

Announcement & Reminder (Sep 18)

- Hw #1: due: Friday Oct 6 11:59pm
 - submission online using the class Moodle site

Please start working on them!

- Next week: Programming Project # 1 will be handed out!

Due Monday Oct 15 11:59pm

We Have Learned Last Time

Weekly Summary

What We Learned Last Time (Sep 11)

- ❖ What is a network? What is a computer/data network?
Compare w/ diff. networks (telephone, postal office, transportation networks, ...)
 - “bolts-&nuts” view vs. service perspective
- ❖ Network is a *shared* resource!
 - ways to share (“multiplex”) resources? TDMA, FDMA, CDMA, ...
- ❖ **Packet switching** vs. circuit switching:
 - **packets, packet switching and statistical multiplexing**
 - For “bursty” data applications
 - store-&-forward
 - delay, losses and congestion vs. call blocking
- ❖ **New: four types of delays**
 - propagation, transmission, processing & queueing delays
- ❖ **New: Architecture: layering & hourglass**
 - ❖ different technologies, “boxes” (routers, switches), & apps
 - ❖ **Protocols and Interfaces (API)**

Switching & Multiplexing

- Network is a **shared** resource
 - Provide services for many people at same time
 - Carry bits/information for many people at same time
- How do we do it?
 - **Switching**: how to deliver information from point A to point B?
 - **Multiplexing**: how to share resources among many users

Think about postal service and telephone system!

Switching and multiplexing are closely related!

Switching/Multiplexing Strategies

- Circuit switching
 - set up a dedicated route (“circuit”) first
 - carry all bits of a “conversation” on one circuit
 - original telephone network
 - Analogy: railroads and trains/subways
- Packet switching
 - divide information into small chunks (“packets”)
 - each packet delivered independently
 - “store-and-forward” packets
 - Internet

(also Postal Service, but they don’t tear your mail into pieces first!)

 - Analogy: highways and cars
- Pros and Cons?
 - think taking subways vs. driving cars, during off-peak vs. rush hours!

Analogy: railroad and train

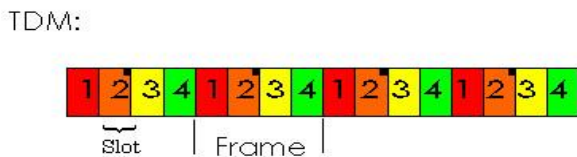
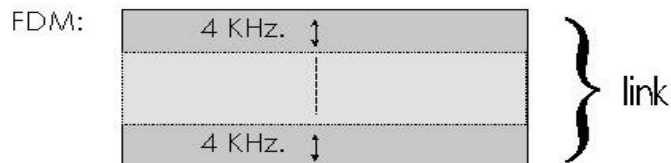


Analogy: Highway and cars



Circuit Switching

- network resources
(e.g., bandwidth)
divided into "pieces"
- pieces allocated to calls
 - resource piece **idle** if not used by owning call
(no sharing)



— All slots labelled  are dedicated to a specific sender-receiver pair.

- dividing link bandwidth into "pieces"
 - ❖ frequency division
 - ❖ time division
 - ❖ code division

□ Trivia Q:

You must have heard of the term "CDMA" (think the company Qualcomm, for which it is most associated with), what does "CD" in CDMA stand for?

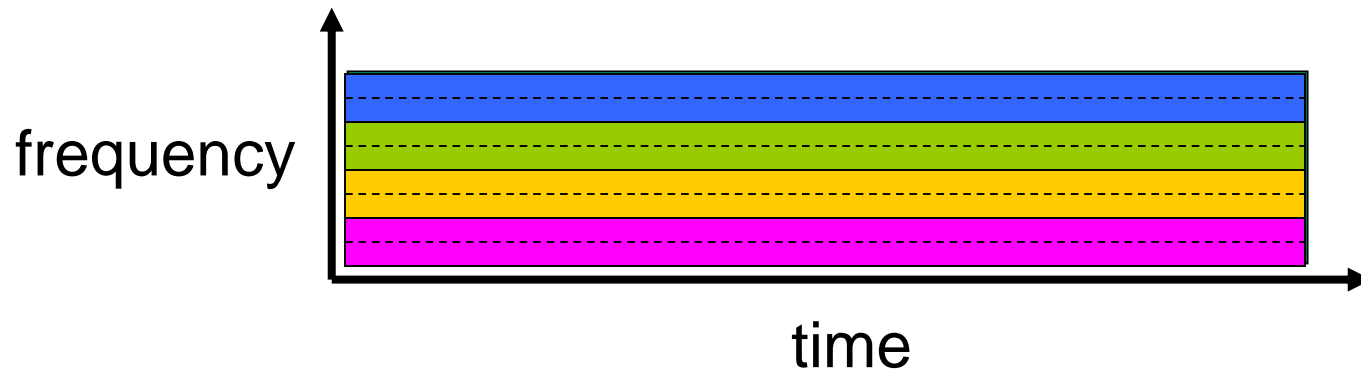
Circuit Switching: FDM and TDM

Example:

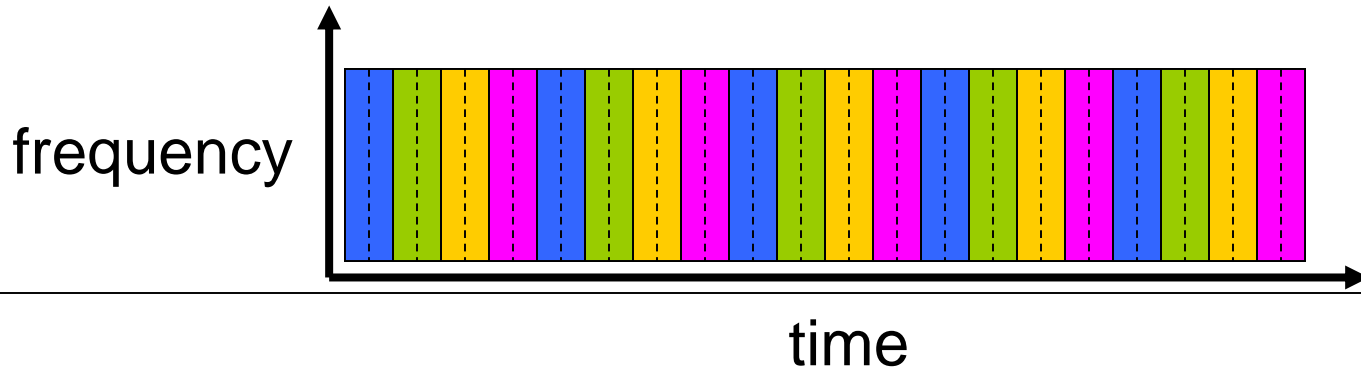
4 users



FDM



TDM

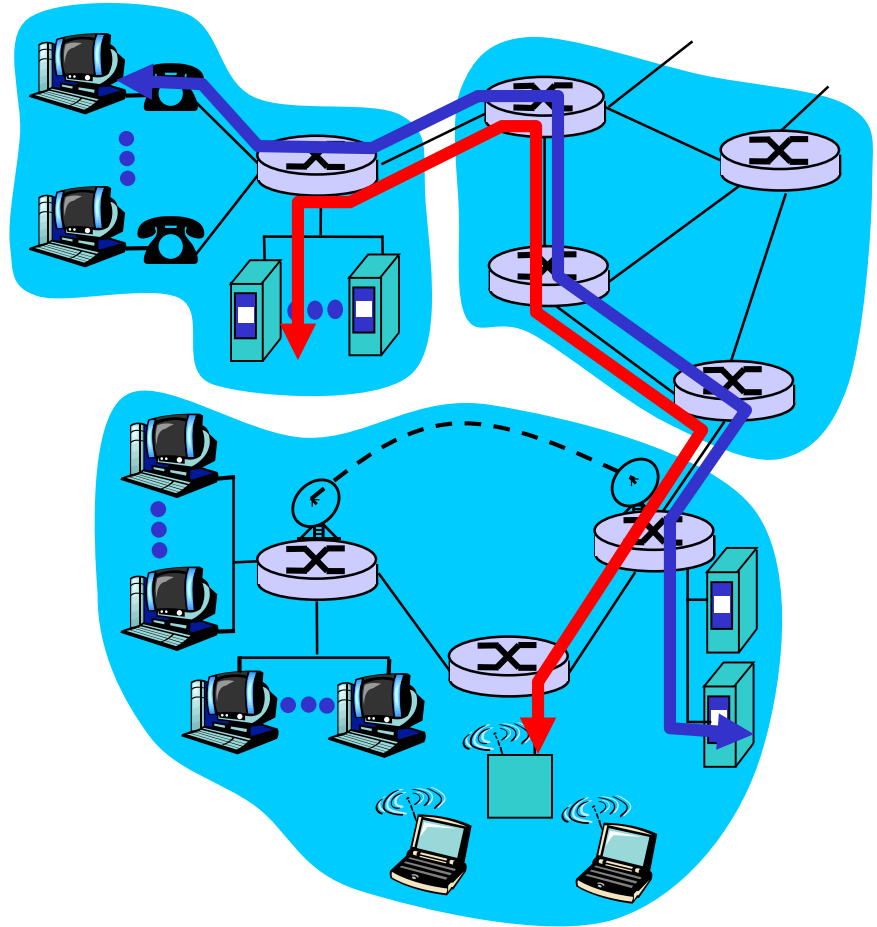


Networks with Circuit Switching

e.g., conventional (fixed-line) telephone networks

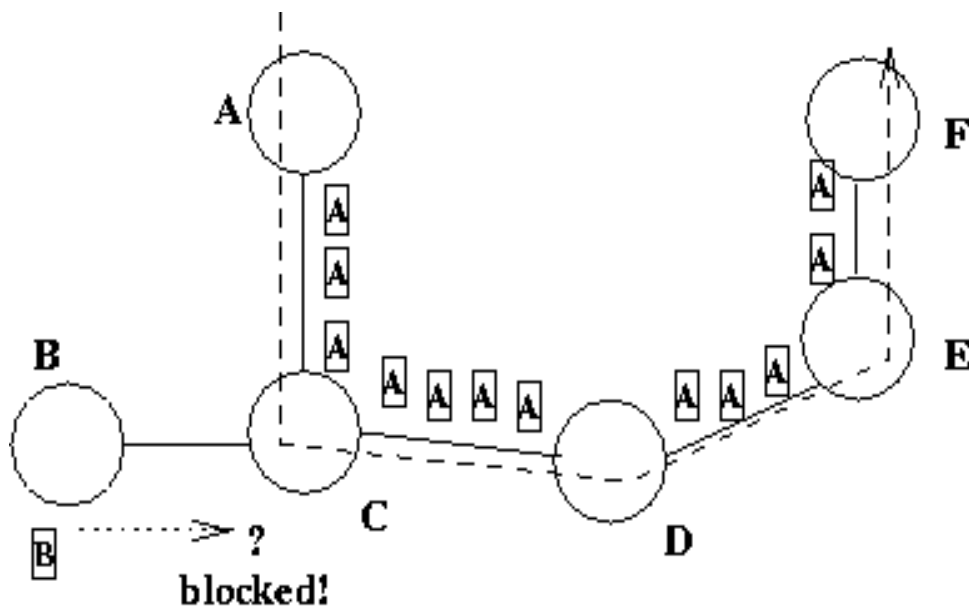
End-end resources reserved for "call"

- link bandwidth, switch capacity
- dedicated resources: no sharing
- circuit-like (guaranteed) performance
- call setup required



Circuit Switched Networks

- All resources (e.g. communication links) needed by a call dedicated to that call for its duration
 - Example: telephone network
 - Call blocking when all resources are used



Note: if one circuit/link, A-to-F call blocks B-to-E call

Numerical example

- How long does it take to send a file of 640,000 bits from host A to host B over a circuit-switched network?
 - All links are 1.536 Mbps
 - Each link uses TDM with 24 slots/sec
 - 500 msec to establish end-to-end circuit

Let's work it out!

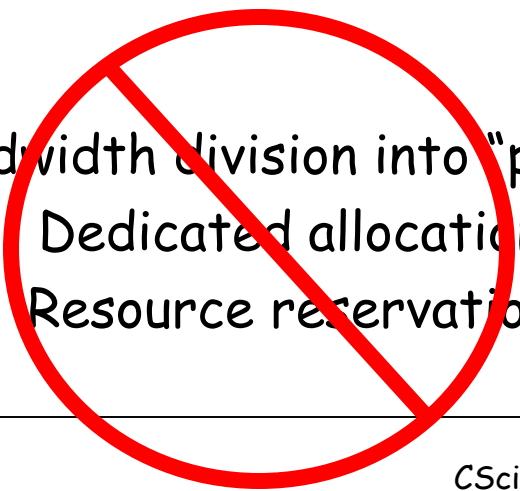
10.5 seconds

Packet Switching

Each end-end "data stream"
divided into *packets*

- users A, B packets share network resources
- each packet uses full link bandwidth
- resources used *as needed*

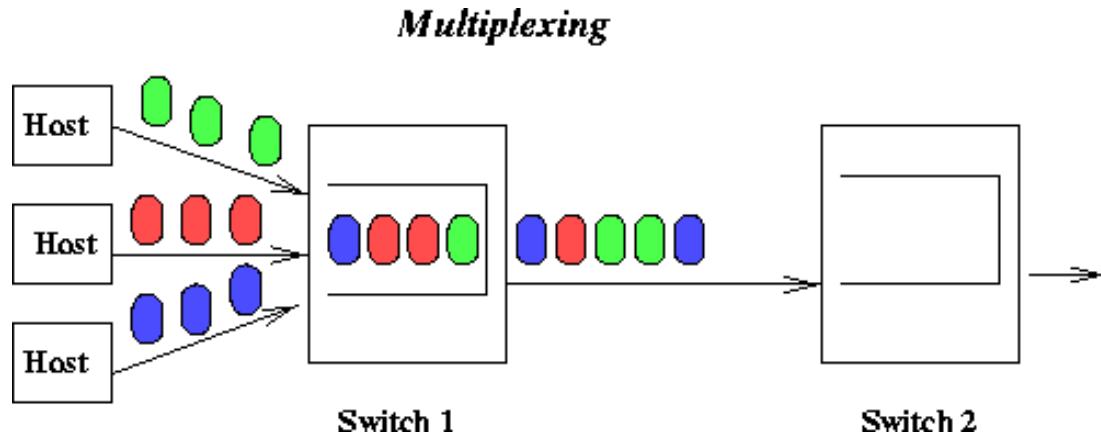
Bandwidth division into "pieces"
Dedicated allocation
Resource reservation



resource contention:

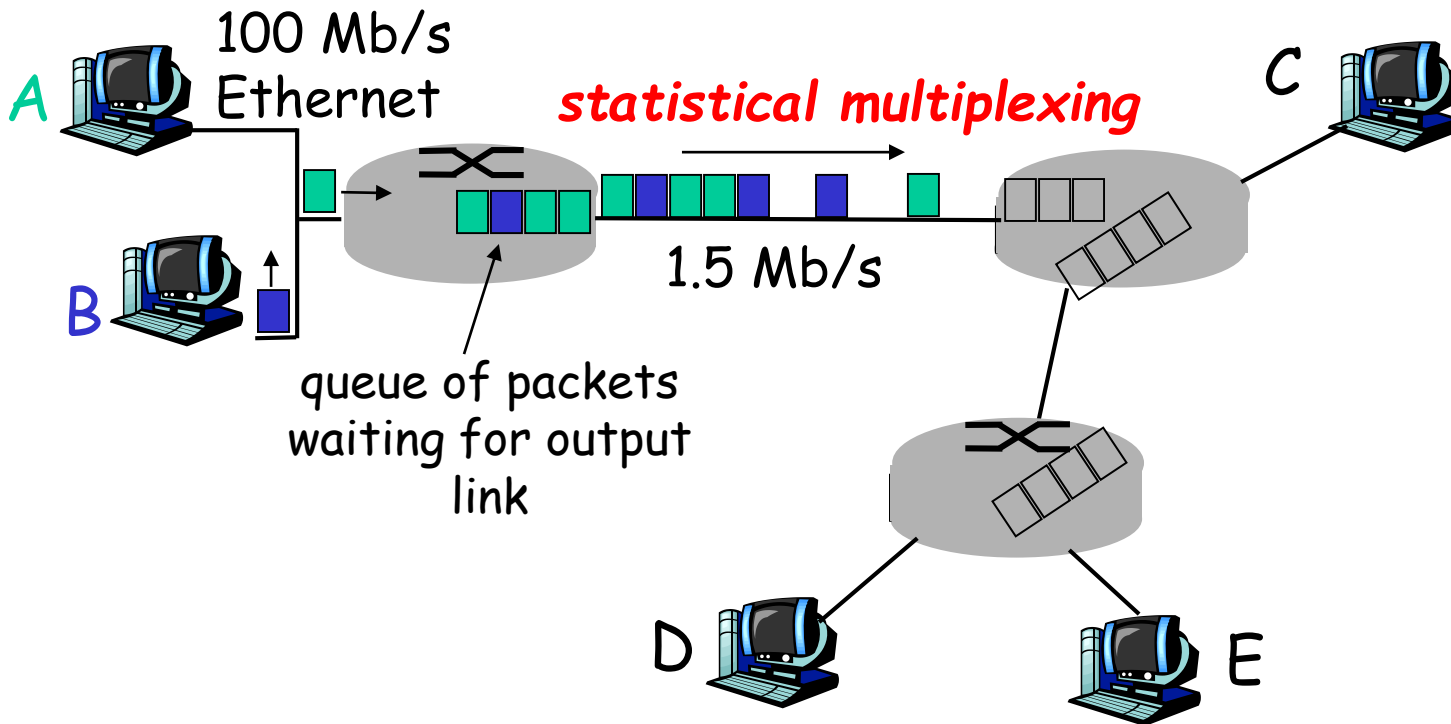
- ❑ aggregate resource demand can exceed amount available
- ❑ congestion: packets queue, wait for link use
- ❑ store and forward: packets move one hop at a time
 - ❖ Node receives complete packet before forwarding
 - ❖ *Packets may suffer delay or losses!*

Statistical Multiplexing



- Time division, but **on demand** rather than fixed
- Reschedule link on a per-packet basis
- Packets from different sources interleaved on the link
- Buffer packets that are **contending** for the link
- Buffer buildup is called **congestion**
- This is **packet switching**, used in computer networks

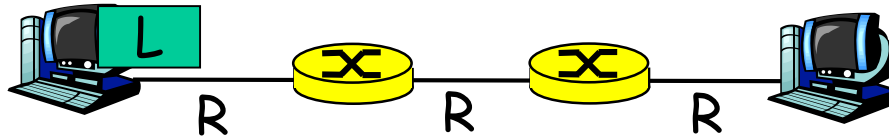
Packet Switching: Statistical Multiplexing



Sequence of A & B packets does not have fixed pattern, shared on demand ☐ *statistical multiplexing*.

TDM: each host gets same slot in revolving TDM frame.

Packet-switching: store-and-forward



- Takes L/R seconds to transmit (push out) packet of L bits on to link of R bps
- Entire packet must arrive at router before it can be transmitted on next link: *store and forward*
- delay = $3L/R$ (assuming zero propagation delay)

Example:

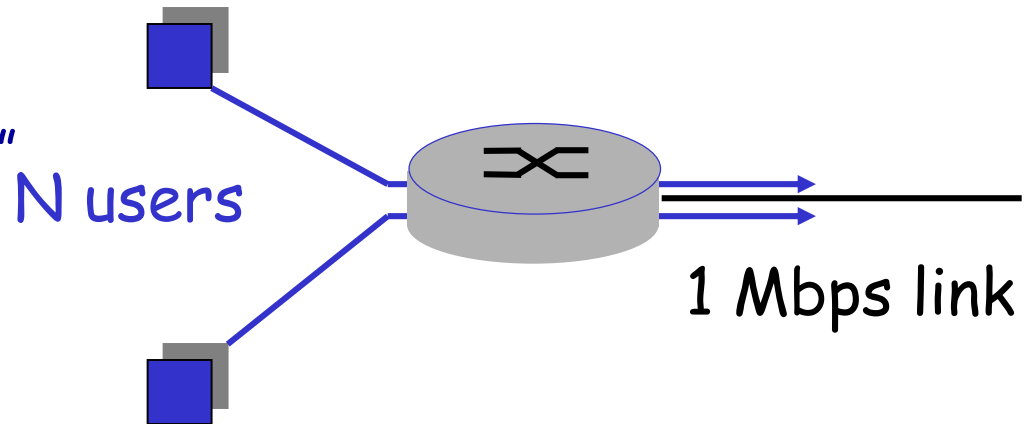
- $L = 7.5$ Mbits
- $R = 1.5$ Mbps
- delay = ? 15 sec

} more on delay later ...

Packet switching versus circuit switching

Packet switching allows more users to use network!

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



- circuit-switching:
 - 10 users
- packet switching:
 - with 35 users,
probability > 10 active
less than .0004

Q: how did we get value 0.0004?

$$\sum_{n=N+1}^M \binom{M}{n} p^n (1-p)^{M-n}$$

Circuit Switching vs Packet Switching

Item	Circuit-switched	Packet-switched
Dedicated “copper” path	Yes	No
Bandwidth available	Fixed	Dynamic
Potentially wasted bandwidth	Yes	No (not really!)
Store-and-forward transmission	No	Yes
Each packet/bit always follows the same route	Yes	Not necessarily
Call setup	Required	Not Needed
When can congestion occur	At setup time	On every packet
Effect of congestion	Call blocking	Queuing delay

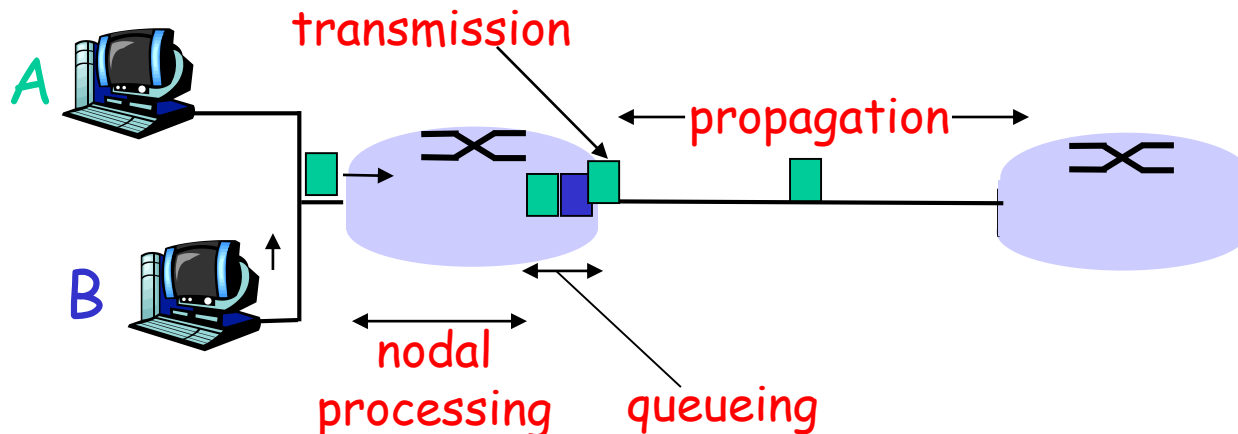
Four sources of packet delay

1. nodal processing:

- check bit errors
- determine output link

2. queueing

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



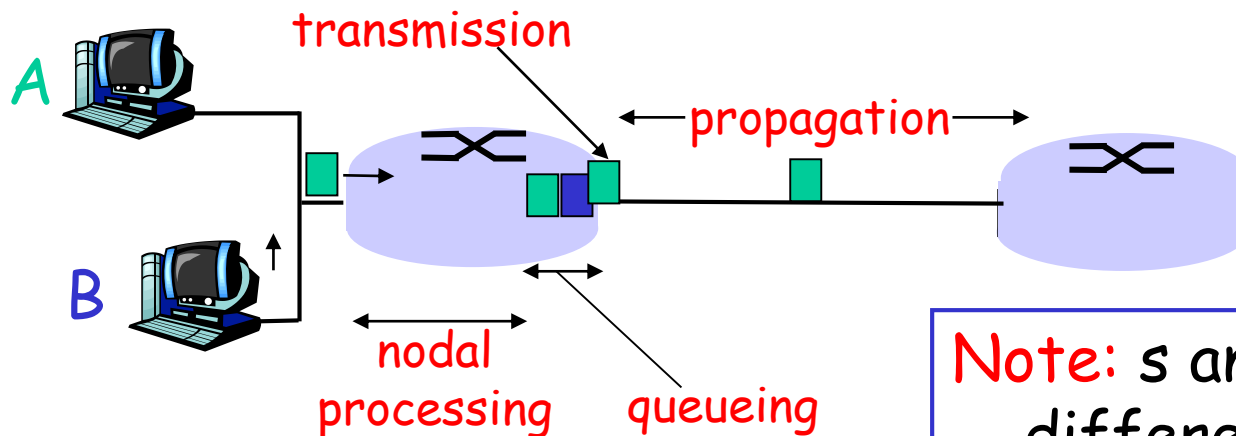
Delay in packet-switched networks

3. Transmission delay:

- R = link bandwidth (bps)
- L = packet length (bits)
- time to send bits into link = L/R

4. Propagation delay:

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



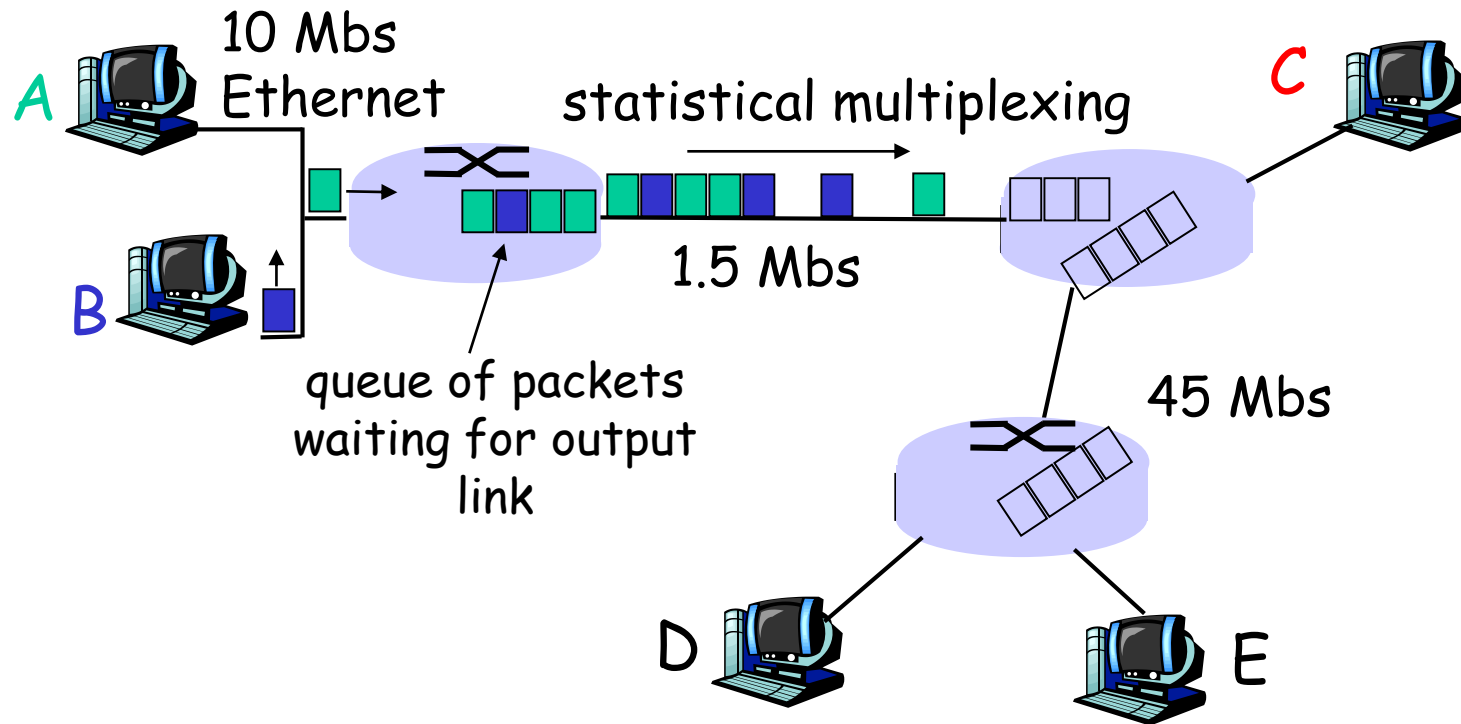
Note: s and R are very different quantities!

Nodal delay

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

- d_{proc} = processing delay
 - typically a few microsecs or less
- d_{queue} = queuing delay
 - depends on congestion
- d_{trans} = transmission delay
 - $= L/R$, significant for low-speed links
- d_{prop} = propagation delay
 - a few microsecs to hundreds of msecs

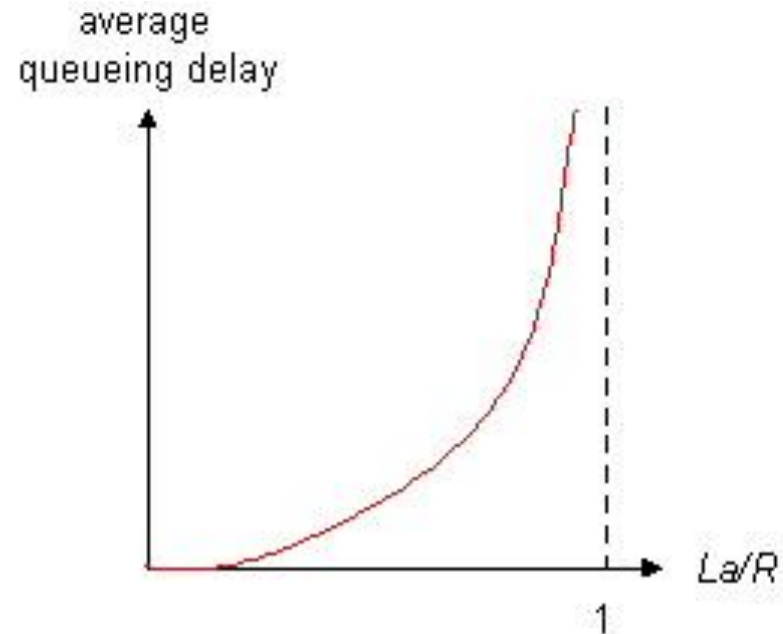
Statistical Multiplexing and Queueing



Queueing delay (revisited)

- R =link bandwidth (bps)
- L =packet length (bits)
- a =average packet arrival rate

traffic intensity = $\lambda a/R$



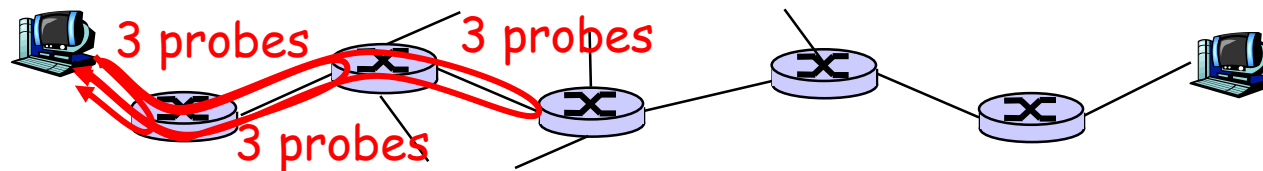
- $\lambda a/R \sim 0$: average queueing delay small
- $\lambda a/R \rightarrow 1$: delays become large
- $\lambda a/R > 1$: more “work” arriving than can be serviced, average delay infinite!

Queueing delay and Packet loss

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

"Real" Internet delays and routes

- What do "real" Internet delay & loss look like?
- Traceroute program: provides delay measurement from source to router along end-end Internet path towards destination. For all i :
 - sends three packets that will reach router i on path towards destination
 - router i will return packets to sender
 - sender times interval between transmission and reply.

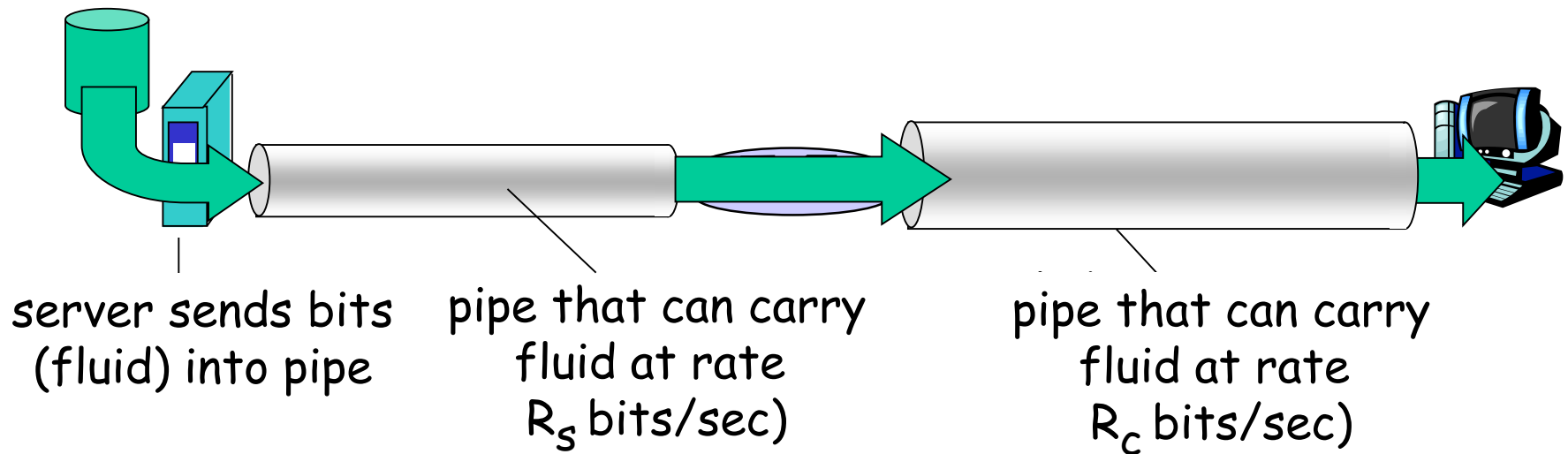


"Real" Internet delays and routes

Let's Traceroute to www.bbc.com

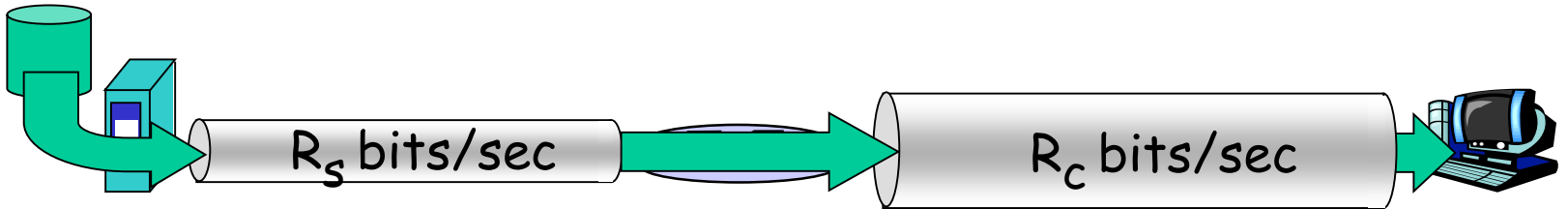
Throughput

- *throughput*: rate (bits/time unit) at which bits transferred between sender/receiver
 - *instantaneous*: rate at given point in time
 - *average*: rate over longer period of time

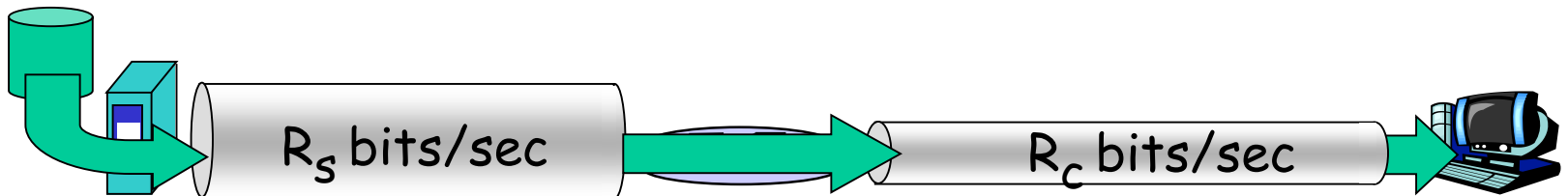


Throughput (cont'd)

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



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

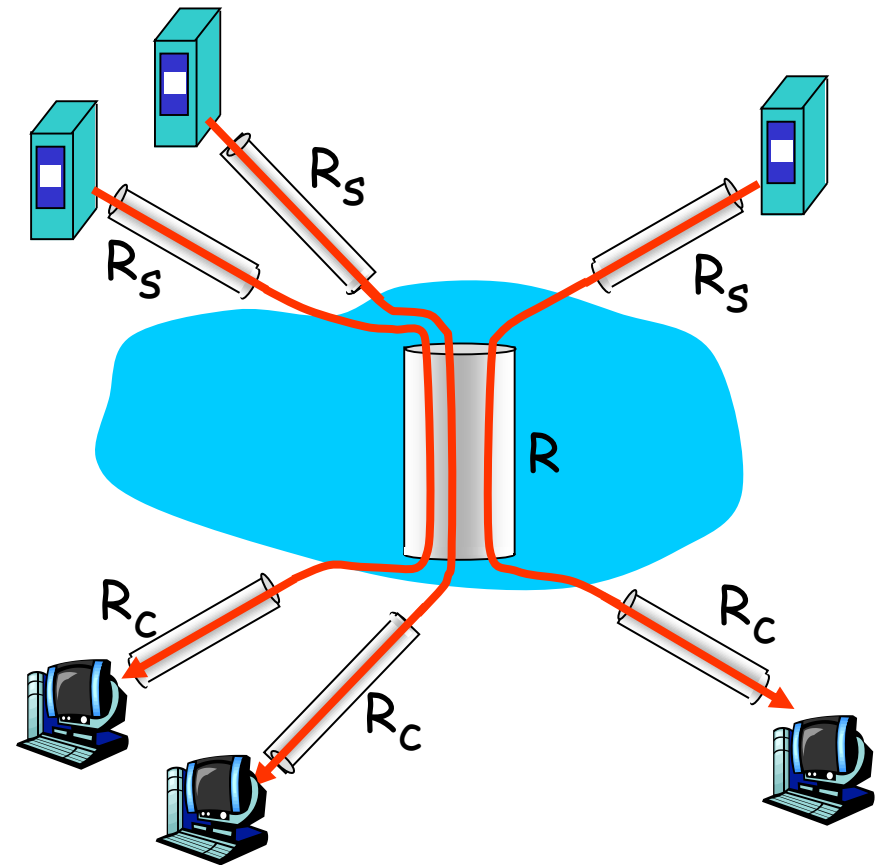


bottleneck link

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

Throughput: Internet scenario

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



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

Questions?

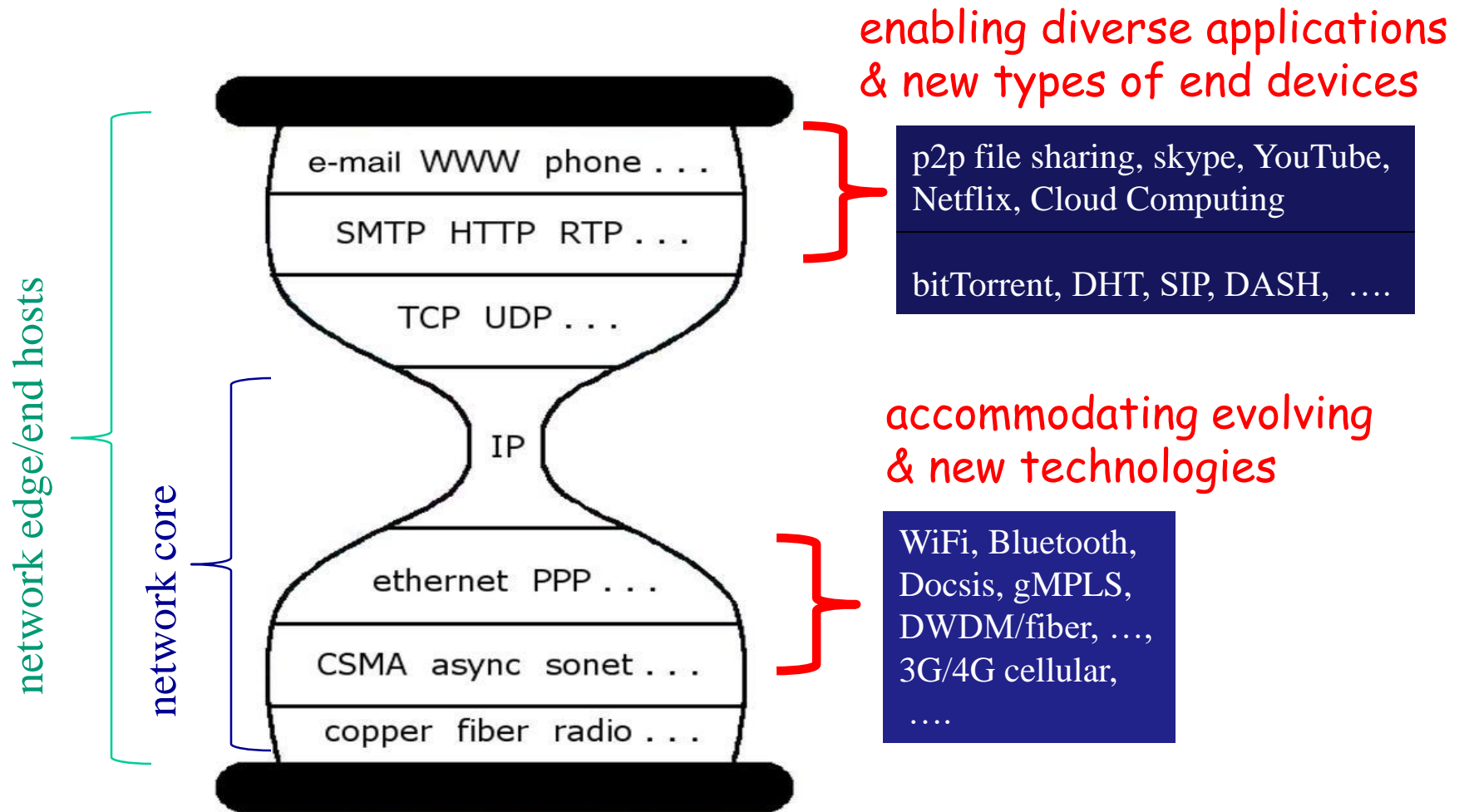
Introduction (cont'd)

- ❖ **Key network functions:**
 - naming, addressing, routing & forwarding
 - ❖ networks are distributed & complex systems!
- ❖ **What's so special about the Internet?**
 - Internet Architecture: layering & hourglass
 - ❖ different technologies, “boxes” (routers, switches), & apps
- ❖ **Protocols and Interfaces (API)**
- ❖ *What may go wrong?*
 - ❖ bit errors, packet losses,, node failures, software bugs, app crashes. and **Attacks !!!**
- ❖ What today' s Internet looks like? economics & policies

What's so special about the Internet?

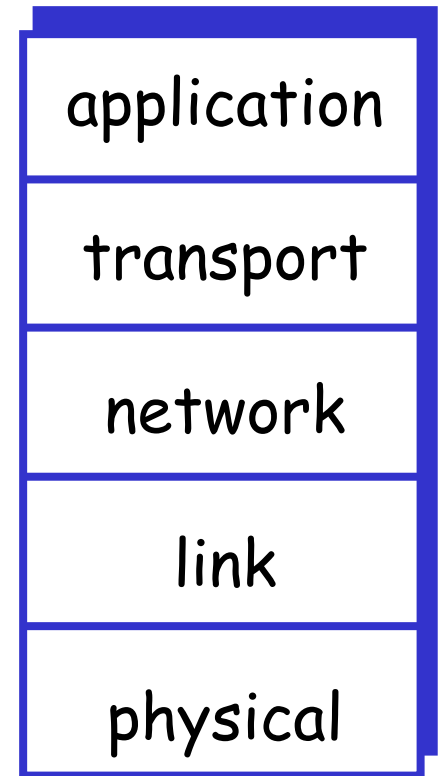
- Internet is based on the notion of “**packet switching**”
 - enables **statistical multiplexing**
 - better utilization of network resources for transfer of “bursty” data traffic
- Internet's key organizational/architectural principle: “smart” end systems + “dumb” networks
 - architecture: functional division & function placement
 - **hourglass Internet architecture**: enables diverse applications and accommodates evolving technologies
 - “**dumb**” **network (core)**: simple packet-switched, store-forward, connectionless “datagram” service, with core functions: global addressing, routing & forwarding
 - “**smart**” **end systems/edges**: servers, PCs, mobile devices, ...; diverse and ever-emerging new applications!

Internet Hourglass Architecture



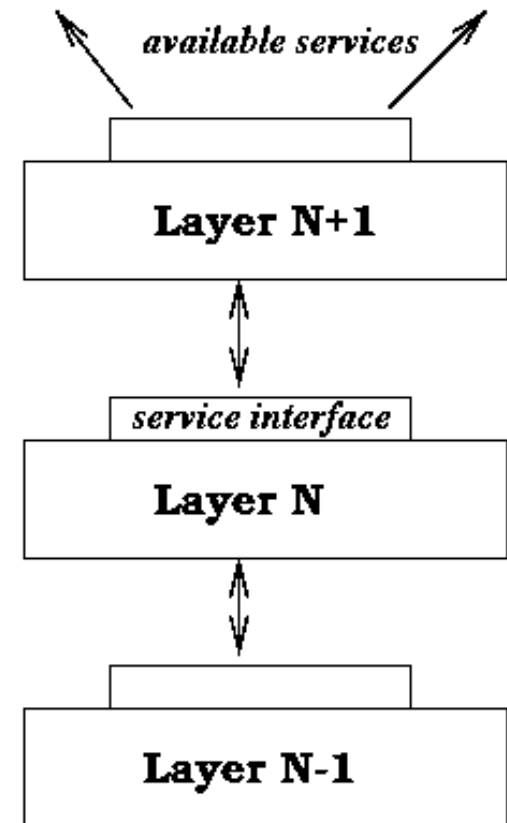
Internet Protocol Stack

- **application:** supporting network applications
 - FTP, SMTP, HTTP, DASH, ...
- **transport:** process-process data transfer
 - TCP, UDP
- **network:** routing of datagrams from source to destination
 - IP, routing protocols
- **link:** data transfer between neighboring network elements
 - PPP, Ethernet
- **physical:** bits "on the wire"



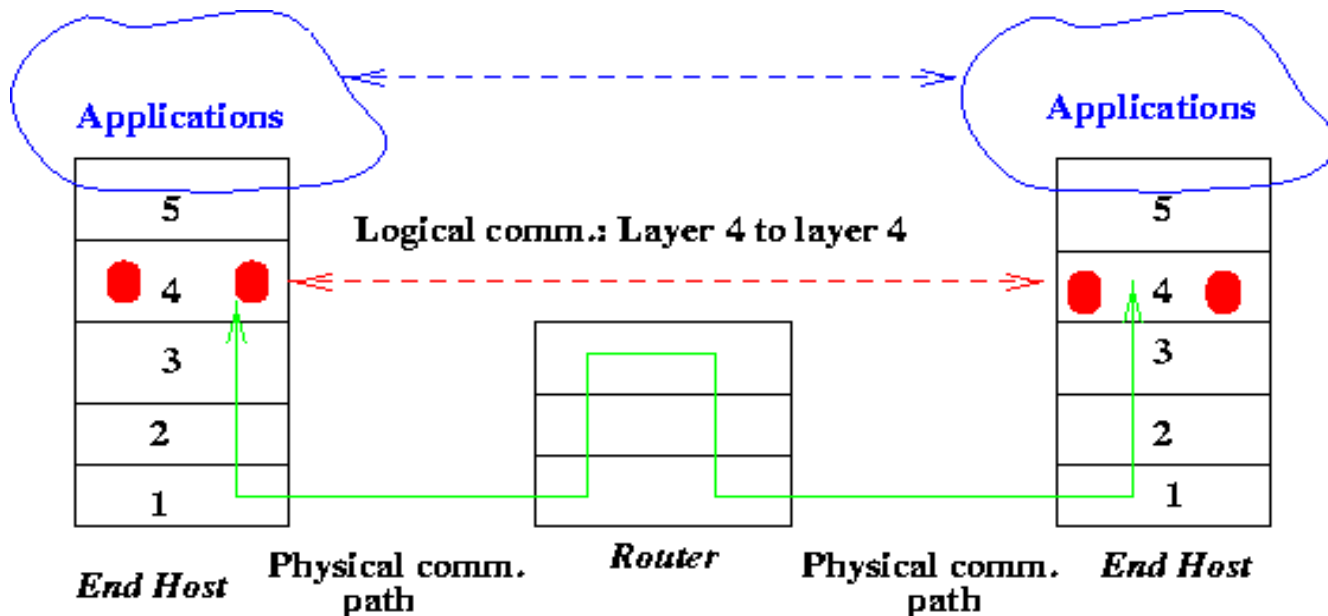
Layered Architecture

- Layering simplifies the architecture of complex system
- Layer N relies on *services* from layer N-1 to provide a *service* to layer N+1
- *Interfaces* define the services offered
- Service required from a lower layer is independent of its implementation
 - Layer N change doesn't affect other layers
 - Information/complexity hiding
 - Similar to object oriented methodology



Protocols and Services

- Protocols are used to implement services
 - Peering entities in layer N provide service by communicating with each other using the service provided by layer N-1
- *Logical vs physical communication*



What's a protocol?

human protocols:

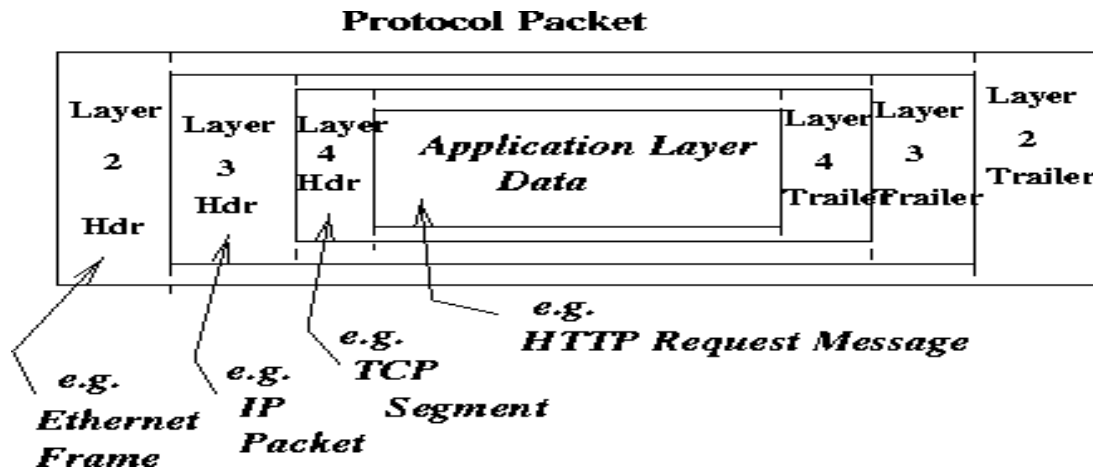
- "what's the time?"
- "I have a question"
- introductions

network protocols:

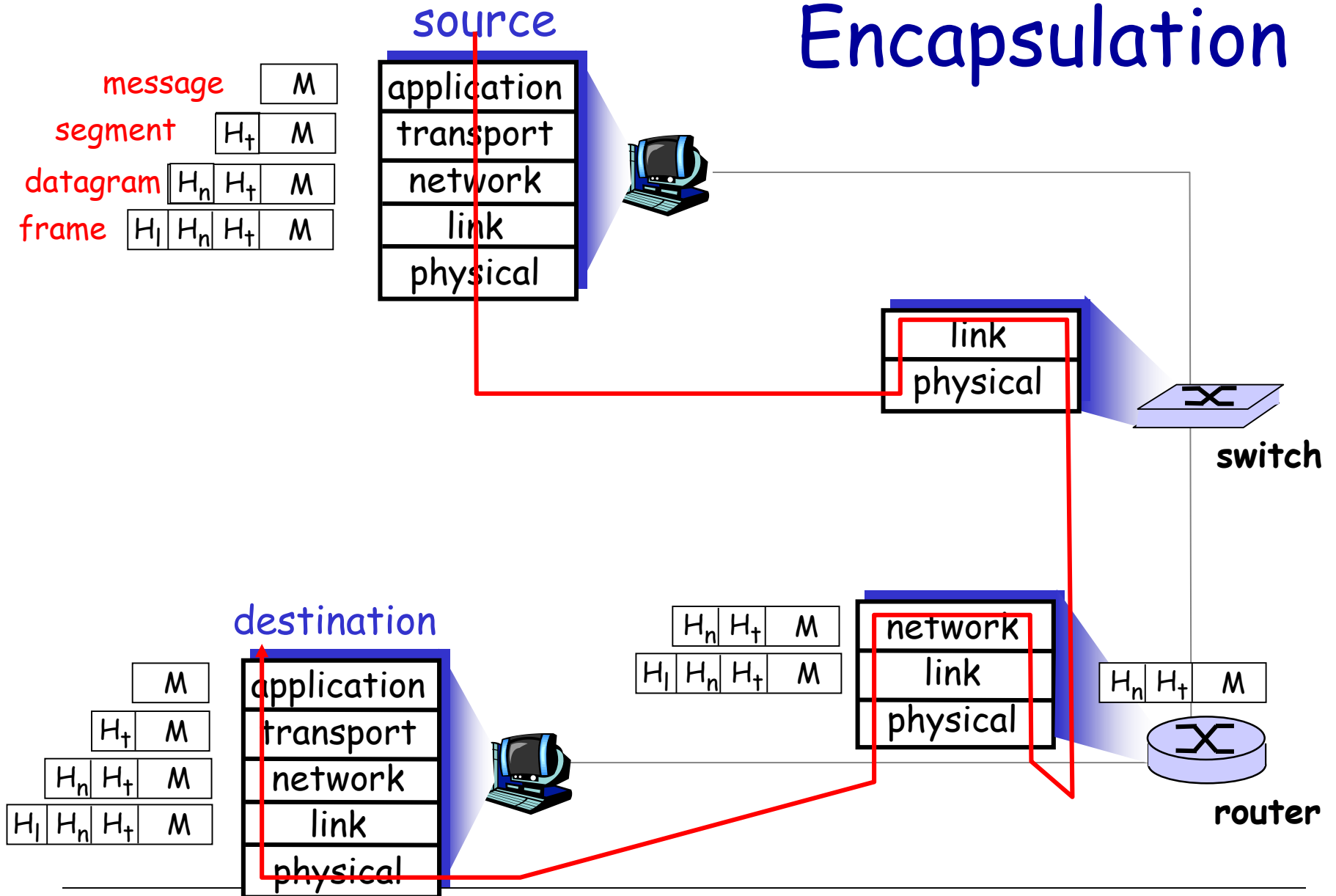
- machines rather than humans
- all communication activity in Internet governed by protocols
(why this concept is so important!!!)

Protocol Packets

- Protocol data units (PDUs):
 - packets exchanged between peer entities
- Service data units (SDUs):
 - packets handed to a layer by an upper layer
- Data at one layer is encapsulated in packet at a lower layer
 - *Envelope within envelope*: PDU = SDU + (optional) header or trailer



Encapsulation



Fundamental Issues in Networking

Network is a shared resource

- Provide services for many people at same time
- Carry bits/information for many people at same time
- Switching and Multiplexing
 - How to share resources among multiple users, and transfer data from one node to another node
- Naming and Addressing
 - How to find name/address of the party (or parties) you would like to communicate with
 - Address: byte-string that identifies a node
 - unicast, multicast and broadcast addresses
- Routing and Switching/Forwarding:
 - process of determining how to send packets towards the destination based on its address: finding out neighbors, building routing tables
 - transferring data from source to destination

Fundamental Problems in Networking ...

Or what can go wrong?

- Bit-level errors: due to electrical interferences
- "Frame-level" errors: media access delay or frame collision due to contention/collision/interference
- Packet-level errors: packet delay or loss due to network congestion/buffer overflow
- Out of order delivery: packets may take different paths
- Link/node failures: cable is cut or system crash
- [and of course, cyber attacks! -> will not be covered in this class: if interested, take "computer security"]

Fundamental Problems in Networking

What can be done?

- Add redundancy to detect and correct erroneous packets
- Acknowledge received packets and retransmit lost packets
- Assign sequence numbers and reorder packets at the receiver
- Sense link/node failures and route around failed links/nodes

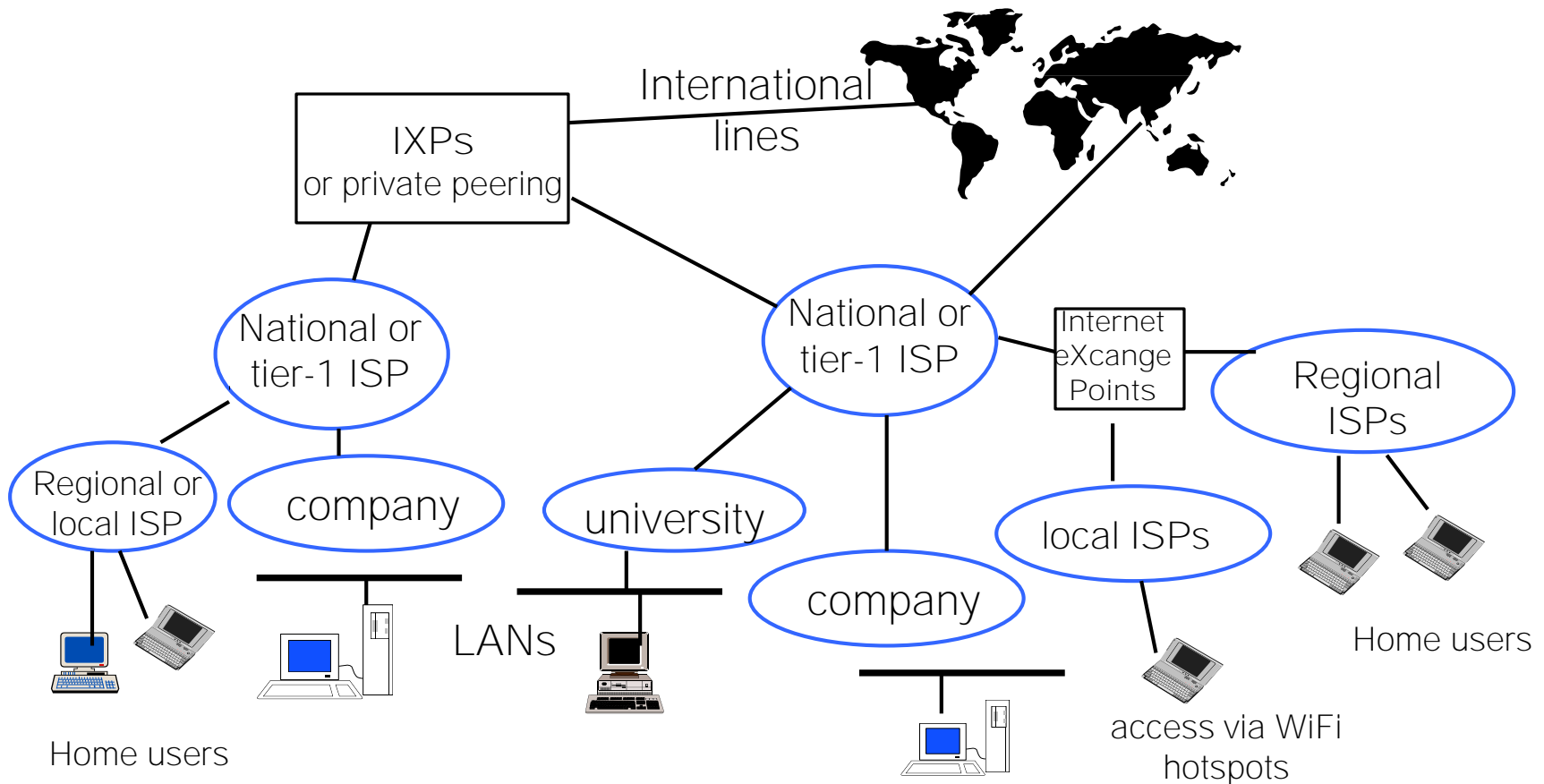
Goal: to fill the gap between what applications expect and what underlying technology provides

Introduction (cont'd)

- ❖ Key network functions:
 - naming, addressing, routing & forwarding
 - ❖ networks are distributed & complex systems!
- ❖ What's so special about the Internet?
 - Internet Architecture: layering & hourglass
 - ❖ different technologies, “boxes” (routers, switches), & apps
- ❖ Protocols and Interfaces (API)
- ❖ What may go wrong?
 - ❖ bit errors, packet losses,, node failures, software bugs, app crashes. and **Attacks !!!**
- ❖ *What today's Internet looks like? economics & policies*

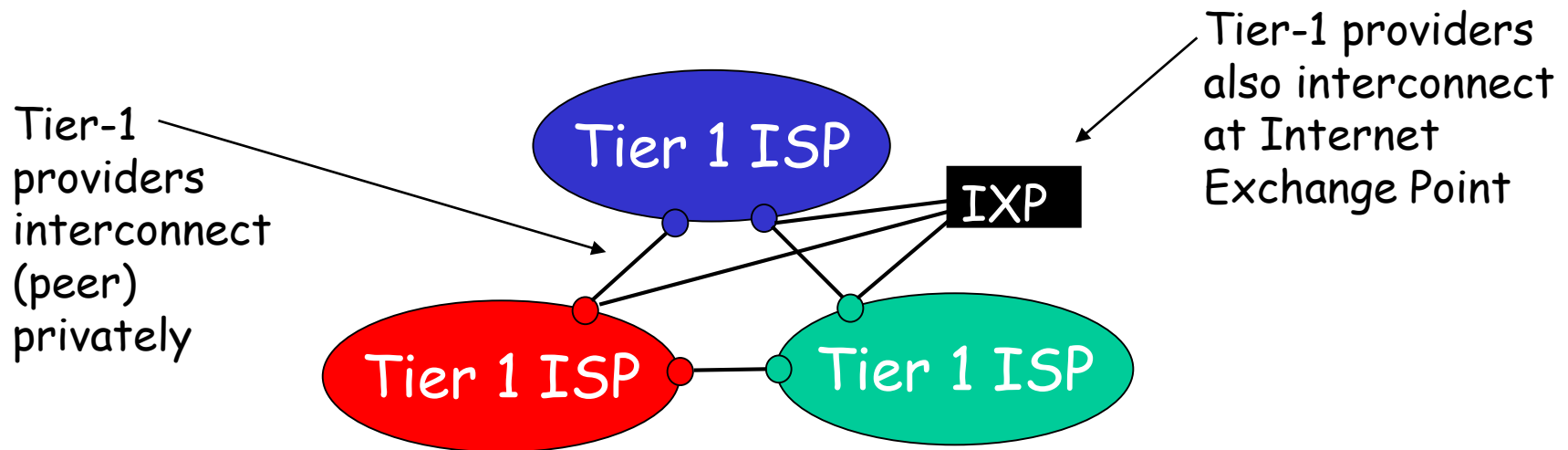
Internet Structure

Internet: “networks of networks”!

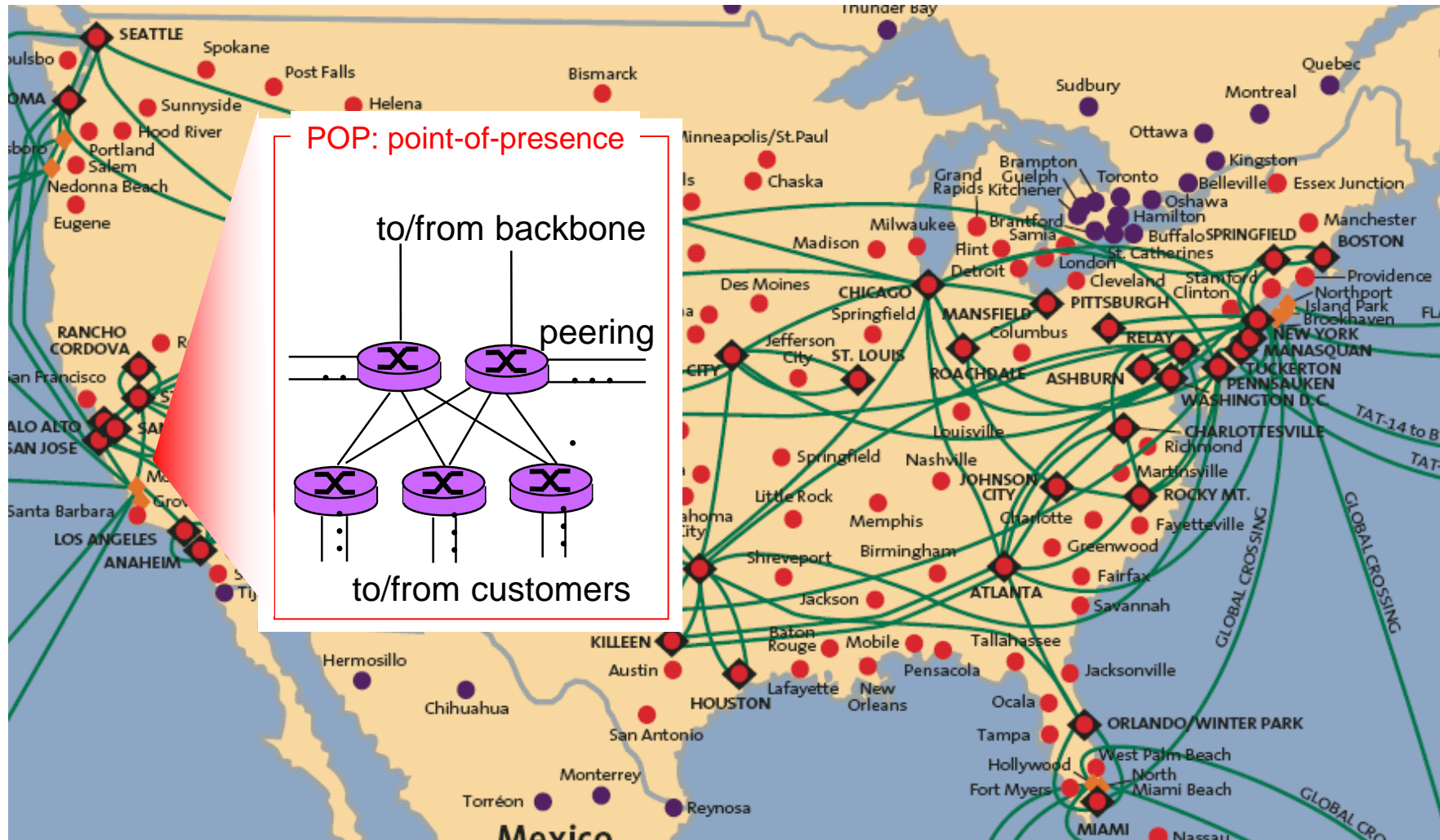


Internet structure: network of networks

- Roughly hierarchical
- **At center: "tier-1" ISPs** (e.g., Verizon, Sprint, AT&T, L3, Cable and Wireless), national/international coverage
 - treat each other as equals

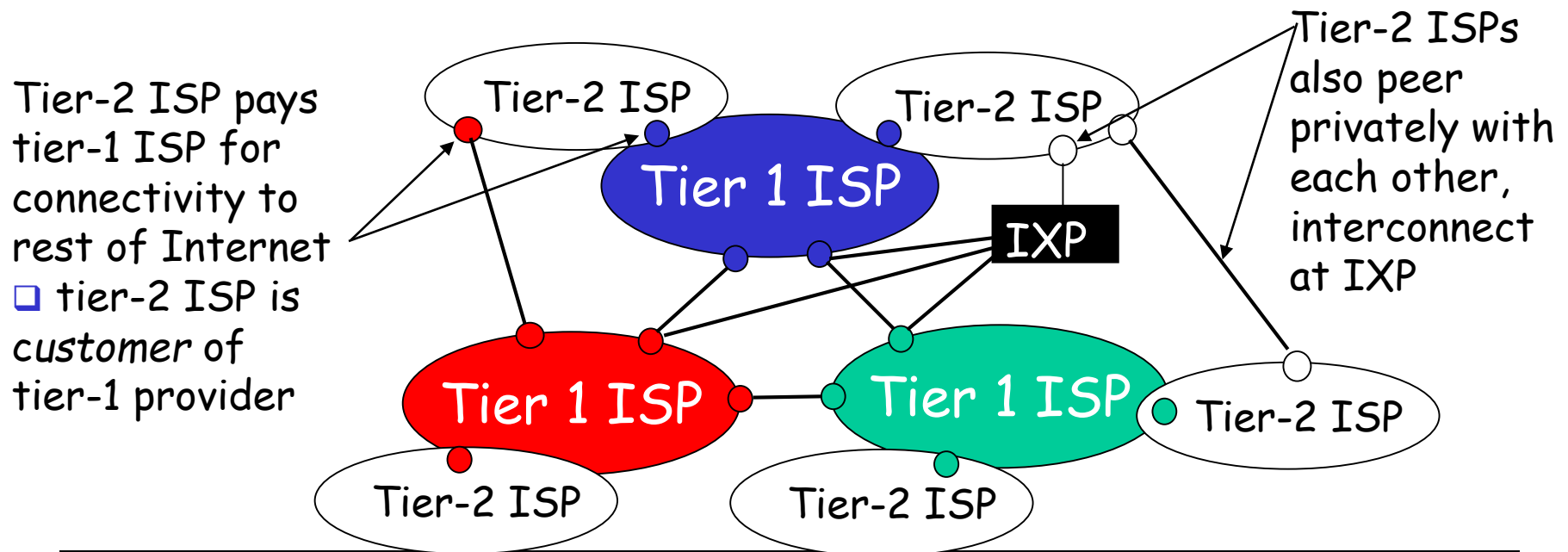


Tier-1 ISP: e.g., Sprint



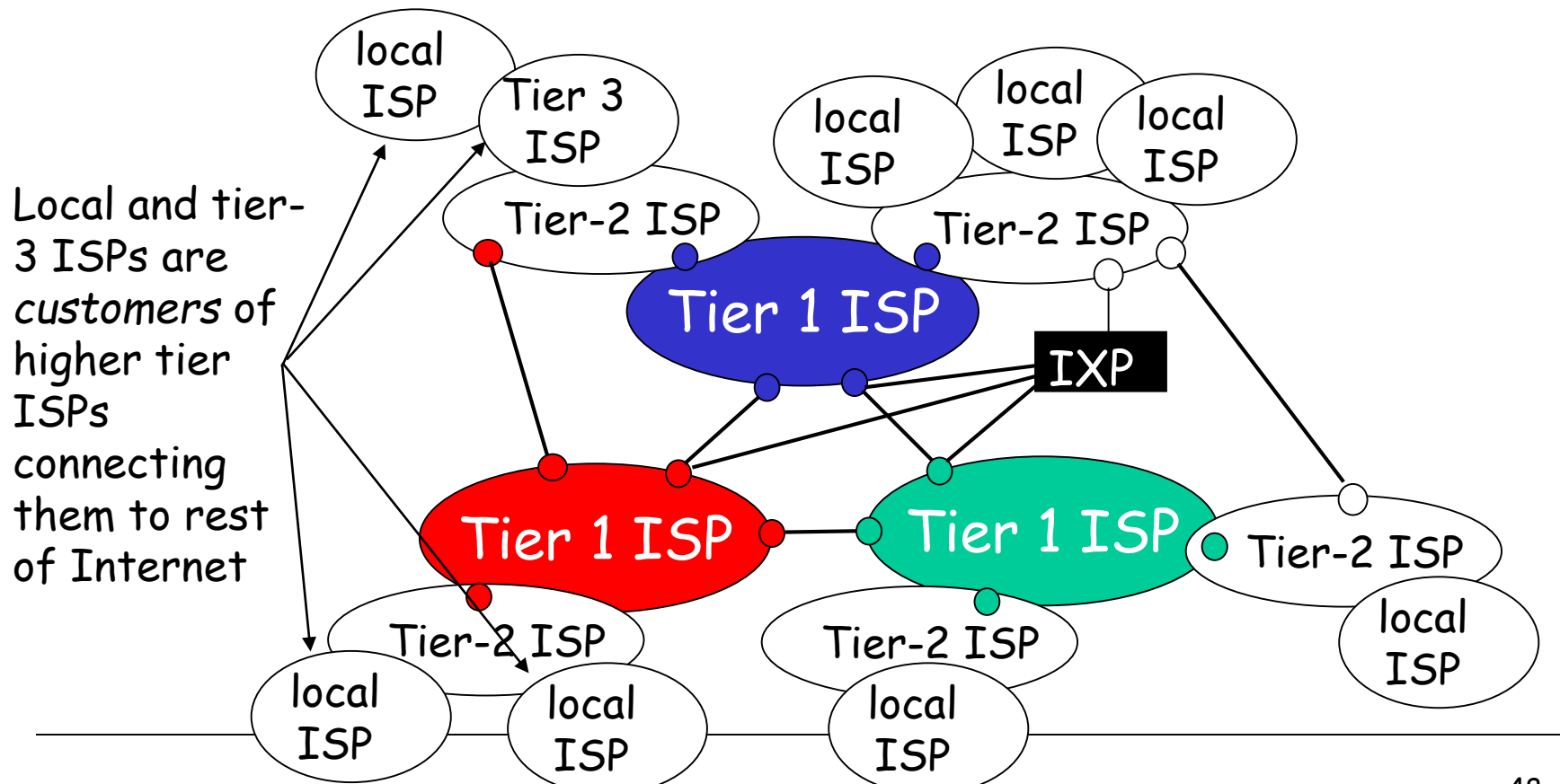
Internet structure: network of networks

- “Tier-2” ISPs: smaller (often regional) ISPs
 - Connect to one or more tier-1 ISPs, possibly other tier-2 ISPs



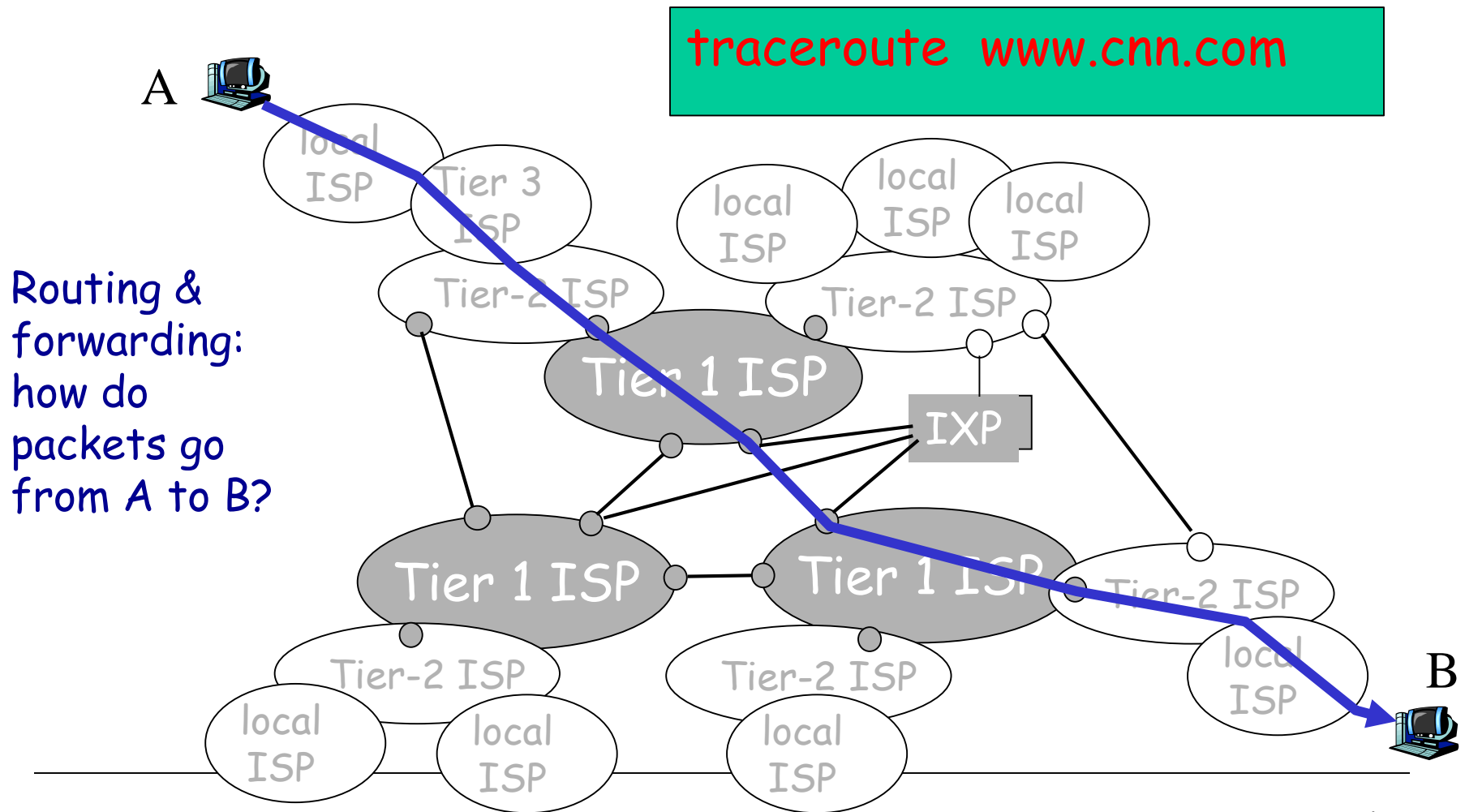
Internet structure: network of networks

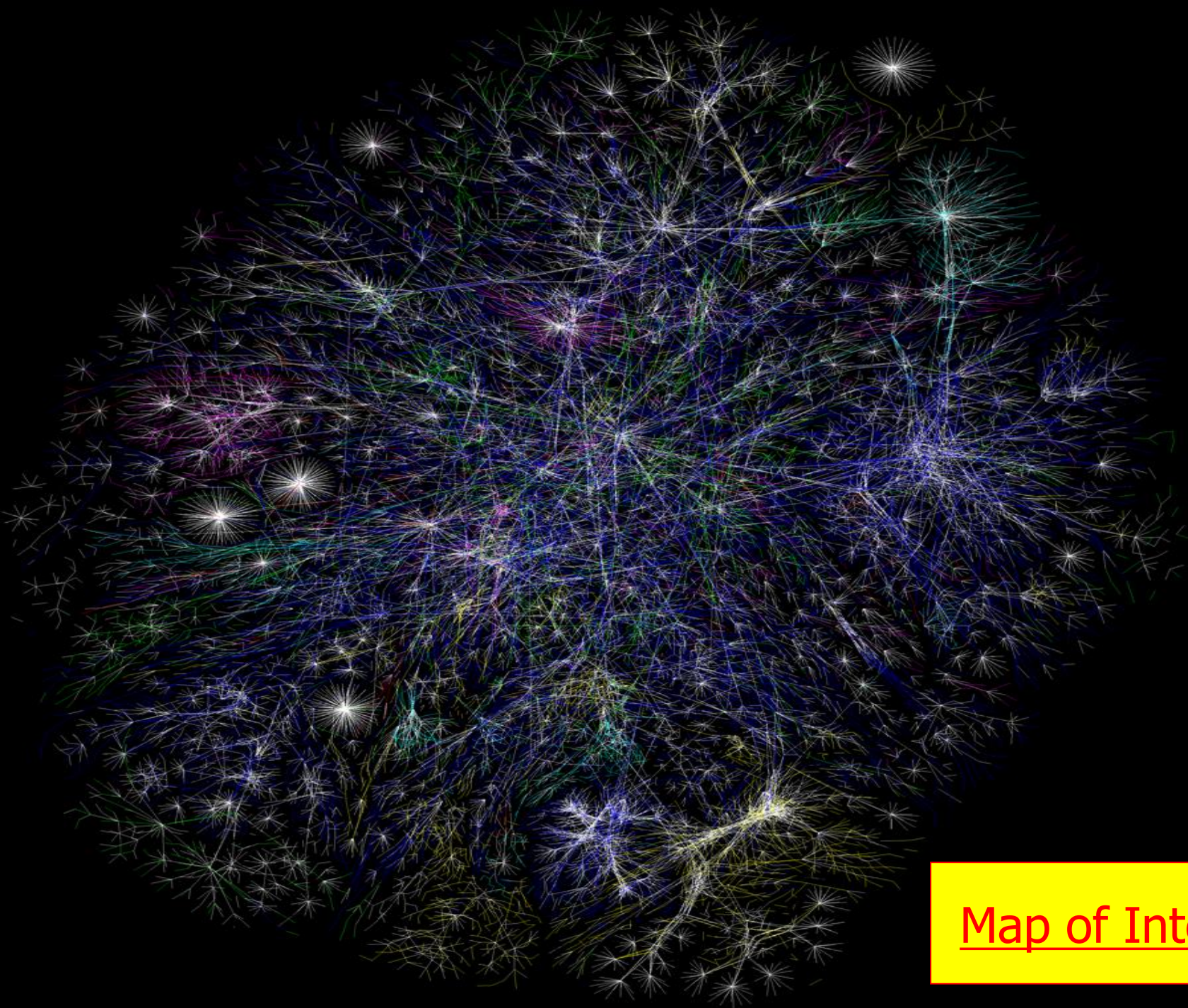
- “Tier-3” ISPs and local ISPs
 - last hop (“access”) network (closest to end systems)



Internet structure: network of networks

- a packet passes through many networks!





Map of Internet

Questions?

Announcement & Reminder (Sep 25)

- Programming Project # 1 has also been posted on the class website:

Due Monday Oct 15 11:59pm

- Hw #1: due: Friday Oct 6 11:59pm

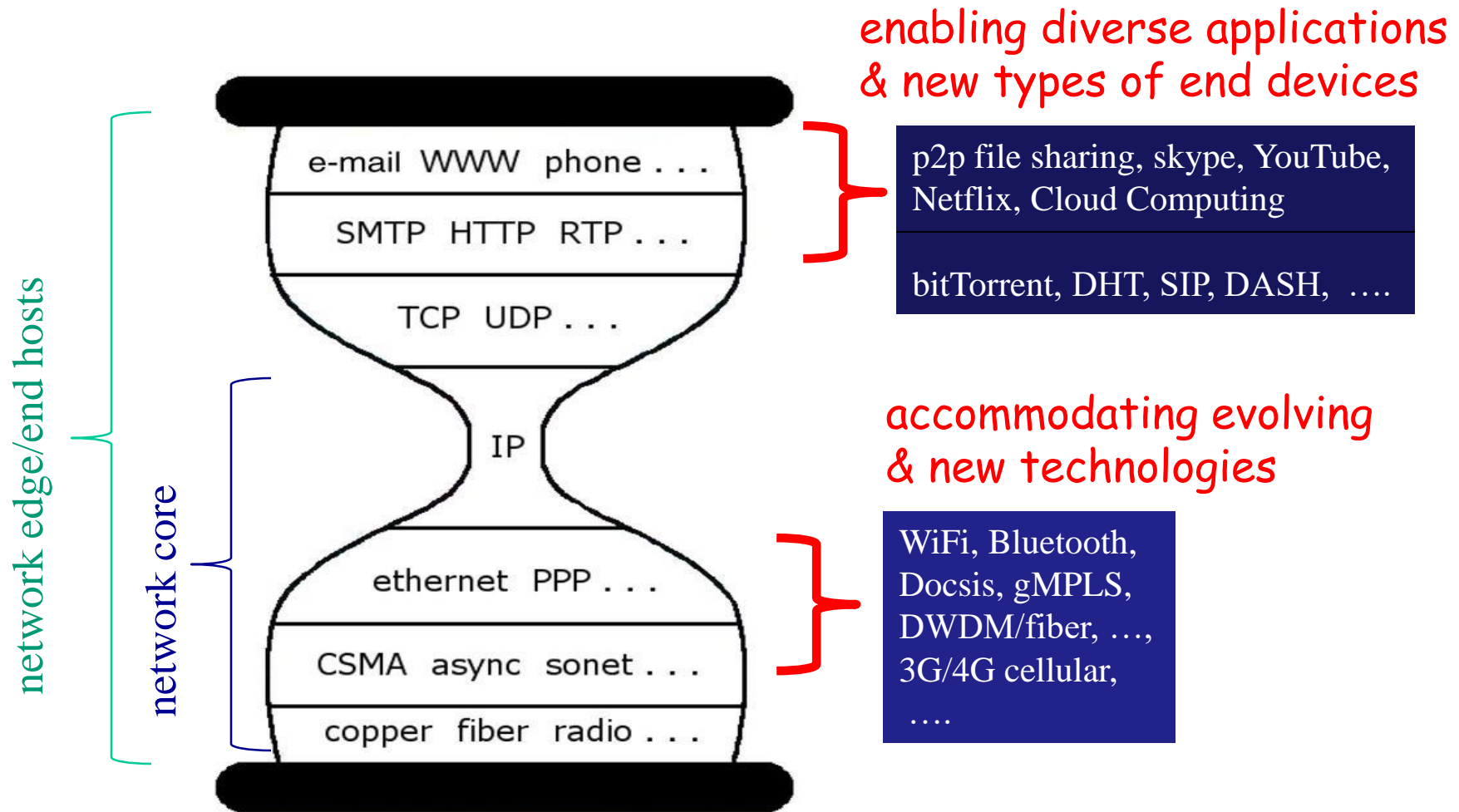
Please start working on them, *if* you haven't started yet!

What We Learned Last Time (Sep 21)

We have reviewed:

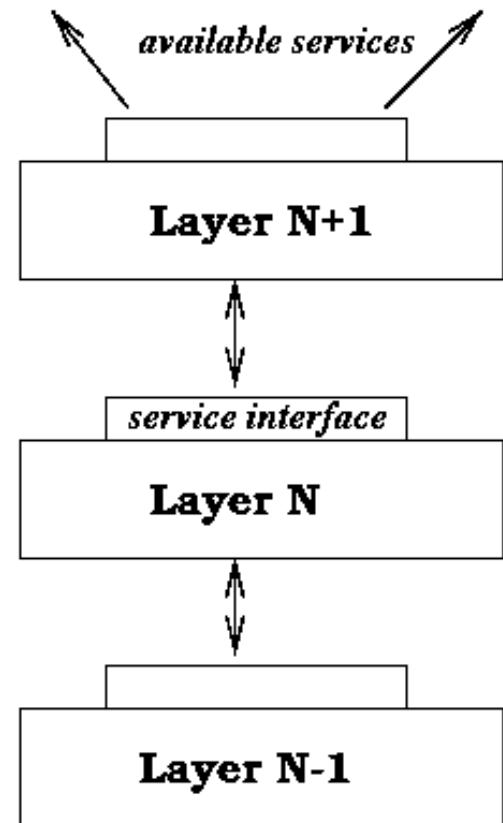
- ❖ What is a network? What is a computer/data network?
Compare w/ diff. networks (telephone, postal office, transportation networks, ...)
 - “bolts-&-nuts” view vs. service perspective
- ❖ Network is a *shared* resource!
 - ways to share (“multiplex”) resources? TDMA, FDMA, CDMA, ...
- ❖ **Packet switching** vs. circuit switching:
 - **packets, packet switching and statistical multiplexing**
 - For “bursty” data applications
 - store-&-forward
 - delay, losses and congestion vs. call blocking
- ❖ **New: four types of delays**
 - propagation, transmission, processing & queueing delays
- ❖ **Architecture: layering & hourglass**
 - ❖ different technologies, “boxes” (routers, switches), & apps
 - ❖ **Protocols and Interfaces (API)**

Internet Hourglass Architecture



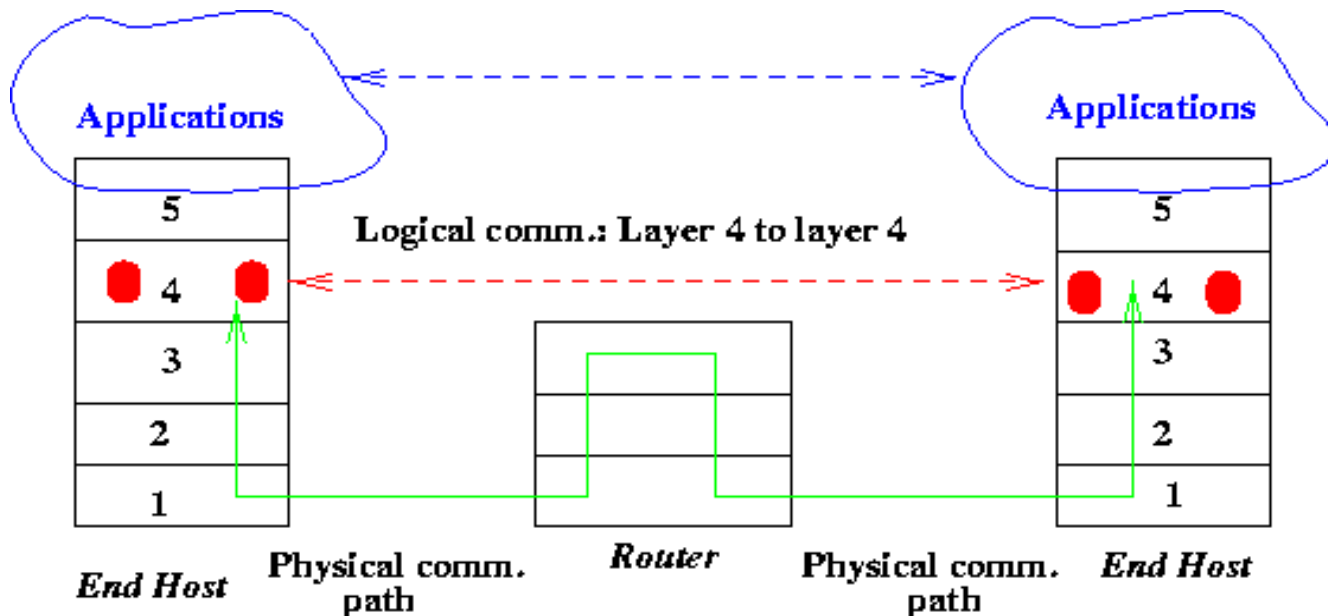
Layered Architecture

- Layering simplifies the architecture of complex system
- Layer N relies on *services* from layer N-1 to provide a *service* to layer N+1
- *Interfaces* define the services offered
- Service required from a lower layer is independent of its implementation
 - Layer N change doesn't affect other layers
 - Information/complexity hiding
 - Similar to object oriented methodology



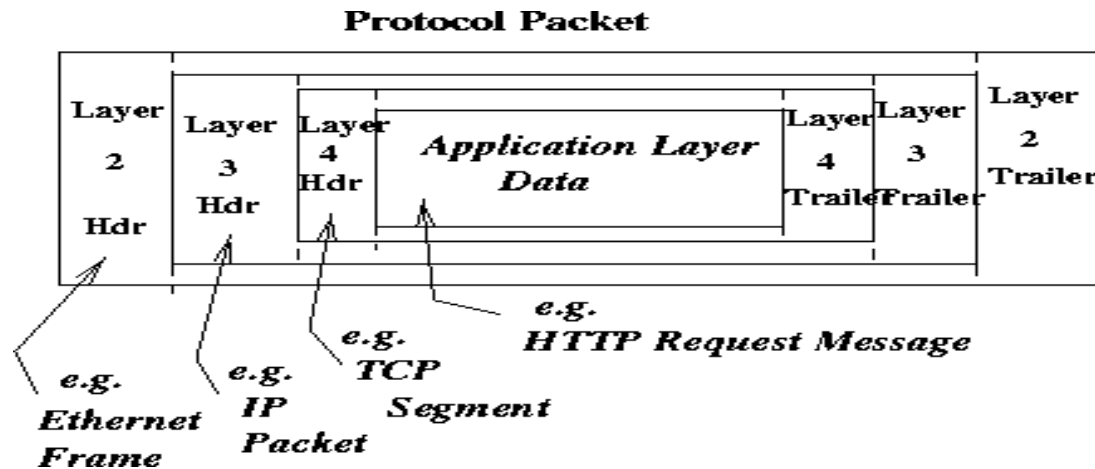
Protocols and Services

- Protocols are used to implement services
 - Peering entities in layer N provide service by communicating with each other using the service provided by layer N-1
- *Logical vs physical communication*

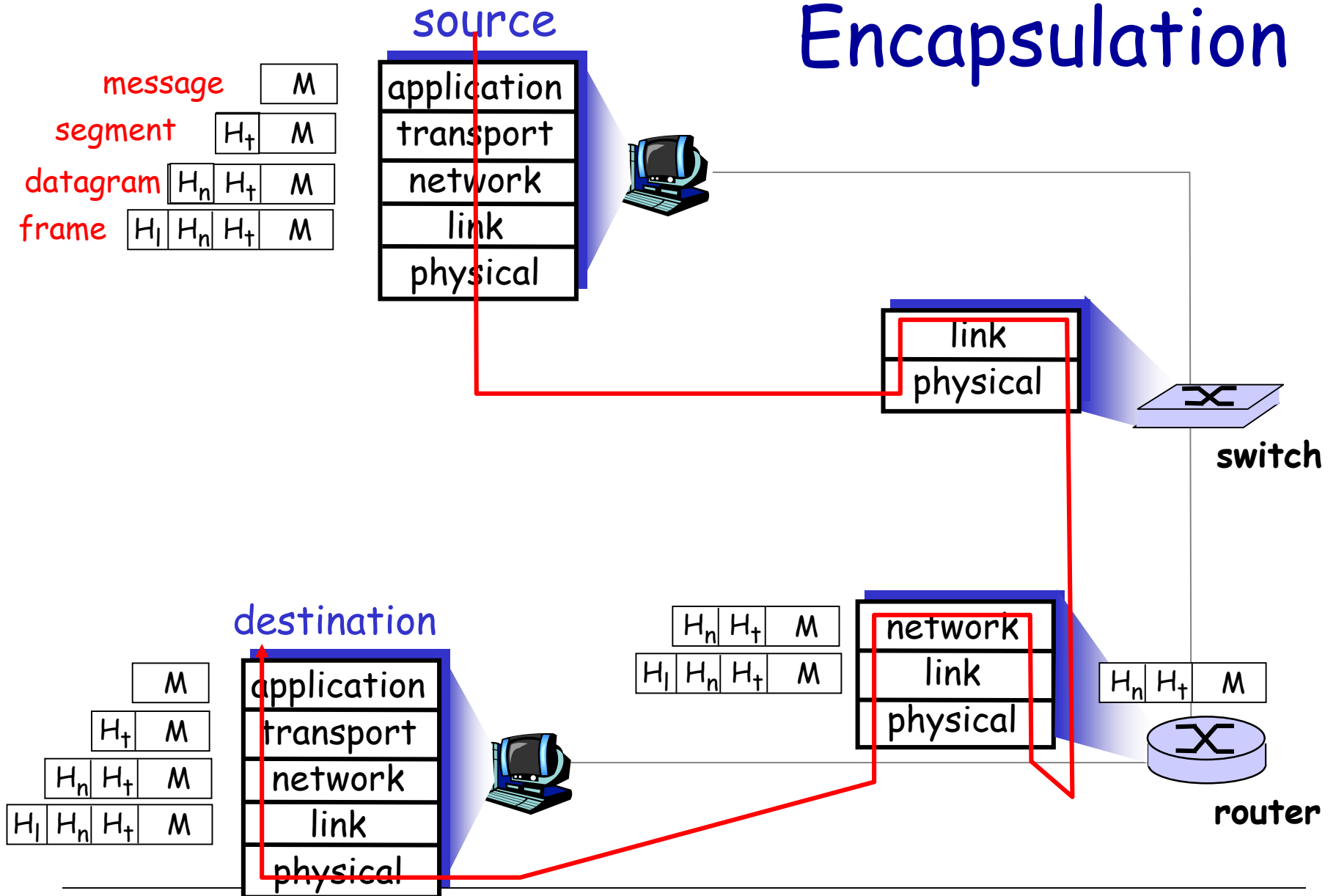


Protocol Packets

- Protocol data units (PDUs):
 - packets exchanged between peer entities
- Service data units (SDUs):
 - packets handed to a layer by an upper layer
- Data at one layer is encapsulated in packet at a lower layer
 - *Envelope within envelope*: PDU = SDU + (optional) header or trailer

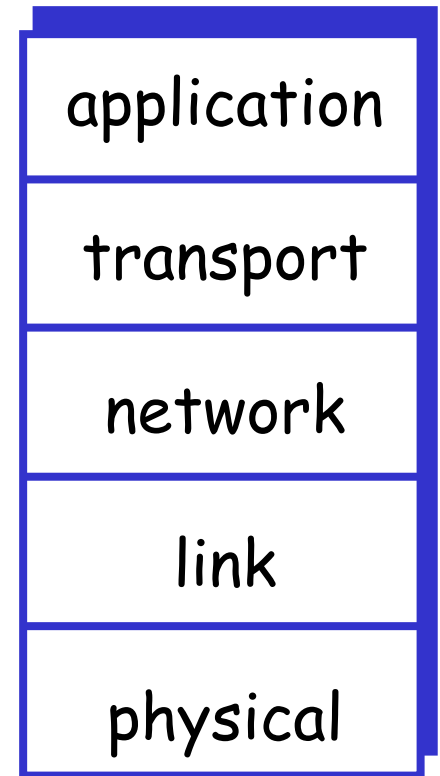


Encapsulation

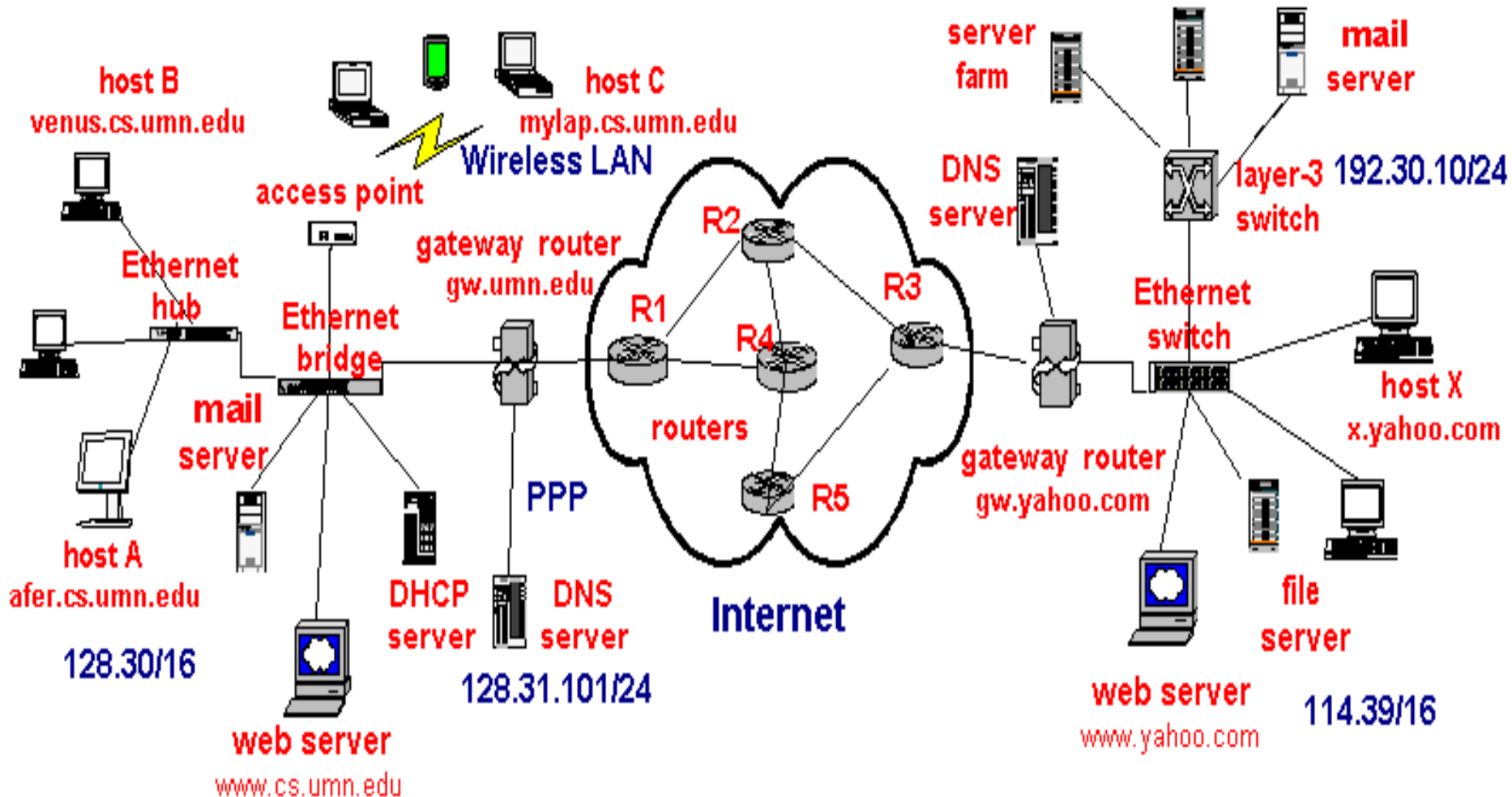


Internet Protocol Stack

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



A Simplified Illustration of Internet

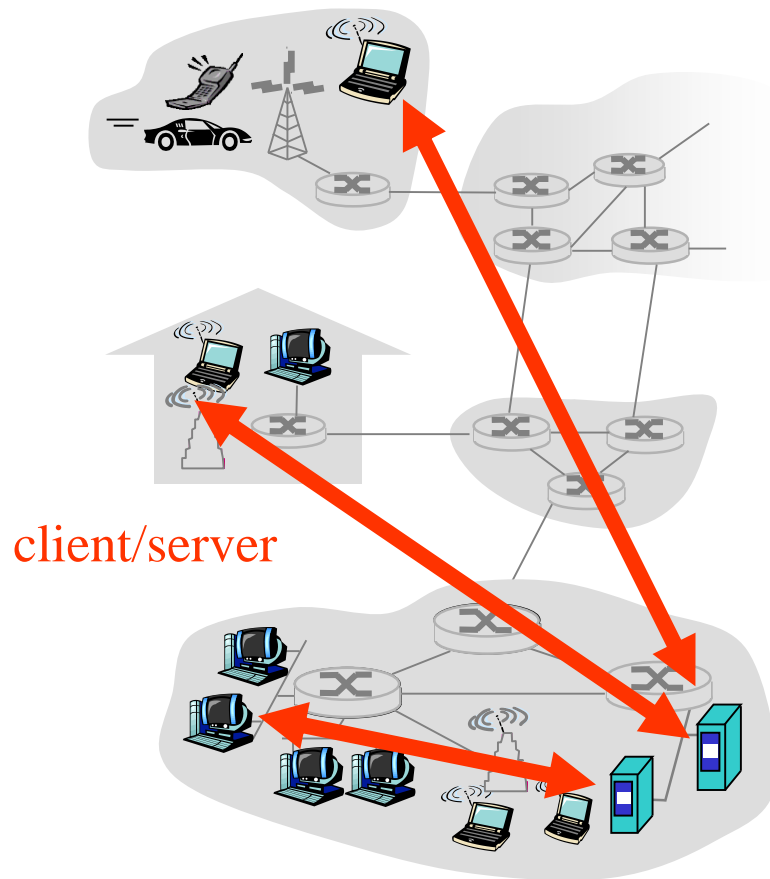


Applications and Application Layer Protocols

Basics of Building Applications: a networking perspective

- ❖ application **processes** and inter-process communications
 - ❖ API: **socket** overview
 - ❖ “Addressing” processes (“whom is the other party is”)
 - ❖ **IP addresses and port numbers**
 - ❖ What transport services to use?
 - ❖ **TCP and UDP**
 - ❖ Application Structures
 - ❖ **client-server** → data centers, cloud services
 - ❖ **peer-to-peer**
 - ❖ Case studies: applications/application protocols
 - ❖ **world wide web and HTTP**: transaction-oriented app protocol
 - ❖ **email and SMTP** (& POP, IMAP): session-based app protocol
- download
“wireshark”
software!

Client-server architecture



server:

- always-on host
- permanent IP address
- server farms for scaling

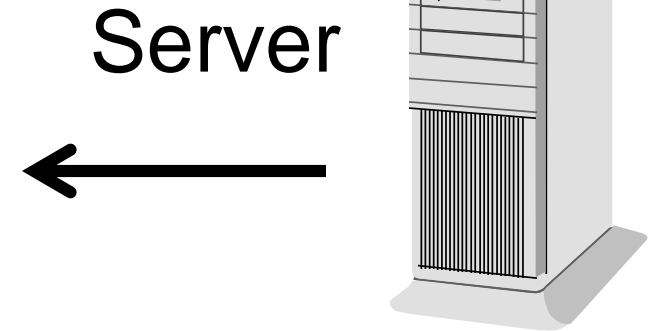
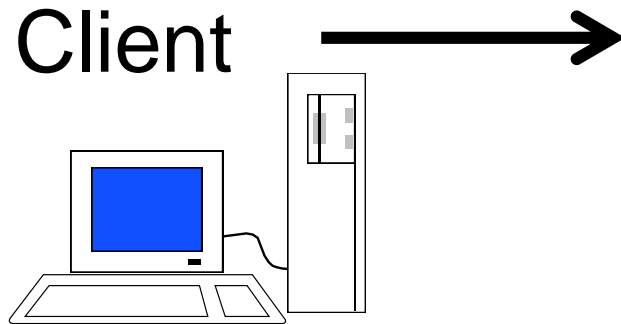
clients:

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

Web and HTTP Summary

Transaction-oriented (request/reply), use TCP, port 80

Non-persistent (HTTP/1.0) vs. persistent (HTTP/1.1)



GET /index.html HTTP/1.0

HTTP/1.0

200 Document follows

Content-type: text/html

Content-length: 2090

-- blank line --

HTML text of the Web page

HTTP request message

- two types of HTTP messages: *request, response*
- **HTTP request message:**
 - ASCII (human-readable format)

request line
(GET, POST,
HEAD commands)

header
lines

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

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

carriage return character

line-feed character

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

HTTP response message

status line
(protocol
status code
status phrase)

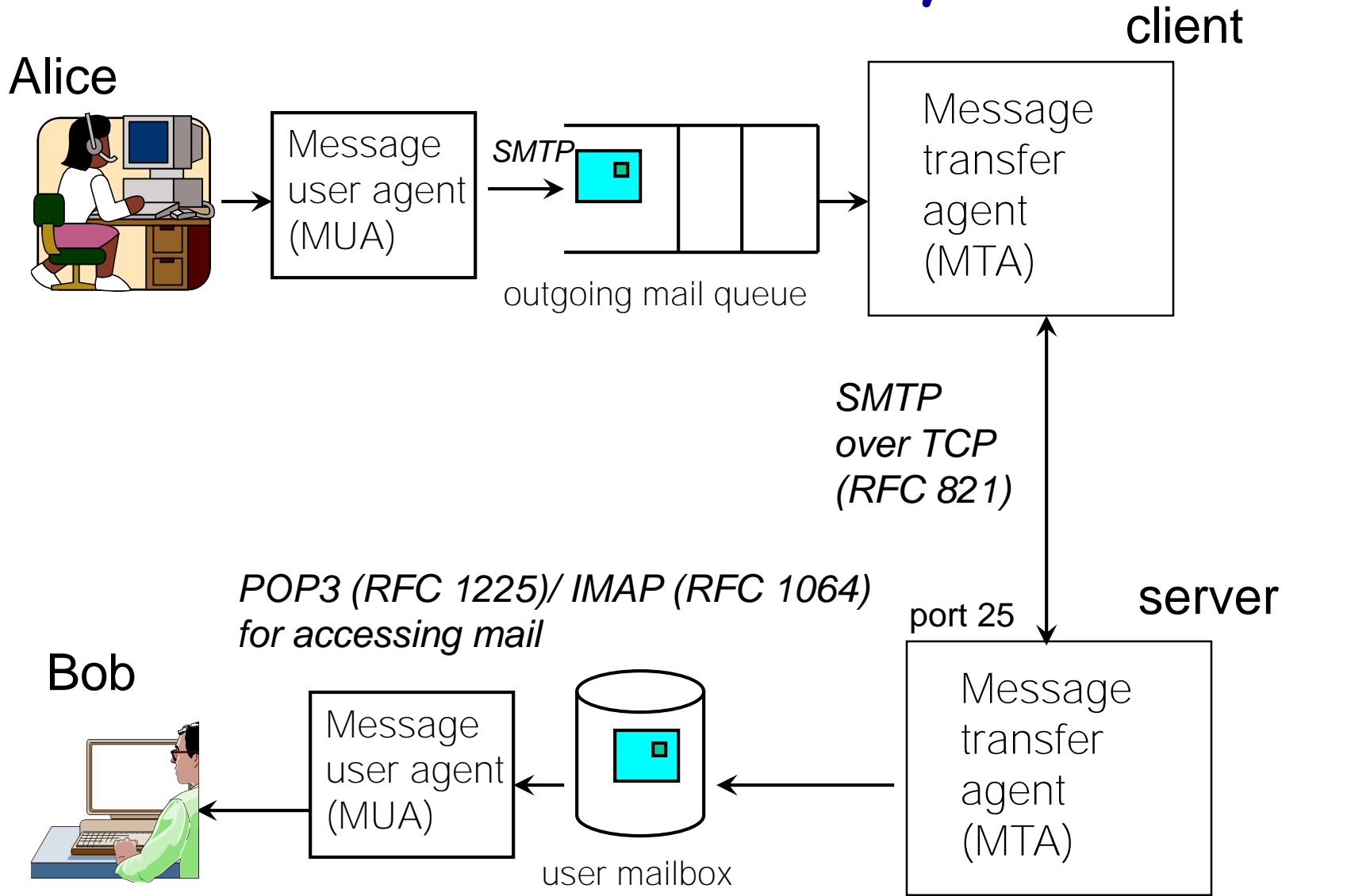
header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
1\r\n
\r\n
data data data data data ...
```

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

Email Summary



Sample SMTP Interaction

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

Reminders (Oct 2)

- Hw #1: due: this Friday Oct 6 11:59pm
 - submission online using the class Moodle site
- Programming Project # 1: due Monday Oct 15 11:59pm

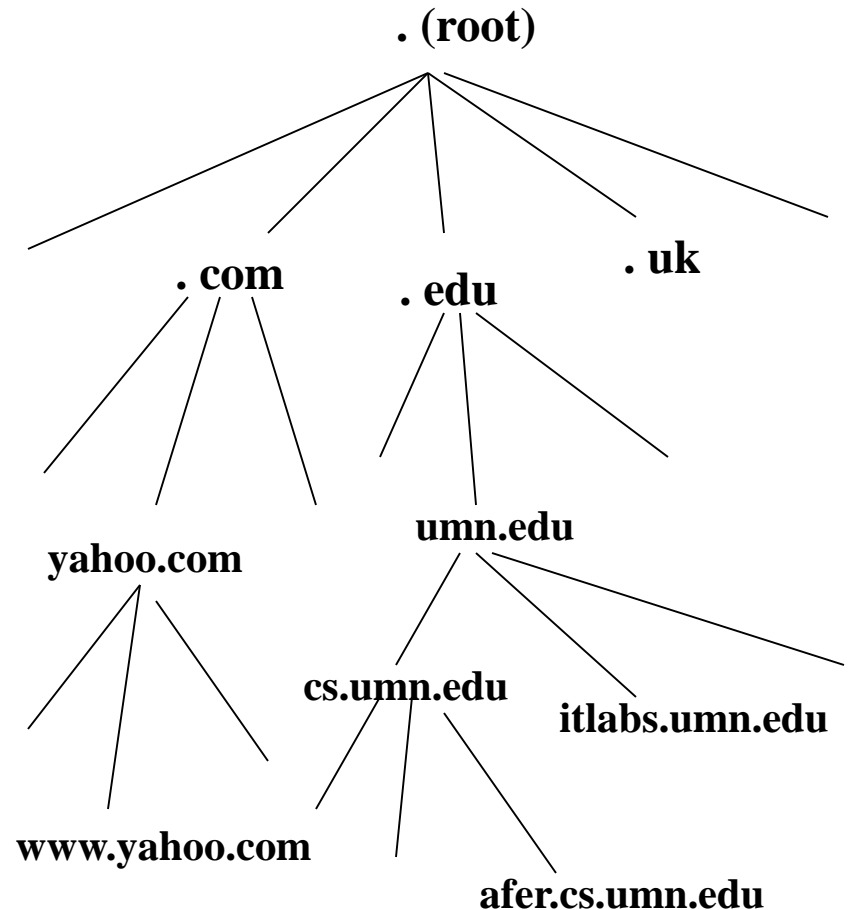
Week of Oct 16: Take-Home Quiz I!

What We Learned Last Time (Sept 25) ...

- ❖ Review applications/application protocols
 - ❖ **world wide web and HTTP**: transaction-oriented app protocol; non-persistent vs. persistent; maintaining "states" across request-reply transactions via cookie
 - ❖ **email and SMTP** (& POP, IMAP): session-based app protocol
- ❖ Domain Name System:
 - ❖ Mapping domain (DNS) name to IP addresses
- ❖ Briefly: Peer-to-Peer File Sharing and DHT
- ❖ Network Socket Programming
 - ❖ BSD socket programming interfaces: TCP vs. UDP sockets
 - ❖ Socket programming in Python
 - ❖ Socket programming in Java

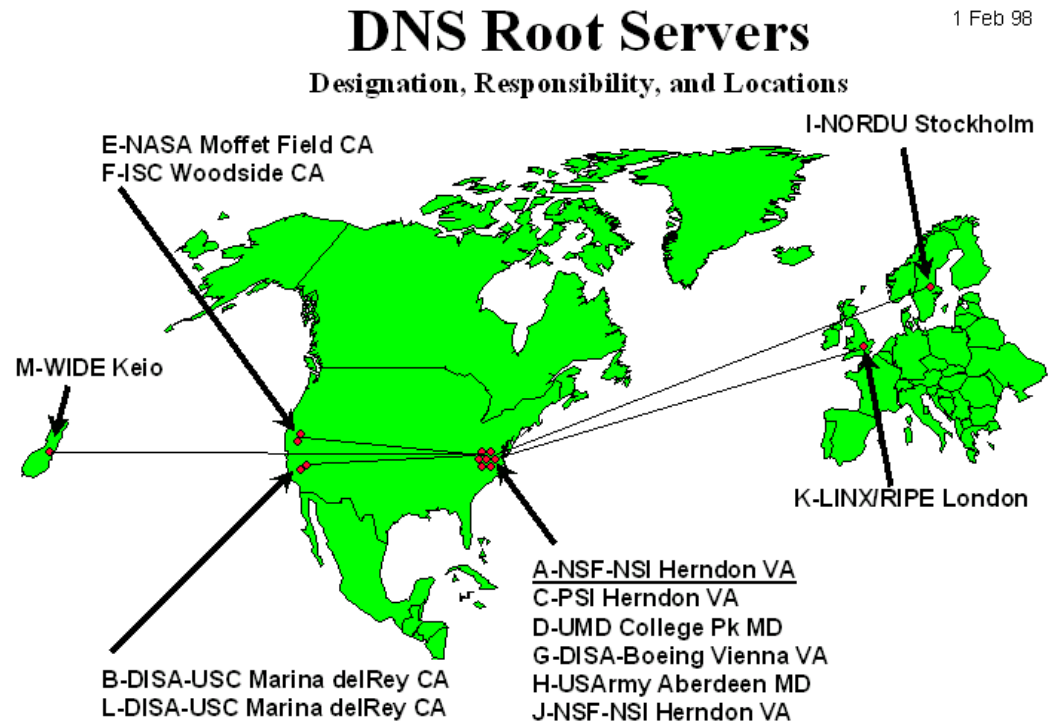
Internet Domain Names

- Hierarchical: anywhere from two to possibly infinity
- Examples:
afer.cs.umn.edu,
lupus.fokus.gmd.de
 - *edu, de*: organization type or country (a "domain")
 - *umn, fokus*: organization administering the "sub-domain"
 - *cs, fokus*: organization administering the host
 - *afer, lupus*: host name (have IP address)



DNS: Root name servers

- contacted by local name server that can not resolve name
- root name server:
 - contacts authoritative name server if name mapping not known
 - gets mapping
 - returns mapping to local name server
- ~ dozen root name servers worldwide



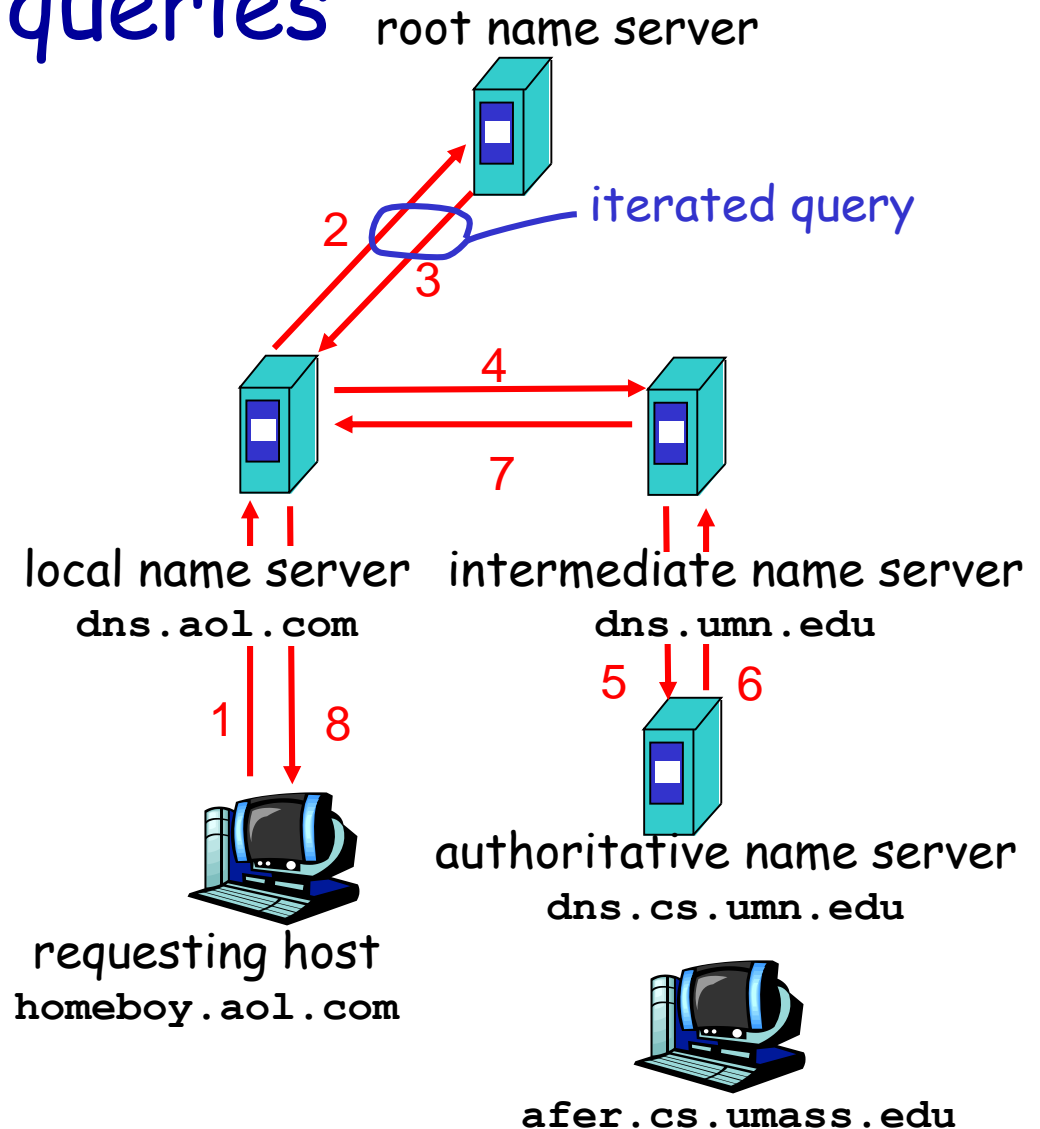
DNS: iterated queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
 - cache entries timeout (disappear) after some time
- update/notify mechanisms under design by IETF
 - RFC 2136
 - <http://www.ietf.org/html.charters/dnsind-charter.html>

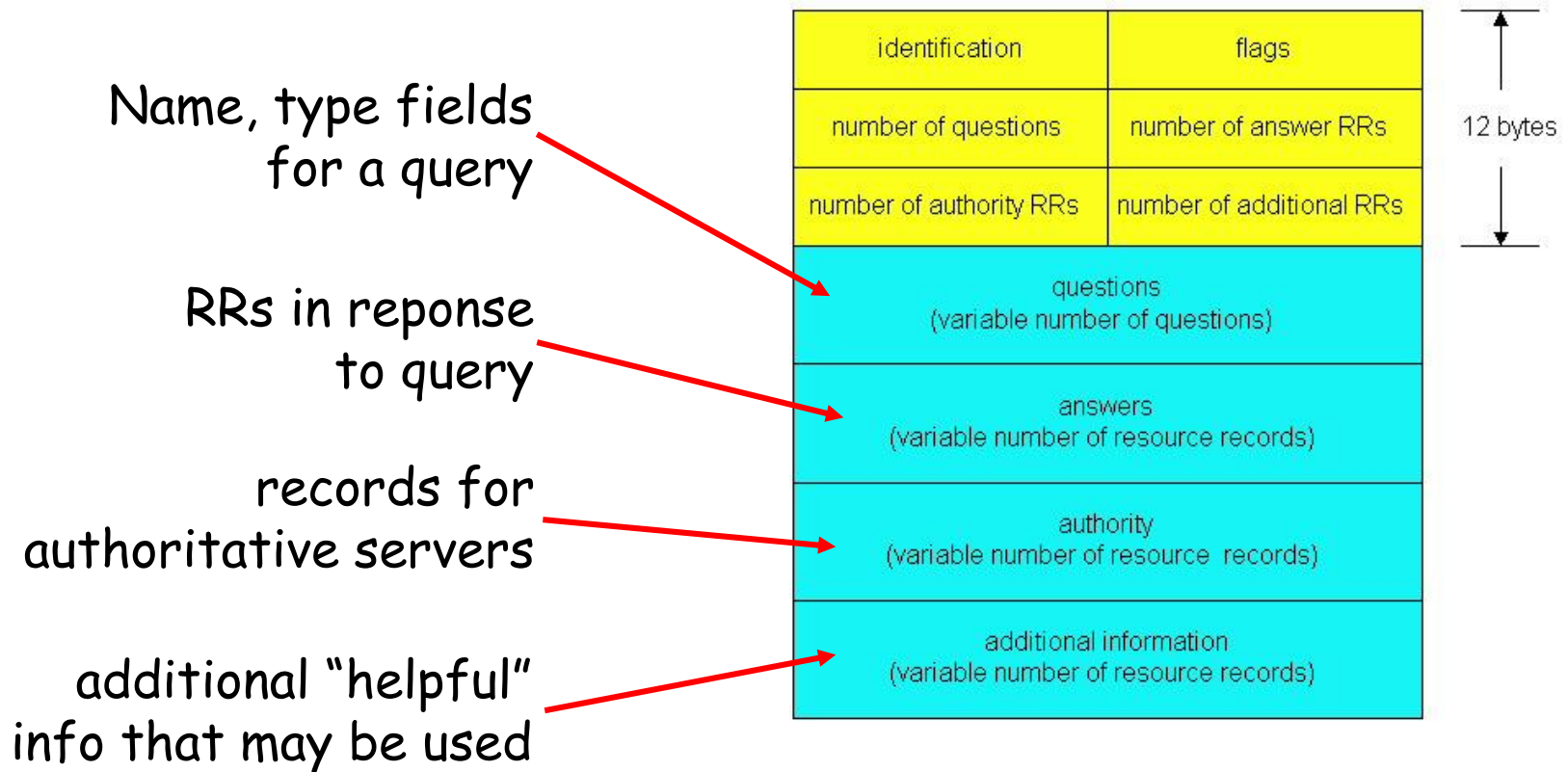
DNS records

DNS: distributed db storing resource records (RR)

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

- Type=A
 - name is hostname
 - value is IP address
- Type=NS
 - name is domain (e.g. foo.com)
 - value is IP address of authoritative name server for this domain
- Type=CNAME
 - name is an alias name for some "canonical" (the real) name
 - value is canonical name
- Type=MX
 - value is hostname of mailserver associated with name

DNS protocol, messages



DNS Protocol

- Query/Reply: use UDP, port 53
- Transfer of DNS Records between authoritative and replicated servers: use TCP

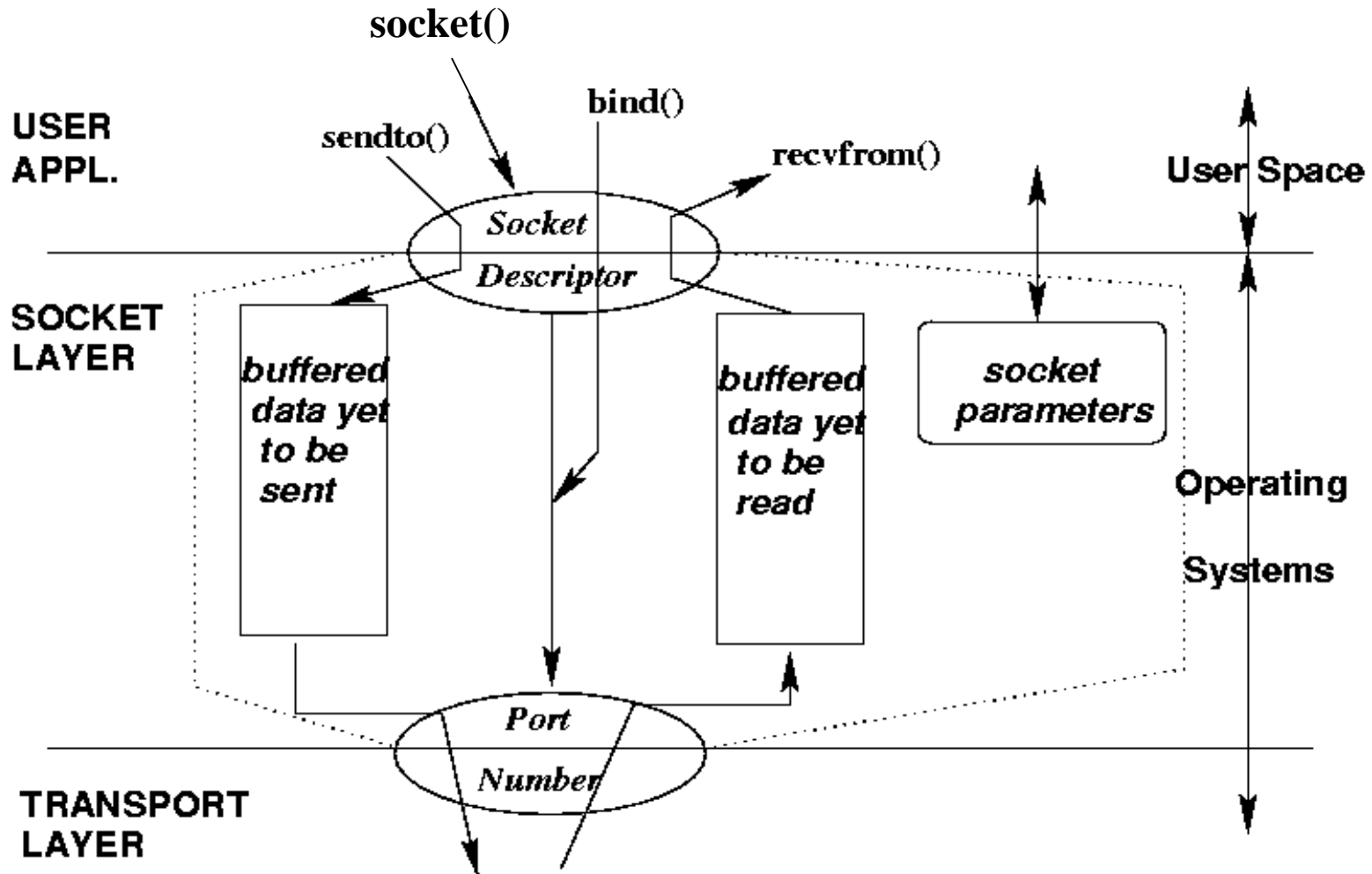
Peer-to-Peer Paradigm

- How do we implement peer-to-peer model?
- Is email peer-to-peer or client-server application?
- How do we implement peer-to-peer using client-server model?

Difficulty in implementing "pure" peer-to-peer model?

- How to locate your peer?
 - Centralized "directory service:" i.e., white pages
 - Napsters
 - Unstructured: e.g., "broadcast" your query: namely, ask your friends/neighbors, who may in turn ask their friends/neighbors,
 - Freenet
 - Structured: Distributed hashing table (DHT)

Socket: Conceptual View



A summary of BSD Socket

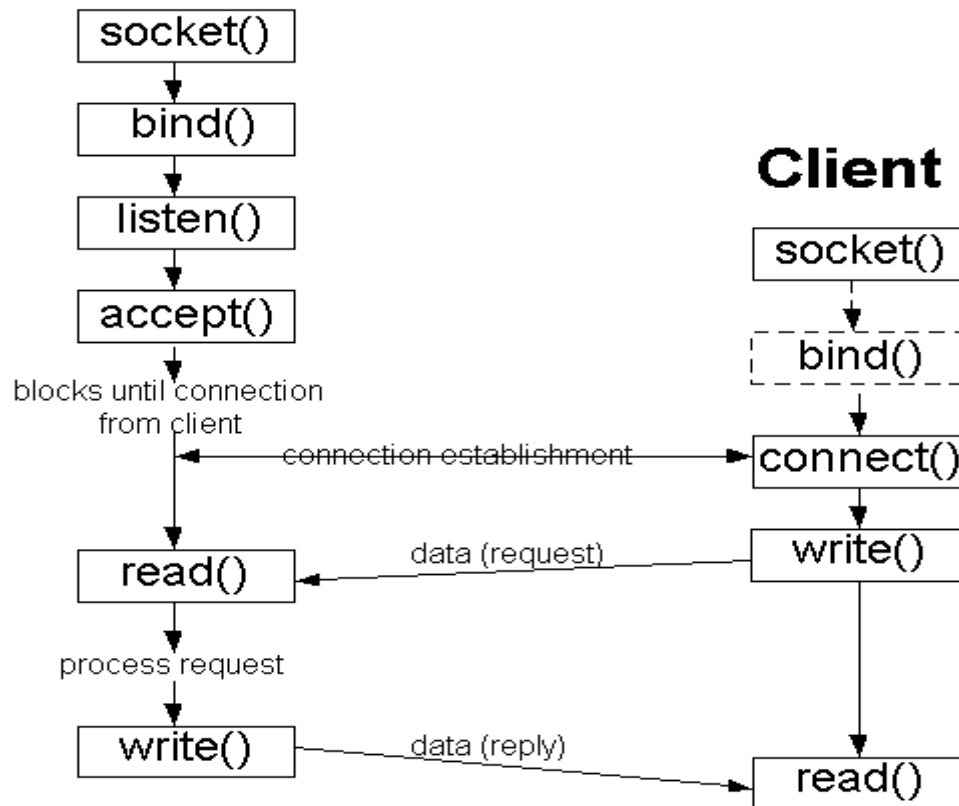
protocol localAddr,localPort remoteAddr, remotePort

conn-oriented server	socket()	bind()	listen(), accept()
conn-oriented client	socket()	connect()	
connectionless server	socket()	bind()	recvfrom()
connectionless client	socket()	bind()	sendto()

BSD Socket Programming Flows (connection-oriented)

Server

(connection-oriented protocol)



BSD Socket Programming (connectionless)

Server

(connectionless protocol)

socket()

bind()

recvfrom()

blocks until data received
from a client

process request

sendto()

Client

socket()

bind()

sendto()

recvfrom()

data (request)

data (reply)

Reminders (Oct9)

- ❖ Programming Project # 1:
due this Sunday Oct 15 11:59pm!
- ❖ ***Next Week: Take-Home Quiz I!***
Hand-out Monday Oct 16,
Due Friday Oct 20 11:59pm

What We Learned Last Time (Oct 2) ...

- ❖ Review Domain Name System (DNS)
 - ❖ Mapping domain (DNS) name to IP addresses
- ❖ Briefly Review Peer-to-Peer File Sharing and DHT
- ❖ **Transport Layer: UDP vs. TCP**
 - ❖ port numbers: multiplexing/de-multiplexing app processes
 - ❖ checksum
- ❖ **UDP: connectionless datagram service**
- ❖ **TCP: connect-oriented, reliable, in-order byte streams (fully duplex)**
 - ❖ Connection set-up and tear-down
 - ❖ **Reliable data transfer (today)**
 - ❖ flow control & control congestion (next Monday)

UDP: User Datagram Protocol [RFC 768]

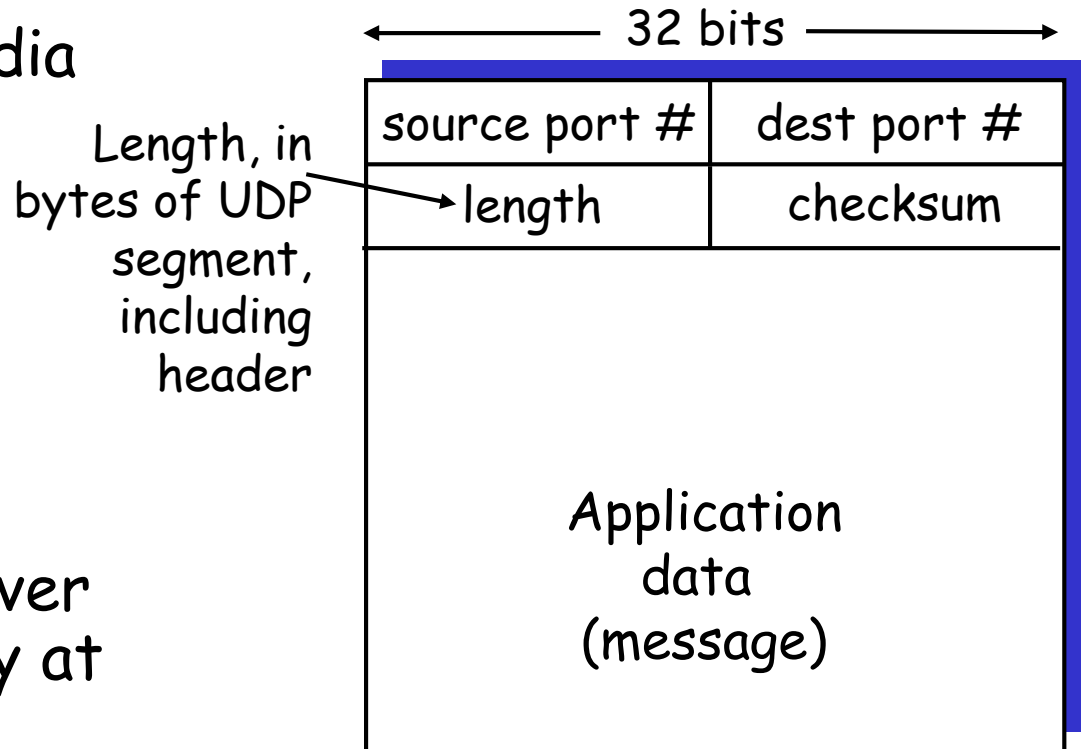
- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
 - lost
 - delivered out of order to app
- *connectionless*:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP Datagram Format

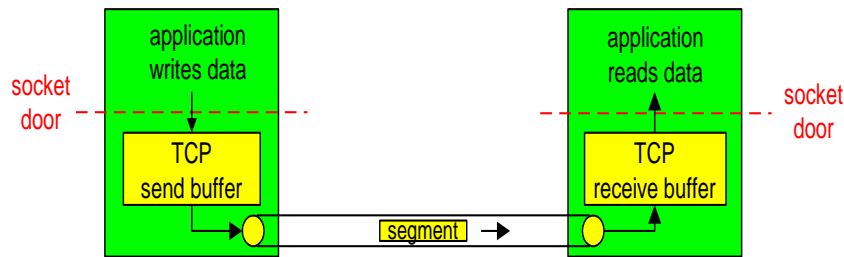
- often used for streaming multimedia apps
 - loss tolerant
 - rate sensitive
- other UDP uses
 - DNS
 - SNMP
- reliable transfer over UDP: add reliability at application layer
 - application-specific error recovery!



UDP segment format

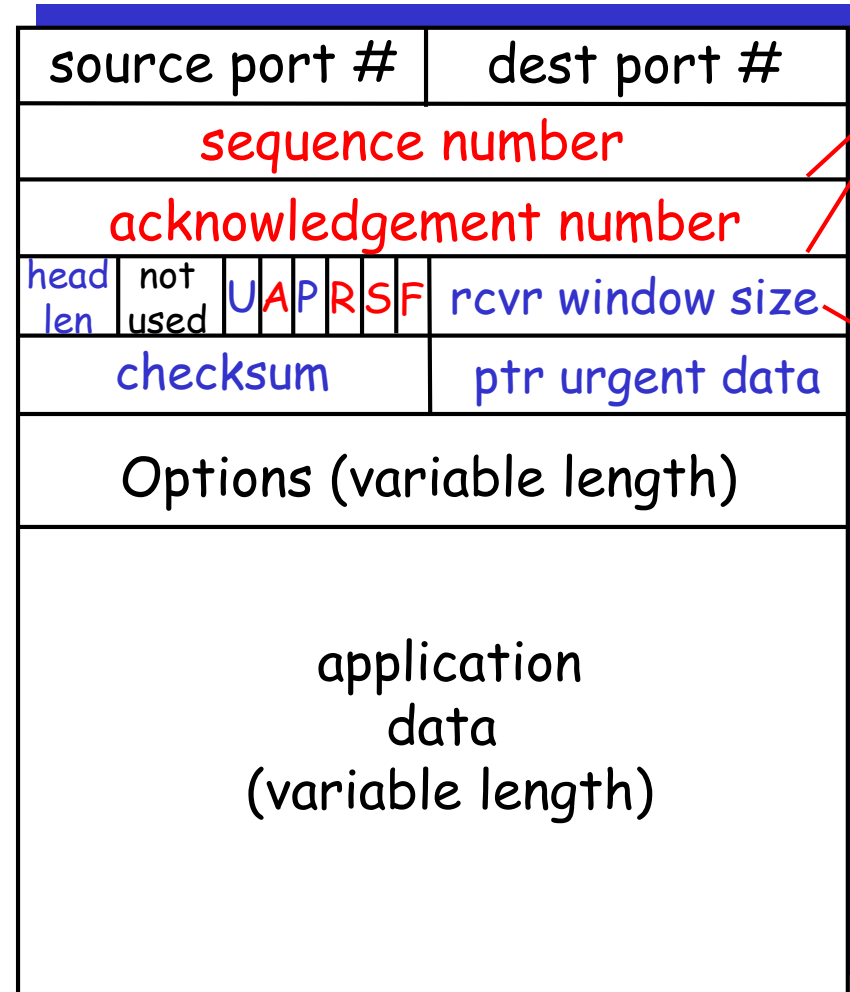
TCP: Overview

- **point-to-point:**
 - one sender, one receiver
- **reliable, in-order *byte stream*:**
 - no “message boundaries”
- **pipelined:**
 - TCP congestion and flow control set window size
- ***send & receive buffers***
- **full duplex data:**
 - bi-directional data flow in same connection
 - MSS: maximum segment size
- **connection-oriented:**
 - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- **flow controlled:**
 - sender will not overwhelm receiver



TCP Segment Structure

← 32 bits →



URG: urgent data
(generally not used)

ACK: ACK #
valid

PSH: push data now
(generally not used)

RST, SYN, FIN:
connection estab
(setup, teardown
commands)

Internet
checksum
(as in UDP)

counting
by bytes
of data
(not segments!)

bytes
rcvr willing
to accept

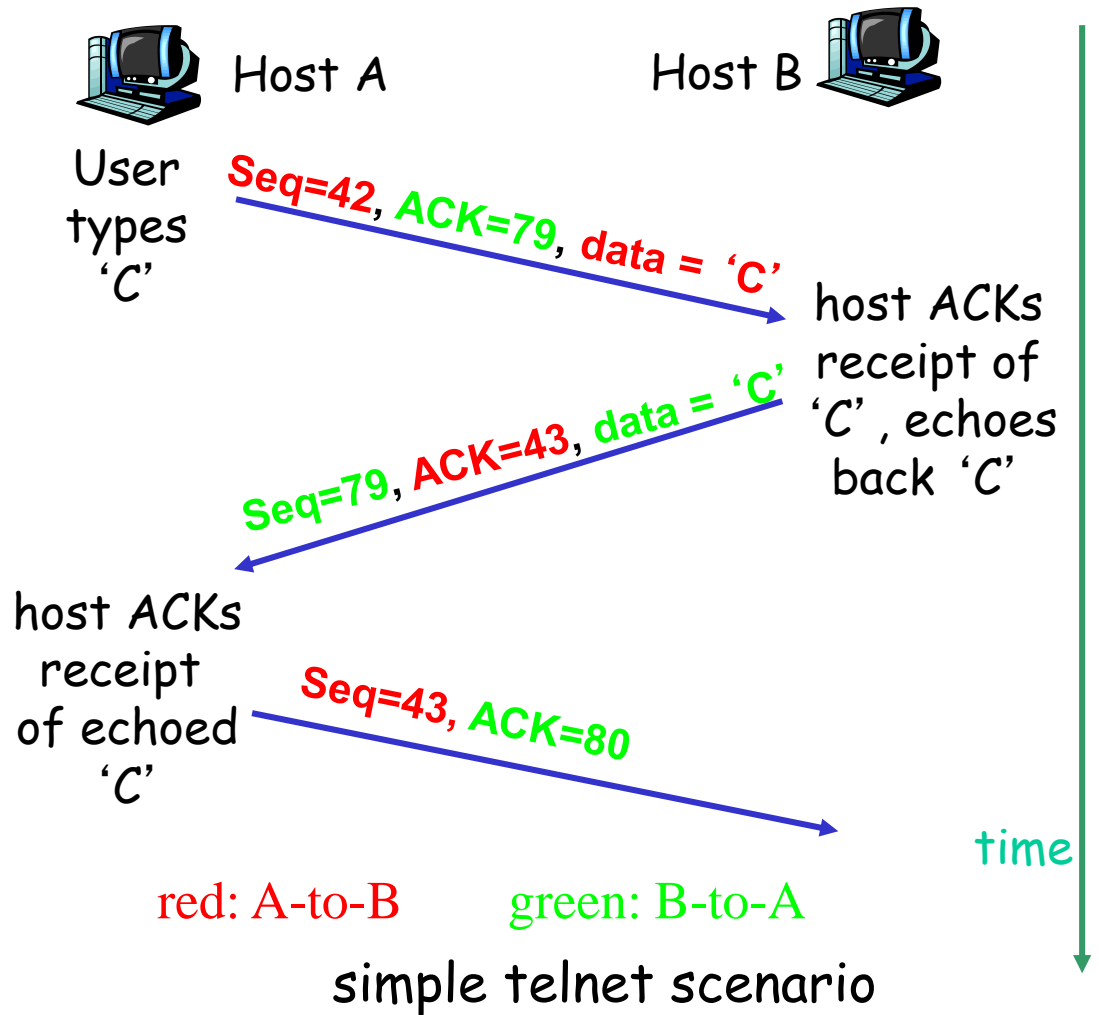
TCP Seq. # & Ack.#

Seq. #'s:

byte stream
“number” of first
byte in segment's
data

ACKs:

seq # of next byte
expected from
other side



TCP: Some Key Issues

How to design a simple “reliable” transfer protocol?

- stop-&-wait protocol (or alternative bit protocol)

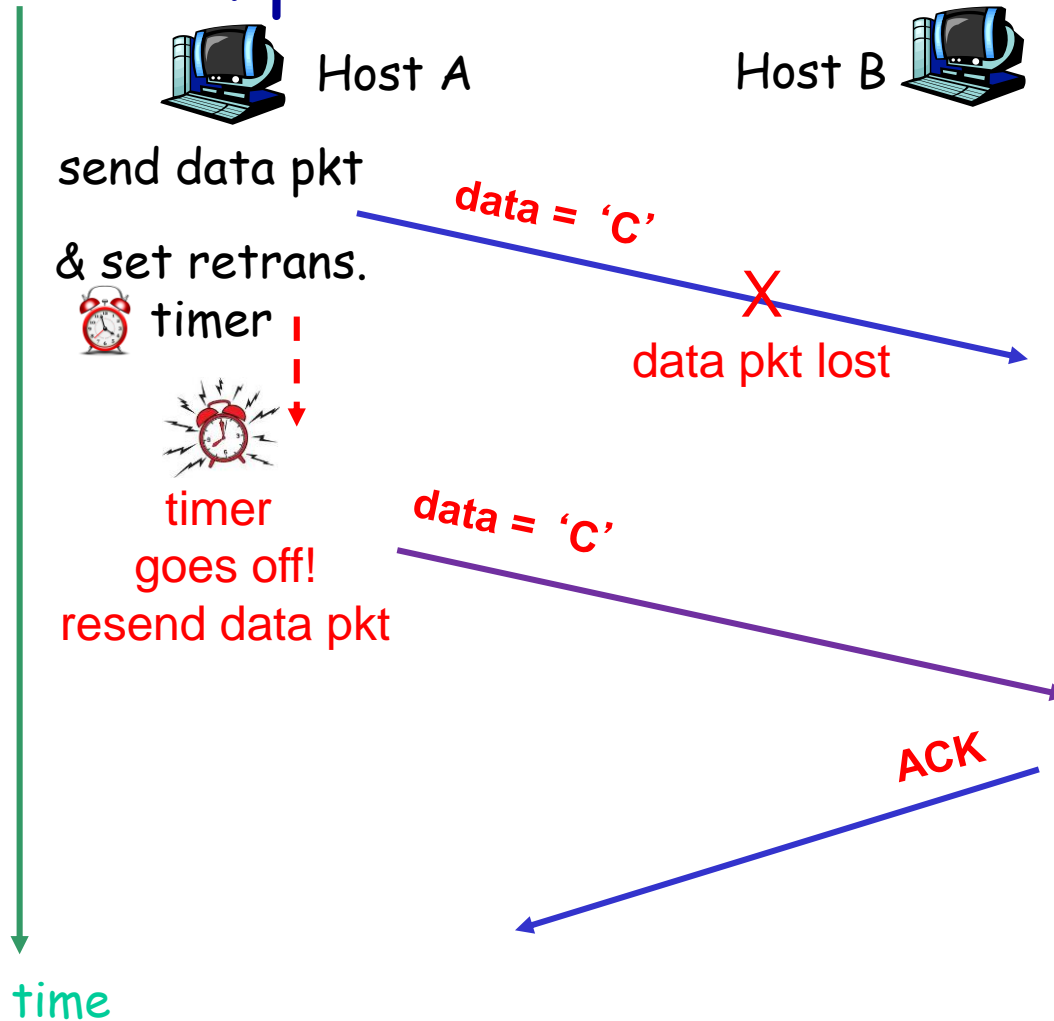
□ How to deal with “lost” packets?

- how to detect a packet is “lost”?
 - need to set a timer (“retransmission timer”)
 - what value shall we choose to set the timer?
- what to do with “lost” packets?
 - retransmit the “lost” packet when timer goes off

□ What problem packet retransmission create?

- (potential) duplicate packets!
 - e.g., when a “lost” packet is not actually lost, but takes longer to get delivered, after timer times out
- how to recognize duplicate packets?
 - sequence #: but why do we need sequence #? how many bits shall we use?

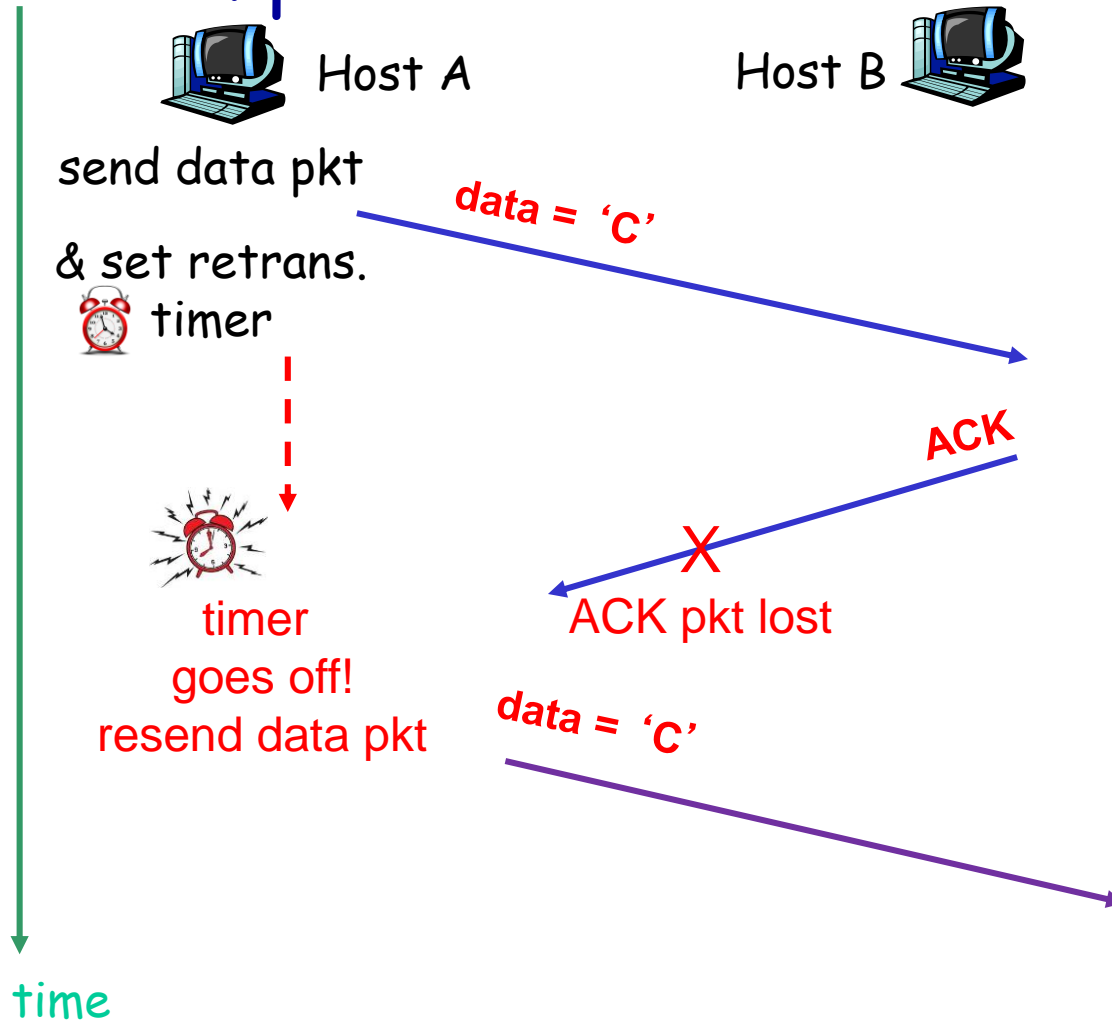
Simple Reliable Data Transfer Protocol



Error Scenarios:

- data pkt lost

Simple Reliable Data Transfer Protocol



Error Scenarios:

- data pkt lost
- ack pkt lost

Note: Host A can't tell whether the two error scenarios apart:
whether data packet or ACK packet is lost

Simple Reliable Data Transfer Protocol



Host A



Host B

send data pkt

& set retrans.



timer



timer goes off!
resend data pkt

data = 'C'

ACK

data = 'C'

time

Error Scenarios:

- data pkt lost
- ack pkt lost
- retrans. timer goes off (but none is lost!)
→ duplicate packets

Host B: did host A send
"C" or "CC" ?

Note: Host B has *no idea* that
retrans. timer at Host A went off!

Simple Reliable Data Transfer Protocol



Host A



Host B

send data pkt

& set retrans.



timer

data = 'C'

Why Do We Need
Seq # ?

ACK

data = 'C'

Host B: Did host A send
"C" or "CC" ?

time

Error Scenario (retrans. timer goes off) vs. **Normal Scenario** (host A sends another data pkt (with char "C")) look exactly the same from Host B's perspective!

Simple Reliable Data Transfer Protocol



Host A



Host B

send data pkt

Seq=42, data = 'C'

& set retrans.



timer



timer
goes off!

Seq=42, data = 'C'

ACK ack=43

*expecting data pkt
w/ seq. 42;*

*received pkt. w/seq=42
send ACK w/ ack=43;*

expecting data pkt w/ seq=43

ACK ack=43

*Duplicate data pkt,
resend ACK w/ ack=43!*

time

Simple Reliable Data Transfer Protocol



Host A



Host B

send data pkt

Seq=42, data = 'C'

& set retrans.



timer



timer
goes off!

Seq=42, data = 'C'

ACK ack=43

*expecting data pkt
w/ seq. 42;*

*received pkt. w/seq=42
send ACK w/ ack=43;*

expecting data pkt w/ seq=43

ACK ack=43

*Duplicate data pkt,
resend ACK w/ ack=43!*

time

Simple Reliable Data Transfer Protocol



Host A



Host B

send data pkt

& set retrans.



timer

Seq=42, data = 'C'

expecting data pkt
w/ seq. 42;

ACK ack=43

received pkt. w/seq=42
send ACK w/ ack=43;

expecting data pkt w/ seq=43

Seq=43, data = 'C'

ACK ack=44

received new data pkt
as expected!
send ACK w/ ack=44;

expecting data pkt w/ seq=43

time

With seq. included in the data packet, error Scenario (retrans. timer goes off) vs. Normal Scenario (host A sends another data pkt (with char "C")) can now be distinguished by host B!

A Simple Reliable Data Transfer Protocol

“Stop & Wait” Protocol (aka “Alternate Bit” Protocol)

Sender algorithm:

- **Send Phase:** send data segment (n bytes) w/ **seq=x**, buffer data segment, set timer
- **Wait Phase:** wait for ack from receiver w/ **ack= x+n**
 - if received ack w/ **ack=x+n**, set **x:=x+n**, and go to sending phase with next data segment
 - if time out, resend data segment w/ **seq=x**.
 - if received ack w/ **ack != x+n**, ignore (or resend data segment w/ **seq=x**)

Receiver algorithm:

Wait-for-Data:

wait for data packet with the (expected) **next-seq = x**

- if **received** Data packet w/ **seq. =x** and of size n bytes: send ACK pkt w/ **ack = x+n**; **set next-seq:= x+n**; go back to “Wait-for-Data”;
- If received Data packet w/ **seq != x**, (re-)send ACK pkt w/ **ack= next-seq**; go back to “Wait-for-Data”;

Q: what is the “state” information maintained at the sender & receiver, resp.?

SRDTP: Finite State Machine

●: state

event
action

: transition

Sender FSM

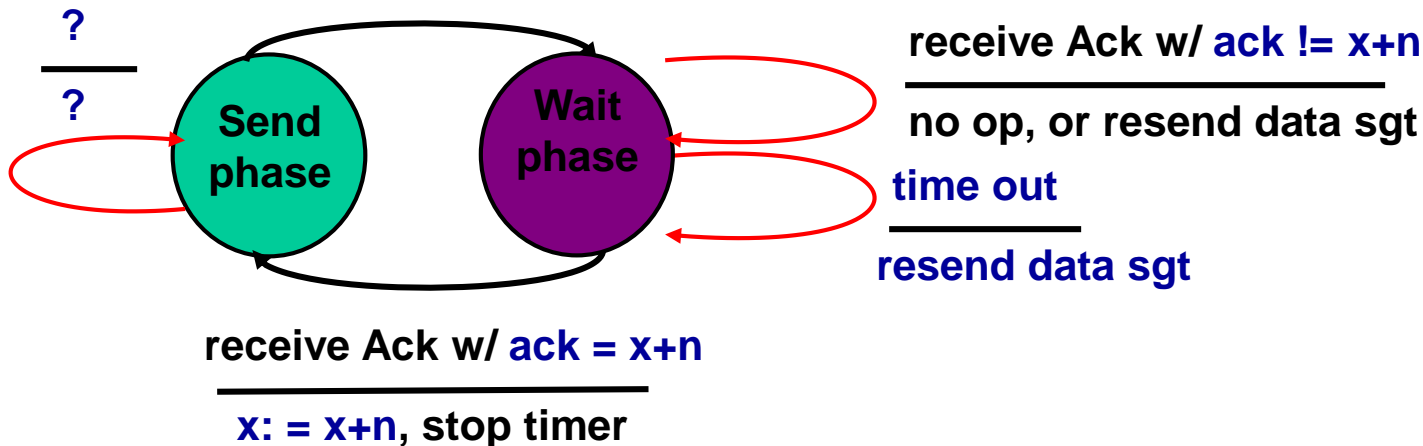
Receiver FSM?

Upper layer:

send data (n bytes)

make data sgt, seq = x, set timer

pass data sgt to lower layer



info ("state") maintained at sender:

phase it is in (*send*, or *wait*), ack expected, data sgt sent (seq #), timer

TCP Connection Set Up

Three way handshake:

TCP sender, receiver establish “connection” before exchanging data segments

- initialize TCP variables:
 - seq. #s
 - buffers, flow control info
- *client*: end host that initiates connection
- *server*: end host contacted by client

Step 1: client sends TCP **SYN** control segment to server

- specifies initial seq #

Step 2: server receives SYN, replies with **SYN+ACK** control segment

- ACKs received SYN
- specifies server → receiver initial seq. #

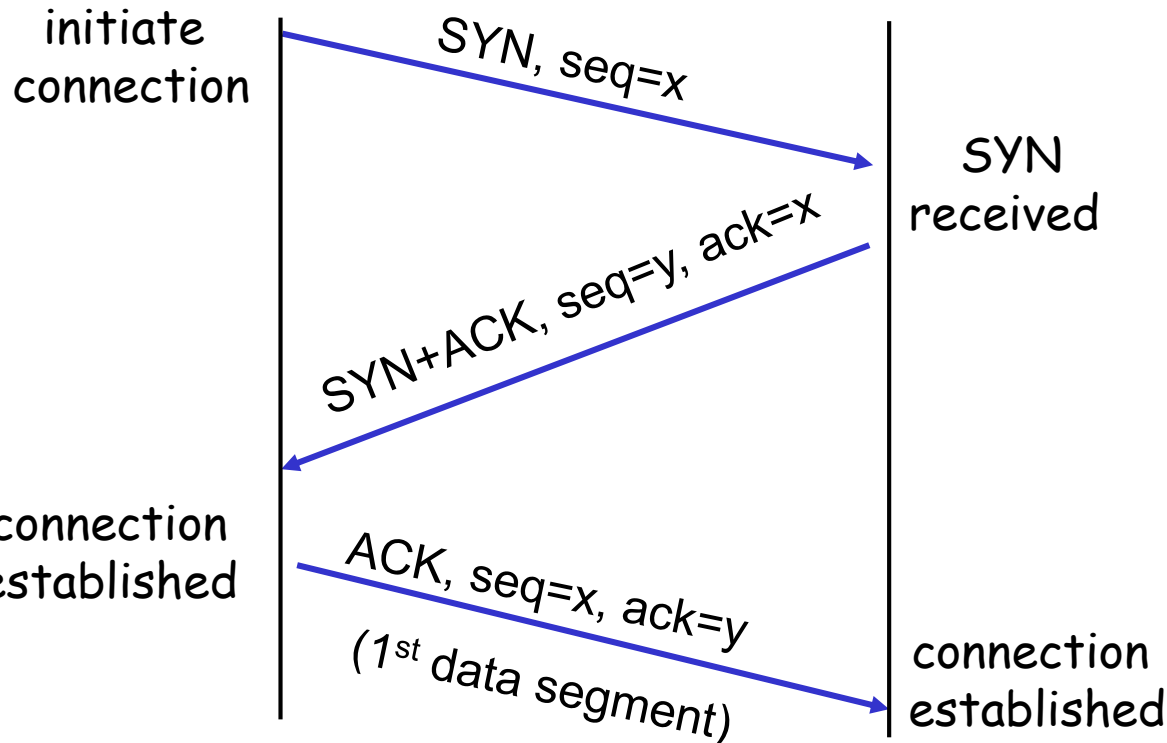
Step 3: client receives **SYN+ACK**, replies with **ACK** segment (which may contain 1st data segment)

TCP 3-Way Hand-Shake



client

server



Question:

- What kind of “state” client and server need to maintain?
- What initial sequence # should client (and server) use?

TCP: Some Key Issues

TCP Connection Set-up Design Questions

- ❑ Why Three-Way, not, say Two-Way, Handshake?
 - e.g., client: SYN(x); server: SYNACK(y,x)
- ❑ Why Do Both Client and Server Need to Choose a *Unique Seq. #* to number its byte stream?
 - Unique → each connection between the same source and destination host pair (and port number pair) must have a different sequence number
 - Recall: each TCP connection is identified by a 5-tuple <src IP, dst IP, src port, dst port, TCP>
- ❑ Problems: “old” duplicate packets (“ghost” packets) from previous (instantiation of the “same”) connection
 - can create “half-open” connection (that we want to avoid!)

Connection Setup Error Scenarios

- Lost (control) packets
 - What happen if SYN lost? client vs. server actions
 - What happen if SYN+ACK lost? client vs. server actions
 - What happen if ACK lost? client vs. server actions
- Duplicate (control) packets
 - What does server do if duplicate SYN received?
 - What does client do if duplicate SYN+ACK received?
 - What does server do if duplicate ACK received?

Connection Setup Error Scenarios (cont' d)

- Importance of (unique) initial seq. no.?
 - When receiving SYN, how does server know it's a new connection request?
 - When receiving SYN+ACK, how does client know it's a legitimate response, i.e., a response to its SYN request?
- Dealing with old duplicate (aka “ghost”) packets from old connections (or from malicious users)
 - If not careful: “TCP Hijacking” [i.e., Mitnick attack]
- How to choose unique initial seq. no.?
 - In practice: randomly choose a number (out of 2^{32} possible #'s) and add to last syn# used
- Other security concern:
 - “SYN Flood” -- denial-of-service attack

Announcement & Reminder (Oct 16)

- **Take-home quiz I: due Friday Oct 20 11:59pm**
 - Has been posted to the class mailing list earlier
 - Please submit via Moodle class website
- **Programming Project # 2 Due Sunday Oct 29 11:55pm**
 - TA Anas will provide a short overview in today's class.
- Hw #1 sample solutions (will be posted/shared with you)

What We Learned Last Time (Oct 9)

Reviewed: transport layer (UDP & TCP)

- multiplexing/de-multiplexing: src & dst port no.'s
- UDP: connectionless transport service - checksum, length
- TCP: connection-oriented, reliable service
 - seq #, ack #, special “flags” (SYN, ACK, FIN, RST)
 - connection management:
 - 3-way handshake set-up & timed wait shut-down
 - Key challenges: old, duplicate messages!

New material: reliable data transfer protocols

- A simply reliable data transfer protocol : stop & wait
- Efficiency of Protocol Design: stop-&-wait
- Pipelining Protocols: Go-Back-N and Selective Repeat

Reliable Data Transfer Protocols

Reliable Data Transfer Protocols

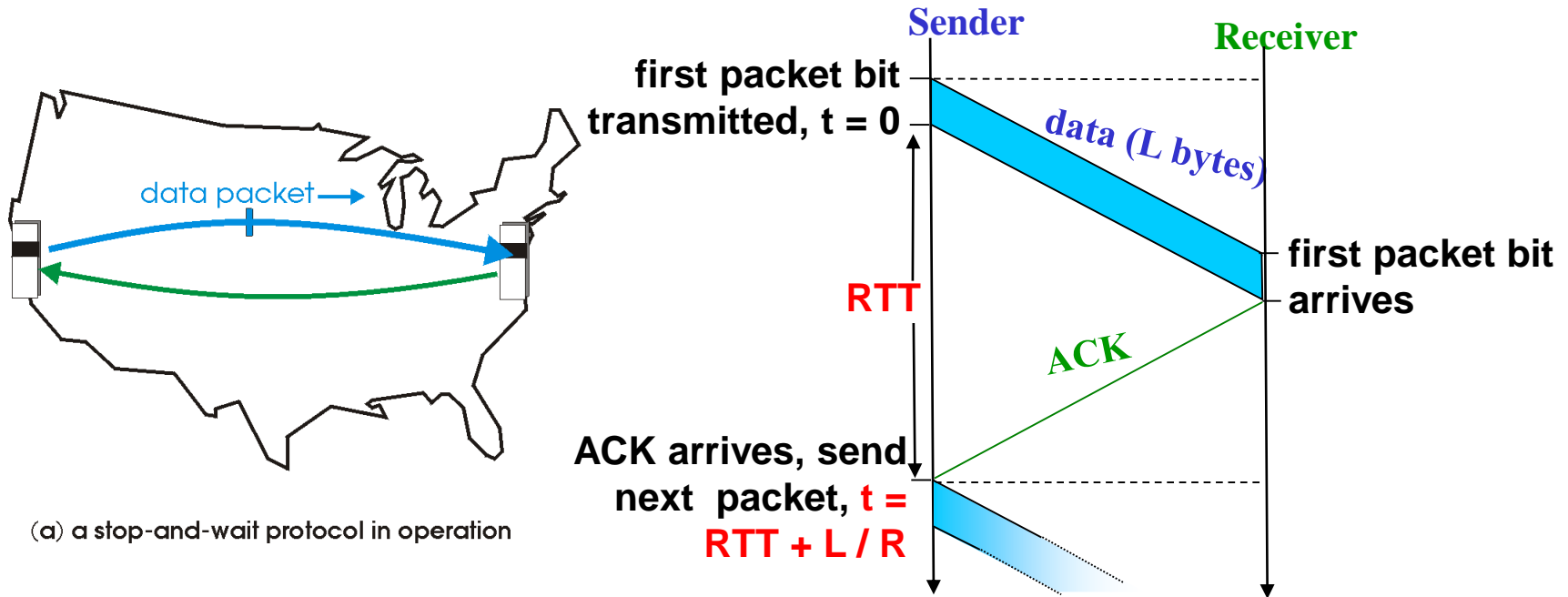
- **basic mechanisms: seq. no, ACK, timer, retransmission**
- Simplest protocol: Stop-&-Wait
 - To ensure correct operations of the protocol:
at least 1-bit (0 or 1) needed for seq. no. (why?)
- *More efficient reliable data transfer protocols*
 - What's the **problem with Stop & Wait protocol?**
 - Sliding window protocols: *Go-Back-N* and *Selective Repeat*
 - **concept of "sliding window"**
 - **sender algorithm:**
 - when to retransmit, when to send new packets? when to move window forward?
 - **receiver algorithm:**
 - when/what to acknowledge? when to move window forward?
 - when to buffer packets, and when to pass to upper layer?
 - **relationship between window size & seq. no. space**

Simple Reliable Data Transfer Protocol

“Stop-and-Wait” Protocol (rdt3.0 in the textbook)

- also called Alternating Bit Protocol
- Sender:
 - i) send data segment (n bytes) w/ $\text{seq} = x$
 - buffer data segment, set timer, retransmit if time out
 - ii) wait for ACK w/ $\text{ack} = x+n$; if received, set $x := x+n$, go to i)
 - retransmit if ACK w/ “incorrect” ack no. received
- Receiver:
 - i) expect data segment w/ $\text{seq} = x$; if received, send ACK w/ $\text{ack} = x+n$, set $x := x+n$, go to i)
 - if data segment w/ “incorrect” seq no received, discard data segment, and retransmit ACK.

Problem with Stop & Wait Protocol



- Can't keep the pipe full
 - Utilization is low
 - when bandwidth-delay product ($R \times RTT$) is large!

Stop & Wait: Performance Analysis

Example:

1 Gbps connection, 15 ms end-end prop. delay,

data segment size: 1 KB = 8Kb; ack segment size: assume negligible

$$T_{\text{transmit}} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8 \text{ kb}}{10^9 \text{ b/s}}$$
$$= 8 \times 10^{-6} \text{ s} = 0.008 \text{ ms}$$

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{L}{RTT * R + L} = \frac{.008}{30.008} = 0.00027$$

- U_{sender} : utilization, i.e., fraction of time sender busy sending
- 1KB data segment every 30 msec (round trip time)
 - > $0.027\% \times 1 \text{ Gbps} = 33\text{KB/sec}$ throughput over 1 Gbps link

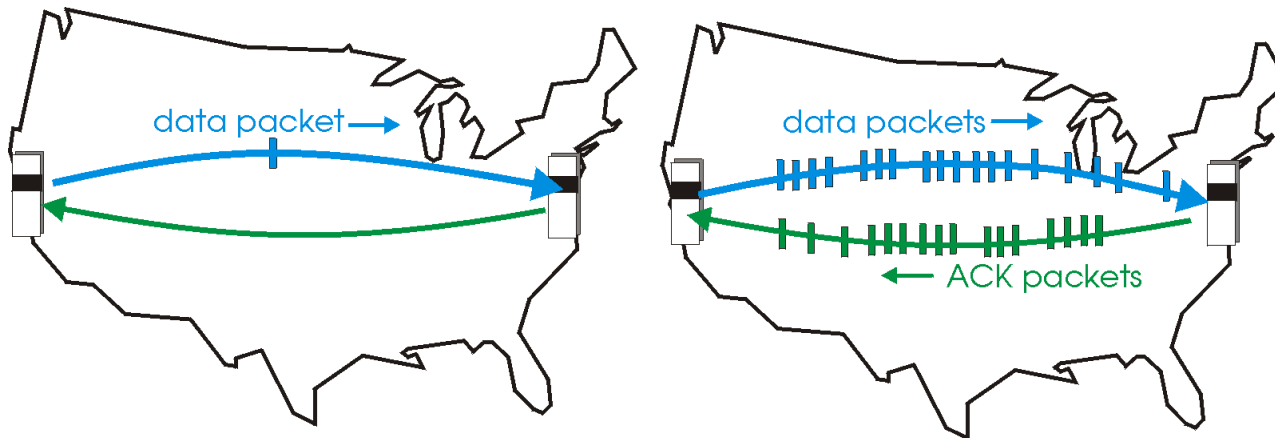
Moral of story:

network protocol *limits* use of physical resources!

Pipelined Protocols

Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged data segments

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

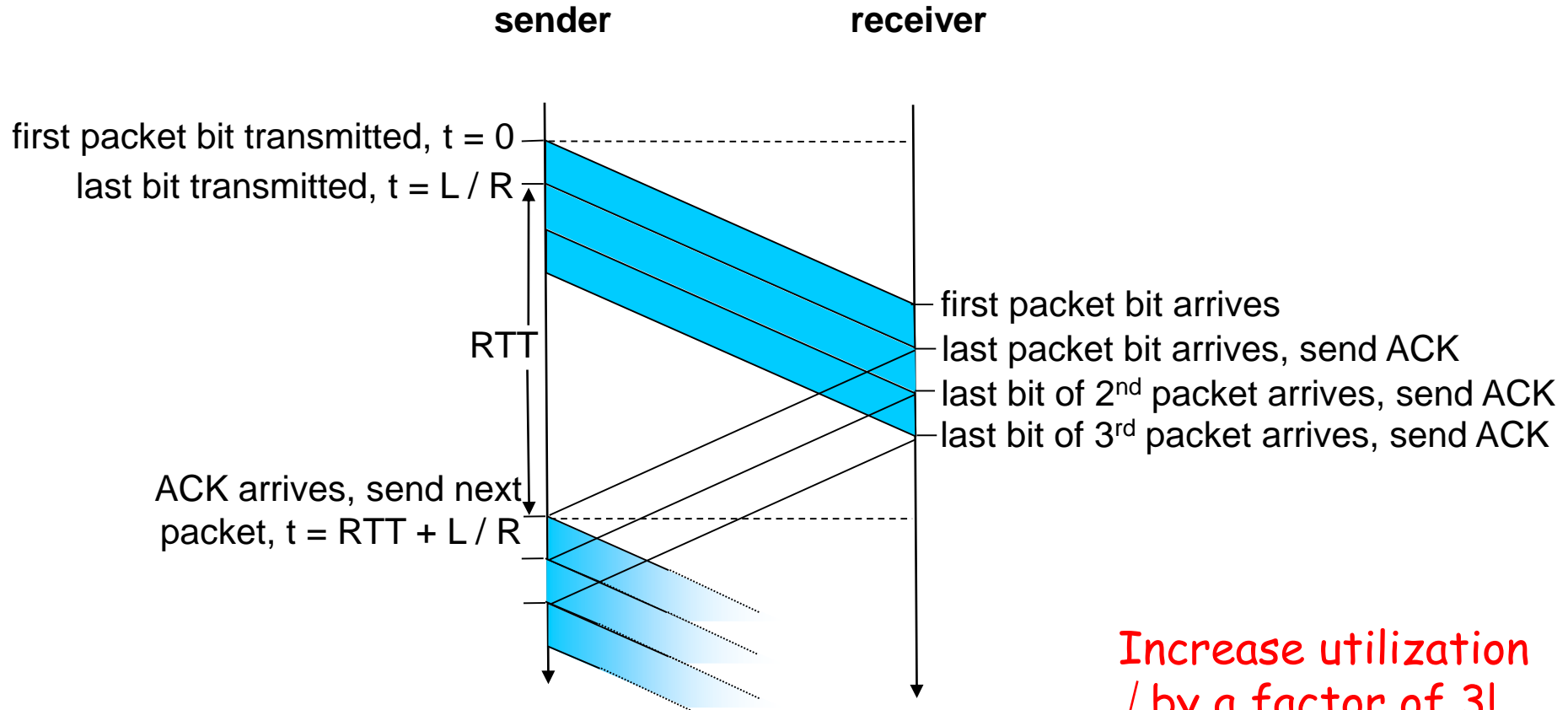


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Two generic forms of pipelined protocols:
Go-Back-N and Selective Repeat

Pipelining: Increased Utilization



Increase utilization
by a factor of 3!

$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Go-Back-N: Basic Ideas

Sender:

- Packets transmitted continually (when available) without waiting for ACK, up to N outstanding, unACK'ed packets
- A logically different timer associated with each “in-flight” (i.e., unACK'ed) packet
 - *timeout(n)*: retransmit pkt n and all higher seq # pkts in window

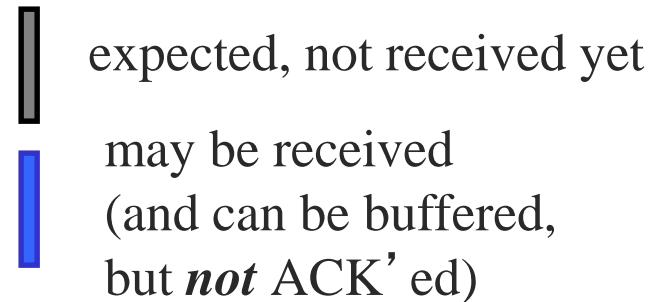
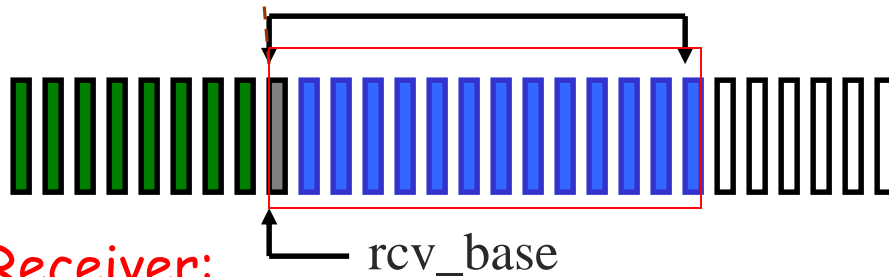
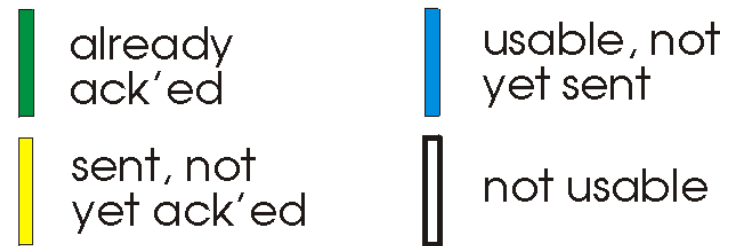
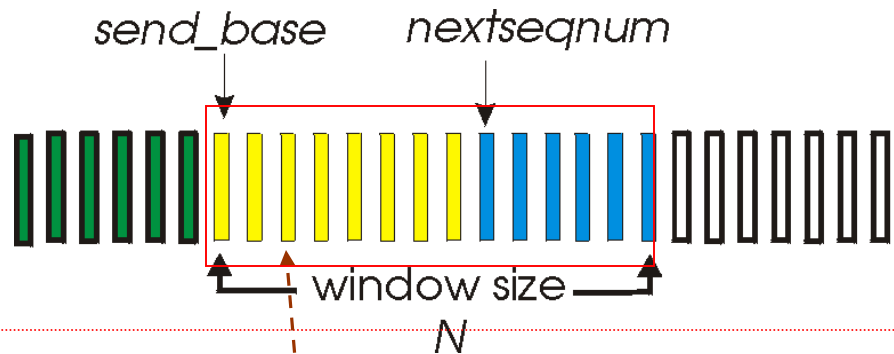
Receiver:

- ACK packet if corrected received and in-order, pass to higher layer, NACK or ignore corrupted or out-of-order packets
- “cumulative” ACK: if multiple packets received corrected and in-order, send only one ACK with ack= next expected seq no.

Go-Back-N: Sliding Windows

Sender:

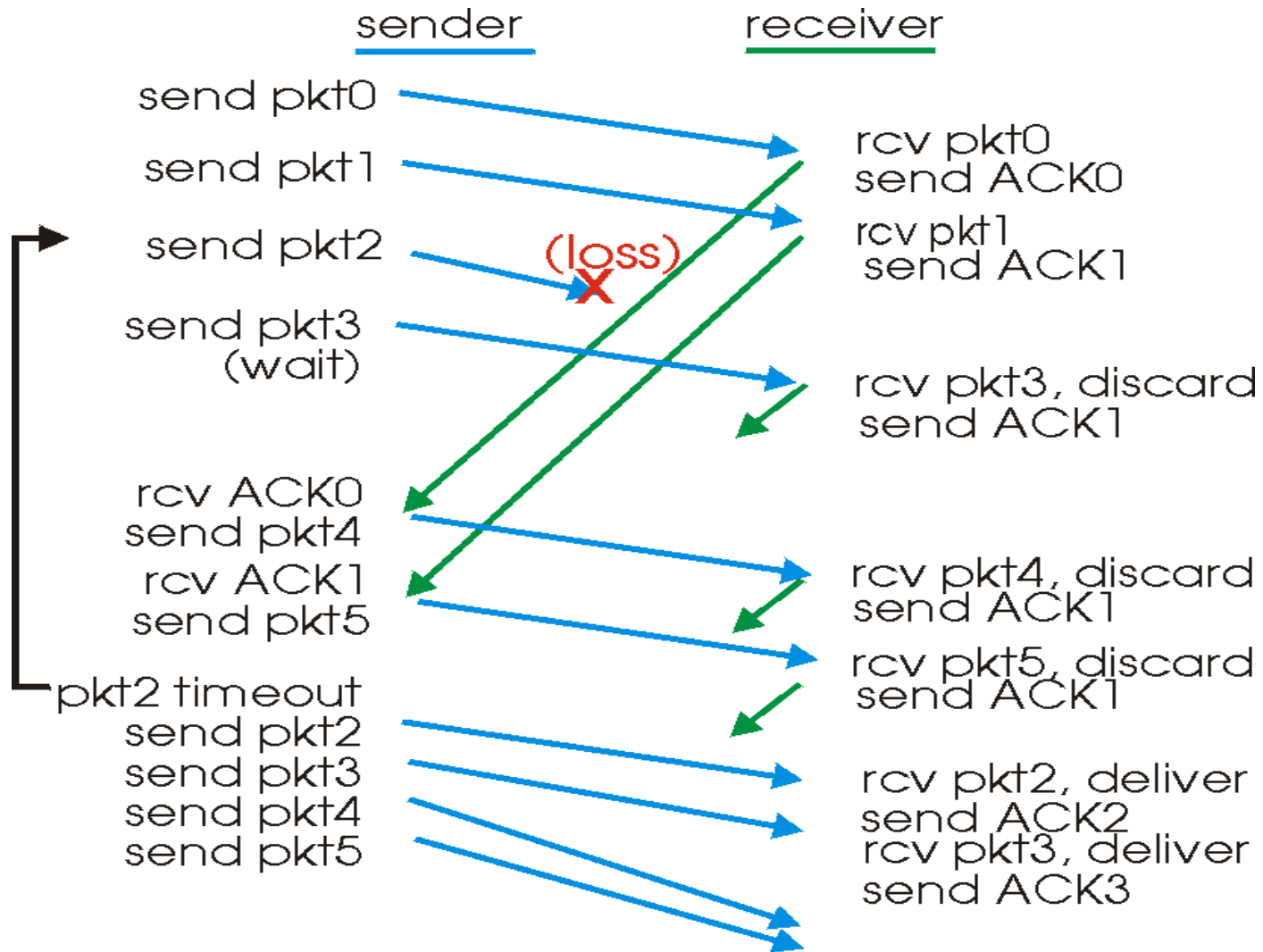
- “window” of up to N , consecutive unack’ed pkts allowed
- **send_base**: first sent but unACKed pkt, move forward when ACK’ed



Receiver:

- **rcv_base**: keep track of next expected seq no, move forward when next in-order (i.e., w/ expected seq no) pkt received

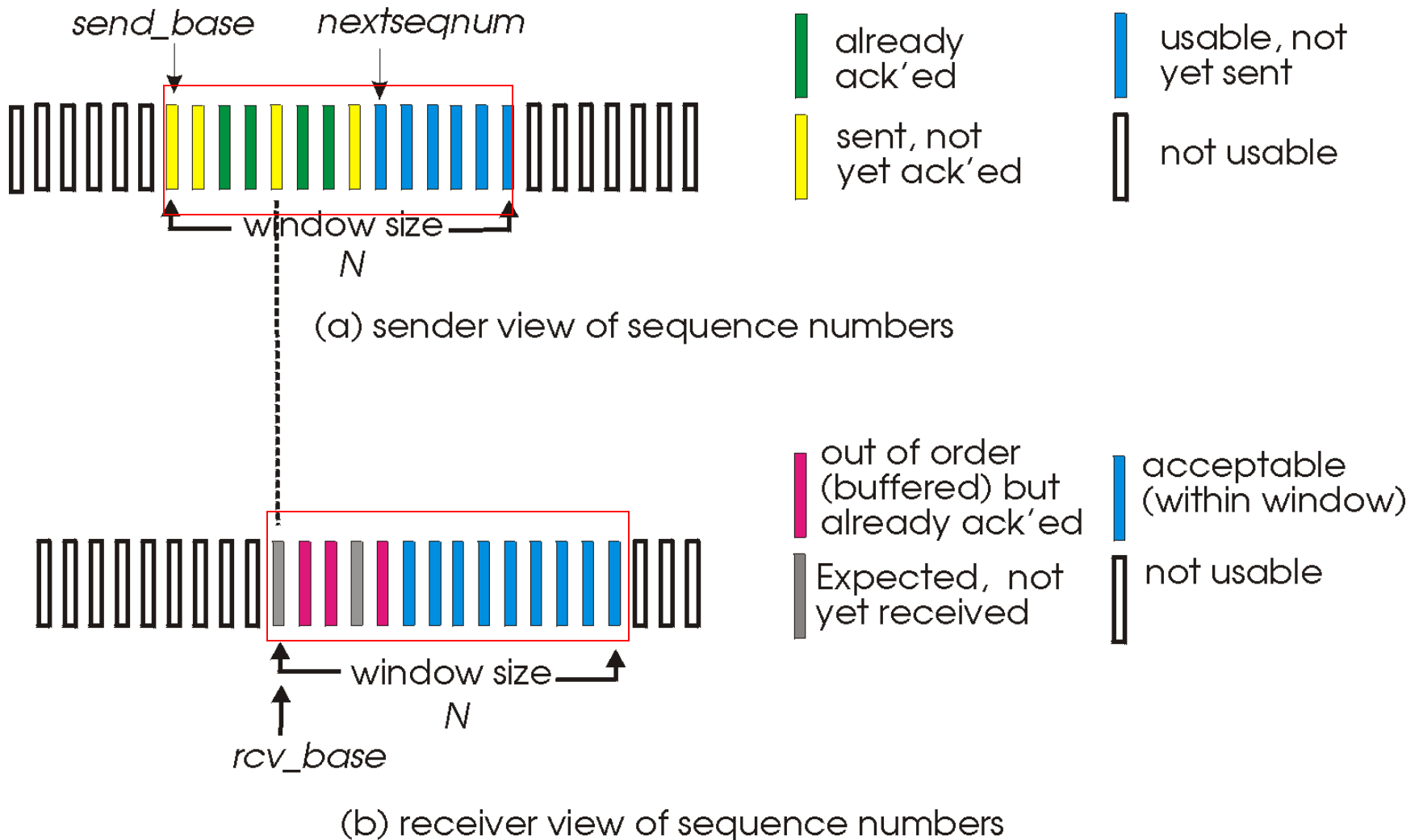
GBN in Action



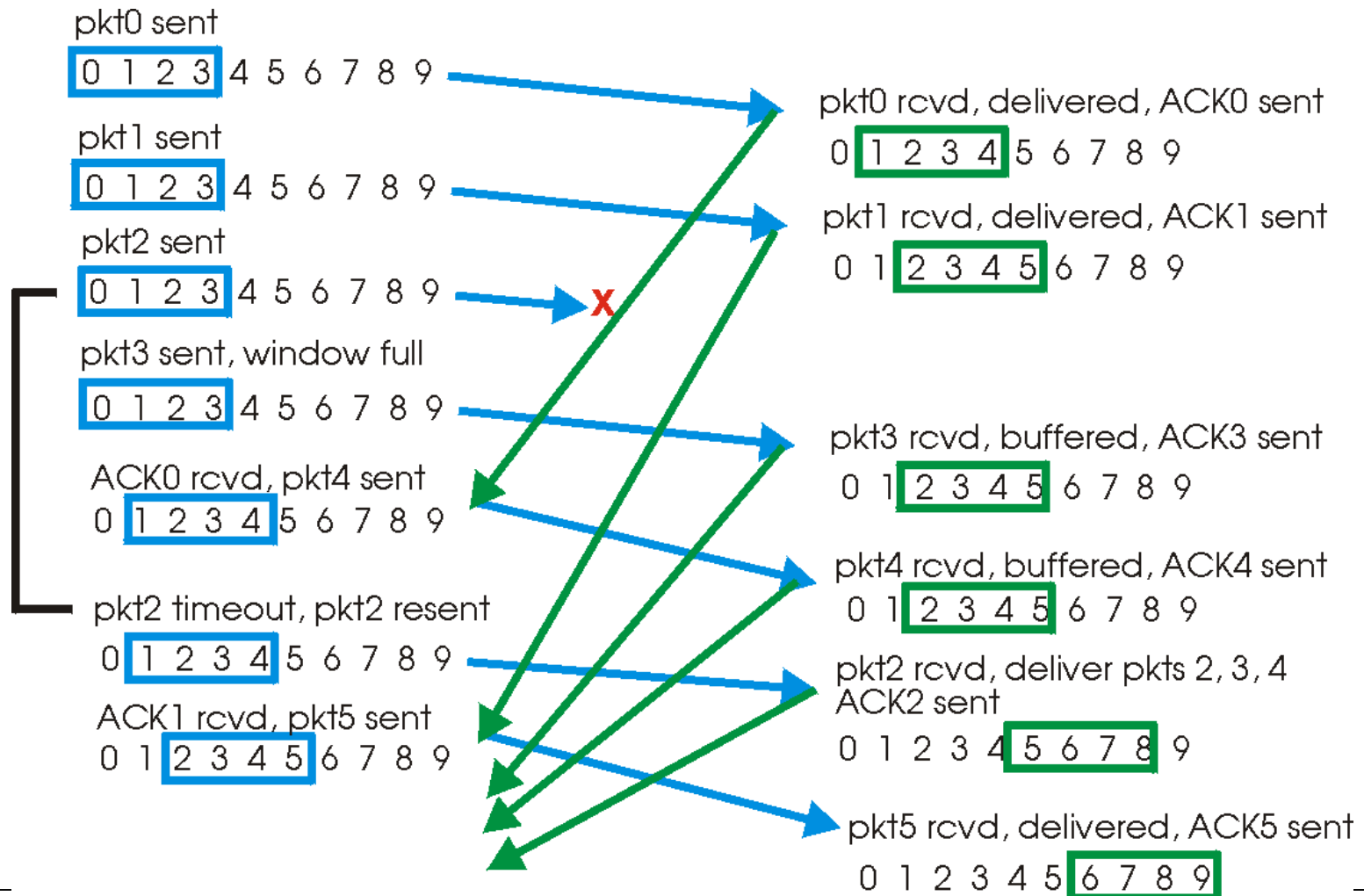
Selective Repeat

- As in Go-Back-N
 - Packet sent when available up to window limit
- Unlike Go-Back-N
 - Out-of-order (but otherwise correct) is ACKed
 - Receiver: buffer out-of-order pkts, no “cumulative” ACKs
 - Sender: on timeout of packet k, retransmit just pkt k
- Comments
 - Can require more receiver buffering than Go-Back-N
 - More complicated buffer management by both sides
 - Save bandwidth
 - no need to retransmit correctly received packets

Selective Repeat: Sliding Windows



Selective Repeat in Action



Seqno Space and Window Size

Check out the companion website (for the 6th edition) for GBN and SR Applets:

GBN:

http://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_6/video_applets/GBNindex.html

SR:

http://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_6/video_applets/SRindex.html

OR check out the interactive Animation
on the 7th edition website

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/interactive-animations-cw.php

Seqno Space and Window Size

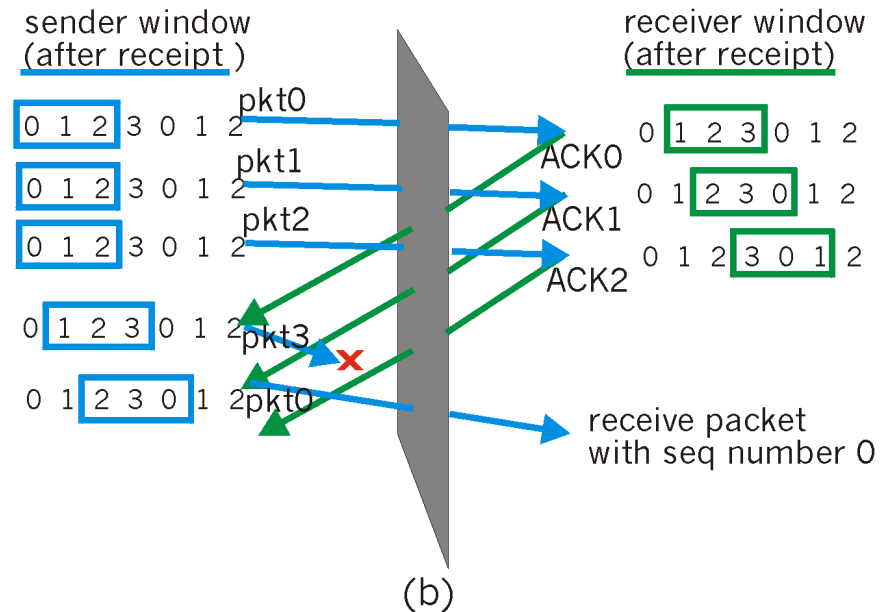
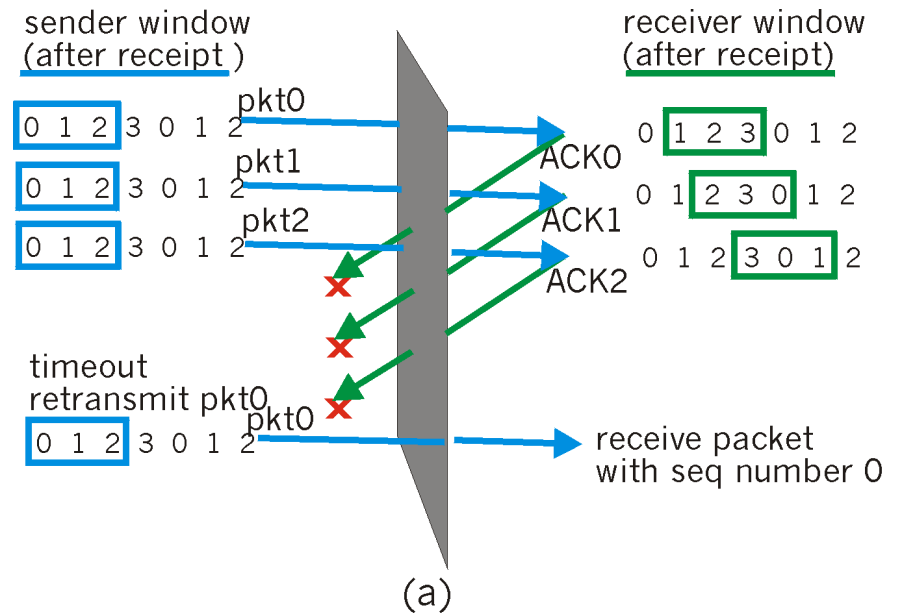
- How big the sliding window (W) can be?
 - MAXSEQNO S : number of available sequence #'s
 - $S = 2^s$ if s bits are used to present the sequence #'s
 - Go-Back-N: $W=S$ won't work! $W \leq S - 1$
 - What about Selective Repeat? $W \leq S/2$
- Assumptions about the underlying network are important!
 - If two machines are *directly* connected via a physical link, the network may corrupt or lose packets, *but cannot deliver two packets successfully but out-of-order*
 - GBN: $W = S - 1$
 - SR: $W = S/2$

Selective Repeat: Dilemma

Example:

- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?



Seqno Space and Window Size

- How big the sliding window (W) can be?
- Assumptions about the underlying network are important!
 - If two machines are directly connected via a physical link, the network may corrupt or lose packets, *but cannot deliver two packets successfully but out-of-order*
 - GBN: $W=S-1$; SR: $W= S/2$
 - If two machines are connected via a general network (more than one link), the underlying network may not only corrupt or lose packets, *but also deliver two packets successfully but out-of-order, what should W be?*
 - **Key Principle:** W must be (small enough relative to S) such that there can't an old packet and a new packet sent by the sender carrying the same sequence # inside the network or at the receiver ! *→ otherwise, the receiver will be confused!*