

---

# Network Services and Applications

EECS 489 Computer Networks

<http://www.eecs.umich.edu/courses/eecs489/w07>

Z. Morley Mao

Monday Jan 22, 2007

# Adminstrivia

---

- Homework 1 is due tomorrow – 1/23
- PA1 will be available tomorrow as well
  - A simplified Web server
  - You need to find a project partner for this assignment
- Reading assignment for this week
  - Chapter 3 of the book
  - You should have read Chapter 1 and 2.

# Recap from last lecture

---

- Nagle's algorithm
  - By default it is on: combines small packets into larger ones before sending to reduce header overhead
  - TCP\_NODELAY socket option disables it
- Internet routing is
  - policy driven, not load-sensitive, generally not QoS-based
- Email is not secure by default
  - Using HTTPS Web-based interface only provides one hop security from mail client to the mail server, not end-to-end security
- Port knocking allows a server to hide its port
  - Reduces overhead in rejecting illegitimate requests
- Akamai is a commercial company providing Web caching service
  - Using DNS-based redirection

# Nodal delay

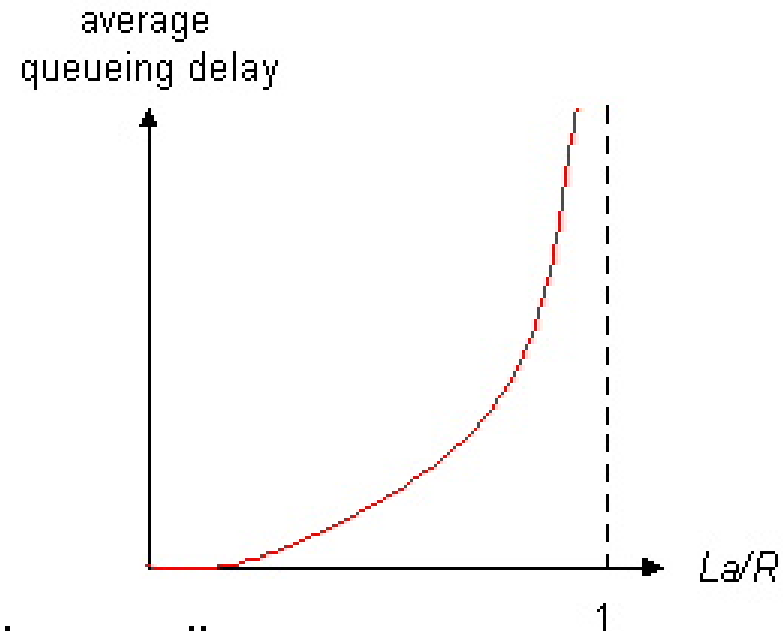
$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

- $d_{\text{proc}}$  = processing delay
  - typically a few microseconds or less
- $d_{\text{queue}}$  = queuing delay
  - depends on congestion
- $d_{\text{trans}}$  = transmission delay
  - $= L/R$ , significant for low-speed links
- $d_{\text{prop}}$  = propagation delay
  - a few microseconds to hundreds of msecs

# Queueing delay (revisited)

- $R$ =link bandwidth (bps)
- $L$ =packet length (bits)
- $a$ =average packet arrival rate

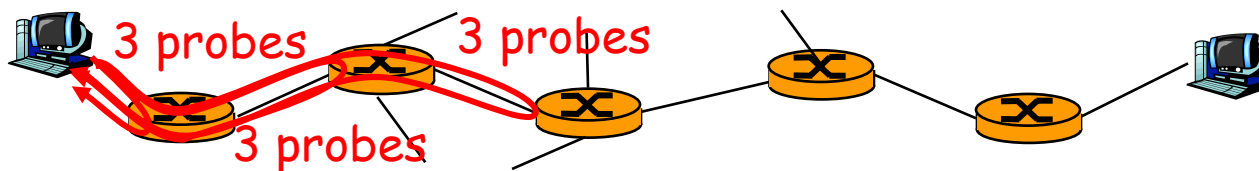
traffic intensity =  $La/R$



- $La/R \sim 0$ : average queueing delay small
- $La/R \rightarrow 1$ : delays become large
- $La/R > 1$ : more “work” arriving than can be serviced, average delay infinite!

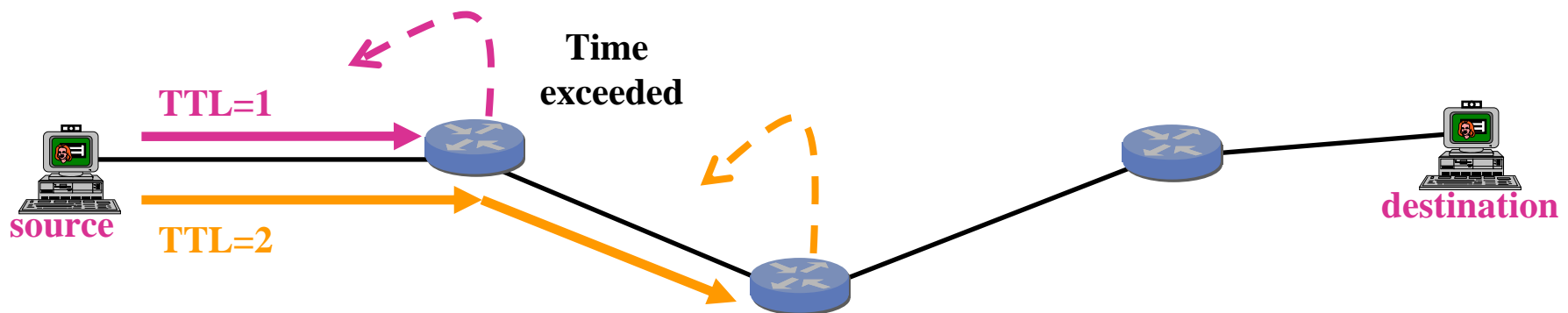
# “Real” Internet delays and routes

- 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.



# Traceroute: Measuring the Forwarding Path

- Time-To-Live field in IP packet header
  - Source sends a packet with a TTL of  $n$
  - Each router along the path decrements the TTL
  - “TTL exceeded” sent when TTL reaches 0
- Traceroute tool exploits this TTL behavior




**Send packets with TTL=1, 2, 3, ... and record source of “time exceeded” message**

# “Real” Internet delays and routes

traceroute: gaia.cs.umass.edu to www.eurecom.fr


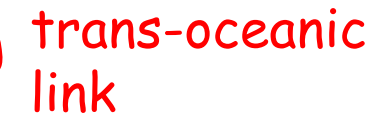
Three delay measurements from  
gaia.cs.umass.edu to cs-gw.cs.umass.edu



1 cs-gw (128.119.240.254) 1 ms 1 ms 2 ms  
2 border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145) 1 ms 1 ms 2 ms  
3 cht-vbns.gw.umass.edu (128.119.3.130) 6 ms 5 ms 5 ms  
4 jn1-at1-0-0-19.wor.vbns.net (204.147.132.129) 16 ms 11 ms 13 ms  
5 jn1-so7-0-0-0.wae.vbns.net (204.147.136.136) 21 ms 18 ms 18 ms  
6 abilene-vbns.abilene.ucaid.edu (198.32.11.9) 22 ms 18 ms 22 ms  
7 nycm-wash.abilene.ucaid.edu (198.32.8.46) 22 ms 22 ms 22 ms  
8 62.40.103.253 (62.40.103.253) 104 ms 109 ms 106 ms  
9 de2-1.de1.de.geant.net (62.40.96.129) 109 ms 102 ms 104 ms  
10 de.fr1.fr.geant.net (62.40.96.50) 113 ms 121 ms 114 ms  
11 renater-gw.fr1.fr.geant.net (62.40.103.54) 112 ms 114 ms 112 ms  
12 nio-n2.cssi.renater.fr (193.51.206.13) 111 ms 114 ms 116 ms  
13 nice.cssi.renater.fr (195.220.98.102) 123 ms 125 ms 124 ms  
14 r3t2-nice.cssi.renater.fr (195.220.98.110) 126 ms 126 ms 124 ms  
15 eurecom-valbonne.r3t2.ft.net (193.48.50.54) 135 ms 128 ms 133 ms  
16 194.214.211.25 (194.214.211.25) 126 ms 128 ms 126 ms  
17 \* \* \*  
18 \* \* \*  
19 fantasia.eurecom.fr (193.55.113.142) 132 ms 128 ms 136 ms

trans-oceanic link

\* means no response (probe lost, router not replying)





# Packet loss

---

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

# Protocol “Layers”

---

## Networks are complex!

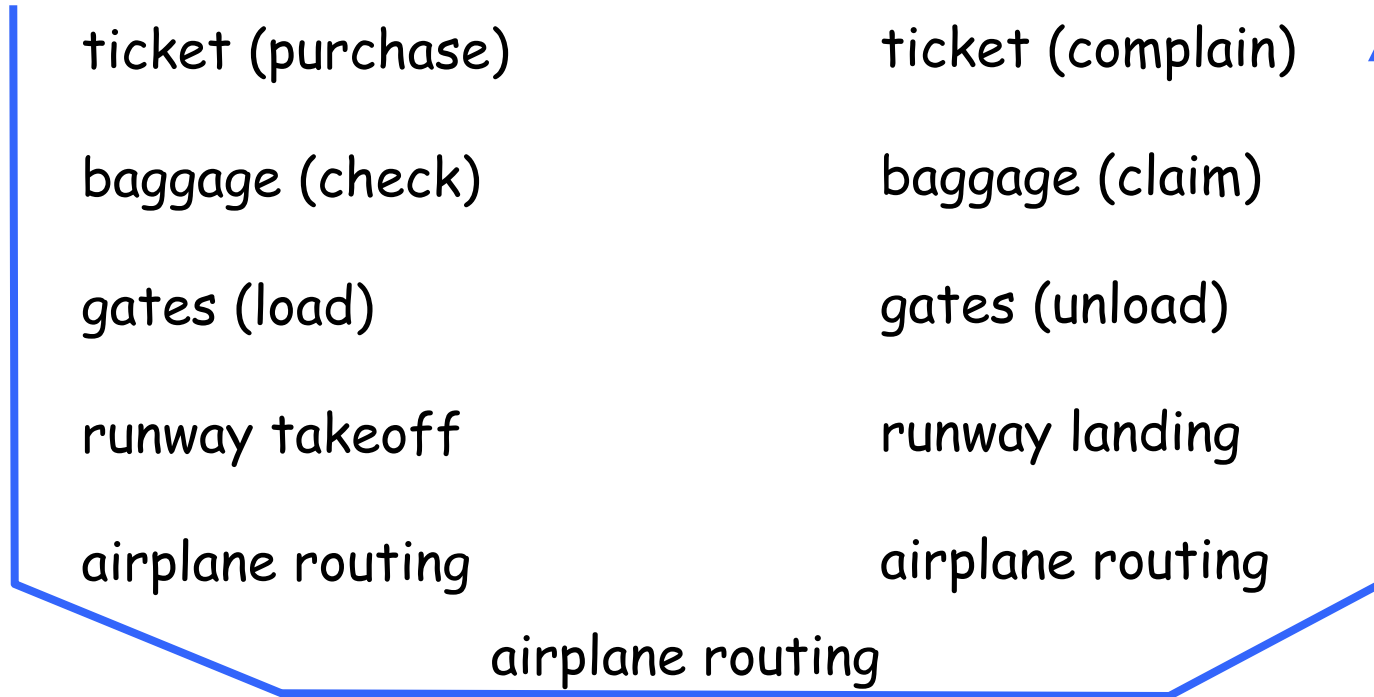
- many “pieces”:
  - hosts
  - routers
  - links of various media
  - applications
  - protocols
  - hardware, software

## Question:

Is there any hope of *organizing* structure of network?

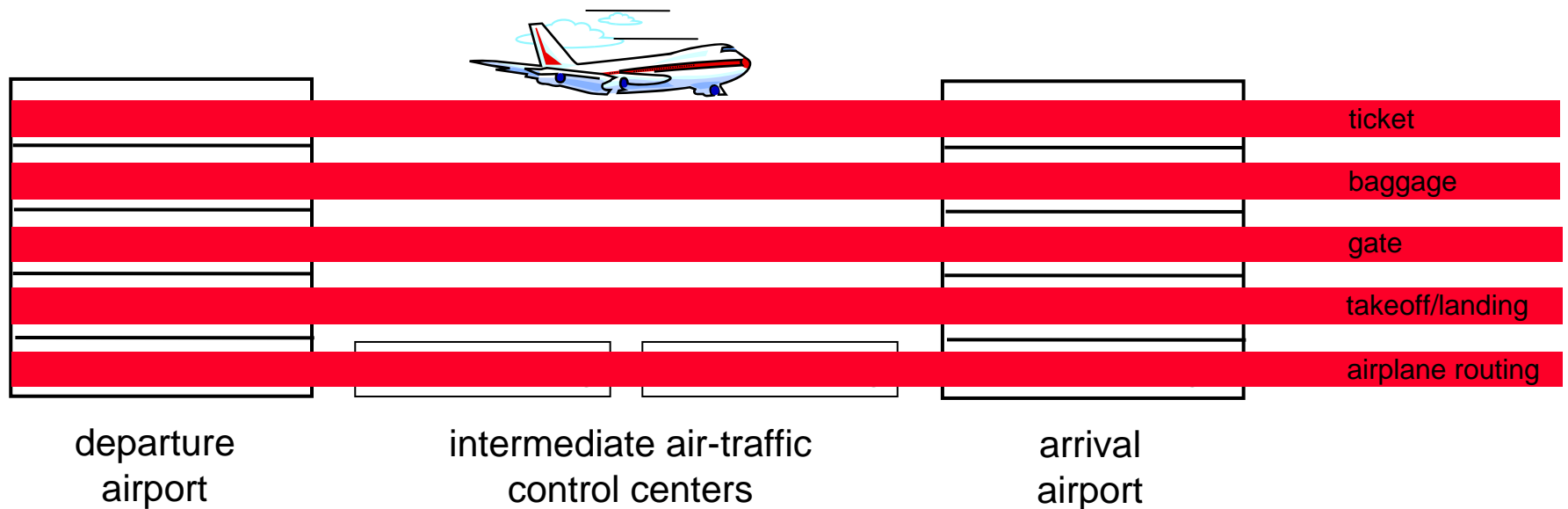
Or at least our discussion of networks?

# 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

# Why layering?

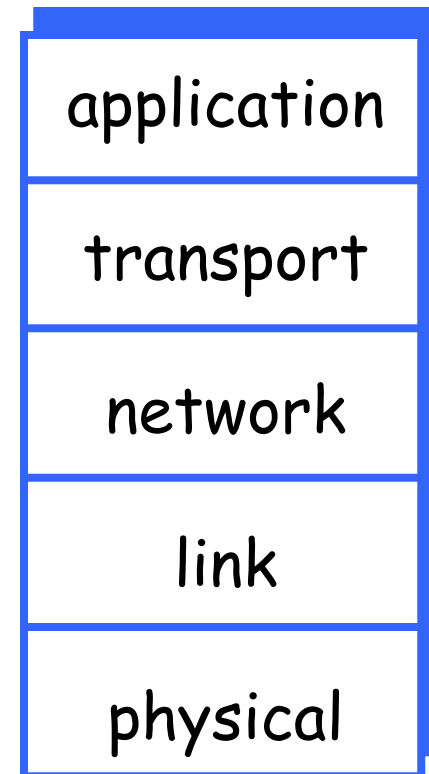
---

## Dealing with complex systems:

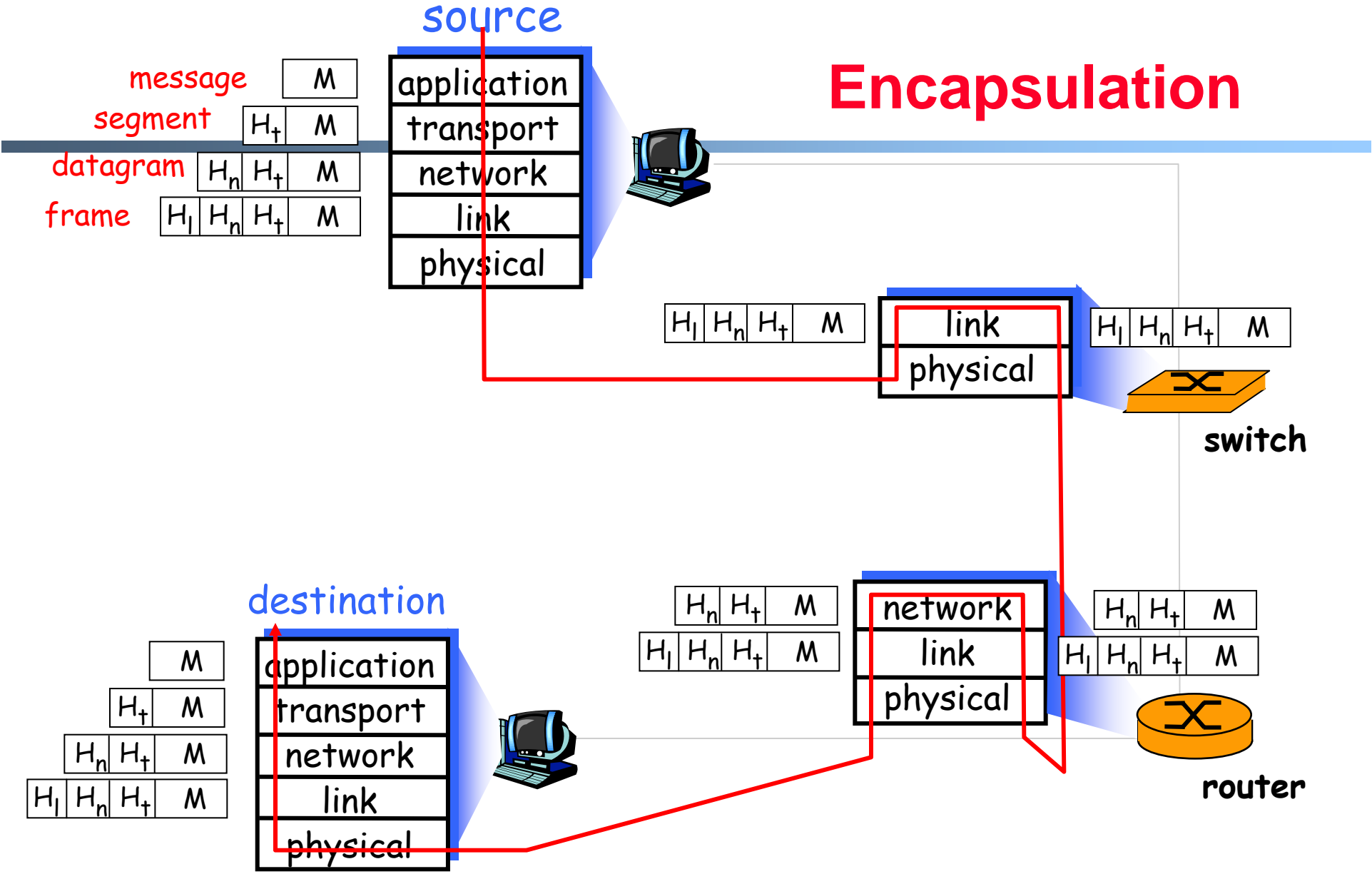
- explicit structure allows identification, relationship of complex system's pieces
  - layered **reference model** for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - e.g., change in gate procedure doesn't affect rest of system
- layering considered harmful?

# Internet protocol stack

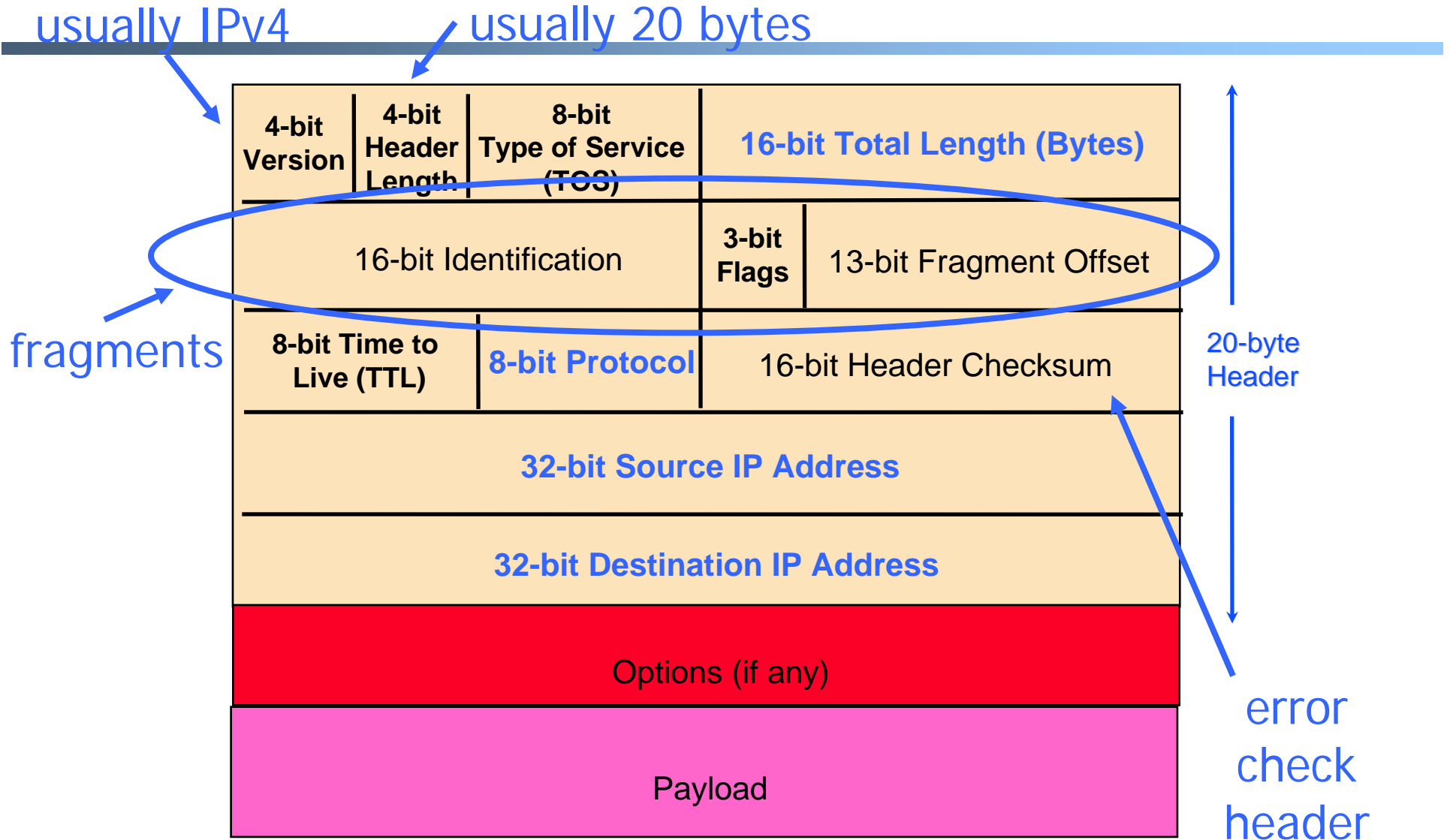
- **application:** supporting network applications
  - FTP, SMTP, STTP
- **transport:** host-host data transfer
  - TCP, UDP
- **network:** routing of datagrams from source to destination
  - IP, routing protocols
- **link:** data transfer between neighboring network elements
  - PPP, Ethernet
- **physical:** bits “on the wire”



# Encapsulation



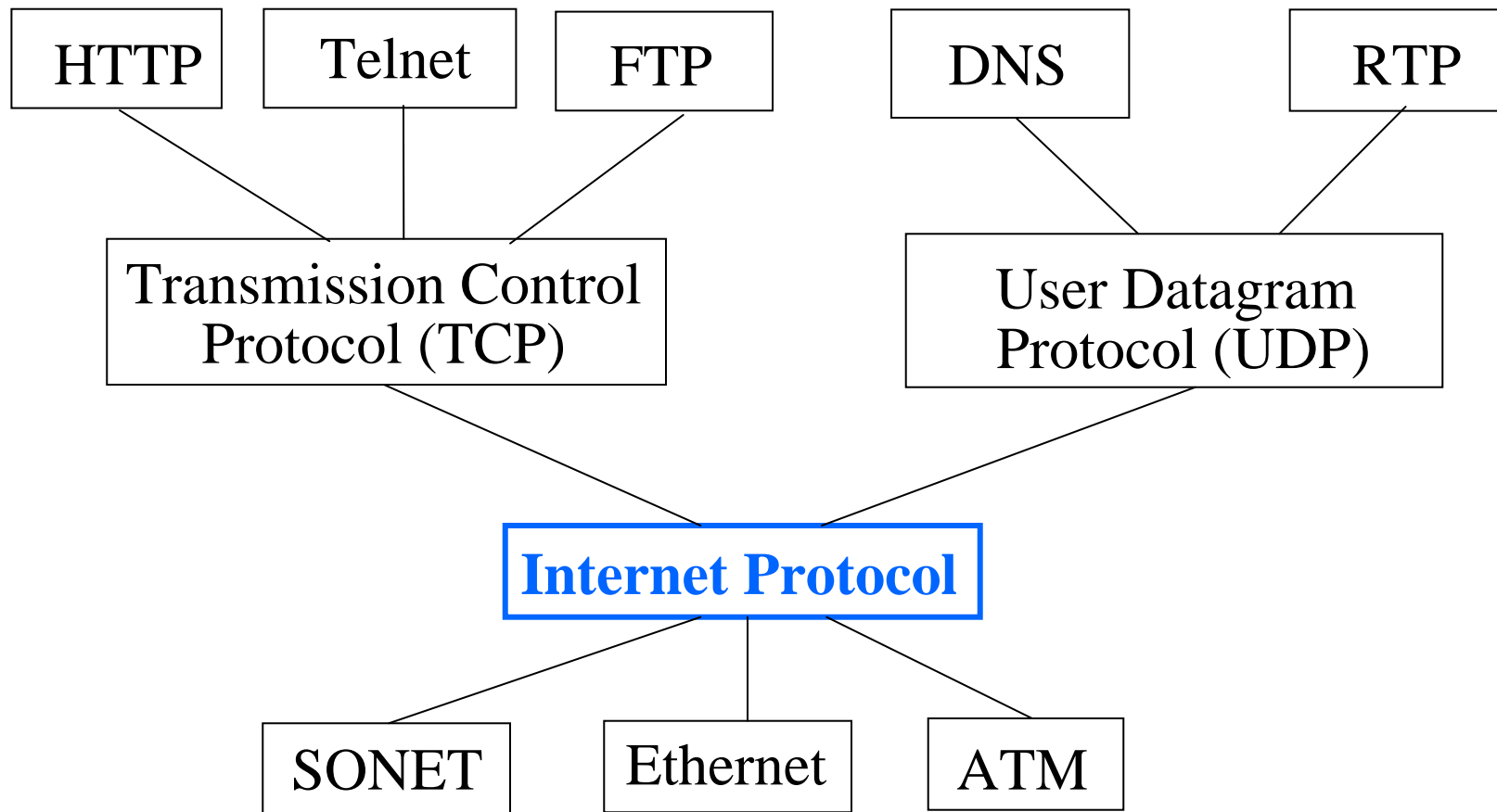
# IP Packet Structure





# Layering in the IP Protocols

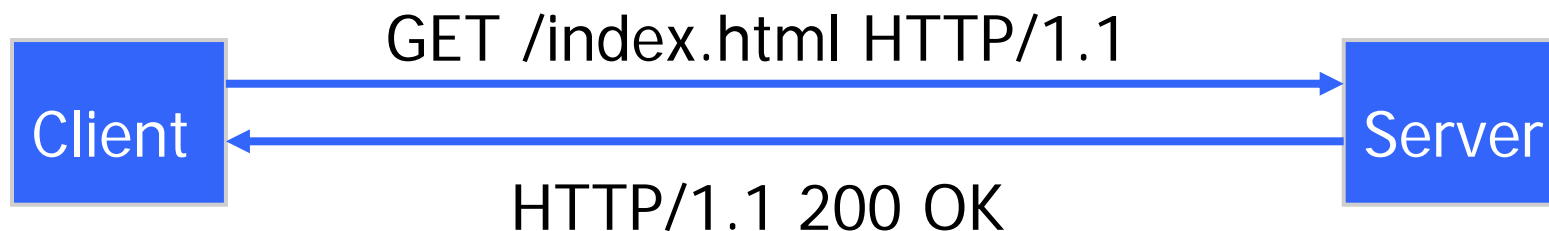
---



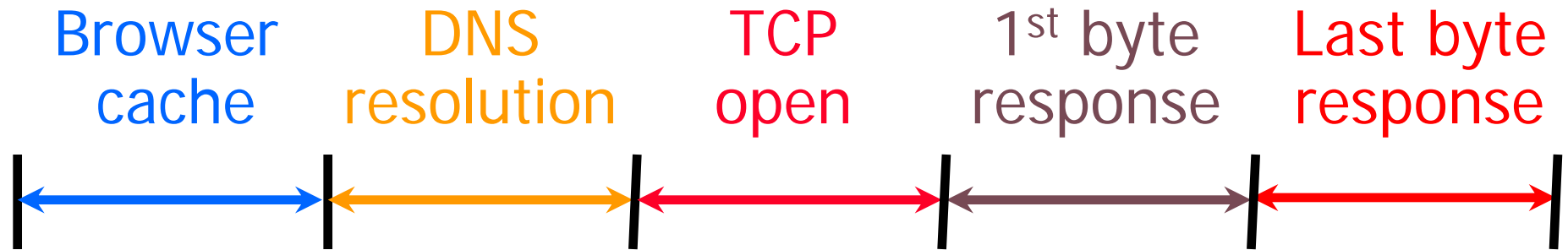
# Application-Layer Protocols

---

- Messages exchanged between applications
  - Syntax and semantics of the messages between hosts
  - Tailored to the specific application (e.g., Web, e-mail)
  - Messages transferred over transport connection (e.g., TCP)
- Popular application-layer protocols
  - Telnet, FTP, SMTP, NNTP, HTTP, ...



# Example: Many Steps in Web Download



## Sources of variability of delay

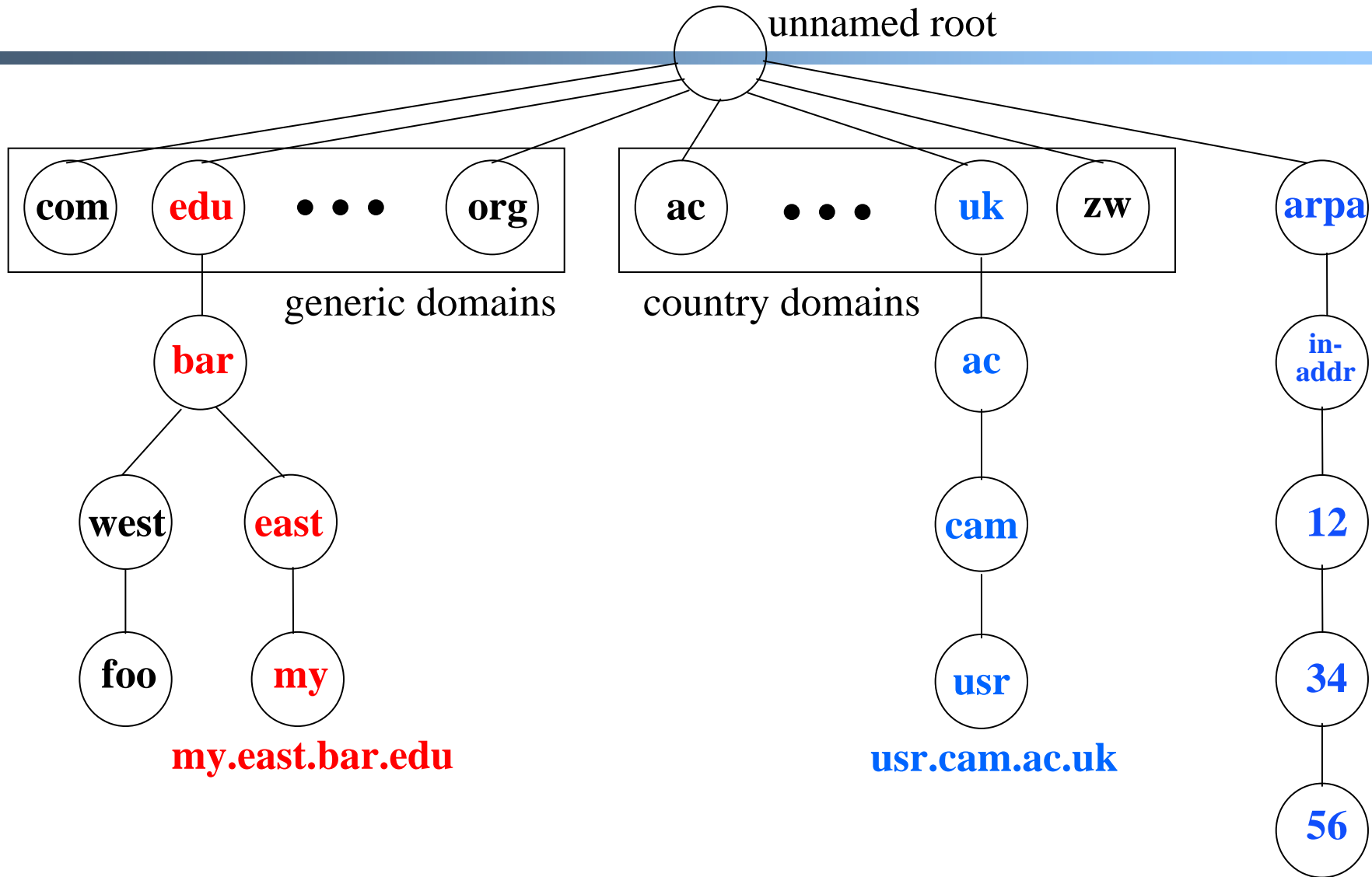
- Browser cache hit/miss, need for cache revalidation
- DNS cache hit/miss, multiple DNS servers, errors
- Packet loss, high RTT, server accept queue
- RTT, busy server, CPU overhead (e.g., CGI script)
- Response size, receive buffer size, congestion
- ... downloading embedded image(s) on the page

# Domain Name System (DNS)

---

- Properties of DNS
  - Hierarchical name space divided into zones
  - Translation of names to/from IP addresses
  - Distributed over a collection of DNS servers
- Client application
  - Extract server name (e.g., from the URL)
  - Invoke system call to trigger DNS resolver code
  - E.g., *gethostbyname()* on “www.foo.com”
- Server application
  - Extract client IP address from socket
  - Optionally invoke system call to translate into name
  - E.g., *gethostbyaddr()* on “12.34.158.5”

# Domain Name System

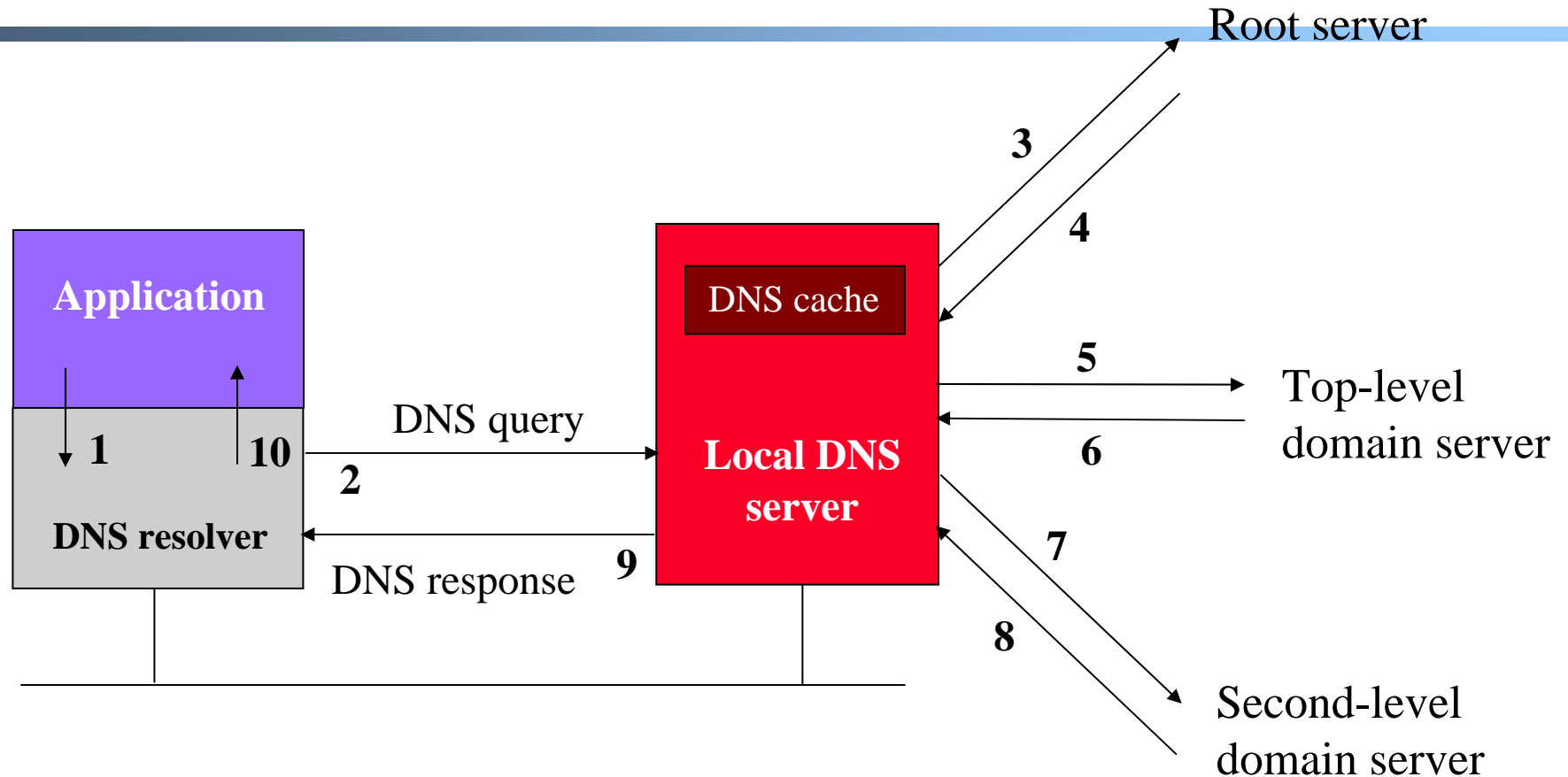


**my.east.bar.edu**

**usr.cam.ac.uk**

Mao W07 21 **12.34.56.0/24**

# DNS Resolver and Local DNS Server



**Caching based on a time-to-live (TTL) assigned by the DNS server responsible for the host name to reduce latency in DNS translation.**

# Web and HTTP

## First some jargon

- **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**
- **Example URL:**

www.someschool.edu / someDept/pic.gif

host name

path name



Have you heard of Google's "PageRank"?  
Is it susceptible to spoofing?

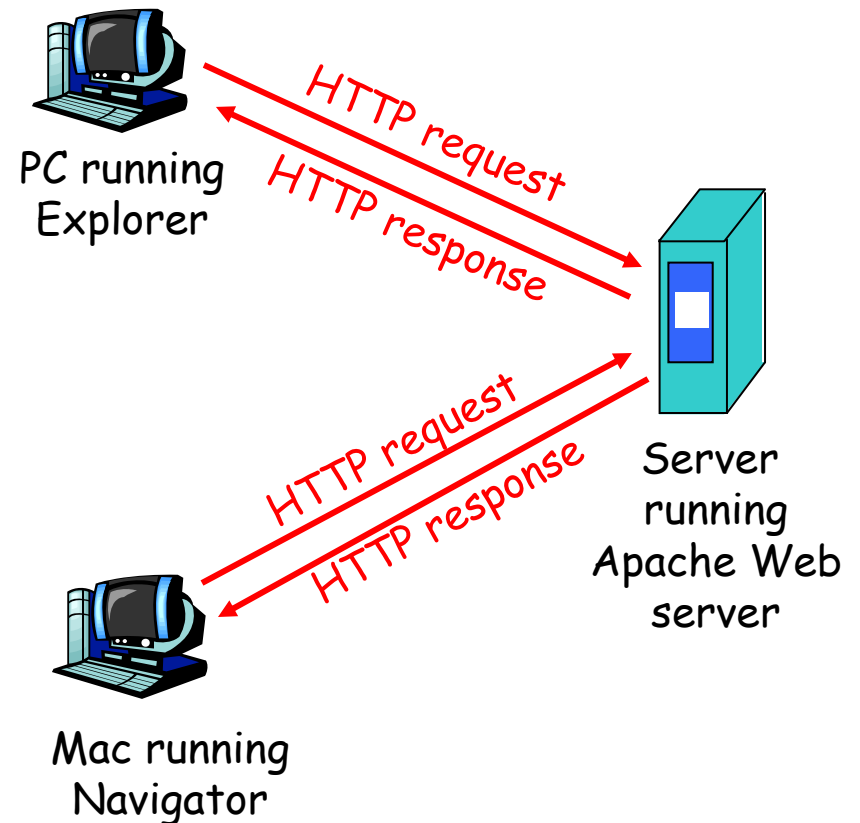
Mao W07

23

# HTTP overview

## HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client*: browser that requests, receives, "displays" Web objects
  - *server*: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068





# 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

Is it better to have a stateful protocol?

# HTTP connections

---

## Nonpersistent HTTP

- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

## Persistent HTTP

- Multiple objects can be sent over a single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

What is the advantage of persistent HTTP?  
Who needs to support this? Server or client?

# Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.index`

(contains text,  
references to 10  
jpeg images)

1a. **HTTP** client initiates a TCP connection to HTTP server (process) at `www.someSchool.edu` on port 80

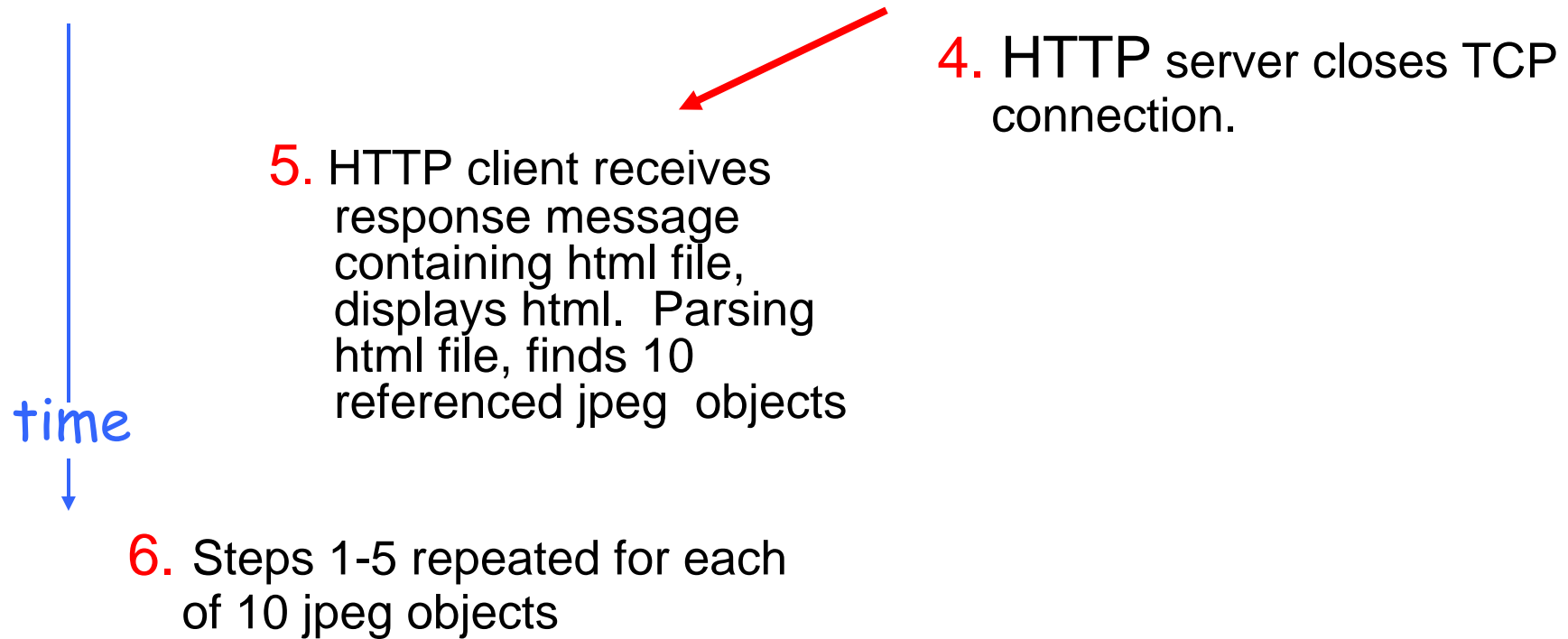
1b. HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `someDepartment/home.index`

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time  
↓

# Nonpersistent HTTP (cont.)



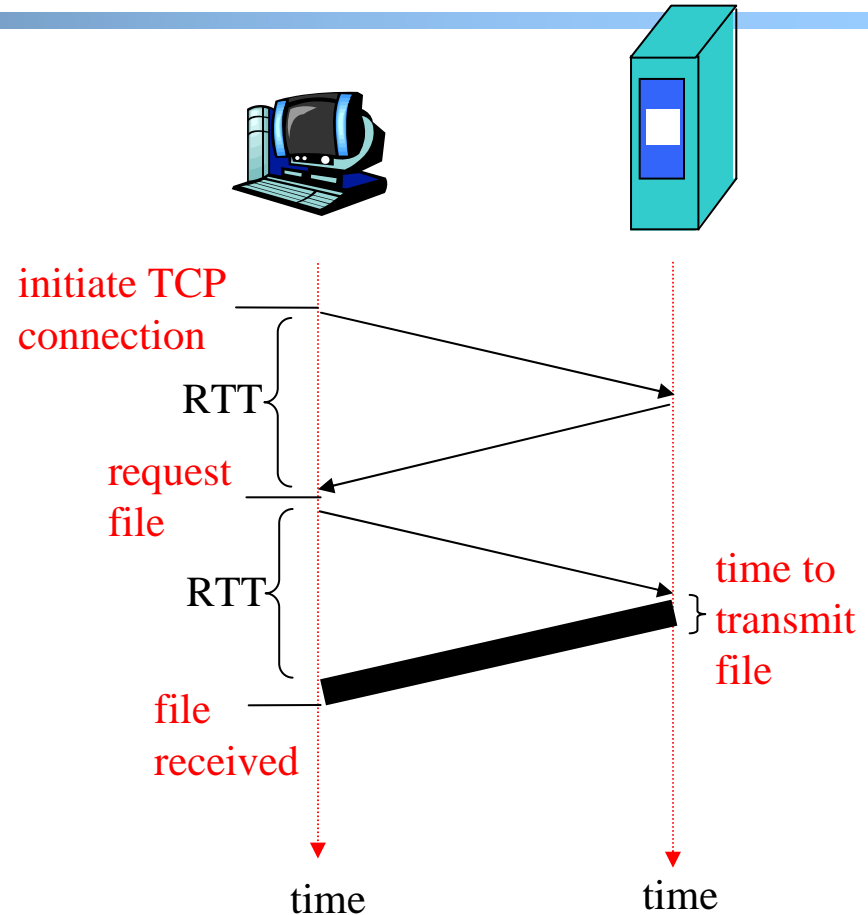
# Response time modeling

**Definition of RTT:** time to send a small packet to travel from client to server and back.

## 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

**total =  $2RTT + \text{transmit time}$**



# Persistent HTTP

## Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but browsers often open parallel TCP connections to fetch referenced objects

## Persistent HTTP

- server leaves connection open after sending responses
- subsequent HTTP messages between same client/server are sent over connection

## Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

## Persistent with pipelining:

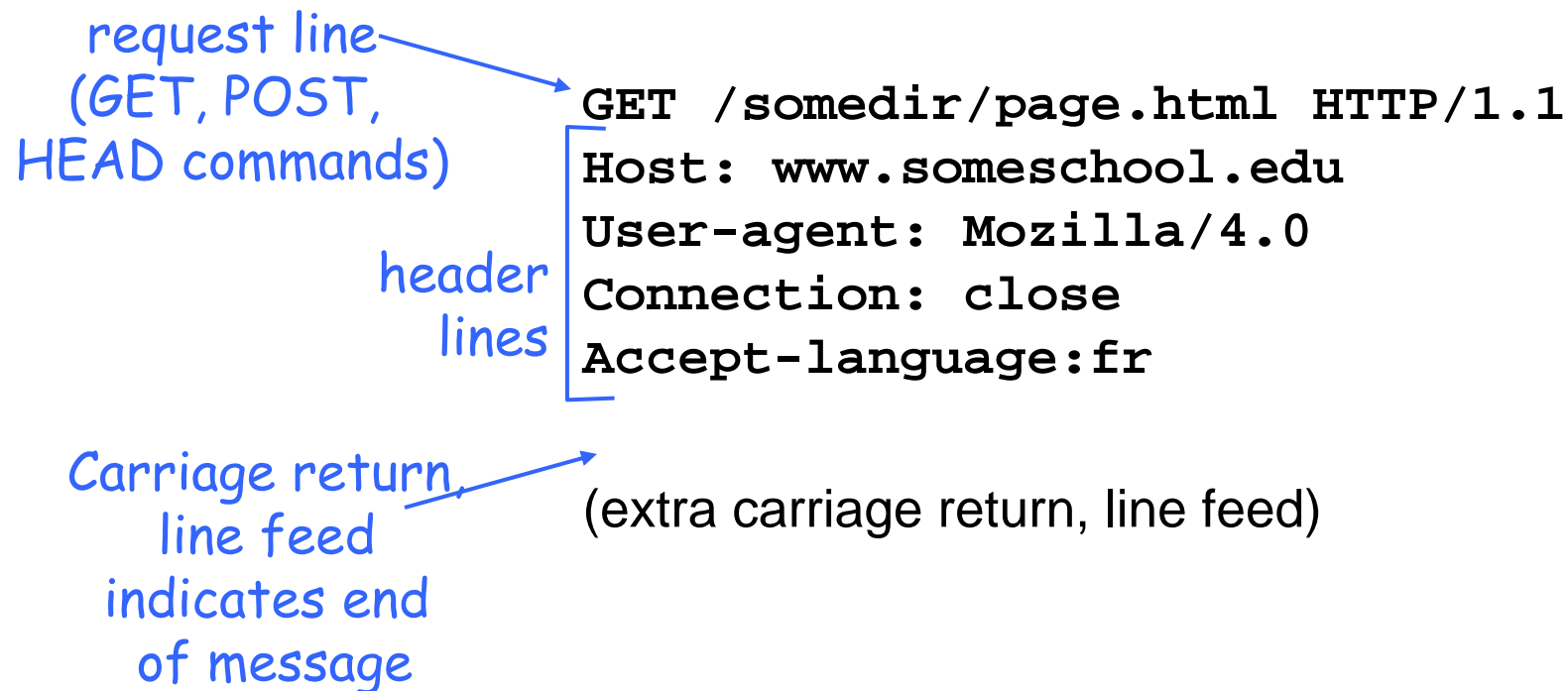
- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

**Several dimensions to help speed up:**

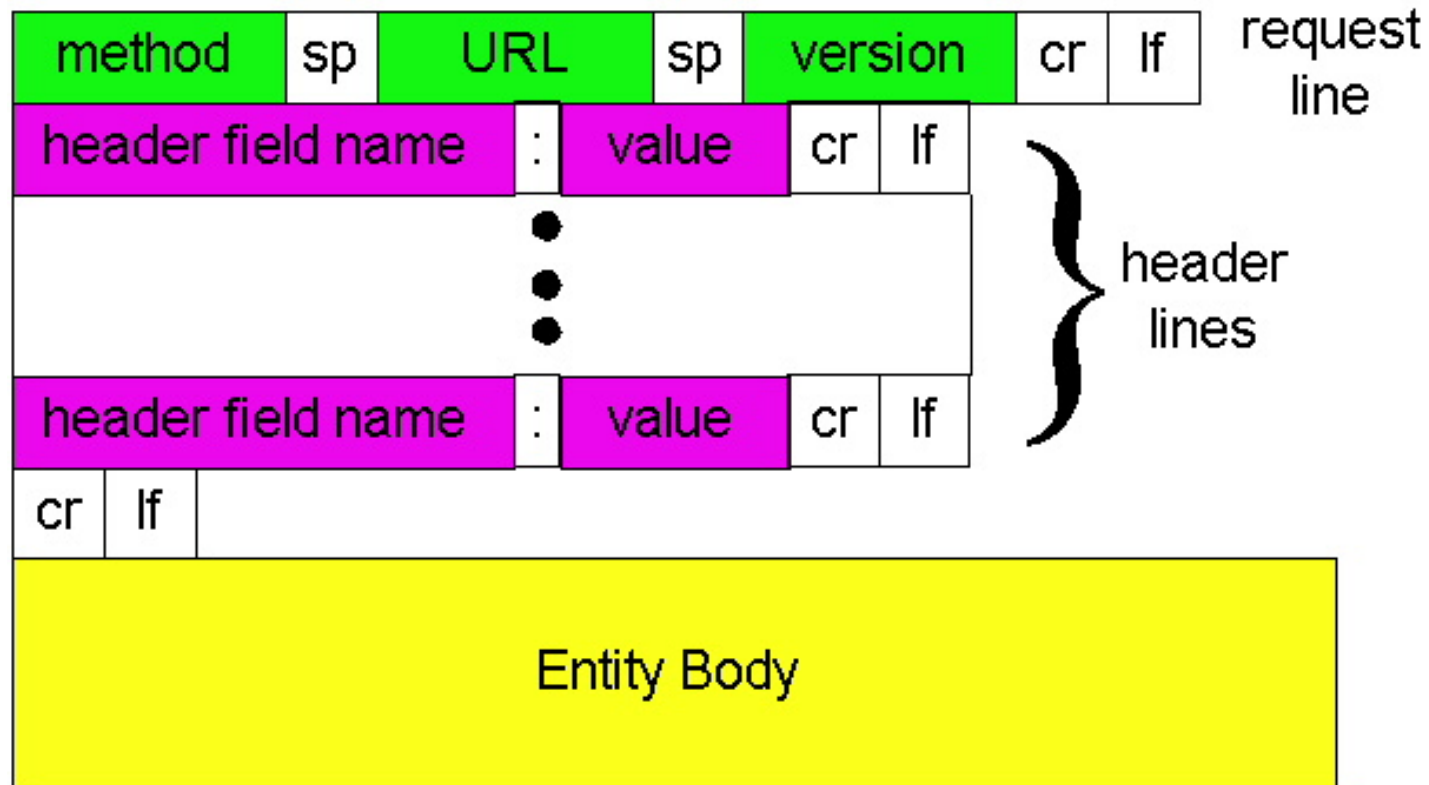
**Persistent connections, pipelining, parallel connections**

# 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

---

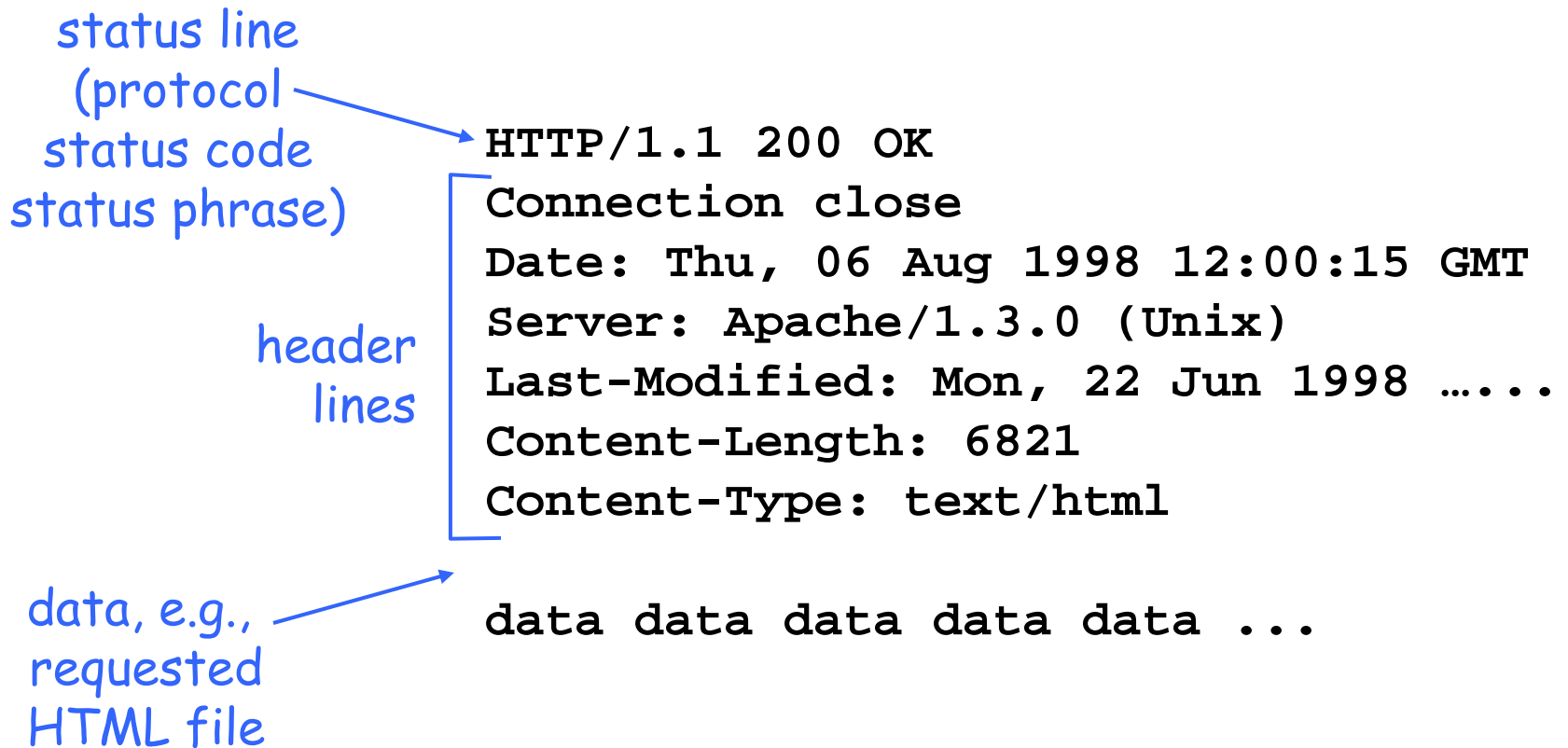
## HTTP/1.0

- 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



# HTTP response status codes

---

In first line in server->client response message.

A few sample codes:

## **200 OK**

- request succeeded, requested object later in this message

## **301 Moved Permanently**

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

## **400 Bad Request**

- request message not understood by server

## **404 Not Found**

- requested document not found on this server

## **505 HTTP Version Not Supported**

# User-server state: cookies

---

Many major Web sites use cookies

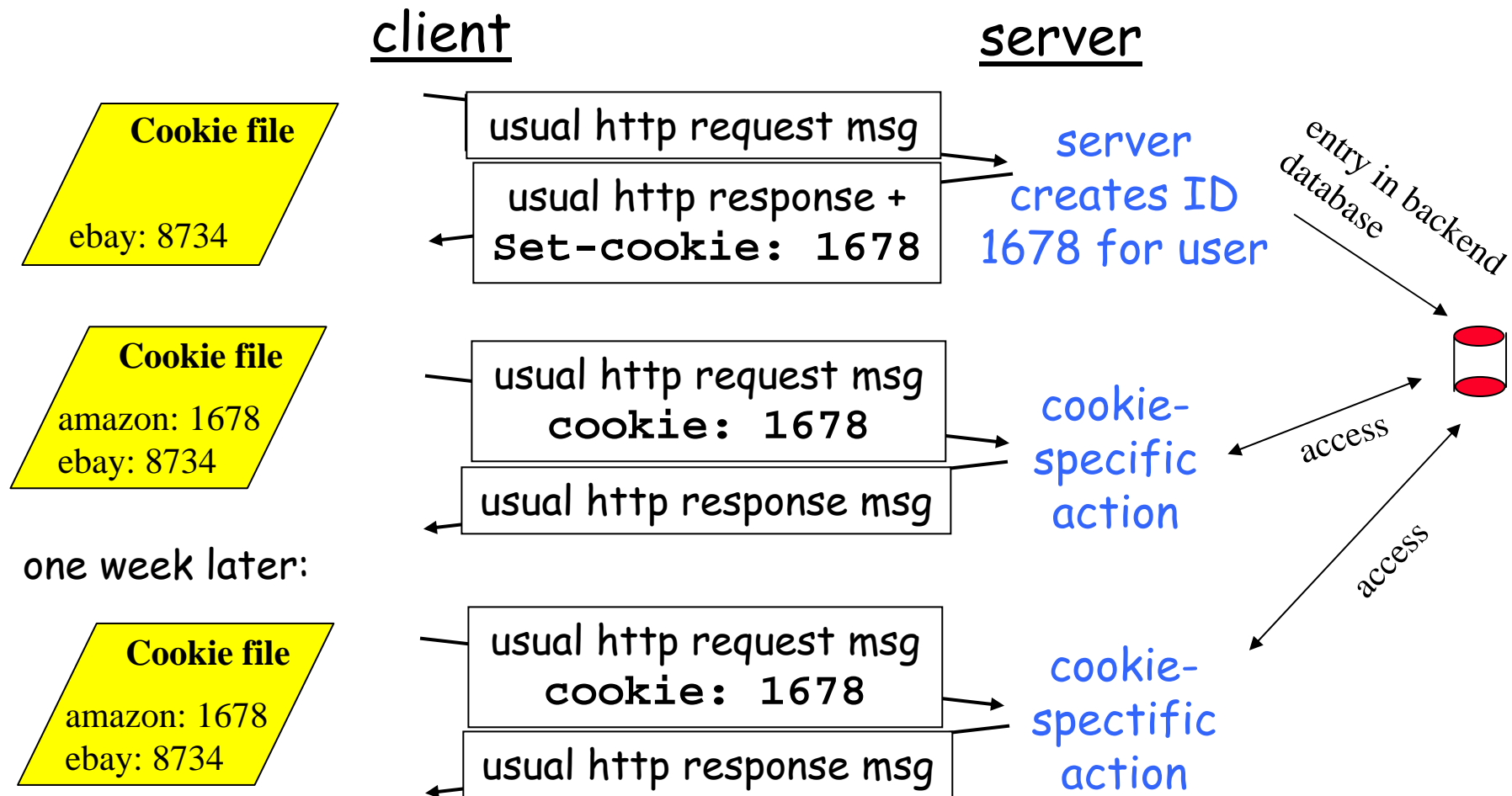
## Four components:

- 1) cookie header line in the HTTP response message
- 2) cookie header line in HTTP request message
- 3) cookie file kept on user's host and managed by user's browser
- 4) back-end database at Web site

## Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

# Cookies: keeping “state” (cont.)



# Cookies (continued)

## What cookies can bring:

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

## Cookies and privacy:

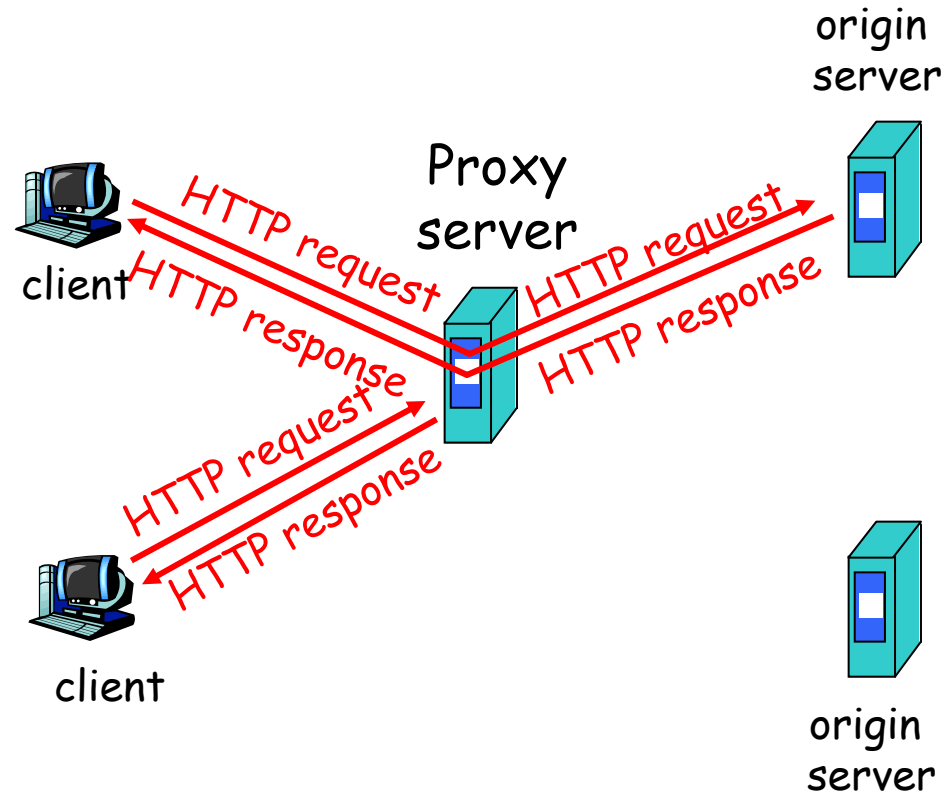
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

Do cookies compromise security?  
Can it be used for authentication?

# Web caches (proxy server)

**Goal:** satisfy client request without involving origin server

- 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



Your Web request may be intercepted using a transparent TCP proxy!



# More about Web caching

---

- Cache acts as both client and 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 (but so does P2P file sharing)

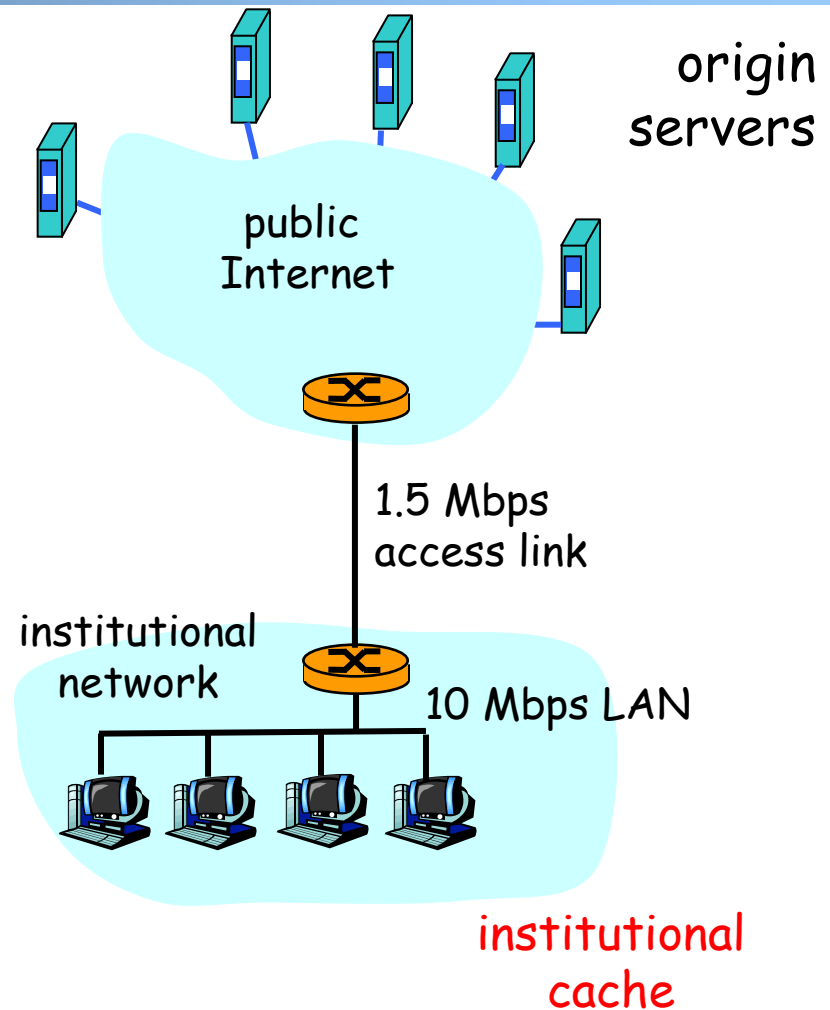
# Caching example

## Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

## Consequences

- utilization on LAN =  
 $1.5\text{Mbps}/10\text{Mbps} = 15\%$
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay  
= 2 sec + minutes + milliseconds



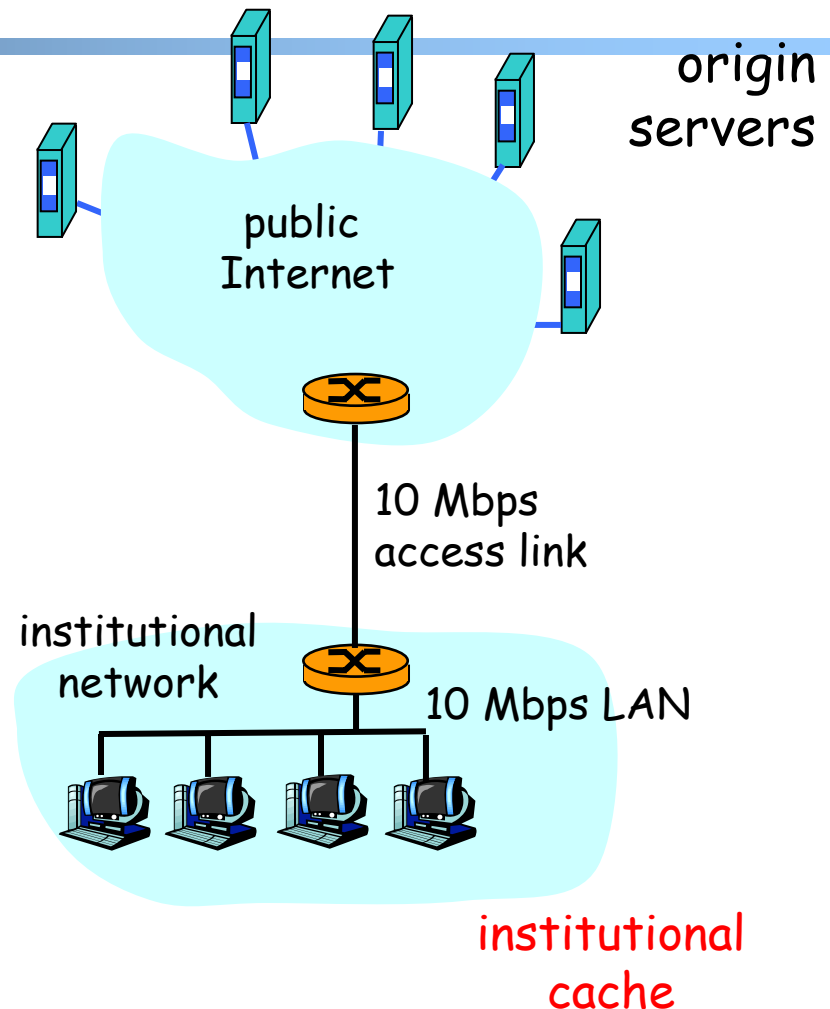
# Caching example (cont)

## Possible solution

- increase bandwidth of access link to, say, 10 Mbps

## Consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay  
= 2 sec + msec + msec
- often a costly upgrade



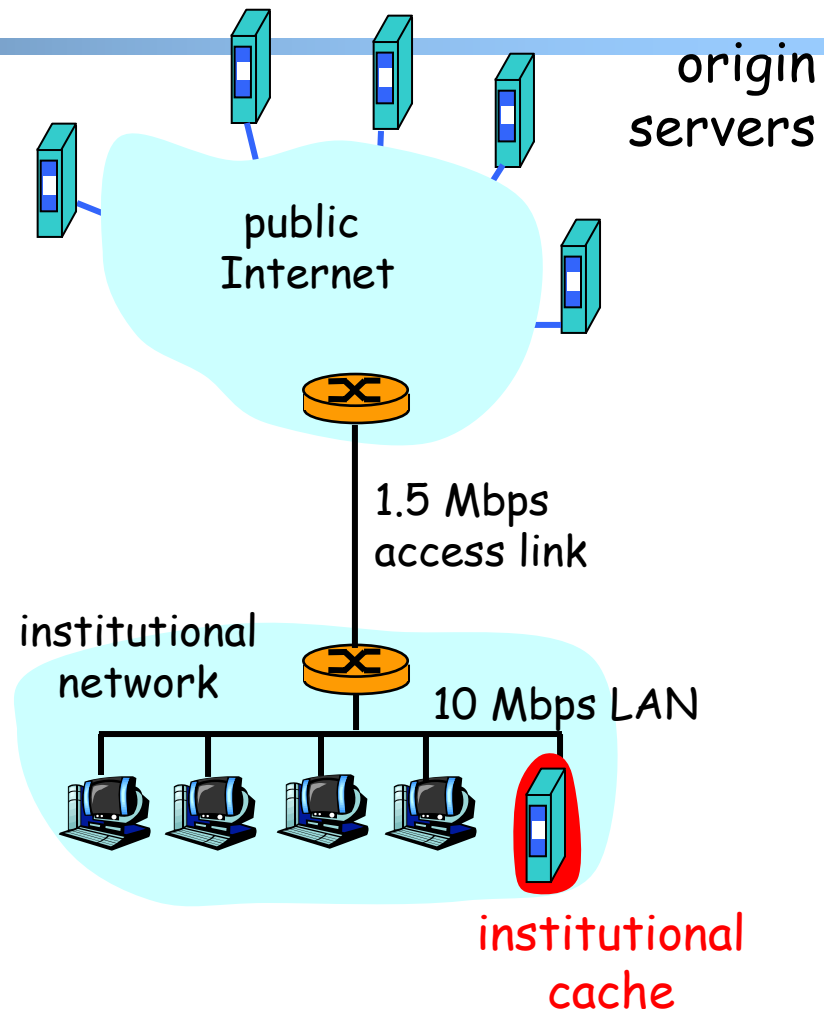
## Caching example (cont)

### Install cache

- suppose hit rate is 0.4

### Consequence

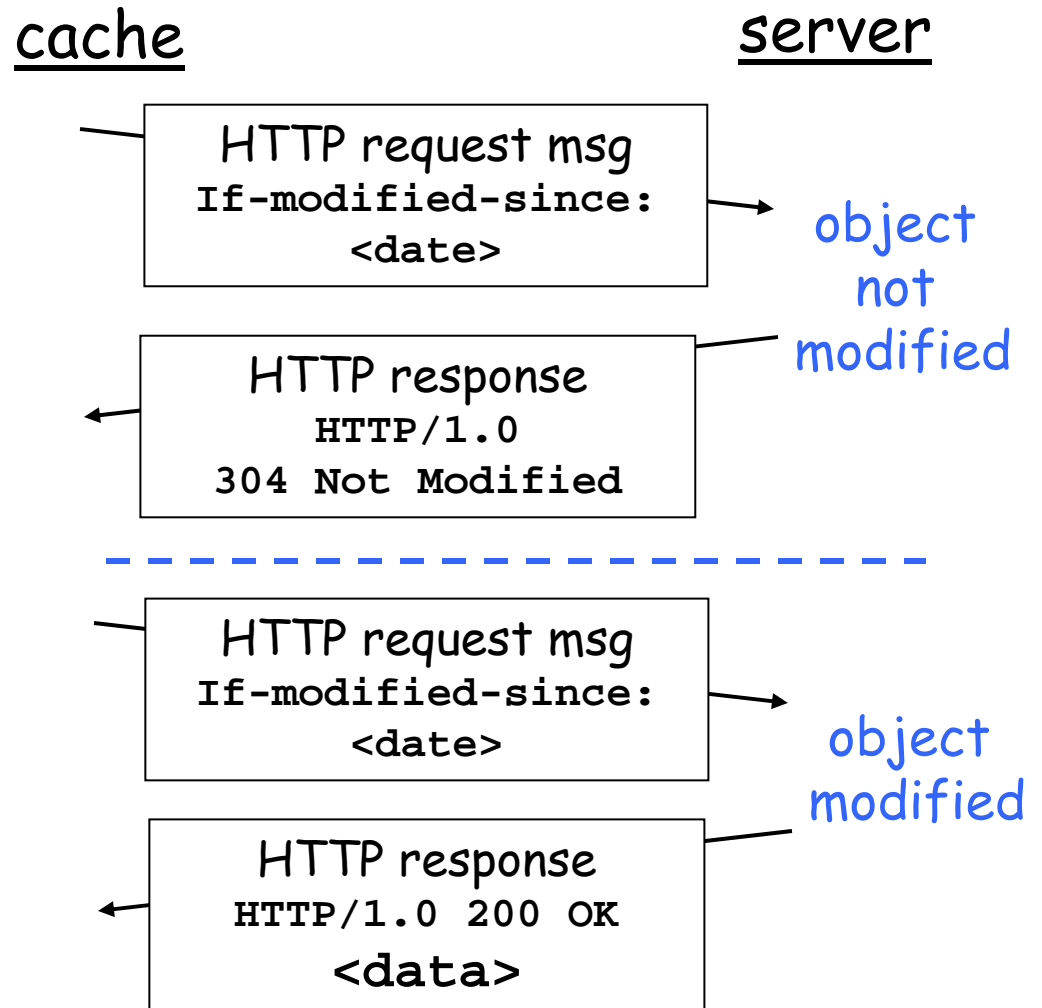
- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay =  $.6 \cdot (2.01) \text{ secs} + \text{milliseconds} < 1.4 \text{ secs}$



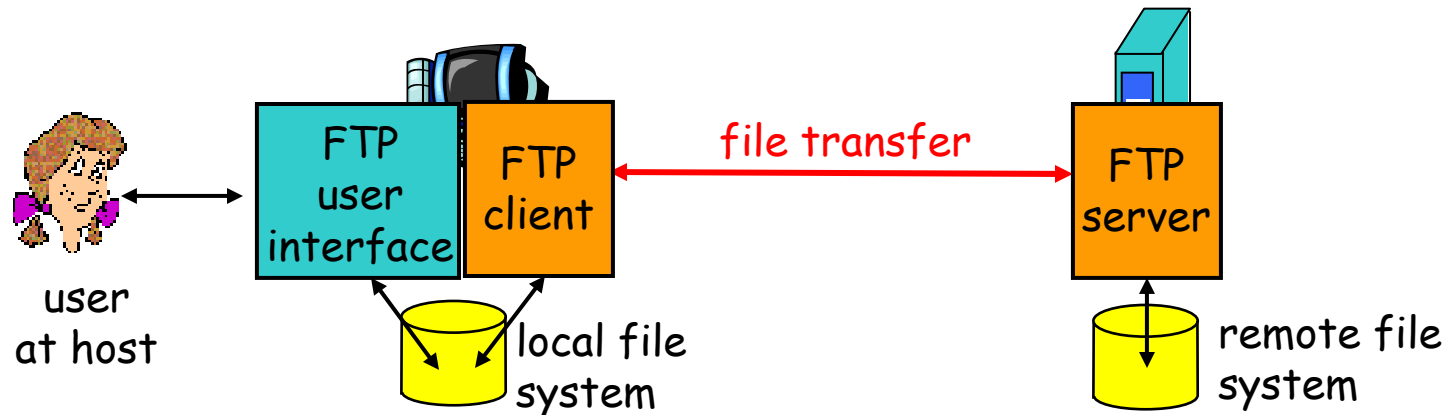
# Conditional GET

- **Goal:** don't send object if cache has up-to-date cached version
- 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`



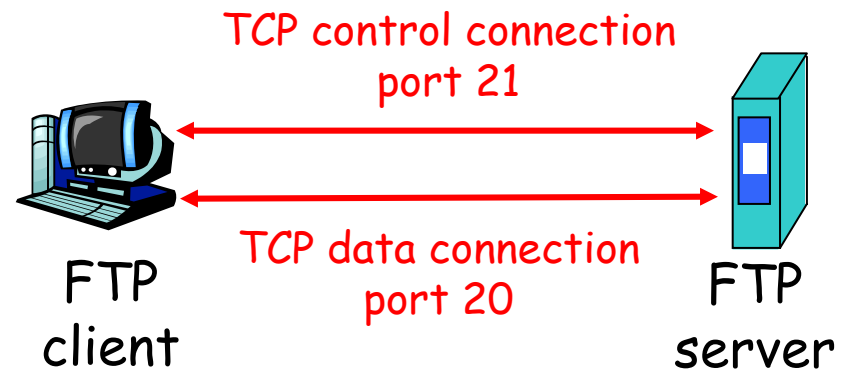
# FTP: the file transfer protocol



- 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, specifying TCP as transport protocol
- Client obtains authorization over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives a command for a file transfer, the server opens a TCP data connection to client
- After transferring one file, server closes connection.



- Server opens a second TCP data connection to transfer another file.
- Control connection: “out of band”
- FTP server maintains “state”: current directory, earlier authentication

What’s the advantage of an out-of-band control channel?

# 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**



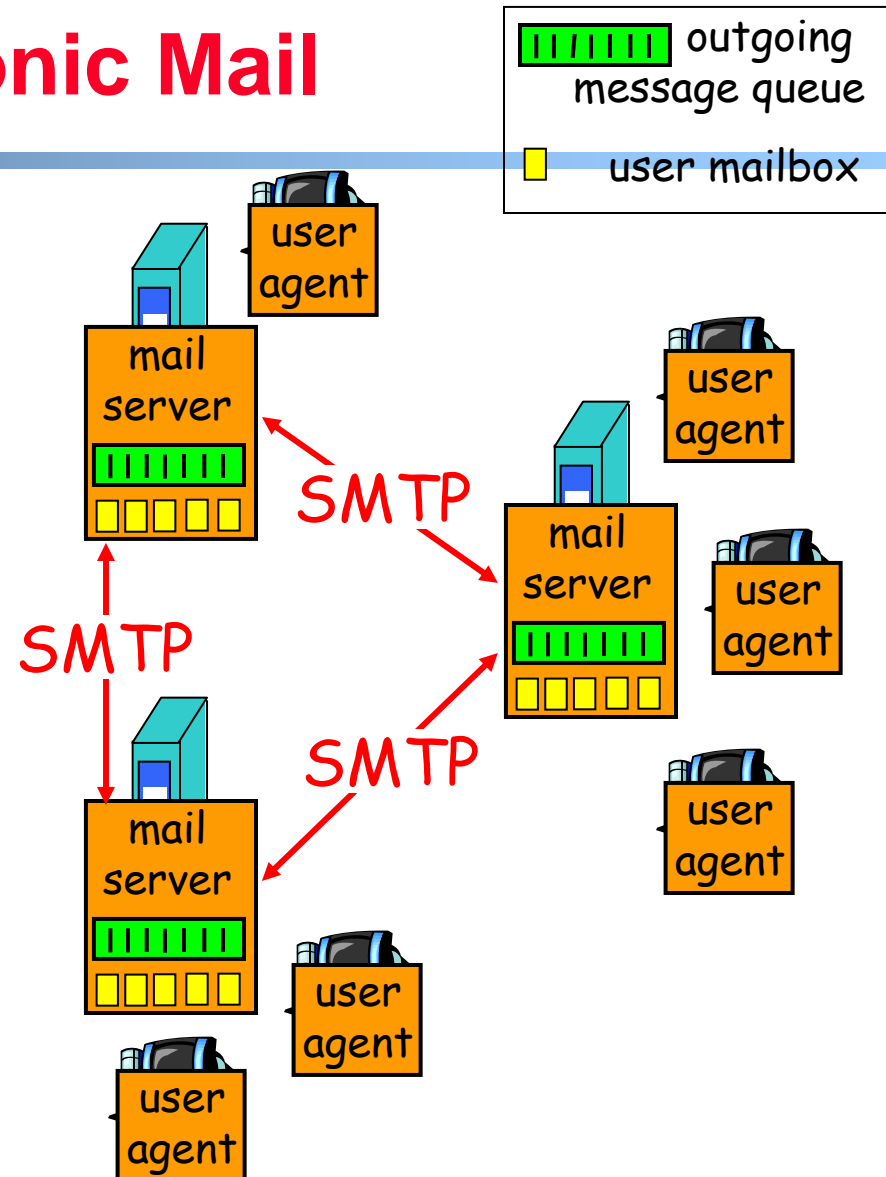
# Electronic Mail

## Three major components:

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

## User Agent

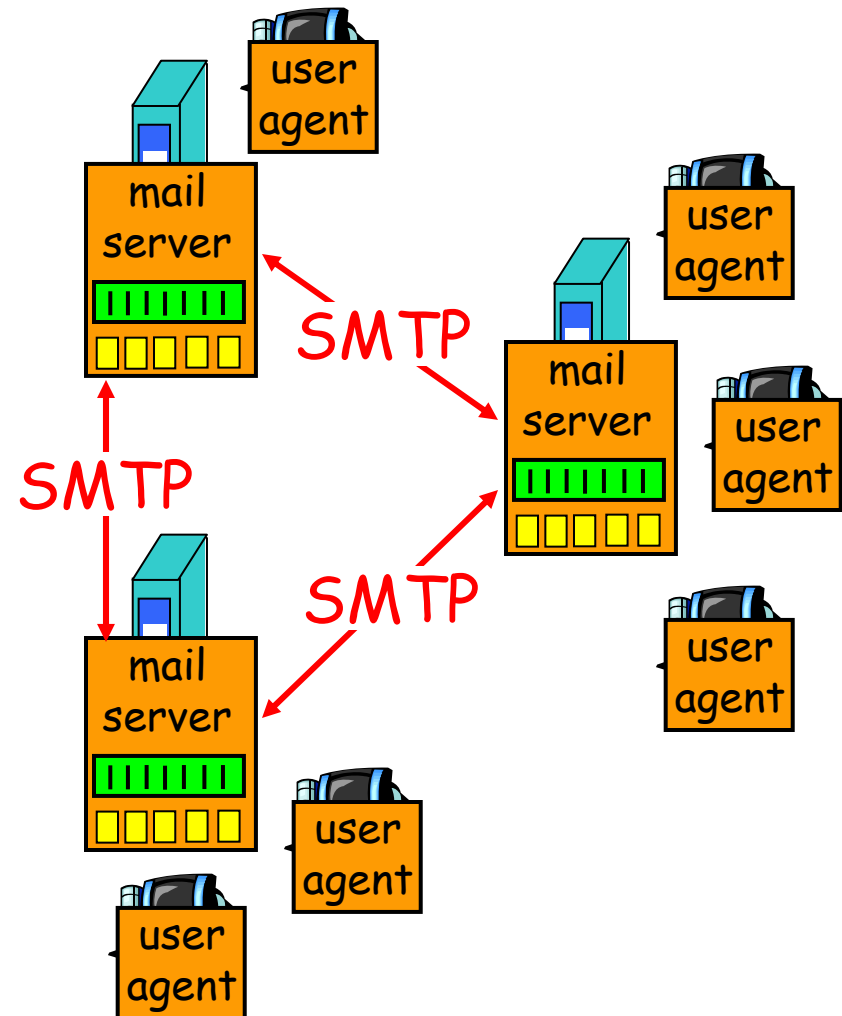
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server



# 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



Where can we find out the mail servers for a domain?

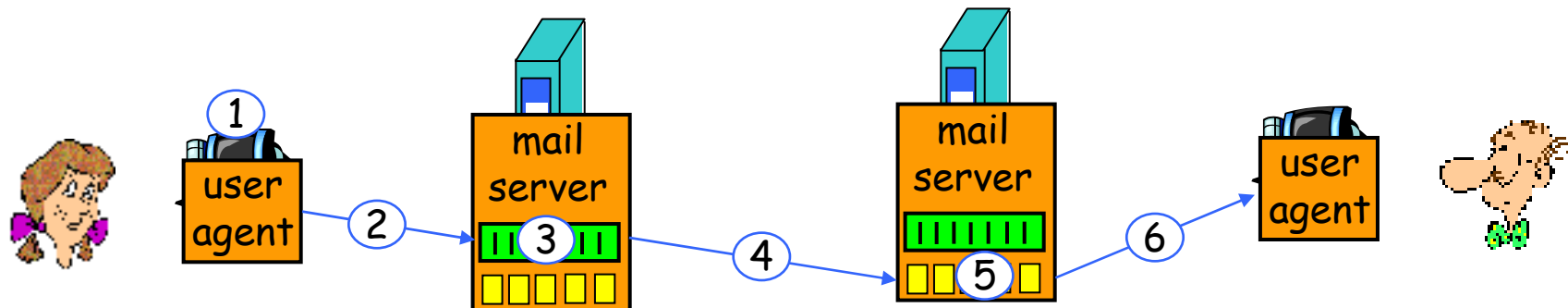
# Electronic Mail: SMTP [RFC 2821]

---

- 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
  - **commands**: ASCII text
  - **response**: status code and phrase
- messages must be in 7-bit ASCII

# Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message and “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:

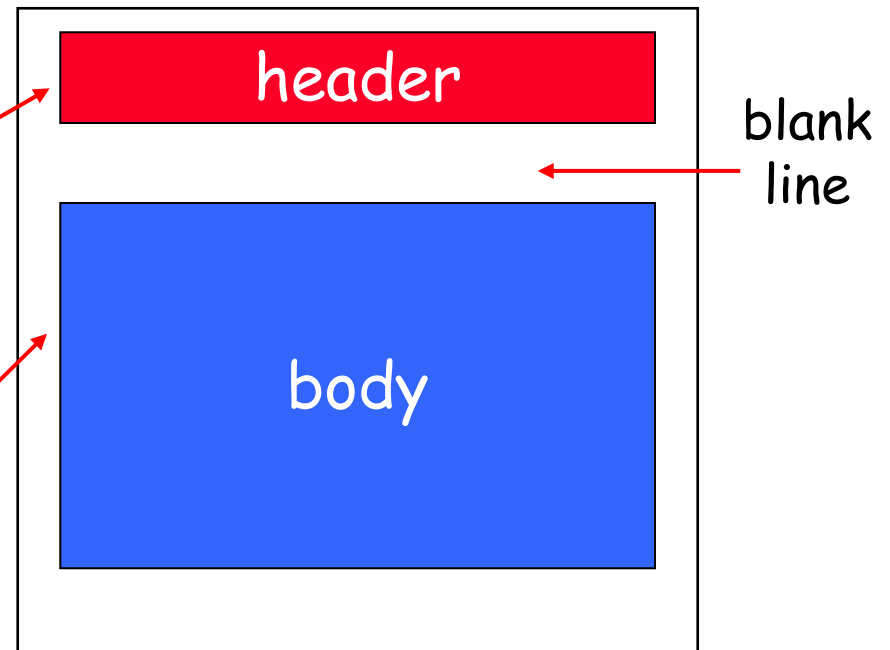
- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

# Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

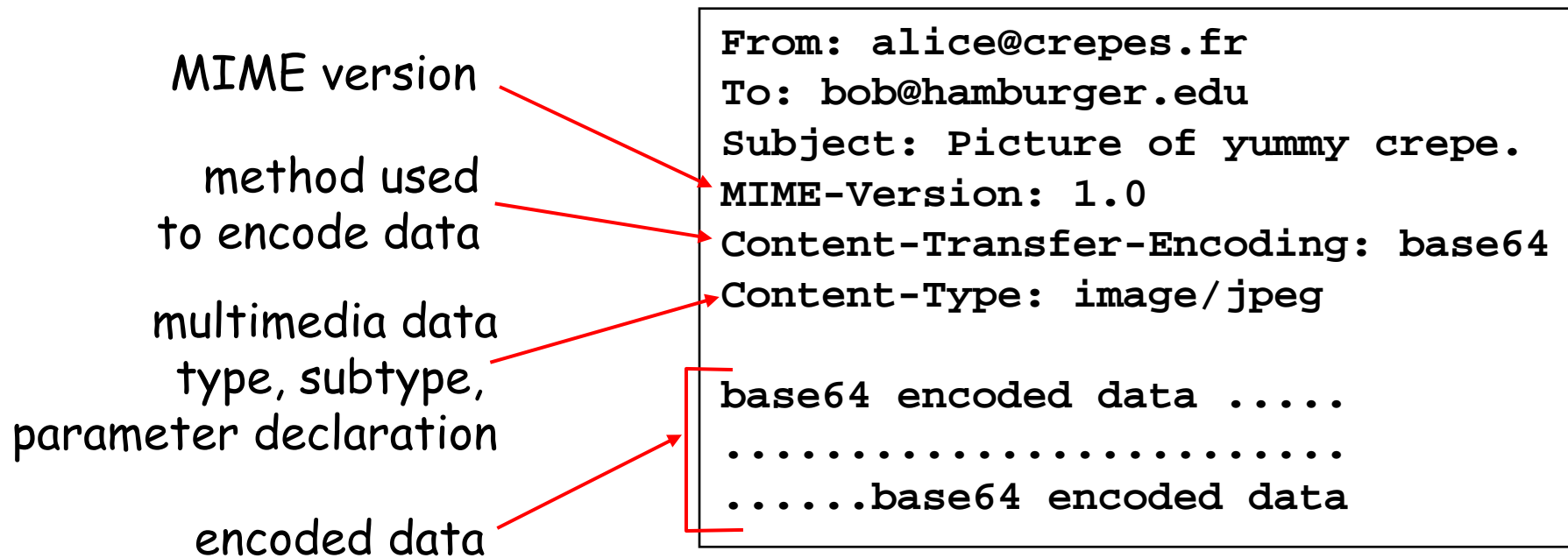
- header lines, e.g.,
  - To:
  - From:
  - Subject:*different from SMTP commands!*
- body
  - the “message”, ASCII characters only



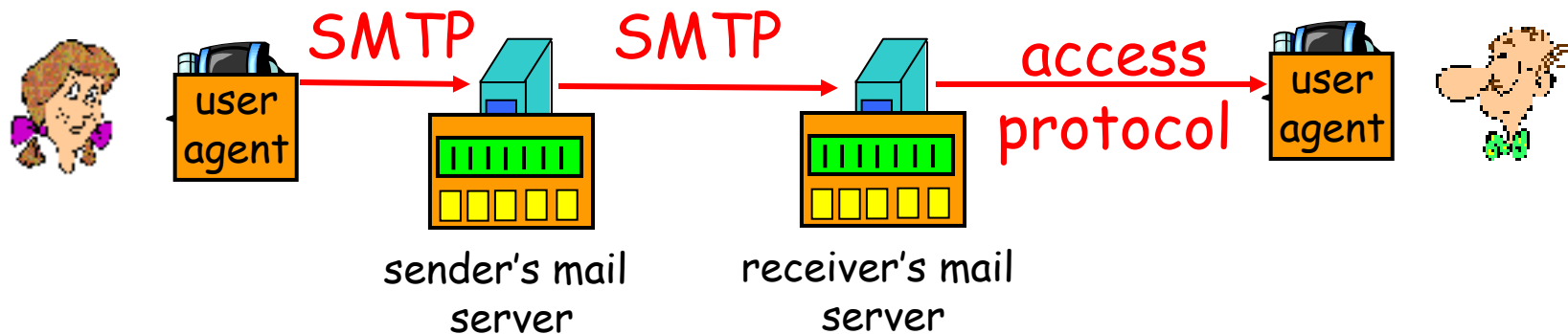


# Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type



# Mail access protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: 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 59
```

# POP3 (more) and IMAP

---

## More about POP3

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

## IMAP

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

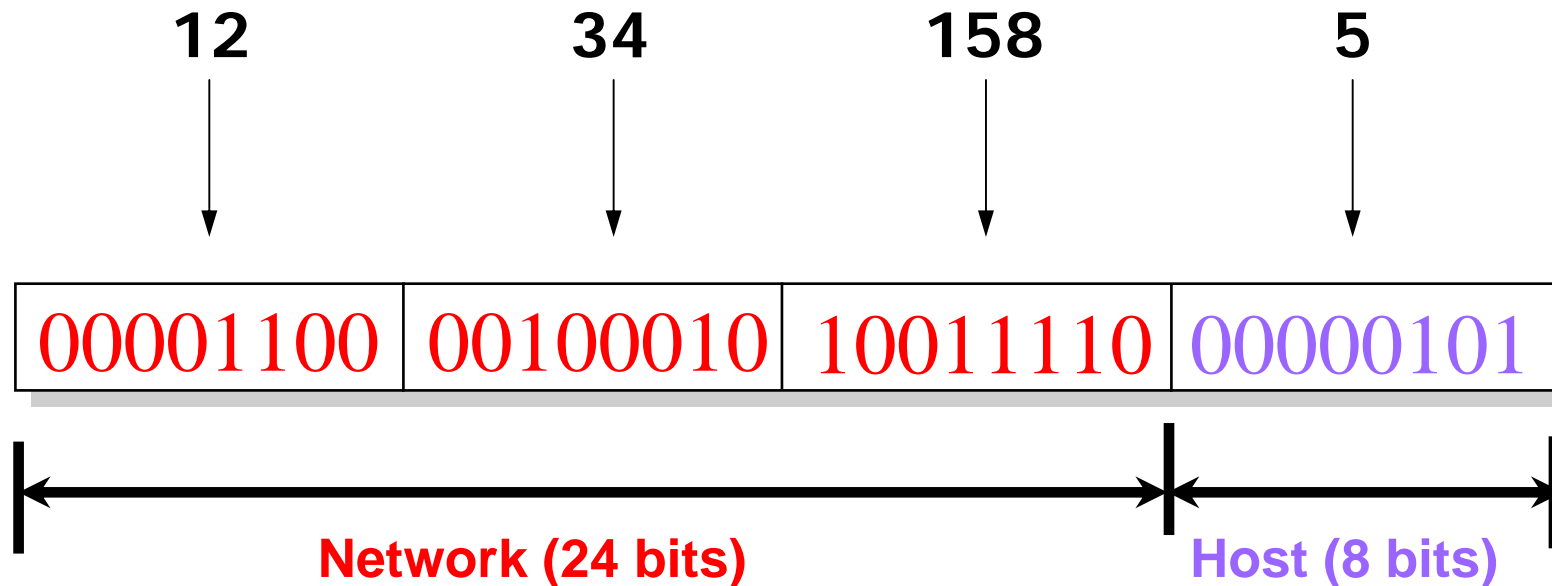
# Discussions

---

- Why do we have so much spam?
  - How would you design the email system to prevent spam?
- How does anonymous email work?

# IP Addressing

- 32-bit number in dotted-quad notation (12.34.158.5)
- Divided into network & host portions (left and right)
- 12.34.158.0/24 is a 24-bit prefix with  $2^8$  addresses



# Some History: Why Dotted-Quad Notation?

---

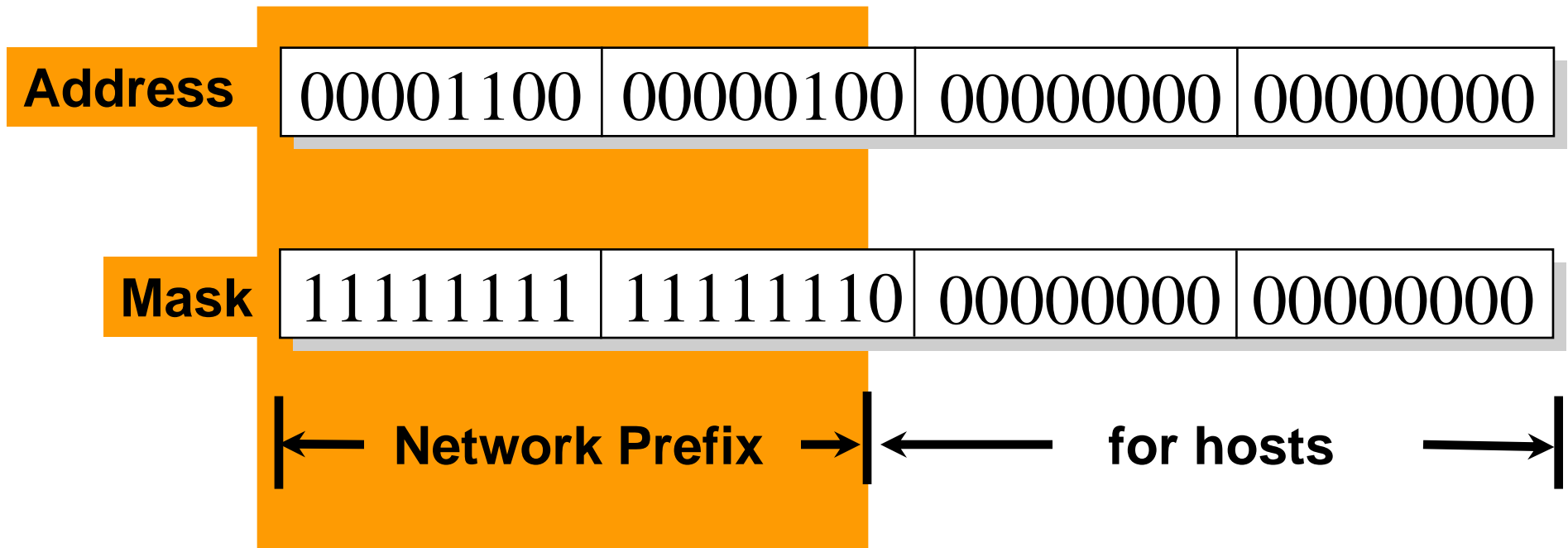
- In the olden days...
  - Class A: 0\*
    - Very large /8 blocks (e.g., MIT has 18.0.0.0/8)
  - Class B: 10\*
    - Large /16 blocks (e.g., UM has 141.213.0.0/16)
  - Class C: 110\*
    - Small /24 blocks (e.g., AT&T Labs has 192.20.225.0/24)
  - Class D: 1110\*
    - Multicast groups
  - Class E: 11110\*
    - Reserved for future use (sounds a bit scary...)
- And then, address space became scarce...

# Classless Inter-Domain Routing (CIDR)

Use two 32-bit numbers to represent a network.  
Network number = IP address + Mask

IP Address : 12.4.0.0

IP Mask: 255.254.0.0

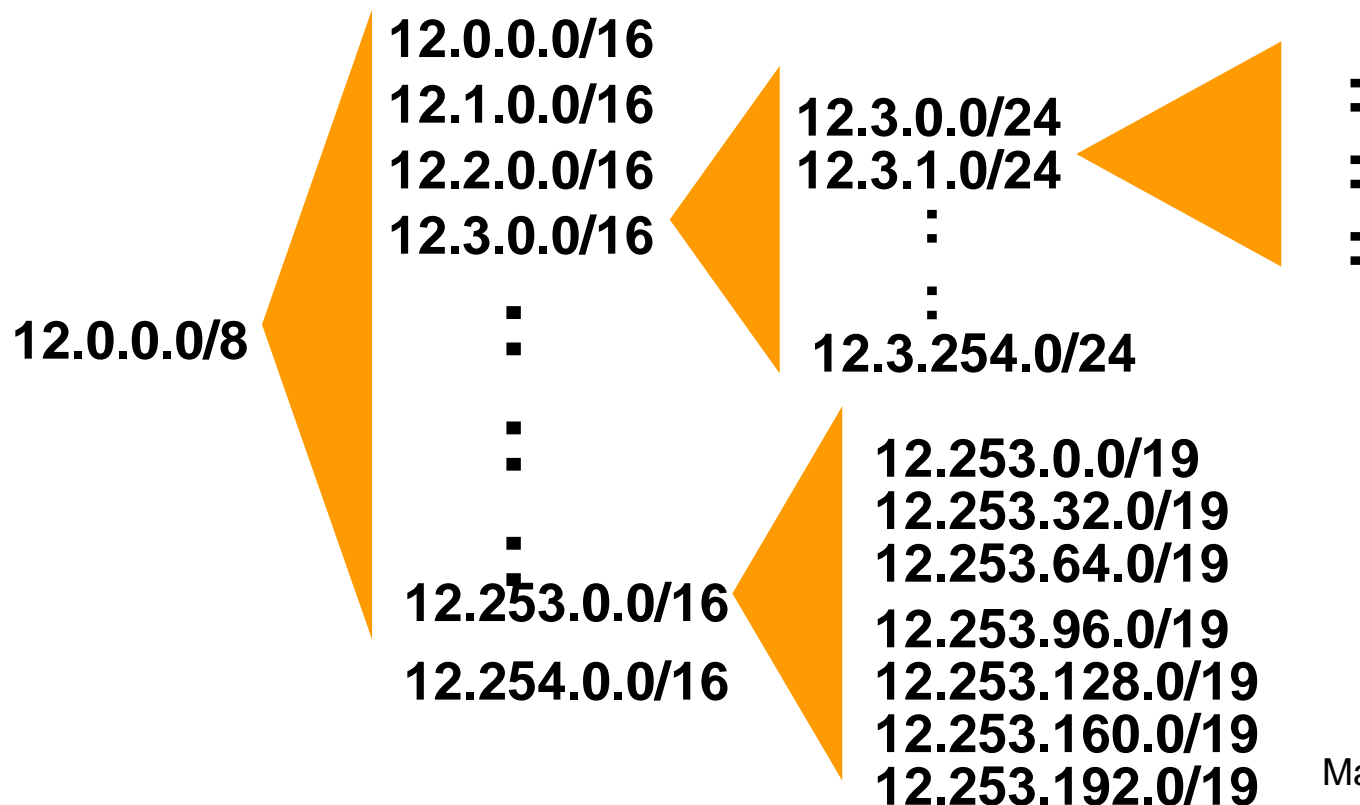


Usually written as 12.4.0.0/15



# CIDR = Hierarchy in Address Allocation

- Prefixes are key to Internet scalability
  - Address allocation by ARIN/RIPE/APNIC and by ISPs
  - Routing protocols and packet forwarding based on prefixes
  - Today, routing tables contain ~150,000-200,000 prefixes



# Figuring Out Who Owns an Address

---

- Address registries
  - Public record of address allocations
  - ISPs should update when giving addresses to customers
  - However, records are notoriously out-of-date
- Ways to query
  - UNIX: “whois -h whois.arin.net 128.112.136.35”
  - <http://www.arin.net/whois/>
  - <http://www.geektools.com/whois.php>
  - ...

# Example Output for 141.213.4.5 (galileo.eecs.umich.edu)

---

OrgName: University of Michigan  
OrgID: UNIVER-118  
Address: IT Communications Services  
Address: 4251 Plymouth Road  
City: Ann Arbor  
StateProv: MI  
PostalCode: 48105-2785  
Country: US

NetRange: 141.213.0.0 - 141.213.255.255  
CIDR: 141.213.0.0/16  
NetName: UMN3  
NetHandle: NET-141-213-0-0-1  
Parent: NET-141-0-0-0-0  
NetType: Direct Assignment  
NameServer: SRVR8.ENGIN.UMICH.EDU  
NameServer: SRVR7.ENGIN.UMICH.EDU  
NameServer: DNS2.ITD.UMICH.EDU

## There is more...

---

Comment: Abuse contact for 141.213.128.0/17 is [abuse@umich.edu](mailto:abuse@umich.edu).

Comment: For DMCA info see <http://www.umich.edu/~itua/copyright/>

RegDate: 1990-08-02

Updated: 2003-03-27

AbuseHandle: CEAC-ARIN

AbuseName: College of Engineering Abuse Contact

AbusePhone: +1-734-936-2486

AbuseEmail: [abuse@engin.umich.edu](mailto:abuse@engin.umich.edu)

TechHandle: PMK5-ARIN

TechName: Killey, Paul M.

TechPhone: +1-734-763-4910

TechEmail: [paul@engin.umich.edu](mailto:paul@engin.umich.edu)

OrgTechHandle: UA11-ORG-ARIN

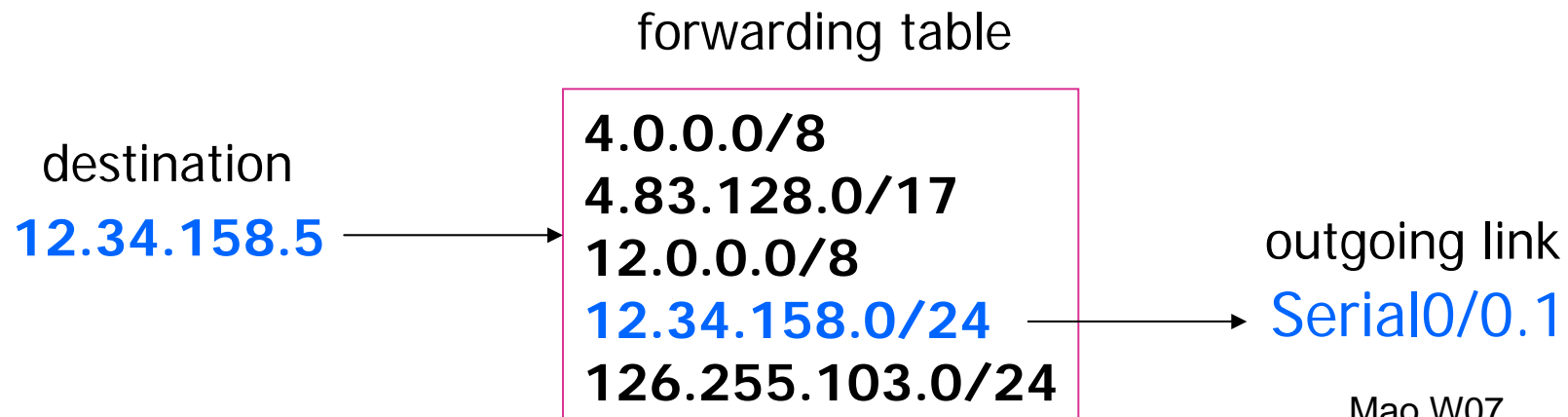
OrgTechName: UMnet Administration

OrgTechPhone: +1-734-647-4200

OrgTechEmail: [umnet-admin@umich.edu](mailto:umnet-admin@umich.edu)

# Longest Prefix Match Forwarding

- Forwarding tables in IP routers
  - Maps each IP prefix to next-hop link(s)
- Destination-based forwarding
  - Packet has a destination address
  - Router identifies longest-matching prefix
  - Cute algorithmic problem: very fast lookups



# How are packets forwarded?

---

- Routers have forwarding tables
  - Map prefix to outgoing link(s)
- Entries can be statically configured
  - E.g., “map 12.34.158.0/24 to Serial0/0.1”
- But, this doesn't adapt
  - To failures
  - To new equipment
  - To the need to balance load
  - ...
- That is where routing protocols come in... [more on this in the next lectures]

# Discussions

---

- IP address space scarcity
  - What can we do about it?
- Increased IP address fragmentation
- Does an IP address identify the actual user?
- How does one achieve mobility while maintaining the same IP address?

# DNS: Domain Name System

---

**People:** many identifiers: **Domain Name System:**

- SSN, name, passport #

**Internet hosts, routers:**

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

▪ *distributed database* implemented in hierarchy of many *name servers*

*application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)

- note: core Internet function, implemented as application-layer protocol
- complexity at network’s “edge”



# DNS

---

## DNS services

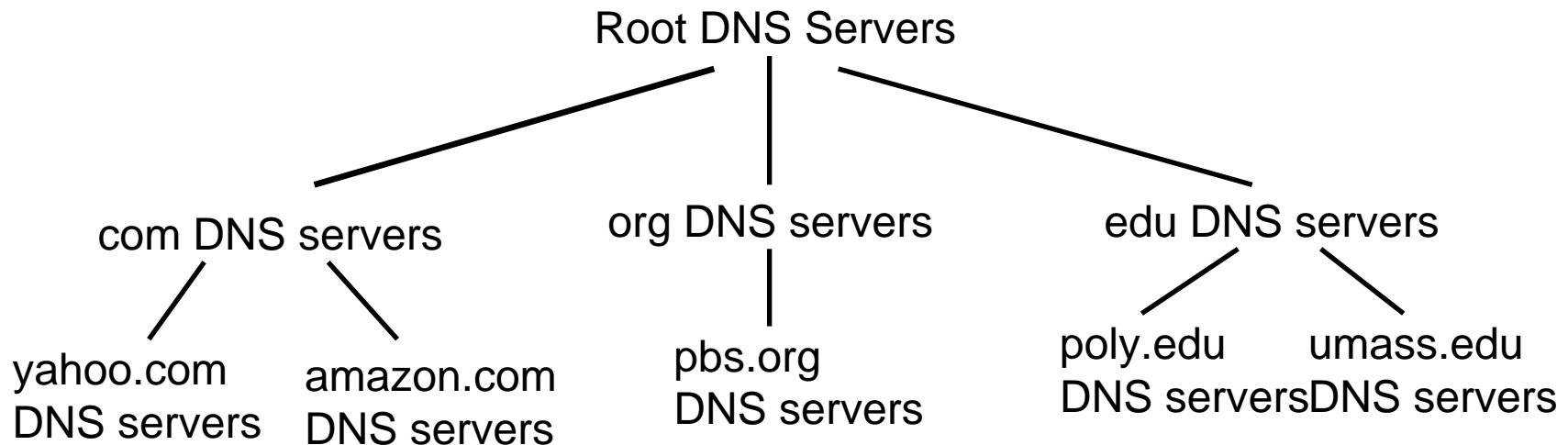
- Hostname to IP address translation
- Host aliasing
  - Canonical and alias names
- Mail server aliasing
- Load distribution
  - Replicated Web servers: set of IP addresses for one canonical name

## Why not centralize DNS?

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

doesn't *scale!*

# Distributed, Hierarchical Database



Client wants IP for [www.amazon.com](http://www.amazon.com); 1<sup>st</sup> approx:

- Client queries a 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](http://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 and Authoritative Servers

---

- **Top-level domain (TLD) servers:** responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network solutions maintains servers for com TLD
- **Authoritative DNS servers:** organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and mail).
  - Can be maintained by organization or service provider

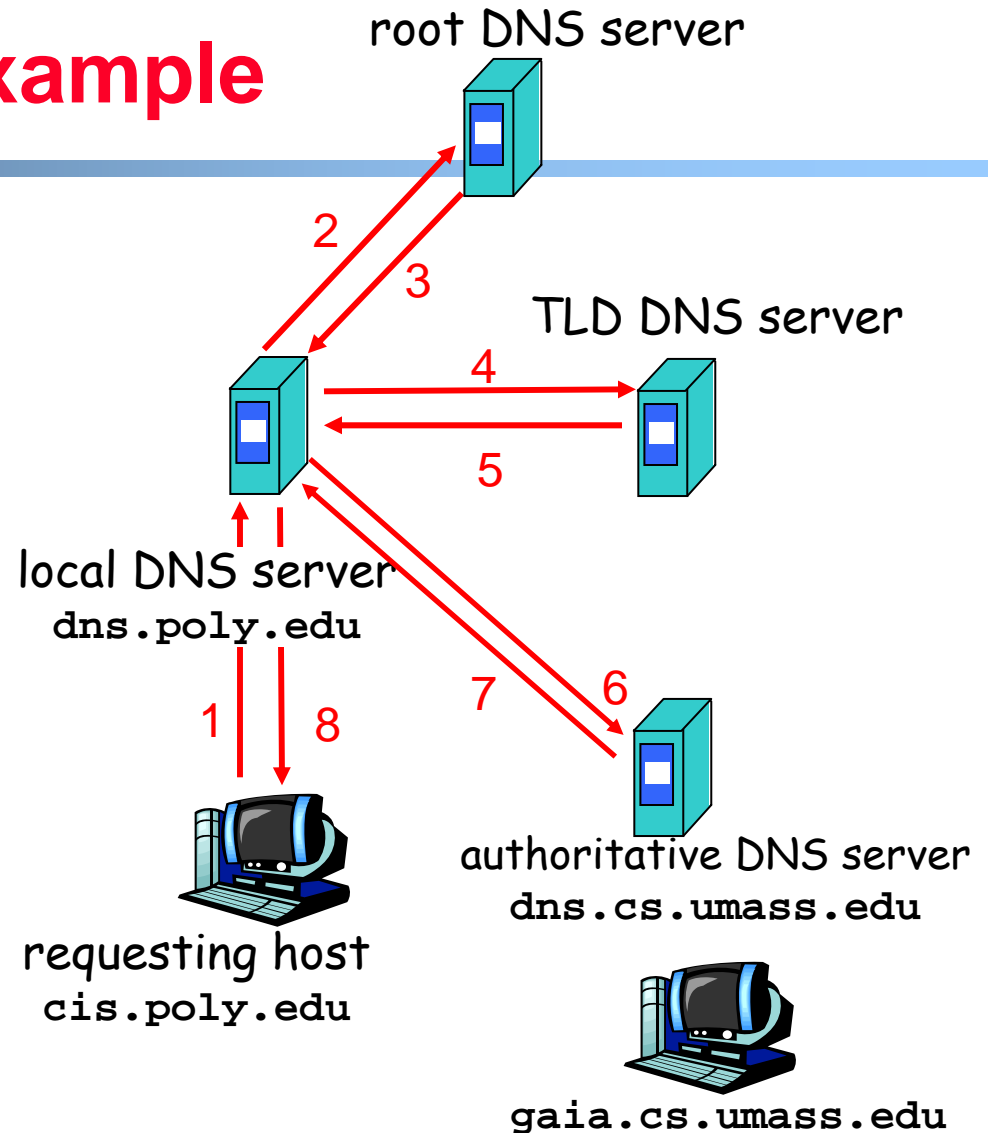
# Local Name Server

---

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - Also called “default name server”
- When a host makes a DNS query, query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy.

# Example

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



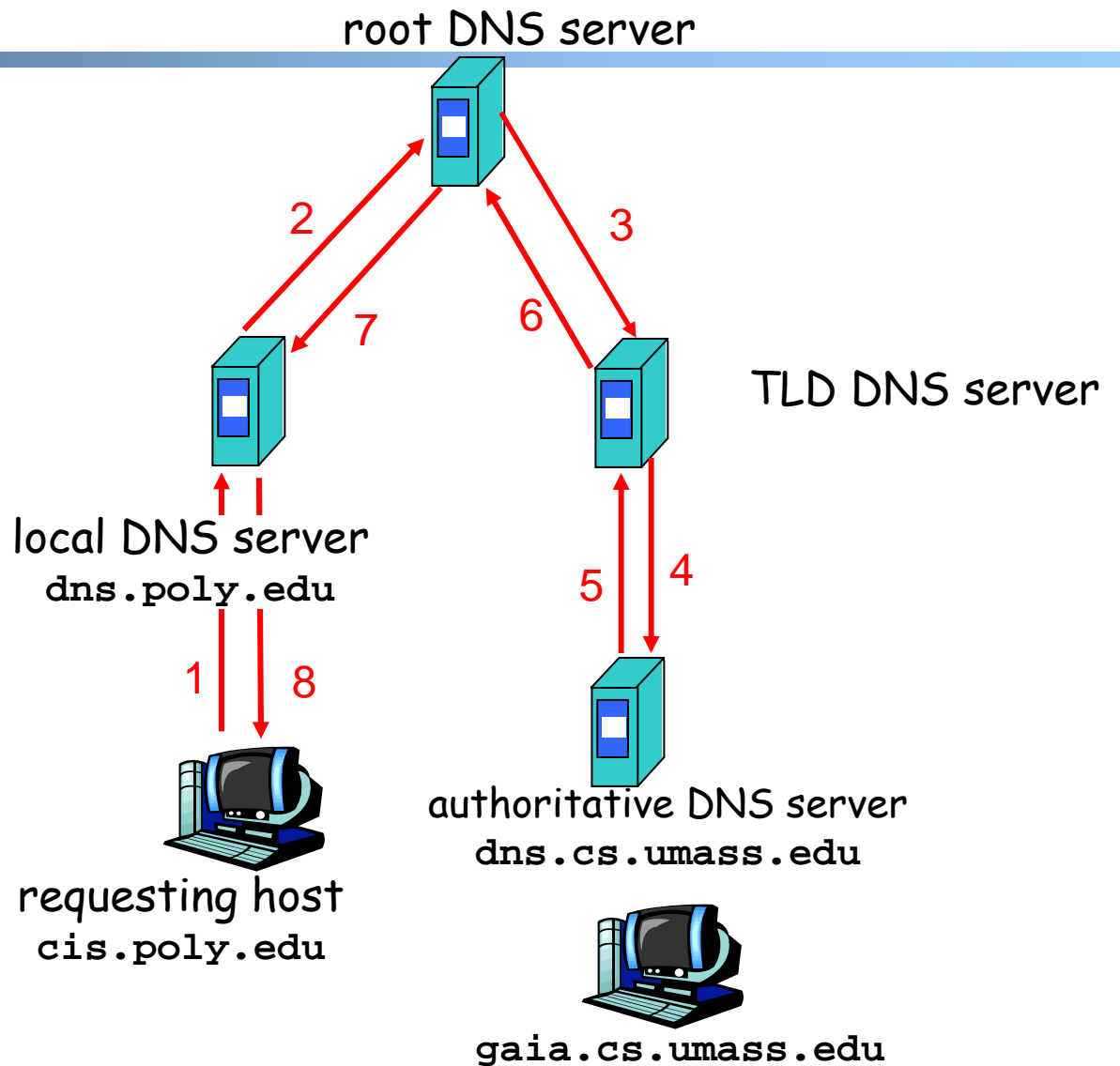
# Recursive queries

## recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

## iterated query:

- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



# DNS: caching and updating records

---

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- update/notify mechanisms under design by IETF
  - RFC 2136
  - <http://www.ietf.org/html.charters/dnsind-charter.html>



# 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 IP address 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

DNS protocol : *query* and *reply* messages, both with same *message format*

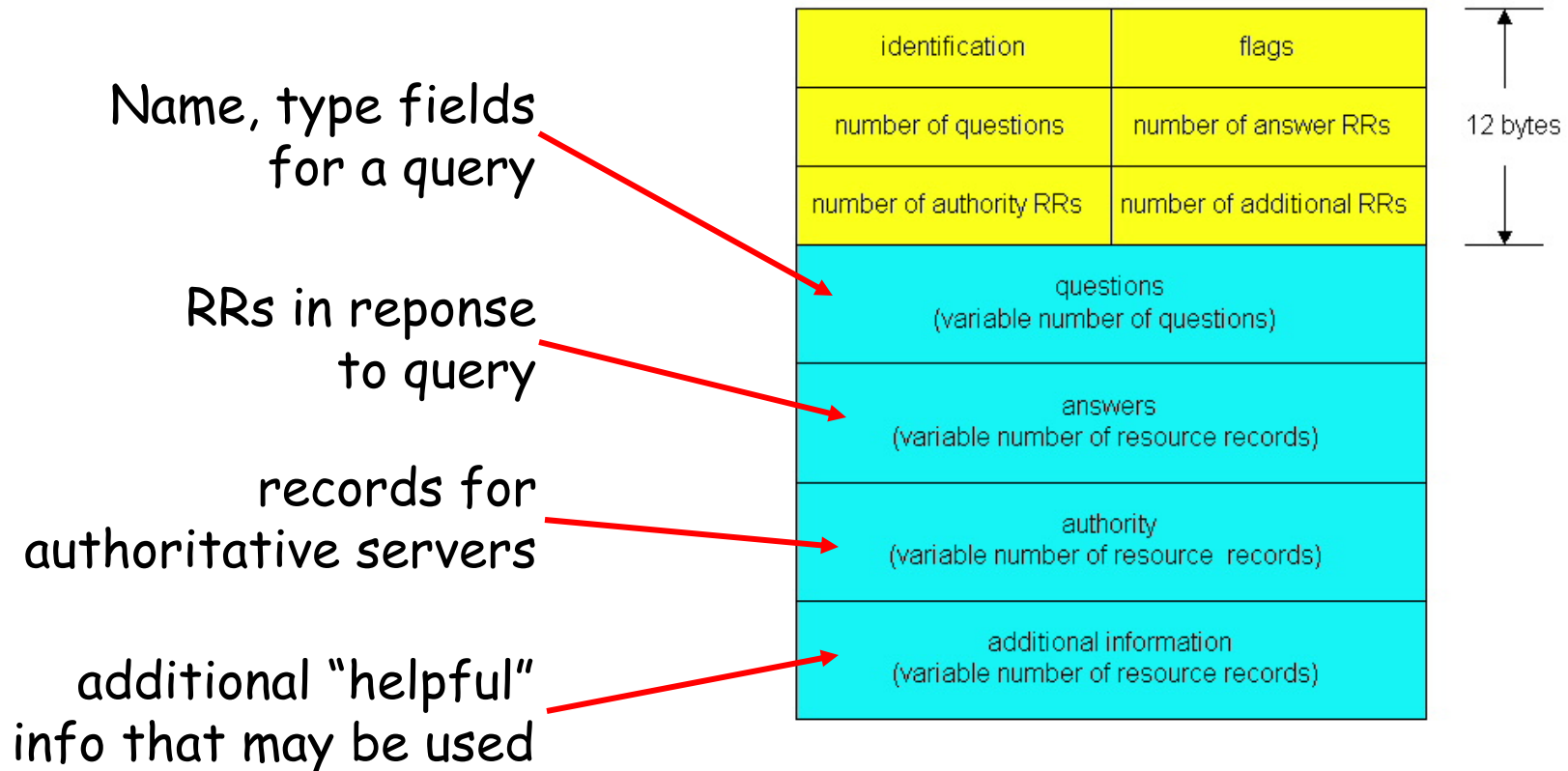
## msg header

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

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	



# DNS protocol, messages



# Inserting records into DNS

---

- Example: just created startup “Network Utopia”
- Register name networkutopia.com at a registrar (e.g., Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
```

```
(dns1.networkutopia.com, 212.212.212.1, A)
```

- Put in authoritative server Type A record for www.networkutopia.com and Type MX record for networkutopia.com
- How do people get the IP address of your Web site?

# Discussions

---

- Is it easy to attack the DNS system?
- Why is DNS caching good?
- Why is DNS caching bad?
- DNS is “exploited” for server load balancing, how?
  - Local DNS servers are usually close to local clients
- If you were to design DNS differently today, how would you?
  - Any problems with the current DNS system?

# P2P: centralized directory

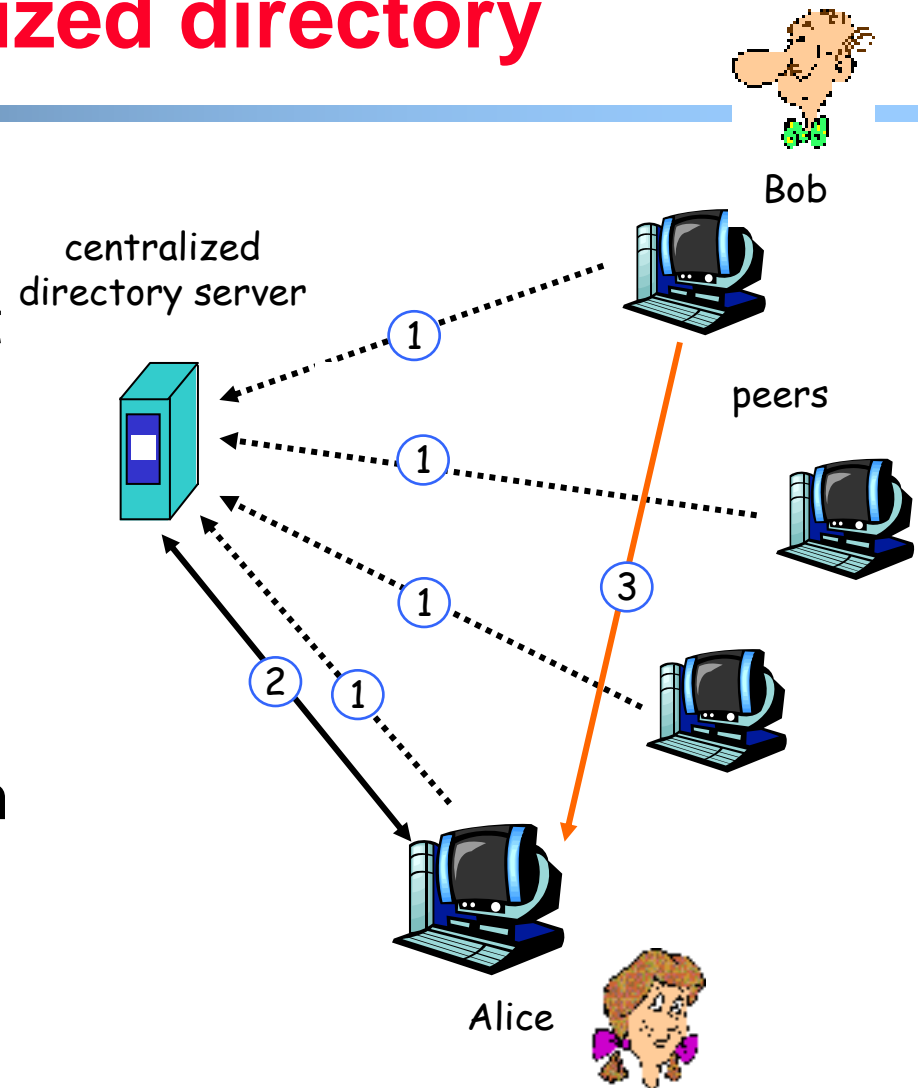
original “Napster” design

1) when peer connects, it informs central server:

- IP address
- content

2) Alice queries for “Hey Jude”

3) Alice requests file from Bob



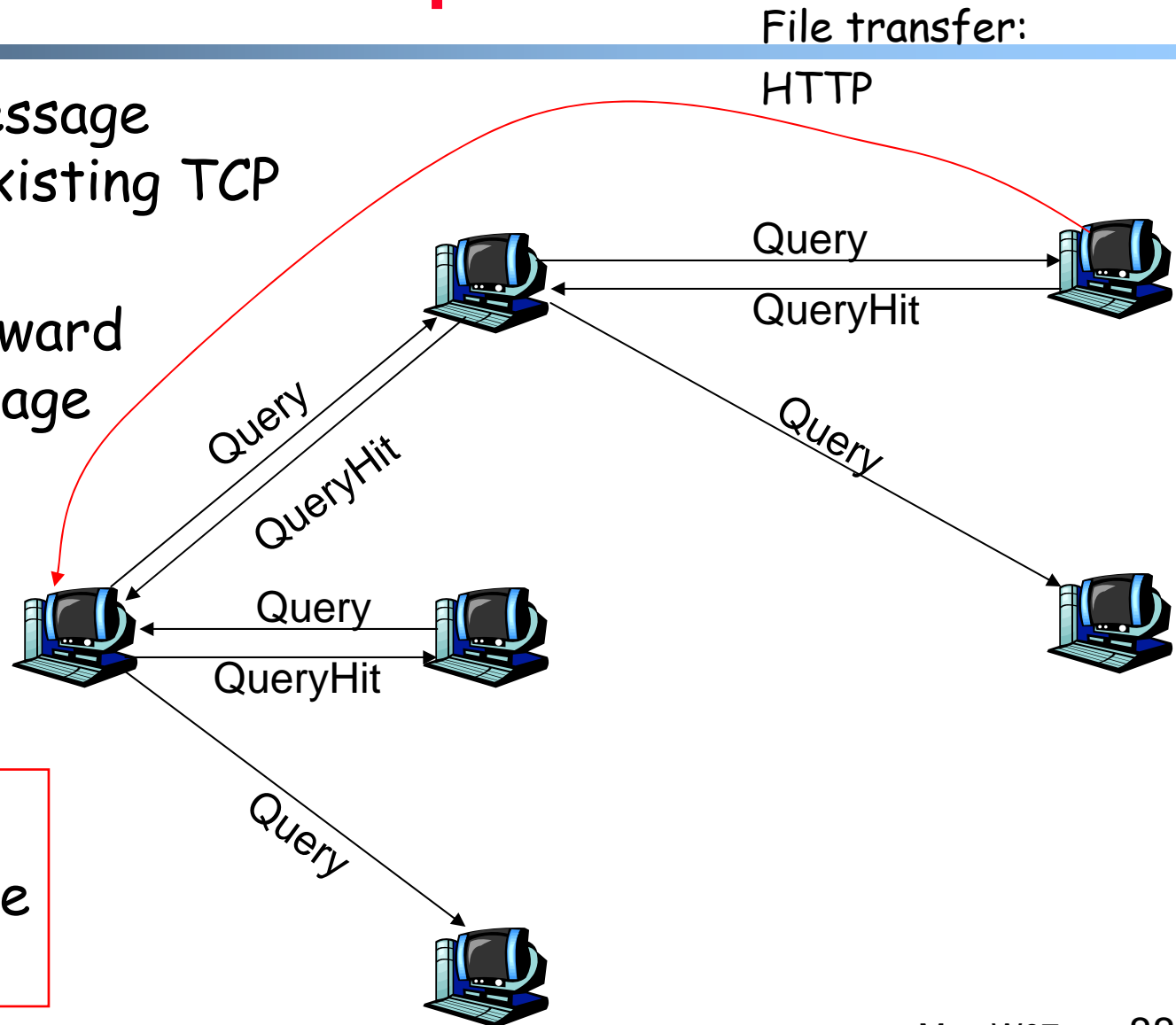
# Query flooding: Gnutella

---

- fully distributed
    - no central server
  - public domain protocol
  - many Gnutella clients implementing protocol
- **overlay network: graph**
    - edge between peer X and Y if there's a TCP connection
    - all active peers and edges is overlay net
    - Edge is not a physical link
    - Given peer will typically be connected with  $< 10$  overlay neighbors

# Gnutella: protocol

- ❑ Query message sent over existing TCP connections
- ❑ peers forward Query message
- ❑ QueryHit sent over reverse path



Scalability:  
limited scope  
flooding



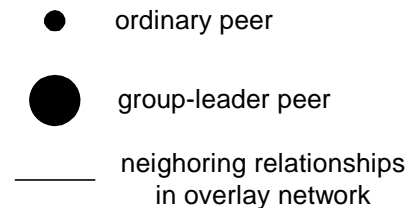
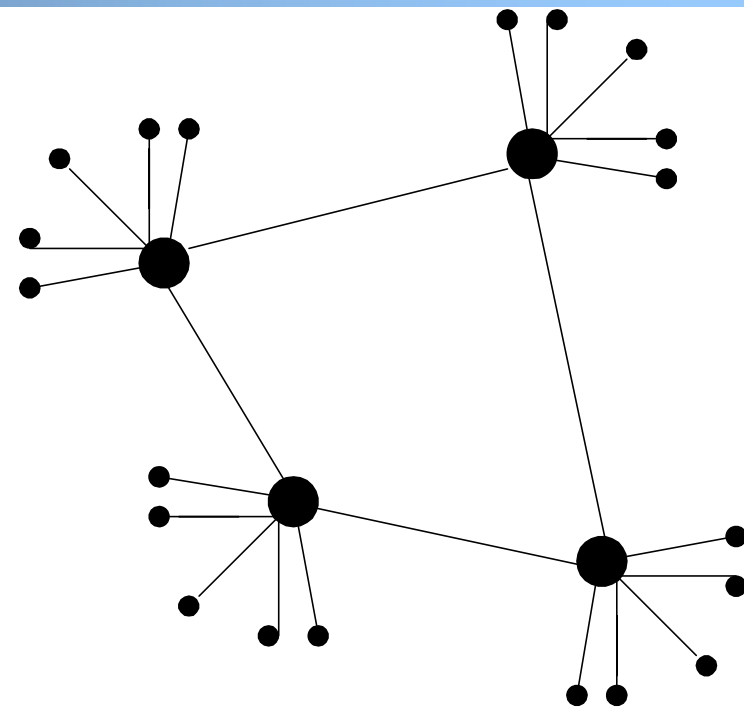
# Gnutella: Peer joining

---

1. Joining peer X must find some other peer in Gnutella network: use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message.
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections

# Exploiting heterogeneity: KaZaA

- Each peer is either a group leader or assigned to a group leader.
  - TCP connection between peer and its group leader.
  - TCP connections between some pairs of group leaders.
- Group leader tracks the content in all its children.



# KaZaA: Querying

---

- Each file has a hash and a descriptor
- Client sends keyword query to its group leader
- Group leader responds with matches:
  - For each match: metadata, hash, IP address
- If group leader forwards query to other group leaders, they respond with matches
- Client then selects files for downloading
  - HTTP requests using hash as identifier sent to peers holding desired file

# Kazaa tricks

---

- Limitations on simultaneous uploads
- Request queuing
- Incentive priorities
- Parallel downloading

# Discussions

---

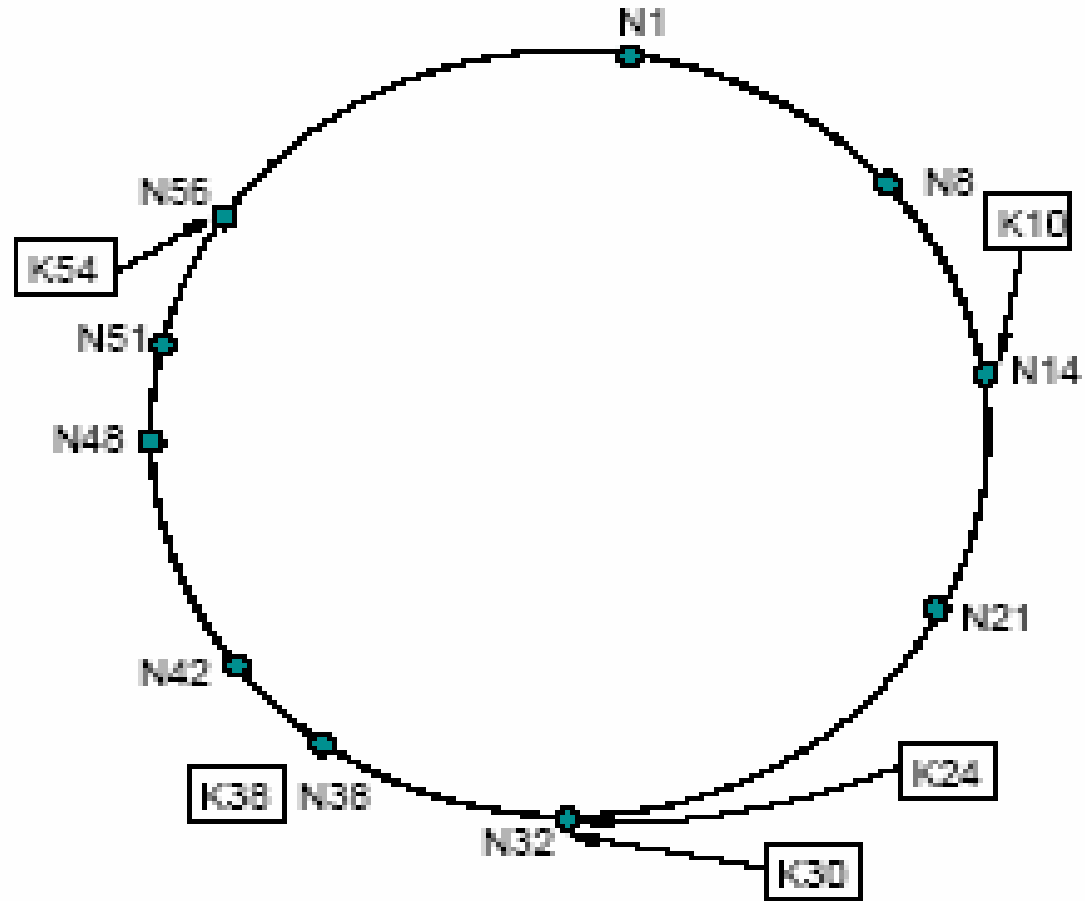
- How would you design a P2P system that is scalable, decentralized, and guarantees the location of the files?
- One solution: DHT (Distributed Hash Table)
  - Guarantees that you can find the file
  - Mapping between the file and the node ID
  - Consistent hashing function assigns each node and key an m-bit identifier using SHA-1 base hash function.

# Chord protocol

---

- Consistent hashing function assigns each node and key an  $m$ -bit identifier using SHA-1 base hash function.
- Node's IP address is hashed.
- Identifiers are ordered on a identifier circle modulo  $2^m$  called a chord ring.
- $successor(k)$  = first node whose identifier is  $\geq$  identifier of  $k$  in identifier space.

# Chord protocol



$m = 6$   
10 nodes