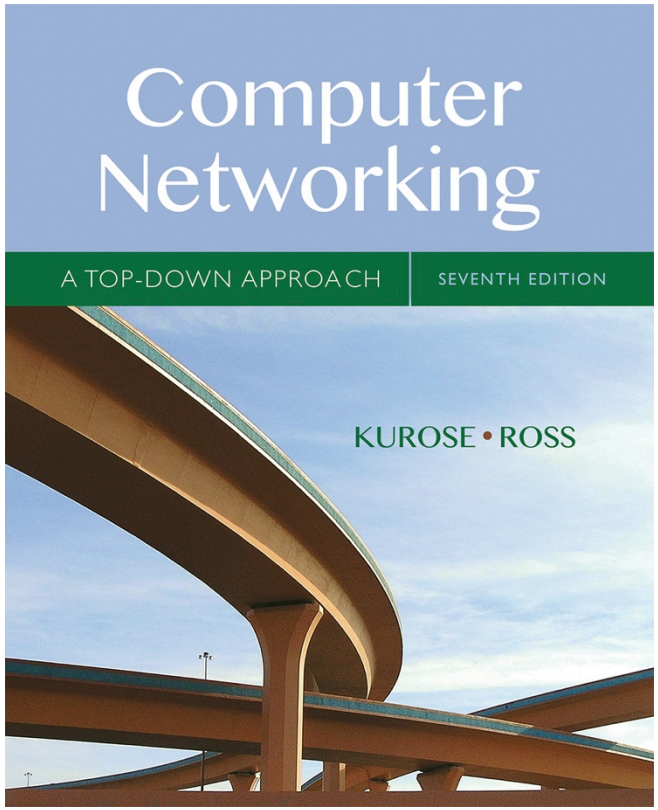
Final Review



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

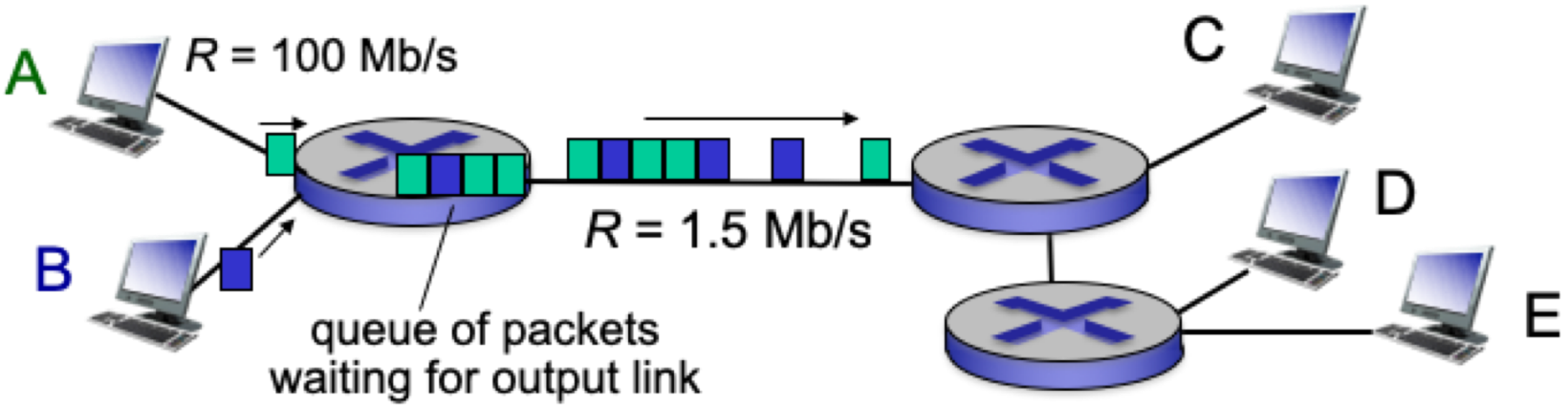
Instructions

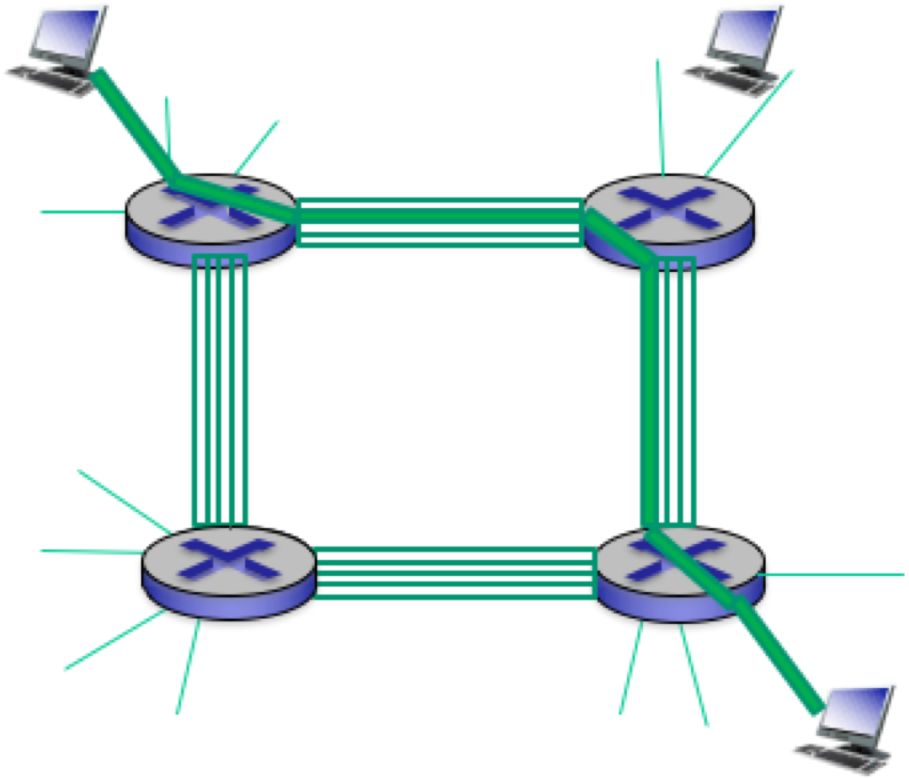
- Put your name initials on each sheet of paper
- The exam is closed book. You cannot use any computer or phone during the exam. You can use calculator, but not one from your phone or laptop
- You are allowed to bring a cheat sheet, which will be collected along with the exam paper sheets
- You have 75 minutes to complete the exam.
- Show all your work. Partial credit is possible for an answer, but only if you show the intermediate steps in obtaining the answer. If you make a mistake, it will also help the grader show you where you made a mistake

Included Contents

- Foundation (Chapter 1: 1.1 – 1.5)
- Application Layer (Chapter 2: 2.2 and 2.4)
- Transport Layer (Chapter 3: 3.1 – 3.7)
- Network Layer – Data Plane (Chapter 4: 4.1 – 4.4)
- Network Layer – Control Plane (Chapter 5: 5.1 – 5.4)
- Link Layer and LANs – (Chapter 6: 6.1- 6.8)

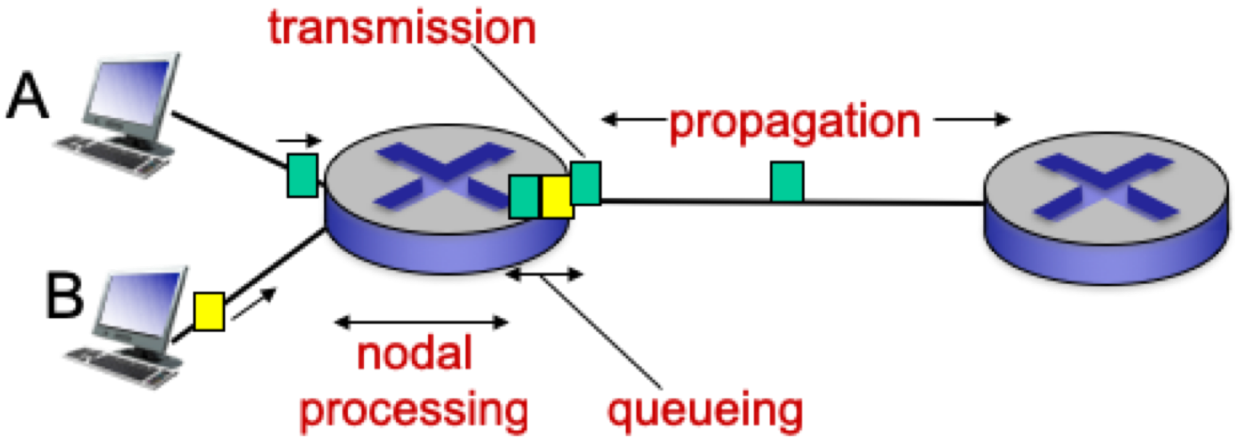
Packet Switching vs Circuit Switching





- Advantages
- Disadvantages

Four sources of packet delay



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

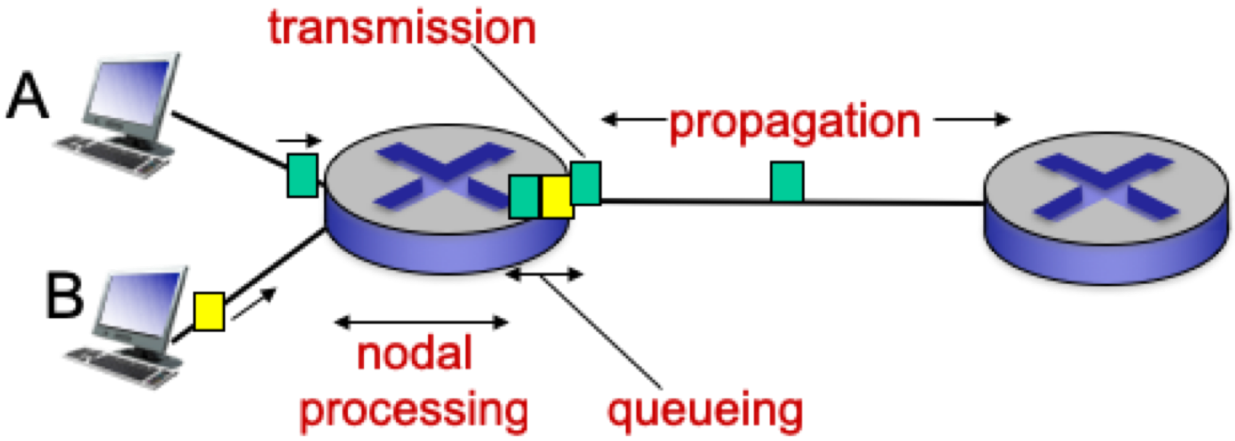
d_{proc}: nodal processing

- 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

Four sources of packet delay



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

d_{trans}: transmission delay:

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

d_{prop}: propagation delay:

- *d*: length of physical link
- *s*: propagation speed (~2×10^8 m/sec)
- $d_{prop} = d/s$

d_{trans} and d_{prop} *very* different

Homework Question

Q: Consider sending a packet from a source host to a destination host over a fixed route. List the delay components in the end-to-end delay. Which of these delays are constant and which are variable?

A:

- Propagation delay (d_{prop}) = d/s (dependent on path)
- Transmission delay (d_{trans}) = L/R (dependent on path)
- Queuing delay (d_{queue}) = (dependent on load)
- Processing delay (d_{proc}) = (minimal-insignificant/node)
- Number of links (Q) = (dependent on path)

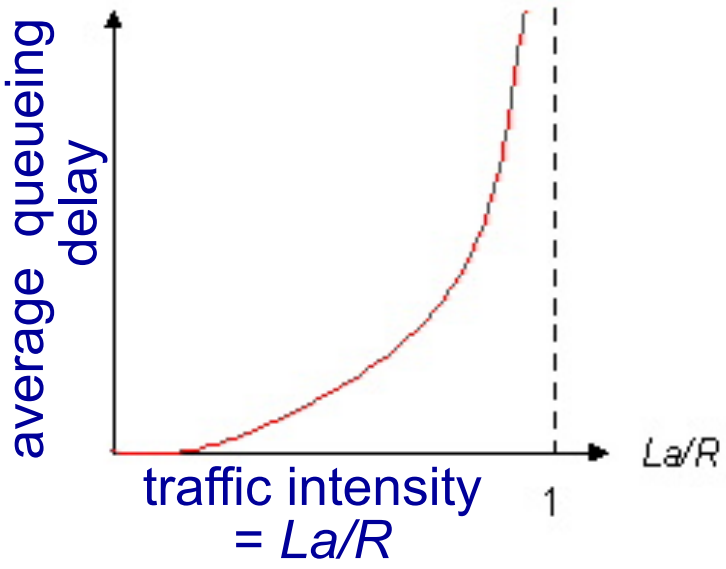
Homework Question

Q: What is the difference between transmission delay and propagation delay? Will the length of the packet affect the propagation delay and why?

A: 1) Transmission delay is the amount of time it takes to push a packet into a link while the propagation delay is the amount of time it takes for a packet to travel over a link. 2) No. Cause the propagation delay is determined by the length of physical link and the propagation speed

Queueing delay

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



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

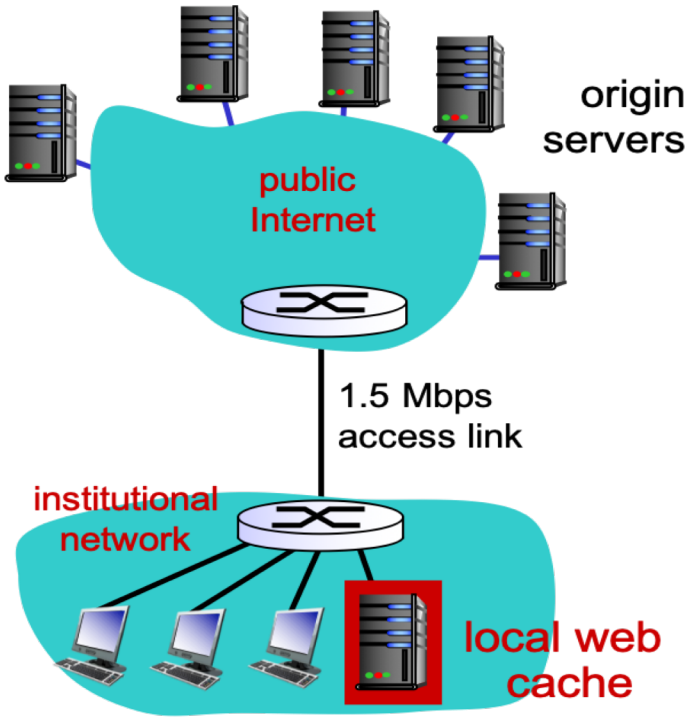
La/R ~ 0

La/R -> 1

Homework Question

Q: Consider the following institutional network system. Suppose the average object size is 100k bits and the average rate from browsers to origin servers is 15 requests/sec. (ignore the LAN delay in the following questions)

Assume that R_{access} = 1.5 Mbps and the RTT on the Internet side of the access link is 2 sec.



Homework Question

a. Considering the queueing delay in the access router, the access delay could be calculated with the equation $D_{access} = \frac{D_{Trans}}{(1-traffic\ intensity)}$. D_{Trans} represents the transmission delay at the access link. Without web cache, what is the total average response time?

A: $traffic\ intensity = \frac{La}{R}$, L is packet length, a is the average packet arrival rate and R is the link bandwidth. So, $traffic\ intensity = \frac{100{,}000 \times 15}{1.5 \times 10^6} = 1.$

As the traffic intensity approaches 1, the queueing delay becomes very large and grows without bound. Thus, there is no guarantee the packets will be delivered

Homework Question

b. Compare this result with the situation where R_{access} = 100 Mbps

A: In this case, $traffic\ intensity = \frac{100{,}000 \times 15}{100 \times 10^6} = 0.015$. Then, $access\ delay = \frac{D_{Trans}}{(1 - traffic\ intensity)} = \frac{100{,}000/100 \times 10^6}{(1 - 0.015)} = 0.1015$ secs.

So, $total\ delay = 2 + 0.102 \approx 2.1$ secs. Therefore, the total delay is significantly shortened by increasing the access link bandwidth.

Total delay = Internet delay + access delay + LAN delay

Homework Question

c. Suppose the local web cache satisfy 60% of the requests, the remaining 40% requests will be satisfied by the origin web servers. What is the total response time in this case?

A: Assume the capacity of local LAN is 100 Mbps, then $total\ delay = 0.6 \times (\sim msec) + 0.4 \times 2.01 \approx 0.8$ secs

Homework Question

Q: Two hosts, A and B, are directly connected via a link $R = 1$ Mbps. The distance between A and B is $10{,}000$ kilometers and the propagation speed over the link is 2.5×10^8 m/s.

a. How long does it take to send a file of 20,000 bits from A to B?

A: $total\ time = D_{transmission} + D_{propagation} = \frac{20{,}000}{10^6} + \frac{10 \times 10^6}{2.5 \times 10^8} = 0.06$ secs

Homework Question

b. Suppose now the file is broken up into 5 packets with each packet containing 4,000 bits. Suppose that each packet is acknowledged by the receiver and the transmission time of an acknowledgment packet is negligible. Finally, assume that the sender cannot send a packet until the preceding one is acknowledged. How long does it take to send the file?

A: $total\ time = 5 \times (D_{transmission} + 2 \times D_{propagation}) =$

$5 \times (\frac{4{,}000}{10^6} + 2 \times \frac{10 \times 10^6}{2.5 \times 10^8}) = 0.42$ secs

Homework Question

c. Now assume there are two separate links between host A and host B, i.e., R_1 = 500 kbps and R_2 =10 Mbps. Roughly, how long does it take to send the same file?

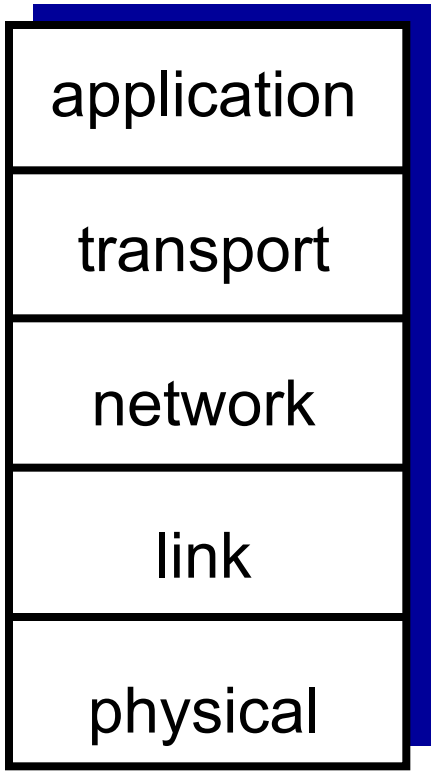
A: The bottleneck link in this case is R_1. Thus, the total end-to-end delay is $\frac{20{,}000}{5 \times 10^5} = 0.04$ secs roughly.

bottleneck link

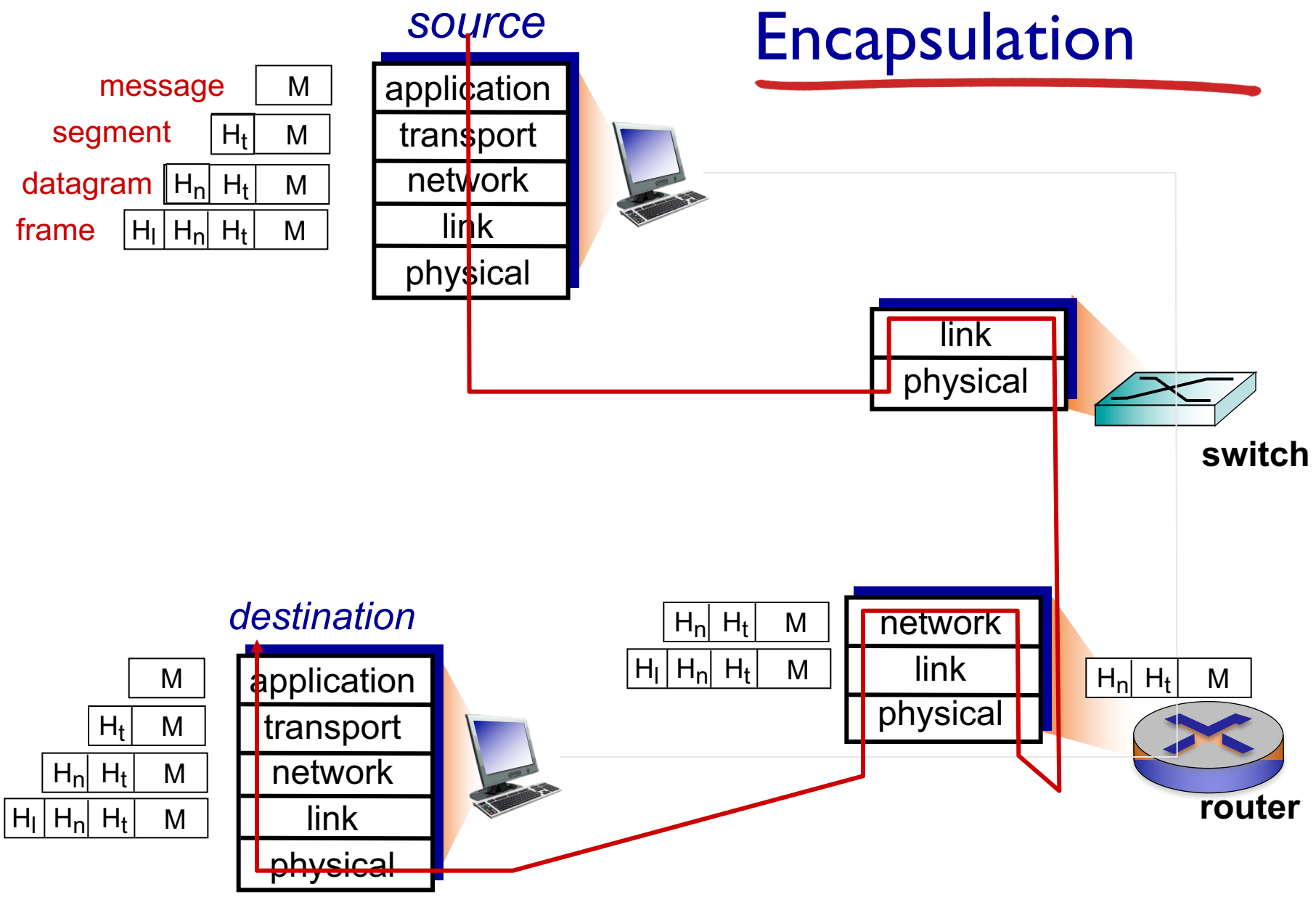
link on end-end path that constrains end-end throughput

Internet protocol stack

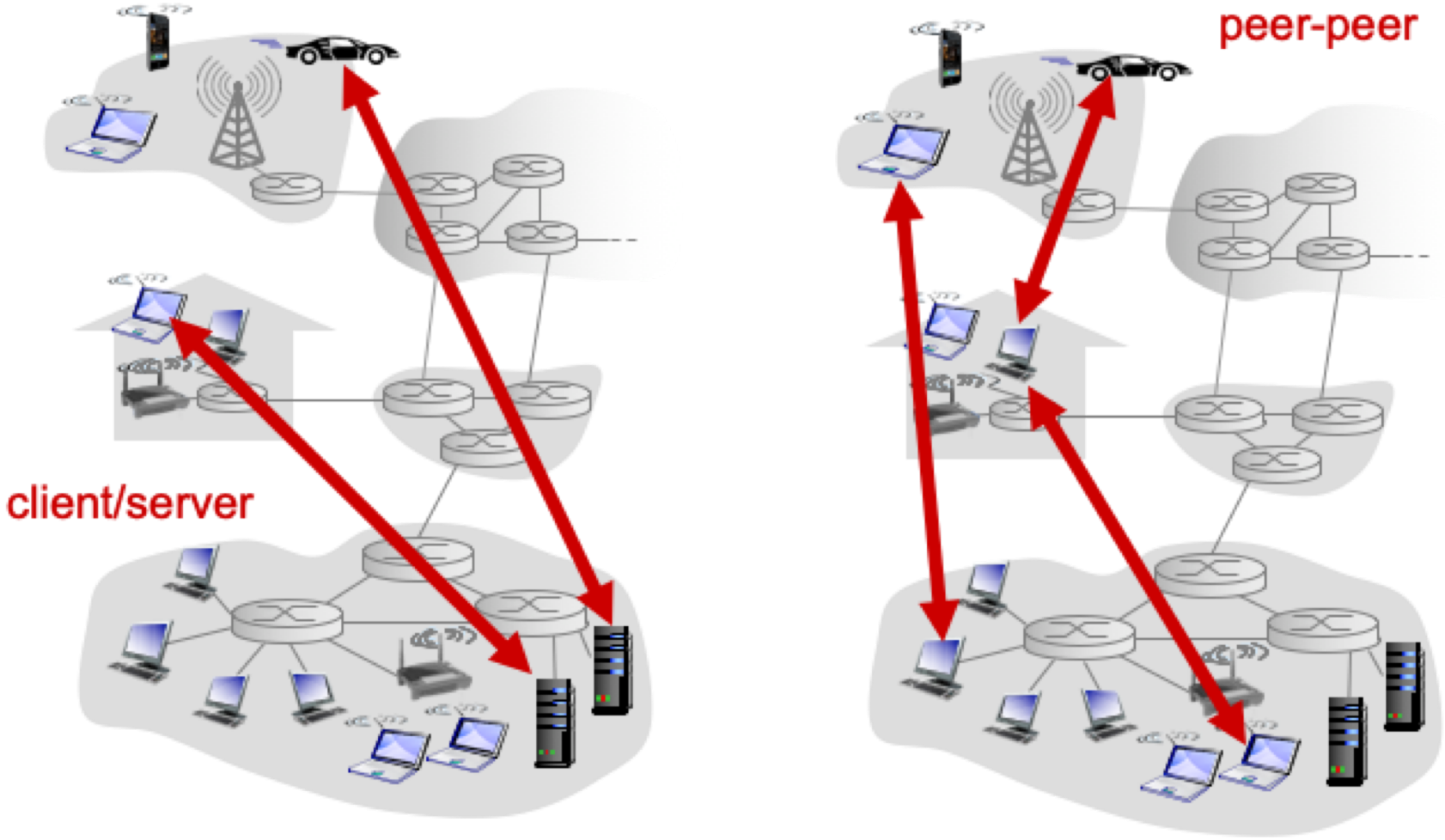
- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



Encapsulation



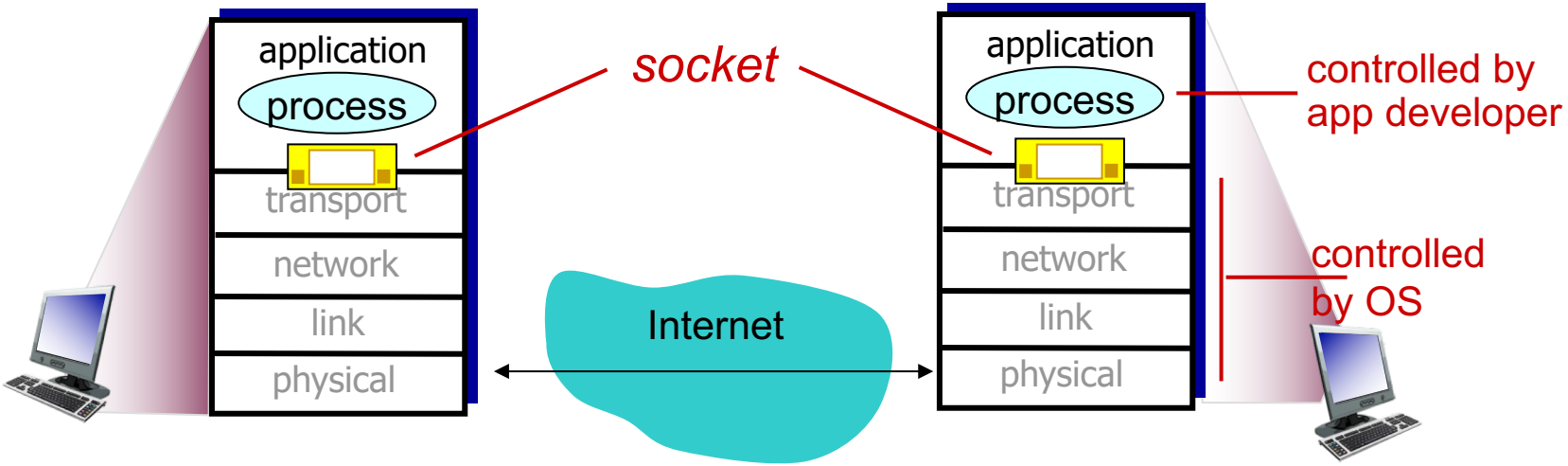
Application architectures



Client-server vs P2P

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?

- *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80

Internet transport protocols services

TCP service:

- *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 throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

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

Q: why bother? Why is there a UDP?

Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Socket programming with UDP

UDP: no "connection" between client & server

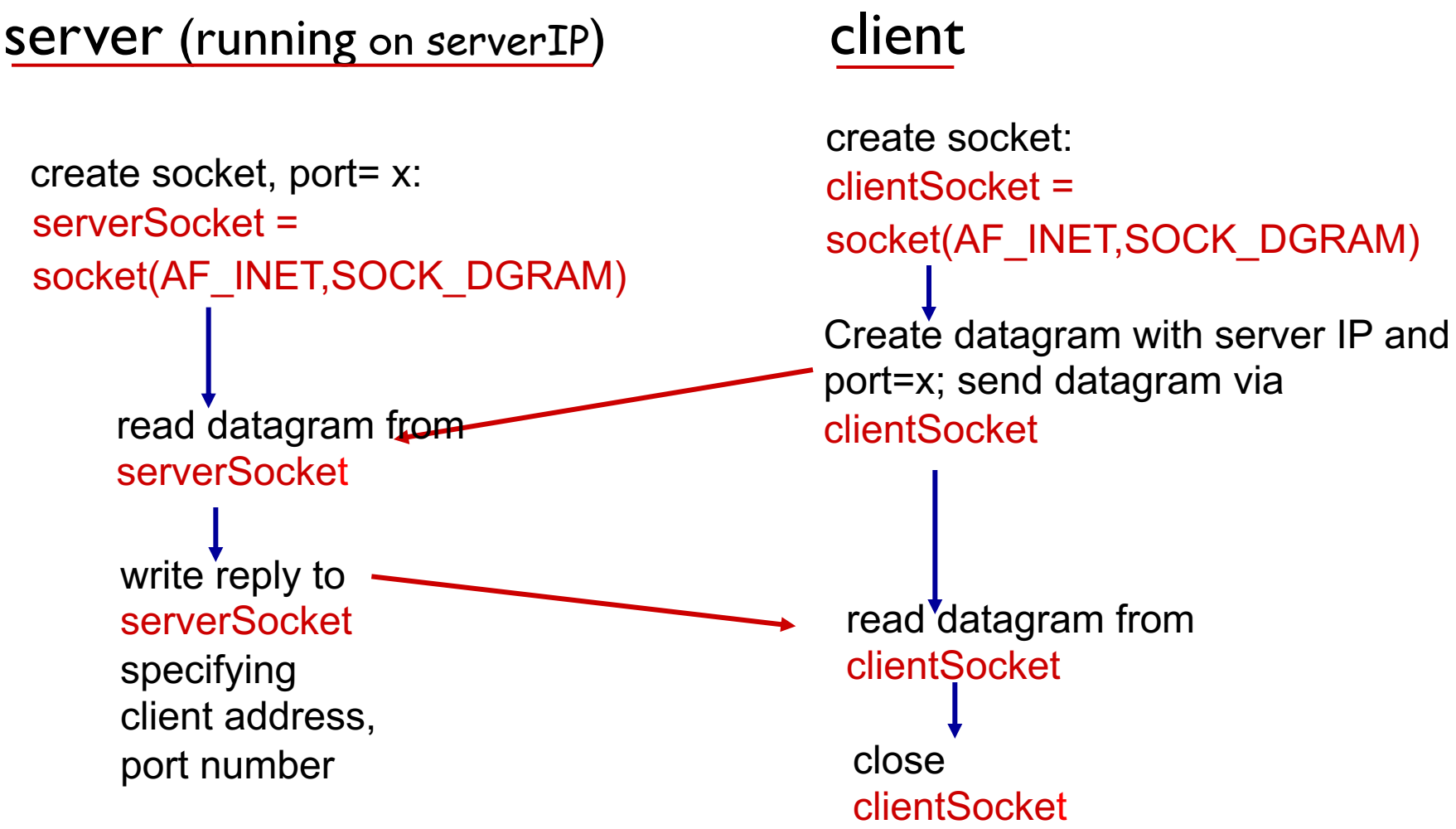
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

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

Application viewpoint:

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

Client/server socket interaction: UDP



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:

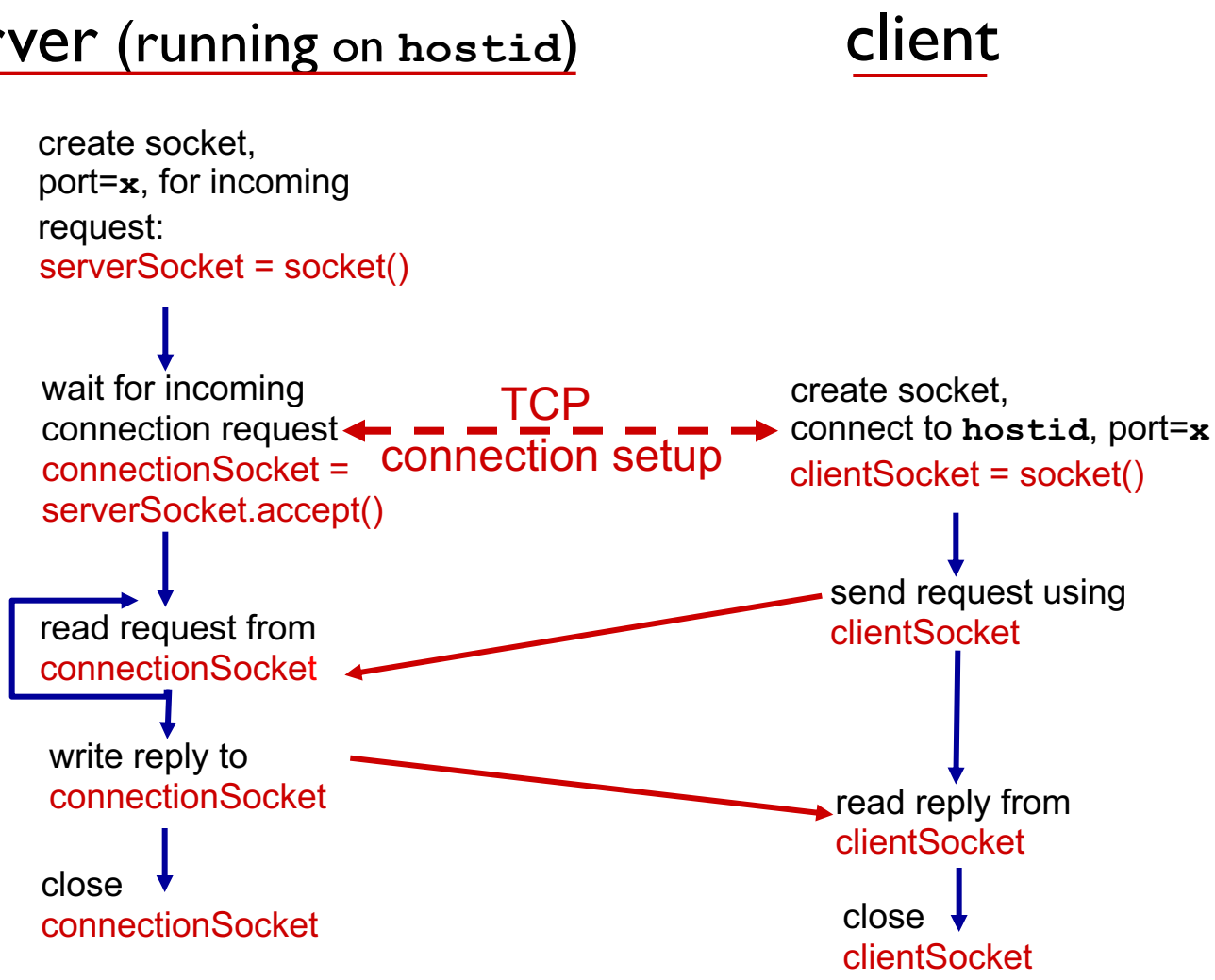
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

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

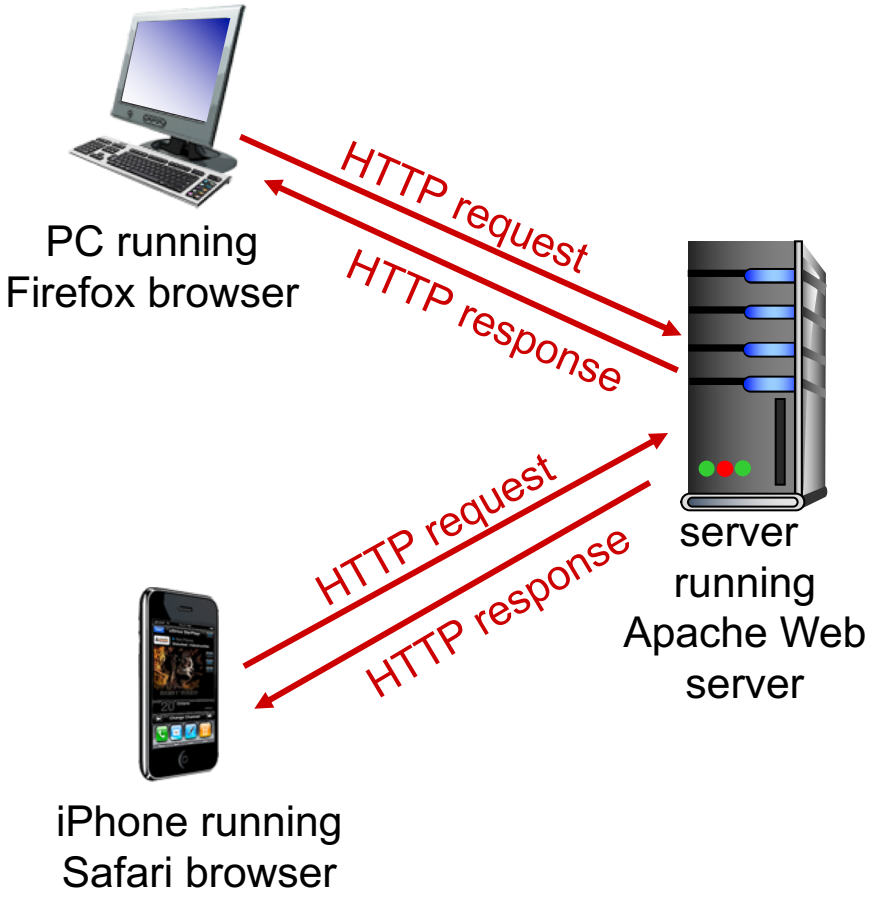
Client/server socket interaction: TCP



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 connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

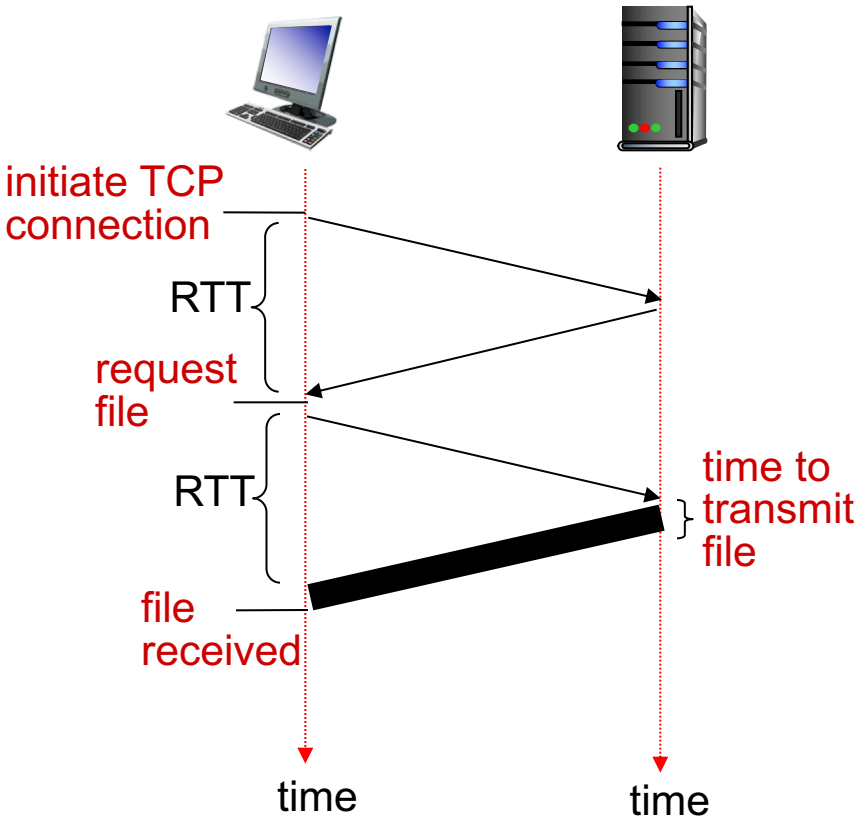
Non-persistent HTTP: response time

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

HTTP response time:

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

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

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

persistent HTTP:

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

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

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

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Caching example:

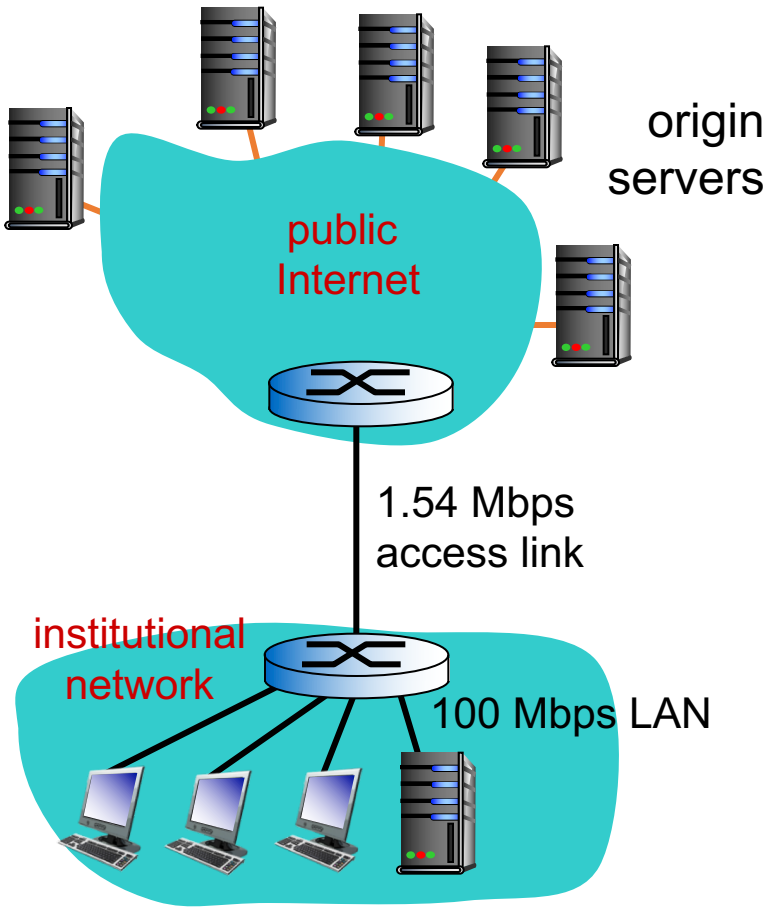
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from public router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 1.5%
- access link utilization = 99% *problem!*
- total delay = Internet delay + access delay + LAN delay

  = 2 sec + secs + usecs



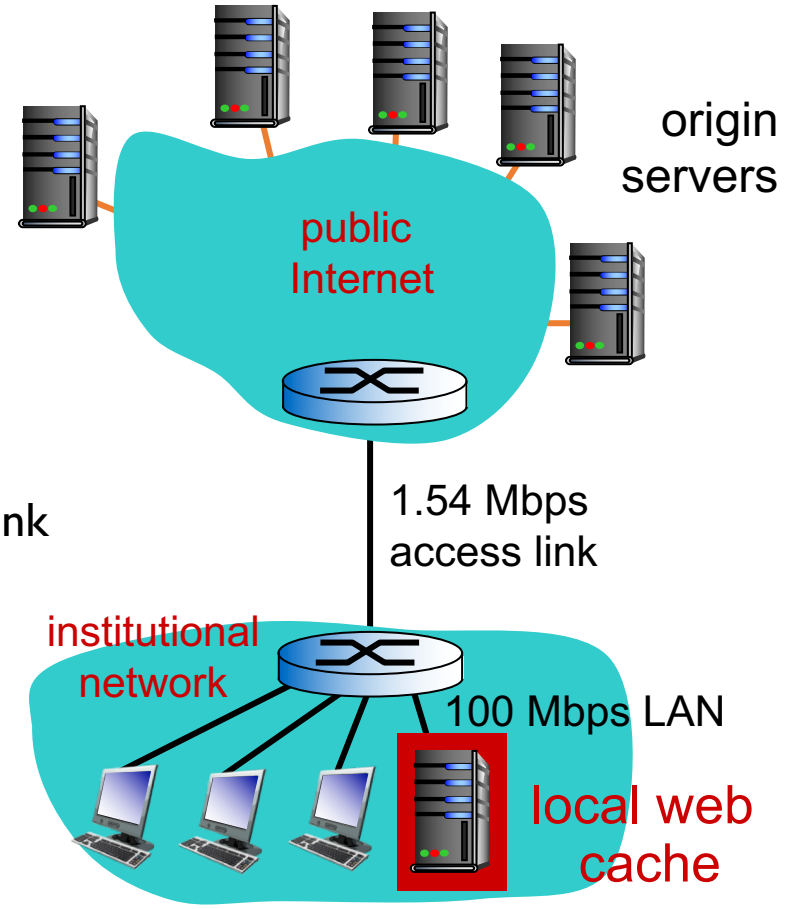
Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin

- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link

  = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)

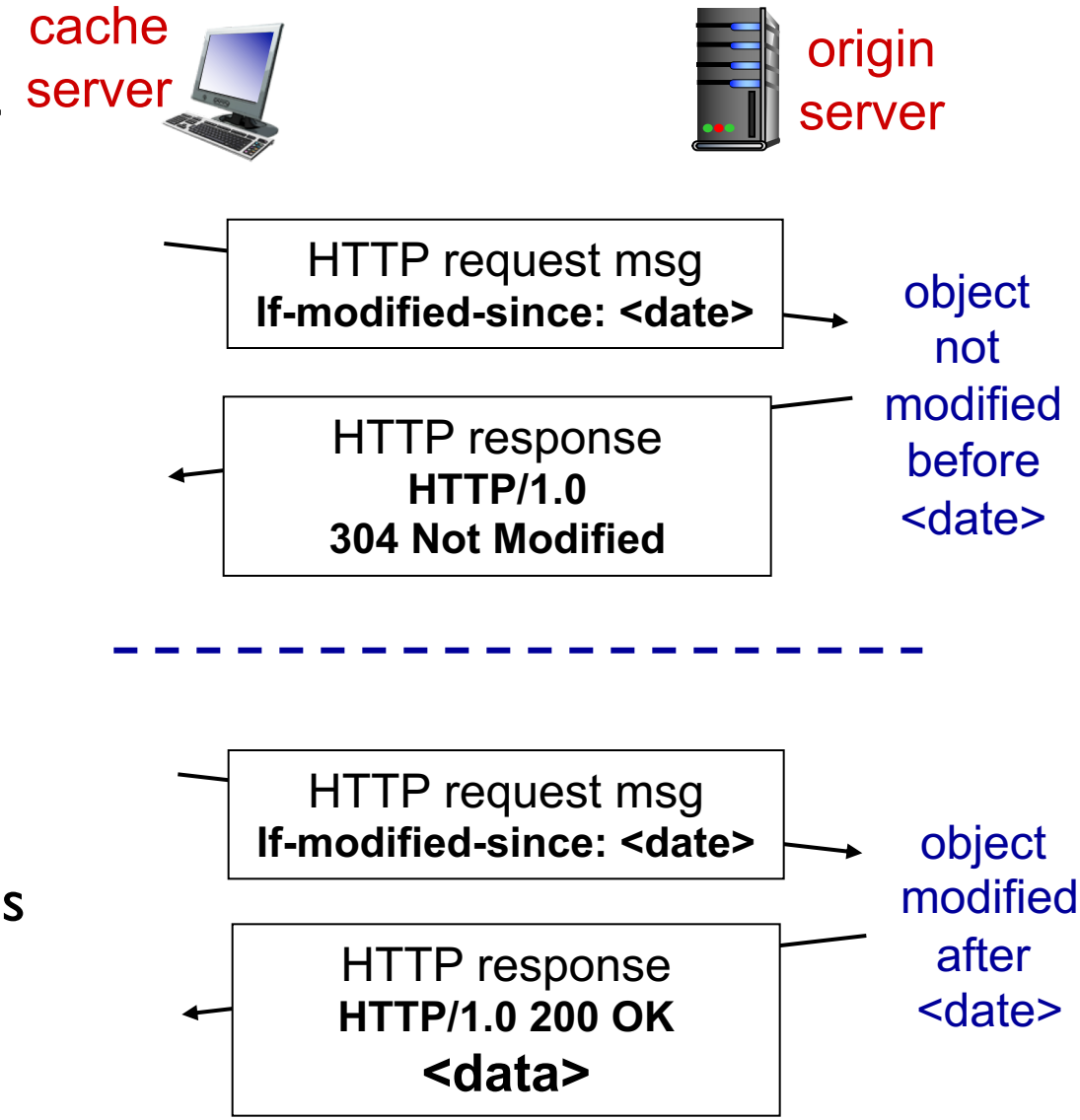


Conditional GET

- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

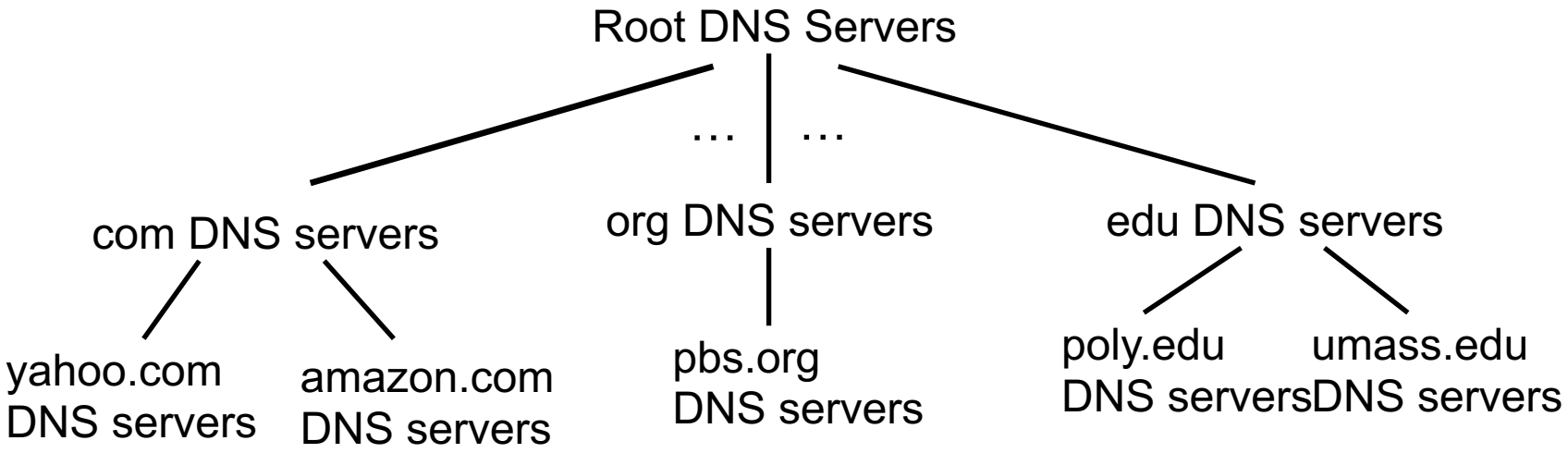
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa ?

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)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS: a distributed, hierarchical database



client wants IP 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

Local DNS name server

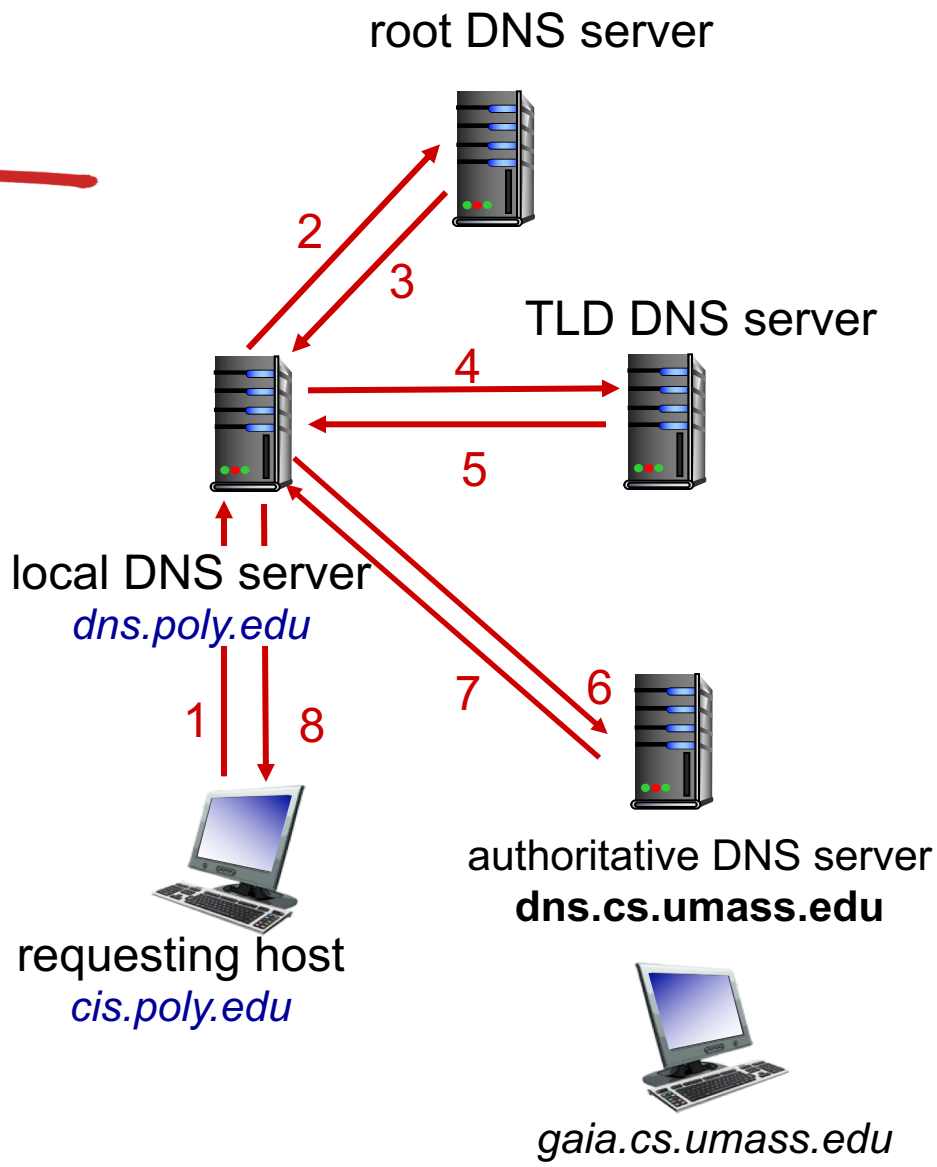
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

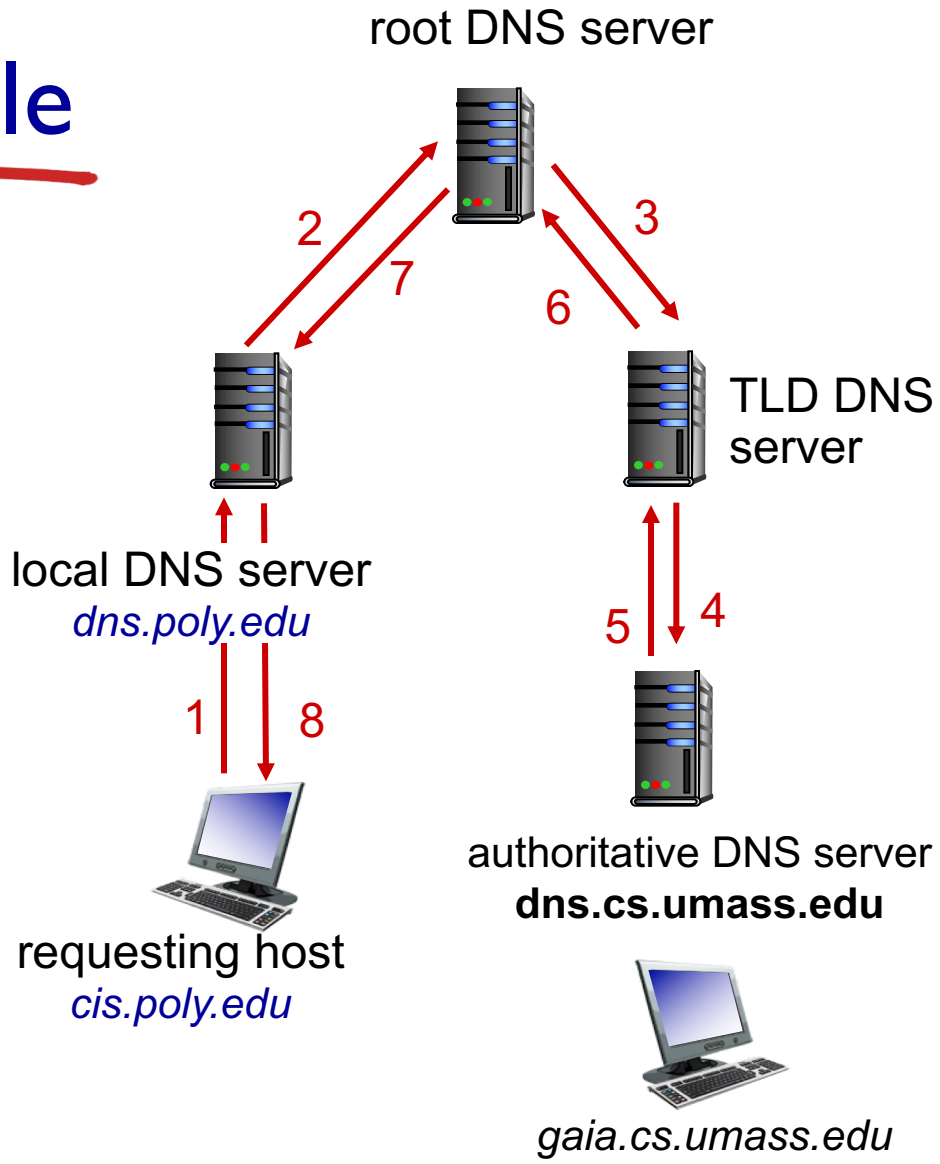
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



DNS records

DNS: distributed database 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 hostname of authoritative name server for this domain

type=CNAME

- **`name`** is alias name for some "canonical" (the real) name
- **`www.ibm.com`** is really **`servereast.backup2.ibm.com`**
- **`value`** is canonical name

Homework Question

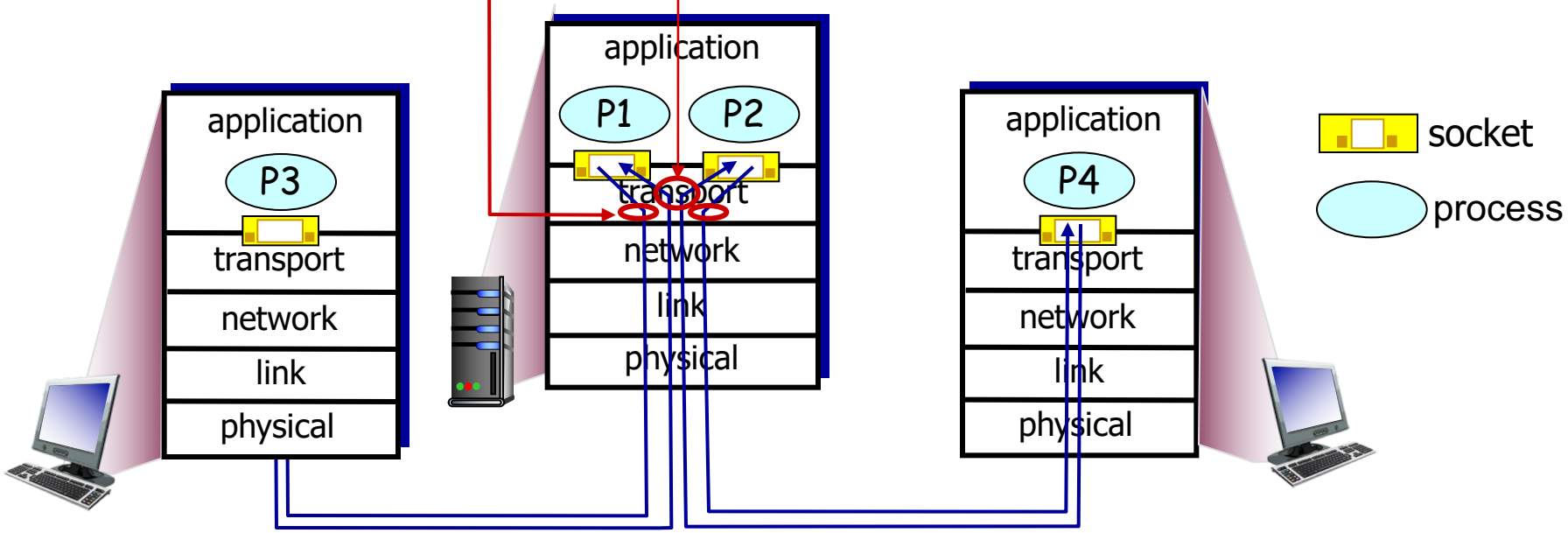
Q: Is it possible for an organization's Web server and mail server to have exactly the same alias for a host name (e.g., foo.com)? What would be the types for the RRs that contain the hostnames of the web and the mail servers?

A: Yes, an organization can have the same alias name for both its Web server and its mail server. An MX resource record type contains the host name of the mail server and a RR resource record type contains the host name of the web server.

Multiplexing/demultiplexing

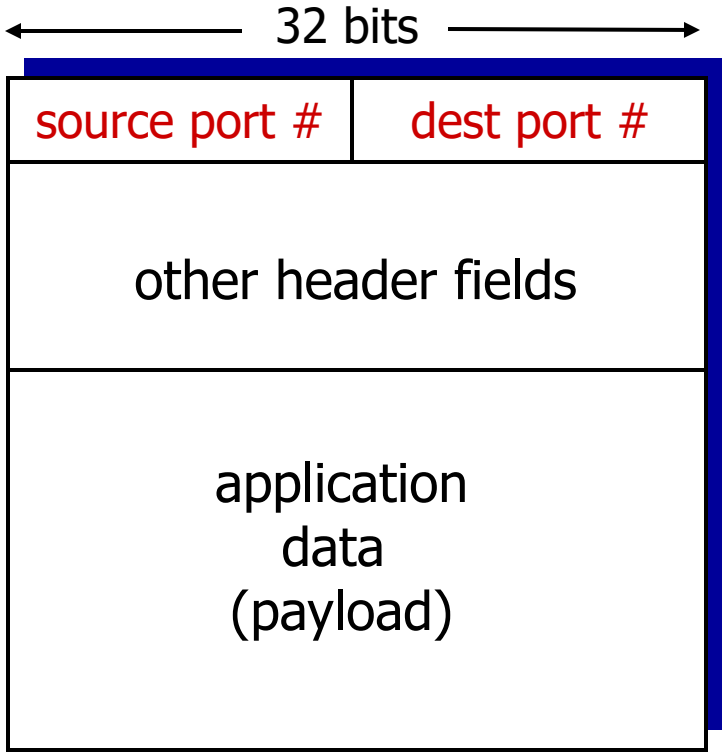
multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket



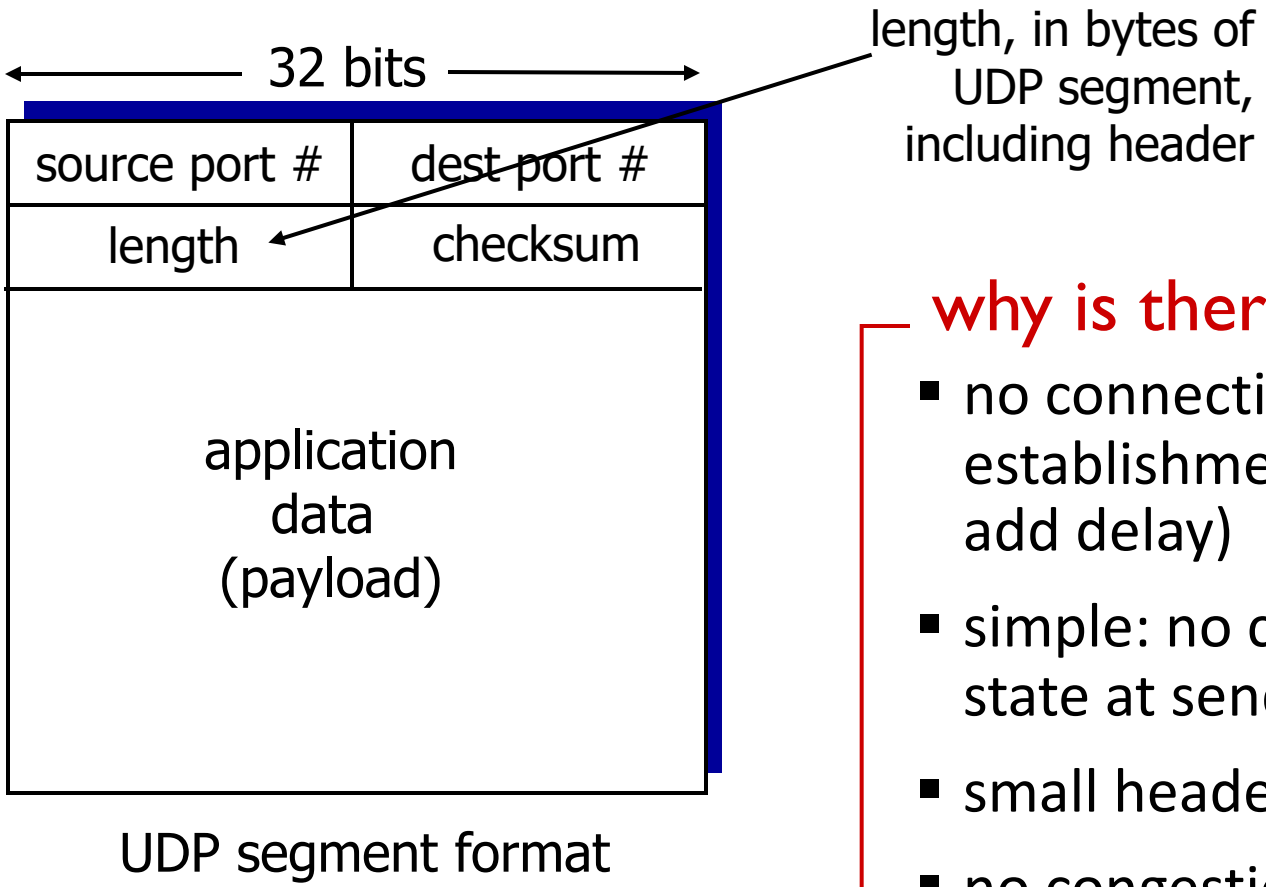
How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

UDP: segment header



UDP segment format

why is there a UDP?

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

UDP checksum

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

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one'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 nonetheless?* More later ….

Internet checksum: example

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	(1)	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Homework Question

Q: Suppose you have the following 8-bit bytes: 01010011 01100110 01110100

a. What is the 1s complement of the sum of these 8-bit bytes?

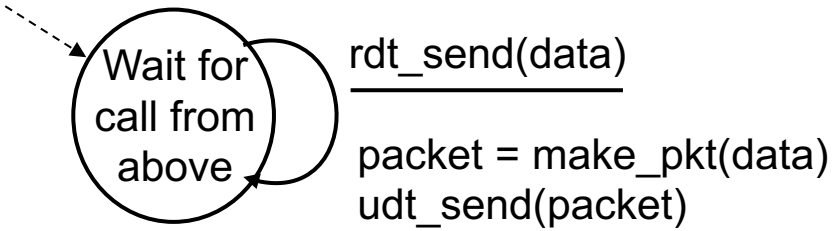
	0	1	0	1	0	0	1	1
	0	1	1	0	0	1	1	0
	1	0	1	1	1	0	0	1
	0	1	1	1	0	1	0	0
wraparound	(1)	0	0	1	0	1	1	0	1
sum	0	0	1	0	1	1	1	0
checksum	1	1	0	1	0	0	0	1

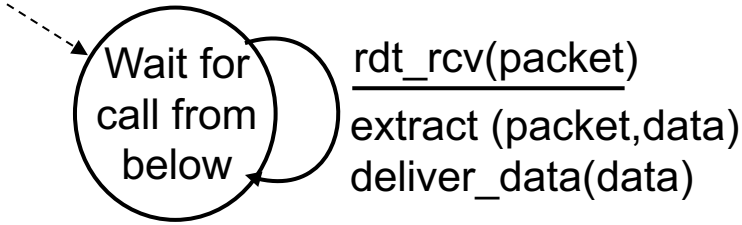
Homework Question

Q: Consider two network entities, A and B, which are connected by a perfect bidirectional channel (i.e., any message sent will be received correctly; the channel will not corrupt, lose, or re-order packets). A and B are to deliver data messages to each other in an alternating manner: First, A must deliver a message to B, then B must deliver a message to A, then A must deliver a message to B and so on. If an entity is in a state where it should not attempt to deliver a message to the other side, and there is an event like `rdt_send(data)` call from above that attempts to pass data down for transmission to the other side, this call from above can simply be ignored with a call to `rdt_unable_to_send(data)`, which informs the higher layer that it is currently not able to send data

Homework Question

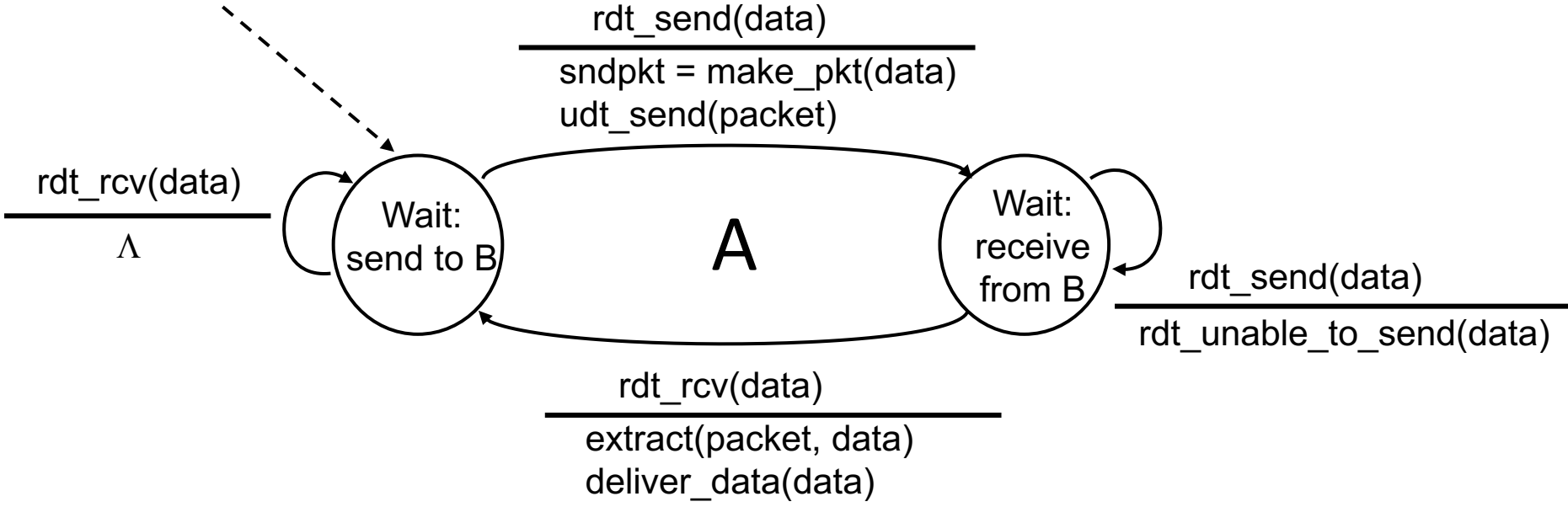
A: Take a look at rdt 1.0 first



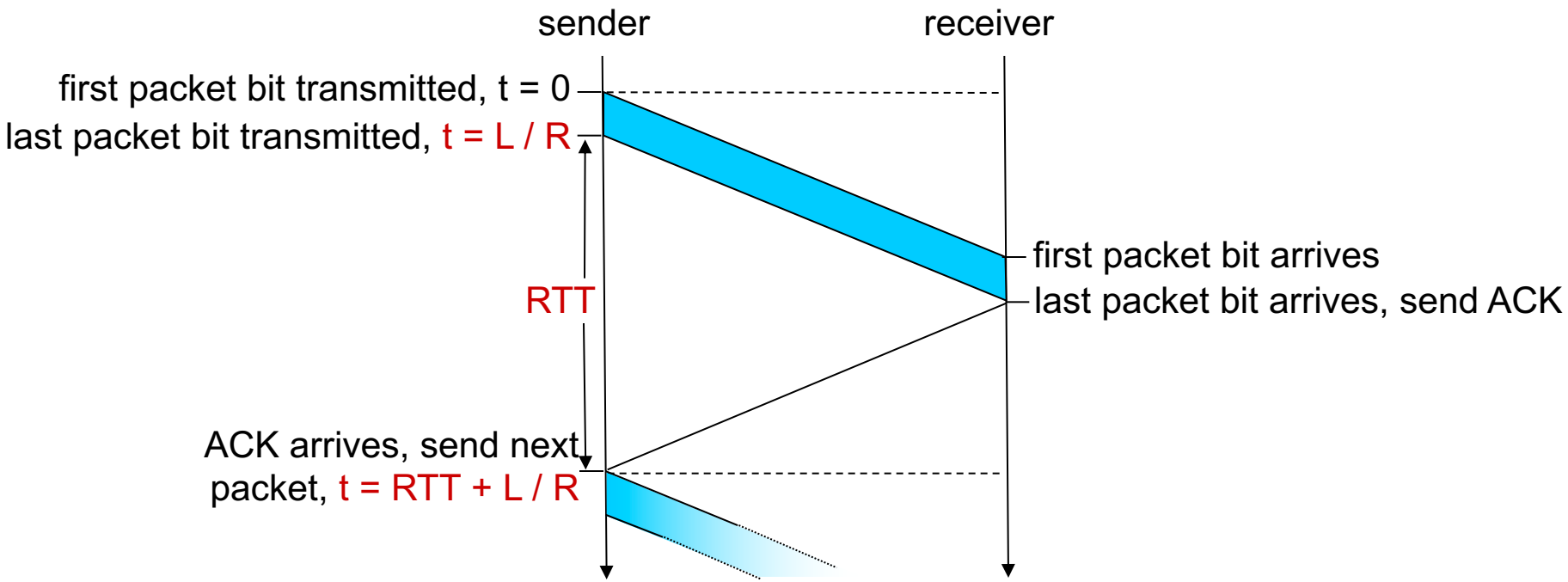


sender receiver

The sender could send as much data as needed; so does the receiver



rdt3.0: stop-and-wait operation

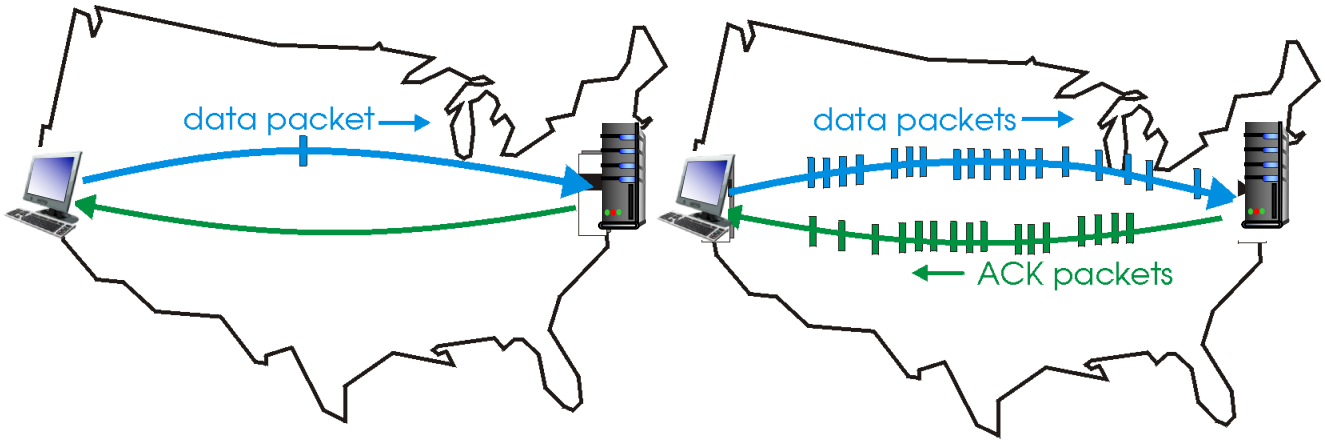


$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

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

Pipelined protocols: overview

Go-back-N:

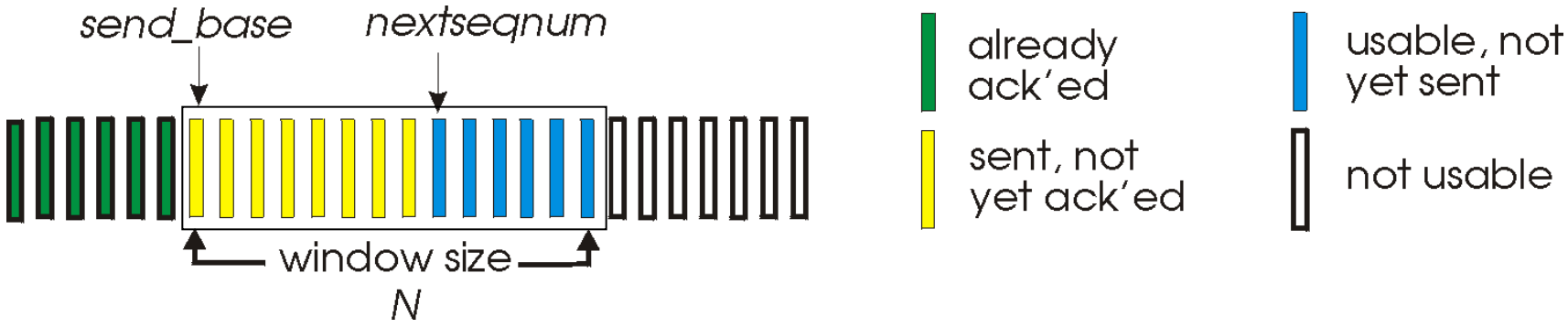
- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

Selective Repeat:

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

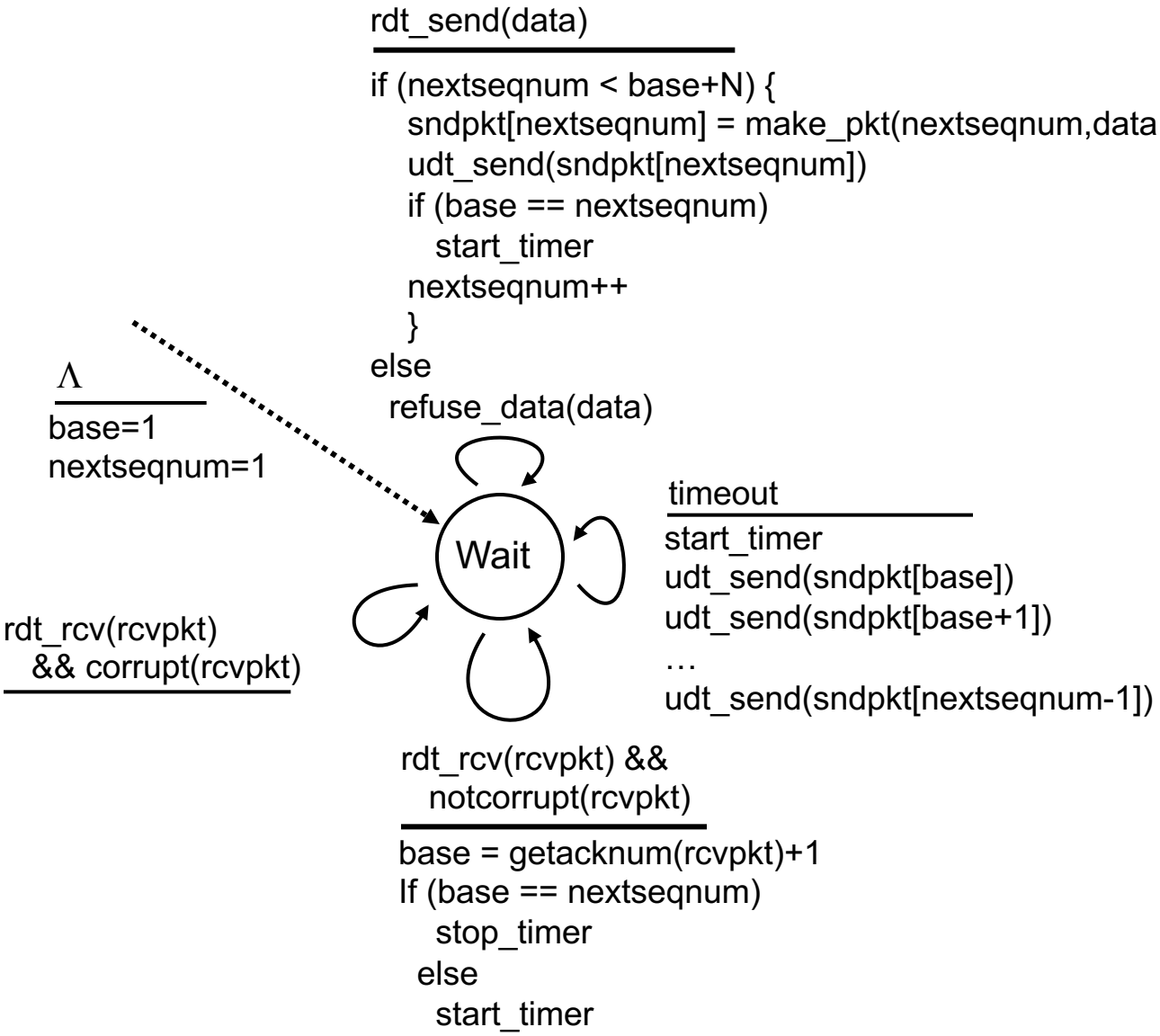
Go-Back-N: sender

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

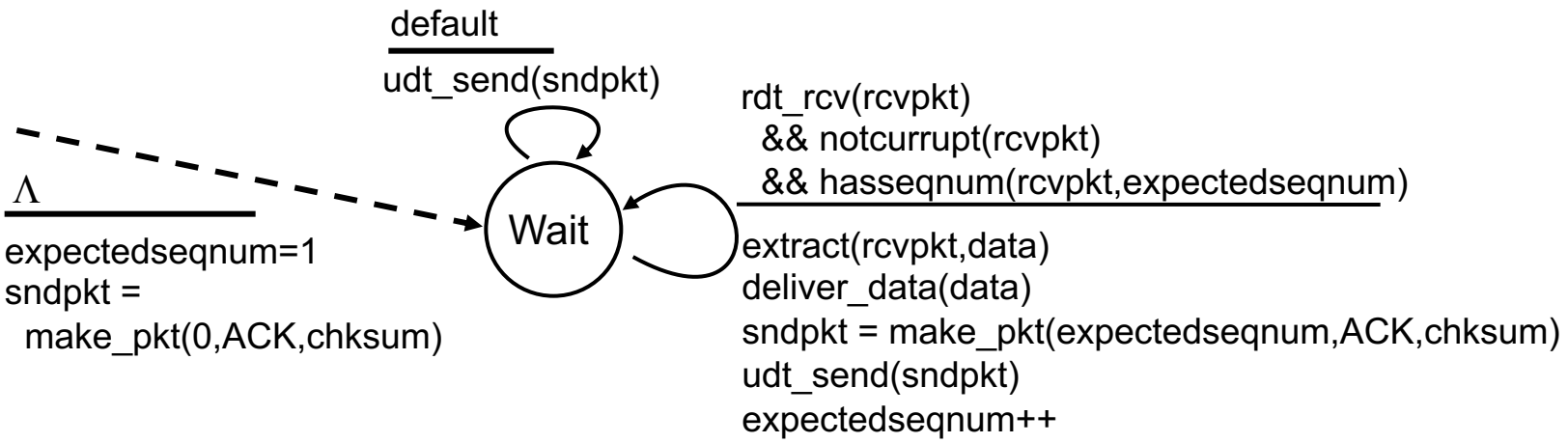


- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

GBN: sender extended FSM



GBN: receiver extended FSM

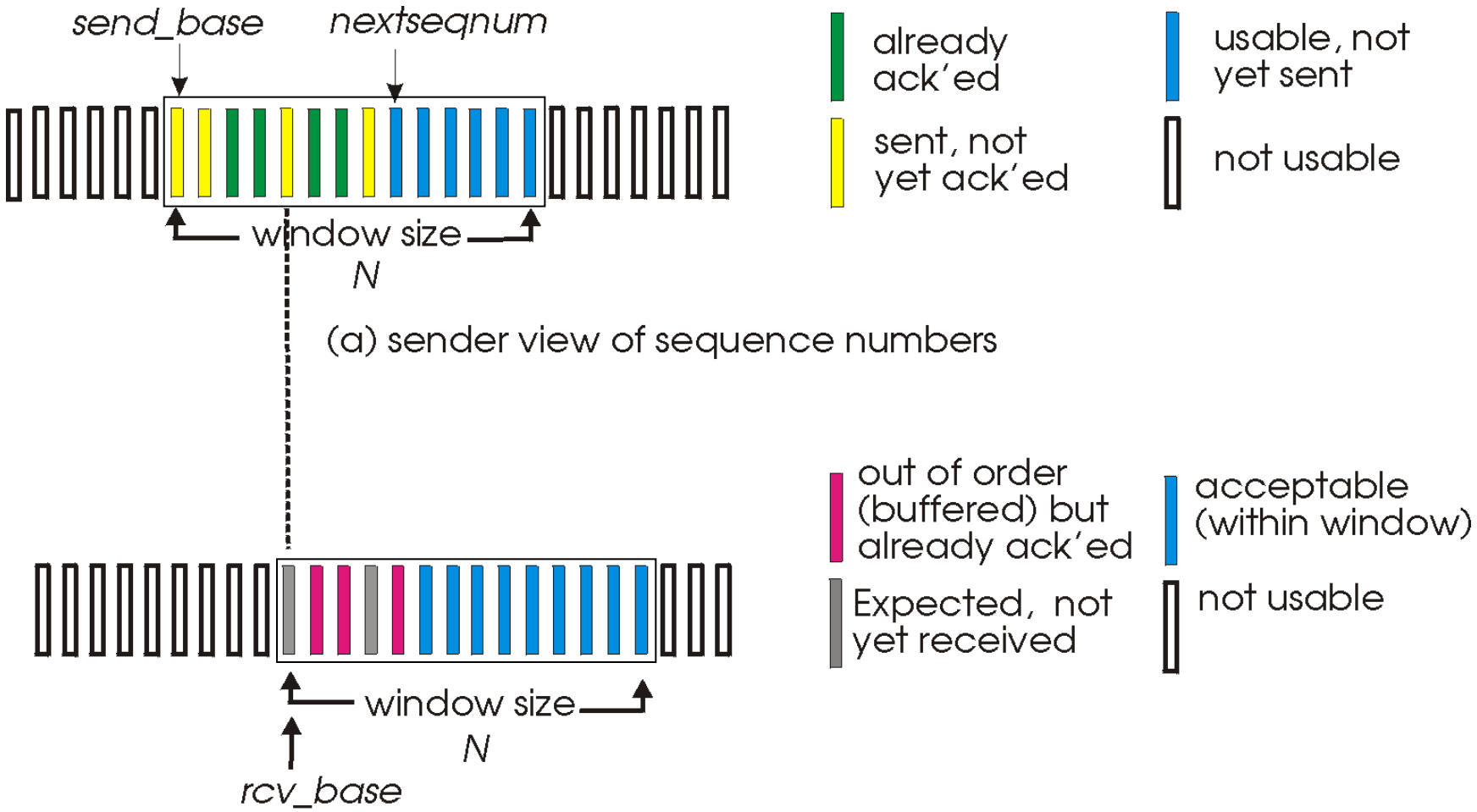


ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

- out-of-order pkt:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

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

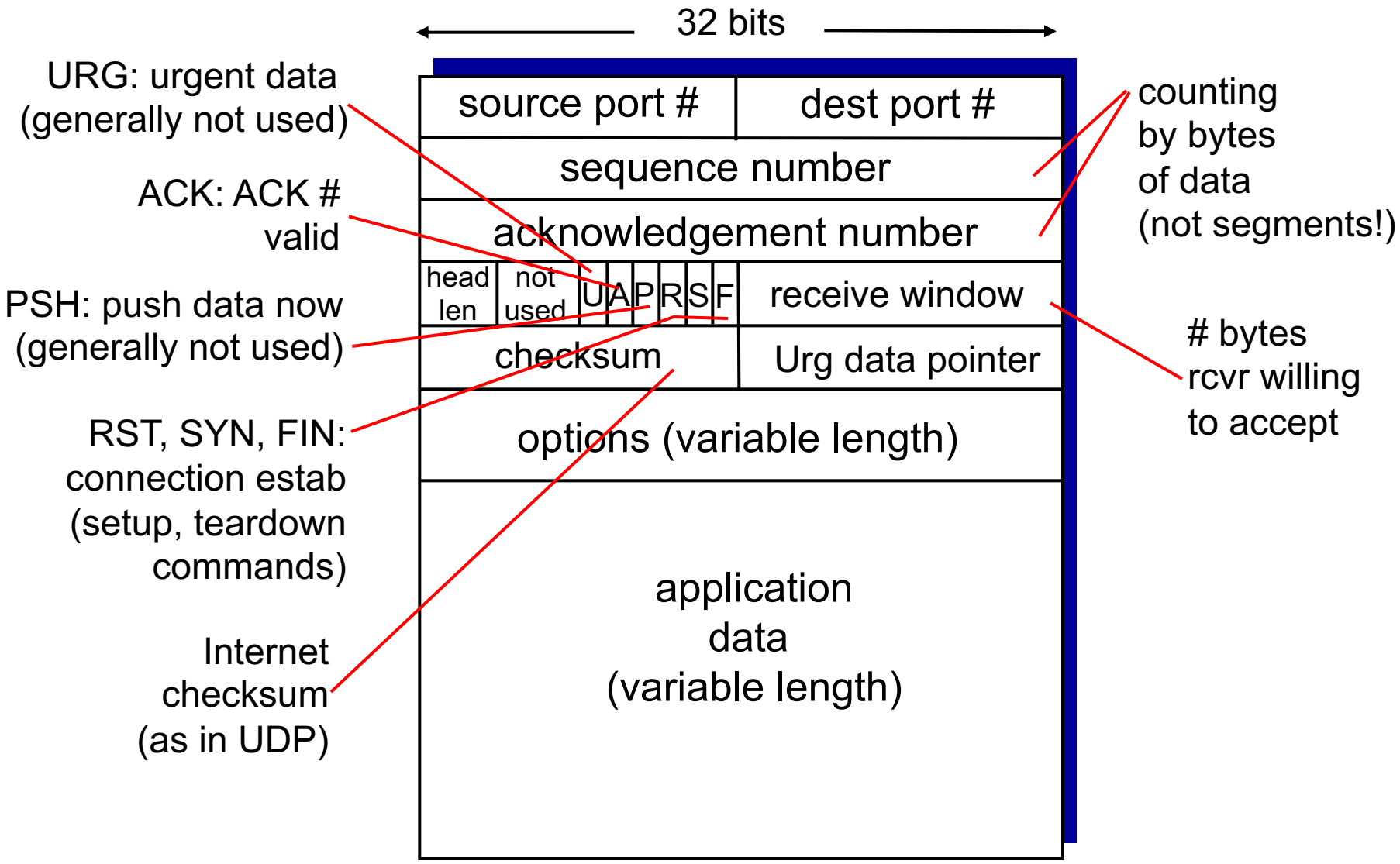
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

TCP segment structure



Homework Question

Q: Suppose TCP operates over a 1 Gbps link.

a. Assume TCP could utilize the full bandwidth continuously and the average packet size is 60 bytes, how long does it take for the TCP sequence numbers to wrap around completely?

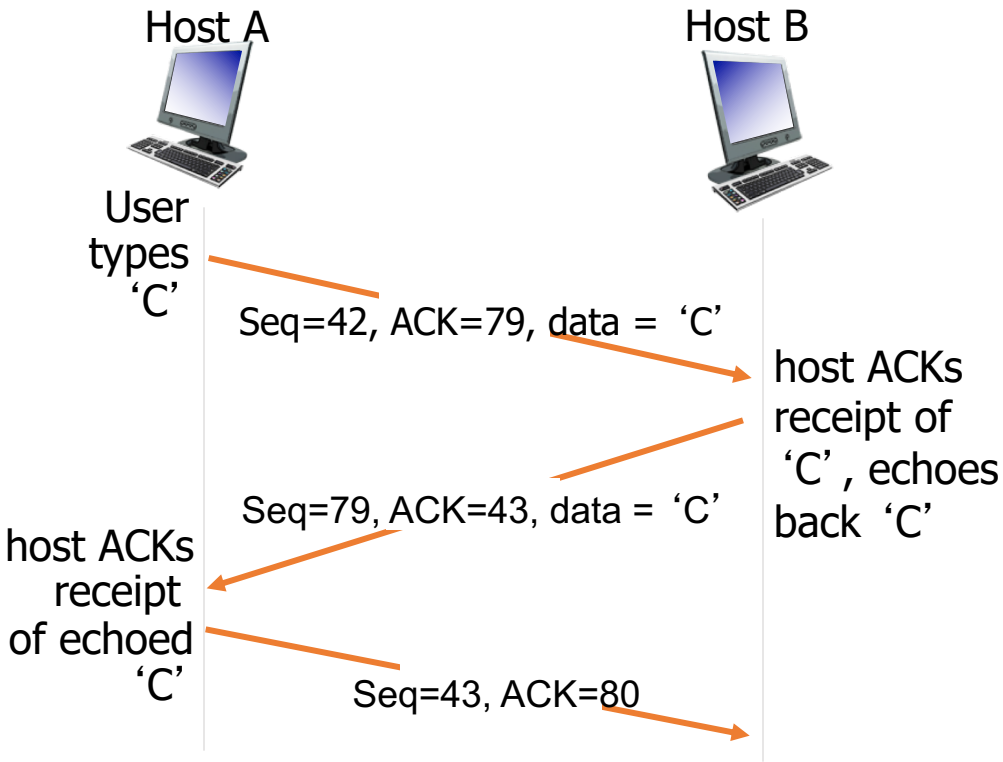
A: Total bits transmitted over the link: 2^{32} packets $\times 60 \times 8 = 2.06 \times 10^{12}$ bits

$$\text{Transmission time} = \frac{2.06 \times 10^{12}}{10^{9}} = 2061.58 \text{ secs} \approx 34 \text{ mins}$$

b. Suppose an added 32-bit timestamp field increments 1000 times during the wraparound time you found above. How long would it take for the timestamp to wrap around?

A: # of timestamps/per 34 mins = 1000. Thus, total wrap around time for timestamp $= \frac{2^{32}}{1000} \times 34 \text{ mins} \approx 111 \text{ yrs}$

TCP seq. numbers, ACKs

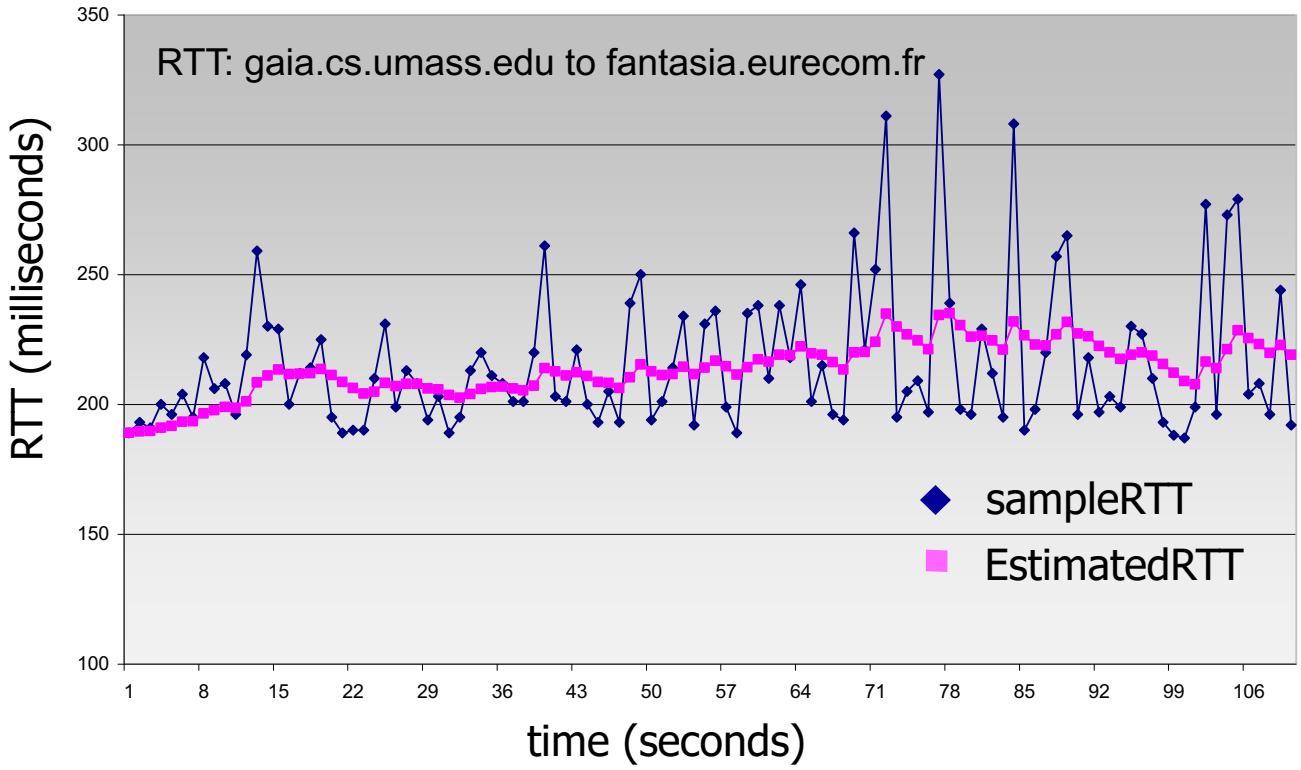


simple telnet scenario

TCP round trip time, timeout

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

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$



TCP round trip time, timeout

- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

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

(typically, β = 0.25)
```

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

estimated RTT

"safety margin"

TCP fast retransmit

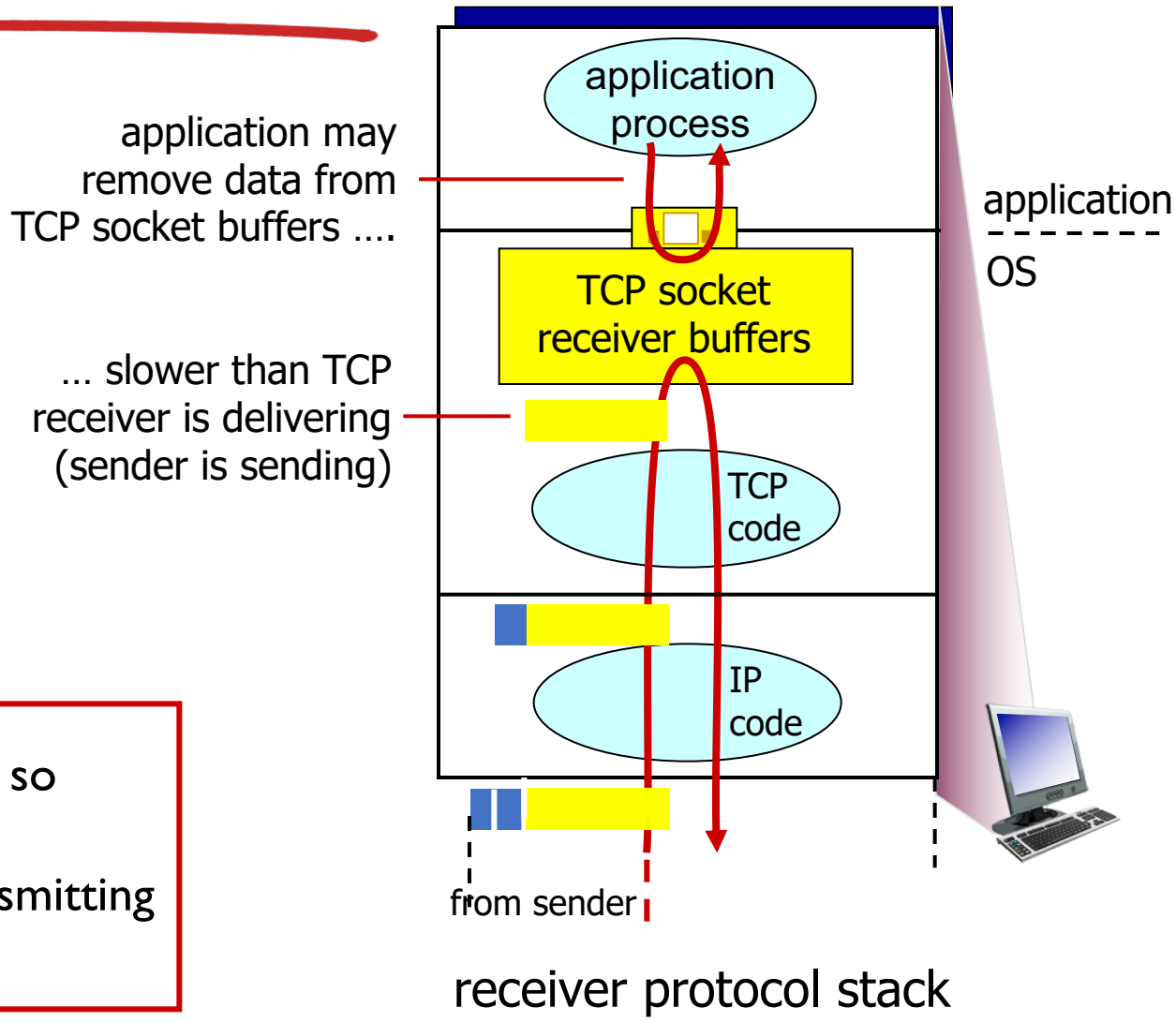
- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

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

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

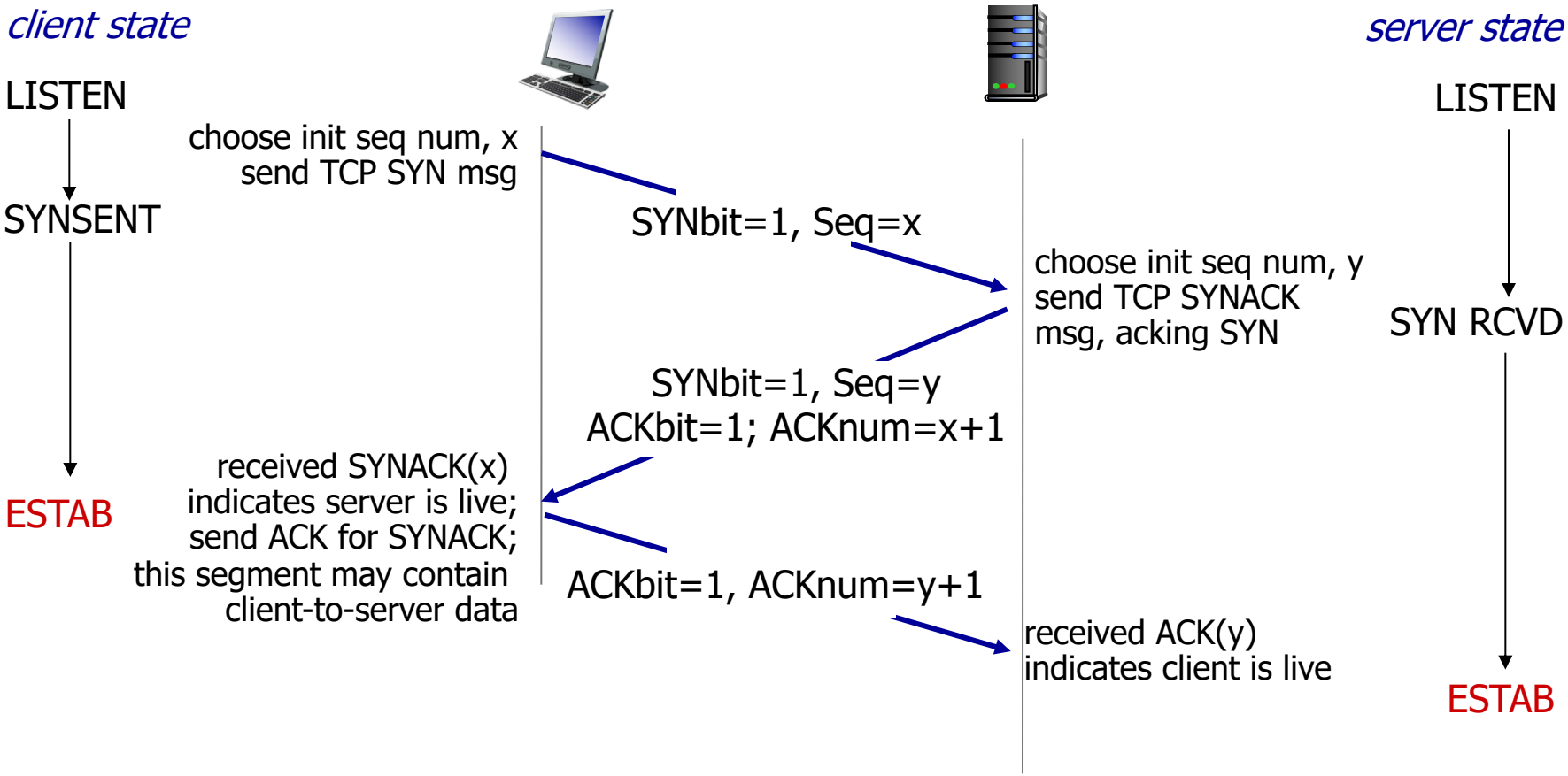
TCP flow control



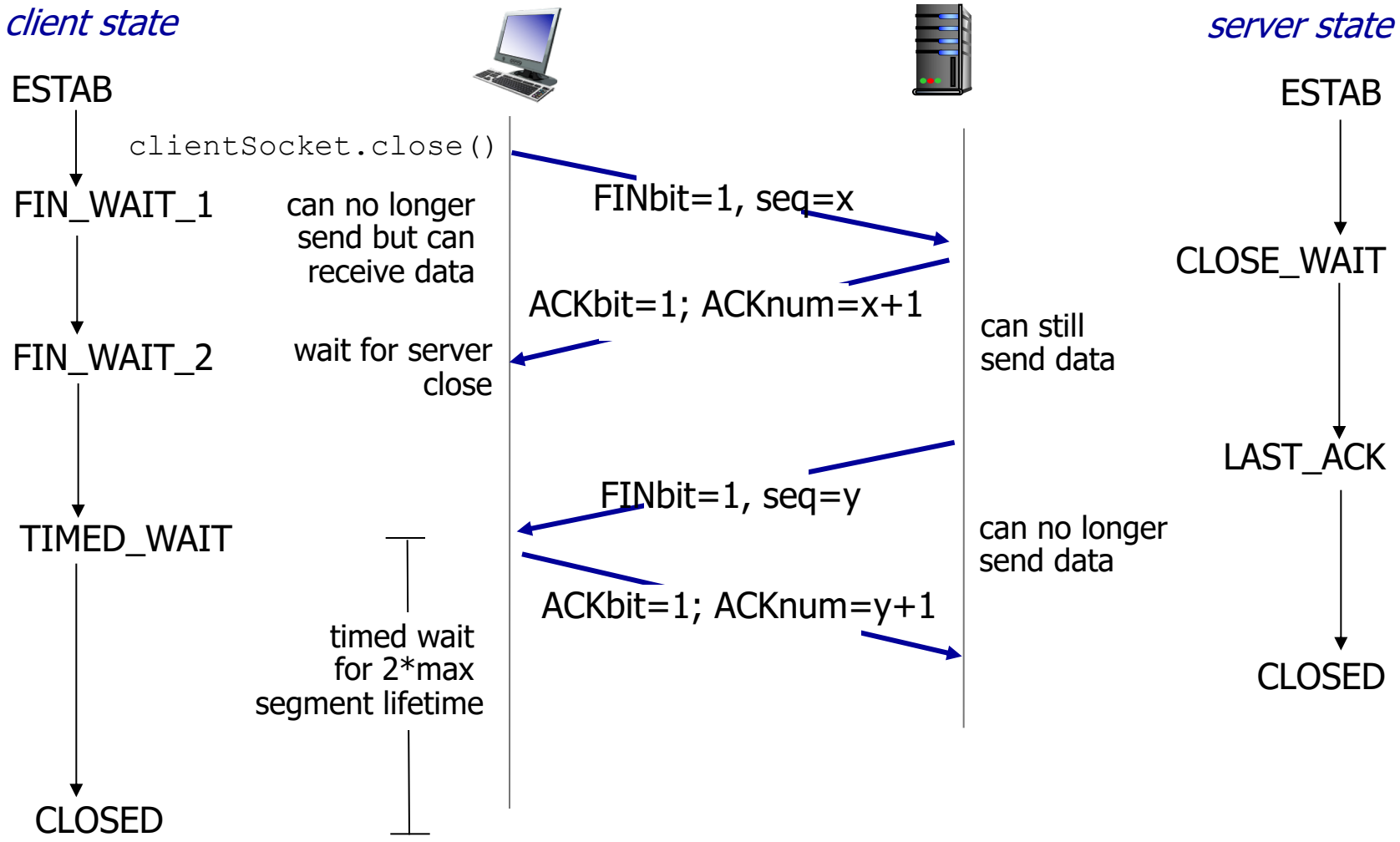
flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

TCP 3-way handshake



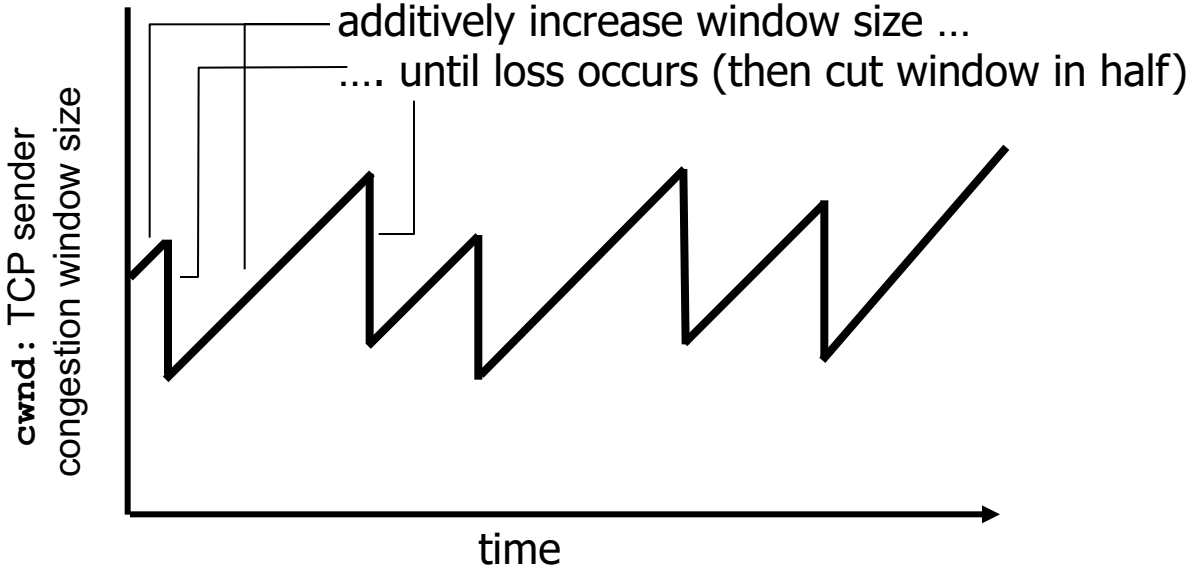
TCP: closing a connection



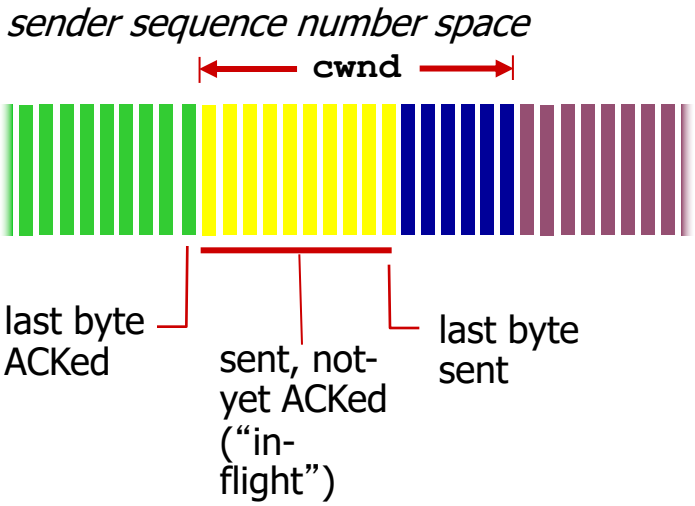
TCP congestion control: additive increase multiplicative decrease

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`cwnd`** by 1 MSS every RTT until loss detected
  - *multiplicative decrease:* cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth



TCP Congestion Control: details



- sender limits transmission:

$$\texttt{LastByteSent-LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** is dynamic, function of perceived network congestion

TCP sending rate:

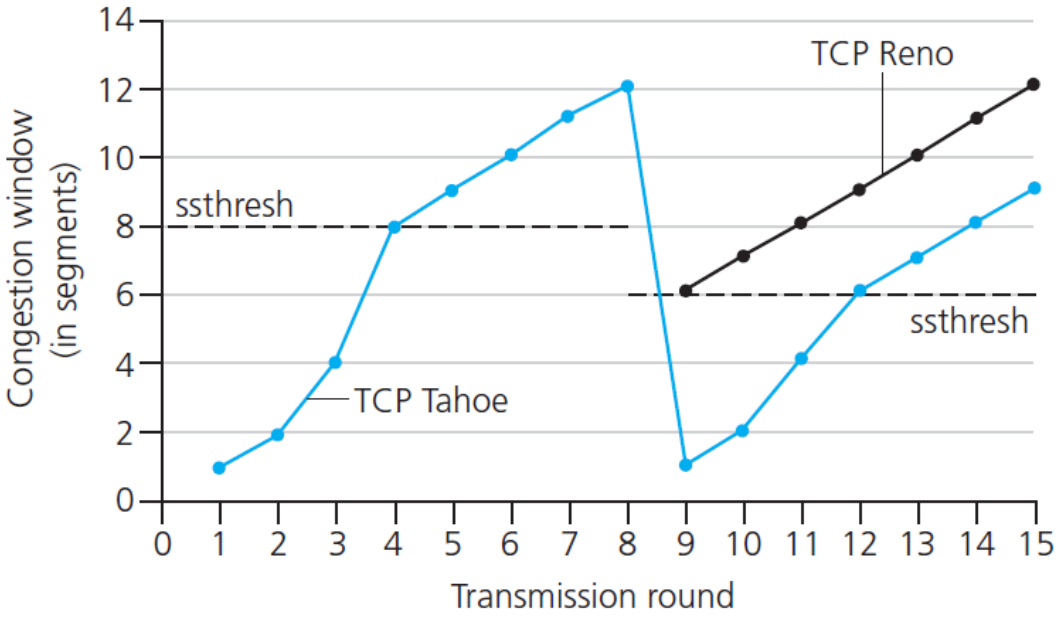
- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

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

TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

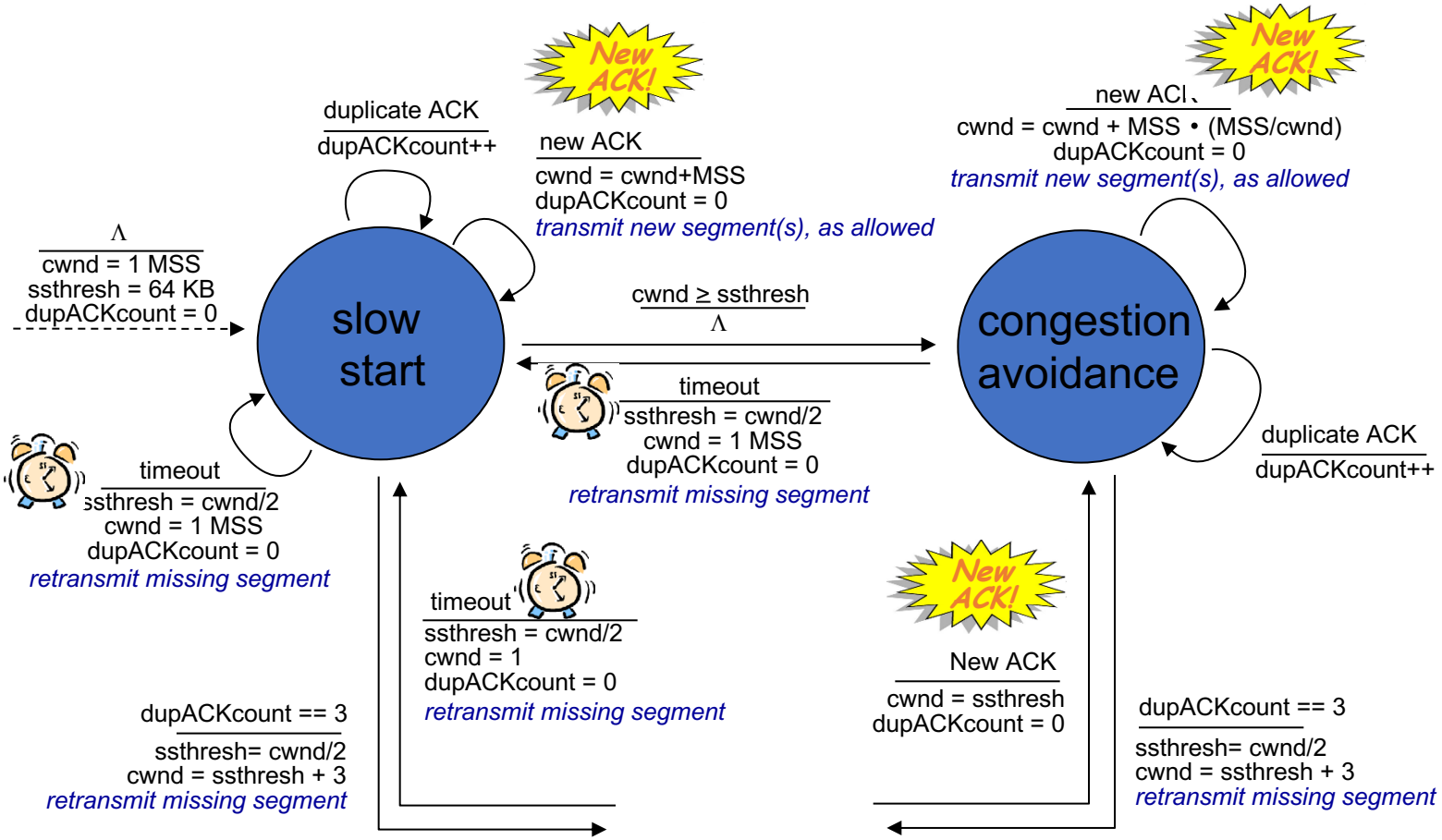
A: when **`cwnd`** gets to 1/2 of its value before timeout.



Implementation:

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

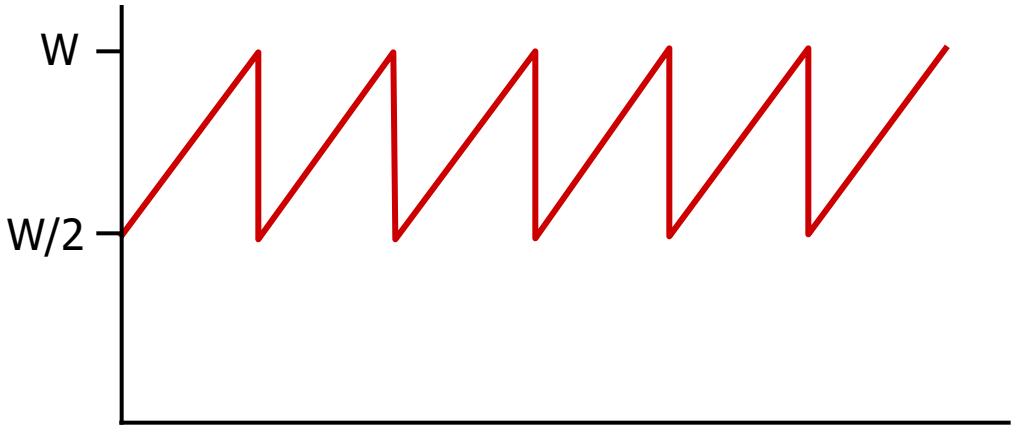
Summary: TCP Congestion Control



TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. thruput is 3/4W per RTT

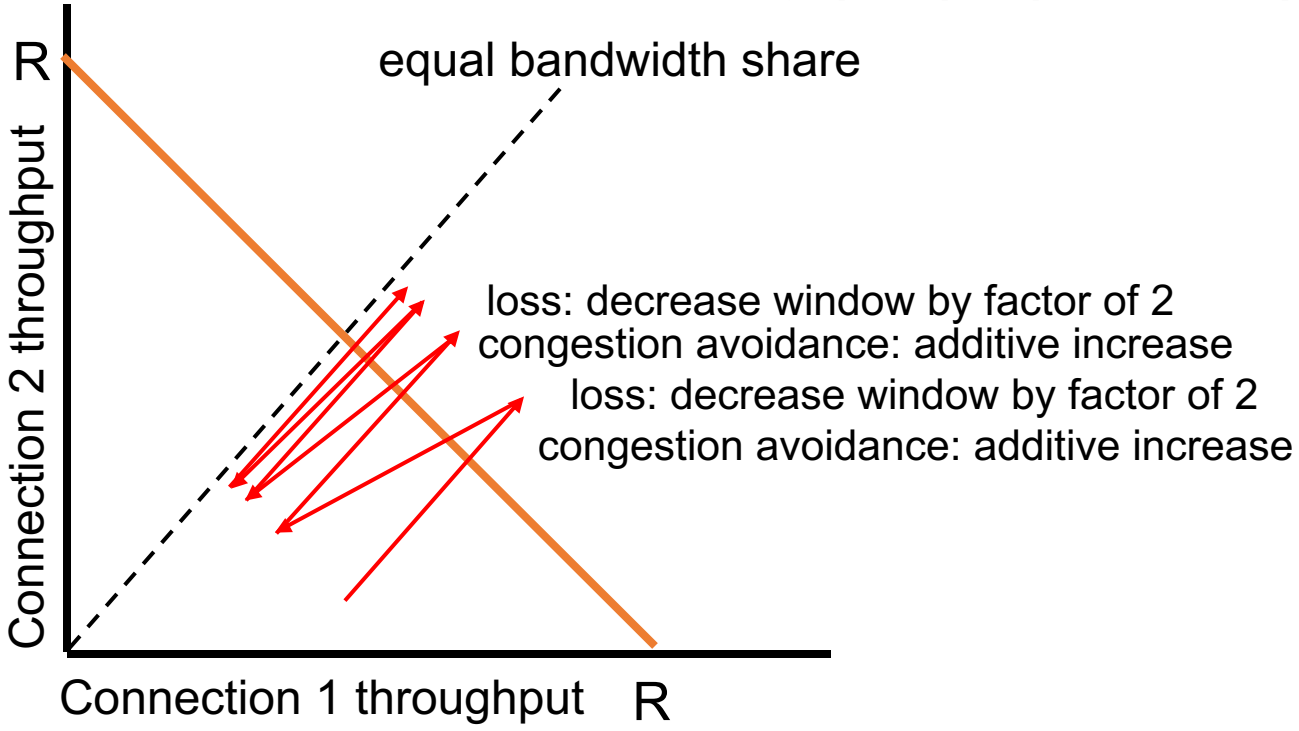
$$\text{avg TCP thruput} = \frac{3}{4}\,\frac{W}{RTT}\ \text{bytes/sec}$$



Why is TCP fair?

two competing sessions:

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



Two key network-layer functions

network-layer functions:

- *forwarding:* move packets from router's input to appropriate router output
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

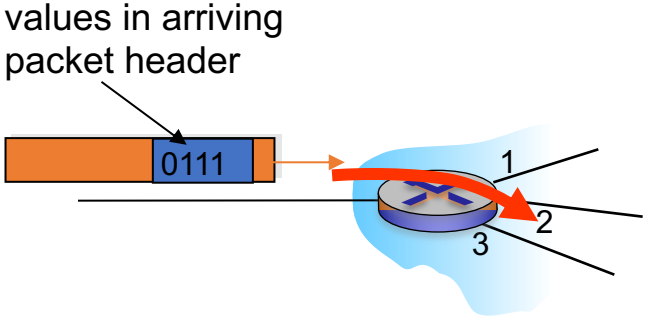
analogy: taking a trip

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

Network layer: data plane, control plane

Data plane

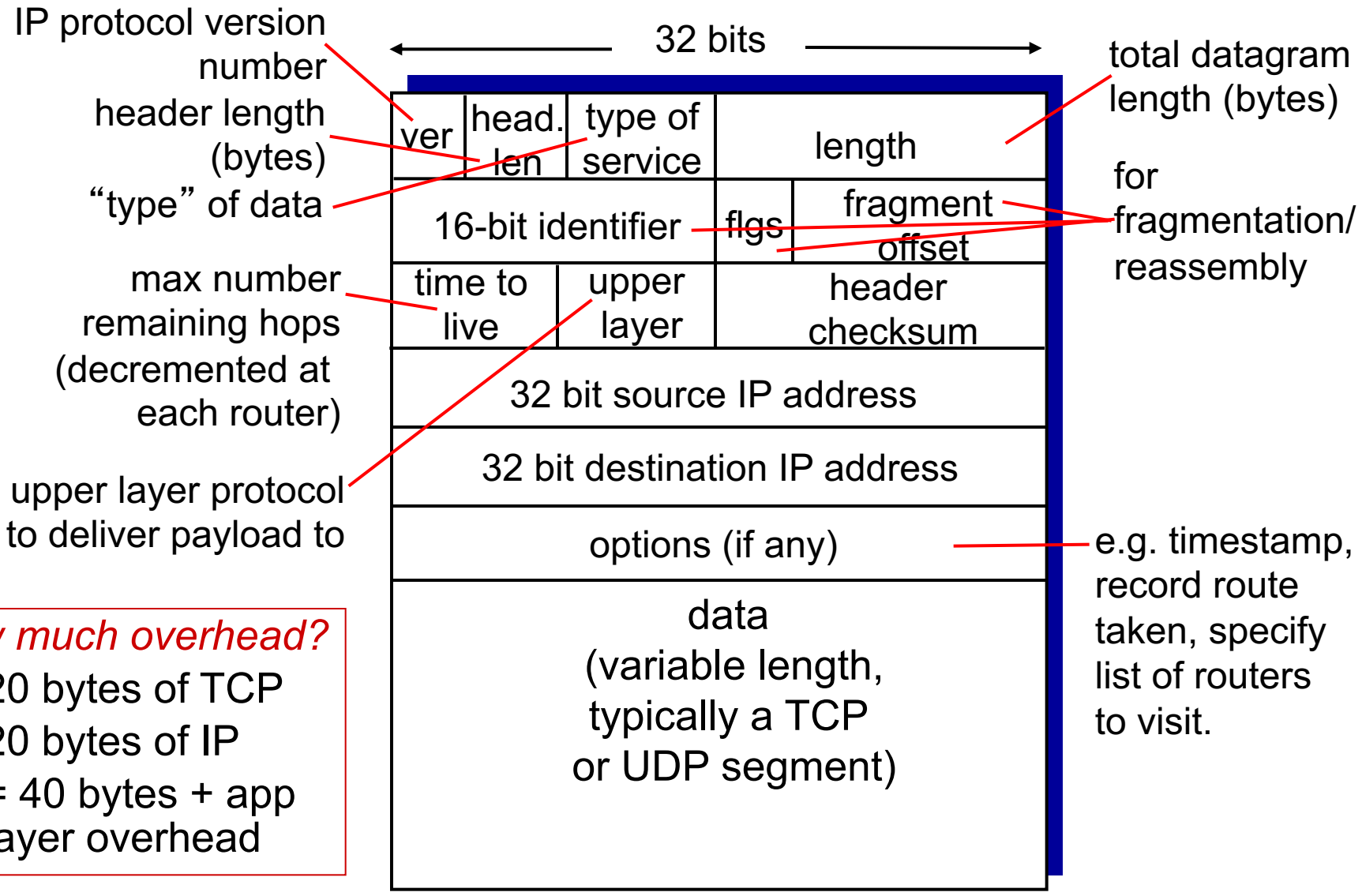
- local, per-router function
- determines how datagram arriving on router input port is forwarded to router output port
- forwarding function



Control plane

- network-wide logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

IP datagram format



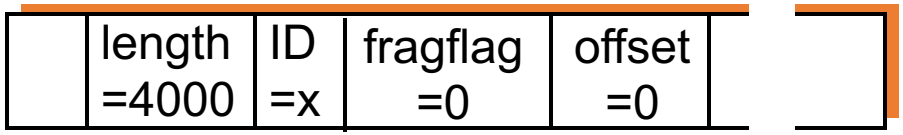
how much overhead?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

IP fragmentation, reassembly

example:

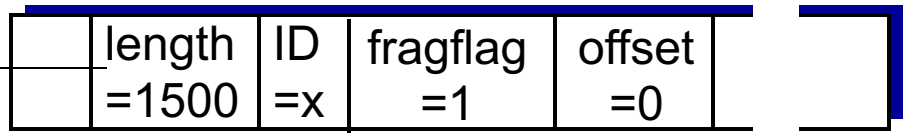
- 4000 byte datagram
- MTU = 1500 bytes

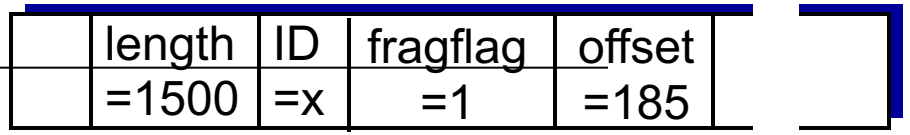


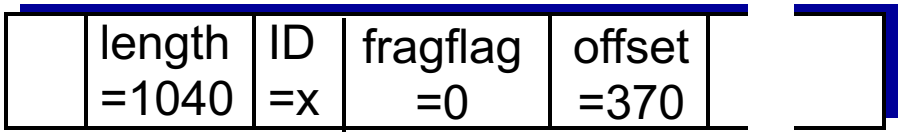
one large datagram becomes several smaller datagrams

1480 bytes in data field

offset = 1480/8

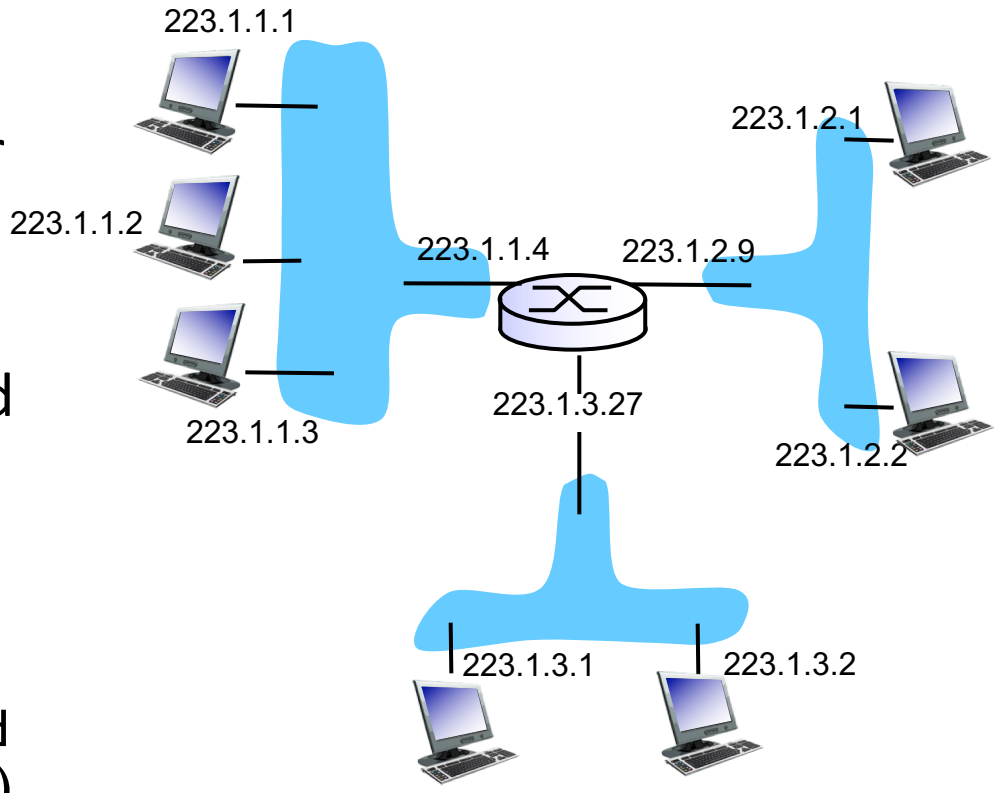






IP addressing: introduction

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

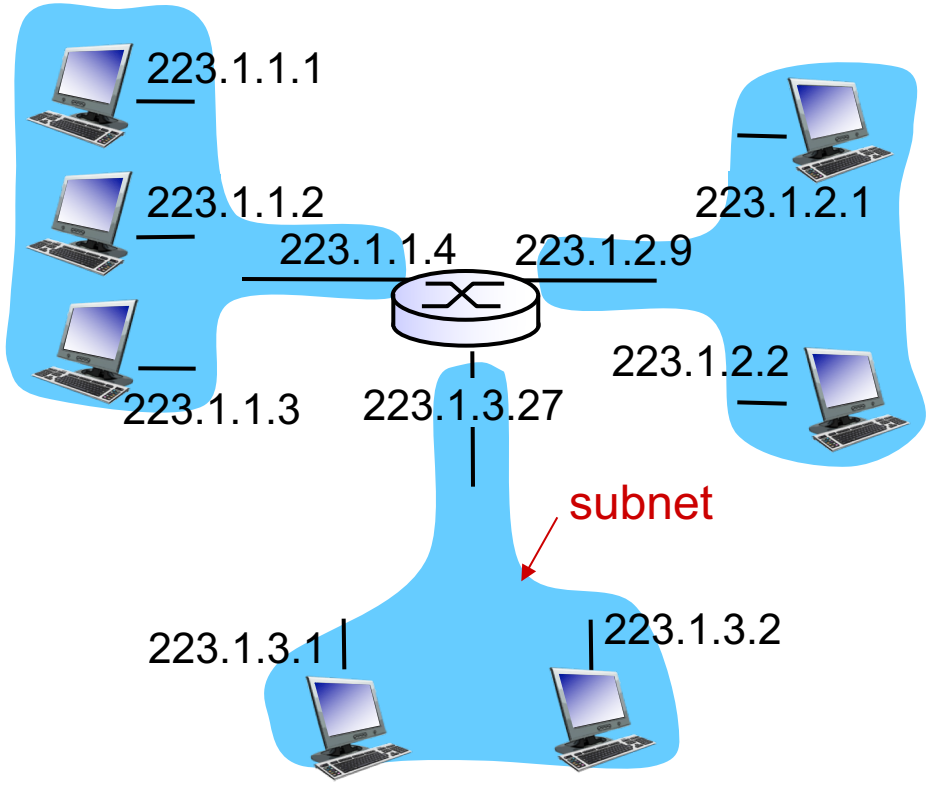


223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

Subnets

- IP address:
  - subnet part - high order bits
  - host part - low order bits
- *what's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without intervening router*



network consisting of 3 subnets

IP addressing: CIDR

CIDR: Classless InterDomain Routing

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

200.23.16.0/23

DHCP: Dynamic Host Configuration Protocol

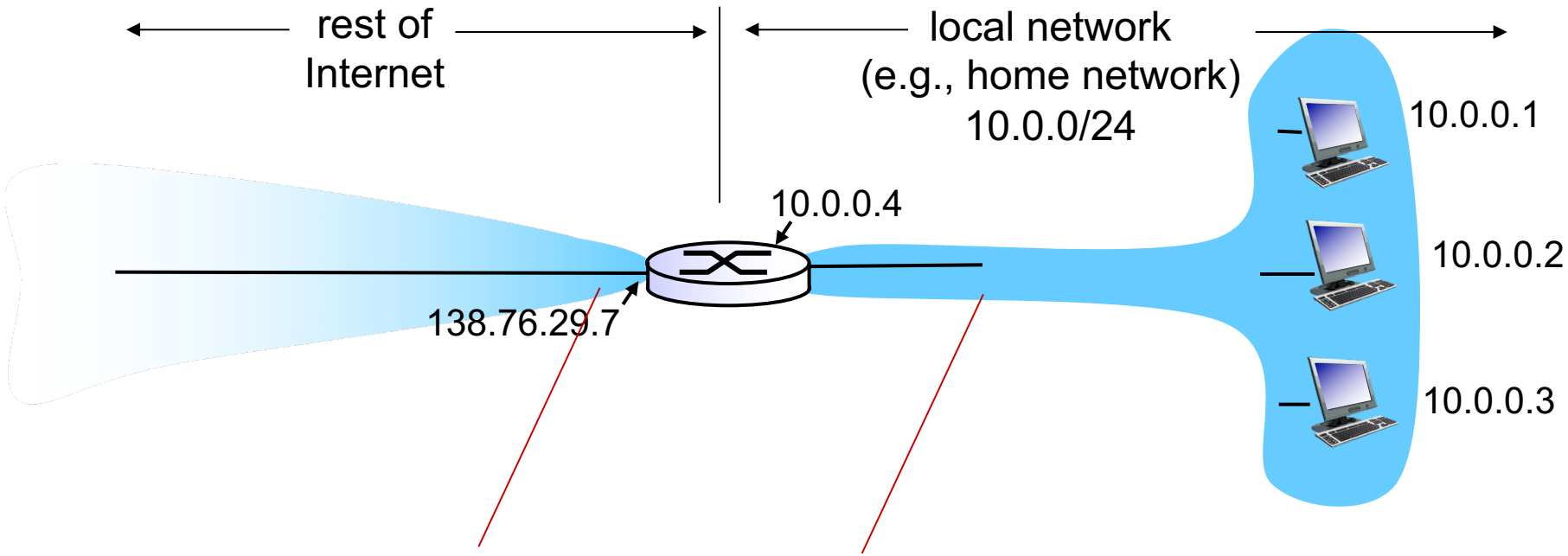
goal: allow host to *dynamically* obtain its IP address from network server when it joins network

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

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

NAT: network address translation



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

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

IPv6: motivation

- *initial motivation:* 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:

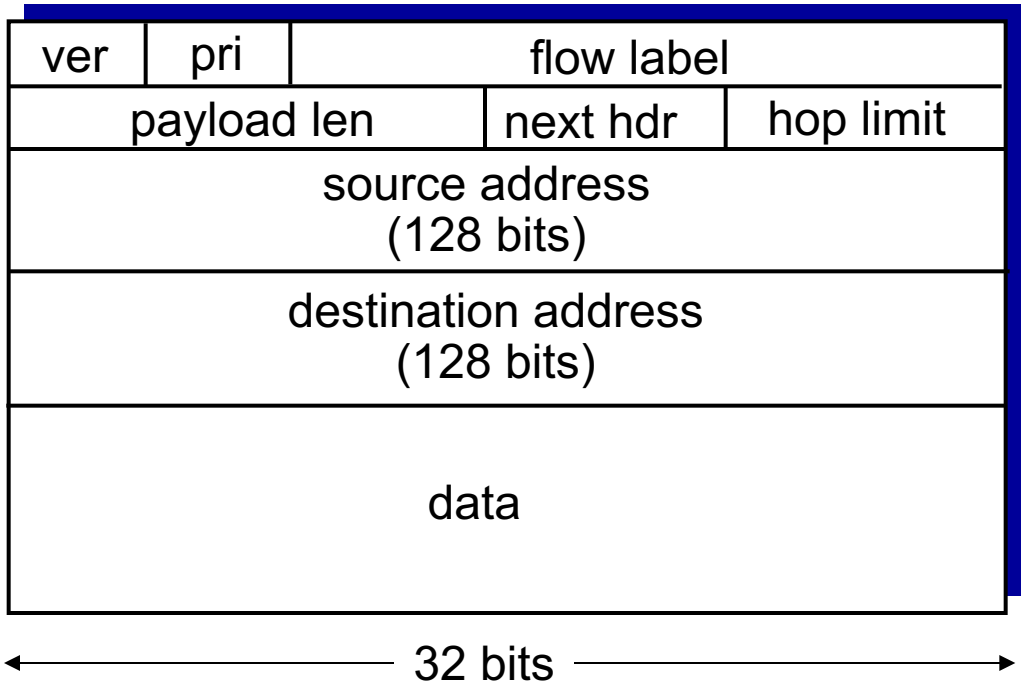
- fixed-length 40 byte header
- no fragmentation allowed

IPv6 datagram format

priority: identify priority among datagrams in flow

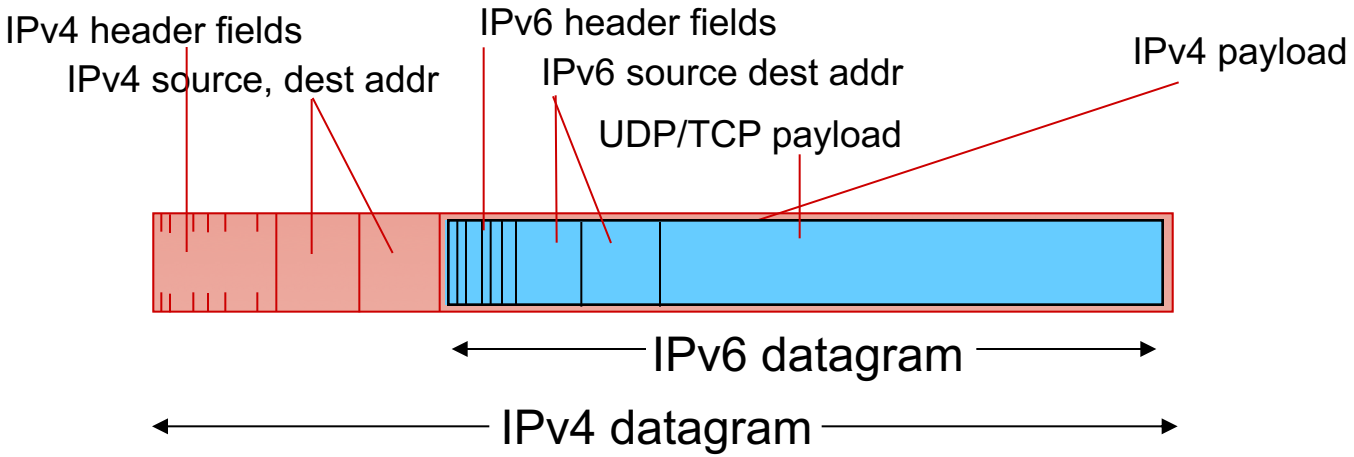
flow Label: identify datagrams in same "flow."
(concept of"flow" not well defined).

next header: identify upper layer protocol for data

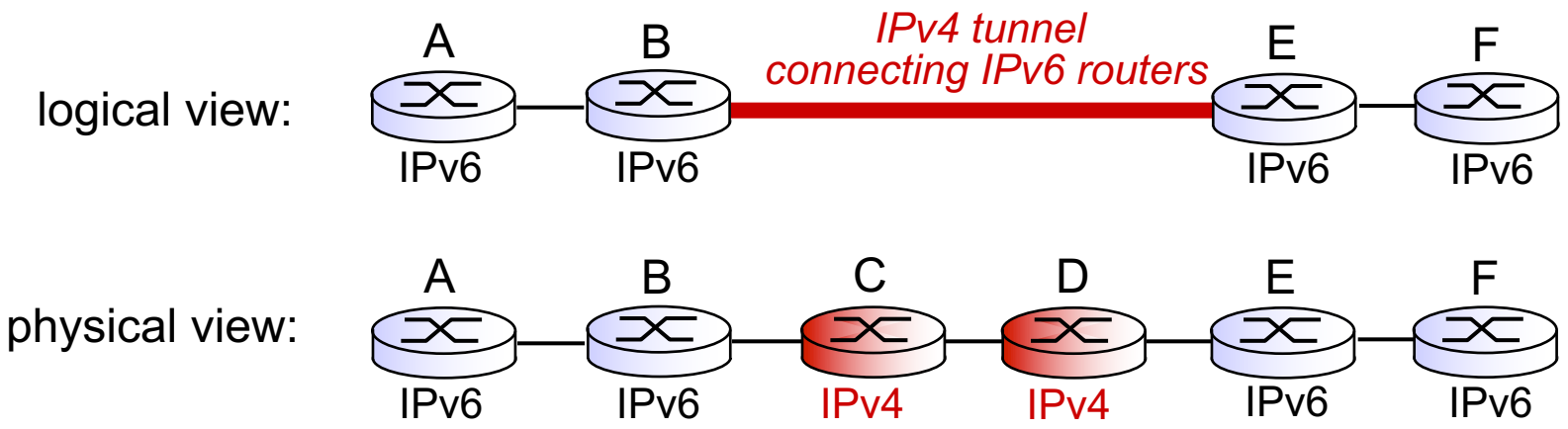


Transition from IPv4 to IPv6

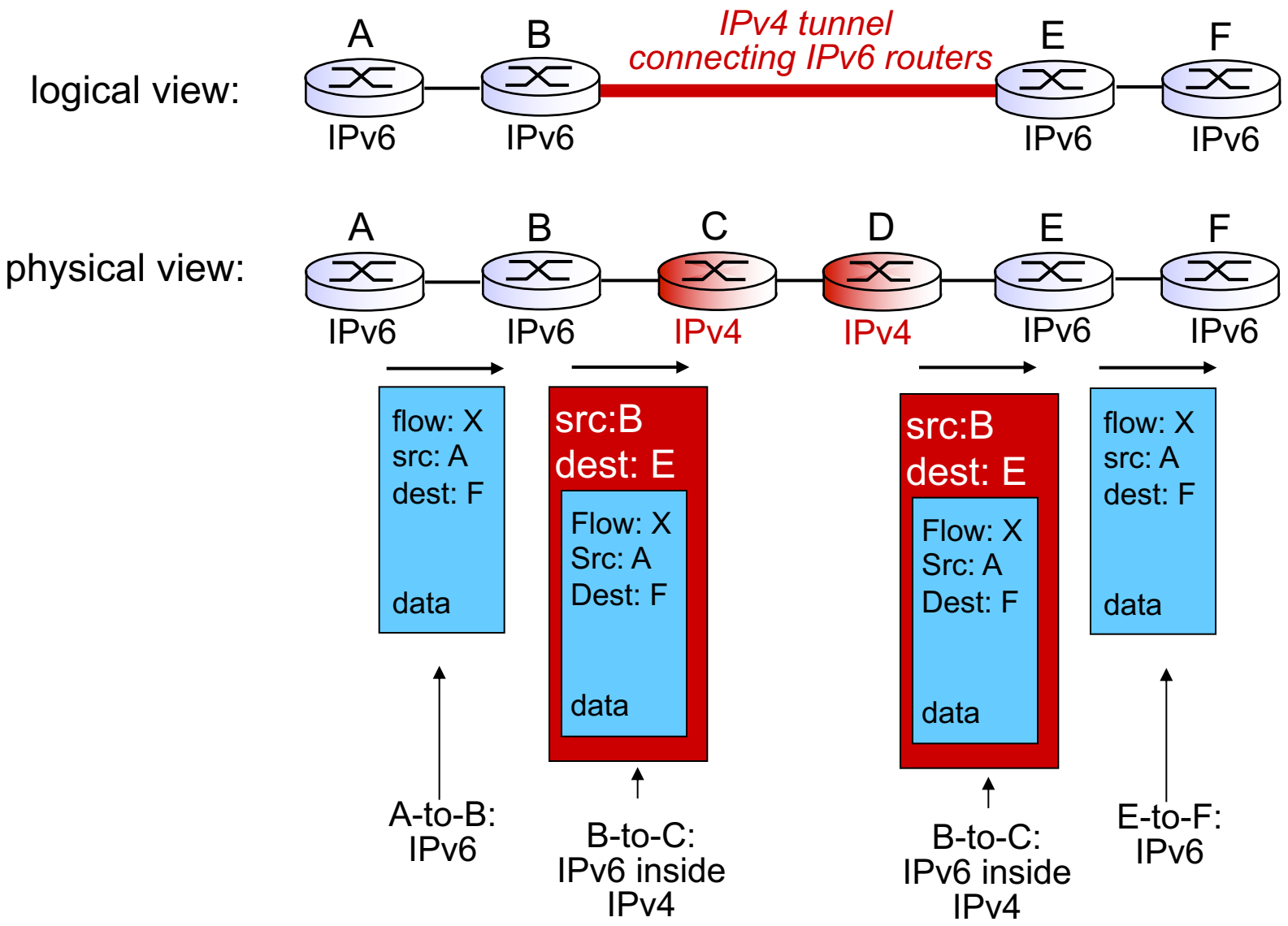
- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- *tunneling:* IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



Tunneling

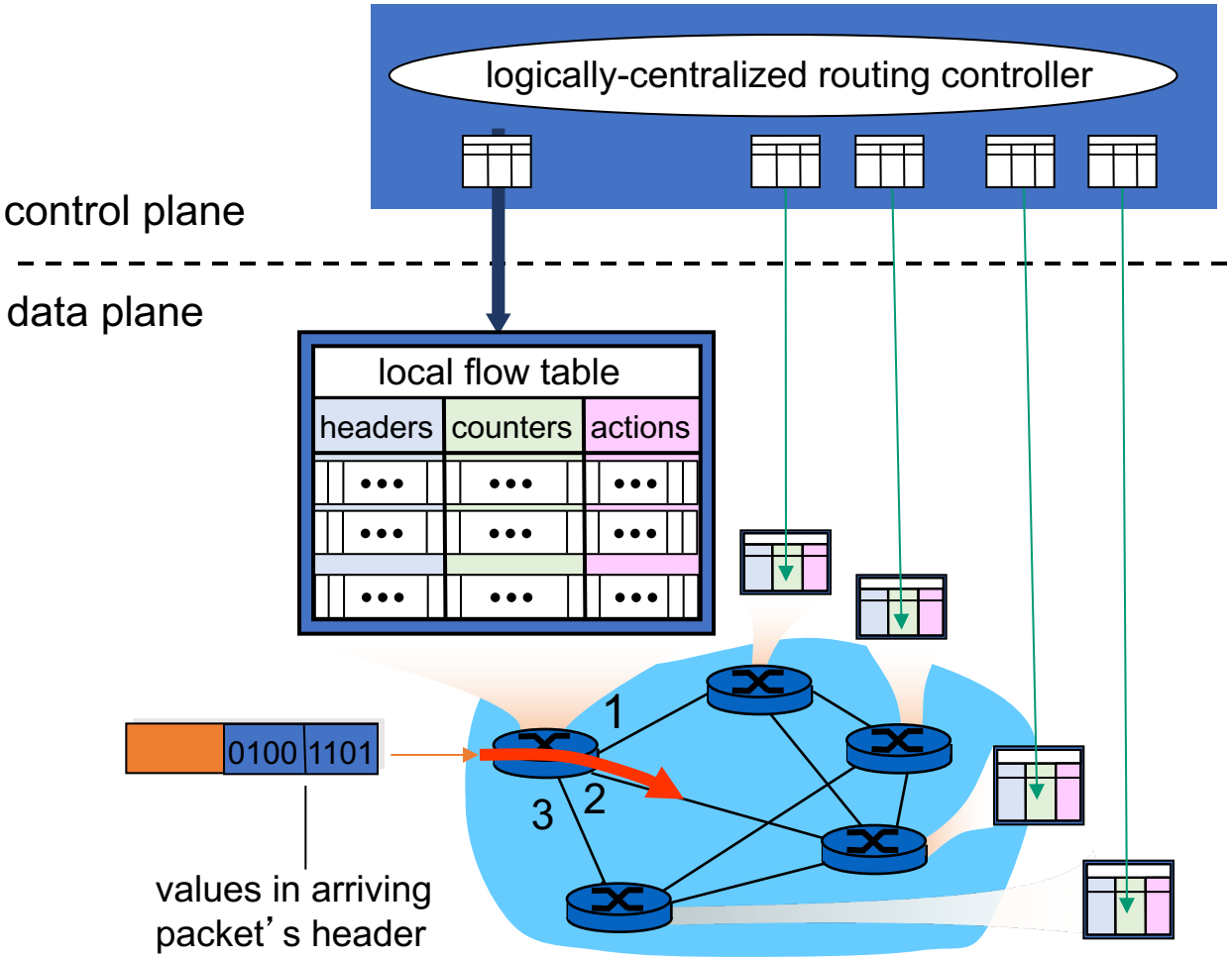

logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6

Tunneling


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6
flow: X
src: A
dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
flow: X
src: A
dest: F
data
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6

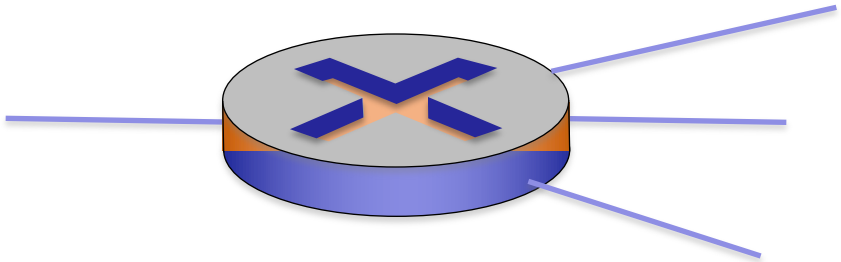
Generalized Forwarding and SDN

Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller



OpenFlow data plane abstraction

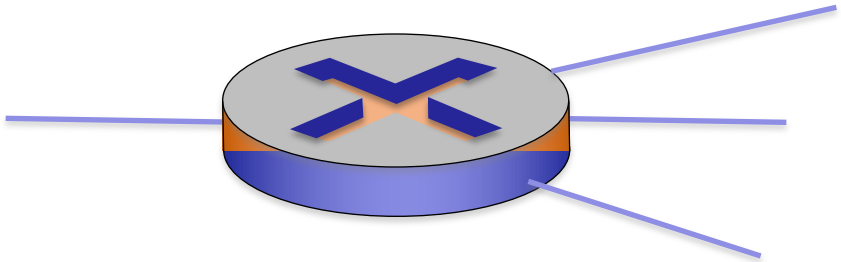
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets



Flow table in a router (computed and distributed by controller) define router's match+action rules

OpenFlow data plane abstraction

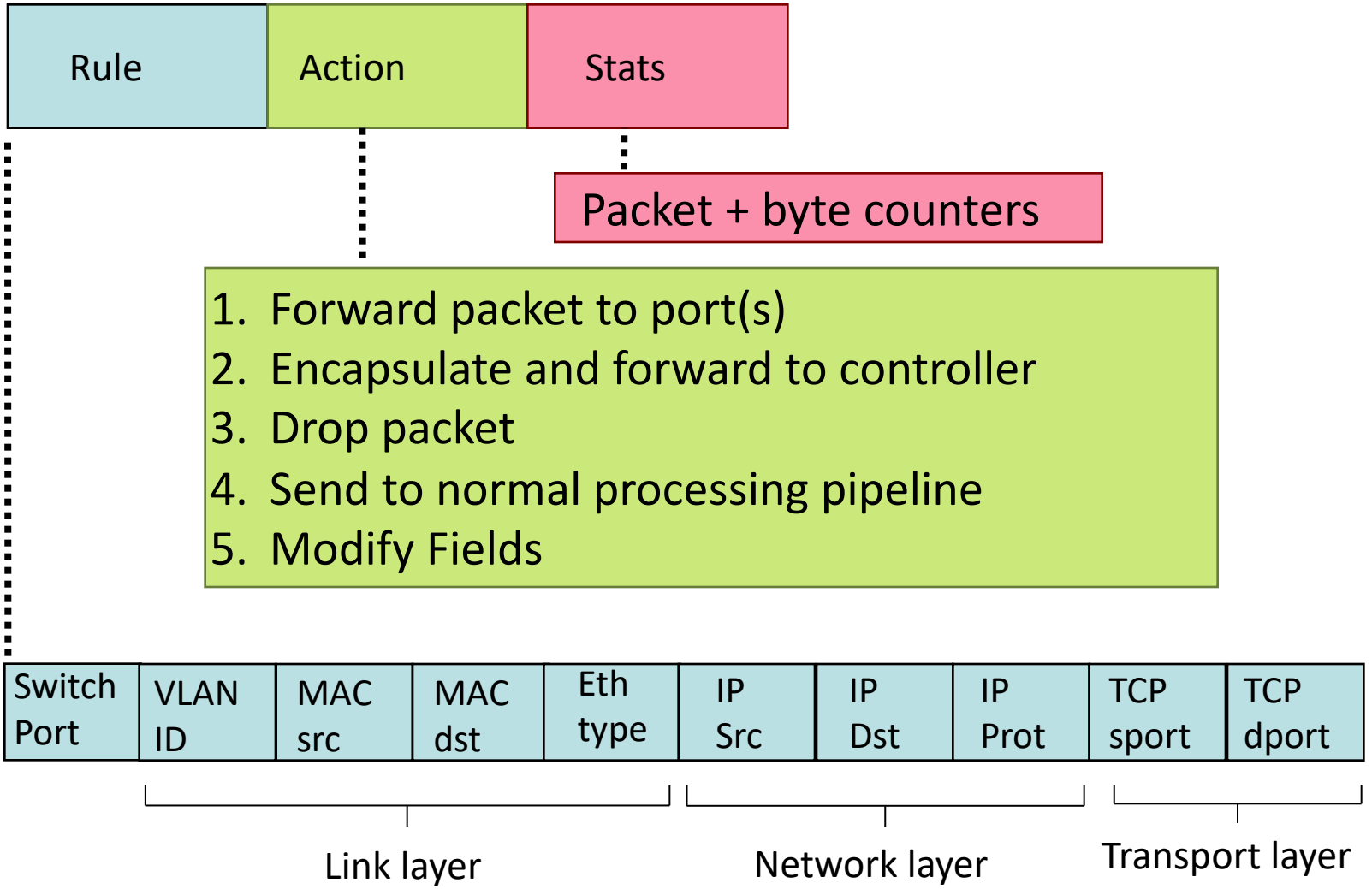
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets



* : wildcard

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow: Flow Table Entries

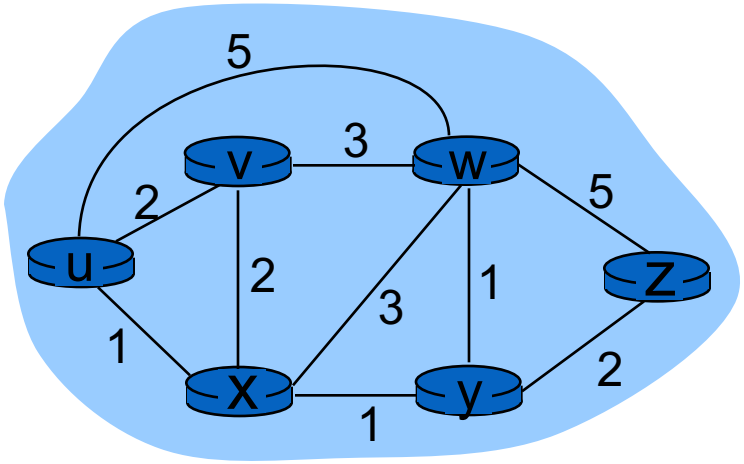


Routing protocols

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

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

Graph abstraction of the network



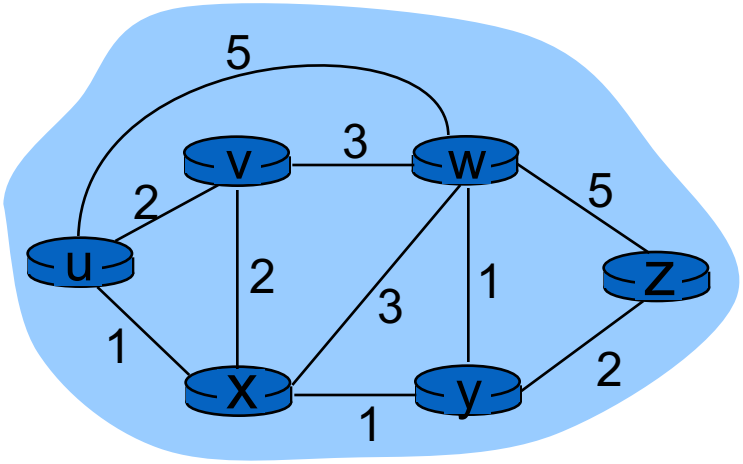
graph: G = (N,E)

N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

aside: graph abstraction is useful in other network contexts, e.g., P2P, where *N* is set of peers and *E* is set of TCP connections

Graph abstraction: costs



c(x,x') = cost of link (x,x')
e.g., c(w,z) = 5

cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

Routing algorithm classification

Q: 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

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

- routes change more quickly
  - 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 *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.'s

notation:

- c(x,y): link cost from node x to y; = ∞ 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
- N': set of nodes whose least cost path definitively known

Dijsktra's algorithm

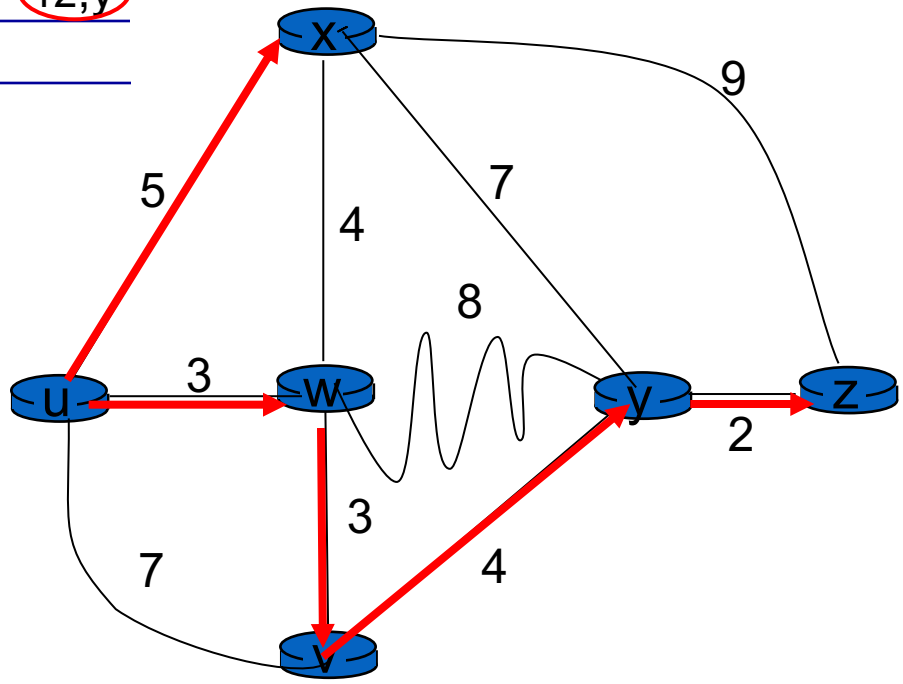
```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5           then D(v) = c(u,v)
6      else D(v) = ∞
7
8   Loop
9     find w not in N' such that D(w) is a minimum
10     add w to N'
11     update D(v) for all v adjacent to w and not in N' :
12        D(v) = min( D(v), D(w) + c(w,v) )
13     /* new cost to v is either old cost to v or known
14      shortest path cost to w plus cost from w to v */
15  until all nodes in N'
```

Dijkstra's algorithm: 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	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

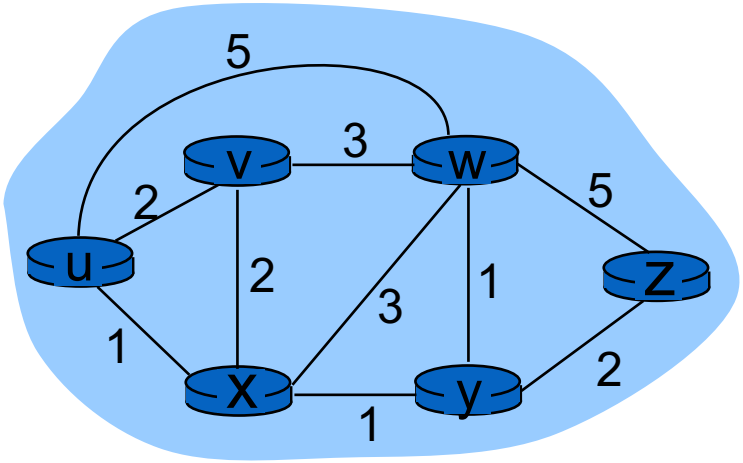
notes:

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



Dijkstra's algorithm: another example

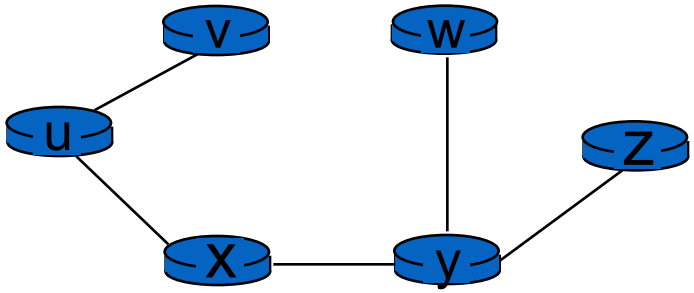
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



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

Dijkstra's algorithm: example (2)

resulting shortest-path tree from u:



resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

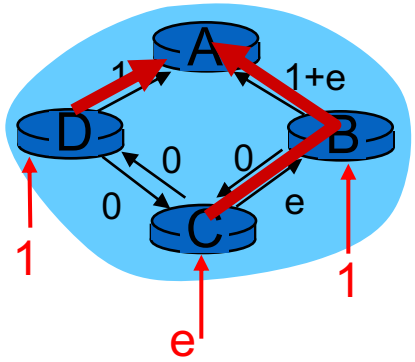
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

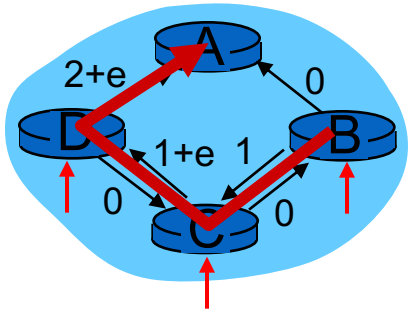
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

oscillations possible:

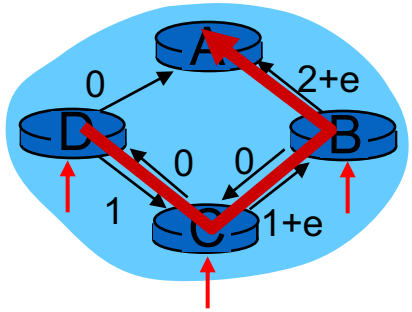
- e.g., support link cost equals amount of carried traffic:



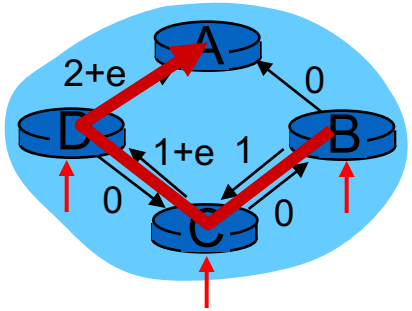
initially



given these costs, find new routing…. resulting in new costs



given these costs, find new routing…. resulting in new costs



given these costs, find new routing…. resulting in new costs

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

$d_x(y)$:= cost of least-cost path from x to y

then

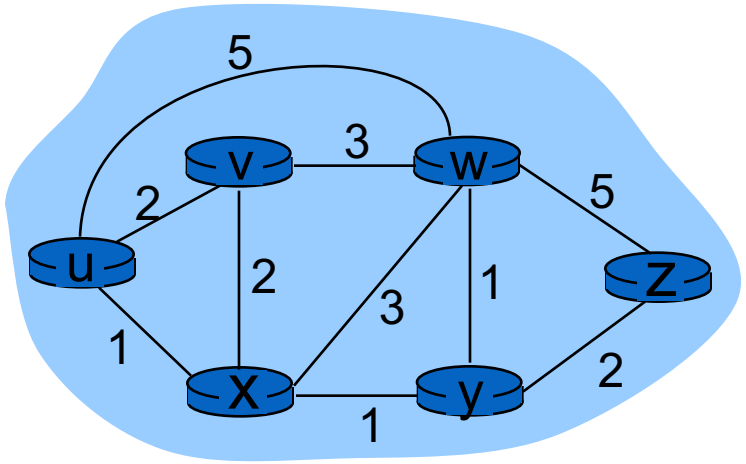
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford example



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F 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 is next hop in shortest path, used in forwarding table

Distance vector algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- node x:
  - knows cost to each neighbor v: c(x,v)
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

Distance vector algorithm

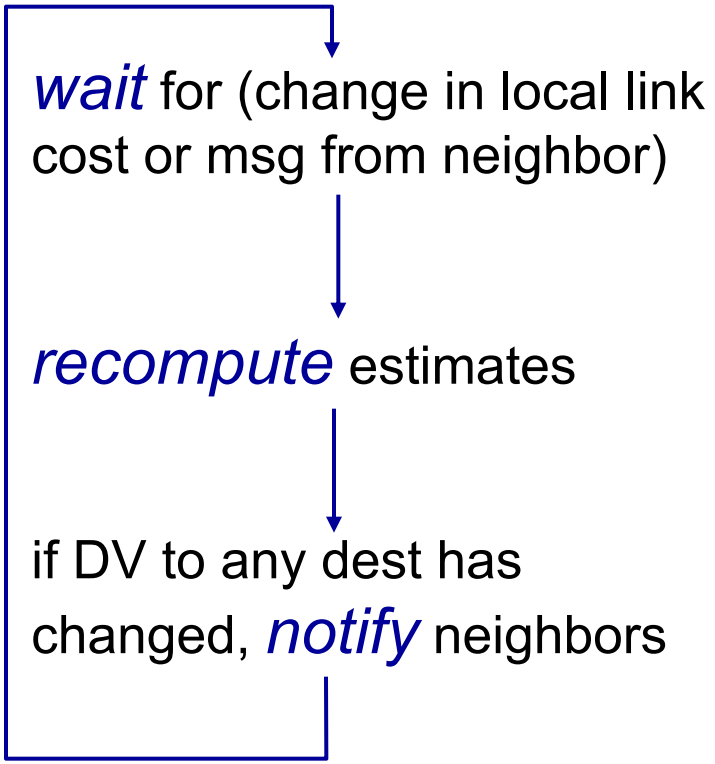
iterative, asynchronous:
each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

each node:



$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) +$
$D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

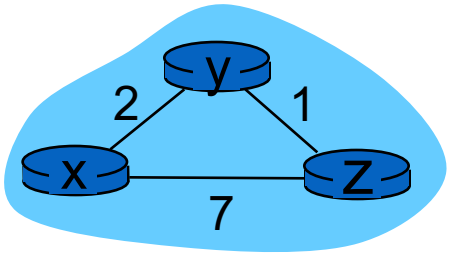
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



time

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) +$
$D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

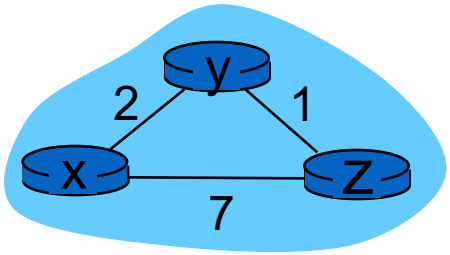
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

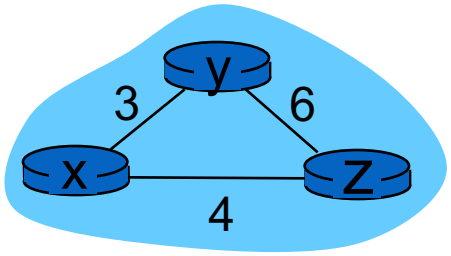
from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



time



node x table

from \ cost to	x	y	z
x	0	3	4
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	3	4
y	3	0	6
z	4	6	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	3	0	6
z	∞	∞	∞

from \ cost to	x	y	z
x	0	3	4
y	3	0	6
z	4	6	0

node z table

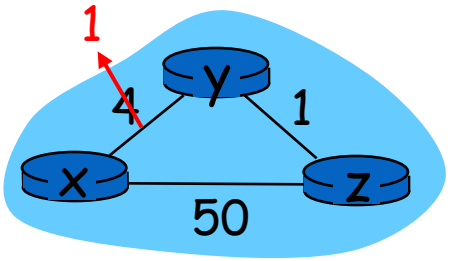
from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	4	6	0

from \ cost to	x	y	z
x	0	3	4
y	3	0	6
z	4	6	0

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

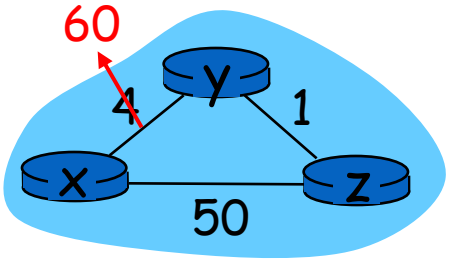
t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

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

Distance vector: link cost changes

link cost changes:

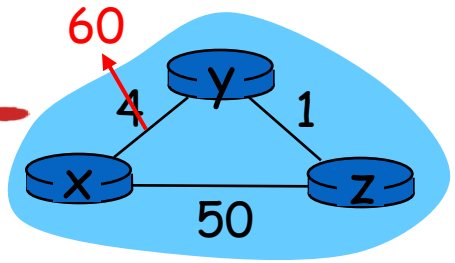
- node detects local link cost change
- *bad news travels slow* - "count to infinity" problem!
- 44 iterations before algorithm stabilizes: see text



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?

Count to infinity



$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{60+0 , 50+1\} = 51$

node x table

from \ cost to	x	y	z
x	0	51	50
y	4	0	1
z	5	1	0

from \ cost to	x	y	z
x	0	51	50
y	6	0	1
z	5	1	0

from \ cost to	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

node y table

from \ cost to	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

6

from \ cost to	x	y	z
x	0	51	50
y	6	0	1
z	5	1	0

from \ cost to	x	y	z
x	0	51	50
y	8	0	1
z	7	1	0

node z table

from \ cost to	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

from \ cost to	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

An example for Distance Vector routing with Poisson reverse (PR)

A's routing table

Dst	Dis	Nex
B	1	B
C	3	B
D	4	B
E	4	B
F	7	B
G	6	H
H	2	H

A's update to B w/o PR

B	1
C	3
D	4
E	4
F	7
G	6
H	2

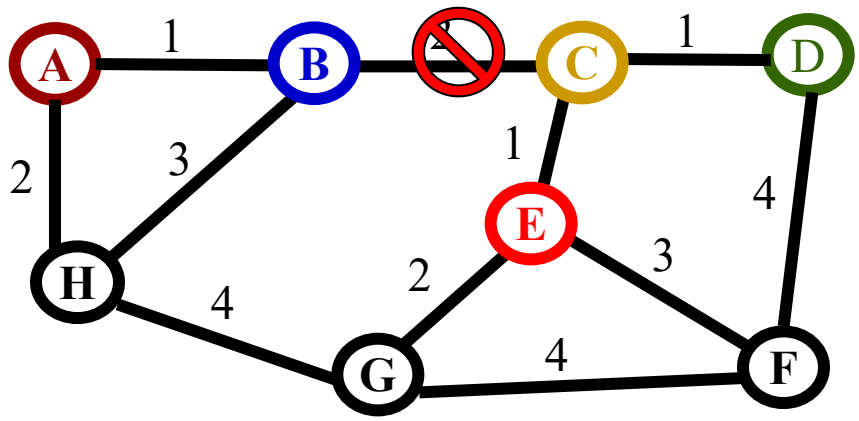
B's routing table

Dst	Dis	Nex
A	1	A
C	2	
D		C
E		
F		
G	5	
H	3	H

Dst	Dis	Nex	
A	1	A	
C	4	A	✖
D	5	A	✖
E	5	A	✖
F	8	A	✖
G	7	A	✓
H	3	H	

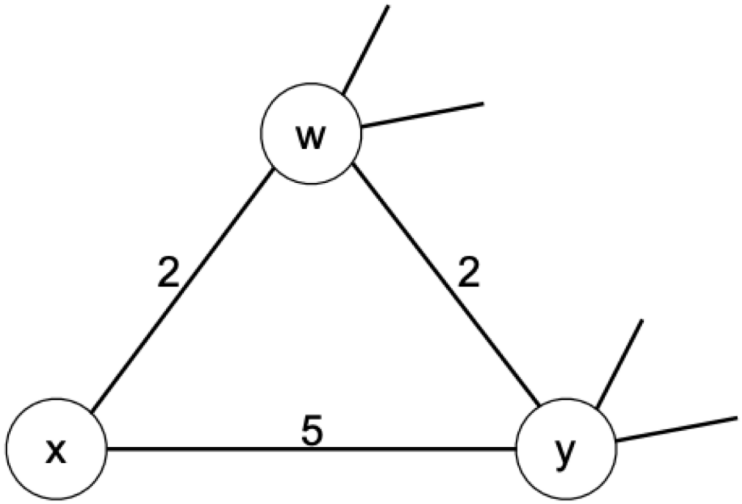
A's update to B with PR:

B	1
C	∞
D	∞
E	∞
F	∞
G	6
H	2



Exercise

Q: Consider the network fragment shown below. x has only two attached neighbors, w and y. w has a minimum-cost path to destination u (not shown) of 5, and y has a minimum-cost path to destination u of 6. The complete path from w and y to u are not shown. All link costs in the network have strictly positive integer values.



Exercise

a. Give x's distance vector for destinations w, y, and u.

A: d(x, w) = 2, d(x, y) = 4, d(x, u) = 7

b. Give a link-cost change for either c(x, w) or c(x, y) such that x will inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.

A: change c(x, w) to >= 6

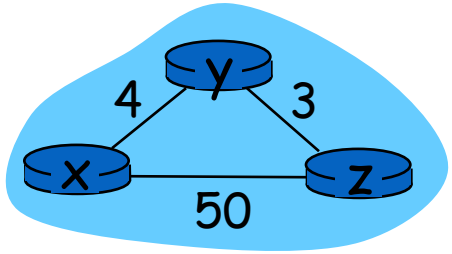
Exercise

c. Give a link-cost change for either c(x, w) or c(x, y) such that x will not inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm.

A: change c(x, y) to >=1

Exercise

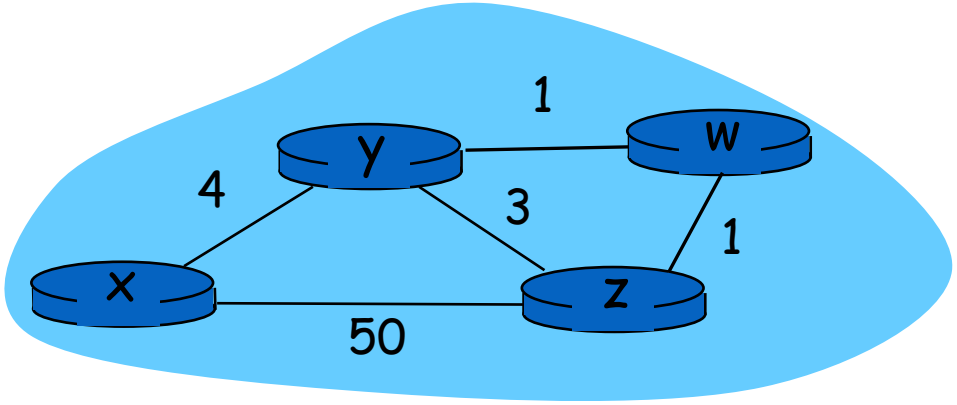
Q: Consider the network shown below. Suppose there is another router w, connected to router y and z. The costs of all links are given as follows: c(x, y) = 4, c(x, z) = 50, c(y, w) = 1, c(z,w) = 1, c(y, z) = 3. Suppose that poisoned reverse is used in the distance-vector routing algorithm.



a. When the distance vector routing is stabilized, router w, y, and z inform their distance to x to each other. What distance values do they tell each other?

Exercise

A:



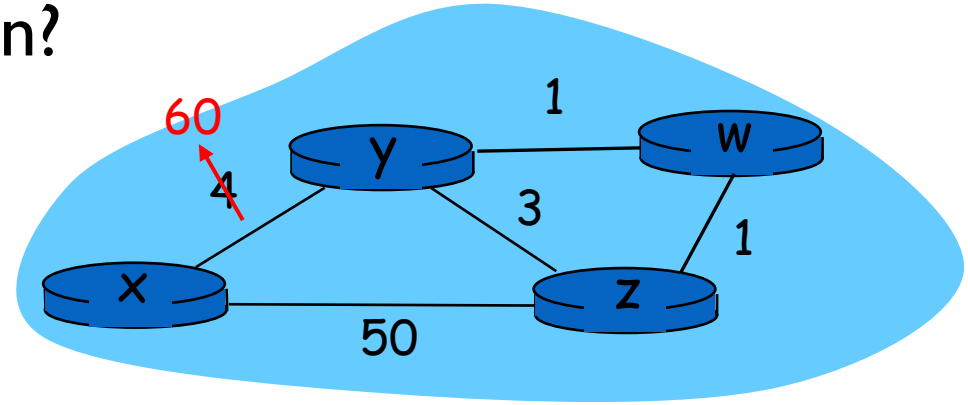
y tells w and z d(y, x) = 4

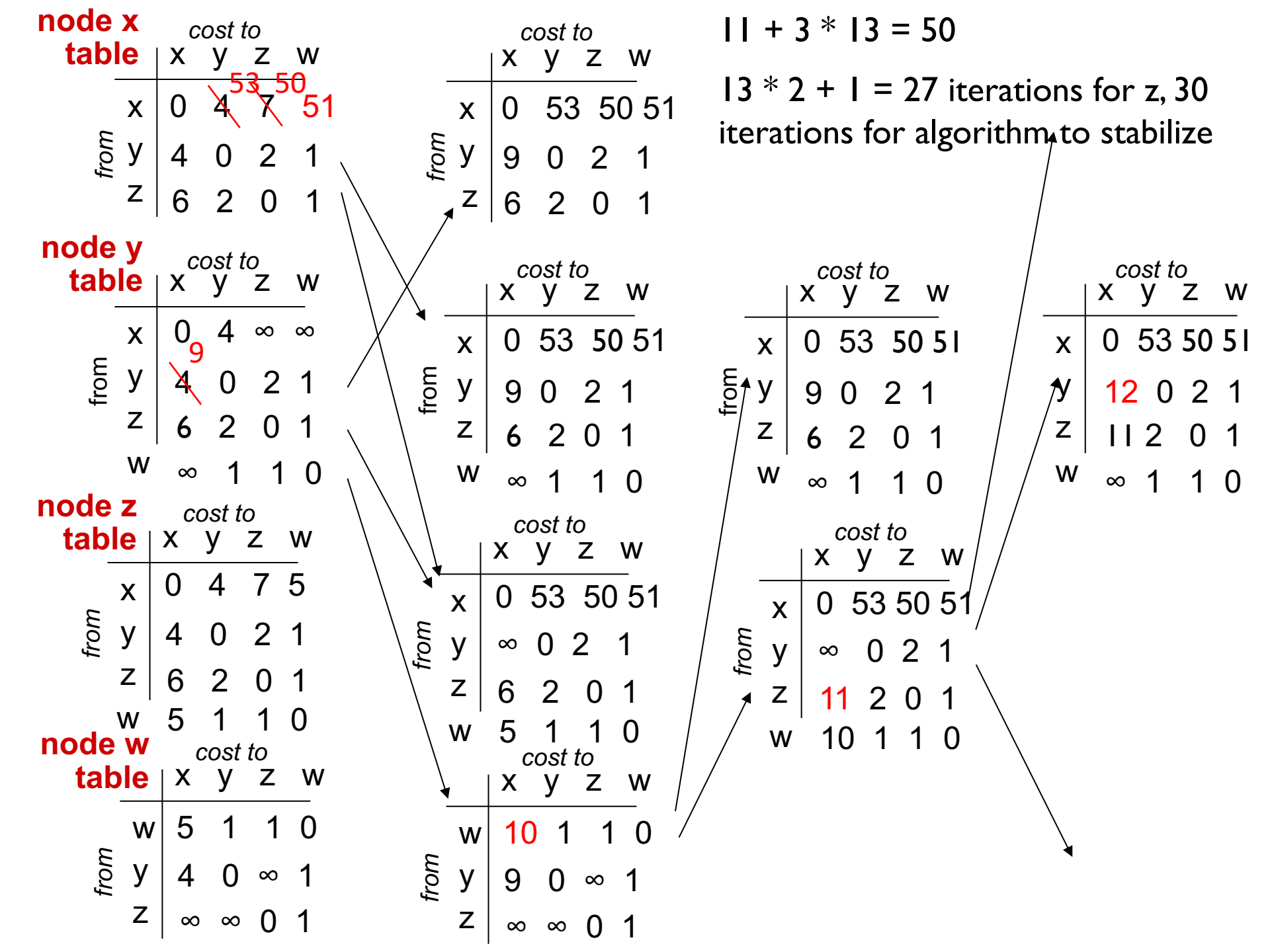
z tells y d(z, x) = 6 and tells w d(z, x) = ∞

w tells y d(w, x) = ∞ and tells z d(w, x) = 5

Exercise

b. Now suppose that the link cost between x and y increases to 60. Will there be a count-to-infinity problem even if poisoned reverse is used? Why or why not? If there is a count-to-infinity problem, then how many iterations needed for the distance-vector routing to reach a stable state again?




node x table
cost to
x y z w
from
x 0 4 7 51
y 4 0 2 1
z 6 2 0 1
53 50
node y table
cost to
x y z w
from
x 0 4 ∞ ∞
y 4 0 2 1
z 6 2 0 1
w ∞ 1 1 0
9
node z table
cost to
x y z w
from
x 0 4 7 5
y 4 0 2 1
z 6 2 0 1
w 5 1 1 0
node w table
cost to
x y z w
from
w 5 1 1 0
y 4 0 ∞ 1
z ∞ ∞ 0 1
cost to
x y z w
from
x 0 53 50 51
y 9 0 2 1
z 6 2 0 1
cost to
x y z w
from
x 0 53 50 51
y 9 0 2 1
z 6 2 0 1
w ∞ 1 1 0
cost to
x y z w
from
x 0 53 50 51
y ∞ 0 2 1
z 6 2 0 1
w 5 1 1 0
cost to
x y z w
from
w 10 1 1 0
y 9 0 ∞ 1
z ∞ ∞ 0 1
11 + 3 * 13 = 50
13 * 2 + 1 = 27 iterations for z, 30 iterations for algorithm to stabilize
cost to
x y z w
from
x 0 53 50 51
y 9 0 2 1
z 6 2 0 1
w ∞ 1 1 0
cost to
x y z w
from
x 0 53 50 51
y ∞ 0 2 1
z 11 2 0 1
w 10 1 1 0
cost to
x y z w
from
x 0 53 50 51
y 12 0 2 1
z 11 2 0 1
w ∞ 1 1 0

c. How do you modify c(y, z) such that there is no count-to-infinity problem at all if c(x, y) changes from 4 to 60?

A:

c(z, y) + d(z, x) = c(z, y) + 6 >= 52,

So, c(z, y) >= 46

Comparison of LS and DV algorithms

message complexity

- ***LS:*** with n nodes, E links, O(nE) msgs sent
- ***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

Making routing scalable

our routing study thus far - idealized

- all routers identical
- network "flat"

… *not* true in practice

scale: with billions of destinations:

- can't store all destinations 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

Internet approach to scalable routing

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

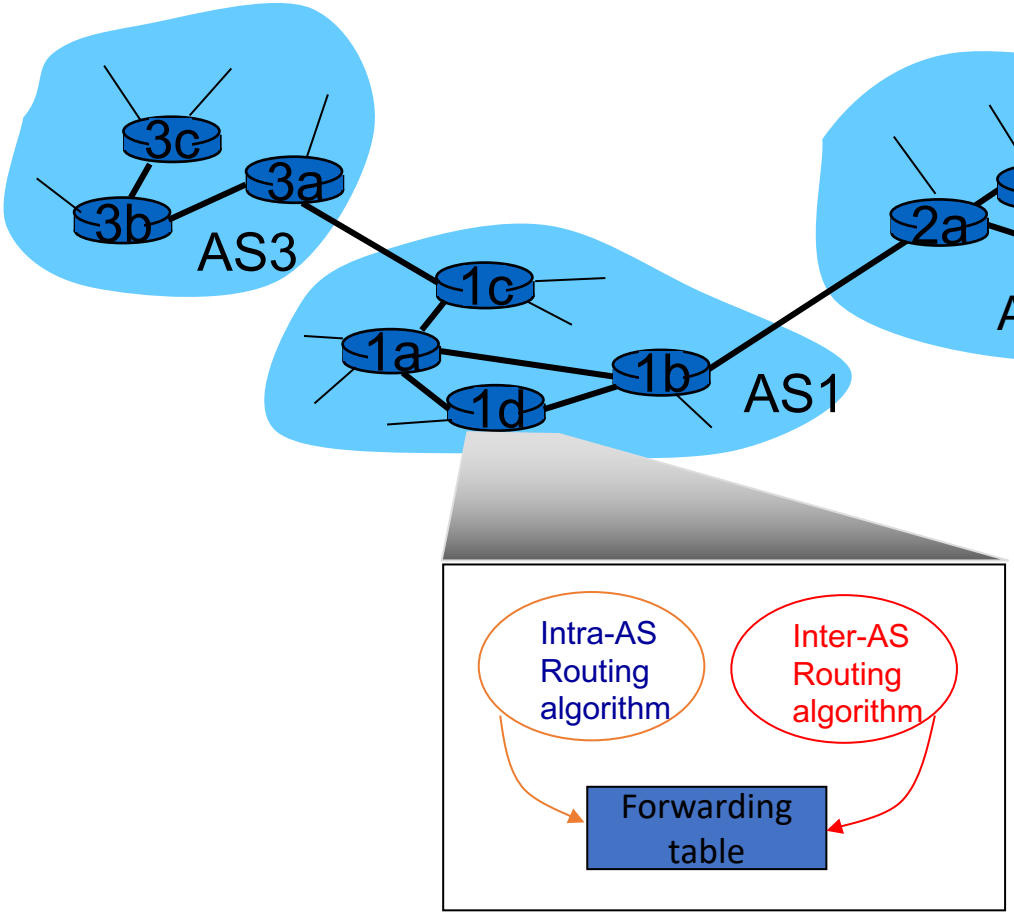
intra-AS routing

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

inter-AS routing

- routing among AS'es
- gateways perform inter-domain routing (as well as intra-domain routing)

Interconnected ASes



- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS routing determine entries for destinations within AS
  - inter-AS & intra-AS determine entries for external destinations

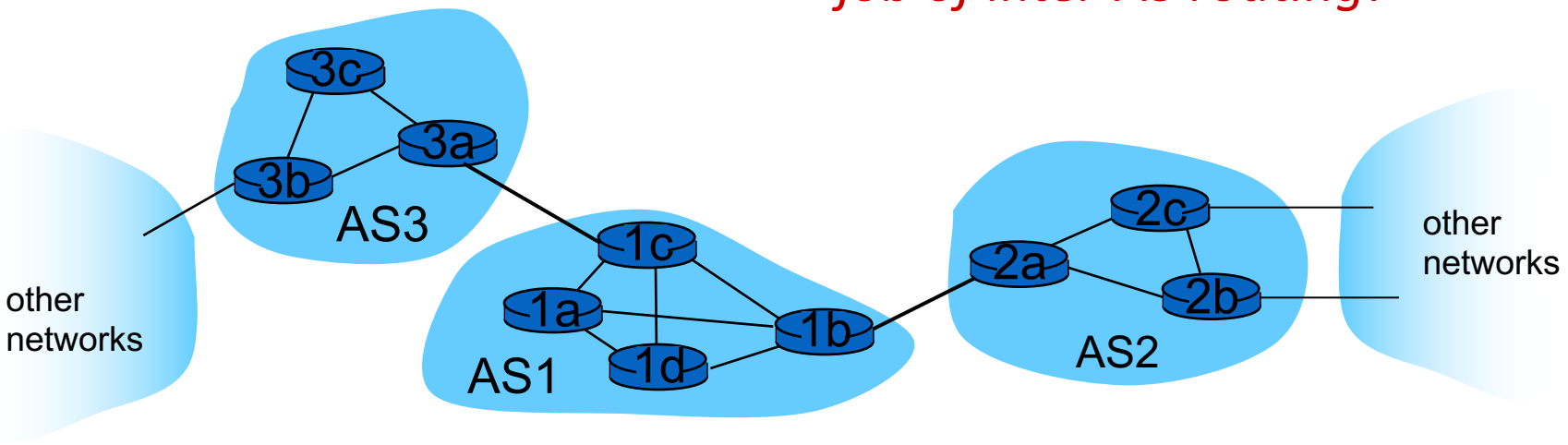
Inter-AS tasks

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router, but which one?

AS1 must:

1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



Intra-AS Routing

- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF)
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)

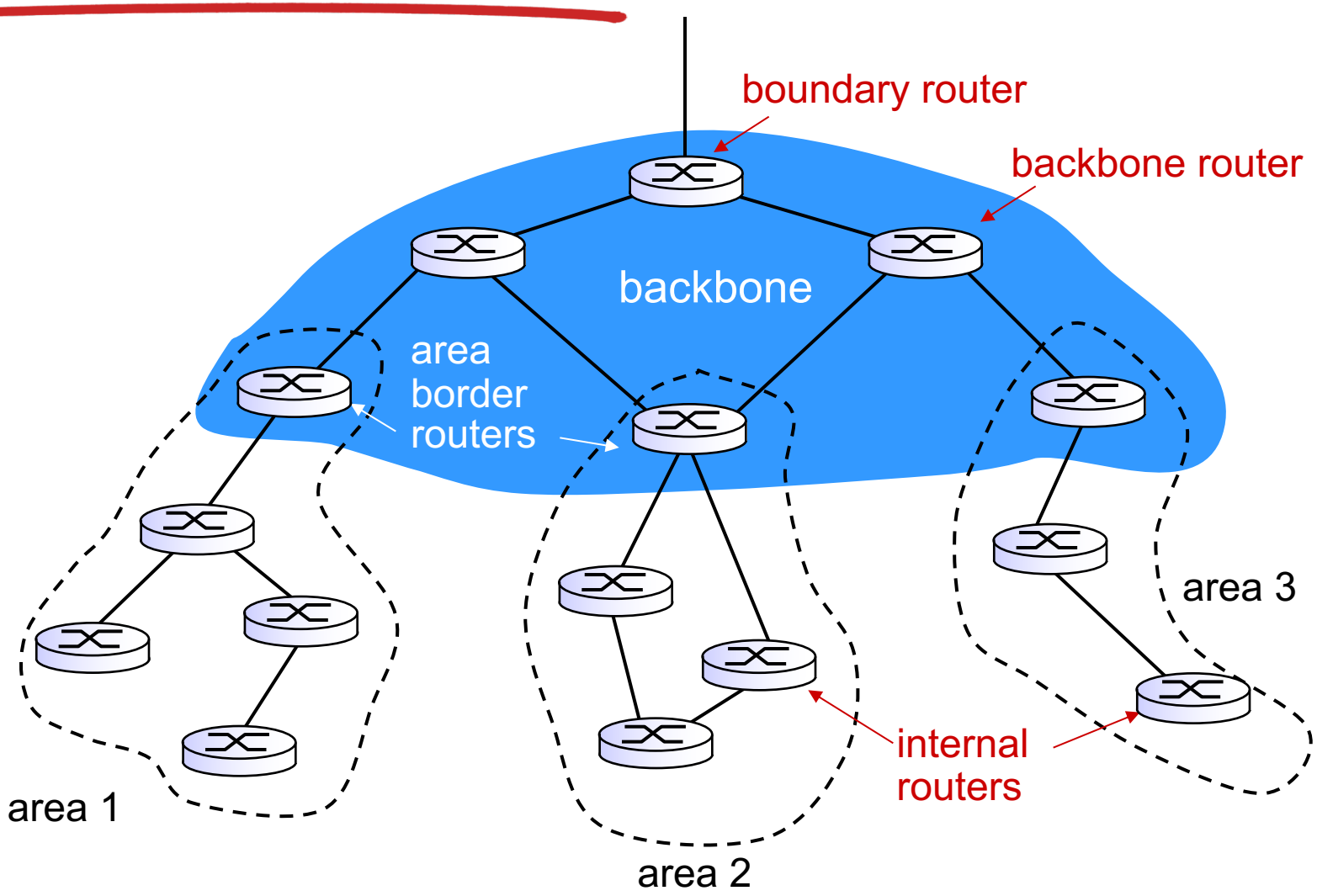
OSPF (Open Shortest Path First)

- "open": publicly available
- uses link-state algorithm
  - link state packet dissemination
  - topology map at each node
  - route computation using Dijkstra's algorithm
- router floods OSPF link-state advertisements to all other routers in *entire* AS
  - carried in OSPF messages directly over IP (rather than TCP or UDP
  - link state: for each attached link
- *IS-IS routing* protocol: nearly identical to OSPF

OSPF "advanced" features

- *security:* all OSPF messages authenticated (to prevent malicious intrusion)
- multiple same-cost paths allowed (only one path in RIP)
- for each link, multiple cost metrics for different TOS (e.g., satellite link cost set low for best effort ToS; high for real-time ToS)
- integrated uni- and multi-cast support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- hierarchical OSPF in large domains.

Hierarchical OSPF



Hierarchical OSPF

- *two-level hierarchy:* local area, backbone.
  - link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- *area border routers:* "summarize" distances to nets in own area, advertise to other Area Border routers.
- *backbone routers:* run OSPF routing limited to backbone.
- *boundary routers:* connect to other AS'es.