CS5222 Final Review (week 13)

Prof Weifa Liang

Weifa.liang@cityu-dg.edu.cn

Application Layer

Goal:

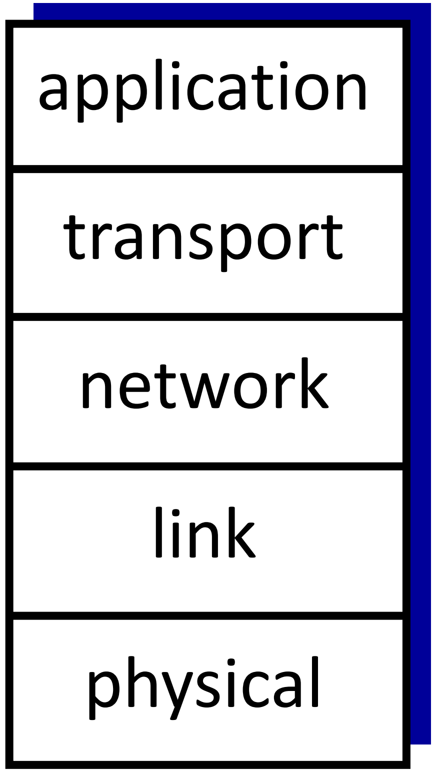
- Overview of "big picture," introduction to terminology
  - more depth & details *later*
- Approach:
  - use the Internet as an example

Roadmap:

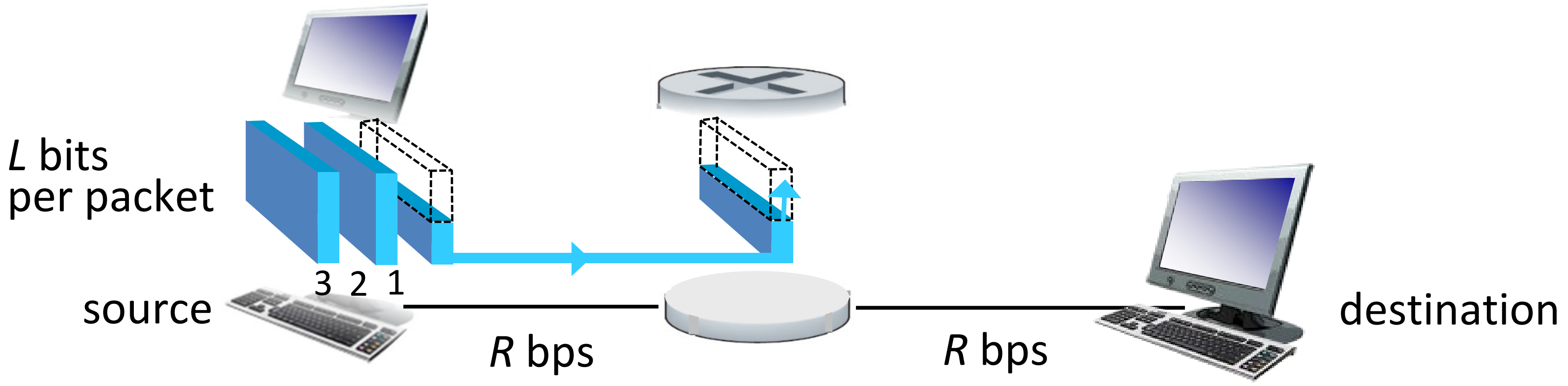
- What *is* the Internet?
- What *is* a protocol?
- Network edge: hosts, access network, physical media
- Network core: packet/circuit switching, internet structure
- Performance: loss, delay, throughput
- Protocol layers, service models

Internet protocol stack

- *application:* supporting network applications
  - HTTP, IMAP, SMTP, …
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits “on the wire”



Packet-switching: store-and-forward

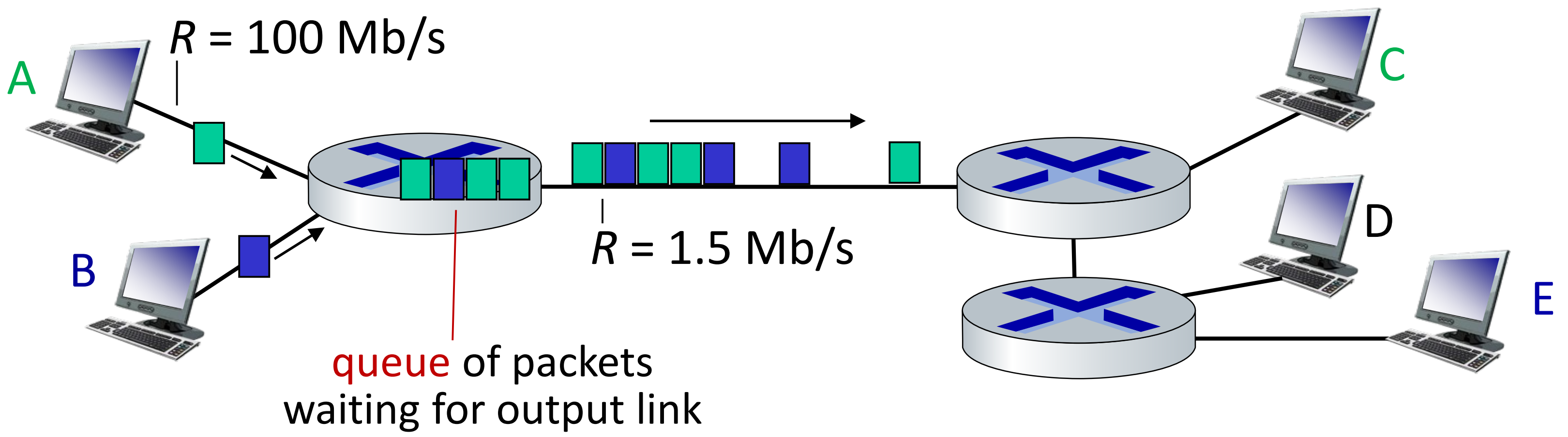


- *Store and forward: the entire* packet must arrive at a router before it can be transmitted on next link
- Transmission delay: takes L/R seconds to transmit (push out) a L-bit packet into a link at R bps
- *End-end delay:* $2L/R$ (above), assuming a zero propagation delay (more on delay shortly)

One-hop numerical example:

- L = 10 Kbits
- R = 100 Mbps
- one-hop transmission delay = 0.1 msec

Packet-switching: queueing delay, loss



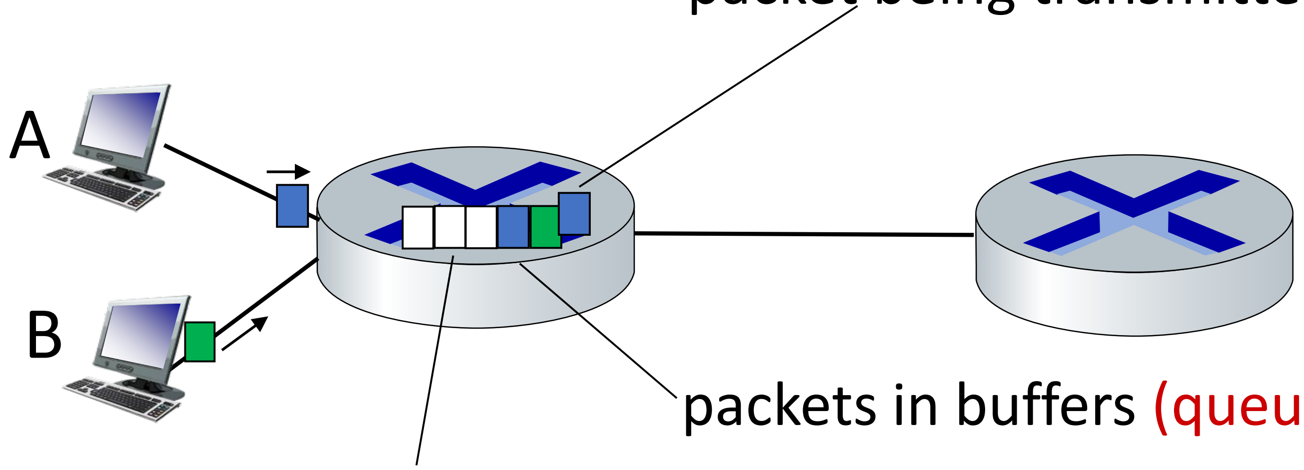
Packet queuing and loss: if the arrival rate (in bps) to a link exceeds the transmission rate (bps) of the link for a period of time:

- packets will queue, waiting to be transmitted on the output link
- packets can be dropped (lost) if memory (buffer) in the router fills up

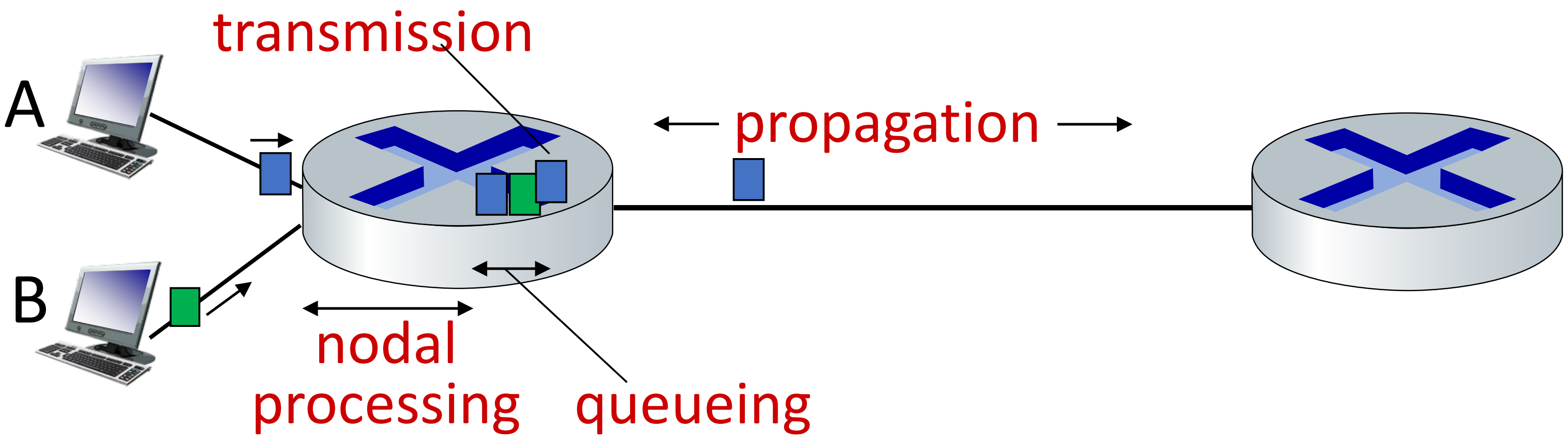
How do packet loss and delay occur?

packets *queue* in router buffers

- packets queue, wait for turn
- arrival rate to link (temporarily) exceeds output link capacity: packet loss



Packet delay: four sources



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

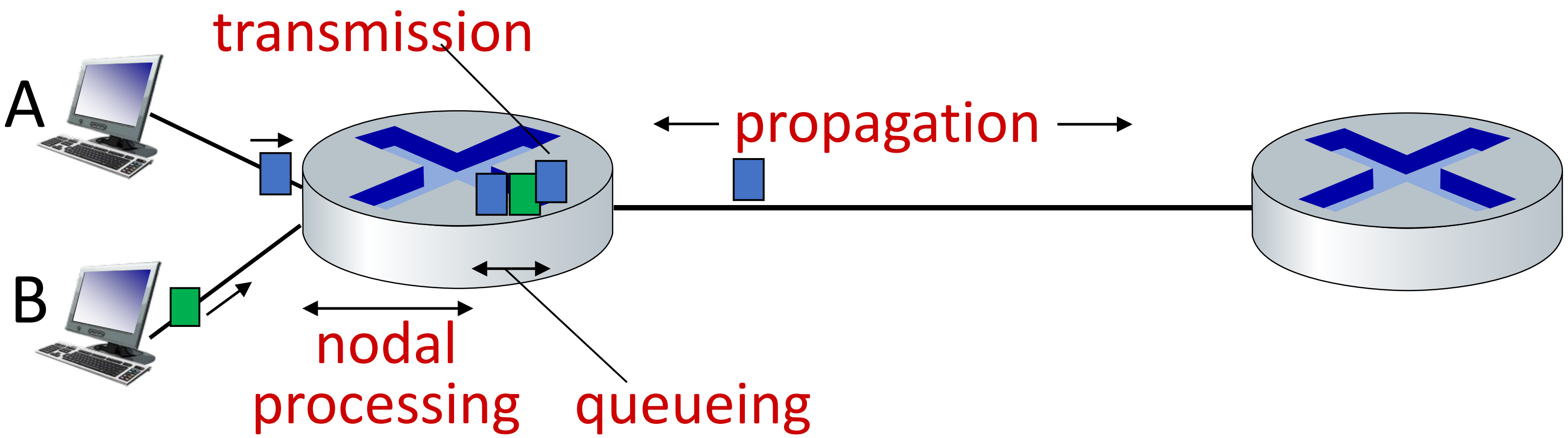
d_{proc}: nodal processing

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

d_{queue}: queueing delay

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

Packet delay: four sources



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

d_{trans}: transmission delay:

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

d_{prop}: propagation delay:

- d: length of physical link
- s: propagation speed (~2x10^8 m/sec)
- $d_{\text{prop}} = d/s$

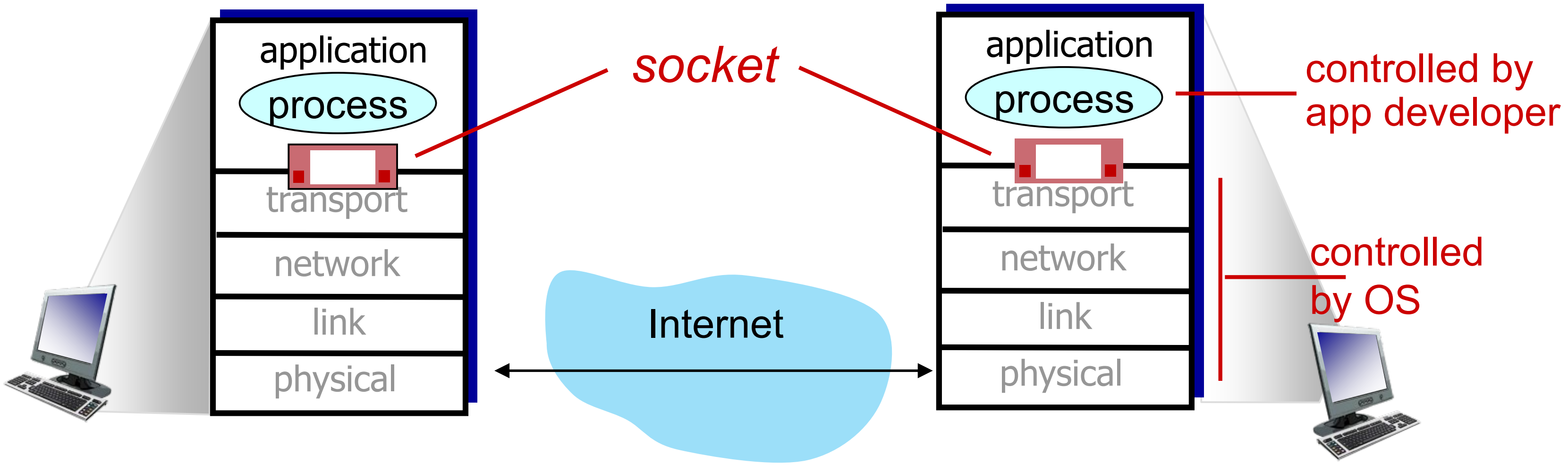
d_{trans} and d_{prop}
very different

Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks

Sockets

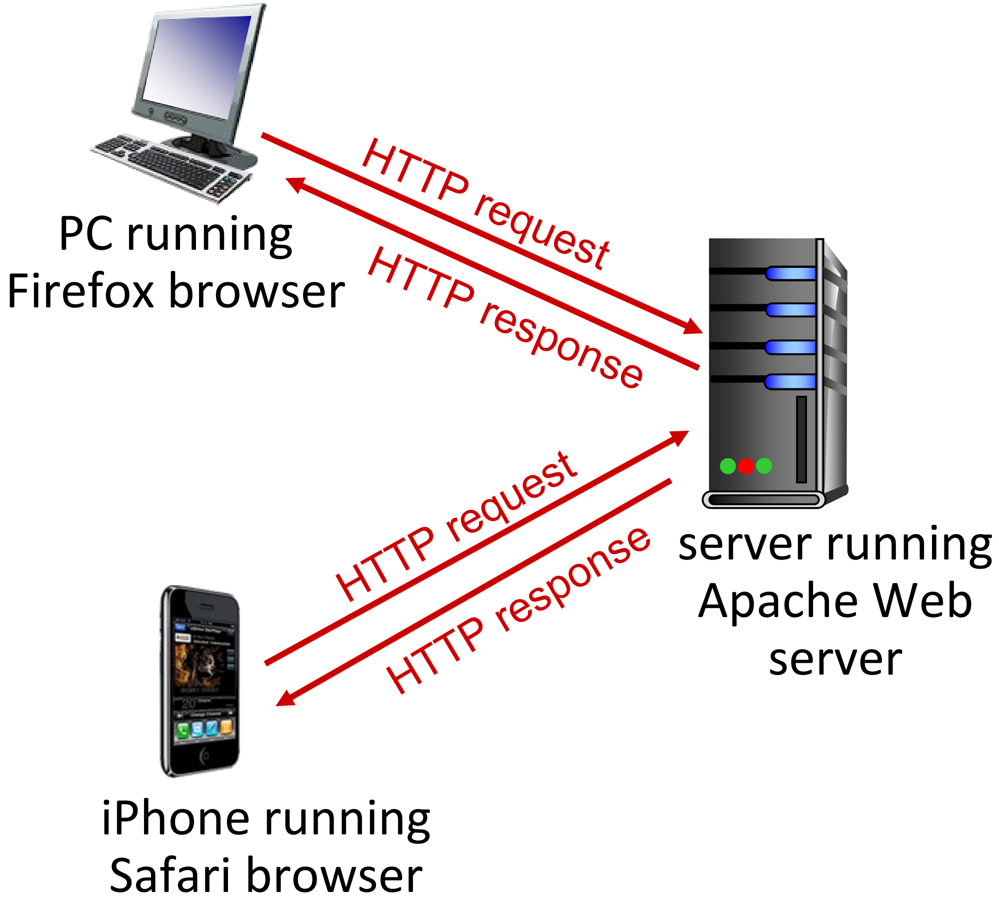
- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message "out the door"
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - two sockets involved: one on each side



HTTP overview

HTTP: hypertext transfer protocol

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



HTTP overview (continued)

HTTP uses TCP:

- 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 Web 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 connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

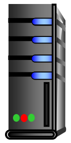
- TCP connection opened
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-persistent HTTP: example

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**

(containing text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80



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. Message indicates that client wants object someDepartment/home.index

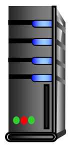
3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP: example (cont.)

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**

(containing text, references to 10 jpeg images)



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

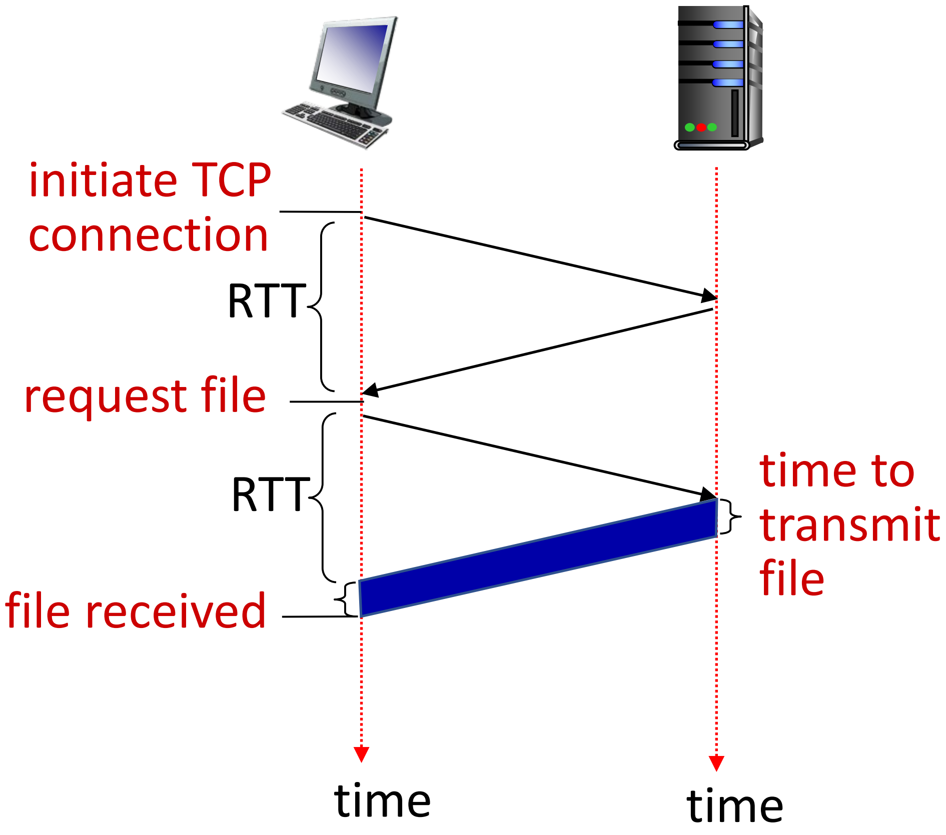
time

Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- obect/file transmission time



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

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

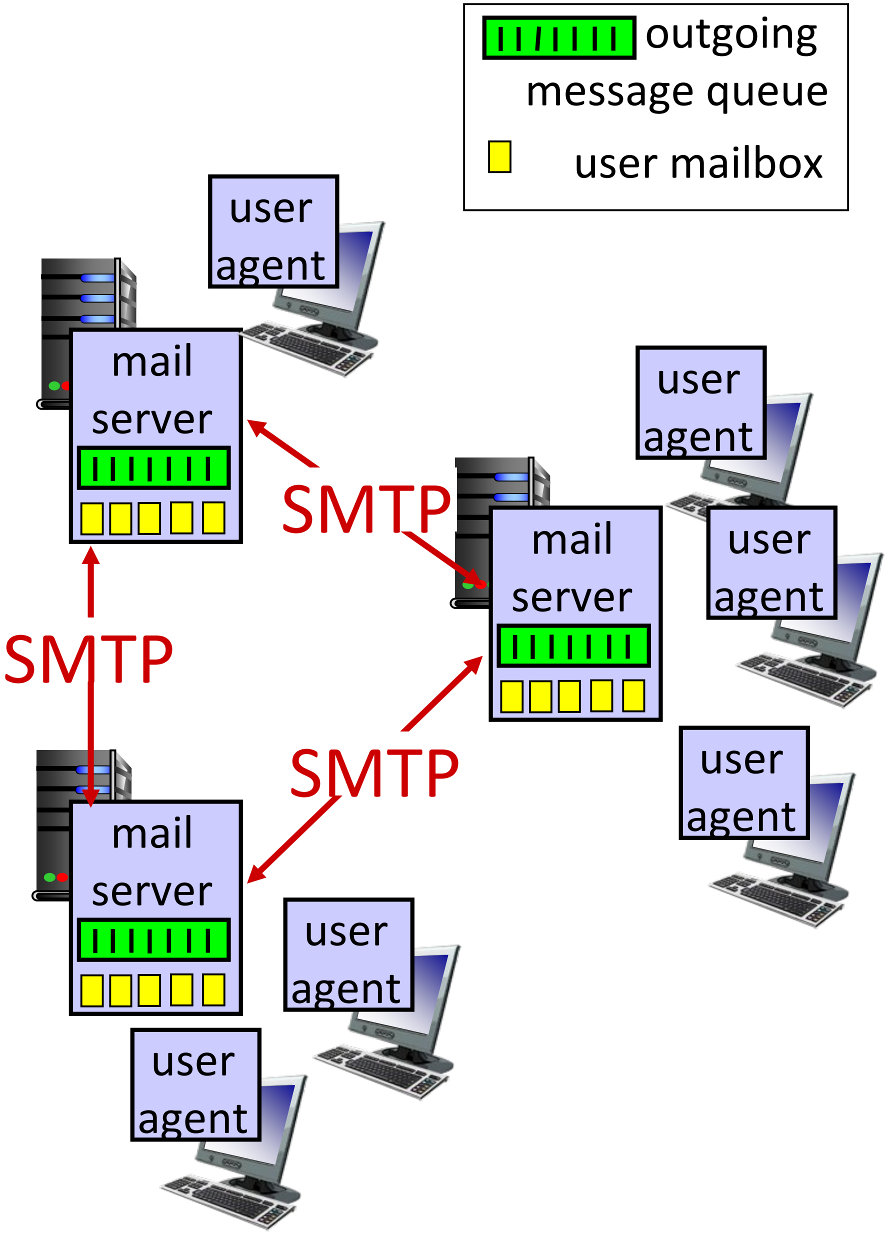
Persistent HTTP (HTTP1.1):

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects (cutting response time in half)

E-mail

Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

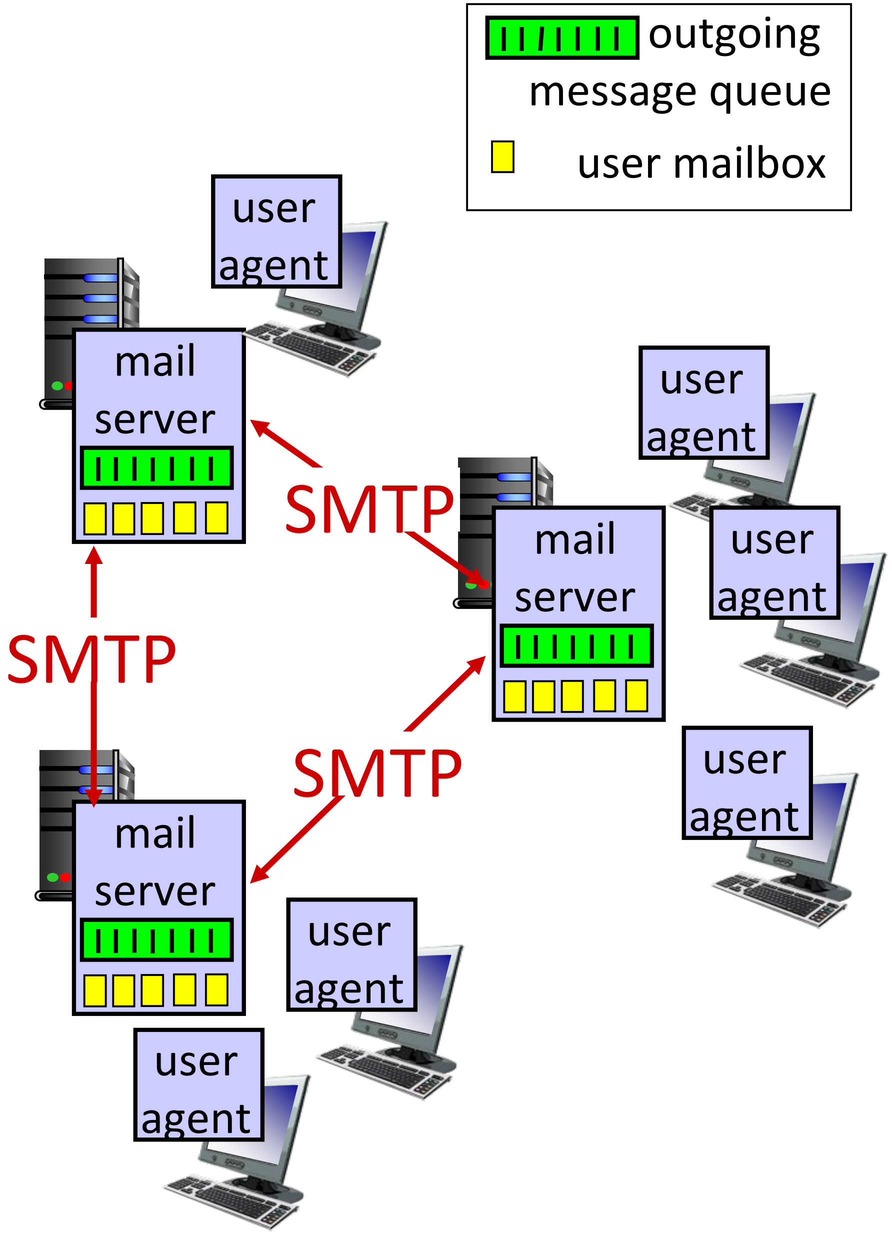
User Agent
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



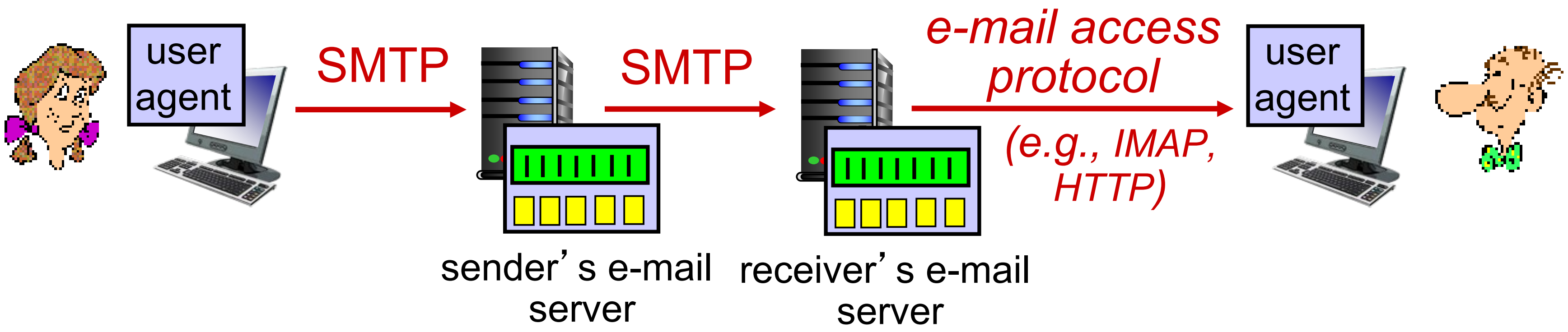
E-mail: mail servers

mail servers:

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



Mail access protocols



- SMTP: delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
  - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP) to retrieve e-mail messages

DNS: Domain Name System

Internet hosts, routers:

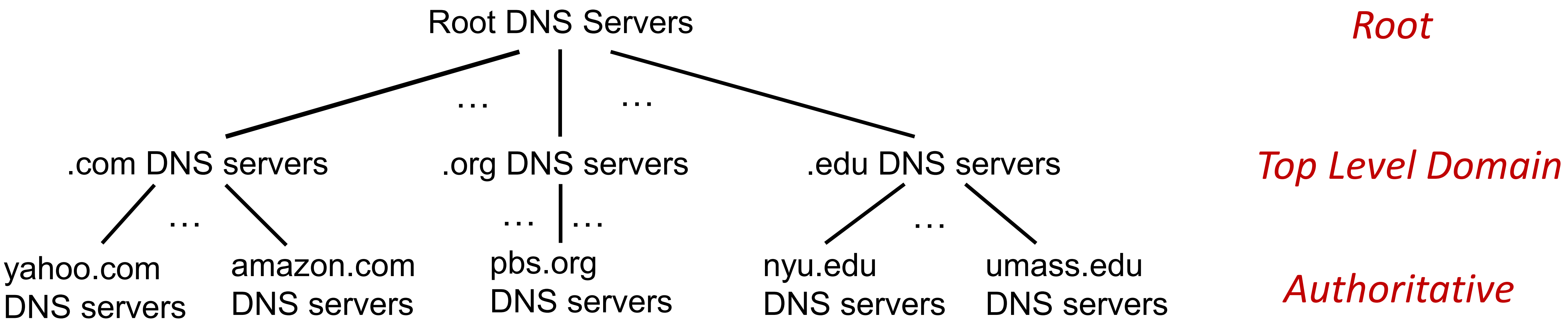
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., cityu.edu.hk used by humans

Q: how to map name to IP address?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - provides core Internet function, but *implemented as application-layer protocol*
  - complexity at network's "edge"

DNS: a distributed, hierarchical database



Client wants IP address for www.amazon.com; (1st approximation):

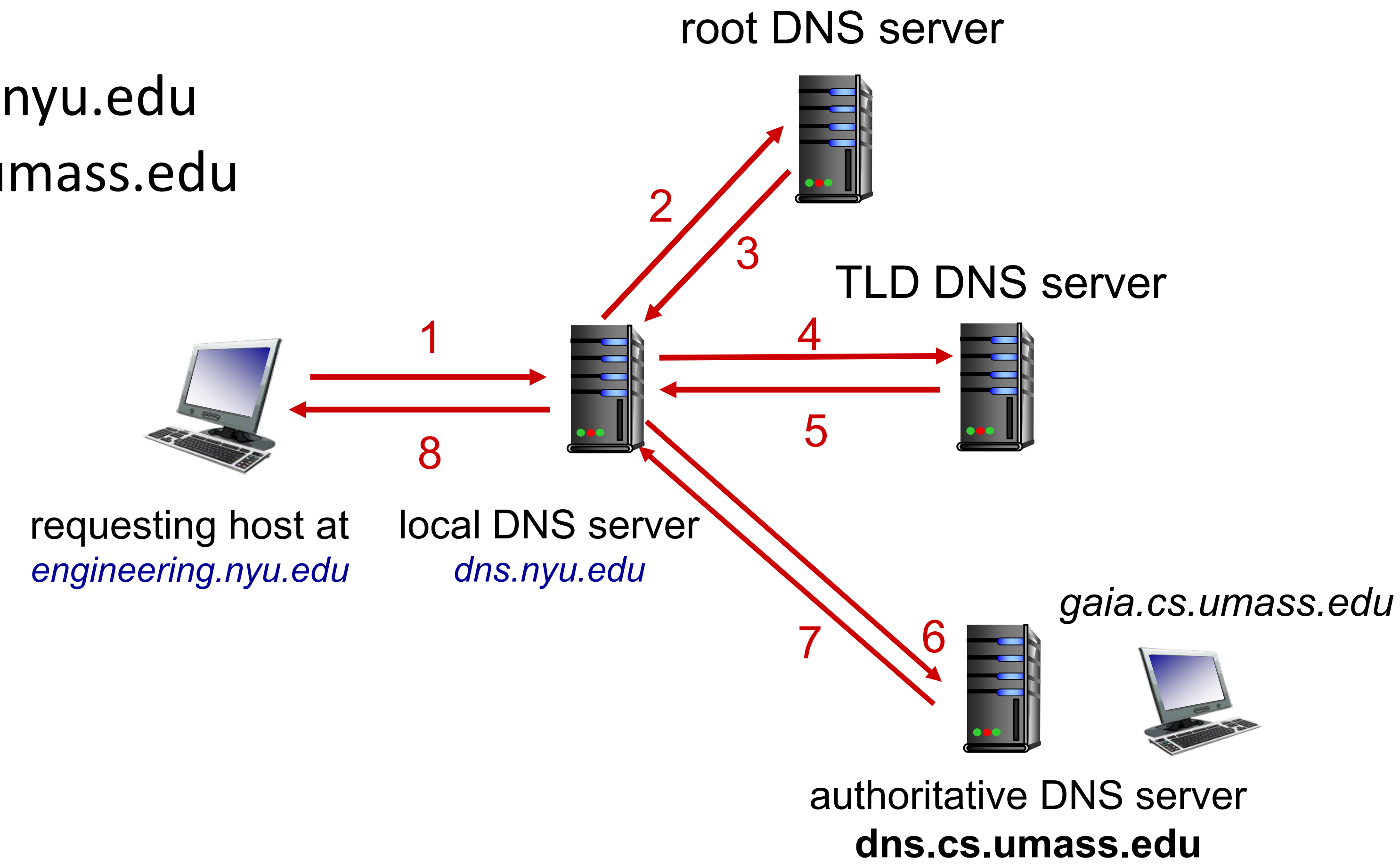
- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS name resolution: iterated query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Iterated query:

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

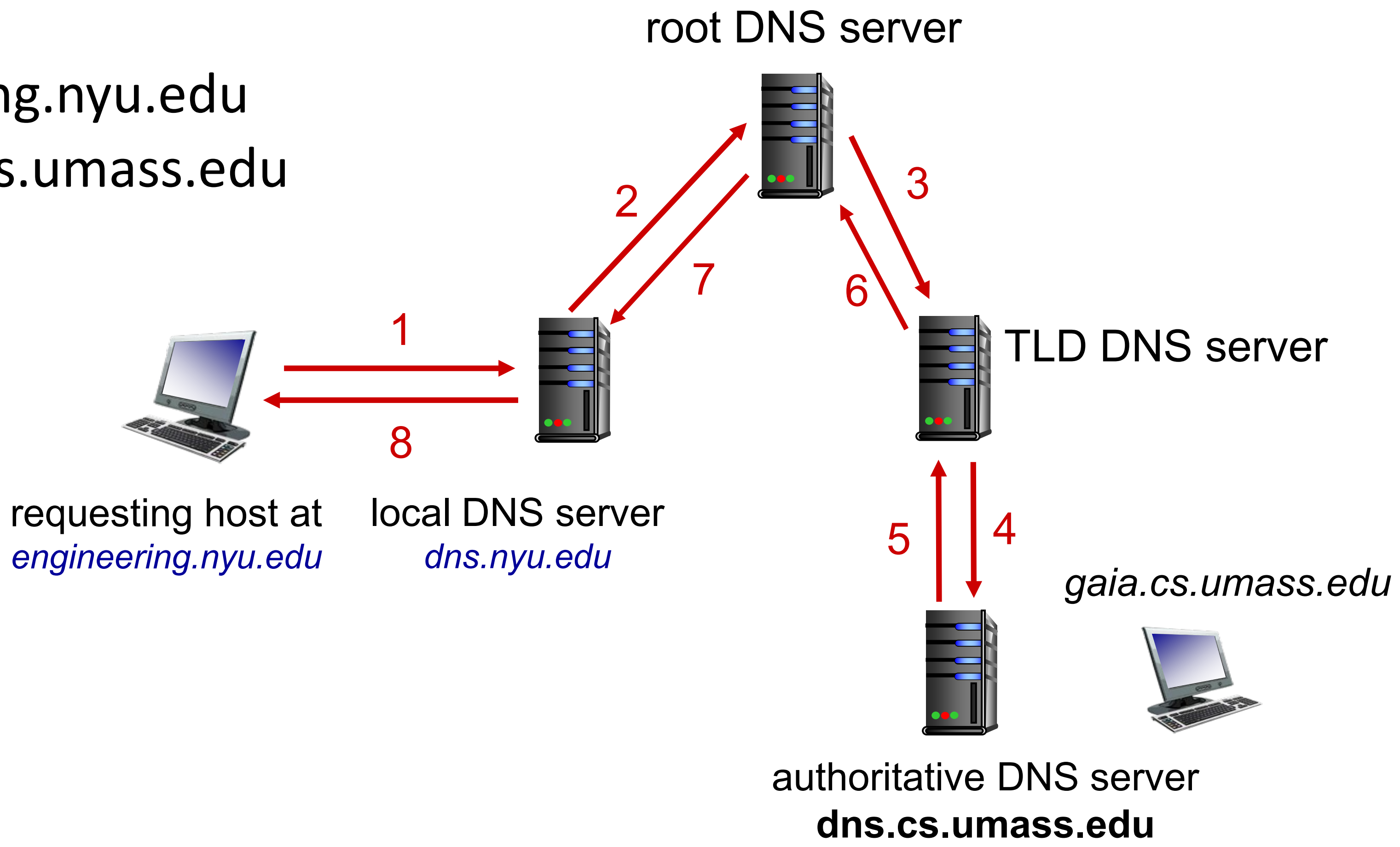


DNS name resolution: recursive query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



Transport layer: overview

Our goal:

- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

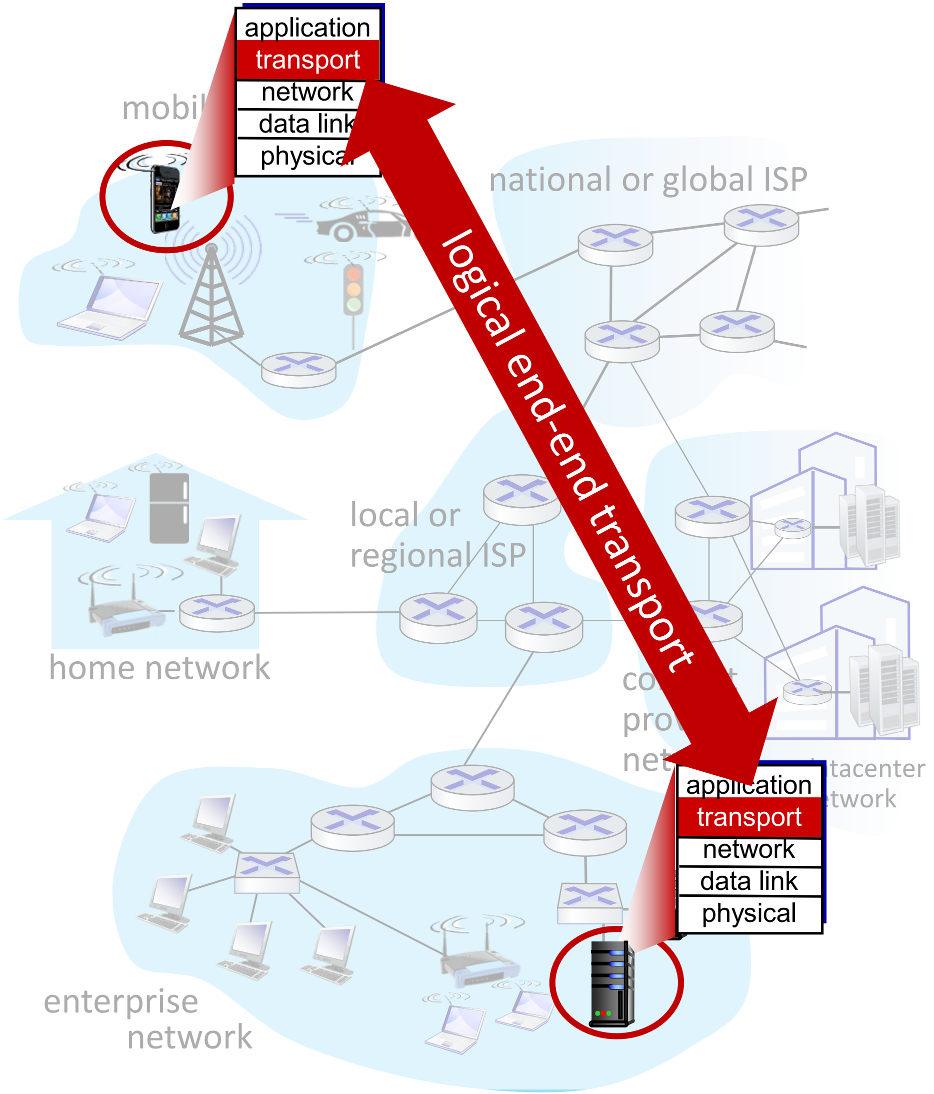
Transport vs. network layer services and protocols

- network layer:
  logical communication between *hosts*

- transport layer*:*
  logical communication between *processes*

Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



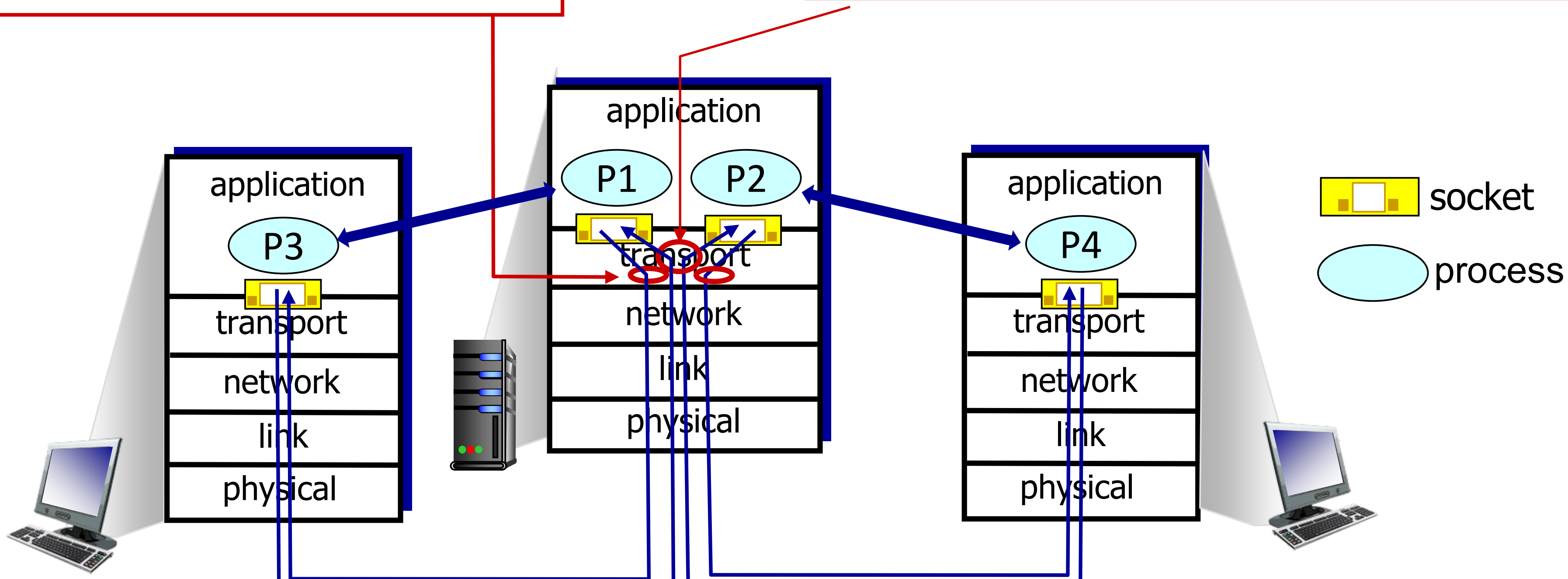
Multiplexing/demultiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

use header info to deliver received segments to correct socket

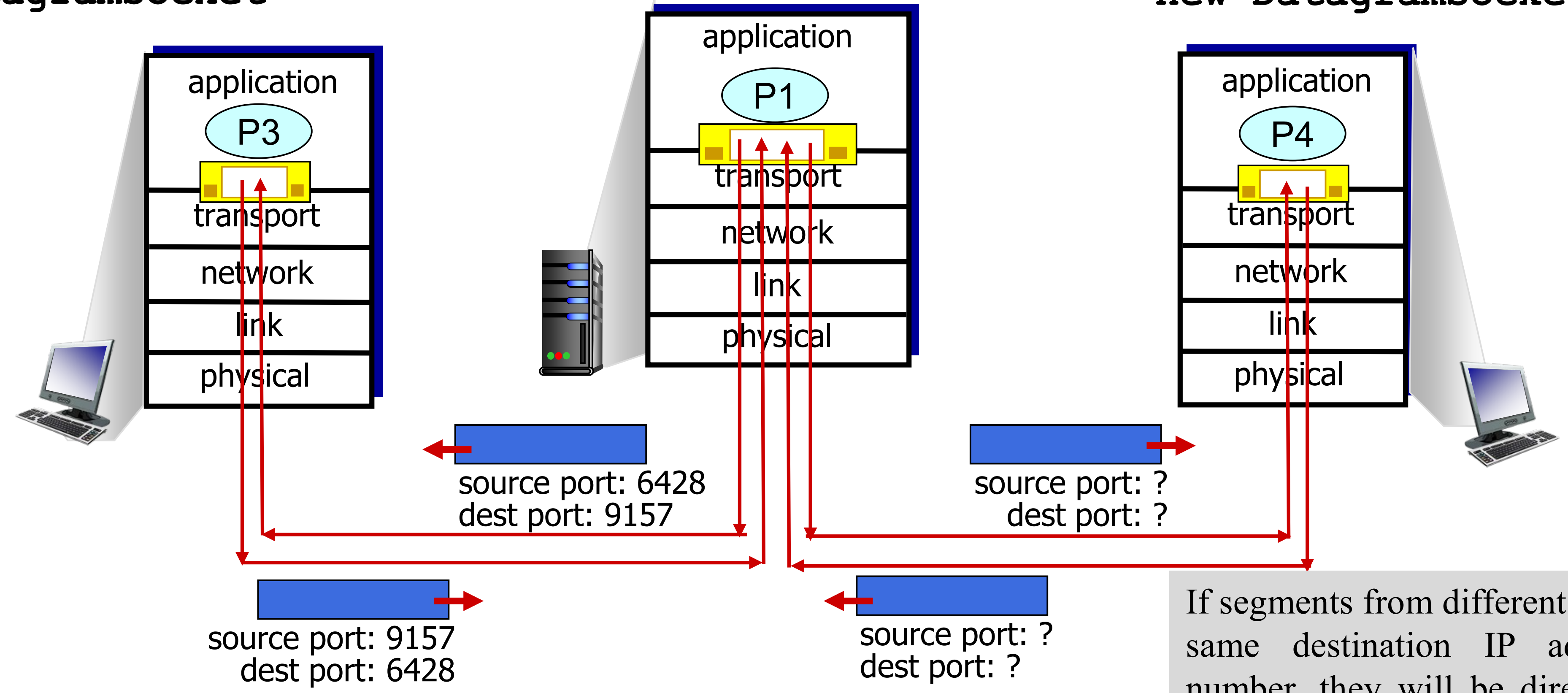


Connectionless demultiplexing: an example

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

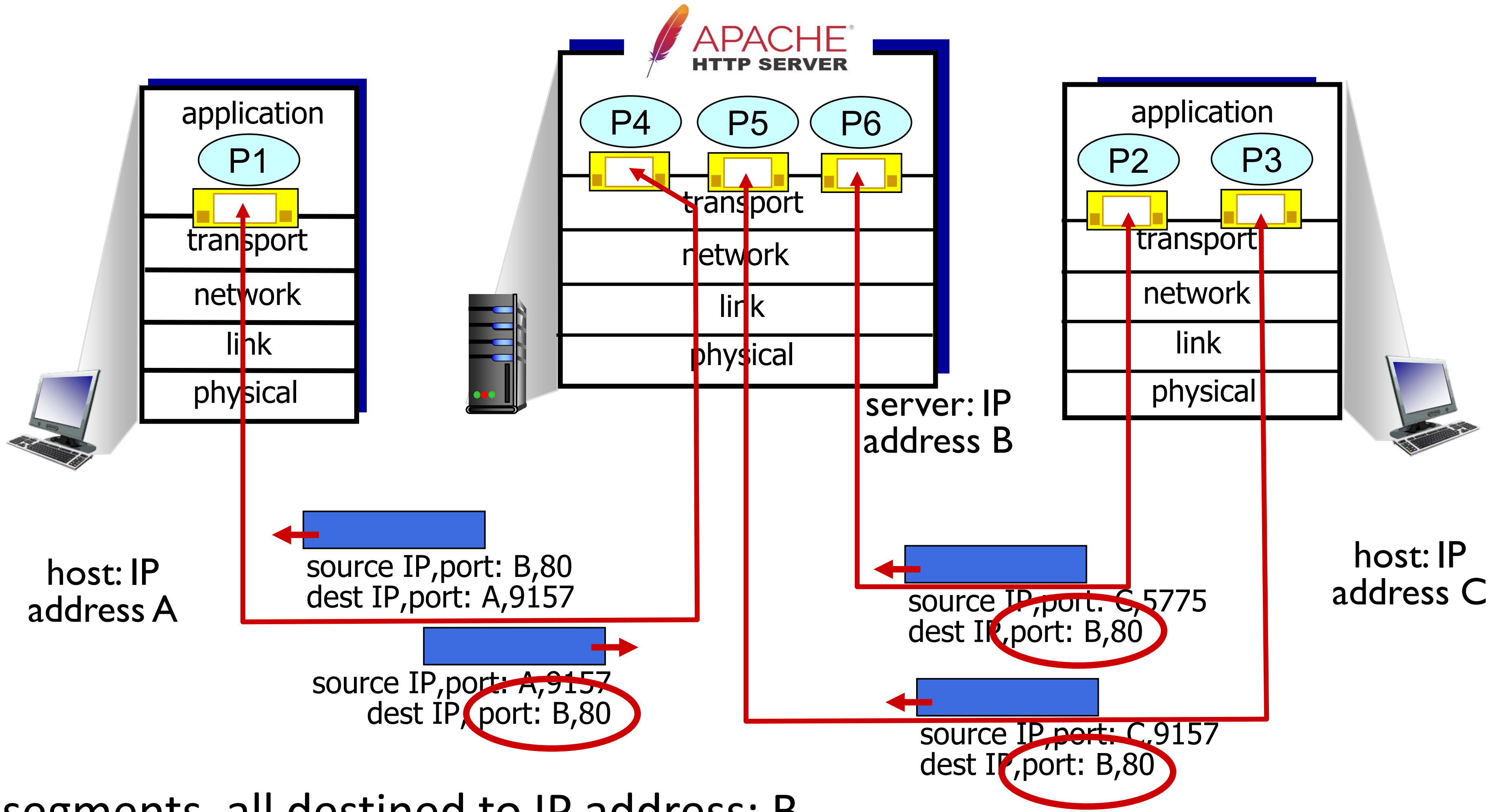
```
DatagramSocket
 serverSocket = new
 DatagramSocket
 (6428);
```

```
DatagramSocket mySocket1 =
 new DatagramSocket (5775);
```



If segments from different IP addresses have same destination IP address and port number, they will be directed to the same process

Connection-oriented demultiplexing: example

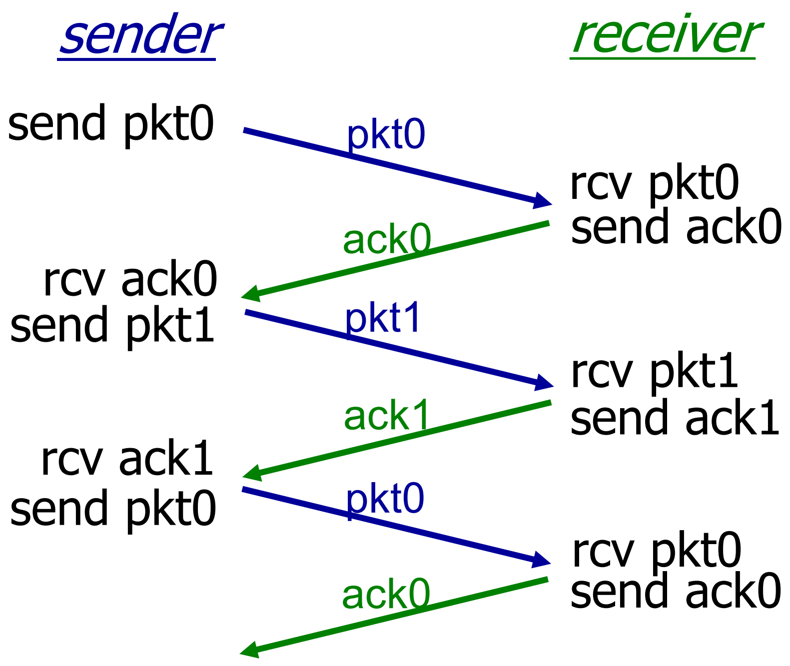


Three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

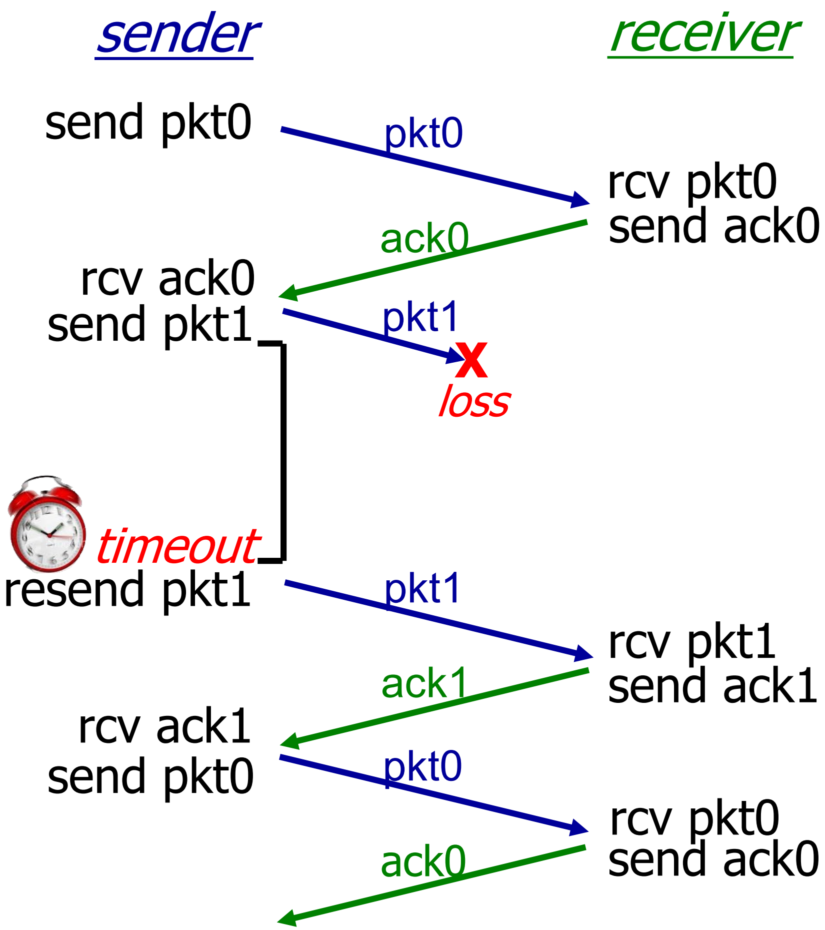
Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at *all* layers

The Stop-and-Wait Protocol

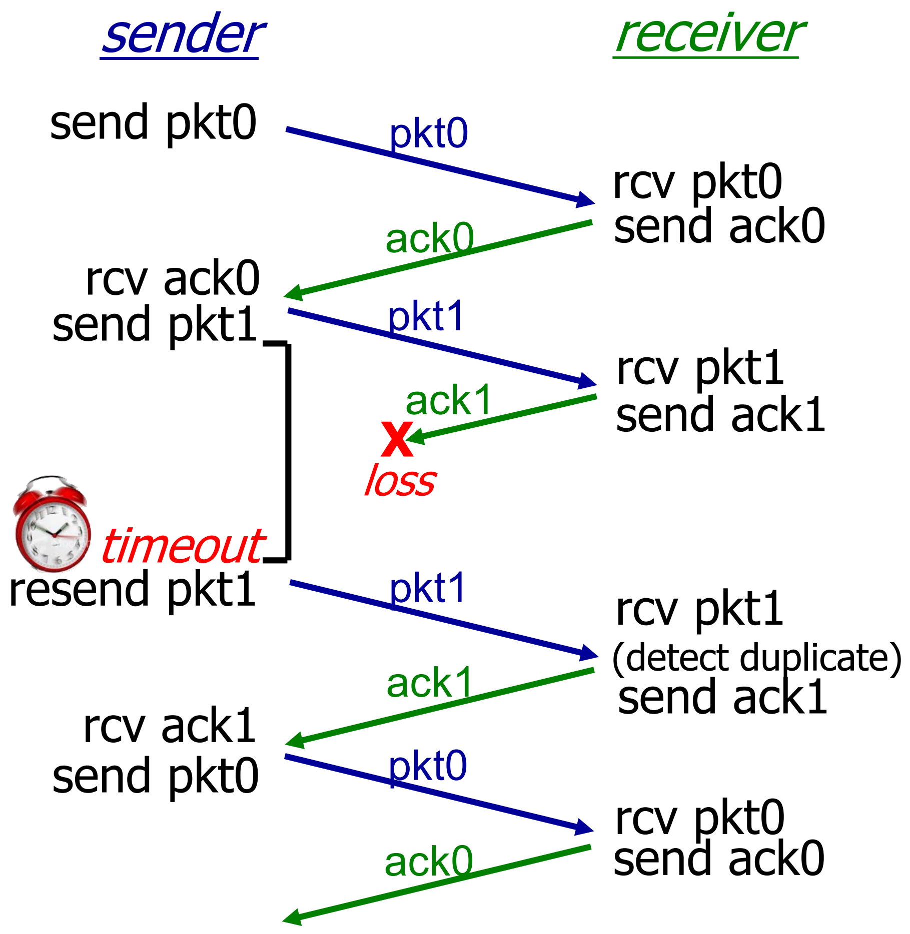


(a) no loss

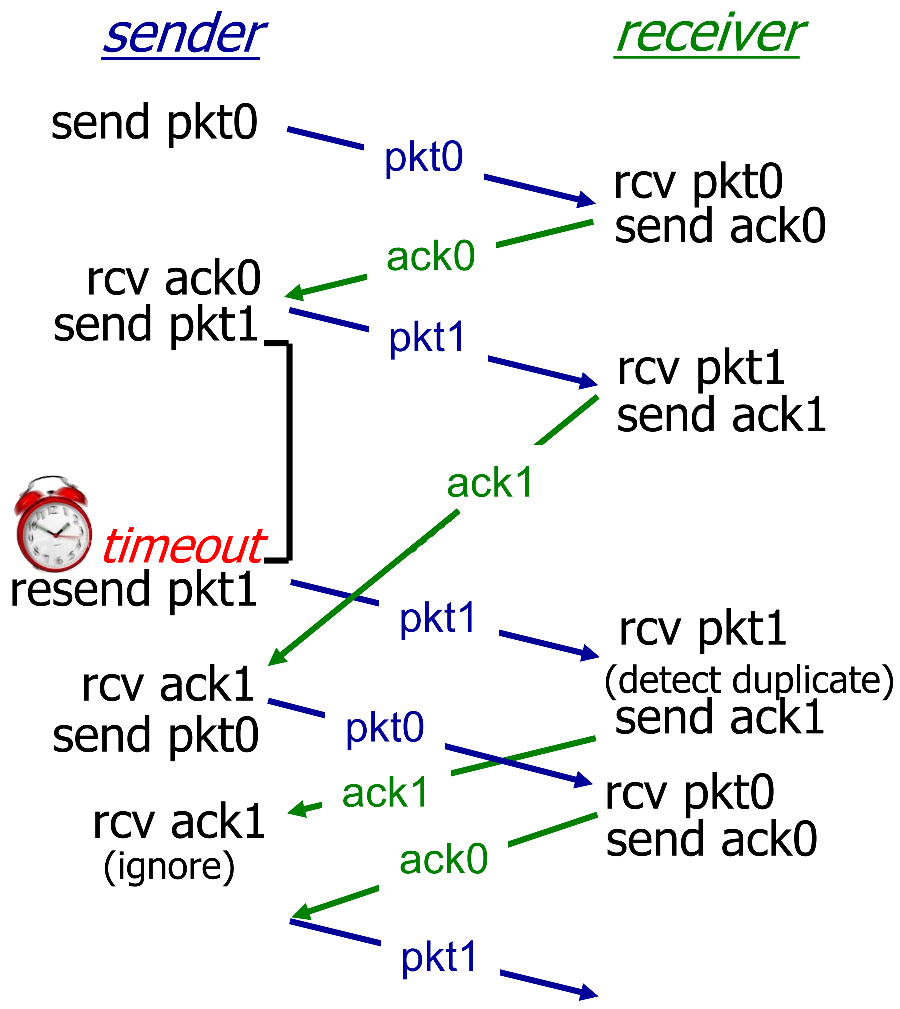


(b) packet loss

The Stop-and-Wait Protocol

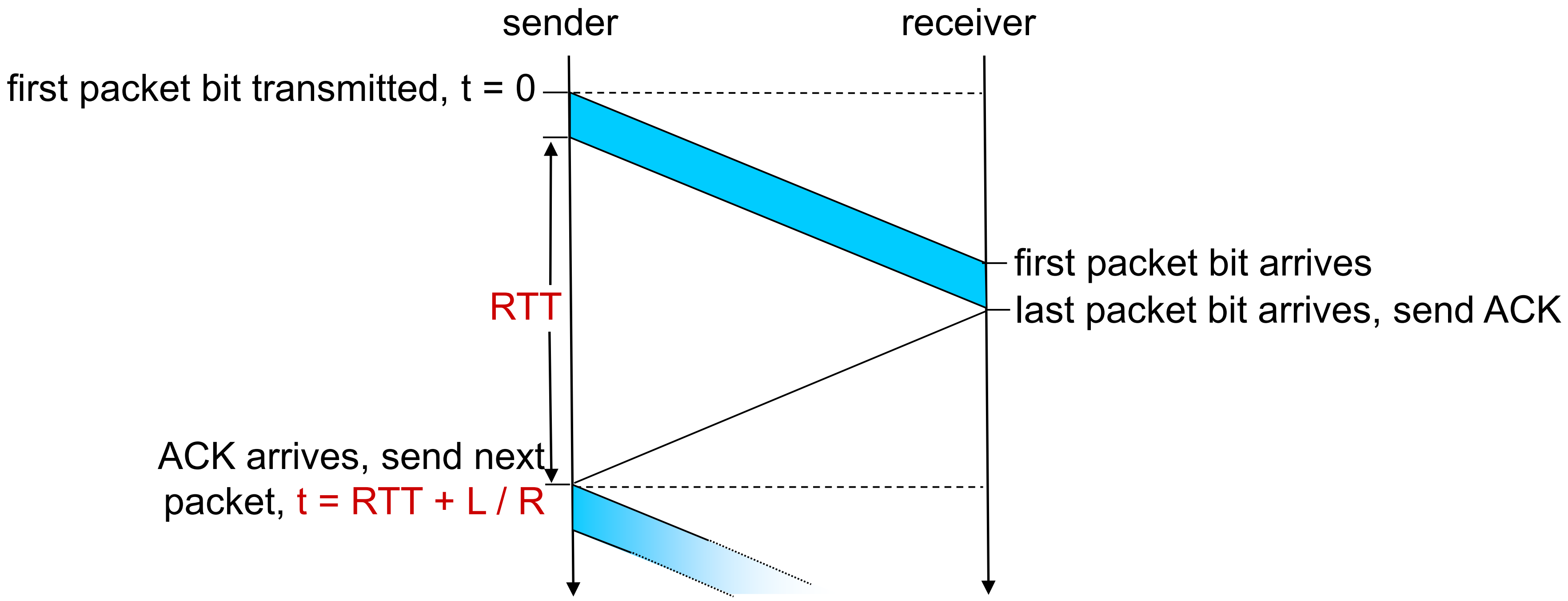


(c) ACK loss



(d) premature timeout/ delayed ACK

stop-and-wait operation



Pipelined protocols: overview

Go-back-N:

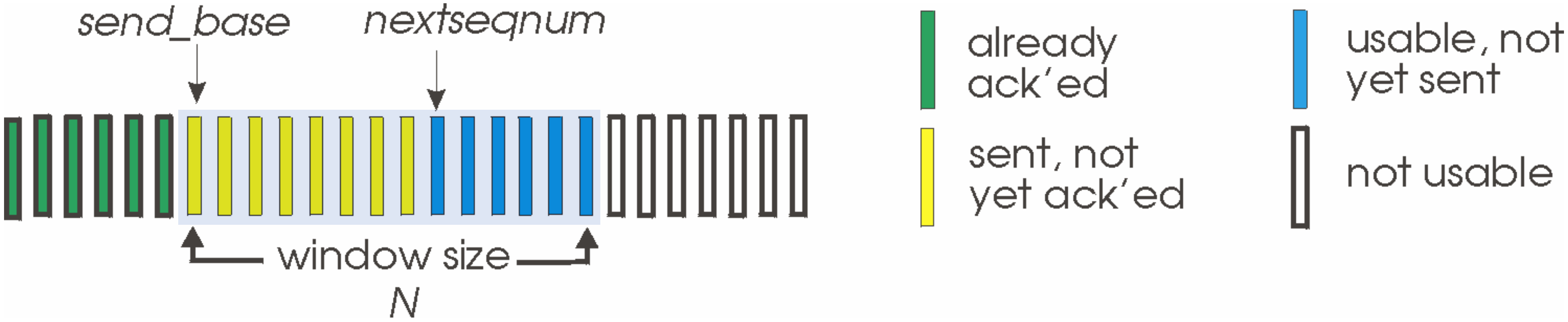
- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- receiver sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet

Go-Back-N: sender

- sender: "window" of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header

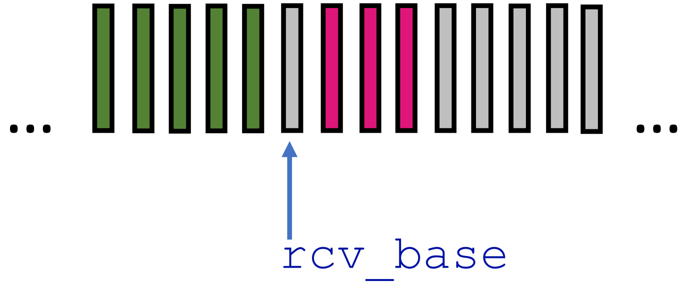


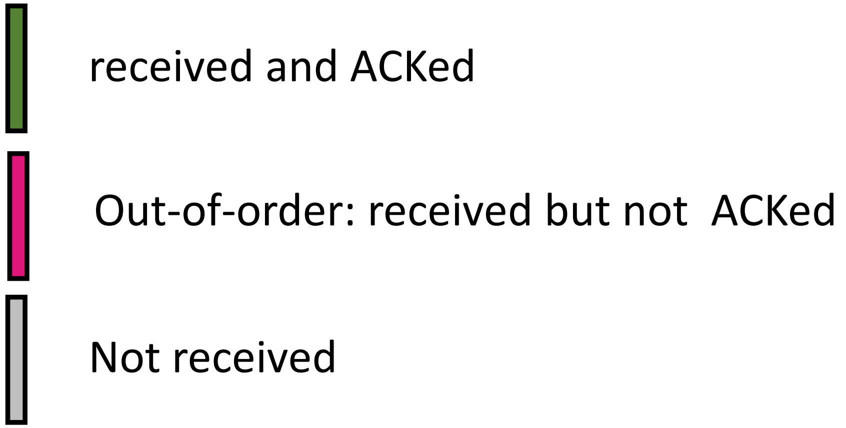
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight/unacked packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

Go-Back-N: receiver side

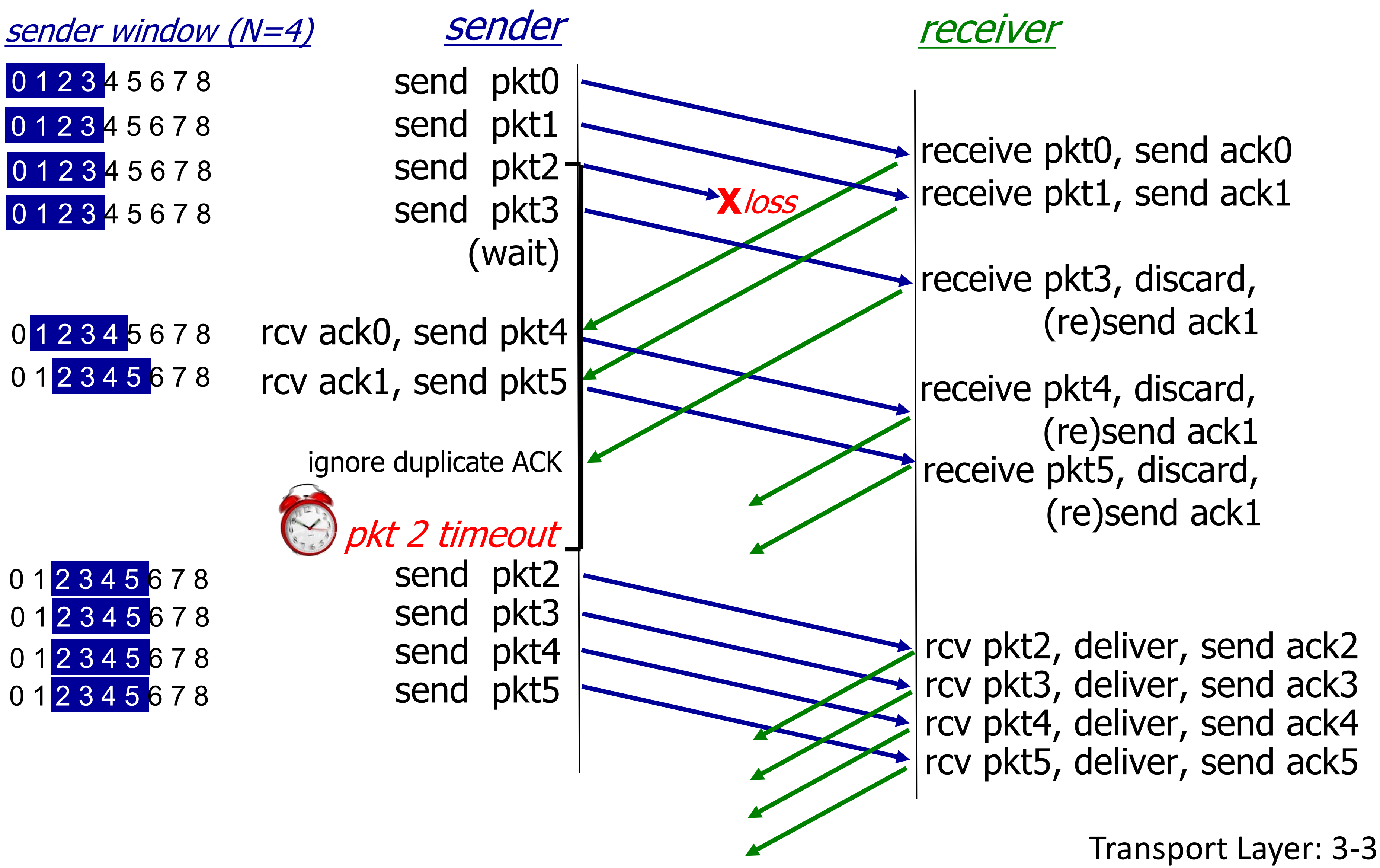
- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - either discard (i.e. don't buffer) or buffer: depends on implementation!
  - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:





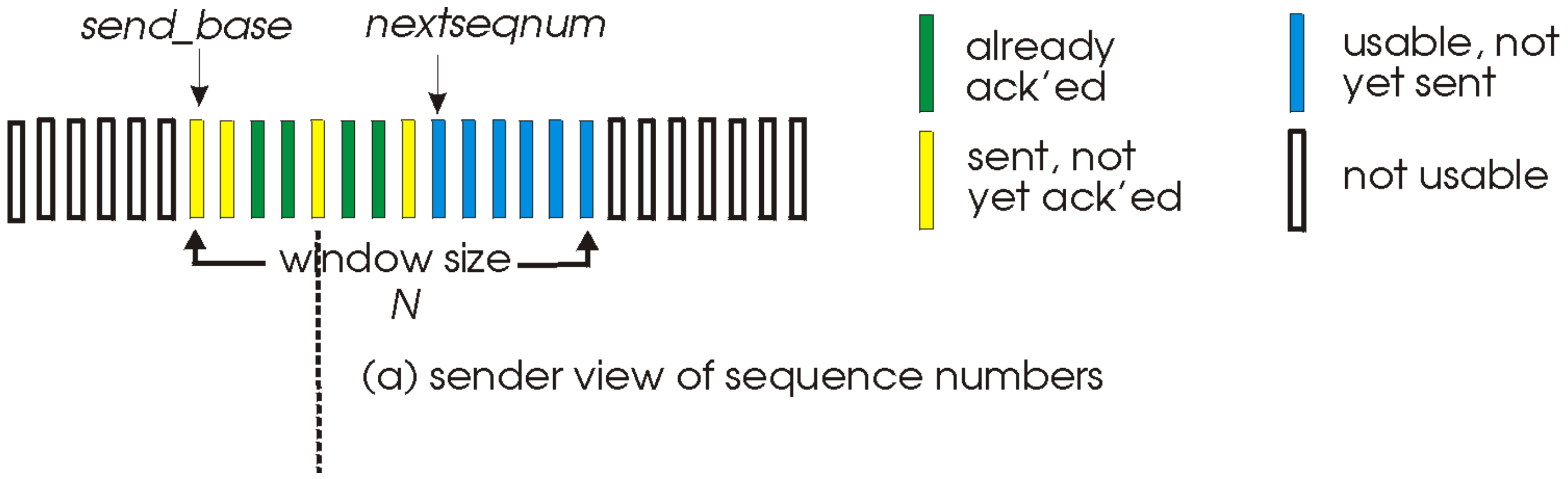
Go-Back-N in action



Selective repeat

- receiver *individually* acknowledges all correctly received packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually unACKed packets
  - sender maintains timer for *each* unACKed pkt
- sender window
  - *N* consecutive seq #s
  - limits seq #s of sent, unACKed packets

Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

Selective repeat: sender and receiver

sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend packet *n*, restart timer

ACK(*n*) in [sendbase,sendbase+N]:

- mark packet *n* as received
- if n smallest unACKed packet, advance window base to next unACKed seq #

receiver

packet *n* in [rcvbase, rcvbase+N]

- send ACK(*n*)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

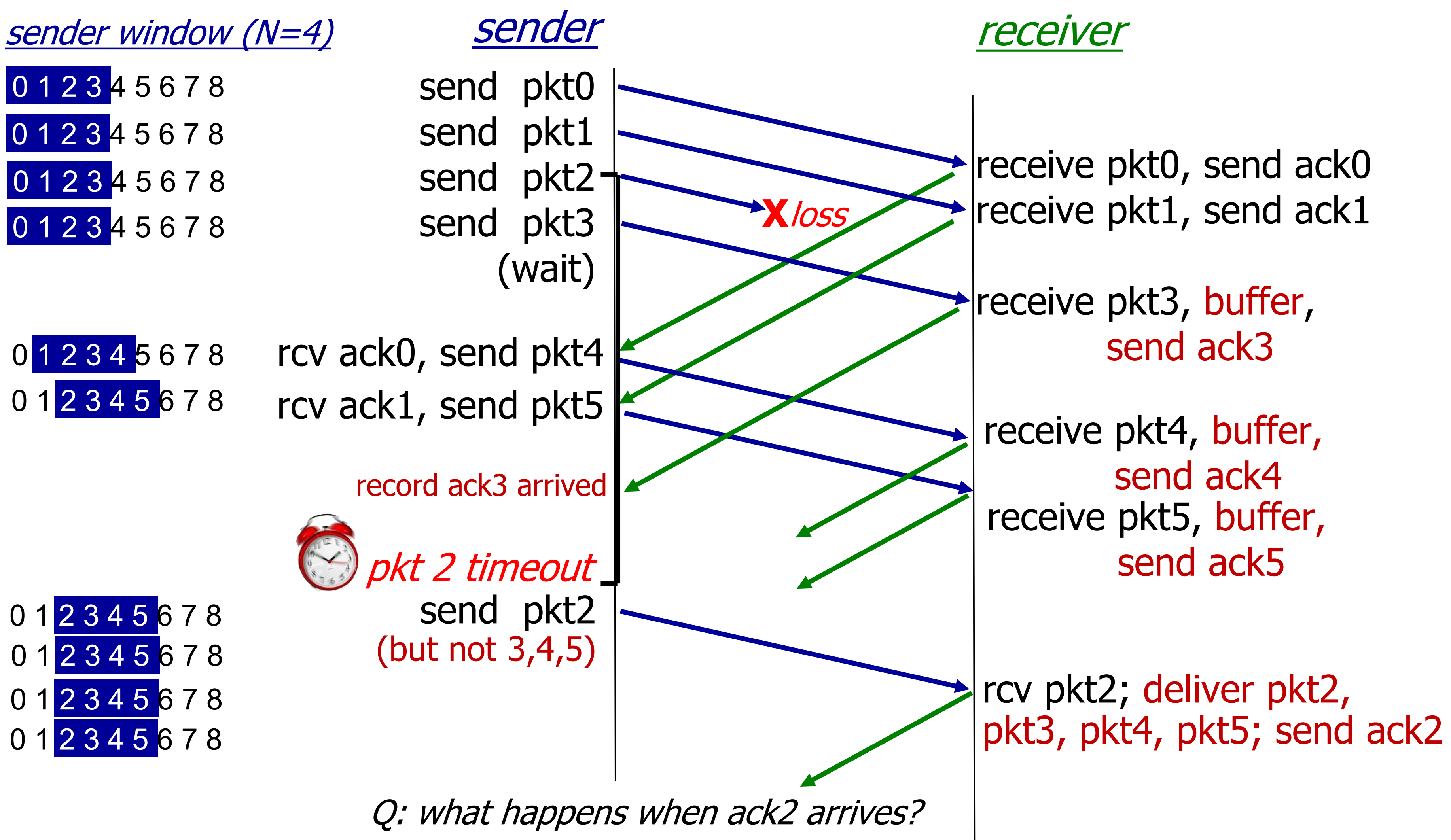
packet *n* in [rcvbase-N,rcvbase-1]

- ACK(*n*)

otherwise:

- ignore

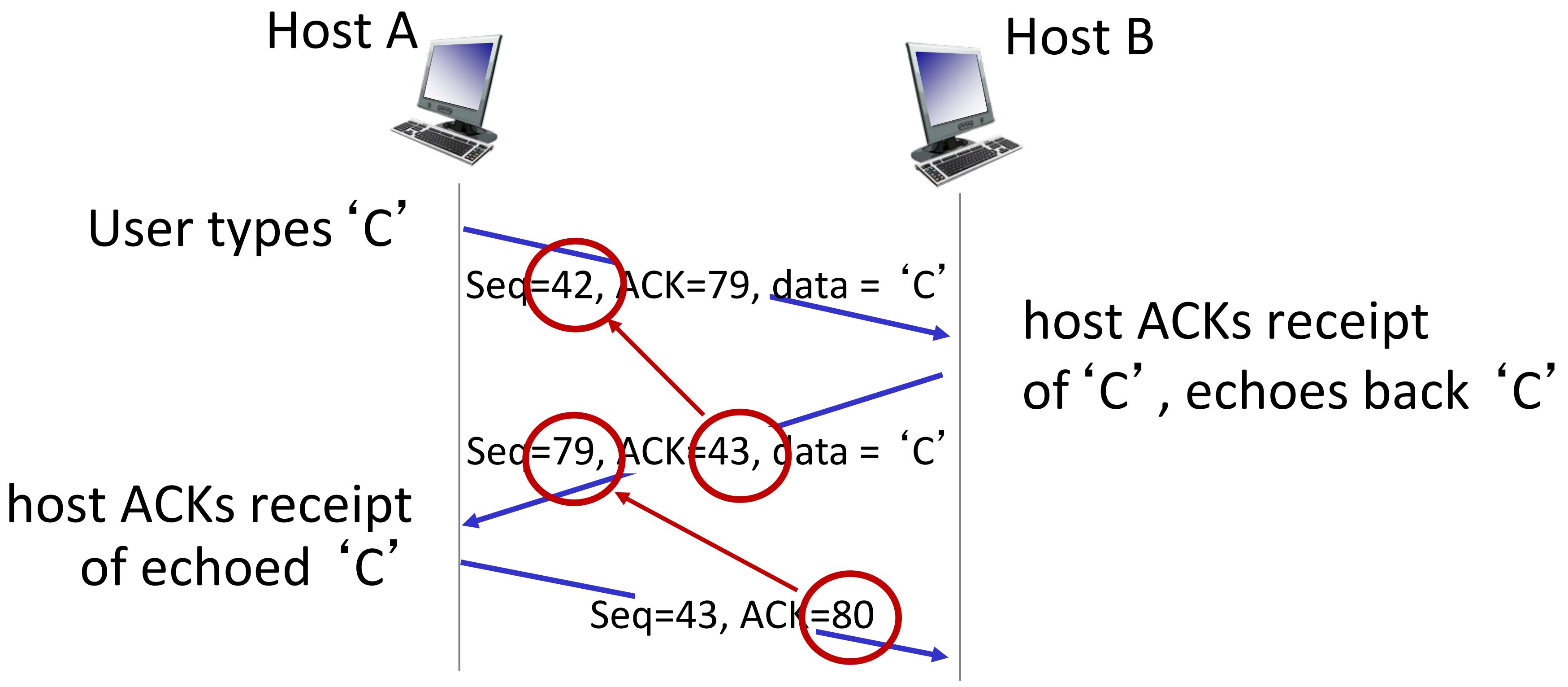
Selective Repeat in action



TCP: overview RFCs: 793,1122, 2018, 5681, 7323

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no “message boundaries"
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- cumulative ACKs
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP sequence numbers, ACKs



simple telnet scenario

TCP round trip time, timeout

Q: how to set TCP timeout value?

- longer than RTT, but RTT varies!
- *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
- `SampleRTT` will vary, → we need "smooth" estimated RTT:
  - average several *recent* measurements, not just current `SampleRTT`

TCP round trip time, timeout

- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** → large safety margin

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

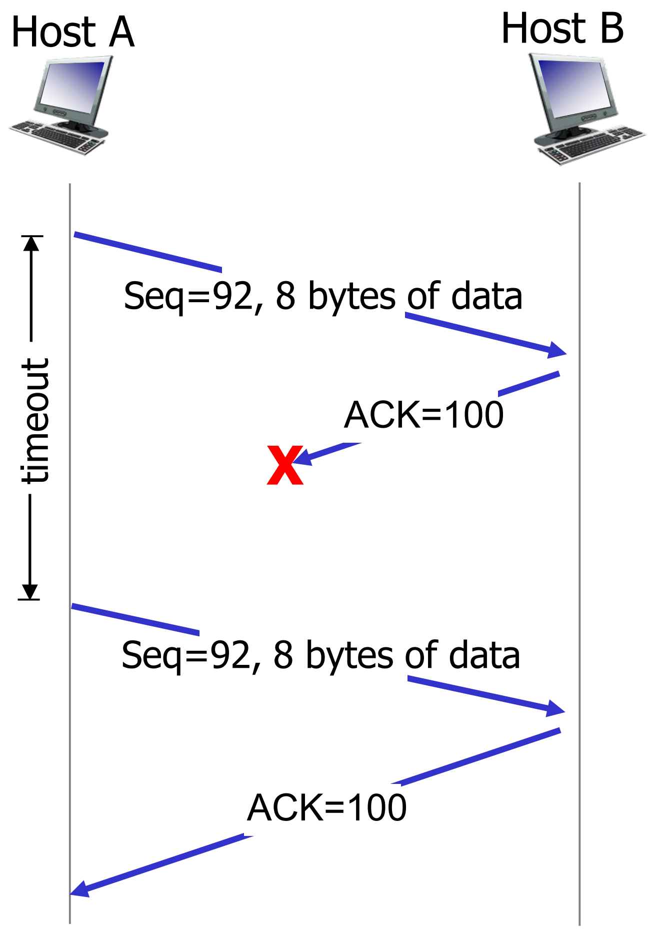
"safety margin"

- **`DevRTT`**: EWMA of |**`SampleRTT - EstimatedRTT`**|:

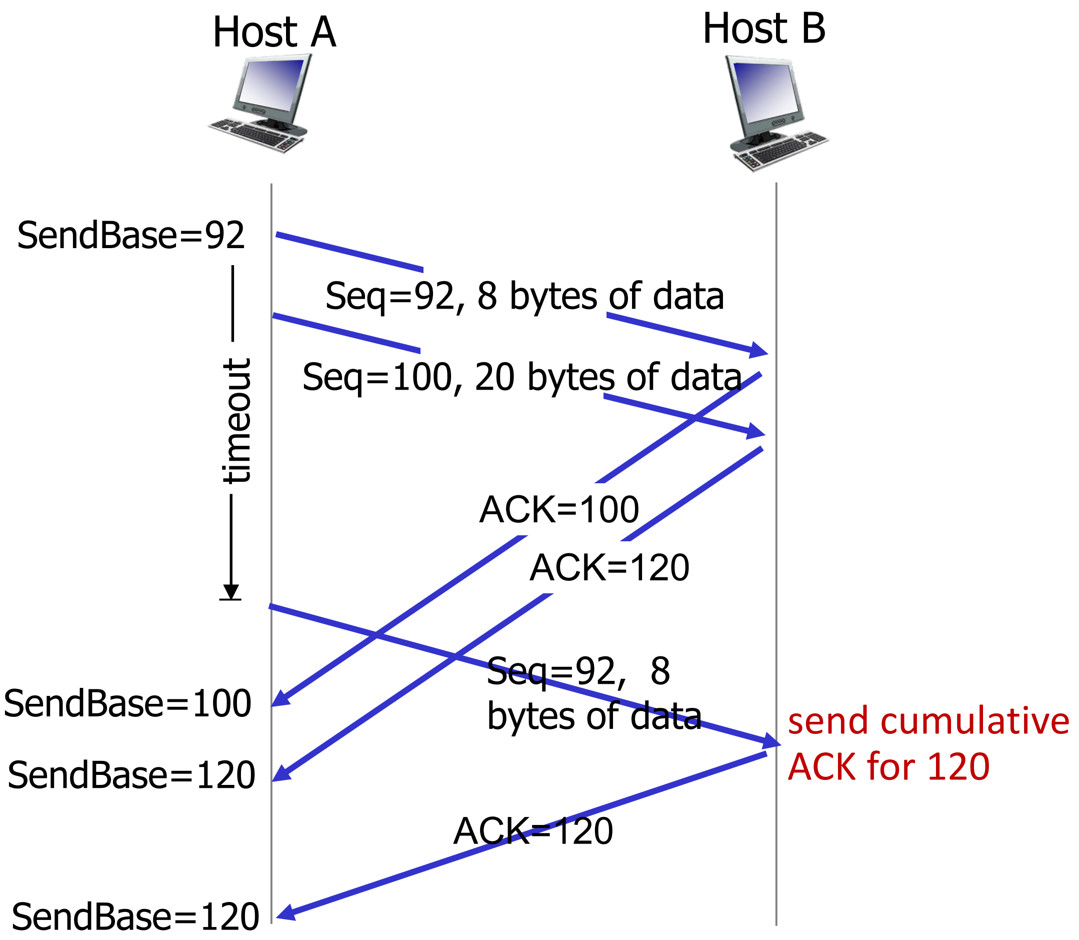
```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
```

(typically, β = 0.25)

TCP: retransmission scenarios



lost ACK scenario



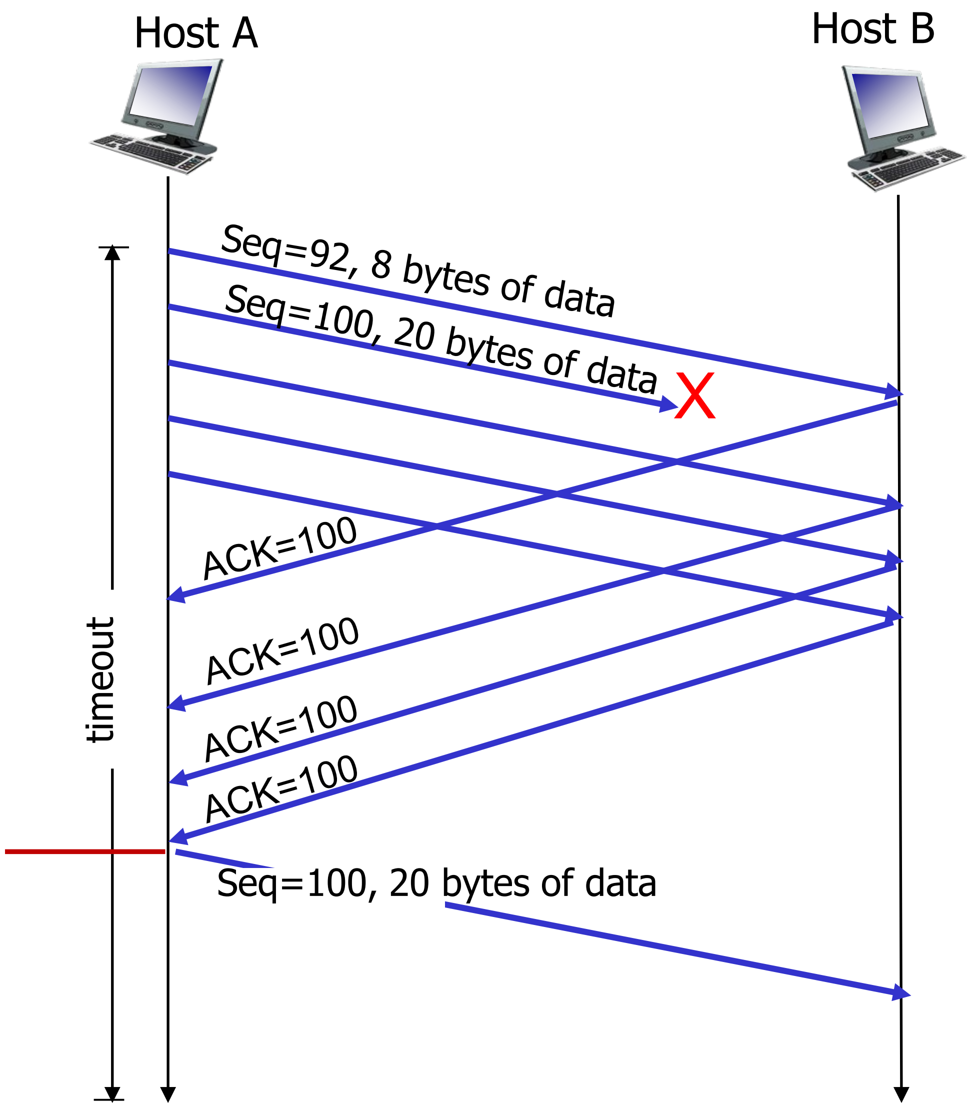
premature timeout

TCP fast retransmit

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

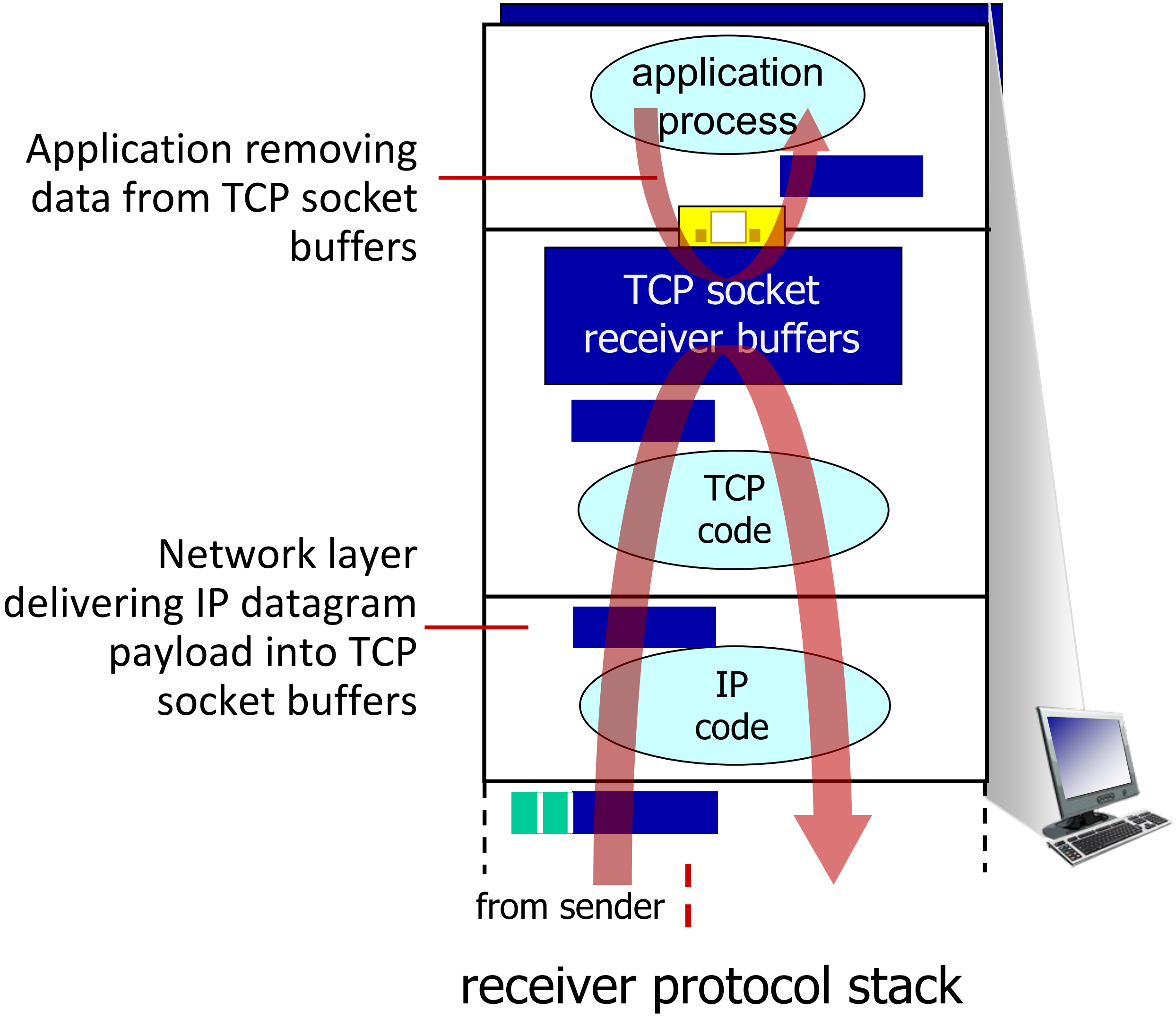
- likely that unACKed segment lost, so don't wait for timeout



Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!

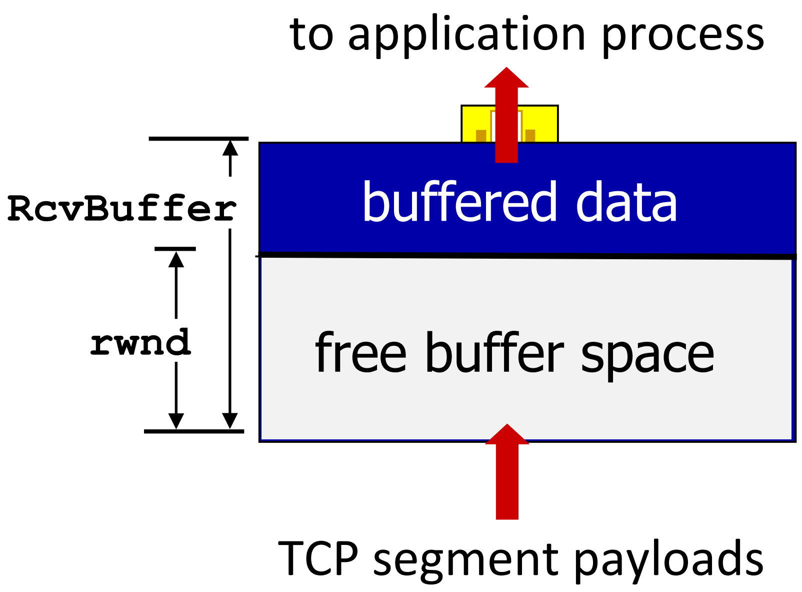
TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



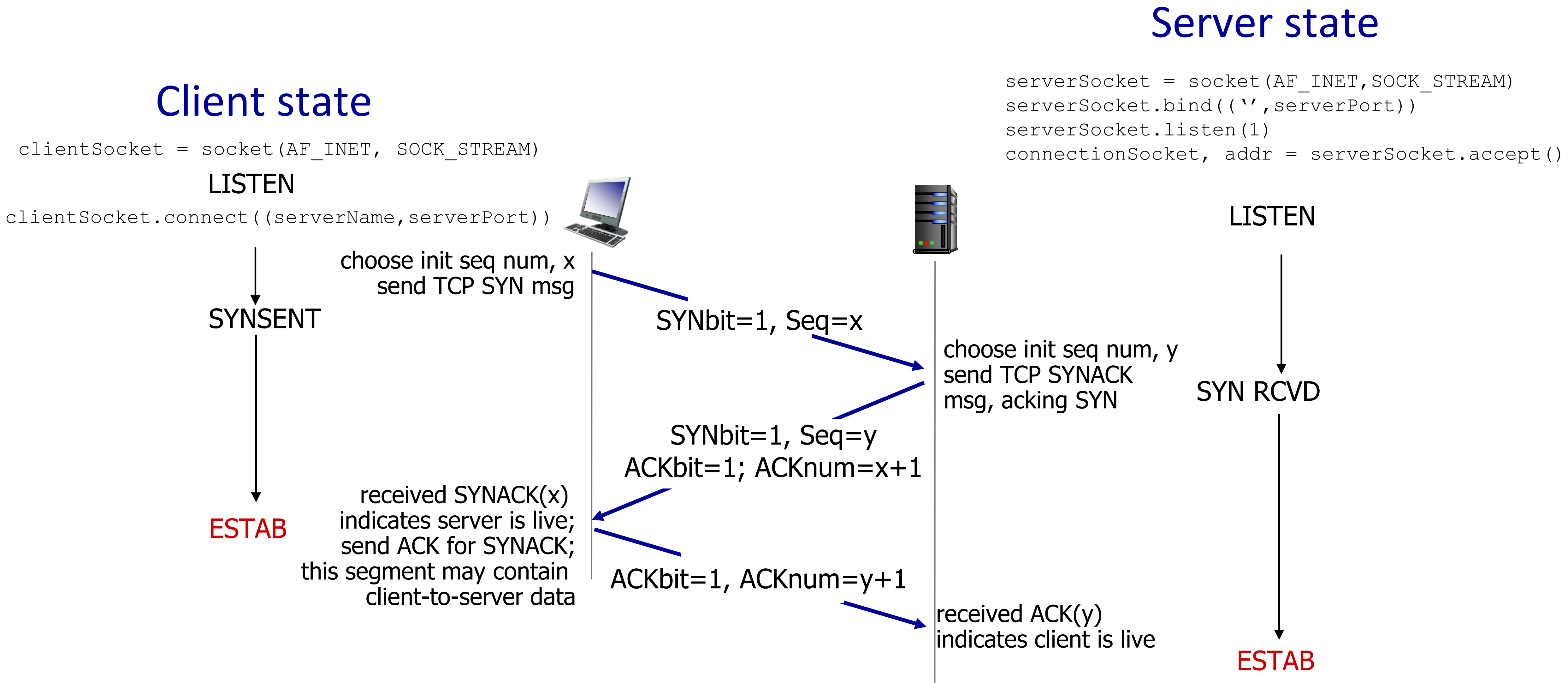
TCP flow control

- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow



TCP receiver-side buffering

TCP 3-way handshake



Principles of congestion control

Congestion:

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

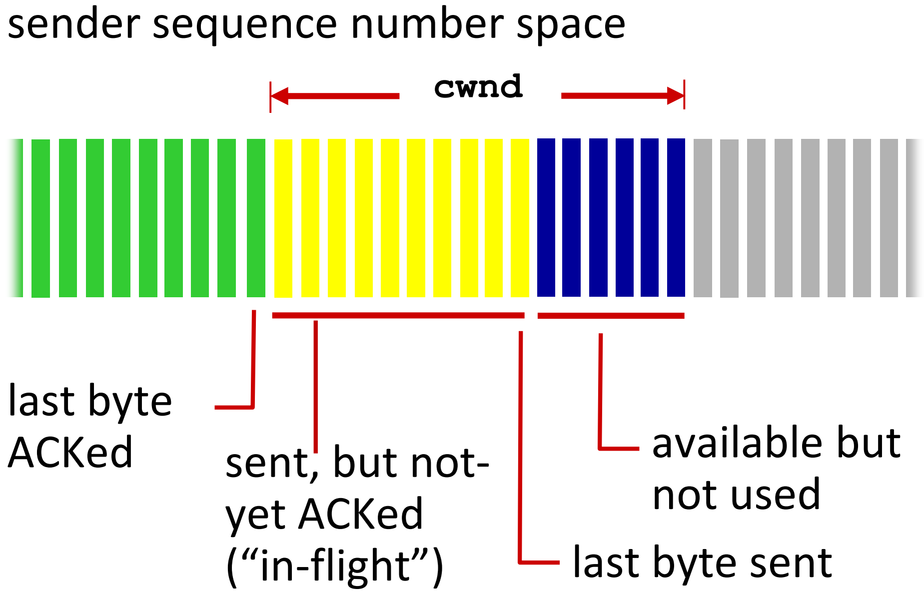
congestion control: too many senders, sending too fast

flow control: one sender too fast for one receiver

Flow Control vs. Congestion Control

- Flow control
  - Keeping *one fast sender* from overwhelming *a slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*
    - E.g., persuade hosts to stop sending, or slow down
    - Typically has notions of fairness (i.e., sharing the pain)
- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - TCP window: *min{congestion window, receiver window}*

TCP congestion control: details



TCP sending behavior:

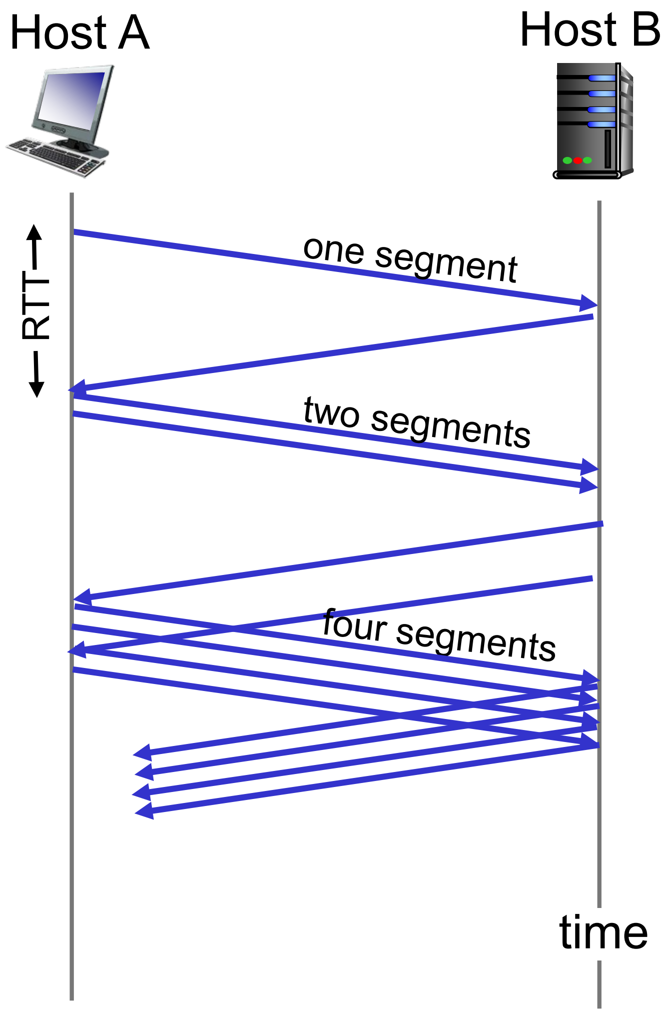
- *roughly:* send `cwnd` bytes, wait RTT for ACKS, then send more bytes

$$\text{TCP rate} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- TCP sender limits transmission: **`LastByteSent - LastByteAcked ≤ cwnd`**
- `cwnd` is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

TCP slow start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



TCP: detecting, reacting to loss

- loss indicated by timeout:
  - **`cwnd`** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly

- loss indicated by 3 duplicate ACKs: **TCP RENO**
  - dup ACKs indicate network capable of delivering *some* segments
  - **`cwnd`** is cut in half window then grows linearly

- **TCP Tahoe** always sets **`cwnd`** to 1 (timeout or 3 duplicate acks)

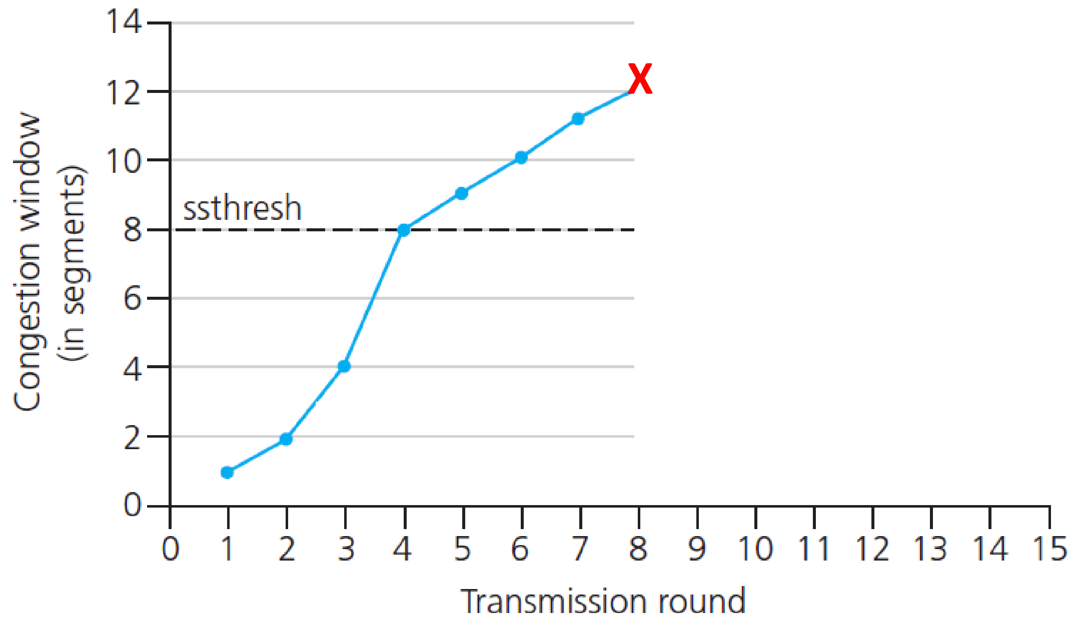
TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.

Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event



Network Layer (Data Plane + Control Plane)

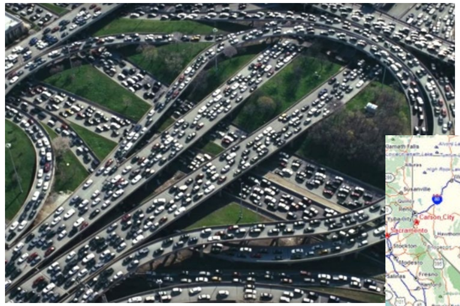
Two key network-layer functions

network-layer functions:

- *forwarding:* move packets from a router's input link to one of its output link
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

analogy: taking a trip

- *forwarding:* process of getting through single interchange
- *routing:* process of planning trip from source to destination



forwarding

routing

Destination-based forwarding

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use the *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

examples:

11001000 00010111 00010110 10100001 which interface?

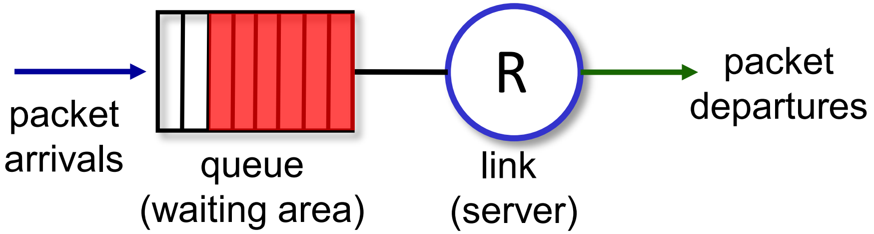
11001000 00010111 00011000 10101010 which interface?

Packet Scheduling: FCFS

packet scheduling: deciding which packet to send next on link

- first come, first served
- priority
- round robin
- weighted fair queueing

Abstraction: queue

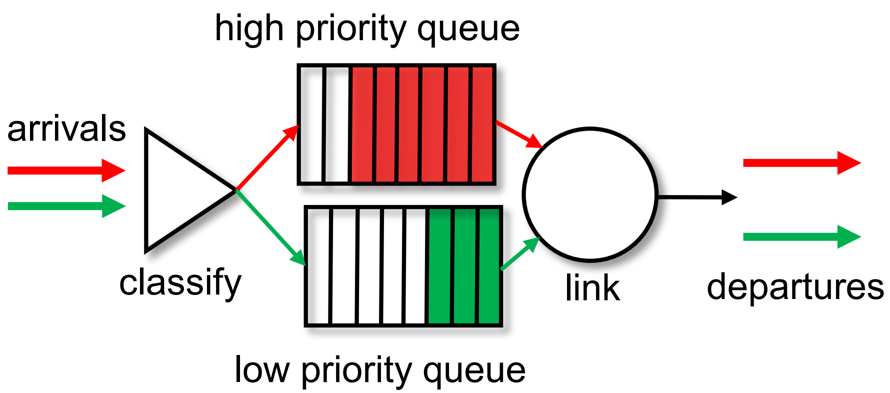


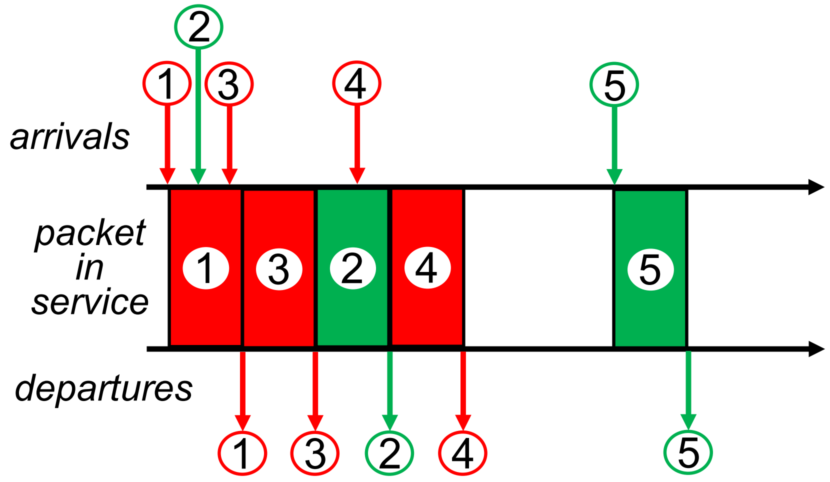
FCFS: packets transmitted in order of arrival to output port

- also known as: First-in-first-out (FIFO)
- Many real world examples

Scheduling policies: priority

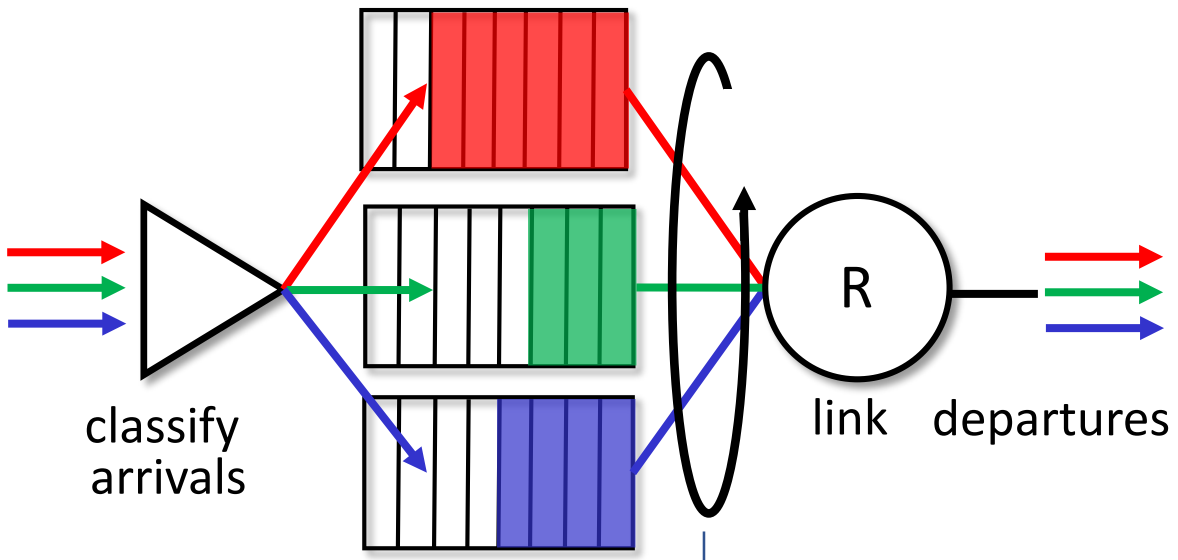
- arriving traffic classified, queued by class
  - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
  - FCFS within priority class





Scheduling policies: round robin (RR)

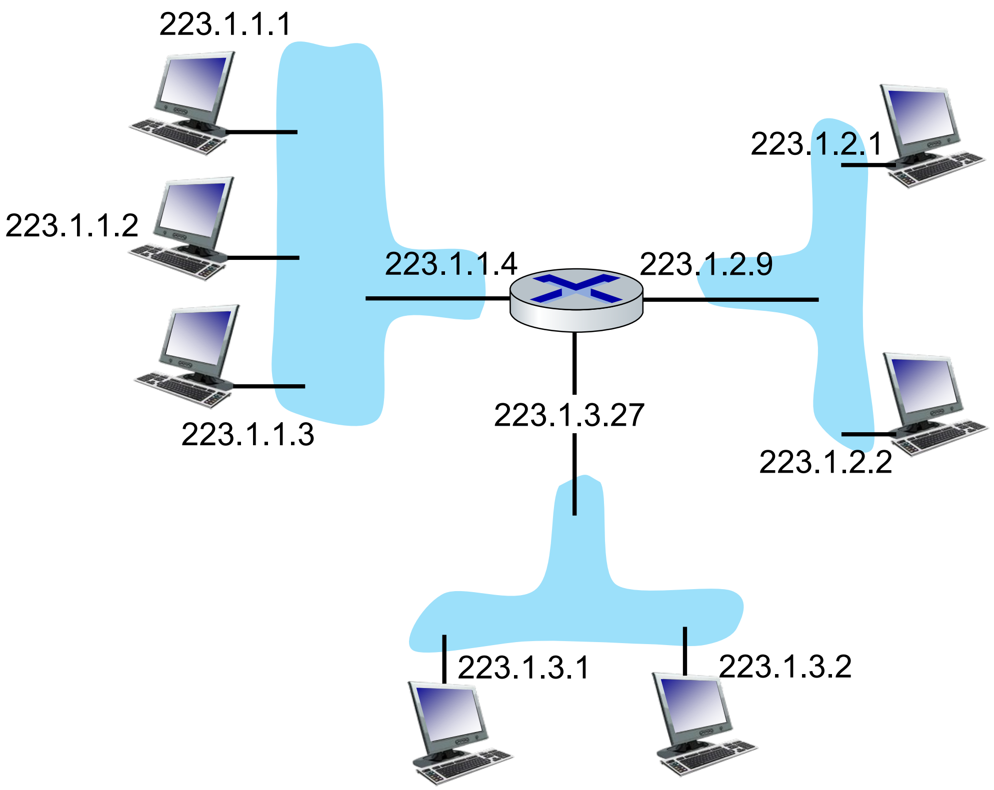
- arriving traffic classified, queued by class
  - any header fields can be used for classification
- Process class queues in cyclic fashion
- sending one (complete) packet from each class (if available) in turn



IP addressing: introduction

- IP address: 32-bit identifier associated with each host/router interface
- interface: connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

223=2^0+2^1+2^2+2^3+2^4+2^6+2^7

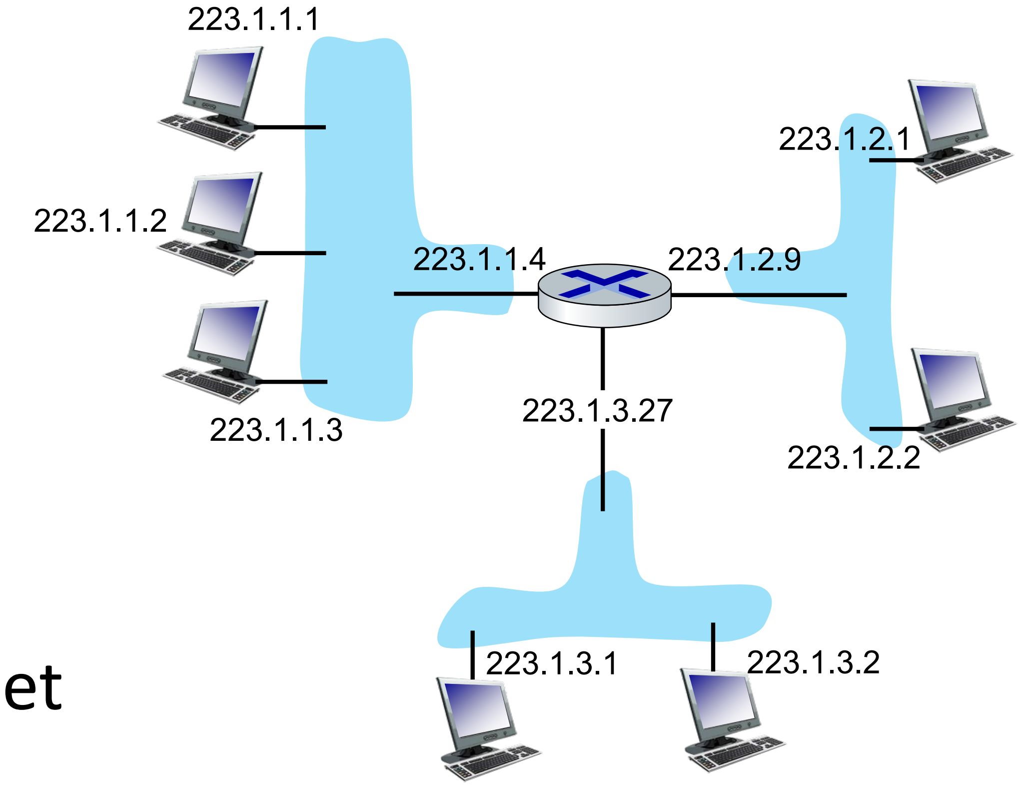


dotted-decimal IP address notation:

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

Subnets

- *What's a subnet ?*
  - device interfaces that can physically reach each other without passing through an intervening router
- IP addresses have structure:
  - subnet part: devices in same subnet have common high order bits
  - host part: remaining low order bits



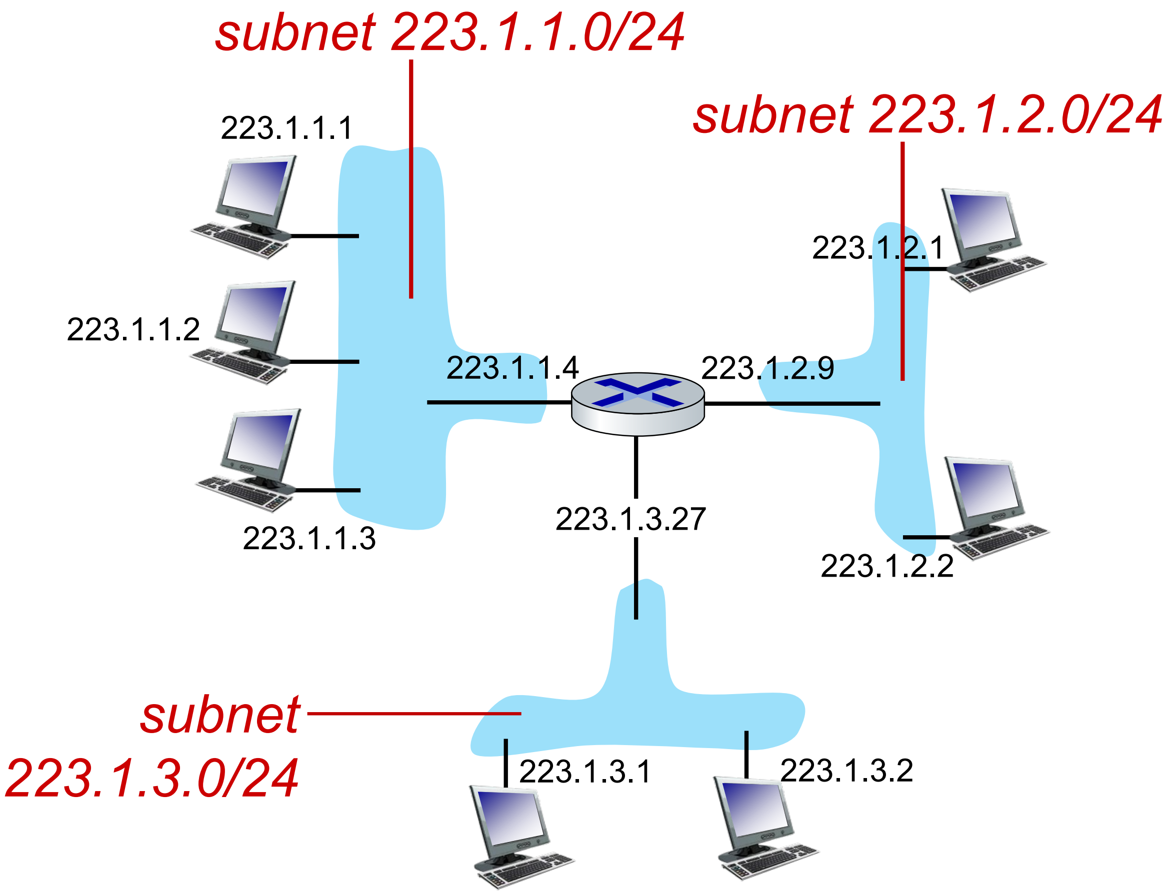
network consisting of 3 subnets

Subnets

Recipe for defining subnets:

- detach each interface from its host or router, creating "islands" of isolated networks
- each isolated network is called a *subnet*
- Subnet mask/24:

255	255	255	0
11111111	111111111	11111111	00000000



subnet mask: /24
(high-order 24 bits: subnet part of IP address)

IP addressing: CIDR

CIDR: Classless InterDomain Routing (pronounced “cider”)

- subnet portion of address of *arbitrary* length
- address format: a.b.c.d/x, where x is # of bits in subnet portion of address

subnet part | host part

11001000 00010111 00010000 00000000

200.23.16.0/23

DHCP: Dynamic Host Configuration Protocol

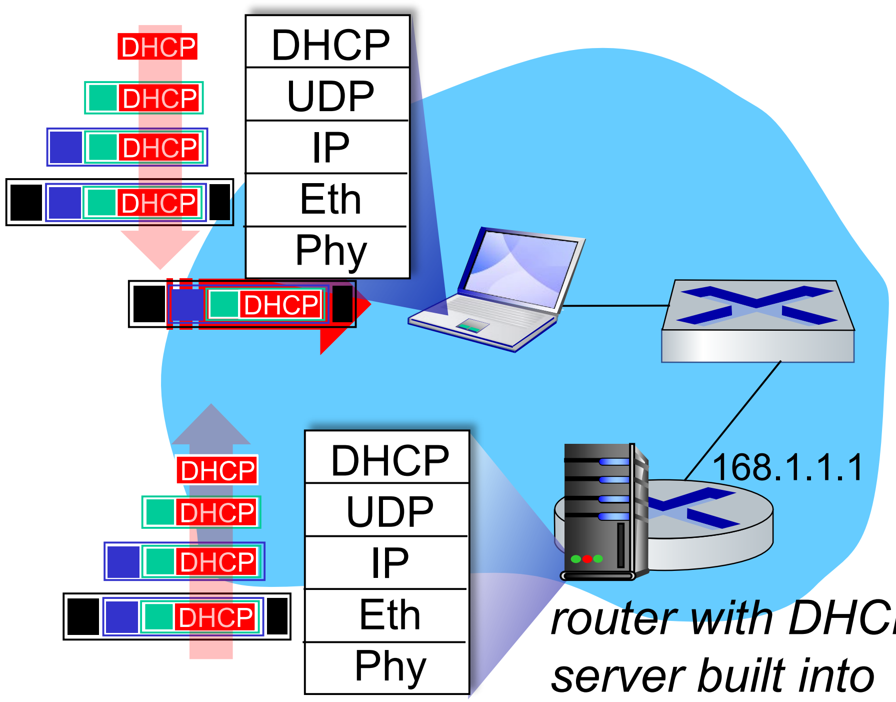
goal: host *dynamically* obtains IP address from network server when it "joins" network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

DHCP overview:

- host broadcasts DHCP discover msg [optional]
- DHCP server responds with DHCP offer msg [optional]
- host requests IP address: DHCP request msg
- DHCP server sends address: DHCP ack msg

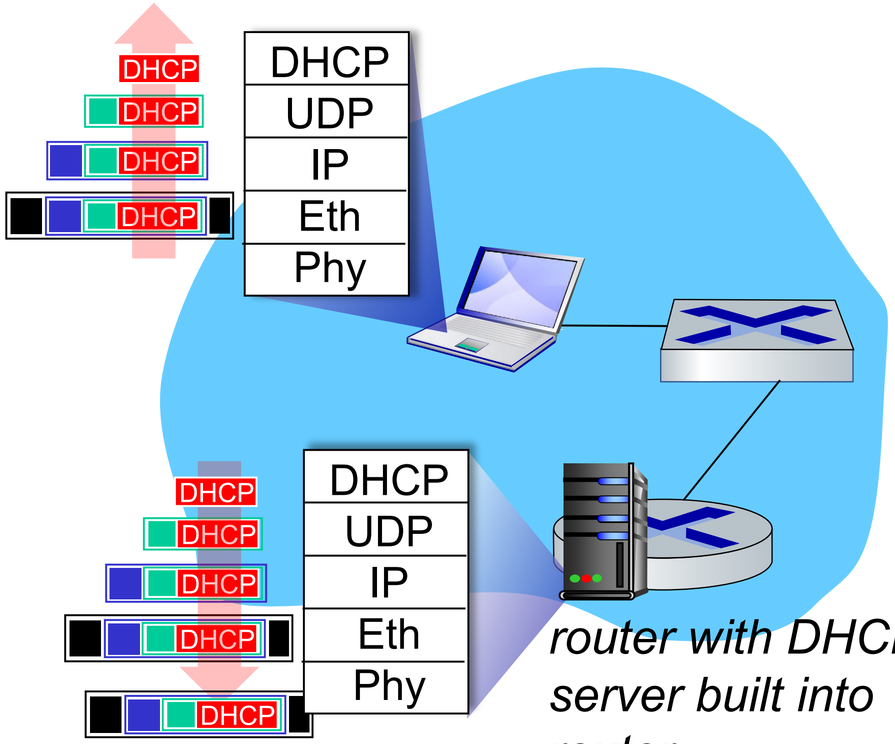
DHCP: example



router with DHCP server built into router

- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP

DHCP: example



router with DHCP server built into router

- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulated DHCP server reply forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

IP addresses: how to get one?

Q: how does *network* get subnet part of IP address?

A: gets allocated portion of its provider ISP's address space

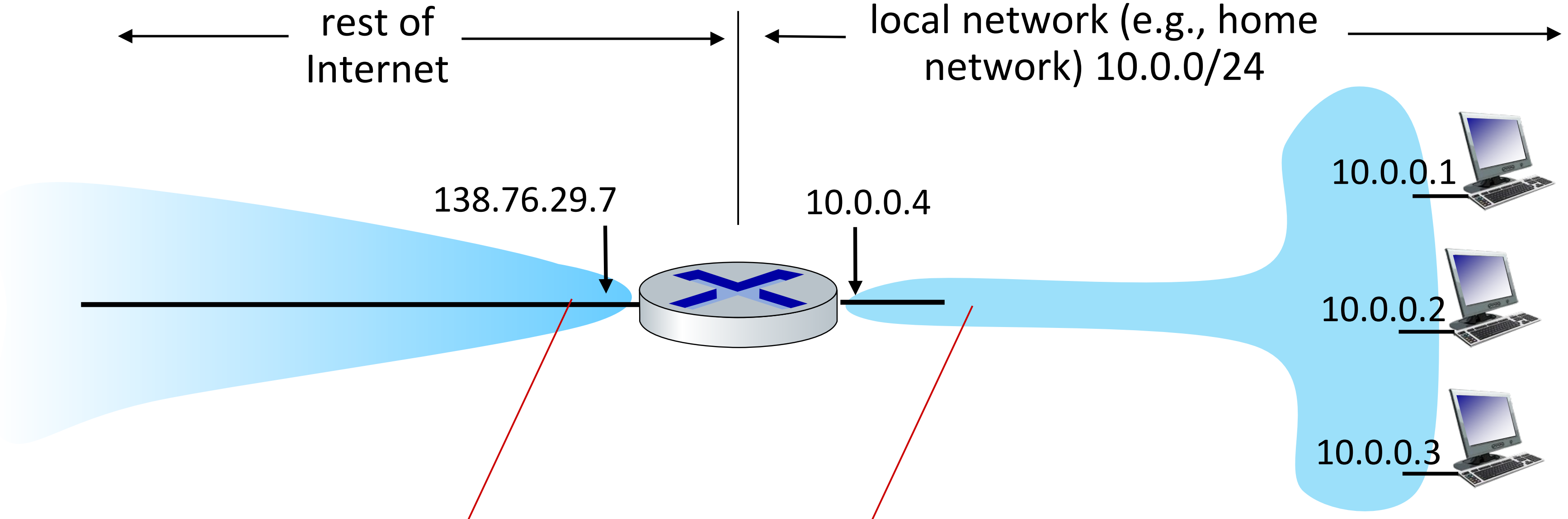
ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20

ISP can then allocate out its address space in 8 blocks:

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

NAT: network address translation

NAT: all devices in local network share just one IPv4 address as far as outside world is concerned



all datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different* source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

NAT: network address translation

- all devices in local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just one IP address needed from provider ISP for *all* devices
  - can change addresses of hosts in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - security: devices inside local net not directly addressable/visible by outside world

NAT: network address translation

implementation: NAT router must (transparently):

- outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - remote clients/servers will respond using (NAT IP address, new port #) as destination address
- remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- incoming datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

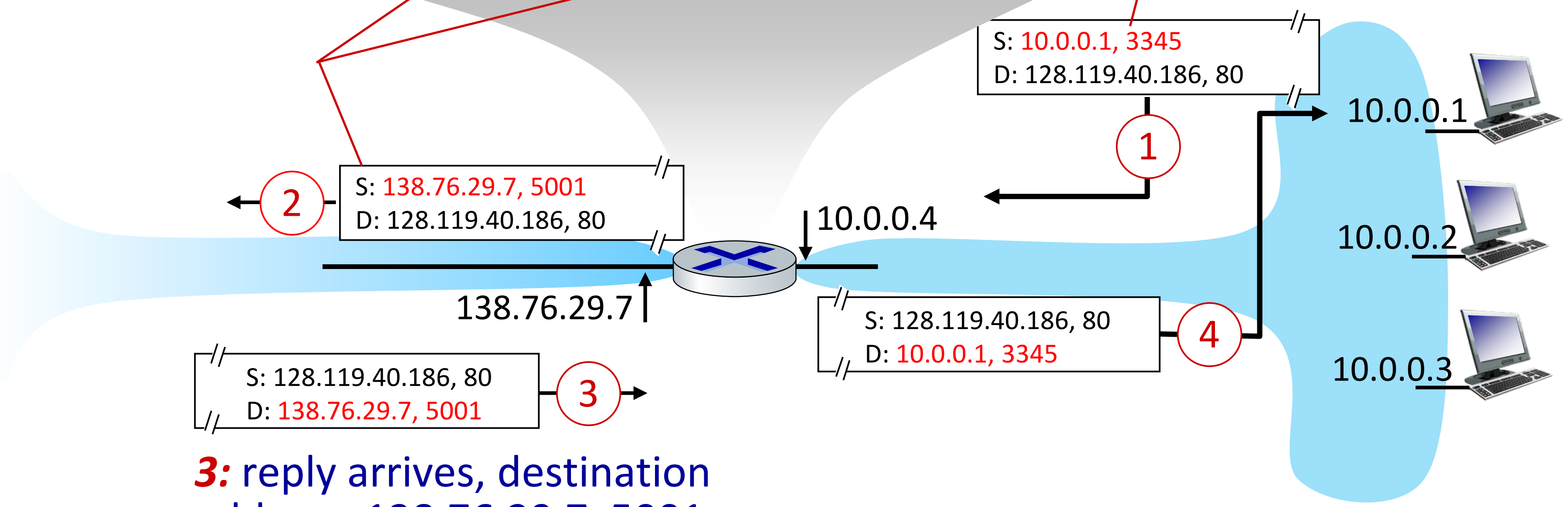
NAT: network address translation

2: NAT router changes datagram source address from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

NAT translation table

WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

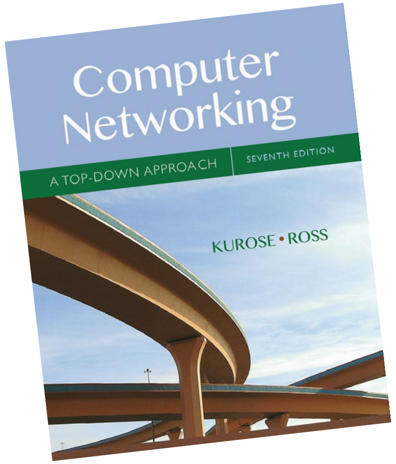
1: host 10.0.0.1 sends datagram to 128.119.40.186, 80



3: reply arrives, destination address: 138.76.29.7, 5001

Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: RIP & OSPF
- routing among ISPs: BGP

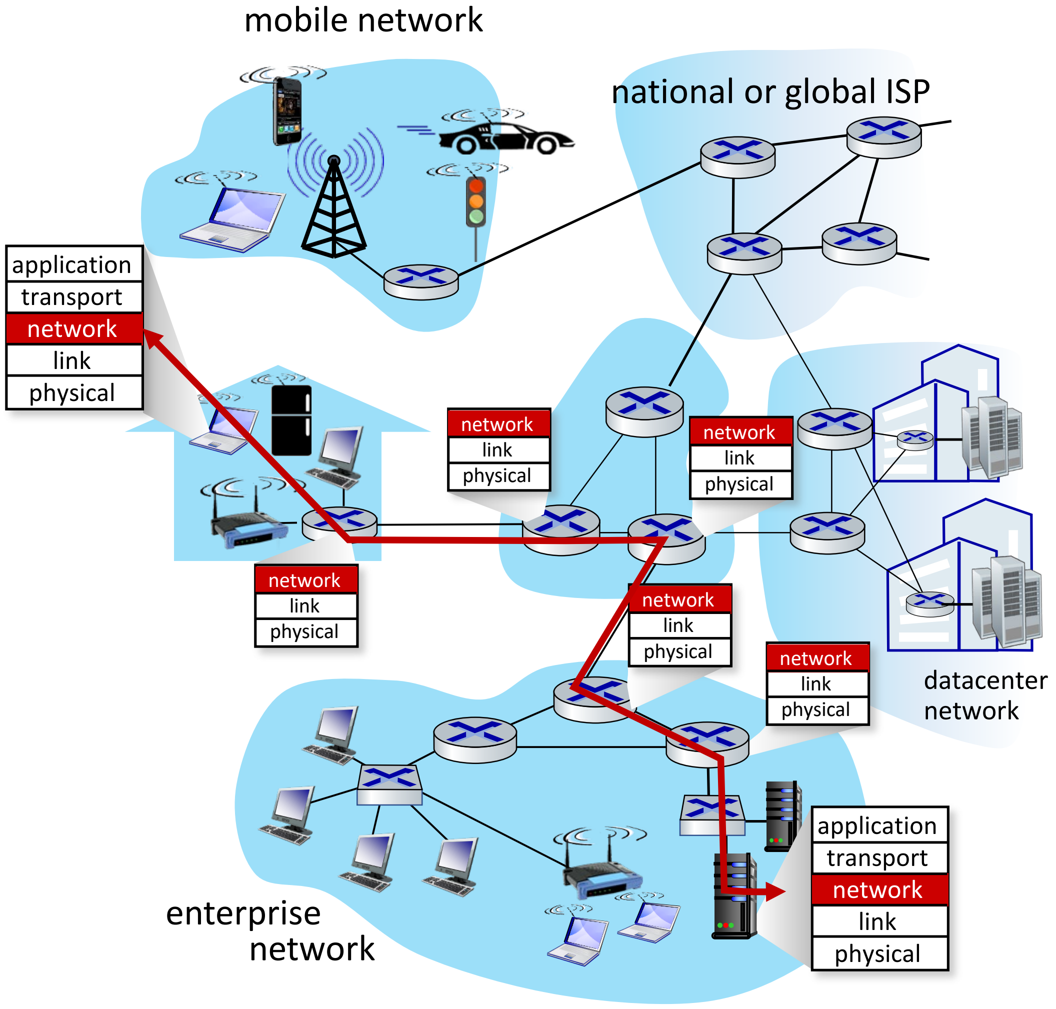


Chapter 5

Routing protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending host to receiving host, through network of routers

- path: sequence of routers packets traverse from given initial source host to final destination host
- "good": least "cost", "fastest", "least congested"
- routing: a "top-10" networking challenge!



Dijkstra's link-state routing algorithm

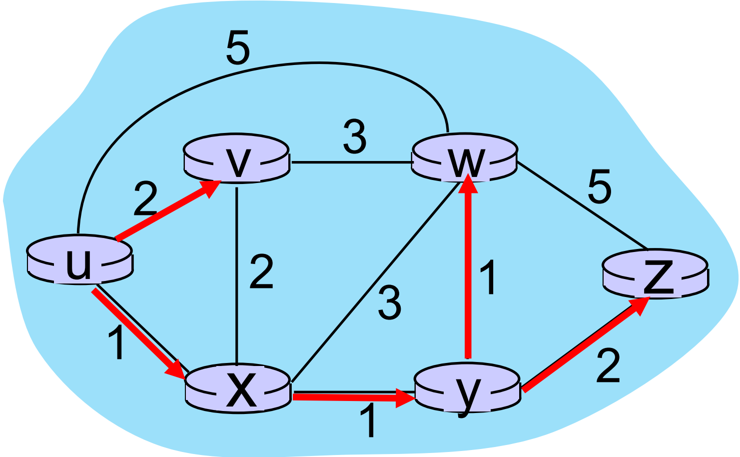
Initialization:
  $N' = \{u\}$ /* compute least cost path from u to all other nodes */
  for all nodes v
    if v adjacent to u /* u initially knows direct-path-cost only to direct neighbors */
      then $D(v) = c_{u,v}$ /* but may not be *minimum* cost! */
    else $D(v) = \infty$

Loop
  find w not in N' such that $D(w)$ is a minimum
  $N'=N'+\{w\}$ --- 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 least-path-cost to v is either old least-cost-path to v or known
  least-cost-path to w plus the link cost from w to v */
until all nodes in N'

Dijkstra's algorithm: an example

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Initialization (step 0): For all a: if *a* adjacent to *u*, then $D(a) = c_{u,a}$



find a vertex (node) *a* not in *N'* such that *D(a)* is a minimum
add *a* to *N'*
update *D(b)* for all *b* adjacent to *a* and not in *N'* :

$$D(b) = \min \{ D(b), D(a) + c_{a,b} \}$$

Distance vector algorithm

Based on *Bellman-Ford* (BF) equation:

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y.
Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

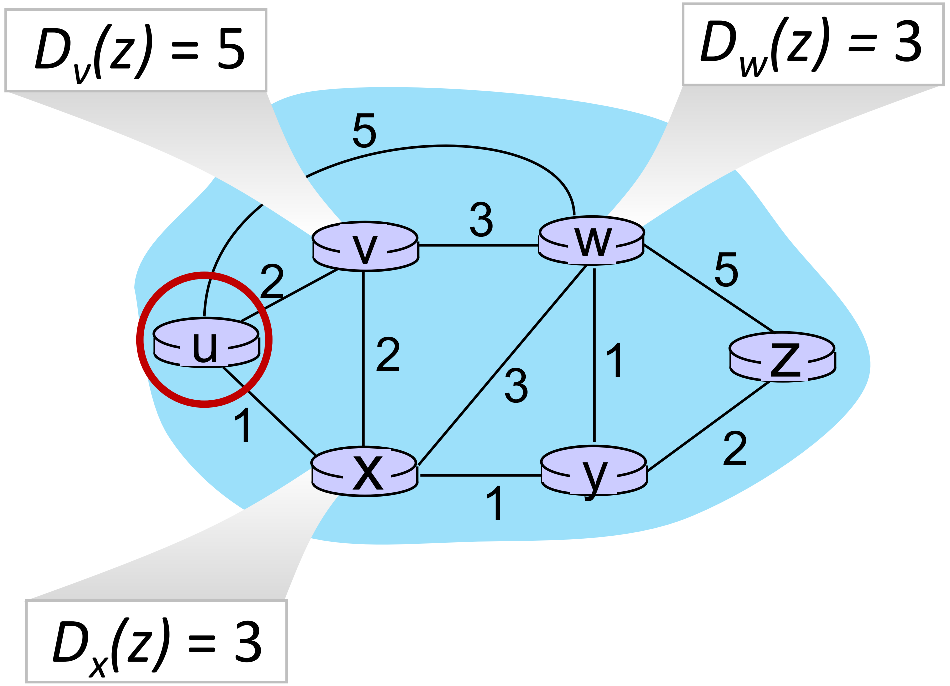
v's estimated least-cost-path cost to y

cost of link from x to v

min taken over all neighbors v of x

Bellman-Ford Example

Suppose that *u*'s neighboring nodes, *x,v,w,* know that for destination *z*:



Bellman-Ford equation says:

$$D_u(z) = \min \{ c_{u,v} + D_v(z), c_{u,x} + D_x(z), c_{u,w} + D_w(z) \}$$
$$= \min \{2 + 5, 1 + 3, 5 + 3\} = 4$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

Internet approach to scalable routing

organize routers into regions known as "autonomous systems" (**AS**) a.k.a. "domains"

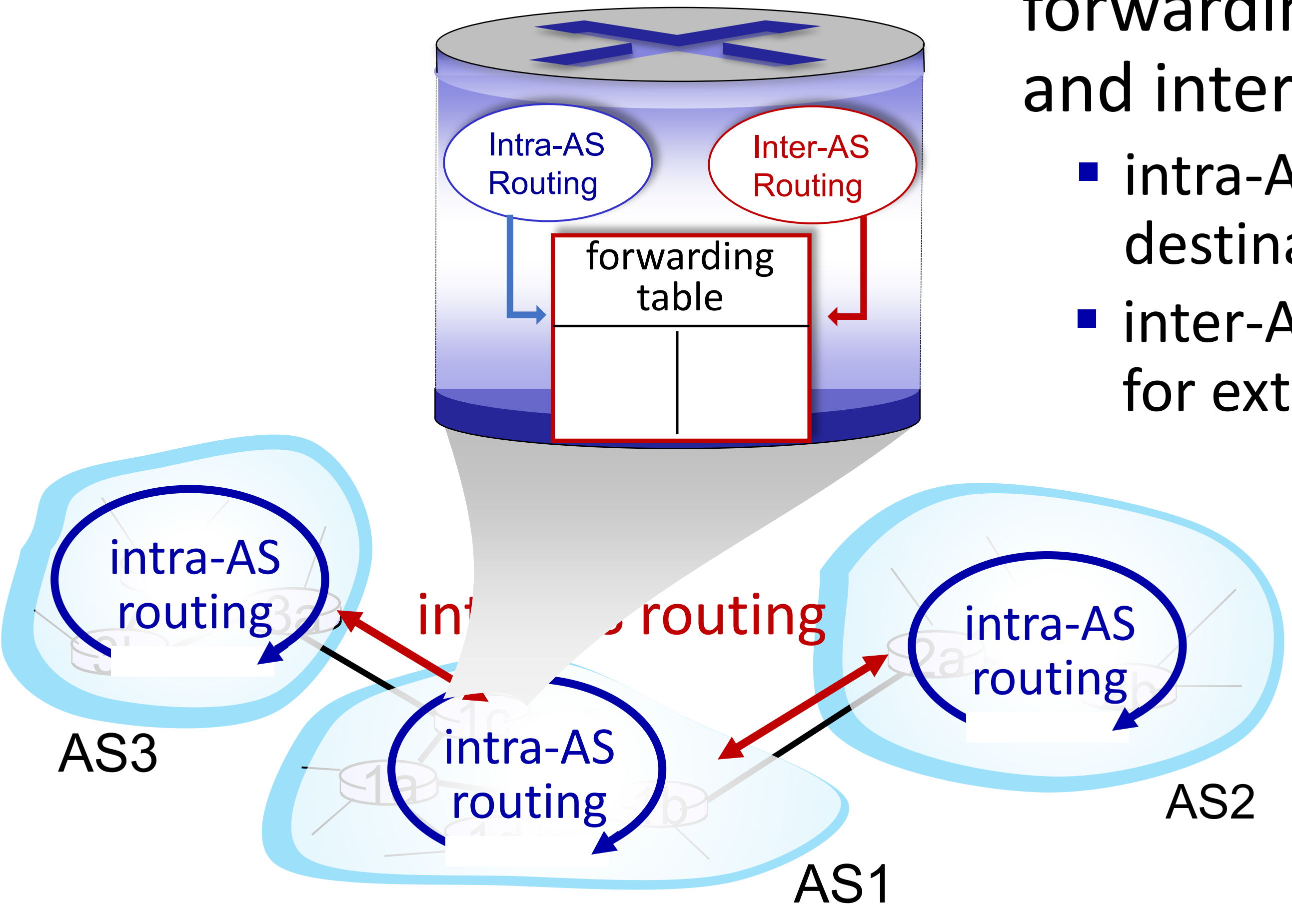
intra-AS (aka "intra-domain"):
routing among *within same AS ("network")*

- all routers in AS must run *same* intra-domain routing protocol
- routers in different ASs can run different intra-domain routing protocols
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other ASs

inter-AS (aka "inter-domain"):
routing *among* ASs

- gateways perform inter-AS routing (as well as intra-AS routing)

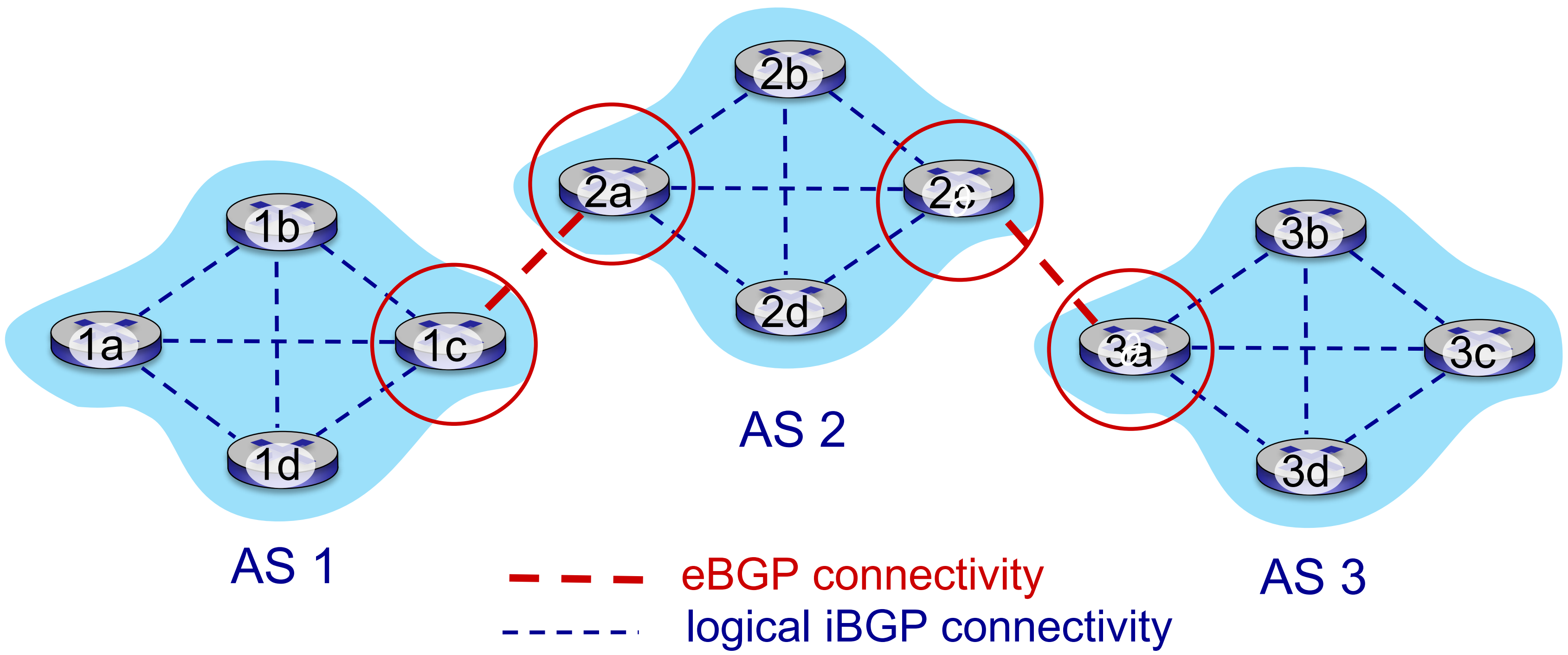
Interconnected ASs

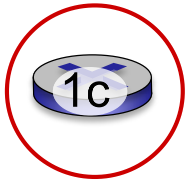


forwarding table configured by intra- and inter-AS routing algorithms

- intra-AS routing determines entries for destinations within an AS
- inter-AS & intra-AS determine entries for external destinations

eBGP, iBGP connections

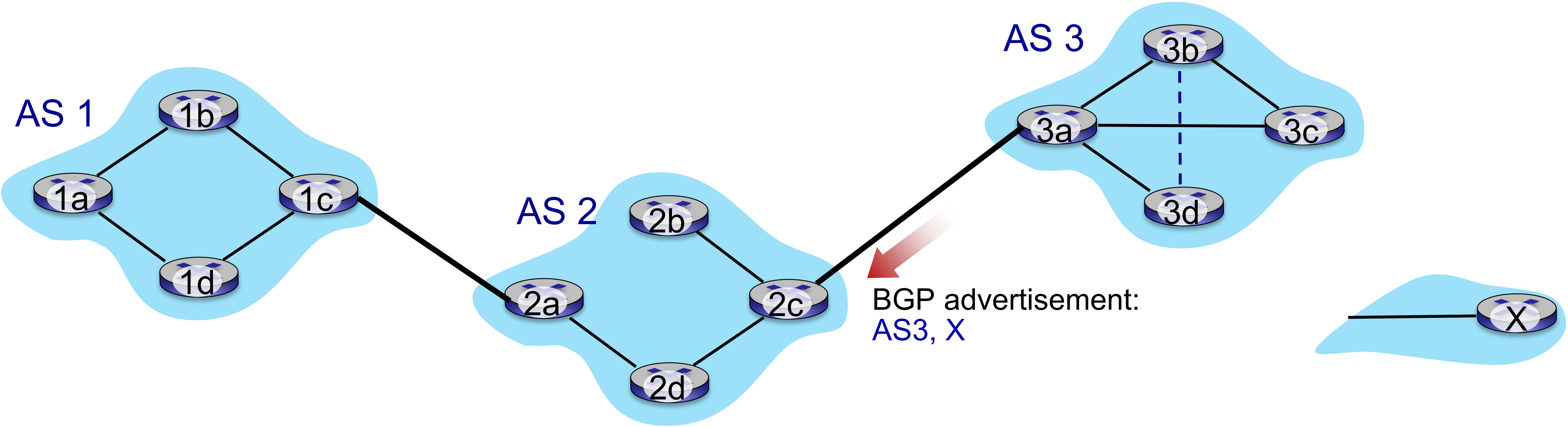




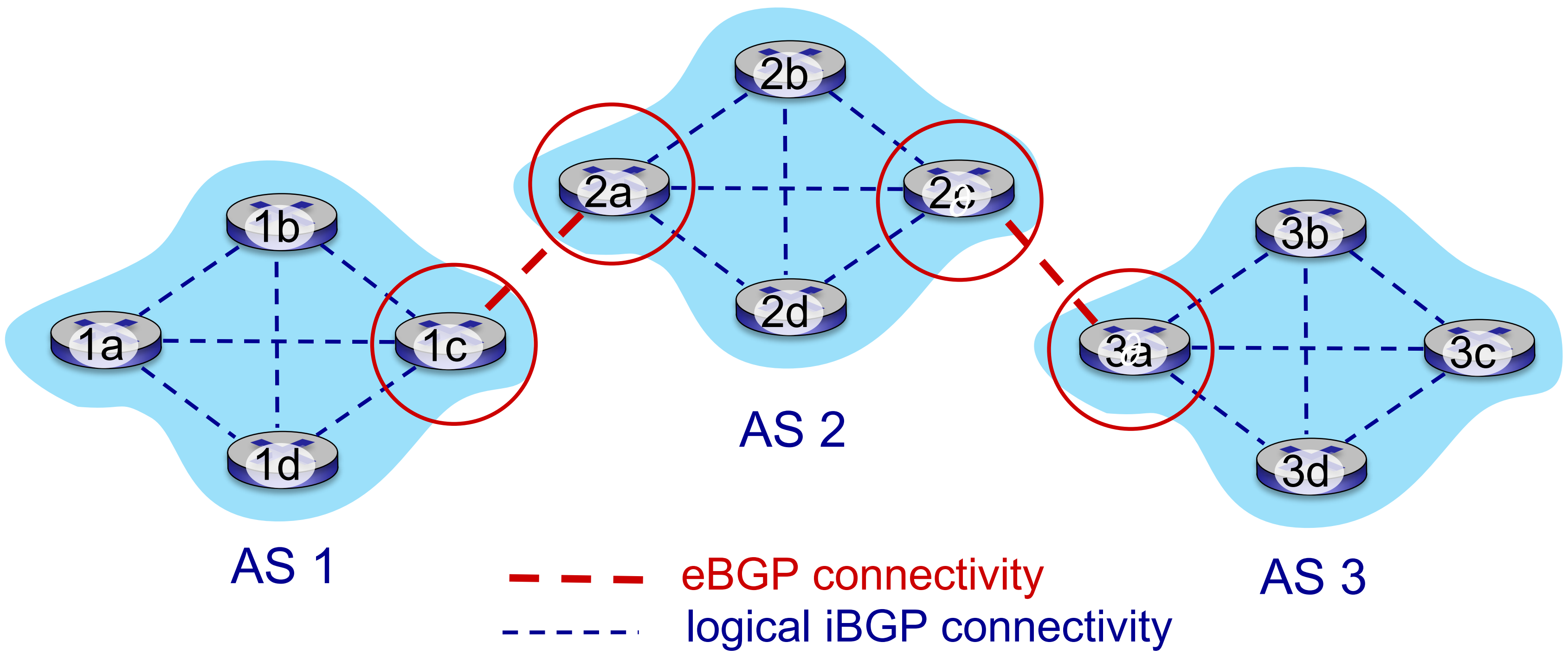
gateway routers run both eBGP and iBGP protocols

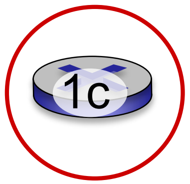
BGP basics

- BGP session: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)
- when AS3 gateway 3a advertises path AS3,X to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



eBGP, iBGP connections

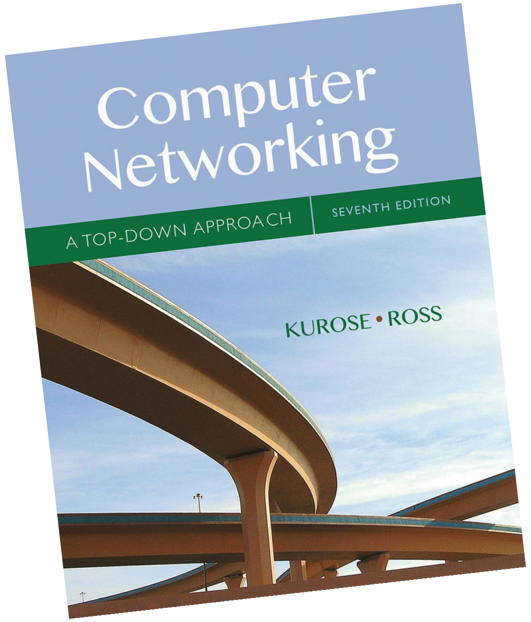




gateway routers run both eBGP and iBGP protocols

Link layer, LANs: roadmap

- Introduction
- Multiple access protocols
- Error detection, correction
- LANs
  - addressing, ARP
  - switches
- Data center networking
- A day in the life of a web request



Chapter 6

Random access protocols

- when node has packet to send
  - transmit at full channel data rate R.
  - no *a priori* coordination among nodes (some exceptions)
- two or more transmitting nodes: “collision”
- random access MAC protocol specifies:
  - how to detect collisions
  - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
  - ALOHA, slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA

Summary of Multiple Access Channel protocols

- channel partitioning, by time, frequency or code
  - Time Division, Frequency Division
- random access (dynamic),
  - ALOHA, SLOTTED-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- taking turns
  - polling from central site, token passing
  - Bluetooth

Slotted ALOHA

assumptions:

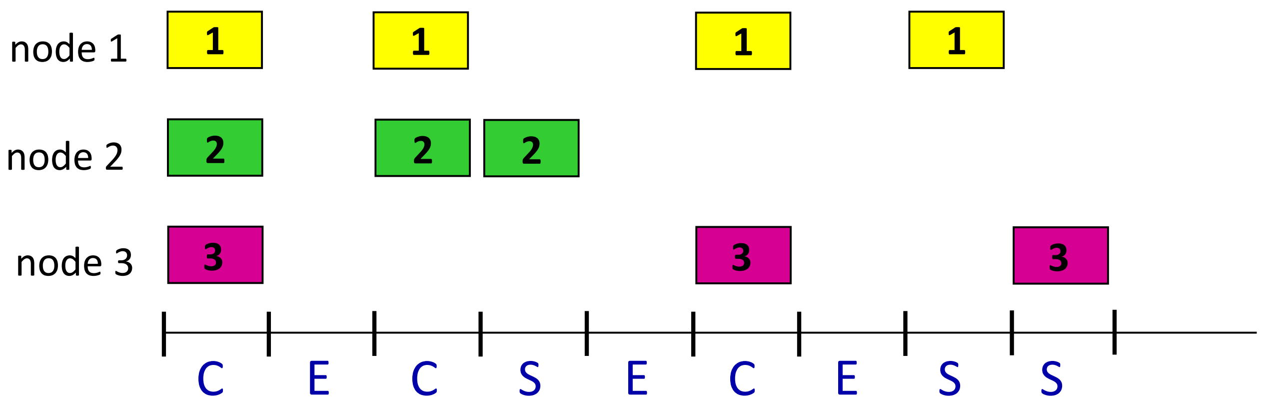
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

- when node obtains fresh frame, transmits in next slot
  - *if no collision:* node can send new frame in next slot
  - *if collision:* node retransmits frame in each subsequent slot with probability *p* until success

must use randomization!

Slotted ALOHA



C: collision
S: success
E: empty

Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

Slotted ALOHA: efficiency

Efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)

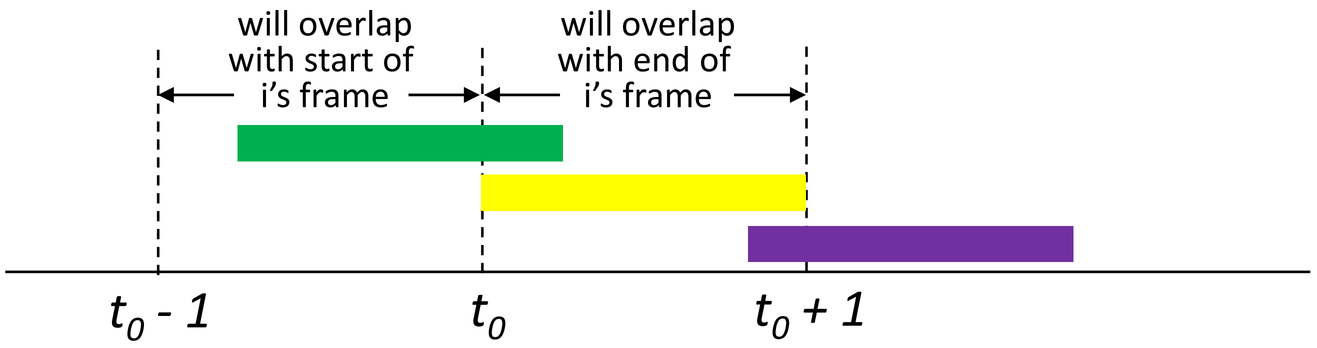
- *Assumptions:* N nodes with many frames to send, each node transmits in a slot with probability p
  - prob that a given node has a success in a slot $= p(1-p)^{N-1}$
  - prob that *any* node has a success $= Np(1-p)^{N-1}$
  - **maximum efficiency**: find a p that maximizes $Np(1-p)^{N-1}$
  - $\rightarrow p = 1/N$
  - for many nodes, take limit of $Np(1-p)^{N-1}$ as N goes to infinity, gives:

  max efficiency = 1/e = .37

- *at best:* channel used for useful transmissions 37% of time!

Pure ALOHA

- unslotted Aloha: simpler, no synchronization
  - when frame first arrives: transmit immediately
- collision probability increases with no synchronization:
  - frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$



Pure ALOHA efficiency

P(success by given node) = P(node transmits) ·

P(no other node transmits in $[t_0\text{-}1,t_0]$ ·

P(no other node transmits in $[t_0,t_{0+1}]$

$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$

$= p \cdot (1-p)^{2(N-1)}$

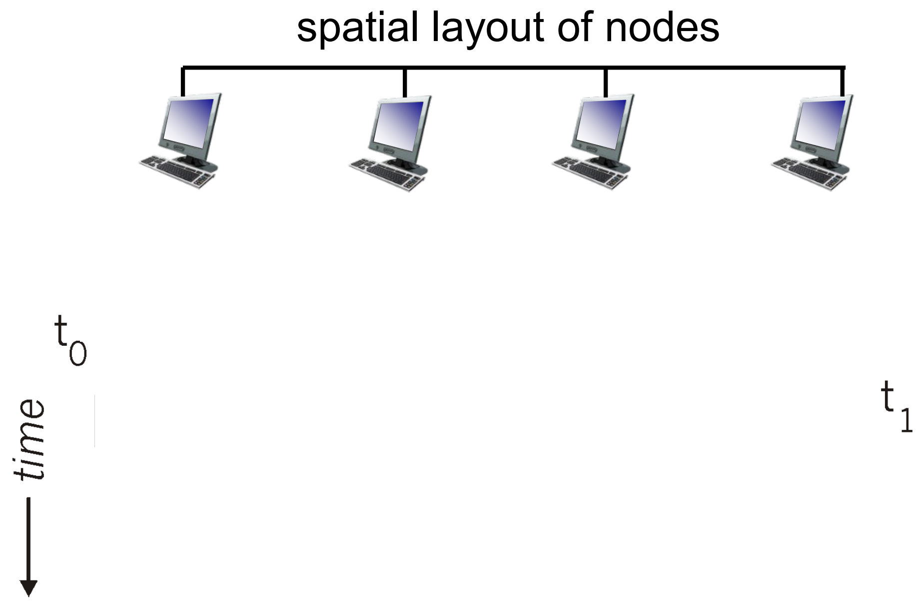
… choosing optimum p and then letting $n \rightarrow \infty$

= $1/(2e)$ = ***0.18***

→ worse than slotted Aloha!

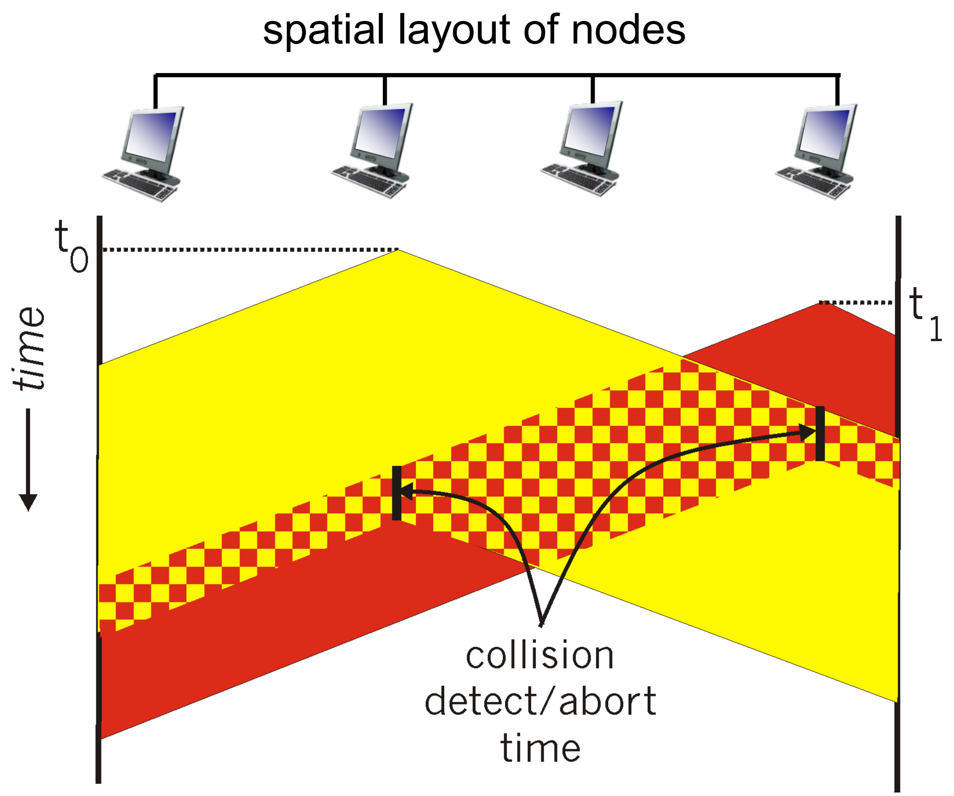
CSMA: collisions

- collisions *can* still occur with carrier sensing:
  - propagation delay means two nodes may not hear each other's just-started transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in determining collision probability



CSMA/CD:

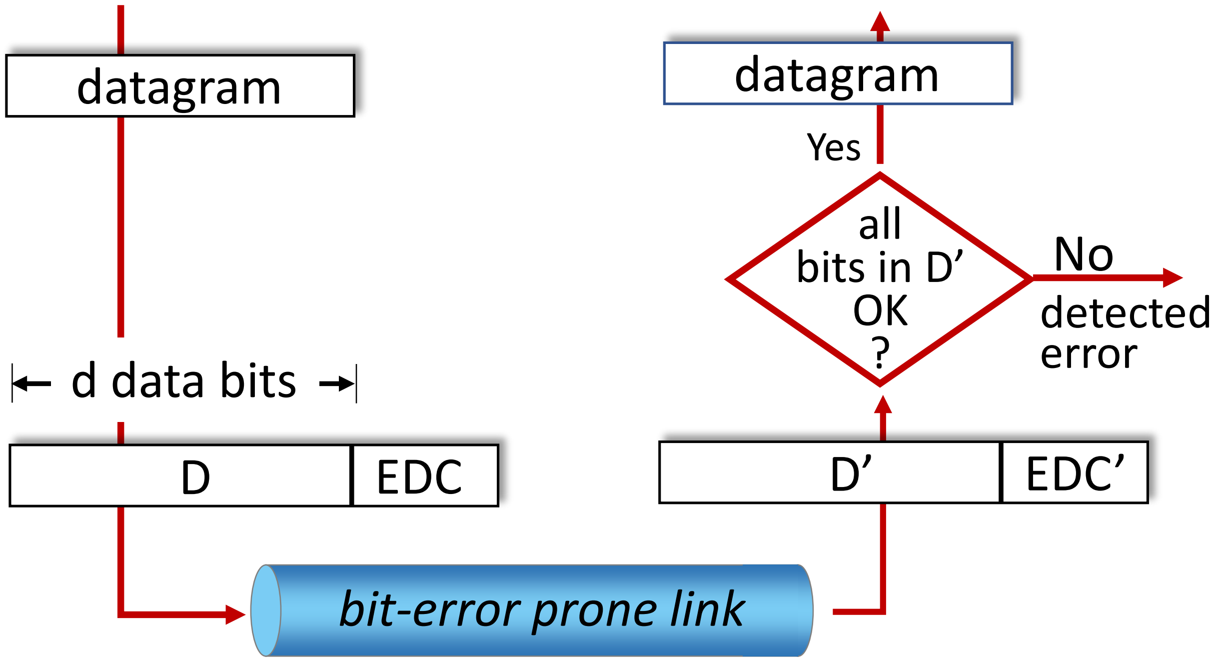
- CSMA/CD reduces the amount of time wasted in collisions
  - transmission aborted on collision detection



Error detection

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



Error detection not 100% reliable!

- protocol may miss some errors (rarely)
- larger EDC field yields better detection and correction

Parity checking

single bit parity:

- detect single bit errors

0111000110101011	1

⟵ d data bits ⟶ parity bit

Even parity: set parity bit so there is an even number of 1's

two-dimensional bit parity:

- detect *and correct* single bit errors

row parity

column parity

$$\begin{array}{cccc|c} d_{1,1} & \cdots & d_{1,j} & & d_{1,j+1} \\ d_{2,1} & \cdots & d_{2,j} & & d_{2,j+1} \\ \cdots & \cdots & \cdots & & \cdots \\ d_{i,1} & \cdots & d_{i,j} & & d_{i,j+1} \\ \hline d_{i+1,1} & \cdots & d_{i+1,j} & & d_{i+1,j+1} \end{array}$$

no errors:

$$\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \end{array}$$

detected and **correctable(!)** single-bit error:

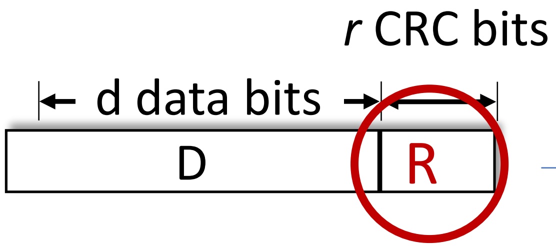
$$\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \end{array}$$

parity error

parity error

Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of *r+1* bits (given)



bit pattern

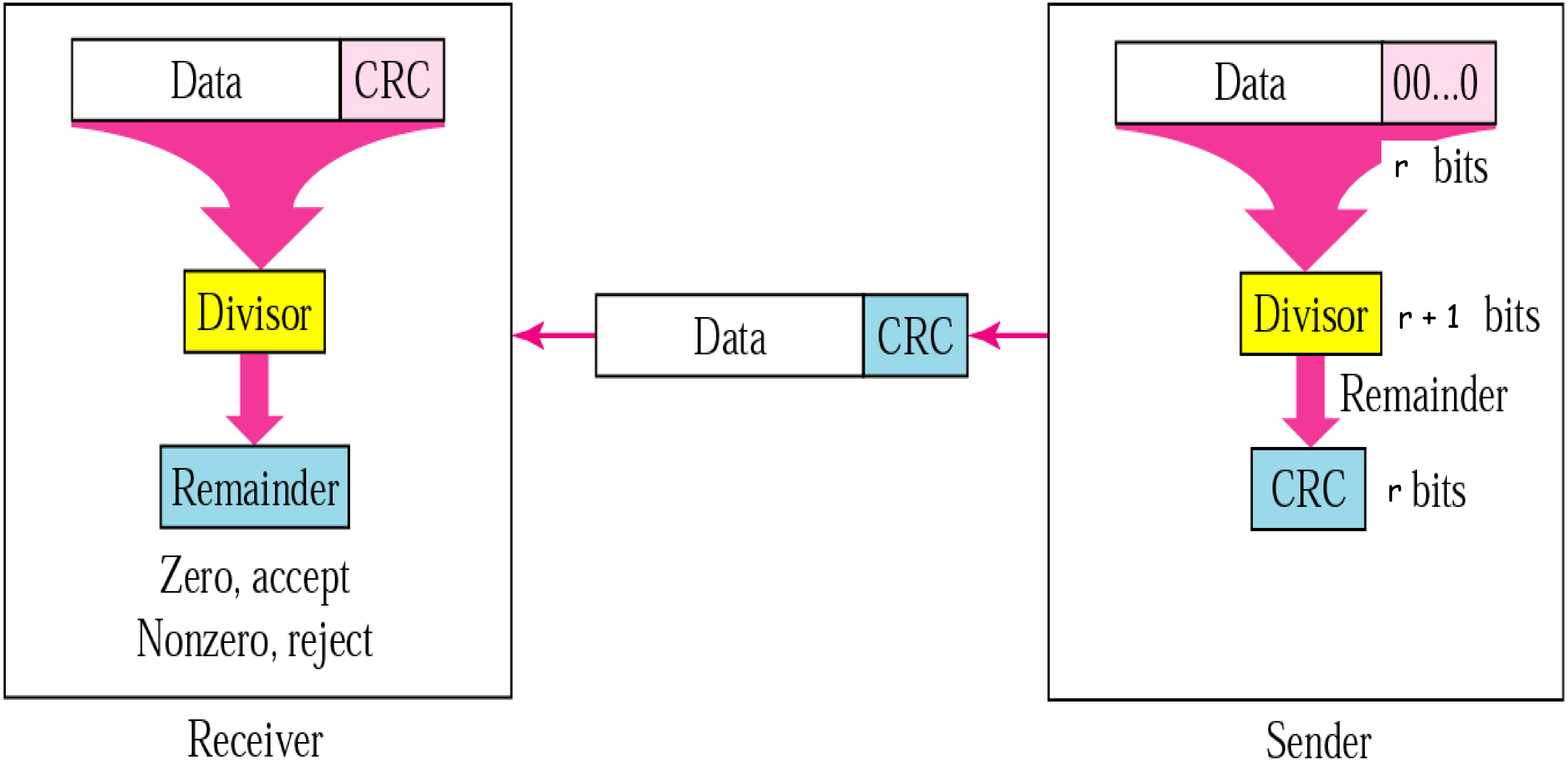
<D,R> = $D * 2^r$ XOR R

formula for bit pattern

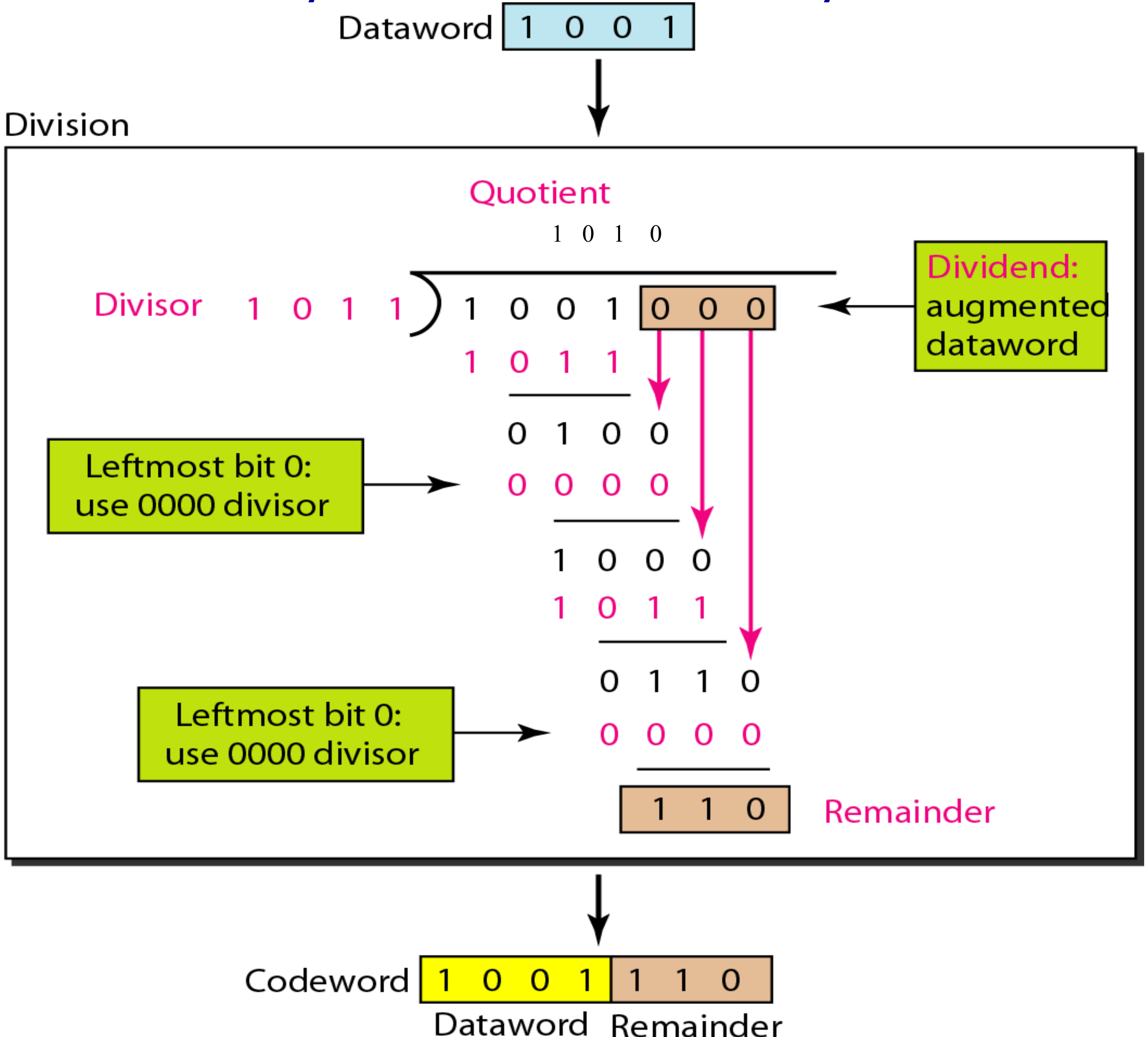
goal: choose *r* CRC bits, R, such that <D,R> exactly divisible by G (mod 2)

- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can detect all burst errors of up to r bits
- widely used in practice (Ethernet, 802.11 WiFi)

Cyclic redundancy check



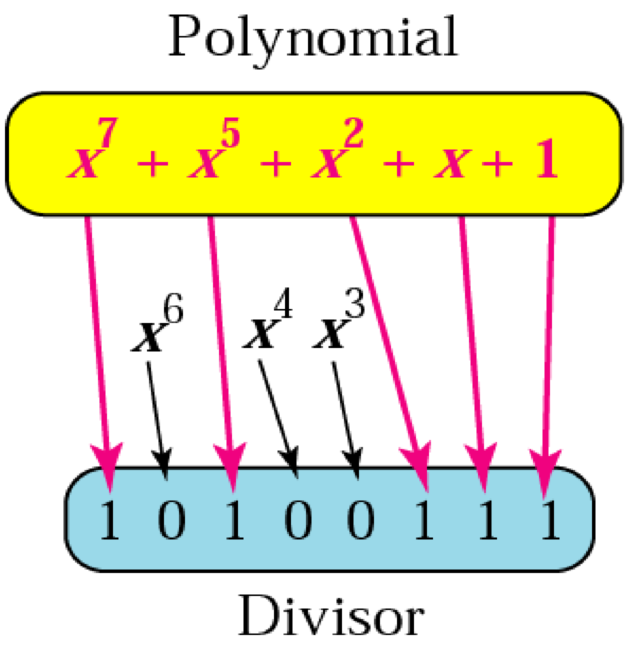
Cyclic redundancy check



In each step: check the **leading most significant bit**

- If it's 0: place a 0 in the quotient and XOR the current bits with 000.
- If it's 1: place a 1 in the quotient and XOR the current bits with the divisor

Clarification: CRC Error Detection
A polynomial representing a divisor (generator)

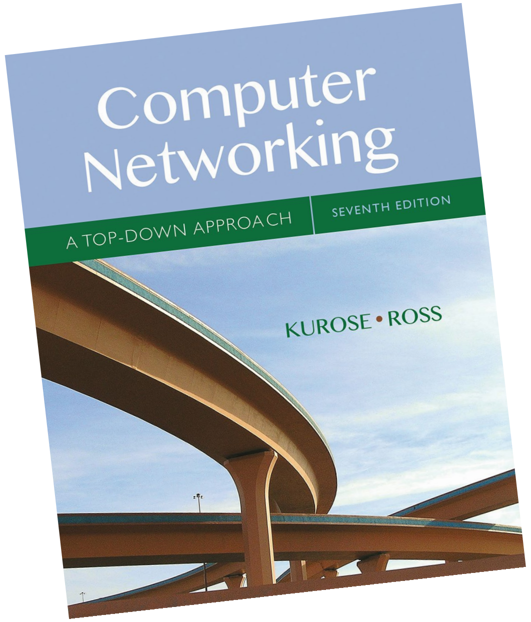


Standard polynomials

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
ITU-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
ITU-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Link layer, LANs: roadmap

- introduction
- multiple access protocols
- error detection, correction
- LANs
  - addressing, ARP
  - switches
- data center networking
- a day in the life of a web request



Chapter 6

MAC addresses

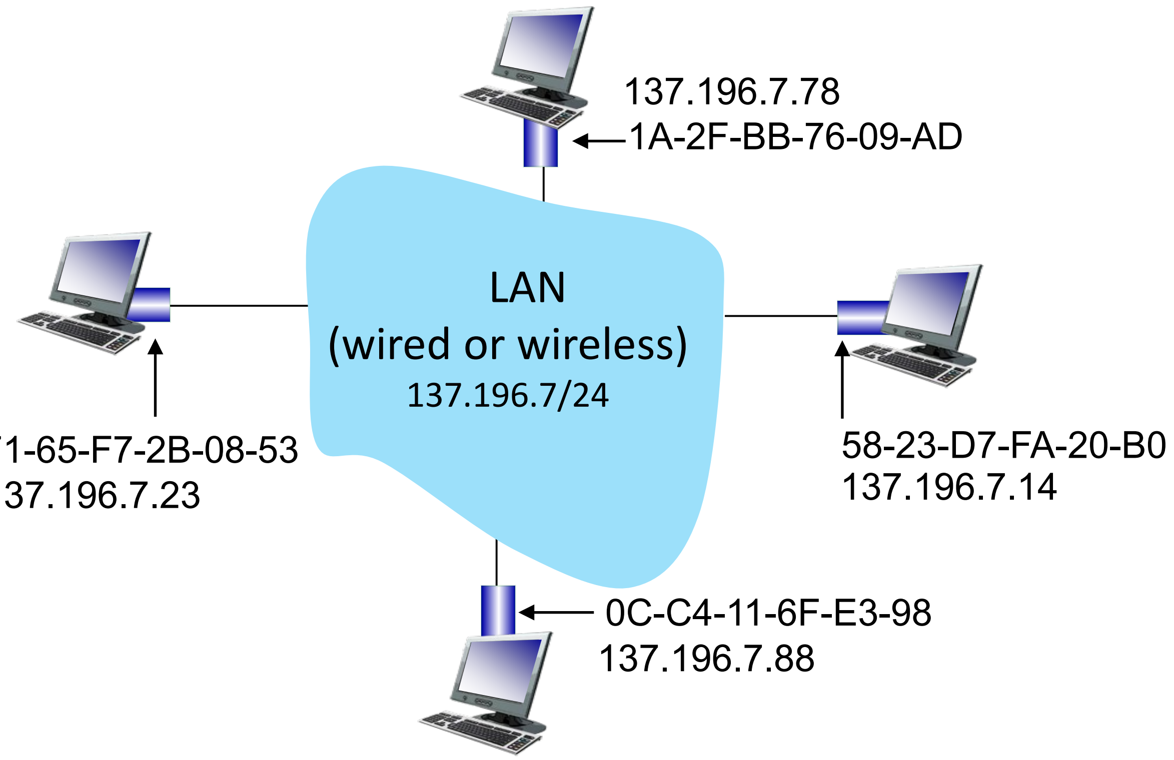
- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
  - used "locally" to get frame from one interface to another physically-connected interface (same subnet, in IP-addressing sense)
  - 48-bit MAC address (for most LANs) in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation
(each "numeral" represents 4 bits)

MAC addresses

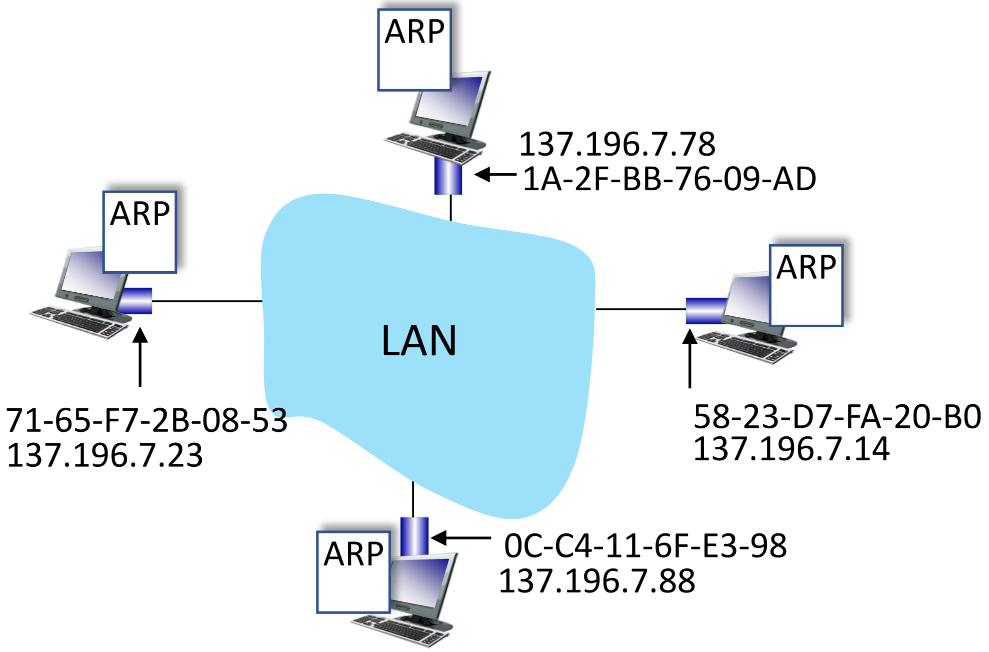
each interface on LAN

- has a (globally) unique 48-bit MAC address
- has a locally unique 32-bit IP address (as we've seen)



ARP: address resolution protocol

Question: how to determine interface's MAC address, given the IP address of the interface?



ARP table: each IP node (host, router) on LAN has a table

- IP/MAC address mappings for some LAN nodes:

  < IP address; MAC address; TTL>

- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP protocol in action

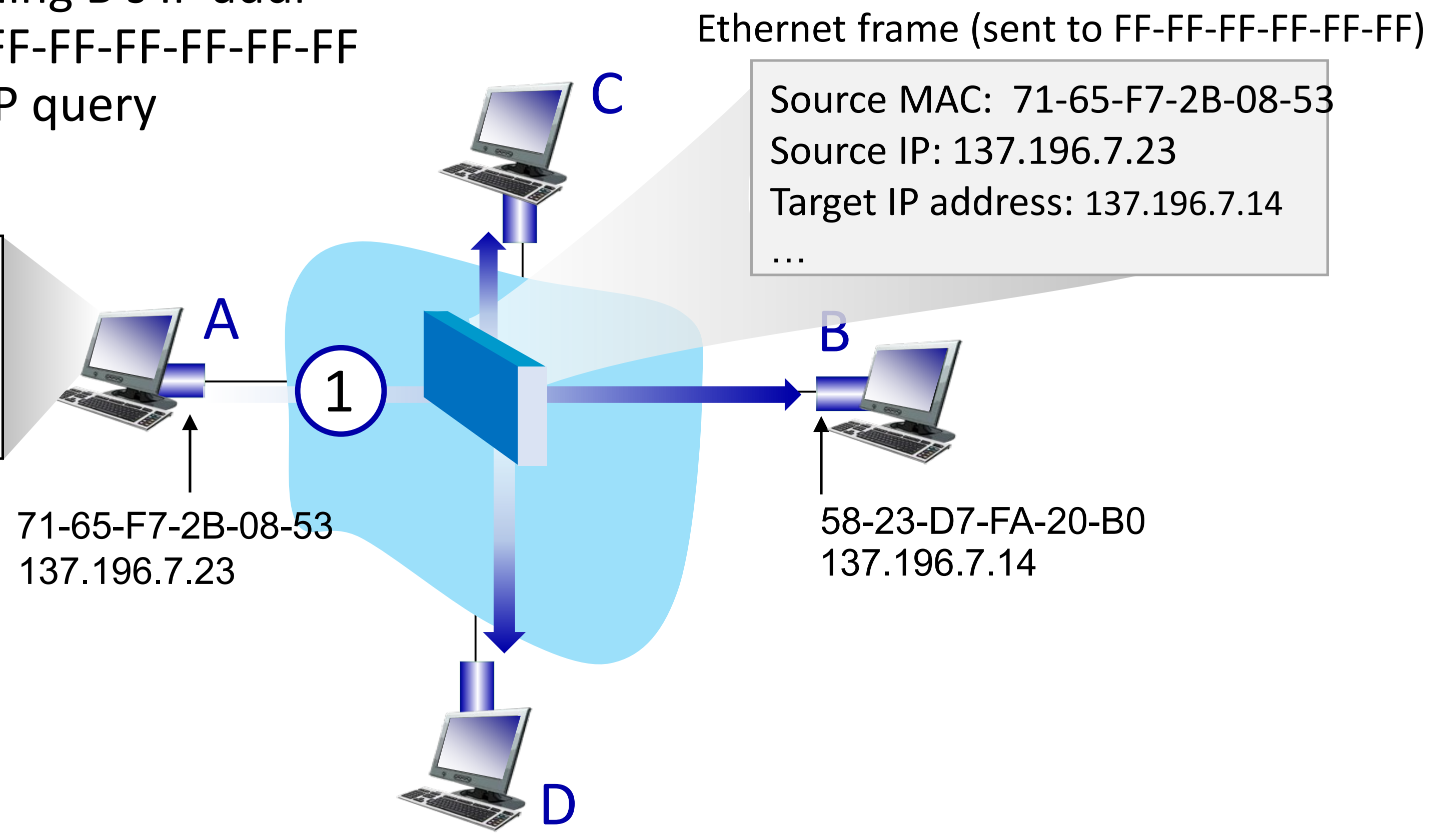
example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

1 A broadcasts ARP query, containing B's IP addr
- destination MAC address = FF-FF-FF-FF-FF-FF
- all nodes on LAN receive ARP query

ARP table in A

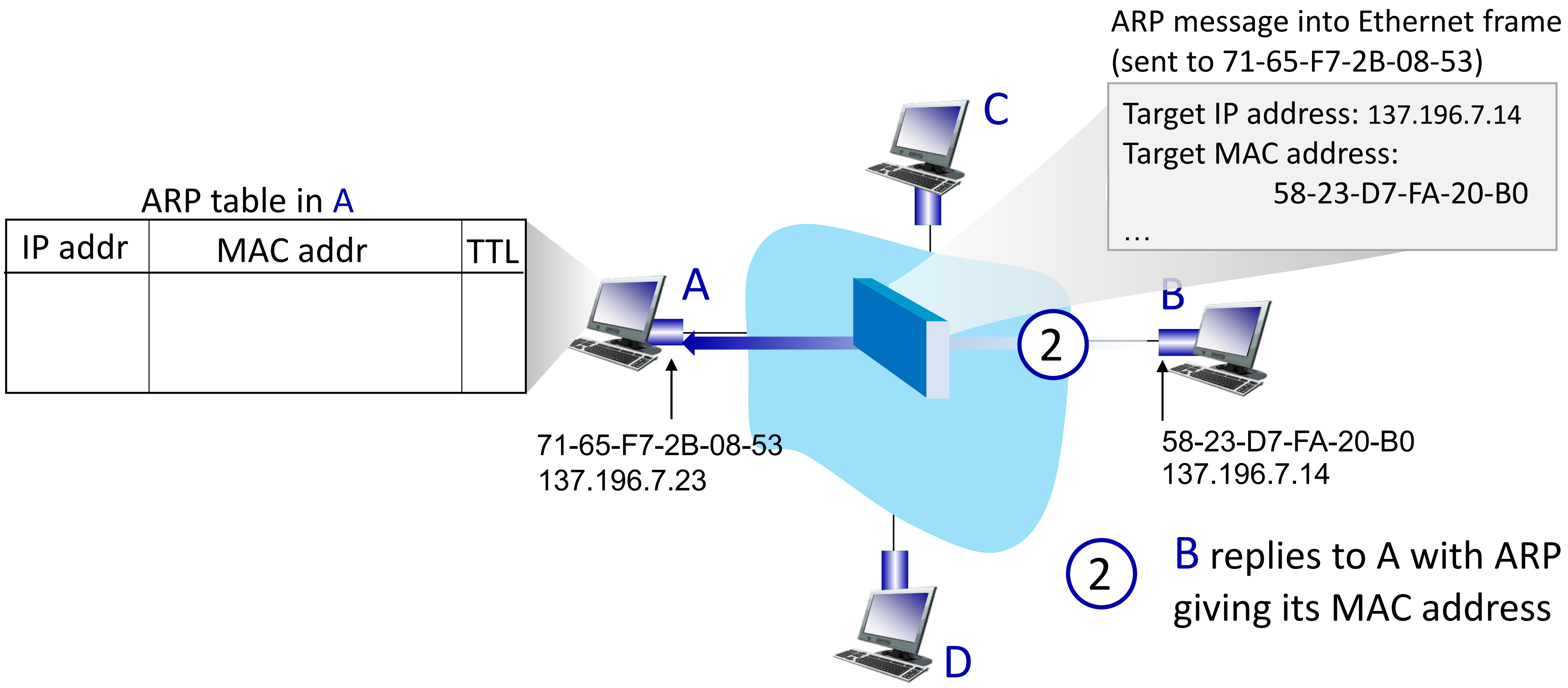
IP addr	MAC addr	TTL



ARP protocol in action

example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



2 B replies to A with ARP response, giving its MAC address

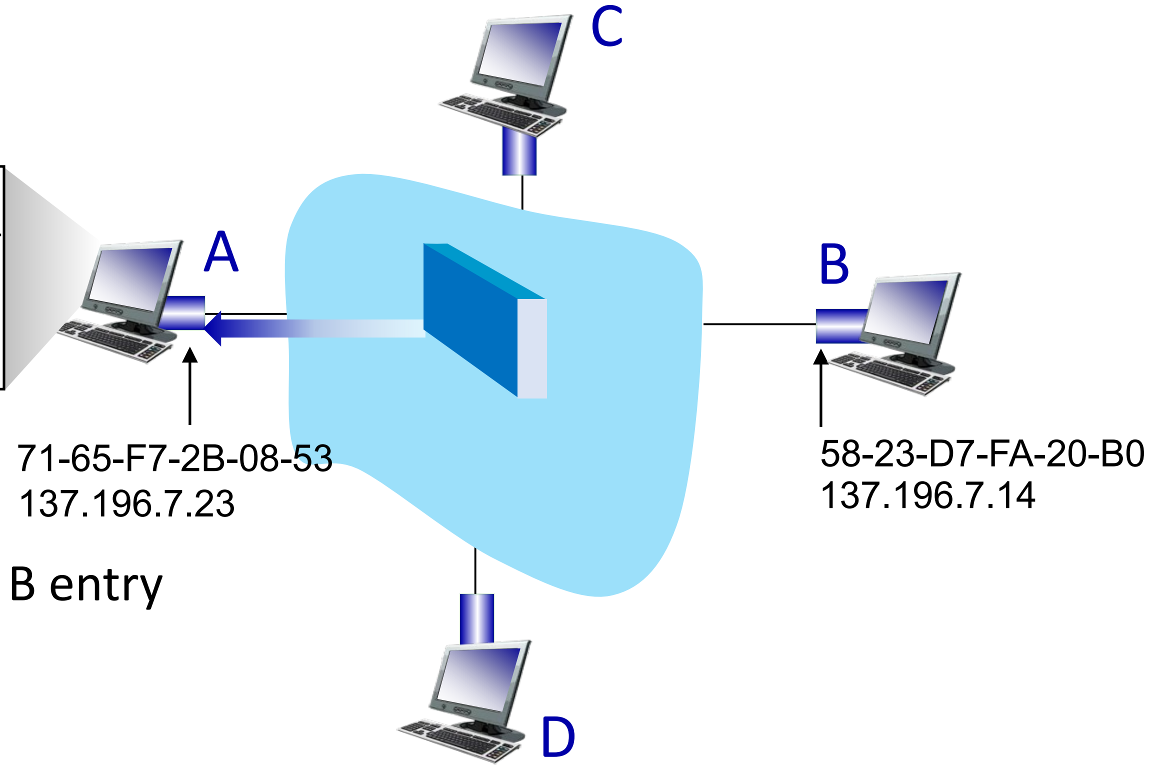
ARP protocol in action

example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

ARP table in A

IP addr	MAC addr	TTL
137.196.7.14	58-23-D7-FA-20-B0	500



3 A receives B's reply, adds B entry into its local ARP table

Ethernet switch

- Switch is a link-layer device: takes active roles
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, and *selectively* forward frame to one-or-more outgoing links, uses CSMA/CD to access each link
- transparent: hosts *unaware* of presence of switches
- plug-and-play, self-learning
  - switches do not need to be configured

Switch table

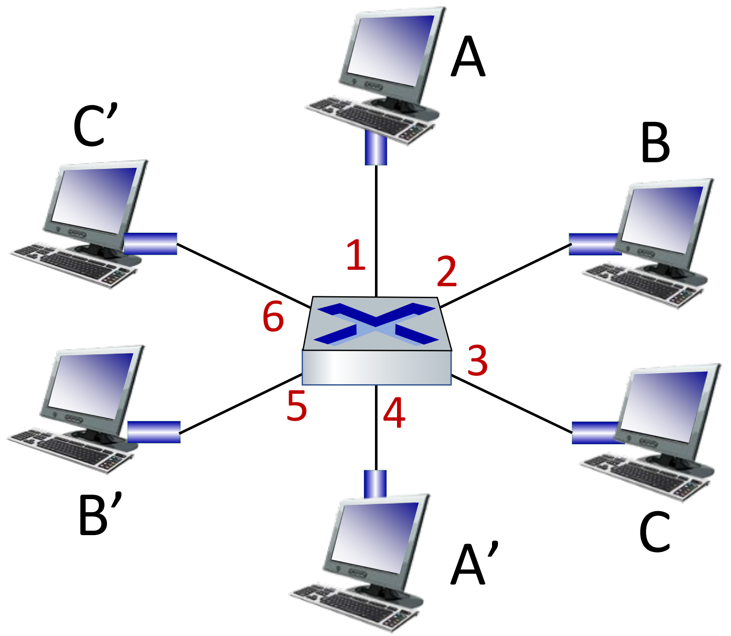
Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

A: a switch has a switch table:

- each entry: (MAC address of host, interface to reach host, time stamp)
- looks like a forwarding table!

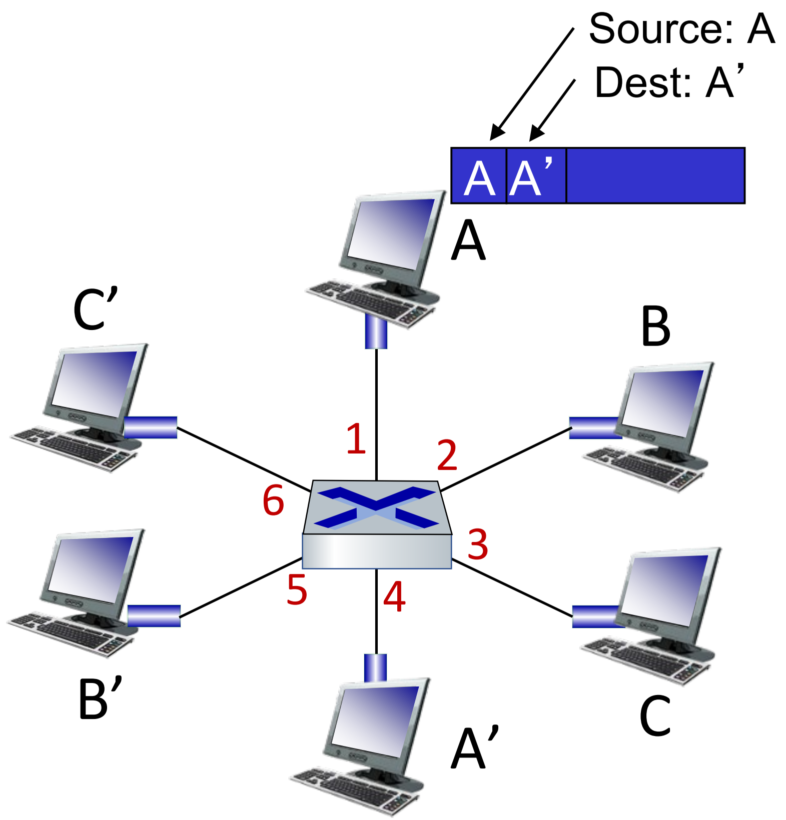
Q: how are entries created, maintained in switch table?

- something like a routing protocol?



Switch: self-learning

- switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch "learns" location of sender: incoming LAN segment
  - records sender/location pair in switch table

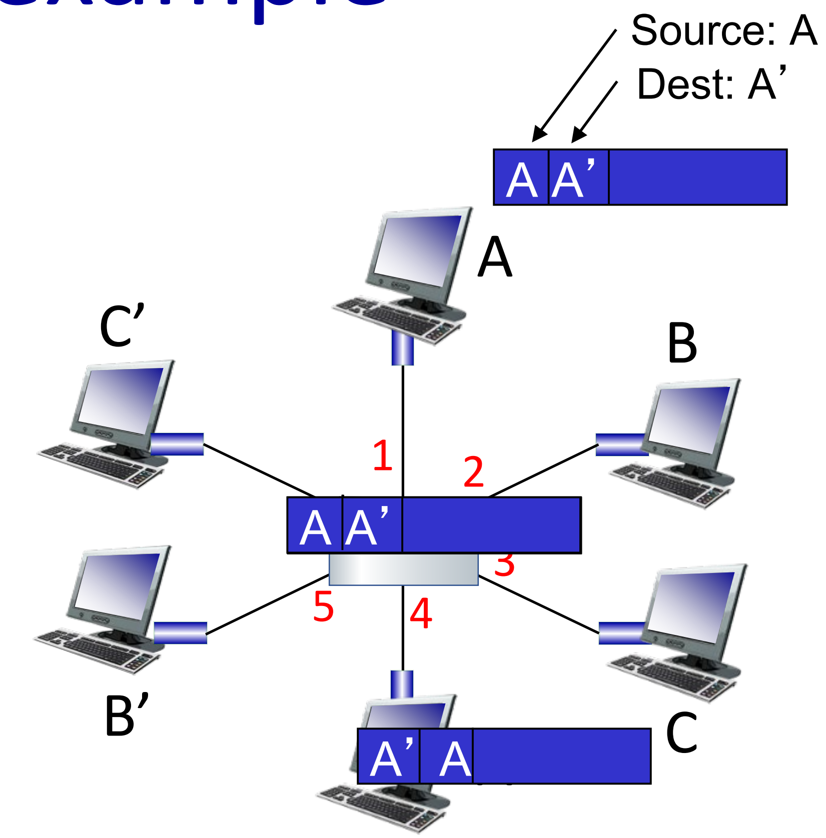


MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

Self-learning, forwarding: example

- frame destination, A', location unknown: flood
- destination A location known: selectively send on just one link



MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

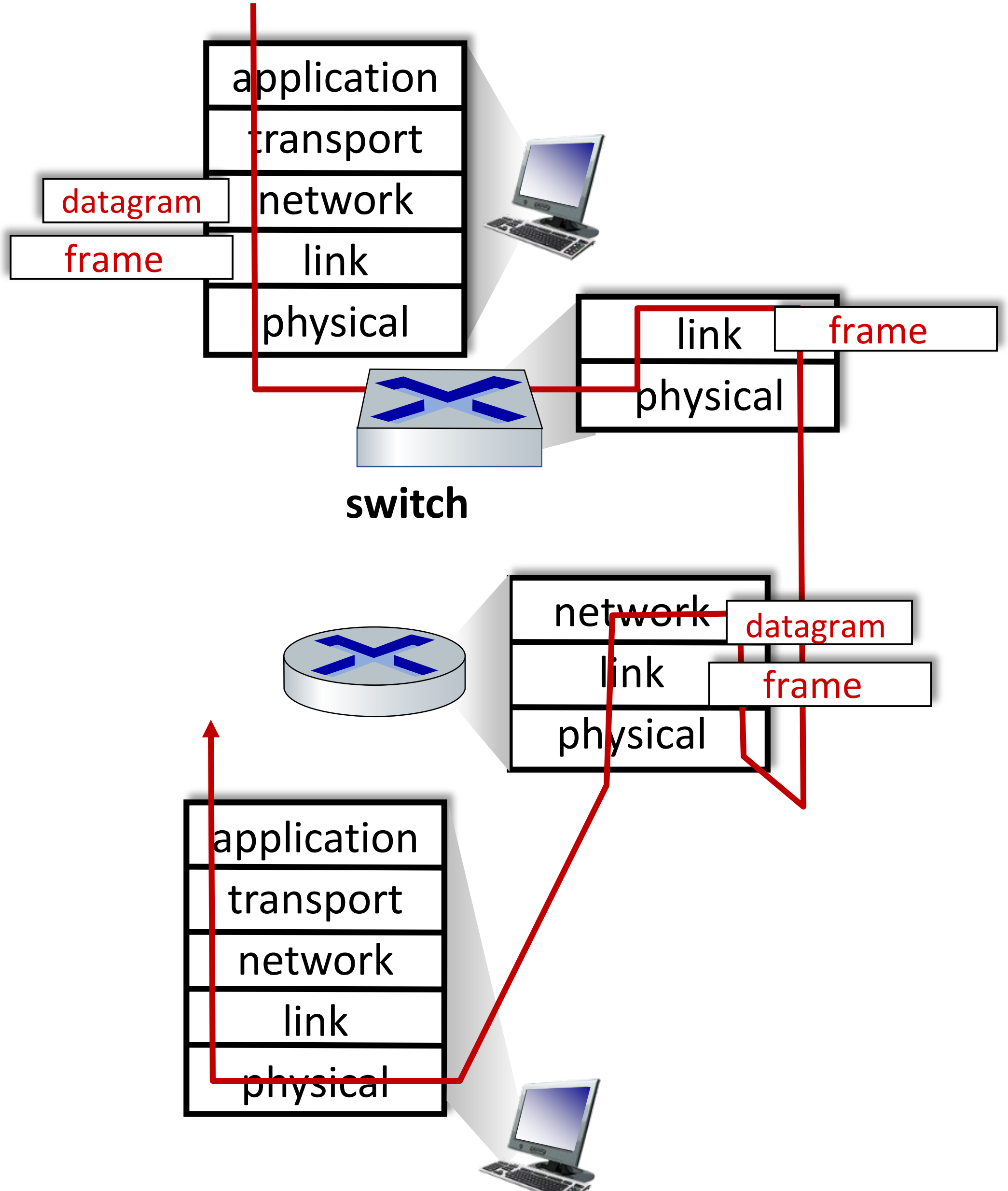
Switches vs. routers

both are store-and-forward:

- *routers*: network-layer devices (examine network-layer headers)
- *switches:* link-layer devices (examine link-layer headers)

both have forwarding tables:

- *routers:* compute tables using routing algorithms, IP addresses
- *switches:* learn forwarding table using flooding, learning, MAC addresses



Final Exam

- 2 hours (written) +15 minutes (reading)
- An open book exam
- 14 questions
  - Each question contains 2 to 4 parts
  - Short answers (one or two lines)
  - Long answer (derivation, calculation, proof/justification, 5 to 10 lines)
  - Tips: Review questions and solutions of 2 homework assignments and 1 programing assignment, and 12 tutorial sessions

What you are (NOT) allowed to bring with you

- This is an open book examination.
- Students are allowed to use the following materials/aids:
- Approved calculator, lecture slides, tutorial slides, personal notes.
- Materials/aids other than those stated above are not permitted.
- Students will be subject to disciplinary action if any unauthorized materials or aids are found on them.
- This question paper should NOT be taken away from the exam room.
- All answers must be written in the exam paper.

Date and Time of the Final Exam

- Date: Dec 16, 2024
- Venue: Not known yet
- Time: 2:00pm -4:15pm