Introduction to Computer Network

Previously in Chapter 1

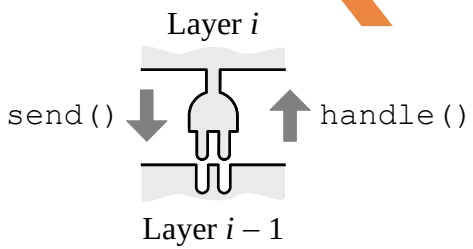
Protocols

Basically, a protocol is an **agreement** between the communicating **peers** on how communication is to proceed.

Protocols

- Protocol defines the interfaces between the layers in the same system and with the layers of peer system
- Building blocks of a network architecture
- Each protocol object has two different interfaces
  - service interface: operations on this protocol
  - peer-to-peer interface: messages exchanged with peer
- Term “protocol” is overloaded
  - specification of peer-to-peer interface
  - module that implements this interface

Protocols



User-to-user interactions obey social norms

Letter (Message)

Person A

Person B

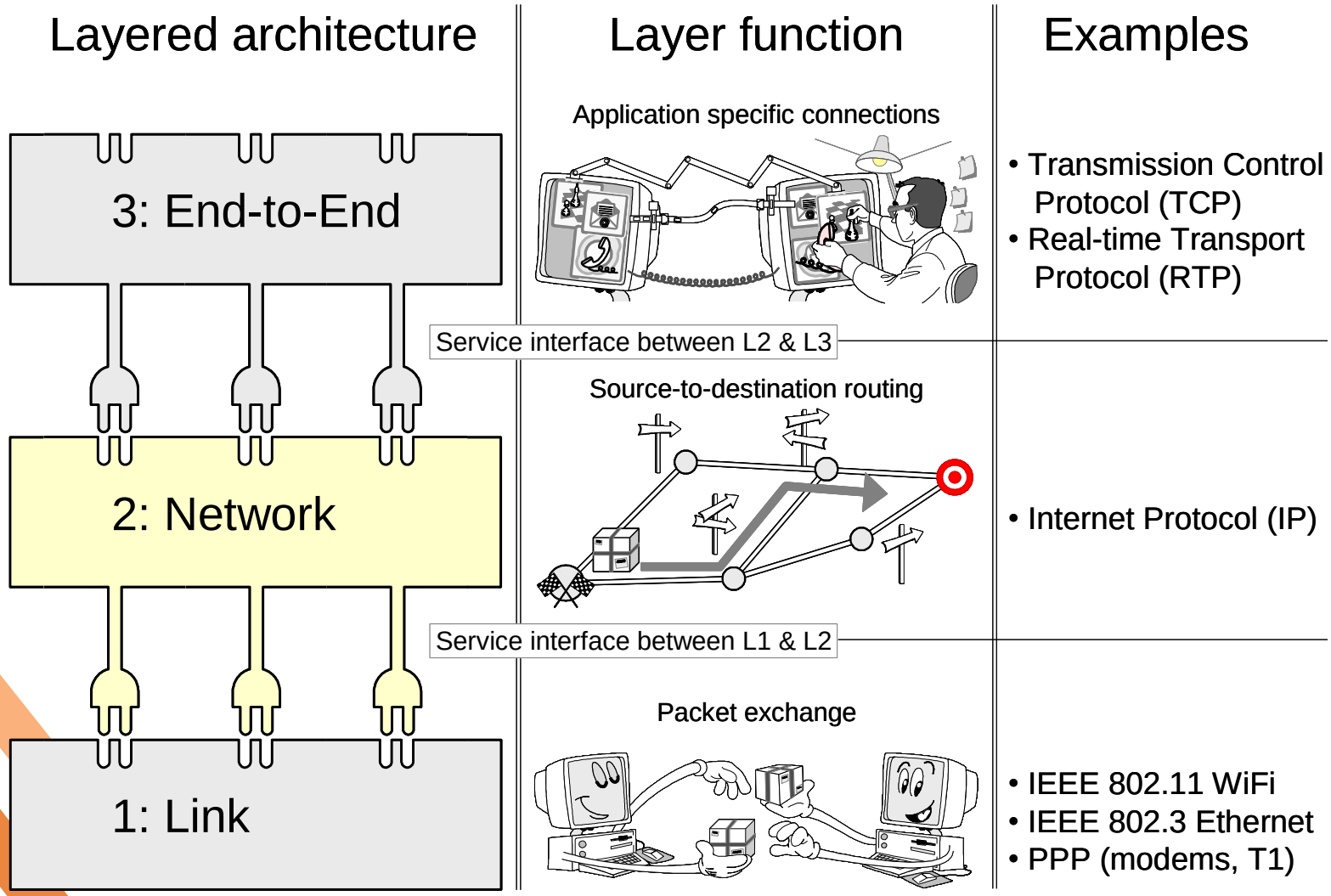
Customer interaction obeys mail acceptance and delivery procedures (Postal Service's Mail Manual)

Letter in envelope (Packet)

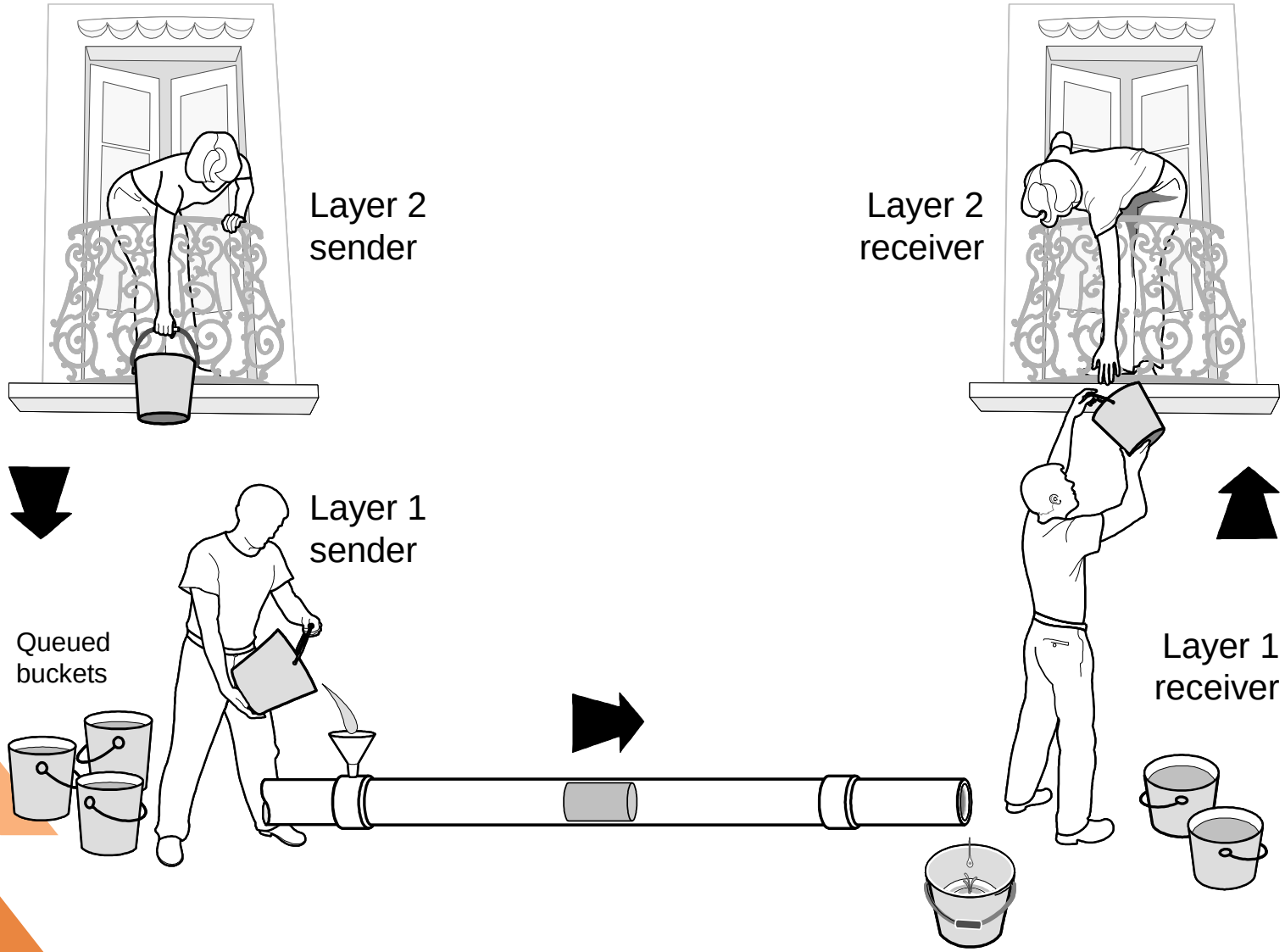
Postal-vehicle service-transportation routes obey carrier-route maps and delivery timetables

Physical transport obeys transportation and traffic rules

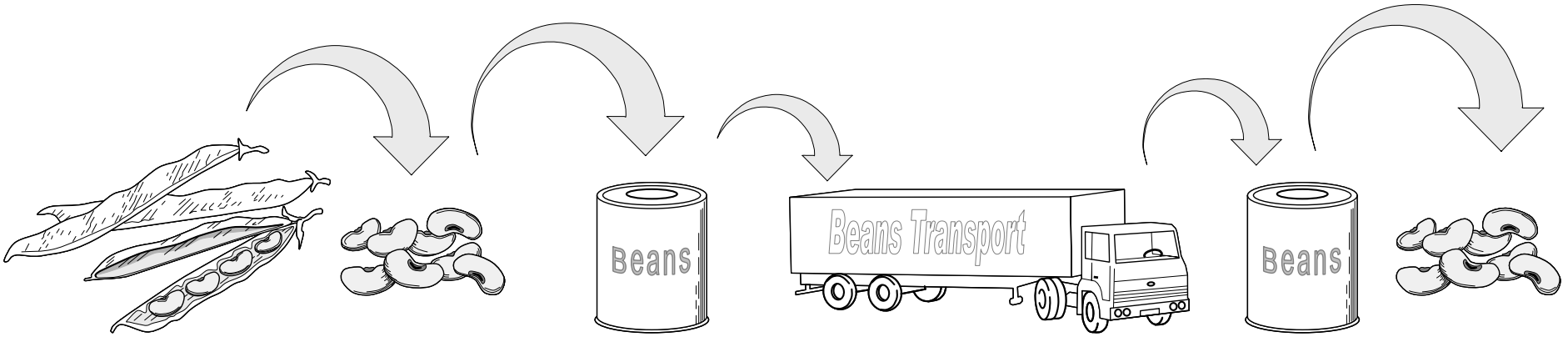
3-Layer Protocol Stack



Fluid Flow Analogy



Source Coding vs. Channel Coding



Beans
(Redundancy = pods)

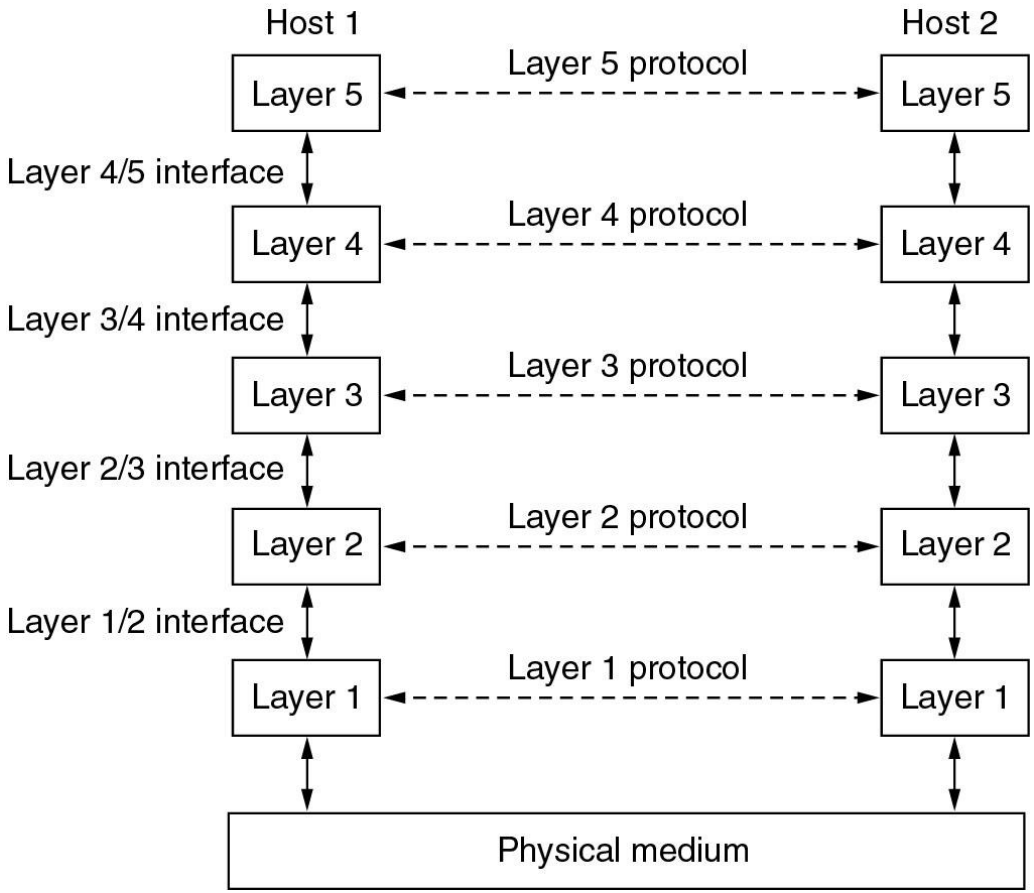
Beans
(No redundancy)

Beans
(Redundancy = can)

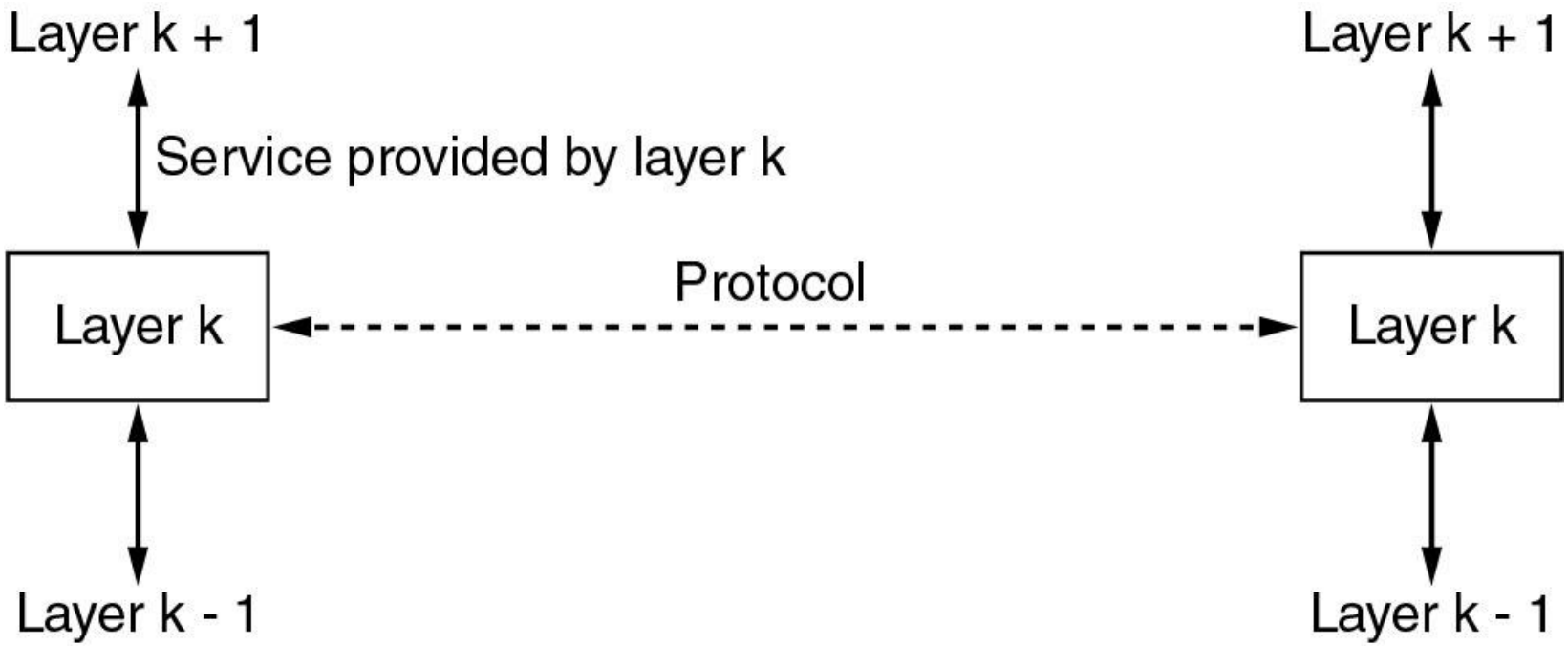
Erroneous
Communication Channel

Beans
(No redundancy)

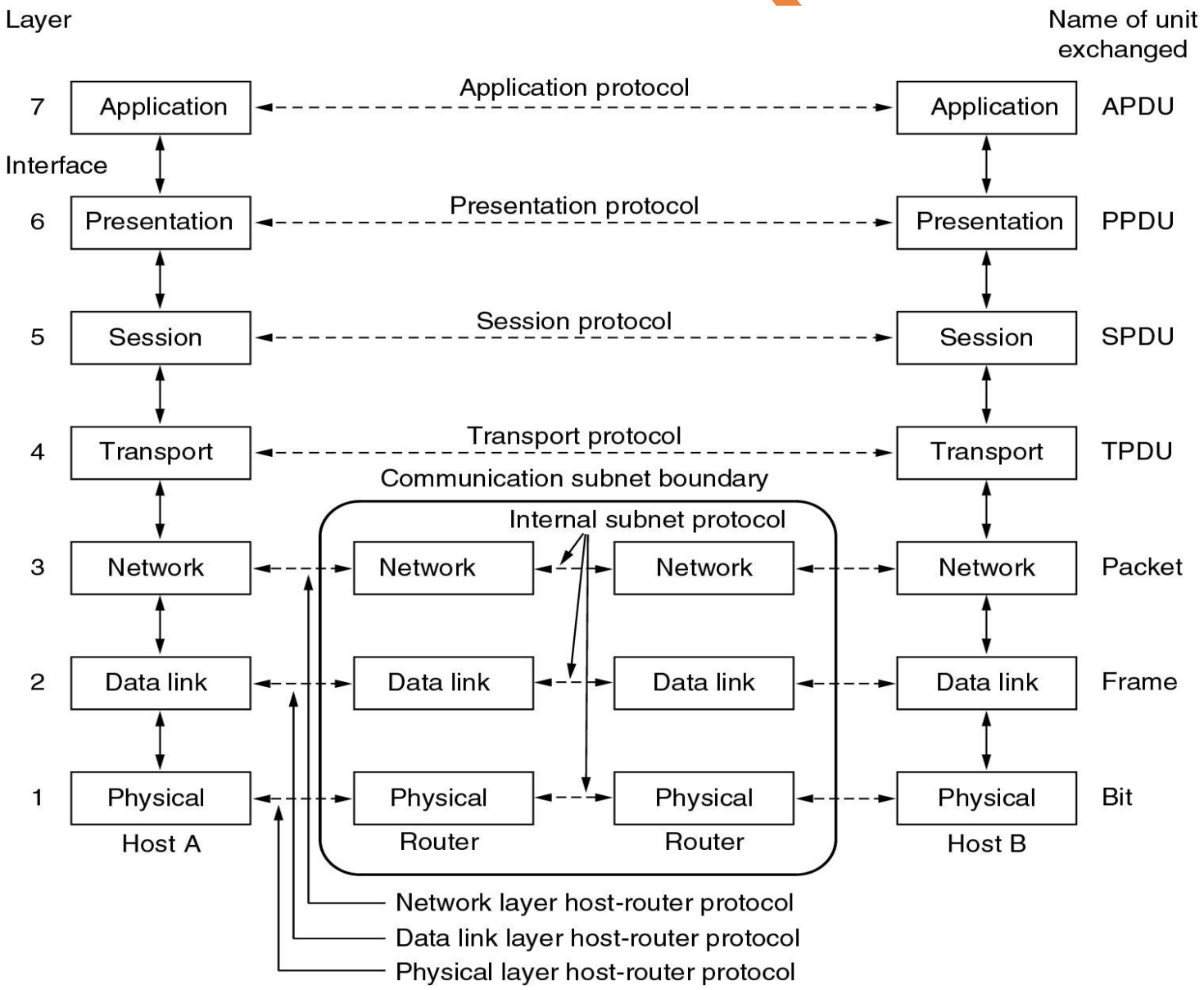
The layering Principle



The relationship of Service



The OSI Reference Model



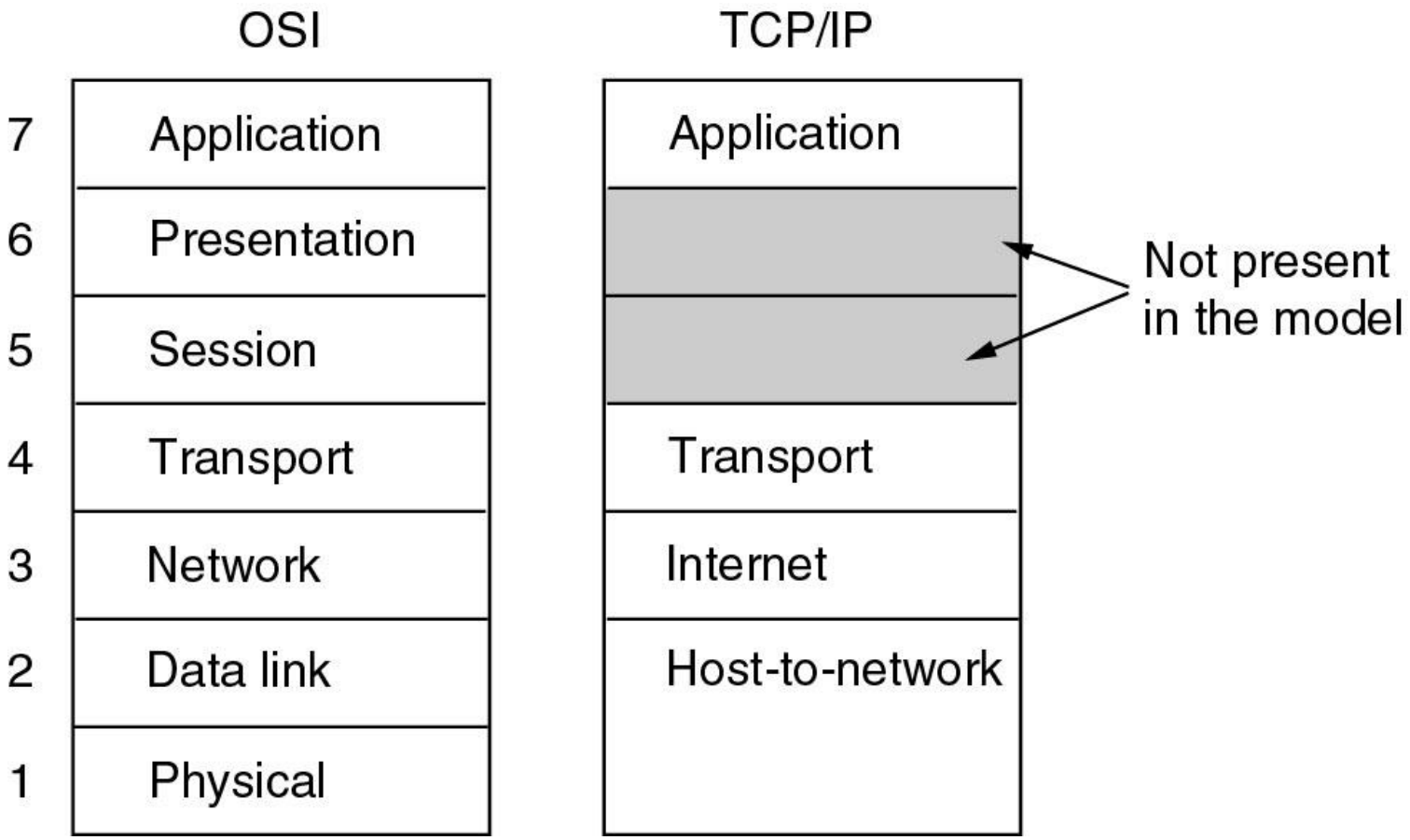
7 Layer

- Layer 1: Physical Layer
  - ทำหน้าที่เชื่อมต่อผ่าน Physical Medium รับผิดชอบแปลงบิตเป็นสัญญาณ เรื่องของการ Interface, สายนำสัญญาณ ,มองเห็นข้อมูลในลักษณะ Bit Stream
- Layer 2: Data Link Layer
  - ประกอบข้อมูลเป็น Frame, รับผิดชอบในการสื่อสารผ่าน แต่ละ Link ทำ Error Control, Flow Control ผ่าน Link
- Layer 3: Network Layer
  - รับผิดชอบในการส่งข้อมูลผ่าน Network, หาทิศทางข้อมูล, เชื่อมต่อกับ Layer บนเข้ากับ Network หลายๆแบบ มองเห็นข้อมูลในลักษณะ Packet

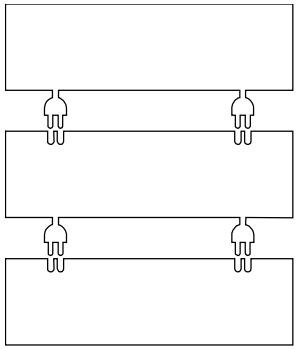
7 Layer

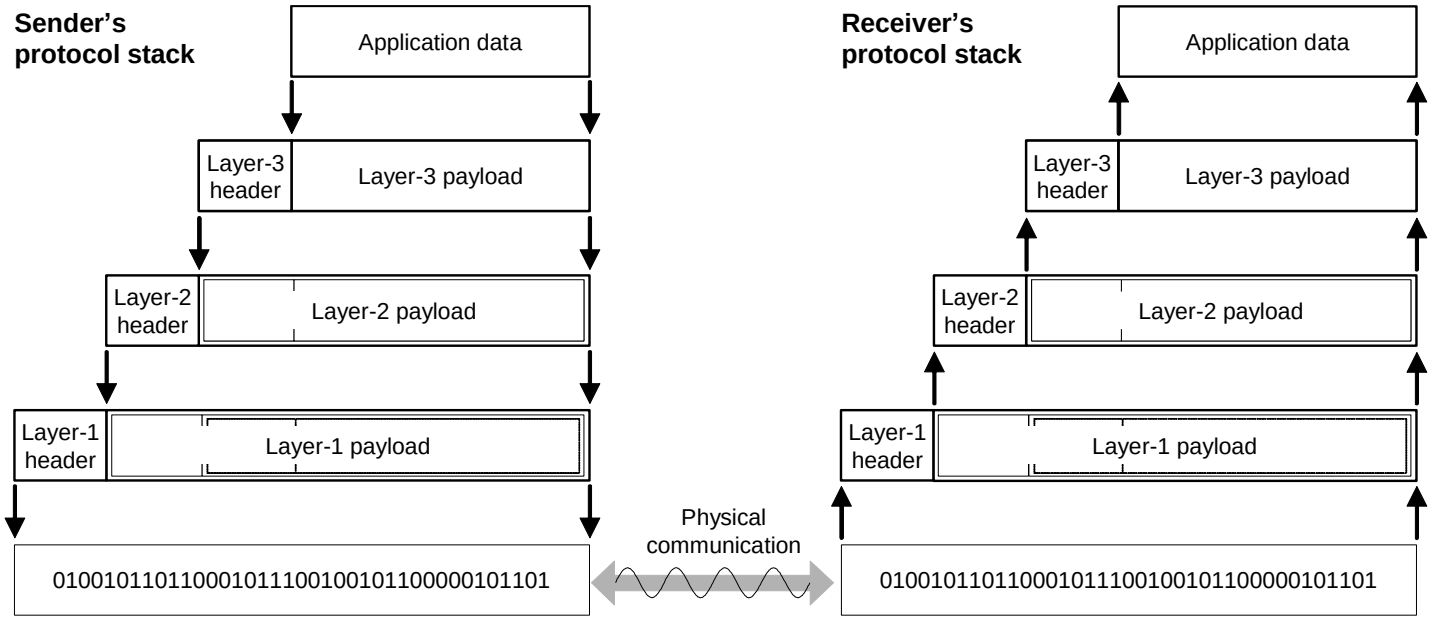
- Layer 4: Transport Layer
  - รับผิดชอบการส่งข้อมูลให้ถูกต้องจากต้นทางถึงปลายทาง(End-to-End), จัดการในเรื่อง Error และ Flow Control ในระดับต้นทางถึงปลายทาง ข้อมูลที่ส่งจะถูกแบ่งเป็น Segment
- Layer 5: Session Layer
  - ทำหน้าที่จัดตั้ง ดูแล การเชื่อมต่อ(Connection) ระหว่าง Applicationต้นทางและปลายทาง แบ่งการเชื่อมต่อสื่อสารออกเป็น Session
- Layer 6: Presentation Layer
  - รับผิดชอบในเรื่องรูปแบบและ Format ของข้อมูล การทำ Encryption รวมถึงการทำ Data Compression ให้อยู่ในรูปแบบที่สื่อสารได้
- Layer 7: Application Layer
  - ทำหน้าที่เชื่อมต่อกับ Application และผู้ใช้

The TCP/IP Reference Model

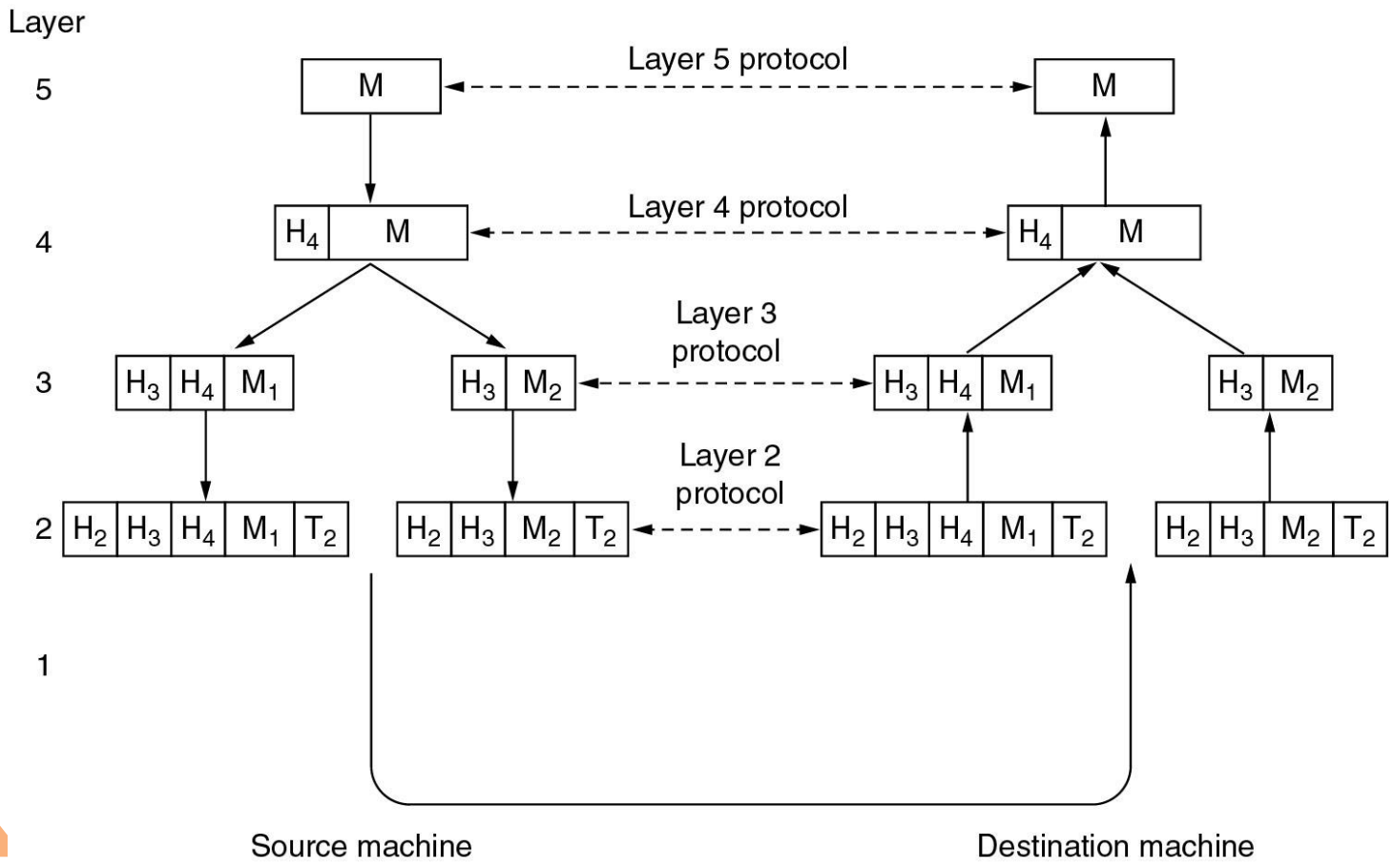


Encapsulation

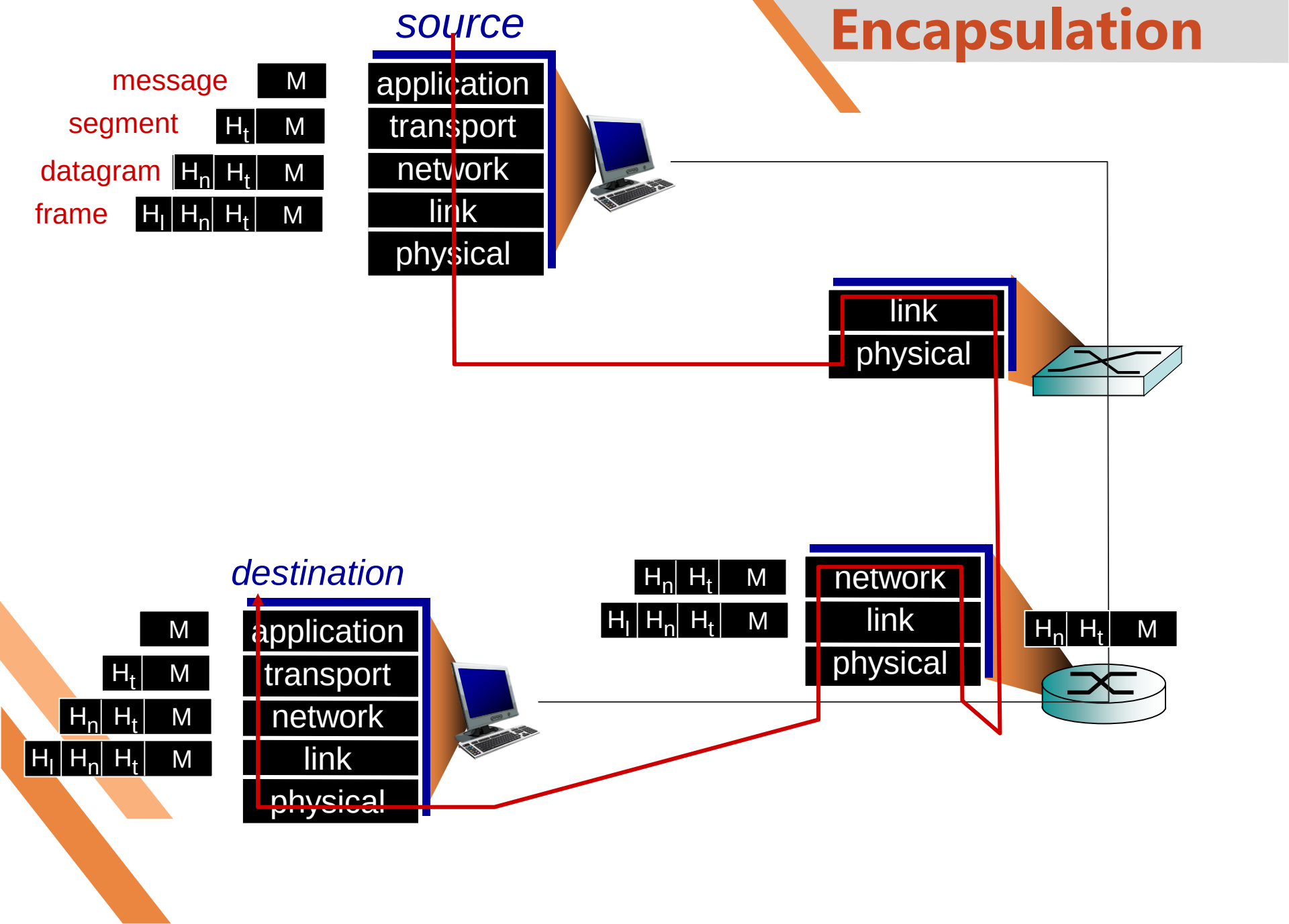




Encapsulation



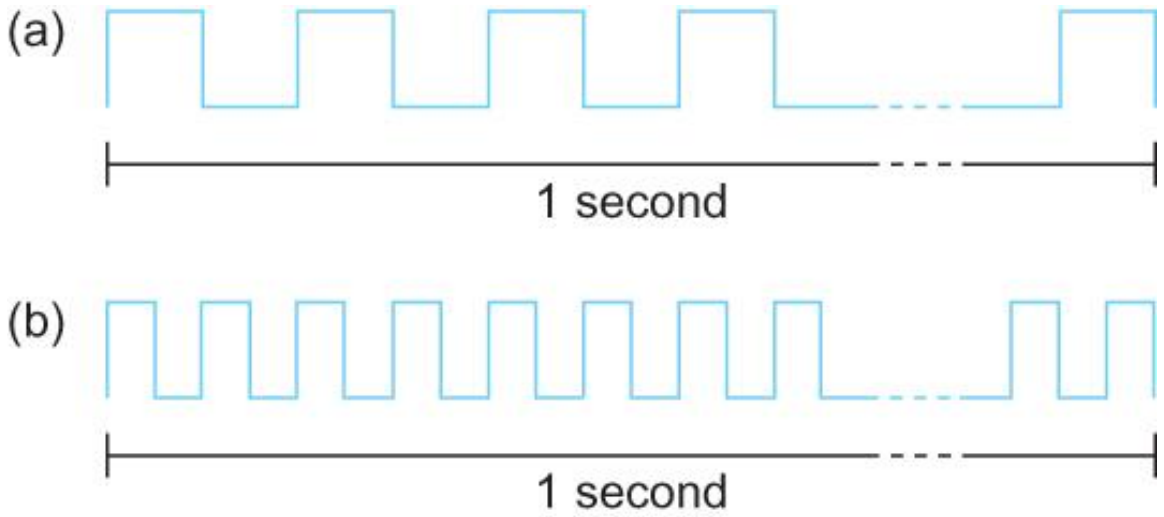
Encapsulation



Performance

- Bandwidth
  - Width of the frequency band
  - Number of bits per second that can be transmitted over a communication link
- 1 Mbps: 1×10^6 bits/second = $1x2^{20}$ bits/sec
- 1×10^{-6} seconds to transmit each bit or imagine that a timeline, now each bit occupies 1 micro second space.
- On a 2 Mbps link the width is 0.5 micro second.
- Smaller the width more will be transmission per unit time.

Bandwidth

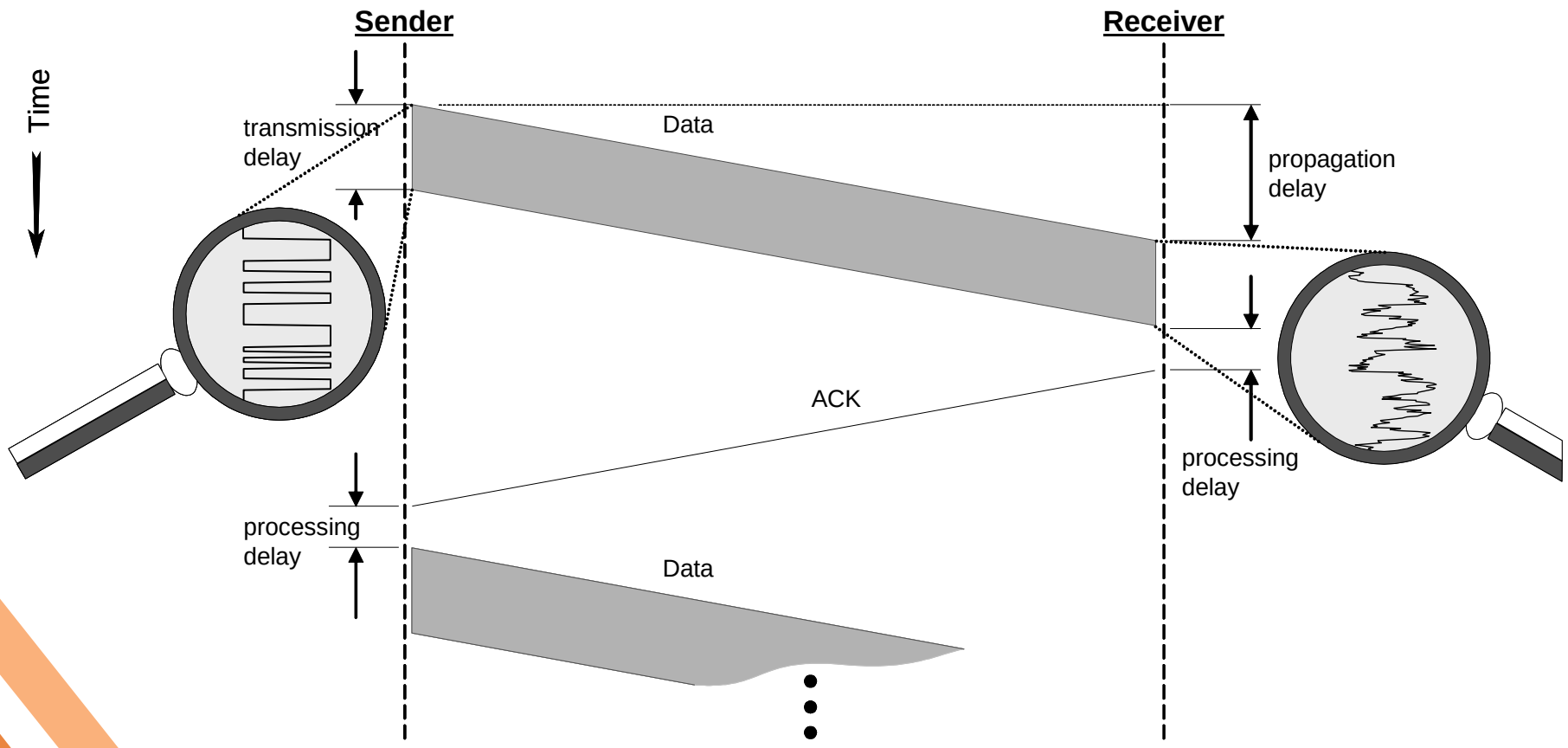


Bits transmitted at a particular bandwidth can be regarded as having some width:

(a) bits transmitted at 1Mbps (each bit 1 μs wide);

(b) bits transmitted at 2Mbps (each bit 0.5 μs wide).

Packet Transmission



Performance

- $Latency = Propagation + Transmit + queue$
- $Propagation = distance/speed\ of\ light$
- $Transmit = size/bandwidth$

- One bit transmission => propagation is important
- Large bytes transmission => bandwidth is important

Delay X Bandwidth

- We think the channel between a pair of processes as a hollow pipe
- Latency (delay) length of the pipe and bandwidth the width of the pipe
- Delay of 50 ms and bandwidth of 45 Mbps

$\Rightarrow$ 50×10^{-3} seconds x 45×10^{6} bits/second

$\Rightarrow$ 2.25×10^{6} bits = 280 KB data.

Network as a pipe

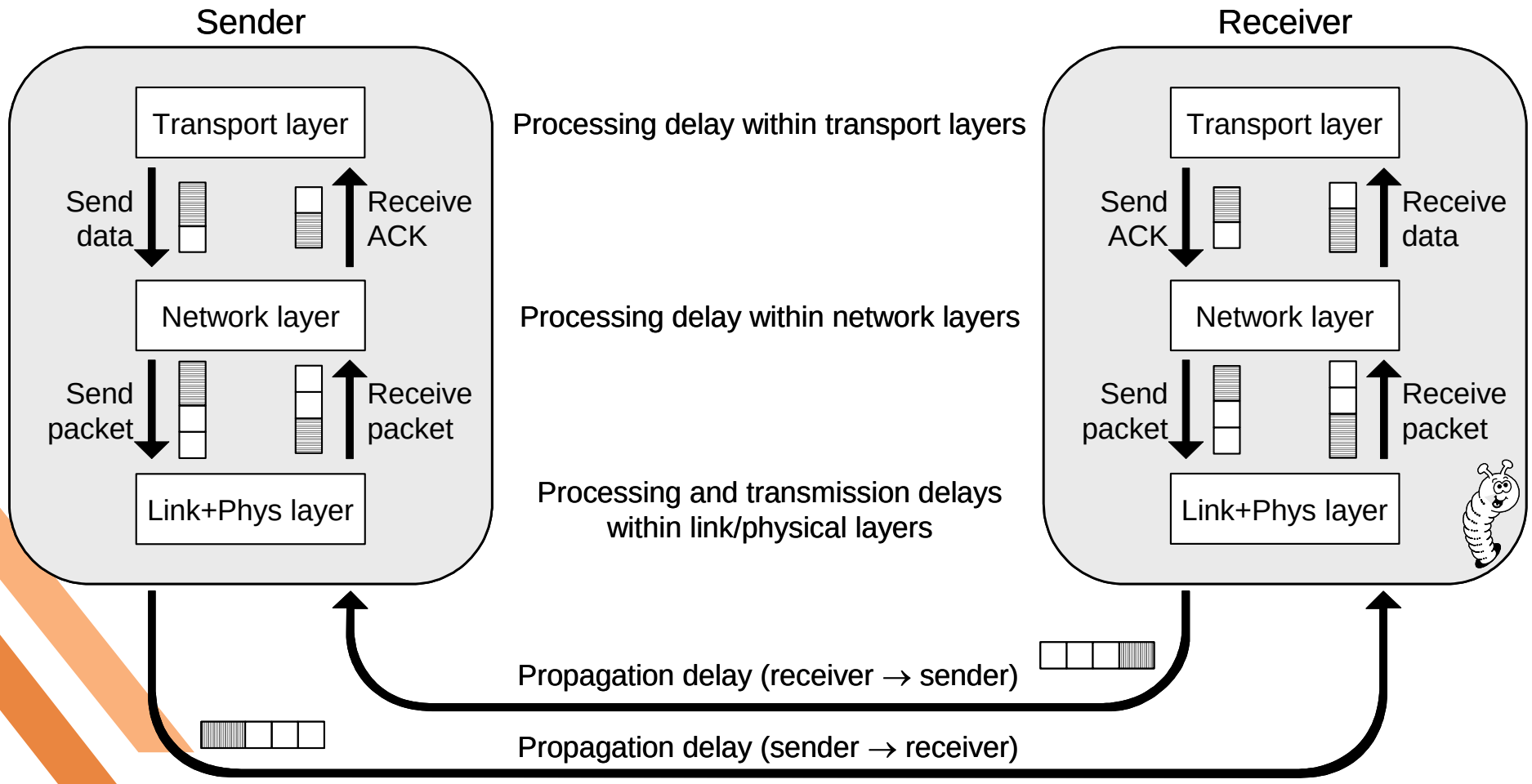
Delay X Bandwidth

- Relative importance of bandwidth and latency depends on application
  - For large file transfer, bandwidth is critical
  - For small messages (HTTP, NFS, etc.), latency is critical
  - Variance in latency (jitter) can also affect some applications (*e.g.*, audio/video conferencing)

Delay X Bandwidth

- How many bits the sender must transmit before the first bit arrives at the receiver if the sender keeps the pipe full
- Takes another one-way latency to receive a response from the receiver
- If the sender does not fill the pipe—send a whole delay × bandwidth product's worth of data before it stops to wait for a signal—the sender will not fully utilize the network

What Contributes to RTT



การบ้าน ฝึกใช้คำสั่ง Ping

การใช้คำสั่ง Ping ตรวจสอบการเชื่อมต่อเครือข่าย

คำสั่ง Ping เป็นคำสั่งที่ใช้ในการตรวจสอบการเชื่อมต่อกับเครือข่ายระหว่างคอมพิวเตอร์แต่ละเครื่องที่อยู่ในเครือข่าย โดยคำสั่ง Ping จะส่งข้อมูลที่เป็นแพ็คเกจ 4 ชุดๆละ 32 Byte ไปยังคอมพิวเตอร์ปลายทางที่ต้องการตรวจสอบ หากมีการตอบรับกลับมาจากคอมพิวเตอร์เป้าหมายก็แสดงว่าการเชื่อมต่อเครือข่ายยังเป็นปกติ แต่หากไม่มีการตอบรับกลับมาก็แสดงว่าคอมพิวเตอร์ปลายทางหรือเครือข่ายอยู่ในช่วงหนาแน่น ดังนั้นจะเห็นว่าคำสั่ง Ping มีประโยชน์อย่างมากในการตรวจสอบสถานะการเชื่อมต่อเครือข่ายเบื้องต้นได้เป็นอย่างดี

งานเดี่ยว ส่งพร้อม Proxy

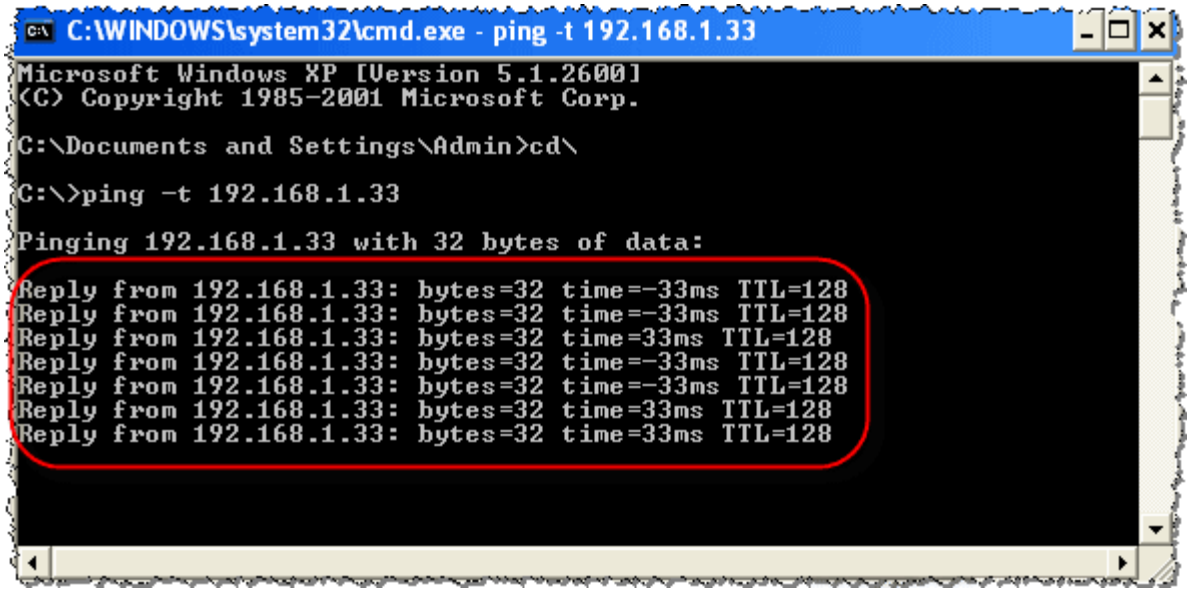
การบ้าน ฝึกใช้คำสั่ง Ping

ขั้นตอนการเรียกใช้งานมีดังนี้

หน้าต่าง Command Prompt ให้พิมพ์คำสั่ง ping ตามด้วยหมายเลข IP Address ของเครื่องคอมพิวเตอร์ที่ต้องการเข้าไปตรวจสอบลงไป จากนั้นกดปุ่ม Enter

หากมีการตอบรับกลับมาจากคอมพิวเตอร์ปลายทาง จะปรากฏคำสั่งเหมือนในกรอบสีแดง แสดงว่าคอมพิวเตอร์ทั้ง 2 เครื่องสามารถติดต่อสื่อสารกันได้ตามปกติ

แต่ถ้าปรากฏคำสั่ง “**Request timed out**” นั่นแสดงว่าคอมพิวเตอร์ทั้ง 2 เกิดปัญหาขัดข้องไม่สามารถติดต่อสื่อสารถึงกันได้ ซึ่งจะต้องทำการตรวจสอบการเชื่อมต่อเครือข่ายรวมถึงการตั้งค่าต่างๆให้ถูกต้อง แล้วลองใช้คำสั่ง Ping ตรวจสอบอีกครั้ง



การบ้าน ฝึกใช้คำสั่ง Ping

ตัวเลือกเพิ่มเติมที่นิยมใช้ร่วมกันกับคำสั่ง Ping

Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS] [-r count] [-s count] [[-j host-list] | [-k host-list] | [-w timeout] target_name

Options:

-t Ping ไปยัง Host ตามที่ระบุเรื่อยๆ จนกว่าจะสั่งยกเลิกโดยกดแป้น Ctrl-C.และหากต้องการดูสถิติให้กดแป้น Ctrl-Break

-a เปลี่ยนหมายเลข IP Address ของ Host เป็นชื่อแบบตัวอักษร

-n count Ping แบบระบุจำนวน echo ที่จะส่ง

-l size กำหนดขนาด buffer

-f ตั้งค่าไม่ให้แยก flag ใน packet.

-i TTL Ping แบบกำหนด Time To Live โดยกำหนดค่าตั้งแต่ 1-255

-v TOS กำหนดประเภทของบริการ (Type of service)

-r count Ping แบบให้มีการบันทึกเส้นทางและนับจำนวนครั้งในการ hops จนกว่าจะถึงปลายทาง

-s count Ping แบบนับเวลาในการ hop แต่ละครั้ง

-j host-list Loose source route along host-list.

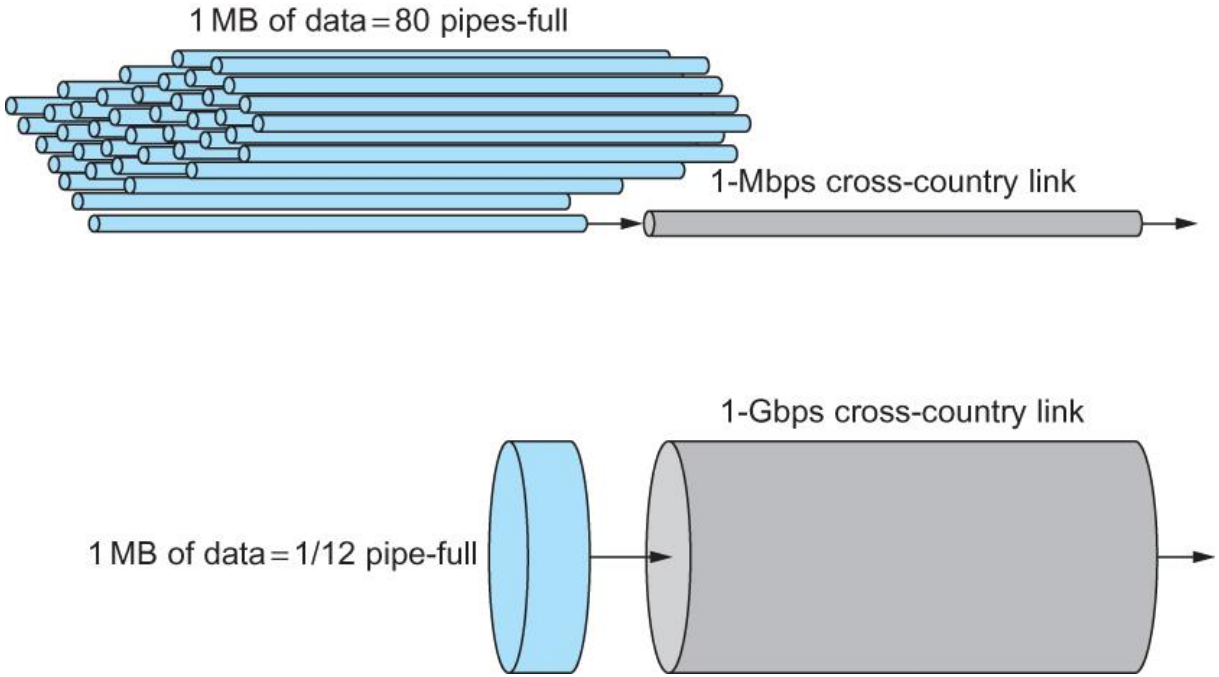
-k host-list Strict source route along host-list.

-w timeout Ping แบบกำหนดเวลารอคอยการตอบรับ

Delay X Bandwidth

- Infinite bandwidth
  - RTT (Round Trip Time) dominates
  - $Throughput = TransferSize / TransferTime$
  - $TransferTime = RTT + 1/Bandwidth \; x \; TransferSize$
- Its all relative
  - 1-MB file to 1-Gbps link looks like a 1-KB packet to 1-Mbps link

Relationship between bandwidth and latency



A 1-MB file would fill the 1-Mbps link 80 times, but only fill the 1-Gbps link 1/12 of one time

Summary

- We have identified what we expect from a computer network
- We have defined a layered architecture for computer network that will serve as a blueprint for our design
- We have discussed the socket interface which will be used by applications for invoking the services of the network subsystem
- We have discussed two performance metrics using which we can analyze the performance of computer networks

Introduction to Computer Network

Chapter 02: Applications

Textbooks

- Computer Networks, Sixth Edition, Andrew S. Tanenbaum :: Chapter 6
- Computer Networks: A Systems Approach, 5e , Larry L. Peterson and Bruce S. Davie :: Chapter 9
- **Computer Networking: A Top-Down Approach 6e, J.F. Kurose and K.W. Ross :: Chapter 2**

Problem

- Applications need their own protocols.
- These applications are part network protocol (in the sense that they exchange messages with their peers on other machines) and part traditional application program (in the sense that they interact with the windowing system, the file system, and ultimately, the user).
- This chapter explores some of the most popular network applications available today.

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

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
- …
- …

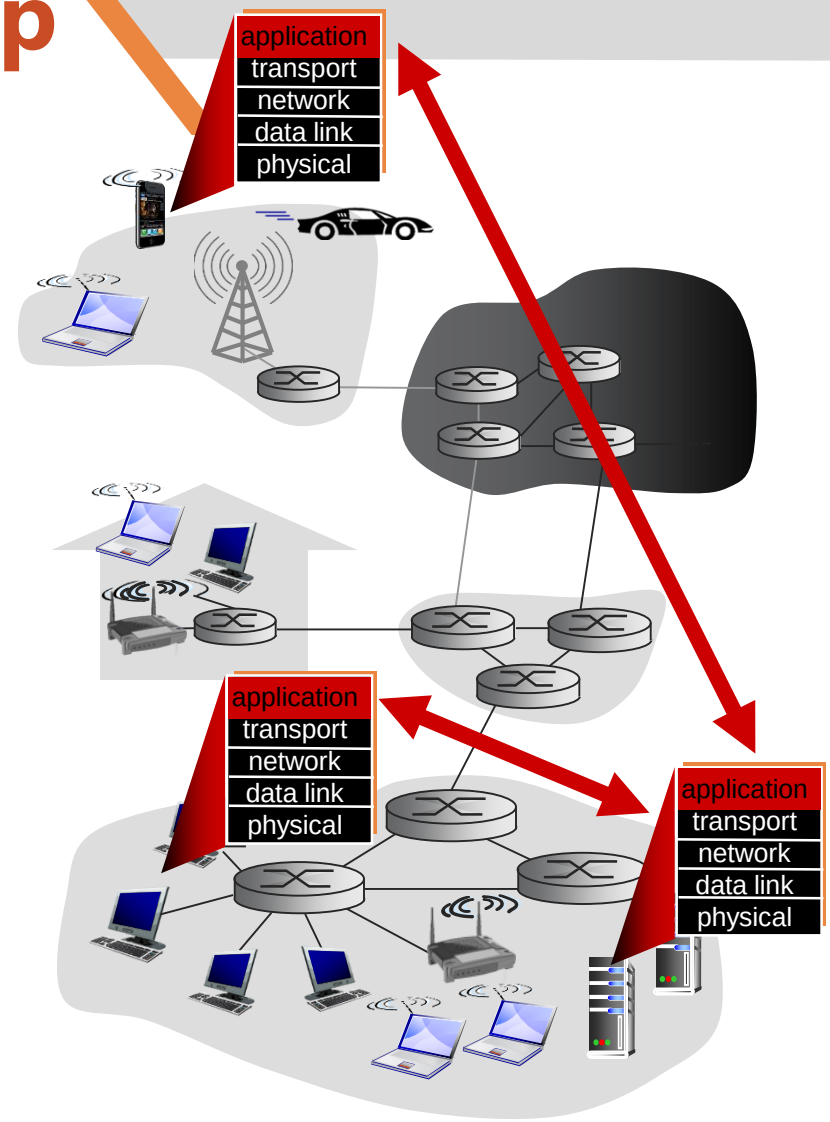
Creating a network app

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
- applications on end systems allows for rapid app development, propagation

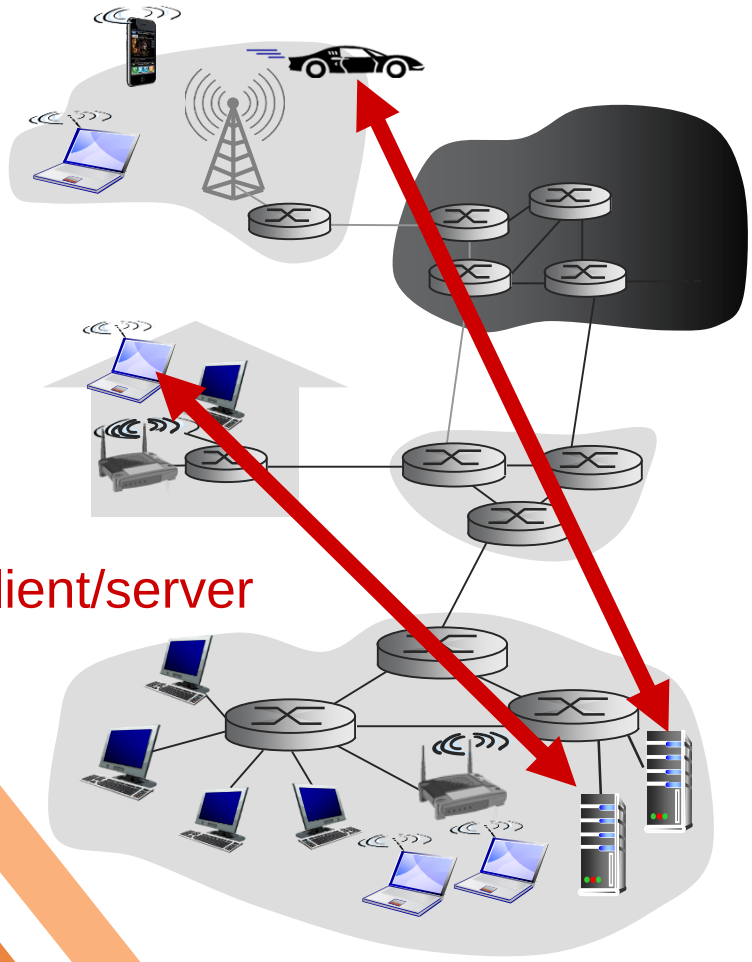


Application architectures

possible structure of applications:

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

Client-server architecture



client/server

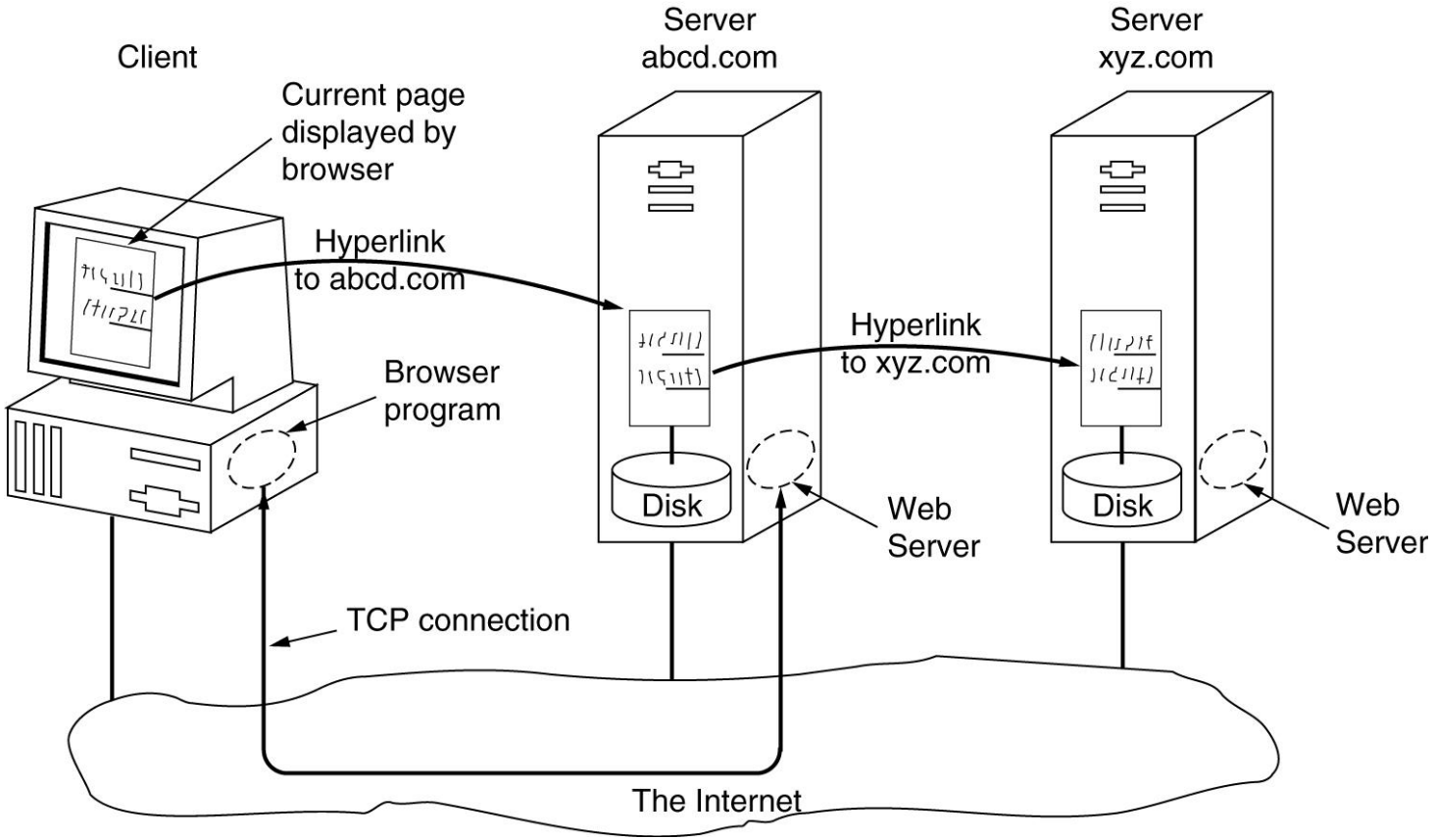
server:

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

clients:

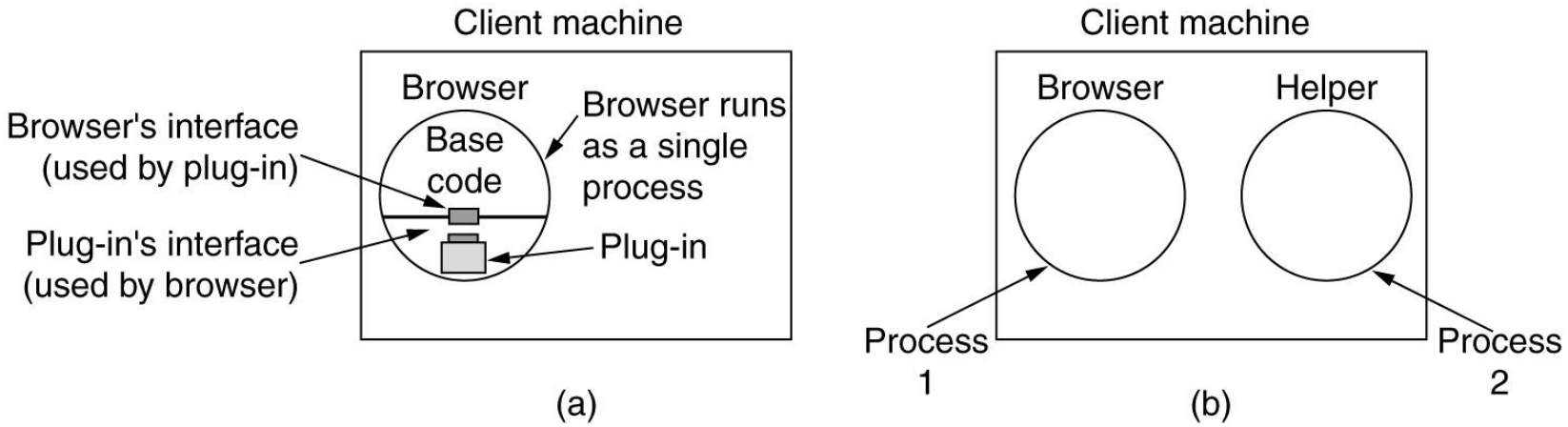
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Architectural Overview



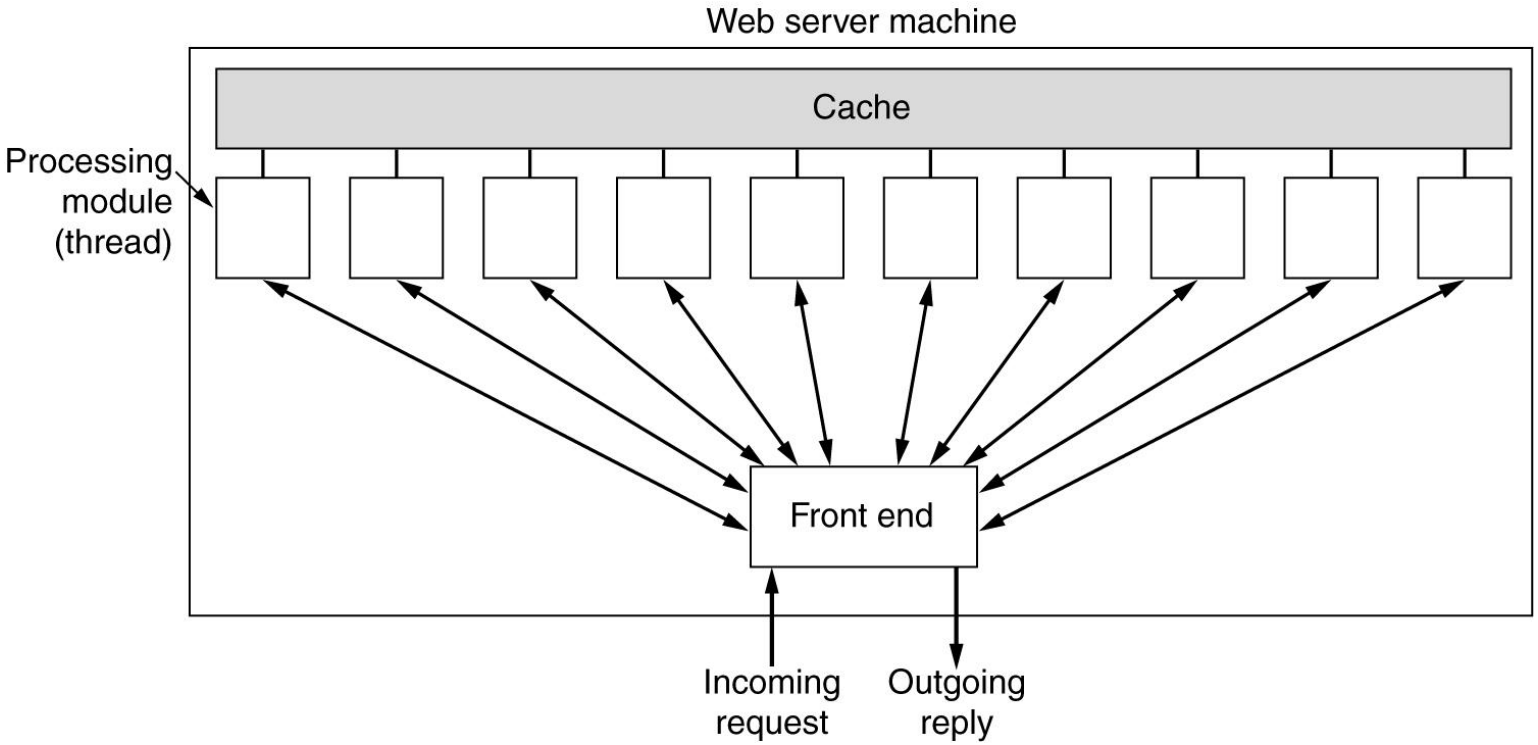
- The parts of the Web model.

The Client Side



- (a) A browser plug-in. (b) A helper application.

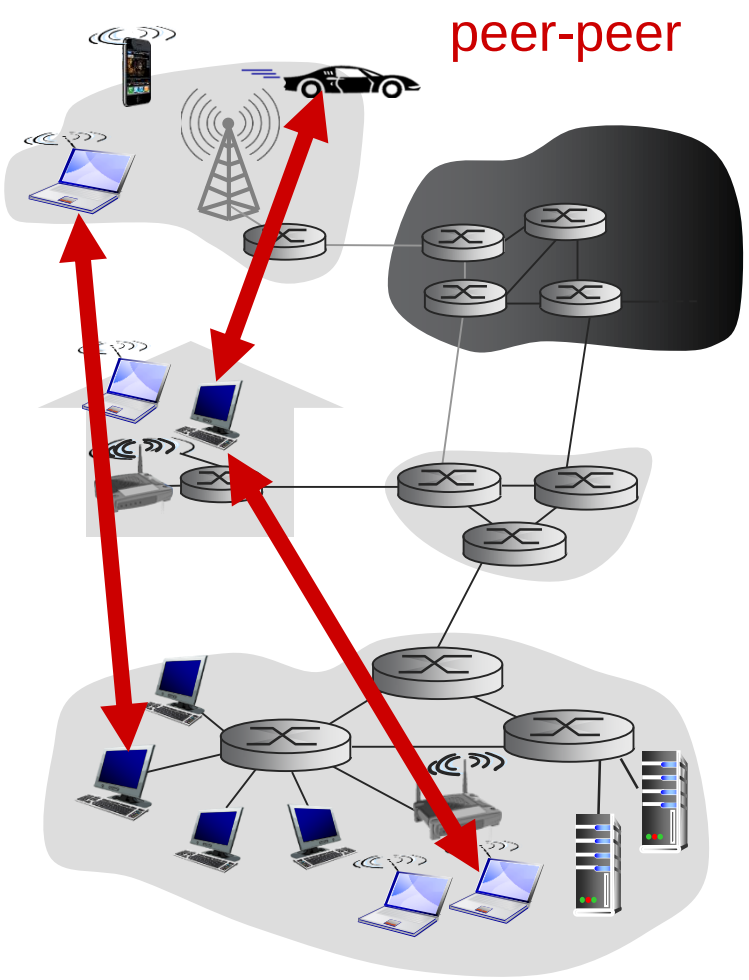
The Server Side



- A multithreaded Web server with a front end and processing modules.

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Processes communicating

process: program running within a host

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

clients, servers

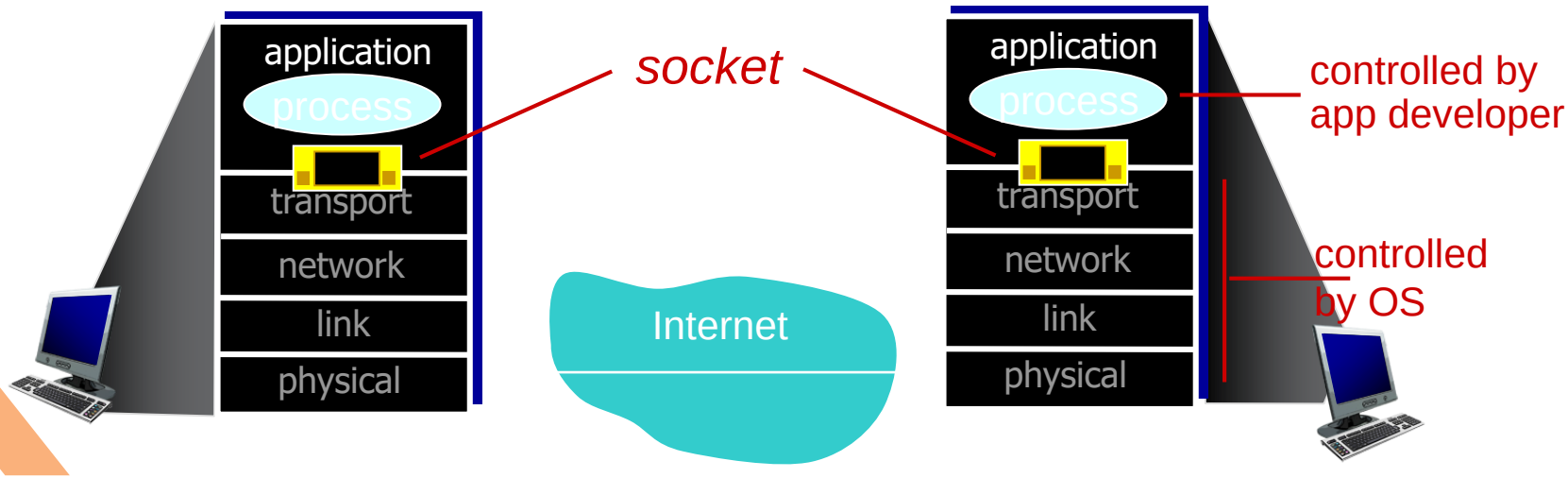
client process: process that initiates communication

server process: process that waits to be contacted

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

Sockets

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



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

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

App-layer protocol defines

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

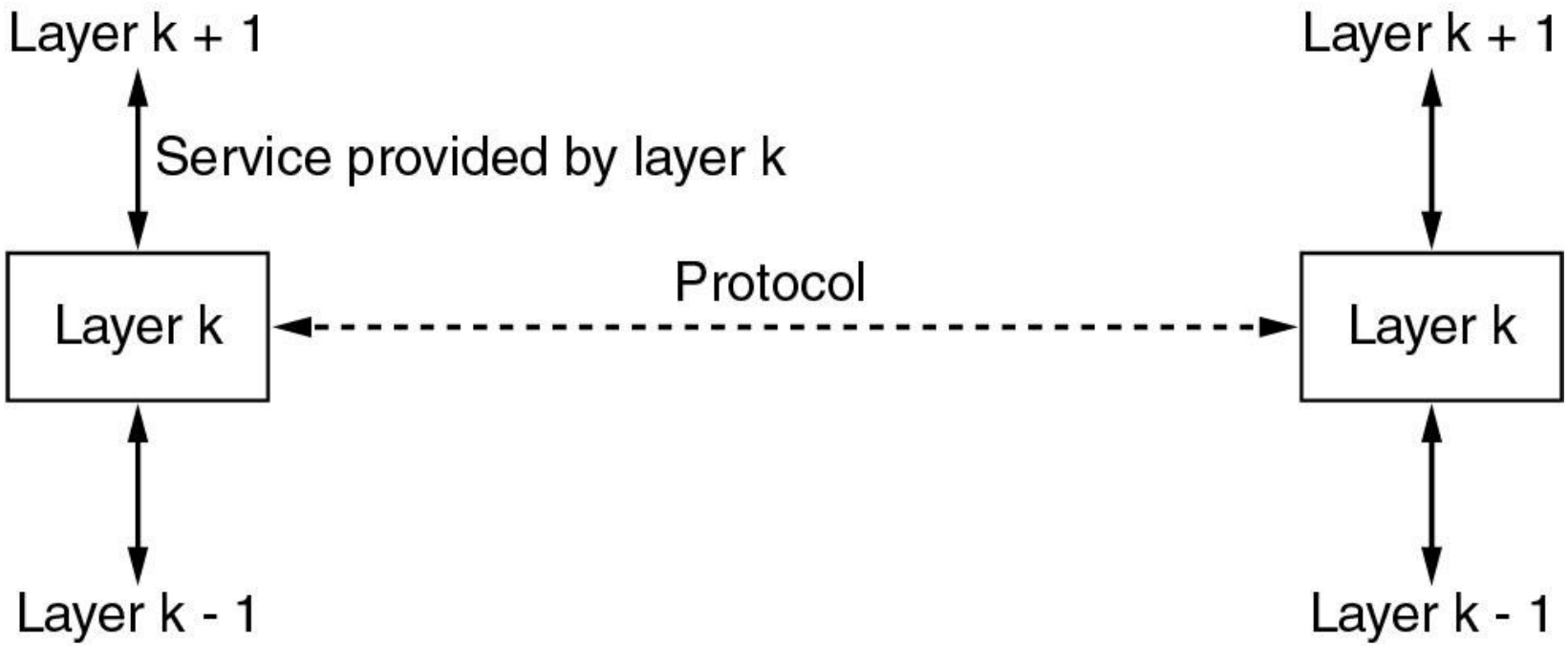
open protocols:

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

proprietary protocols:

- e.g., Skype

The relationship of Service



What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, …

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100’s msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100’s msec yes and no

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

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

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

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

Securing TCP

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- Apps use SSL libraries, which “talk” to TCP

SSL socket API

- cleartext passwds sent into socket traverse Internet encrypted

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

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

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

non-persistent HTTP

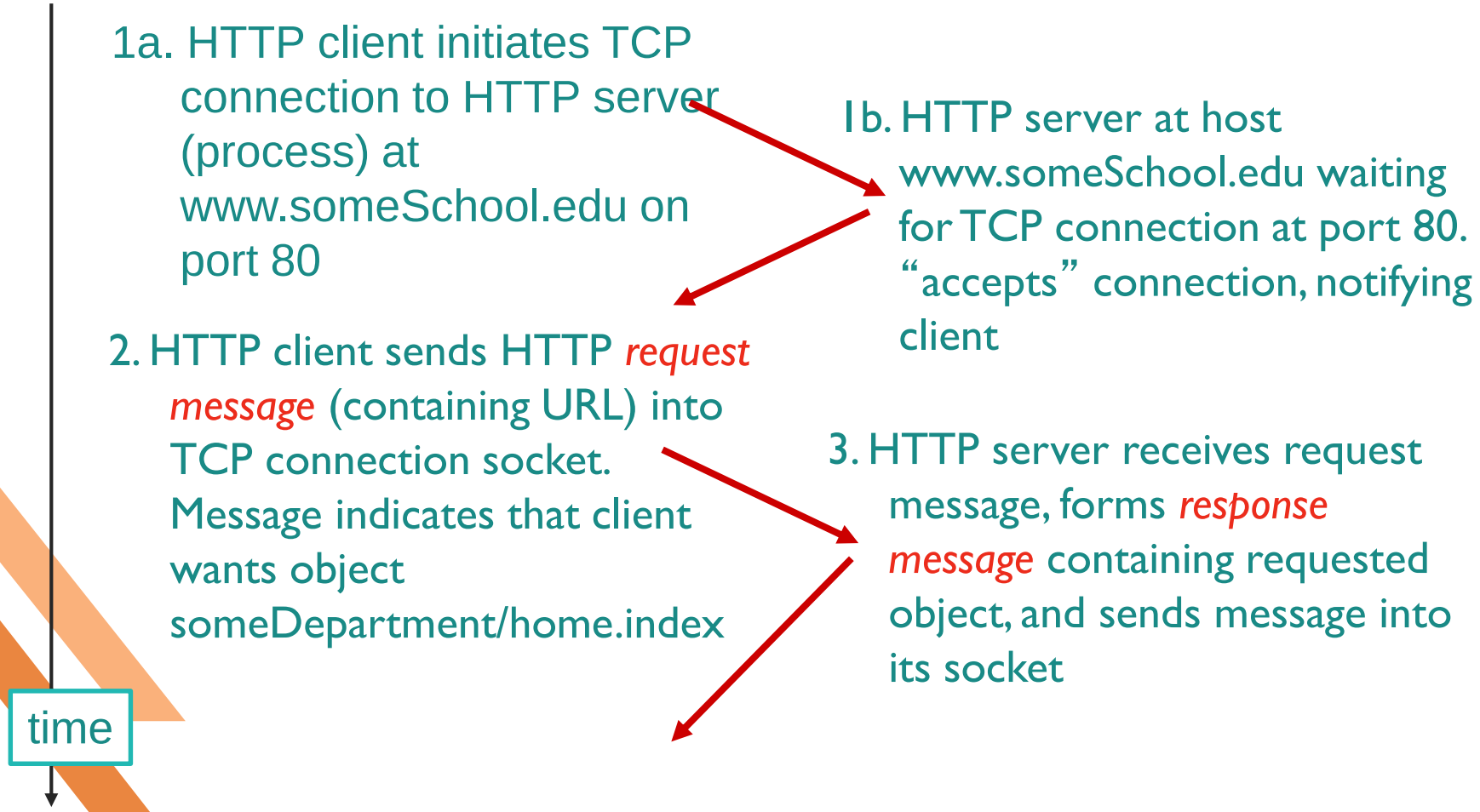
- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

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

Non-persistent HTTP

suppose user enters URL:



Non-persistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

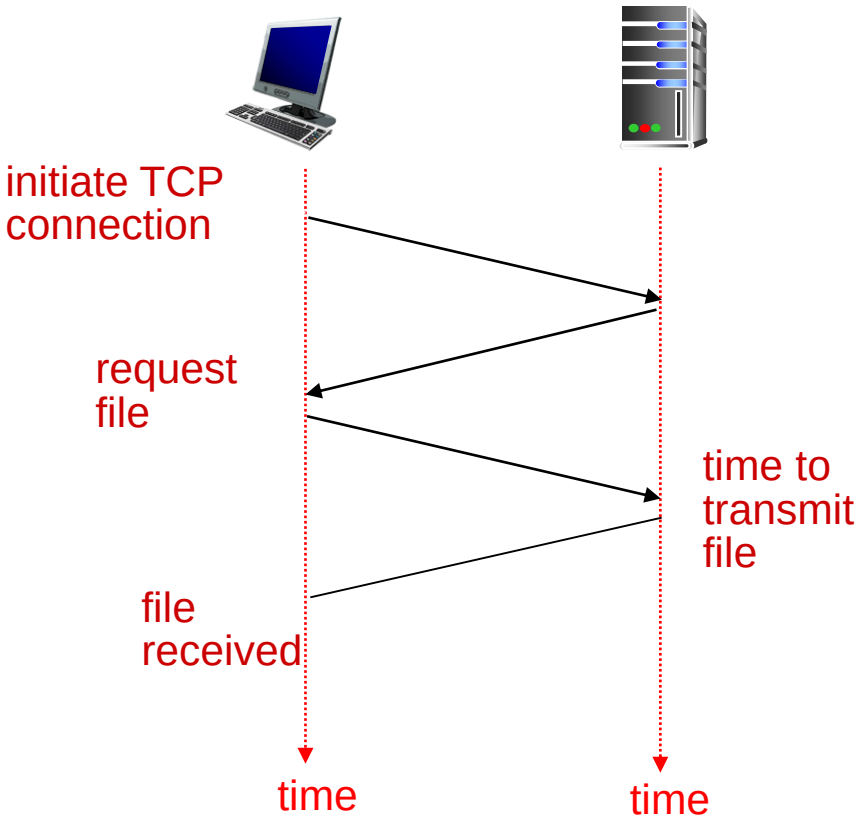
Non-persistent HTTP: response time

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

HTTP response time:

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

  2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

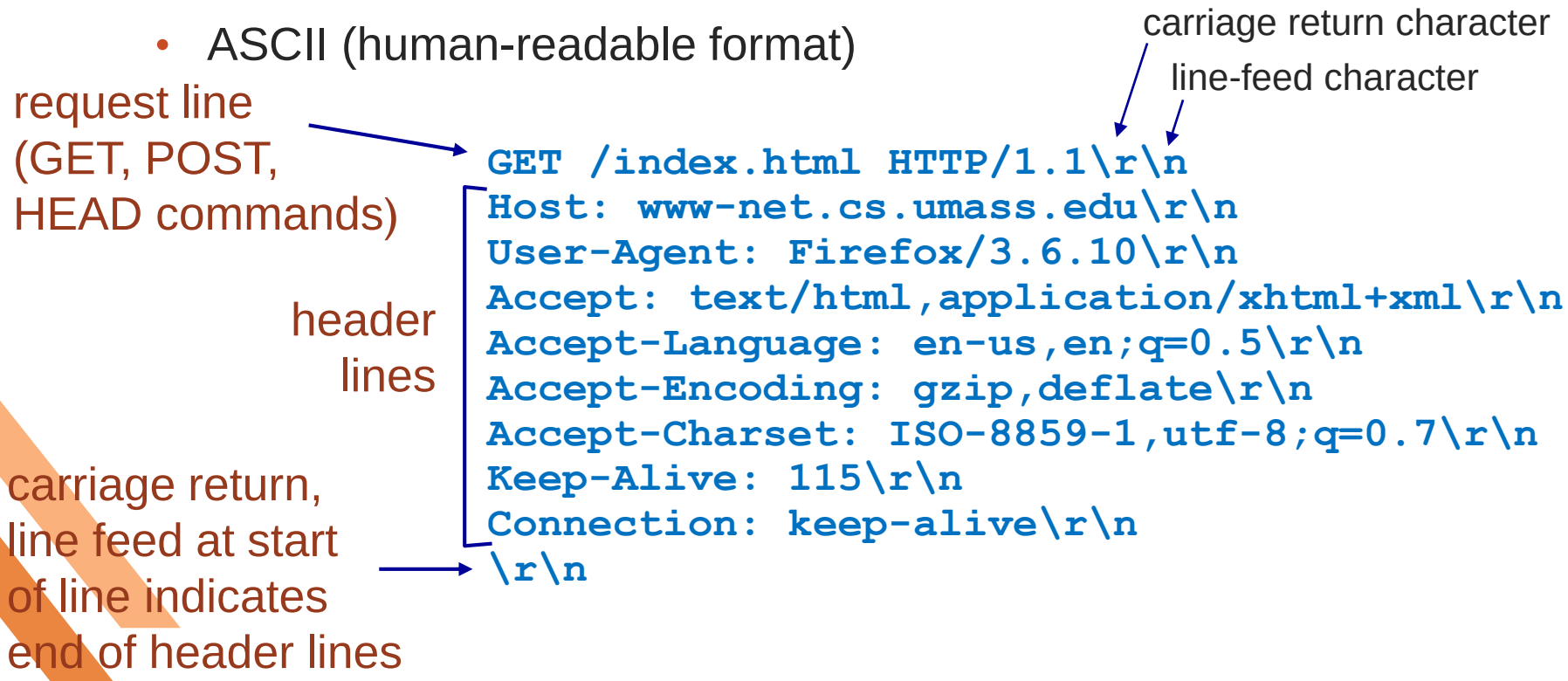
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

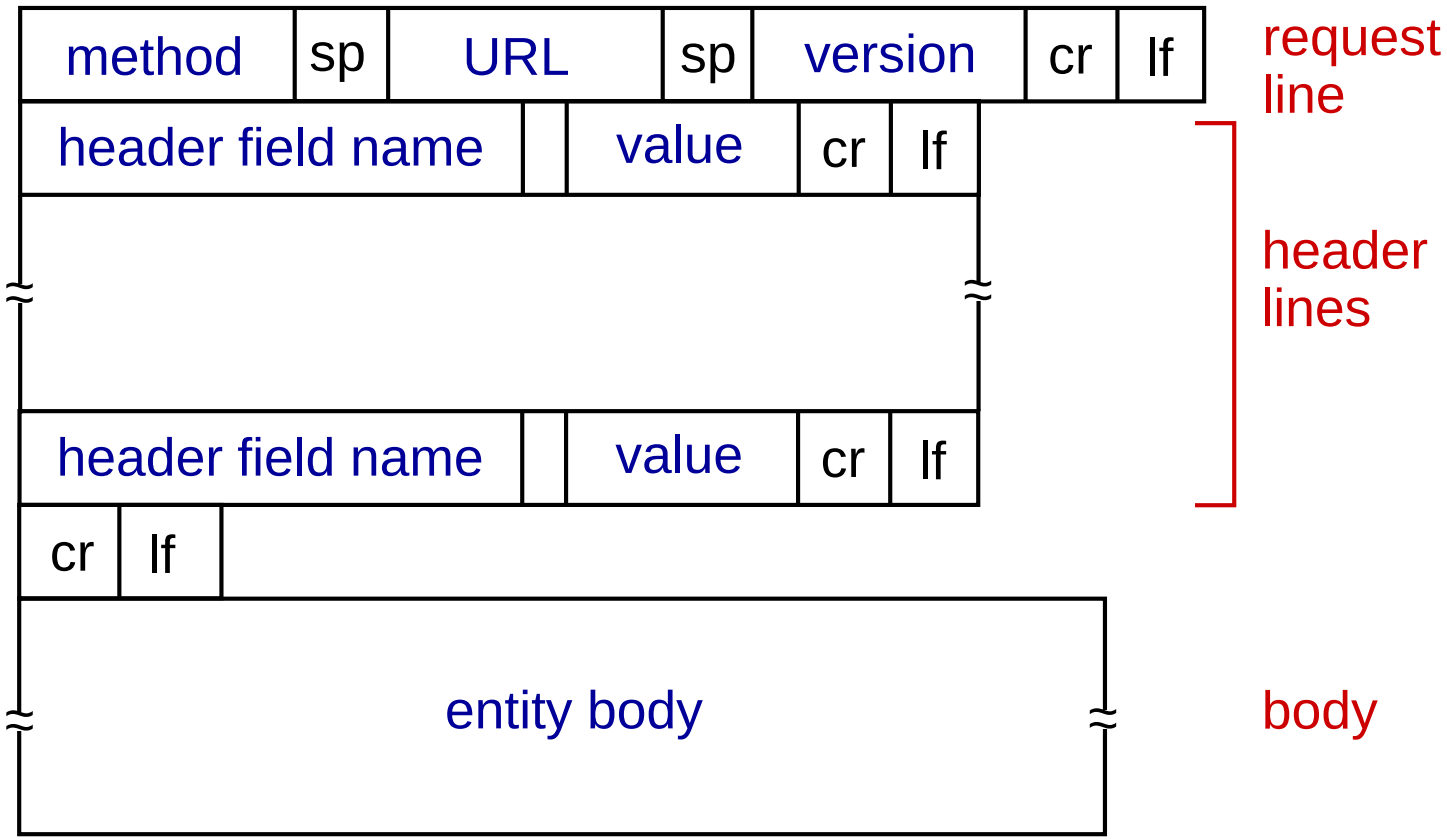
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

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



HTTP request message: general format



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.somesite.com/animalsearch?monkeys&banana
```

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 ...
```

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

Trying out HTTP (client side) for yourself การบ้าน

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

การบ้าน (คู่ละสองคะแนน)

การใช้งาน Telnet บน Windows

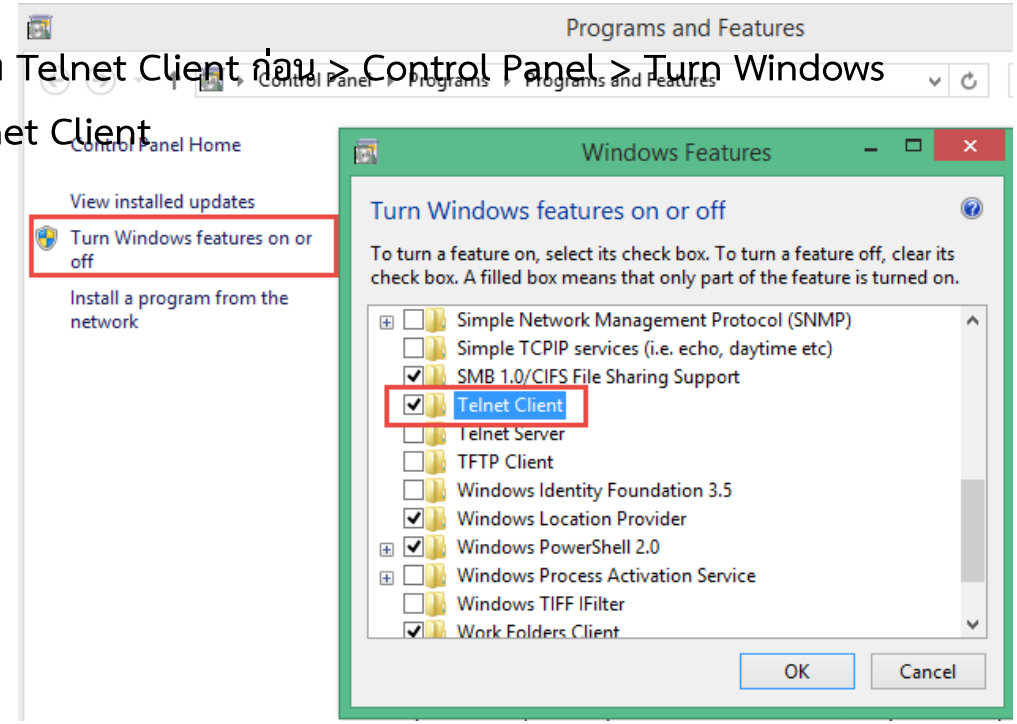
สำหรับการใช้คำสั่ง telnet เราสามารถทำได้ใน Command Prompt โดยสำหรับใน Windows 7 , Windows 8.1 และ Windows 10 จะต้องทำการเปิด Features ก่อนการใช้งาน โดยใน Function Telnet ส่วนมากจะเป็นคำสั่งของ Admin IT ที่เอาไว้ตรวจสอบ port ที่ให้บริการว่าปลายทางสามารถทำการเชื่อมต่อจากเครื่องต้นทางไปปลายทางได้หรือไม่

การ Telnet ของ Windows

โดยในคำสั่งจะเป็นการพิมพ์ > Telnet IP Address Port

อาทิเช่น telnet 192.168.100.120 25

1. โดยการใช้คำสั่งเราต้องทำการเปิด Feature ของ Telnet Client ก่อน > Control Panel > Turn Windows feature on or off . จากนั้นทำการเลือก (/) Telnet Client



User-server state: cookies

many Web sites use cookies

four components:

1) cookie header line of HTTP *response* message

2) cookie header line in next HTTP *request* message

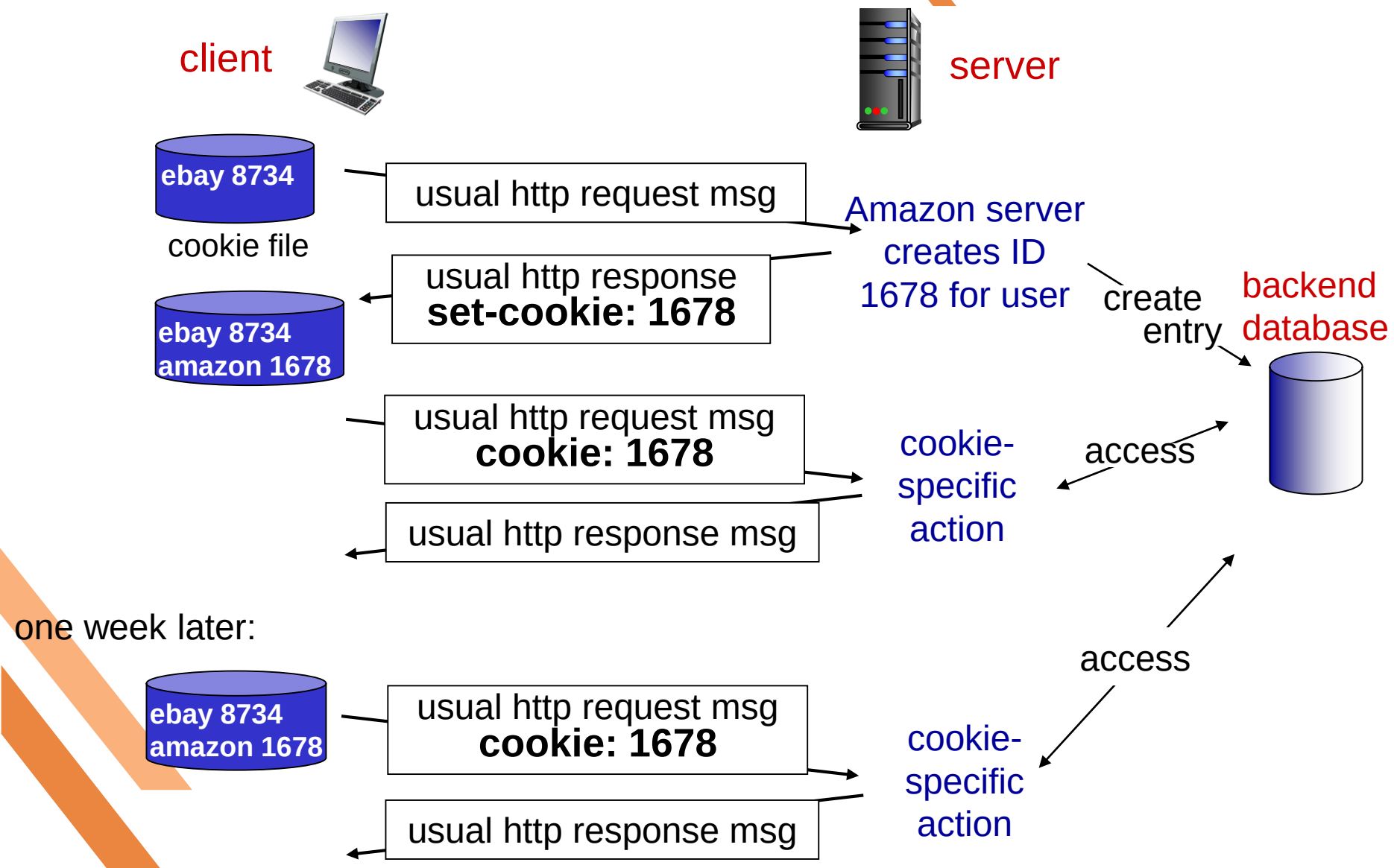
3) cookie file kept on user's host, managed by user's browser

4) back-end database at Web site

example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

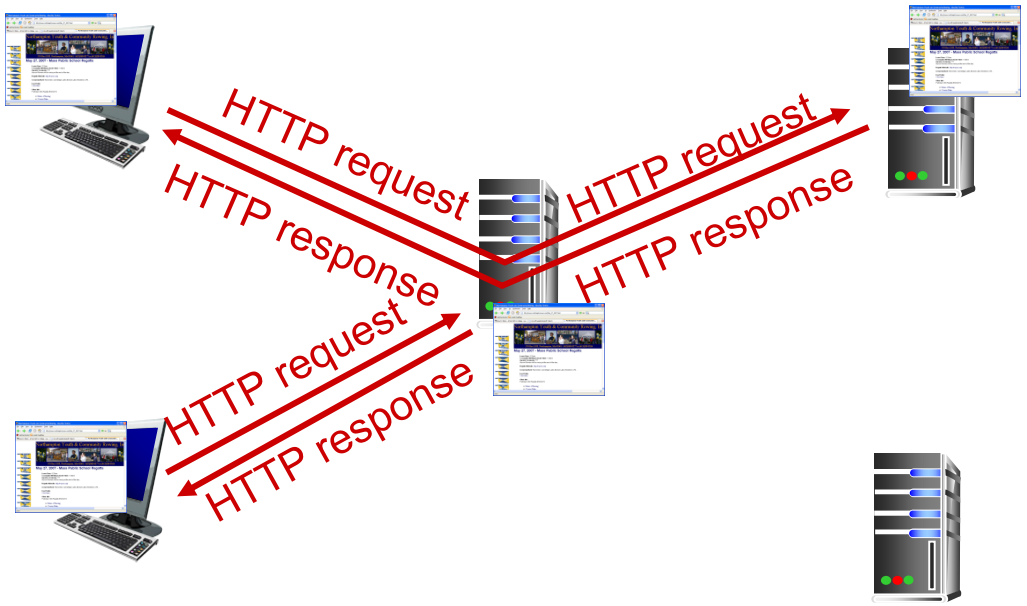
how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Web caches (proxy server)

goal:

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

การบ้าน

- ตั้งค่า browser ให้เข้าถึง proxy ห้องสมุดของมหาวิทยาลัย เพื่อทำการโหลดบทความวิชาการ
  - โหลดแล้วอย่าลืมลบ proxy ออกจาก setting

  (งานเดี่ยวทำทุกคน ส่งพร้อม proxy server และ ping)

- ติดตั้ง proxy server ใช้เอง หลังติดตั้ง Linux (SQUID Server) (งานกลุ่ม 5 คน)

Caching example:

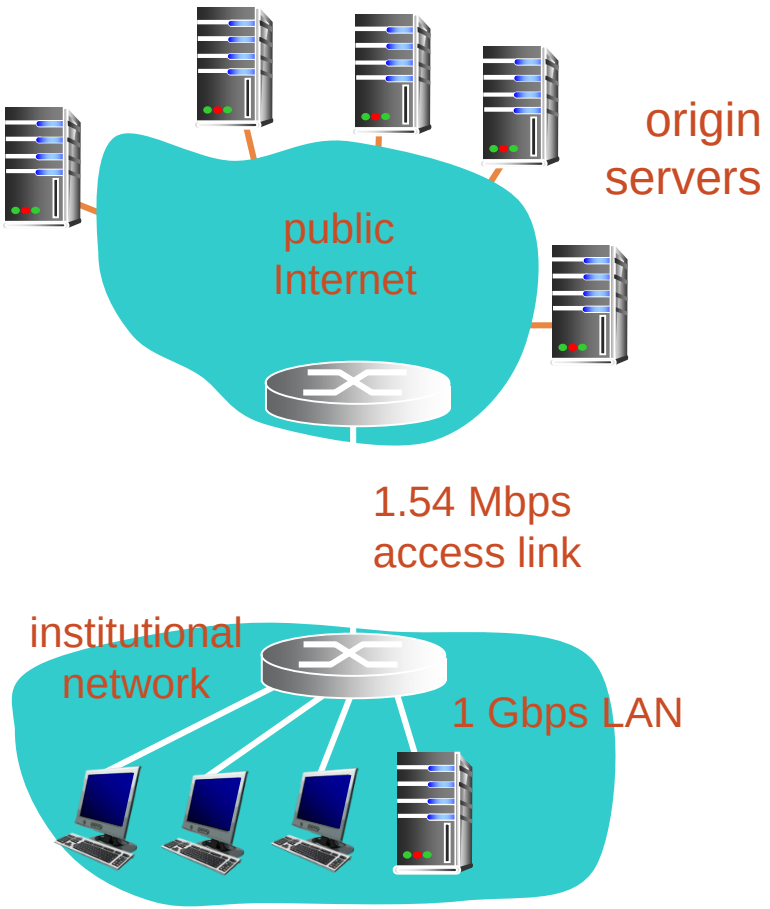
assumptions:

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

consequences:

- LAN utilization: 15%
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs

problem!



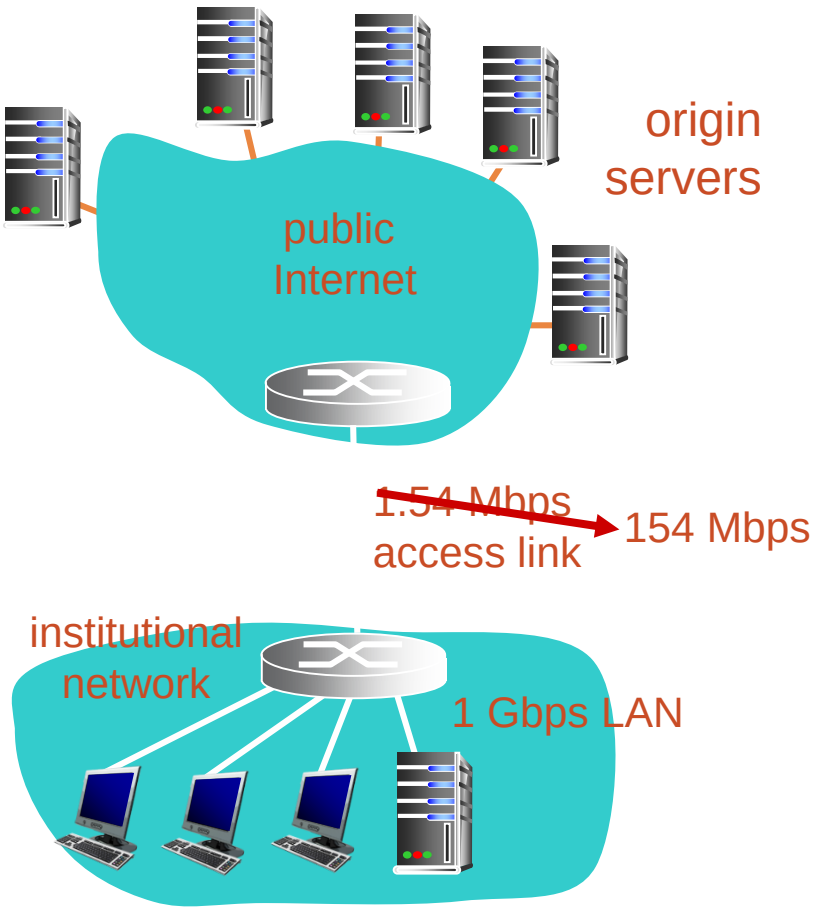
Caching example: fatter access link

assumptions:

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

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs msecs



Cost: increased access link speed (not cheap!)

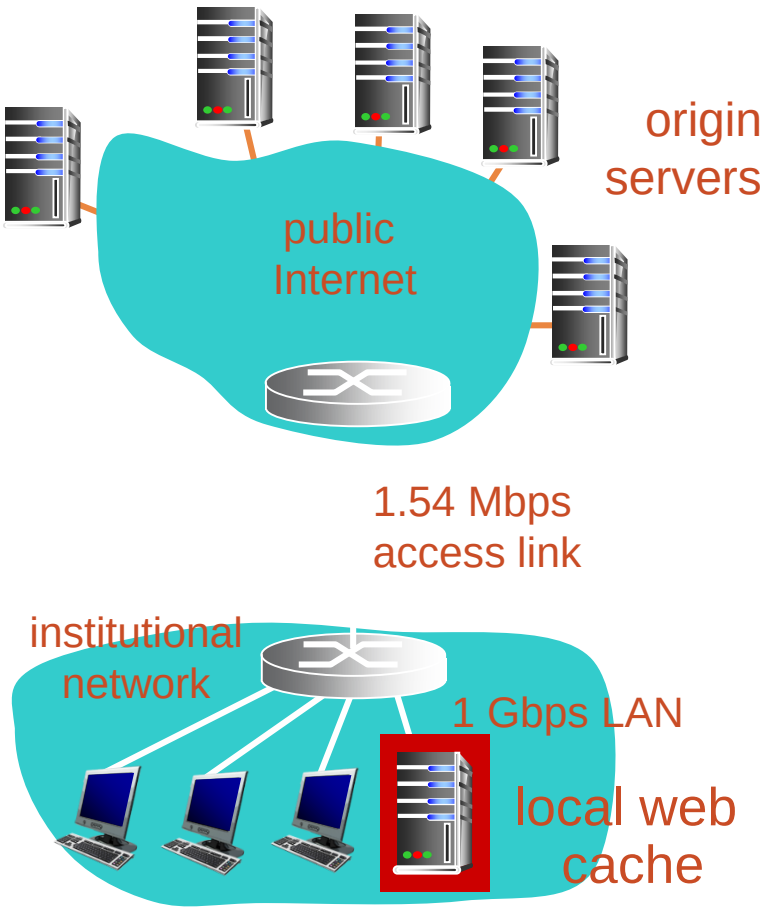
Caching example: install local cache

assumptions:

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

consequences:

- LAN utilization: 15%
- access link utilization = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs

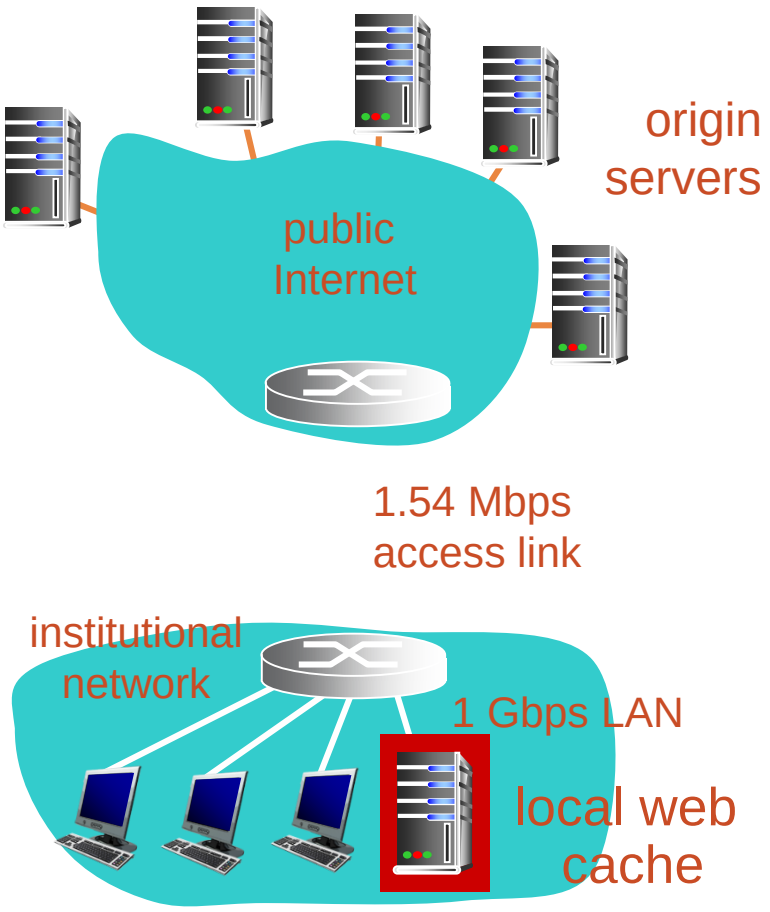


Cost: web cache (cheap!)

Caching example: install local cache

Calculating access link utilization, delay with cache:

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



- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58

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

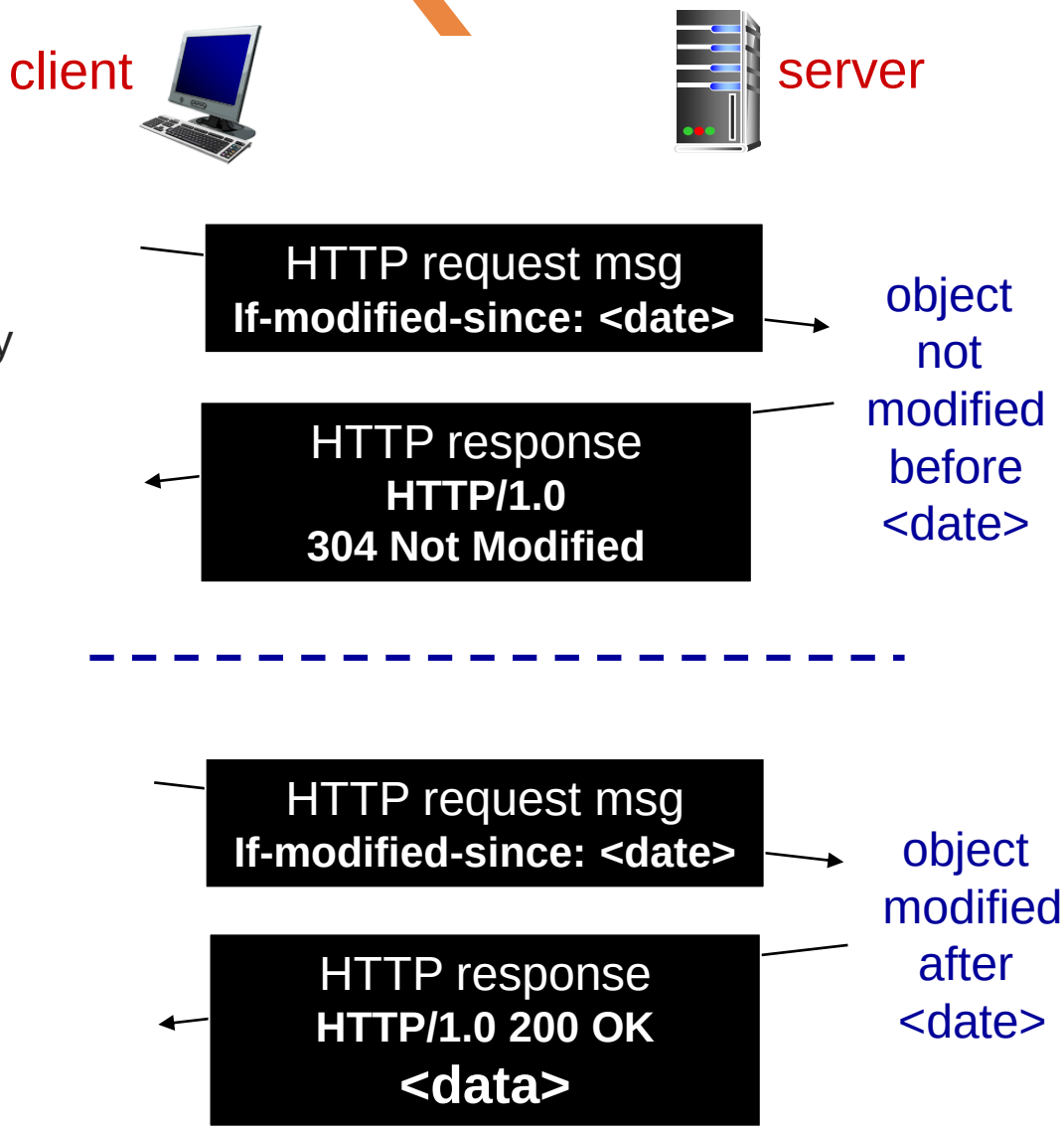
Conditional GET

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

```
If-modified-since:
   <date>
```

- *server:* response contains no object if cached copy is up-to-date:

```
HTTP/1.0 304 Not
   Modified
```



Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

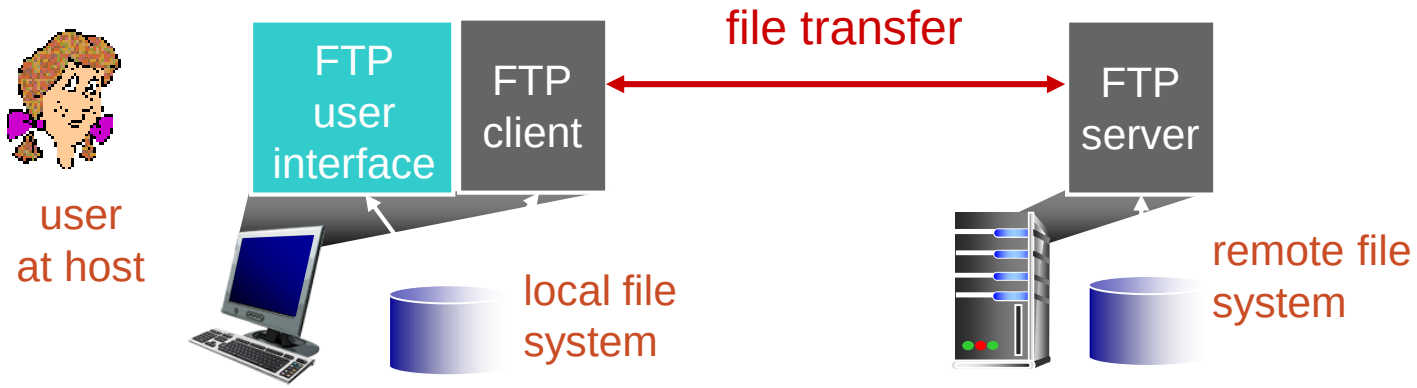
- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

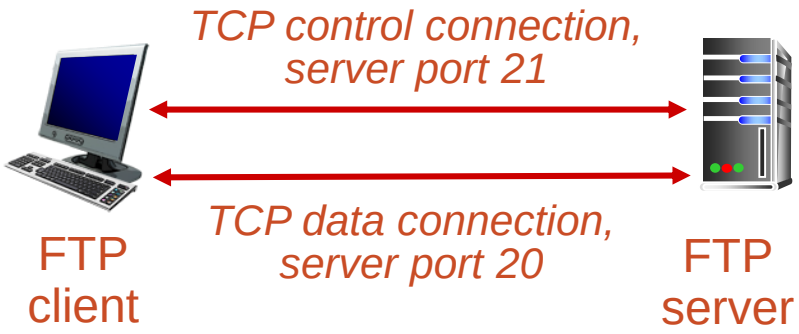
FTP: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens *2nd* TCP data connection (for file) *to* client
- after transferring one file, server closes data connection



- server opens another TCP data connection to transfer another file
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

FTP commands, responses

sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts) file onto remote host

sample return codes

- status code and phrase (as in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

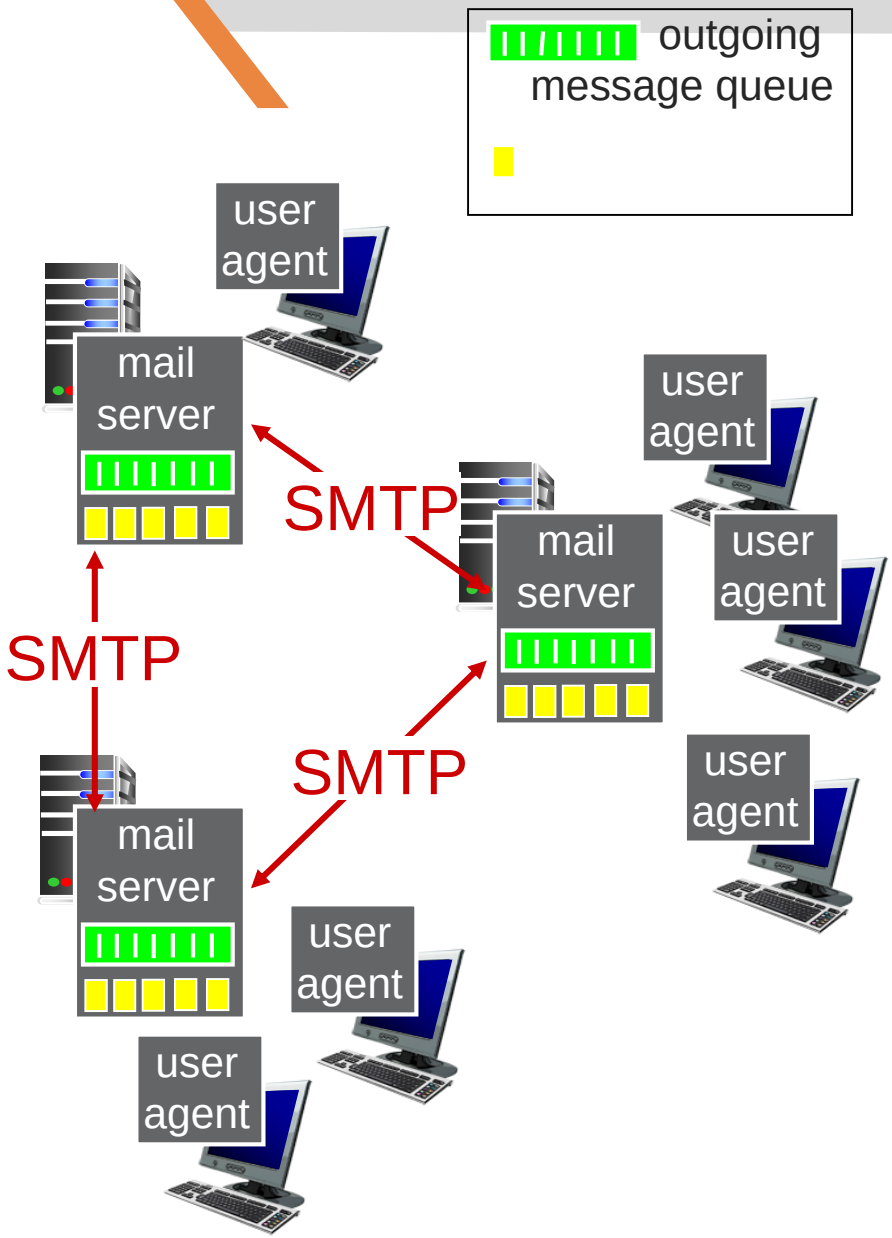
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

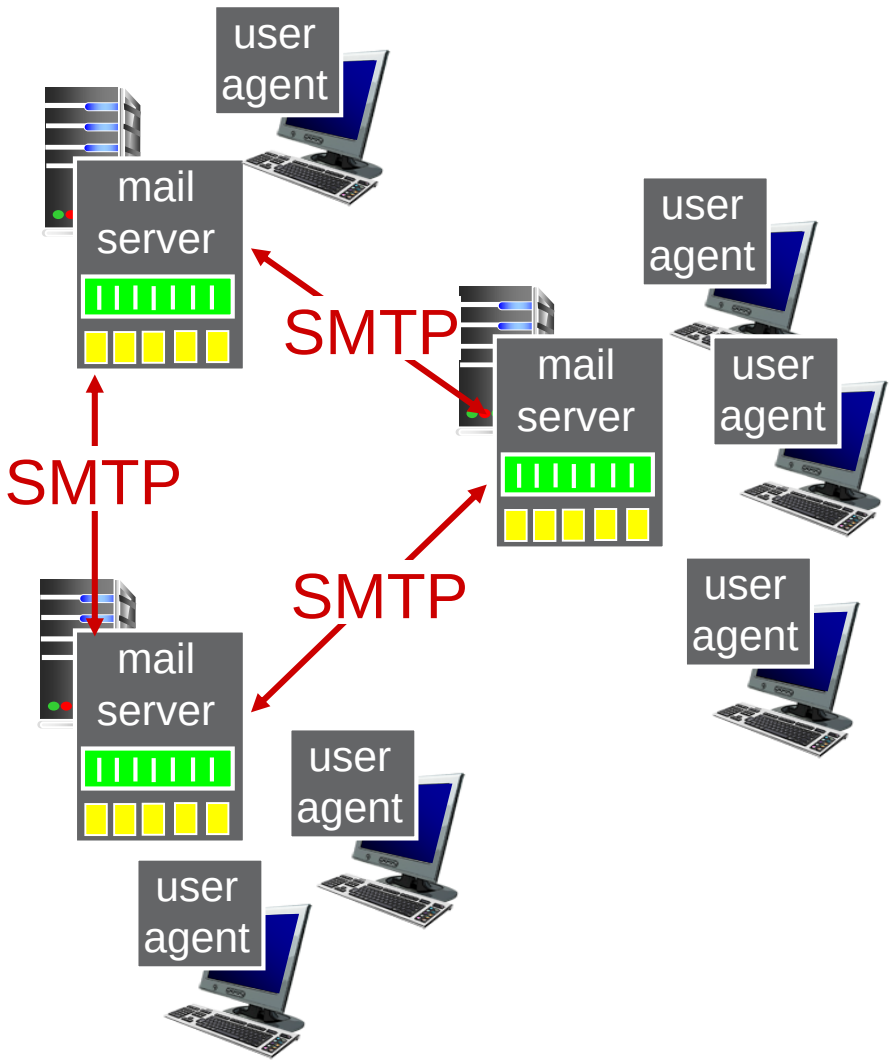
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

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



Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`

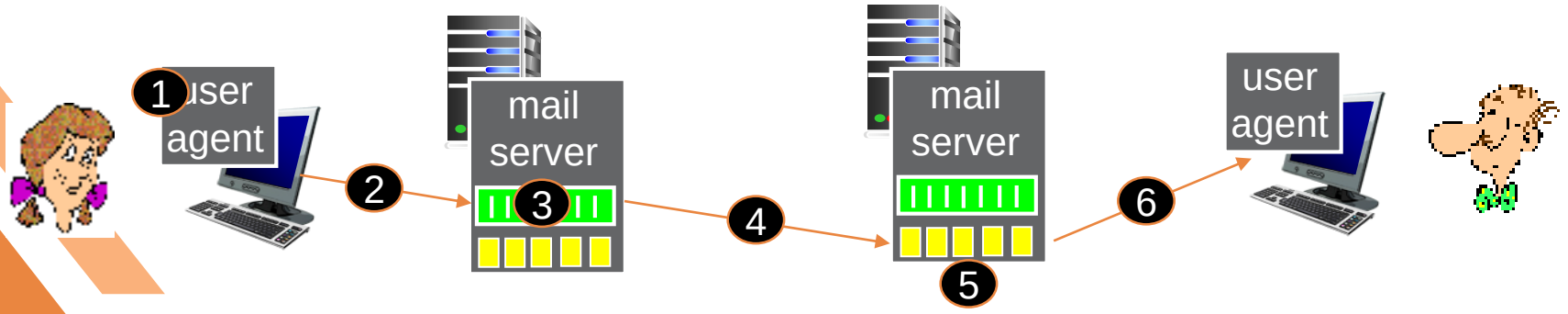
2) Alice's UA sends message to her mail server; message placed in message queue

3) client side of SMTP opens TCP connection with Bob's mail server

4) SMTP client sends Alice's message over the TCP connection

5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his user agent to read message



Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Try SMTP interaction for yourself:

- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

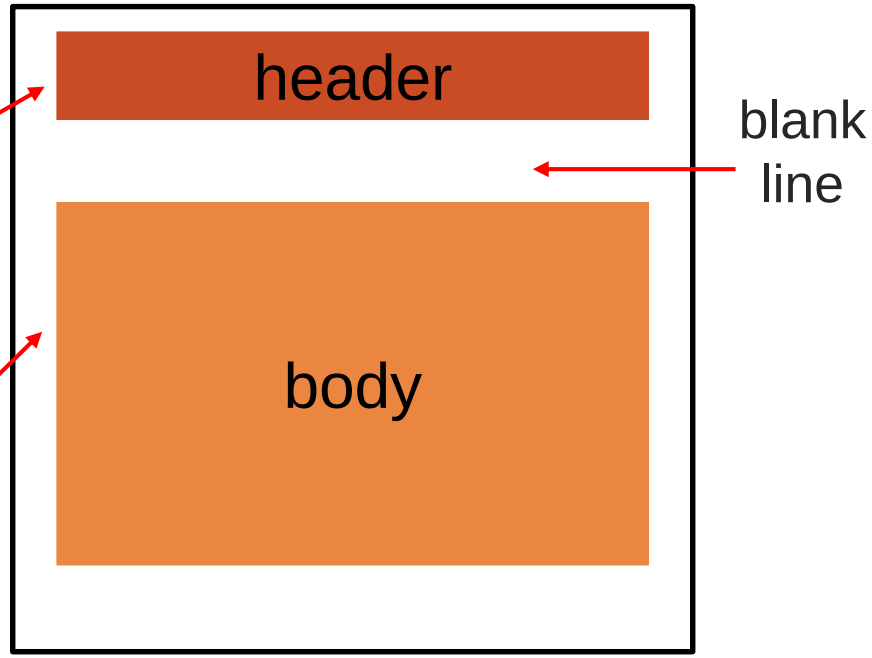
Mail message format

SMTP: protocol for exchanging email msgs

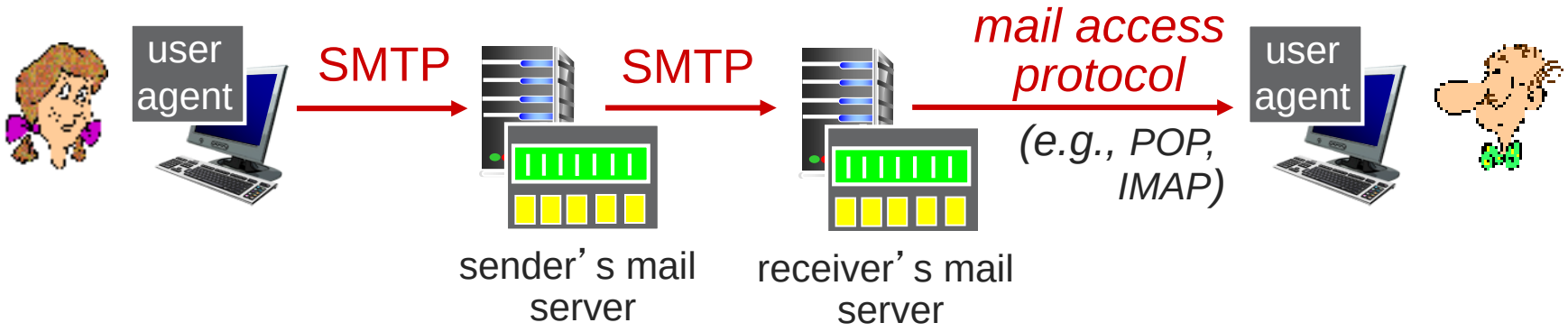
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

authorization phase

- client commands:
  - **`user:`** declare username
  - **`pass:`** password
- server responses
  - **`+OK`**
  - **`-ERR`**

transaction phase, client:

- **`list:`** list message numbers
- **`retr:`** retrieve message by number
- **`dele:`** delete
- **`quit`**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 “download and delete” mode
  - Bob cannot re-read e-mail if he changes client
- POP3 “download-and-keep”: copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

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

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

Domain Name System:

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

DNS: services, structure

DNS services

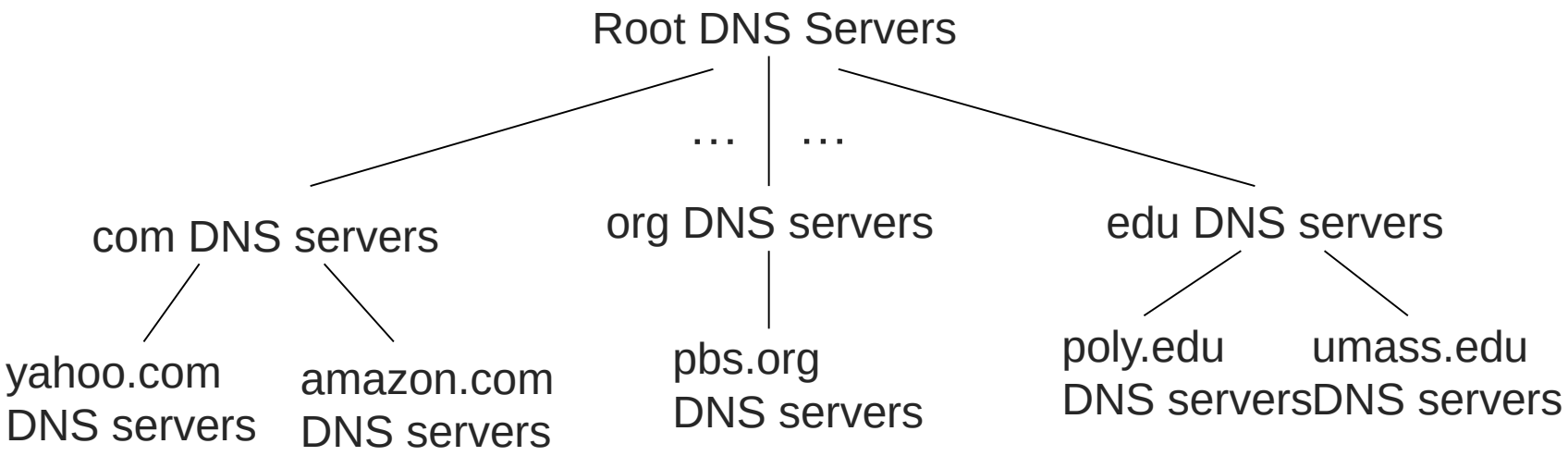
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

doesn't scale!

DNS: a distributed, hierarchical database

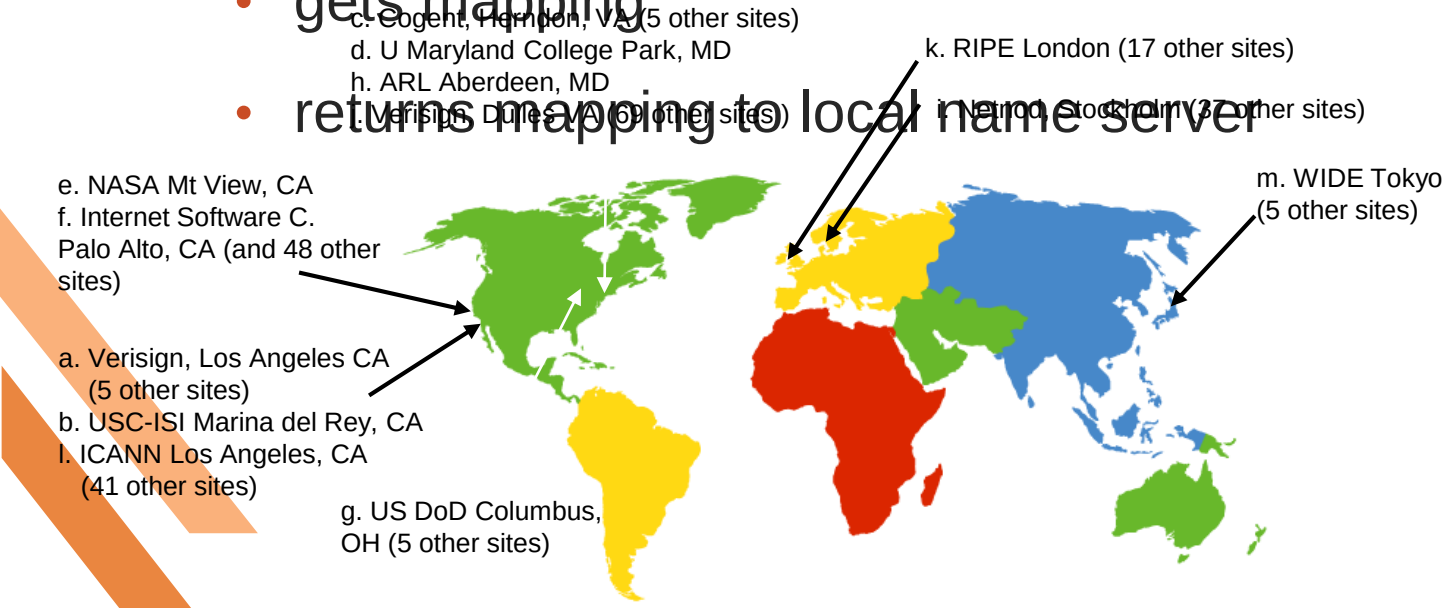


client wants IP for www.amazon.com; 1st approx:

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

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

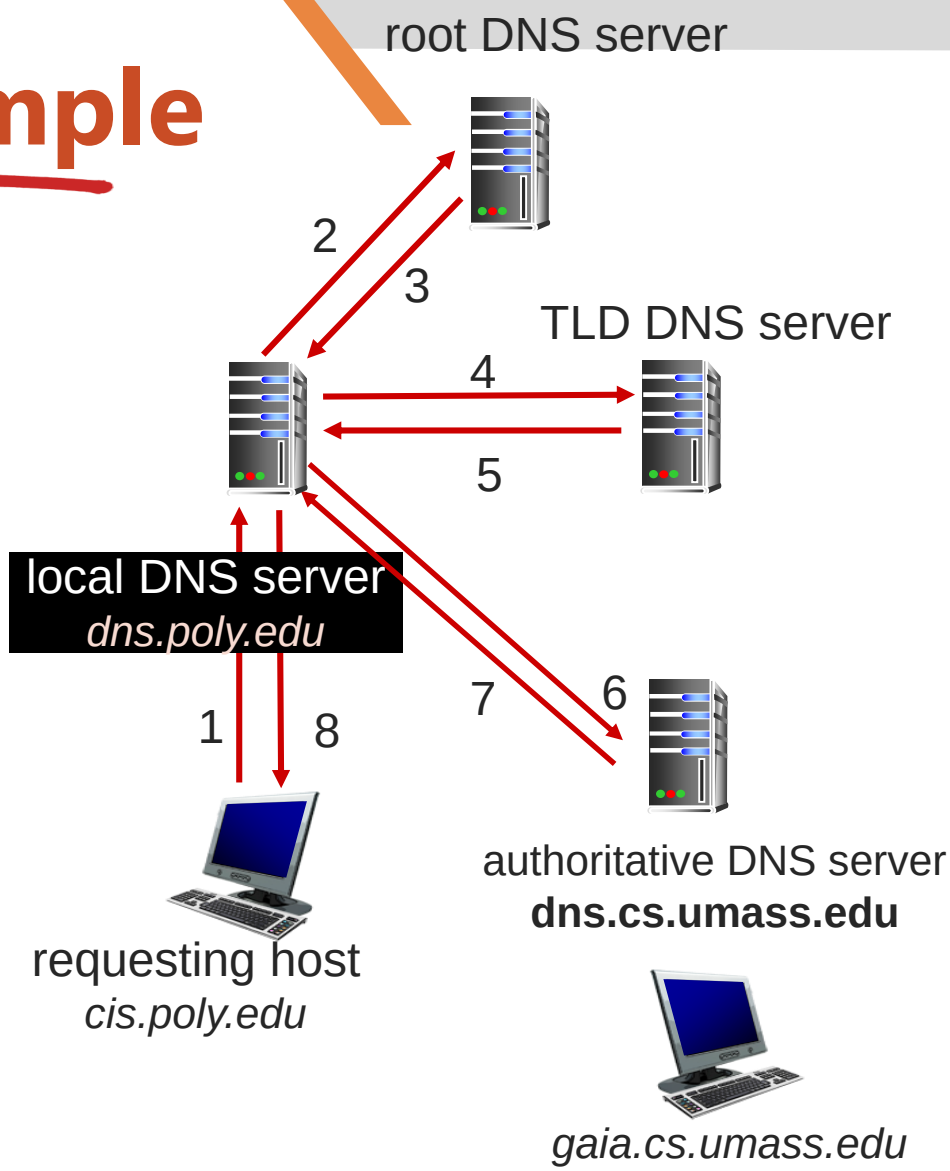
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts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

DNS name resolution example

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

iterated query:

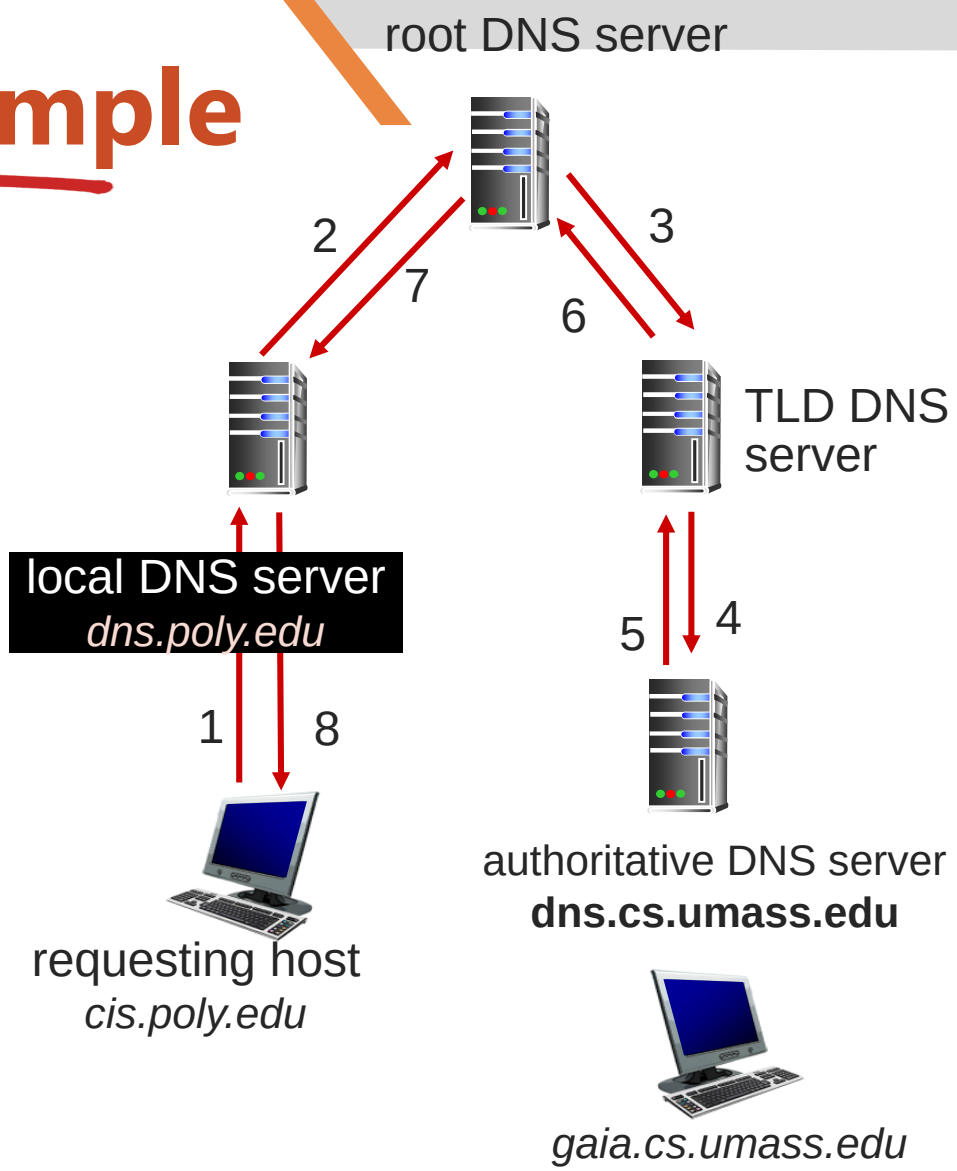
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS records

DNS: distributed db storing resource records (RR)

RR format: **(name, value, type, ttl)**

type=A

- **name** is hostname
- **value** is IP address

type=NS

- **name** is domain (e.g., foo.com)
- **value** is hostname of authoritative name server for this domain

type=CNAME

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

type=MX

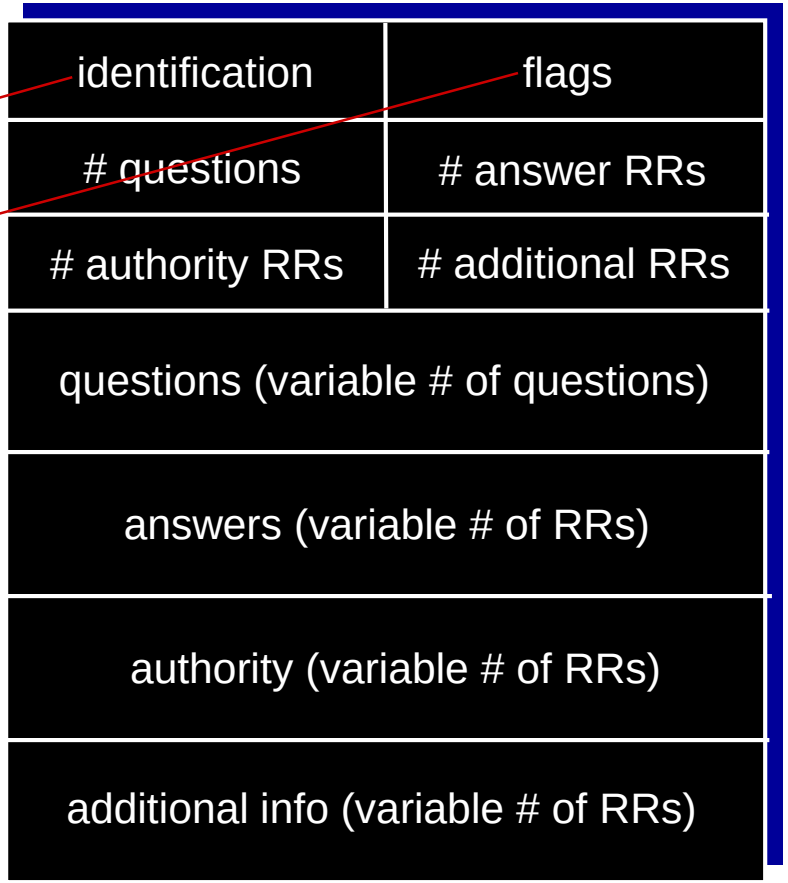
- **value** is name of mailserver associated with **name**

DNS protocol, messages

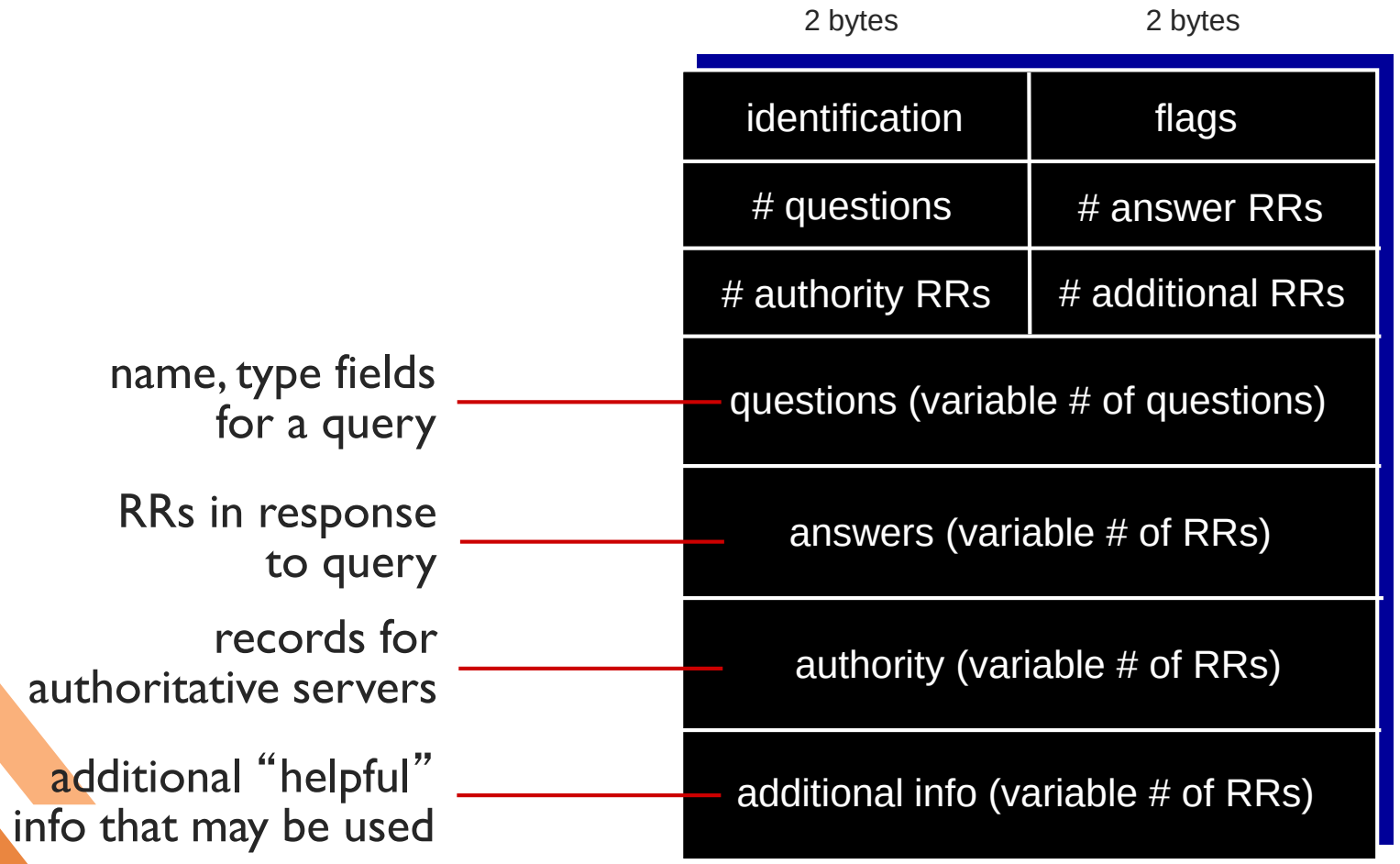
- *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS protocol, messages



Inserting records into DNS

- example: new startup “Network Utopia”
- register name networkuptopia.com at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into .com TLD server:
    (networkutopia.com, dns1.networkutopia.com, NS)

    (dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server type A record for www.networkuptopia.com; type MX record for networkutopia.com

Attacking DNS

DDoS attacks

- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus relies to DNS server, which caches

Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

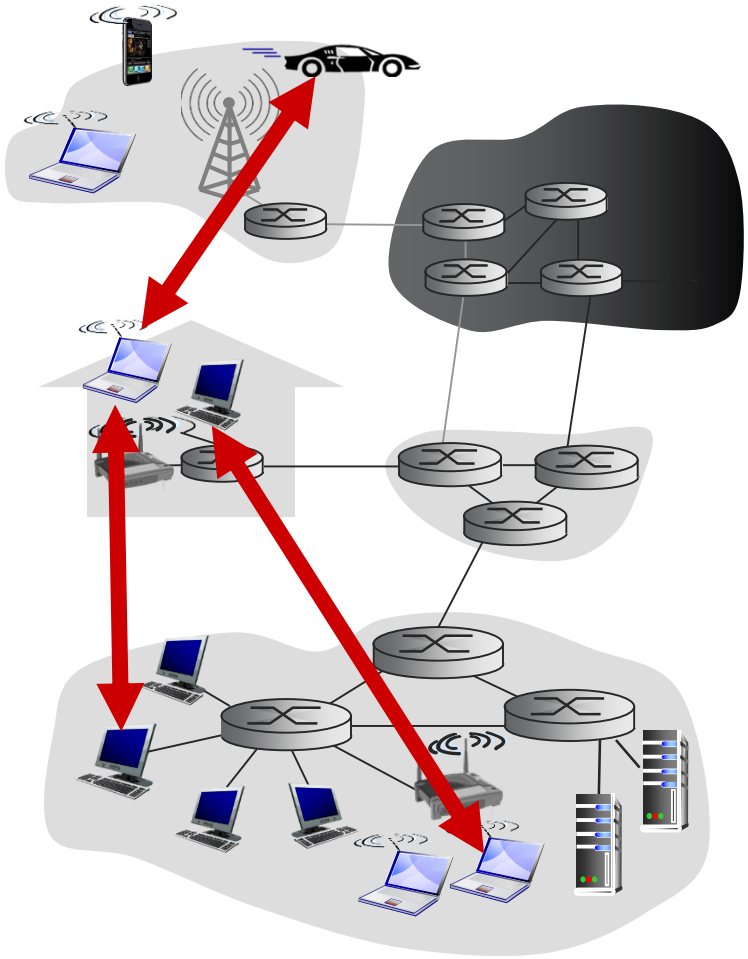
2.7 socket programming with UDP and TCP

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

examples:

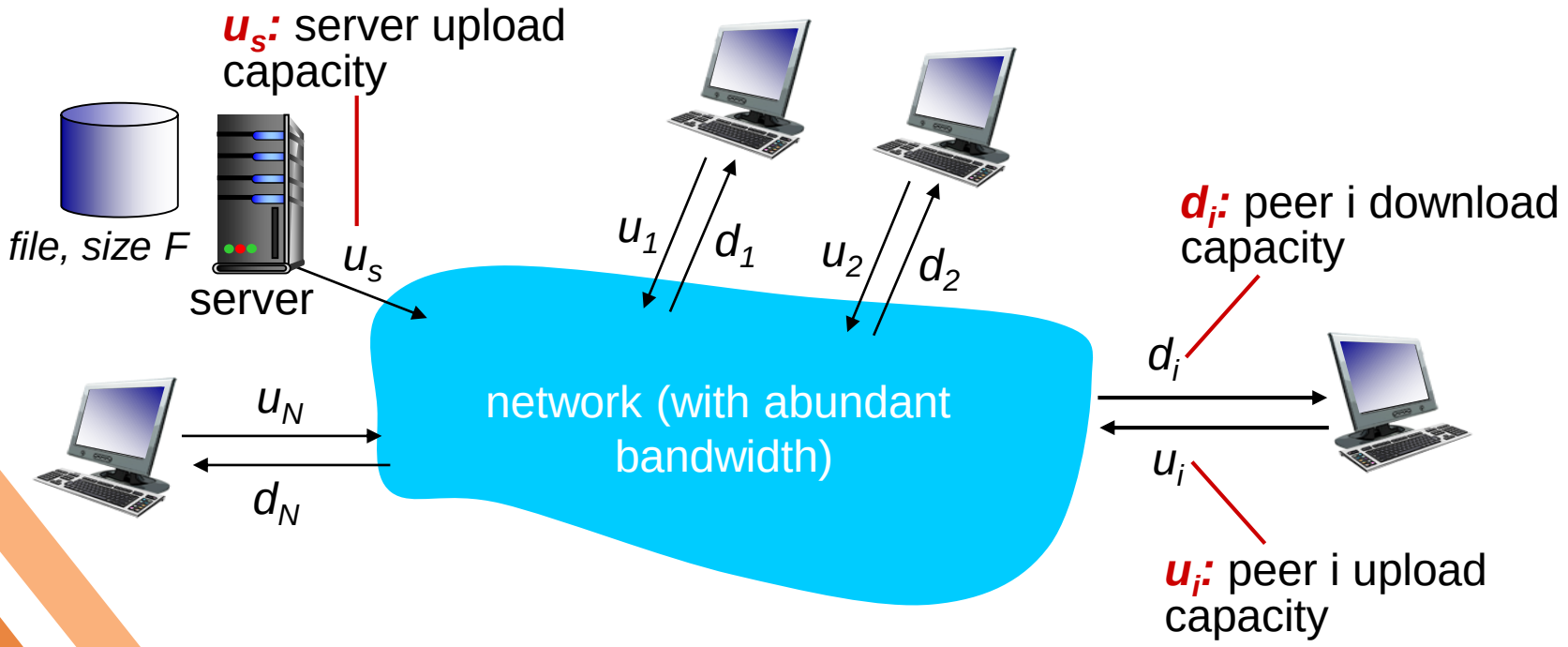
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

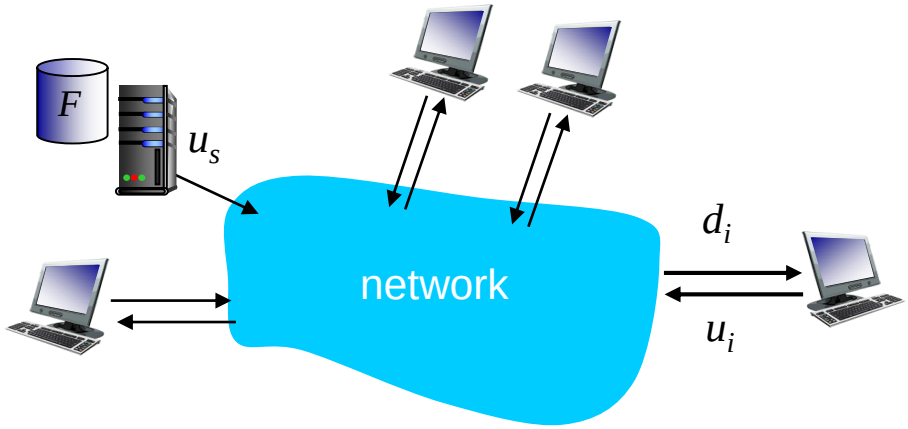
Question: how much time to distribute file (size *F*) from one server to *N* *peers*?

- peer upload/download capacity is limited resource



File distribution time: client-server

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



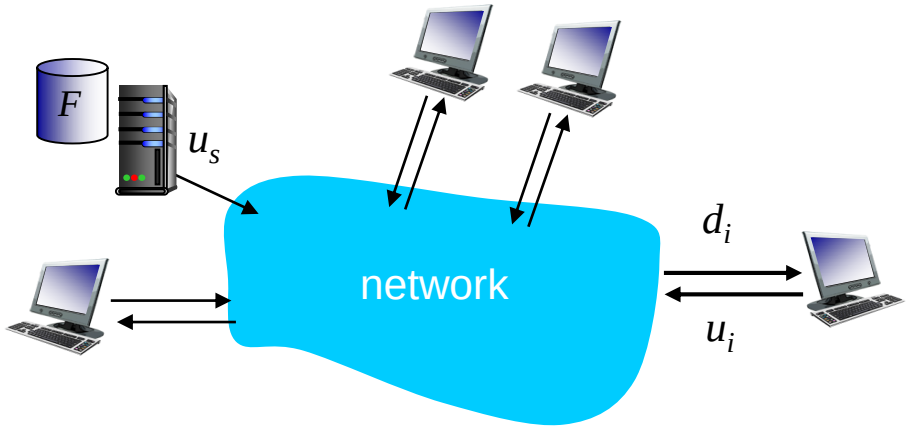
time to distribute F to N clients using client-server approach

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

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload **at least one** copy
  - time to send one copy: F/u_s



- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download NF bits
  - max upload rate (limting max download rate) is $u_s + Su_i$

time to distribute F to N clients using P2P approach

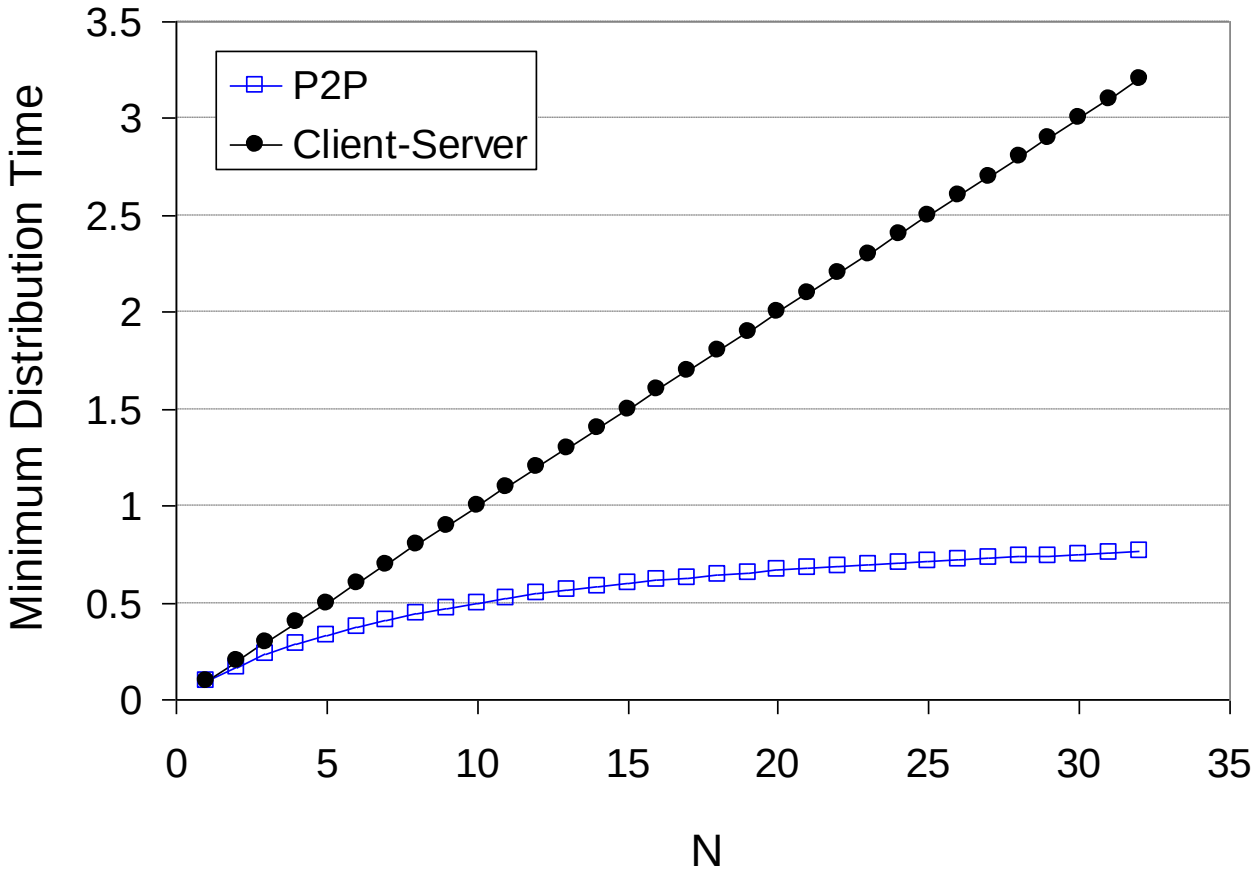
$$D_{P2P} > max\{F/u_s, F/d_{min}, NF/(u_s + Su_i)\}$$

increases linearly in N …

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

Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

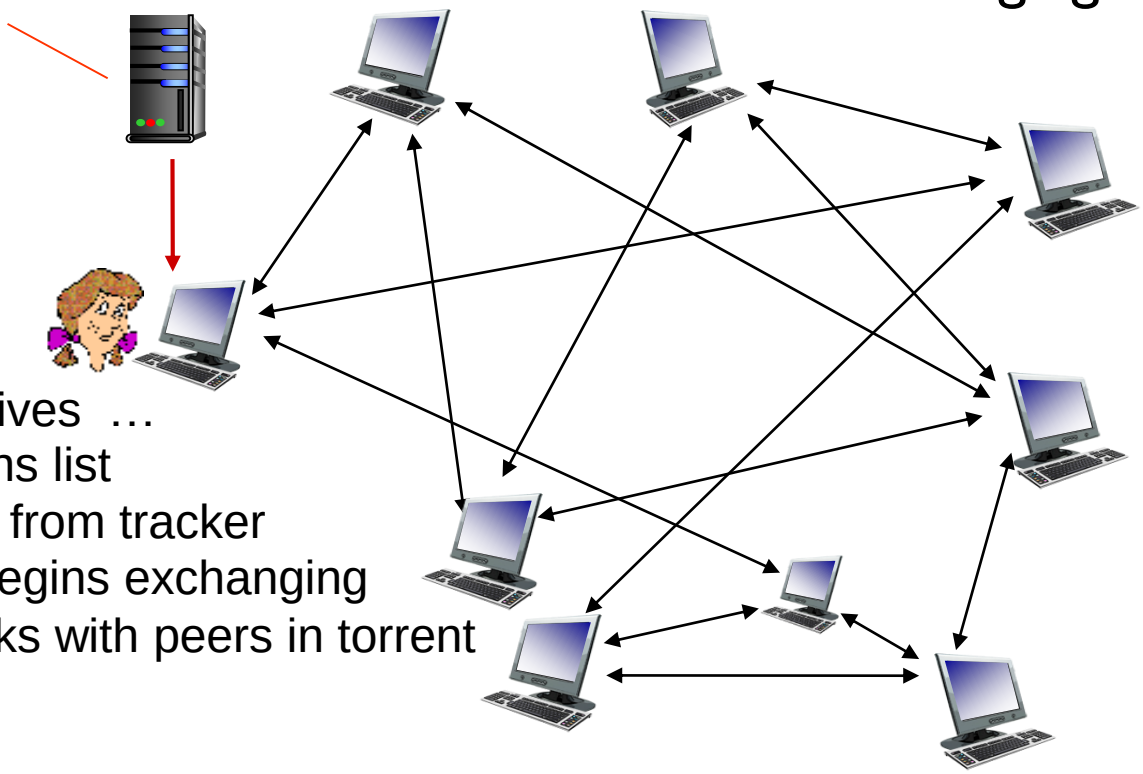


P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

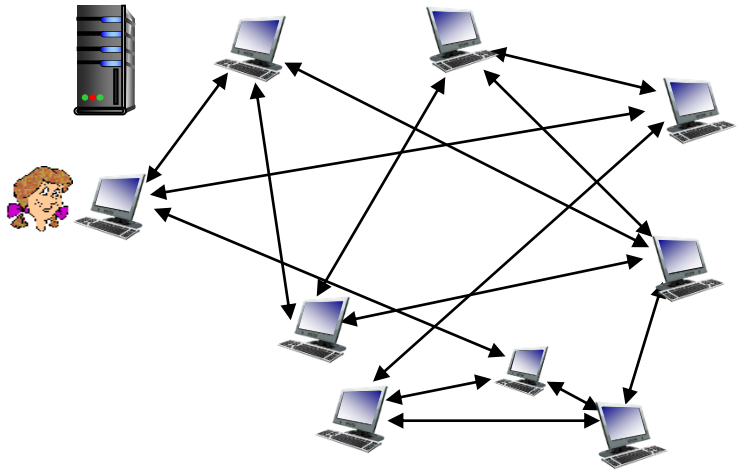
torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list of peers from tracker
… and begins exchanging file chunks with peers in torrent

P2P file distribution: BitTorrent

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



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

BitTorrent: requesting, sending file chunks

requesting chunks:

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

sending chunks: tit-for-tat

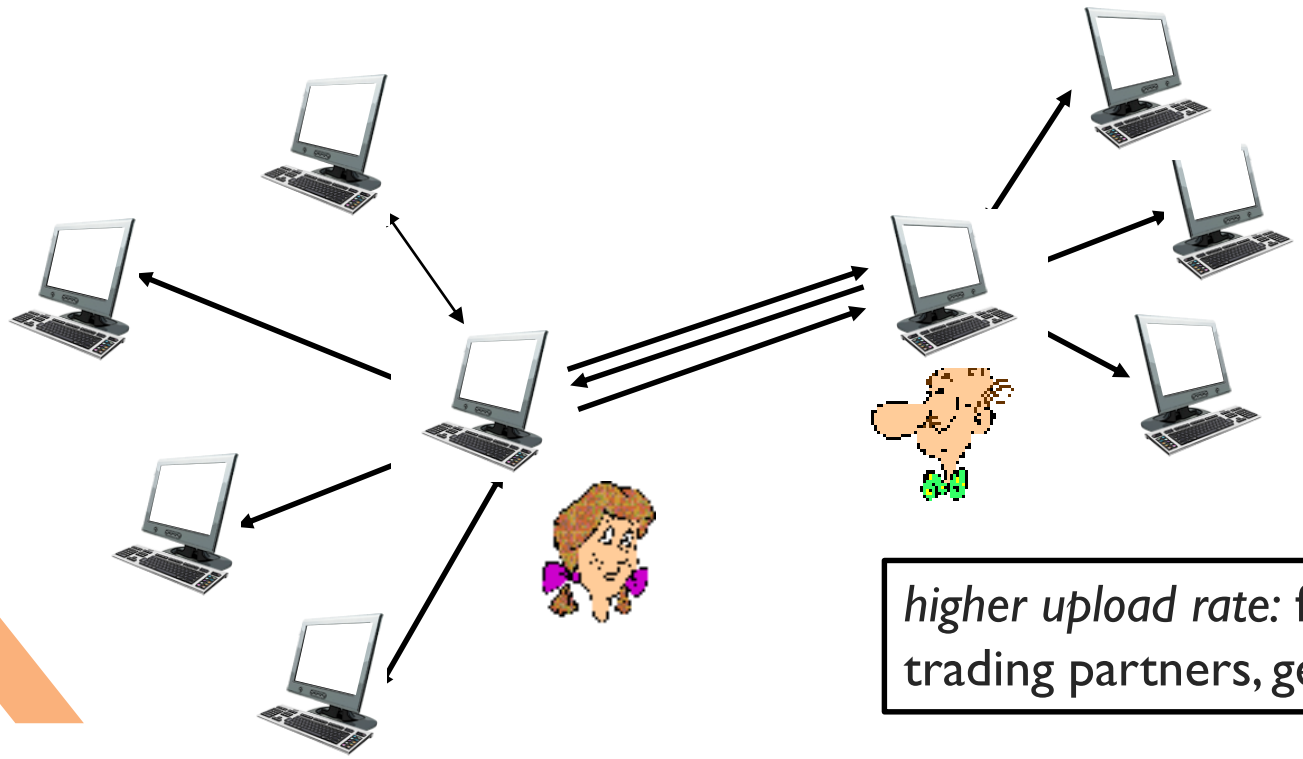
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
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

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

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

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



higher upload rate: find better trading partners, get file faster !

Distributed Hash Table (DHT)

- Hash table
- DHT paradigm
- Circular DHT and overlay networks
- Peer churn

Simple Database

Simple database with (key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	…….
Lisa Kobayashi	177-23-0199

Hash Table

- More of key
- key = hash(original key)

convenient to store and search on numerical representation

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		……
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

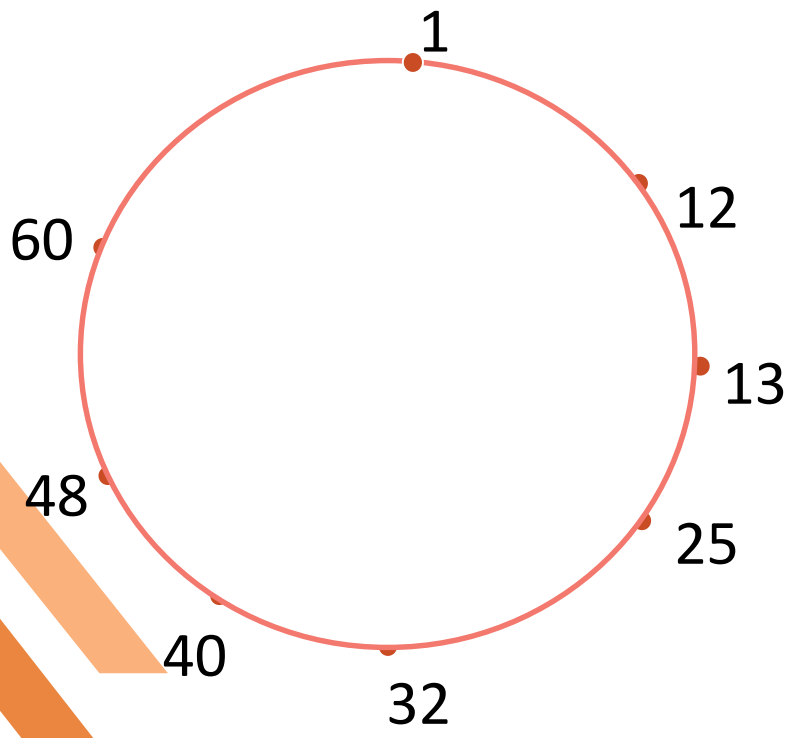
- Distribute (key, value) pairs over millions of peers
  - pairs are evenly distributed over peers
- Any peer can query database with a key
  - database returns value for the key
  - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

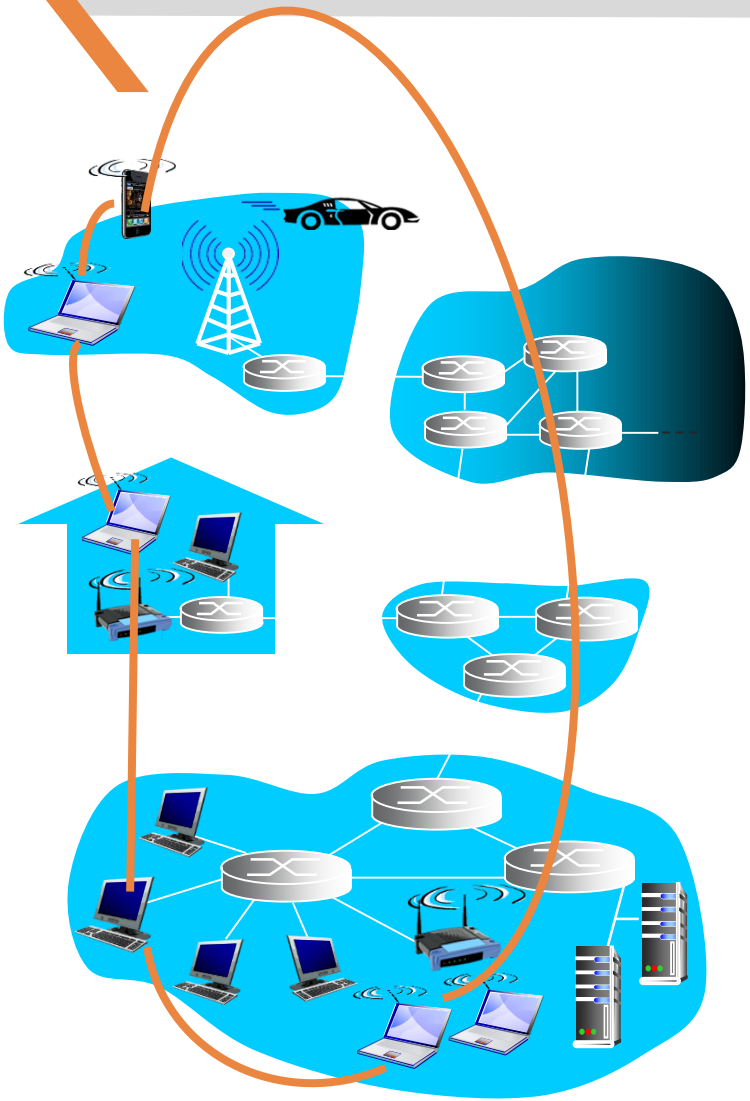
Assign key-value pairs to peers

- rule: assign key-value pair to the peer that has the *closest* ID.
- convention: closest is the *immediate successor* of the key.
- e.g., ID space {0,1,2,3,…,63}
- suppose 8 peers: 1,12,13,25,32,40,48,60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

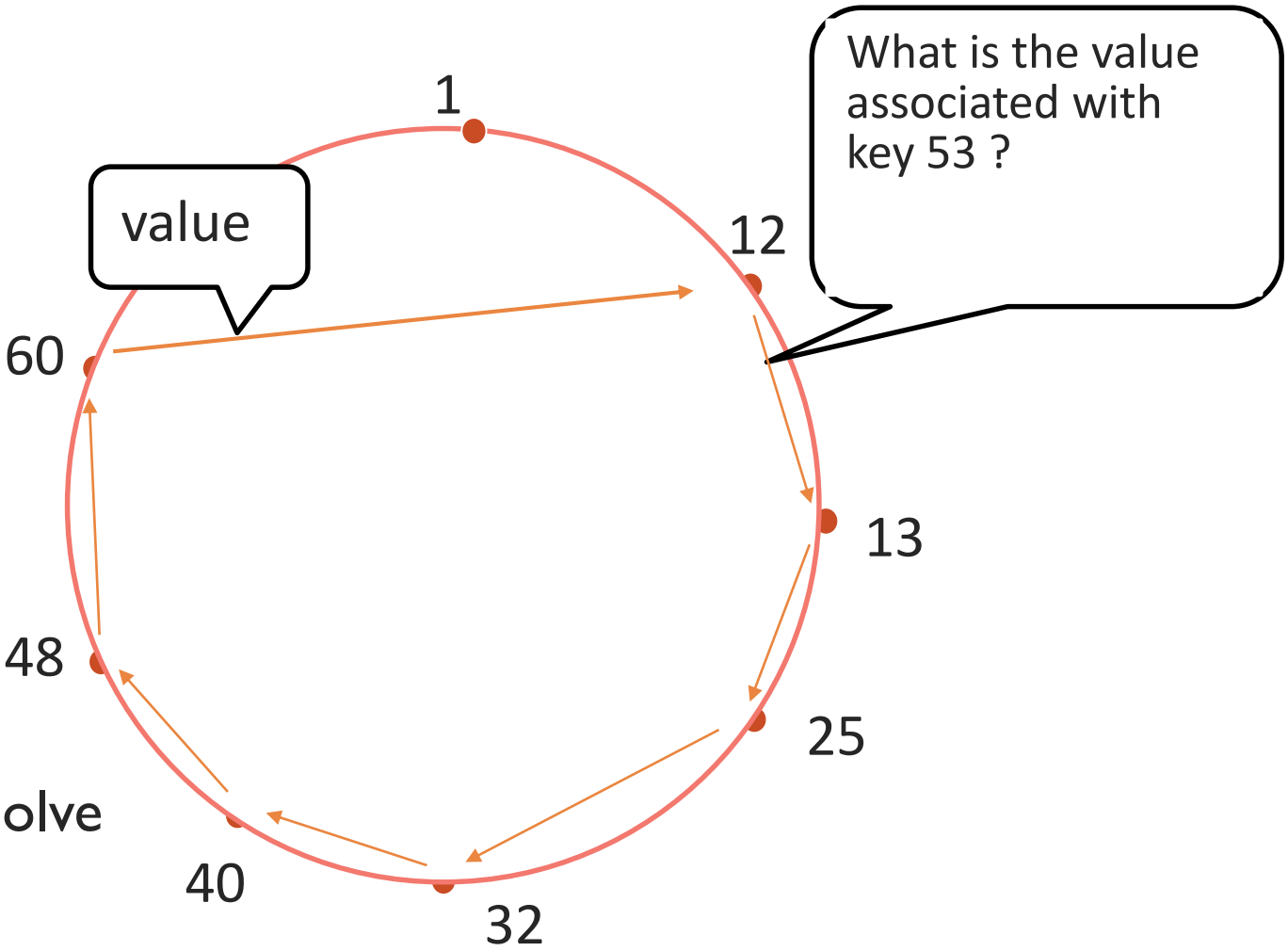
Circular DHT

- each peer *only* aware of immediate successor and predecessor.



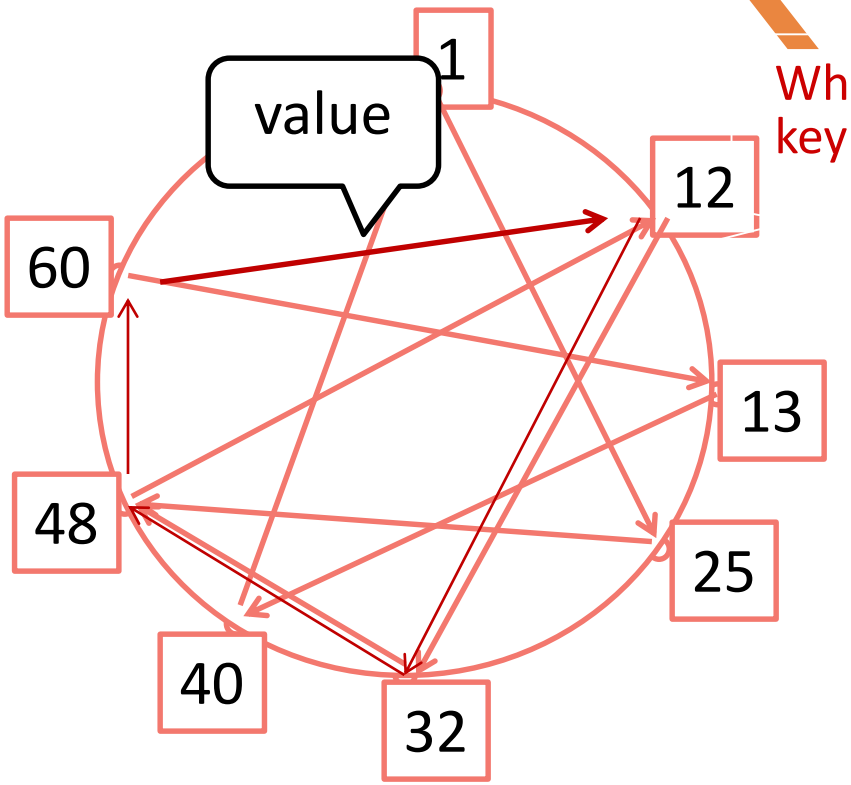


Resolving a query



O(N) messages on avgerage to resolve query, when there are *N* peers

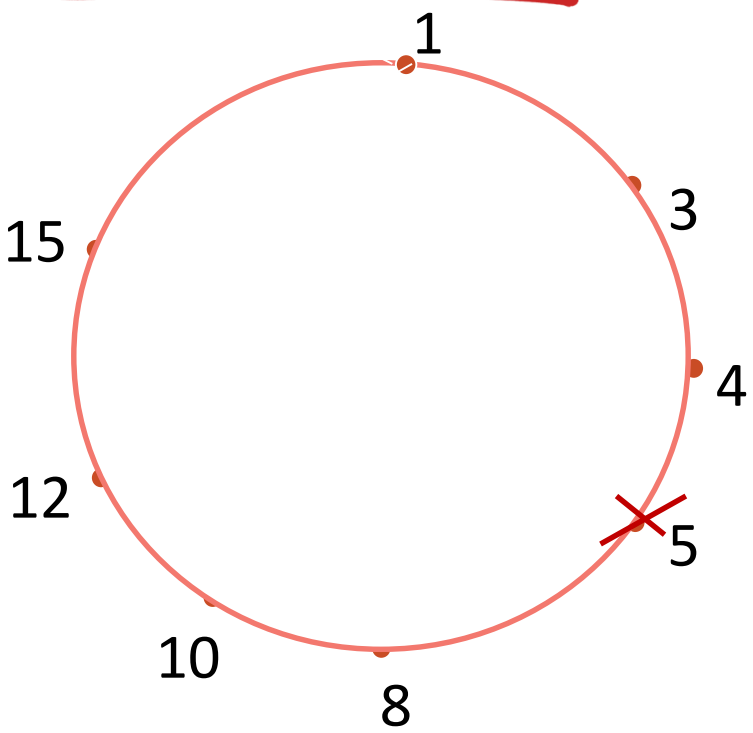
Circular DHT with shortcuts



What is the value for key 53

- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.
- possible to design shortcuts with *O(log N)* neighbors, *O(log N)* messages in query

Peer churn

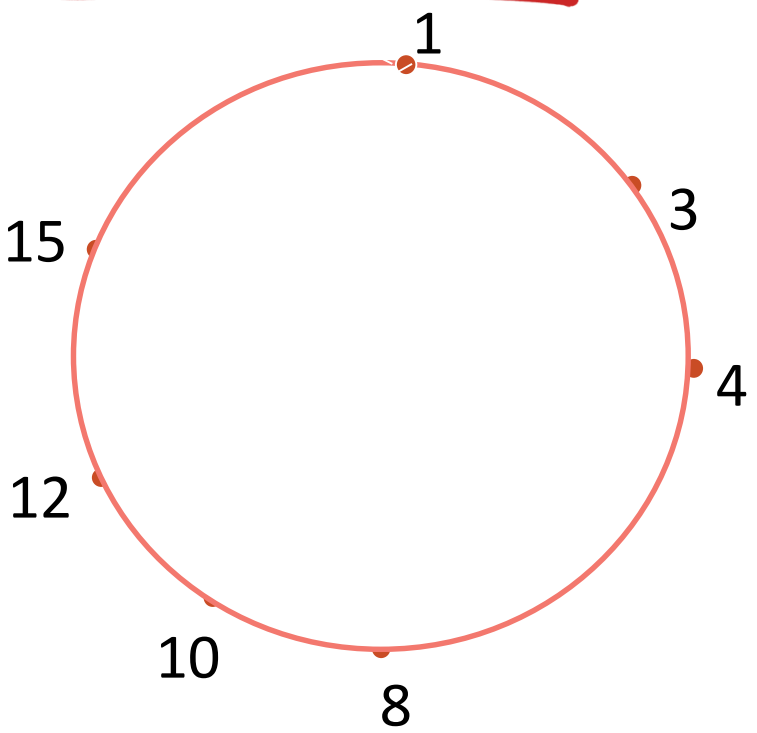


handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

Peer churn



handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

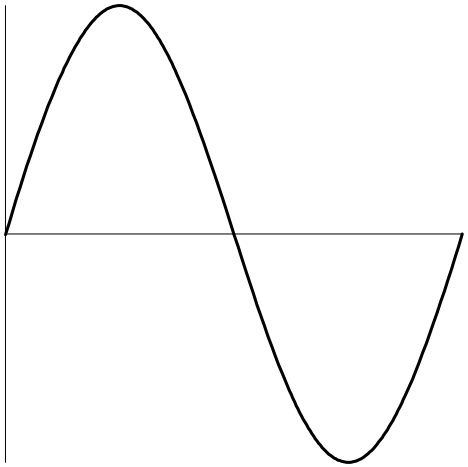
Multimedia Applications

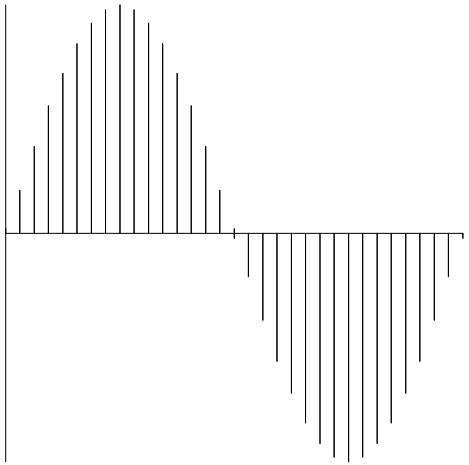
- Just like the traditional applications described earlier in this chapter, multimedia applications such as telephony and videoconferencing need their own protocols.
- We have already seen a number of protocols that multimedia applications use.
- The Real-Time Transport Protocol (RTP) provides many of the functions that are common to multimedia applications such as conveying timing information and identifying the coding schemes and media types of an application.

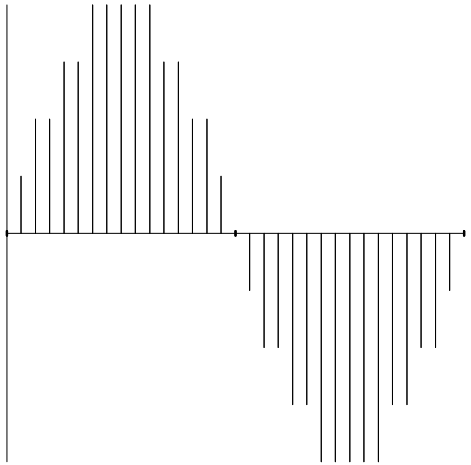
Multimedia Applications

- The Resource Reservation Protocol, RSVP can be used to request the allocation of resources in the network so that the desired quality of service (QoS) can be provided to an application.
- In addition to these protocols for multimedia transport and resource allocation, many multimedia applications also need a signaling or *session control protocol.*
- For example, suppose that we wanted to be able to make telephone calls across the internet ("voice over IP" or VOIP).

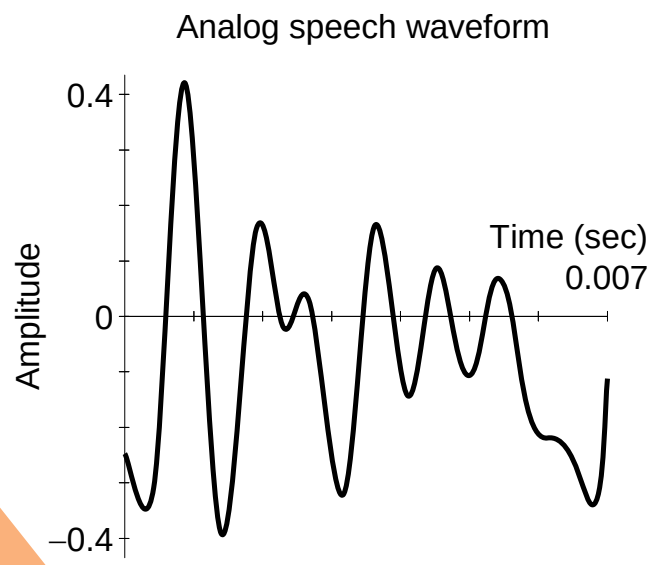
Signal Digitalization

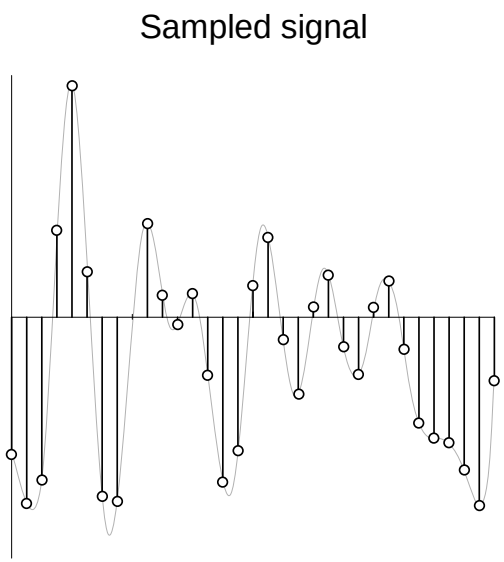


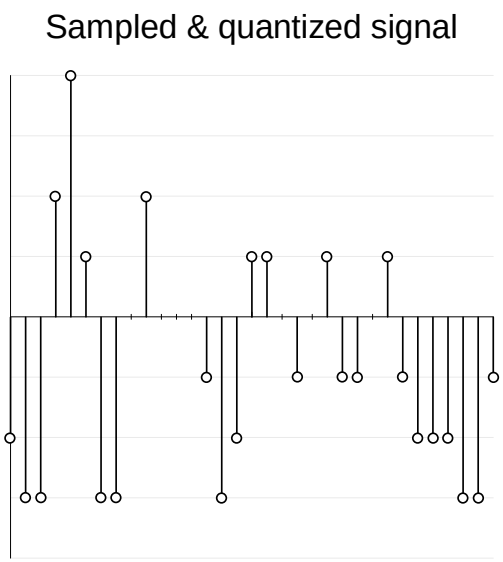




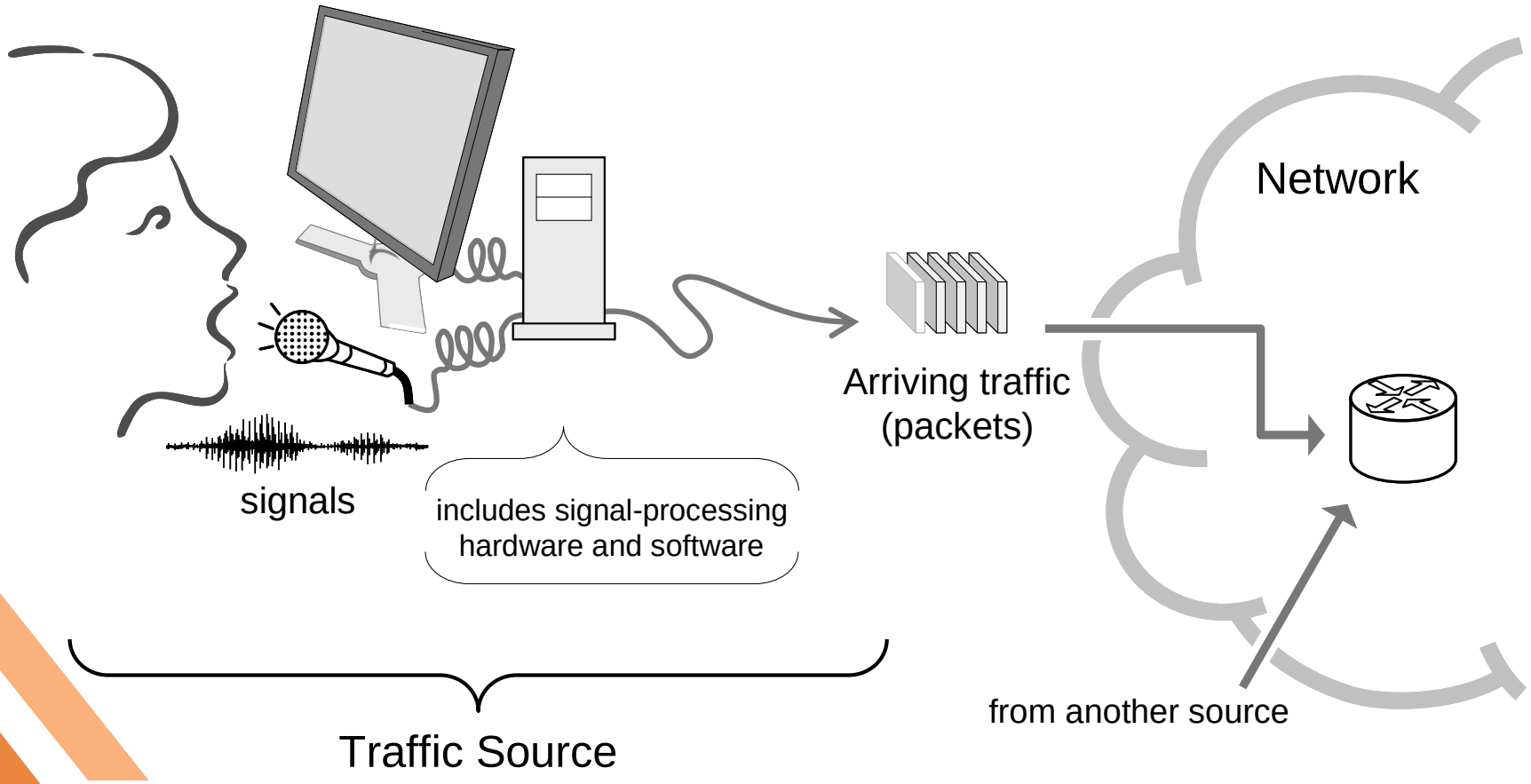
Speech Signal Digitalization







What is a Traffic Source



Multimedia Applications

- Session Control and Call Control (SDP, SIP, H.323)
  - To understand some of the issues of session control, consider the following problem.
    - Suppose you want to hold a videoconference at a certain time and make it available to a wide number of participants. Perhaps you have decided to encode the video stream using the MPEG-2 standard, to use the multicast IP address 224.1.1.1 for transmission of the data, and to send it using RTP over UDP port number 4000.
    - How would you make all that information available to the intended participants?
    - One way would be to put all that information in an email and send it out, but ideally there should be a standard format and protocol for disseminating this sort of information.

Multimedia Applications

- Session Control and Call Control (SDP, SIP, H.323)
  - The IETF has defined protocols for just this purpose. The protocols that have been defined include
    - SDP (Session Description Protocol)
    - SAP (Session Announcement Protocol)
    - SIP (Session Initiation Protocol)
    - SCCP (Simple Conference Control Protocol)

Multimedia Applications

- Session Description Protocol (SDP)
  - The Session Description Protocol (SDP) is a rather general protocol that can be used in a variety of situations and is typically used in conjunction with one or more other protocols (e.g., SIP).
  - It conveys the following information:
    - The name and purpose of the session
    - Start and end times for the session
    - The media types (e.g. audio, video) that comprise the session
    - Detailed information needed to receive the session (e.g. the multicast address to which data will be sent, the transport protocol to be used, the port numbers, the encoding scheme, etc.)

Multimedia Applications

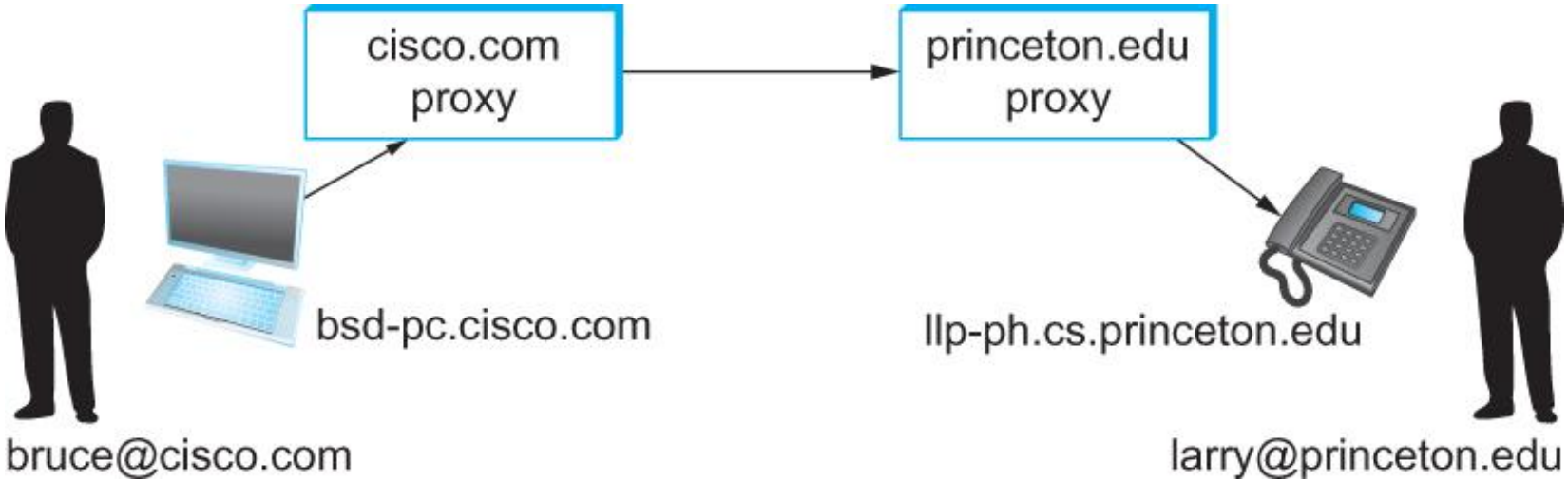
- SIP
  - SIP is an application layer protocol that bears a certain resemblance to HTTP, being based on a similar request/response model.
  - However, it is designed with rather different sorts of applications in mind, and thus provides quite different capabilities than HTTP.

Multimedia Applications

- SIP
  - The capabilities provided by SIP can be grouped into five categories:
    - User location: determining the correct device with which to communicate to reach a particular user;
    - User availability: determining if the user is willing or able to take part in a particular communication session;
    - User capabilities: determining such items as the choice of media and coding scheme to use;
    - Session setup: establishing session parameters such as port numbers to be used by the communicating parties;
    - Session management: a range of functions including transferring sessions (e.g. to implement “call forwarding”) and modifying session parameters.

Multimedia Applications

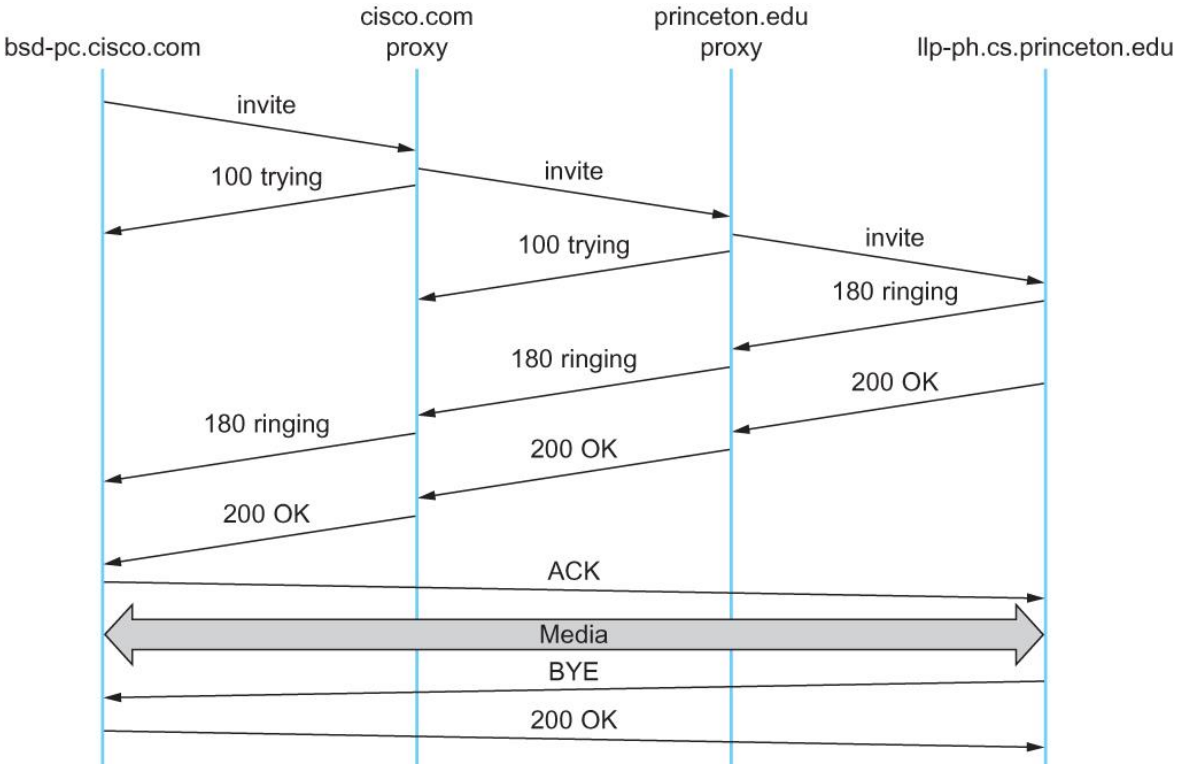
- SIP



Establishing communication through SIP proxies.

Multimedia Applications

- SIP



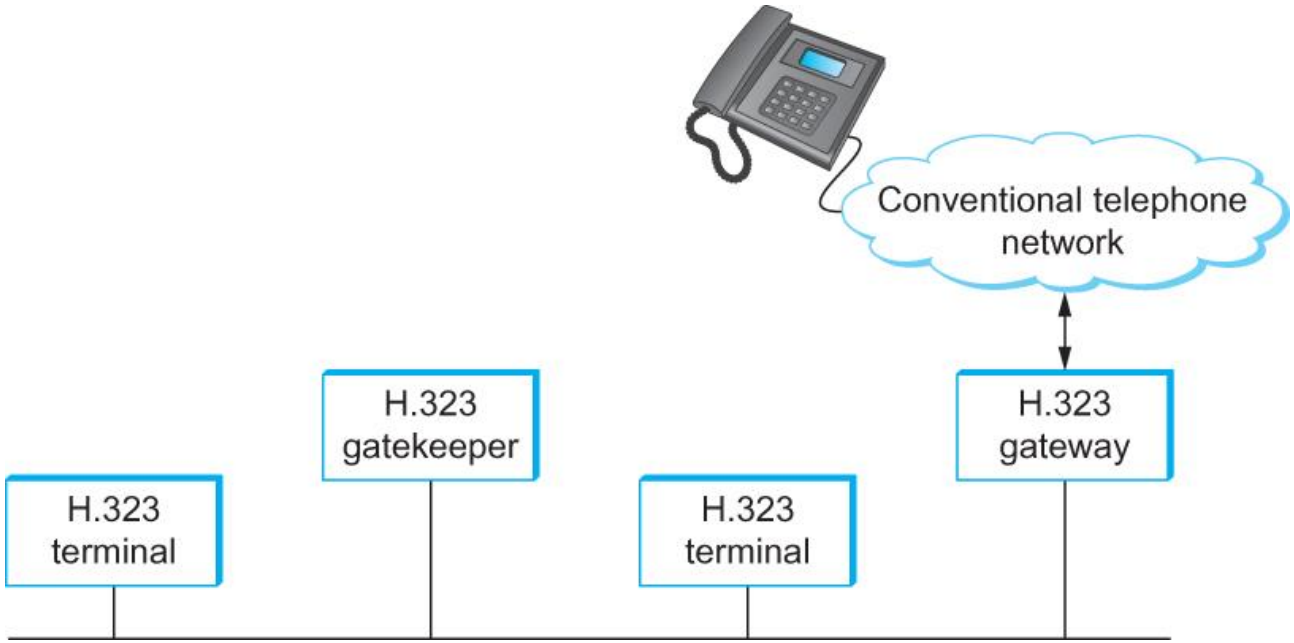
Message flow for a basic SIP session

Multimedia Applications

- H.323
  - The ITU has also been very active in the call control area, which is not surprising given its relevance to telephony, the traditional realm of that body.
  - Fortunately, there has been considerable coordination between the IETF and the ITU in this instance, so that the various protocols are somewhat interoperable.
  - The major ITU recommendation for multimedia communication over packet networks is known as H.323, which ties together many other recommendations, including H.225 for call control.
  - The full set of recommendations covered by H.323 runs to many hundreds of pages, and the protocol is known for its complexity

Multimedia Applications

- H.323



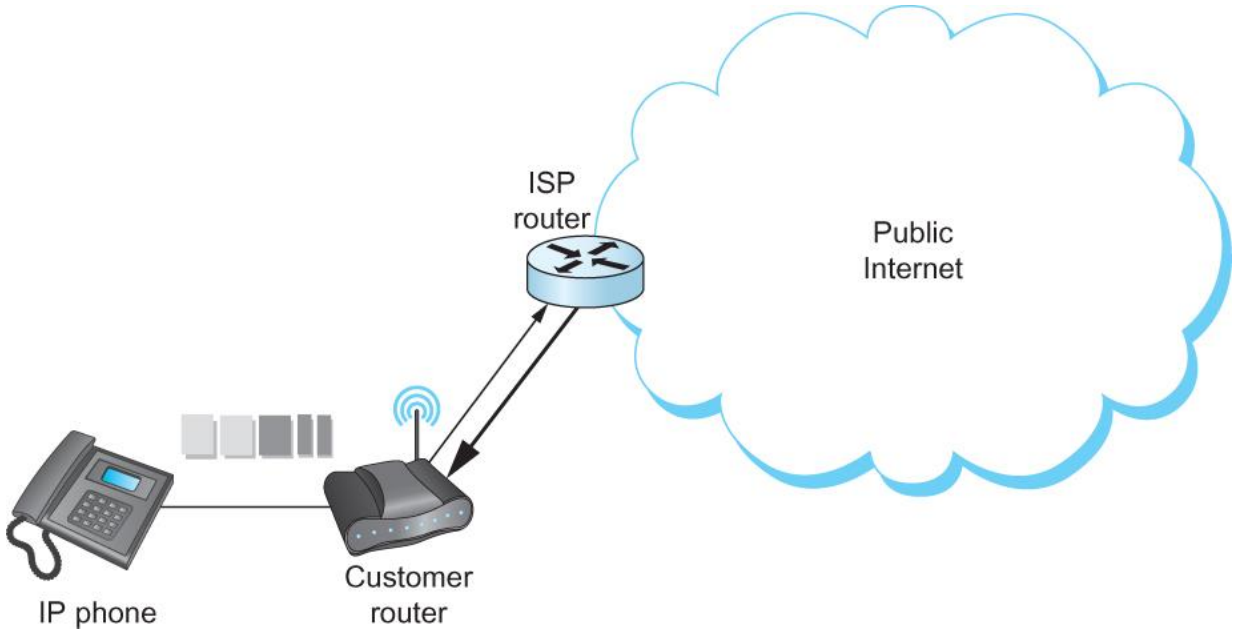
Devices in an H.323 network.

Multimedia Applications

- Resource Allocation for Multimedia Applications
  - As we have just seen, session control protocols like SIP and H.323 can be used to initiate and control communication in multimedia applications, while RTP provides transport level functions for the data streams of the applications.
  - A final piece of the puzzle in getting multimedia applications to work is making sure that suitable resources are allocated inside the network to ensure that the quality of service needs of the application are met.
  - Differentiated Services can be used to provide fairly basic and scalable resource allocation to applications.
  - A multimedia application can set the DSCP (differentiated services code point) in the IP header of the packets that it generates in an effort to ensure that both the media and control packets receive appropriate quality of service.

Multimedia Applications

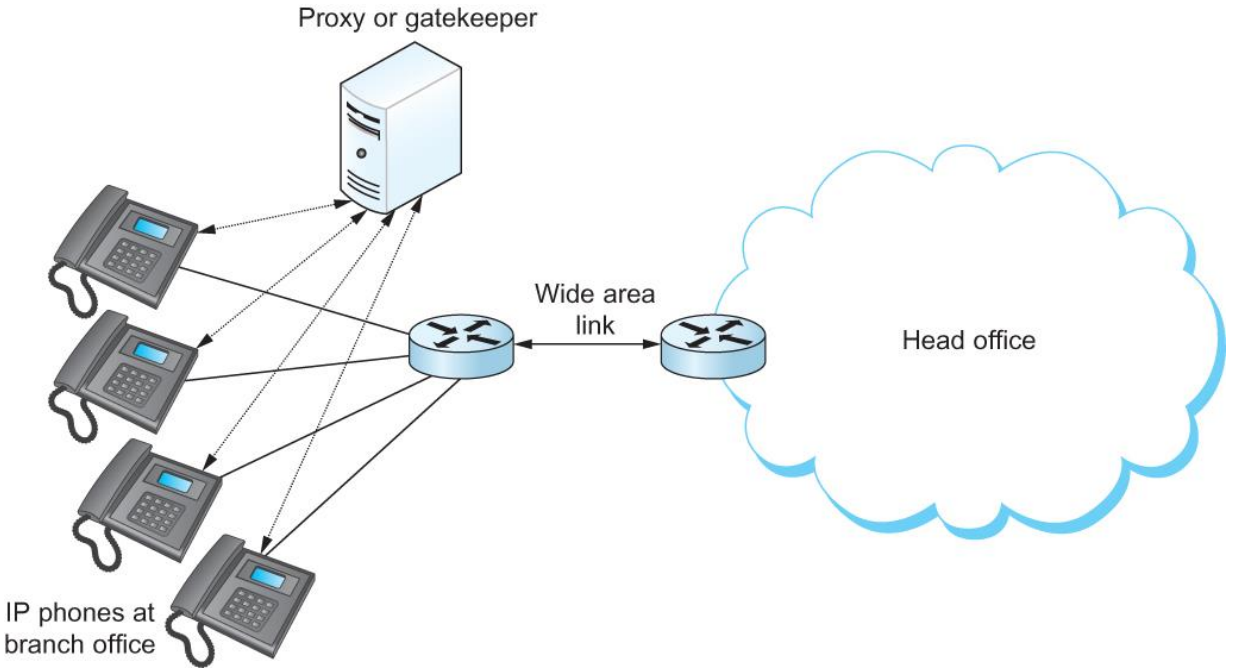
- Resource Allocation for Multimedia Applications



Differentiated Services applied to a VOIP application. DiffServ queueing is applied only on the upstream link from customer router to ISP.

Multimedia Applications

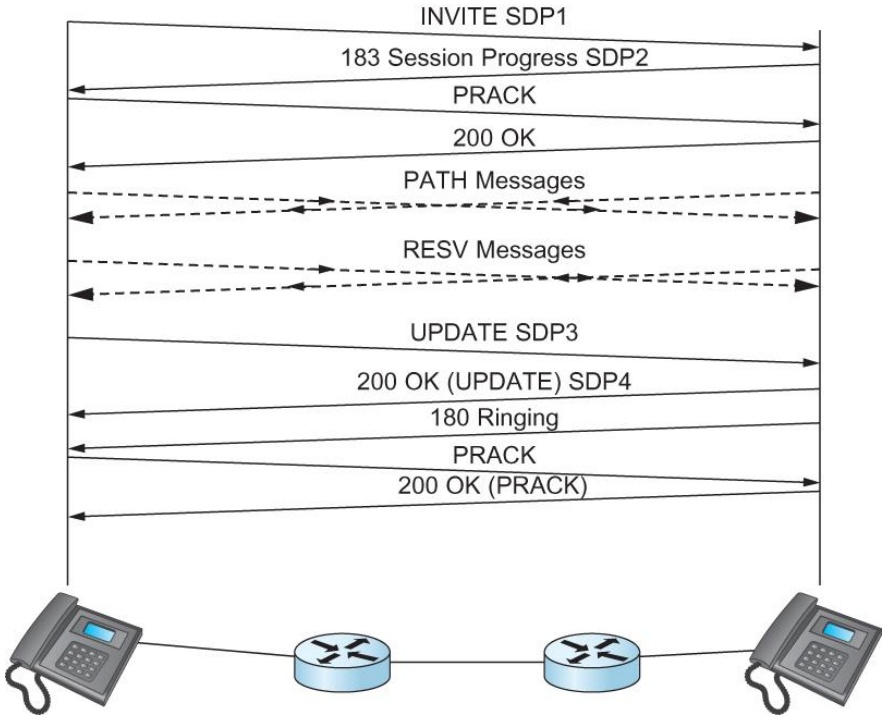
- Resource Allocation for Multimedia Applications



Admission control using session control protocol.

Multimedia Applications

- Resource Allocation for Multimedia Applications



Co-ordination of SIP signalling and resource reservationl.

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

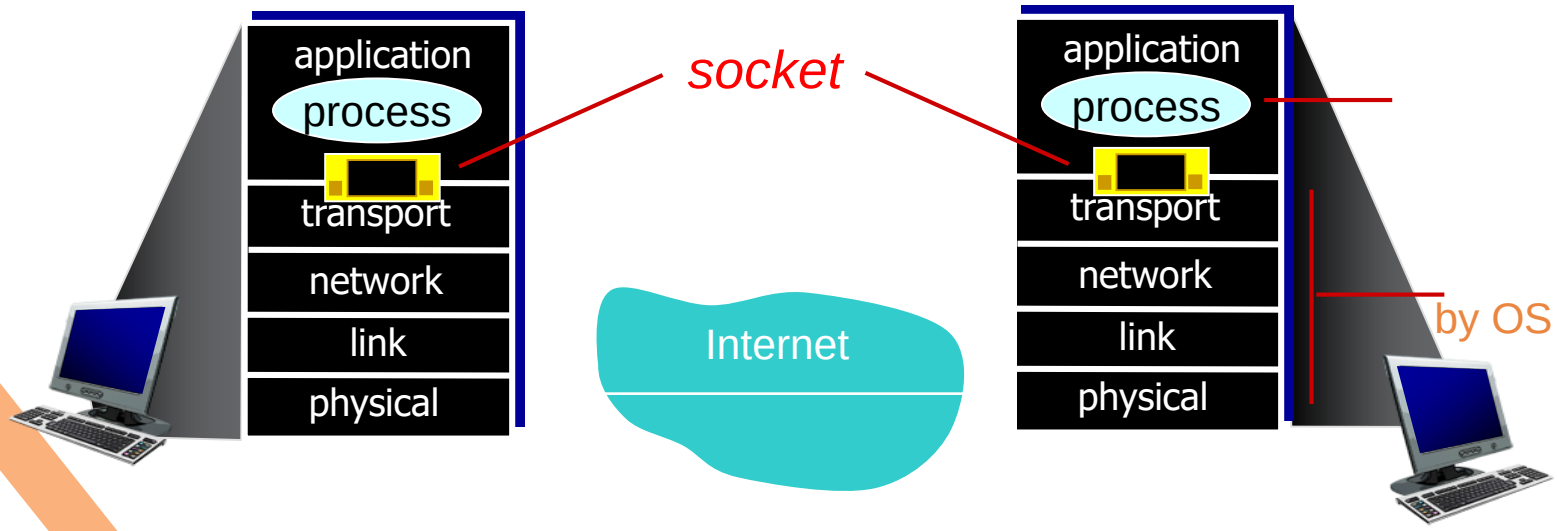
2.6 P2P applications

2.7 socket programming with UDP and TCP

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

Two socket types for two transport services:

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

Application Example:

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

Socket programming *with UDP*

UDP: no "connection" between client & server

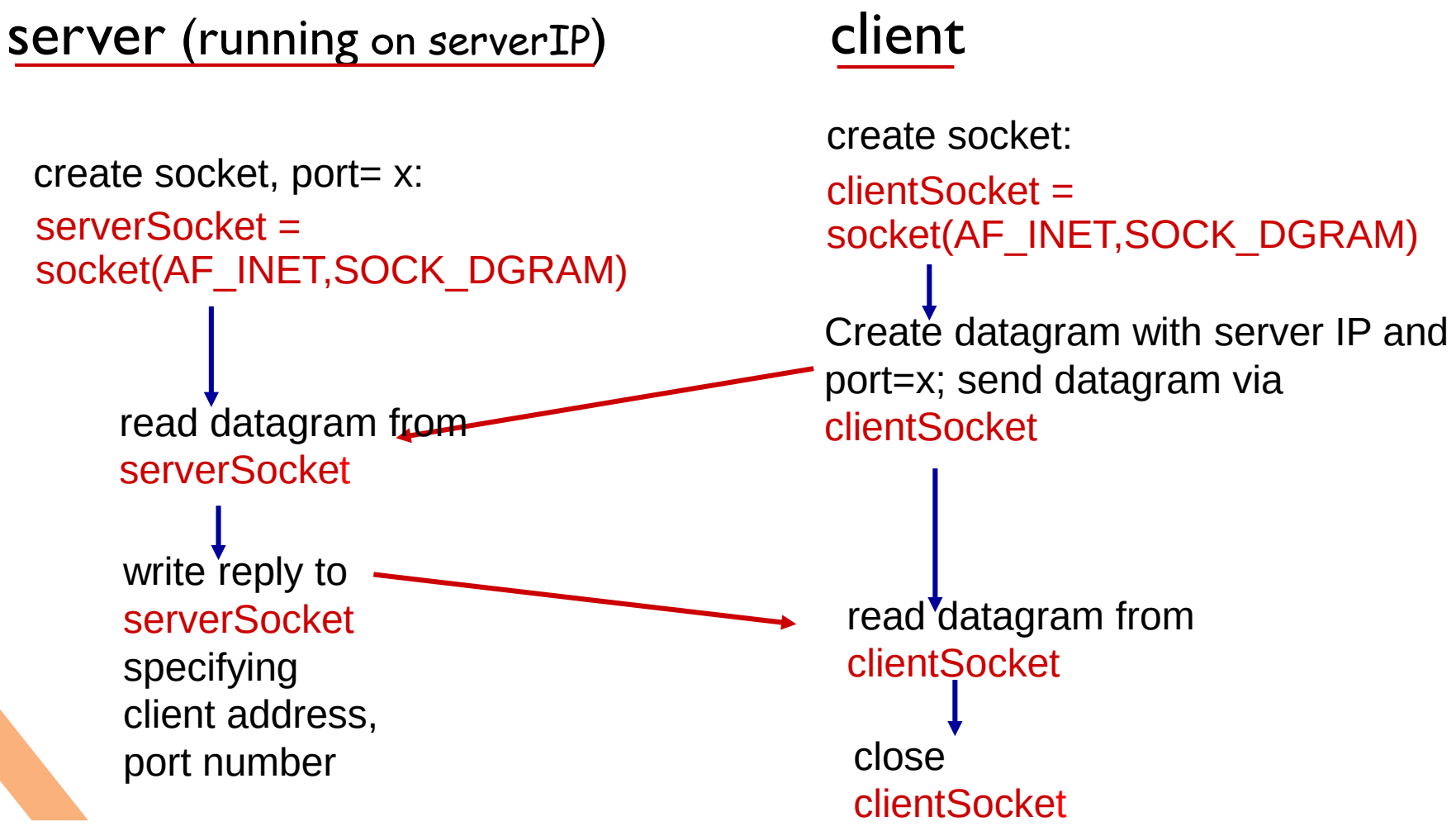
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

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

Application viewpoint:

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

Client/server socket interaction: UDP



Example app: UDP client

Python UDPClient

include Python's socket library

create UDP socket for server

get user keyboard input

Attach server name, port to message; send into socket

read reply characters from socket into string

print out received string and close socket

```
from socket import *

serverName = 'hostname'

serverPort = 12000

clientSocket = socket(socket.AF_INET,
                                        socket.SOCK_DGRAM)

message = raw_input('Input lowercase sentence:')

clientSocket.sendto(message,(serverName, serverPort))

modifiedMessage, serverAddress =
                                        clientSocket.recvfrom(2048)

print modifiedMessage

clientSocket.close()
```

Example app: UDP client

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

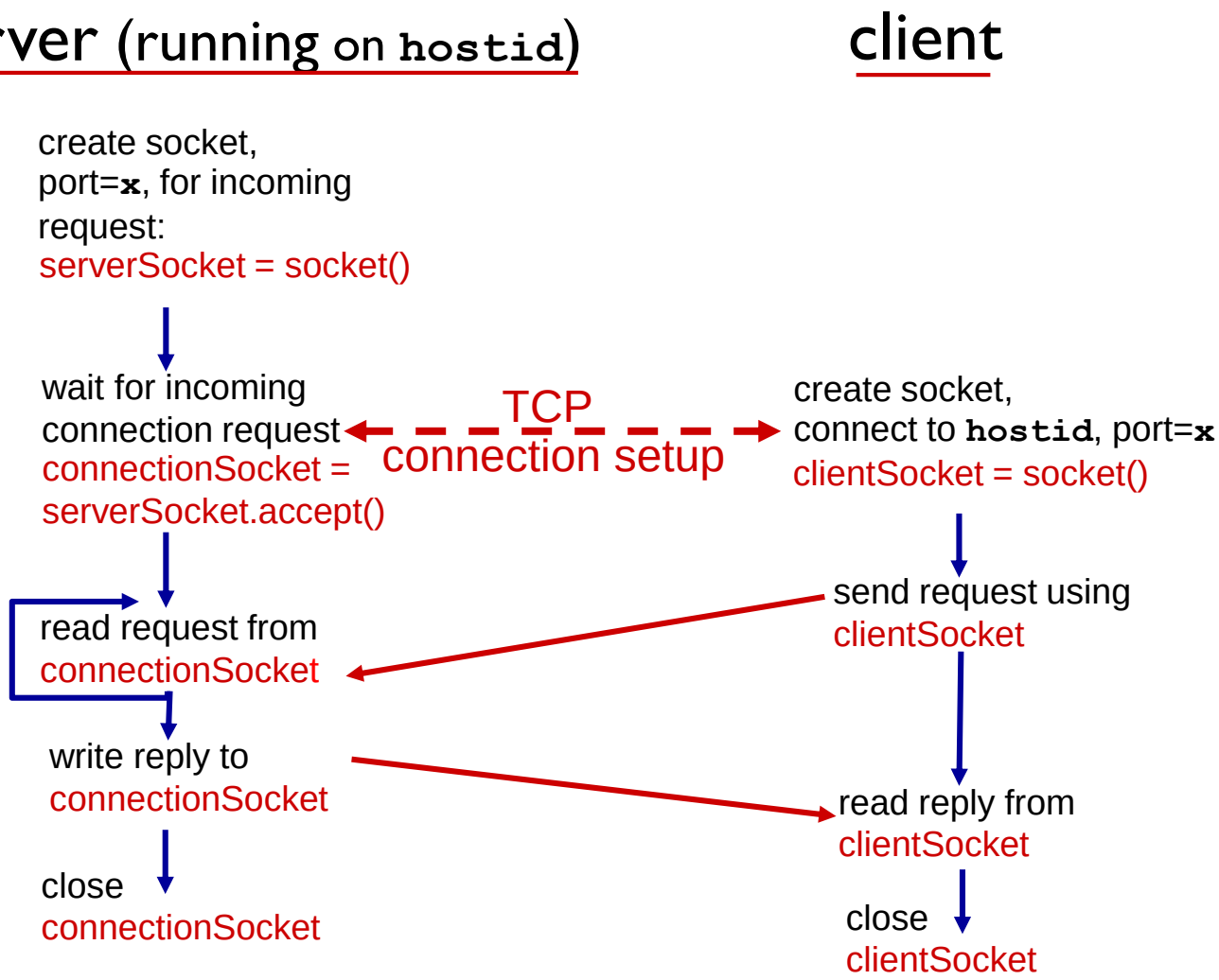
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

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

application viewpoint:

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

Client/server socket interaction: TCP



Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
      connectionSocket, addr = serverSocket.accept()

      sentence = connectionSocket.recv(1024)
      capitalizedSentence = sentence.upper()
      connectionSocket.send(capitalizedSentence)
      connectionSocket.close()
```

Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP

- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- socket programming: TCP, UDP sockets

Chapter 2: summary

most importantly: learned about protocols!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated

important themes:

- control vs. data msgs
  - in-band, out-of-band
- centralized vs. decentralized
- stateless vs. stateful
- reliable vs. unreliable msg transfer
- "complexity at network edge"