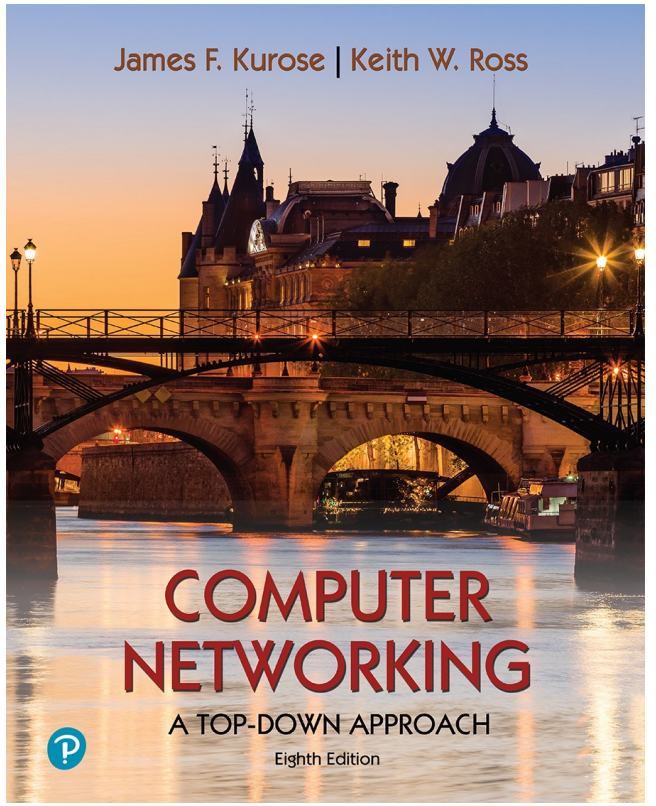
Mid-Term Overview

Yaxiong Xie

Department of Computer Science and Engineering
University at Buffalo, SUNY



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Adapted from the slides of the book's authors

Several Points

- I don’t have the mid-term exam question prepared yet
- Here, I will list all the ***topics*** that I think are important
  - If one topic I didn’t mention, then I won’t test it
  - It is about the topic, not the slides
    - If I didn’t mention one slides, but I do mention the topic, I probably will cover it
    - There are too many slides if I include every slides about that topic
- It will be fast, I won’t teach it again
- Ask questions, if you have any

Chapter 1: introduction

Chapter goal:

- Get "feel," "big picture," introduction to terminology
  - more depth, detail *later* in course

Overview/roadmap:

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

The Internet: a "nuts and bolts" view

Billions of connected computing *devices*:

- *hosts = end systems*
- running *network apps* at Internet's "edge"

Packet switches: forward packets (chunks of data)

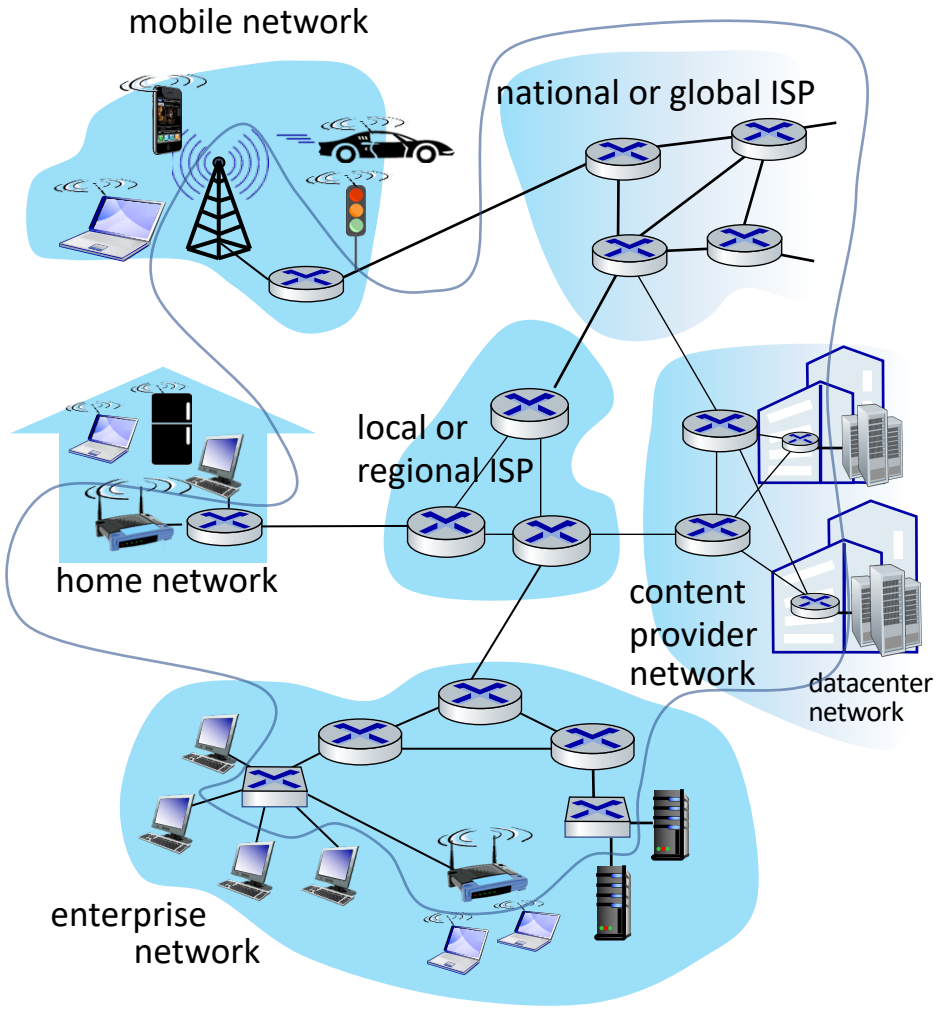
- *routers, switches*

Communication links

- fiber, copper, radio, satellite
- transmission rate: *bandwidth*

Networks

- collection of devices, routers, links: managed by an organization



What's a protocol?

Human protocols:

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

Rules for:

… specific messages sent

… specific actions taken when message received, or other events

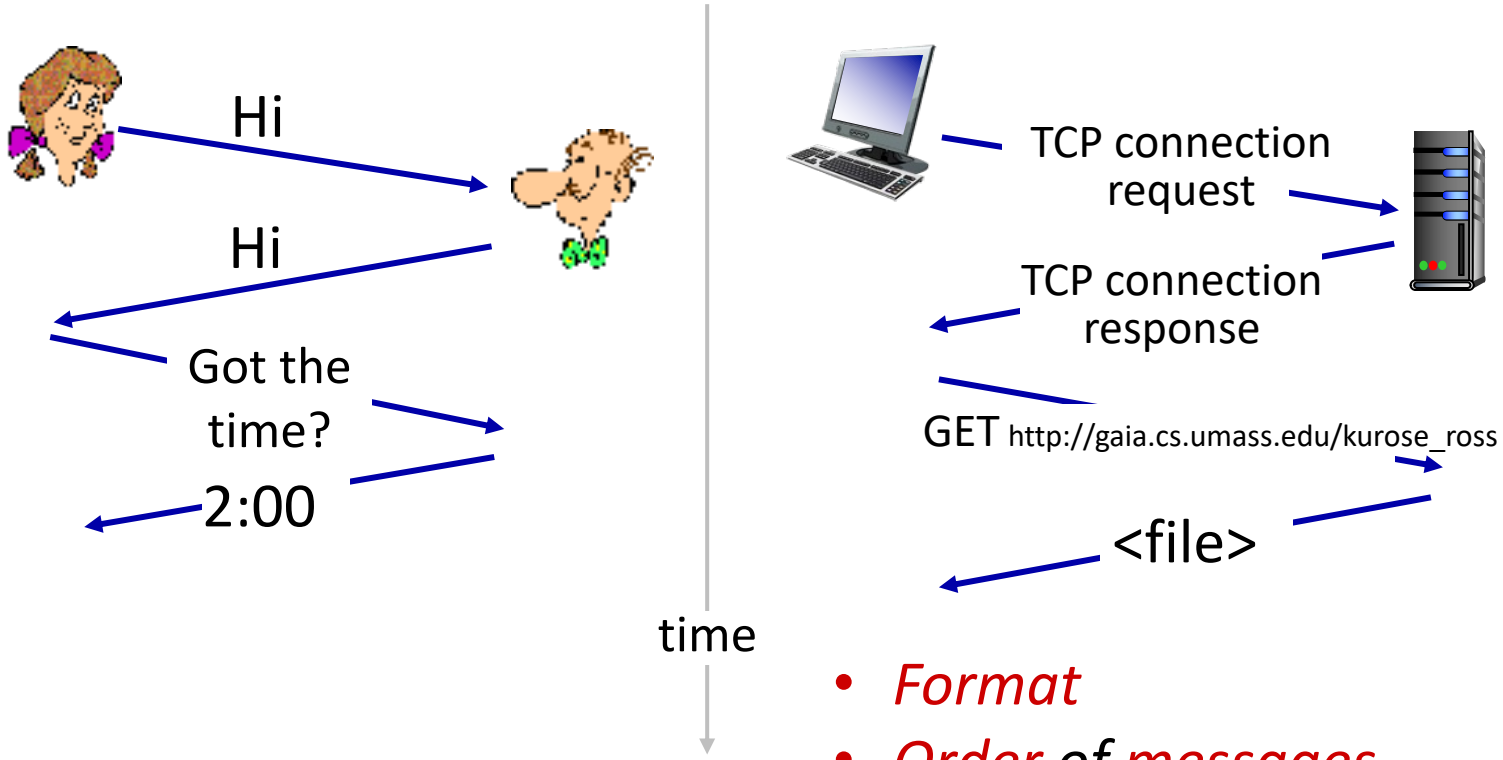
Network protocols:

- computers (devices) rather than humans
- all communication activity in Internet governed by protocols

Protocols define the format, order of messages sent and received among network entities, and actions taken on message transmission, receipt

What's a protocol?

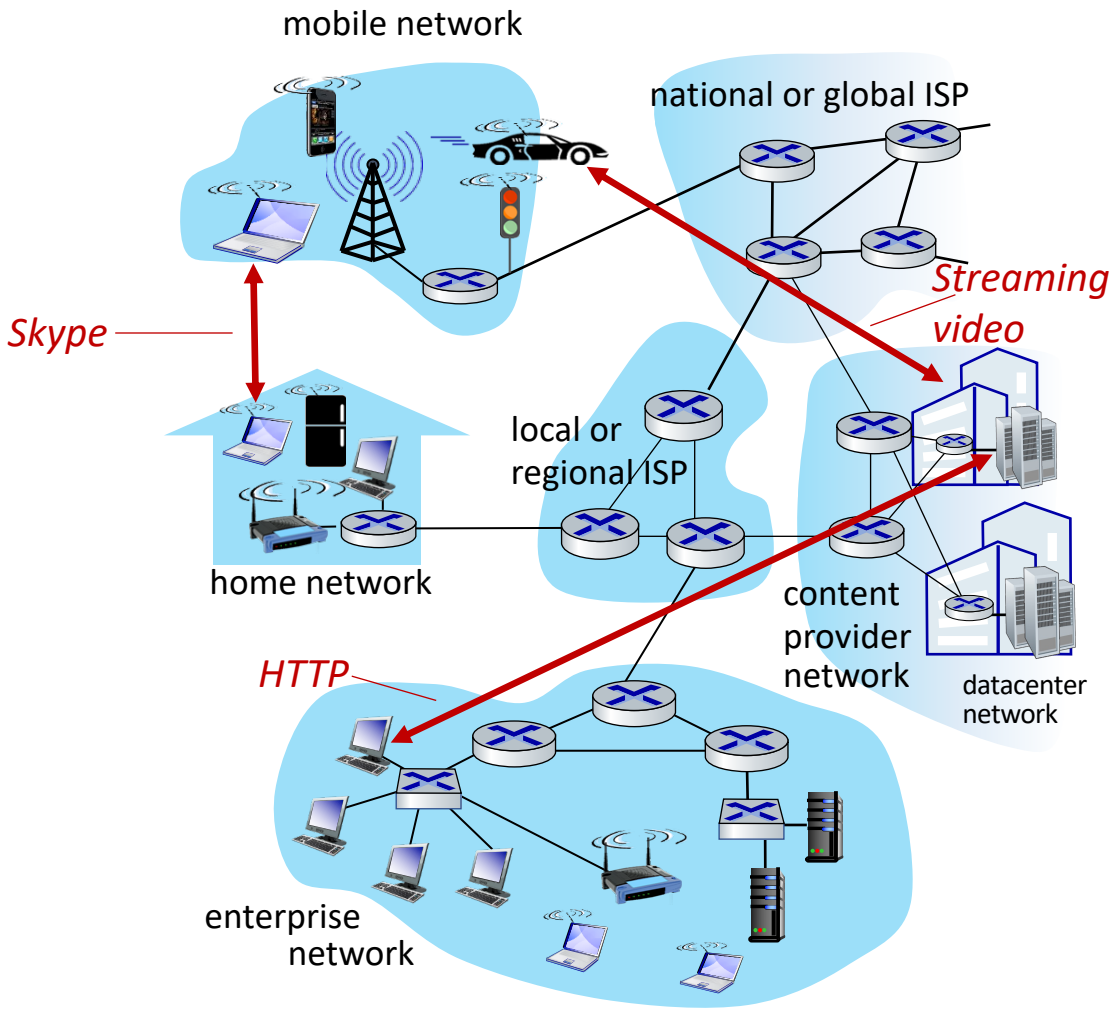
A human protocol and a computer network protocol:



- *Format*
- *Order of messages*
- *Actions taken on message Tx and RX*

The Internet: a "services" view

- *As an Infrastructure* that provides services to applications:
  - Web, streaming video, multimedia teleconferencing, social media,…
  - provided by hardware and software (*protocols*)
- provides *programming interface* to distributed applications:
  - "hooks" allowing sending/receiving apps to "connect" to, use Internet transport service
  - provides service options, analogous to postal service



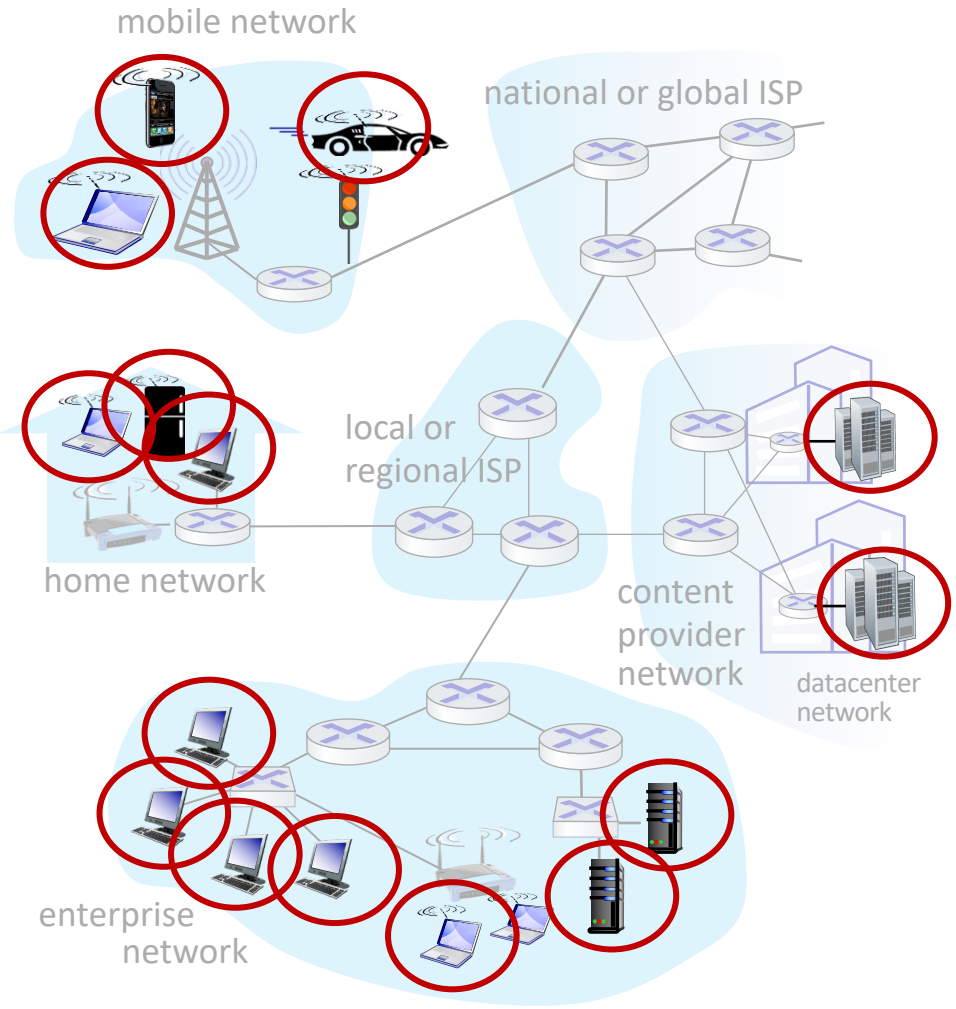
Chapter 1: roadmap

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

A closer look at Internet structure

Network edge:

- hosts: clients and servers
- servers often in data centers



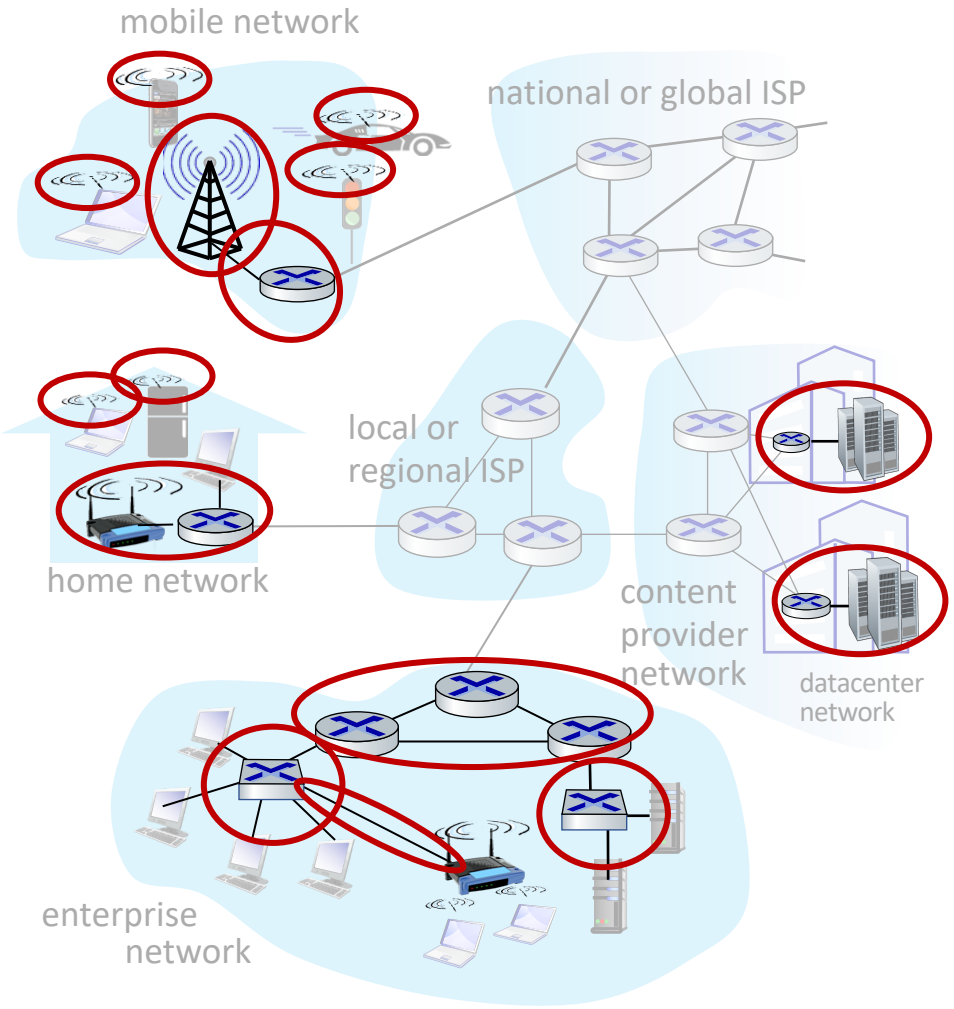
A closer look at Internet structure

Network edge:

- hosts: clients and servers
- servers often in data centers

Access networks, physical media:

- wired, wireless communication links



A closer look at Internet structure

Network edge:

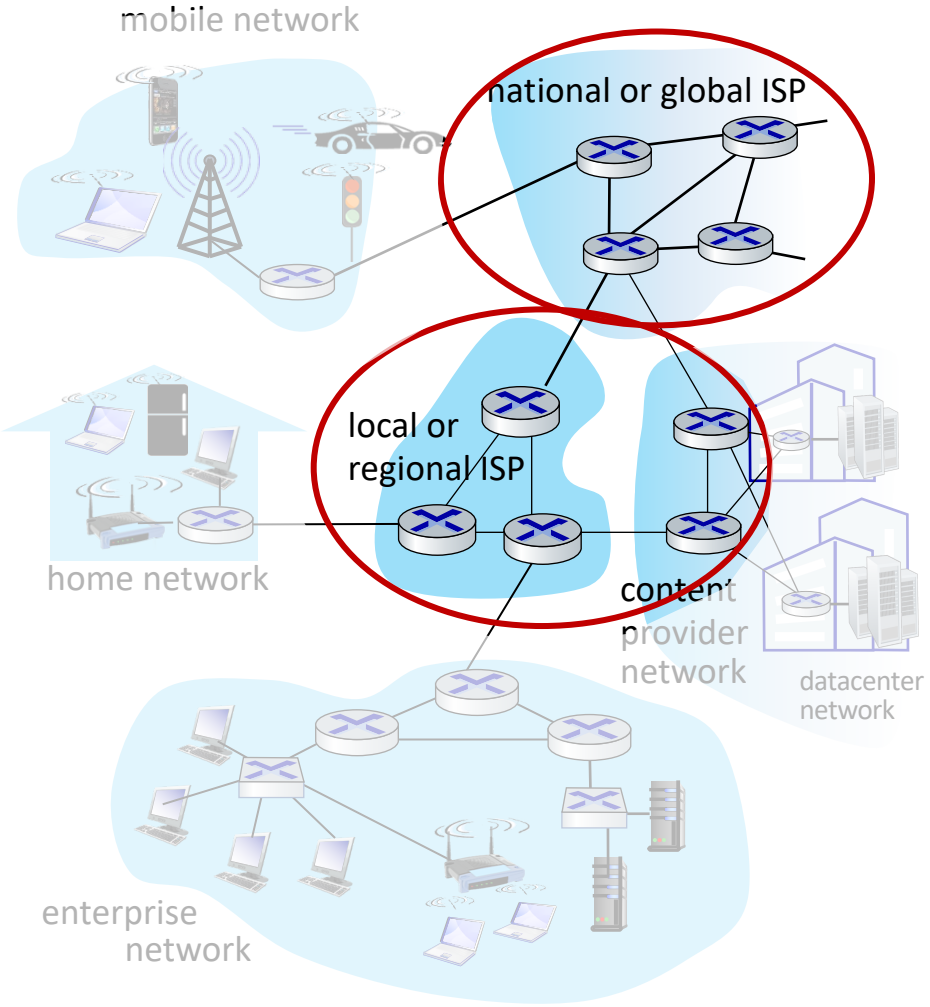
- hosts: clients and servers
- servers often in data centers

Access networks, physical media:

- wired, wireless communication links

Network core:

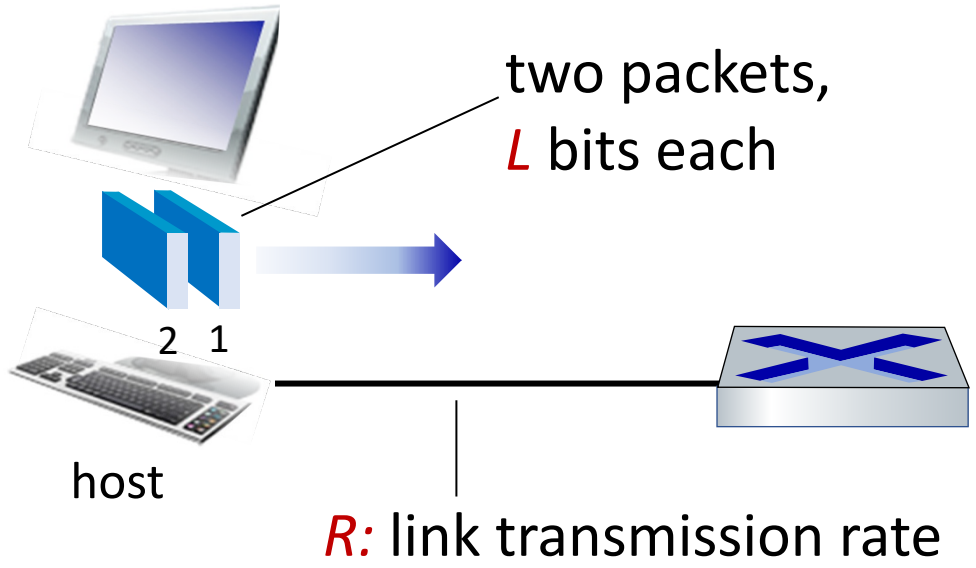
- interconnected routers
- network of networks



Host: sends *packets* of data

host sending function:

- takes application message
- breaks into smaller chunks, known as *packets*, of length *L* bits
- transmits packet into access network at *transmission rate R*
  - link transmission rate, aka link *capacity, aka link bandwidth*



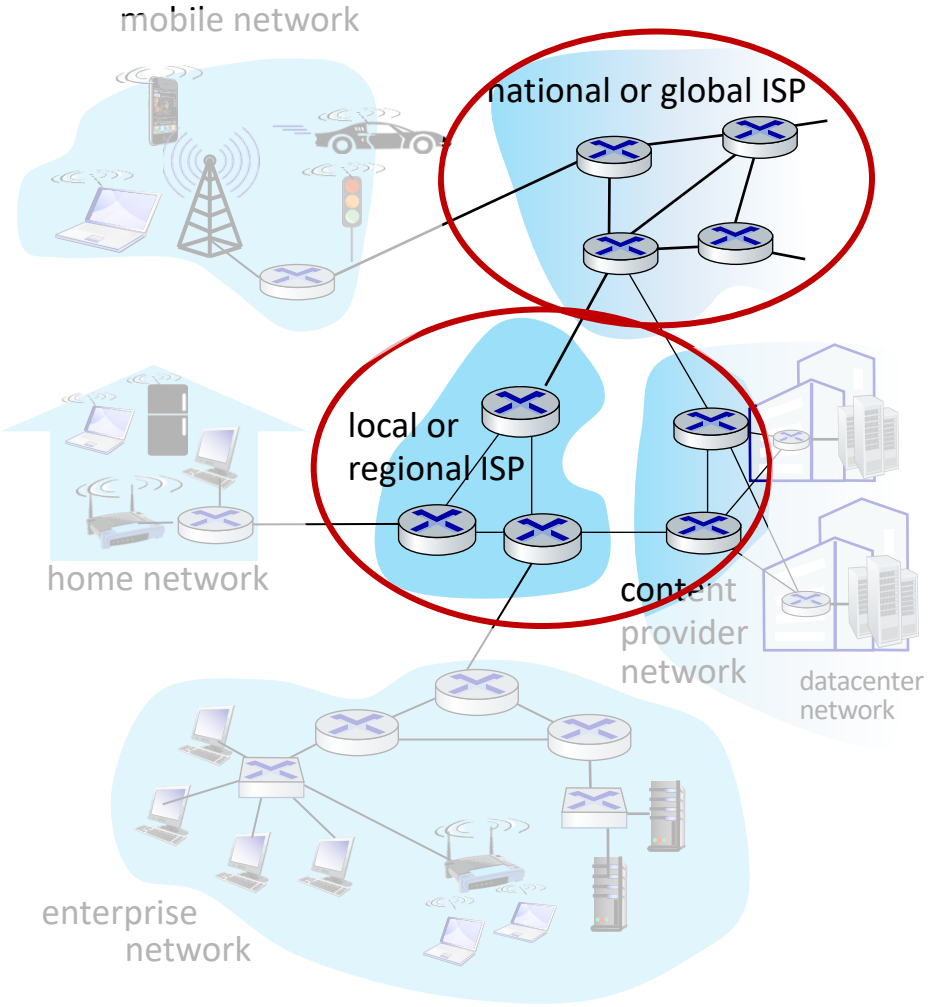
$$\text{packet transmission delay} = \text{time needed to transmit } L\text{-bit packet into link} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

Chapter 1: roadmap

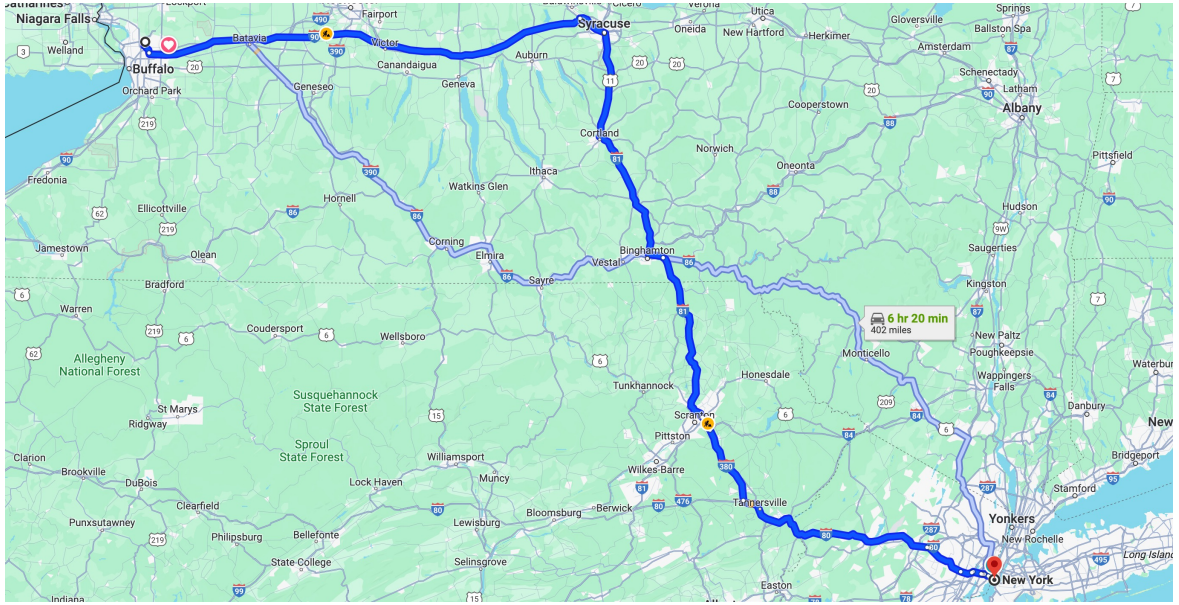
- What *is* the Internet?
- What *is* a protocol?
- Network edge: hosts, access network, physical media
- **Network core:** packet/circuit switching, internet structure
- Performance: loss, delay, throughput
- Security
- Protocol layers, service models
- History

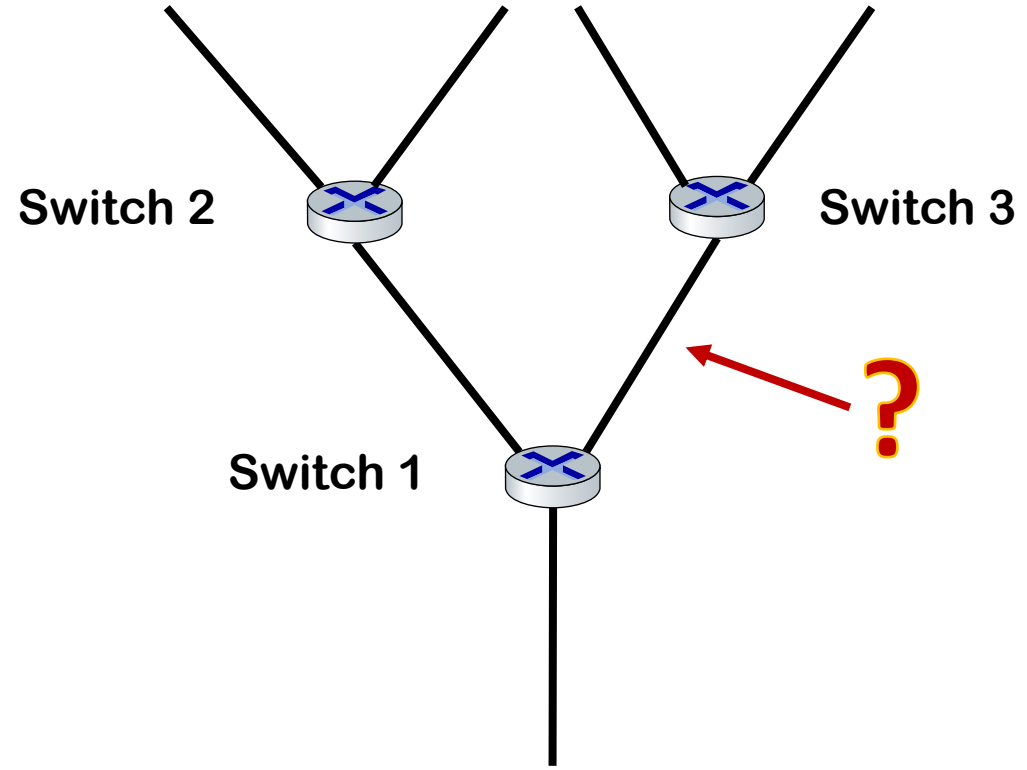
The network core

- mesh of interconnected routers
- packet-switching: hosts break application-layer messages into *packets*
  - network forwards packets from one router to the next, across links on path from source to destination

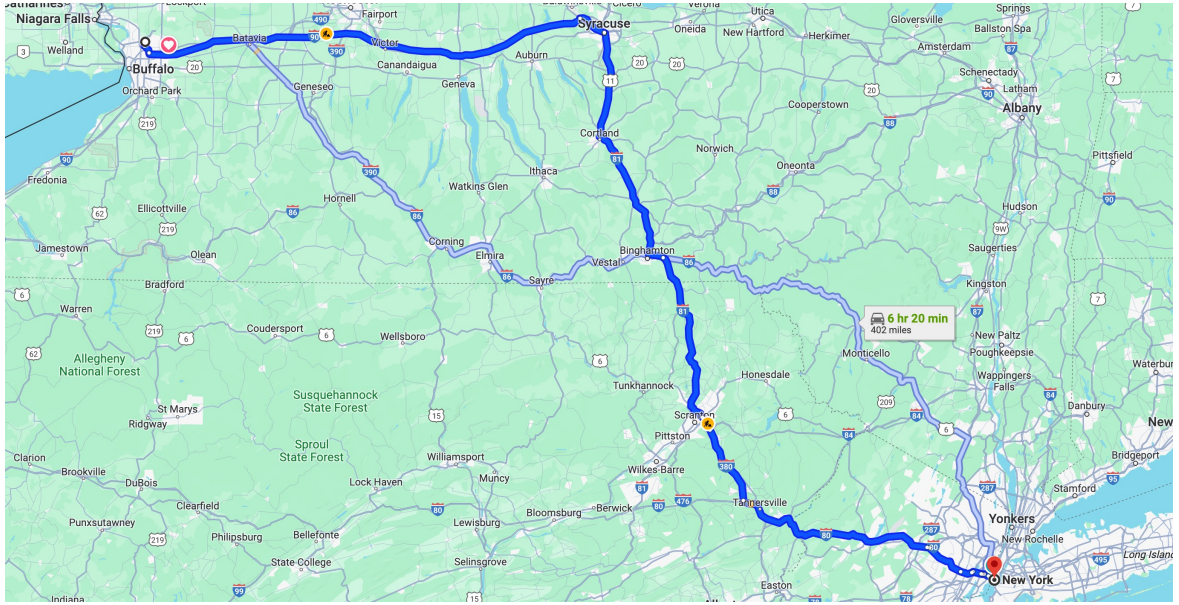


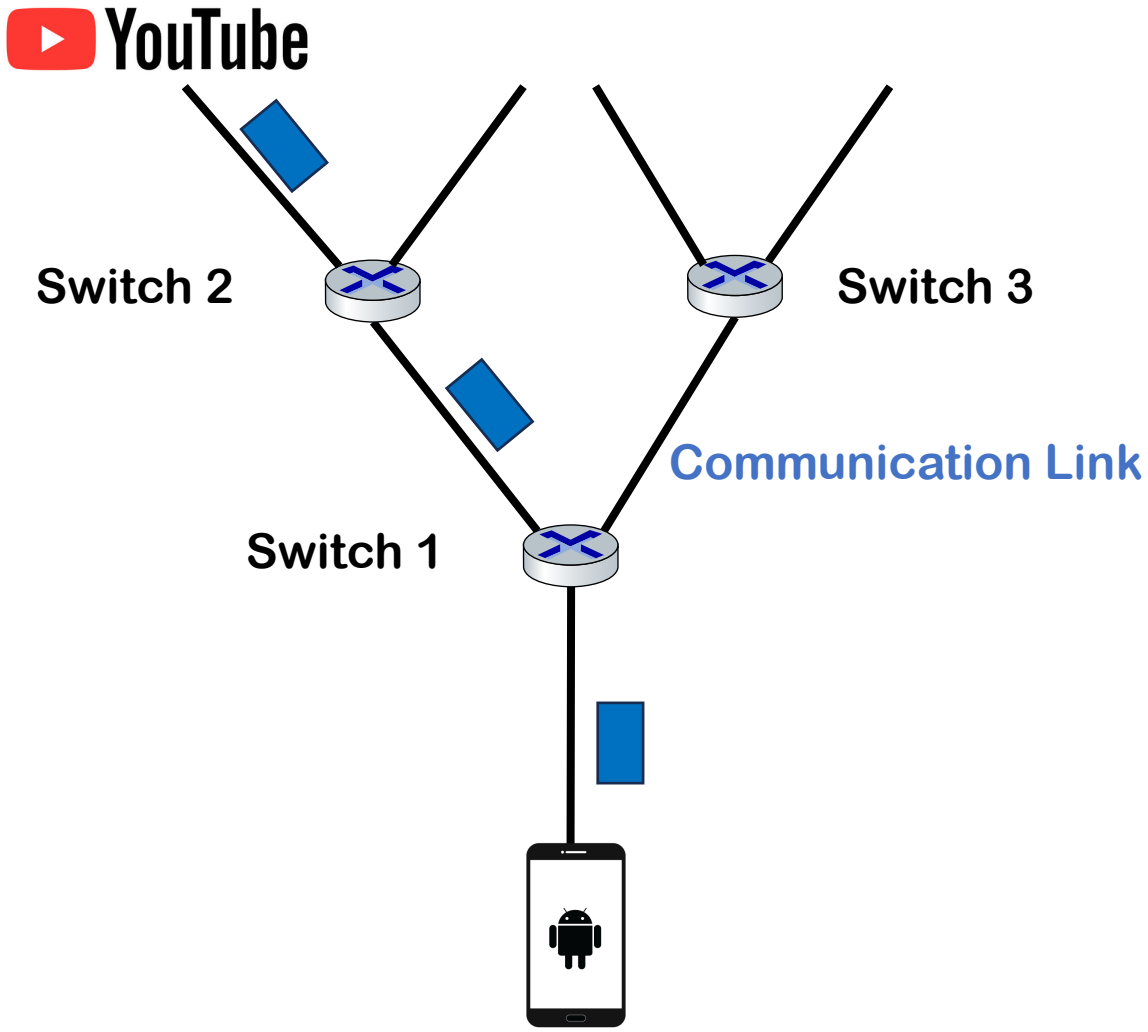
Network Switch



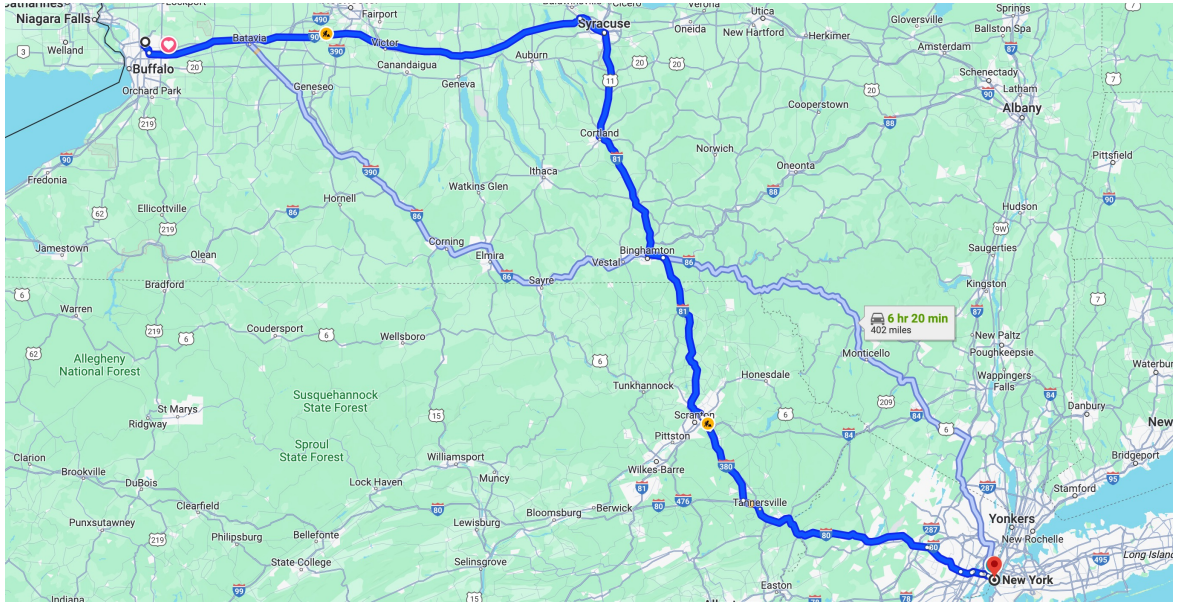


Network Switch

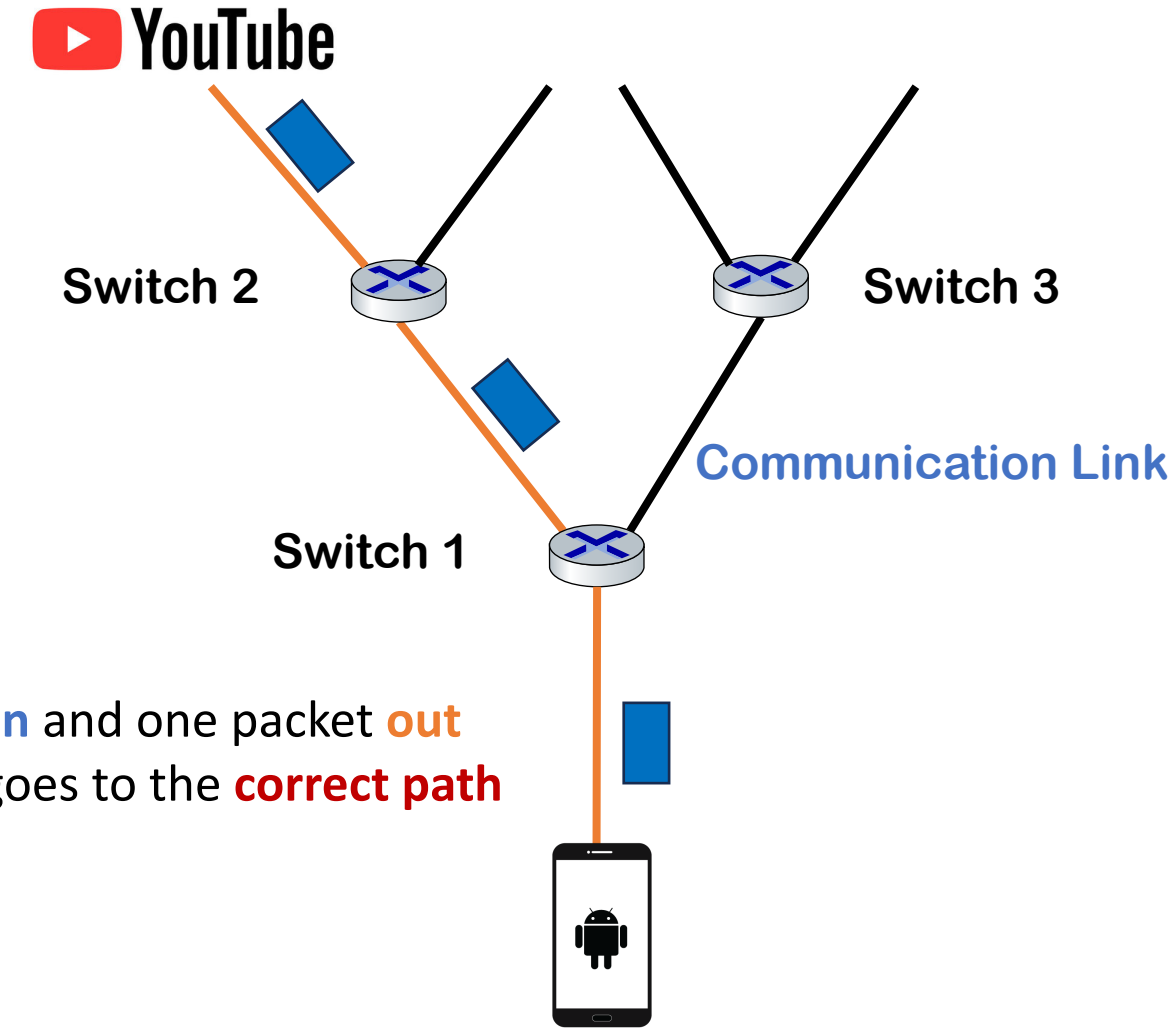




Network Switch



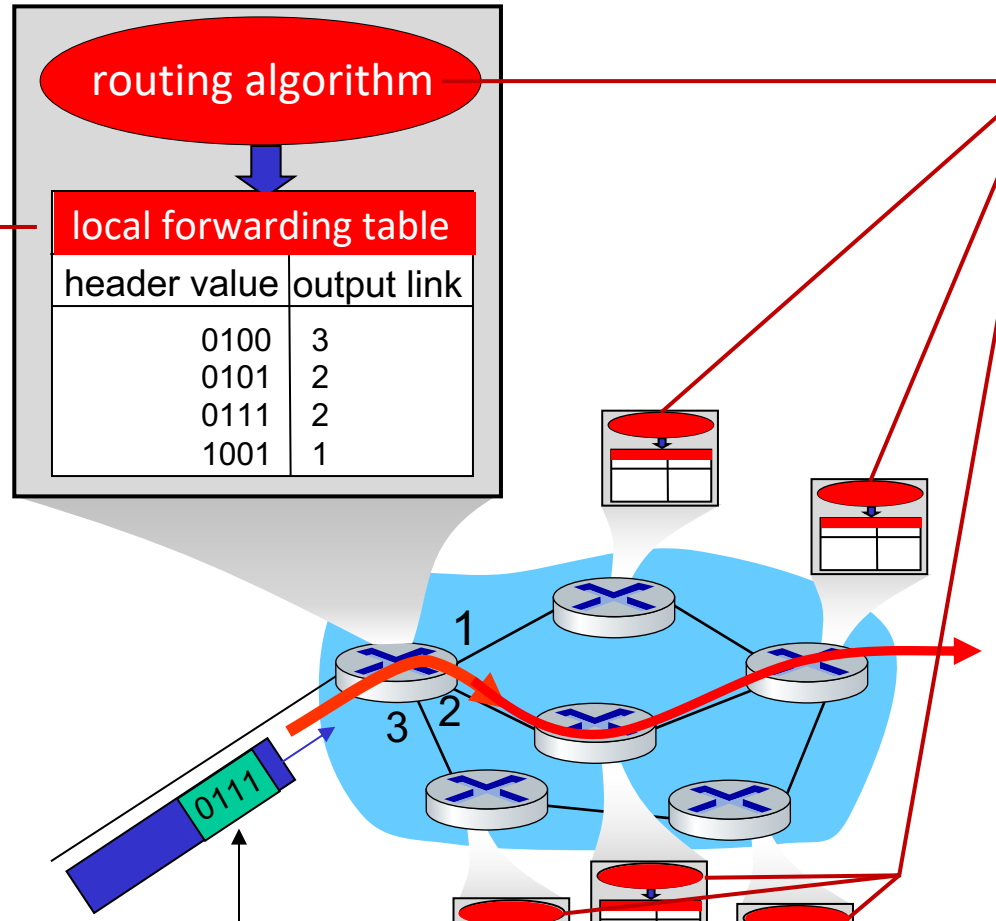
- One packet **in** and one packet **out**
- The packet goes to the **correct path**



Two key network-core functions

Forwarding:

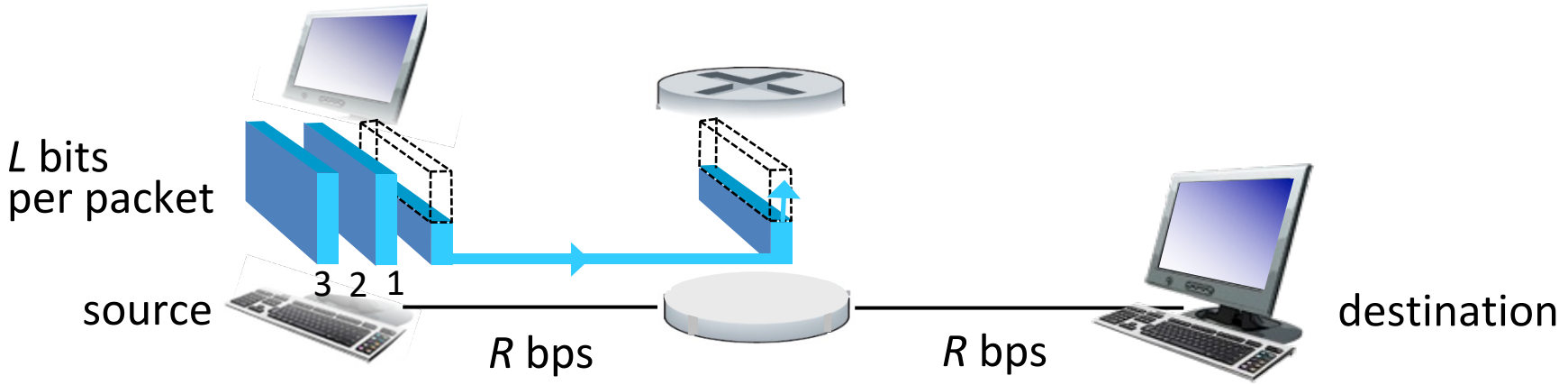
- aka "switching"
- *local* action: move arriving packets from router's input link to appropriate router output link



Routing:

- *global* action: determine source-destination paths taken by packets
- routing algorithms

Packet-switching: store-and-forward

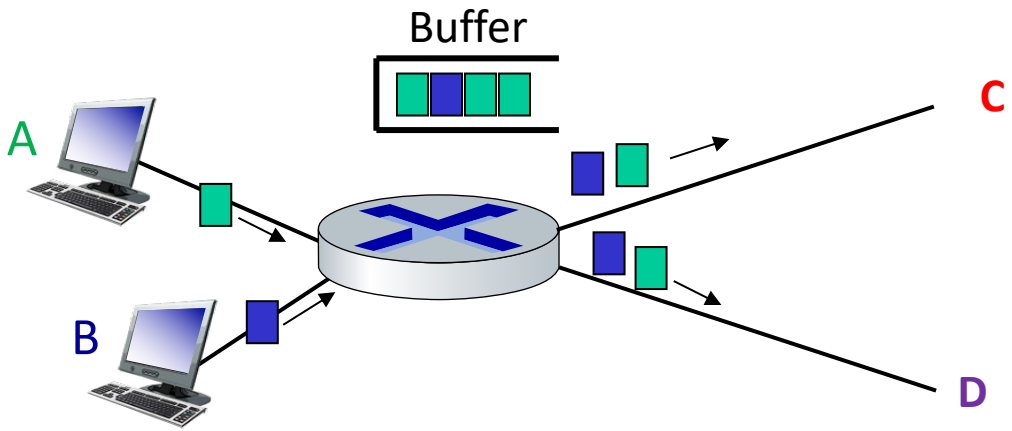


- packet transmission delay: takes L/R seconds to transmit (push out) L-bit packet into link at R bps
- *store and forward: entire* packet must arrive at router before it can be transmitted on next link

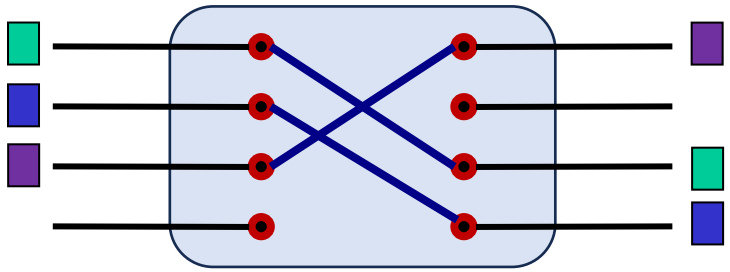
One-hop numerical example:

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

Alternative to packet switching: circuit switching (e.g., plain old telephone networks or POTS)



Packet Switching



Circuit Switching

Packet switching versus circuit switching

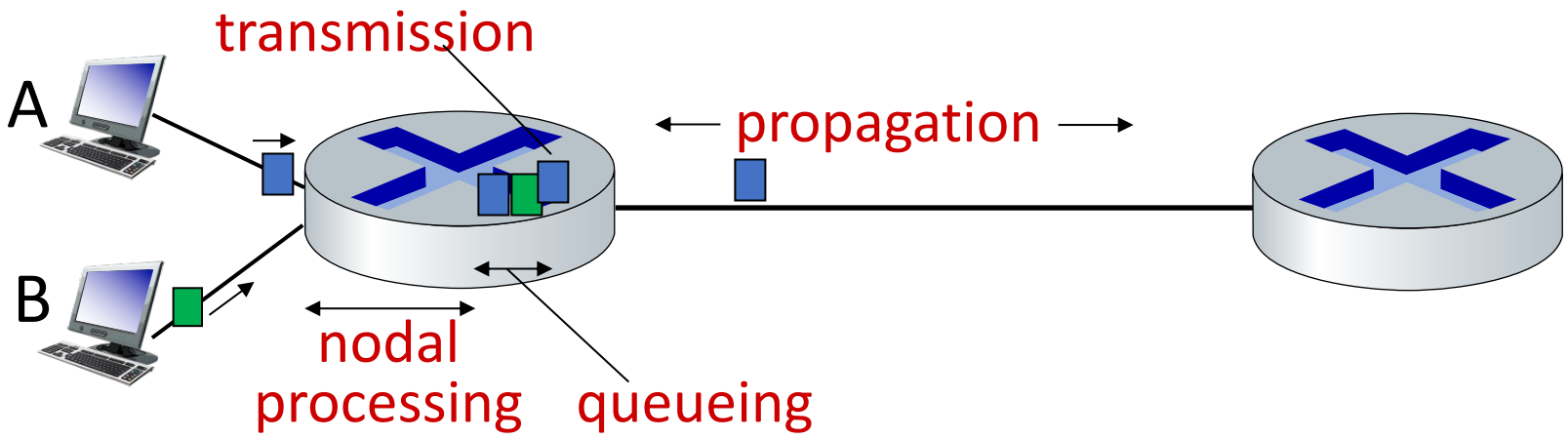
Is packet switching a “slam dunk winner”?

- great for “bursty” data – sometimes has data to send, but at other times not
  - resource sharing
  - simpler, no call setup
- excessive congestion possible: packet delay and loss due to buffer overflow
  - protocols needed for reliable data transfer, congestion control
- *Q:* How to provide circuit-like behavior with packet-switching?
  - “It’s complicated.” We’ll study various techniques that try to make packet switching as “circuit-like” as possible.

Chapter 1: roadmap

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

Packet delay: four sources



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

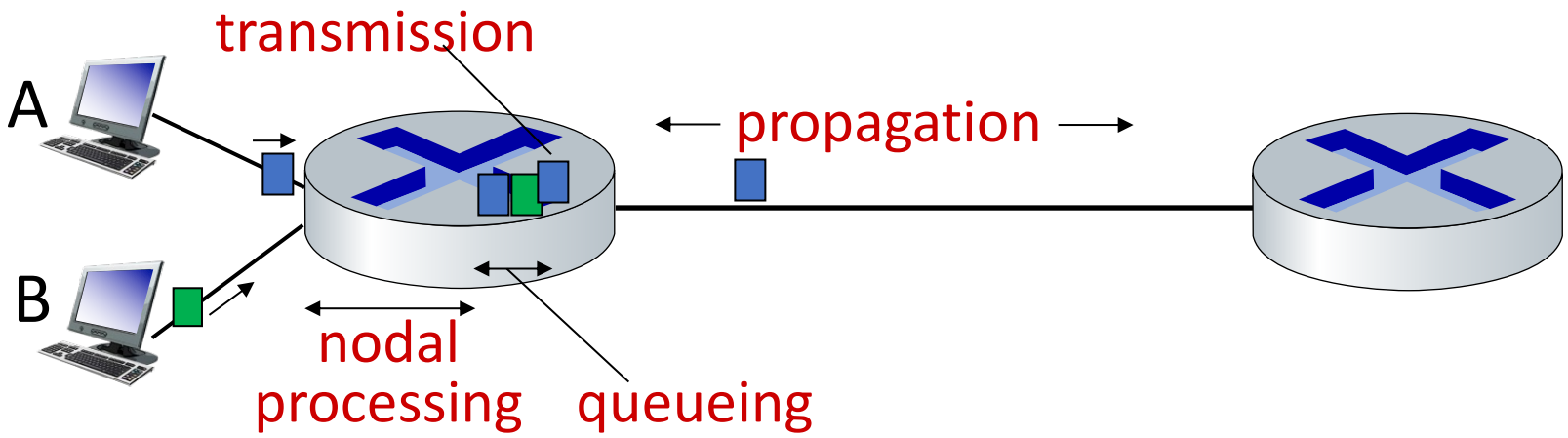
d_{proc}: nodal processing

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

d_{queue}: queueing delay

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

Packet delay: four sources



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

d_{trans}: transmission delay:

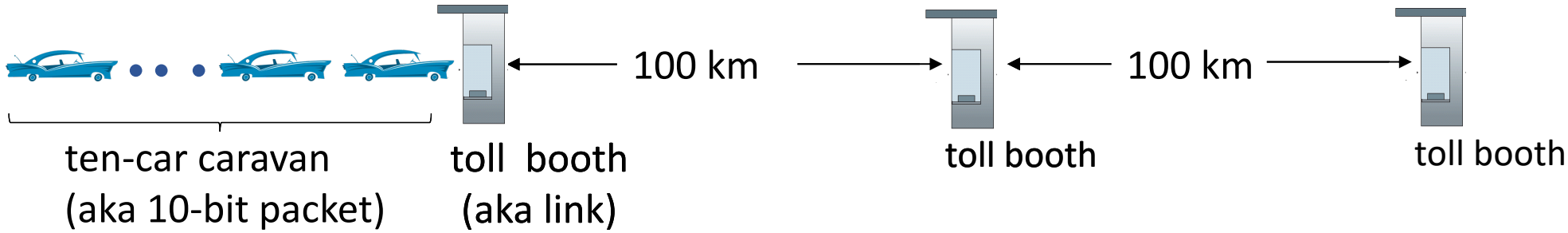
- L: packet length (bits)
- R: link *transmission rate (bps)*
- $d_{trans} = L/R$

d_{prop}: propagation delay:

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

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

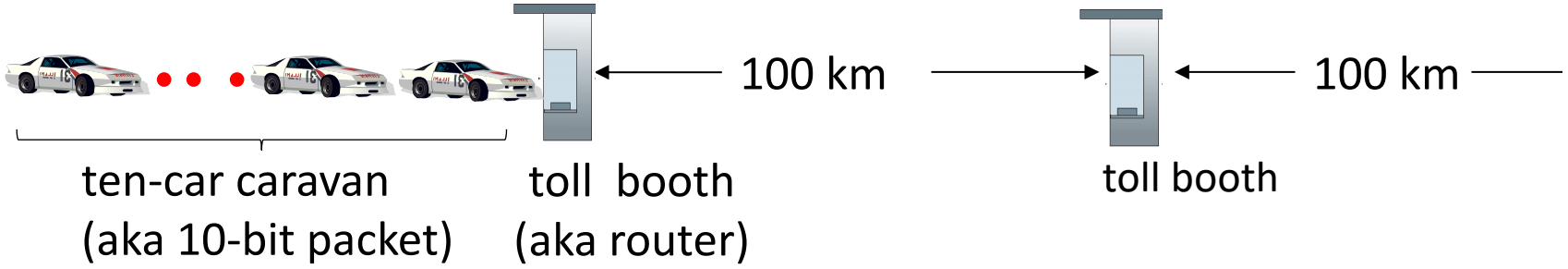
Caravan analogy



- car ~ bit; caravan ~ packet; toll service ~ link transmission
- toll booth takes 12 sec to service car (bit transmission time)
- "propagate" at 100 km/hr
- *Q:* How long until caravan is lined up before 2nd toll booth?

- time to "push" entire caravan through toll booth onto highway = 12*10 = 120 sec
- time for last car to propagate from 1st to 2nd toll both: 100km/(100km/hr) = 1 hr
- *A:* 62 minutes

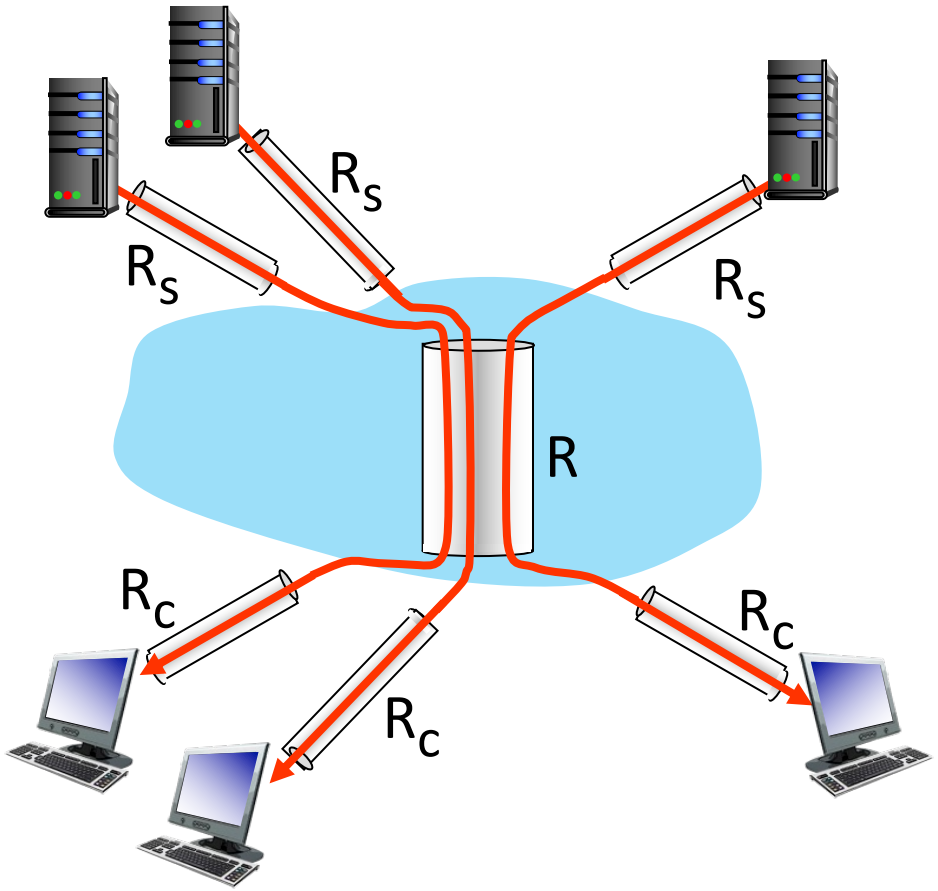
Caravan analogy



- suppose cars now "propagate" at 1000 km/hr
- and suppose toll booth now takes one min to service a car
- *Q:* Will some cars arrive at 2nd booth before all cars serviced at first booth?

  A: Yes! after 7 min, first car arrives at second booth; three cars still at first booth

Throughput: network scenario



10 connections (fairly) share backbone bottleneck link *R* bits/sec

- per-connection end-end throughput: *min(R_c,R_s,R/10)*
- in practice: R_c or R_s is often bottleneck

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

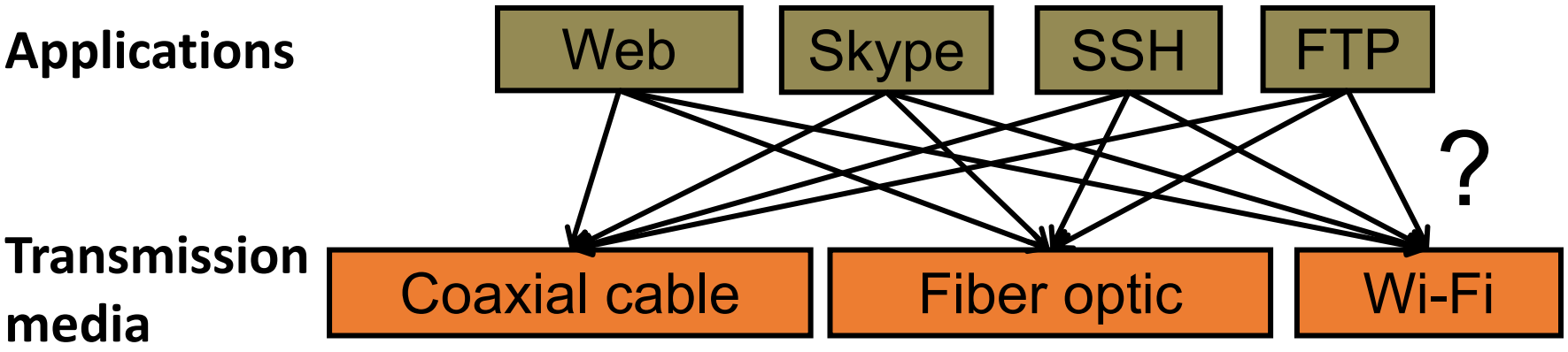
Chapter 1: roadmap

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

Chapter 1: roadmap

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

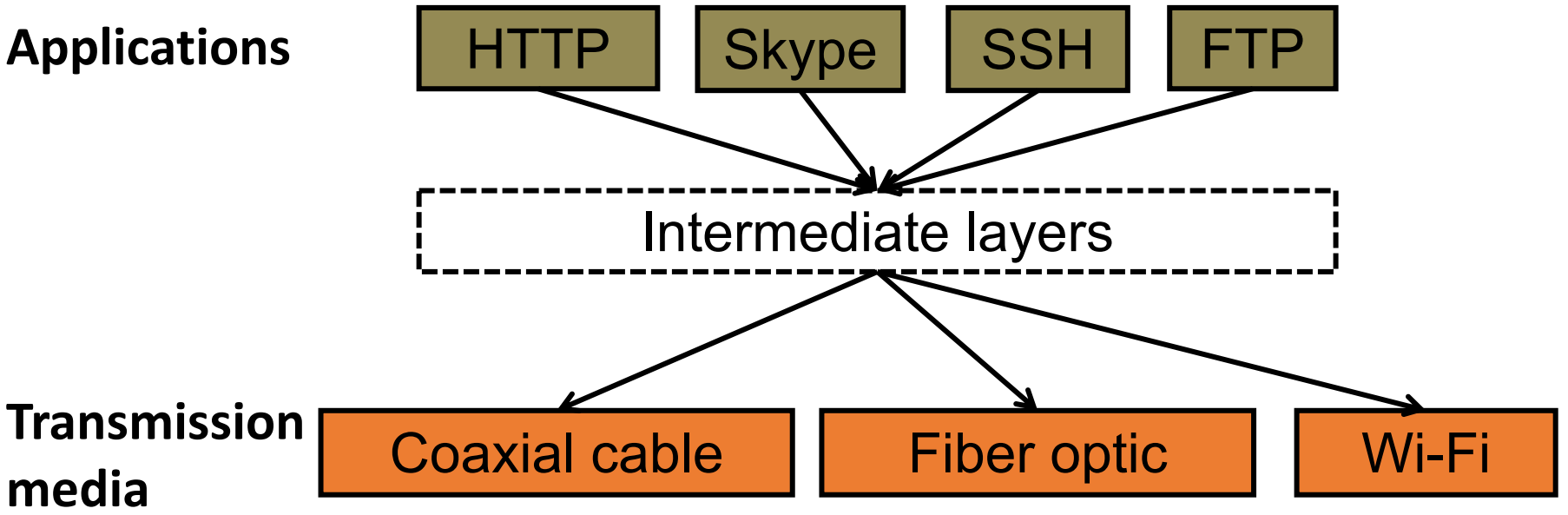
Layering: Motivation



- **Re-implement every application** for every new underlying transmission medium?
  - **Change** every application on any **change** to an underlying transmission medium (and vice-versa)?

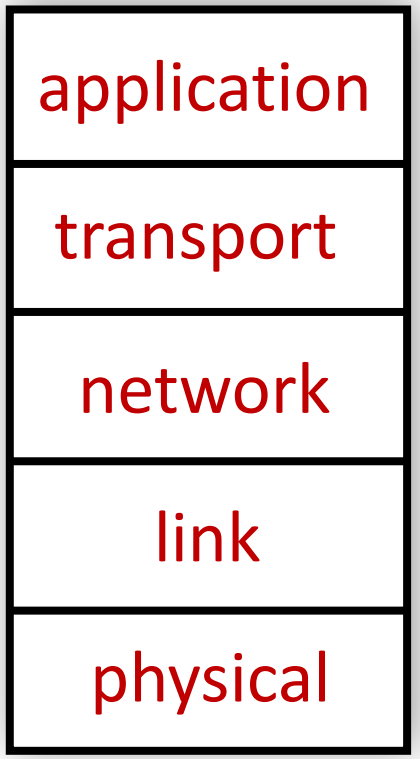
- **No!** But how does the Internet design avoid this?

Internet solution: Intermediate layers



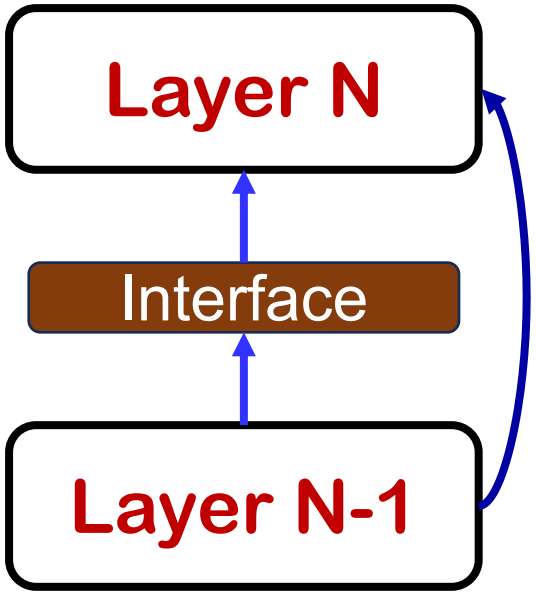
- **Intermediate layers** provide a **set of abstractions** for applications and media
- New applications or media **need only implement for** intermediate layer's **interface**

Properties of layers



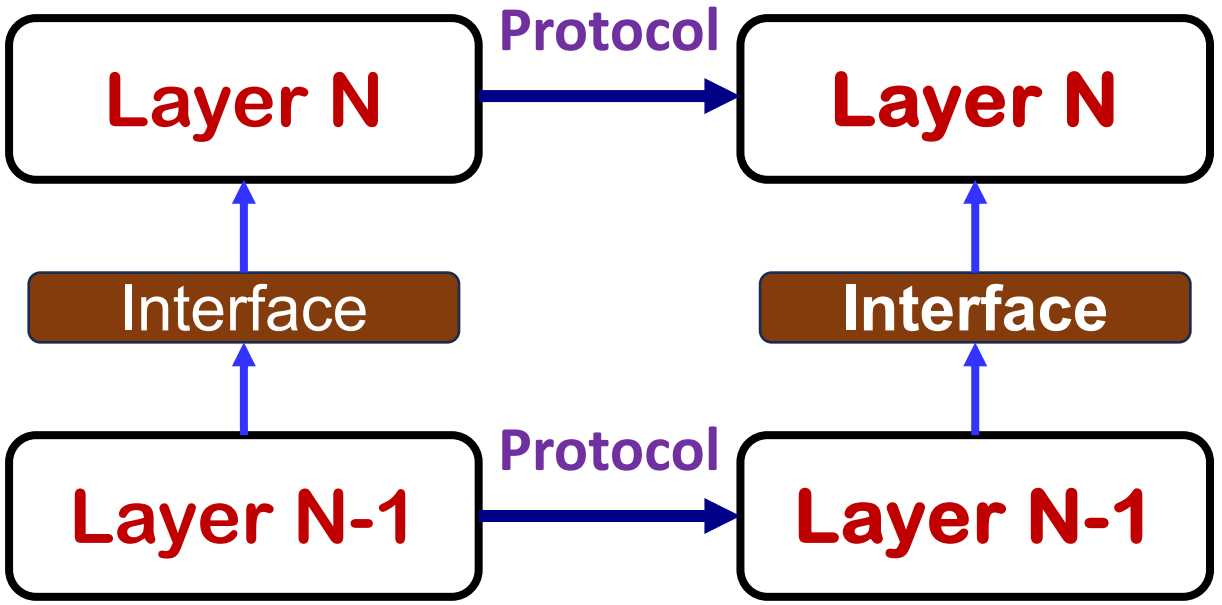
Properties of layers

- **Service:** **What** a layer does
- **Service interface:** **How to access** the service
  - Interface for the layer **above**



Layer N uses the services provided by **layer N-1**

Properties of layers



- **Service:** **What** a layer does
- **Service interface:** **How to access** the service
  - Interface for the layer **above**

- **Protocol interface:** **How peers communicate** to implement service
  - Set of rules and formats that govern the communication **between two Internet hosts**

Why stack or layering?

Approaches to dealing with complex systems

- Explicit structure allows identification, relationship of system's pieces
  - Layered reference model for discussion

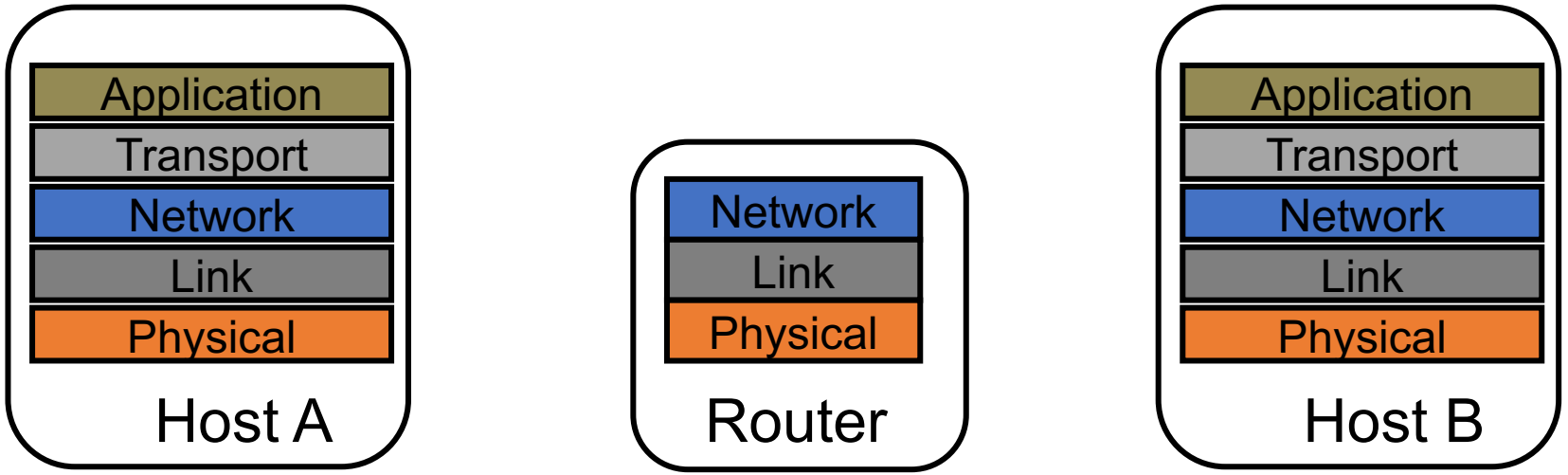
- Modularization ease maintenance, updating the system
  - Change in layer's service implementation: transparent to rest of system

Drawbacks of layering

- Layer *n* may **duplicate** lower level functionality
  - *e.g.*, error recovery to retransmit lost data
- Layers may need **same information in headers**
  - *e.g.*, timestamps, maximum transmission unit size
- Layering can **hurt performance**
  - *e.g.*, headers

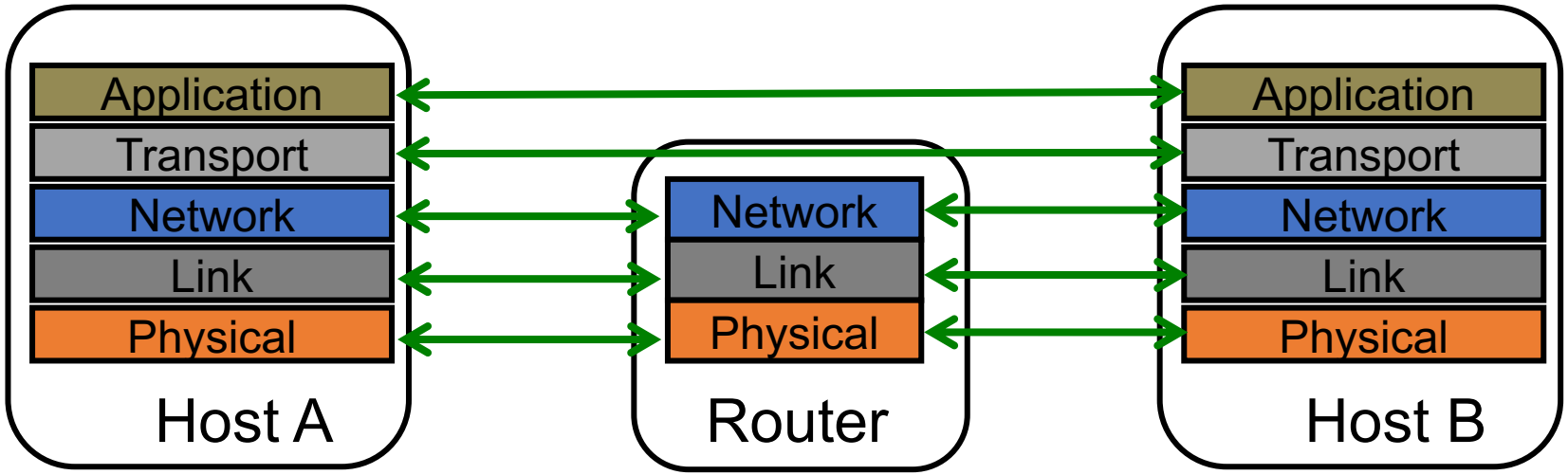
Who does what?

- Five layers
  - **Lower three layers** are implemented **everywhere**
  - **Top two layers** are implemented **only at end hosts**
    - **Their protocols are *end-to-end***



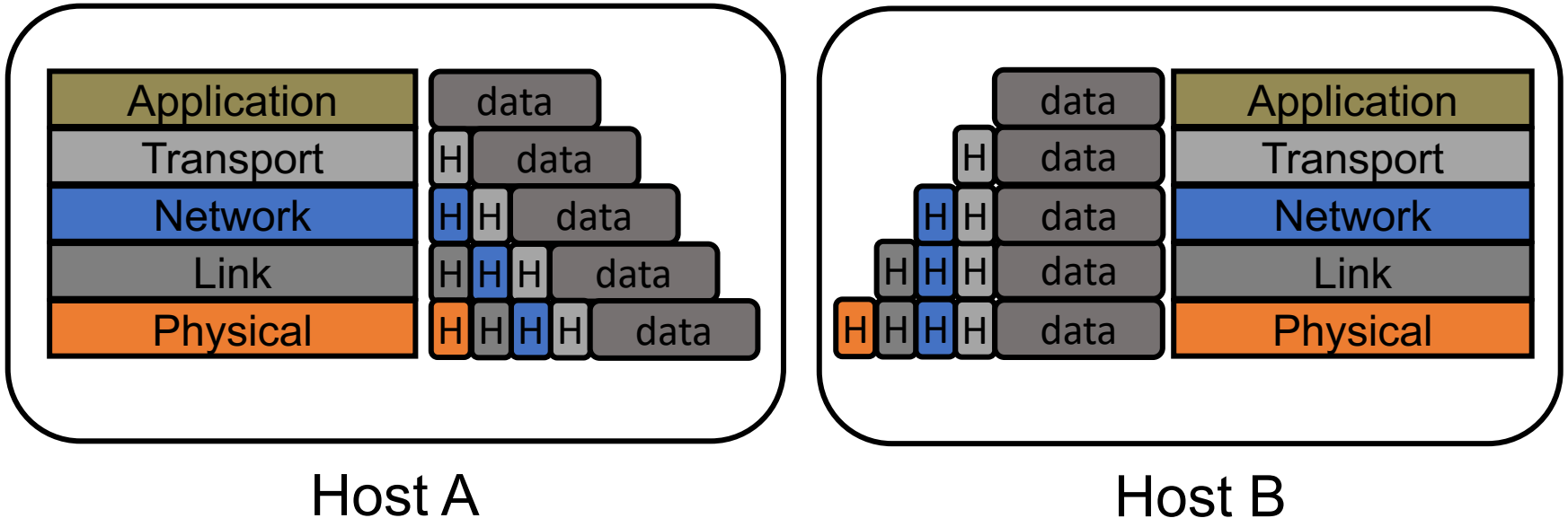
Logical communication

- Each layer on a host interacts with its **peer** host's corresponding layer via the **protocol interface**

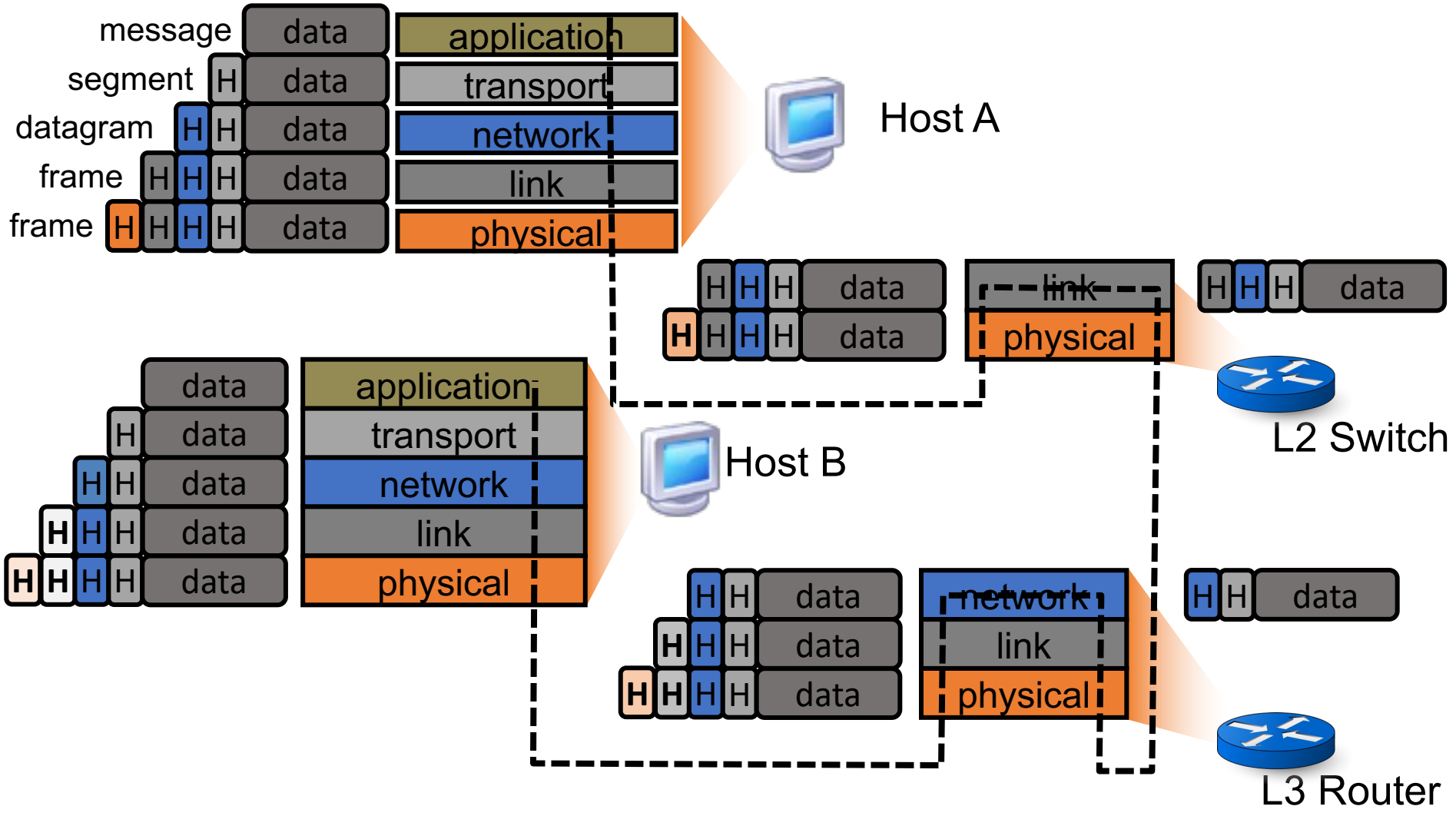


Protocol headers

- Each layer attaches its own header (H) to facilitate communication between peer protocols
- On reception, layer **inspects and removes** its own header
  - Higher layers **don't see** lower layers' headers



Encapsulation in the Internet



Application layer: overview

- Principles of network applications
- socket programming with UDP and TCP
  - Transport layer interface
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks

Client-server paradigm

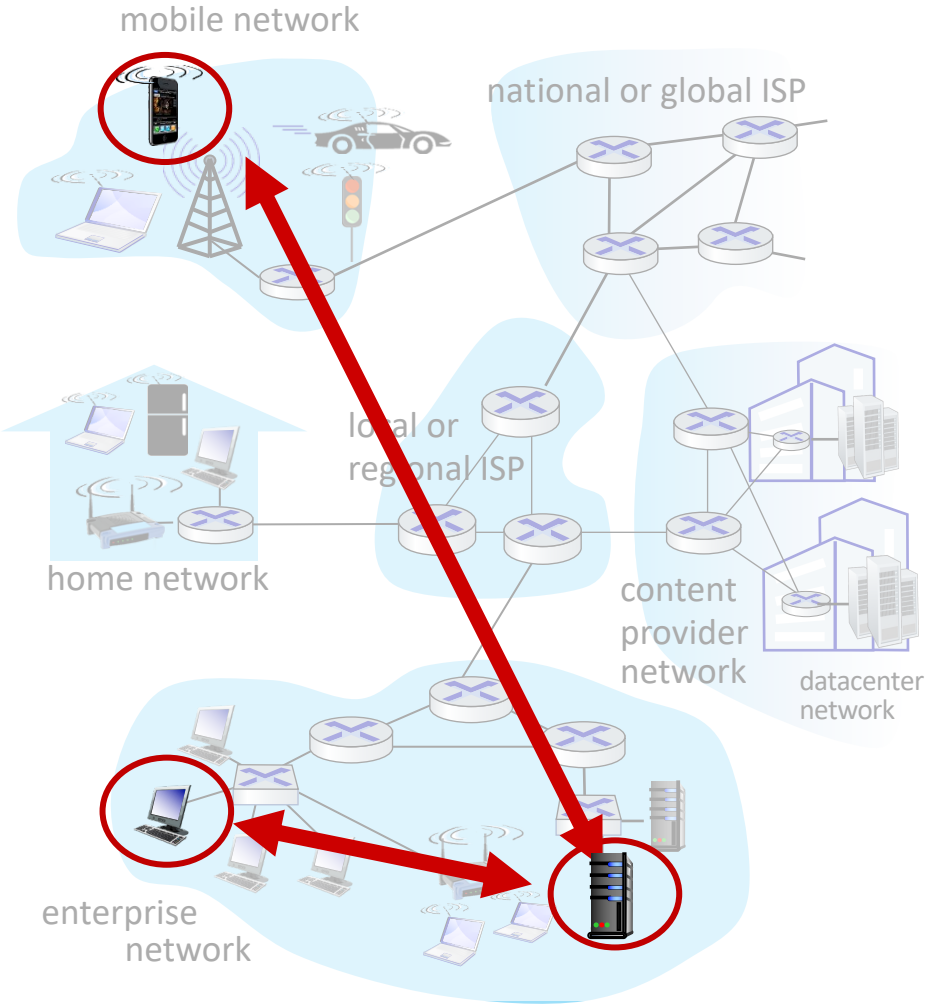
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

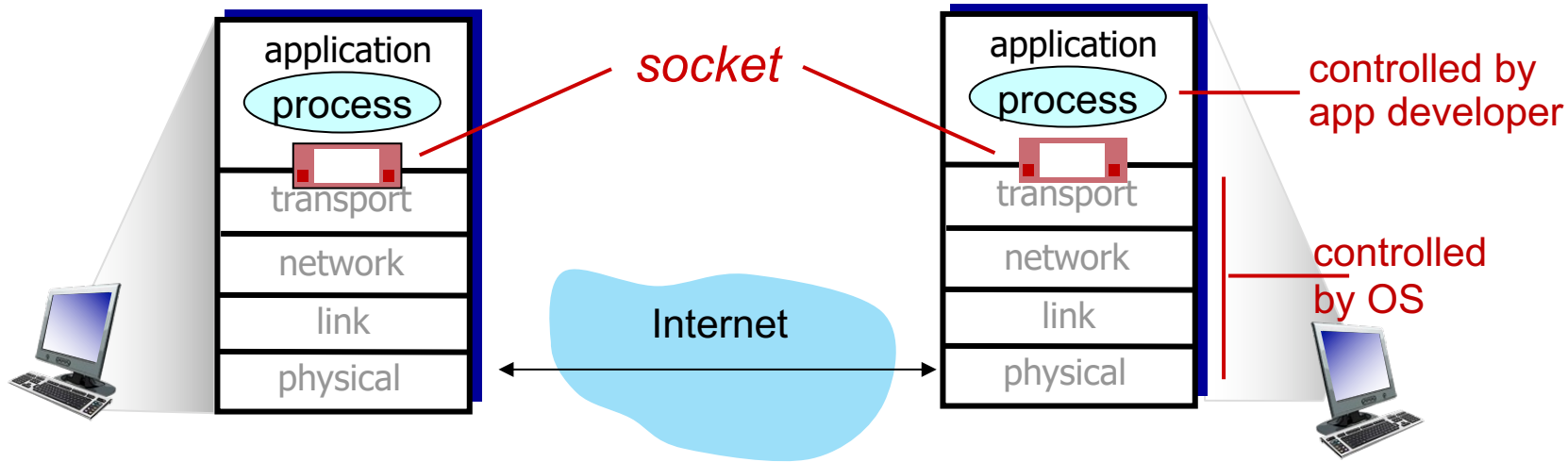
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



Sockets

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

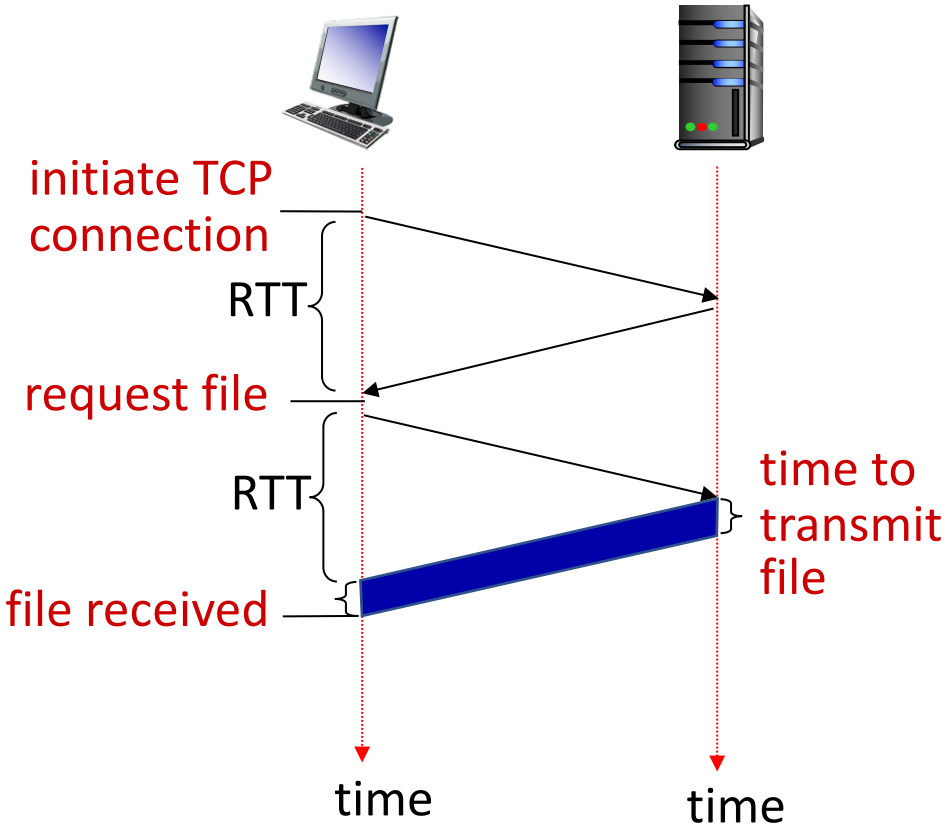


Non-persistent HTTP: response time

RTT (Round-Trip Time): time for a small packet to travel from client to server and back

HTTP response time (per object):

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



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

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

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

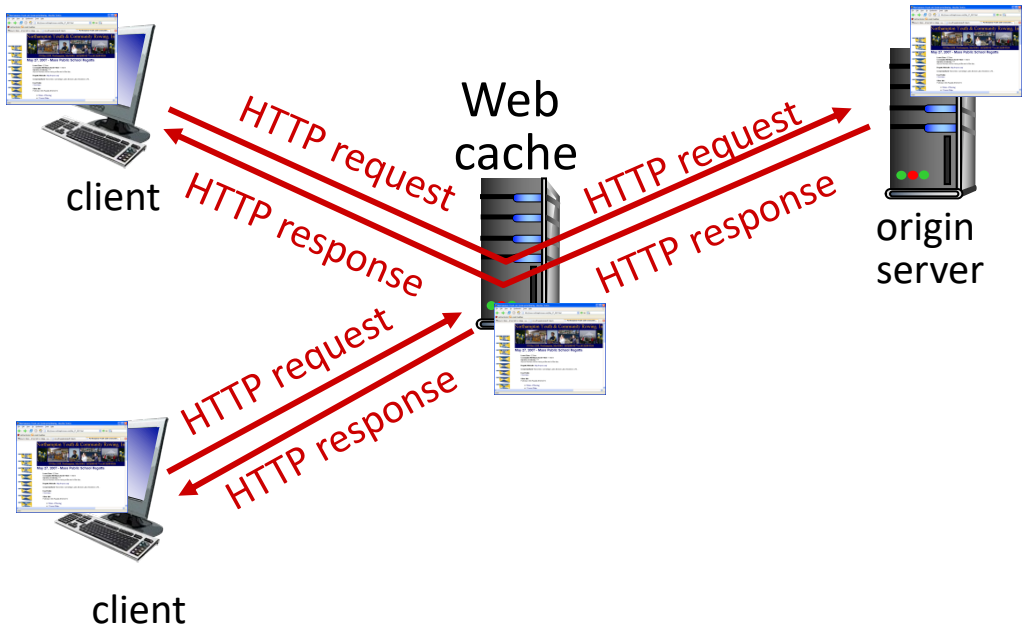
Persistent HTTP (HTTP1.1):

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

Web caches

Goal: satisfy client requests without involving origin server

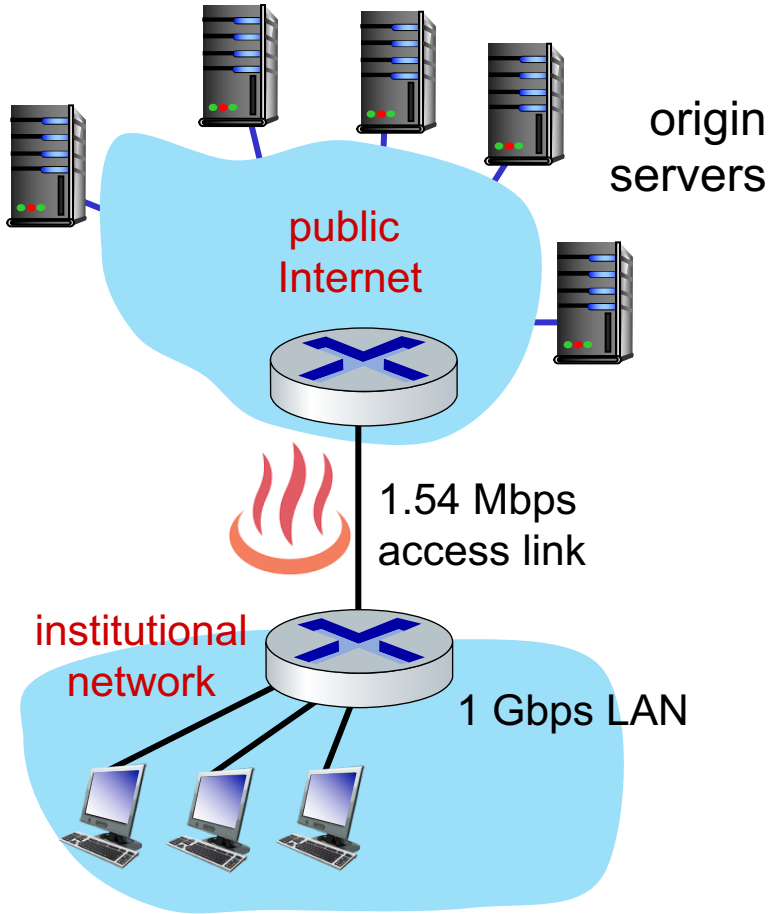
- user configures browser to point to a (local) *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



Caching example

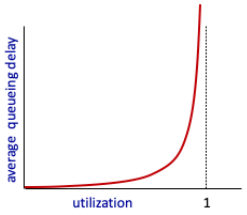
Scenario:

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



Performance:

- access link utilization = .97
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + minutes + usecs



problem: large queueing delays at high utilization!

Option 1: buy a faster access link

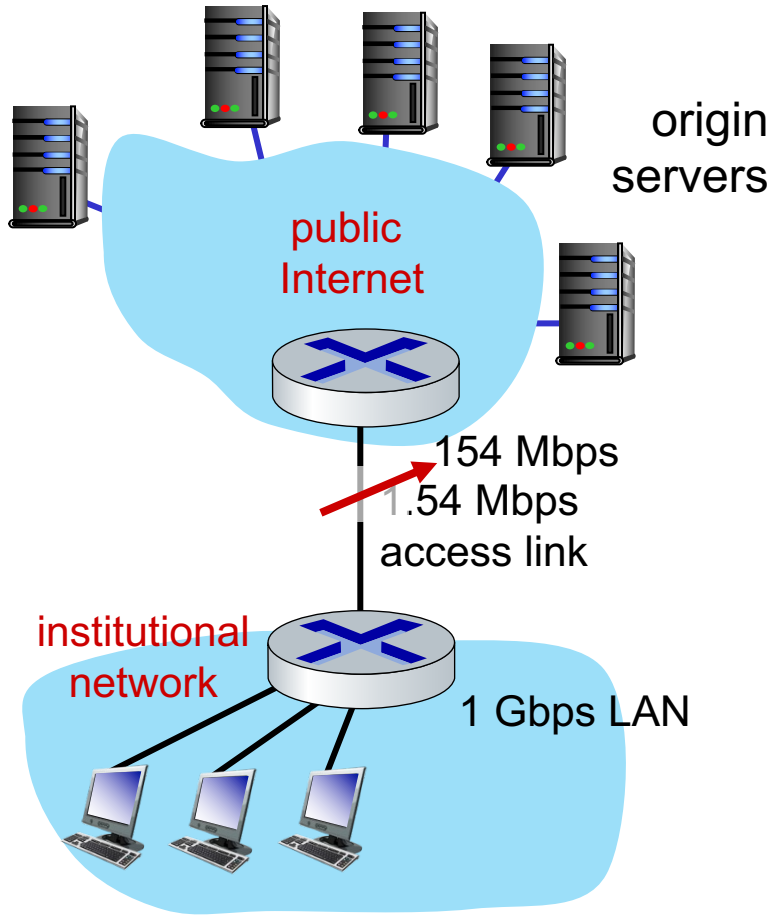
Scenario:

- access link rate: 1.54 Mbps → 154 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = .97 → .0097
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + minutes + usecs → msecs

Cost: faster access link (expensive!)



Option 2: install a web cache

Scenario:

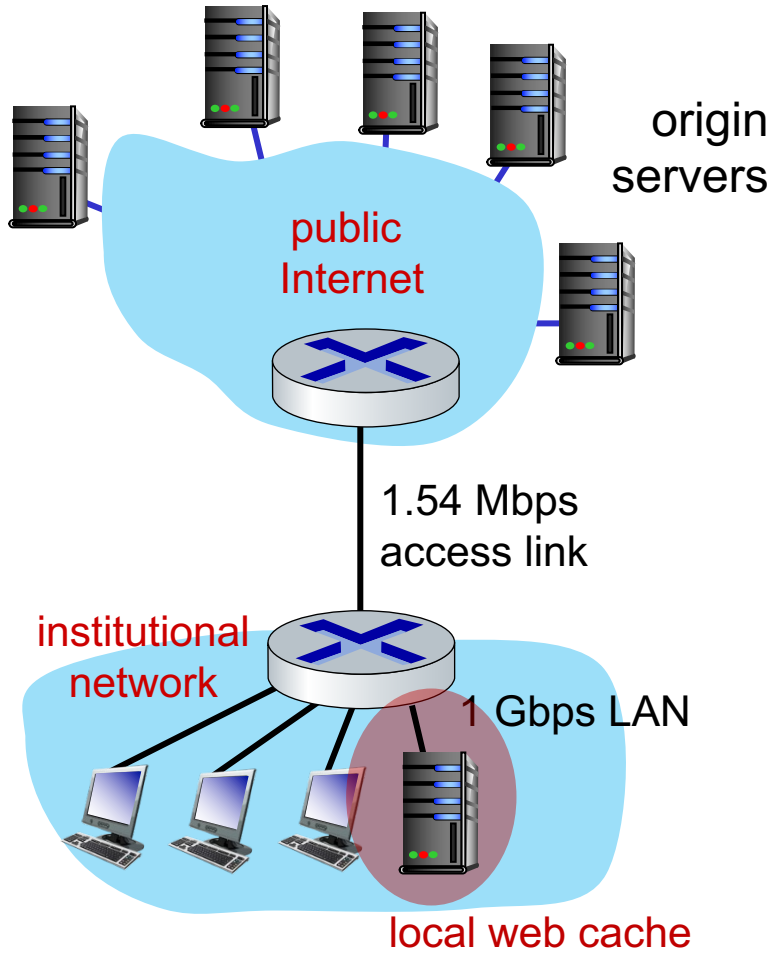
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

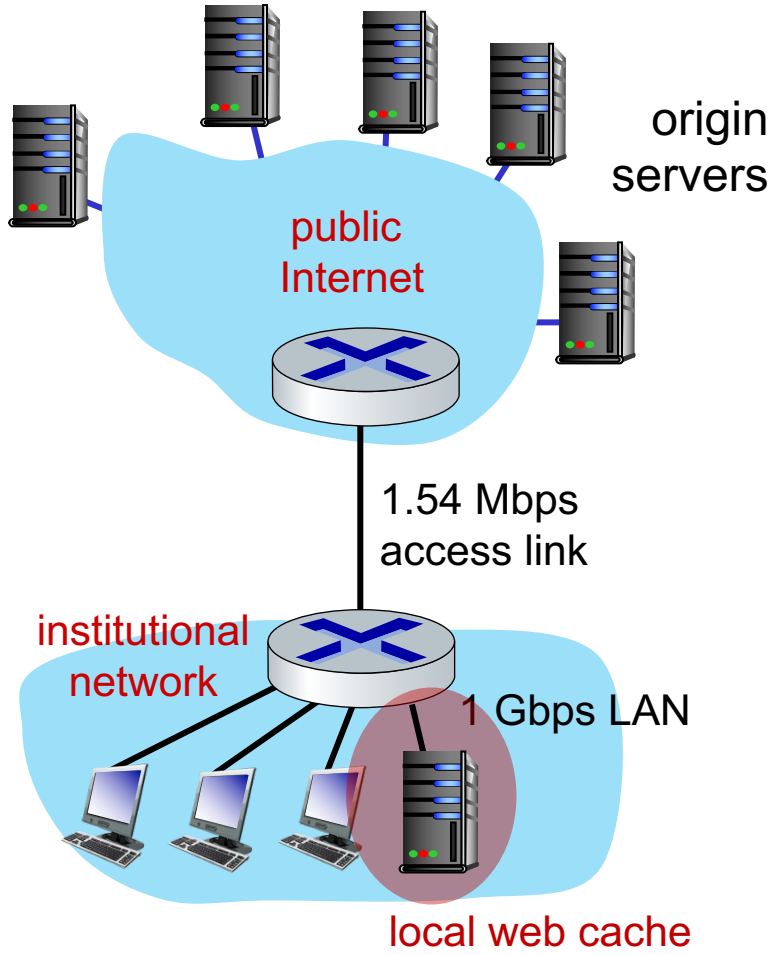
How to compute link utilization, delay?



Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:

- 40% requests served by cache, with low (msec) delay
- 60% requests satisfied at origin
  - rate to browsers over access link
    = 0.6 * 1.50 Mbps = .9 Mbps
  - access link utilization = 0.9/1.54 = .58 means low (msec) queueing delay at access link
- average end-end delay:
  = 0.6 * (delay from origin servers)
  + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

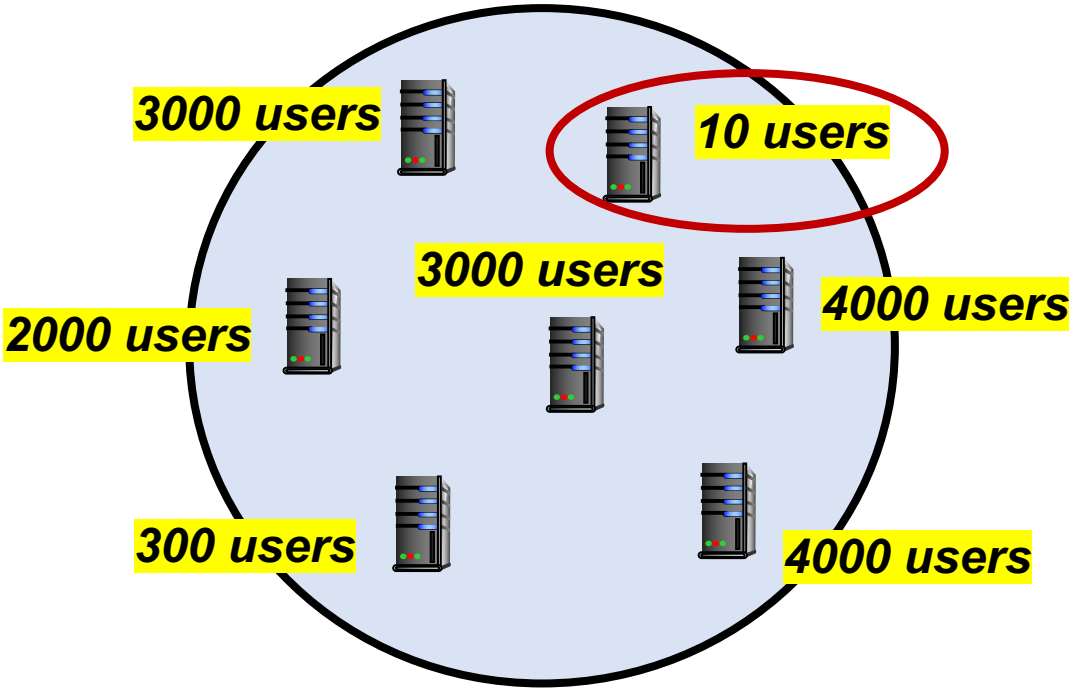


lower average end-end delay than with 154 Mbps link (and cheaper too!)

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



Multimedia: video coding

- Why we need video coding?

8 MByte = 64 Mbit

Example: a video with 30 fps

30 images per second

Data Rate: $30 \times 60Mbit = 1800Mbps = 1.8Gbps$

Transport layer: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

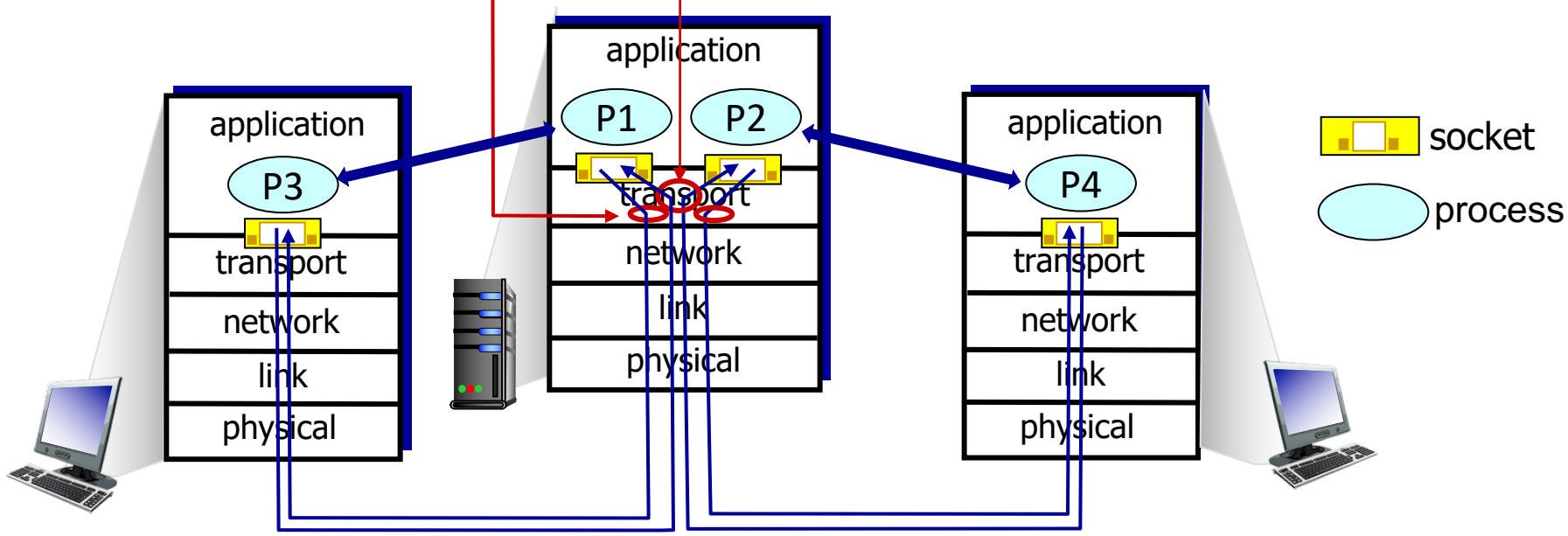
Multiplexing/demultiplexing

multiplexing at sender:

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

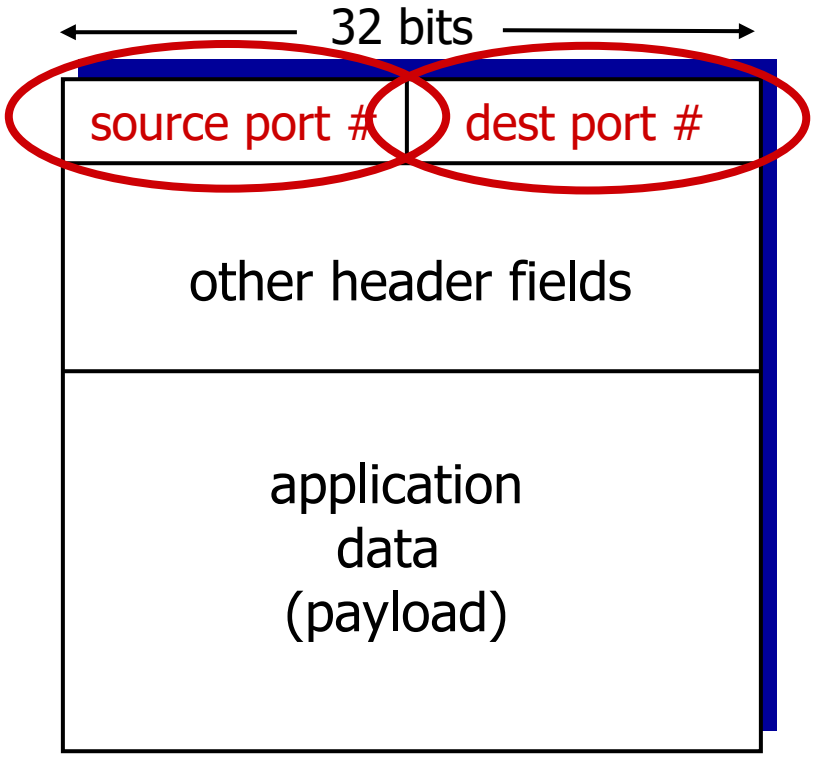
demultiplexing at receiver:

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

Connection-oriented demultiplexing

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

TCP sequence numbers, ACKs

Sequence numbers:

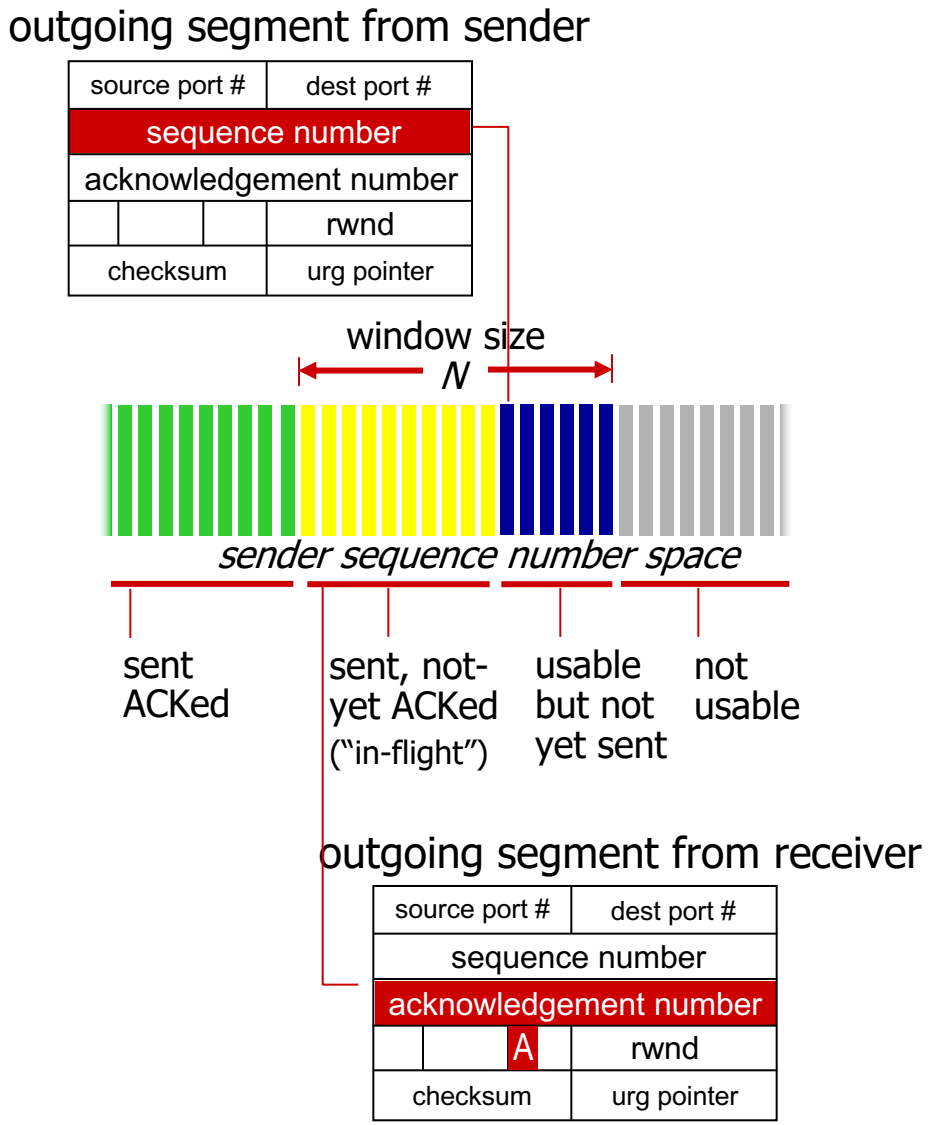
- byte stream "number" of first byte in segment's data

Acknowledgements:

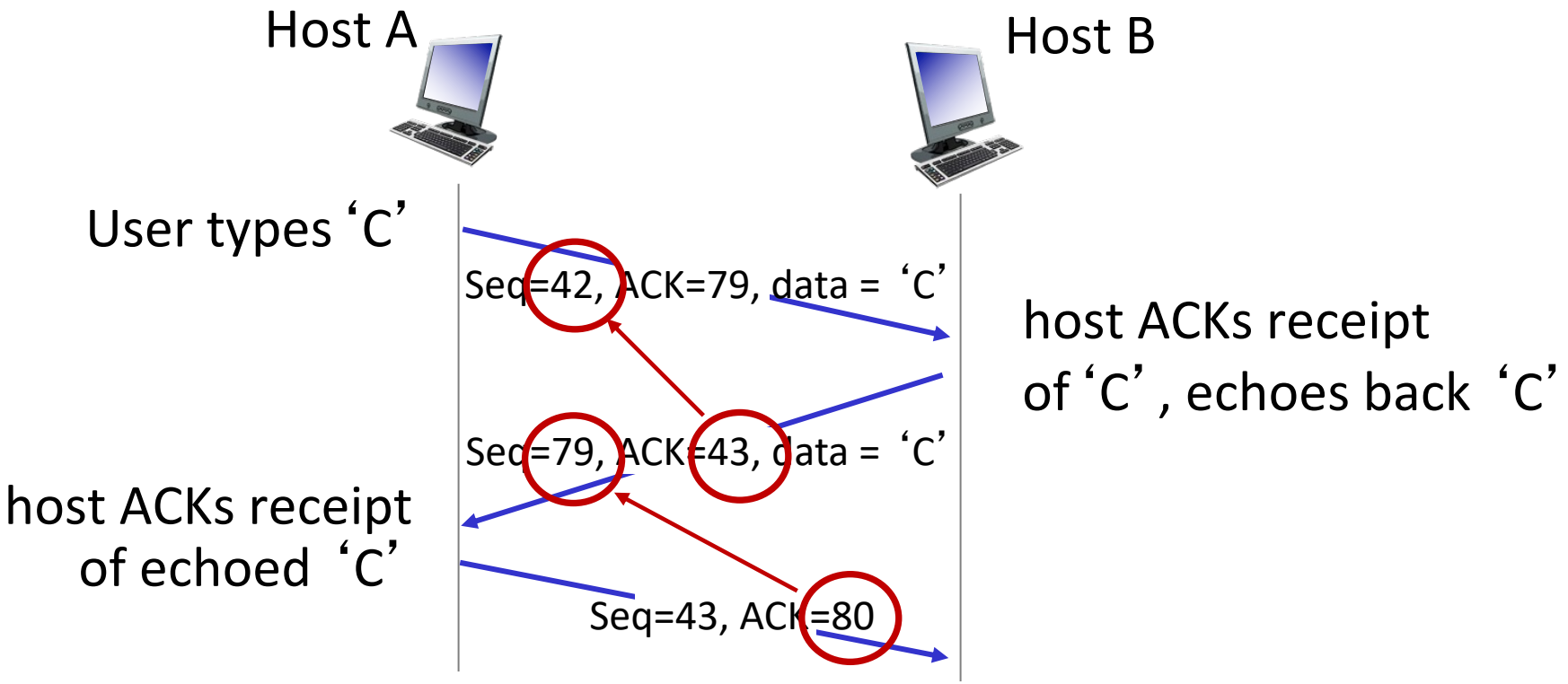
- seq # of next byte *expected* from other side
- *cumulative* ACK

Q: how receiver handles out-of-order segments (buffer or discard?)

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



TCP sequence numbers, ACKs



simple telnet scenario

TCP round trip time, timeout

- timeout interval: **EstimatedRTT** plus "safety margin"
  - large variation in **EstimatedRTT**: want a larger safety margin

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

estimated RTT

"safety margin"

- **DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

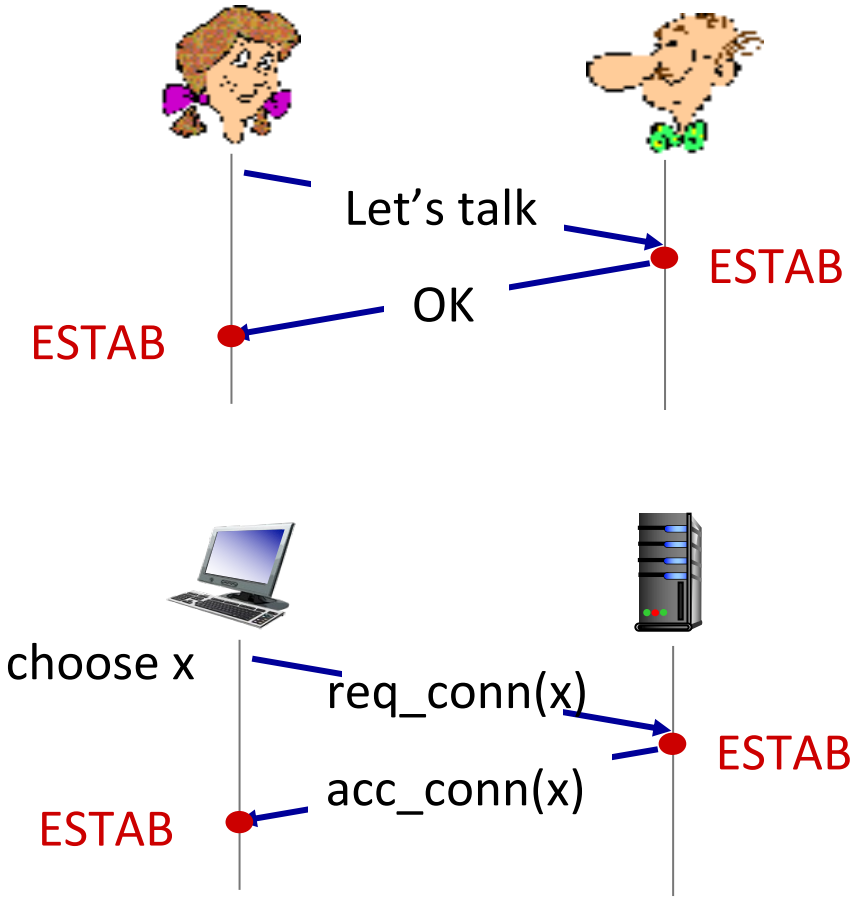
```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
```

(typically, β = 0.25)

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

Agreeing to establish a connection

2-way handshake:



Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

TCP 3-way handshake

Client state

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

LISTEN

```
clientSocket.connect((serverName,serverPort))
```

choose init seq num, x
send TCP SYN msg

SYNSENT

SYNbit=1, Seq=x

choose init seq num, y
send TCP SYNACK
msg, acking SYN

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ESTAB

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

Server state

```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
connectionSocket, addr = serverSocket.accept()
```

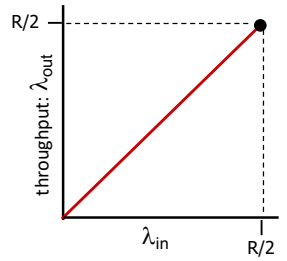
LISTEN

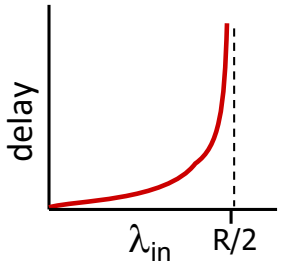
SYN RCVD

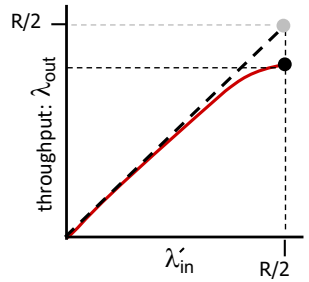
ESTAB

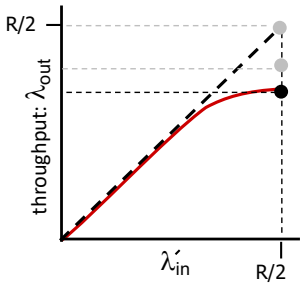
Causes/costs of congestion: insights

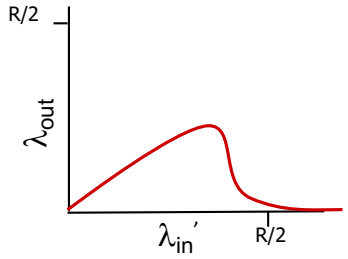
- throughput can never exceed capacity
- delay increases as capacity approached
- loss/retransmission decreases effective throughput
- un-needed duplicates further decreases effective throughput
- upstream transmission capacity / buffering wasted for packets lost downstream











TCP congestion control: AIMD

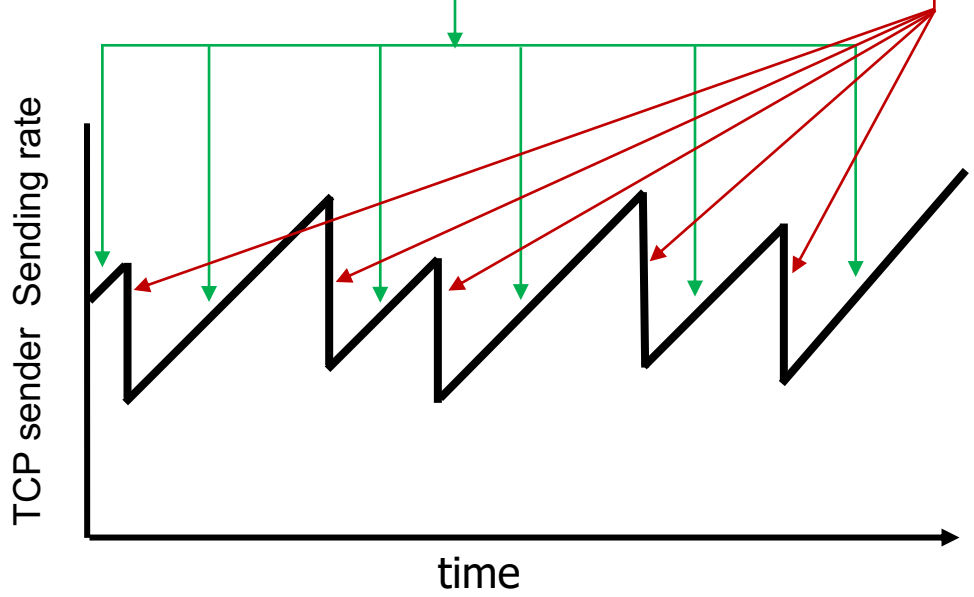
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP AIMD: more

Multiplicative decrease detail: sending rate is

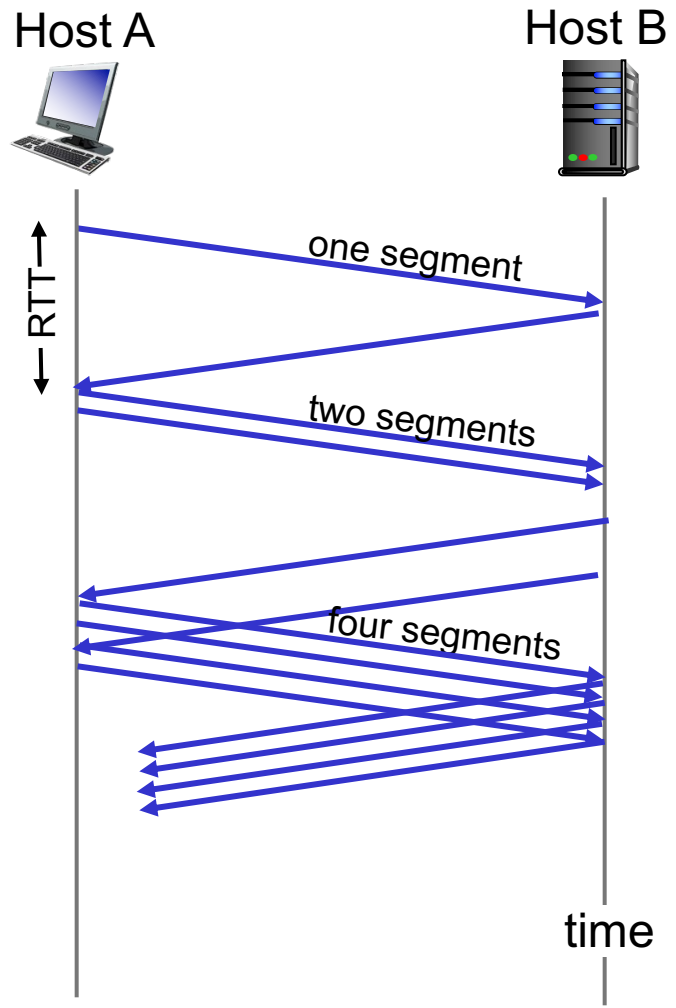
- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

Why AIMD?

- AIMD – a distributed, asynchronous algorithm – has been shown to:
  - optimize congested flow rates network wide!
  - have desirable stability properties

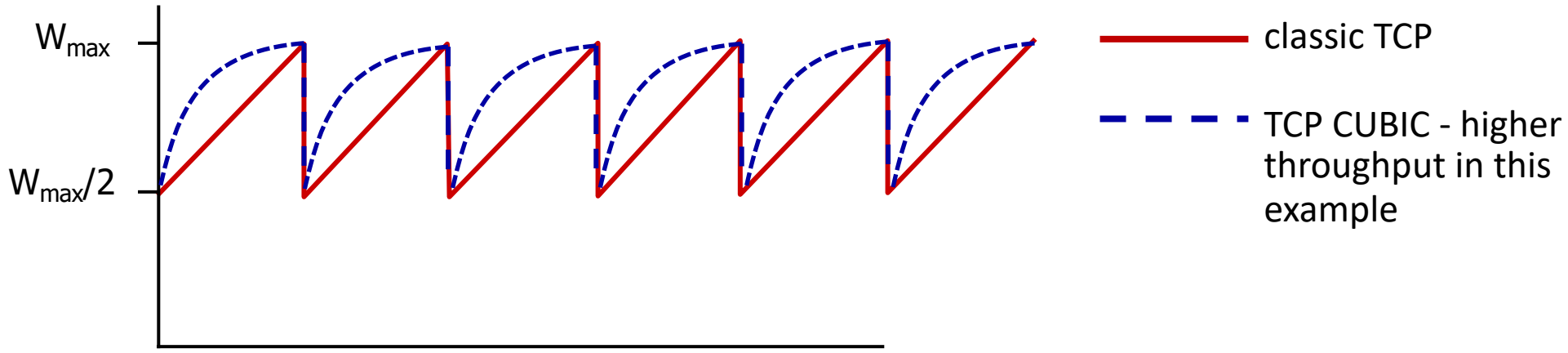
TCP slow start

- 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



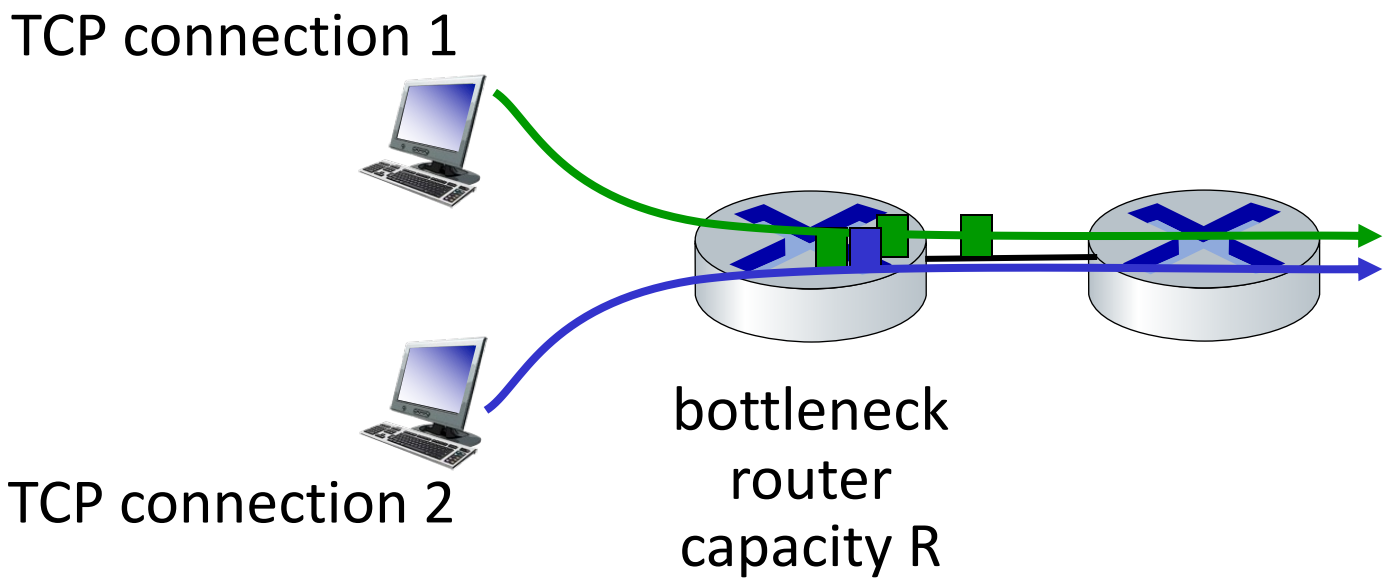
TCP CUBIC

- Is there a better way than AIMD to "probe" for usable bandwidth?
- Insight/intuition:
  - W_{max}: sending rate at which congestion loss was detected
  - congestion state of bottleneck link probably (?) hasn't changed much
  - after cutting rate/window in half on loss, initially ramp to to W_{max} *faster*, but then approach W_{max} more *slowly*



TCP fairness

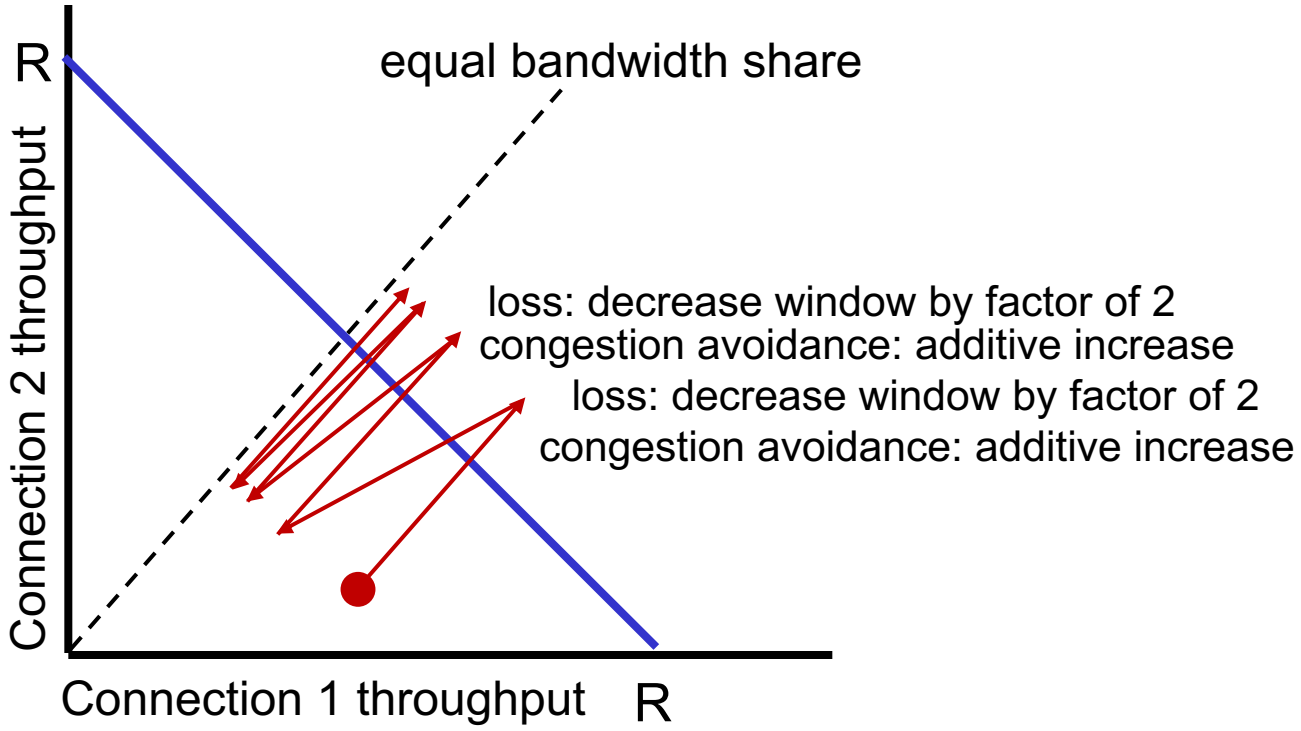
Fairness goal: if *K* TCP sessions share same bottleneck link of bandwidth *R*, each should have average rate of *R/K*



Q: is TCP Fair?

Example: two competing TCP sessions:

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



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance