

# An Awesome Introduction to Computer Networks

# 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

- 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

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 known as *hosts* or *end systems*.



PC



laptop



server

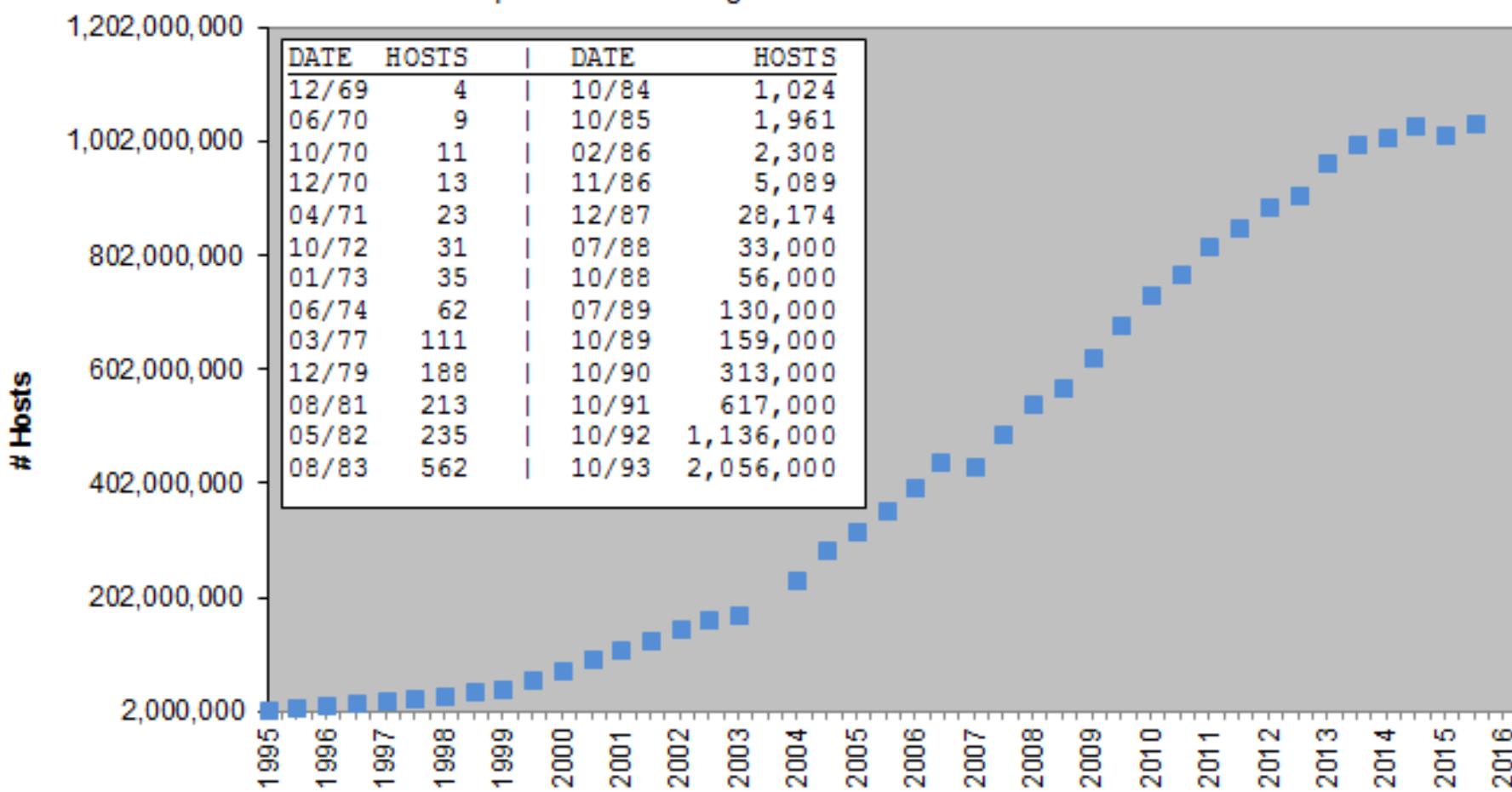


smartphone

- ❖ Hosts run *network applications*.
  - Examples: Web, Pokémon Go, BitTorrent, Skype etc.

# Growth of Internet Hosts

Hobbes' Internet Timeline Copyright ©2016 Robert H Zakon  
<http://www.zakon.org/robert/internet/timeline/>



# Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

- hosts, access networks, links

1.3 Network Core

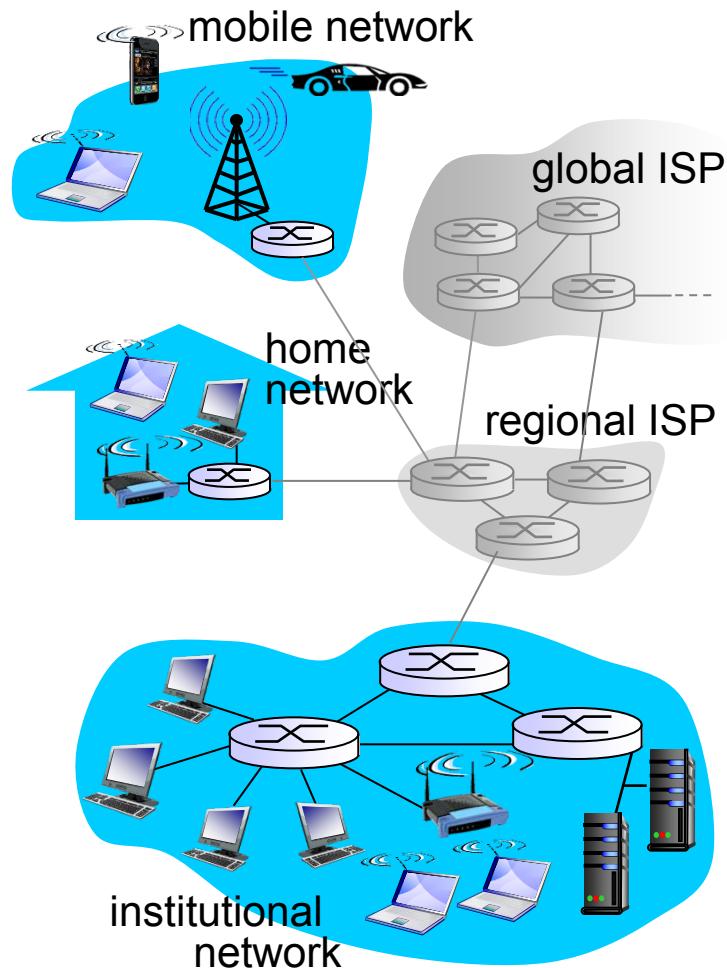
- packet switching, circuit switching, network structure

1.4 Delay, Loss and Throughput in Networks

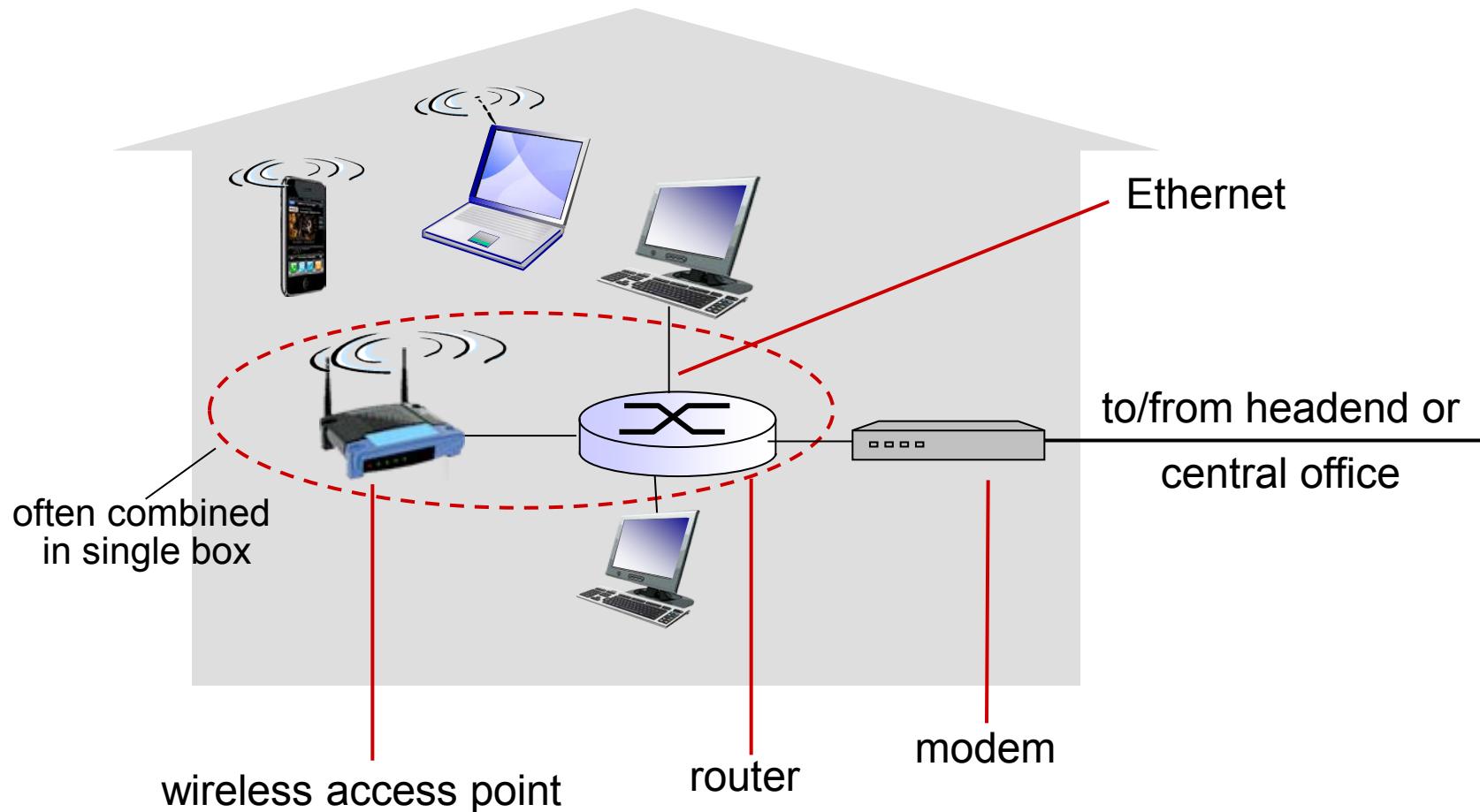
1.5 Protocol Layers and Service Models

# Access Networks

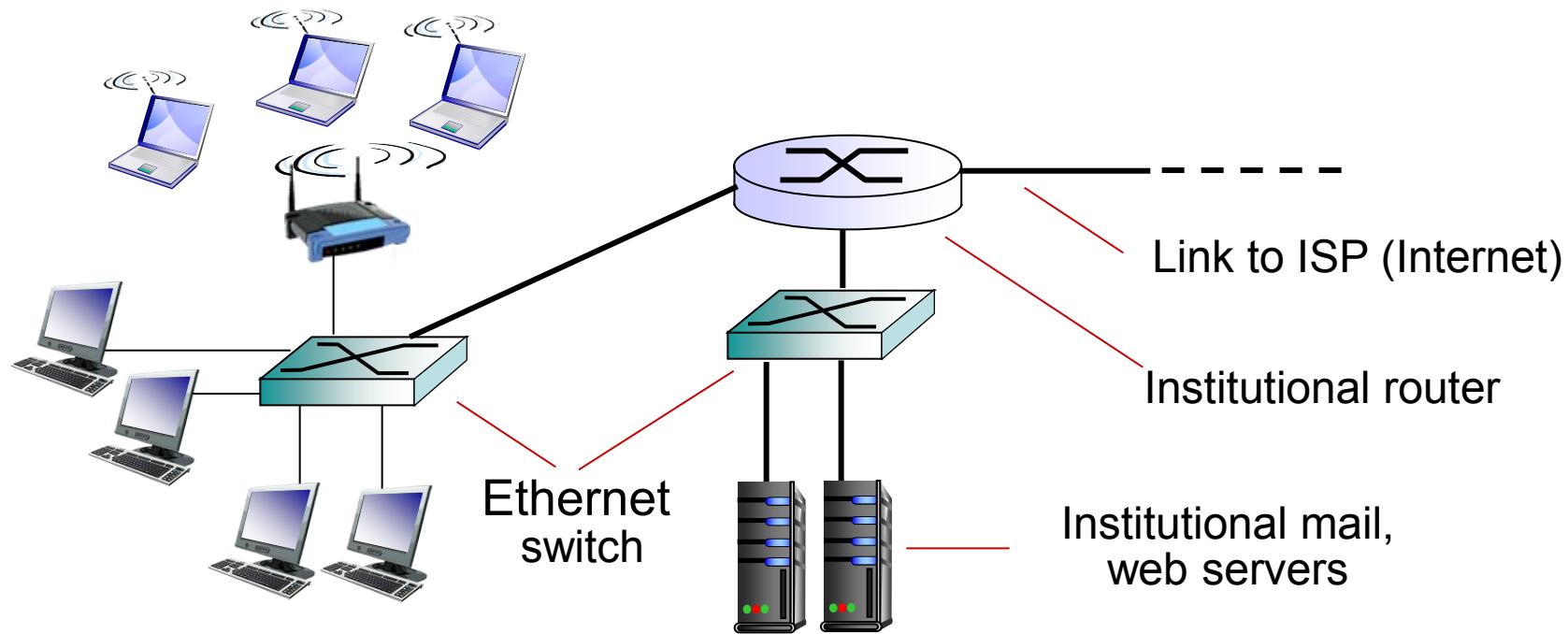
- ❖ Hosts access the Internet through *access network*.
  - Residential access networks
  - Institutional access networks (school, company)
  - Mobile access networks



# Home Networks



# Enterprise Access Networks (Ethernet)



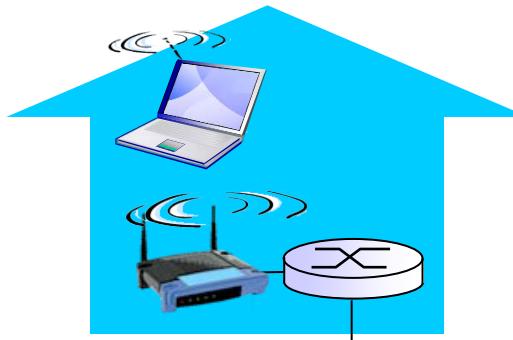
- ❖ Typically used in companies, universities, etc.
- ❖ 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- ❖ Today, hosts typically connect to Ethernet switch

# Wireless Access Networks

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

## Wireless LANs:

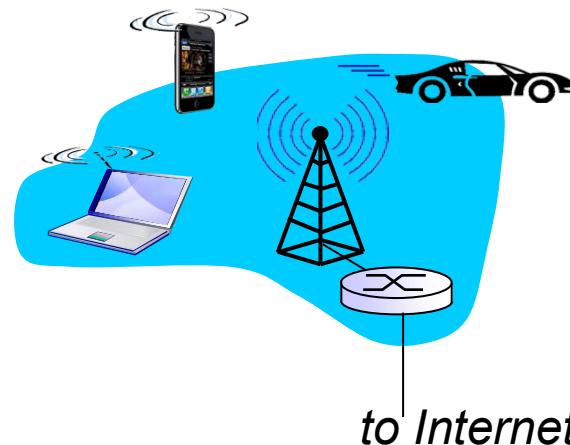
- within building (100 ft)
- 802.11b/g/n/ac (WiFi)



*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 (cable).
  - Guided media:
    - signals propagate in solid media, e.g. fiber
  - Unguided media:
    - signals propagate freely, e.g., Wi-Fi, cellular



*Twisted pair cable*



*Coaxial cable*



*Fiber optic cable*

# Lecture 1: Roadmap

1.1 What is the Internet?

1.2 Network Edge

- hosts, access networks, links

1.3 Network Core

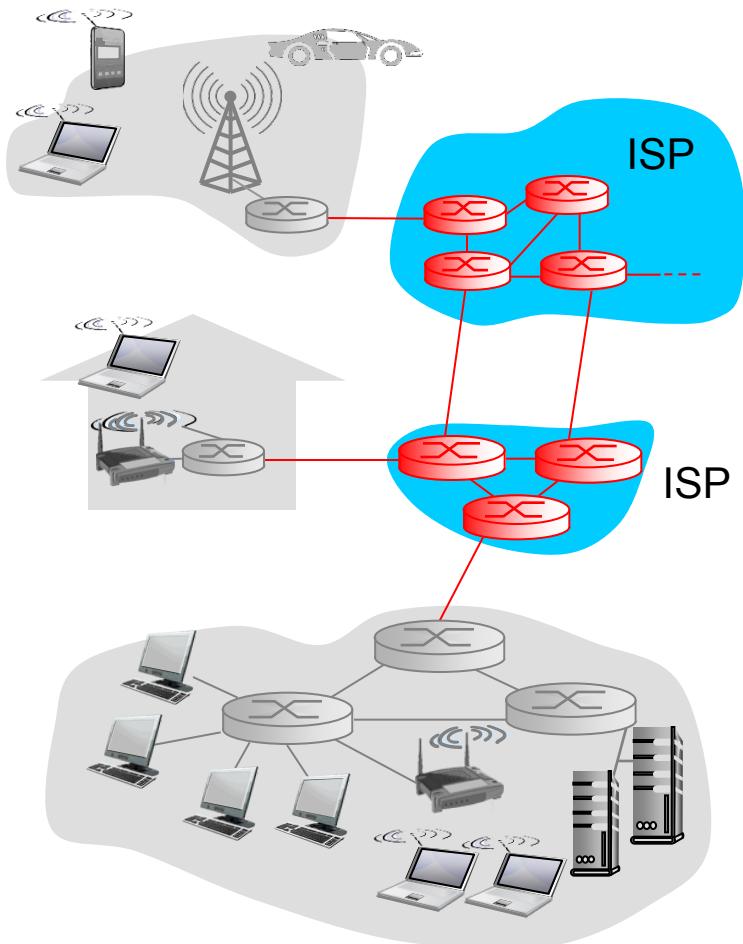
- packet switching, circuit switching, network structure

1.4 Delay, Loss and Throughput in Networks

1.5 Protocol Layers and Service Models

# The Network Core

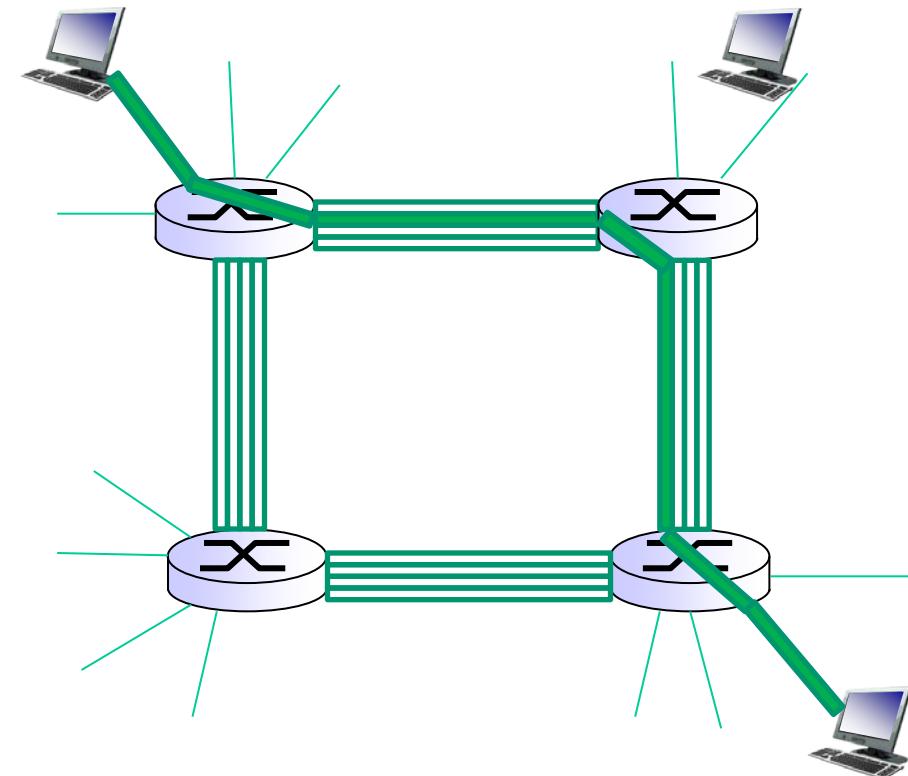
- ❖ A mesh of interconnected routers
- ❖ The fundamental question: **how is data transmitted through net?**
  - **Circuit switching:** dedicated circuit per call
  - **Packet switching:** data sent thru net in discrete “chunks”



# 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
- ❖ divide link bandwidth into “pieces”
  - frequency division
  - time division

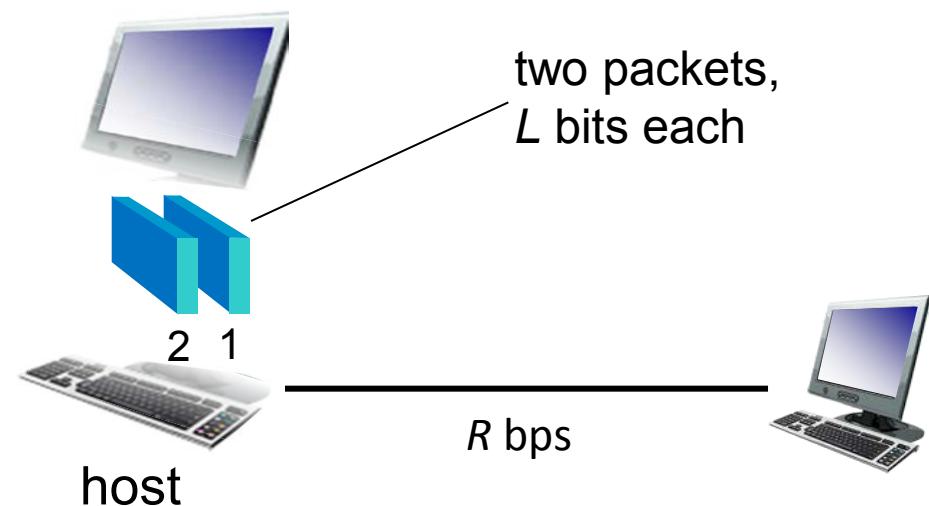


In above diagram, each link has four circuits. A “call” gets 2nd circuit in top link and 1st circuit in right link.

# Packet Switching

## Host sending function:

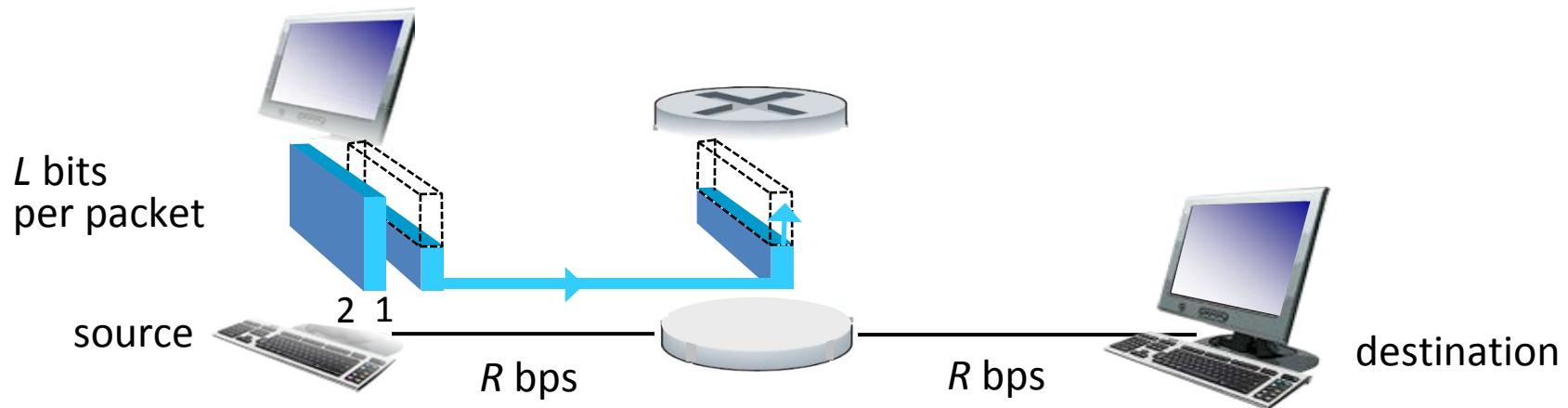
- ❖ breaks application message into smaller chunks, known as *packets*, of length  $L$  bits
- ❖ transmits packets onto the link at *transmission rate R*
  - link transmission rate is aka *link capacity* or *link bandwidth*



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

# Packet-switching: store-and-forward

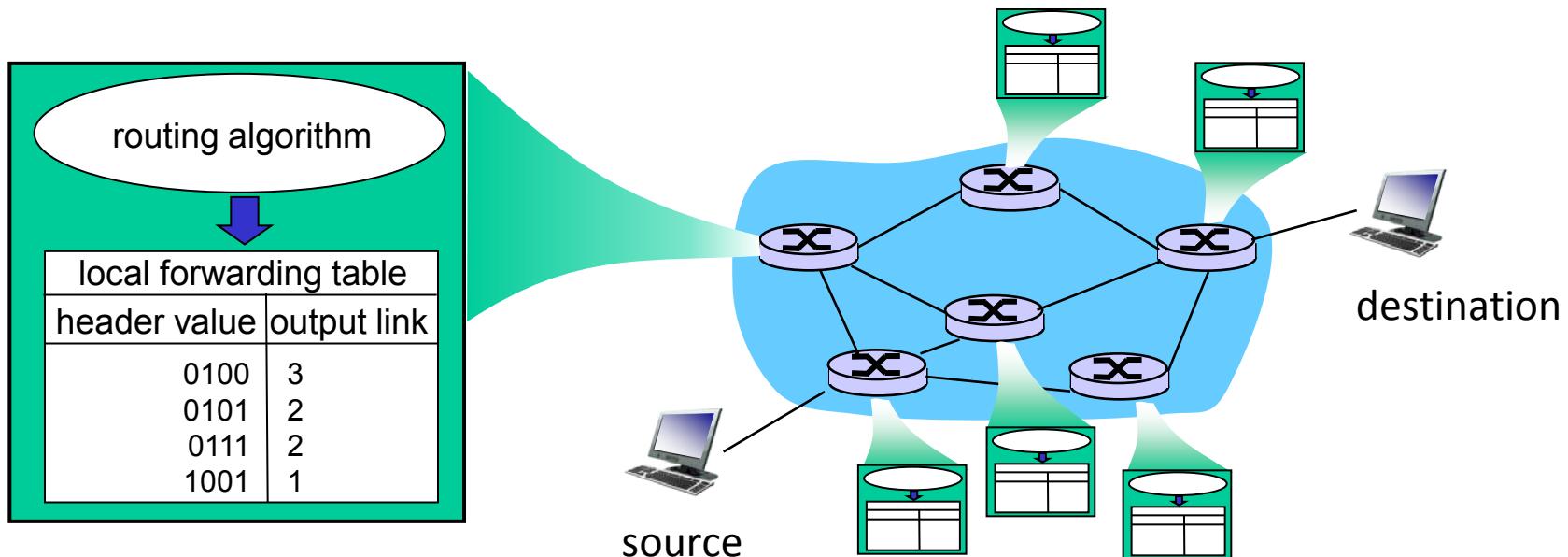
- ❖ Packets are passed from one **router** to the next, across links on path from source to destination.
- ❖ *Store and forward*: entire packet must arrive at a router before it can be transmitted on the next link.



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

# Routing and Addressing

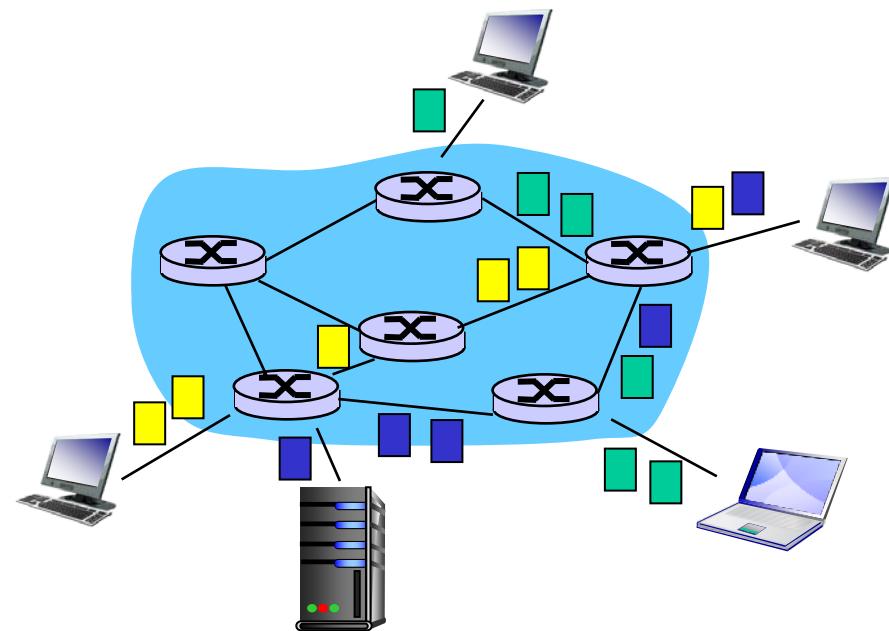
- ❖ Routers determine source-destination route taken by packets.
  - Routing algorithms
- ❖ **Addressing:** each packet needs to carry source and destination information



# 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.

# Who Runs the Internet?

- ❖ IP address & Internet Naming administered by Network Information Centre (NIC)
  - Refer to: [www.sgnic.net.sg](http://www.sgnic.net.sg); [www.apnic.org](http://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](http://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

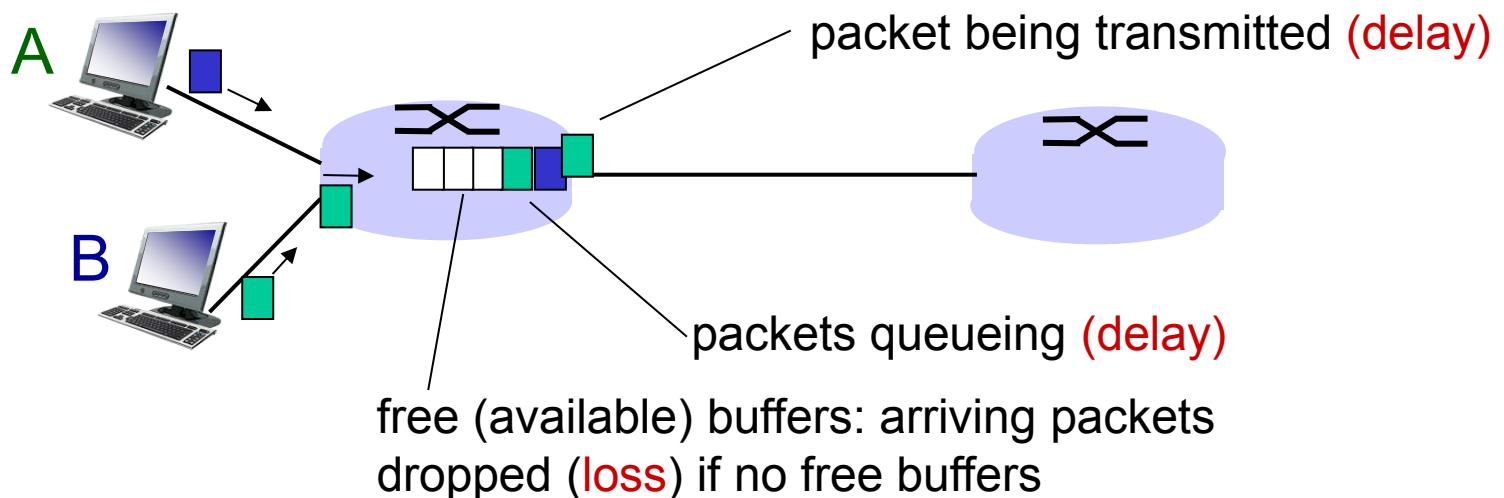
# Recall: Packet Switching Network

- ❖ 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.

# How do Delay and Loss Occur?



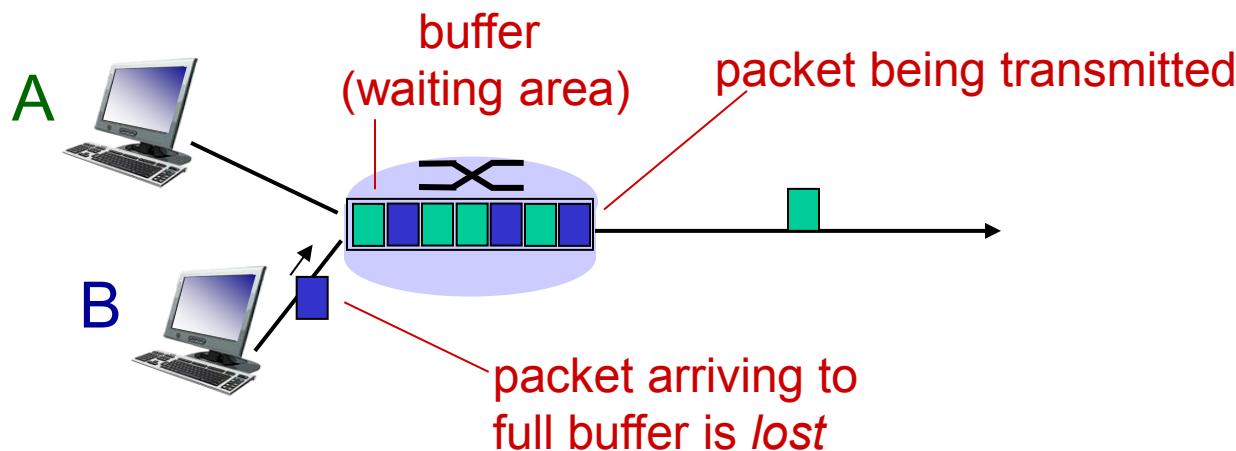
- ❖ Packets *queue* in router buffers
  - wait for turn to be sent out one by one



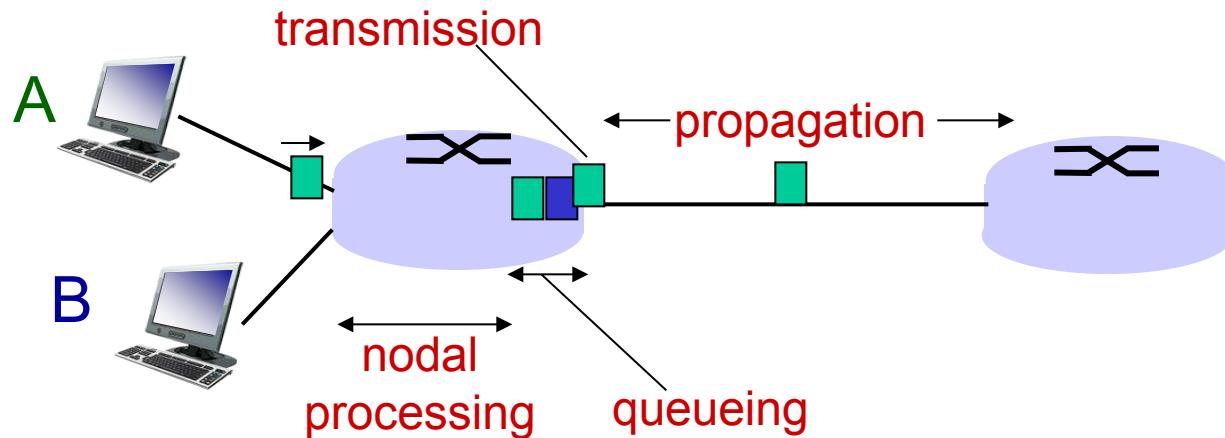
Q: What if packet arrival rate exceeds departure rate?

# Packet Loss

- ❖ Queue (aka **buffer**) of a router has finite capacity.
- ❖ Packet arriving to full queue will be dropped (aka lost).
- ❖ Lost packet may be retransmitted by previous node, by source host, or not at all.



# Four Sources of Packet Delay



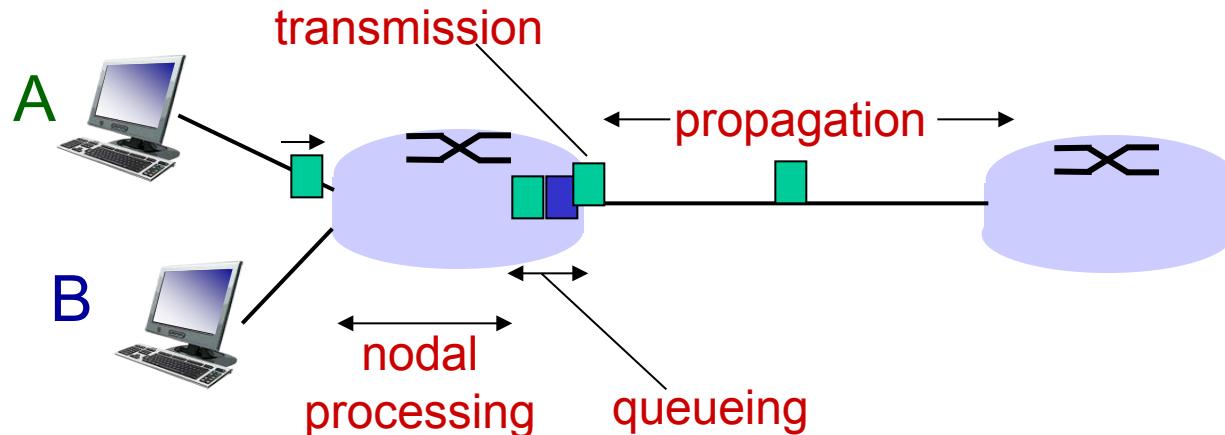
$d_{\text{proc}}$ : nodal processing

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

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

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

# Four Sources of Packet Delay



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

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

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

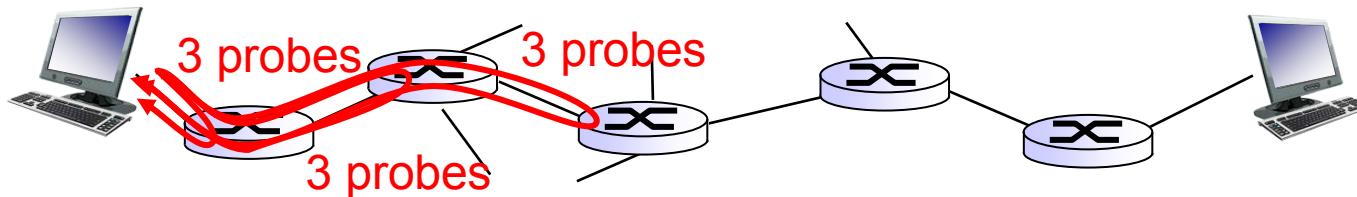
- $d$ : length of physical link
- $s$ : propagation speed in medium ( $\sim 2 \times 10^8$  m/sec)
- $d_{\text{prop}} = d/s$

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

# End-to-end Packet Delay

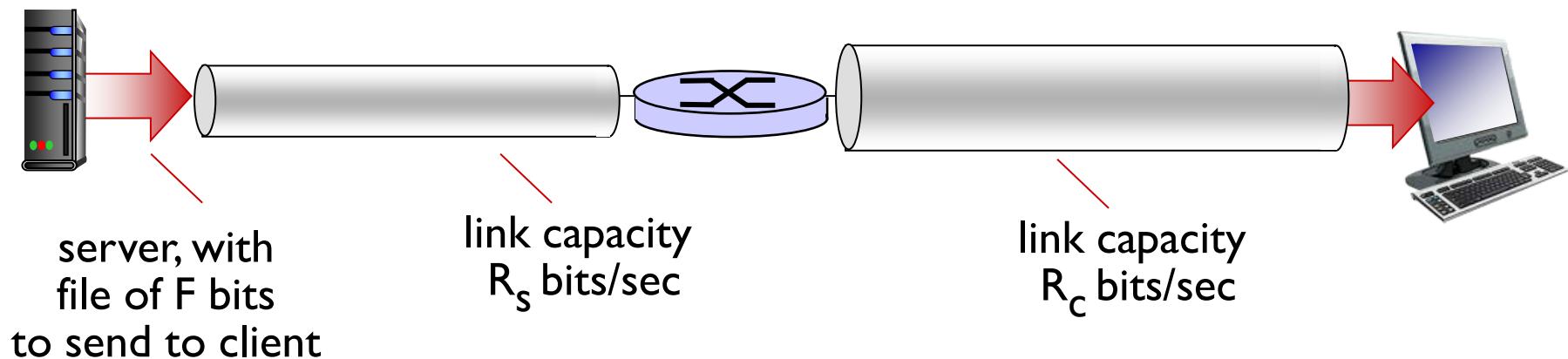
- ❖ End-to-end packet delay is the time taken for a packet to travel from source to destination. It consists of:
  - transmission delay
  - propagation delay
  - processing delay
  - queueing delay

**traceroute** program displays the route (path) from source to destination and measures the delay from source to each router along the end-end Internet path.



# 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

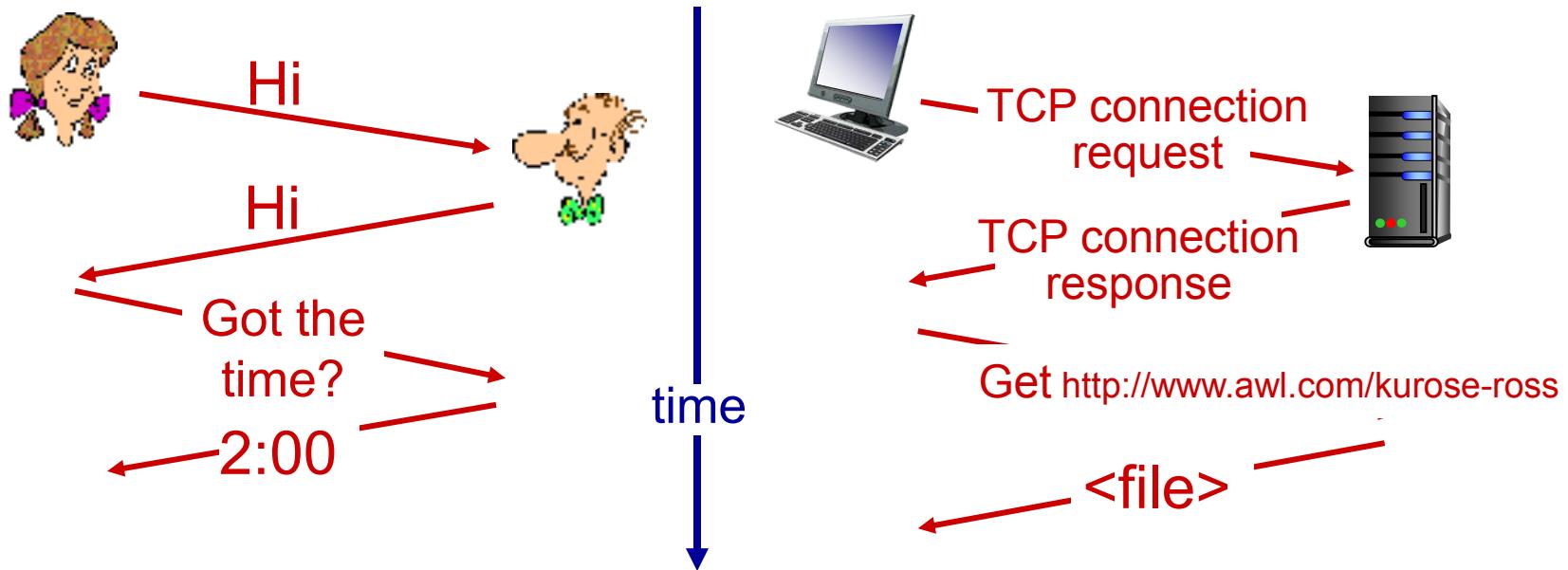
1.5 Protocol Layers and Service Models

# Internet: a Service View

- ❖ 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**.

# What's a Protocol?

a human protocol and a computer network protocol:



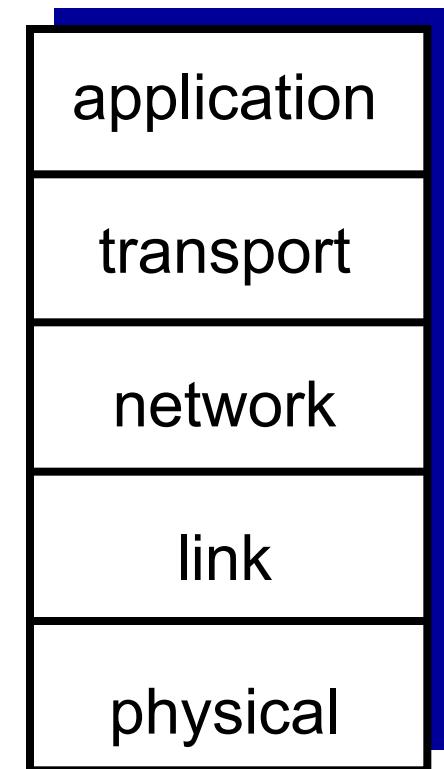
**Protocols define format and order of messages exchanged and the actions taken after messages are sent or received.**

# Protocol “Layers”

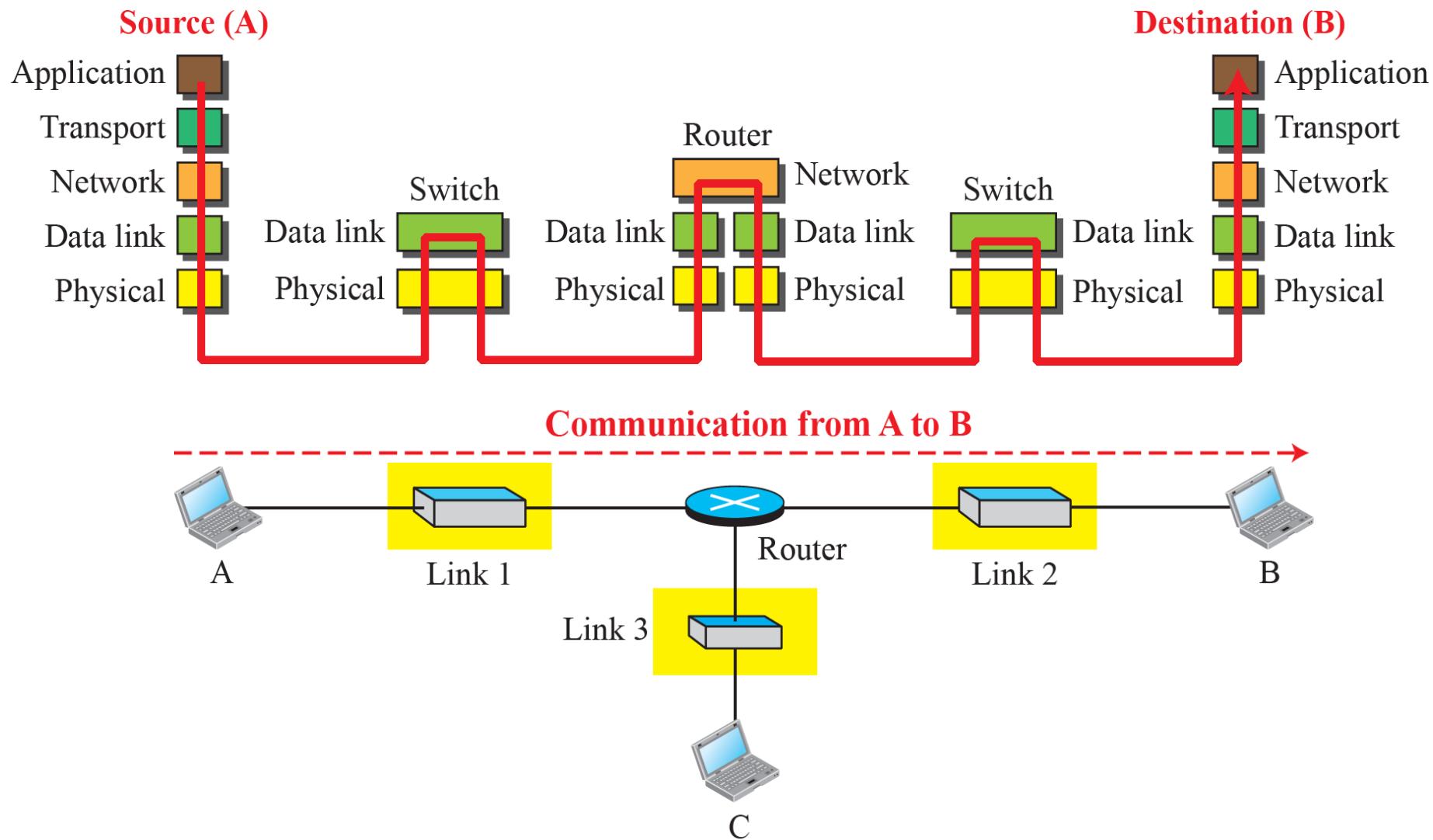
- ❖ Protocols in the Internet are logically organized into “layers” according to their purposes.
  - Each layer provides a service
  - Simple interfaces between layers
  - Hide details from each other
- ❖ Layering is a common CS trick to deal with large and complex systems.
  - Explicit structure allows identification, relationship of complex system’s pieces
  - Modularization eases maintenance, updating of system
    - E.g. change of implementation of one layer’s service is transparent to rest of system

# Internet Protocol Stack

- ❖ *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”

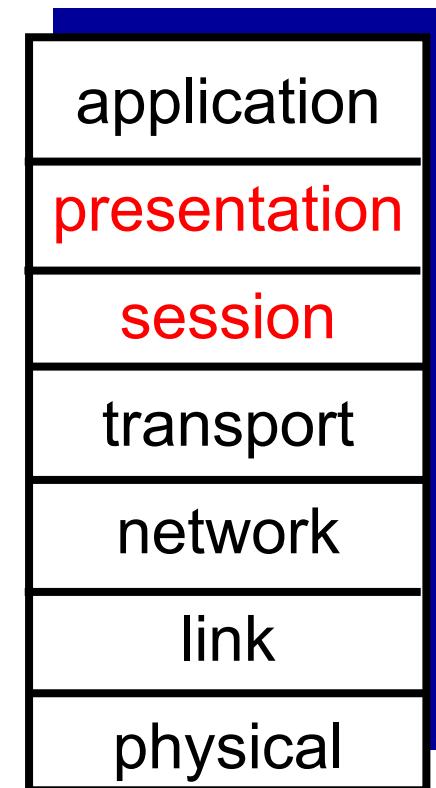


# Example



# 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:*

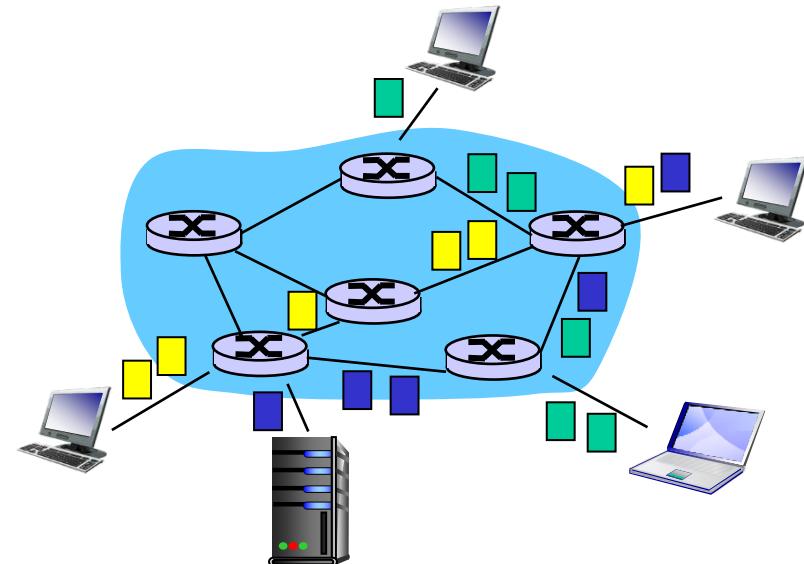
- ❖ Context, overview, “feel” of networking
- ❖ More depth, detail *to follow!*

# An *Awesome* Introduction to Computer Networks

# Packet Switching

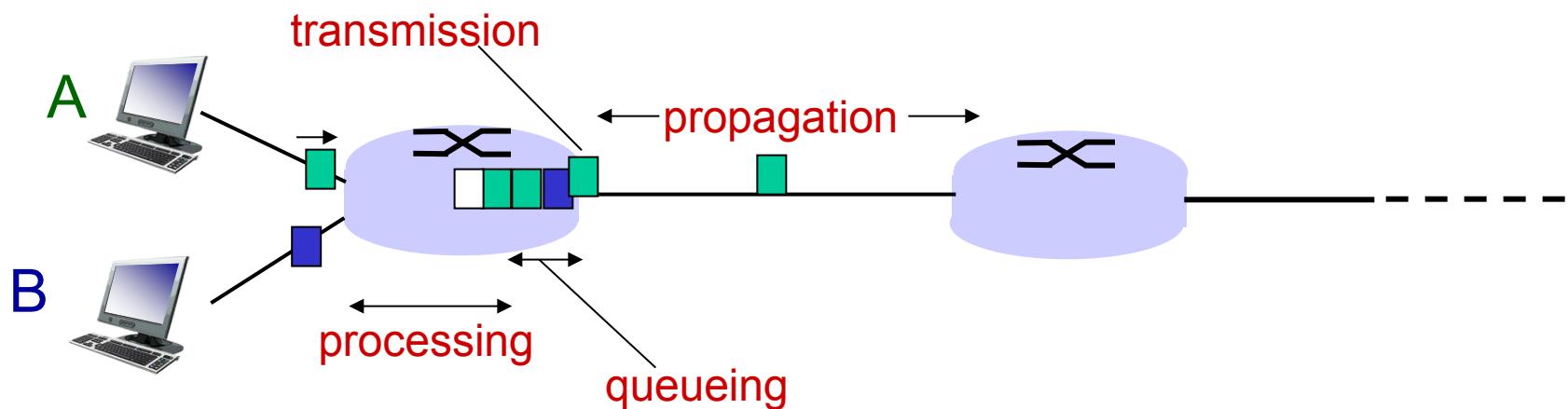
- ❖ The Internet is a **packet switching** network
  - Hosts share and contend network resources.
  - **Application message** is broken into a bunch of packets and sent onto the link one by one.
  - A router stores and forwards packets.
  - Receiver assembles all the packets to restore the **application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

**Bandwidth division into “pieces”**  
**Dedicated allocation**  
**Resource reservation**



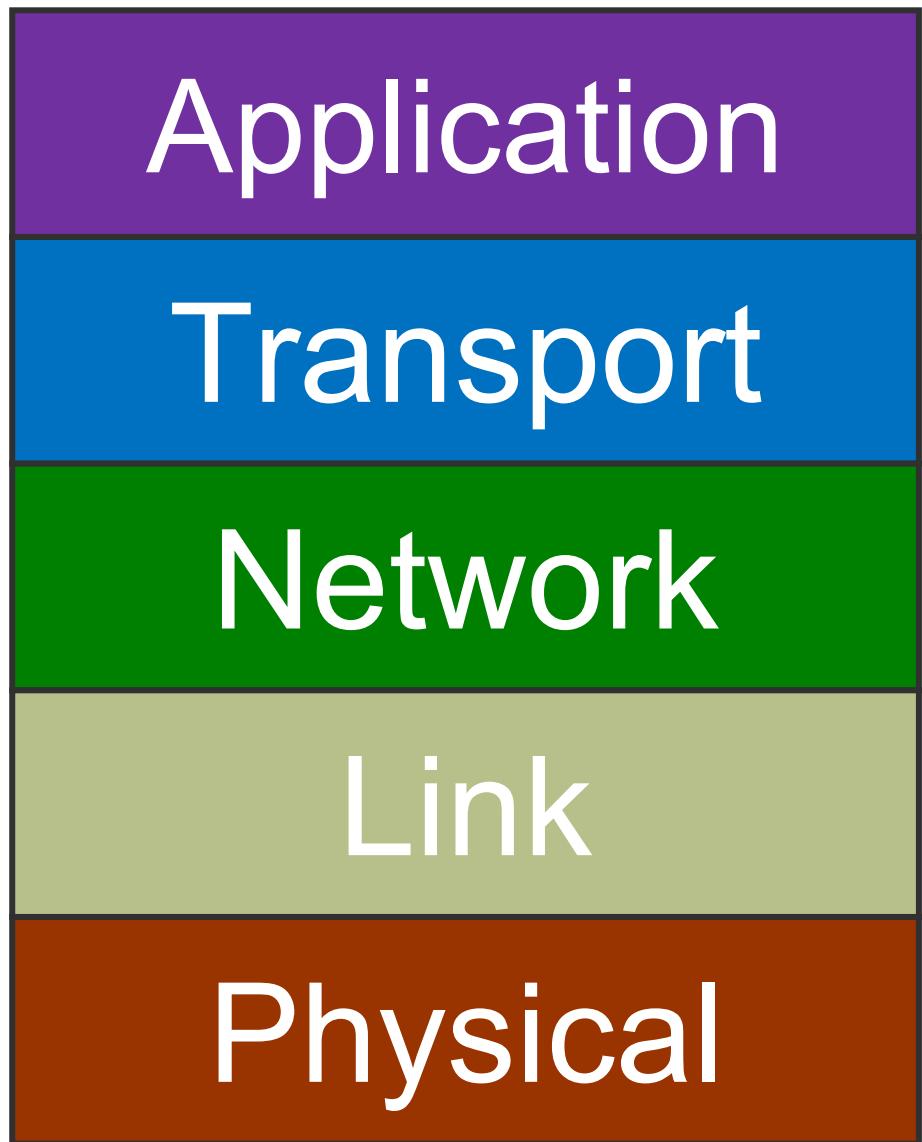
# Packet Delay

- ❖ 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.



# Lecture 2: Application Layer

*After this class, you are expected to:*

- ❖ understand the basic HTTP interactions between the client and the server, including HTTP request (GET and head fields) and HTTP response.
- ❖ understand the concepts of persistent connection, parallel HTTP connections and stateless protocol.
- ❖ understand the services provided by DNS and how a query is resolved.

# Lecture 2: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.5 DNS

2.7 Socket programming with TCP

2.8 Socket programming with UDP

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

# Application architectures

possible structure of network applications:

- ❖ Client-server
- ❖ Peer-to-peer (P2P)
- ❖ Hybrid of client-server and P2P

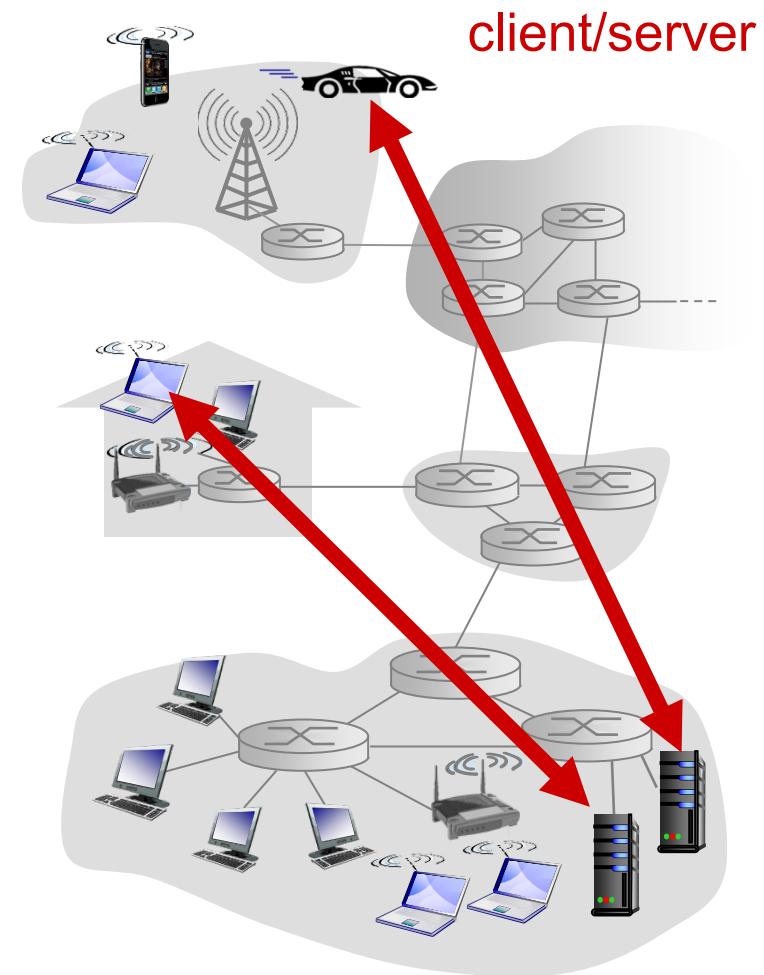
# Client-Server Architecture

## Server:

- ❖ Waits for incoming requests
- ❖ Provides requested service to client

## Client:

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

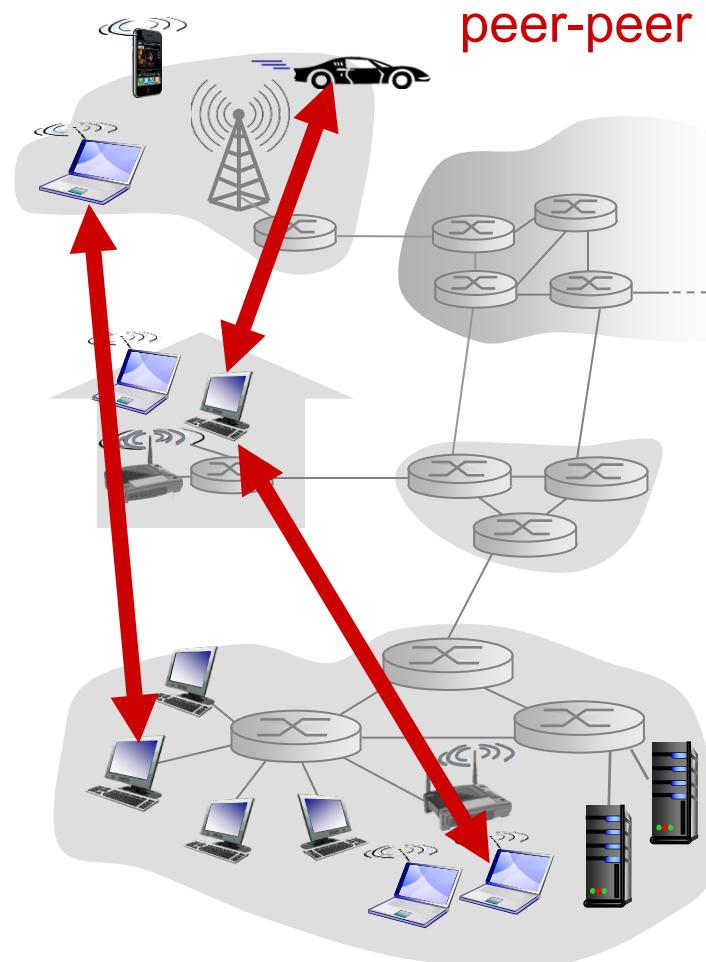


# P2P Architecture

- ❖ No always-on server
- ❖ Arbitrary end systems directly communicate.
- ❖ Peers request service from other peers, provide service in return to other peers

Highly scalable

But difficult to manage



# Hybrid of Client-Server and P2P

- ❖ Example: instant messaging
  - Chatting between two users is P2P
  - Presence detection/location is centralized:
    - User registers its IP address with central server when it comes online
    - User contacts central server to find IP addresses of buddies

# What transport service does an app need?

## Data integrity

- ❖ some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- ❖ other apps (e.g., audio 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<br>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

# Internet Transport Protocols

## TCP service:

- ❖ *reliable transport* between sending and receiving process
- ❖ *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* between sending and receiving process
- ❖ *does not provide*: reliability, flow control, congestion control, timing, throughput guarantee or security

# 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)  | TCP                           |
| Internet telephony     | SIP [RFC 3261],<br>RTP [RFC 3550],<br>or proprietary<br>(e.g., Skype) | TCP or UDP                    |

# Lecture 2: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.5 DNS

2.7 Socket programming with TCP

2.8 Socket programming with UDP

# 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`

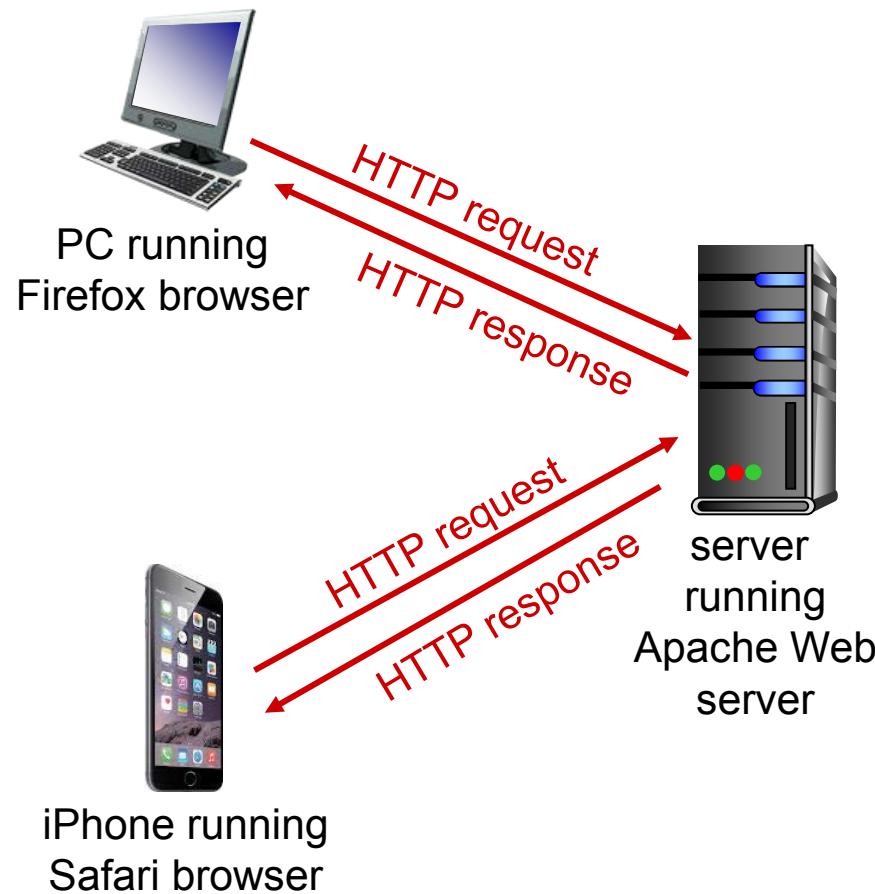
host name

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**

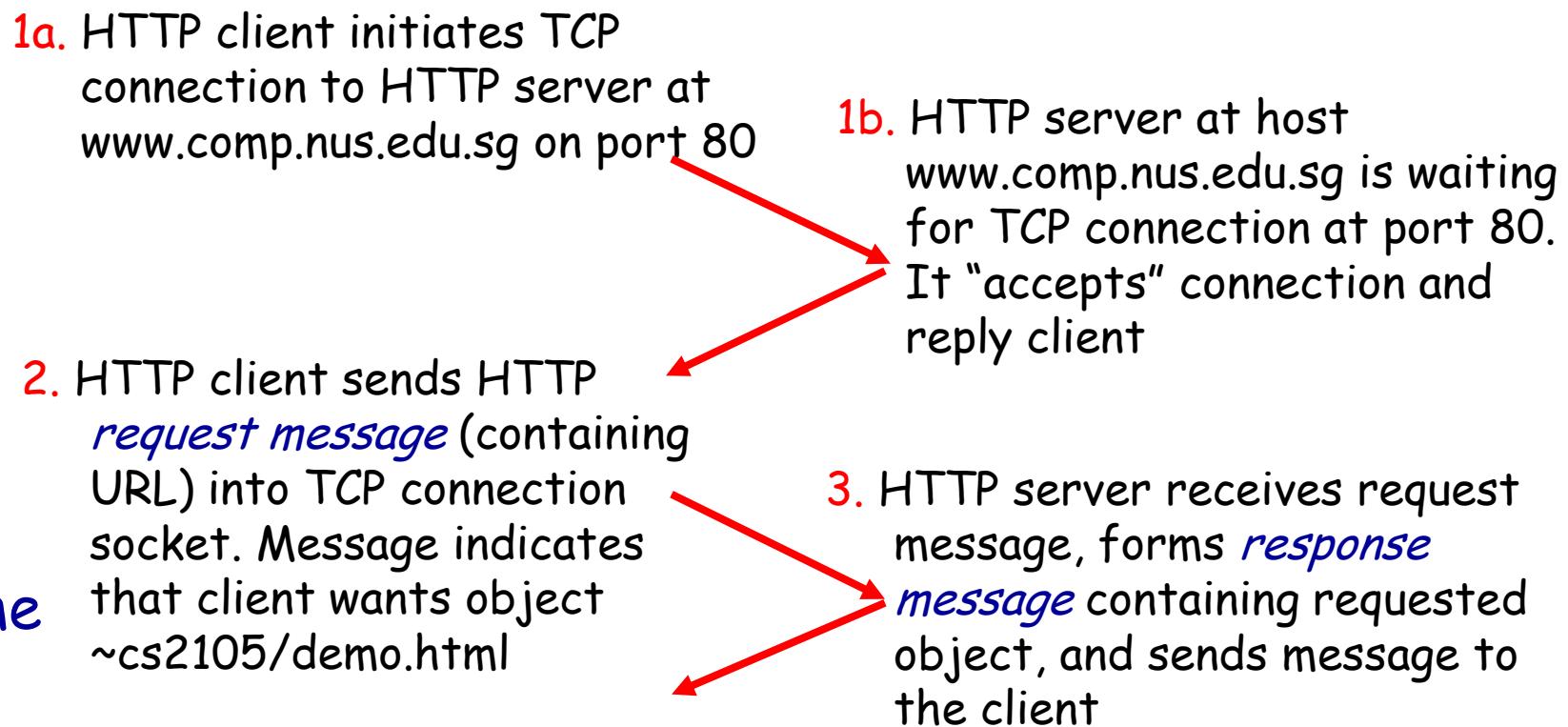
- ❖ Client initiates TCP connection to server.
- ❖ Server accepts TCP connection request from client.
- ❖ HTTP messages are exchanged between browser (HTTP client) and Web server (HTTP server) over TCP connection.
- ❖ TCP connection closed.

# Non-persistent HTTP Example

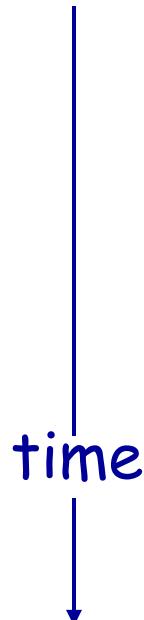
suppose user enters URL:

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

(contains text,  
reference to a  
jpeg image)



# 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).
- ❖ http 1.1 allows *persistent connection* (to discuss).

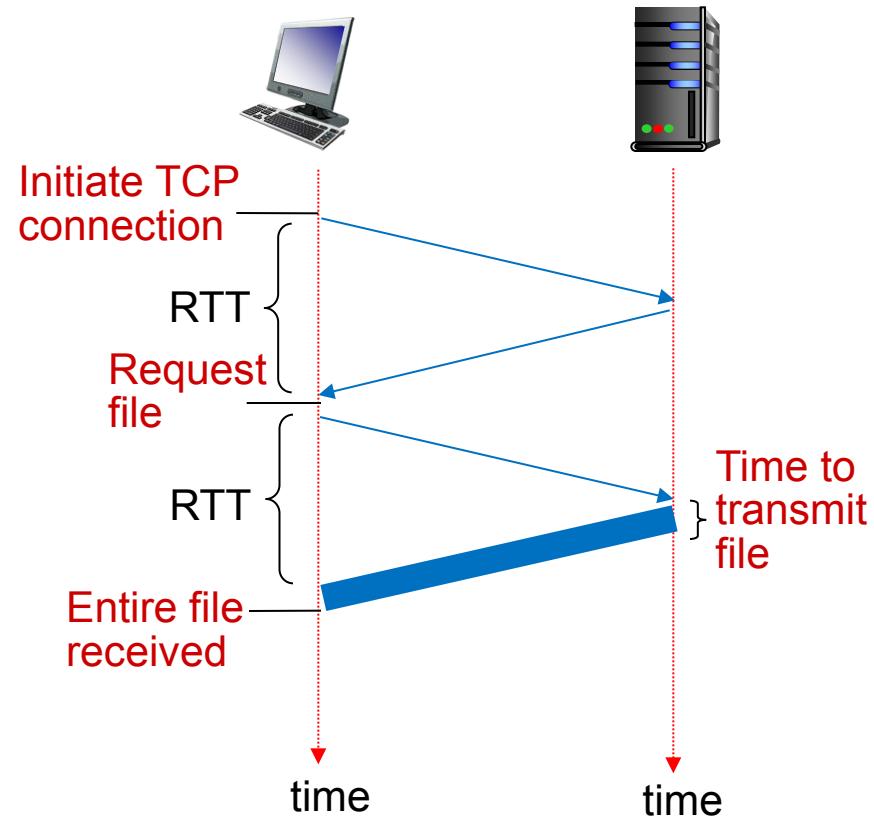
# Non-persistent HTTP: Response Time

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
- ❖ client sends 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)

**GET /~cs2105/demo.html HTTP/1.1**

header lines [ **Host: www.comp.nus.edu.sg**  
**User-Agent: Mozilla/5.0**  
**Connection: close**

\r\n

Extra "blank" line indicates  
the end of header lines

# Example HTTP Response Message

status line  
(protocol  
status code)

HTTP/1.1 200 OK

header  
lines

Date: Thu, 15 Jan 2015 13:02:41  
GMT

Server: Apache/2.4.6 (Unix)

Content-Type: text/html

\r\n

data, e.g.,  
requested  
HTML file

data data data data data ...

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 1<sup>st</sup> 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

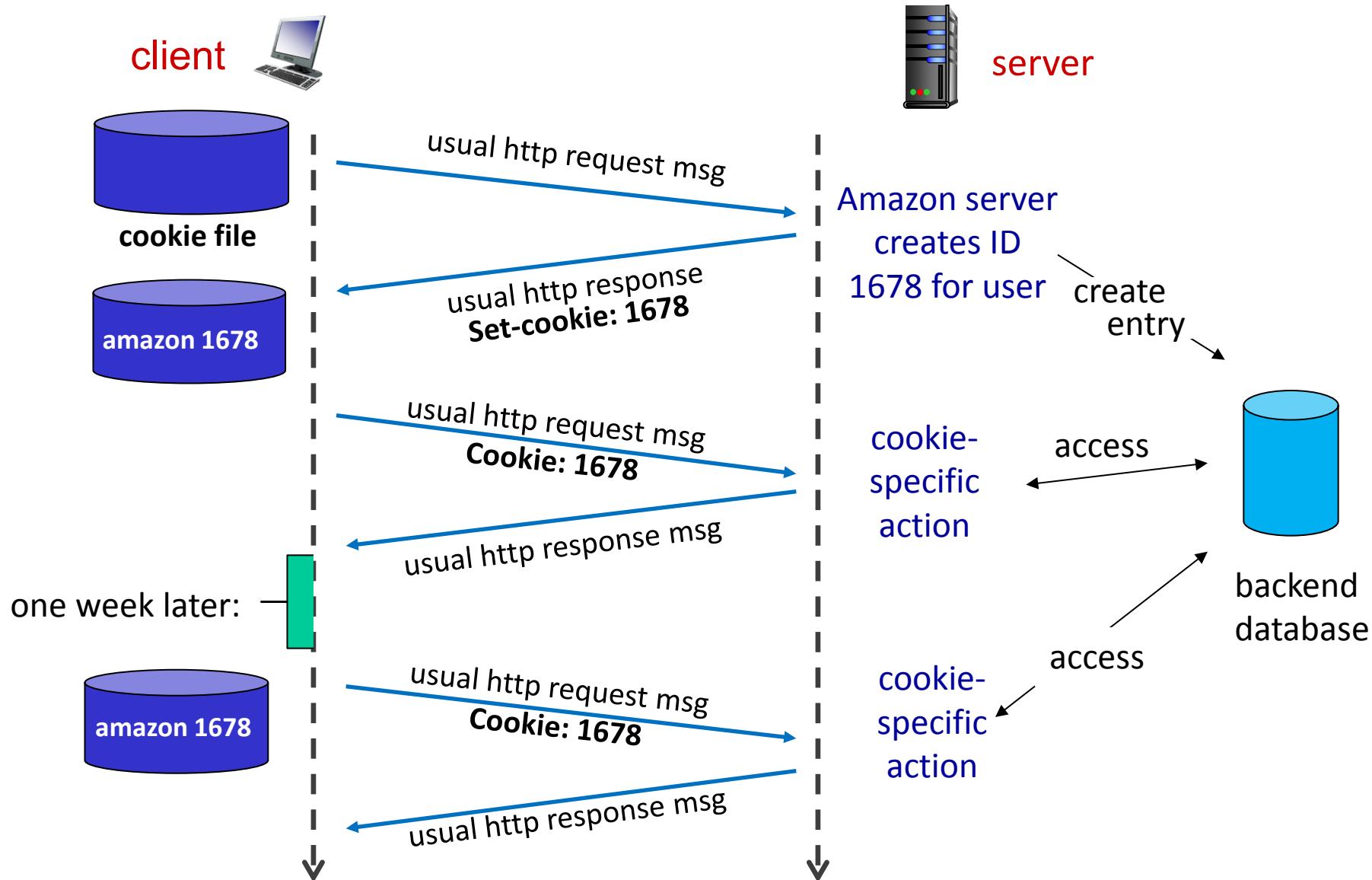
For a full list of status codes, check

<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
  - 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



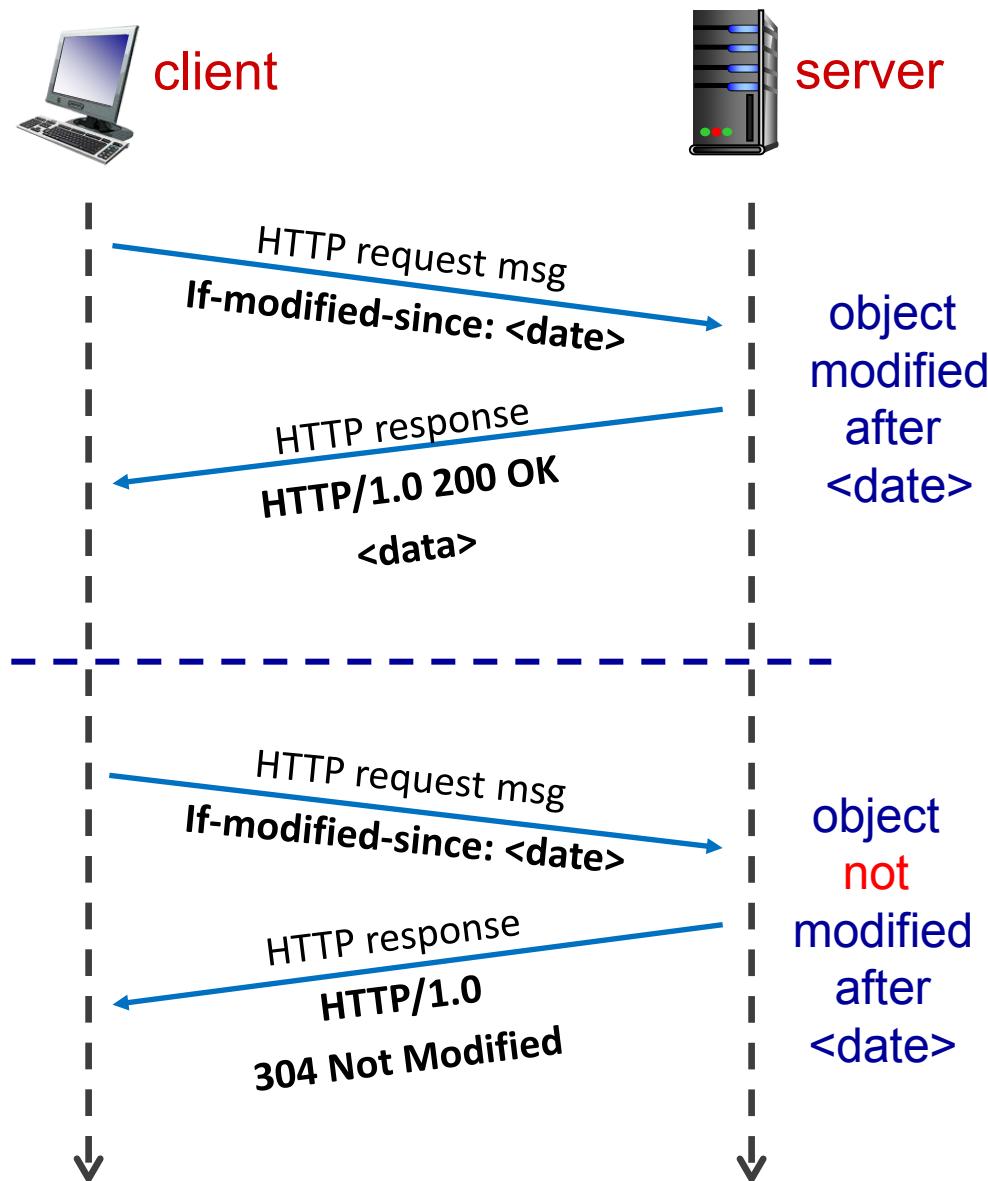
# Conditional GET

- ❖ ***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**



# Lecture 2: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.5 DNS

2.7 Socket programming with TCP

2.8 Socket programming with UDP

# Domain Name System

- ❖ Two ways to identify a host:
  - **Hostname**, e.g., www.comp.nus.edu.sg
  - **IP address**, e.g., 137.132.80.57
- ❖ **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.comp.nus.edu.sg) prior to the connection.

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

## **type = CNAME**

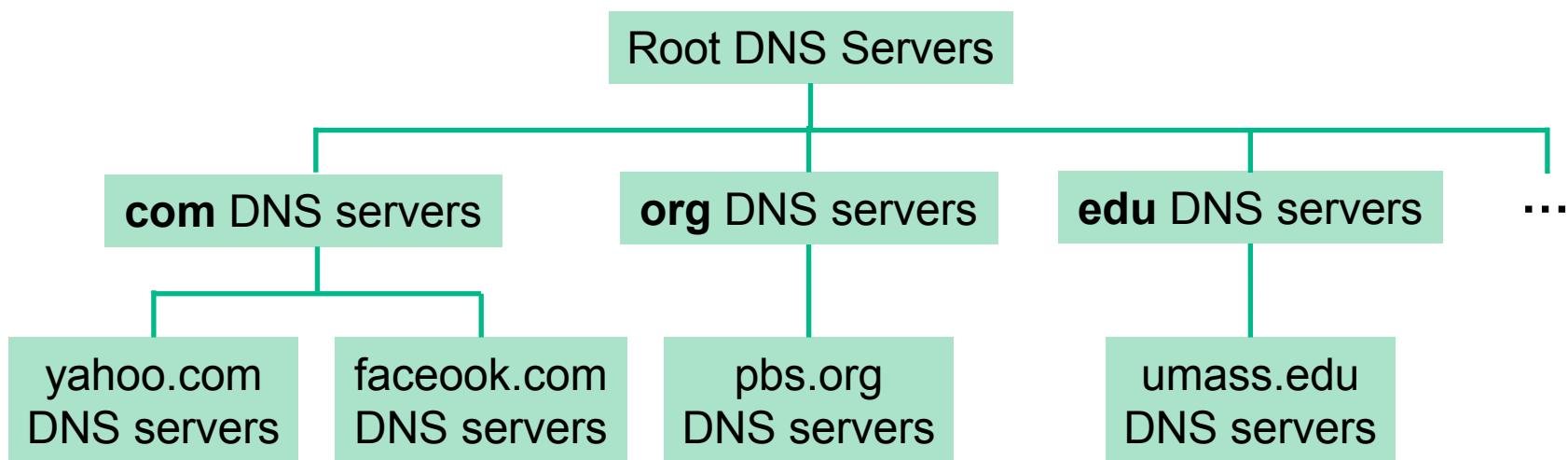
- **name** is alias name for some “canonical” (the real) name
- **value** is canonical name
- `www.comp.nus.edu.sg` is really `www0.comp.nus.edu.sg`

## **type = MX**

- **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.

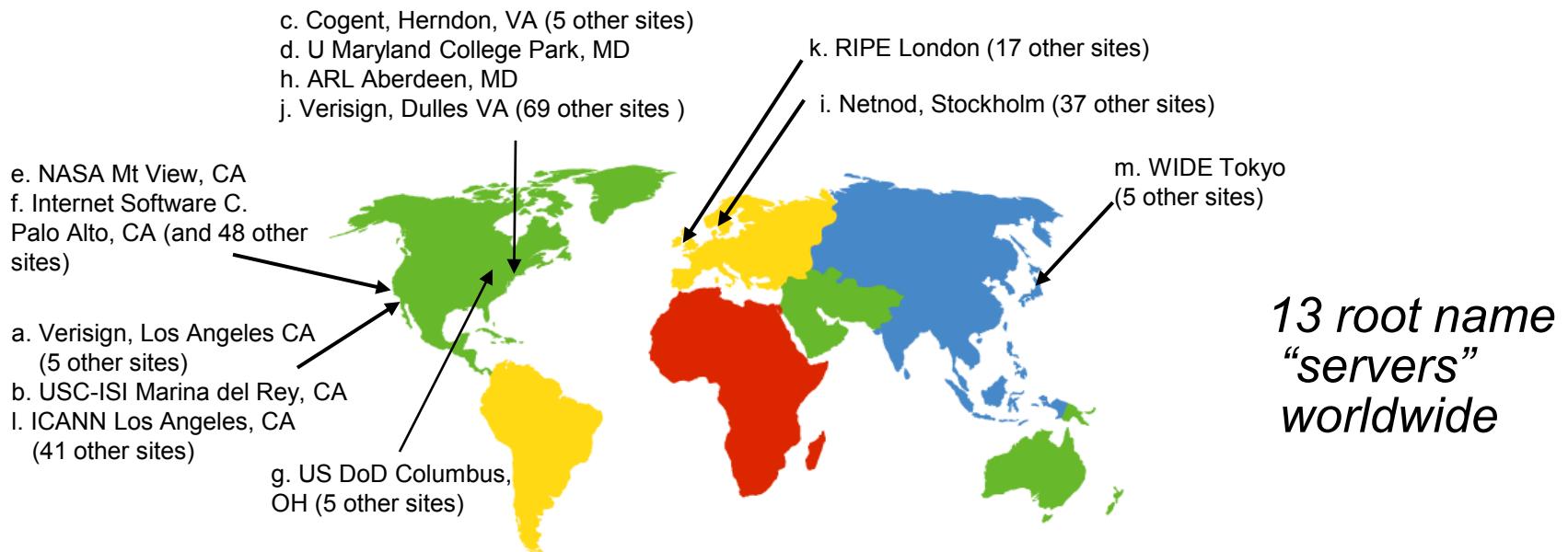


*A client wants IP address for www.facebook.com:*

- ❖ 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

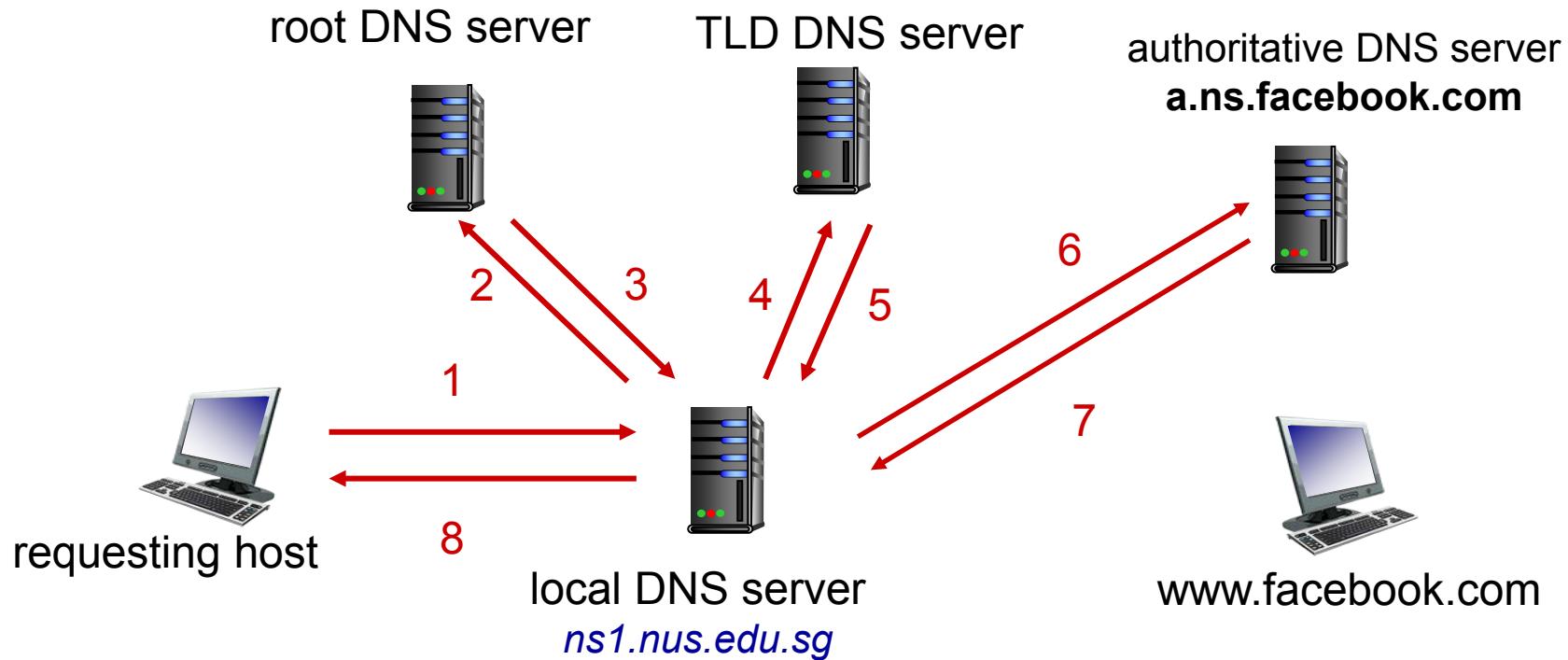
## *Authoritative servers:*

- ❖ 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 Query



- ❖ Once a name server learns mapping, it *caches* mapping
  - cache entries expire after some time (TTL).
- ❖ DNS runs over **UDP**.

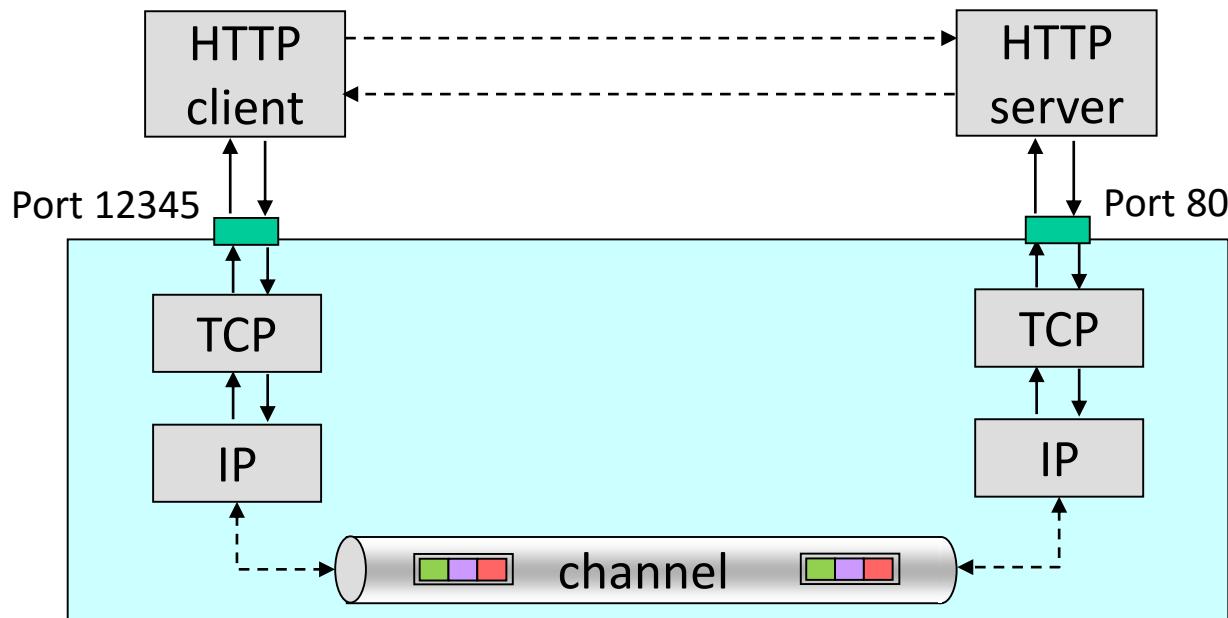
# Lecture 2: Summary

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

# An Awesome Introduction to Computer Networks

# 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**!



# HTTP Connections

## *HTTP 1.0: non-persistent*

- ❖ At most one object is sent over one TCP connection.
  - connection is then closed.
- ❖ Downloading multiple objects requires multiple connections.
  - TCP connections may be launched in parallel

## *HTTP 1.1: persistent*

- ❖ Server leaves connection open after sending a Web object.
- ❖ Multiple objects can be sent over a single TCP connection.
  - Requests may be sent in parallel

# Lecture 3: Socket Programming

*After this class, you are expected to:*

- ❖ understand the concept of socket.
- ❖ be able to write simple client/server programs through socket programming.

# Lecture 3: Roadmap

2.1 Principles of Network Applications

2.2 Web and HTTP

2.5 DNS

2.7 Socket programming with TCP

2.8 Socket programming with UDP

# Processes

- ❖ Applications runs in hosts as **processes**.
  - 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

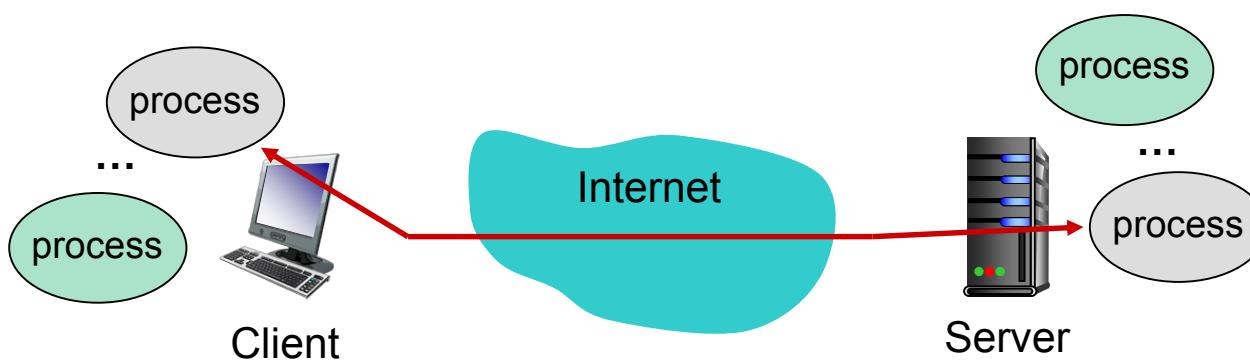
*server process* waits to be contacted

*client process* initiates the communication

# Addressing Processes

- ❖ IP address is used to identify a host device
  - 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
  - Mail 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:*

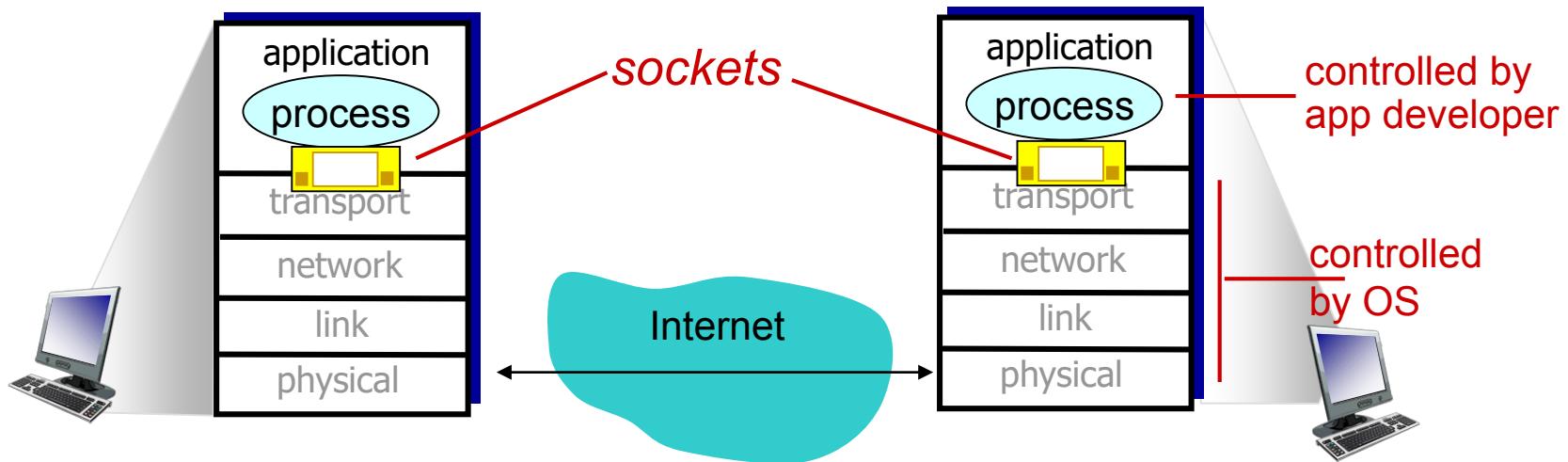
- ❖ *deliver letter to the doorstep:* home address
- ❖ *dispatch letter to the right person in the house:* name of the receiver as stated on the letter

## *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

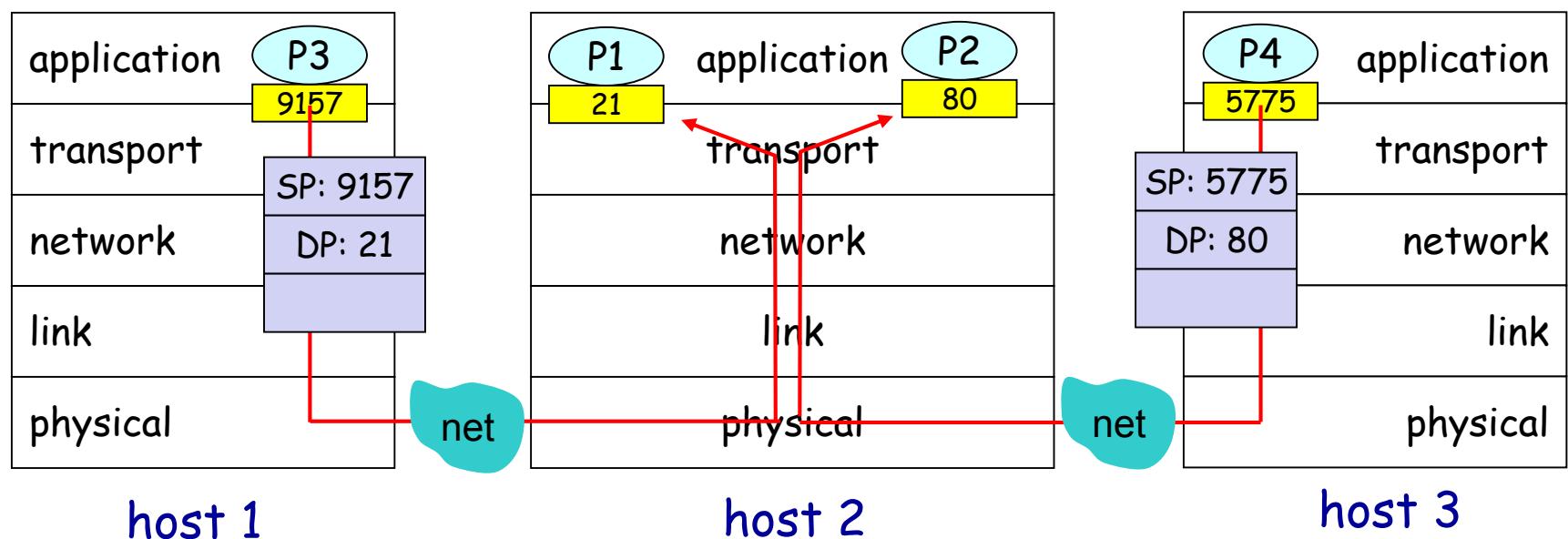
# Sockets

- ❖ **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 **API** calls



# Multiplexing / de-multiplexing

= socket      = process



use IP address + port number to locate a process

# Socket Programming



- ❖ Applications (or processes) treat the Internet as a black box, sending and receiving messages through sockets.
- ❖ Two types of sockets
  - **stream socket** (aka TCP socket) that uses **TCP** as its transport layer protocol.
    - Connection-oriented, reliable
  - **datagram socket** (aka UDP socket) that uses **UDP**.
    - Connection-less, unreliable (transmitted data may be lost, corrupted or received out-of-order)

# TCP Socket and UDP 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.**

## Client must contact server

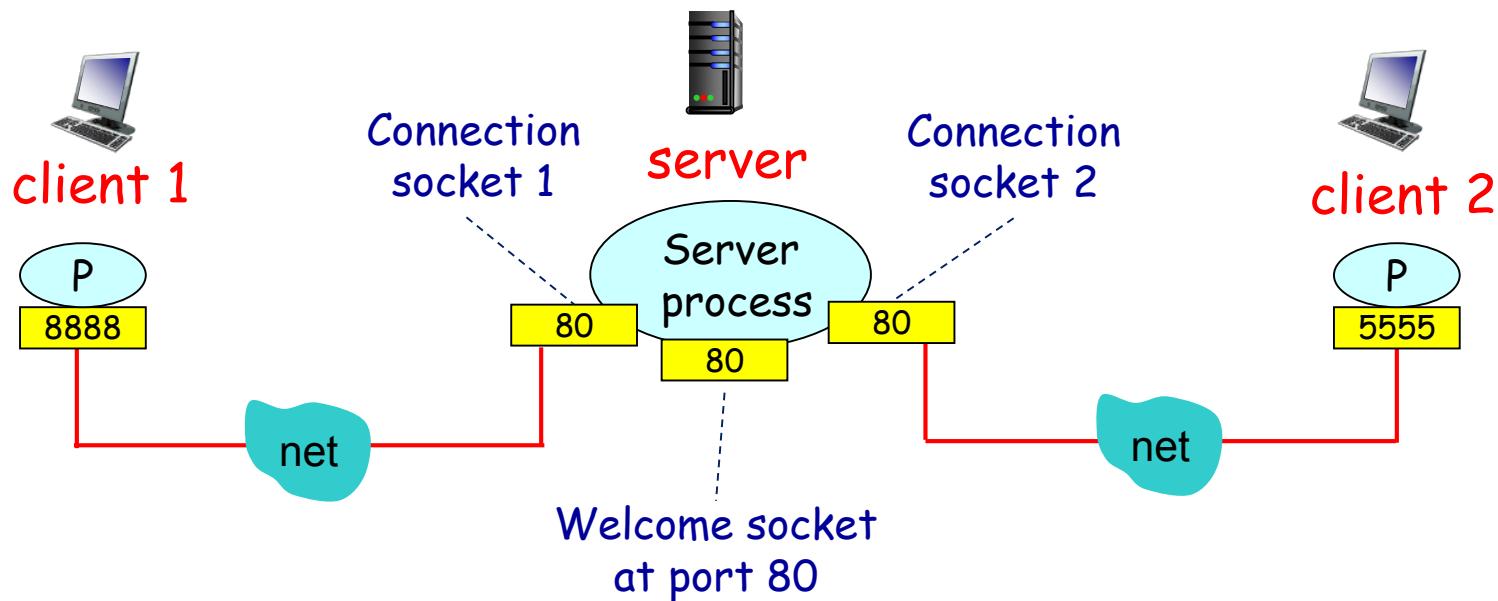
- ❖ server process must first be running
- ❖ server must have created socket that waits for client's contact

## Client contacts server by

- ❖ creating client-local socket
- ❖ specifying IP address and port number of server process

# Socket Programming with *TCP*

- ❖ With TCP sockets, a process **establishes a connection** to another process.
- ❖ While the connection is in place, data flows between the processes in continuous streams.
- ❖ When contacted by client, **server TCP creates a new socket** for server process to communicate with client.



# TCP: Client/server Socket Interaction

Server (running on `hostid`)

create socket, port = `x`,  
for incoming request:

```
welcomeSocket = ServerSocket(x)
```

wait for incoming  
connection request

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

read request from  
`connectionSocket`

write reply to  
`connectionSocket`

close `connectionSocket`

Client

create socket,  
connect to `hostid`, port=`x`

```
clientSocket = Socket(hostid, x)
```

send request using `clientSocket`

read reply from `clientSocket`

close `clientSocket`

TCP  
connection setup

# Example: TCP Echo Server (1/2)

```
import java.io.*;
import java.net.*; ← This package defines Socket
import java.util.*;

class SimpleTCPEchoServer {

    public static void main(String[] args) throws IOException {

        int port = 5678; // server listens to this example port

        // server is waiting
        ServerSocket welcomeSocket = new ServerSocket(port);

        while (true) { // server is always alive
            Socket connectionSocket = welcomeSocket.accept(); ← accept() method returns a
// to continue next page                                         new socket to communicate
// with client socket
```

# Example: TCP Echo Server (2/2)

```
System.out.println("Connected to a client...");  
  
read from  
socket → Scanner scanner = new  
          Scanner(connectionSocket.getInputStream());  
          // read data from the connection socket  
          String fromClient = scanner.nextLine();  
  
write to  
socket → PrintWriter toClient = new PrintWriter(  
          connectionSocket.getOutputStream(), true);  
          // write data to the connection socket  
          toClient.println(fromClient);  
          }  
      }  
  }  
}  
end of while loop,  
loop back and wait for  
another client connection
```

# Example: TCP Echo Client (1/2)

```
import java.io.*;
import java.net.*;
import java.util.*;

class SimpleTCPEchoClient {

    public static void main(String[] args) throws IOException {

        String serverIP = "127.0.0.1"; // local host, example
        int serverPort = 5678;           // just an example

        // create a client socket and connect to the server
        Socket clientSocket = new Socket(serverIP, serverPort);

        // read user input from keyboard
        Scanner scanner = new Scanner(System.in);
        String fromKeyboard = scanner.nextLine();

        // to continue next page
```

# Example: TCP Echo Client (2/2)

```
// create output stream to server
PrintWriter toServer = new
    PrintWriter(clientSocket.getOutputStream(), true);
// write user input to the socket
toServer.println(fromKeyboard);

// create input stream from server
Scanner sc =
    new Scanner(clientSocket.getInputStream());
// read server reply from the socket
String fromServer = sc.nextLine();

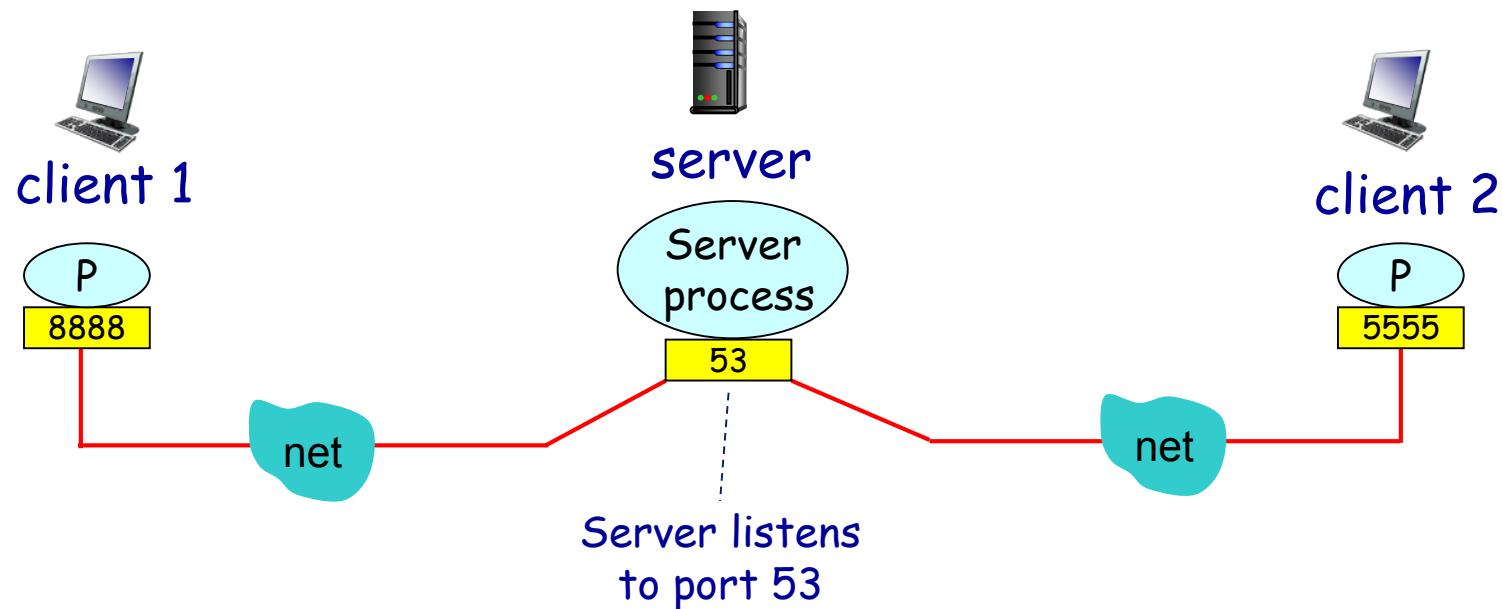
// show on screen
System.out.println("Echo from server: " + fromServer);

clientSocket.close();
}
```

# Socket Programming with *UDP*

UDP: no “connection” between client and server

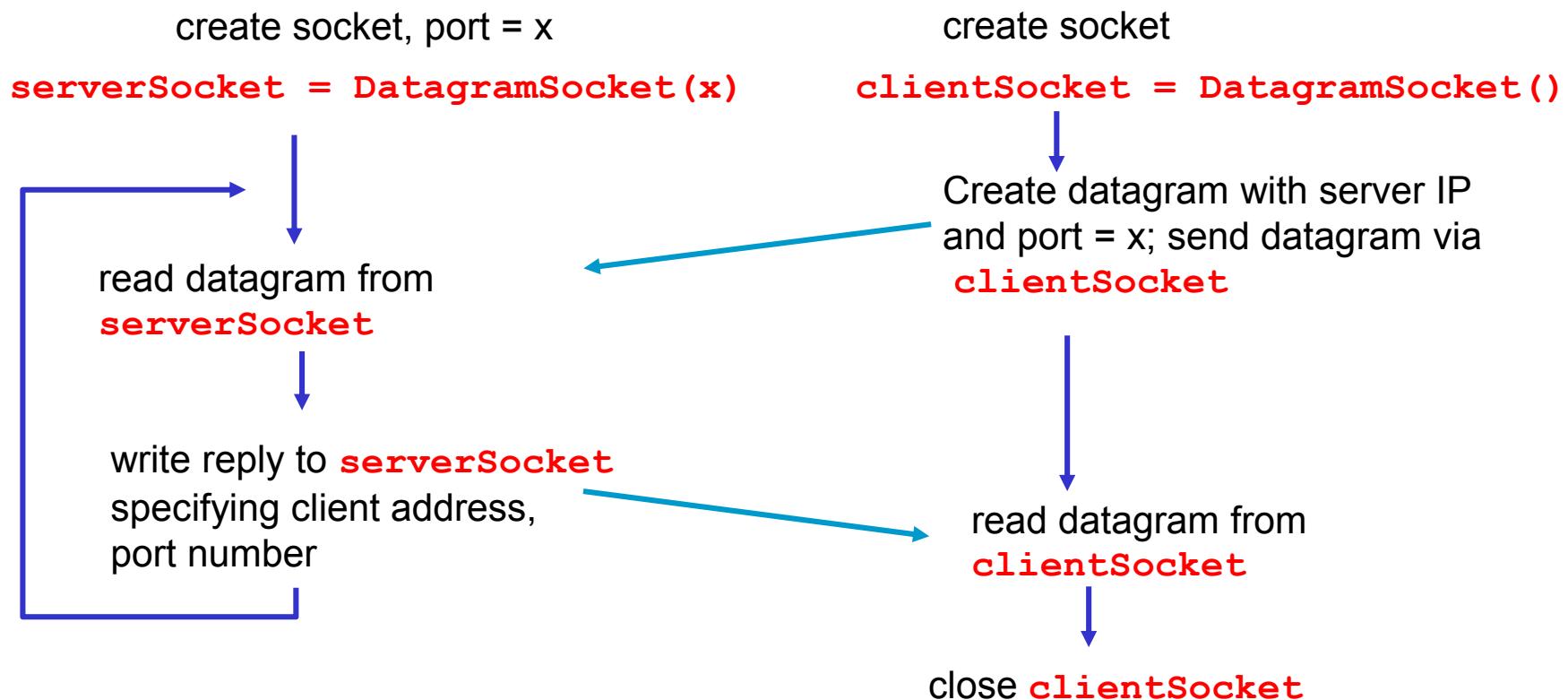
- ❖ Sender (client) explicitly attaches destination IP address and port number to every packet.
- ❖ Receiver (server) extracts sender IP address and port number from the received packet.



# UDP: Client/server Socket Interaction

Server (running on hostid)

Client



# Example: UDP Echo Server (1/2)

```
import java.io.*;
import java.net.*;

class SimpleUDPEchoServer {

    public static void main(String[] args) throws IOException {

        int port = 5678; // server listens to this example port
        DatagramSocket serverSocket = new DatagramSocket(port);

        byte[] recvBuffer = new byte[1024];

        while (true) { // server is always alive
            DatagramPacket rcvedPkt = new
                DatagramPacket(recvBuffer, recvBuffer.length);

            serverSocket.receive(rcvedPkt);
        }
    }
}
```

// to continue next page

receive() method blocks till a packet is received

# Example: UDP Echo Server (2/2)

```
String rcvedData = new String(rcvedPkt.getData(),  
                           0, rcvedPkt.getLength());
```

extract client  
address and  
port number

```
InetAddress clientAddress = rcvedPkt.getAddress();  
int clientPort = rcvedPkt.getPort();
```

```
byte[] sendData = rcvedData.getBytes();
```

```
DatagramPacket sendPkt =  
    new DatagramPacket(sendData, sendData.length,  
                      clientAddress, clientPort);
```

```
serverSocket.send(sendPkt);
```

```
}
```

end of while loop,  
loop back and wait for  
another client connection

# Example: UDP Echo Client (1/2)

```
import java.io.*;
import java.net.*;
import java.util.*;

class SimpleUDPEchoClient {

    public static void main(String[] args) throws IOException {
        InetAddress serverAddress =           // server IP address
            InetAddress.getByName("localhost");
        int serverPort = 5678;                  // just an example

        // create a client socket
        DatagramSocket clientSocket = new DatagramSocket();

        // read user input from keyboard
        Scanner scanner = new Scanner(System.in);
        String fromKeyboard = scanner.nextLine();

        // to continue next page
    }
}
```

translate hostname to IP address using DNS

// server IP address

// just an example

create a client socket

# Example: UDP Echo Client (2/2)

```
// create a datagram and send to server
byte[] sendData = fromKeyboard.getBytes();
DatagramPacket sendPkt = new DatagramPacket(sendData,
                                             sendData.length, serverAddress, serverPort);

clientSocket.send(sendPkt);           create a datagram to send

// receive a packet sent by server from socket
byte[] rcvBuffer = new byte[1024];
DatagramPacket rcvedPkt = new DatagramPacket(rcvBuffer,
                                              rcvBuffer.length);

clientSocket.receive(rcvedPkt);

System.out.println("Echo from server: " +
                    new String(rcvedPkt.getData(), 0,
                               rcvedPkt.getLength()));

read a datagram
from server

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 address and the 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.

# Lecture 3: Summary

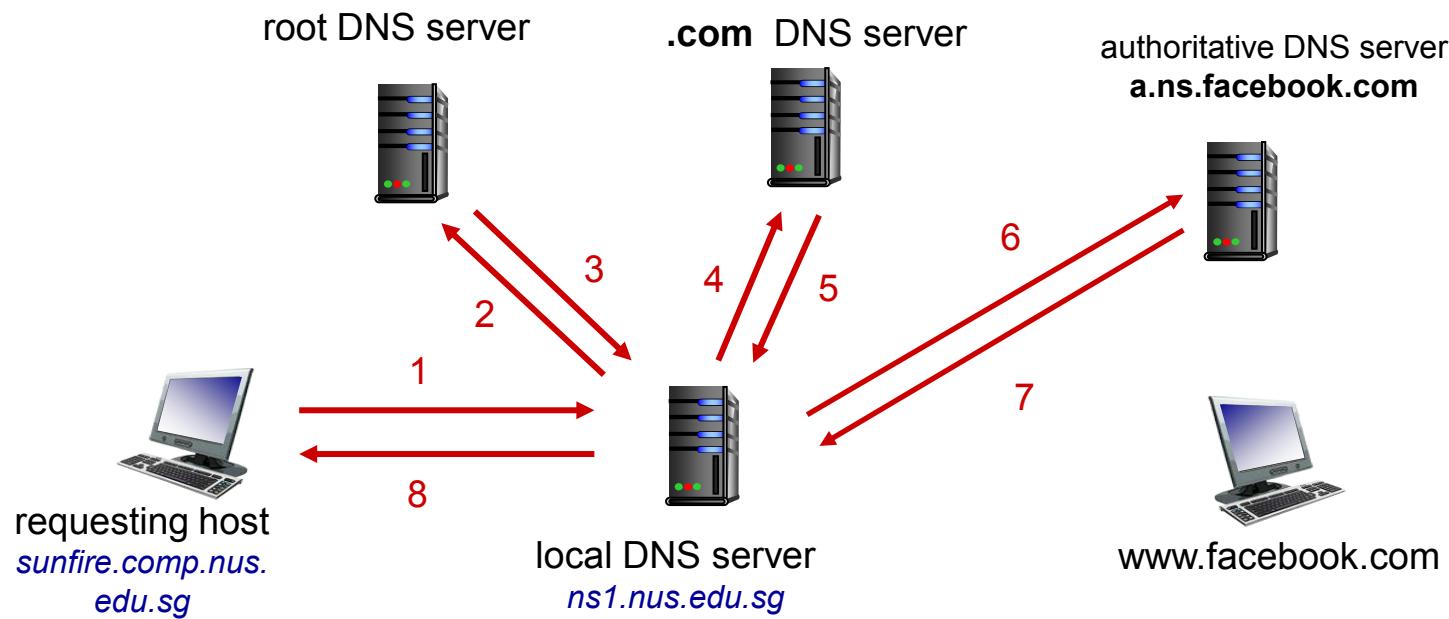
## ❖ Socket programming

- **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 use **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.

# An ~~Awesom~~e Introduction to Computer Networks

# 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.



# Socket

- ❖ Applications (processes) send messages over the network through sockets.
  - Conceptually, socket = IP address + port number
  - Programming wise, socket = a set of APIs
- ❖ TCP and UDP sockets
  - TCP socket (**stream socket**) uses **TCP** as transport layer protocol.
    - Connection-oriented, reliable
  - UDP socket (**datagram socket**) uses **UDP**.
    - Connection-less, unreliable (transmitted data may be lost, corrupted or received out-of-order)

# Lecture 4: UDP, Reliable Protocol

*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.

# Lecture 4: Roadmap

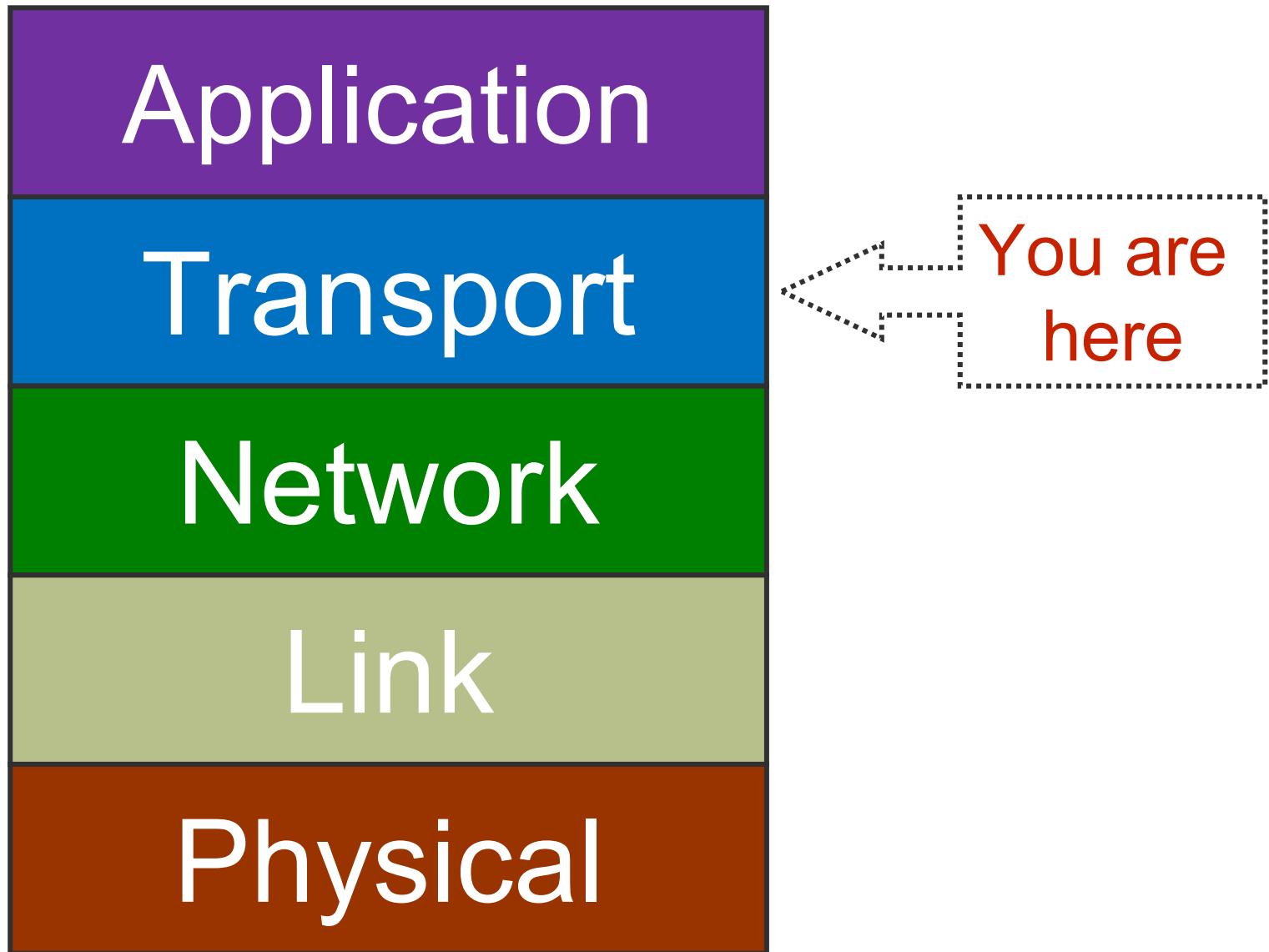
3.1 Transport-layer Services

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)

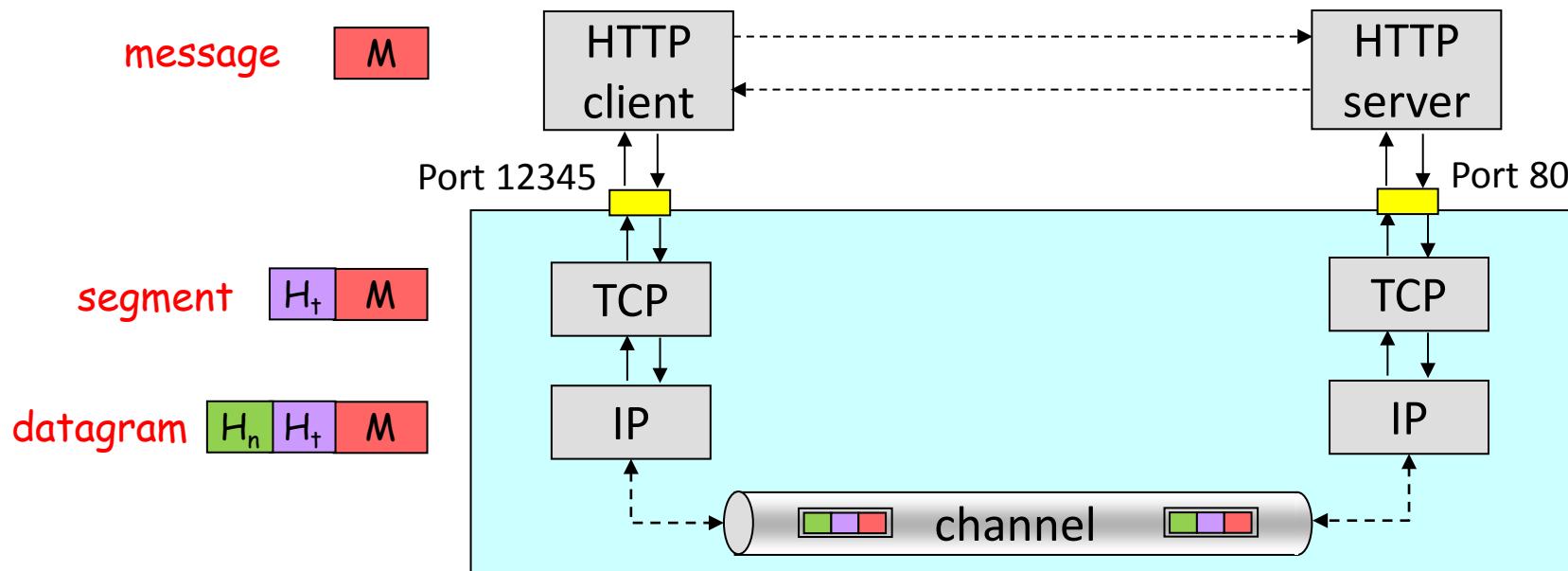


# Transport Layer Services

- ❖ Internet transport layer protocols:
  - TCP: connection-oriented and reliable
  - UDP: connection-less and unreliable
- ❖ 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 (routers) 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.



# Lecture 4: 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:
  - Connectionless multiplexing / de-multiplexing
  - Checksum
- ❖ UDP transmission is **unreliable**
  - Often used by streaming multimedia apps (loss tolerant & rate sensitive)
- ❖ To achieve reliable transmission over UDP
  - Application implements error detection and recovery mechanisms!

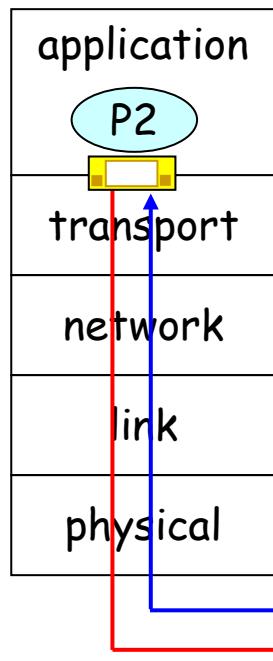
# Connectionless De-multiplexing

- ❖ UDP sender:
  - Creates a socket with local port #.
  - When creating a datagram to send to UDP socket, sender must specify dest. IP address and port #.
- ❖ 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.

# Connectionless De-multiplexing

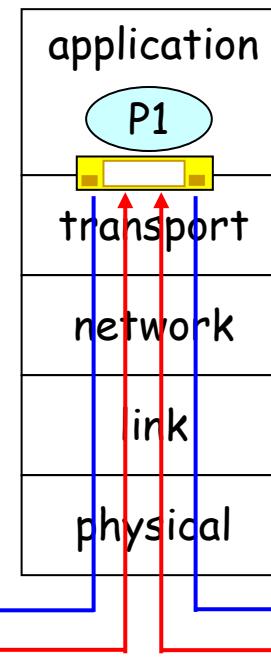
**DatagramSocket**

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



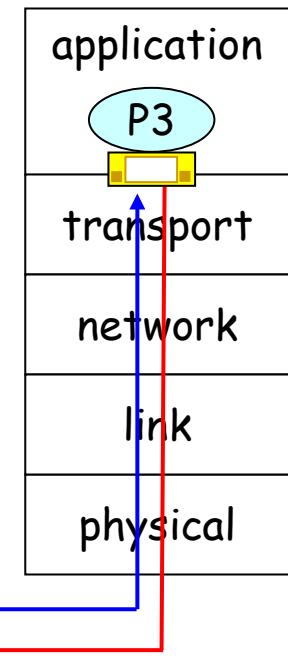
**DatagramSocket**

```
mySocket1 = new  
DatagramSocket(6428);
```



**DatagramSocket**

```
mySocket3 = new  
DatagramSocket(5775);
```



host 1

source port: 9157  
dest port: 6428

host 2

source port: 5775  
dest port: 6428

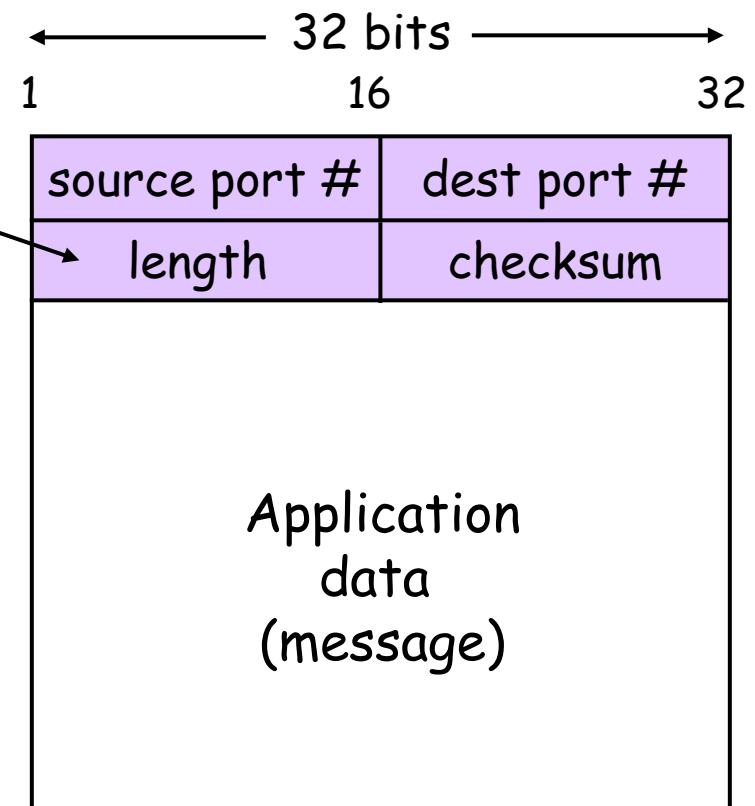
host 3

source port: 6428  
dest port: 9157

source port: 6428  
dest port: 5775

# 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

**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?)

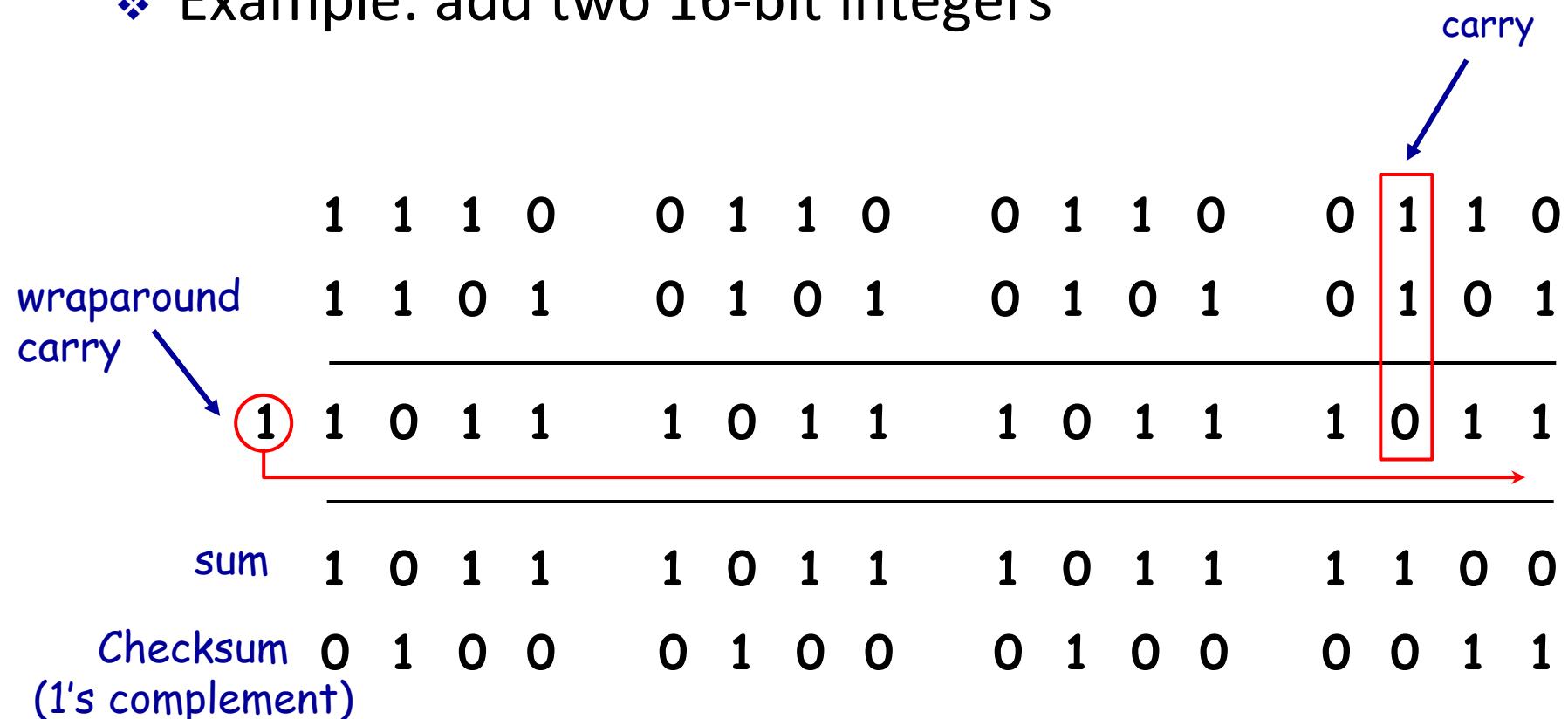
# Checksum Computation

- ❖ 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



# Lecture 4: Roadmap

3.1 Transport-layer Services

3.3 Connectionless Transport: UDP

3.4 Principles of Reliable Data Transfer

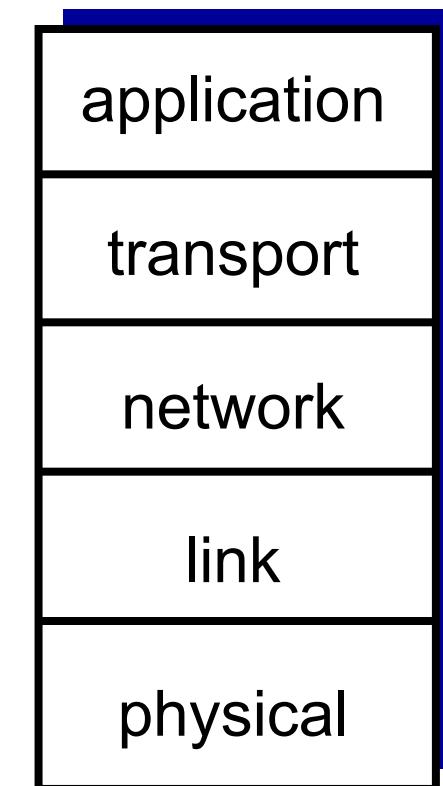
3.5 Connection-oriented transport: TCP



*“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.

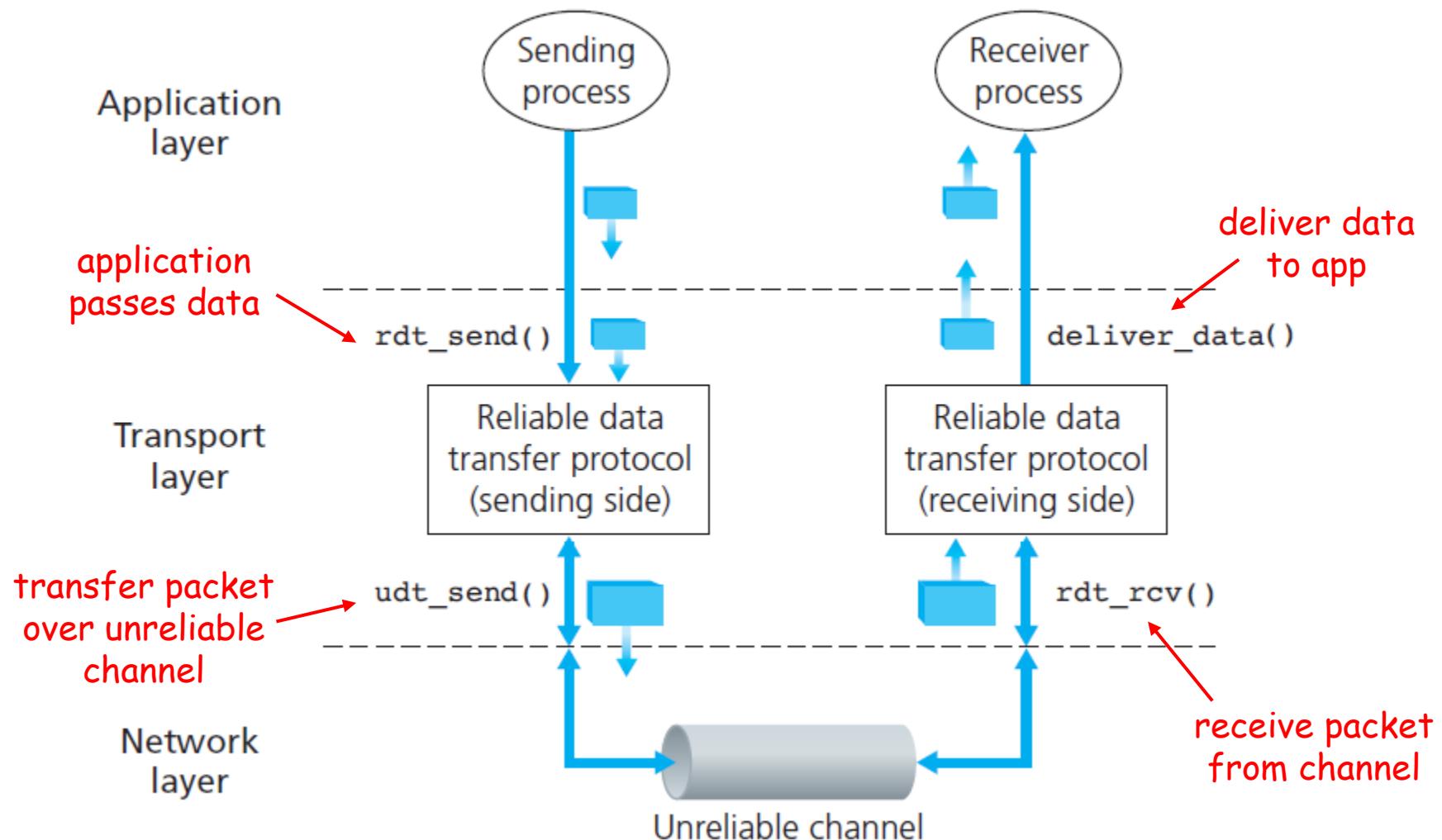


**Question:** How to build a **reliable transport layer protocol** on top of **unreliable communication**?

# 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 application) in the same order they are sent

# Reliable Data Transfer: Service Model

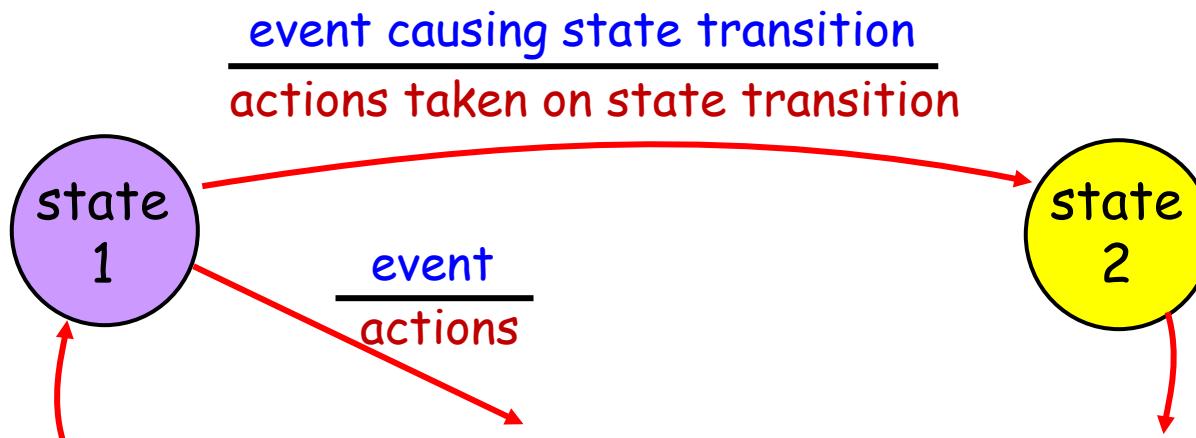


# Reliable Data Transfer Protocols

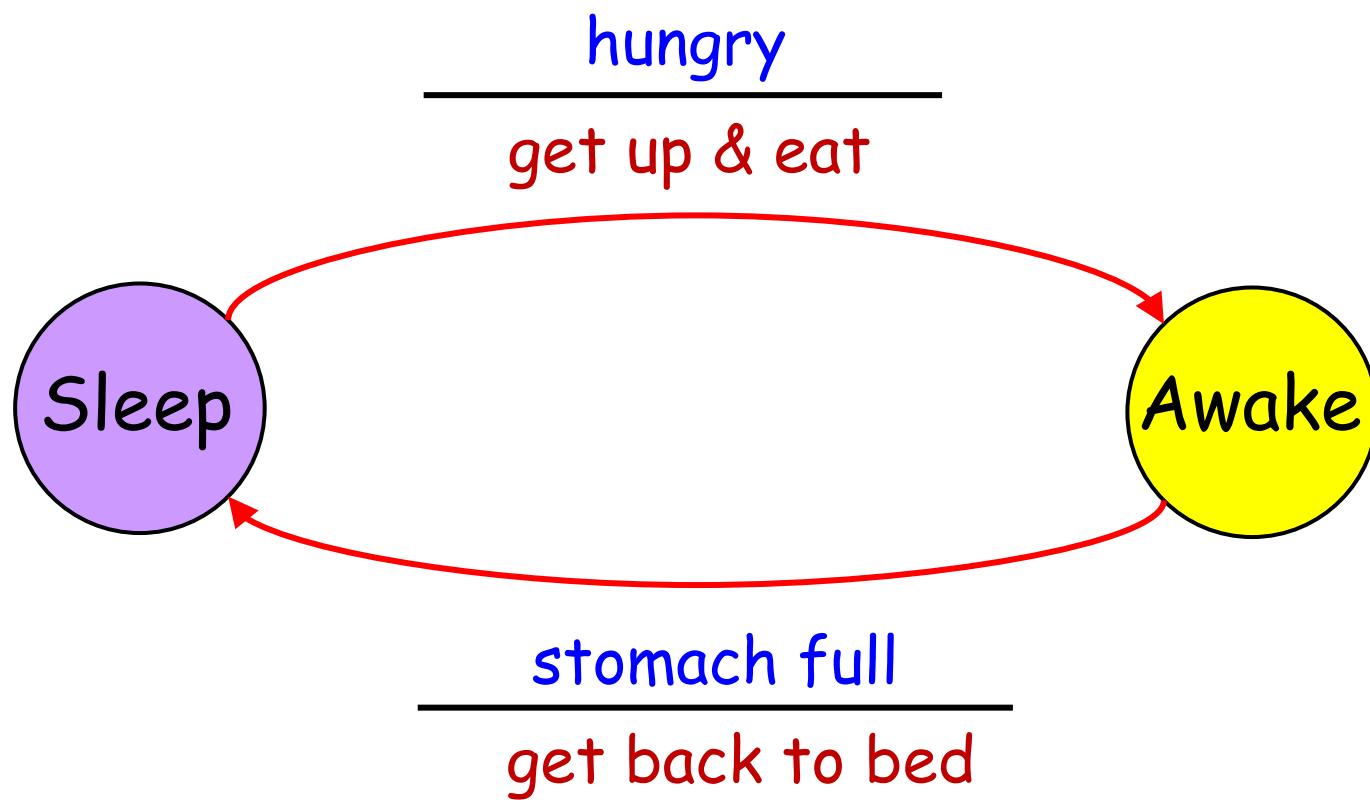
- ❖ Characteristics of unreliable channel will determine the complexity of reliable data transfer protocols (**rdt**).
- ❖ We will incrementally develop sender & receiver sides of **rdt** protocols, considering increasingly complex models of unreliable channel.
- ❖ We consider only unidirectional data 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.

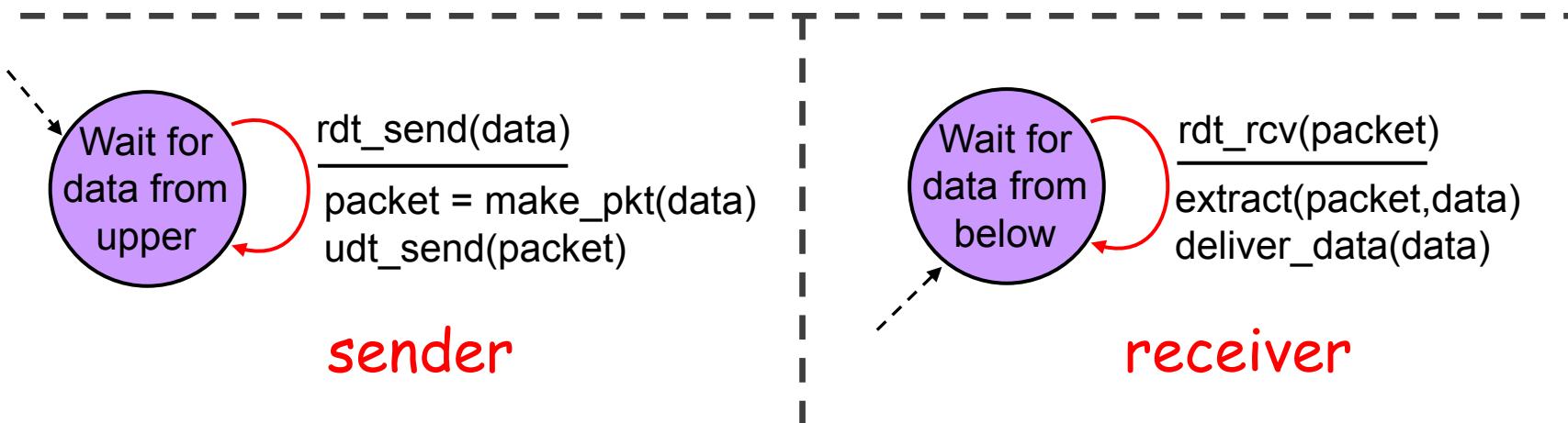


# Example FSM



# rdt 1.0: Perfectly Reliable Channel

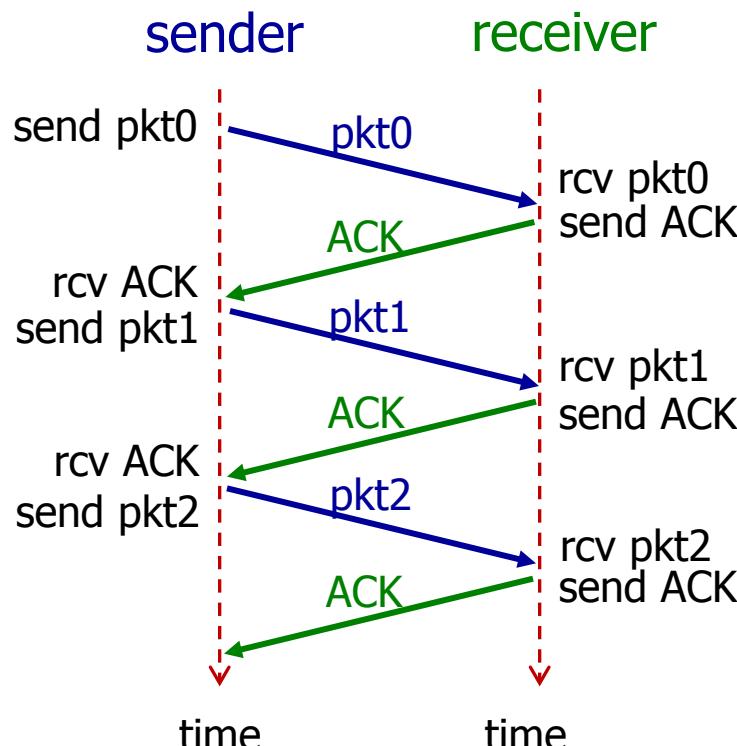
- ❖ Assume underlying channel is perfectly reliable.
- ❖ Separate FSMs for sender, receiver:
  - Sender sends data into underlying (perfect) channel
  - Receiver reads data from underlying (perfect) channel



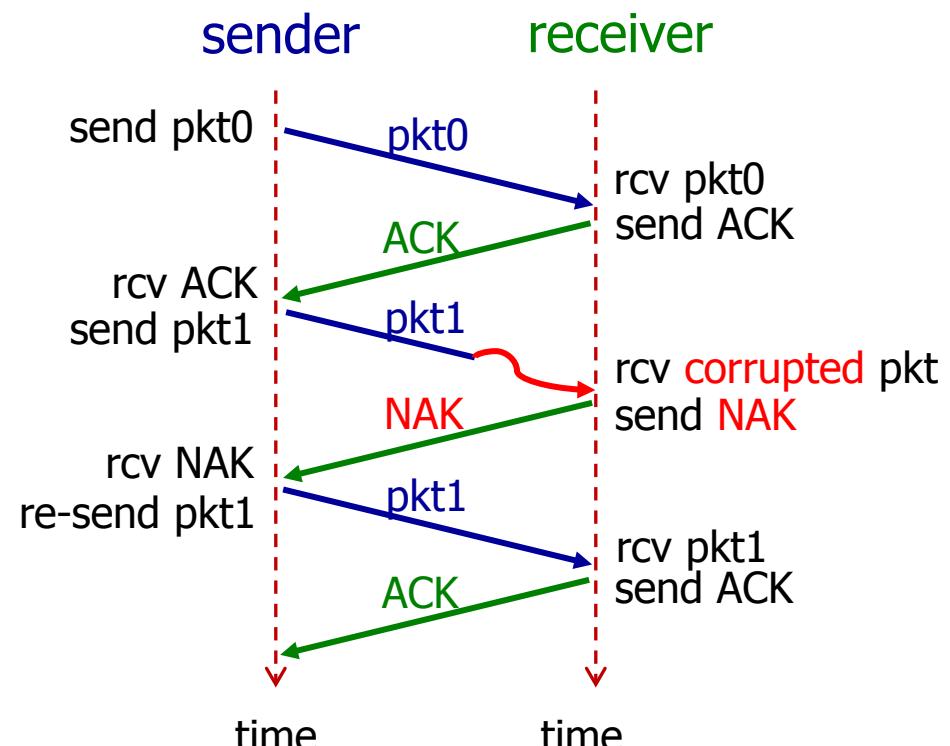
# rdt 2.0: Channel with *Bit Errors*

- ❖ Assumption:
  - underlying channel **may flip bits in packets**
  - other than that, the channel is perfect
- ❖ Receiver may use **checksum** to detect bit errors.
- ❖ **Question:** 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

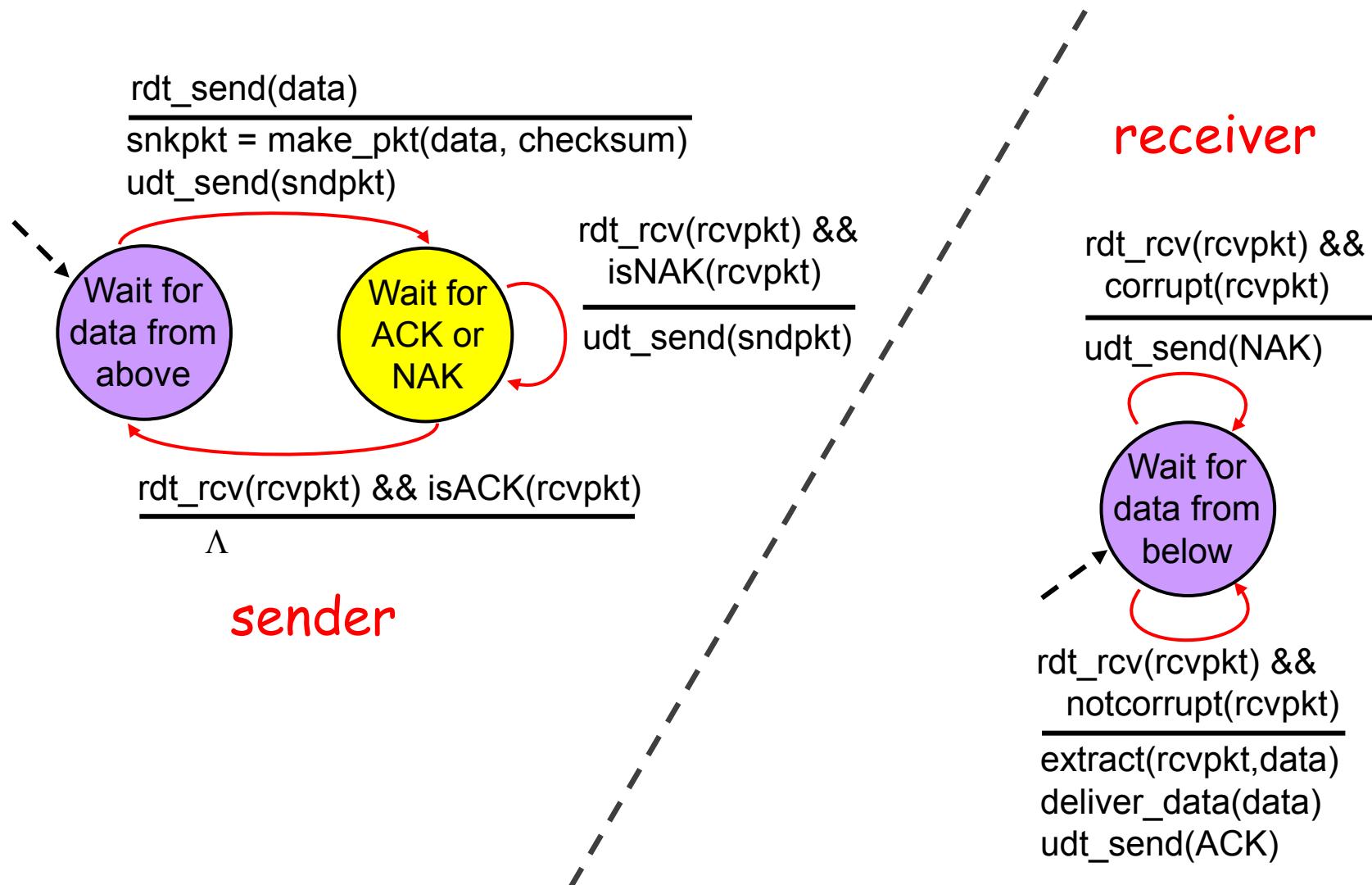


(b) with bit error

**stop and wait protocol**

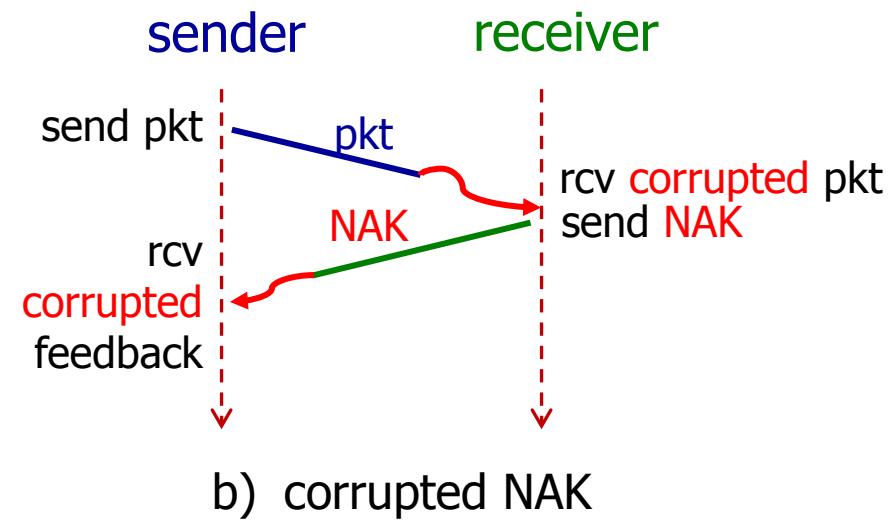
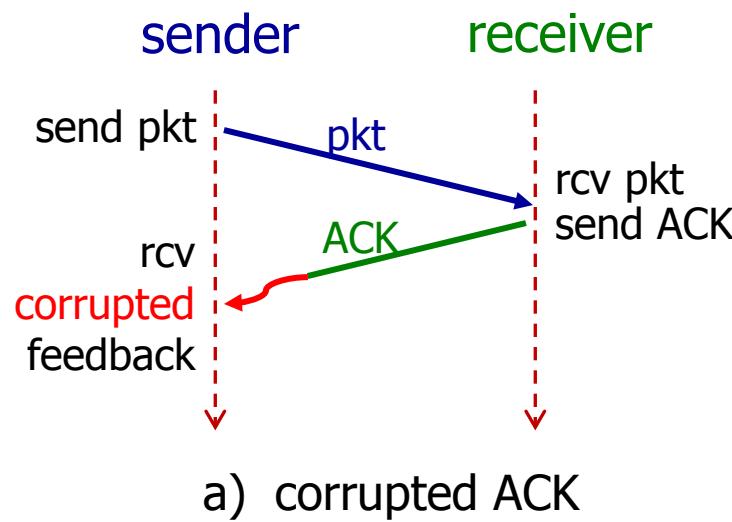
Sender sends one packet at a time, then waits for receiver response

# rdt 2.0: FSM



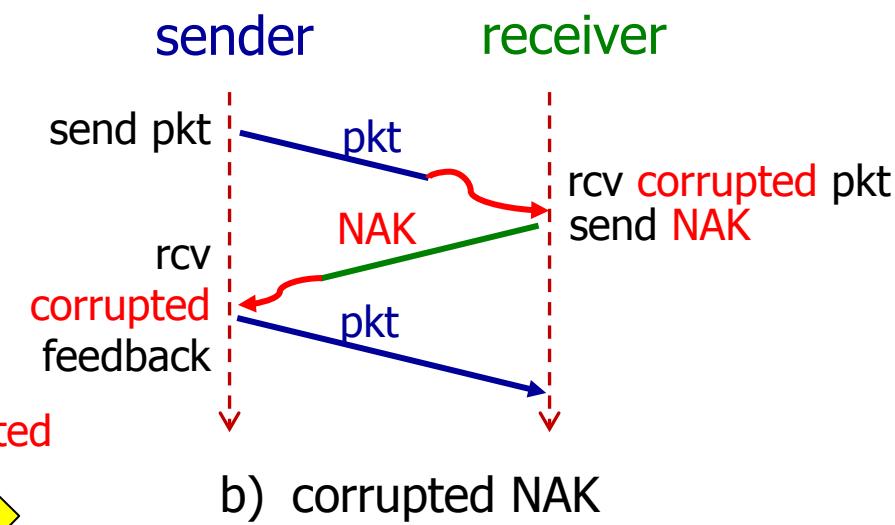
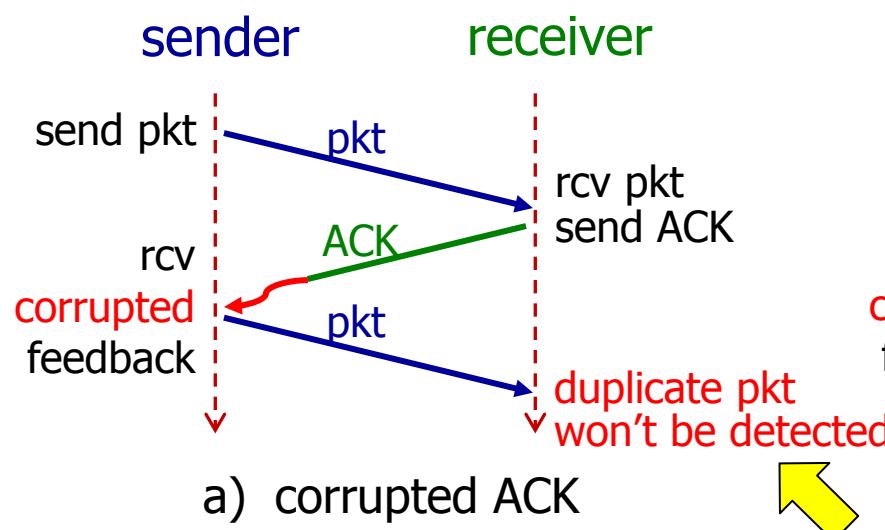
# rdt 2.0 has a Fatal Flaw!

- ❖ 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?

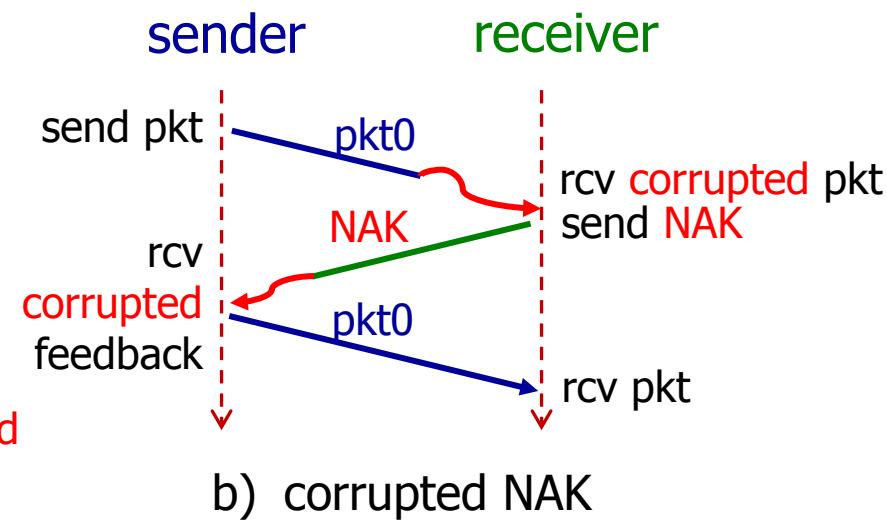
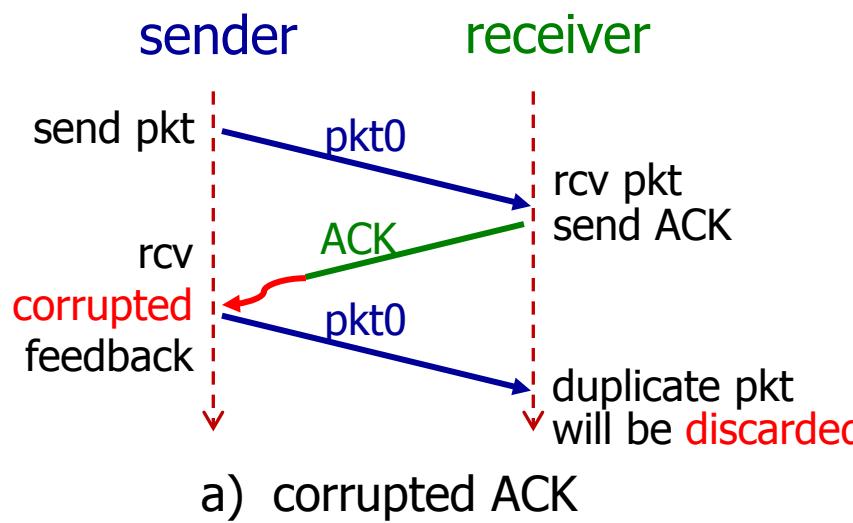


# rdt 2.1: rdt 2.0 + Packet Seq. #

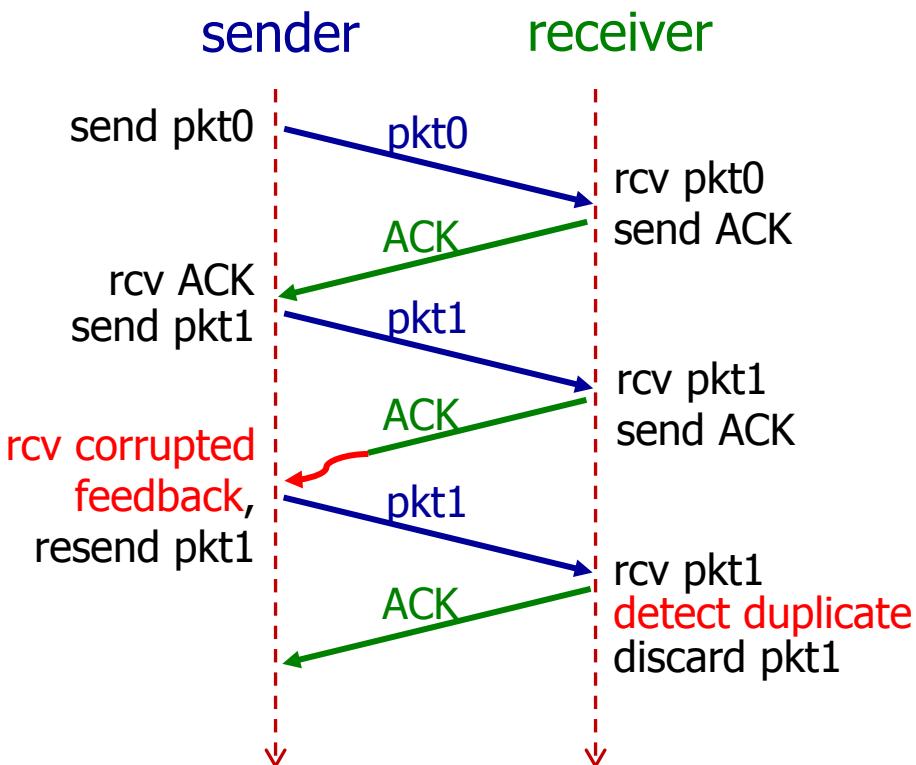
## ❖ To handle duplicates:

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

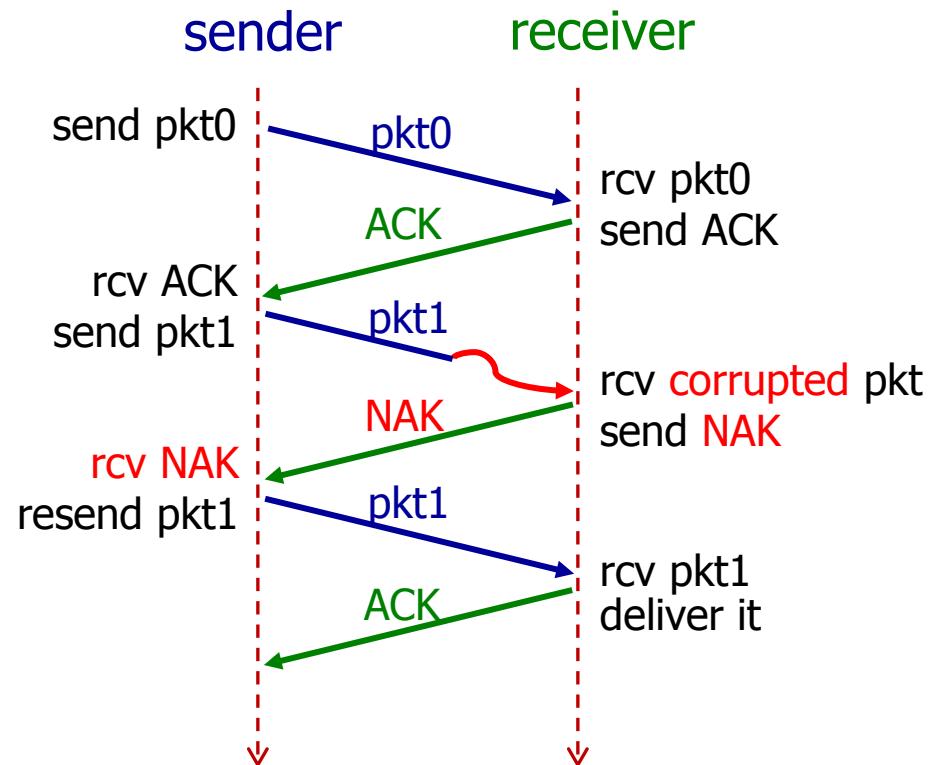
## ❖ This gives rise to protocol rdt 2.1.



# rdt 2.1 In Action

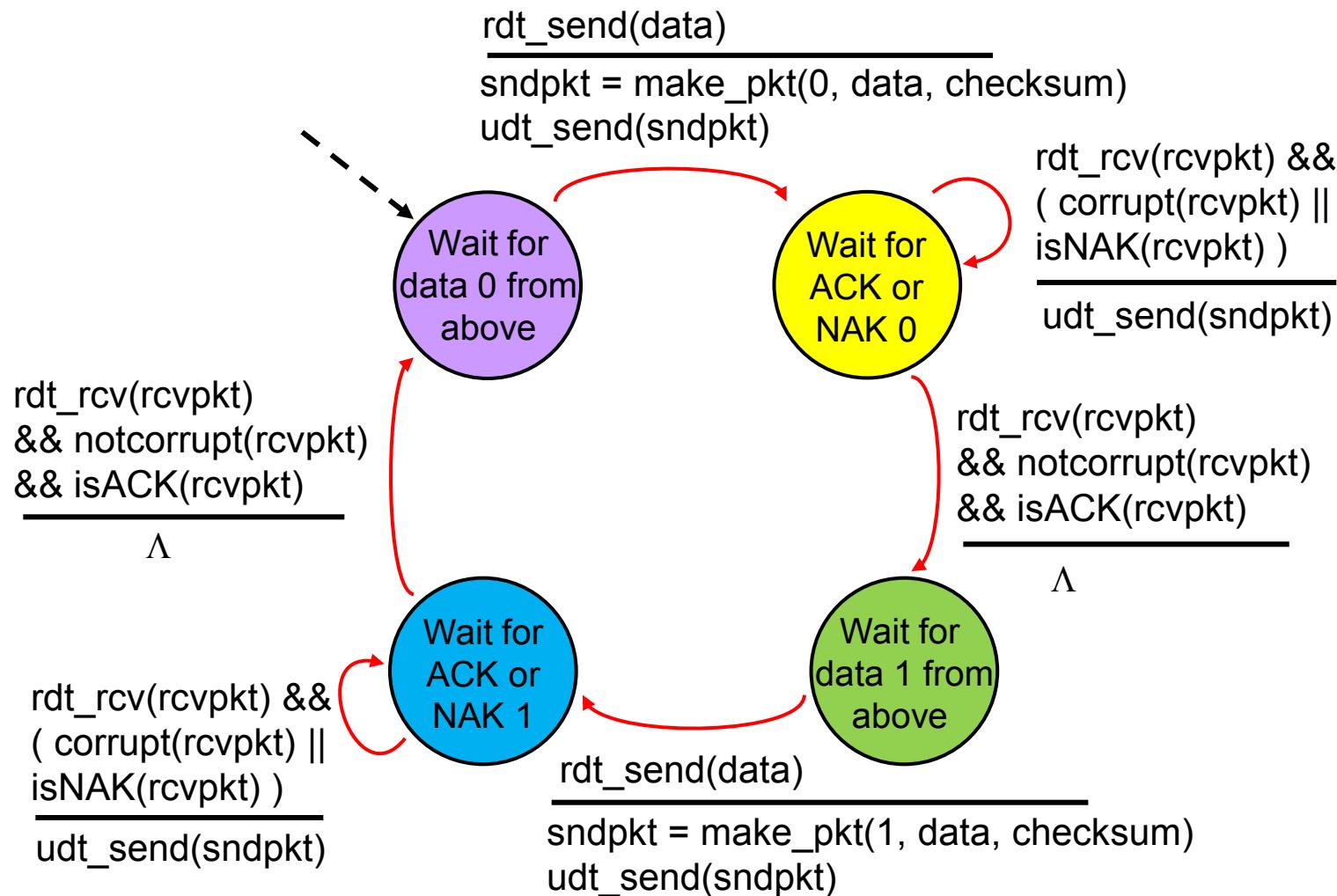


a) resend due to  
corrupted ACK

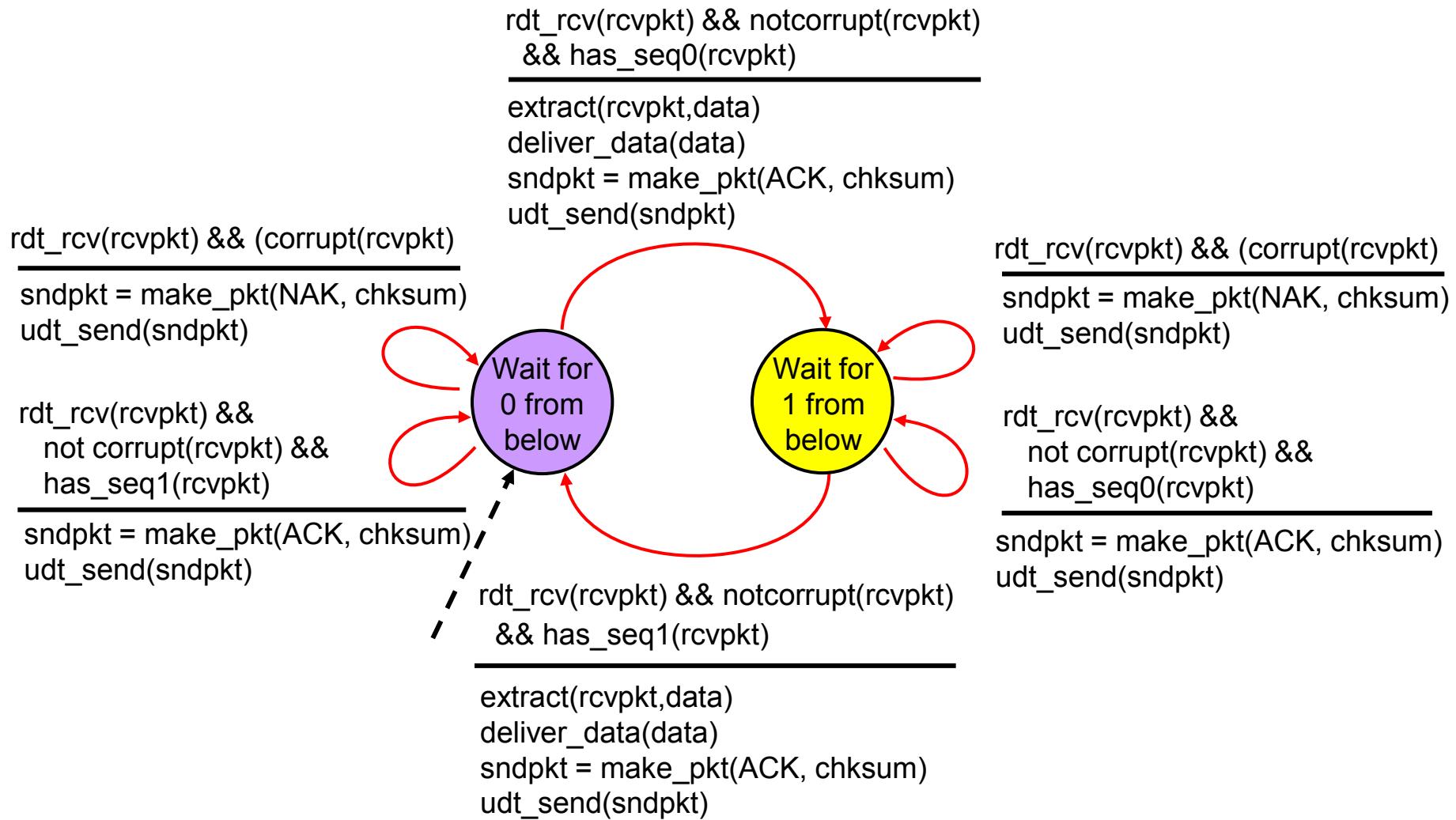


b) resend due to  
corrupted packet

# rdt 2.1 Sender FSM



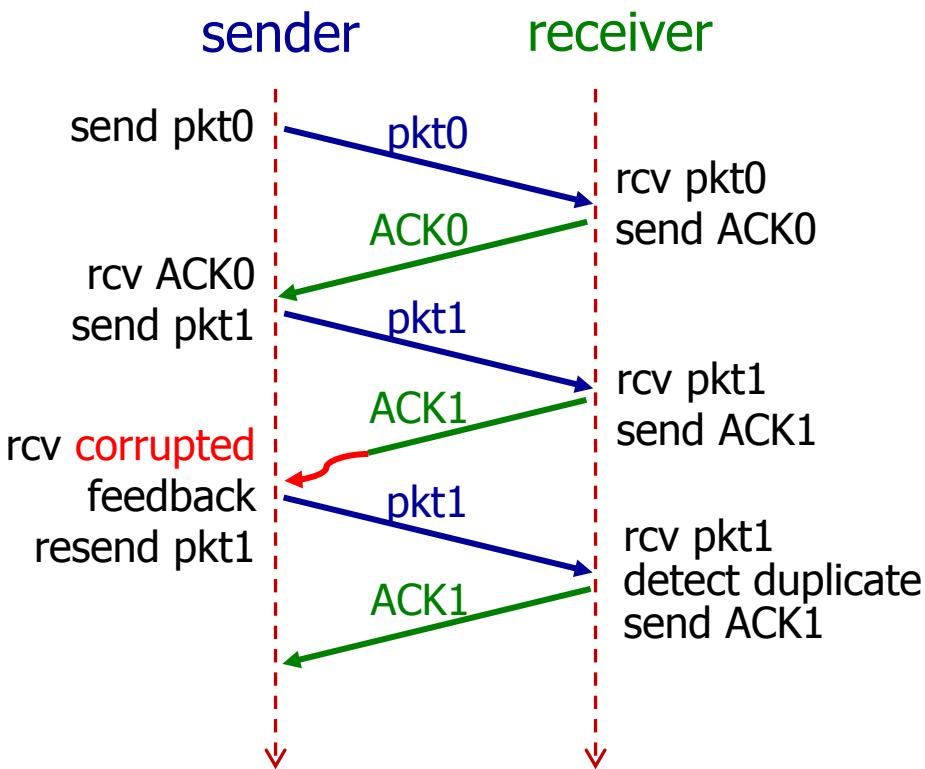
# rdt 2.1 Receiver FSM



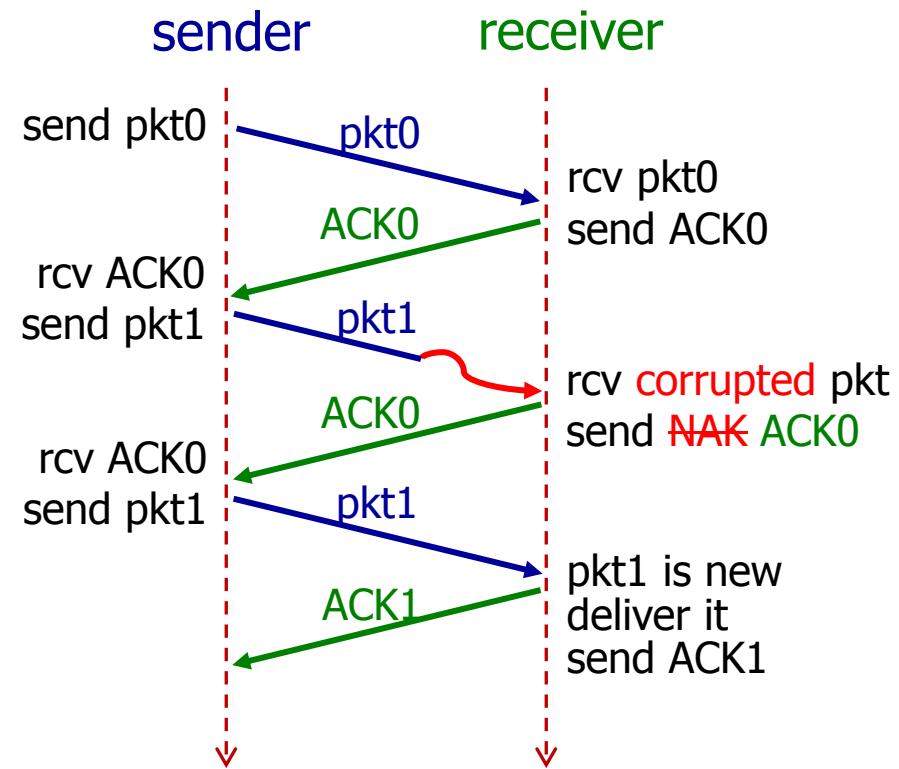
# rdt 2.2: a NAK-free Protocol

- ❖ 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*

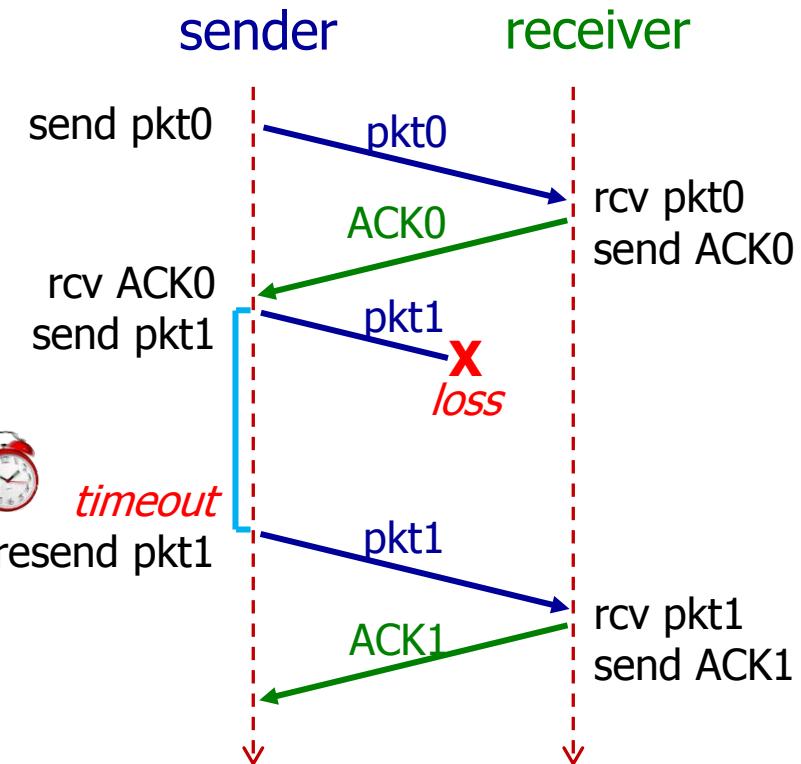
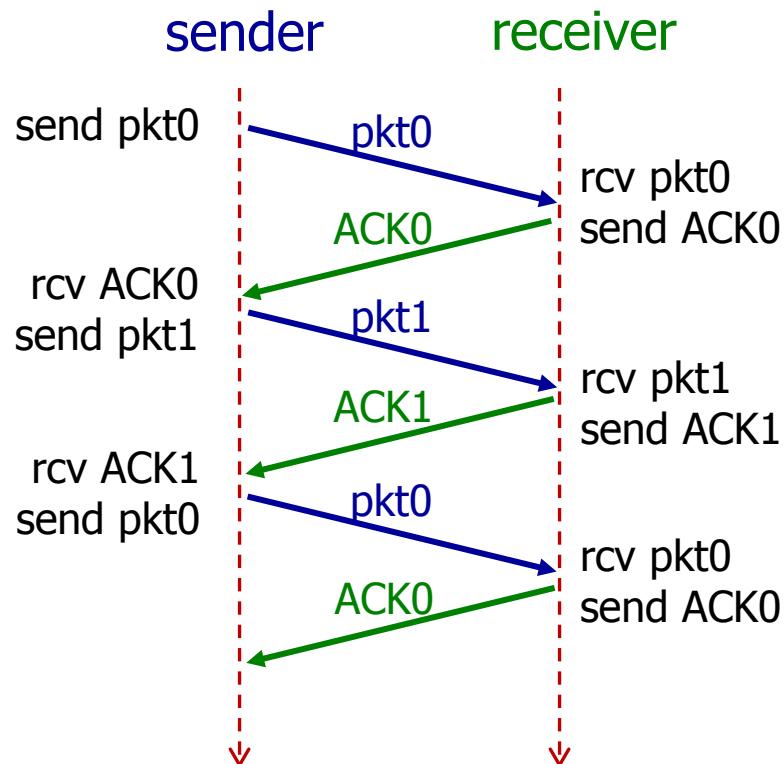
- ❖ To handle packet loss:

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

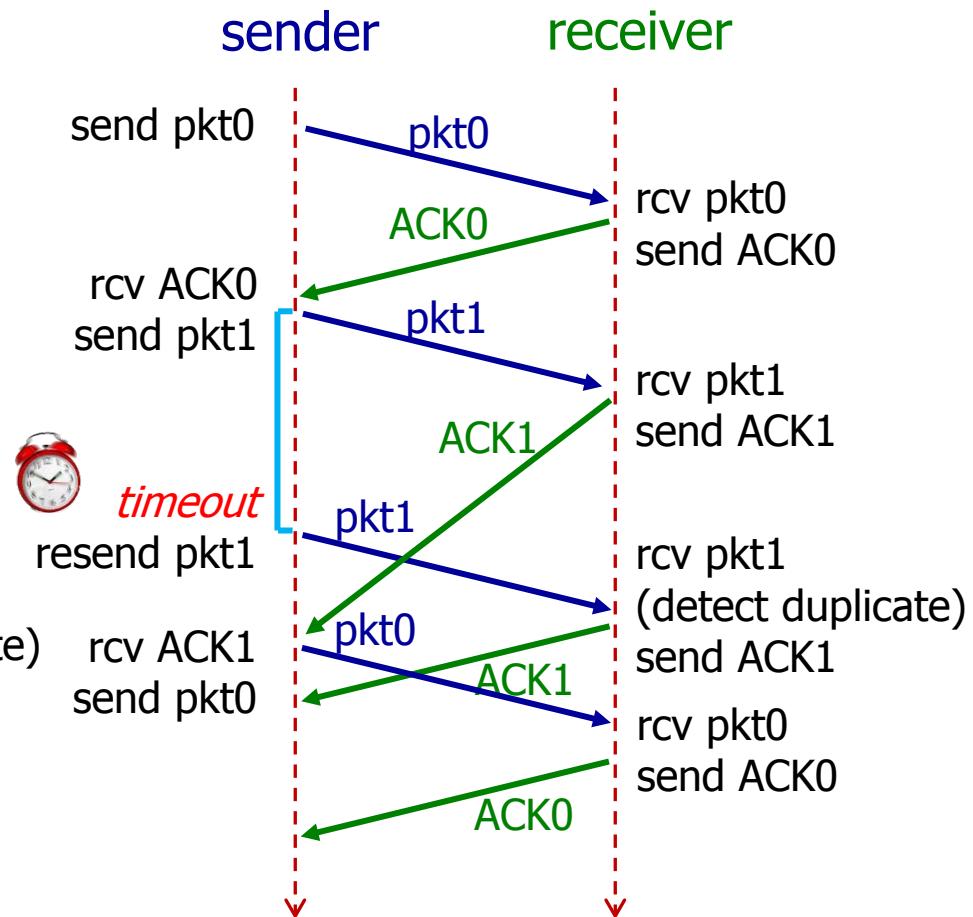
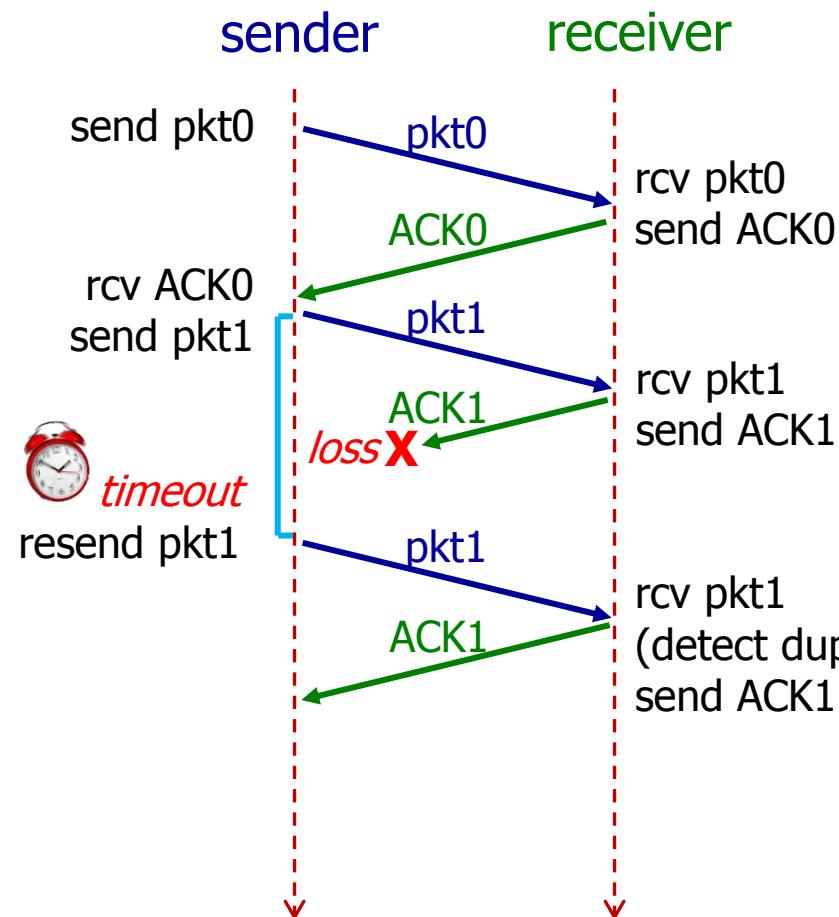
- ❖ Question: what if packet (or ACK) is just delayed, but not lost?

- 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).

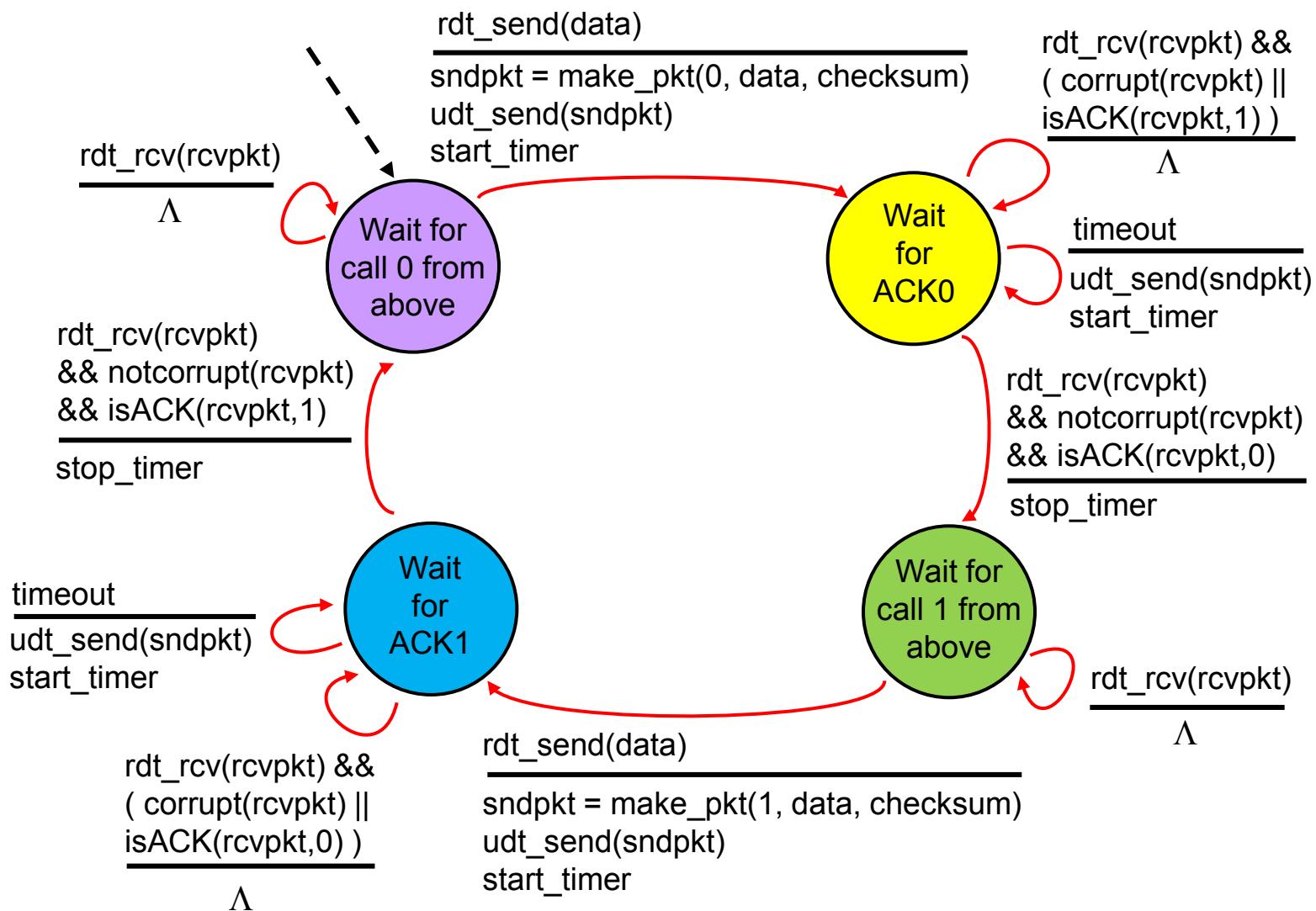
# rdt 3.0 In Action



# rdt 3.0 In Action



# rdt 3.0 Sender FSM



# 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.

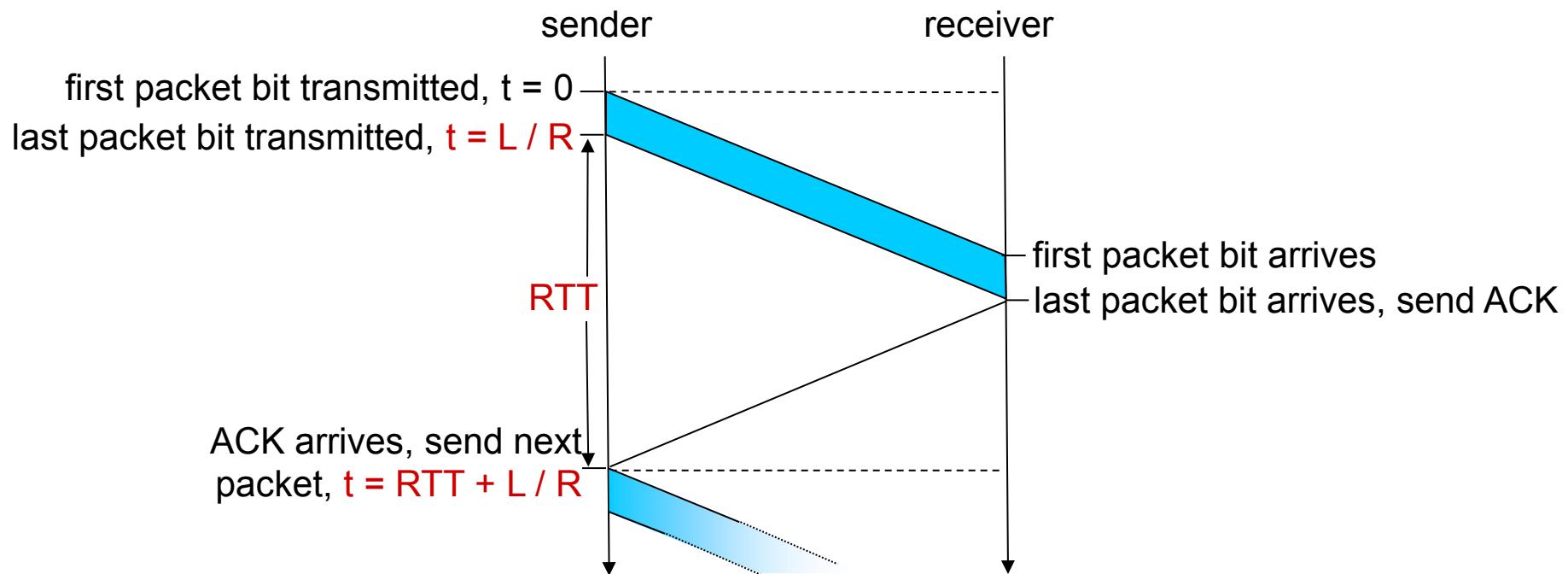
$$\text{throughput} = \frac{L}{RTT + d_{trans}} = \frac{8000}{30.008} = 267 \text{ kbps}$$

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

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

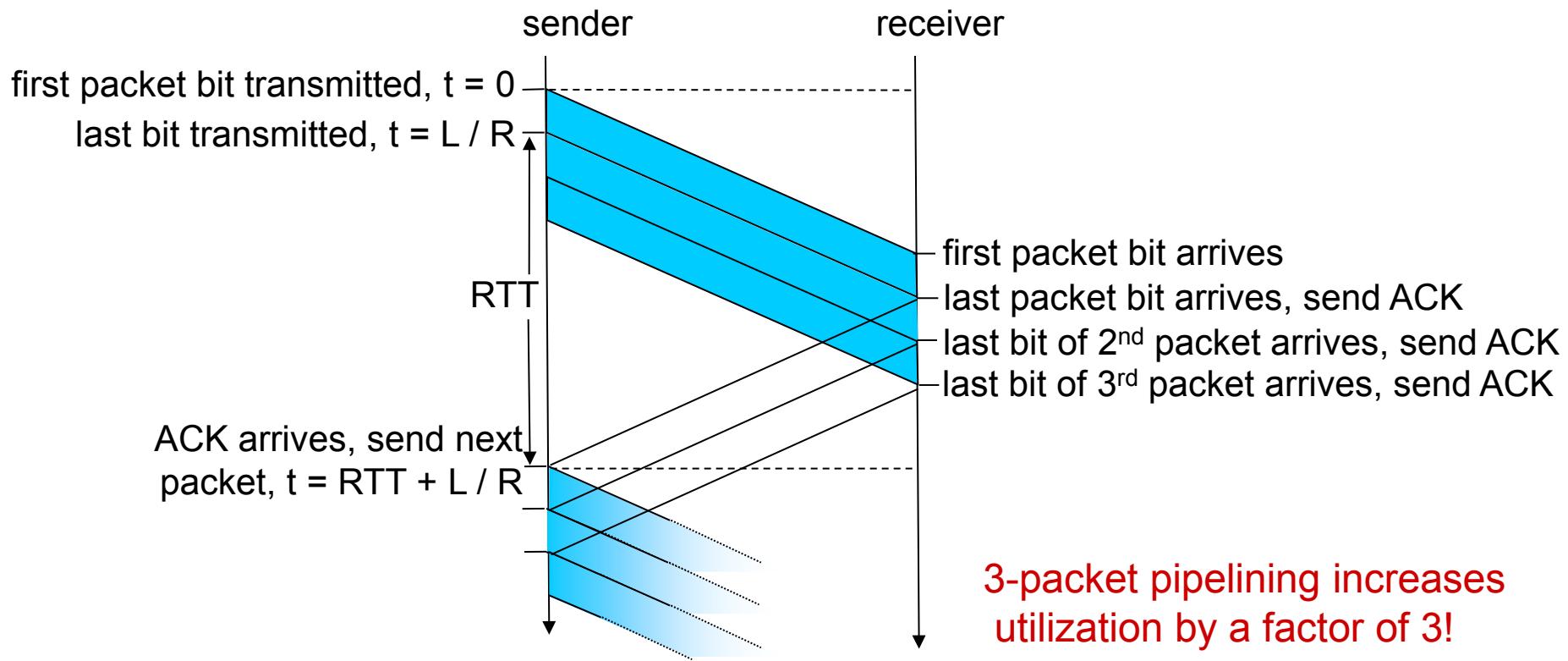
# rdt 3.0: Stop-and-wait Operation

- ❖ Network protocol limits use of physical resources!



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

# Pipelining: Increased Utilization

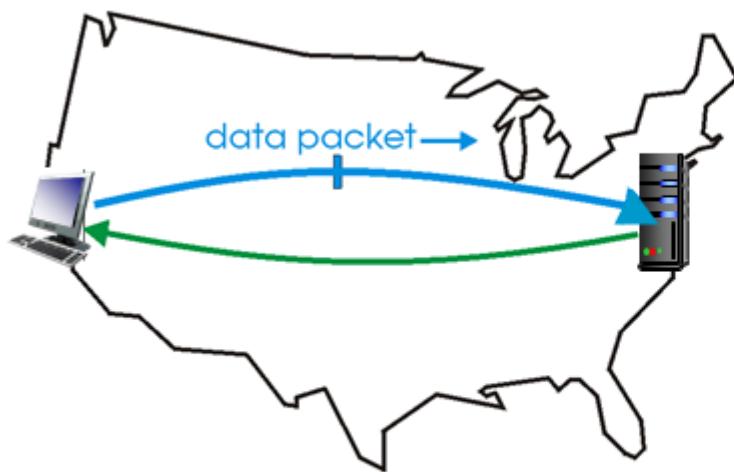


$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L/R} = \frac{0.024}{30 + 0.008} = 0.00081$$

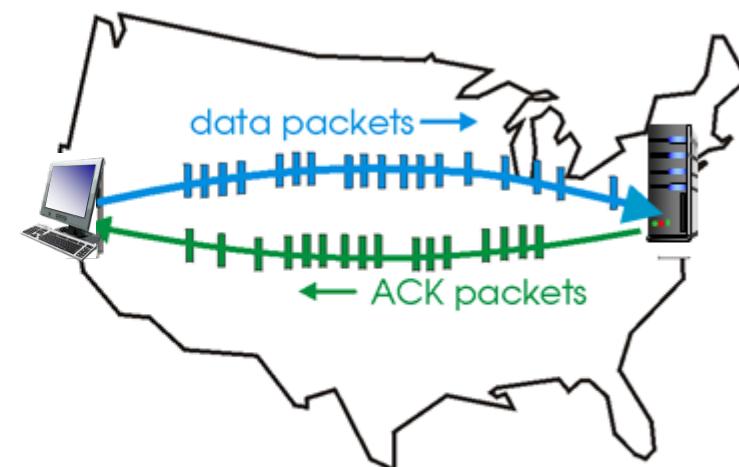
# Pipelined Protocols

**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*
  - *Selective repeat*
- ❖ 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 |

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 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 pkt3, **discard**,  
(re)send ACK1

rcv ACK0, send pkt4  
rcv ACK1, send pkt5

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

ignore duplicate ACK



*pkt 2 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 pkt2  
(re)send pkt3  
(re)send pkt4  
(re)send pkt5

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 |
|---|---|---|---|---|---|---|---|

rcv ACK2, send pkt4  
send pkt5  
send pkt6  
(wait)

loss X  
loss X  
loss X

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 |

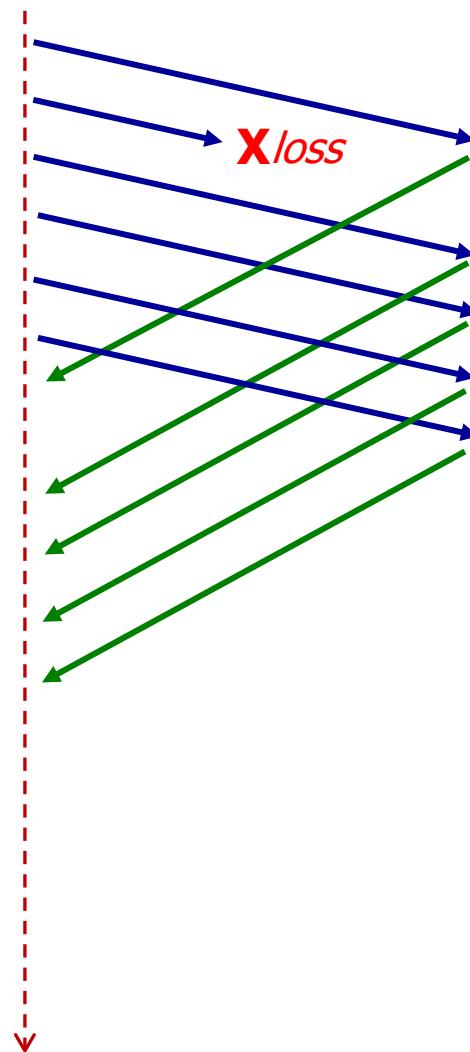
|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|

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.

# 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 |

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 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 ACK0, send pkt4  
rcv ACK1, send pkt5

pkt 2 timeout  
(re)send pkt2

*receiver*

receive pkt0, send ACK0  
receive pkt1, send ACK1

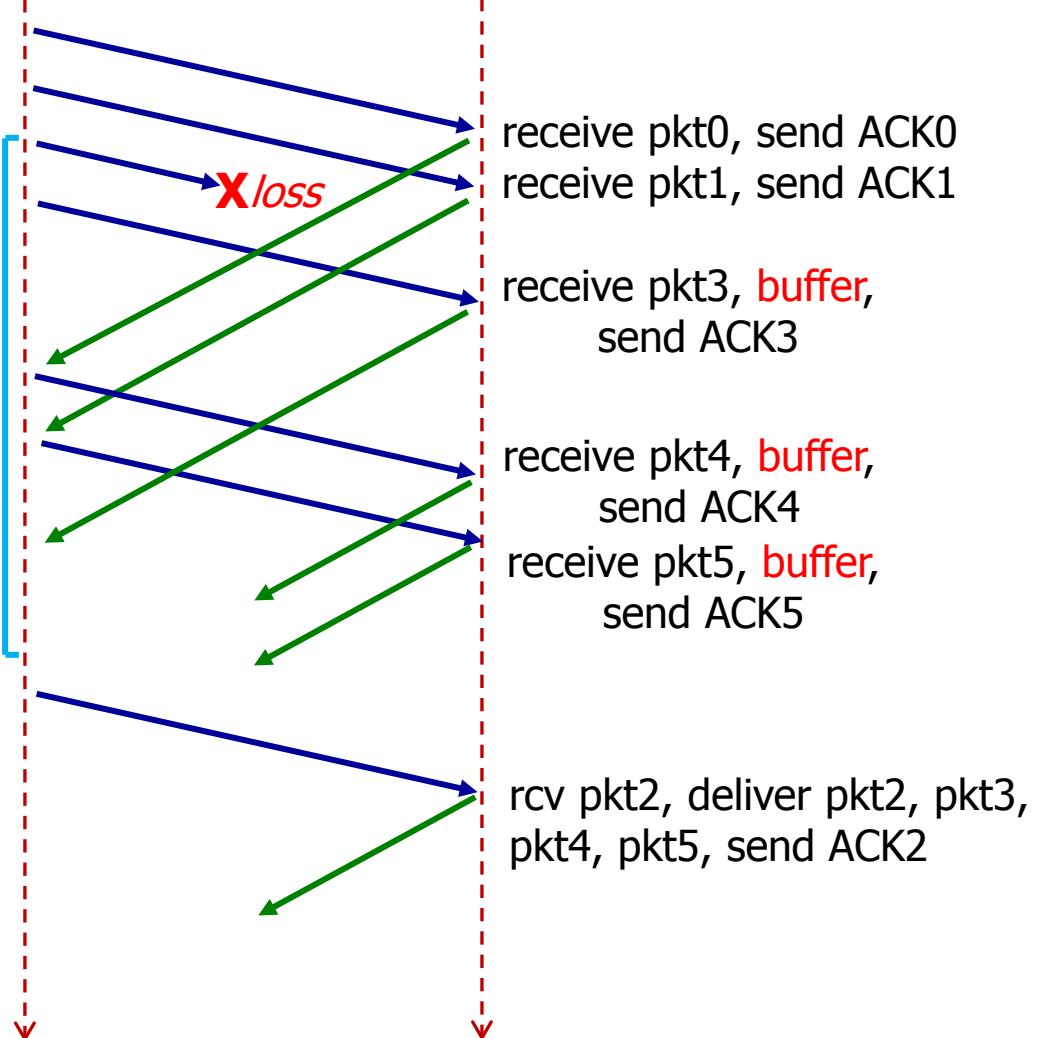
receive pkt3, buffer,  
send ACK3

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

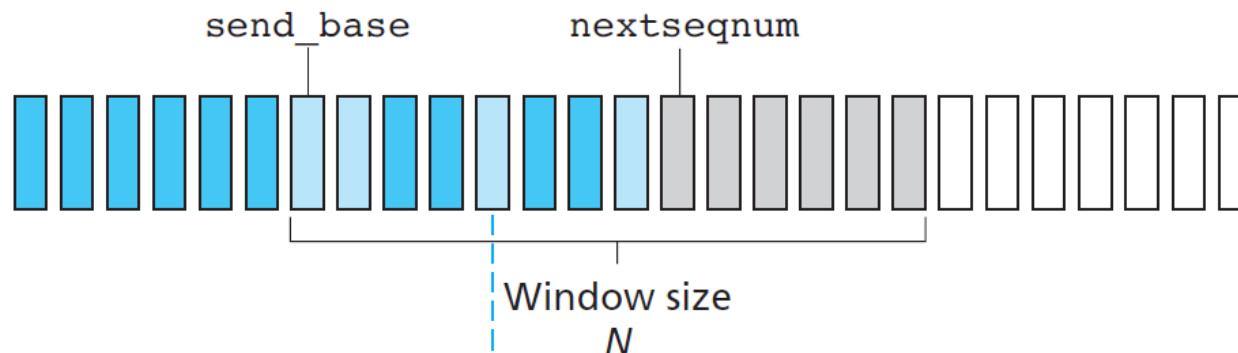
rcv pkt2, deliver pkt2, pkt3,  
pkt4, pkt5, send ACK2



*pkt 2 timeout*



# SR Sender and Receiver Windows

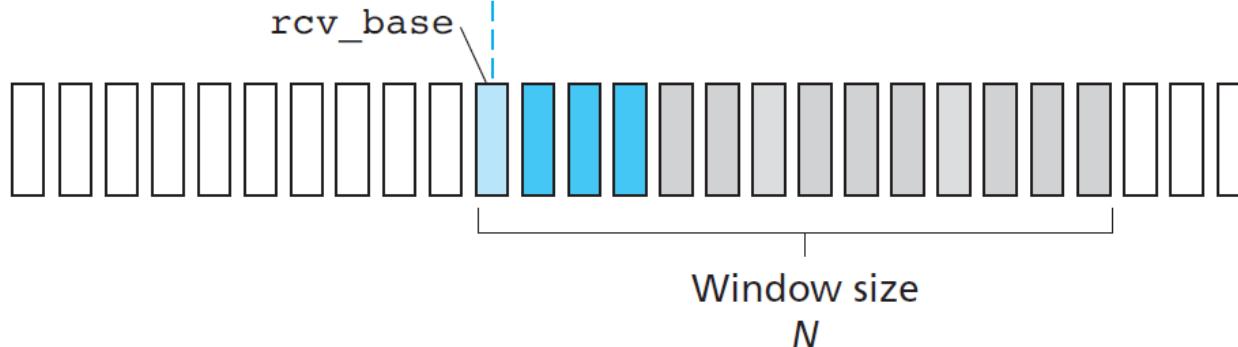


a. Sender view of sequence numbers

Key:

Already  
ACK'dSent, not  
yet ACK'dUsable,  
not yet sent

Not usable



b. Receiver view of sequence numbers

Key:

Out of order  
(buffered) but  
already ACK'dExpected, not  
yet receivedAcceptable  
(within  
window)

Not usable

# 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

# Lecture 4: Summary

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

# Lecture 4: 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

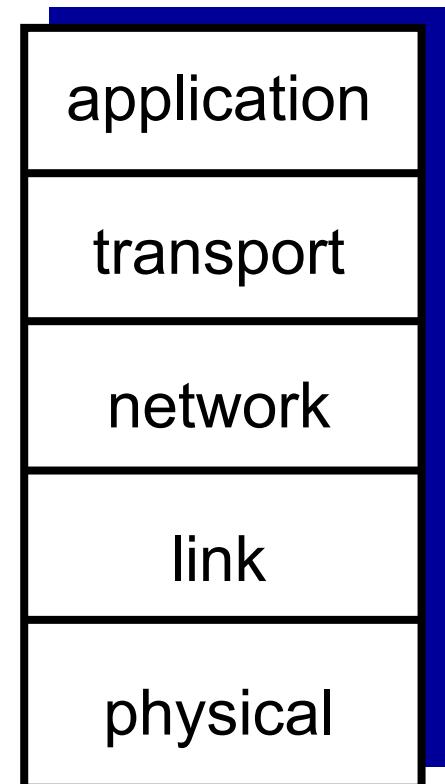
## Selective Repeat

- ❖ 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

# An Awesome Introduction to Computer Networks

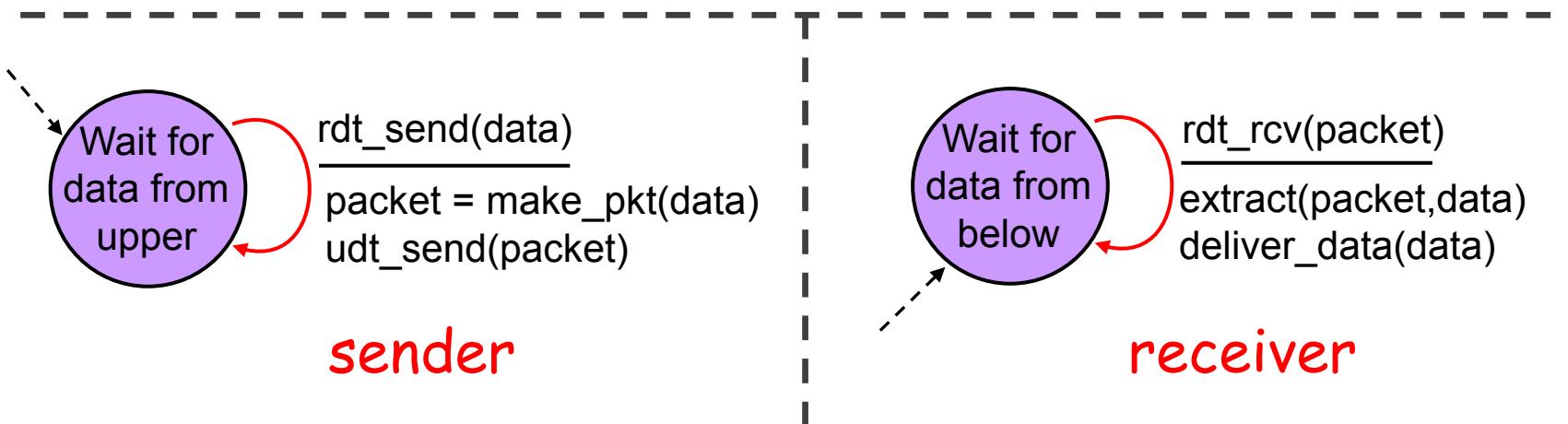
# Designing Reliable Protocols

- ❖ Network layer service is unreliable.
  - Packets may be **lost or corrupted** during transmission.
- ❖ A reliable transport protocol should
  - ensure that packets are received by receiver in good order.
- ❖ Scenario: **one sender, one receiver**
  - Sender sends data packets to receiver
  - Receiver feeds back to sender



# rdt 1.0: Reliable Channel

- ❖ Assume underlying channel is perfectly reliable.
- ❖ No error checking is needed.
  - Sender sends data into underlying (perfect) channel
  - Receiver reads data from underlying (perfect) channel

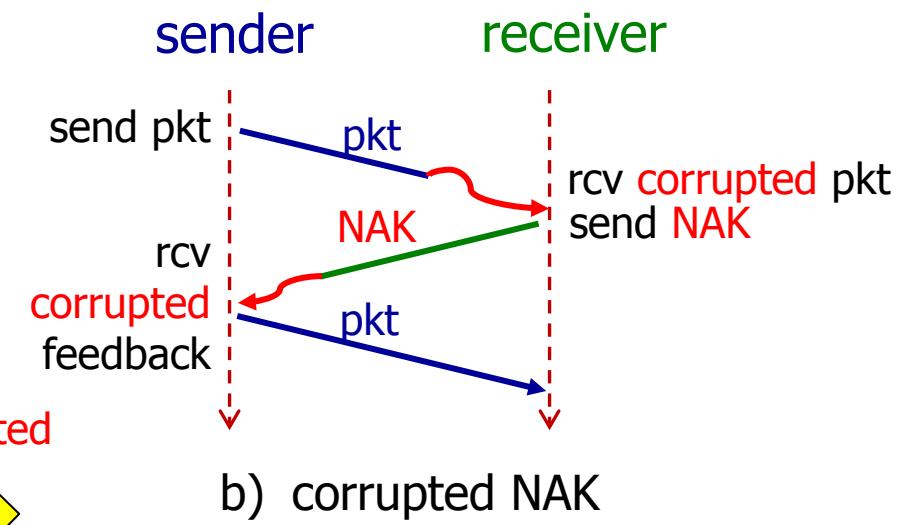
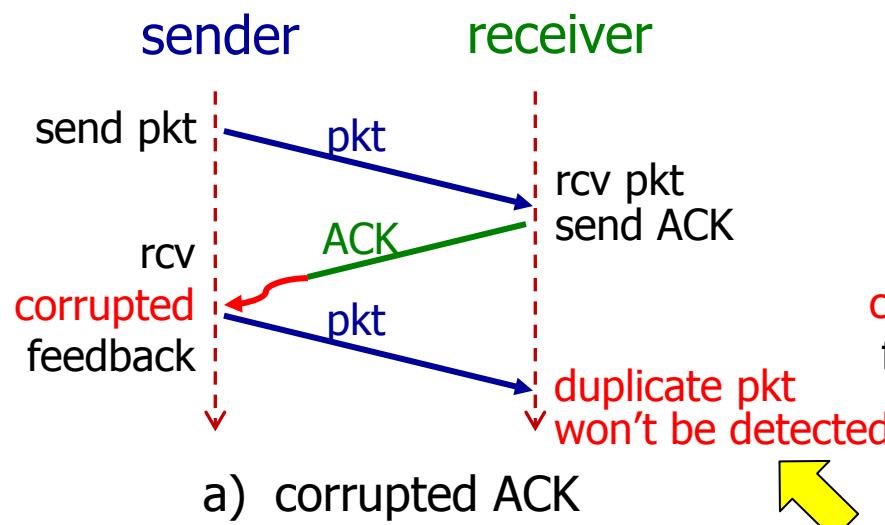


# rdt 2.0: Channel with *Bit Errors*

- ❖ Assumption:
  - underlying channel **may flip bits in packets**
  - other than that, the channel is perfect
- ❖ Receiver may use **checksum** to detect bit errors.
- ❖ Receiver will feed back to sender:
  - *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 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 should retransmit when receives garbled ACK or NAK.
    - **Issue:** receiver may receive duplicate packet.

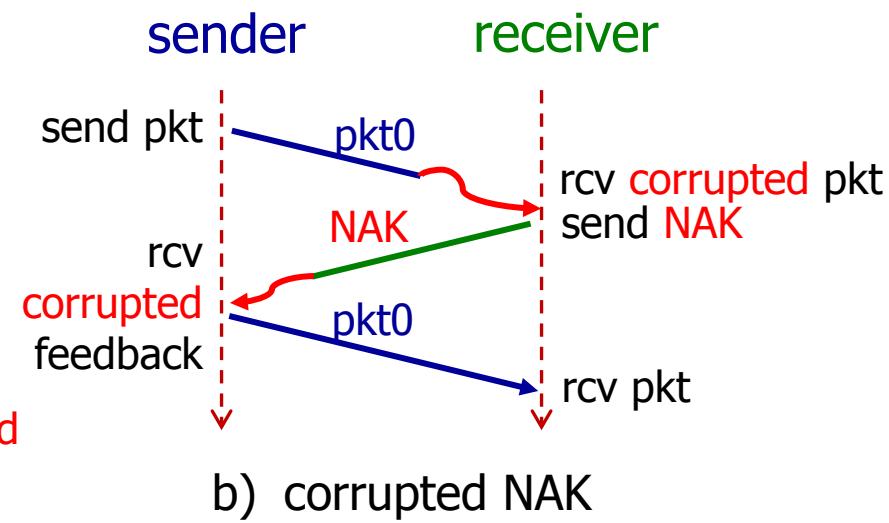
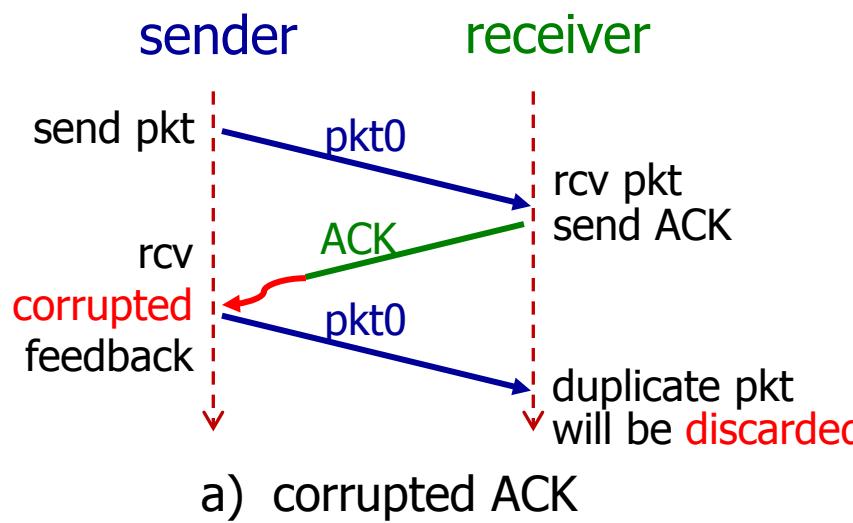


# rdt 2.1: rdt 2.0 + Packet Seq. #

- ❖ 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 2.2: 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 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
- ❖ To handle packet loss:
  - Sender waits “reasonable” amount of time for ACK.
  - Sender retransmits if no ACK is received till *timeout*.
- ❖ Sender relies on timeout/retransmission to deal with both packet corruption and packet loss.

# Lecture 5: TCP

*After this class, you are expected to:*

- ❖ understand the operations of TCP, in particular, the sequence number, the acknowledgement number, retransmission, and connection setup/termination.

# Lecture 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**

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

# TCP: Transport Control Protocol

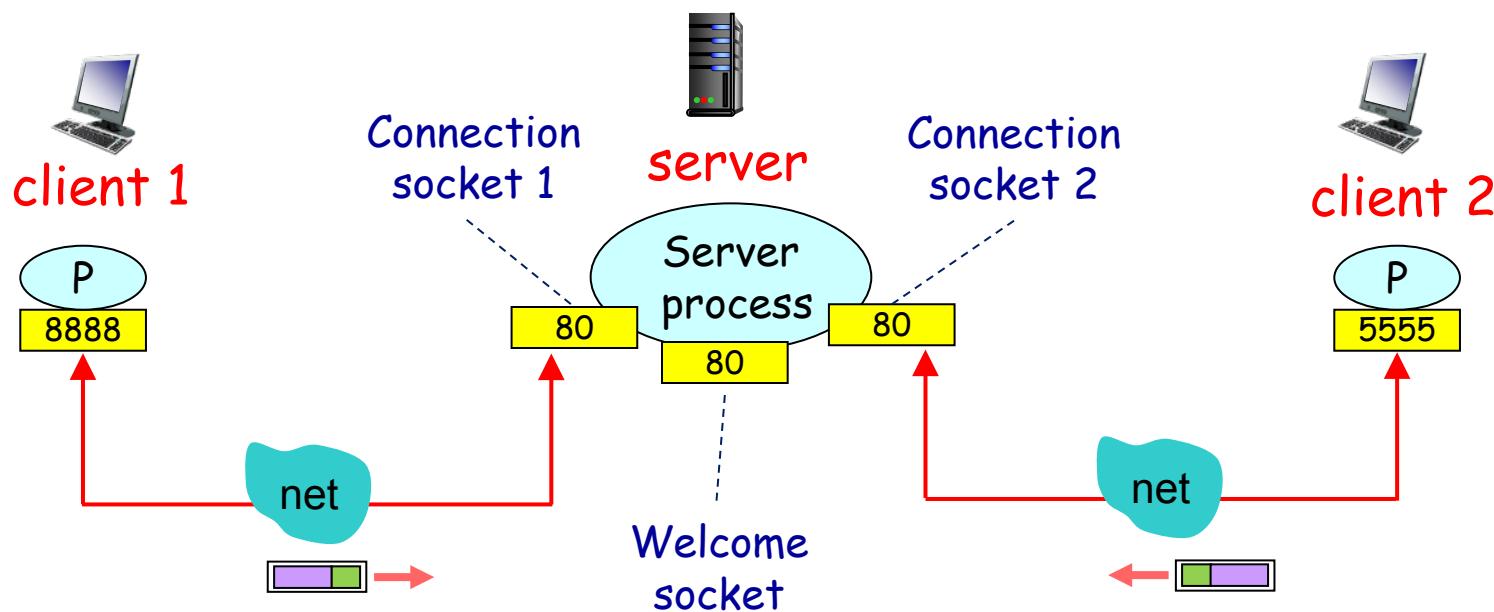
- ❖ In contrast to UDP, TCP is complex and is described in tens of RFCs, with new mechanisms or tweaks introduced throughout the years, resulting in many variants of TCP.
  
- ❖ We will only scratch the surface of TCP in CS2105.
  - More will be covered in CS3103.

# TCP Overview

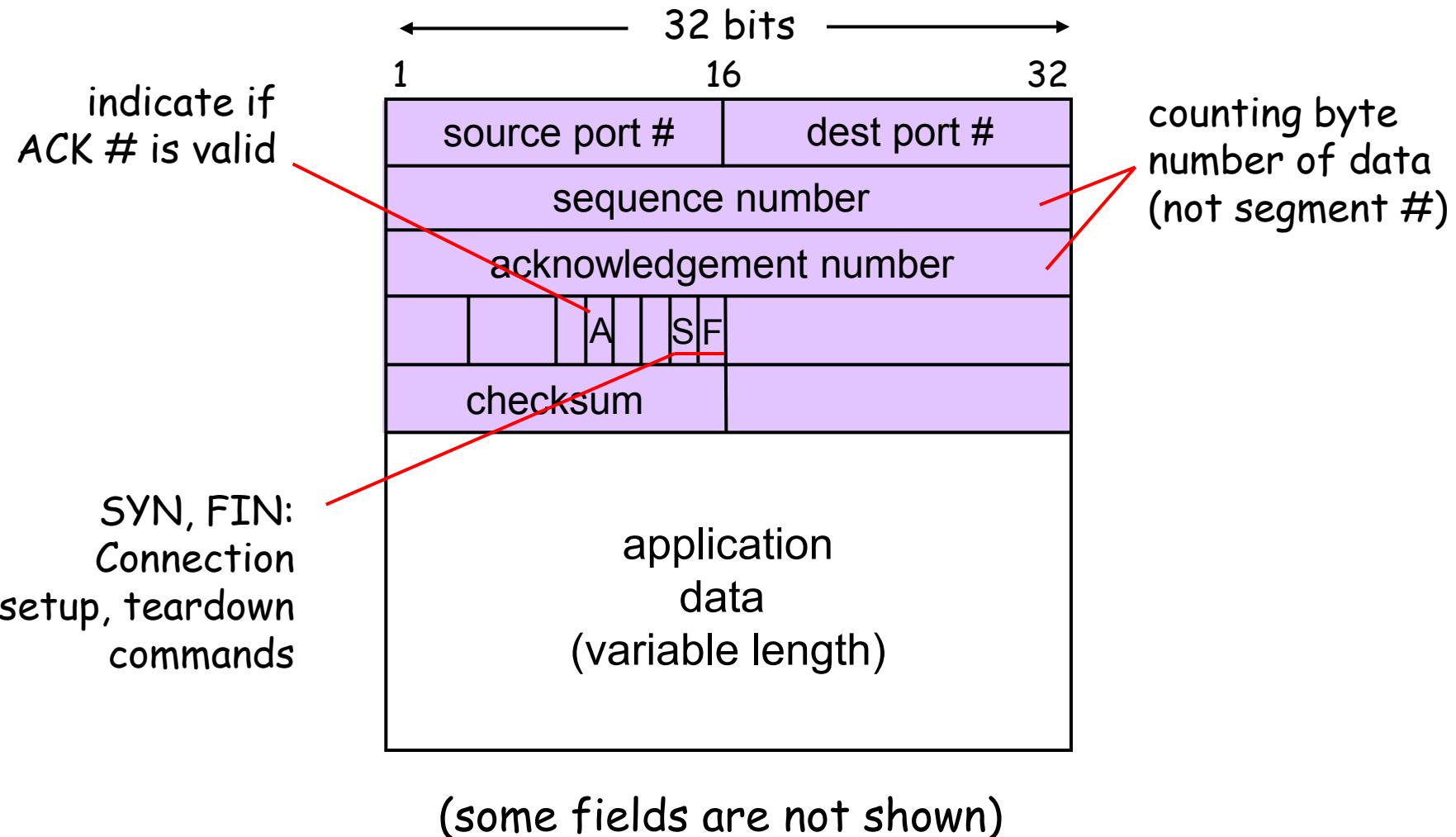
- ❖ **Connection-oriented:**
  - handshaking (exchange of control messages) before sending app data.
- ❖ **Reliable, in-order *byte steam*:**
  - Application passes data to TCP and TCP forms packets in view of **MSS (maximum segment size)**.  
(For UDP, app forms packets: **DatagramPacket**)
- ❖ **Flow control and congestion control**
  - Not discussed!

# 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.



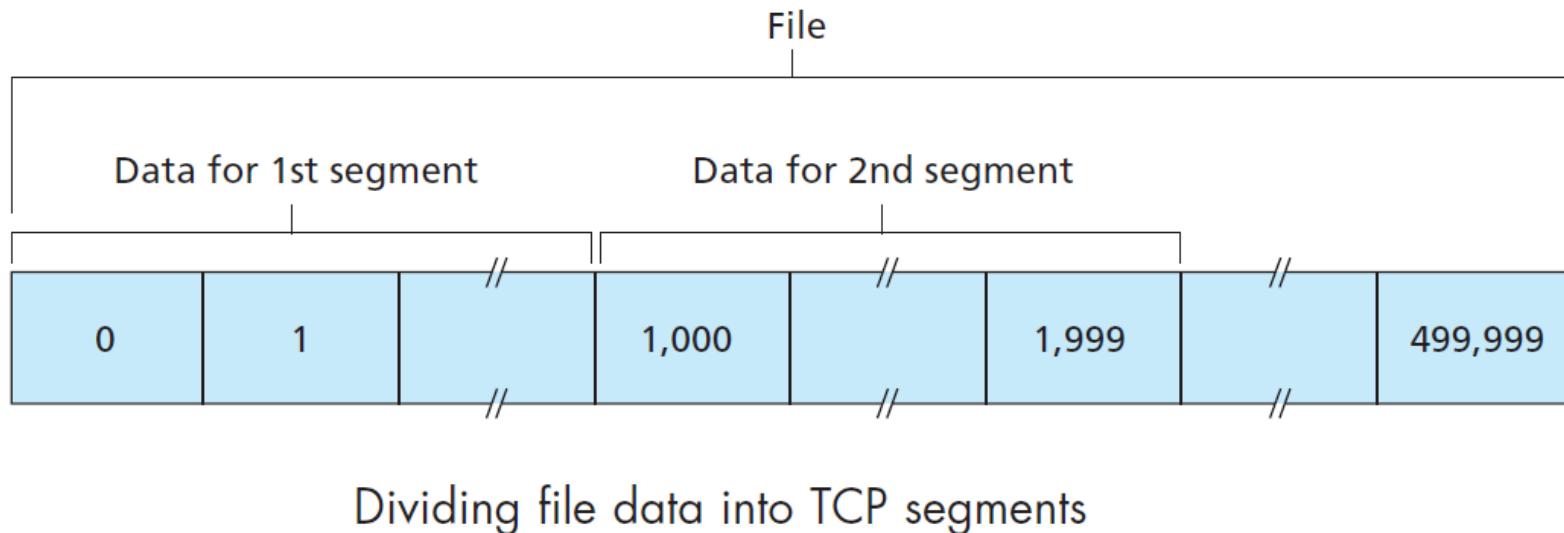
# TCP Header



|                 |             |
|-----------------|-------------|
| source port #   | dest port # |
| sequence number |             |
| ACK number      |             |
|                 |             |
| checksum        |             |

# TCP Sequence Number

- ❖ TCP sequence number: “byte number” of the first byte of data in a segment.
- ❖ Example: send a file of 500,000 bytes; MSS is 1000 bytes.



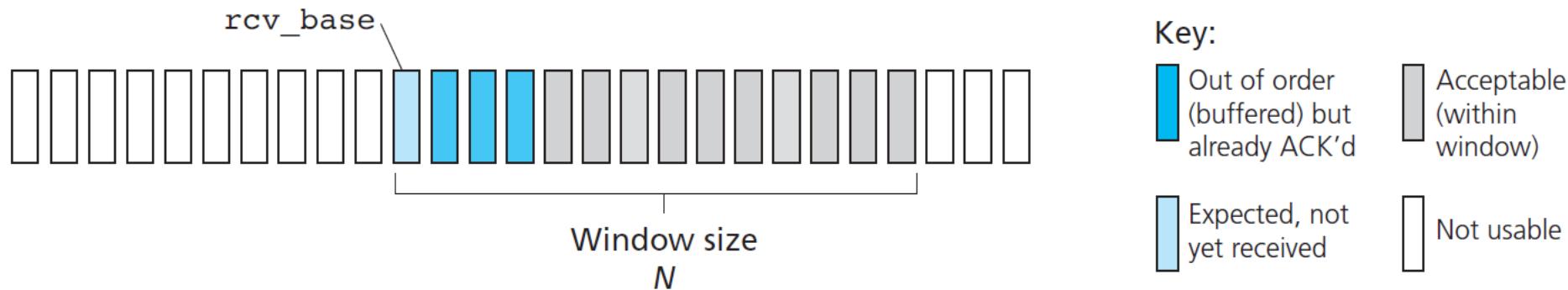
- ❖ Seq. # of 1<sup>st</sup> TCP segment: 0, 2<sup>nd</sup> TCP segment: 1000, 3<sup>rd</sup> TCP segment: 2000, 4<sup>th</sup> TCP segment: 3000, etc.

# TCP ACK Number

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

- ❖ Acknowledgement number:

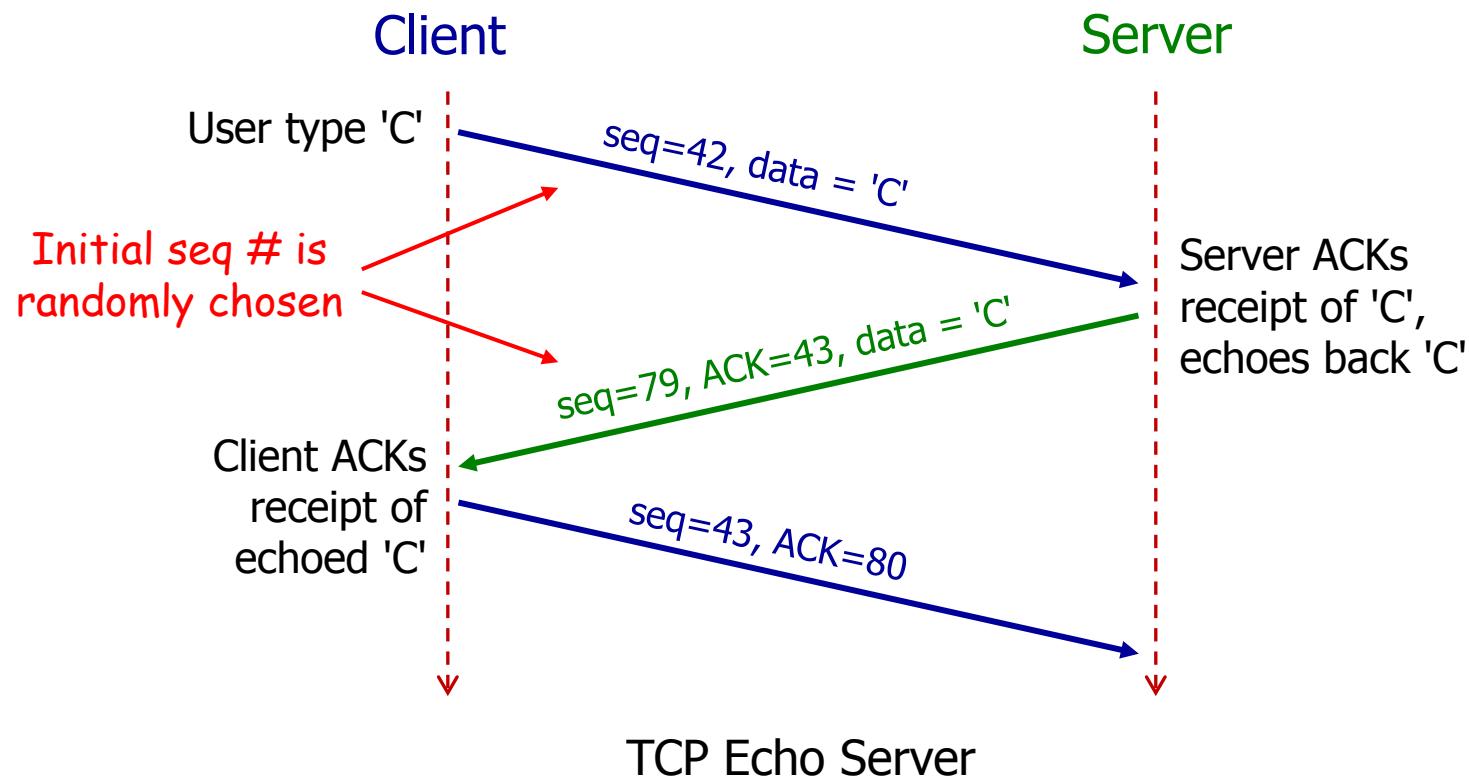
- seq # of the next byte of data expected by receiver.
- TCP ACKs up to the first missing byte in the stream (**cumulative ACK**).



**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 */

```

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 \*/

Sender keep one timer only

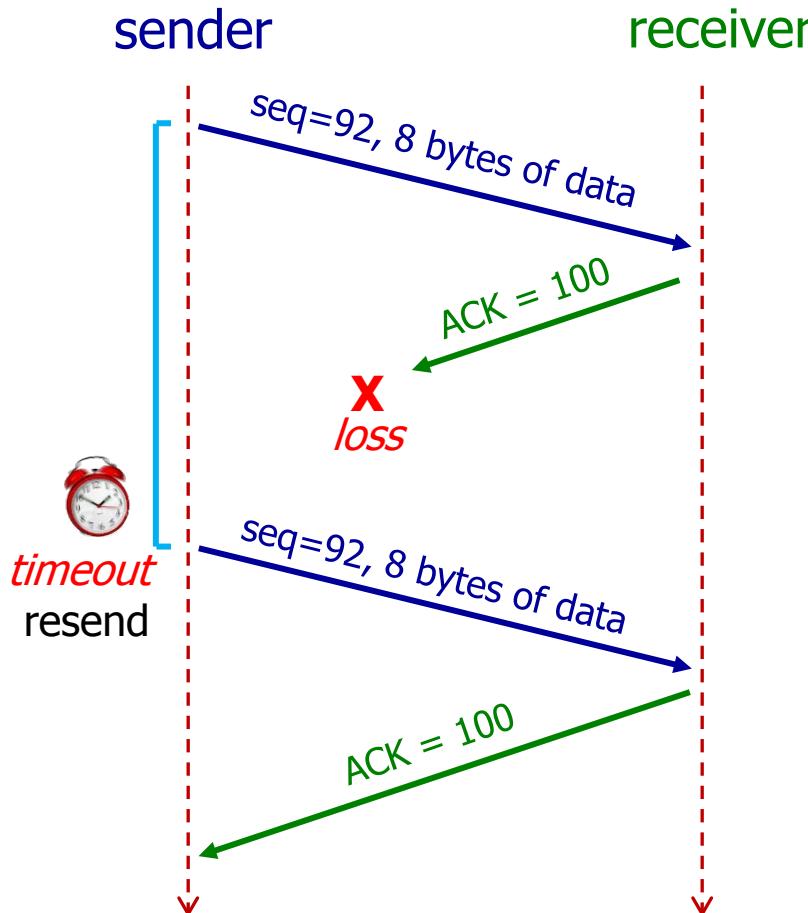
Retransmit only oldest unACKed packet

Cumulative ACK

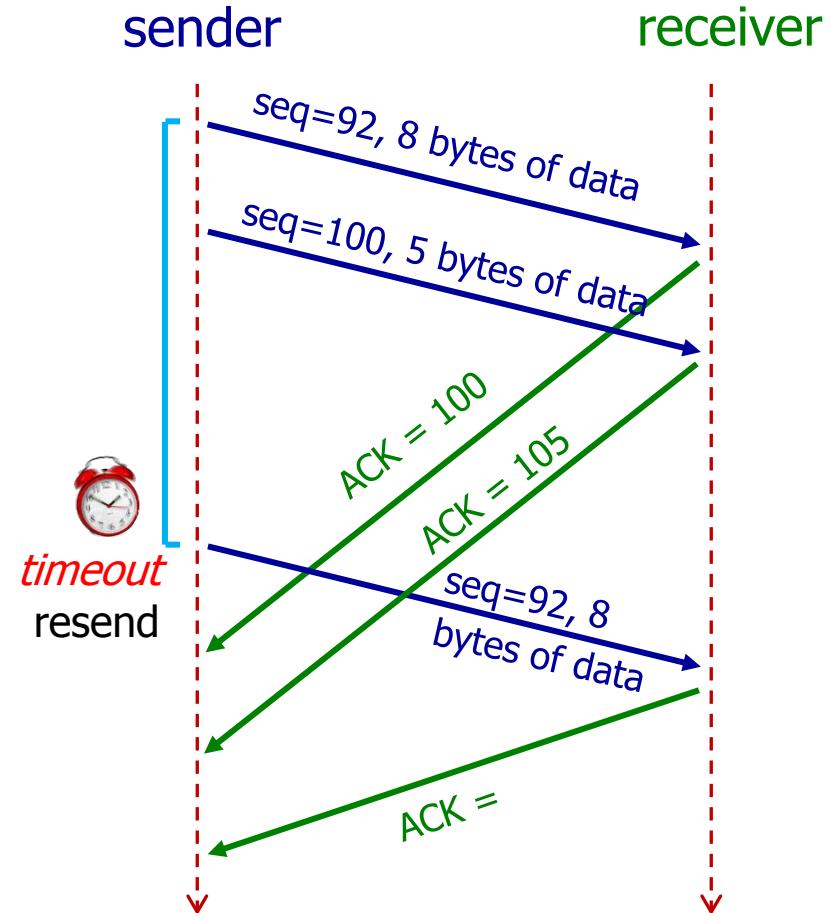
# TCP Receiver Events

| <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 formed and detected)          | Immediately send <i>duplicate ACK</i> , indicating seq. # of next expected byte (cumulative ACK) |
| Arrival of segment that partially or completely fills gap                                    | Immediately send ACK, provided that segment starts at lower end of gap                           |

# TCP Timeout / Retransmission



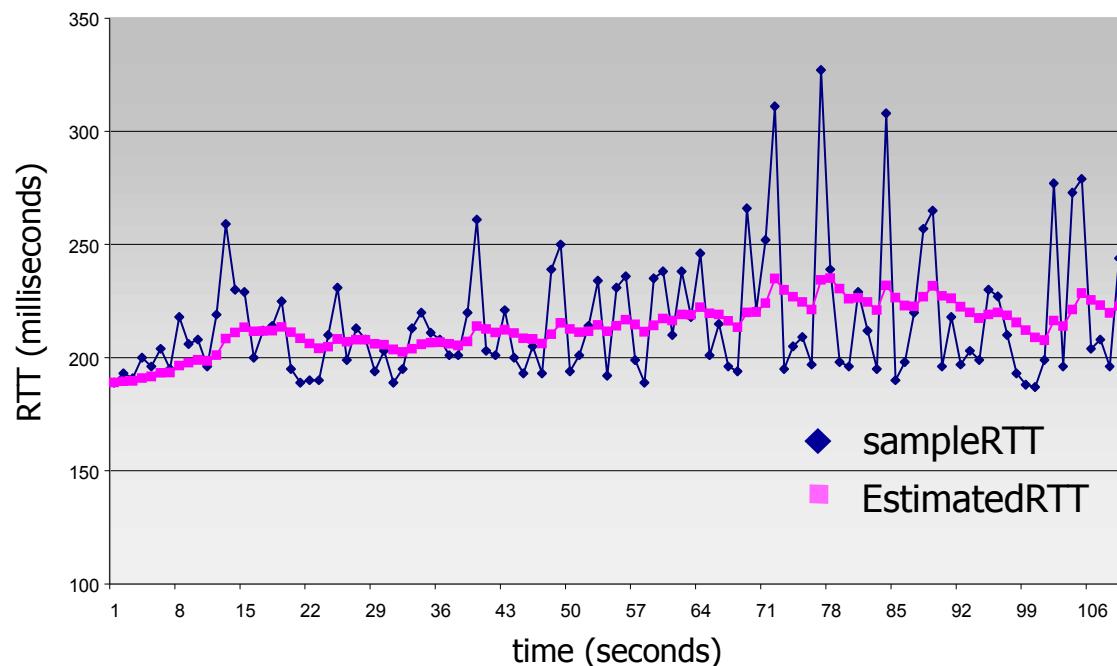
a) Lost ACK



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** =  $(1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$   
(typical value of  $\alpha$  : 0.125)

**DevRTT** =  $(1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$   
(typical value of  $\beta$  : 0.25)

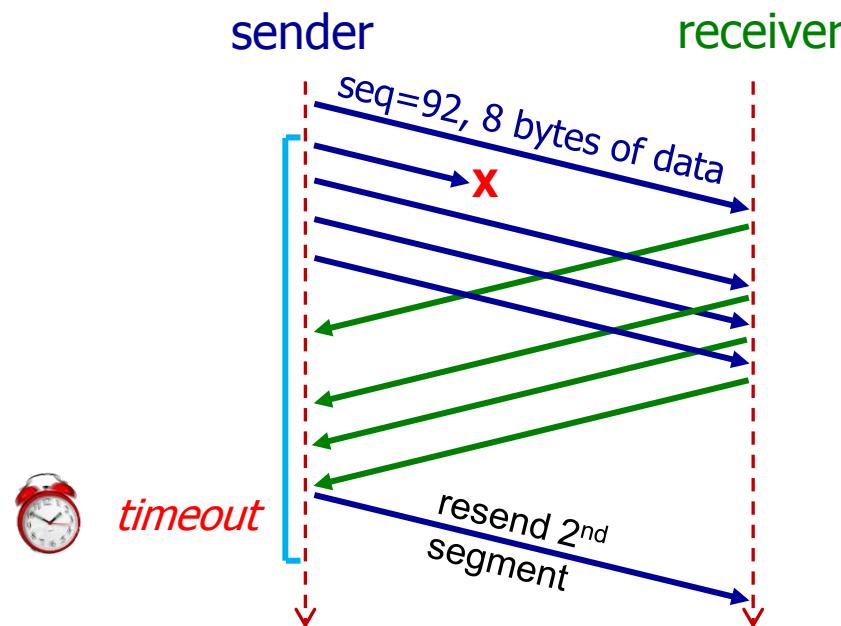
**TimeoutInterval** = **EstimatedRTT** +  $4 * \text{DevRTT}$



↑  
“safety margin”

# TCP Fast Retransmission

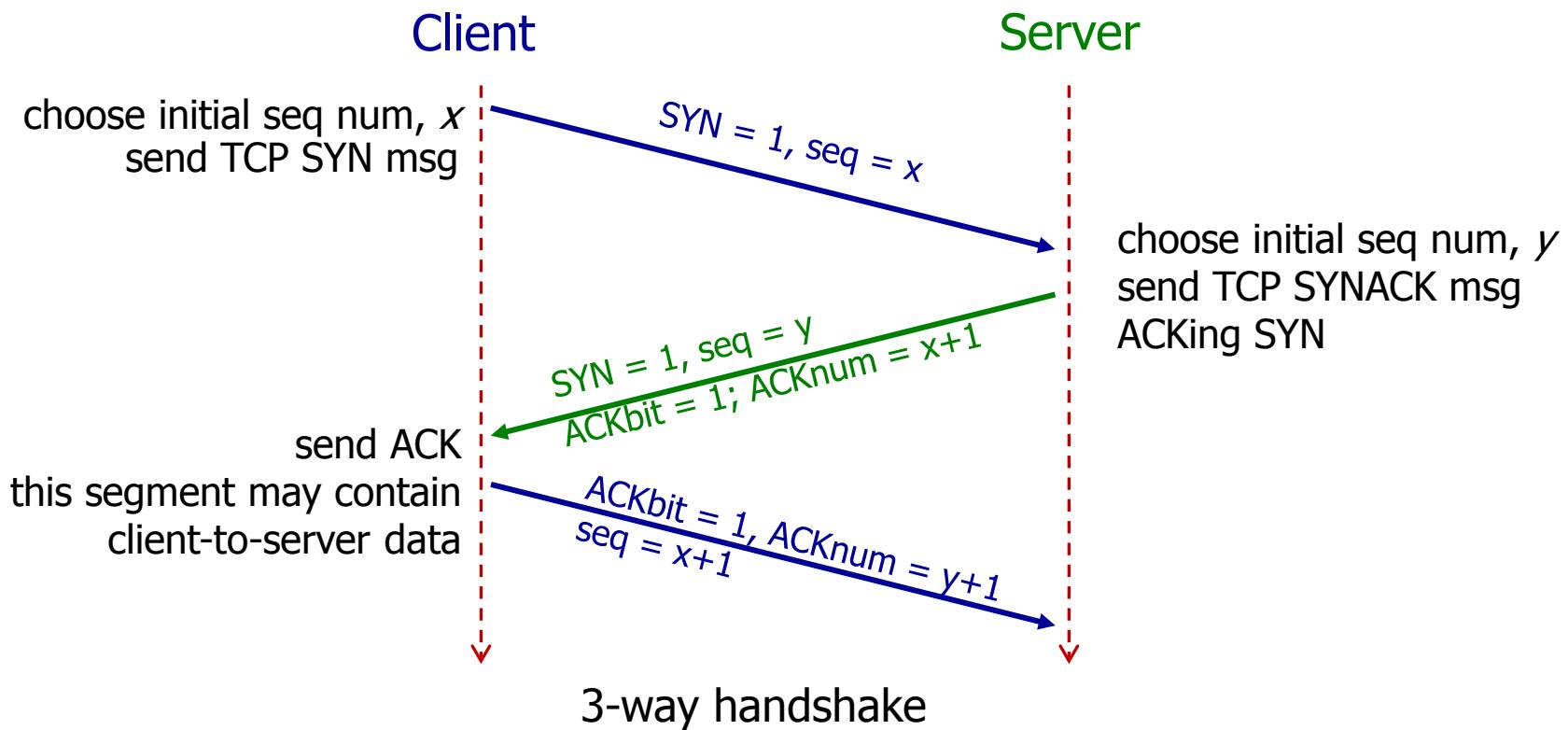
- ❖ Timeout period is often relatively long.
- ❖ [RFC2001] **Fast retransmission**:
  - **Event**: If sender receives 3 duplicate ACKs for the same data, it supposes that segment after ACKed data is lost.
  - **Action**: resend segment (even before timer expires).



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

# Establishing Connection

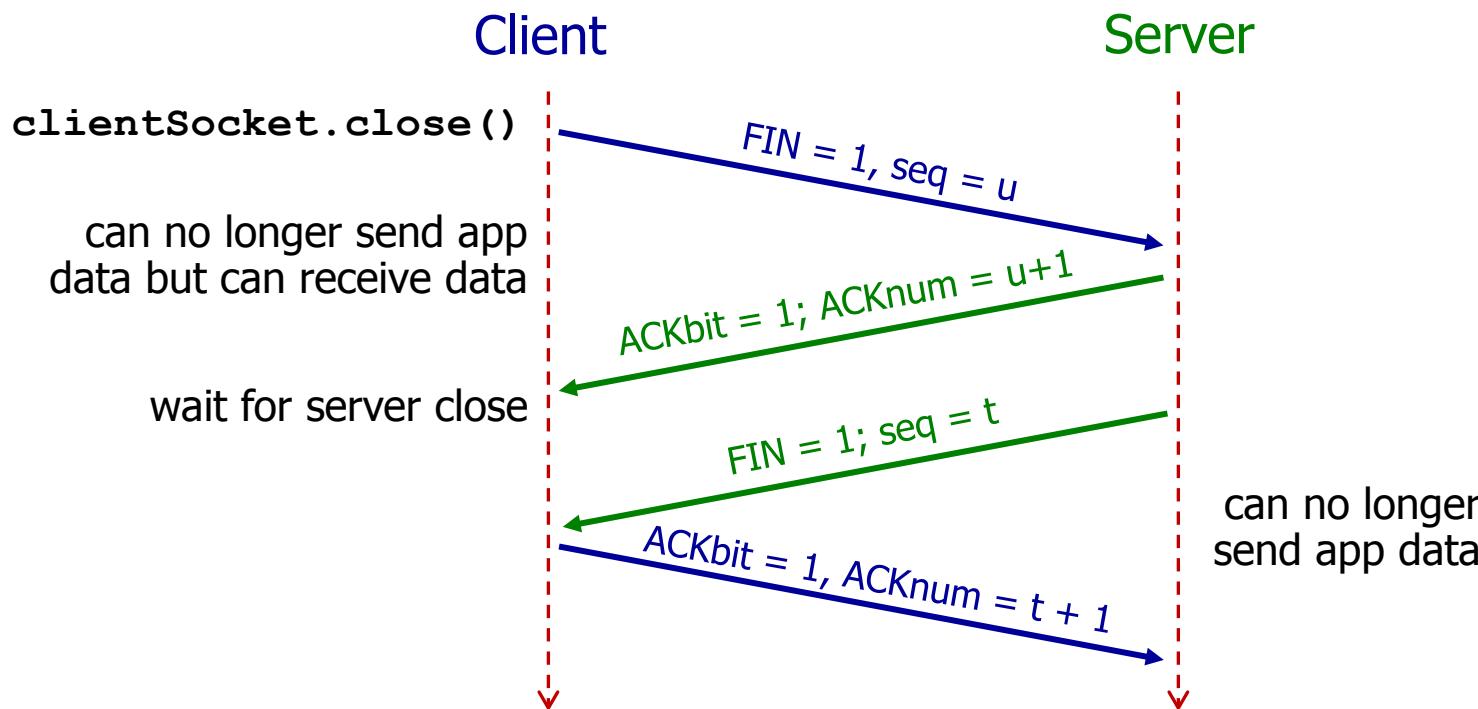
- ❖ Before exchanging data, TCP sender and receiver “shake hands”.
  - Agree on connection and exchange connection parameters.



# 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)

# Lecture 5: Summary

- ❖ Connection-oriented transport: TCP
  - Segment structure
  - Reliable data transfer
  - Setting and updating retransmission time interval
  - 3-way handshake

# An Awesome Introduction to Computer Networks

# Stop-and-wait Protocols

| rdt Version | Scenario  | Features Used  |
|-------------|---|--|
| 1.0         | no error  | nothing  |
| 2.1         | data packet Bit Error<br>feedback packet Bit Error                | checksum, ACK/NAK,<br>sequence number                          |
| 2.2         | data packet Bit Error<br>feedback packet Bit Error                | checksum, ACK,<br>sequence number                              |
| 3.0         | data packet Bit Error<br>feedback packet Bit Error<br>packet Loss | checksum, ACK, sequence<br>number, timeout/re-<br>transmission |

# Pipelined Protocols

|                      | Go-Back-N        | Selective Repeat       |
|----------------------|------------------|------------------------|
| Out-of-order packets | ignore           | buffer                 |
| ACK                  | cumulative       | individual             |
| Timer                | earliest unACKed | one per unACKed packet |
| Retransmission       | all unACKed      | one unACKed            |

# Lecture 6: IP Addresses

*After this class, you are expected to:*

- ❖ describe the basic services network layer provides.
- ❖ understand the purpose of DHCP and how it works.
- ❖ understand IP address, subnet, subnet mask and address allocation.
- ❖ understand how longest prefix forwarding in a router works.

# Lecture 6: Roadmap

## 4.1 Introduction

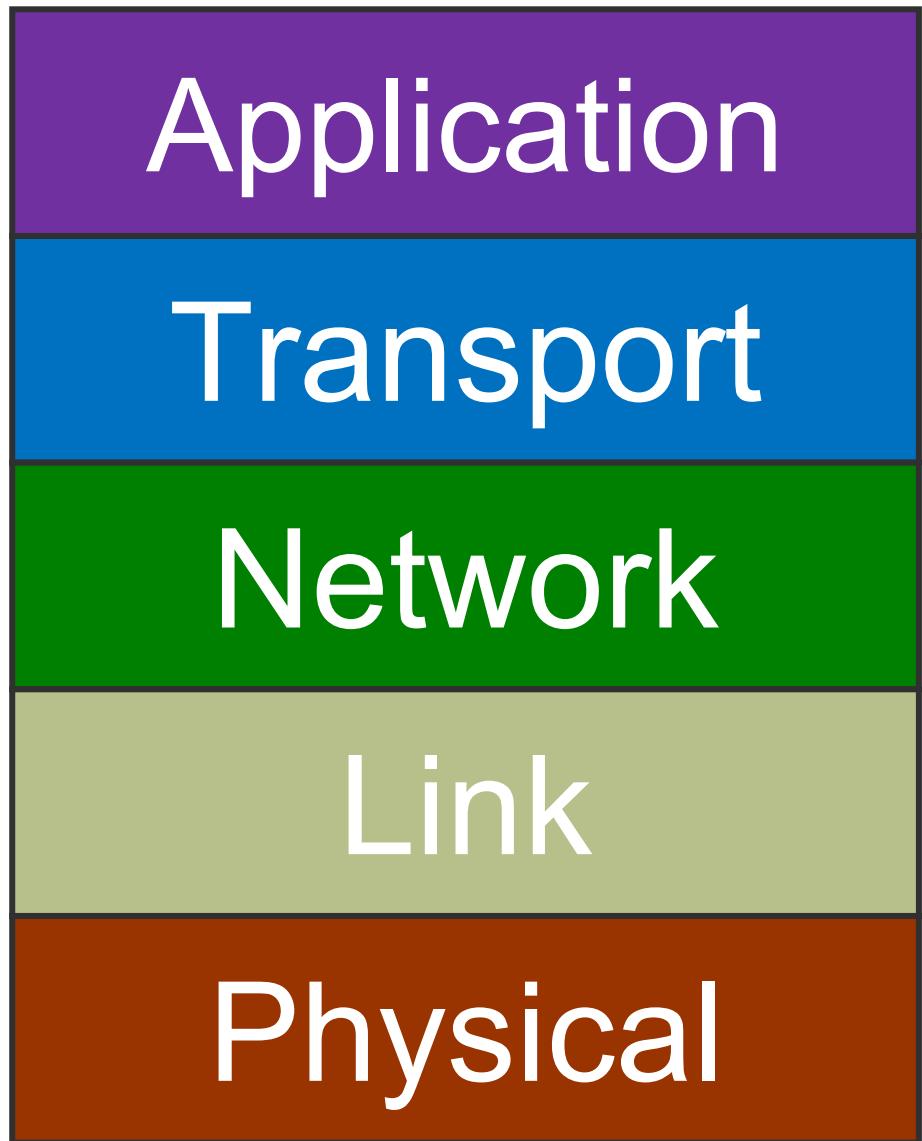
## 4.4 IP: Internet Protocol

- 4.4.1 Datagram Format
- 4.4.2 IPv4 Addressing
- 4.4.3 ICMP

## 4.5 Routing Algorithms

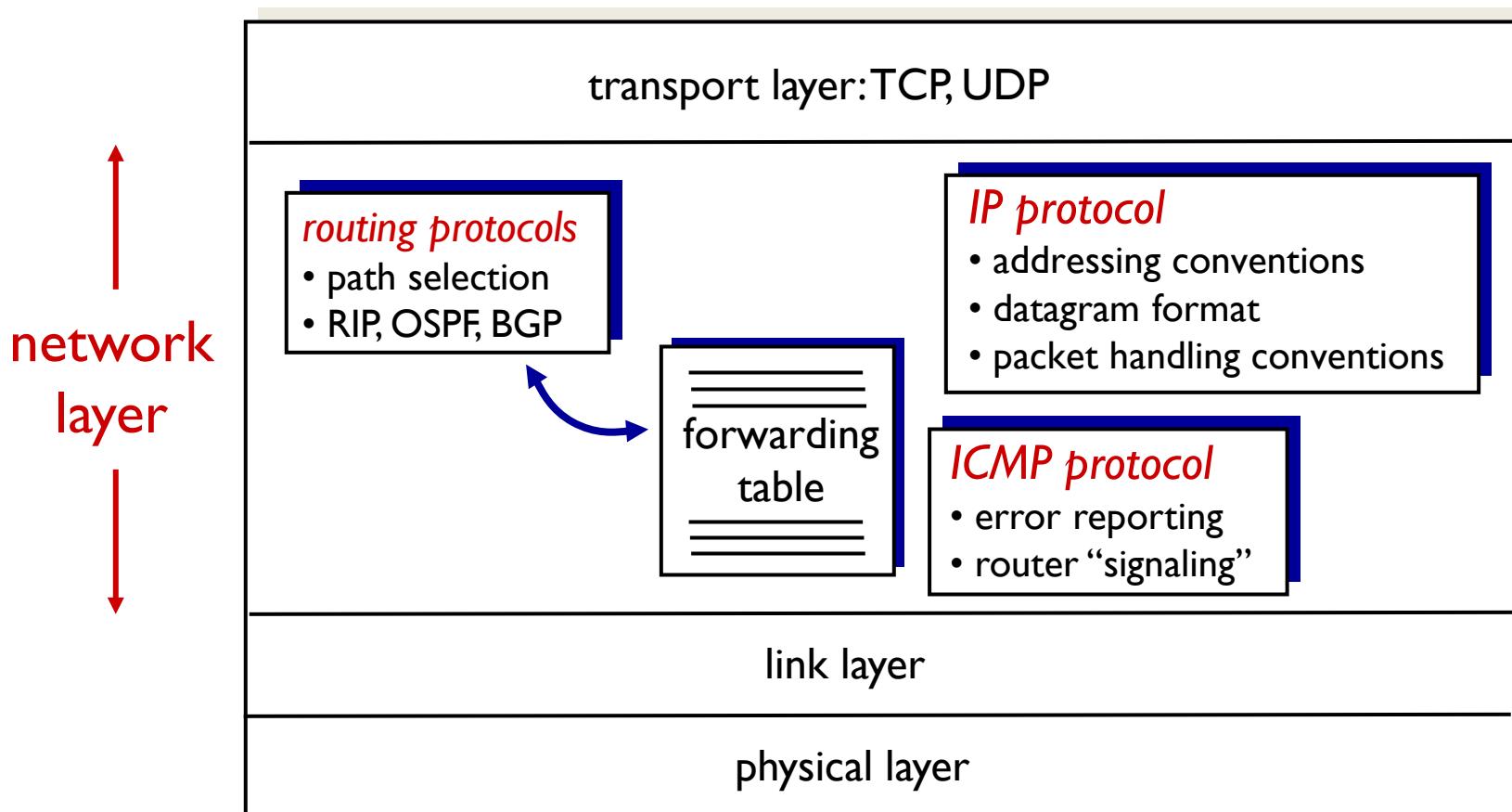
To discuss  
next week

Kurose Textbook, Chapter 4  
(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.



# Lecture 6: Roadmap

## 4.1 Introduction

## 4.4 IP: Internet Protocol

- 4.4.1 Datagram Format
- 4.4.2 IPv4 Addressing
- 4.4.3 ICMP

## 4.5 Routing Algorithms

# IP Address

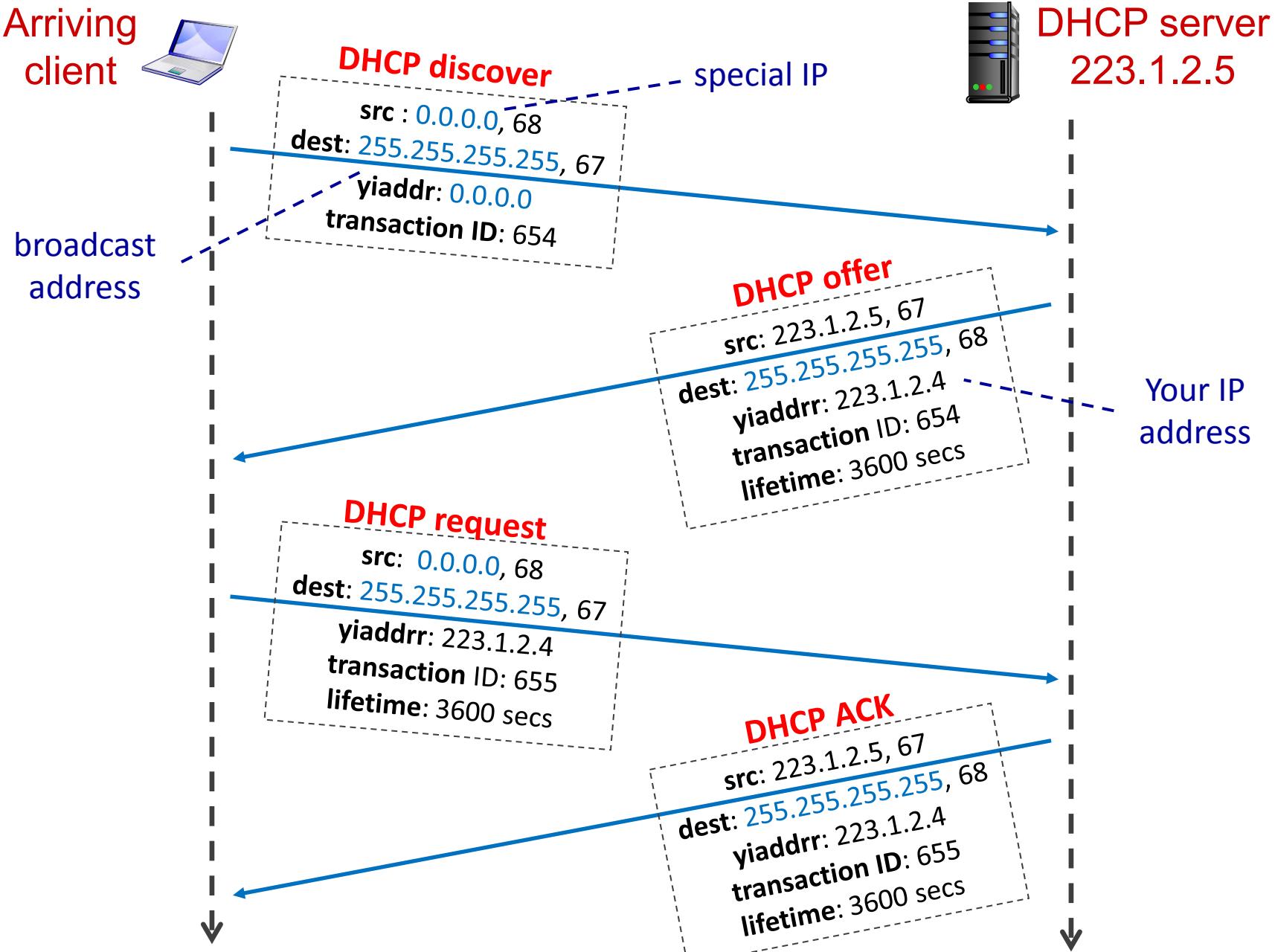
- ❖ IP address is used to identify a host (or a router).
  - A 32-bit integer expressed in either binary or decimal

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

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

# DHCP

- ❖ **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



# 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

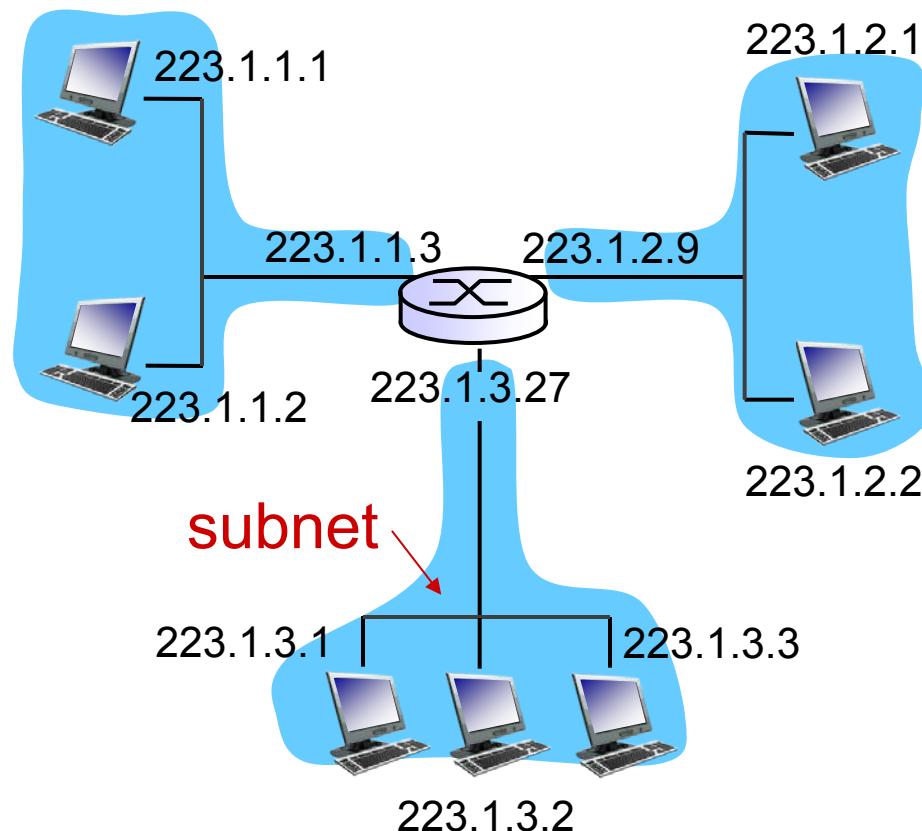
# Some Special IP Addresses

| Special Addresses                             | Present Use   |
|---|---|
| 0.0.0.0/8                                     | Non-routable meta-address for special use   |
| 127.0.0.0/8                                   | 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<br>172.16.0.0/12<br>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

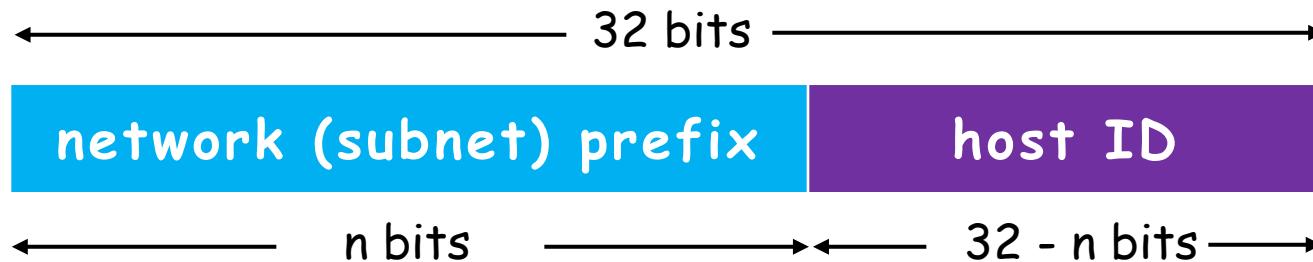
- ❖ An IP address is associated with a **network interface**.
  - A host usually has one or two network interfaces (e.g. wired Ethernet and WiFi).
  - A router typically has multiple interfaces.



A network consisting of 3 subnets  
(first 24 bits of IP addr. are network prefix)

# IP Address and Subnet

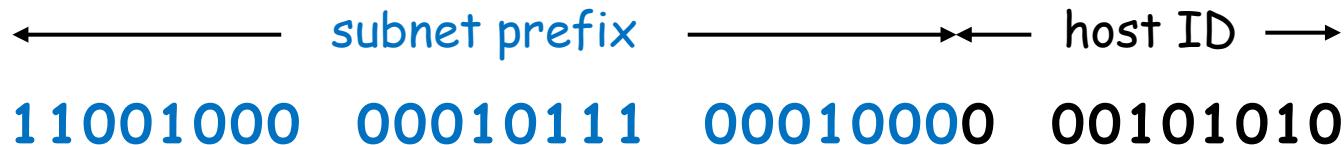
- ❖ An IP address logically comprises two parts:



- ❖ **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.
  - 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.



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 what 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<br>in binary   | 11001000          | 00010111 | 00010000 | 00101010 |           |
| Subnet mask               | 11111111          | 11111111 | 11111110 | 00000000 |           |
| Subnet mask<br>in decimal | 255.255.254.0     |          |          |          |           |

# Quiz

- ❖ 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

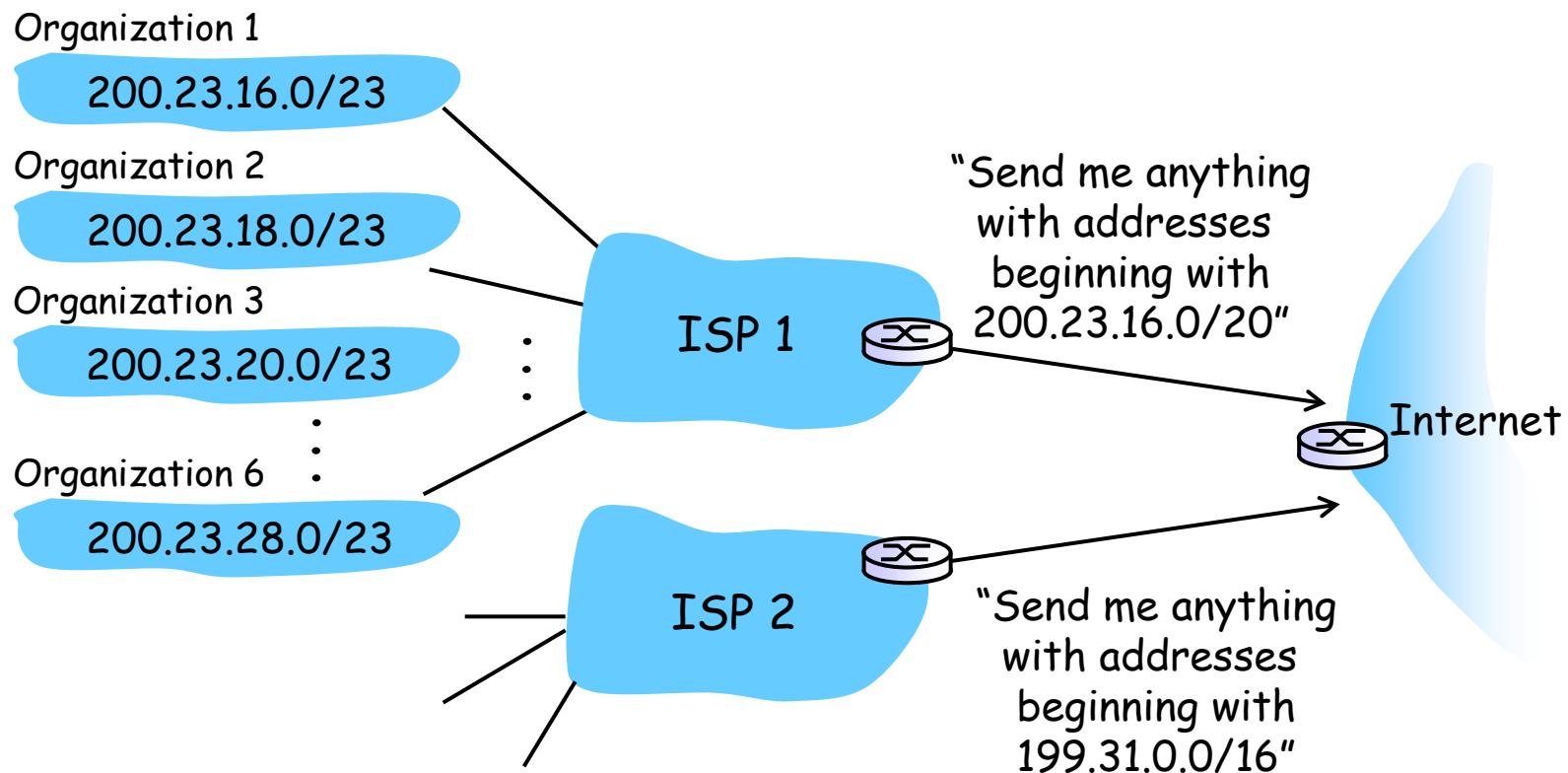
- ❖ **Q:** How does an organization obtain a block of IP addresses?
- ❖ **A:** Buy from registry or rent from ISP's address space.

|                | Binary Address                                | Decimal Address |
|----------------|---|-----------------|
| ISP's block    | 11001000 00010111 0001 000 0 00000000         | 200.23.16.0/20  |
| Organization 1 | 11001000 00010111 0001 0 <b>0</b> 0 00000000  | 200.23.16.0/23  |
| Organization 2 | 11001000 00010111 0001 0 <b>01</b> 0 00000000 | 200.23.18.0/23  |
| Organization 3 | 11001000 00010111 0001 0 <b>10</b> 0 00000000 | 200.23.20.0/23  |
| ...            | ...   | ...             |
| Organization 6 | 11001000 00010111 0001 1 <b>10</b> 0 00000000 | 200.23.28.0/23  |

use 3 more bits to differentiate  
6 organizations

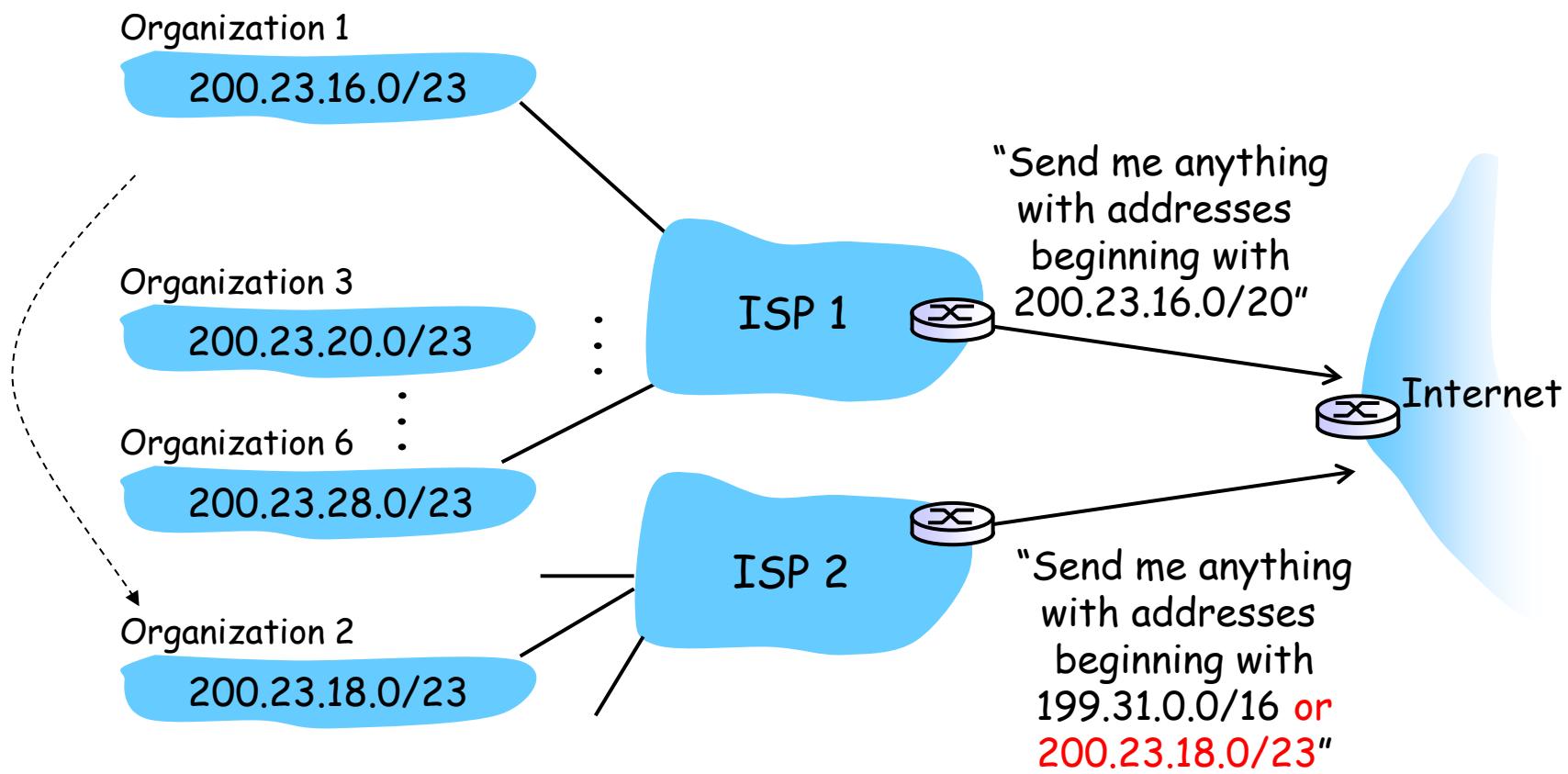
# Hierarchical Addressing

Hierarchical addressing allows efficient advertisement of routing information:



# Hierarchical Addressing

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



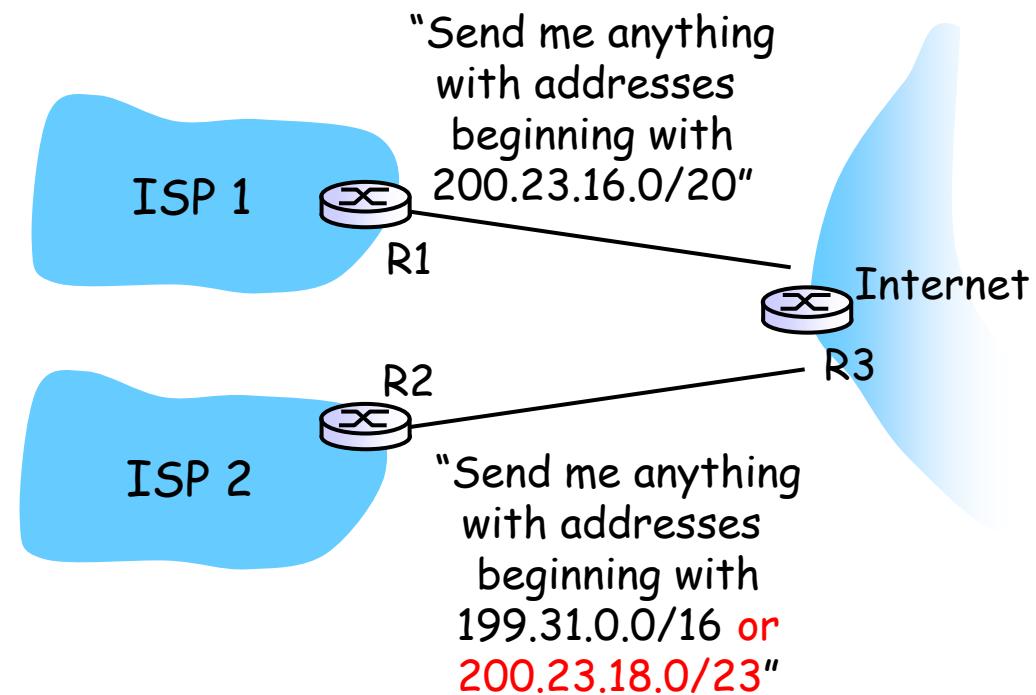
# Longest Prefix Match (1/2)

❖ **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       |

...

...

# IP Address Allocation

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

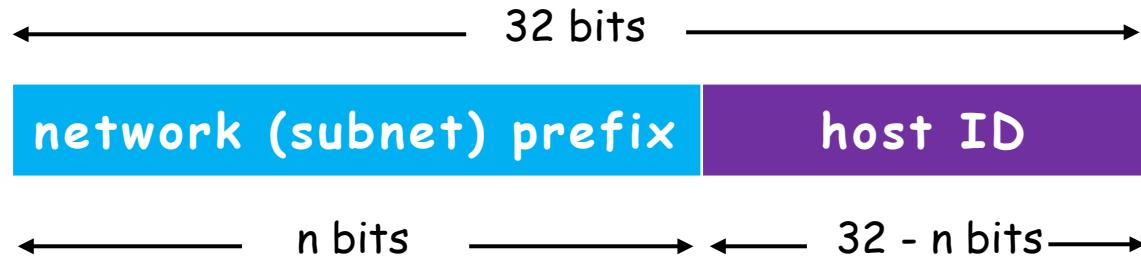
# Lecture 6: Summary

- ❖ 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.

# An **AWESOME** Introduction to Computer Networks

# IP Address

- ❖ **IP address** is used to identify a host (or router).
  - IP address is a 32-bit integer comprising two parts:

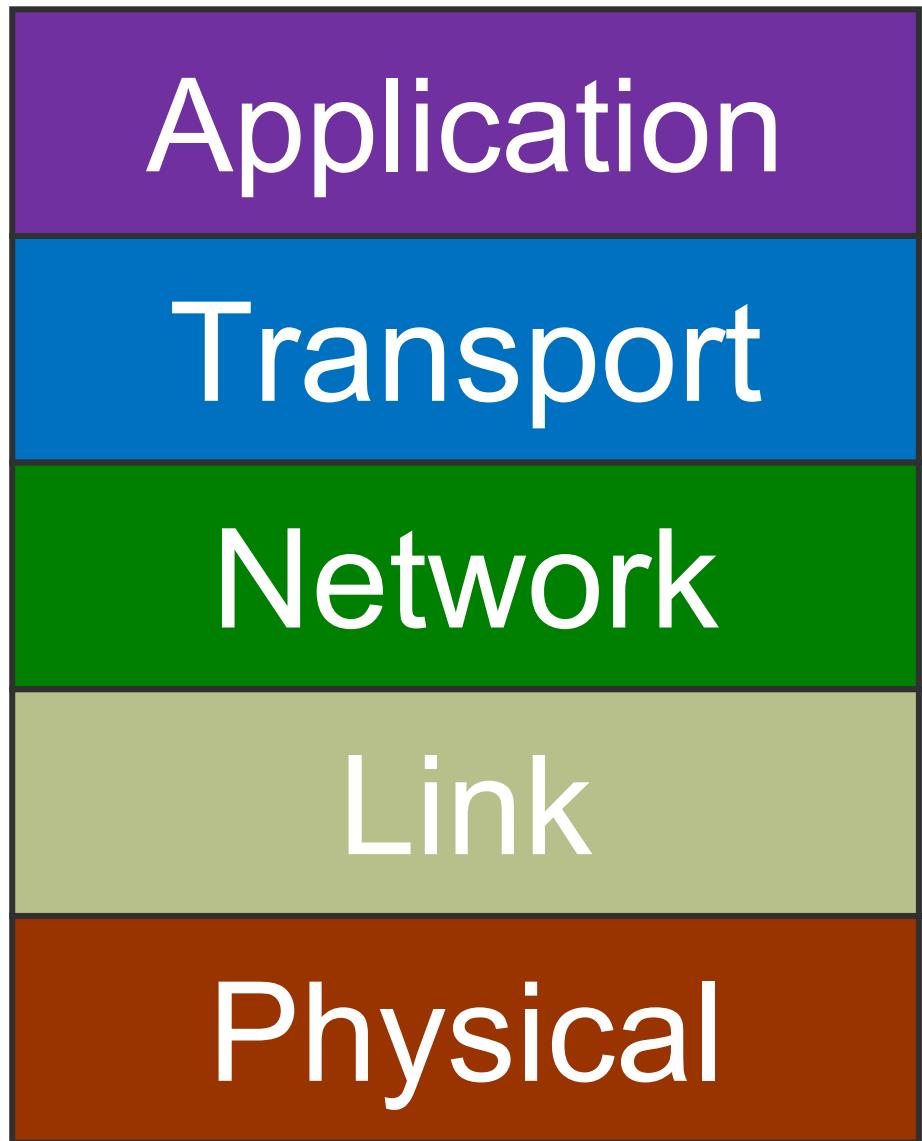


- ❖ **Subnet** is a network of directly connected hosts.
  - Hosts in the same subnet have the same network prefix of IP address, but different host id.
  - Hosts in the same subnet can physically reach each other directly, but connect to outside world through a router.

# Lecture 7: IP and Routing

*After this class, you are expected to understand:*

- ❖ the purpose of routing protocols on the Internet.
- ❖ the principle of Bellman-Ford equation.
- ❖ the workings of distance vector algorithm.
- ❖ how RIP works.
- ❖ the purpose of NAT and how it works.
- ❖ the Internet Protocol (IP) and how datagram fragmentation works.



# Lecture 7: Roadmap

## 4.1 Introduction

## 4.4 IP: Internet Protocol

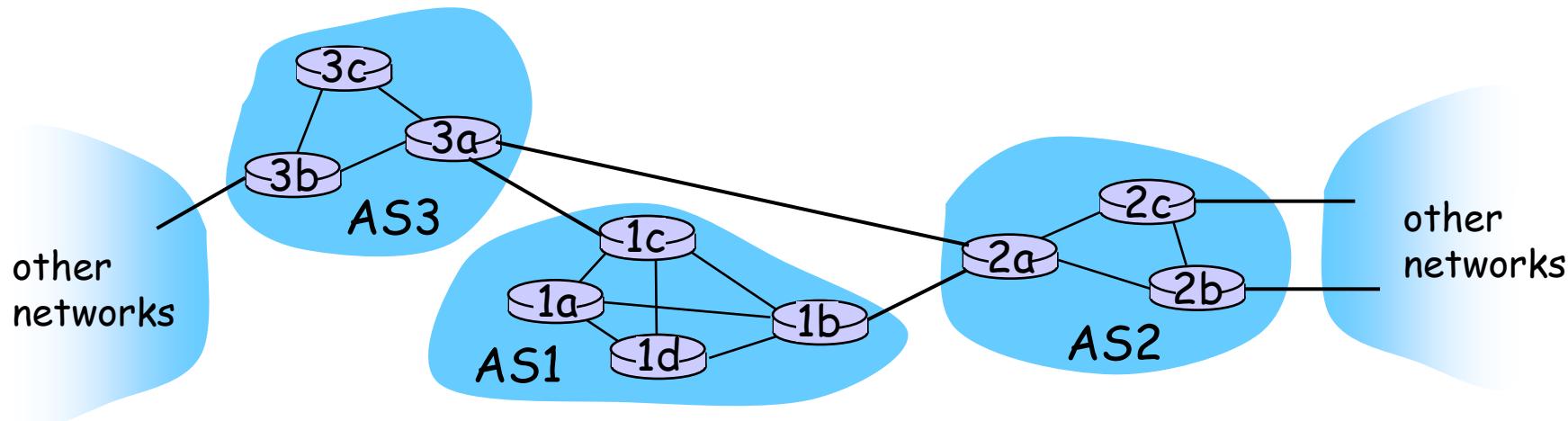
- 4.4.1 Datagram Format
- 4.4.2 IPv4 Addressing
- 4.4.3 ICMP

## 4.5 Routing Algorithms

Kurose Textbook, Chapter 4  
(Some slides are taken from the book)

# 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.



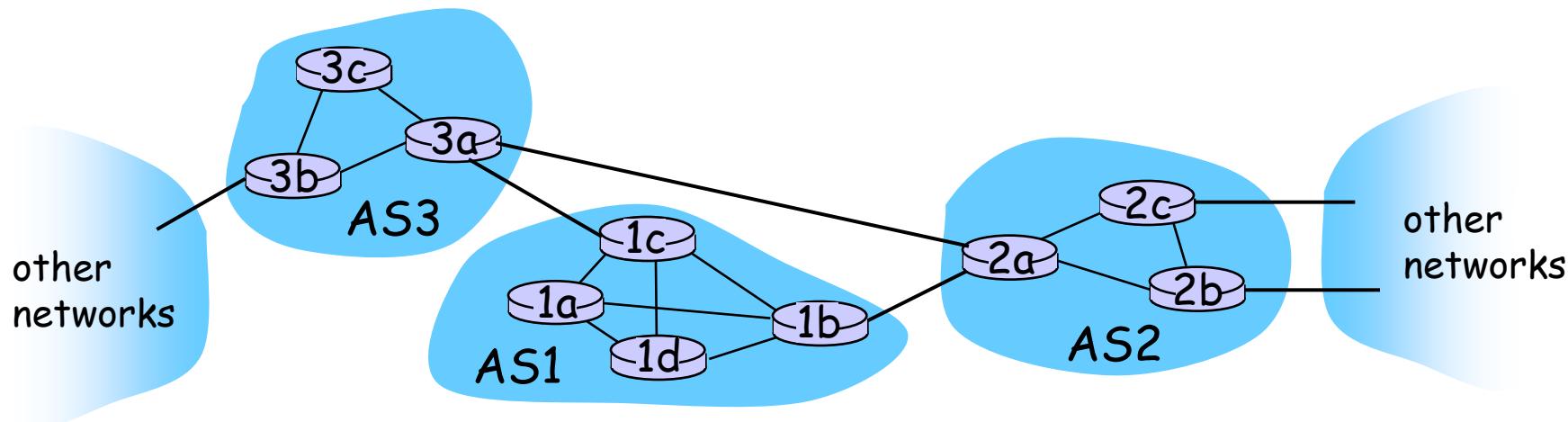
# Routing in The Internet

## ❖ 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**



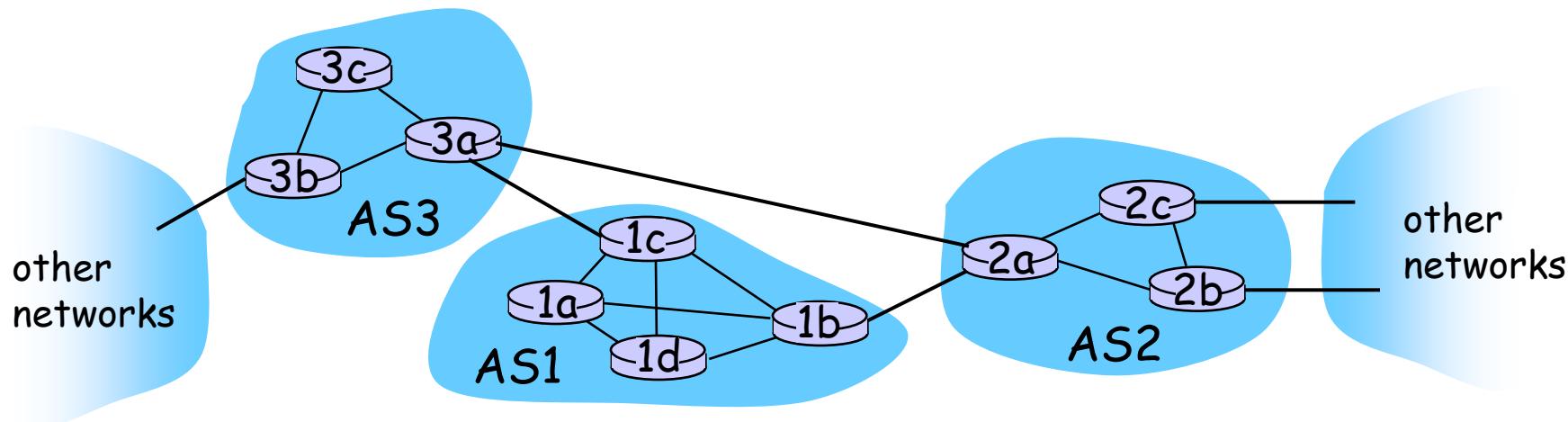
# Routing in The Internet

## ❖ Intra-AS routing

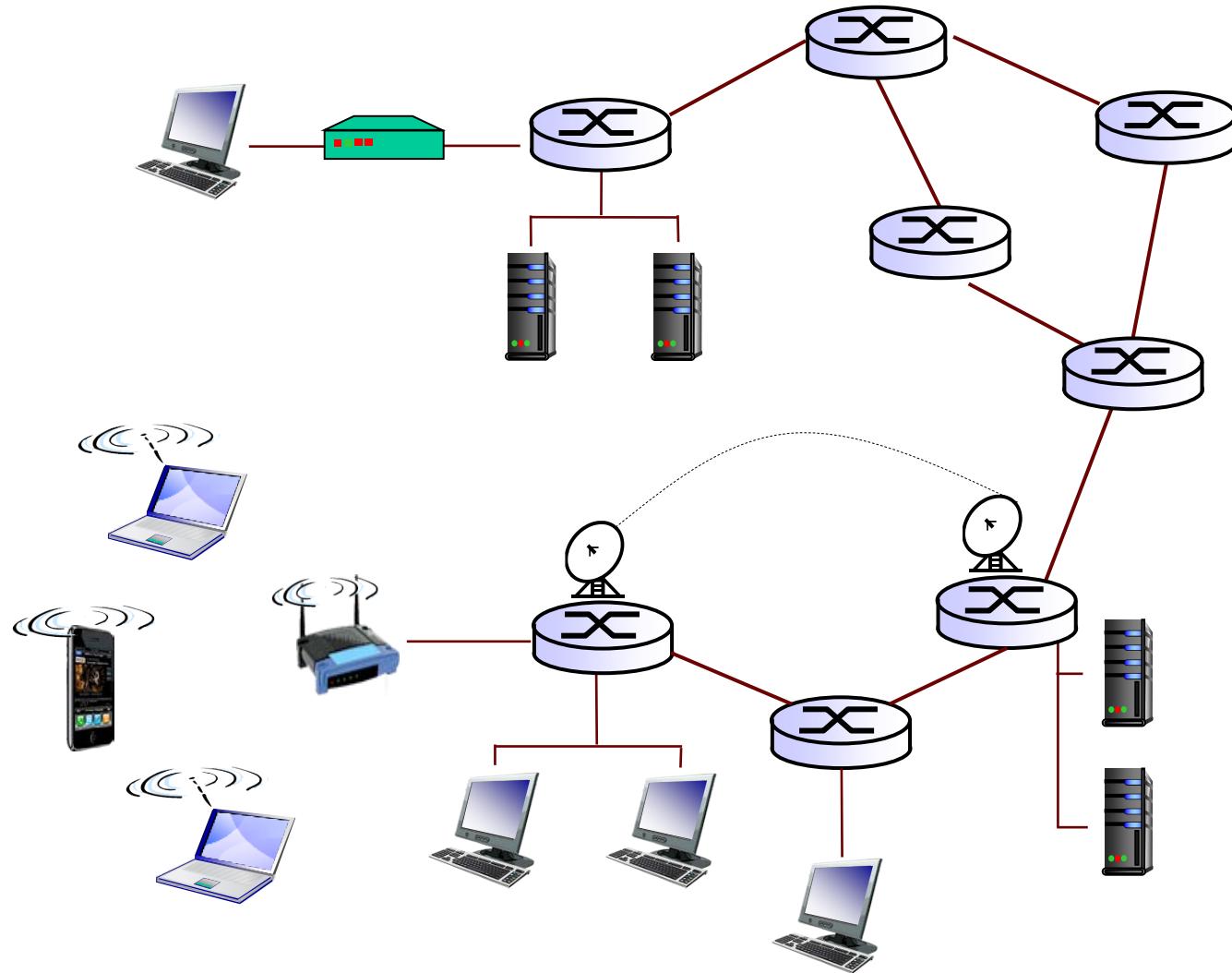
- Single admin, so no policy decisions is needed.
- Routing mostly focus on performance.

## ❖ 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.

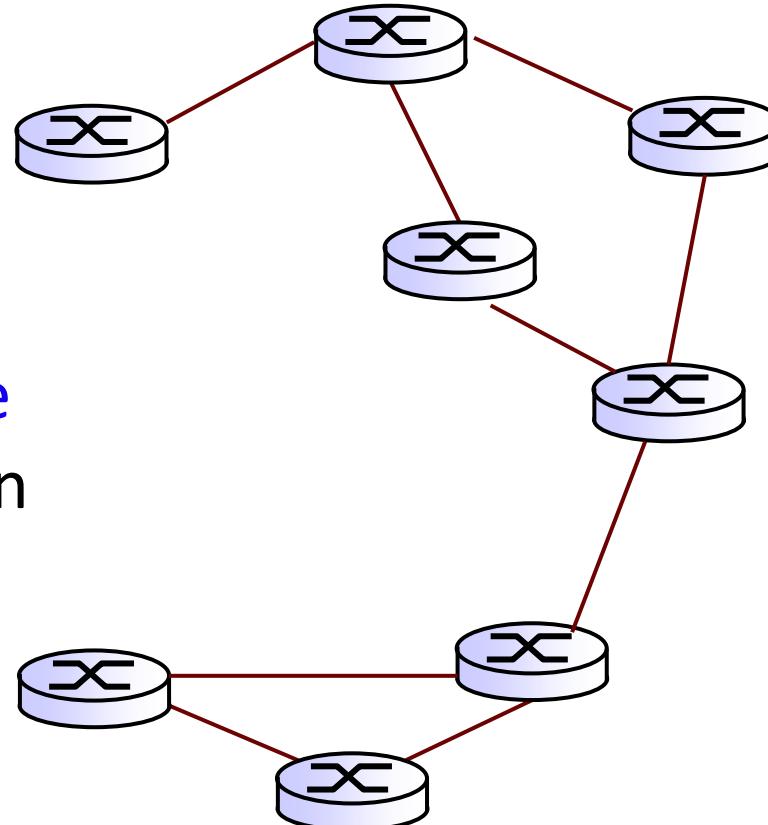


# Abstract View of Intra-AS Routing



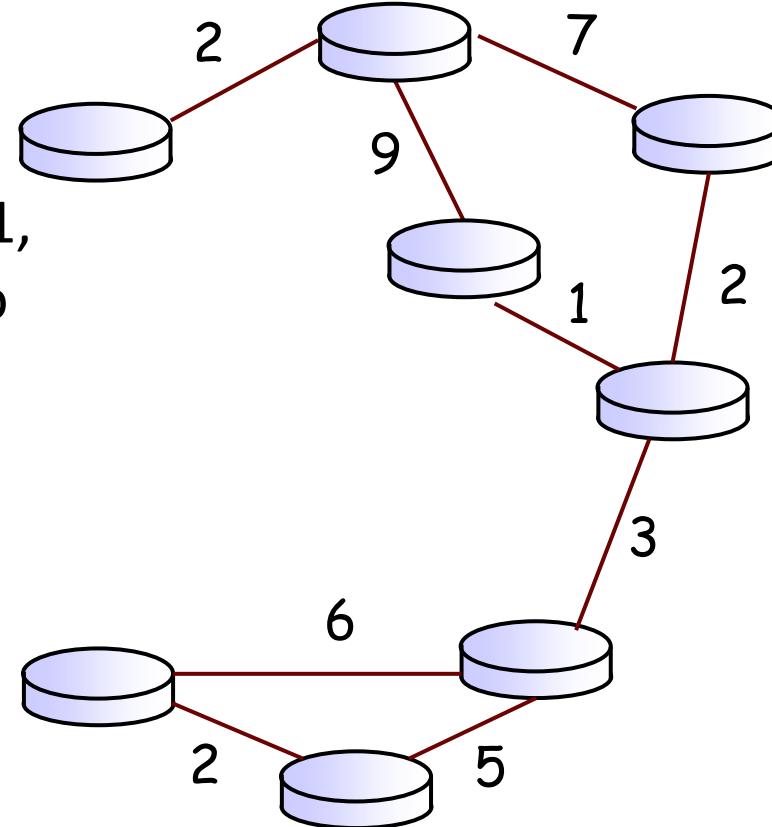
# Abstract View of Intra-AS Routing

- ❖ 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.

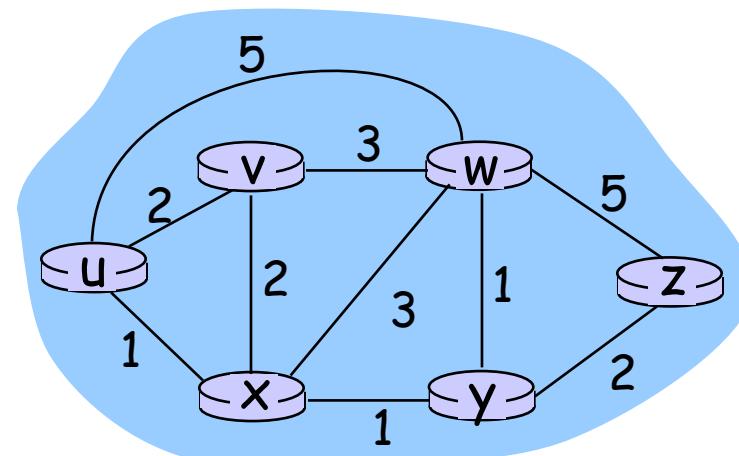


Routing: finding a least cost path between two vertices in a graph

# Routing Algorithms Classification

## *“link state” algorithms*

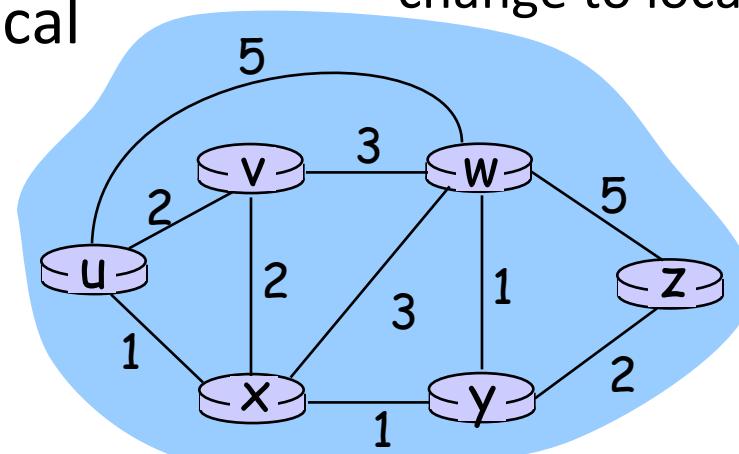
- ❖ All routers have the complete knowledge of network topology and link cost.
  - Routers periodically broadcast link costs to each other.
- ❖ Use Dijkstra algorithm to compute least cost path locally (using global map).
- ❖ Not covered in CS2105 😊



# Routing Algorithms Classification

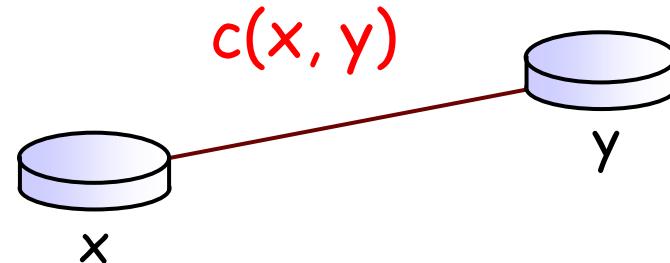
## *“distance vector” algorithms*

- ❖ 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.

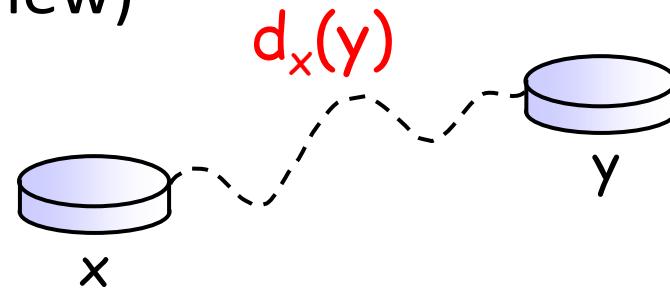


# Some Graph Notations

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



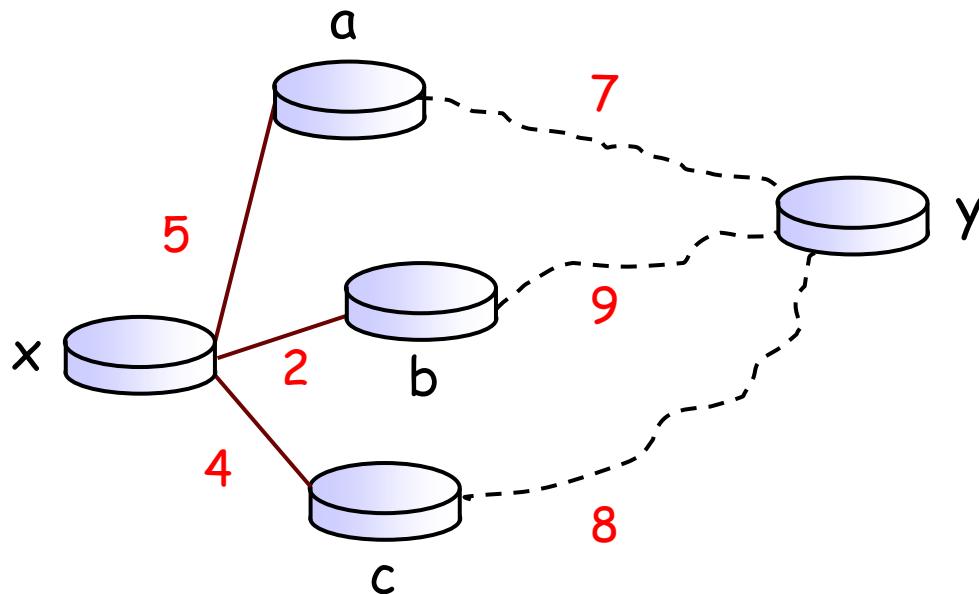
- ❖  $d_x(y)$ : the cost of the least-cost path from  $x$  to  $y$  (from  $x$ 's view)



# Bellman-Ford Equation

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

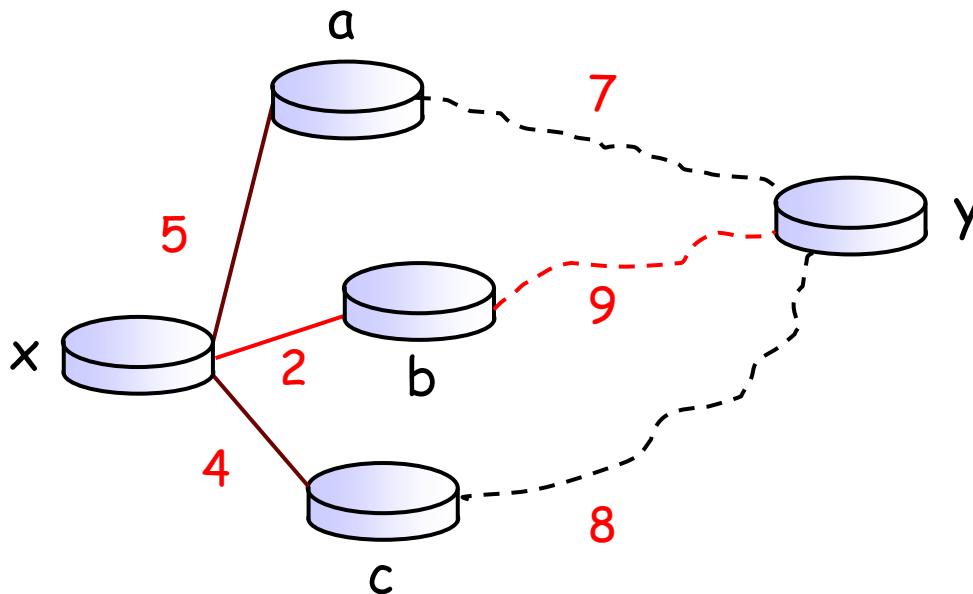
where min is taken over all direct neighbors v of x



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

# Bellman-Ford Equation

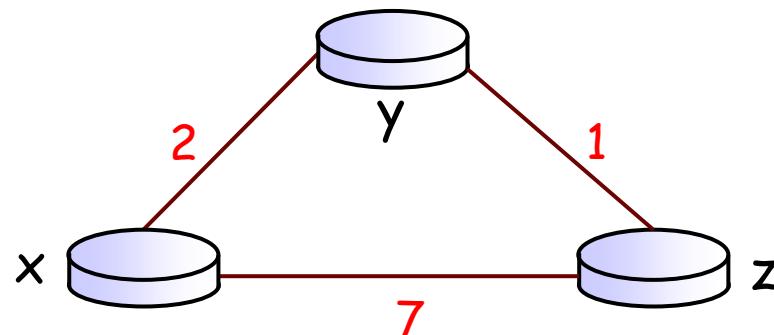
- ❖ To find the least cost path,  $x$  needs to know the cost from each of its direct neighbour to  $y$ .
- ❖ Each neighbour  $v$  sends its **distance vector**  $(y, k)$  to  $x$ , telling  $x$  that the cost from  $v$  to  $y$  is  $k$ .



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 |
| from | x |         |   |   |
|      | y |         |   |   |
|      | z |         |   |   |
|      |   | cost to |   |   |
|      |   | x       | y | z |
| from | x |         |   |   |
|      | y |         |   |   |
|      | z |         |   |   |
|      |   | cost to |   |   |
|      |   | x       | y | z |
| from | 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).
- ❖ Entries in the routing table are aggregated subnet masks (so we are routing to destination subnet).
- ❖ 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.

# Lecture 7: Roadmap

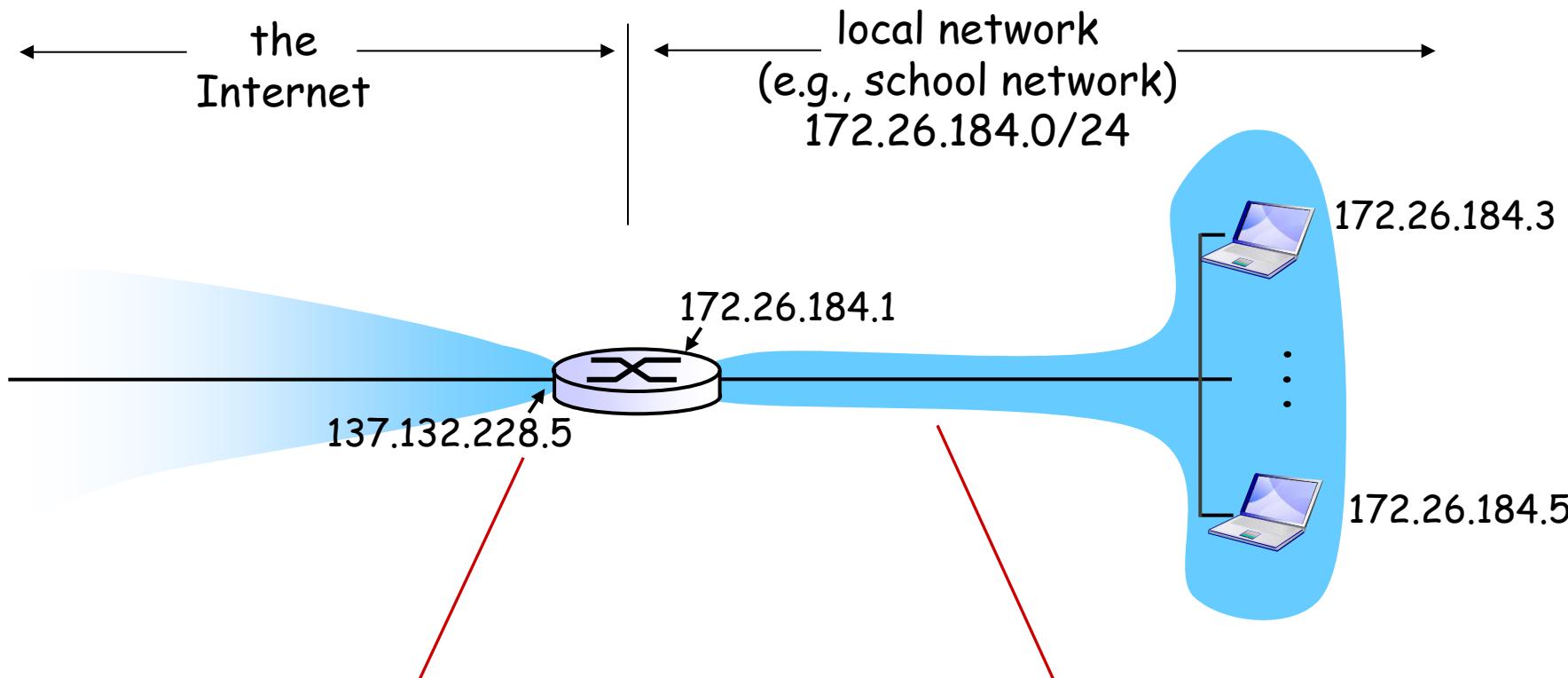
## 4.1 Introduction

## 4.4 IP: Internet Protocol

- 4.4.1 Datagram Format
- 4.4.2 IPv4 Addressing
- 4.4.3 ICMP

## 4.5 Routing Algorithms

# NAT: Network Address Translation



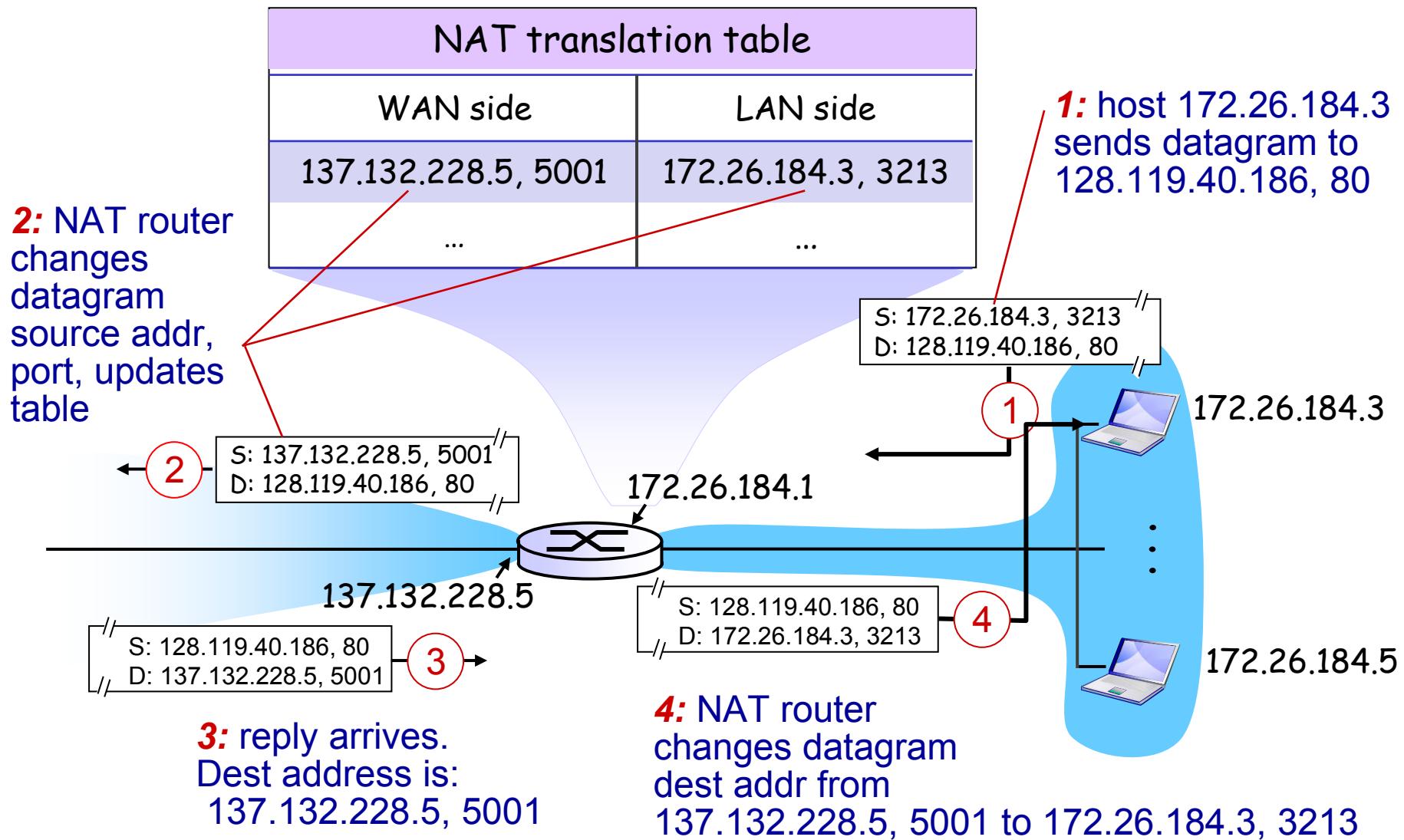
**all** datagrams *leaving* local network have the *same* source NAT IP address: 137.132.228.5

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).

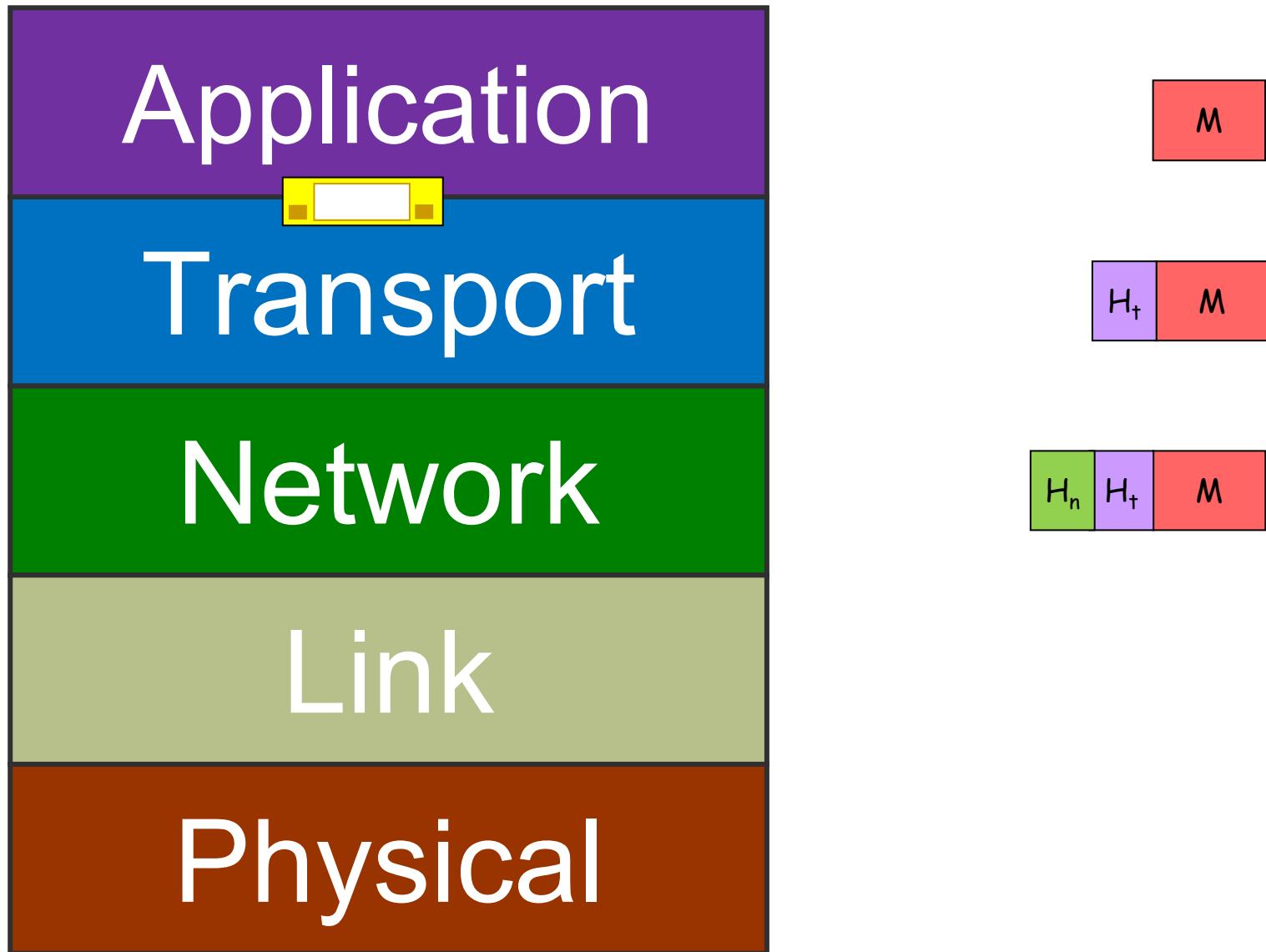
# Lecture 7: Roadmap

## 4.1 Introduction

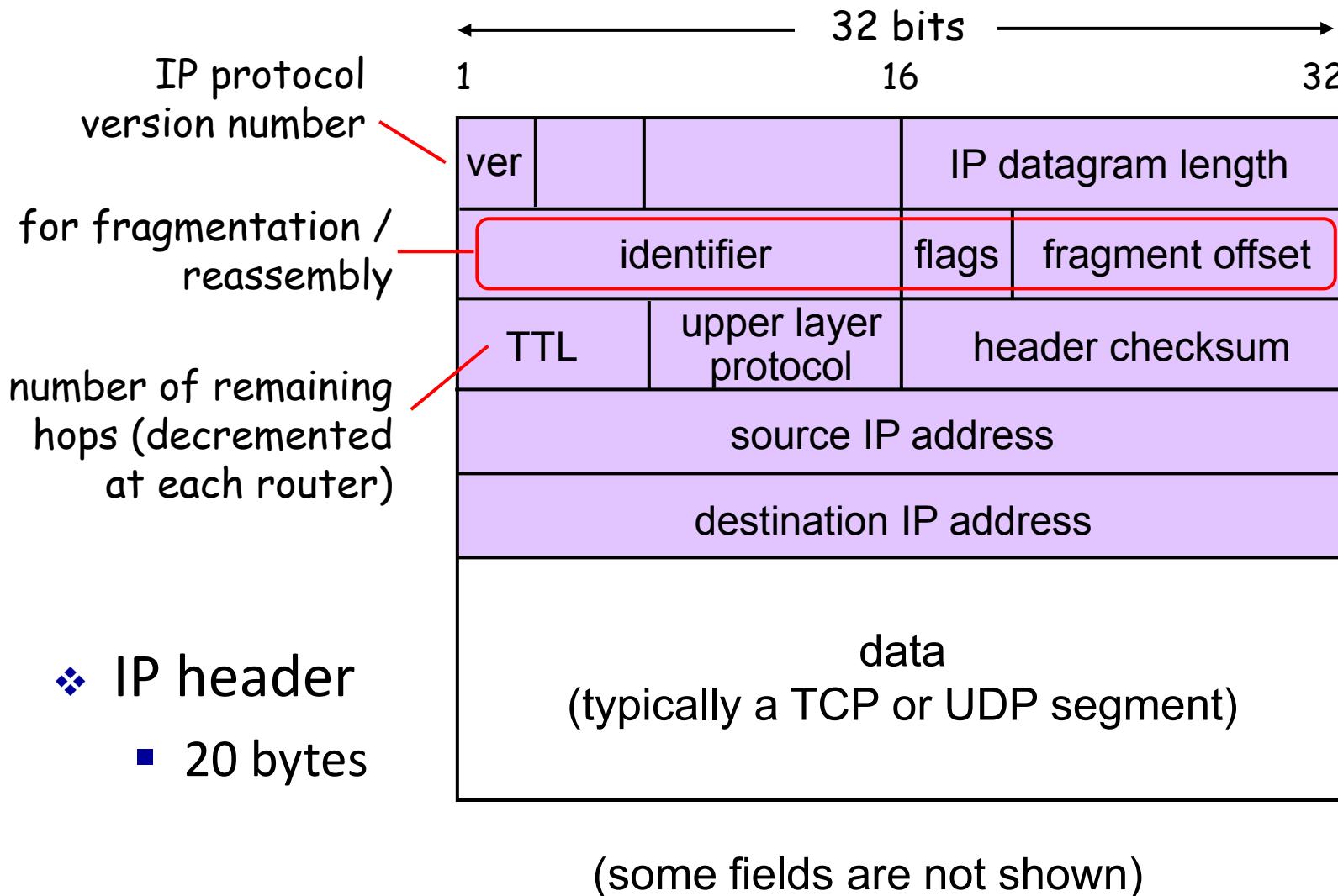
## 4.4 IP: Internet Protocol

- 4.4.1 Datagram Format
- 4.4.2 IPv4 Addressing
- 4.4.3 ICMP

## 4.5 Routing Algorithms

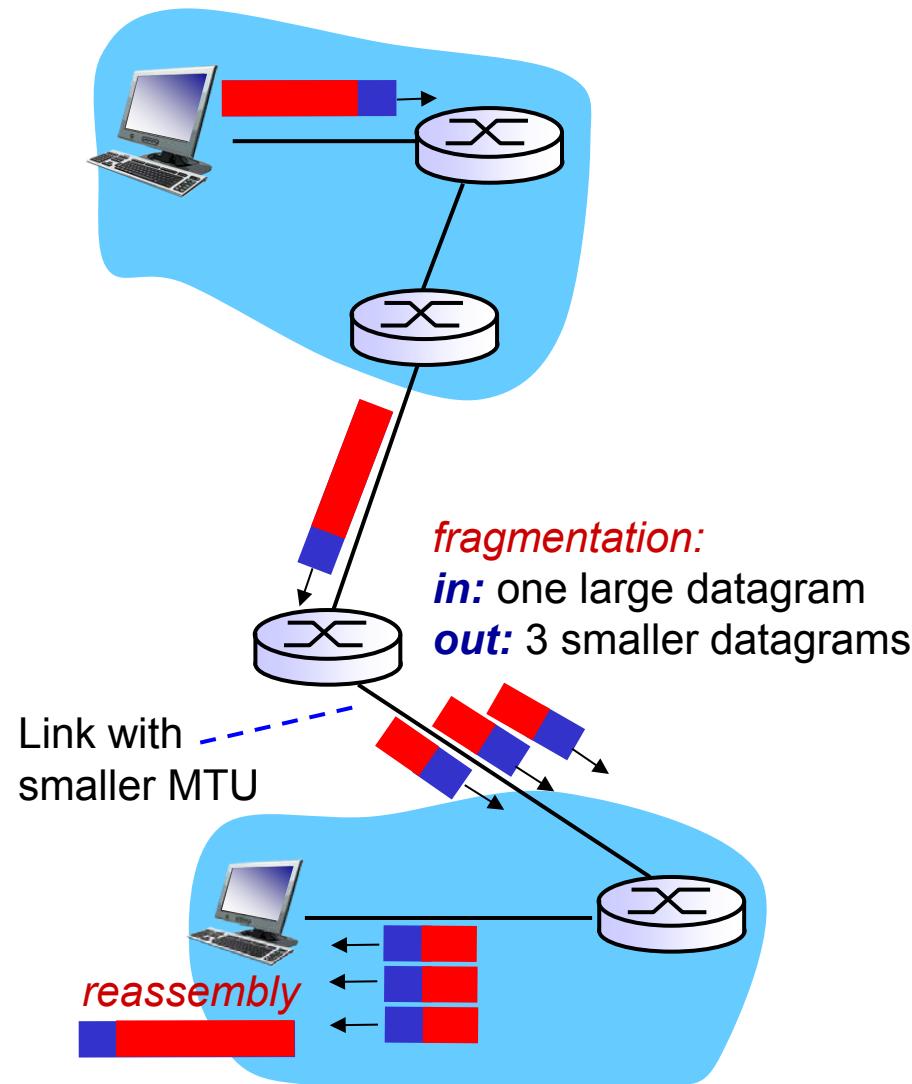


# IPv4 Datagram Format

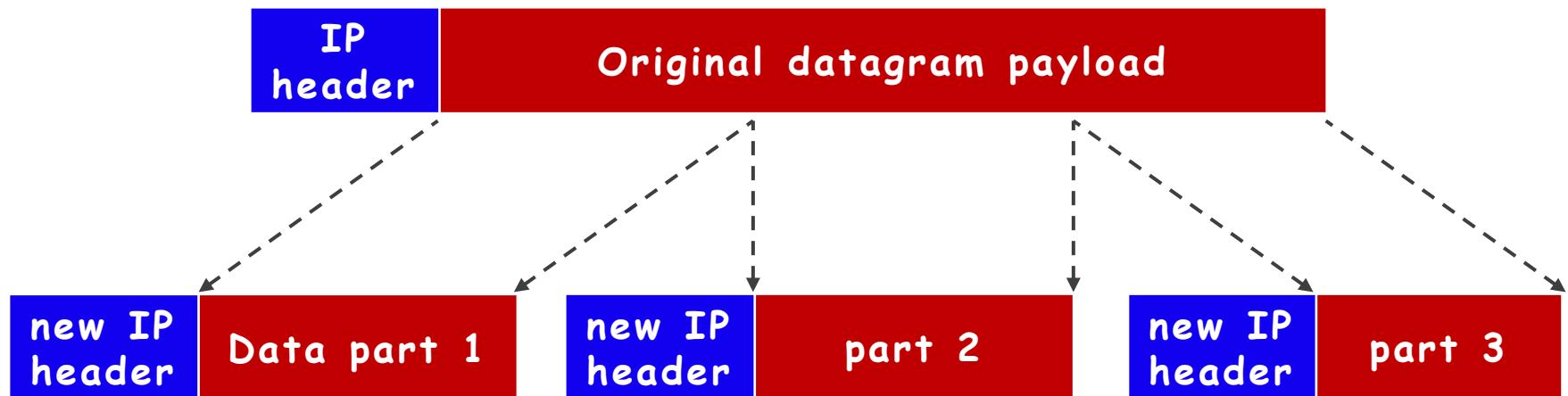


# IP Fragmentation & Reassembly

- ❖ 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 |       |        |        |

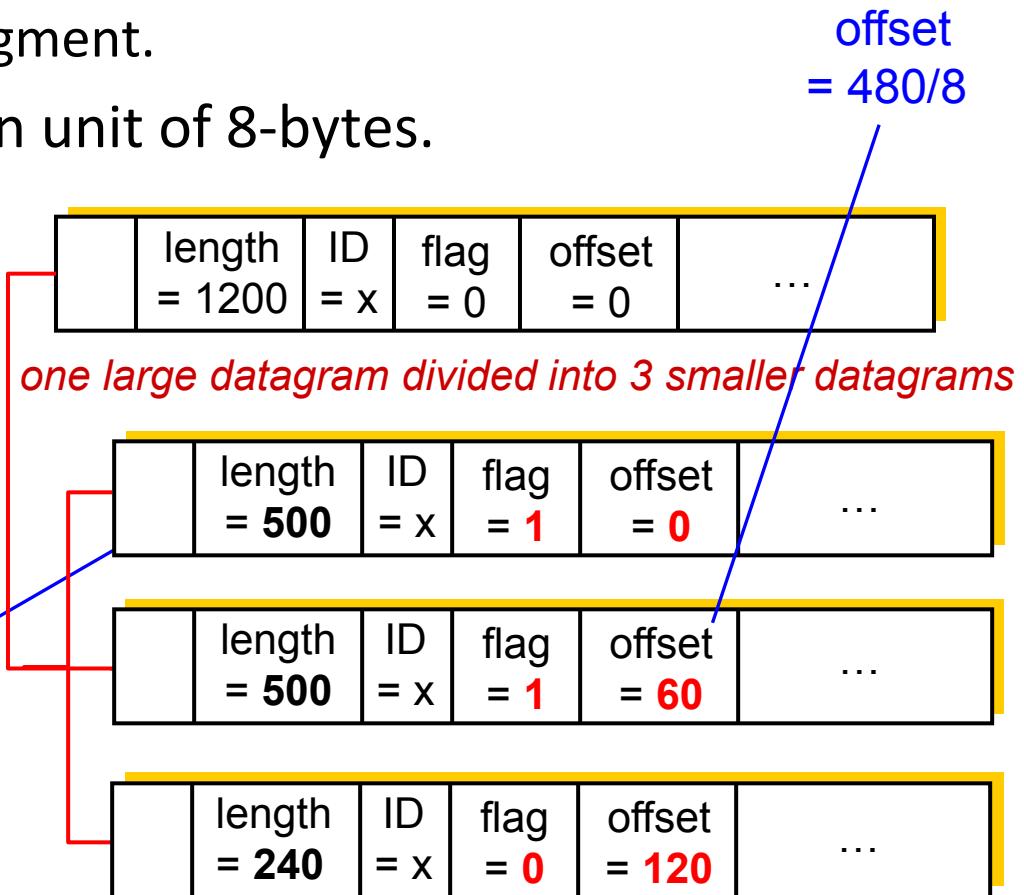
- ❖ Flag (frag flag) is set to
  - 1 if there is next fragment from the same segment.
  - 0 if this is the last fragment.

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

- ❖ Example

- 1200 byte IP datagram
- MTU = 500 bytes

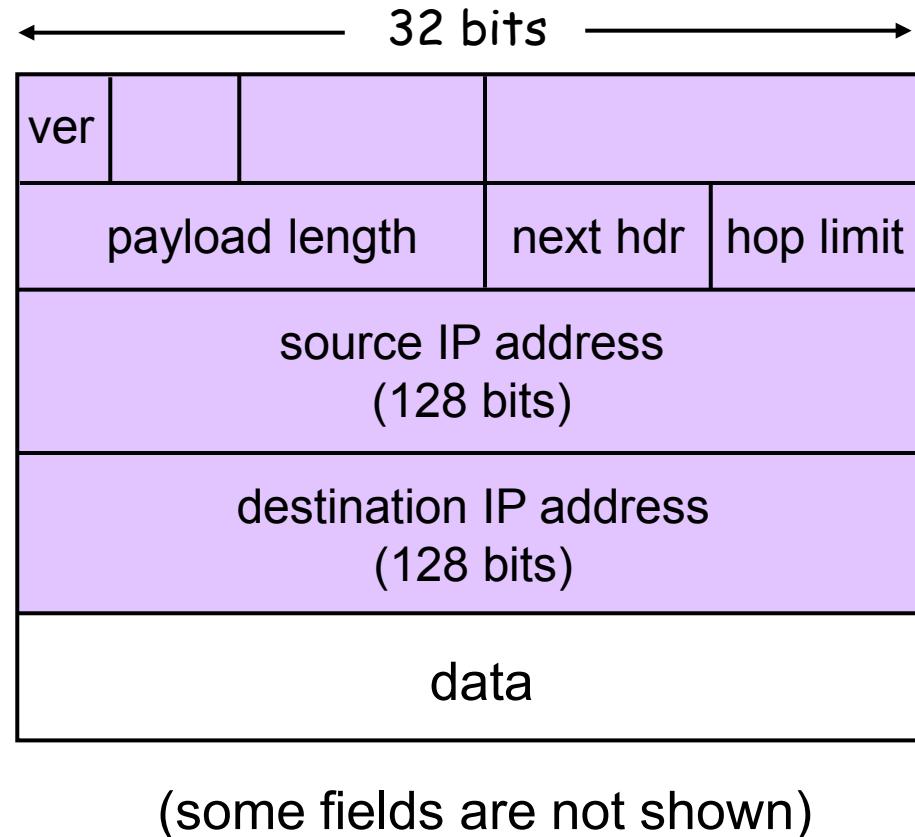
carry 480 bytes of data



# IPv6

Non-examinable

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



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

# Lecture 7: Roadmap

## 4.1 Introduction

## 4.4 IP: Internet Protocol

- 4.4.1 Datagram Format
- 4.4.2 IPv4 Addressing
- 4.4.3 ICMP

## 4.5 Routing Algorithms

# 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

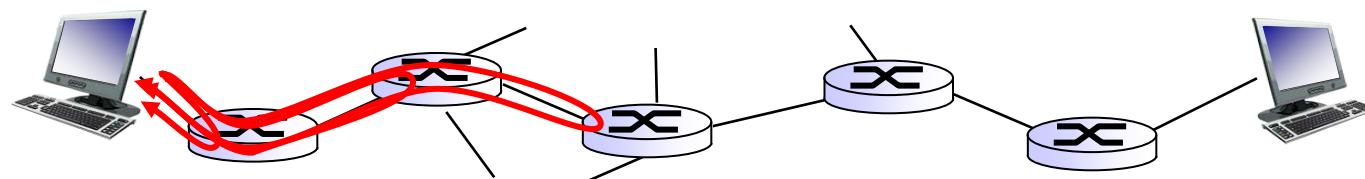
- ❖ ICMP header: Type + Code + Checksum + others.

| 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)

# *ping and traceroute*

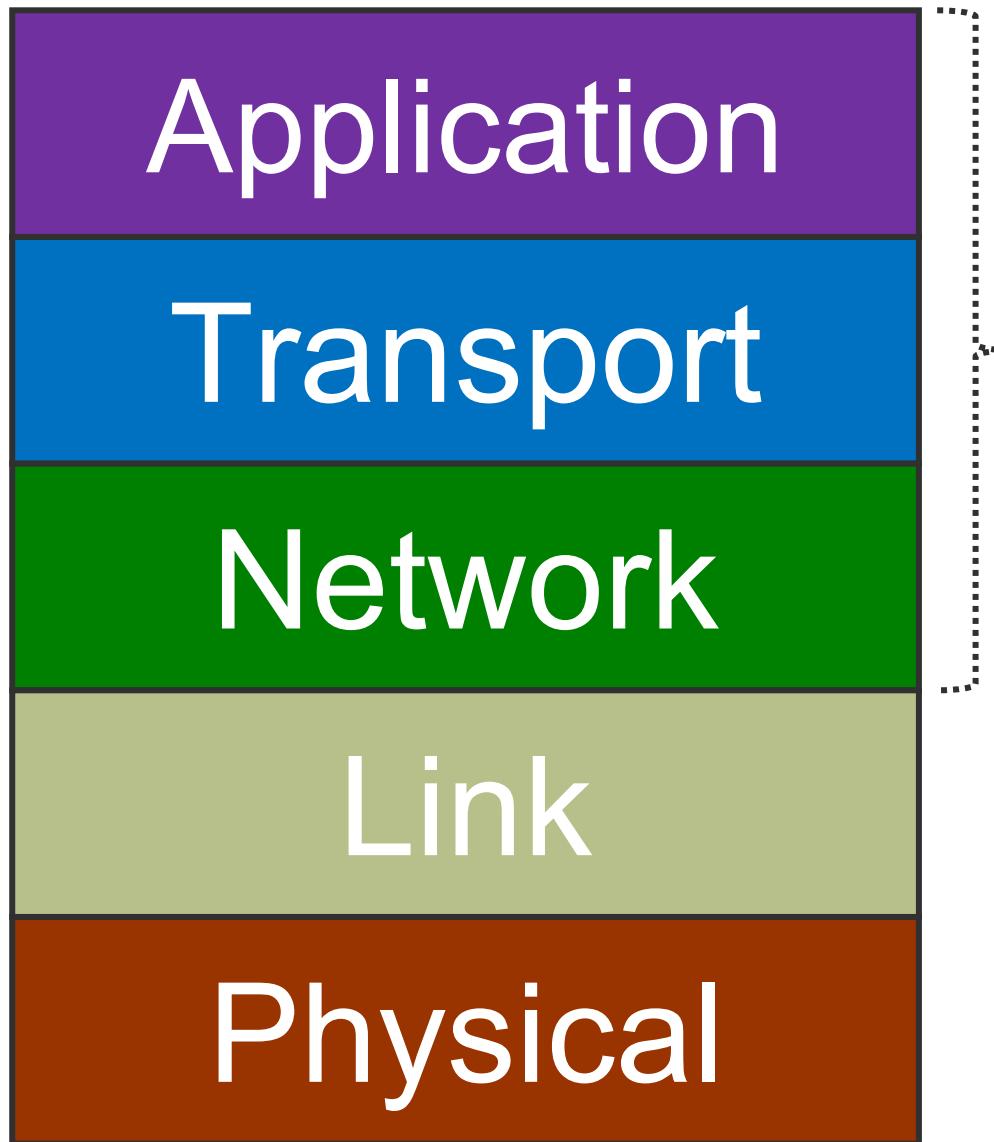
- ❖ The command **ping** sees if a remote host will respond to us – do we have a connection?
- ❖ The command **traceroute** sends a series of small packets across a network, and attempts to display the route (or path) that the messages would take to get to a remote host.



# Lecture 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.

# An Awesome Introduction to Computer Networks



Security  
perspective

# Lecture 8: Network Security

*After this class, you are expected to understand:*

- ❖ how *symmetric key* cryptography and *public key* cryptography can be used to ensure **message confidentiality**.
- ❖ how *message authentication code* ensures **message integrity**.
- ❖ how *digital signature* ensure **message authenticity**.

# Lecture 8: Roadmap

8.1 What is Network Security?

8.2 Principles of Cryptography

8.3 Message Integrity and Digital Signatures

8.6 Securing TCP Connections: SSL

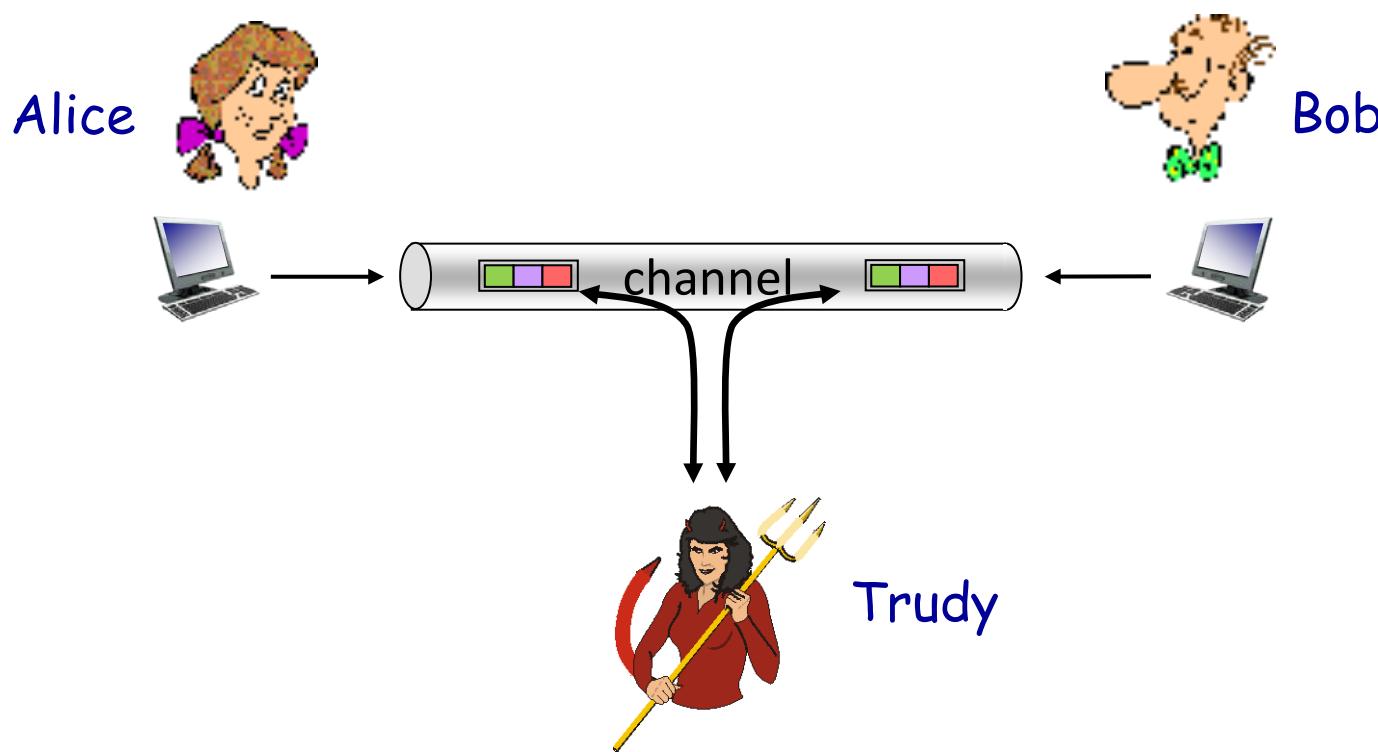
8.7 Network Layer Security: IPsec

Non-examinable

Kurose Textbook, Chapter 8  
(Some slides are taken from the book)

# Friends and Enemies: Alice, Bob, Trudy

- ❖ Alice and Bob (lovers!) want to communicate “secretly”.
- ❖ Trudy (intruder) wants to interfere.



# What Can Bad Guy Trudy Do?



Trudy may:

- intercept messages of Alice and Bob (*eavesdrop*).
  - Need to ensure **message confidentiality**.
- modify messages between Alice and Bob.
  - Need to ensure **message integrity**.
- forge messages and insert into communication.
  - Need to ensure **message authenticity**.
- attack the communication channel between Alice and Bob (e.g. denial-of-service attack).
  - Need to ensure **service availability** (not covered).

# Network Security: Algorithms

- ❖ We will not discuss any security algorithms in great details.
  
- ❖ Interested students may read chapter 8 of the textbook or take security courses offered by SoC, e.g.
  - **CS2107** Introduction to Information Security
  - **CS3235** Computer Security
  - **CS4236** Cryptography Theory and Practice
  - **CS5321** Network Security

# Lecture 8: Roadmap

8.1 What is Network Security?

8.2 Principles of Cryptography

- 8.2.1 Symmetric Key Cryptography
- 8.2.2 Public Key Encryption

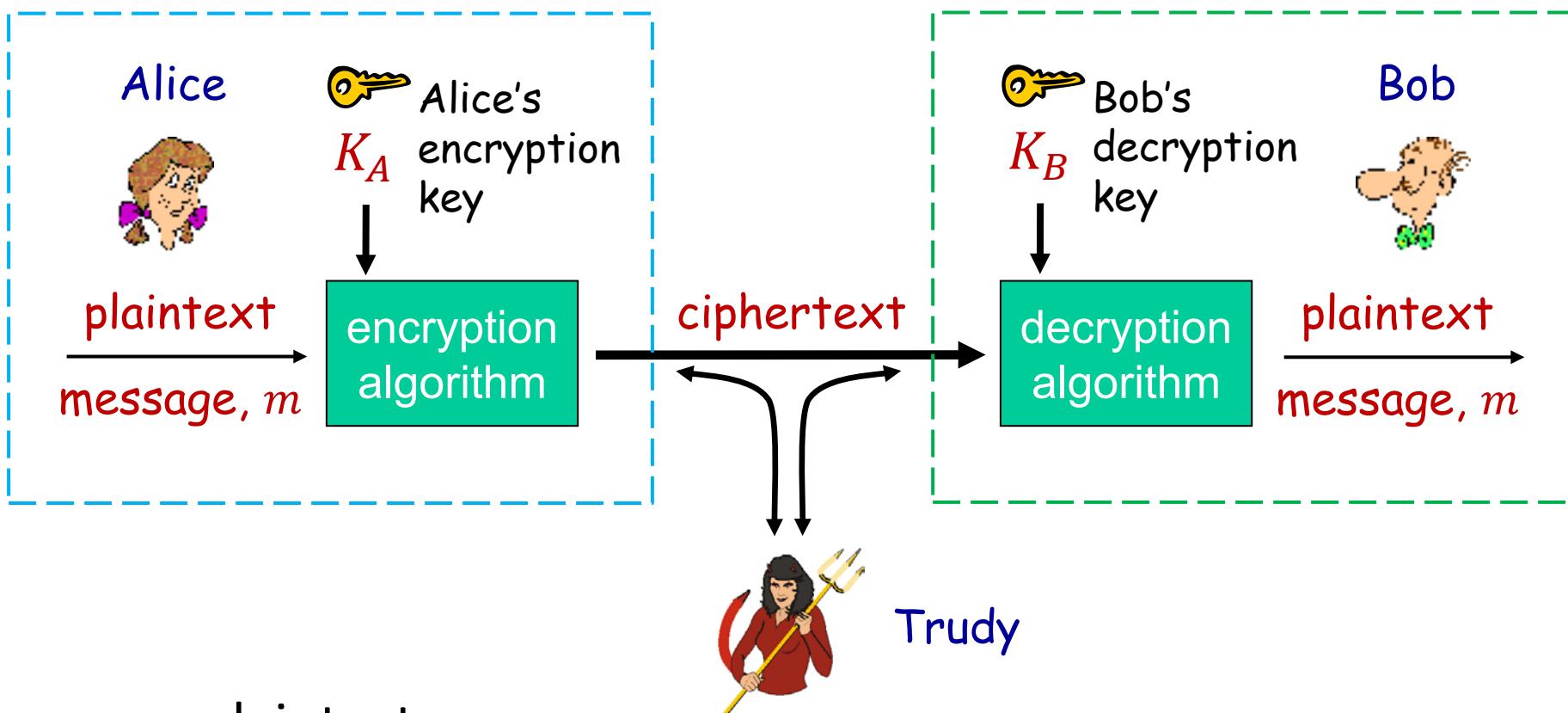
8.3 Message Integrity and Digital Signatures

8.6 Securing TCP Connections: SSL

8.7 Network Layer Security: IPsec

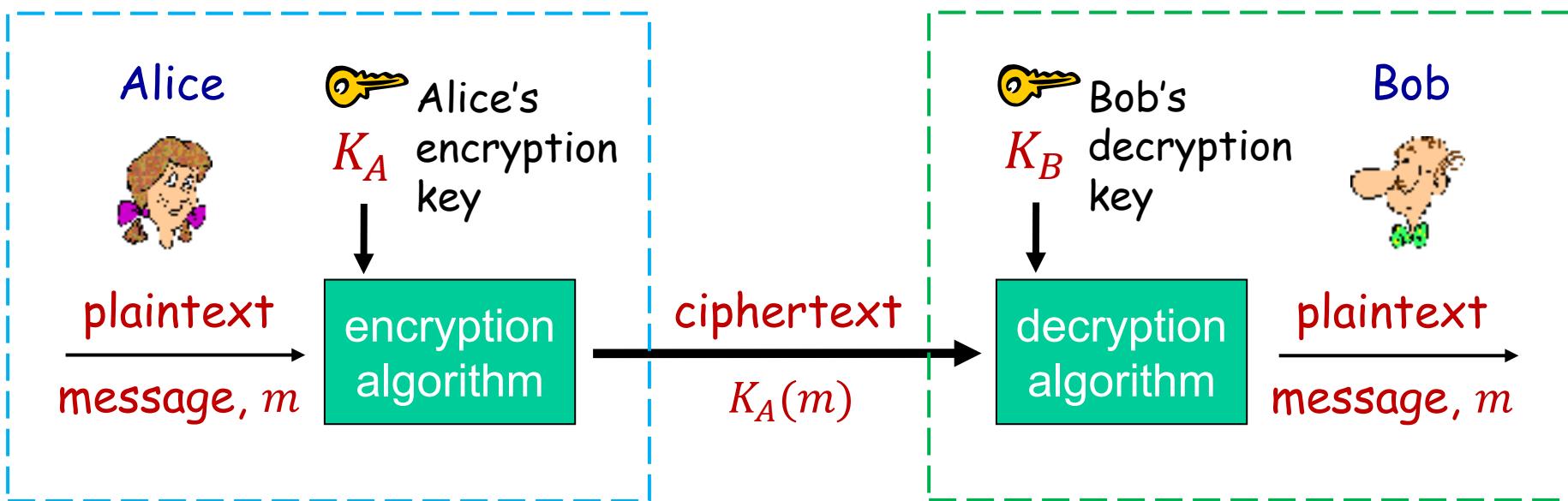
Non-examinable

# The Language of Cryptography



- ❖  $m$  plaintext message
- ❖  $K_A(m)$  ciphertext, encrypted with key  $K_A$
- ❖  $K_B(K_A(m)) = m$

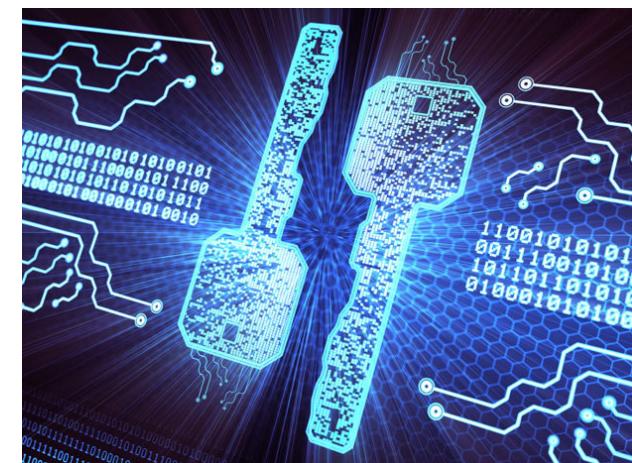
# The Language of Cryptography



- ❖ Given ciphertext  $K_A(m)$ , it should be computationally hard to find plaintext  $m$  without knowing decryption key  $K_B$ .
- ❖ We will skip the mathematical details on how to derive  $K_A$  and  $K_B$ .

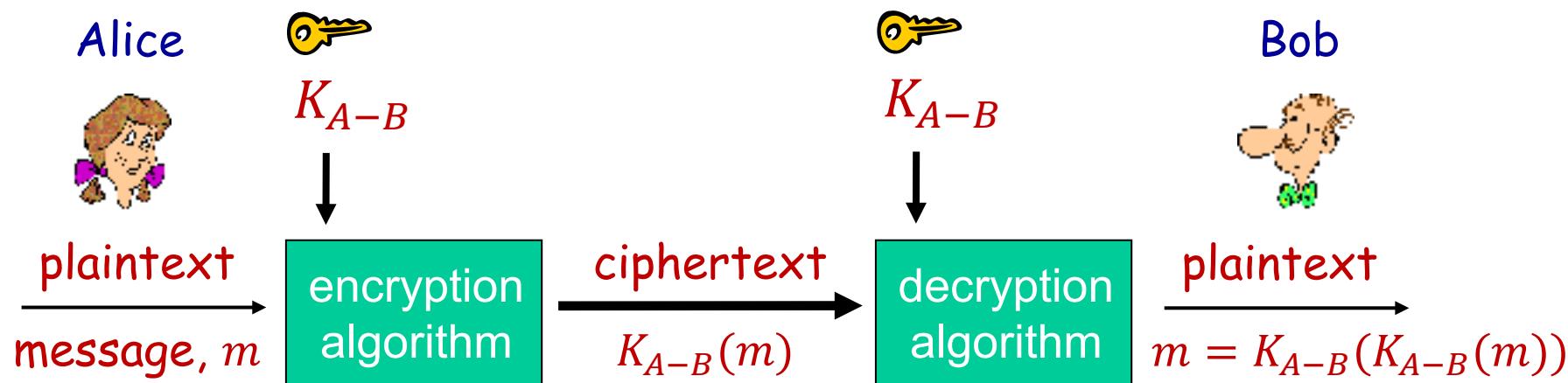
# Types of Cryptography

- ❖ The purpose of cryptography is to make it difficult for an unauthorized third party to understand private communication between two parties.
- ❖ Cryptography often uses **keys**:
  - Algorithms are known to everyone
  - Only “keys” are secret
- ❖ **Symmetric key** cryptography
  - Involves the use of one key
- ❖ **Public key** cryptography
  - Involves the use of a pair of keys



Source: IEEE Spectrum

# Symmetric Key Cryptography



- ❖ **Symmetric key crypto:** Bob and Alice share and use the same (symmetric) key:  $K_{A-B}$ 
  - Popular algorithms: **DES** (Data Encryption Standard), **AES** (Advanced Encryption Standard)

# Example Encryption Scheme

- ❖ **Mono-alphabetic cipher:** substituting one letter for another.

|            |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| plaintext  | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| ciphertext | m | n | b | v | c | x | z | a | s | d | f | g | h | j | k | l | p | o | i | u | y | t | r | e | w | q |

E.g: **Plaintext:** bob, i love you. alice

**ciphertext:** nkn, s gktc wky. mgsbc

🔑 **Encryption key:** mapping from a set of 26 letters to another set of 26 letters

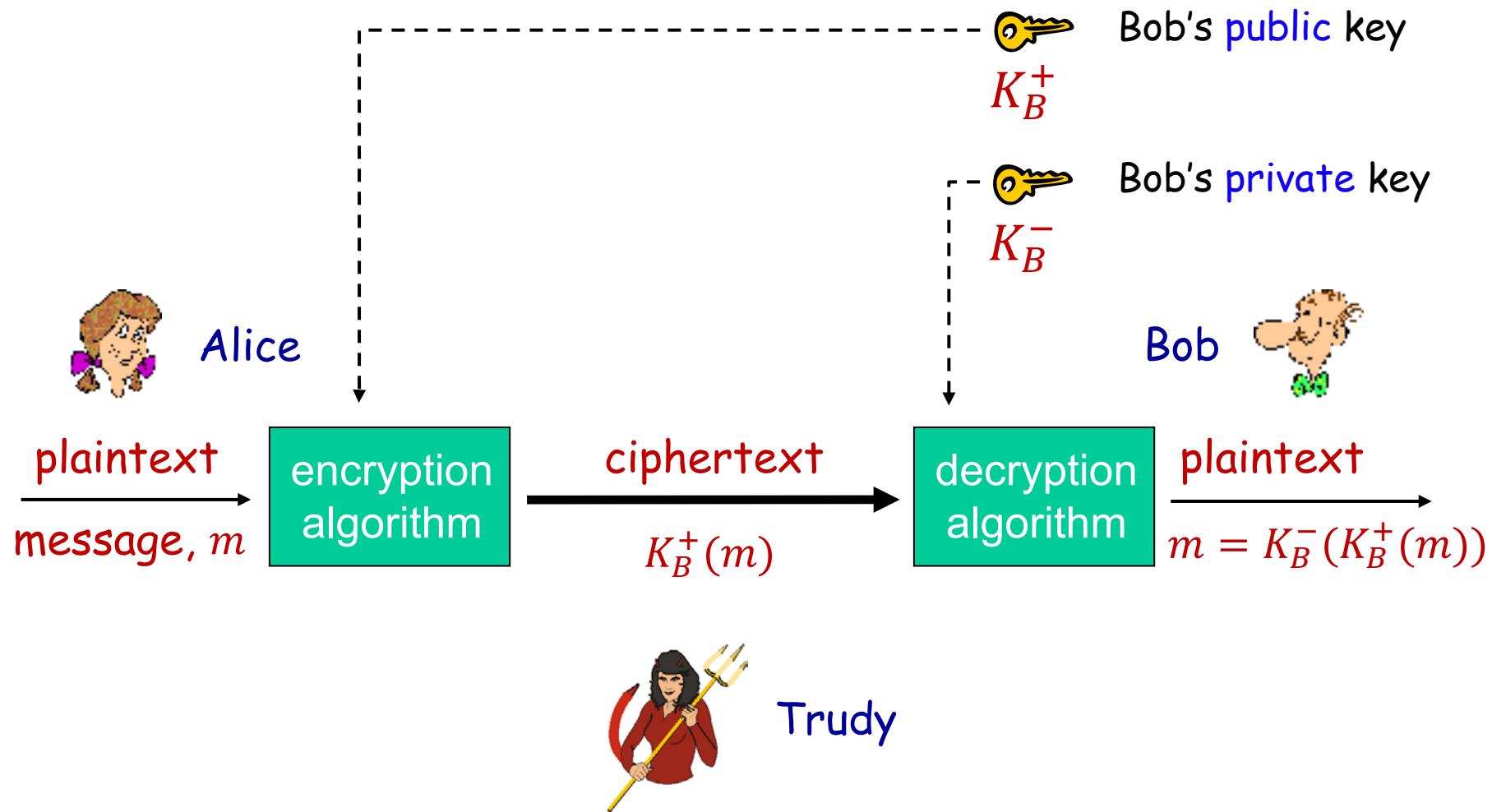
# Public Key Cryptography

- ❖ Symmetric key crypto issues:
  - Require sender and receiver to **share a secret key**.
  - Use the same secret key to encrypt and decrypt data.
  - **Question:** how to agree on a key in the first place?

- ❖ Public key crypto:
  - Sender and receiver do not share secret key.
  - Use a pair of keys. One for encryption and the other for decryption.
  - **Public encryption key:** known to the world.
  - **Private decryption key:** known only to receiver.



# Public Key Cryptography



# Public Key Encryption Algorithms

- ❖ Key points of public key encryption:

① Need to find a pair of public/private keys such that

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

② Given public key  $K_B^+$ , it should be very difficult to find private key  $K_B^-$ .

- ❖ Most popular algorithm: RSA (Rivest, Shamir, Adelson algorithm)

# Public Key: RSA Algorithm

- ❖ In RSA
  - The **public key** is the product of two very large primes.
  - The **private key** is derived from these two large primes.
- ❖ The security of RSA relies on the difficulty of factoring a large composite number.
  - It would be too slow to “guess” the two large primes, given the current state of the art of number theory.
- ❖ We will skip the mathematical details.

# An Important Property of RSA

- ❖ The following property of RSA will be *very* useful for our discussion later:

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

use **public** key first,  
followed by  
**private** key

use **private** key  
first, followed by  
**public** key

*Result is the same!*

# RSA in Practice: Session Key

- ❖ RSA (public key encryption) is computationally intensive (but doesn't require key sharing).
- ❖ DES (symmetric key encryption) is at least 100 times faster than RSA.
- ❖ Question: how to take advantage of both?
  - use public key crypto to establish secure connection, then second key – symmetric 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.
- ❖ No need to remember  $K_S$ , it's valid for one session only.

# Lecture 8: Roadmap

8.1 What is Network Security?

8.2 Principles of Cryptography

8.3 Message Integrity and Digital Signatures

- 8.3.1 Cryptographic Hash Functions
- 8.3.2 Message Authentication Code
- 8.3.3 Digital Signatures

8.6 Securing TCP Connections: SSL

8.7 Network Layer Security: IPsec

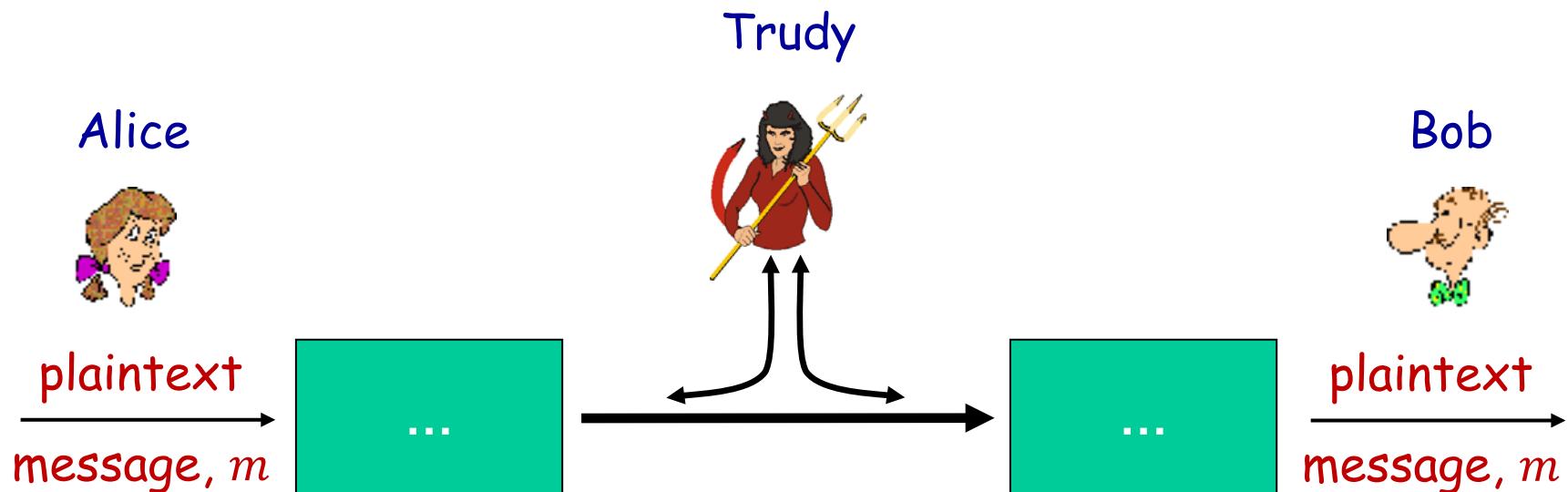
Non-examinable

# Message Integrity and Authentication

- ❖ We have seen how encryption can be used to provide confidentiality to two communicating entities.
- ❖ On the other hand, we often need to
  - ensure message has not been modified during transmission.
  - verify the creator of a message.

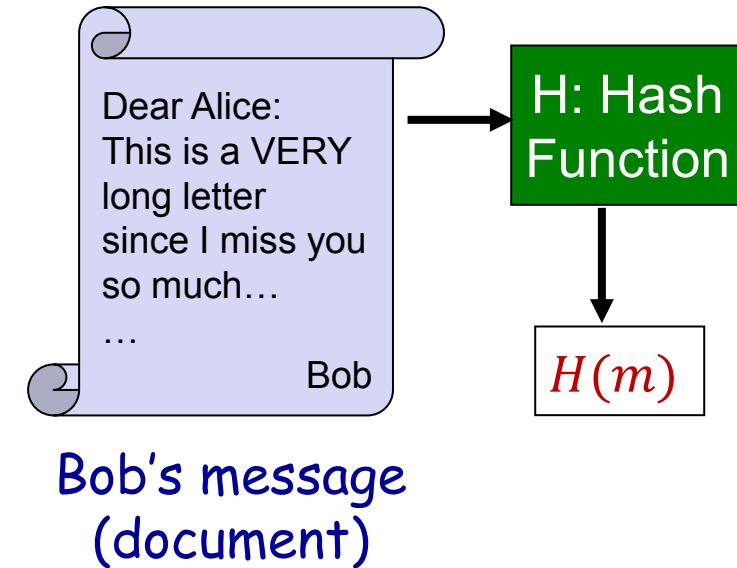
# Message Integrity and Authentication

- ❖ We are going to study the following two topics:
  - ① Ensuring message integrity with **message authentication code (MAC)**.
  - ② Verifying message source with **digital signature**.
- ❖ The basics of both is cryptographic hash function.



# Cryptographic Hash Functions

- ❖ A hash function takes an input,  $m$ , and generates a fixed size string  $H(m)$  known as message digest (hash or finger print).



- ❖ Popular algorithms: **MD5** (Message Digest) and **SHA-1** (Secure Hash Algorithm)
  - Example usage: both have been widely used to ensure a file downloaded from server has arrived intact.

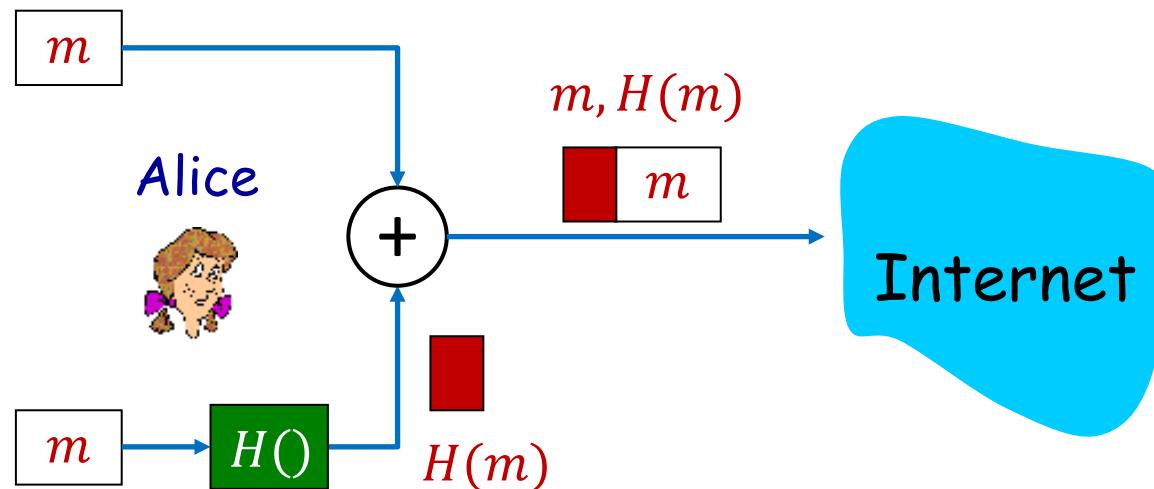
# Cryptographic Hash Functions

- ❖ Cryptographic hash functions are one-way functions:
  - It is computationally infeasible to find two different messages  $m$  and  $m'$  such that  $H(m) = H(m')$ .
  - Therefore impossible for Trudy to forge another message  $m'$  with the same message digest as  $m$ .
- ❖ When using cryptographic hash functions,
  - A small change in the message (say, by eavesdropper) will create a significant change in the message digest.

# Example Usage (1/2)

*For Alice:*

1. Alice creates message  $m$  and calculates the hash  $H(m)$ .
2. Alice then appends  $H(m)$  to the message  $m$ , creating an extended message  $(m, H(m))$ , and sends the extended message to Bob.



# Example Usage (1/2)

*For Bob:*

1. Bob receives an extended message  $(m', h')$  .
  2. Bob calculates  $H(m')$ . If  $H(m') = h'$ , Bob concludes that everything is fine.
- 
- ❖ **Q:** Can Bob be sure the source of message is Alice?
    - **No.** Because Trudy can create a bogus message  $m'$  in which she says she is Alice, calculates  $H(m')$ , and sends Bob  $(m', H(m'))$ .

# Message Authentication Code

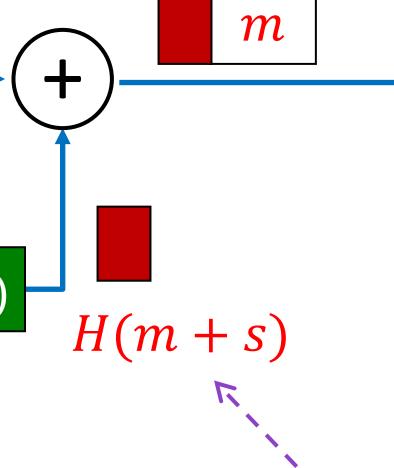
- ❖ If a **key**  is used as part of the message digest generation, such an algorithm is said to generate a **message authentication code** (MAC).
  - Can detect accidental and intentional changes to a message.
  - Can affirm to the receiver, the message's origin.
- ❖ Java supports the following standard MAC algorithms:
  - HmacMD5, HmacSHA1, HmacSHA256

# Message Authentication Code

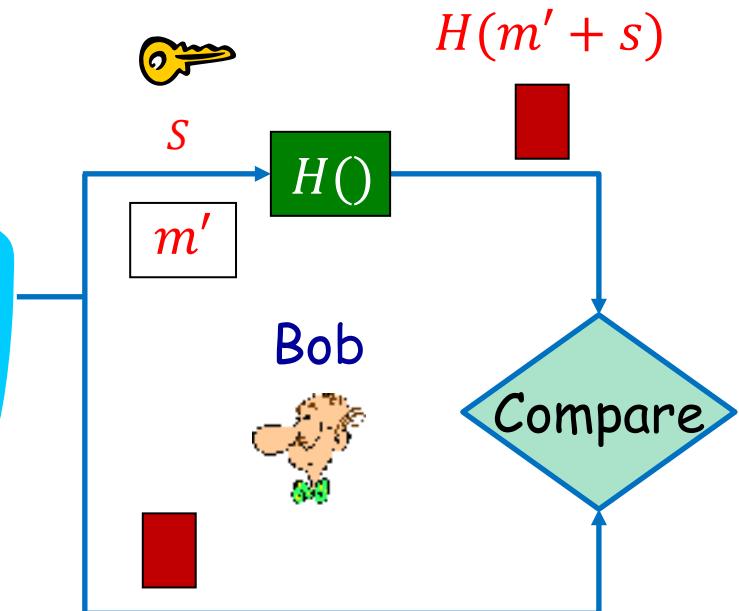
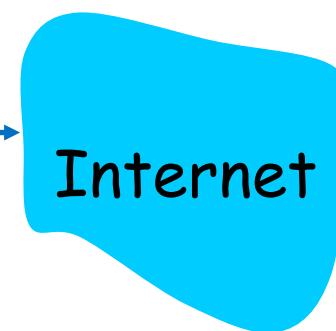
Alice



$m$



Message authentication  
code (MAC)



$s$ : shared secret known  
as authentication key

- ❖ MAC proves message integrity to Bob (origin of message is Alice and message is not corrupted).

# Digital Signature

Sender (Alice) signs a document digitally (analogous to hand-written signatures).

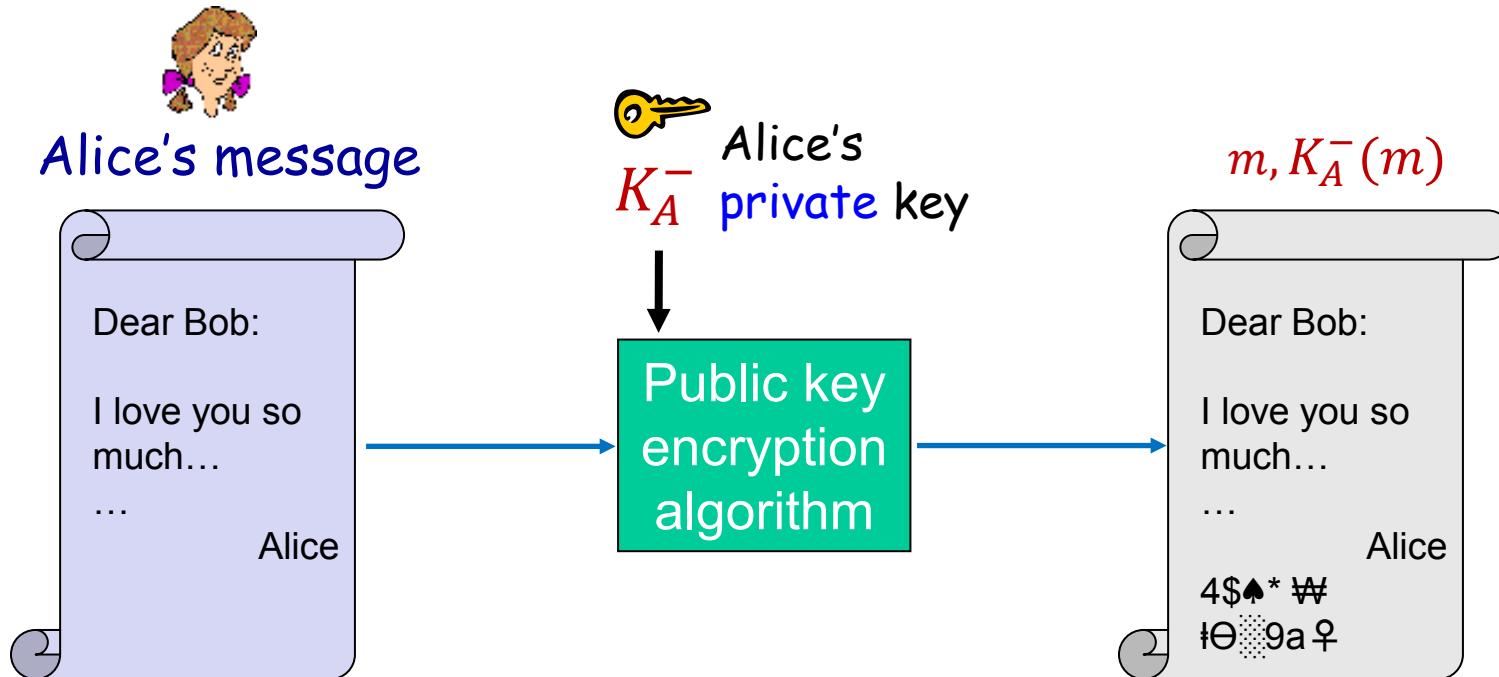
- ❖ *verifiable*: recipient (Bob) can verify that Alice, and no one else, has signed this document.
- ❖ *non-repudiation*: If Bob shows this document and digital signature to a third party (e.g. court), the third party is confident that this document is indeed signed by Alice (but no one else including Bob).

# Digital Signature vs. MAC

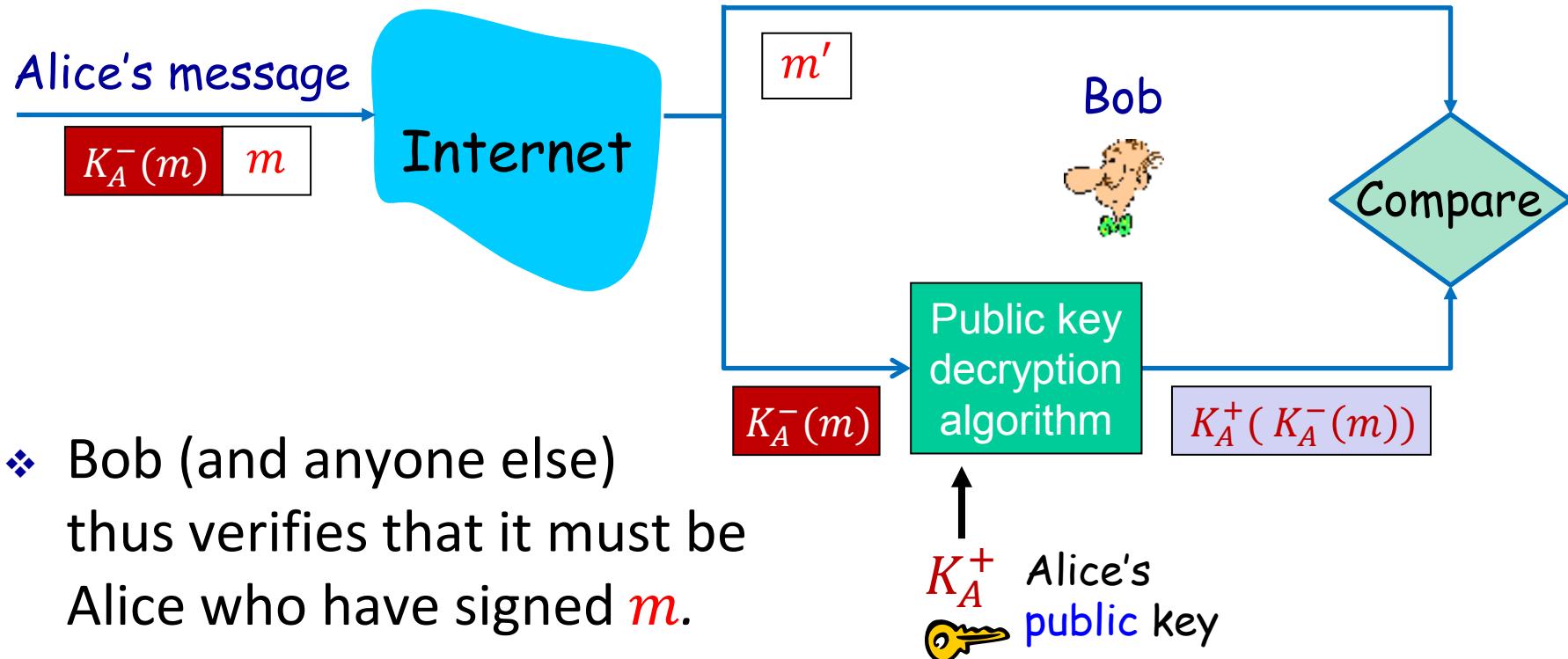
- ❖ Message authentication code (MAC) uses an authentication key **shared** between sender (Alice) and receiver (Bob).
  - Either Alice or Bob can produce the same MAC on a document, using the shared key.
  - Cannot prove to a third party MAC is produced by Alice or Bob.
- ❖ When Alice signs document digitally, she must put something on the doc that is **unique to her**.
  - her private key 

# Digital Signature Example (1/2)

- ❖ Alice signs  $m$  by encrypting it with her private key  $K_A^-$ , creating a “signed” message,  $K_A^-(m)$ .
  - Send both  $m$  and  $K_A^-(m)$  to Bob.

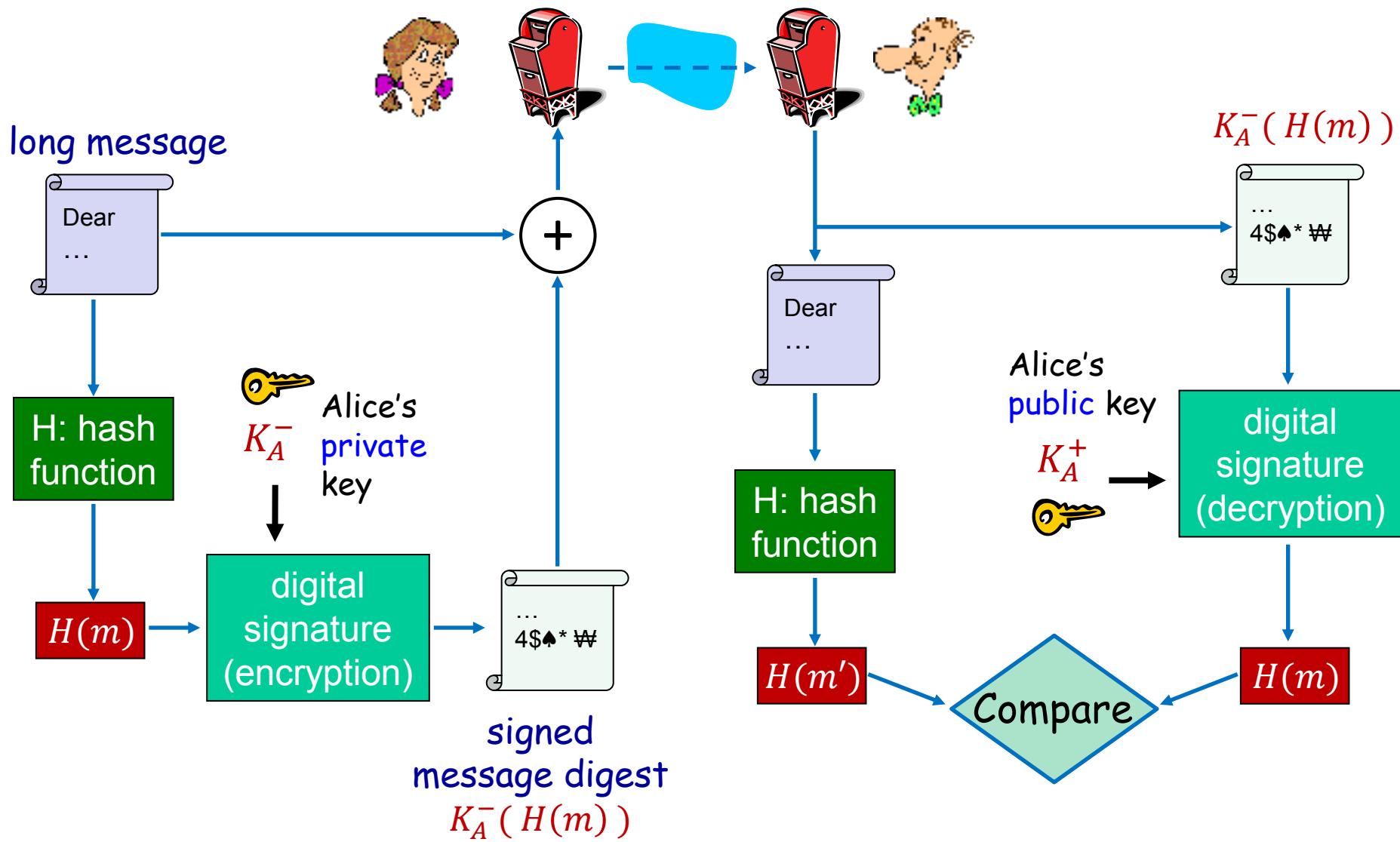


# Digital Signature Example (2/2)



- ❖ Bob (and anyone else) thus verifies that it must be Alice who have signed  $m$ .
- ❖ Just one minor point:
  - Public key encryption is very slow.
  - Efficiency is a concern if  $m$  is long.

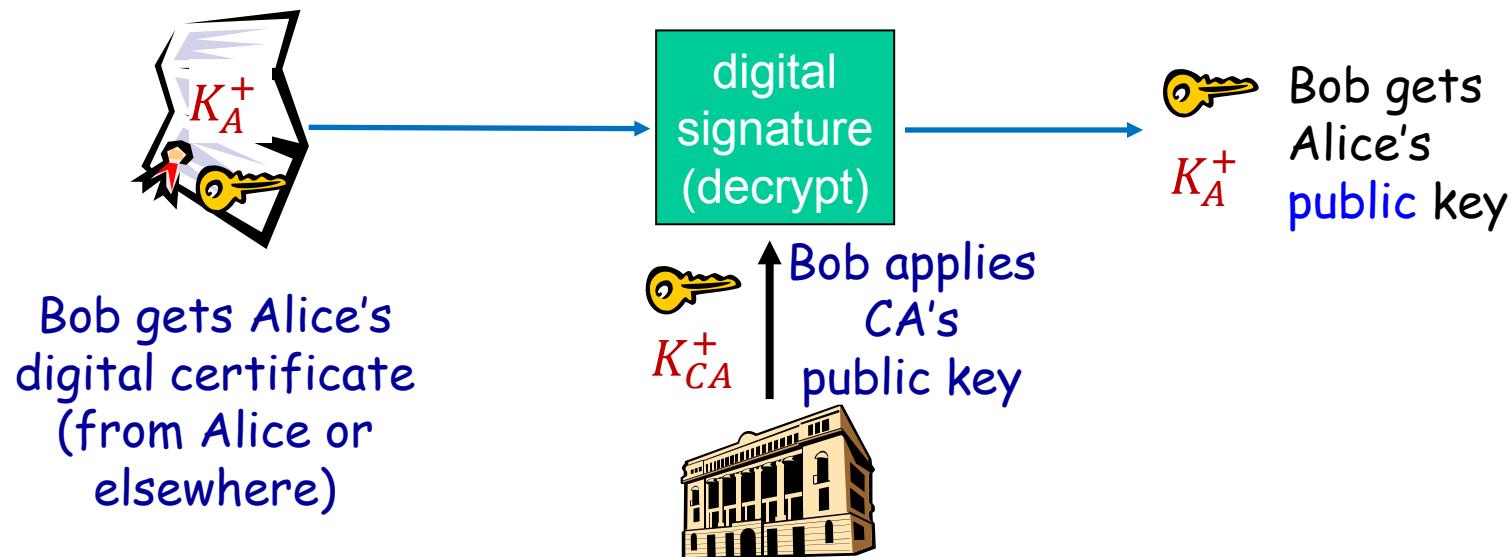
# Digital Signature = Signed Message Digest



# Digital Certificate

Non-examinable

- ❖ Bob may wonder if the public key he uses is indeed Alice's.
- ❖ Certificate authority (CA) is an entity that issues digital certificates.
  - A digital certificate certifies the ownership of a public key by the named subject of the certificate.



# Lecture 8: Roadmap

8.1 What is Network Security?

8.2 Principles of Cryptography

8.3 Message Integrity and Digital Signatures

8.6 Securing TCP Connections: SSL

8.7 Network Layer Security: IPsec

Non-examinable

# SSL: Secure Sockets Layer

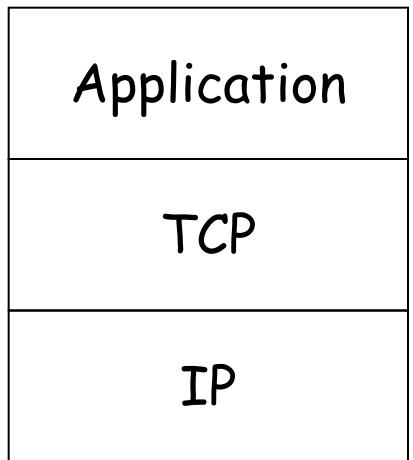
SSL is a widely deployed security protocol.

- ❖ Applicable to TCP applications
- ❖ A variation is **TLS (Transport Layer Security)** defined in RFC 2246.
- ❖ Supported by almost all modern browsers and web servers.
- ❖ For example, **https = http + SSL/TLS**
  - adding security capabilities of SSL/TLS to standard HTTP communications.

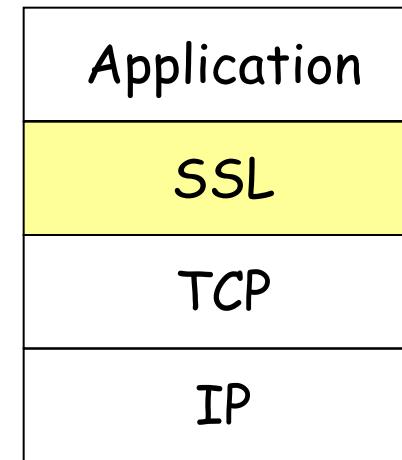
Common SSL symmetric ciphers

- DES - Data Encryption Standard: block
  - 3DES - Triple strength: block
  - RC2 - Rivest Cipher 2: block
  - RC4 - Rivest Cipher 4: stream
- SSL public key encryption
- RSA

# SSL: Secure Sockets Layer



Normal Application

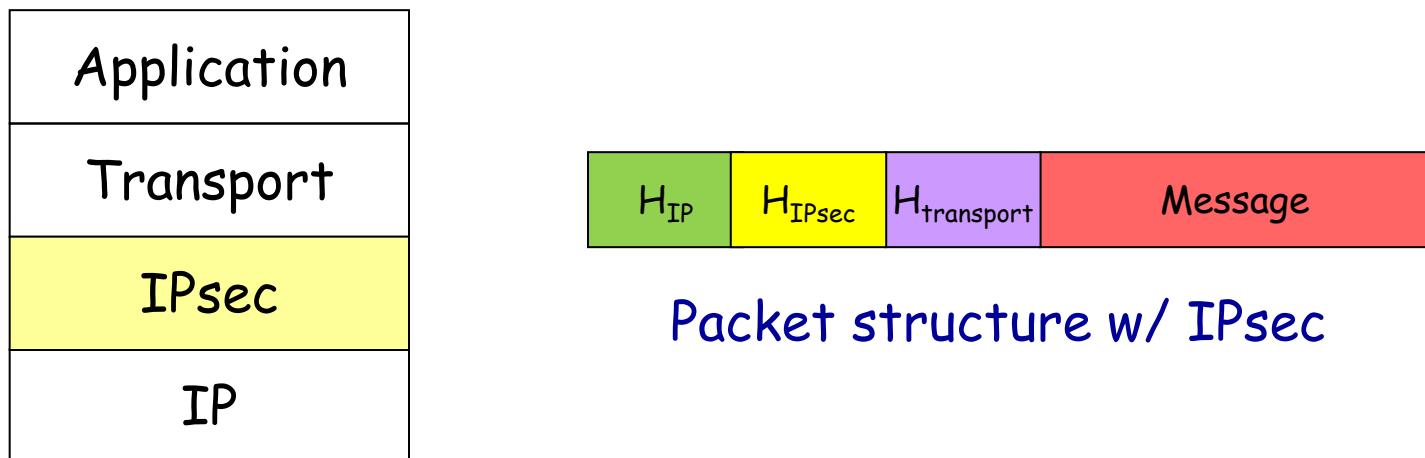


Application with SSL

- ❖ SSL provides application programming interface (API) to applications.
  - Java SSL libraries/classes readily available.

# Internet Protocol Security (IPsec)

- ❖ IPsec is a suite of protocols that secure communications by authenticating and encrypting each IP packet of a communication session.



- ❖ Both SSL and IPsec can be used to build VPN.
  - SoC and NUS WebVPN run over SSL.

# Lecture 8: Summary

basic techniques ....

- data confidentiality (symmetric and public keys)
- message digest
- message authentication code
- digital signature

.... used in many different security scenarios

- https
- secure transport (SSL)
- IPsec
- 802.11 WEP



# An *Awesome* Introduction to Computer Networks

# Network Security

## ❖ Basic concepts:

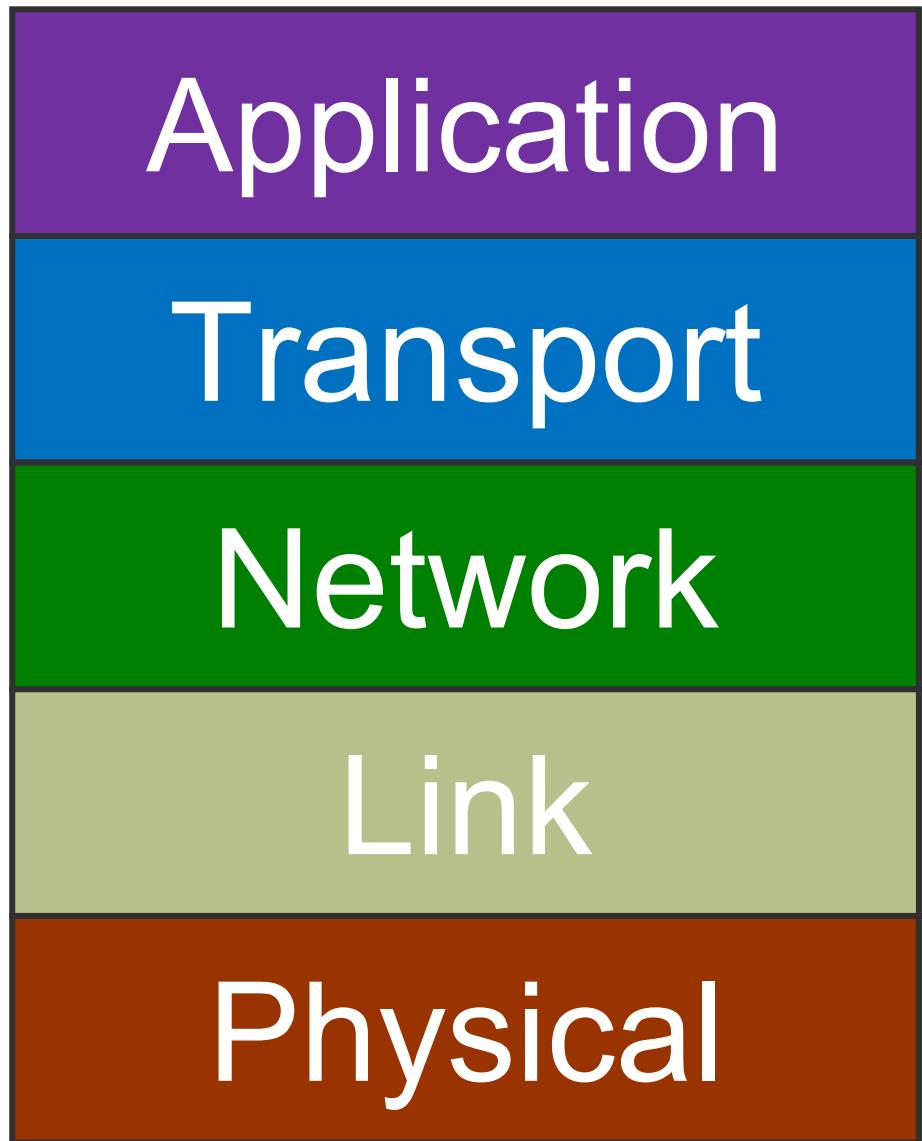
- Message confidentiality (symmetric and public key)
- Message integrity and authenticity (see table below)

| Security Goal                | Hash function | MAC           | Digital signature |
|------------------------------|---------------|---------------|-------------------|
| Accidental damage to message | Yes           | Yes           | Yes               |
| Message integrity            | No            | Yes           | Yes               |
| Message authenticity         | No            | No            | Yes               |
| Kind of keys used            | None          | Symmetric key | Public key        |

# Lecture 9: 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 schemes work.
- ❖ different methods for accessing shared medium.
- ❖ how ALOHA, Slotted ALOHA, CSMA and CSMA/CD protocols work.



# Lecture 9: Roadmap

## 5.1 Introduction to the Link Layer

## 5.2 Error Detection and Correction

## 5.3 Multiple Access Links and Protocols

- 5.3.1 Channel Partitioning Protocols
- 5.3.2 Random Access Protocols
- 5.3.3 Taking-Turns Protocols

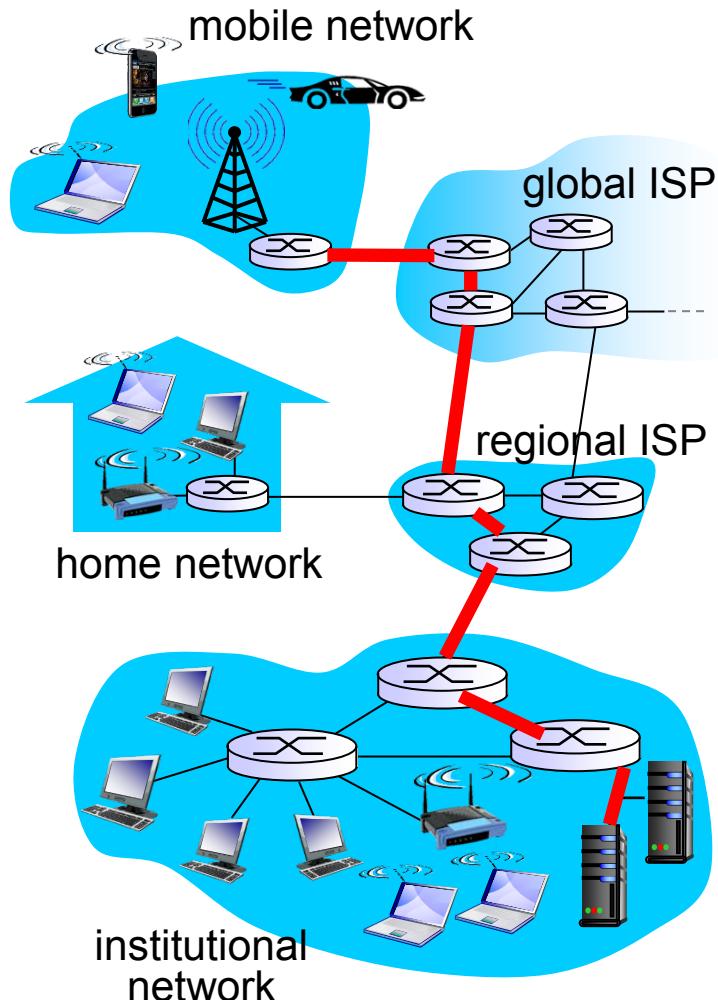
## 5.4 Switched Local Area Networks }

To discuss  
next week

Kurose Textbook, Chapter 5  
(Some slides are taken from the book)

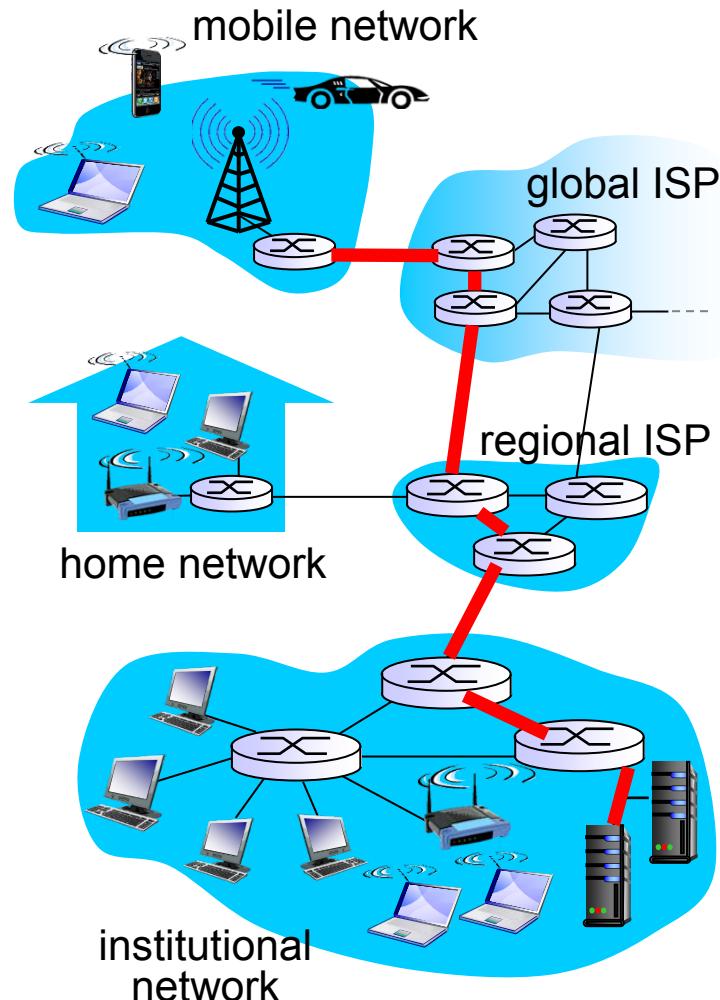
# Link Layer: Introduction (1/2)

- ❖ **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.



# 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.



# Possible Link Layer Services (2/2)

## ❖ Reliable delivery

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

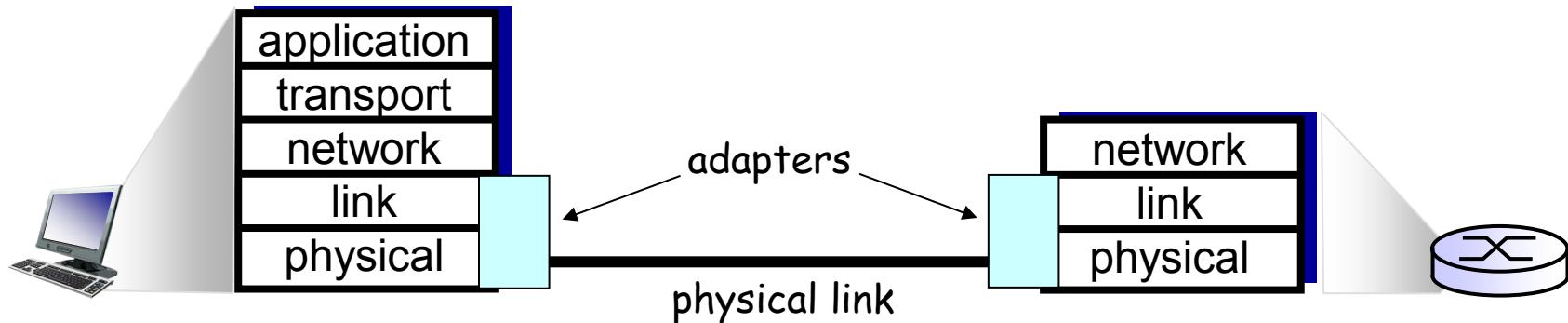
## ❖ Error detection

- 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.



# Lecture 9: Roadmap

5.1 Introduction to the Link Layer

5.2 Error Detection and Correction

5.3 Multiple Access Links and Protocols

- 5.3.1 Channel Partitioning Protocols
- 5.3.2 Random Access Protocols
- 5.3.3 Taking-Turns Protocols

5.4 Switched Local Area Networks

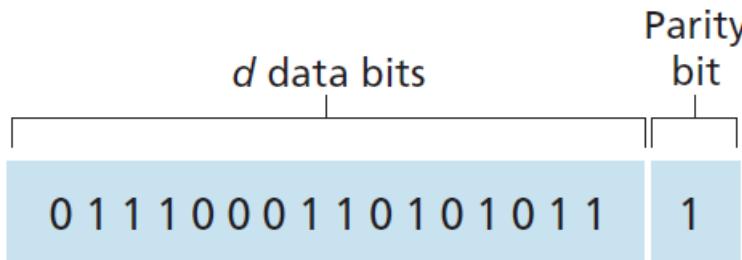
# Error Detection and Correction

- ❖ 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).

# Parity Checking

## Single bit parity

- ❖ can detect single bit errors in data.



## Two-dimensional bit parity

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

No errors

|       |   |   |   |   |   |
|-------|---|---|---|---|---|
| 1     | 0 | 1 | 0 | 1 | 1 |
| 1     | 1 | 1 | 1 | 0 | 0 |
| 0     | 1 | 1 | 1 | 0 | 1 |
| <hr/> |   |   |   |   |   |
| 0     | 0 | 1 | 0 | 1 | 0 |

Correctable  
single-bit error

|       |   |   |   |   |   |
|-------|---|---|---|---|---|
| 1     | 0 | 1 | 0 | 1 | 1 |
| 1     | 0 | 1 | 1 | 0 | 0 |
| 0     | 1 | 1 | 1 | 0 | 1 |
| <hr/> |   |   |   |   |   |
| 0     | 0 | 1 | 0 | 1 | 0 |

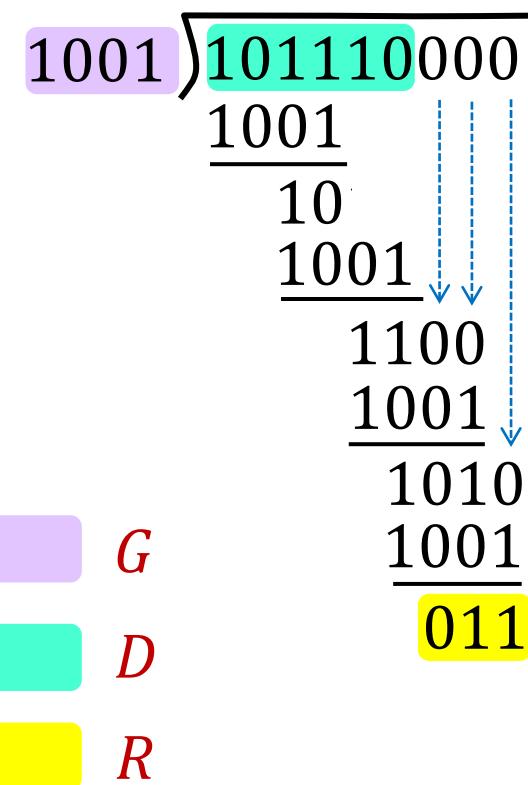
Parity error

Parity error

# 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.

Example:  $r = 3$



# CRC Example (Continued)

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

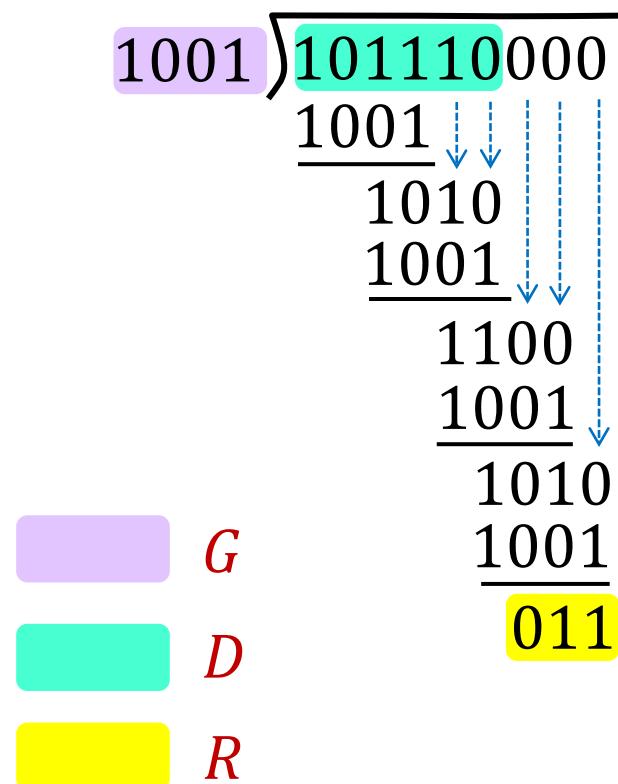
Example:  $r = 3$

- ❖ Sender sends  $(D, R)$

101110011

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

- If non-zero remainder:  
error is detected!



# Lecture 9: Roadmap

5.1 Introduction to the Link Layer

5.2 Error Detection and Correction

5.3 Multiple Access Links and Protocols

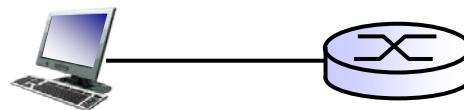
- 5.3.1 Channel Partitioning Protocols
- 5.3.2 Random Access Protocols
- 5.3.3 Taking-Turns Protocols

5.4 Switched Local Area Networks

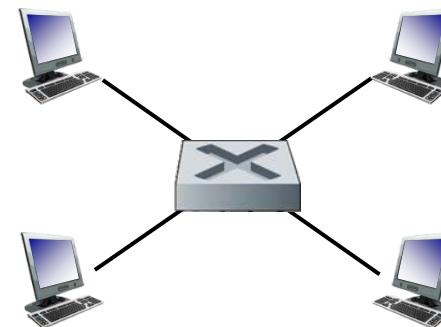
# Two Types of Network Links

## ❖ 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



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

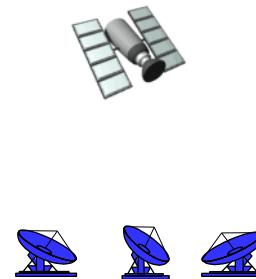
# Two Types of Network Links

## ❖ Type 2: broadcast link (shared medium)

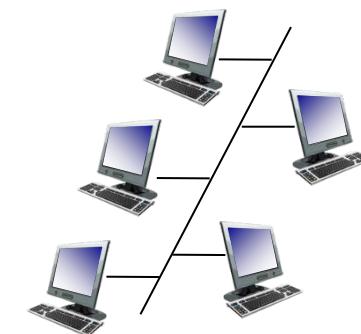
- Multiple nodes connected to a shared broadcast channel.
- When a node transmits a frame, the channel broadcasts the frame and each other node receives a copy.



802.11 Wi-Fi



Satellite



Ethernet with bus topology

# Multiple Access Protocols

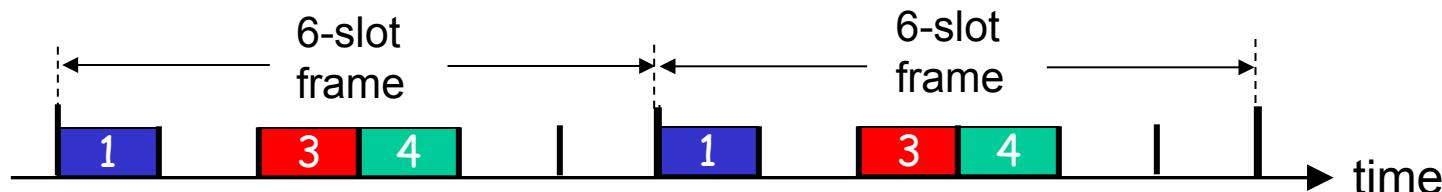
- ❖ In a broadcast channel, if two or more nodes transmit simultaneously
  - Every node receives multiple frames at the same time  
→ frames *collide* at nodes and none would be correctly read.
- ❖ Multiple Access Protocol
  - distributed algorithm that determines how nodes share channel, i.e., when 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 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

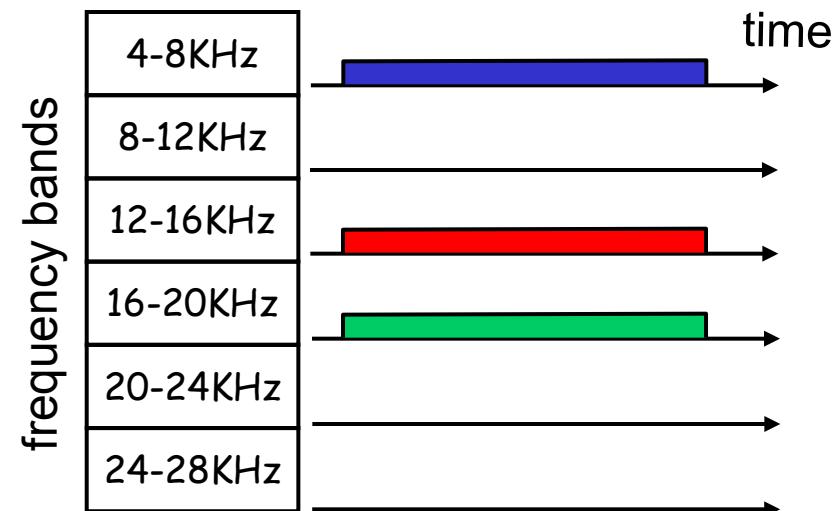
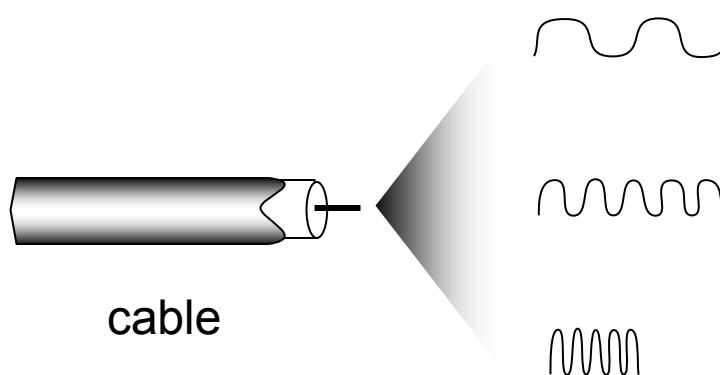
- ❖ **TDMA** (time division multiple access)
  - Access to channel in “rounds”.
  - Each node gets fixed length slot (length = frame transmission time) in each round.
  - Unused slots go idle.
  - Example: 6 nodes sharing a link, 1, 3, 4 have frames, slots 2, 5, 6 are idle.



# 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.



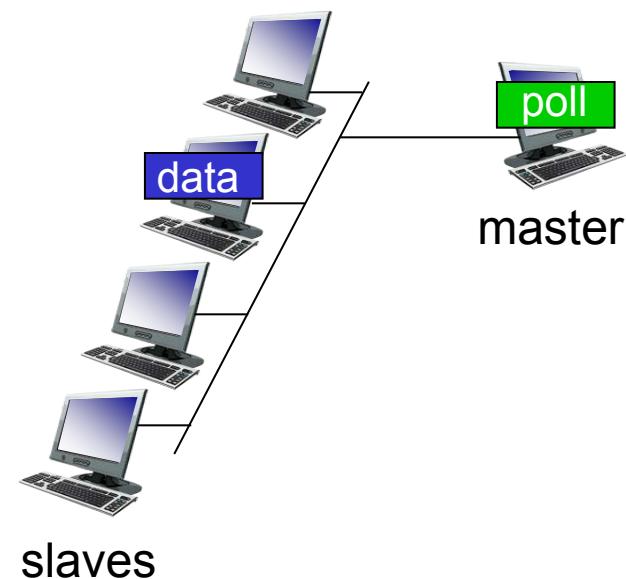
# Multiple Access Protocols

- ❖ Multiple access protocols can be categorized into three broad classes:
  - Channel partitioning
    - divide channel into 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:

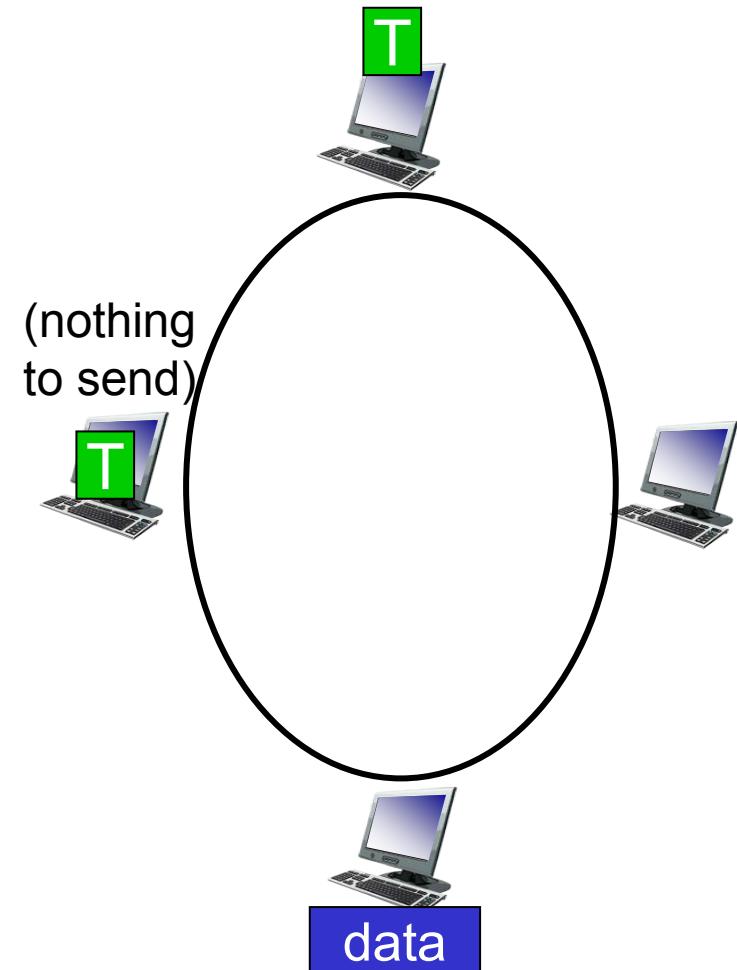
- ❖ master node “invites” slave nodes to transmit in turn.
- ❖ concerns:
  - polling overhead
  - single point of failure (master node)



# “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 skip the mathematical formulas on the efficiency of random access protocols.

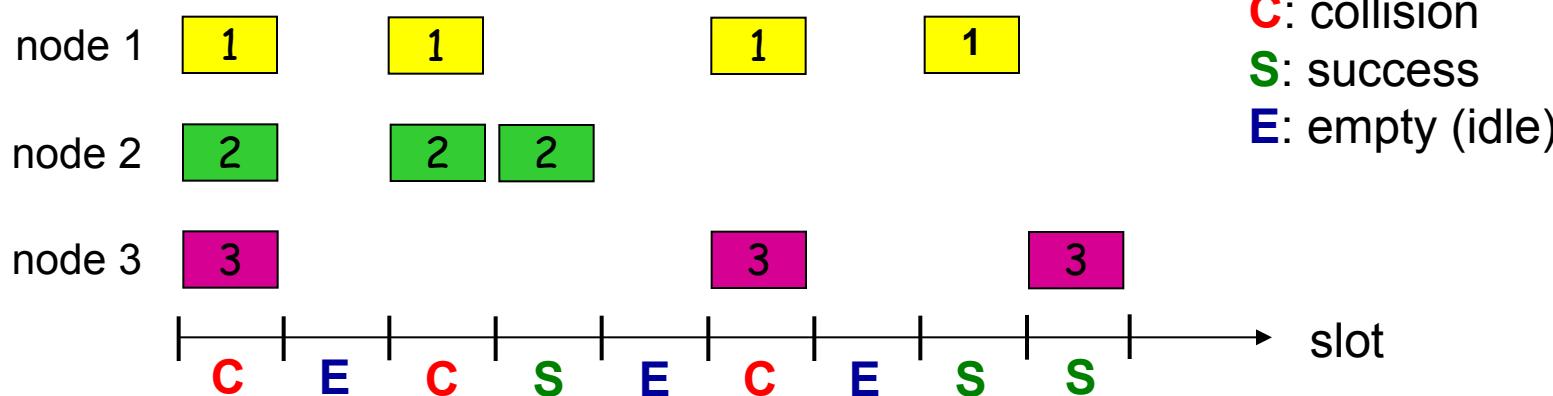
# Slotted ALOHA

## 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.

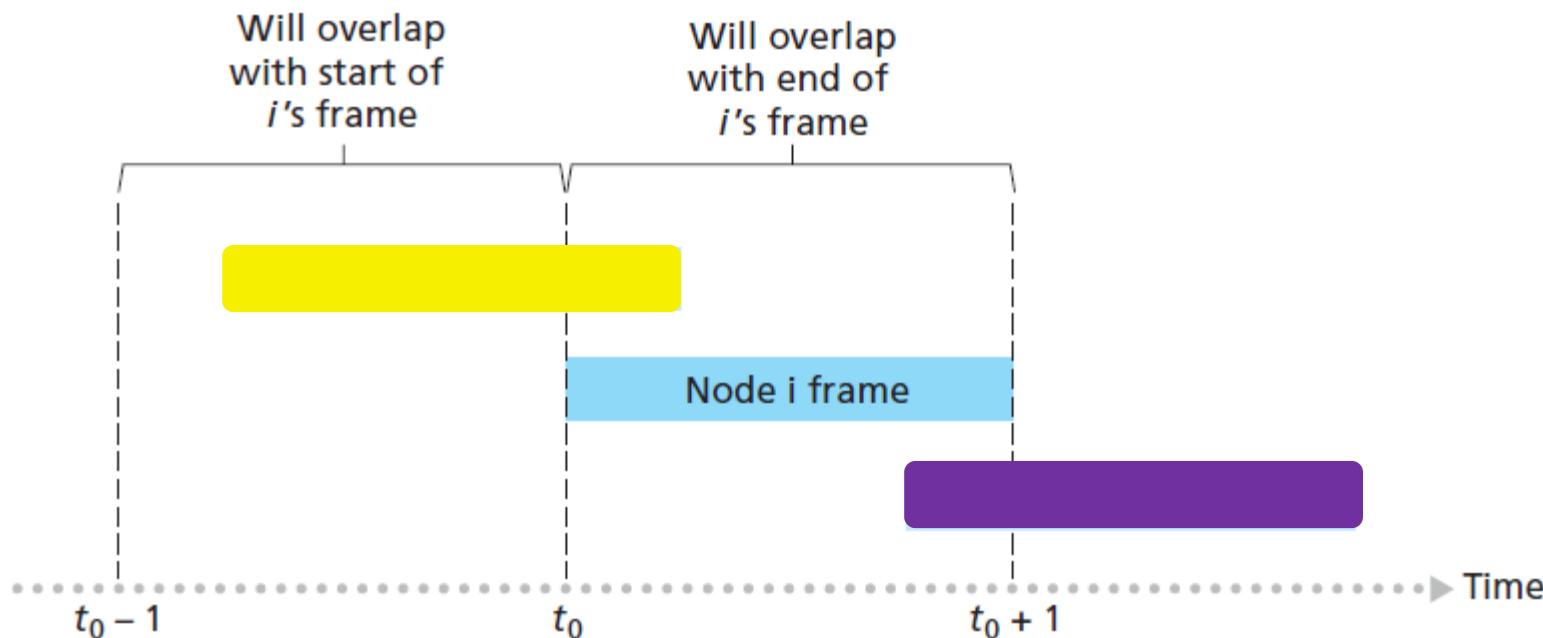
## Operations:

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



# 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  $(t_0 - 1, t_0 + 1)$

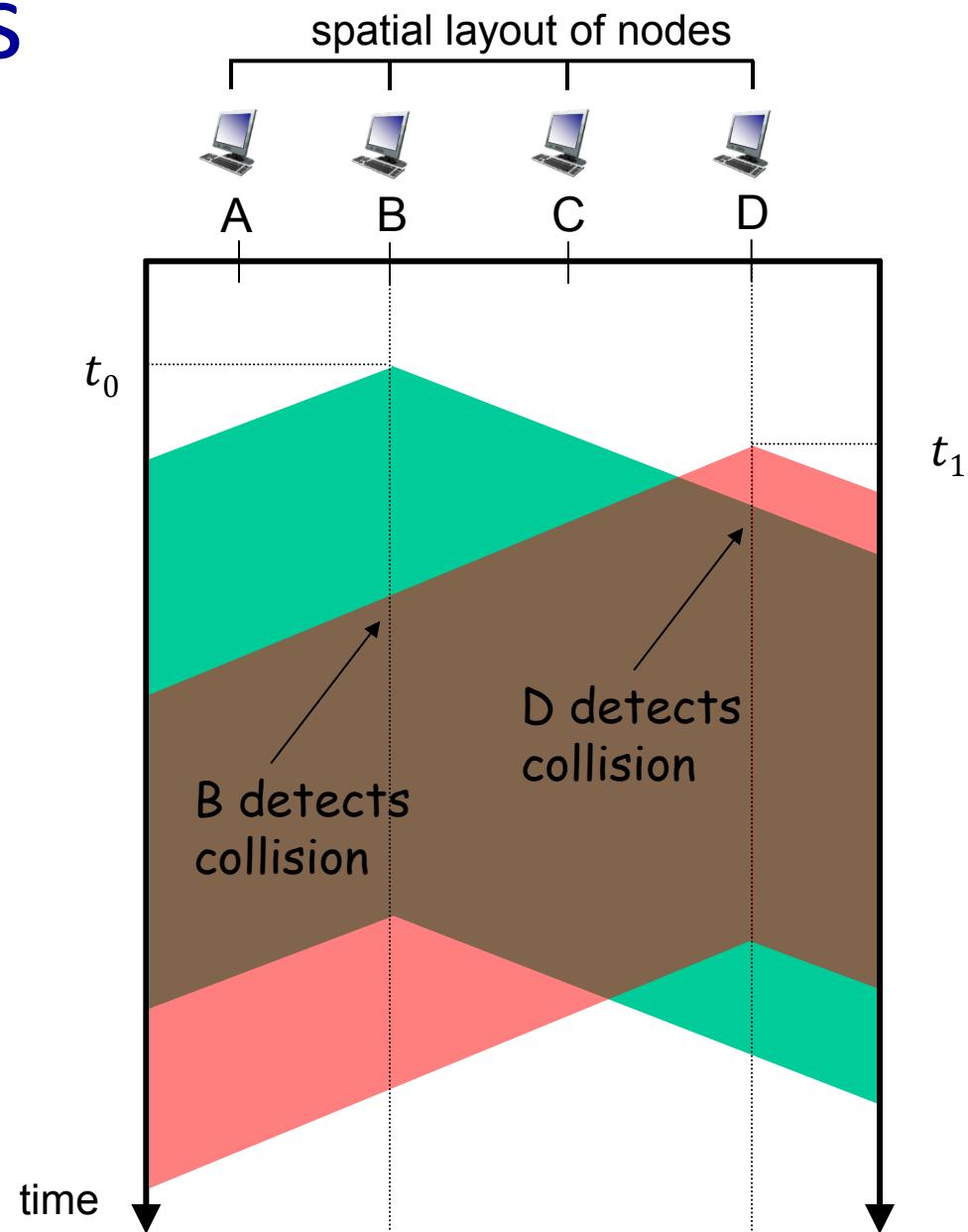


# Carrier Sense Multiple Access

- ❖ **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.

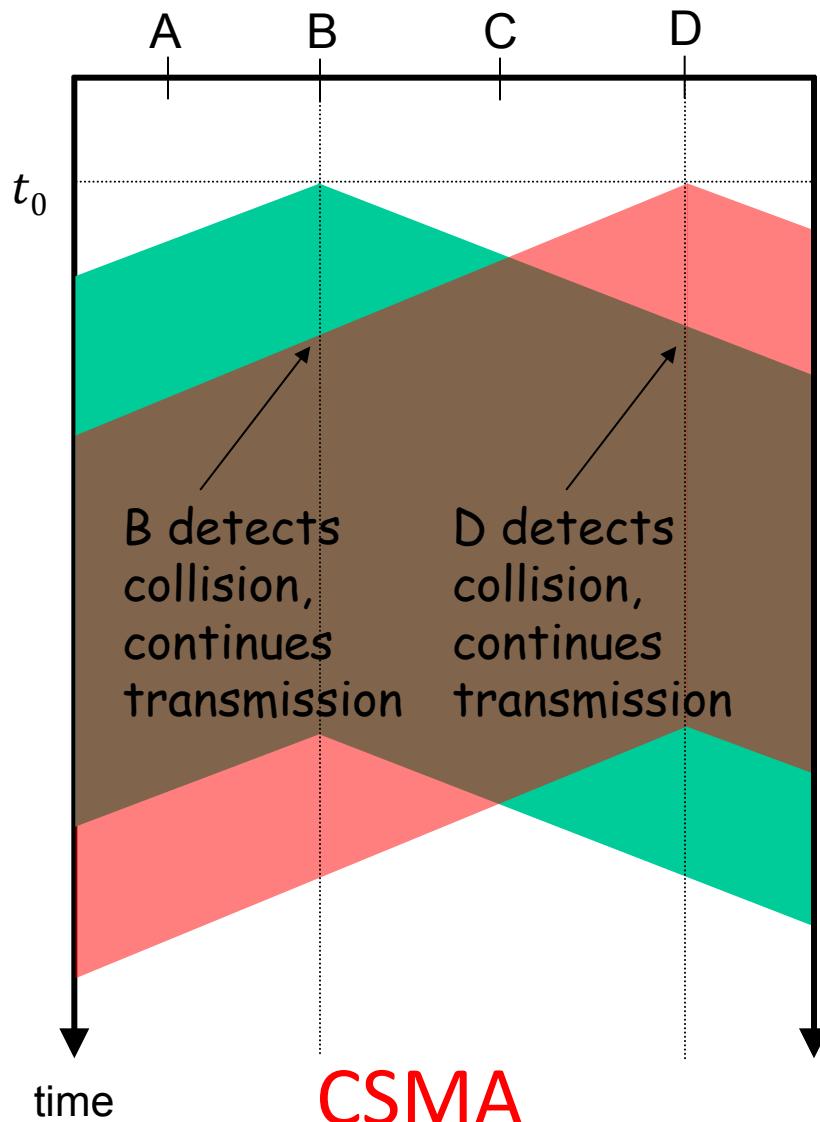


# CSMA/CD (Collision Detection)

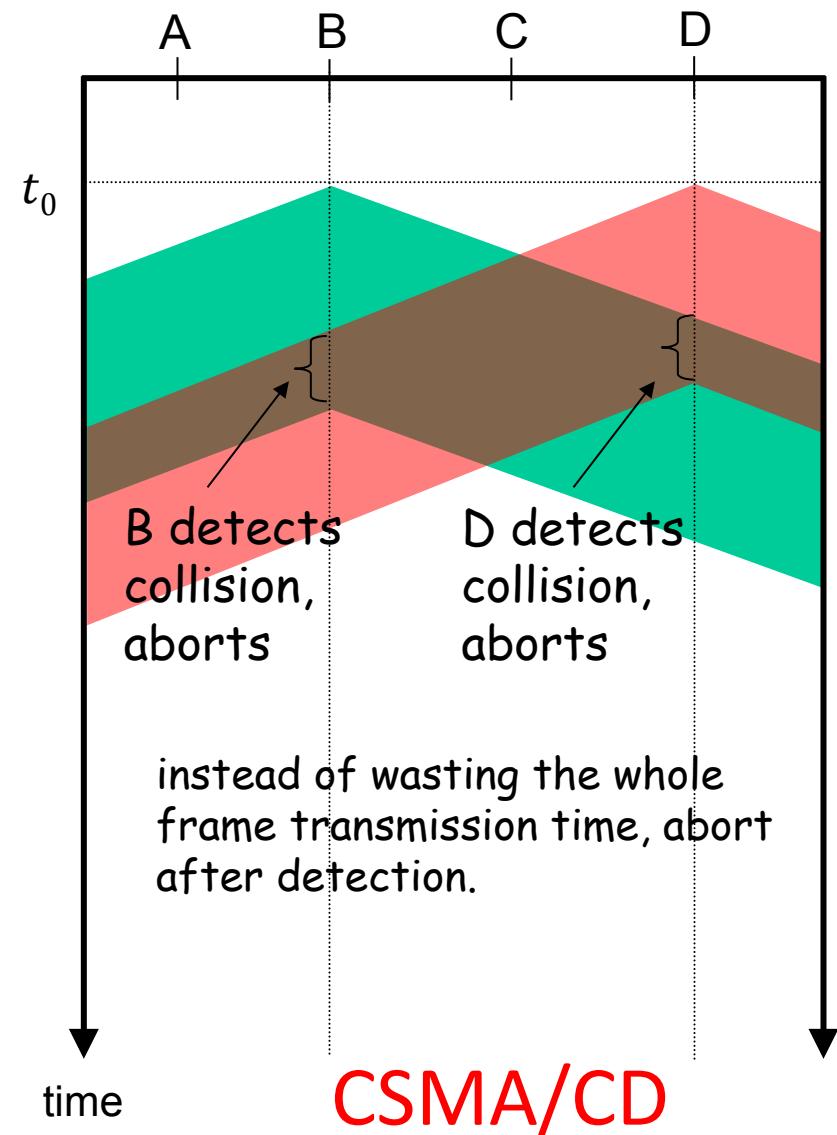
## ❖ CSMA/CD

- 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

spatial layout of nodes

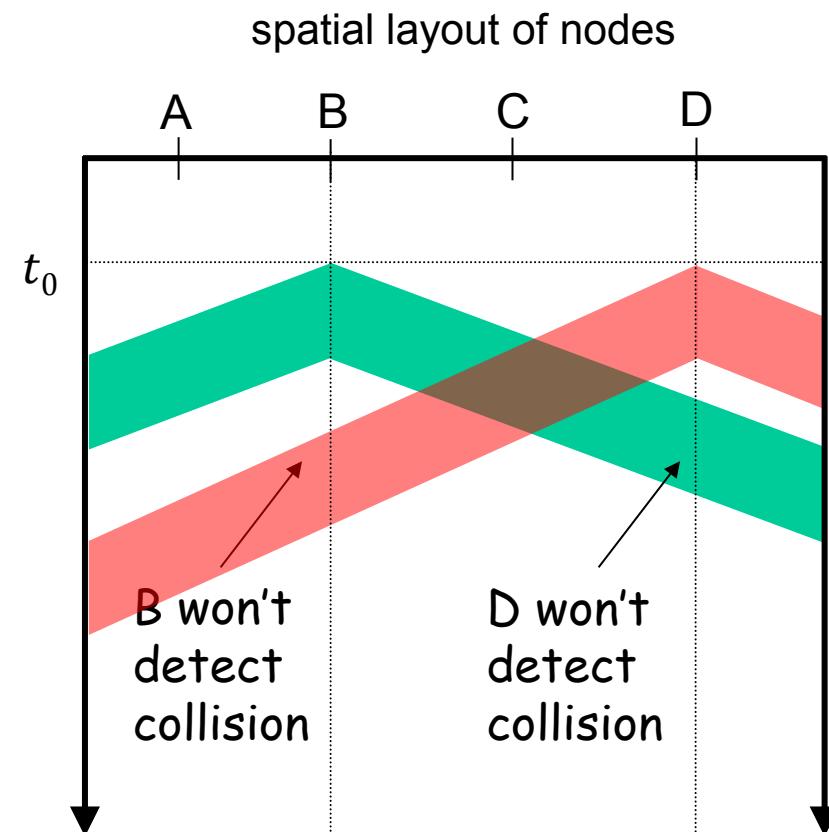


spatial layout of nodes



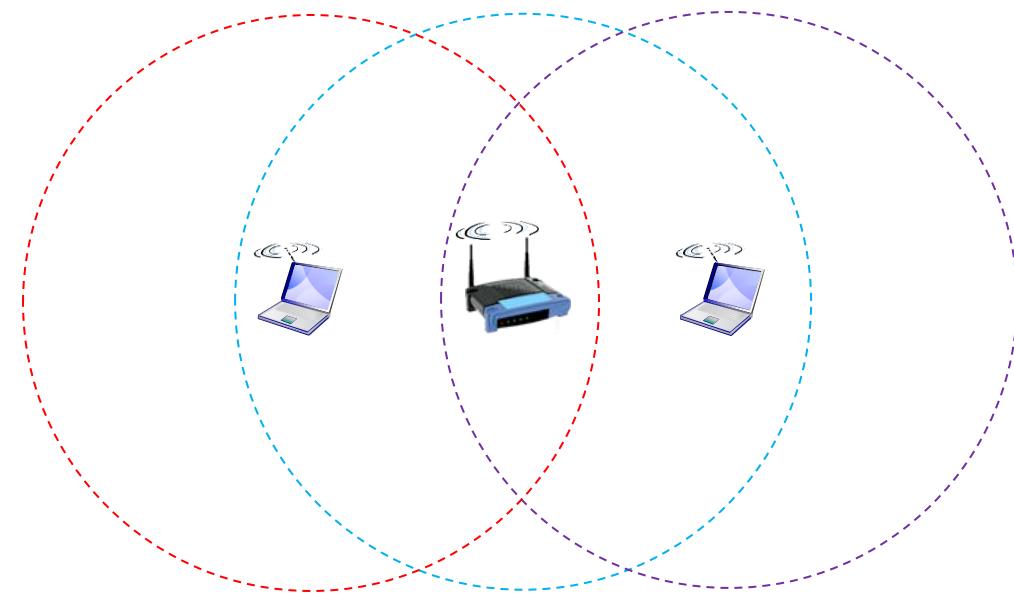
# Minimum Frame Size

- ❖ 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.



# CSMA/CA (Collision Avoidance)

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



Hidden node problem  
(two laptops cannot  
detect each other)

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

# Lecture 9: 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

- ALOHA, S-ALOHA, CSMA
- CSMA/CD used in Ethernet
- CSMA/CA used in 802.11 Wi-Fi

# An Awesome Introduction to Computer Networks

# Link Layer

- ❖ Link layer transmits data over every single link.
- ❖ Possible services by link layer protocols:
  - Framing (basic)
    - IP **datagrams** are encapsulated in link-layer **frames** for transmission.
  - Error checking (optional)
    - Parity check, CRC
  - Multiple access control (optional)
    - Determine in which manner multiple nodes share a broadcast channel.

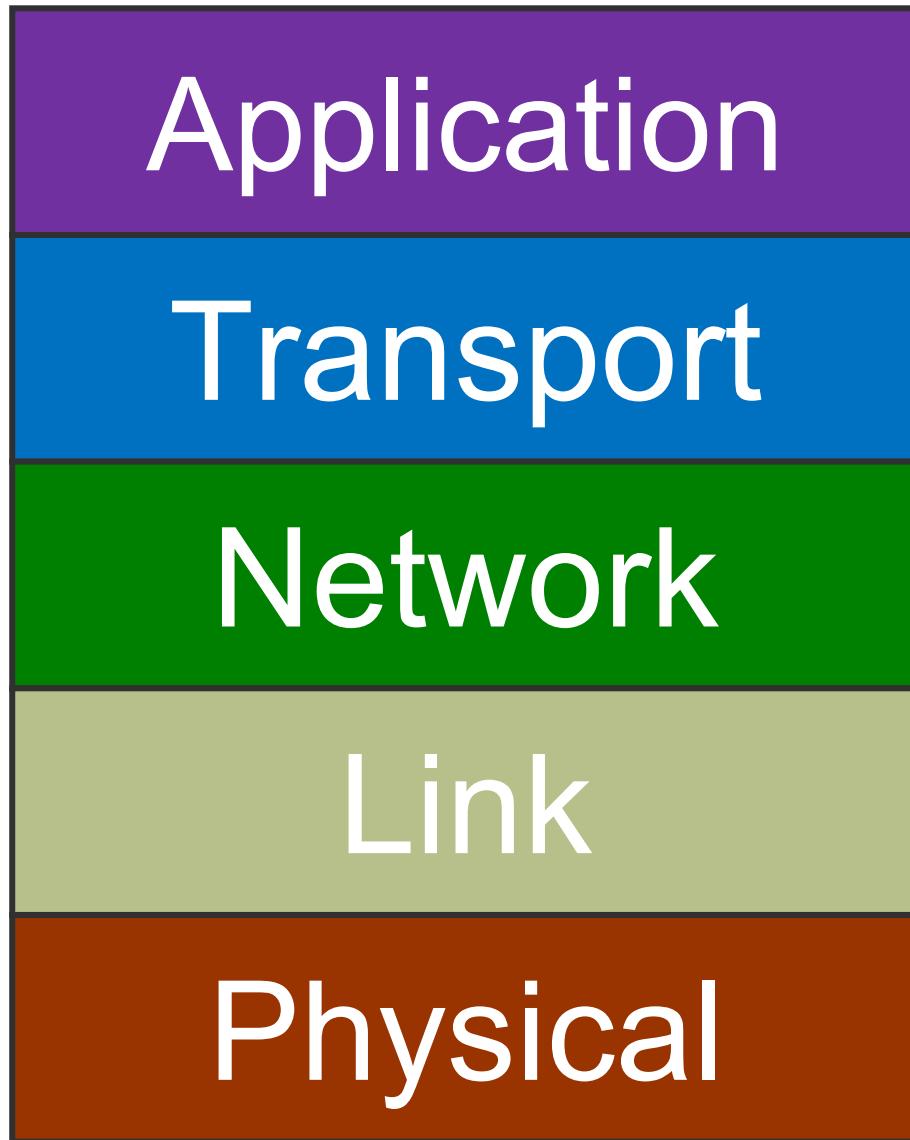
# Multiple Access Protocols

- ❖ **Channel partitioning** protocols:
  - Divide channel into slots by time or frequency
  - Channel is shared fairly, and efficiently if most nodes have data to send.
- ❖ **Random access** protocols
  - Efficient at low load: single node can fully utilize channel
  - When there is a high load, chance of collision is high and channel time is wasted.
- ❖ **“Taking turns”** protocols
  - polling from master node, token passing
  - Efficient at both low and high load
  - Single point of failure

# Lecture 10: Local Area Network

*After this class, you are expected to understand:*

- ❖ the role of MAC address.
- ❖ how ARP allows a host to discover the MAC addresses of other nodes in the same subnet.
- ❖ the CSMA/CD algorithm of Ethernet.
- ❖ the role of switch in interconnecting subnets in a LAN.
- ❖ how switching table is built and how it is used to forward link-layer frames.



You are  
still here

# Lecture 10: Roadmap

5.1 Introduction to the Link Layer

5.2 Error Detection and Correction

5.3 Multiple Access Protocols

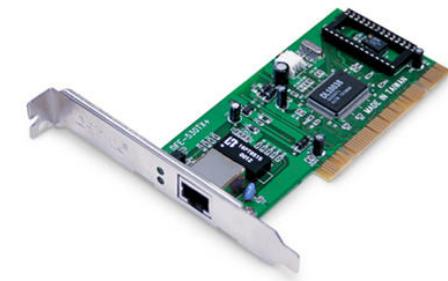
## 5.4 Switched Local Area Networks

- 5.4.1 Link Layer Addressing & ARP
- 5.4.2 Ethernet
- 5.4.3 Link-layer Switches

Kurose Textbook, Chapter 5  
(Some slides are taken from the book)

# 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.
  - 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.:

[http://www.coffer.com/mac\\_find/](http://www.coffer.com/mac_find/)

# IP Address vs. MAC Address

## ❖ IP address

- 32 bits in length
- network-layer address used to move **datagram** 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 **frame** 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?
  - Use ARP
- ❖ 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; TTL >

time after which address mapping will be forgotten (typically 1 - 2 min on Windows)

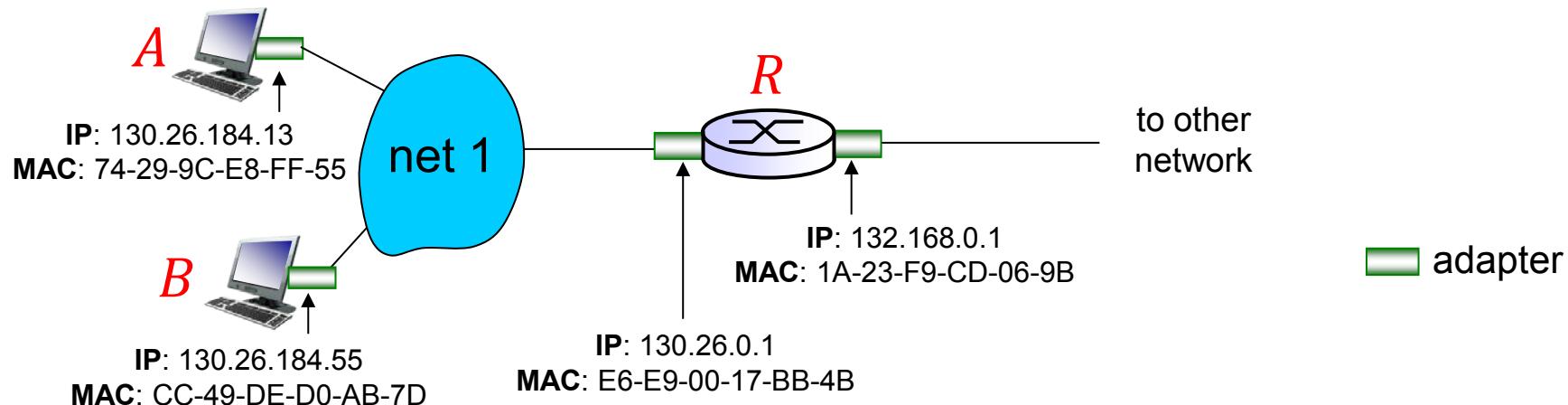
# Sending Frame in the Same Subnet

- ❖ 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.
- All other nodes will receive but 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.

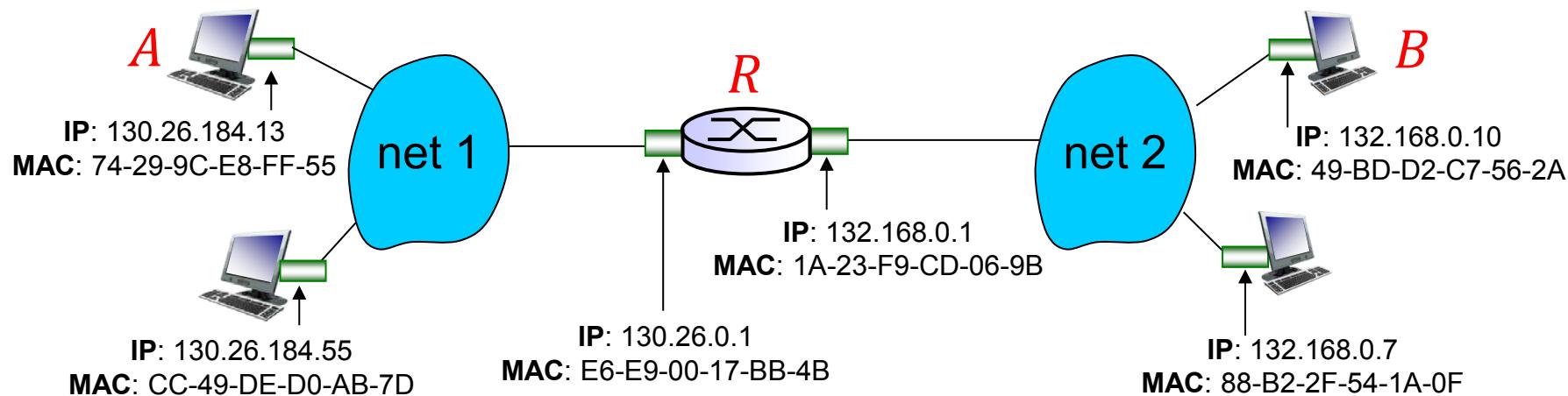
- Reply frame is sent to  $A$ 's MAC address.

③  $A$  caches  $B$ 's IP-to-MAC address mapping in its ARP table.

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

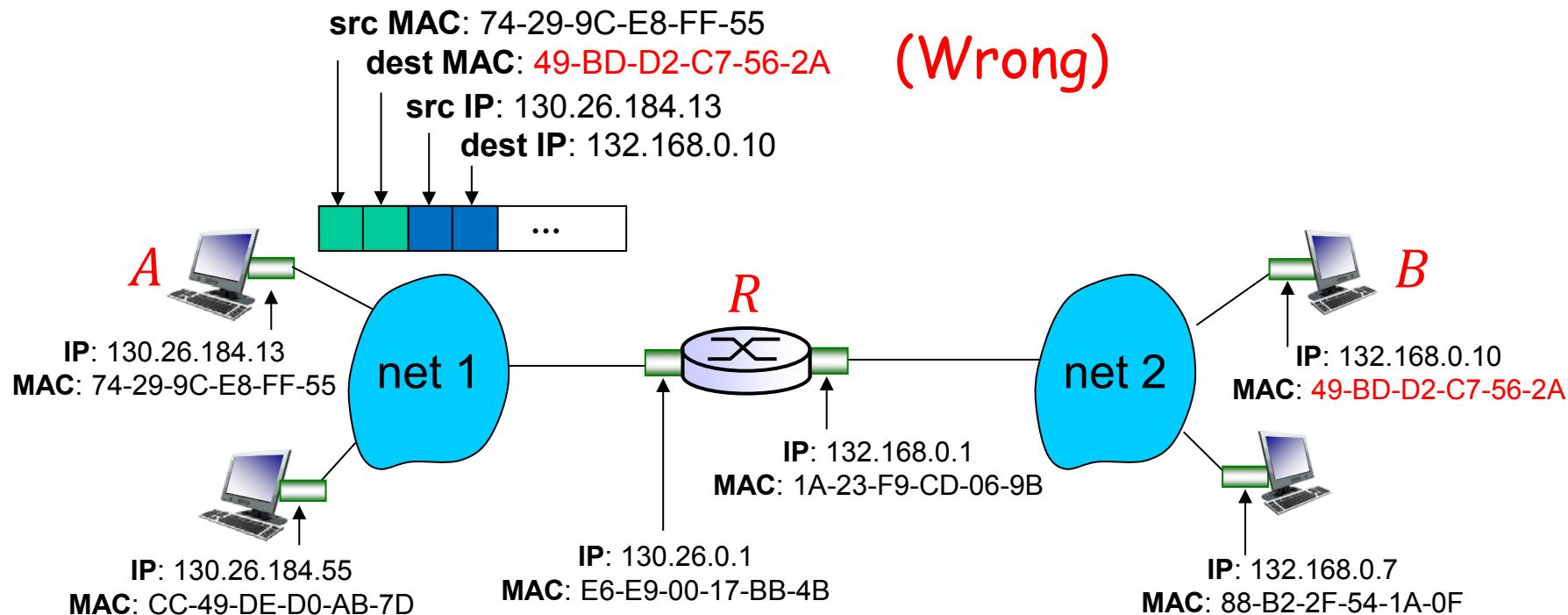
# Sending Frame to Another Subnet

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



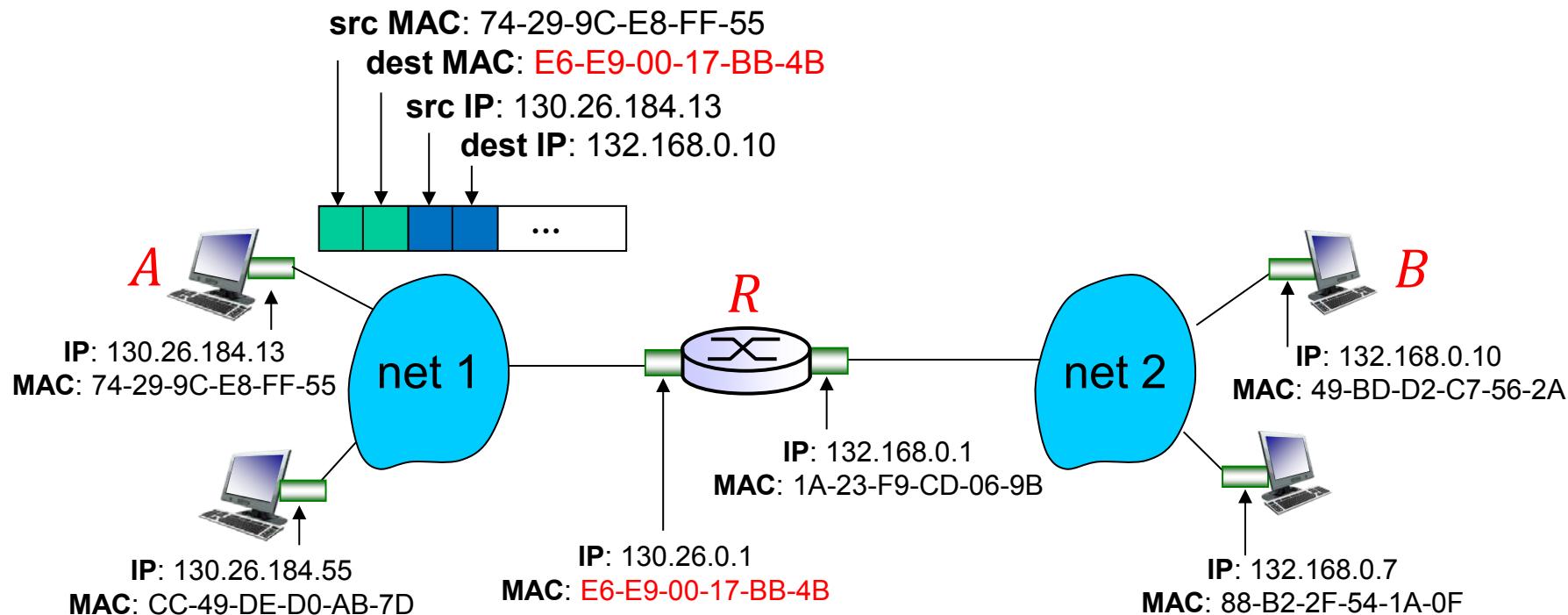
# Sending Frame to Another Subnet

- ❖ 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.



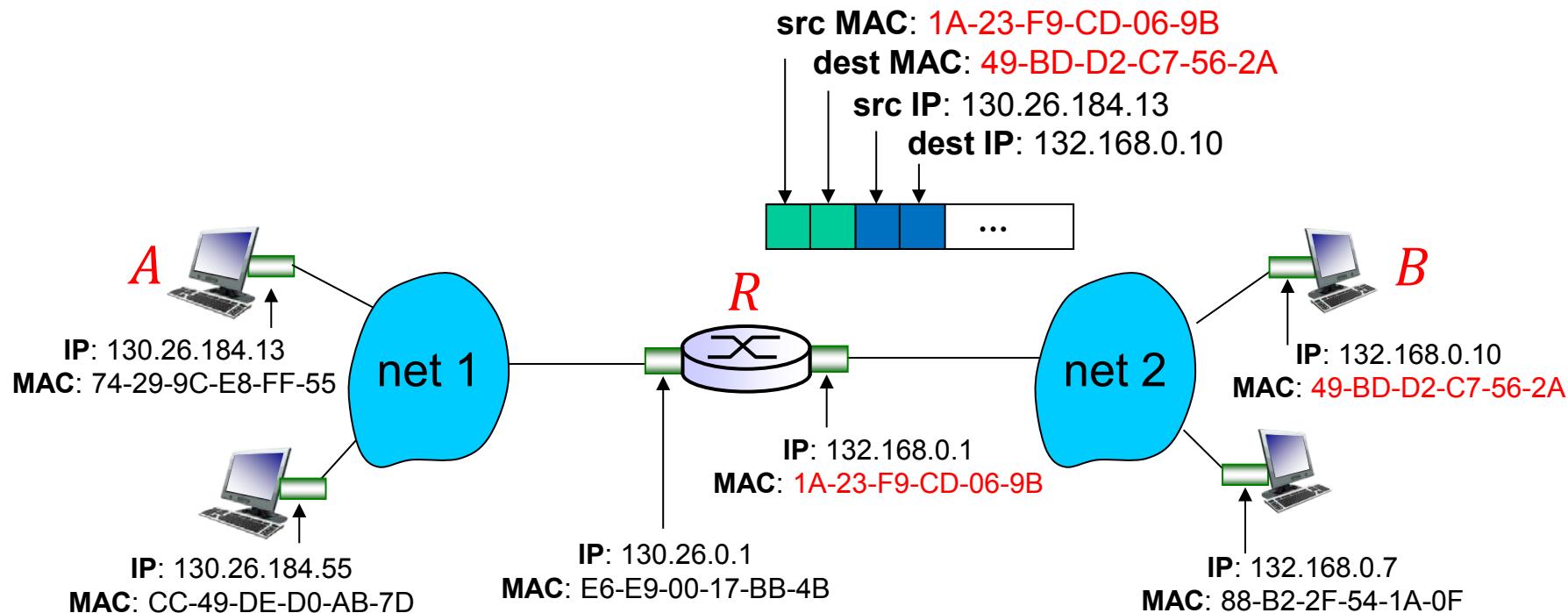
# Sending Frame to Another Subnet

- ❖ A sends datagram to B in another subnet.
  - A 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.



# Lecture 10: Roadmap

5.1 Introduction to the Link Layer

5.2 Error Detection and Correction

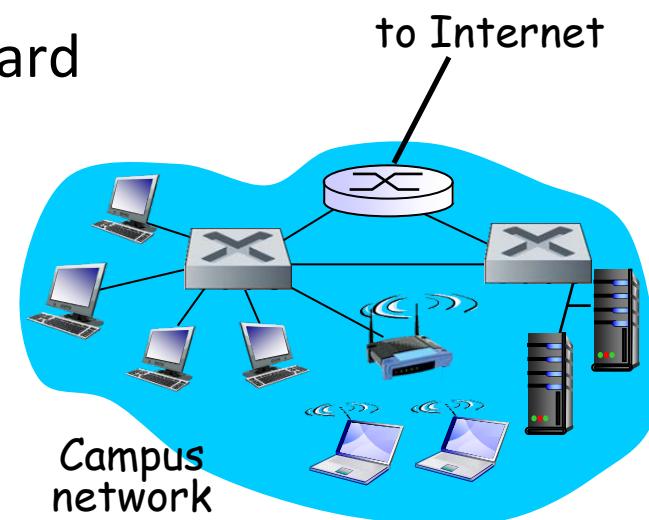
5.3 Multiple Access Protocols

## 5.4 Switched Local Area Networks

- 5.4.1 Link Layer Addressing & ARP
- 5.4.2 Ethernet
- 5.4.3 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 technologies:
  - IBM Token Ring: IEEE 802.5 standard
  - Ethernet: IEEE 802.3 standard
  - Wi-Fi: IEEE 802.11 standard
  - Others

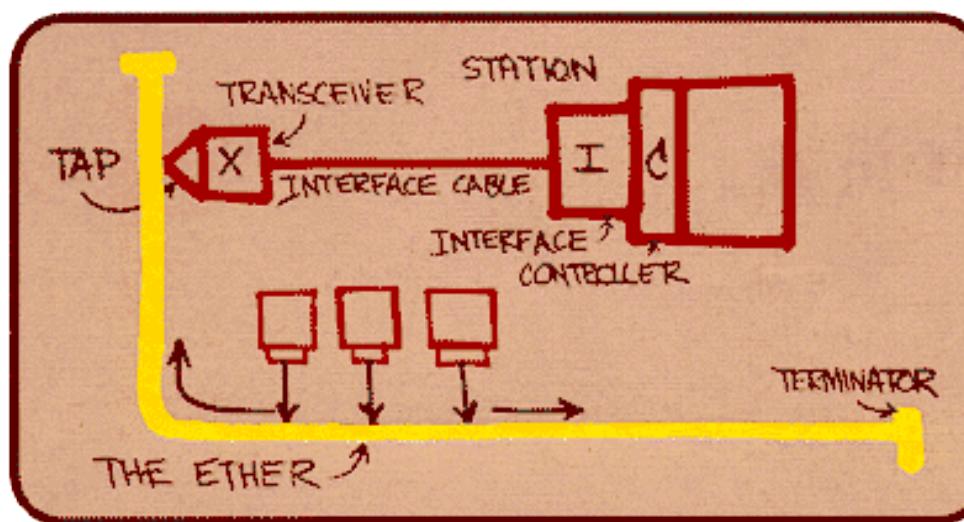


# Ethernet

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



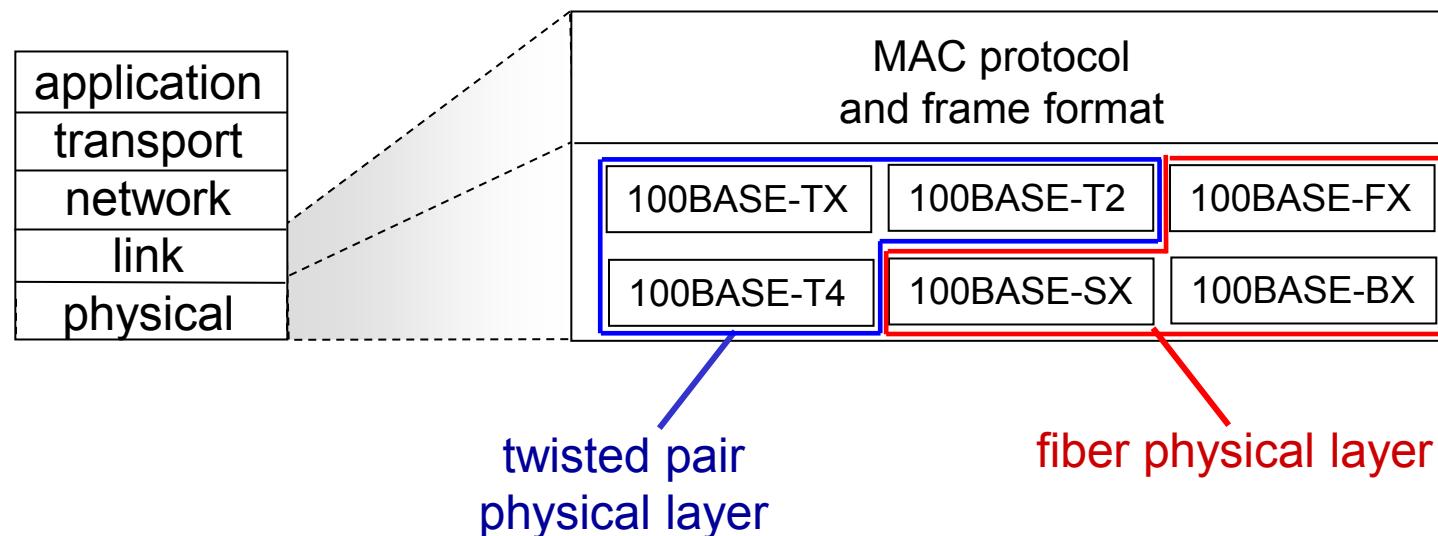
Ethernet connection  
(Source: Wikipedia)



*Metcalfe's  
Ethernet sketch*

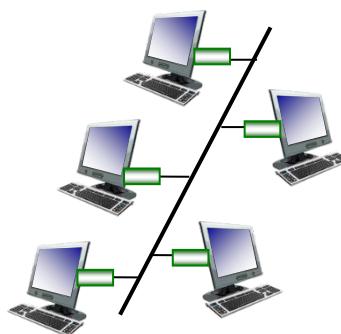
# 802.3 Ethernet Standards

- ❖ A series of Ethernet standards are 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

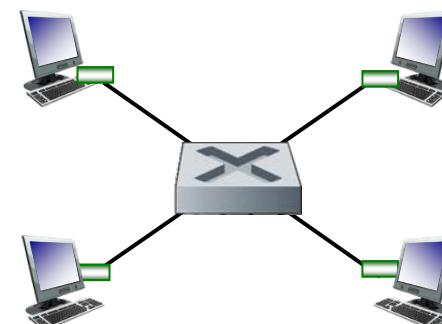


# Ethernet: Physical Topology

- ❖ **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



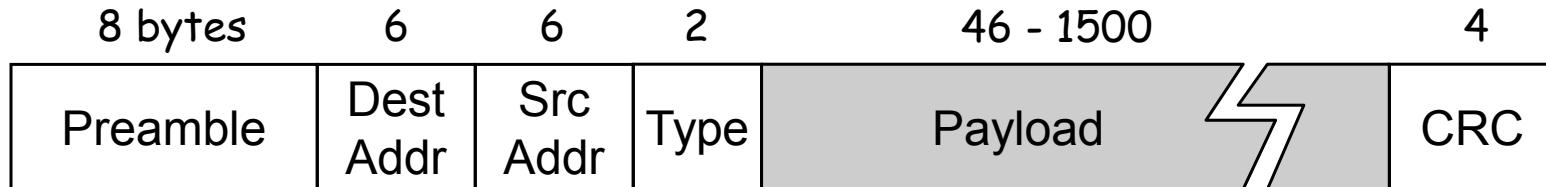
Ethernet with bus topology



Ethernet with star topology

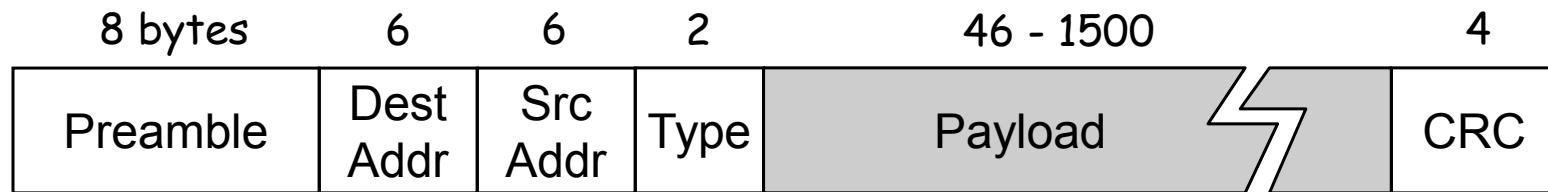
# Ethernet Frame Structure (1/2)

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



- ❖ *Preamble:*
  - 7 bytes with pattern **10101010** followed by 1 byte with pattern **10101011**.
  - used to synchronize receiver and sender clock rates.

# Ethernet Frame Structure (2/2)



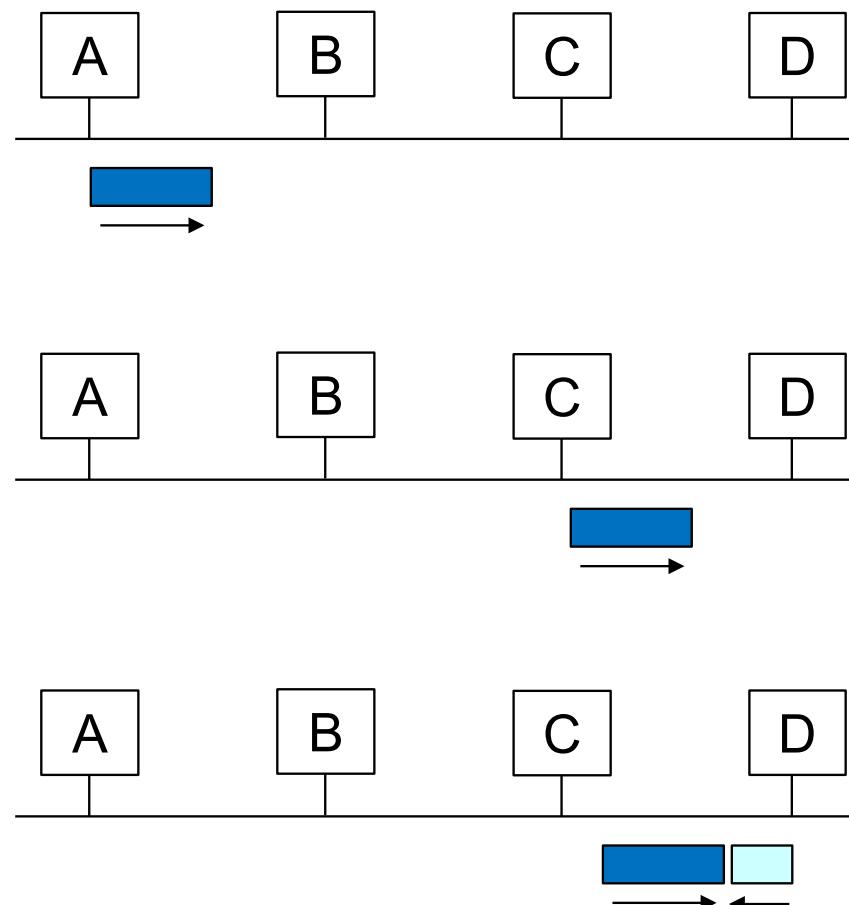
- ❖ *Source and dest MAC address:*
  - If NIC receives a frame with matching destination address, or with broadcast address, it passes data in the frame to network layer protocol.
  - Otherwise, NIC discards frame.
- ❖ *Type:* Indicates higher layer protocol (mostly IP).
- ❖ *CRC:* corrupted frame will 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 1<sup>st</sup> collision: choose  $K$  at random from {0, 1}; wait  $K * 512$  bit transmission times before retransmission.
- ❖ After 2<sup>nd</sup> 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.

# Lecture 10: Roadmap

5.1 Introduction to the Link Layer

5.2 Error Detection and Correction

5.3 Multiple Access Protocols

## 5.4 Switched Local Area Networks

- 5.4.1 Link Layer Addressing & ARP
- 5.4.2 Ethernet
- 5.4.3 Link-layer Switches

# Ethernet Switch

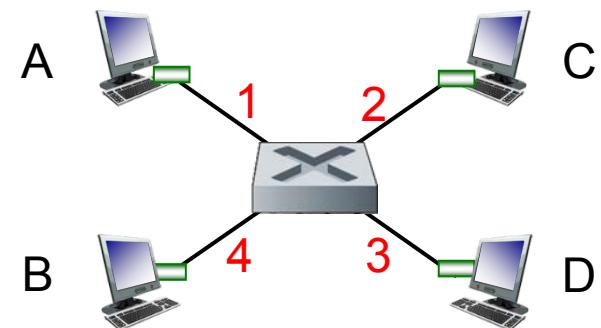
- ❖ A link-layer device used in LAN
  - Store and forward Ethernet frames
  - Examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links.
- ❖ Transparent to hosts
  - No IP address
  - Hosts are unaware of the presence of switches



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.
- ❖ CSMA/CD protocol is used on each link, but no collisions!



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

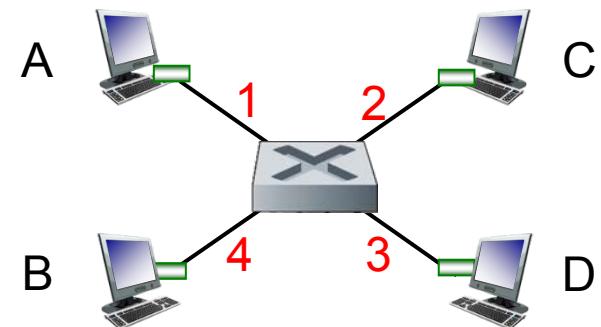
# Switch Forwarding Table

❖ **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 >

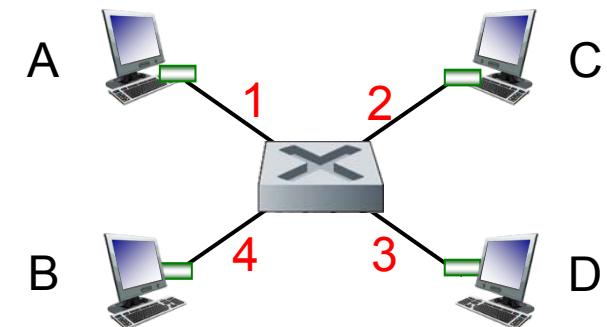


A switch with 4 interfaces  
(1, 2, 3, 4)

❖ **Q:** how are entries created and maintained in a switch table?

# 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.



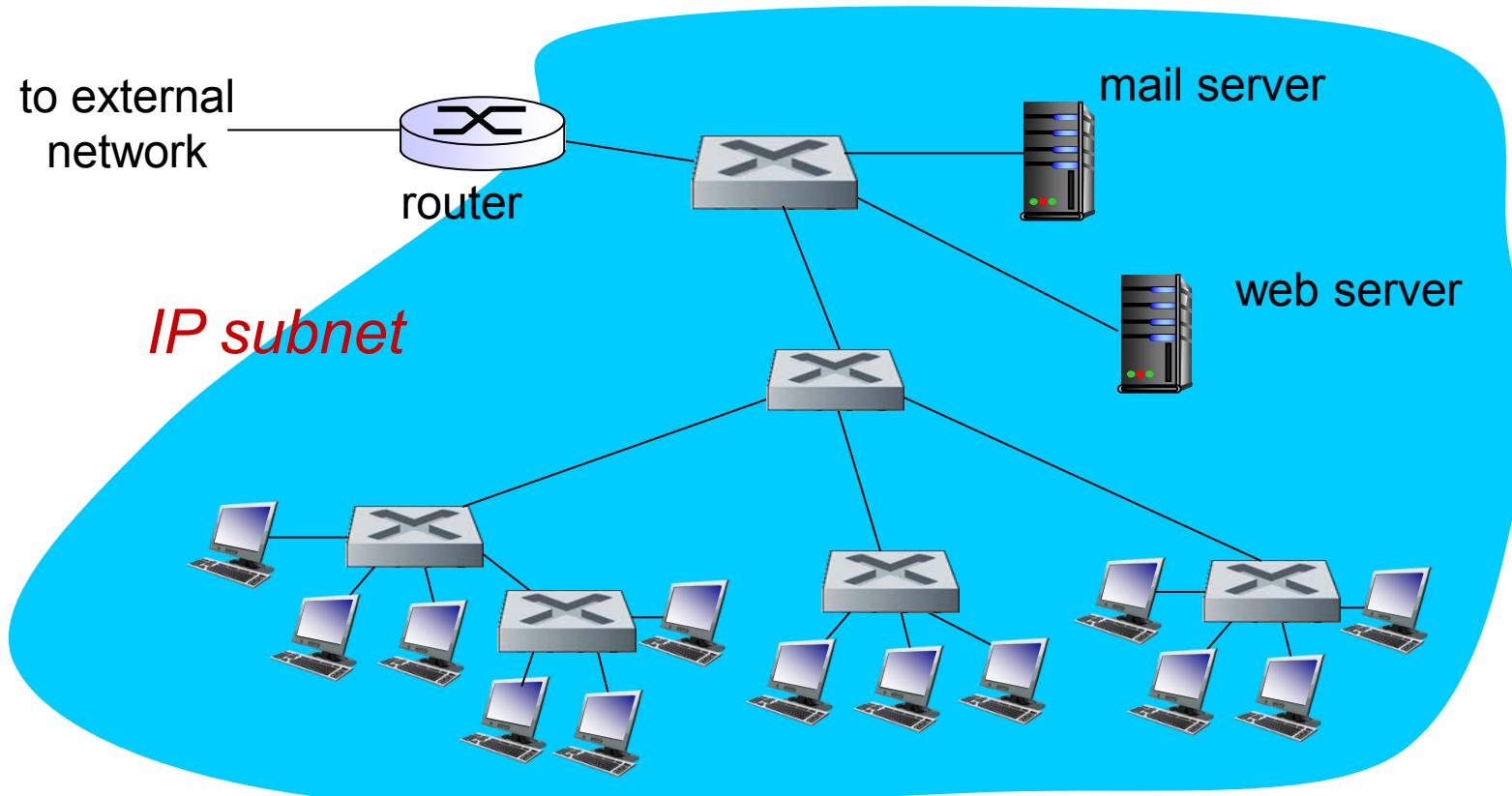
A switch with 4 interfaces  
(1, 2, 3, 4)

| MAC addr | Interface | TTL |
|----------|-----------|-----|
| A        | 1         | 60  |
|          |           |     |
|          |           |     |
|          |           |     |

Switch table (initially empty)

# Interconnecting Switches

- ❖ Switches can be connected in hierarchy.



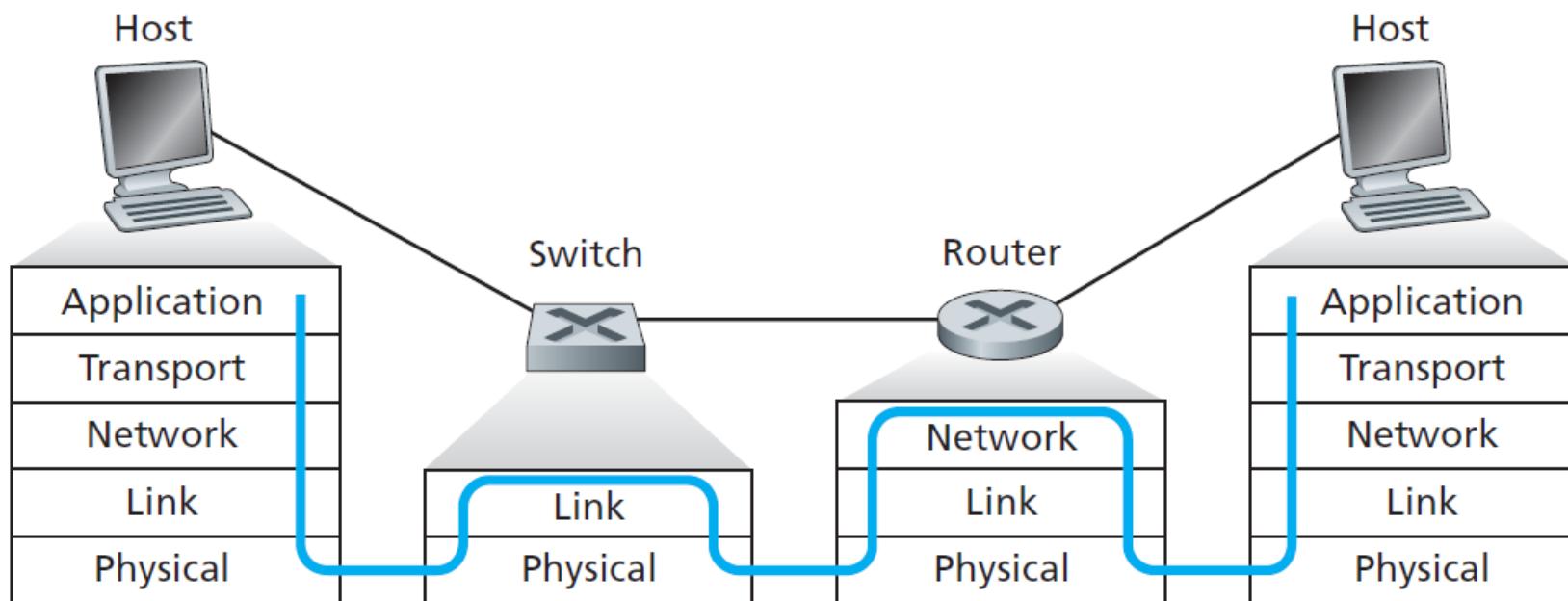
# Switches vs. Routers

## ❖ 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 10: 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 switch and switch table

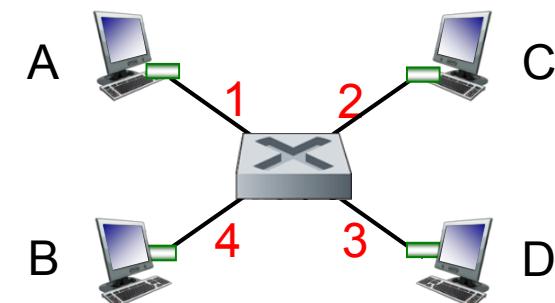
# An Awesome Introduction to Computer Networks



# Last lecture!

# Local Area Network (LAN)

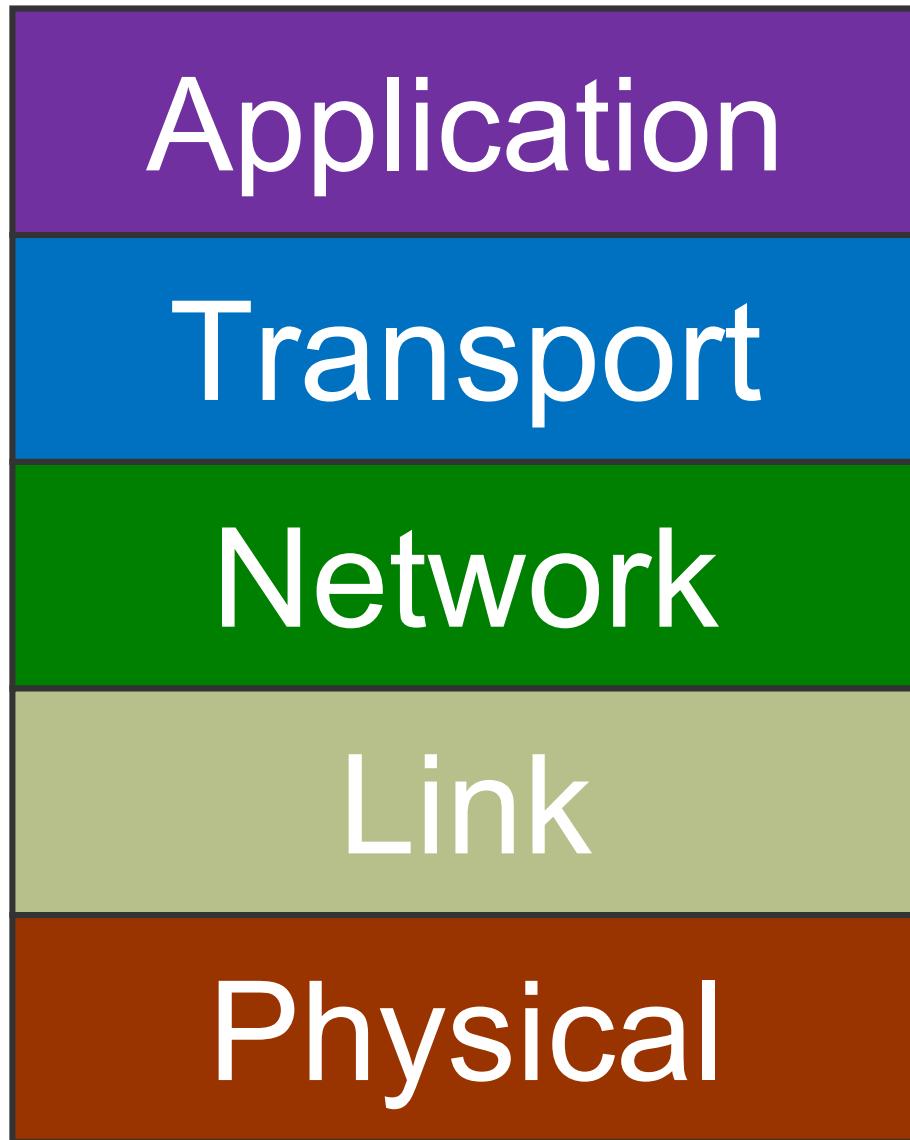
- ❖ Every adapter has a unique MAC address.
- ❖ ARP is used to discover MAC addresses of other nodes in the same subnet.
- ❖ Ethernet: CSMA/CD with binary back-off
- ❖ Switch self-learns the topology of a subnet.



# Lecture 11: Physical Layer

*After this class, you are expected to understand:*

- ❖ different methods of digital transmission.
- ❖ the theoretical capacity of a channel calculated from Shannon's formula.
- ❖ how modulation techniques work and the concept of constellation diagram.



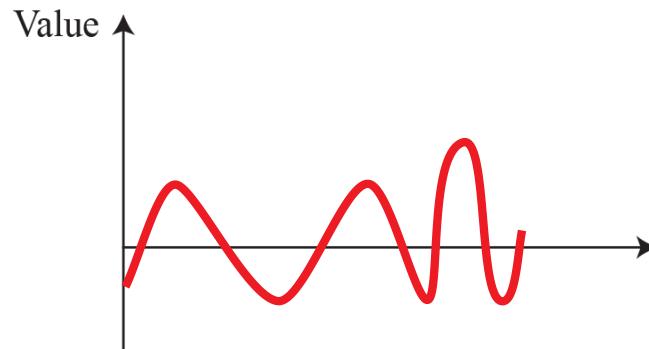
You are  
here

# Lecture 11: Roadmap

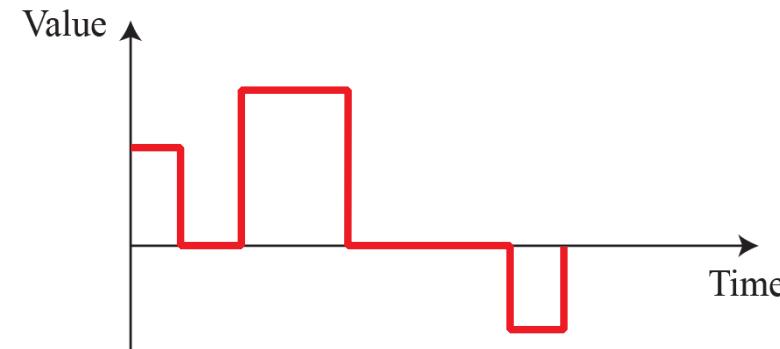
1. Digital transmission
2. Analog transmission
3. A quick revision
4. Exam matters

# Digital and Analog Signals

- ❖ Physical layer moves data in the form of electro-magnetic signals across transmission medium.
- ❖ 0s and 1s can be transmitted as either analog signal or digital signal.
  - **Analog signal** is continuous, with infinitely many levels.
  - **Digital signal** has a limited number of defined values.



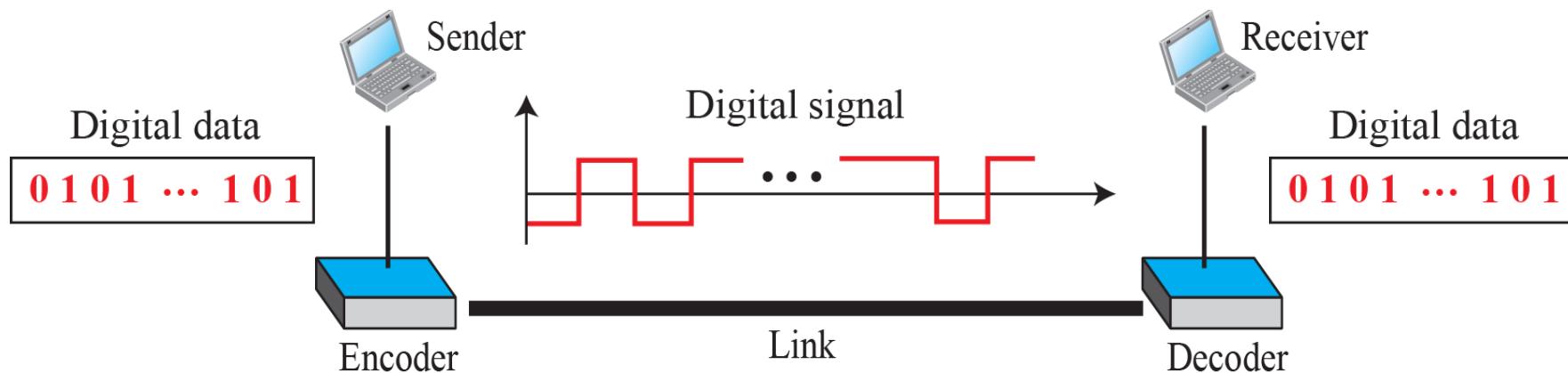
a. Analog signal



b. Digital signal

# Digital Transmission

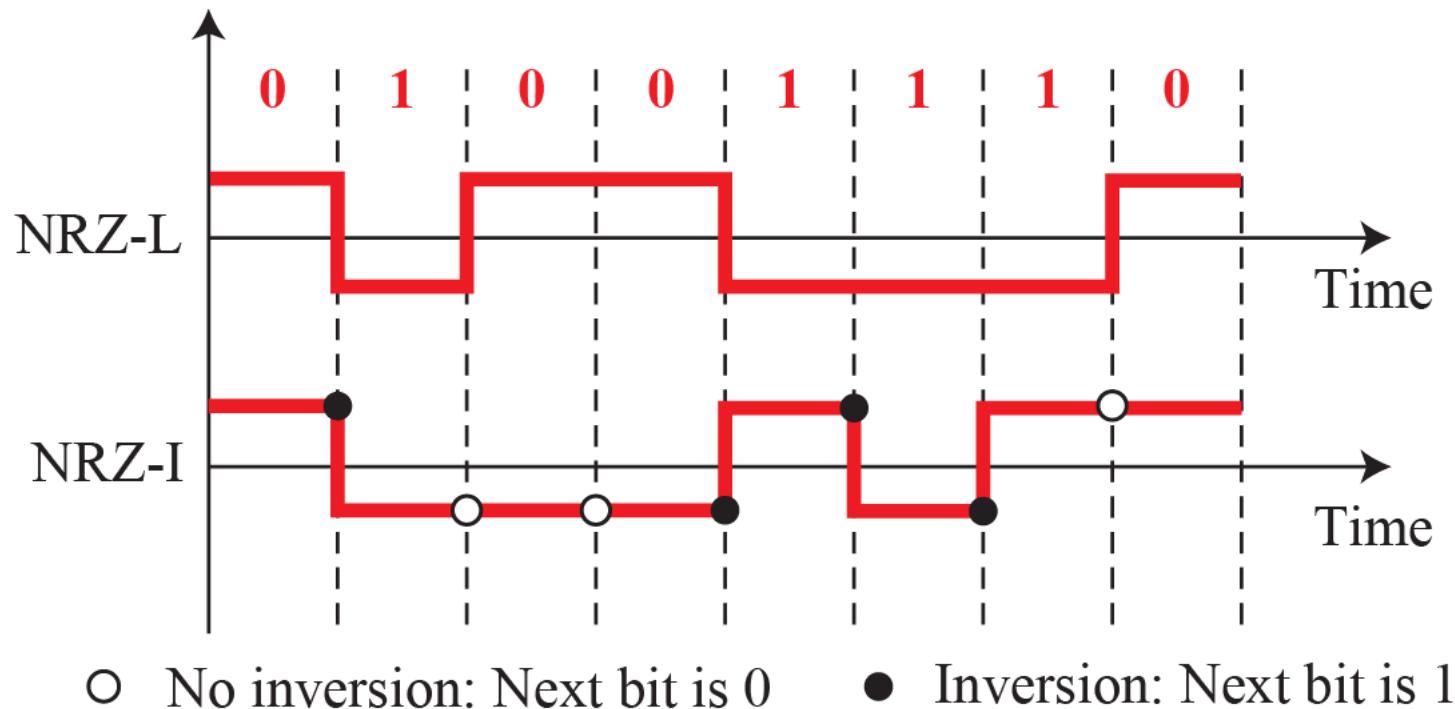
- ❖ In digital transmission, we encode 0s and 1s with different voltages to be transmitted over the wire.



- ❖ We will discuss 3 digital encoding methods.

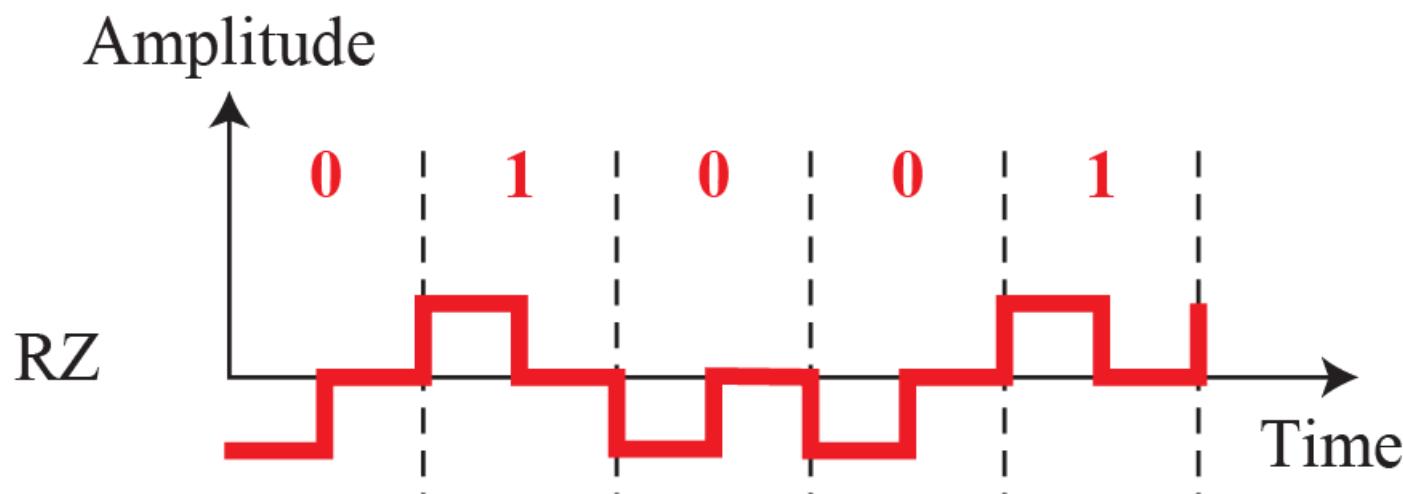
# NRZ (Non-Return-to-Zero)

- ❖ NRZ encoding uses two voltage levels. It has two variations.
  - NRZ-L: absolute voltage level determines value of a bit.
  - NRZ-I: inverts the voltage if bit 1 is encountered.



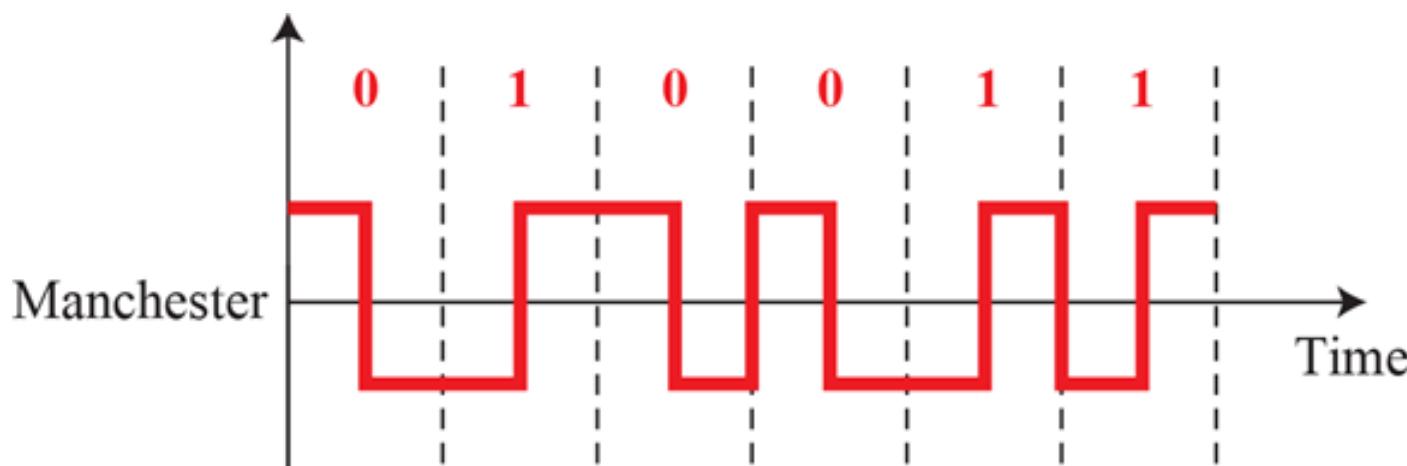
# RZ (Return-to-Zero)

- ❖ RZ encoding uses three voltage levels. It always returns the voltage to zero halfway through a bit interval.



# Manchester

- ❖ Manchester coding inverts the signal in the middle of a bit.
  - A  $-ve$  to  $+ve$  transition represents 1. A  $+ve$  to  $-ve$  transition represents 0.



# Lecture 11: Roadmap

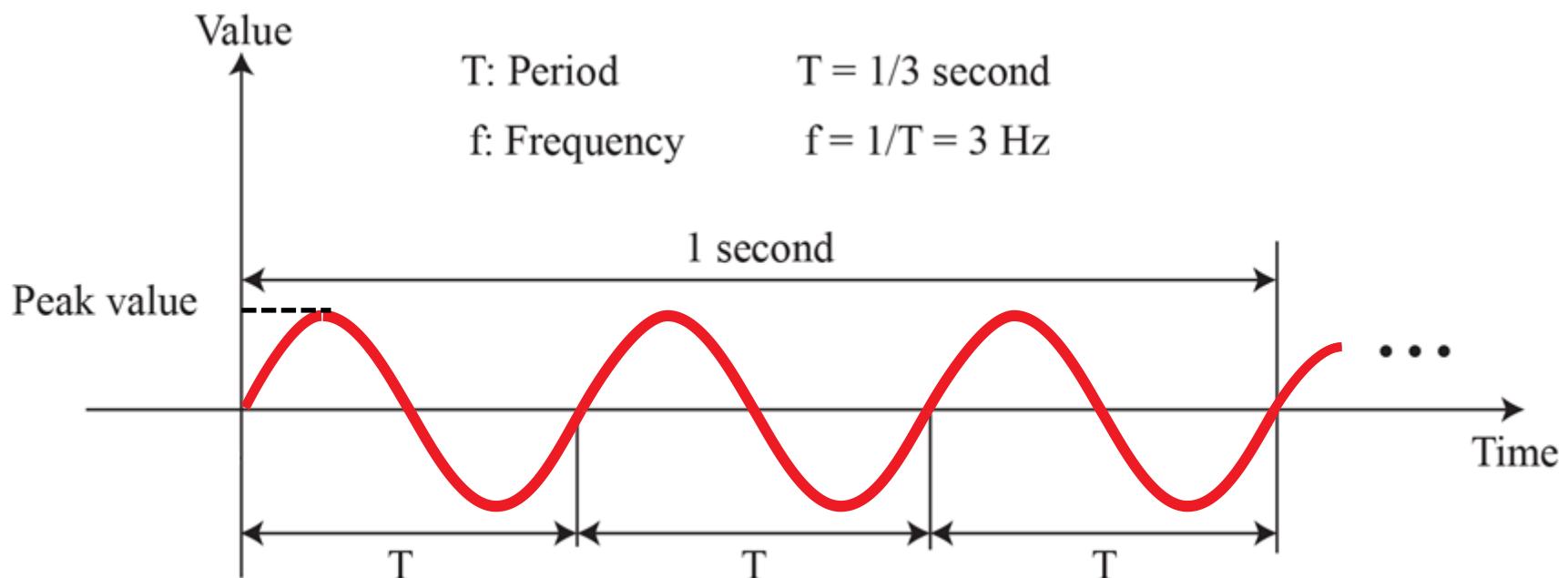
1. Digital transmission
2. Analog transmission
3. A quick revision
4. Exam matters

# Analog Signal

- The most basic analog signal is a sine wave.

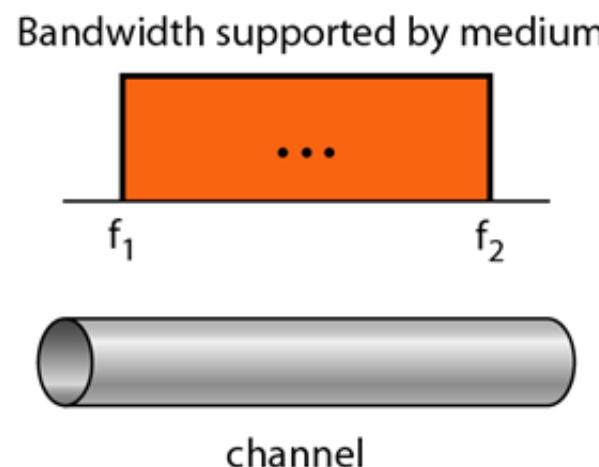
$$A \sin (2\pi f t + \phi)$$

Peak amplitude      frequency      phase



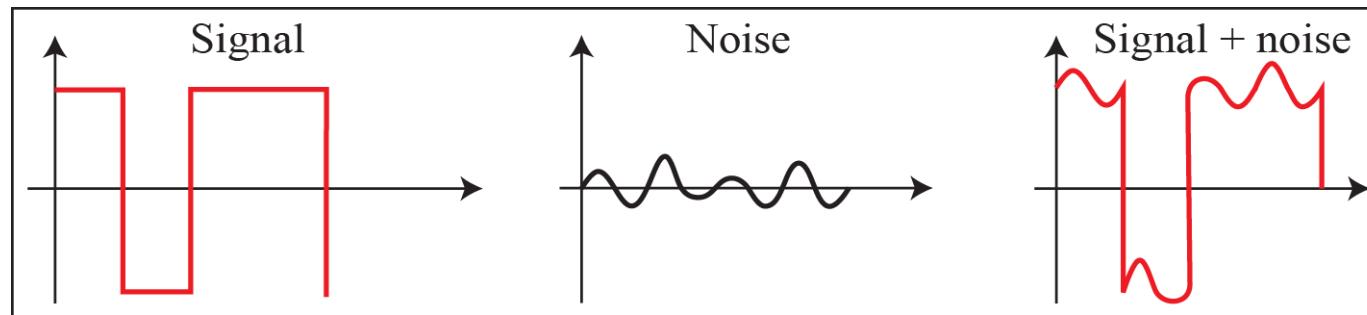
# Channel Bandwidth

- ❖ A transmission channel only allows signals in a certain frequency range to pass through.
- ❖ The difference in the highest frequency and lowest frequency that can pass through a channel is known as the **bandwidth of the channel**.

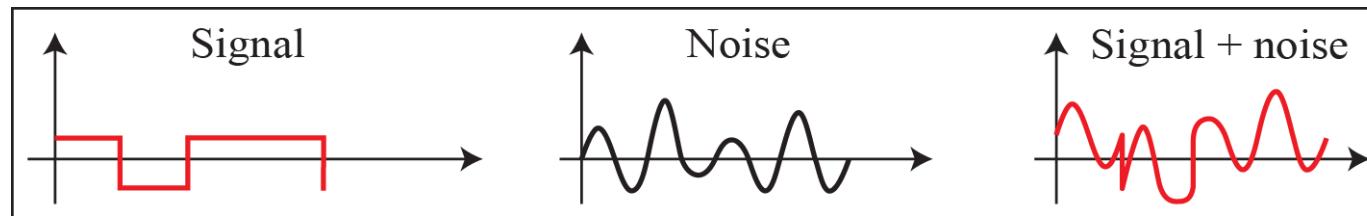


# Signal to Noise Ratio (SNR)

- ❖ A transmission channel introduces noise that distorts signal.
  - **Signal to noise ratio (SNR)** measures the strength of signal over noise.



a. High SNR



b. Low SNR

# Shannon Channel Capacity

- ❖ The theoretical maximum bit rate of a noisy channel is given by **Shannon Capacity**:

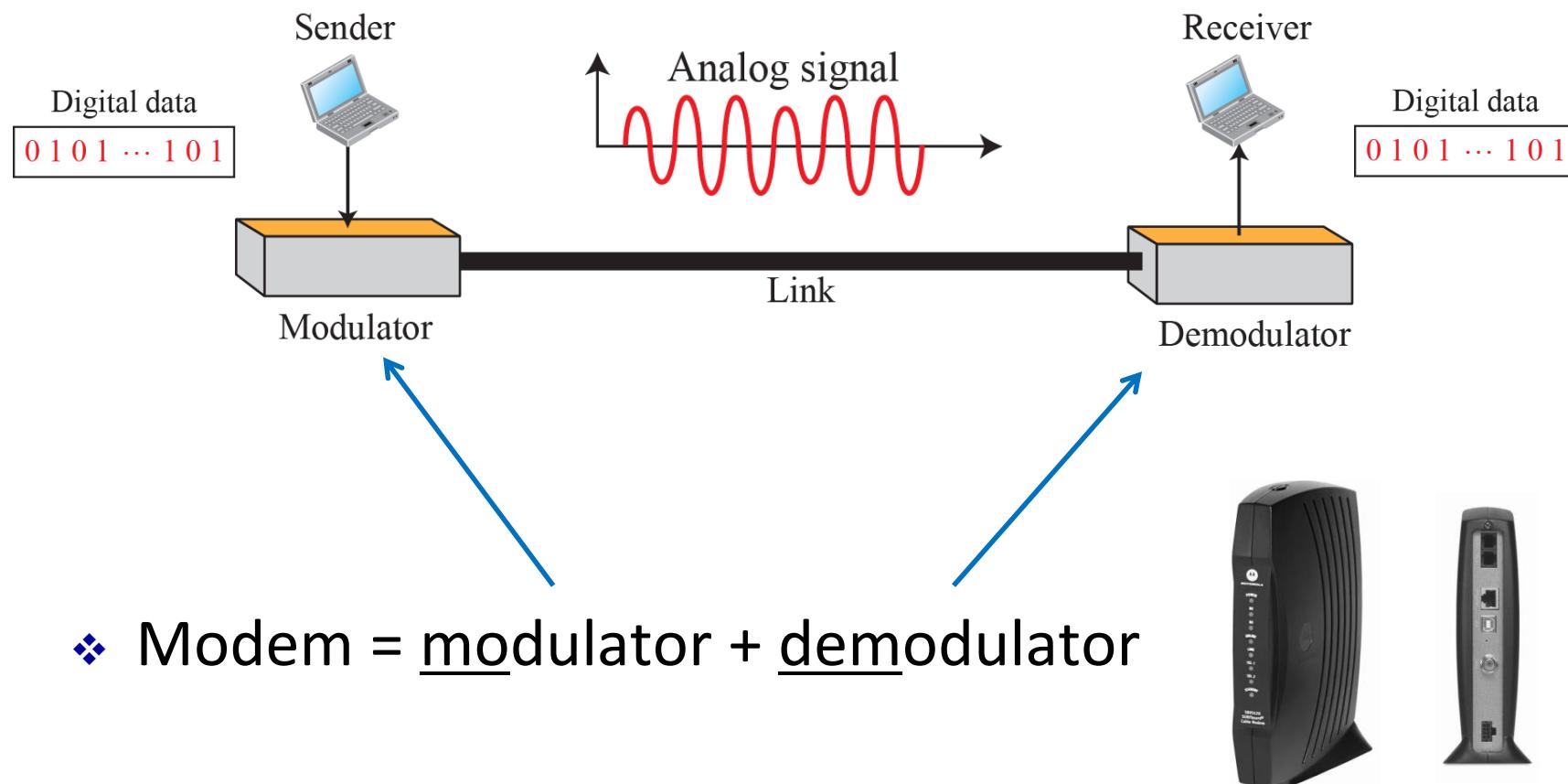
$$C = B * \log_2(1 + SNR)$$

Channel bandwidth

Signal to noise ratio of channel

- ❖ Example: Phone line has a bandwidth of 3000 Hz (300 to 3300 Hz) and SNR of 3162. The capacity of the channel is 34881 bps.
  - The highest bit rate for a telephone line is 34.881 kbps

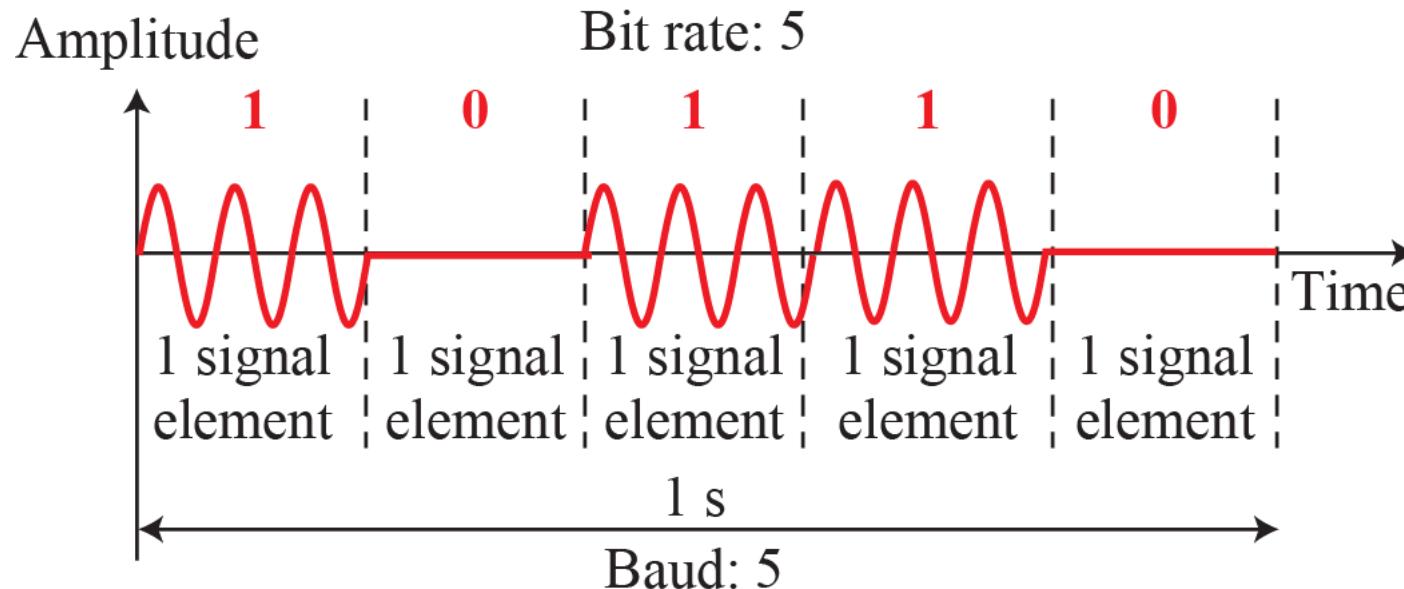
# Analog Transmission



# Analog Encoding

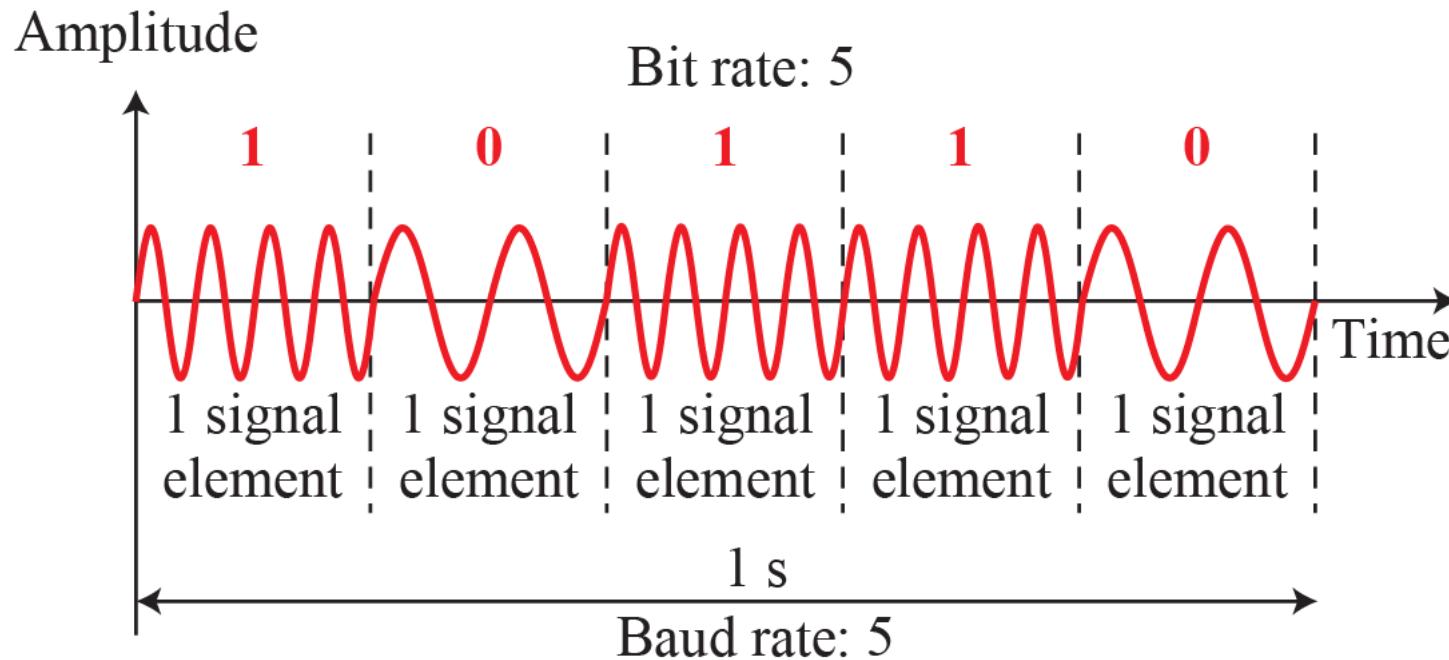
- ❖ To transmit 0s and 1s with analog signal, we can change  $A$ ,  $f$ , or  $\phi$ .
- ❖ **Amplitude Shift Keying (ASK)** changes peak amplitude ( $A$ ) to represent 0s and 1s.
- ❖ **Frequency Shift Keying (FSK)** changes frequency ( $f$ ) to represent 0s and 1s.
- ❖ **Phase Shift Keying (PSK)** changes phase ( $\phi$ ) to represent 0s and 1s.

# Amplitude Shift Keying (ASK)



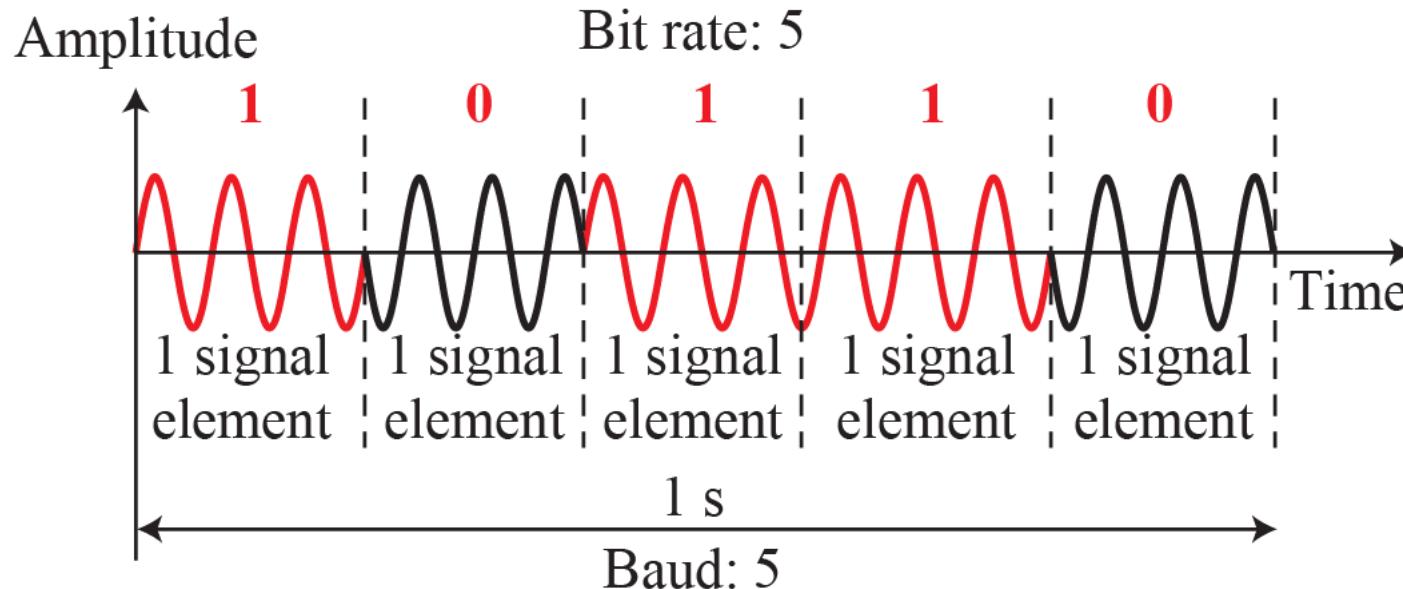
- ❖ Peak amplitude of the signal varies with data values.
- ❖ ASK is susceptible to noise.

# Frequency Shift Keying (FSK)



- ❖ Amplitude and phase remain constant.
- ❖ FSK is limited by the bandwidth of the channel.

# Phase Shift Keying (PSK)

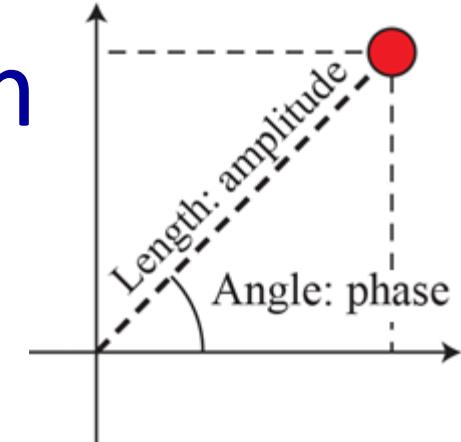
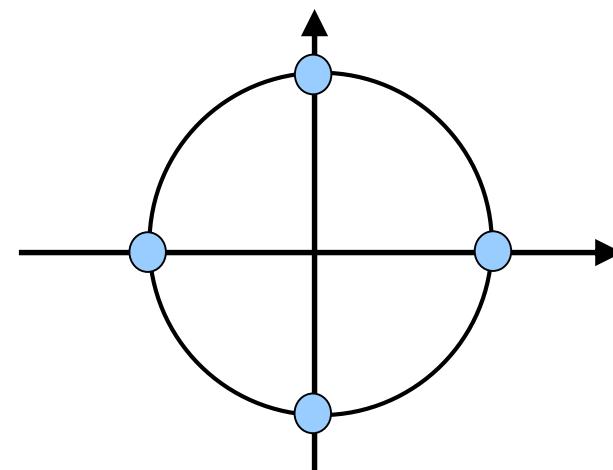


- ❖ One signal element with phase  $0^0$
- ❖ Another with phase  $180^0$

# QPSK Constellation Diagram

- ❖ Can we transmit faster?
  - Send signals with 4 possible phases:

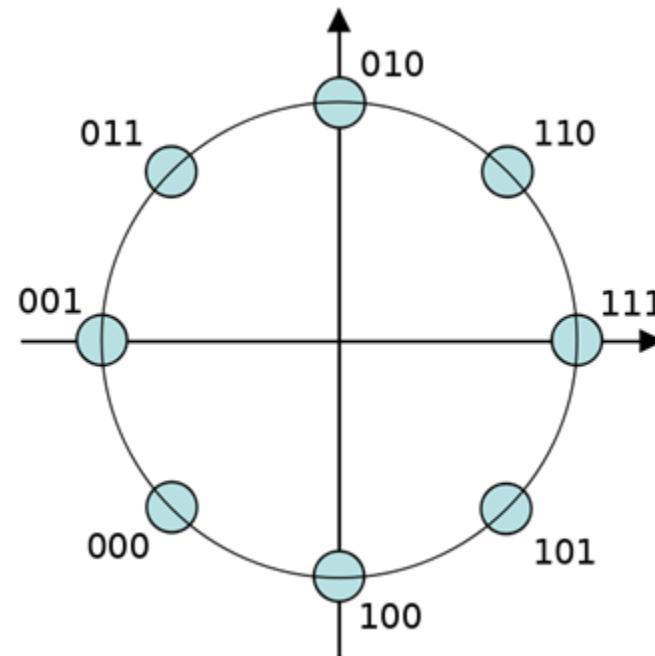
| Phase       | Values represent |
|-------------|------------------|
| $0^\circ$   | <b>11</b>        |
| $90^\circ$  | <b>01</b>        |
| $180^\circ$ | <b>00</b>        |
| $270^\circ$ | <b>10</b>        |



- ❖ Now every signal tells receiver **2** bits of data!

# 8-PSK Constellation Diagram

- ❖ Let's use more phases to carry more data over every signal.



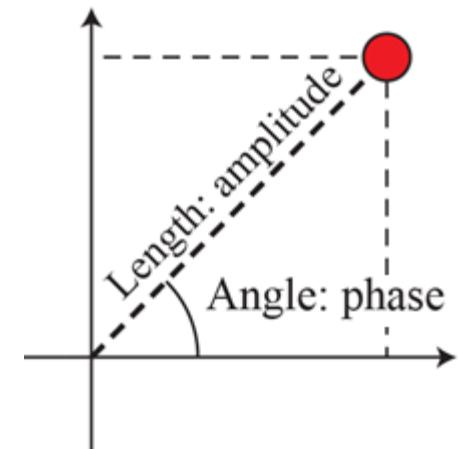
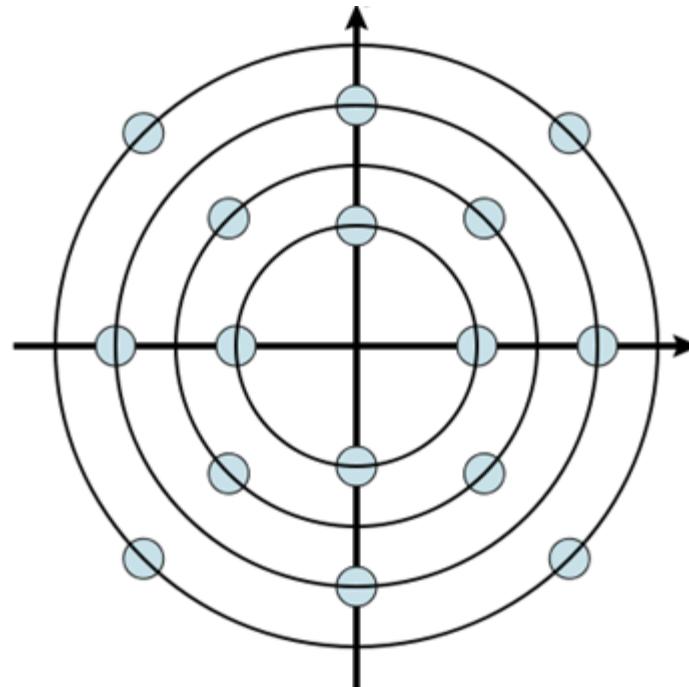
- ❖ Now every signal tells receiver **3** bits of data!

# QAM

- ❖ Can we transfer even faster?
- ❖ Quadrature Amplitude Modulation (QAM) combines ASK and PSK. Many combinations are possible.
  - A signal unit in a  $2^k$ -QAM scheme is a combination of amplitude and phase that represents  $k$  bits.
  - Baud rate is the number of signal units per second.
  - Bit rate is the number of bits receiver receives per second.

# 16-QAM

- ❖ 16-QAM: 16 different signal elements.
- ❖ Every signal differs in either **amplitude** or **phase**.
  - Receiver checks both to determine the data carried by a signal.



# Summary

- ❖ Wi-Fi transmits analog signal and Ethernet transmits digital signal.
- ❖ Ethernet, RFID, and NFC use **Manchester coding**.
- ❖ USB uses **NRZ-I**.
- ❖ Singapore TV broadcast uses DVB-T, which uses **QPSK, 16-QAM, or 64-QAM**.
- ❖ Wi-Fi uses **PSK, QPSK, 16-QAM or 64-QAM**.

# Lecture 11: Roadmap

1. Digital transmission
2. Analog transmission
3. A quick revision
4. Exam matters

# A day in the life of a web request

- ❖ You enter Programming Lab 5, turn on a PC and want to visit [www.facebook.com](http://www.facebook.com).
  - Let's sketch out the steps and protocols involved in such a seemingly simple scenario.
  - Some details are omitted and can be referred to from previous lecture notes ☺

# A day in the life of a web request

## ❖ Step 1:



- On start-up, your PC needs an IP → from **DHCP server**
  - **DHCP request** encapsulated in **UDP segment**, then in **IP datagram**, then in **Ethernet frame**.
  - Frame is broadcasted on subnet.
- DHCP server receives and processes this frame, starts negotiation with your PC for IP.
- Intermediate switches learn your position when forwarding your frames.



Details in  
lecture 6  
notes

Details in  
lecture 10  
notes

# A day in the life of a web request

## ❖ Step 2:

- DHCP server also tells you IP addresses of **first-hop router** and **local DNS server**.
- After you type **www.facebook.com**, browser needs to know IP of this website → from local **DNS server**

Details in  
lecture 10  
notes

- To know the MAC address of local DNS server, PC broadcasts **ARP query**. Local DNS server replies with its MAC address.

- **DNS query** encapsulated in **UDP segment**, then in **IP datagram**, then in **Ethernet frame**, sent to local DNS server.

Details in  
lecture 2  
notes

- Local DNS server reply your PC with IP of Facebook.

# A day in the life of a web request

## ❖ Step 3:

- PC sends **HTTP request** to Facebook.



Details in lecture 3,  
5 notes

- TCP socket opened; 3-way handshake with Facebook server.

- http messages exchanged after TCP connection setup

- Frames sent to first-hop router.

Details in lecture 2 notes

- IP datagrams forwarded from campus network to ISP SingNet.

Details in lecture 7 notes

- Private IP translated by NUS NAT router.

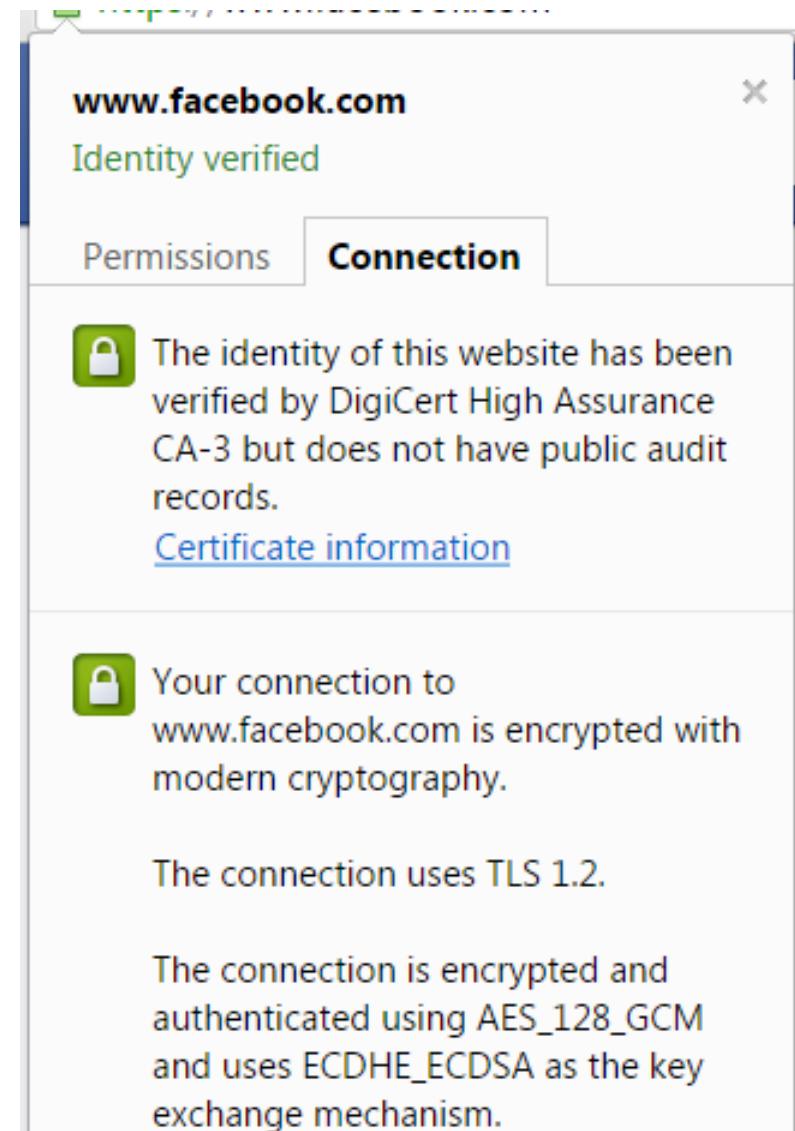
- IP Datagram routed on the Internet using RIP or other protocols.

# A day in the life of a web request

## ❖ Step 4:

- When Facebook is contacted
  - Negotiate for secure connection.
  - $\text{https} = \text{http} + \text{SSL/TLS}$
  - Digital certificate of Facebook verified.
  - Message encryption and authentication

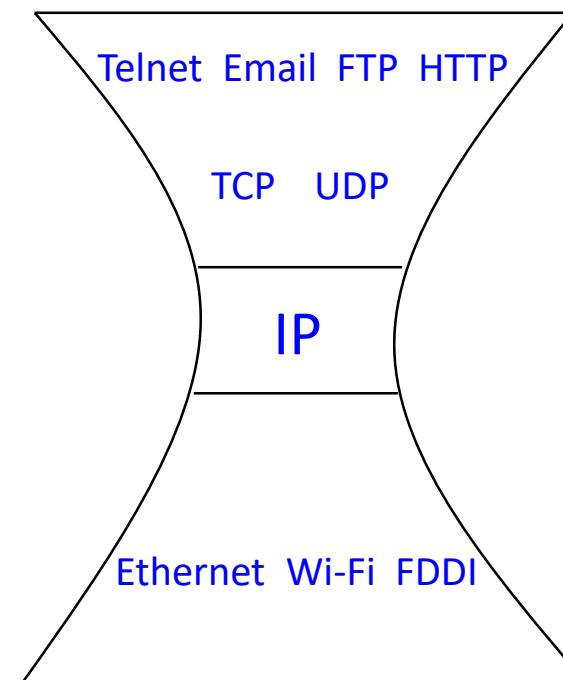
Details in  
lecture 8  
notes



# Lessons from CS2105

- ❖ Network system is so complex!
  - There are many issues to consider, to support different applications running on large number of hosts through different access technology and physical media.

- ❖ To deal with complexity:
  - separation of concerns
  - 5 protocol layer



# Lecture 11: Roadmap

1. Digital transmission
2. Analog transmission
3. A quick revision
4. Exam matters

# Tips for Exam

## ❖ Preparation

- Review lecture notes and tutorial questions for basic concepts and skills.
- Focus on **understanding** rather than **memorization**.

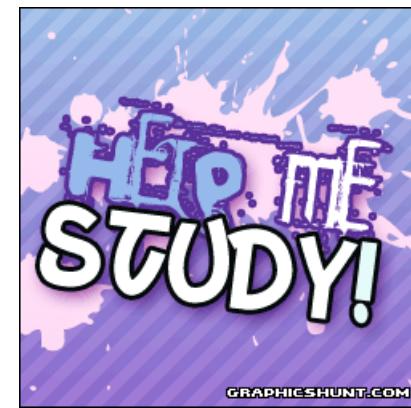
## ❖ During exam

- Read instructions carefully
- A calculator will be helpful



# Consultation

- ❖ Email and Microsoft Teams
  - Anytime
  - Email: ebramkw@aun.edu.eg
  
- ❖ Office hour
  - Upon email appointment
  - Office: Labs 3rd Story



# Thank you!

