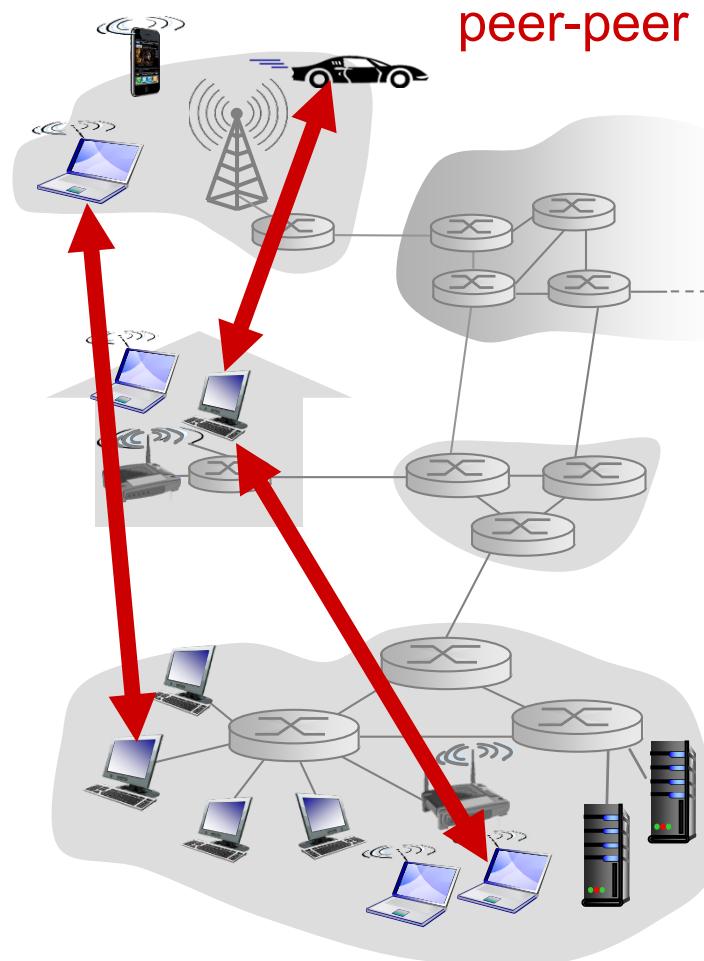
- ❖ Initiates contact with server (“speaks first”)
- ❖ Typically requests service from server
- ❖ For Web, client is usually implemented in browser



P2P Architecture

download and upload simultaneously

- ❖ 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



What transport service does an app need?

retransmission is not useful since the old/corrupted data might not be useful anymore

Data integrity

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

Timing

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

Throughput

- ❖ some apps (e.g., multimedia) require minimum amount of bandwidth to be “effective”
- ❖ other apps (e.g., file transfer) make use of whatever throughput available

Security

- ❖ encryption, data integrity, authentication ...

Requirements of Example Apps

Application	Data loss	Throughput	Time-sensitive
File transfer	No loss	Elastic	No
Electronic mail	No loss	Elastic	No
Web documents	No loss	Elastic	No
Real-time audio/video	Loss-tolerant	Audio: 5kbps-1Mbps Video:10kbps-5Mbps	Yes: 100s of msec
Stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps – 10 kbps	Yes: 100s of msec
Text messaging	No loss	Elastic	Yes and no

App-layer Protocols Define...

- ❖ 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 applications send & respond to messages
-
- ❖ open protocols:
 - defined in RFCs
 - allows for interoperability
 - e.g., HTTP, SMTP
 - ❖ proprietary protocols:
 - e.g., Skype

Transport Layer Protocols: TCP/UDP

- ❖ App-layer protocols ride on transport layer protocols:

TCP service:

- ❖ *reliable data transfer*
- ❖ *flow control*: sender won't overwhelm receiver
- ❖ *congestion control*: throttle sender when network is overloaded
- ❖ *does not provide*: timing, minimum throughput guarantee, security

UDP service:

- ❖ *unreliable data transfer*
- ❖ *no flow control*
- ❖ *no congestion control*
- ❖ *does not provide*: timing, throughput guarantee or security

cannot guarantee any performance/throughput because the packet switching network does not allow for any reservation of the resource - inherent problem of internet architecture

Example App/Transport Protocols

Application	Application Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube)	Typically TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	TCP or UDP

Lectures 2&3: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.4 DNS

2.7 Socket programming

The Web: Some Jargon

- ❖ A Web page typically consists of:
 - *base HTML file*, and
 - *several referenced objects*.
- ❖ An object can be HTML file, JPEG image, Java applet, audio file, ...
- ❖ Each object is addressable by a *URL*, e.g.,

`www.comp.nus.edu.sg/~cs2105/img/doge.jpg`

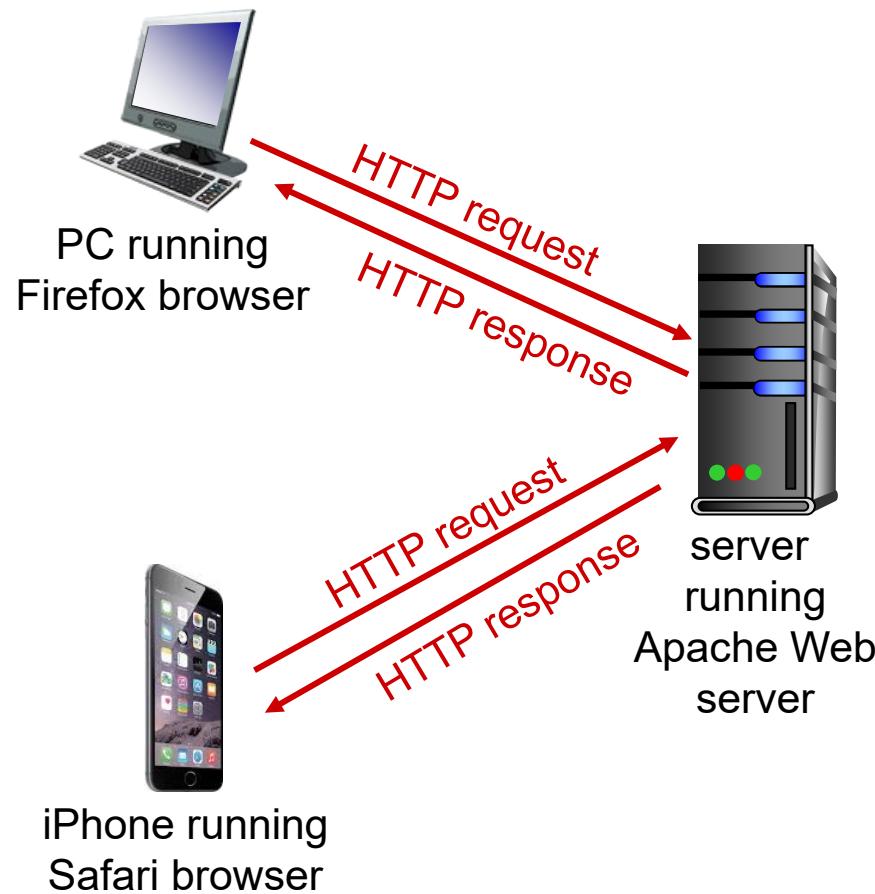
host name

path name

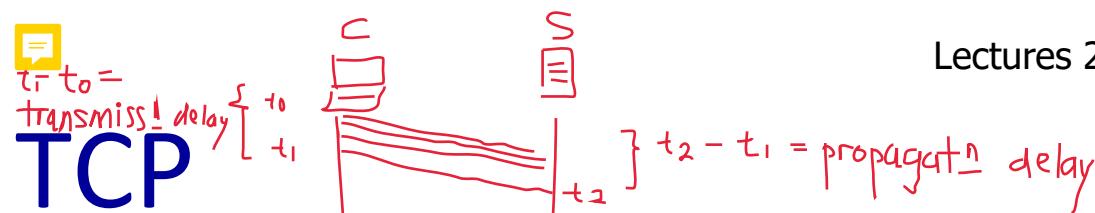
HTTP Overview

HTTP: Hypertext transfer protocol

- ❖ Web's application layer protocol
- ❖ Client/server model
 - *client*: usually is browser that requests, receives and displays Web objects
 - *server*: Web server sends objects in response to requests
- ❖ http 1.0: RFC 1945
- ❖ http 1.1: RFC 2616



HTTP Over TCP



HTTP uses TCP as transport service

- ❖ Client initiates TCP connection to server.
- ❖ Server accepts TCP connection request from client.
- ❖ HTTP messages are exchanged between browser (HTTP client) and Web server (HTTP server) over TCP connection.
- ❖ TCP connection closed.

workflow

Two Versions of HTTP

Non-persistent HTTP

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

Persistent HTTP

- ❖ multiple objects can be sent over single TCP connection between client, server

Non-persistent HTTP Example

suppose user enters URL:

`www.comp.nus.edu.sg/~cs2105/demo.html`

(contains text,
reference to a
jpeg image)

1a. HTTP client initiates TCP connection to HTTP server at `www.comp.nus.edu.sg` on port 80

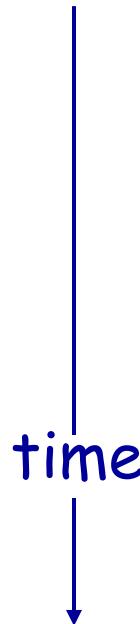
1b. HTTP server at host `www.comp.nus.edu.sg` is waiting for TCP connection at port 80. It "accepts" connection and reply client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `~cs2105/demo.html`

3. HTTP server receives request message, forms *response message* containing requested object, and sends message to the client

time ↓

Non-persistent HTTP Example



4. HTTP server closes TCP connection.
5. HTTP client receives response message containing html file, displays html. Parsing html file, client notices the referenced jpeg object
6. Steps 1-5 repeated for the jpeg object

- ❖ This is an example of *non-persistent connection* (http 1.0).
 - One object per connection
- ❖ HTTP 1.1 allows *persistent connection* (to discuss).
 - Multiple objects per connection

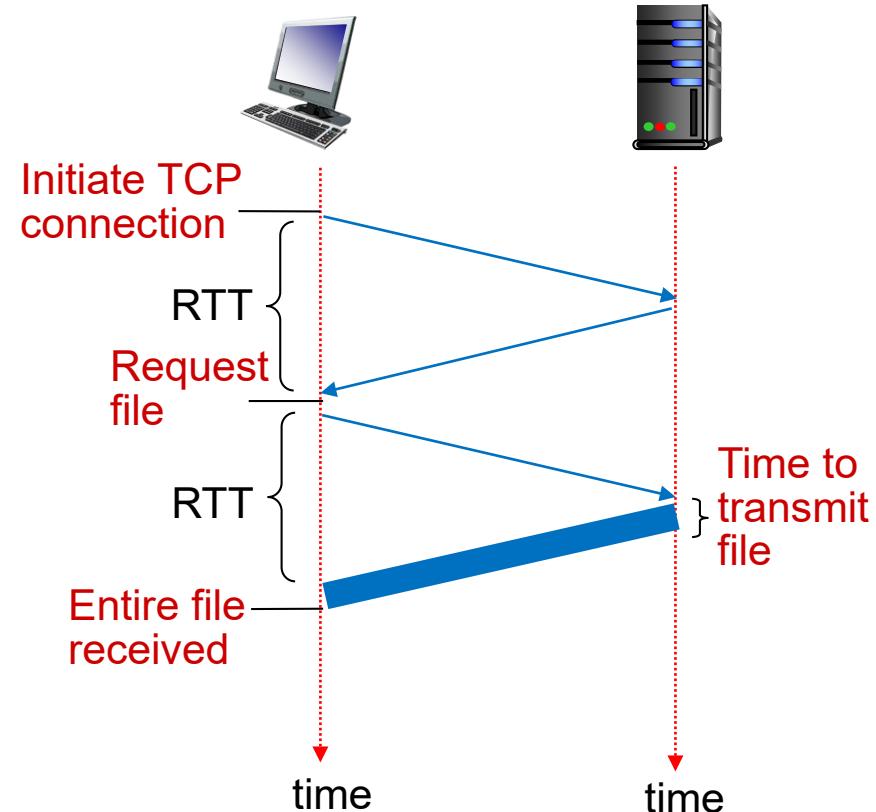
Non-persistent HTTP: Response Time

 RTT: time for a packet to travel from client to server and go back

HTTP response time:

- ❖ one RTT to establish TCP connection
- ❖ one RTT for HTTP request and the first few bytes of HTTP response to return
- ❖ file transmission time
- ❖ non-persistent HTTP response time =

$$2 * \text{RTT} + \text{file transmission time}$$

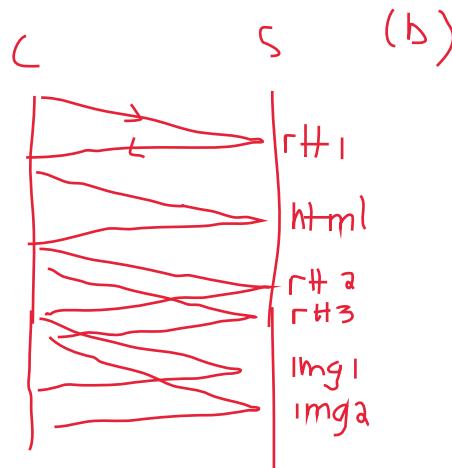


Persistent HTTP

non-persistent HTTP

issues:

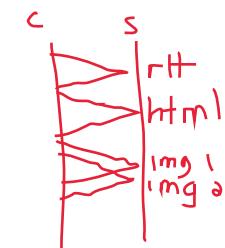
- ❖ requires 2 RTTs per object
- ❖ OS overhead for *each* TCP connection
- ❖ browsers often open parallel TCP connections to fetch referenced objects



persistent HTTP:

- ❖ server leaves connection open after sending response
- ❖ subsequent HTTP messages between same client/server sent over the same TCP connection
- ❖ moreover, client may send requests as soon as it encounters a referenced object (**persistent with pipelining**)

- as little as one RTT for all the referenced objects



Example HTTP Request Message

- ❖ Two types of HTTP messages: *request, response*
- ❖ HTTP request message:

request line
(GET method)

header
lines

GET /index.html HTTP/1.1\r\n

Host: www.example.org\r\n

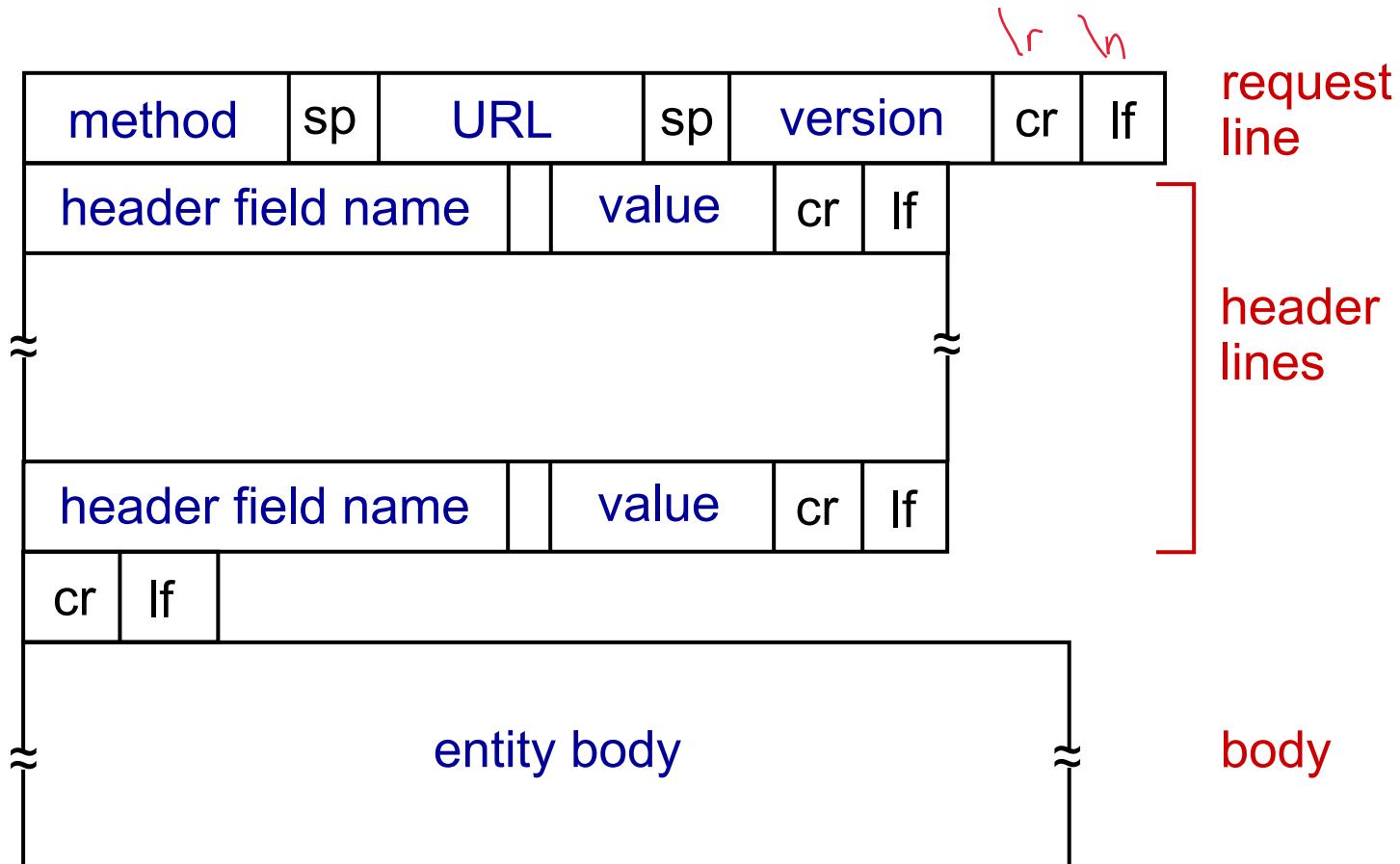
Connection: keep-alive\r\n

...

\r\n

Extra "blank" line indicates
the end of header lines

HTTP Request Message: General Format



HTTP Request Method Types

HTTP/1.0:

- ❖ GET
- ❖ POST
 - web page often includes form input
 - input is uploaded to server in entity body
- ❖ HEAD
 - asks server to leave requested object out of response

HTTP/1.1:

- ❖ GET, POST, HEAD
- ❖ PUT
 - uploads file in entity body to path specified in URL field
- ❖ DELETE
 - deletes file specified in the URL field

Example HTTP Response Message

status line
(protocol status code)

client request is successful - shows the response to the request here

HTTP/1.1 200 OK\r\n

Date: Wed, 23 Jan 2019 13:11:15 GMT\r\n

Content-Length: 606\r\n

Content-Type: text/html\r\n

...

\r\n

data data data data data ...

header
lines

data, e.g.
requested
HTML file

For a full list of request/response header fields, check
<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

HTTP Response Status Codes

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

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

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

403 Forbidden

- server declines to show the requested webpage

404 Not Found

- requested document not found on this server

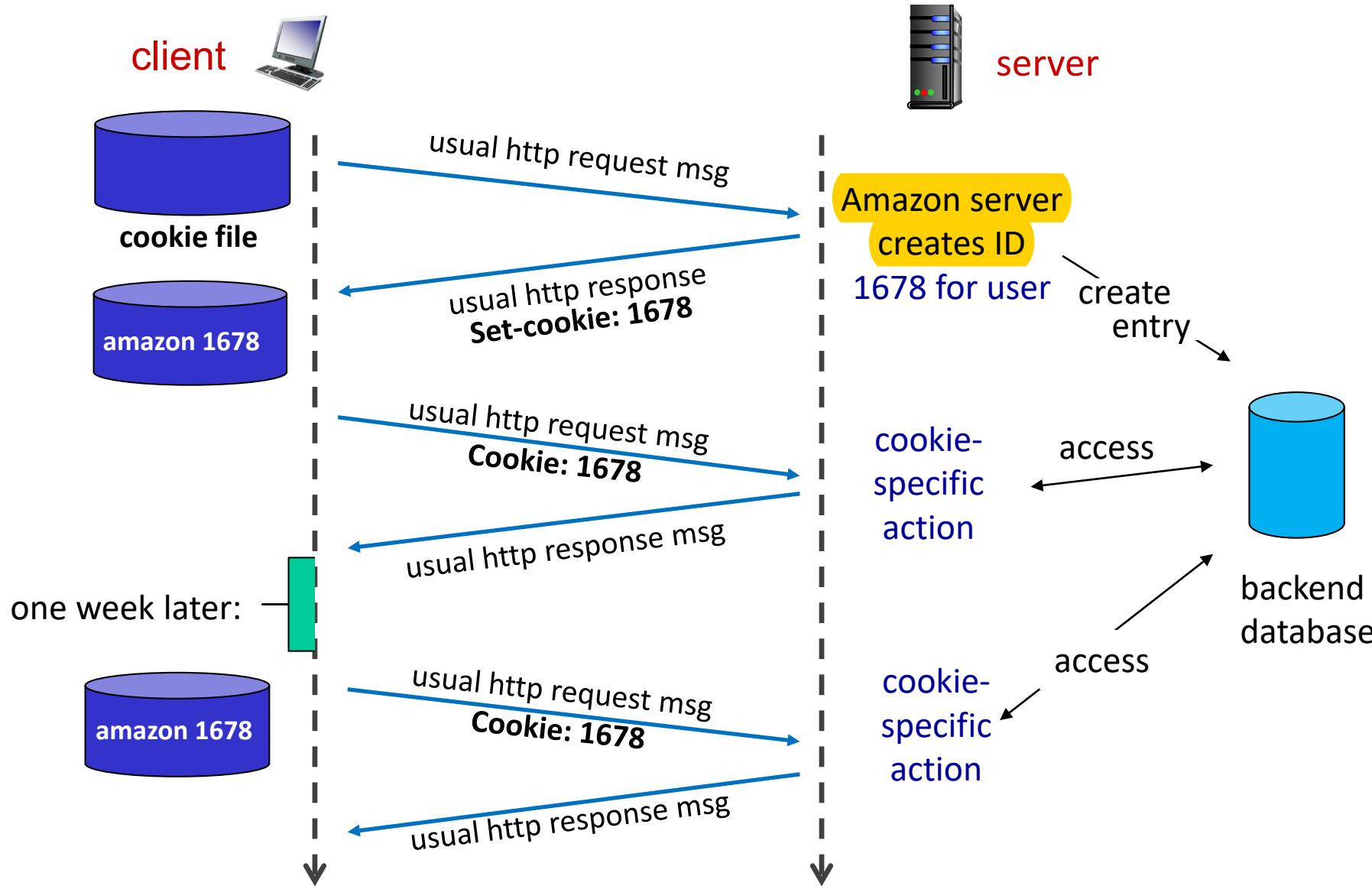
For a full list of status codes, check

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Cookies

- ❖ HTTP is designed to be “stateless”.
 - Server maintains no information about past client requests.
- ❖ Sometimes it’s good to maintain states (history) at server/client over multiple transactions.
 - E.g. shopping carts
- ❖ Cookie: http messages carry “state”
 - 1) cookie header field of HTTP *request / response* messages can be used as a token to prove identity
 - 2) cookie file kept on user’s host, managed by user’s browser
 - 3) back-end database at Web site

Keep User States with Cookie



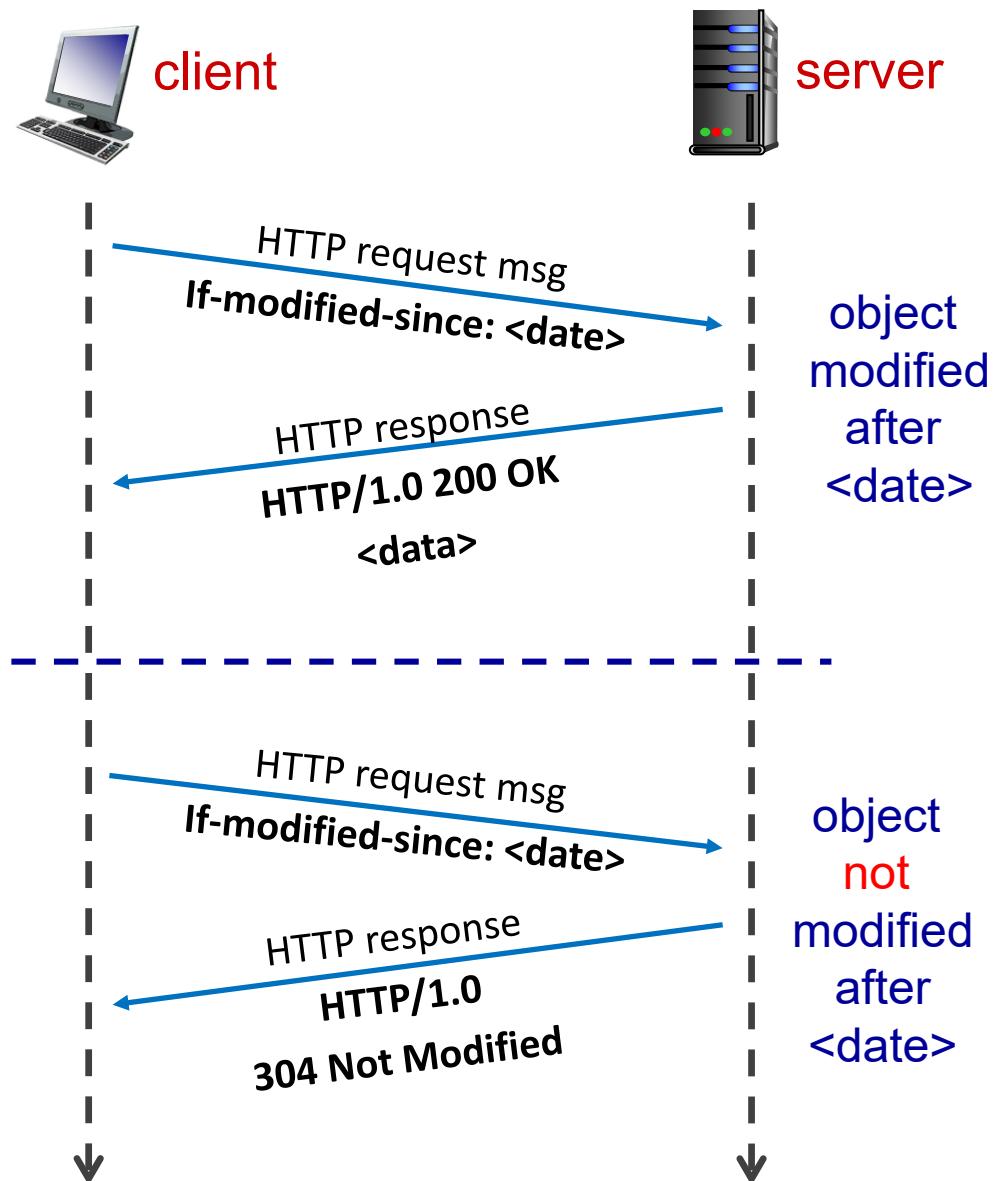
Conditional GET

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

saves bandwidth



Lectures 2&3: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.4 DNS

2.7 Socket programming

Domain Name System [RFC 1034, 1035]

- ❖ Two ways to identify a host:
 - **Hostname**, e.g., www.example.org
 - **IP address**, e.g., 93.184.216.34
- ❖ **DNS (Domain Name System)** translates between the two.
 - A client must carry out a DNS query to determine the IP address corresponding to the server name (e.g., www.example.org) prior to the connection.

 canonical name = actual / true name

DNS: Resource Records (RR)

- ❖ Mapping between host names and IP addresses (and others) are stored as resource records (RR).

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

type = A

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

type = NS name server

- **name** is domain (e.g., **nus.edu.sg**)
- **value** is hostname of authoritative name server for this domain

type = CNAME

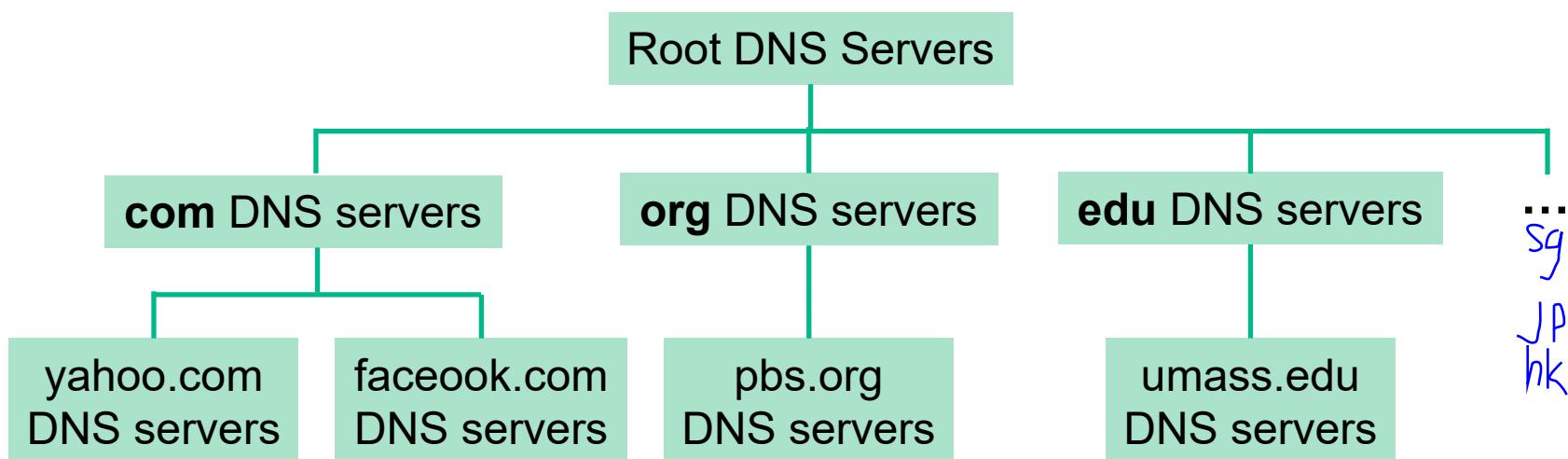
- **name** is alias name (e.g. **www.nus.edu.sg**) for some “canonical” (the real) name
- **value** is canonical name (e.g. **mgnzsqc.x.incapdns.net**)

type = MX mail exchanger = email server

- **value** is name of mail server associated with **name**

Distributed, Hierarchical Database

- ❖ DNS stored RR in distributed databases implemented in hierarchy of many name servers.

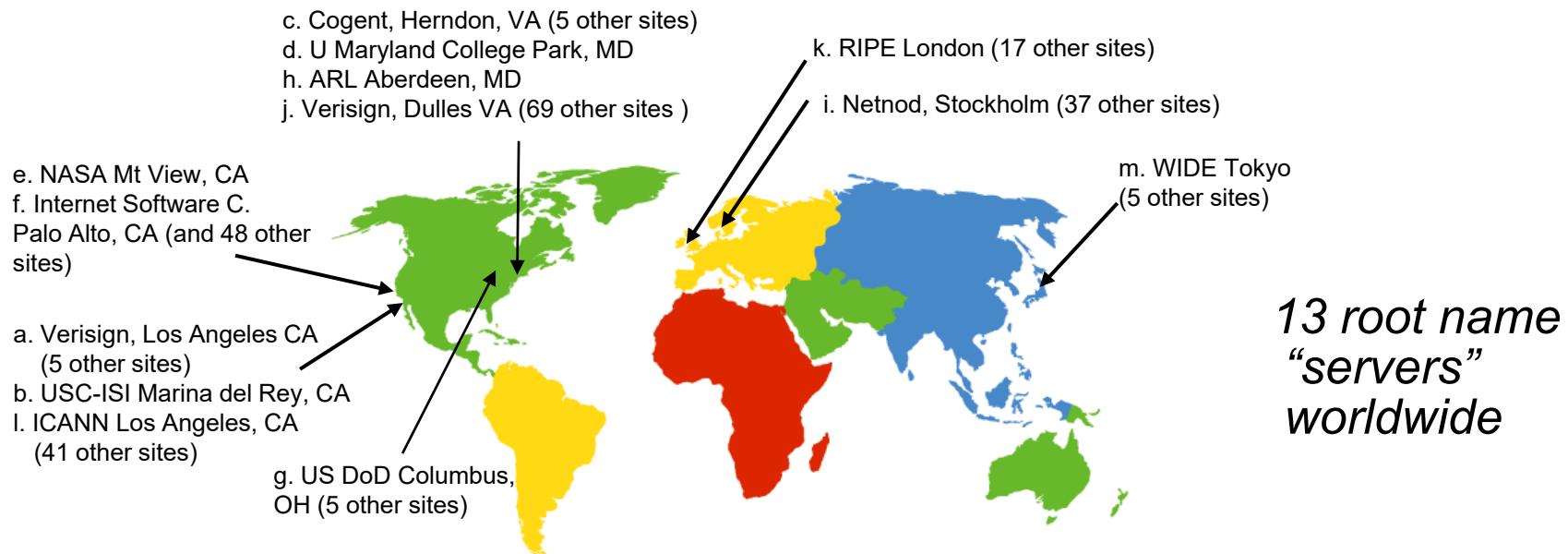


A client wants IP address for www.facebook.com:

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

Root Servers

- ❖ Answers requests for records in the root zone by returning a list of the authoritative name servers for the appropriate top-level domain (TLD).



TLD and Authoritative Servers

Top-level domain (TLD) servers:

- ❖ responsible for com, org, net, edu, ... and all top-level country domains, e.g., uk, sg, jp

Authoritative servers:

- ❖ Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts (e.g. Web, mail)
- ❖ can be maintained by organization or service provider

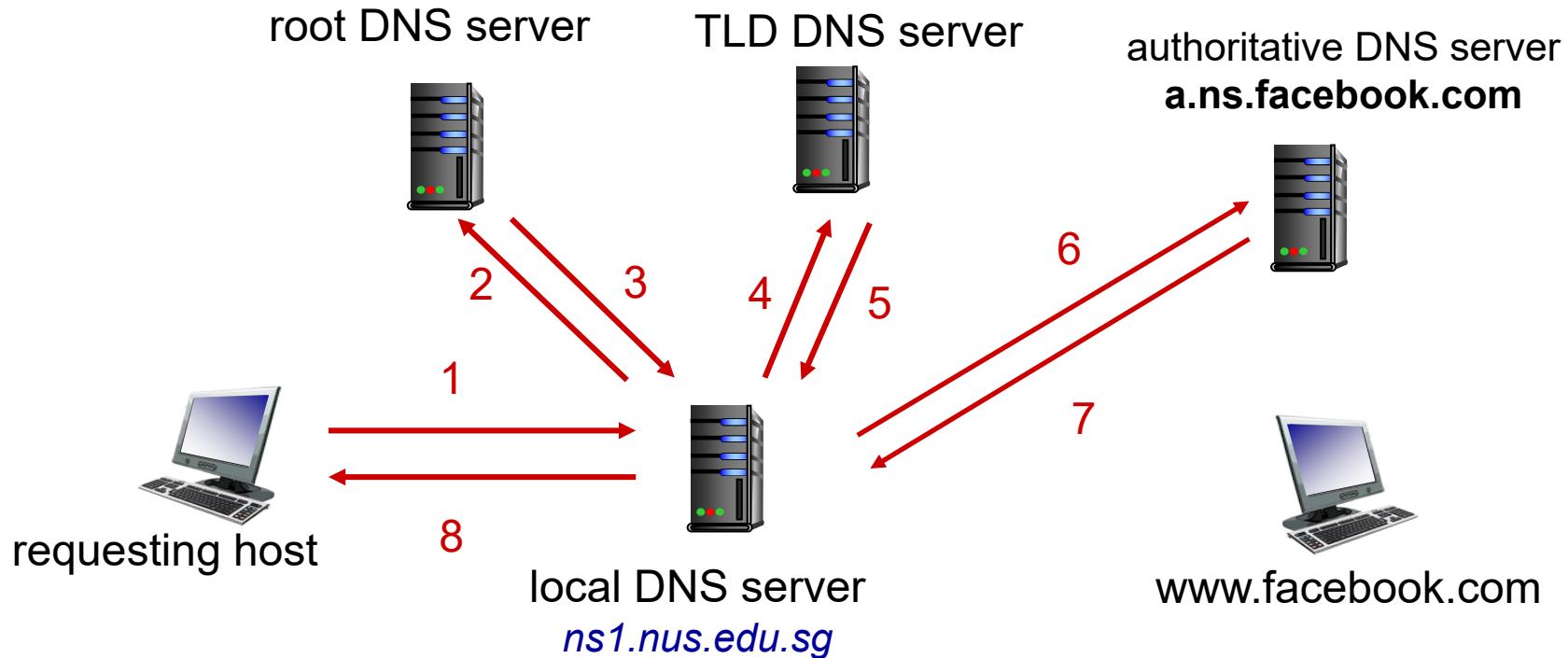
Local DNS Server

- ❖ Does not strictly belong to hierarchy
- ❖ Each ISP (residential ISP, company, university) has one local DNS server.
 - also called “default name server”
- ❖ When host makes a DNS query, query is sent to its local DNS server
 - Retrieve name-to-address translation from local cache
 - Local DNS server acts as proxy and forwards query into hierarchy if answer is not found locally

DNS Caching

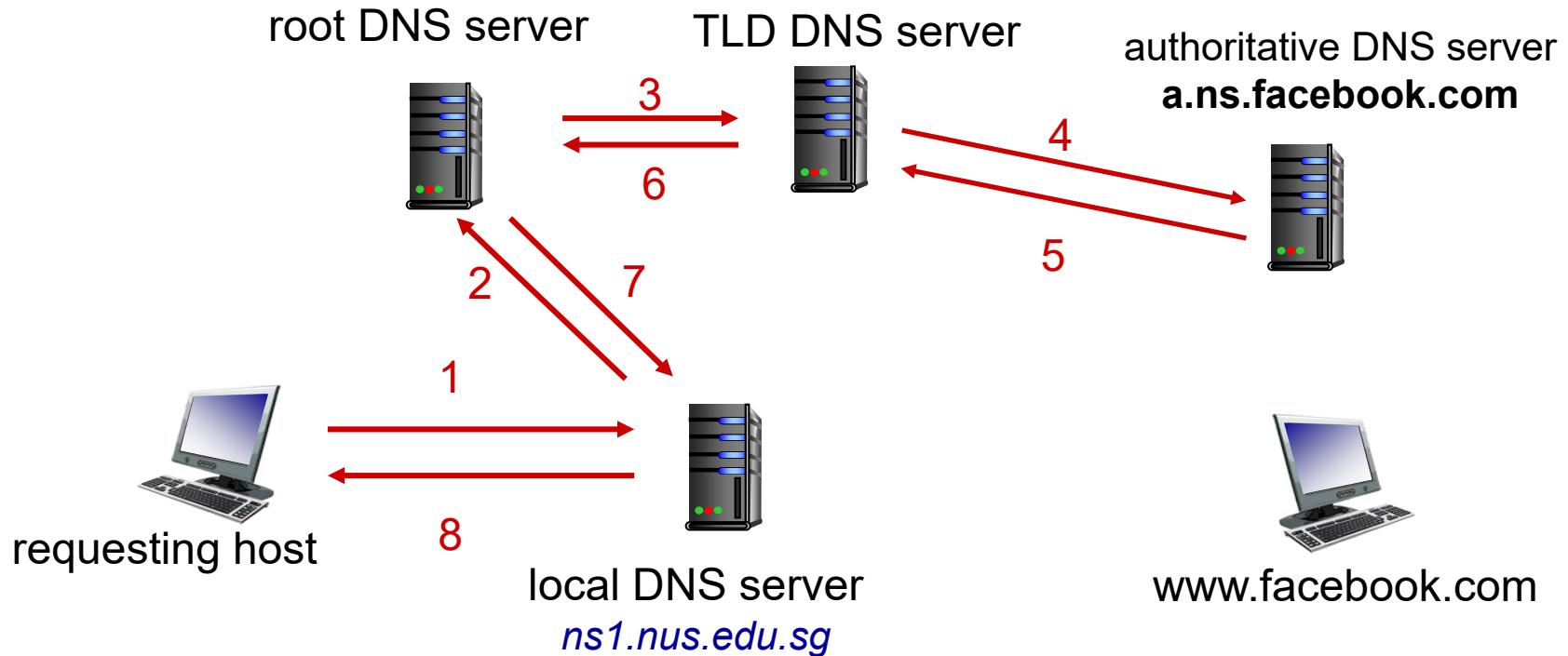
- ❖  Once a name server learns mapping, it *caches* mapping.
 - cached entries may be out-of-date (best effort name-to-address translation!)
 - cached entries expire after some time (**TTL**).
 - if name host changes IP address, may not be known Internet-wide until all TTLs expire.
- ❖ Update/notify mechanisms proposed IETF standard
 - RFC 2136
- ❖ DNS runs over  **JDP**.

DNS Name Resolution



- ❖ This is known as *iterative query*.

DNS Name Resolution



- ❖ This is known as *recursive query*.
 - rarely used in practice

Lectures 2&3: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.4 DNS

2.7 Socket programming: Creating Network Applications

Processes

- ❖ Process: **program** running within a host.
 - Within the same host, two processes communicate using **inter-process communication** (defined by OS).
 - Processes in different hosts communicate by exchanging **messages** (according to protocols).

In C/S model

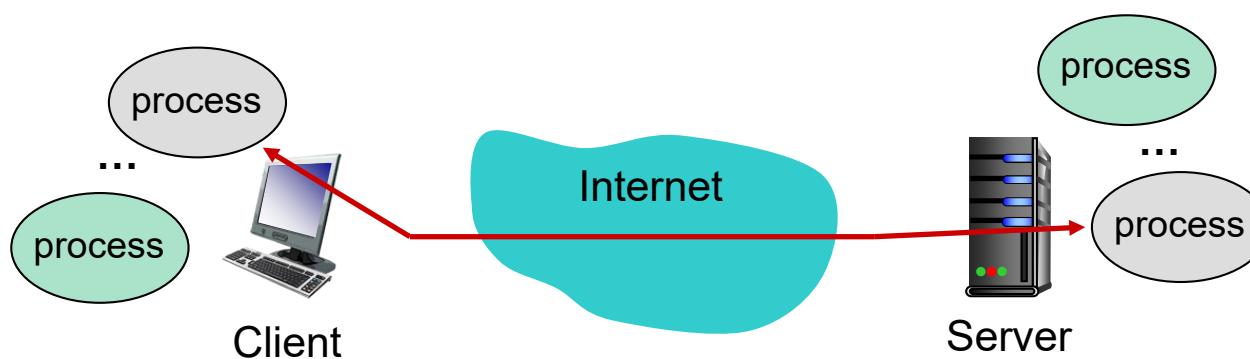
server process waits to be contacted

client process initiates the communication

Addressing Processes

- ❖ IP address is used to identify a host
 - A 32-bit integer (e.g. 137.132.21.27)
- ❖ Question: is IP address of a host suffice to identify a process running inside that host?

A: no, many processes may run concurrently in a host.



Addressing Processes

- ❖ A process is identified by (IP address, port number).
 - Port number is 16-bit integer (1-1023 are reserved for standard use).
- ❖ Example port numbers
 - HTTP server: 80
 - SMTP server: 25
- ❖ IANA coordinates the assignment of port number:
 - <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>

Analogy

Postal service:

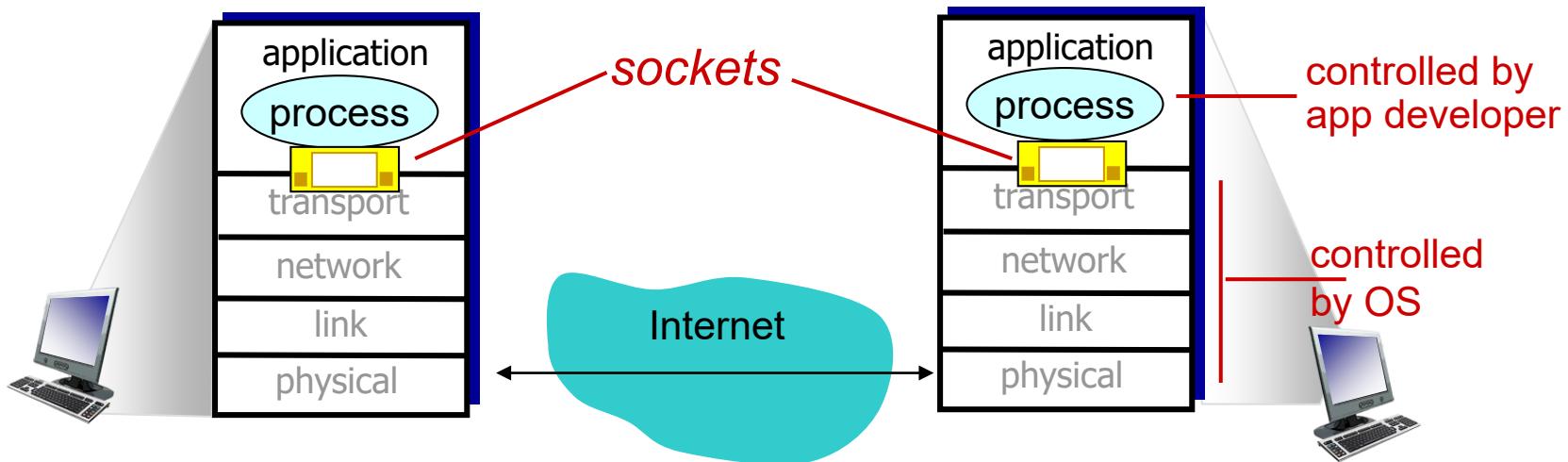
- ❖ *deliver letter to the doorstep:* home address
- ❖ *dispatch letter to the right person in the house:* name of the receiver as stated on the letter

Protocol service:

- ❖ *deliver packet to the right host:* IP address of the host
- ❖ *dispatch packet to the right process in the host:* port number of the process

Sockets

- ❖ **Socket** is the software interface between app processes and transport layer protocols.
 - Process sends/receives messages to/from its **socket**.
 - Programming-wise: a set of **APIs**



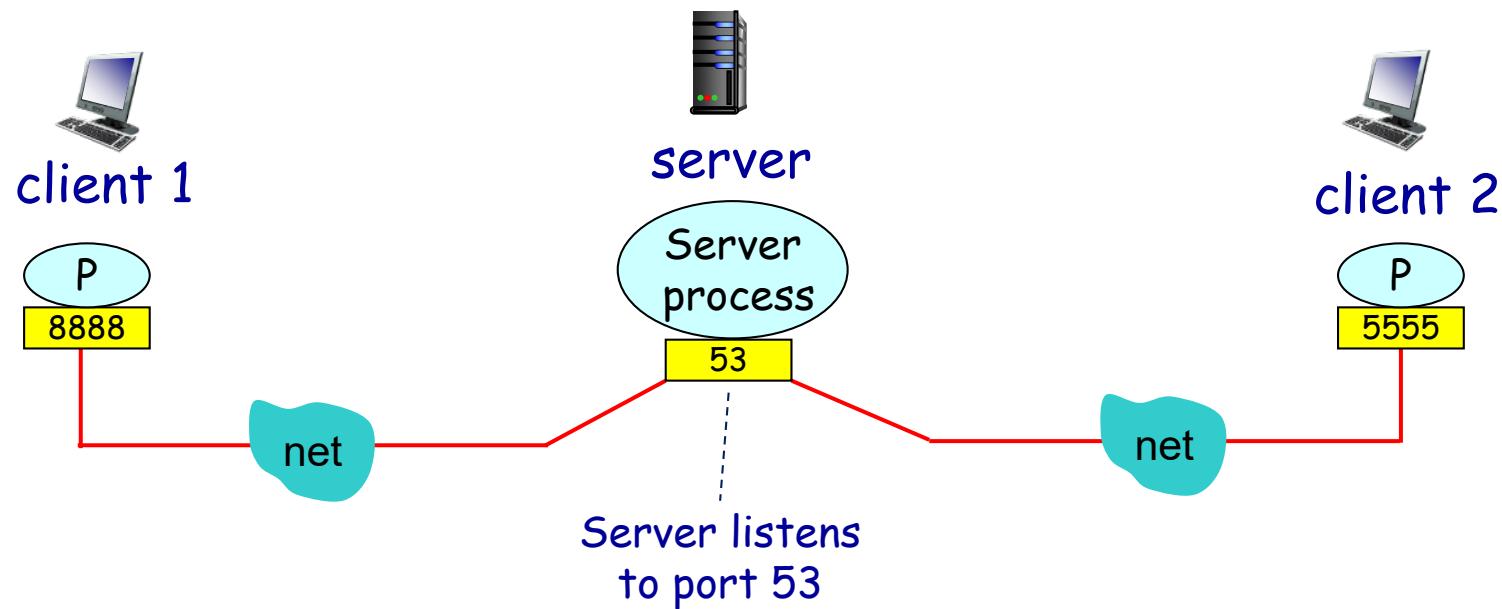
Socket Programming

- ❖ Applications (or processes) treat the Internet as a black box, sending and receiving messages through sockets.
- ❖ Two types of sockets
 - **TCP:** reliable, byte stream-oriented socket
 - **UDP:** unreliable datagram socket
- ❖ Now let's write a simple client/server application that **client sends a line of text to server, and server echoes it.**
 - We will demo both **TCP socket version** and **UDP socket version.**

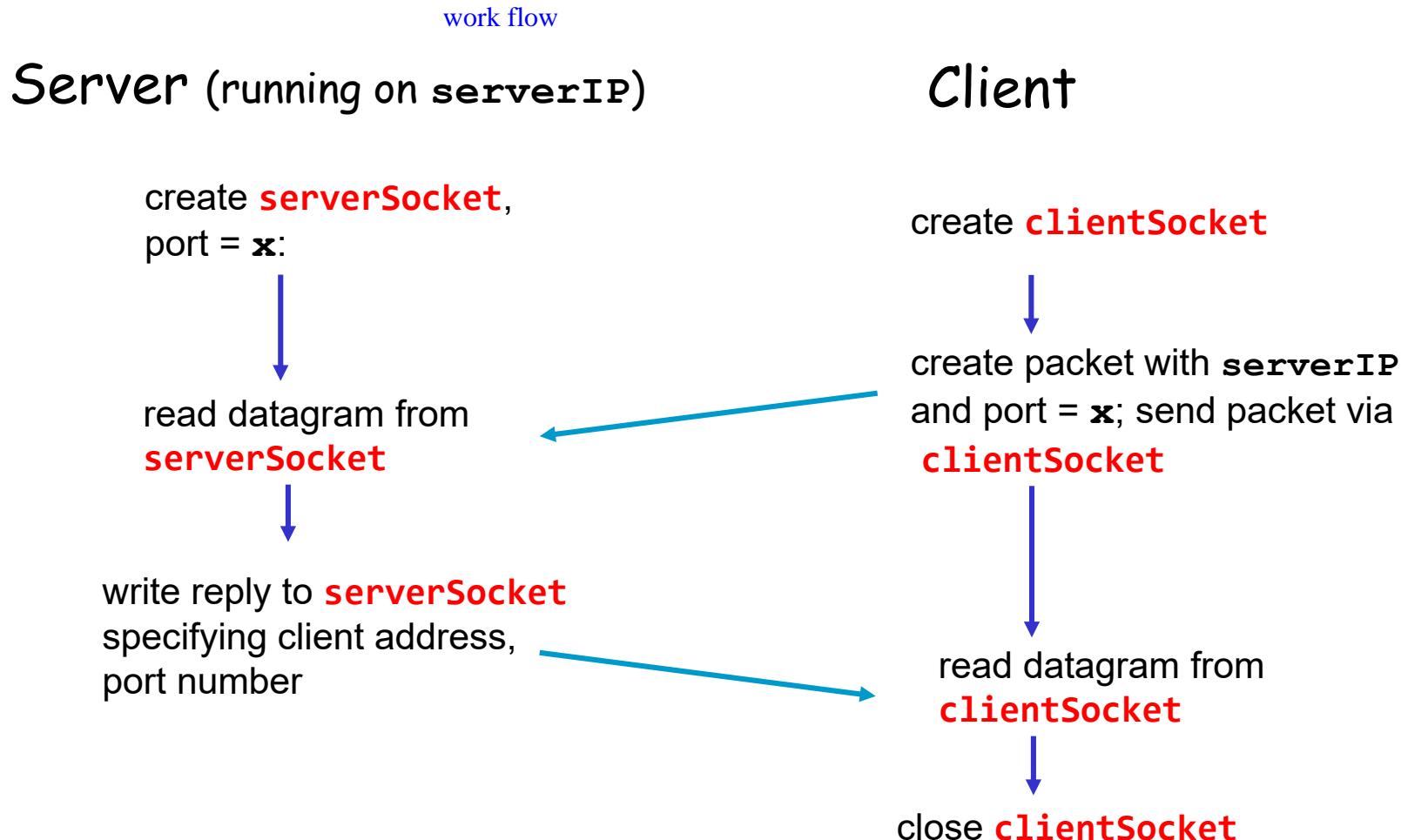
Socket Programming with *UDP*

UDP: no “connection” between client and server

- ❖ Sender (client) explicitly attaches destination IP address and port number to each packet.
- ❖ Receiver (server) extracts sender IP address and port number from the received packet.



UDP: Client/server Socket Interaction



Example: UDP Echo Server

code for udp echo

```
from socket import * ← include Python's socket library
```

```
serverPort = 2105
```

IPv4

UDP socket

```
# create a socket
```

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
```

```
# bind socket to local port number 2105
```

```
serverSocket.bind(('', serverPort))
```

```
print('Server is ready to receive message')
```

receive datagram
buffer size: 2048B

```
# extract client address from received packet
```

```
message, clientAddress = serverSocket.recvfrom(2048)
```

```
serverSocket.sendto(message, clientAddress)
```

```
serverSocket.close()
```

Example: UDP Echo Client

```
from socket import *

serverName = 'localhost' # client, server on the same host
serverPort = 2105

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input('Enter a message: ')
# send msg to server address
clientSocket.sendto(message.encode(), (serverName, serverPort))

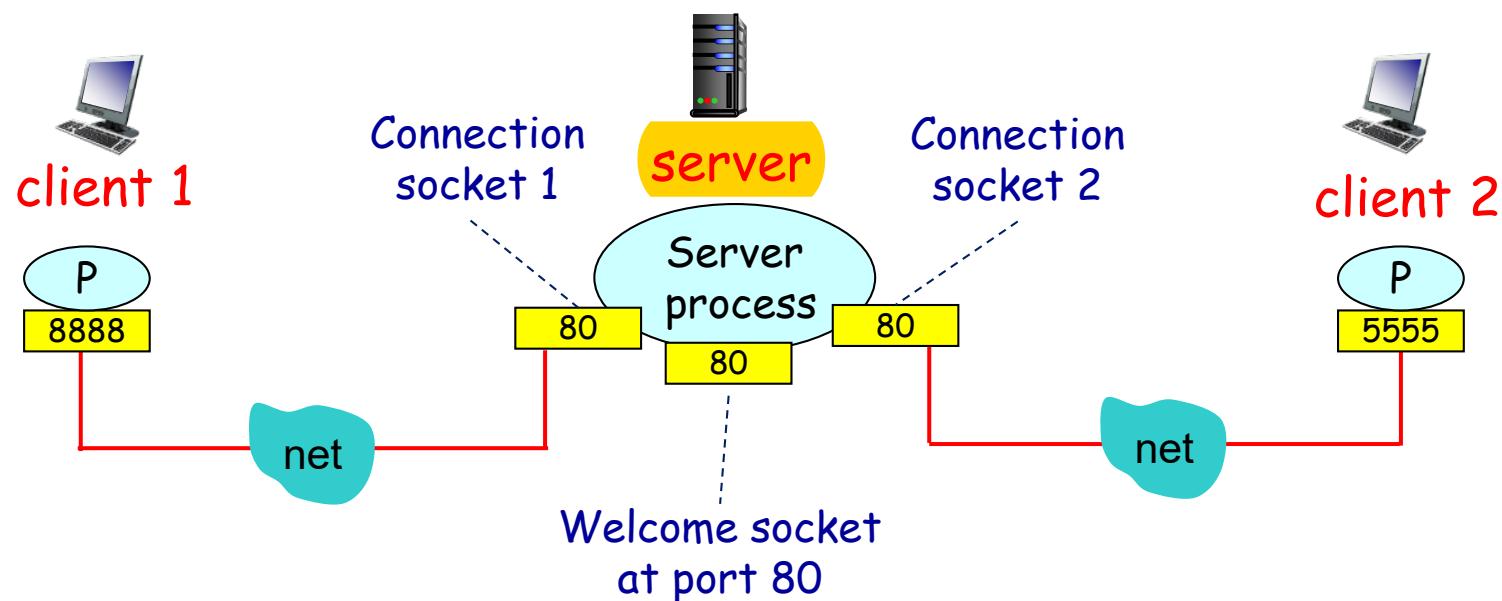
receivedMsg, serverAddress = clientSocket.recvfrom(2048)
print('from server:', receivedMsg.decode())
clientSocket.close()
```

convert message from string to bytes and send it

convert from bytes to string

Socket Programming with *TCP*

- ❖ When client creates socket, client TCP establishes a connection to server TCP.
- ❖ When contacted by client, **server TCP creates a new socket** for server process to communicate with that client.
 - allows server to talk with multiple clients individually.



TCP: Client/server Socket Interaction

Server (running on `serverIP`)

Client

create `serverSocket`, port = `x`

wait for incoming
connection request
`connectionSocket`

read request from
`connectionSocket`

write reply to
`connectionSocket`

close `connectionSocket`

TCP
connection setup

create `clientSocket`,
connect to `serverIP`, port = `x`

send request using `clientSocket`

read reply from `clientSocket`

close `clientSocket`

Example: TCP Echo Server

```
from socket import *
```

```
serverPort = 2105
```

TCP socket
↓

```
serverSocket = socket(AF_INET, SOCK_STREAM)
```

```
serverSocket.bind(('', serverPort))
```

listens for incoming TCP request
(not available in UDP socket)

```
serverSocket.listen()
```

```
print('Server is ready to receive message')
```

```
connectionSocket, clientAddr = serverSocket.accept()
```

```
message = connectionSocket.recv(2048)
```

```
connectionSocket.send(message)
```

no need to specify the receiver here, since the connectionSocket
is specific for this client

```
connectionSocket.close()
```

returns a new socket
to communicate with
client socket

Example: TCP Echo Client

```
from socket import *

serverName = 'localhost'
serverPort = 2105

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort)) ← establish a connection

message = input('Enter a message: ')

clientSocket.send(message.encode()) ← no need to attach server name, port

receivedMsg = clientSocket.recv(2048)

print('from server:', receivedMsg.decode())

clientSocket.close()
```

TCP Socket vs. UDP Socket

- ❖ In TCP, two processes communicate as if there is a pipe between them. The pipe remains in place until one of the two processes closes it.
 - When one of the processes wants to send more bytes to the other process, it simply writes data to that pipe.
 - The sending process doesn't need to attach a destination IP address and port number to the bytes in each sending attempt as the logical pipe has been established (which is also reliable).
- ❖ In UDP, programmers need to form UDP datagram packets explicitly and attach destination IP address / port number to every packet.

Lectures 2&3: Summary

- ❖ Application architectures
 - Client-server
 - P2P
- ❖ Application service requirements:
 - reliability, throughput, delay, security
- ❖ Specific protocols:
 - HTTP
 - DNS
- ❖ Internet transport service model
 - connection-oriented, reliable: TCP
 - Connection-less, unreliable: UDP

Lectures 2&3: Summary

❖ Socket programming

- **TCP socket**
 - When contacted by client, server TCP creates new socket.
 - Server uses **(client IP + port #)** to distinguish clients.
 - When client creates its socket, client TCP establishes connection to server TCP.
- **UDP socket**
 - Server uses **one socket** to serve all clients.
 - No connection is established before sending data.
 - Sender explicitly attaches **destination IP address** and **port #** to each packet.
 - Transmitted data may be lost or received out-of-order.

CS2105

An Awesome Introduction to Computer Networks

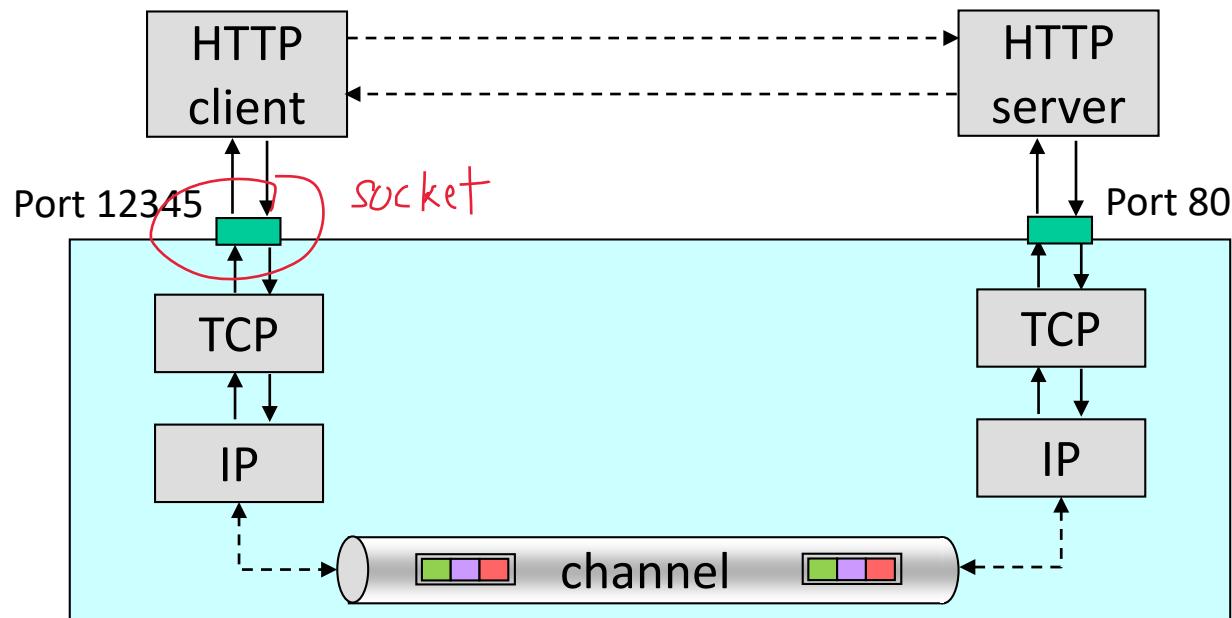
Lectures 4&5: The Transport Layer



Department of Computer Science
School of Computing

Web and HTTP

- ❖ A Web page consists of a *base HTML file* and *some other objects* referenced by the HTML file.
- ❖ HTTP uses **TCP** as transport service.
 - TCP, in turn, uses service provided by **IP**!

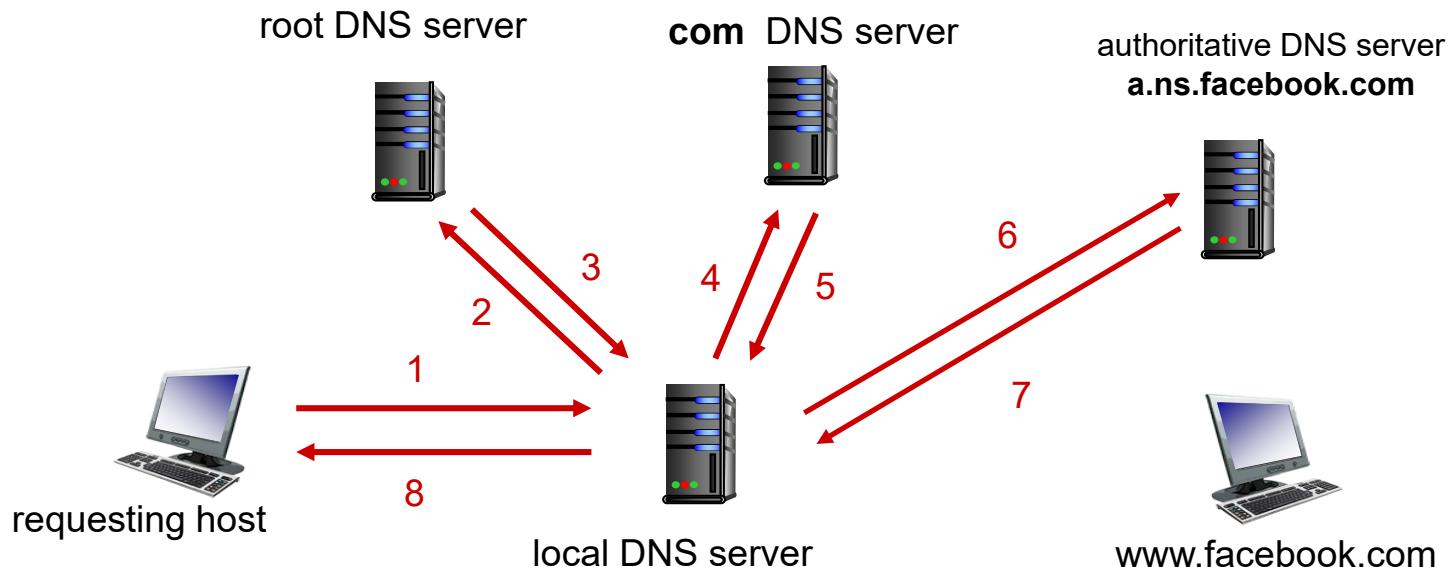


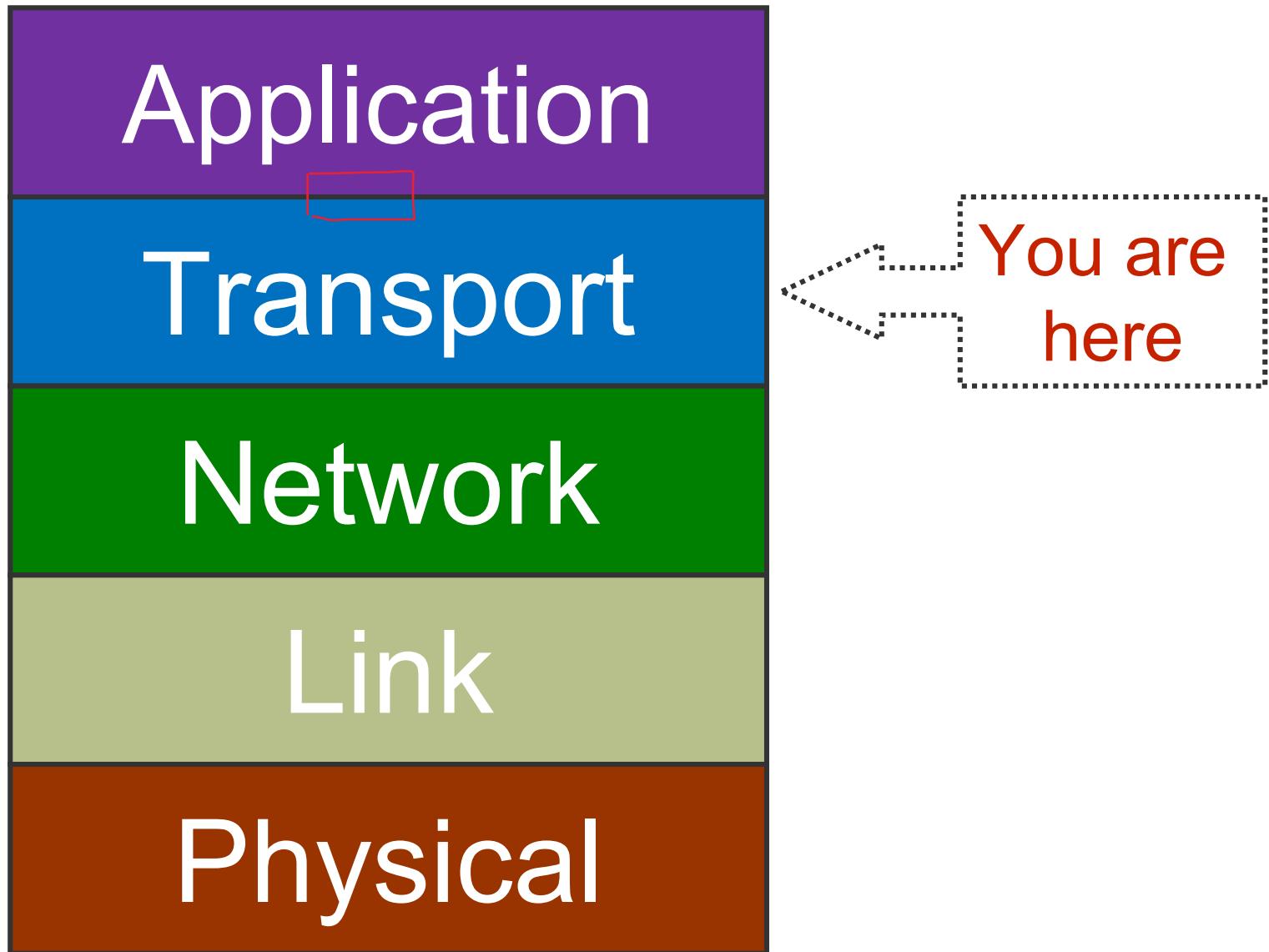
Socket

- ❖ Applications (processes) send messages over the network through sockets.
 - Conceptually, socket = IP address + port number
 - Programming wise, socket = a set of APIs
- ❖ UDP socket
 - Server uses **one socket** to serve all clients.
 - **No connection** is established before sending data.
 - Sender explicitly attaches **destination IP address + port #**.
- ❖ TCP socket
 - Server creates **a new socket** for each client.
 - Client establishes **connection** to server.
 - Server uses **connection** to identify client.

Domain Name System

- ❖ DNS is the Internet's primary directory service.
 - It translates **host names**, which can be easily memorized by humans, to **numerical IP addresses** used by hosts for the purpose of communication.





Lectures 4&5: The Transport Layer

After this class, you are expected to:

- ❖ appreciate the simplicity of UDP and the service it provides.
- ❖ know how to calculate the checksum of a packet.
- ❖ be able to design your own reliable protocols with *ACK, NAK, sequence number, timeout and retransmission.*
- ❖ understand the working of *Go-Back-N* and *Selective Repeat* protocols.
- ❖ understand the operations of TCP.

Lectures 4&5: Roadmap

3.1 Transport-layer Services

3.3 Connectionless Transport: UDP

3.4 Principles of Reliable Data Transfer

3.5 Connection-oriented Transport: TCP

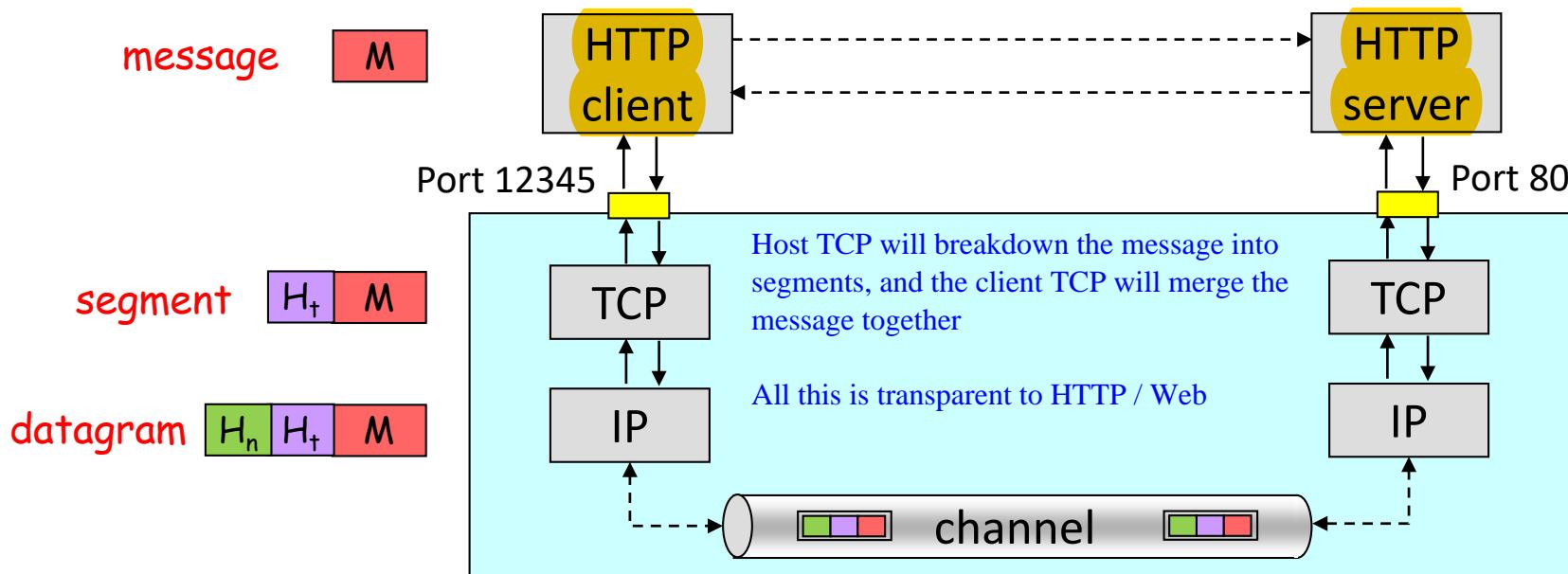
Kurose Textbook, Chapter 3
(Some slides are taken from the book)

Transport Layer Services

- ❖ Deliver messages between application processes running on different hosts
 - Two popular protocols: **TCP** and **UDP**
- ❖ Transport layer protocols run in hosts.
 - **Sender side**: breaks app message into *segments* (as needed), passes them to **network layer** (aka IP layer).
 - **Receiver side**: reassembles segments into message, passes it to app layer.
 - **Packet switches** (outers) **in between**: only check destination IP address to decide routing.

Transport / Network Layers

- ❖ Each IP datagram contains source and dest IP addresses.
 - Receiving host is identified by dest IP address.
 - Each IP datagram carries one transport-layer segment.
 - Each segment contains source and dest port numbers.



Lectures 4&5: Roadmap

3.1 Transport-layer Services

3.3 Connectionless Transport: UDP

3.4 Principles of Reliable Data Transfer

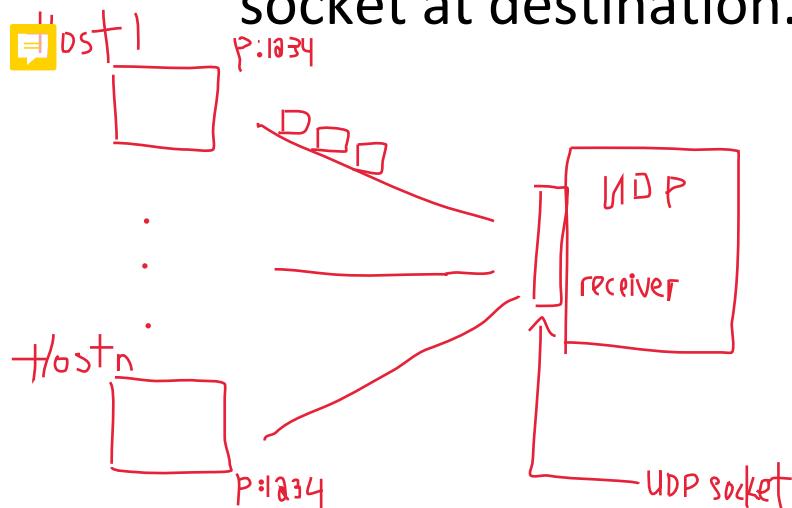
3.5 Connection-oriented Transport: TCP

UDP: User Datagram Protocol [RFC 768]

- ❖ UDP adds very little service on top of IP:
 - Multiplexing at sender: UDP gathers data from processes, forms packets and passes them to IP
 - De-multiplexing at receiver: UDP receives packets from lower layer and dispatches them to the right processes.
 - Checksum
- ❖ UDP transmission is unreliable
 - Often used by streaming multimedia apps (loss tolerant & rate sensitive)

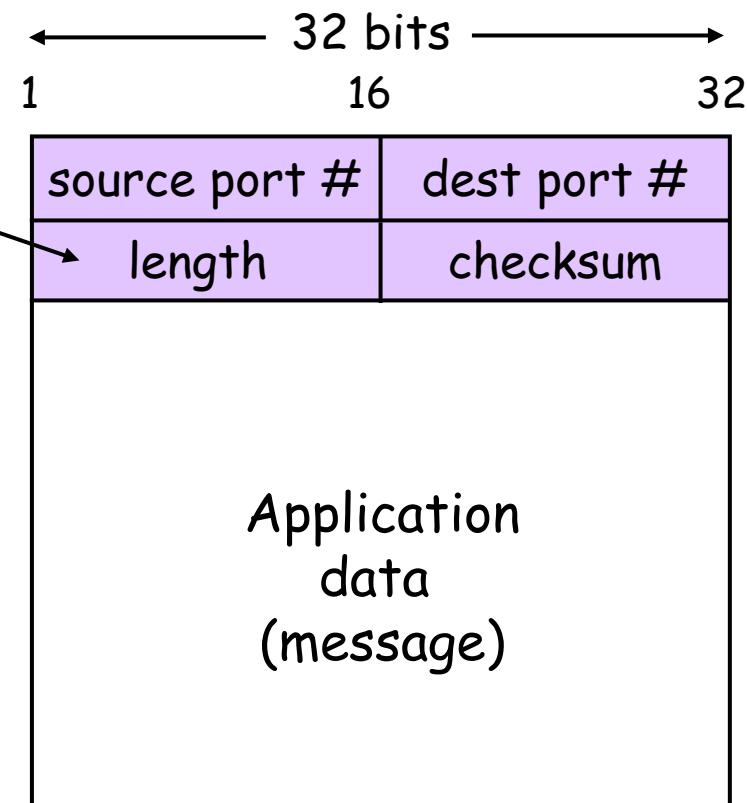
Connectionless De-multiplexing

- ❖ When **UDP receiver** receives a UDP segment:
 - Checks **destination port #** in segment.
 - Directs UDP segment to the socket with that port #.
 - IP datagrams (from different sources) with the **same destination port #** will be directed to the same UDP socket at destination.



UDP Header

Length (in bytes) of UDP segment, including header



Why is there a UDP?

- ❖ No connection establishment (which can add delay)
- ❖ Simple: no connection state at sender, receiver
- ❖ Small header size
- ❖ No congestion control: UDP can blast away as fast as desired

UDP segment format

UDP Checksum

the checksum will include everything - from the UDP header to the data that is being sent
- There is also a Pseudo header which is part of the IP address

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

Sender:

- ❖ compute checksum value (next page)
- ❖ put checksum value into UDP checksum field

Receiver:

- ❖ compute checksum of received segment
- ❖ check if computed checksum equals checksum field value:
 - NO - error detected
 - YES - no error detected
(but really no error?)

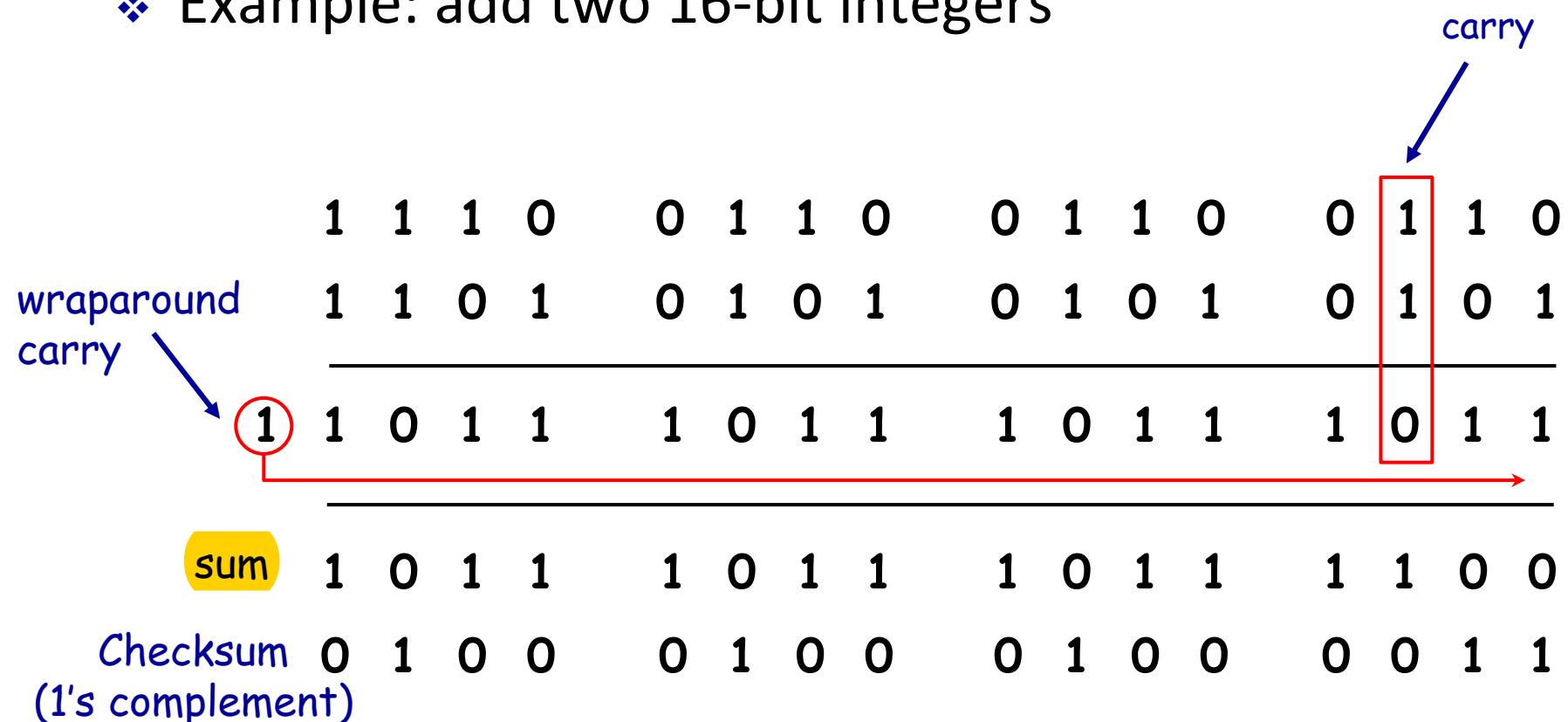
Checksum Computation

- ❖ How is UDP checksum computed?
 1. Treat UDP segment as a sequence of 16-bit integers.
 2. Apply binary addition on every 16-bit integer (checksum field is currently 0).
 3. Carry (if any) from the most significant bit will be added to the result.
 4. Compute 1's complement to get UDP checksum.

x	y	$x \oplus y$	carry
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	1

Checksum Example

- ❖ Example: add two 16-bit integers



Lectures 4&5: Roadmap

3.1 Transport-layer Services

3.3 Connectionless Transport: UDP

3.4 Principles of Reliable Data Transfer

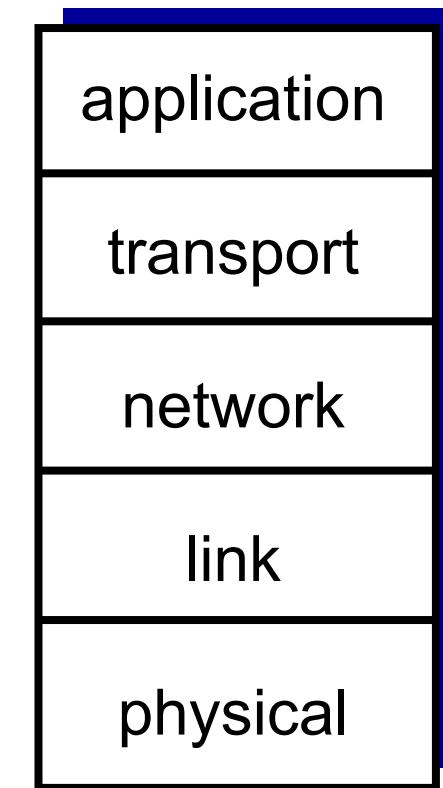
3.5 Connection-oriented transport: TCP



*“Sending Data Reliably
Over the Internet is Much
Harder Than You Think.
The Intricacy Involved in
Ensuring Reliability Will
Make Your Head Explode.”*

Transport vs. Network Layer

- ❖ **Transport layer** resides on end hosts and provides **process-to-process** communication.
- ❖ **Network layer** provides **host-to-host, best-effort** and **unreliable** communication.

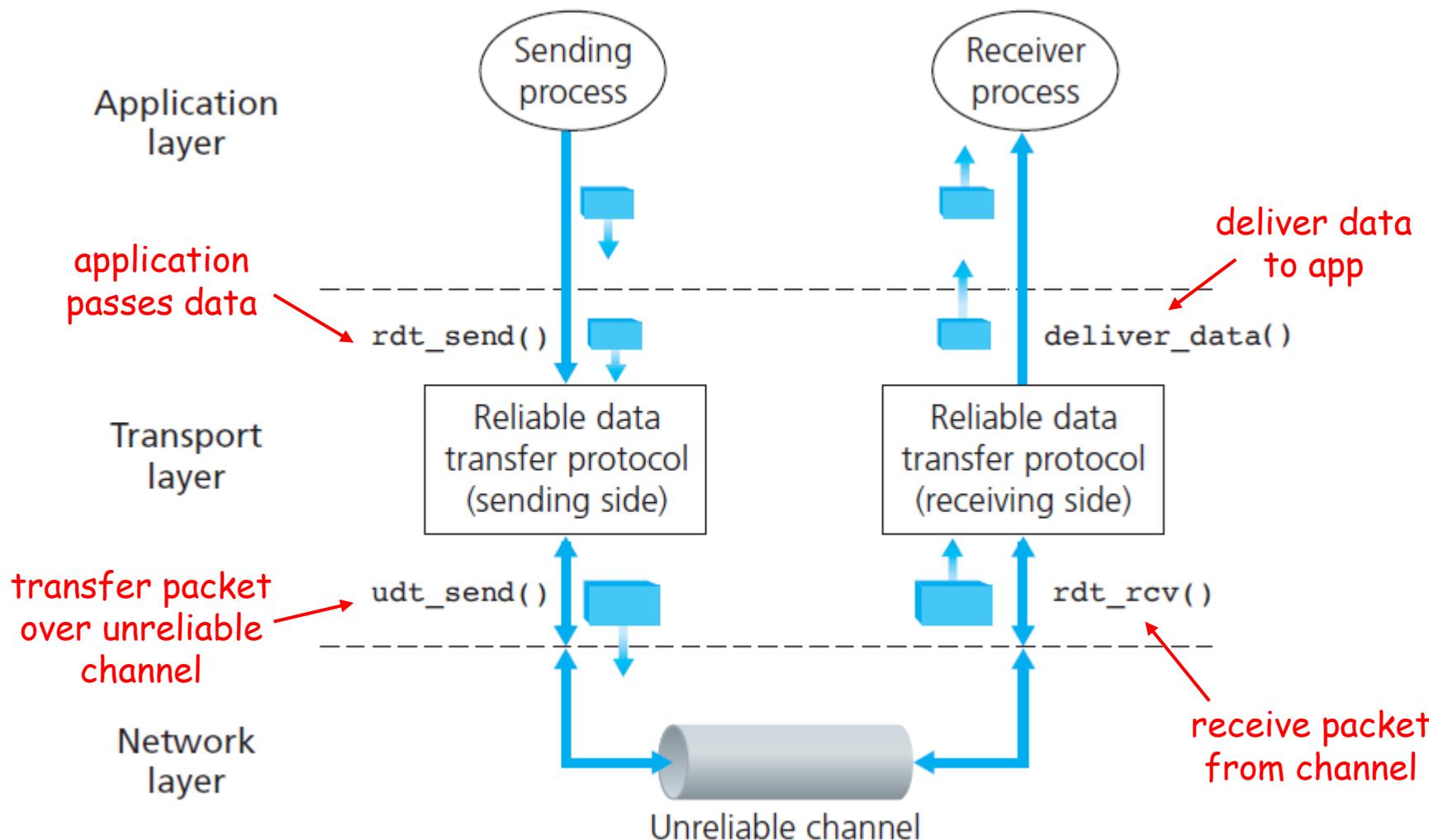


Question: How to build a **reliable transport layer protocol** on top of **unreliable communication**?

Reliable Transfer over Unreliable Channel

- ❖ Underlying network may
 - corrupt packets
 - drop packets
 - re-order packets (not considered in this lecture)
 - deliver packets after an arbitrarily long delay
- ❖ End-to-end reliable transport service should
 - guarantee packets delivery and correctness
 - deliver packets (to receiver application) in the same order they are sent

Reliable Data Transfer: Service Model

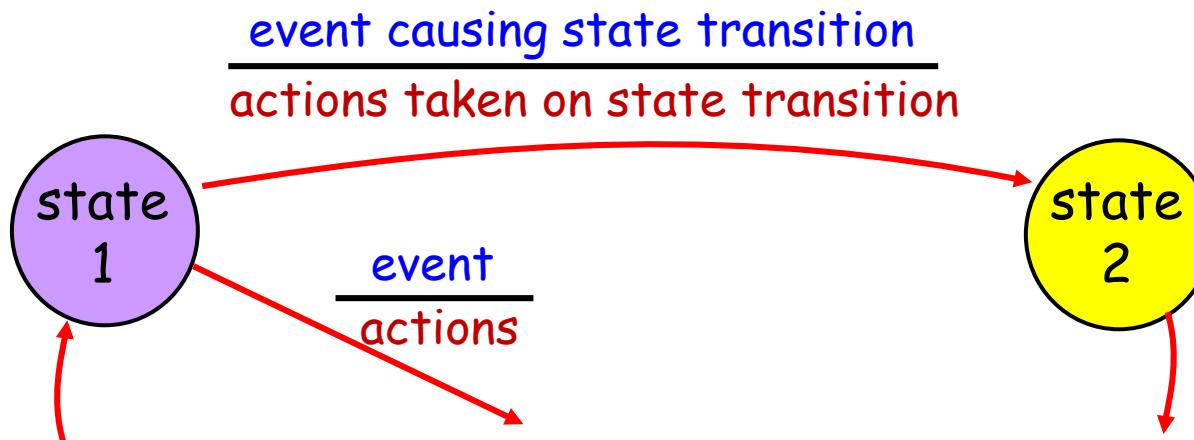


Reliable Data Transfer Protocols

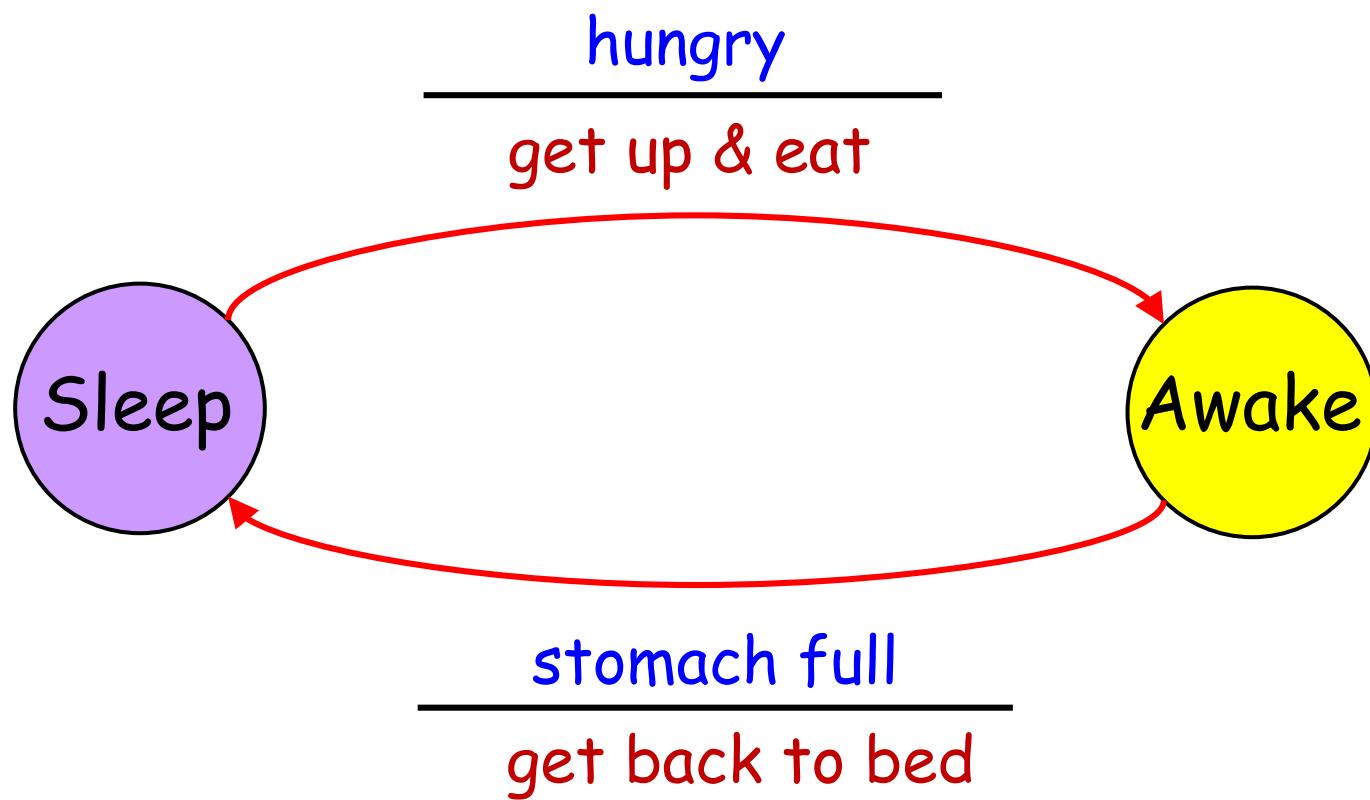
- ❖ Characteristics of unreliable channel will determine the complexity of reliable data transfer protocols (**rdt**).
- ❖ We will incrementally develop sender & receiver sides of **rdt** protocols, considering increasingly complex models of unreliable channel.
- ❖ We consider only unidirectional data transfer
 - but control info may flow in reverse direction!

Finite State Machine (FSM)

- ❖ We will use finite state machines (FSM) to describe sender and receiver of a protocol.
 - We will learn a protocol by examples, but FSM provides you the complete picture to refer to as necessary.

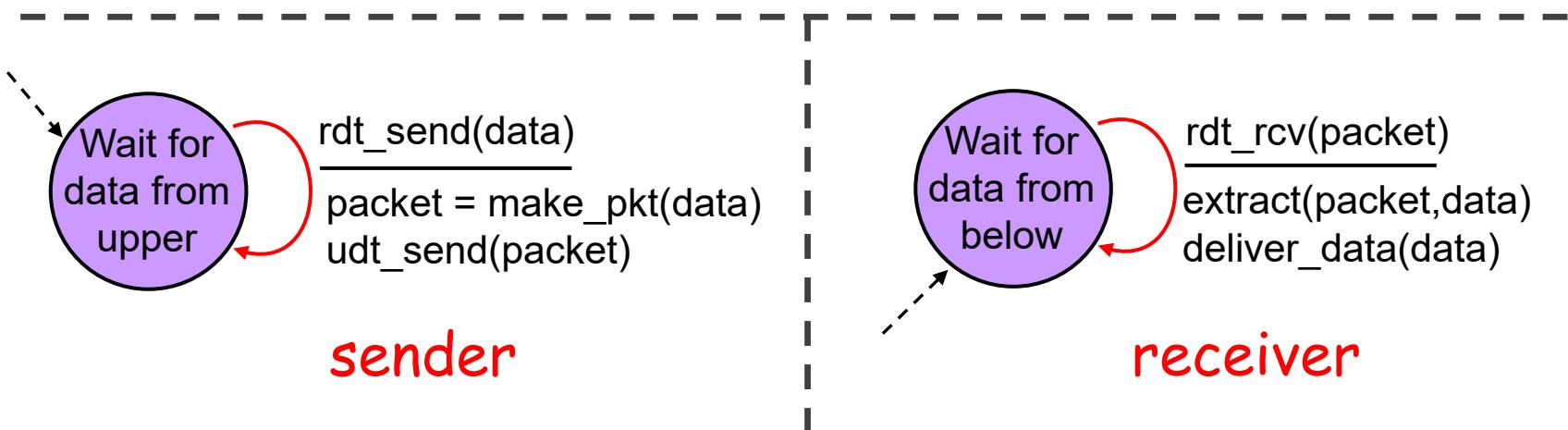


Example FSM



rdt 1.0: Perfectly Reliable Channel

- ❖ Assume underlying channel is **perfectly reliable**.
- ❖ Separate FSMs for sender, receiver:
 - Sender sends data into underlying (perfect) channel
 - Receiver reads data from underlying (perfect) channel



rdt 2.0: Channel with *Bit Errors*

❖ Assumption:

- underlying channel **may flip bits in packets**
- other than that, the channel is perfect

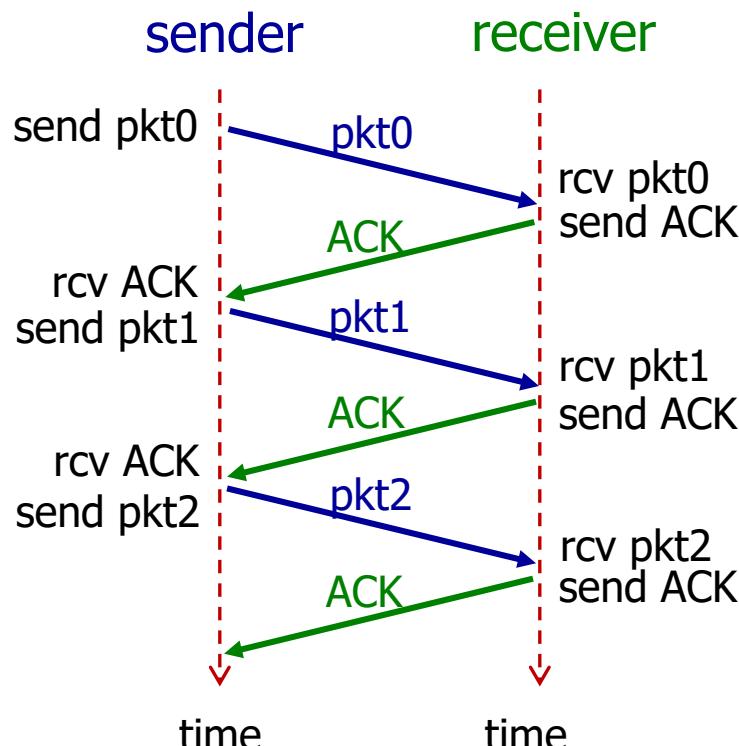
❖ Q1: how to detect bit errors?

- Receiver may use **checksum** to detect bit errors.

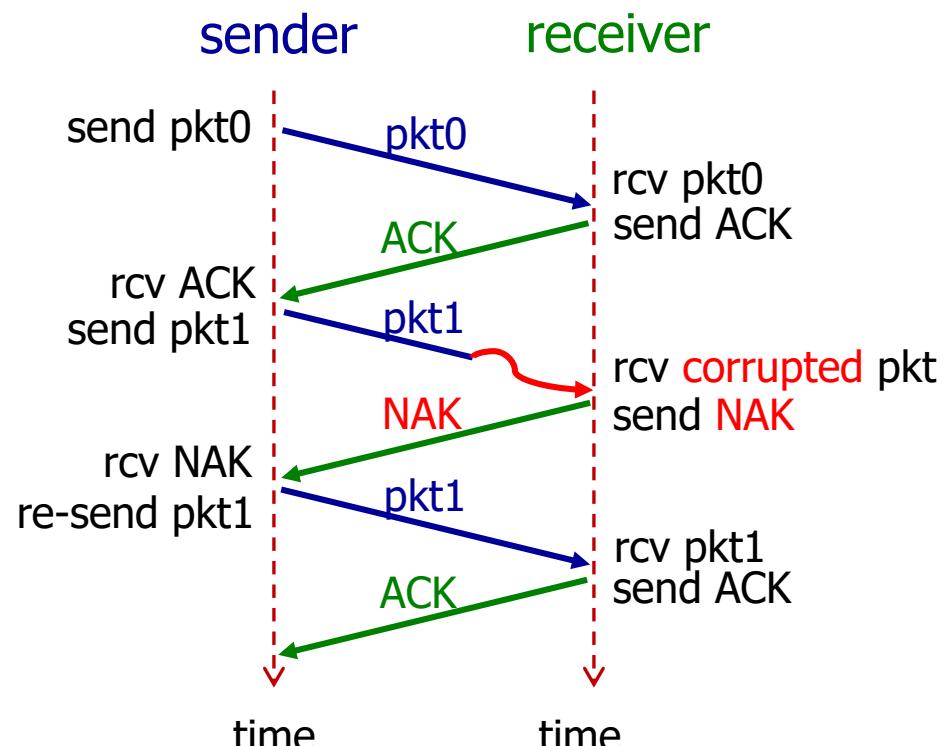
❖ Q2: how to recover from bit errors?

- *Acknowledgements (ACKs)*: receiver explicitly tells sender that packet received is OK.
- *Negative acknowledgements (NAKs)*: receiver explicitly tells sender that packet has errors.
 - Sender retransmits packet on receipt of NAK.

rdt 2.0 In Action



(a) no bit error

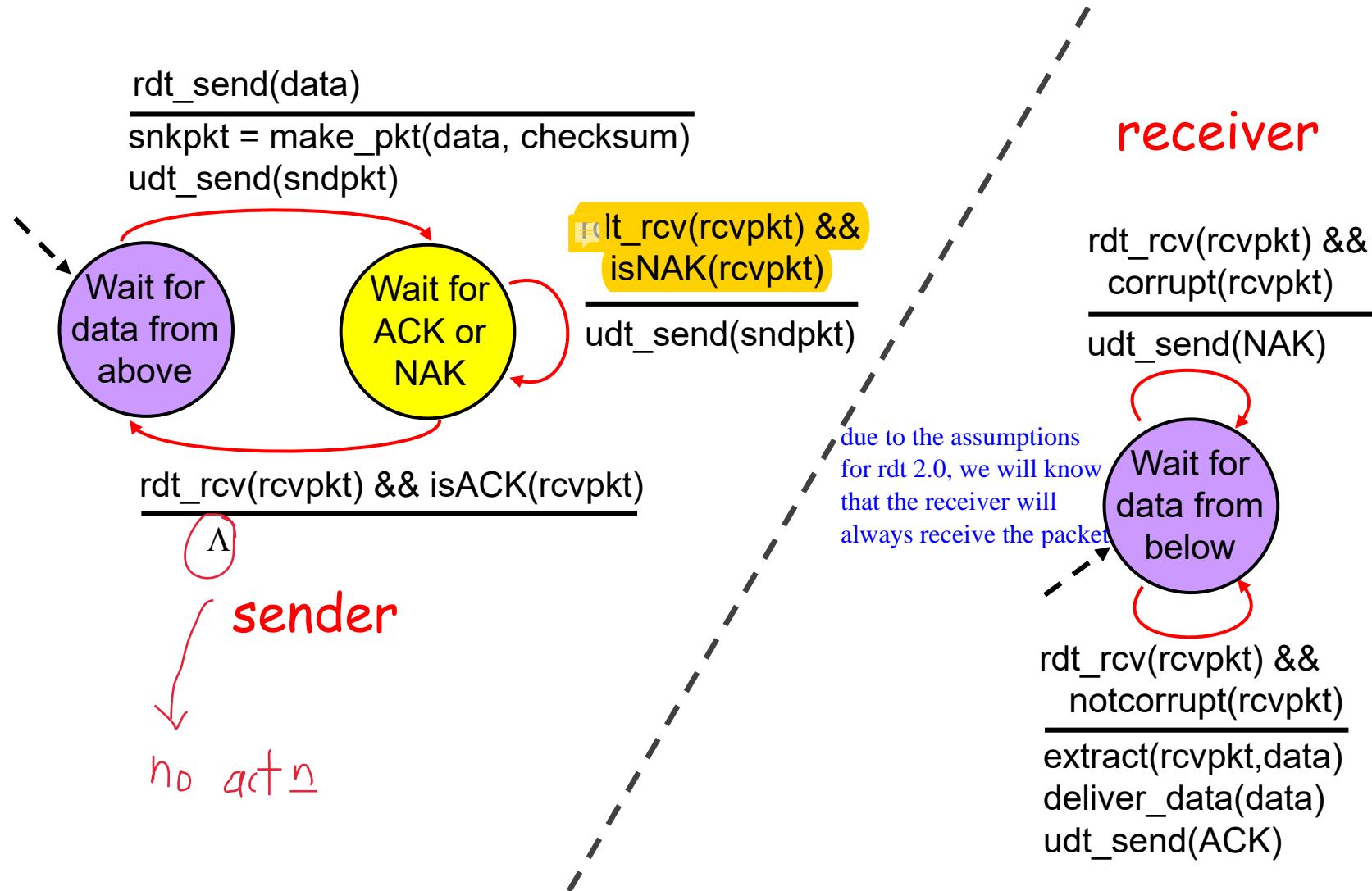


(b) with bit error

☛ top and wait protocol

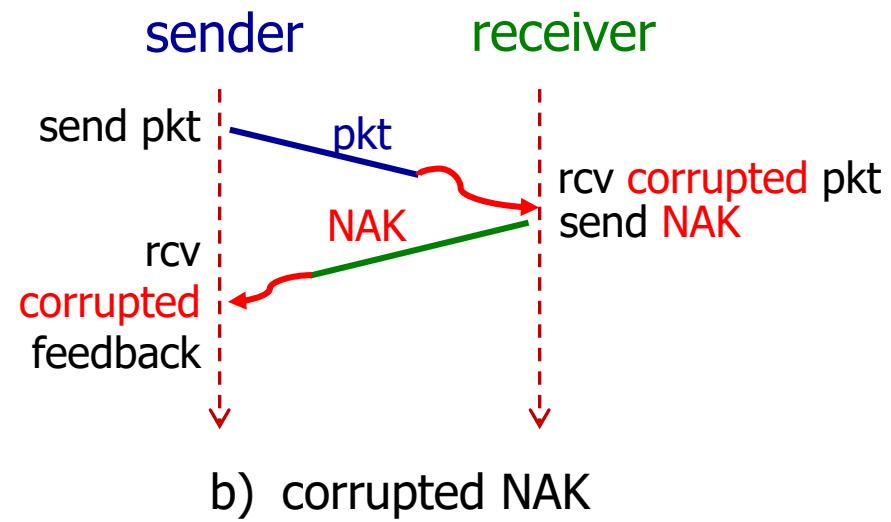
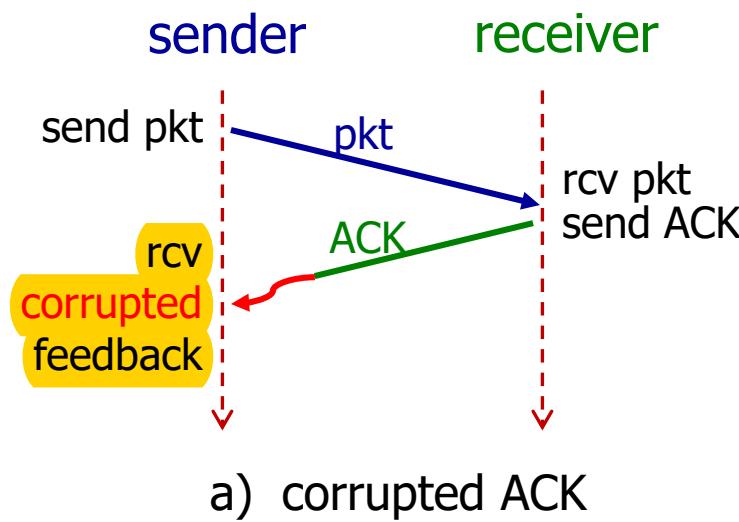
Sender sends one packet at a time, then waits for receiver response

rdt 2.0: FSM



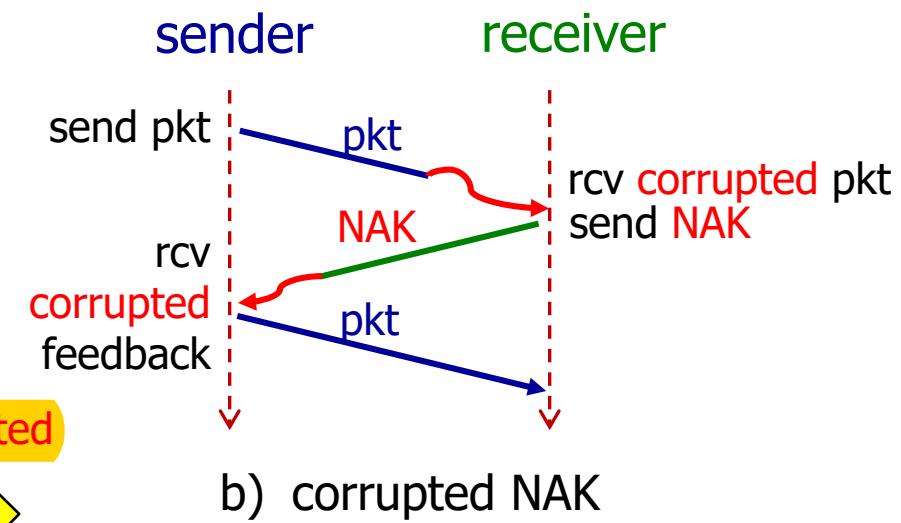
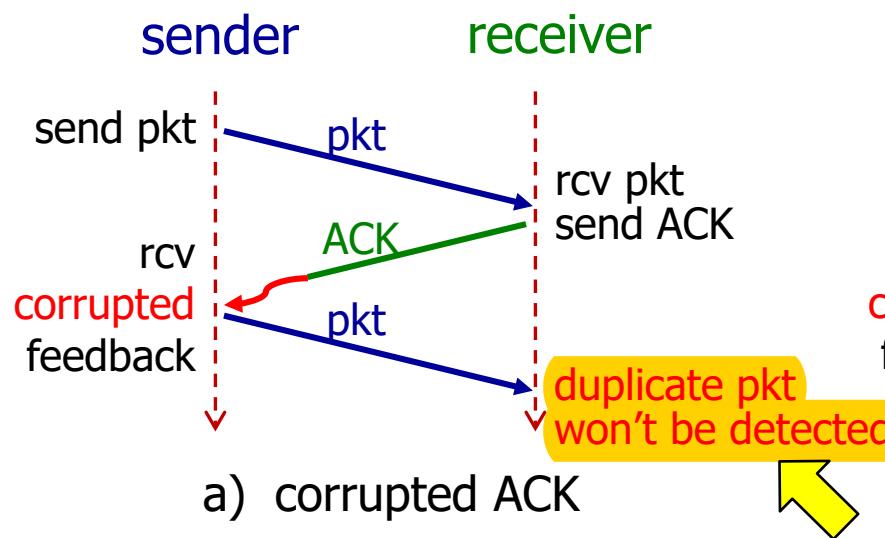
rdt 2.0 has a Fatal Flaw!

- ❖ What happens if ACK/NAK is corrupted?
 - Sender doesn't know what happened at receiver!
- ❖ So what should the sender do?
 - Sender just retransmits when receives garbled ACK or NAK.
 - **Questions:** does this work?



rdt 2.0 has a Fatal Flaw!

- ❖ Sender just retransmits when it receives garbled feedback.
 - This may cause retransmission of correctly received packet!
 - **Question:** how can receiver identify duplicate packet?

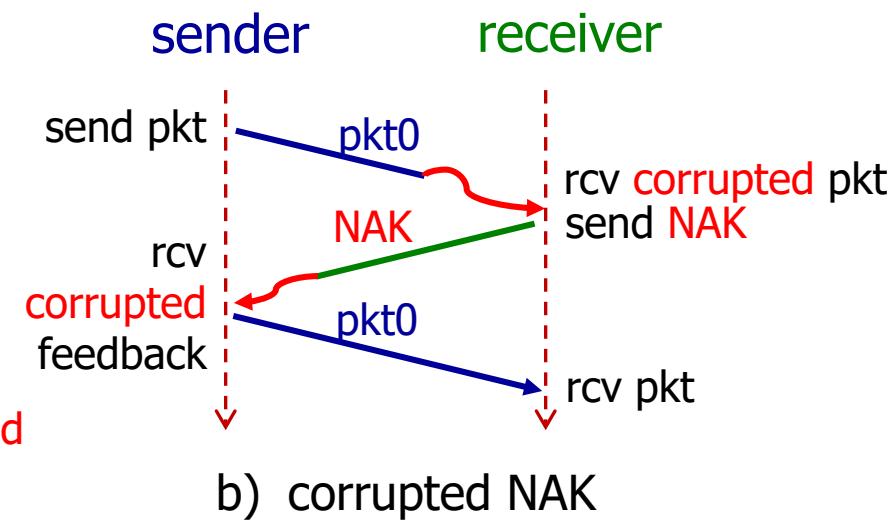
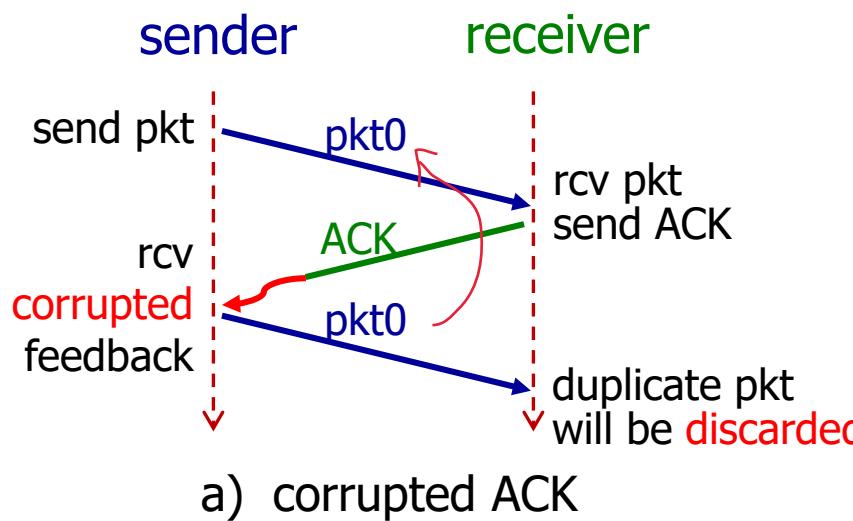


rdt 2.1: rdt 2.0 + Packet Seq.

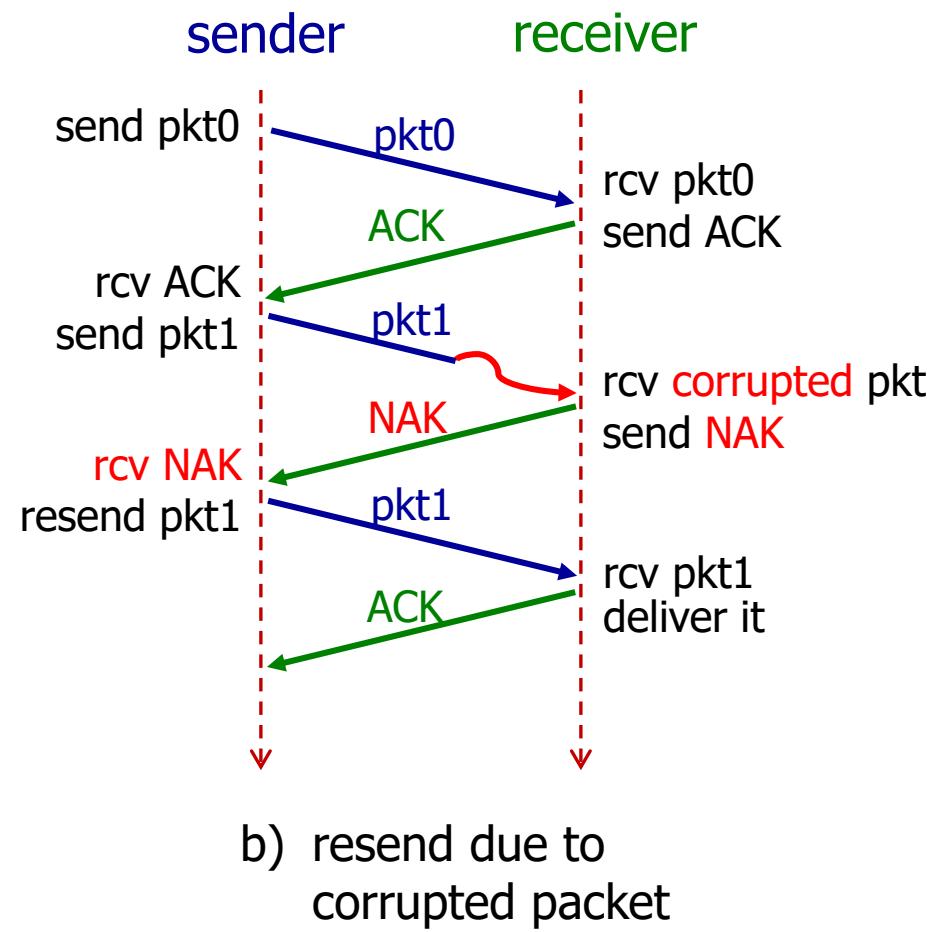
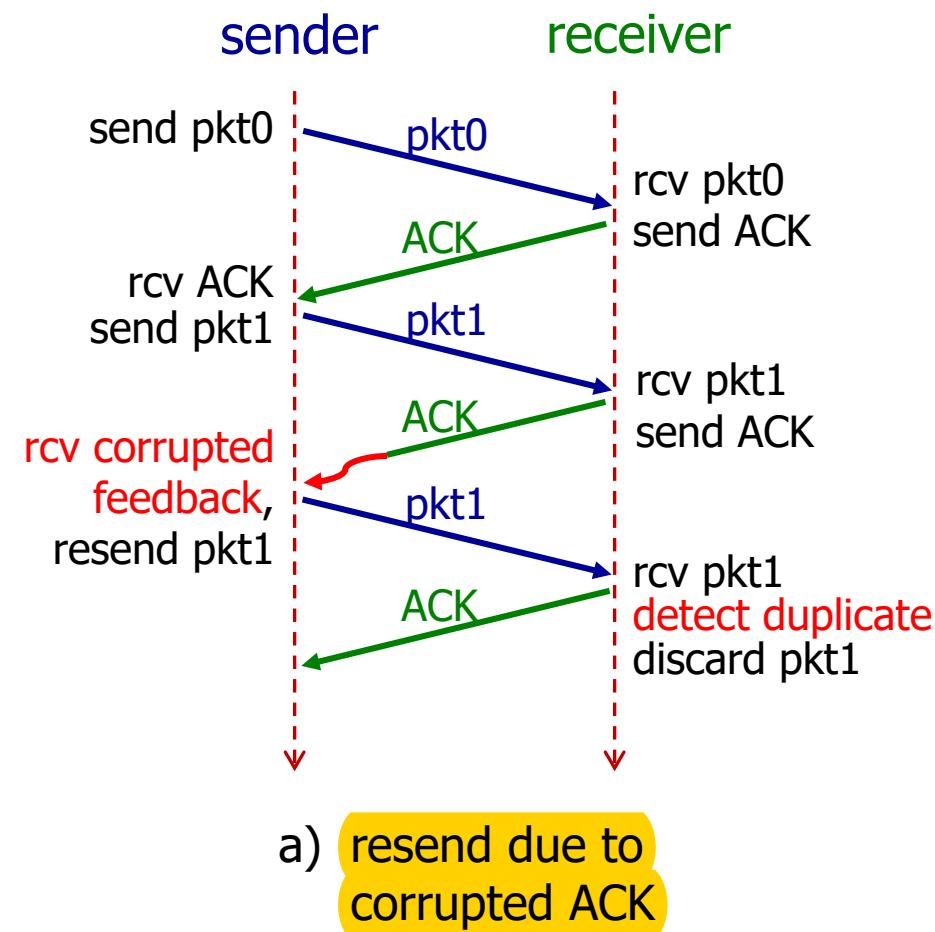
- ❖ To handle duplicates:

- Sender retransmits current packet if ACK/NAK is garbled.
- Sender adds *sequence number* to each packet.
- Receiver discards (doesn't deliver up) duplicate packet.

- ❖ This gives rise to protocol rdt 2.1.

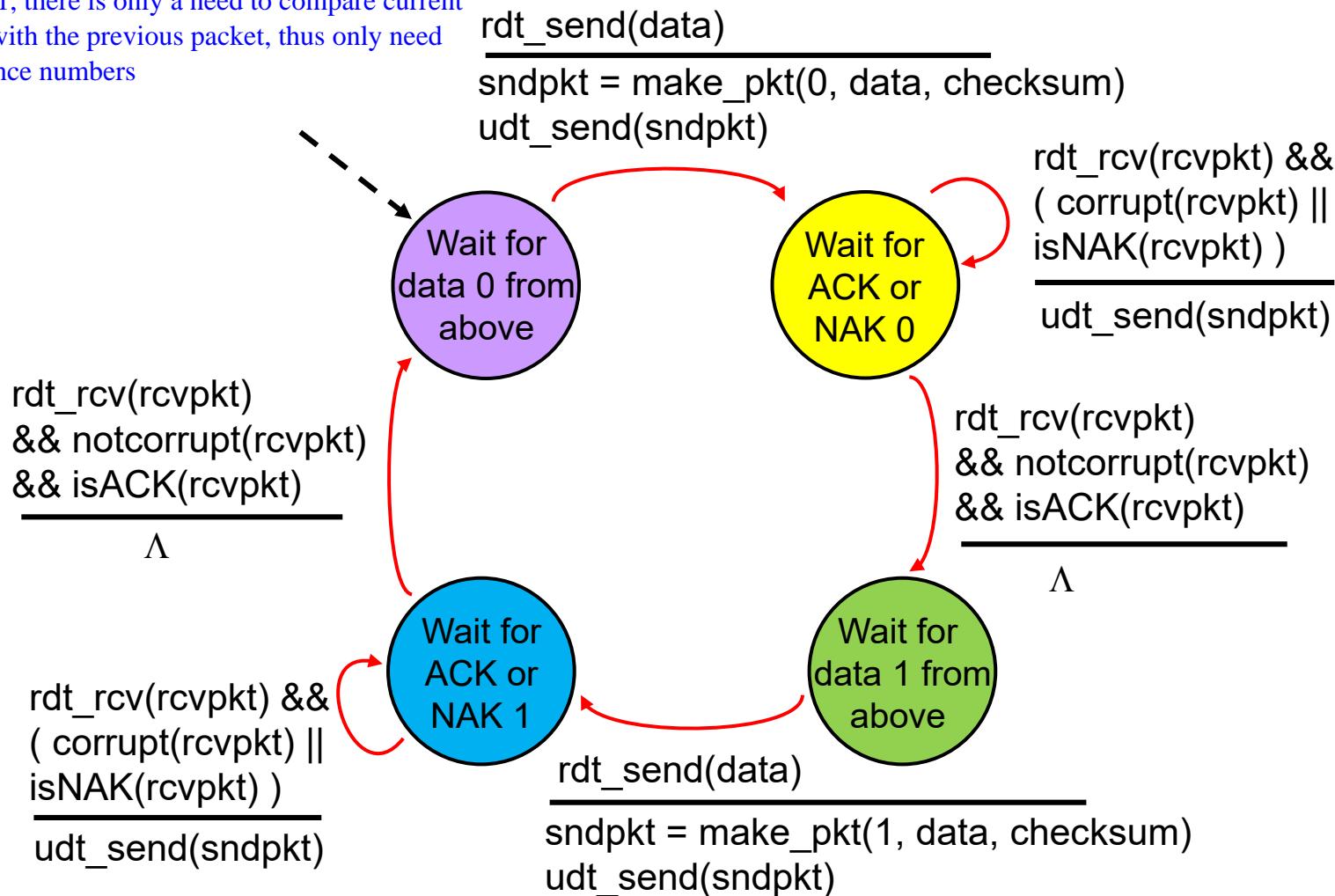


rdt 2.1 In Action

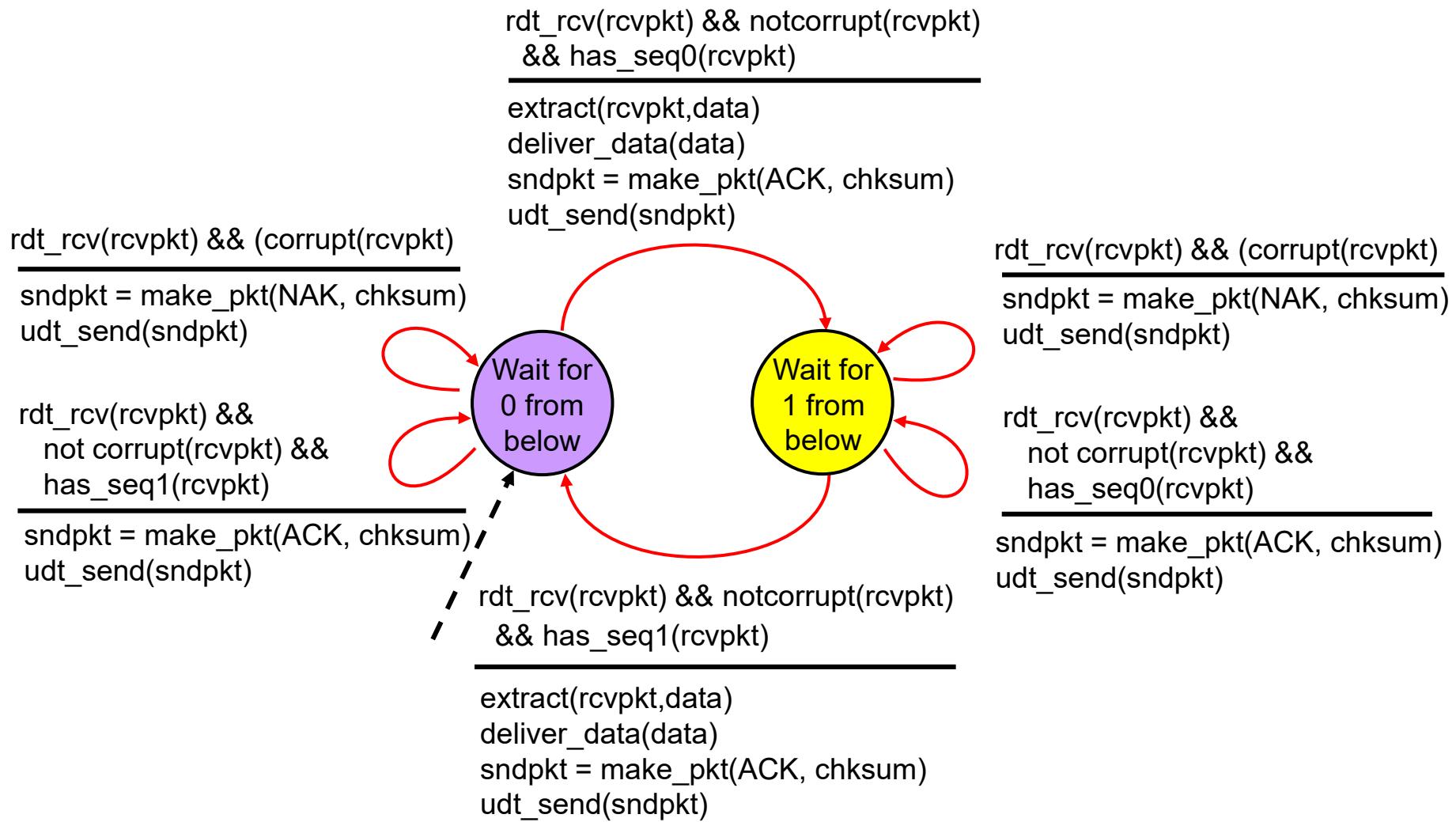


rdt 2.1 Sender FSM

 in rdt 2.1, there is only a need to compare current packet with the previous packet, thus only need 2 sequence numbers



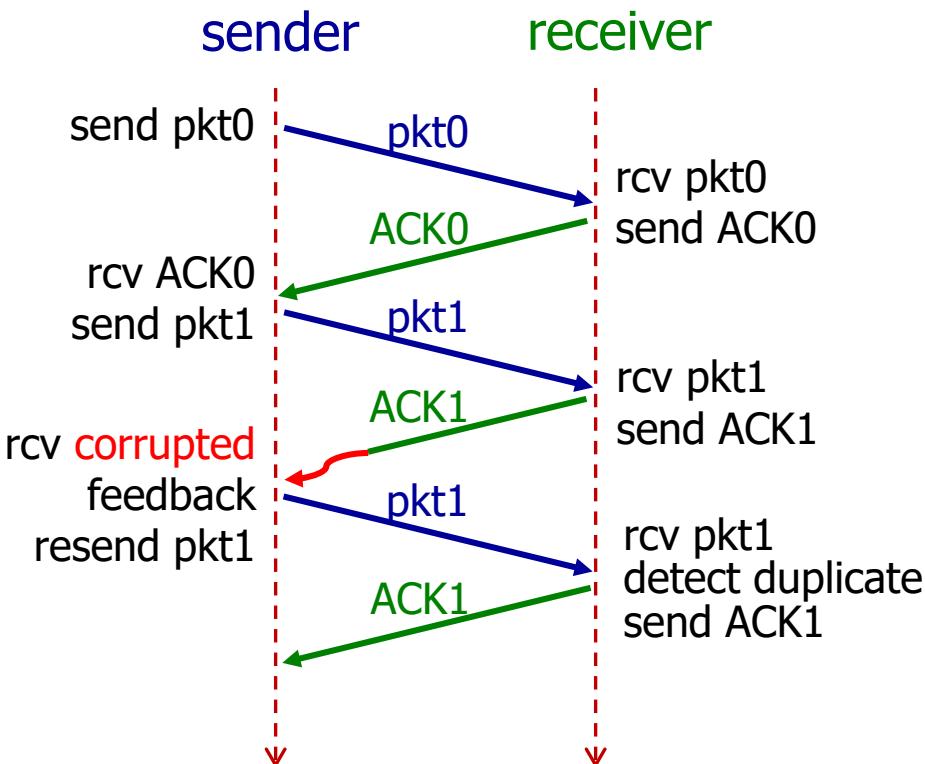
rdt 2.1 Receiver FSM



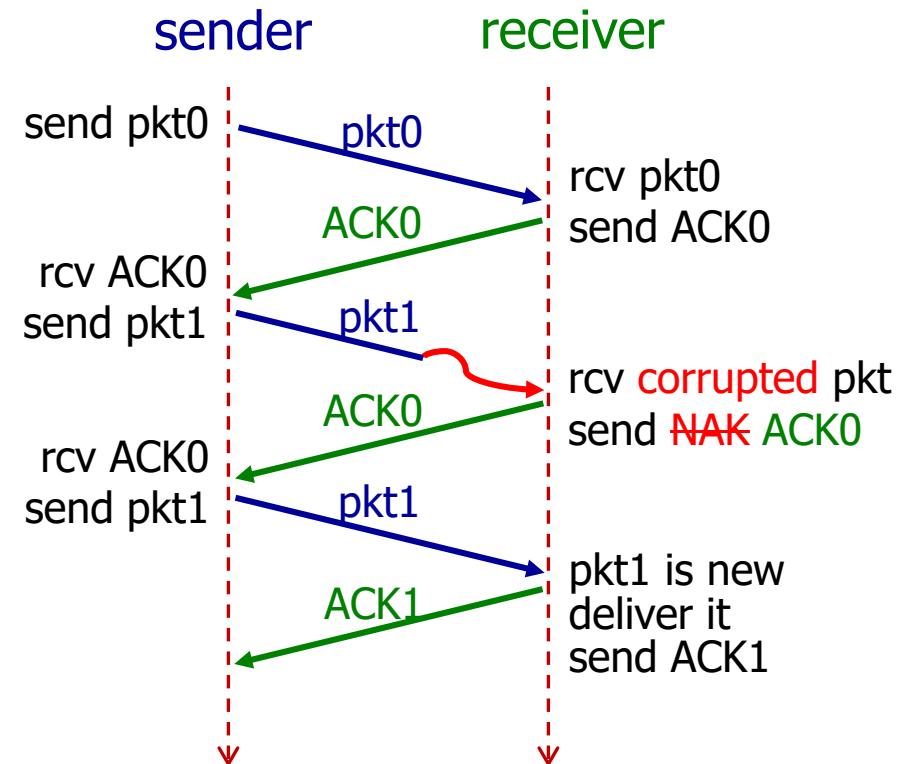
rdt 2.2: a NAK-free Protocol

- ❖ Same assumption and functionality as rdt 2.1, but use ACKs only.
- ❖  Instead of sending NAK, receiver sends ACK for the last packet received OK.
 - Now receiver must *explicitly* include seq. # of the packet being ACKed.
- ❖ Duplicate ACKs at sender results in same action as NAK: *retransmit current pkt.*

rdt 2.2 In Action



a) resend due to
corrupted ACK



b) resend due to
duplicate ACK

rdt 3.0: Channel with *Errors* and *Loss*

- ❖ Assumption: underlying channel
 - may flip bits in packets
 - may lose packets
 - may incur arbitrarily long packet delay
 - but won't re-order packets

- ❖ Question: how to detect packet loss?
 - checksum, ACKs, seq. #, retransmissions will be of help... but not enough

rdt 3.0: Channel with *Errors* and *Loss*

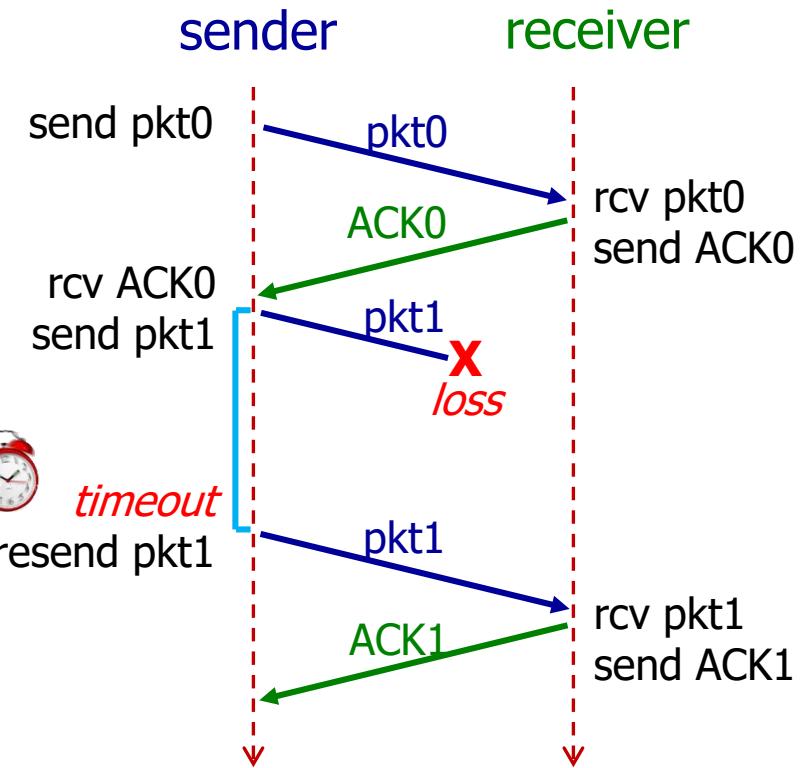
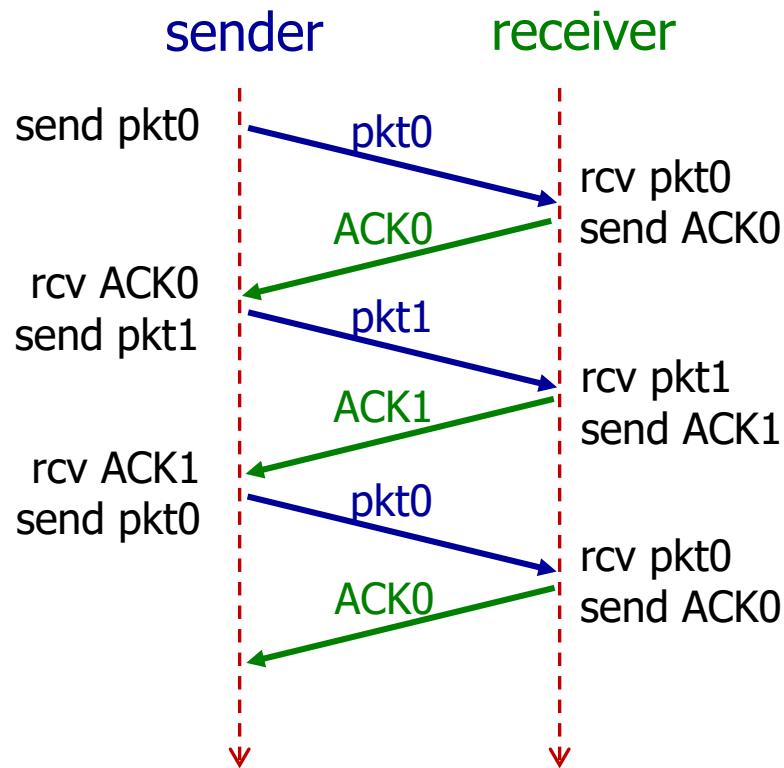
- ❖ To handle packet loss:

- Sender waits “reasonable” amount of time for ACK.
- Sender retransmits if no ACK is received till *timeout*.

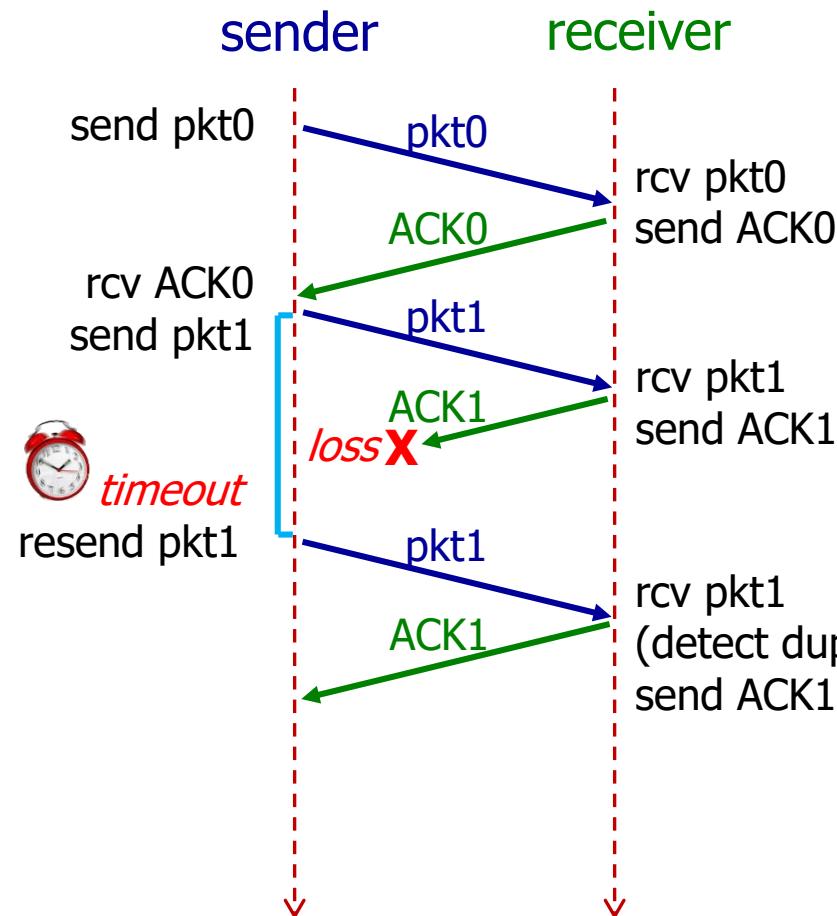
- ❖ Question: what if packet (or ACK) is just delayed, but not lost?

- Timeout will trigger retransmission.
- Retransmission will generate duplicates in this case, but receiver may use seq. # to detect it.
- Receiver must specify seq. # of the packet being ACKed (check scenario (d) two pages later).

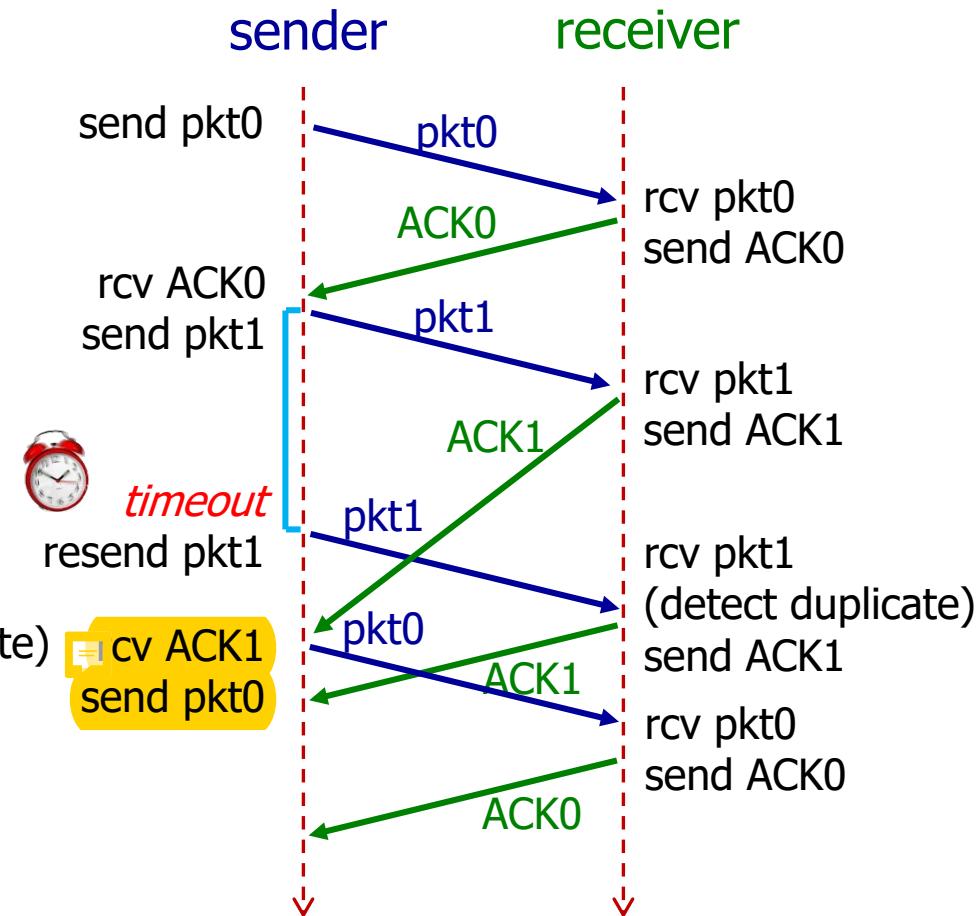
rdt 3.0 In Action



rdt 3.0 In Action

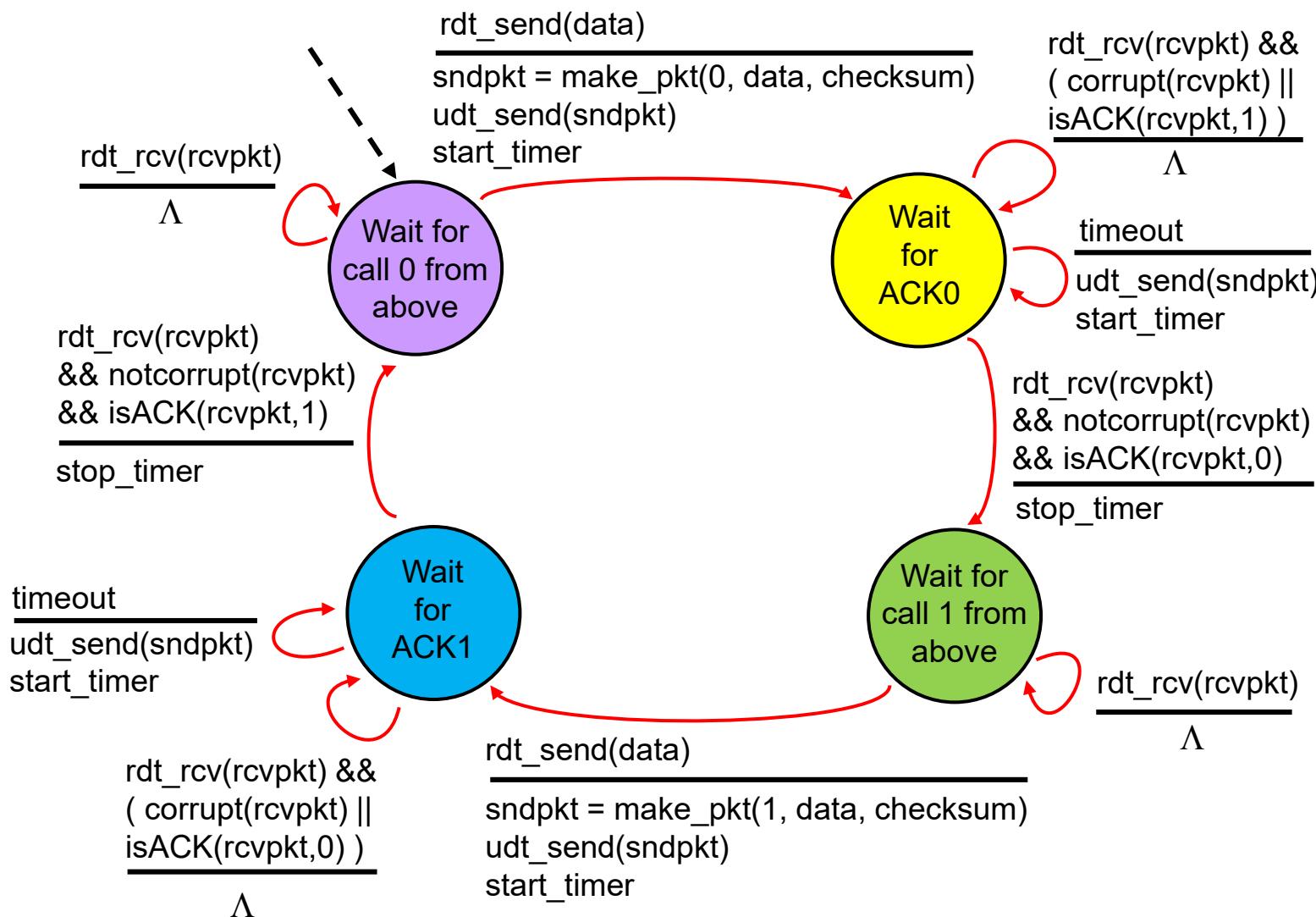


c) lost ACK



d) premature timeout / delayed ACK

rdt 3.0 Sender FSM



RDT Summary

rdt Version	Scenario	Features Used
1.0	no error	nothing
2.0	data Bit Error	checksum, ACK/NAK
2.1	data Bit Error ACK/NAK Bit Error	checksum, ACK/NAK, sequence Number
2.2	Same as 2.1	NAK free
3.0	data Bit Error ACK/NAK Bit Error packet Loss	checksum, ACK, sequence Number, timeout/re-transmission

Performance of rdt 3.0

- ❖ rdt 3.0 works, but performance stinks.
- ❖ Example: packet size = 8000 bits, link rate = 1 Gbps:

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 0.008 \text{ msec}$$

- If RTT = 30 msec, sender sends 8000 bits every 30.008 msec.

throughput

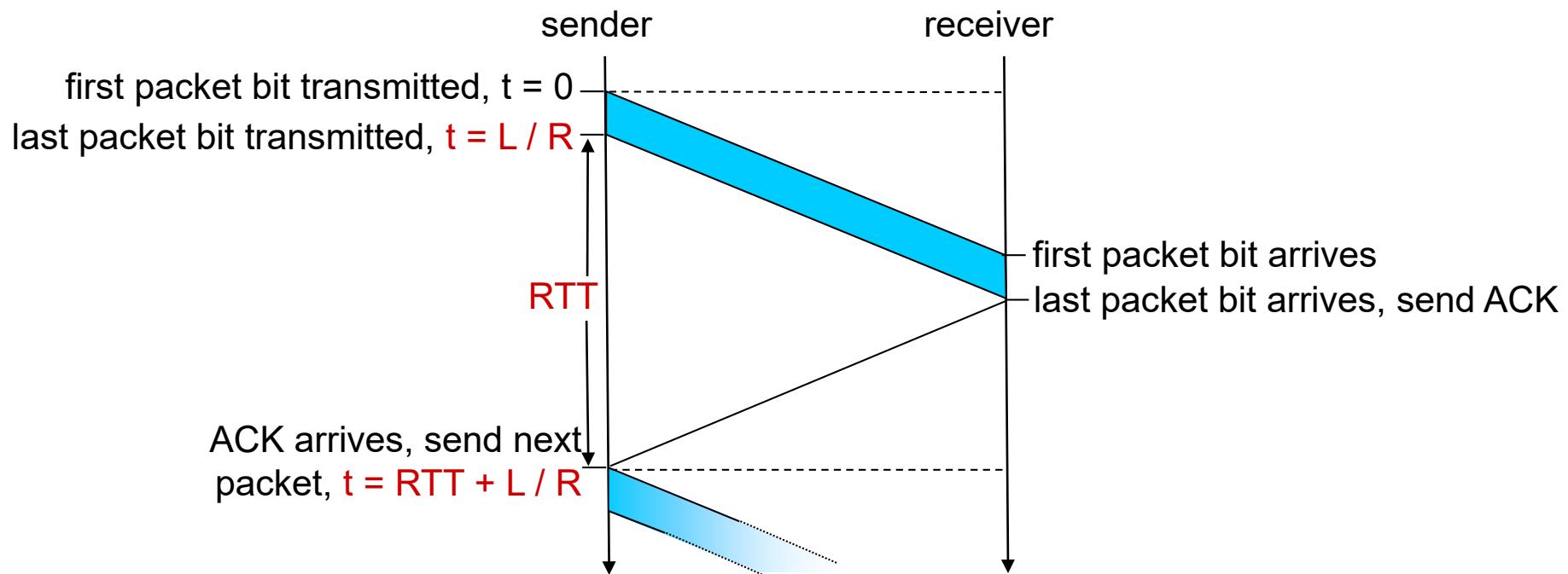
$$\text{throughput} = \frac{L}{RTT + d_{trans}} = \frac{8000}{30.008} = 267 \text{ kbps}$$

- U_{sender} : *utilization* – fraction of time sender is busy sending

$$U_{sender} = \frac{d_{trans}}{RTT + d_{trans}} = \frac{0.008}{30 + 0.008} = 0.00027$$

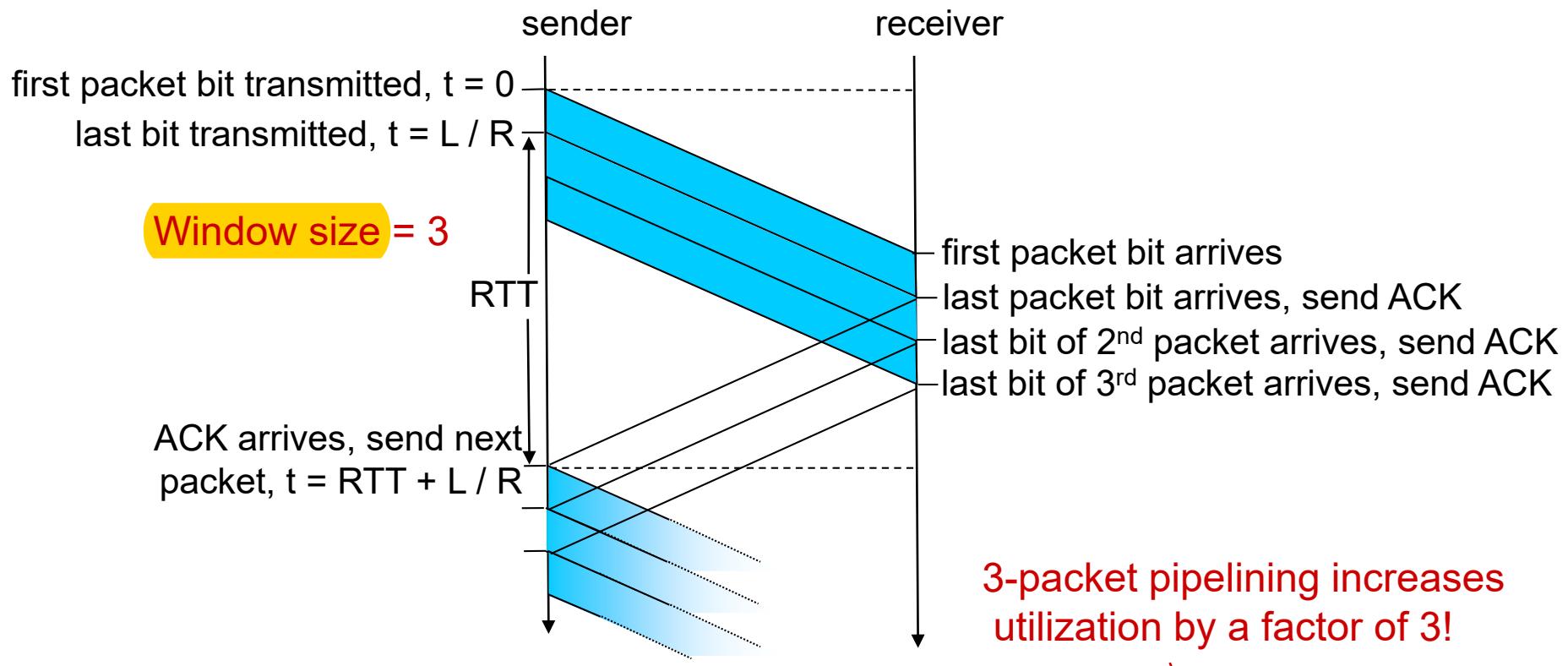
rdt 3.0: Stop-and-wait Operation

- ❖ Network protocol limits use of physical resources!



$$U_{\text{sender}} = \frac{L / R}{RTT + L/R} = \frac{0.008}{30 + 0.008} = 0.00027$$

Pipelining: Increased Utilization

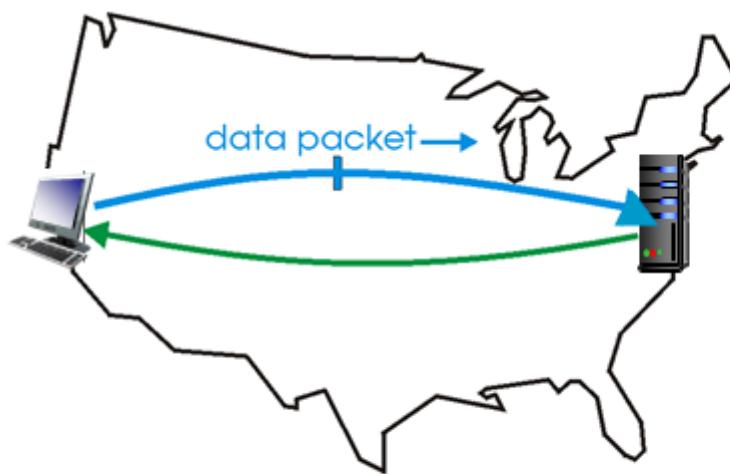


$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L/R} = \frac{0.024}{30 + 0.008} = 0.00081$$

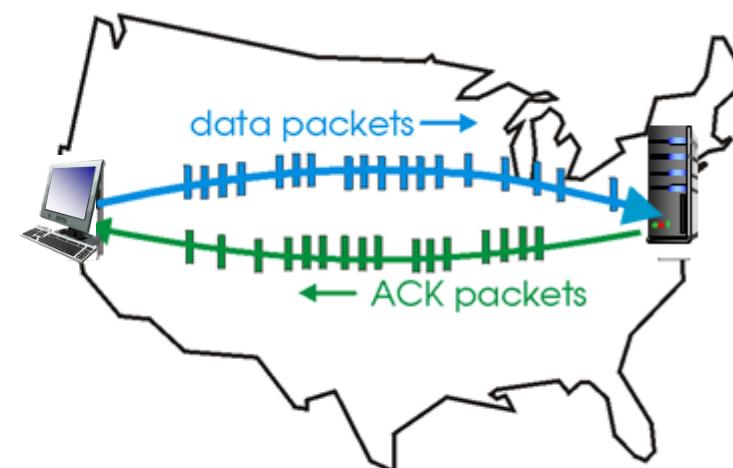
Pipelined Protocols

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



(b) a pipelined protocol in operation

Benchmark Pipelined Protocols

- ❖ Two generic forms of pipelined protocols:
 - *Go-Back-N (GBN)*
 - *Selective repeat (SR)*
- ❖ Assumption (same as rdt 3.0): underlying channel
 - may flip bits in packets
 - may lose packets
 - may incur arbitrarily long packet delay
 - but won't re-order packets

Go-back-N In Action

sender window (N=4)

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

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

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

rcv ACK0, send pkt4
rcv ACK1, send pkt5

ignore duplicate ACK

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



pkt 2 timeout

(re)send pkt2
(re)send pkt3
(re)send pkt4
(re)send pkt5

receiver

receive pkt0, send ACK0
receive pkt1, send ACK1
receive pkt3, **discard**,
(re)send ACK1

receive pkt4, **discard**,
(re)send ACK1
receive pkt5, **discard**,
(re)send ACK1

rcv pkt2, deliver, send ACK2
rcv pkt3, deliver, send ACK3
rcv pkt4, deliver, send ACK4
rcv pkt5, deliver, send ACK5

Go-back-N: Key Features

❖ GBN Sender

- can have up to N unACKed packets in pipeline.
- insert k-bits sequence number in packet header.
- use a “sliding window” to keep track of unACKed packets.
- keep a timer for the oldest unACKed packet.
- $\text{timeout}(n)$: retransmit packet n and all subsequent packets in the window.

❖ GBN Receiver

- only ACK packets that arrive in order.
 - simple receiver: need only remember `expectedSeqNum`
- discard out-of-order packets and ACK the last in-order seq. #.
 - *Cumulative ACK*: “ACK m ” means all packets up to m are received.

Go-back-N In Action

sender window ($N=4$)

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

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

receiver

receive pkt0, send ACK0
receive pkt1, send ACK1
receive pkt2, send ACK2
receive pkt3, send ACK3

cumulative ACK

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

rcv ACK2, send pkt4

loss X
loss X
loss X

send pkt5
send pkt6
(wait)

receive pkt4, send ACK4
receive pkt5, send ACK5
receive pkt6, send ACK6



pkt 3 timeout

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

(re)send pkt3
(re)send pkt4
(re)send pkt5
(re)send pkt6
(wait)

receive pkt3, discard,
send ACK6

Go-back-N In Action

sender window ($N=6$)

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

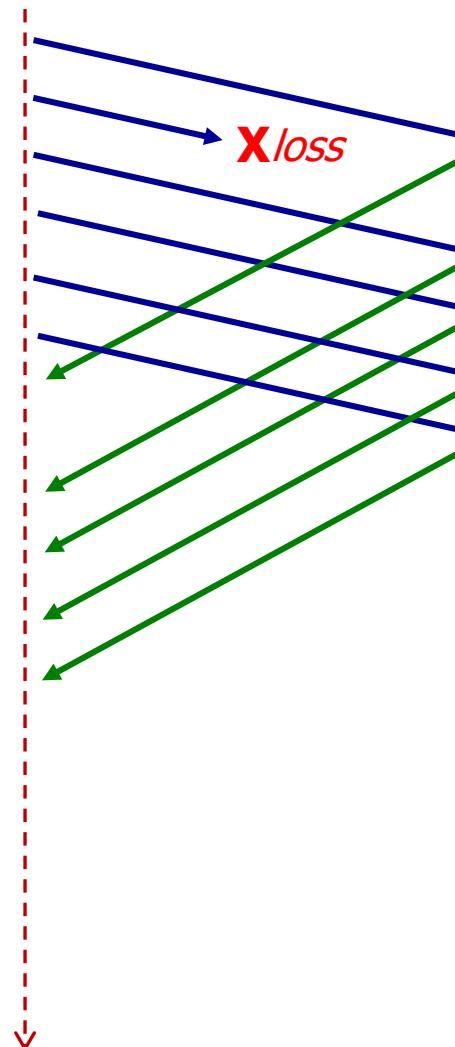
0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

sender

send pkt0
send pkt1
send pkt2
send pkt3
send pkt4
send pkt5
(wait)

receiver

receive pkt0, send ACK0
receive pkt2, **discard**
send ACK0
receive pkt3, **discard**,
send ACK0
receive pkt4, **discard**,
send ACK0
receive pkt5, **discard**,
send ACK0



Selective Repeat: Key Features

- ❖ Receiver *individually acknowledges* all correctly received packets.
 - Buffers out-of-order packets, as needed, for eventual in-order delivery to upper layer.
- ❖ Sender maintains timer for **each** unACKed packet.
 - When timer expires, retransmit only that unACKed packet.

More efficient, but much more complicated

Sender: need to have multiple timers

Receiver: needs to keep a buffer of all the received packets

Selective Repeat In Action

sender window (N=4)

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

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

rcv ACK3



pkt 2 timeout
(re)send pkt2

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

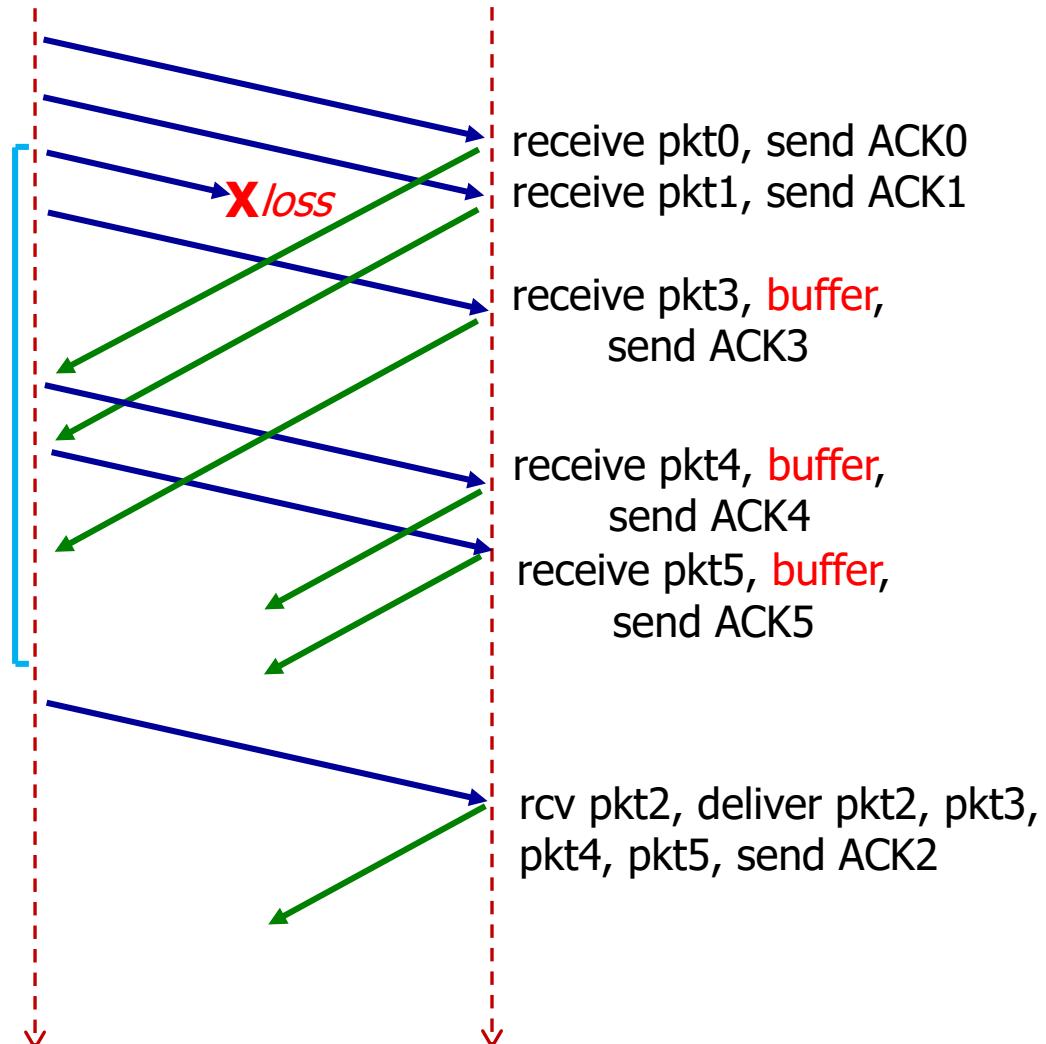
receiver

receive pkt0, send ACK0
receive pkt1, send ACK1

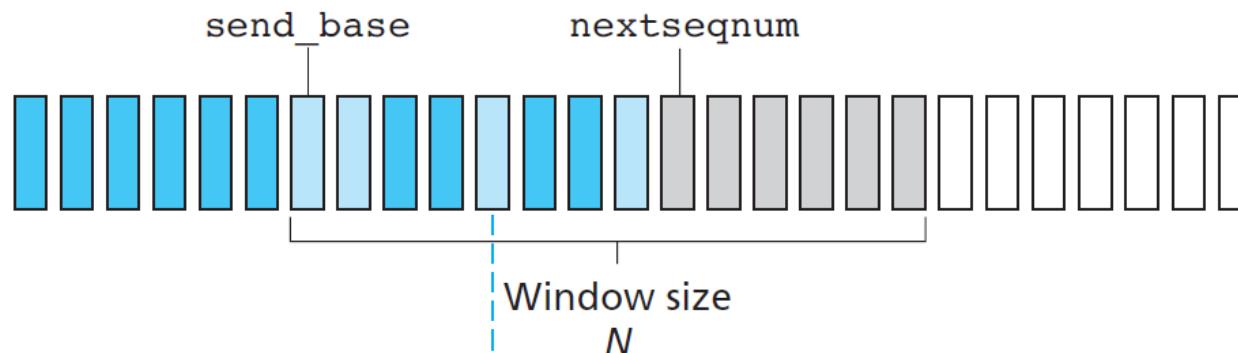
receive pkt3, **buffer**,
send ACK3

receive pkt4, **buffer**,
send ACK4
receive pkt5, **buffer**,
send ACK5

rcv pkt2, deliver pkt2, pkt3,
pkt4, pkt5, send ACK2



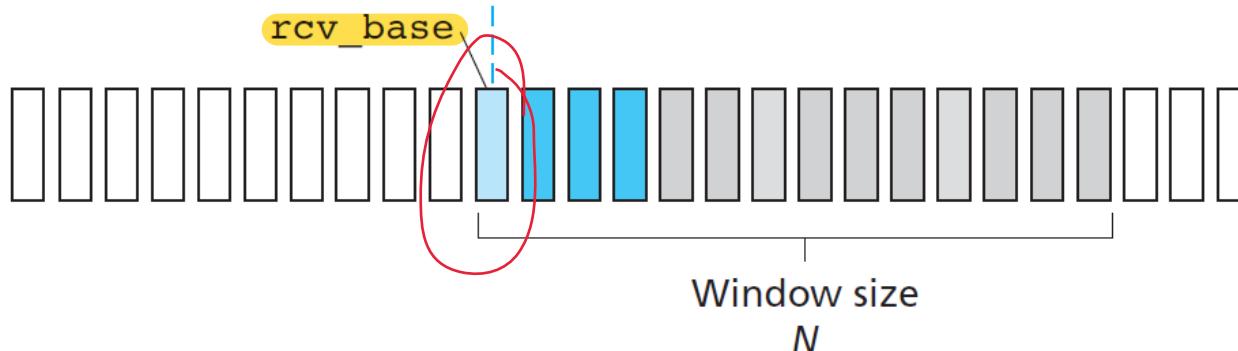
SR Sender and Receiver Windows



Key:

	Already ACK'd		Usable, not yet sent
	Sent, not yet ACK'd		Not usable

a. Sender view of sequence numbers



Key:

	Out of order (buffered) but already ACK'd		Acceptable (within window)
	Expected, not yet received		Not usable

b. Receiver view of sequence numbers

Selective Repeat: Behaviors

sender

data from above:

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

timeout(n):

- ❖ resend pkt n , restart timer

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

- ❖ mark pkt n as received
- ❖ if n is smallest unACKed pkt, advance window base to next unACKed seq. #

receiver

pkt n in $[rcvbase, rcvbase+N-1]$

- ❖ send ACK(n)
- ❖ out-of-order: buffer
- ❖ in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in $[rcvbase-N, rcvbase-1]$

- ❖ ACK(n)

otherwise:

- ❖ ignore

Lectures 4&5: Roadmap

3.1 Transport-layer Services

3.2 Multiplexing and De-multiplexing

3.3 Connectionless Transport: UDP

3.4 Principles of Reliable Data Transfer

3.5 Connection-oriented transport: TCP

TCP: Transport Control Protocol

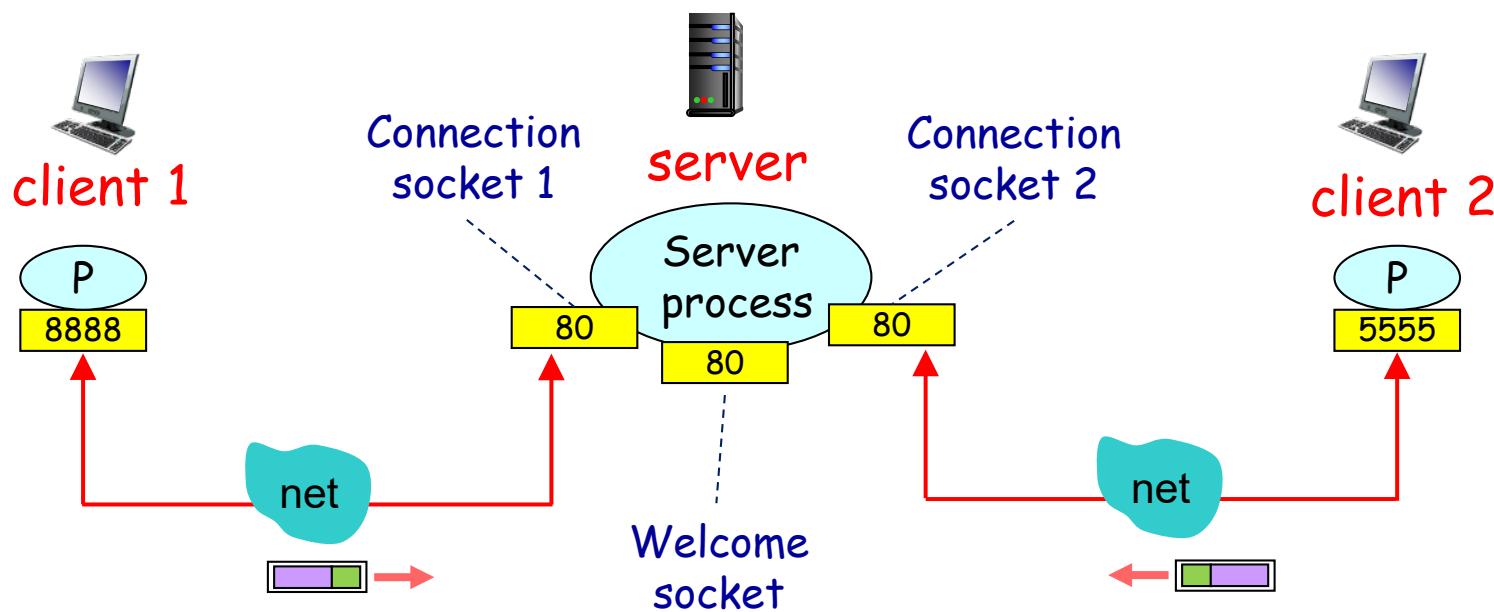
- ❖ In contrast to UDP, TCP is complex and is described in tens of RFCs, with new mechanisms or tweaks introduced throughout the years, resulting in many variants of TCP.
- ❖ We will only scratch the surface of TCP in CS2105.
 - More will be covered in CS3103.

TCP Overview [RFC 793, 1122, ... 2581 ...]

- ❖ **Point-to-point:**
 - One sender, one receiver.
- ❖ **Connection-oriented:**
 - handshaking (exchange of control messages) before sending app data.
- ❖ **Full duplex service:**
 - bi-directional data flow in the same connection
- ❖ **Reliable, in-order *byte stream*:**
 - use sequence numbers to label bytes

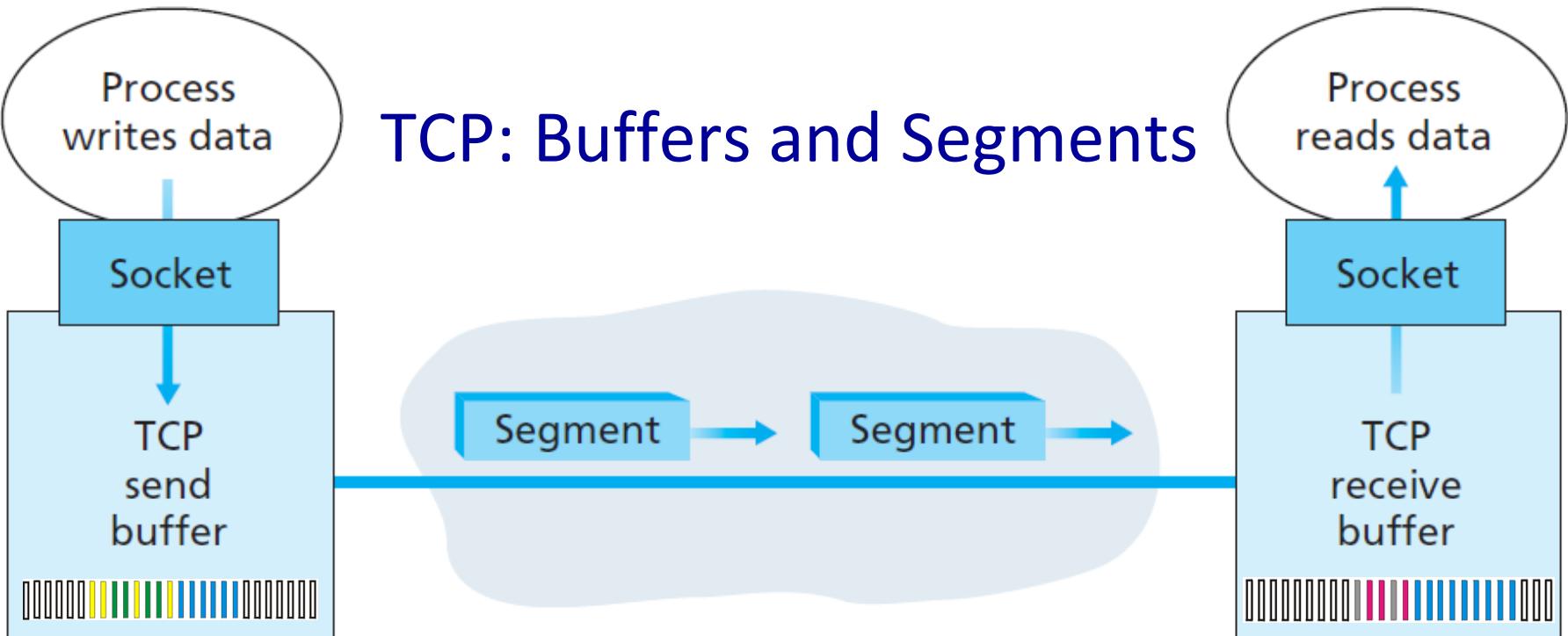
Connection-oriented De-mux

- ❖ A TCP connection (socket) is identified by 4-tuple:
 - (`srcIPAddr, srcPort, destIPAddr, destPort`)
 - Receiver uses all four values to direct a segment to the appropriate socket.



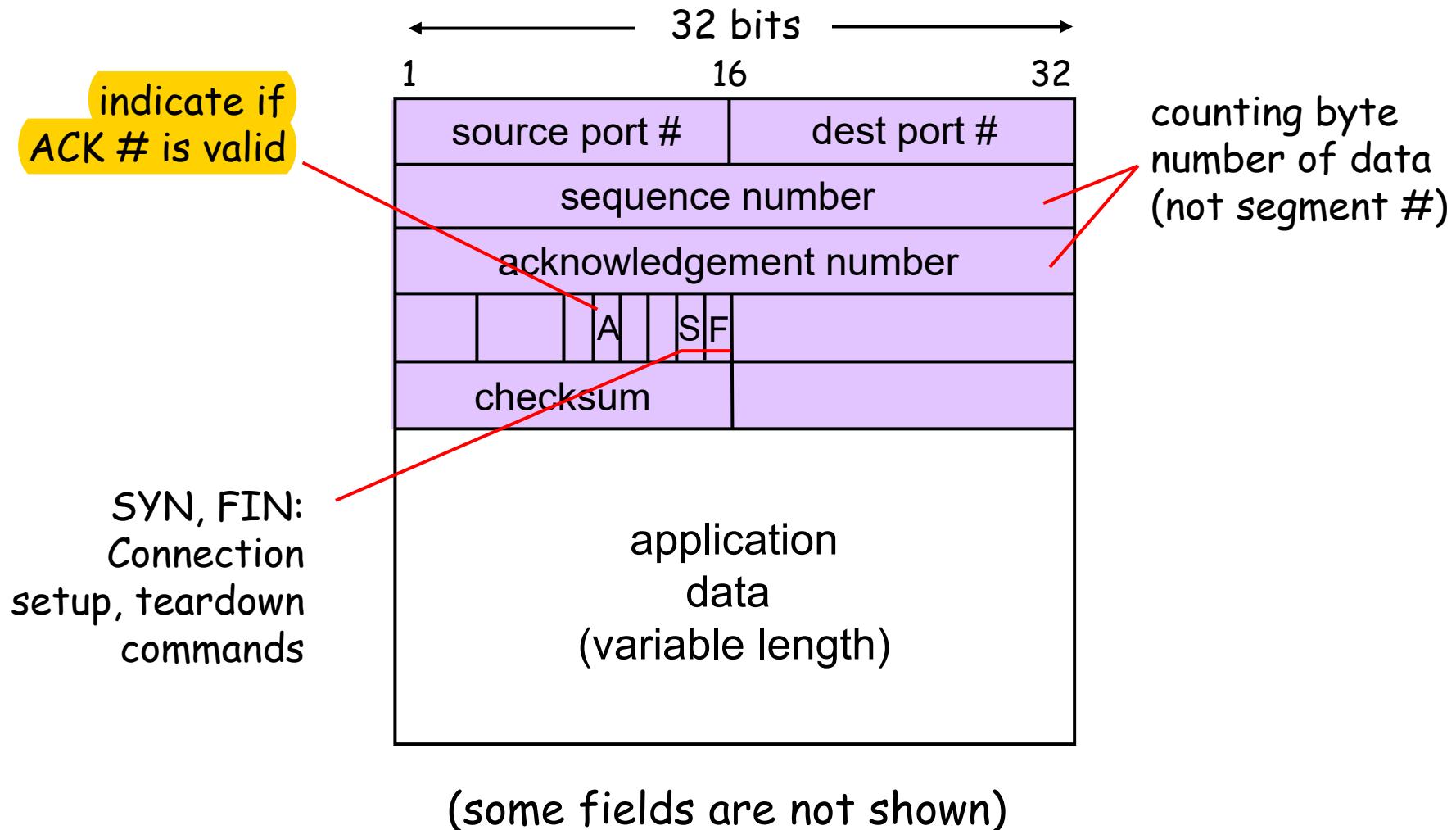
from client IP, client port 8888 to server IP, server port 80 > Client socket 1
from client IP2, client port 5555 to server IP, server port 80 > Client socket 2

TCP: Buffers and Segments



- ❖ TCP send and receive buffers
 - two buffers created after handshaking at any side.
- ❖ How much app-layer data a TCP segment can carry?
 - **maximum segment size (MSS)**, typically 1,460 bytes
 - app passes data to TCP and TCP forms packets in view of MSS.

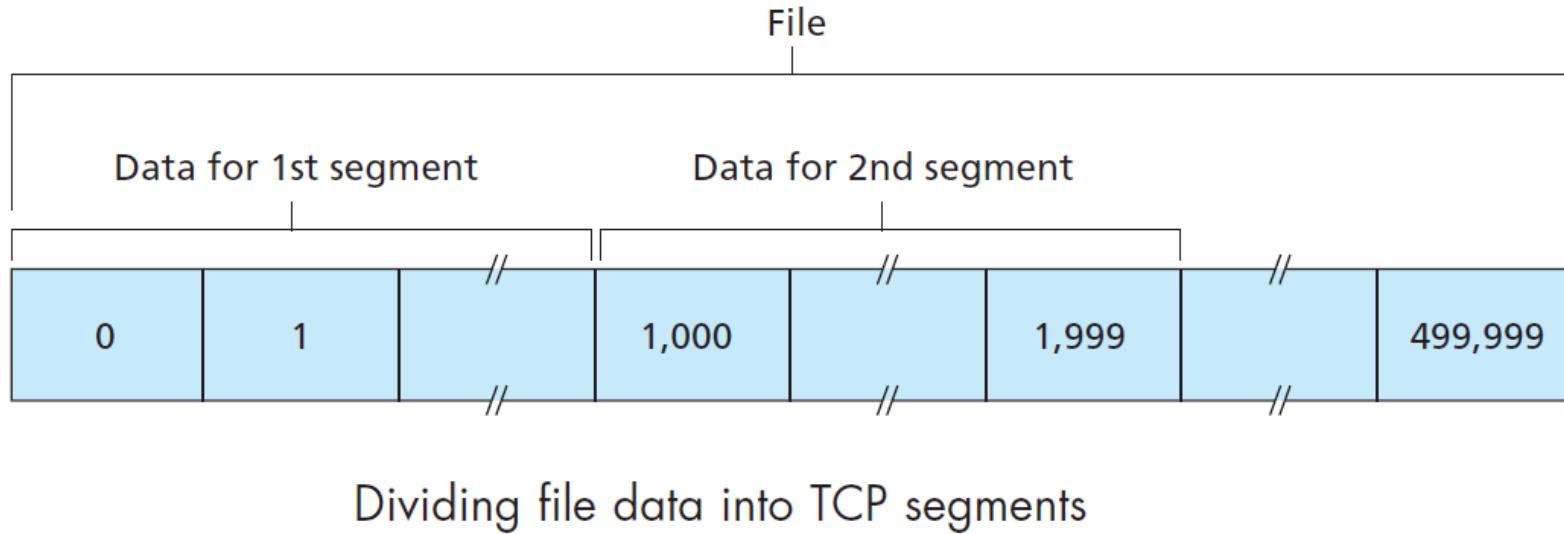
TCP Header



source port #	dest port #
sequence number	
ACK number	
checksum	

TCP Sequence Number

- ❖ “Byte number” of the first byte of data in a segment.
- ❖ Example: send a file of 500,000 bytes; MSS is 1,000 bytes.



- ❖ Seq. # of 1st TCP segment: 0, 2nd TCP segment: 1,000, 3rd TCP segment: 2,000, 4th TCP segment: 3,000, etc.

TCP ACK Number

source port #	dest port #
sequence number	
	ACK number
	A
checksum	

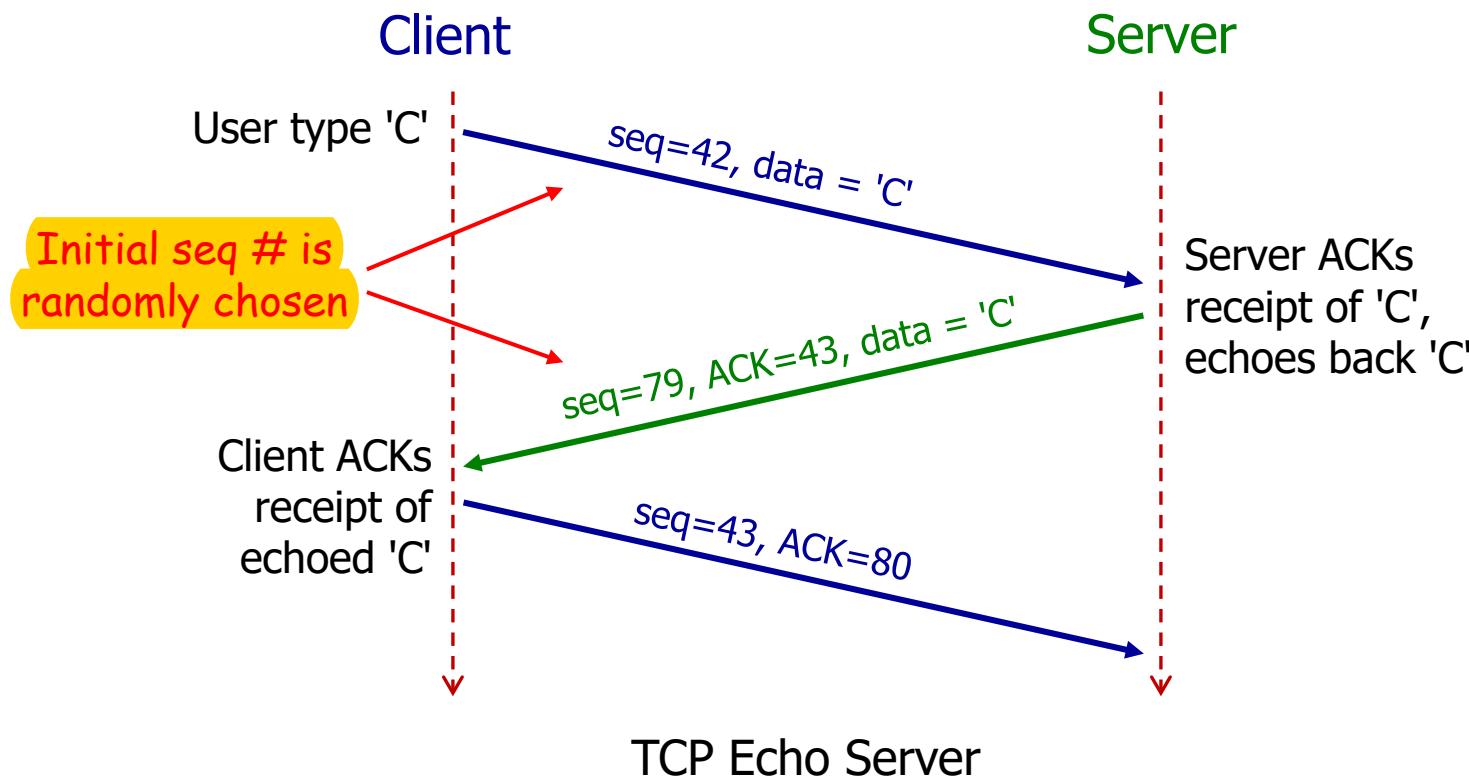
- ❖ Seq # of the next byte of data expected by receiver.

Sequence number of a segment	Amount of data carried	Corresponding ACK number
0 – 999	1,000	1,000
1,000 – 1999	1,000	2,000
2,000	1,000	3,000
3,000	1,000	4,000
...

- ❖ TCP ACKs up to the first missing byte in the stream (Cumulative ACK).
 - Note: TCP spec doesn't say how receiver should handle out-of-order segments - it's up to implementer.

Example: TCP Echo Server

- ❖ TCP (and also UDP) is a full duplex protocol
 - bi-directional data flow in the same TCP connection.
- ❖ Example:



TCP Sender Events (simplified)

```
NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch(event)

        event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length(data)
            break;

        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged segments)
                    start timer
            }
            break;

    } /* end of loop forever */

```

first byte of data to be ACKed.

event: data received from application above

create TCP segment with sequence number NextSeqNum

if (timer currently not running)

start timer

pass segment to IP

NextSeqNum=NextSeqNum+length(data)

break;

event: timer timeout

retransmit not-yet-acknowledged segment with

smallest sequence number

start timer

break;

event: ACK received, with ACK field value of y

if (y > SendBase) {

SendBase=y

if (there are currently any not-yet-acknowledged segments)

start timer

}

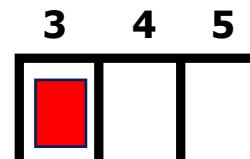
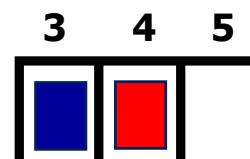
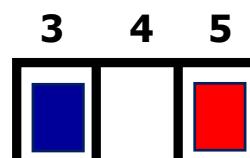
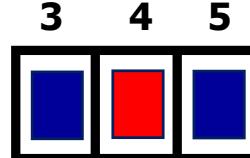
break;

Sender keeps one timer only

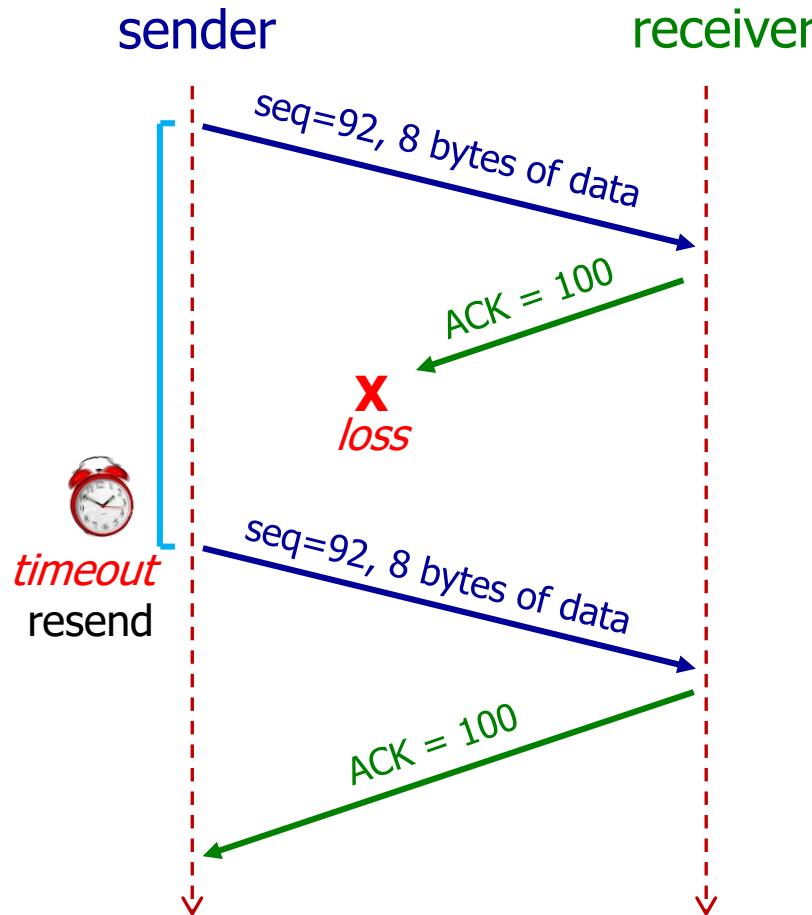
Retransmit only oldest unACKed packet

Cumulative ACK

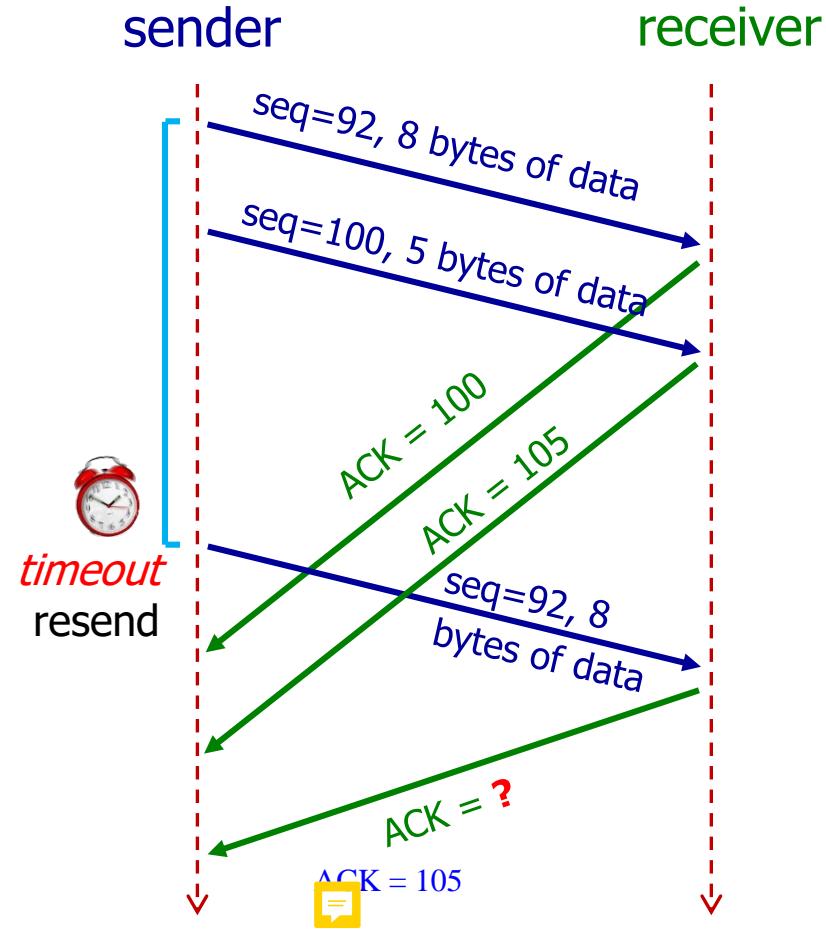
TCP ACK Generation [RFC 2581]

<i>Event at TCP 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 duplicate ACK , indicating seq. # of next expected byte 
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap 

TCP Timeout / Retransmission



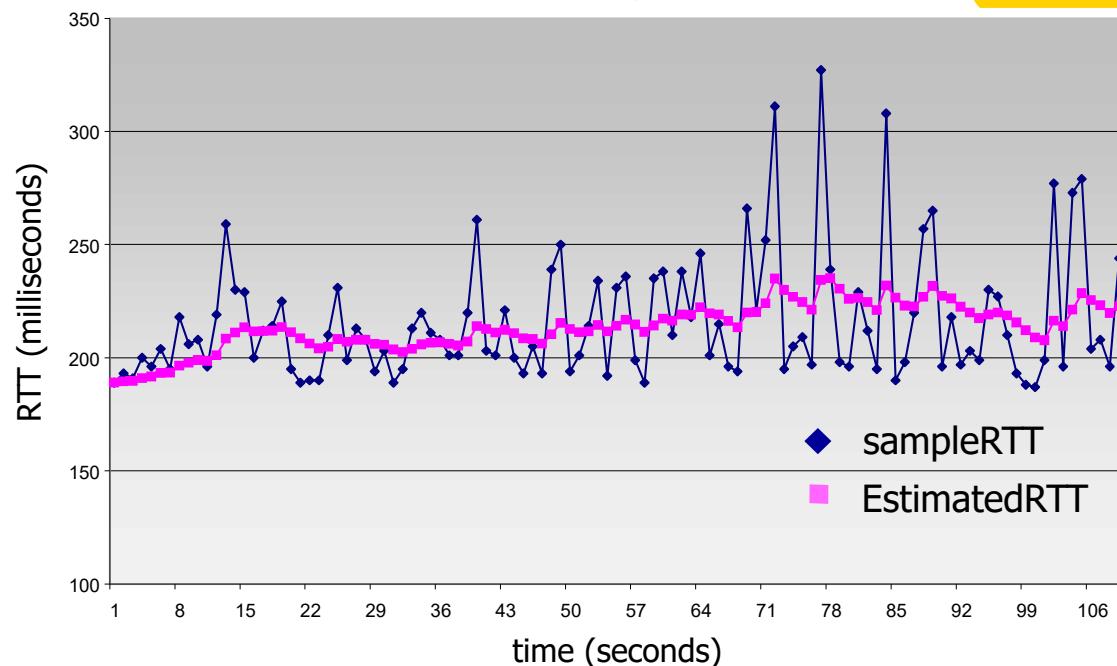
a) Lost ACK



b) premature timeout

TCP Timeout Value

- ❖ How does TCP set appropriate timeout value?
 - **too short timeout**: premature timeout and unnecessary retransmissions.
 - **too long timeout**: slow reaction to segment loss.
 - Timeout interval must be longer than RTT – **but RTT varies!**



TCP Timeout Value

- ❖ TCP computes (and keeps updating) timeout interval based on estimated RTT.

EstimatedRTT = (1 - α) * EstimatedRTT + α * SampleRTT

(typical value of α : 0.125)

DevRTT = (1 - β) * DevRTT + β * | SampleRTT - EstimatedRTT |

(typical value of β : 0.25)

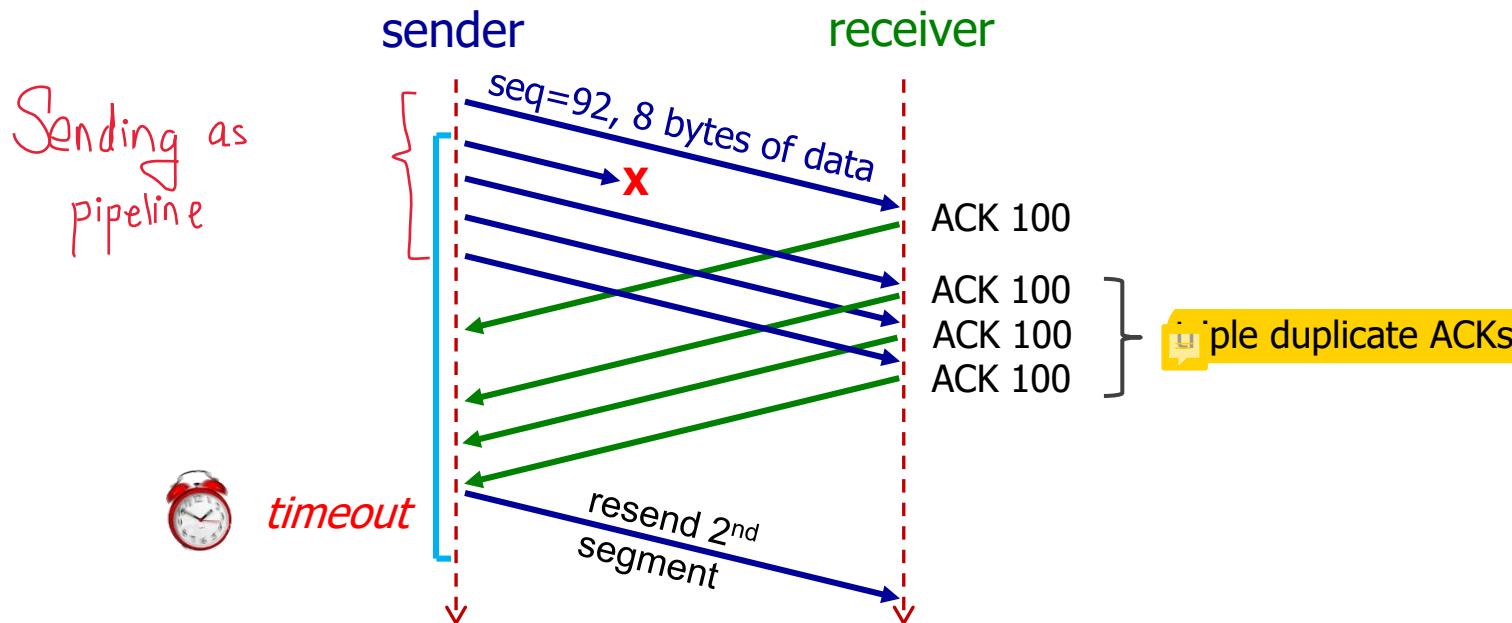
TimeoutInterval = EstimatedRTT + 4 * DevRTT



↑
“safety margin”

TCP Fast Retransmission [RFC 2001]

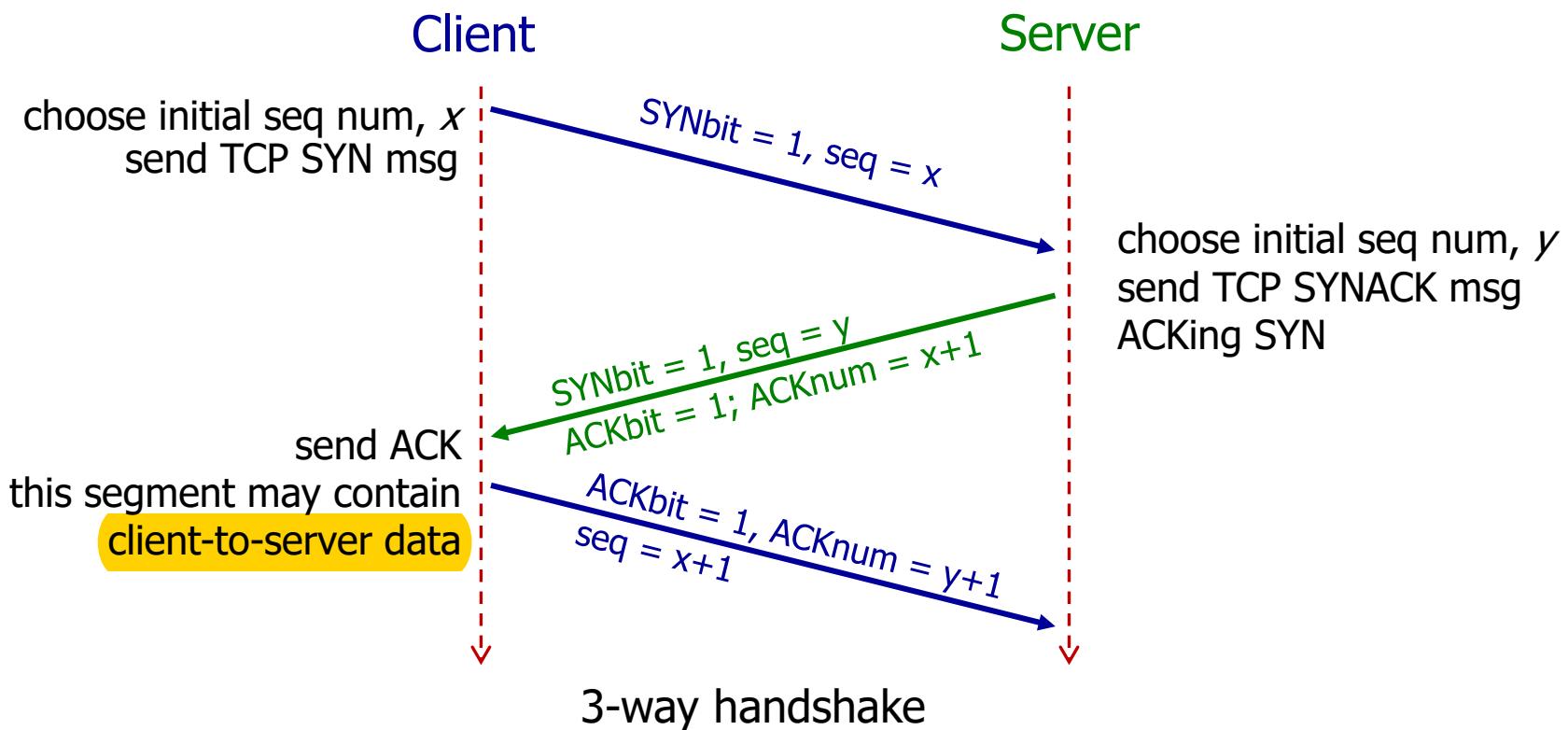
- ❖ Timeout period is often relatively long.
 - long delay before resending lost packet
- ❖ **Fast retransmission:**
 - **Event:** If sender receives **4 ACKs** for the same segment, it supposes that segment is lost.
 - **Action:** resend segment (even before timer expires).



Establishing Connection

source port #	dest port #
sequence number	
ACK number	
A	S
checksum	

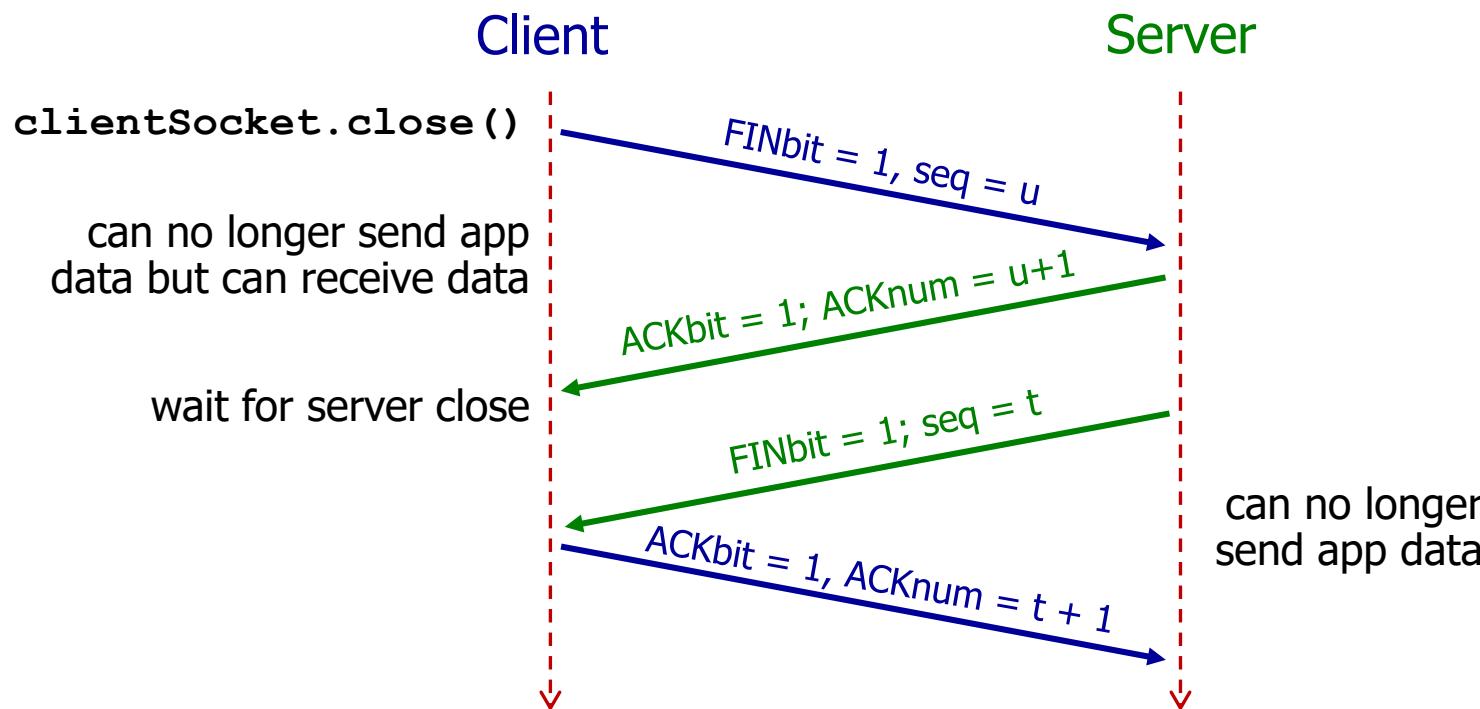
- ❖ Before exchanging app data, TCP sender and receiver “shake hands”.
 - Agree on connection and exchange connection parameters.



Closing Connection

source port #	dest port #
sequence number	
ACK number	
A	F
checksum	

- ❖ Client, server each close their side of connection.
 - send TCP segment with FIN bit = 1



What we did not cover....

- ❖ TCP flow control (Chapter 3.5.5)
 - Sender won't overflow receiver's buffer by sending too much or too fast.
 - Receiver feeds back to sender how many more bytes it is willing to accept.
- ❖ TCP congestion control (Chapter 3.6 & 3.7)
 - Be polite and send less if network is congested.
- ❖ They will be covered in the next course (CS3103)

Lectures 4&5: Summary

Go-back-N

- ❖ Sender can have up to N unACKed packets in pipeline
- ❖ Receiver only sends *cumulative ACKs*
 - Out-of-order packets discarded
- ❖ Sender sets timer for the oldest unACKed packet
 - when timer expires, retransmit *all* unACKed packets

Selective Repeat

- ❖ Sender can have up to N unACKed packets in pipeline
- ❖ Receiver sends *individual ACK* for each packet
 - Out-of-order packets buffered
- ❖ Sender maintains timer for *each* unACKed packet
 - when timer expires, retransmit only that unACKed packet

Lectures 4&5: Summary

- ❖ Connection-oriented transport: TCP
 - Segment structure
 - Reliable data transfer
 - Sequence number
 - Acknowledgement number
 - Cumulative ACK
 - Setting and updating retransmission time interval
 - Fast retransmission
 - 3-way handshake

CS2105

An Awesome Introduction to Computer Networks

Lectures 6&7: The Network Layer



Department of Computer Science
School of Computing

Lectures 6&7: The Network Layer

After this class, you are expected to understand:

- ❖ the basic services network layer provides.
- ❖ the purpose of DHCP and how it works.
- ❖ IP address, subnet, subnet mask and address allocation.
- ❖ how longest prefix forwarding in a router works.
- ❖ the purpose of routing protocols on the Internet.
- ❖ the principle of Bellman-Ford equation.
- ❖ the workings of distance vector algorithm.
- ❖ the purpose of NAT and how it works.
- ❖ the Internet Protocol (IP) and how datagram fragmentation works.

Lectures 6&7: Roadmap

4.1 Overview of Network Layer

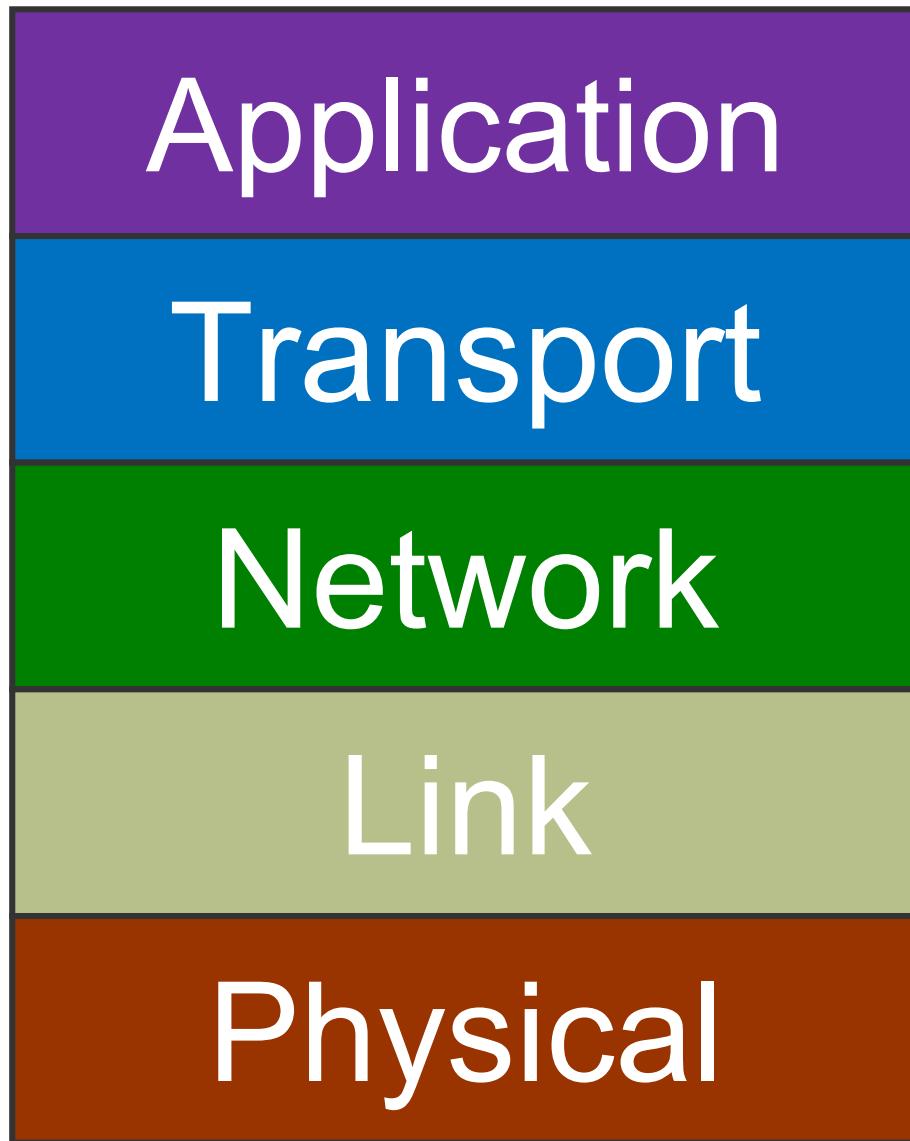
4.2 What's Inside a Router

4.3 The Internet Protocol (IP)

5.2 Routing Algorithms

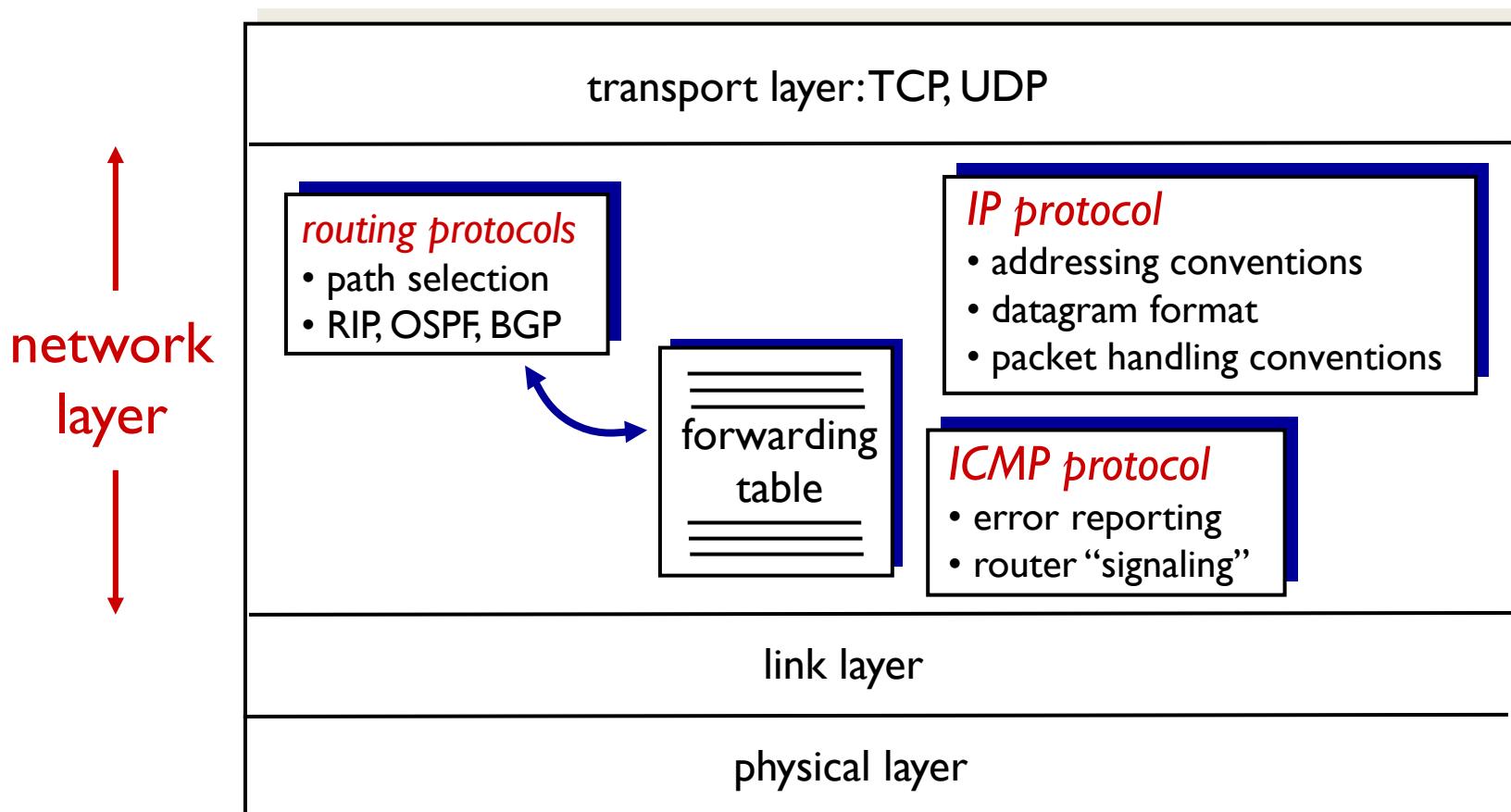
5.6 ICMP

Kurose Textbook, Chapters 4&5
(Some slides are taken from the book)



Network Layer Services

- ❖ Network layer delivers packets to receiving hosts.
 - Routers examine header fields of IP datagrams passing it.



Lectures 6&7: Roadmap

4.1 Overview of Network Layer

4.2 What's Inside a Router

- 4.2.1 Destination-Based Forwarding

4.3 The Internet Protocol (IP)

- 4.3.3 IPv4 Addressing

5.2 Routing Algorithms

5.6 ICMP

IP Address

- ❖ IP address is used to identify a host (or a router).
 - A 32-bit integer expressed in either binary or decimal

Binary:	00000001	00000010	00000011	10000001
	_____	_____	_____	_____
Decimal:	1	2	3	129

IP address use dotted decimal notation

- ❖ How does a host get an IP address?
 - manually configured by system administrator, or
 - automatically assigned by a DHCP (Dynamic Host Configuration Protocol) server.

Dynamic Host Configuration Protocol

- ❖ **DHCP** allows a host to dynamically obtain its IP address from DHCP server when it joins network.
 - IP address is **renewable**
 - allow reuse of addresses (only hold address while connected)
 - support mobile users who want to join network.
- ❖ **DHCP**: 4-step process:
 - 1) Host broadcasts “**DHCP discover**” message
 - 2) DHCP server responds with “**DHCP offer**” message
 - 3) Host requests IP address: “**DHCP request**” message
 - 4) DHCP server sends address: “**DHCP ACK**” message

Arriving
client



all the messages will be broadcasted on the network -
since there might be many DHCP servers, thus need to
broadcast which offer is chosen so that the other DHCP
servers can retract their offer and save that IP address



DHCP server
223.1.2.5

broadcast
address

DHCP discover

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

special IP

DHCP offer

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

Your IP
address

DHCP request

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

DHCP ACK

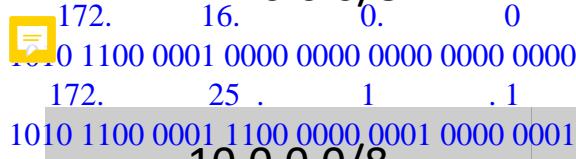
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 655
lifetime: 3600 secs

transaction ID is so that you know which reply from server
is for your request

More on DHCP

- ❖ In addition to host IP address assignment, DHCP may also provide a host additional network information:
 - IP address of first-hop router
 - IP address of local DNS server
 - Network mask (indicating network prefix versus host ID of an IP address)
- ❖ DHCP runs over UDP
 - DHCP server port number: 67
 - DHCP client port number: 68

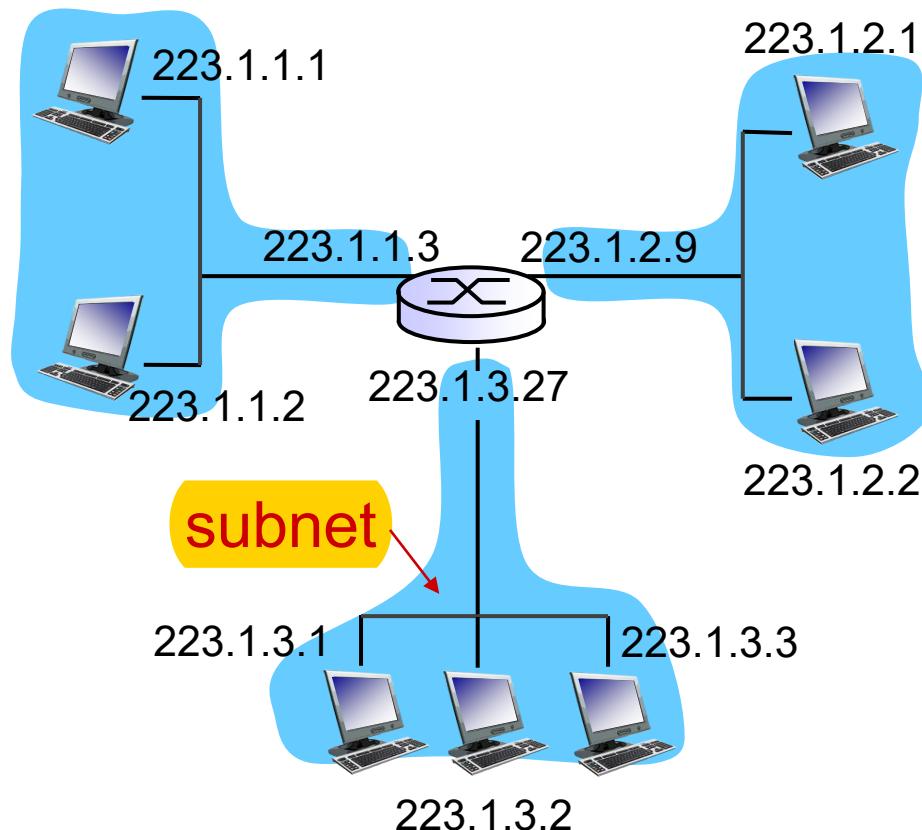
Some Special IP Addresses

Special Addresses	Present Use
0.0.0.0/8	Non-routable meta-address for special use
127.0.0.0/8  1010 1100 0001 0000 0000 0000 0000 172. 16. 0. 0 1010 1100 0001 1100 0000 0001 0000 0001	 Loopback address. A datagram sent to an address within this block loops back inside the host. This is ordinarily implemented using only 127.0.0.1/32.
10.0.0.0/8 172.16.0.0/  192.168.0.0/16	Private addresses, can be used without any coordination with IANA or an Internet registry.
255.255.255.255/32	Broadcast address. All hosts on the same subnet receive a datagram with such a destination address.

The full list of special IP addresses can be found in RFC5735:
<https://tools.ietf.org/rfc/rfc5735.txt>

IP Address and Network Interface

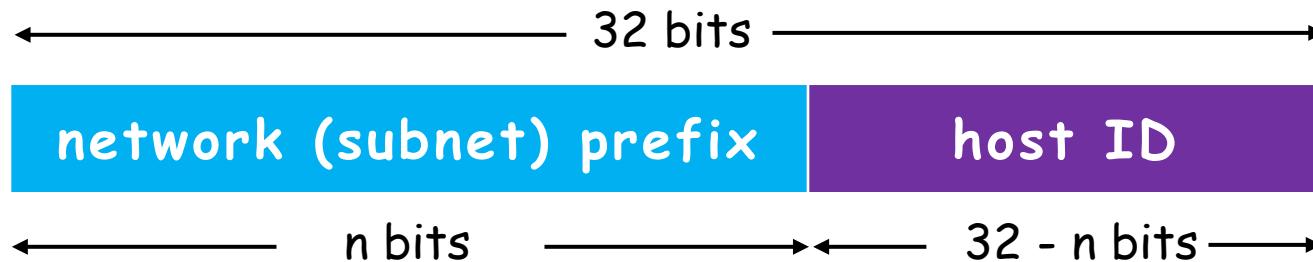
- ❖ An IP address is associated with a network interface.
 - A host usually has one or two network interfaces (e.g. wired Ethernet and WiFi).
 - A router typically has multiple interfaces.



A network consisting of 3 subnets
First 24 bits of IP addr. are network prefix

IP Address and Subnet

- ❖ An IP address logically comprises two parts:



- ❖ **Subnet** is a network formed by a group of “directly” interconnected hosts.
 - Hosts in the same subnet have the same network prefix of IP address.
 - Hosts in the same subnet can physically reach each other without intervening router.  |^{s+} hop router ↴
 - They connect to the outside world through a router.

IP Address: CIDR

- ❖ The Internet's IP address assignment strategy is known as **Classless Inter-domain Routing (CIDR)**.
 - Subnet prefix of IP address is of arbitrary length.
 - Address format: **a.b.c.d/x**, where **x** is the number of bits in subnet prefix of IP address.

←———— subnet prefix —————→ host ID →
11001000 00010111 00010000 00101010

this  Subnet contains 2^9 IP addresses
subnet prefix: 200.23.16.42/23

/23 indicates the no. of bits of subnet prefix

Subnet Mask

- ❖ **Subnet mask** is used to determine which subnet an IP address belongs to.
 - made by setting all subnet prefix bits to "1"s and host ID bits to "0"s.
- ❖ Example: for IP address **200.23.16.42/23**:

	← subnet prefix →				host ID →
IP address in binary	11001000	00010111	00010000	00101010	
Subnet mask	11111111	11111111	11111110	00000000	
Subnet mask in decimal	255.255.254.0				

Quiz

subnet prefix allows only 6 bits for host id - thus it should end with 10

- ❖ For the following 4 IP addresses, which one is in a different subnet from the rest 3?
 - a. 172.26.185.128/26
 - b. 172.26.185.130/26
 - c. 172.26.185.160/26
 - d. 172.26.185.192/26

IP Address Allocation

- ❖ **Q:** How does an organization obtain a block of IP addresses?
- ❖ **A:** Buy from registry or rent from ISP's address space.

	Binary Address	Decimal Address
ISP's block	11001000 00010111 0001 000 0 00000000	200.23.16.0/20
Organization 1	11001000 00010111 0001 000 0 00000000	200.23.16.0/23
Organization 2	11001000 00010111 0001 001 0 00000000	200.23.18.0/23
Organization 3	11001000 00010111 0001 010 0 00000000	200.23.20.0/23
...
Organization 6	11001000 00010111 0001 101 0 00000000	200.23.28.0/23

use 3 more bits to differentiate
6 organizations

Hierarchical Addressing

 Hierarchical addressing allows efficient advertisement of routing information:

Organization 1

200.23.16.0/23

Organization 2

200.23.18.0/23

Organization 3

200.23.20.0/23

⋮

Organization 6

⋮

200.23.28.0/23

ISP 1

"Send me anything
with addresses
beginning with
200.23.16.0/20"

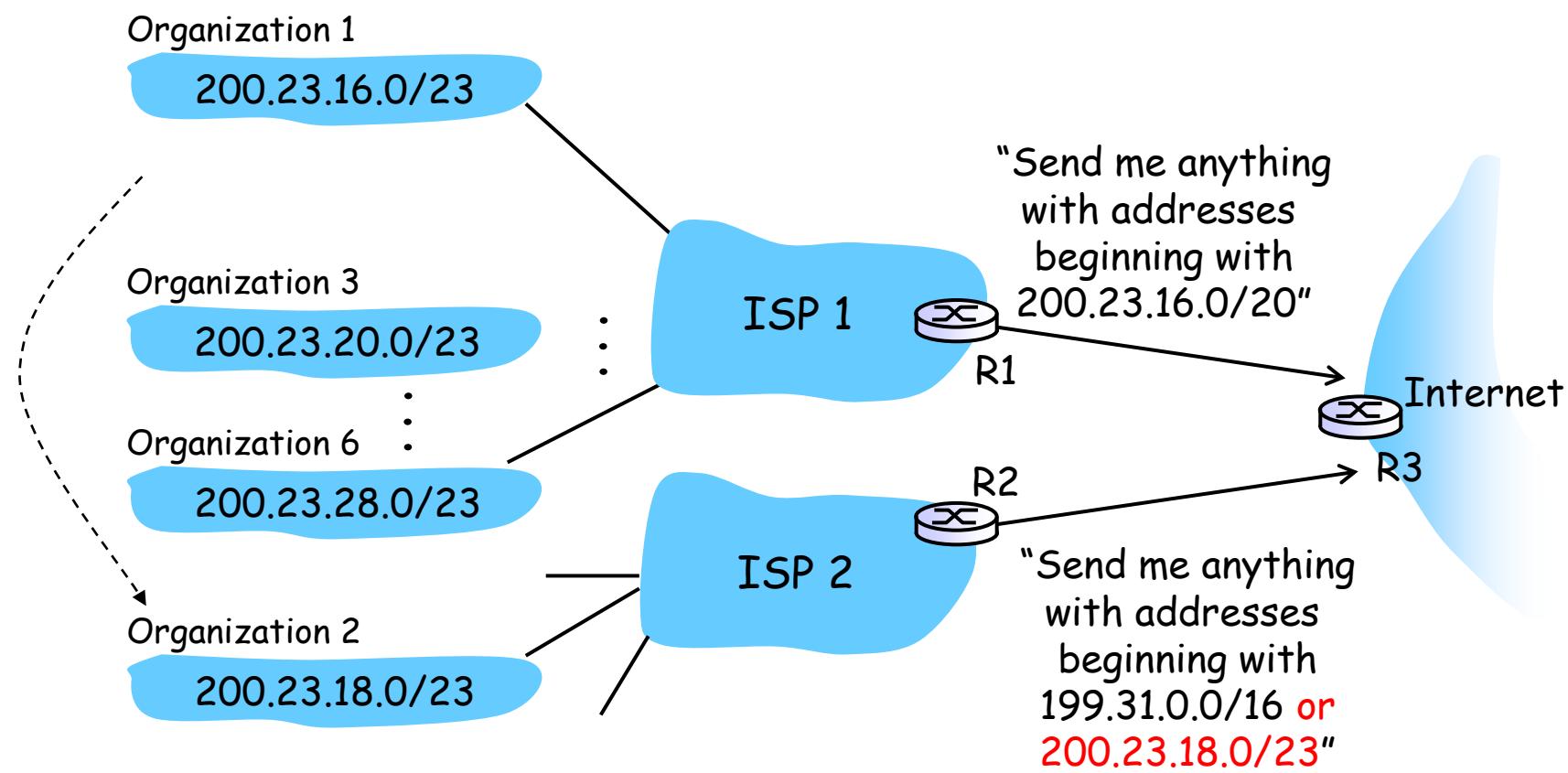
Internet

ISP 2

"Send me anything
with addresses
beginning with
199.31.0.0/16"

Hierarchical Addressing

Suppose Organization 2 now switches to ISP 2, but doesn't want to renumber all of its routers and hosts.

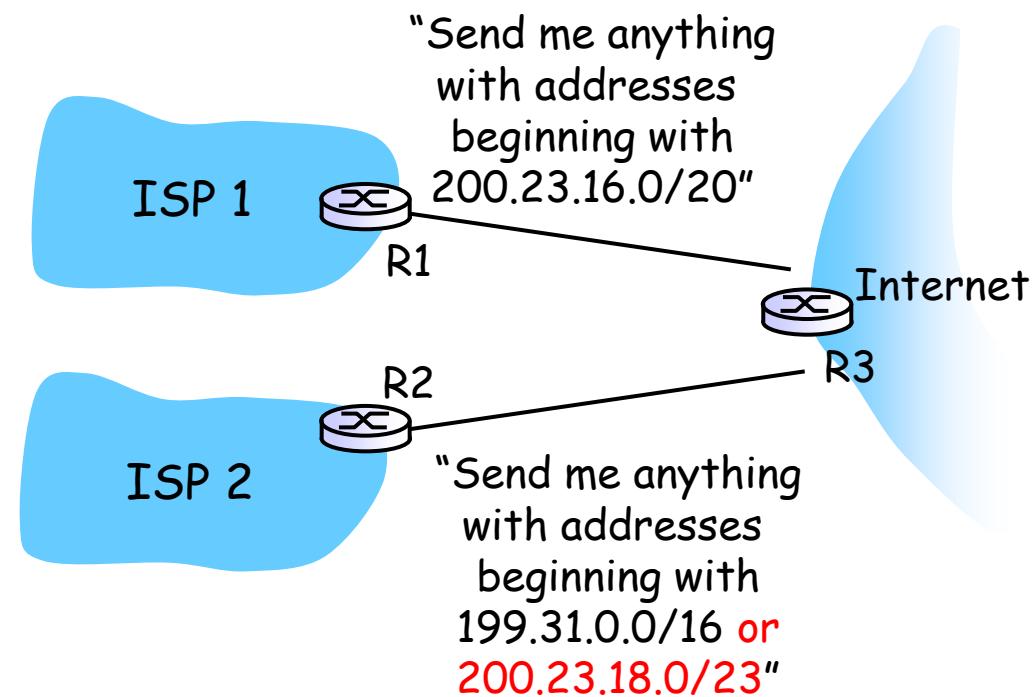


Longest Prefix Match (1/2)

- ❖ **Question:** which router to deliver to,
 - if a packet has destination IP **200.23.20.2**?
 - if a packet has destination IP **200.23.19.3**?

Forwarding Table at R3

Net mask	Next hop
200.23.16.0/20	R1
200.23.18.0/23	R2
199.31.0.0/16	R2
...	...



Longest Prefix Match (2/2)

- ❖ Packet with destination IP 200.23.20.2  R1
 - (Binary: 11001000 00010111 00010100 00000010)
- ❖ Packet with destination IP 200.23.19.3  R2
 - (Binary: 11001000 00010111 00010011 00000011)

Forwarding Table at R3

match the
longest prefix

Net mask	Net mask in binary	Next hop
200.23.16.0/20	11001000 00010111 00010000 00000000	R1
200.23.18.0/23	11001000 00010111 00010010 00000000	R2
199.31.0.0/16	11000111 00011111 00000000 00000000	R2

...

...

More on IP Address Allocation

- ❖ **Q1:** How does an organization obtain a block of IP addresses?
- ❖ **A1:** Buy from registry or rent from ISP's address space.

- ❖ **Q2:** How does an ISP get a block of addresses?
- ❖ **A2:** ICANN: Internet Corporation for Assigned Names and Numbers
 - Allocates addresses
 - Manages DNS
 - Assigns domain names, resolves disputes

Lectures 6&7: Roadmap

4.1 Overview of Network Layer

4.2 What's Inside a Router

4.3 The Internet Protocol (IP)

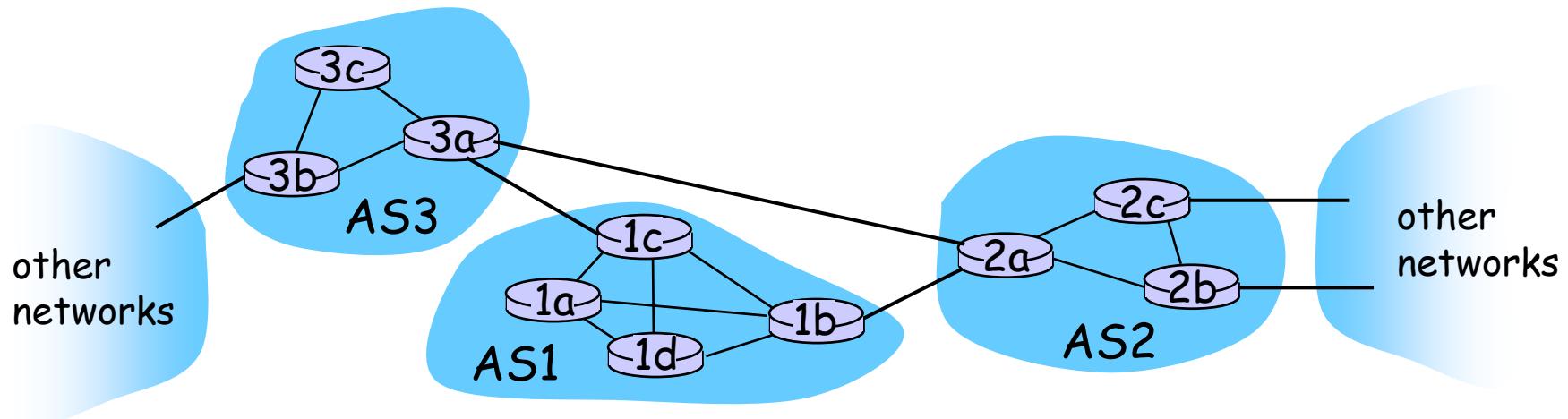
5.2 Routing Algorithms

- 5.2.2 The Distance Vector Routing Algorithm

5.6 ICMP

Internet: Network of Networks

- ❖ The Internet is a “network-of-networks”.
 - A hierarchy of Autonomous Systems (AS), e.g., ISPs, each owns routers and links.
- ❖ Due to the size of the Internet and the decentralized administration of the Internet, routing on the Internet is done hierarchically.



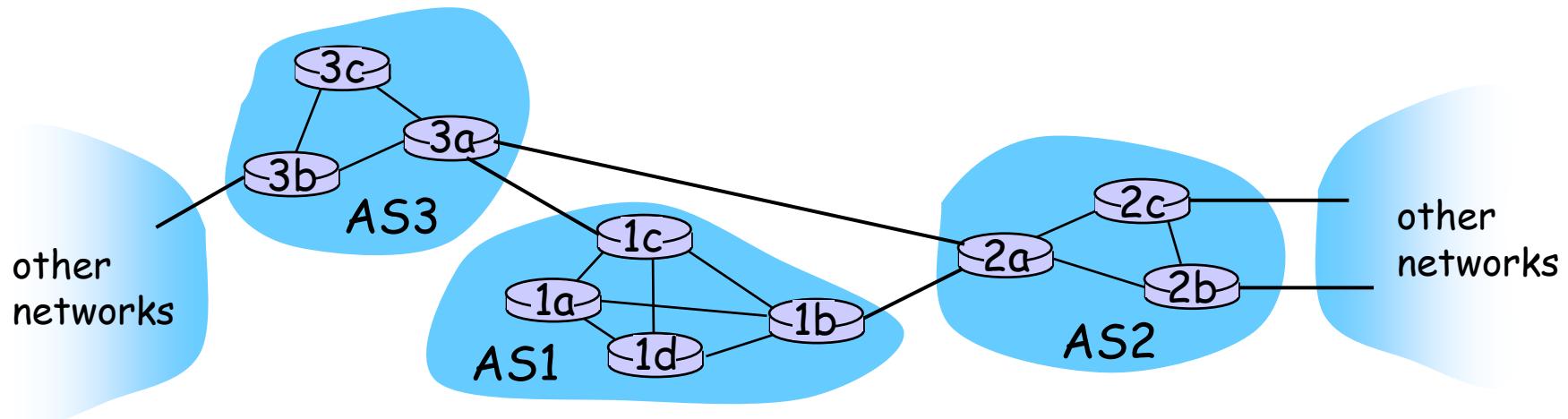
Routing in The Internet

❖ Intra-AS routing

- Finds a good path between two routers within an AS.
- Commonly used protocols: **RIP, OSPF**

❖ Inter-AS routing (not covered)

- Handles the interfaces between ASs.
- The de facto standard protocol: **BGP**



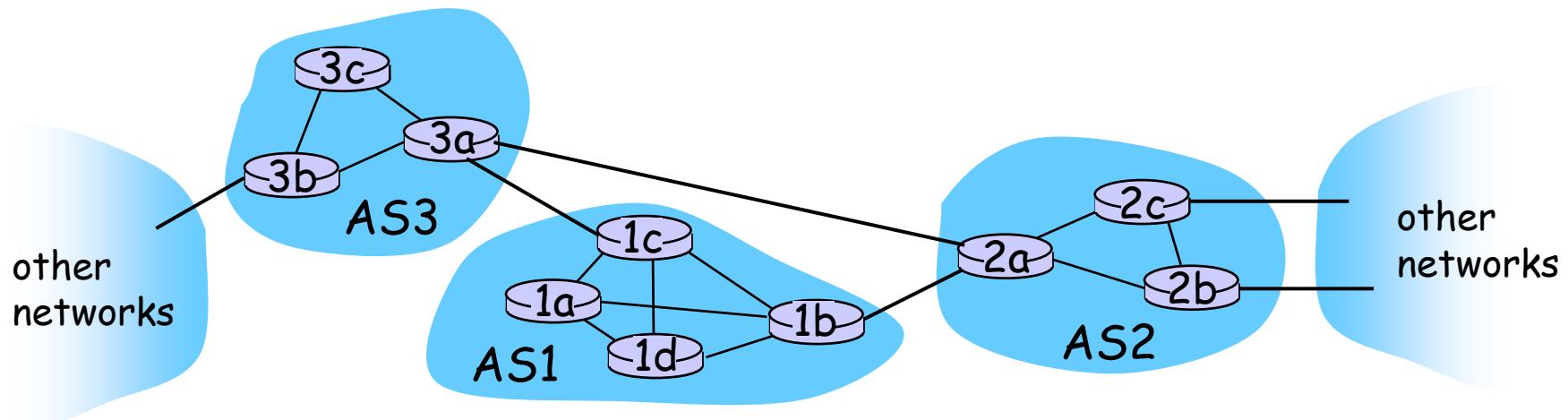
Routing in The Internet

❖ Intra-AS routing

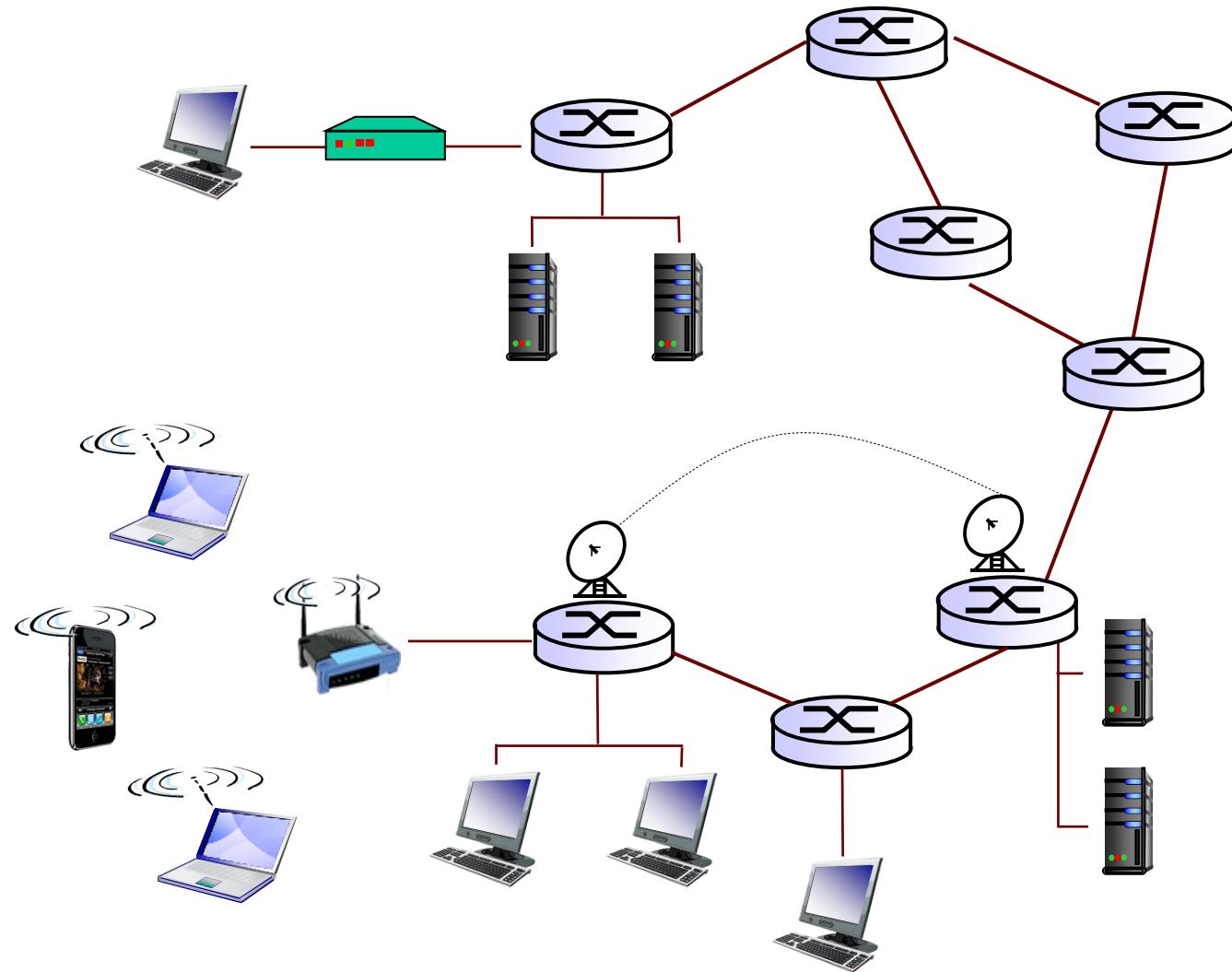
- Single admin, so no policy decisions are needed.
- Routing mostly focus on performance.

❖ Inter-AS routing (not covered)

- Admin often wants to control over how its traffic is routed, who routes through its net, etc.
- Policy may dominate over performance.

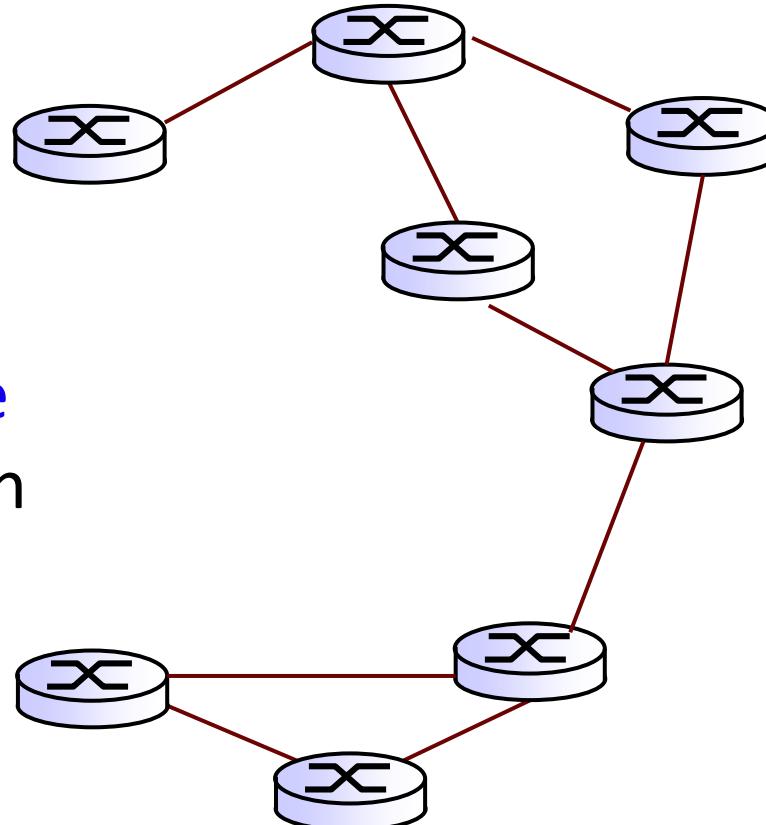


Abstract View of Intra-AS Routing



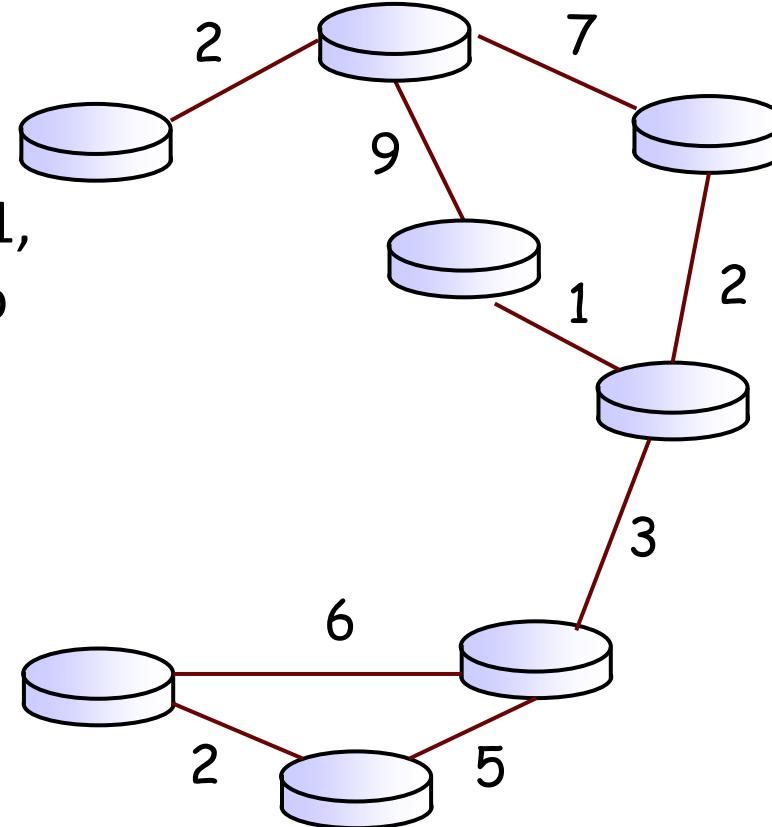
Abstract View of Intra-AS Routing

- ❖ We can abstractly view a network of routers as a **graph**, where **vertices** are routers and **edges** are **physical links** between routers.



Abstract View of Intra-AS Routing

- ❖ We can associate a **cost** to each link.
 - cost could always be 1, or inversely related to bandwidth, or related to congestion.

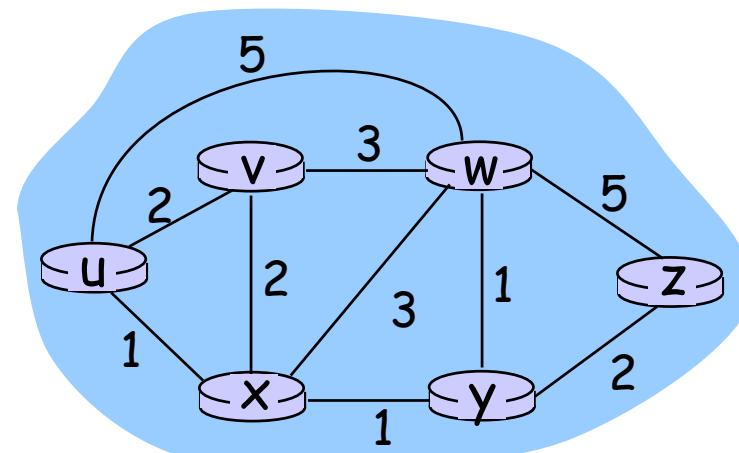


Routing: finding a least cost path between two vertices in a graph

Routing Algorithms Classification

“link state” algorithms

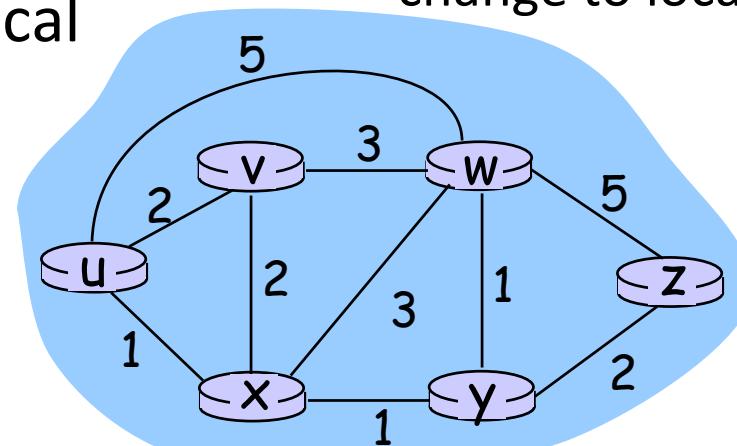
- ❖ All routers have the complete knowledge of network topology and link cost.
 - Routers periodically broadcast link costs to each other.
- ❖ Use Dijkstra algorithm to compute least cost path locally (using global map).
- ❖ Non-examinable ☺



Routing Algorithms Classification

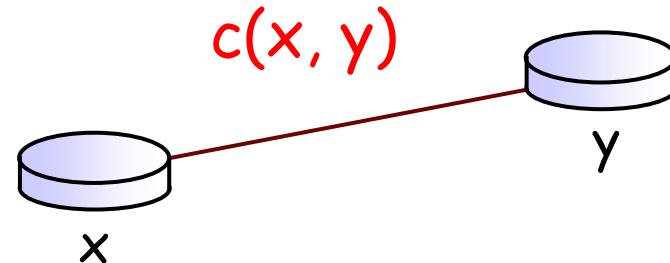
“distance vector” algorithms

- ❖ Routers know physically-connected neighbors and link costs to neighbors.
- ❖ Routers exchange “local views” with neighbors and update own “local views” (based on neighbors’ view).
- ❖ Iterative process of computation
 1. Swap local view with direct neighbours.
 2. Update own’s local view.
 3. Repeat 1 - 2 till no more change to local view.

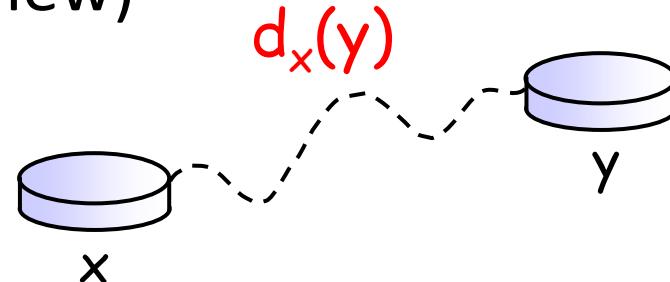


Some Graph Notations

- ❖ $c(x, y)$: the cost of link between routers x and y
 - $= \infty$ if x and y are not direct neighbours



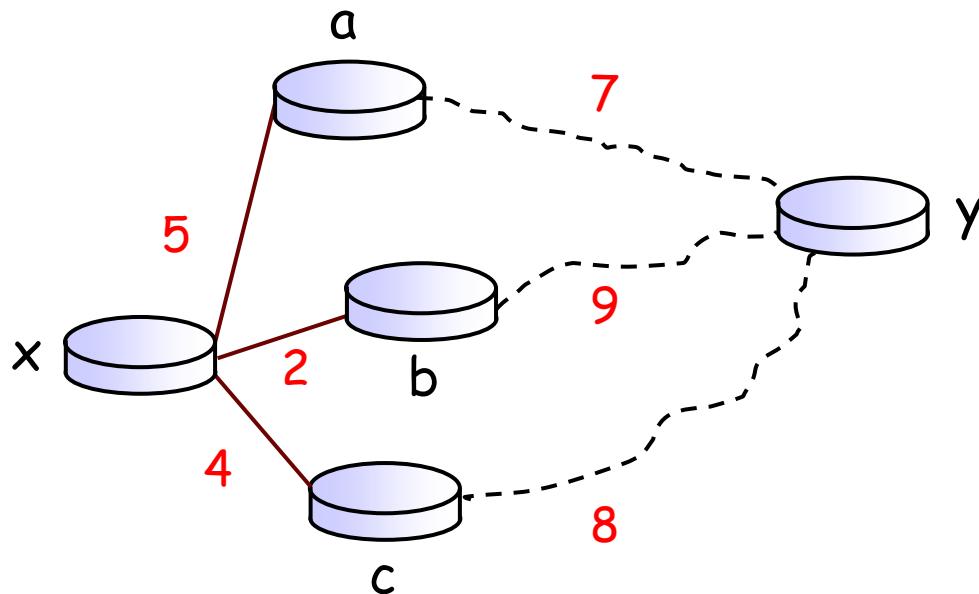
- ❖ $d_x(y)$: the cost of the least-cost path from x to y
(from x 's view)
already computed



Bellman-Ford Equation

$$d_x(y) = \min_v \{ c(x, v) + d_v(y) \}$$

where min is taken over all direct neighbors v of x

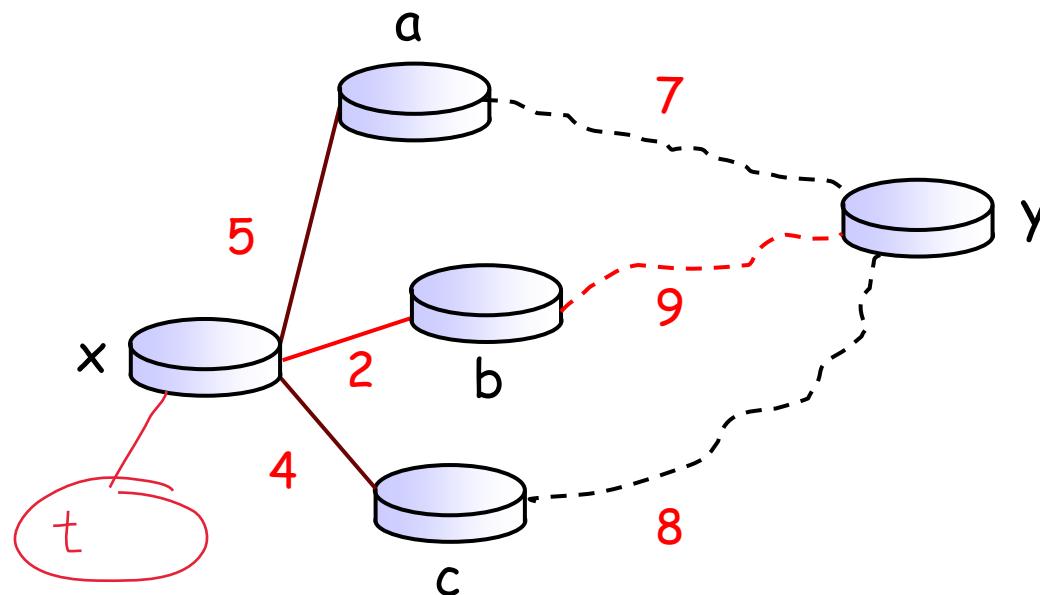


$$\begin{aligned} d_x(y) &= \min_v \{ c(x, a) + d_a(y), \\ &\quad c(x, b) + d_b(y), \\ &\quad c(x, c) + d_c(y) \} \\ &= \min \{ 12, 11, 12 \} = 11 \end{aligned}$$

Bellman-Ford Equation

dynamic programming through the recursive process

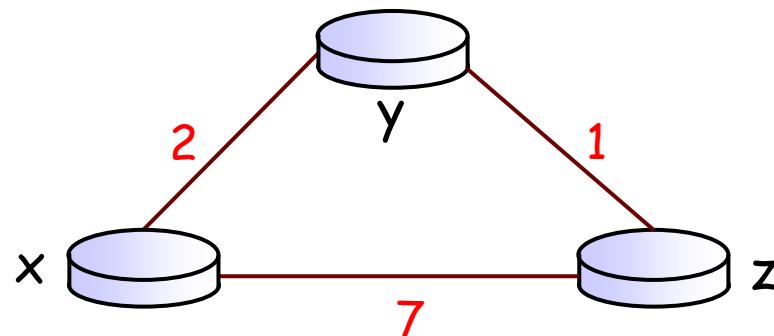
- ❖ To find the least cost path, x needs to know the cost from each of its direct neighbour to y .
- ❖ Each neighbour v sends its **distance vector** (y, k) to x , telling x that the cost from v to y is k .



Now x knows, to reach y , packet should be forward to b and the total cost would be 11.

Bellman-Ford Example

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\}$$



cost to

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from

	x	y	z
x			
y			
z			

from

	x	y	z
x			
y			
z			

x' view

y' view

z' view

Distance Vector Algorithm

- ❖ Every router, x , y , z , sends its distance vectors to its directly connected neighbors.
- ❖ When x finds out that y is advertising a path to z that is cheaper than x currently knows,
 - x will update its distance vector to z accordingly.
 - In addition, x will note down that all packets for z should be sent to y . This info will be used to create forwarding table of x .
- ❖ After every router has exchanged several rounds of updates with its direct neighbors, all routers will know the least-cost paths to all the other routers.

RIP

- ❖ RIP (Routing Information Protocol) implements the DV algorithm. It uses hop count as the cost metric (i.e., insensitive to network congestion).
- ❖ Exchange routing table every 30 seconds over UDP port 520.
- ❖ “Self-repair”: if no update from a neighbour router for 3 minutes, assume neighbour has failed.

Lectures 6&7: Roadmap

4.1 Overview of Network Layer

4.2 What's Inside a Router

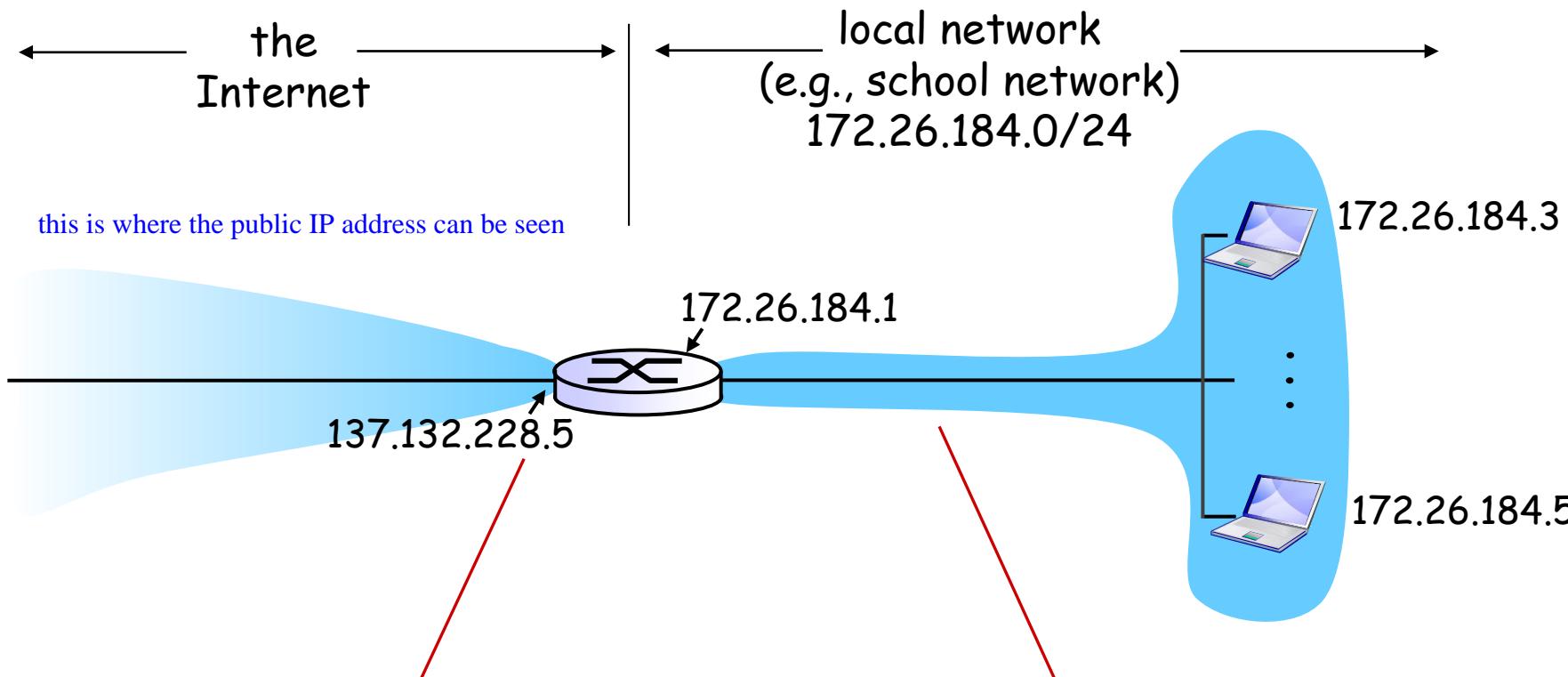
4.3 The Internet Protocol (IP)

- 4.3.4 Network Address Translation

5.2 Routing Algorithms

5.6 ICMP

NAT: Network Address Translation



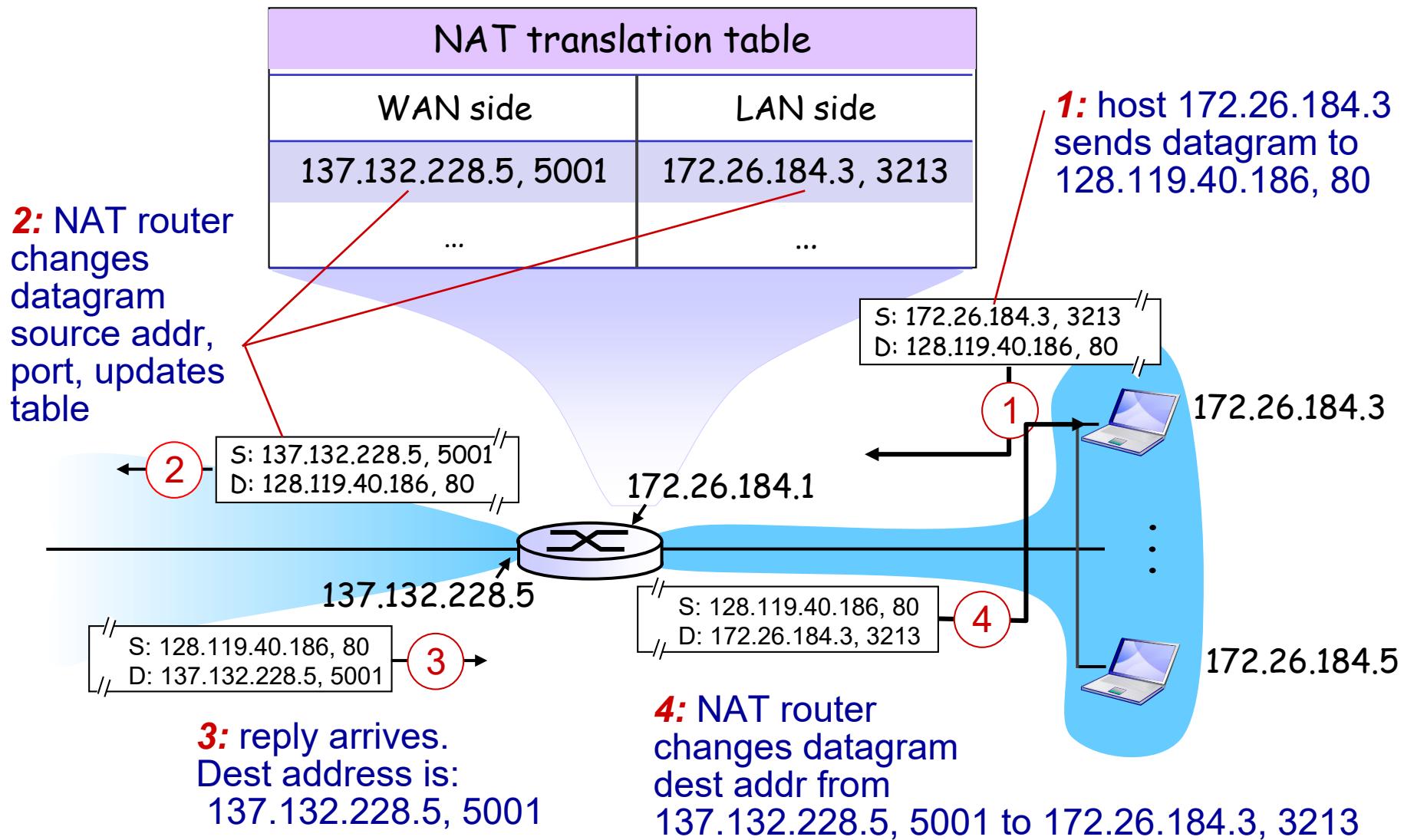
all datagrams *leaving* local network have the *same* source NAT IP address: 137.132.228.5

Within local network, hosts use **private IP addresses** 172.26.184.* for communication

NAT: Implementation

- ❖ NAT routers must:
 - Replace (source IP address, port #) of every **outgoing datagram** to (NAT IP address,  new port #).
 - Remember (in NAT translation table) the mapping from (source IP address, port #) to (NAT IP address, new port #).
 - Replace (NAT IP address, new port #) in destination fields of every **incoming datagram** with corresponding (source IP address, port #) stored in NAT translation table.

NAT: Illustration



NAT: Motivation and Benefits

- ❖ No need to rent a range of public IP addresses from ISP: just one public IP for the NAT router.
- ❖ All hosts use private IP addresses. Can change addresses of hosts in local network without notifying the outside world.
- ❖ Can change ISP without changing addresses of hosts in local network.
- ❖ Hosts inside local network are not explicitly addressable and visible by outside world (a security plus).

Lectures 6&7: Roadmap

4.1 Overview of Network Layer

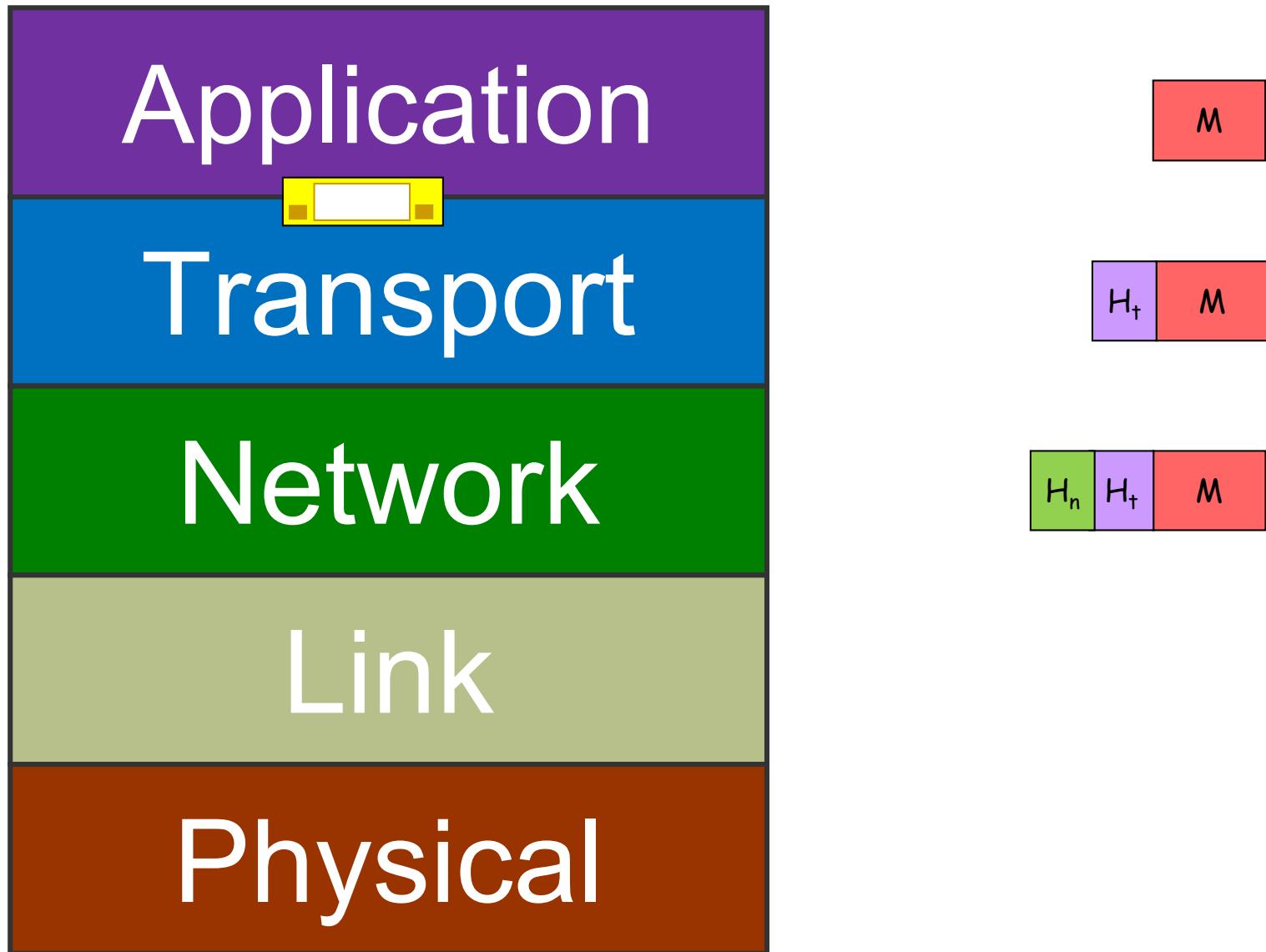
4.2 What's Inside a Router

4.3 The Internet Protocol (IP)

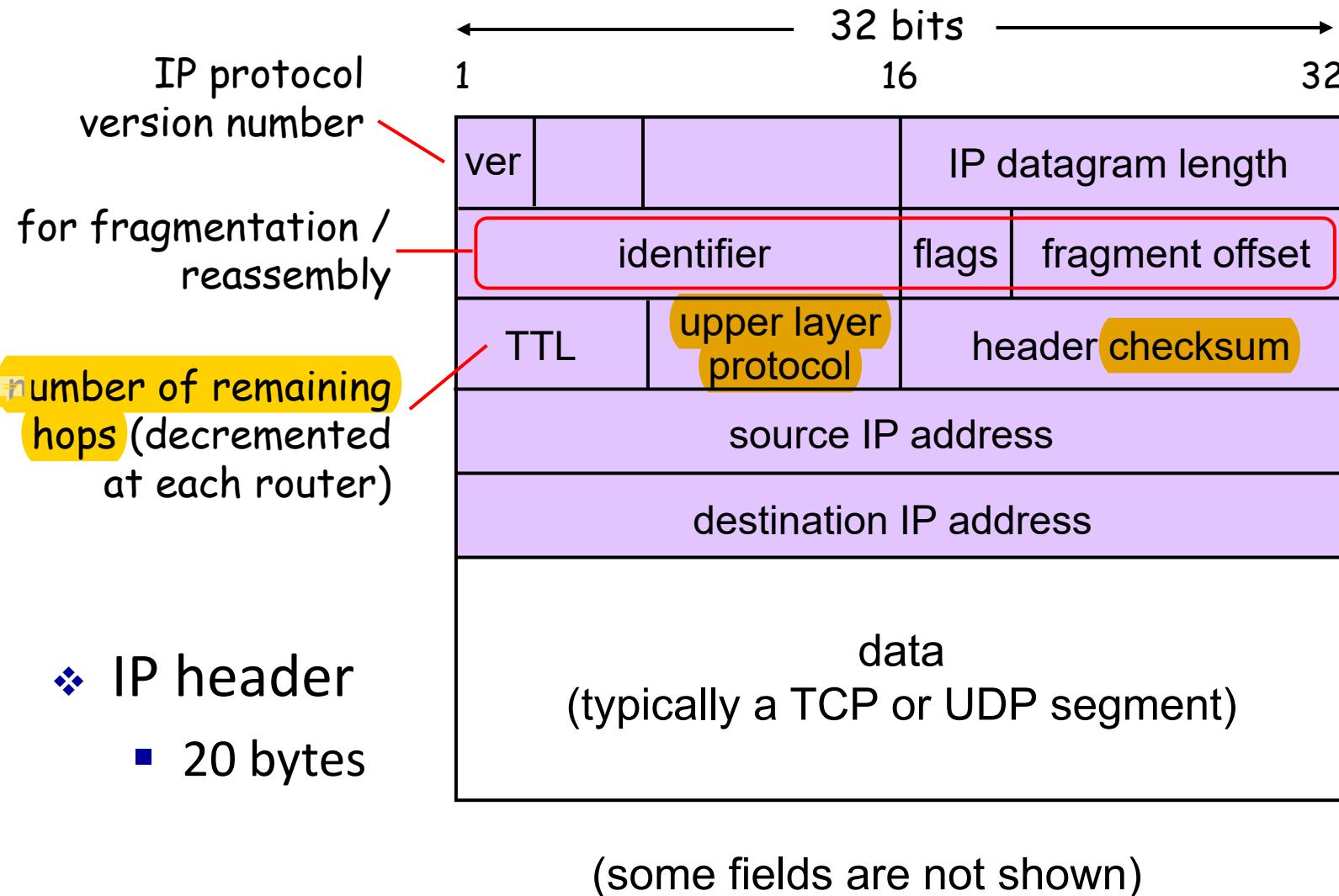
- 4.3.1 IPv4 Datagram Format
- 4.3.2 IPv4 Datagram Fragmentation
- 4.3.5 IPv6 (non-examinable)

5.2 Routing Algorithms

5.6 ICMP

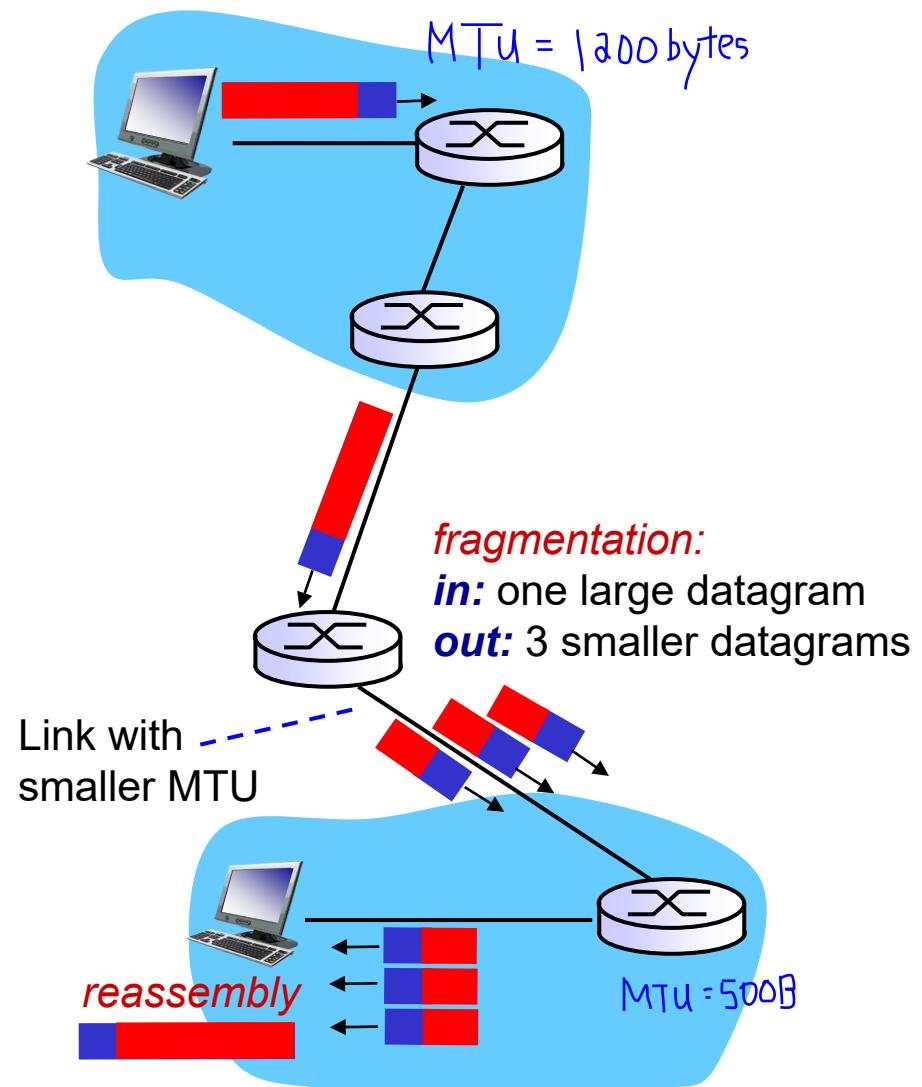


IPv4 Datagram Format

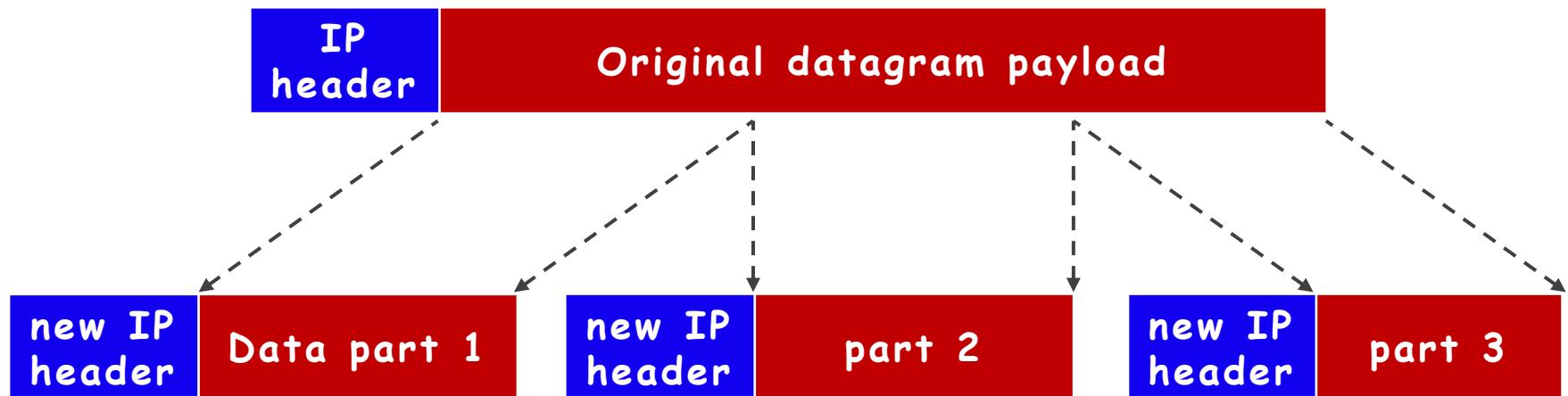


IP Fragmentation & Reassembly

- ❖ Different links may have different **MTU (Max Transfer Unit)** – the maximum amount of data a link-level frame can carry.
- ❖ “Too large” IP datagrams may be fragmented by routers.



IP Fragmentation Illustration



- ❖ Destination host will reassemble the packet.
- ❖ IP header fields are used to identify fragments and their relative order.

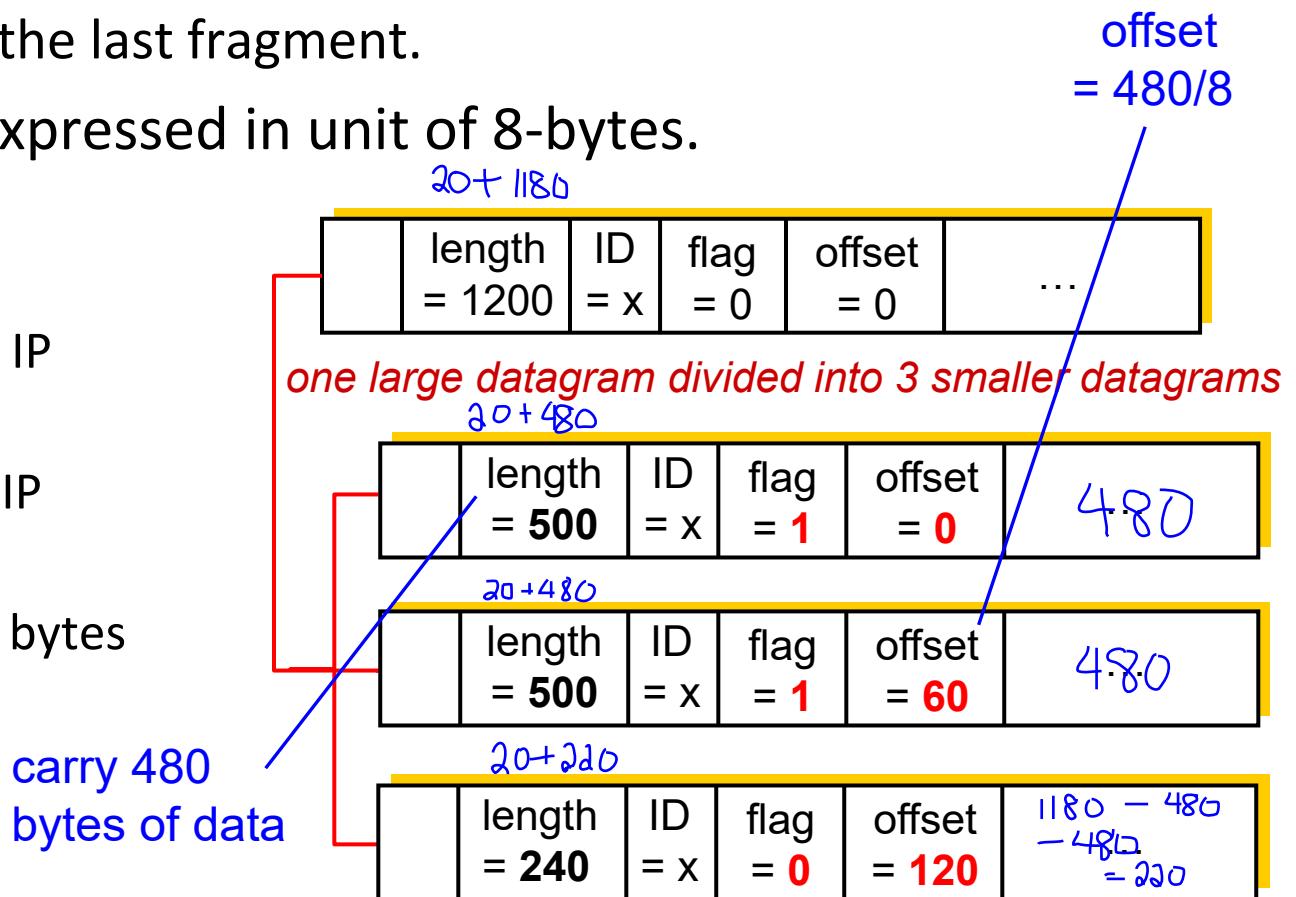
IP Fragmentation

			length
identifier	flags	offset	
source IP address			
destination IP address			

- ❖ Flag (frag flag) is set to
 - 1 if there is next fragment from the same segment.
 - 0 if this is the last fragment.

- ❖ Offset is expressed in unit of 8-bytes.

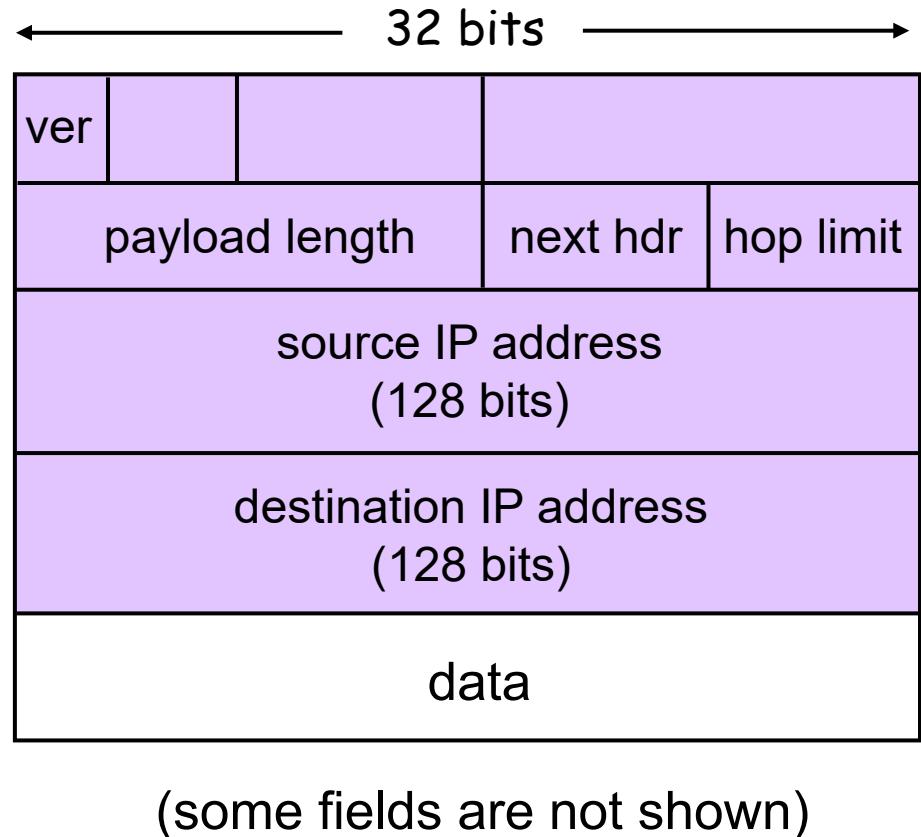
- ❖ Example
 - 20 bytes of IP header
 - 1,200 byte IP datagram
 - MTU = 500 bytes



IPv6

Non-examinable

- ❖ IPv6 is designed to replace IPv4.
- ❖ Primary motivation: 32-bit IPv4 address space is soon to be completely allocated.
- ❖ IPv6 datagram:
 - 40 byte header



Example IPv6 address (in hexadecimal):
2001:0db8:85a3:0042:1000:8a2e:0370:7334

Lectures 6&7: Roadmap

4.1 Overview of Network Layer

4.2 What's Inside a Router

4.3 The Internet Protocol (IP)

5.2 Routing Algorithms

5.6 ICMP

ICMP

- ❖ ICMP (Internet Control Message Protocol) is used by hosts & routers to communicate network-level information.
 - Error reporting: unreachable host / network / port / protocol
 - Echo request/reply (used by ping)
- ❖ ICMP messages are carried in IP datagrams.
 - ICMP header starts after IP header.

ICMP Type and Code

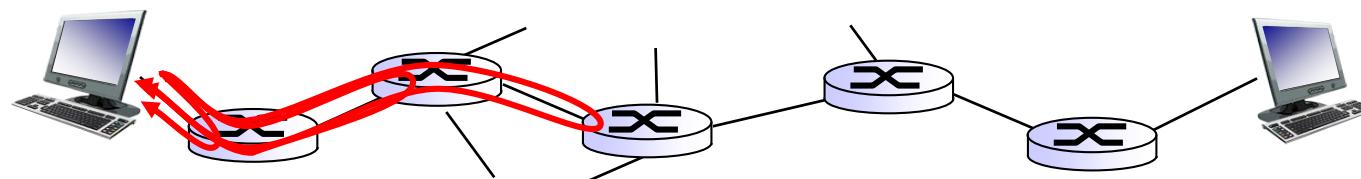
- ❖ ICMP header: Type + Code + Checksum + others.

Type	Code	Description
8	0	echo request (ping)
0	0	echo reply (ping)
3	1	dest host unreachable
3	3	dest port unreachable
11	0	TTL expired
12	0	bad IP header

Selected ICMP Type and subtype (Code)

Examples: *ping* and *traceroute*

- ❖ The command **ping** sees if a remote host will respond to us – do we have a connection?
- ❖ The command **traceroute** sends a series of small packets across a network, and attempts to display the route (or path) that the messages would take to get to a remote host.



Lectures 6&7: Summary

- ❖ An IP address is associated with a network interface. A device may have multiple network interfaces, thus multiple IP addresses.
- ❖ DHCP automates the assignment of IP addresses in an organization's network.
- ❖ On TCP/IP networks, subnets are defined as all devices whose IP addresses have the same network (subnet) prefix.
- ❖ Subnet mask is useful in checking if two hosts are on the same subnet.

Lectures 6&7: Summary

- ❖ Routing is the process of selecting best paths in a network.
- ❖ **NAT** maps one IP addresses space into another.
 - Commonly used to hide an entire private IP address space behind a single public IP address.
 - NAT router uses stateful translation tables to remember the mapping.
- ❖ **ICMP** is used by routers to send error messages.
 - E.g. when TTL is 0, a packet is discarded and an ICMP error message is sent to the datagram's source address.

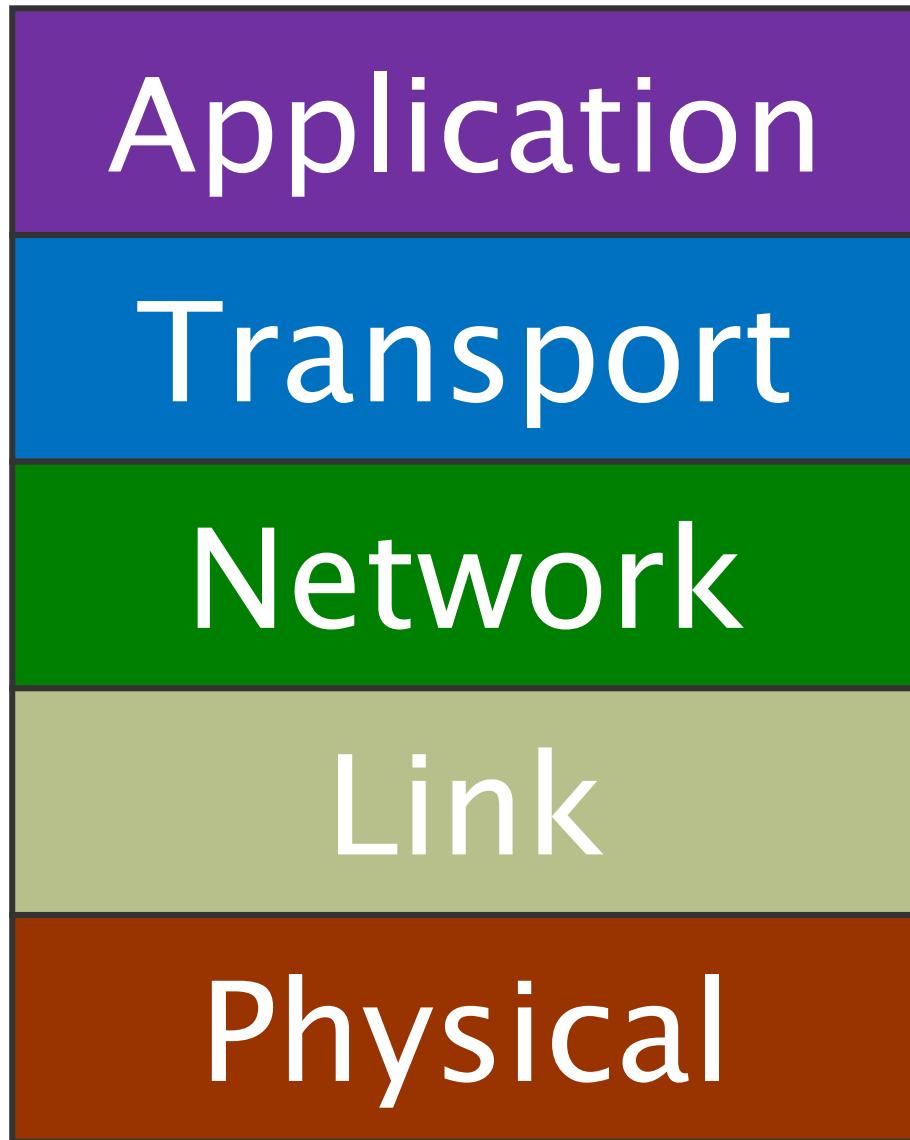
CS2105

An *Awesome* Introduction to Computer Networks

Lecture 8: The Link Layer, Part I



Department of Computer Science
School of Computing



You are
here

Lectures 8&9: The Link Layer

After the next 2 classes, you are expected to understand:

- ❖ the role of link layer and the services it could provide.
- ❖ how parity and CRC scheme work.
- ❖ different methods for accessing shared medium.
- ❖ how ARP allows a host to discover the MAC addresses of other nodes in the same subnet.
- ❖ the role of switches in interconnecting subnets in a LAN.

Lecture 8: Roadmap

6.1 Introduction to the Link Layer

6.2 Error Detection and Correction

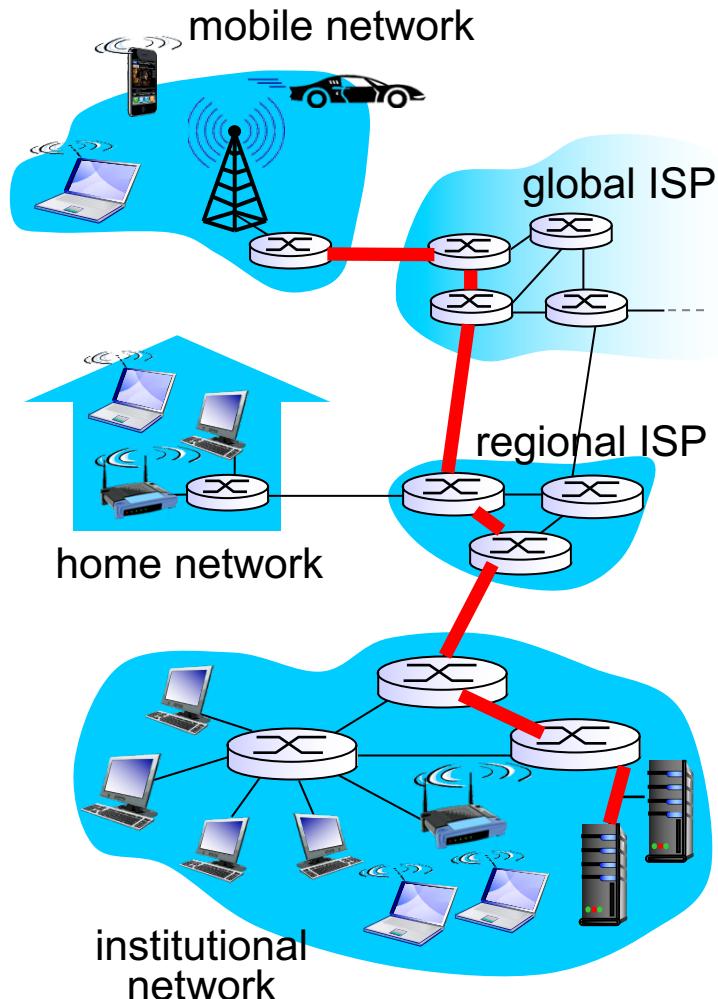
6.3 Multiple Access Links and Protocols

6.4 Switched Local Area Networks

Kurose Textbook, Chapter 6
(Some slides are taken from the book)

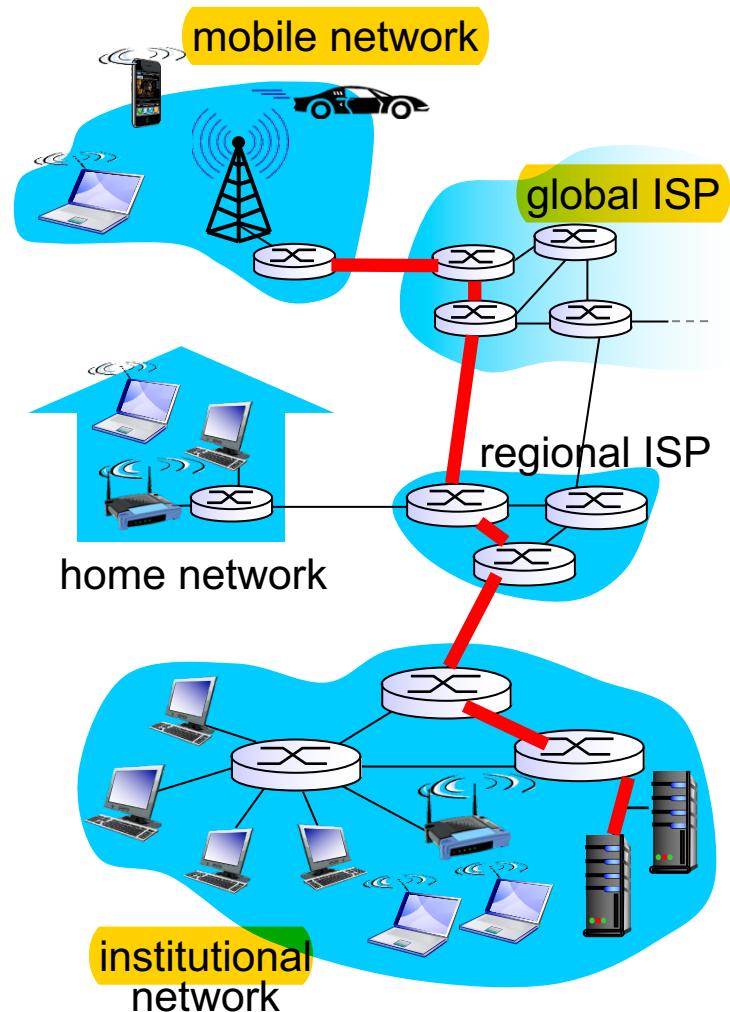
Link Layer: Introduction (1/2)

- ❖ **Network layer** provides communication service between any two hosts.
- ❖ An IP datagram may travel through multiple routers and links before it reaches destination.

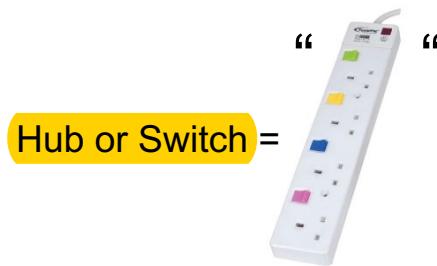


Link Layer: Introduction (2/2)

- ❖ Link layer sends datagram between adjacent nodes (hosts or routers) over a single link.
 - IP **datagrams** are encapsulated in link-layer **frames** for transmission.
 - Different link-layer protocols may be used on different links.
 - **Each protocol may provide a different set of services.**



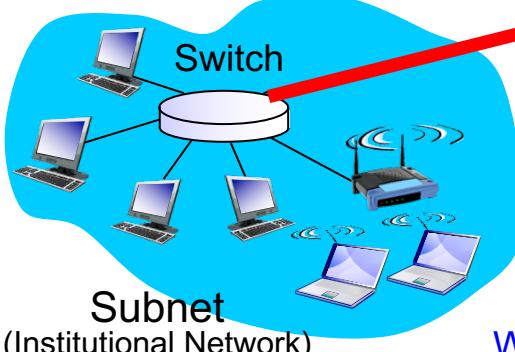
Routing: Big Picture



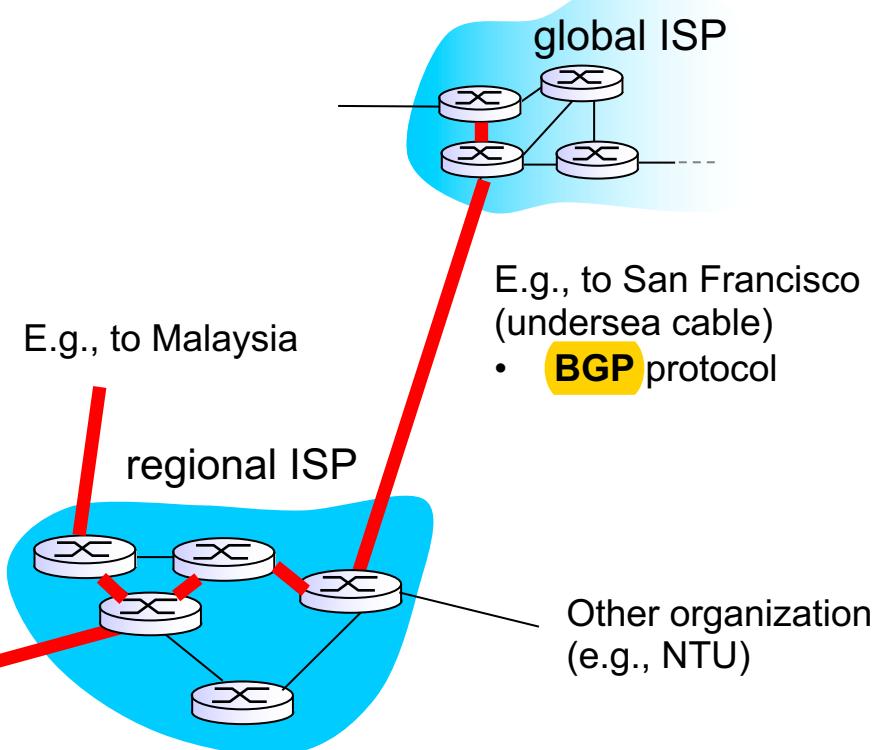
Hub or Switch =

DHCP protocol provides:

- IP address, e.g., 192.168.0.x/24
- Subnet mask, e.g., 255.255.255.0
- IP of DNS server
- IP of Default Gateway (e.g.: 192.168.0.1)



Within subnet
• ARP protocol



Which link/path to choose?

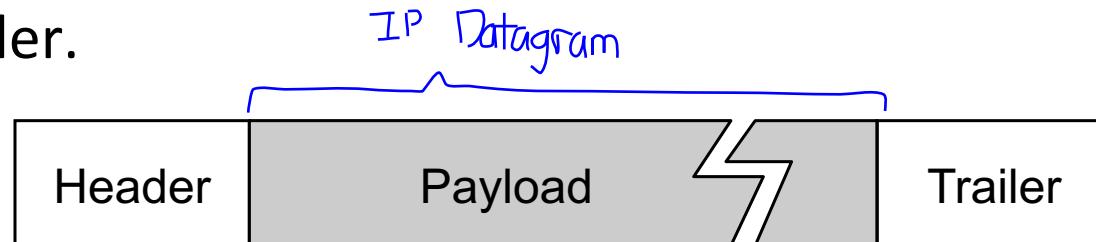
Intra-AS routing

- RIP, OSPF protocols
- Distributed algo.
- Build routing table

Possible Link Layer Services (1/2)

❖ Framing

- Encapsulate datagram into frame, adding header and trailer.



❖ Link access control

- When multiple nodes *share* a single link, need to coordinate which nodes can send frames at a certain point of time.



humans at a
cocktail party
(shared air)

Possible Link Layer Services (2/2)

❖ Reliable delivery

- Seldom used on low bit-error link (e.g., fibre) but often used on error-prone links (e.g., wireless link).

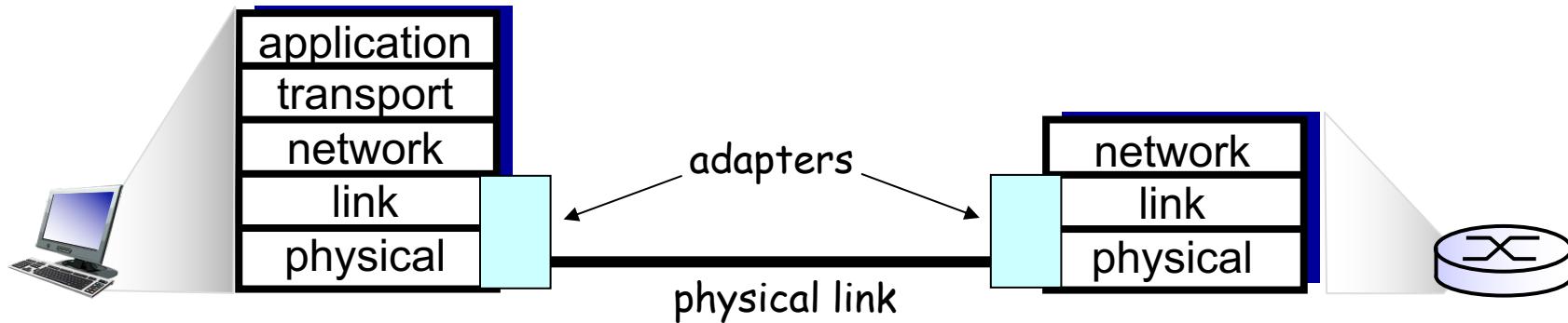
❖ Error detection

- Errors are usually caused by signal attenuation or noise.
- Receiver detects presence of errors.
 - may signal sender for retransmission or simply drops frame

❖ Error correction

- Receiver identifies and corrects bit error(s) without resorting to retransmission.

Link Layer Implementation



- ❖ Link layer is implemented in “adapter” (aka  NIC) or on a chip.
 - E.g., Ethernet card/chipset, 802.11 card
- ❖ Adapters are semi-autonomous, implementing both link & physical layers.



Lectures 8&9: Roadmap

6.1 Introduction to the Link Layer

6.2 Error Detection and Correction

- 6.2.1 Parity Checks
- 6.2.3 Cyclic Redundancy Check (CRC)

6.3 Multiple Access Links and Protocols

6.4 Switched Local Area Networks

Error Detection and Correction

- ❖ Popular error detection schemes:
 - Checksum (used in TCP/UDP/IP)
 - Parity Checking
 - CRC (commonly used in link layer)
- ❖ Error detection schemes are not 100% reliable!
 - may miss some errors, but rarely.
 - larger error detection and correction (EDC) field yields better detection (and even correction).

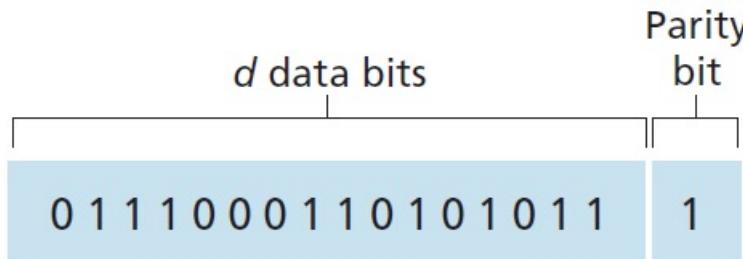


Tradeoff - the better the error detection / correction scheme is, the higher the overhead will be

Parity Checking

Single bit parity

- ❖ can detect single bit errors in data.



This will fail when 2 bits flip

- since their effects will be cancelled out

2d bit parity has a much higher overhead in terms of space
 - requires 9 more extra bits for the same 15 bit data
 as compared to single bit parity only needing 1

Two-dimensional bit parity

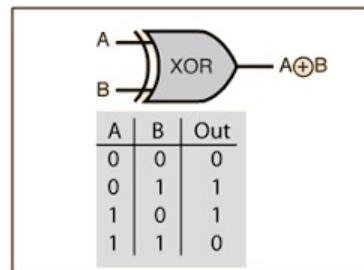
- ❖ can detect and correct single bit errors in data.
- ❖ can detect any two-bit error in data.

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

Cyclic Redundancy Check (CRC)

- ❖ Powerful error-detection coding that is widely used in practice (e.g., Ethernet, Wi-Fi)
 - D : data bits, viewed as a binary number.
 - G : generator of $r + 1$ bits, agreed by sender and receiver beforehand.
 - R : will generate CRC of r bits.



$$\begin{array}{r}
 1001 \overline{) 101110000} \\
 1001 \\
 \hline
 10 \\
 1001 \\
 \hline
 1100 \\
 1001 \\
 \hline
 1010 \\
 1001 \\
 \hline
 011
 \end{array}$$

G
 D
 R

Cyclic Redundancy Check (CRC)

- ❖ CRC calculation is done in bit-wise XOR operation without carry or borrow.

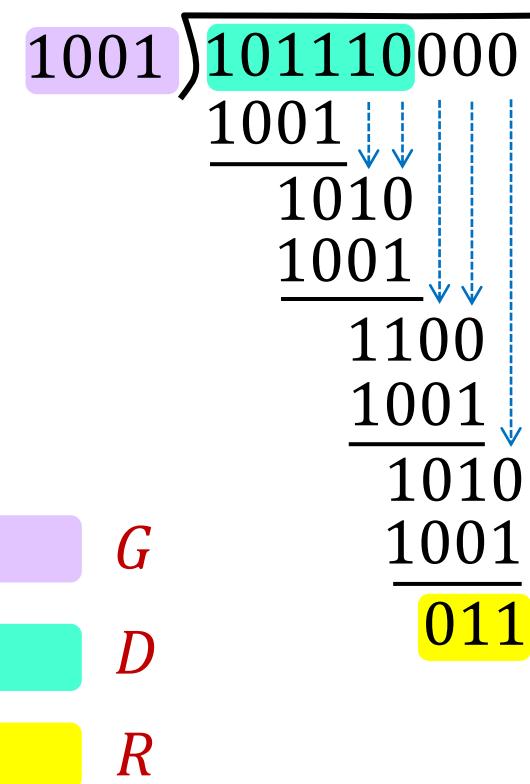
- ❖ Sender sends (D, R)

101110011

- ❖ Receiver knows G , divides (D, R) by G .

- If non-zero remainder:
error is detected!

Example: $r = 3$



Lectures 8&9: Roadmap

6.1 Introduction to the Link Layer

6.2 Error Detection and Correction

6.3 Multiple Access Links and Protocols

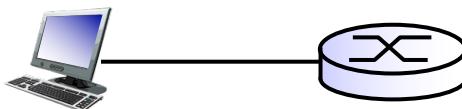
- 6.3.1 Channel Partitioning Protocols
- 6.3.2 Random Access Protocols
- 6.3.3 Taking-Turns Protocols

6.4 Switched Local Area Networks

Two Types of Network Links

❖ Type 1: point-to-point link

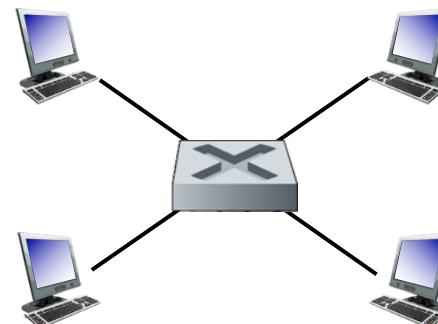
- A sender and a receiver connected by a dedicated link
- Example protocols: Point-to-Point Protocol (PPP), Serial Line Internet Protocol (SLIP)
 - No need for multiple access control



A host connects to router through a dedicated link



RJ45



A point-to-point link between Ethernet switch and a host

Two Types of Network Links

❖ Type 2: broadcast link (shared medium)

- Multiple nodes connected to a shared broadcast channel.
- When a node transmits a frame, the channel broadcasts the frame and each other node receives a copy.



802.11 Wi-Fi



Satellite

Ethernet with bus topology

Multiple Access Protocols

- ❖ In a broadcast channel, if two or more nodes transmit simultaneously
 - Every node receives multiple frames at the same time
→ frames *collide* at nodes and none would be correctly read.
- ❖ Multiple Access Protocol
 - distributed algorithm that determines how nodes share channel, i.e. when a node can transmit.
 - However, coordination about channel sharing must use channel itself!
 - no out-of-band channel signaling

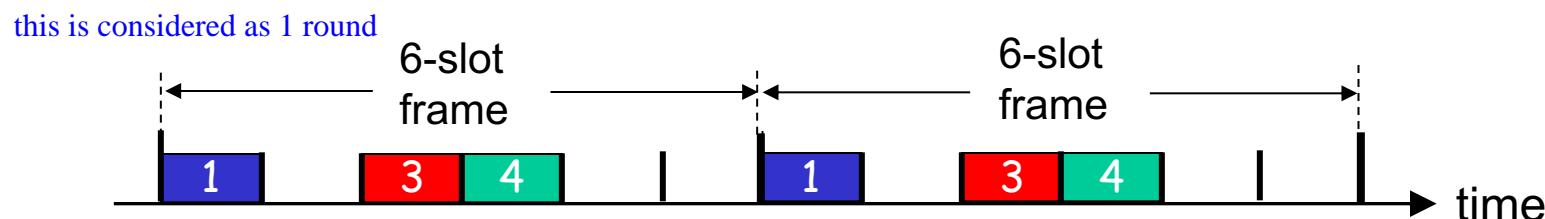
Multiple Access Protocols

- ❖ Multiple access protocols can be categorized into three broad classes:
 - **Channel partitioning**
 - divide channel into fixed, smaller “pieces” (e.g., time slots, frequency).
 - allocate piece to node for exclusive use.
 - **“Taking turns”**
 - nodes take turns to transmit.
 - **Random Access**
 - channel is not divided, collisions are possible.
 - “recover” from collisions.

Channel Partitioning Protocols

❖ TDMA (time division multiple access)

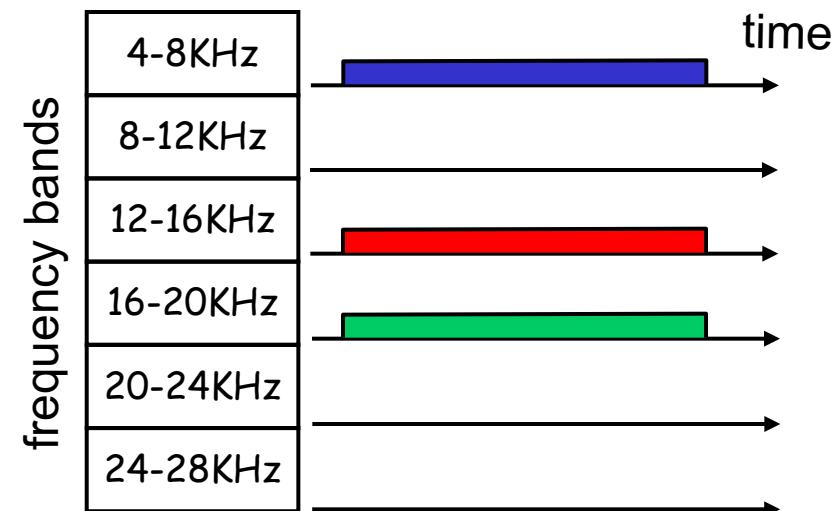
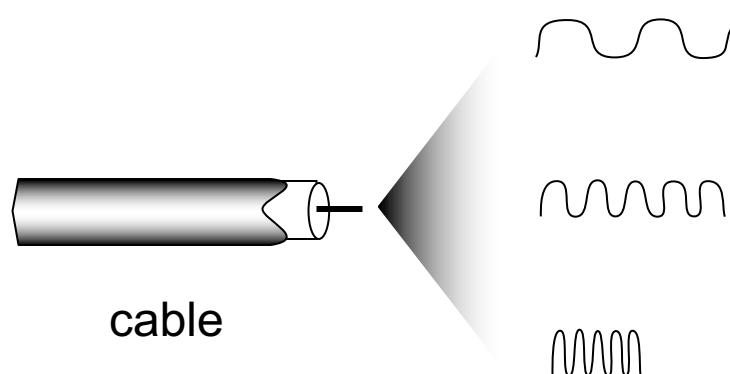
- Access to channel in “rounds”.
- Each node gets fixed length slot (length = frame transmission time) in each round.
- Unused slots go idle.
- Example: 6 nodes sharing a link, 1, 3, 4 have frames, slots 2, 5, 6 are idle.



Channel Partitioning Protocols

❖ FDMA (frequency division multiple access)

- Channel spectrum is divided into frequency bands.
- Each node is assigned a fixed frequency band.
- Unused transmission time in frequency bands go idle.
- Example: 6 nodes, 1, 3, 4 have frames, frequency bands 2, 5, 6 are idle.



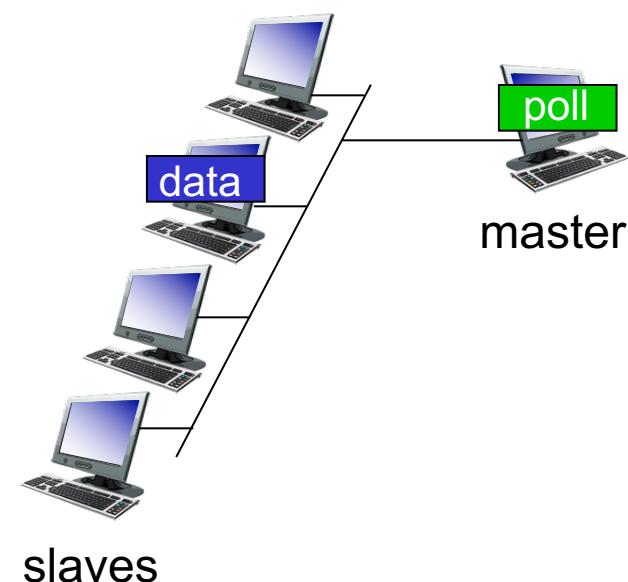
Multiple Access Protocols

- ❖ Multiple access protocols can be categorized into three broad classes:
 - Channel partitioning
 - divide channel into fixed, smaller “pieces” (e.g., time slots, frequency).
 - allocate piece to node for exclusive use.
 - **“Taking turns”**
 - nodes take turns to transmit.
 - Random Access
 - channel is not divided, collisions are possible.
 - “recover” from collisions.

“Taking Turns” Protocols

Polling:

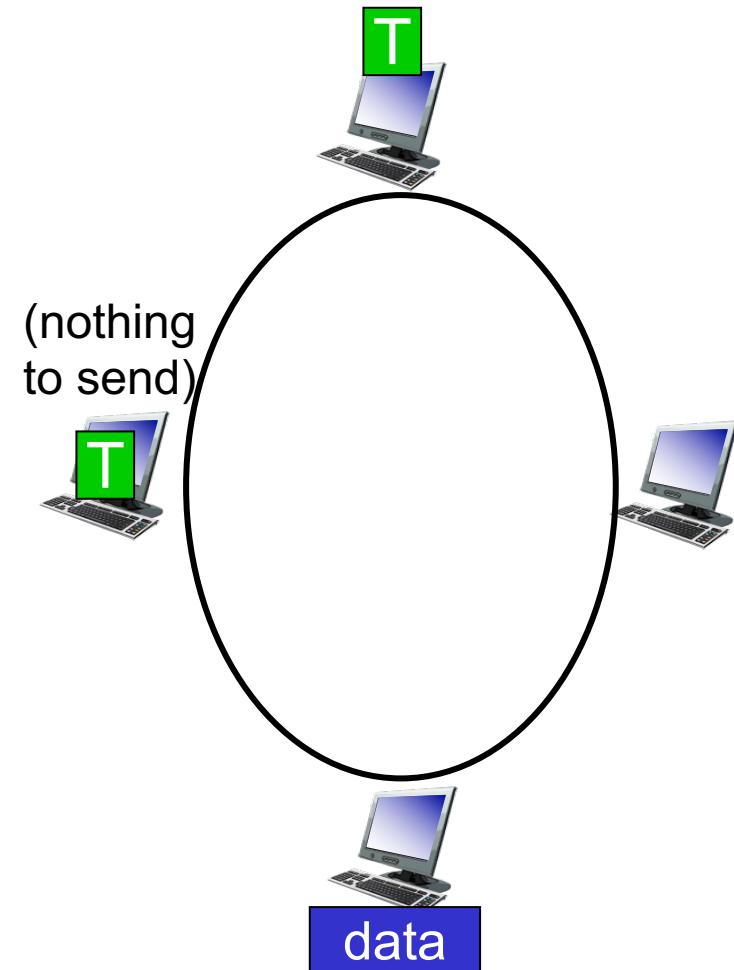
- ❖ master node “invites” slave nodes to transmit in turn.
- ❖ concerns:
 - polling overhead
 - single point of failure (master node)



“Taking Turns” Protocols

Token passing:

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



Multiple Access Protocols

- ❖ Multiple access protocols can be categorized into three broad classes:
 - Channel partitioning
 - divide channel into smaller “pieces” (e.g., time slots, frequency).
 - allocate piece to node for exclusive use.
 - “Taking turns”
 - nodes take turns to transmit.
 - **Random Access**
 - channel is not divided, collisions are possible.
 - “recover” from collisions.

Random Access Protocols

- ❖ When node has packet to send
 - no *a priori* coordination among nodes
 - two or more transmitting nodes → “collision”
- ❖ Random access protocols specify:
 - how to detect collisions
 - how to recover from collisions (e.g., via delayed retransmissions)
- ❖ We will skip the mathematical formulas on the efficiency of random access protocols.

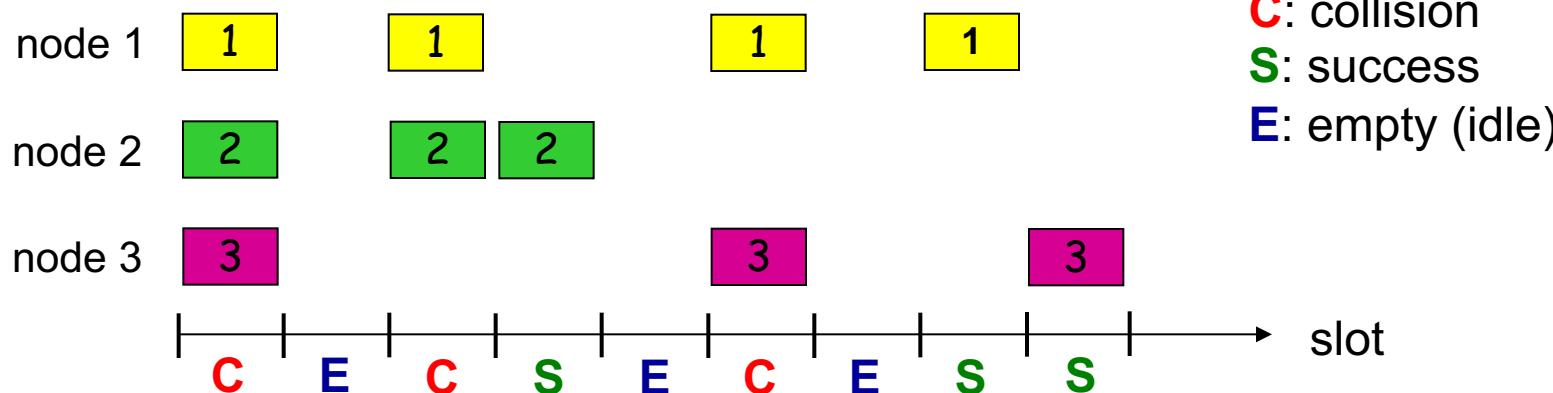
Slotted ALOHA

Assumptions:

- ❖ All frames are of equal size.
- ❖ Time is divided into slots of equal length (length = time to transmit 1 frame).
- ❖ Nodes start to transmit only at the beginning of a slot.

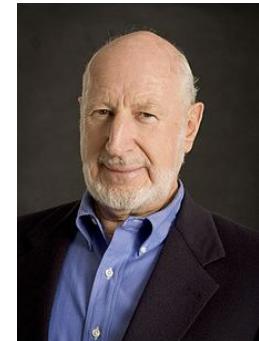
Operations:

- ❖ Listens to the channel while transmitting (**collision detection**).
- ❖ *if collision happens:* node retransmits a frame in each subsequent slot with probability p until success.



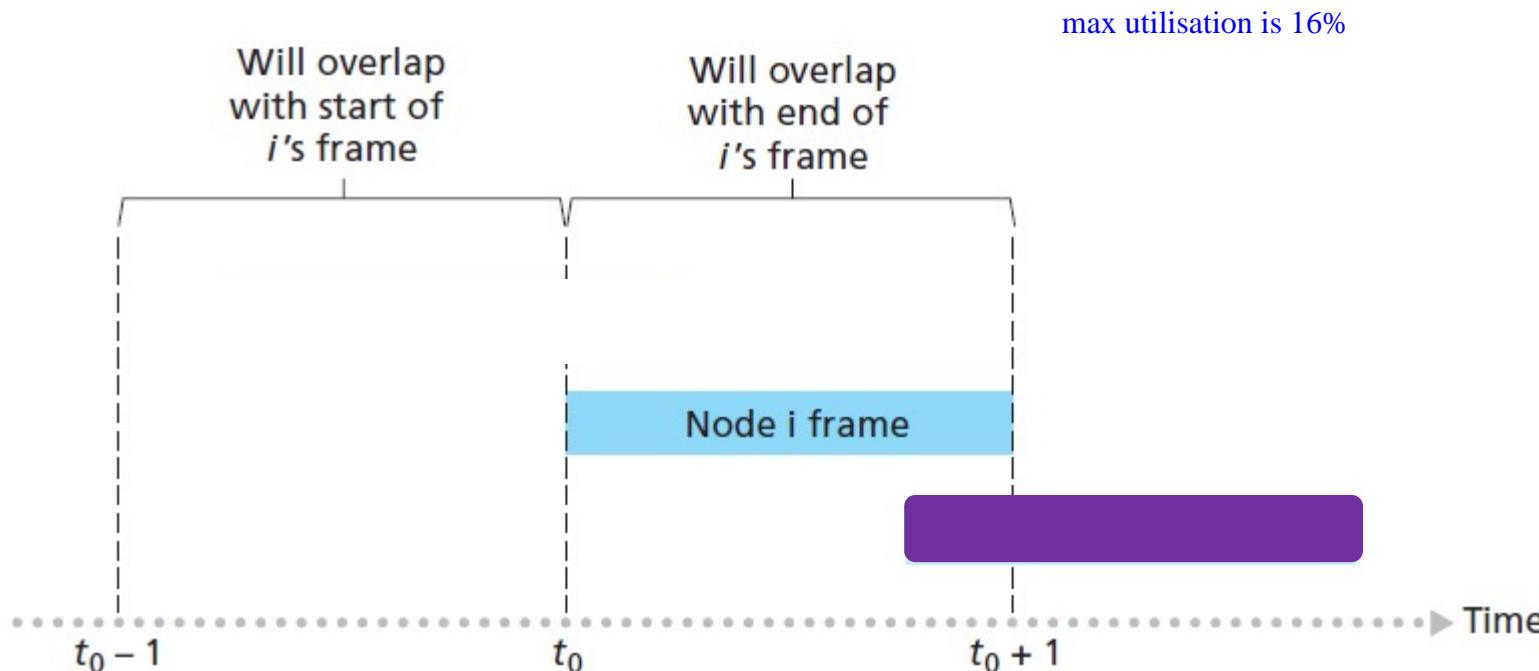
A Little Side Note

- ❖ Q: Why is it called ALOHA?
- ❖ A: The **ALOHA**net, also known as the ALOHA System, or simply ALOHA, was a pioneering computer networking system developed – maybe you can guess it – at the University of Hawaii.
- ❖ Norman Abramson was the leader of the team.
- ❖ The idea was to use a radio network to connect Oahu and the other Hawaiian islands together. ALOHA made use of one, shared, inbound channel, and thus requiring a novel ***multiple access protocol***.



Pure (unslotted) ALOHA

- ❖ Even simpler: no slot, no synchronization
 - When there is a fresh frame: transmit immediately
 - Chance of collision increases:
 - frame sent at t_0 collides with other frames sent in $(t_0 - 1, t_0 + 1)$

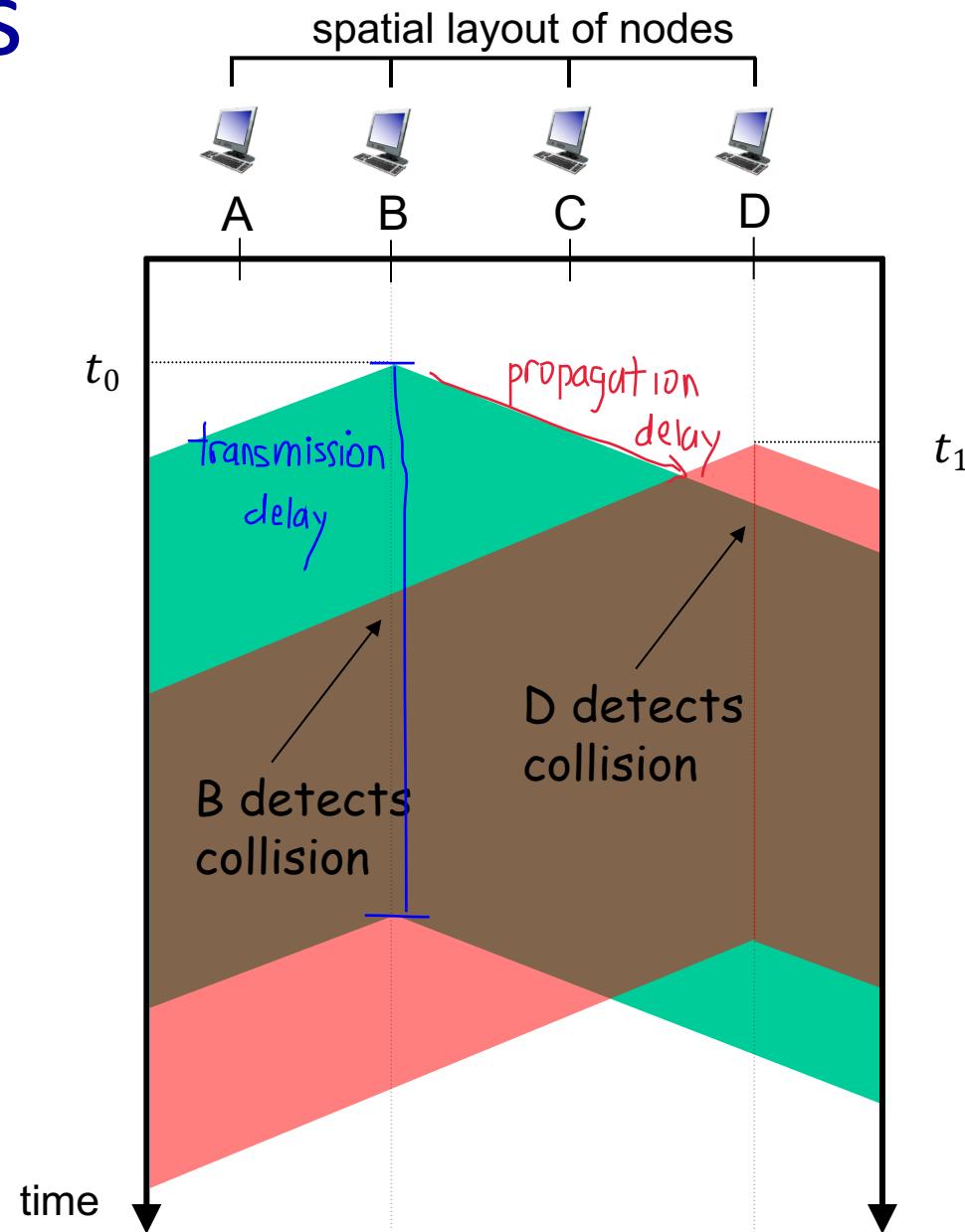


Carrier Sense Multiple Access

- ❖ **CSMA** (carrier sense multiple access)
 - Sense the channel before transmission:
 - if channel is sensed idle, transmit frame
 - if channel sensed busy, defer transmission
- ❖ Human analogy: don't interrupt others!
- ❖ **Q:** Will collision ever happen in CSMA?
 - collisions may still exist, e.g., when two nodes sense the channel idle at the same time and both start transmission.

CSMA Collisions

- ❖ Collisions can still occur:
 - propagation delay means two nodes may not hear each other's transmission immediately.



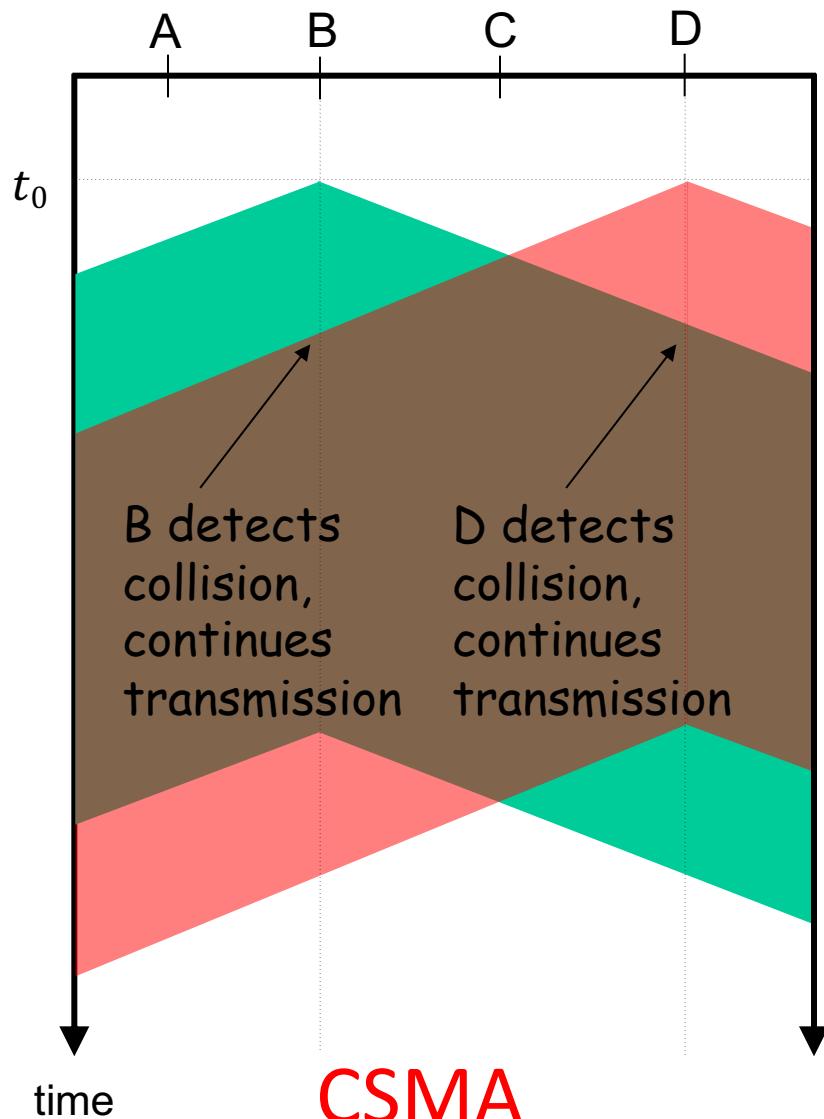
CSMA/CD (Collision Detection)

❖ CSMA/CD

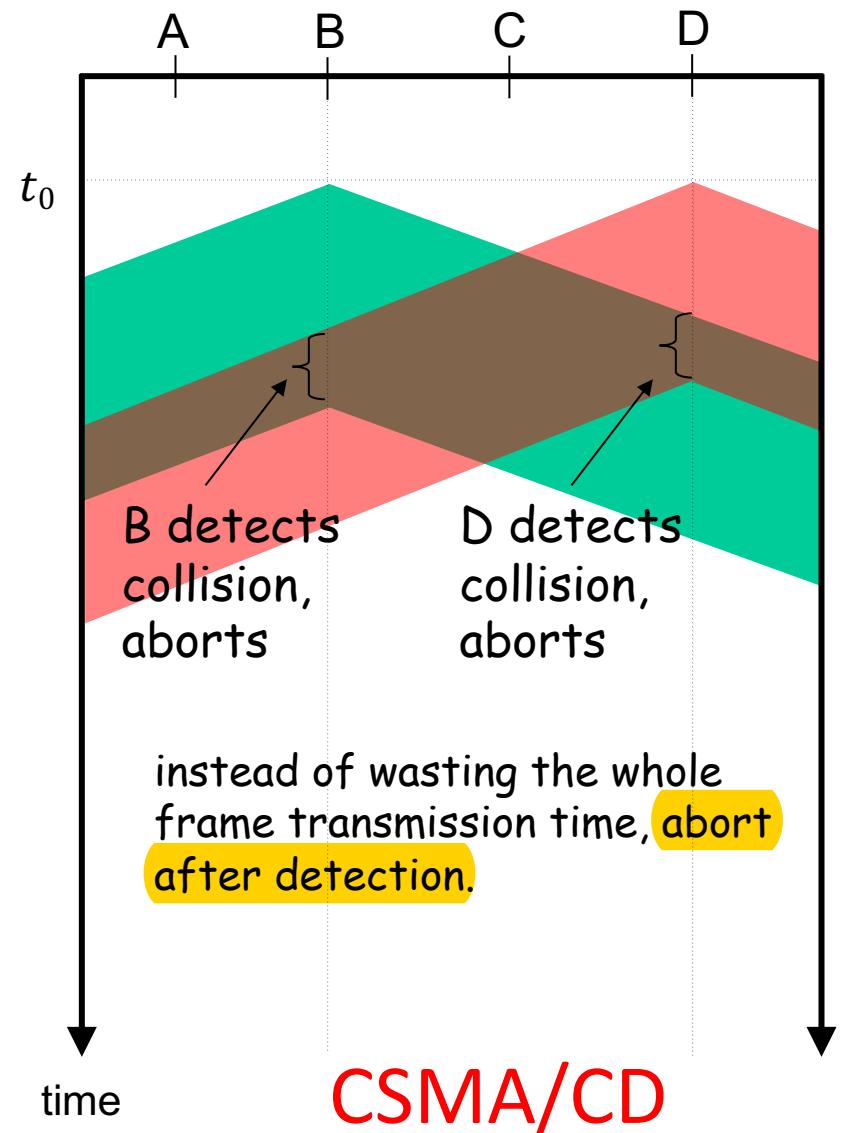
- Carrier sensing & deferral as in CSMA
- When collision is detected, transmission is aborted (reducing channel wastage).
- Retransmit after a random amount of time.
 - An example algorithm will be given in the next lecture
- CSMA/CD is the underlying method of the early Ethernet, invented by Bob Metcalfe.



spatial layout of nodes

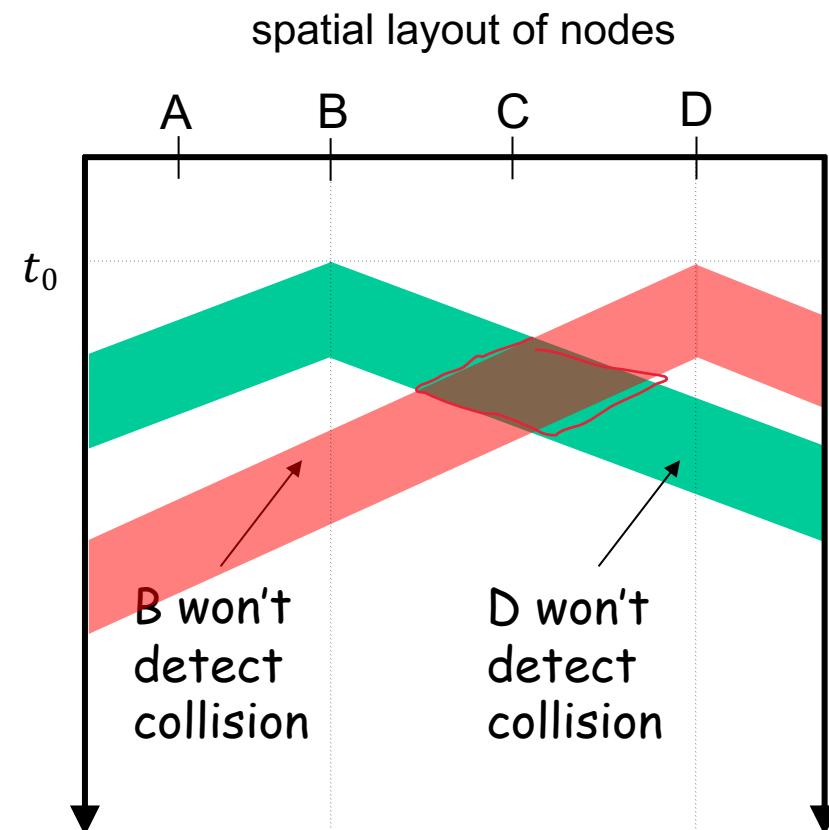


spatial layout of nodes



Minimum Frame Size

- ❖ What if the frame size is too small?
 - Collision happens but may not be detected by sending nodes.
 - No retransmission!
- ❖ For example, Ethernet requires a minimum frame size of 64 bytes.



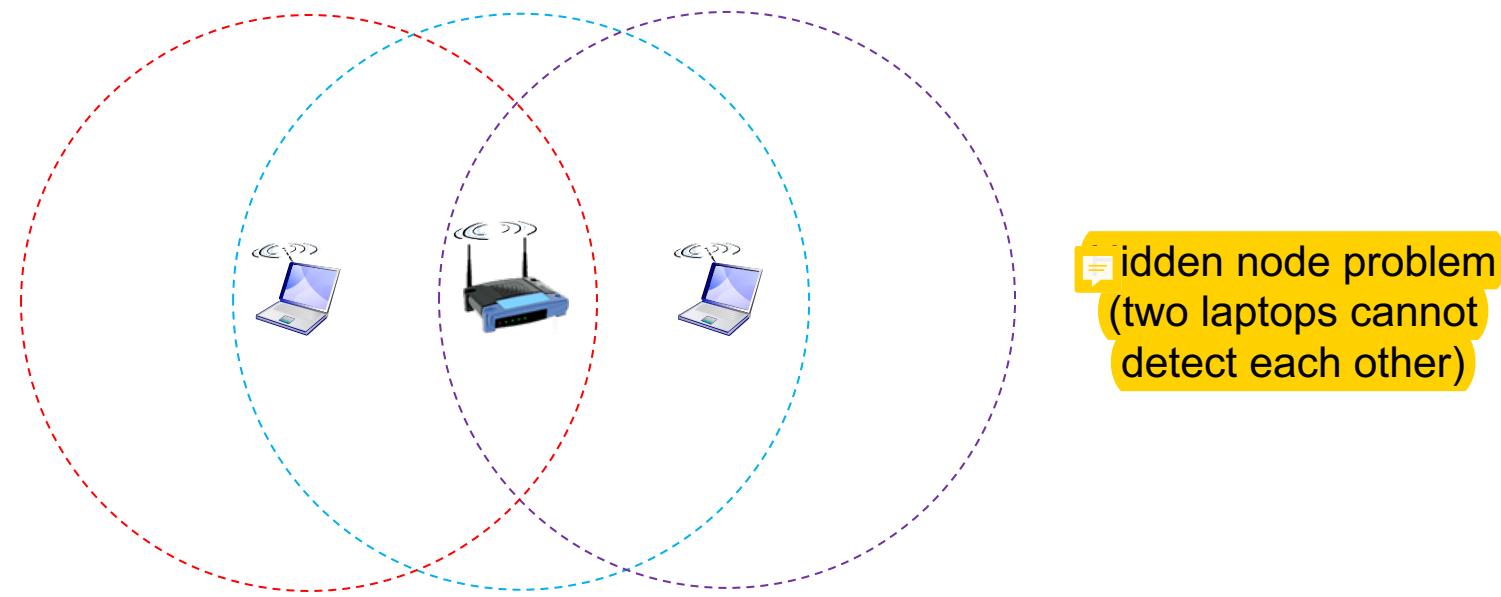
A Little Side Note

- ❖ **Q:** Why was early Ethernet using CSMA/CD?
Nowadays Ethernet is mostly point-to-point, e.g., directly from a computer to a switch, so no need for multiple access.
- ❖ **A:** What we use today is called switched Ethernet.
Switches are cheap and we connect every computer in a star-topology to a switch.
- ❖ In the early days, Ethernet was using a shared coaxial cable.
Computers were connected to this one, long cable with **vampire taps** ☺.



CSMA/CA (Collision Avoidance)

- ❖ Collision detection is easy in wired LANs, but difficult in wireless LANs. For example,



- ❖ 802.11 (Wi-Fi) uses **CSMA/CA** protocol instead.
 - Receiver needs to return ACK if a frame is received OK.

Lecture 8: Summary

❖ Channel partitioning

- Divide channel by time, used in GSM
- Divide channel by frequency, commonly used in radio, satellite systems

❖ Taking turns

- polling from central site, used in Bluetooth
- token passing, used in FDDI and token ring

❖ Random access

- CSMA/CD used in Ethernet
- CSMA/CA used in 802.11 Wi-Fi

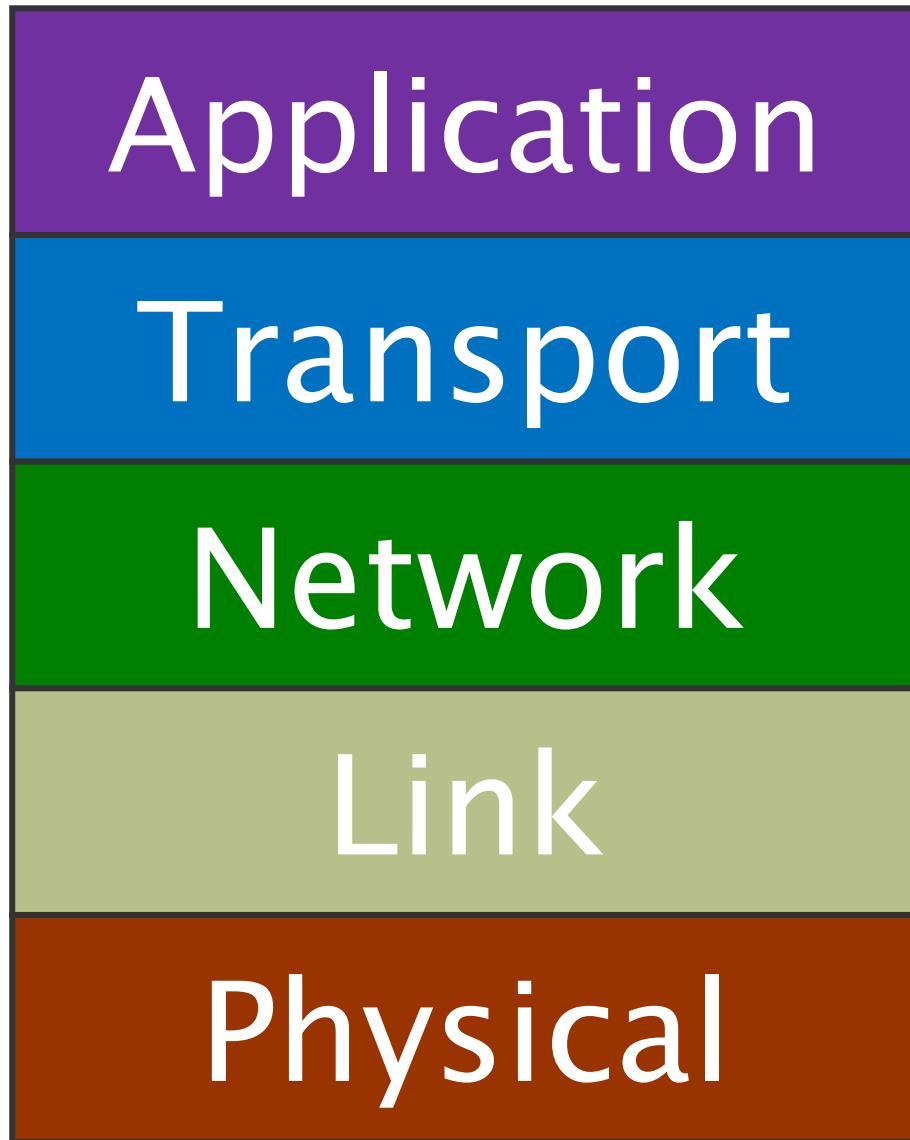
CS2105

An Awesome Introduction to Computer Networks

Lecture 9: The Link Layer, Part II



Department of Computer Science
School of Computing



You are
here

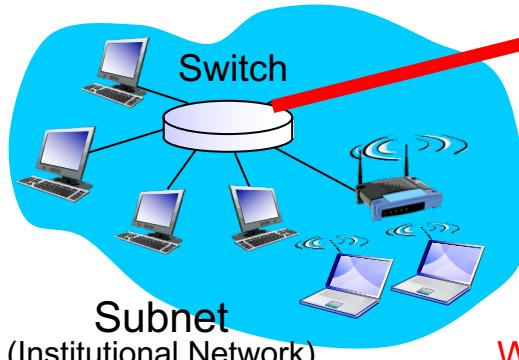
Lecture 9: The Link Layer

After this class, you are expected to understand:

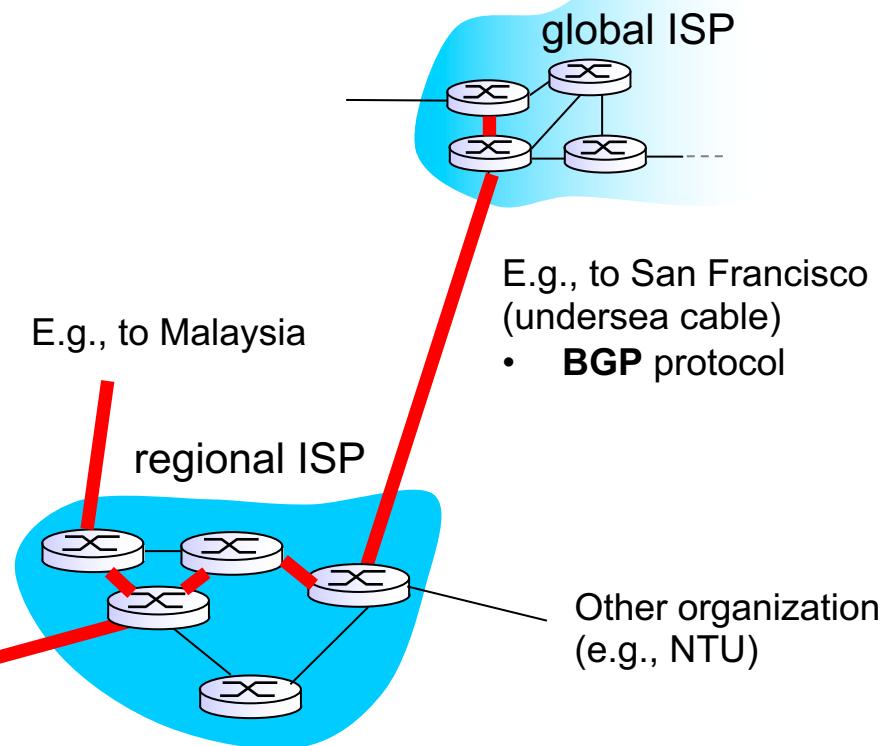
- ❖ the role of link layer and the services it could provide.
- ❖ how parity and CRC scheme work.
- ❖ different methods for accessing shared medium.
- ❖ how ARP allows a host to discover the MAC addresses of other nodes in the same subnet.
- ❖ the role of switches in interconnecting subnets in a LAN.

L8

Routing: Big Picture



Within subnet
• ARP protocol



Which link/path to choose?

Intra-AS routing

- RIP, OSPF protocols
- Distributed algo.
- Build routing table

Lecture 8&9: Roadmap

6.1 Introduction to the Link Layer

6.2 Error Detection and Correction

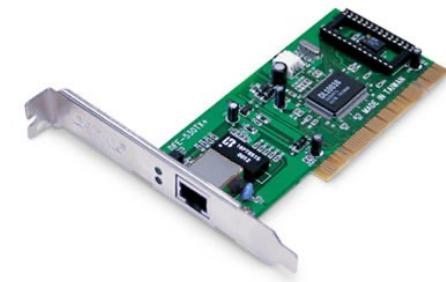
6.3 Multiple Access Links and Protocols

6.4 Switched Local Area Networks

- 6.4.1 Link Layer Addressing & ARP
- 6.4.2 Ethernet
- 6.4.3 Link-layer Switches

MAC Address (1/2)

- ❖ Every adapter (NIC) has a **MAC address** (aka physical or LAN address).
 - Used to send and receive link layer frames.
 - When an adapter receives a frame, it checks if the destination MAC address of the frame matches its own MAC address.
 - If **yes**, adapter extracts the enclosed datagram and passes it to the protocol stack.
 - If **no**, adapter simply discards the frame without interrupting the host.



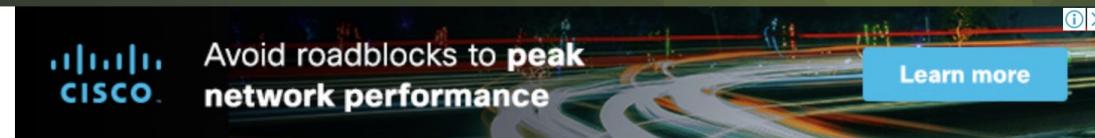
MAC Address (2/2)

- ❖ MAC address is typically 48 bits, burned in NIC ROM (sometimes software settable).
 - Example: **5C-F9-DD-E8-E3-D2** — hexadecimal
(base 16) notation
 - MAC address allocation is administered by IEEE.
 - The **first three bytes** identifies the vendor of an adapter.
 - Several websites allow us to check the vendor given a MAC address, e.g.:

<https://macvendors.com/>

MAC Address (Ex: 1/3)

The screenshot shows the homepage of macvendors.com. At the top, there's a navigation bar with a lock icon, the URL 'macvendors.com', and a star icon. Below the navigation is a search bar containing the text 'MA:CV:en:do:rs'. To the right of the search bar are links for 'Home', 'API', 'Plans', 'About', 'Register', and 'Login'. The main content area features a large green polygonal background with the text 'Find MAC Address Vendors. Now.' in white. Below this is a search input field with placeholder text 'Enter a MAC Address' and a value '00:00:00:00:00:00'.



// Features



Data

Our list of vendors is provided directly from the IEEE Standards Association and is updated multiple times each day. The IEEE is the registration authority and provides us data on over 16,500 registered vendors.



Speed

Our API was designed from the ground up with performance in mind. We have stripped our API down to the bare essentials, optimized our servers, and organized our data so that whether your app is making 100 requests a day, or 100,000, you'll never be left waiting.



Simple

We have eliminated all unnecessary overhead from our systems. Simply send us an HTTP GET/POST request with your MAC address and we'll return the



Reliable

We want you to feel comfortable building your systems around ours. Since launching in 2011, we have grown at an incredible pace. Today our API receives

MAC Address (Ex: 2/3)

macvendors.com

MA:CV:en:do:rs

Home API Plans About Register Login

Find MAC Address Vendors. Now.

Enter a MAC Address

5C:F9:DD:E8:E3:D2

Dell Inc.



// Features



Data

Our list of vendors is provided directly from the IEEE Standards Association and is updated multiple times each day. The IEEE is the registration authority and provides us data on over 16,500 registered vendors.



Speed

Our API was designed from the ground up with performance in mind. We have stripped our API down to the bare essentials, optimized our servers, and organized our data so that whether your app is making 100 requests a day, or 100,000, you'll never be left waiting.



Simple

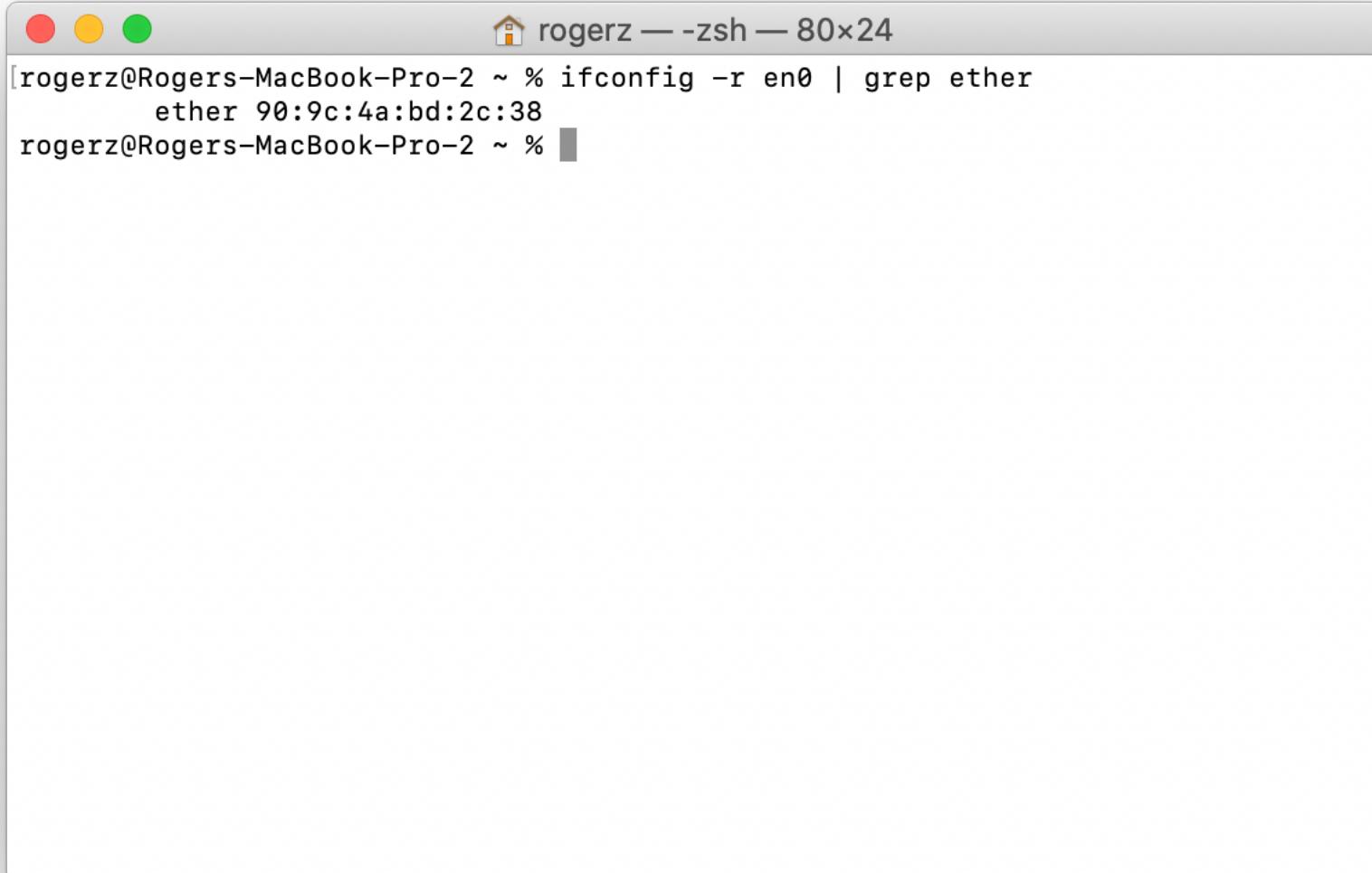
We have eliminated all unnecessary overhead from our systems. Simply send us an HTTP GET/POST request with your MAC address and we'll return the



Reliable

We want you to feel comfortable building your systems around ours. Since launching in 2011, we have grown at an incredible pace. Today our API receives

MAC Address (Ex: 3/3)

A screenshot of a macOS terminal window titled "rogerz — -zsh — 80x24". The window shows the command "ifconfig -r en0 | grep ether" being run, which outputs the MAC address "ether 90:9c:4a:bd:2c:38".

```
[rogerz@Rogers-MacBook-Pro-2 ~ % ifconfig -r en0 | grep ether
      ether 90:9c:4a:bd:2c:38
rogerz@Rogers-MacBook-Pro-2 ~ % ]
```

IP Address vs. MAC Address

❖ IP address

- 32 bits in length
- network-layer address used to move **datagrams** from source to dest.
- **Dynamically assigned;** hierarchical (to facilitate routing)
- **Analogy:** postal address

❖ MAC address

- 48 bits in length
- link-layer address used to move **frames** over every single link.
- Permanent, to identify the hardware (adapter)
- **Analogy:** NRIC number

ARP: Address Resolution Protocol

- ❖ **Question:** How to know the MAC address of a receiving host, knowing its IP address?
 - Use ARP [RFC 826] most protocols have 2 mechanisms
 - 1) lookup function to search
 - 2) way to optimise to ensure that the search is faster
- ❖ Each IP node (host, router) has an **ARP table**.
 - Stores the mappings of IP address and MAC address of other nodes in the same subnet.

< IP address; MAC address;  TL >

time after which address mapping will be forgotten (typically a few minutes on Windows)

As Maverick (Top Gun) likes to say:



ARP Demo: Office Windows PC

```
Command Prompt
Microsoft Windows [Version 10.0.18363.752]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\dcsrz>arp -a

Interface: 172.26.184.178 --- 0xe
Internet Address      Physical Address      Type
 172.26.184.1          00-00-0c-07-ac-00    dynamic
 172.26.184.52         b8-ca-3a-b5-15-09    dynamic
 172.26.184.128        6c-2b-59-d4-6e-4e    dynamic
 172.26.184.135        8c-ec-4b-b1-96-96    dynamic
 172.26.184.139        8c-ec-4b-b0-b8-13    dynamic
 172.26.184.179        00-50-04-05-23-64    dynamic
 172.26.184.203        8c-ec-4b-b1-97-6e    dynamic
 172.26.184.210        00-50-b6-5c-92-1b    dynamic
 172.26.184.230        8c-ec-4b-b0-b8-19    dynamic
 172.26.184.231        8c-ec-4b-b2-4b-4f    dynamic
 172.26.184.255        ff-ff-ff-ff-ff-ff    static
 224.0.0.22              01-00-5e-00-00-16    static
 224.0.0.251             01-00-5e-00-00-fb    static
 224.0.0.252             01-00-5e-00-00-fc    static
 239.255.255.250        01-00-5e-7f-ff-fa    static
 255.255.255.255        ff-ff-ff-ff-ff-ff    static

C:\Users\dcsrz>
```

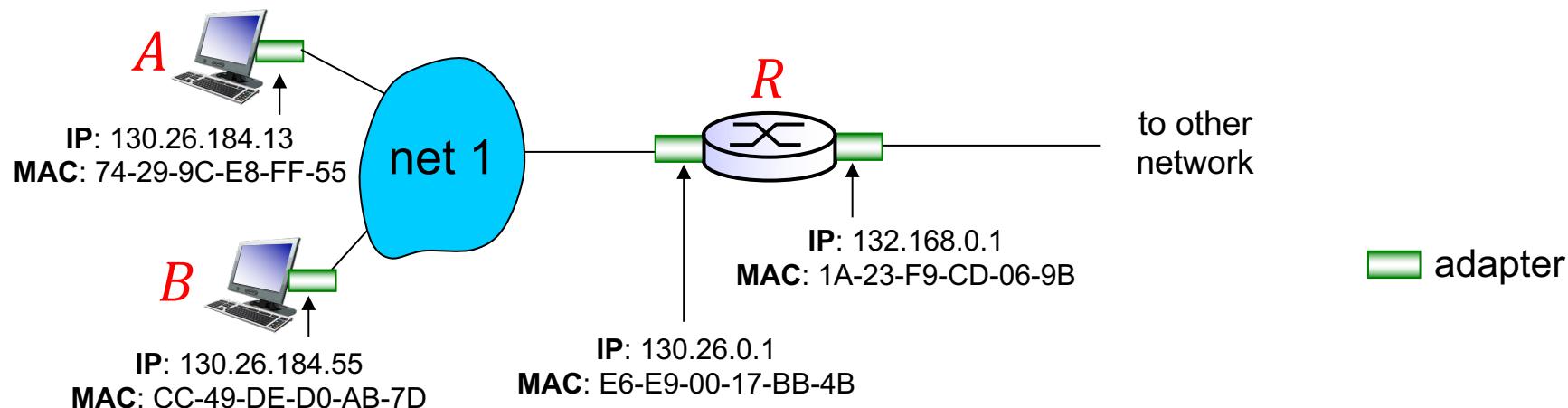
Sending Frame in the Same Subnet

- ❖ Suppose *A* wants to send data to *B*. They are in the same subnet.

① If *A* knows *B*'s MAC address from its ARP table

- create a frame with *B*'s MAC addresses and send it.
- Only *B* will process this frame.
- Other nodes may receive but will ignore this frame.

② What if *A* is not aware of *B*'s MAC address?



Sending Frame in the Same Subnet

- ❖ What if B 's MAC address is not in A 's ARP table?

① A broadcasts an ARP query packet, containing B 's IP address.

- Dest MAC address set to FF-FF-FF-FF-FF-FF
- All the other nodes in the same subnet will receive this ARP query packet, but only B will reply it.

②  B replies to A with its MAC address.

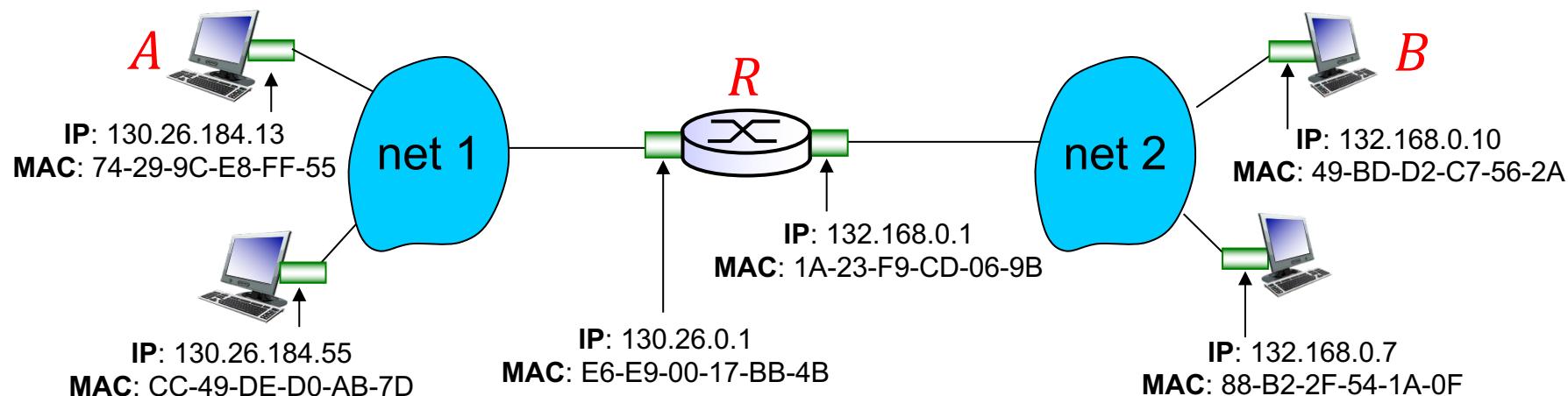
- Reply frame is sent to A 's MAC address.

③ A caches B 's IP-to-MAC address mapping in its ARP table (until TTL expires).

Question: how to determine if B is in the same subnet?

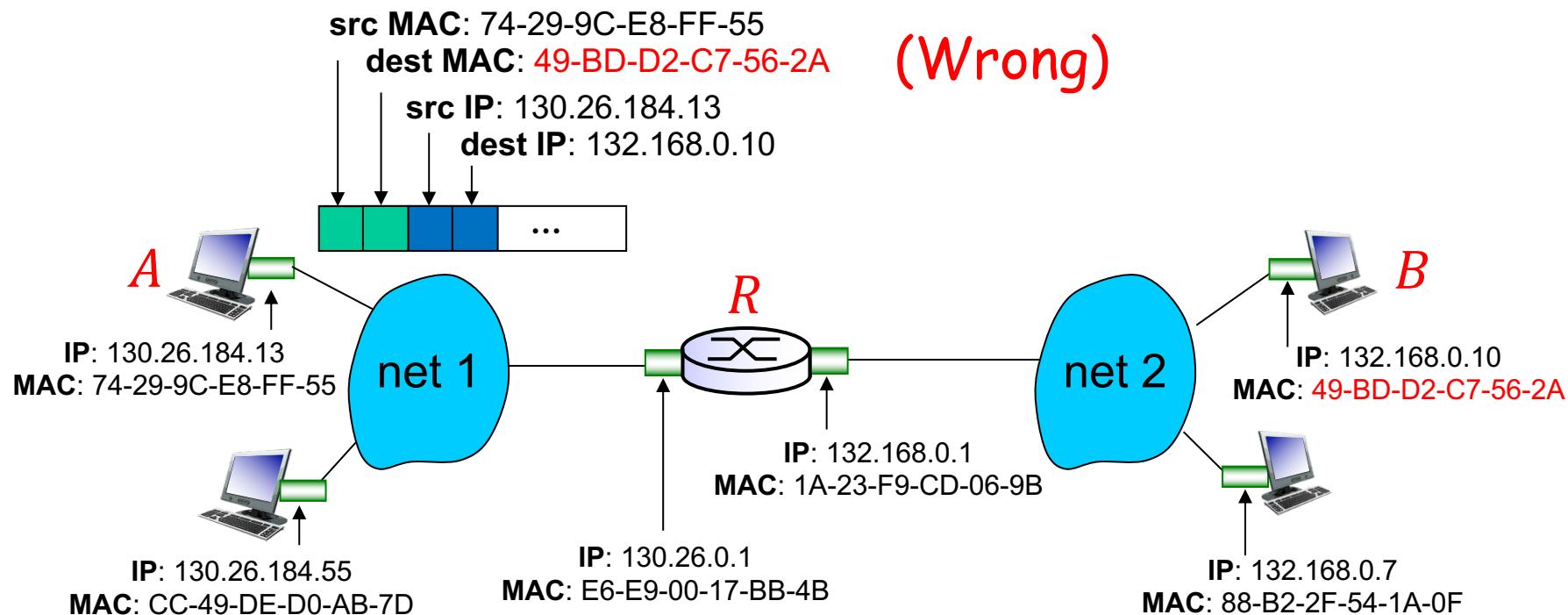
Sending Frame to Another Subnet

- ❖ **Question:** What if we send data to a host in another subnet?
 - For example, *A* sends datagram to *B* in another subnet.



Sending Frame to Another Subnet

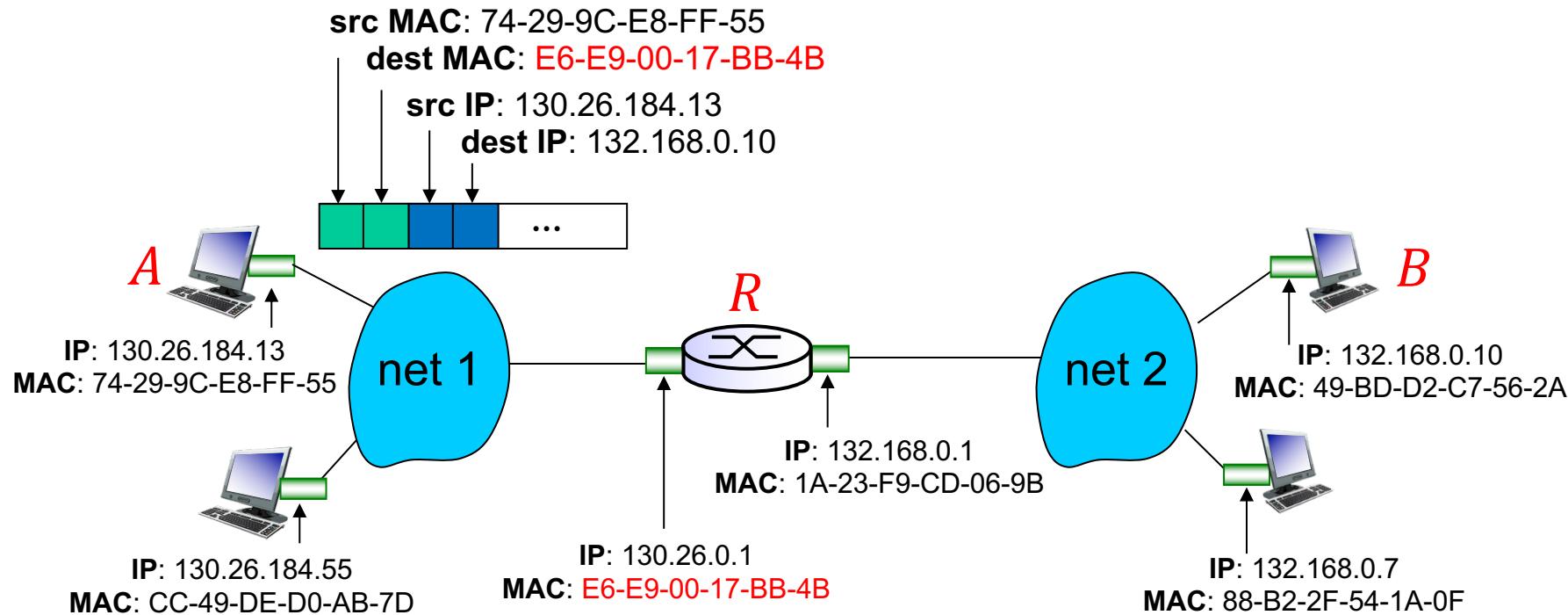
- ❖ A sends datagram to B in another subnet.
 - Can A create a frame as follows?
 - No. all adapters in net 1 will ignore this frame because of the mismatch of destination MAC address.



Sending Frame to Another Subnet

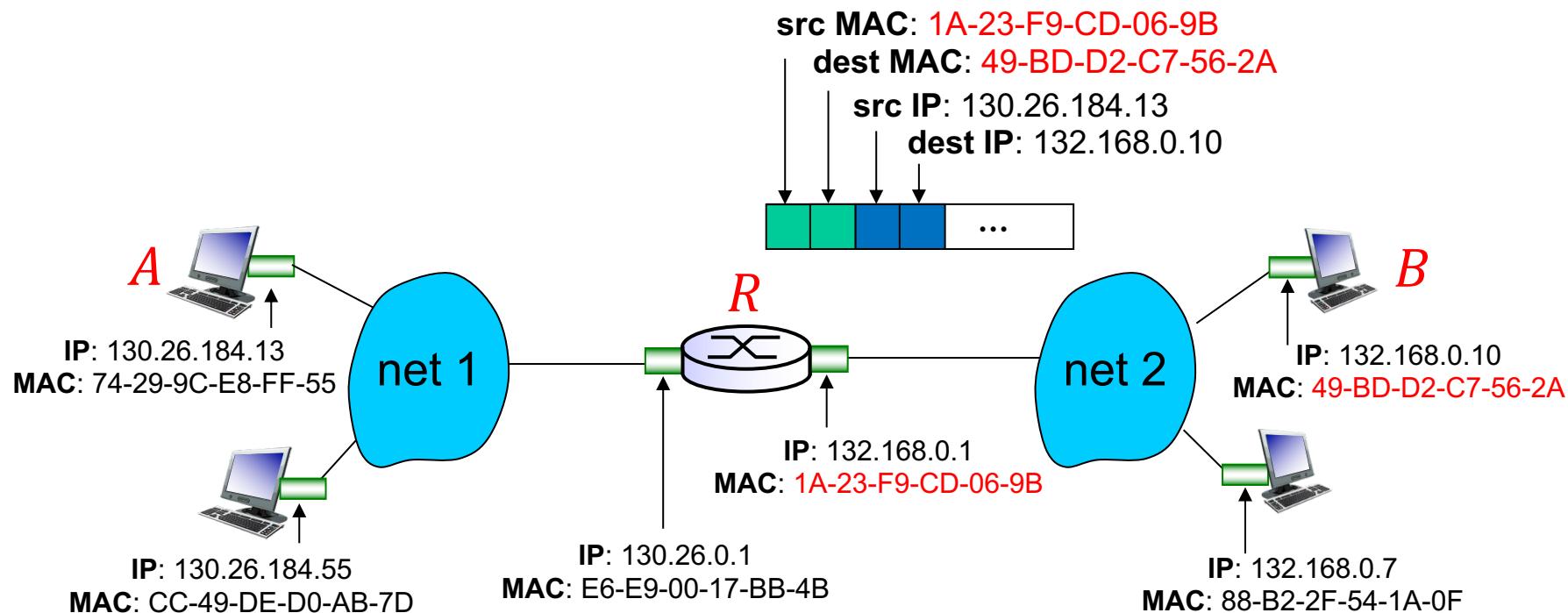
- ❖ A sends datagram to B in another subnet.

- I should create a link-layer frame with (1) R's MAC address (2) B's IP address as destination.



Sending Frame to Another Subnet

- ❖ A sends datagram to B in another subnet.
 - R will move datagram to outgoing link and construct a new frame with B's MAC address.



Lectures 8&9: Roadmap

6.1 Introduction to the Link Layer

6.2 Error Detection and Correction

6.3 Multiple Access Links and Protocols

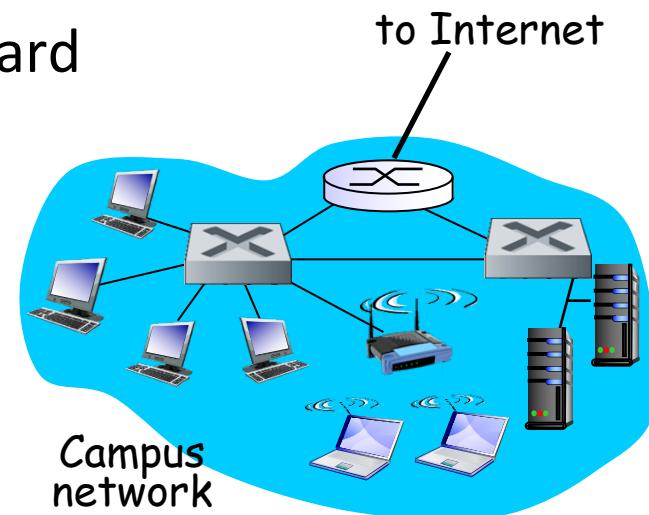
6.4 Switched Local Area Networks

- 6.4.1 Link Layer Addressing & ARP
- 6.4.2 Ethernet
- 6.4.3 Link-layer Switches

Local Area Network (LAN)

- ❖ LAN is a computer network that interconnects computers within a geographical area such as office building or university campus.

LAN can contain multiple subnets



- ❖ LAN technologies:

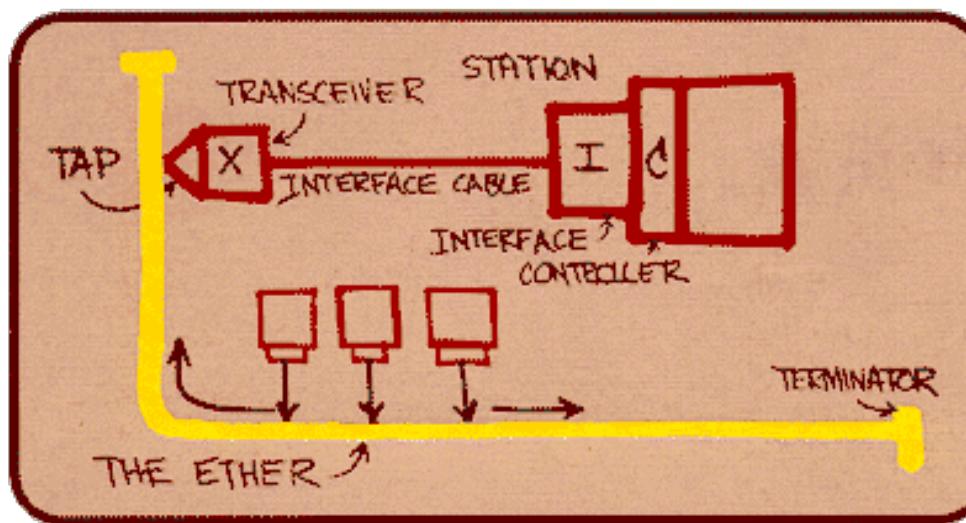
- IBM Token Ring: IEEE 802.5 standard
- Ethernet: IEEE 802.3 standard
- Wi-Fi: IEEE 802.11 standard
- Others

Ethernet

- ❖ “Dominant” wired LAN technology:
 - Developed in mid 1970s
 - Standardized by Xerox, DEC, and Intel in 1978
 - Simpler and cheaper than token ring and ATM



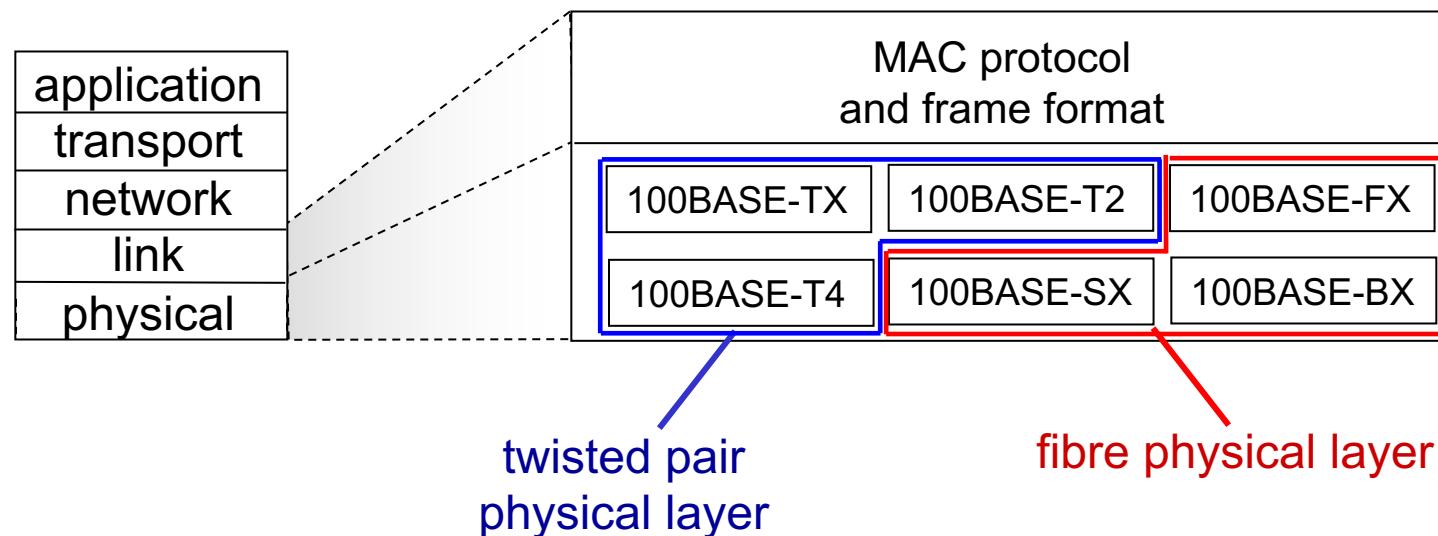
Ethernet connection
(Source: Wikipedia)



*Metcalfe's
Ethernet sketch*

802.3 Ethernet Standards (1/2)

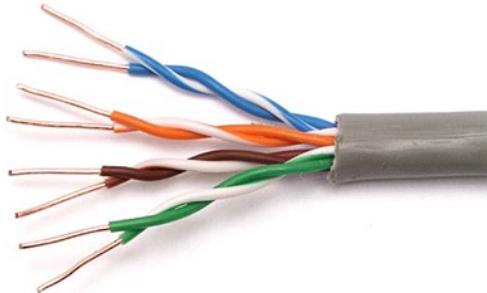
- ❖ A series of Ethernet standards have been developed over the years.
 - Different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps, 100 Gbps
 - Different physical layer media: cable, fiber optics
 - **MAC protocol** and **frame format** remain unchanged



802.3 Ethernet Standards (2/2)

- ❖ Twisted Pair Copper Connectors:

- RJ45
- CAT 6
 - Max. speed: 10 Gbps
 - Max. length: 100m



- ❖ Optical Fibre Connectors:

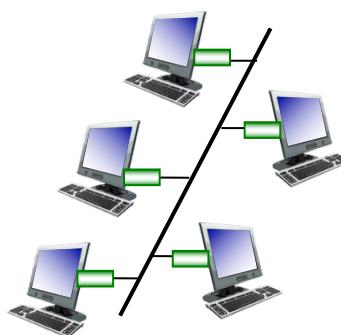
- Left: LC/PC connectors
Right: SC/PC connectors
- Single-mode fibre
 - Max. speed: 10 or 40 Gbps
 - Max. length: > 80 km



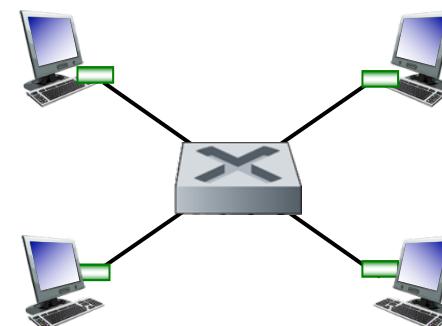
Source: Wikipedia

Ethernet: Physical Topology

- ❖ **Bus topology:** popular in mid 90s
 - all nodes can collide with each other
- ❖ **Star topology:** prevails today
 - switch in center
 - nodes do not collide with each other



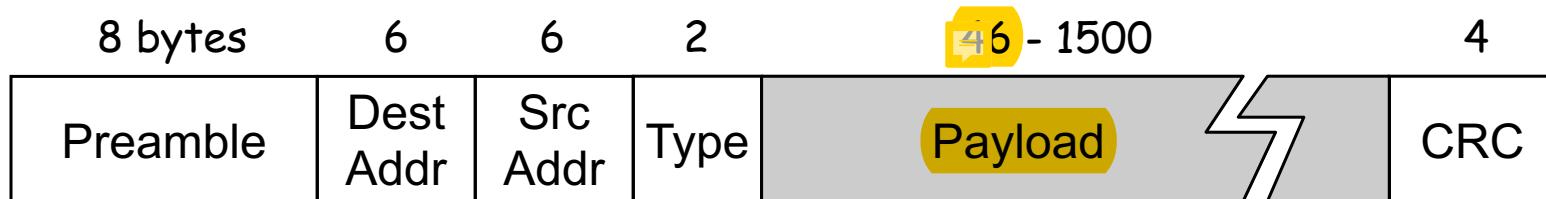
Ethernet with **bus** topology



Ethernet with **star** topology

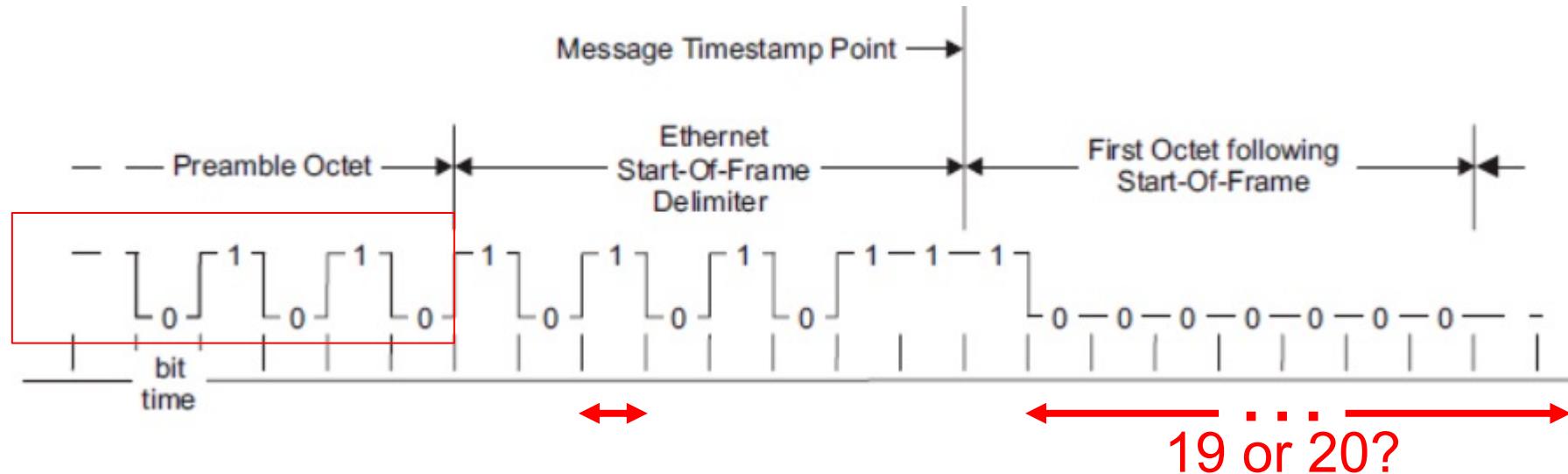
Ethernet Frame Structure (1/3)

- ❖ Sending NIC (adapter) encapsulates IP datagram in Ethernet frame.



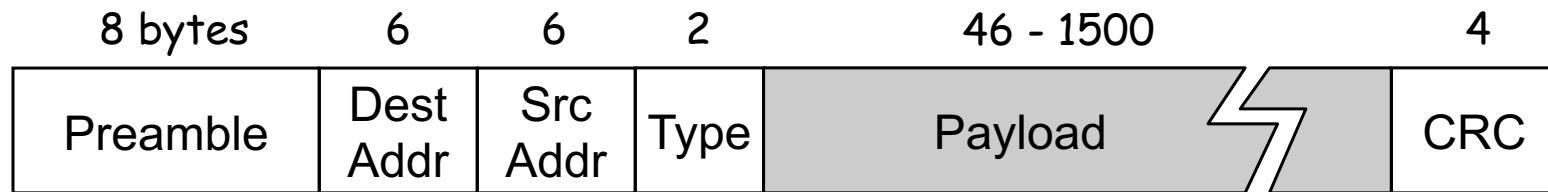
- ❖ *Preamble:*
 - 7 bytes with pattern 10101010 (AA_{Hex}) followed by 1 byte with pattern 10101011 (AB_{Hex}).
 - used to synchronize receiver and sender clock rates.

Ethernet Frame Structure (2/3)



- ❖ The preamble provides a “square wave” pattern that tells the receiver the sender’s clock rate;
- ❖ and it tells the receiver the width of a bit;
- ❖ which is important if there is a long string of bits of the same value, e.g., 19 or 20 zeros.

Ethernet Frame Structure (3/3)



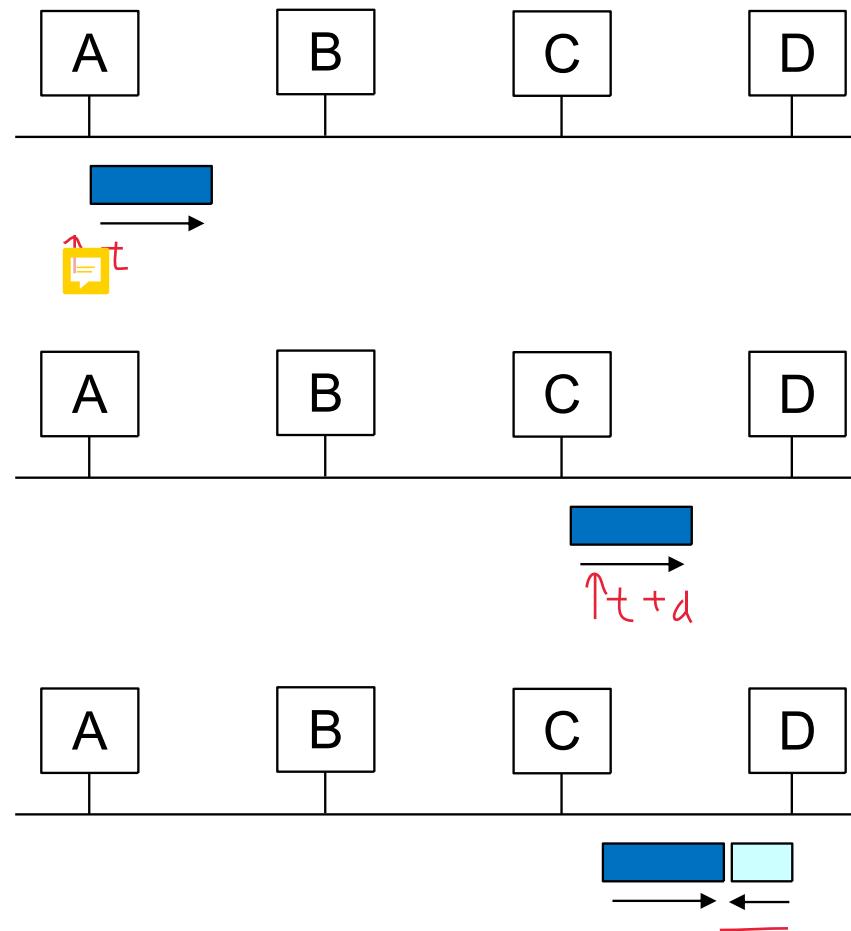
- ❖ *Source and dest MAC address:*
 - If NIC receives a frame with matching destination address, or with broadcast address, it passes data in the frame to network layer protocol.
 - Otherwise, NIC discards frame.
- ❖ *Type:* Indicates higher layer protocol (mostly IP).
- ❖ *CRC:* corrupted frame will be dropped.

Ethernet Data Delivery Service

- ❖ **Connectionless**: no handshaking between sending and receiving NICs.
- ❖ **Unreliable**: receiving NIC doesn't send ACK or NAK to sending NIC.
 - data in dropped frames will be recovered only if initial sender uses higher layer rdt (e.g. TCP); otherwise dropped data is lost.
- ❖ Ethernet's multiple access protocol: CSMA/CD with binary (exponential) backoff.

Collisions in Bus Topology Ethernet

- ❖ Collision may happen in Ethernet of bus topology.
- ❖ For example:
 - A sends a frame at time t .
 - A 's frame reaches D at time $t + d$.
 - D begins transmission at time $t + d - 1$ and collides with A 's frame.



Ethernet CSMA/CD Algorithm

1. NIC receives datagram from network layer, creates frame.
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame!
4. If NIC detects another transmission while transmitting, aborts and sends  jam signal.
5. After aborting, NIC enters binary back-off:
 - after m^{th} collision, NIC chooses K at random from $\{0, 1, 2, \dots, 2^m - 1\}$.
 - NIC waits $K * 512$ bit times, returns to Step 2.

Ethernet CSMA/CD Algorithm

Exponential backoff:

- ❖ After 1st collision: choose K at random from {0, 1}; wait $K * 512$ bit transmission times before retransmission.
- ❖ After 2nd collision: choose K from {0, 1, ..., 2^2-1 }.
- ...
- ❖ After m^{th} collision, choose K at random from {0, 1, ..., 2^m-1 }
- ❖ *Goal*: adapt retransmission attempts to estimated current load
 - More collisions implies heavier load.
 - longer back-off interval with more collisions.

Lectures 8&9: Roadmap

6.1 Introduction to the Link Layer

6.2 Error Detection and Correction

6.3 Multiple Access Links and Protocols

6.4 Switched Local Area Networks

- 6.4.1 Link Layer Addressing & ARP
- 6.4.2 Ethernet
- 6.4.3 Link-layer Switches

Ethernet Switch



hub will send out any information it receives to all connected devices

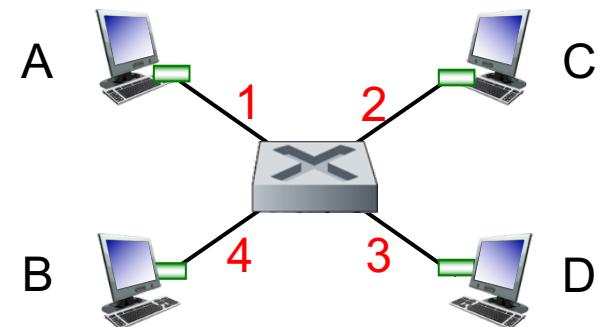
- ❖ A link-layer device used in LAN
 - Store and forward Ethernet frames
 - Examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links.
- ❖ Transparent to hosts
 - No IP address
 - Hosts are unaware of the presence of switches



a 50-port Ethernet switch
(Source: Wikipedia)

Ethernet Switch

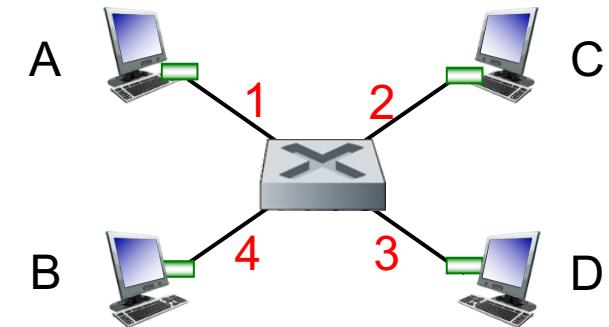
- ❖ In Ethernet of star topology, hosts have dedicated connection to switch.
- ❖ Switch buffers frames and is full duplex.
 - A and D can send frames to each other simultaneously.
- ❖ Ethernet protocol is used on each link, but no collisions!



A switch with 4 interfaces
(1, 2, 3, 4)

Switch Forwarding Table

- ❖ **Q:** how does switch know A is reachable via interface 1, B is reachable via interface 4?



- ❖ **A:** each switch has a **switch table**.

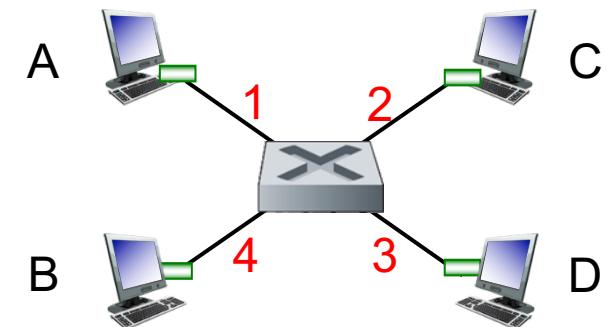
- Format of entry:

< MAC address of host, interface to reach host, TTL >

- ❖ **Q:** how are entries created and maintained in a switch table? can be maintained similarly using TTL

Switch: Self-learning

- ❖ Switch *learns* which hosts can be reached through which interfaces.
 - When receiving a frame from *A*, note down the location of *A* in switch table.
 - If destination *B* is *found* in the table, forward the frame onto that link.
 - If destination *B* is *unknown*, broadcast the frame to all outgoing links.



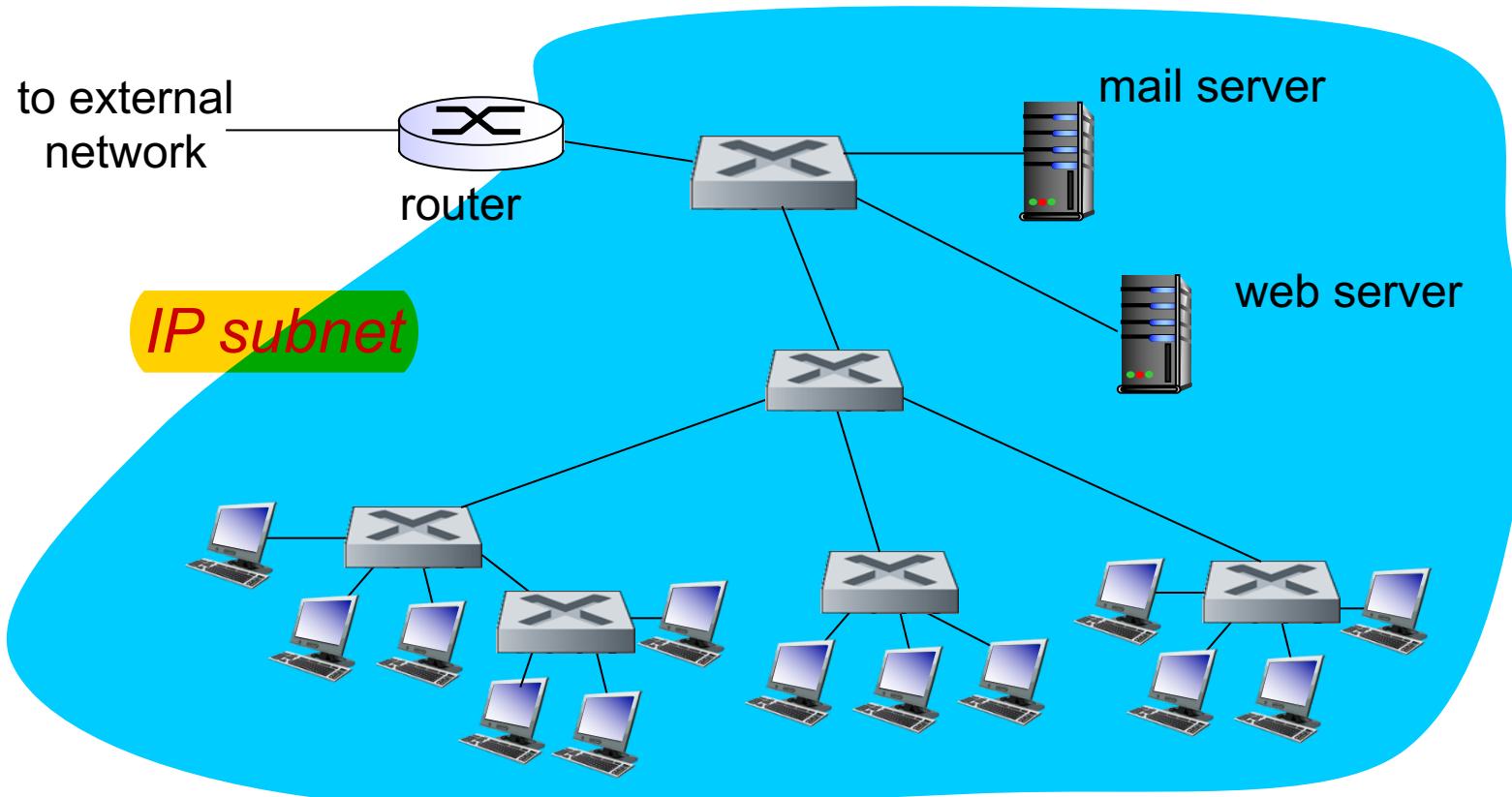
A switch with 4 interfaces
(1, 2, 3, 4)

MAC addr	Interface	TTL
A	1	60

Switch table (initially empty)

Interconnecting Switches

- ❖ Switches can be connected in hierarchy.



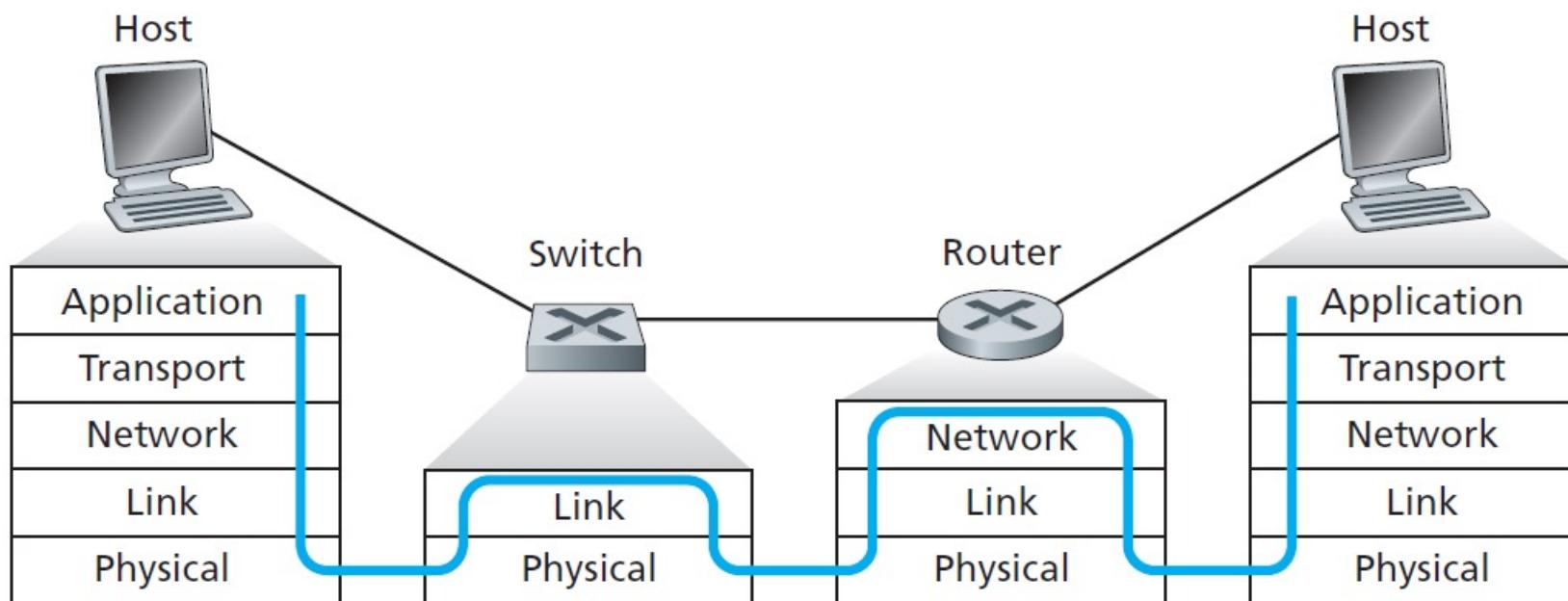
Switches vs. Routers

❖ Routers

- Check IP address
- Store-and-forward
- Compute routes to destination

❖ Switches

- Check MAC address
- Store-and-forward
- Forward frame to outgoing link or broadcast



Lecture 9: Summary

- ❖ **ARP** [RFC 826] resolves the mapping from network layer (IP) address to link layer (MAC) address.
- ❖ Instantiation and implementation of link layer technologies.
 - Ethernet
 - CSMA/CD protocol with binary back-off
 - Ethernet switches and switch tables

CS2105

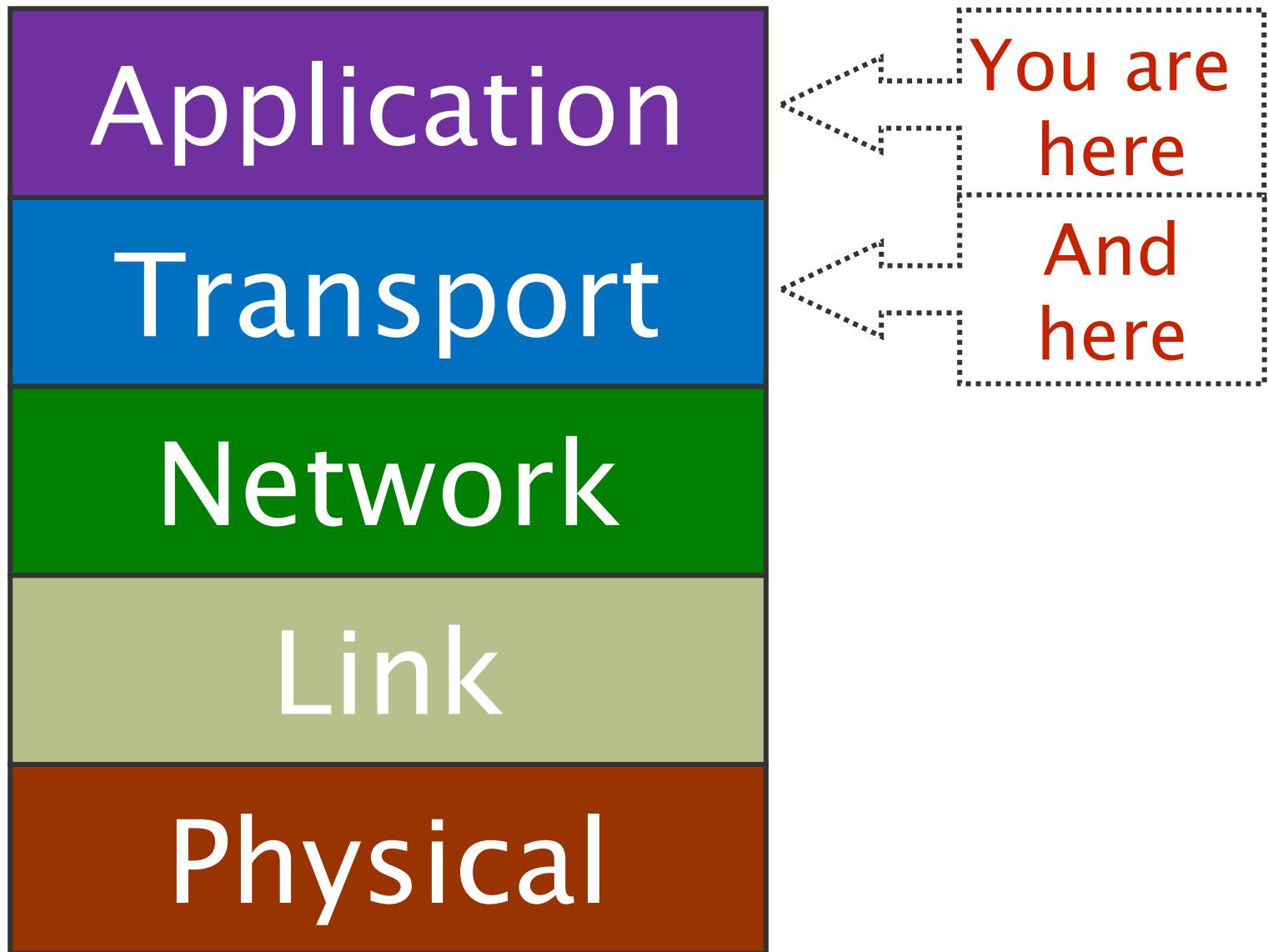
An *Awesome* Introduction to Computer Networks

Lecture 10: Multimedia Networking



Department of Computer Science
School of Computing

Some material copyright 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved



Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming stored video

9.3 voice-over-IP

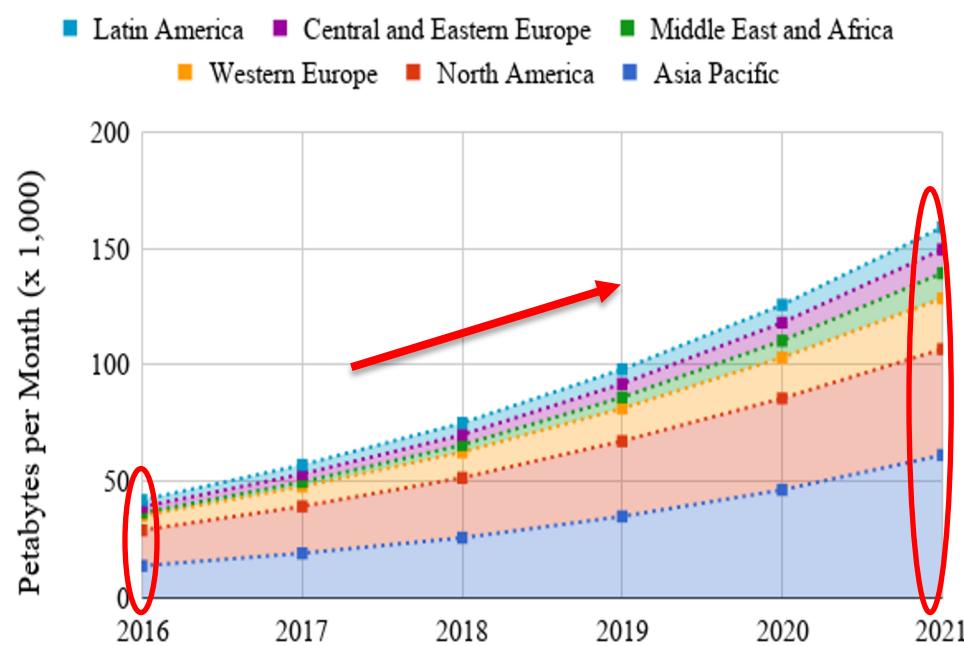
9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

Why learn about multimedia networking?

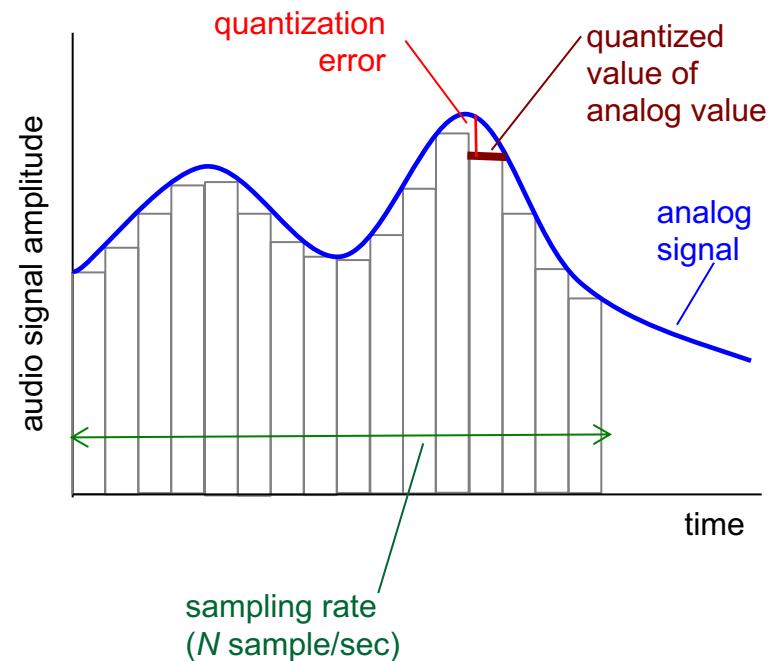
Video is predominant on the Internet!

- Cisco reported in their annual VNI:
 - In 2016, **67%** of the global Internet traffic was video, with a projection to reach **80%** by 2021
- Popular services:
 - YouTube (14.0%)
 - Netflix (34.9%)
 - Amazon Video (2.6%)
 - Hulu (1.4%)
- All these are delivered as  OTT



Multimedia: audio

- analog audio signal sampled at constant rate
 - telephone: 8,000 samples/sec
 - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
 - e.g., $2^8=256$ possible quantized values
 - each quantized value represented by bits, e.g., 8 bits for 256 values
 - $2^{16}=65,536$ for CD



- *Nyquist-Shannon sampling theorem:*
$$f_s \geq d \times 2$$
- f_s : sampling frequency
- d : highest signal frequency

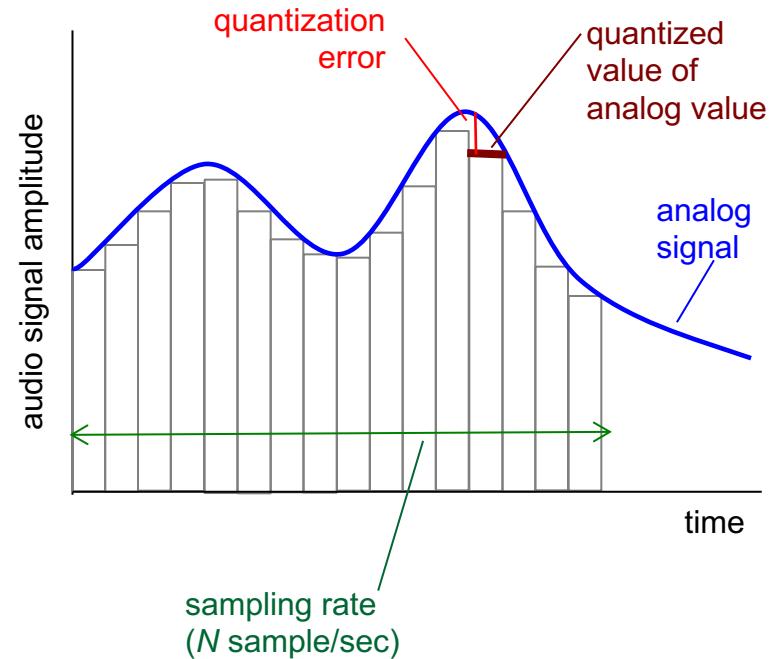
Multimedia: audio

- example: 8,000 samples/sec, 256 quantized values (8 bits): 64,000 bps
- receiver converts bits back to analog signal (DAC):
 - some quality reduction

example rates

- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: 5.3 kbps and up

uncompressed audio and video

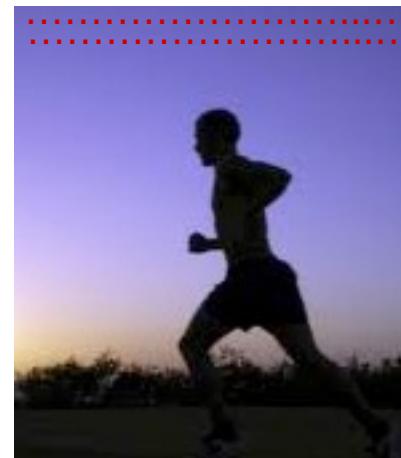


- ADC: analog-to-digital converter
- DAC: digital-to-analog converter

Multimedia: video

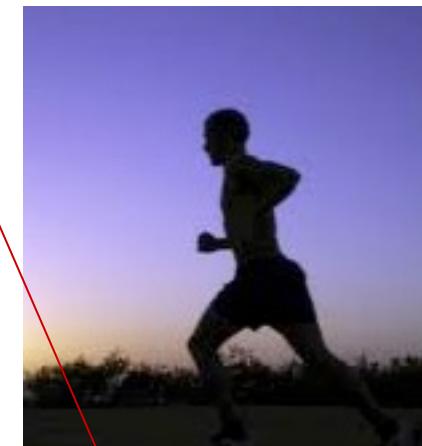
- video: sequence of images displayed at constant rate
 - e.g., 30 images/sec
- digital image: array of pixels
 - **Each pixel represented by bits**
- coding: use redundancy ***within*** and ***between*** images to decrease # bits used to encode image
 - **spatial** (within image)
 - **temporal** (from one image to next)

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



frame i

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

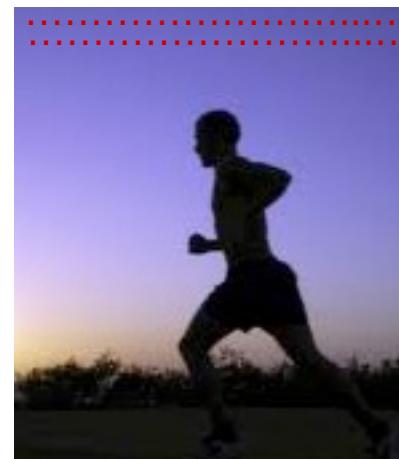


frame $i+1$

Multimedia: video

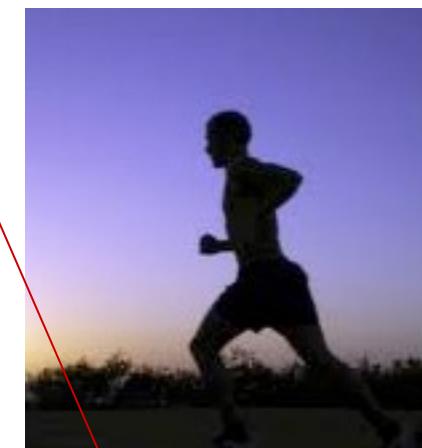
- **CBR: (constant bit rate):**
video encoding rate fixed
- **VBR: (variable bit rate):**
video encoding rate changes
as amount of spatial,
temporal coding changes
- **examples:**
 - MPEG I (CD-ROM) 1.5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4/H.264 (often used in Internet, < 2 Mbps)
 - H.265
 - 4K video < 85 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*

Video: original frame i



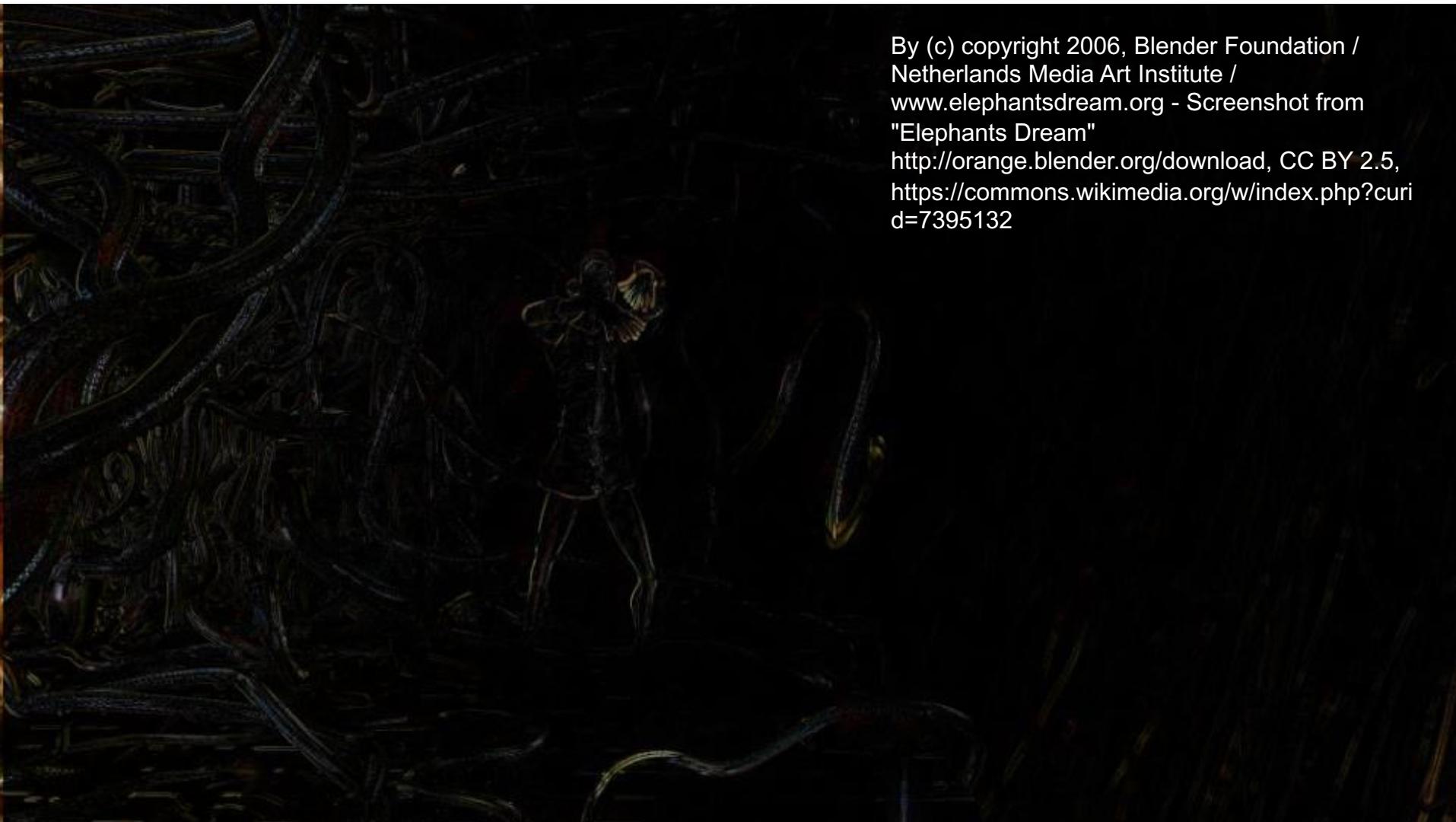
By (c) copyright 2006, Blender Foundation /
Netherlands Media Art Institute /
www.elephantsdream.org - Screenshot from
"Elephant's Dream"
<http://orange.blender.org/download>, CC BY 2.5,
<https://commons.wikimedia.org/w/index.php?curid=7395123>

Difference between 2 frames



By (c) copyright 2006, Blender Foundation /
Netherlands Media Art Institute /
www.elephantsdream.org - Screenshot from
"Elephant's Dream"
<http://orange.blender.org/download>, CC BY 2.5,
<https://commons.wikimedia.org/w/index.php?curid=7395129>

Motion compensated difference



By (c) copyright 2006, Blender Foundation /
Netherlands Media Art Institute /
www.elephantsdream.org - Screenshot from
"Elephant's Dream"
<http://orange.blender.org/download>, CC BY 2.5,
<https://commons.wikimedia.org/w/index.php?curid=7395132>

Multimedia networking: 3 application types

- ***streaming, stored*** audio, video
 - *streaming*: can begin playout before downloading entire file
 - *stored (at server)*: can transmit faster than audio/video will be rendered (implies *storing/buffering at client*)
 - e.g., YouTube, Netflix, Hulu
- ***conversational (“two-way live”)*** voice/video over IP
 - interactive nature of human-to-human conversation limits delay tolerance
 - e.g., Skype, Zoom
- ***streaming live (“one-way live”)*** audio, video
 - e.g., live sporting event (soccer, football)

Multimedia networking: outline

9.1 multimedia networking applications

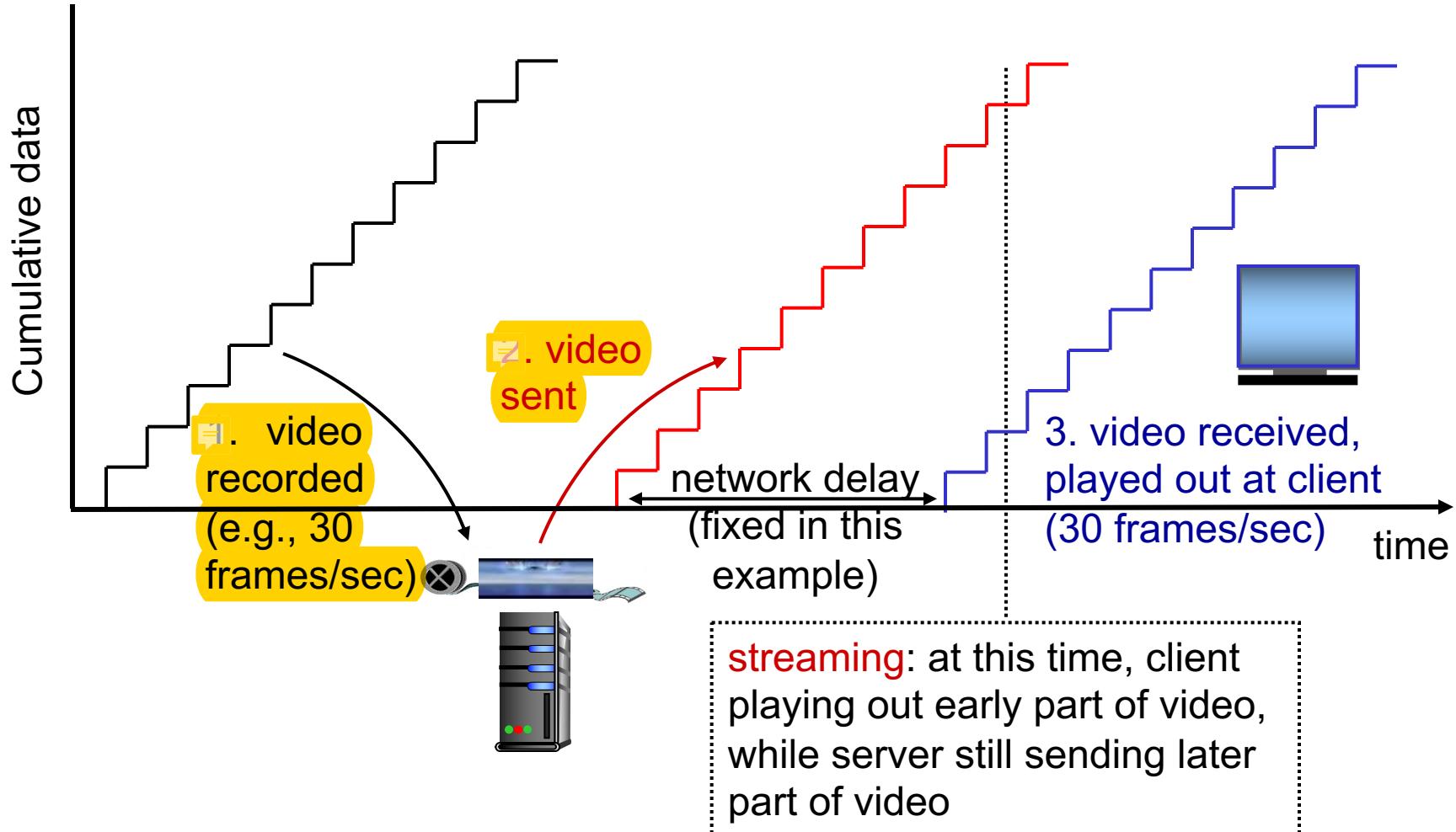
9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

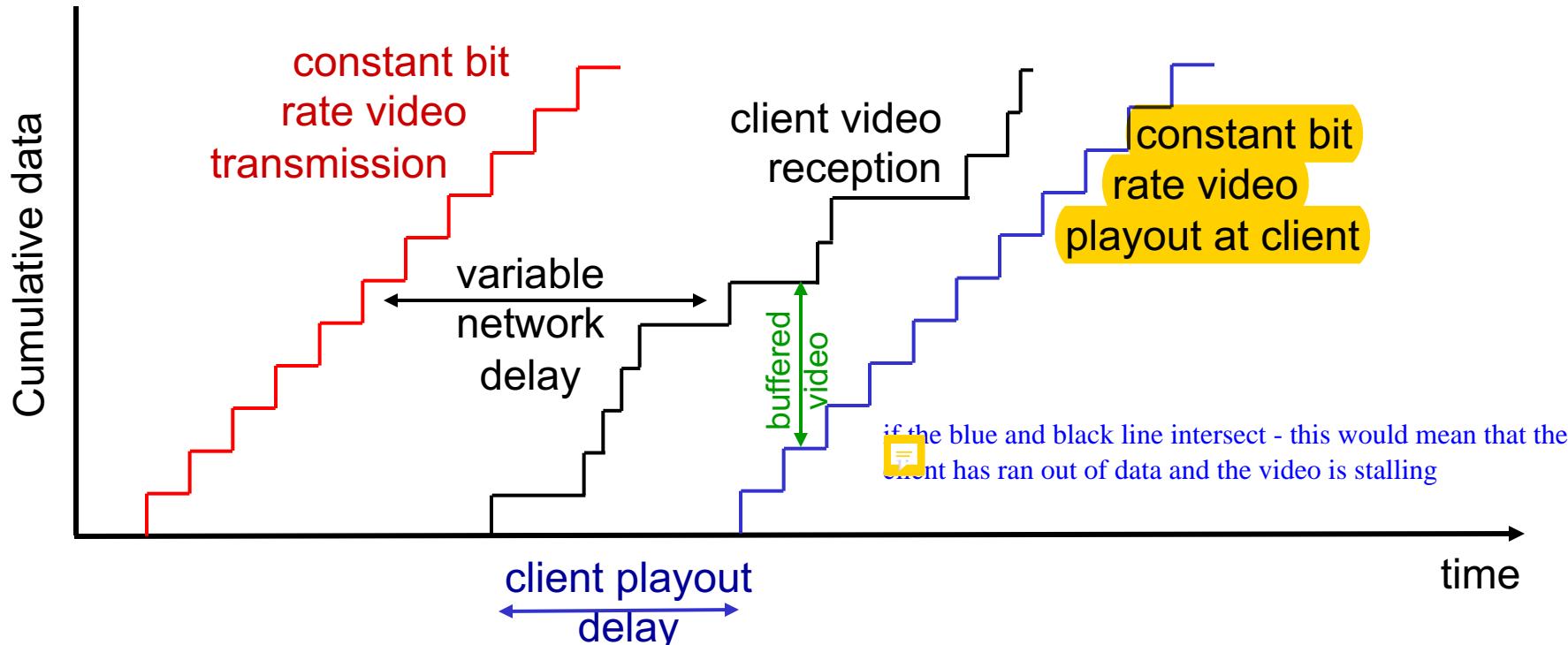
Streaming stored video:



Streaming stored video: challenges

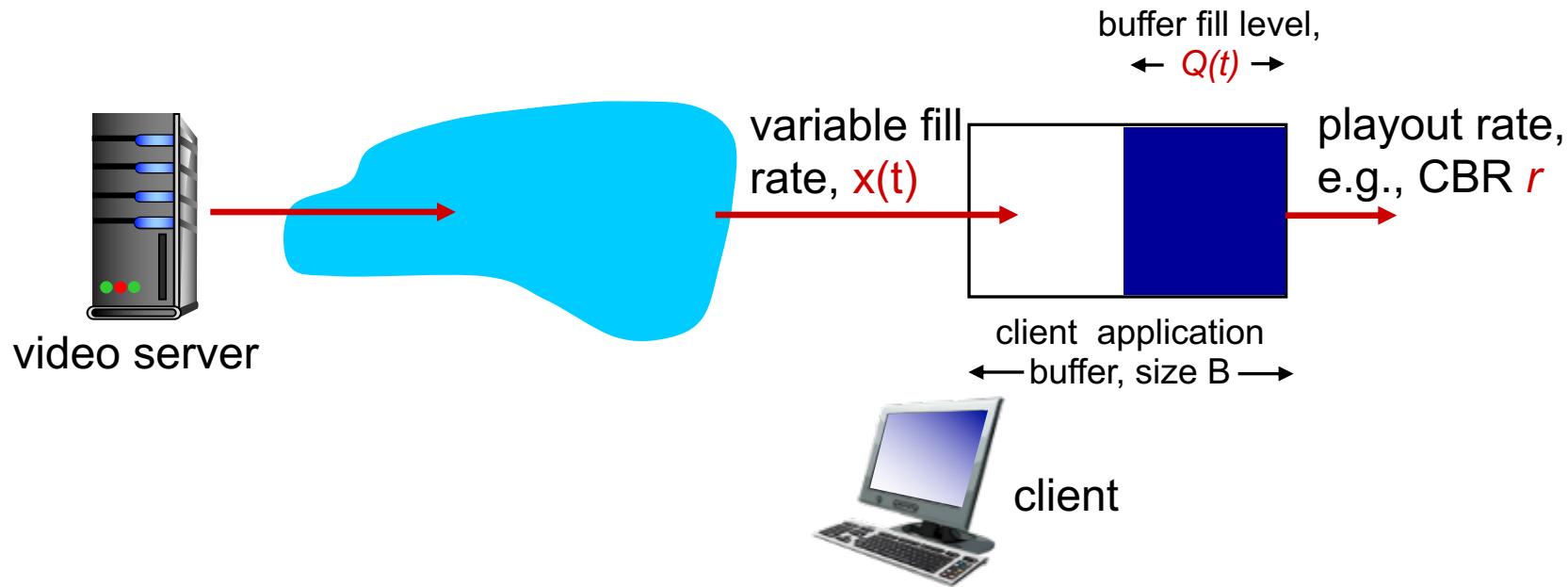
- **continuous playout constraint:** once client playout begins, playback must match original timing
 - ... but **network delays are variable** (jitter), so will need **client-side buffer** to match playout requirements
- other challenges:
 - client interactivity: pause, fast-forward, rewind, jump through video
 - video packets may be lost, retransmitted

Streaming stored video: revisited

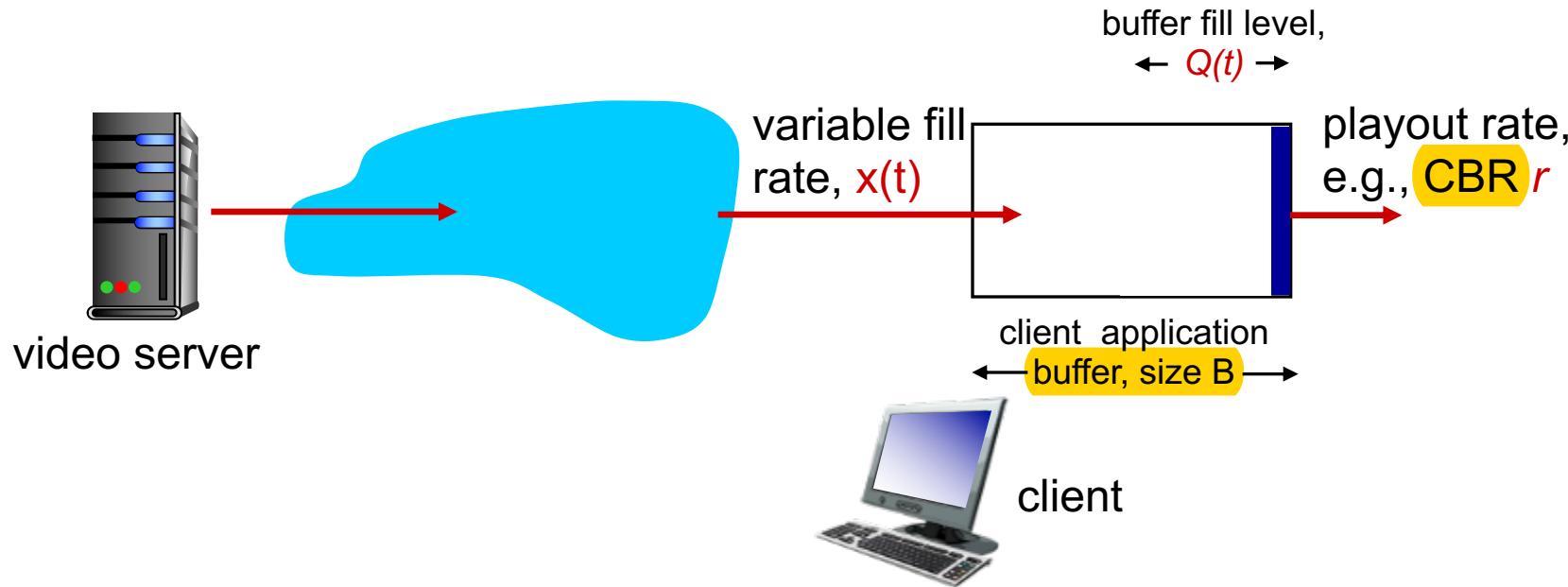


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout

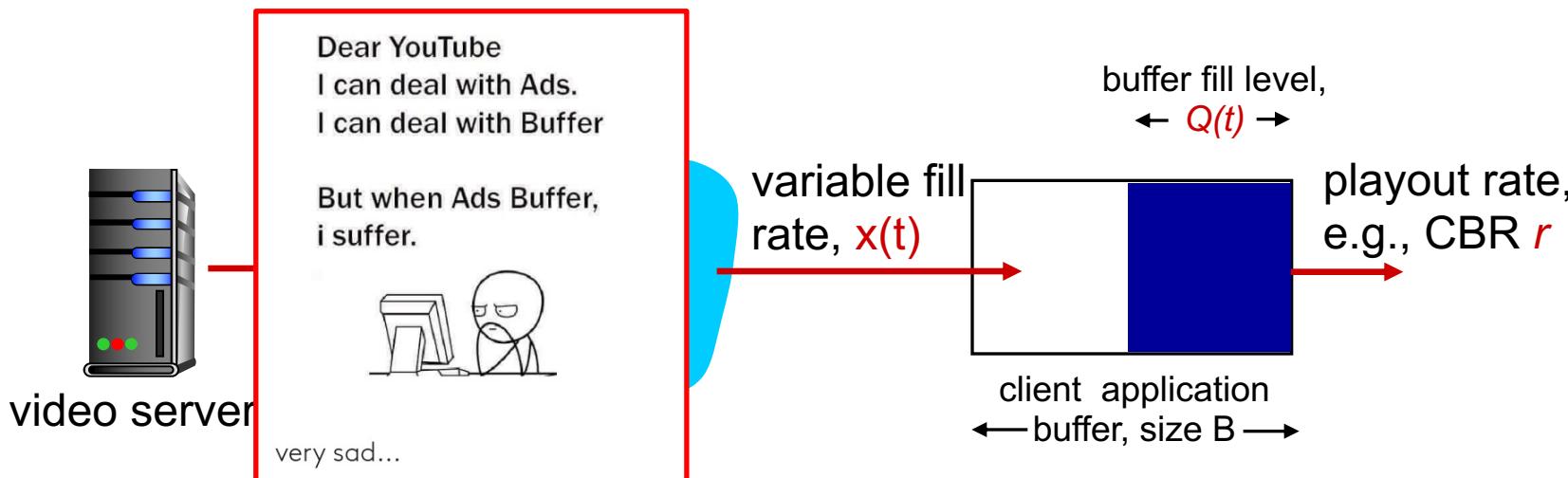


Client-side buffering, playout



1. Initial fill of buffer until playout begins at t_p
2. playout begins at t_p ,
3. buffer fill level varies over time as fill rate $x(t)$ varies and playout rate r is constant

Client-side buffering, playout



playout buffering: average fill rate (\bar{x}), playout rate (r):

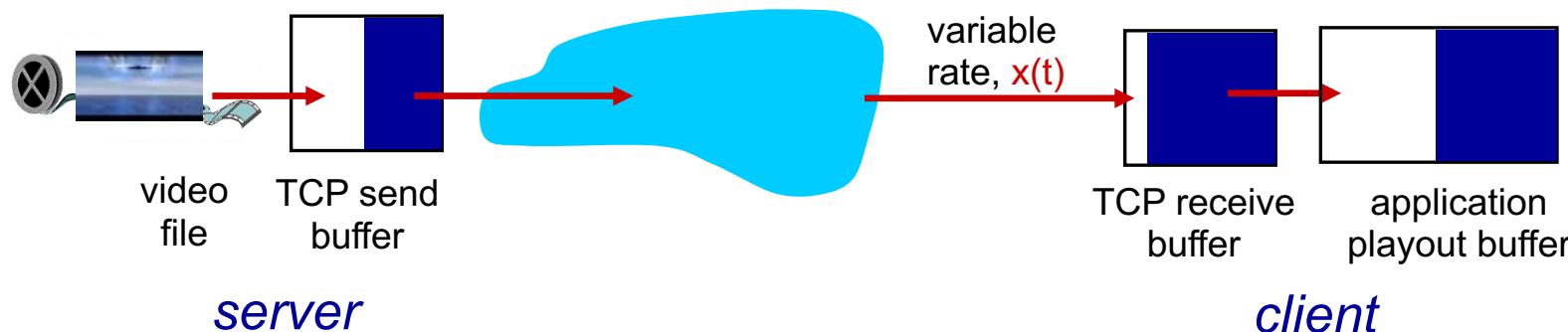
- $\bar{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\bar{x} > r$: buffer will not empty, provided initial playout delay is large enough to **absorb variability** in $x(t)$
 - *initial playout delay tradeoff*: buffer starvation less likely with larger delay, but larger delay until user begins watching

Streaming multimedia: UDP

- server sends at rate appropriate for client
 - often: send rate = encoding rate = constant rate,  **push-based streaming** (*server push*)
 - transmission rate can be oblivious to congestion levels
- short playout delay (2-5 seconds) to remove network jitter
- error recovery: application-level, time permitting
- RTP [RFC 2326]: multimedia payload types
- UDP may *not* go through firewalls

Streaming multimedia: HTTP

- multimedia file retrieved via HTTP GET,
☞ **pull-based streaming (client pull)**
- send at **maximum possible rate under TCP**



- fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery)
- larger playout delay: **smooth TCP delivery rate**
- HTTP/TCP passes more easily through firewalls

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming stored video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

Voice-over-IP (VoIP)

- ***VoIP end-end-delay requirement:*** needed to maintain “conversational” aspect
 - higher delays noticeable, impair interactivity
 - < 150 msec: good
 - > 400 msec bad
 - includes application-level (packetization, playout), network delays
- ***session initialization:*** how does callee advertise IP address, port number, encoding algorithms?
- ***value-added services:*** call forwarding, screening, recording
- ***emergency services:*** 911

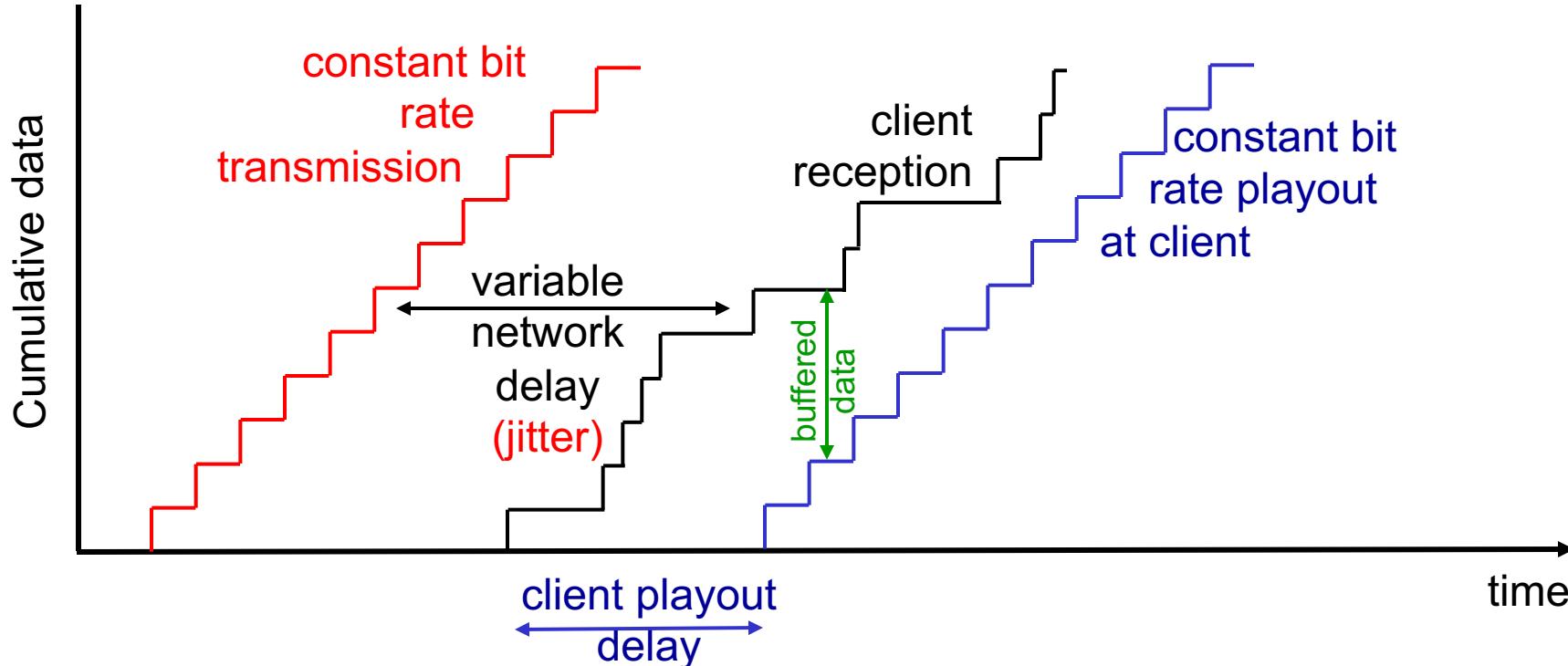
VoIP characteristics

- Speaker's audio: alternating talk spurts, silent periods.
 - 64 kbps during talk spurt
 - pkts generated only during talk spurts
 - 20 msec chunks at 8 Kbytes/sec: 160 bytes of data
- application-layer header added to each chunk
- chunk+header encapsulated into UDP or TCP segment
- application sends segment into socket every 20 msec during talkspurt

VoIP: packet loss, delay

- ***network loss***: IP datagram lost due to network congestion (router buffer overflow)
- ***delay loss***: IP datagram arrives too late for playout at receiver
 - delays: processing, queueing in network; end-system (sender, receiver) delays
 - typical maximum tolerable delay: 400 ms
- ***loss tolerance***: depending on voice encoding, loss concealment, packet loss rates between 1% and 10% can be tolerated

Delay jitter



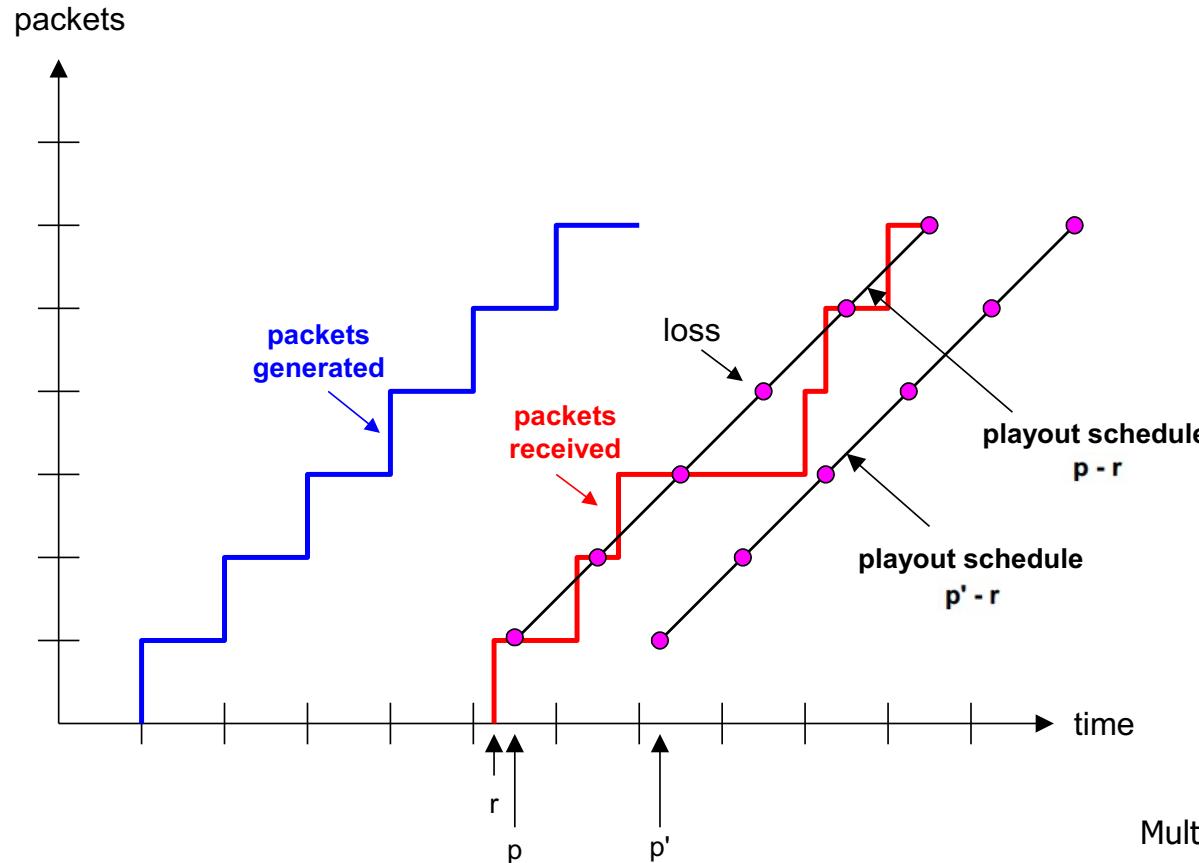
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

VoIP: fixed playout delay

- receiver attempts to playout each chunk exactly q msec after chunk was generated.
 - chunk has time stamp t : play out chunk at $t+q$
 - chunk arrives after $t+q$: data arrives too late for playout: data “lost”
- tradeoff in choosing q :
 - *large q : less packet loss*
 - *small q : better interactive experience*

VoIP: fixed playout delay

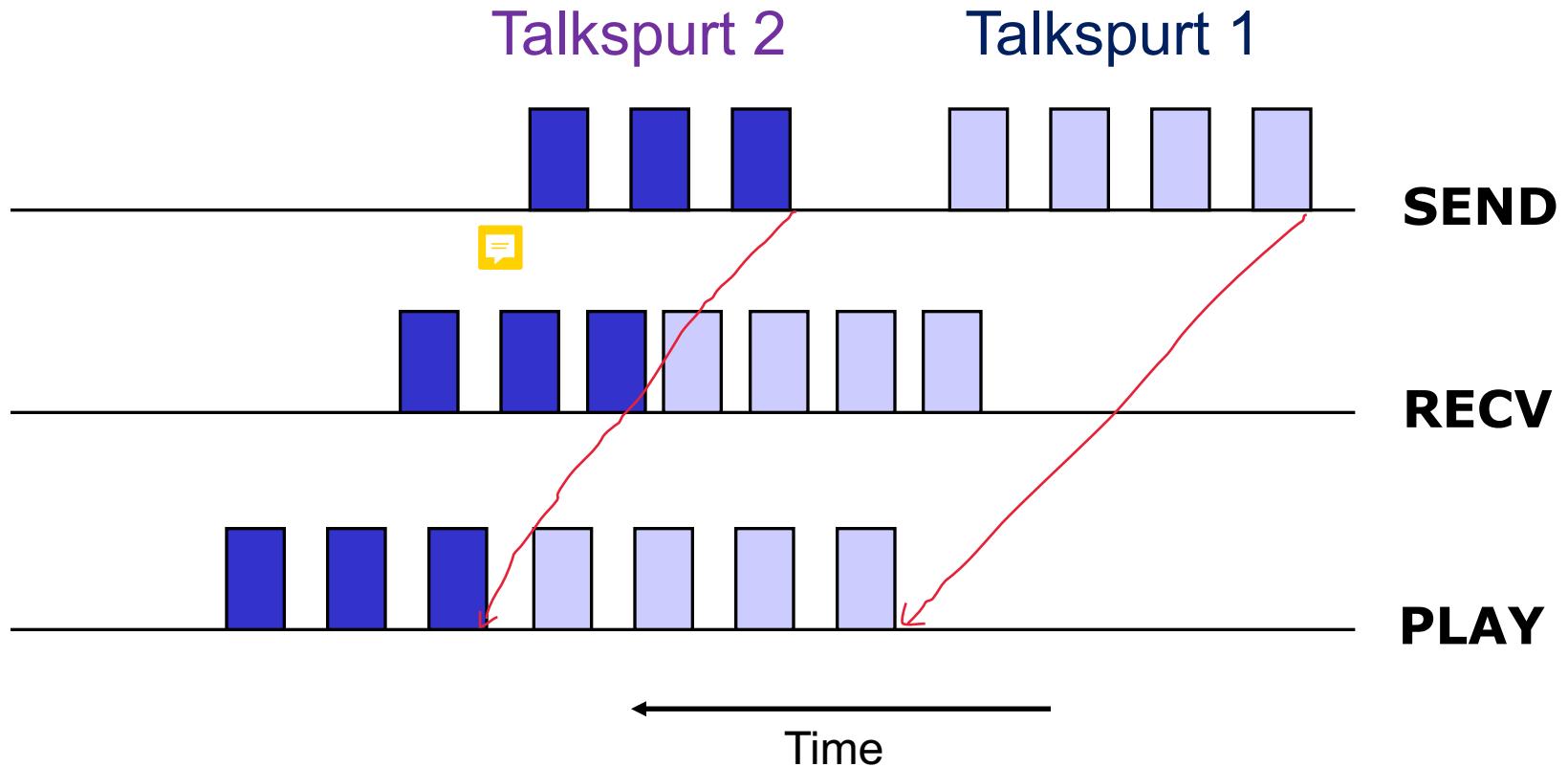
- sender generates packets every 20 msec during talk spurt.
- first packet received at time r
- first playout schedule: begins at p
- second playout schedule: begins at p'



Adaptive playout delay (I)

- **goal:** low playout delay, low late loss rate
 - **approach:** adaptive playout delay adjustment:
 - estimate network delay, adjust playout delay at beginning of each talk spurt
 - silent periods compressed and elongated
 - chunks still played out every 20 msec during talk spurt
 - adaptively estimate packet delay: (EWMA - exponentially weighted moving average, recall TCP RTT estimate):

Adaptive playout delay (2)



Adaptive playout delay (3)

- also useful to estimate average deviation of delay, v_i :

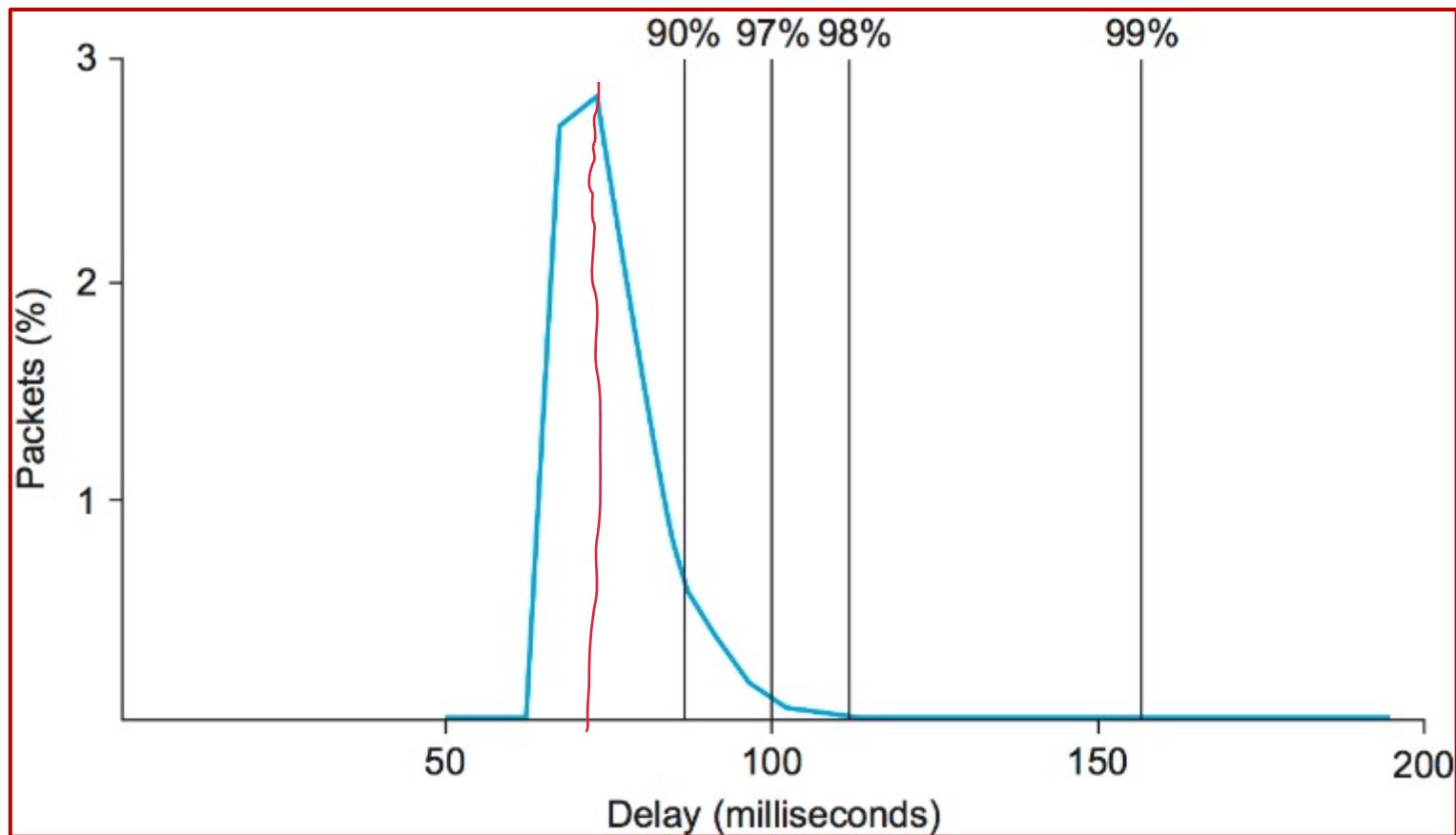
$$v_i = (1-\beta)v_{i-1} + \beta|r_i - t_i - d_i|$$

- estimates d_i , v_i calculated for every received packet, but used only at start of talk spurt
- for first packet in talk spurt, playout time is:

$$\text{playout-time}_i = t_i + d_i + Kv_i$$

- remaining packets in talkspurt are played out periodically

Adaptive playout delay (4)



VoIP: recovery from packet loss (I)

retransmission is not good since it will take up 1 RTT and thus will be too late

Challenge: recover from packet loss given small tolerable delay between original transmission and playout

- each ACK/NAK takes \sim one RTT
- alternative: *Forward Error Correction (FEC)*
 - send enough bits to allow recovery without retransmission (recall two-dimensional parity in Ch. 5)

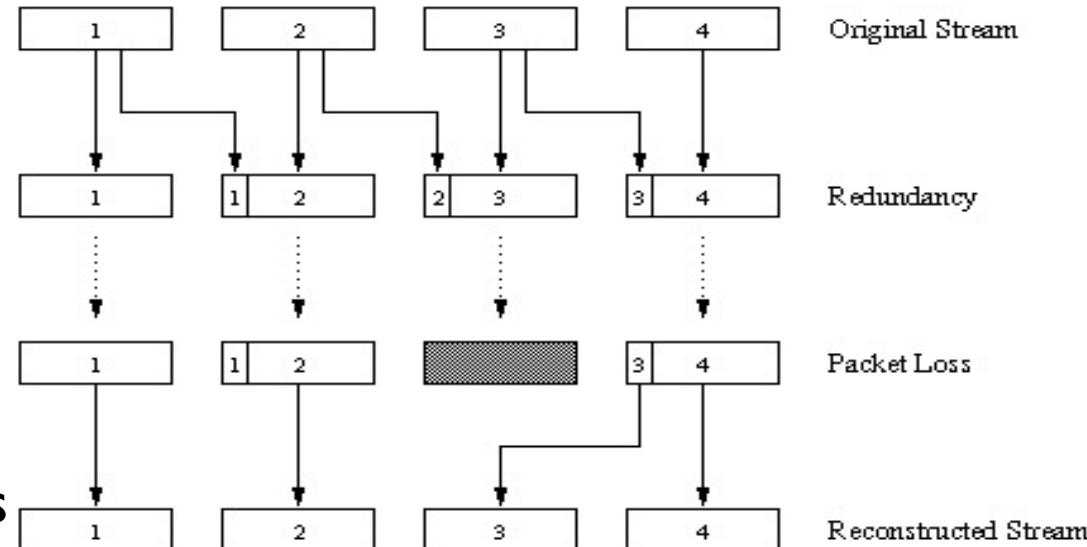
simple FEC

- for every group of n chunks, create redundant chunk by exclusive OR-ing n original chunks
- send $n+1$ chunks, increasing bandwidth by factor $1/n$
- can reconstruct original n chunks if at most one lost chunk from $n+1$ chunks, with playout delay

VoIP: recovery from packet loss (2)

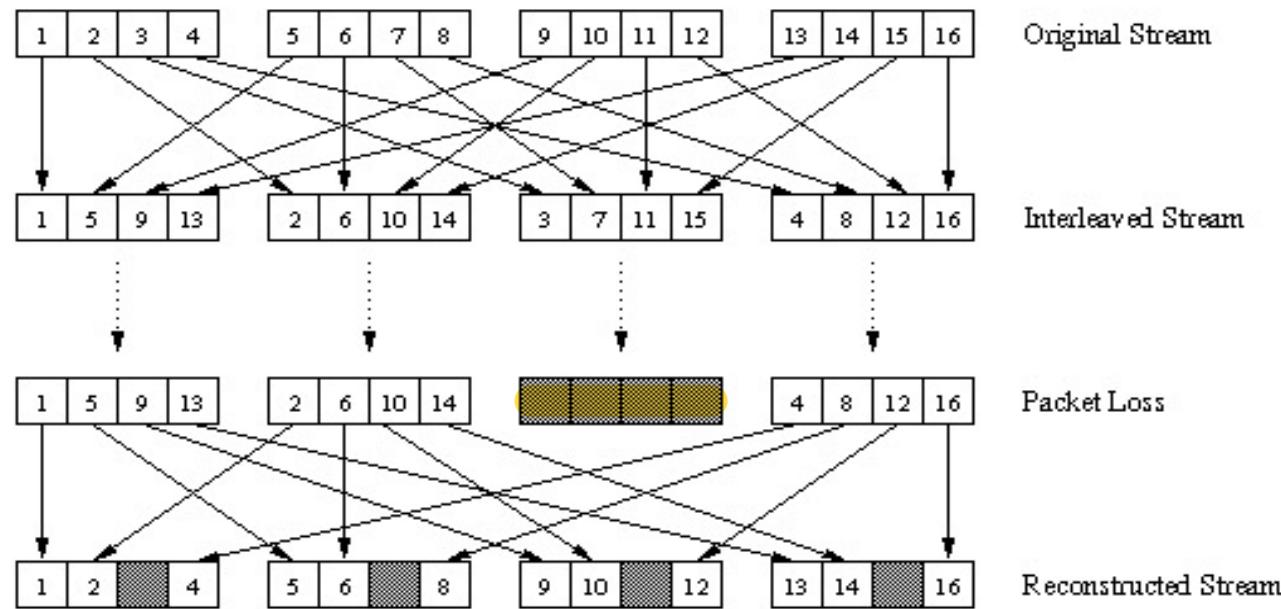
another FEC scheme:

- “piggyback lower quality stream”
- send lower resolution audio stream as redundant information
- e.g., nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps
- non-consecutive loss: receiver can conceal loss
- generalization: can also append $(n-1)^{\text{st}}$ and $(n-2)^{\text{nd}}$ low-bit rate chunk



VoIP: recovery from packet loss (3)

not really error recovery -
will not recover any data



interleaving to conceal loss:

- audio chunks divided into smaller units, e.g. four 5 msec units per 20 msec audio chunk
- packet contains small units from different chunks

- if packet lost, still have **most** of every original chunk
- no redundancy overhead, **but increases playout delay**

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming stored video

9.3 voice-over-IP

9.4 protocols for *real-time conversational*
applications: RTP, SIP

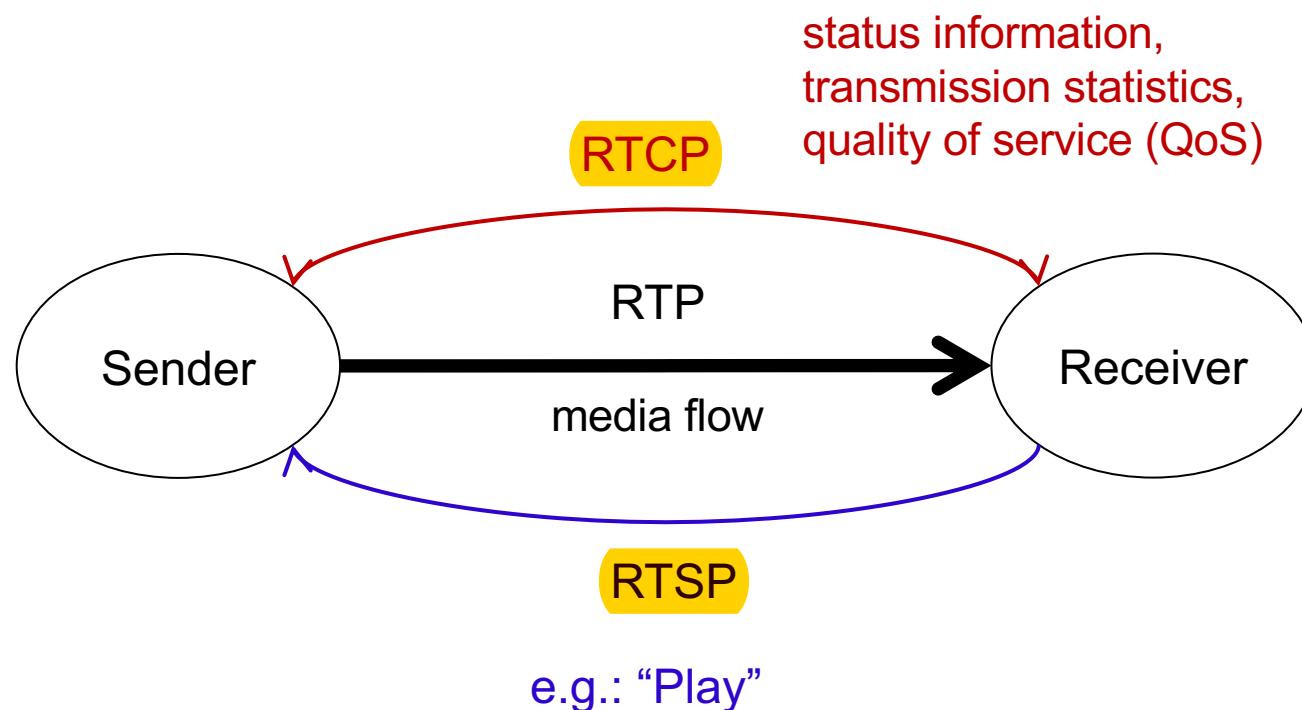
9.5 dynamic adaptive streaming over HTTP
(DASH)

Real-Time Protocol (RTP)

- RTP specifies packet structure for packets carrying audio, video data
- RFC 3550
- RTP packet provides
 - payload type identification
 - packet sequence numbering
 - time stamping
- RTP runs in end systems
- RTP packets encapsulated in UDP segments
- interoperability: if two VoIP applications run RTP, they may be able to work together

Real-Time Protocol Suite

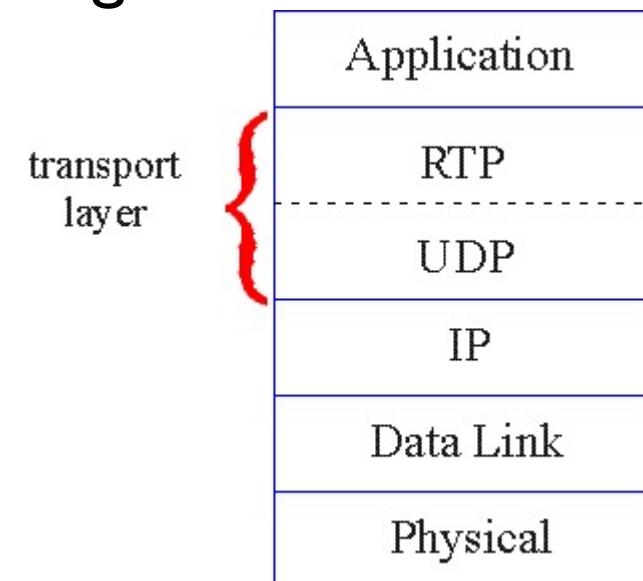
- Flow diagram: RTP, RTCP, RTSP



RTP runs on top of UDP

RTP libraries provide transport-layer interface that extends UDP:

- port numbers, IP addresses
- payload type identification
- packet sequence numbering
- time-stamping



RTP example

example: sending 64 kbps
PCM-encoded voice over
RTP

- application collects encoded data in chunks, e.g., every 20 msec = 160 bytes in a chunk
- audio chunk + RTP header form RTP packet, which is encapsulated in UDP segment

- RTP header indicates type of audio encoding in each packet
 - sender can change encoding during conference
- RTP header also contains sequence numbers, timestamps

RTP and QoS

- RTP does *not* provide any mechanism to ensure timely data delivery or other QoS guarantees
- RTP encapsulation only seen at end systems (*not* by intermediate routers)
 - routers provide best-effort service, making no special effort to ensure that RTP packets arrive at destination in timely matter

RTP header

<i>payload type</i>	<i>sequence number</i>	<i>time stamp</i>	<i>Synchronization Source ID</i>	<i>Miscellaneous fields</i>
---------------------	------------------------	-------------------	----------------------------------	-----------------------------

payload type (7 bits): indicates type of encoding currently being used. If sender changes encoding during call, sender informs receiver via payload type field

Payload type 0: PCM mu-law, 64 kbps

Payload type 3: GSM, 13 kbps

Payload type 7: LPC, 2.4 kbps

Payload type 26: Motion JPEG

Payload type 31: H.261

Payload type 33: MPEG2 video

sequence # (16 bits): increment by one for each RTP packet sent

- ❖ detect packet loss, restore packet sequence

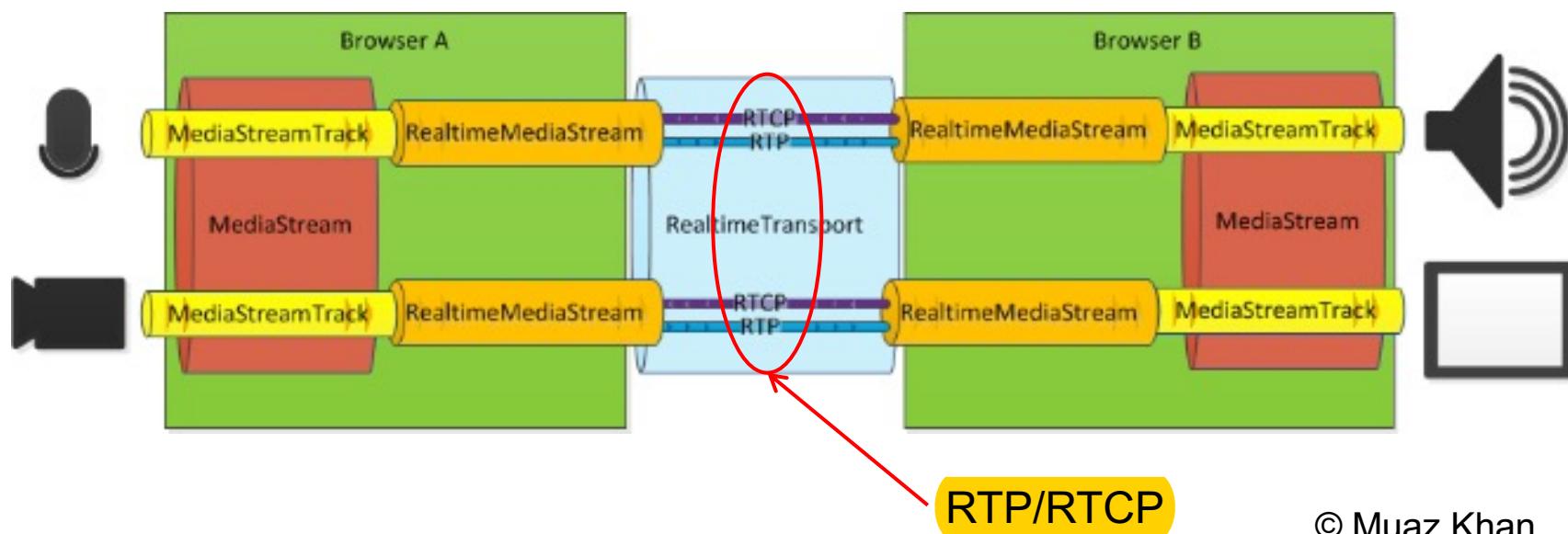
RTP header

<i>payload type</i>	<i>sequence number</i>	<i>time stamp</i>	<i>Synchronization Source ID</i>	<i>Miscellaneous fields</i>
---------------------	------------------------	-------------------	----------------------------------	-----------------------------

- ***timestamp field (32 bits long)***: sampling instant of first byte in this RTP data packet
 - for audio, timestamp clock increments by one for each sampling period (e.g., each 125 usecs for 8 KHz sampling clock)
 - if application generates chunks of 160 encoded samples, timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.
- ***SSRC field (32 bits long)***: identifies source of RTP stream. Each stream in RTP session has distinct SSRC

WebRTC (www.webrtc.org)

- Web browsers with Real-Time Communications (RTC) capabilities via simple JavaScript APIs.
- Pipeline for video conferencing in WebRTC (only one-way shown):



© Muaz Khan

WebRTC (Demo)



SIP: Session Initiation Protocol [RFC 3261]

long-term vision:

- all telephone calls, video conference calls take place over Internet
- people identified by names or e-mail addresses, rather than by phone numbers
- can reach callee (*if callee so desires*), no matter where callee roams, no matter what IP device callee is currently using

SIP services

- SIP provides mechanisms for call setup:
 - for caller to let callee know she wants to establish a call
 - so caller, callee can agree on media type, encoding
 - to end call
- determine current IP address of callee:
 - maps mnemonic identifier to current IP address
- call management:
 - add new media streams during call
 - change encoding during call
 - invite others
 - transfer, hold calls

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming stored video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

Notes on Streaming

- RTP/RTSP/RTCP streaming faces several **challenges**
 - Special-purpose server for media, e.g., fine-grained packet scheduling, keep state (complex)
 - Protocols use TCP and UDP transmissions (firewalls)
 - Difficult to cache data (no “web caching”)
- **Advantage**
 - Short end-to-end latency (<100-500 milliseconds)
 -  **still used in WebRTC**

Notes on HTTP Streaming

- Video-on-Demand (VoD) video streaming increasingly uses HTTP streaming
- Simple HTTP streaming just GETs a (whole) video file from an HTTP server
 -  can be wasteful, needs large client buffer
- Solution: Dynamic Adaptive Streaming over HTTP
- Main idea of DASH
 - Use HTTP protocol to “stream” media
 - Divide media into small chunks, i.e., **streamlets**



Notes on HTTP Streaming

■ Advantages of DASH

- Server is simple, i.e., regular web server (no state, proven to be scalable)
- No firewall problems (use port 80 for HTTP)
- Standard (image) web caching works

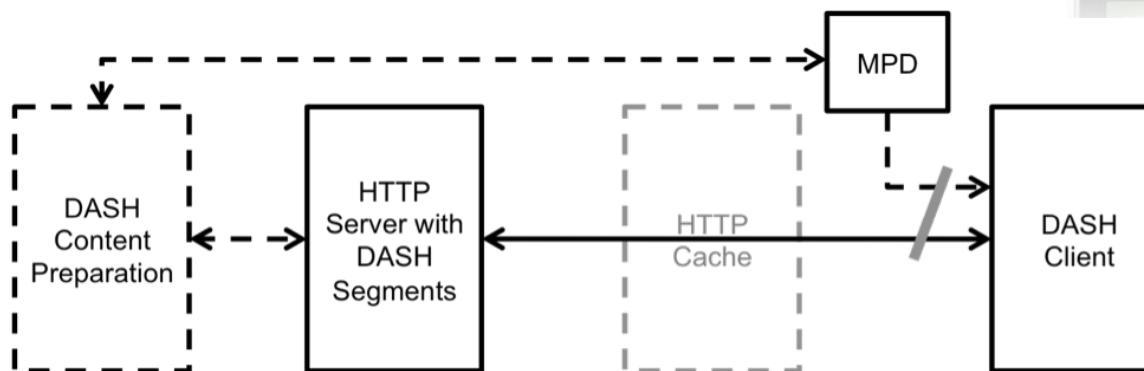
■ Disadvantages

- DASH is based on media segment transmissions, typically 2-10 seconds in length
- By buffering a few segments at the client side, DASH does **not**:
 - Provide low latency for interactive, two-way applications (e.g., video conferencing)

[http://en.wikipedia.org/wiki/Dash_\(disambiguation\)](http://en.wikipedia.org/wiki/Dash_(disambiguation))

How DASH works (I)

- Original DASH implementation by Move Networks
 - Introduced concept of **streamlets**
 - Additional idea: make playback **adaptive**
 - Encode media into multiple different streamlet files, e.g., a **low**, **medium**, and **high** quality version (different bandwidth)

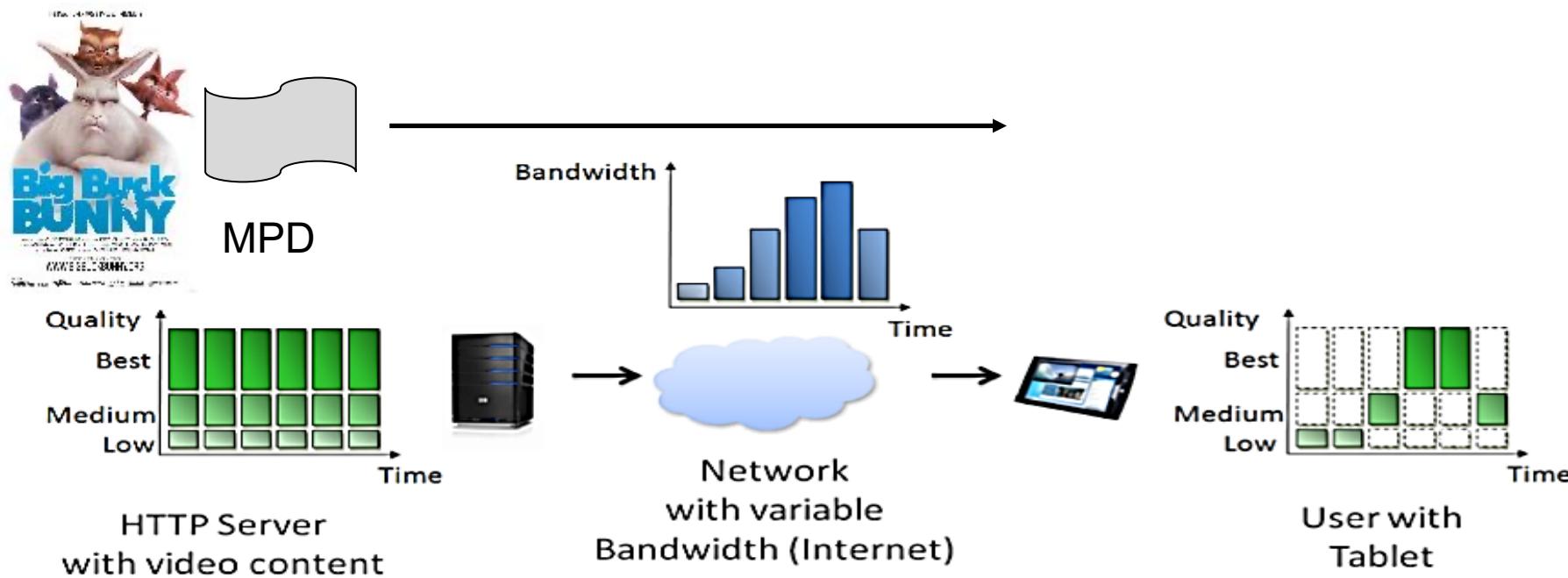


MPD: Media Presentation Description

How DASH works (2)

- Web server provides a **playlist**
 - The playlist is a file in a specific format that lists all the available qualities and all the streamlets for each quality
 - Playlist file extension is .m3u8/.mpd
 - Content preparation:
 - Original media file needs to be **split** into streamlets
 - Streamlets need to be **transcoded** into different qualities
- ISO/IEC Standard:
 - “Information technology — MPEG systems technologies — Part 6: Dynamic adaptive streaming over HTTP (DASH)”
 - JTC I/SC 29; FCD [23009-1](#)

How DASH works (3)



- Data is encoded into **different qualities** and cut into **short segments** (streamlets, chunks).
- Client first downloads MPD, which describes the available videos and qualities.
- Client/player executes an **adaptive bitrate algorithm** (ABR) to determine which segment do download next.

How DASH works (4)

- DASH is very popular now; it has replaced almost all other VoD streaming protocols
- Used by YouTube, Netflix, Facebook, etc.
- Recent focus is on low(er) latency
 - If latency is too long for live events (e.g., sports) then social media (e.g., Twitter) is “out-of-sync”
- Lots of interesting work on how to build the best ABR algorithm
 - high avg. quality, no stalls, low latency

Lecture 10: Summary

- There are various media applications on the Internet, even though it provides (few) guarantees
- Three types of media applications:
 - **VoD**: e.g., YouTube, Netflix
 - one-way, on-demand, media is stored, long latency okay
 - ➡ use DASH
 - **Live streaming**: e.g., Twitch.tv, Periscope
 - one-way, live source, latency should not be too long
 - ➡ use DASH
 - **VoIP, Teleconferencing**: e.g., Skype, Zoom, Webex
 - two-way, low latency critical
 - ➡ use RTP (WebRTC)

CS2105

An Awesome Introduction to Computer Networks

Lecture 11: Network Security



Department of Computer Science
School of Computing

Some material copyright 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved

Lecture 11: Network Security

Chapter goals:

- understand principles of network security:
 - cryptography and its *many* uses beyond “confidentiality”
 - authentication
 - message integrity
- security in practice:
 - firewalls and intrusion detection systems
 - security in application, transport, network, link layers

Chapter 8 roadmap

- 8.1 *What is network security?***
- 8.2 Principles of cryptography**
- 8.3 Message integrity and digital signatures**
- 8.7 Network layer security: VPNs**
- 8.9 Operational security: firewalls**

What is network security?

confidentiality: only sender, intended receiver should “understand” message contents

- sender encrypts message
- receiver decrypts message

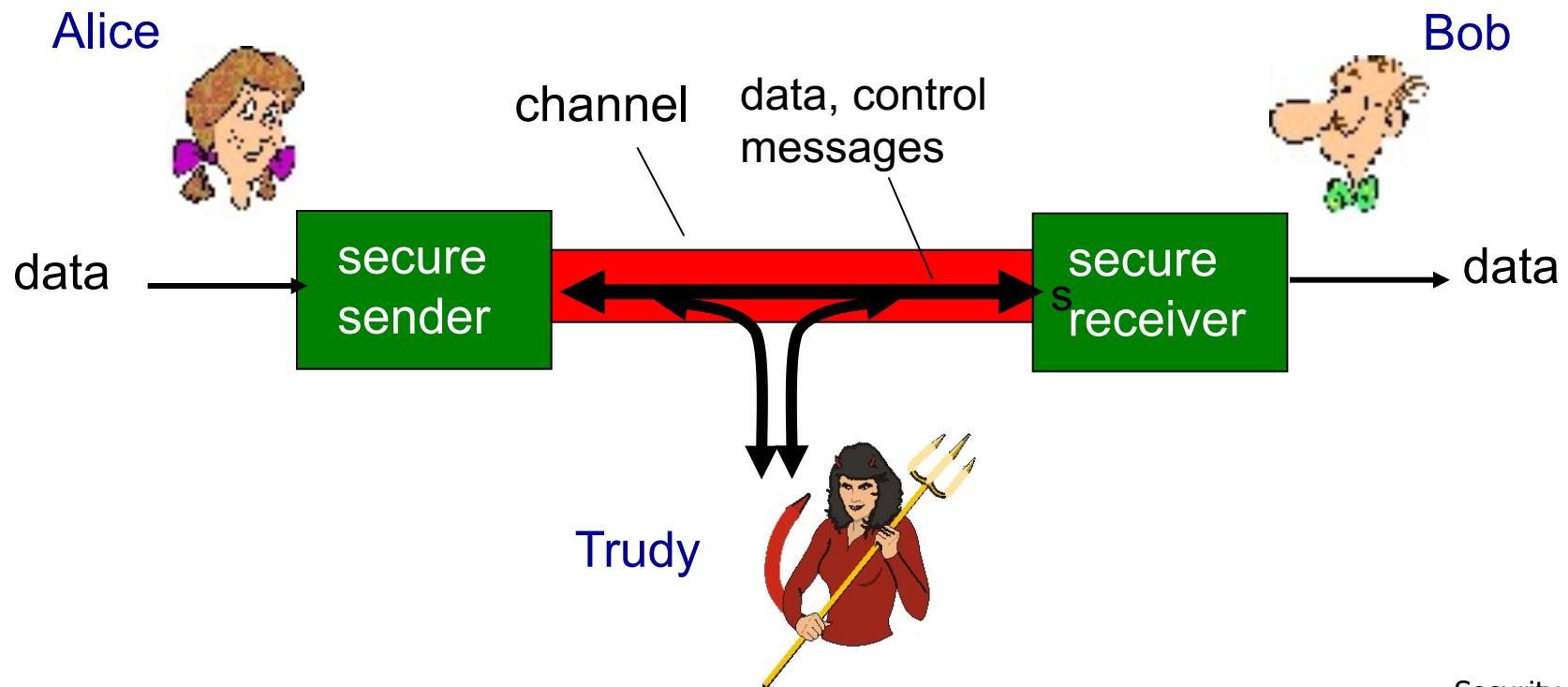
authentication: sender, receiver want to confirm identity of each other

message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

access and availability: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages



Who might Bob, Alice be?

- ... well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions
(e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?



There are bad guys (and girls) out there!

Q: What can a “bad guy” do?

A: A lot!

- **eavesdrop**: intercept messages
- actively **insert** messages into connection
- **impersonation**: can fake (spoof) source address in packet (or any field in packet)
- **hijacking**: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service**: prevent service from being used by others (e.g., by overloading resources)

Chapter 8 roadmap

8.1 What is network security?

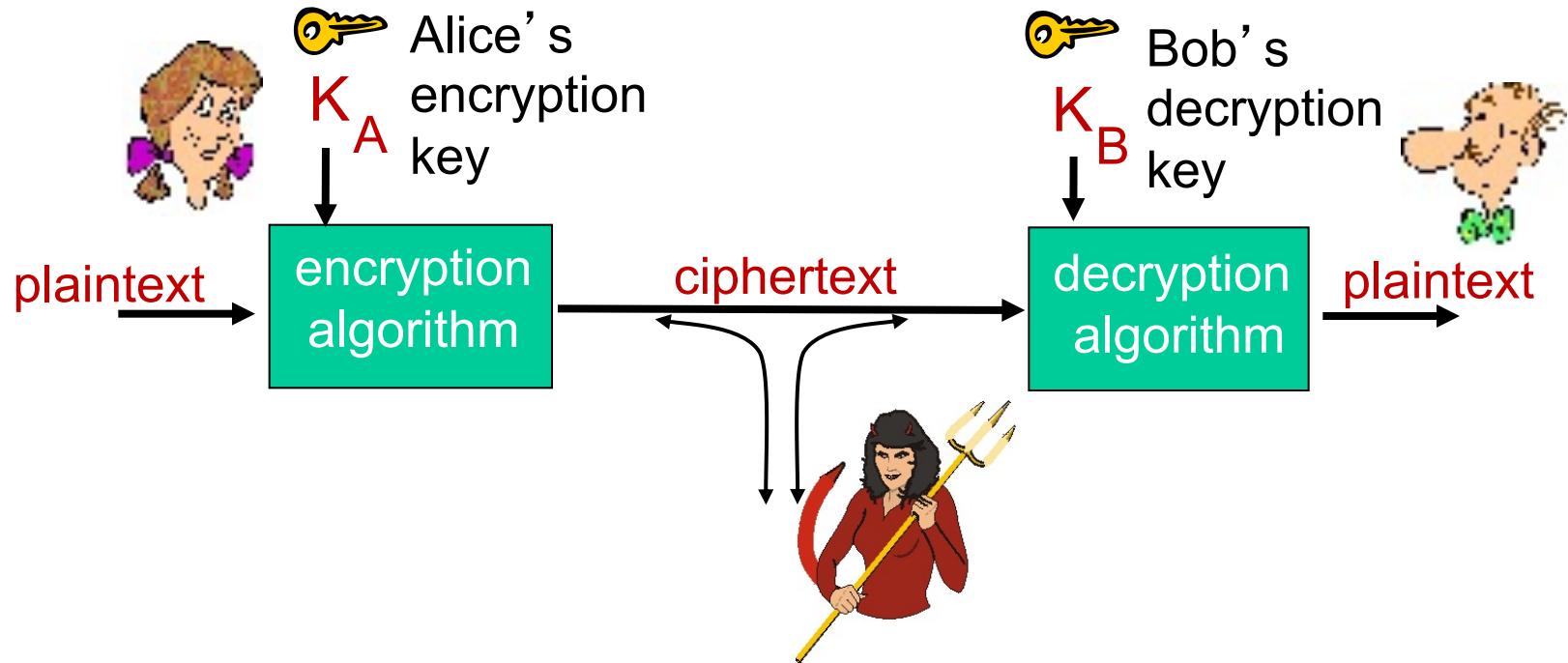
8.2 *Principles of cryptography*

8.3 Message integrity and digital signatures

8.7 Network layer security: VPNs

8.9 Operational security: firewalls

The language of cryptography



m plaintext message

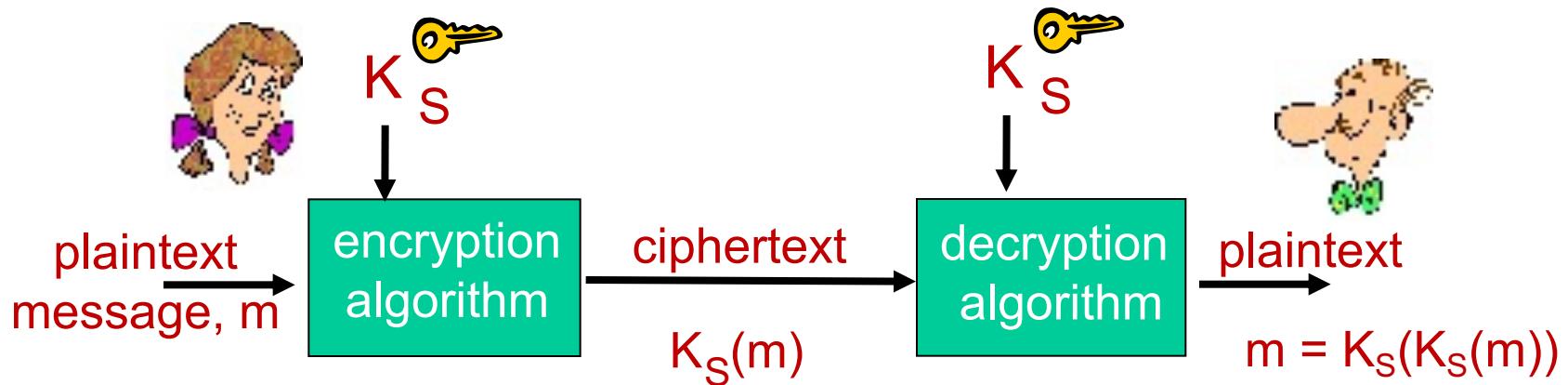
$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

Breaking an encryption scheme

- **cipher-text only attack:**
Trudy has ciphertext she can analyze
- **two approaches:**
 - brute force: search through all keys
 - **statistical analysis**
- **known-plaintext attack:**
Trudy has plaintext corresponding to ciphertext
 - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- **chosen-plaintext attack:**
Trudy can get ciphertext for chosen plaintext

Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

Simple encryption scheme

substitution cipher: substituting one thing for another

- monoalphabetic cipher: substitute one letter for another

plaintext: abcdefghijklmnopqrstuvwxyz
ciphertext: mnbvcxzasdfghjklpoiuytrewq

The diagram illustrates a monoalphabetic substitution cipher. It shows two rows of letters. The top row is labeled "plaintext" and contains the letters a through z. The bottom row is labeled "ciphertext" and contains the letters m through q. Red arrows point from each letter in the plaintext row to its corresponding letter in the ciphertext row, showing the mapping rule.

e.g.: Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc



Encryption key: mapping from set of 26 letters
to set of 26 letters

Side-note: Caesar's cipher

- This method is named after Julius Caesar, who used it in his private correspondence
- Fixed shift of alphabet, e.g., left rotation by 3 (or right by 23):

plaintext: abcdefghijklmnopqrstuvwxyz

ciphertext: xyzabcdefghijklmnopqrstuvwxyz

The diagram illustrates a Caesar cipher shift of 1. It shows the plaintext alphabet from 'a' to 'z' above the ciphertext alphabet from 'x' to 'z'. A blue arrow points from 'a' to 'b', indicating the shift. Two red arrows point from 'z' to 'x' and 'y' respectively, highlighting the wrap-around nature of the shift.

e.g.: plaintext: the quick brown fox
 ciphertext: qeb nrfzh yoltk clu



Encryption key: only need shift number

A more sophisticated encryption approach

- n substitution ciphers, M_1, M_2, \dots, M_n
- cycling pattern:
 - e.g., $n=4$: $M_1, M_3, M_4, M_3, M_2; M_1, M_3, M_4, M_3, M_2; \dots$
- for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
 - dog: d from M_1 , o from M_3 , g from M_4



Encryption key: n substitution ciphers, and cyclic pattern

- key need not be just n-bit pattern

Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
 - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
 - no known good analytic attack
- making DES more secure:
 - 3DES: encrypt 3 times with 3 different keys

AES: Advanced Encryption Standard

- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 second on DES, takes 149 trillion years for AES



Adding 1 bit will double the amount of time to brute force

Public Key Cryptography

symmetric key crypto

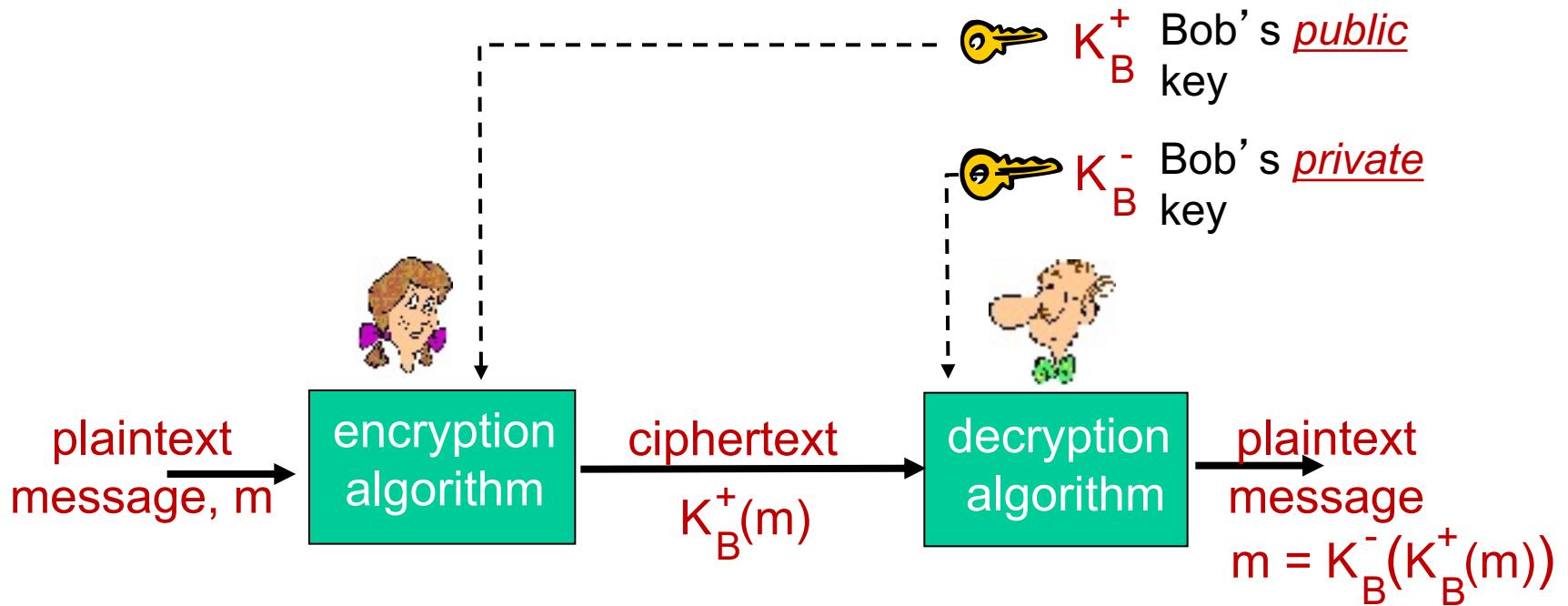
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

public key crypto

- radically different approach [Diffie-Hellman'76, RSA'78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver



Public key cryptography



Public key encryption algorithms

requirements:

- ① need $K_B^+(.)$ and $K_B^-(.)$ such that

$$K_B^-(K_B^+(m)) = m$$

- ② given public key K_B^+ , it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adleman algorithm

Prerequisite: modular arithmetic

- $x \bmod n = \text{remainder of } x \text{ when divide by } n$
- facts:
 - $[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$
 - $[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$
 - $[(a \bmod n) * (b \bmod n)] \bmod n = (a*b) \bmod n$
- thus
 - $(a \bmod n)^d \bmod n = a^d \bmod n$
- example: $x=14, n=10, d=2:$
 - $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
 - $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

RSA: getting ready

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number

example:

- $m = 10010001$. This message is uniquely represented by the decimal number 145.
- to encrypt m , we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA: Creating public/private key pair

1. choose two large prime numbers p, q .
(e.g., 1024 bits each)
2. compute $n = pq$, $z = (p-1)(q-1)$
3. choose e (with $e < n$) that has no common factors with z (e, z are “relatively prime”).
4. choose d such that $ed - 1$ is exactly divisible by z .
(in other words: $ed \bmod z = 1$).
5. public key is $\underbrace{(n,e)}_{K_B^+}$. private key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: encryption, decryption

0. given (n,e) and (n,d) as computed above

I. to encrypt message $m (< n)$, compute

$$c = m^e \bmod n$$

2. to decrypt received bit pattern, c , compute

$$m = c^d \bmod n$$

magic happens! $m = \underbrace{(m^e \bmod n)^d}_{c} \bmod n$

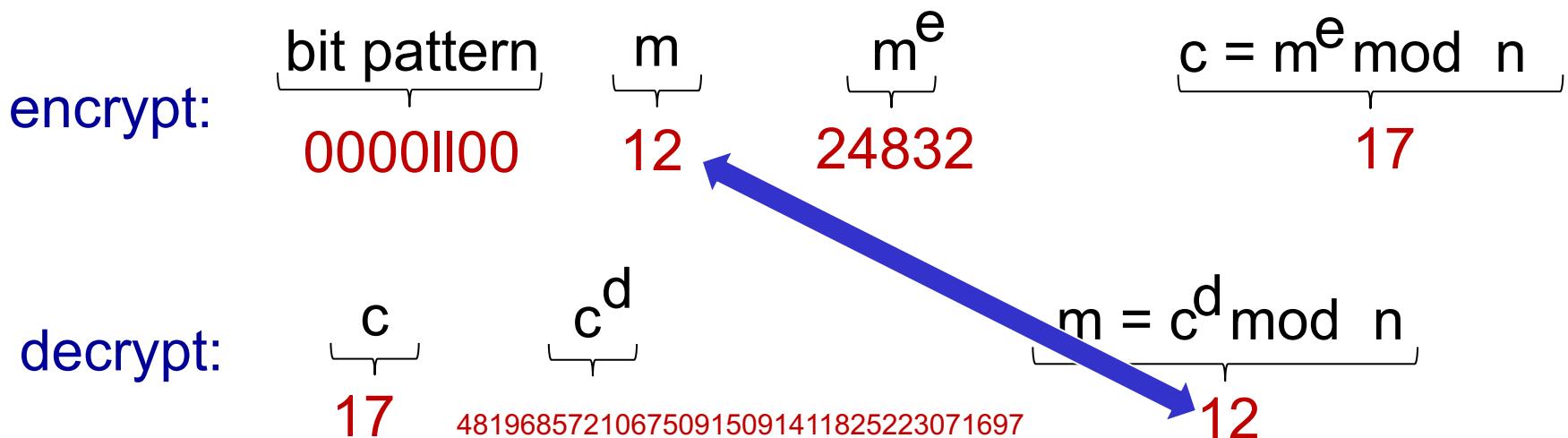
RSA example:

Bob chooses $p=5$, $q=7$. Then $n=35$, $z=24$.

$e=5$ (so e , z relatively prime).

$d=29$ (so $ed-1$ exactly divisible by z).

encrypting 8-bit messages.



Why does RSA work?

- must show that $c^d \bmod n = m$
where $c = m^e \bmod n$
- fact: for any x and y : $x^y \bmod n = x^{(y \bmod z)} \bmod n$
 - where $n = pq$ and $z = (p-1)(q-1)$
- thus,
$$\begin{aligned} c^d \bmod n &= (m^e \bmod n)^d \bmod n \\ &= m^{ed} \bmod n \\ &= m^{(ed \bmod z)} \bmod n \\ &= m^l \bmod n \\ &= m \end{aligned}$$

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first,}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

use public key first,
followed by
private key

use private key
first, followed by
public key

result is the same!

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$\begin{aligned}(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\&= m^{de} \bmod n \\&= (m^d \bmod n)^e \bmod n\end{aligned}$$

Why is RSA secure?

- suppose you know Bob's public key (n, e). How hard is it to determine d ?
- essentially need to find factors of n without knowing the two factors p and q
 - fact: factoring a big number is hard

RSA in practice: session keys

- exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- use public key crypto to establish secure connection, then establish second key –
symmetric session key – for encrypting data

session key, K_S

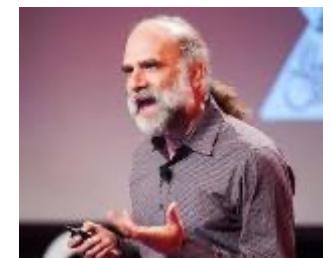
- Bob and Alice use RSA to exchange a symmetric key K_S
- once both have K_S , they use symmetric key cryptography

Recommended key lengths

- Recommendations [Lenstra and Verheul]

	1982	1995	2002	2010	2020	2030	2040
Sym	56	66	72	78	86	93	101
RSA	417	777	1028	1369	1881	2493	3214

- Bruce Schneier's Applied Cryptography, p.18
 - Probability to get hit by lightning per day (10^{-10} , 2^{-33})
 - Number of atoms on earth (10^{51} , 2^{170})
 - Number of atoms in the universe (10^{77} , 2^{265})
 - Time until next ice age (14,000, 2^{14} years)
 - Duration until sun goes nova (10^9 , 2^{30} years)
 - Age of the Universe (10^{10} , 2^{33} years)



Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 *Message integrity and digital signatures*

8.7 Network layer security: VPNs

8.9 Operational security: firewalls

Digital signatures

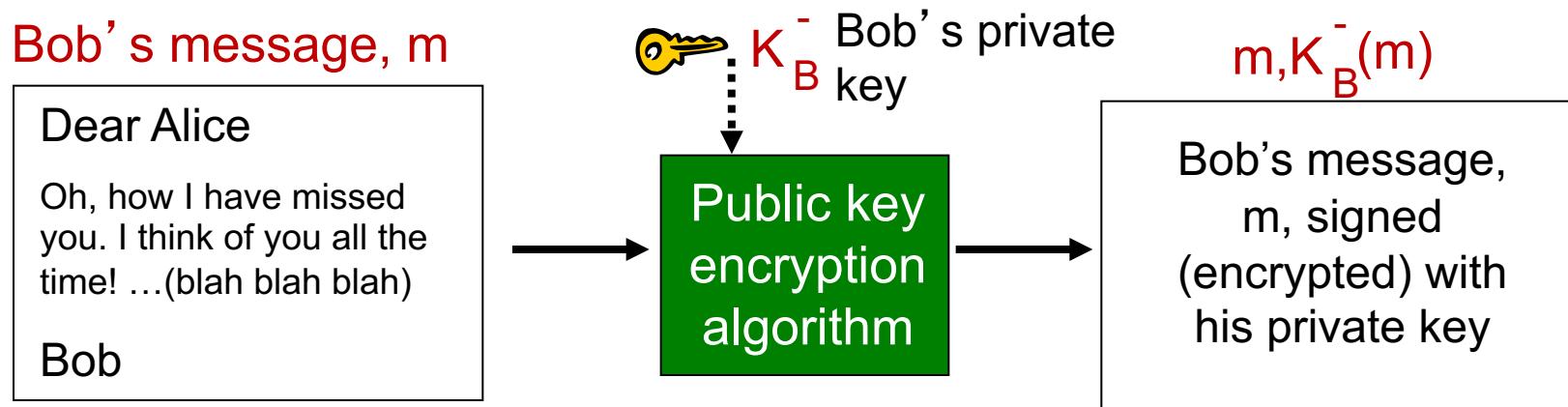
cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital signatures

simple digital signature for message m :

- Bob signs m by encrypting with his private key K_B^- , creating “signed” message, $K_B^-(m)$



💡 Pending the plaintext here is just as a concept - since usually signing a document will still allow the document to be read normally

Digital signatures

- Suppose Alice receives msg m , with signature: $m, K_B^-(m)$.
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

Alice thus verifies that:

- Bob signed m
- no one else signed m
- Bob signed m and not m'

non-repudiation:

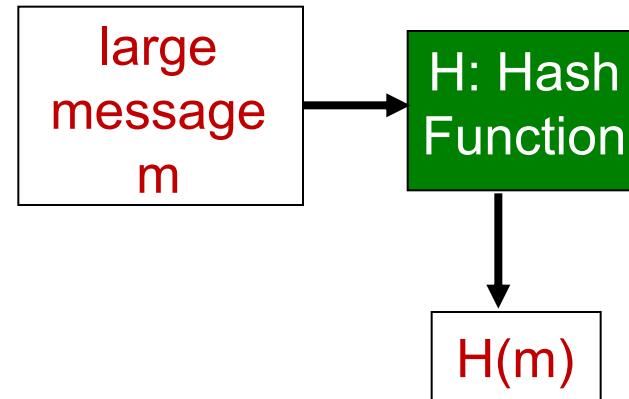
- ✓ Alice can take m , and signature $K_B^-(m)$ to court and prove that Bob signed m

Message digests

computationally expensive to public-key-encrypt long messages

goal: fixed-length, easy-to-compute digital “fingerprint”

- apply hash function H to m , get fixed size message digest, $H(m)$.



Hash function properties:

- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest x , computationally infeasible to find m such that $x = H(m)$

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

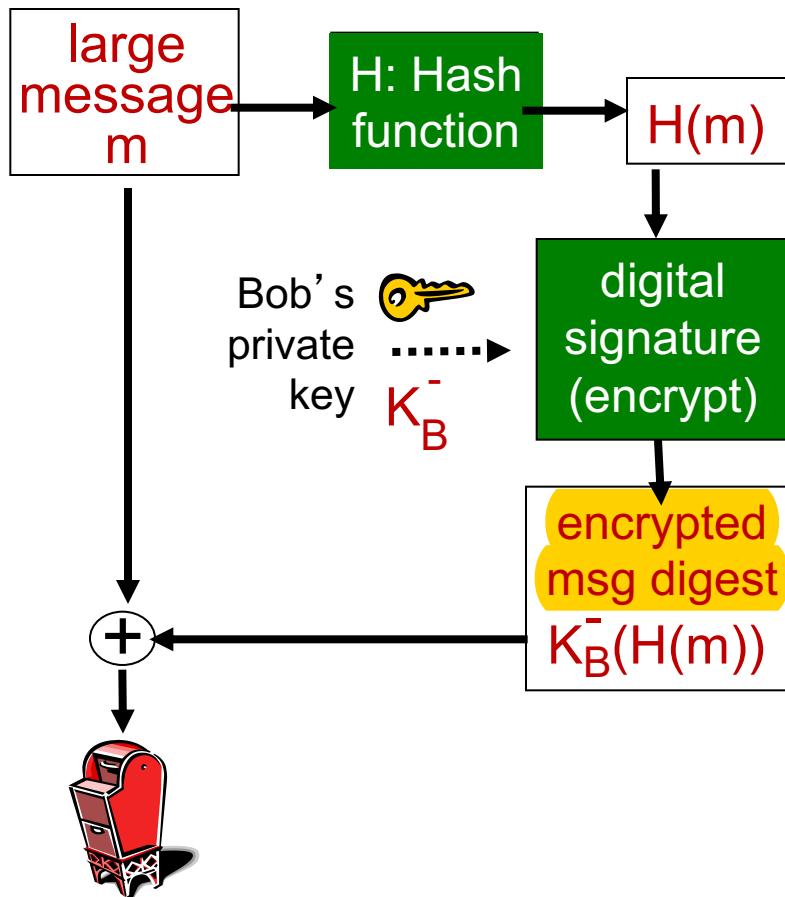
- produces fixed length digest (16-bit sum) of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

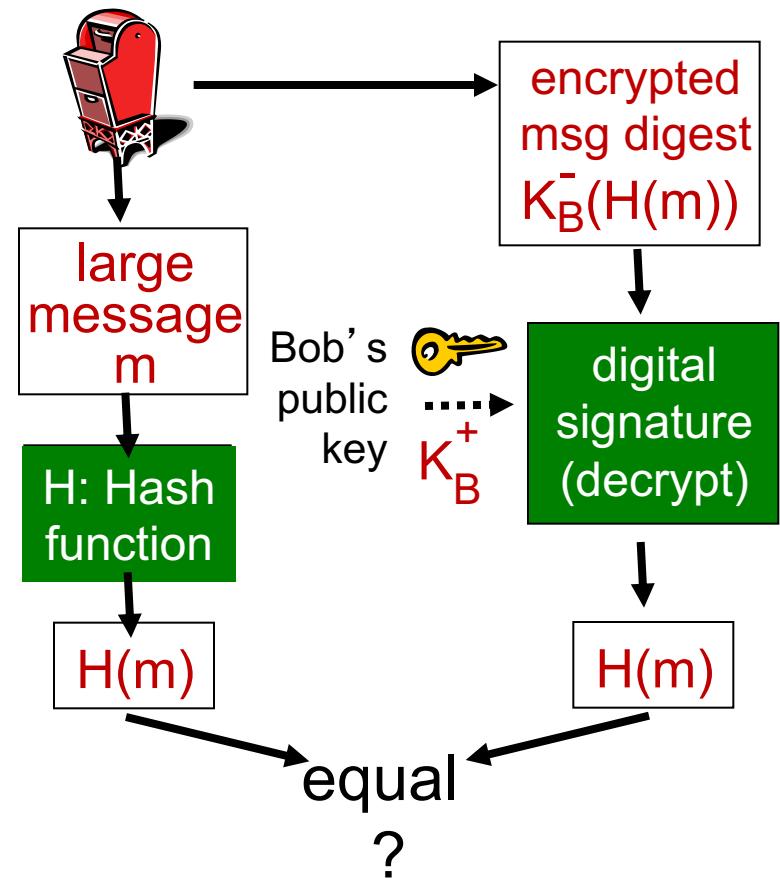
<u>message</u>	<u>ASCII format</u>	<u>message</u>	<u>ASCII format</u>
I O U 1	49 4F 55 31	I O U <u>9</u>	49 4F 55 <u>39</u>
0 0 . 9	30 30 2E 39	0 0 . <u>1</u>	30 30 2E <u>31</u>
9 B O B	39 42 D2 42	9 B O B	39 42 D2 42
<hr/>		<hr/>	
B2 C1 D2 AC		different messages but identical checksums!	

Digital signature = signed message digest

Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:



Hash function algorithms

- **MD5 hash function widely used (RFC 1321)**
 - computes 128-bit message digest in 4-step process.
 - arbitrary 128-bit string x , appears difficult to construct msg m whose MD5 hash is equal to x
- **SHA-1 is also used**
 - US standard [NIST, FIPS PUB 180-1]
 - 160-bit message digest

Hash function, e.g., md5sum (I)

- Generate short, fixed-length outputs (or digests); 128 bits
 - especially useful for longer inputs; “fingerprint”

```
$ cat smallfile
This is a very small file with a few characters

$ cat bigfile
This is a larger file that contains more characters.
This demonstrates that no matter how big the input
stream is, the generated hash is the same size (but
of course, not the same value). If two files have
a different hash, they surely contain different data.

$ ls -l empty-file smallfile bigfile linux-kernel
-rw-rw-r--    1 steve      steve          0 2004-08-20 08:58 empty-file
-rw-rw-r--    1 steve      steve         48 2004-08-20 08:48 smallfile
-rw-rw-r--    1 steve      steve        260 2004-08-20 08:48 bigfile
-rw-r--r--    1 root       root     1122363 2003-02-27 07:12 linux-kernel

$ md5sum empty-file smallfile bigfile linux-kernel
d41d8cd98f00b204e9800998ecf8427e  empty-file
75cdbfeb70a06d42210938da88c42991  smallfile
6e0b7a1676ec0279139b3f39bd65e41a  bigfile
c74c812e4d2839fa9acf0aa0c915e022  linux-kernel
```

Hash function, e.g., md5sum (2)

- A small change in the input should result in a large change in the hash output

```
$ cat file1
This is a very small file with a few characters
$ cat file2
this is a very small file with a few characters
$ md5sum file?
75cdbfeb70a06d42210938da88c42991  file1
6fbe37fleea0f802bd792ea885cd03e2  file2
```

Password hashing

- Passwords are not stored in plaintext but hashed and stored

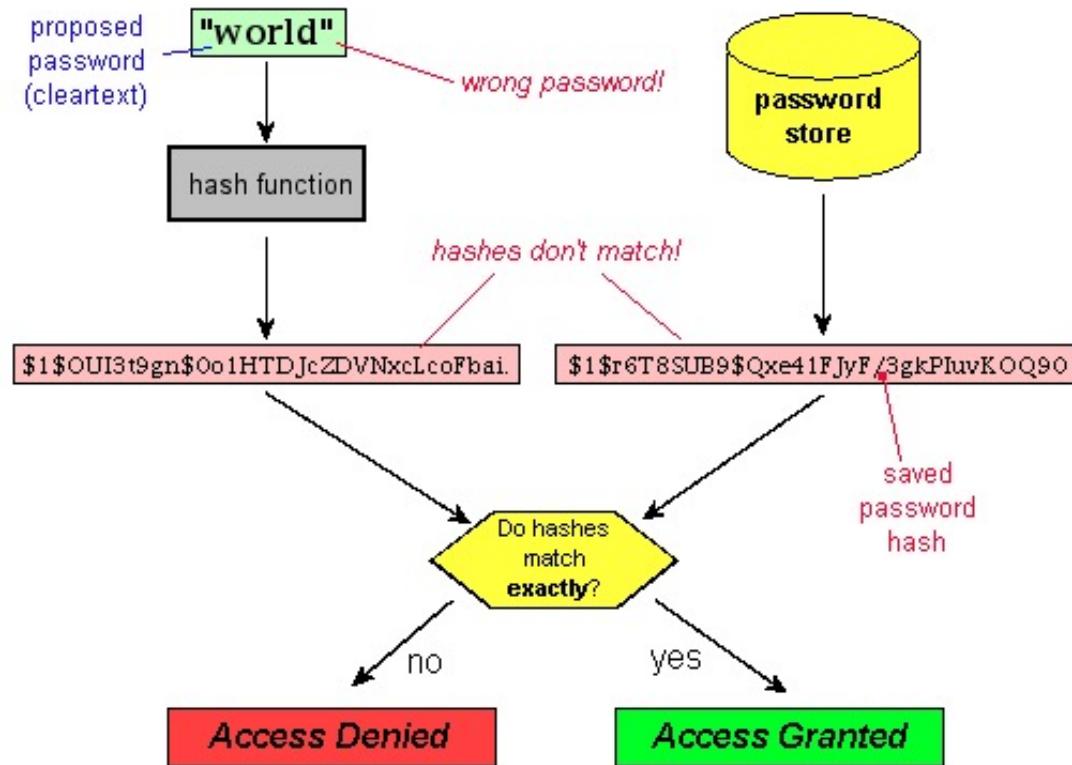


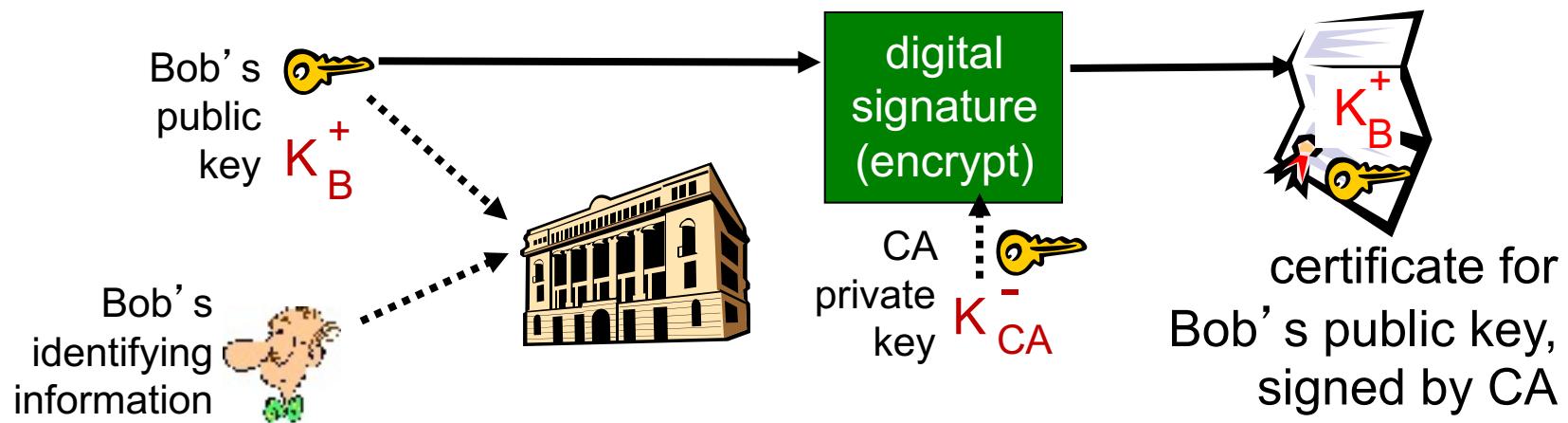
Fig. 5: Testing a proposed password against the stored hash

Public-key certification

- motivation: Trudy plays pizza prank on Bob
 - Trudy creates e-mail order:
Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
 - Trudy signs order with her private key
 - Trudy sends order to Pizza Store
 - Trudy sends to Pizza Store her public key, but says it's Bob's public key
 - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
 - Bob doesn't even like pepperoni

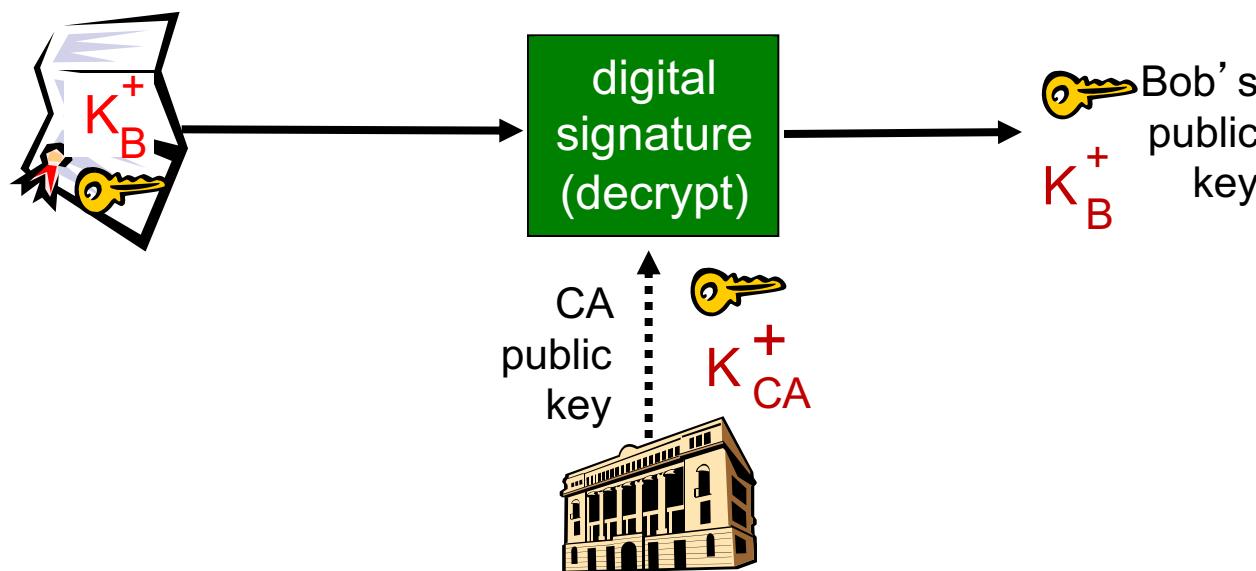
Certification authorities

- ***certification authority (CA)***: binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
 - E provides “proof of identity” to CA.
 - CA creates certificate binding E to its public key.
 - certificate containing E’s public key digitally signed by CA – CA says “this is E’s public key”



Certification authorities

- when Alice wants Bob's public key:
 - gets Bob's certificate (Bob or elsewhere).
 - apply CA's public key to Bob's certificate, get Bob's public key



Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and digital signatures

8.7 *Network layer security: VPNs*

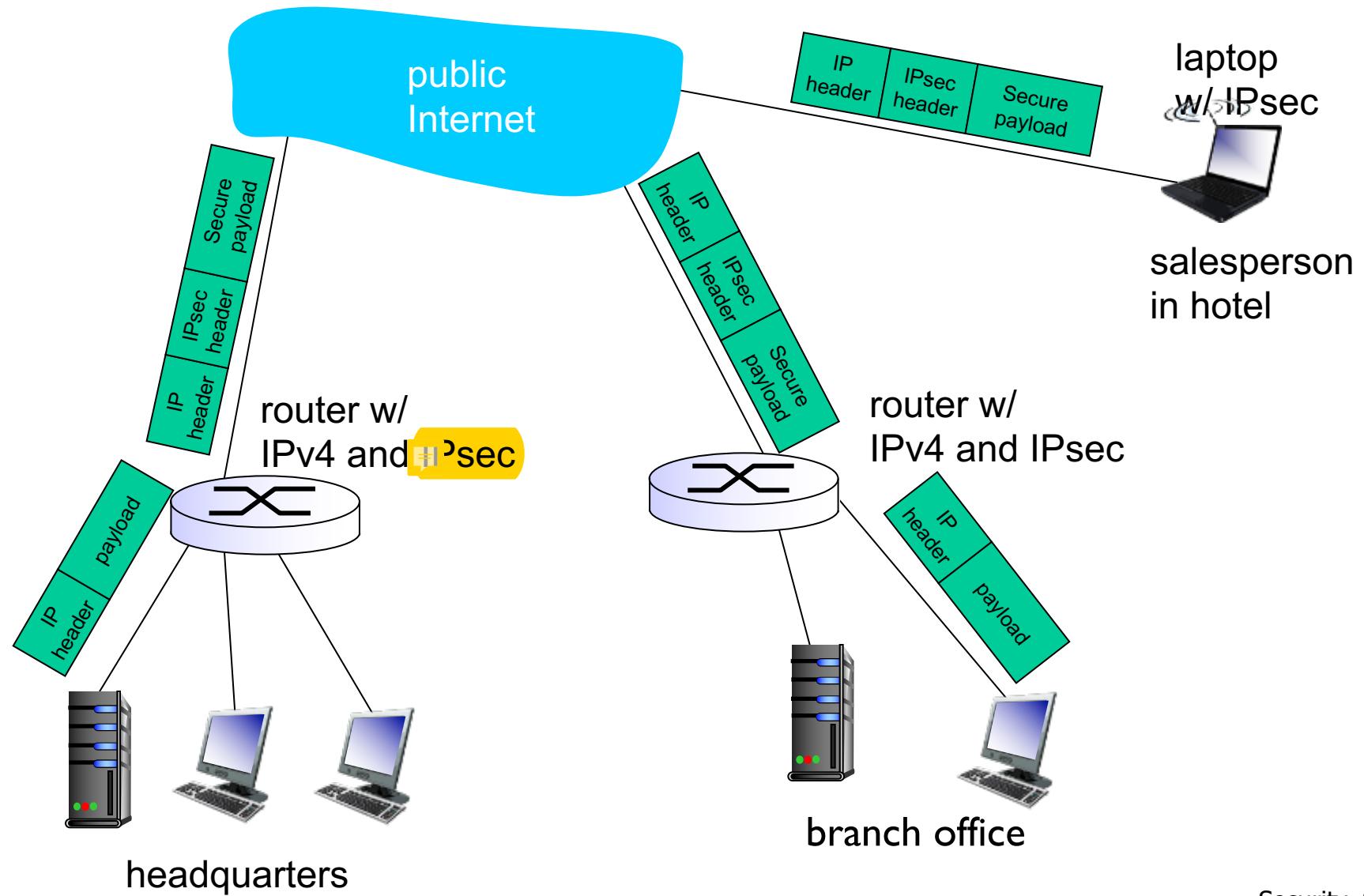
8.9 Operational security: firewalls

Virtual Private Networks (VPNs)

motivation:

- institutions often want private networks for security.
 - costly: separate routers, links, DNS infrastructure.
- VPN: institution's inter-office traffic is sent over public Internet instead
 - encrypted before entering public Internet
 - logically separate from other traffic

Virtual Private Networks (VPNs)



Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity and digital signatures

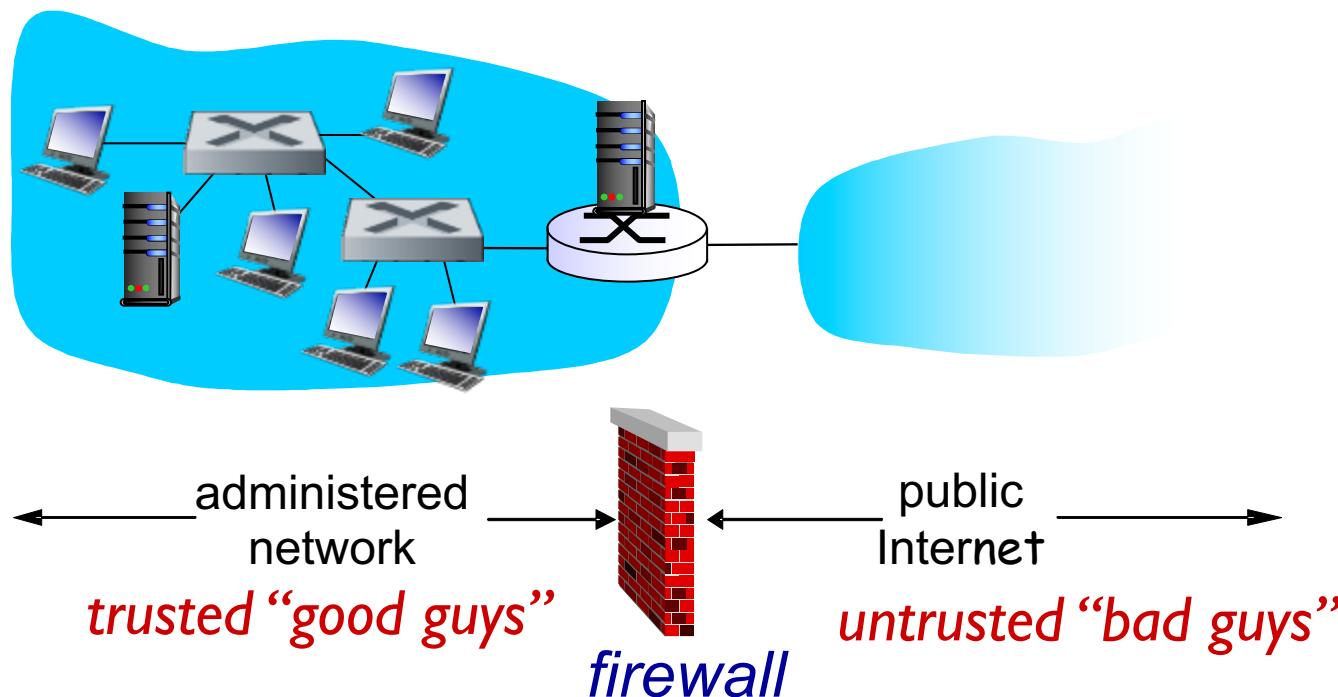
8.7 Network layer security: VPNs

8.9 *Operational security: firewalls*

Firewalls

firewall

isolates organization's internal net from larger Internet,
allowing some packets to pass, blocking others



Firewalls: why

prevent denial of service attacks:

- SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections

prevent illegal modification/access of internal data

- e.g., attacker replaces CIA’s homepage with something else

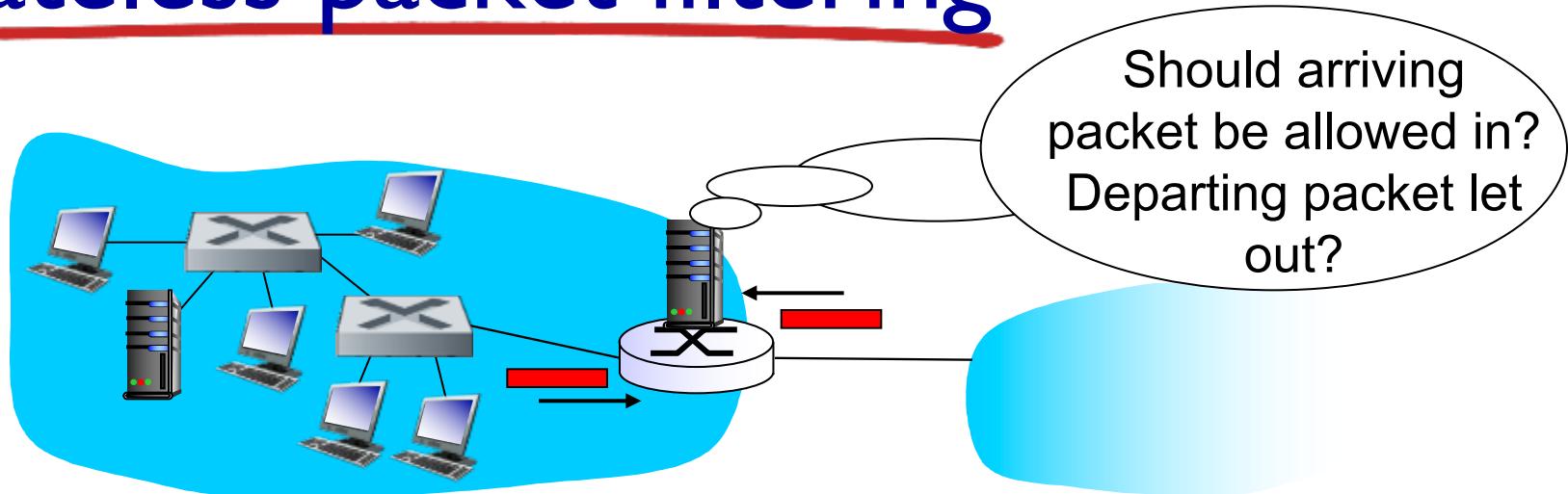
allow only authorized access to inside network

- set of authenticated users/hosts

three types of firewalls:

- stateless packet filters
- stateful packet filters
- application gateways

Stateless packet filtering



- internal network connected to Internet via *router firewall*
- router **filters packet-by-packet**, decision to forward/drop packet based on:
 - source IP address, destination IP address
 - TCP/UDP source and destination port numbers
 - ICMP message type
 - TCP SYN and ACK bits

Stateless packet filtering: example

- *example 1:* block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
 - *result:* all incoming, outgoing UDP flows and telnet connections are blocked
- *example 2:* block inbound TCP segments with ACK=0.
 - *result:* prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.

Stateless packet filtering: more examples

<i>Policy</i>	<i>Firewall Setting</i>
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Stateful packet filtering

- *stateless packet filter*: heavy handed tool
 - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established:

action	source address	dest address	protocol	source port	dest port	flag bit
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK

- *stateful packet filter*: track status of every TCP connection
 - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
 - timeout inactive connections at firewall: no longer admit packets

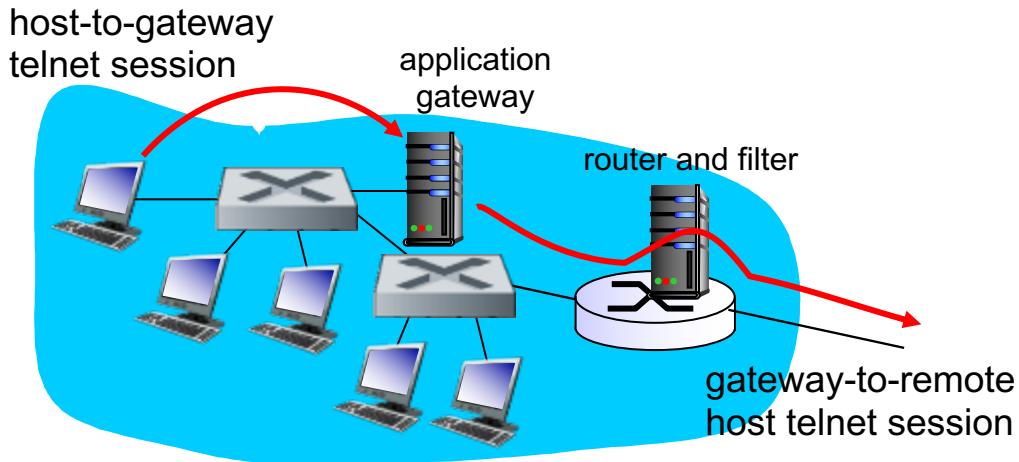
Stateful packet filtering

ACL augmented to indicate need to check connection state table before admitting packet

action	source address	dest address	proto	source port	dest port	flag bit	check connxion
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any	
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK	X
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---	
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----	X
deny	all	all	all	all	all	all	

Application gateways

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside



1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway.

Limitations of firewalls, gateways

- *IP spoofing*: router can't know if data "really" comes from claimed source
- if multiple app's. need special treatment, each has own app. gateway
- client software must know how to contact gateway.
 - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- *tradeoff*: degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks

Network Security (summary)

basic techniques.....

- cryptography (symmetric and public)
- message integrity, digital signatures
- end-point authentication

.... used in many different security scenarios (not covered in lecture)

- secure email
- secure transport (SSL)
- IP sec
- 802.11

operational security: firewalls and VPNs