IT 304
Computer Networks
Autumn 2022

Instructor: Kalyan Sasidhar P S

Associate Professor, FB 2109

kalyan_sasidhar@daiict.ac.in

Course Logistics

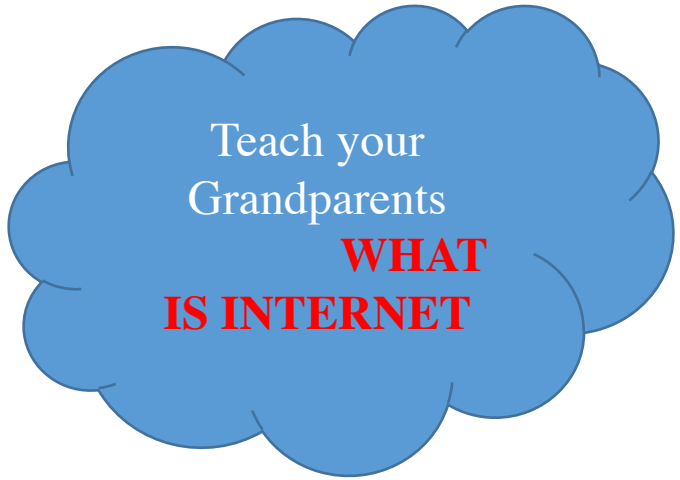
- Timings
  - Monday 11-11:50 AM
  - Wed 9-9:50 AM
  - Fri 8-8:50 AM
- Philosophy
  - Attend and pay attention
  - No MARKS/GRADE negotiations
- Textbooks:
  - **Computer Networking: A Top-Down Approach 7th edition, Jim Kurose, Keith Ross**
  - Computer Networks: A Systems Approach, 5th Edition, Larry Peterson and Bruce Davie

Experience

Batch taught	Years
MSc IT	2016-2020
BTech	2017, 2021

Component	Percentage
In-Sem I	15
In-Sem II	20
Labs	25
End sem	40

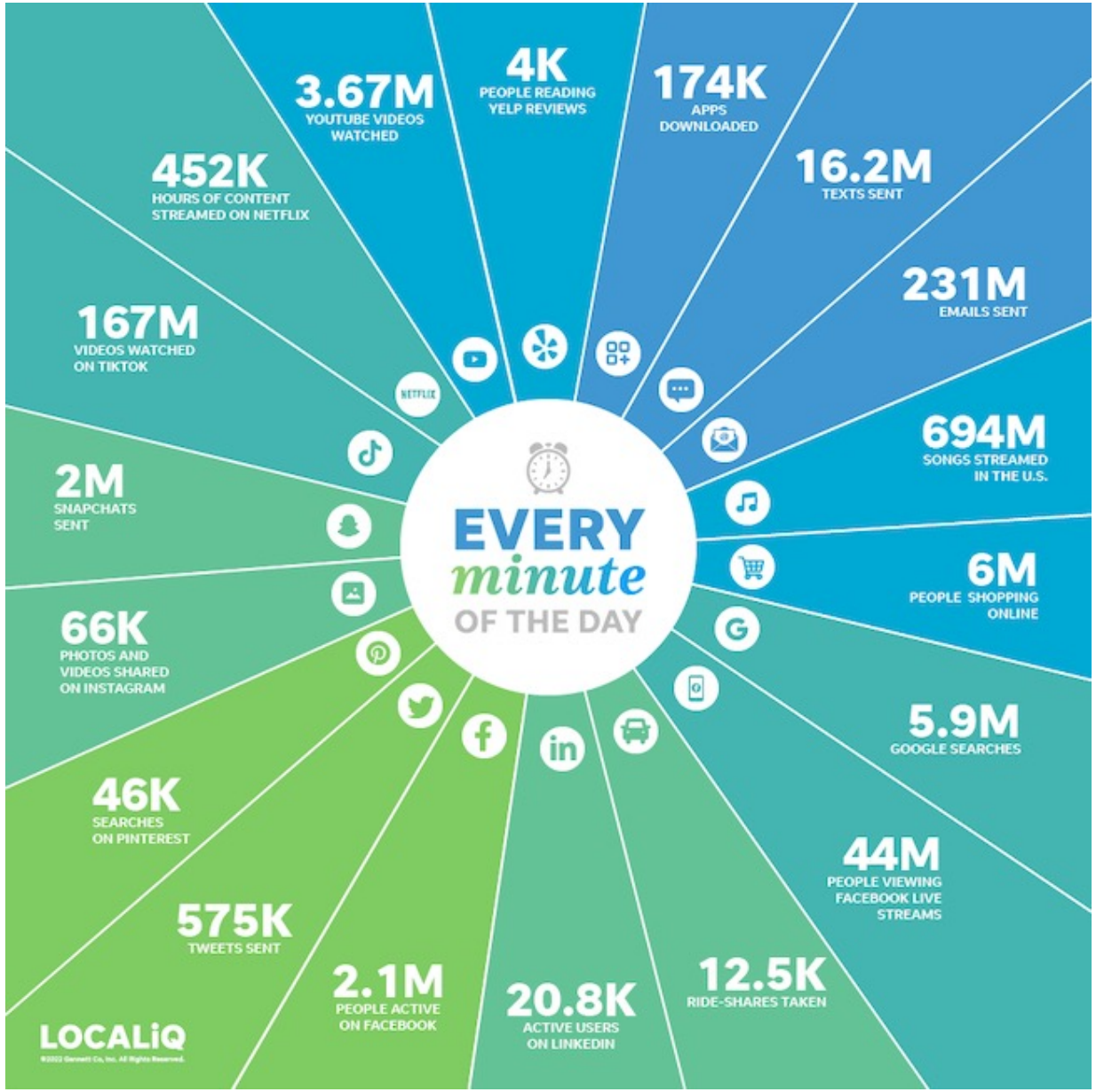
Goal



- Learn concepts underlying networks
- How do networks work? What can one do with them?
- Gain a basic understanding of the Internet
- Gain experience writing protocols
- Tools to understand new protocols and applications

What comes to your mind, when I say …..

- INTERNET
- World Wide Web
- IT 304



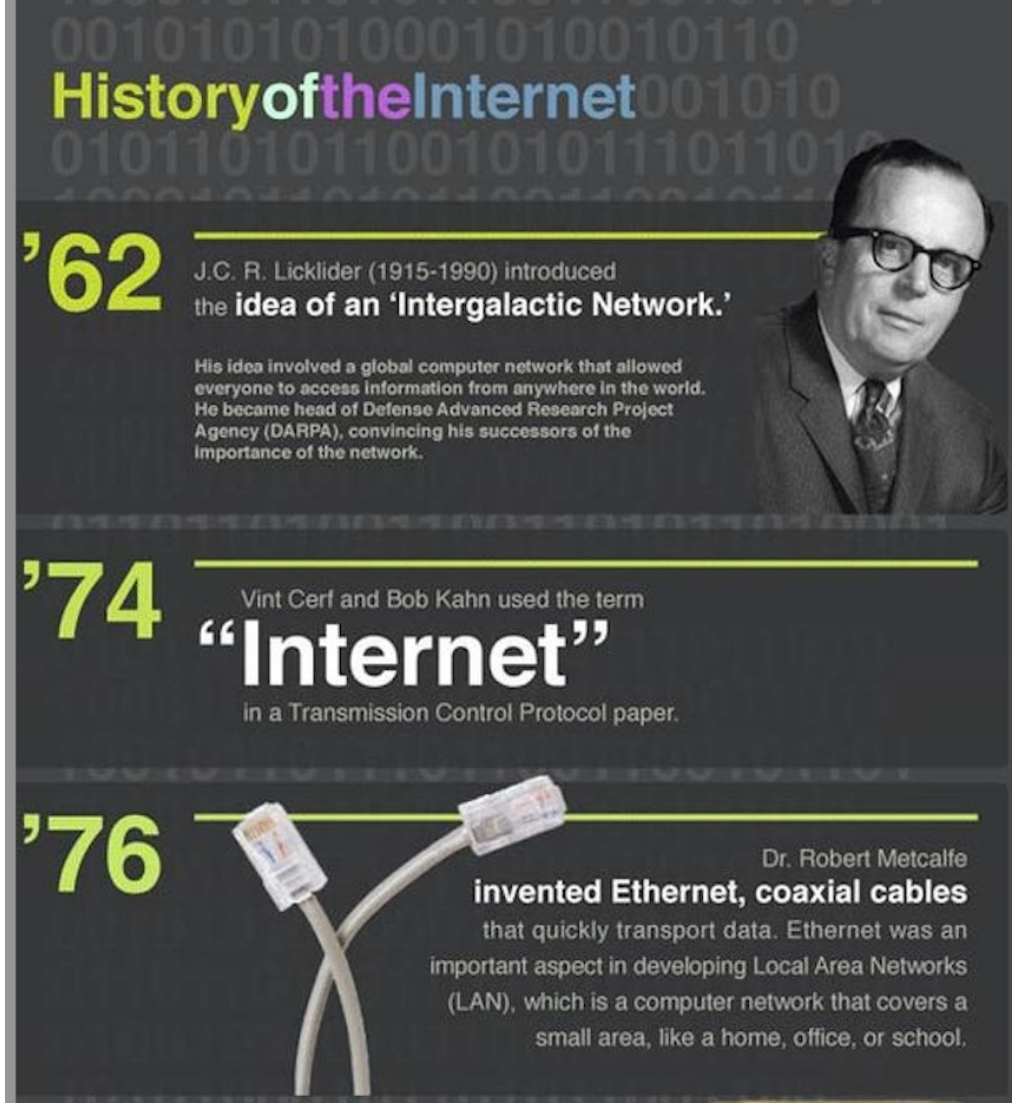
https://localiq.com/blog/what-happens-in-an-internet-minute/

Vincent Cerf

Tim Berners Lee

Paul Baran

JCR LickLider



https://www.history.com/topics/inventions/invention-of-the-internet

https://www.computerhistory.org/timeline/networking-the-web/

Chapter 1: Introduction

our goal:

- get "feel" and terminology
- more depth, detail *later* in course
- approach:
  - use Internet as example

overview:

- History
- What's the Internet?
- What's a protocol?
- Network edge; hosts, access net, physical media
- network core: packet/circuit switching, Internet structure
- performance: loss, delay, throughput
- security
- protocol layers, service models

Chapter 1: roadmap

1.1 history

1.2 what *is* the Internet?

1.3 network edge

- end systems, access networks, links

1.4 network core

- packet switching, circuit switching, network structure

1.5 delay, loss, throughput in networks

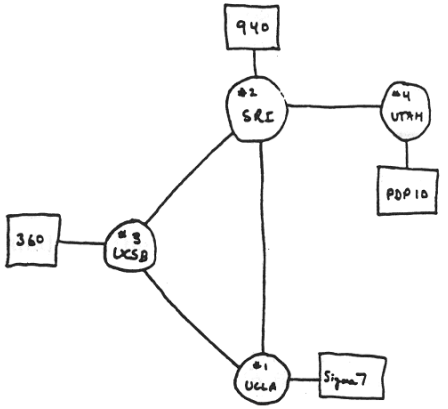
1.6 protocol layers, service models

Internet history

1961-1972: Early packet-switching principles

- 1961: Kleinrock - queueing theory shows effectiveness of packet-switching
- 1964: Baran - packet-switching in military nets
- 1967: ARPAnet conceived by Advanced Research Projects Agency
- 1969: first ARPAnet node operational

- 1972:
  - ARPAnet public demo
  - NCP (Network Control Protocol) first host-host protocol
  - first e-mail program
  - ARPAnet has 15 nodes



THE ARPA NETWORK

Internet history

1972-1980: Internetworking, new and proprietary nets

- 1970: ALOHAnet satellite network in Hawaii
- 1974: Cerf and Kahn - architecture for interconnecting networks
- 1976: Ethernet at Xerox PARC
- late70's: proprietary architectures: DECnet, SNA, XNA
- late 70's: switching fixed length packets
- 1979: ARPAnet has 200 nodes

Cerf and Kahn's internetworking principles:

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

define today's Internet architecture

Internet history

1980-1990: new protocols, a proliferation of networks

- 1983: deployment of TCP/IP
- 1982: smtp e-mail protocol defined
- 1983: DNS defined for name-to-IP-address translation
- 1985: FTP protocol defined
- 1988: TCP congestion control

- new national networks: Csnet, BITnet, NSFnet, Minitel
- 100,000 hosts connected to confederation of networks

Internet history

1990, 2000's: commercialization, the Web, new apps

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

late 1990's – 2000's:

- more killer apps: instant messaging, P2P file sharing
- network security to forefront
- est. 50 million host, 100 million+ users
- backbone links running at Gbps

Internet history

2005-present

- ~750 million hosts
  - Smartphones and tablets
- Aggressive deployment of broadband access
- Increasing ubiquity of high-speed wireless access
- Emergence of online social networks:
  - Facebook: soon one billion users
- Service providers (Google, Microsoft) create their own networks
  - Bypass Internet, providing “instantaneous” access to search, emai, etc.
- E-commerce, universities, enterprises running their services in “cloud” (eg, Amazon EC2)

INTERNET

What is a network?

A Network is…..

- **A**ll the components (H/W & S/W) involved in connecting computer across small and large distances

- **Importance of Networks:**
  - ✓Easy access and sharing of information
  - ✓Sharing of expensive devices and network resources
  - ✓Modern Technologies (IP telephony, Video on Demand, ….etc)

What are all part of a network?

Network components

- **Network has three main components**
  - ✓ **Computers** (servers and hosts)
    - Source of applications (network aware applications)
    - ex: HTTP (Hyper Text Transmission Protocol),
      FTP (File Transfer Protocol),
      SNMP (Simple Network Management Protocol)
      Telnet

  - ✓ **Network Devices**
    - Devices that interconnect different computers together
    - ex: Repeaters, hub, bridge, switch, router, NIC and modems

  - ✓ **Connectivity**
    - Media that physically connect the computers and network devices
    - ex: Wireless and cables

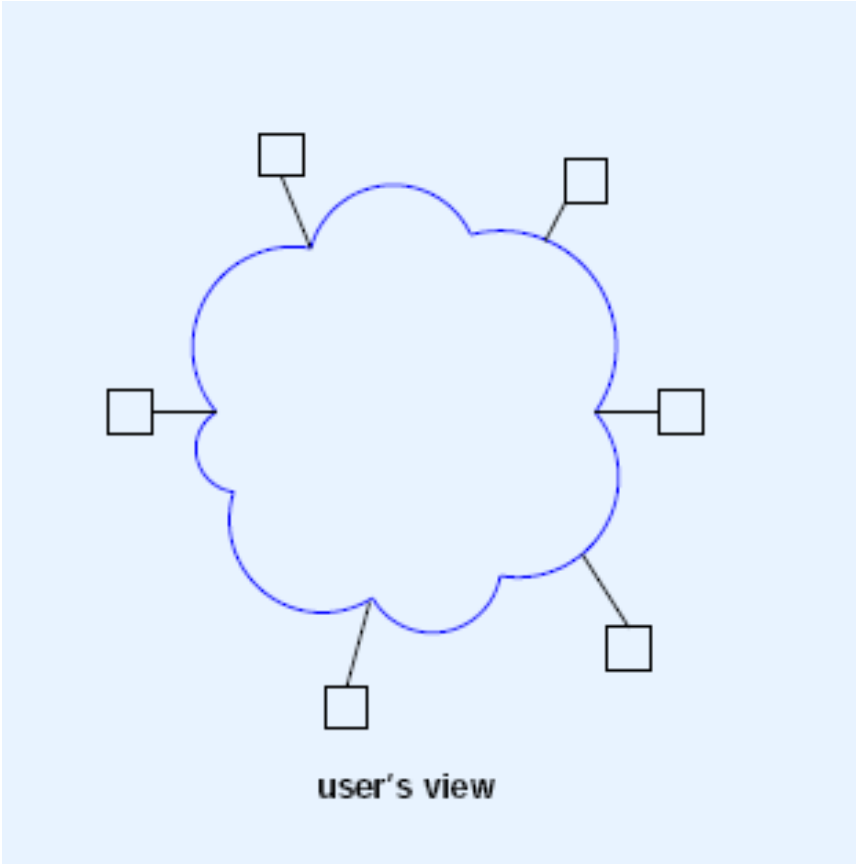
Network Types

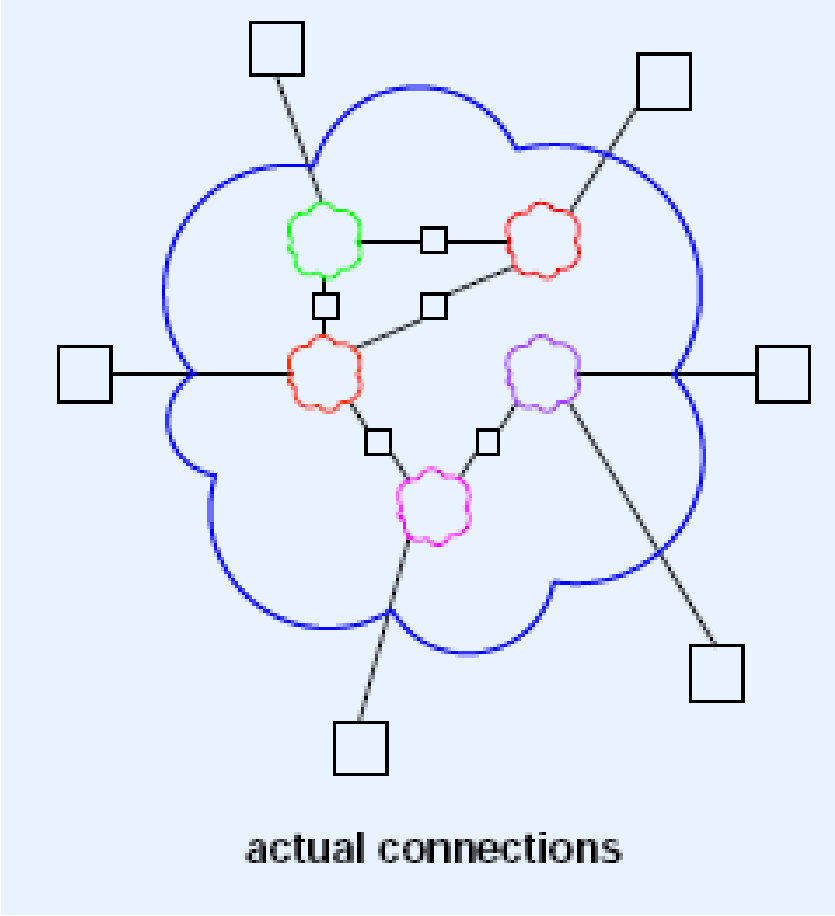
- **LAN (Local Area Network):**
  It is a group of network components that work within small area

- **MAN (Metropolitan Area Network):**
  It is a group of LANs that are interconnected within small area

- **WAN (Wide Area Network):**
  It is a group of LANs that are interconnected within large area

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





What is the use of Internet?

"We set up a telephone connection between us and the guys at SRI ...", Kleinrock ... said in an interview: "We typed the L and we asked on the phone,

> "Do you see the L?"
>
> "Yes, we see the L," came the response.
>
> We typed the O, and we asked, "Do you see the O."
>
> "Yes, we see the O."
>
> Then we typed the G, and the system crashed ...

https://www.youtube.com/watch?v=p5mASvEvDZc

IT 304
Computer Networks
Autumn 2022

Week 2-Lecture 1

8/8/2022

Recap: Network components

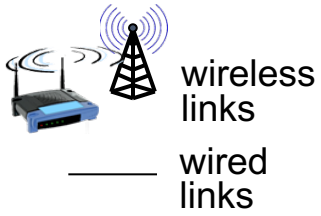
A host is a device that connects to the network. It can be a desktop computer, laptop, smartphone, etc. The hosting machine contains software that enables it to communicate over the network.

- **Network has three main components**
  - ✓ **Computers** (servers and hosts)
    - ✓ Hosts/End systems: hosts files, webpages, other resources
    - ✓ Clients: ask for the resrouces
    - ✓ Servers: give the requested resources
  - ✓ **Network Devices**
    - Devices that interconnect different computers together
  - ✓ **Connectivity**
    - Media that physically connect the computers and network devices

A server is a software or a hardware device that provides services to the other devices in the network. A client is a software or hardware that obtain services of a server. A server can connect multiple clients, and a single client can connect to multiple servers.

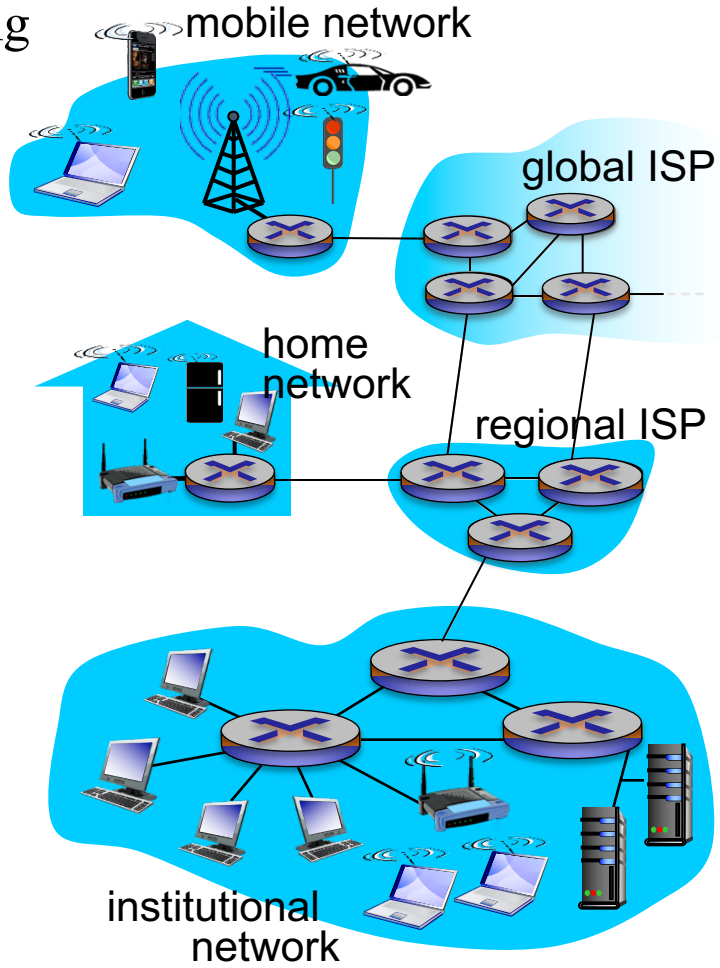
What's the Internet: "nuts and bolts" view

- billions of connected computing devices:
  - *hosts = end systems*
  - running *network apps-Meaning?*



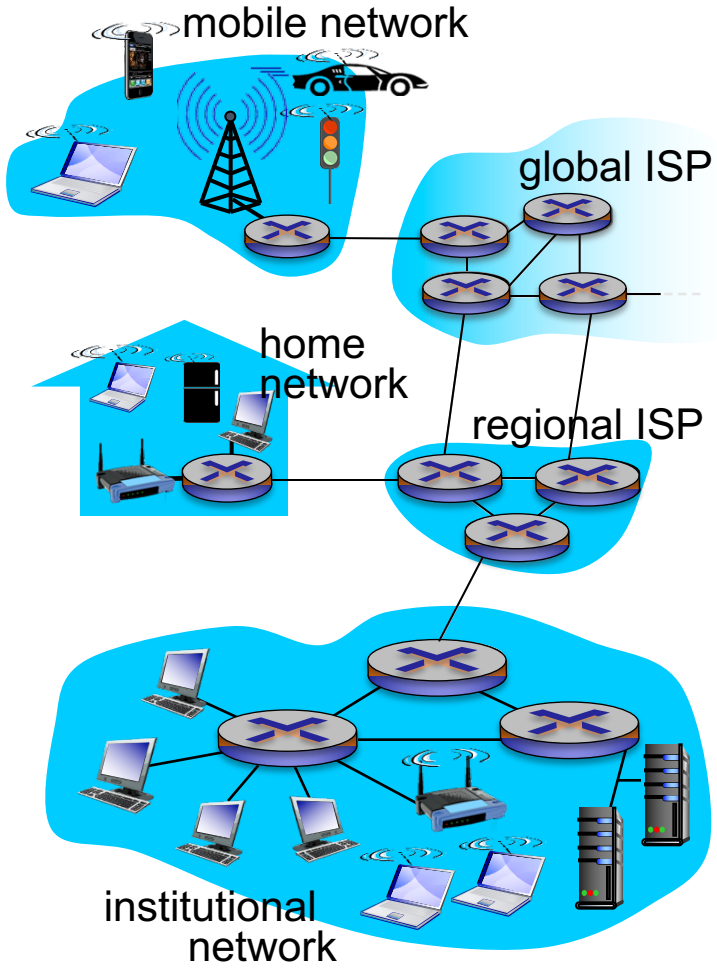
- *communication links*
  - fiber, copper, radio, satellite
  - transmission rate: *bandwidth*

- *packet switches:* forward packets (chunks of data)
  - *routers* and *switches*



What's the Internet: a service view

- *infrastructure that provides services to applications:*
  - Web, VoIP, email, games, e-commerce, social nets, …
- *provides programming interface to apps*
  - hooks that allow sending and receiving app programs to "connect" to Internet
  - provides service options, analogous to postal service



What is a computer network?

A set of network elements connected together, that implement a set of protocols for the purpose of sharing resources at the end hosts

- **Three important components**:
  - **Core infrastructure:**
    - A set of network elements connected together
  - **Protocols:**
    - Needed to use the network
  - **Purpose:**
    - Sharing resources at the end hosts (computing devices)

What do computer networks do?

A computer network delivers data between the end points/hosts

- **One and only one task**: Delivering the data

- This delivery is done by:
  - Chopping the data into **packets**
  - Sending individual packets across the network
  - Reconstructing the data at the end points

Evolution of three components of computer networks!

- Infrastructure, protocols, purpose

Data delivery as a fundamental goal

- Support the logical equivalence of **Interprocess Communication (IPC)**
  - Mechanism for "processes on the same host" to exchange messages
- Computer networks allow "processes on two different hosts" to exchange messages
- Clean separation of concerns – Division of roles
  - Computer networks deliver data
  - Applications running on end hosts decide what to do with the data
- Keeps networks simple, general and application-independent

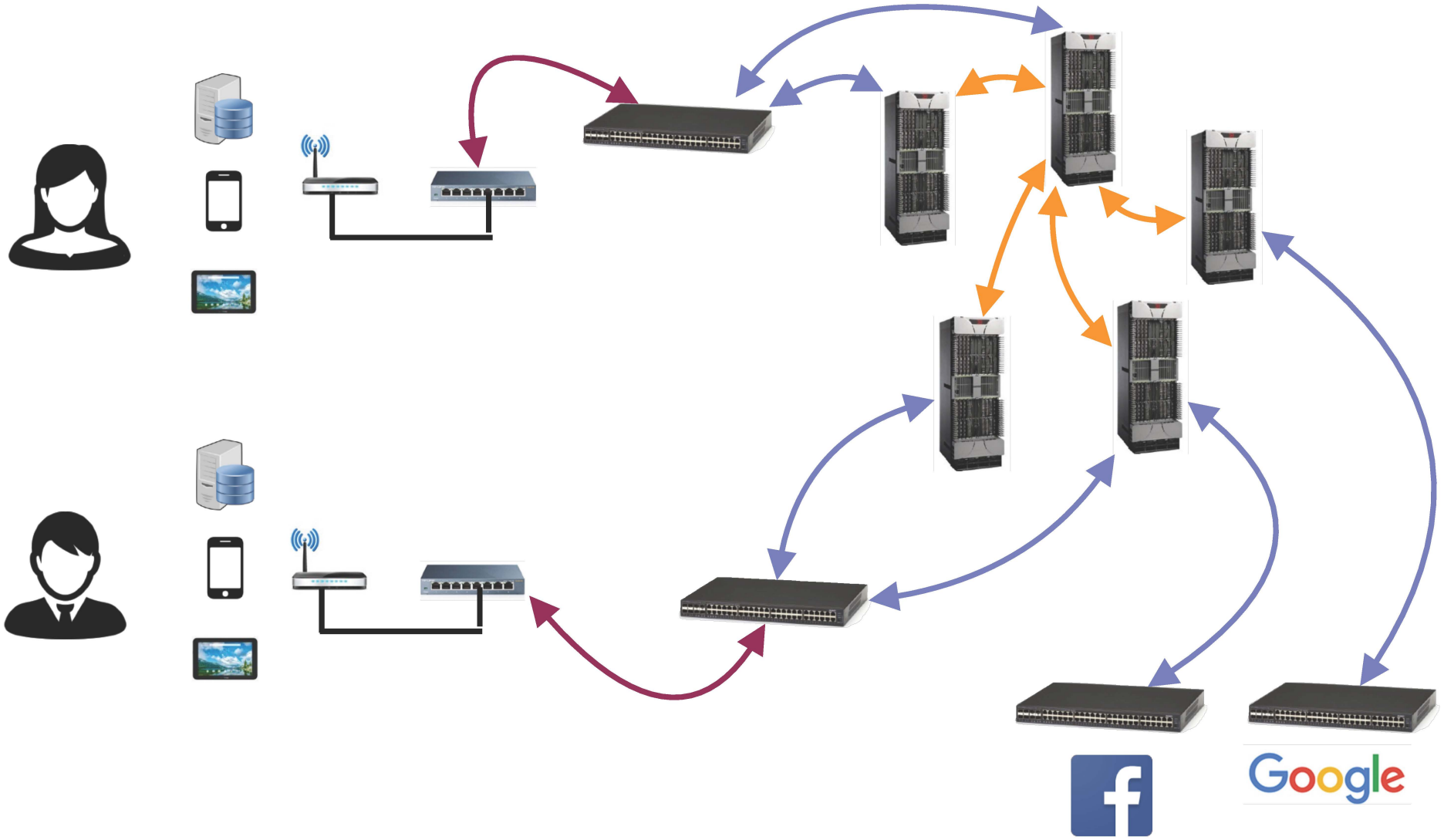
1. A network consists of
A. End hosts
B. Network core devices
C. Links
D. All of the above
E. None of the above

2. Is there a difference between host and an end system?
A. Yes
B. No

3. The Internet provides
A. Best effort service
B. Guaranteed service

What do computer networks look like?

End hosts, switches/routers, links



Chapter 1: roadmap

1.1 what *is* the Internet?

1.2 network edge

- end systems, access networks, links

1.3 network core

- packet switching, circuit switching, network structure

1.4 delay, loss, throughput in networks

1.5 protocol layers, service models

1.6 networks under attack: security

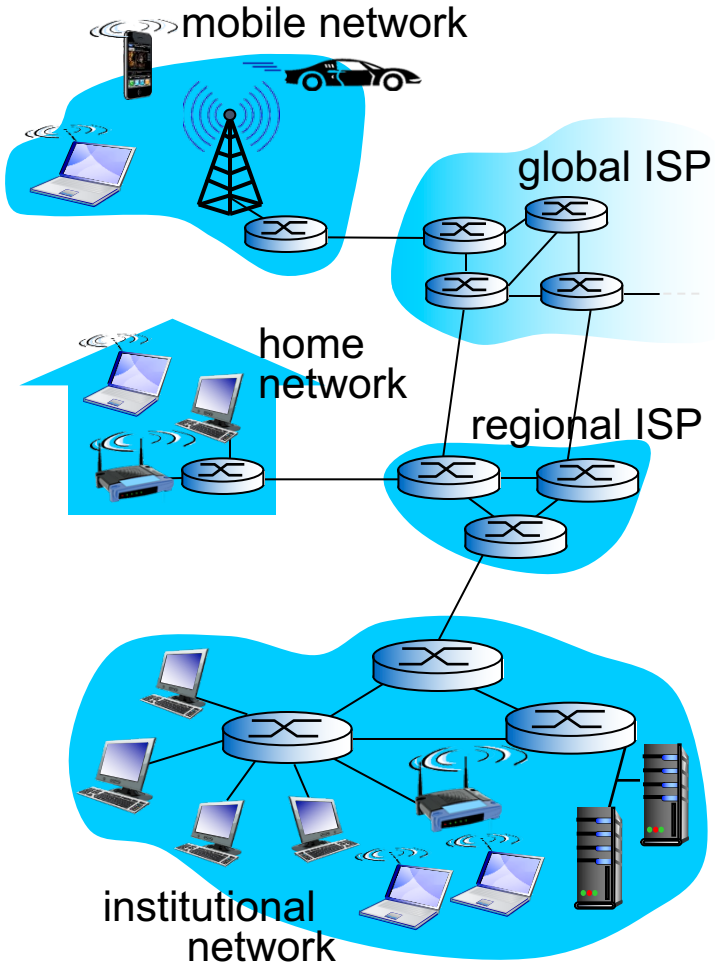
1.7 history

A closer look at network structure:

- *network edge:*
  - hosts: clients and servers
  - servers often in data centers

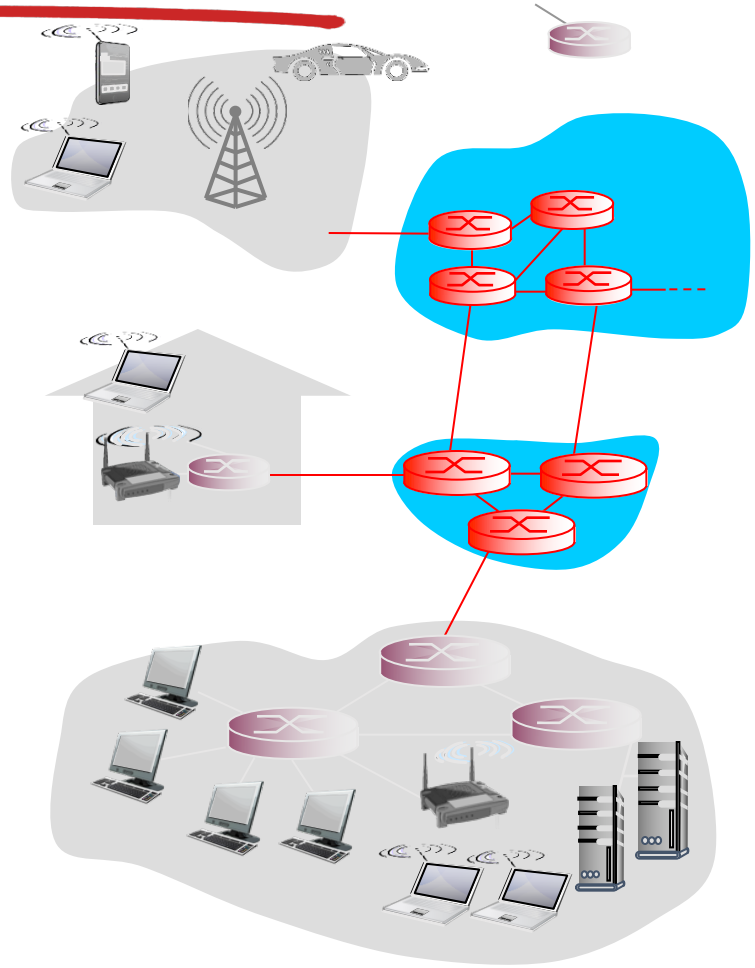
- *access networks, physical media:* wired, wireless communication links

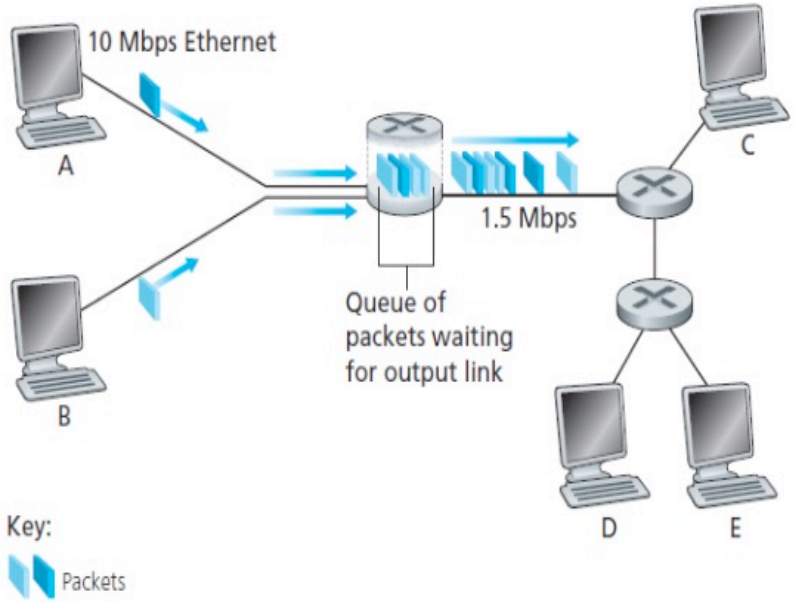
- *network core:*
  - interconnected routers
  - network of networks



The network core

- mesh of interconnected routers
- packet-switching: hosts break application messages into *packets*
  - forward packets from one router to the next, across links on path from source to destination
  - each packet transmitted at full link capacity





Architectural principles, design goals and performance objectives in wired networks

- **What tasks get done?**
  - What is delivered (packets, files, …)?
  - What are the semantics (reliability, ordering, …)?

- **Where do tasks get done?**
  - At the network elements? At the end-hosts?
  - How do end hosts interface with network elements?
  - How do different network elements interface with each other?

- **How tasks get done?**
  - What protocols and algorithms do each of these use?
  - How to achieve various performance objectives (latency, etc.)?

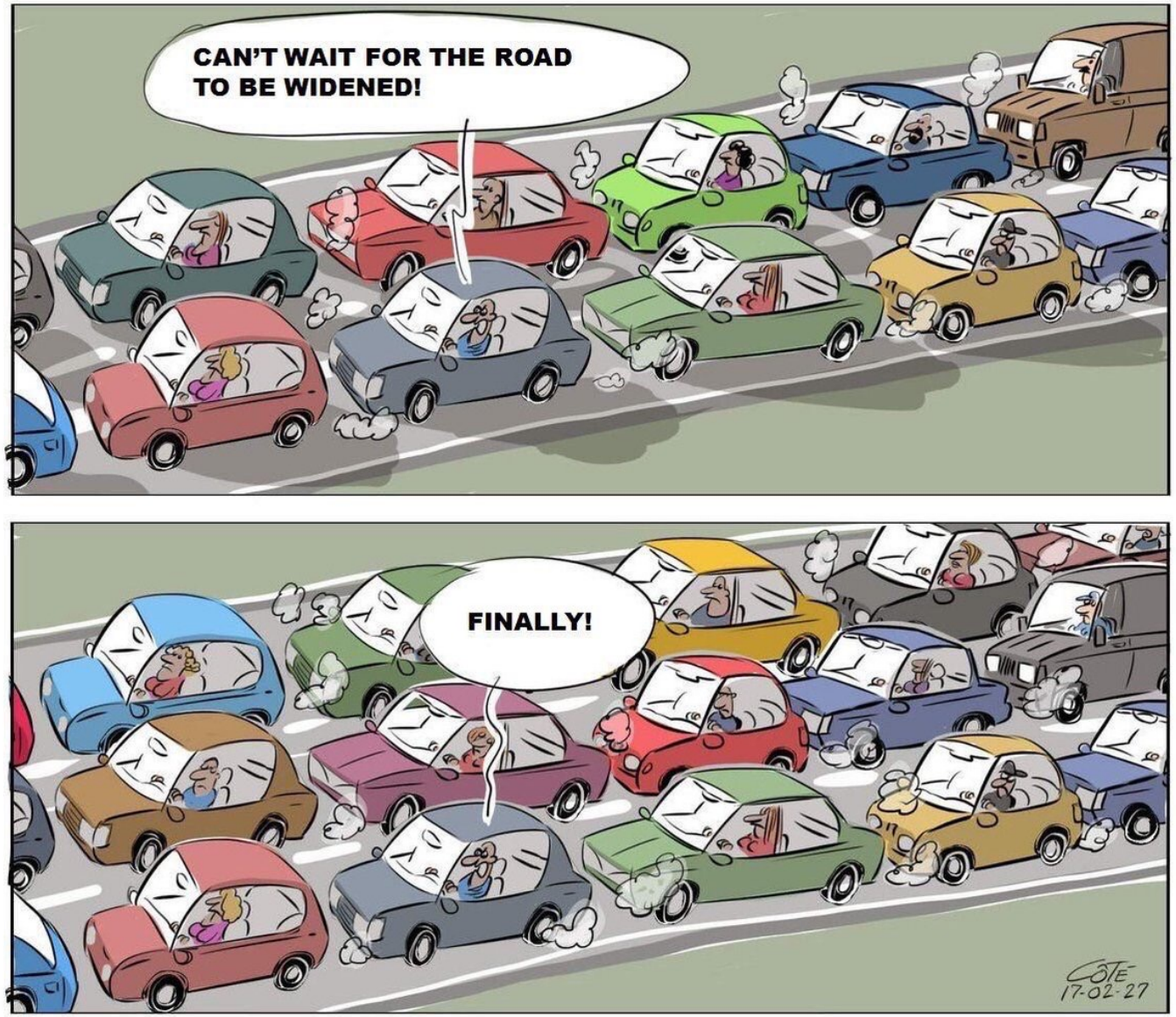
Many mechanisms: What do we mean by …

- Locating a destination? →Naming, Addressing
- Finding path to the destination → Routing
- Sending data to the destination → Forwarding
- Failures, reliability,etc..→ Congestion control

What are the performance metrics?

Capacity

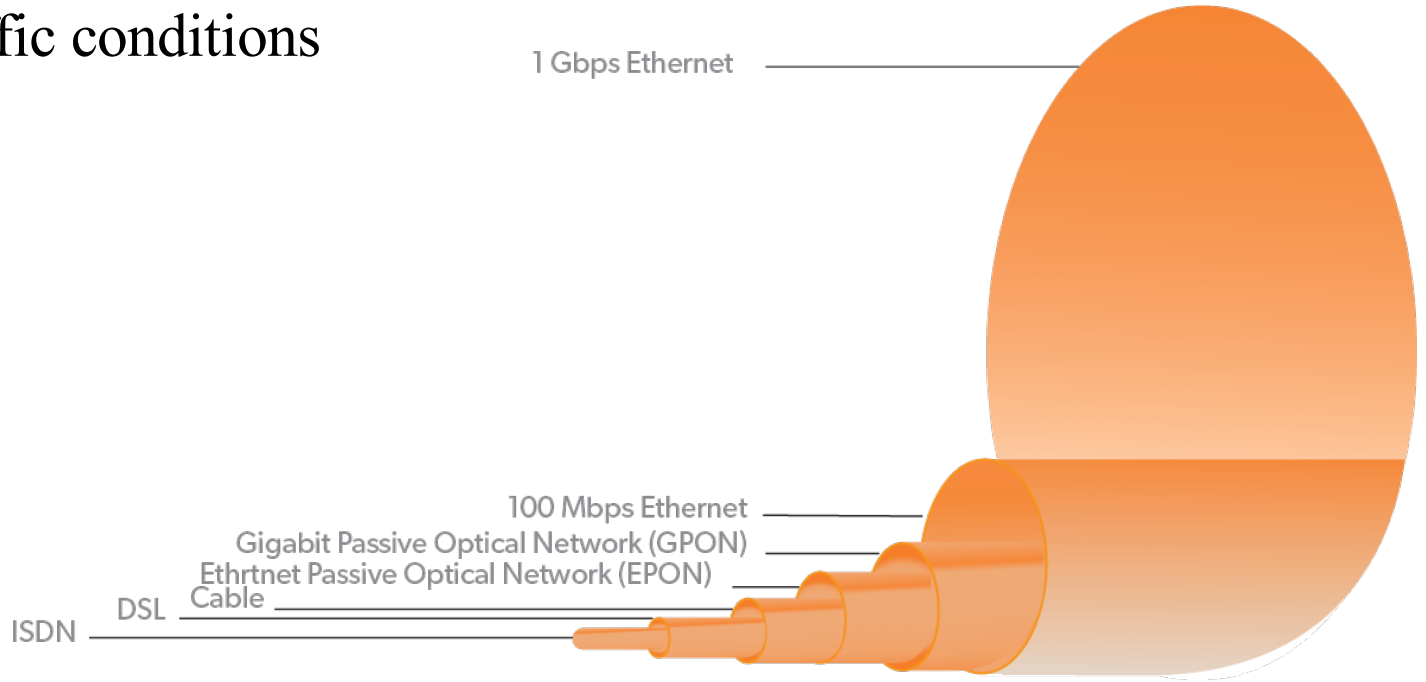
- How wide is the road?
- How fat is the tunnel?
- How many cars can fit at a time?
  - One, two, three…?



Performance metrics in computer networks!

Capacity is Bandwidth: Number of bits sent per unit time (bits per second, or bps)

- Depends on
  - Hardware
  - Network traffic conditions

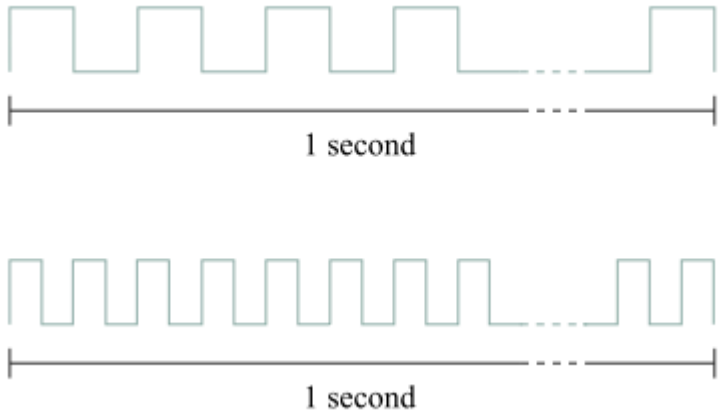


Ref [1]

Ref [1]: https://www.elandcables.com/the-cable-lab/faqs/faq-what-is-bandwidth-in-cables

- Each bit is a pulse of some width.
- For example, each bit on a
  - 1-Mbps link is **1 µs** wide
  - 2-Mbps link is **0.5 µs** wide,

- The narrower each bit can become, the higher the bandwidth.
  - This means more bits can get inside the tunnel
  - So MORE DATA CAN FLOW WITHIN A TIME



Time taken is Propagation delay: Time for one bit to move through the link (seconds)

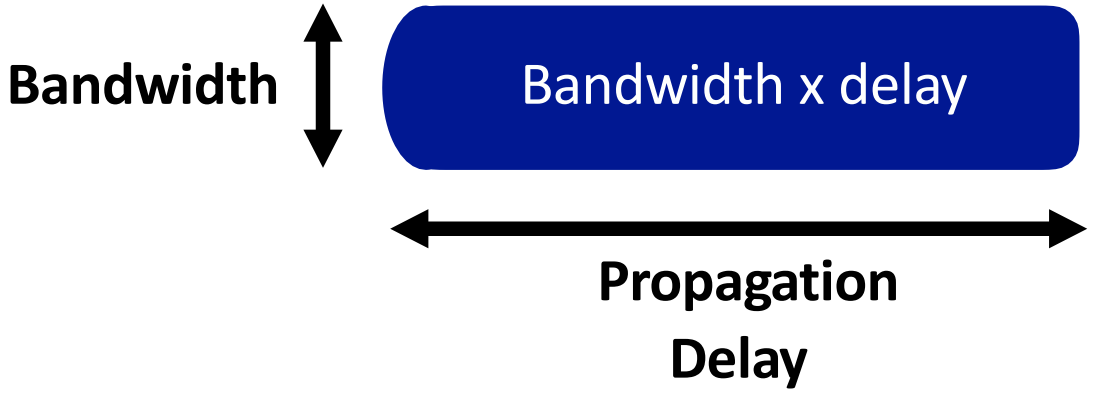
- Depends on
  - Hardware
  - Distance between machines

Propagation Delay

Bandwidth-delay product (BDP)

Number of bits "in flight" at any point of time (bits)

- Bits sent, but not received



- Same city over a slow link
  - Bandwidth: ~100Mbps
  - propagation delay: ~0.1ms
  - BDP = 10,000 bits (1.25KBytes)
- Between cities over fast link:
  - Bandwidth: ~10Gbps
  - propagation delay: ~10ms
  - BDP = 100,000,000 bits (12.5MBytes)

Access network means

A. The communication links

B. The routers and switches

C. A

D. B

E. Neither of them

Bandwidth is

A. The number of bits sent per unit time

B. Size of the data generated

Traffic conditions impact both bandwidth and delay

A. True

B. False

by traffic available bandwidth changes

Google celebrates its birthday on September 27, although no one really knows the exact date when it was founded. Started by two Stanford college friends, Larry Page and Sergey Brin, in 1998, it is a multi-billion dollar enterprise now. The name comes from a simple misunderstanding when they were searching for another, actual, word that existed in academia and meant a particular number. What word is that and what number does it denote?

Googol, 1 followed by 100 zeros

Gmail was launched by Google on April 1, 2004, which led many to believe it was an April Fool's joke. Before this service the term 'G-mail' already existed from as early as 1998. This was used online by fans of a certain fictional obese cat, and the original G-mail was known as "e-mail with cattitude". What does the G stand for in the original G-mail?

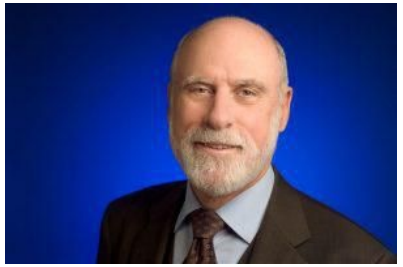
Garfield

When Page and Brin built the first server rack for Google at Stanford, they were looking for a cabinet to house it that was easy to assemble and disassemble. The server contained ten 4GB hard disks and two cooling fans. What colourful and bountifully found system did they use to build the server stack?

Lego bricks

Who is the father of the Internet?

Vincent Cerf



Why study computer networks?

#1: Has transformed and more importantly, <u>is transforming everything</u>!

- **Industry: core to and creator of many large and influential companies**
  - Google, Facebook, Apple, Cisco, Juniper, Akamai
- **Communication**
  - Email, messenger, phones, VoIP, …
- **Travel**
  - AirBnB, Uber, Maps, …
- **Health**
  - Digital health, remote diagnostics, ….
- **Entertainment**
  - Netflix, Prime
- **Relationships**
  - Facebook, Instagram, Snapchat, the list is endless…

Why study computer networks?

#2: To learn how to design for scale!

- **Tremendous scale**
  - 51% of world population
  - 1.24 trillion unique web pages
  - Every **second**, approximately
    - > 2 million emails
    - > 40000 Google search queries
    - > 6000 Tweets
- **Introduced the phrase "Internet Scale"**

Why study computer networks?

#3: To learn how to design for diversity!

- **Communication latency:** Microseconds to seconds
- **Bandwidth:** 1Kilobits/second to 100Gigabits/second
- **Packet Loss:** 0-90%
- **Technology:** Wireless, satellite, optical, copper, …
- **End hosts:** Sensors, cell phones, computers, servers, datacenters, …
- **Applications:** **www,** voice, video, gaming, remote medicine
- **Trust models:** selfish (users), malicious (attackers), greedy (companies), …

And yet, everything needs to work in tandem!

Recap: Packet switching summary

Packet switching is a method of transferring the data to a network in form of packets. In order to transfer the file fast and efficiently manner over the network and minimize the transmission latency, the data is broken into small pieces of variable length, called Packet. At the destination, all these small parts (packets) have to be reassembled, belonging to the same file. A packet composes of payload and various control information. No pre-setup or reservation of resources is needed.

- **Goods:**
  - Easier to handle failures
  - No resource underutilization
    - A source can send more if others don't use resources
  - No blocked connection problem
  - No per-connection state
  - No set-up cost
- **Not-so-goods:**
  - Unpredictable performance
  - High latency
  - Packet header overhead

Packet Switching doesn't give packets in order, whereas Circuit Switching provides ordered delivery of packets because all the packets follow the same path.
Since the packets are unordered, we need to provide sequence numbers for each packet.
Complexity is more at each node because of the facility to follow multiple paths.
Transmission delay is more because of rerouting.
Packet Switching is beneficial only for small messages, but for bursty data (large messages) Circuit Switching is better

Recap: Deep dive into one link: packet delay/latency

- **Consists of six components**
  - Link properties:
    - Transmission delay
    - Propagation delay
  - OS internals:
    - Processing delay
    - Queueing delay
  - Traffic matrix and switch internals:
    - Processing delay
    - Queueing delay
- First, consider transmission, propagation delays
- Queueing delay and processing delays later in the course

IT 304 Computer Networks
Introduction
Week 2-Lecture 2

Recap: Week 2-Lec 1

- Internet
  - Big picture View
  - Service view
- Components of a network
  - Edge, access and core
- Performance metrics

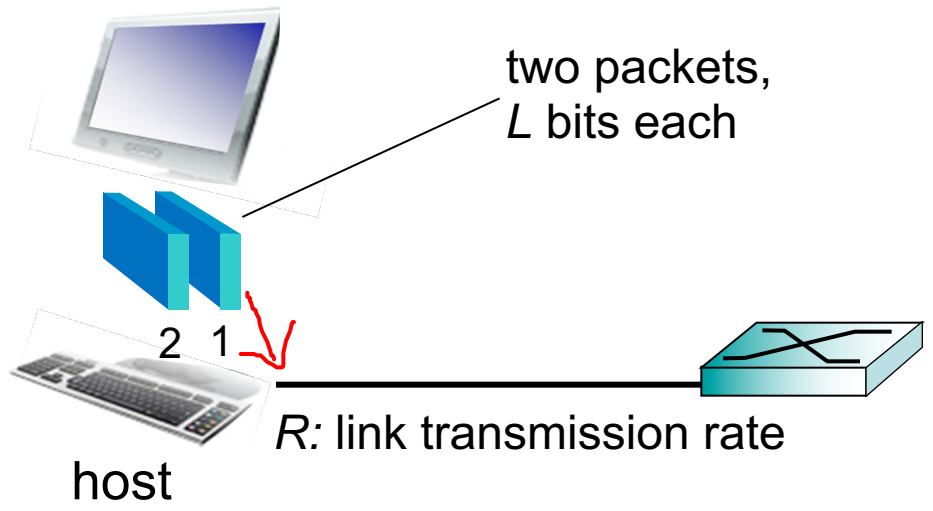
Today's lecture

- Cont.. performance metrics

Host: sends *packets* of data

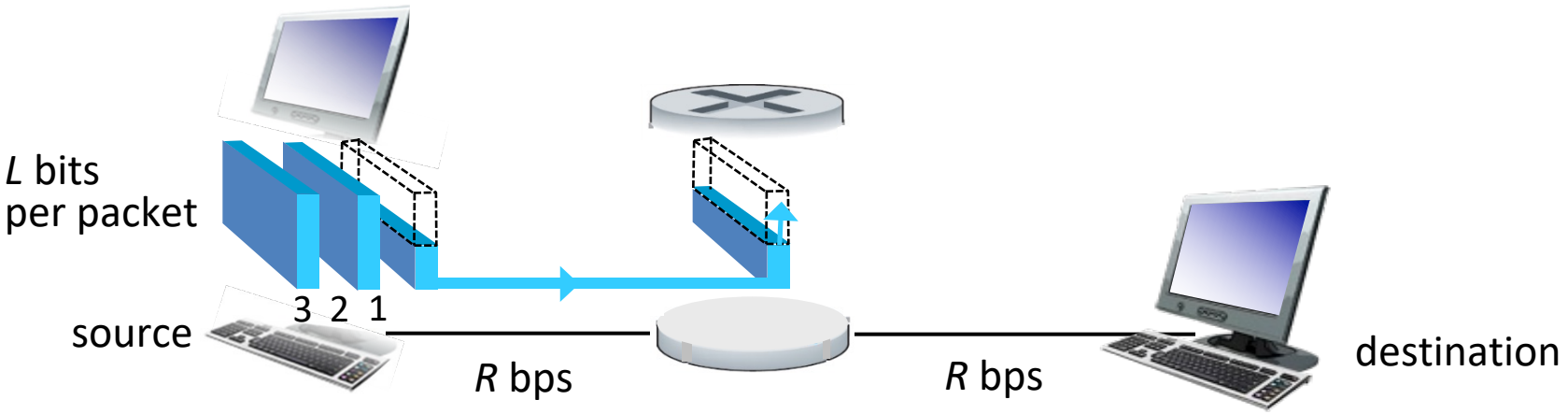
host sending function:

- 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*



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

Packet-switching: store-and-forward



- 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
- end-end delay = $2L/R$ (assuming zero propagation delay)

one-hop numerical example:

- L = 7.5 Mbits
- R = 1.5 Mbps
- one-hop transmission delay = 5 sec

more on delay shortly …

What are the main components of delay when we use packet switching?

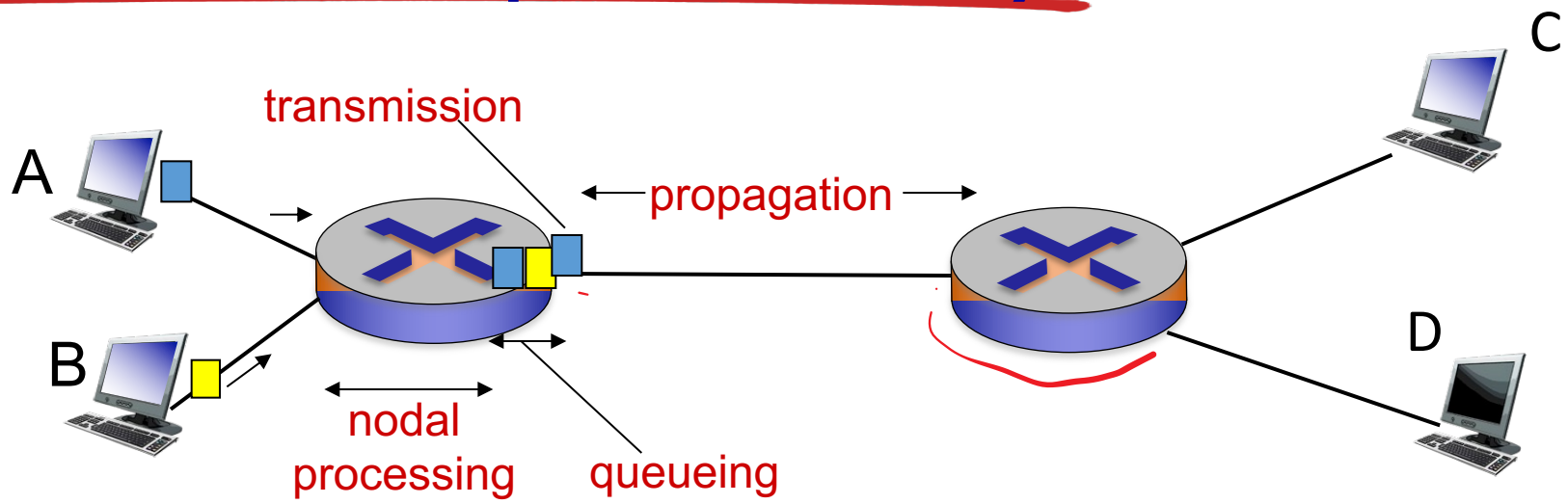
→ *Processing delay*

→ *queuing delay*

→ *transmission delay*

→ *propagation delay*

Four sources of packet delay



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

d_{trans}: transmission delay: func(L, R)

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

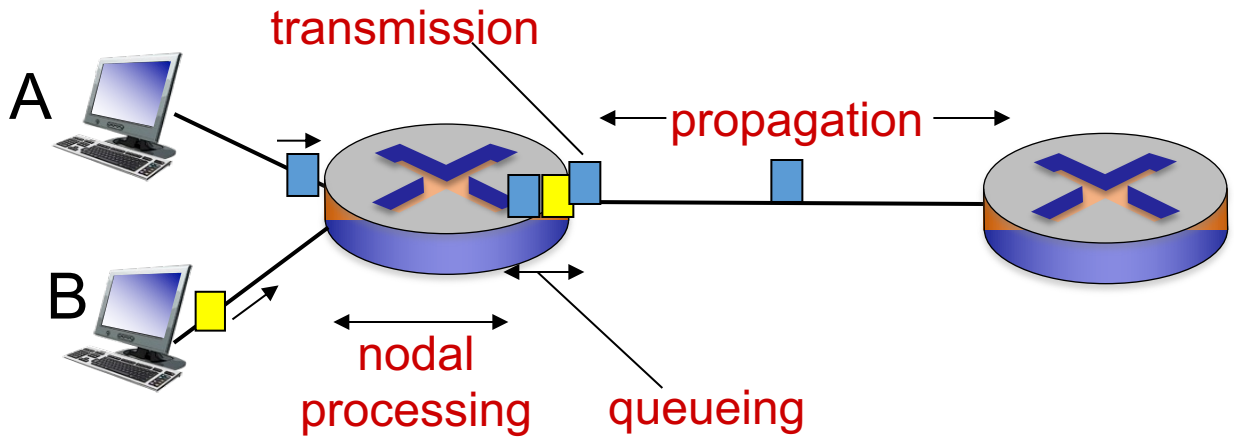
d_{prop}: propagation delay: func(d)

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

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

* Check out the online interactive exercises for more examples:

Four sources of packet delay



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

d_{proc}: nodal processing

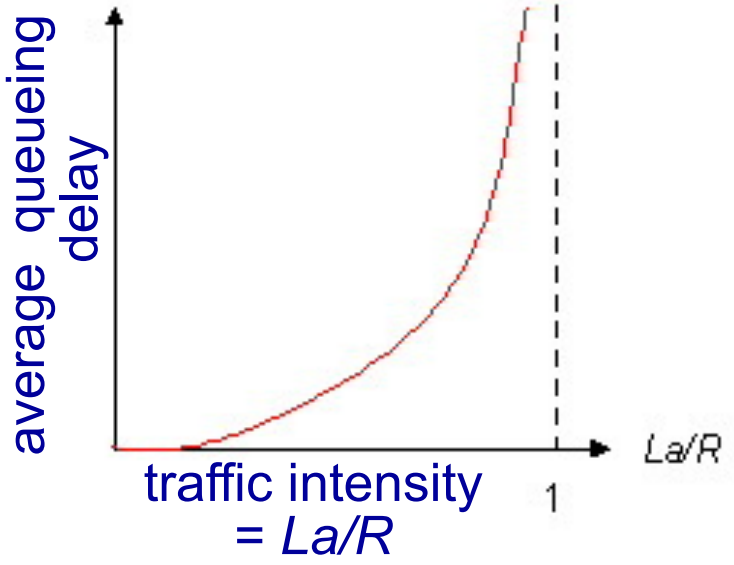
- check bit errors
- determine output link
- typically < msec

d_{queue}: queueing delay

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

Queueing delay

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



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

La/R ~ 0

La/R -> 1

* Check online interactive animation on queuing and loss

Delay factor contributions to Total Delay

- d_{prop}
  - for a link connecting two routers on the same university campus → negligible
  - for two routers interconnected by a satellite link(100s km) → 100+ milliseconds
- d_{trans}
  - > bandwidths (10 Mbps and higher) → negligible [more bits/sec]
  - If large Internet packets sent over low-speed dial-up modem links → hundreds of milliseconds
    .
- d_{proc} is often negligible
  - Super fast routers

Propagation delay is defined as the time taken for bits in a packet to go over a transmission link.

A. True
B. False

Propagation delay is a function of

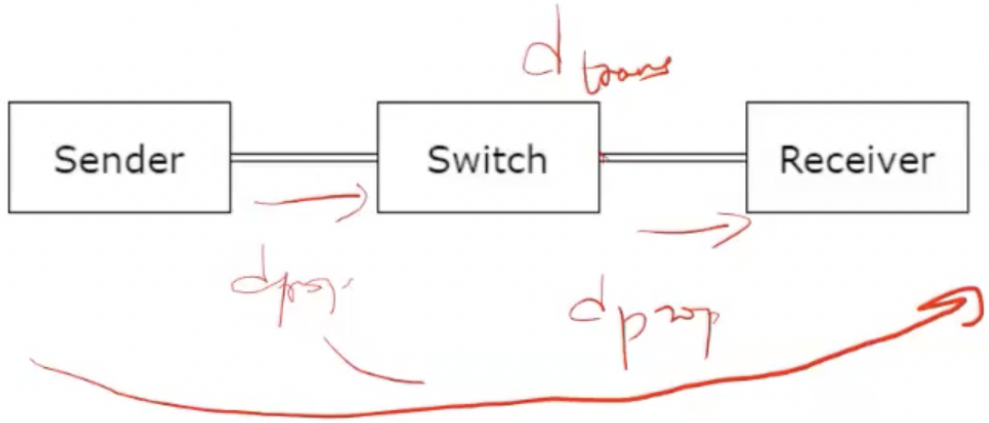
A. Distance
B. Bandwidth
C. Packet size
D. Speed of light in a medium

Tranmission delay is a function of

A. Distance
B. Speed of light in a medium
C. Bandwidth
D. Packet size

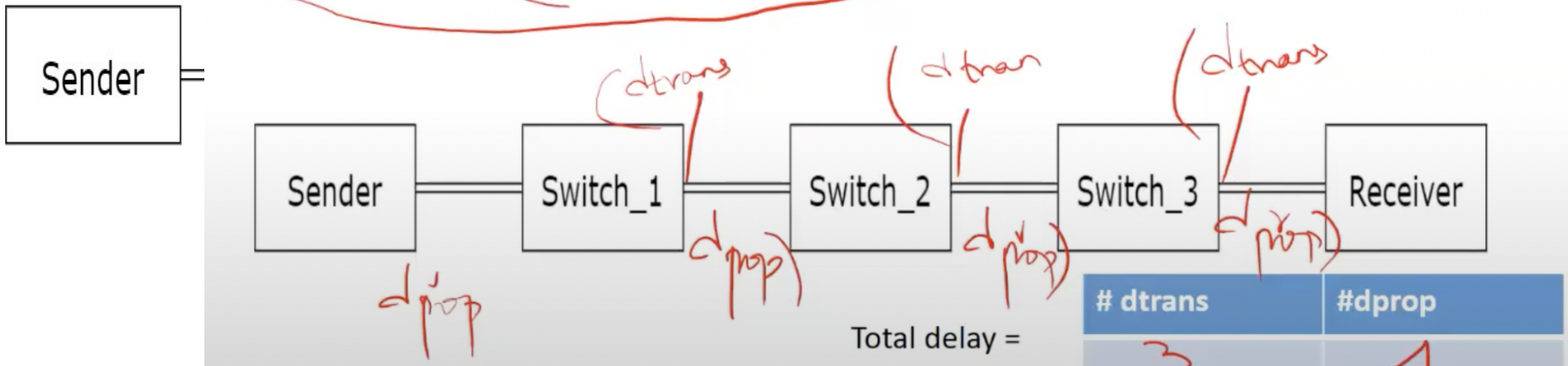
Total delay =

# dtrans	#dprop



Total delay =

# dtrans	#dprop
1	2

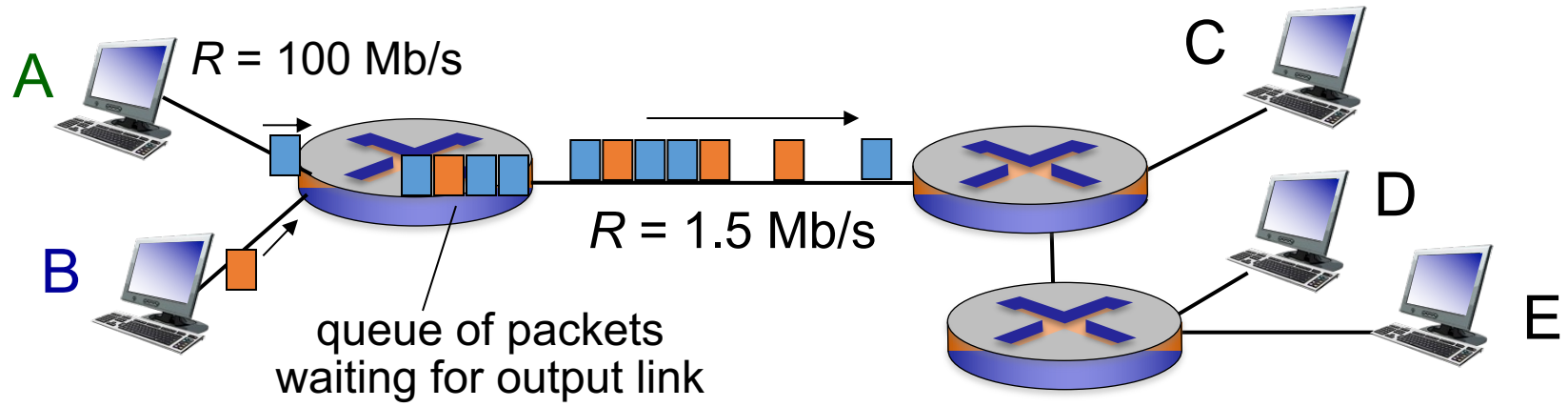


Total delay =

# dtrans	#dprop
3	4

$$d_{End\text{-}End} = N(d_{proc} + d_{queue} + d_{trans} + d_{prop})$$

Packet Switching: queueing delay, loss



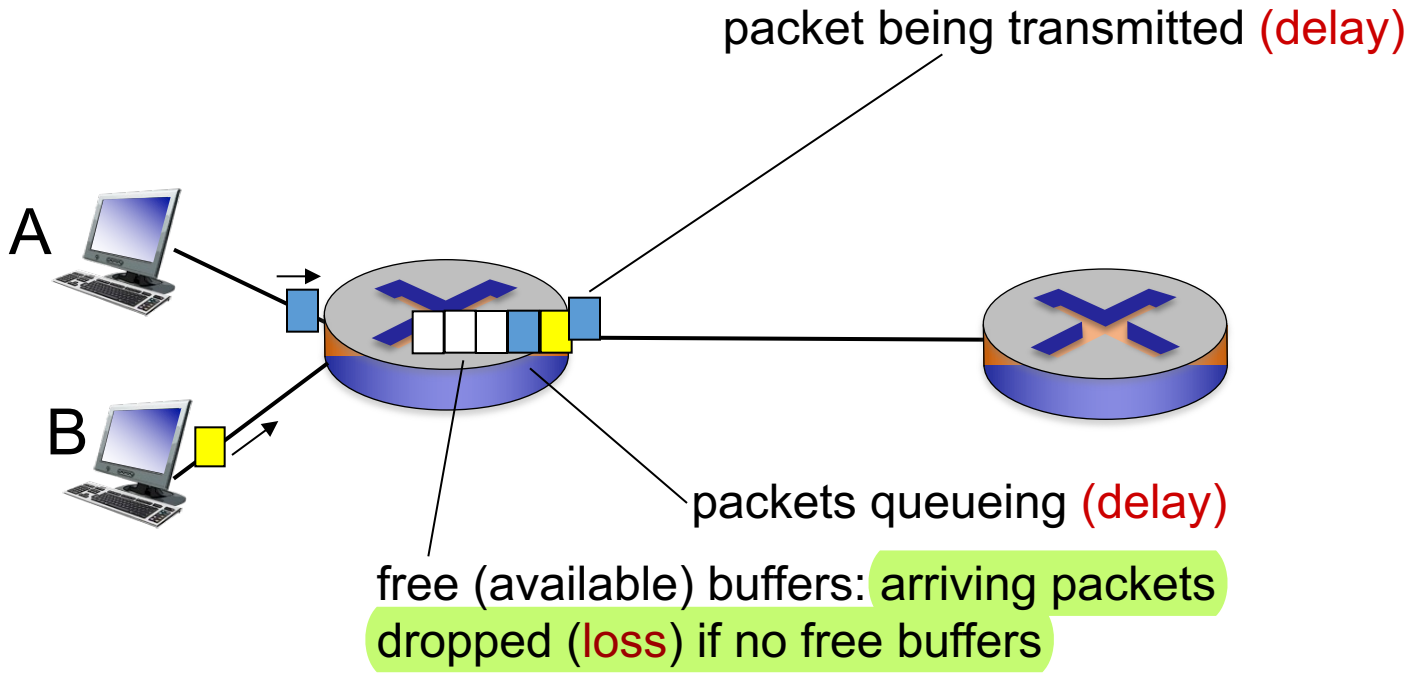
queuing and loss:

- if arrival rate (in bits) to link exceeds transmission rate of link for a period of time:
  - packets will queue, wait to be transmitted on link
  - packets can be dropped (lost) if memory (buffer) fills up

How do loss and delay occur?

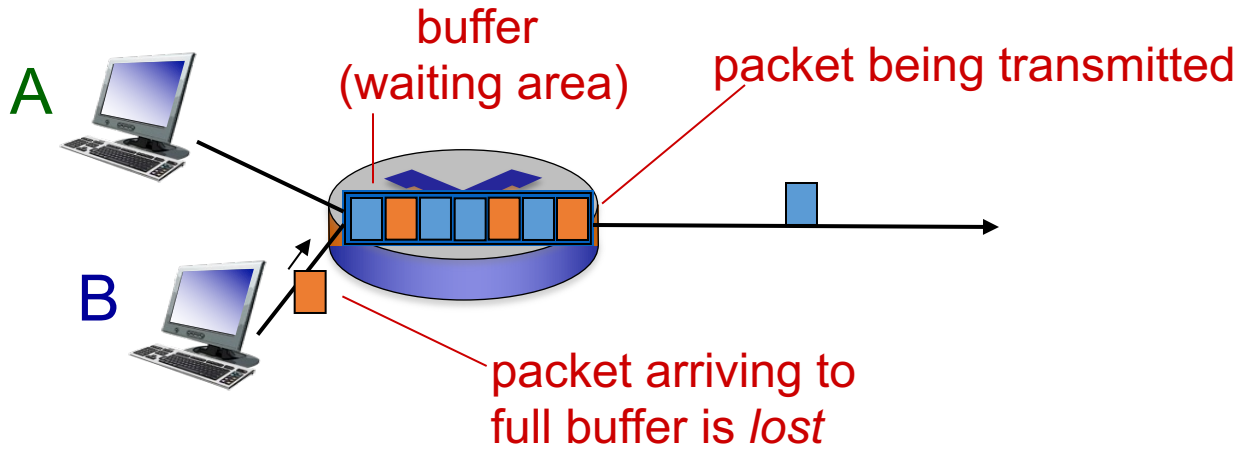
packets *queue* in router buffers

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



Packet loss

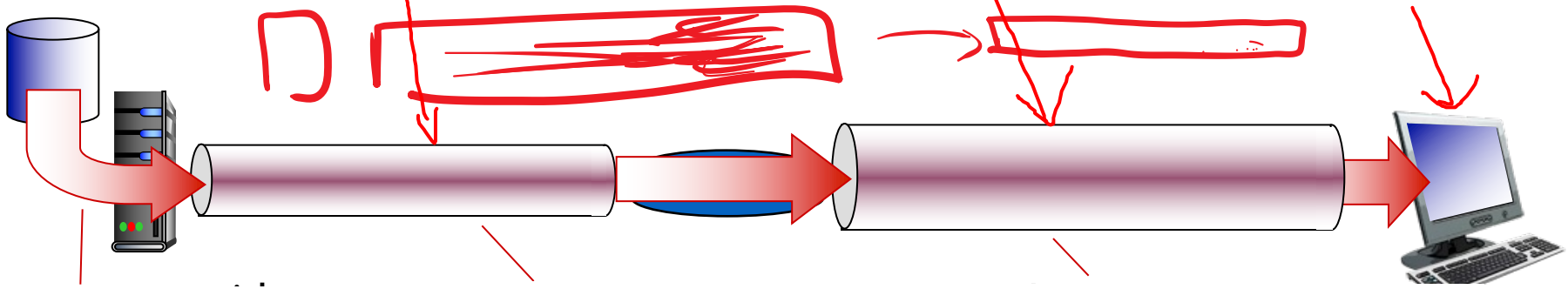
- queue (aka buffer) preceding link in buffer has finite capacity
- packet arriving to full queue dropped (aka lost)
- lost packet may be retransmitted by previous node, by source end system, or not at all



* Check out the Java applet for an interactive animation on queuing and loss

Throughput

- *throughput:* rate (bits/time unit) at which bits transferred between sender/receiver
  - *instantaneous:* rate at given point in time
  - *average:* rate over longer period of time



If four routers are separated by 10km, but with different capacity optical fibre cables, which delay component will be constant and which one will vary?

A. d_{prop} constant, d_{trans} varying

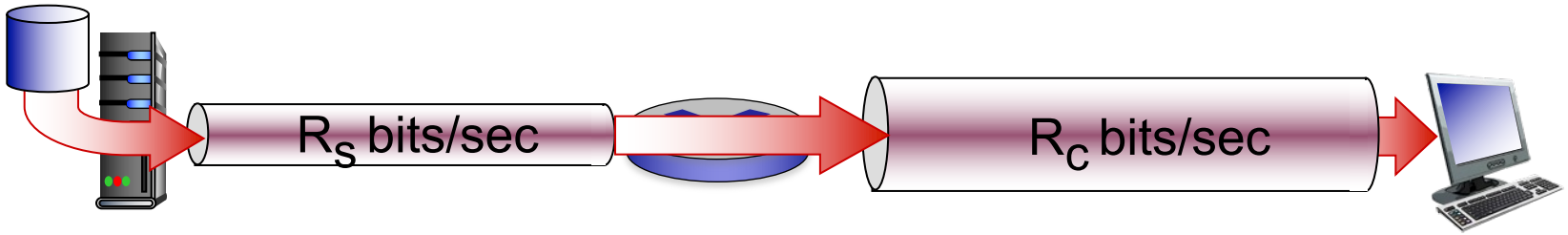
B. d_{prop} varying, d_{trans} varying

C. d_{prop} varying, d_{trans} constant

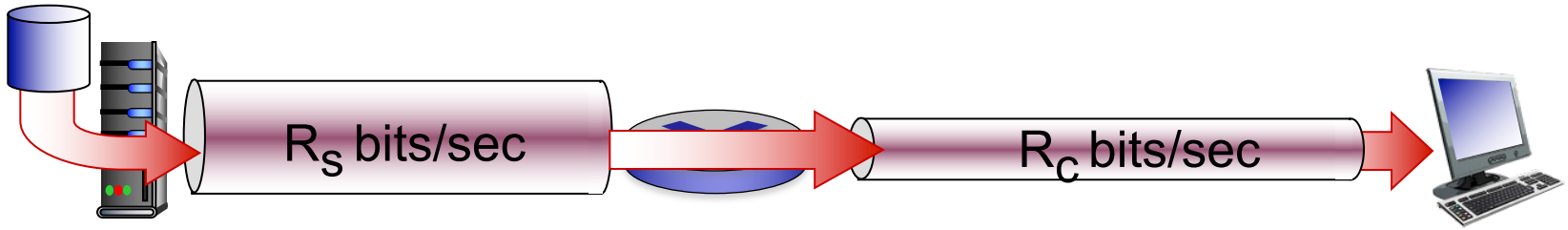
D. d_{prop} constant, d_{trans} constant

Throughput (more)

- $R_s < R_c$ What is average end-end throughput?



- $R_s > R_c$ What is average end-end throughput?

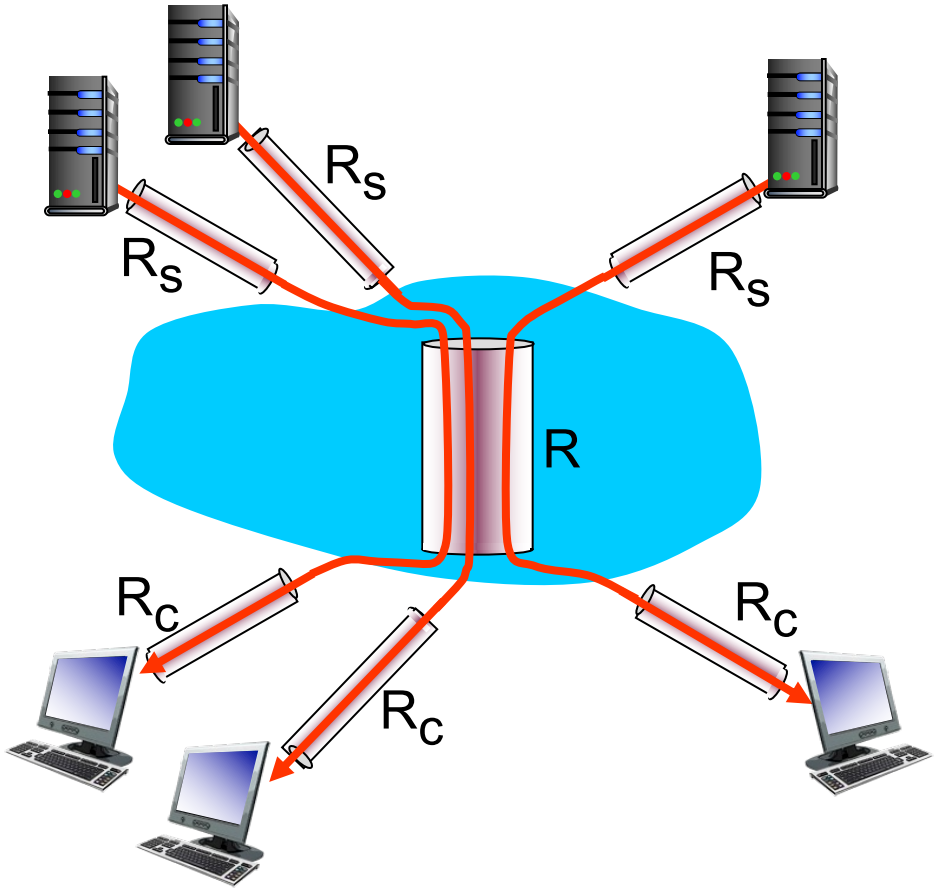


bottleneck link

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

Throughput: Internet scenario

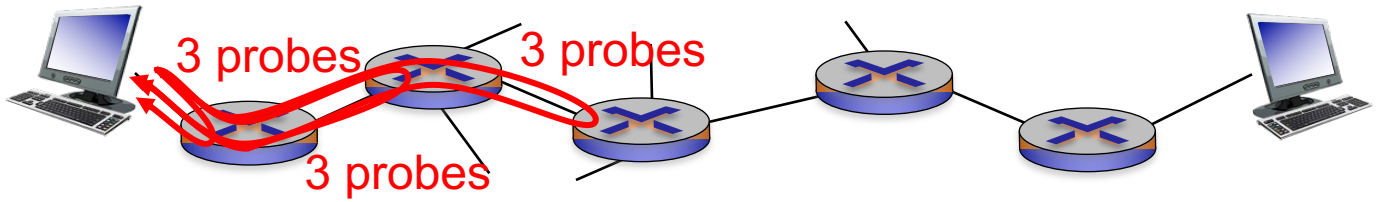
- per-connection end-end throughput: *min(R_c,R_s,R/10)*
- in practice: R_c or R_s is often bottleneck



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

"Real" Internet delays and routes

- what do "real" Internet delay & loss look like?
- **`traceroute`** program: provides delay measurement from source to router along end-end Internet path towards destination. For all *i:*
  - sends three packets that will reach router *i* on path towards destination
  - router *i* will return packets to sender
  - sender times interval between transmission and reply.



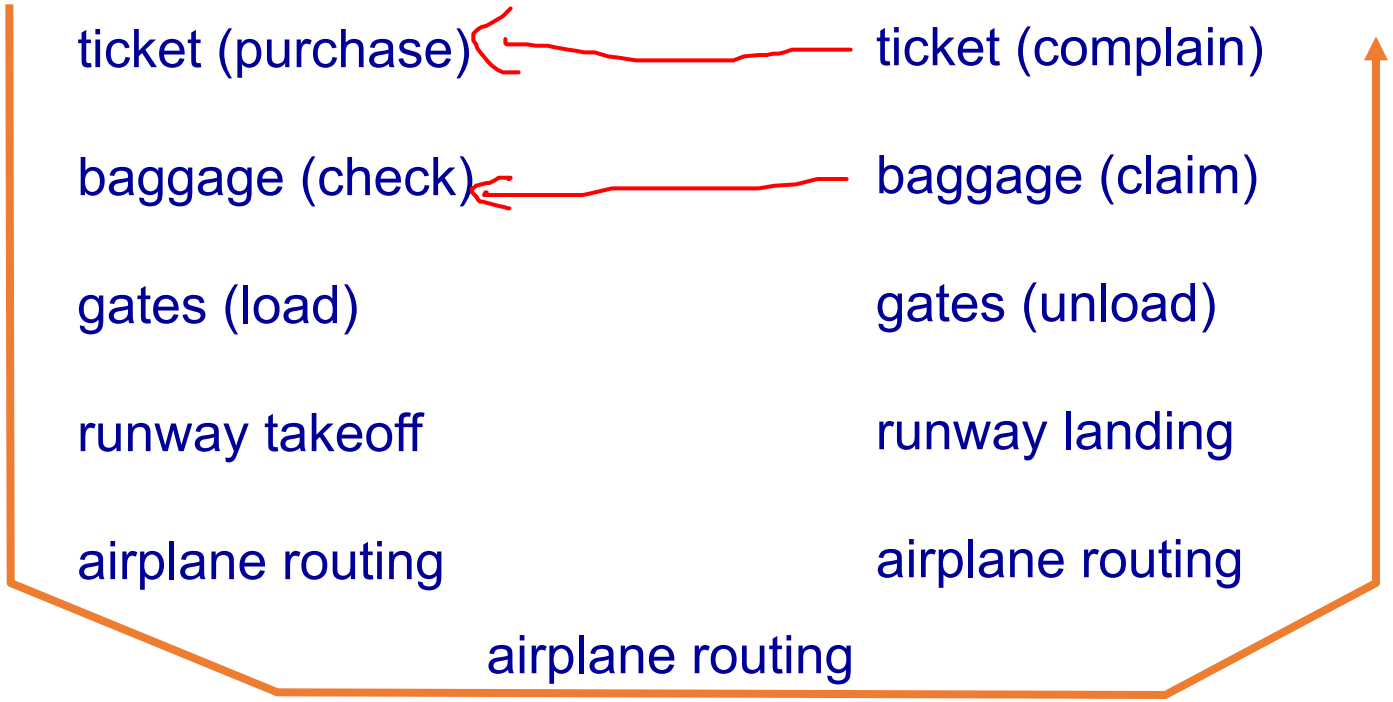
IT 304 Computer Networks
Introduction
Week 2-Lecture 3

Protocol "layers"

Networks are complex, with many "pieces":

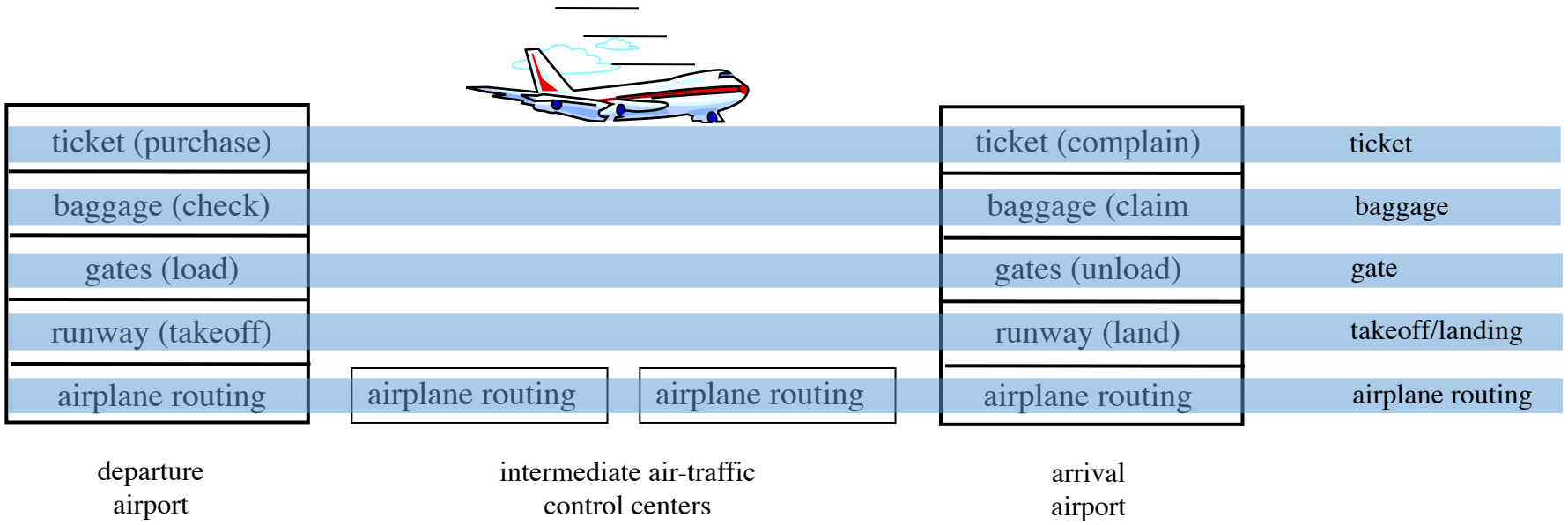
- hosts
- routers
- links of various media
- applications
- protocols
- hardware, software

Organization of air travel



- a series of steps

Layering of airline functionality



layers: each layer implements a service

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

Question:

How is the Internet structured?

Why layering?

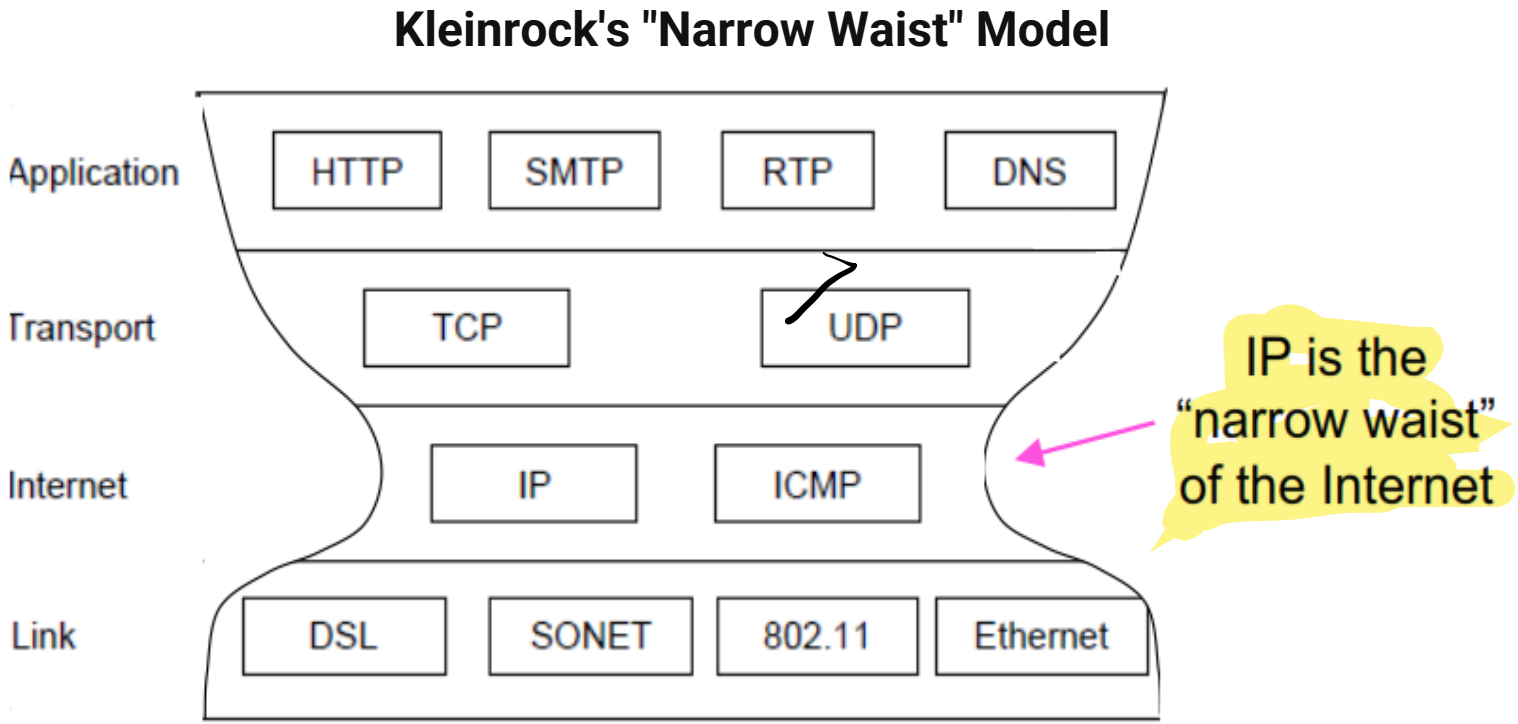
dealing with complex systems:

- explicit structure allows identification, relationship of complex system's pieces
  - layered *reference model* for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system

The TCP/IP Model

Vint Cerf and **Bob Kahn**

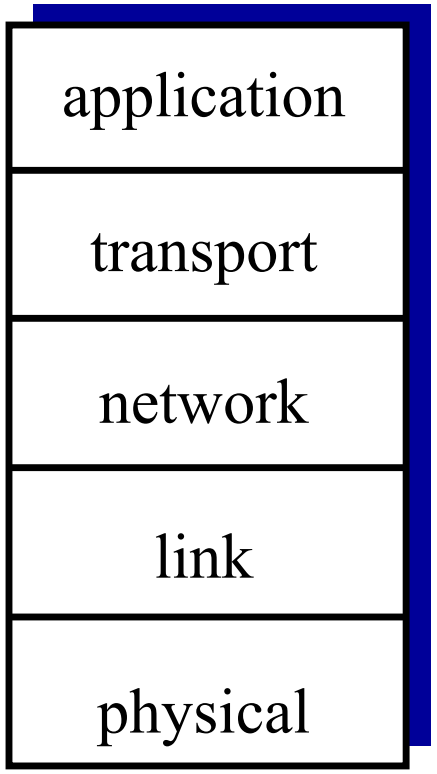
- Link layer
- Internet layer
- Transport layer
- Application layer



LITA

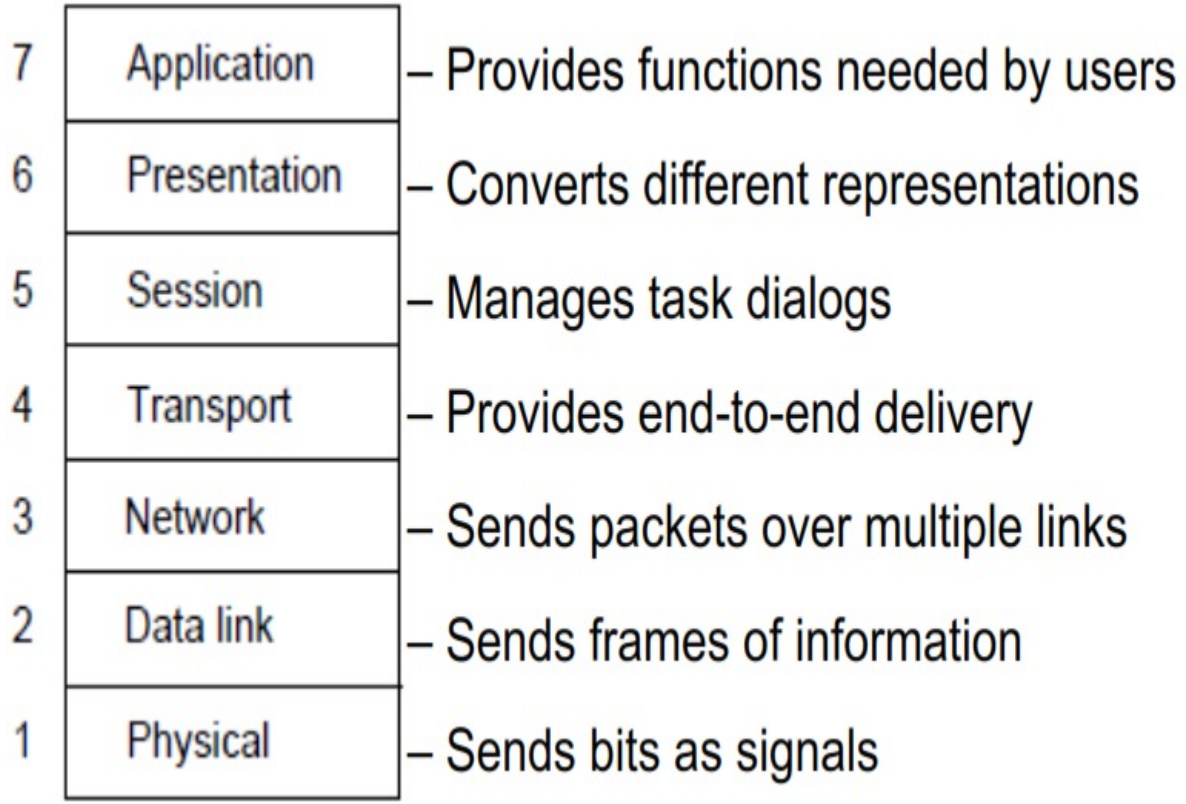
Internet protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet (802.3), WiFi (802.11), PPP
- *physical:* bits "on the wire"



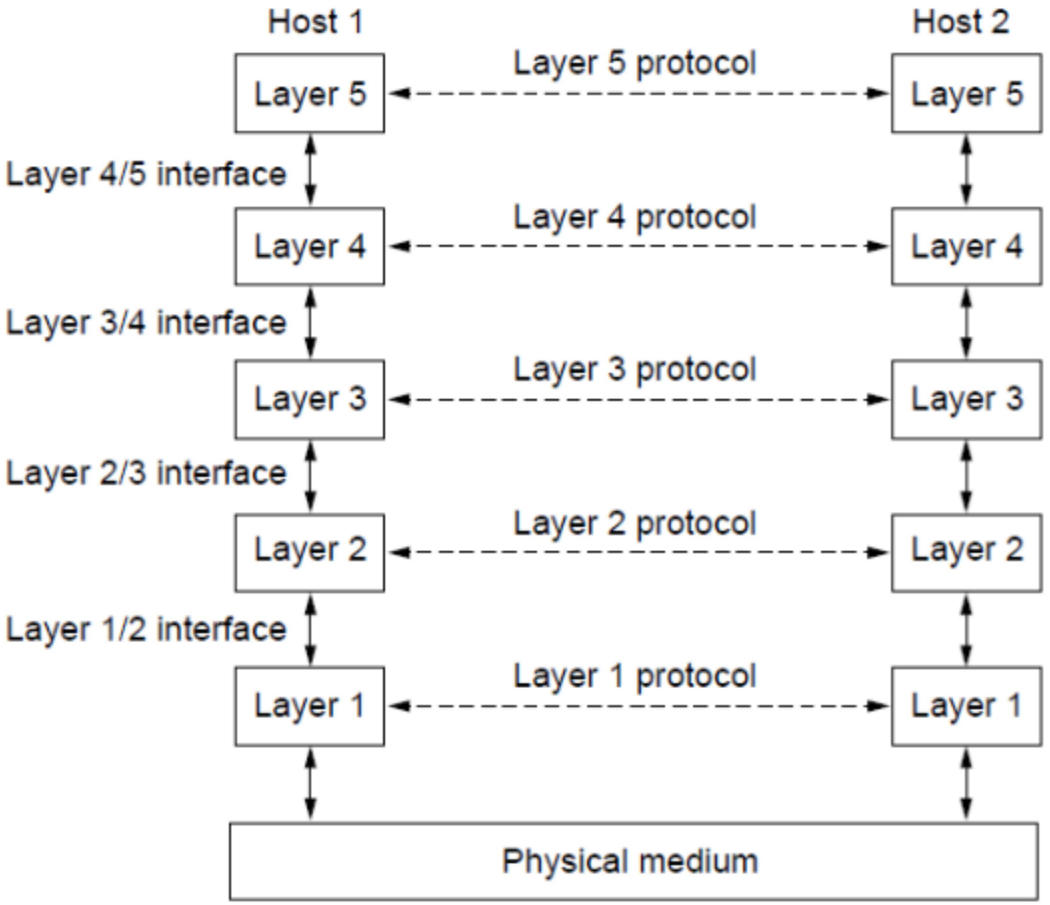
The Open Systems Interconnection (OSI) Reference Model by International Standards Organization (ISO)

- Layers created for different abstractions
- Each layer performs well-defined function
- Function of layer chosen with standards in mind
- Minimize information flow across layer interfaces
- Find the optimum number of layers



Protocol Layers (1)

Protocol layering is the main structuring method used to divide up network functionality.

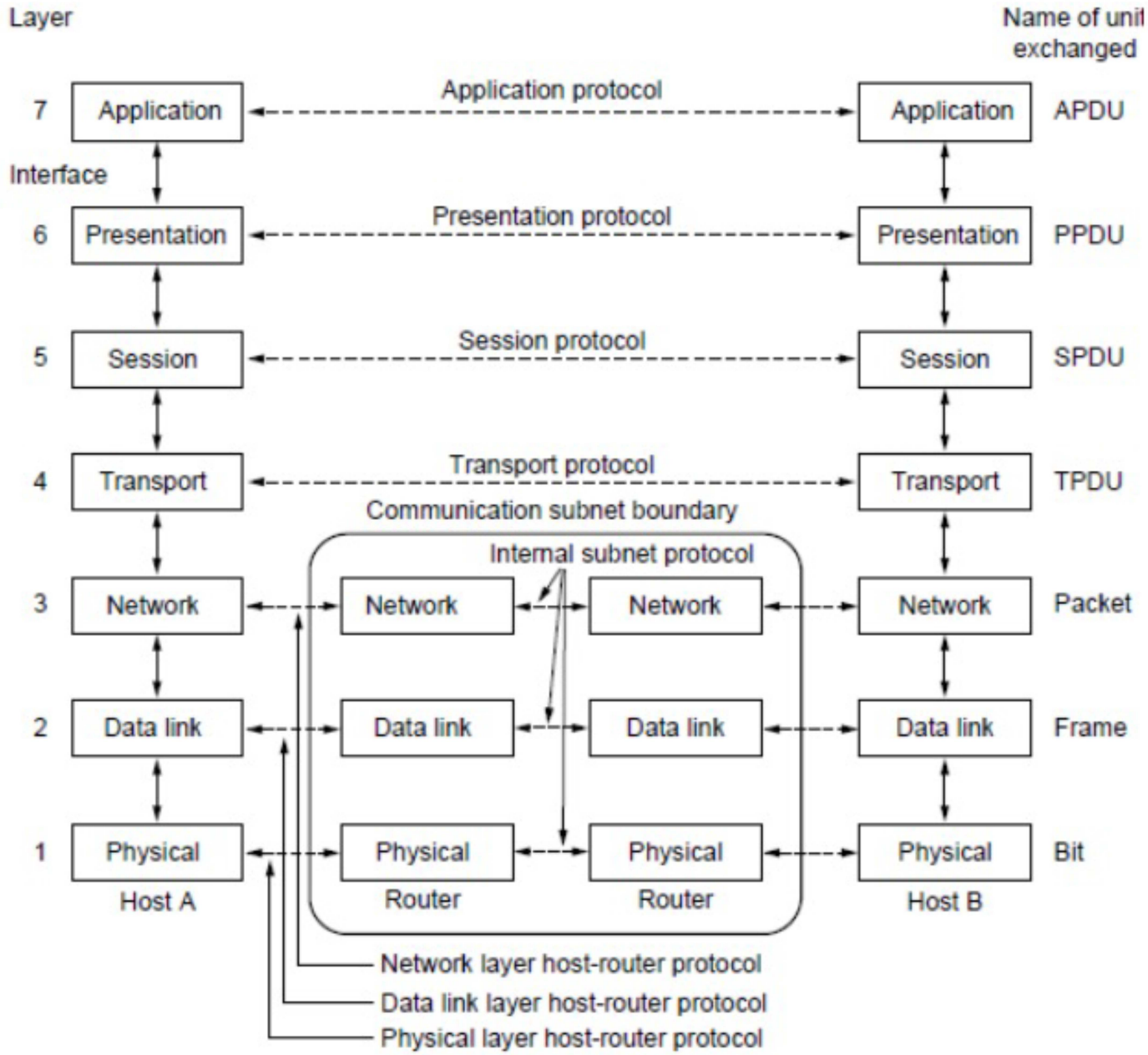


- Each protocol instance talks virtually to its peer
- Each layer communicates only by using the one below
- Lower layer services are accessed by an interface
- At bottom, messages are carried by the medium

Design Issues for the Layers

Each layer solves a particular problem but must include mechanisms to address a set of recurring design issues

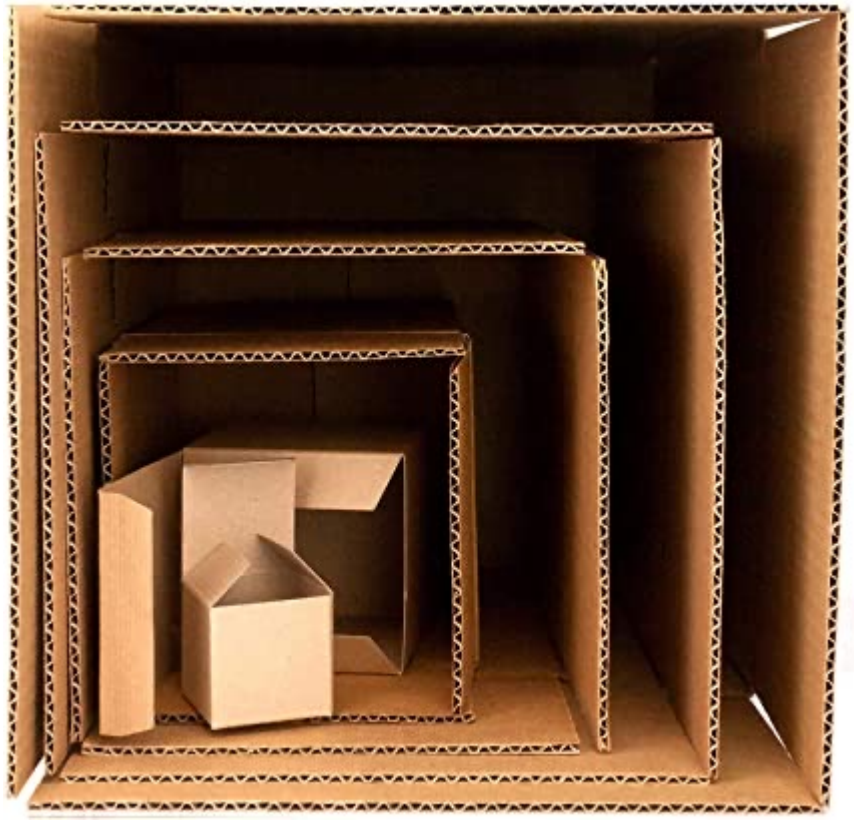
Issue	**Example mechanisms at different layers**
Reliability despite failures	Codes for error detection/correction Routing around failures
Network growth and evolution	Addressing and naming Protocol layering
Allocation of resources like bandwidth	Multiple access Congestion control
Security against various threats	Confidentiality of messages Authentication of communicating parties



PDU is Protocol Data Unit

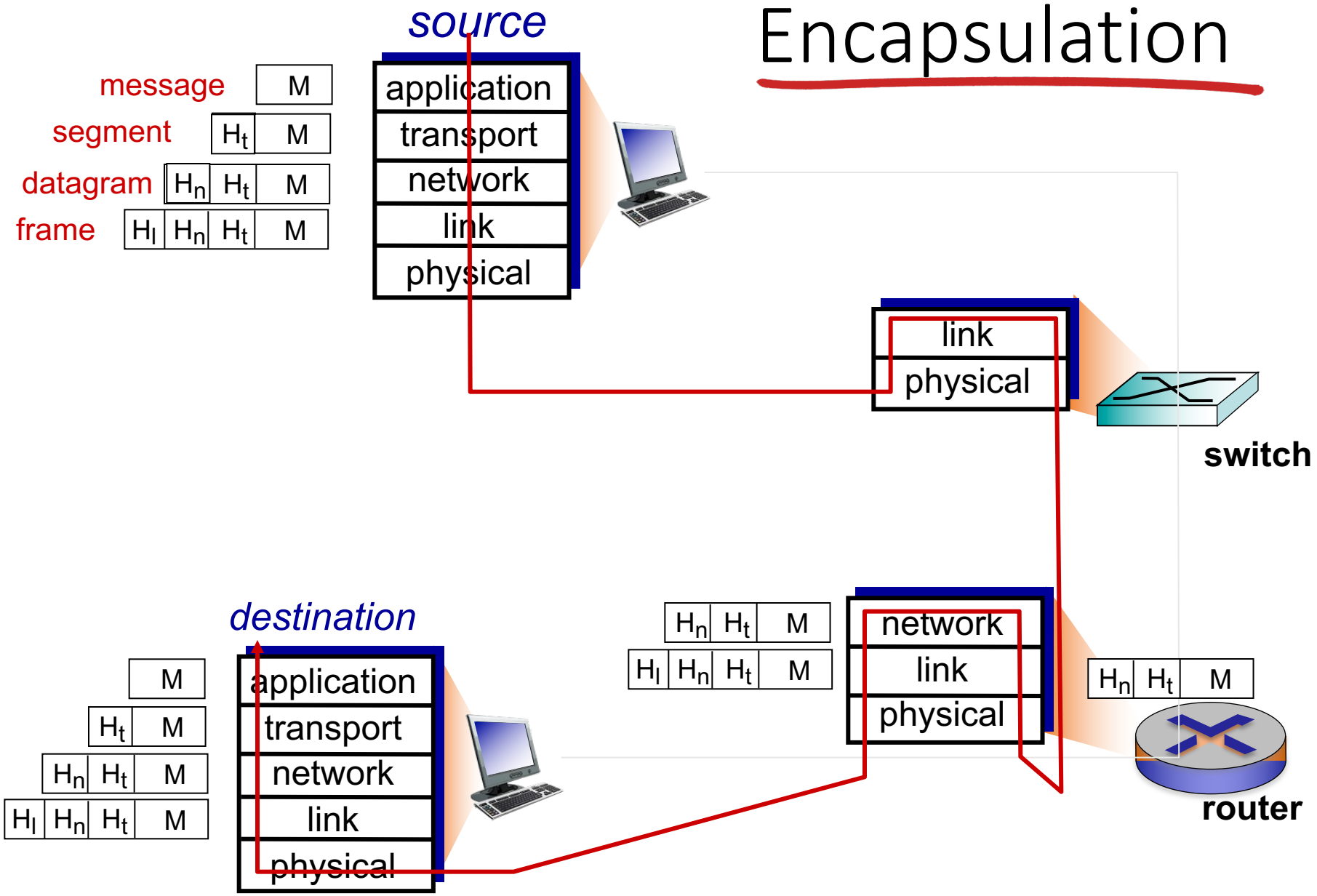
Encapsulation

- Who is the source (S) computer?
- Who is the destination (D) computer?
- Who is passing this message to whom?



A gift box example for encapsulating / packing message from S to D

Encapsulation

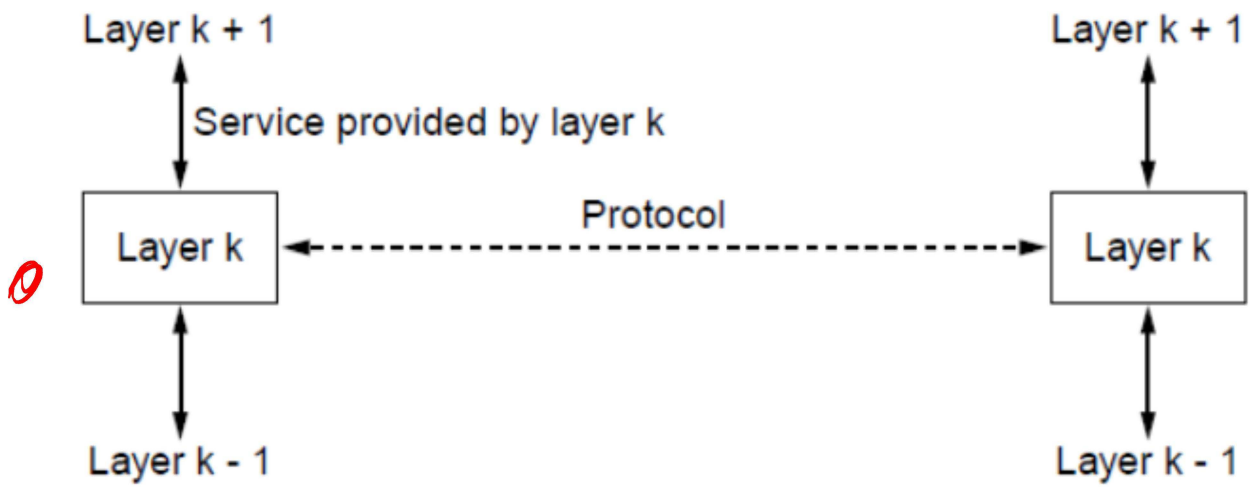


Relationship of Services to Protocols

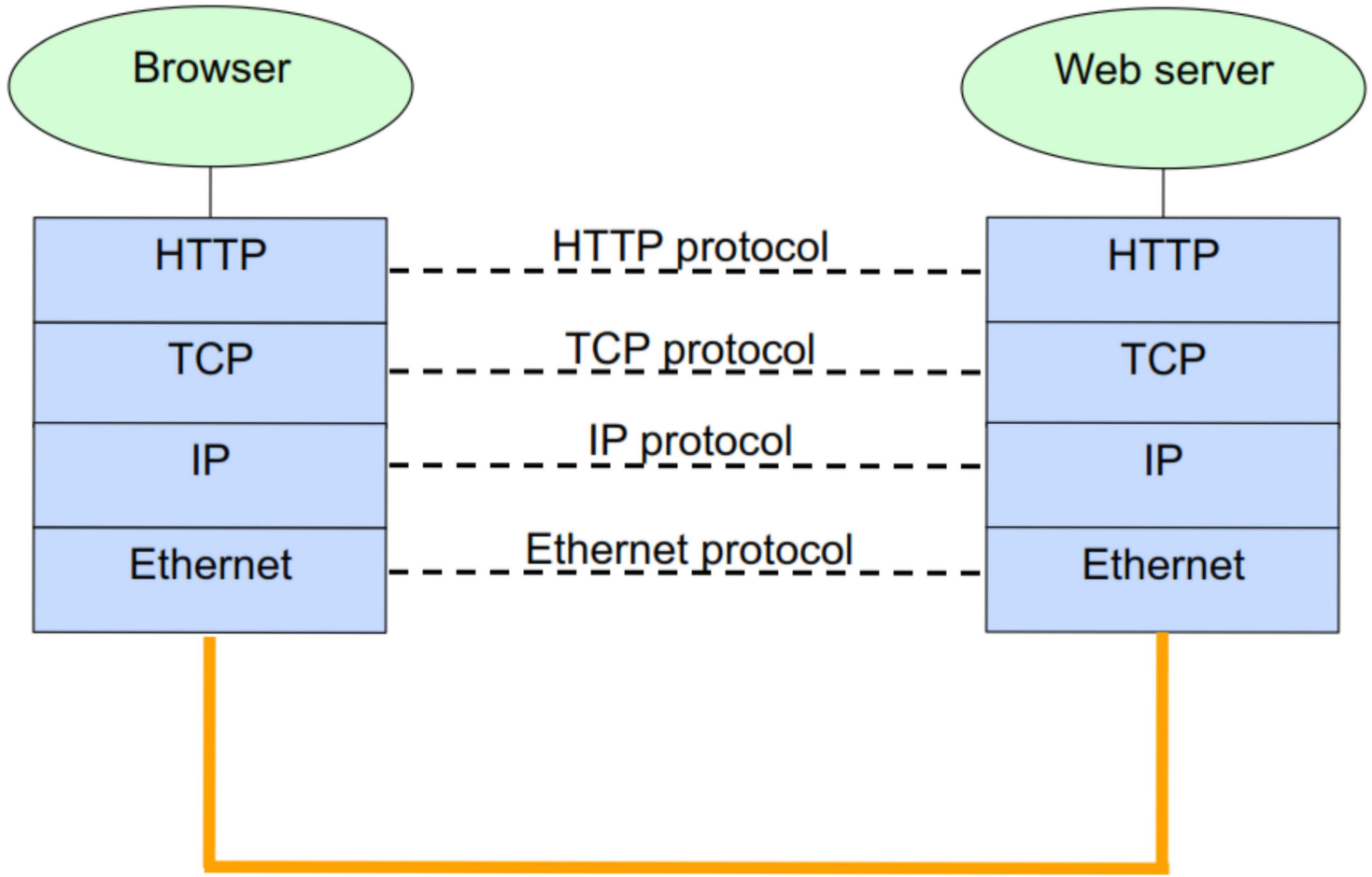
Recap:

- A layer provides a service to the one above (vertical)
- A layer talks to its peer using a protocol (horizontal)

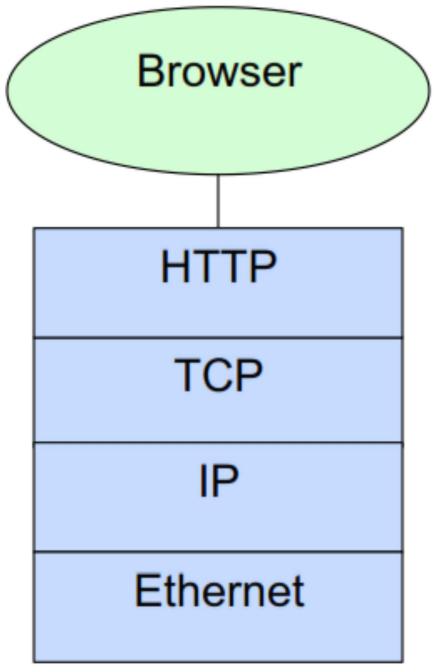
LISTEN
CONNECT
ACCEPT

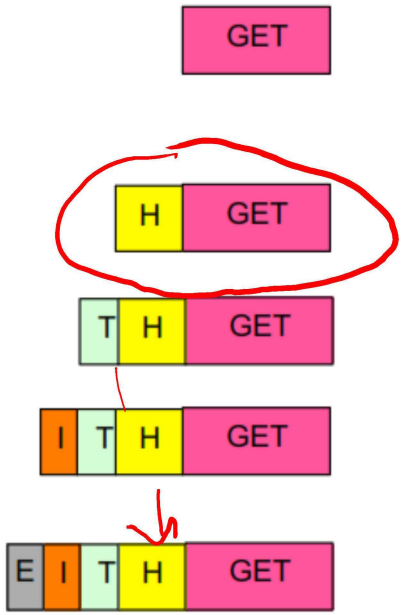


Example Protocol Stack



Encapsulation





IT 304 Computer Networks
Introduction
Week 4-Lecture 1

Connection-Oriented vs. Connectionless

Service provided by a layer may be kinds of either:

- Connection-oriented, must be set up for ongoing use (and torn down after use), e.g., phone call
- Connectionless, messages are handled separately, e.g., postal delivery

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Movie download
	Unreliable connection	Voice over IP
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Text messaging
	Request-reply	Database query

Service Primitives (1)

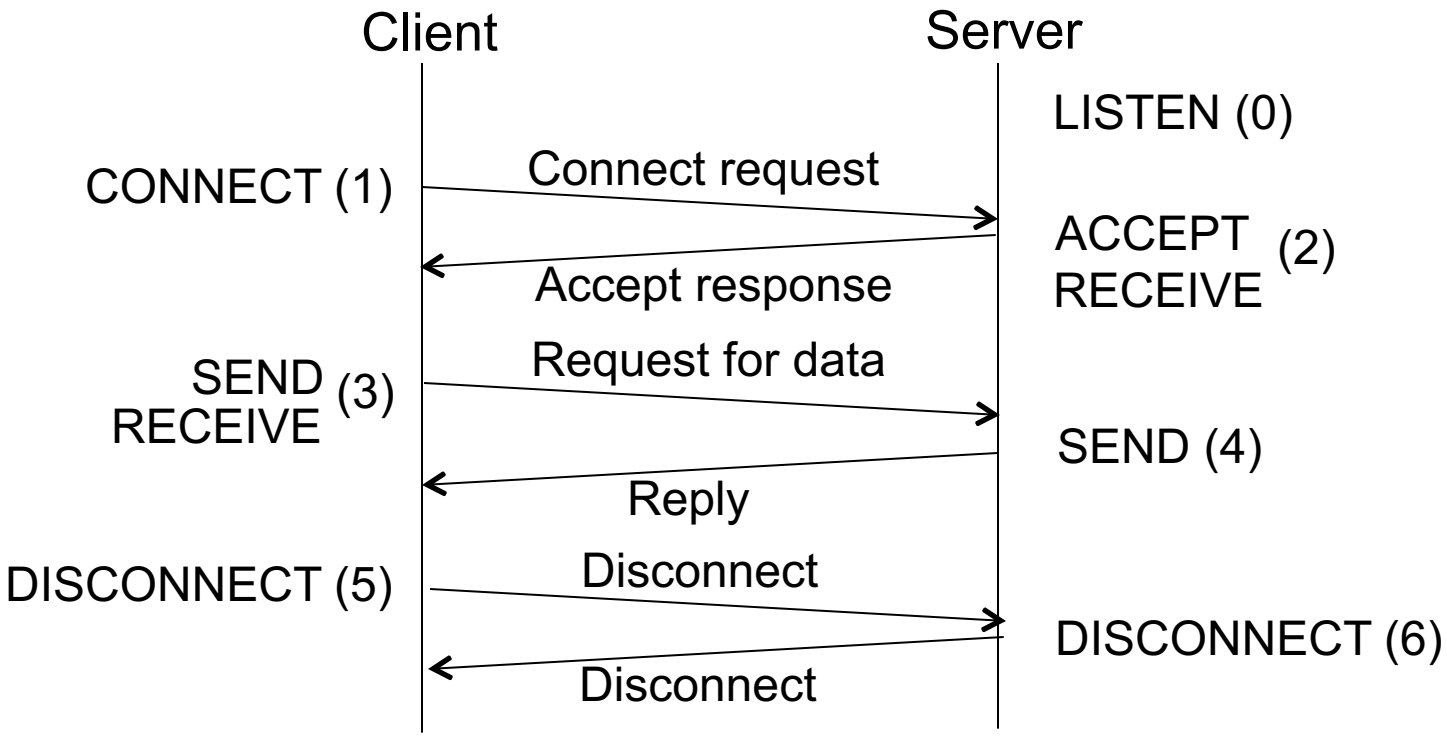
A service is provided to the layer above as primitives

Hypothetical example of service primitives that may provide a reliable byte stream (connection-oriented) service:

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
ACCEPT	Accept an incoming connection from a peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

Service Primitives (2)

Hypothetical example of how these primitives may be used for a client-server interaction



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.,

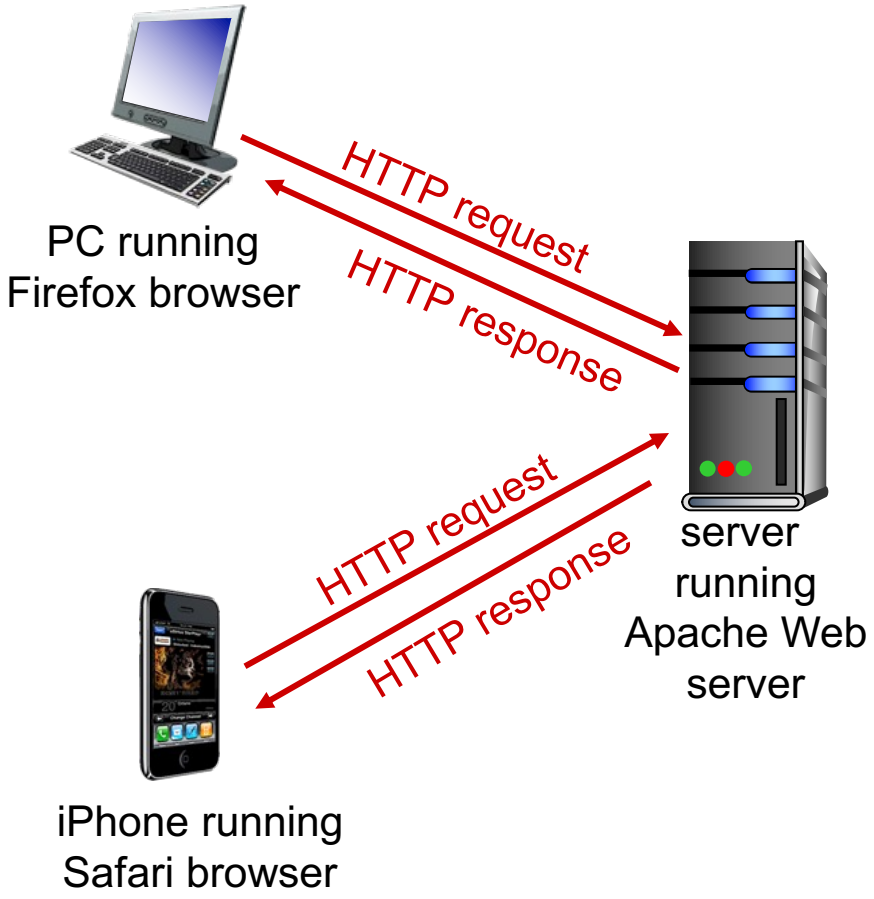
`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

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



HTTP overview (continued)

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

HTTP connections

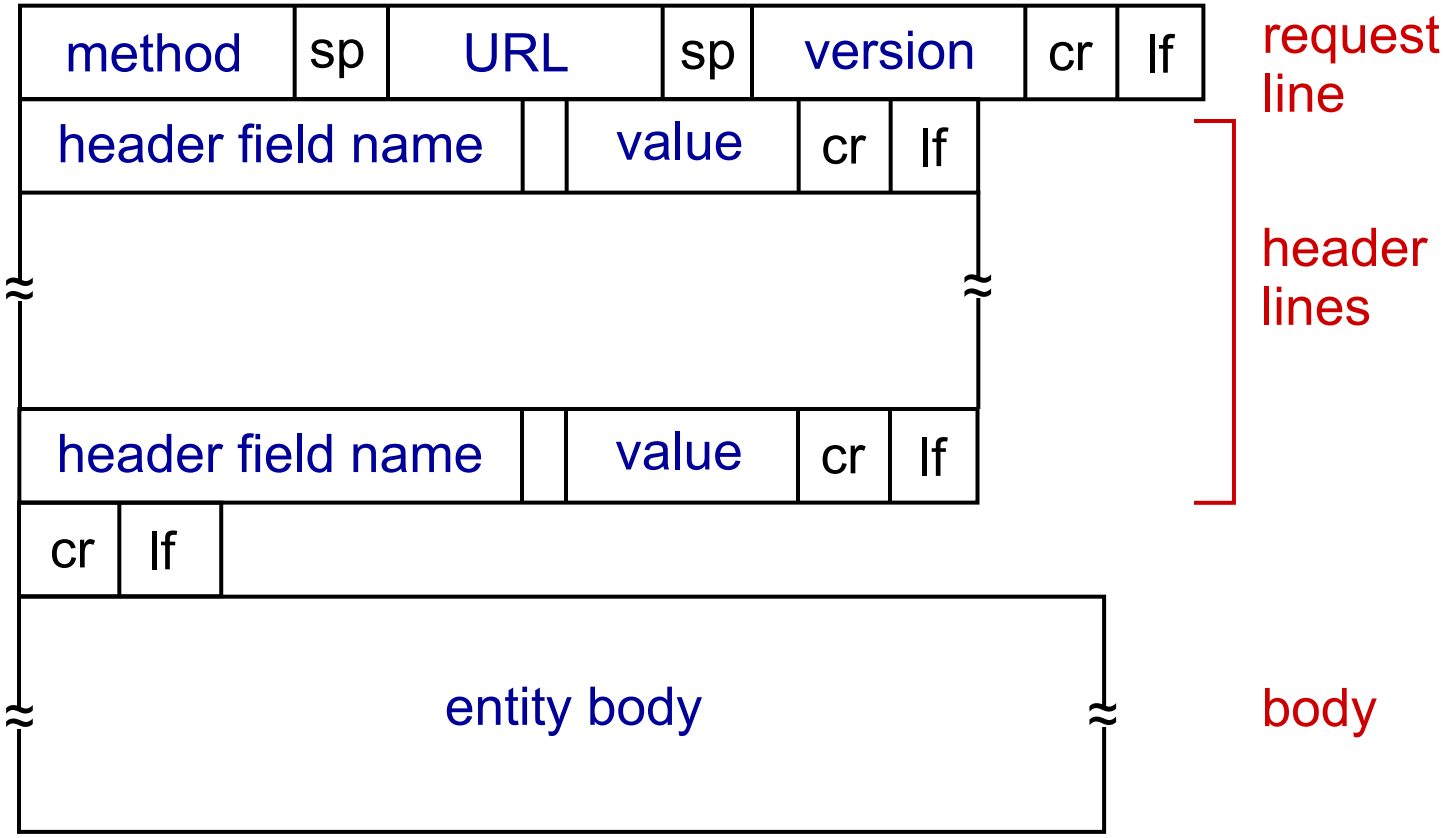
non-persistent HTTP

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

persistent HTTP

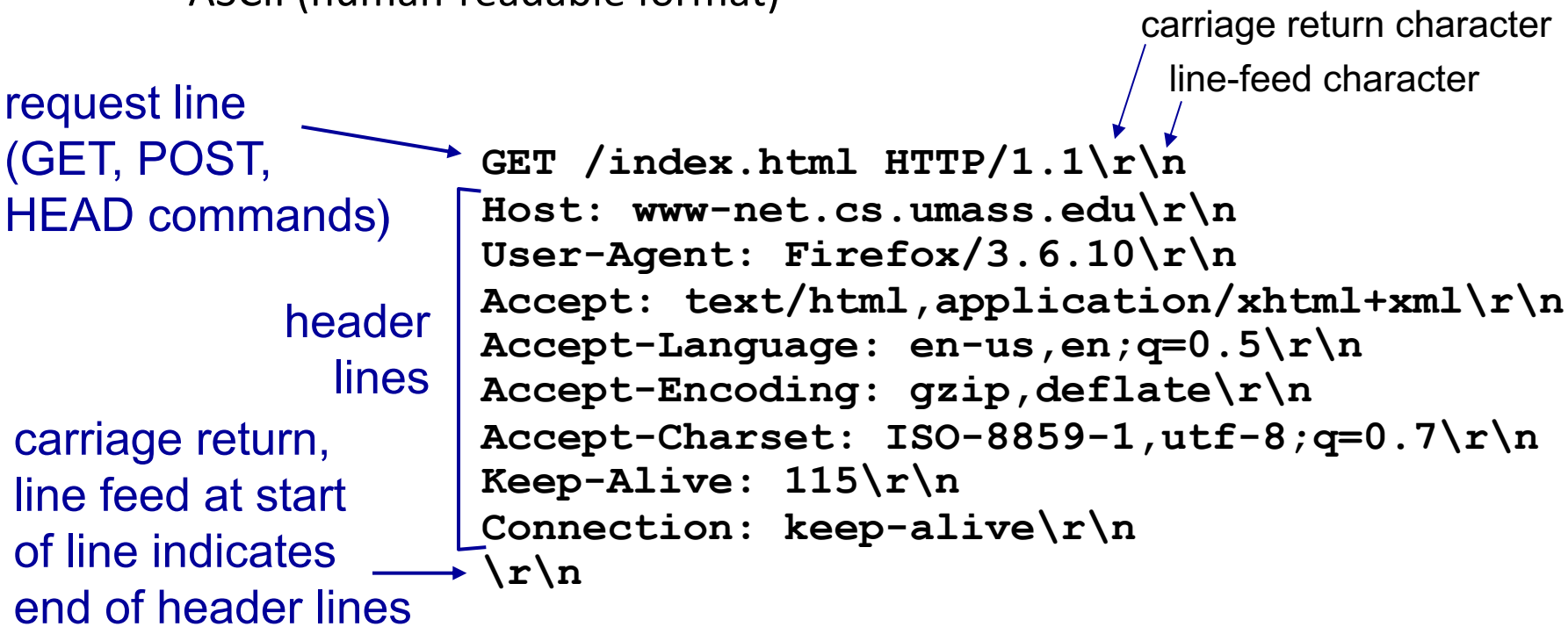
- multiple objects can be sent over single TCP connection between client, server

HTTP request message: general format



HTTP request message

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



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

Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.daiict.ac.in/faculty
```

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

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

GET /kurose_ross_sandbox/interactive/quotation6.htm
HTTP/1.1
Host: gaia.cs.umass.edu
Accept: text/plain, text/html, image/gif, image/jpeg, audio/mp4,
audio/vnf.wave, video/mp4, video/wmv,
Accept-Language: en-us, en-gb;q=0.2, en;q=0.3, fr, fr-ch, da,
de, fi, ar
If-Modified-Since: Tue, 23 Aug 2022 19:27:10 -0700
User Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/535.19 (KHTML, like Gecko)
Chrome/18.0.1025.168 Safari/535.19

What version of HTTP is the client running?

HTTP response status codes

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

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

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

400 Bad Request

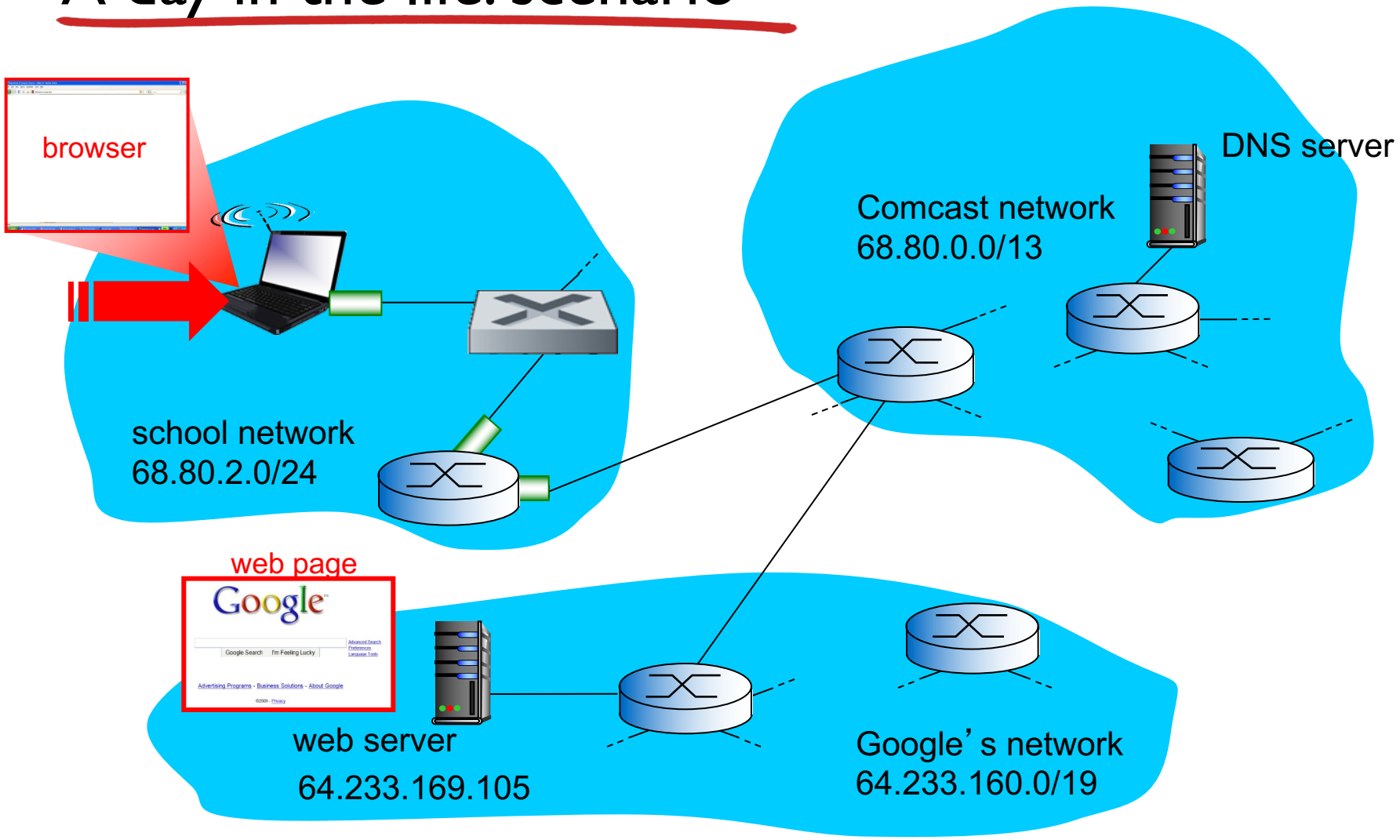
- request msg not understood by server

404 Not Found

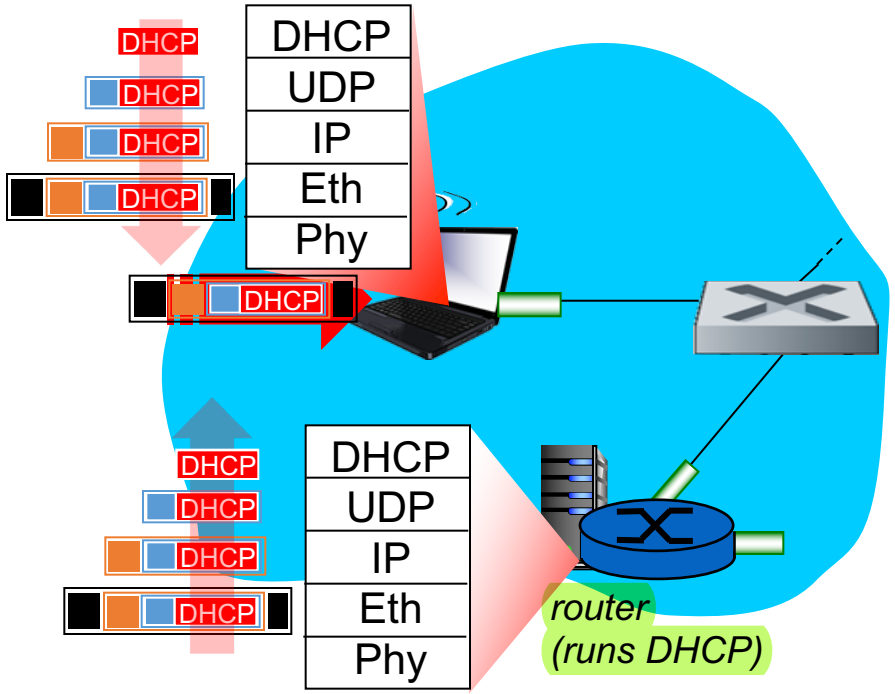
- requested document not found on this server

505 HTTP Version Not Supported

A day in the life: scenario



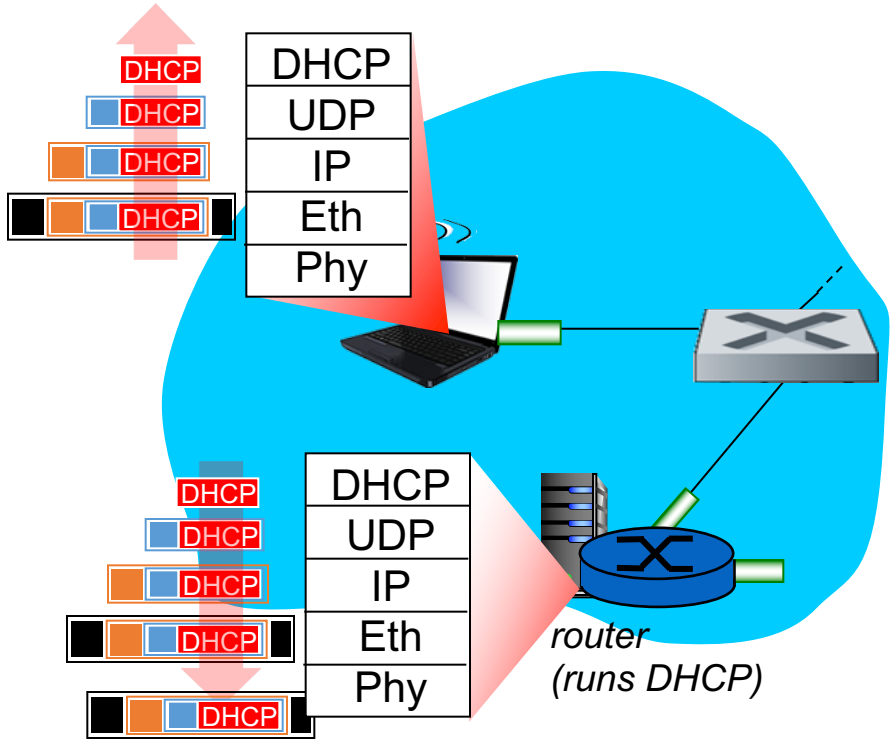
A day in the life… connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*

- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

A day in the life… connecting to the Internet



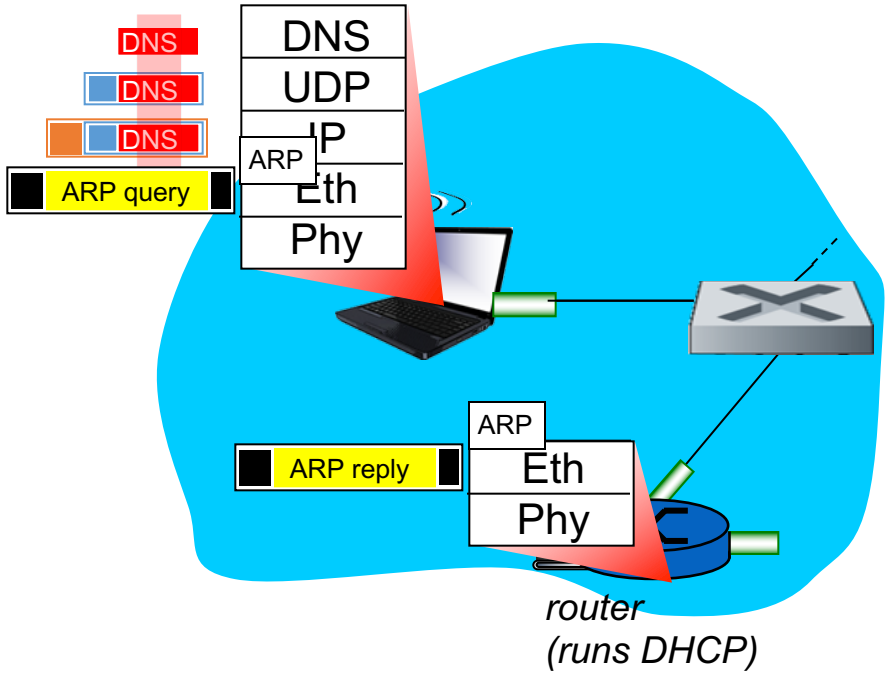
- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server

- encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client

- DHCP client receives DHCP ACK reply

Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

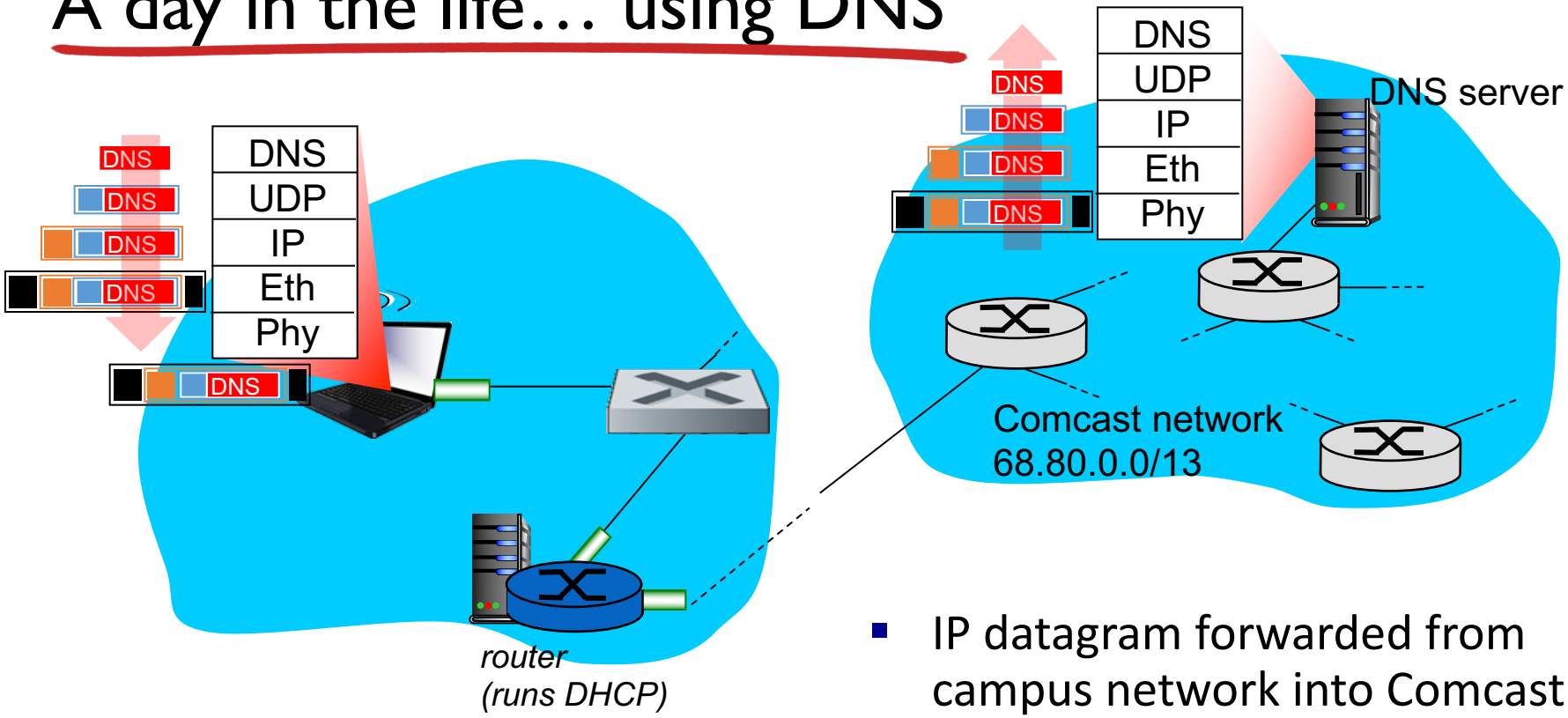
A day in the life… ARP (before DNS, before HTTP)



- before sending *HTTP* request, need IP address of www.google.com: *DNS*

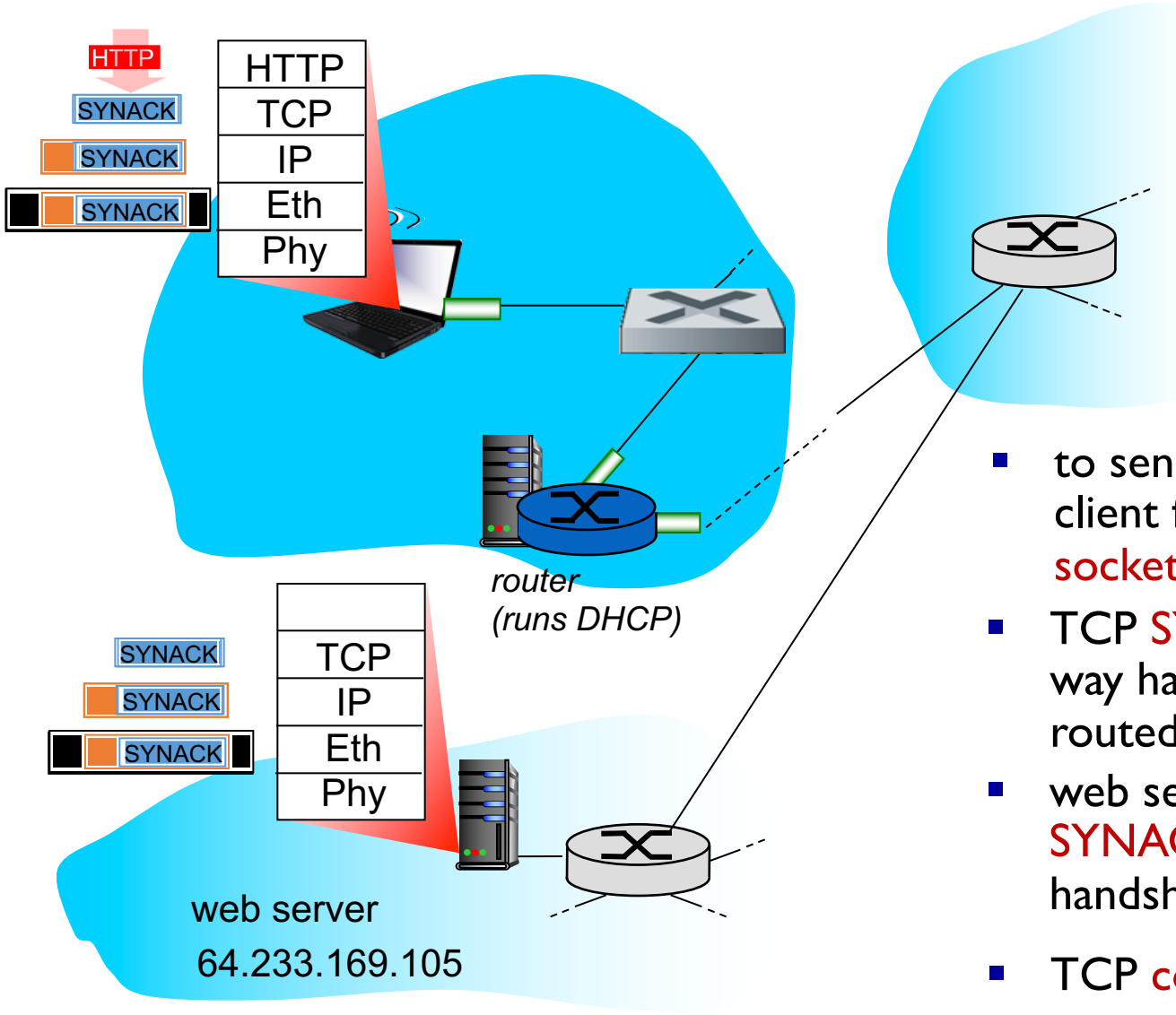
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: ARP
- ARP query broadcast, received by router, which replies with ARP reply giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

A day in the life… using DNS



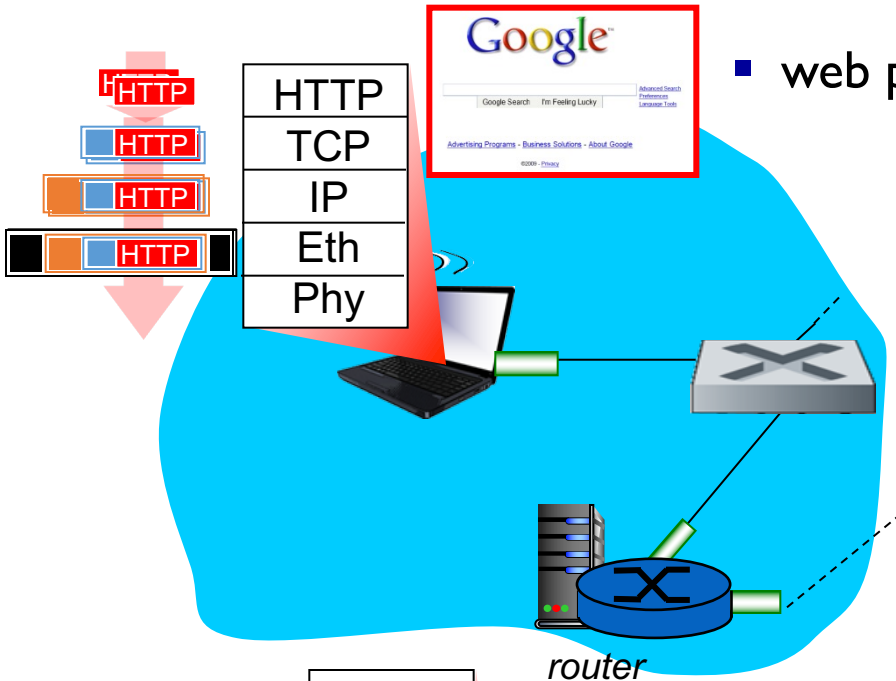
- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com

A day in the life…TCP connection carrying HTTP



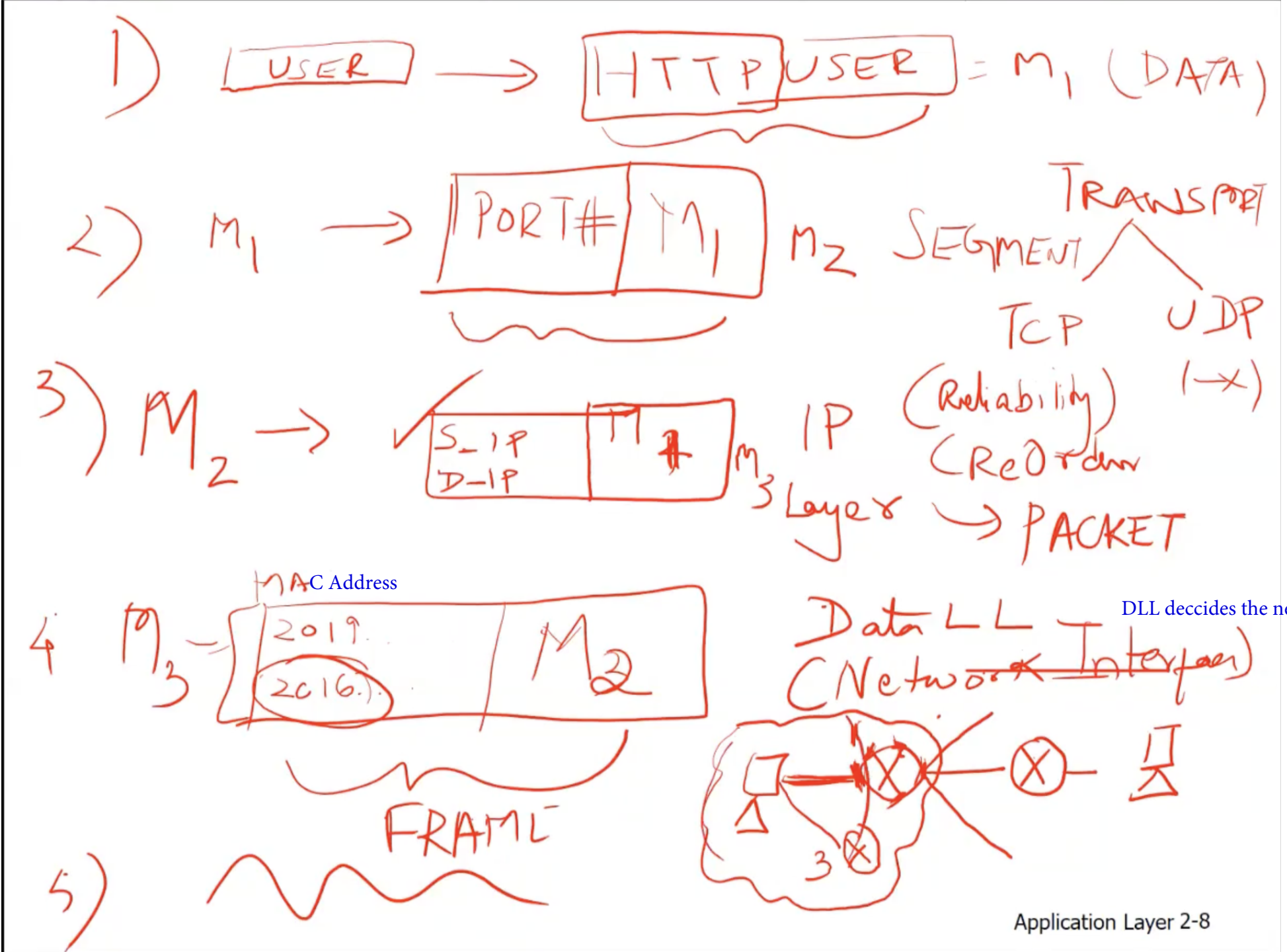
- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in 3-way handshake)
- TCP connection established!

A day in the life… HTTP request/reply



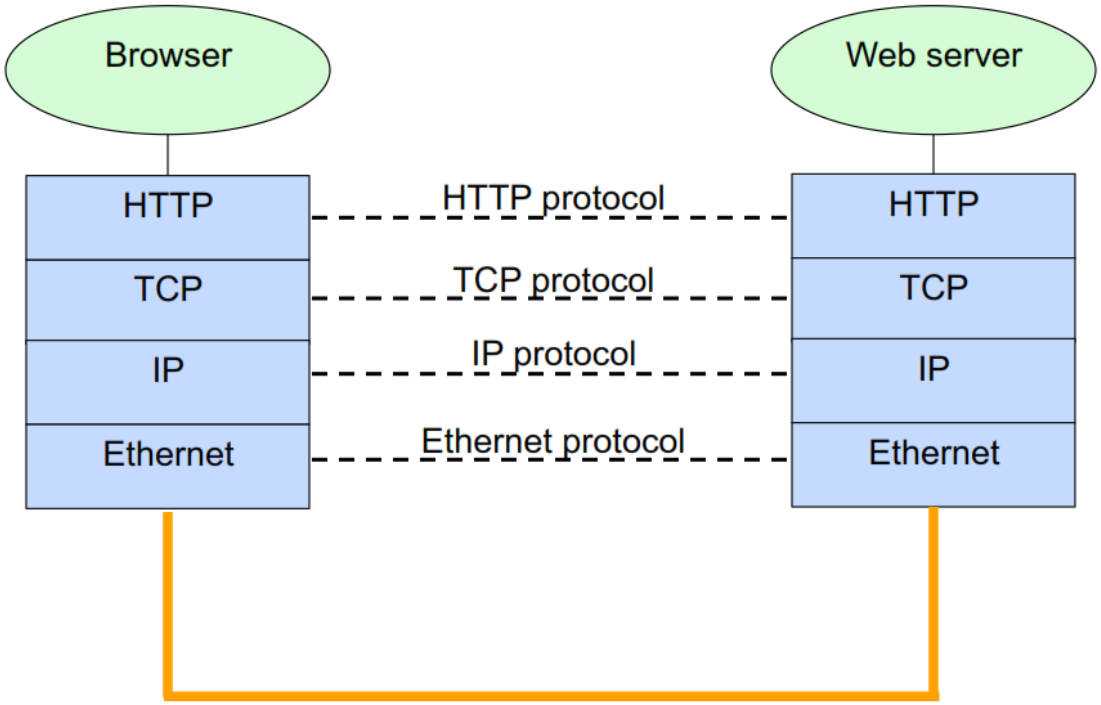
- web page finally (!!!) displayed

- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client


1)
USER
HTTP USER
= M1 (DATA)
2)
M1
PORT#
M1
M2
SEGMENT
TRANSPORT
TCP
UDP
(Reliability)
(ReOrdering)
(X)
3)
M2
S_IP
D-IP
M3
IP
Layer
PACKET
MAC Address
4
M3
2019
2016.)
M2
FRAME
Data LL
CNetwork Interface)
DLL deccides the next router
3
5)

Computer Networks
Transport Layer
Week 5-Lecture 1

Example Protocol Stack

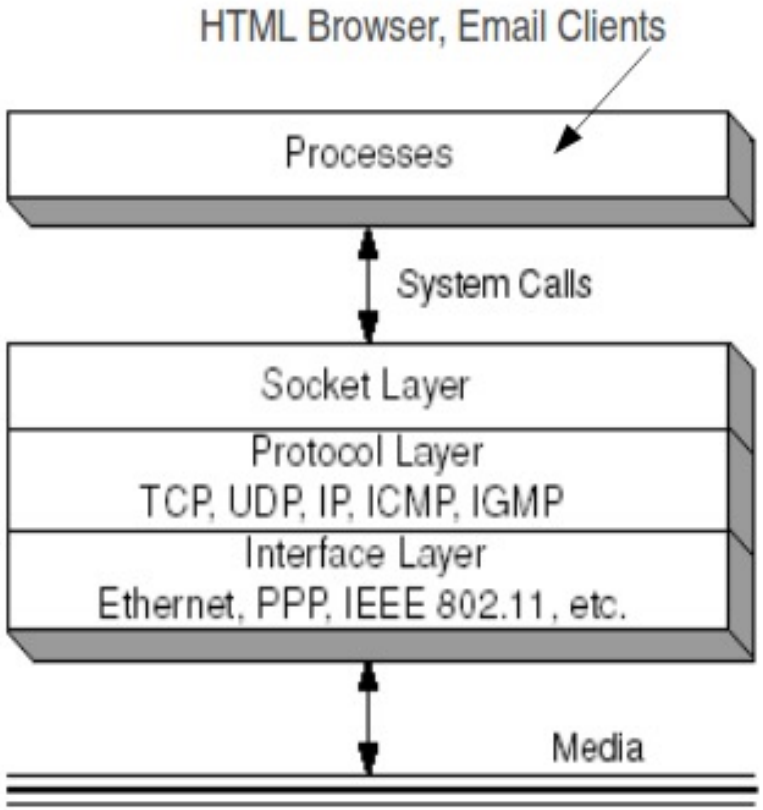


One response per

- A user requests a Web page that consists of some text and three images. For this page, the client will send one request message and receive four response messages
  - FALSE
- Two distinct Web pages (for example, www.daiict.ac.in/faculty.html and www.daiict.edu/staff.html) can be sent over the same persistent connection.
  - TRUE

Networking Code Organization

- Most applications are implemented as *user space* processes.
- Protocols are implemented in the system kernel
  - Socket layer
  - Protocol layer
  - Interface layer



Some network apps

- e-mail, web , text messaging , remote login
- P2P file sharing, multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype), real-time video conferencing
- social networking. search

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	RTSP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

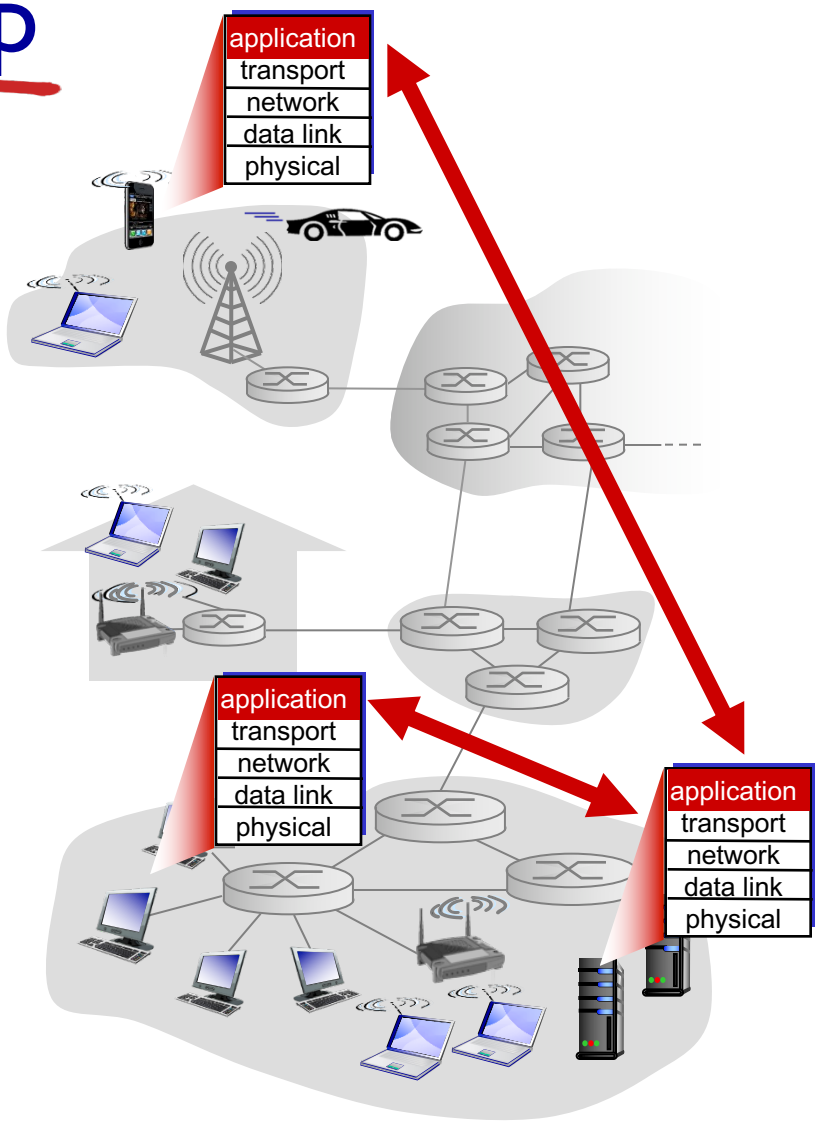
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications



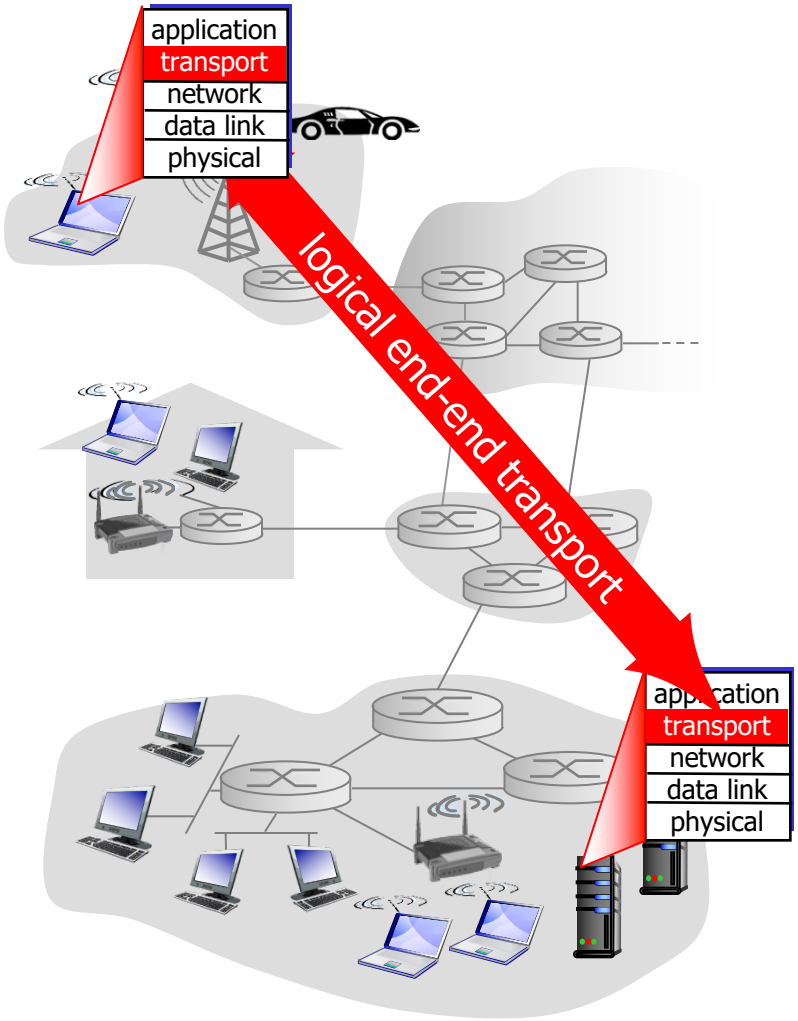
Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

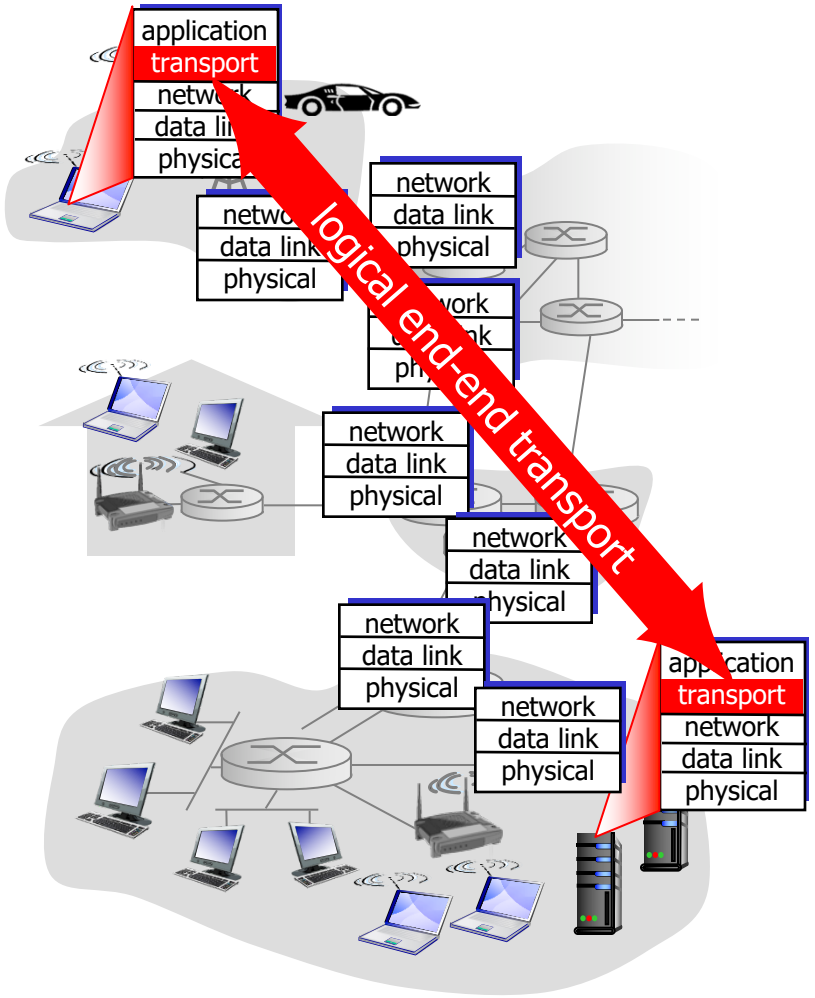
Transport services and protocols

- 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

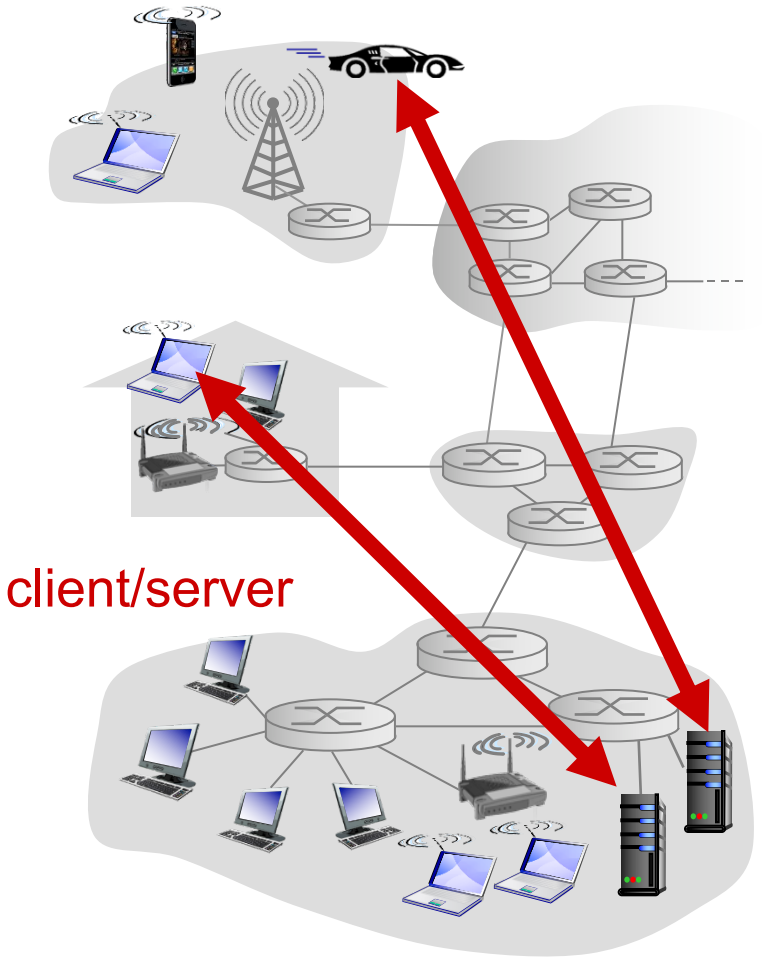


Internet transport-layer protocols

- 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



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

A **server** is any application that provides a service and allows clients to communicate with it.

A **client** is any application that requests a service from a server.

Processes communicating

process: program running within a host

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

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

How do processes exchange messages?

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

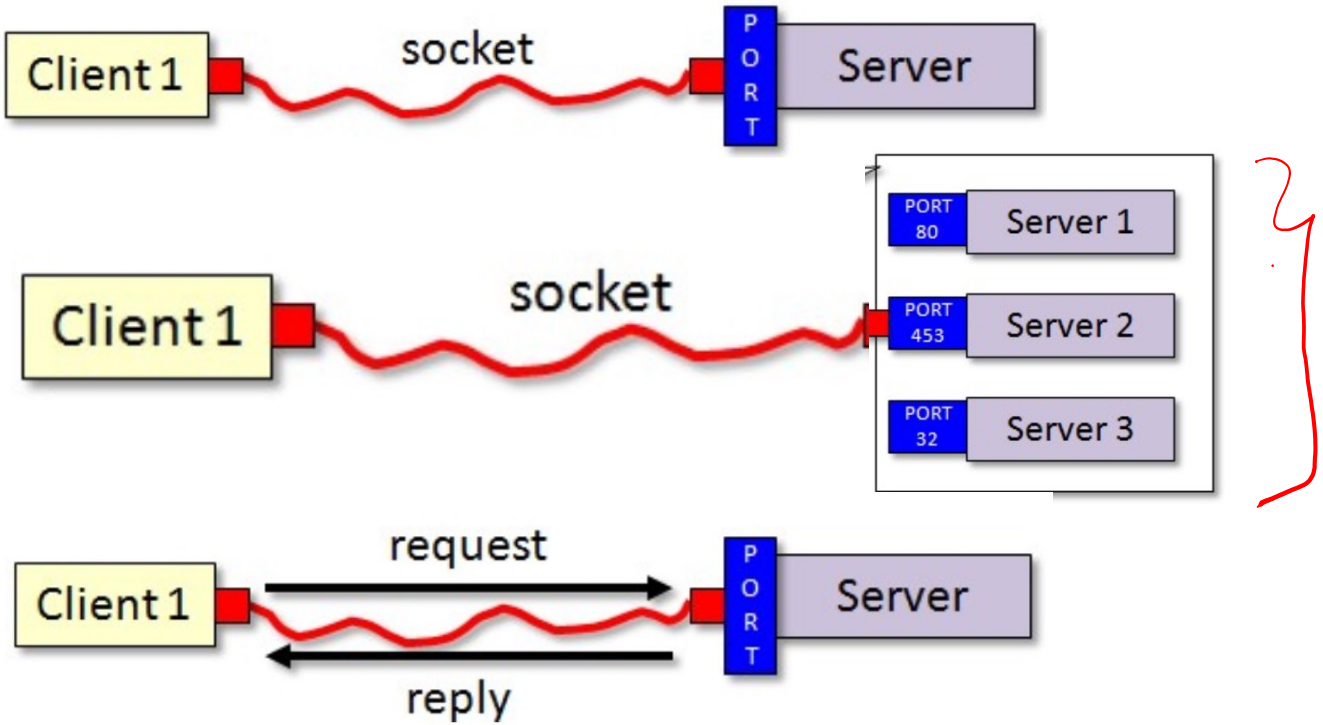
- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25

*A **port** is used as a gateway or "entry point" into an application.*

The port number is a 16-bit number ranging from 0 to 65,535, with ports 0-1023 restricted by well-known applications like HTTP and FTP.

Sockets: (port #, IP addr)

- One endpoint of a two way communication link between 2 programs(processes) running **On** the network.
- Bound to a port number
  - TCP layer → identify the application to which the data is to be sent.
- It is similar to the idea of plugging the two together with a cable.



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

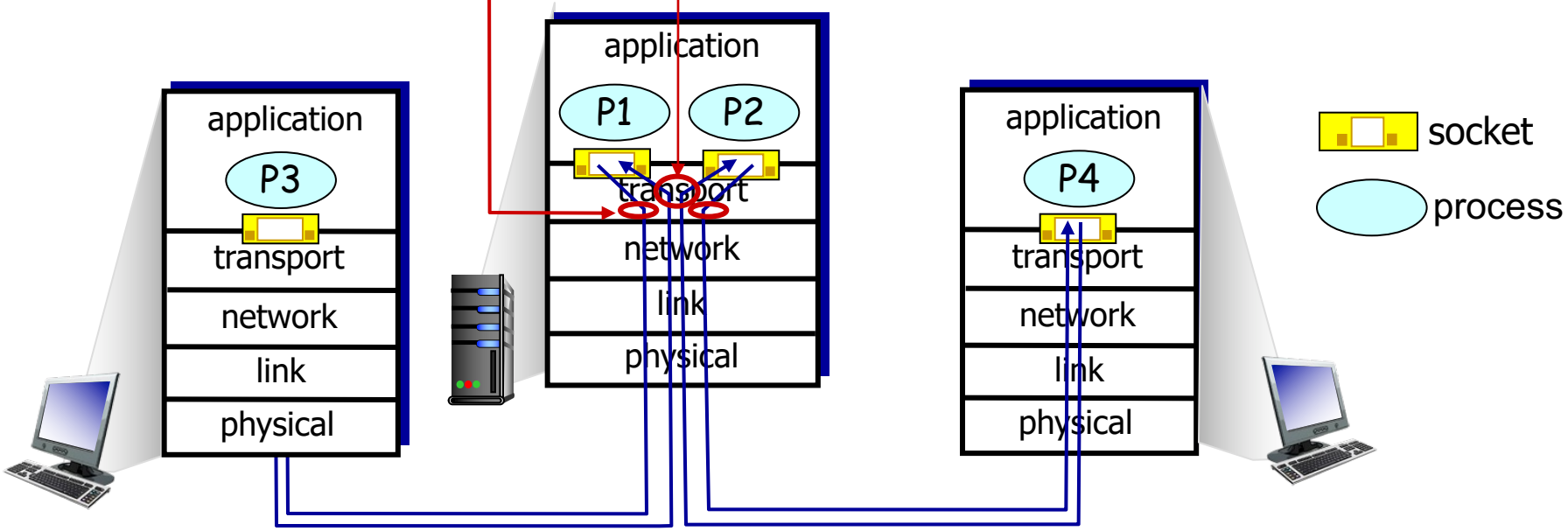
3.6 principles of congestion control

3.7 TCP congestion control

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



Connection-oriented demux

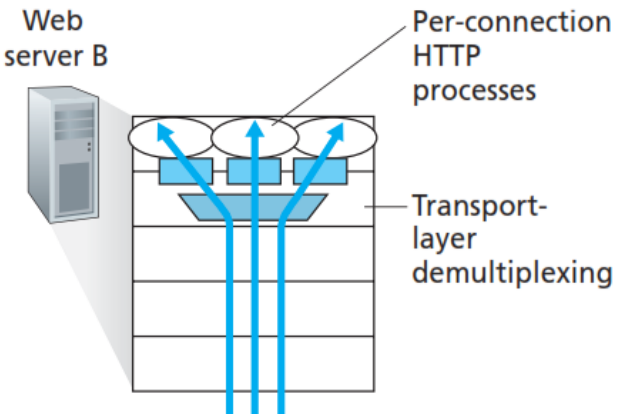
- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple

Connection-oriented demux: example

source port: 7532	dest. port: 80
source IP: C	dest. IP: B

source port: 26145	dest. port: 80
source IP: C	dest. IP: B

source port: 26145	dest. port: 80
source IP: A	dest. IP: B



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connection-oriented transport: TCP

- segment structure

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- reliable data transfer
- flow control
- connection management

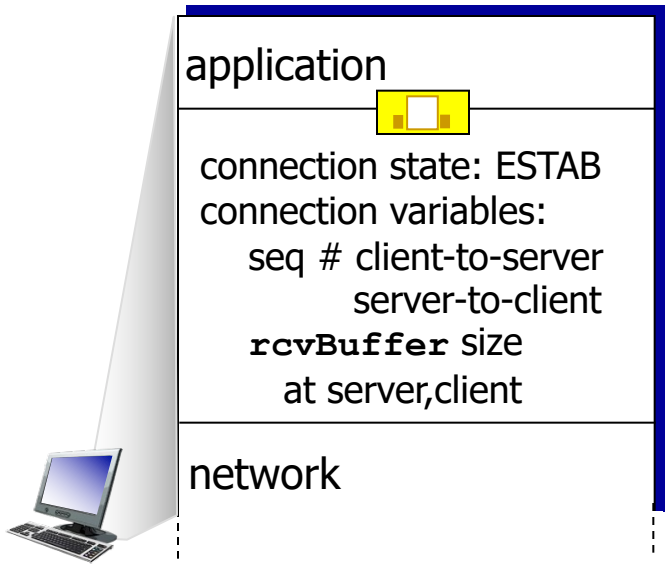
3.6 principles of congestion control

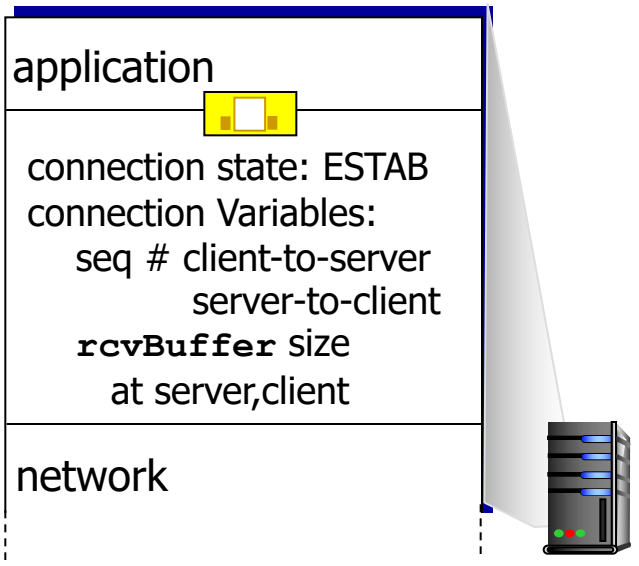
3.7 TCP congestion control

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();
```

Passive And Active Opens

Two sides of a connection

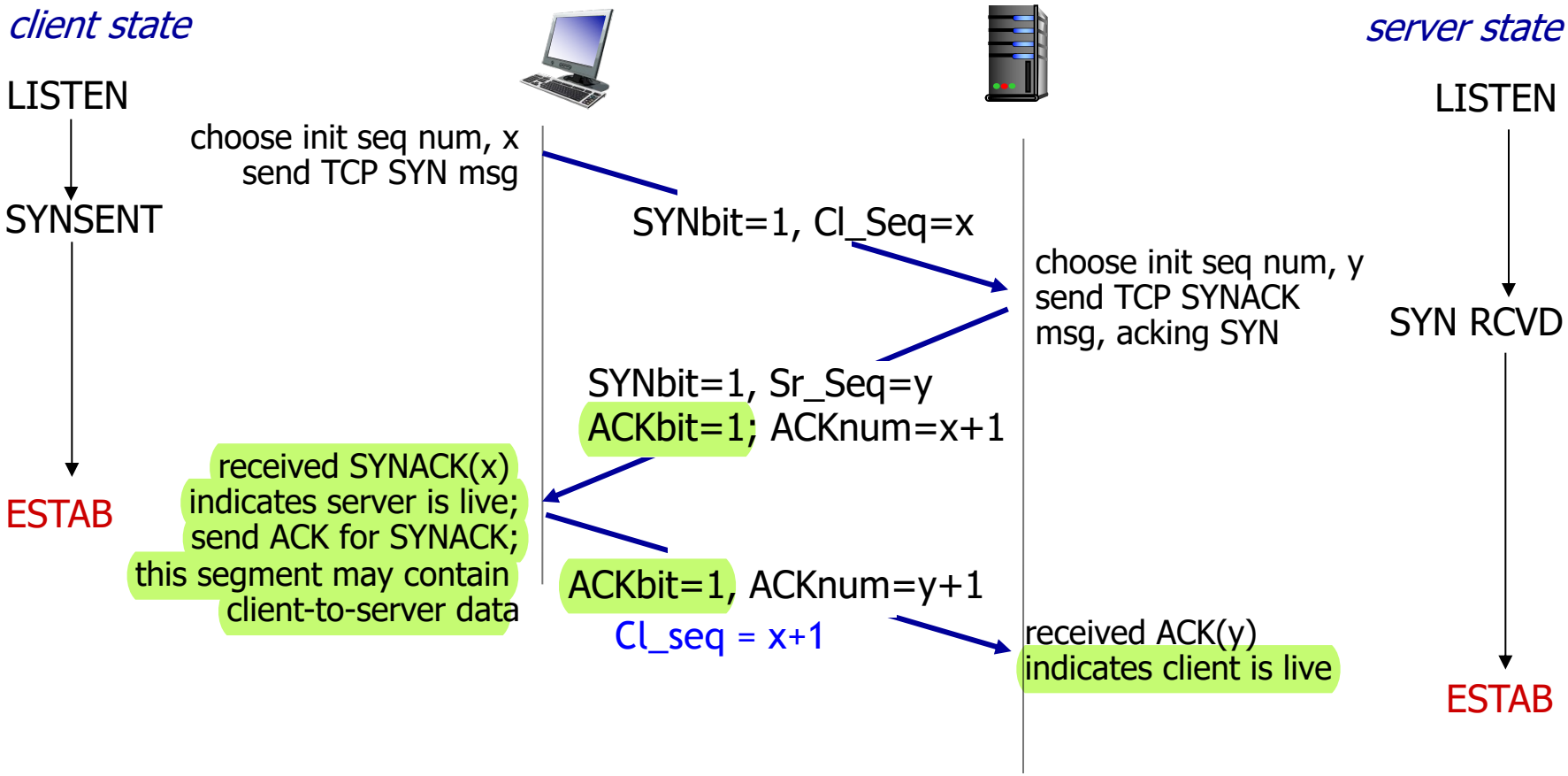
One side waits for contact

- A server program
- Uses TCP's *passive open*

One side initiates contact

- A client program
- Uses TCP's *active open*

TCP 3-way handshake



Transport level -Segment

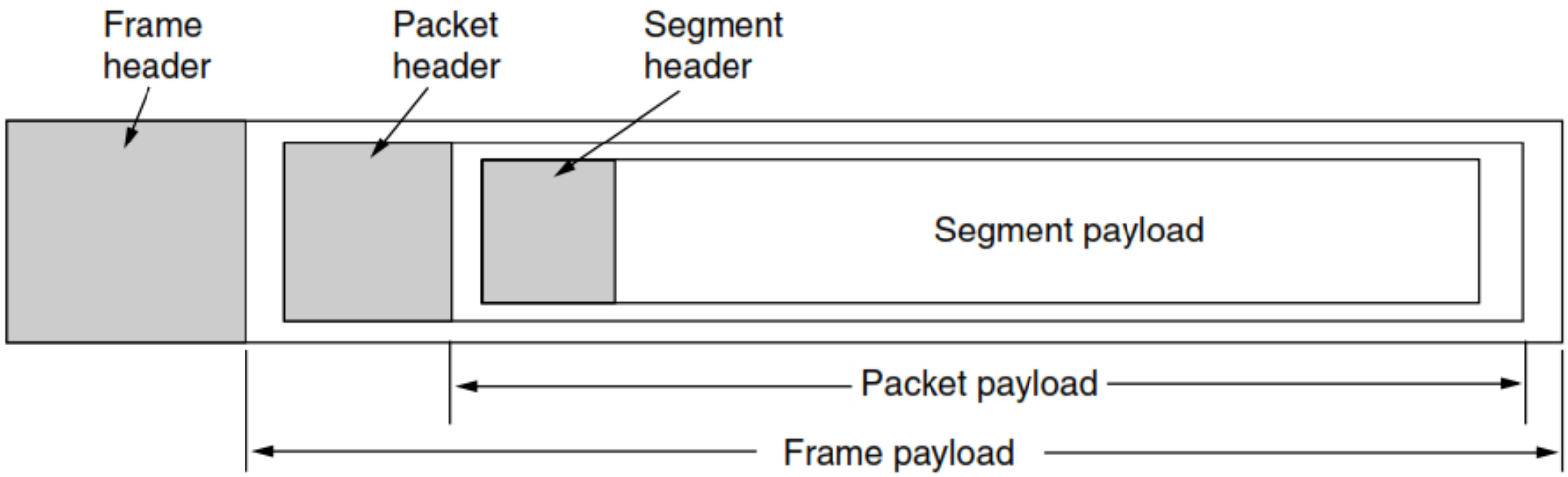
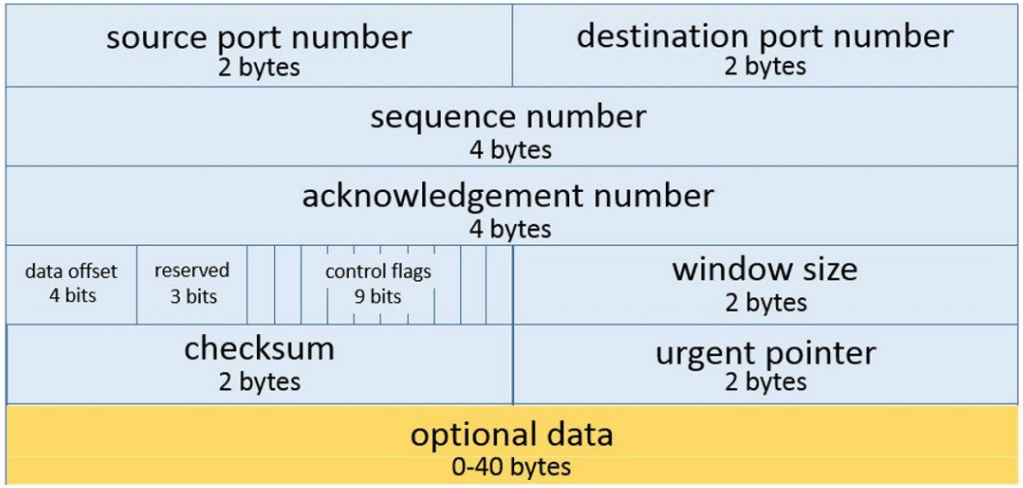


Figure 6-3. Nesting of segments, packets, and frames.

TCP Segment structure



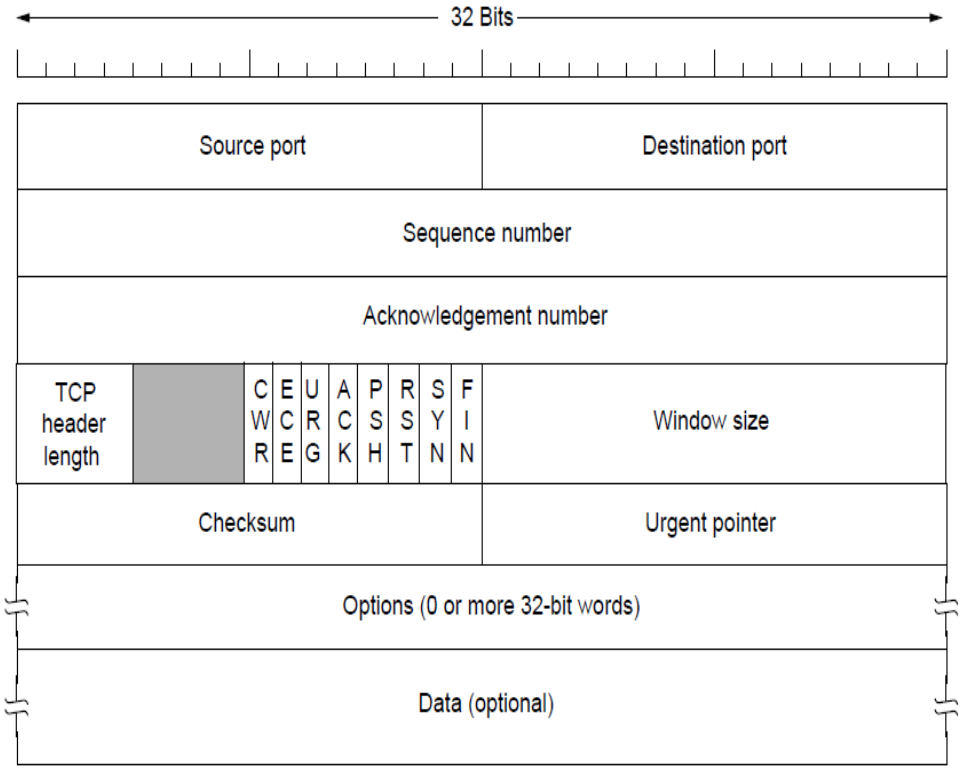
Fixed 20 bytes, options: 0-40 bytes

- Data: 65535-20-20 = 66495 B may follow
- Src, dst ports: 16 bits ident conn endpoints
- Connection identifier: 5-tuple
  - protocol (TCP)
  - source and destination IP addresses
  - source and destination port numbers

Fields

- Sequence number: # of 1st byte in segment
- Ack number: next in-order byte expected
- Header length: 32-bit words (options var)
- Urgent pointer: offset of which urgent data start from current sequence number
- Window size:
  - how many bytes may be sent after ack number
  - 0: ack#-1 rcvd, no more data please

Code flag fields



- ECE: tell sender to slow down
- CWR: tell receiver Congestion Win Reduced
- URG: set to 1 when Urgent Pointer is used
- ACK: indicate Ack number is valid
- PSH: request rcvr to push data, not buffer
- RST: reset connection
  - host crash, invalid segment, refuse connection
- SYN: used in establishing connection
- FIN: release connection, no more data

TCP Segment Format

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

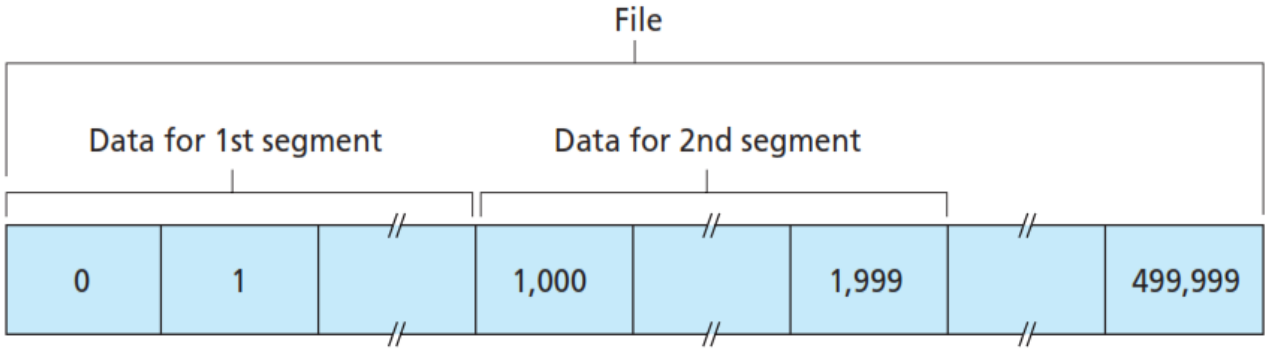
The sequence number for a segment is the byte-stream number of the first byte in the segment.

The acknowledgment number that Host A puts in its segment is the sequence number of the next byte Host A is expecting from Host B.

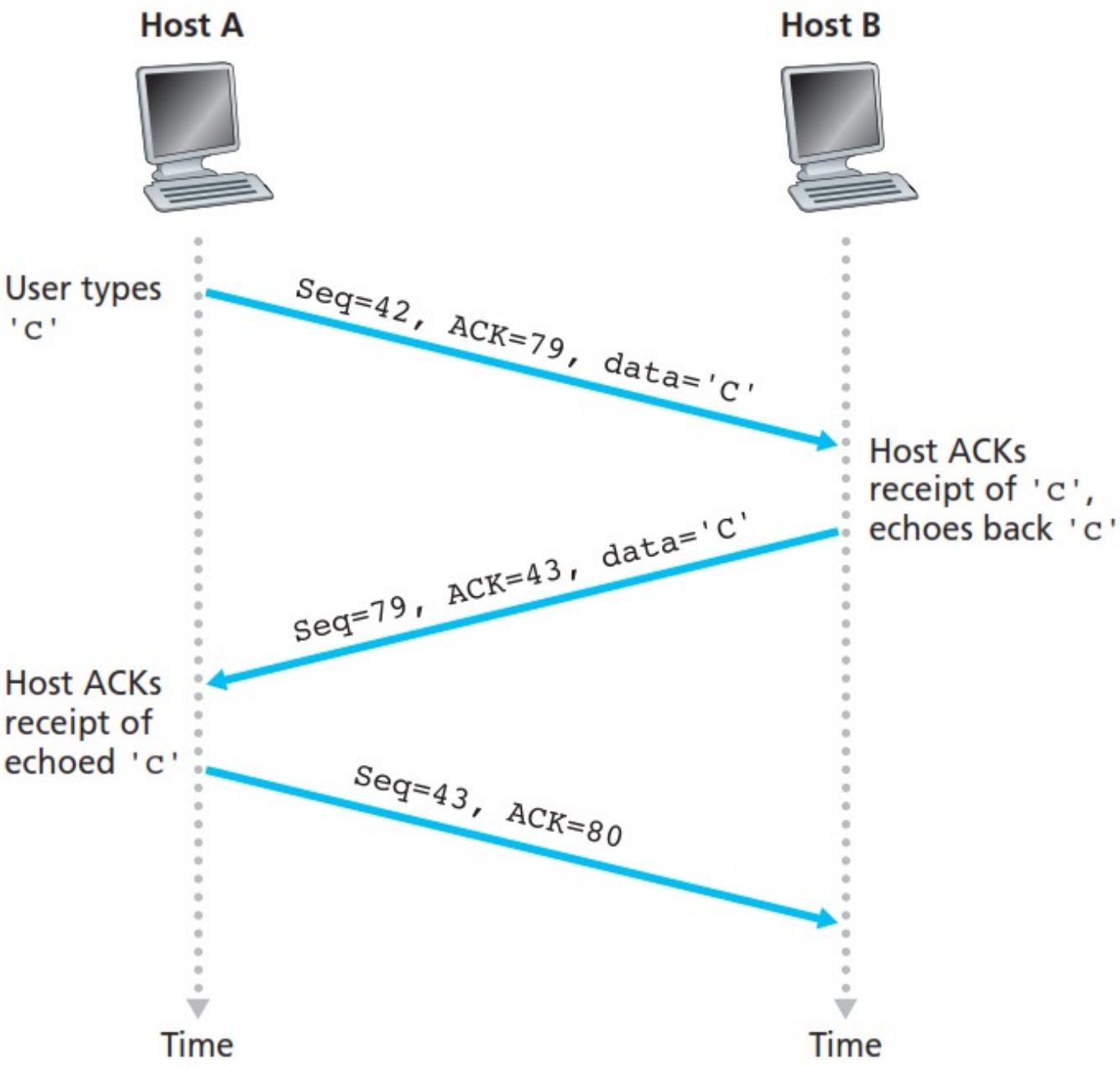
Seq and ACK numbers

Suppose that a process in Host A wants to send a stream of data to a process in Host B over a TCP connection. The TCP in Host A will implicitly number each byte in the data stream. Suppose that the data stream consists of a file consisting of 500,000 bytes, that the MSS is 1,000 bytes

How many segments is the data divided into?



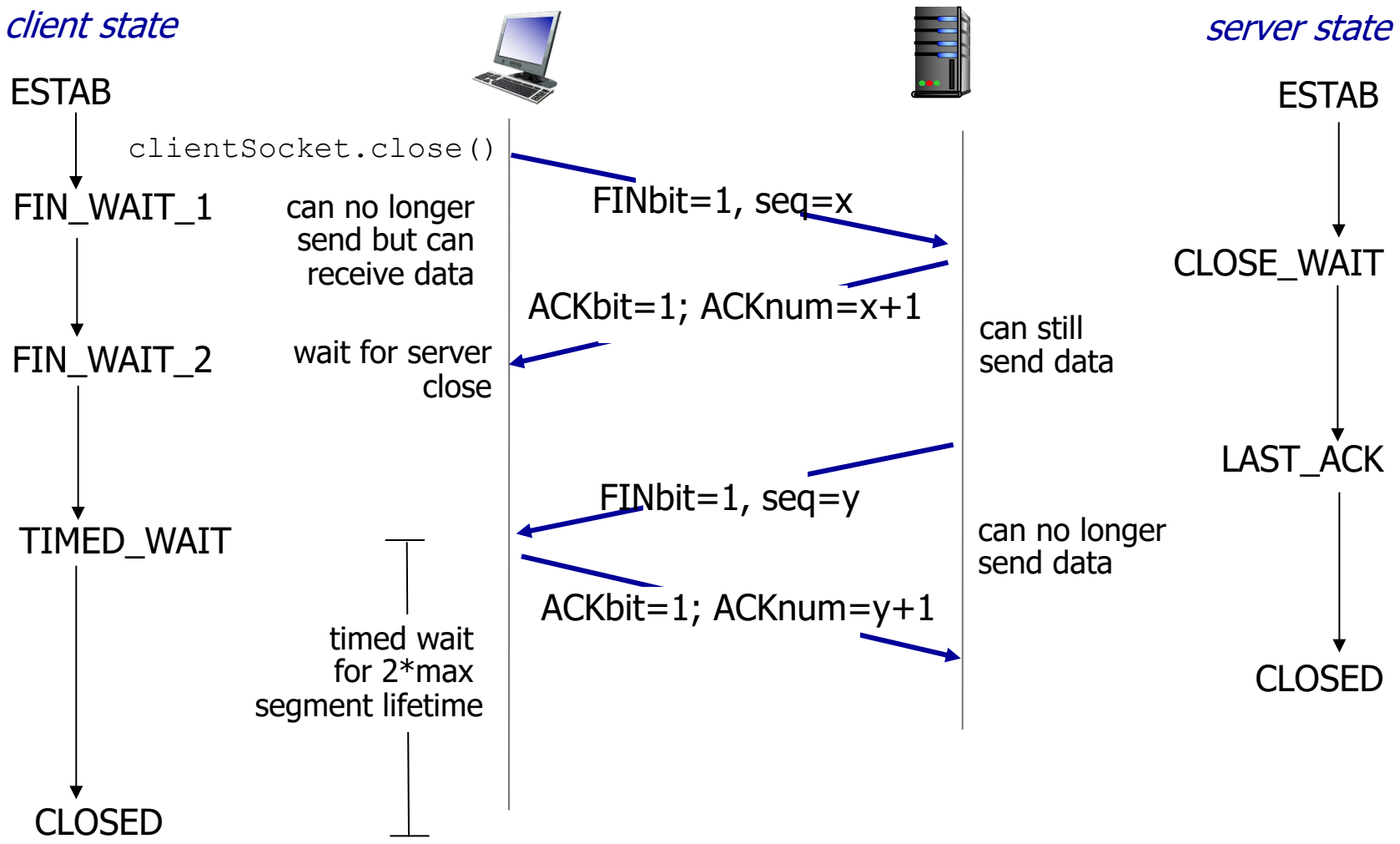
Dividing file data into TCP segments


Host A
Host B
User types 'c'
Seq=42, ACK=79, data='c'
Host ACKs receipt of 'c', echoes back 'c'
Seq=79, ACK=43, data='c'
Host ACKs receipt of echoed 'c'
Seq=43, ACK=80
Time
Time

TCP: closing a connection

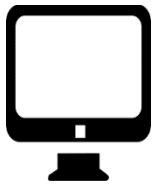
- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

TCP: closing a connection



TCP Segment transaction example

16	16	32	32	4	6	6	16	16	16	0 or 32	bits
Source Port	Dest. Port	SEQ No.	ACK No.	HLEN	Res.	Code Bits	Window	Checksum	Urgent Pointer	IP Options	

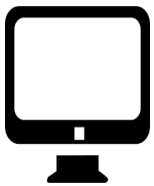


160.221.172.250

Type of segment	**160.221.172.250**	**160.221.73.26**
SYN	Seq.no. 8656	
	Ack.no. 0	
	Window 8192	
	LEN = 0 bytes	

160.221.73.26

SYN-ACK		Seq.no. 80009
		Ack.no. 8657
		Window 8760
		LEN = 0 bytes



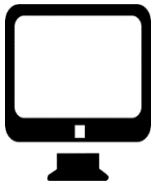
ACK	Seq.no. 8657	
	Ack.no. 80010	
	Window 8760	
	LEN = 0 bytes	

HANDSHAKE SESSION COMPLETE

DATA TRANSFER SESSION STARTS

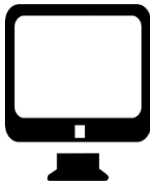
	Seq.no. 8657	
	Ack.no. 80010	
	Window 8760	
	LEN = 72 bytes of data	

		Seq.no. 80010
		Ack.no. 8729
		Window 8688
		LEN = 60 bytes of data



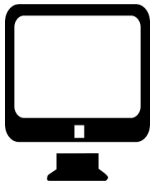
	Seq.no. 8729	
	Ack.no. 80070	
	Window 8700	
	LEN = 156 bytes of data	

		Seq.no. 80070
		Ack.no. 8885
		Window 8532
		LEN = 152 bytes of data



FIN	Seq.no. 8885	
	Ack.no. 80222	
	Window 8548	
	LEN = 0 bytes	

FIN-ACK		Seq.no. 80222
		Ack.no. 8886
		Window 8532
		LEN = 0 bytes



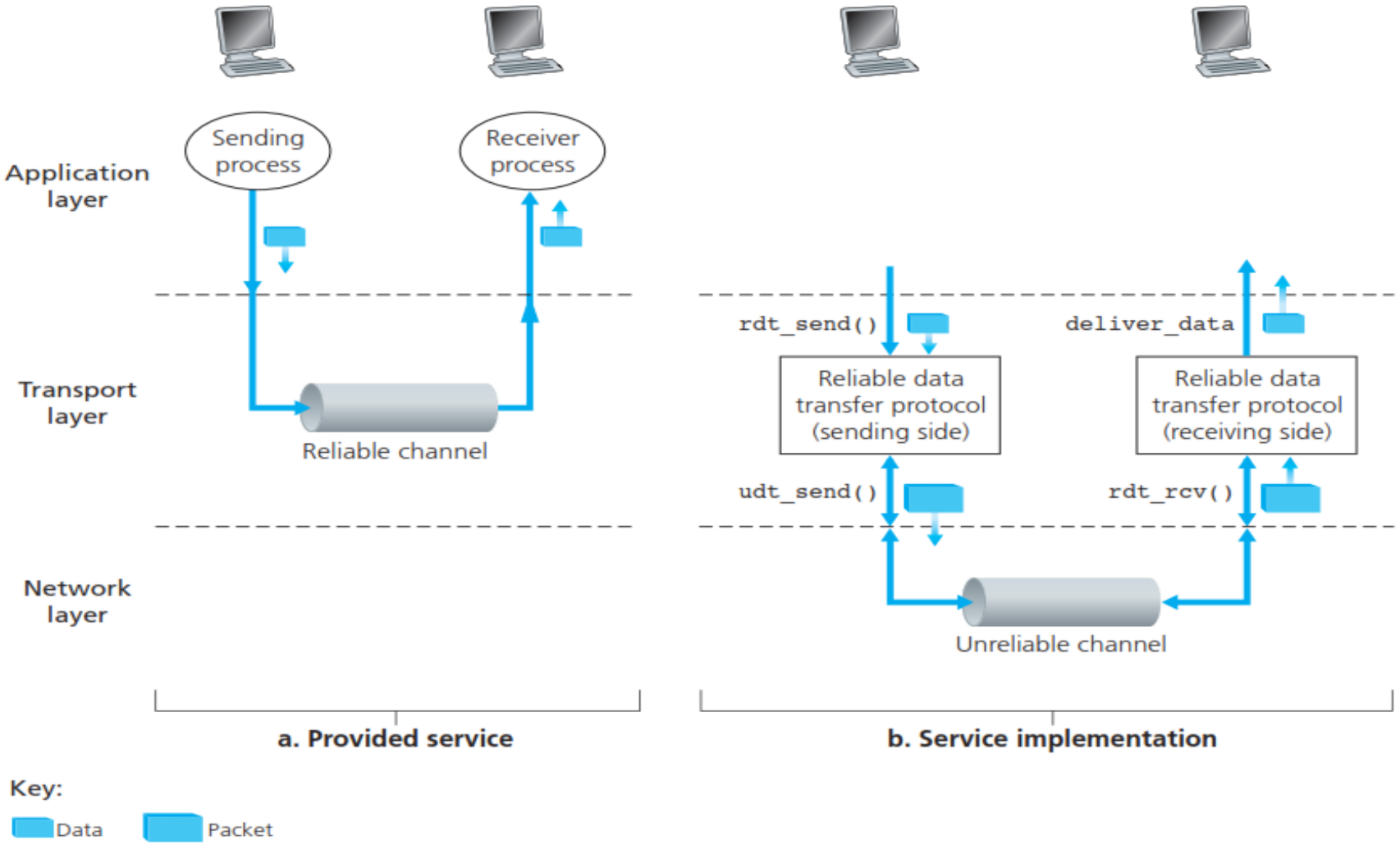
ACK	Seq.no. 8886	
	Ack.no. 80223	
	Window 8548	
	LEN = 0 bytes	

Computer Networks
Transport layer
Week 6-Lecture 1

TCP reliable data transfer

- How does TCP provide a rdt service on top of IP's unreliable service.
- How does TCP regulate end-end byte transmissions?
- How does TCP utilize the bandwidth effectively?

TCP reliable service and its implementation view

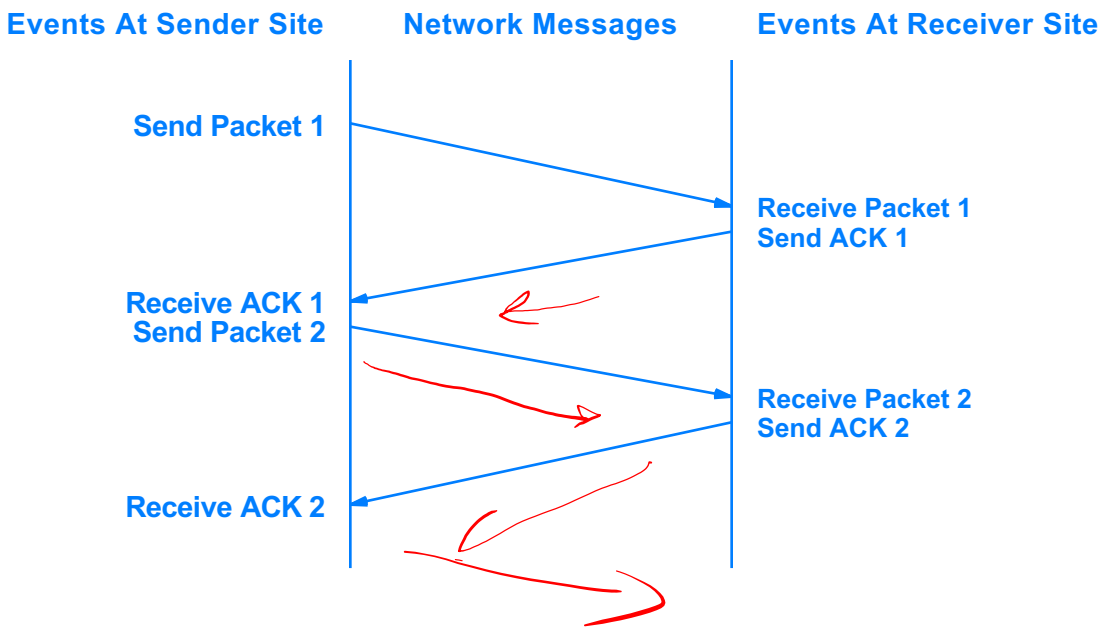


Providing Reliability

Traditional technique: Positive Acknowledgement with Retransmission (PAR)

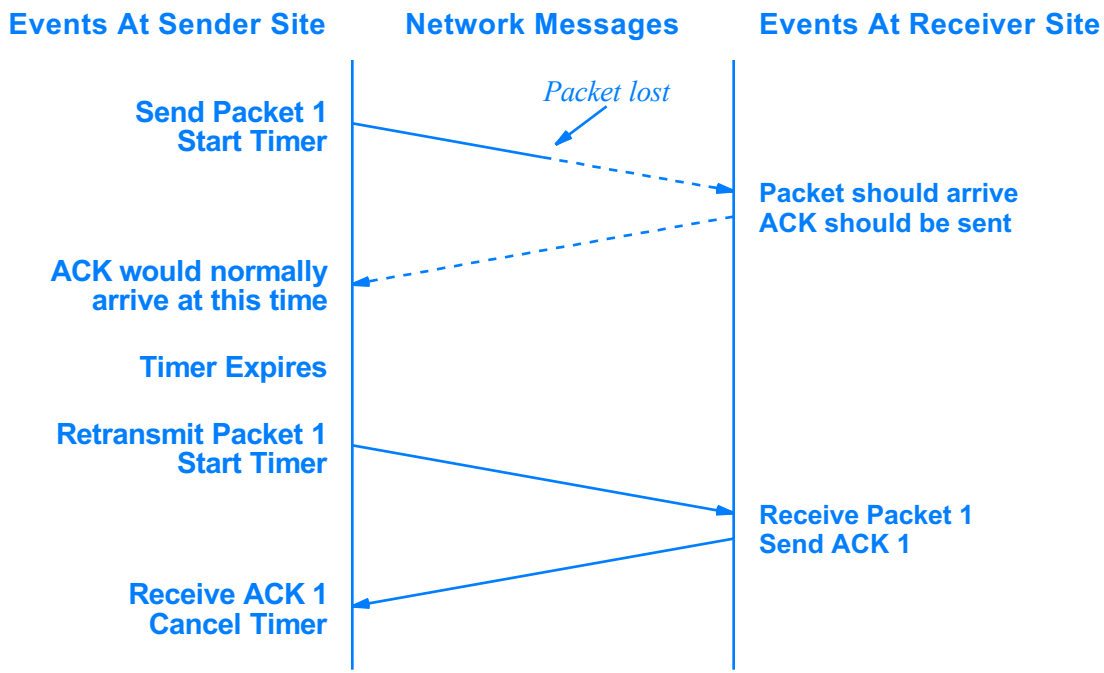
- Sender starts timer whenever transmitting
- Receiver sends *acknowledgement* when data arrives
- Sender retransmits if timer expires before acknowledgement arrives

Illustration Of Acknowledgements



Time moves from top to bottom in the diagram

Illustration Of Recovery After Packet Loss

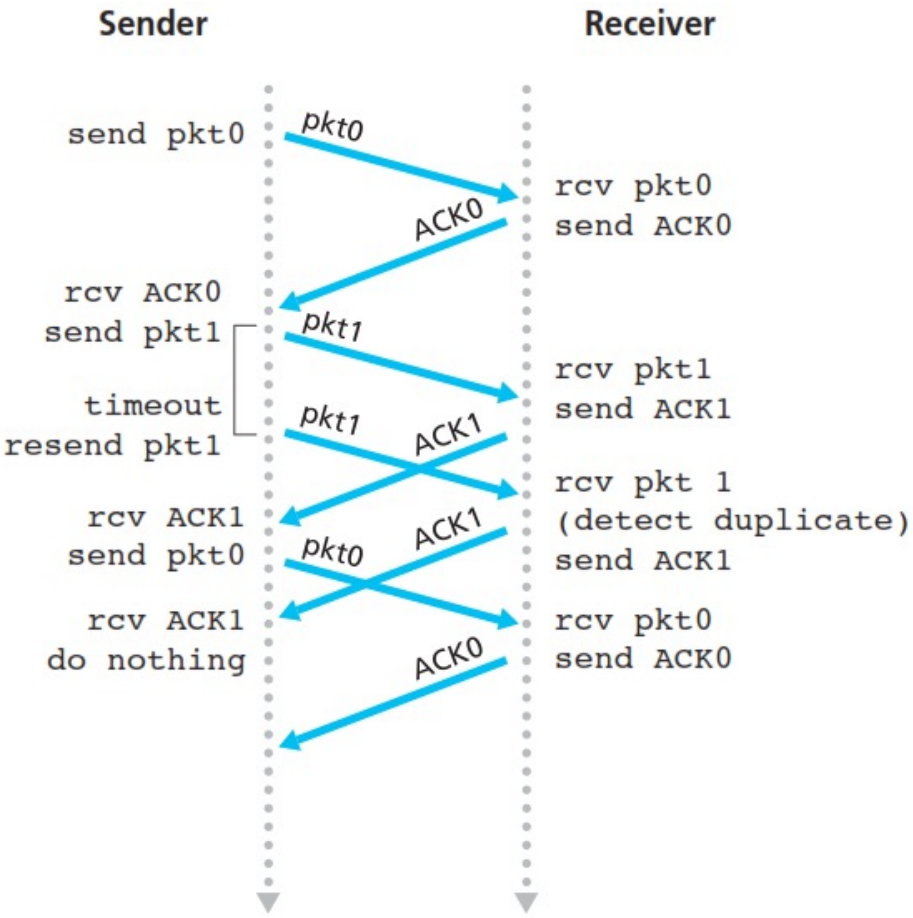


- But what if duplicate segments are delivered to the end hosts?

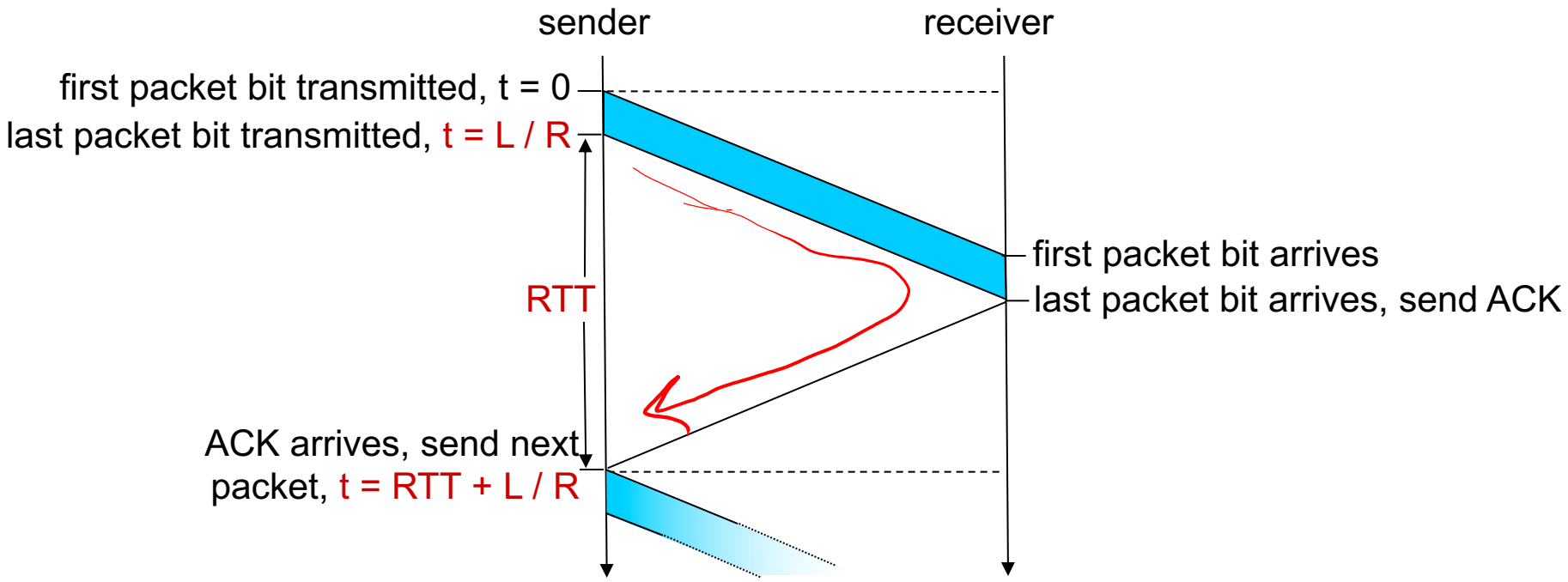
- When would duplication arise?
  - when networks experience high delays
    - that cause premature retransmission. (EARLY)

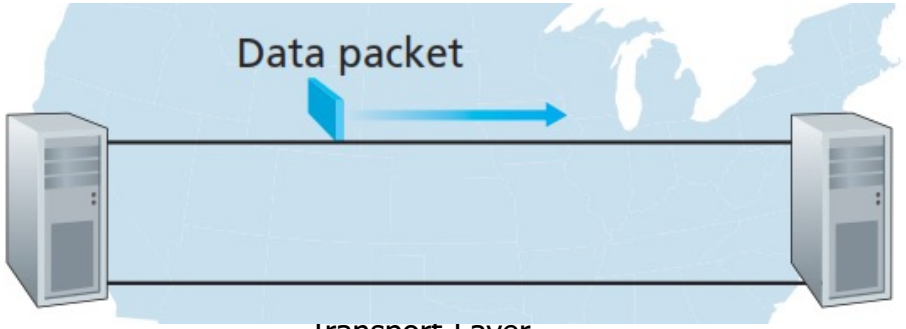
  - Both segments and acknowledgements can be duplicated

- How does TCP distinguish dup segment?
  - detect duplicate segments → Sequence number (SN)
  - send SN via Acknowledgements, so the receiver can correctly associate acknowledgements with packets.



PAR a.k.a stop-and-wait operation





Performance of Stop and Wait

- RDT is good, but performance is bad
- e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

  - U_{sender}: *utilization* – fraction of time sender busy sending

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

- What is the effective throughput?
  - How much max could he send in a give time?

The Problem With Simplistic PAR

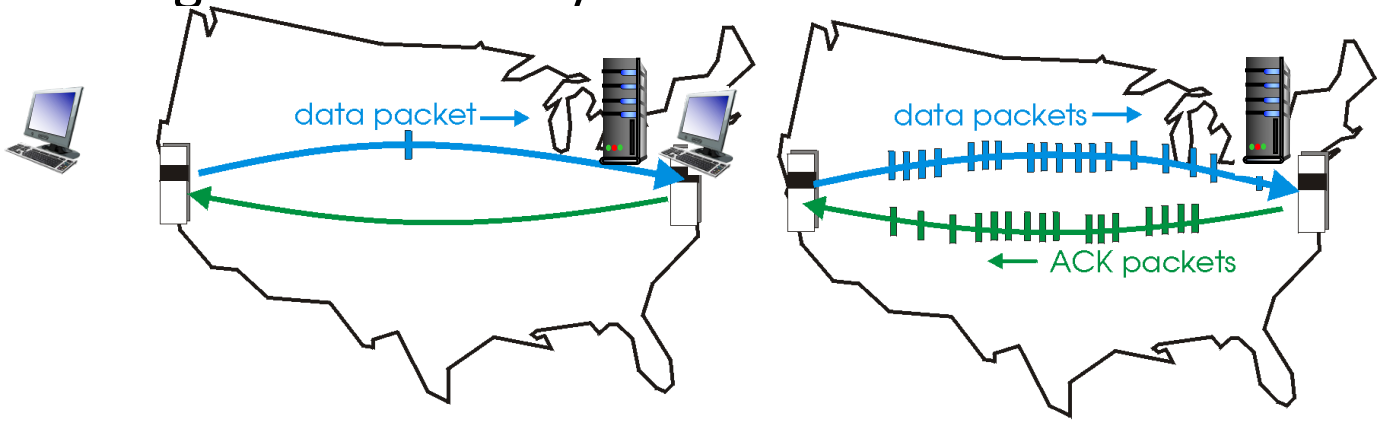
A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.

Problem is especially severe if network has long latency

Pipelined protocols

pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

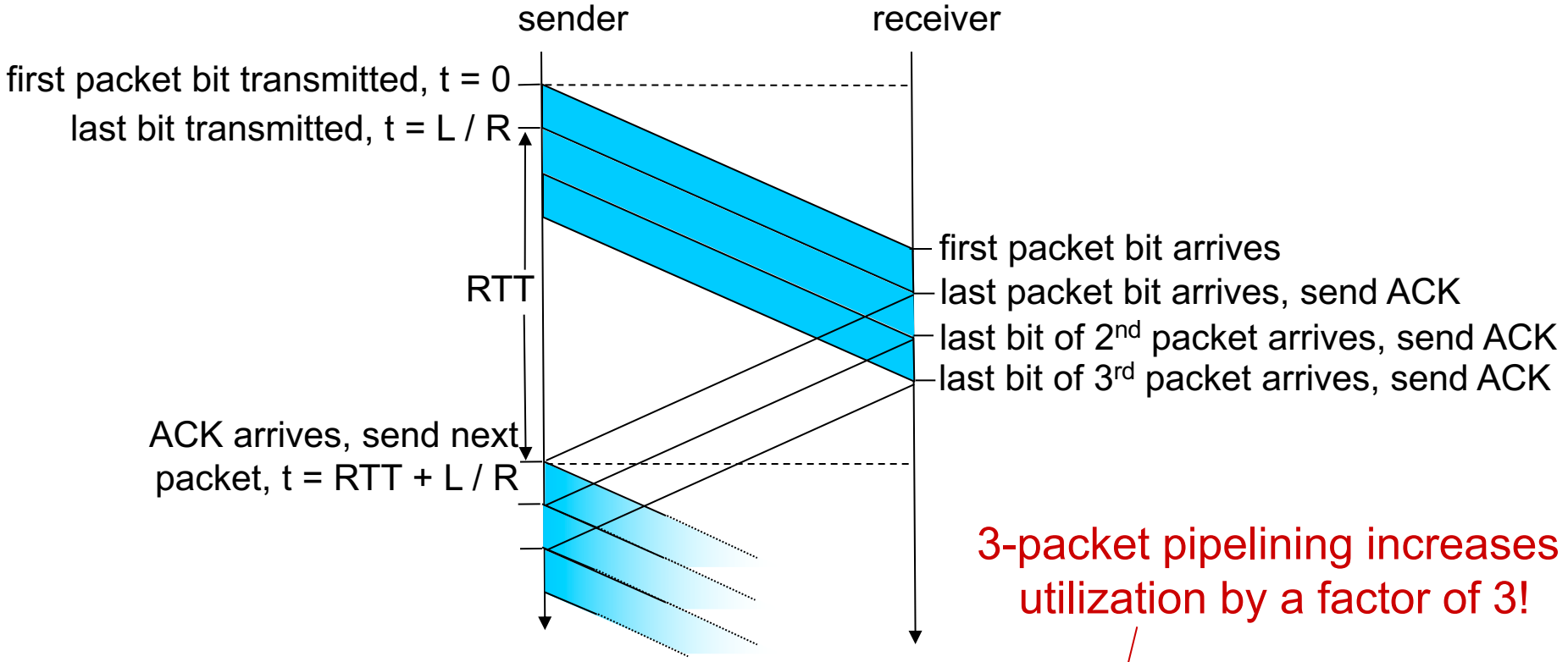
- range of sequence numbers must be increased
- 0 TO 2^k-1
- buffering at sender and/or receiver



(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



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

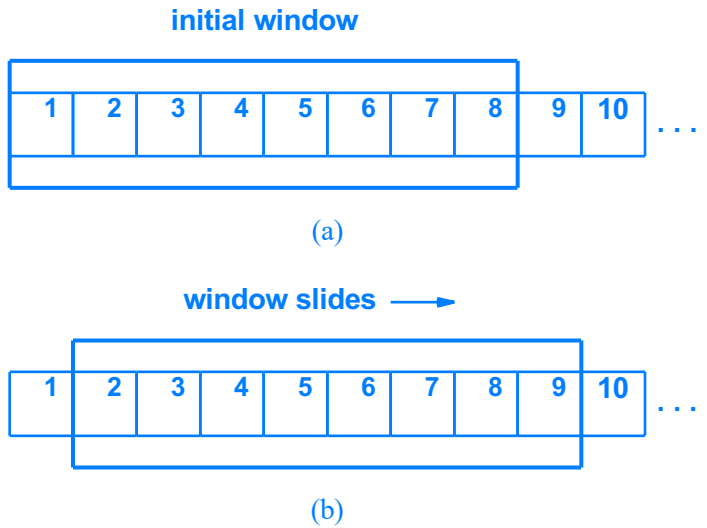
$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Solving The Problem

Allow multiple packets to be outstanding at any time

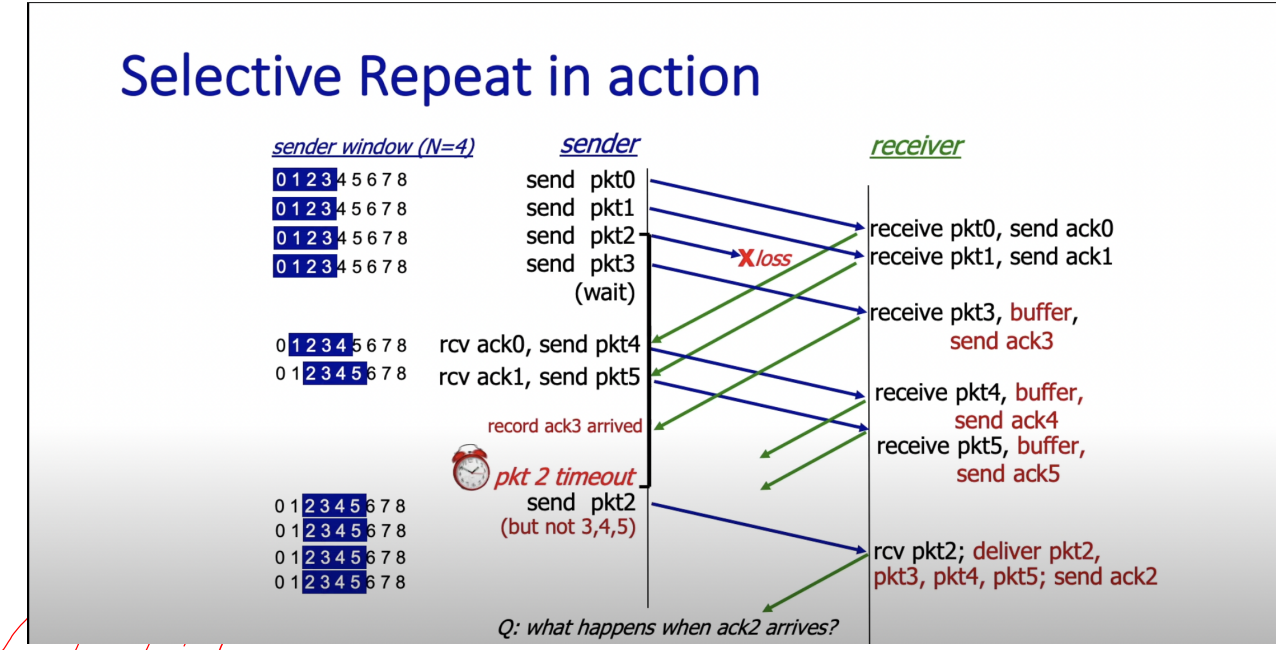
Still require acknowledgements and retransmission

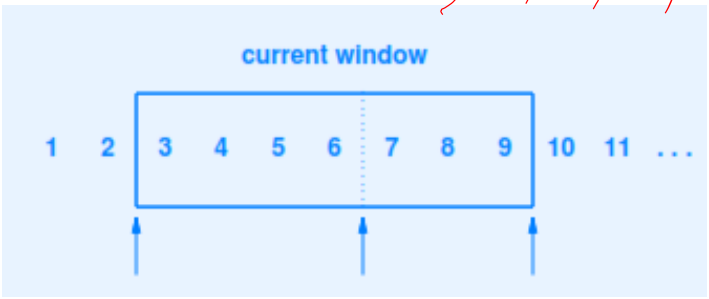
Known as *Pipelining* and it is implemented through a *sliding window*



Window size VARIES

As acknowledgement arrives, window moves forward





- Bytes through 2 are acknowledged
- Bytes 3 through 6 not yet acknowledged
- Bytes 7 though 9 waiting to be sent
- Bytes above 9 lie outside the window and cannot be sent

Why Sliding Window Works

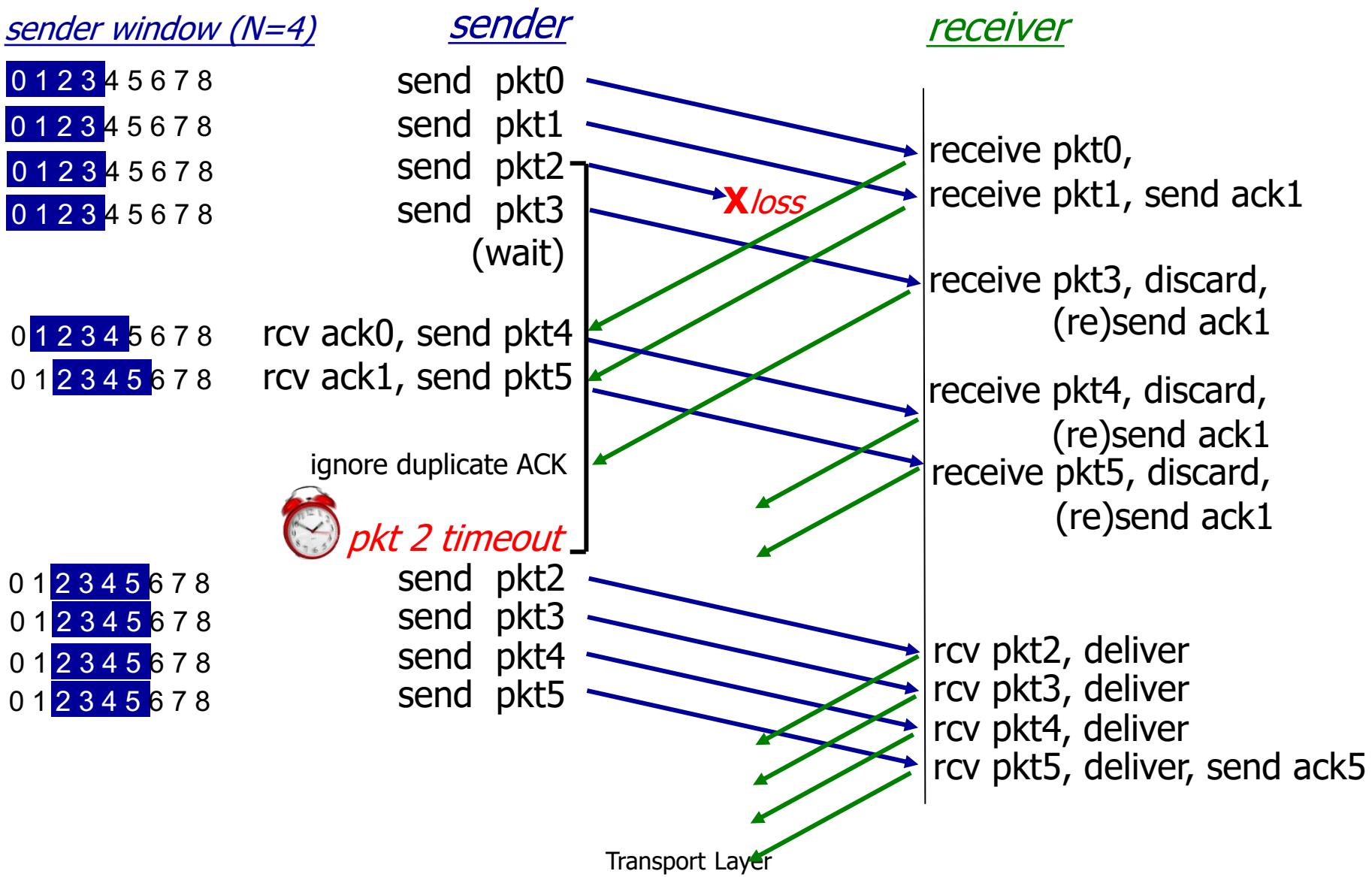
Because a well-tuned sliding window protocol keeps the network completely saturated with packets, it obtains substantially higher throughput than a simple positive acknowledgement protocol.

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

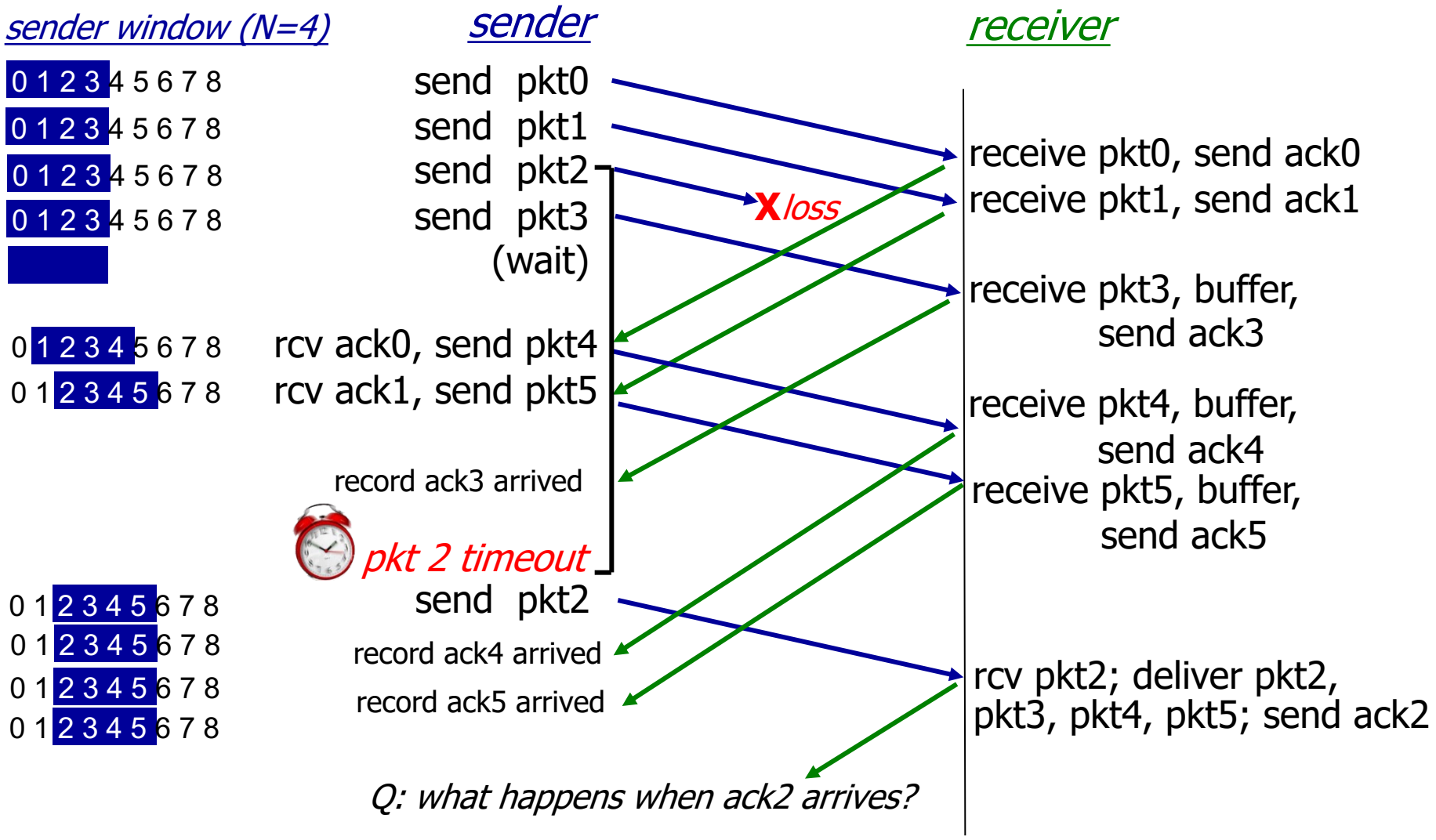
GBN in action



Selective repeat

- 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

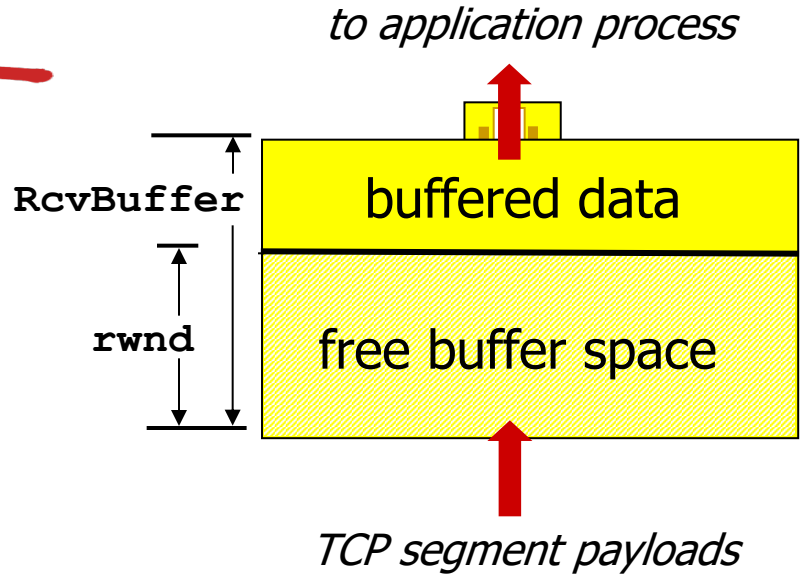
Selective repeat in action



window wont move forward as ack for 2 not received

TCP flow control

- receiver "advertises" free buffer space by including **WindowSize** (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

flow control

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

Selective repeat

sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n)

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

receiver

pkt n within [rcvbase, rcvbase+N-1]

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

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

- ACK(n)

otherwise:

- ignore

Computer Networks
Transport layer
Week 6-Lecture 2

Principles of congestion control

Congestion:

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

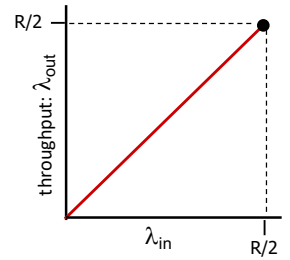
flow control: one sender too fast for one receiver

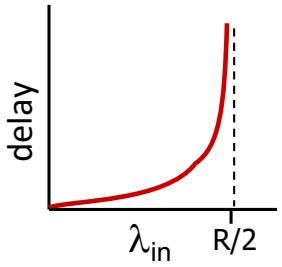
congestion control:

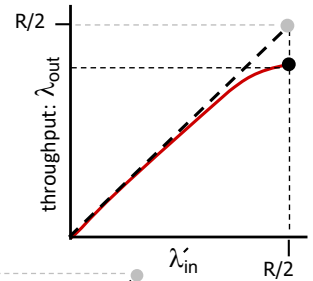
too many senders, sending too fast

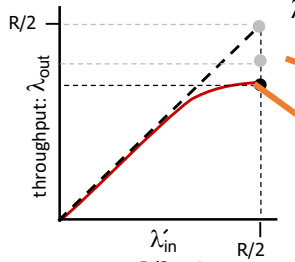
Causes/costs of congestion: insights

- 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



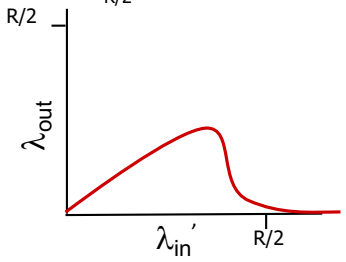






"wasted" capacity due to un-needed retransmissions

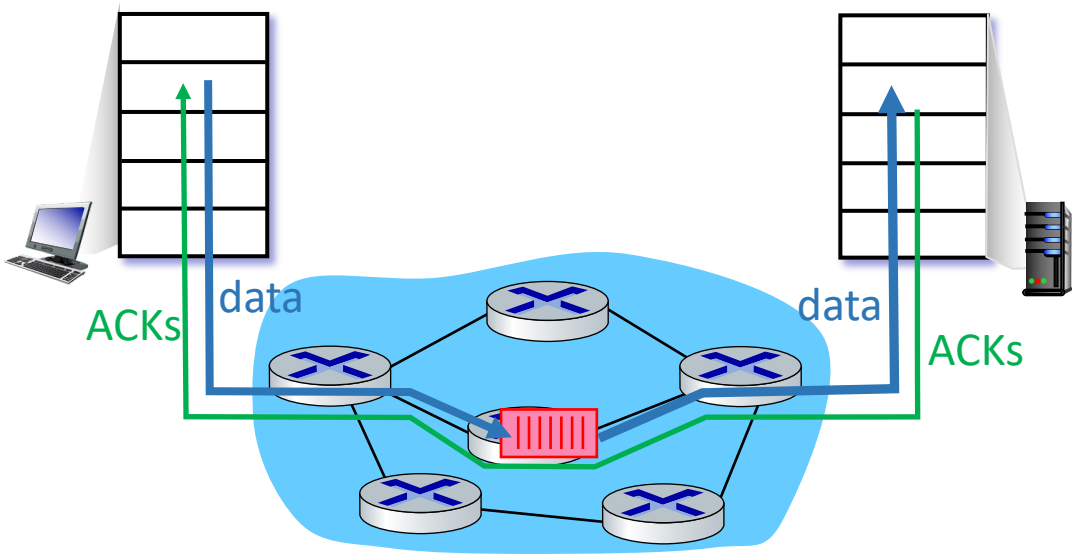
when sending at R/2, some packets are retransmissions, including needed and *un-needed* duplicates, that are delivered!



Approaches towards congestion control

End-end congestion control:

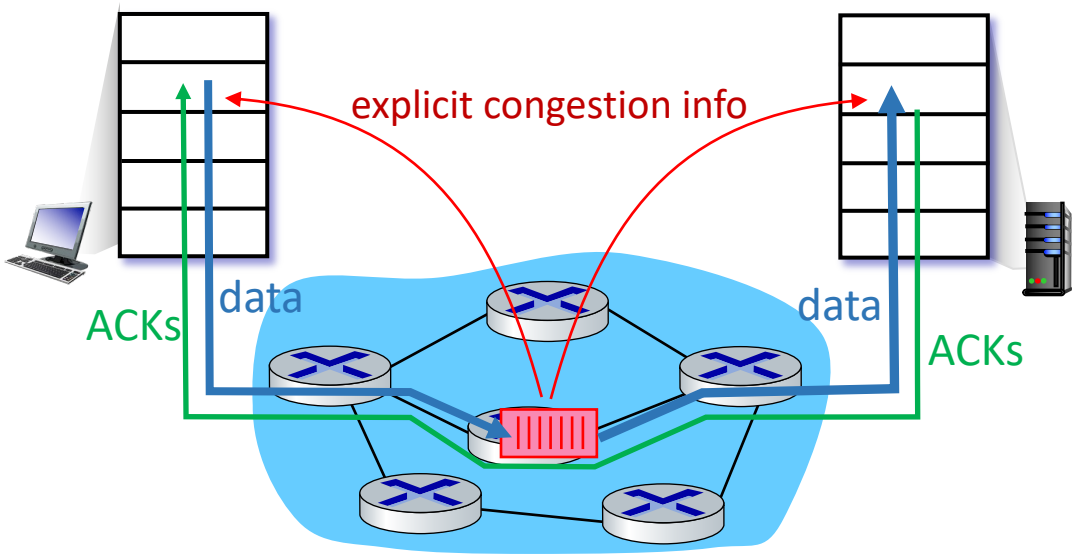
- no explicit feedback from network
- congestion *inferred* from observed loss, delay
  - approach taken by TCP



Approaches towards congestion control

Network-assisted congestion control:

- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate
- TCP ECN, ATM, DECbit protocols



TCP congestion control: AIMD

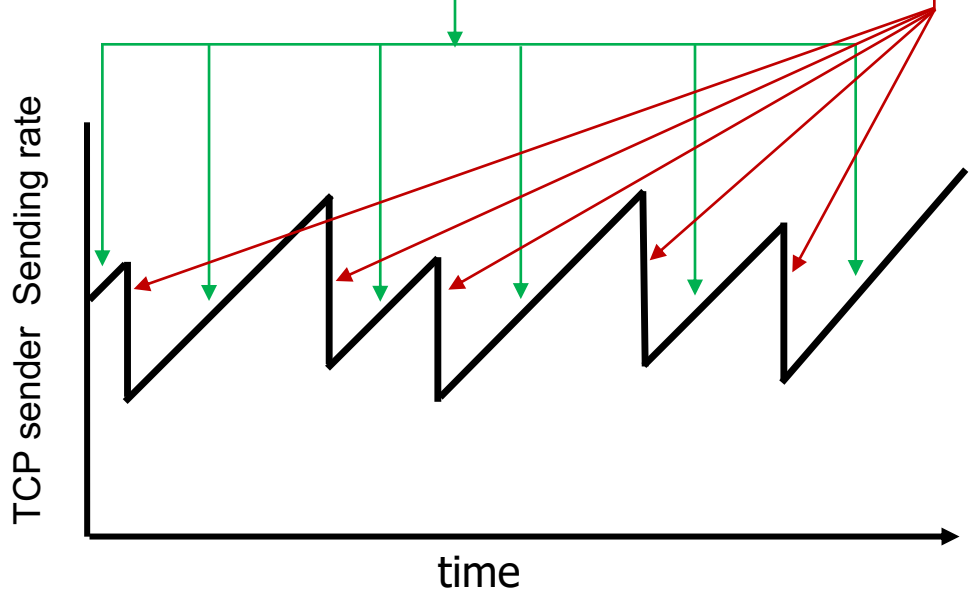
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

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

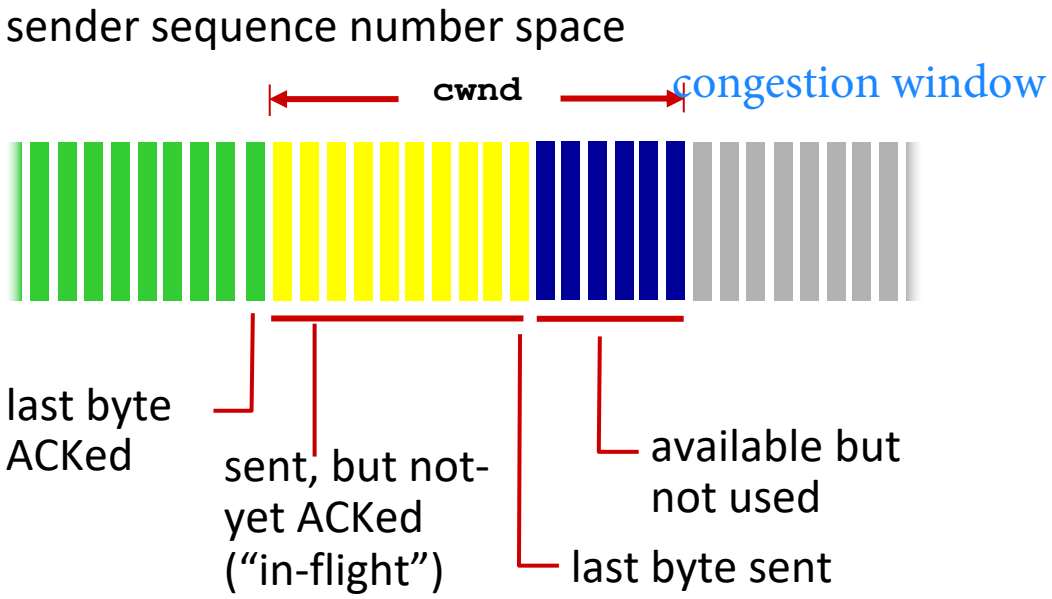
when three dup ack arrives for same degment of data

- 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 congestion control: details



TCP sending behavior:

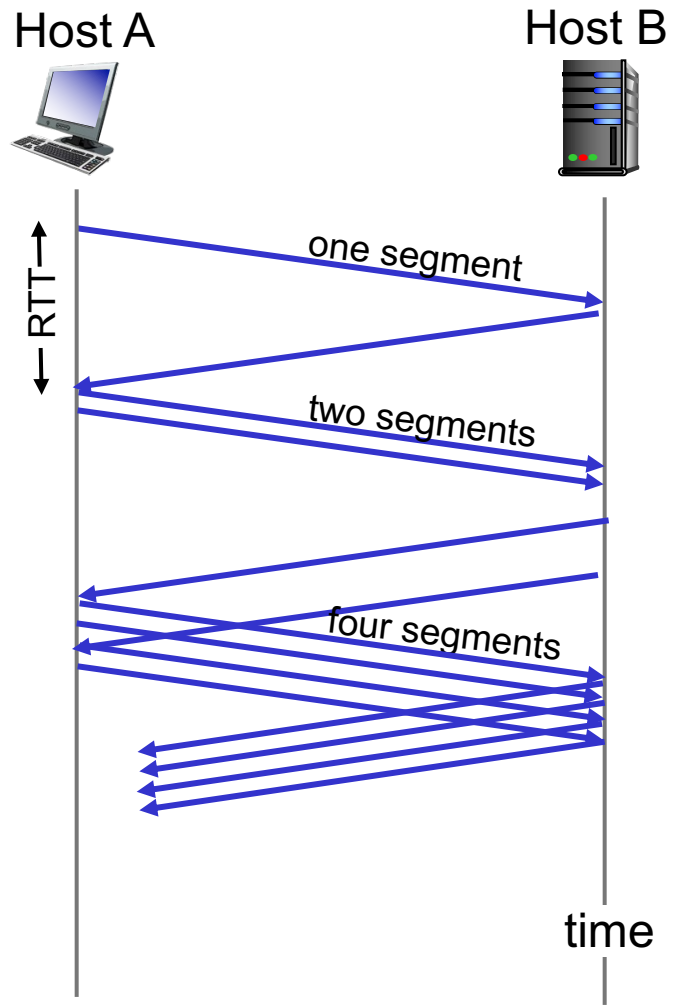
- *roughly:* send `cwnd` bytes, wait RTT for ACKS, then send more bytes

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

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

TCP slow start

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



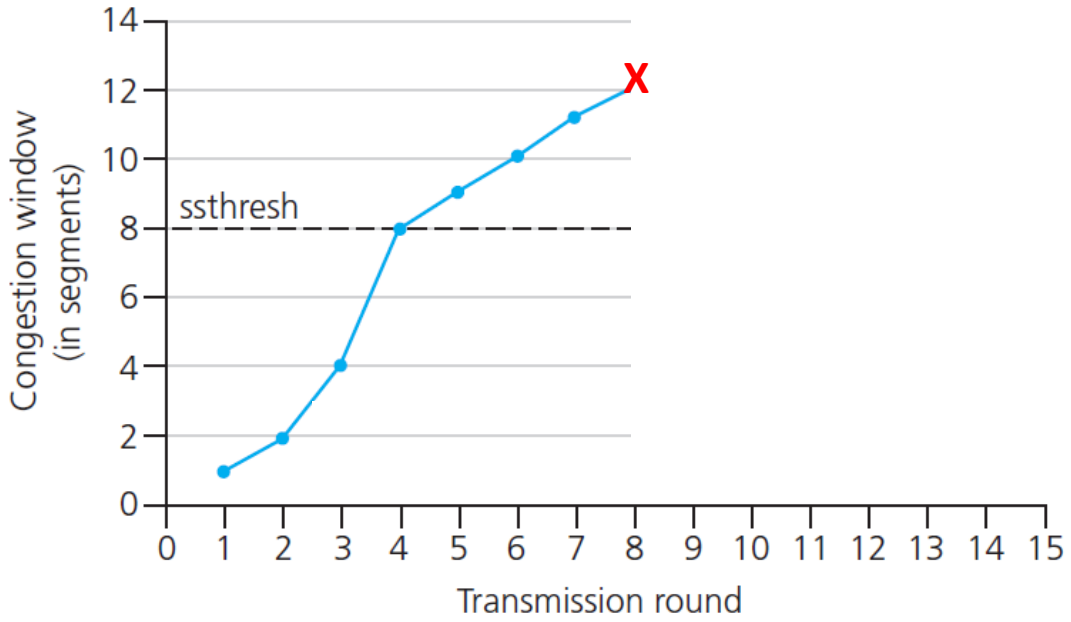
TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

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

Implementation:

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



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

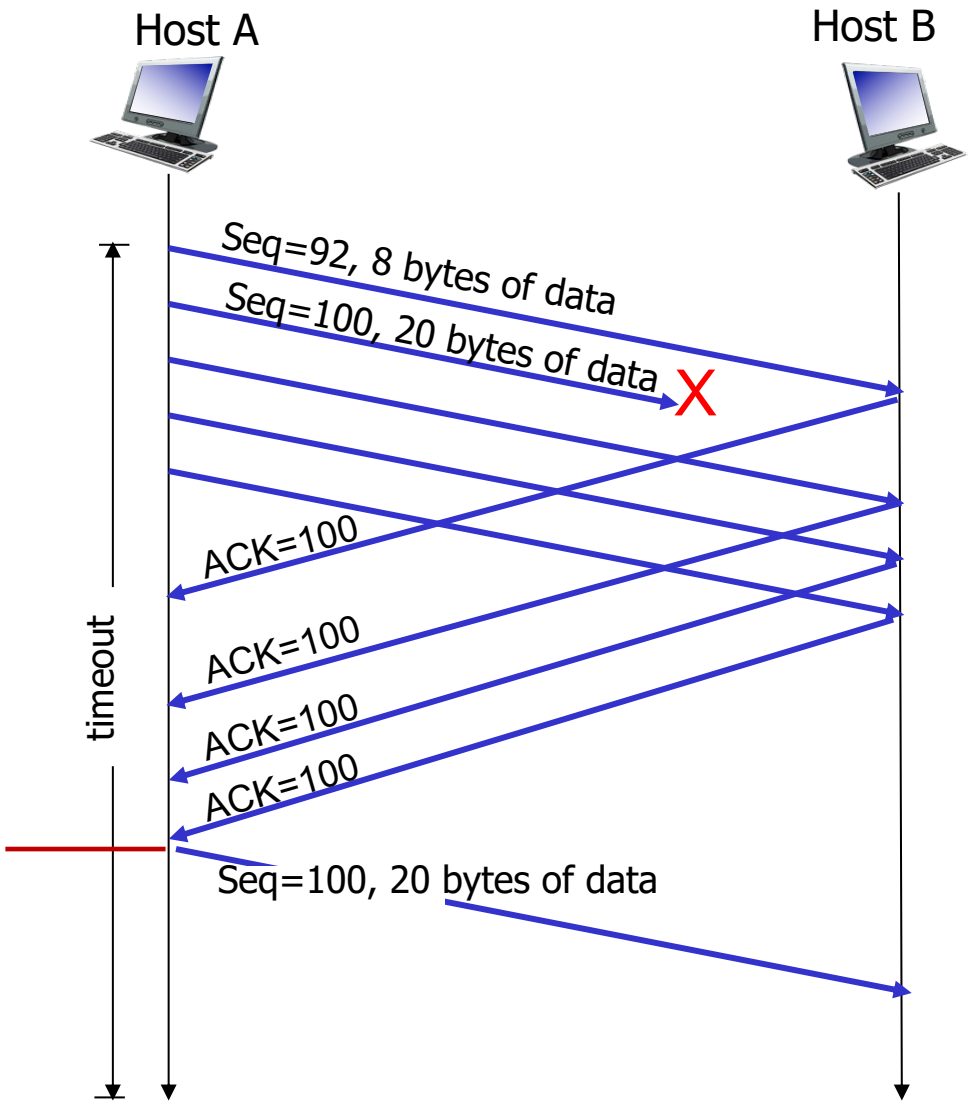
TCP fast retransmit

TCP fast retransmit

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

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

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



TCP round trip time, timeout

Q: how to set TCP timeout value?

- longer than RTT, but RTT varies!
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

Q: how to estimate RTT?

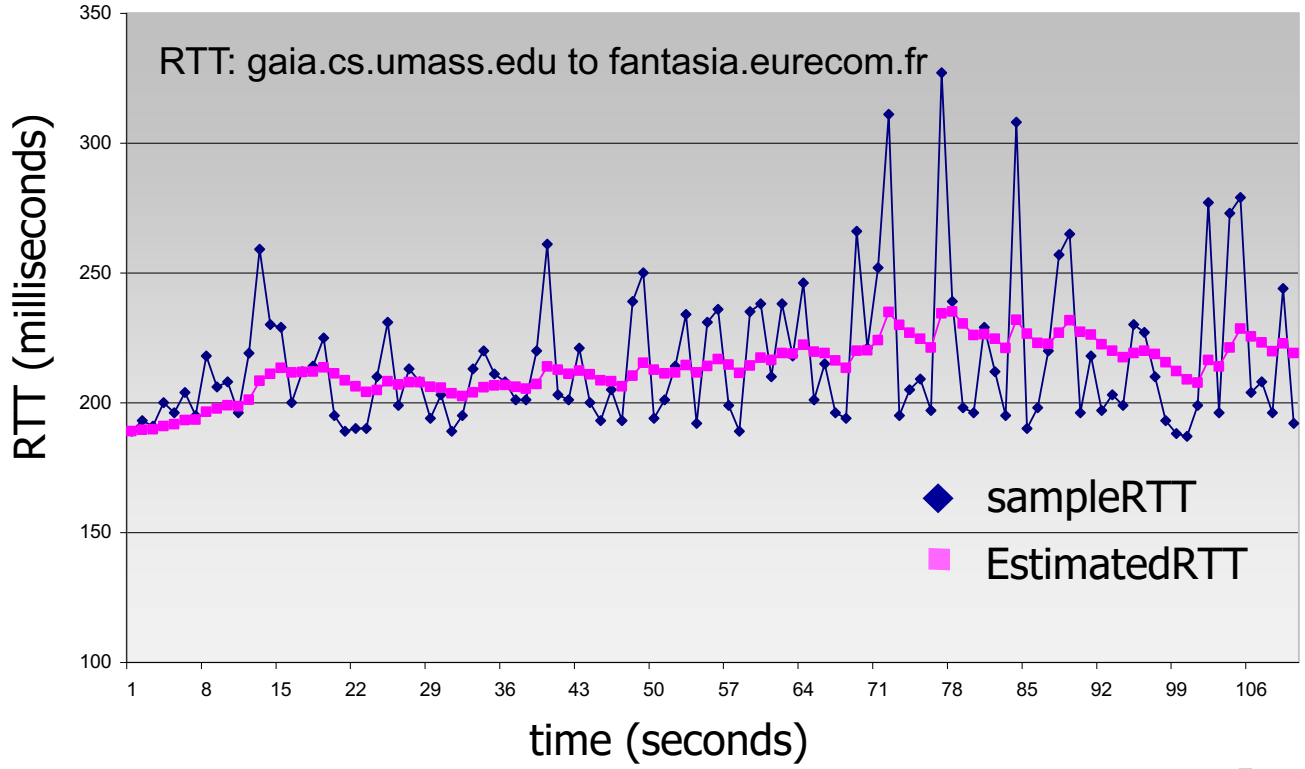
- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current `SampleRTT`

It is possible that after a very short time, we receive the ACK for the retransmitted data segment. But that could actually be a delayed ACK for the original TCP segment. Either the data transmission or the ACK reply was probably delayed. Both of them could result in a late ACK. If we count this time in to compute the RTT estimation, we would mistakenly drop the RTT average by a big percentage. This RTT would trigger a faster retransmission, which would worsen the already jammed network traffic.

TCP round trip time, timeout

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

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



TCP round trip time, timeout

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

TCP Sender (simplified)

event: data received from application

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unACKed segment
  - expiration interval: **`TimeOutInterval`**

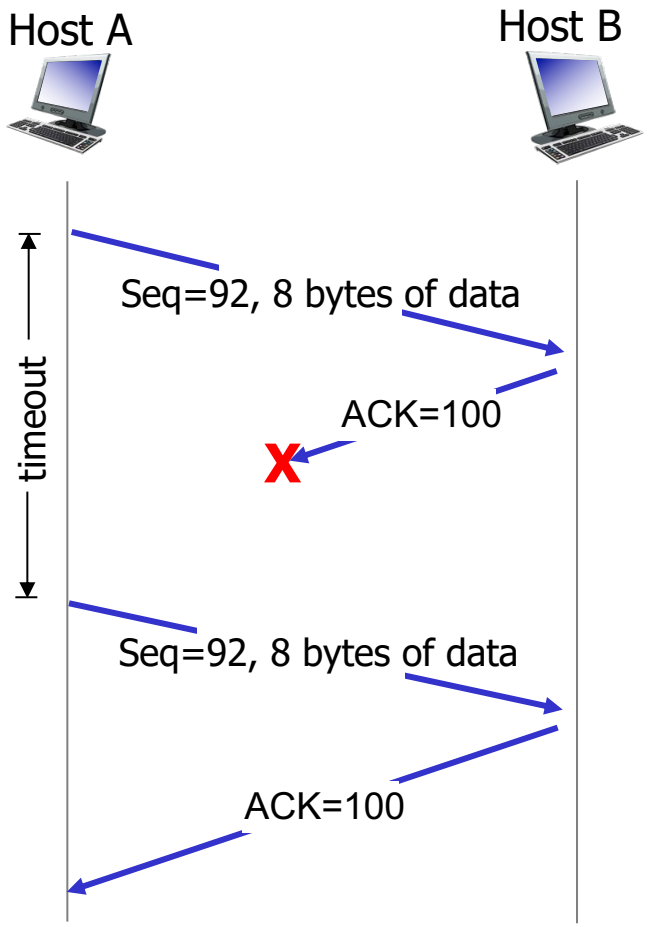
event: timeout

- retransmit segment that caused timeout
- restart timer

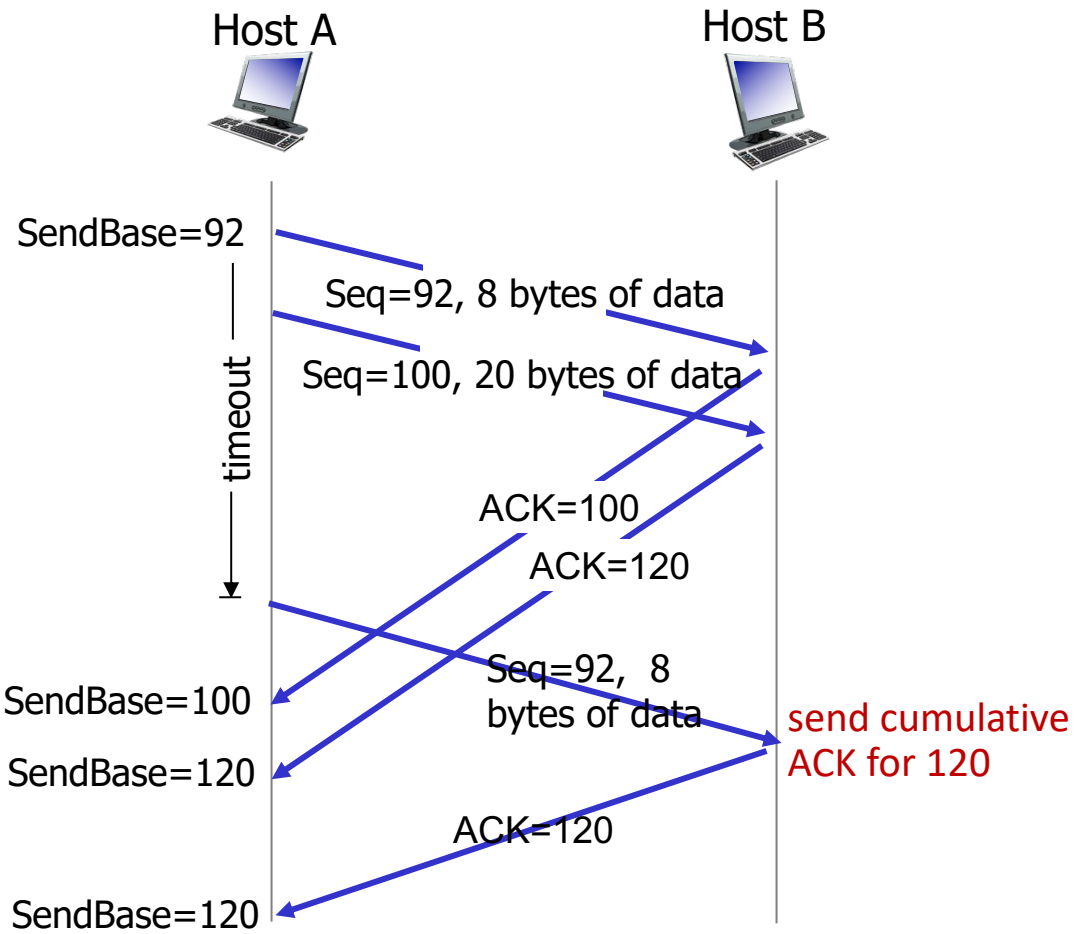
event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

TCP: retransmission scenarios

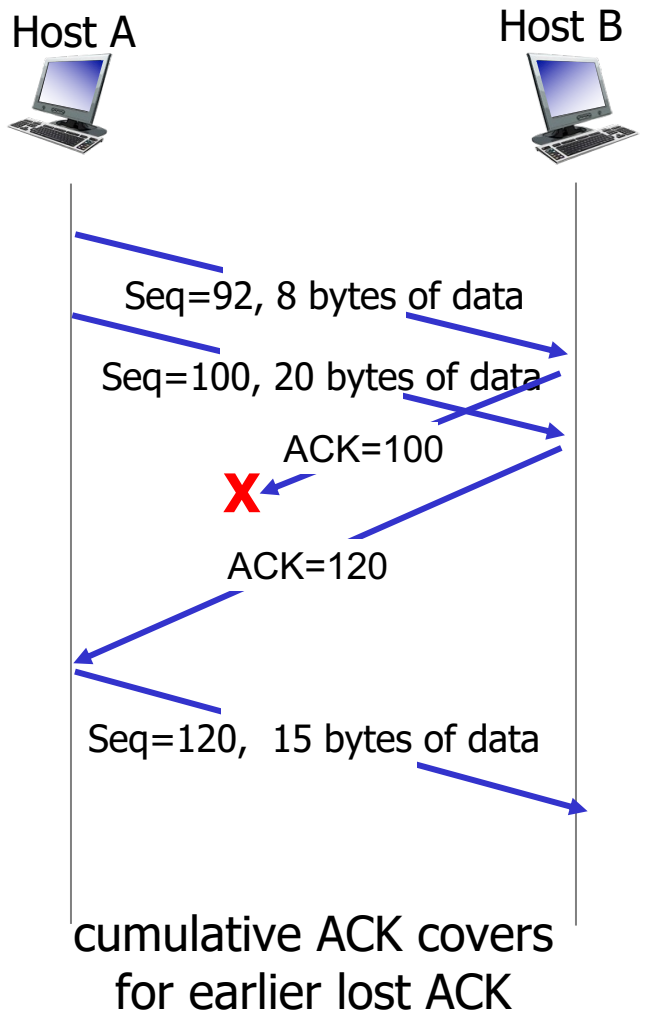


lost ACK scenario



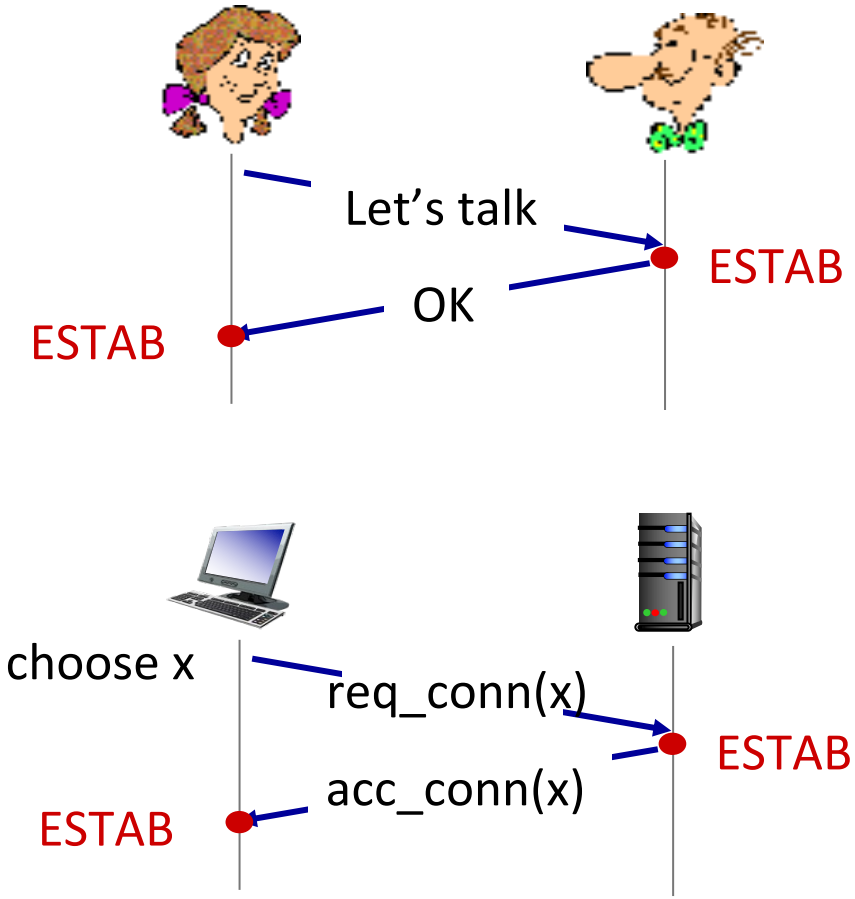
premature timeout

TCP: retransmission scenarios



Agreeing to establish a connection

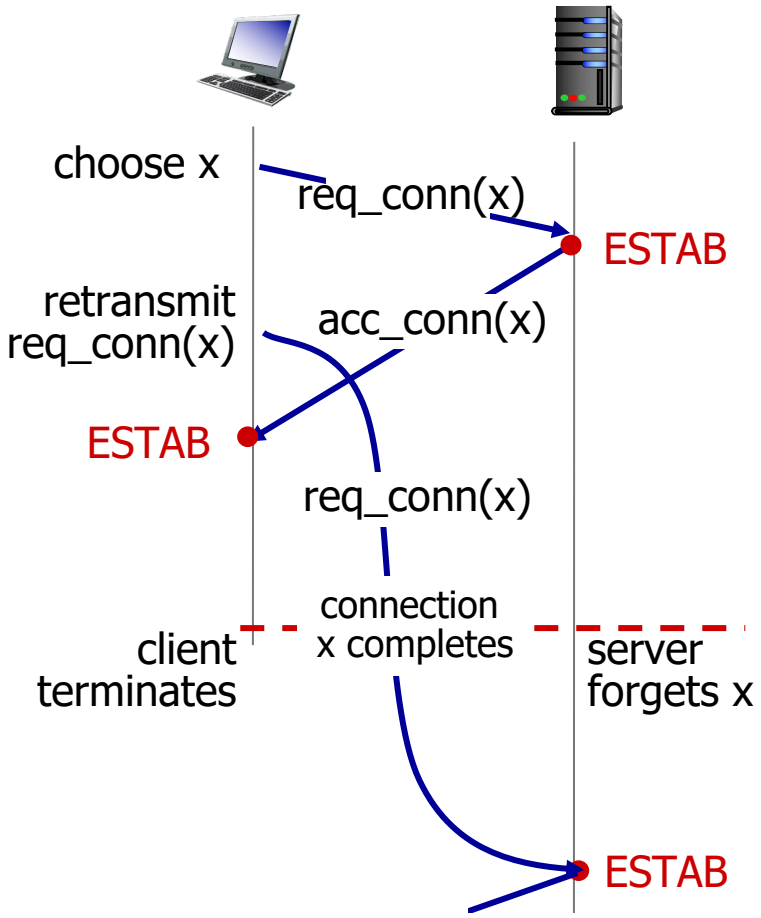
2-way handshake:



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

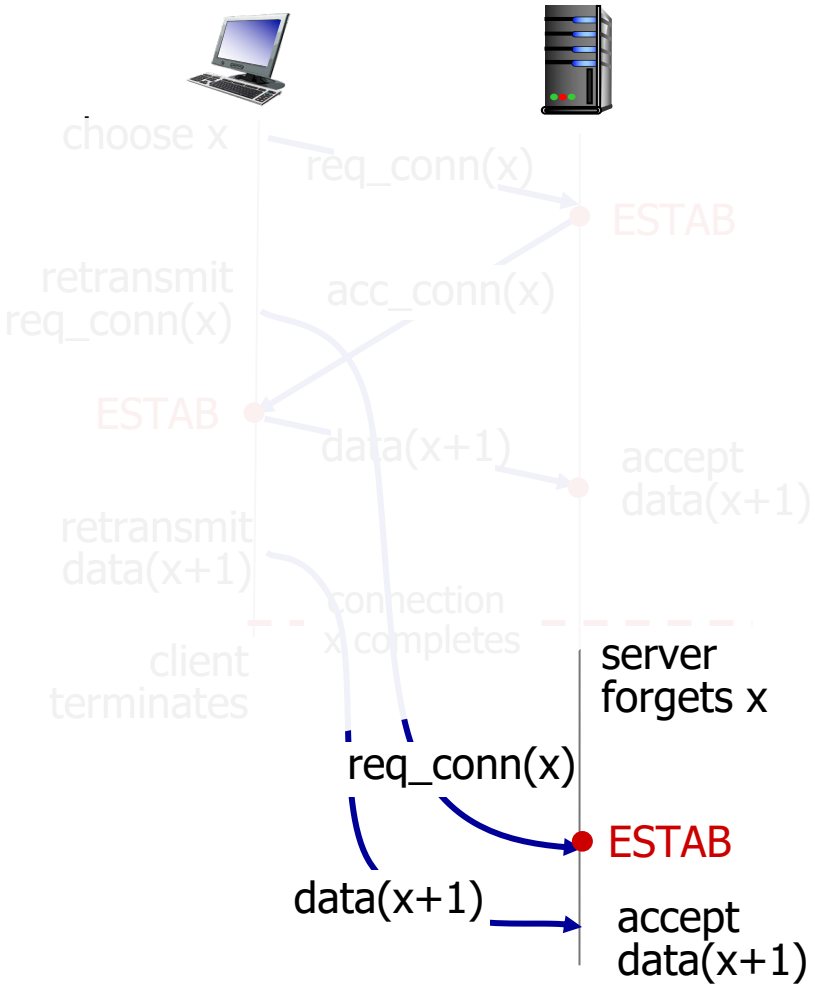
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

2-way handshake scenarios



Problem: half open connection! (no client)

2-way handshake scenarios



Problem: dup data accepted!