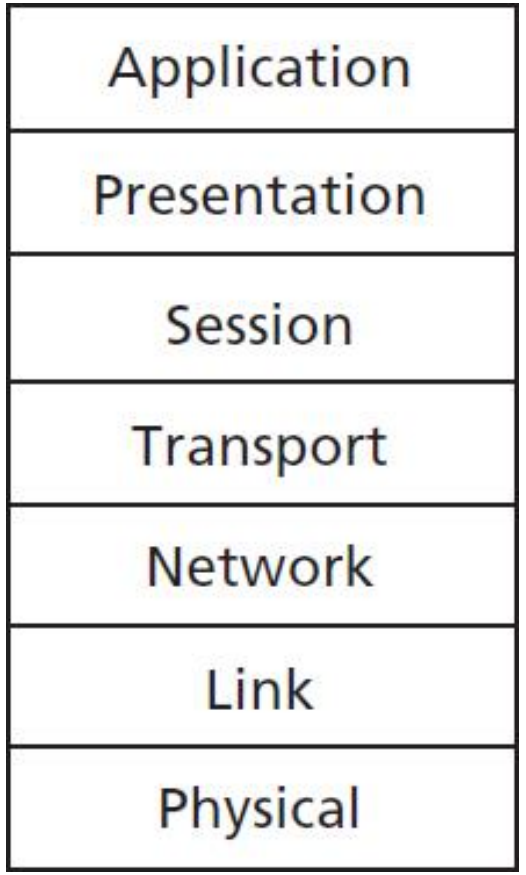
ISO/OSI Reference Model

presentation: allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions

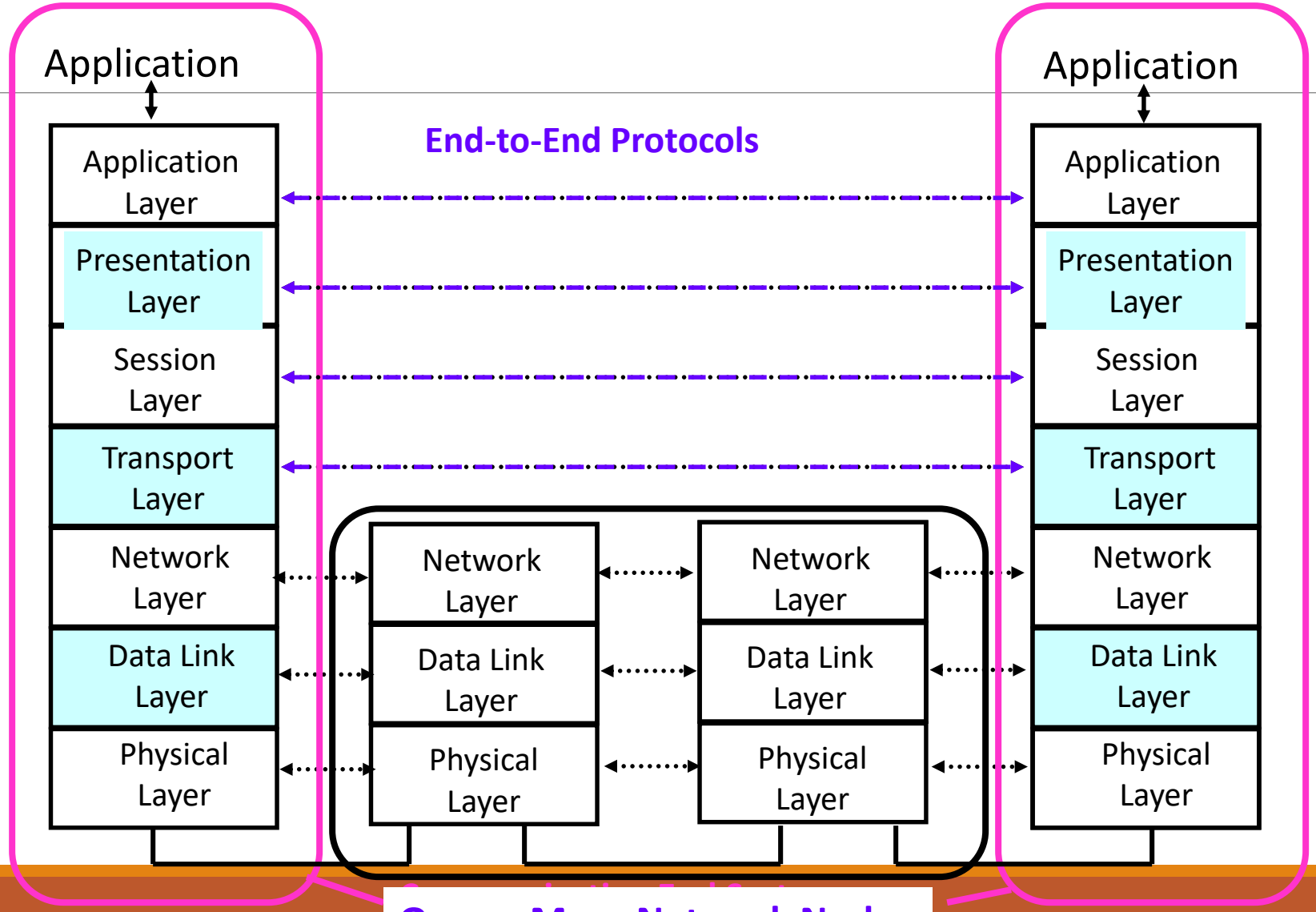
session: synchronization, checkpointing, recovery of data exchange

Internet stack "missing" these layers!

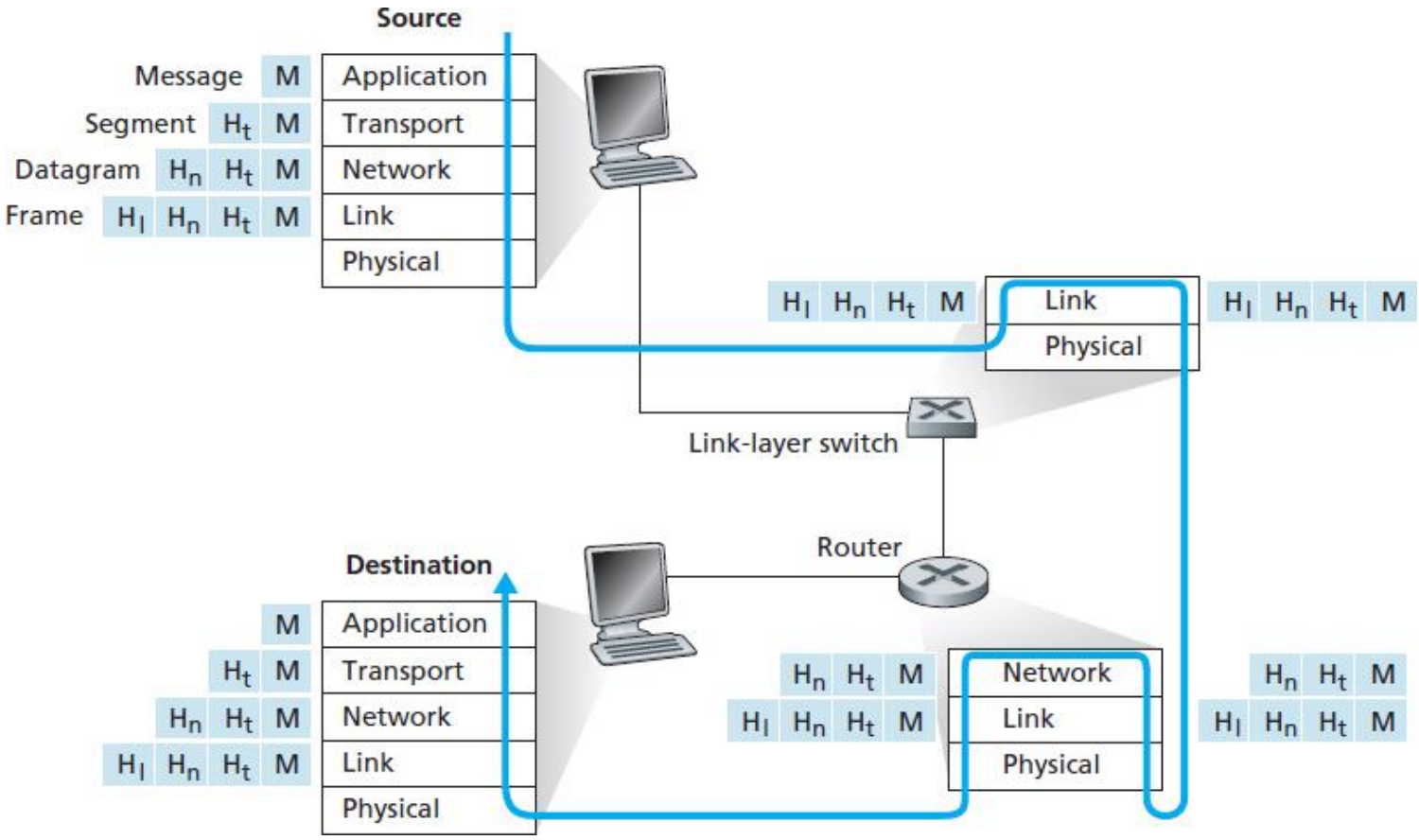
- these services, **if needed**, must be implemented in application
- needed?



7-layer OSI Reference Model



Encapsulation



Application Layer Protocols -1

DR DUY NGO

SCHOOL OF ELECTRICAL ENGINEERING &COMPUTING

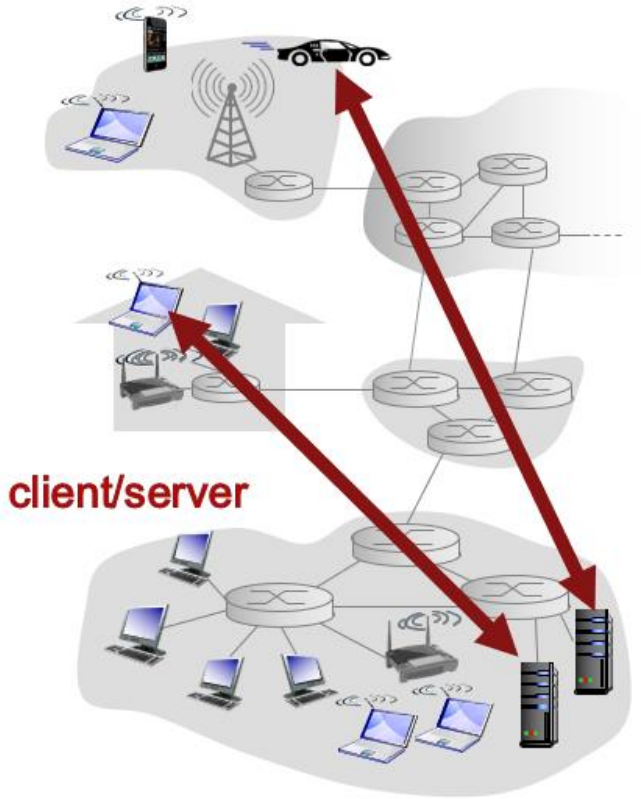
Client-Server Architecture

server:

- always-on host
- permanent IP address
- data centers for scaling

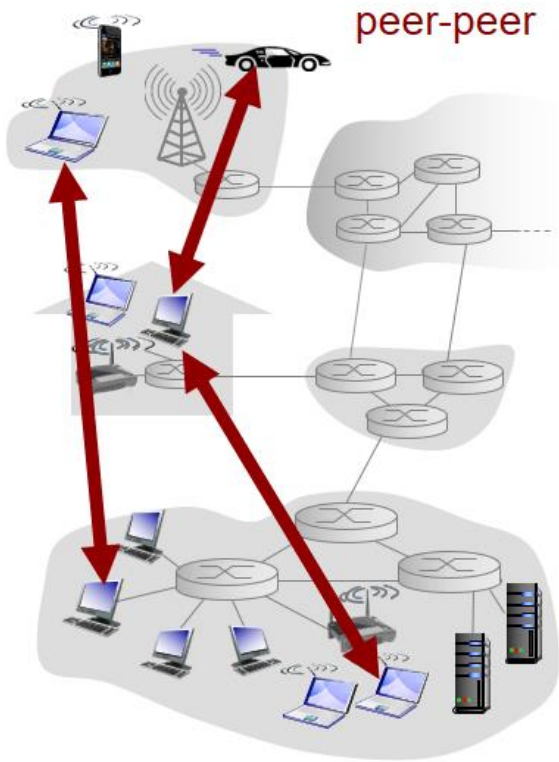
clients:

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



P2P (Peer to Peer) Architecture

- **Minimum** reliance on dedicated servers
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - **self scalability** – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Processes Communicating

process*:* program running within a host

- within same host, two processes communicate using **inter-process communication** (defined by OS: Operating System)
- processes in different hosts communicate by exchanging **messages**

clients, servers

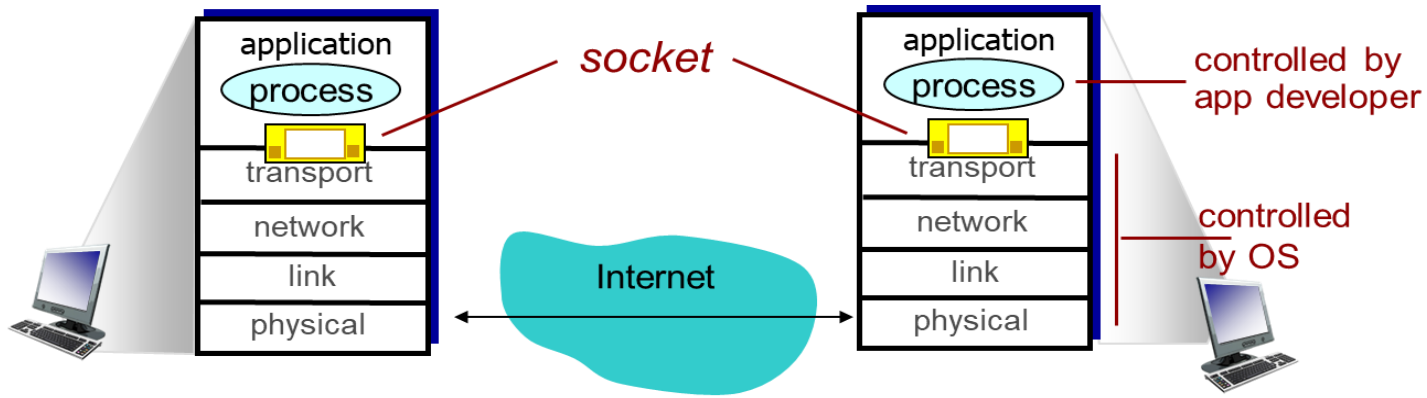
client process: process that initiates communication

server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
  - sending process sends message to 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
- more shortly…

App-layer protocol defines

- **types of messages exchanged,**
  - e.g., request, response
- **message syntax:**
  - what fields in messages & how fields are delineated
- **message semantics**
  - meaning of information in fields
- **rules** for when and how processes send & respond to messages

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

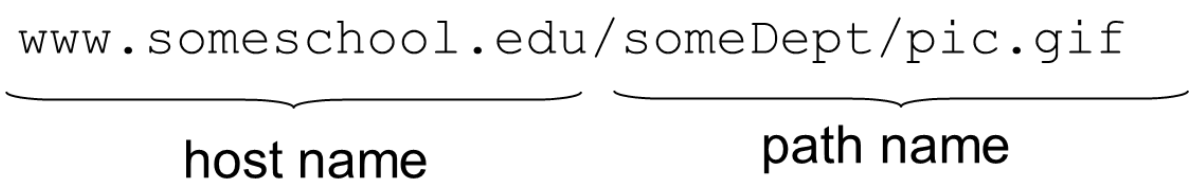
Internet Apps: Application, Transport Protocols

Application	**Application layer protocol**	**Underlying transport protocol**
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Web and HTTP

First, a review…

- **web page** consists of **objects**
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of **base HTML-file** which includes **several referenced objects**
- each object is addressable by a **URL,** e.g.,



HTTP Overview (1 of 2)

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 (2 of 2)

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

non-persistent HTTP

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

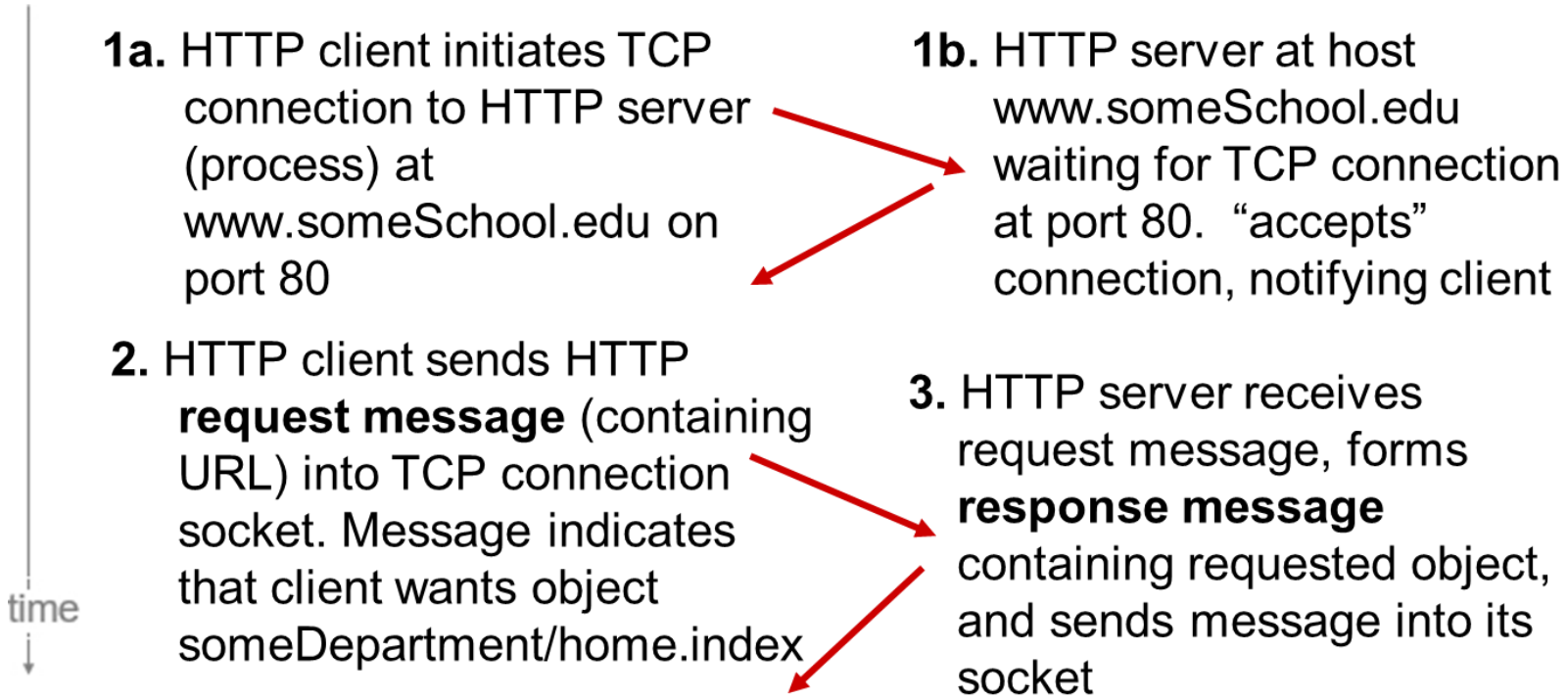
persistent HTTP

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

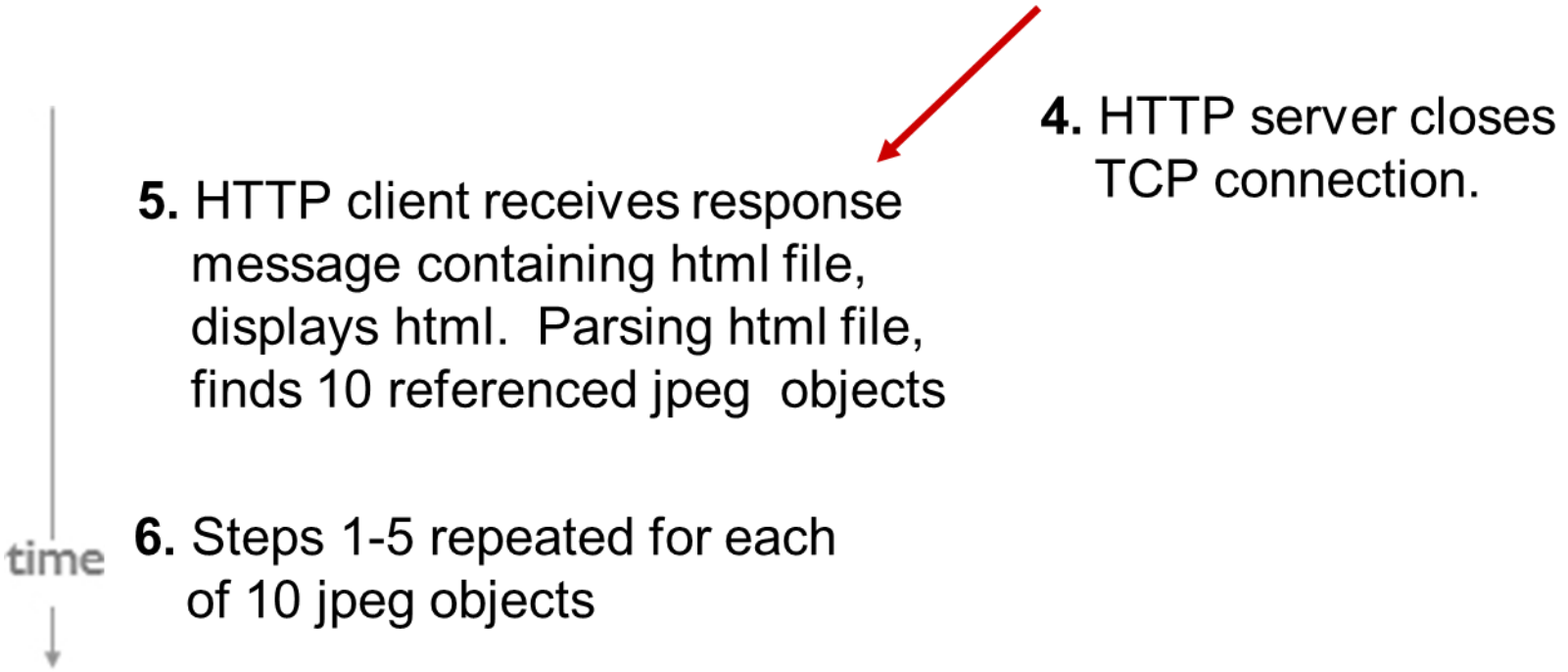
Non-Persistent HTTP (1 of 2)

suppose user enters URL:

www.someSchool.edu/someDepartment/home.index



Non-Persistent HTTP (2 of 2)



Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTT s 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

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: Services, Structure

DNS services

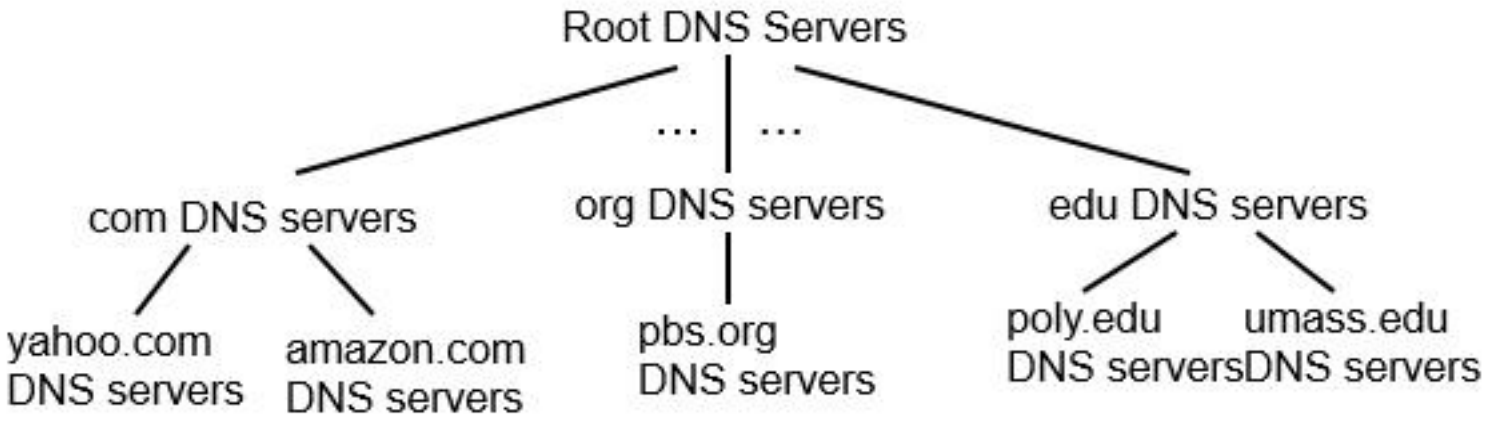
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

A: **doesn‘t scale!**

DNS: A Distributed, Hierarchical Database



client wants IP for www.amazon.com; **1st approximation:**

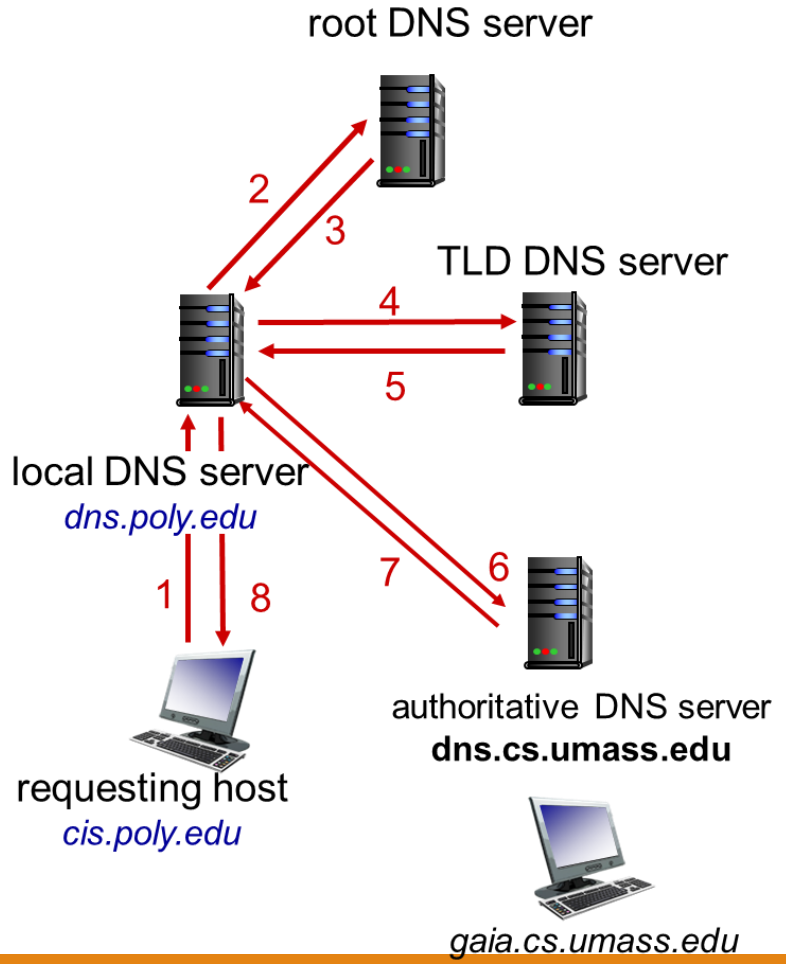
- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS Name Resolution Example (1 of 2)

- 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"

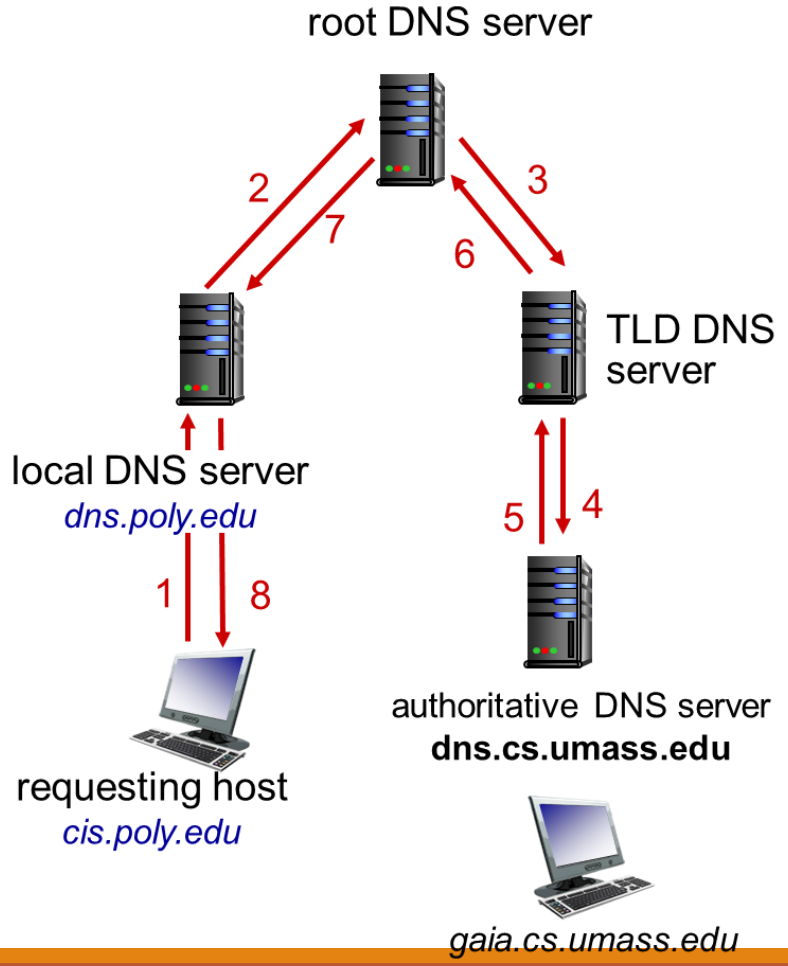


DNS Name Resolution Example (2 of 2)

recursive query:

puts burden of name resolution on contacted name server

heavy load at upper levels of hierarchy?

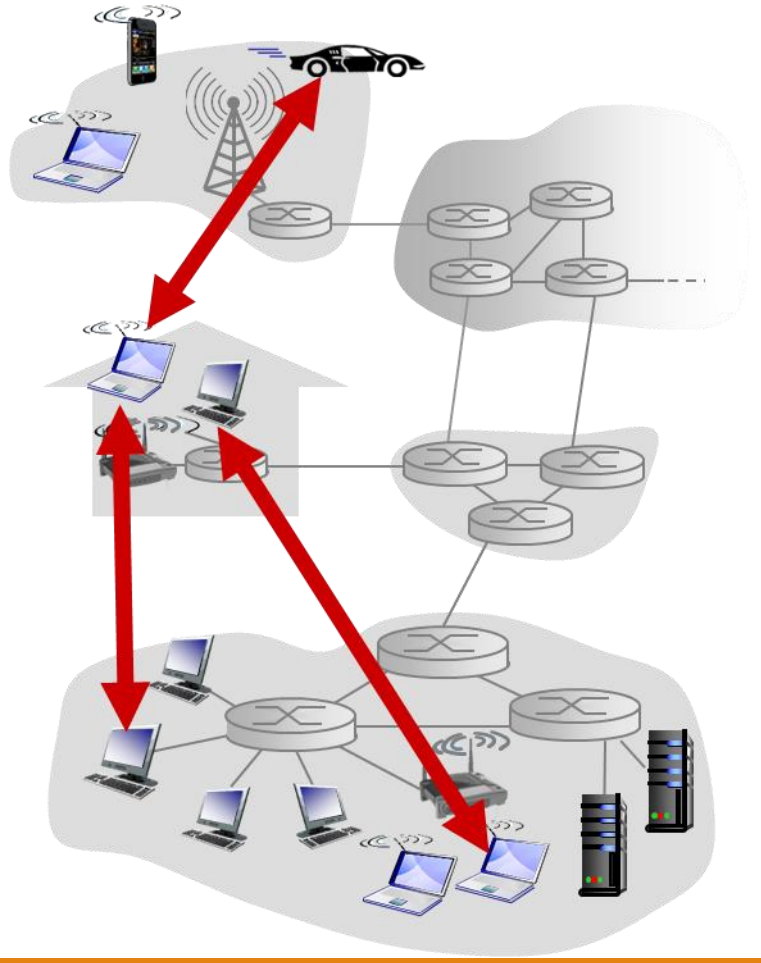


Pure P2P Architecture

- **Not required a** always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

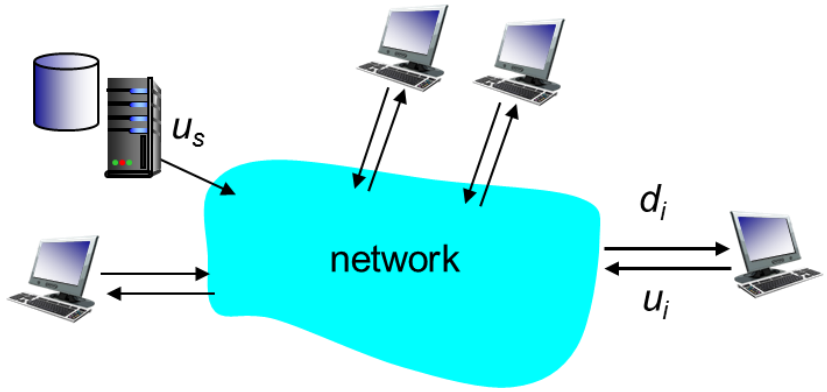
examples:

- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File Distribution Time: Client-Server

- **server transmission:** must sequentially send (upload) *N* file copies:
  - time to send one copy: $\frac{F}{u_s}$
  - time to send *N* copies: $\frac{NF}{u_s}$
- **client:** each client must download file copy
  - d_{min} = min client download rate
  - min client download time: $\frac{F}{d_{min}}$



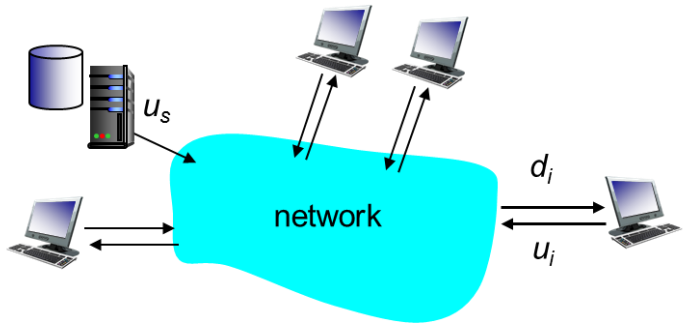
time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File Distribution Time: P2P

- **server transmission:** must upload at least one copy
  - time to send one copy: $\frac{F}{u_s}$
- **client:** each client must download file copy
  - min client download time: $\frac{F}{d_{min}}$
- **clients:** as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \sum u_i$



time to distribute F to N clients using P2P approach

$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

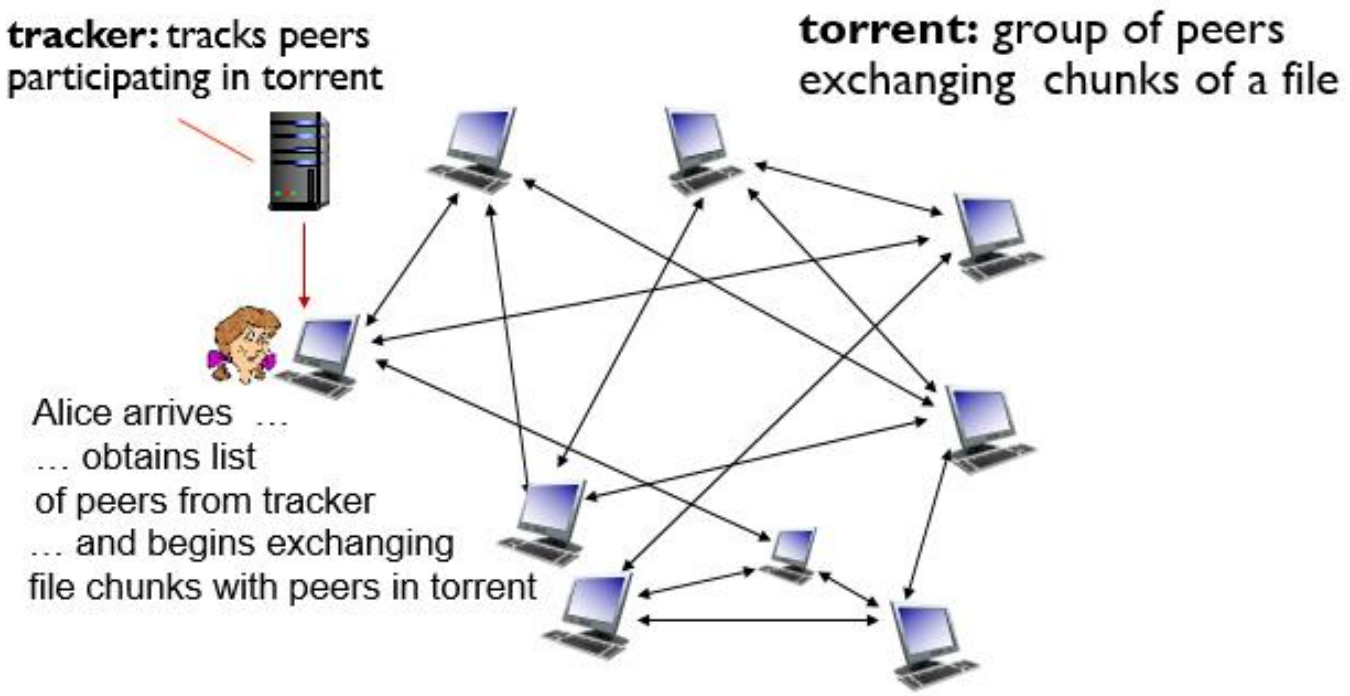
increases linearly in *N* …

… but so does this, as each peer brings service capacity

P2P File Distribution: BitTorrent (1 of 2)

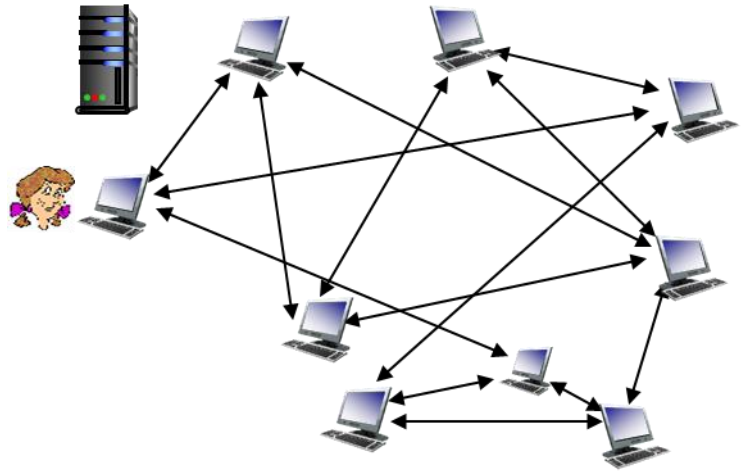
file divided into 256Kb chunks

peers in torrent send/receive file chunks



P2P File Distribution: BitTorrent (2 of 2)

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- **churn:** peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: Requesting, Sending File Chunks

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

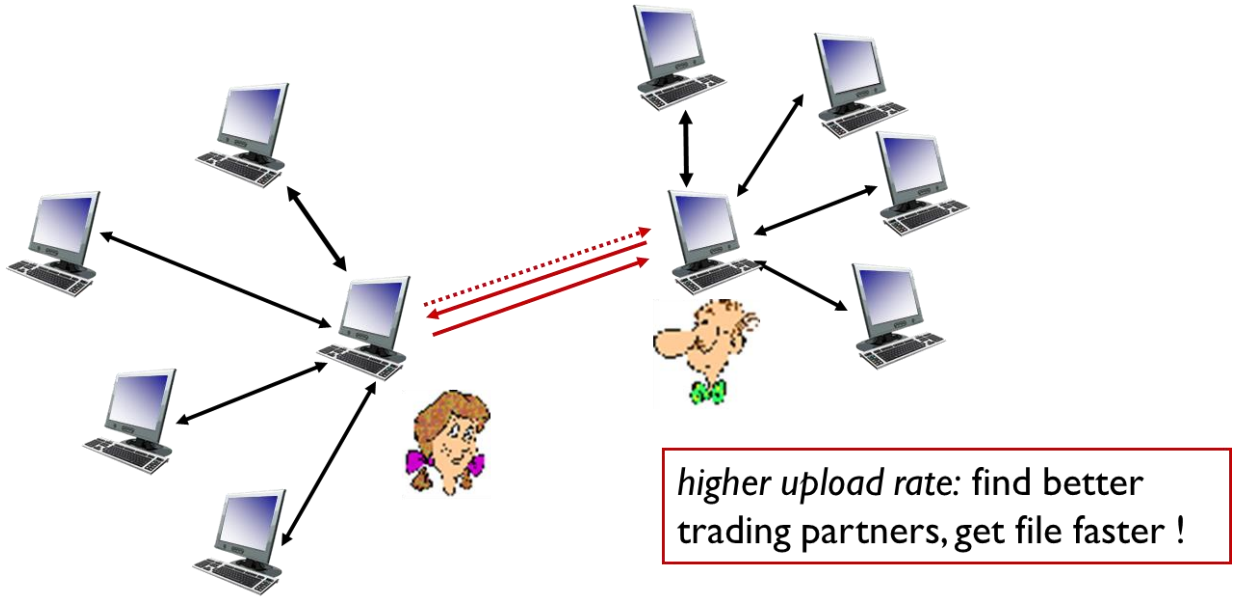
- Alice sends chunks to those four peers currently sending her chunks **at highest rate**
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: Tit-For-Tat

(1) Alice “optimistically unchokes” Bob

(2) Alice becomes one of Bob’s top-four providers; Bob reciprocates

(3) Bob becomes one of Alice’s top-four providers



Streaming Multimedia: DASH (1 of 2)

DASH: Dynamic, **A**daptive **S**treaming over **H**TTP

server:

- divides video file into multiple chunks
- each chunk stored, encoded at different rates
- **manifest file:** provides URLs for different chunks

client:

- periodically measures server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time)

Streaming Multimedia: DASH (2 of 2)

"intelligence" at client: client determines

- **when** to request chunk (so that buffer starvation, or overflow does not occur)
- **what encoding rate** to request (higher quality when more bandwidth available)
- **where** to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

Content Distribution Networks (1 of 2)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of **simultaneous** users?

option 1: single, large "mega-server"

- single point of failure
- point of network congestion
- long path to distant clients
- multiple copies of video sent over outgoing link

….quite simply: this solution **doesn't scale**

Content Distribution Networks (2 of 2)

option 2: store/serve multiple copies of videos at multiple geographically distributed sites **(CDN)**

- **enter deep:** push CDN servers deep into many access networks
  - close to users
  - used by Akamai, 1700 locations
- **bring home:** smaller number (10's) of larger clusters in POPs near (but not within) access networks
  - used by Limelight

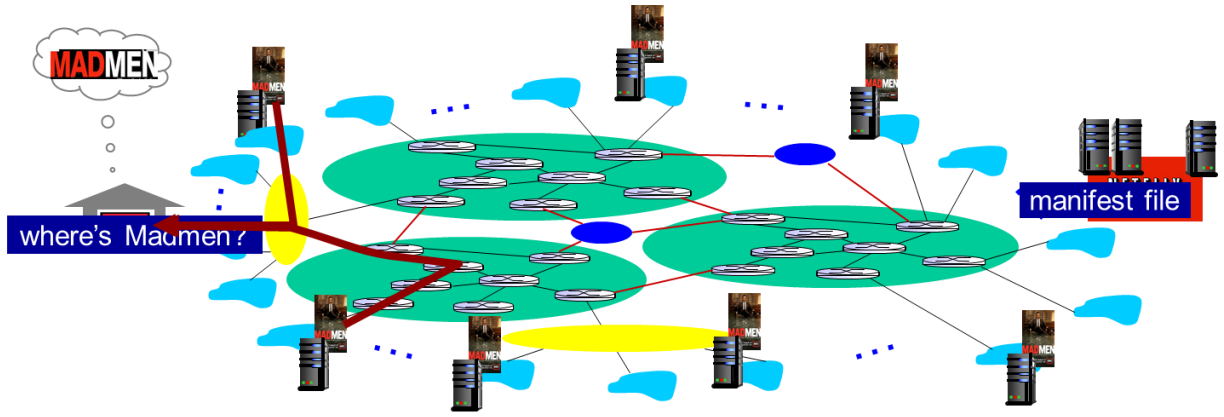
Content Distribution Networks (CDNs) (1 of 2)

CDN: stores copies of content at CDN nodes

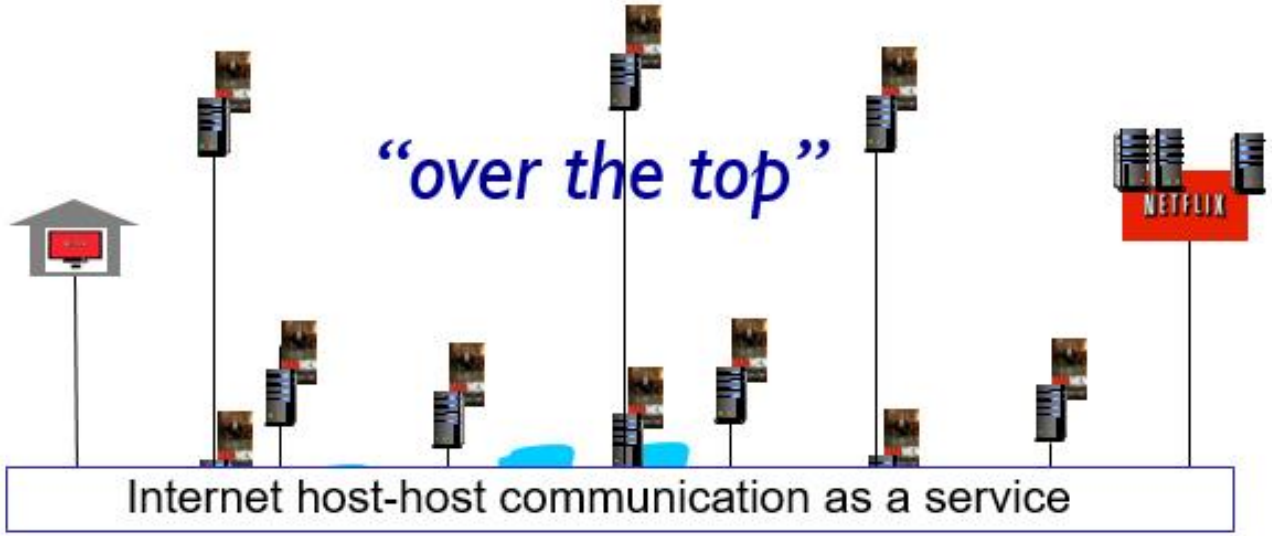
- e.g. Netflix stores copies of MadMen

subscriber requests content from CDN

- directed to nearby copy, retrieves content
- may choose different copy if network path congested



Content Distribution Networks (CDNs) (2 of 2)



OTT challenges: coping with a congested Internet

- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

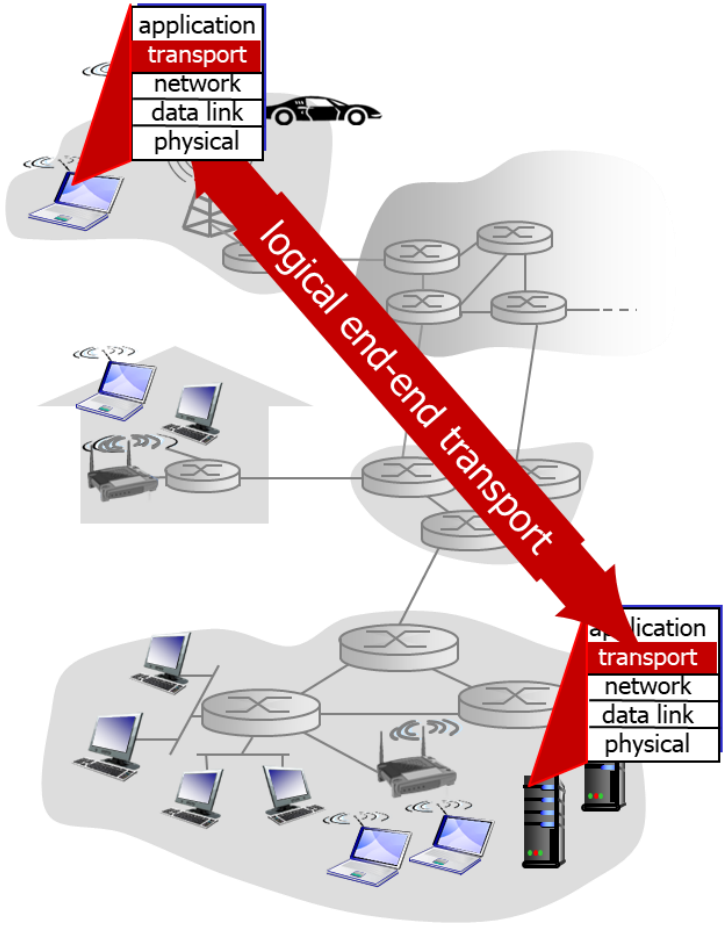
Transport Layer Protocols -1

DR DUY NGO

SCHOOL OF ELECTRICAL ENGINEERING &COMPUTING

Transport Services and Protocols

- provide **logical communication** between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into **segments**, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport vs. Network Layer

- **network layer***:* logical communication between hosts
- **transport layer***:* logical communication between processes
  - relies on, enhances, network layer services

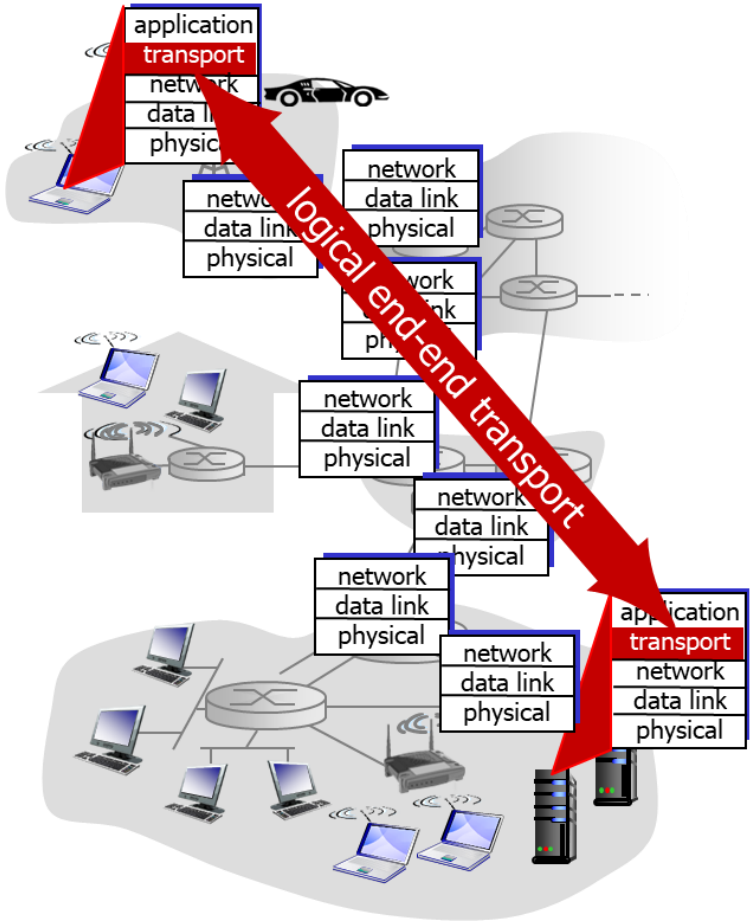
household analogy:

12 kids in house X sending letters to 12 kids in house Y:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

Internet Transport-Layer Protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



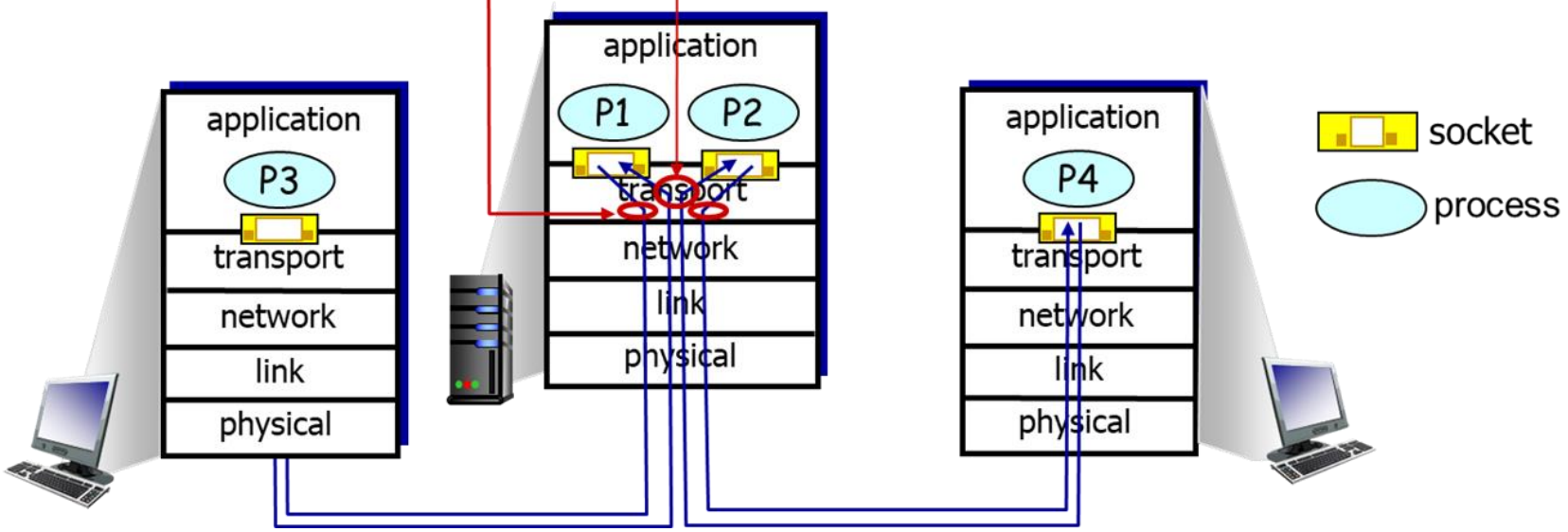
Multiplexing/Demultiplexing

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

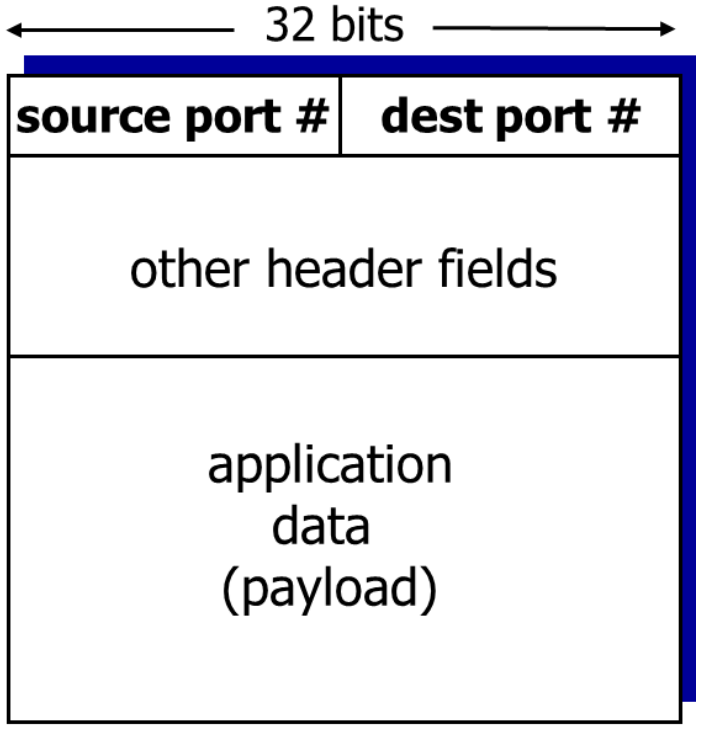


How Demultiplexing Works

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

Connectionless Demultiplexing

- **recall:** created socket has host-local port #:

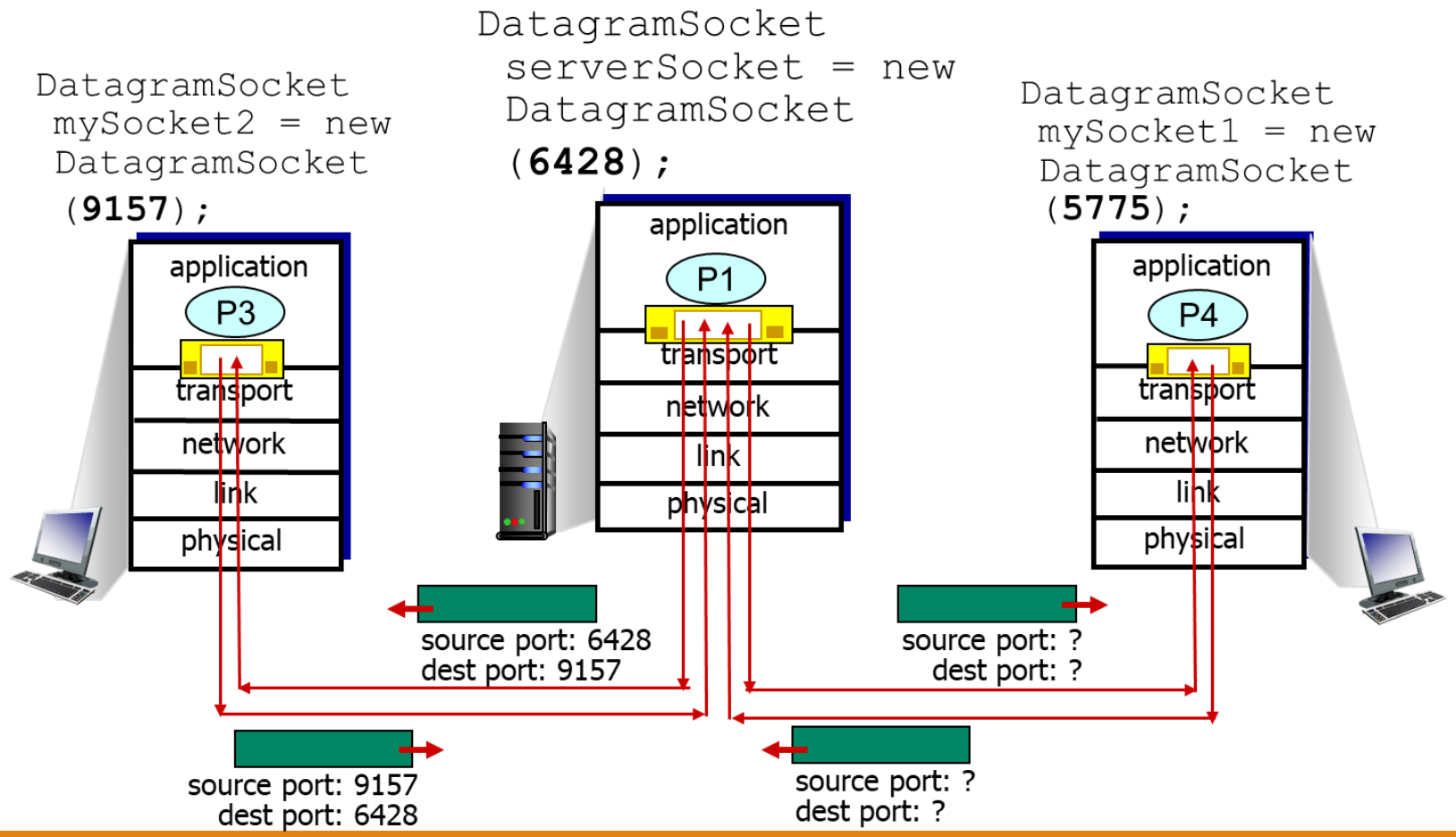
```
DatagramSocket mySocket1
= new DatagramSocket(12534);
```

- **recall:** when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

- when host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

IP datagrams with **same dest. port #,** but different source IP addresses and/or source port numbers will be directed to **same socket** at dest

Connectionless Demux: Example



Connection-Oriented Demux

TCP socket identified by 4-tuple:

- **source IP address**
- **source port number**
- **dest IP address**
- **dest port number**

demux: receiver uses all four values to direct segment to appropriate socket

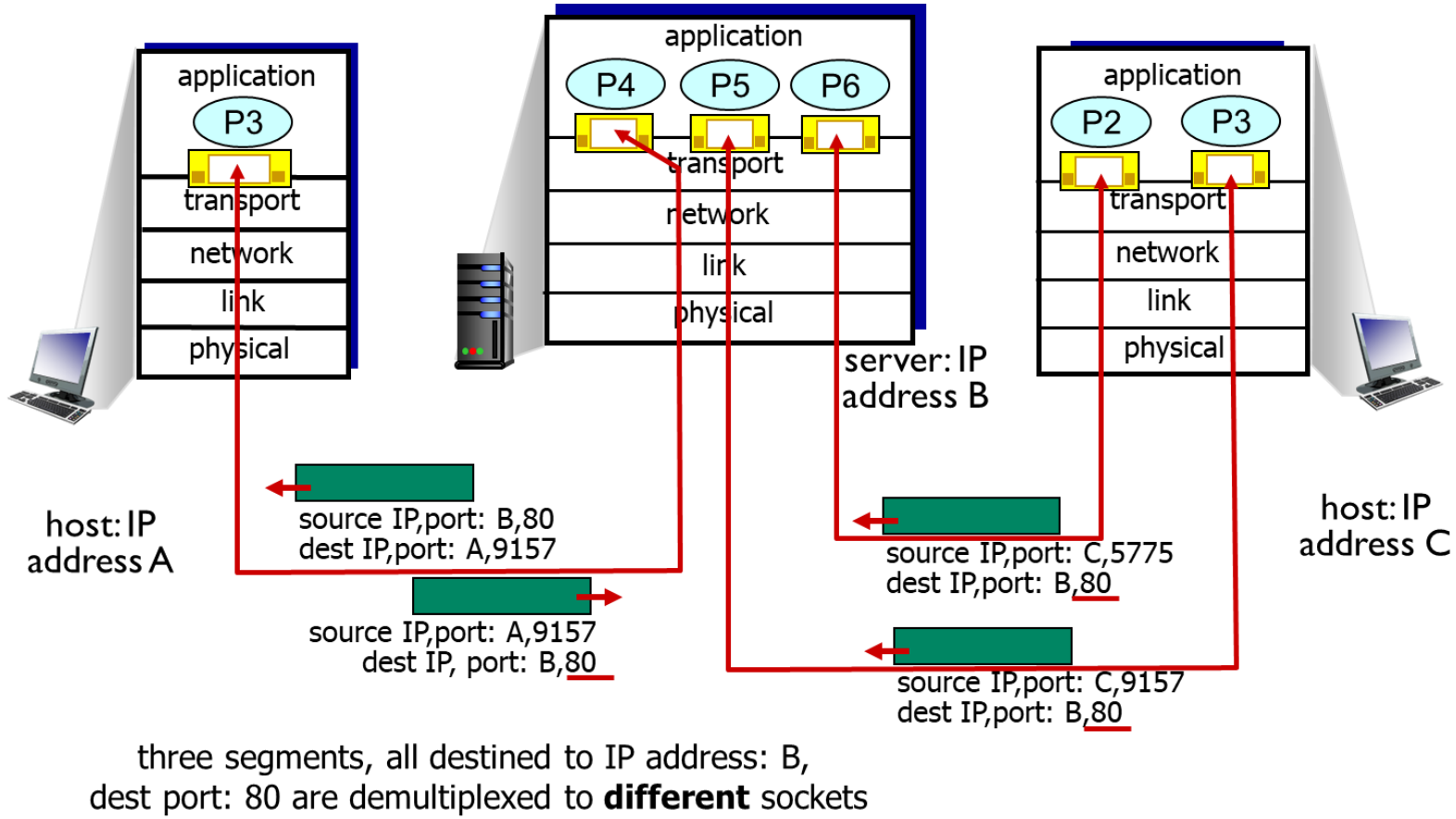
server host may support many simultaneous TCP sockets:

- each socket identified by its own 4-tuple

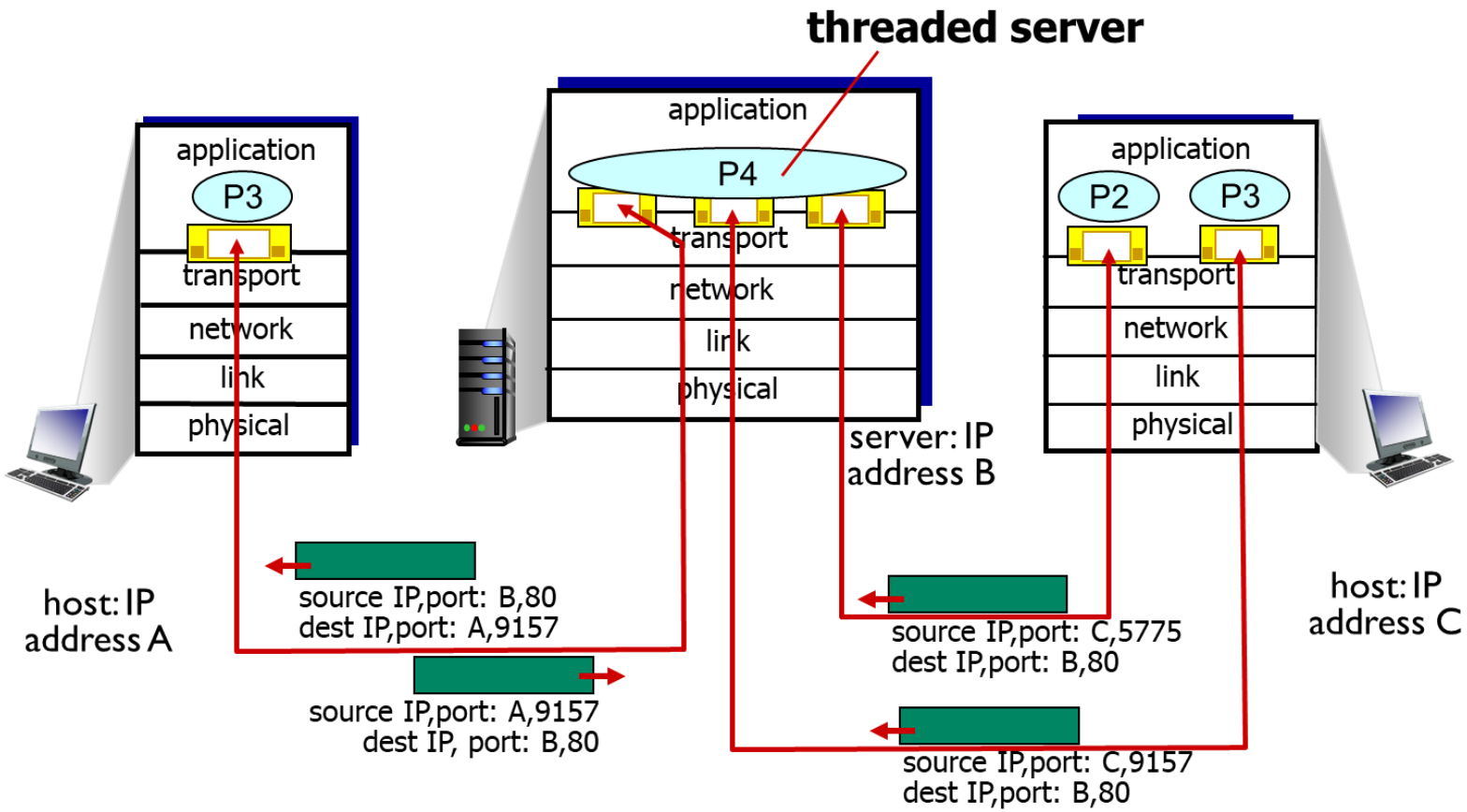
web servers have different sockets for each connecting client

- non-persistent HTTP will have different socket for each request

Connection-Oriented Demux: Example (1 of 2)



Connection-Oriented Demux: Example (2 of 2)

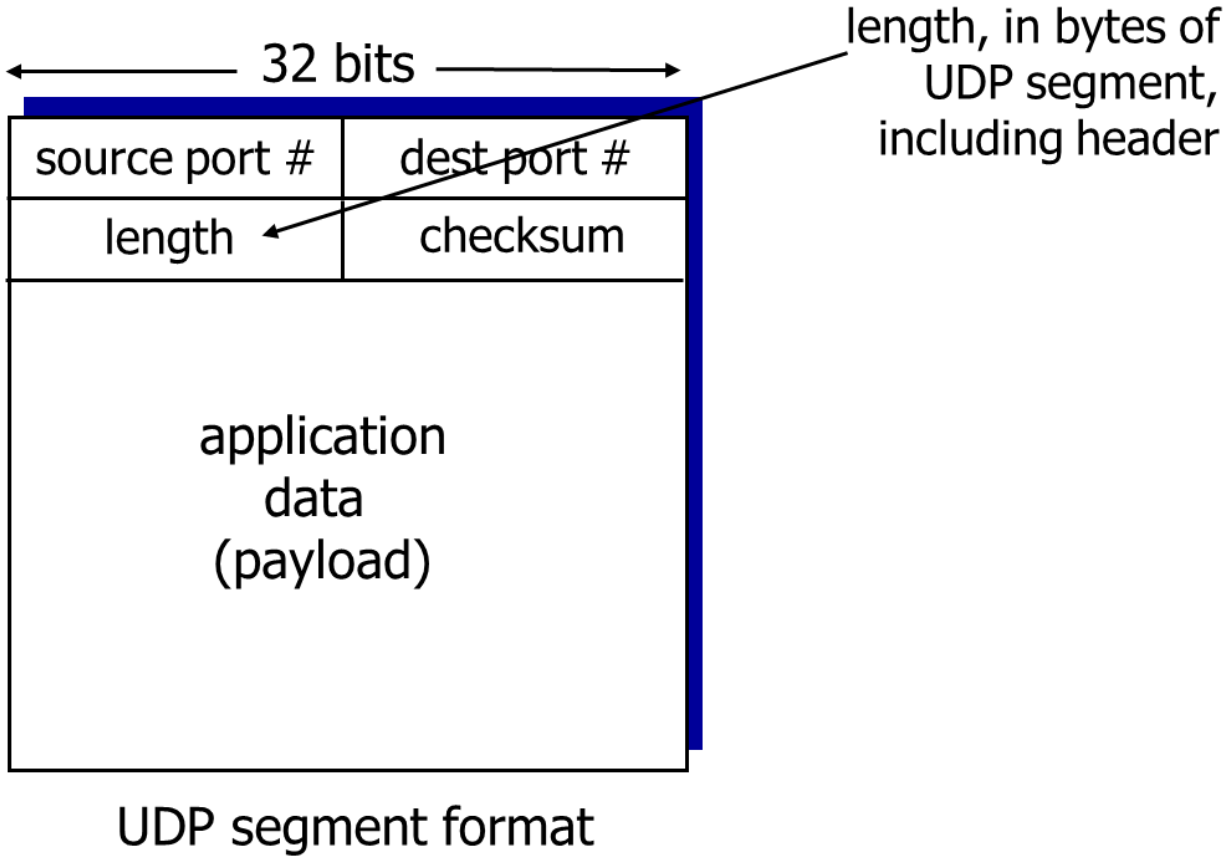


UDP: User Datagram Protocol [RFC 768]

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

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP: Segment Header (1 of 2)



UDP segment format

UDP Checksum

Goal*:* detect "errors" (example, flipped bits) in transmitted segment

sender:

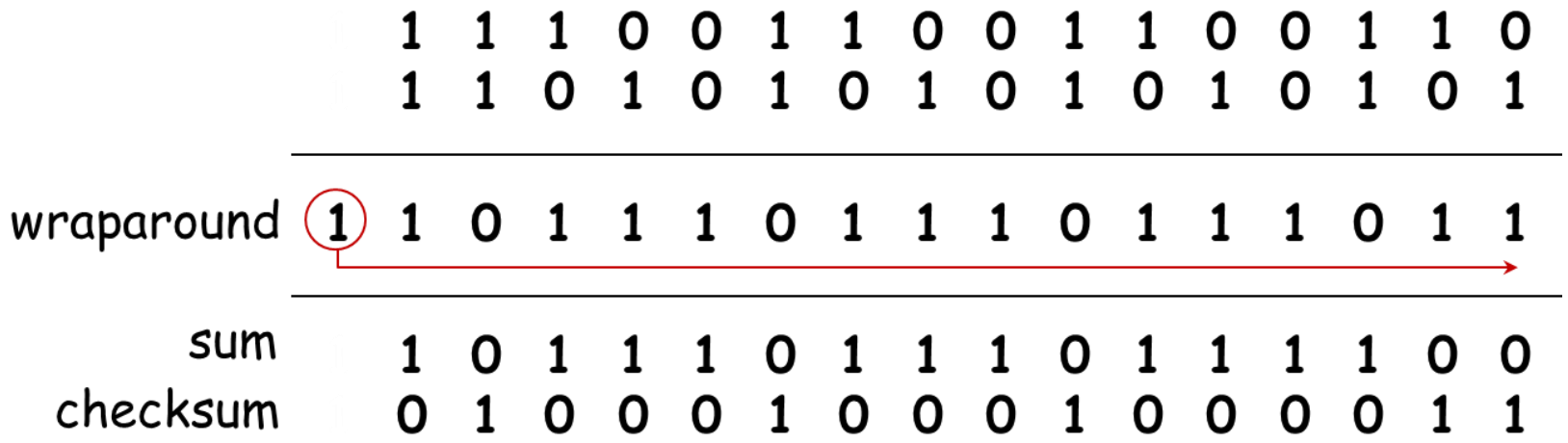
- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

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

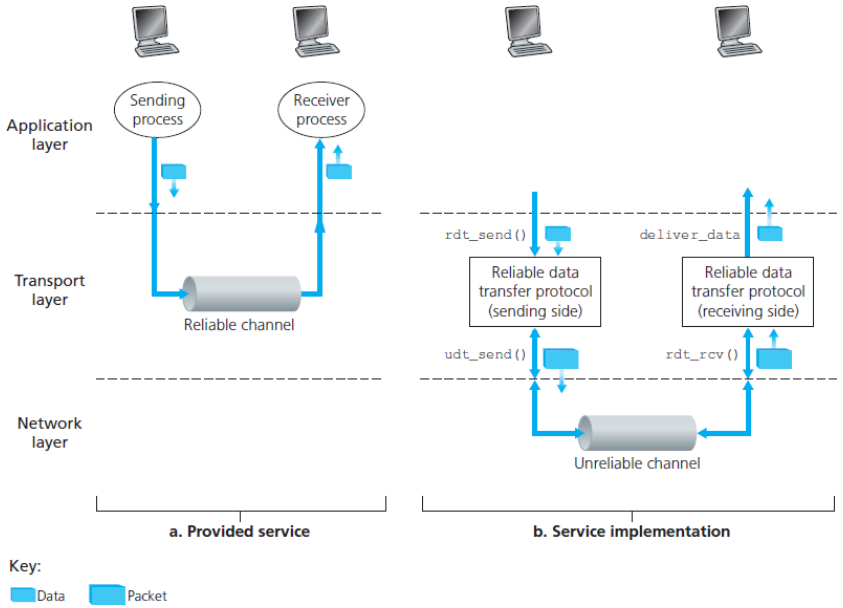
example: add two 16-bit integers



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

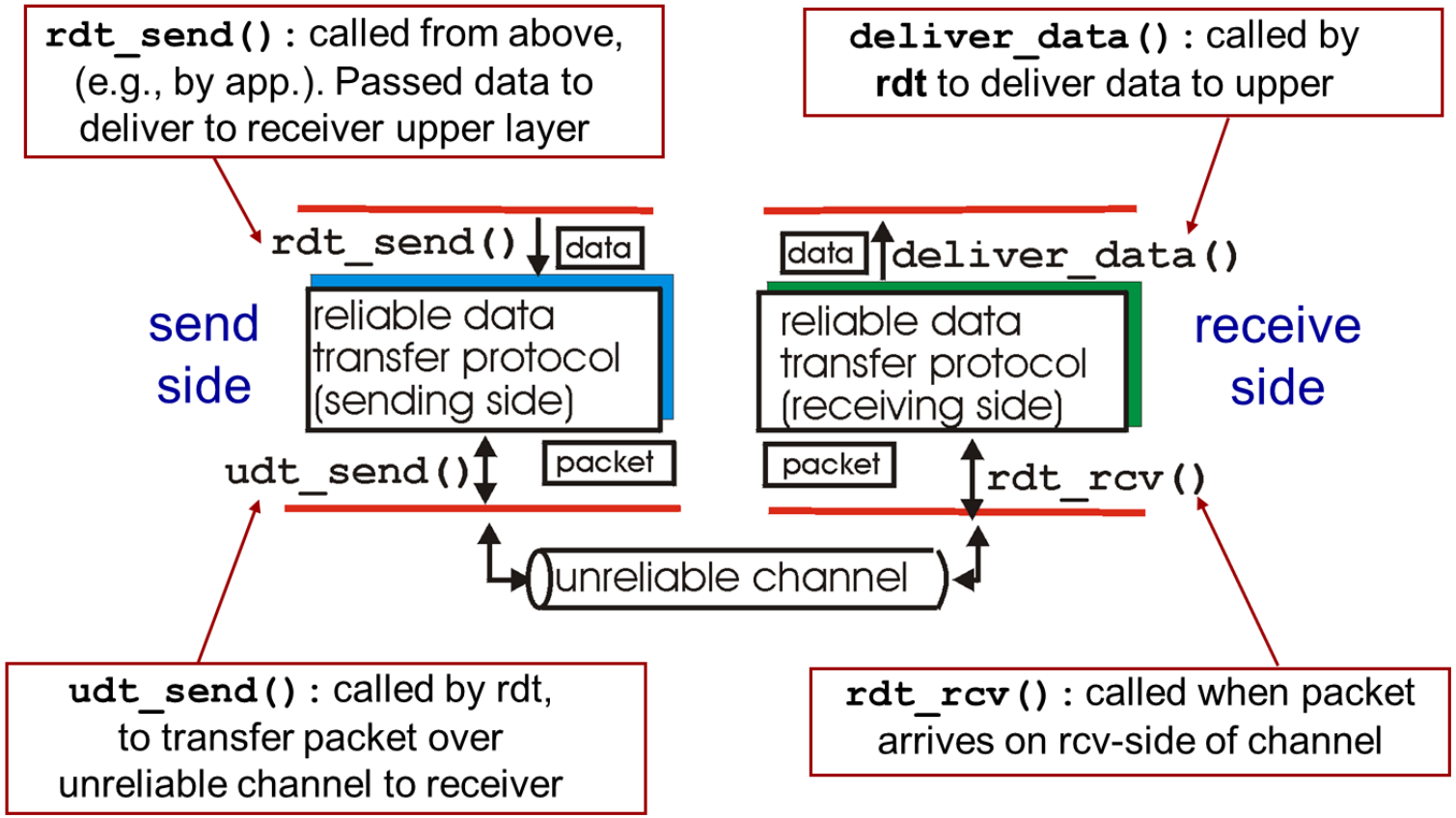
Principles of Reliable Data Transfer

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



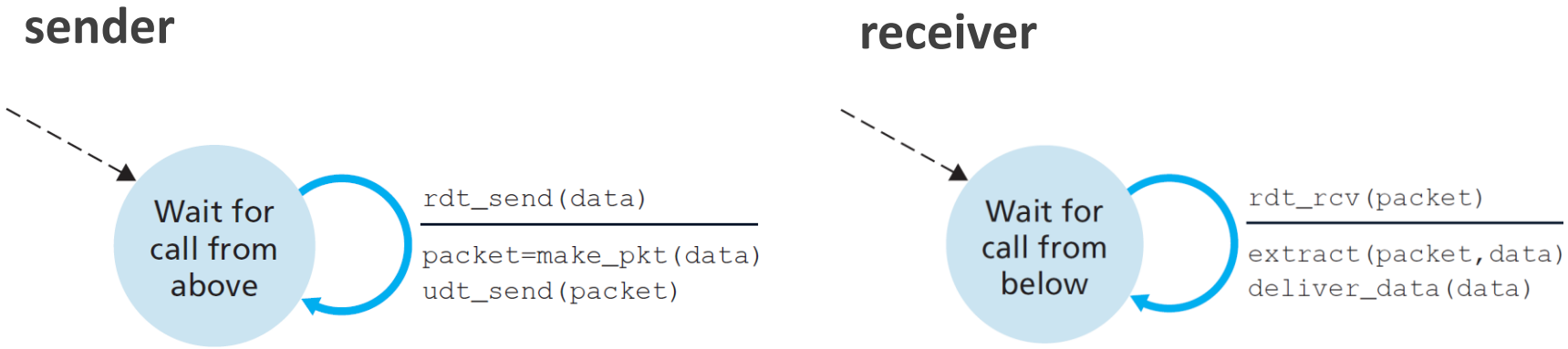
- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable Data Transfer: Getting Started (1 of 2)



rdt1.0: Reliable Transfer over a Reliable Channel

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



rdt2.0: Channel with Bit Errors (1 of 2)

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- **the** question: how to recover from errors:

How do humans recover from "errors" during conversation?

rdt2.0: Channel with Bit Errors (2 of 2)

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

rdt2.0 Has a Fatal Flaw!

what happens if ACK / NAK corrupted?

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

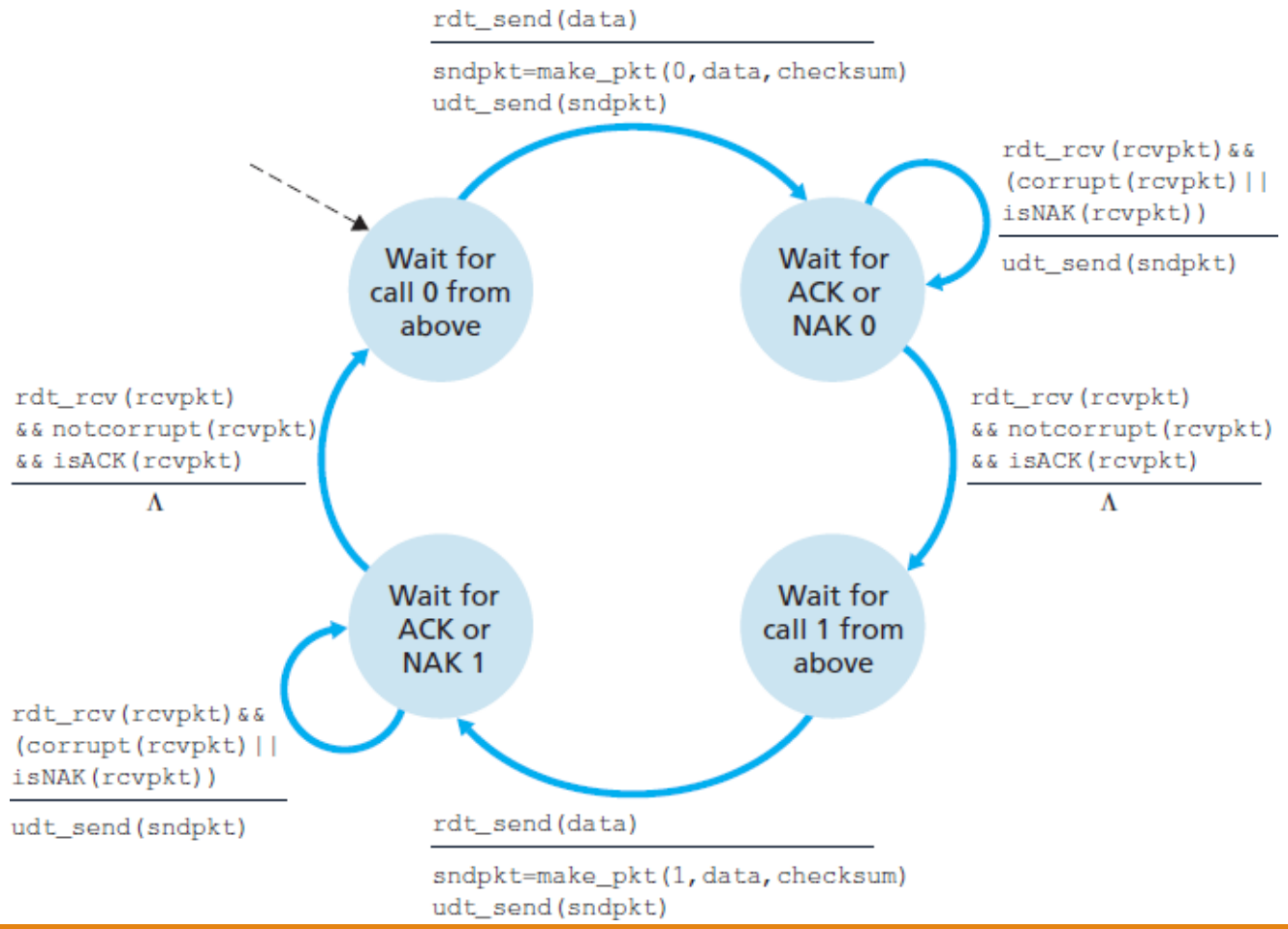
stop and wait

sender sends one packet, then waits for receiver response

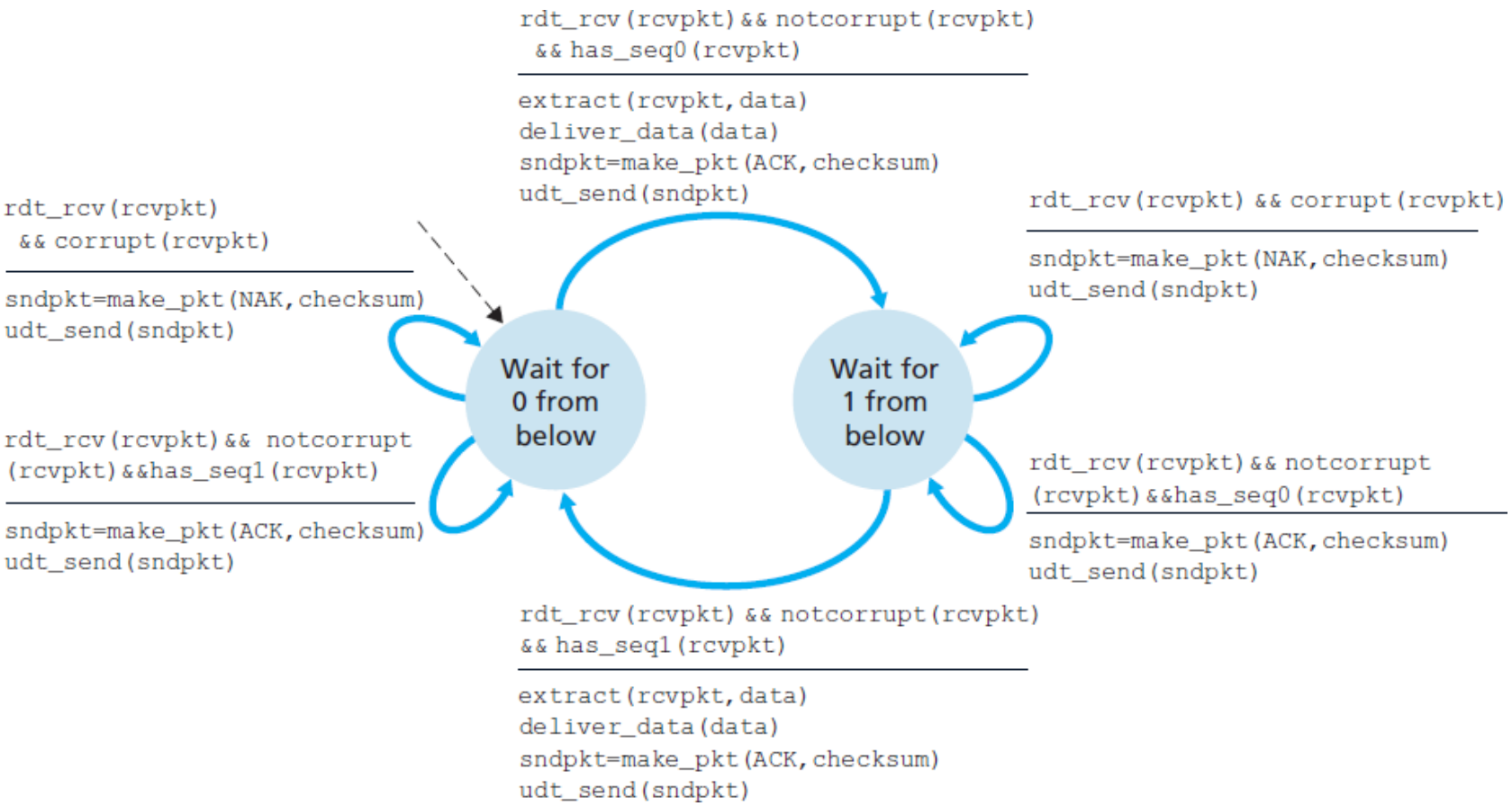
handling duplicates:

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

rdt2.1: Sender, Handles *Garbled* ACK/NAK s



rdt2.1: Receiver, Handles Garbled ACK/NAKs



rdt2.1: Discussion

sender:

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

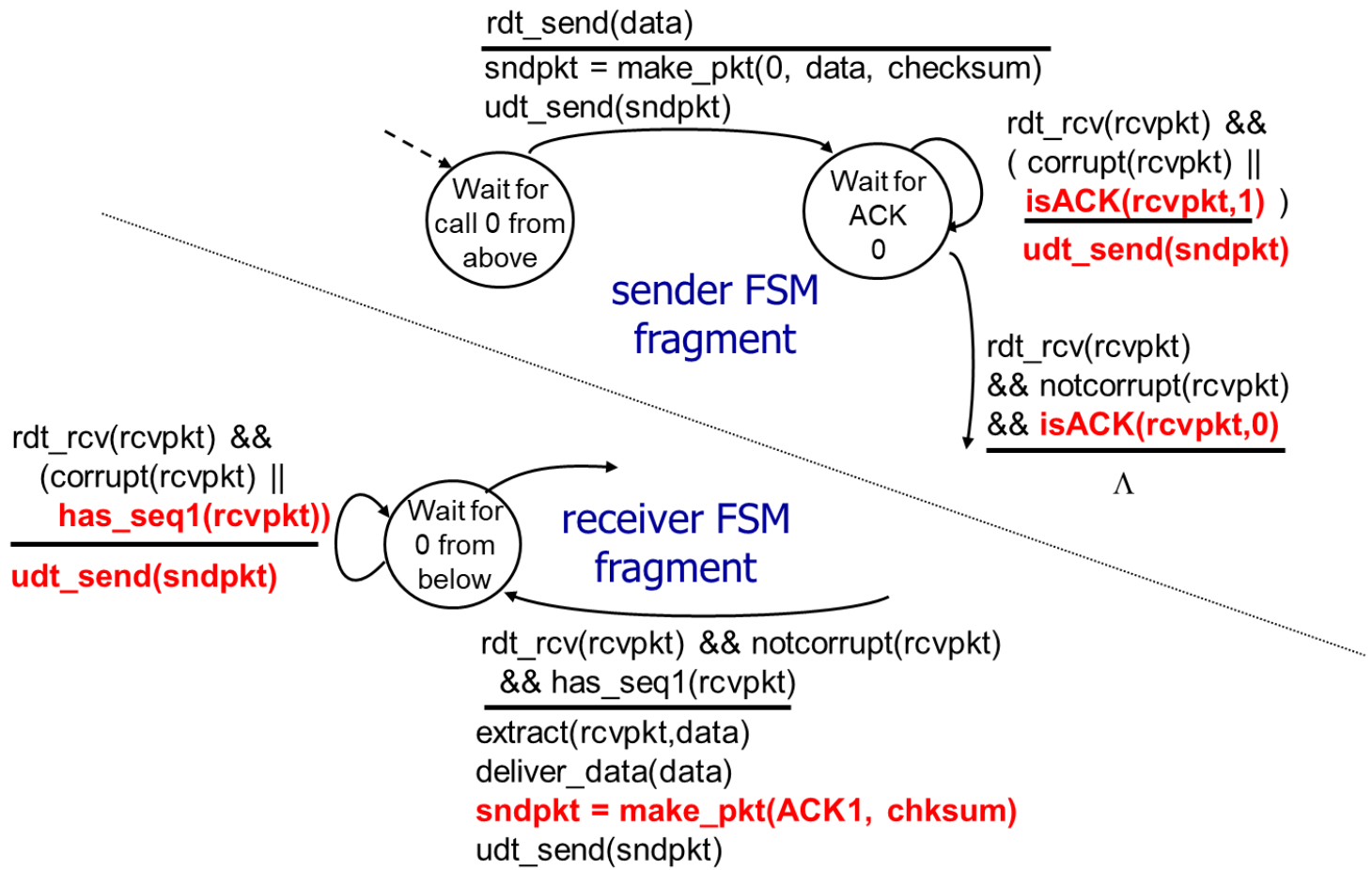
receiver:

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

rdt2.2: A NAK - free Protocol

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

rdt2.2: Sender, Receiver Fragments



rdt3.0: Channels with Errors and Loss

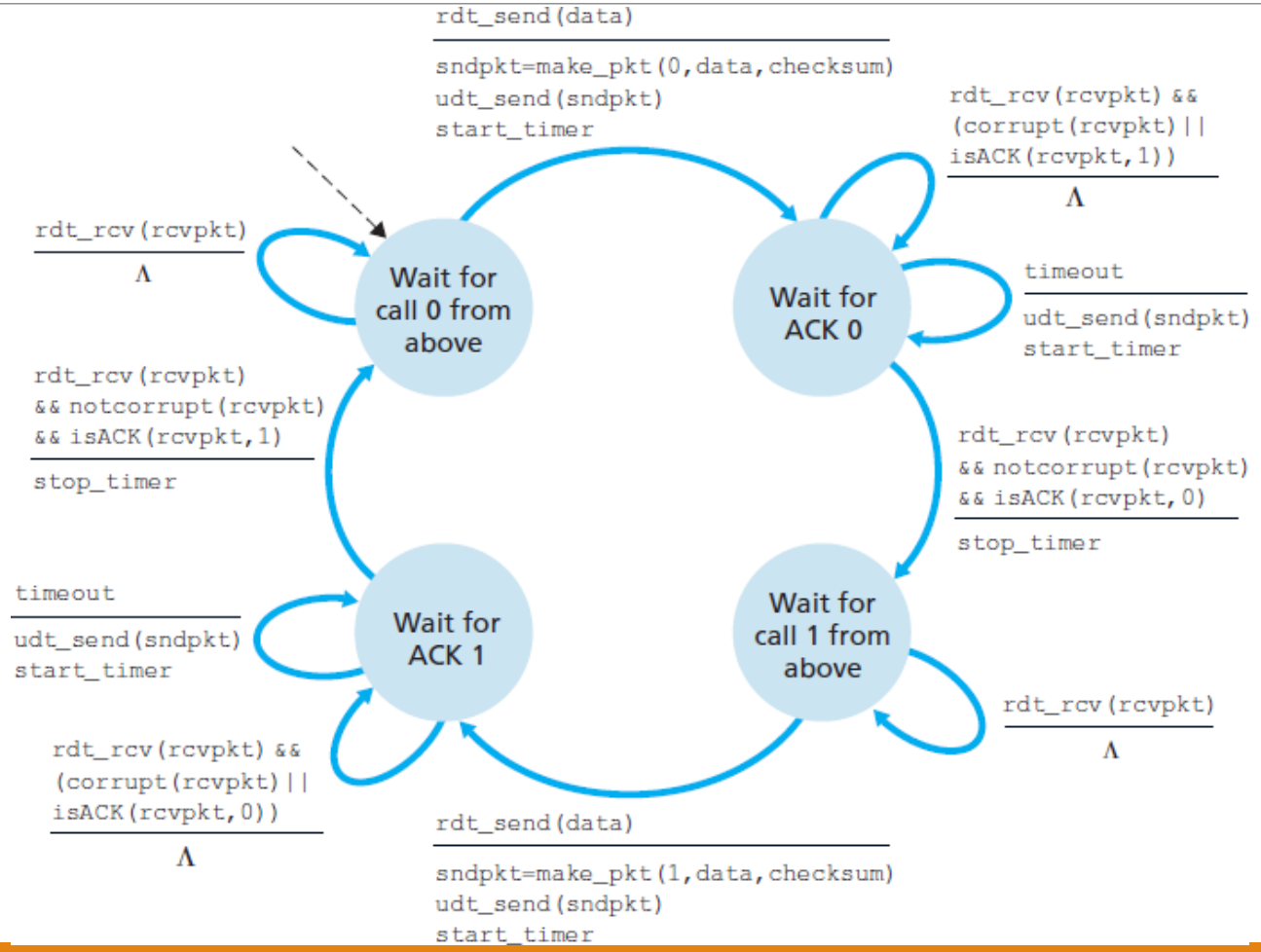
new assumption: underlying channel can also lose packets (data, ACKs)

- checksum, Sequence #, ACKs, retransmissions will be of help … but not enough

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

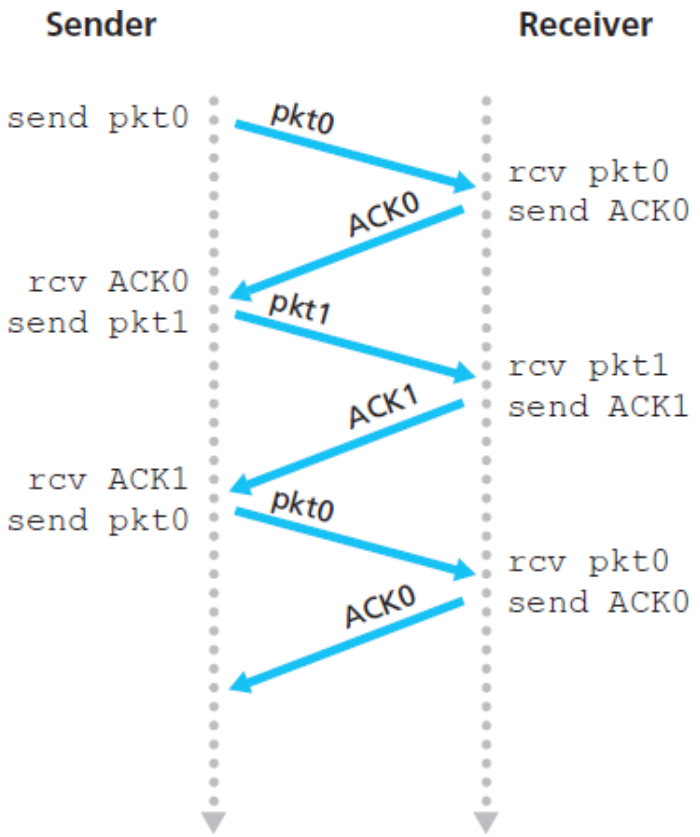
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but Sequence #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

rdt3.0 Sender

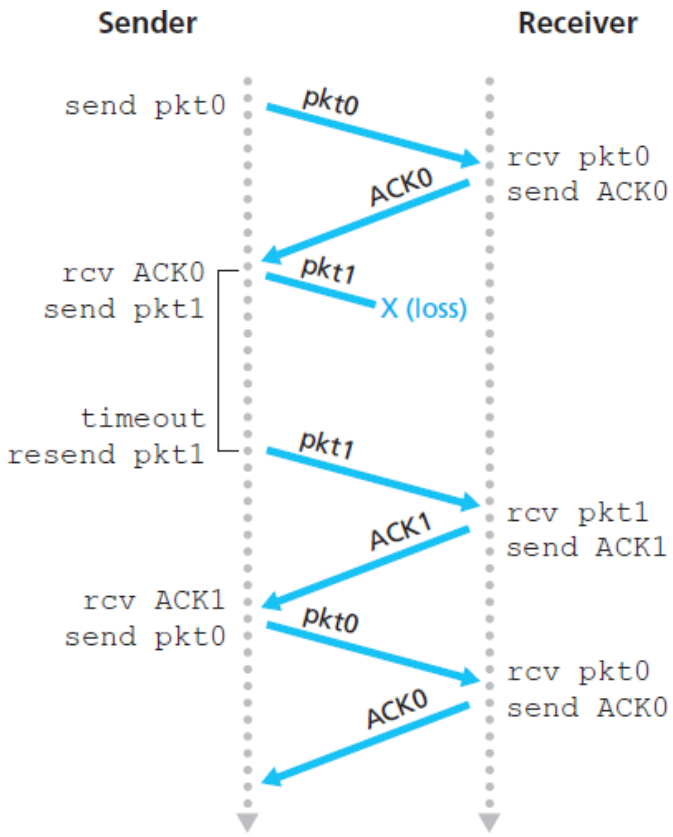


rdt3.0 in Action (1 of 2)

(a) no loss

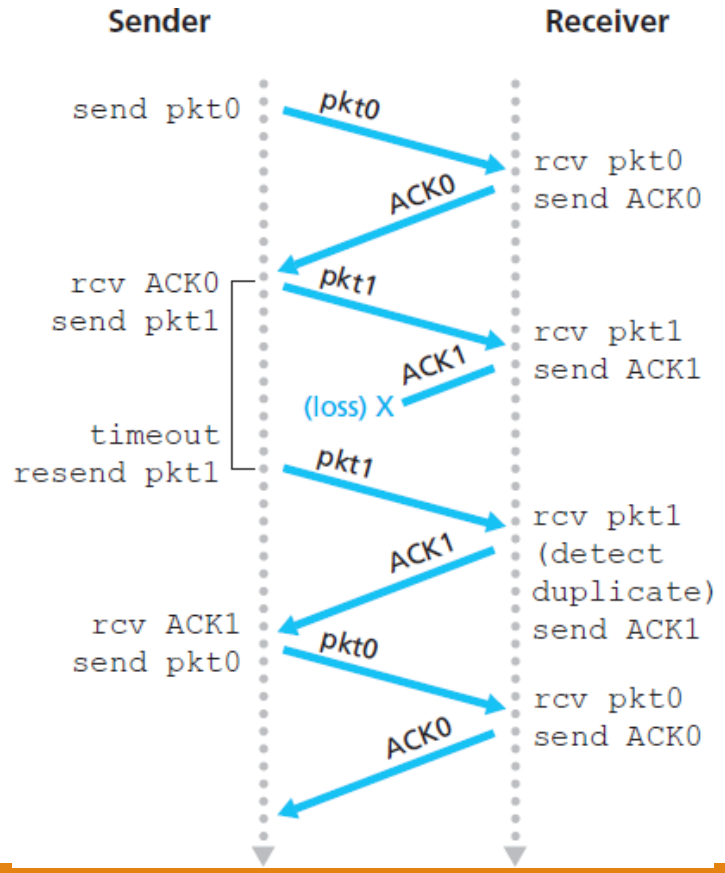


(b) packet loss

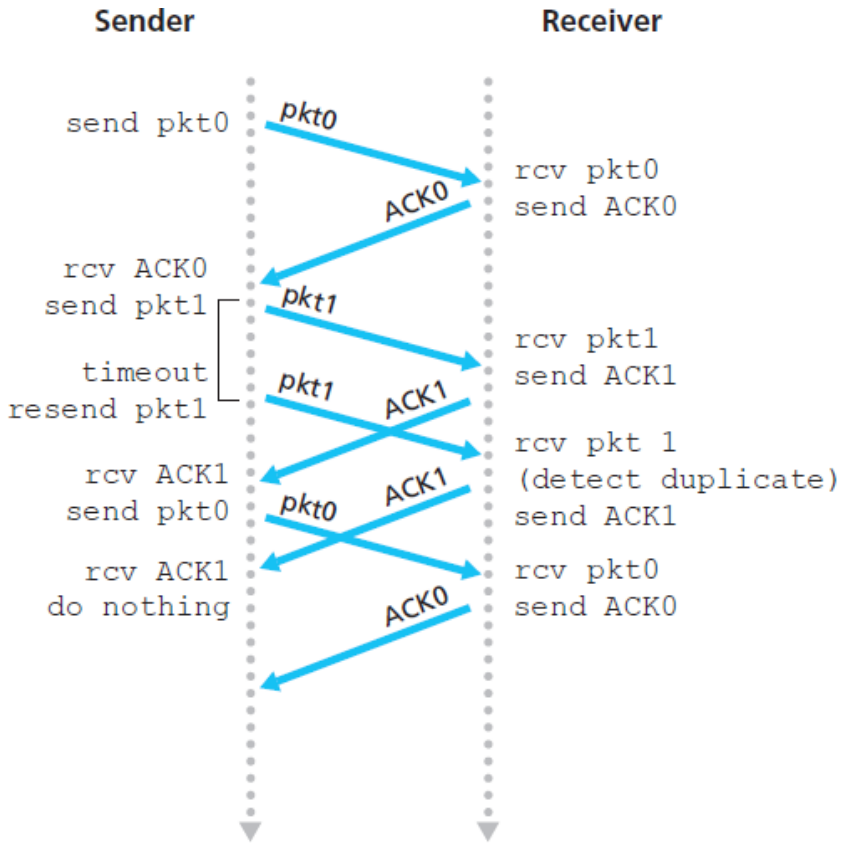


rdt3.0 in Action (2 of 2)

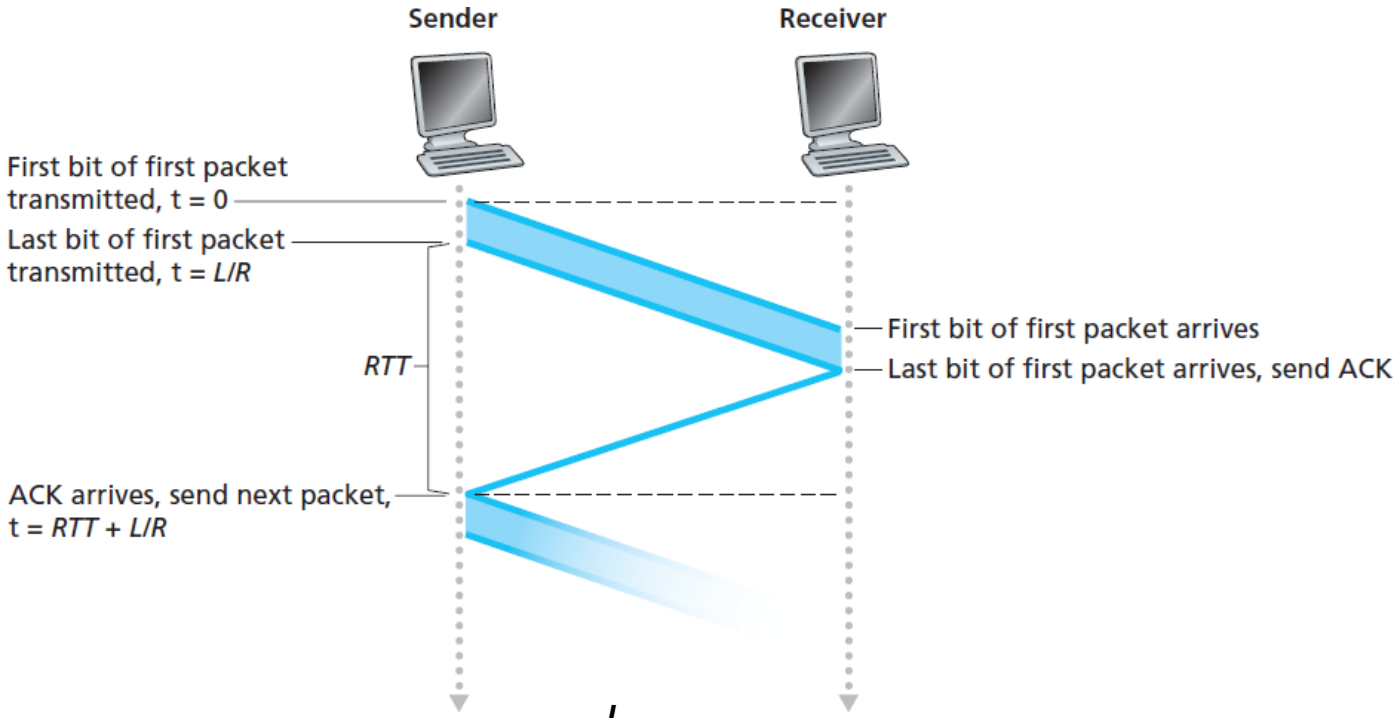
(c) ACK loss



(d) premature timeout/ delayed ACK



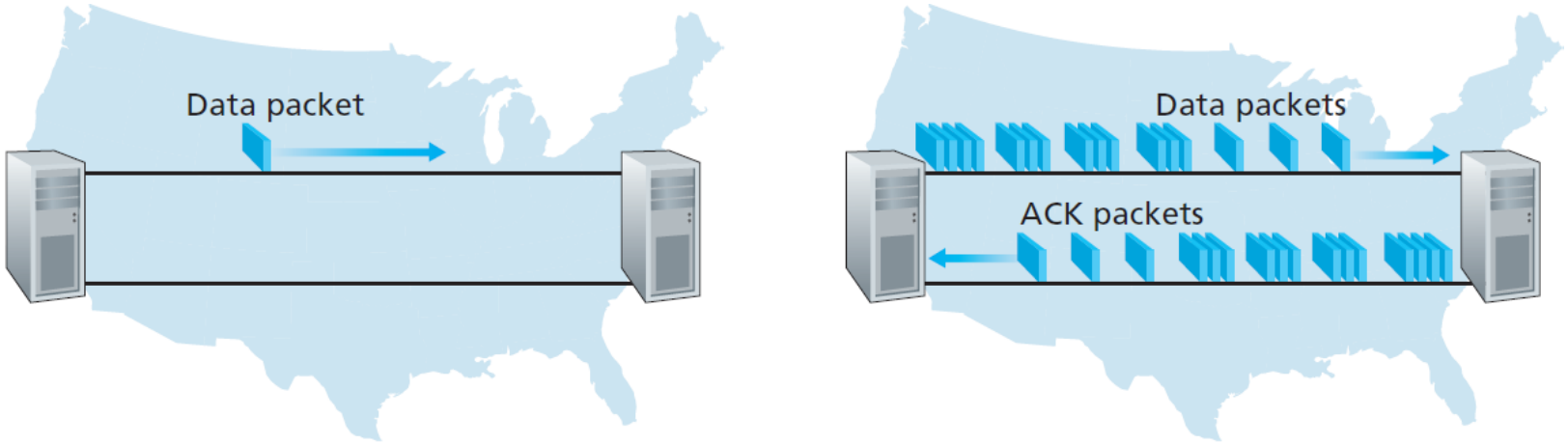
rdt3.0: Stop-and-wait Operation



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

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

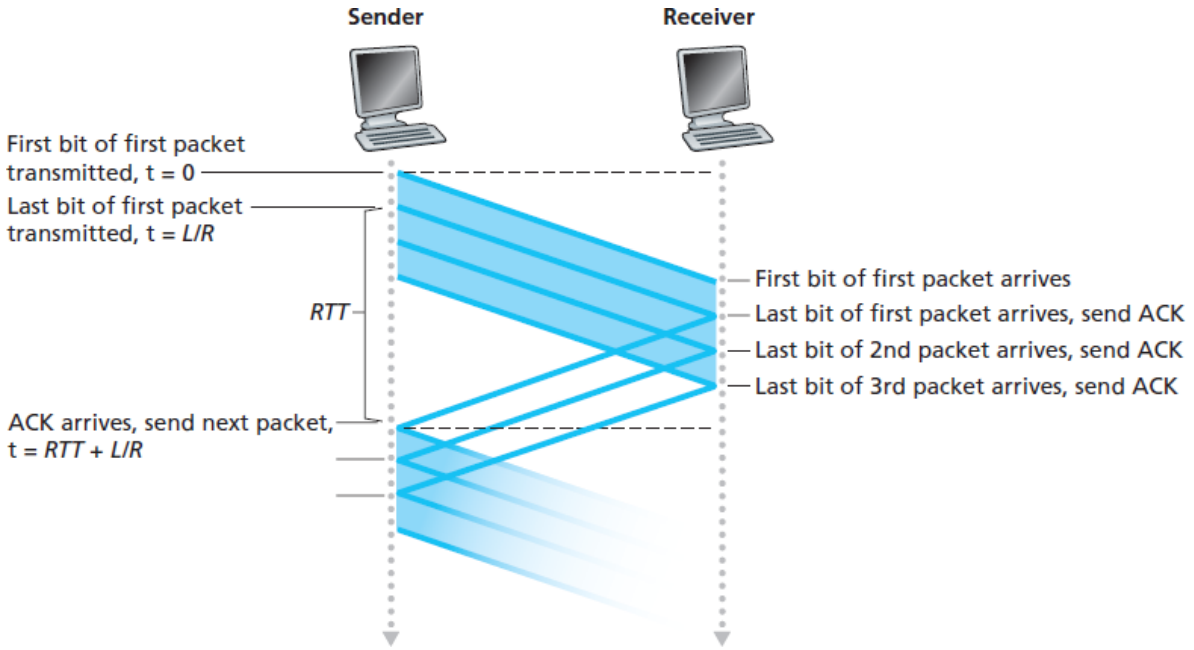


a. A stop-and-wait protocol in operation

b. A pipelined protocol in operation

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

Pipelining: Increased Utilization

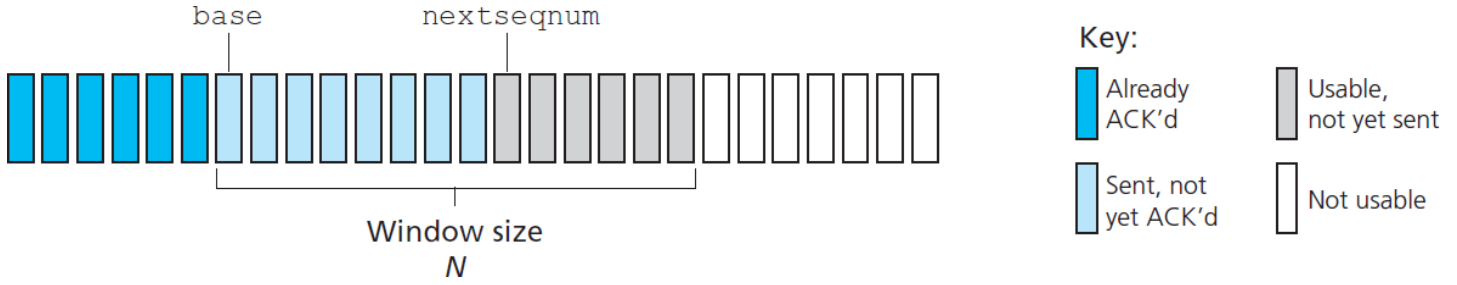


$$U_{\text{sender}} = \frac{\frac{3L}{R}}{RTT + \frac{L}{R}} = \frac{.0024}{30.008} = 0.00081$$

3-packet pipelining increases utilization by a factor of 3!

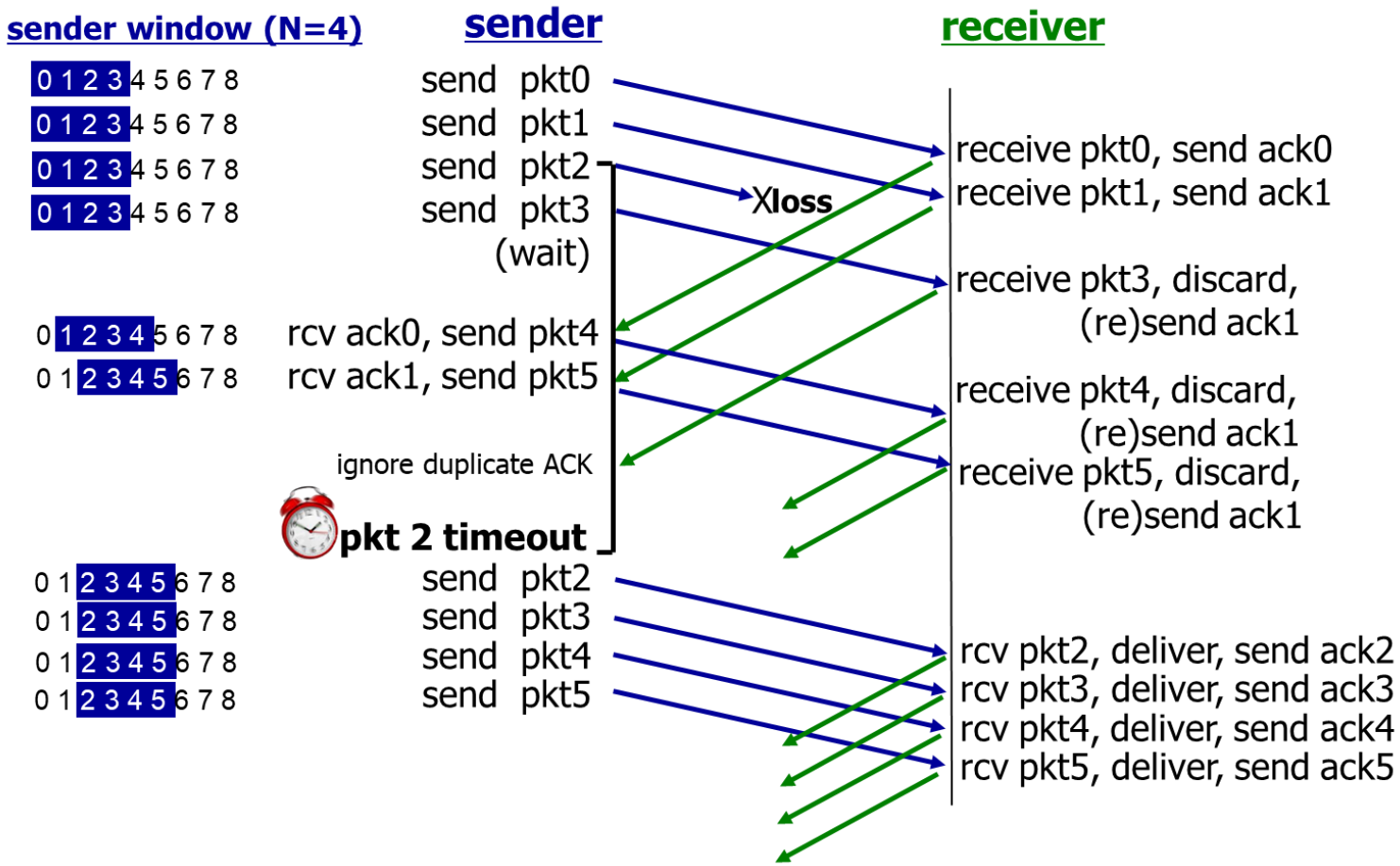
Go-Back-N: Sender

- 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

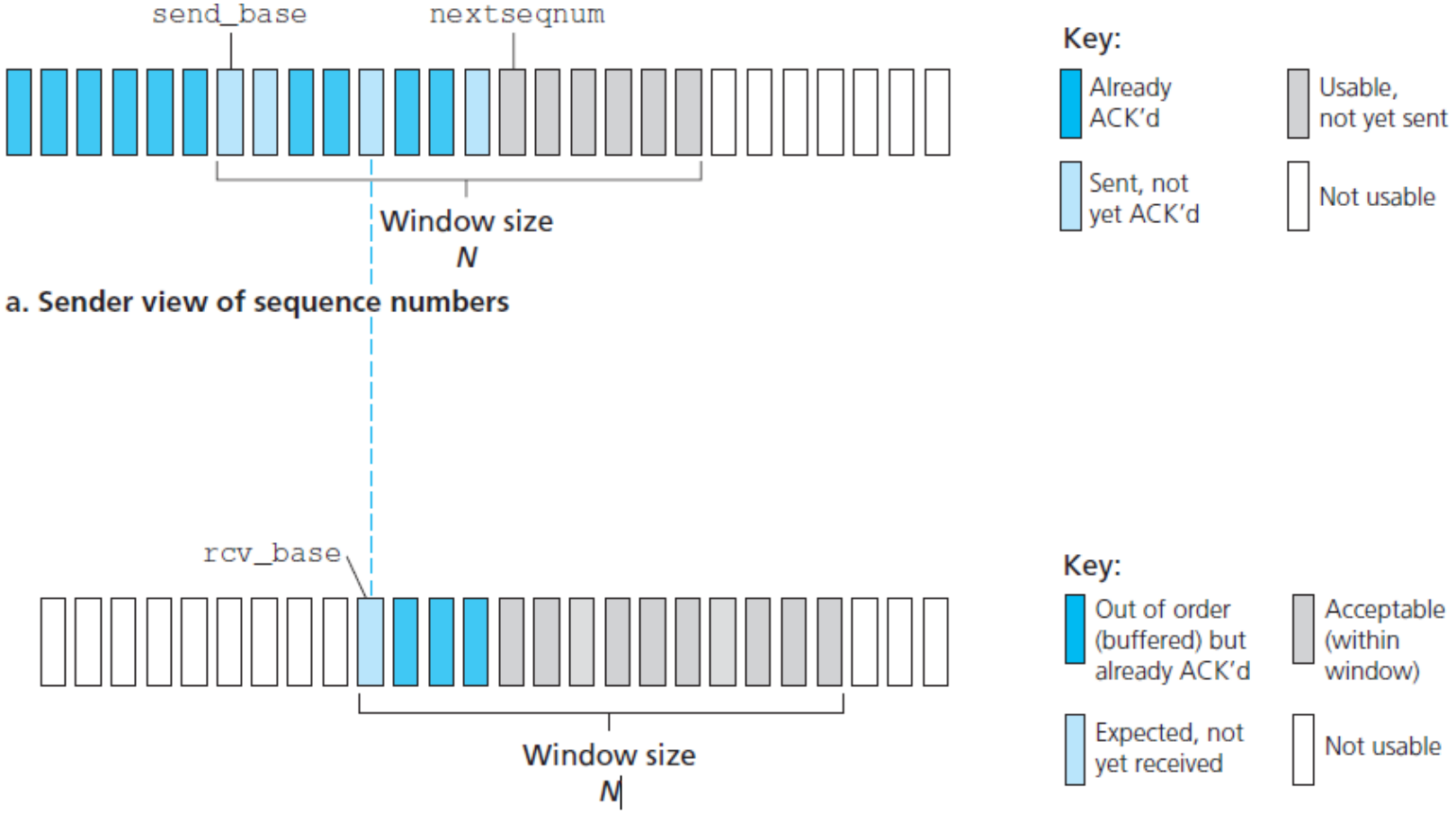
GBN in Action



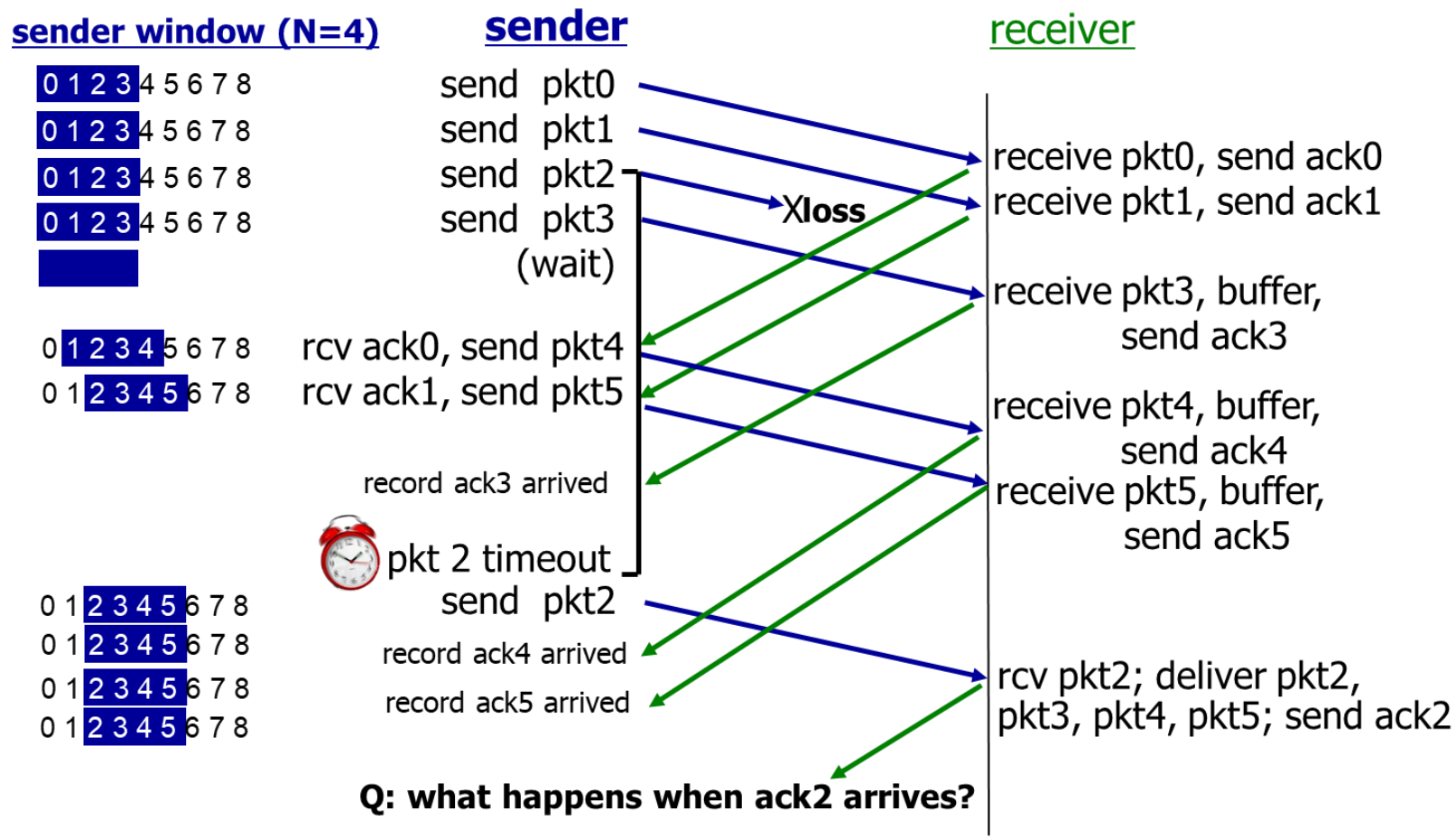
Selective Repeat (1 of 3)

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

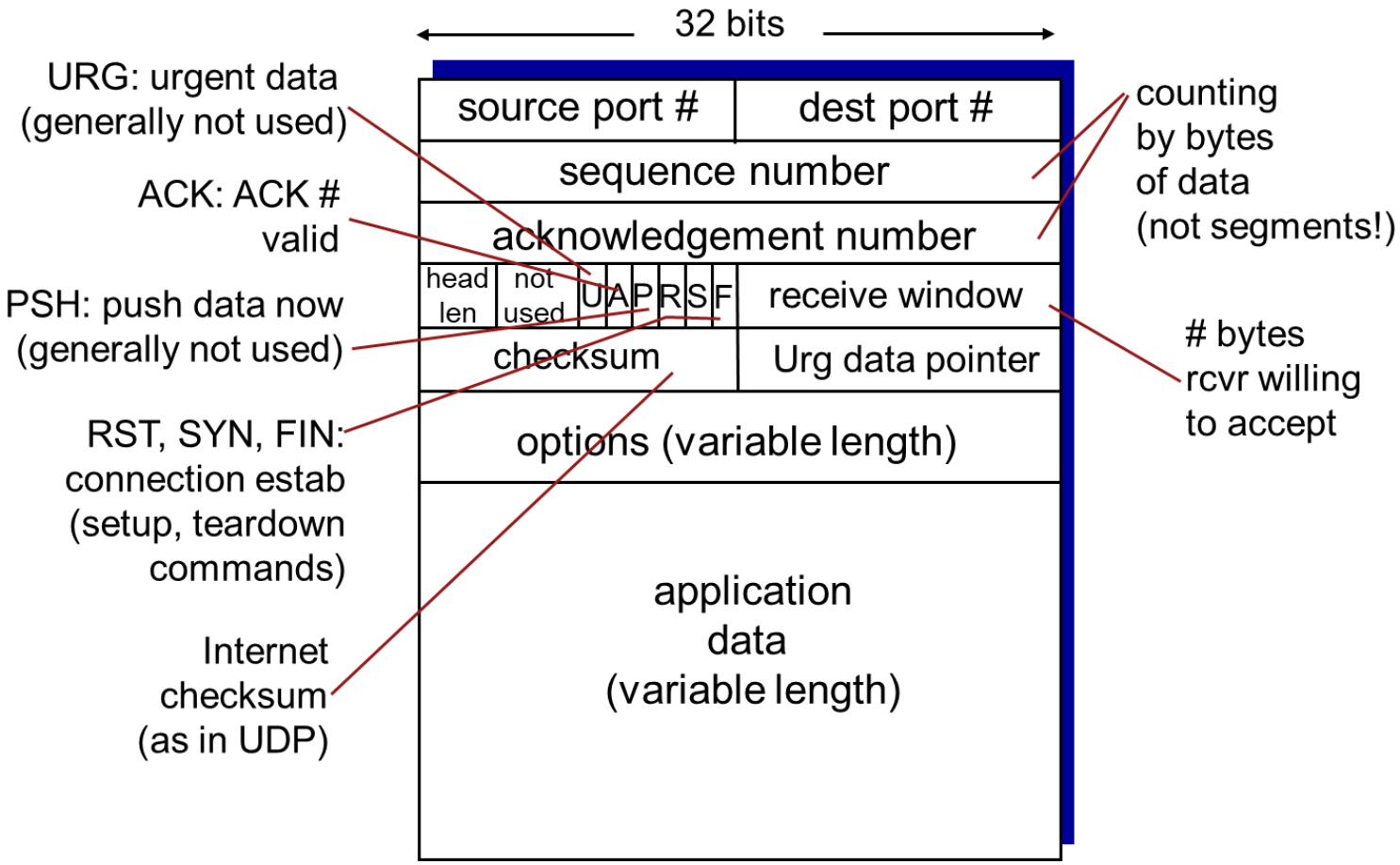
Selective Repeat: Sender, Receiver Windows



Selective Repeat in Action



TCP Segment Structure



TCP Sequence Numbers, ACKs (1 of 2)

sequence numbers:

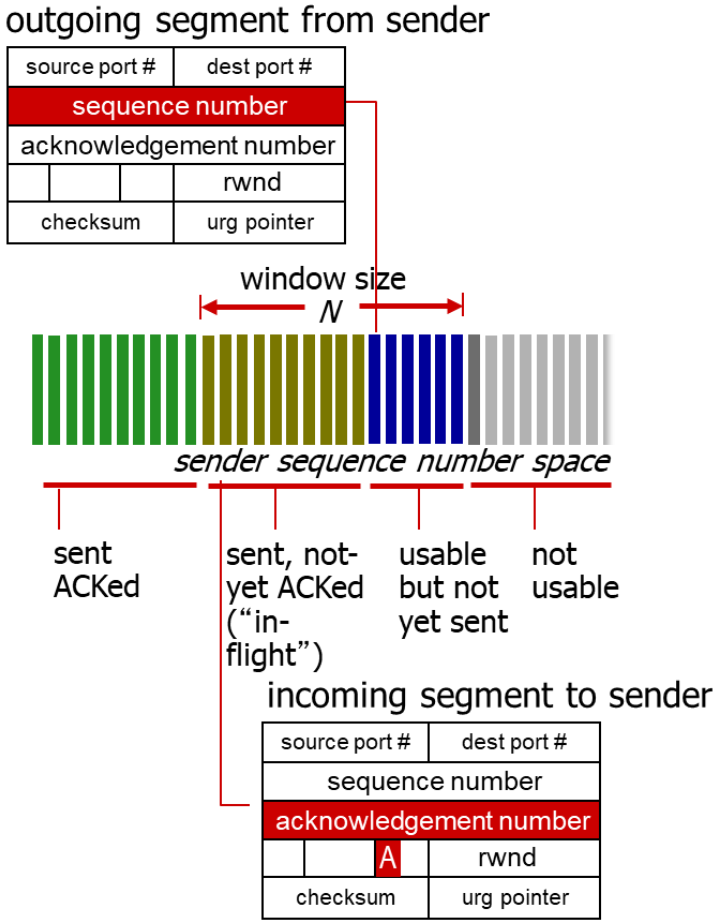
- byte stream "number" of first byte in segment's data

acknowledgements:

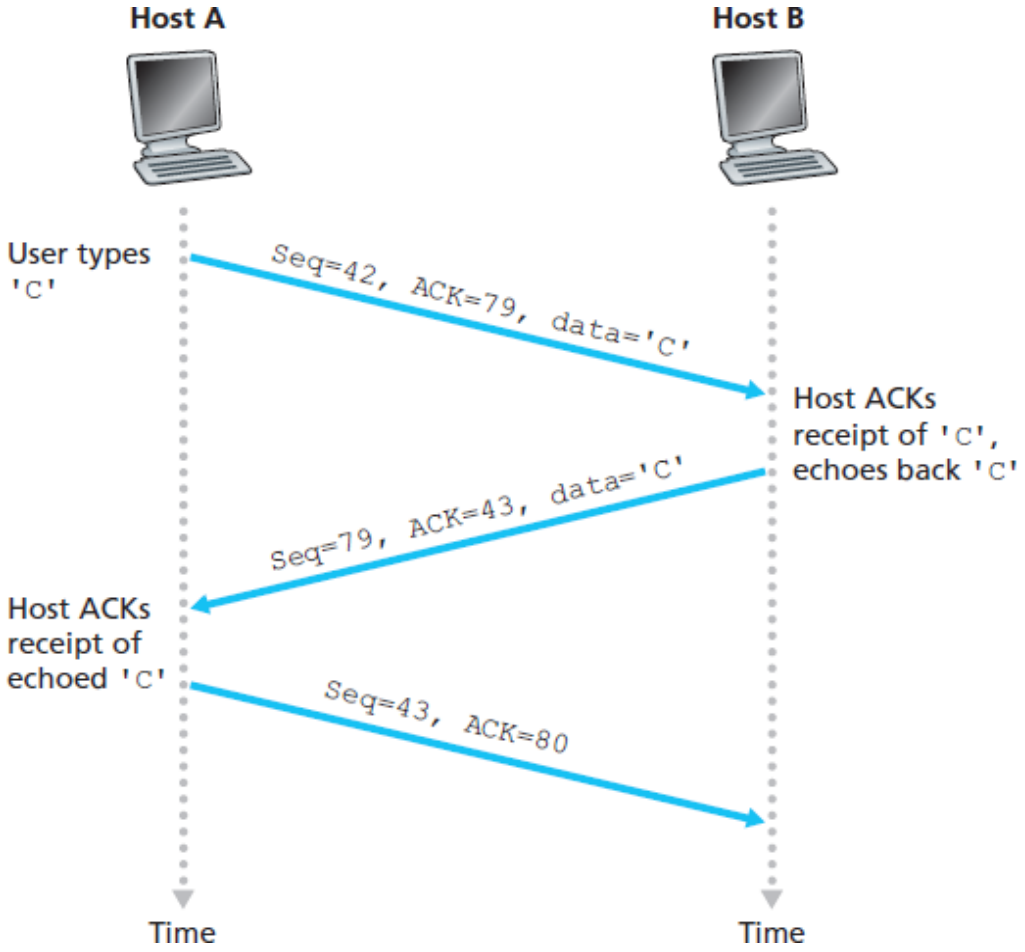
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

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



TCP Sequence Numbers, ACKs (2 of 2)



TCP Round Trip Time, Timeout (1 of 3)

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

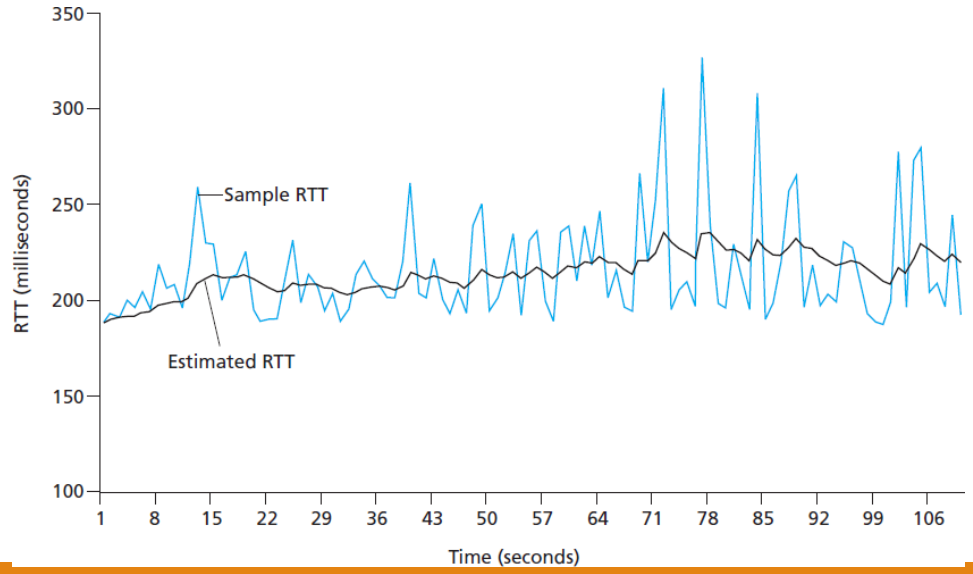
Q: how to estimate RTT?

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

TCP Round Trip Time, Timeout (2 of 3)

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

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



TCP Round Trip Time, Timeout (3 of 3)

timeout interval: `EstimatedRTT` plus "safety margin"

- large variation in `EstimatedRTT` → larger safety margin.

- estimate SampleRTT deviation from EstimatedRTT:

$$DevRTT = (1-\beta)*DevRTT + \beta*|SampleRTT - EstimatedRTT|$$

(typically, $\beta = 0.25$)

$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

estimated RTT

"safety margin"

TCP Reliable Data Transfer

TCP creates rdt service on top of IP's unreliable service

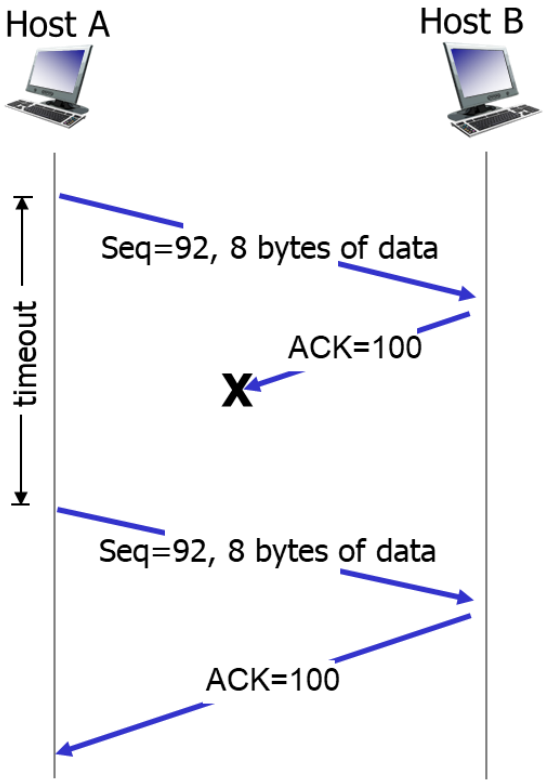
- pipelined segments
- cumulative acks
- single retransmission timer

retransmissions triggered by:

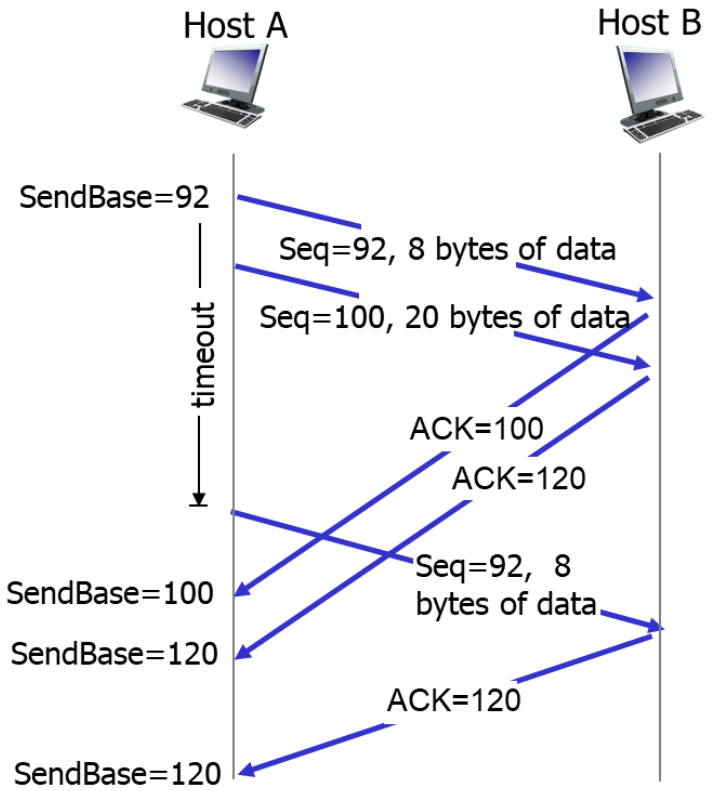
- timeout events
- duplicate acks

- let's initially consider simplified TCP sender:
  - ignore duplicate acks
  - ignore flow control, congestion control

TCP: Retransmission Scenarios (1 of 2)

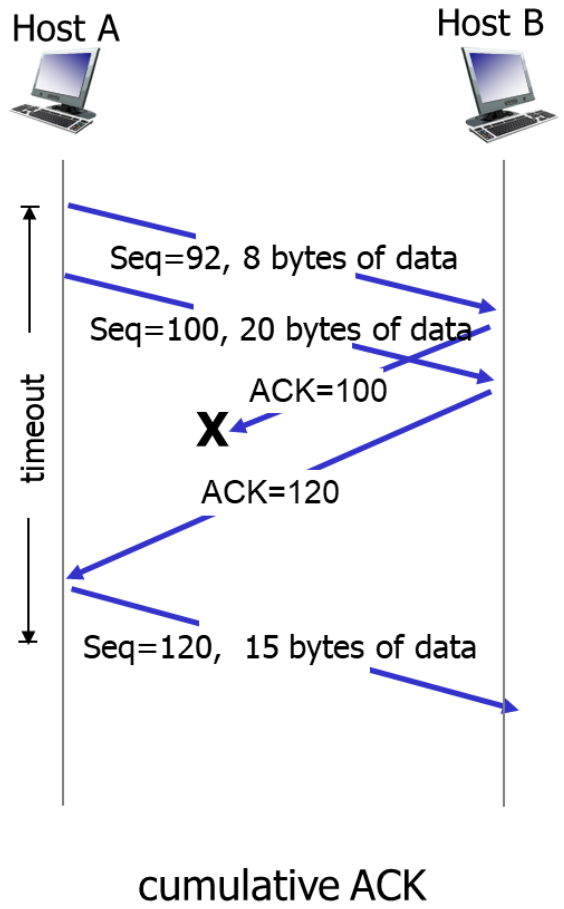


lost ACK scenario



premature timeout

TCP: Retransmission Scenarios (2 of 2)



cumulative ACK

TCP Fast Retransmit (1 of 2)

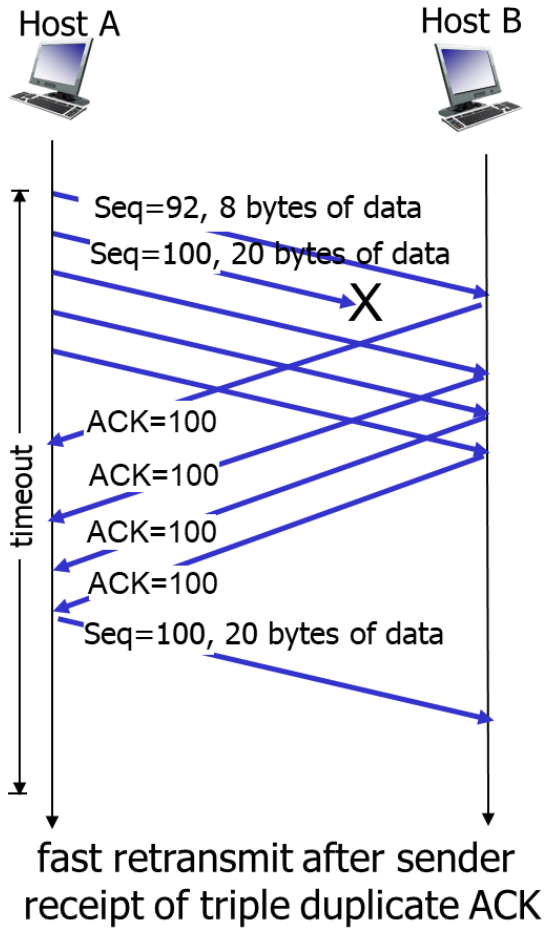
- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

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

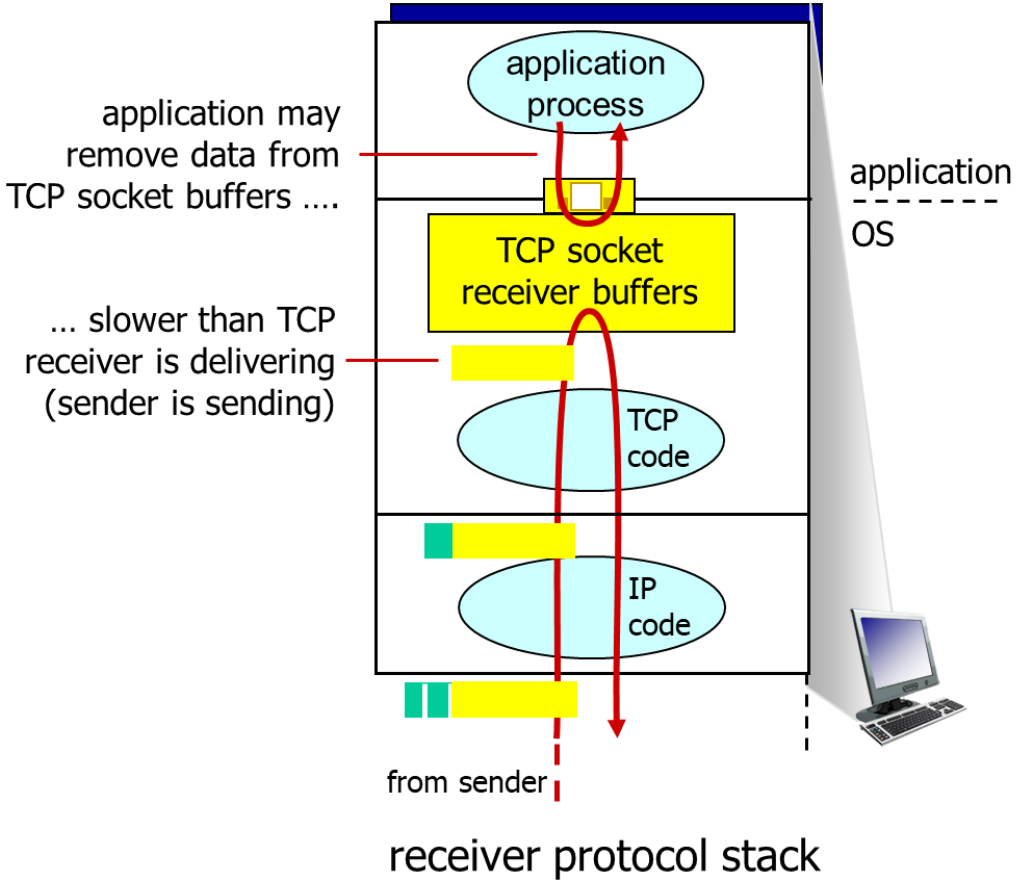
TCP Fast Retransmit (2 of 2)



TCP Flow Control (1 of 2)

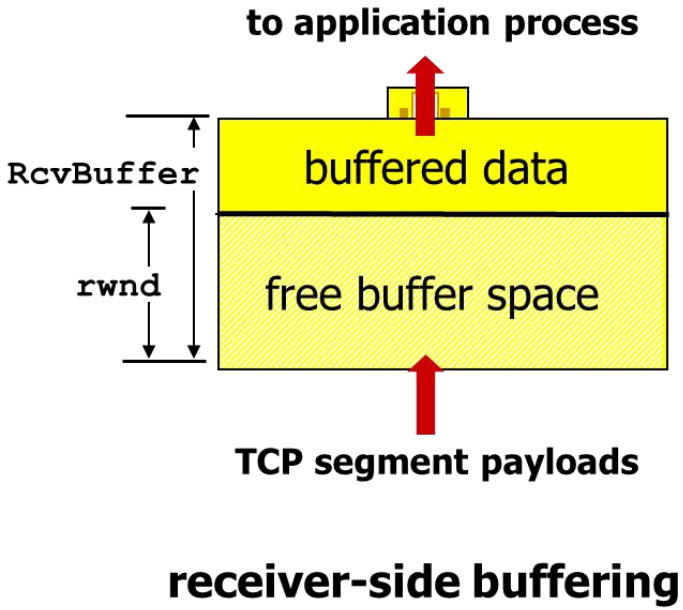
flow control

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



TCP Flow Control (2 of 2)

- receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`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 receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



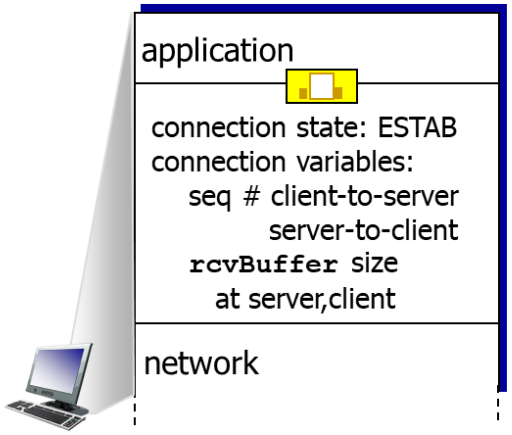
receiver-side buffering

Connection Management

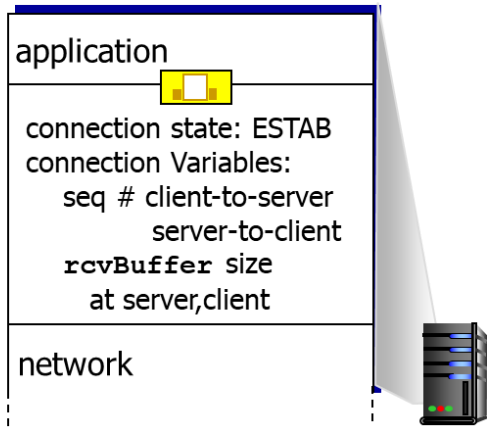
before exchanging data, sender/receiver “handshake”:

agree to establish connection (each knowing the other willing to establish connection)

agree on connection parameters

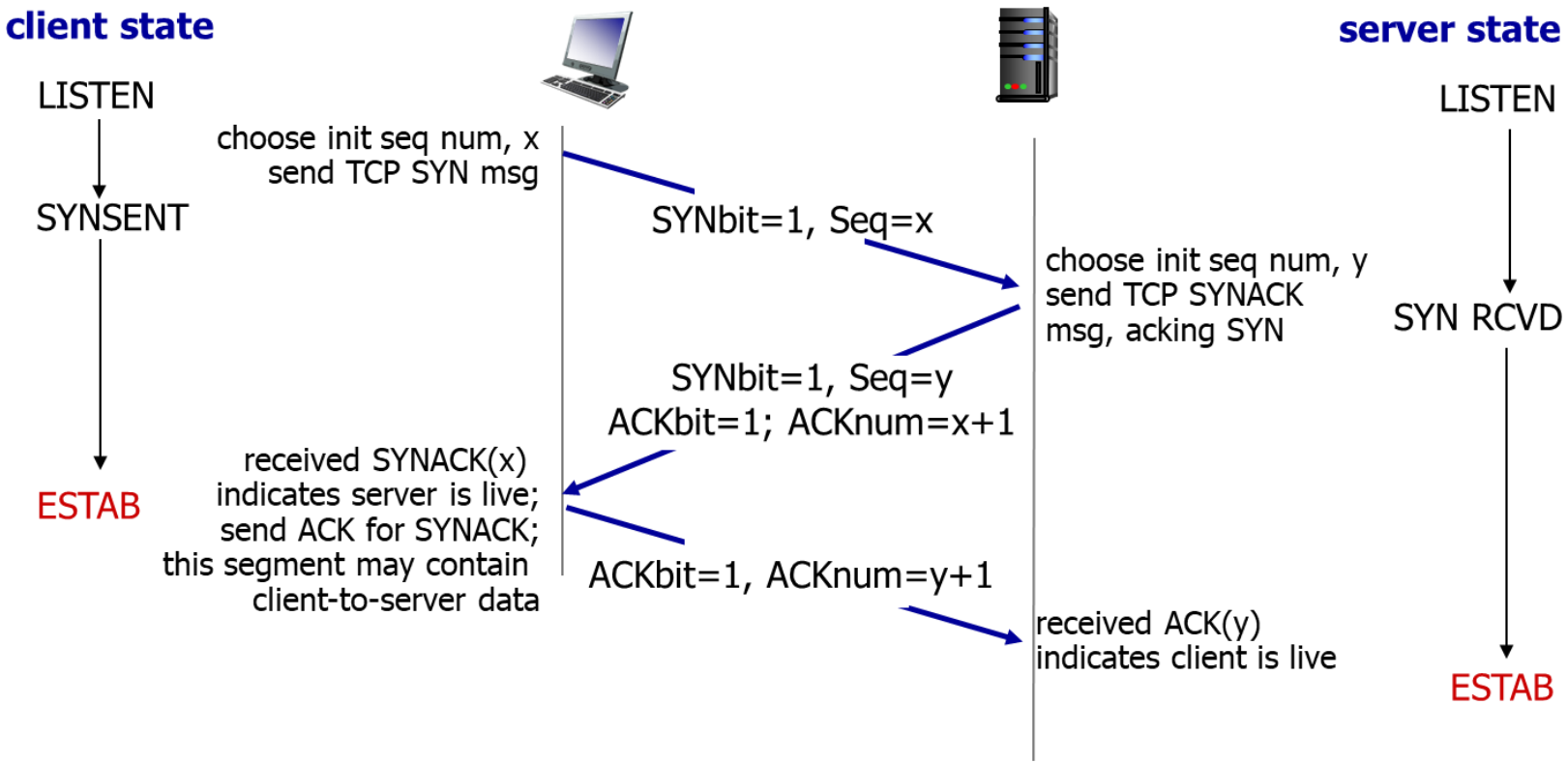



```
Socket clientSocket =
  newSocket("hostname","port
   number");
```




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

TCP 3-Way Handshake



Principles of Congestion Control

congestion:

informally: “too many sources sending too much data too fast for **network** to handle”

different from flow control!

manifestations:

- lost packets (buffer overflow at routers)
- long delays (queueing in router buffers)

a top-10 problem!

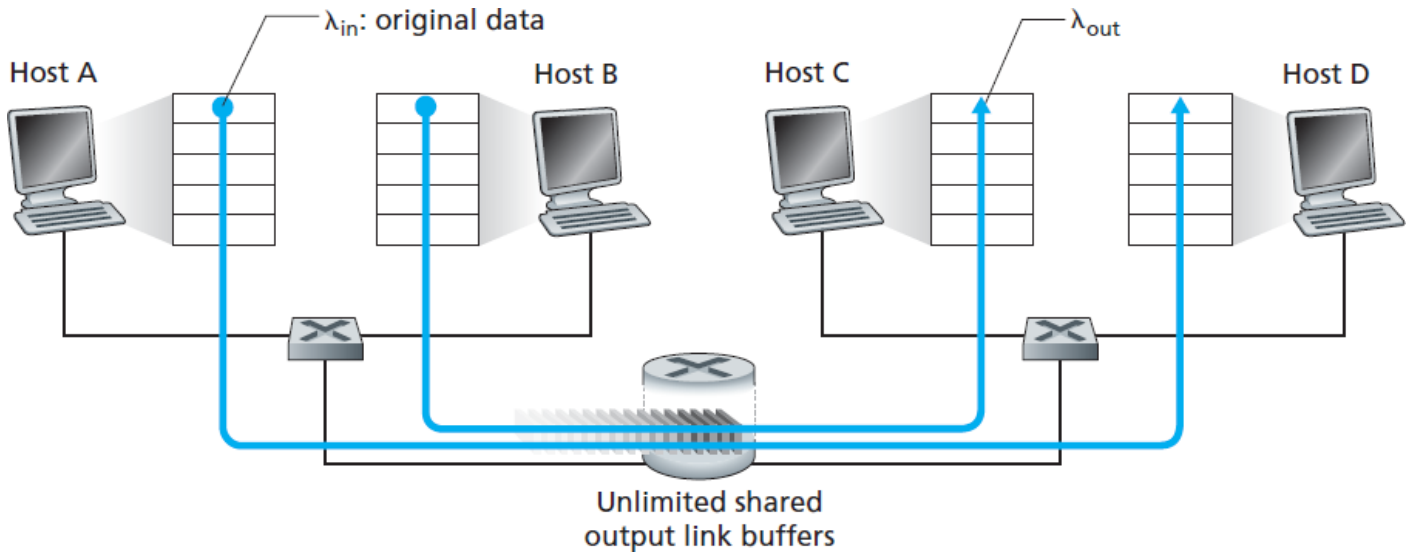
Causes/Costs of Congestion: Scenario 1 (1 of 2)

two senders, two receivers

one router, infinite buffers

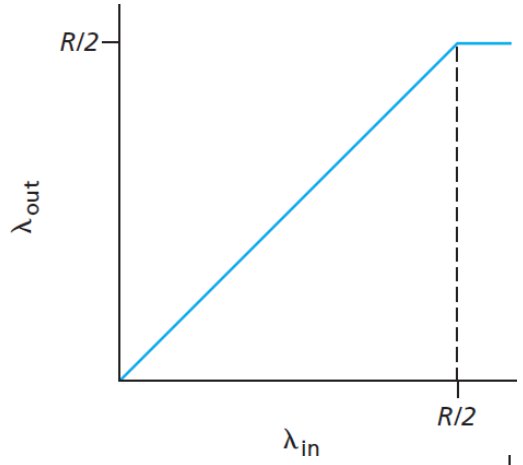
output link capacity: R

no retransmission

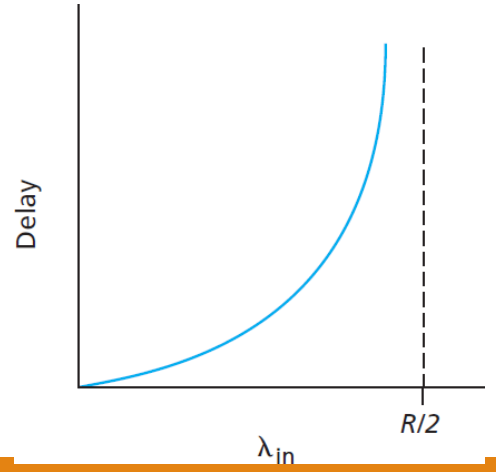


Causes/Costs of Congestion: Scenario 1 (2 of 2)

- maximum per-connection throughput: $\frac{R}{2}$



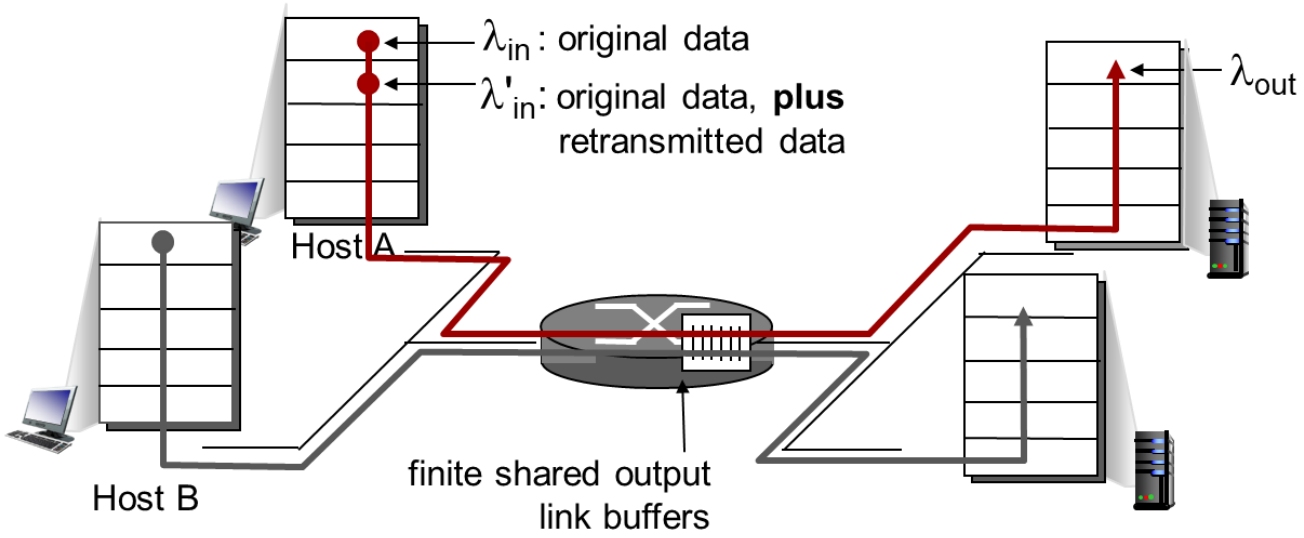
- large delays as arrival rate, λ_{in}, approaches capacity



Causes/Costs of Congestion: Scenario 2 (1 of 6)

- one router, **finite** buffers
- sender retransmission of timed-out packet
  - application-layer input = application-layer output:
  - transport-layer input includes **retransmissions** :

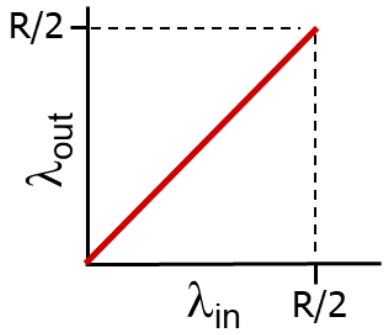
$$\lambda_{in} = \lambda_{out} \quad \lambda'_{in} \geq \lambda_{in}$$

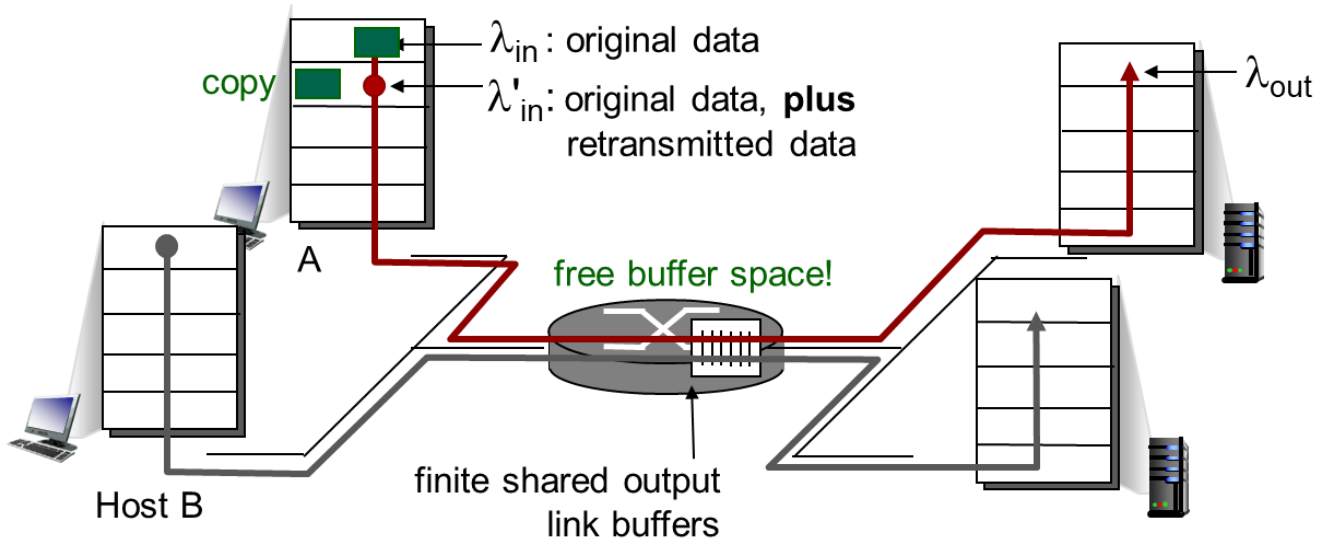


Causes/Costs of Congestion: Scenario 2 (2 of 6)

idealization: perfect knowledge

- sender sends only when router buffers available

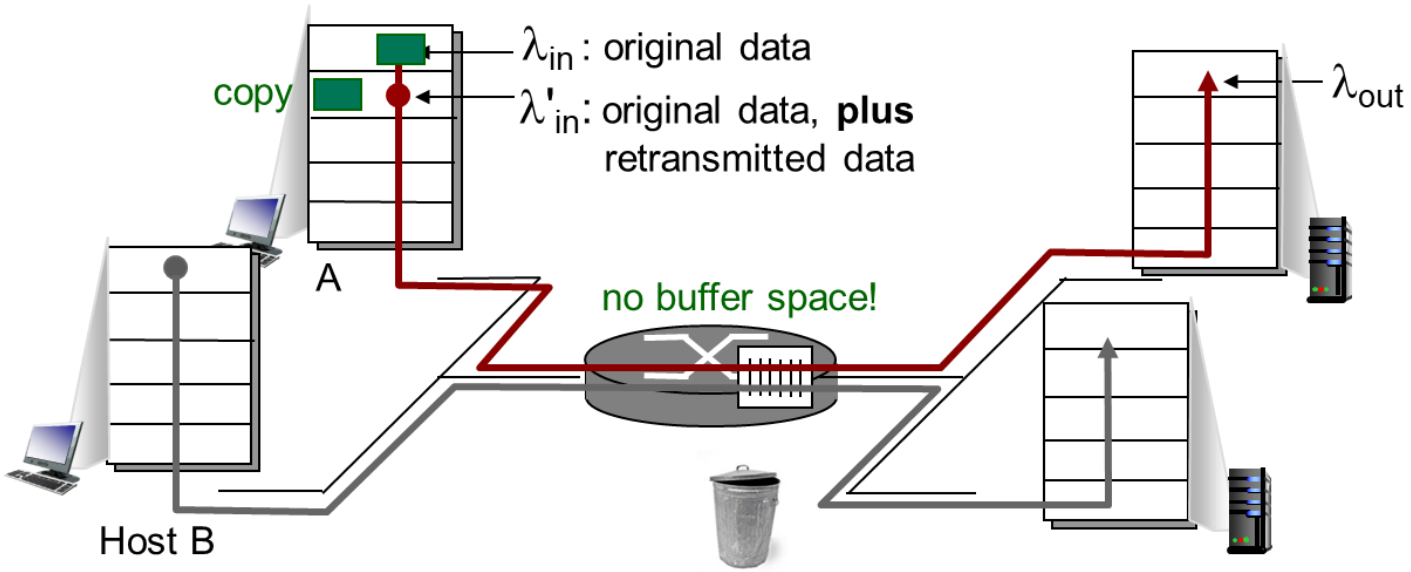




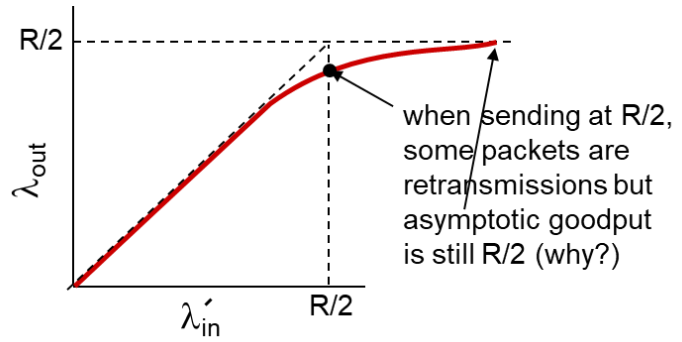
Causes/Costs of Congestion: Scenario 2 (3 of 6)

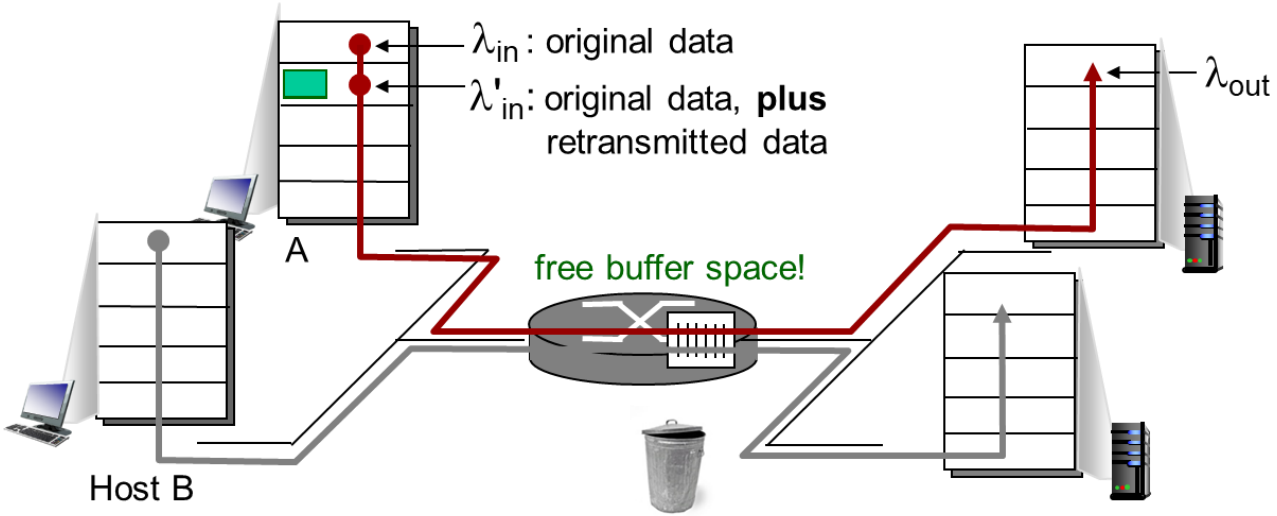
Idealization: known loss packets can be lost, dropped at router due to full buffers

sender only resends if packet **known** to be lost



Causes/Costs of Congestion: Scenario 2 (4 of 6)



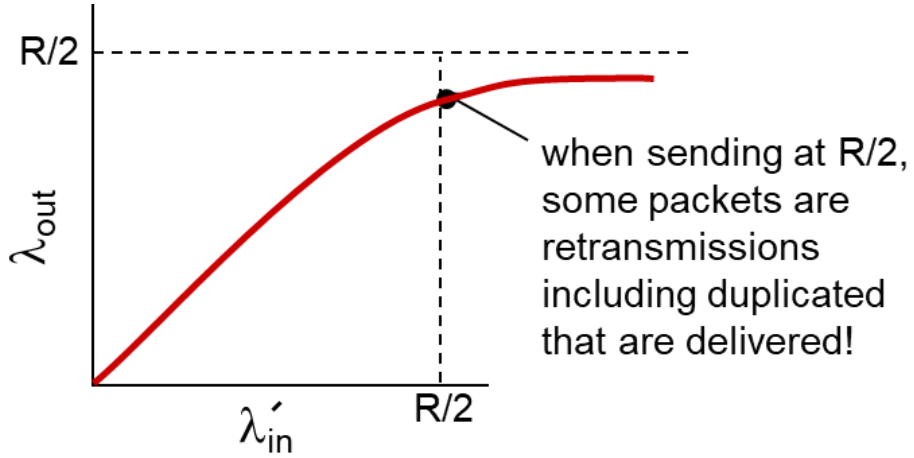


Causes/Costs of Congestion: Scenario 2 (5 of 6)

Realistic: duplicates

packets can be lost, dropped at router due to full buffers

sender times out prematurely, sending **two** copies, both of which are delivered



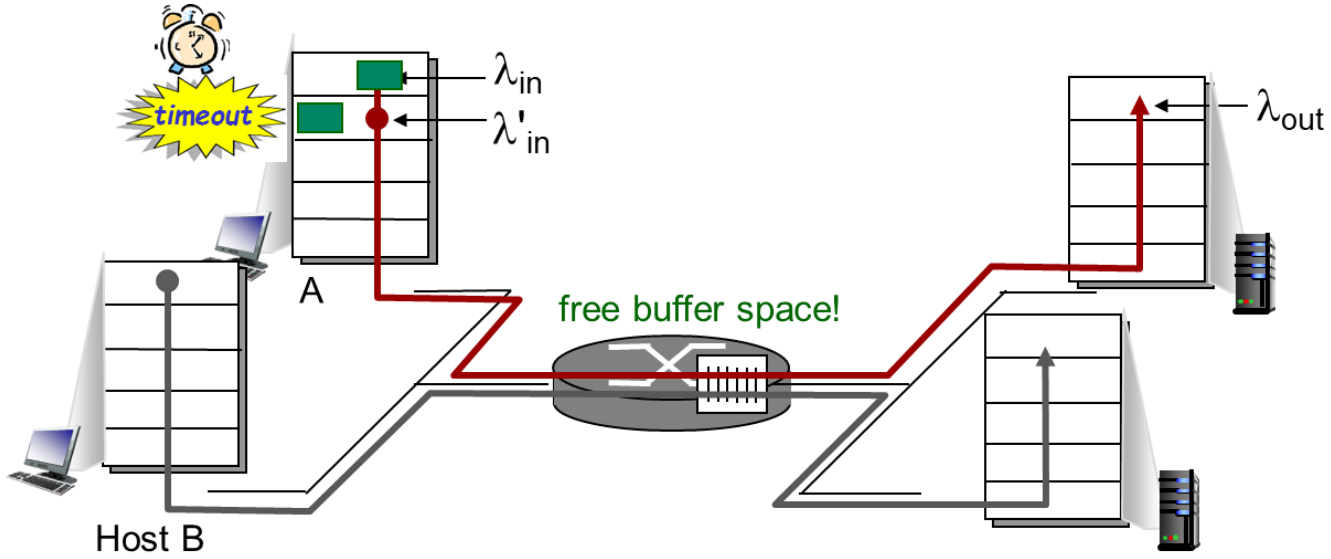
Causes/Costs of Congestion: Scenario 2 (6 of 6)

“costs” of congestion:

more work (retrans) for given “goodput”

unneeded retransmissions: link carries multiple copies of pkt

- decreasing goodput

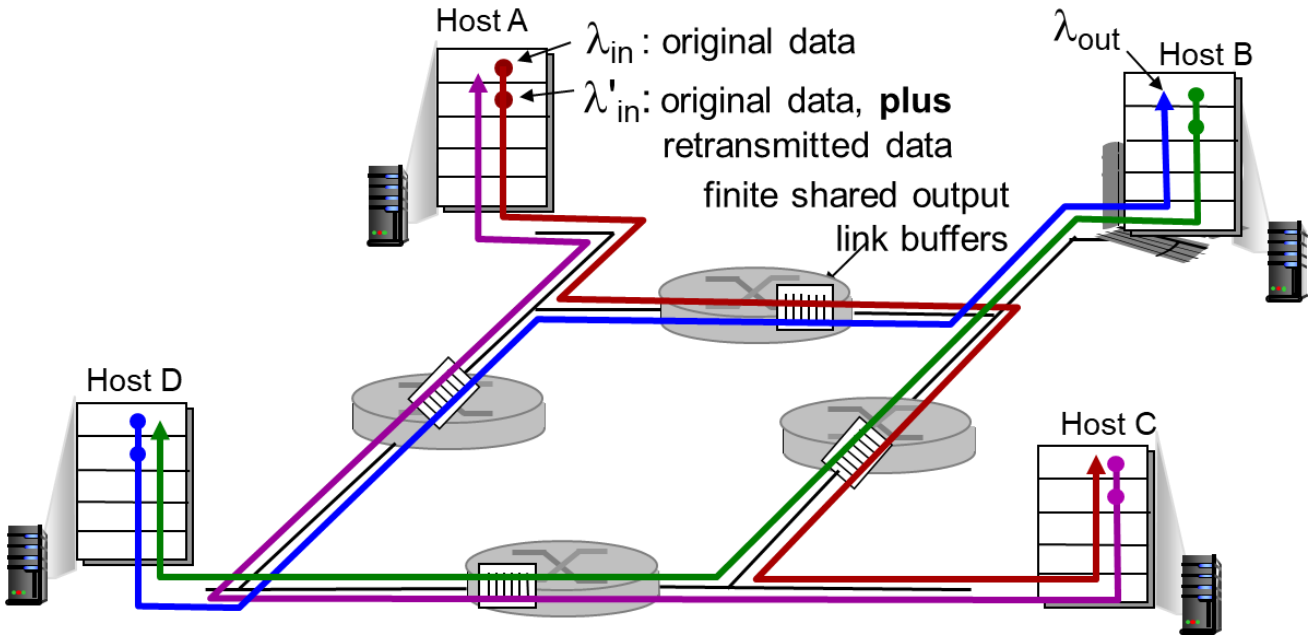


Causes/Costs of Congestion: Scenario 3 (1 of 2)

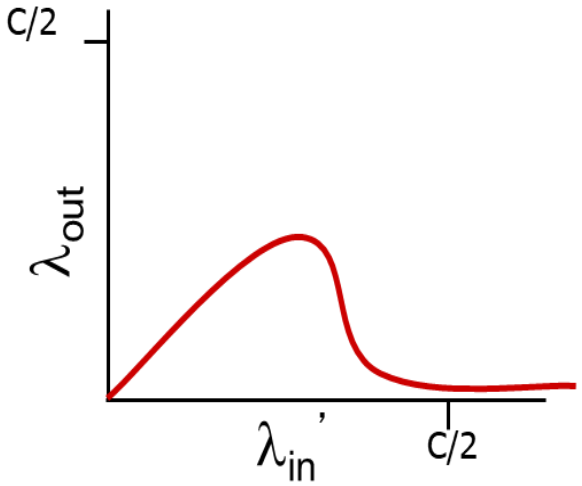
- four senders
- multihop paths
- timeout/retransmit

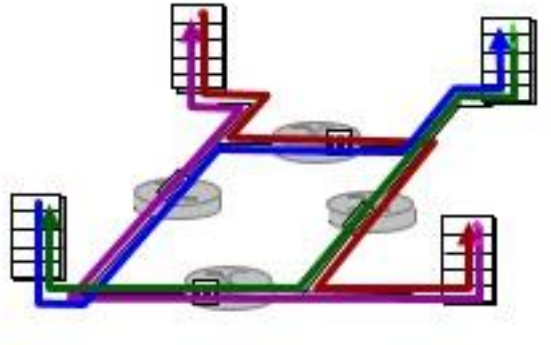
Q: what happens as λ_{in} and λ_{in}' increase ?

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



Causes/Costs of Congestion: Scenario 3 (2 of 2)





another "cost" of congestion:

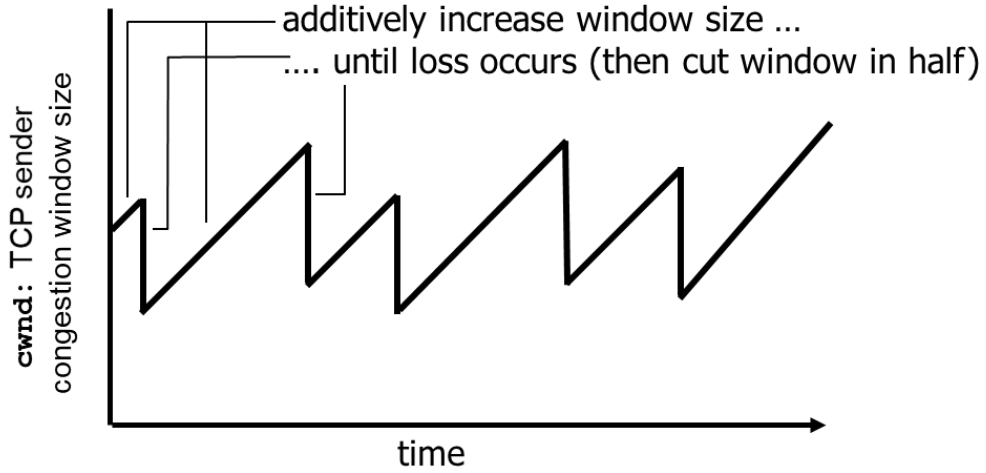
when packet dropped, any "upstream transmission capacity used for that packet was wasted!

TCP Congestion Control: Additive Increase Multiplicative Decrease

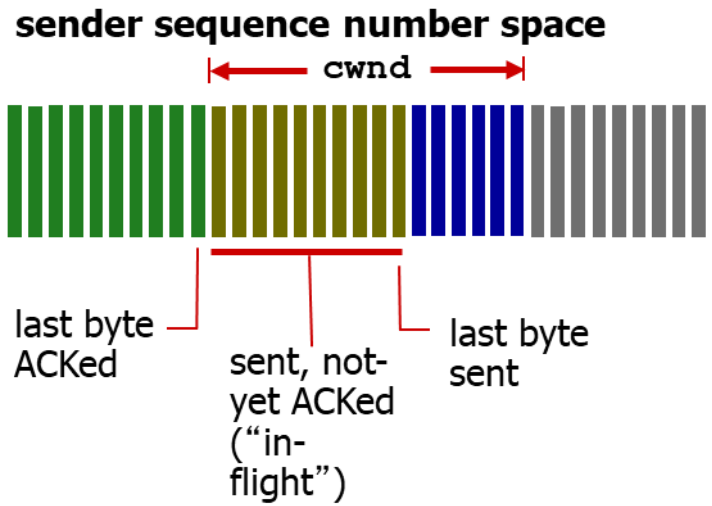
approach: sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs

- **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:

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

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

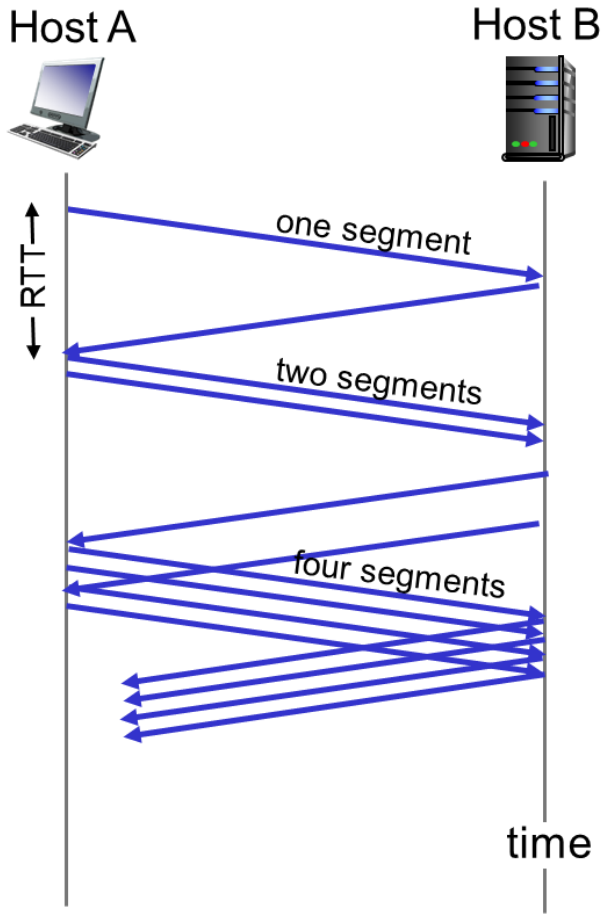
TCP sending rate:

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

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

TCP 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**: **TCP RENO**

- **`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 plus 3 (for Fast Recovery), then grows linearly

TCP Tahoe always sets **`cwnd`** to 1 (for both timeout and 3 duplicate acks) and enter Slow Start

TCP: Switching from Slow Start to CA

Q: when should the exponential increase switch to linear?

A: when **`cwnd`** gets to $\frac{1}{2}$ of its value before timeout.

Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to $\frac{1}{2}$ of **`cwnd`** just before loss event

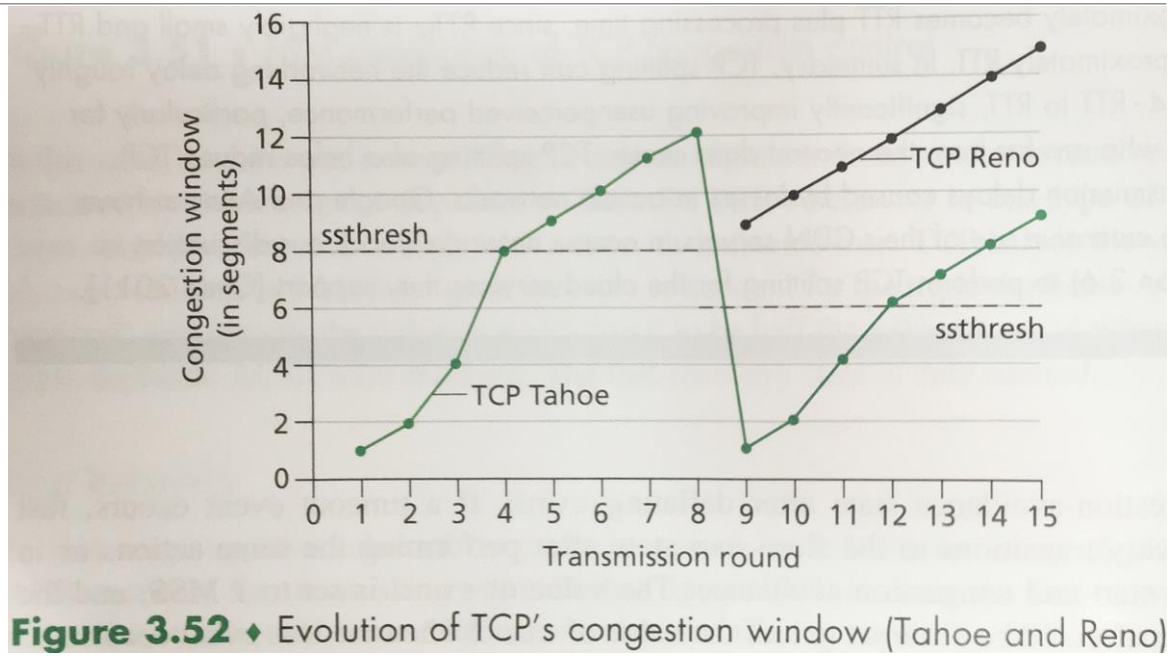


Figure 3.52 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

In Fig 3.52, at transmission round 9, triple duplicate ACKs occur

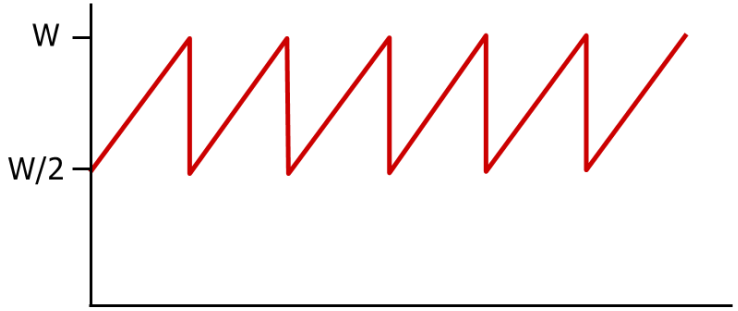
- ssthresh := cwnd/2 = 12/2 = 6 MSS
- **Reno**: cwnd := ssthresh + 3 = 6 + 3 = 9 MSS
- **Tahoe**: cwnd := 1 MSS

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

TCP Throughput

- average 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
  - average window size (# in-flight bytes) is
  - average thruput is $\frac{3}{4}W$ per RTT $\frac{3}{4}W$

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

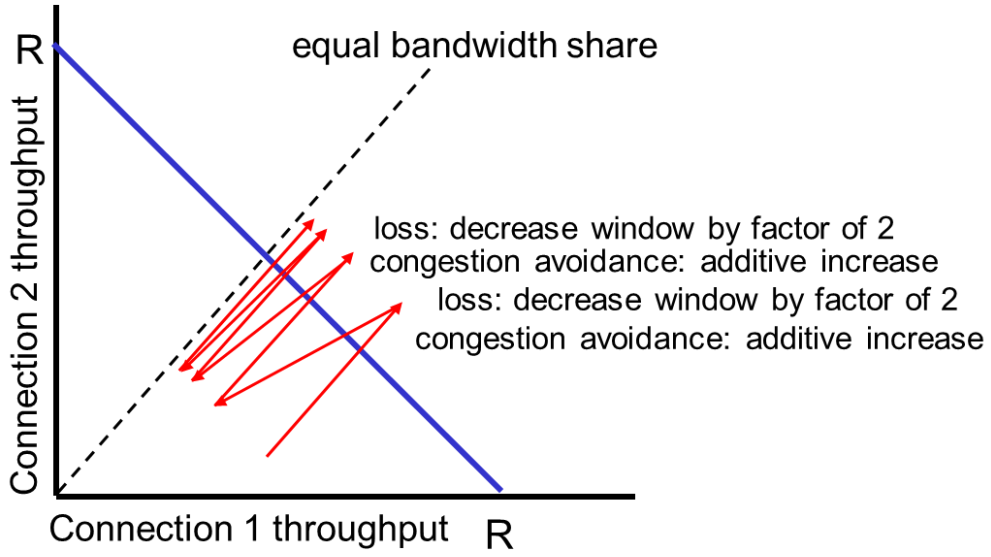


Why is TCP Fair?

two competing sessions:

additive increase gives slope of 1, as throughout increases

multiplicative decrease decreases throughput proportionally



Network Layer: Router

DR DUY NGO

SCHOOL OF ELECTRICAL ENGINEERING &COMPUTING

Network Layer

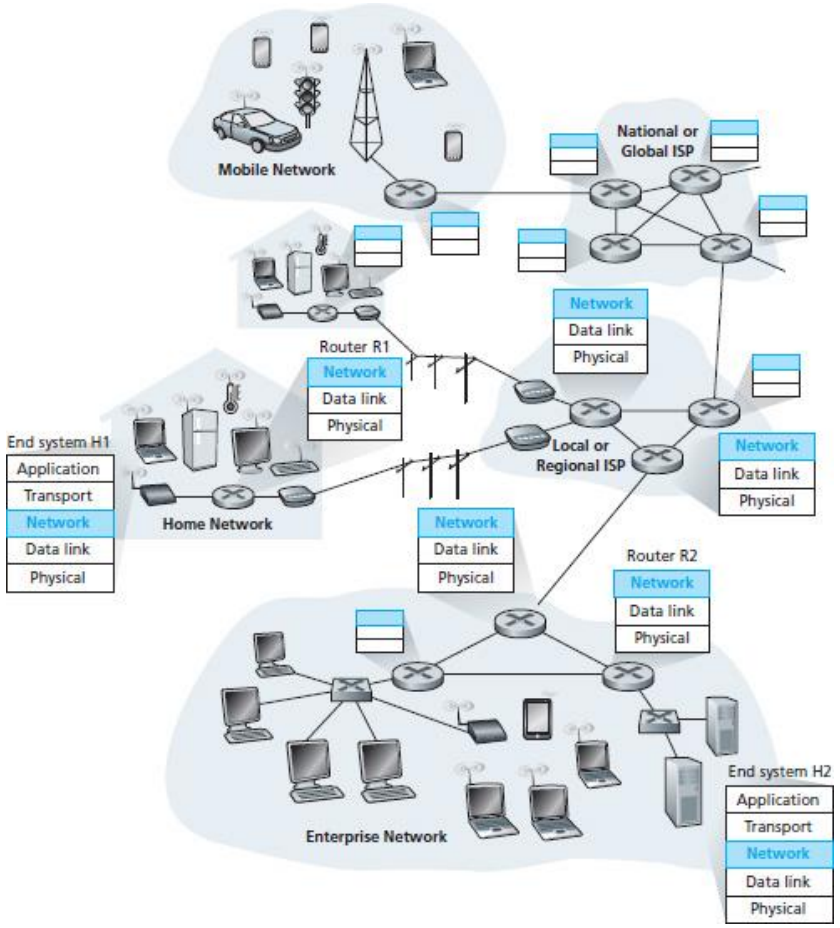
transport segment from sending to receiving host

on sending side encapsulates segments into datagrams

on receiving side, delivers segments to transport layer

network layer protocols in **every** host, router

router examines header fields in all IP datagrams passing through it



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

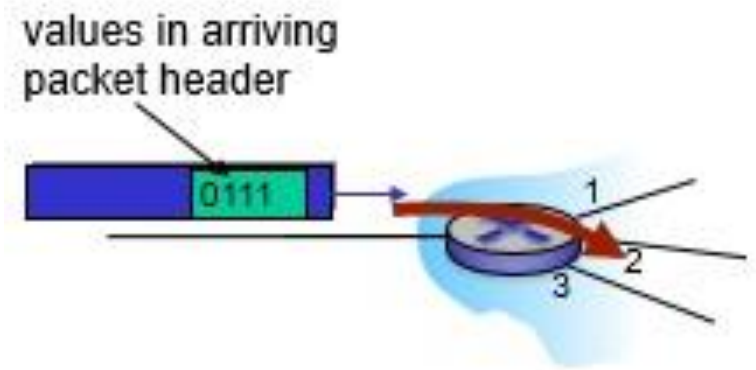
Network Layer: Data Plane, Control Plane

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

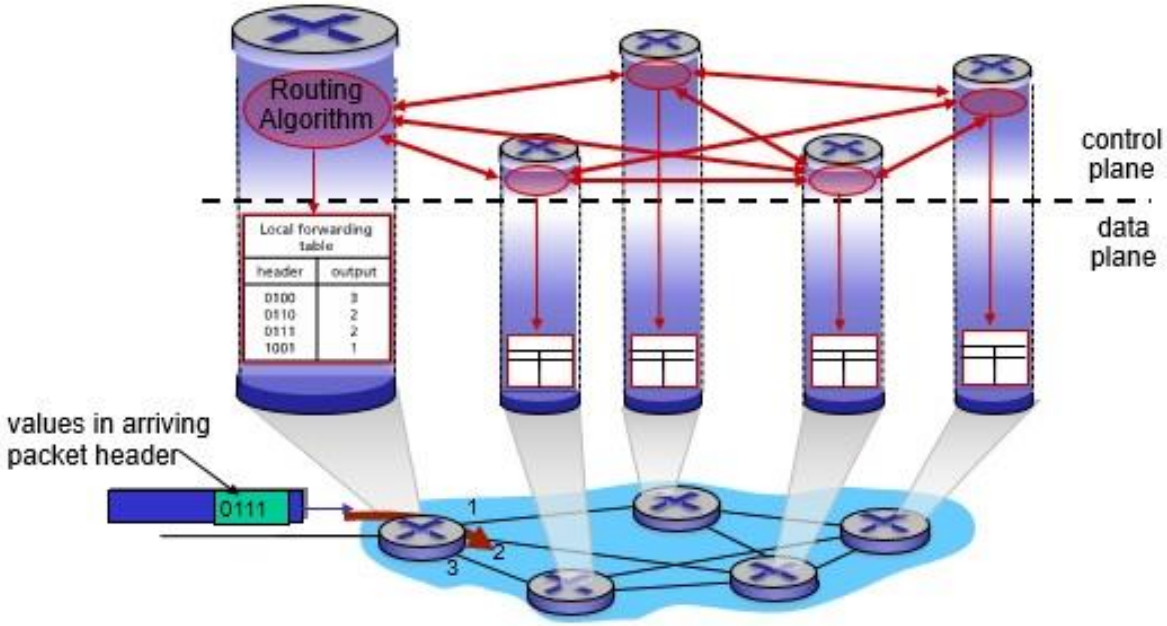
determines how datagram is routed among routers along end-end path from source host to destination host

two control-plane approaches:

- **traditional routing algorithms:** implemented in routers
- **software-defined networking (SDN):** implemented in (remote) servers

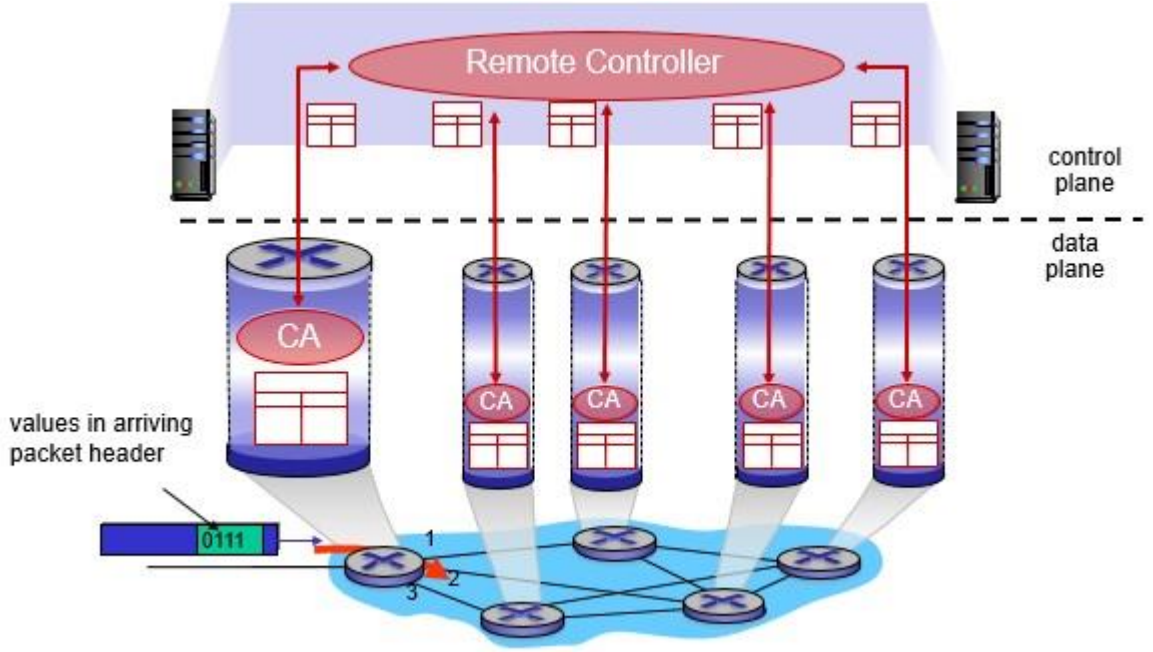
Per-Router Control Plane

Individual routing algorithm components **in each and every router** interact in the control plane



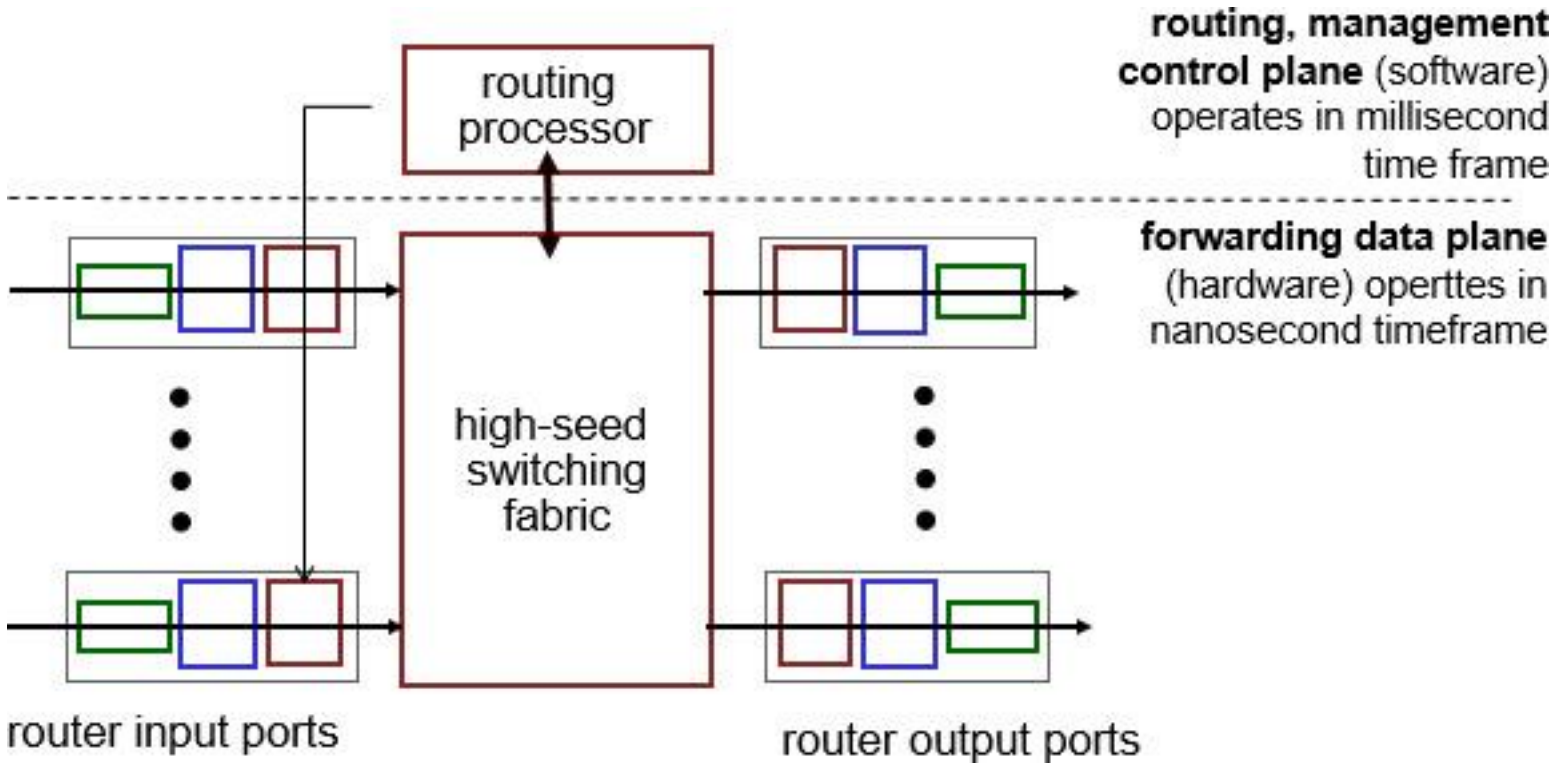
Logically Centralized Control Plane

A distinct (typically remote) controller interacts with local control agents (CAs)

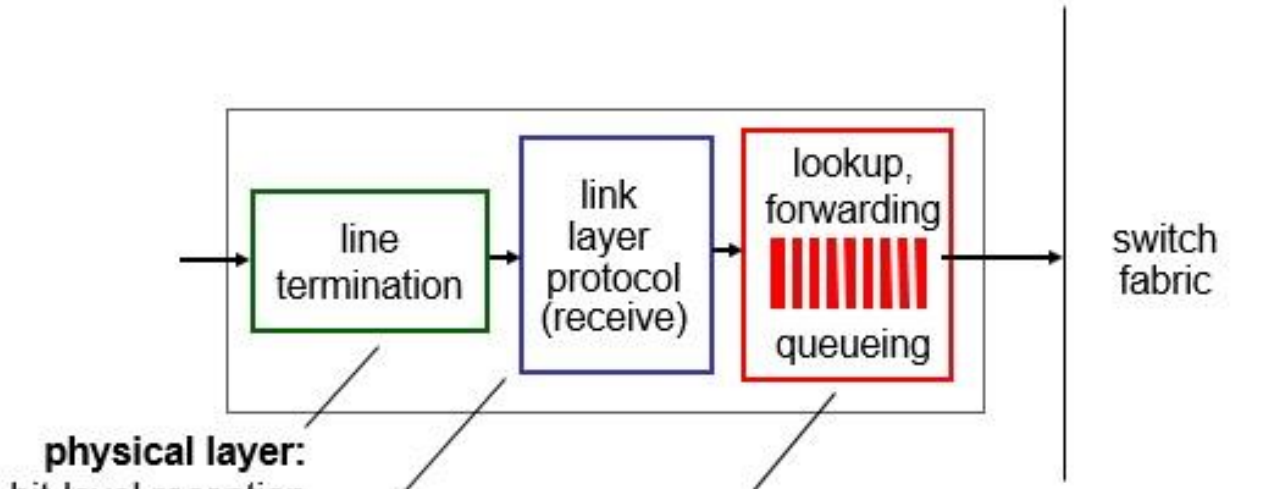


Router Architecture Overview

high-level view of generic router architecture:



Input Port Functions (1 of 2)



decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory **("match plus action")**
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

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 (1 of 2)

longest prefix matching

when looking for forwarding table entry for given destination address, use **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:

DA: 11001000 00010111 00010110 10100001 **which interface?**

DA: 11001000 00010111 00011000 10101010 **which interface?**

Longest Prefix Matching (2 of 2)

we'll see **why** longest prefix matching is used shortly, when we study addressing

longest prefix matching: often performed using ternary content addressable memories (TCAMs)

- **content addressable:** present address to TCAM : retrieve address in one clock cycle, regardless of table size
- Cisco Catalyst: can up ~1M routing table entries in TCAM

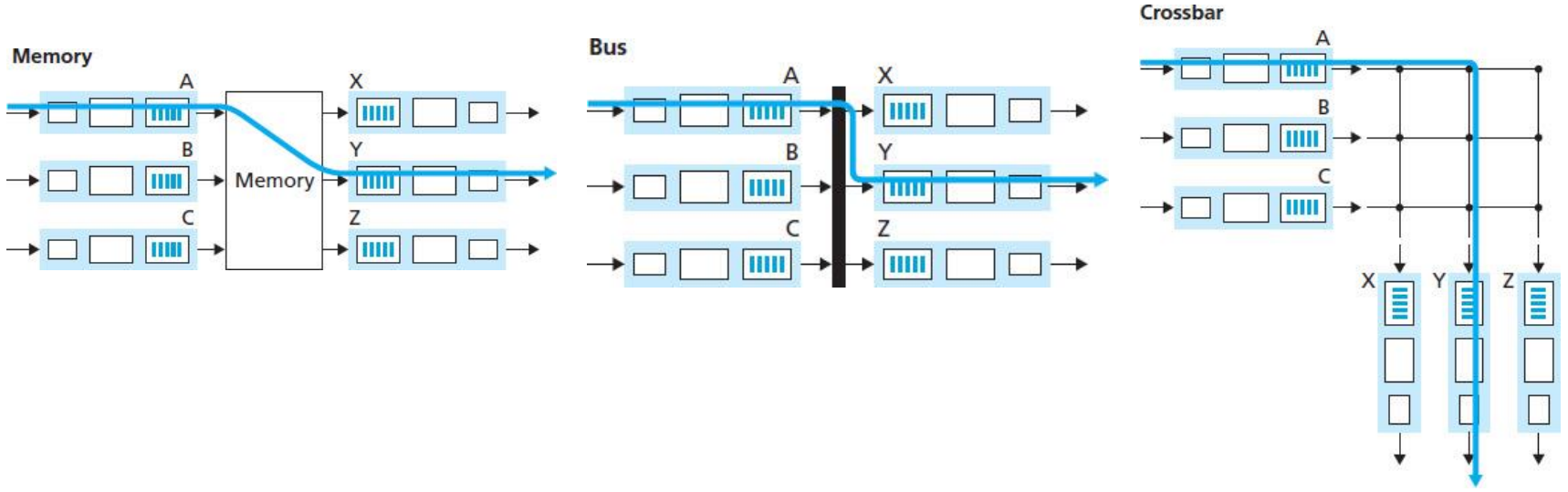
Switching Fabrics

transfer packet from input buffer to appropriate output buffer

switching rate: rate at which packets can be transfer from inputs to outputs

- often measured as multiple of input/output line rate
- N inputs: switching rate N times line rate desirable

three types of switching fabrics

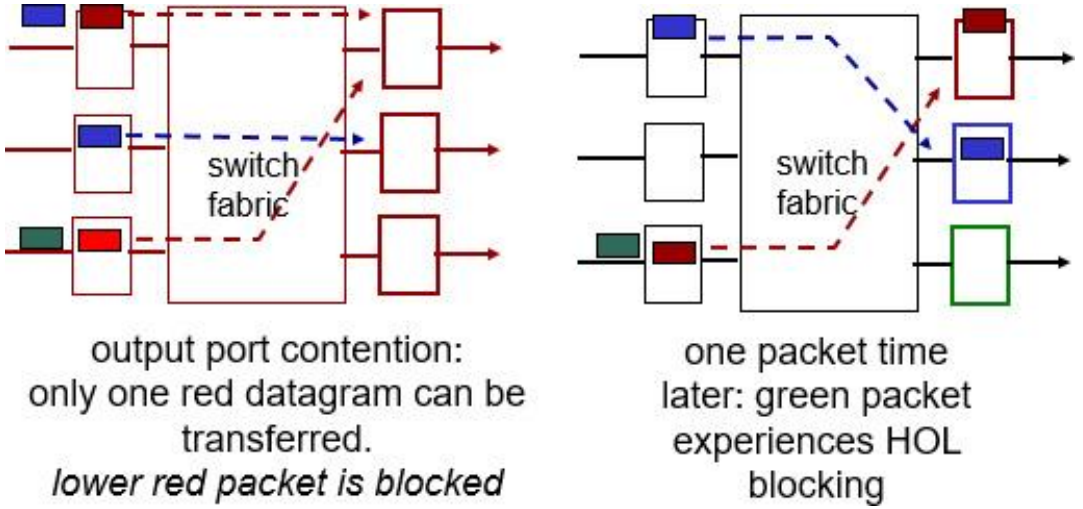


Input Port Queuing

fabric slower than input ports combined -> queueing may occur at input queues

- **queueing delay and loss due to input buffer overflow!**

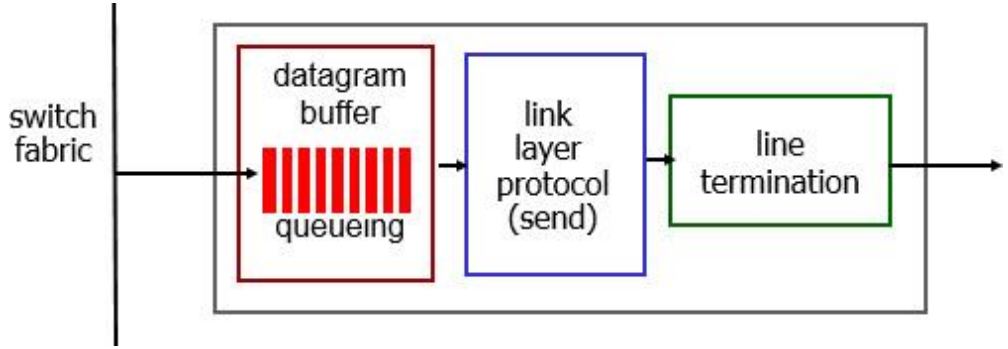
Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward



output port contention: only one red datagram can be transferred. *lower red packet is blocked*

one packet time later: green packet experiences HOL blocking

Output Ports



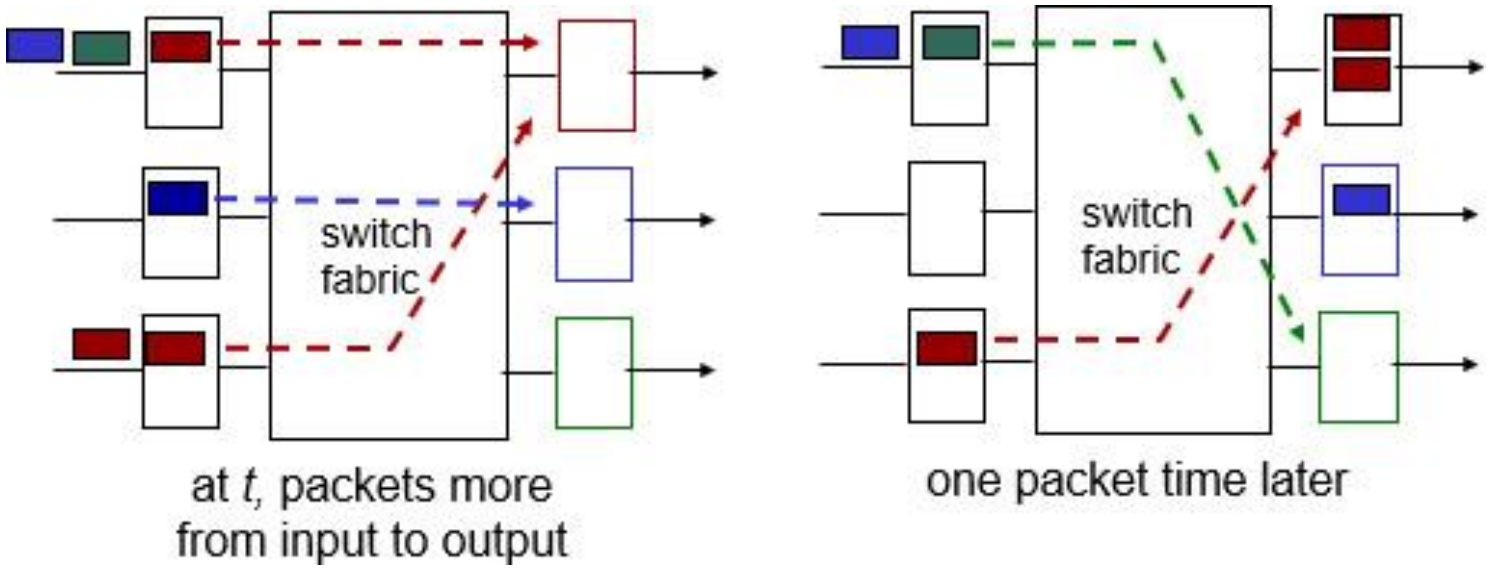
buffering required when datagrams arrive from fabric faster than the transmission rate

Datagram (packets) can be lost due to congestion, lack of buffers

scheduling discipline chooses among queued datagrams for transmission

Priority scheduling – who gets best performance, network neutrality

Output Port Queueing



buffering when arrival rate via switch exceeds output line speed

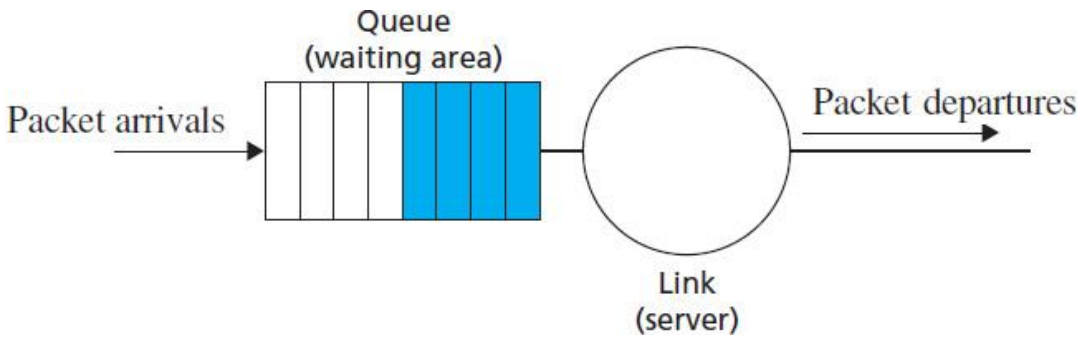
queueing (delay) and loss due to output port buffer overflow!

Scheduling Mechanisms

scheduling: choose next packet to send on link

FIFO (first in first out) scheduling: send in order of arrival to queue

- real-world example?
- **discard policy:** if packet arrives to full queue: who to discard?
  - **tail drop:** drop arriving packet
  - **priority:** drop/remove on priority basis
  - **random:** drop/remove randomly

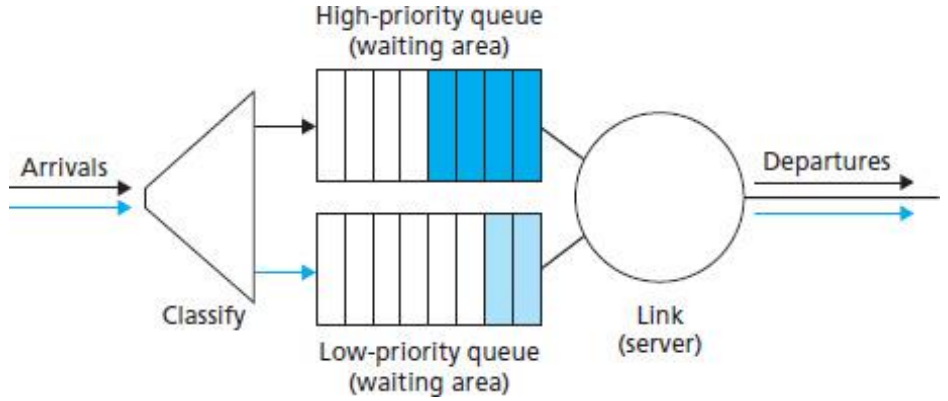


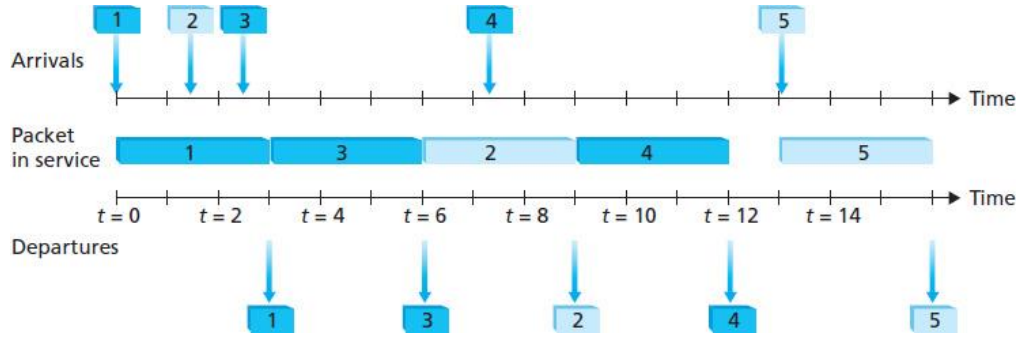
Scheduling Policies: Priority

priority scheduling: send highest priority queued packet

multiple **classes**, with different priorities

- class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
- real world example?





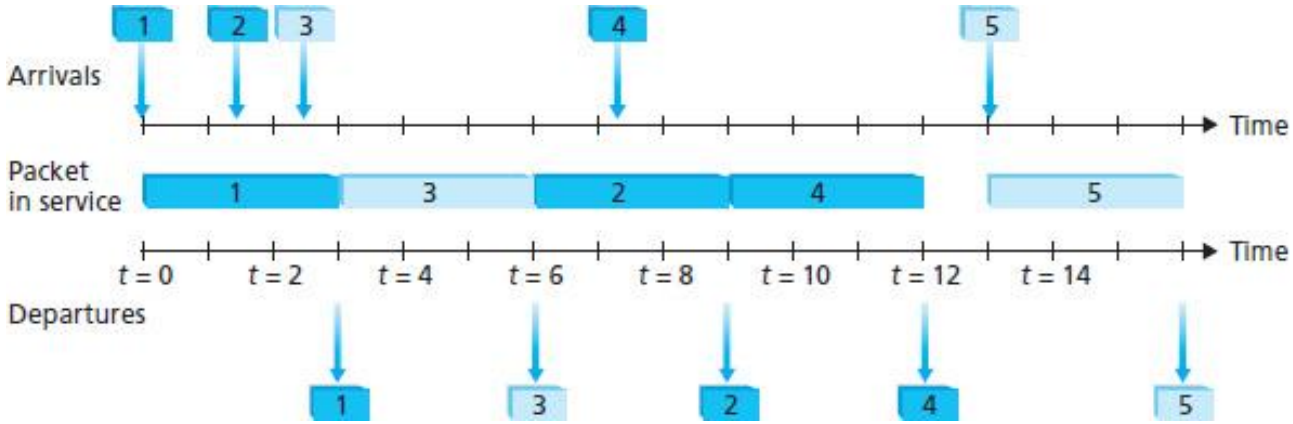
Scheduling Policies: Round Robin

Round Robin (RR) scheduling:

multiple classes

cyclically scan class queues, sending one complete packet from each class (if available)

real world example?



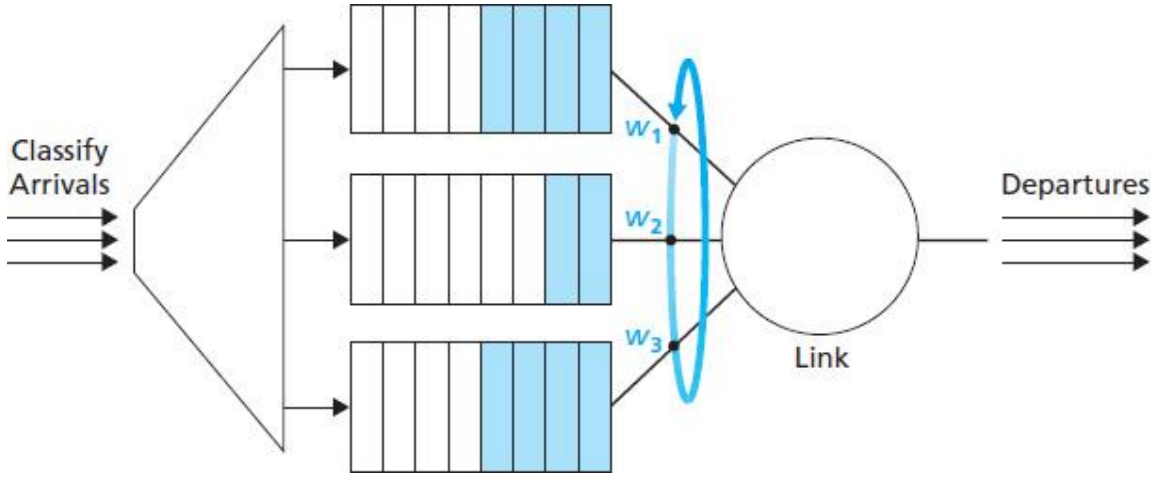
Scheduling Policies: Weighted Fair Queuing

Weighted Fair Queuing (WFQ):

generalized Round Robin

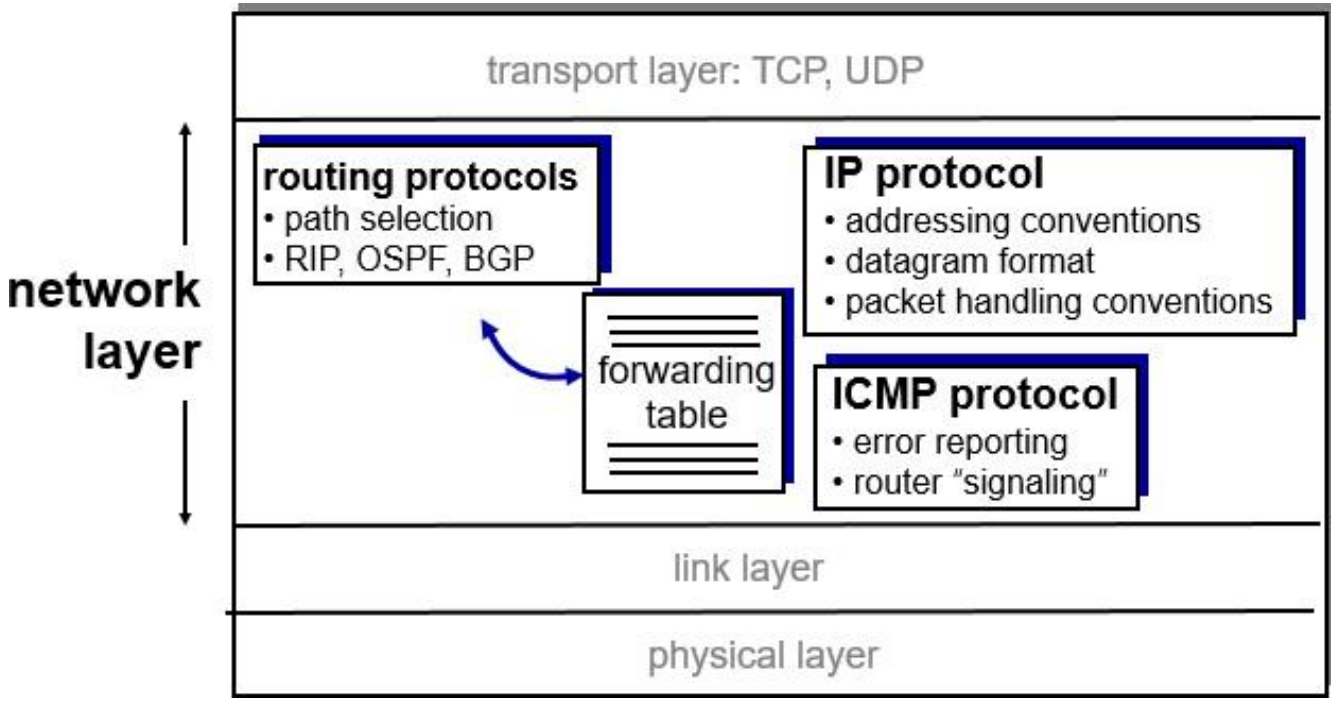
each class gets weighted amount of service in each cycle

real-world example?

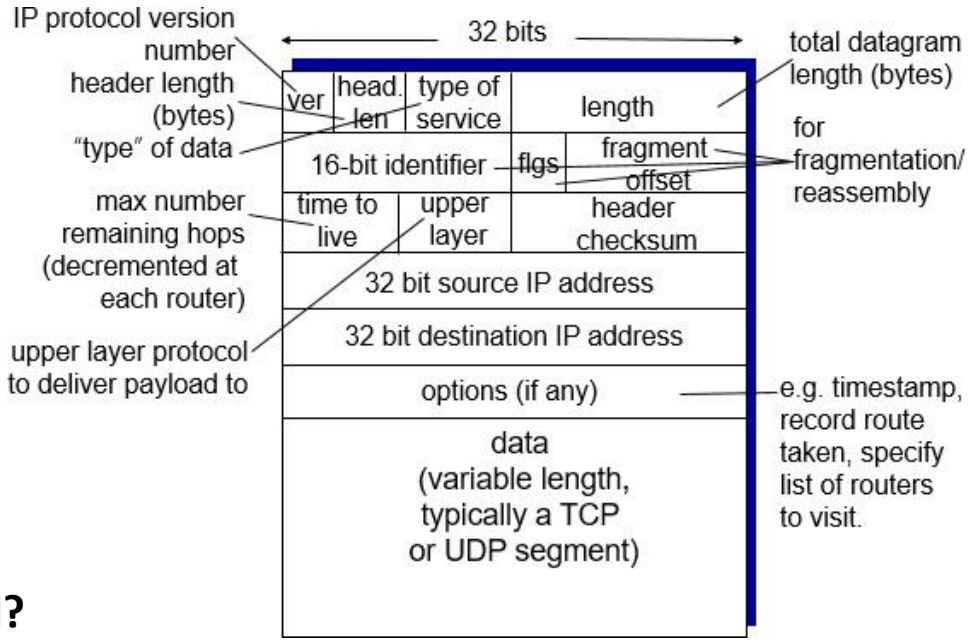


The Internet Network Layer

host, router network layer functions:



IP Datagram Format



how much overhead?

20 bytes of TCP

20 bytes of IP

= 40 bytes + app layer overhead

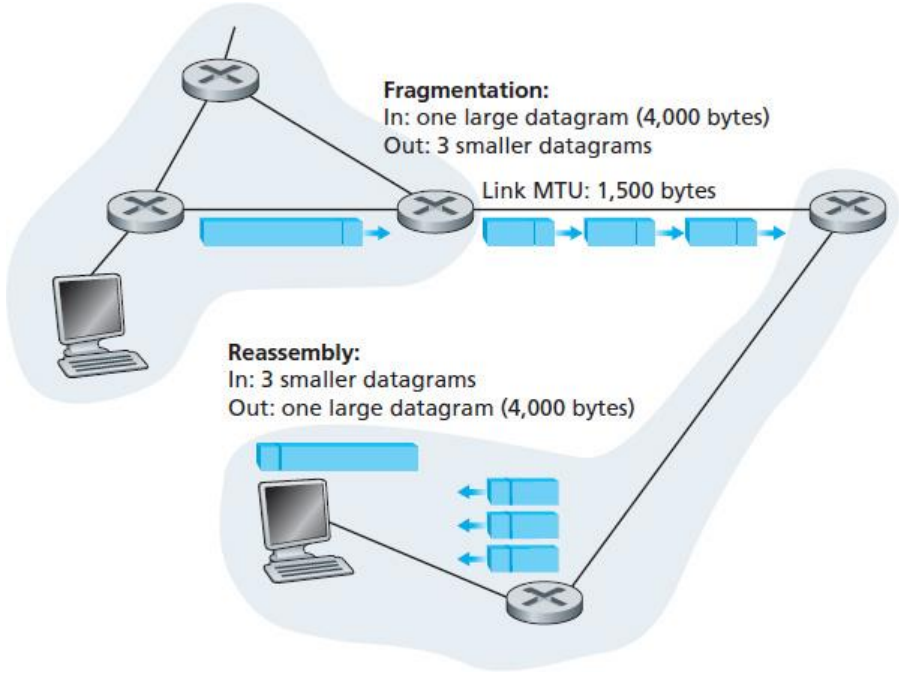
IP Fragmentation, Reassembly

network links have MTU (max.transfer size) - largest possible link-level frame

- different link types, different MTUs

large IP datagram divided ("fragmented") within net

- one datagram becomes several datagrams
- "reassembled" only at final destination
- IP header bits used to identify, order related fragments

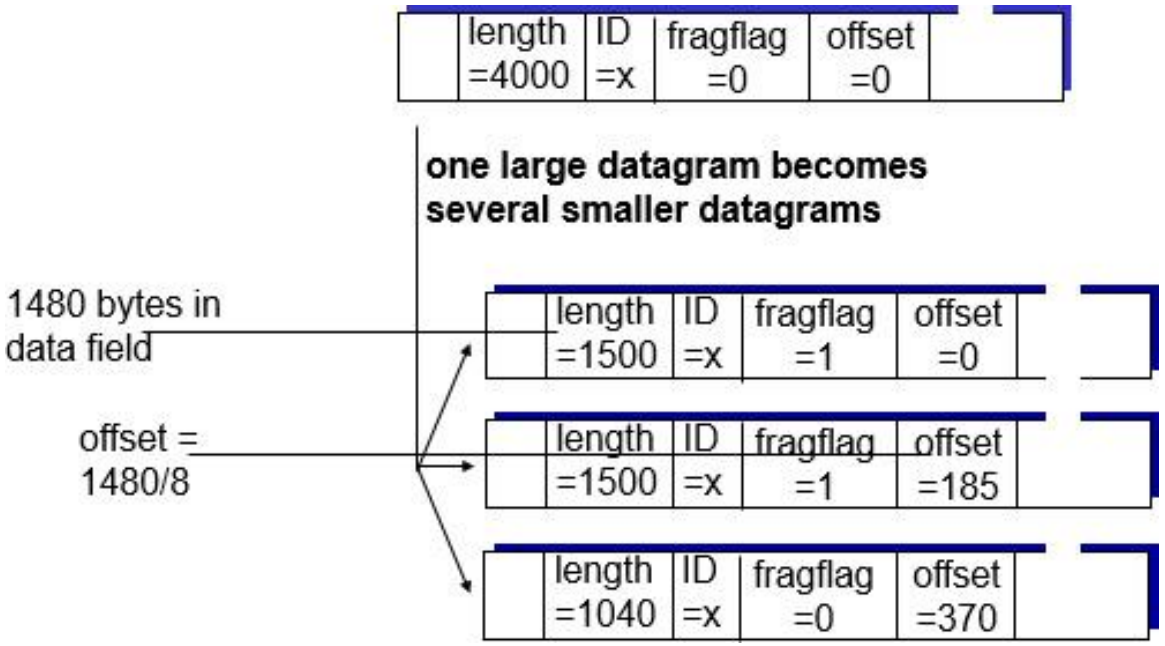


IP Fragmentation, Reassembly

example:

4000 byte datagram

MTU = 1500 bytes



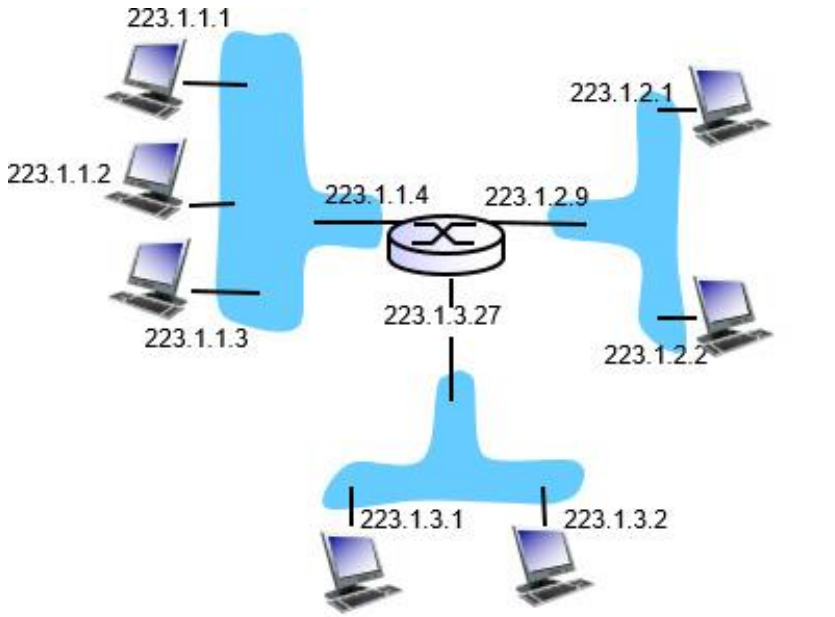
IP Addressing: Introduction (1 of 2)

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)

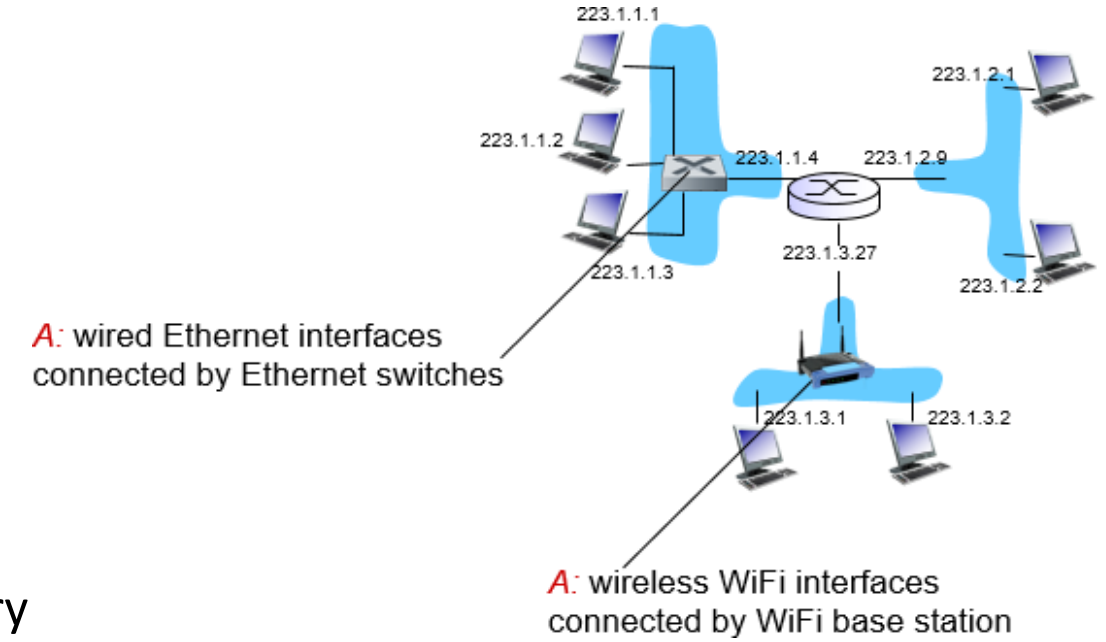
IP addresses associated with each interface



IP Addressing: Introduction (2 of 2)

Q: how are interfaces actually connected?

A: we'll learn about that in chapter 5, 6.



For now: don't need to worry about how one interface is connected to another (with no intervening router)

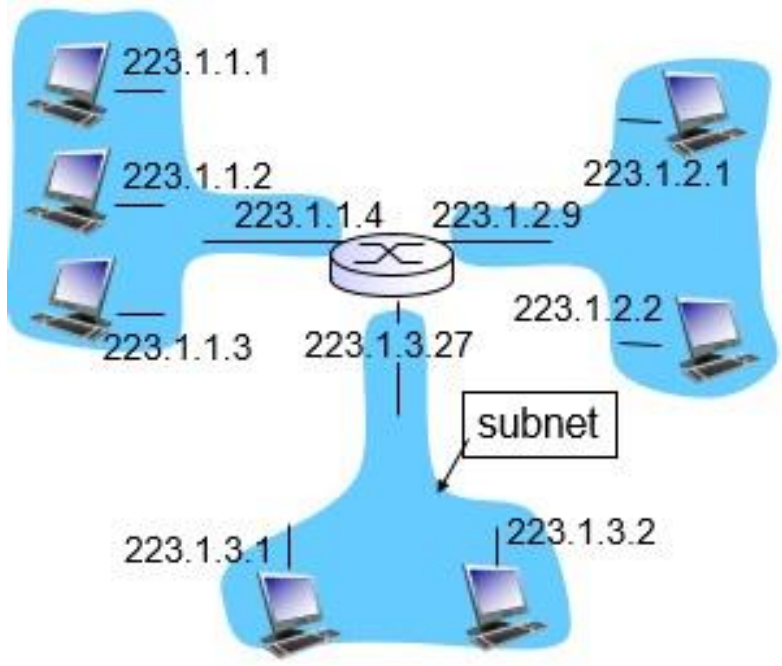
Subnets (1 of 3)

IP address:

- subnet part - high order bits
- host part - low order bits

What's 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

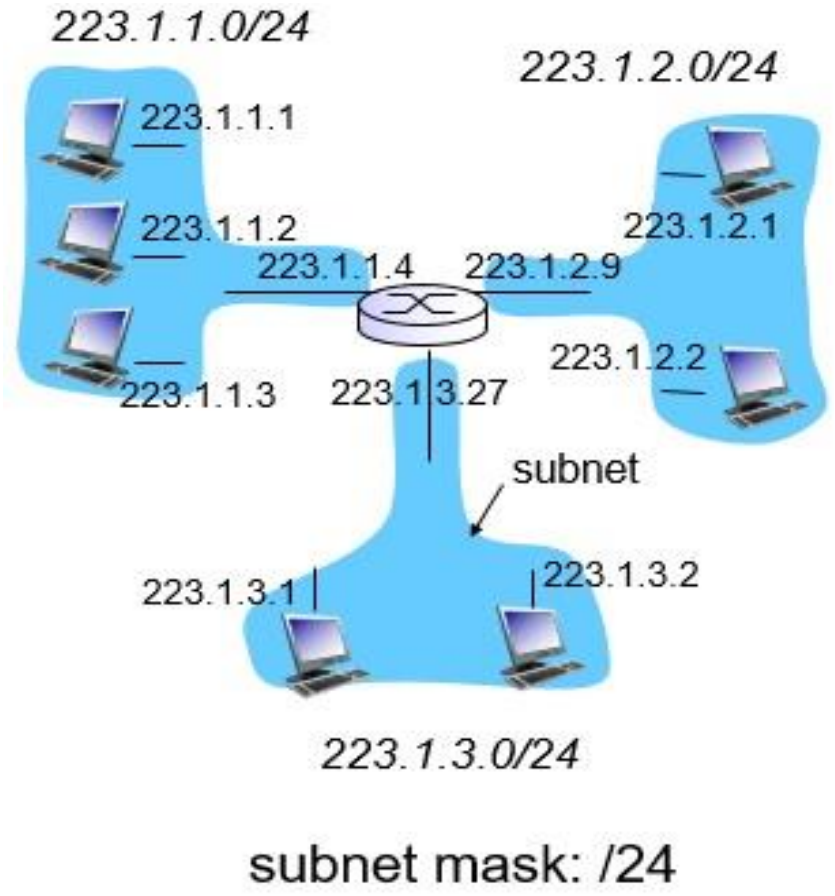
- Here we will use the IPv4 addresses.
- The IP header is 32 bit long, used as a unique identifier to locate a device on the IP network
- To make networks scalable, the address structure is subdivided into the fields: the **Network(Subnet) ID** and the **Host ID**
- The **Network/subnet ID** identifies a network where the device with the IP address is connected
- The **Host ID** identifies the device within the networl
- The IP address is subdivided into five classes as below:
  - Class A: fewer networks, but large networks, 7 bit Net/subnet ID, 24 bit Host ID
  - Class B: more networks than class A, moderate number of hosts/network, 14 bit Net/subnet ID, 16 bit Host ID
  - Class C: Very large no. of networks, few hosts/network, 21 Net/subnet ID, 8 bit Host ID
  - Class D: Multicast addressing
  - Class E: Reserved for experiments

Subnets (2 of 3)

recipe

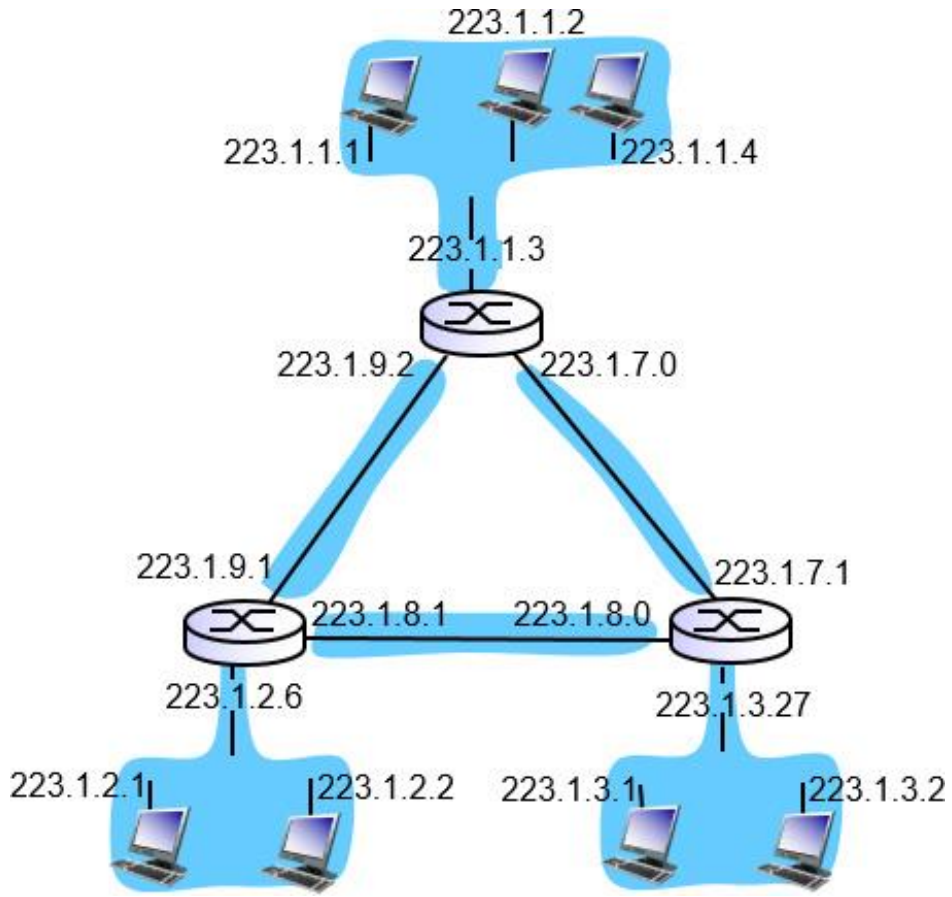
to determine the subnets, detach each interface from its host or router, creating islands of isolated networks

each isolated network is called a **subnet**



Subnets (3 of 3)

how many?



Classless Interdomain Routing (CIDR)

- A certain C class address space to an organisation doesn't guarantee that all addresses with the space can be used and therefor some addresses may be wasted
- This kind of situation is inflexible and would exhaust the IP address space
- The classful addressing scheme consists of class A, B, C, D and E results in an inefficient use of the address space for certain gateway routers
- Motivation:
  - A new scheme with no restriction on the classes emerged as the CIDR which is more flexible, allowing:
    - A variable length ***prefix field*** to represent the network
    - The remaining bits of the 32 field address to represent the hosts (normally don't care field for a gateway or router)
  - Example: One organisation may choose a 20 bit network ID, whereas another organisation may choose a 21 bit network ID, with first 20 bits of these two network IDs being identical. That means the address space of one organisation contains that of another one

IP Addressing: CIDR

CIDR: Classless **I**nter **D**omain **R**outing

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

DHCP: Dynamic Host Configuration Protocol

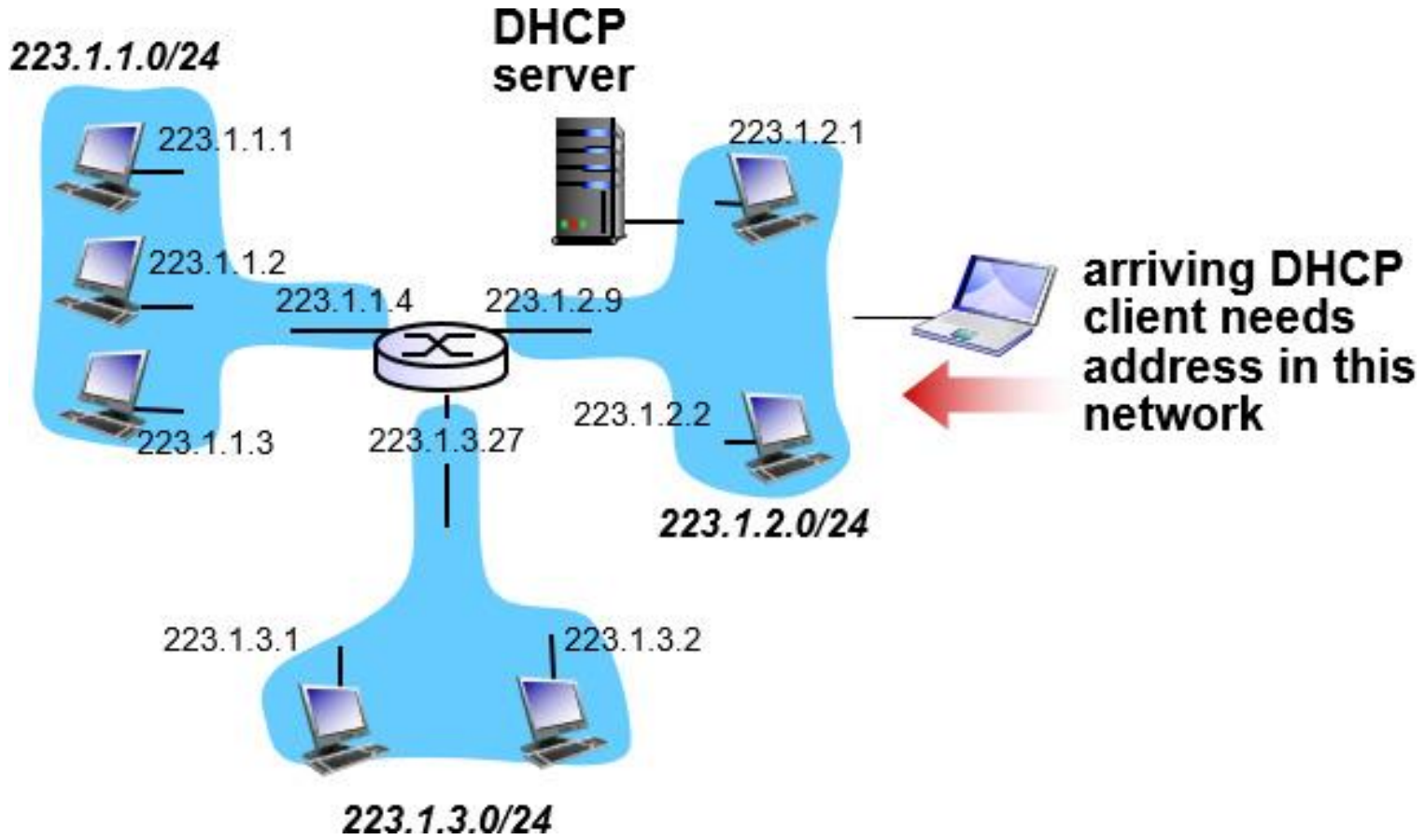
goal: allow host to **dynamically** obtain its IP address from network server when it joins network

- 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)

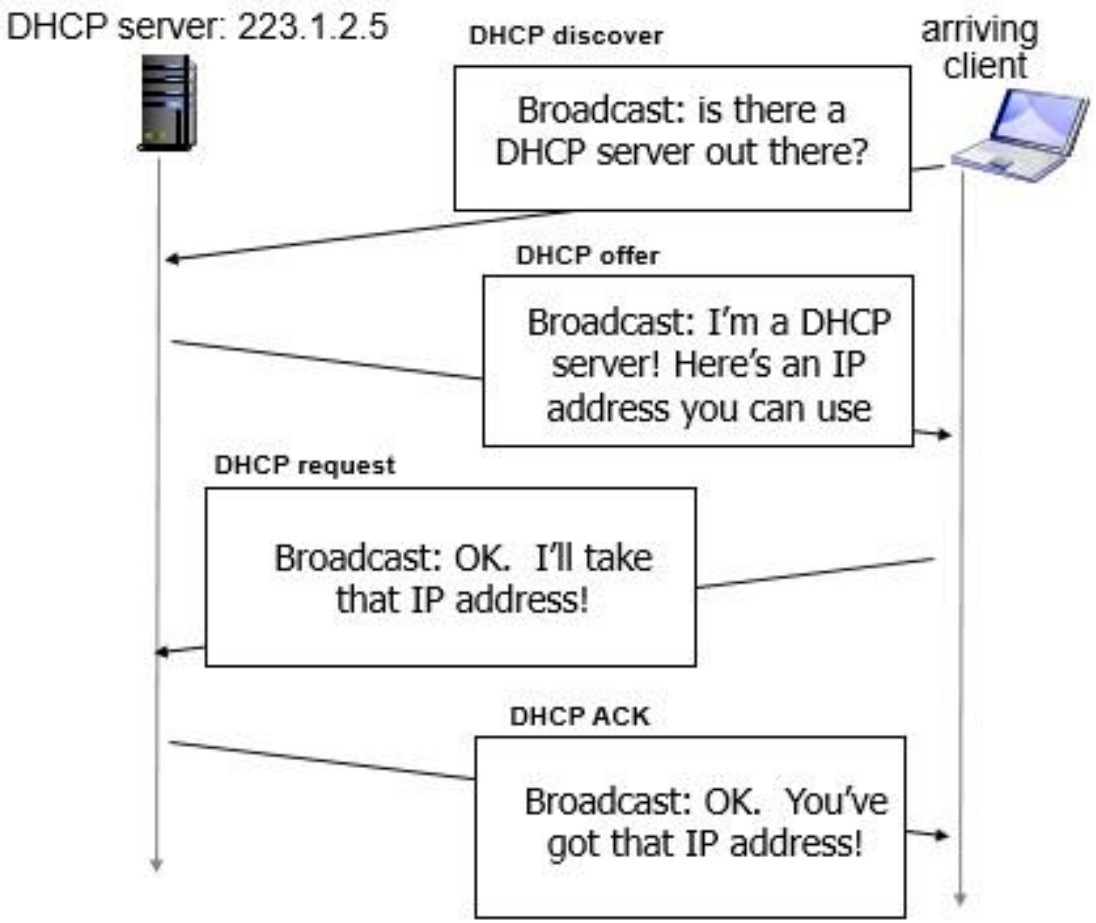
DHCP overview:

- 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

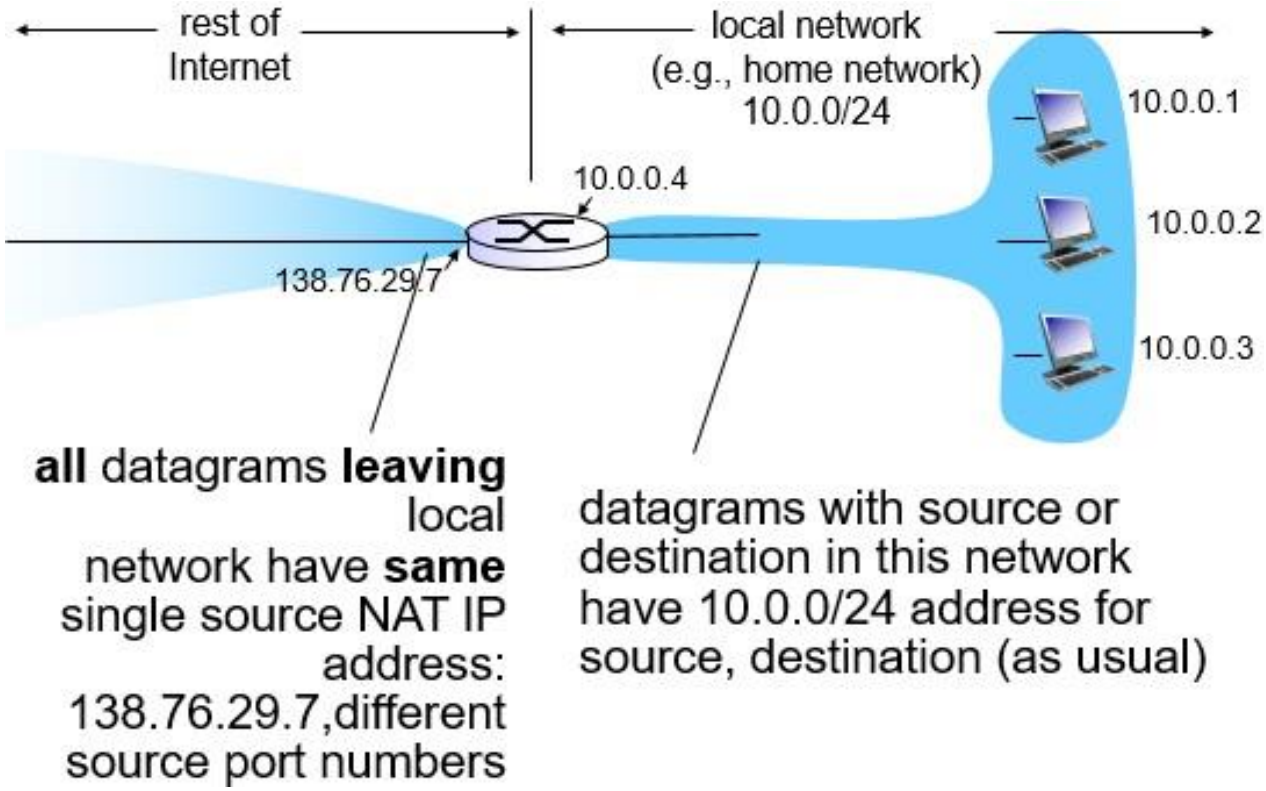
DHCP Client-Server Scenario (1 of 2)



DHCP Client-Server Scenario (2 of 2)



NAT: Network Address Translation (1 of 5)



NAT: Network Address Translation (2 of 5)

motivation: local network uses just one IP address as far as outside world is concerned:

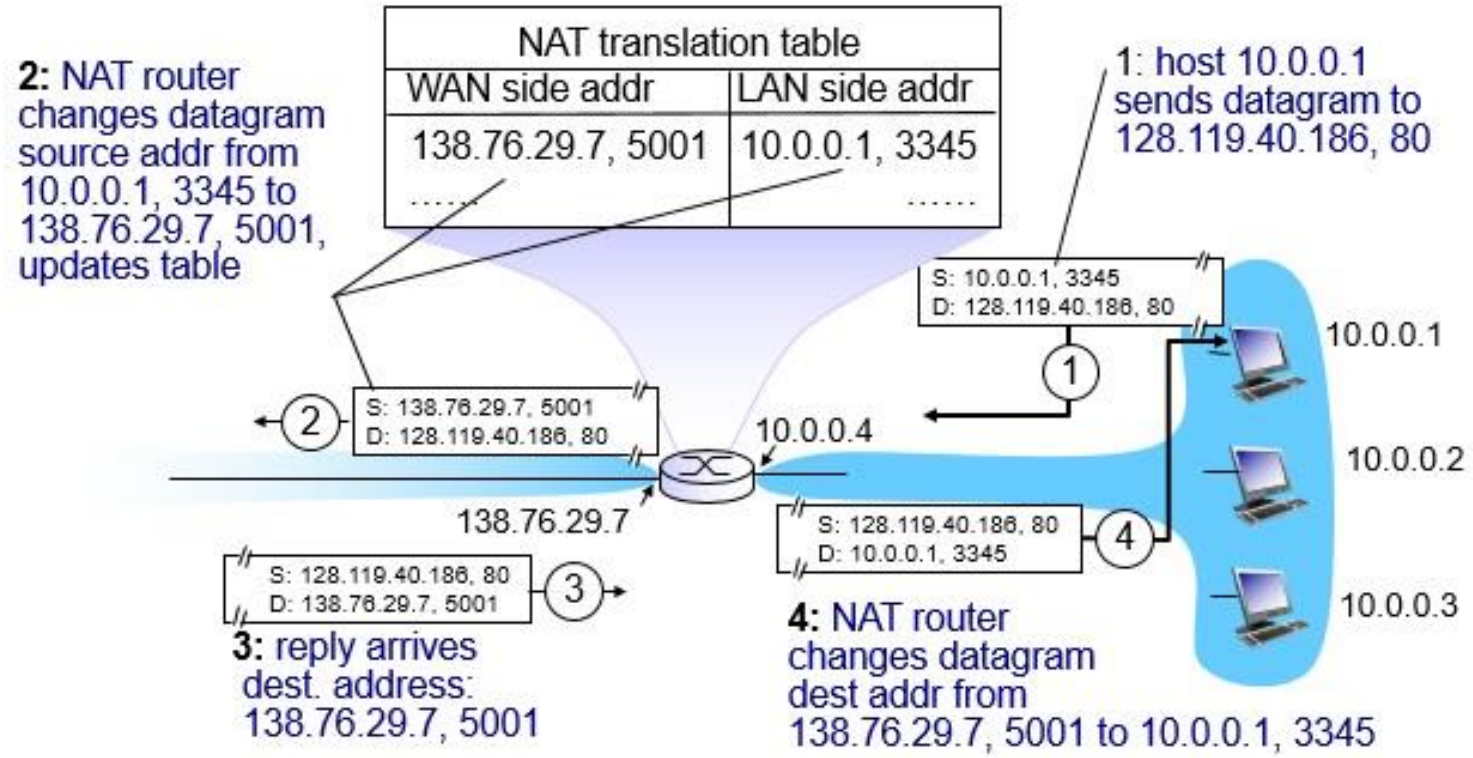
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

NAT: Network Address Translation (3 of 5)

implementation: NAT router must:

- **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 addr
- **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 dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT: Network Address Translation (4 of 5)



NAT: Network Address Translation (5 of 5)

16-bit port-number field:

- 60,000 simultaneous connections with a single LAN-side address!

NAT is controversial:

- routers should only process up to layer 3
- address shortage should be solved by IPv6
- violates end-to-end argument
  - NAT possibility must be taken into account by app designers, e.g., P2P applications
- NAT traversal: what if client wants to connect to server behind NAT?

IPv6: Motivation

- **initial motivation:** 32-bit address space has been exhausted.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:

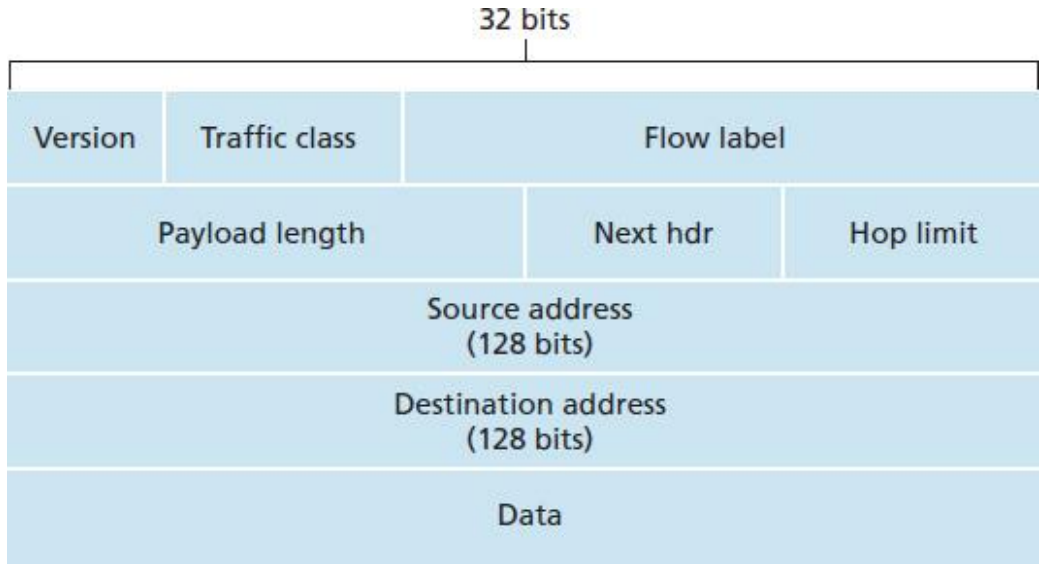
- fixed-length 40 byte header
- no fragmentation allowed

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



Other Changes from IPv4

checksum: removed entirely to reduce processing time at each hop

options: allowed, but outside of header, indicated by “Next Header” field

ICMPv6: new version of ICMP

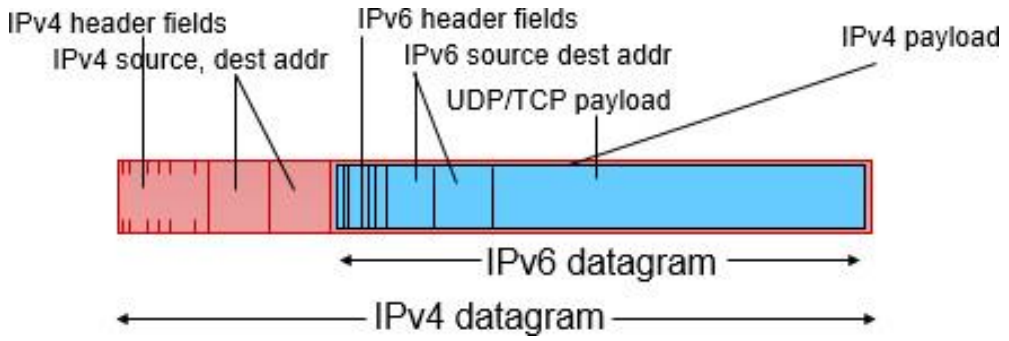
- additional message types, e.g. “Packet Too Big”
- multicast group management functions

Transition from IPv4 to IPv6

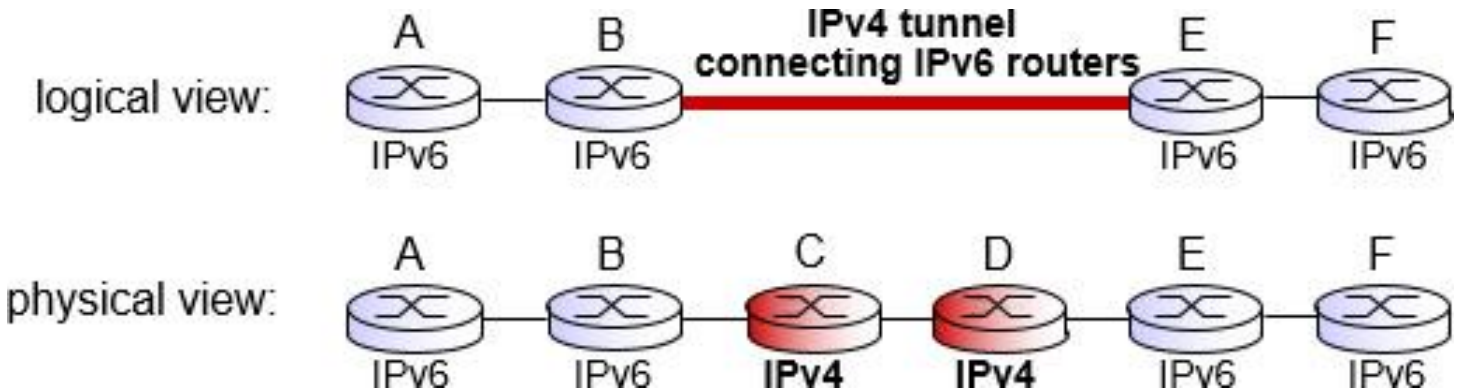
not all routers can be upgraded simultaneously

- no “flag days”
- how will network operate with mixed IPv4 and IPv6 routers?

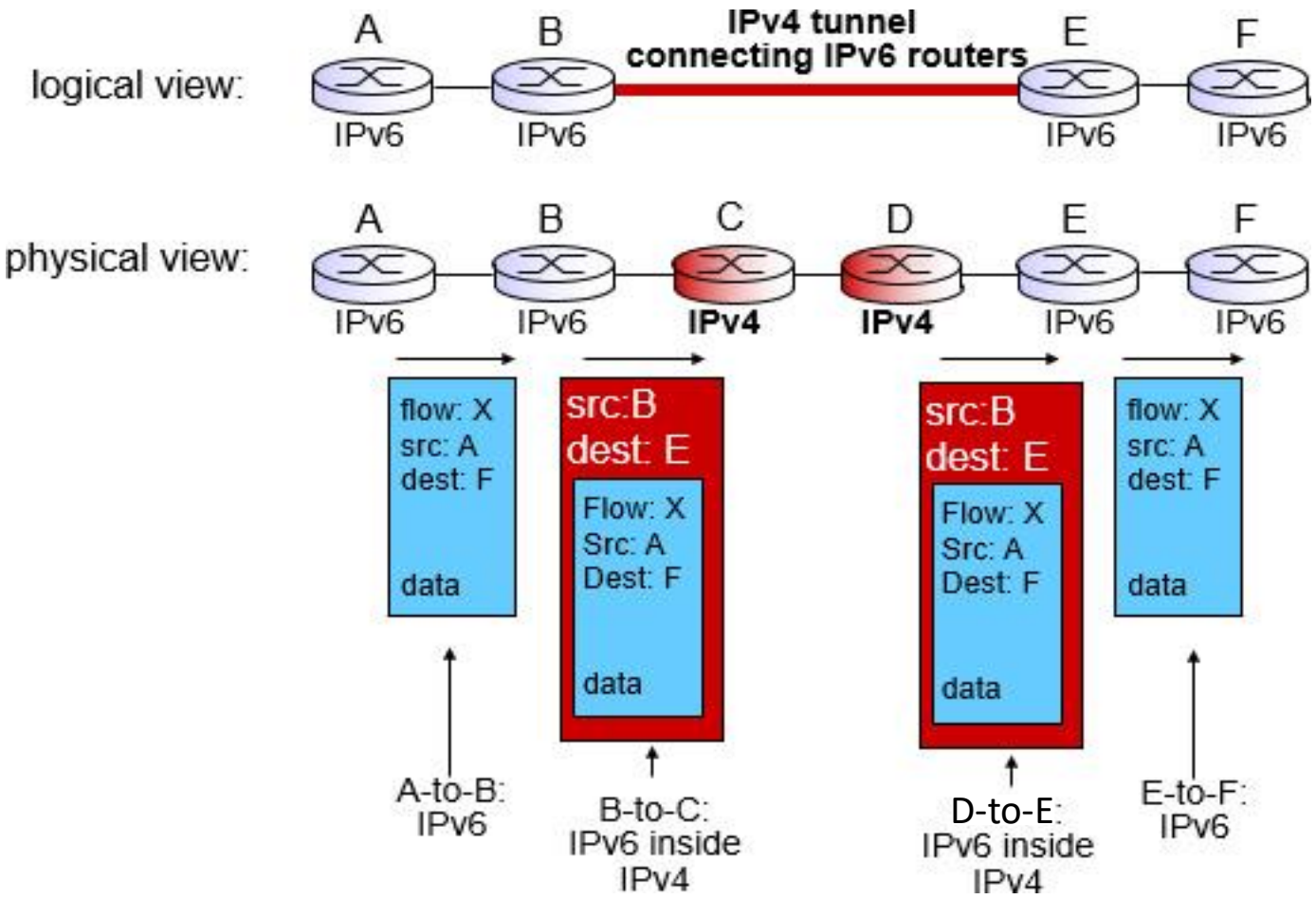
tunneling: IPv6 datagram carried as **payload** in IPv4 datagram among IPv4 routers



Tunneling (1 of 2)



Tunneling (2 of 2)

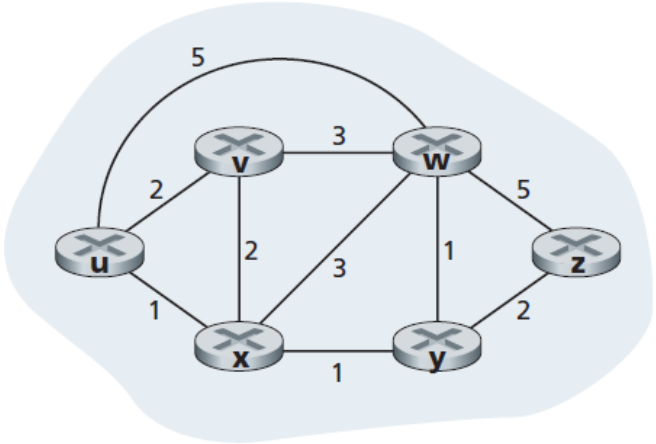


Routing Protocols

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



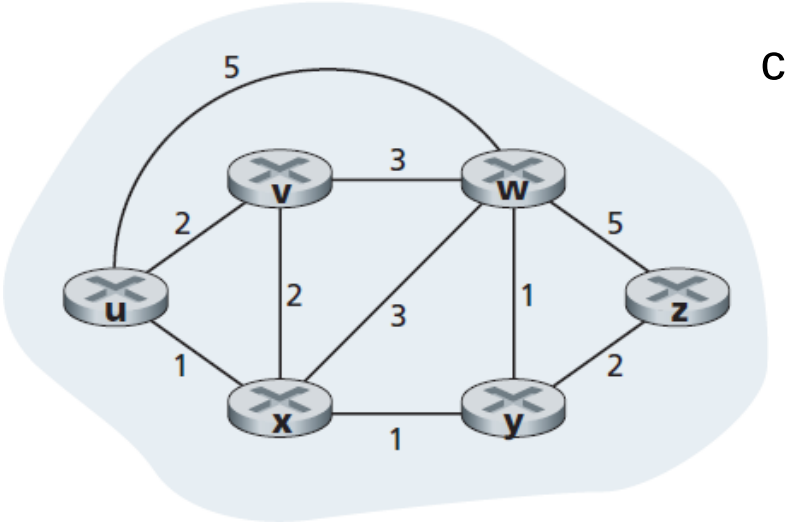
graph: G = (N, E)

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

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

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

- network topology, link costs: known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have the same information
- compute the least-cost paths from one node ("source") to all other nodes
  - gives **forwarding table** for that node
- iterative: after k iterations, know the least-cost paths to k destinations

Notation:

- **c(x,y):** link cost between node x and node y
- If **(x,y)** does not belong to E (that is, x and y are not direct neighbours), then we set **c(x,y)** = ∞
- **D(v):** current cost of the least-cost path from the source to destination **v** (as of this iteration of the routing algorithm)
- **p(v):** previous node (neighbour of **v**) along the current least-cost path from the source to **v**
- **N':** a subset of nodes whose least-cost path is definitively known

Dijsktra's Algorithm

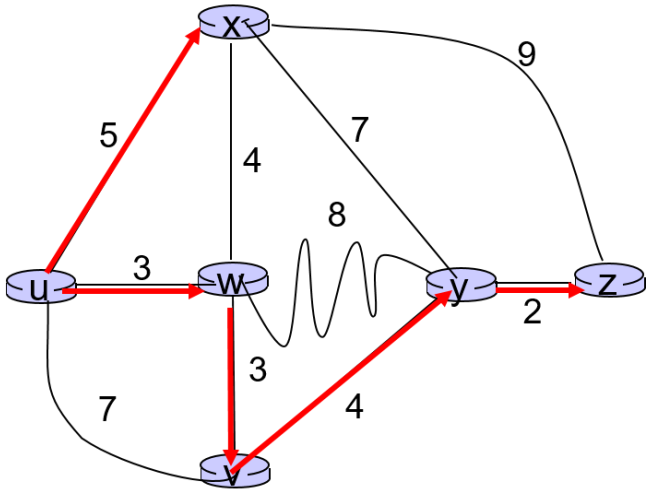
```
1  Initialization:
2   N' = {u}
3   for all nodes v
4    if v adjacent to u
5       then D(v) = c(u,v)
6    else D(v) = ∞
7
8  Loop
9   find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N' :
12     D(v) = min( D(v), D(w) + c(w,v) )
13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

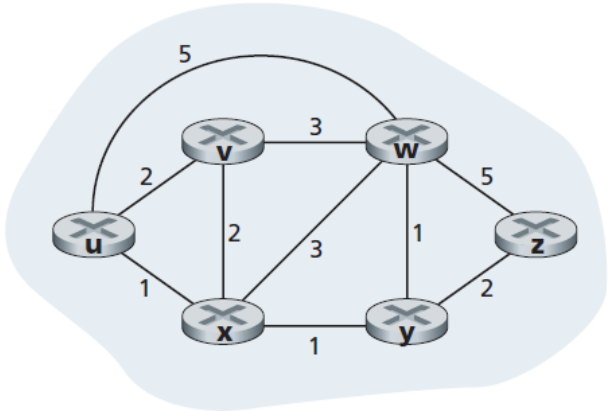
Notes:

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



Dijkstra's Algorithm: Another Example

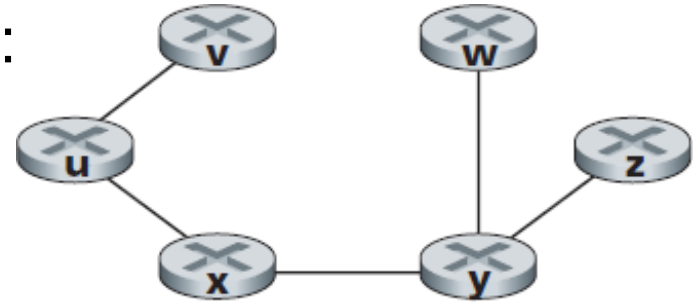
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



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

Dijkstra's Algorithm: Example (2 of 2)

- The resulting **least-cost paths** from u:



- The resulting **forwarding table** in u:

Destination	Link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Distance Vector Algorithm (1 of 6)

Bellman-Ford equation (dynamic programming)

let

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

then

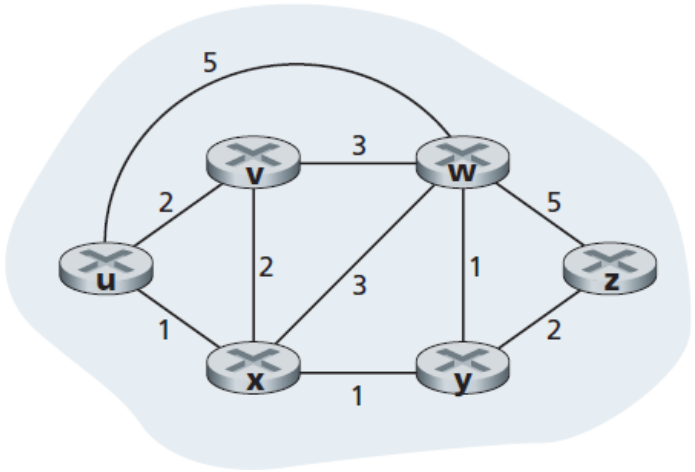
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

$d_v(y)$: cost from neighbor v to destination y

$c(x,v)$: cost to neighbor v

$\min_v$: **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:

$$
\begin{aligned}
d_u(z) = \min\{\, & c(u,v) + d_v(z), \\
& c(u,x) + d_x(z), \\
& c(u,w) + d_w(z)\,\} \\
= \min\{\, & 2+5, \\
& 1+3, \\
& 5+3\,\} = 4
\end{aligned}
$$

Node achieving minimum is the next hop in the shortest path, used in the forwarding table

Distance Vector Algorithm (2 of 6)

Initialisation step:

- Each node x begins with $D_x(y)$, an estimate of the cost of the least-cost path from x to y, for all nodes y in N.
- Let $\mathbf{D}_x = [D_x(y): y \text{ in } N]$ be node x's distance vector of cost estimates from x to all other nodes y in N.
- Each node x maintains the following routing information:
  - The cost c(x,v) from x to each directly attached neighbor v
  - Node x's distance vector $\mathbf{D}_x = [D_x(y): y \text{ in } N]$
  - The distance vectors of each of its neighbours v, that is, $\mathbf{D}_v = [D_v(y): y \text{ in } N]$

Distance Vector Algorithm (3 of 6)

Update step:

- From time-to-time, each node sends a copy of its own distance vector to its neighbours
- When a node x receives a new distance vector from its neighbor w, it saves w's distance vector and updates its own distance vector using B-F equation: $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ for each node $y \in N$
- If node x's distance vector has changed as a result of this update step, node x sends its updated distance vector to each of its neighbours, which in turn update their own distance vectors.
- Miraculously, each cost **estimate** $D_x(y)$ ***converges*** to $d_x(y)$, the **actual** cost of the least-cost path from node x to node y!

Distance Vector Algorithm (4 of 6)

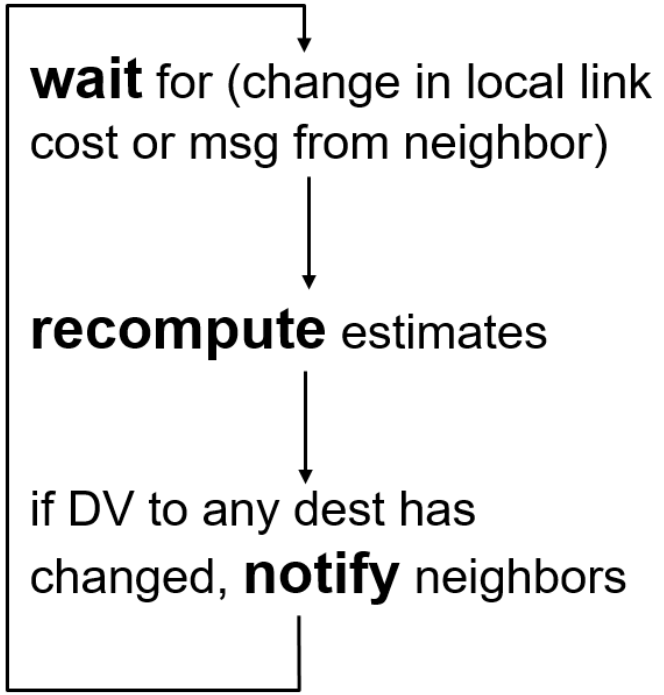
iterative, asynchronous*:* each local iteration caused by:

- local link cost change
- Distance vector update message from neighbour

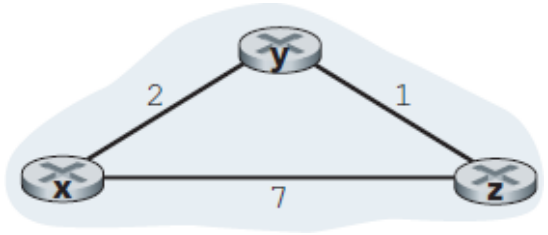
distributed:

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

each node:



Distance Vector Algorithm (5 of 6)



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

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

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

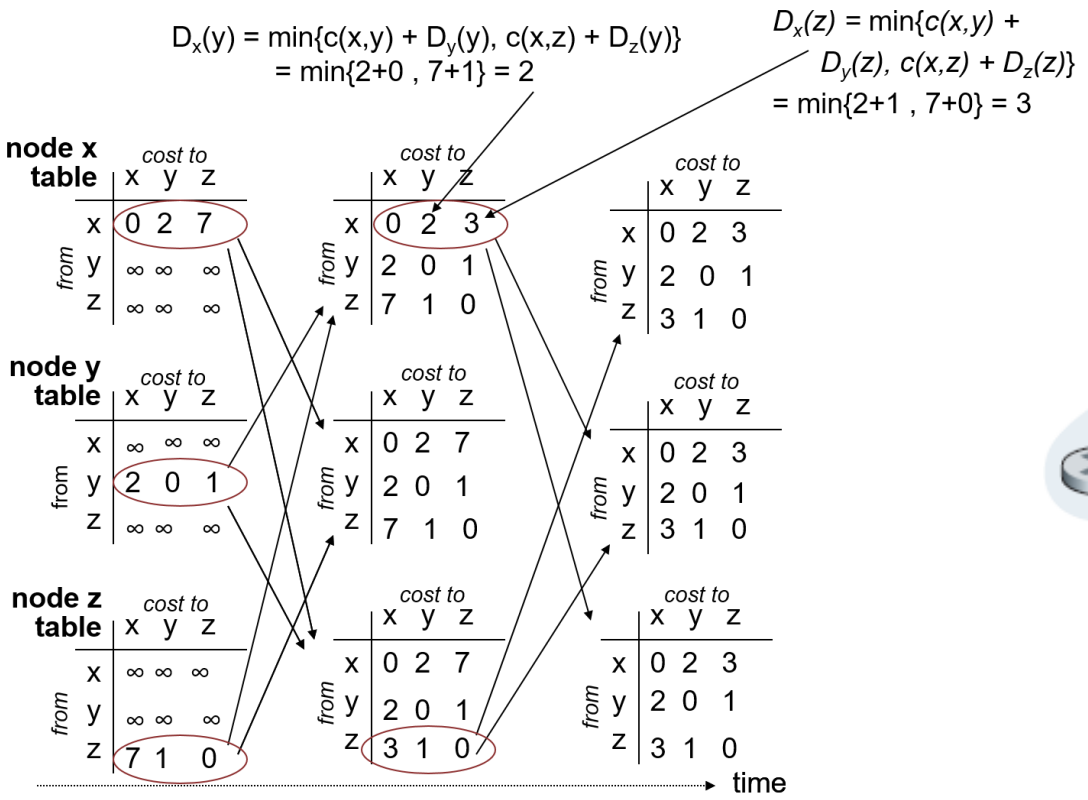
from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

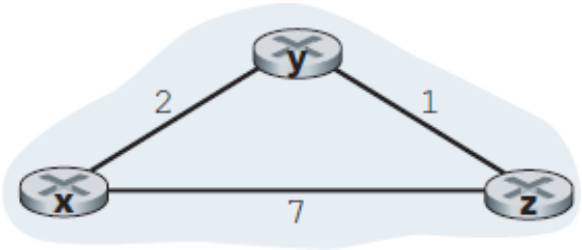
node z table

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

time

Distance Vector Algorithm (6 of 6)





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

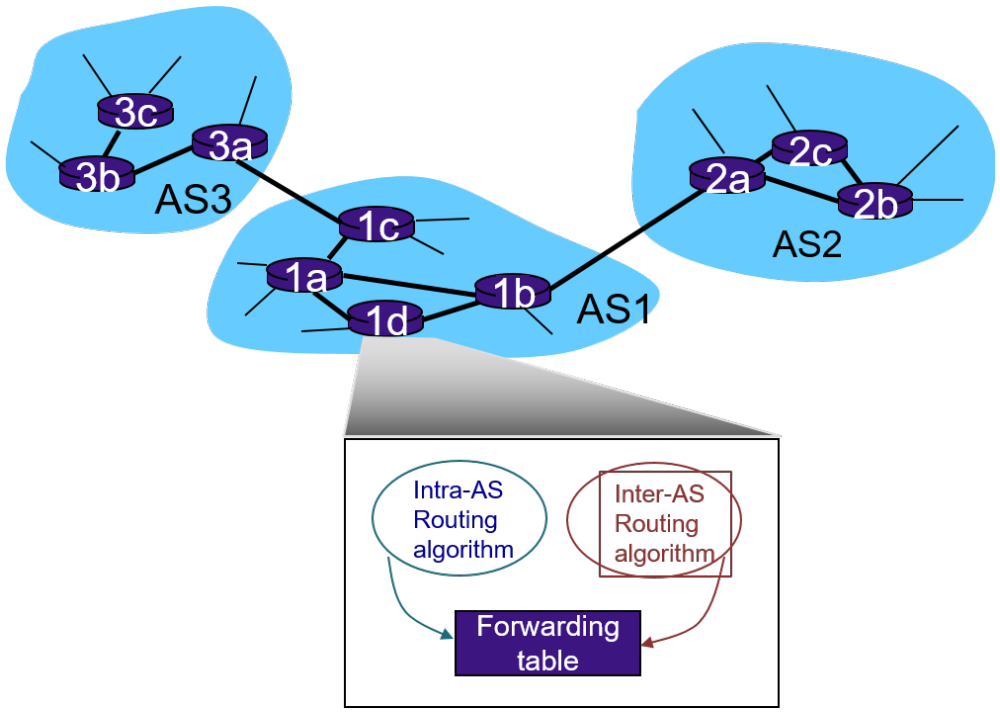
- routing among hosts, routers in same AS ("network")
- all routers in AS must run **same** intra-domain protocol
- routers in **different** AS can run **different** intra-domain routing protocol
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

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



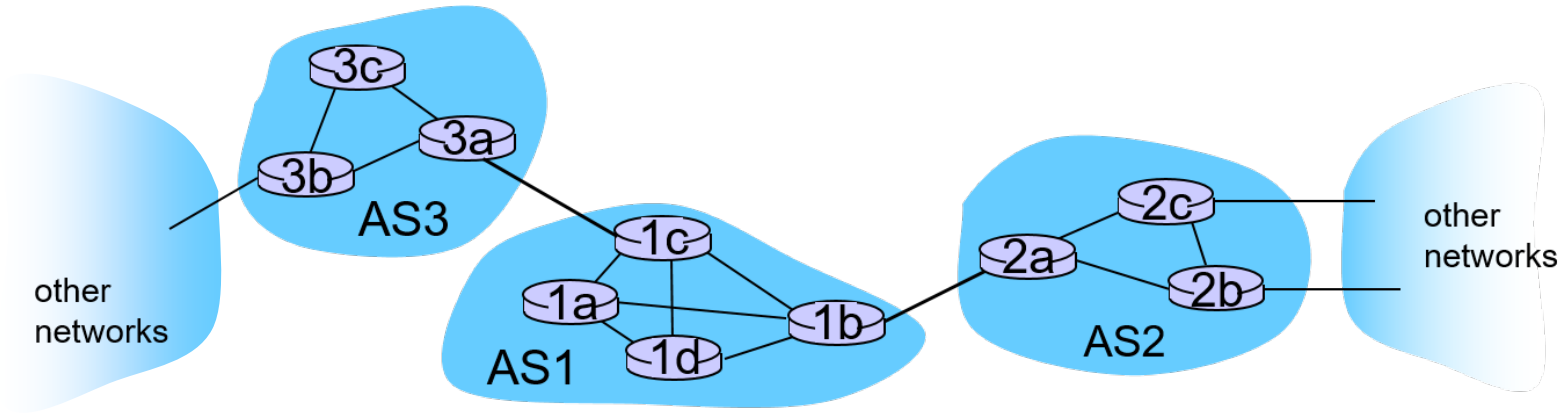
Inter-AS Tasks

- 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)

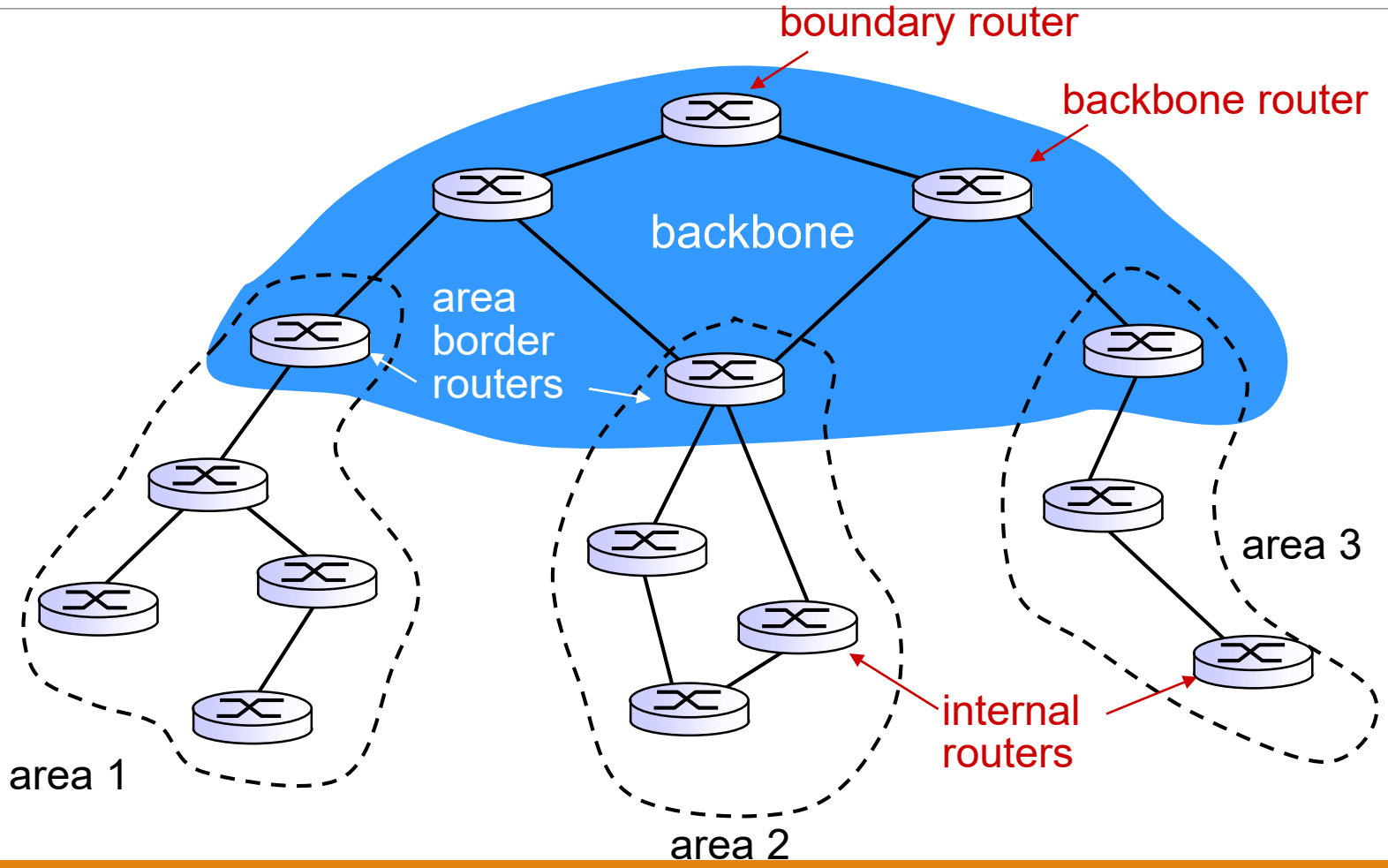
OSPF (Open Shortest Path First)

- "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 database as OSPF
- **hierarchical** OSPF in large domains.

Hierarchical OSPF (1 of 2)



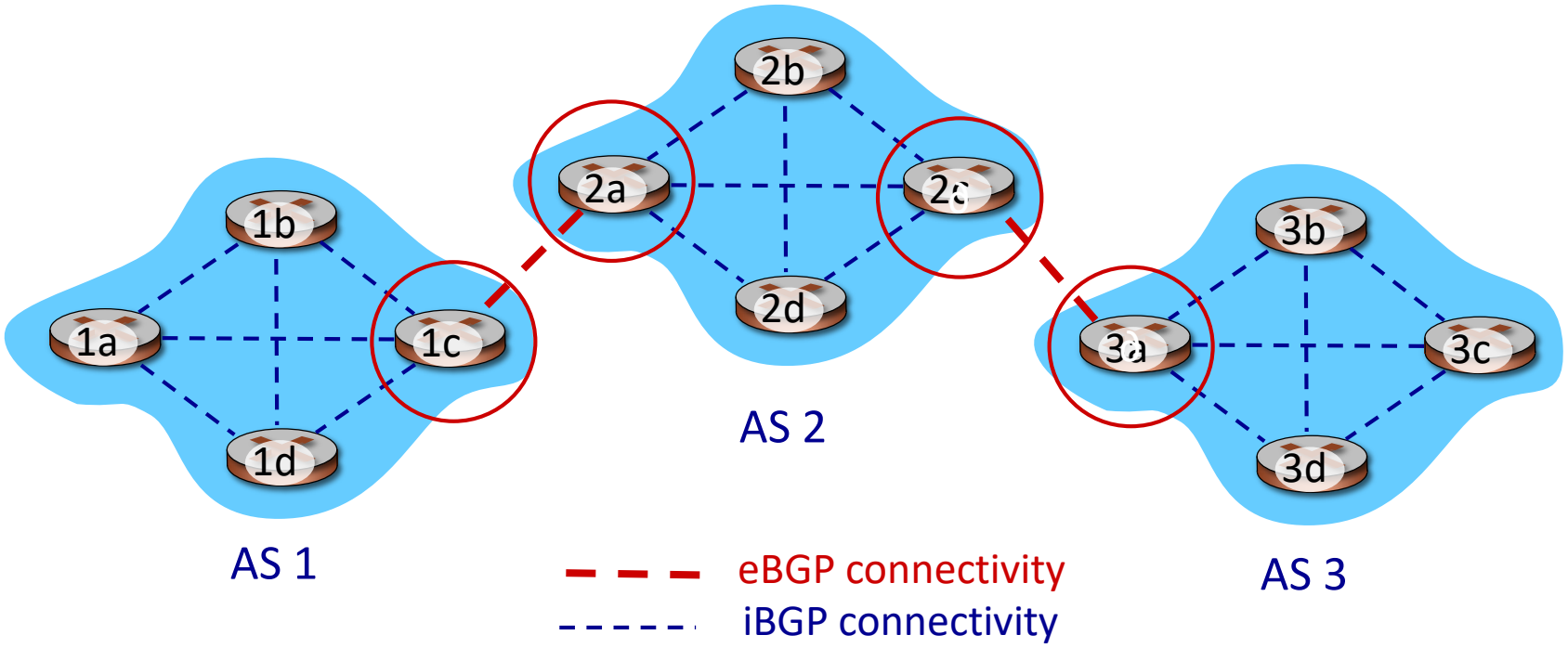
Hierarchical OSPF (2 of 2)

- **two-level hierarchy:** local area, backbone.
  - link-state advertisements only in area
  - each node 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.

Internet inter-AS Routing: BGP

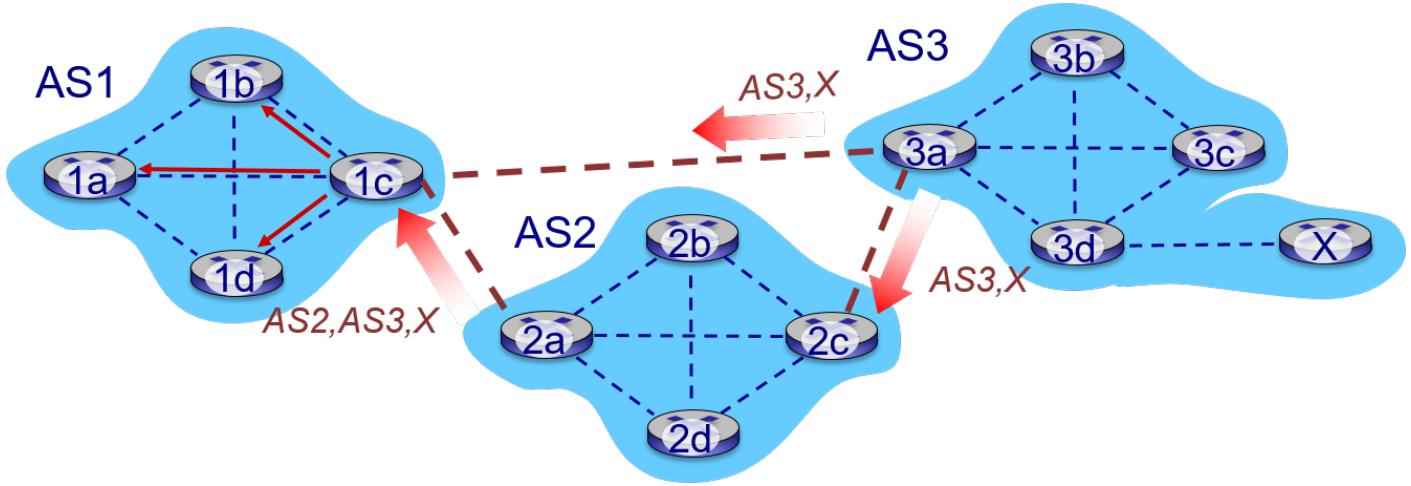
- **BGP (Border Gateway Protocol):** the *de facto* inter-domain routing protocol
  - "glue that holds the Internet together"
- BGP provides each router a means to:
  - **eBGP:** obtain subnet reachability information from neighboring ASes
  - **iBGP:** propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and **policy**
- allows subnet to advertise its existence to the rest of Internet: **"I am here"**

eBGP, iBGP Connections



gateway routers run both eBGP and iBGP protocols

BGP Path Advertisement (3 of 3)



Gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path **AS2, AS3, X** from 2a
- AS1 gateway router 1c learns path **AS3, X** from 3a
- Based on policy, AS1 gateway router 1c chooses path **AS3, X, and advertises path within AS1 via iBGP**

Path Attributes and BGP Routes

- Question: Potentially many paths from a given router to a destination subnet. How does a router choose among these paths (and configure it forwarding table accordingly)?
- advertised prefix includes BGP attributes
  - prefix + attributes = “route”
- two important attributes:
  - **AS-PATH:** list of ASes through which prefix advertisement has passed
  - **NEXT-HOP:** IP address of the router interface that begins the AS-PATH
- Each BGP route is written as a list with three components: NEXT-HOP; AS-PATH; destination prefix.
- **Policy-based routing:**
  - gateway receiving route advertisement uses **import policy** to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to **advertise** path to other other neighboring ASes

BGP Route Selection Algorithm

- In practice, BGP uses an algorithm that is more complicated than Hot Potato routing!
- A router may learn about **more than one route** to destination AS, BGP sequentially invokes elimination rules until one route remains.
  1. Assign a route with a local preference value (policy decision). Select routes with the highest local preference values.
  2. Among all routes with the same highest local preference value, select routes with shortest AS–PATH (by DV algorithm, using number of AS hops as distance metric)
  3. From the remaining routes, select the closest NEXT-HOP router: hot potato routing!
  4. If more than route still remains, use additional criteria, e.g., BGP identifiers

BGP Routing Policy

- When a router selects a route to a destination, the AS routing policy can trump all other considerations, such as shortest AS path or hot potato routing
- See the first step of BGP route selection algorithm

Data Link Layer: Error Detection, Multiple Access

DR DUY NGO

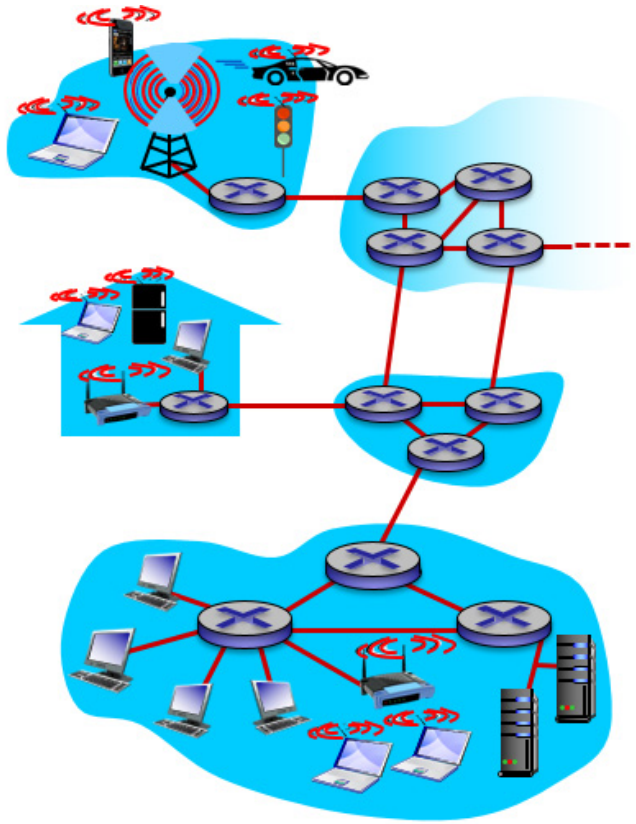
SCHOOL OF ELECTRICAL ENGINEERING &COMPUTING

Link Layer: Introduction

terminology:

- hosts and routers: **nodes**
- communication channels that connect adjacent nodes along communication path: **links**
  - wired links
  - wireless links
  - LANs
- layer-2 packet: **frame,** encapsulates datagram

data-link layer has responsibility of transferring datagram from one node to **physically adjacent** node over a link



Link Layer: Context

- datagram transferred by different link protocols over different links:
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- each link protocol provides different services
  - e.g., may or may not provide rdt over link

transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = **datagram**
- transport segment = **communication link**
- transportation mode = **link layer protocol**
- travel agent = **routing algorithm**

Link Layer Services (1 of 2)

framing, link access:

- encapsulate datagram into frame, adding header, trailer
- channel access if shared medium
- "MAC" addresses used in frame headers to identify source, destination
  - different from IP address!

reliable delivery between adjacent nodes

- we learned how to do this already (chapter 3)!
- seldom used on low bit-error link (fiber, some twisted pair)
- wireless links: high error rates
  - **Q:** why both link-level and end-end reliability?

Link Layer Services (2 of 2)

flow control:

- pacing between adjacent sending and receiving nodes

error detection:

- errors caused by signal attenuation, noise.
- receiver detects presence of errors:
  - signals sender for retransmission or drops frame

error correction:

- receiver identifies **and corrects** bit error(s) without resorting to retransmission

half-duplex and full-duplex

- with half duplex, nodes at both ends of link can transmit, but not at same time

Where is the Link Layer Implemented?

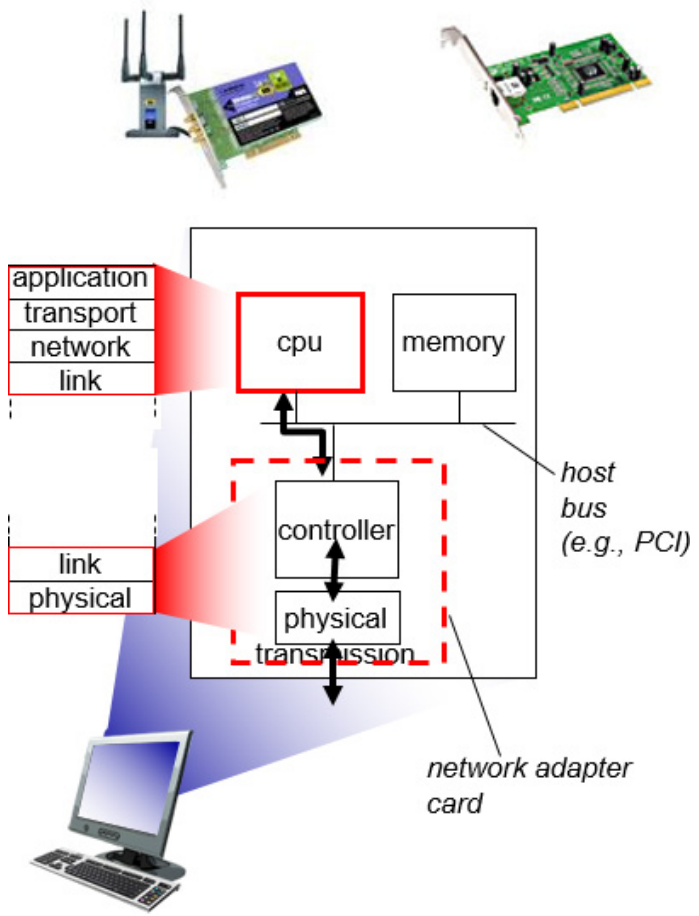
in each and every host

link layer implemented in “adaptor” (aka **network interface card** NIC) or on a chip

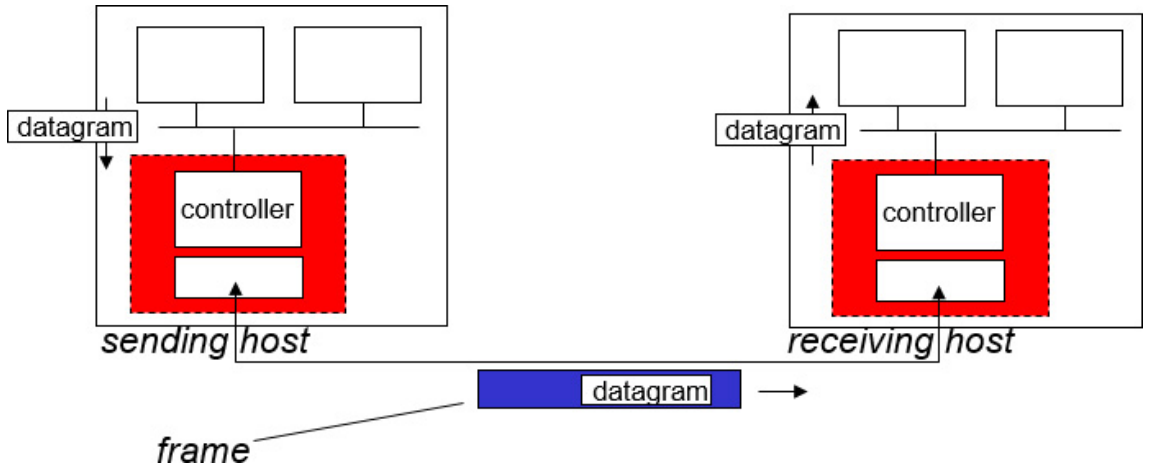
- Ethernet card, 802.11 card; Ethernet chipset
- implements link, physical layer

attaches into host’s system buses

combination of hardware, software, firmware



Adaptors Communicating



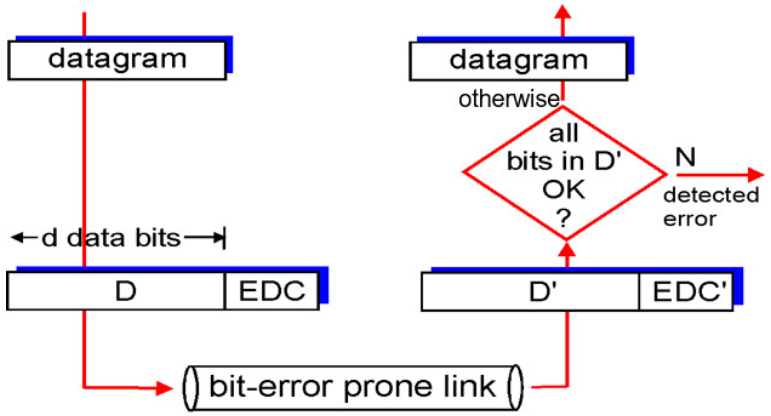
- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc.
  - extracts datagram, passes to upper layer at receiving side

Error Detection

EDC = Error Detection and Correction bits (redundancy)

D = Data protected by error checking, may include header fields

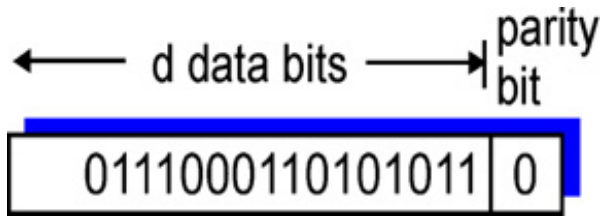
- Error detection not 100% reliable!
  - protocol may miss some errors, but rarely
  - larger EDC field yields better detection and correction



Parity Checking

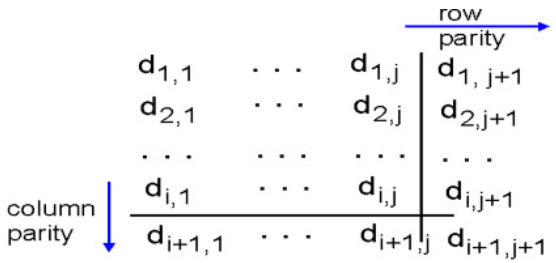
single bit parity:

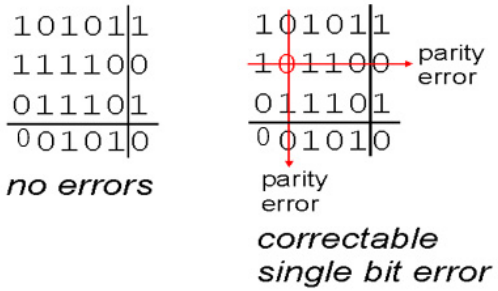
detect single bit errors



two-dimensional bit parity:

detect and correct single bit errors





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

Multiple Access Links, Protocols

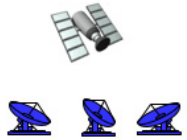
two types of “links”:

point-to-point
- PPP for dial-up access
- point-to-point link between Ethernet switch, host

broadcast (shared wire or medium)
- old-fashioned Ethernet
- upstream HFC
- 802.11 wireless LAN

shared wire (e.g., cabled Ethernet)

shared RF (e.g., 802.11 WiFi)



shared RF (satellite)

humans at a cocktail party (shared air, acoustical)

Multiple Access Protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - **collision** if node receives two or more signals at the same time

multiple access protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

An Ideal Multiple Access Protocol

given: broadcast channel of rate R bps

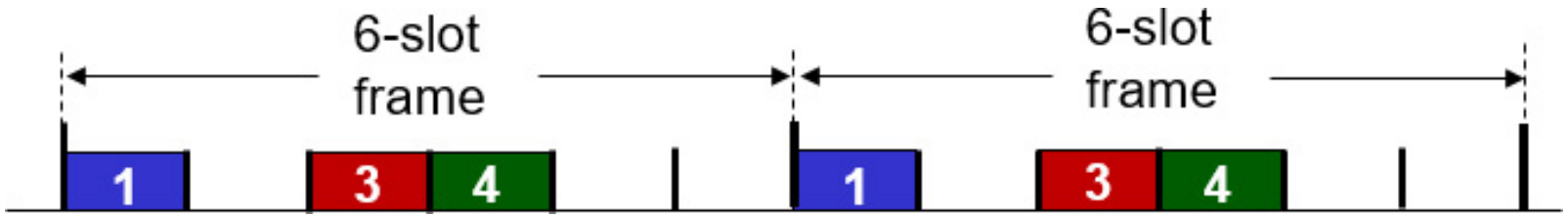
MAC scenario:

1. when one node wants to transmit, it can send at rate R.
2. when M nodes want to transmit, each can send at average rate R/M
3. fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

Channel Partitioning MAC Protocols: TDMA

TDMA: time division multiple access

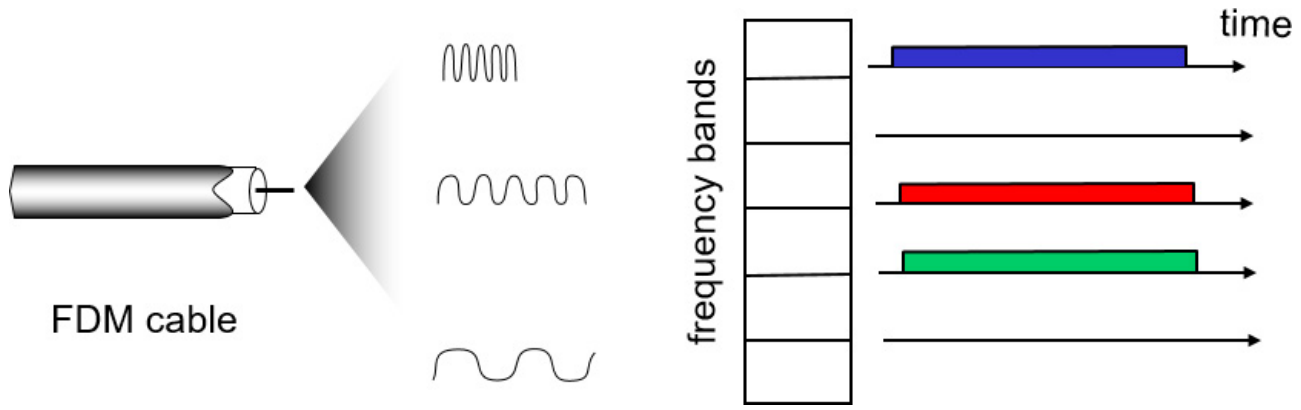
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel Partitioning MAC Protocols: FDMA

FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random Access Protocols

- when node has packet to send
  - transmit at full channel data rate R.
  - no *a* **priori** coordination among nodes
- 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

Slotted ALOHA

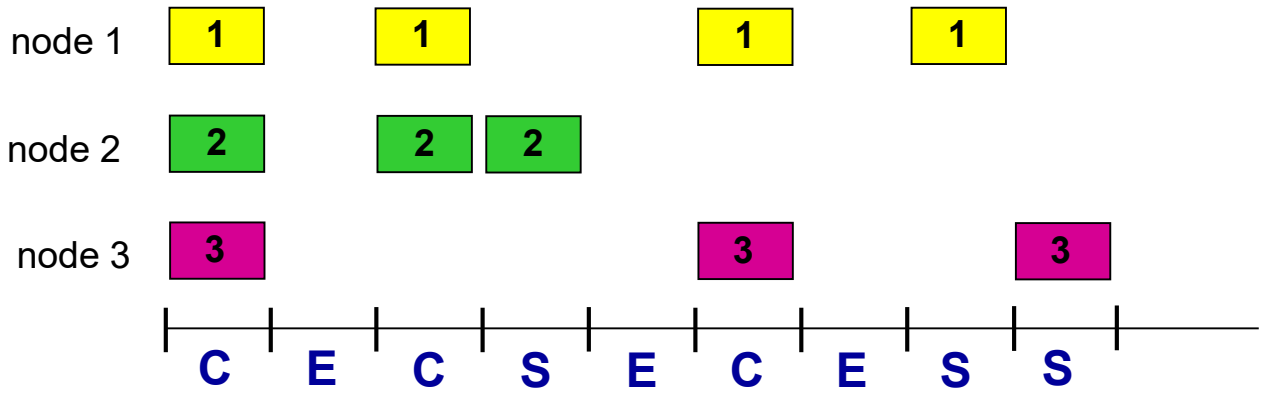
assumptions:

- all frames same size
- time divided into equal size slots (time to transmit one frame)
- nodes start to transmit only slot beginning
- nodes are synchronised
- 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 a new frame in the next slot
  - *if collision:* node retransmits the frame in each subsequent slot with probability *p* until success

Slotted ALOHA



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)

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

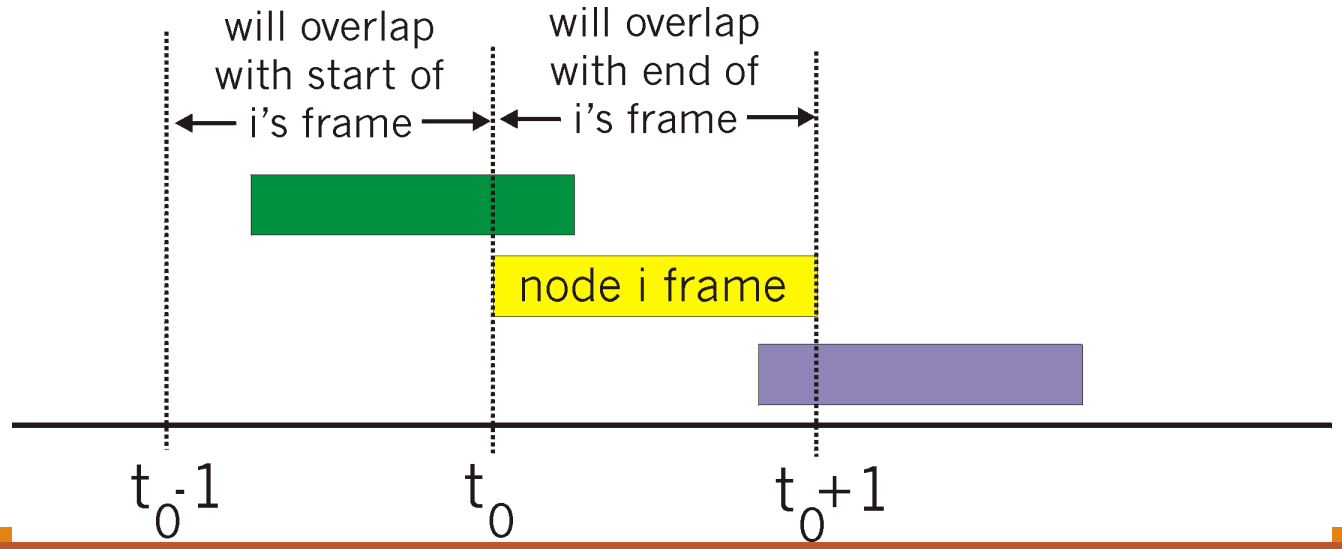
max efficiency = 1/e = .37

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

!

Pure (unslotted) ALOHA

- unslotted Aloha: simpler, no synchronization
- when frame first arrives
  - transmit immediately
- collision probability increases:
  - 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 $[t_0\text{-}1, t_0]$)

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

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

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

Maximum efficiency = 1/(2e) = .18

even *worse* than slotted Aloha!

CSMA (Carrier Sense Multiple Access)

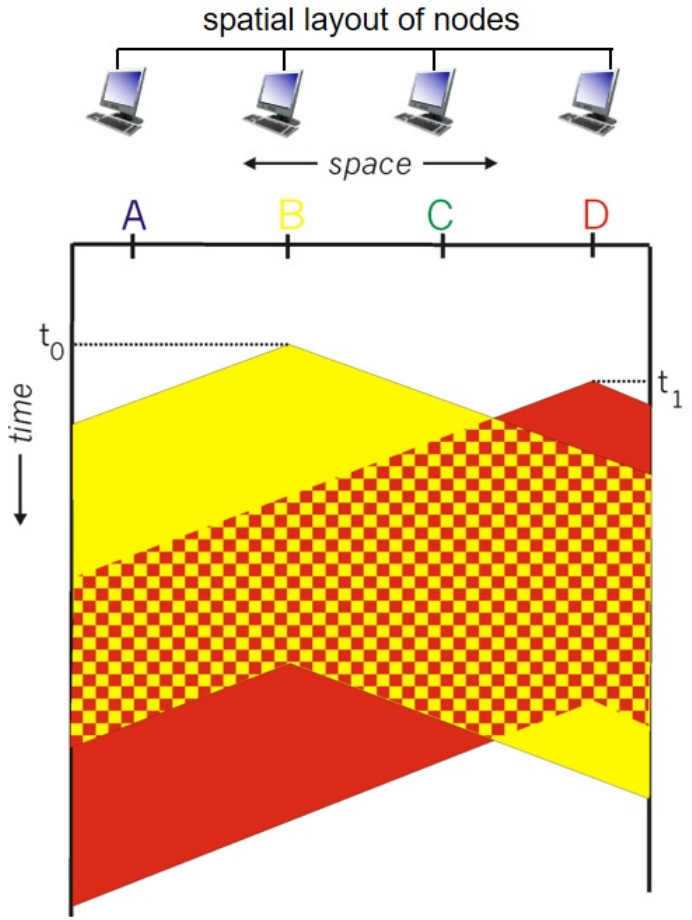
CSMA: listen before transmit:

- **if channel sensed idle:** transmit entire frame
- **if channel sensed busy,** defer transmission

human analogy: “don’t interrupt others!”

CSMA Collisions

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



CSMA/CD (Collision Detection) (1 of 2)

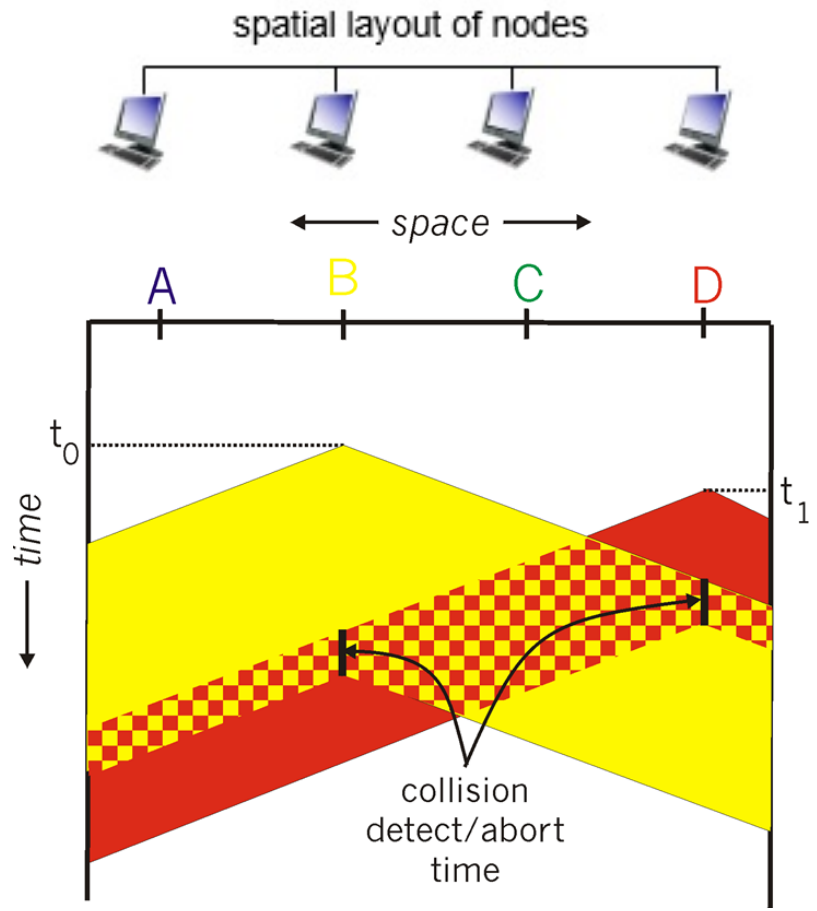
CSMA/CD: carrier sensing, deferral as in CSMA

- collisions **detected** within short time
- colliding transmissions aborted, reducing channel wastage

- collision detection:
  - easy in wired LANs: measure signal strengths, compare transmitted, received signals
  - difficult in wireless LANs: received signal strength overwhelmed by local transmission strength

- human analogy: the polite conversationalist

CSMA/CD (Collision Detection) (2 of 2)



Ethernet CSMA/CD Algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame!
4. If NIC detects another transmission while transmitting, aborts transmission
5. After aborting, NIC enters **binary (exponential) backoff:**

   – after *m*th collision, NIC chooses *K* at random from $\{0, 1, 2, \ldots, 2^m - 1\}$.

   NIC waits K·512 bit times, returns to Step 2

   – longer backoff interval with more collisions

CSMA/CD Efficiency

- t_{prop} = max prop delay between 2 nodes in LAN
- t_{trans} = time to transmit max-size frame

$$efficiency = \frac{1}{1 + 5t_{prop} / t_{trans}}$$

- efficiency goes to 1
  - as t_{prop} goes to 0
  - as t_{trans} goes to infinity
- better performance than ALOHA, and simple, cheap, decentralised!

“Taking Turns” MAC Protocols (1 of 3)

channel partitioning MAC protocols:

- share channel **efficiently and fairly** at high load
- inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

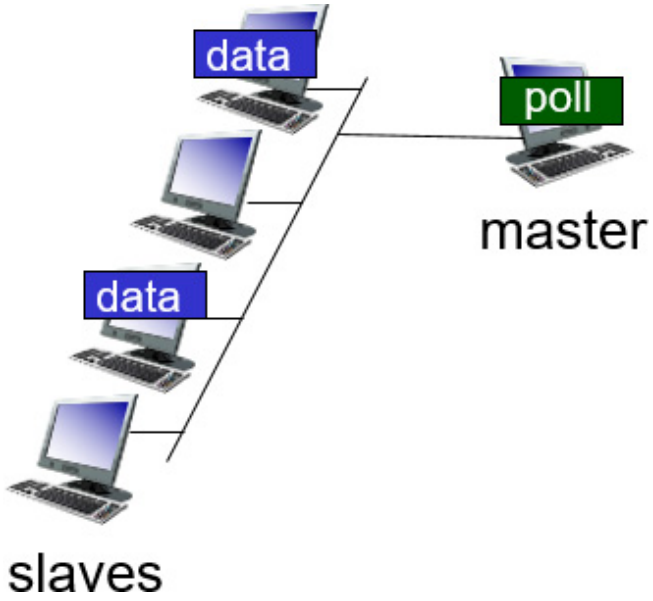
“taking turns” protocols

- look for best of both worlds!

“Taking Turns” MAC Protocols (2 of 3)

polling:

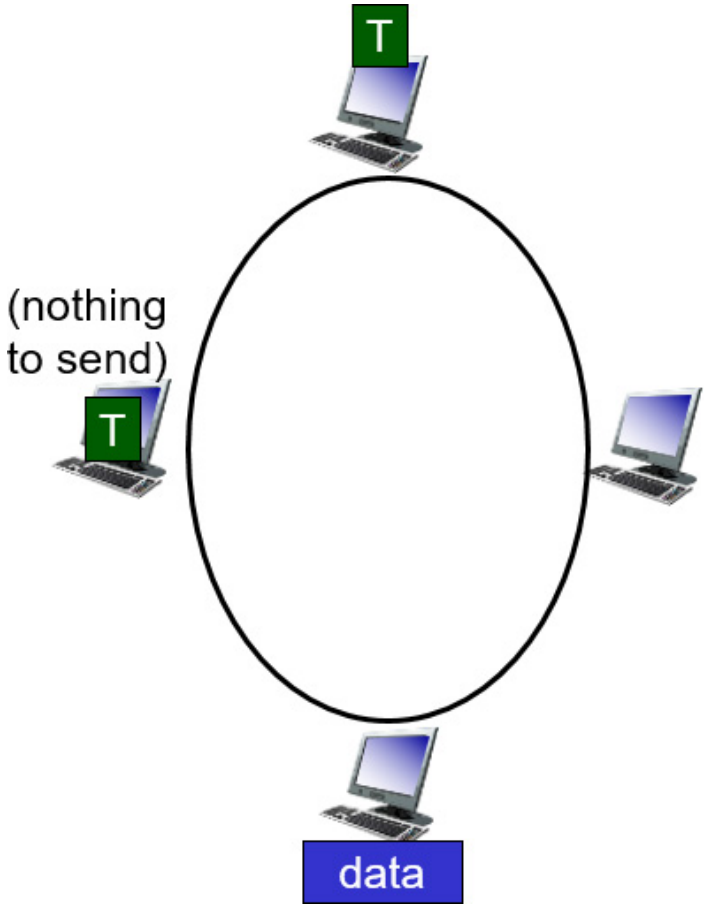
- master node “invites” slave nodes to transmit in turn
- typically used with “dumb” slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



"Taking Turns" MAC Protocols (3 of 3)

token passing:

- control ***token*** passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)



Summary of MAC Protocols

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

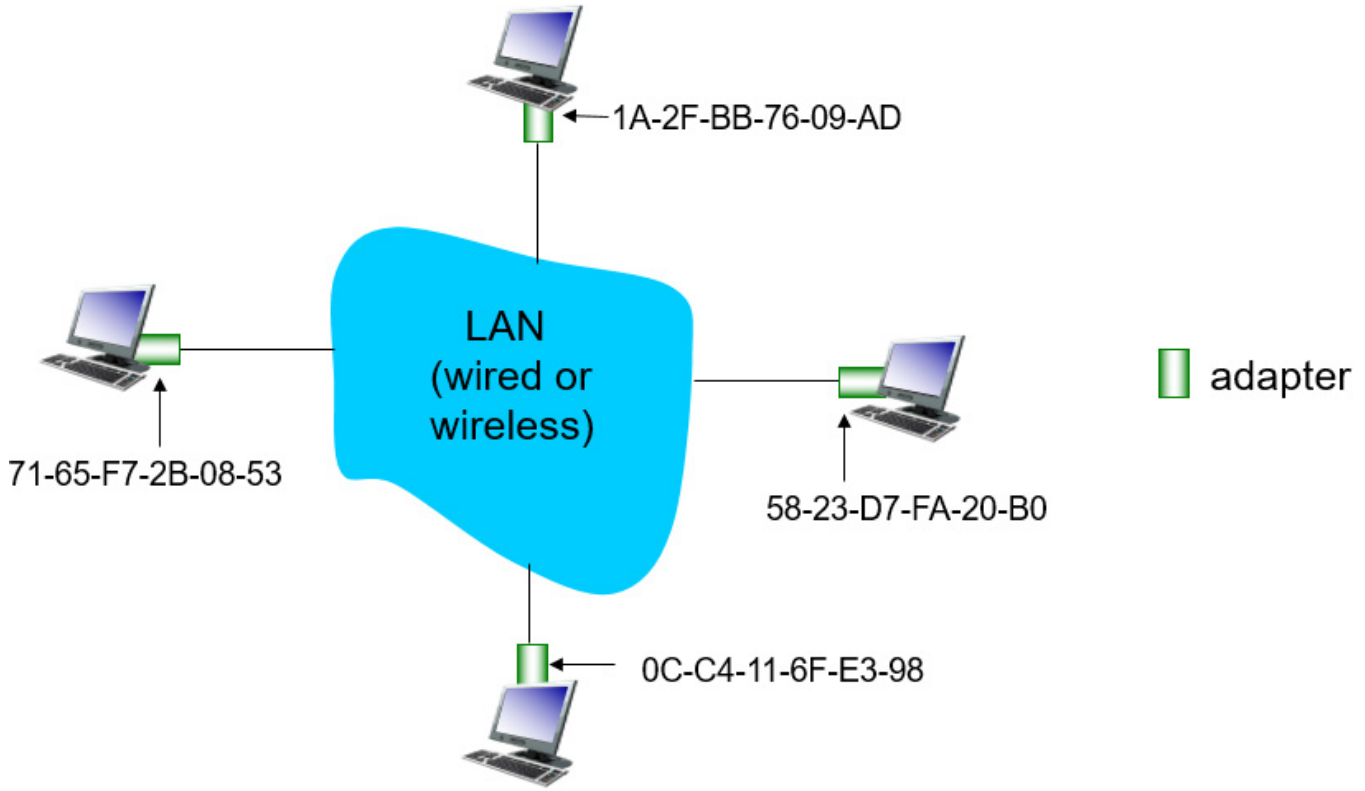
MAC Addresses and ARP

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

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

LAN Addresses and ARP

each adapter on LAN has unique **LAN** address

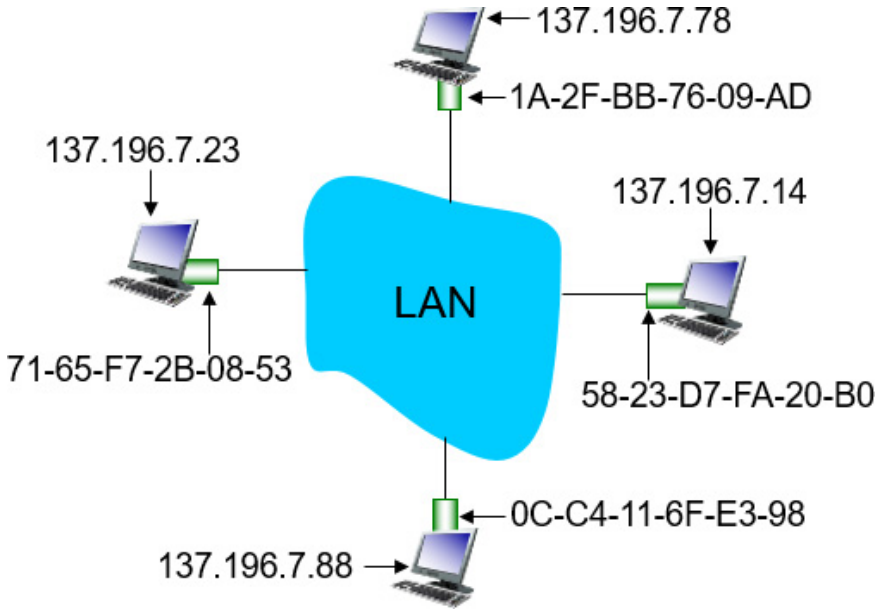


LAN Addresses

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like Social Security Number
  - IP address: like postal address
- MAC flat address → portability
  - can move LAN card from one LAN to another
- IP hierarchical address **not** portable
  - address depends on IP subnet to which node is attached

ARP: Address Resolution Protocol

Question: how to determine interface's MAC address, knowing its IP address?



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

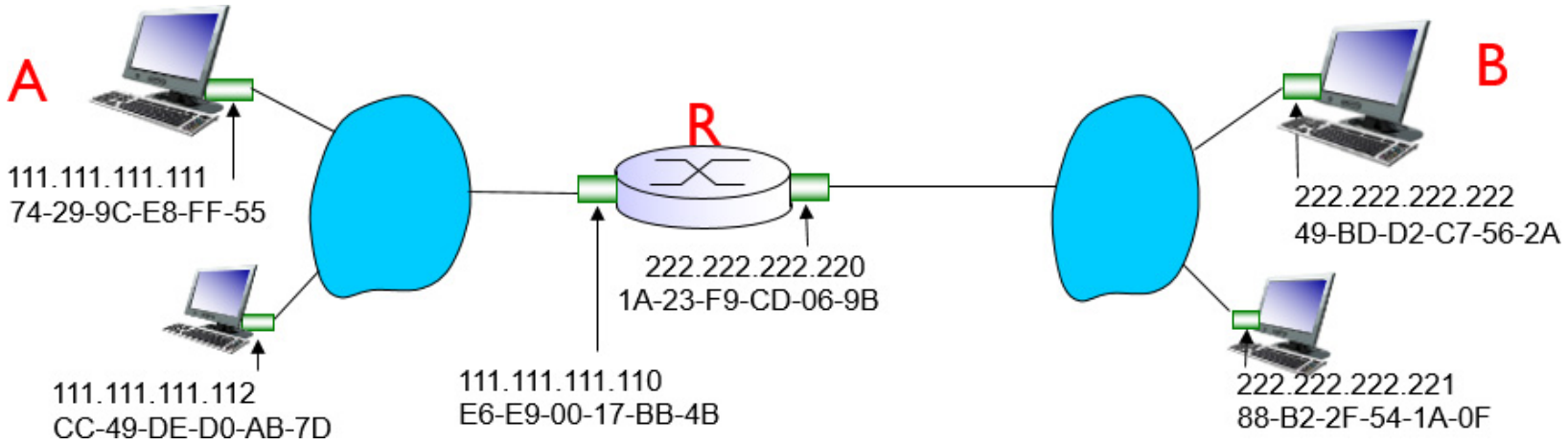
ARP Protocol: Same LAN

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A **broadcasts** ARP query packet, containing B's IP address
  - destination MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed
- ARP is "plug-and-play":
  - nodes create their ARP tables **without intervention from net administrator**

Addressing: Routing to Another LAN (1 of 5)

walkthrough: **send datagram from A to B via R**

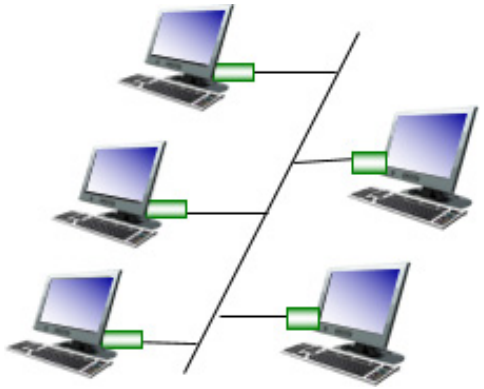
- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)



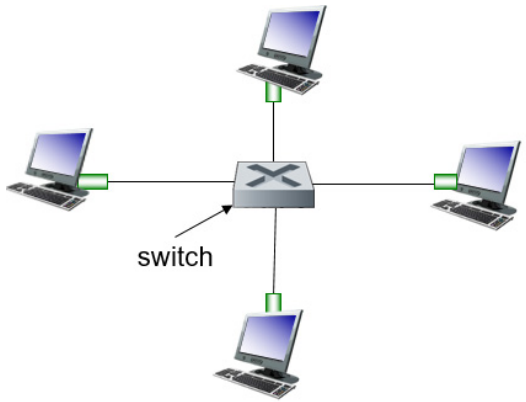
Ethernet: Physical Topology

- **bus:** popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- **star:** prevails today
  - active **switch** in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

bus: coaxial cable

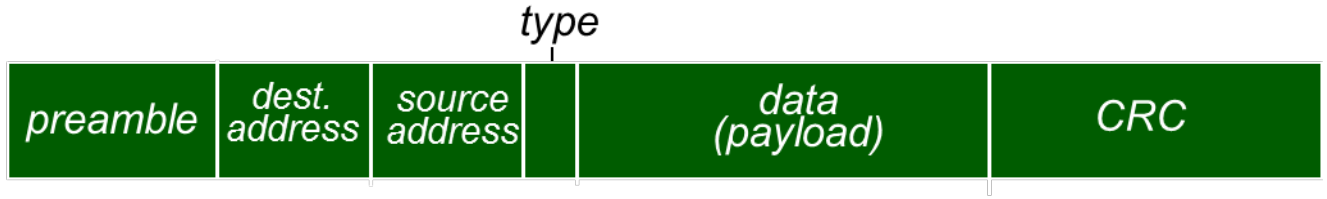


star



Ethernet Frame Structure (1 of 2)

- sending adapter encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**

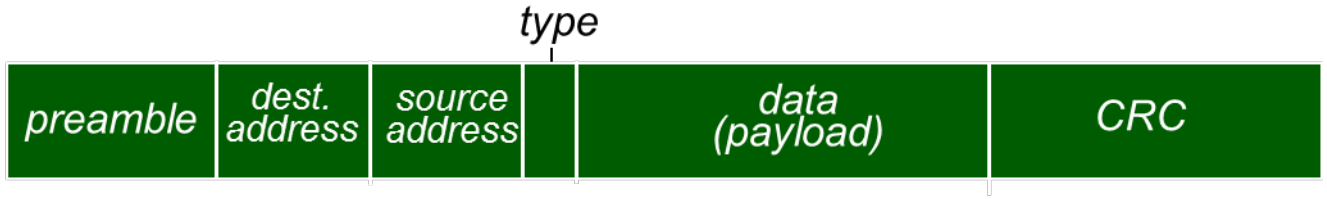


preamble:

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates

Ethernet Frame Structure (2 of 2)

- **addresses:** 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- **type:** indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- **CRC:** cyclic redundancy check at receiver
  - error detected: frame is dropped

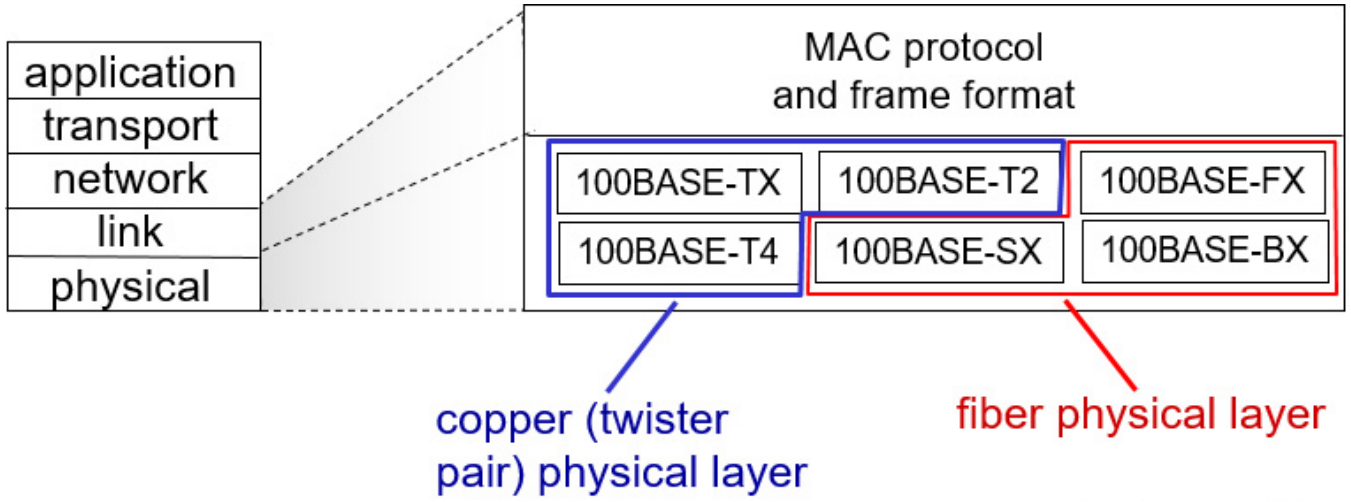


Ethernet: Unreliable, Connectionless

- **connectionless:** no handshaking between sending and receiving NICs
- **unreliable:** receiving NIC doesn't send ACKs or NACKs to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted **CSMA/CD with binary backoff** (for coaxial-cable-based and hub-based Ethernet; not for switch-based Ethernet)

802.3 Ethernet Standards: Link & Physical Layers

- **many** different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
  - different physical layer media: fiber, cable

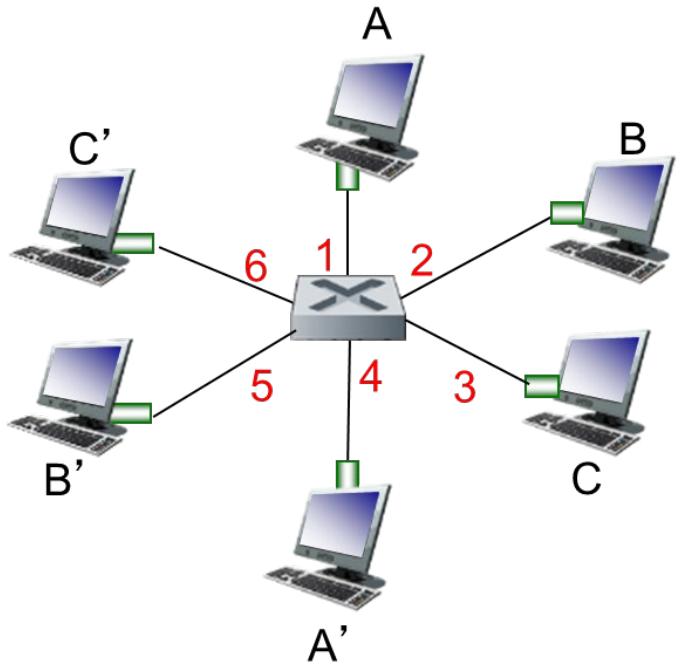


Ethernet Switch

- **link-layer device: takes an active role**
  - store, forward Ethernet frames
  - examine incoming frame's MAC address
  - **selectively** forward frame to one or more outgoing links when frame is to be forwarded on segment
- **transparent**
  - hosts are unaware of presence of switches
- **plug-and-play, self-learning**
  - switches do not need to be configured

Switch: Multiple Simultaneous Transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on **each** incoming link, but **no collisions**; full duplex
  - each link is its own collision domain
- **switching:** A-to-A' and B-to-B' can transmit simultaneously, without collisions



switch with six interfaces (1,2,3,4,5,6)

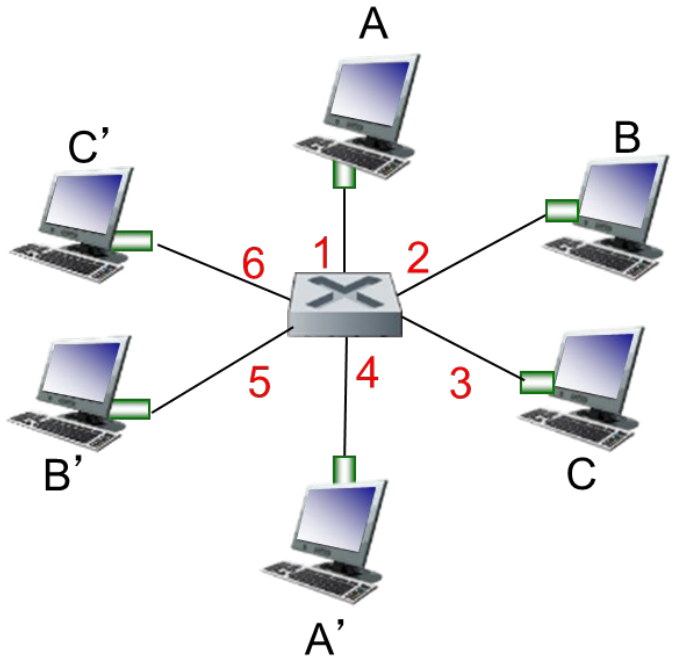
Switch Forwarding Table

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

- **A:** each switch has a **switch table,** each entry:
  - (MAC address of host, interface to reach host, time stamp)
  - looks like a routing table!

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

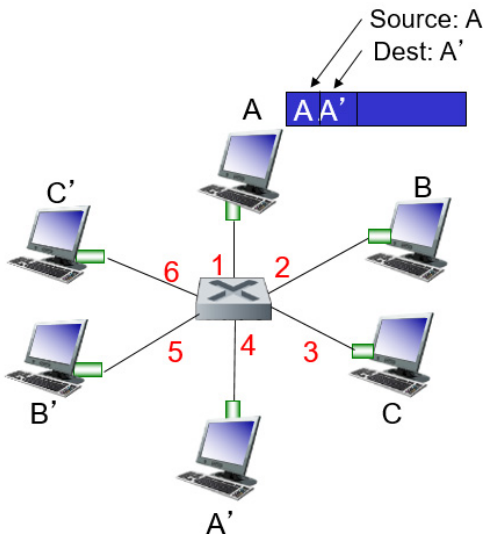
- something like a routing protocol?



switch with six interfaces (1,2,3,4,5,6)

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)

Switch: Frame Filtering/Forwarding

when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. **if** entry found for destination

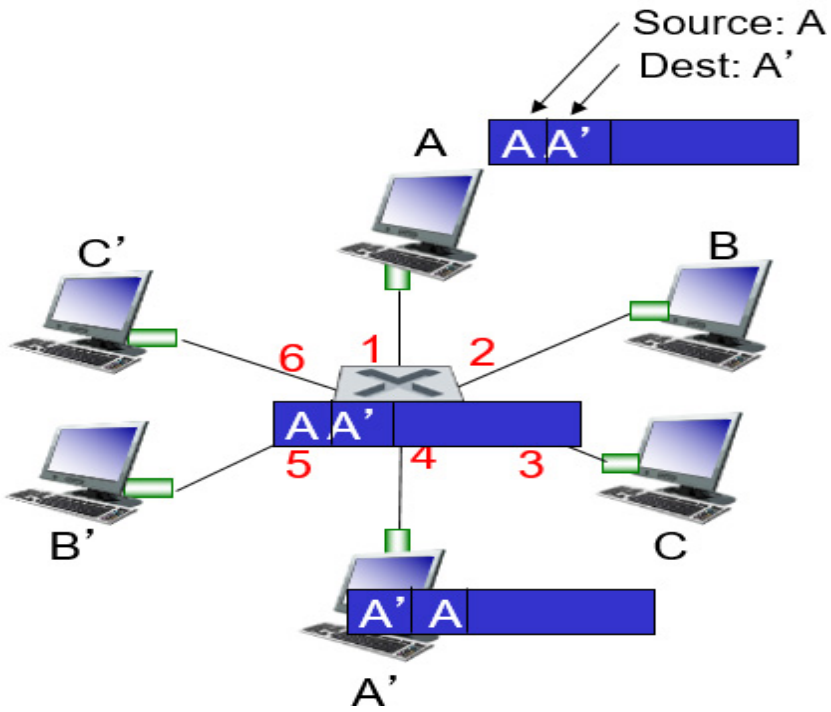
```
then {
if destination on segment from which frame arrived
    then drop frame
    else forward frame on interface indicated by entry
  }
  else flood /* forward on all interfaces except arriving interface */
```

Self-Learning, Forwarding: Example

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

MAC addr	interface	TTL
A	1	60
A'	4	60

switch table (initially empty)



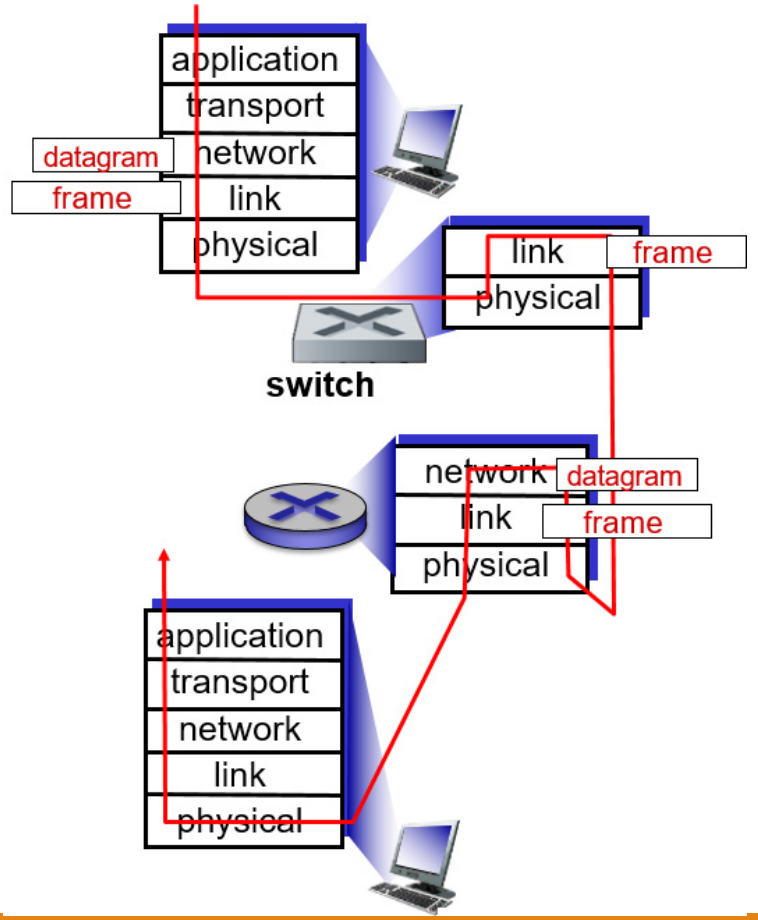
Switches vs. Routers

both are store-and-forward:

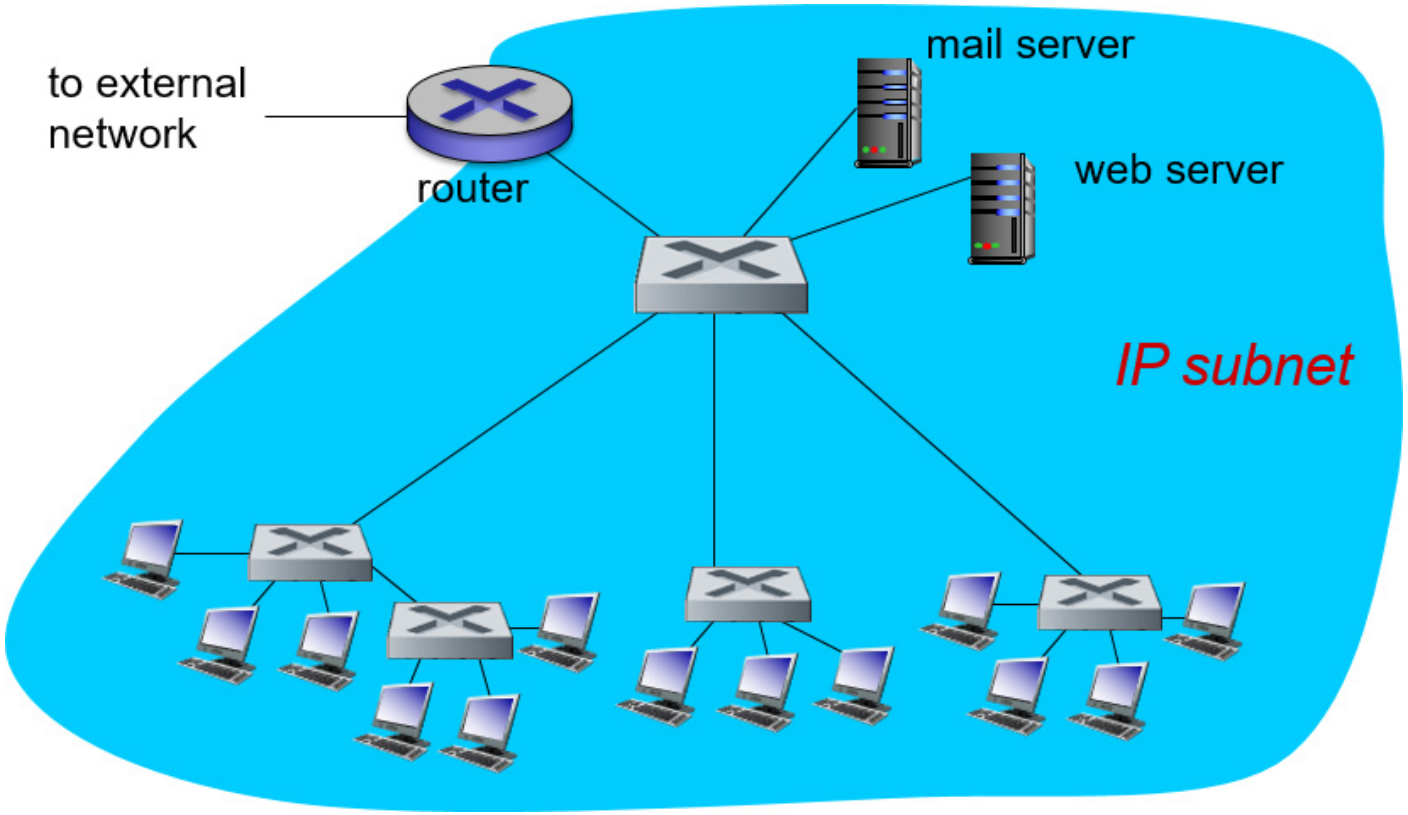
- **routers:** network-layer devices (examine network-layer headers)
- **switches:** link-layer devices (examine link-layer headers)

both have forwarding tables:

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



Institutional Network



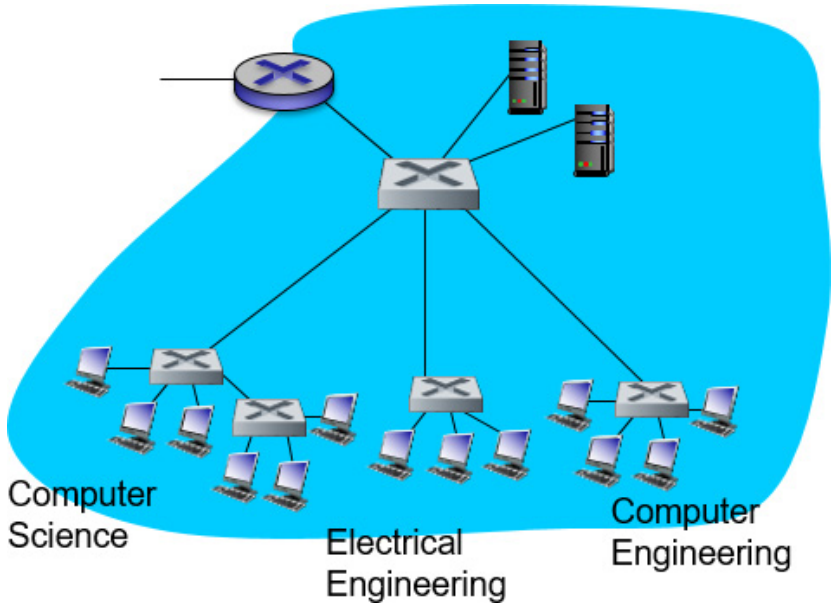
VLANs: Motivation

consider:

CS user moves office to EE, but wants connect to CS switch?

single broadcast domain - issues:

- all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
- security/privacy, efficiency issues

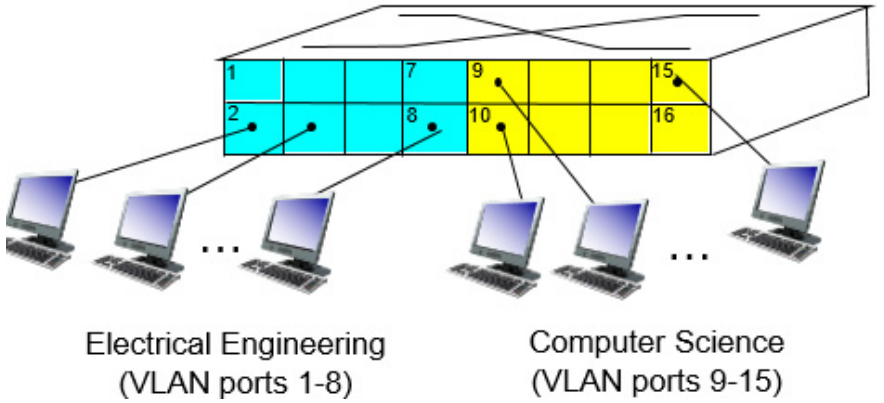


VLANs

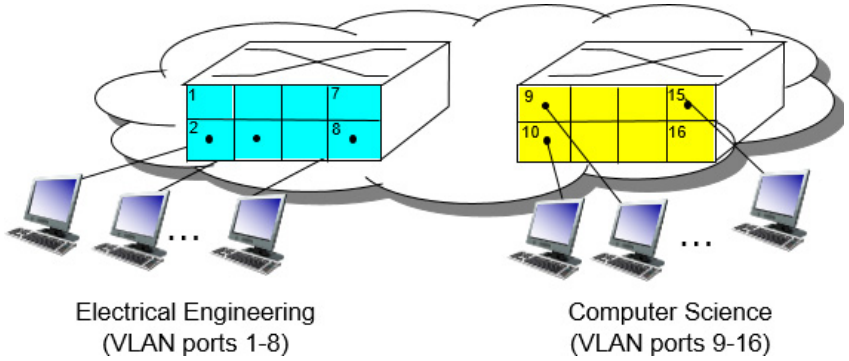
Virtual Local Area Network

switch(es) supporting VLAN capabilities can be configured to define multiple **virtual** LANS over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that **single** physical switch ……

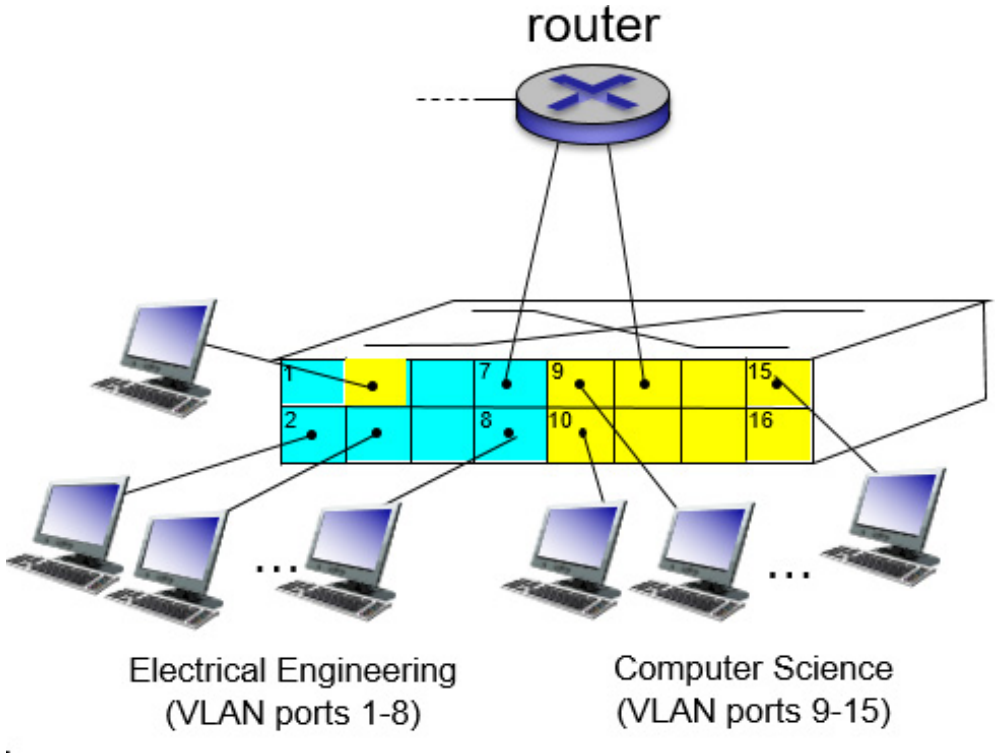


… operates as **multiple** virtual switches

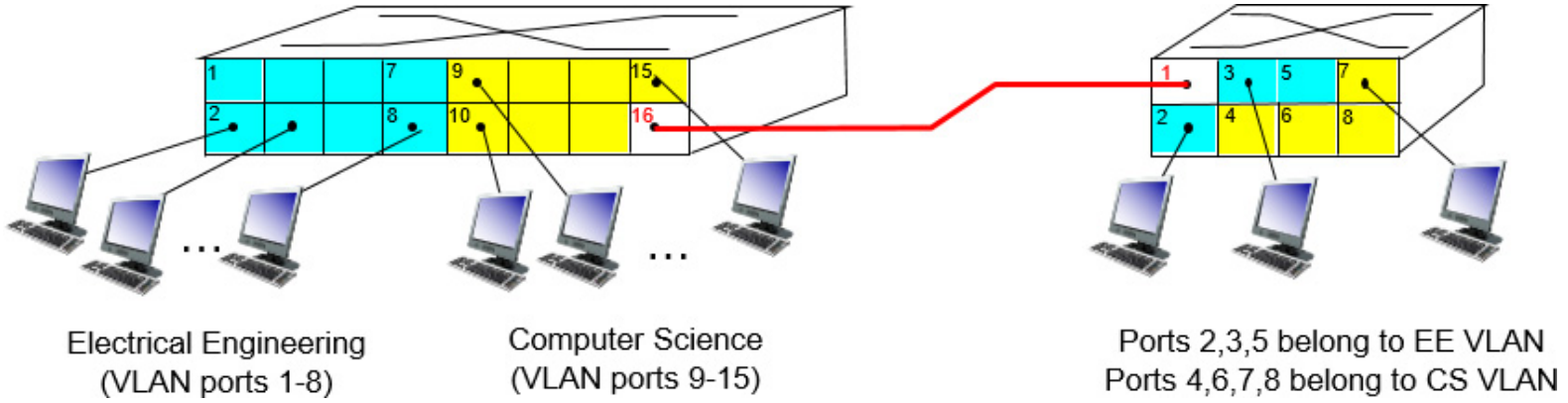


Port-Based VLAN

- **traffic isolation:** frames to/from ports 1-8 can **only** reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- **dynamic membership:** ports can be dynamically assigned among VLANs
- **forwarding between V LANS:** done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers



VLANS Spanning Multiple Switches



trunk port: carries frames between VLANs defined over multiple physical switches

- frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
- 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

802.1Q VLAN Frame Format