

Internet Overview

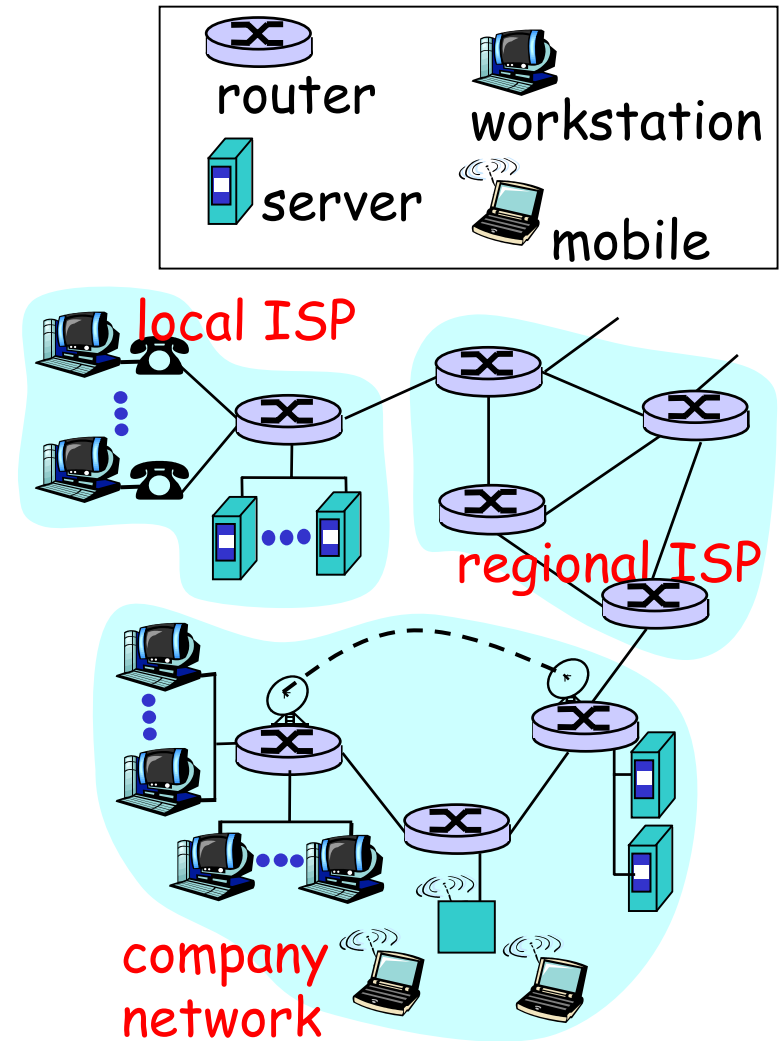
Stuart D. Milner, Ph.D.
Clark School of Engineering
Institute for Systems Research
April 9 and 11, 2002

Acknowledgement:

The briefing slides from: Kurose, J.F. and Ross, K.W. Computer Networking: A Top Down Approach Featuring the Internet. Addison-Wesley, 2001.

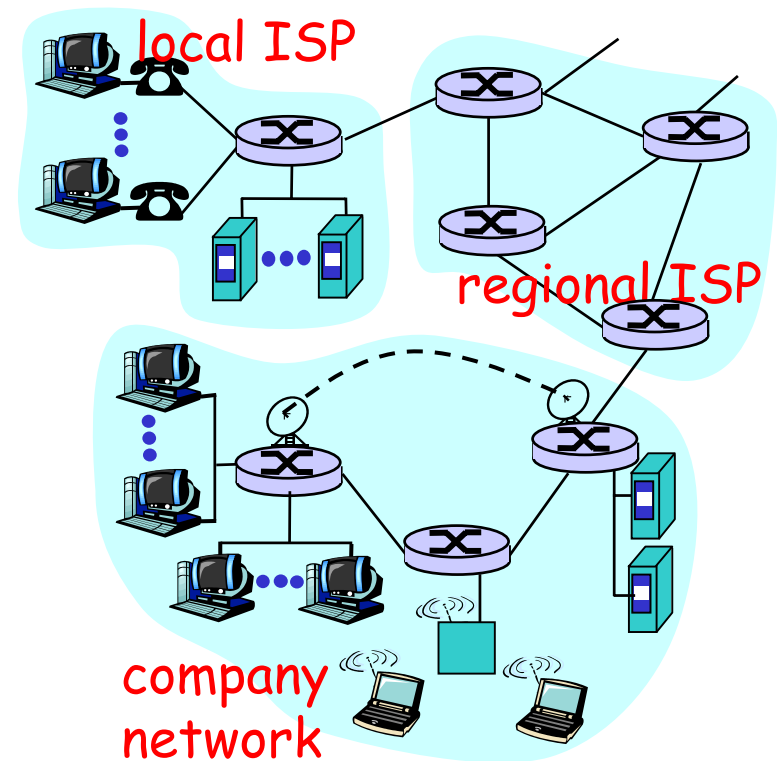
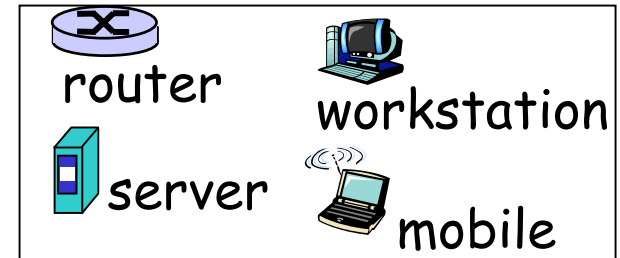
What's the Internet: Hardware and Software

- > 50M connected computing devices (>100M users): *hosts, end-systems*
 - pc's workstations, servers
 - PDA's, phones
 - Emerging convergence of **telephony, wireless** ("bluetooth" and 802.11); and **deeply embedded systems** (e.g, toasters, sensors)
running *network apps*
- *communication links*
 - fiber, copper, radio, satellite, wireless optical
- *routers*: forward packets (chunks) of data thru network



What's the Internet: Hardware and Software

- *protocols*: control sending, receiving of msgs
 - e.g., TCP, IP, HTTP, FTP, PPP
- *Internet: "network of networks"*
 - hierarchical topology of networks connected via backbones
 - public **I**nternet versus private intranet
- Internet standards
 - IETF: Internet Engineering Task Force



The network edge:

□ end systems (hosts):

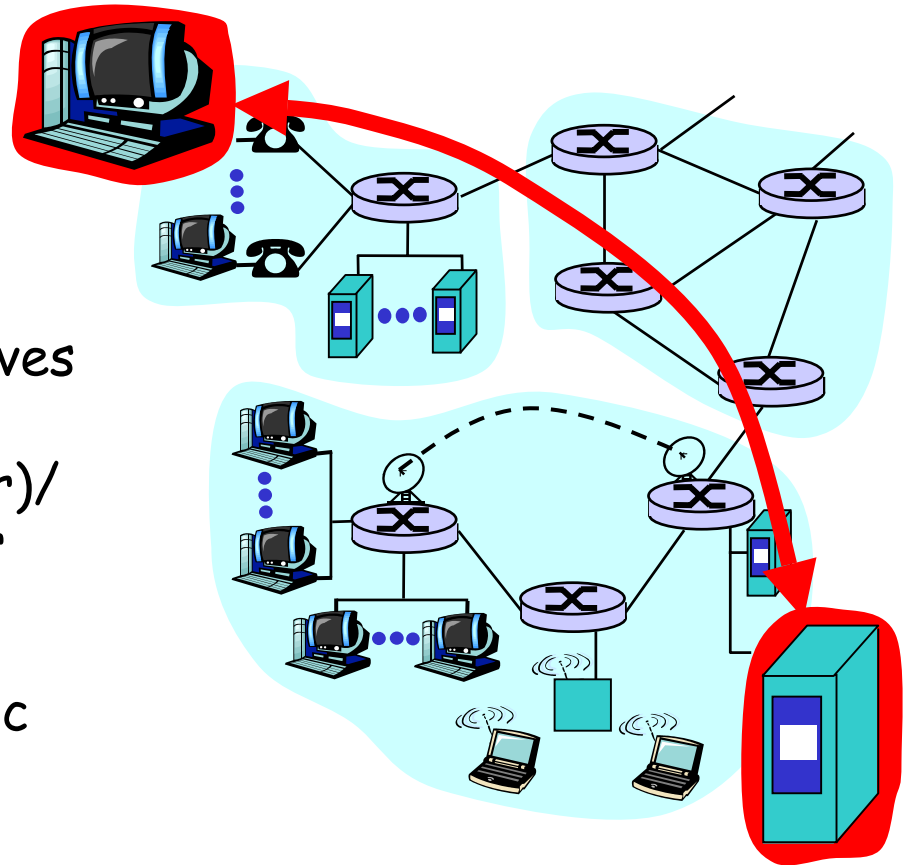
- run application programs
- e.g., WWW, email
- at "edge of network"

□ client/server model

- client host requests, receives service from server
- e.g., WWW client (browser)/server; email client/server

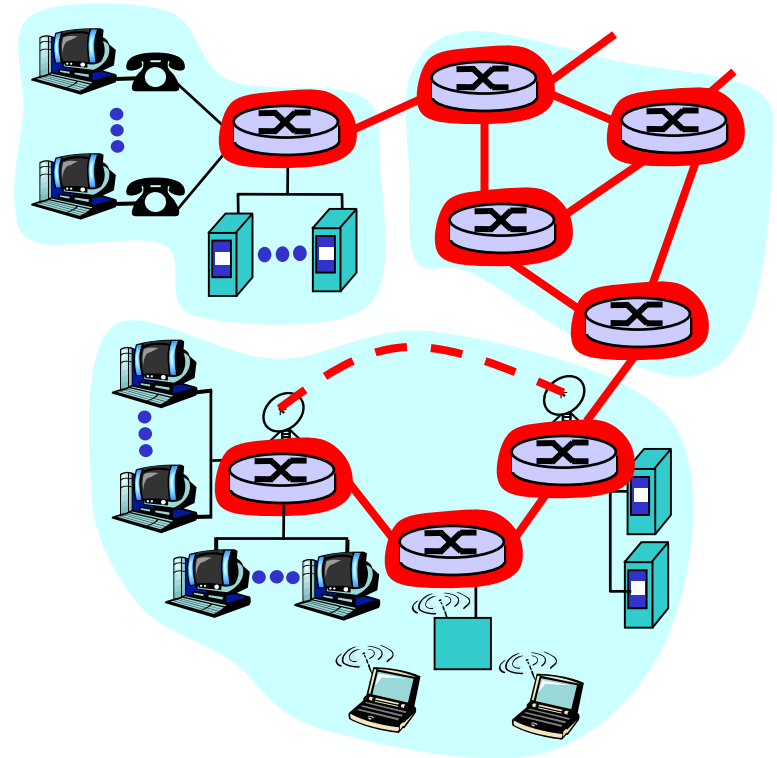
□ peer-peer model:

- host interaction symmetric
- e.g.: teleconferencing



The Network Core

- mesh of interconnected routers
- **the fundamental question:** how is data transferred through net?
 - **circuit switching:** dedicated circuit per call: telephone net
 - **packet-switching:** data sent thru net in discrete "chunks"



What's a protocol?

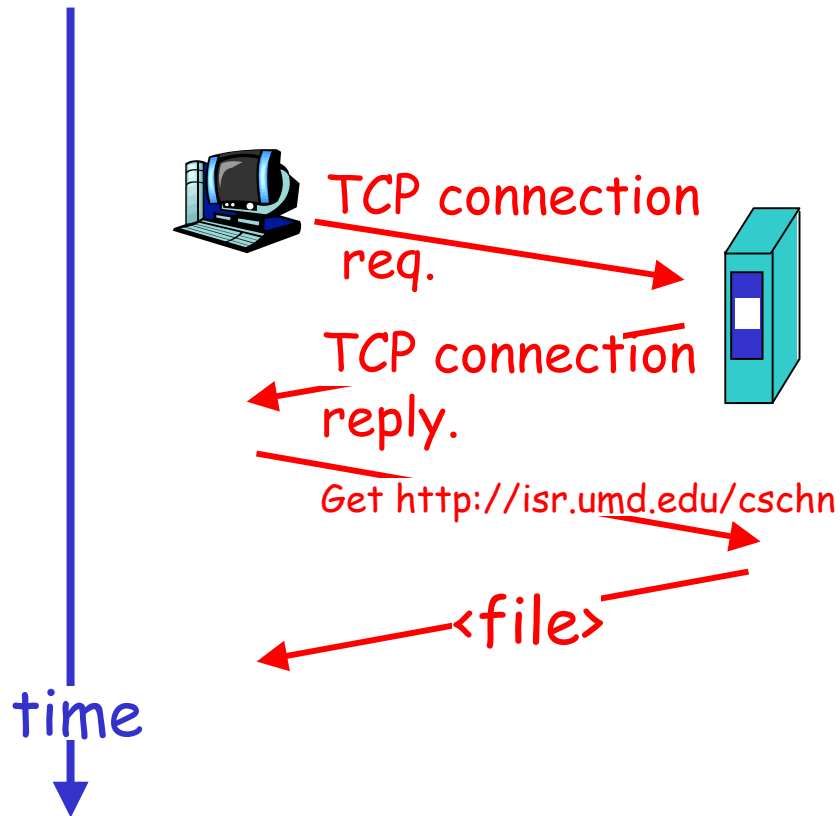
Examples:

Routers: packet path from source to destination

network interface card: control flow of bits on the "wire"

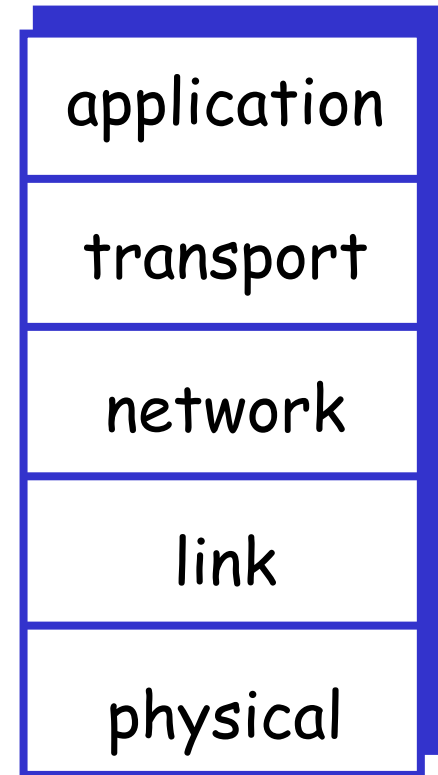
Host computers: control congestion and rate at which pkts. transmitted between sender and receiver

Everywhere in the Internet!



Internet protocol stack

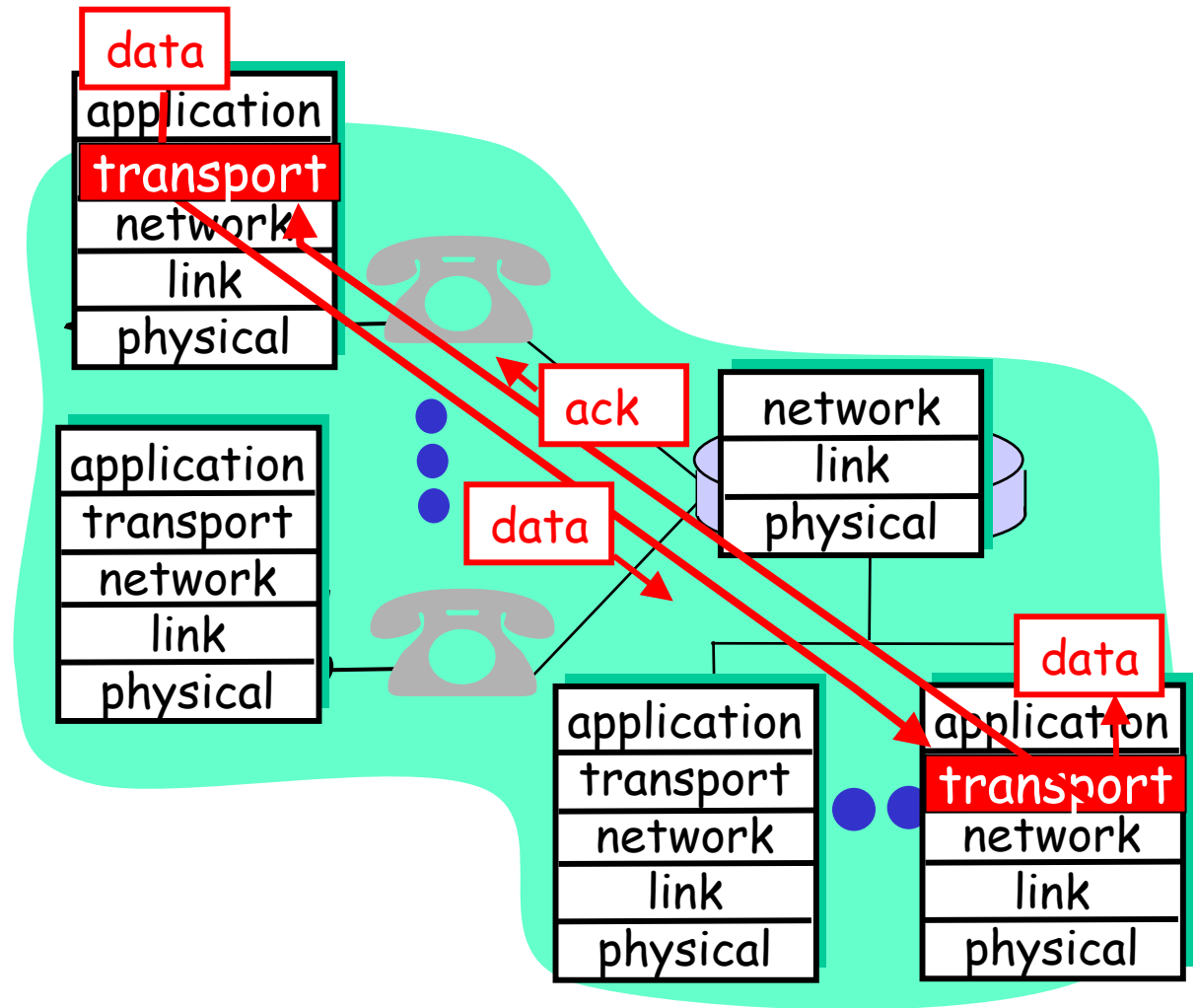
- ❑ **application:** supporting network applications
 - ftp, smtp, http
- ❑ **transport:** host-host data transfer, congestion control, segmentation
 - tcp, udp
- ❑ **network:** routing of datagrams from source to destination
 - ip, routing protocols
- ❑ **link:** data transfer between neighboring network elements
 - ppp, ethernet
- ❑ **physical:** bits "on the wire"



Layering: logical communication

E.g.: transport

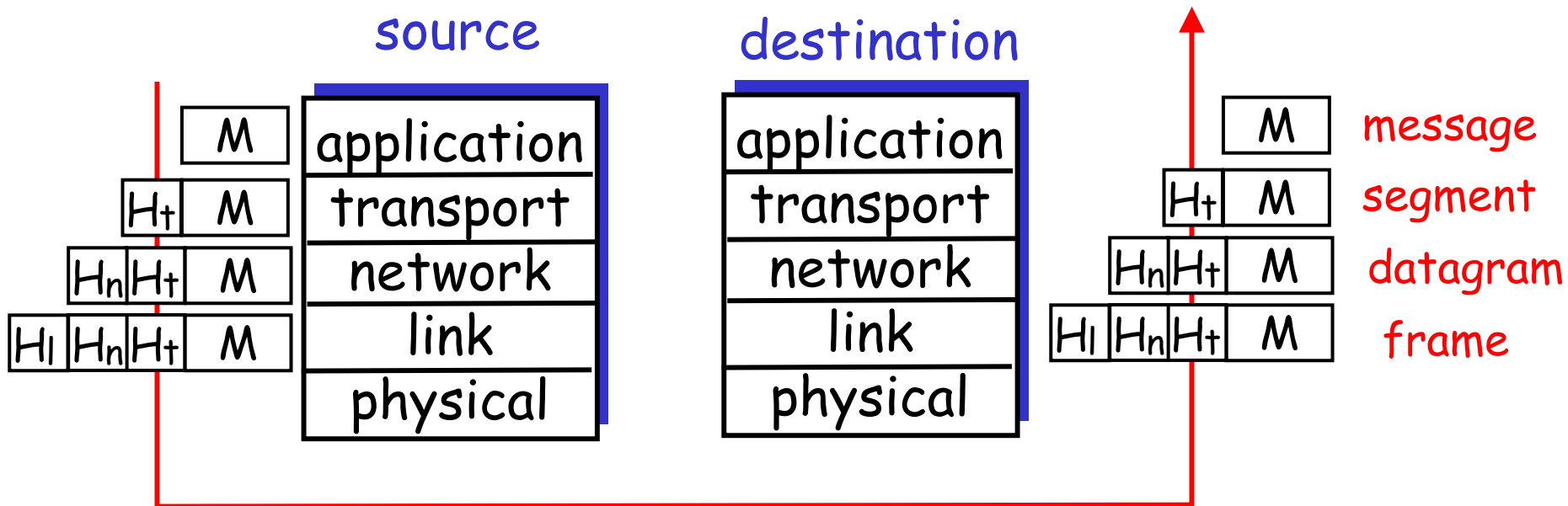
- ❑ take data from app
- ❑ add addressing, reliability check info to form "datagram"
- ❑ send datagram to peer
- ❑ wait for peer to ack receipt



Protocol layering and data: protocol data units (PDUs)

Each layer takes data from above (**SERVICE MODEL**)

- adds header information to create new data unit
- passes new data unit to layer below



SERVICE MODEL

- ❑ Layer n-1 offers SERVICES to Layer n
- ❑ For example:
 - Layer n-1 guarantees that n-PDU will arrive without error at Layer n in the destination within 1 second
 - Or, Layer n-1 might only guarantee that n-PDU will eventually arrive at destination without assurances about error

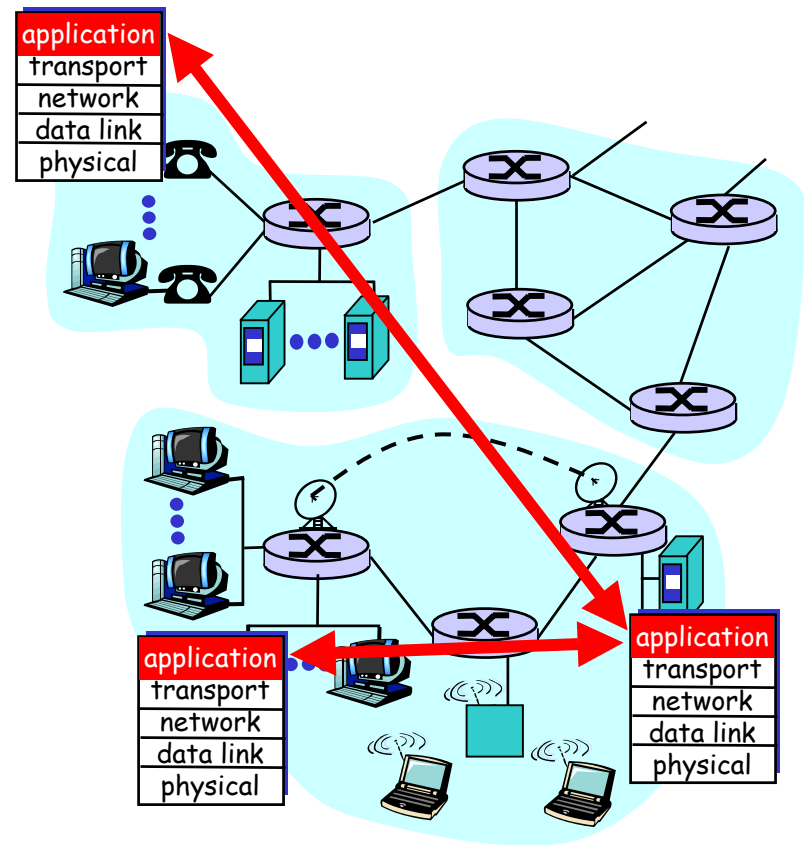
I. Applications and application-layer protocols

Application: communicating, distributed processes

- running in network hosts in "user space"
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- user services provided by lower layer protocols



Client-server paradigm

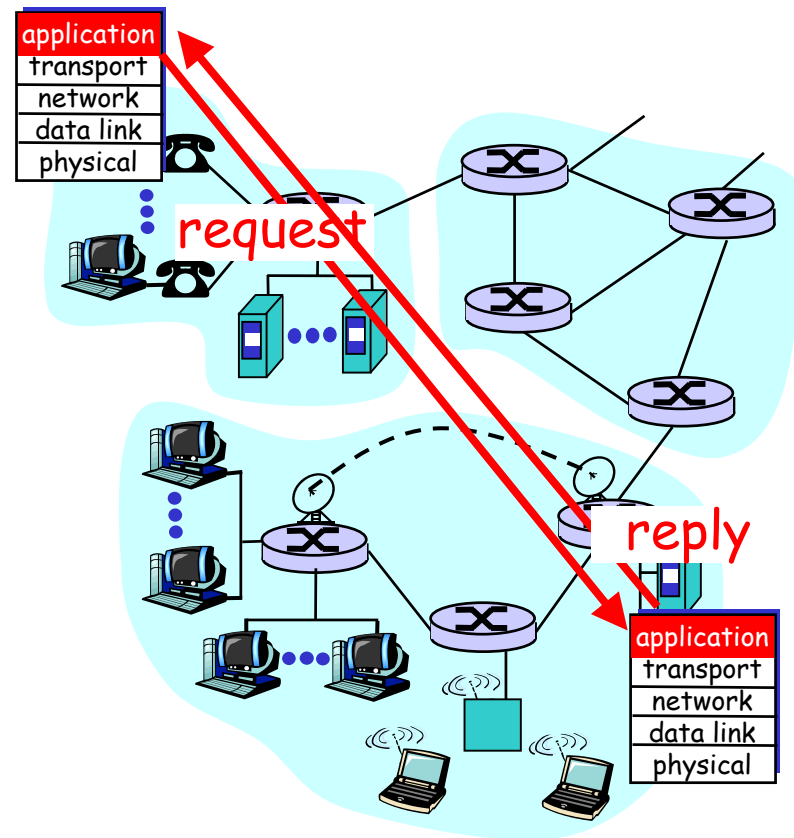
Typical network app has two pieces: *client* and *server*

Client:

- ❑ initiates contact with server ("speaks first")
- ❑ typically requests service from server,
- ❑ for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- ❑ provides requested service to client
- ❑ e.g., Web server sends requested Web page, mail server delivers e-mail



Application-layer protocols (cont).

API: application programming interface

- defines interface between application and transport layer
- socket: Internet API
 - two processes communicate by sending data into socket, reading data out of socket
 - interface between application and transport layers

Q: how does a process "identify" the other process with which it wants to communicate?

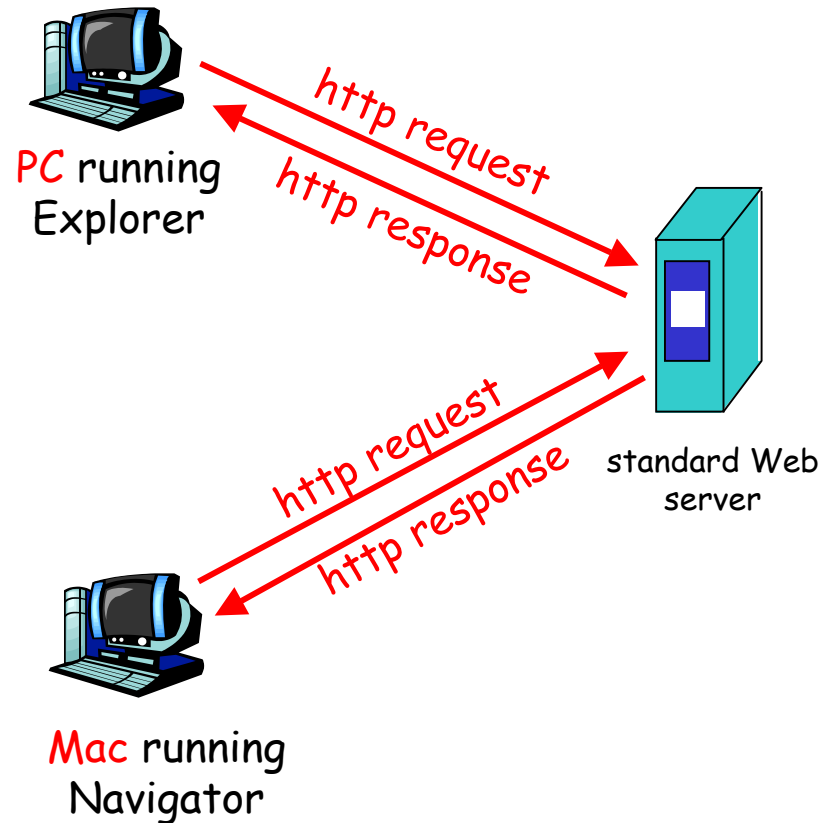
- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered
 - # 80 for HTTP
 - # 25 for SMTP
 - RFC 1700

... more on this later.

The Web: the http protocol

http: hypertext transfer protocol

- Web's application layer protocol
- client/server model
 - *client*: browser that requests, receives, "displays" Web objects
 - *server*: Web server sends objects in response to requests



http example

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

(contains text,
references to 10
jpeg images)

1a. http client initiates TCP connection to http server (process) at www.someSchool.edu. Port 80 is default for http server.

1b. http server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (someDepartment/home.index), sends message into socket

time
↓

http example (cont.)

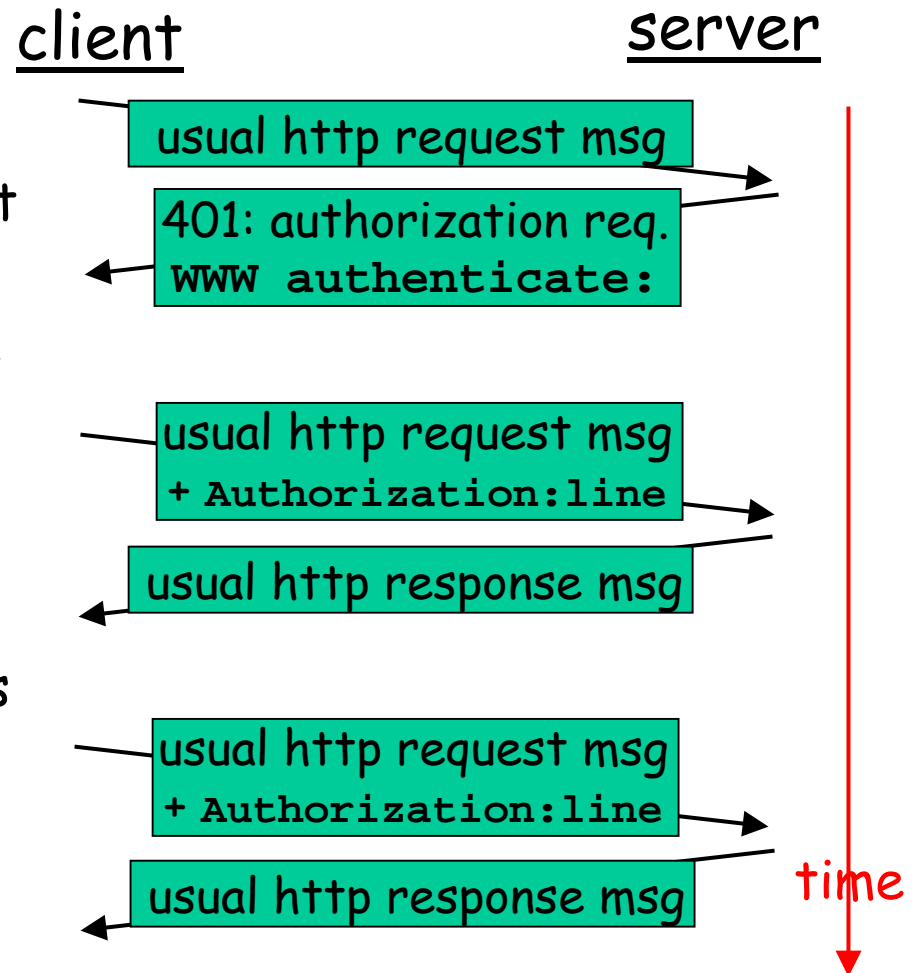
4. http server closes TCP connection.
5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects
6. Steps 1-5 repeated for each of 10 jpeg objects

time

User-server interaction: authentication

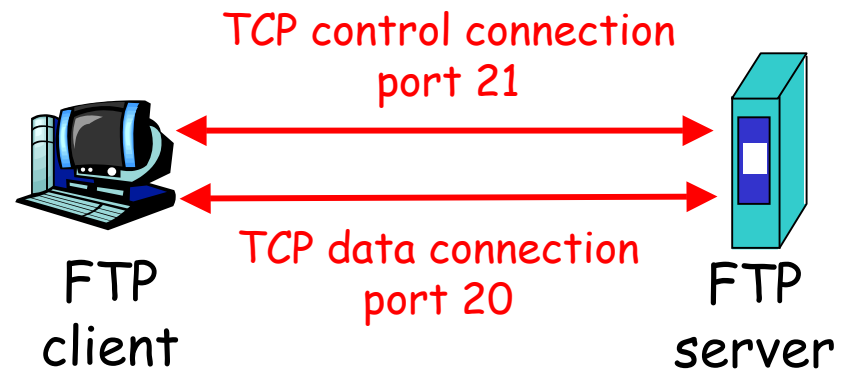
- Authentication goal:** control access to server documents
- ❑ **stateless:** client must present authorization in each request
 - ❑ authorization: typically name, password
 - authorization: header line in request
 - if no authorization presented, server refuses access, sends
WWW authenticate:
header line in response

Browser caches name & password so that user does not have to repeatedly enter it.



ftp: separate control, data connections

- ❑ ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- ❑ two parallel TCP connections opened:
 - **control**: exchange commands, responses between client, server.
"out of band control"
 - **data**: file data to/from server
- ❑ ftp server maintains "state": current directory, earlier authentication



DNS: Domain Name System

Internet hosts, routers
identified by:

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

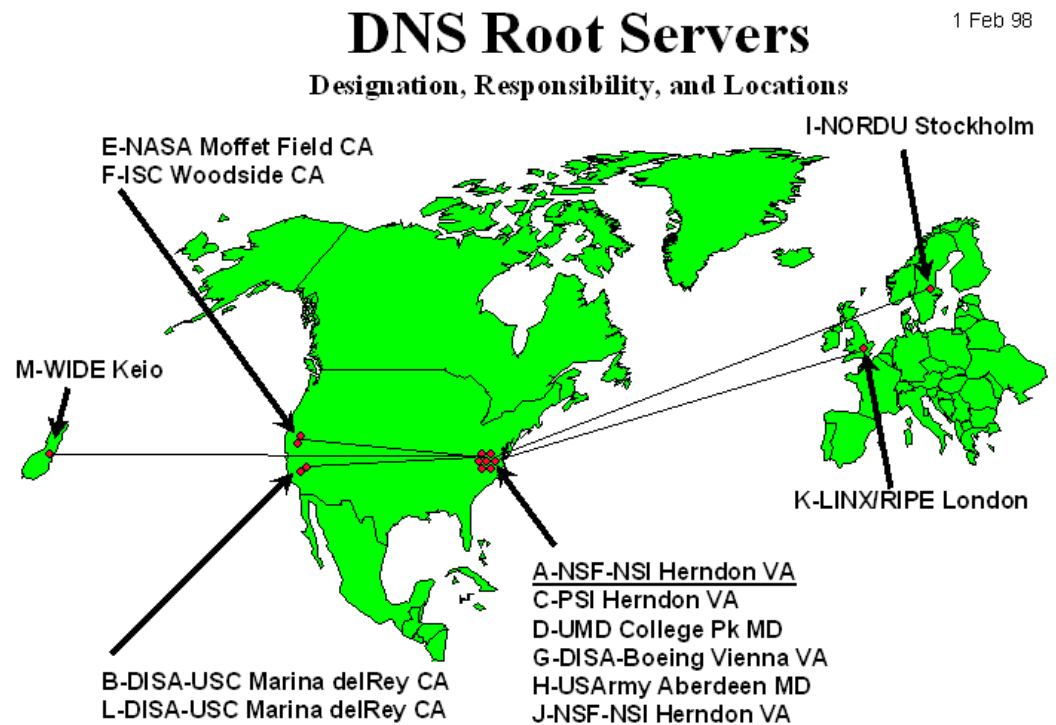
Directory Service
translates host names
to IP addresses

Domain Name System:

- *distributed database*
implemented in hierarchy of
many *name servers*
- *application-layer protocol*
host, routers, name servers to
communicate to *resolve* names
(address/name translation)
 - note: core Internet function
implemented as application-layer
protocol
 - complexity at network's "edge"
 - commonly used by other
application layer protocols (e.g.,
http)

Root name servers

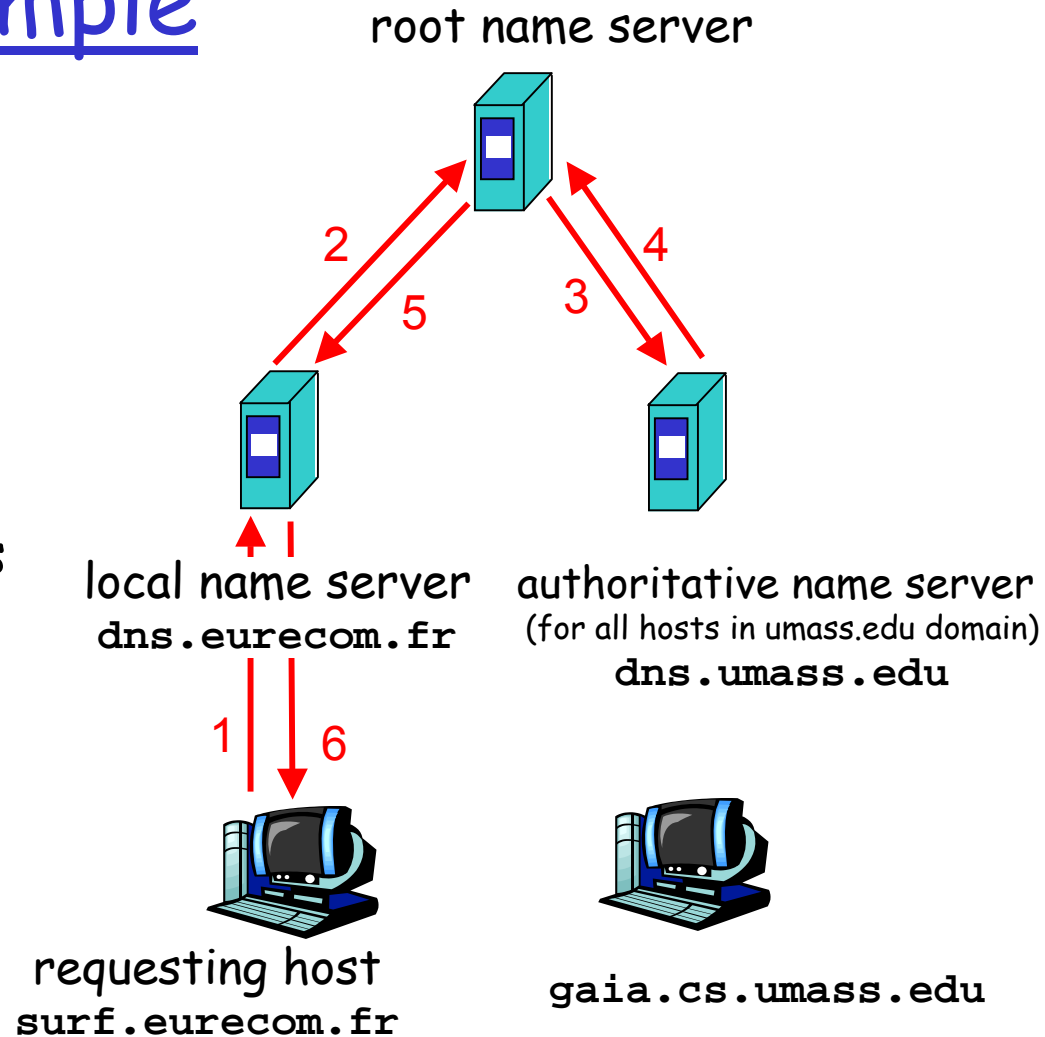
- contacted by local name server that can not resolve name
- root name server:
 - contacts authoritative name server if name mapping not known
 - gets mapping
 - returns mapping to local name server
- ~ dozen root name servers worldwide



Simple DNS example

host `surf.eurecom.fr`
wants IP address of
`gaia.cs.umass.edu`

1. Contacts its local DNS server, `dns.eurecom.fr`
2. `dns.eurecom.fr` contacts root name server, if necessary
3. root name server contacts authoritative name server, `dns.umass.edu`, if necessary



Socket programming

How client/server applications/processes communicate using sockets

Socket API

- ❑ introduced in BSD4.1 UNIX, 1981
- ❑ explicitly created, used, released by apps
- ❑ client/server paradigm
 - implementation of a protocol defined by an RFC (e.g., FTP, Netscape)
 - "proprietary" client server application not necessarily conforming to an RFC (MSN and Erols security/authentication?)
- ❑ two types of transport service via socket API:
 - unreliable datagram --UDP
 - reliable, byte stream-oriented - TCP

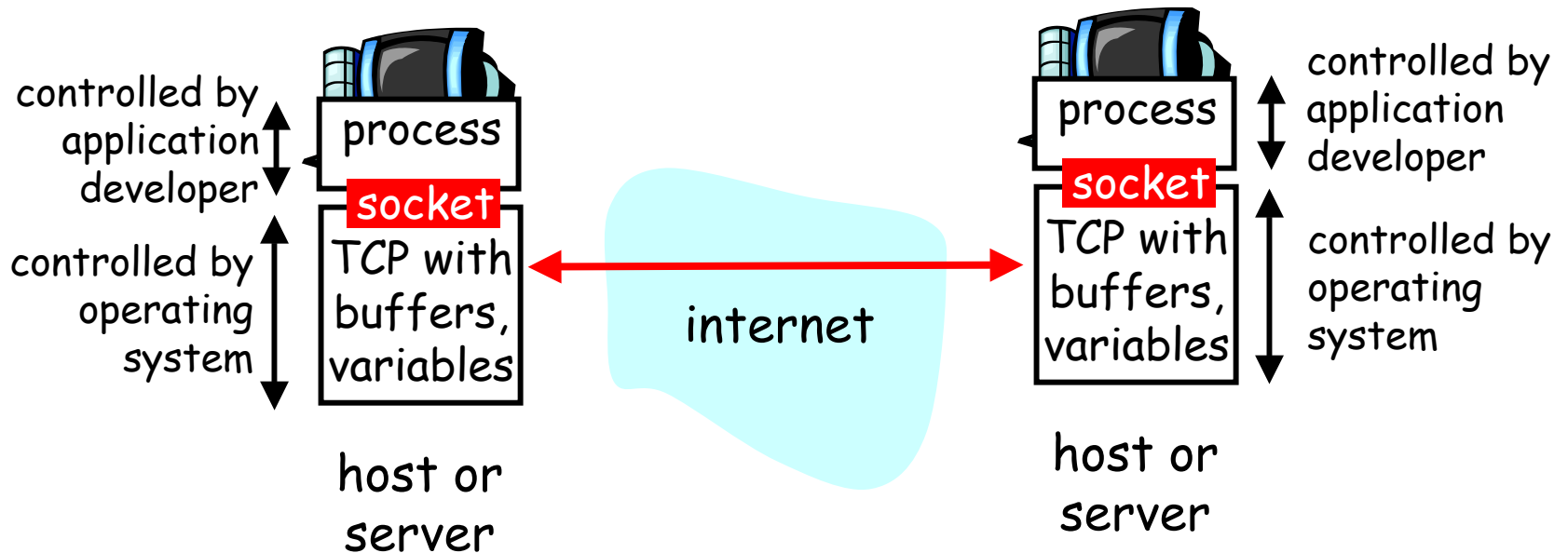
socket

a *host-local, application-created/owned, OS-controlled* interface (a "door") into which application process can **both send and receive** messages to/from another (remote or local) application process

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

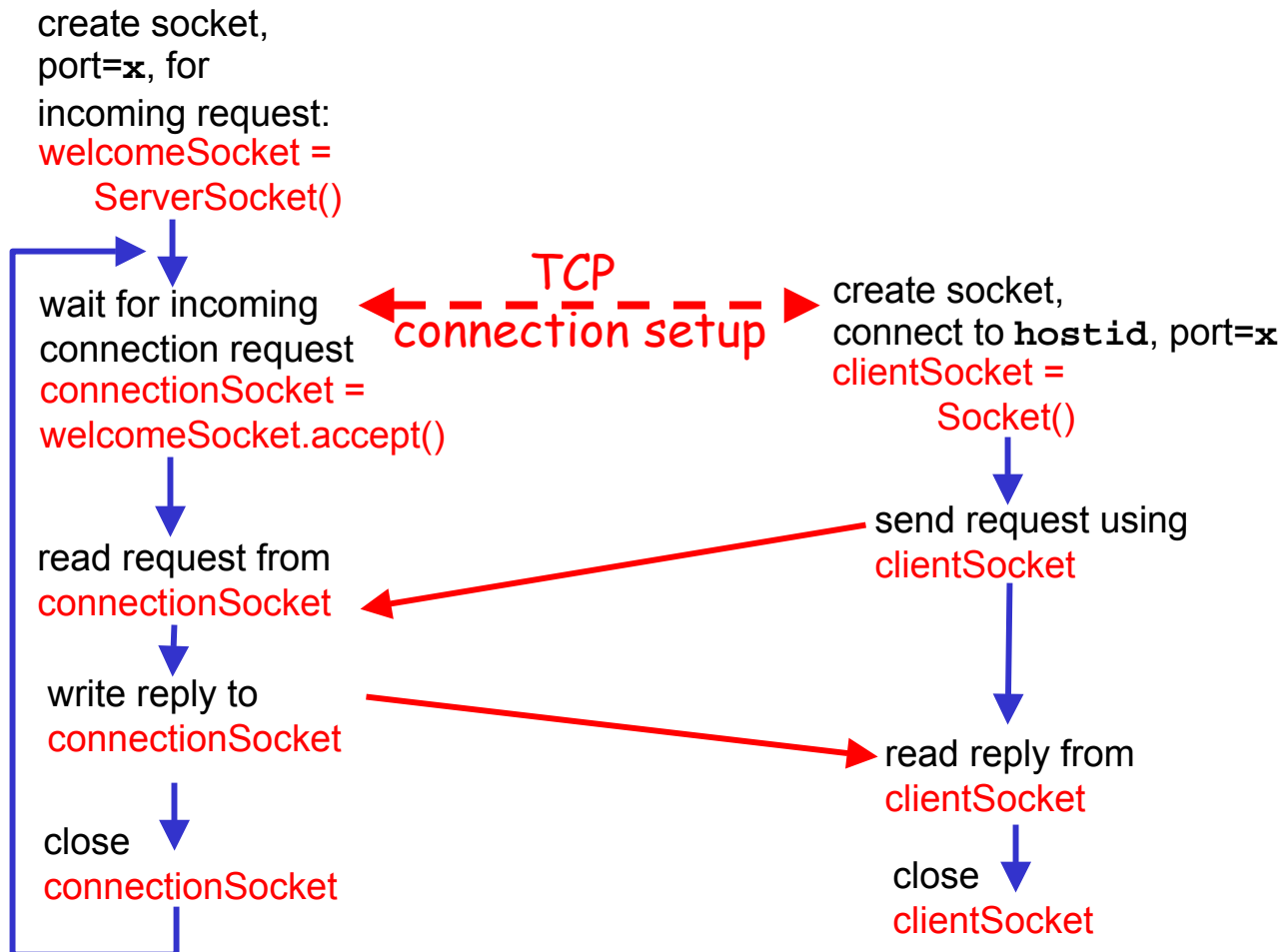
TCP service: reliable transfer of bytes from one process to another



Client/server socket interaction: TCP

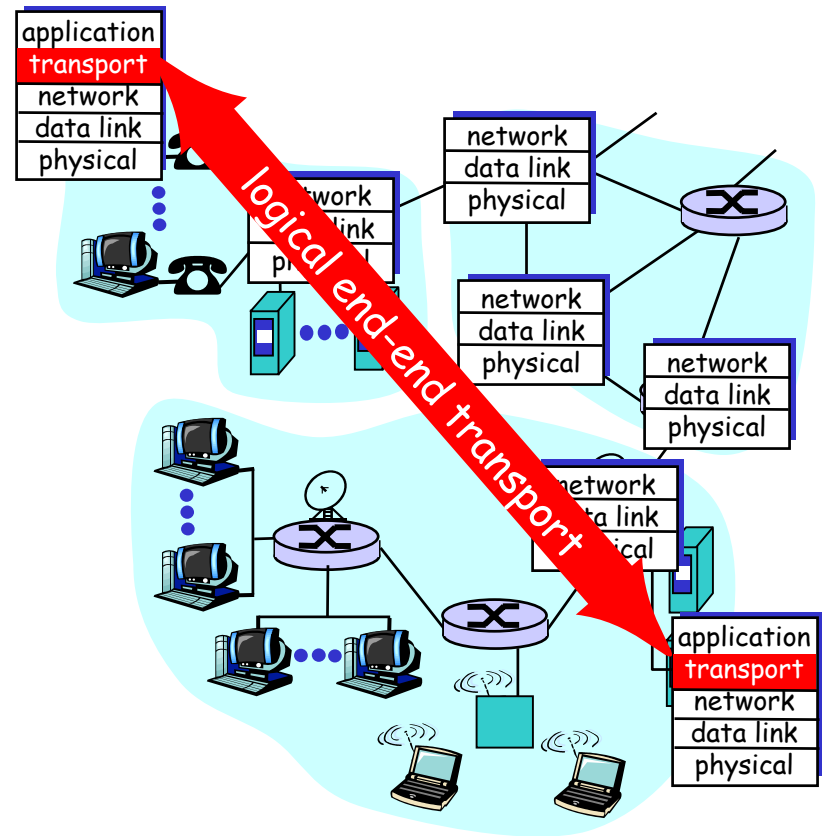
Server (running on `hostid`)

Client



II. Transport services and protocols

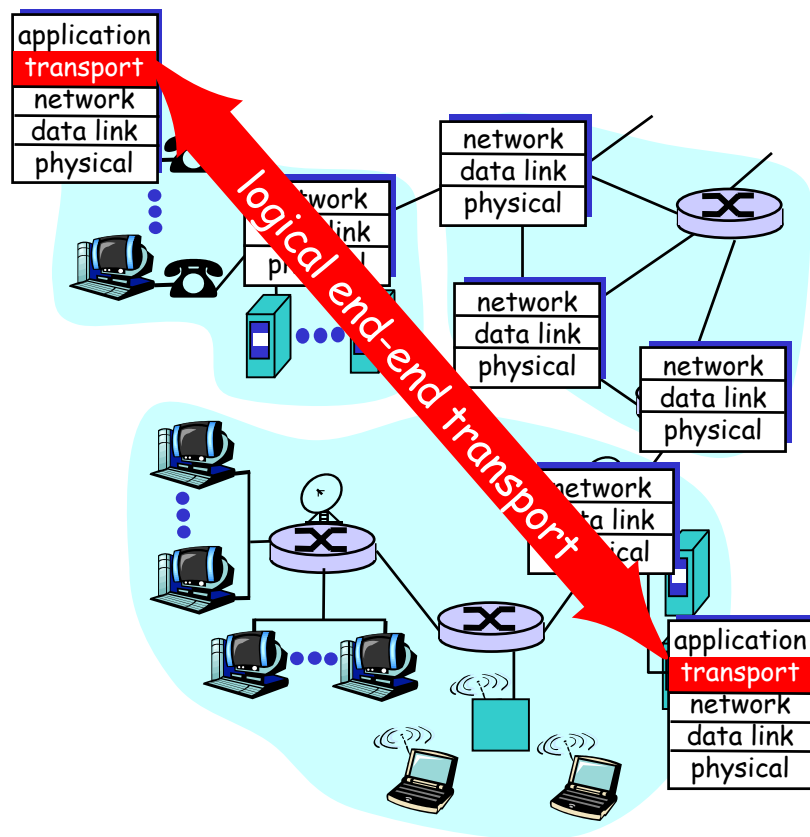
- ❑ provide *logical communication* between app' processes running on different hosts
- ❑ transport protocols run in end systems
- ❑ *transport vs network layer services:*
- ❑ *network layer:* data transfer between end systems
- ❑ *transport layer:* data transfer between processes
 - relies on, enhances, network layer services



Transport-layer protocols

Internet transport services:

- ❑ reliable, in-order unicast delivery (TCP)
 - congestion
 - flow control
 - connection setup
- ❑ unreliable ("best-effort"), unordered unicast or multicast delivery: UDP
- ❑ services not available:
 - real-time
 - bandwidth guarantees
 - reliable multicast



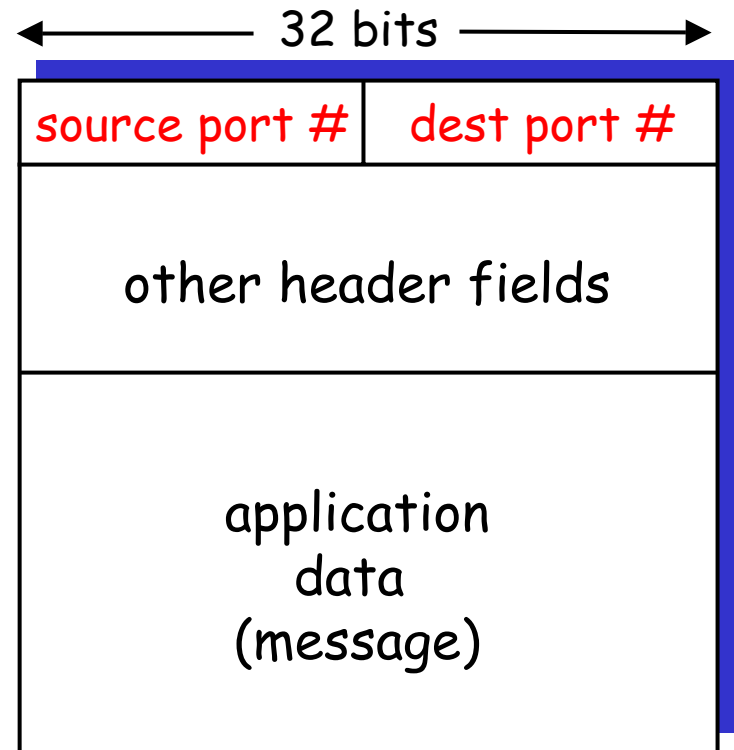
Multiplexing/demultiplexing

Multiplexing:

gathering data from multiple app processes, enveloping data with header (later used for demultiplexing)

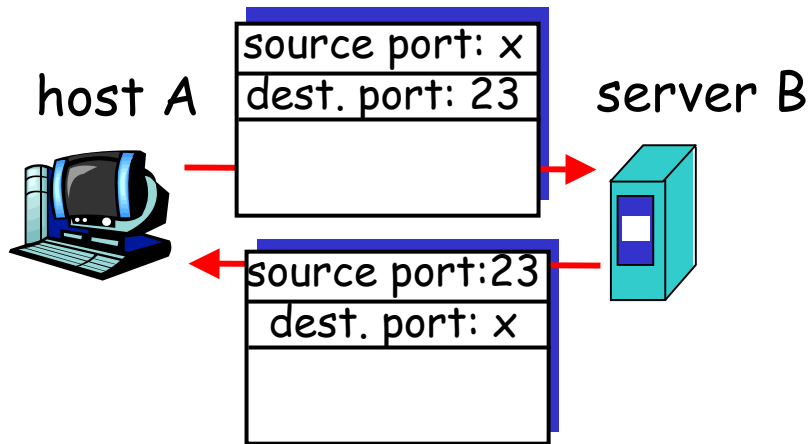
multiplexing/demultiplexing:

- based on sender, receiver port numbers, IP addresses
 - source, dest port #s in each segment
 - recall: well-known port numbers for specific applications



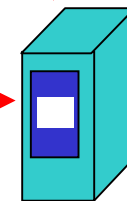
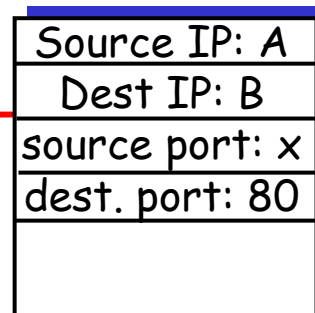
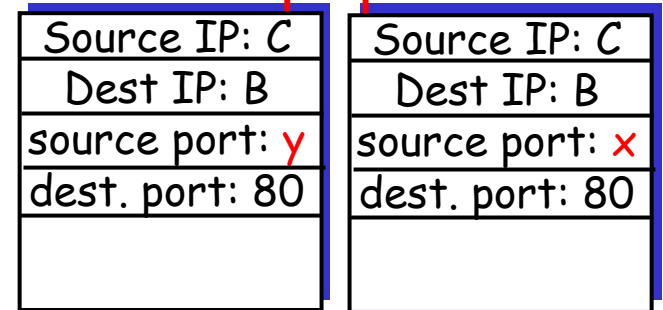
TCP/UDP segment format

Multiplexing/demultiplexing: examples



port use: simple telnet app

2 processes,
2 source
ports →
1 app.



Web
server B

port use: Web server

UDP: User Datagram Protocol [RFC 768]

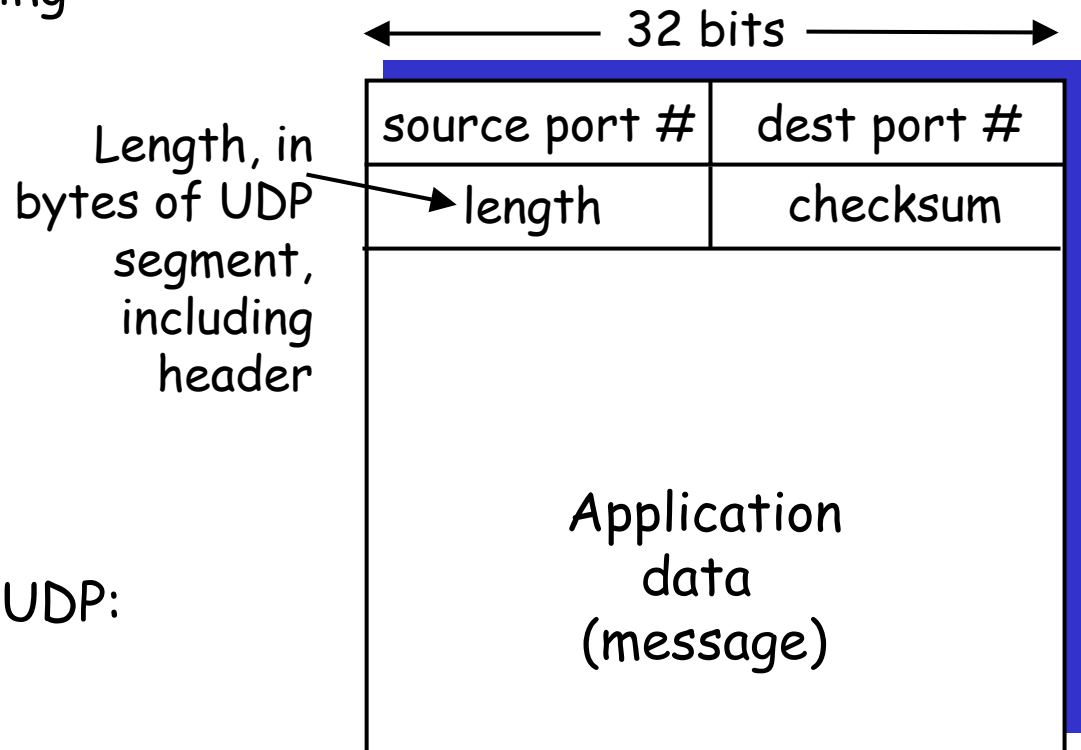
- ❑ “no frills,” “bare bones” Internet transport protocol
- ❑ “best effort” service, UDP segments may be:
 - lost
 - delivered out of order to app
- ❑ *connectionless*:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

Why is there a UDP?

- ❑ no connection establishment (which can add delay)
- ❑ simple: no connection state at sender, receiver
- ❑ small segment header (8 B vs. 20 B for TCP)
- ❑ no congestion control: UDP can blast away as fast as desired -- **but this is an issue for Internet** (p. 180)

UDP: more

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



UDP segment format

TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

□ point-to-point:

- one sender, one receiver
- no concept of multicasting in TCP
(see p. 349)

□ reliable, in-order *byte stream*:

- no "message boundaries"

□ pipelined:

- TCP congestion and flow control set window size

□ *send & receive buffers*

□ full duplex data:

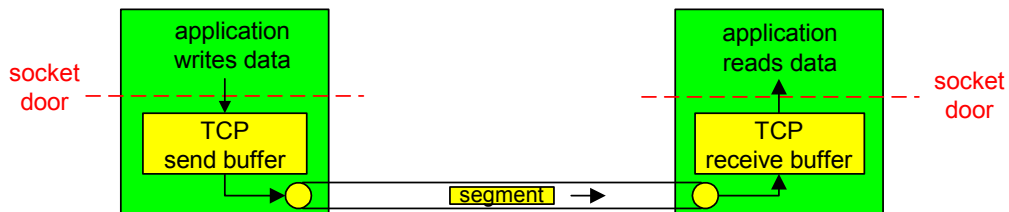
- bi-directional data flow in same connection
- MSS: maximum segment size for app layer data (configurable; e.g., 1500B; 512B)

□ connection-oriented:

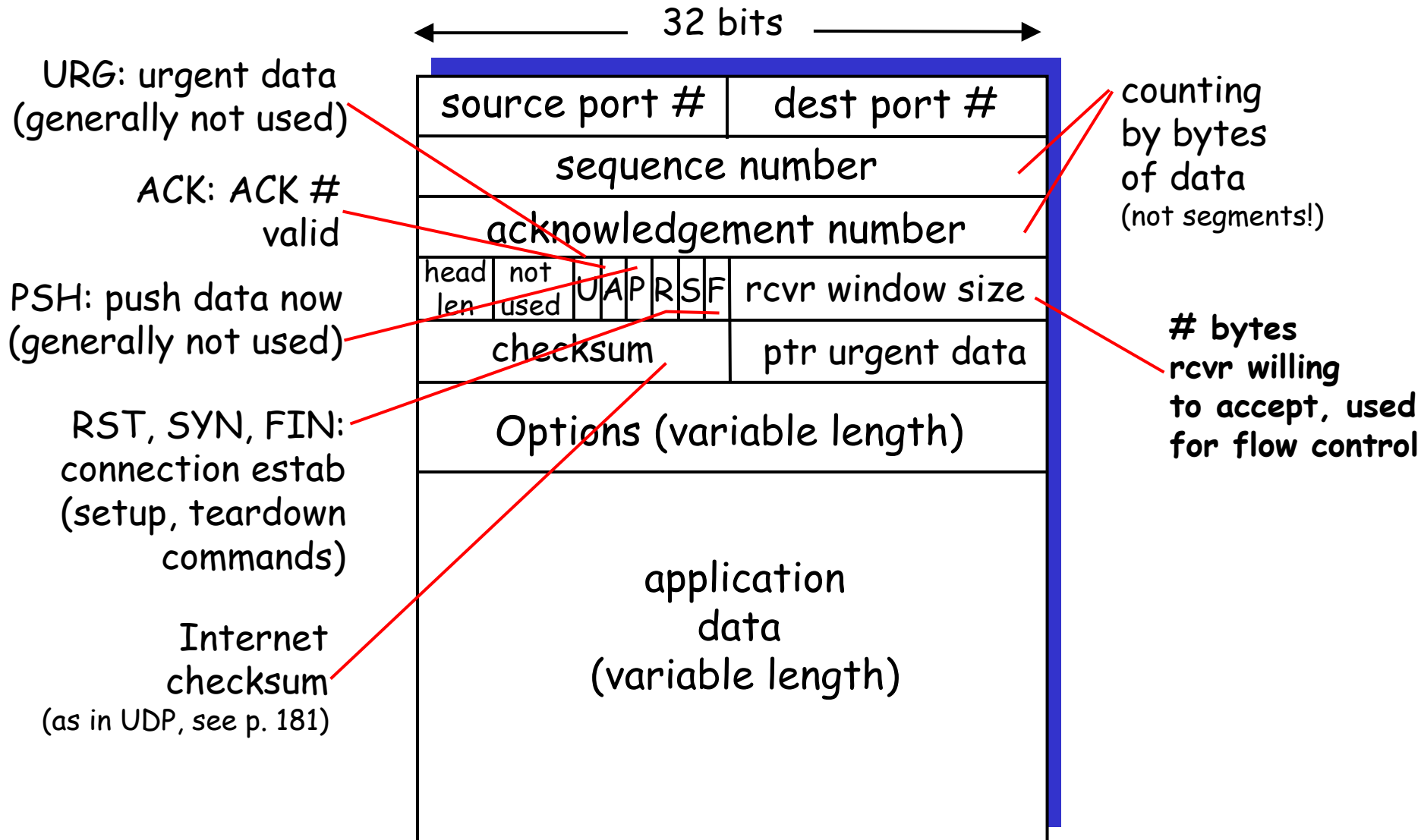
- handshaking (exchange of control msgs) init's sender, receiver state before data exchange

□ flow controlled:

- sender will not overwhelm receiver



TCP segment structure



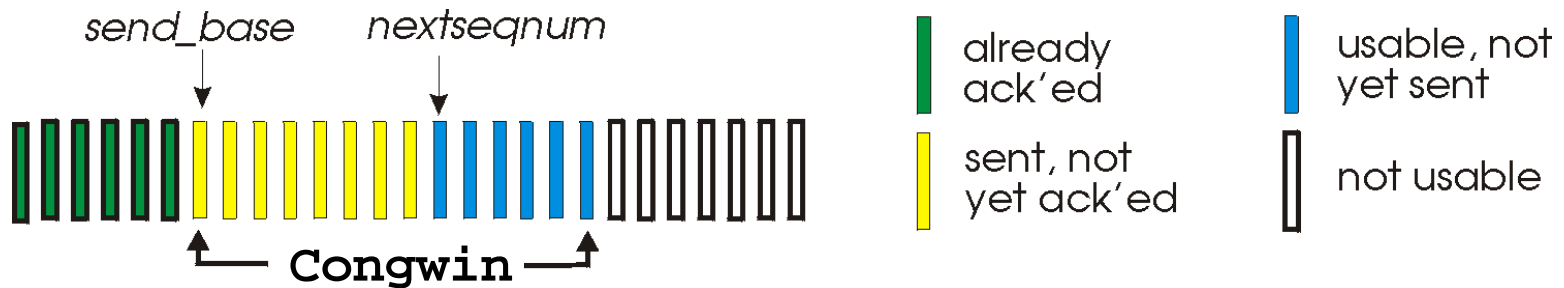
Principles of Congestion Control

Congestion:

- ❑ informally: "too many sources sending too much data too fast for *network* to handle"
- ❑ different from flow control!
- ❑ manifestations:
 - lost packets (buffer overflow at routers)
 - long delays (queueing in router buffers)
- ❑ typically results from overflowing of router buffers as network becomes congested and packets dropped/lost
- ❑ a top-10 problem!

TCP Congestion Control

- end-end control (no network assistance)
- transmission rate limited by congestion window size over segments:



TCP congestion control:

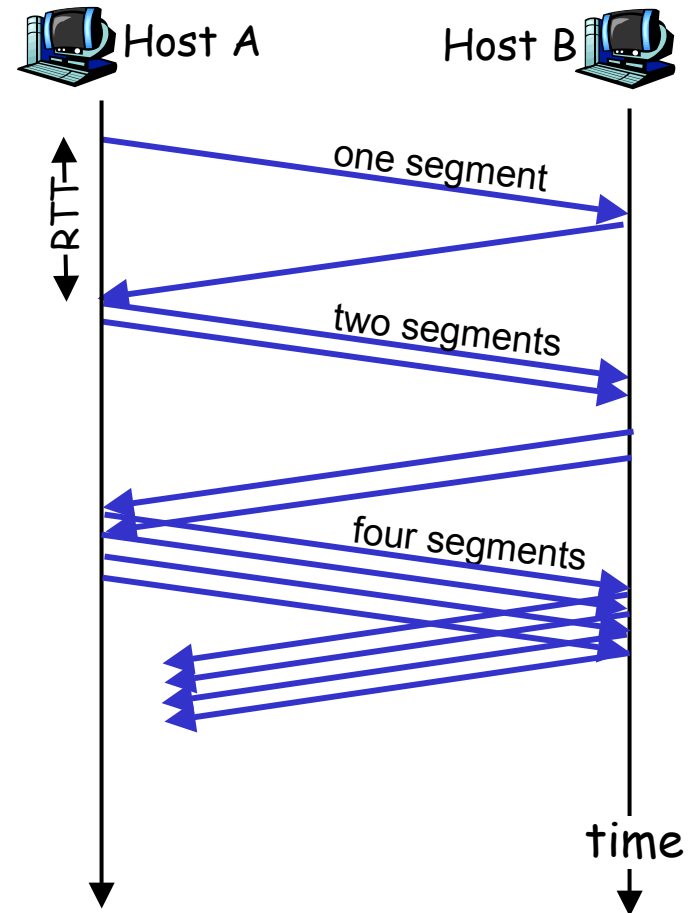
- “probing” for usable bandwidth:
 - ideally: transmit as fast as possible (Congwin as large as possible) without loss
 - increase Congwin until loss (congestion)
 - loss: decrease Congwin, then begin probing (increasing) again
- two “phases”
 - I. slow start
 - II. congestion avoidance
- important variables:
 - Congwin (in segments)
 - threshold: defines threshold between two slow start phase, congestion control phase

TCP Slowstart

Slowstart algorithm

initialize: Congwin = 1
for (each segment ACKed)
 Congwin++
until (loss event OR
 CongWin > threshold)

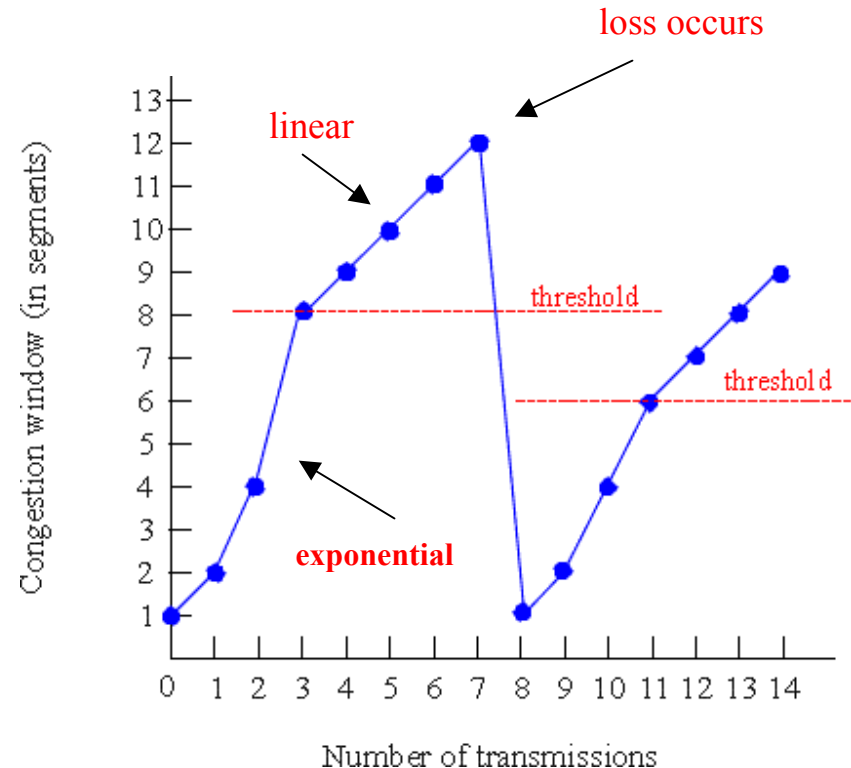
- exponential increase (per RTT) in window size (not so slow!)
- loss event: e.g., timeout



TCP Congestion Avoidance

Congestion avoidance

```
/* slowstart is over */
/* Congwin > threshold */
Until (loss event) {
  every w segments ACKed:
    Congwin++
}
threshold = Congwin/2
Congwin = 1
perform slowstart1
```



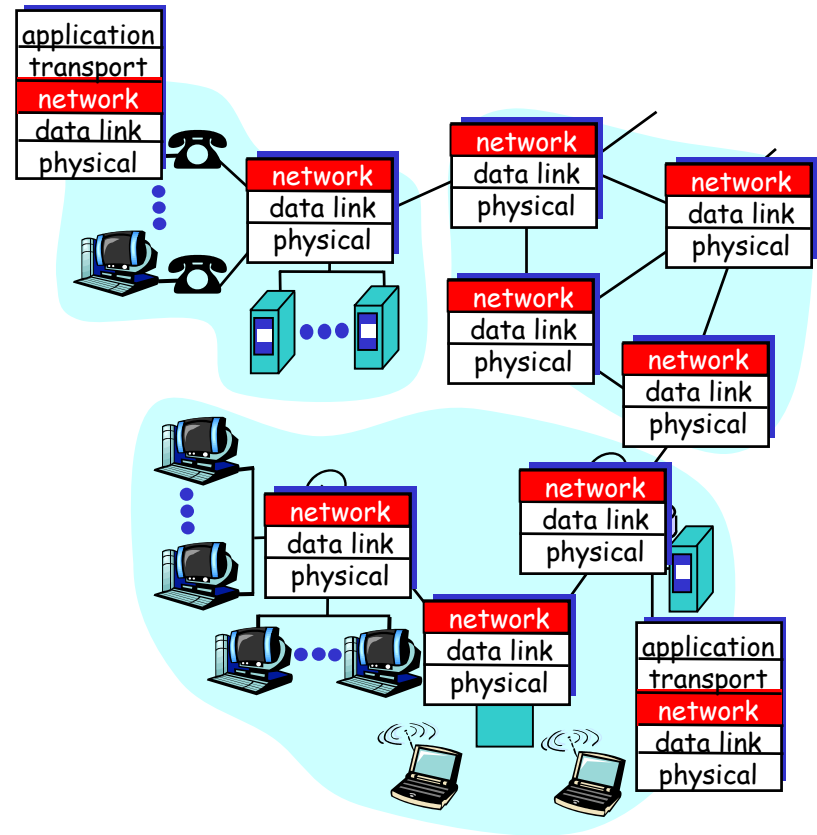
1. TCP Tahoe
- 2: TCP Reno skips slowstart (fast recovery) after three duplicate ACKs

III. Network layer functions

- ❑ transport packet from sending to receiving hosts
- ❑ network layer protocols in *every* host, router

three important functions:

- ❑ *path determination*: route taken by packets from source to dest. *Routing algorithms*
- ❑ *switching*: move packets from router's input to appropriate router output
- ❑ *call setup*: some network architectures require router call setup along path before data flows



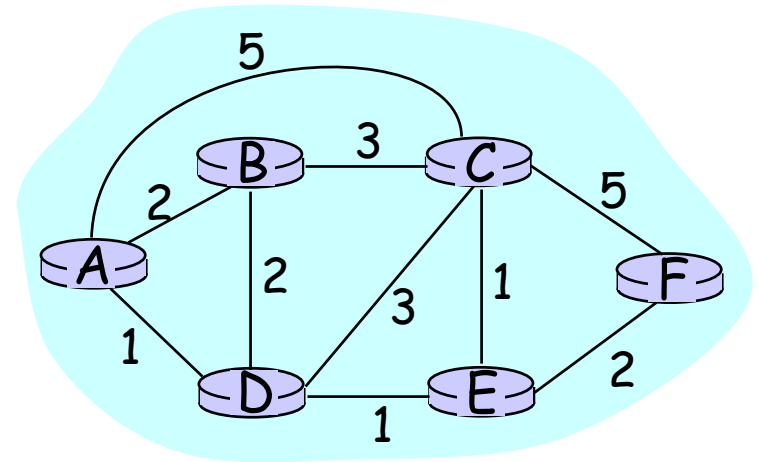
Routing

Routing protocol

Goal: determine "good" path (sequence of routers) thru network from source to dest.

Graph abstraction for routing algorithms:

- graph nodes are routers
- graph edges are physical links
 - link cost: delay, \$ cost, or congestion level



- "good" path:
 - typically means minimum cost path
 - other def's possible
 - least cost for A-C is ADEC

Finding least-cost path

Given the graph abstraction, the problem of finding the least-cost path from a source to a destination requires identifying a series of links such that:

- the first link in the path is connected to the source
- the last link in the path is connected to the destination
- for all i , the i and $i-1$ st link in the path are connected to the same node
- for the **least-cost path**, the sum of the cost of the links on the path is the minimum over all possible paths between the source and destination. Note: **shortest path** is, the path crossing the smallest number of links between the source and the destination

A Link-State Routing Algorithm for Computing Least-cost Path from One Node to All Other Nodes

Dijkstra's algorithm

- net topology, link costs known to all nodes
 - accomplished via "link state broadcast"
 - node knows about costs to directly attached neighbors and learns about topology from broadcasts from other nodes
 - all nodes have same info
- computes least cost paths from one node ('source') to all other nodes
 - gives routing table for that node
- iterative: after k iterations, know least cost path to k dest.'s

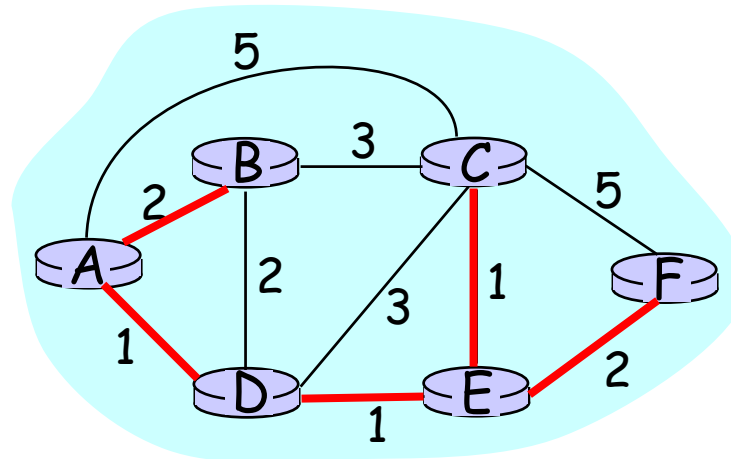
Notation:

- $c(i,j)$: link cost from node i to j . cost infinite if not direct neighbors
- $D(v)$: current value of cost of path from source to dest. V for given iteration
- $p(v)$: predecessor node along path from source to v , that is next v (neighbor)
- N : set of nodes whose least cost path definitively known

Dijkstra's algorithm: example

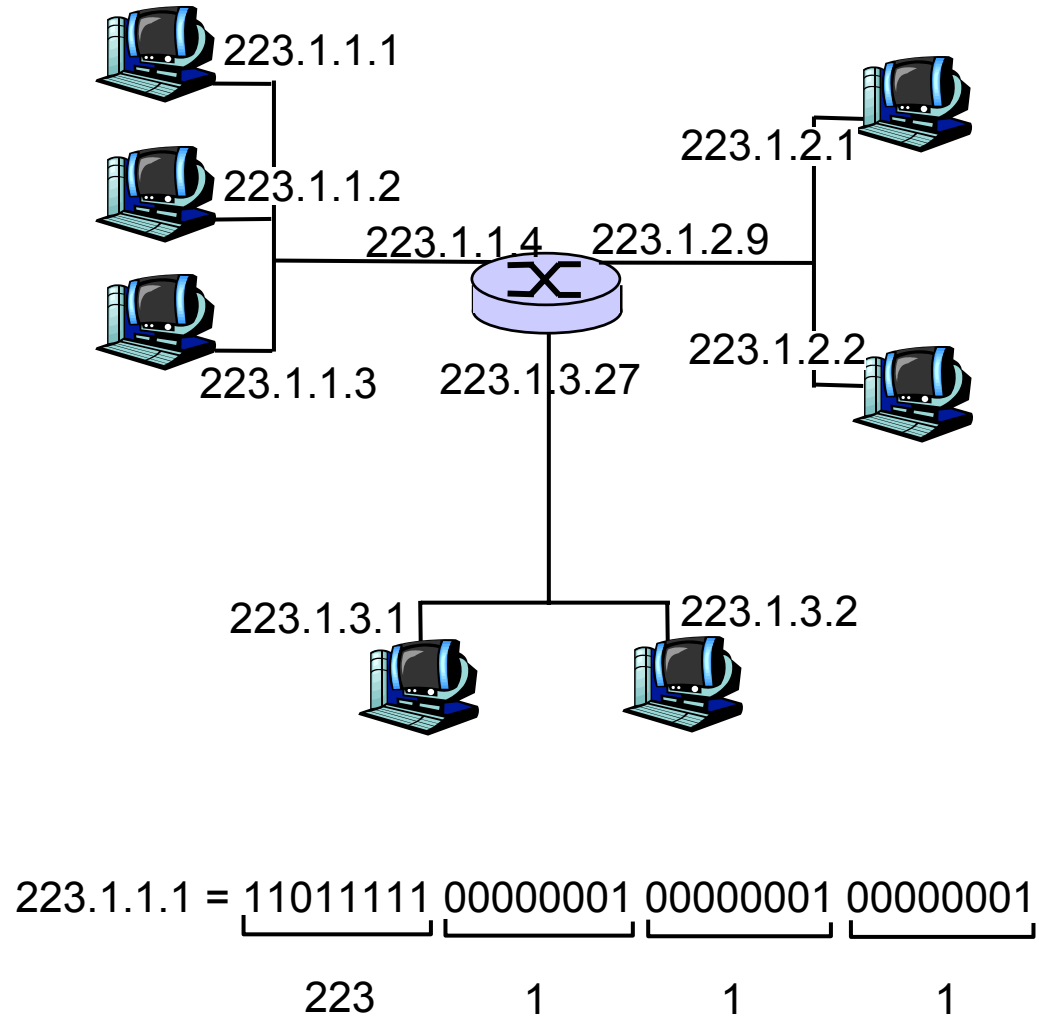
Each line in table gives values at end of the iteration

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
→ 0	A	2,A	5,A	1,A	infinity	infinity
→ 1	AD	2,A	4,D		2,D	infinity
→ 2	ADE	2,A	3,E			4,E
→ 3	ADEB		3,E			4,E
→ 4	ADEBC					4,E
5	ADEBCF					



IP Addressing: introduction

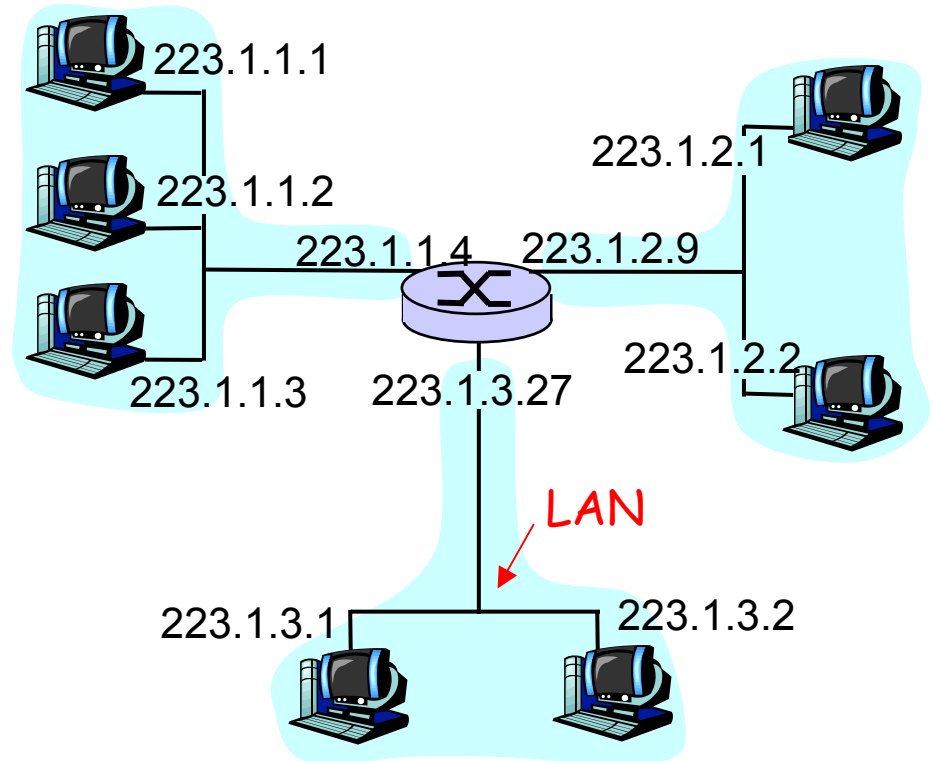
- ❑ **IP address:** 32-bit identifier for host, router *interface*
- ❑ **interface:** connection between host, router and physical link
 - router's typically have multiple interfaces
 - host may have multiple interfaces
 - IP addresses associated with interface, not host, router



dotted decimal notation

IP Addressing

- IP address:
 - network part (high order bits)
 - host part (low order bits)
- *What's a network ?*
(from IP address perspective)
 - device interfaces with same network part of IP address
 - can physically reach each other without intervening router



• network consisting of 3 IP networks (for IP addresses starting with 223, first 24 bits are **network address or prefix**)

• leftmost 24 bits defines net address in **"/24"** notation or **network mask**

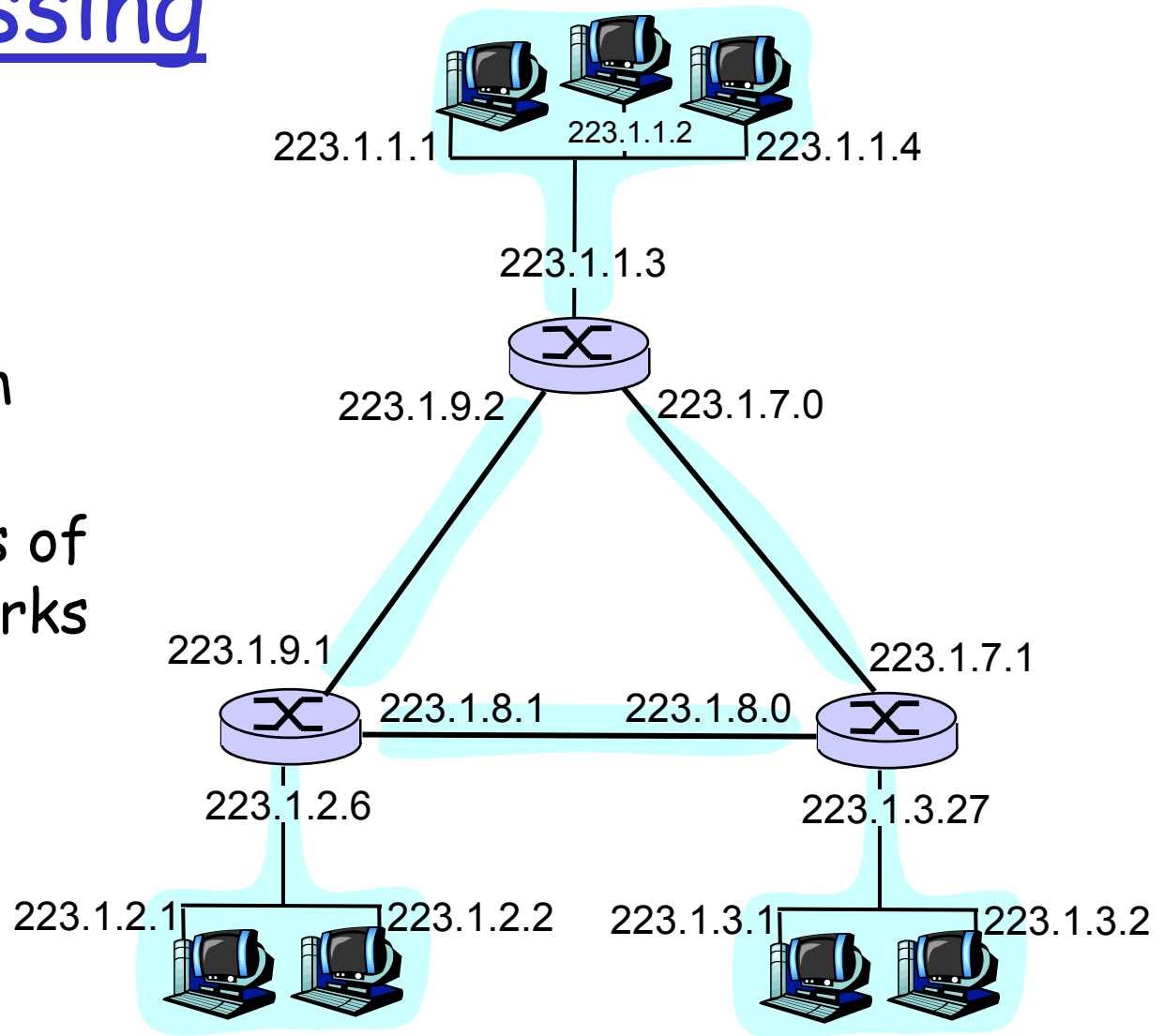
IP Addressing

How to find the networks?

- Detach each interface from router, host
- create "islands of isolated networks"

Interconnected system consisting of six networks

NB. 3 routers interconnected by point-to-point links



Getting a datagram from source to dest.

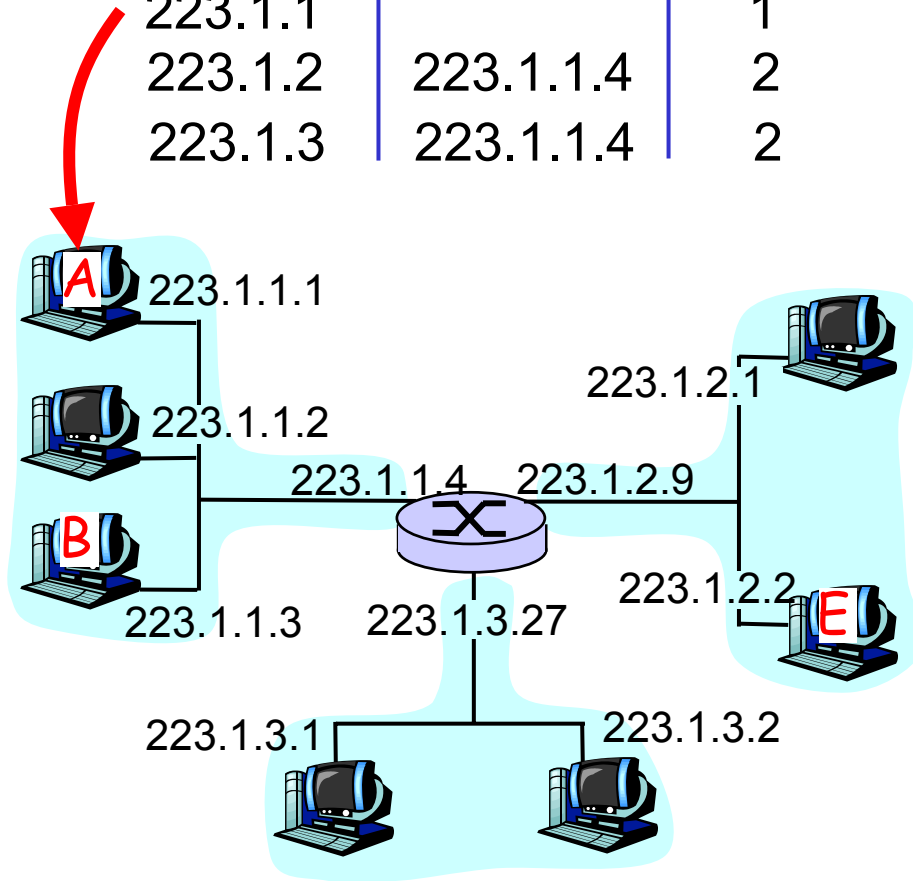
IP datagram:

misc fields	source IP addr	dest IP addr	data
-------------	----------------	--------------	------

- ❑ datagram remains unchanged, as it travels source to destination
- ❑ addr fields of interest here

routing table in A

Dest. Net.	next router	Nhops
223.1.1		1
223.1.2	223.1.1.4	2
223.1.3	223.1.1.4	2



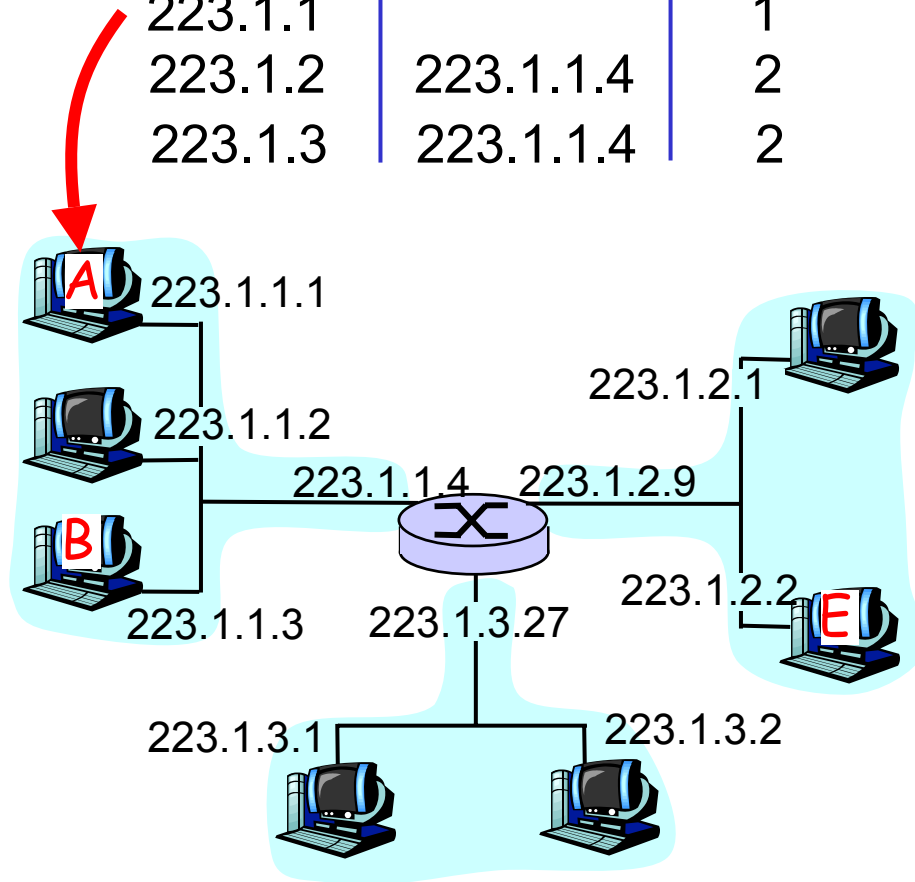
Getting a datagram from source to dest.

misc fields	223.1.1.1	223.1.1.3	data
-------------	-----------	-----------	------

Starting at A, given IP datagram addressed to B:

- ❑ look up net. address of B
- ❑ find B is on same net. as A
- ❑ link layer will send datagram directly to B inside link-layer frame
 - B and A are directly connected

Dest. Net.	next router	Nhops
223.1.1		1
223.1.2	223.1.1.4	2
223.1.3	223.1.1.4	2



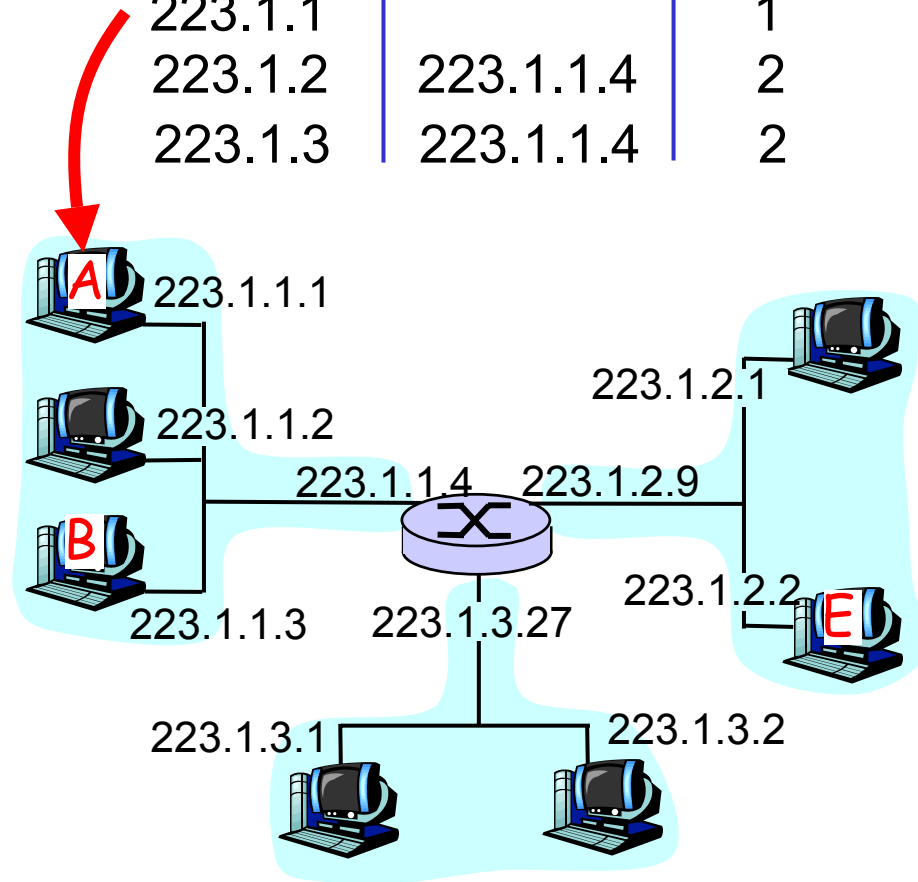
Getting a datagram from source to dest.

misc fields	223.1.1.1	223.1.2.2	data
-------------	-----------	-----------	------

Starting at A, dest. E:

- ❑ look up network address of E
- ❑ E on *different* network
 - A, E not directly attached
- ❑ routing table: next hop router to E is 223.1.1.4
- ❑ link layer sends datagram to router 223.1.1.4 inside link-layer frame
- ❑ datagram arrives at 223.1.1.4
- ❑ continued.....

Dest. Net.	next router	Nhops
223.1.1		1
223.1.2	223.1.1.4	2
223.1.3	223.1.1.4	2



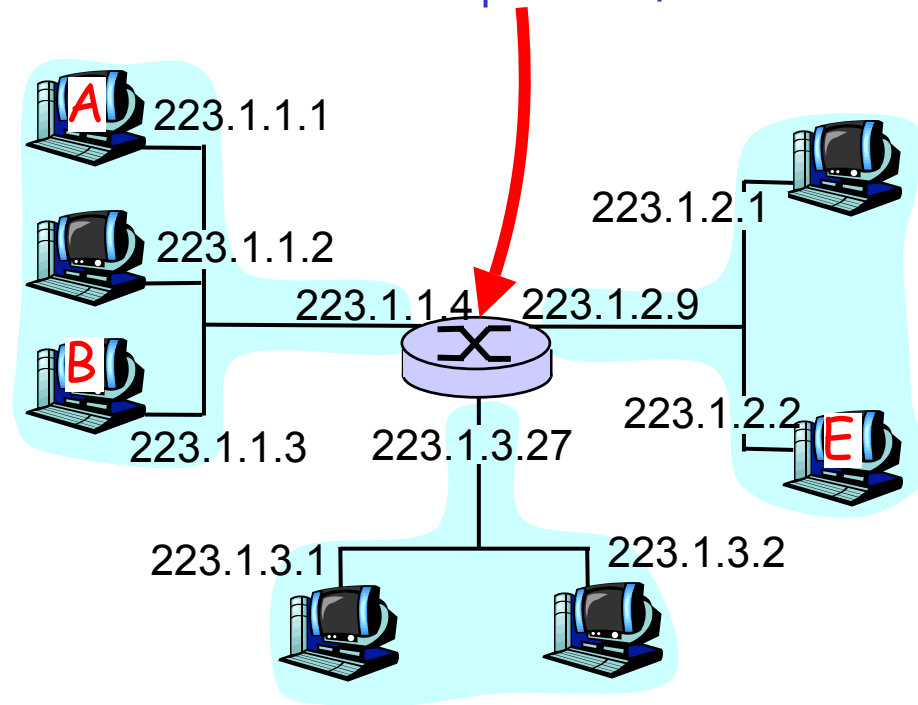
Getting a datagram from source to dest.

misc fields	223.1.1.1	223.1.2.2	data
-------------	-----------	-----------	------

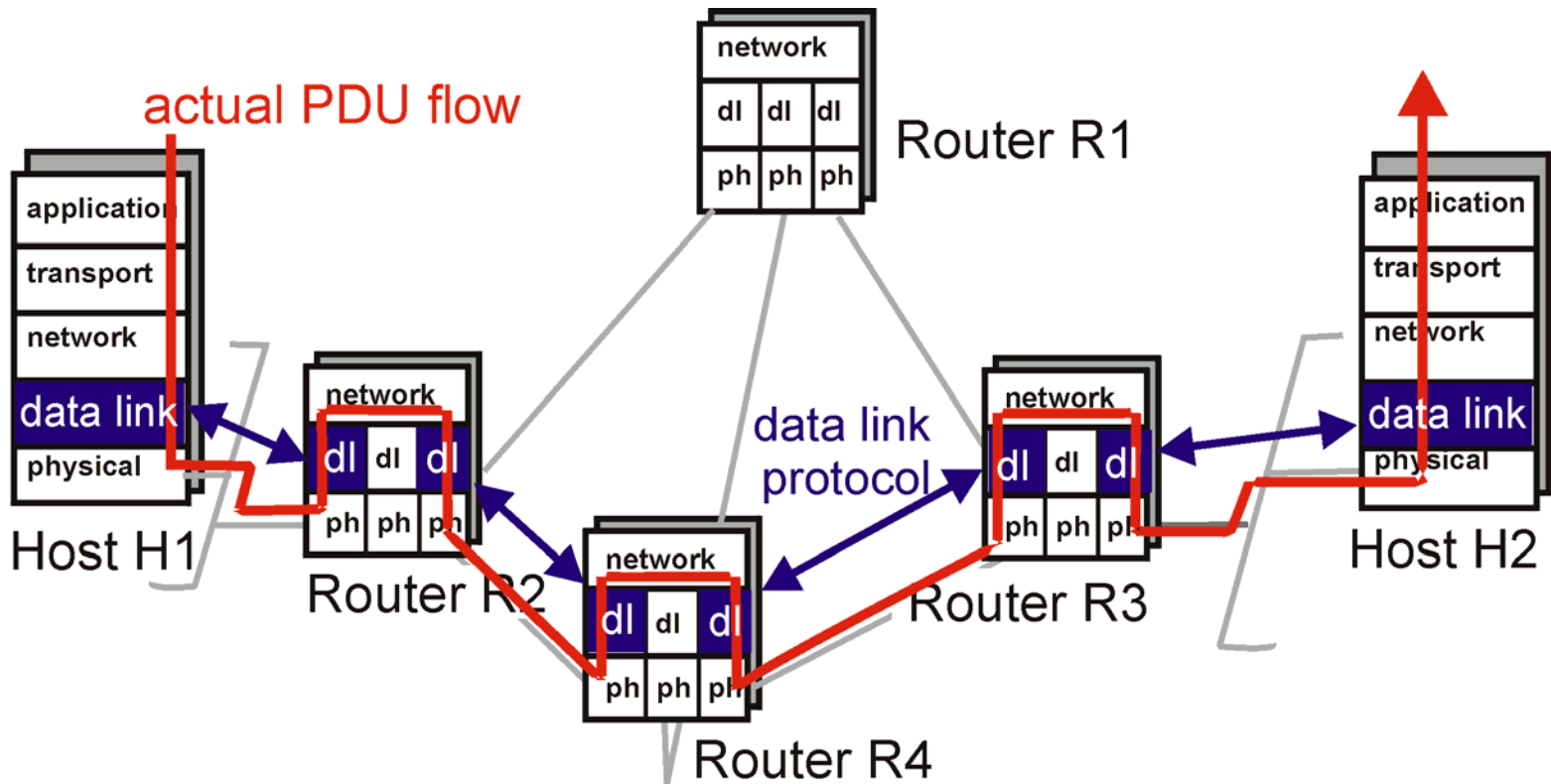
Arriving at 223.1.4,
destined for 223.1.2.2

- ❑ look up network address of E
- ❑ E on *same* network as router's interface 223.1.2.9
 - router, E directly attached
- ❑ link layer sends datagram to 223.1.2.2 inside link-layer frame via interface 223.1.2.9
- ❑ datagram arrives at 223.1.2.2

Dest. network	next router	Nhops	interface
223.1.1	-	1	223.1.1.4
223.1.2	-	1	223.1.2.9
223.1.3	-	1	223.1.3.27



IV. Link Layer Protocols



Link Layer Services

□ *Framing and link access:*

- encapsulate datagram into frame adding header and trailer,
- implement channel access if shared medium,
- 'physical addresses' are used in frame headers to identify source and destination of frames on broadcast links (vis-à-vis IP address)

□ *Reliable Delivery:*

- seldom used on fiber optic, co-axial cable and some twisted pairs too due to low bit error rate.
- Used on wireless links, where the goal is to reduce errors thus avoiding end-to-end retransmissions

Link Layer Services (more)

□ *Flow Control:*

- pacing between senders and receivers

□ *Error Detection:*

- errors are caused by signal attenuation and noise.
- Receiver detects presence of errors:
- it signals the sender for retransmission or just drops the corrupted frame

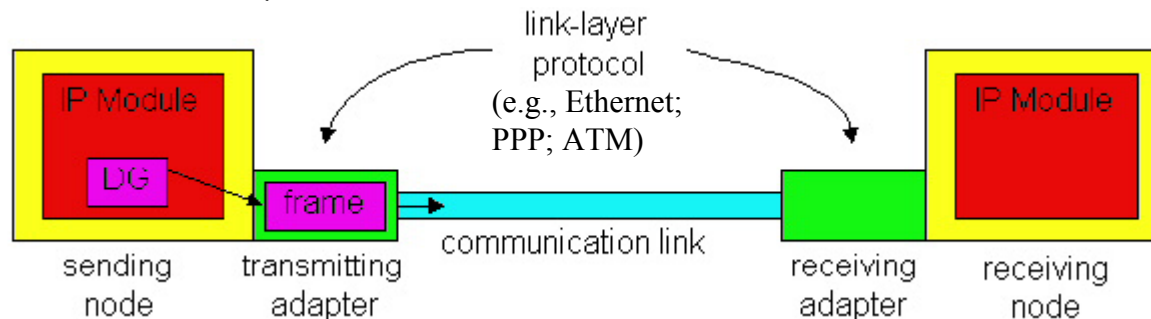
□ *Error Correction:*

- mechanism for the receiver to locate and correct the error without resorting to retransmission

Link Layer Protocol Implementation

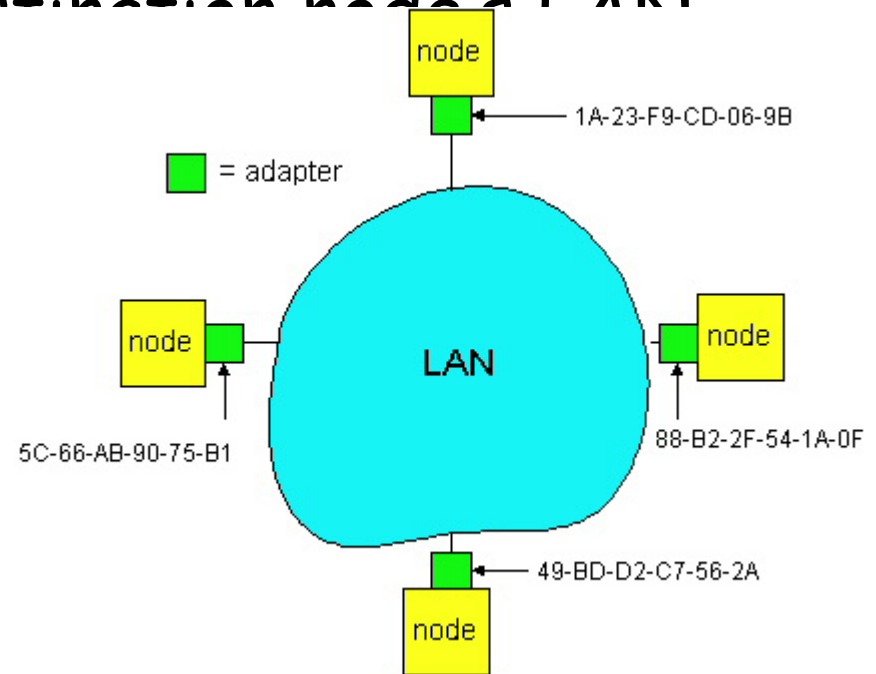
- Link layer protocol entirely implemented in the **adapter** (eg, PCMCIA card). Adapter typically includes: RAM, DSP chips, host bus interface, and link interface
- Adapter **send** operations: encapsulates (set sequence numbers, feedback info, etc.), adds error detection bits, implements channel access for shared medium, transmits on link

□ Adc
cor
pro
fee



LAN Addresses and ARP

- ❑ **IP address:** drives the packet to destination **network**
- ❑ **LAN (or MAC or Physical) address:** drives the packet to the destination **node's LAN interface card (adapter)**
- ❑ **48 bit, 6 byte MAC (for most LANs);** burned in the adapter **ROM**



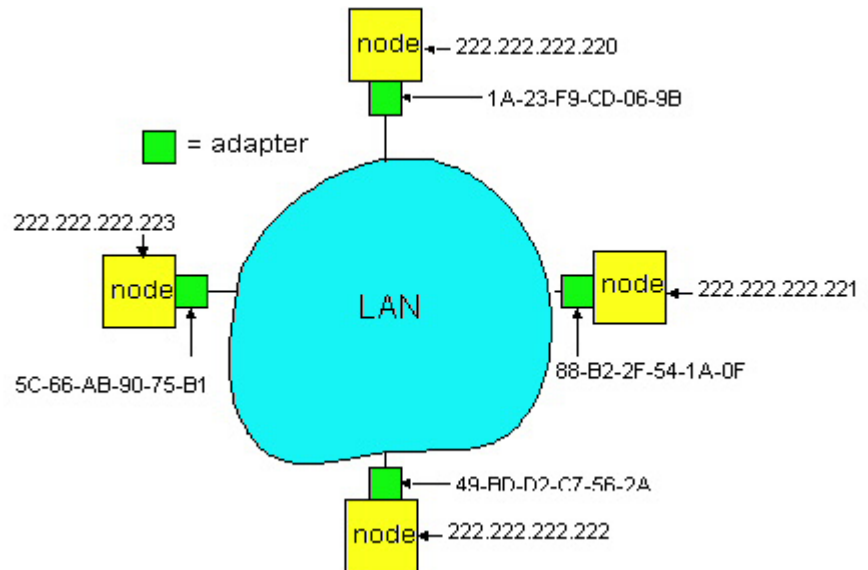
LAN Address (more)

- ❑ MAC address allocation administered by IEEE
- ❑ A manufacturer buys a portion of the address space (to assure uniqueness)
- ❑ Analogy:
 - (a) MAC address: like Social Security Number
 - (b) IP address: like postal address

- ❑ MAC flat address => portability
- ❑ IP hierarchical address NOT portable (need mobile IP)
- ❑ Broadcast LAN address: 1111.....1111

ARP: Address Resolution Protocol

- ❑ Each IP node (Host, Router) on the LAN has **ARP** module and Table
- ❑ ARP Table: IP/MAC address mappings for **some** LAN nodes
< IP address; MAC address; TTL >
< >
- ❑ TTL (Time To Live):
timer, typically
20 min



ARP (more)

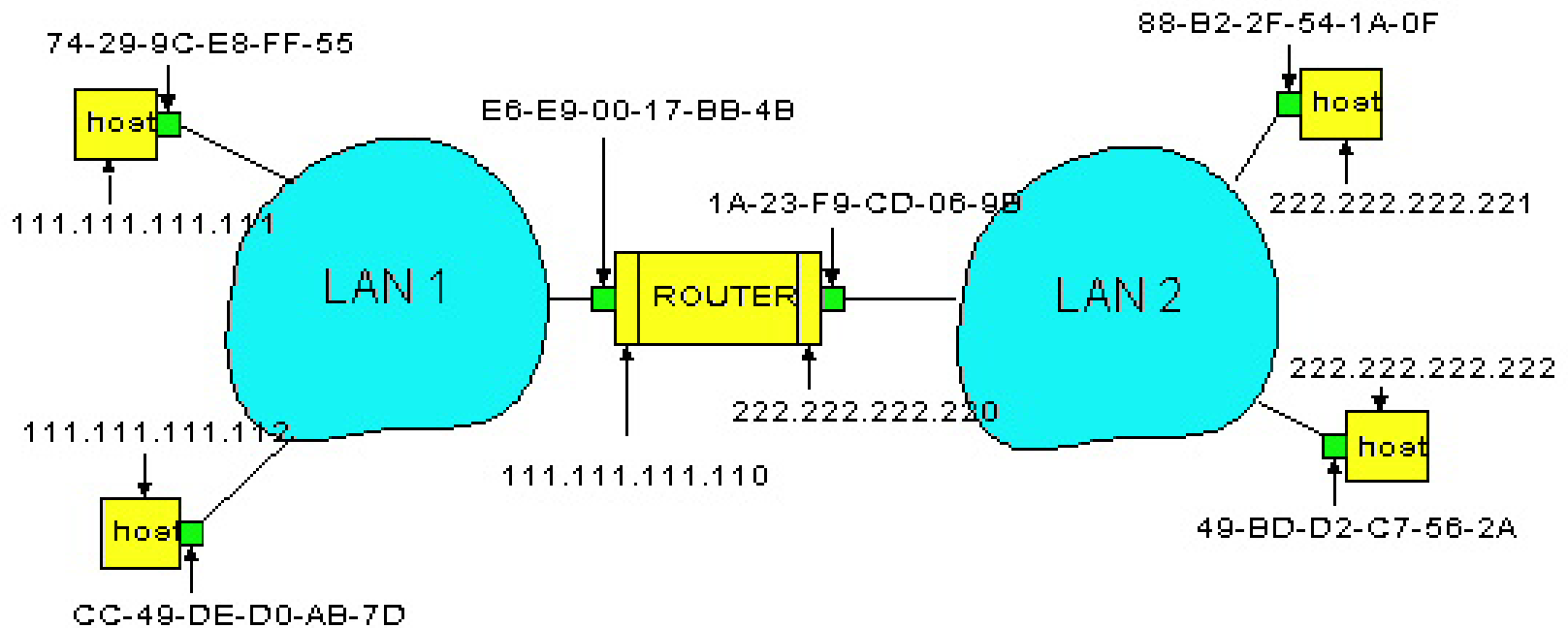
- ❑ Host A wants to send packet to destination IP addr XYZ on same LAN
- ❑ Source Host first checks own ARP Table for IP addr XYZ
- ❑ If XYZ **not** in the ARP Table, ARP module **broadcasts** ARP pkt:

< XYZ, MAC (?) >

- ❑ ALL nodes on the LAN accept and inspect the ARP pkt
- ❑ Node XYZ responds with **unicast** ARP pkt carrying own MAC addr:

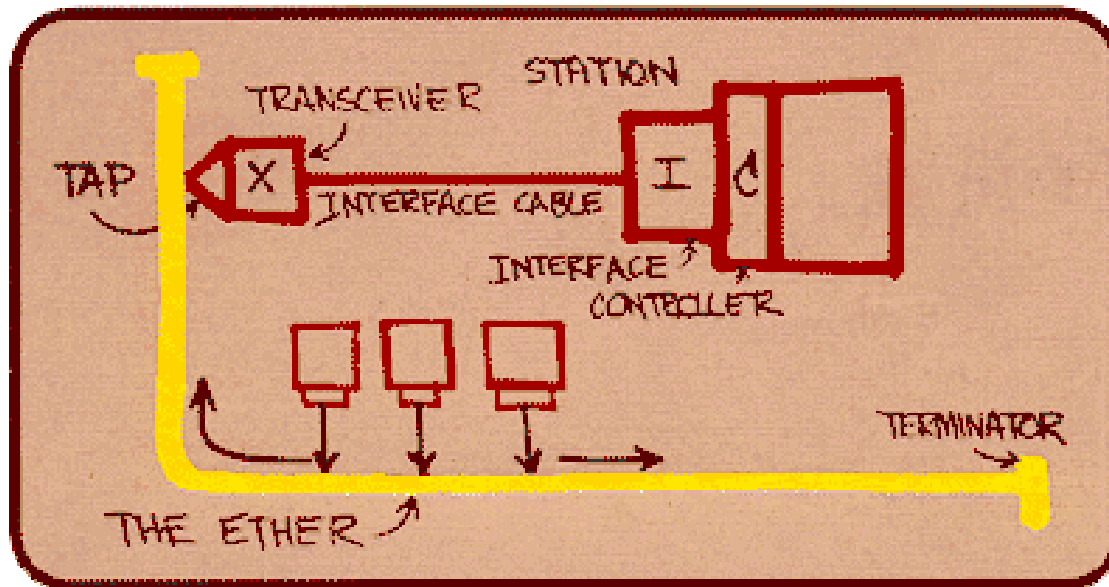
Routing pkt to another LAN

- Say, route packet from source IP addr <111.111.111.111> to destination addr <222.222.222.222>



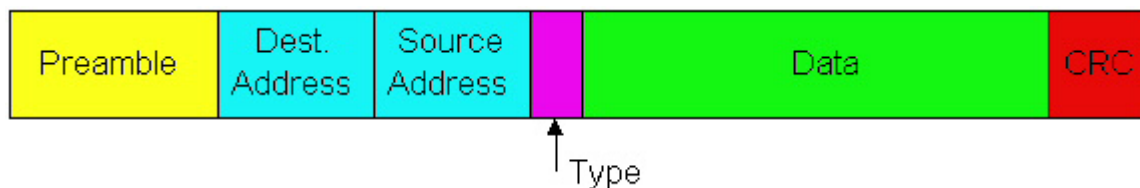
Ethernet

- Widely deployed because:
 - Cheap as dirt! \$20 for 100Mbps!
 - First LAN technology
 - Simpler and less expensive than token LANs and ATM
 - Kept up with the speed race: 10, 100, 1000 Mbps
 - Many E-net technologies (cable, fiber etc). But they all share common characteristics



Ethernet Frame Structure

- ❑ Sending adapter encapsulates an IP datagram (or other network layer protocol packet) in **Ethernet Frame** which contains a Preamble, a Header, Data, and CRC fields
- ❑ **Preamble**: 7 bytes with the pattern 10101010 followed by one byte with the pattern 10101011; used for synchronizing receiver to sender clock (clocks are never exact, some drift is highly likely)



Ethernet's Multiple Access Protocol CSMA/CD

1. An adapter may begin to transmit at any time, that is, no slots are used.
2. An adapter never transmits a frame when it senses that some other adapter is transmitting, that is, it uses carrier-sensing.
3. A transmitting adapter aborts its transmission as soon as it detects that another adapter is also transmitting, that is, it uses collision detection.
4. Before attempting a retransmission, an adapter waits a random time that is typically small compared to a frame time.

CSMA/CD

A: sense channel, if idle

then {

transmit and monitor the channel;

If detect another transmission

then {

abort and send jam signal;

update # collisions;

delay as required by exponential backoff algorithm;

goto A

}

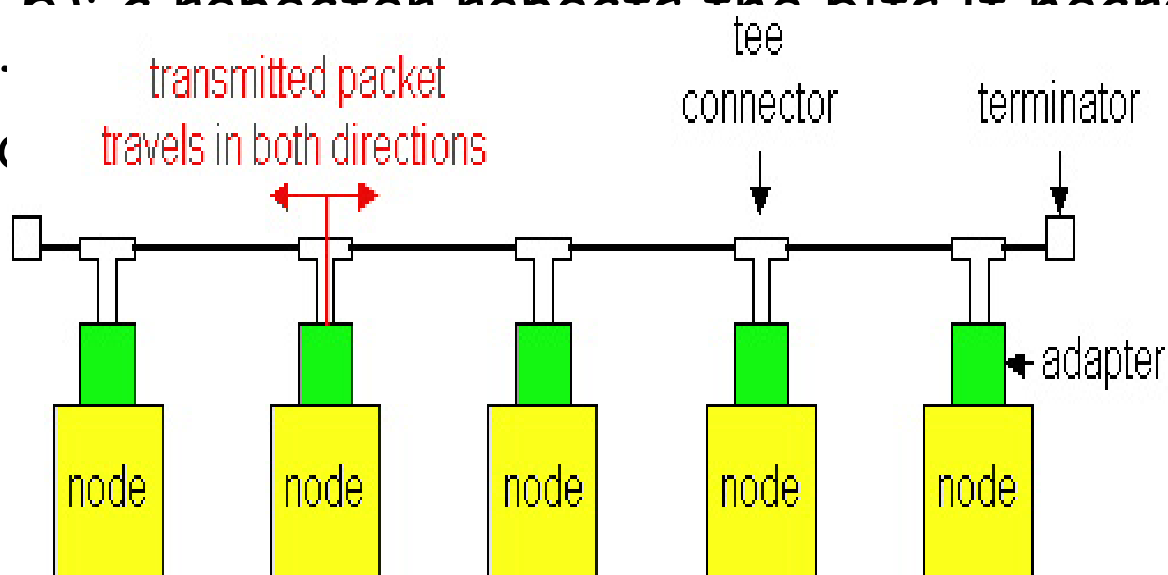
else {done with the frame; set collisions to zero}

}

else {wait until ongoing transmission is over and **goto A}**

Ethernet Technologies: 10Base2

- ❑ 10==10Mbps; 2==under 200 meters maximum length of a cable segment; also referred to as "Cheapnet"
 - ❑ Uses thin coaxial cable in a bus topology
 - ❑ Repeaters are used to connect multiple segments (up to 5): a repeater repeats the bits it hears on one in-
- layer (



Gbit Ethernet

- ❑ Use standard Ethernet frame format
- ❑ Allows for Point-to-point links and shared broadcast channels
- ❑ In shared mode, CSMA/CD is used; short distances between nodes to be efficient
- ❑ Uses Hubs called here "Buffered Distributors"
- ❑ Full-Duplex at 1 Gbps for point-to-point links

Summary: Internet protocol stack

- ❑ **application:** supporting network applications
 - ftp, smtp, http
- ❑ **transport:** host-host data transfer, congestion control, segmentation
 - 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"

