Part I: Introduction

Goal:

- get context, overview, "feel" of networking
- more depth, detail *later* in course
- approach:
  - descriptive
  - use Internet as example

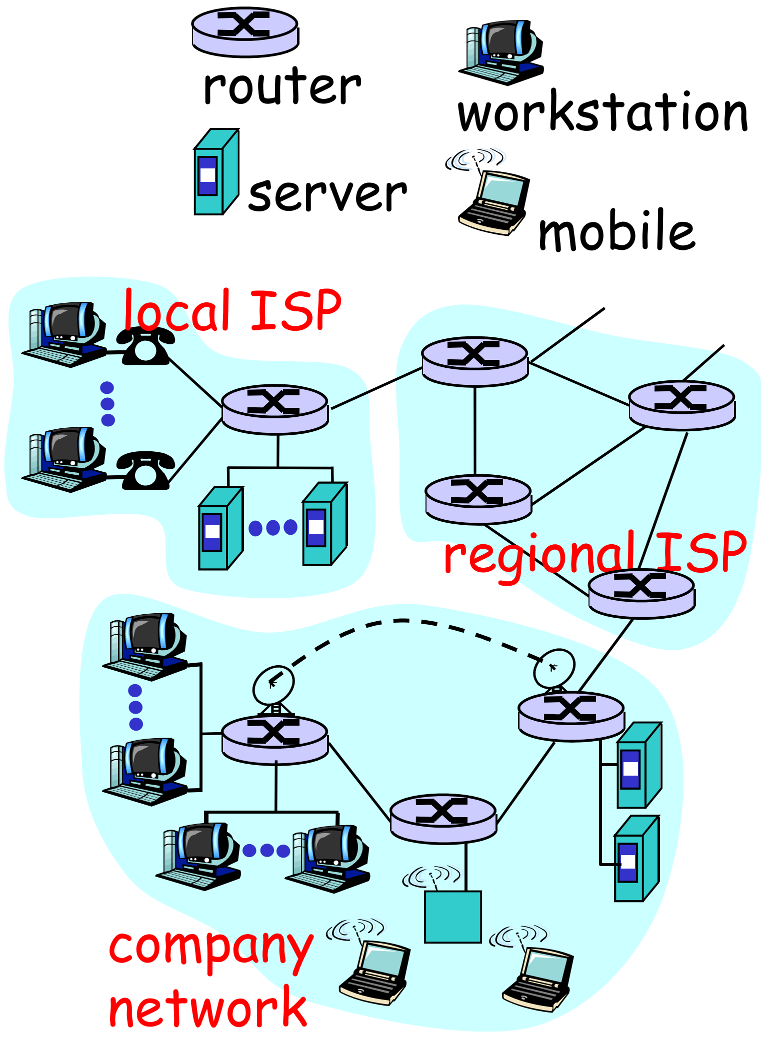
Overview:

- what's the Internet
- what's a protocol?
- network edge
- network core
- access net, physical media
- performance: loss, delay
- protocol layers, service models
- backbones, NAPs, ISPs
- history
- ATM network

What's the Internet: "nuts and bolts" view

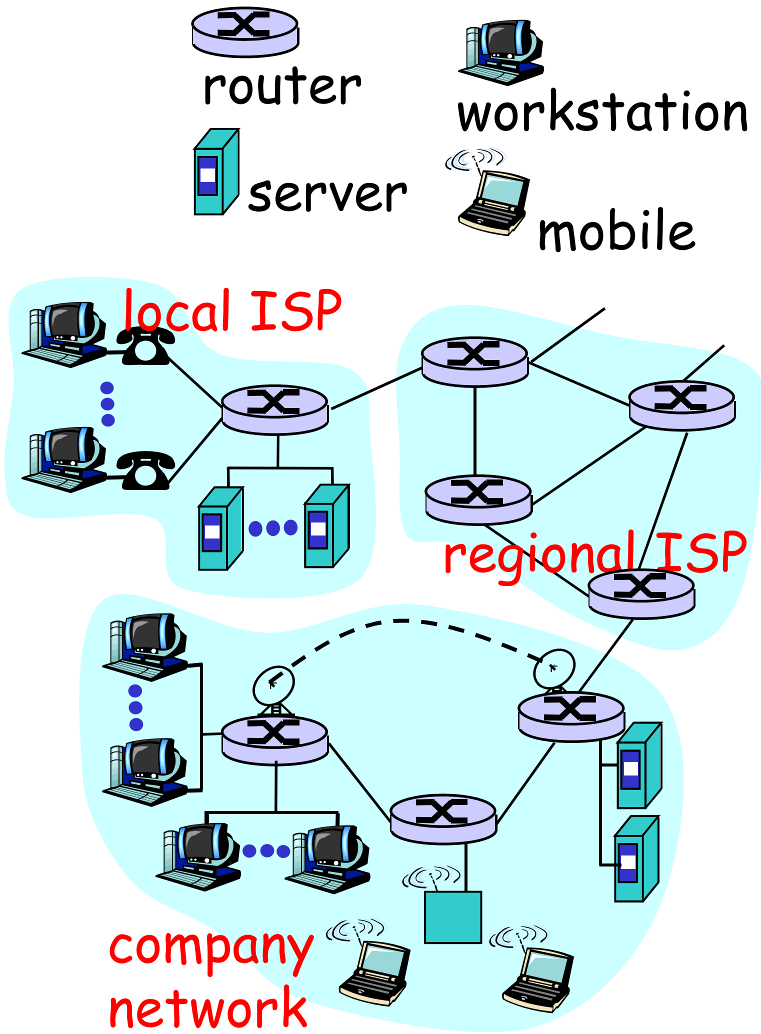
- millions of connected computing devices: *hosts, end-systems*
  - pc's, workstations, servers
  - PDA's, phones, toasters

  running *network apps*
- *communication links*
  - fiber, copper, radio, satellite
- *routers:* forward packets (chunks) of data thru network



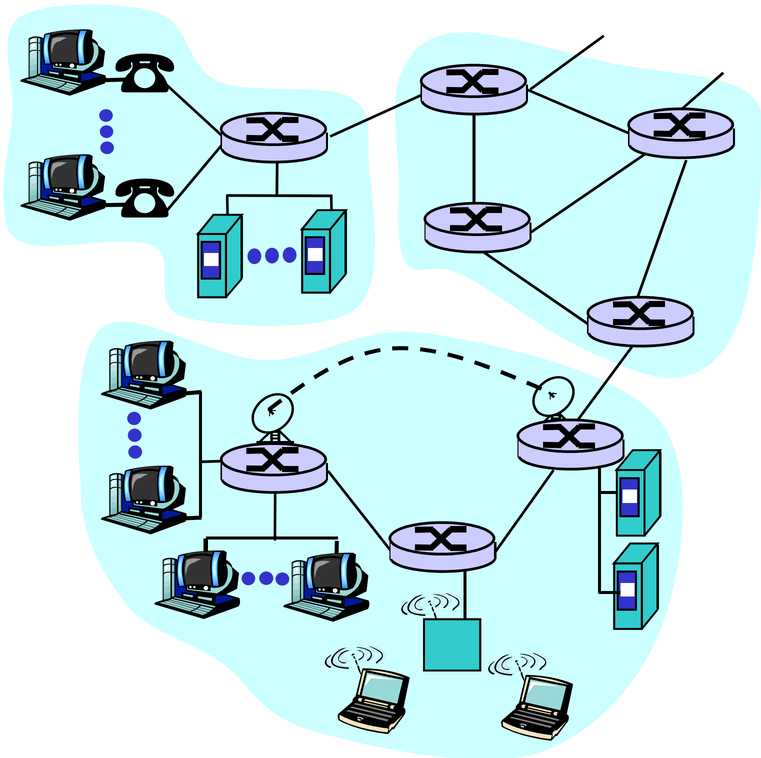
What's the Internet: "nuts and bolts" view

- *protocols*: control sending, receiving of msgs
  - e.g., TCP, IP, HTTP, FTP, PPP
- *Internet: "network of networks"*
  - loosely hierarchical
  - public Internet versus private intranet
- Internet standards
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force



What's the Internet: a service view

- communication *infrastructure* enables distributed applications:
  - WWW, email, games, e-commerce, databases, voting,
  - more?
- communication services provided:
  - connectionless
  - connection-oriented
- cyberspace [Gibson]



What's a protocol?

human protocols:

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

… specific msgs sent

… specific actions taken when msgs received, or other events

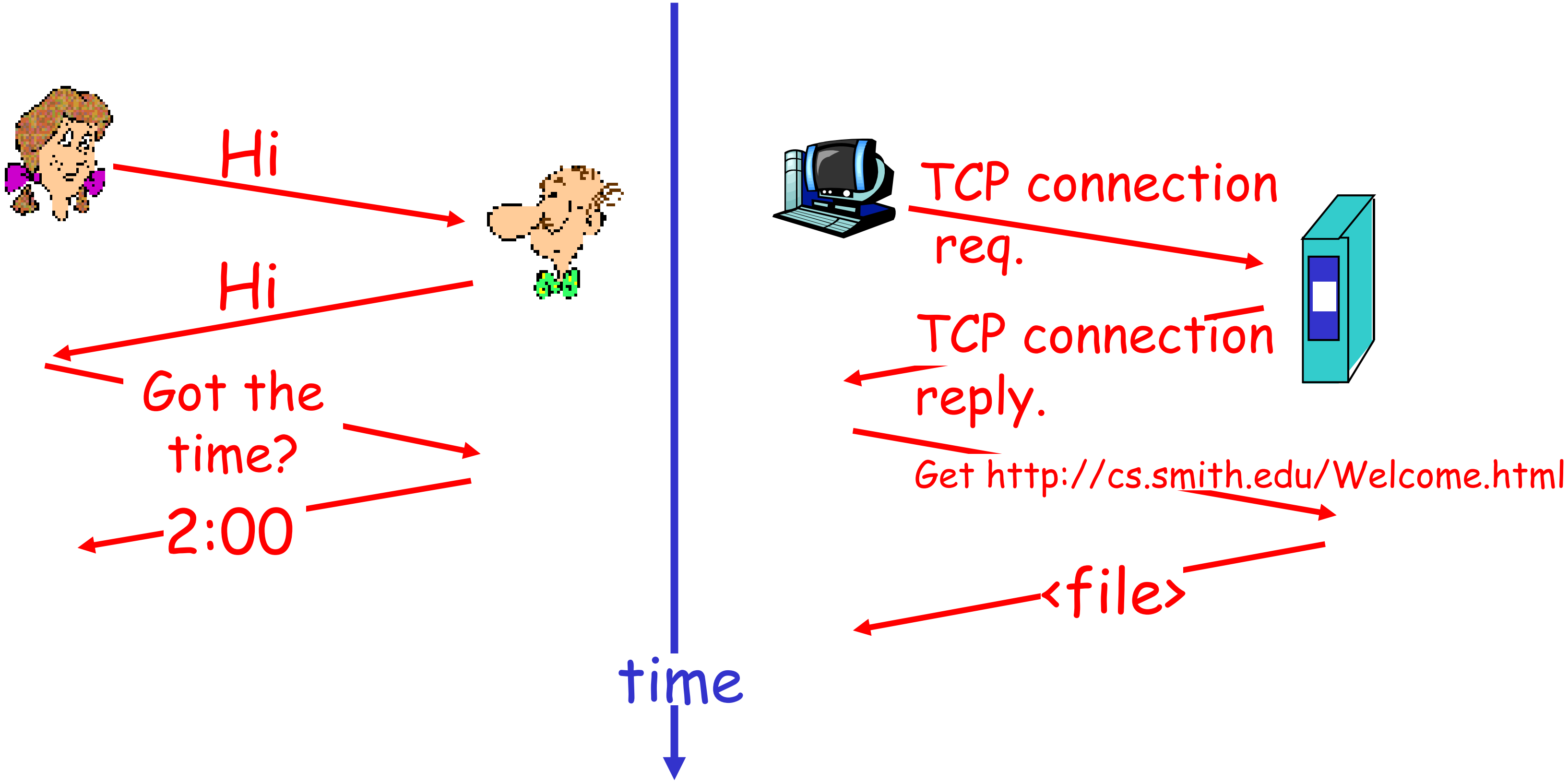
network protocols:

- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt

What's a protocol?

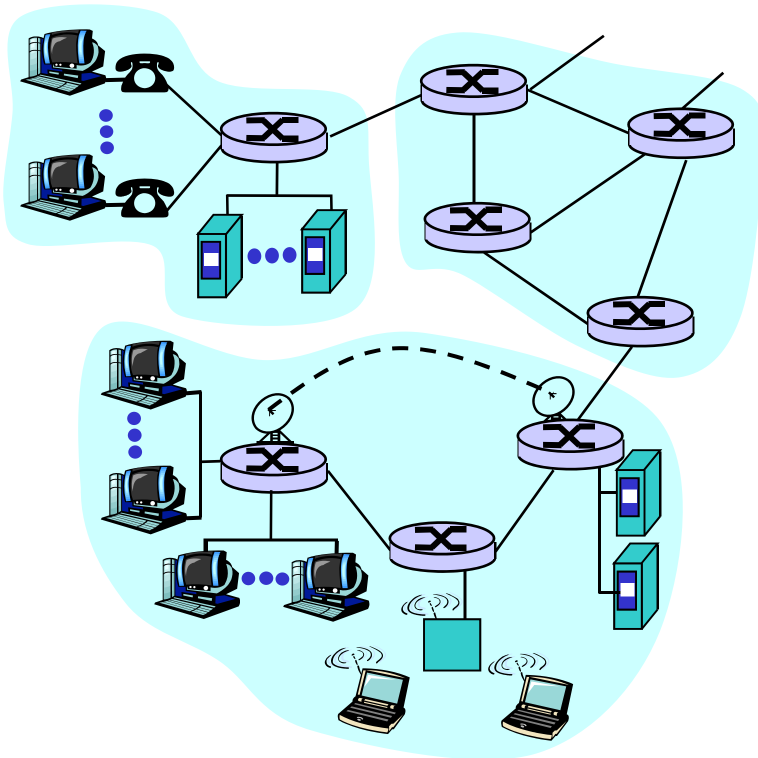
a human protocol and a computer network protocol:



Q: Other human protocol?

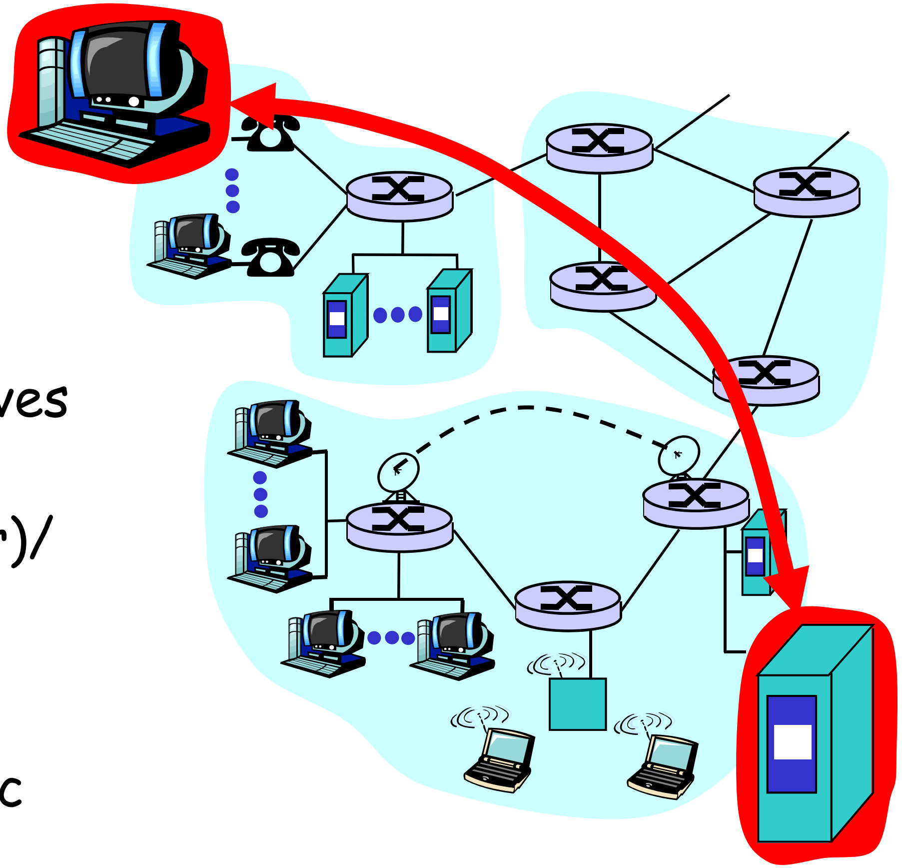
A closer look at network structure:

- network edge: applications and hosts
- network core:
  - routers
  - network of networks
- access networks, physical media: communication links



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



Network edge: connection-oriented service

Goal: data transfer between end sys.

- *handshaking:* setup (prepare for) data transfer ahead of time
  - Hello, hello back human protocol
  - *set up "state"* in two communicating hosts
- TCP - Transmission Control Protocol
  - Internet's connection-oriented service

TCP service [RFC 793]

- *reliable, in-order* byte-stream data transfer
  - loss: acknowledgements and retransmissions
- *flow control:*
  - sender won't overwhelm receiver
- *congestion control:*
  - senders "slow down sending rate" when network congested

Network edge: connectionless service

Goal: data transfer between end systems
- same as before!

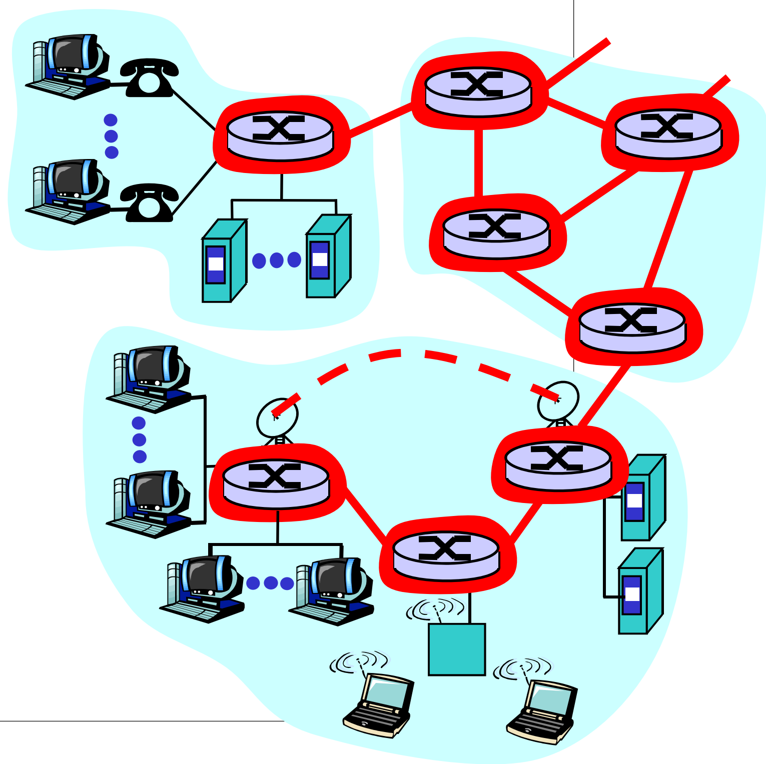
- UDP - User Datagram Protocol [RFC 768]: Internet's connectionless service
  - unreliable data transfer
  - no flow control
  - no congestion control

App's using TCP:
- HTTP (WWW), FTP (file transfer), Telnet (remote login), SMTP (email)

App's using UDP:
- streaming media, teleconferencing, Internet telephony

The Network Core

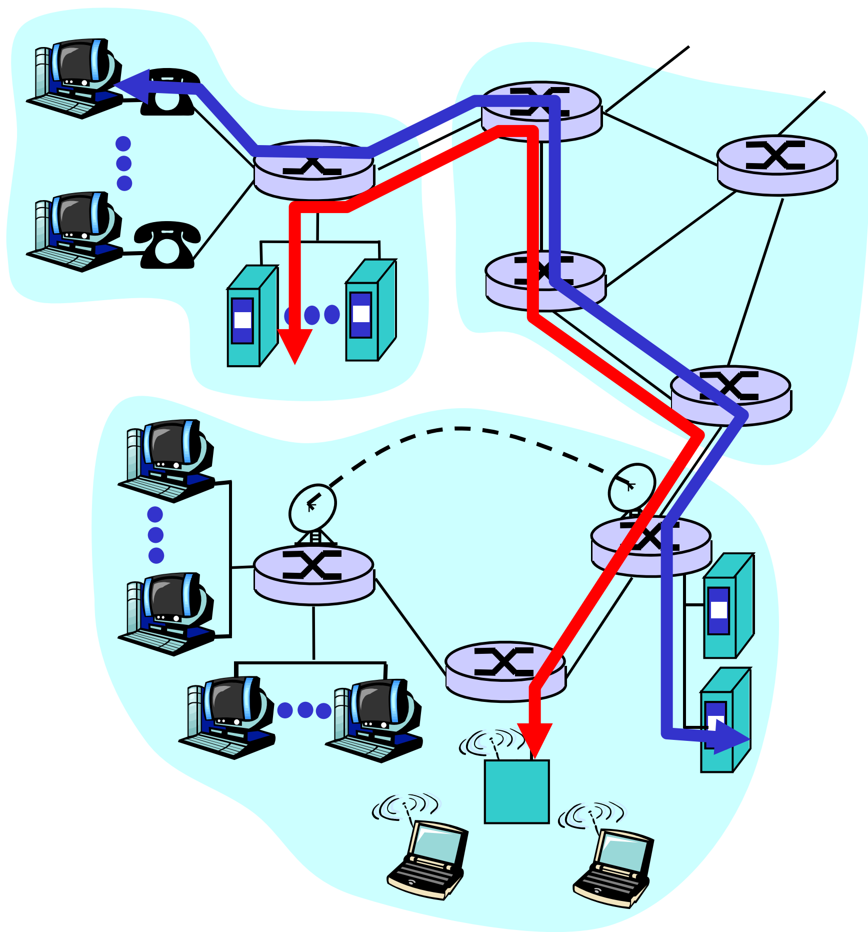
- mesh of interconnected routers
- *the fundamental question:* how is data transferred through net?
  - circuit switching: dedicated circuit per call: telephone net
  - packet-switching: data sent thru net in discrete "chunks"



Network Core: Circuit Switching

End-to-end resources reserved for "call"

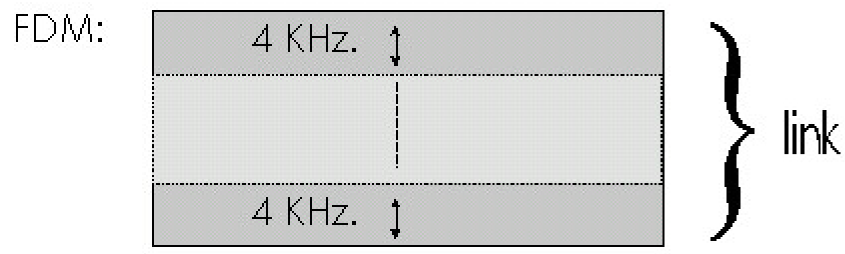
- link bandwidth, switch capacity
- dedicated resources: no sharing
- circuit-like (guaranteed) performance
- call setup required

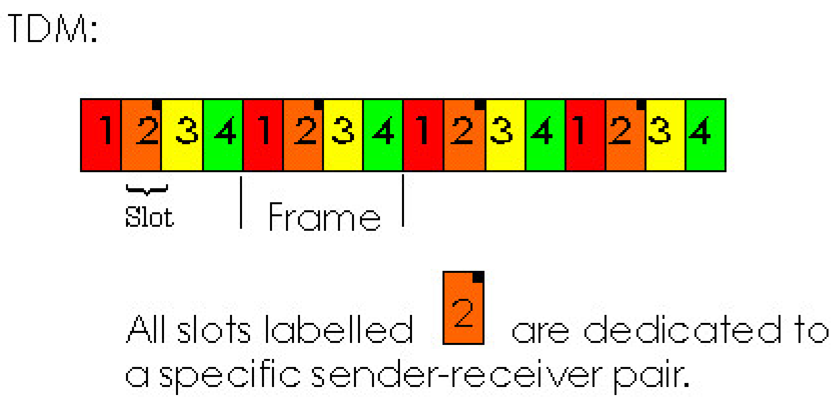


Network Core: Circuit Switching

network resources (e.g., bandwidth) divided into "pieces"

- pieces allocated to calls
- resource piece *idle* if not used by owning call *(no sharing)*
- dividing link bandwidth into "pieces"
  - frequency division
  - time division





Network Core: Packet Switching

each end-end data stream divided into *packets*

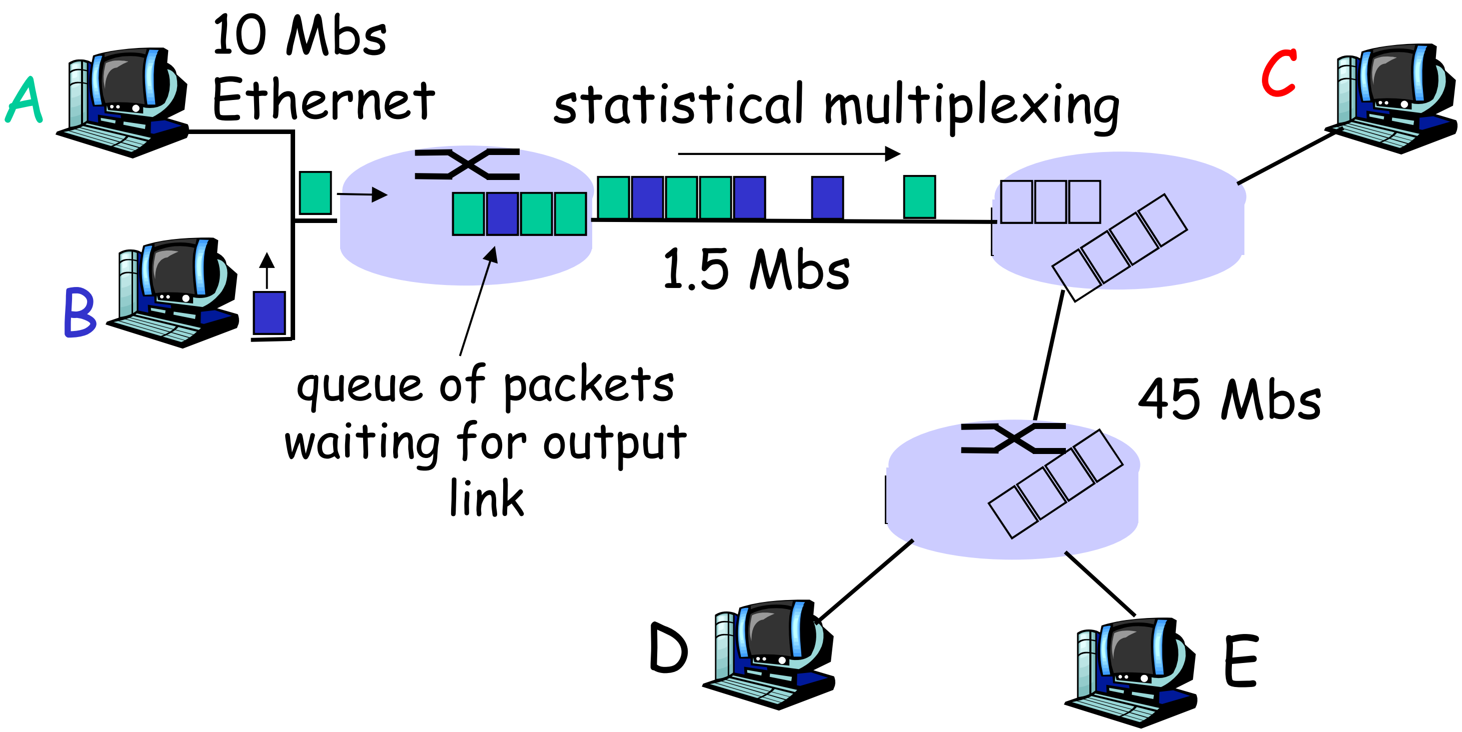
- user A, B packets *share* network resources
- each packet uses full link bandwidth
- resources used *as needed,*

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
  - transmit over link
  - wait turn at next link

Network Core: Packet Switching

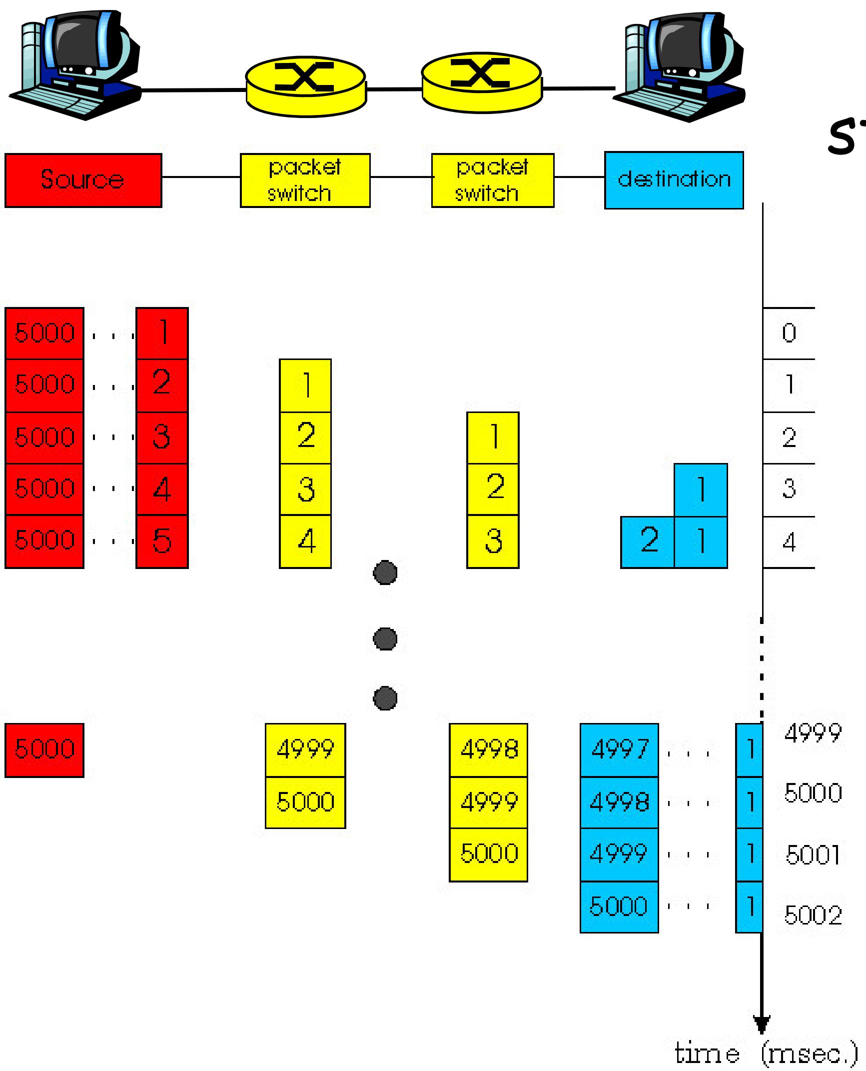


Packet-switching versus circuit switching: human restaurant analogy

- other human analogies?

Network Core: Packet Switching

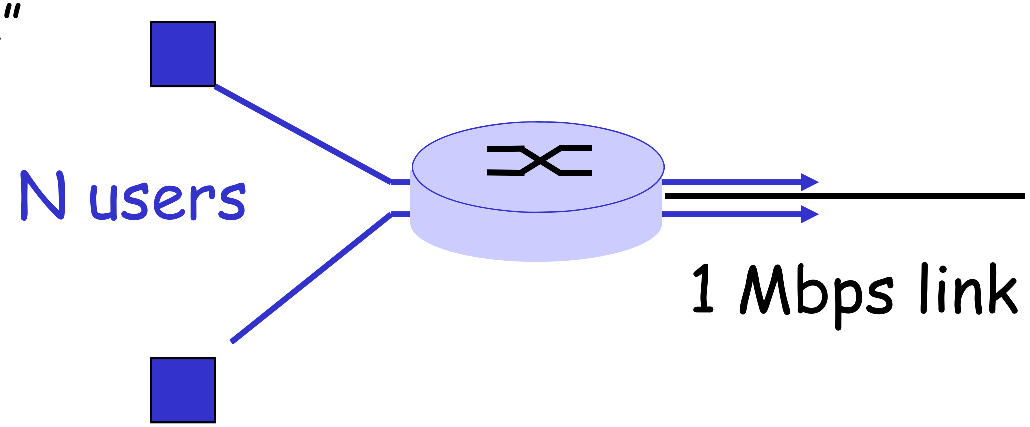
Packet-switching:
store and forward behavior



Packet switching versus circuit switching

Packet switching allows more users to use network!

- 1 Mbit link (1Mbps)
- each user:
  - 100Kbps when "active"
  - active 10% of time
- circuit-switching:
  - 10 users
- packet switching:
  - with 35 users, probability > 10 active less that .0017



Packet switching versus circuit switching

Is packet switching a "slam dunk winner?"

- Great for bursty data
  - resource sharing
  - no call setup
- Excessive congestion: packet delay and loss
  - protocols needed for reliable data transfer, congestion control
- Q: How to provide circuit-like behavior?
  - bandwidth guarantees needed for audio/video apps

  still an unsolved problem (chapter 6)

Packet-switched networks: routing

- *Goal:* move packets among routers from source to destination
  - we'll study several path selection algorithms (chapter 4)
- datagram network:
  - *destination address* determines next hop
  - routes may change during session
  - analogy: driving, asking directions
- virtual circuit network:
  - each packet carries tag (virtual circuit ID), tag determines next hop
  - fixed path determined at *call setup time*, remains fixed thru call
  - routers maintain per-call state

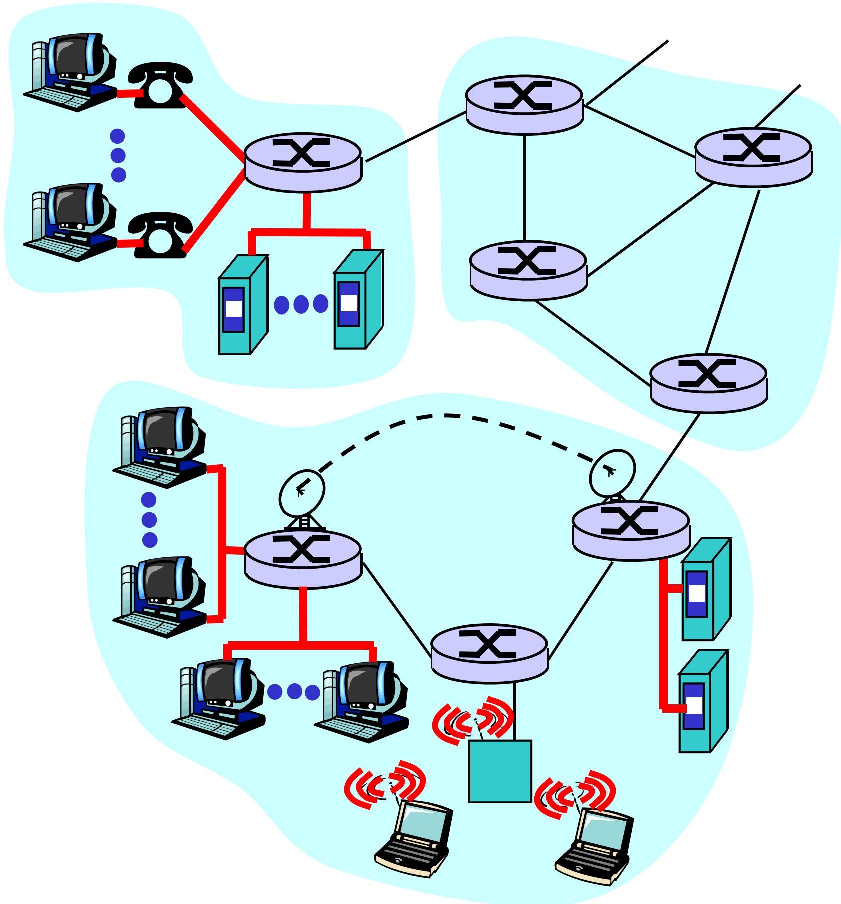
Access networks and physical media

Q: How to connect end systems to edge router?

- residential access nets
- institutional access networks (school, company)
- mobile access networks

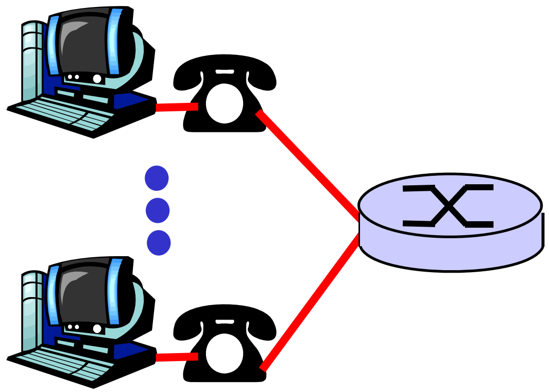
Keep in mind:

- bandwidth (bits per second) of access network?
- shared or dedicated?



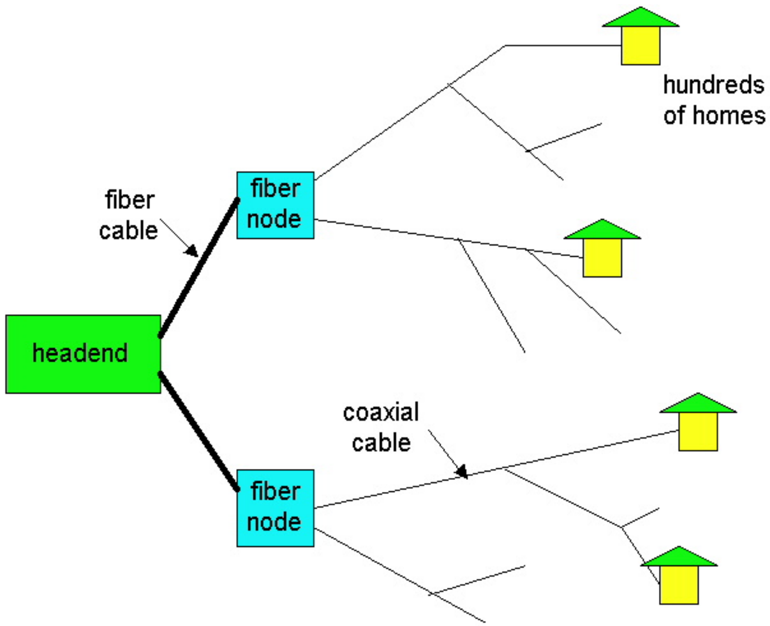
Residential access: point to point access

- Dialup via modem
  - up to 56Kbps direct access to router (conceptually)
- ISDN: integrated services digital network: 128Kbps all-digital connection to router
- ADSL: asymmetric digital subscriber line
  - up to 1 Mbps home-to-router
  - up to 8 Mbps router-to-home
  - ADSL deployment - 2 million lines in U.S. and Canada



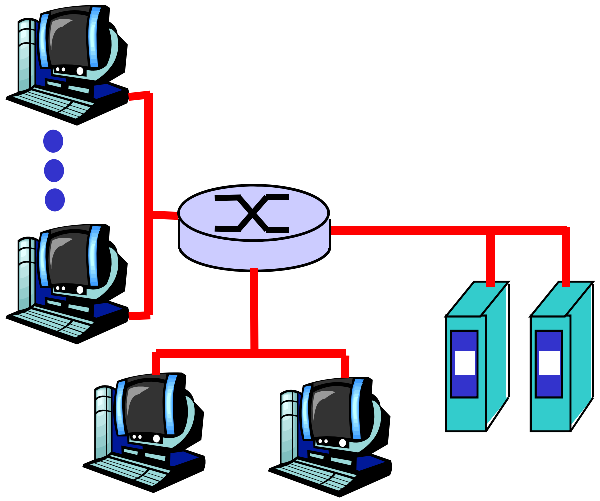
Residential access: cable modems

- HFC: hybrid fiber coax
  - asymmetric: up to 10Mbps downstream, 1 Mbps upstream
- network of cable and fiber attaches homes to ISP router
  - shared access to router among homes
  - issues: congestion, dimensioning
- deployment: available via cable companies, e.g., MediaOne



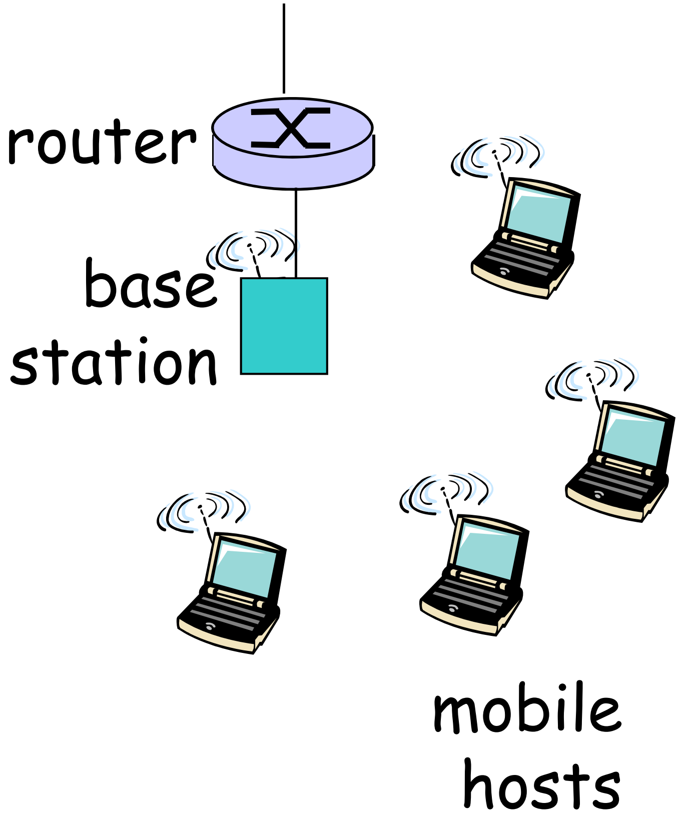
Institutional access: local area networks

- company/univ local area network (LAN) connects end system to edge router
- Ethernet:
  - shared or dedicated cable connects end system and router
  - 10 Mbs, 100Mbps, Gigabit Ethernet
- deployment: institutions, home LANs soon
- LANs: chapter 5



Wireless access networks

- shared *wireless* access network connects end system to router
- wireless LANs:
  - radio spectrum replaces wire
  - e.g., Lucent Wavelan 10 Mbps
- wider-area wireless access
  - CDPD: wireless access to ISP router via cellular network



Physical Media

- physical link: transmitted data bit propagates across link
- guided media:
  - signals propagate in solid media: copper, fiber
- unguided media:
  - signals propagate freely e.g., radio

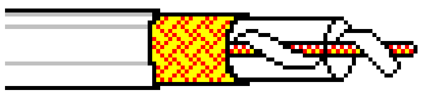
Twisted Pair (TP)

- two insulated copper wires
  - Category 3: traditional phone wires, 10 Mbps ethernet
  - Category 5 TP: 100Mbps ethernet

Physical Media: coax, fiber

Coaxial cable:

- wire (signal carrier) within a wire (shield)
  - baseband: single channel on cable
  - broadband: multiple channel on cable
- bidirectional
- common use in 10Mbs Ethernet



Fiber optic cable:

- glass fiber carrying light pulses
- high-speed operation:
  - 100Mbps Ethernet
  - high-speed point-to-point transmission (e.g., 5 Gps)
- low error rate

Physical media: radio

- signal carried in electromagnetic spectrum
- no physical "wire"
- bidirectional
- propagation environment effects:
  - reflection
  - obstruction by objects
  - interference

Radio link types:

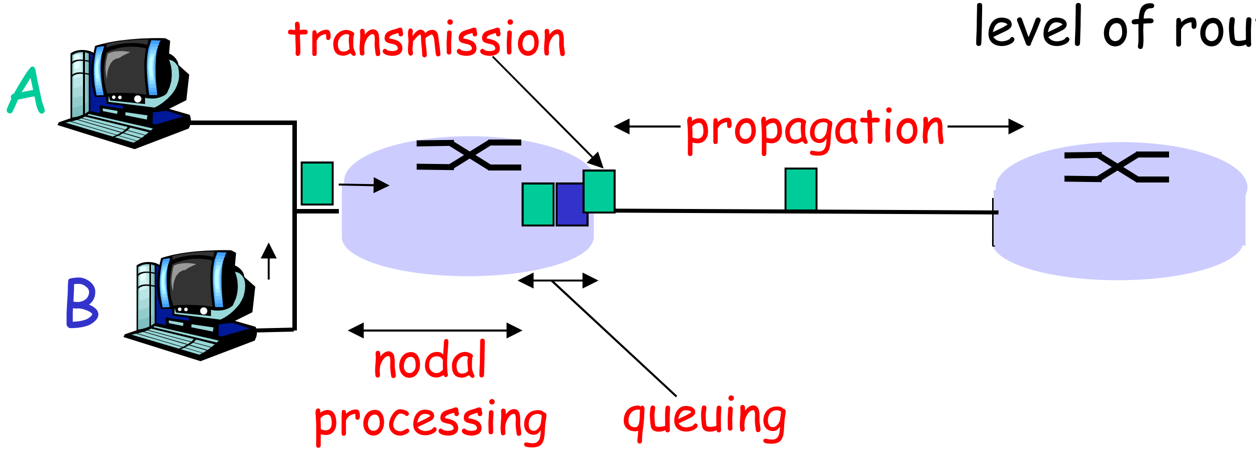
- microwave
  - e.g. up to 45 Mbps channels
- LAN (e.g., waveLAN)
  - 2Mbps, 11Mbps
- wide-area (e.g., cellular)
  - e.g. CDPD, 10's Kbps
- satellite
  - up to 50Mbps channel (or multiple smaller channels)
  - 270 Msec end-end delay
  - geosynchronous versus LEOS

Delay in packet-switched networks

packets experience delay on end-to-end path

- four sources of delay at each hop

- nodal processing:
  - check bit errors
  - determine output link
- queuing
  - time waiting at output link for transmission
  - depends on congestion level of router

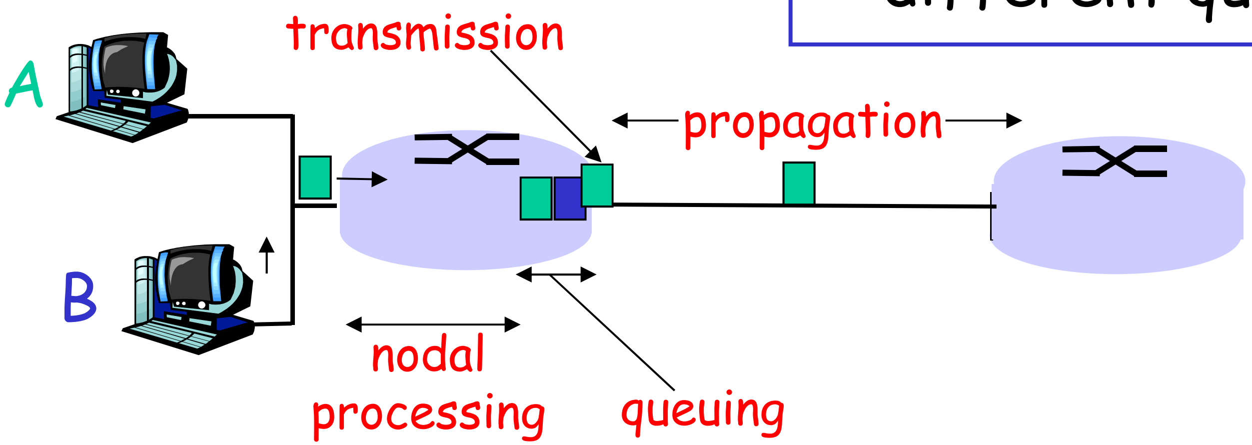


Delay in packet-switched networks

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

Propagation delay:
- d = length of physical link
- s = propagation speed in medium (~$2x10^8$ m/sec)
- propagation delay = d/s

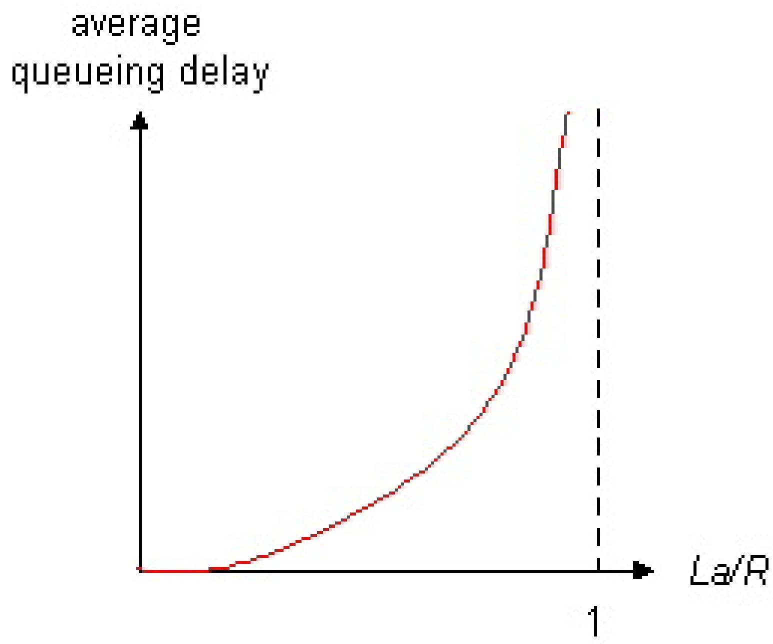
Note: s and R are very different quantities!



Queuing delay (revisited)

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

traffic intensity = La/R



- La/R ~ 0: average queueing delay small
- La/R -> 1: delays become large
- La/R > 1: more "work" arriving than can be serviced, average delay infinite!

Protocol "Layers"

Networks are complex!

- many "pieces":
  - hosts
  - routers
  - links of various media
  - applications
  - protocols
  - hardware, software

Question:

Is there any hope of *organizing* structure of network?

Or at least our discussion of networks?

Organization of air travel

ticket (purchase)	ticket (complain)
baggage (check)	baggage (claim)
gates (load)	gates (unload)
runway takeoff	runway landing
airplane routing	airplane routing

airplane routing

- a series of steps

Organization of air travel: a different view

ticket (purchase)	ticket (complain)
baggage (check)	baggage (claim)
gates (load)	gates (unload)
runway takeoff	runway landing
airplane routing	airplane routing
airplane routing	

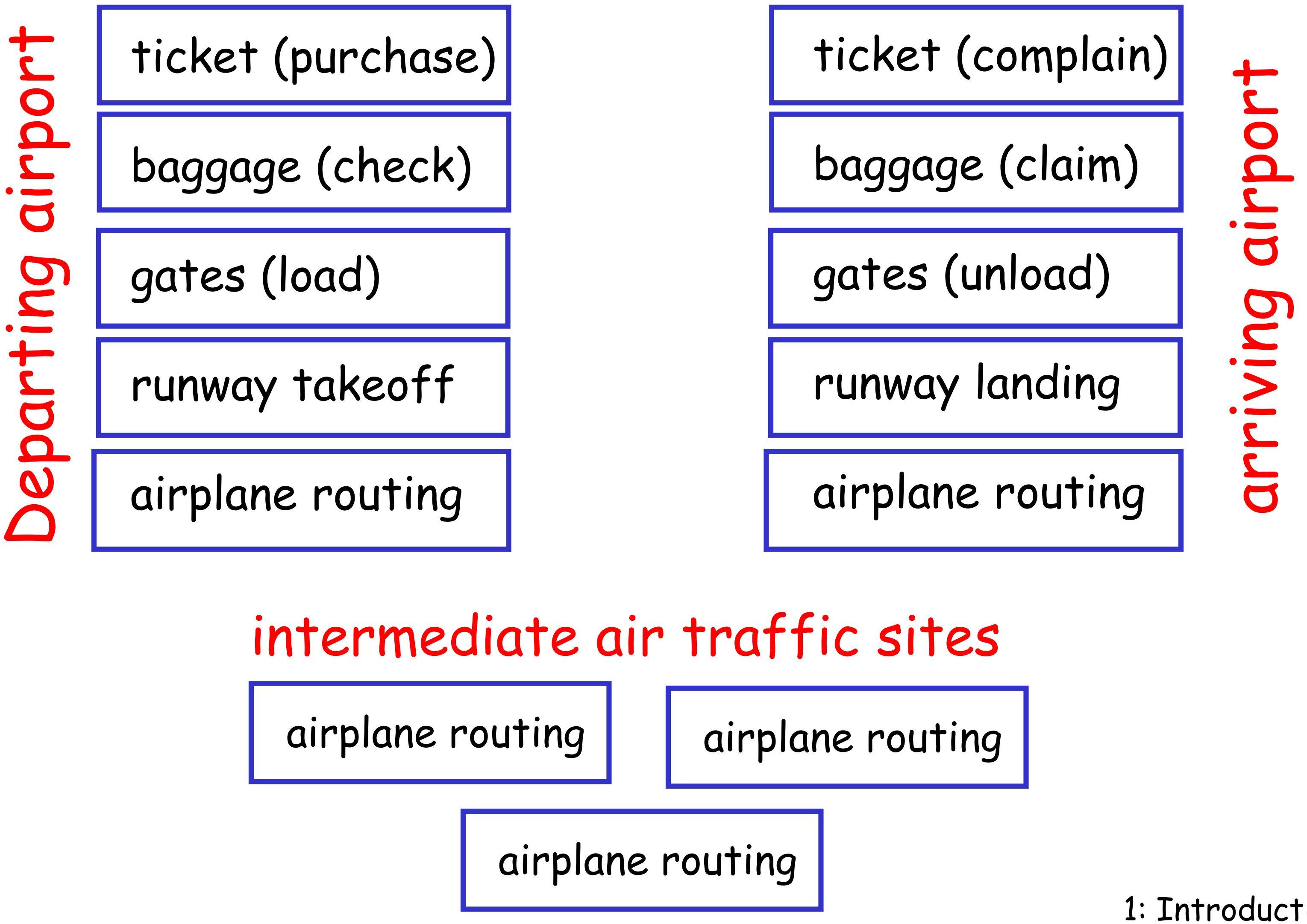
Layers: each layer implements a service

- via its own internal-layer actions
- relying on services provided by layer below

Layered air travel: services

- Counter-to-counter delivery of person+bags
- baggage-claim-to-baggage-claim delivery
- people transfer: loading gate to arrival gate
- runway-to-runway delivery of plane
- airplane routing from source to destination

Distributed implementation of layer functionality



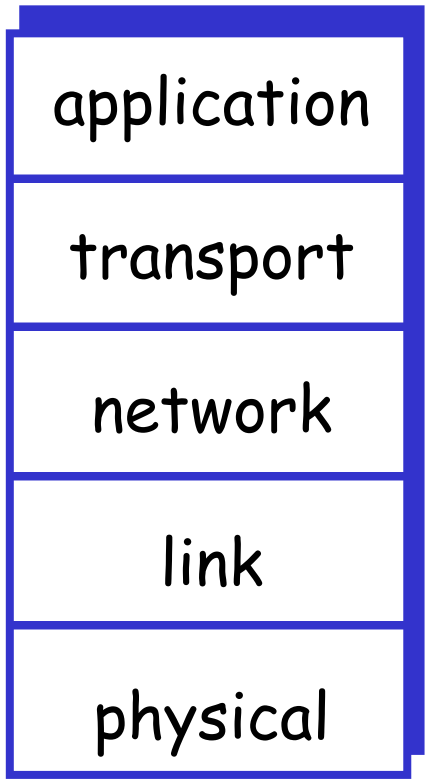
Why layering?

Dealing with complex systems:

- explicit structure allows identification, relationship of complex system's pieces
  - layered reference model for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in gate procedure doesn't affect rest of system
- layering considered harmful?

Internet protocol stack

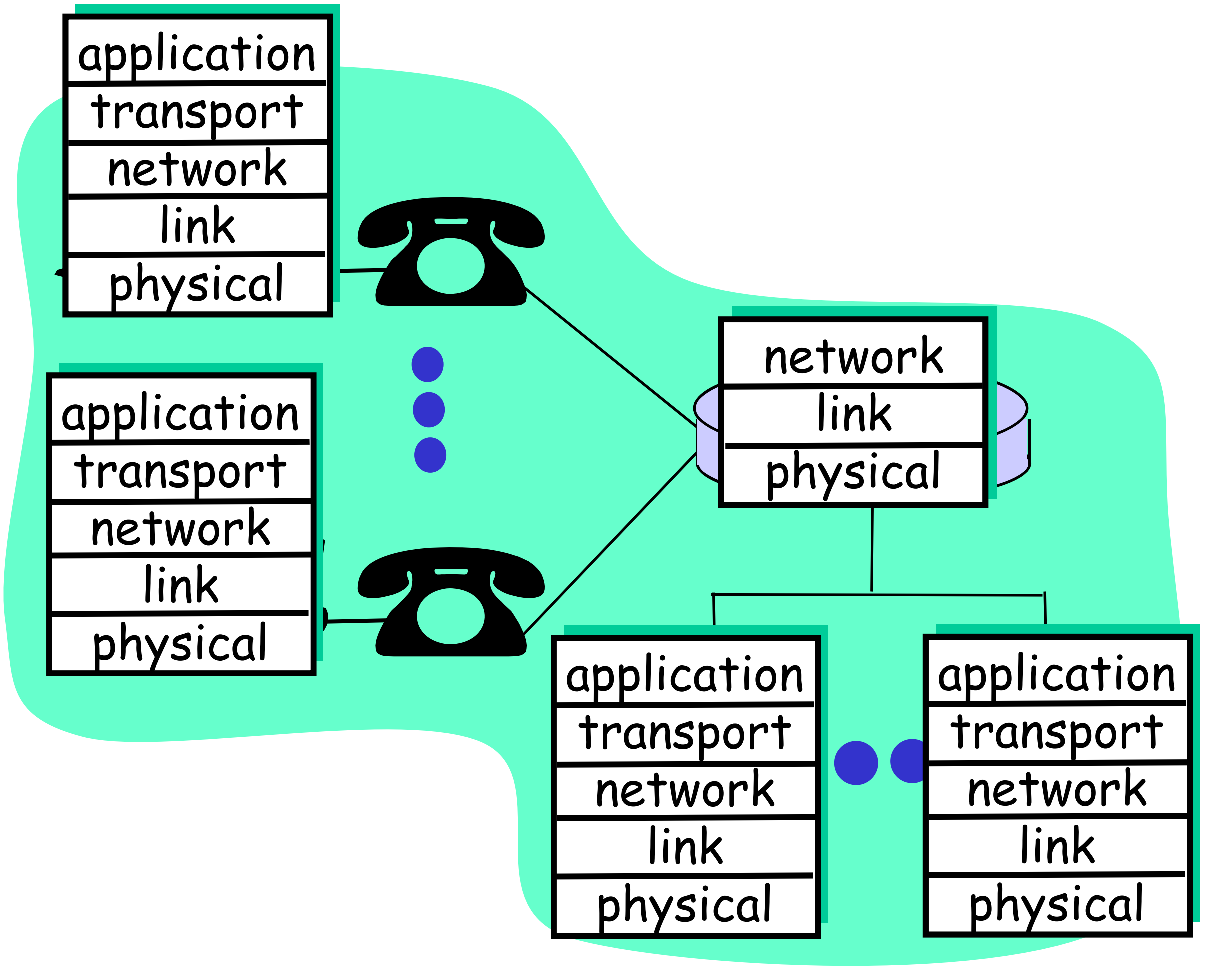
- application: supporting network applications
  - ftp, smtp, http
- transport: host-host data transfer
  - tcp, udp
- network: routing of datagrams from source to destination
  - ip, routing protocols
- link: data transfer between neighboring network elements
  - ppp, ethernet, ATM
- physical: bits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



Layering: logical communication

Each layer:

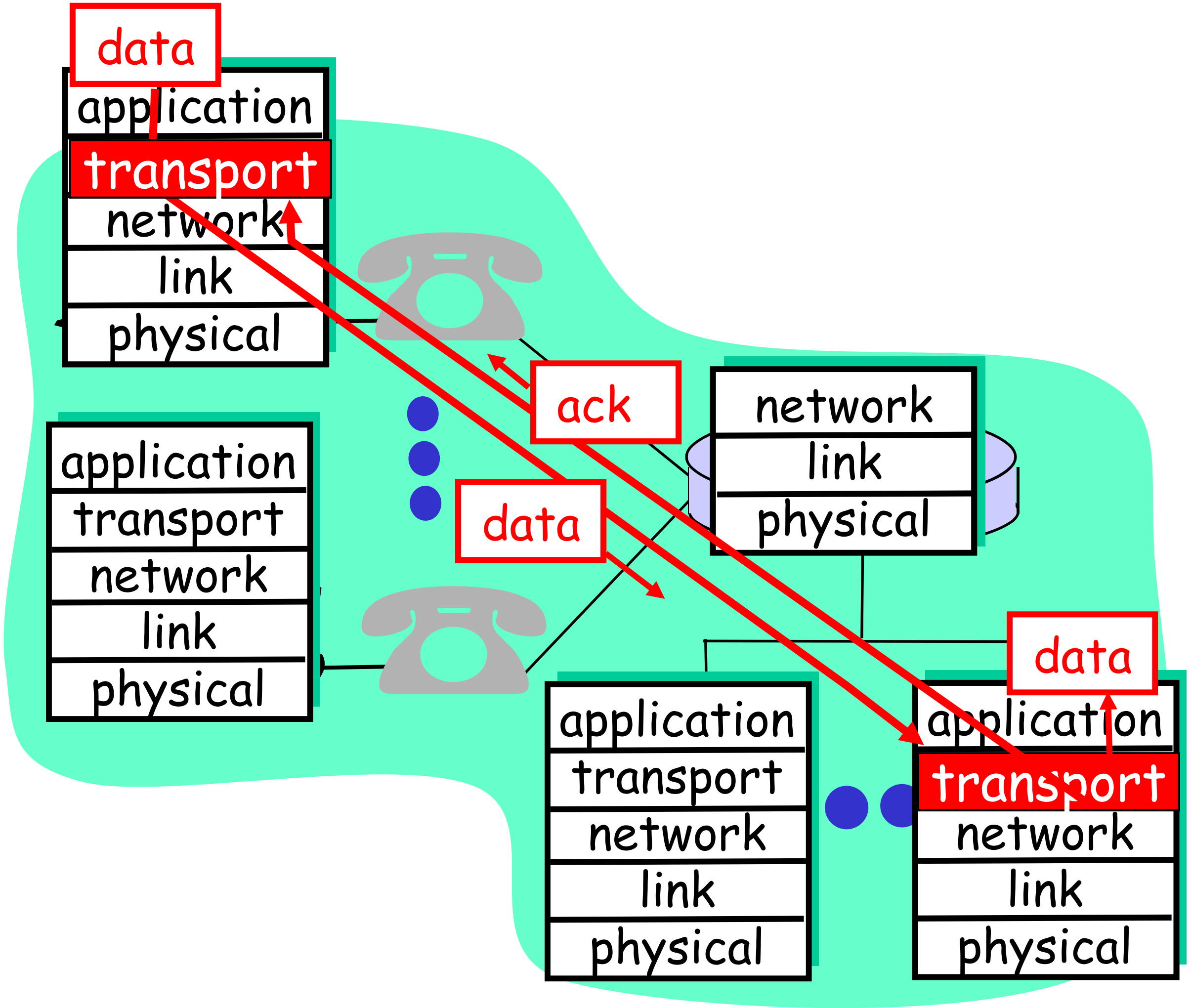
- distributed
- "entities" implement layer functions at each node
- entities perform actions, exchange messages with peers



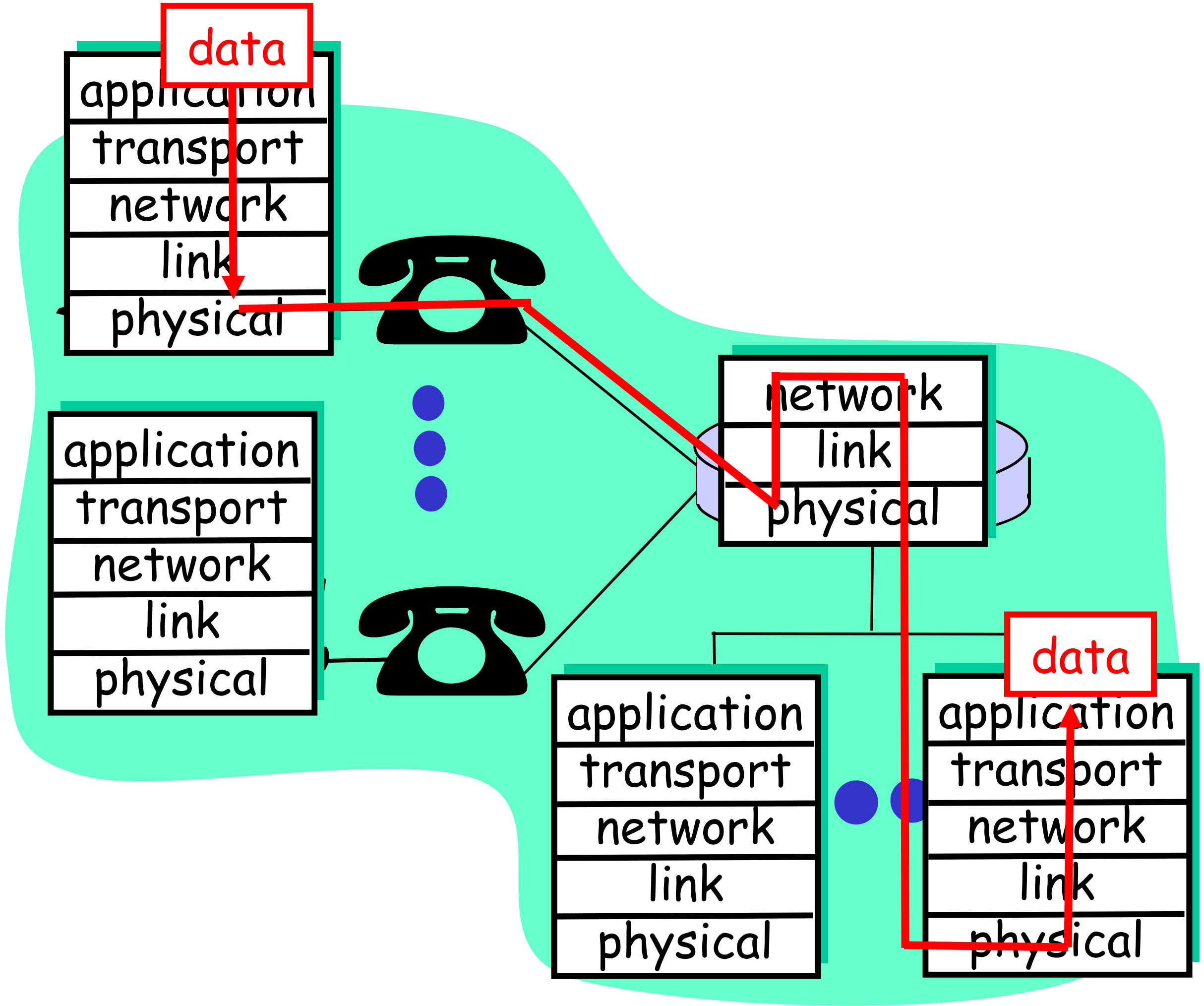
Layering: *logical* communication

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
- analogy: post office



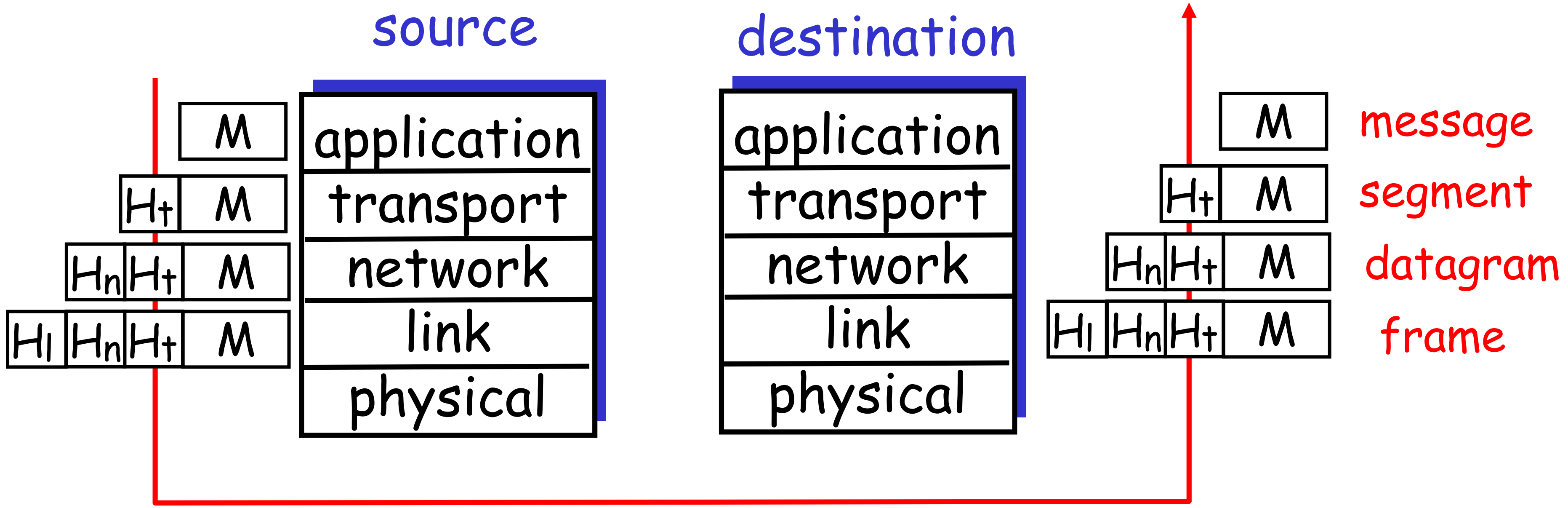
Layering: physical communication



Protocol layering and data

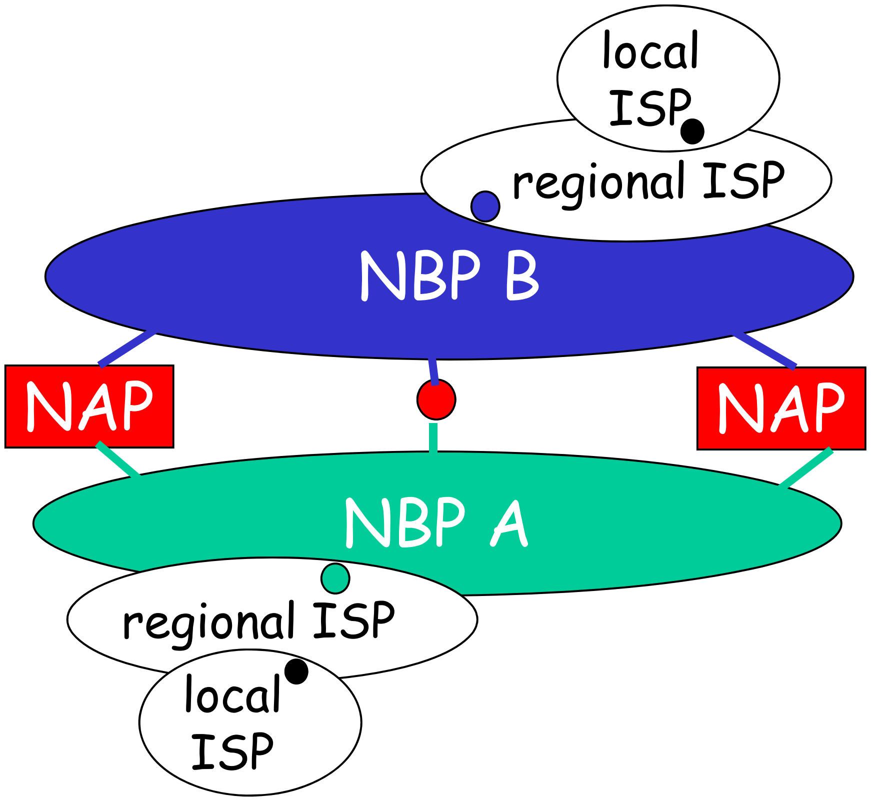
Each layer takes data from above

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



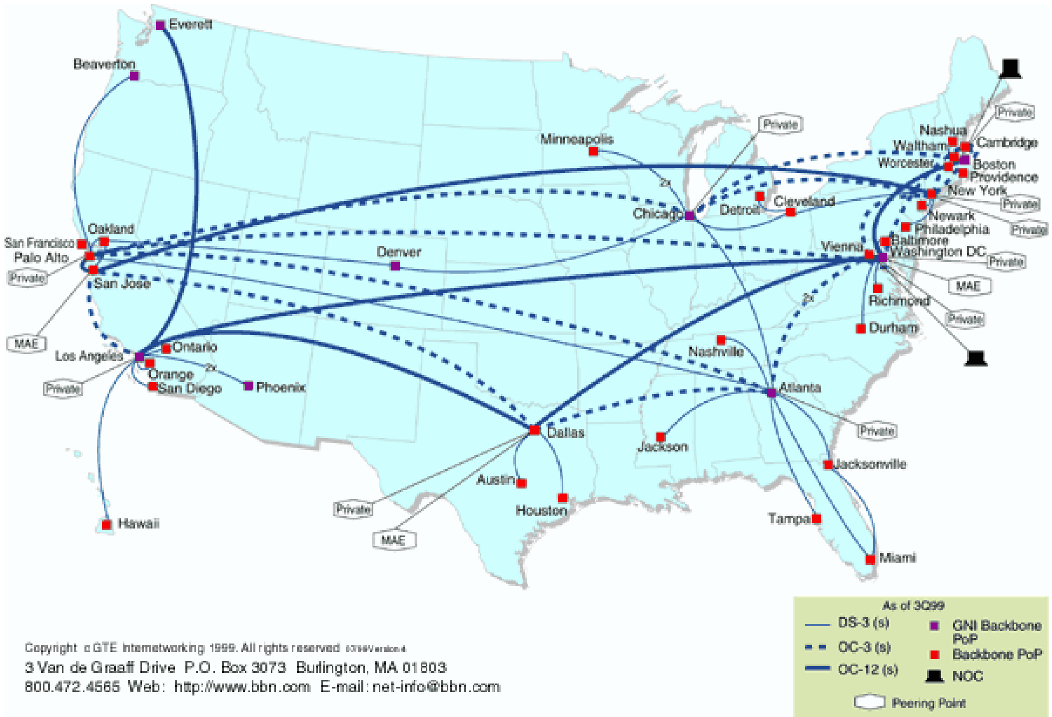
Internet structure: network of networks

- roughly hierarchical
- national/international backbone providers (NBPs)
  - e.g. BBN/GTE, Sprint, AT&T, IBM, UUNet
  - interconnect (peer) with each other privately, or at public Network Access Point (NAPs)
- regional ISPs
  - connect into NBPs
- local ISP, company
  - connect into regional ISPs



National Backbone Provider

e.g. BBN/GTE US backbone network



Internet History

1961-1972: Early packet-switching principles

- 1961: Kleinrock - queueing theory shows effectiveness of packet-switching
- 1964: Baran - packet-switching in military nets
- 1967: ARPAnet conceived by Advanced Reearch Projects Agency
- 1969: first ARPAnet node operational
- 1972:
  - ARPAnet demonstrated publicly
  - NCP (Network Control Protocol) first host-host protocol
  - first e-mail program
  - ARPAnet has 15 nodes

Internet History

1972-1980: Internetworking, new and proprietary nets

- 1970: ALOHAnet satellite network in Hawaii
- 1973: Metcalfe's PhD thesis proposes Ethernet
- 1974: Cerf and Kahn - architecture for interconnecting networks
- late70's: proprietary architectures: DECnet, SNA, XNA
- late 70's: switching fixed length packets (ATM precursor)
- 1979: ARPAnet has 200 nodes

Cerf and Kahn's internetworking principles:

- minimalism, autonomy - no internal changes required to interconnect networks
- best effort service model
- stateless routers
- decentralized control

define today's Internet architecture

Internet History

1980-1990: new protocols, a proliferation of networks

- 1983: deployment of TCP/IP
- 1982: smtp e-mail protocol defined
- 1983: DNS defined for name-to-IP-address translation
- 1985: ftp protocol defined
- 1988: TCP congestion control
- new national networks: Csnet, BITnet, NSFnet, Minitel
- 100,000 hosts connected to confederation of networks

Internet History

1990's: commercialization, the WWW

- Early 1990's: ARPAnet decomissioned
- 1991: NSF lifts restrictions on commercial use of NSFnet (decommissioned, 1995)
- early 1990s: WWW
  - hypertext [Bush 1945, Nelson 1960's]
  - HTML, http: Berners-Lee
  - 1994: Mosaic, later Netscape
  - late 1990's: commercialization of the WWW

Late 1990's:

- est. 50 million computers on Internet
- est. 100 million+ users
- backbone links runnning at 1 Gbps

ATM: Asynchronous Transfer Mode nets

Internet:

- today's *de facto* standard for global data networking

1980's:

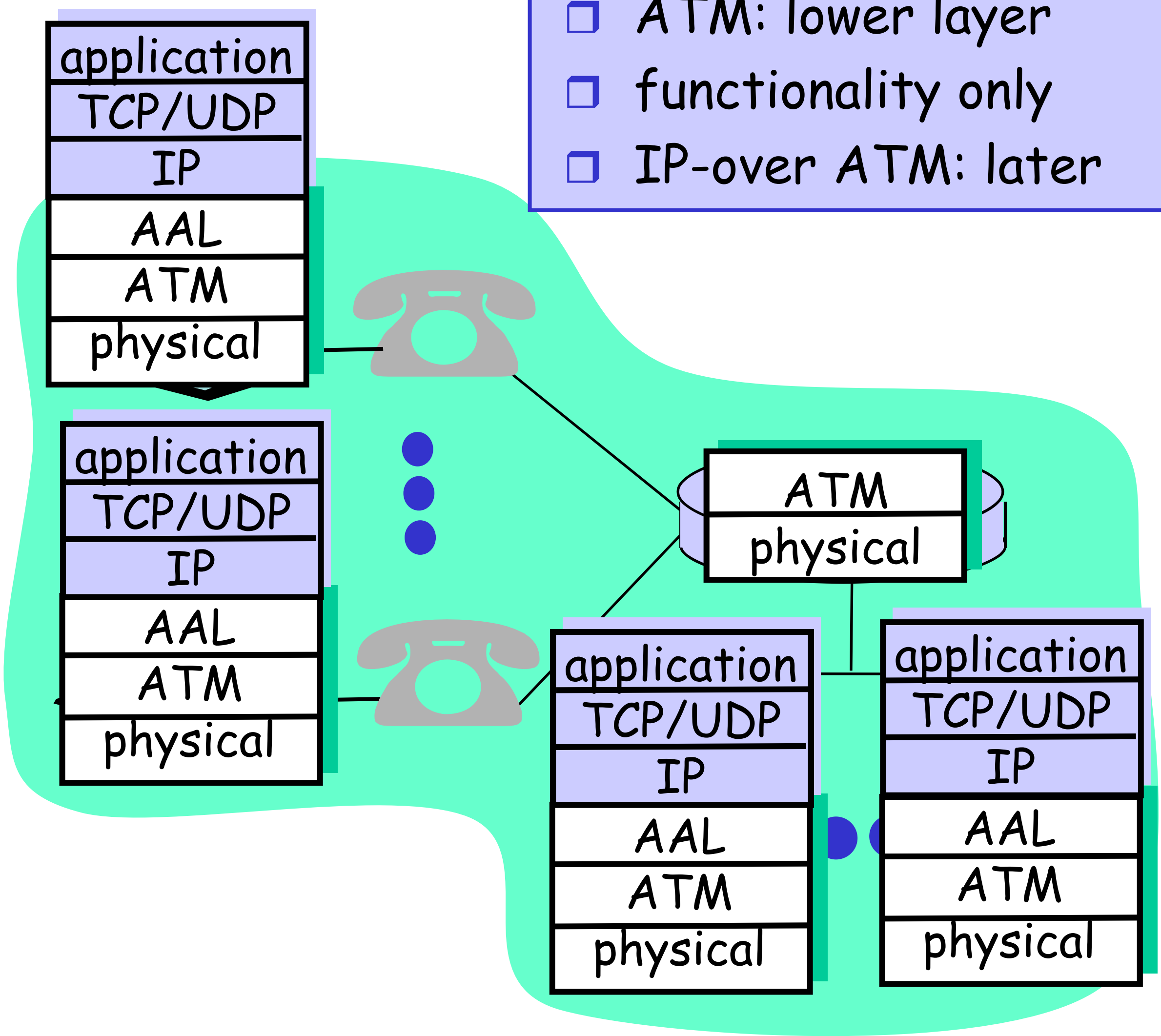
- telco's develop ATM: competing network standard for carrying high-speed voice/data
- standards bodies:
  - ATM Forum
  - ITU

ATM principles:

- small (48 byte payload, 5 byte header) fixed length *cells* (like packets)
  - fast switching
  - small size good for voice
- virtual-circuit network: switches maintain state for each "call"
- well-defined interface between "network" and "user" (think of telephone company)

ATM layers

- ATM Adaptation Layer (AAL): interface to upper layers
  - end-system
  - segmentation/reassembly
- ATM Layer: cell switching
- Physical



Where's the application?

- ATM: lower layer
- functionality only
- IP-over ATM: later

Summary on Introduction

Covered a "ton" of material!

- Internet overview
- what's a protocol?
- network edge, core, access network
- performance: loss, delay
- layering and service models
- backbones, NAPs, ISPs
- history
- ATM network

You now hopefully have:

- context, overview, "feel" of networking
- more depth, detail *later* in course

Application Layer

Goals:

- conceptual + implementation aspects of network application protocols
  - client server paradigm
  - service models
- learn about protocols by examining popular application-level protocols

More goals

- specific protocols:
  - http
  - ftp
  - smtp
  - pop
  - dns
- programming network applications
  - socket programming

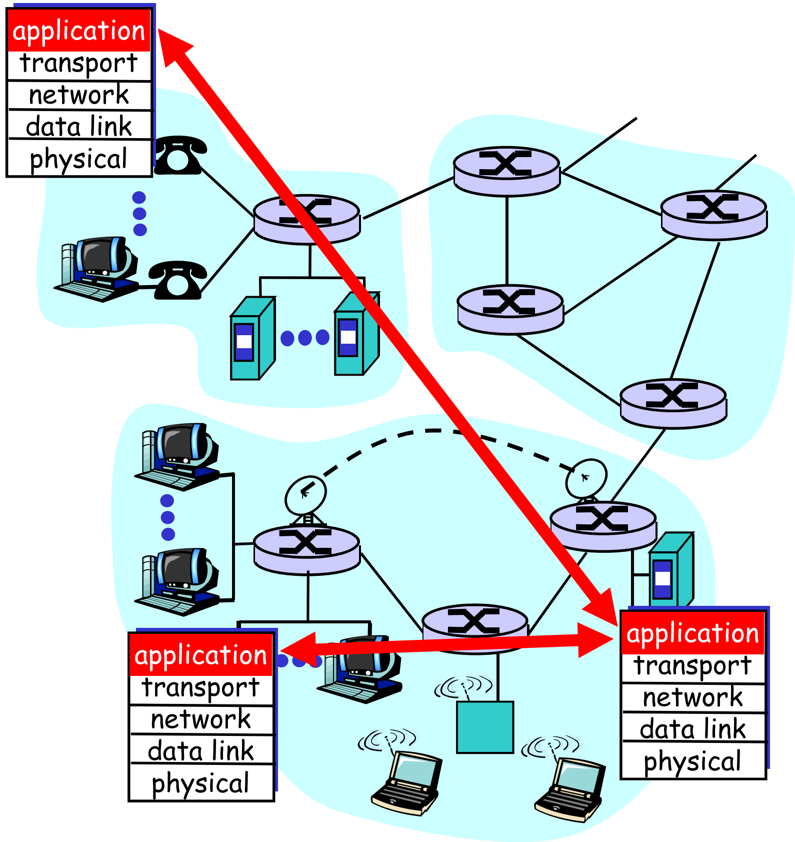
Applications and application-layer protocols

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

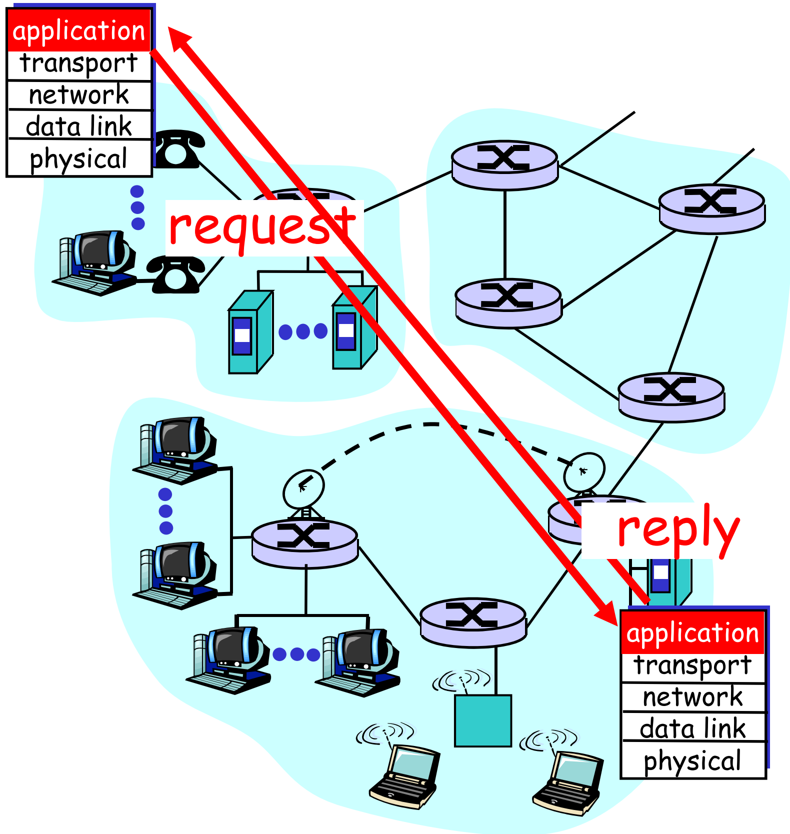
Typical network app has two pieces: *client* and *server*

Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- e.g.: request WWW page, send email

Server:

- provides requested service to client
- e.g., sends requested WWW page, receives/stores received email



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

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

… lots more on this later.

What transport service does an app need?

Data loss

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

Bandwidth

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

Timing

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

Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
financial apps	no loss	elastic	yes and no

Internet apps: their protocols and transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NSF	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

Services provided by Internet transport protocols

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

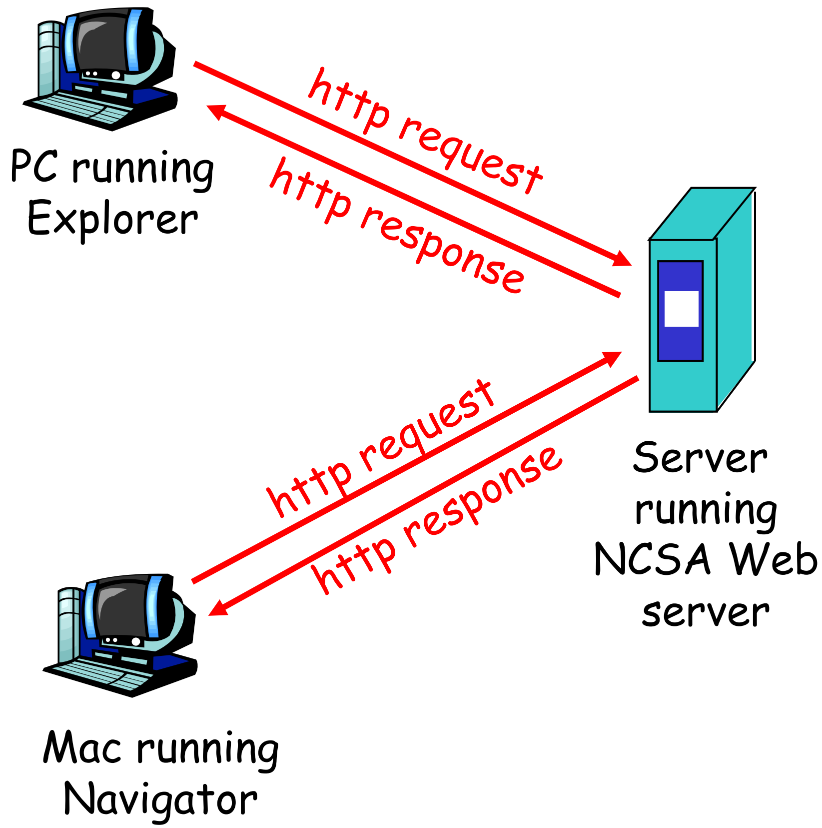
Internet apps: their protocols and transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NSF	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

WWW: the http protocol

http: hypertext transfer protocol

- WWW's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" WWW objects
  - *server:* WWW server sends objects in response to requests
- http1.0: RFC 1945
- http1.1: RFC 2068



The http protocol: more

http: TCP transport service:

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

http is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

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

http 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

- non-persistent connection: one object in each TCP connection
  - some browsers create multiple TCP connections *simultaneously* - one per object
- persistent connection: multiple objects transferred within one TCP connection

http message format: request

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

request line (GET, POST, HEAD commands)

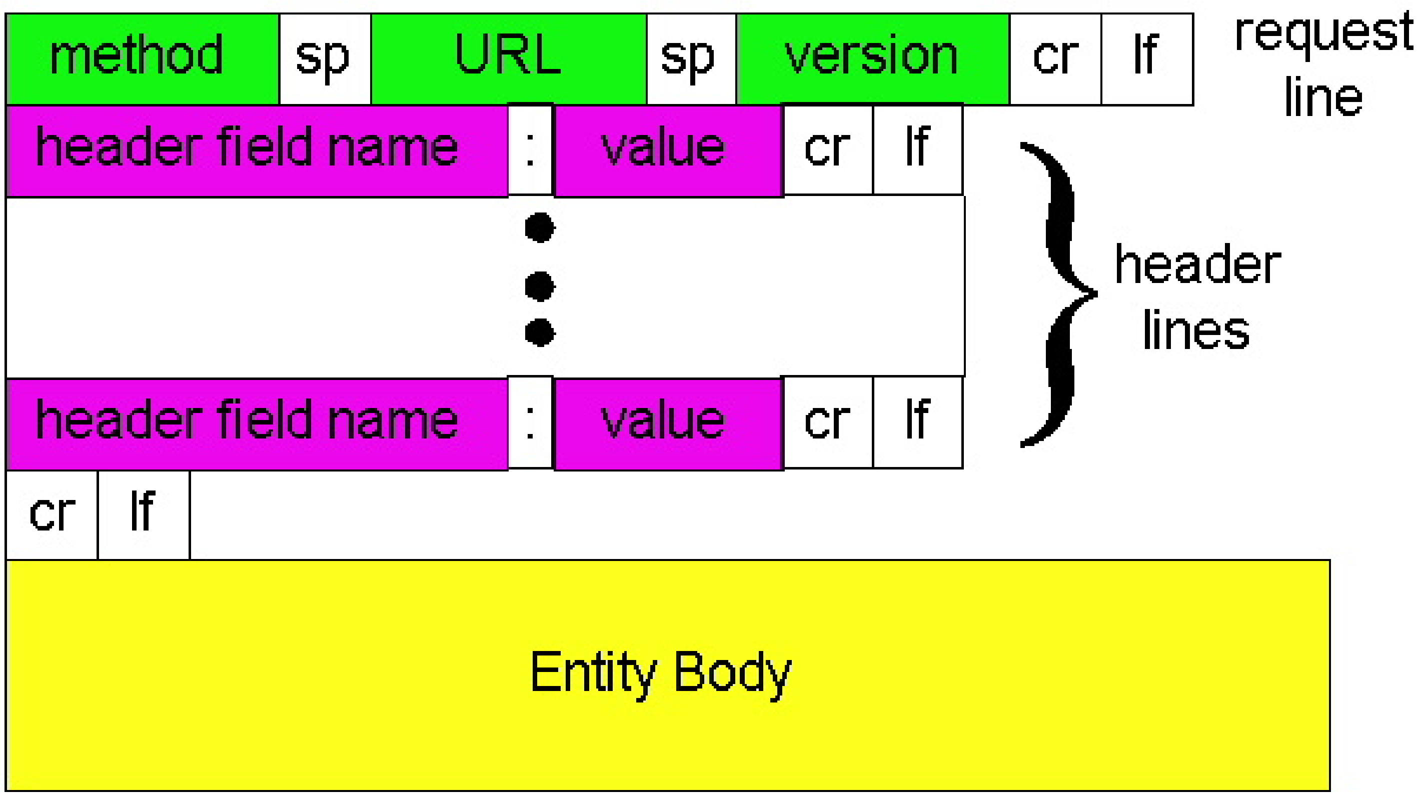
header lines

Carriage return, line feed indicates end of message

```
GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr

(extra carriage return, line feed)
```

http request message: general format



http message format: reply

status line
(protocol
status code
status phrase)

header
lines

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html
data data data data data ...
```

data, e.g.,
requested
html file

http reply status codes

In first line in server->client response message.

A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out http (client side) for yourself

1. Telnet to your favorite WWW server:

```
telnet www.eurecom.fr 80
```

Opens TCP connection to port 80 (default http server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET http request:

```
GET /~ross/index.html HTTP/1.0
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by http server!

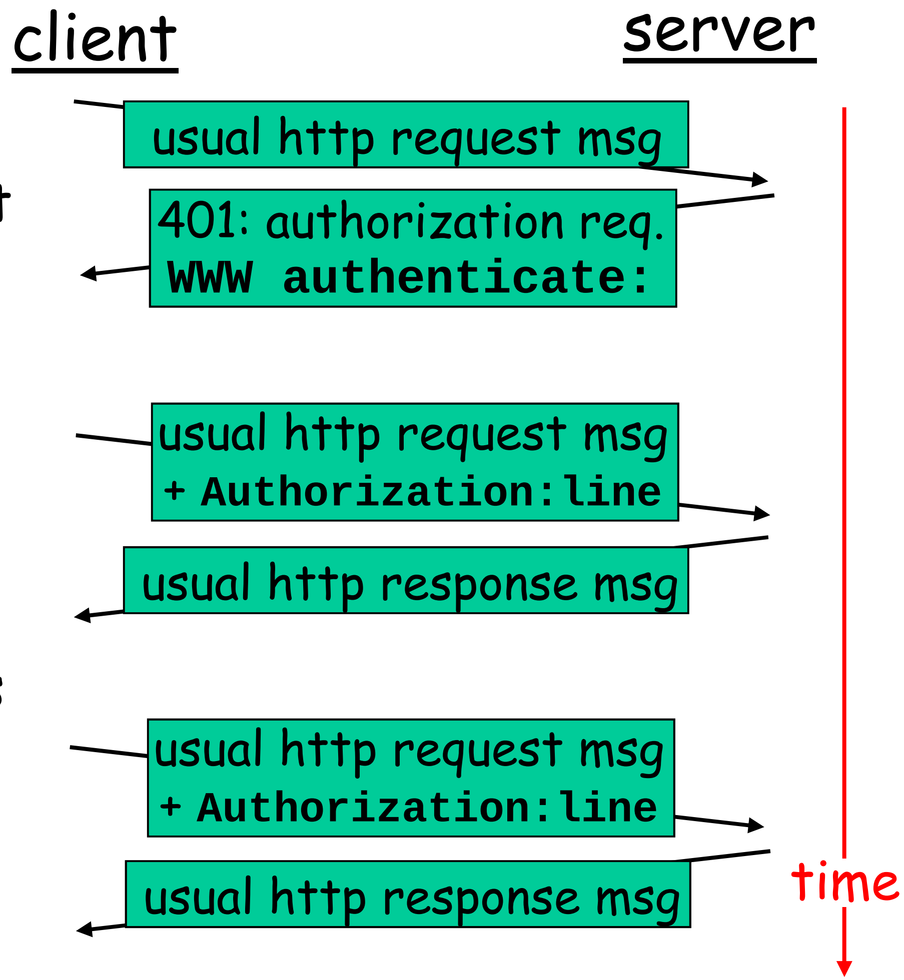
User-server interaction: authentication

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

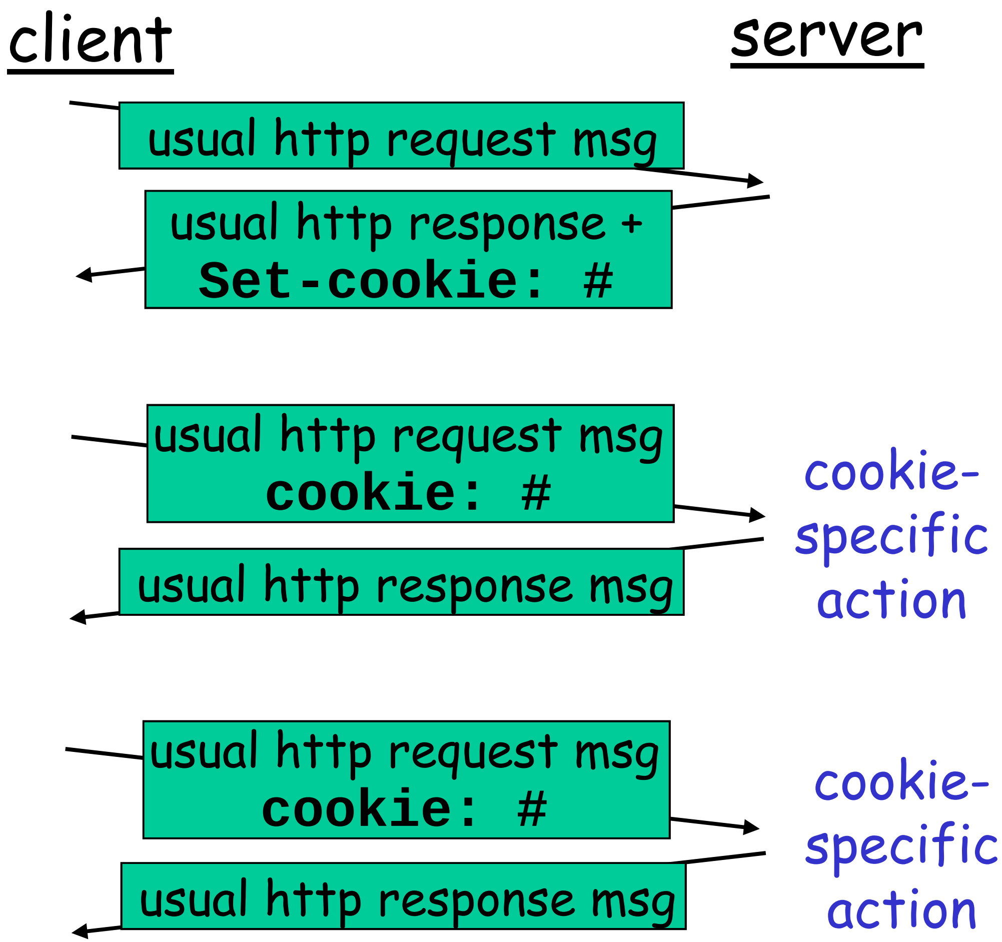
    `WWW authenticate:`

    header line in response



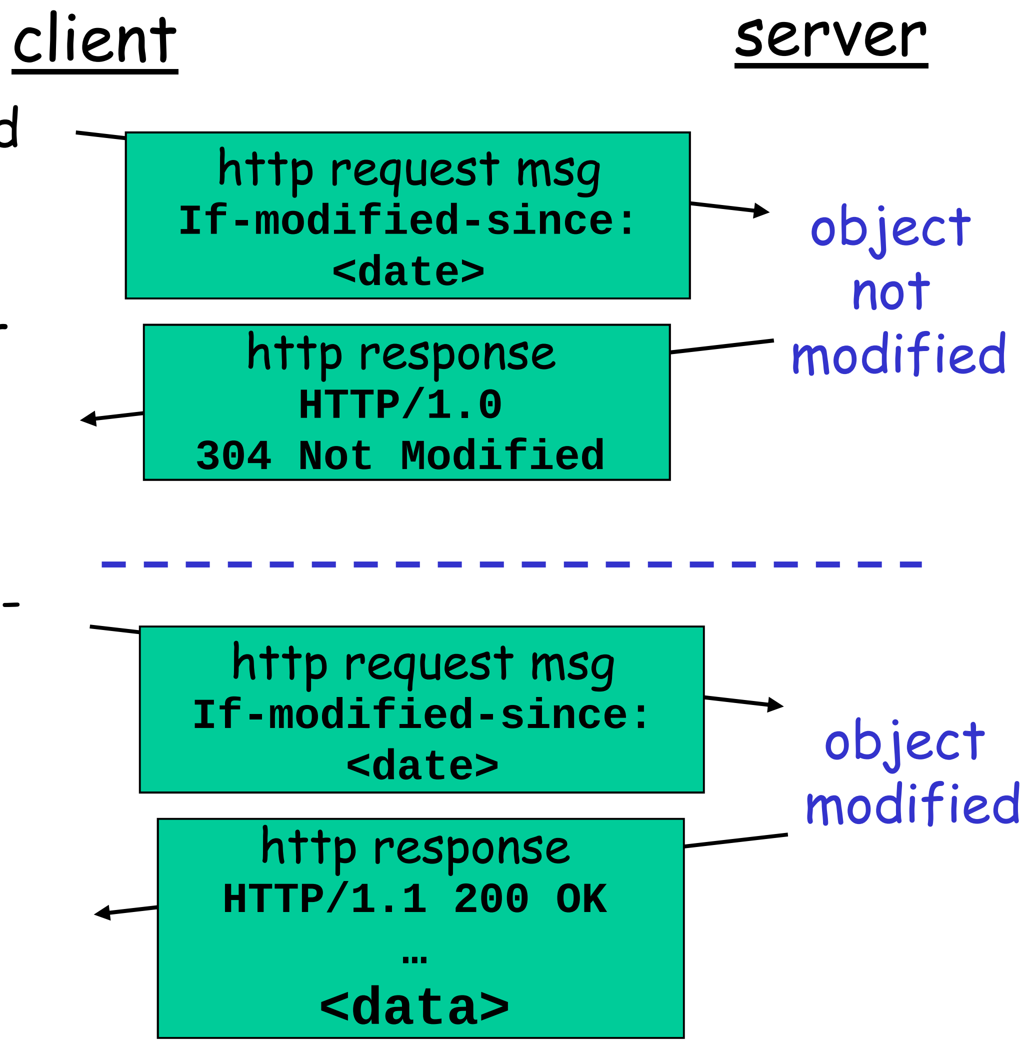
User-server interaction: cookies

- server sends "cookie" to client in response
  `Set-cookie: #`
- client present cookie in later requests
  `cookie: #`
- server matches presented-cookie with server-stored cookies
  - authentication
  - remembering user preferences, previous choices



User-server interaction: conditional GET

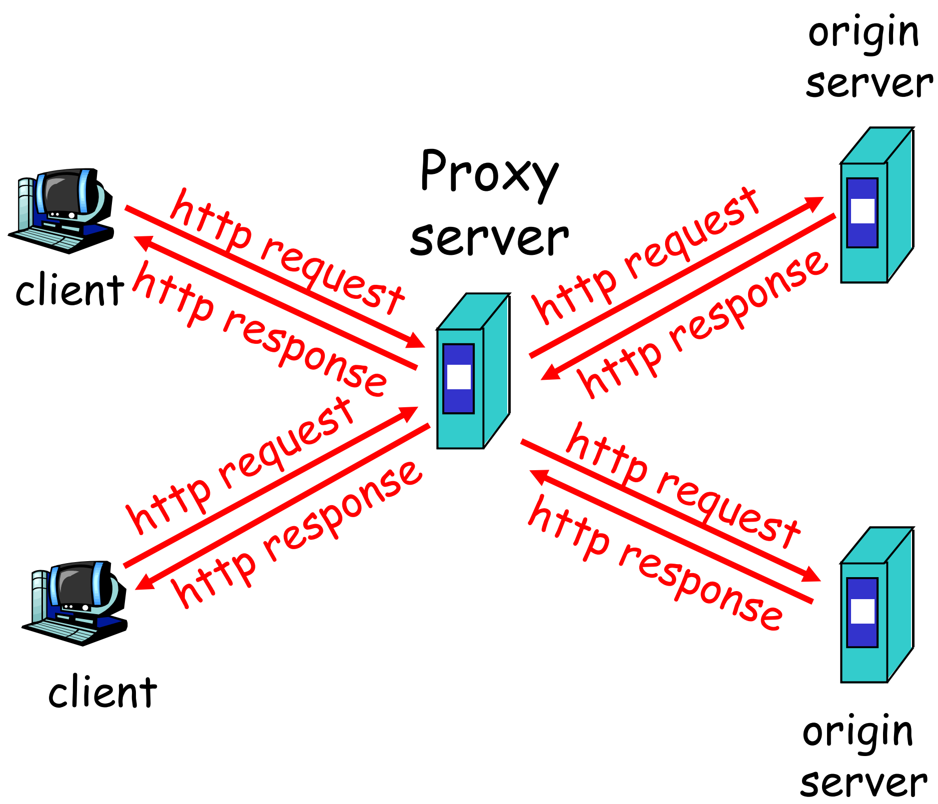
- **Goal:** don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request
  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:
  `HTTP/1.0 304 Not Modified`



Web Caches (proxy server)

Goal: satisfy client request without involving origin server

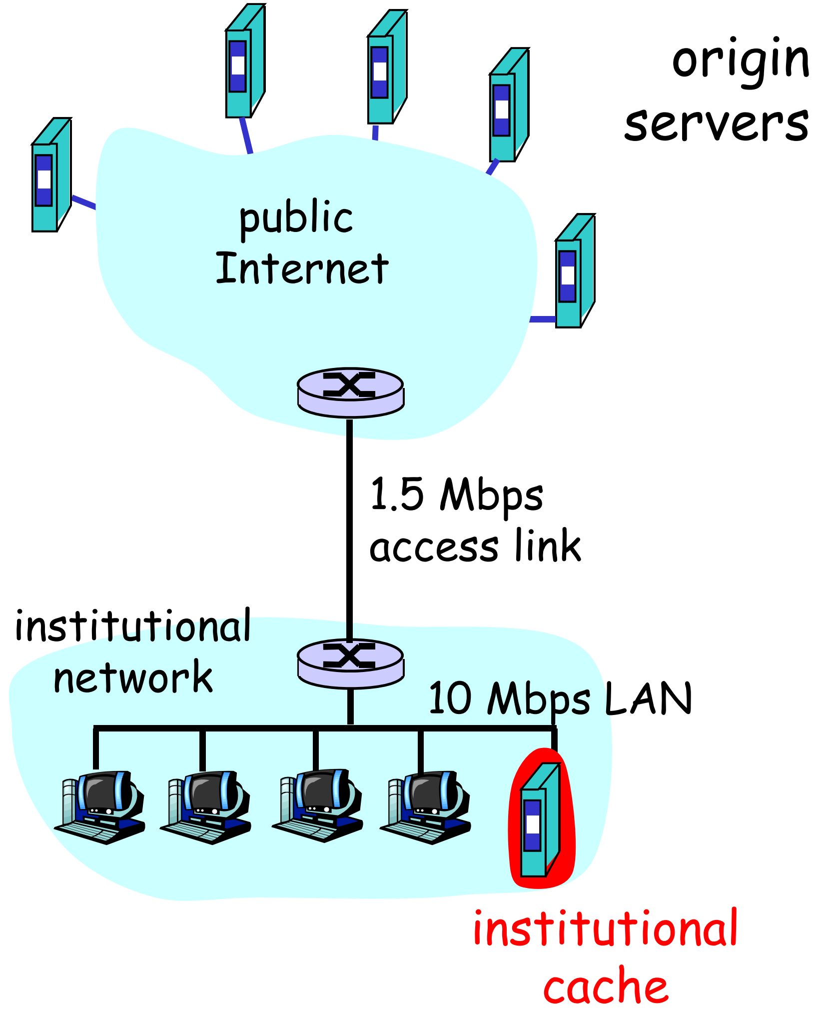
- user sets browser: WWW accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



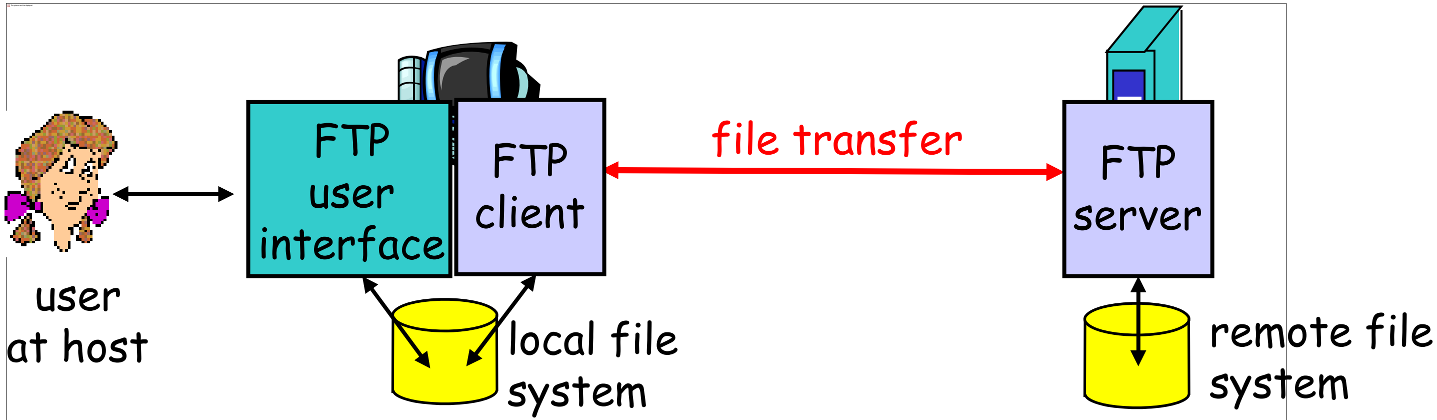
Why WWW Caching?

Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck



ftp: the file transfer protocol

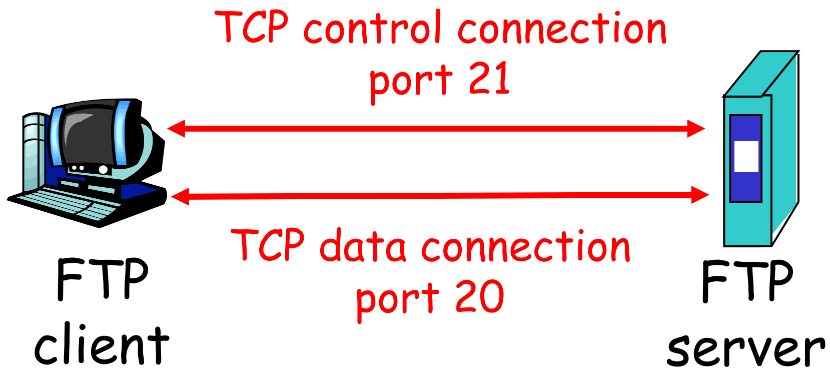


- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

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



ftp commands, responses

Sample commands:

- sent as ASCII text over control channel
- **USER** ***username***
- **PASS** ***password***
- **LIST** return list of files in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

Sample return codes

- status code and phrase (as in http)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

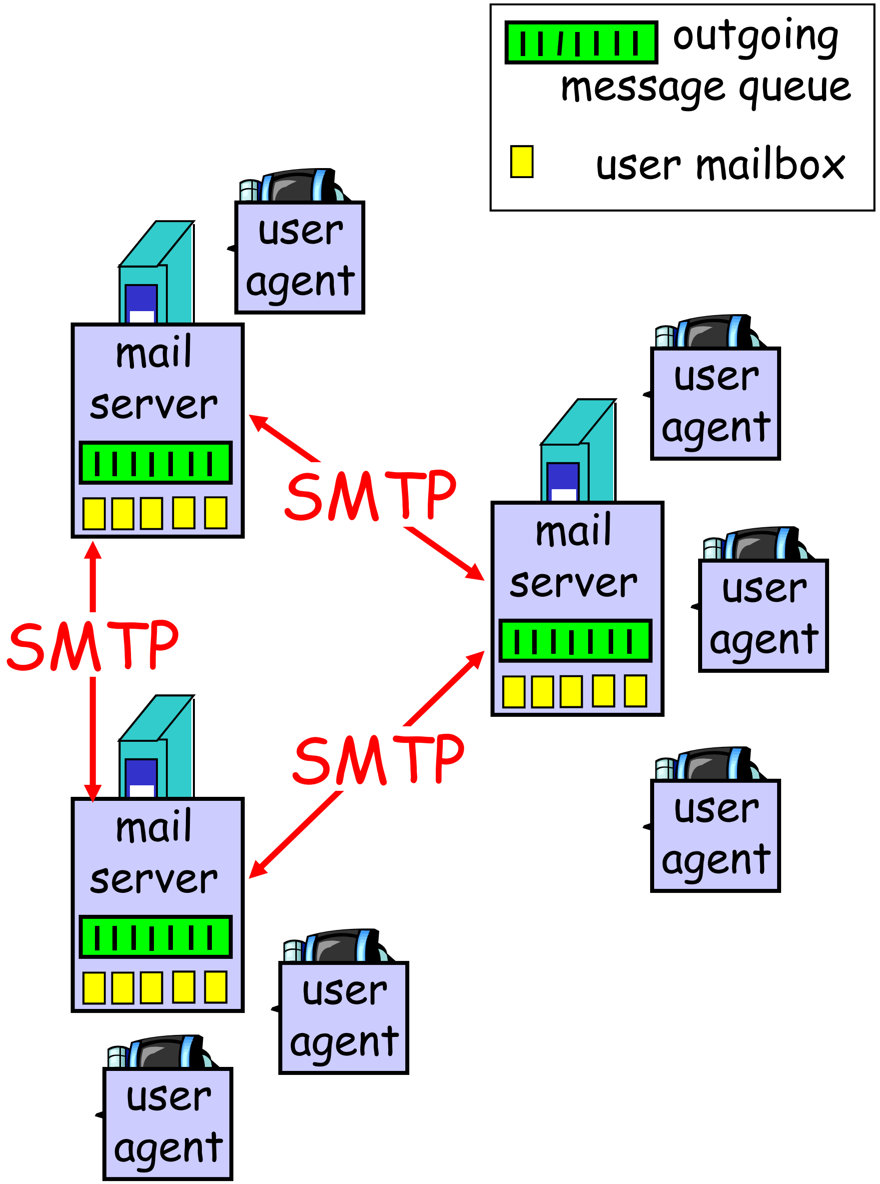
Electronic Mail

Three major components:

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

User Agent

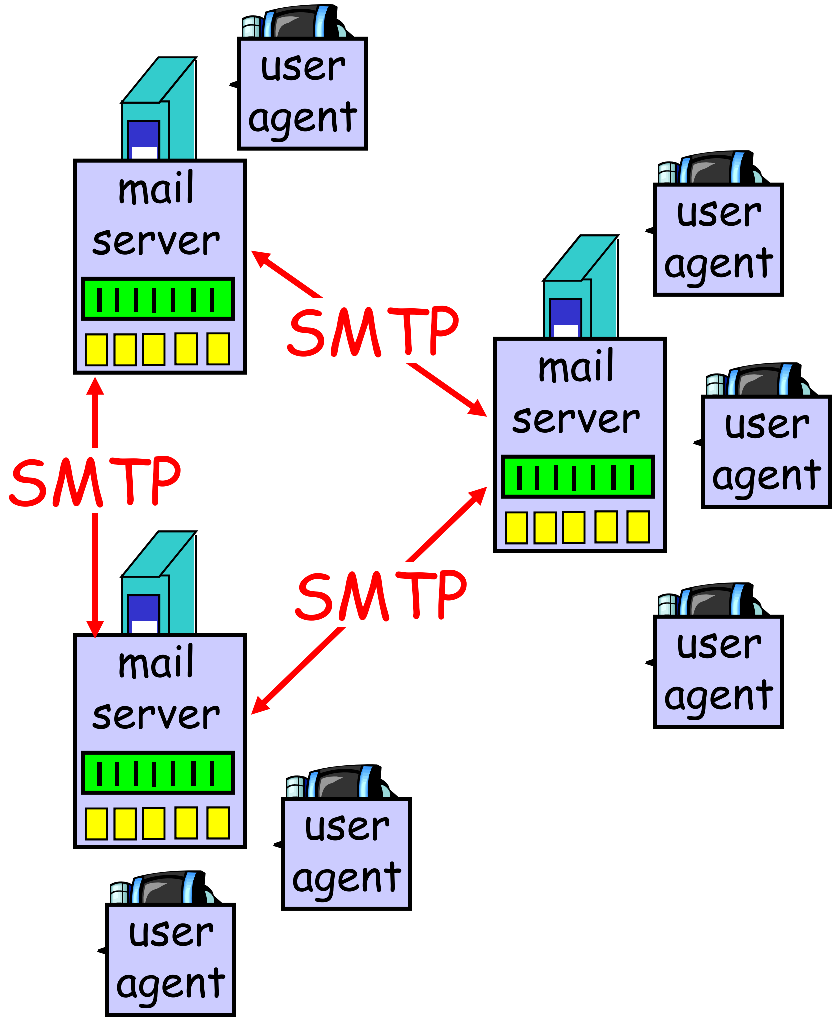
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, pine, elm, Netscape Messenger
- outgoing, incoming messages stored on server



Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages (yet to be read) for user
- message queue of outgoing (to be sent) mail messages
- smtp protocol between mail server to send email messages
  - client: sending mail server
  - "server": receiving mail server



Electronic Mail: smtp [RFC 821]

- uses tcp to reliably transfer email msg from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer
  - closure
- command/response interaction
  - commands: ASCI text
  - response: status code and phrase

Sample smtp interaction

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

smtp: final words

try smtp interaction for yourself:

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

Comparison with http

- http: pull
- email: push
- both have ASCII command/response interaction, status codes
- http: each object encapsulated in its own response (if v.1.0 or so specified in 1.1)
- smtp: multiple message parts sent in one connection (multipart mess)

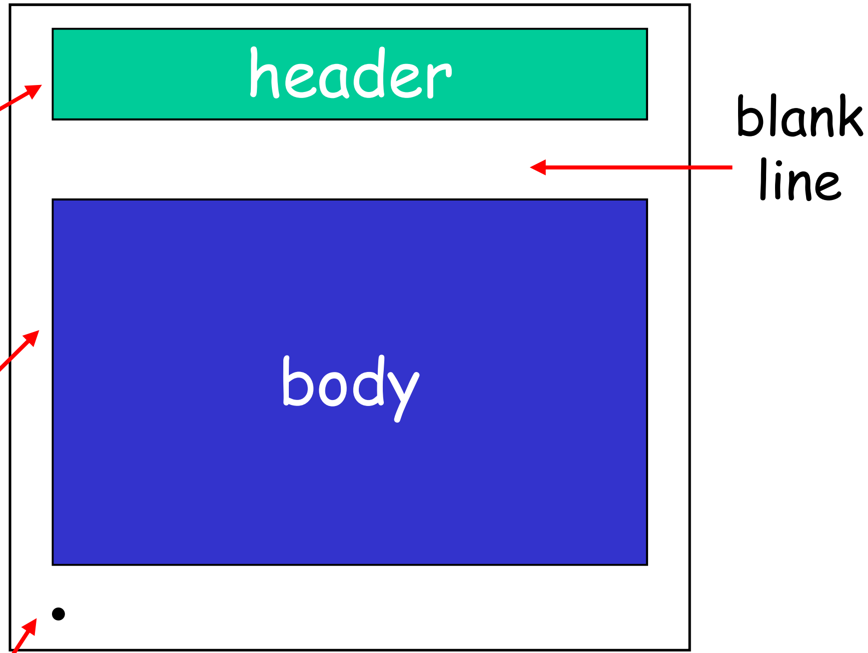
Mail message format

smtp: protocol for exchanging email msgs

RFC 822: standard for text message format:

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

  different from smtp commands!
- body
  - the “message”, ASCII characters only
- line containing only `.'



Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

MIME version

method used to encode data

multimedia data type, subtype, parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
.
```

MIME types

Content-Type: type/subtype; parameters

Text

- example subtypes: **plain, html**

Image

- example subtypes: **jpeg, gif**

Audio

- example subtypes: **basic** (8-bit mu-law encoded), **32kadpcm (32 kbps coding)**

Video

- example subtypes: **mpeg, quicktime**

Application

- other data that must be processed by reader before "viewable"
- example subtypes: **msword, octet-stream**

Multipart Type

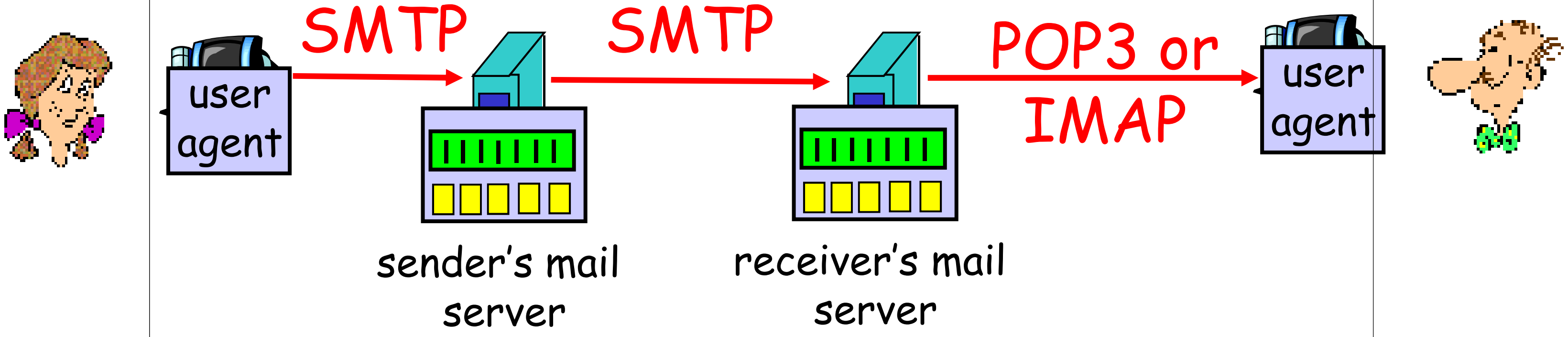
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```

Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server

POP3 protocol

authorization phase

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - **+OK**
  - **-ERR**

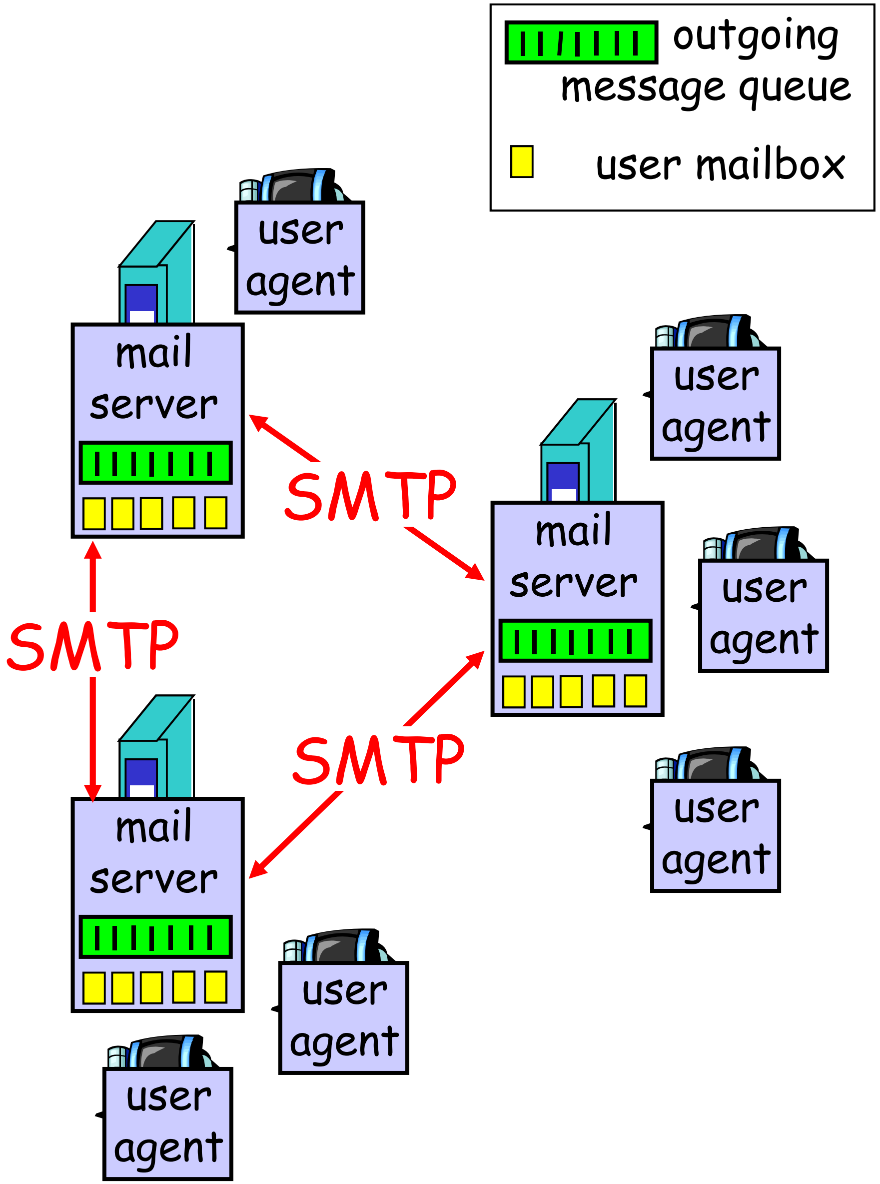
transaction phase, client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

Electronic Mail



Three major components:

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

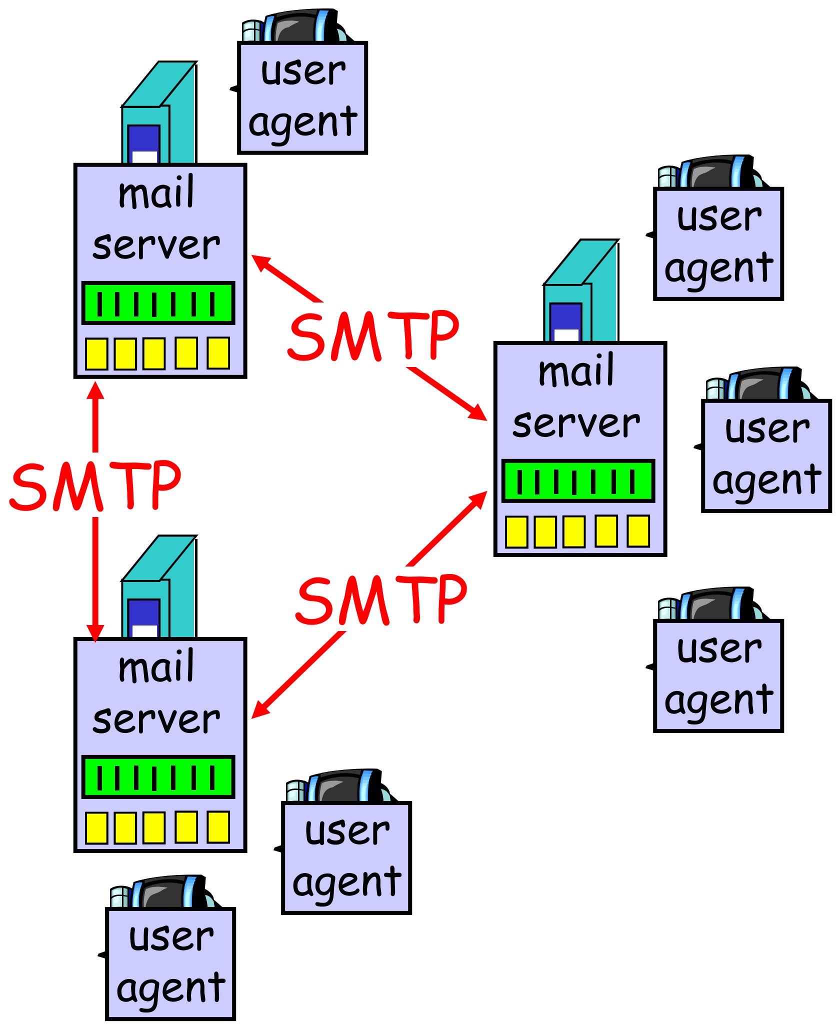
User Agent

- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Eudora, pine, elm, Netscape Messenger
- outgoing, incoming messages stored on server

Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages (yet ot be read) for user
- message queue of outgoing (to be sent) mail messages
- smtp protocol between mail server to send email messages
  - client: sending mail server
  - "server": receiving mail server



Electronic Mail: smtp [RFC 821]

- uses tcp to reliably transfer email msg from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer
  - closure
- command/response interaction
  - commands: ASCI text
  - response: status code and phrase

Sample smtp interaction

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

smtp: final words

try smtp interaction for yourself:

- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

Comparison with http

- http: pull
- email: push
- both have ASCII command/response interaction, status codes
- http: multiple objects in file sent in separate connections
- smtp: multiple message parts sent in one connection

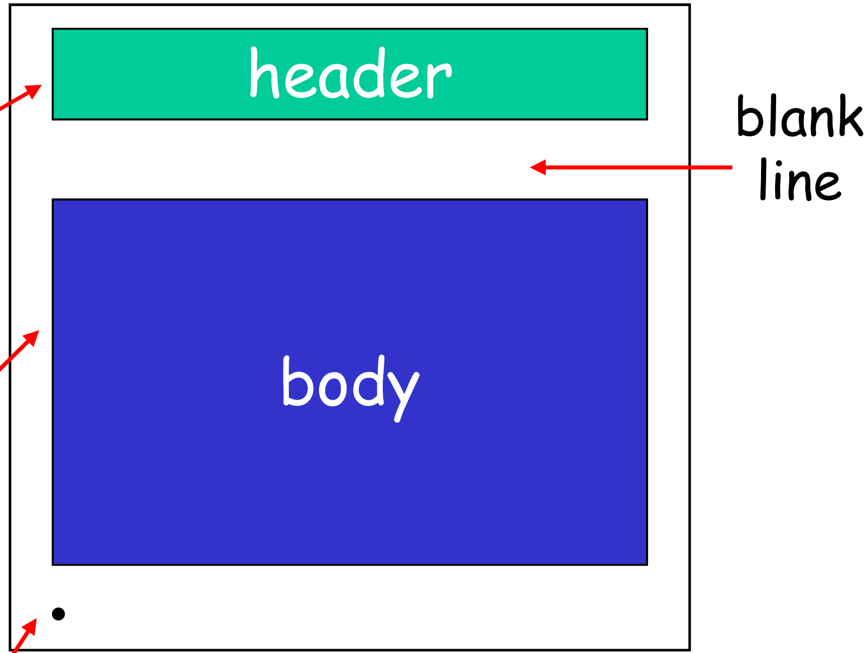
Mail message format

smtp: protocol for exchanging email msgs

RFC 822: standard for text message format:

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

  different from smtp commands!
- body
  - the "message", ASCII characters only
- line containing only `.'



Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type

MIME version

method used to encode data

multimedia data type, subtype, parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
.
```

MIME types

Text

- example subtypes: **plain, html**

Image

- example subtypes: **jpeg, gif**

Audio

- exampe subtypes: **basic** (8-bit mu-law encoded), **32kadpcm (32 kbps coding)**

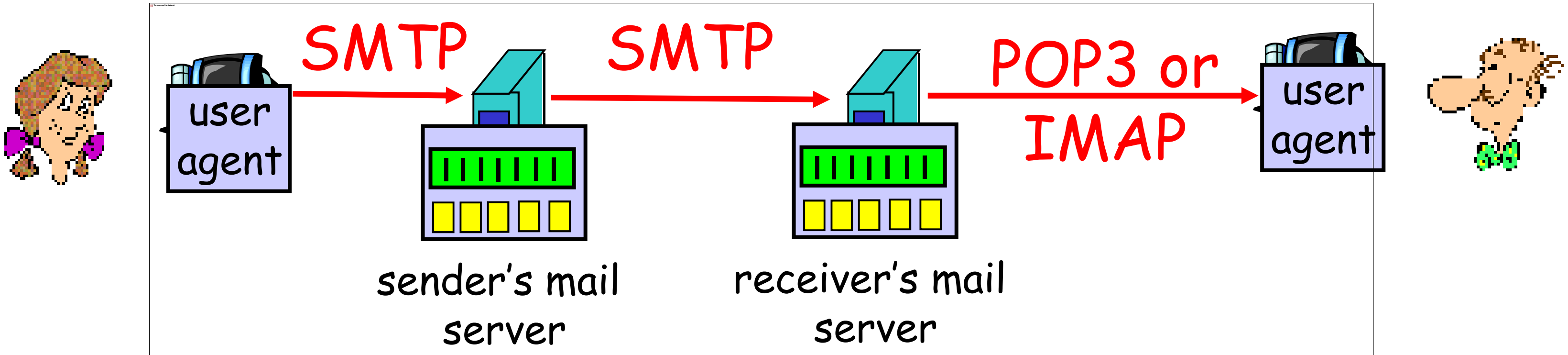
Video

- example subtypes: **mpeg, quicktime**

Application

- other data that must be processed by reader before "viewable"
- example subtypes: **msword, octet-stream**

Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server

POP3 protocol

authorization phase

- client commands:
  - **user**: declare username
  - **pass**: password
- server responses
  - **+OK**
  - **-ERR**

transaction phase, client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

DNS: Domain Name System

People: many identifiers:
- SSN, name, Passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., hermite.cs.smith.edu - used by humans

Q: map between IP addresses and name ?

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"

DNS name servers

Why not centralize DNS?

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

doesn't *scale!*

- no server has all name-to-IP address mappings

local name servers:

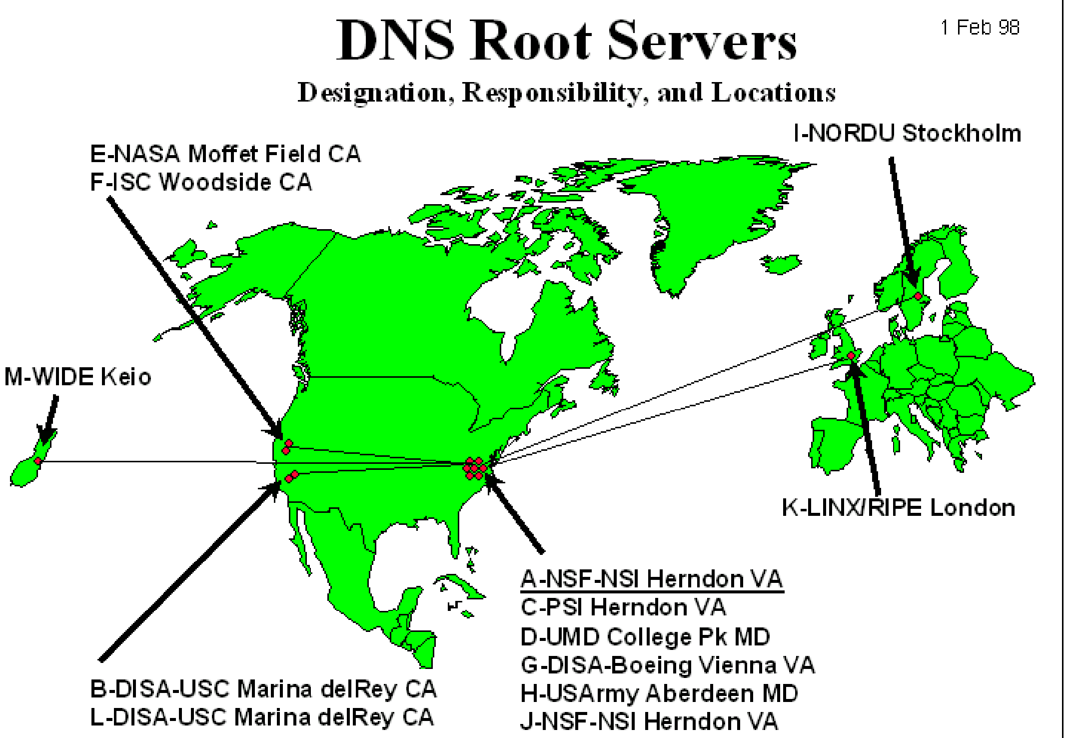
- each ISP, company has *local (default) name server*
- host DNS query first goes to local name server

authoritative name server:

- for a host: stores that host's IP address, name
- can perform name/address translation for that host's name

DNS: Root name servers

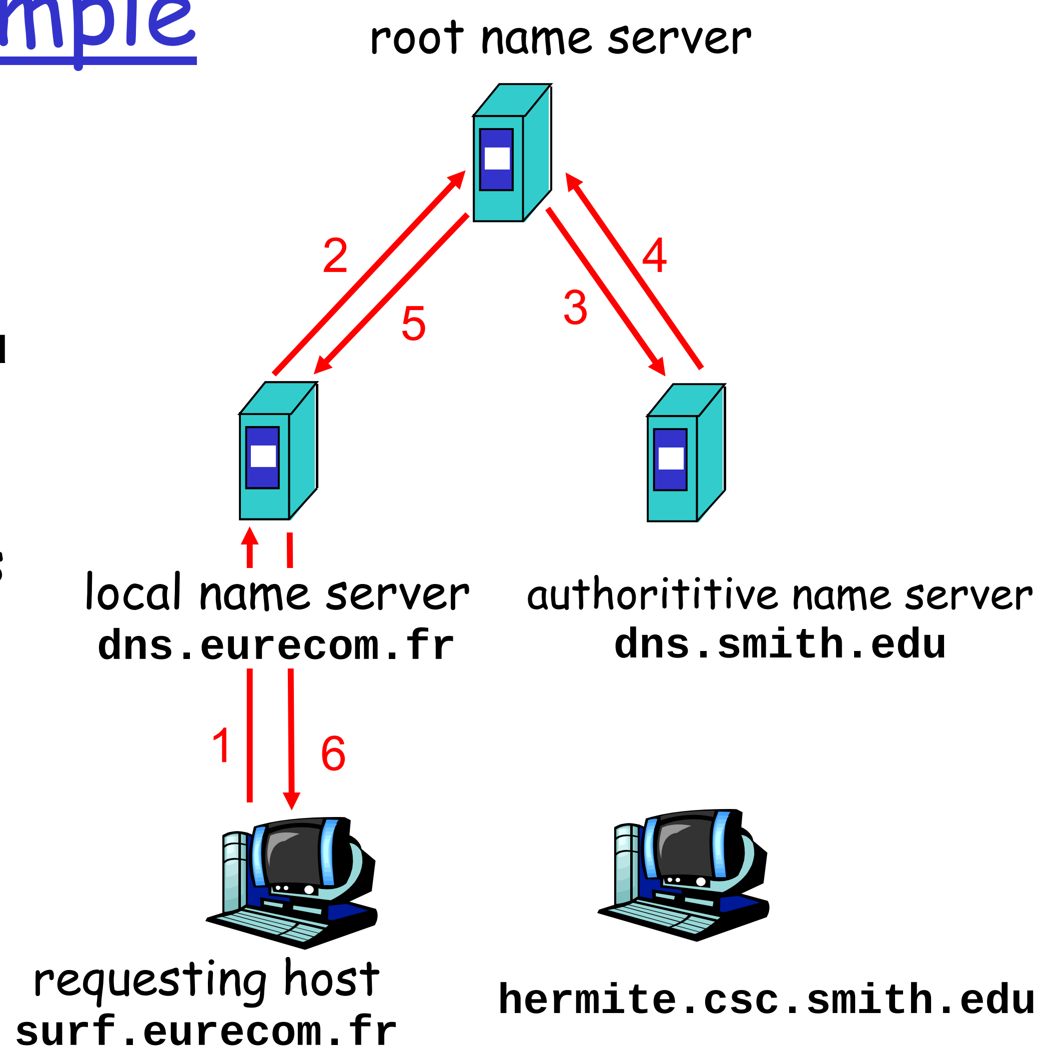
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server
- ~ dozen root name servers worldwide



Simple DNS example

host **surf.eurecom.fr** wants IP address of **hermite.csc.smith.edu**

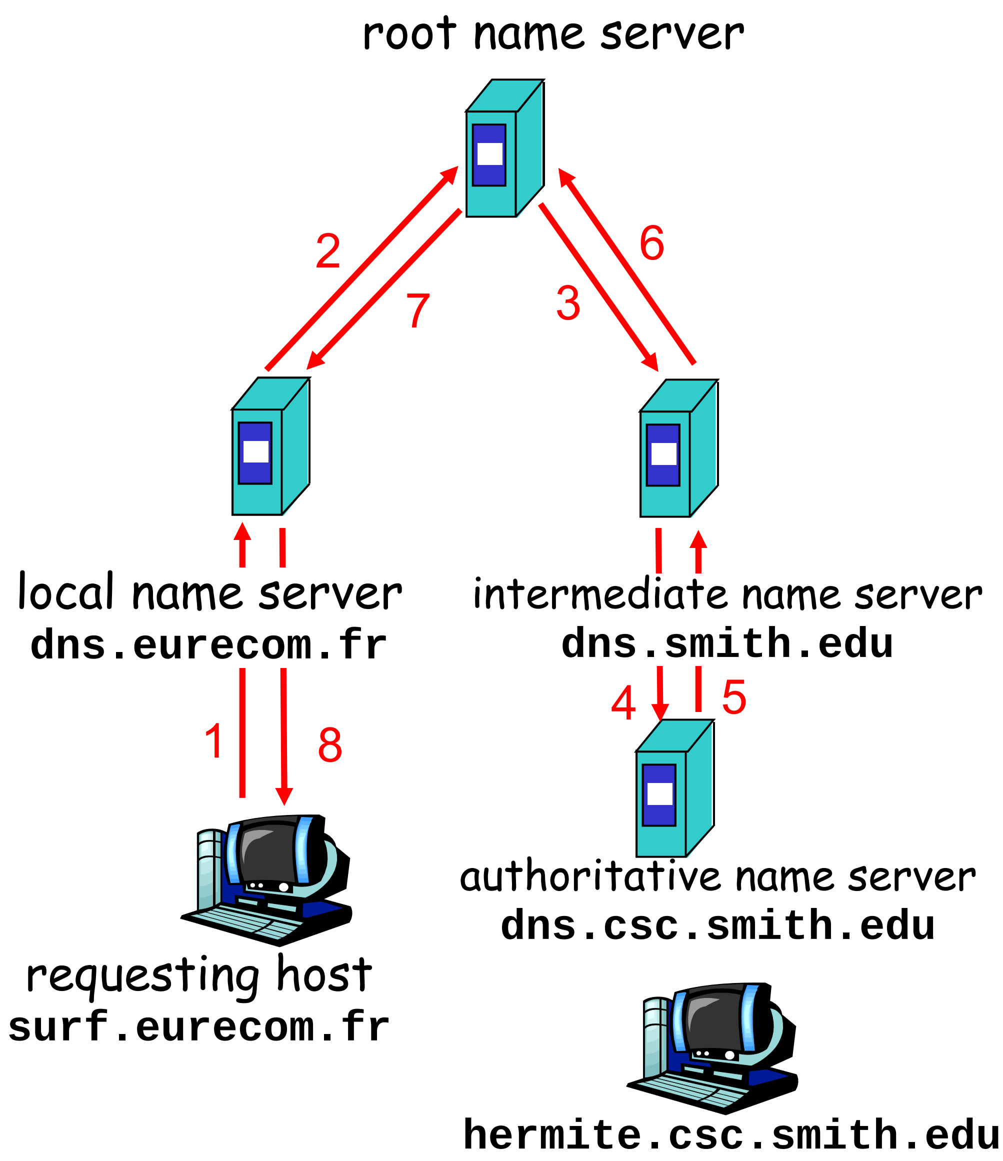
1. Contacts its local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** contacts root name server, if necessary
3. root name server contacts authoritative name server, **dns.umass.edu**, if necessary



DNS example

Root name server:

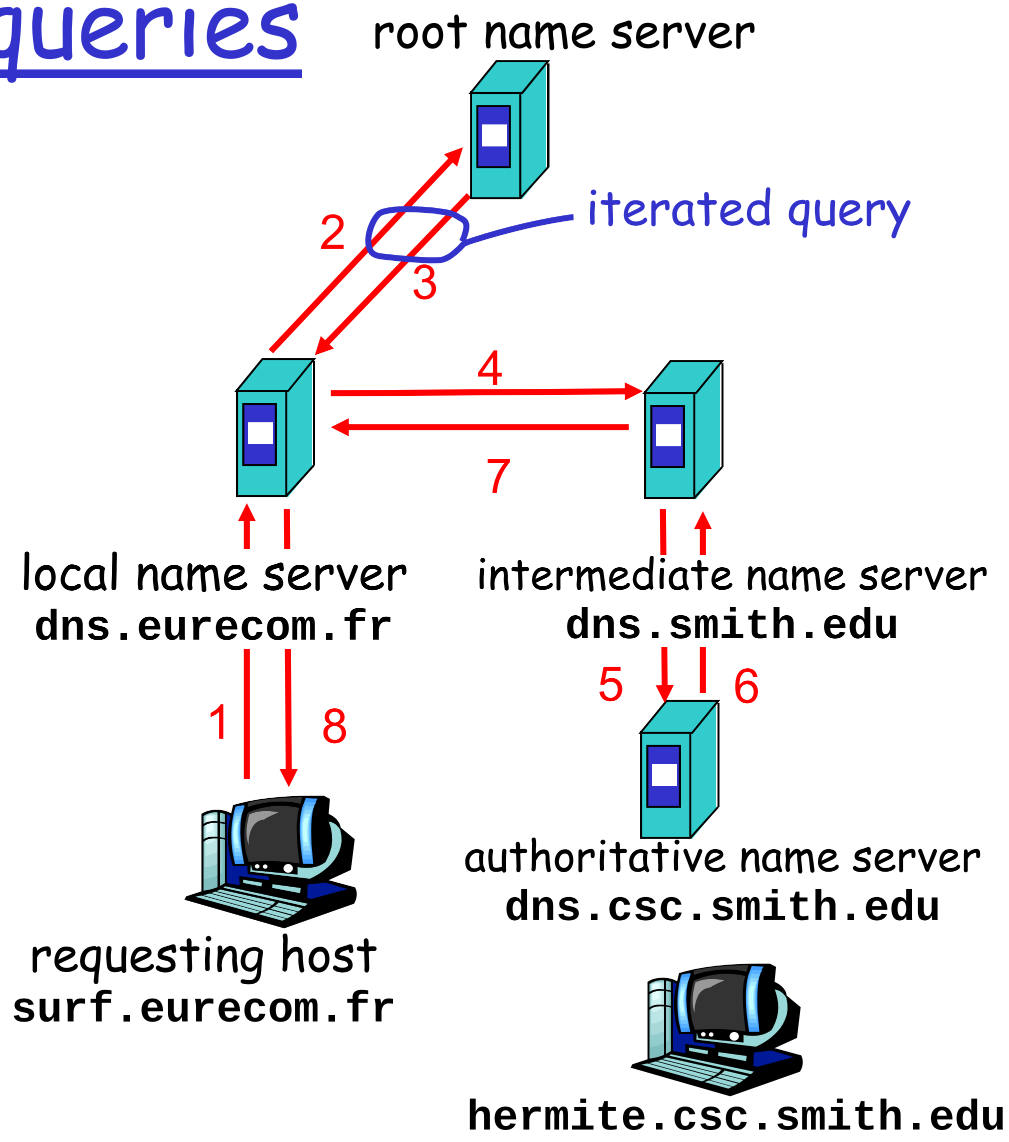
- may not know authoratiative name server
- may know *intermediate name server:* who to contact to find authoritative name server



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)`

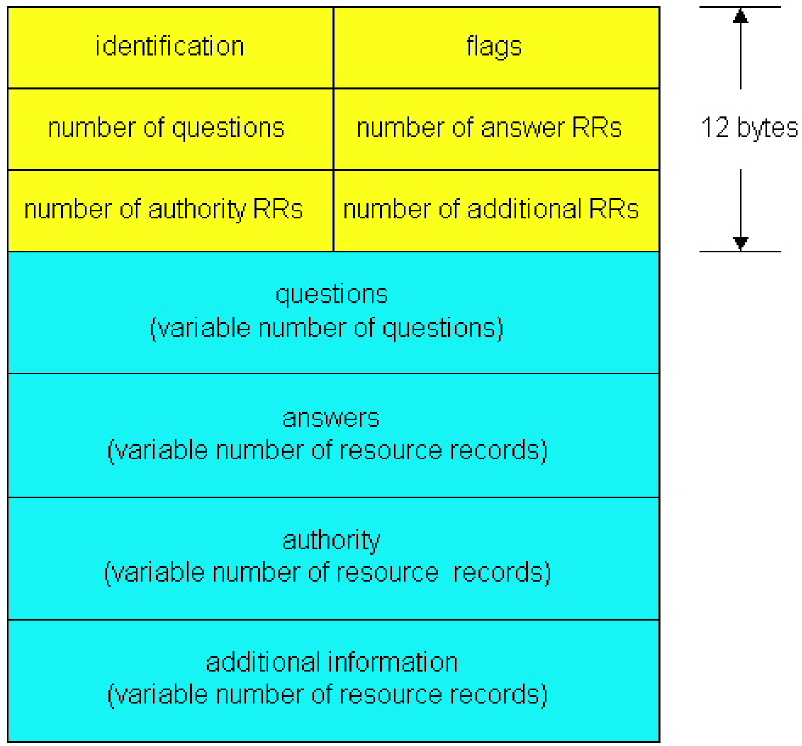
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is IP address of authoritative name server for this domain
- Type=CNAME
  - **name** is an alias name for some "canonical" (the real) name
  - **value** is canonical name
- Type=MX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

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

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



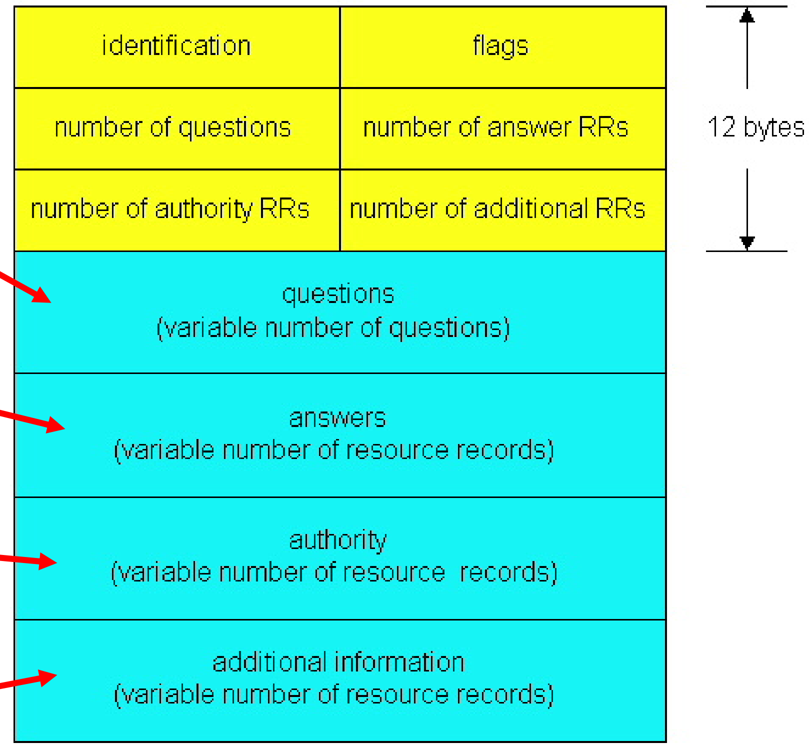
DNS protocol, messages

Name, type fields
for a query

RRs in reponse
to query

records for
authoritative servers

additional "helpful"
info that may be used



Socket programming

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

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

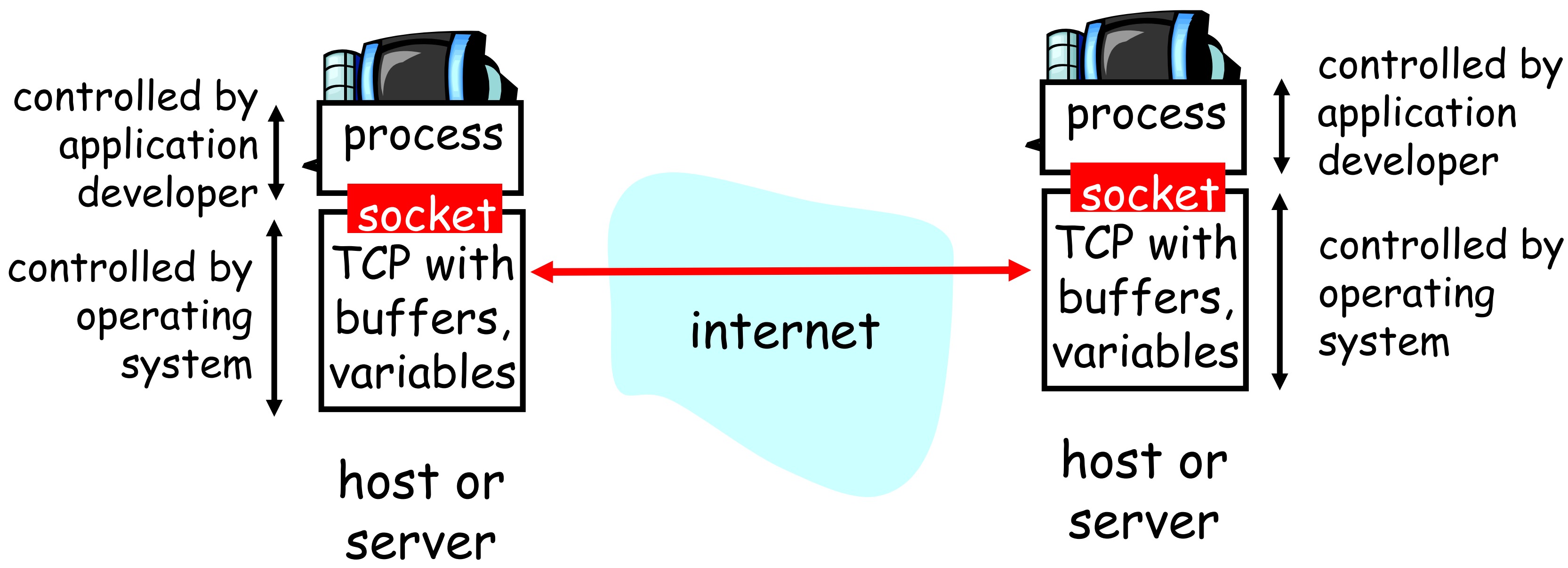
socket

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 (UDP or TCP)

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



Socket programming with TCP

Client must contact server

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

Client contacts server by:

- creating client-local TCP socket
- specifying IP address, port number of server process

- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

application viewpoint

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

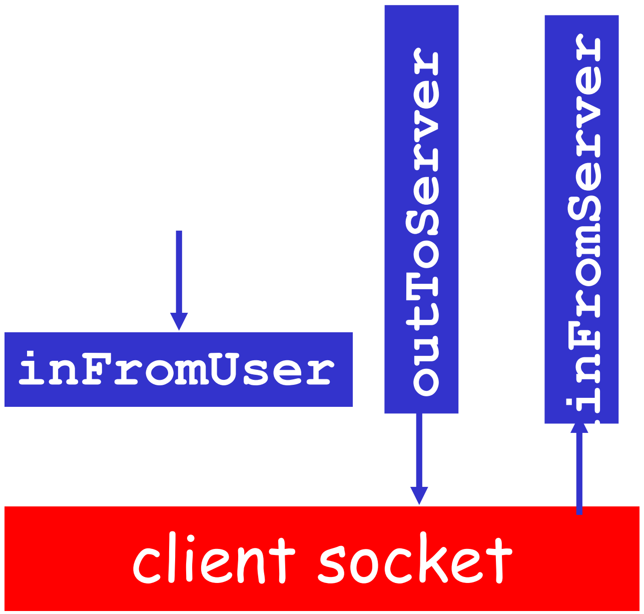
Socket programming with TCP

Example client-server app:

- client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**`inFromServer`** stream)

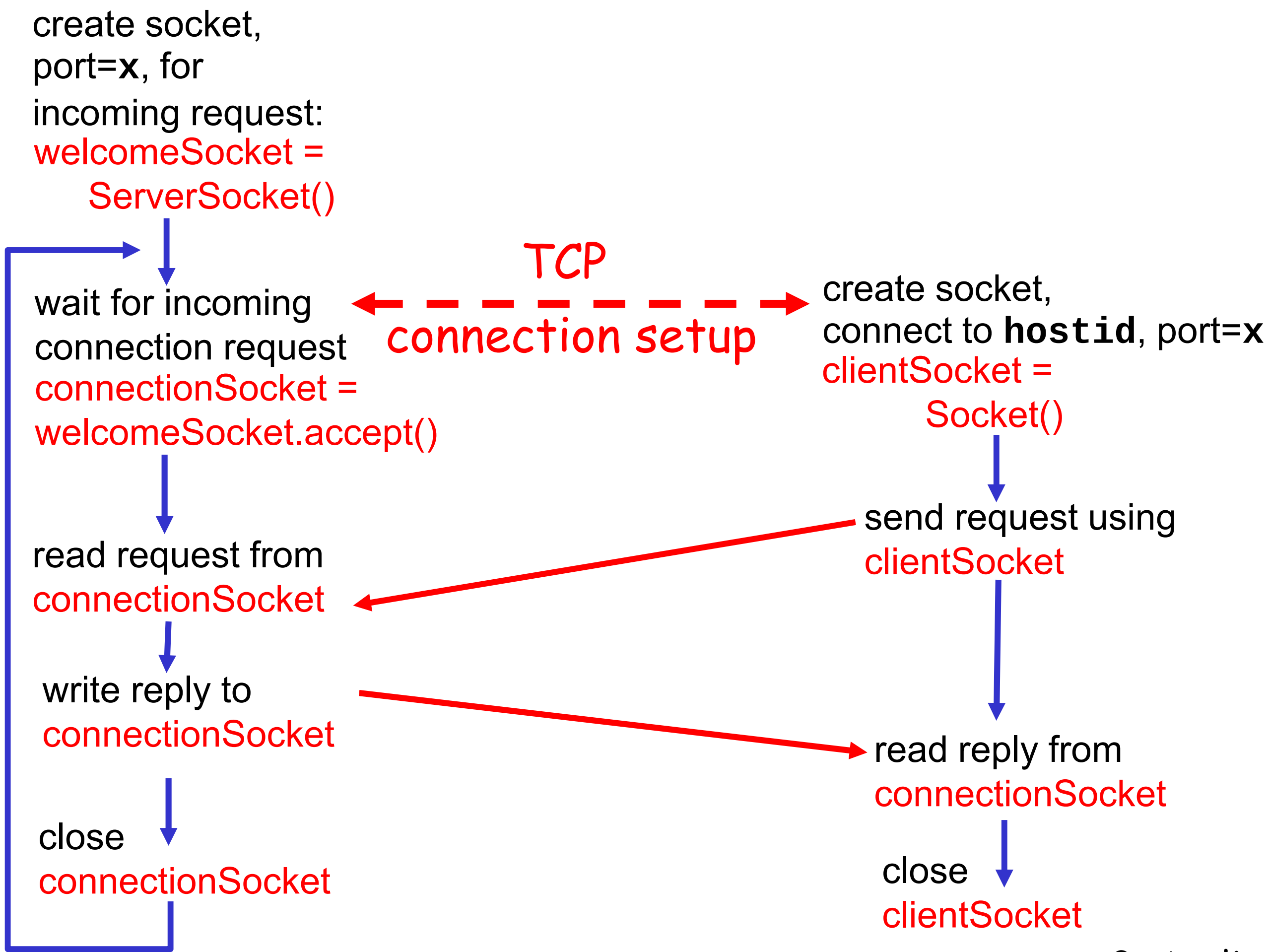
Input stream: sequence of bytes into process

Output stream: sequence of bytes out of process



Client/server socket interaction: TCP

Server (running on **hostid**) | **Client**



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream → BufferedReader inFromUser

Create client socket, connect to server → Socket clientSocket

Create output stream attached to socket → DataOutputStream outToServer

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));
```

```
sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Example: Java server (TCP)

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

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

class TCPServer {

 public static void main(String argv[]) throws Exception
  {
   String clientSentence;
   String capitalizedSentence;

   ServerSocket welcomeSocket = new ServerSocket(6789);

   while(true) {

      Socket connectionSocket = welcomeSocket.accept();

     BufferedReader inFromClient =
       new BufferedReader(new
       InputStreamReader(connectionSocket.getInputStream()));
```

Example: Java server (TCP), cont

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

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

Socket programming with UDP

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination
- server must extract IP address, port of sender from received datagram

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

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP

Server (running on **hostid**)

create socket,
port=**x**, for
incoming request:
serverSocket =
DatagramSocket()

read request from
serverSocket

write reply to
serverSocket
specifying client
host address,
port number

Client

create socket,
clientSocket =
DatagramSocket()

Create, address (**hostid, port=x**),
send datagram request
using clientSocket

read reply from
clientSocket

close
clientSocket

Example: Java client (UDP)

Create input stream

Create client socket

Translate hostname to IP address using DNS

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

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
  new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
   new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```

Example: Java server (UDP)

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

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

class UDPServer {
  public static void main(String args[]) throws Exception
   {

     DatagramSocket serverSocket = new DatagramSocket(9876);

     byte[] receiveData = new byte[1024];
     byte[] sendData  = new byte[1024];

     while(true)
      {

        DatagramPacket receivePacket =
           new DatagramPacket(receiveData, receiveData.length);

         serverSocket.receive(receivePacket);
```

Example: Java server (UDP), cont

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

        String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress,
                port);

serverSocket.send(sendPacket);
    }
  }
}
```

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

Summary on Application Layer

Our study of network apps now complete!

- application service requirements:
  - reliability, bandwidth, delay
- client-server paradigm
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - http
  - ftp
  - smtp, pop3
  - dns
- socket programming
  - client/server implementation
  - using tcp, udp sockets

Summary on Application Layer

Most importantly: learned about *protocols*

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated

- control vs. data msgs
  - in-based, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"
- security: authentication

Transport Layer

Goals:

- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation and implementation in the Internet

Overview:

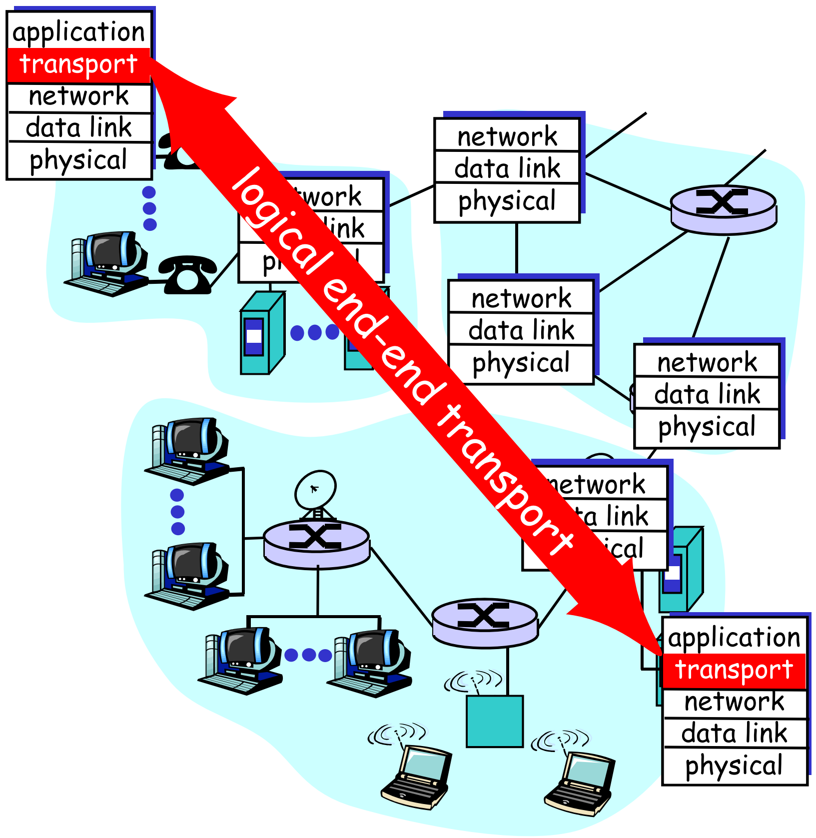
- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
  - flow control
  - connection management
- principles of congestion control
- TCP congestion control

Transport services and protocols

- provide *logical communication* between app' processes running on different hosts
- transport protocols run in end systems (primarily)

transport vs network layer services:

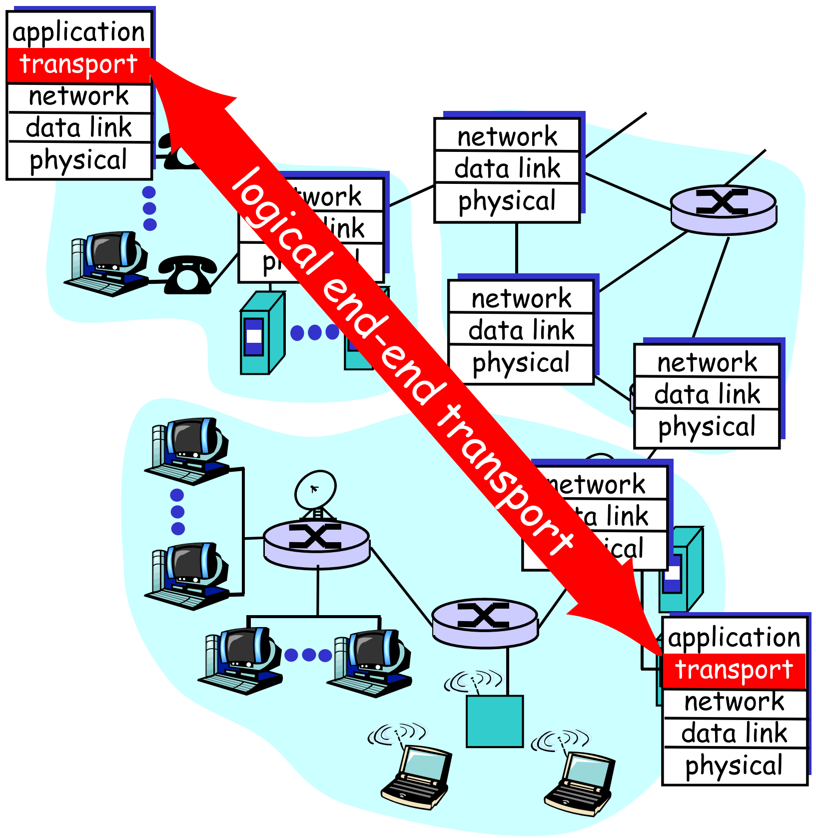
- *network layer:* data transfer between end systems
- *transport layer:* data transfer between processes
  - relies on, enhances, network layer services



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

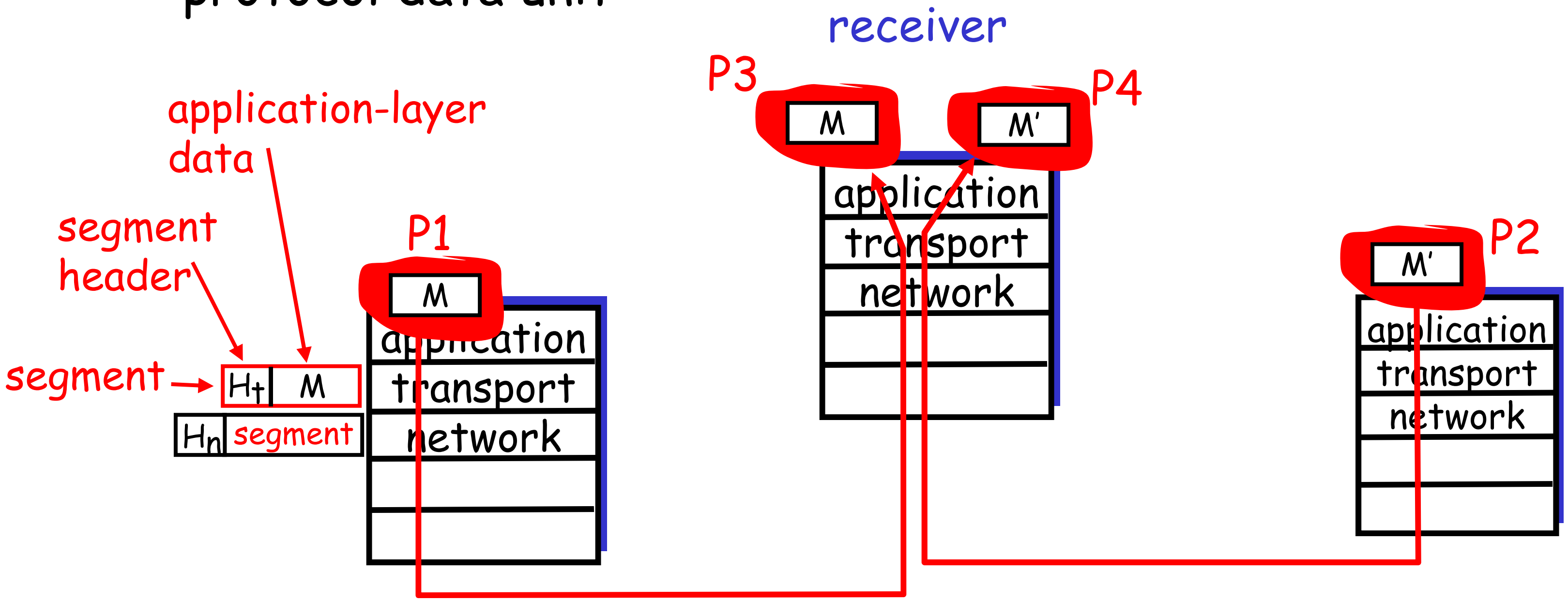


Multiplexing/demultiplexing

Recall: *segment* - unit of data exchanged between transport layer entities

- aka TPDU: transport protocol data unit

Demultiplexing: delivering received segments (TPDUs) to correct app layer processes

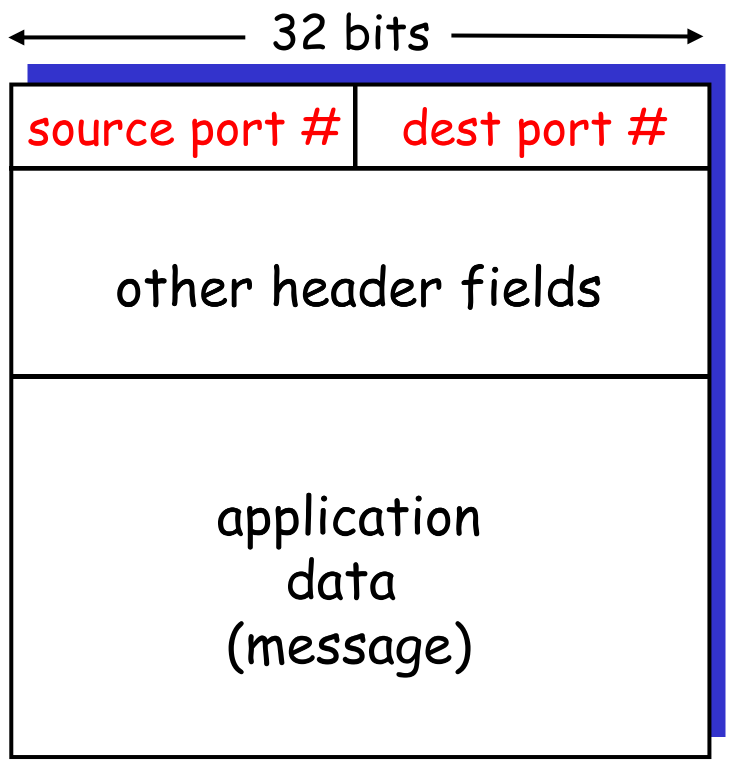


Multiplexing/demultiplexing

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

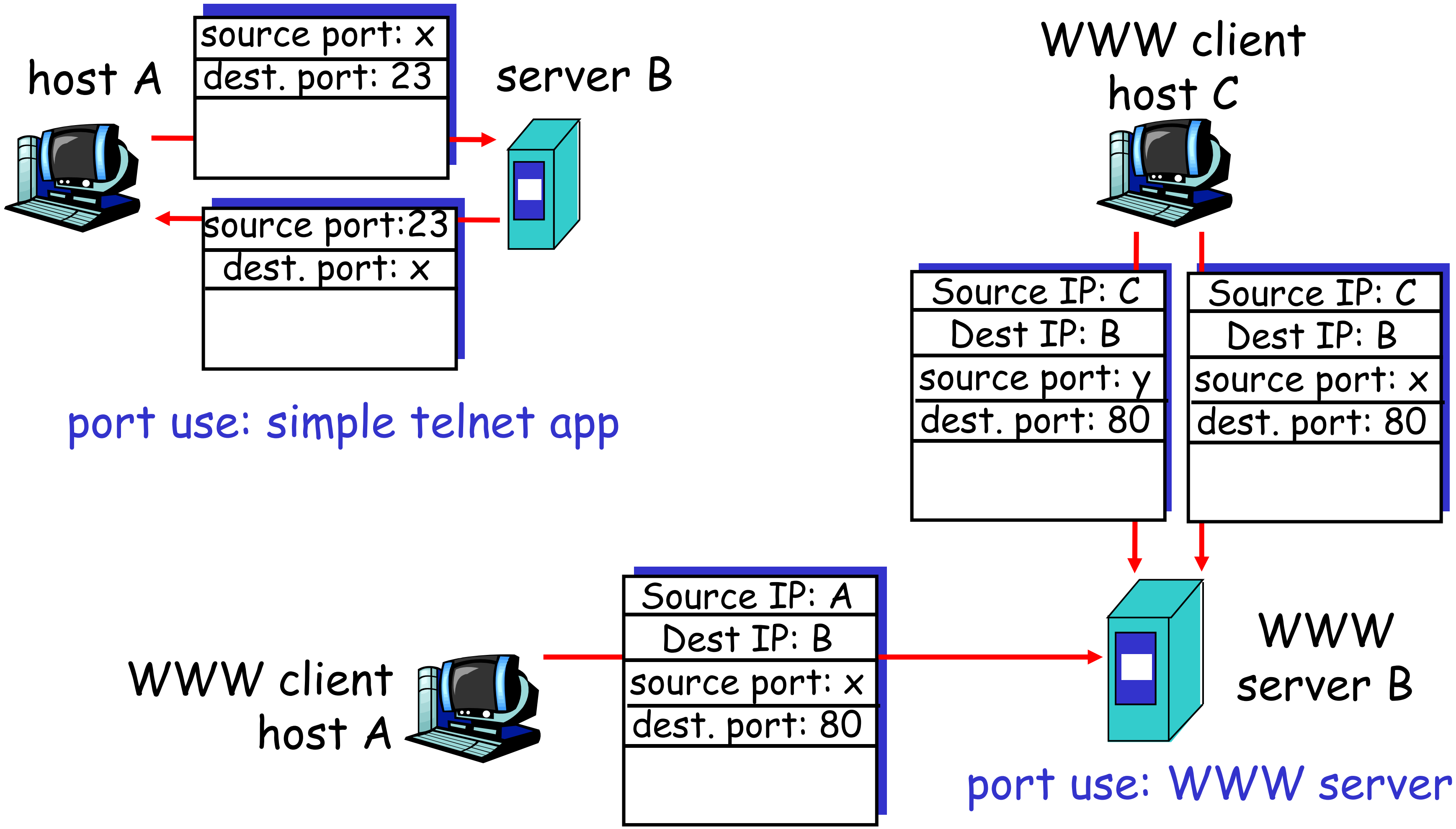
multiplexing/demultiplexing:

- 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



UDP: User Datagram Protocol [RFC 768]

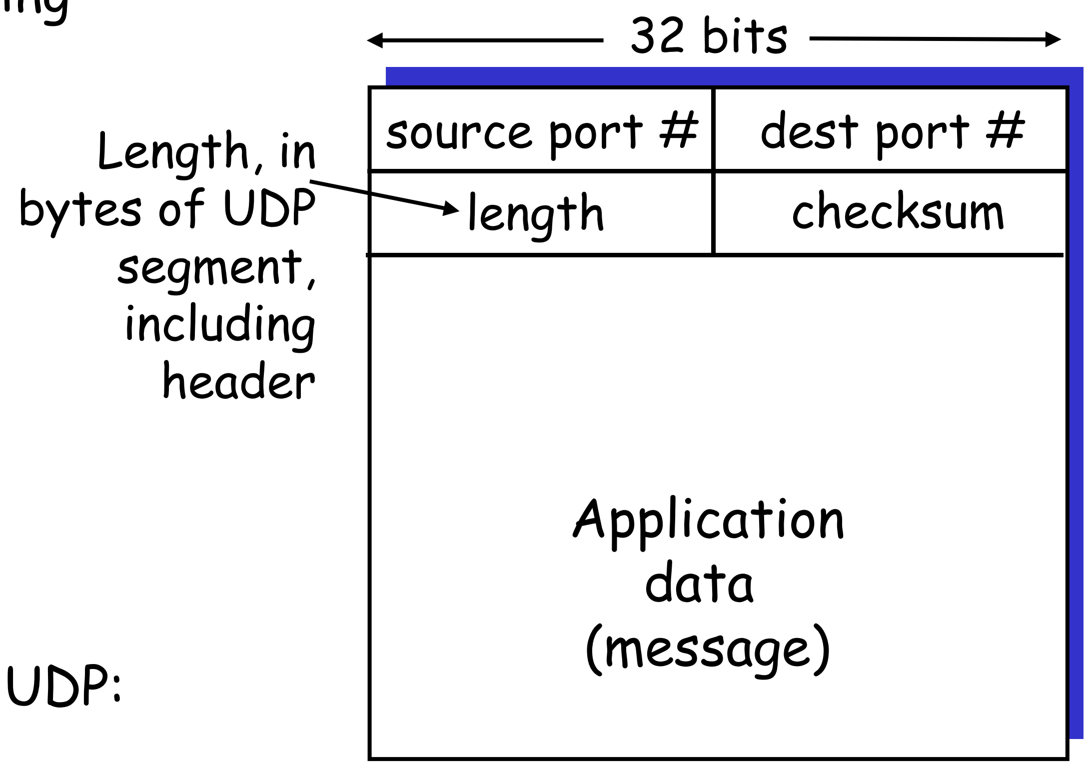
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

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: more

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



UDP segment format

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

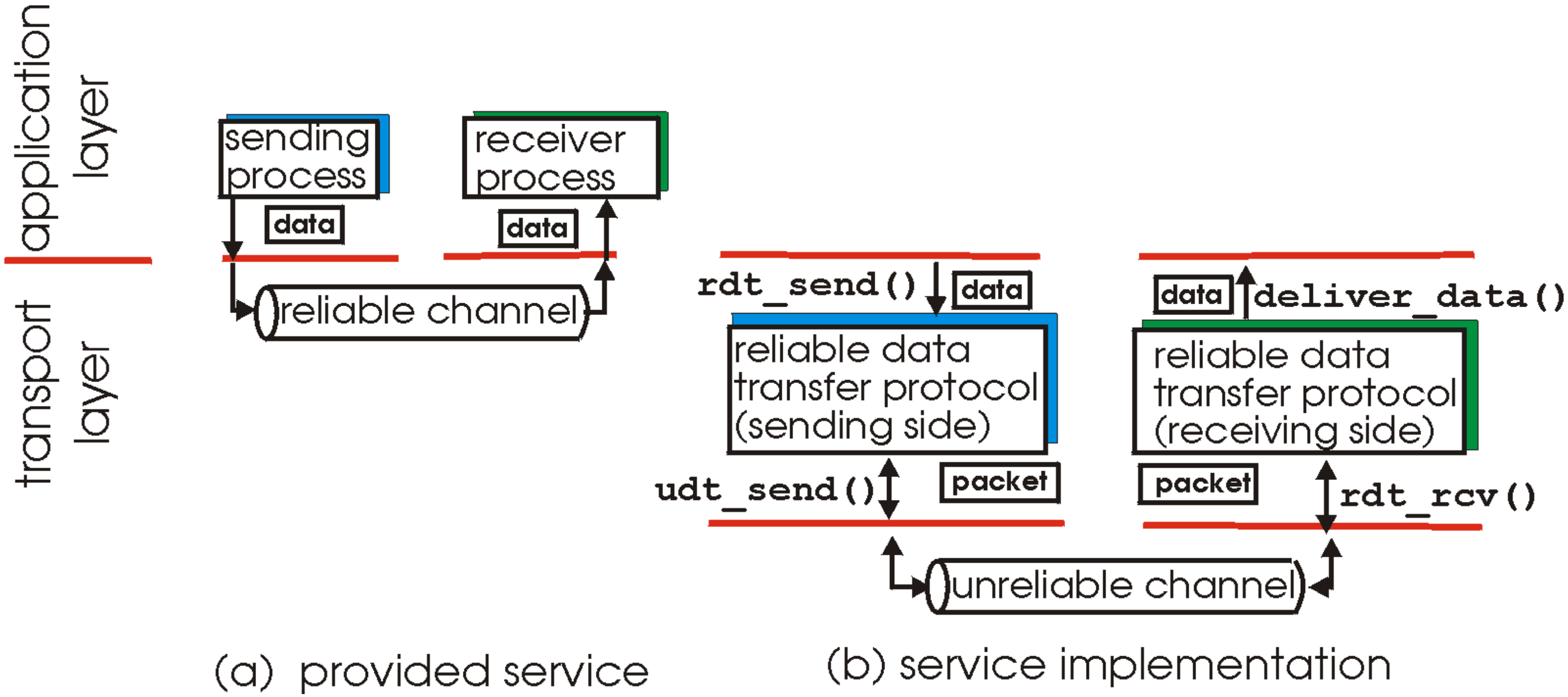
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonethless?* More later ....

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

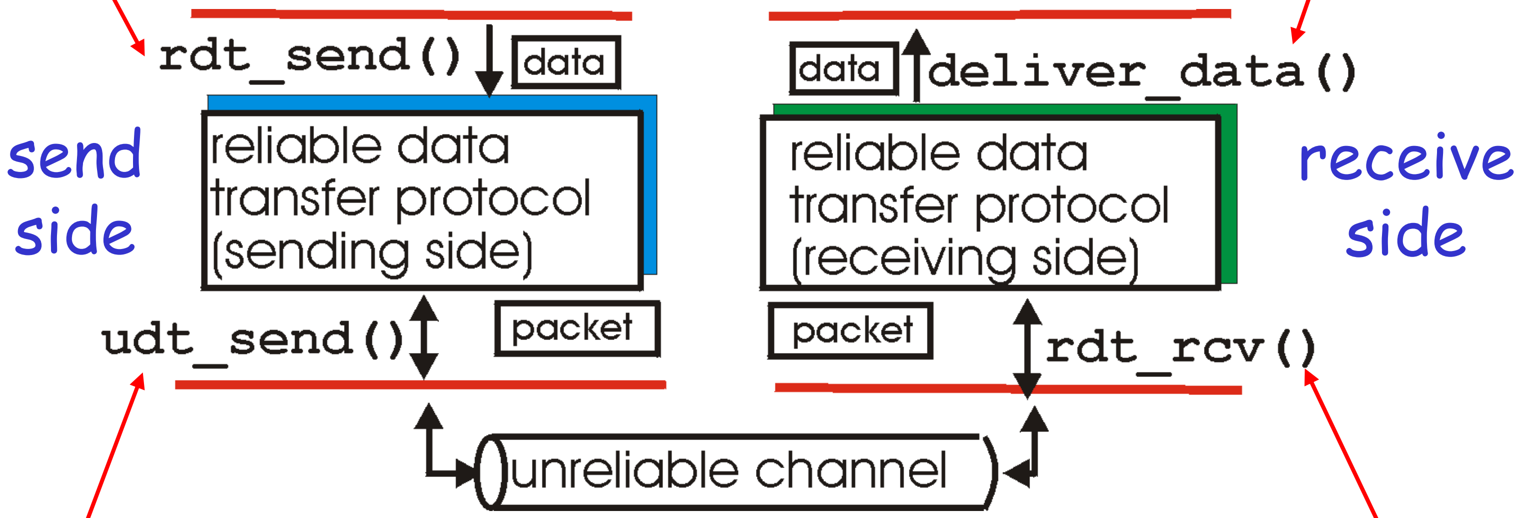


- characteristics of unreliable channel underneath it will determine complexity of reliable data transfer protocol (rdt)

Reliable data transfer: getting started

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

deliver_data(): called by **rdt** to deliver data to upper



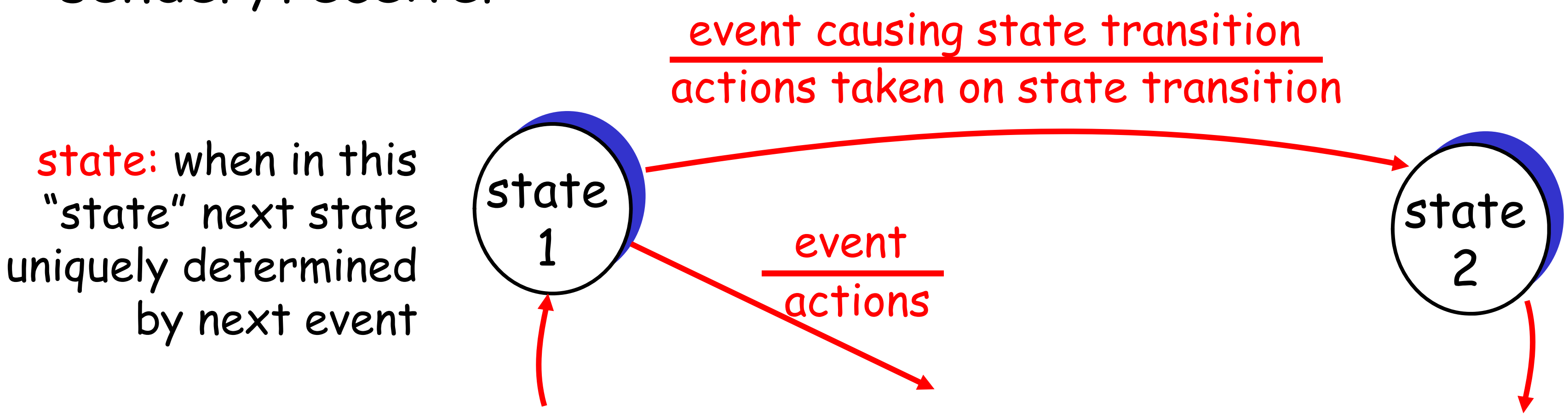
udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

rdt_rcv(): called when packet arrives on rcv-side of channel

Reliable data transfer: getting started

We'll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver



Rdt1.0: reliable transfer over a reliable channel

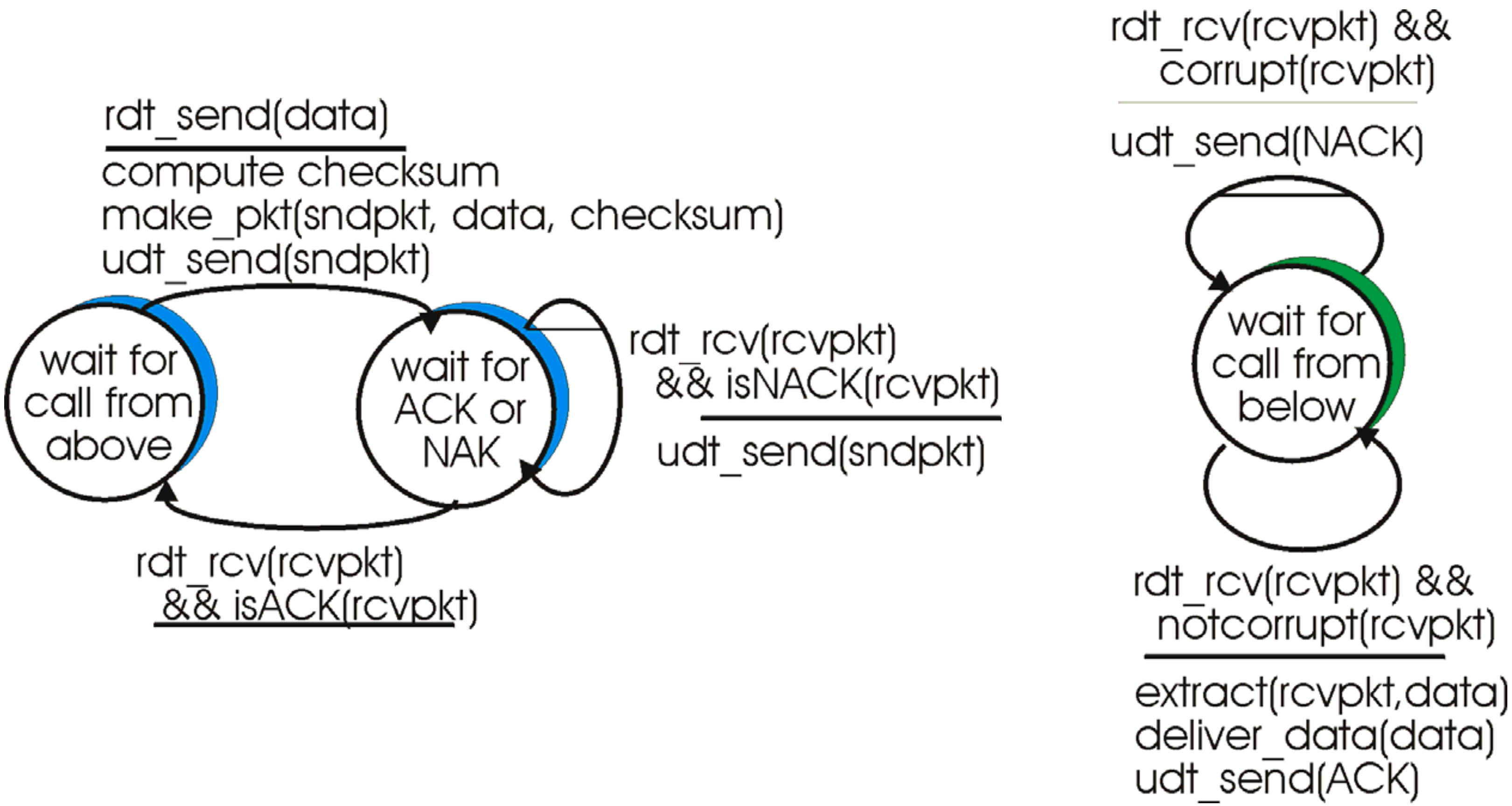
- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel

(a) rdt1.0: sending side (b) rdt1.0: receiving side

Rdt2.0: channel with bit errors

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

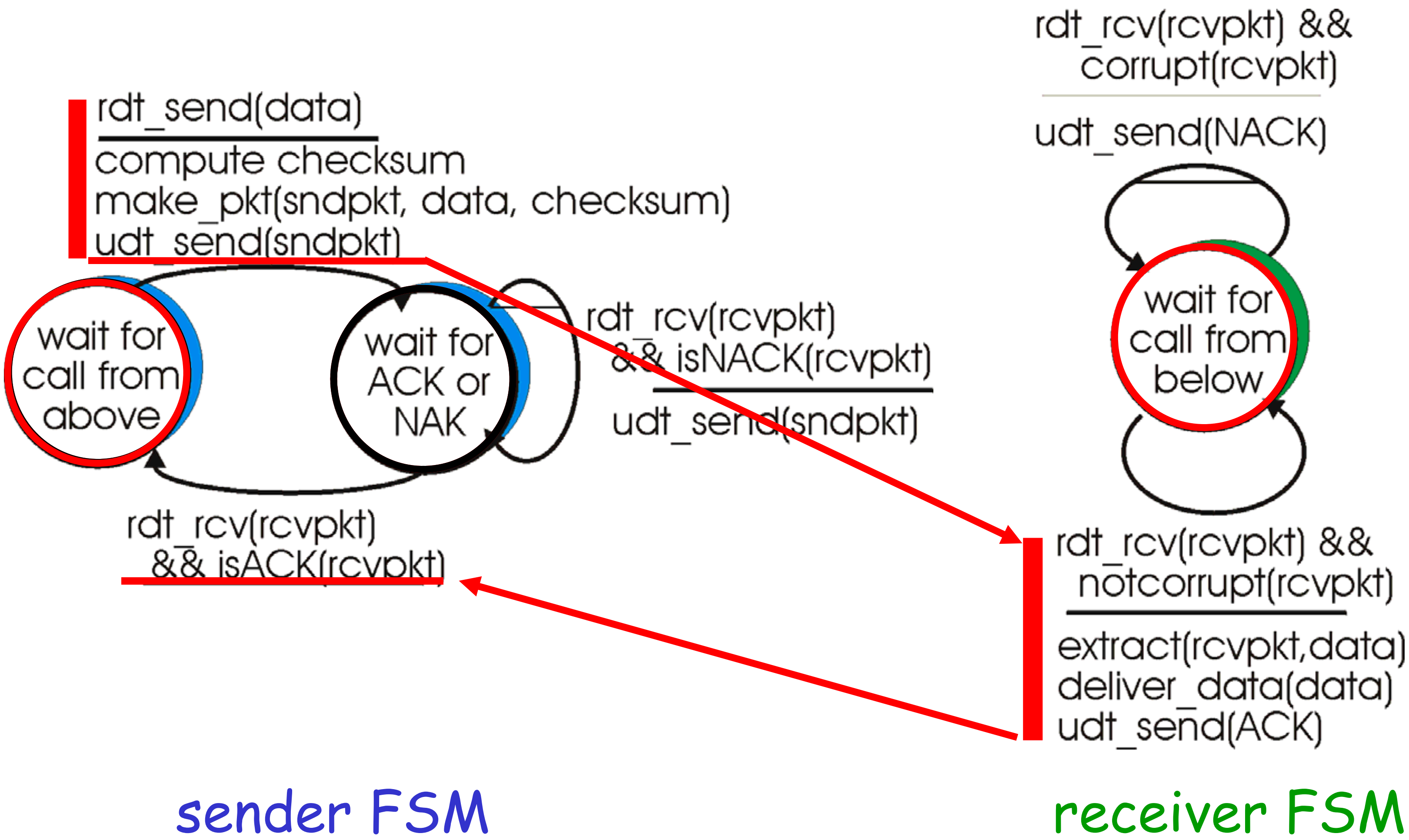
rdt2.0: FSM specification



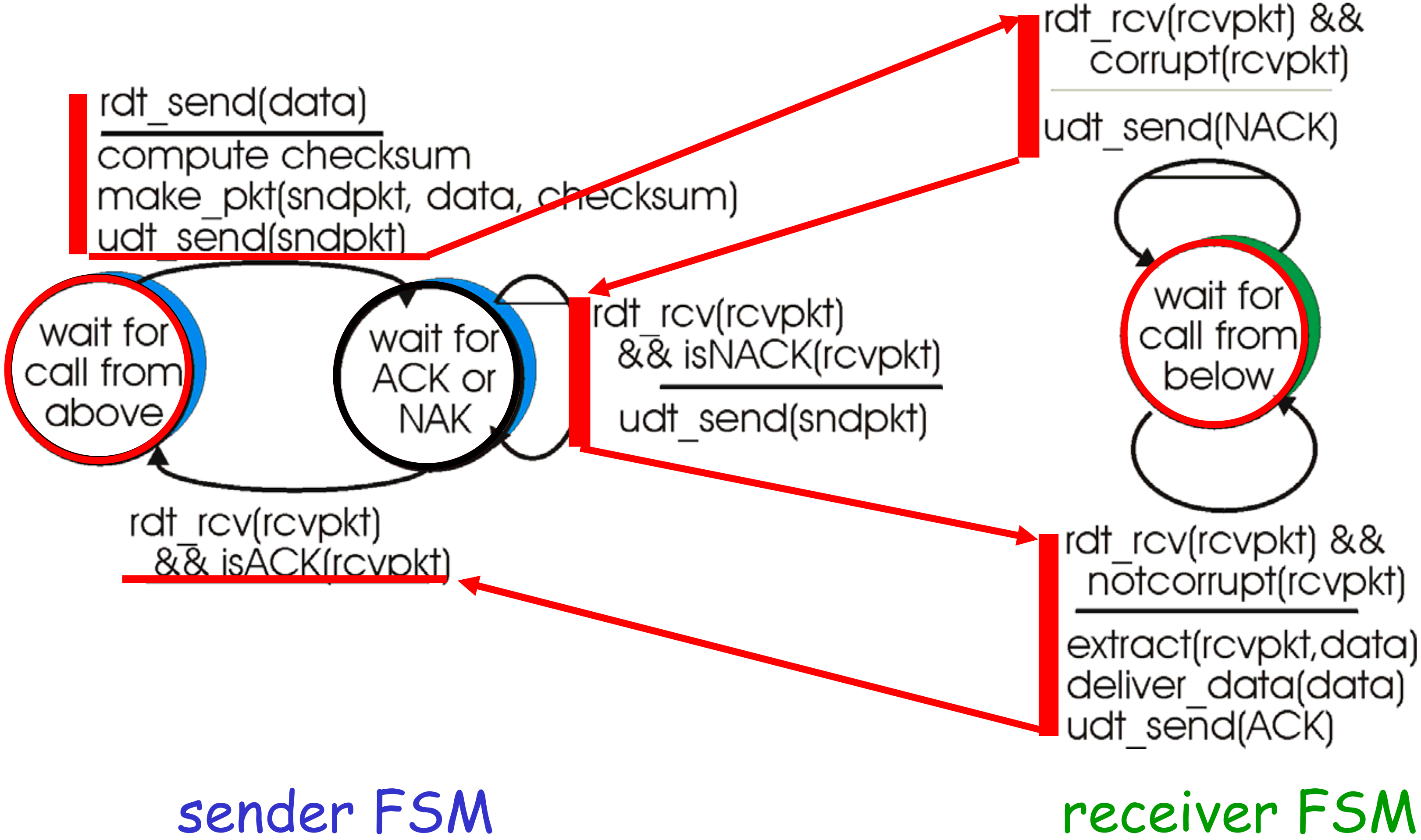
sender FSM

receiver FSM

rdt2.0: in action (no errors)



rdt2.0: in action (error scenario)



rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

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

What to do?

- sender ACKs/NAKs receiver's ACK/NAK? What if sender ACK/NAK lost?
- retransmit, but this might cause retransmission of correctly received pkt!

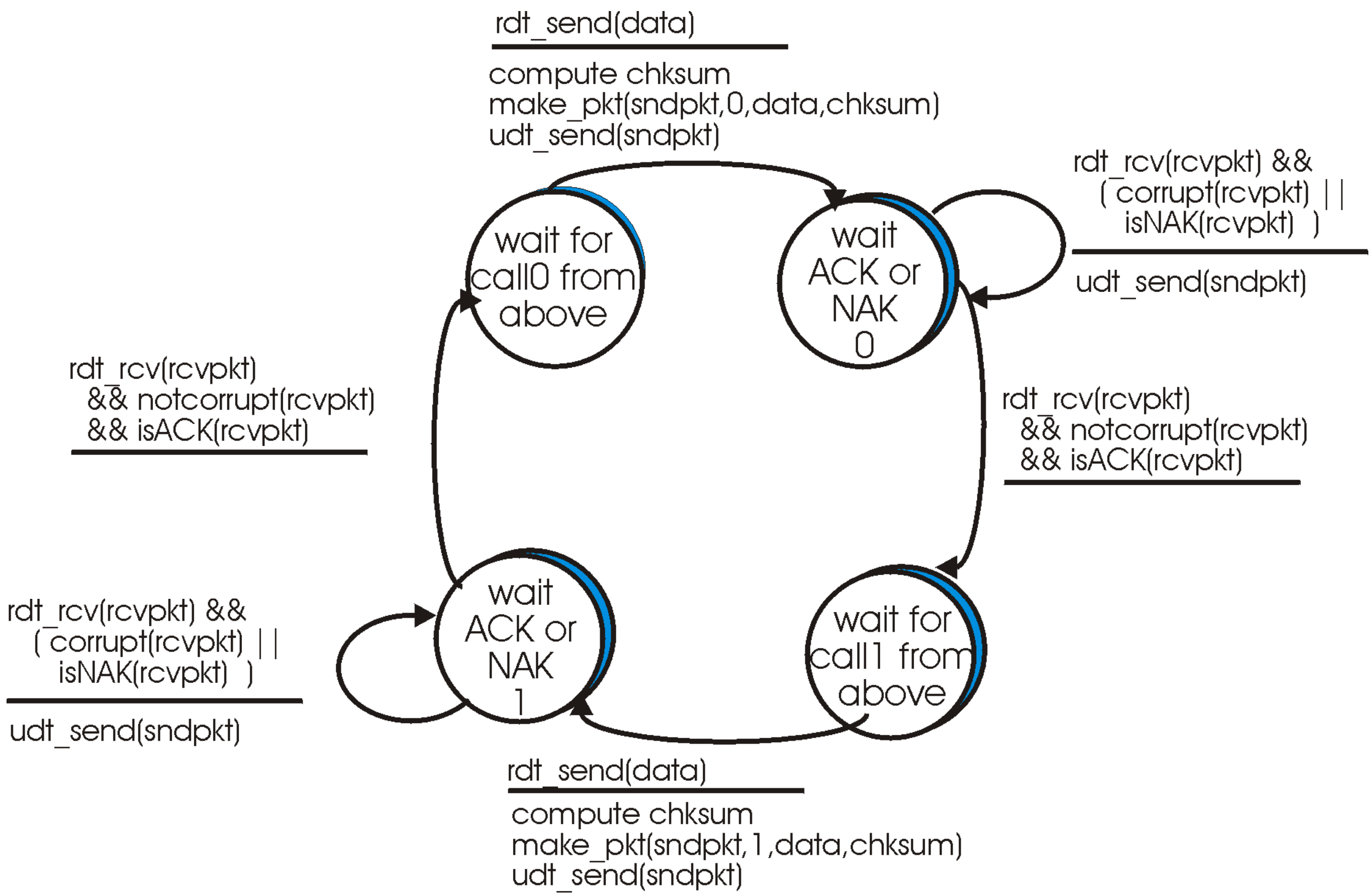
Handling duplicates:

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

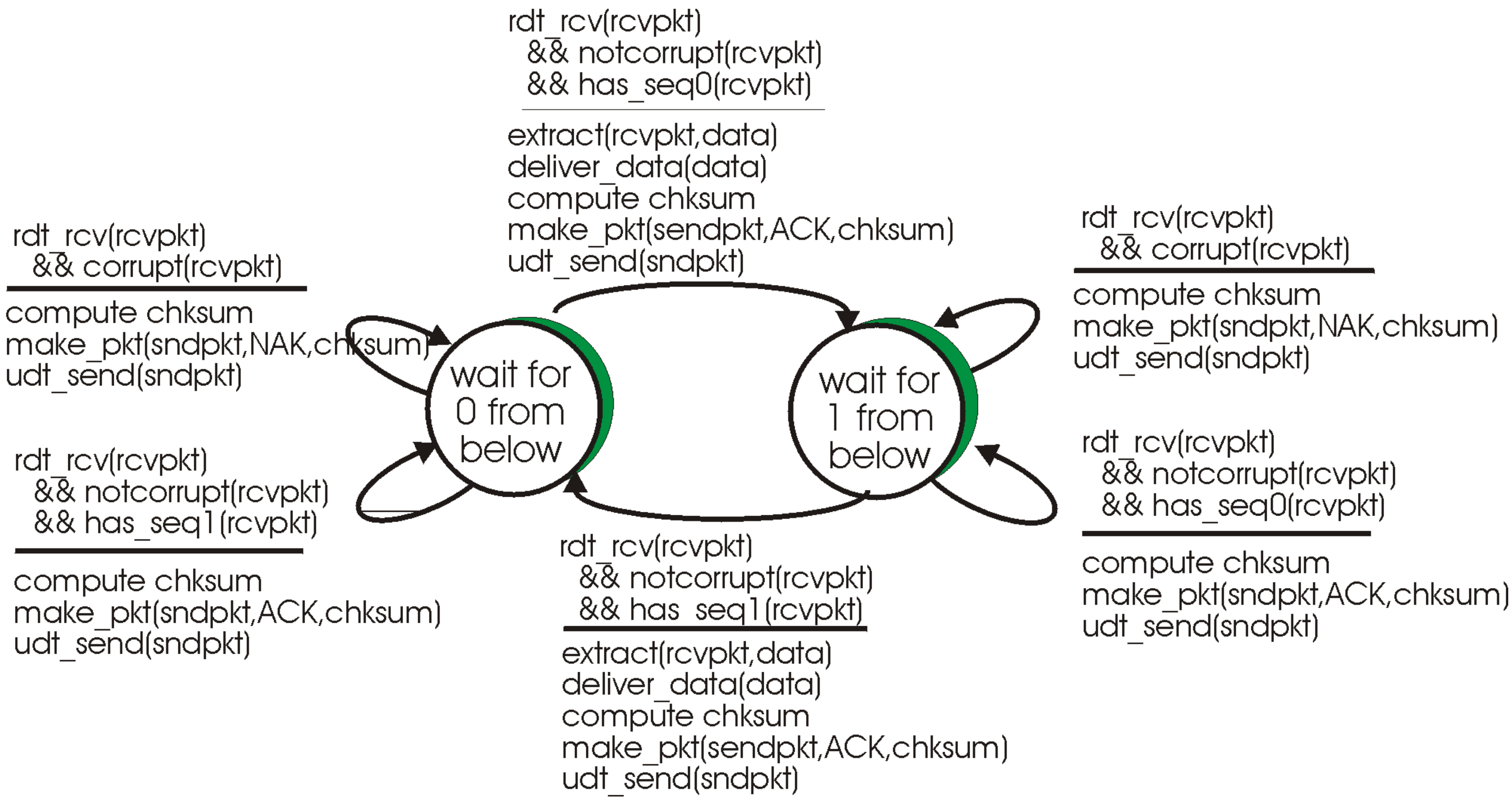
stop and wait

Sender sends one packet, then waits for receiver response

rdt2.1: sender, handles garbled ACK/NAKs



rdt2.1: receiver, handles garbled ACK/NAKs



rdt2.1: discussion

Sender:

- seq # added to pkt
- two seq. #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "current" pkt has 0 or 1 seq. #

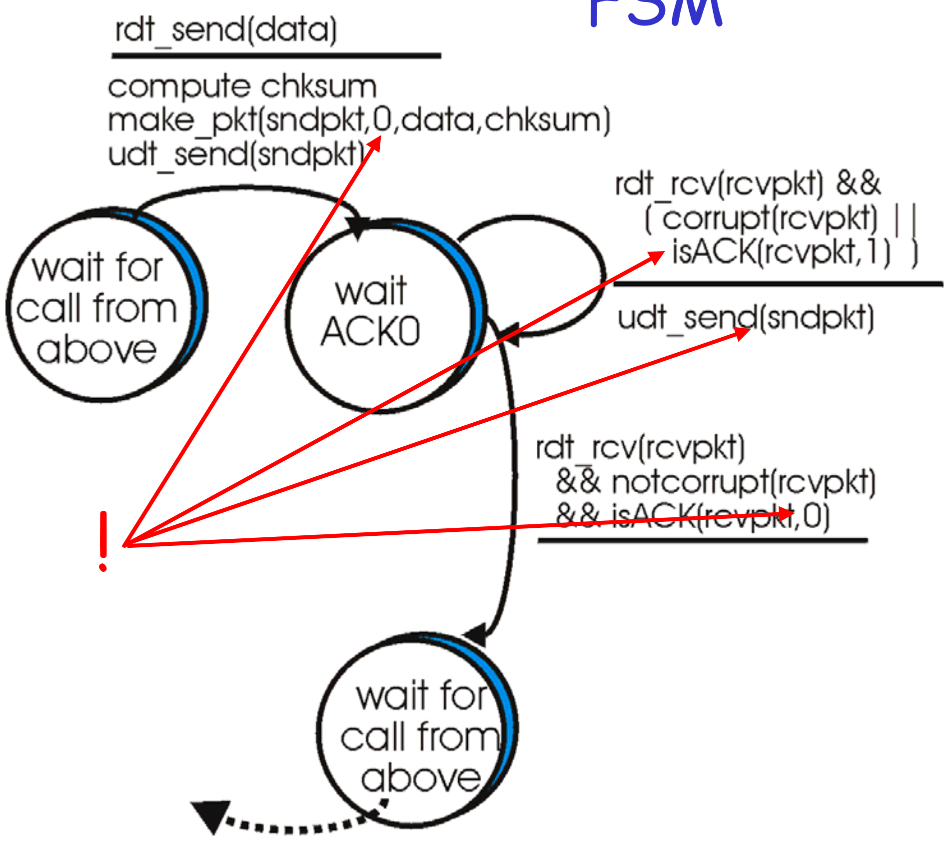
Receiver:

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

rdt2.2: a NAK-free protocol

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

sender FSM



rdt3.0: channels with errors *and* loss

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

- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

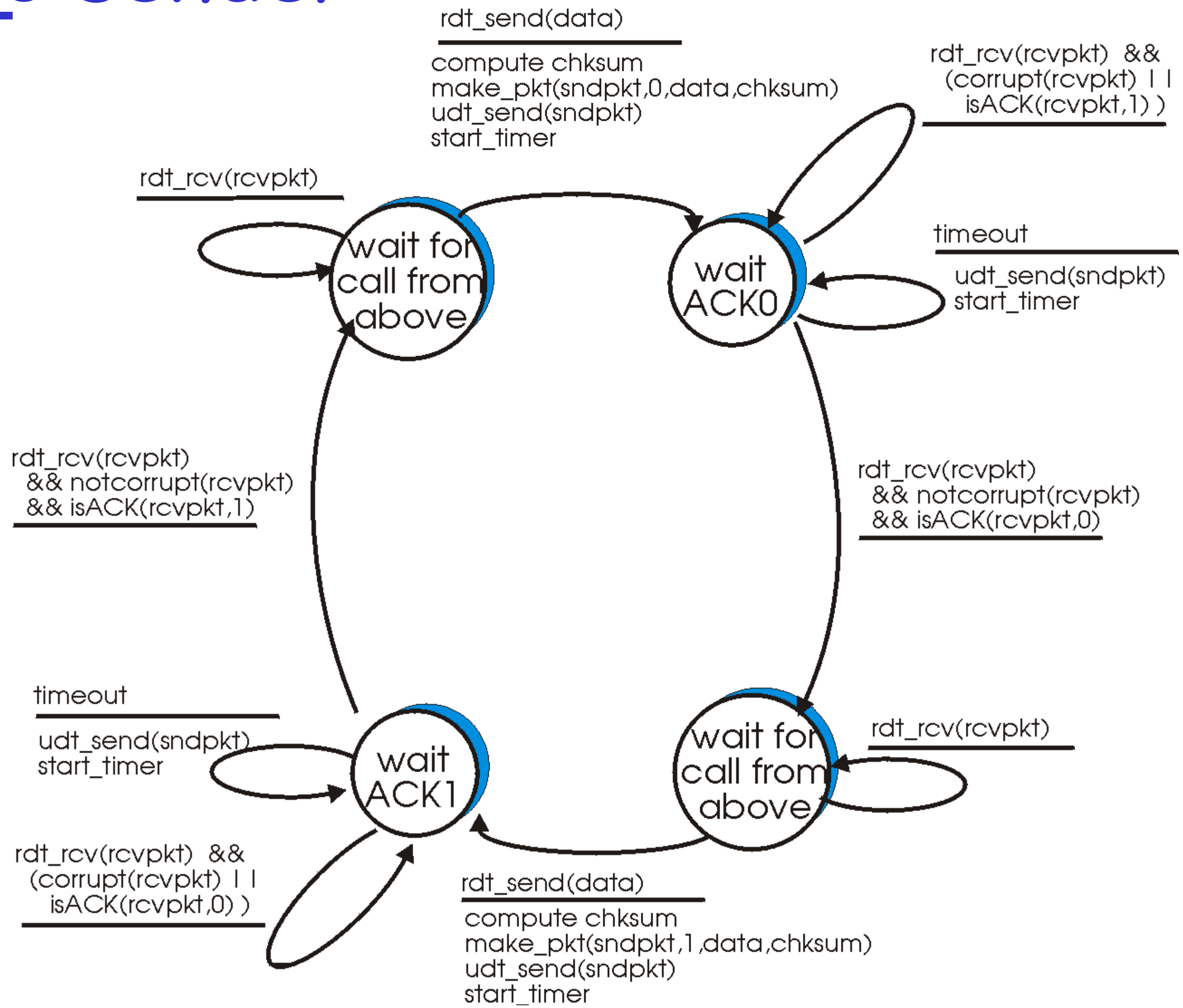
Q: how to deal with loss?

- sender waits until certain data or ACK lost, then retransmits
- yuck: drawbacks?

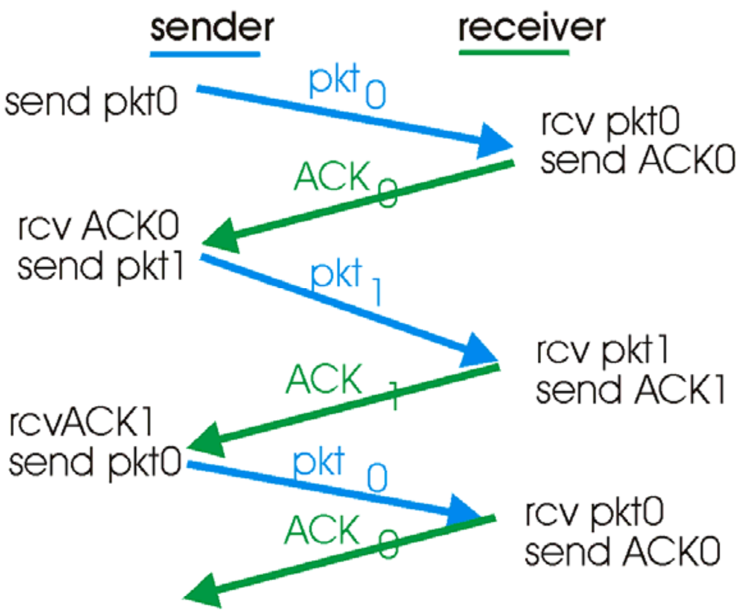
Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

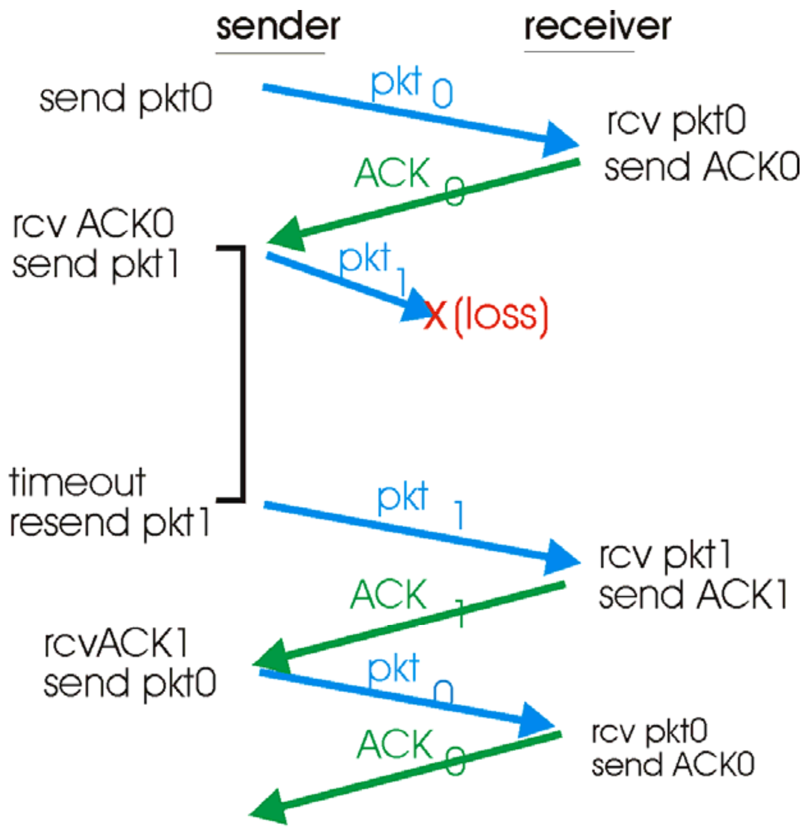
rdt3.0 sender



rdt3.0 in action

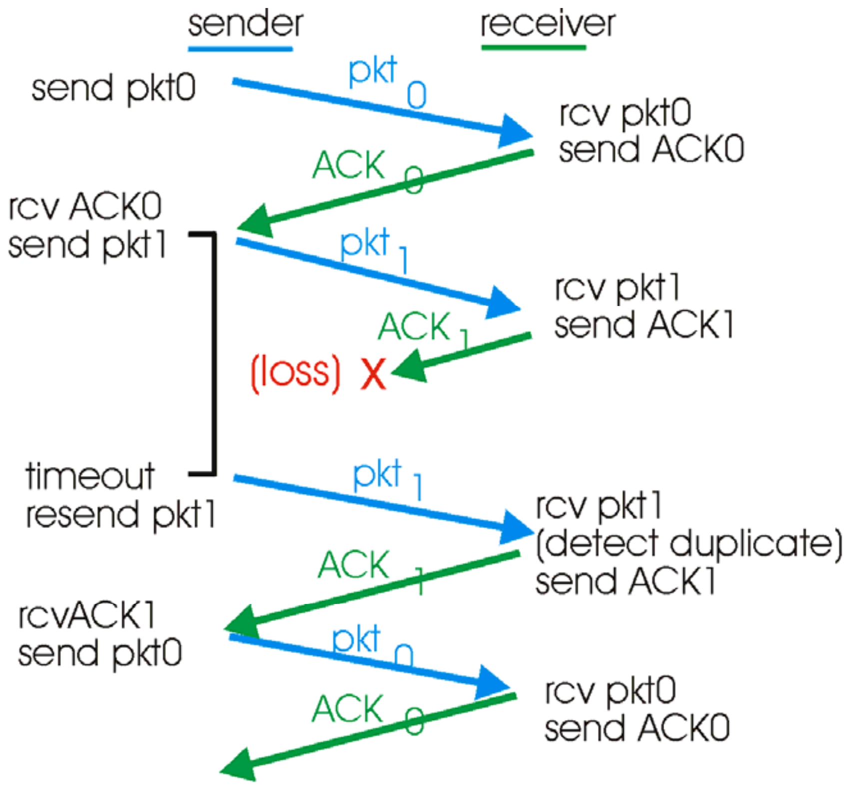


(a) operation with no loss

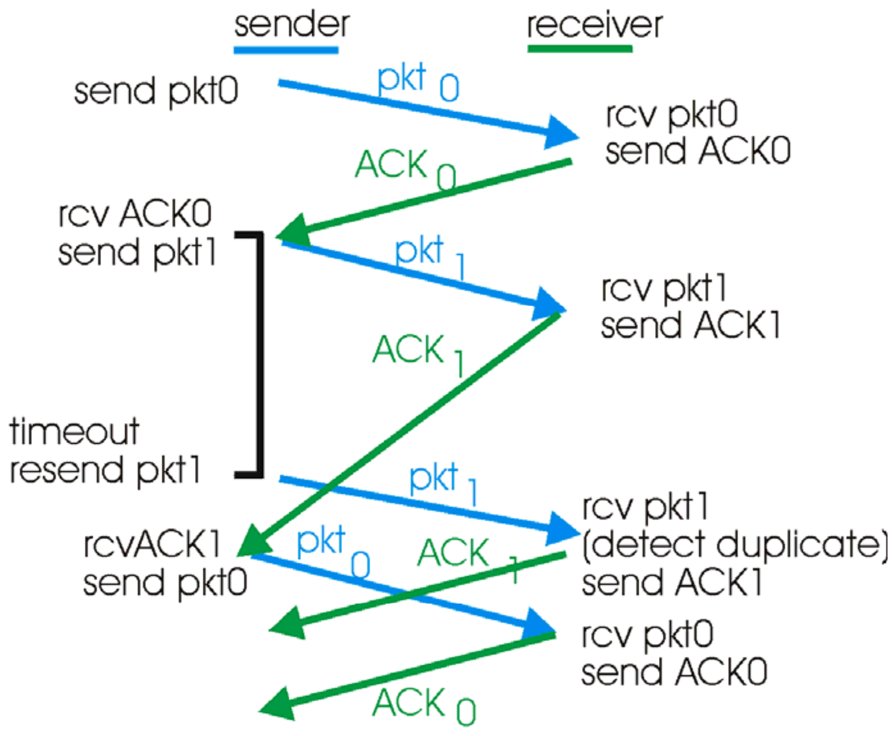


(b) lost packet

rdt3.0 in action



(c) lost ACK



(d) premature timeout

Performance of rdt3.0

- rdt3.0 works, but performance stinks
- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{8\text{kb/pkt}}{10^{**}9\text{ b/sec}} = 8\text{ microsec/pkt}$$

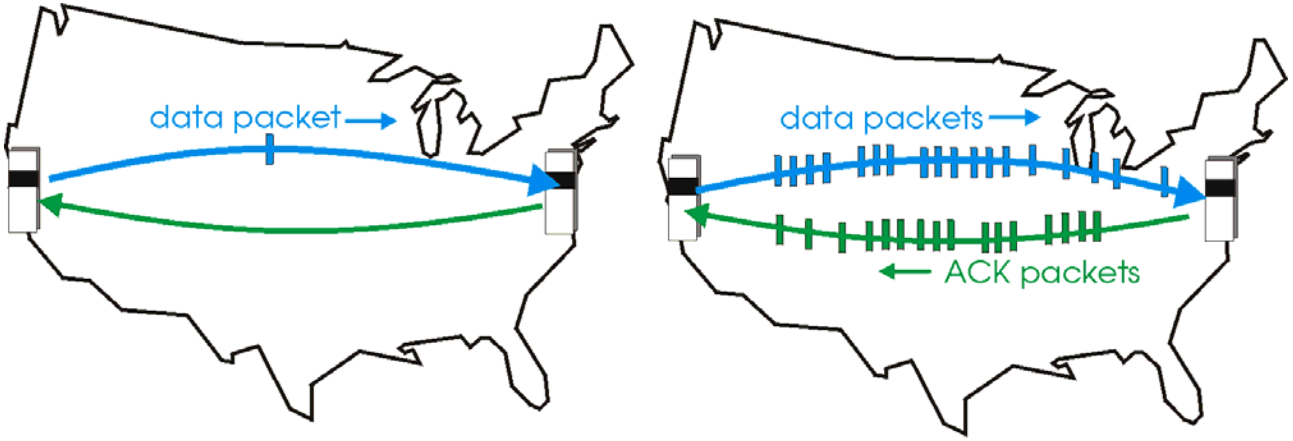
$$\color{red}{\text{Utilization}} = U = \frac{\text{fraction of time}}{\text{sender busy sending}} = \frac{8\text{ microsec}}{30.016\text{ msec}} = 0.00015$$

  - 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - network protocol limits use of physical resources!

Pipelined protocols

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

- 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, selective repeat*

Performance of rdt3.0

- rdt3.0 works, but performance stinks
- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{8kb/pkt}{10^{**}9\ b/sec} = 8 \text{ microsec}$$

$$\text{Utilization} = U = \frac{\text{fraction of time}}{\text{sender busy sending}} = \frac{8 \text{ microsec}}{30.016 \text{ msec}} = 0.00015$$

  - 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - network protocol limits use of physical resources!

Pipelined protocols

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

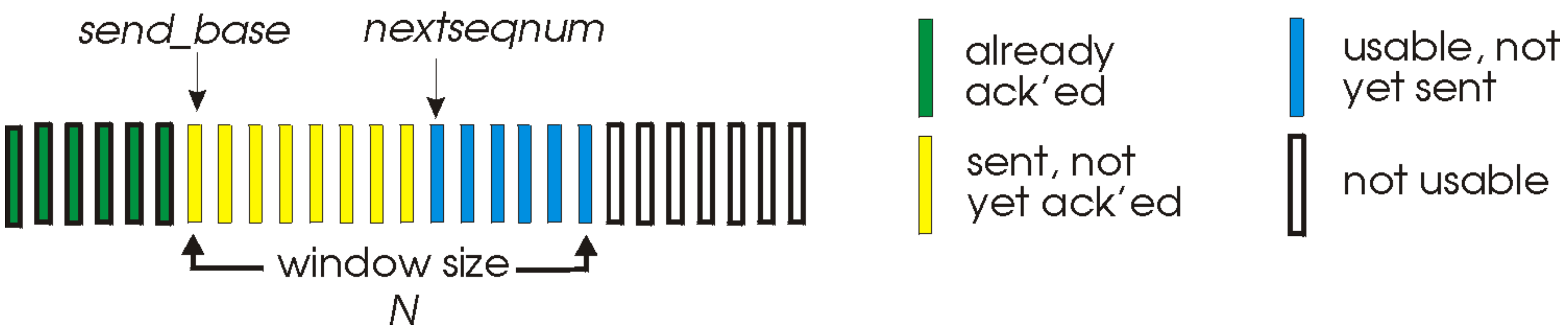
- range of sequence numbers must be increased
- buffering at sender and/or receiver

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Go-Back-N

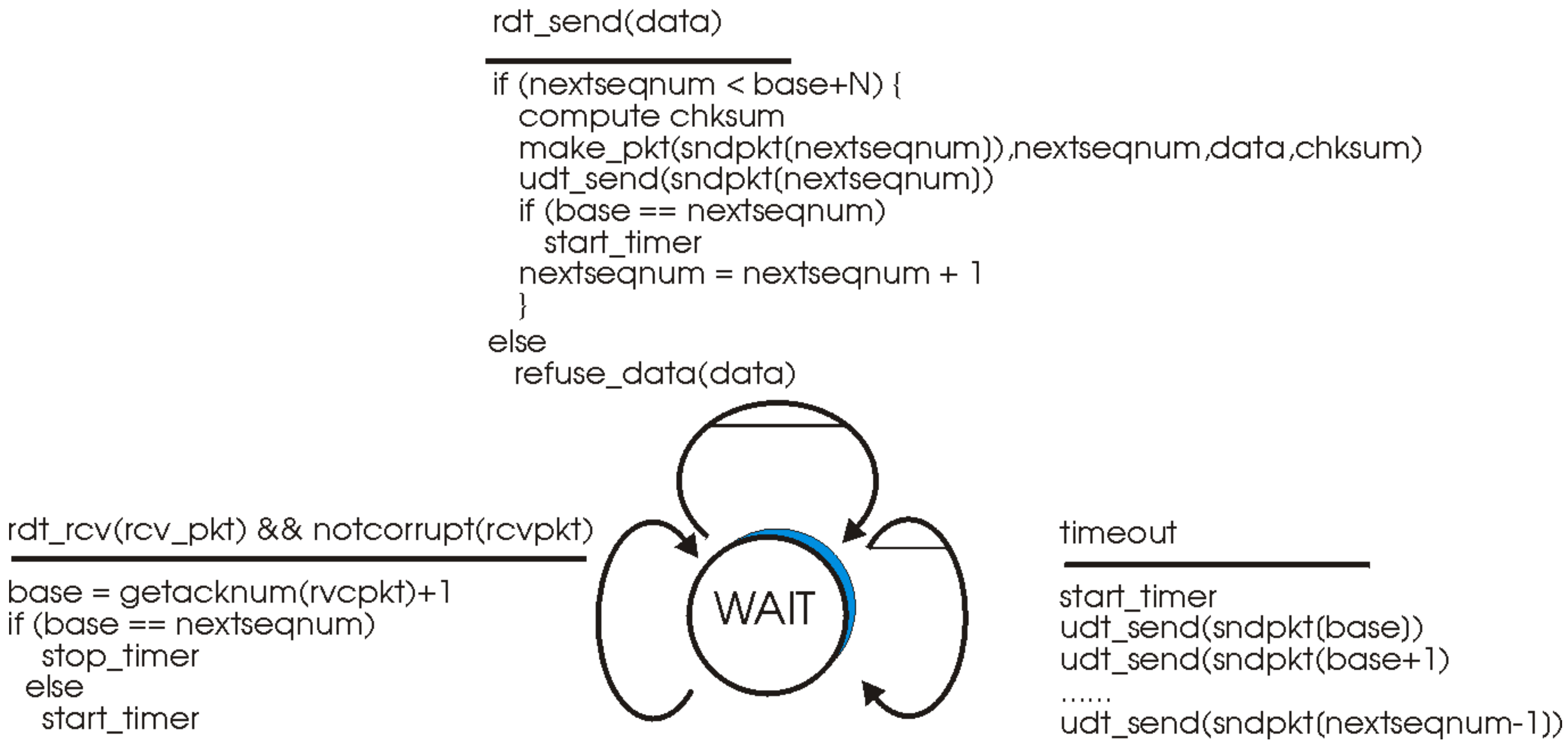
Sender:

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed

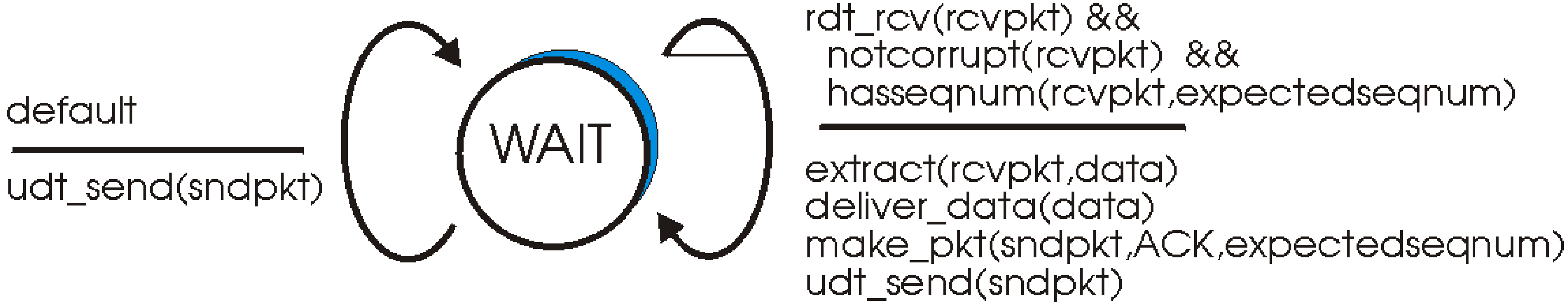


- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

GBN: sender extended FSM



GBN: receiver extended FSM

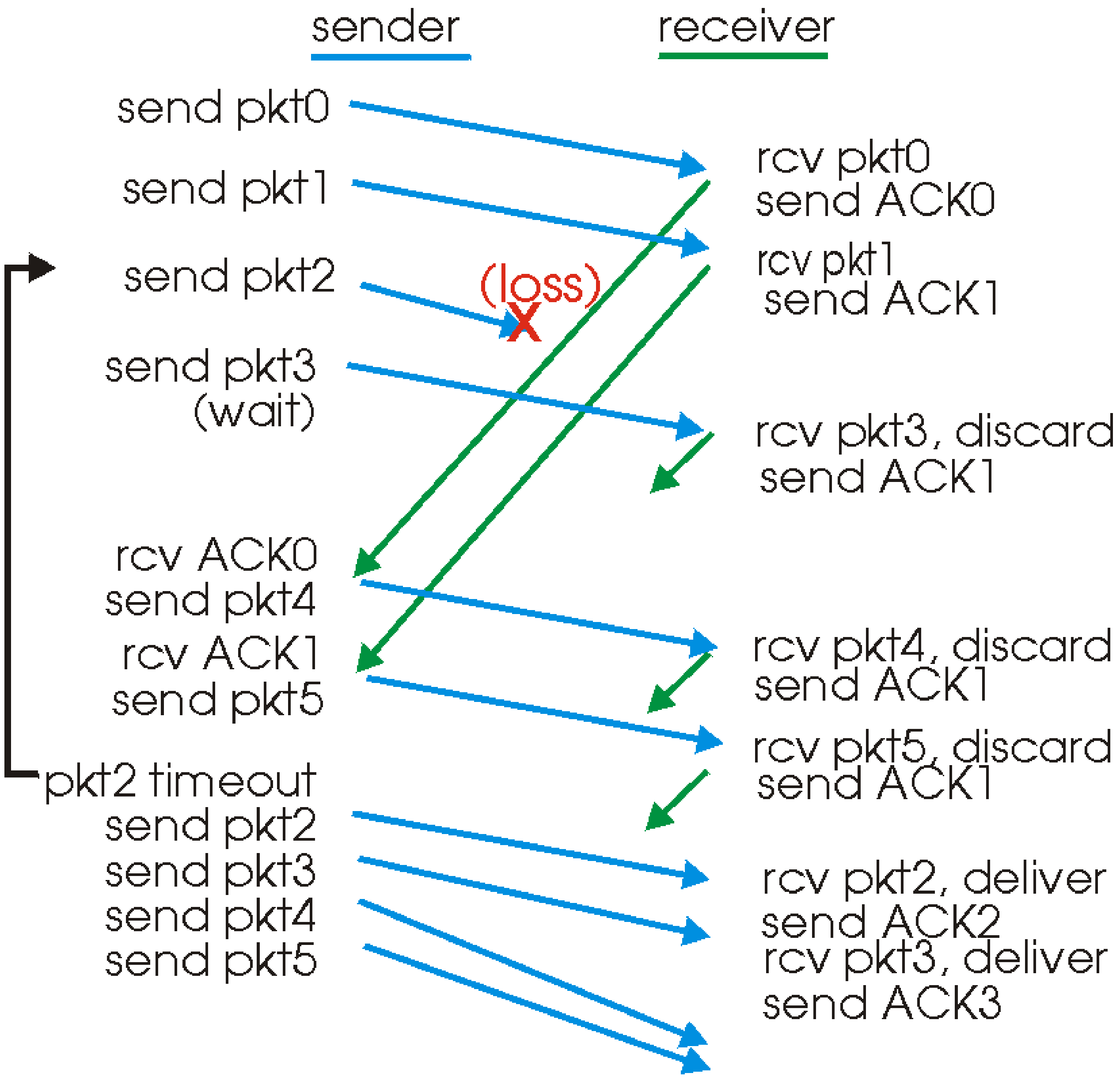


receiver simple:

- ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember **expectedseqnum**
- out-of-order pkt:
  - discard (don't buffer) -> no receiver buffering!
  - ACK pkt with highest in-order seq #

GBN in action

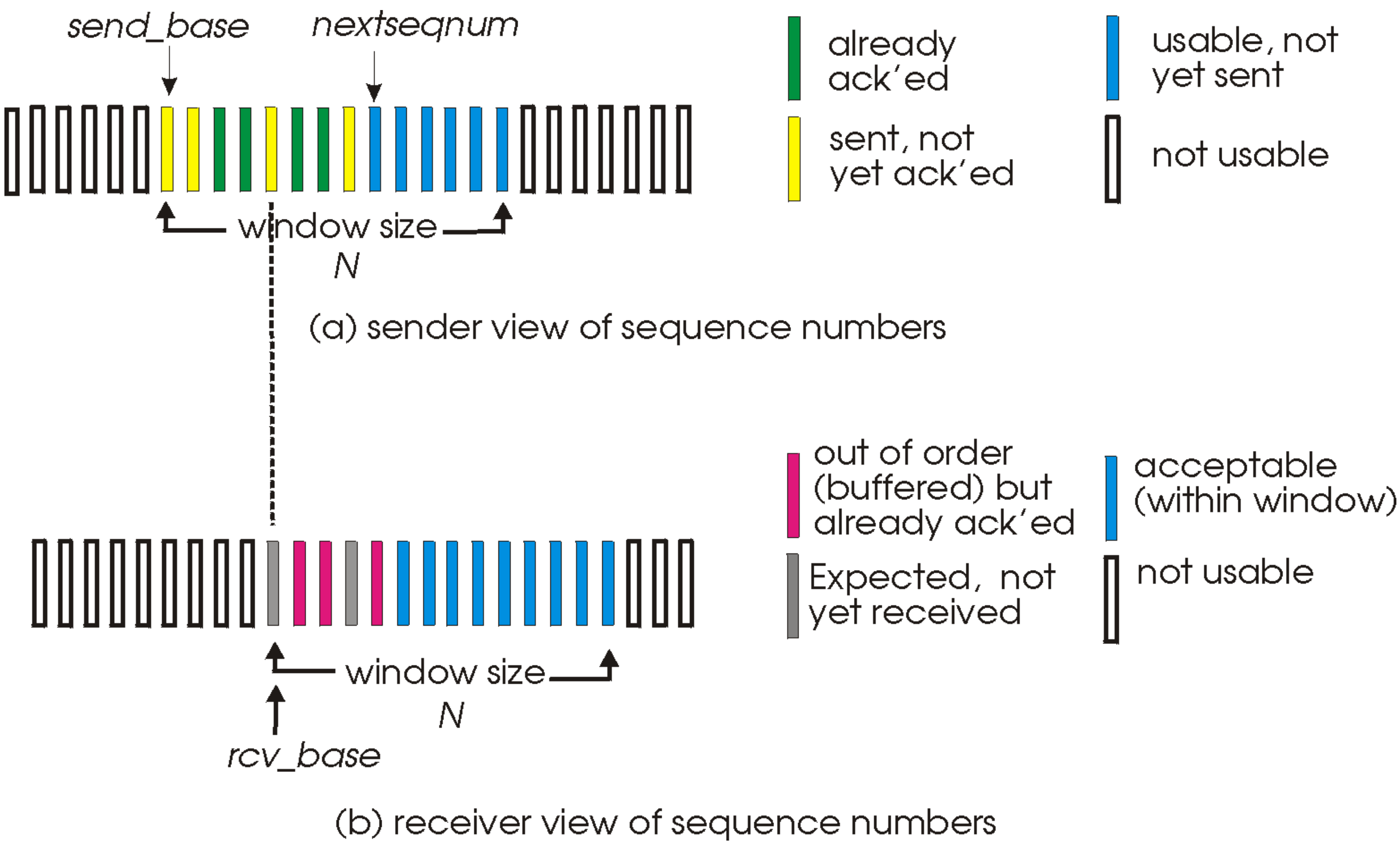
N=4



Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #s of sent, unACKed pkts

Selective repeat: sender, receiver windows



Selective repeat

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 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

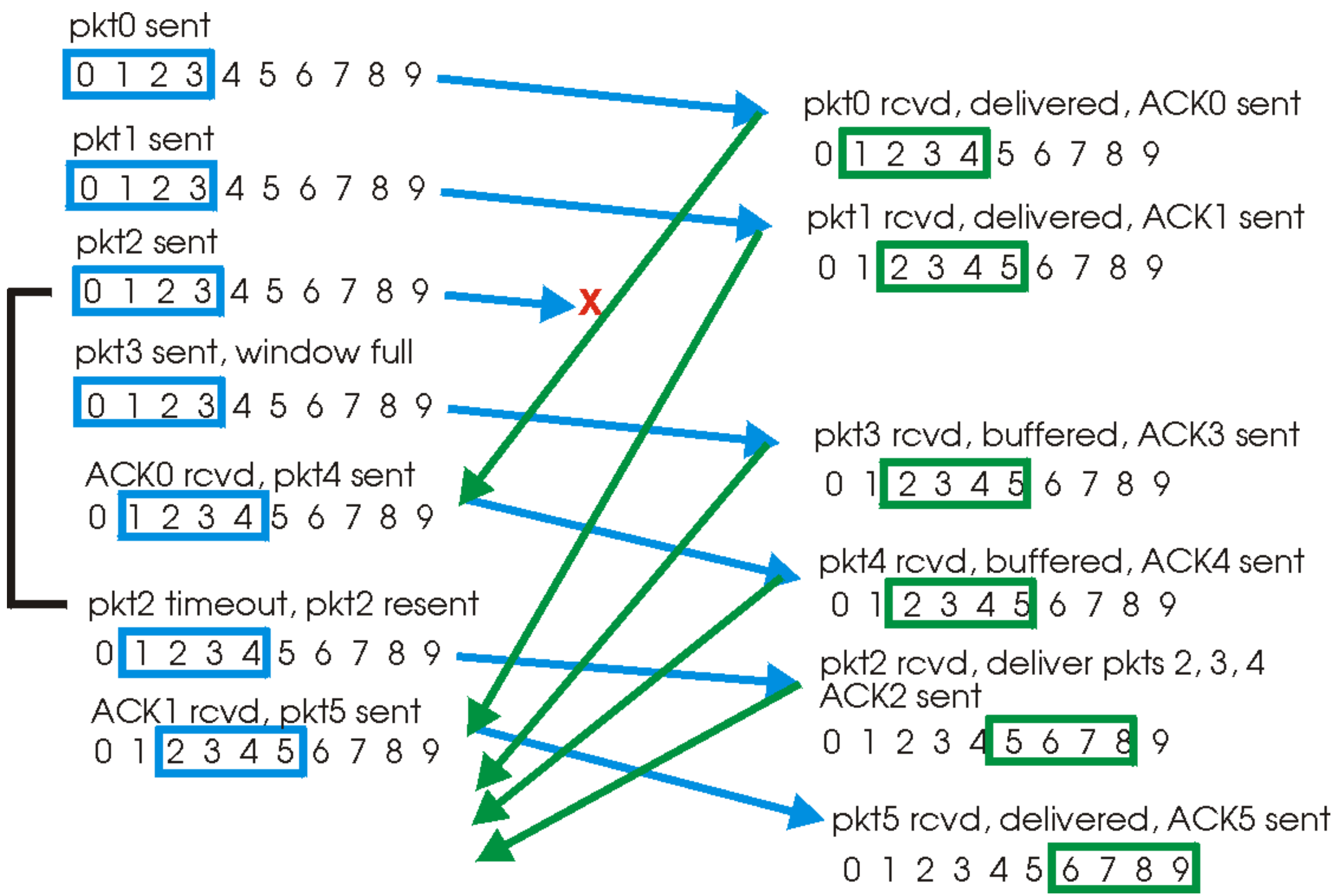
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

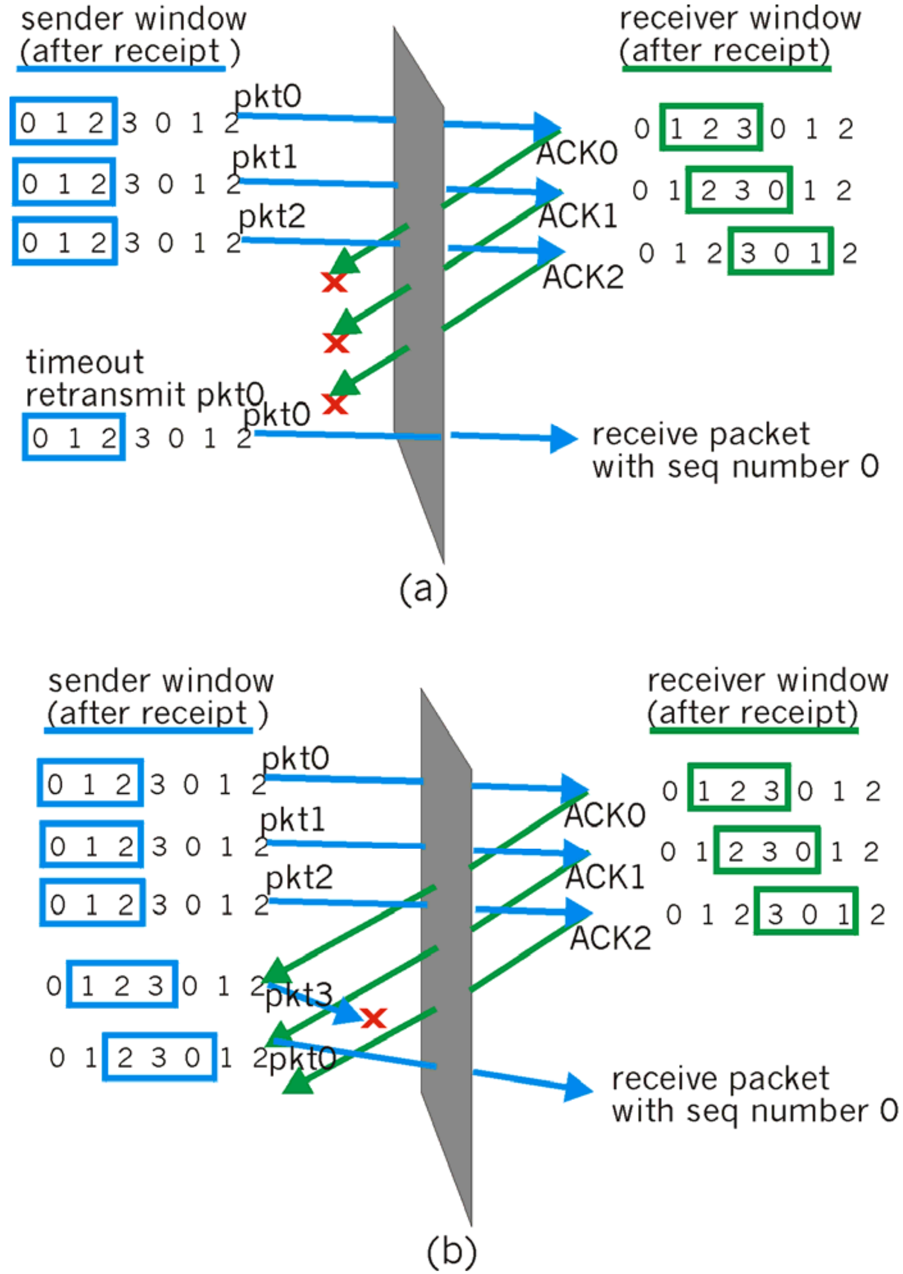
Selective repeat in action



Selective repeat: dilemma

Example:

- seq #'s: 0, 1, 2, 3
- window size=3



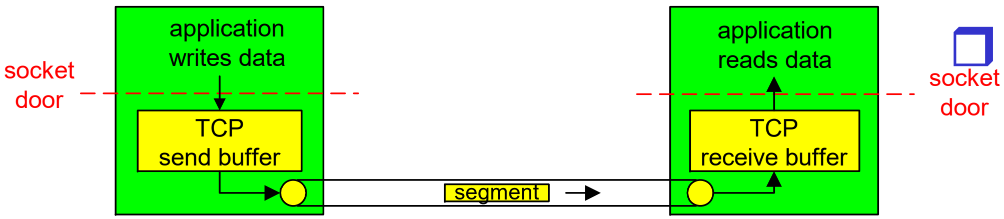
- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

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

TCP: Overview

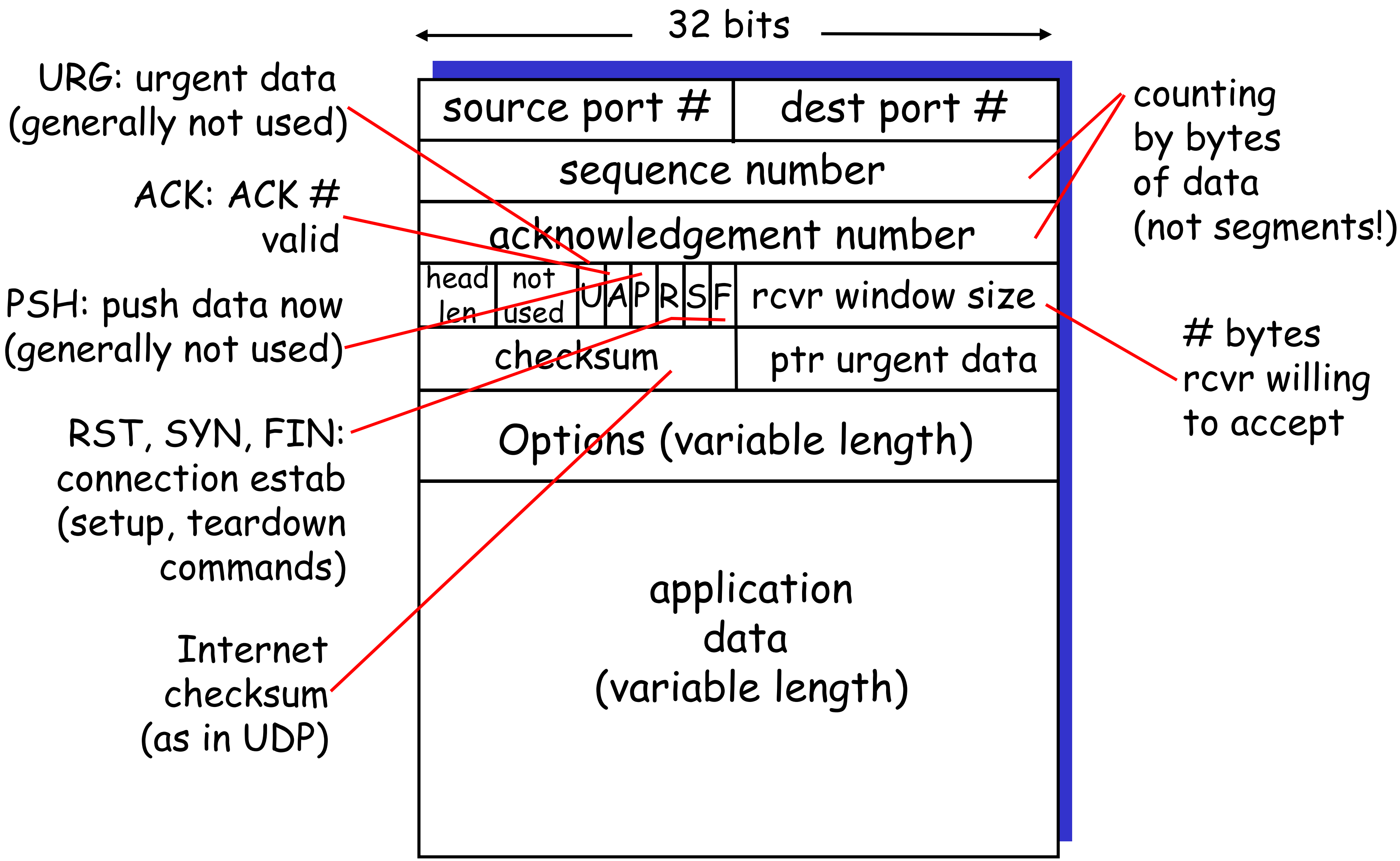
RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size



- 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



TCP seq. #'s and ACKs

Seq. #'s:

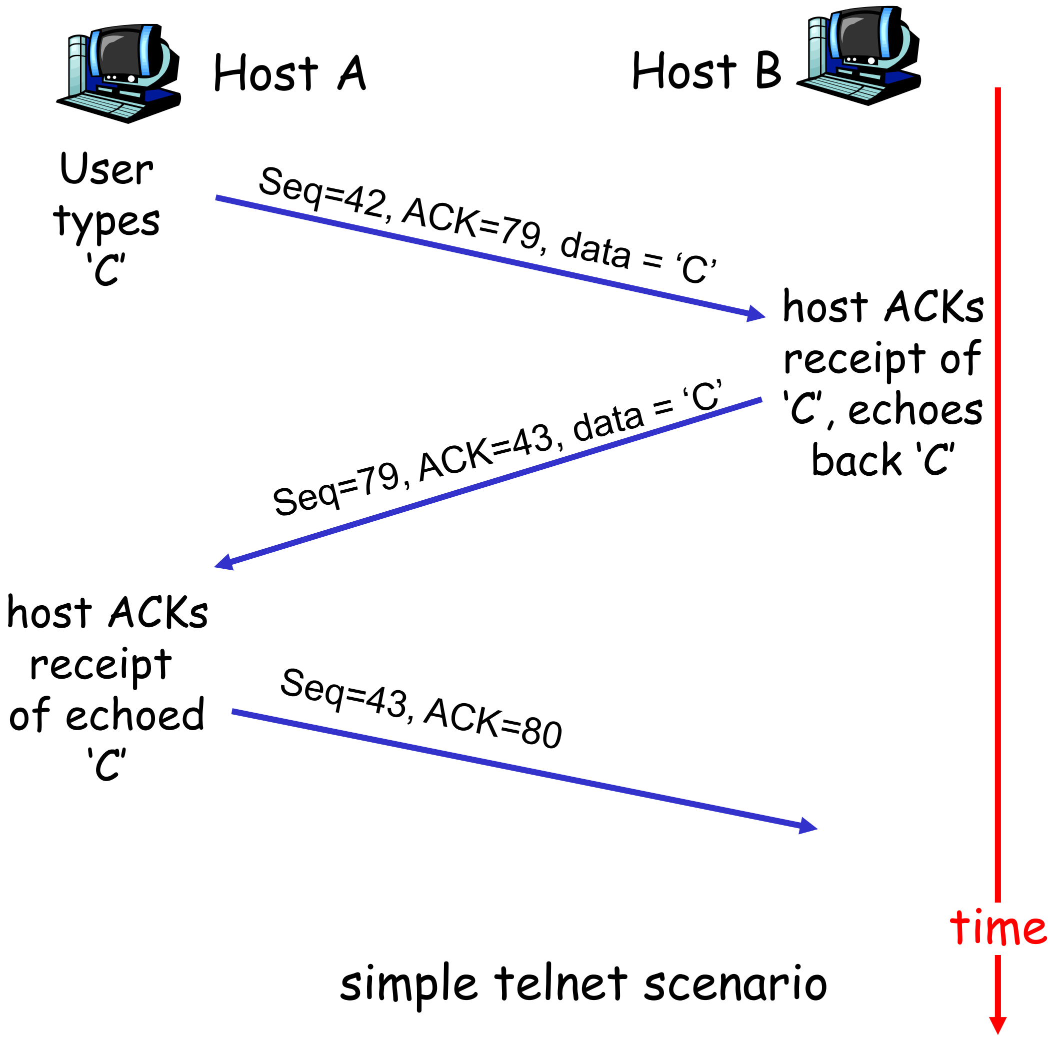
- byte stream "number" of first byte in segment's data

ACKs:

- seq # of next byte expected from other side
- cumulative ACK

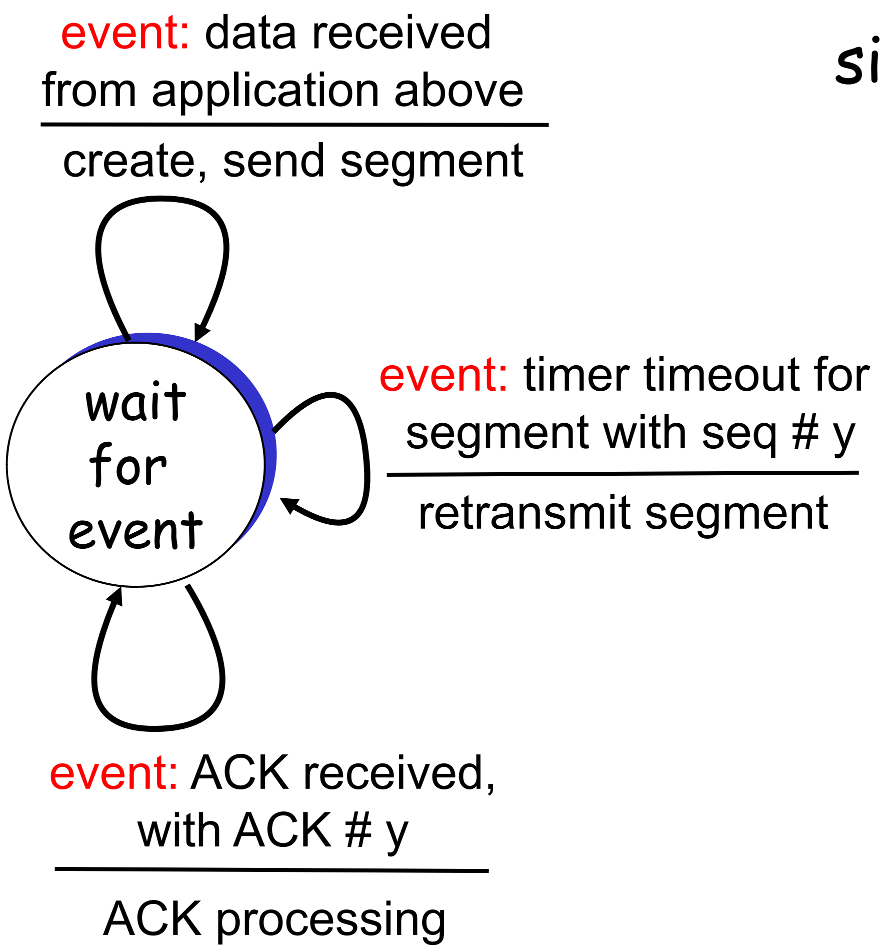
Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor



simple telnet scenario

TCP: reliable data transfer



simplified sender, assuming

- one way data transfer
- no flow, congestion control

TCP: reliable data transfer

Simplified TCP sender

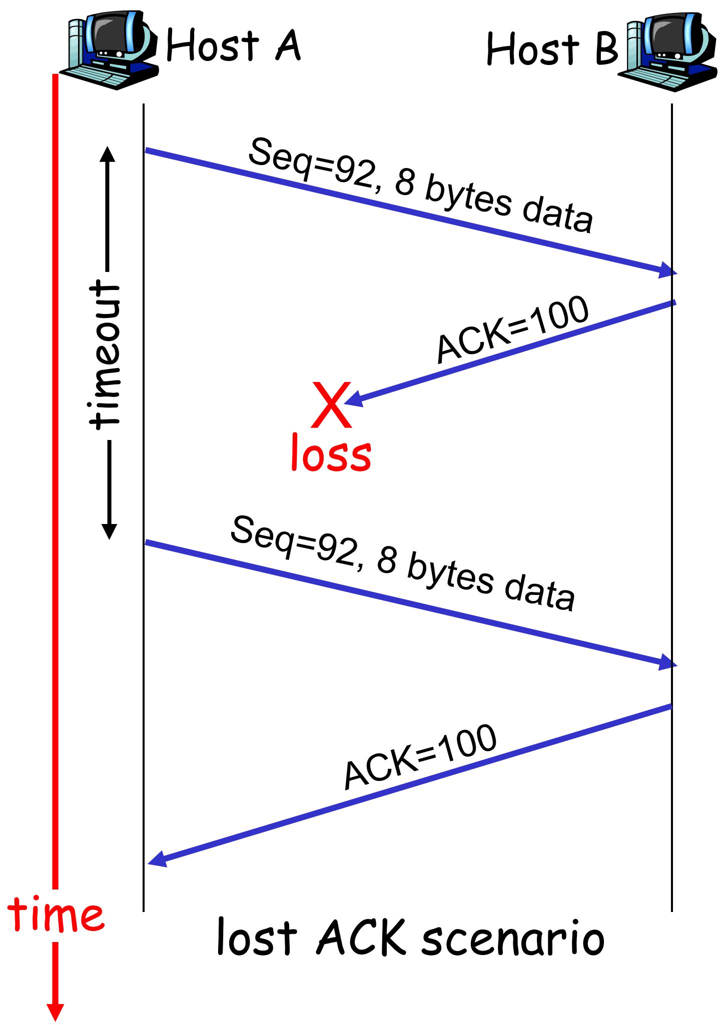
```
sendbase = initial_sequence number
nextseqnum = initial_sequence number

loop (forever) {
  switch(event)
  event: data received from application above
      create TCP segment with sequence number nextseqnum
      start timer for segment nextseqnum
      pass segment to IP
      nextseqnum = nextseqnum + length(data)
  event: timer timeout for segment with sequence number y
      retransmit segment with sequence number y
      compute new timeout interval for segment y
      restart timer for sequence number y
  event: ACK received, with ACK field value of y
      if (y > sendbase) { /* cumulative ACK of all data up to y */
         cancel all timers for segments with sequence numbers < y
         sendbase = y
         }
      else { /* a duplicate ACK for already ACKed segment */
         increment number of duplicate ACKs received for y
         if (number of duplicate ACKS received for y == 3) {
            /* TCP fast retransmit */
            resend segment with sequence number y
            restart timer for segment y
         }
} /* end of loop forever */
```

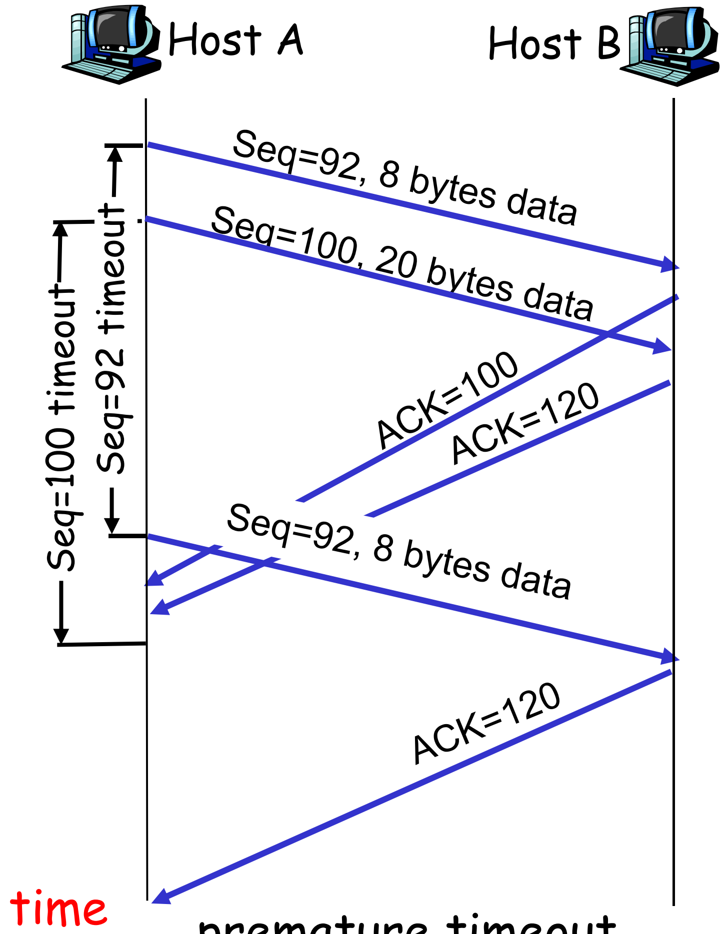
TCP ACK generation [RFC 1122, RFC 2581]

Event	TCP Receiver action
in-order segment arrival, no gaps, everything else already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
in-order segment arrival, no gaps, one delayed ACK pending	immediately send single cumulative ACK
out-of-order segment arrival higher-than-expected seq. # gap detected	send duplicate ACK, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate ACK if segment starts at lower end of gap

TCP: retransmission scenarios



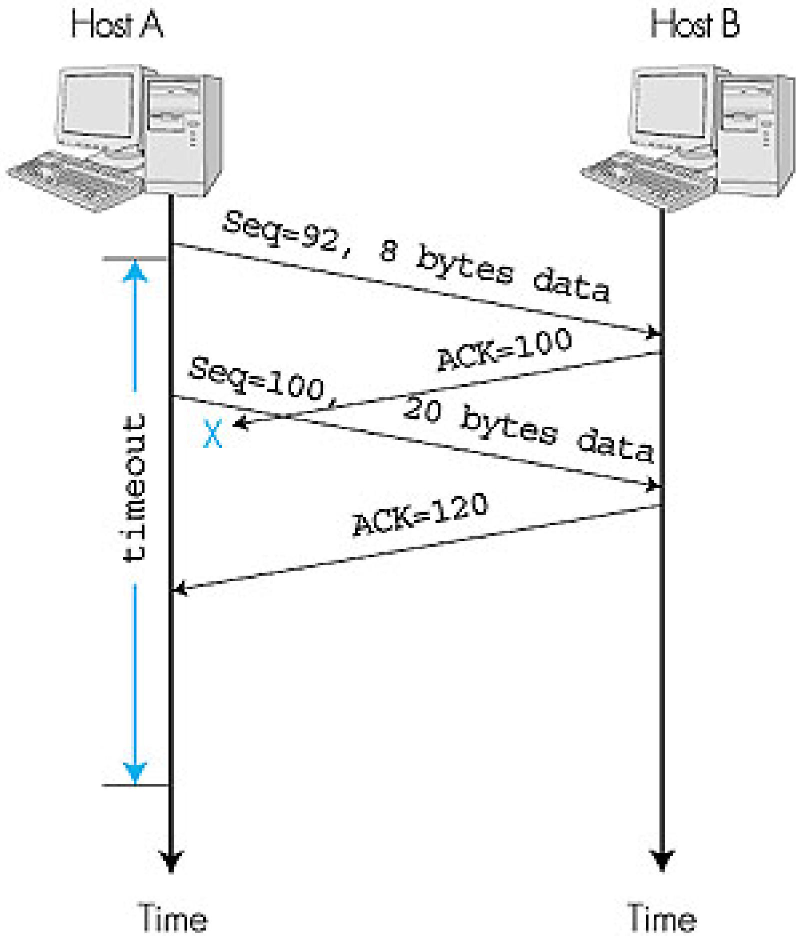
lost ACK scenario



premature timeout, cumulative ACKs

TCP: third retransmission scenario

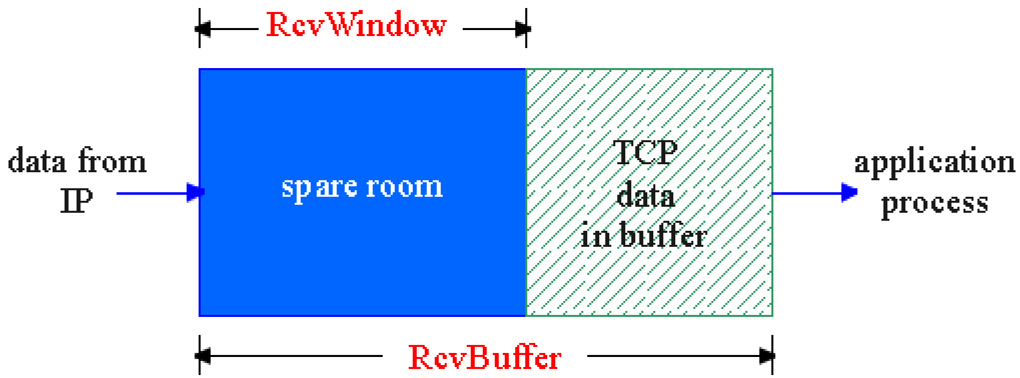
- Cumulative acknowledgement avoids retransmission of first segment



TCP Flow Control

flow control

sender won't overrun receiver's buffers by transmitting too much, too fast



receiver buffering

receiver: explicitly informs sender of (dynamically changing) amount of free buffer space

- `rcvr window size` **field** in TCP segment

sender: amount of transmitted, unACKed data less than most recently-received `rcvr window size`

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?

- longer than RTT
  - note: RTT will vary
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?

- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions, cumulatively ACKed segments
- **SampleRTT** will vary, want estimated RTT "smoother"
  - use several recent measurements, not just current **SampleRTT**

TCP Round Trip Time and Timeout

`EstimatedRTT = (1-x)*EstimatedRTT + x*SampleRTT`

- Exponential weighted moving average
- influence of given sample decreases exponentially fast
- typical value of x: 0.1

Setting the timeout

- RTT plus "safety margin"
- large variation in **`EstimatedRTT`** -> larger safety margin

```
Timeout = EstimatedRTT + 4*Deviation

Deviation = (1-x)*Deviation +
             x*abs(SampleRTT-EstimatedRTT)
```

TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments

- initialize TCP variables:
  - seq. #s
  - buffers, flow control info (e.g. `RcvWindow`)
- *client:* connection initiator

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

- *server:* contacted by client

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

Three way handshake:

step 1: client end system sends TCP SYN control segment to server
- specifies initial seq #

step 2: server end system receives SYN, replies with SYNACK control segment
- ACKs received SYN
- allocates buffers
- specifies server-> receiver initial seq. #

step 3: client allocates buffer and variables, send ACK to ack SYNACK

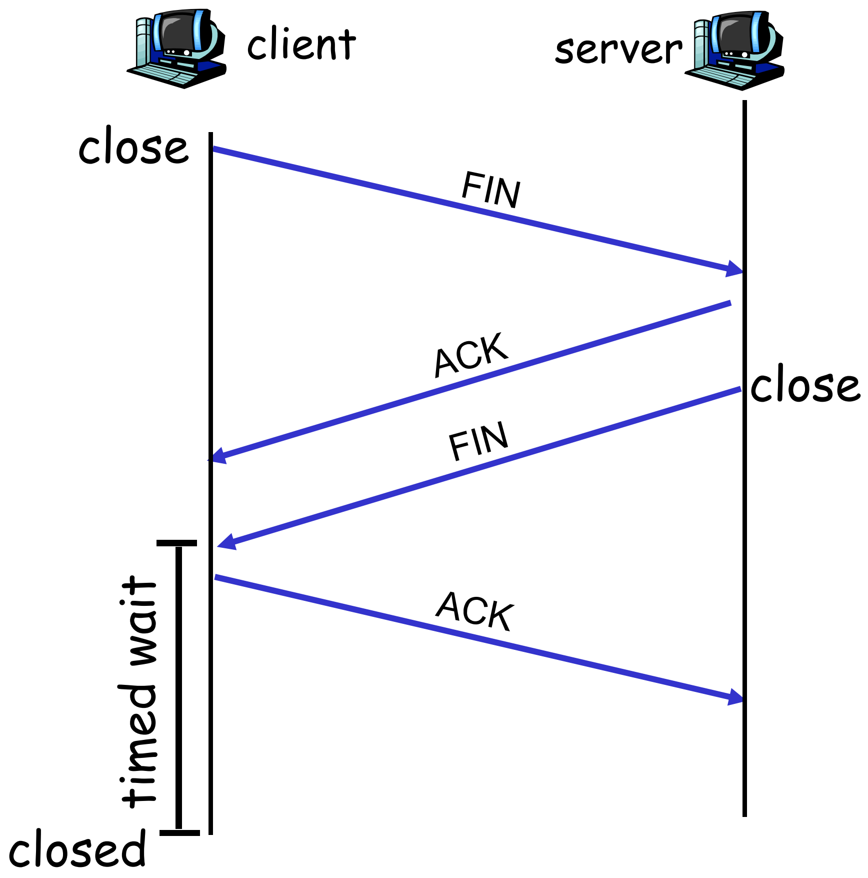
TCP Connection Management (cont.)

Closing a connection:

client closes socket:
`clientSocket.close();`

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.

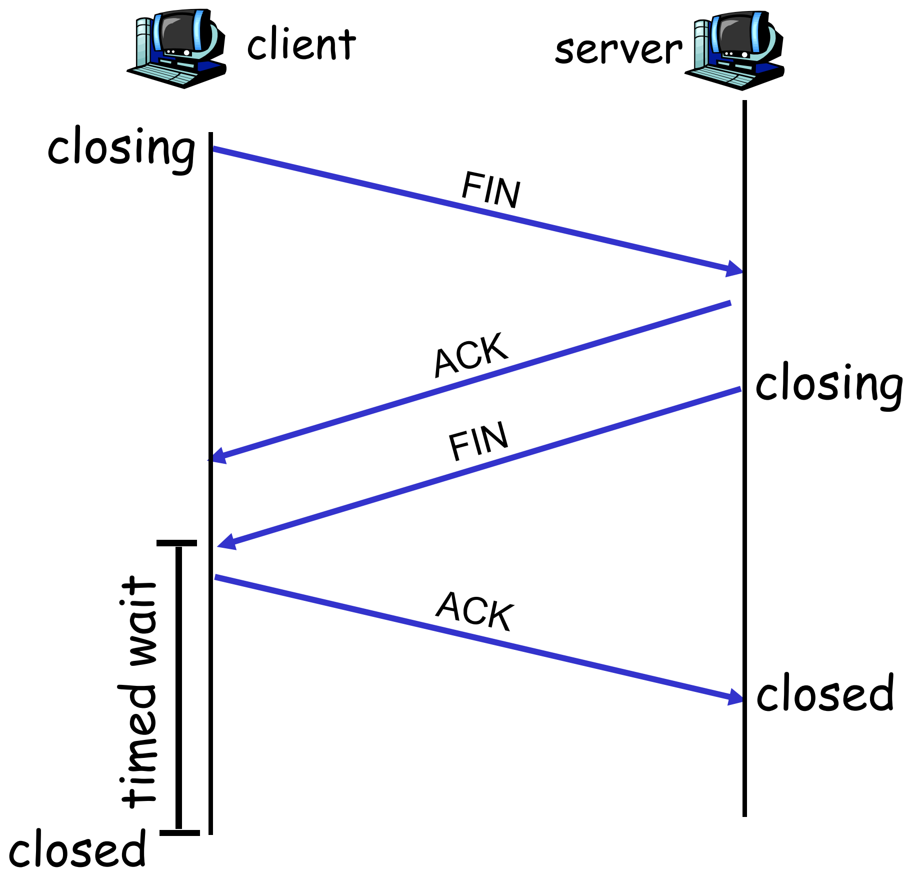


TCP Connection Management (cont.)

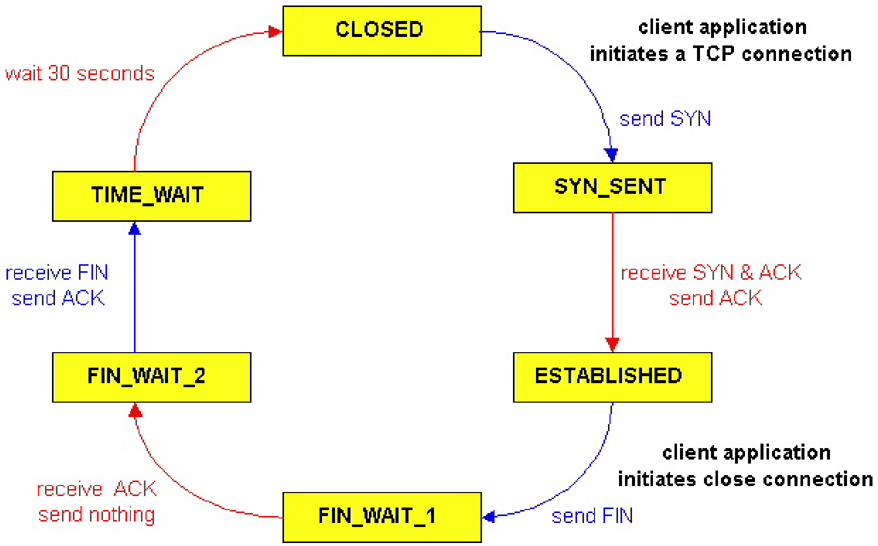
Step 3: client receives FIN, replies with ACK.

- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

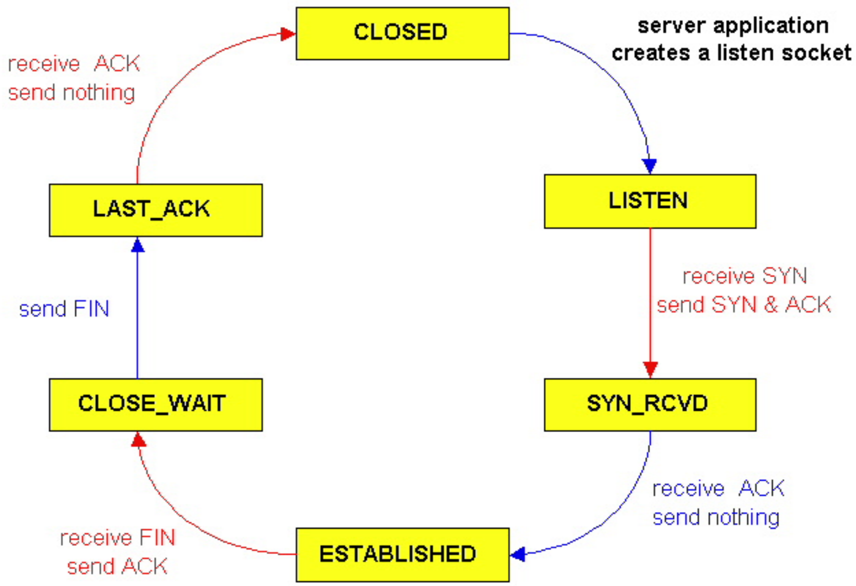


TCP Connection Management (cont)



TCP client lifecycle

TCP server lifecycle



Principles of Congestion Control

Congestion:

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

Principles of Congestion Control

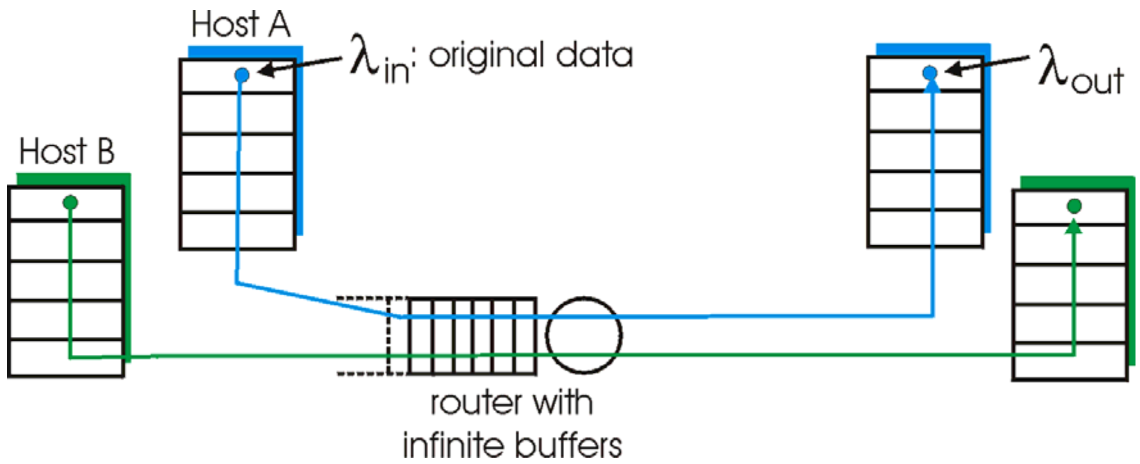
Congestion:

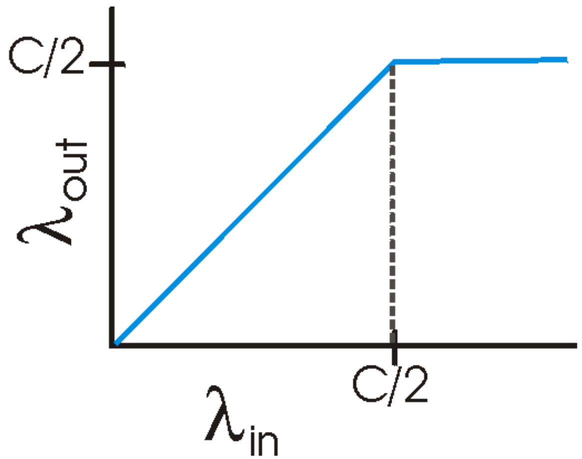
- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- Manifestations (symptoms):
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

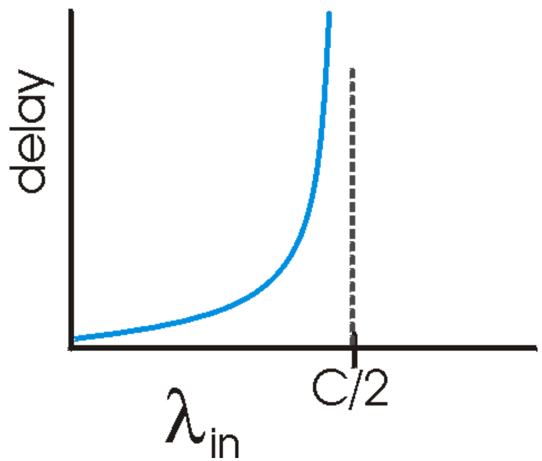
Causes/costs of congestion: scenario 1

Assumptions:

- two senders, two receivers
- one router, infinite buffers
- no retransmission
- C - link capacity



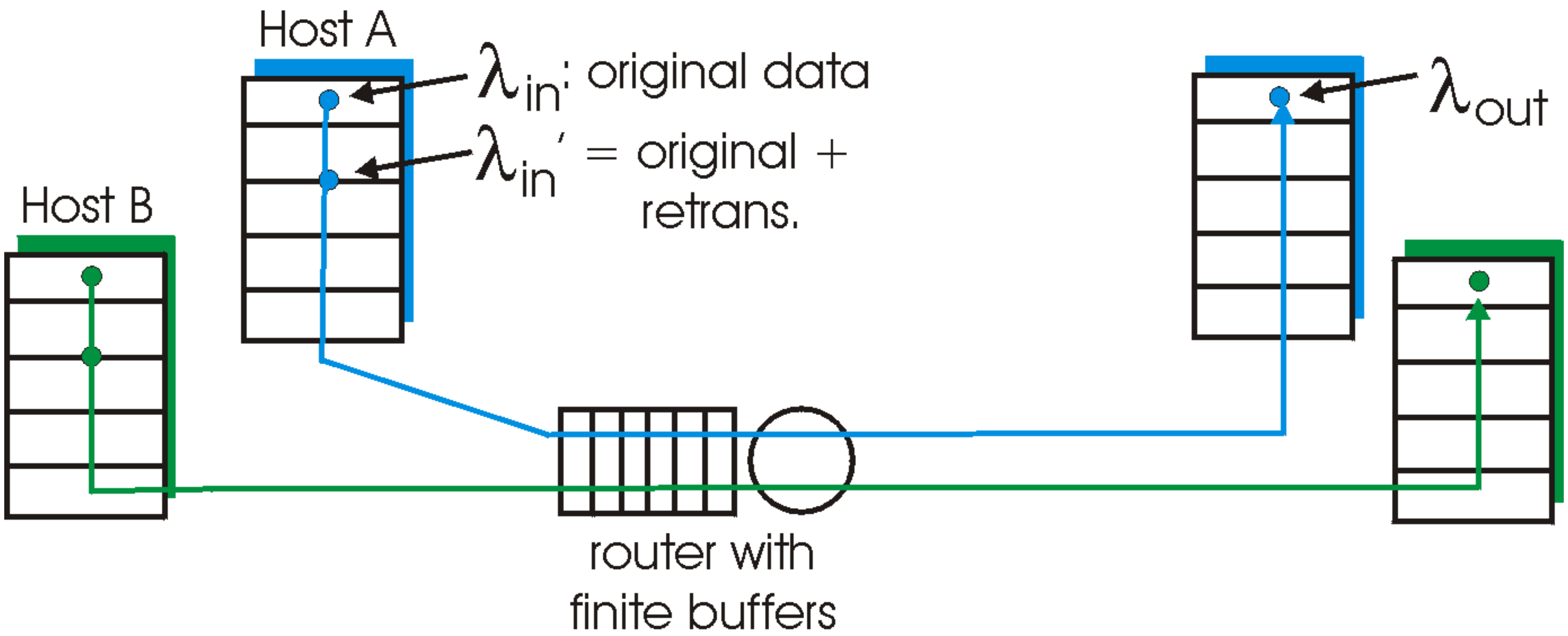




- large delays when congested (nearing link capacity; at maximum achievable throughput)

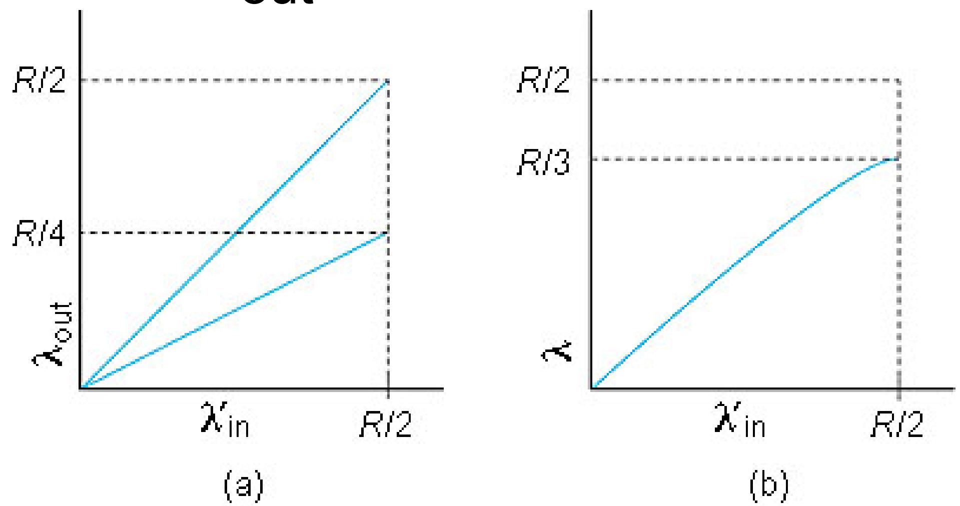
Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet



Causes/costs of congestion: scenario 2

- Ideally, no loss: $\lambda_{in} = \lambda_{out}$ (top line in fig. (a) below)
- "perfect" retransmission, only when loss: $\lambda'_{in} > \lambda_{out}$ (fig. (b) e.g. 1/3 of bytes retransmitted, so 2/3 original received, per host)
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out} (fig. (a) second line e.g.)



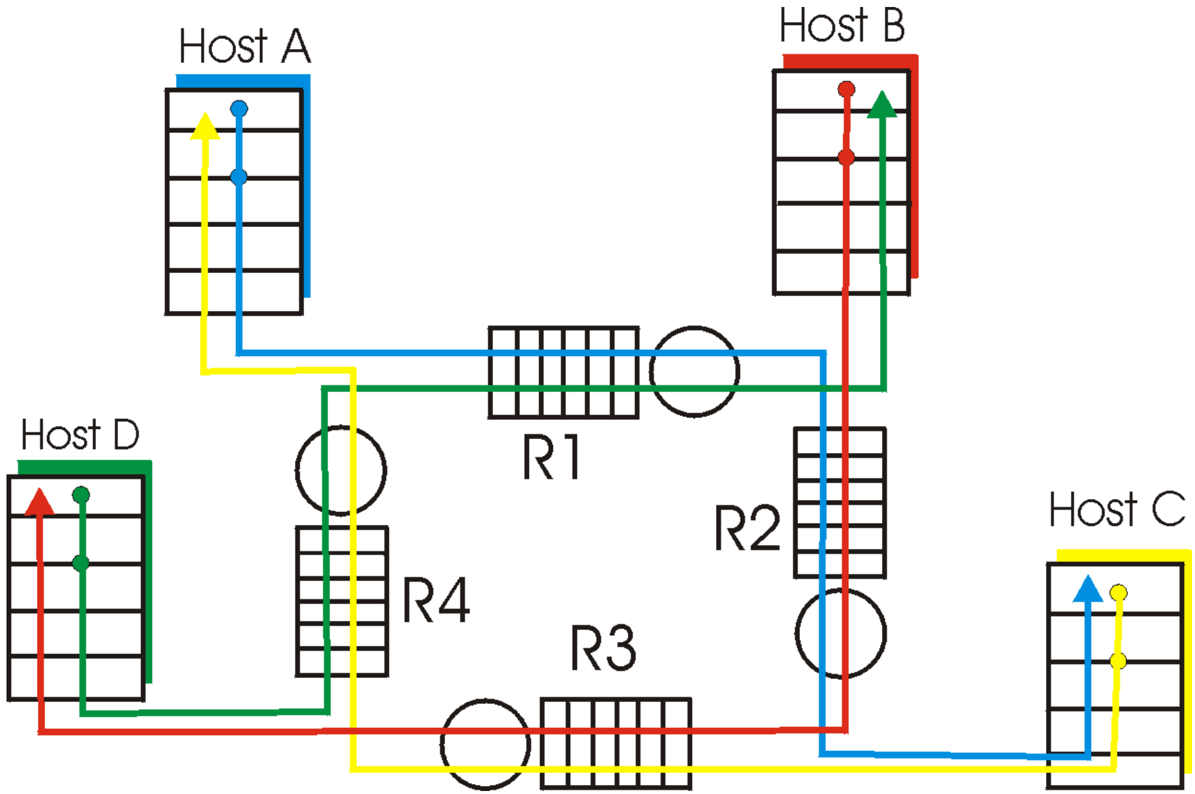
"costs" of congestion:

- more work (retrans) of original data (goodput)
- unneeded retransmissions: link carries multiple copies of pkt

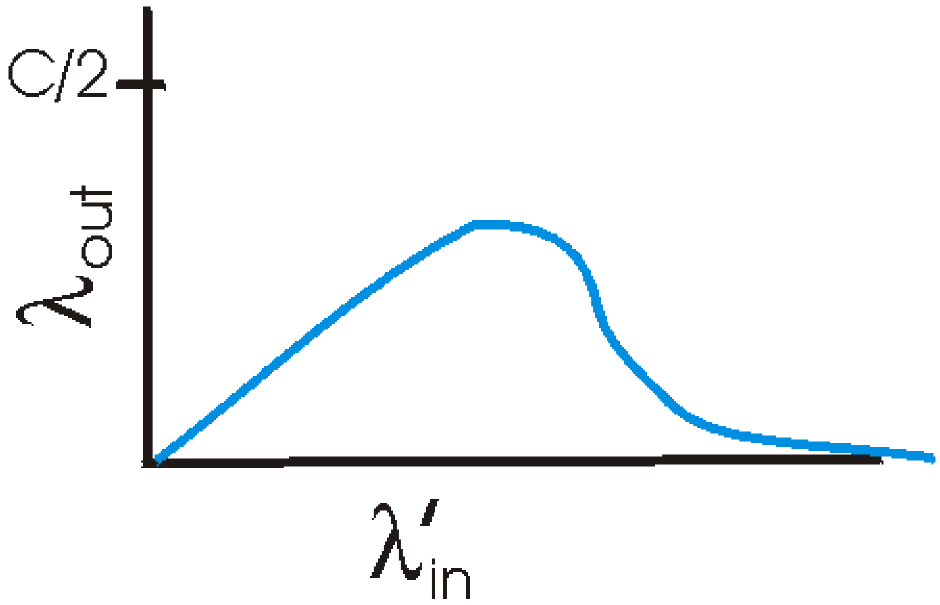
Causes/costs of congestion: scenario 3

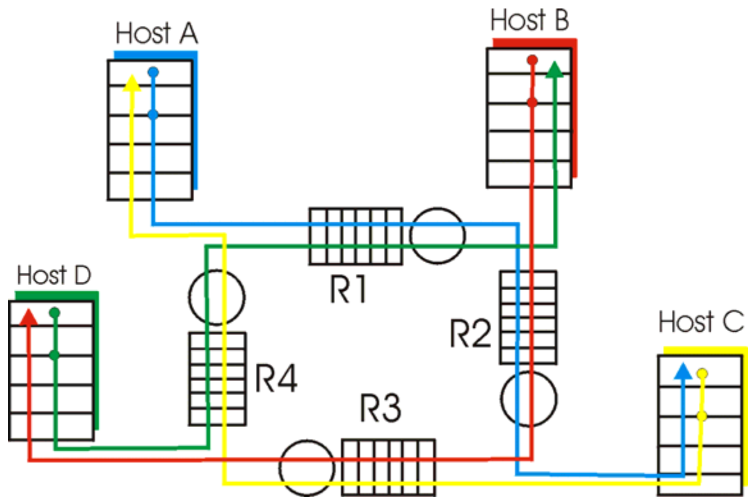
- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ'_{in} increase ?



Causes/costs of congestion: scenario 3





Another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

Approaches towards congestion control

Two broad approaches towards congestion control:

End-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP (IP provides no feedback)

Network-assisted congestion control:

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

Case study: ATM ABR congestion control

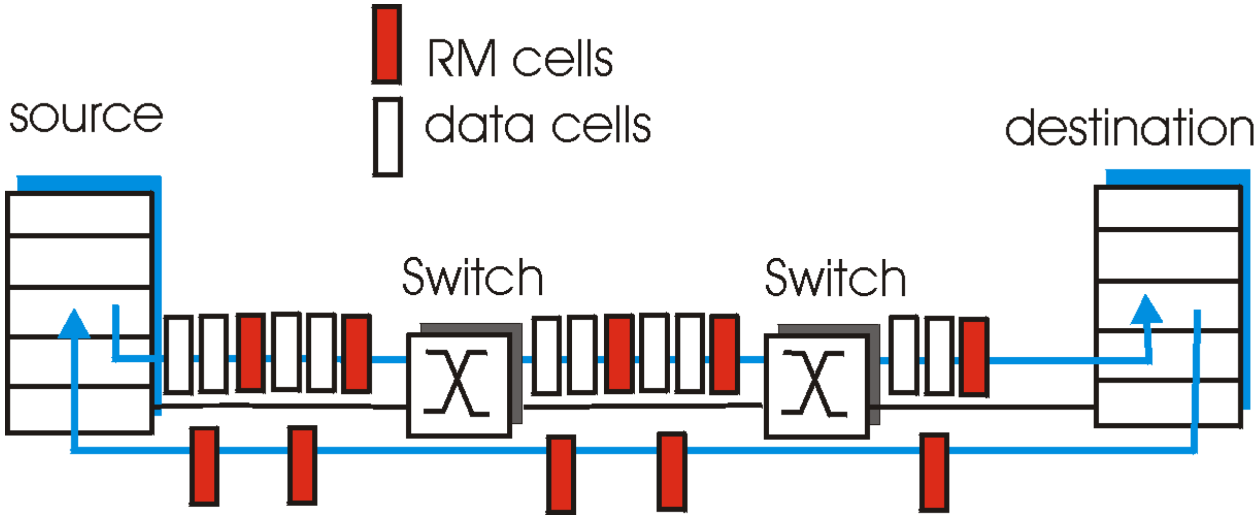
ABR: available bit rate:

- "elastic service"
- if sender's path "underloaded":
  - sender should use available bandwidth
- if sender's path congested:
  - sender throttled to minimum guaranteed rate

RM (resource management) cells:

- sent by sender, interspersed with data cells
- bits in RM cell set by switches ("*network-assisted*")
  - NI bit: no increase in rate (mild congestion)
  - CI bit: congestion indication
- RM cells returned to sender by receiver, with bits intact
- (note: switch can also generate RM cell)

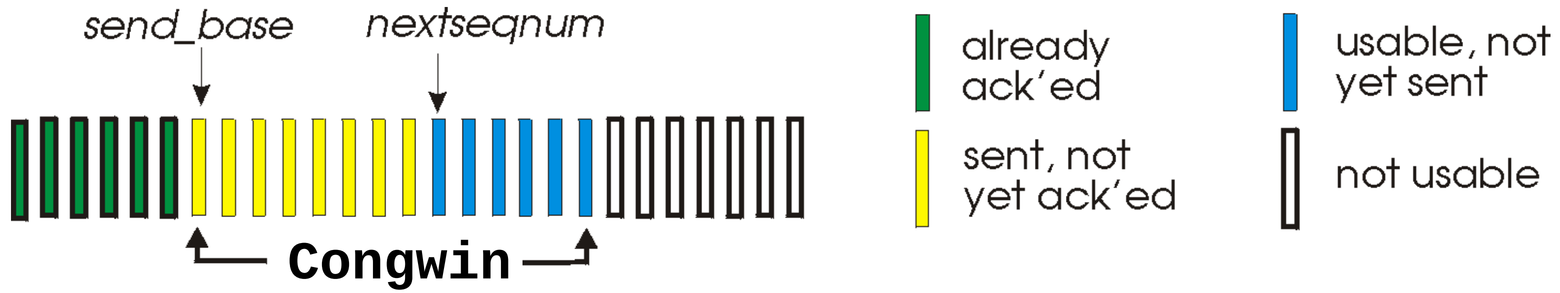
Case study: ATM ABR congestion control



- two-byte ER (explicit rate) field in RM cell
  - congested switch may lower ER value in cell
  - sender's send rate thus minimum supportable rate of all switches on path
- EFCI (explicit forward congestion indication) bit in data cells: set to 1 in congested switch
  - if data cell preceding RM cell has EFCI set, receiver sets CI bit in returned RM cell

TCP Congestion Control

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



- w segments, each with MSS bytes sent in one RTT:

$$\text{throughput} = \frac{w * MSS}{RTT} \text{ Bytes/sec}$$

TCP congestion control:

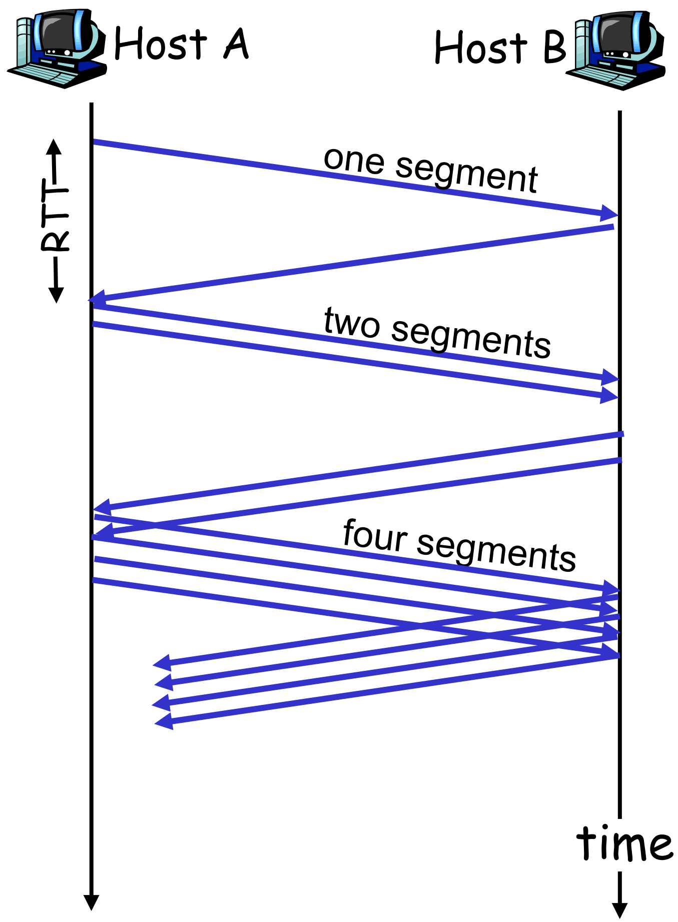
- "probing" for usable bandwidth:
  - ideally: transmit as fast as possible (**Congwin** as large as possible) without loss
  - *increase* **Congwin** until loss (congestion)
  - loss: *decrease* **Congwin**, then begin probing (increasing) again
- two "phases"
  - slow start
  - congestion avoidance
- important variables:
  - **Congwin**
  - **threshold:** defines threshold between slow start phase and congestion avoidance phase

TCP Slowstart

Slowstart algorithm

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

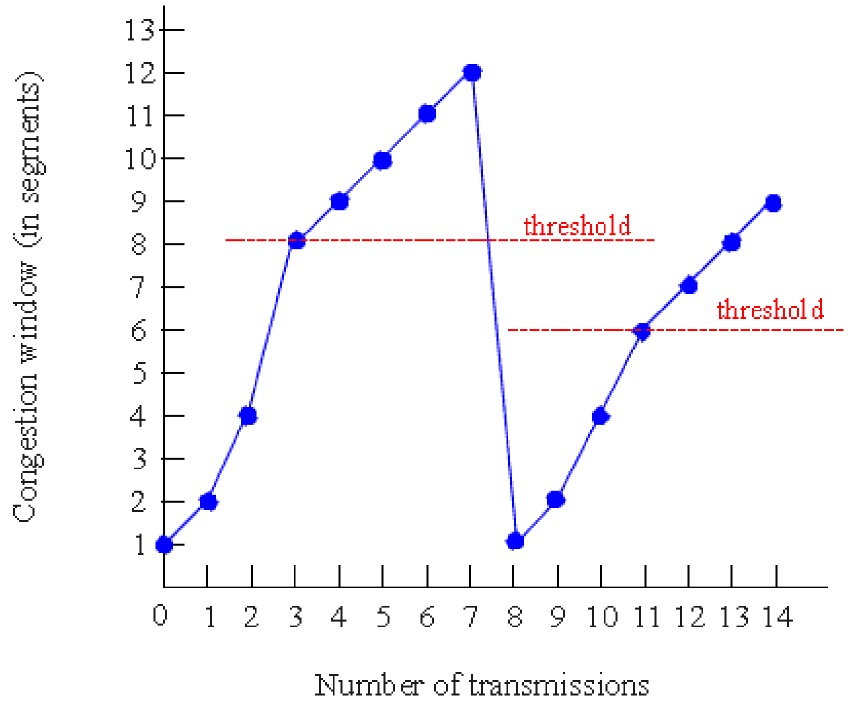
- exponential increase (per RTT) in window size (not so slow!)
- loss event: timeout (Tahoe TCP) and/or three duplicate ACKs (Reno TCP)



TCP Congestion Avoidance

Congestion avoidance

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



1: TCP Reno skips slowstart (fast recovery) after three duplicate ACKs

AIMD (*additive increase, multiplicative decrease*)

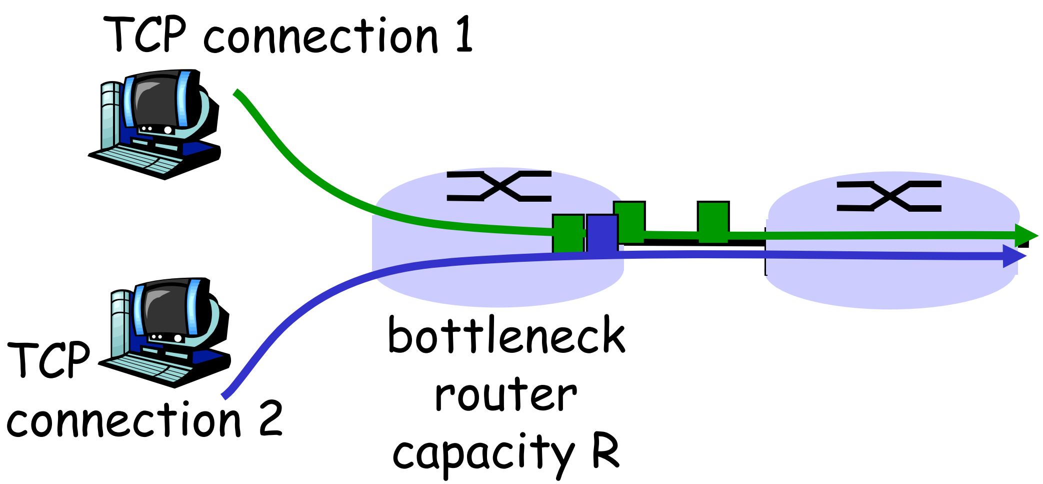
TCP congestion avoidance

(ignoring slow start):

- AIMD:
  - increase window by 1 per RTT
  - decrease window by factor of 2 on loss event

TCP Fairness

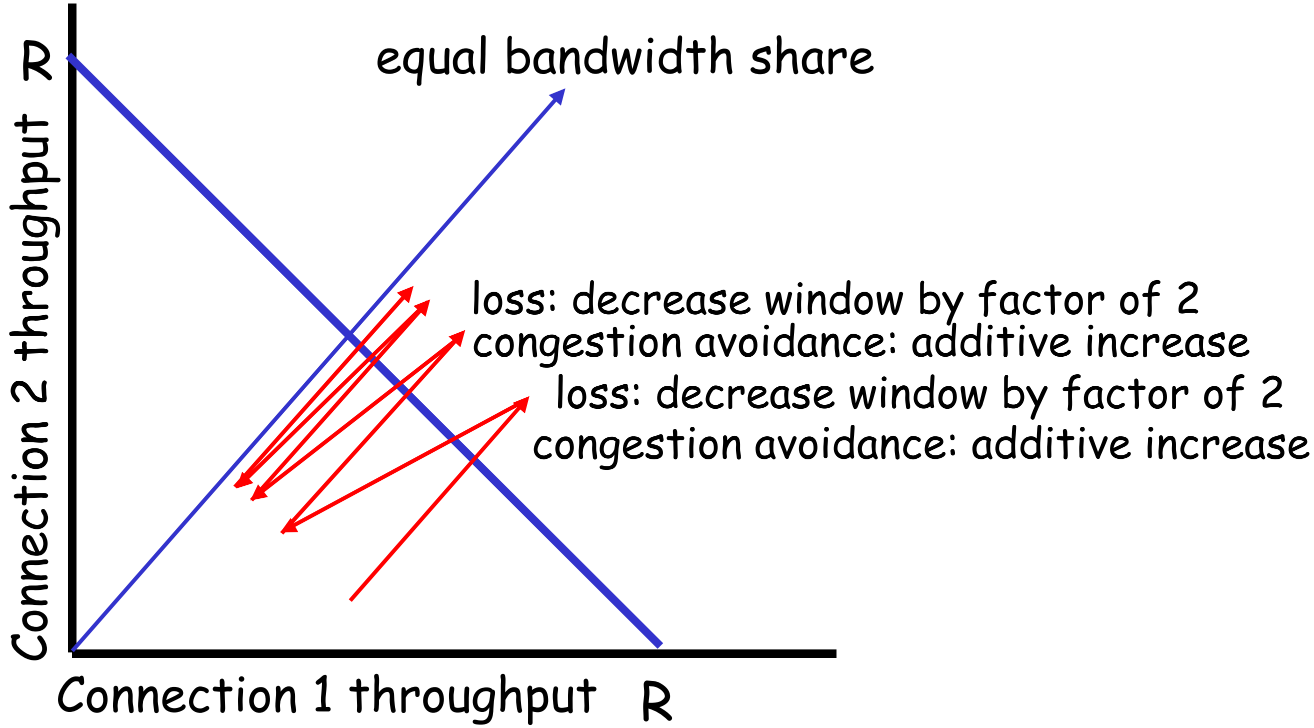
Fairness goal: if N TCP sessions share same bottleneck link, each should get 1/N of link capacity



Why is TCP fair?

Assume two competing sessions (same MSS and RTT):

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Effects of TCP latencies

Q: client latency from object request from WWW server to receipt?

- TCP connection establishment
- data transfer delay

Notation, assumptions:

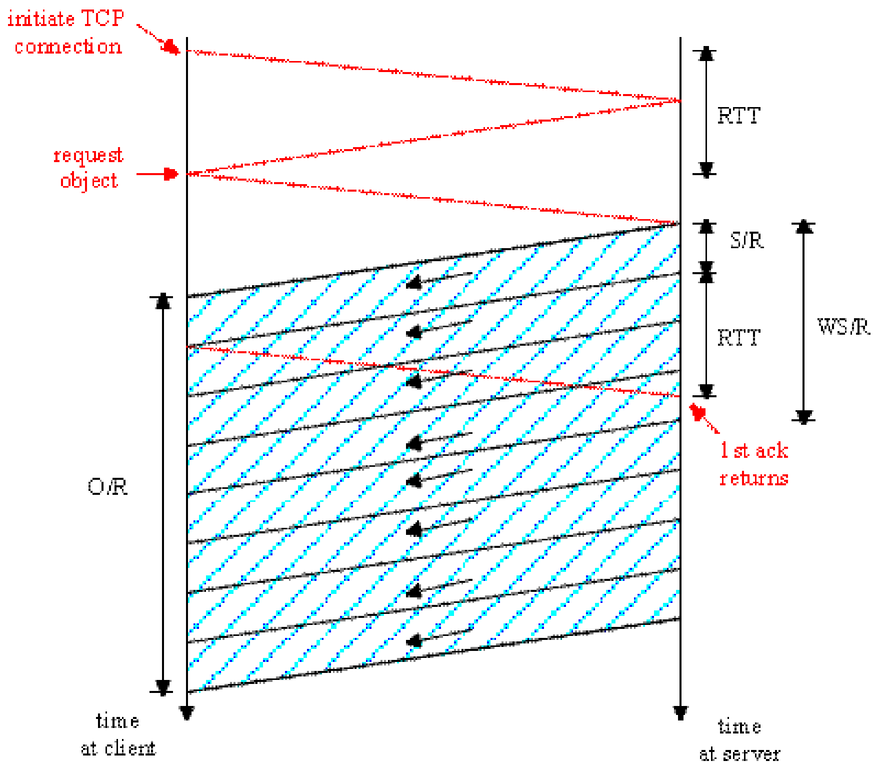
- Assume: fixed congestion window, W, giving throughput of R bps
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)

Two cases to consider:

- WS/R > RTT + S/R: server receives ACK for first segment in window before window's worth of data sent
- WS/R < RTT + S/R: server must wait for ACK after sending window's worth of data sent

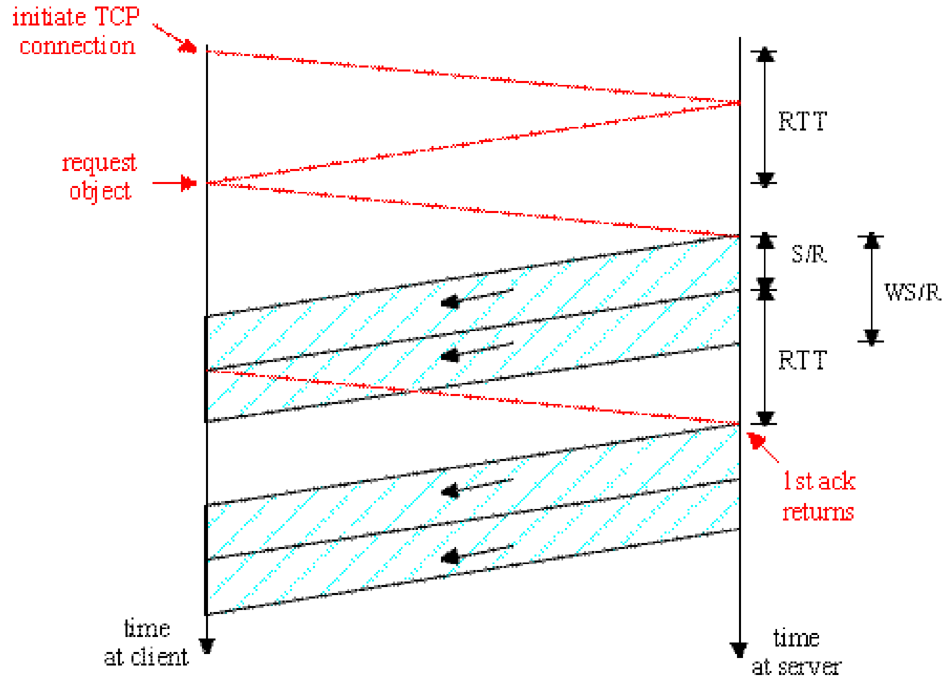
Effects of TCP latencies

W=4, WS/R > RTT + S/R



Case 1: latency = 2RTT + O/R

W=2, WS/R < RTT + S/R



Case 2: latency = 2RTT + O/R + (K-1)[S/R + RTT - WS/R]

(O is num bits in entire object)

K = O/(WS)

Summary on Transport Layer

- principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation and implementation in the Internet
  - UDP
  - TCP

Next:

- leaving the network "edge" (application transport layer)
- into the network "core"

Network Layer

Goals:

- understand principles behind network layer services:
  - routing (path selection)
  - dealing with scale
  - how a router works
  - advanced topics: IPv6, multicast
- instantiation and implementation in the Internet

Overview:

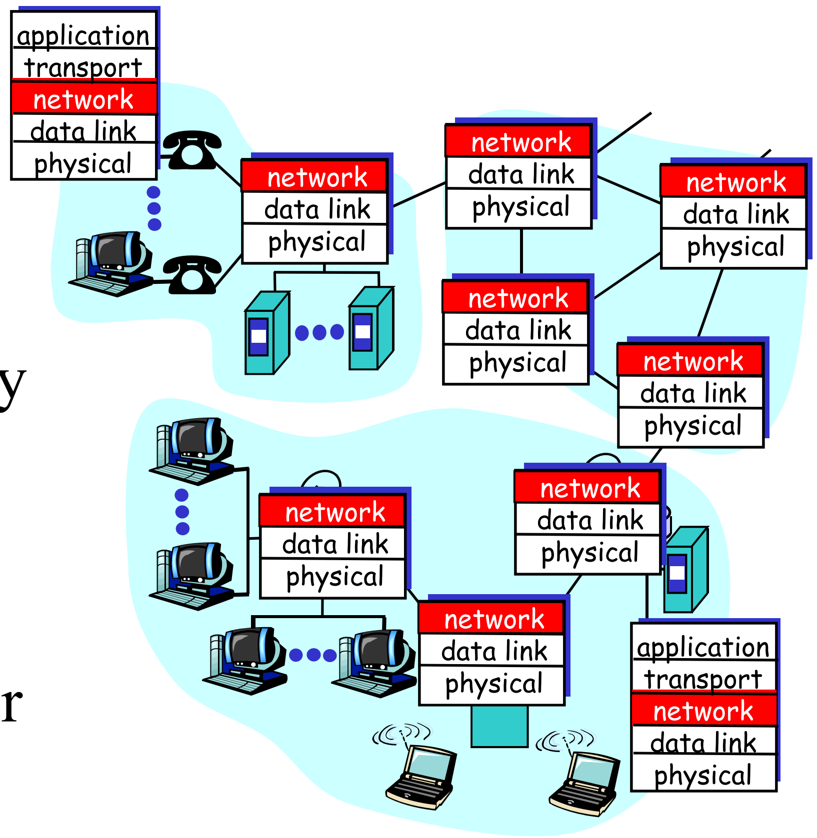
- network layer services
- routing principle: path selection
- hierarchical routing
- IP
- Internet routing protocols reliable transfer
  - intra-domain
  - inter-domain
- what's inside a router?
- IPv6
- multicast routing

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



Network service model

Q: What *service model* for "channel" transporting packets from sender to receiver?

service abstraction

- guaranteed bandwidth?
- preservation of inter-packet timing (no jitter)?
- loss-free delivery?
- in-order delivery?
- congestion feedback to sender?

The most important abstraction provided by network layer:

virtual circuit
or
datagram?

Virtual circuits

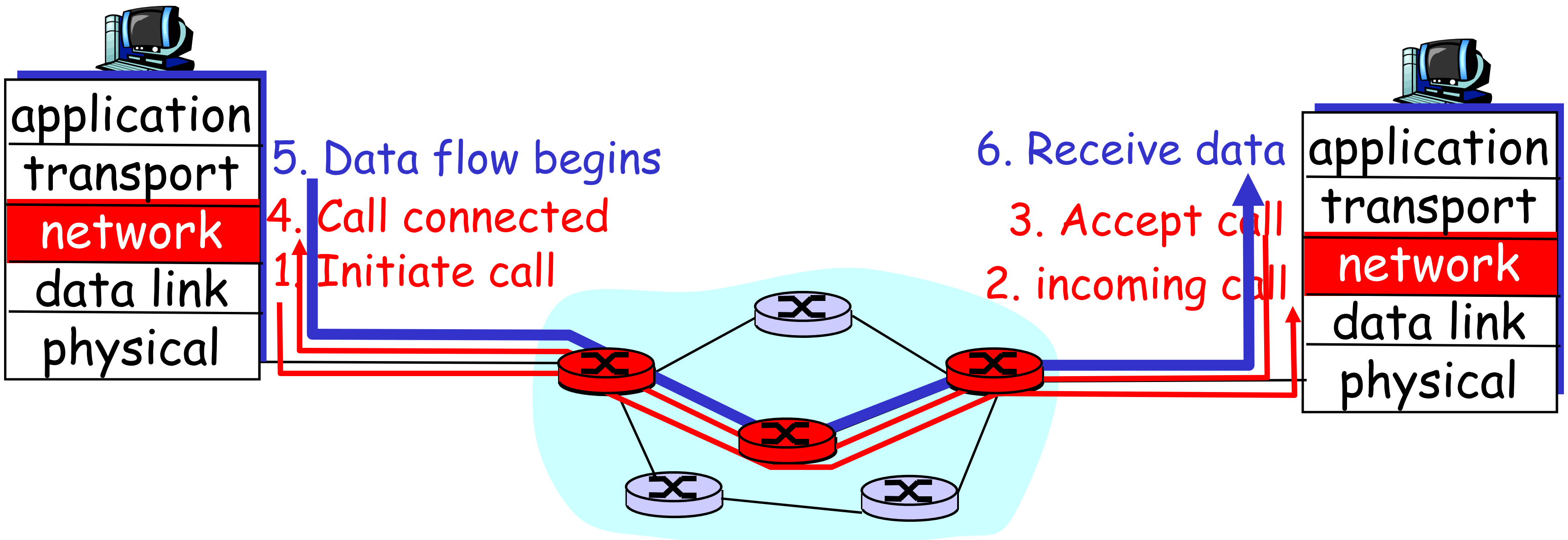
"source-to-dest path behaves much like telephone circuit"

- performance-wise
- network actions along source-to-dest path

- call setup, teardown for each call *before* data can flow
- each packet carries VC identifier (not destination host ID)
- *every* router on source-dest path maintains "state" for each passing connection
  - (in contrast, transport-layer connection only involved two end systems)
- link, router resources (bandwidth, buffers) may be *allocated* to VC
  - to get circuit-like performance

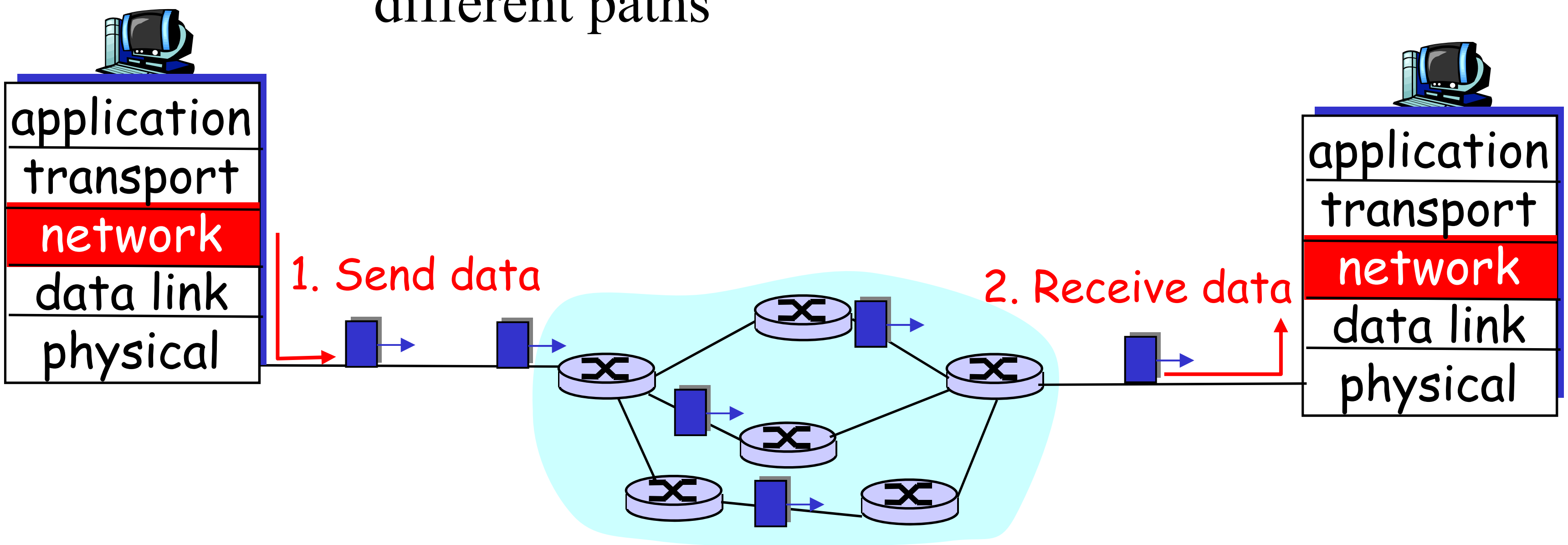
Virtual circuits: signaling protocols

- used to set up, maintain, and tear down VC
- used in ATM, frame-relay, X.25
- not used in today's Internet



Datagram networks: the Internet model

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of "connection"
- packets typically routed using destination host ID
  - packets between same source-dest pair may take different paths



Network layer service models:

Network Architecture	Service Model	Guarantees ? Bandwidth	Loss	Order	Timing	Congestion feedback
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

- Internet model being extended: Intserv, Diffserv
  - Chapter 6

Datagram or VC network: why?

Internet

- data exchange among computers
  - "elastic" service, no strict timing req.
- "smart" end systems (computers)
  - can adapt, perform control, error recovery
  - simple inside network, complexity at "edge"
- easier to connect many link types
  - different characteristics
  - uniform service difficult

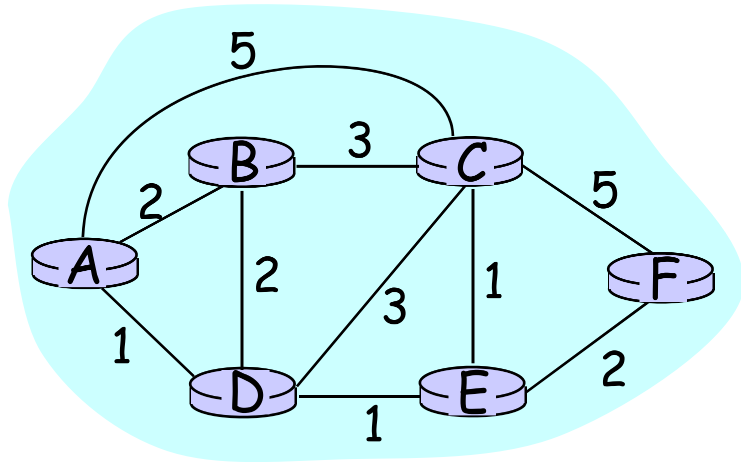
ATM

- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- "dumb" end systems
  - telephones
  - complexity inside network

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 definitions possible

Routing Algorithm classification

Global or decentralized information?

Global:

- all routers have complete topology, link cost info
- "link state" algorithms

Decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Static or dynamic?

Static:

- routes change slowly over time (usually by humans)

Dynamic:

- routes change more quickly/automatically
  - periodic update
  - in response to link cost changes

A Link-State Routing Algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives routing table for that node
- iterative: after k iterations, know least cost path to k destinations

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
- p(v): predecessor node along path from source to v, that is next v
- N: set of nodes whose least cost path definitively known

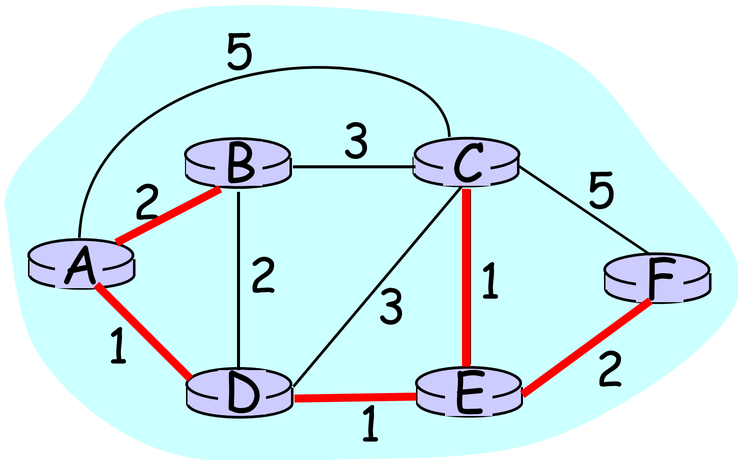
Dijsktra's Algorithm

```
Initialization:
  N = {A}
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v)
      else D(v) = infty

 Loop
   find w not in N such that D(w) is a minimum (of nodes adjacent to previous w)
    add w to N
    update D(v) for all v adjacent to w and not in N:
       D(v) = min( D(v), D(w) + c(w,v) )
    /* new cost to v is either old cost to v or known
     shortest path cost to w plus cost from w to v */
 until all nodes in N
```

Dijkstra's algorithm: example

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					



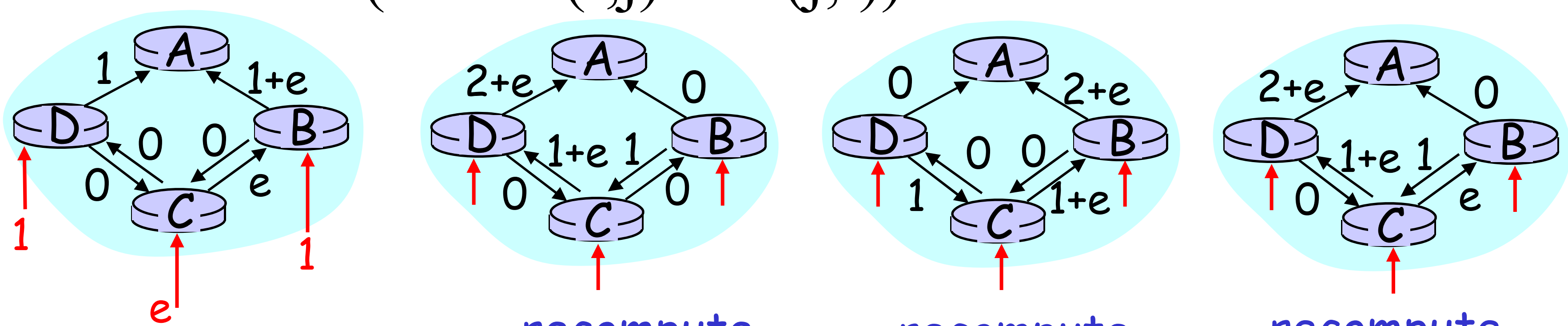
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- n*(n+1)/2 comparisons: O(n**2)
- more efficient implementations possible: O(nlogn)

Oscillations possible:

- e.g., Suppose link cost = amount of carried traffic
  (note: c(i,j) != c(j,i))



initially … recompute routing … recompute … recompute

Distance Vector Routing Algorithm

iterative:

- continues until no nodes exchange info.
- *self-terminating*: no "signal" to stop

asynchronous:

- nodes need *not* exchange info/iterate in lock step!

distributed:

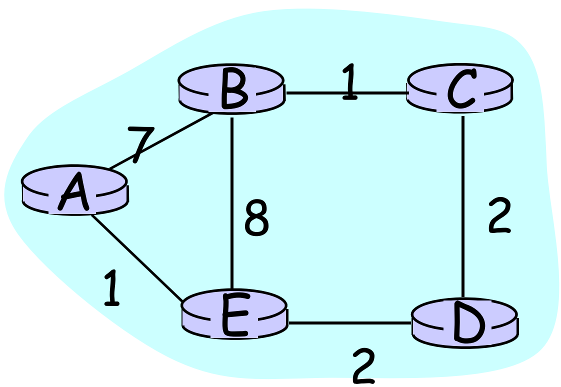
- each node communicates *only* with directly-attached neighbors

Distance Table data structure

- each node has its own
- row for each possible destination
- column for each directly-attached neighbor node
- example: in node X, for dest. Y via neighbor Z:

$$D^X(Y,Z) = \text{distance } \textit{from} \text{ X } \textit{to} \text{ Y, } \textit{via} \text{ Z as next hop} = c(X,Z) + \min_w \{D^Z(Y,w)\}$$

Distance Table: example



$$D^E(C,D) = c(E,D) + \min_w \{D^D(C,w)\}$$
$$= 2+2 = 4$$

$$D^E(A,D) = c(E,D) + \min_w \{D^D(A,w)\}$$
$$= 2+3 = 5$$

loop!

$$D^E(A,B) = c(E,B) + \min_w \{D^B(A,w)\}$$
$$= 8+6 = 14$$

loop!

cost to destination via

$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destination

Distance table gives routing table

cost to destination via

$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destination

	Outgoing link to use, cost
A	A,1
B	D,5
C	D,4
D	D,4

destination

Distance table → Routing table

Distance Vector Routing: overview

Iterative, asynchronous: each local iteration caused by:

- local link cost change
- message from neighbor: its least cost path change from neighbor

Distributed:

- each node notifies neighbors *only* when its least cost path to any destination changes
  - neighbors then notify their neighbors if necessary

Each node:

wait for (change in local link cost of msg from neighbor)

recompute distance table

if least cost path to any dest has changed, *notify* neighbors

Distance Vector Algorithm:

At all nodes, X:

```
Initialization:
 for all adjacent nodes v:
    D^X(*,v) = infinity        /* the * operator means "for all rows" */
    D^X(v,v) = c(X,v)
 for all destinations, y
    send min_w D^X(y,w) to each neighbor  /* w over all X's neighbors */
```

Distance Vector Algorithm (cont.):

loop

  wait (until I see a link cost change to neighbor V

        or until I receive update from neighbor V)

  if (c(X,V) changes by d)

    /* change cost to all dest's via neighbor v by d */

    /* note: d could be positive or negative */

    for all destinations y: $D^X(y,V) = D^X(y,V) + d$

  else if (update received from V wrt destination Y)

    /* shortest path from V to some Y has changed */

    /* V has sent a new value for its $\min_w DV(Y,w)$ */

    /* call this received new value is "newval" */

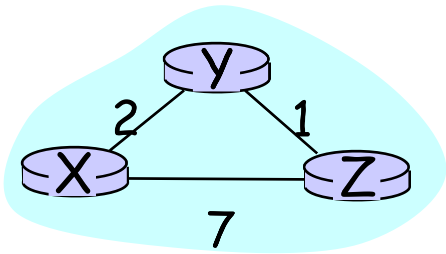
    for the single destination y: $D^X(Y,V) = c(X,V) + \text{newval}$

  if we have a new $\min_w D^X(Y,w)$for any destination Y

     send new value of $\min_w D^X(Y,w)$ to all neighbors

forever

Distance Vector Algorithm: example



D^X (dest)	cost via Y	cost via Z
Y	2	∞
Z	∞	7

D^Y (dest)	cost via X	cost via Z
X	2	∞
Z	∞	1

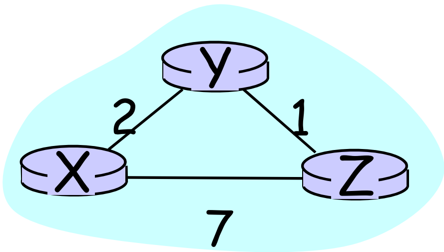
D^Z (dest)	cost via X	cost via Y
X	7	∞
Y	∞	1

D^X (dest)	cost via Y	cost via Z
Y	2	8
Z	3	7

$$D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\}$$
$$= 7+1 = 8$$

$$D^X(Z,Y) = c(X,Y) + \min_w\{D^Y(Z,w)\}$$
$$= 2+1 = 3$$

Distance Vector Algorithm: example



D^X (dest)	cost via Y	cost via Z
Y	2	∞
Z	∞	7

D^Y (dest)	cost via X	cost via Z
X	2	∞
Z	∞	1

D^Z (dest)	cost via X	cost via Y
X	7	∞
Y	∞	1

D^X (dest)	cost via Y	cost via Z
Y	2	8
Z	3	7

D^Y (dest)	cost via X	cost via Z
X	2	8
Z	9	1

D^Z (dest)	cost via X	cost via Y
X	7	3
Y	9	1

D^X (dest)	cost via Y	cost via Z
Y		
Z		

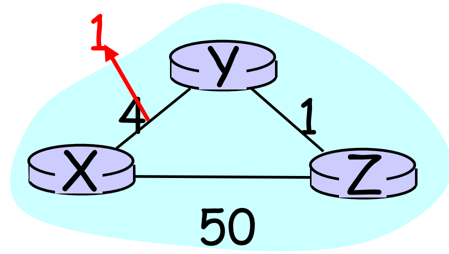
D^Y (dest)	cost via X	cost via Z
X		
Z		

D^Z (dest)	cost via X	cost via Y
X		
Y		

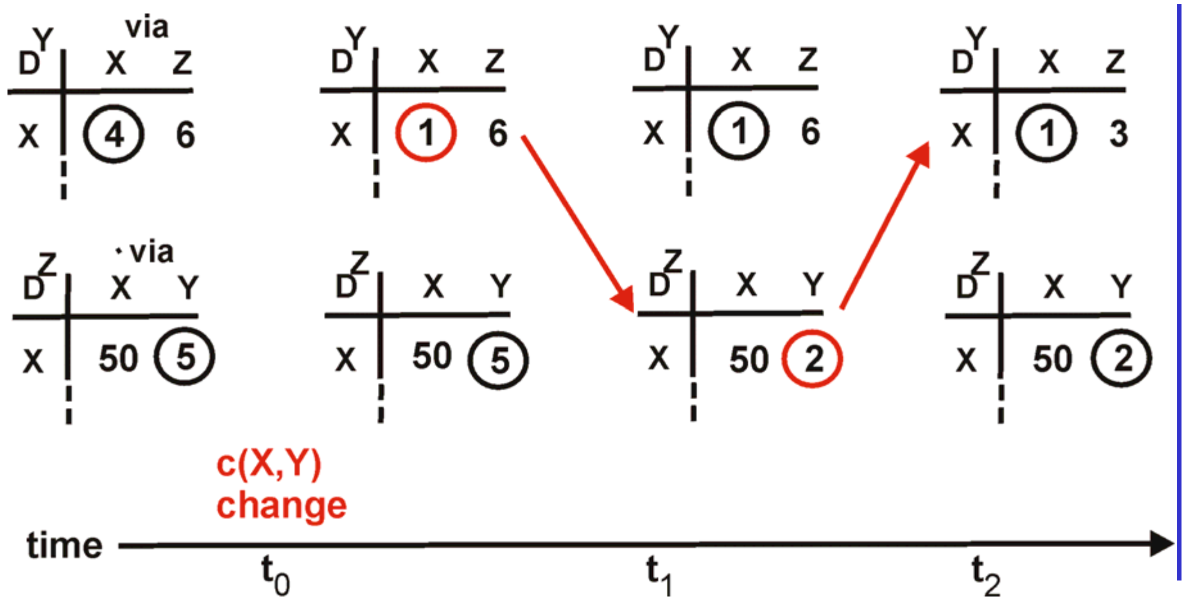
Distance Vector: link cost changes

Link cost changes:

- node detects local link cost change
- updates distance table (line 15)
- if cost change in least cost path, notify neighbors (lines 23,24)



"good news travels fast"

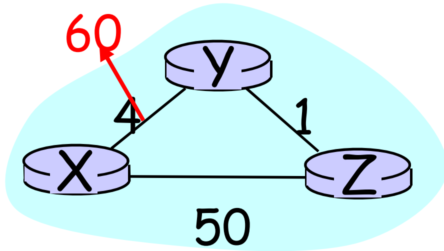


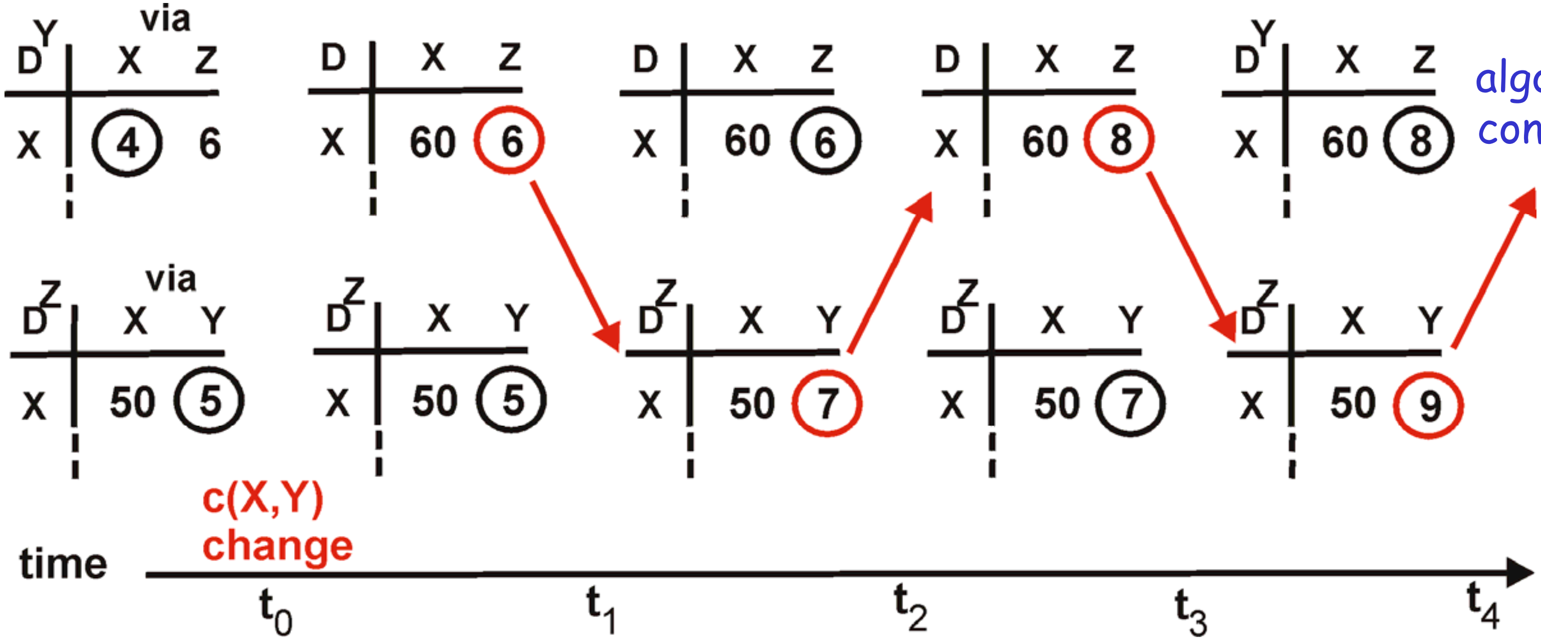
algorithm terminates

Distance Vector: link cost changes

Link cost changes:

- good news travels fast
- bad news travels slow - "count to infinity" problem!

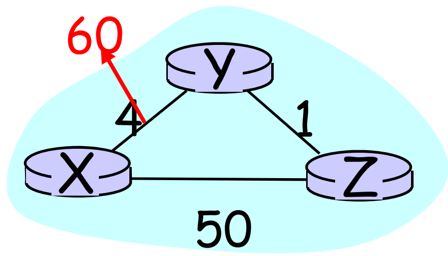


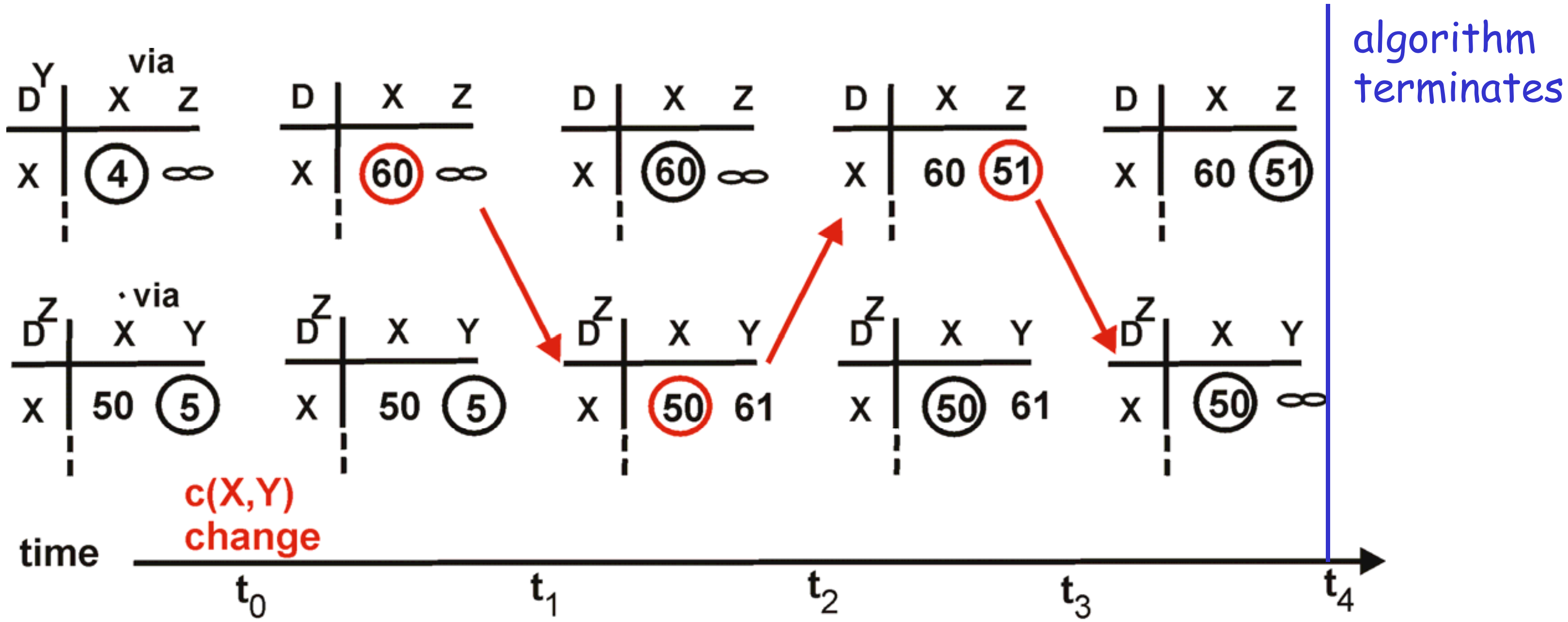


Distance Vector: poisoned reverse

If Z routes through Y to get to X :

- Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- will this completely solve count to infinity problem?





Comparison of LS and DV algorithms

Message complexity

- LS: with n nodes, E links, O(nE) msgs sent each
- DV: exchange between neighbors only
  - convergence time varies

Speed of Convergence

- LS: O(n**2) algorithm requires O(nE) msgs
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

Hierarchical Routing

Our routing study thus far - idealization

- all routers identical
- network "flat"

... *not* true in practice

scale: with 50 million destinations:

- can't store all dest's in routing tables!
- routing table exchange would swamp links!

administrative autonomy

- internet = network of networks
- each network admin may want to control routing in its own network

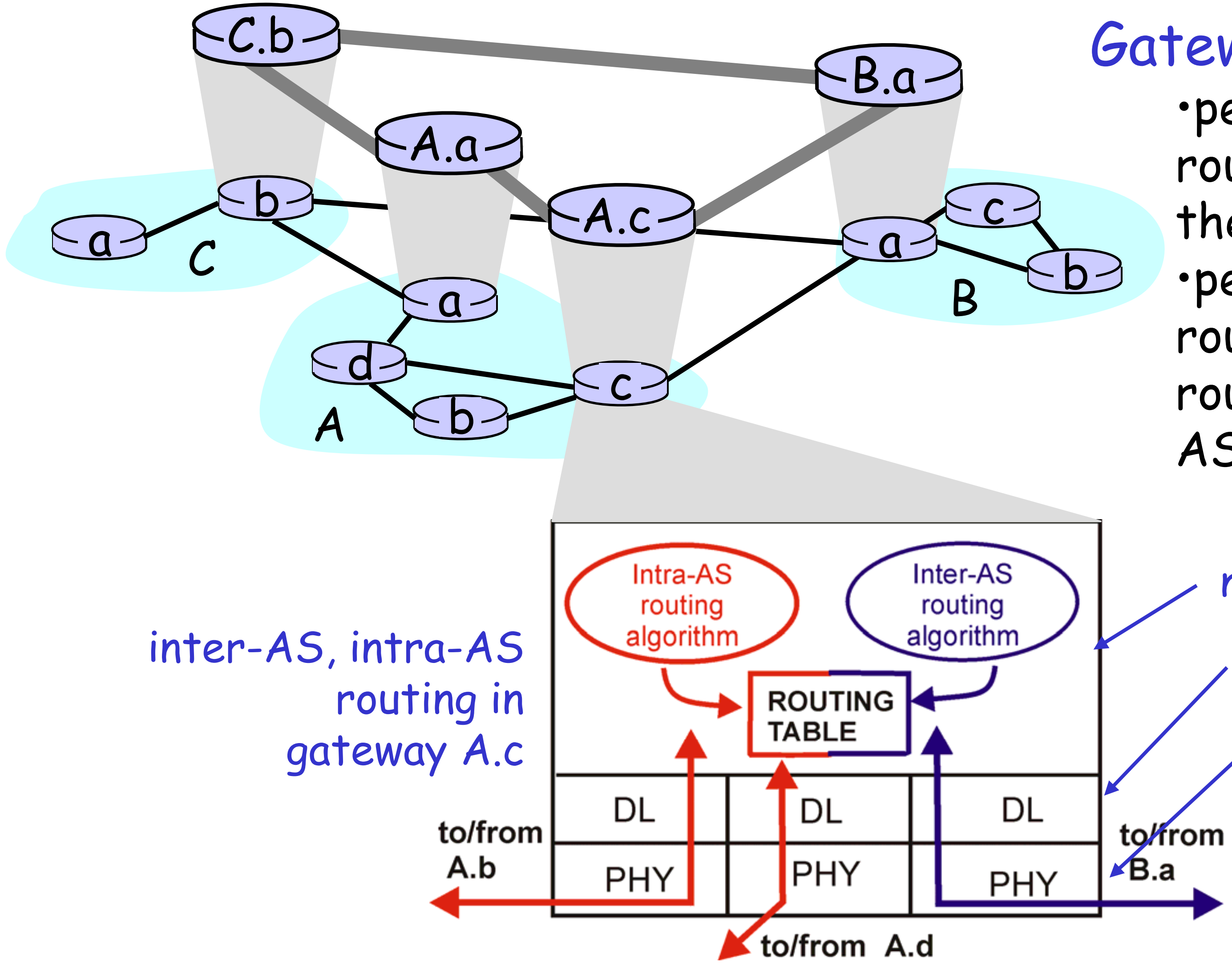
Hierarchical Routing

- aggregate routers into regions, "autonomous systems" (AS)
- routers in same AS run same routing protocol
  - "intra-AS" routing protocol
  - routers in different AS can run different intra-AS routing protocol

gateway routers

- special routers in AS
- run intra-AS routing protocol with all other routers in AS
- *also* responsible for routing to destinations outside AS
  - run *inter-AS routing* protocol with other gateway routers

Intra-AS and Inter-AS routing



Gateways:

- •perform inter-AS routing amongst themselves
- •perform intra-AS routers with other routers in their AS

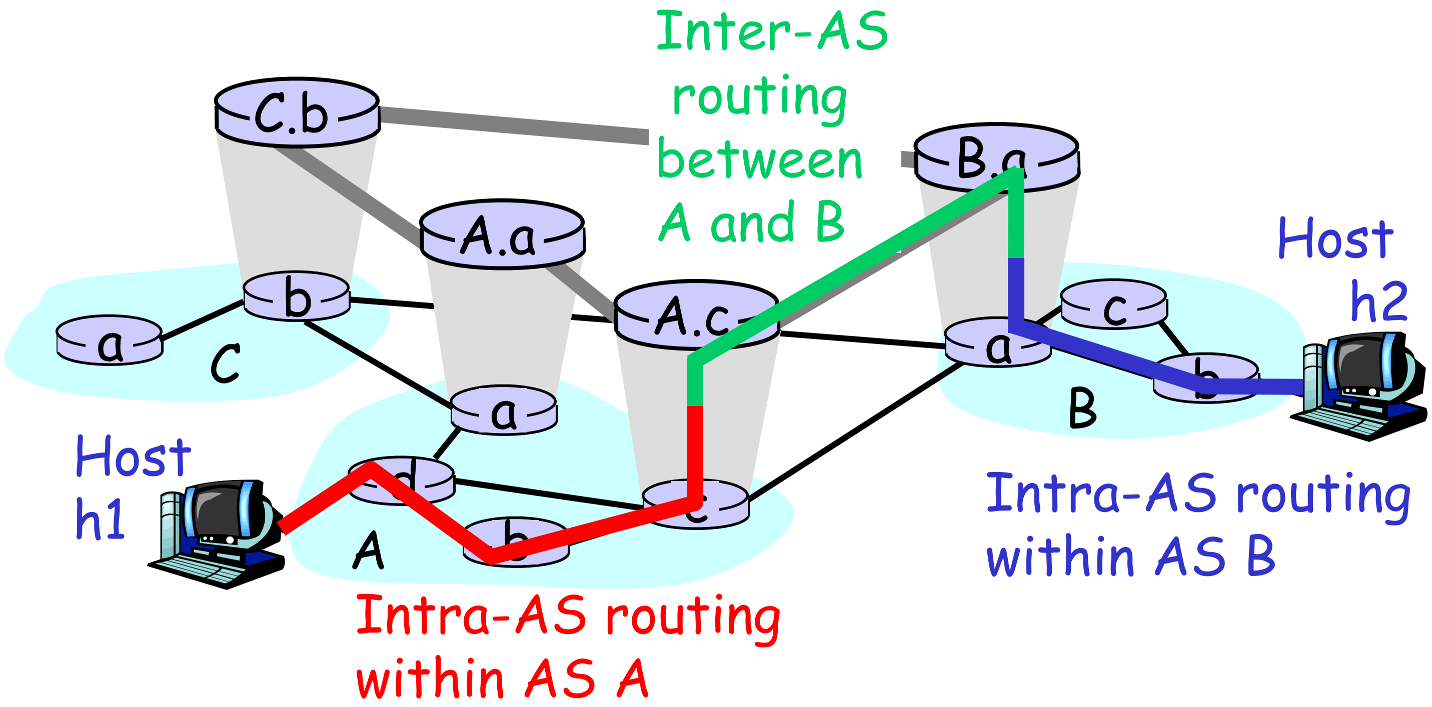
inter-AS, intra-AS routing in gateway A.c

network layer

link layer

physical layer

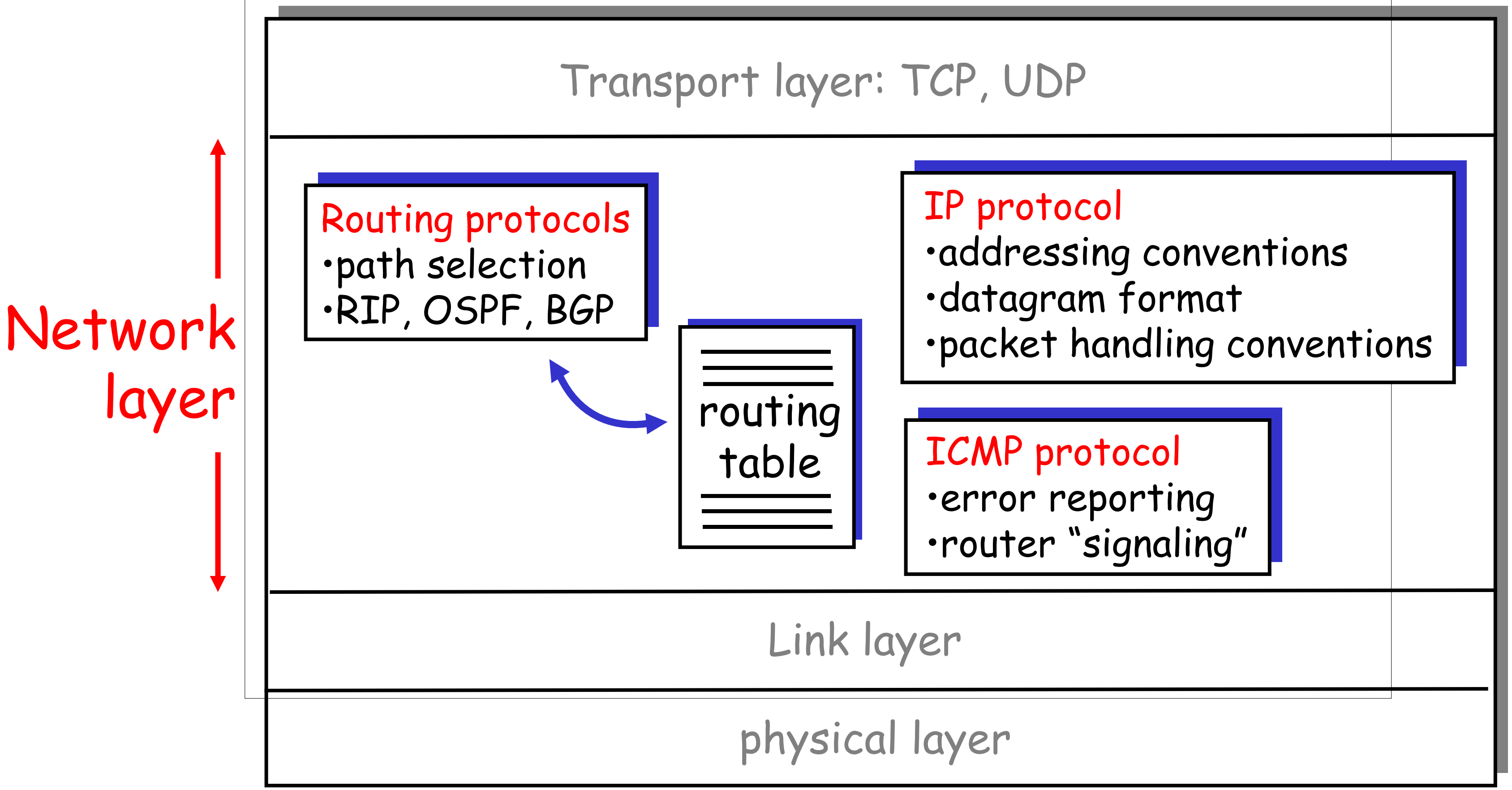
Intra-AS and Inter-AS routing



- We'll examine specific inter-AS and intra-AS Internet routing protocols shortly

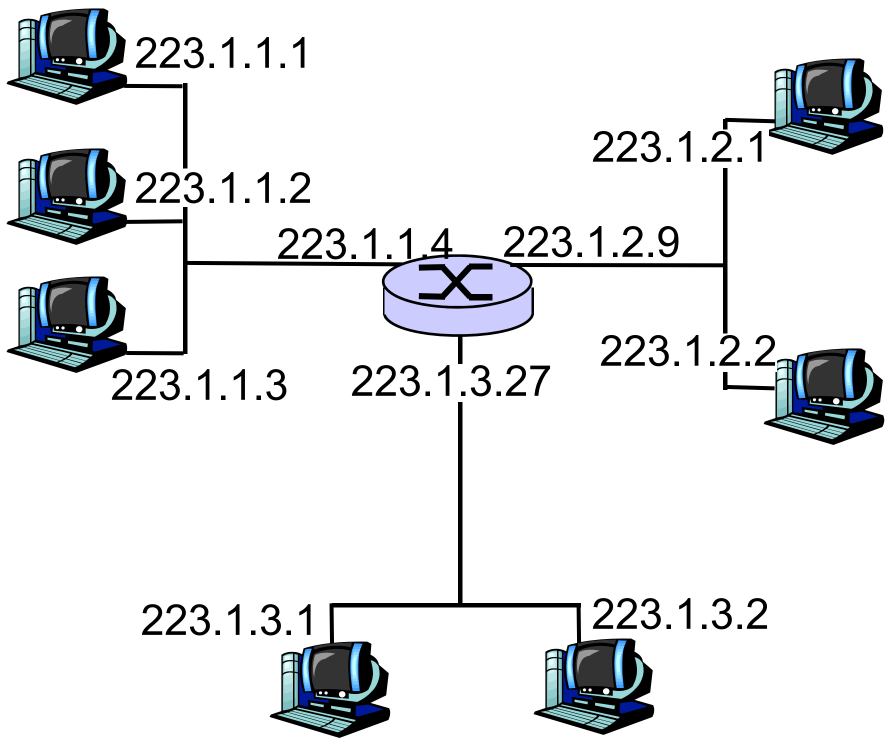
The Internet Network layer

Host, router network layer functions:



IP Addressing: introduction

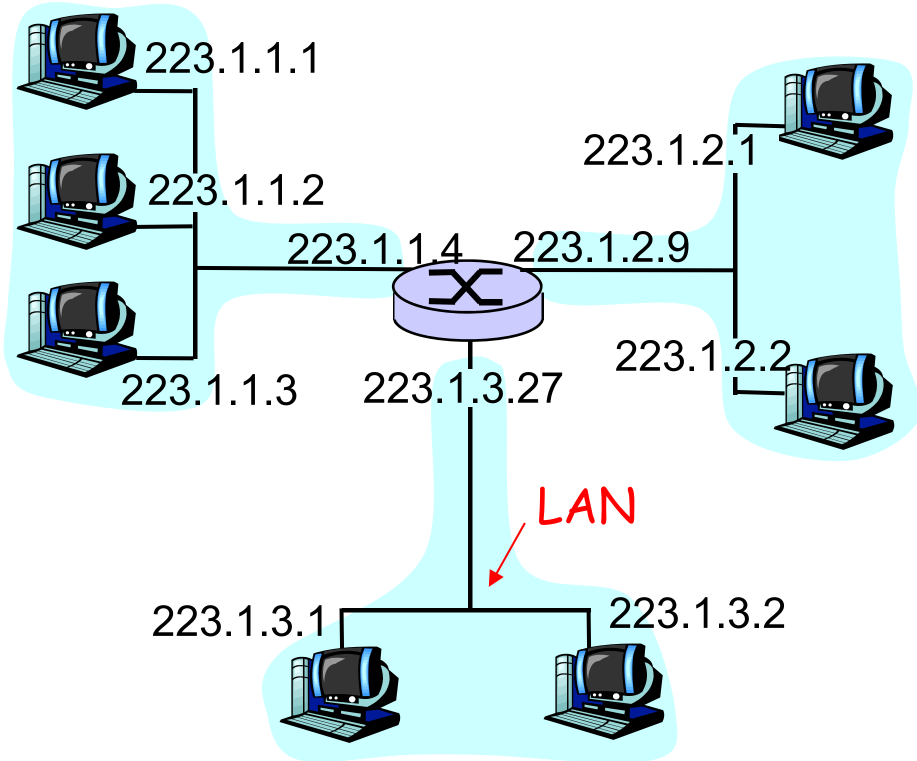
- **IP address:** 32-bit identifier for host, router *interface*
- *interface:* connection between host, router and physical link
  - routers typically have multiple interfaces
  - host may have multiple interfaces
  - IP addresses associated with interface, not host, router



$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

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

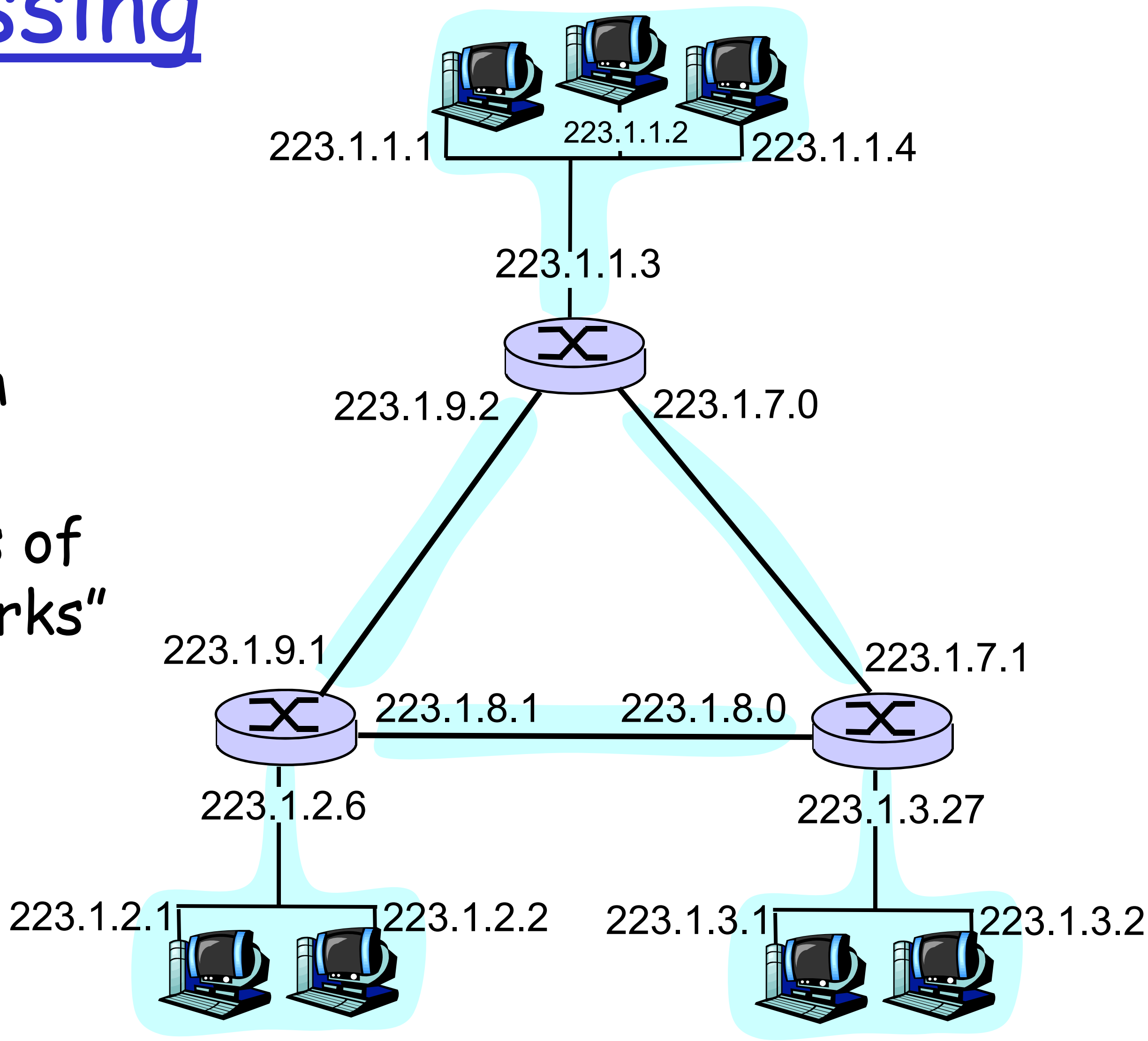


network consisting of 3 IP networks (for IP addresses starting with 223, first 24 bits are network address)

IP Addressing

How to find the networks?

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

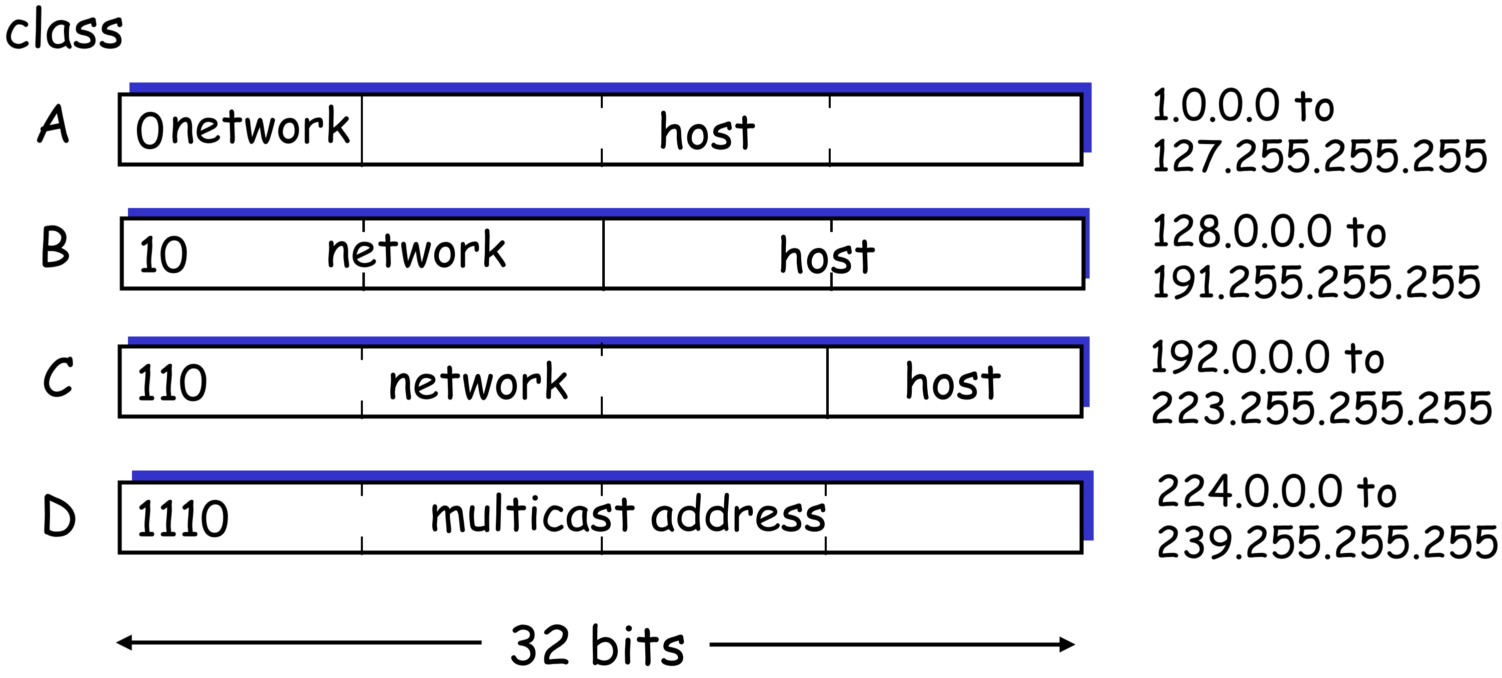


Interconnected system consisting of six networks

IP Addresses

given notion of "network", let's re-examine IP addresses:

"class-full" addressing:



IP addressing: CIDR

- classfull addressing:
  - inefficient use of address space, address space exhaustion
  - e.g., class B net allocated enough addresses for 65K hosts, even if only 2K hosts in that network
- CIDR: Classless InterDomain Routing
  - network portion of address of arbitrary length
  - address format: a.b.c.d/x, where x is # bits in network portion of address

network part | host part

11001000 00010111 00010000 00000000

200.23.16.0/23

IP addresses: how to get one?

Hosts (host portion):

- hard-coded by system admin in a file
- DHCP: Dynamic Host Configuration Protocol: dynamically get address: "plug-and-play"
  - host broadcasts "DHCP discover" msg
  - DHCP server responds with "DHCP offer" msg
  - host requests IP address: "DHCP request" msg
  - DHCP server sends address: "DHCP ack" msg

IP addresses: how to get one?

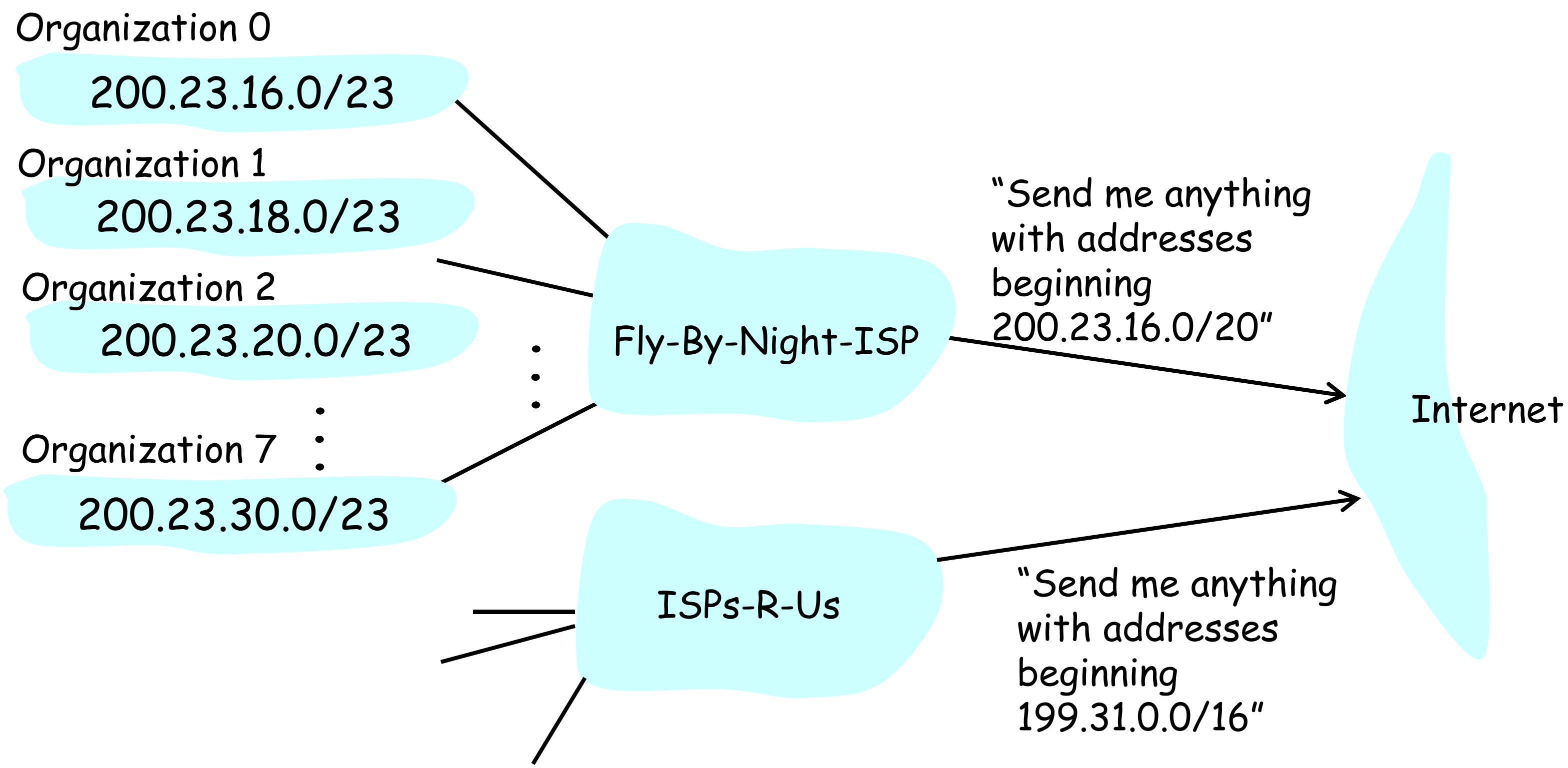
Network (network portion):

- get allocated portion of ISP's address space:

ISP's block	11001000	00010111	00010000	00000000	200.23.16.0/20
Organization 0	11001000	00010111	00010000	00000000	200.23.16.0/23
Organization 1	11001000	00010111	00010010	00000000	200.23.18.0/23
Organization 2	11001000	00010111	00010100	00000000	200.23.20.0/23
...		.....		....	....
Organization 7	11001000	00010111	00011110	00000000	200.23.30.0/23

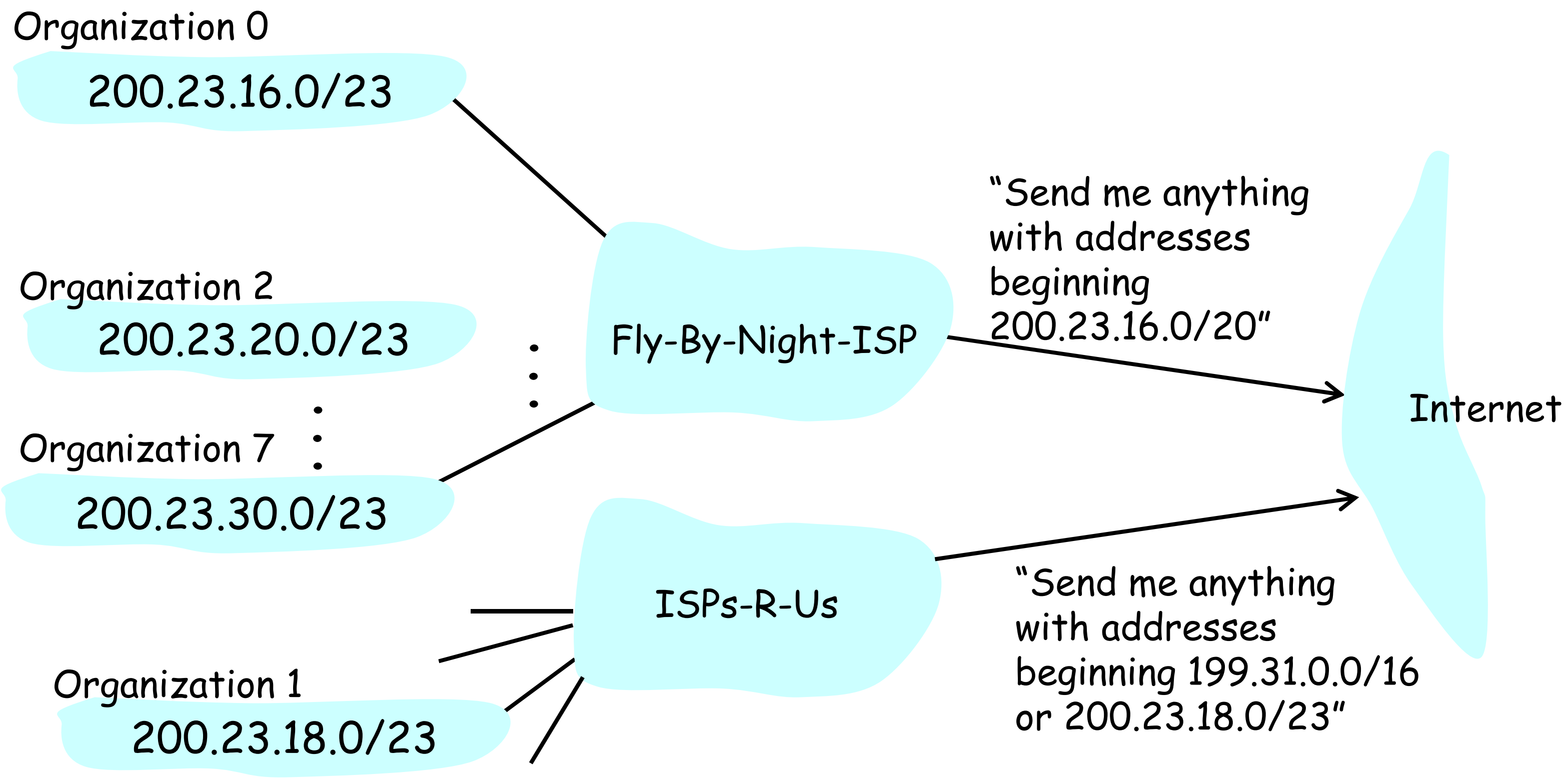
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1



IP addressing: the last word...

Q: How does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

Getting a datagram from source to dest.

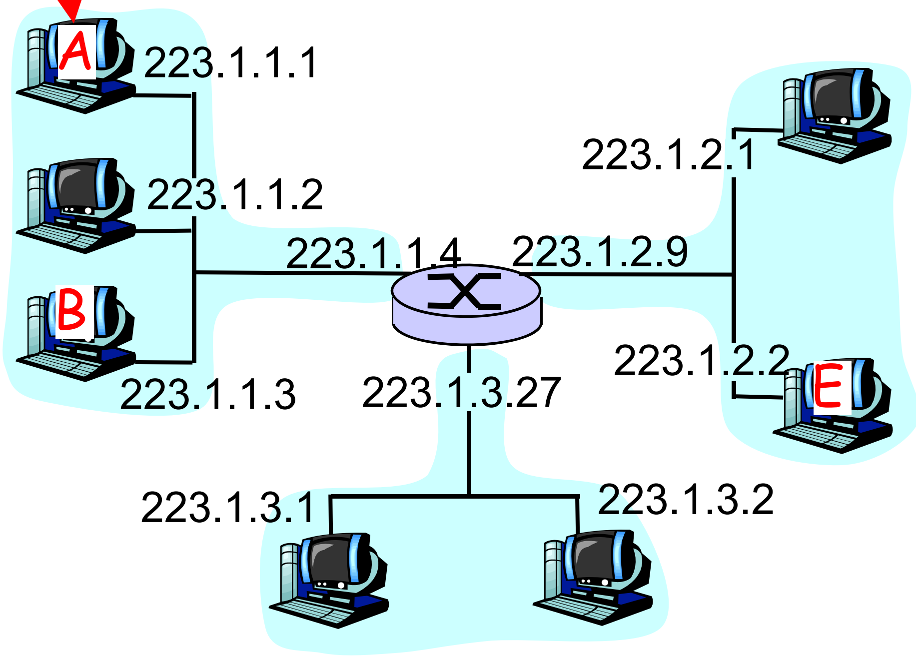
routing table in A

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

IP datagram:

misc fields	source IP addr	dest IP addr	data

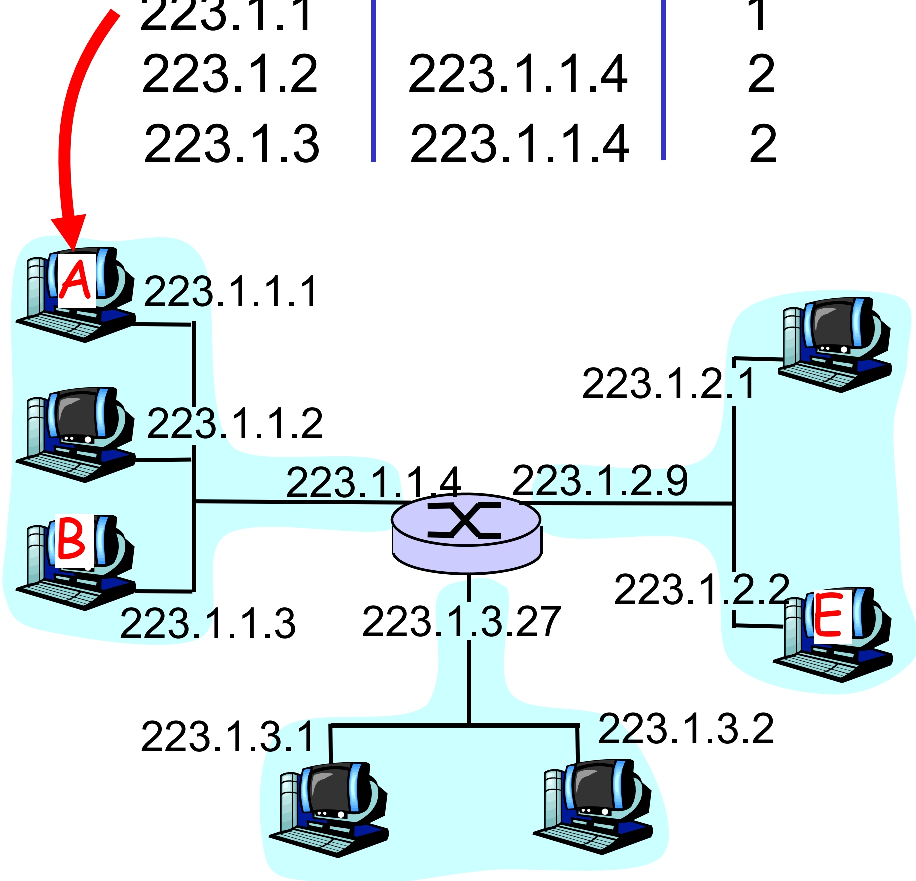
- datagram remains unchanged, as it travels source to destination
- addr fields of interest here



Getting a datagram from source to dest.

misc fields	223.1.1.1	223.1.1.3	data

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



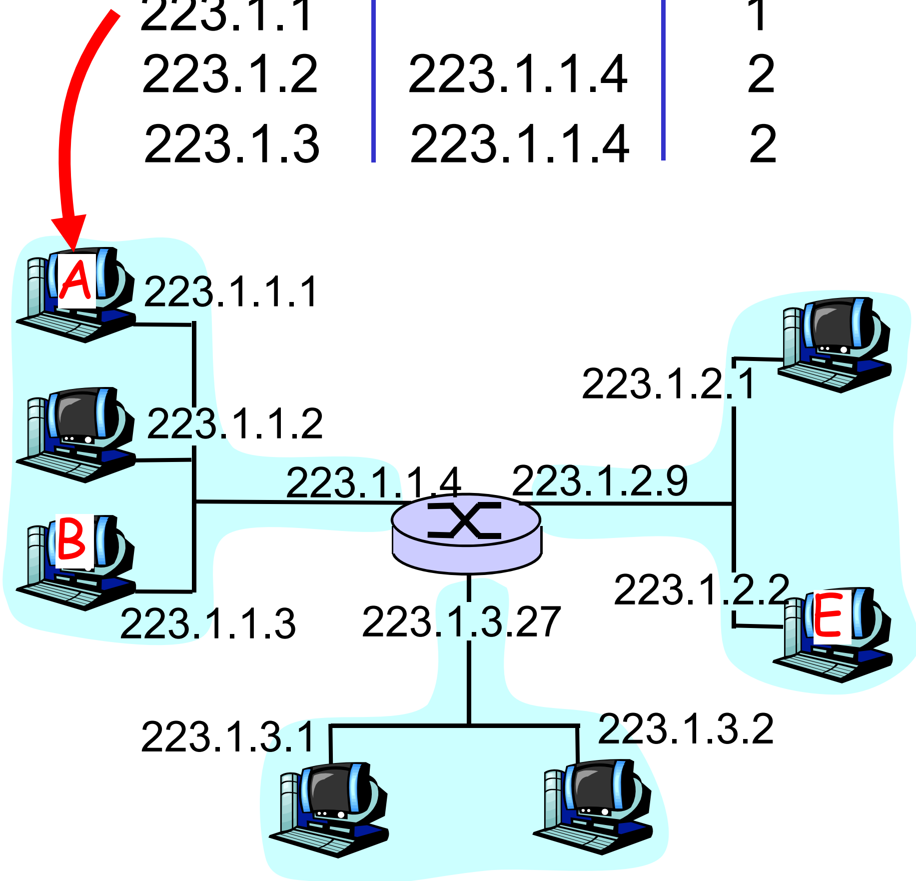
Starting at A, given IP datagram addressed to B:

- 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

Getting a datagram from source to dest.

misc fields	223.1.1.1	223.1.2.2	data

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



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…..

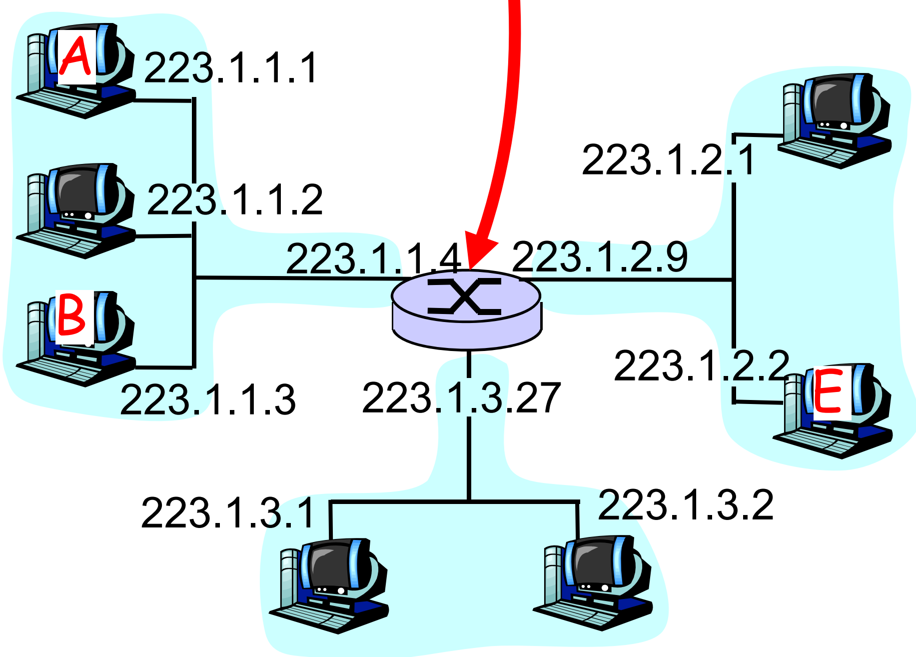
Getting a datagram from source to dest.

misc fields	223.1.1.1	223.1.2.2	data

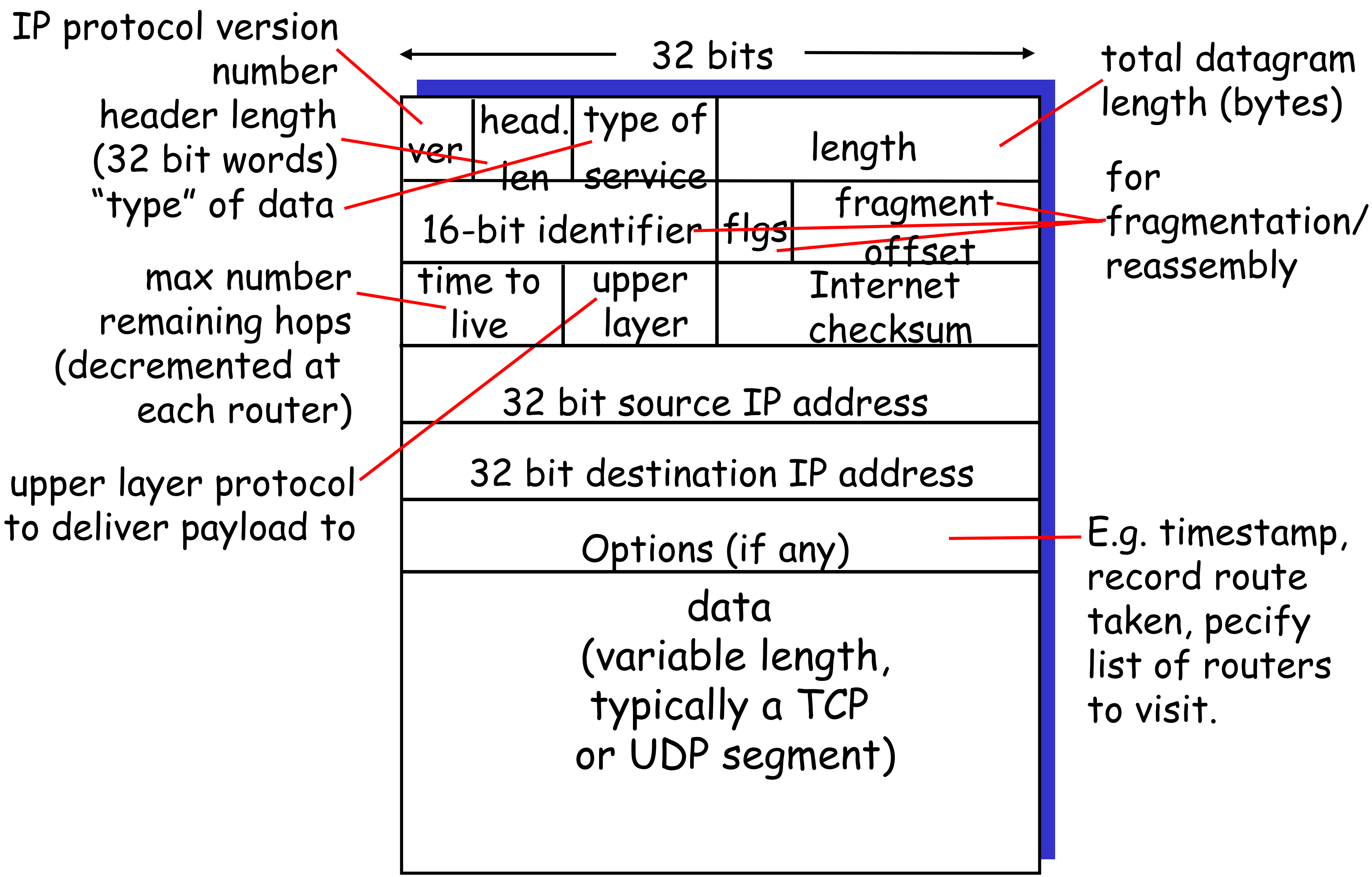
Dest. network	next router	Nhops	interface
223.1.1	-	1	223.1.1.4
223.1.2	-	1	223.1.2.9
223.1.3	-	1	223.1.3.27

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!!! (hooray!)

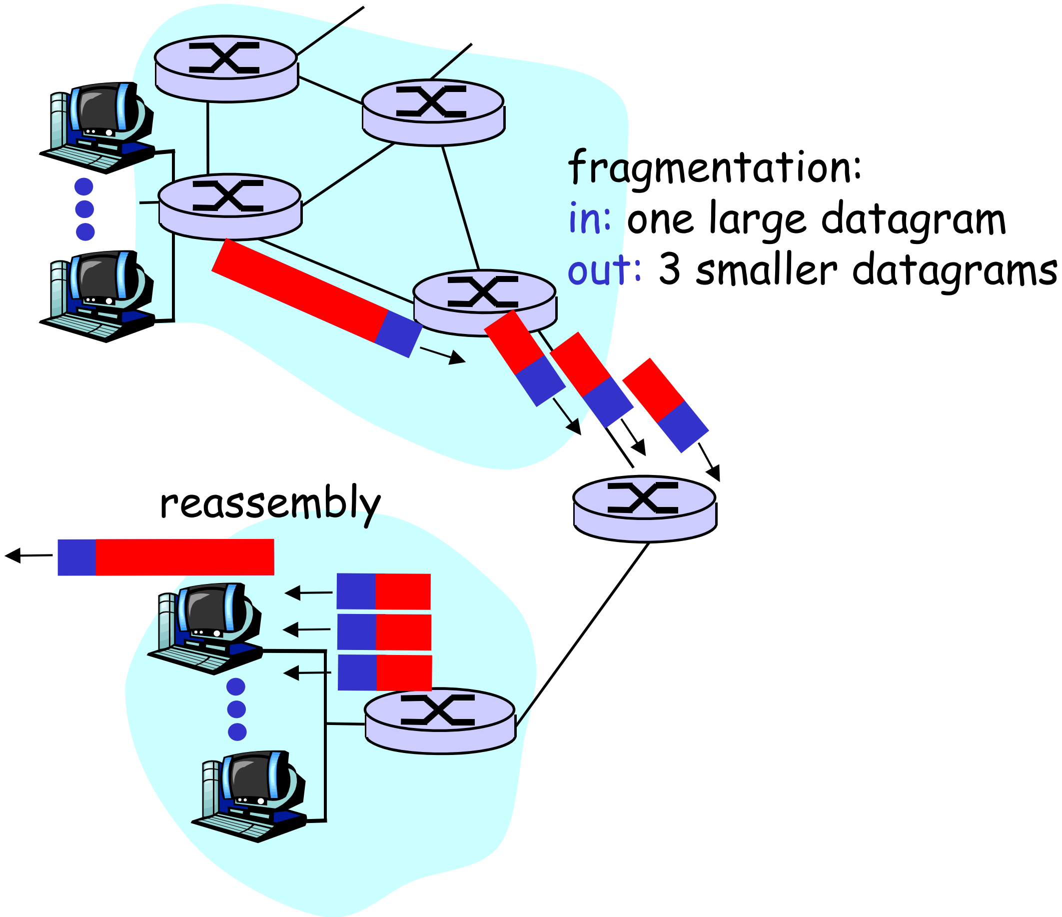


IP datagram format

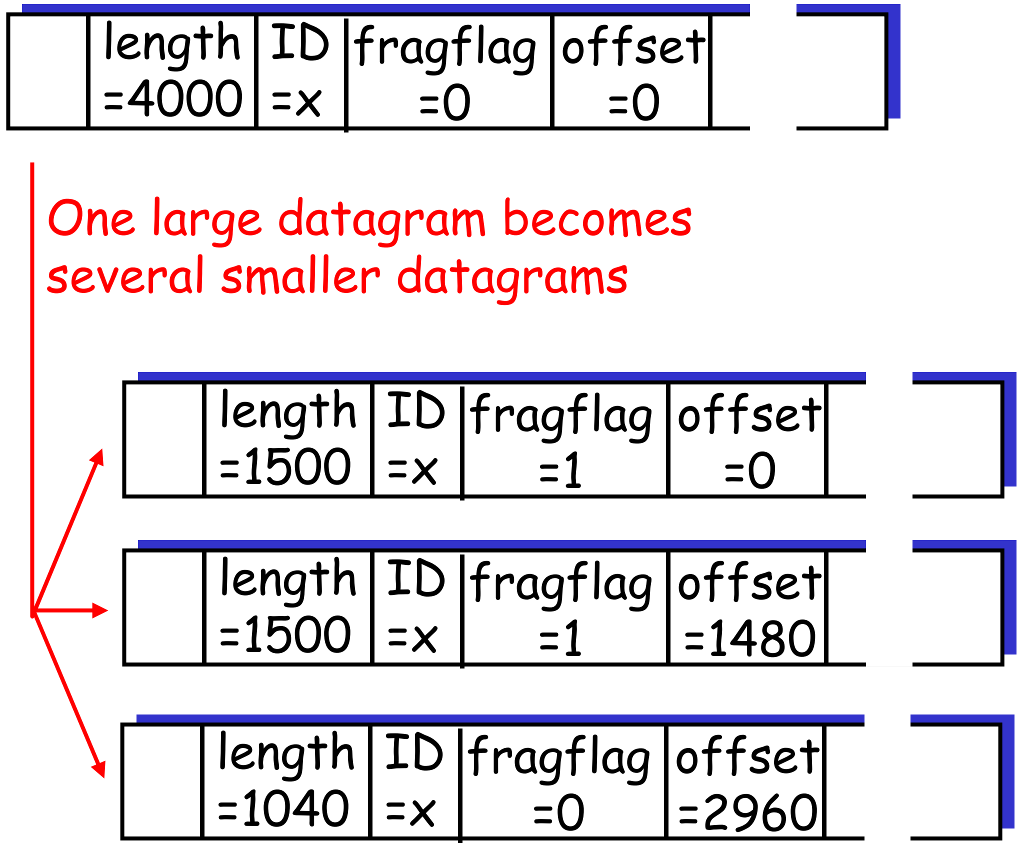


IP Fragmentation & Reassembly

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



IP Fragmentation and Reassembly



ICMP: Internet Control Message Protocol

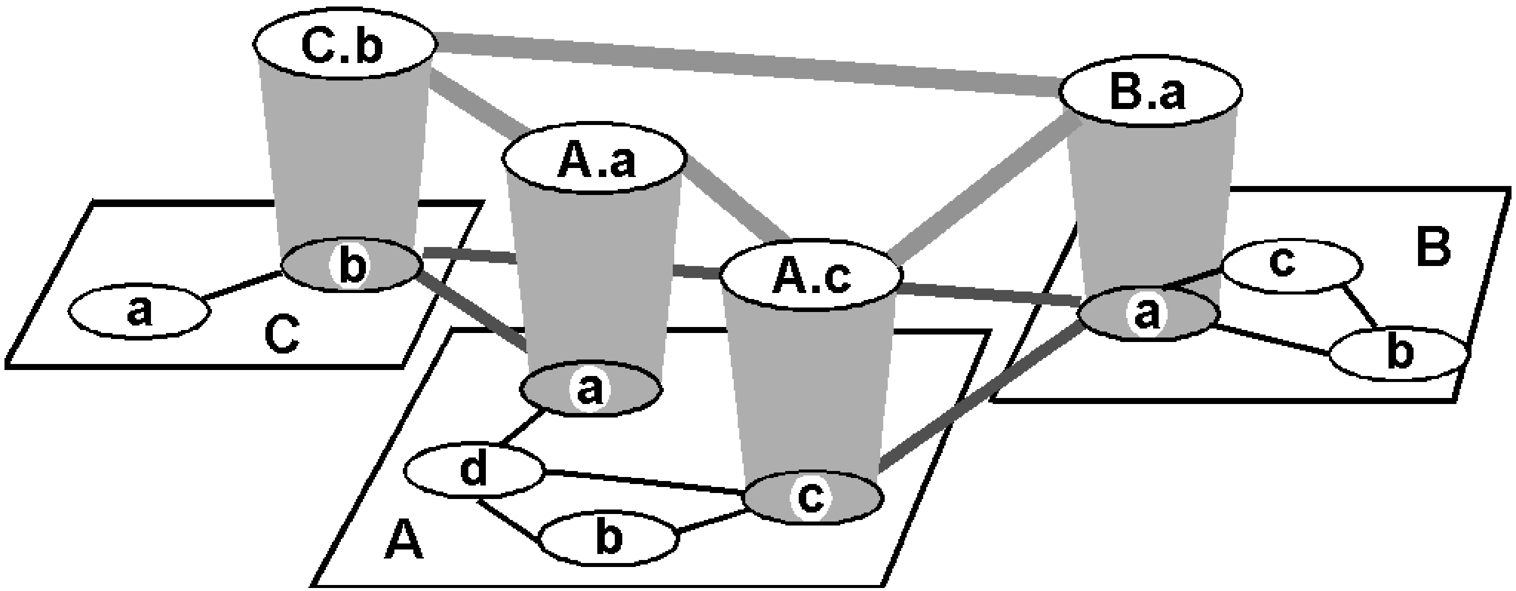
- used by hosts, routers, gateways to communication network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Routing in the Internet

- The Global Internet consists of Autonomous Systems (AS) interconnected with each other:
  - **Stub AS**: small corporation
  - **Multihomed AS**: large corporation (no transit)
  - **Transit AS**: provider
- Two level routing:
  - Intra-AS: administrator is responsible for choice
    - RIP: Routing Information Protocol - distance vector
    - OSPF: Open Shortest Path First - link-state
    - EIGRP: Enhanced Internal Gateway Routing Protocol (Cisco proprietary successor for RIP)
  - Inter-AS: unique standard : BGP

Internet AS Hierarchy



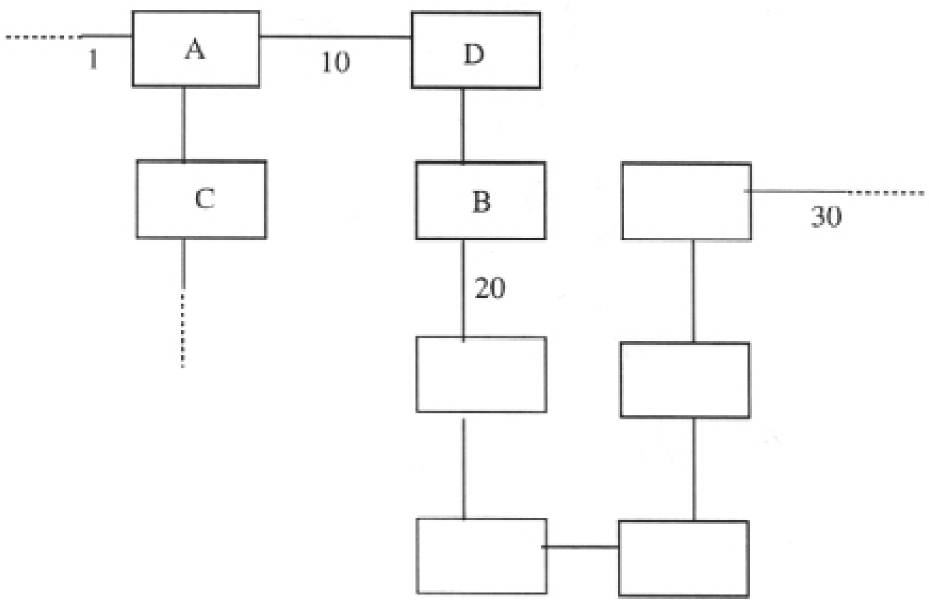
RIP (Routing Info Protocol)

- Distance vector type scheme
- Included in BSD-UNIX Distribution in 1982
- Distance metric: # of hops (max = 15 hops)
- Distance vector: exchanged every 30 sec via a Response Message (also called **Advertisement**)
- Each Advertisement contains up to 25 destination nets

RIP (from perspective of router D)

Letters are routers and numbers on links are network addresses

dest net	next router	number of hops to destination
1	A	2
20	B	2
30	B	7
10	--	1
....	....	....



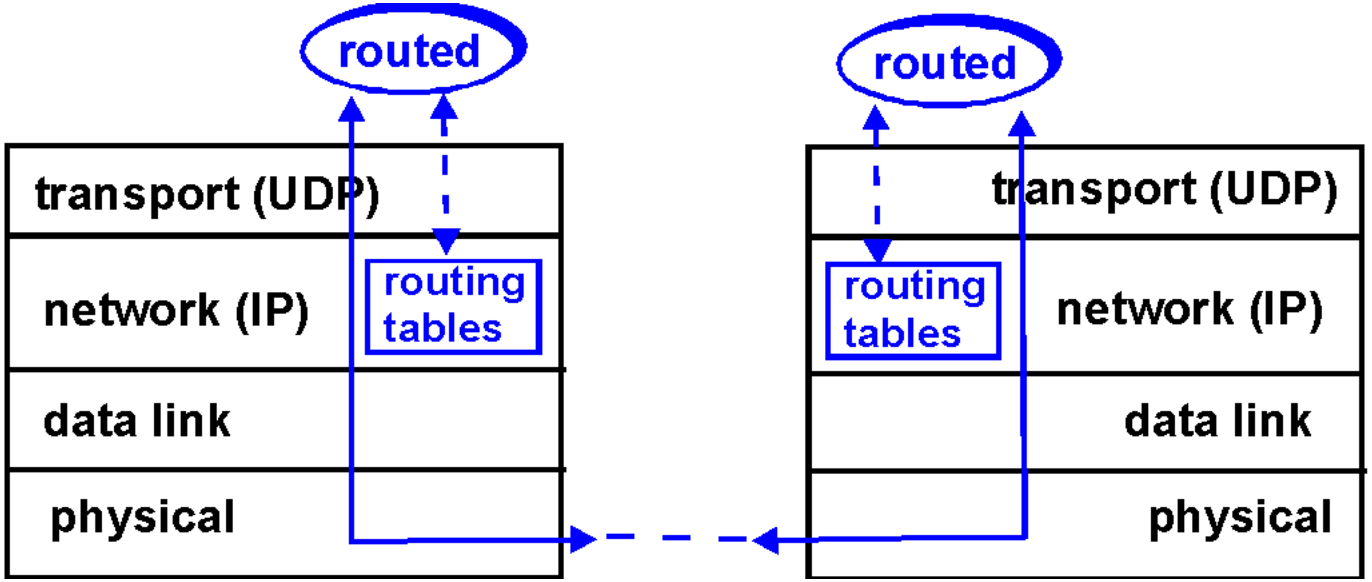
RIP: Link Failure and Recovery

- If no advertisement heard after 180 sec, neighbor/link dead
- Routes via the neighbor are invalidated; new advertisements sent to neighbors
- Neighbors in turn send out new advertisements if their tables changed
- Link failure info quickly propagates to entire net
- Poison reverse used to prevent ping-pong loops (infinite distance = 16 hops)
- Routers can request info about neighbor's cost
- Advertisements are sent via UDP using port #520 as standard IP datagram

RIP Table processing

- RIP routing tables managed by an **application** process called routed (daemon)
- routed is pronounced route-d
- The application process is a part of the Unix OS and uses socket programming as we know it
- Each routed exchanges information with other routed processes running on other machines
- advertisements encapsulated in UDP packets (no reliable delivery required; advertisements are periodically repeated)

RIP Table processing



RIP Table example

Destination	Gateway	Flags	Ref	Use	Interface
127.0.0.1	127.0.0.1	UH	0	26492	lo0
192.168.2.	192.168.2.5	U	2	13	fa0
193.55.114.	193.55.114.6	U	3	58503	le0
192.168.3.	192.168.3.5	U	2	25	qaa0
224.0.0.0	193.55.114.6	U	3	0	le0
default	193.55.114.129	UG	0	143454	

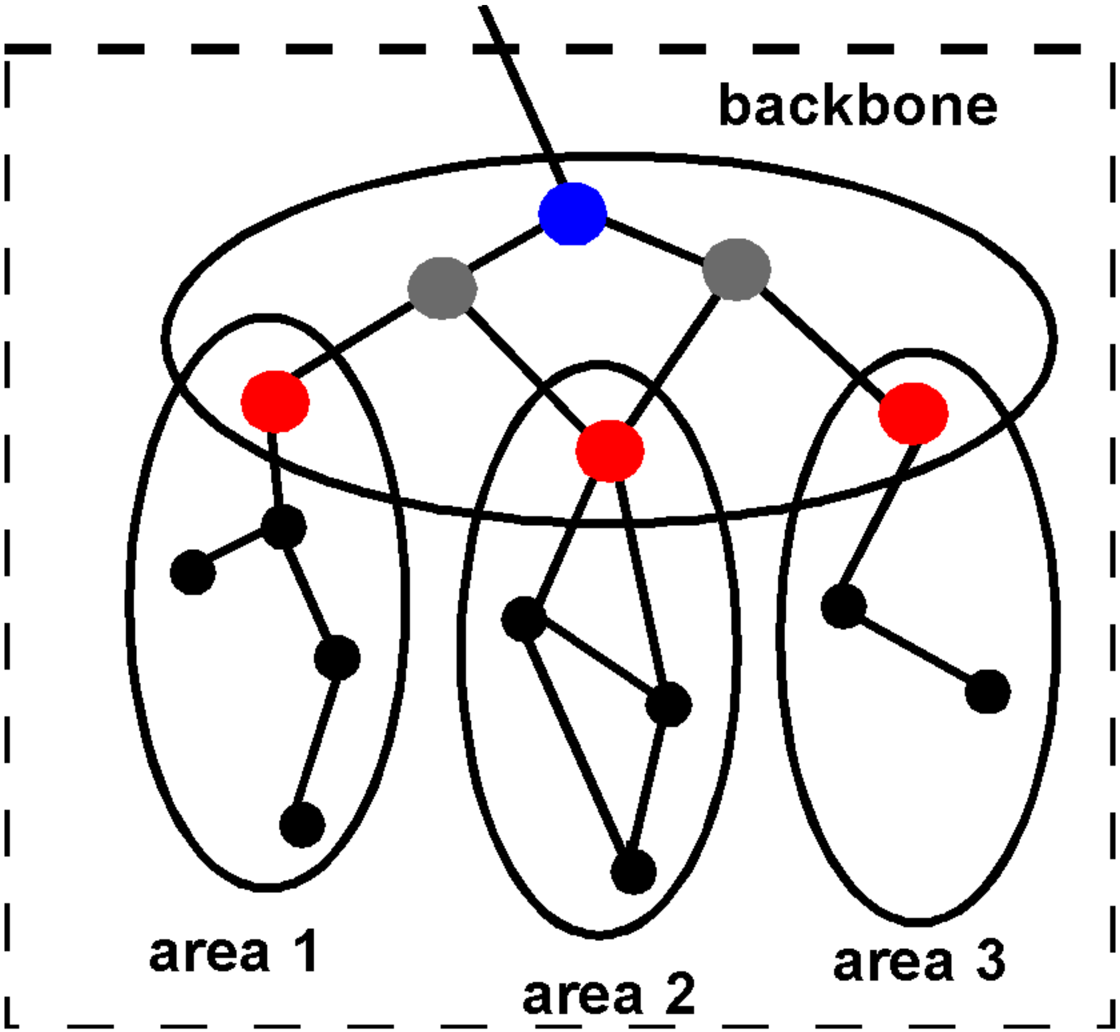
- Three attached class C networks (LANs)
  - Router only knows routes to attached LANs
  - Default router used to “go up”
  - Route multicast address: 224.0.0.0
  - Loopback interface (for debugging)

OSPF (Open Shortest Path First)

- "open": publicly available
- uses the Link State algorithm (ie, LS packet dissemination; topology map at each node; route computation using Dijkstra's alg)
- OSPF advertisement carries one entry per neighbor router
- advertisements disseminated to ENTIRE Autonomous System (via flooding)

OSPF "advanced" features (not in RIP)

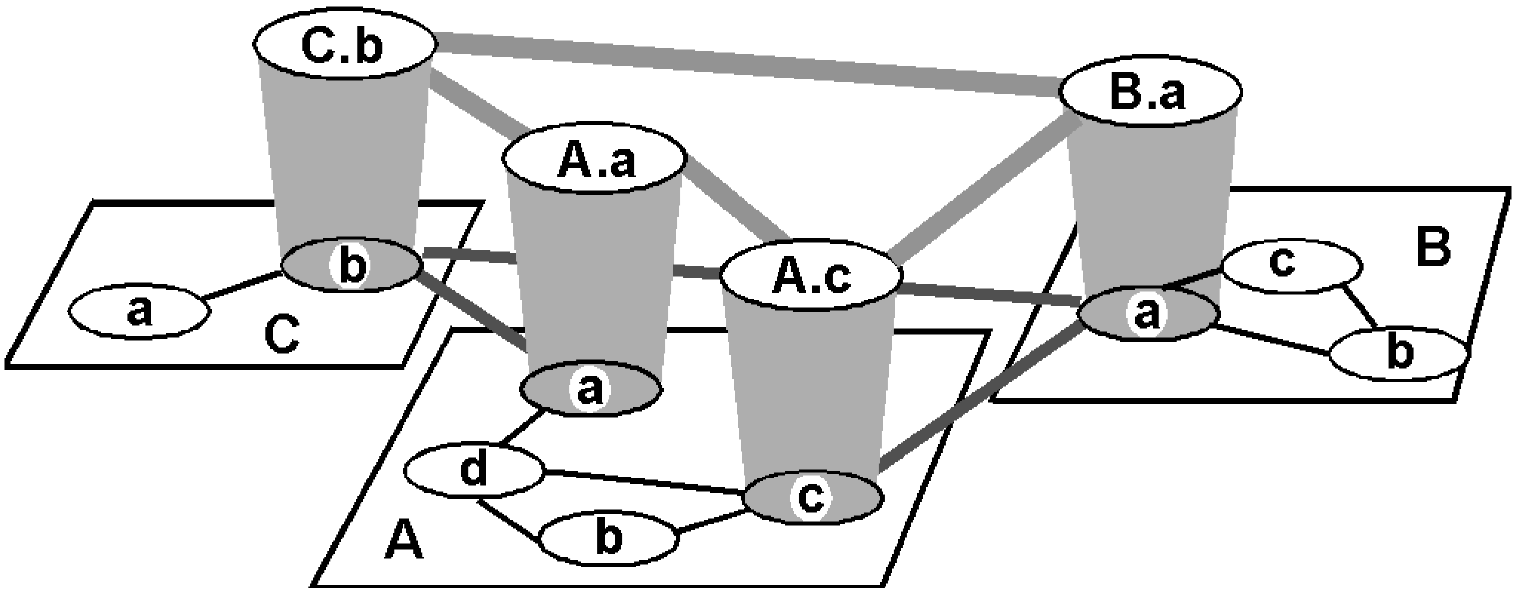
Hierarchical OSPF in large domains; thousands of routers



Hierarchical OSPF

- Two level hierarchy: local area and backbone
- Link state advertisements do not leave respective areas
- Nodes in each area have detailed area topology; they only know direction (shortest path) to networks in other areas
- **Area Border routers** "summarize" distances to networks in the area and advertise them to other Area Border routers
- **Backbone routers** run an OSPF routing alg limited to the backbone

Inter-AS routing



Why different Intra- and Inter-AS routing ?

- **Scale**: Inter provides an extra level of routing table size and routing update traffic reduction above the Intra layer
- **Policy**: Inter is concerned with policies (which provider we must select/avoid, etc). Intra is contained in a single organization, so, no policy decisions necessary
- **Performance**: Intra is focused on performance metrics; needs to keep costs low. In Inter it is difficult to propagate performance metrics efficiently (latency, privacy etc). Besides, policy related information is more meaningful.

We need **BOTH**!

Inter-AS routing (cont)

- BGP (Border Gateway Protocol): the de facto standard
- **Path Vector** protocol: an extension of Distance Vector
- Each Border Gateway broadcasts to neighbors (peers) the entire path (ie, sequence of ASes) to destination (no cost info is sent)
- For example, Gwy X may store the following path to destination Z:Path (X,Z) = 102,111,120,…,2012

Path (X,Z) = 102,111,120,…,2012

  - Loop Avoidance
  - Policy Routing

Inter-AS routing (cont)

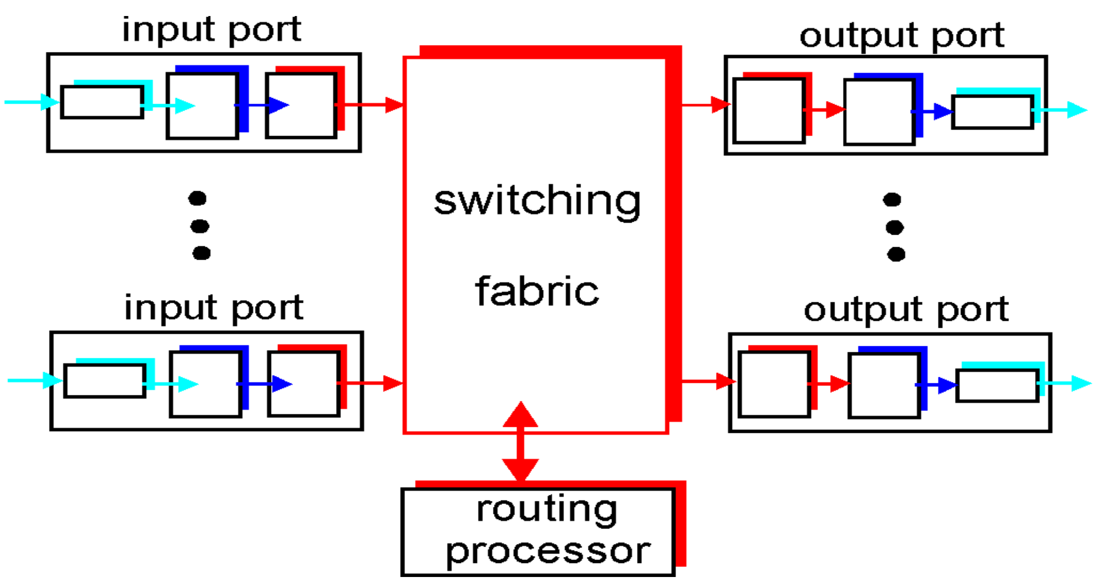
- Peers exchange BGP messages using TCP
  (peers are immediate neighbor ASs)
- OPEN msg opens TCP connection to peer
- UPDATE msg advertises new path (or withdraws old)
- KEEPALIVE msg keeps connection alive in absence of UPDATES; it also serves as ACK to an OPEN request
- NOTIFICATION msg reports errors in previous msg; also used to close a connection

Address Management

- As Internet grows, we run out of addresses
- Solution (a): **subnetting**. Eg, Class B Host field (16bits) is subdivided into <subnet;host> fields
- Solution (b): **CIDR** (Classless Inter Domain Routing): assign block of contiguous Class C addresses to the same organization; these addresses all share a common prefix

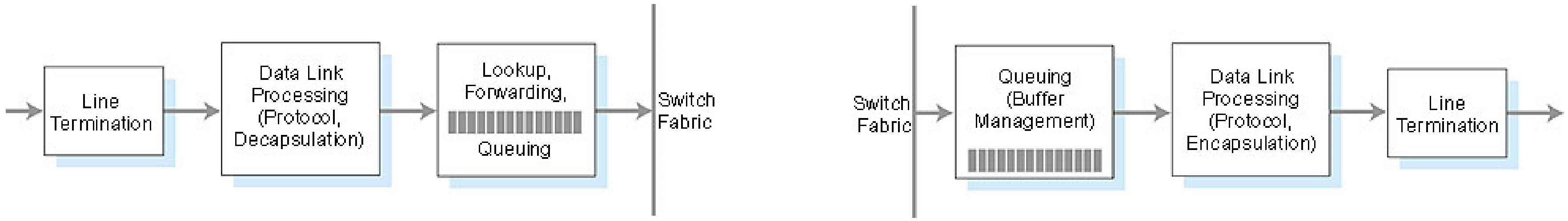
Router Architecture Overview

- Router main functions: *routing* algorithms and protocols processing, *switching* datagrams from an incoming link to an outgoing link



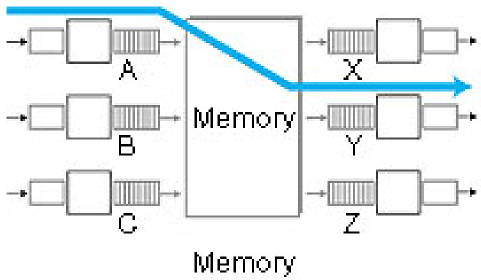
Router Components

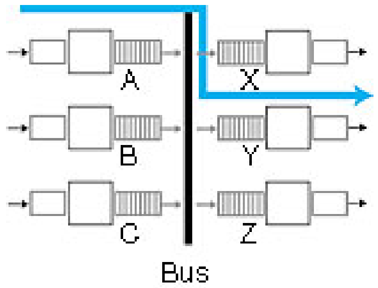
Input and Output Port Processing

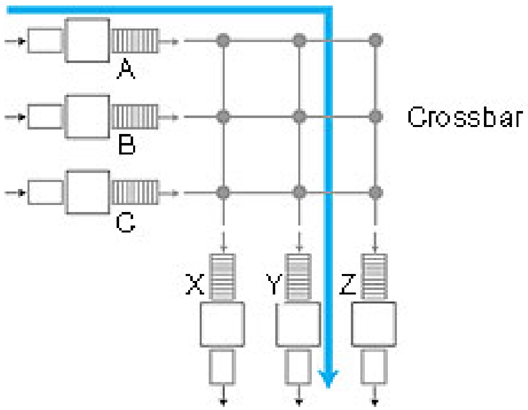


- Line Termination corresponds to physical layer
- Data link processing corresponds to link layer
- Usually, copy of routing table is stored at each input port - avoids using one central CPU
- Packet dropping occurs at input and output queues

The switching fabric



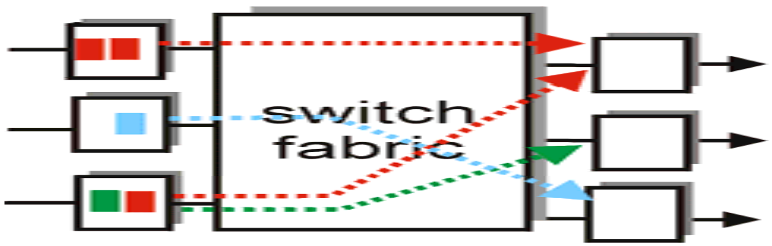




- Switching via memory, a) by shared memory with processors at ports or b) via CPU & ports as IO devices
- Switching via bus, only one packet at time (one bus - (but there are gigabit buses)
- Switching via interconnection network - (crossbar) 2N buses for N output and N input ports

Queuing At Input and Output Ports

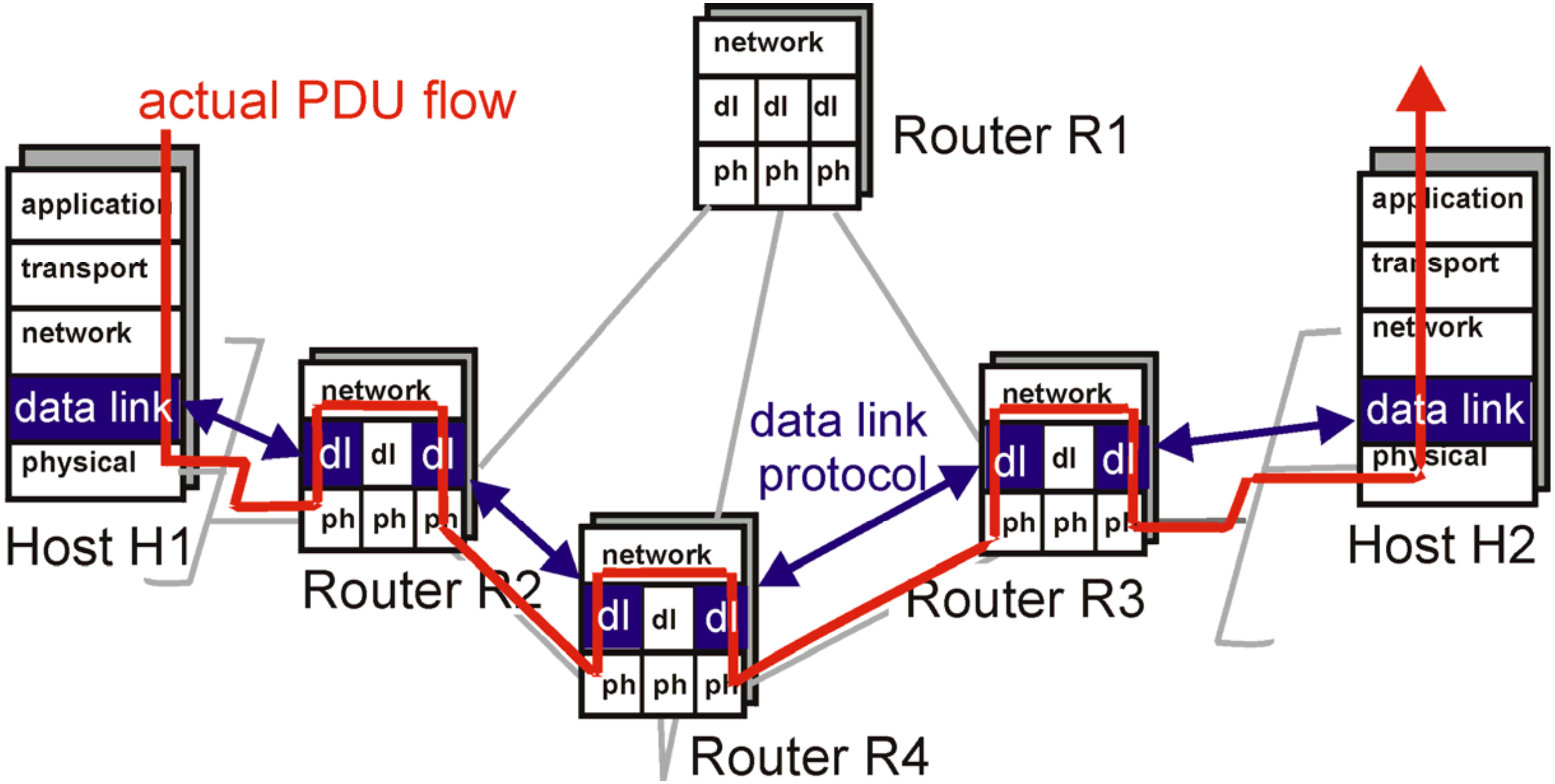
- Queues build up whenever there is a rate mismatch or blocking. Consider the following scenarios:
  - Fabric speed is faster than all input ports combined; more datagrams are destined to an output port than other output ports; queuing occurs at output port
  - Fabric bandwidth is not as fast as all input ports combined; queuing may occur at input queues;
  - HOL blocking: fabric can deliver datagrams from input ports in parallel, except if datagrams are destined to same output port; in this case datagrams are queued at input queues; there may be queued datagrams that are held behind HOL conflict, even when their output port is available



output port contention at time t - only one red packet can be transferred

green packet experiences HOL blocking

Link Layer Protocols

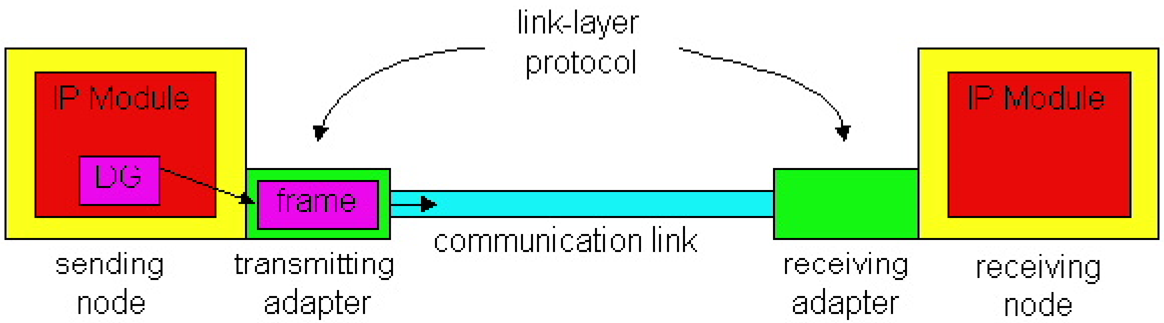


Link Layer Services

- ***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
- ***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
- ***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

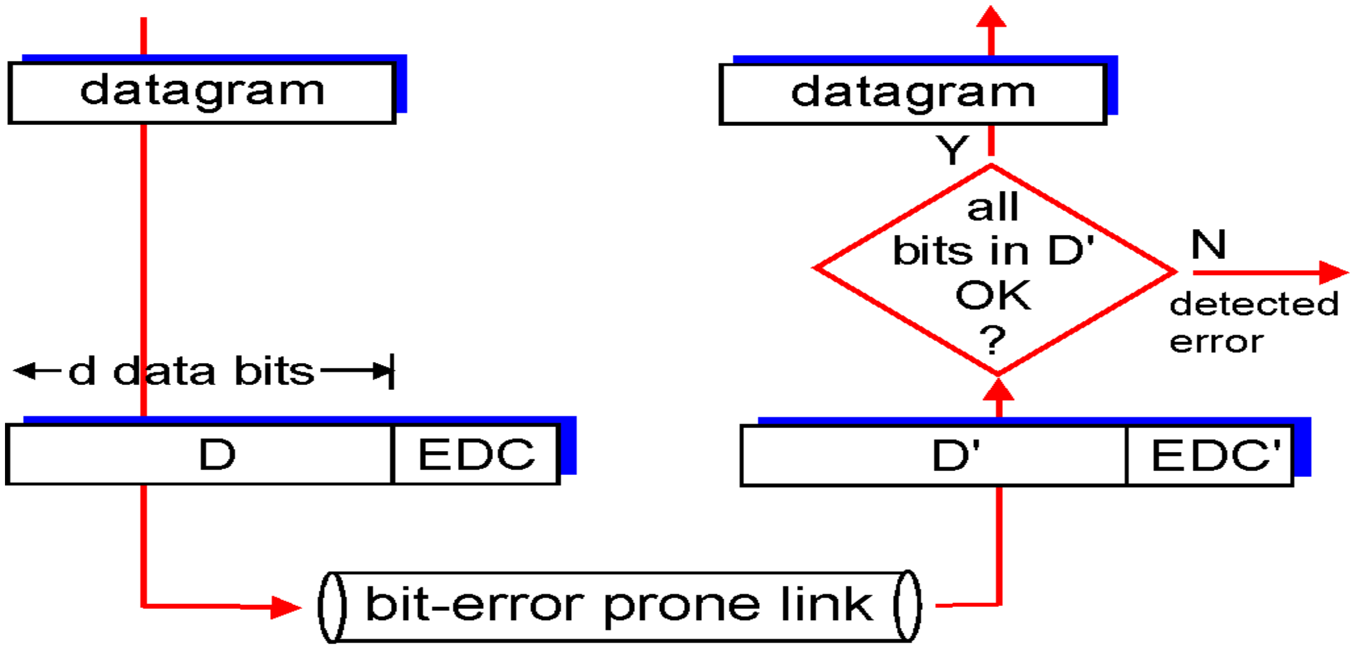
Link Layer Protocol Implementation

- Everything is implemented in the adapter
  - includes: RAM, DSP chips, host bus interface, and link interface
- Adapter **send** operations: encapsulates (set sequence numbers, feedback info), adds error detection bits, implements channel access for shared medium, transmits on link
- Adapter **receive** operations: error checking and correction, interrupts host to send frame up the protocol stack, updates state info regarding feedback to sender, sequence numbers, etc.



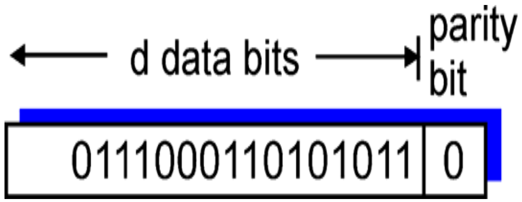
Error Detection

EDC= Error Detection and Correction bits (redundancy)
D= data protected by error checking, may include some header fields
Error detection is not 100%; protocol may miss some errors, but rarely
Larger EDC field yields better detection and correction, more overhead

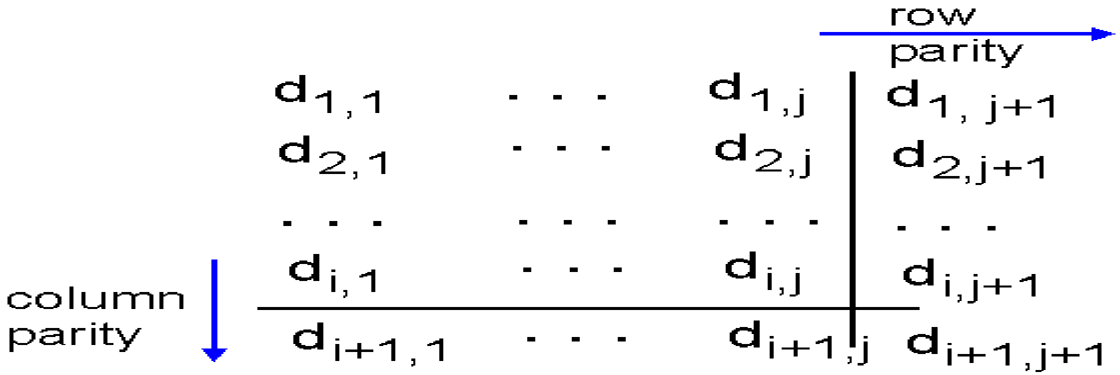


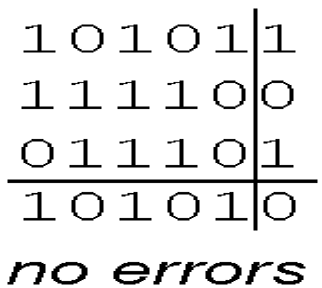
Parity Checking (technique 1 of 3)

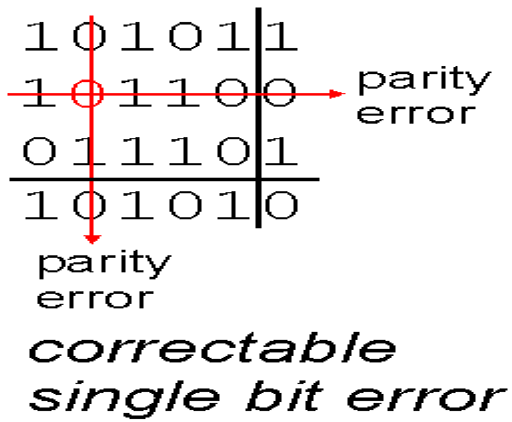
Single Bit Parity:
Detect single bit errors



Two Dimensional Bit Parity:
Detect and correct single bit errors





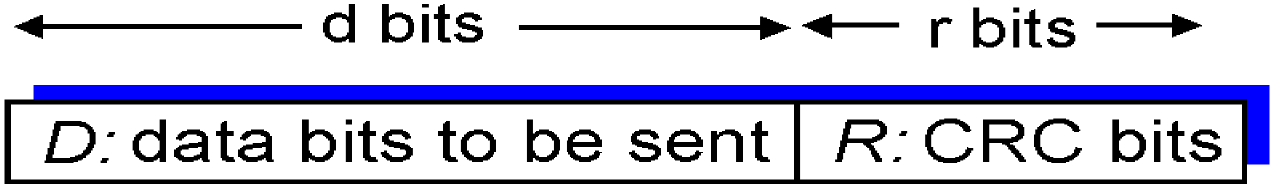


Checksumming Methods (technique 2 of 3)

- ***Internet Checksum***: View data as made up of 16 bit integers; add all the 16 bit fields (one's complement arithmetic) and append the frame with the resulting sum; the receiver repeats the same operation and matches the checksum sent with the frame

Cyclic Redundancy Codes (technique 3 of 3)

- ***CRC or polynomial codes***:
  - Data is viewed as a string of coefficients of a polynomial (D)
  - A *Generator* polynomial is chosen (=> r+1 bits), (G)
  - Multiply D by 2^r (I.e. shift left r bits).
  - Divide (modulo 2) the D*2^r polynomial by G. Append the remainder (R) to D. Note that, by construction, the new string <D,R> is now divisible exactly by G using mod 2 arithmetic
  - addition is defined as XOR. No borrows or carried => addition and subtraction are the same

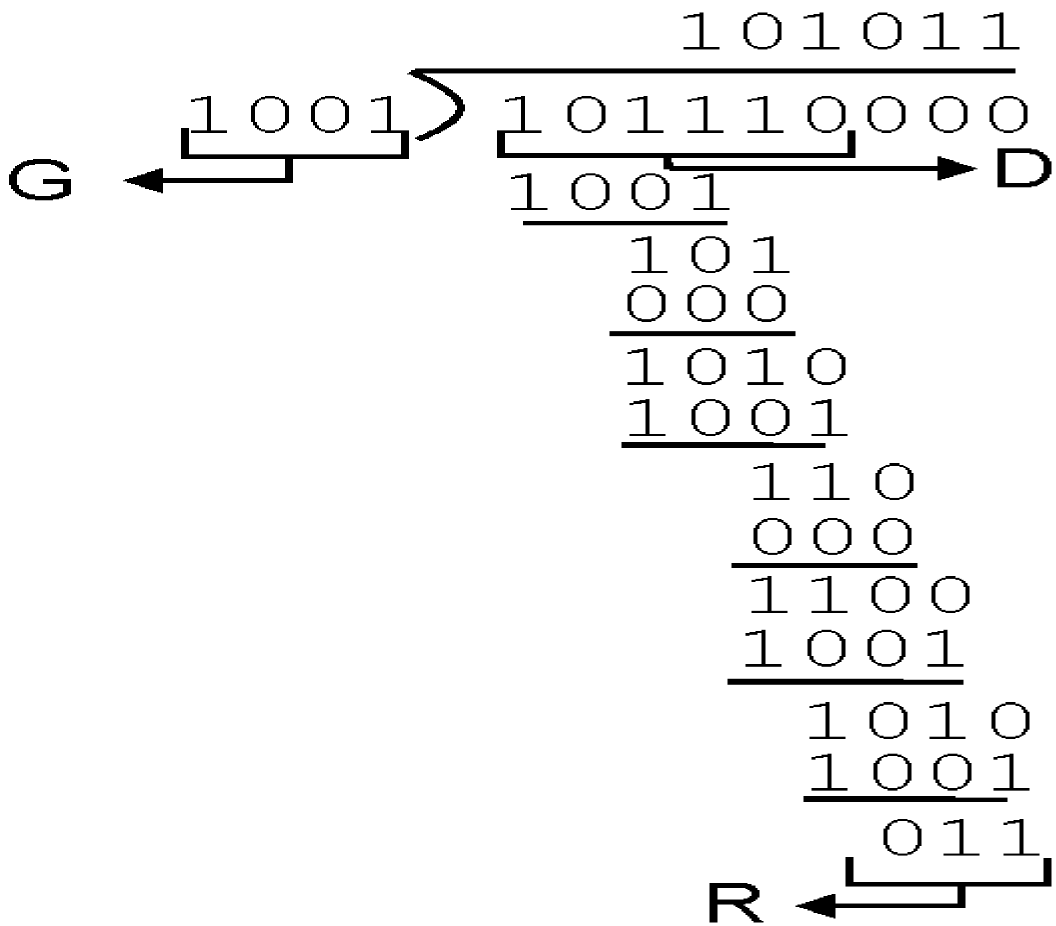


bit pattern

$$D * 2^{r} \quad \text{XOR} \quad R$$

mathematical formula

CRC Example



101011
1001
101110000
G
D
1001
101
000
1010
1001
110
000
1100
1001
1010
1001
011
R

CRC Implementation (cont)

- Sender carries out on-line, in Hardware, the division of the string D by polynomial G and appends the remainder R to it
- Receiver divides < D,R> by G; if the remainder is non-zero, the transmission was corrupted
- Can detect burst errors of less than r+1 bits and any odd number of bit errors
- International standards for G polynomials of degrees 8, 12, 15 and 32 have been defined
- ARPANET was using a 24 bit CRC for the alternating bit link protocol

Multiple Access Links and Protocols

Three types of links:

(a) Point-to-point (single wire)

PPP, HDLC

(b) Broadcast: shared wire or medium

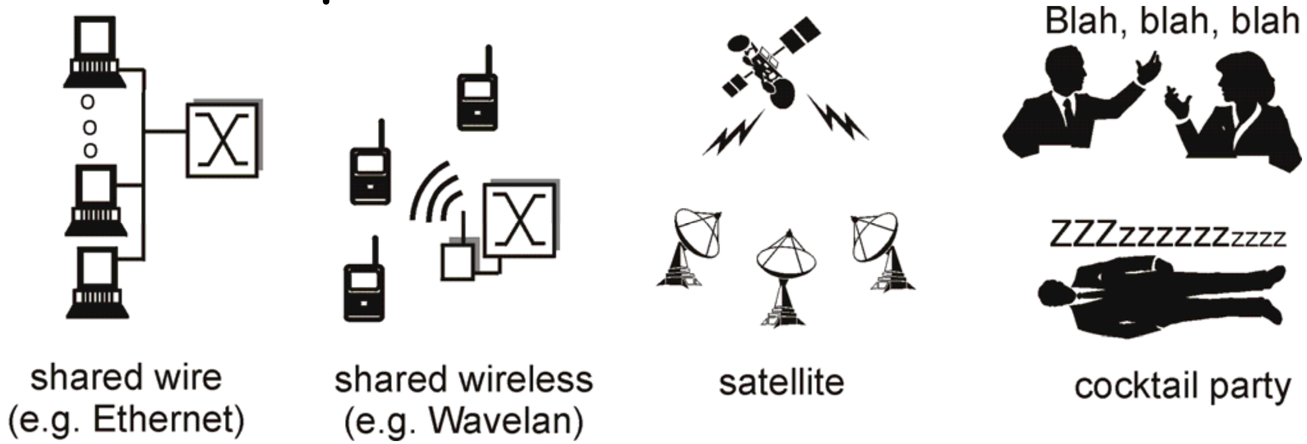
Ethernet, wireless

(c) Switched

switched Ethernet, ATM

We start with **Broadcast** links. Main challenge:

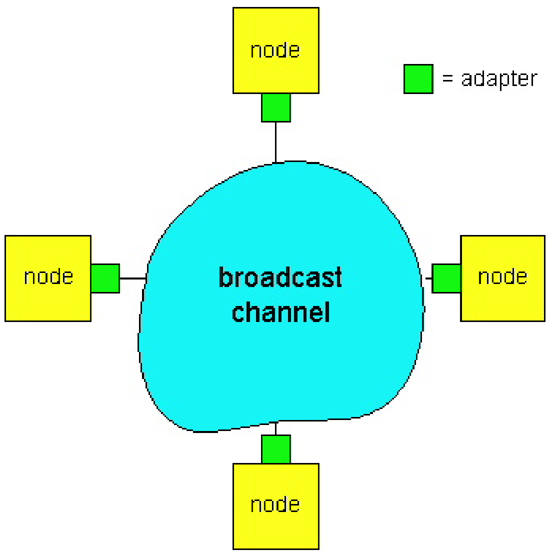
Multiple Access Protocol



Multiple Access Control (MAC) Protocols

- MAC protocol: coordinates transmissions from different stations in order to minimize/avoid collisions
  - **Channel Partitioning**
  - **Random Access**
  - **"Taking turns"**

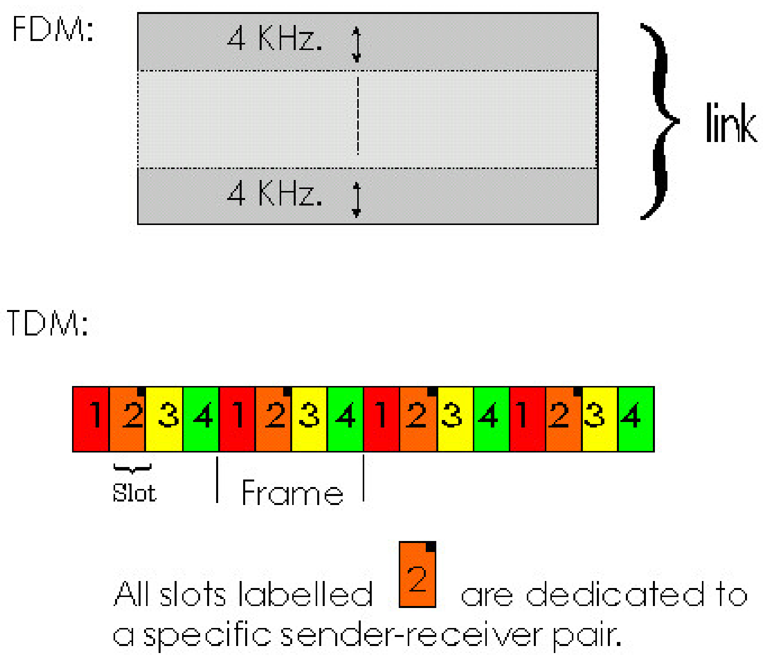
- Goal: **efficient, fair, simple, decentralized**



Channel Partitioning MAC protocols

Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM)

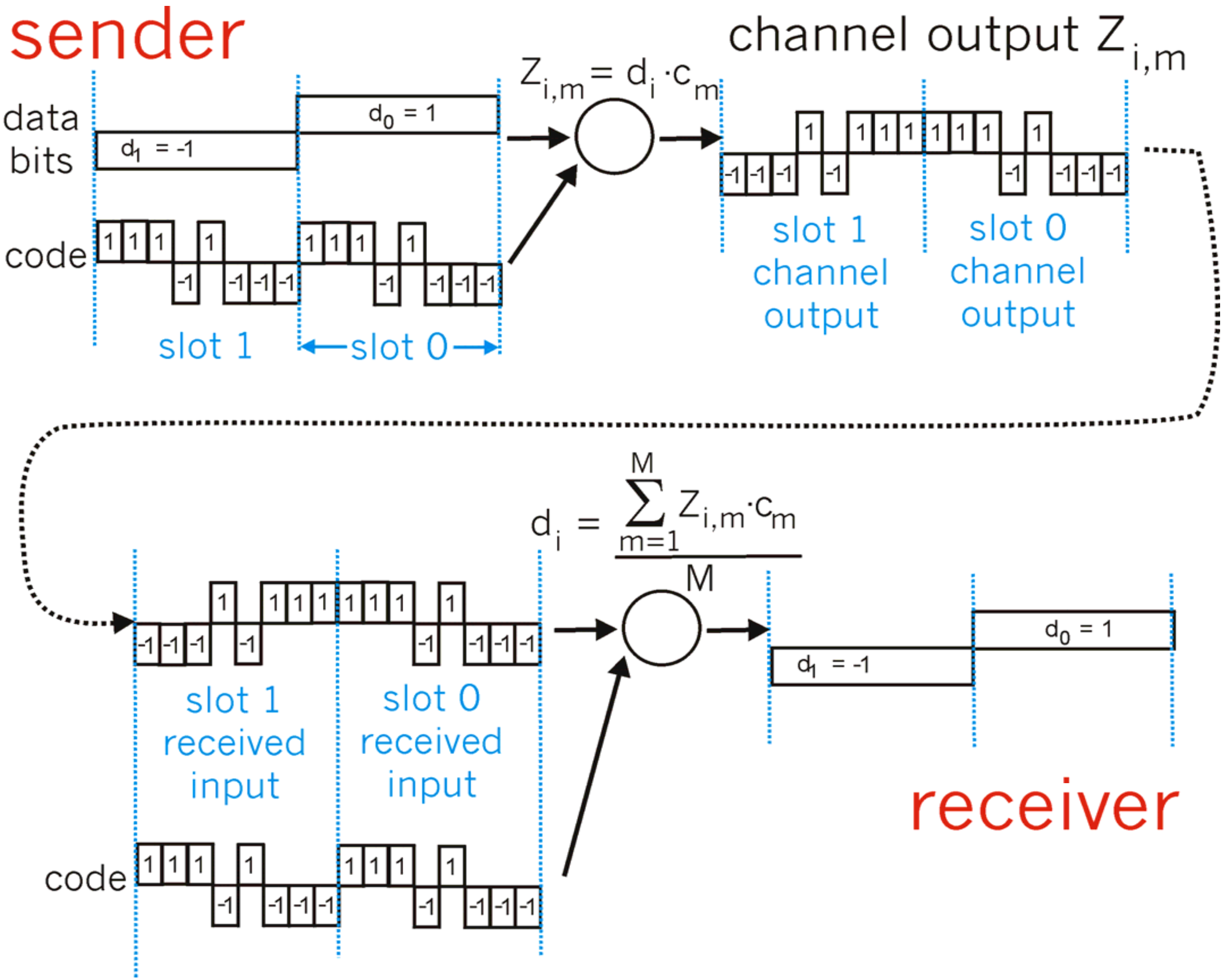
- TDM (Time Division Multiplexing): channel divided into N time slots, one per user; inefficient with low duty cycle.
  Note: Frame in TDM diagram below refers to Time Frame. A single link Frame data unit is sent in one of the four time slots.
- FDM (Frequency Division Multiplexing): frequency subdivided.



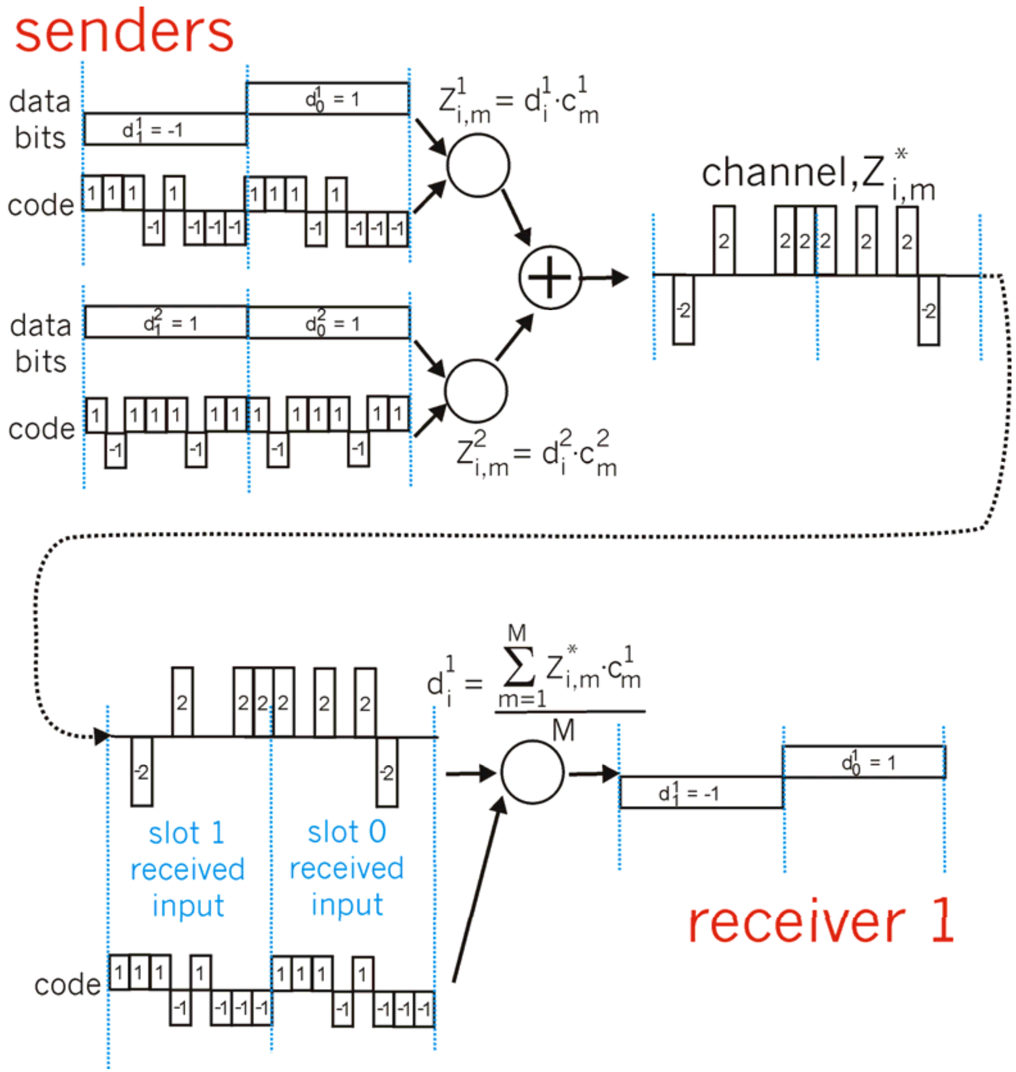
Channel Partitioning (CDMA)

- **CDMA: Code Division Multiple Access**
  - exploits **spread spectrum** encoding scheme
- Used mostly in **wireless** broadcast channels (cellular, satellite,etc)
- All users share the **same frequency**, but each user has **own "chipping" sequence**

CDMA Encode/Decode



CDMA: two-sender interference



CDMA Properties

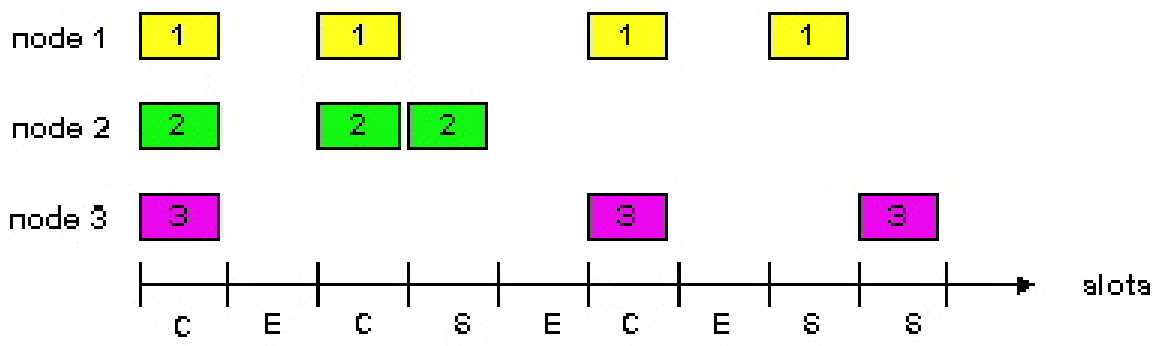
- protects users from interference and jamming (used in WW II)
- protects users from radio multipath fading
- allows multiple users to "coexist" and transmit simultaneously with minimal interference (if codes are "orthogonal")
- CDMA used in Qualcomm cellphones:
  - channel efficiency improved by factor of 4 with respect to TDMA

Random Access protocols

- A node transmits **at random** at **full** channel data rate R.
- If two or more nodes “**collide**”, they retransmit at random times
- The **random access MAC** protocol specifies how to detect collisions and how to recover from them (via delayed retransmissions, for example)
- Examples of random access MAC protocols
  - SLOTTED ALOHA
  - ALOHA
  - CSMA and CSMA/CD

Slotted Aloha

- Time is divided into equal size slots (= full packet size)
- a newly arriving station transmits at the beginning of the next slot
- if collision occurs (assume channel feedback, eg the receiver informs the source of a collision), the source retransmits the packet at each slot with probability P, until successful.
- Success (S), Collision (C), Empty (E) slots
- S-ALOHA is efficient; it is fully decentralized.



Slotted Aloha efficiency

If N stations have packets to send, and each transmits in each slot with probability P, the probability of successful transmission S is:

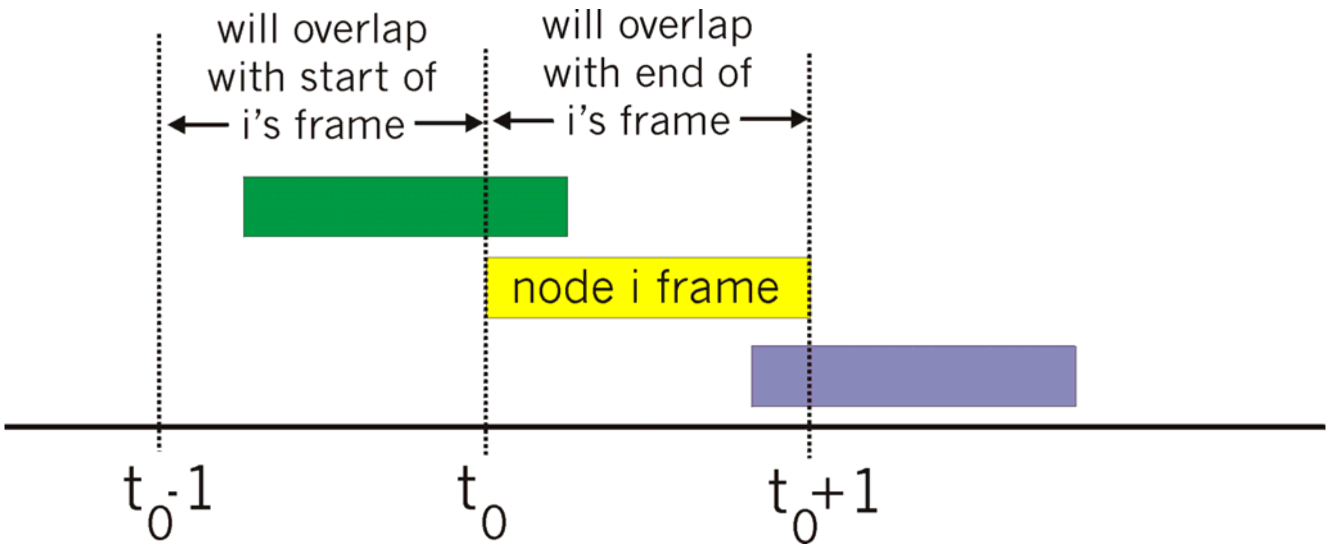
S = Prob (only one transmits) = N P (1-P)^(N-1)

Optimal value of P: P = 1/N
For example, if N=2, S= .5

For N very large one finds S= 1/e (approximately, .37)

Pure (unslotted) ALOHA

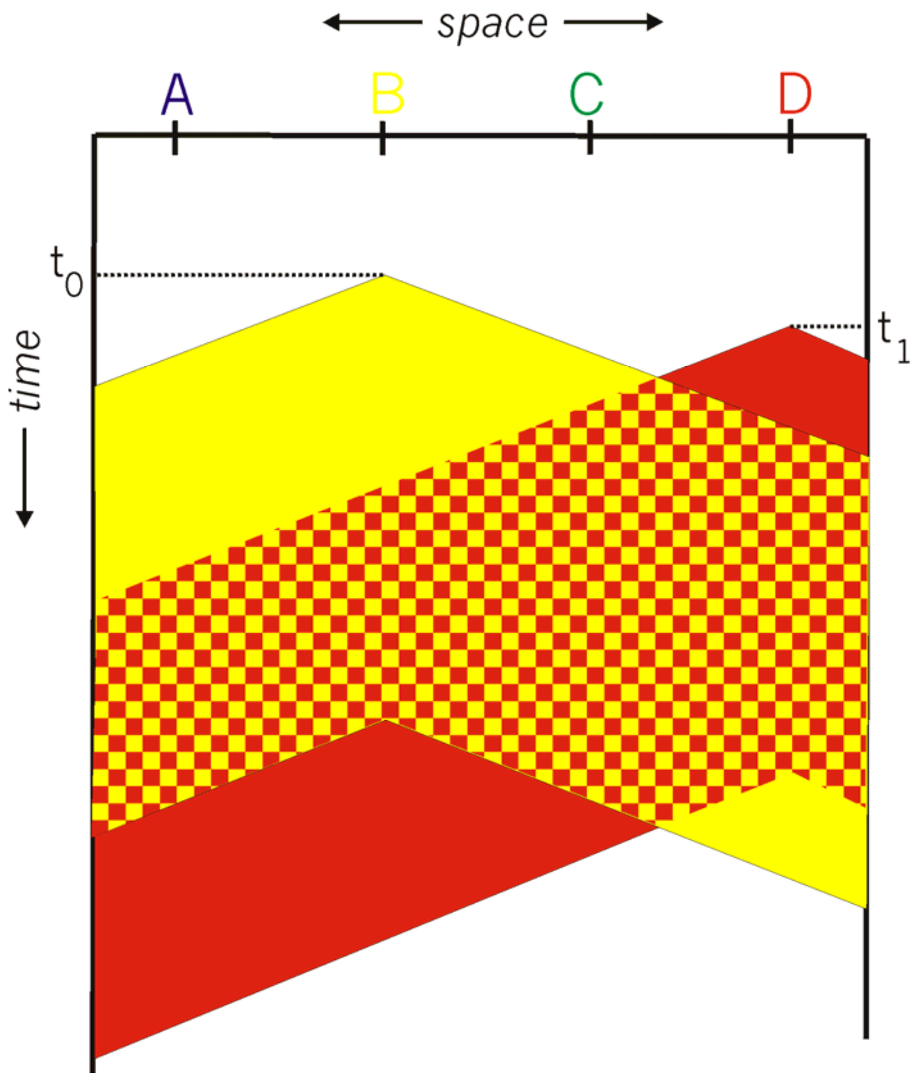
- Slotted ALOHA requires slot synchronization
- A simpler version, pure ALOHA, does not require slots
- A node transmits without awaiting for the beginning of a slot
- Collision probability increases (packet can collide with other packets which are transmitted within a window twice as large as in S-Aloha)
- Throughput is reduced by one half, i.e. S= 1/2e



CSMA (Carrier Sense Multiple Access)

- **CSMA**: listen before transmit. If channel is sensed busy, defer transmission
- **Persistent** CSMA: retry immediately when channel becomes idle (this may cause instability)
- **Non persistent CSMA**: retry after random interval
- Note: collisions may still exist, since two stations may sense the channel idle at the same time (or better, within a "vulnerable" window = round trip delay)
- In case of collision, the entire packet transmission time is wasted

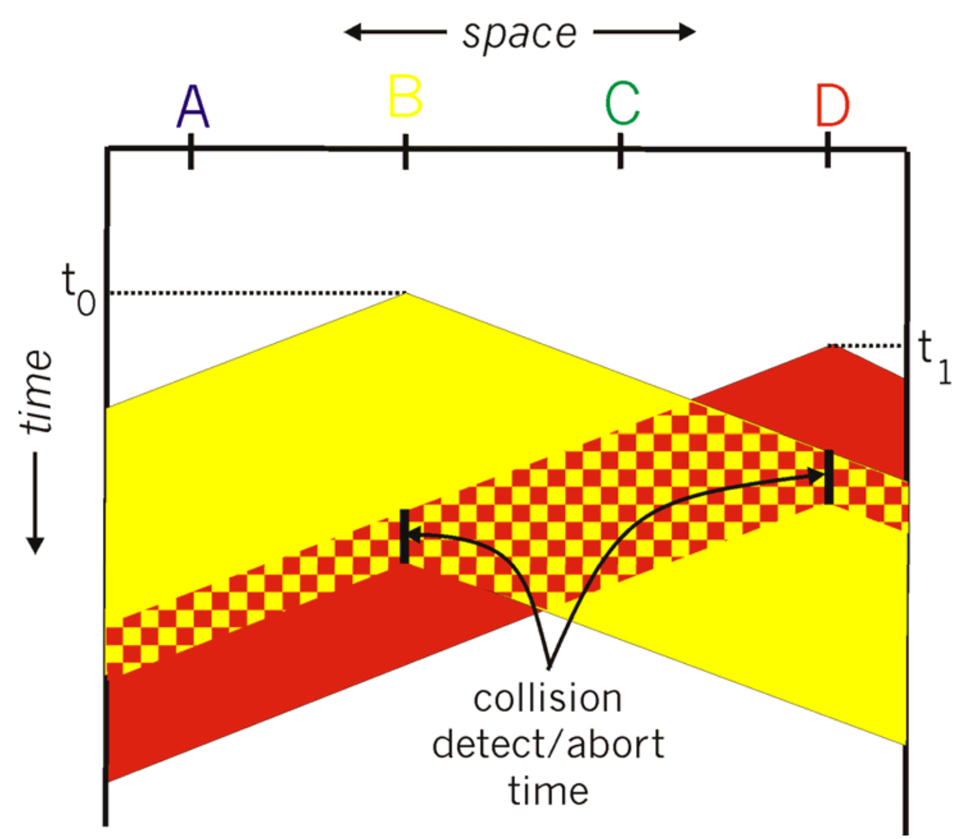
CSMA collisions



CSMA/CD (Collision Detection)

- **CSMA/CD**: like in CSMA
  - collisions are detected within a few bit times
  - Transmission is then aborted, reducing the channel wastage considerably
  - **persistent** retransmission is implemented
- Collision detection is **easy in wired LANs**:
  - can measure signal strength on the line
- Collision detection **cannot be done in wireless LANs** :
  - receiver is off while transmitting, to avoid damaging it with excess power
- CSMA/CD can approach channel utilization =1 in LANs:
  - low ratio of propagation over packet transmission time

CSMA/CD collision detection



"Taking Turns" MAC protocols

- **channel partitioning** MAC protocols :TDM, FDM and CDMA
  - + can share the channel fairly
  - - a single station cannot use it all
- **Random access** MAC protocols
  - + a single station can use full channel rate
  - - cannot share the channel fairly
- **Taking Turns** MAC protocols:
  - Achieve both fair and full rate
  - with some extra control overhead

  (a) Polling: Master "invites" the slave

  - Request/Clear overhead, latency, single point of failure

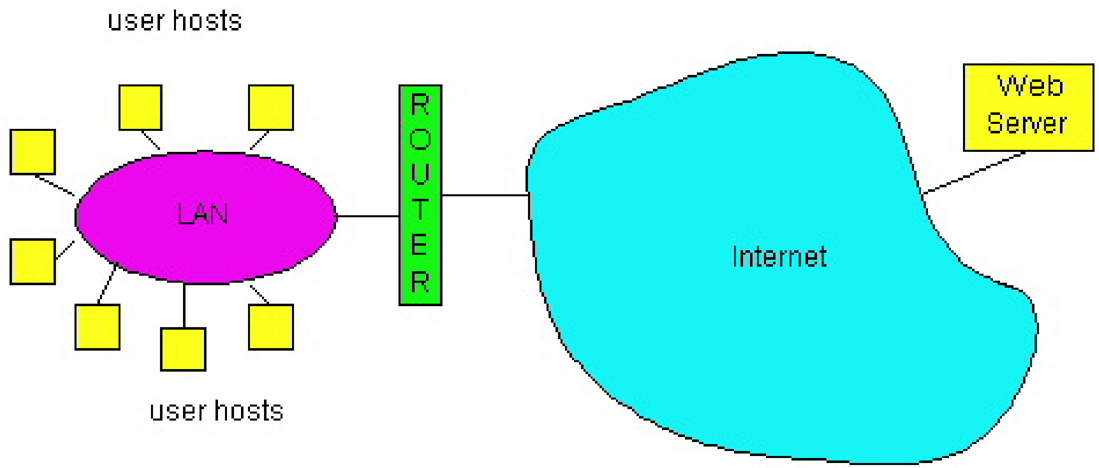
  (b) Token passing: **token** is passed from one node to the next

  + Reduce latency, improve fault tolerance

  - elaborate procedures to recover from **lost token**

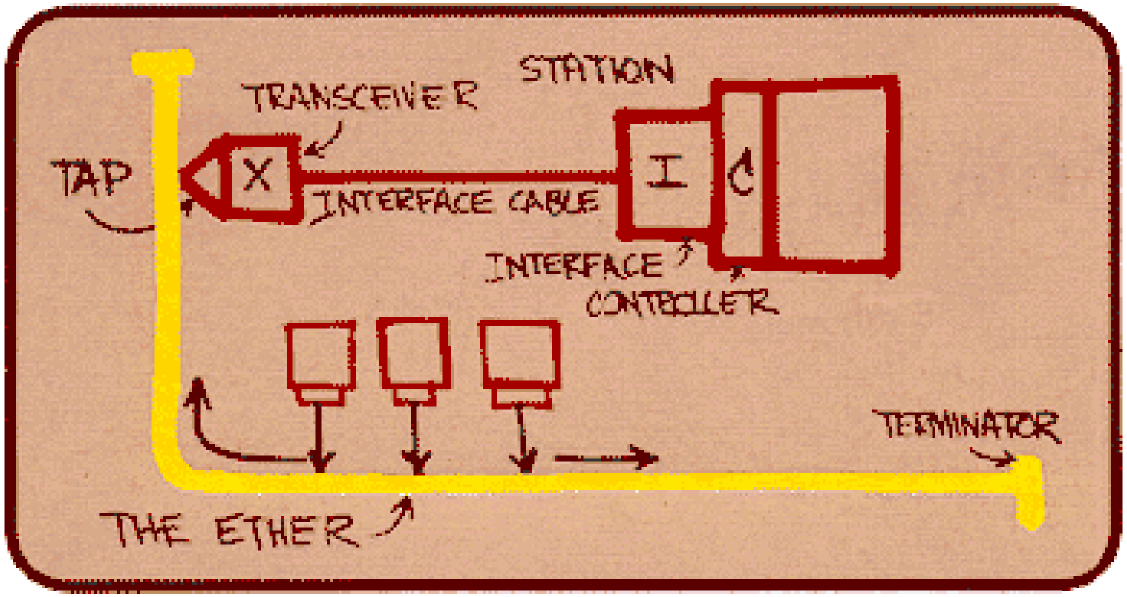
LAN technologies

- MAC protocols used in LANs, to control access to the channel
- **Token Rings**: IEEE 802.5 (IBM token ring), for computer room, or Department connectivity, up to 16Mbps; FDDI (Fiber Distributed Data Interface), for Campus and Metro connectivity, up to 200 stations, at 100Mbps.
- **Ethernets**: employ the CSMA/CD protocol; 10Mbps (IEEE 802.3), Fast E-net (100Mbps), Giga E-net (1,000 Mbps); by far the most popular LAN technology



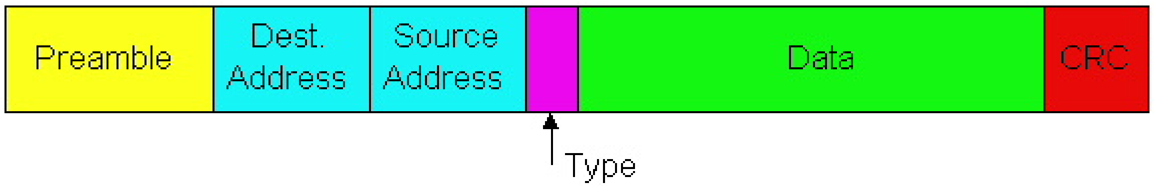
Ethernet

- Widely deployed because:
  - 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 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)
- **Header** contains Destination and Source Addresses and a Type field
- Addresses: 6 bytes, frame is received by all adapters on a LAN and dropped if address does not match
- **Type**: indicates the higher layer protocol, mostly IP but others may be supported such as Novell IPX and AppleTalk)
- **CRC**: checked at receiver, if error is detected, the frame is simply dropped



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;
                   go to A
                   }
             else {done with the frame}
          }
     else {wait until ongoing transmission is over and go to A}
```

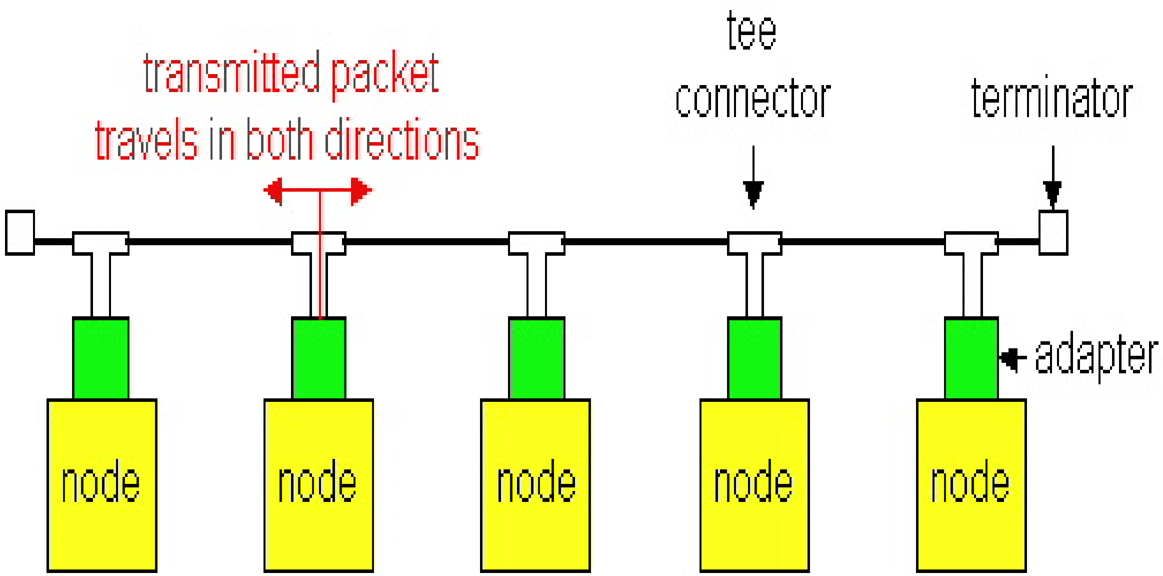
CSMA/CD (Cont.)

- **Jam Signal**: to make sure all other transmitters are aware of the collision; 48 bits;
- **Exponential Backoff**:
  - Goal is to adapt the offered rate by transmitters to the estimated current load (ie backoff when load is heavy)
  - After the first collision Choose K from {0,1}; delay is K x 512 bit transmission times
  - After second collision choose K from {0,1,2,3}…
  - After ten or more collisions, choose K from {0,1,2,3,4,…,1023}
- Under this scheme a new frame has a chance of sneaking in the first attempt, even in heavy traffic
- **Ethernet Efficiency**: under heavy traffic and large number of nodes:

$$Efficiency = \frac{1}{1 + (5 * \frac{t_{prop}}{t_{trans}})}$$

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 interface to its other interfaces, ie a physical layer device only!



10BaseT and 100BaseT

- 10/100 Mbps rate; latter called "fast ethernet"
- T stands for Twisted Pair
- Hub to which nodes are connected by twisted pair, thus "star topology"
- CSMA/CD implemented at the Hub
- Max distance from node to Hub is 100 meters
- Hub can disconnect a "jabbering adapter"; 10base2 would not work if an adapter does not stop transmitting on the cable
- Hub can gather monitoring information and statistics for display to LAN administrators

Gbit Ethernet

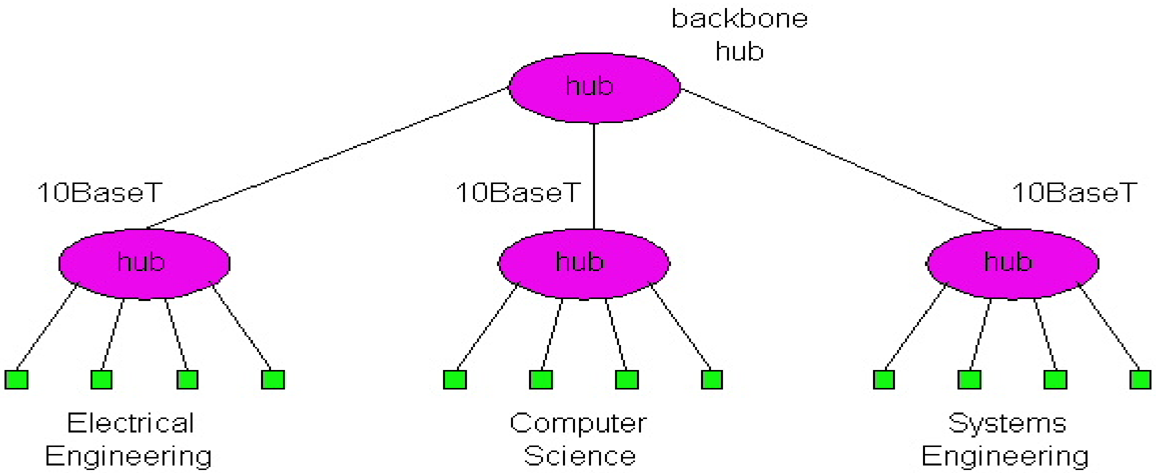
- Use standard Ethernet frame format
- Allows for Point-to-point links and shared broadcast channels
- In shared mode, CSMA/CD is used
- Full-Duplex at 1 Gbps for point-to-point links

Hubs, Bridges and Switches

- Used for extending LANs in terms of geographical coverage, number of nodes, administration capabilities, etc.
- Differ in regards to:
  - collision domain isolation
  - layer at which they operate

Hubs

- Physical Layer devices: essentially repeaters operating at bit levels: repeat received bits on one interface to all other interfaces
- Hubs can be arranged in a hierarchy (or **multi-tier design**), with a **backbone** hub at its top
- Each connected LAN is referred to as a LAN **segment**
- Hubs **do not isolate** collision domains: a node may collide with any node residing at any segment in the LAN



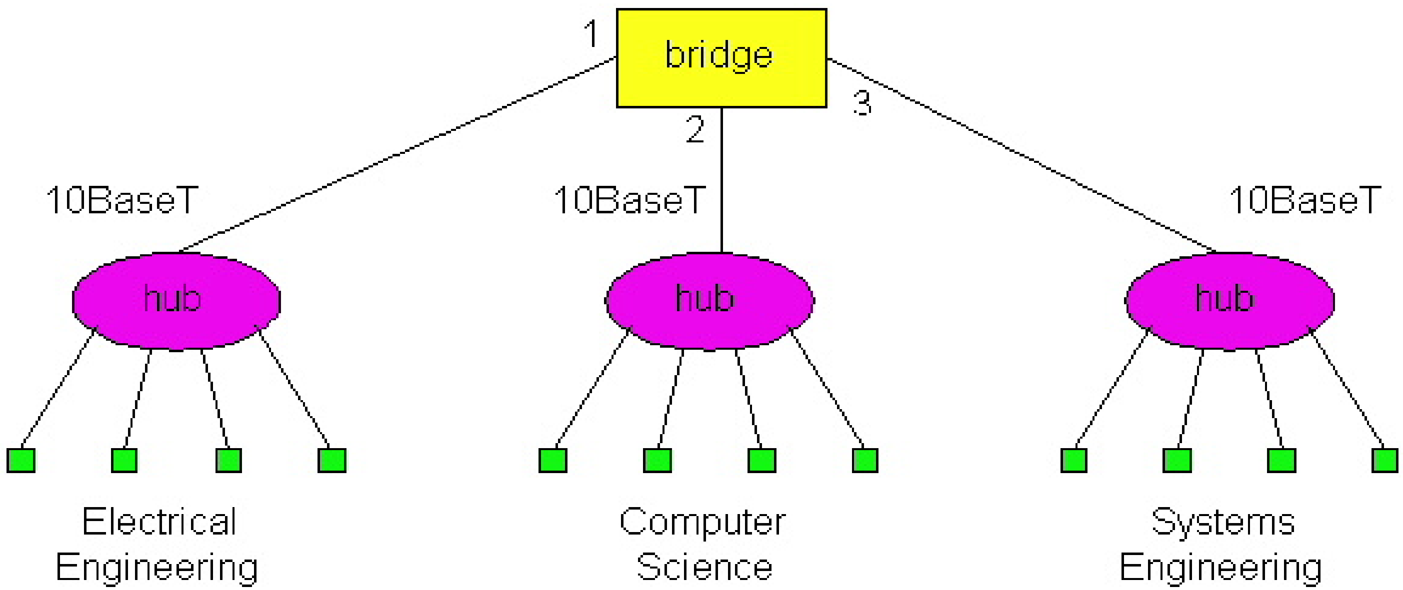
Hubs (Cont.)

- Hub Advantages:
  - \+ Simple, inexpensive device
  - \+ Multi-tier provides graceful degradation: portions of the LAN continue to operate if one of the hubs malfunction
  - \+ Extends maximum distance between node pairs (100m per Hub)
  - \+ Interdepartmental Communication
- Hub Limitations:
  - \- Single collision domain results in no increase in max throughput; the multi-tier throughput same as the the single segment throughput
  - \- Cannot connect different Ethernet types (eg 10BaseT and 100baseT)

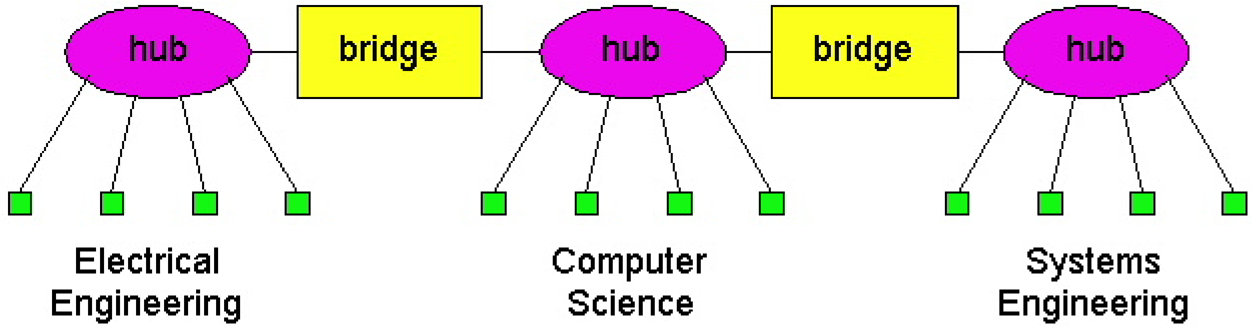
Bridges

- **Link Layer devices**: operate on Ethernet frames, examining the frame header and selectively forwarding a frame based on its destination
- Bridge **isolates collision** domains since it buffers frames
- When a frame is to be forwarded on a segment, the bridge uses CSMA/CD to access the segment and transmit
- Bridge advantages:
  - + Isolates collision domains resulting in higher total max throughput, and does not limit the number of nodes nor geographical coverage
  - + Can connect different type Ethernet since it is a store and forward device
  - + Transparent: no need for any change to hosts LAN adapters

Backbone Bridge



Interconnection Without Backbone



- **Not recommended** for two reasons:
  - Single point of failure at Computer Science hub
  - All traffic between EE and SE must path over CS segment

Bridge Filtering

- Bridges learn which hosts can be reached through which interfaces and maintain filtering tables (bridge tables)
- A filtering table entry:

  (Node LAN Address, Bridge Interface, Time Stamp)

  where Node LAN Address is the 6 byte physical address
- Filtering procedure:

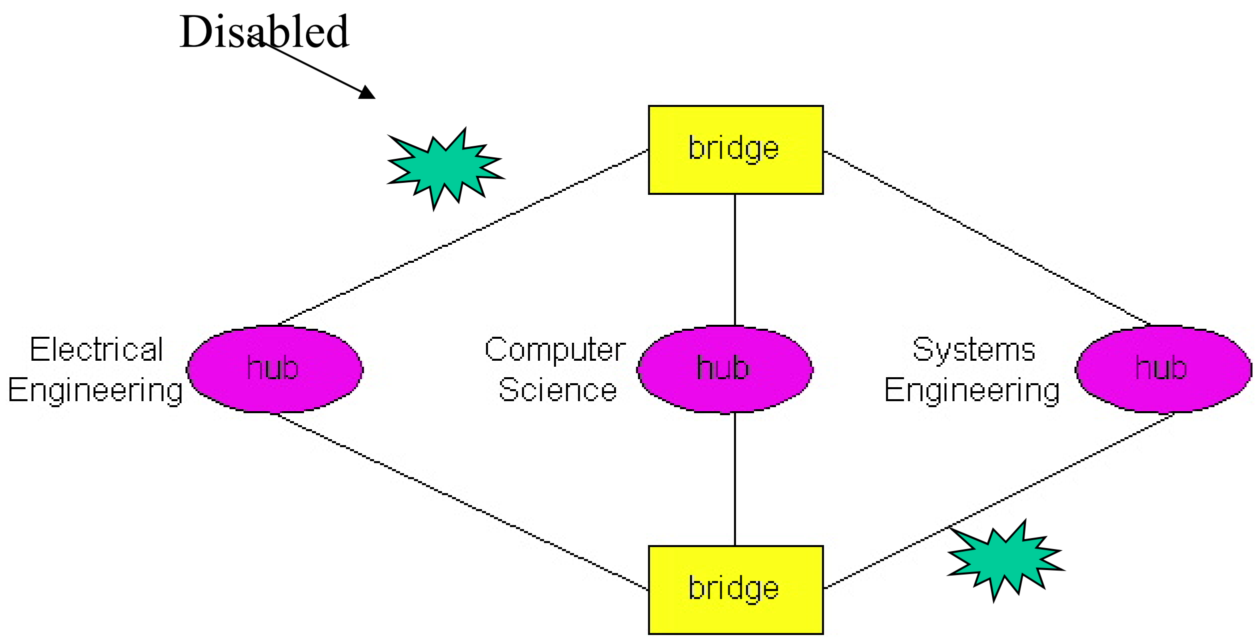
```
if destination is on LAN on which frame was received
    then drop the frame
    else { lookup filtering table
          if entry found for destination
             then forward the frame on interface indicated;
             else flood; /* forward on all but the interface on
               which the frame arrived*/
         }
```

Bridge Learning

- When a frame is received, the bridge "learns" from the source address and updates its filtering table
  (Node LAN Address, Bridge Interface, Time Stamp)
- Stale entries in the Filtering Table are dropped (TTL can be 60 minutes)

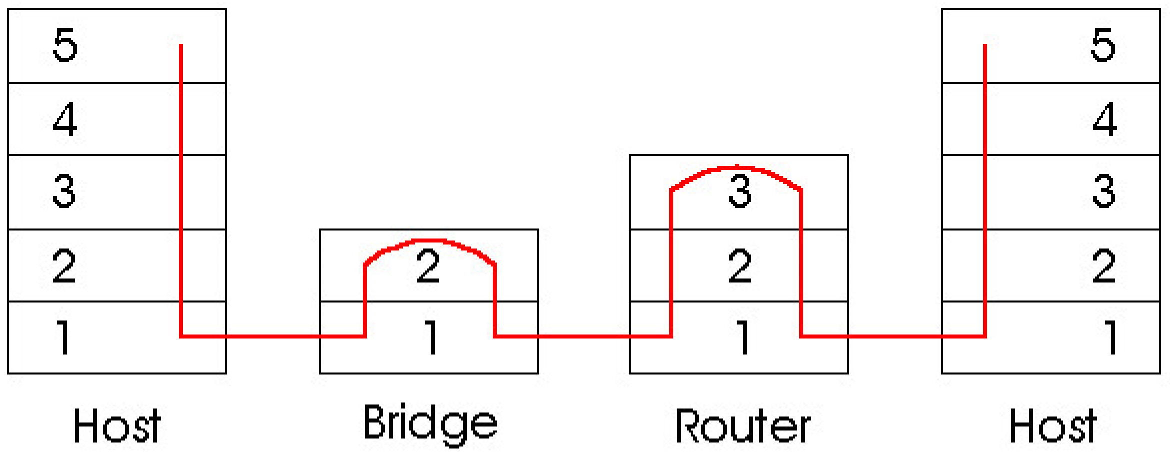
Bridges Spanning Tree

- For increased reliability, it is desirable to have redundant, alternate paths from a source to a destination
- With multiple simultaneous paths however, cycles result on which bridges may multiply and forward a frame forever
- Solution is organizing the set of bridges in a spanning tree by disabling a subset of the interfaces in the bridges:



Bridges Vs. Routers

- Both are store-and-forward devices, but Routers are Network Layer devices (examine network layer headers) and Bridges are Link Layer devices
- Routers maintain routing tables and implement routing algorithms, bridges maintain filtering tables and implement filtering, learning and spanning tree algorithms



Routers Vs. Bridges (Cont)

- Bridges + and -

+ Bridge operation is simpler requiring less processing bandwidth

- Topologies are restricted with bridges: a spanning tree must be built to avoid cycles

- Bridges do not offer protection from broadcast storms (endless broadcasting by a host will be forwarded by a bridge)

- Routers + and -

+ Arbitrary topologies can be supported, cycling is limited by TTL counters

+ Provide firewall protection against broadcast storms

- Require IP address configuration (not plug and play)

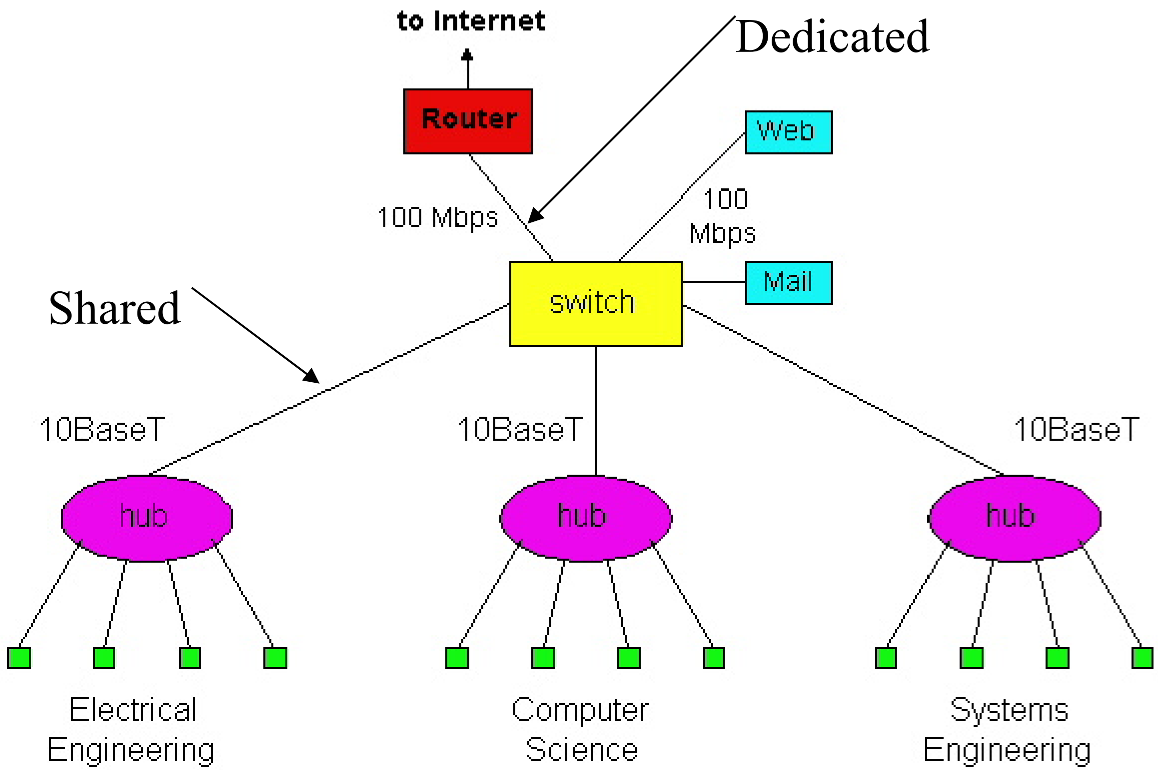
- Require higher processing bandwidth

- Bridges do well in small (few hundred hosts) while routers are required in large networks (thousands of hosts)

Ethernet Switches

- A switch is a device that incorporates bridge functions as well as point-to-point 'dedicated connections'
- A host attached to a switch via a dedicated point-to-point connection; will always sense the medium as idle; no collisions ever!
- Ethernet Switches provide a combinations of shared/dedicated, 10/100/1000 Mbps connections
- Some E-net switches support cut-through switching: frame forwarded immediately to destination without awaiting for assembly of the entire frame in the switch buffer; slight reduction in latency
- Ethernet switches vary in size, with the largest ones incorporating a high bandwidth interconnection network

Ethernet Switches (Cont)



Point to Point protocol (PPP)

- Point to point, wired data link easier to manage than broadcast link: no Media Access Control
- Several Data Link Protocols: PPP, HDLC…
- PPP (Point to Point Protocol) is very popular: used in dial up connection between residential Host and ISP; on SONET/SDH connections, etc
- PPP is extremely simple (the simplest in the Data Link protocol family) and very streamlined

PPP requirements

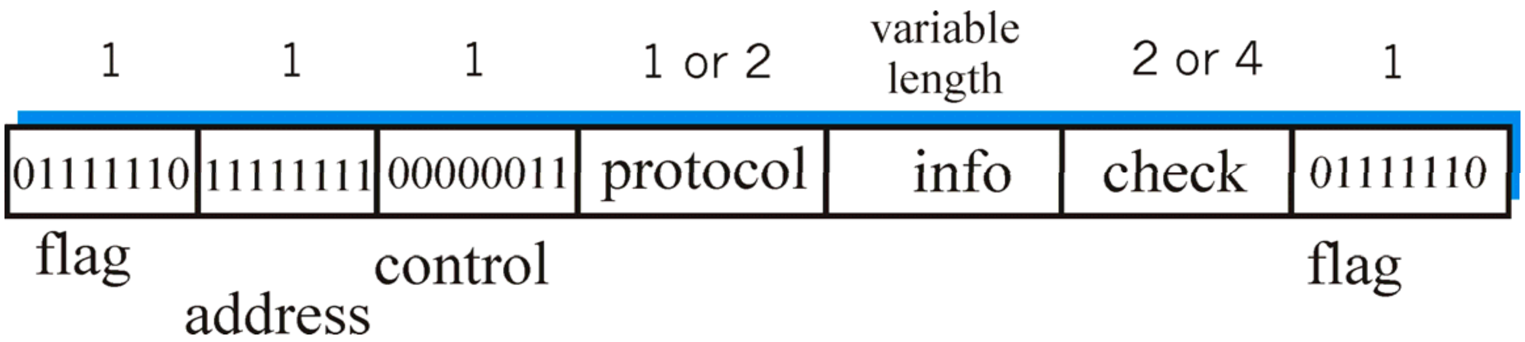
- Pkt framing: encapsulation of packets
- bit transparency: must carry any bit pattern in the data field
- error detection (no correction)
- multiple network layer protocols
- connection liveness
- Network Layer Address negotiation: Hosts/nodes across the link must learn/configure each other's network address

PPP non-requirements

- error correction/recovery
- flow control
- sequencing
- multipoint links (eg, polling)

PPP Data Frame

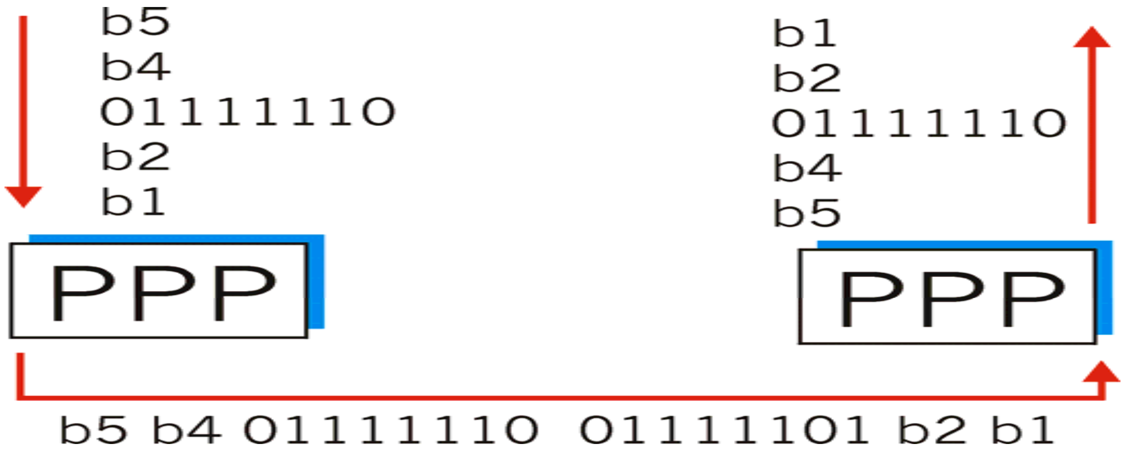
- Flag: delimiter (framing)
- Address: does nothing (only one option)
- Control: does nothing; in the future possible multiple control fields
- Protocol: upper layer to which frame must be delivered (eg, PPP-LCP, IP, IP-CP, etc)



Byte Stuffing

- For "data transparency", the data field must be allowed to include the pattern <01111110> ; ie, this must not be interpreted as a flag
- to alert the receiver, the transmitter "stuffs" an extra < 01111101> byte after each < 01111110> data byte
- the receiver discards each 01111101 after 01111110, and continues

  data reception



PPP Link Control Protocol

- PPP-LCP establishes/releases the PPP connection; negotiates options
- Starts in DEAD state
- LCP Options: max frame length; authentication protocol
- Once PPP link established, IP-CP (Contr Prot) moves in (on top of PPP) to configure IP network addresses etc.