IF2230 Jaringan Komputer

Application Layer

Robithoh Annur
Andreas Bara Timur
Monterico Andrian
Prof. Nana Rachmana

Building Networked Applications/Systems

- Naming Issues
- Application Architectures
  - Client-Server vs. Peer-to-Peer
- Look-up Systems
  - Hierarchical: DNS
  - Peer-to-Peer: Unstructured vs. Structured
- API and Transport Layer Services: TCP, UDP, RTP
- Example Applications and Application Layer Protocols
  - HTTP, SMTP, SIP, …
- Content Distribution Networks (read yourself)

Objectives

- Understand
  - Service requirements applications place on network infrastructure
  - Protocols applications use to implement applications
- Conceptual + implementation aspects of network application protocols
  - client server paradigm
  - peer-to-peer paradigm (DHT)
- Learn about protocols by examining popular application-level protocols
  - DNS
  - Wide Wide Web, Web Caching
  - Electronic Mail

Common Applications and Requirements

"Non-Interactive" Data Transfer of Various Types

- web download/upload, ftp of text files, images, audio, video, etc.
- sending or retrieving of emails (from mail servers)
- file sharing, podcasting, …

Requirements

- 100% reliability (no data loss)

-- may be relaxed for images/audio/video

Desirables/Comments

- fast response time desirable
- bursty and "bandwidth-elastic"

-- allocate as much as possible

"Interactive" Text-based Applications

- telnet and other remote terminal operations
- instant messaging
- interactive on-line gaming, ……

Requirements

- 100% reliability (no data loss)
- short message delay (time scale ~ 10s?)

Desirables/Comments

- generally require low bandwidth
- bursty

Common Applications and Requirements…

Streaming (Stored) Audio/Video

- Internet radio
- IPTV/video playback on-demand

Requirements
- can tolerate some data loss
- can tolerate some "start-up" delay
- delay-sensitive, threshold for minimal quality

Desirables/Comments
- minimal bandwidth needed
- as good quality as possible (but more bw maybe wasteful!)
- fast start-up desirable

"Real-Time" or "Interactive" Audio/Video Applications

- VoIP, audio/video conferencing
- real-time audio/video broadcasting
- Interactive multimedia on-line gaming

Requirements
- can tolerate some data loss
- delay-sensitive, threshold for minimal quality

Desirables/Comments
- minimal bandwidth needed
- as good quality as possible (but more bw maybe wasteful!)

Summary: Application Requirements

Data loss

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

Timing

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

Bandwidth

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

Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	loss-tolerant	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
financial apps	no loss	elastic	yes and no

Network Transport Services

Functionalities

- end host to end host communication services
- "packaging" what network provides to serve common application needs
- introduce common abstractions and service interfaces

What network (Internet) provides:

"Best-Effort" Datagram Service

- data delivered in chunks (packets)
- no guarantee of delivery time
  - try to deliver as fast as possible
- may be delivered out of order
- may be lost
  - no effort made to recover them
- may even be duplicated

What applications want:

- data consist of byte-stream with application semantics
- many want reliable data delivery (no data loss)
- some want timely data delivery
- ….

What Transport Services?

Network provides "shoddy" service, what "packaging" services can we provide end-to-end?

Unreliable, datagram service:

UDP

- minimal "packaging" except service interfaces & mux/demux functions
- leave everything to apps

Unreliable, real-time service:

RTP/RTCP (wrapped over UDP)

- ordered media stream
  - for loss detection, no recovery
- timestamp
  - to assist playback and other ops
- other info
  - time, inter-media sync., loss/delay reports, …

Reliable, "virtual pipe" service :

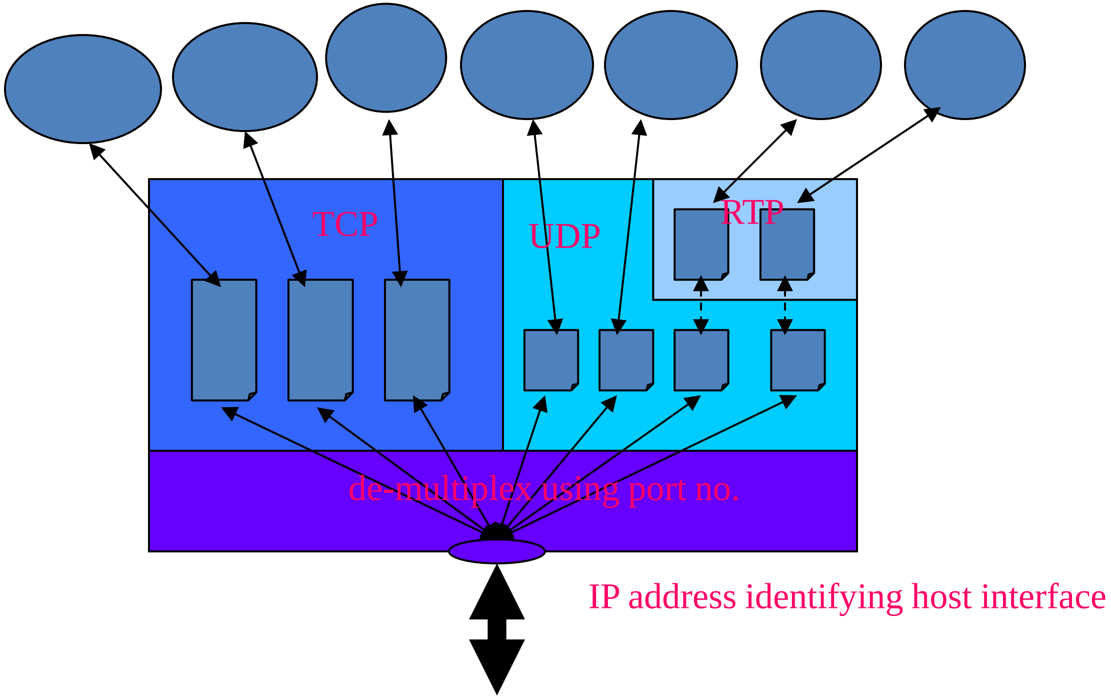
TCP

- connection-oriented
  - maintain connection states at end hosts
  - ordered byte streams within a connection
- reliable data delivery
  - re-order data, recover lost data, remove duplicates
  - no effort made to recover the packets with error
- flow and congestion control
  - Try not overflow receiver or network

Internet apps: their protocols and transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NFS	TCP or UDP
Internet telephony	proprietary (e.g., Vonage)	typically RTP/UDP

Apps, Transport Services and Network



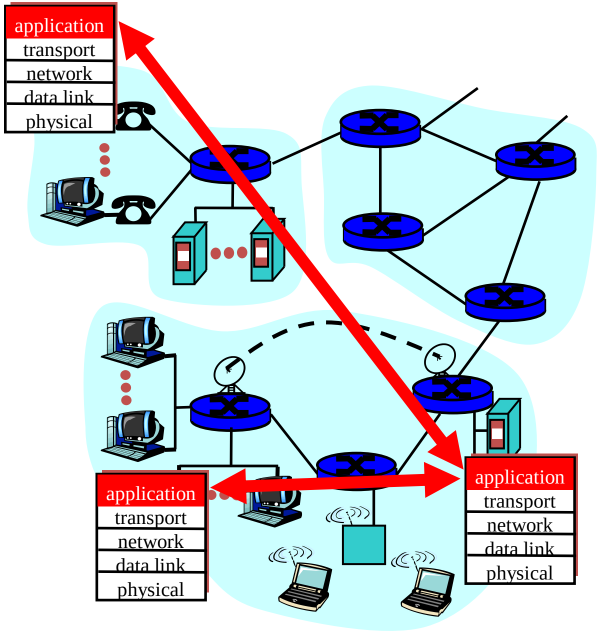
Applications and Application-Layer Protocols

Application: communicating, distributed processes

- running in network hosts in “user space”
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols

- one “piece” of an app
- define messages exchanged by apps and actions taken
- use services provided by lower layer protocols



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 (via application layer protocol)

Client process: process that initiates communication

Server process: process that waits to be contacted

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

Application Programming Interface

API: application programming interface

- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

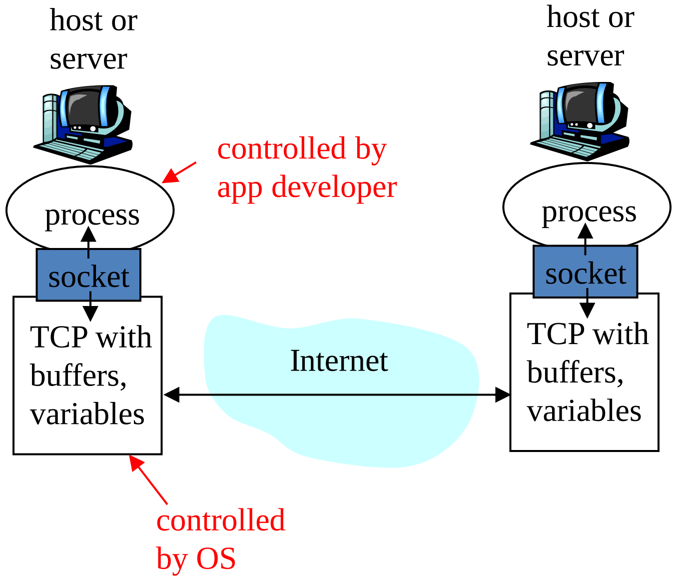
Q: how does a process "identify" the other process with which it wants to communicate?

- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

… lots more on this later.

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 which brings message to socket at receiving process



- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

Addressing processes

- For a process to receive messages, it must have an identifier
- A host has a unique32-bit IP address
- Q: does the IP address of the host on which the process runs suffice for identifying the process?
- Answer: No, many processes can be running on same host
- Identifier includes both the IP address and port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- More on this later

App-layer protocol defines

- Types of messages exchanged, eg, request & response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, ie, meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:

- eg, KaZaA

Building Networked Apps/Systems

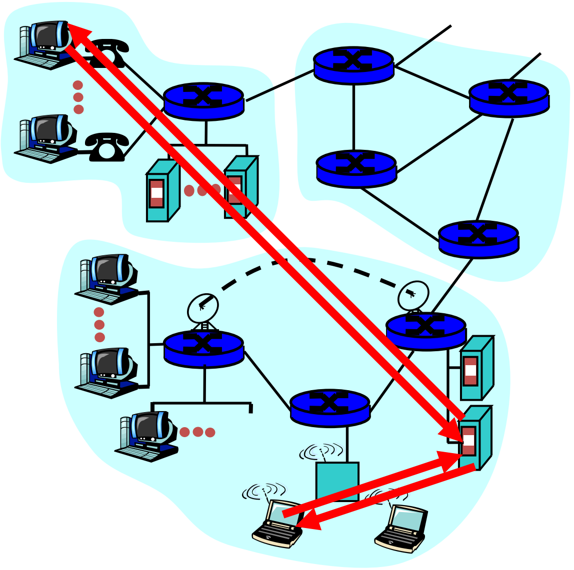
Key Issues:

- Identify and locate service/data you or other want
  - Naming service/data, and locating host and app process providing service/data
- Decide on Application/System Structure
  - Client-Server: host service/data on (“fixed”) servers
  - Peer-to-Peer: service/data provided by (“on-off”) peers
  - Hybrid?
- Session Establishment
  - Resolve names, bind app peer processes to addresses, establish sessions, exchange messages based on application-layer transfer protocols
- Presentation and Processing of Messages
  - Sender: presentation and formatting of app data for transfer
  - Receiver: interpreting and re-presentation of app data received
  - ……

Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server archicture



server:

– always-on host
– permanent IP address
– server farms for scaling

clients:

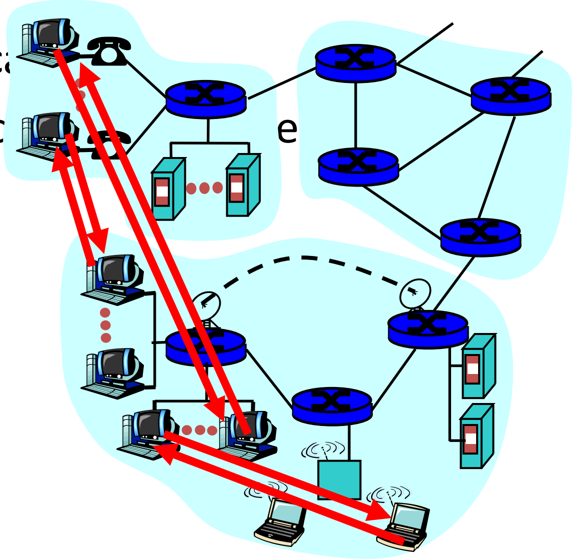
– communicate with server
– may be intermittently connected
– may have dynamic IP addresses
– do not communicate directly with each other

Pure P2P architecture

- no always on server
- arbitrary end systems directly communica
- peers are intermittently connected and c e
- example: Gnutella

Highly scalable

But difficult to manage



Hybrid of client-server and P2P

Napster

- File transfer P2P
- File search centralized:
  - Peers register content at central server
  - Peers query same central server to locate content

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

Example Application Protocols

- Domain Name Service
- Web
- Internet Mail

DNS: Domain Name System

People: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

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

Q: map between IP addresses and name ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host and 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

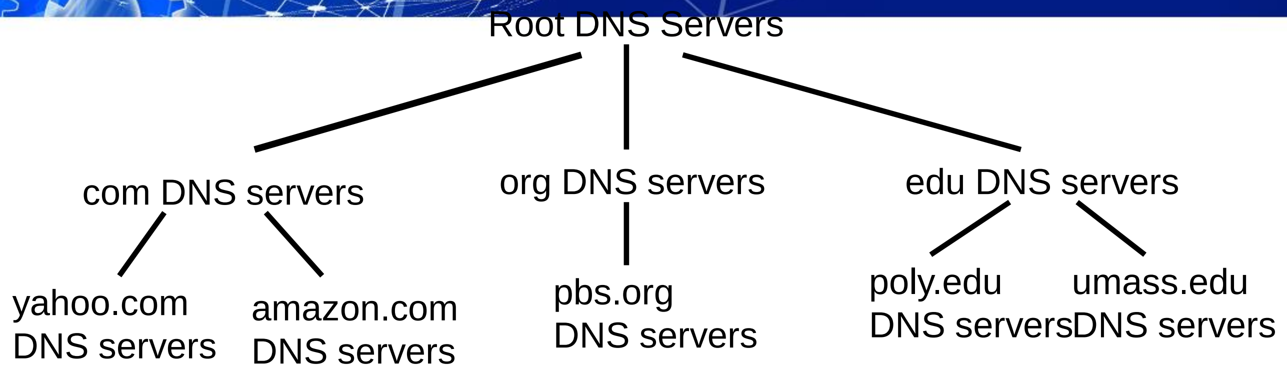
- Hostname to IP address translation
- Host aliasing
  - Canonical and alias names
- Mail server aliasing
- Load distribution
  - Replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn’t *scale!*

Distributed, Hierarchical Database

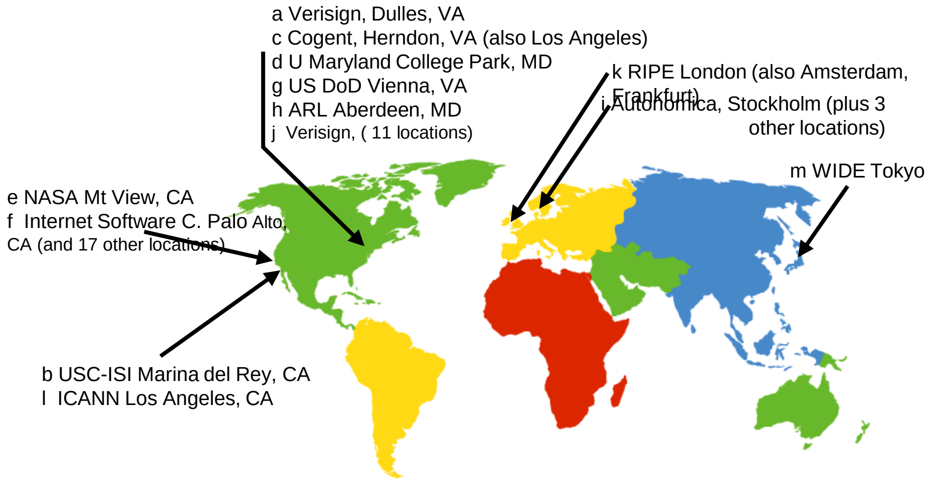


Client wants IP for www.amazon.com; 1st approx:

- 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

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



13 root name servers worldwide

TLD and Authoritative Servers

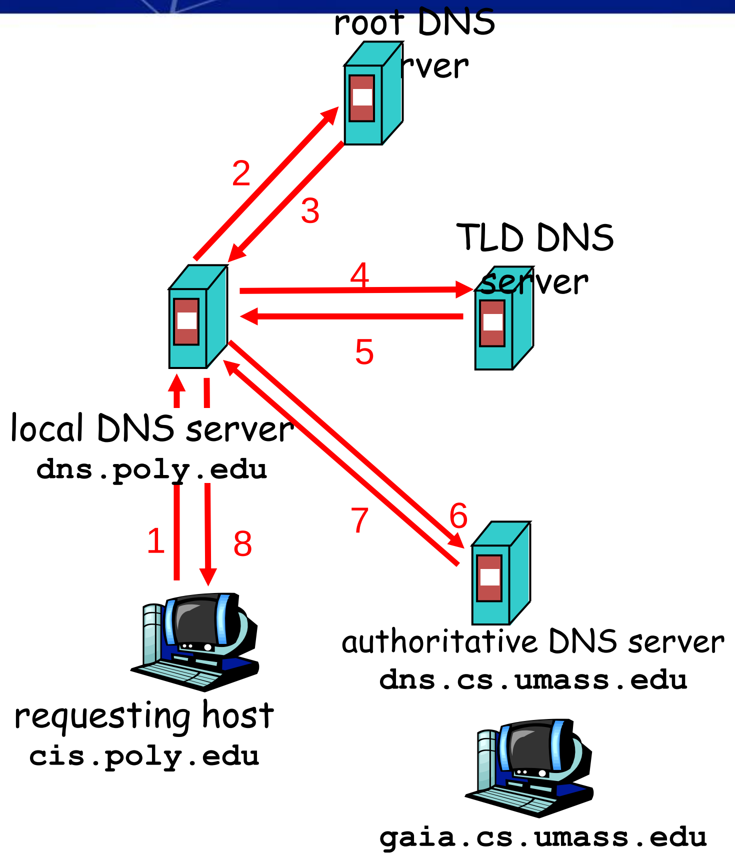
- Top-level domain (TLD) servers: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network solutions maintains servers for com TLD
  - Educause for edu TLD
- Authoritative DNS servers: organization's DNS servers, providing authoritative hostname to IP mappings 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



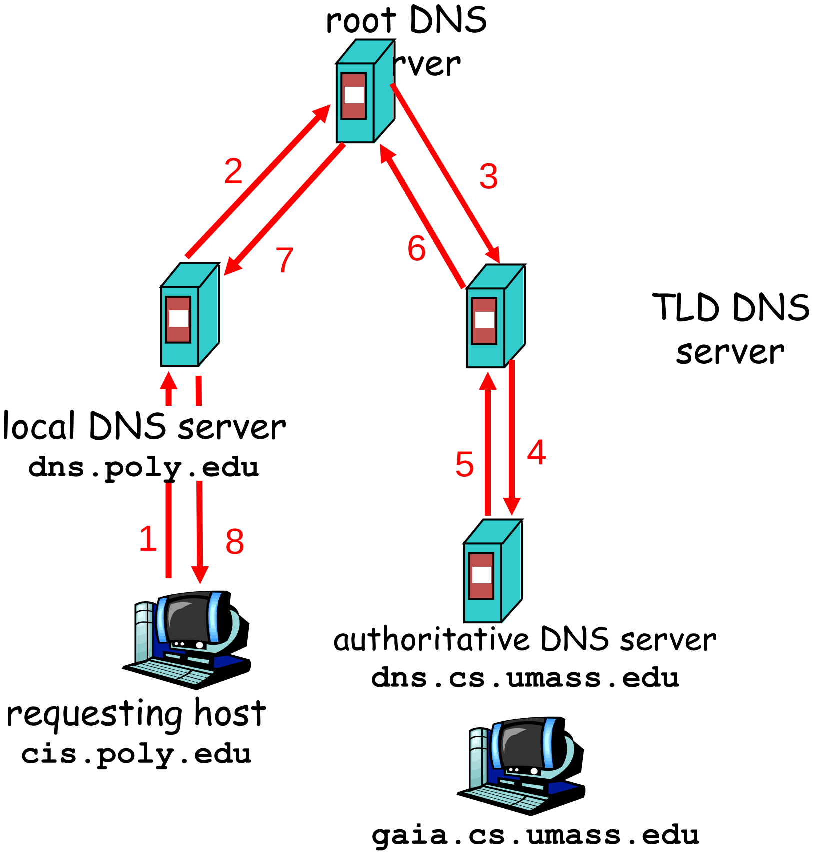
Recursive queries

<u>recursive query:</u>

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

<u>iterated query:</u>

- 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, class ttl)`

- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is domain name of the host running a name server for that domain
- Type=CNAME
  - **name** is alias name for some "cannonical" (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**

Inserting records into DNS

- Example: just created startup "Network Utopia"
- Register name networkuptopia.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.networkuptopia.com and Type MX record for networkutopia.com
- How do people get the IP address of your Web site?

Web and HTTP

First some jargon

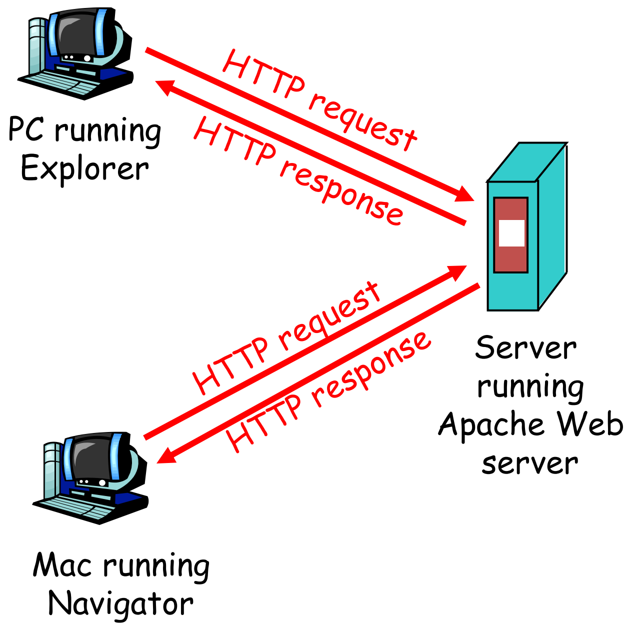
- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

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

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 single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

Suppose user enters URL `www.someSchool.edu/someDepartment/home.index` (contains text, references to 10 jpeg images)

1a. HTTP client initiates 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.)

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

time

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

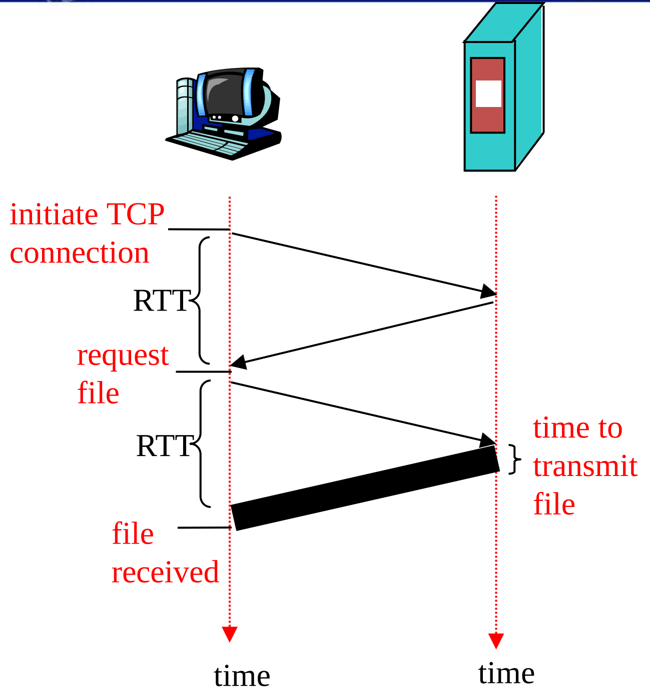
Response time modeling

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+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 response
- 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

HTTP request message

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

request line (GET, POST, HEAD commands)

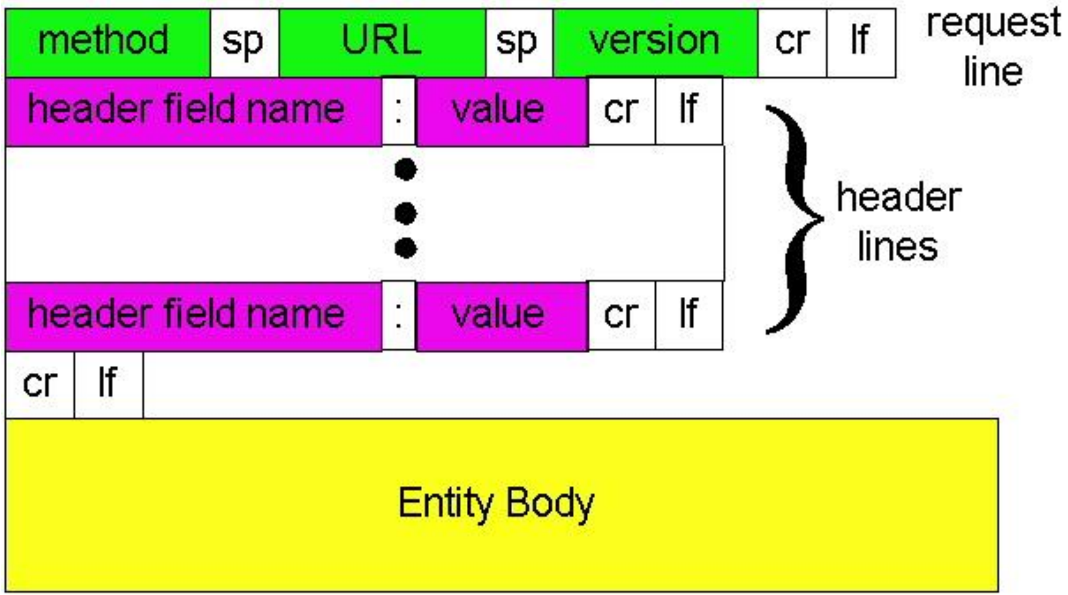
header lines

Carriage return, line feed indicates end of message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(extra carriage return, line feed)

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
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

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 subsequent HTTP request message

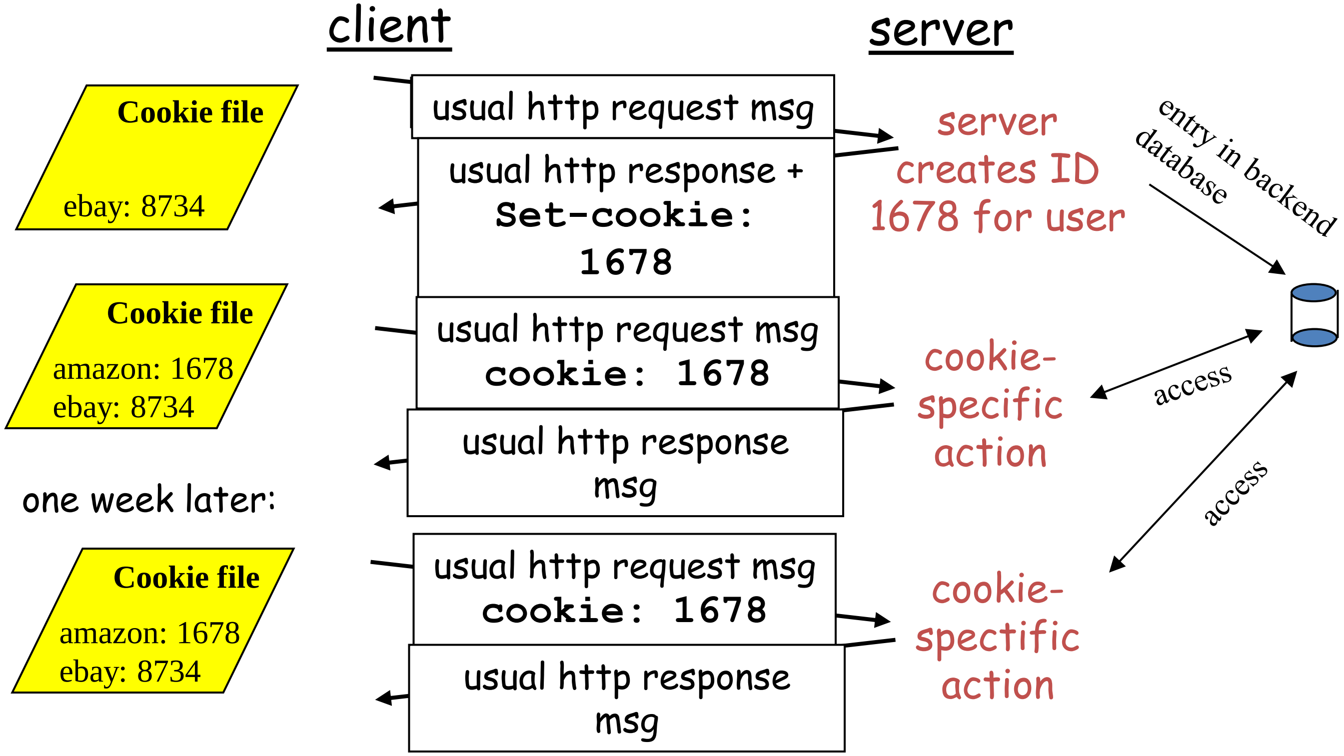
3) cookie file kept on user's host and managed by user's browser

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)

aside

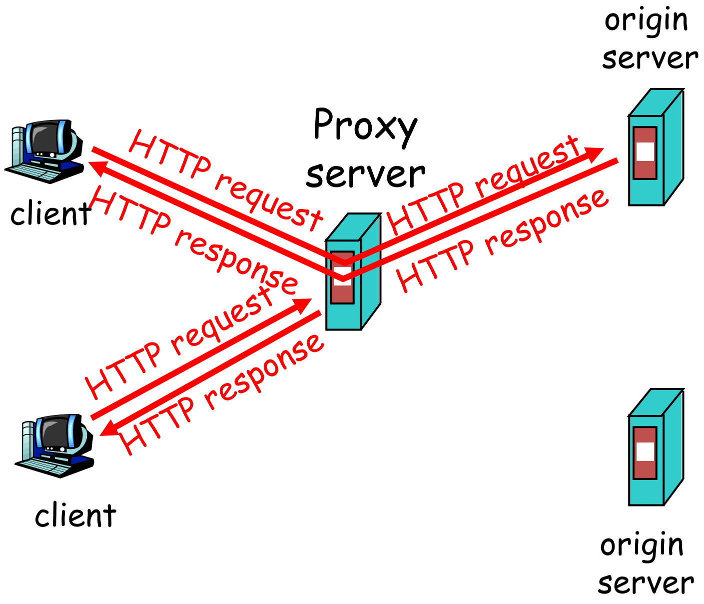
Cookies and privacy:

- 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

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



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

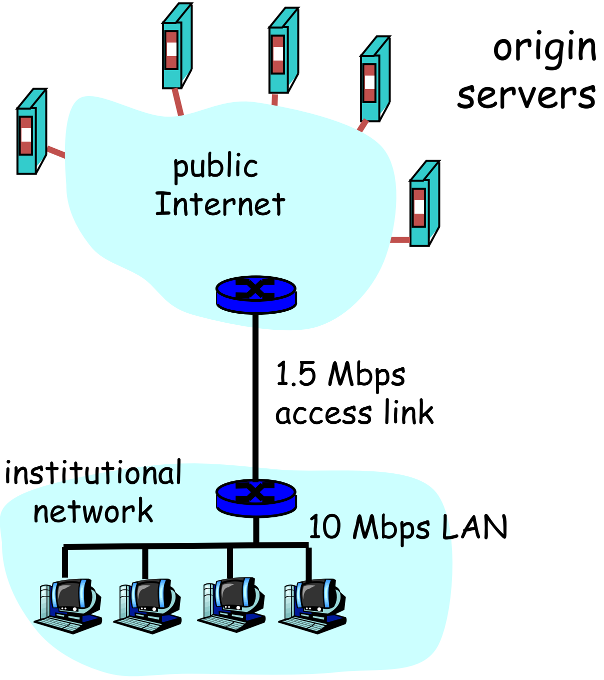
Assumptions

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

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching example (cont)

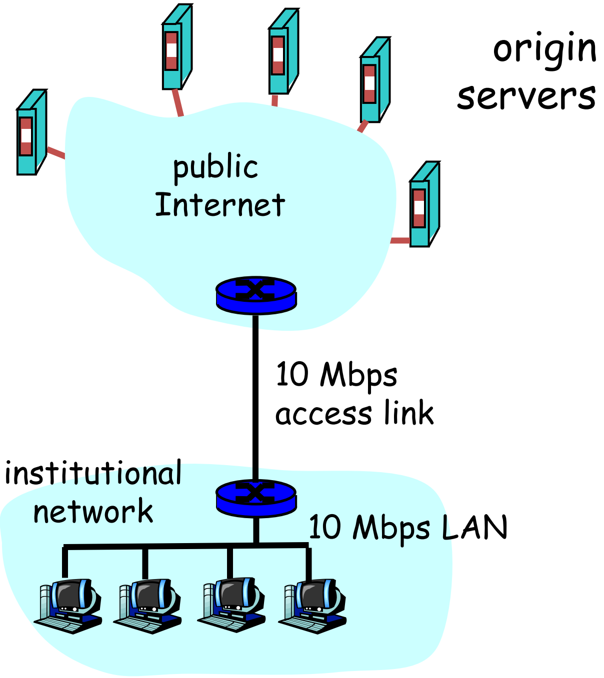
<u>Possible solution</u>

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

<u>Consequences</u>

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- often a costly upgrade



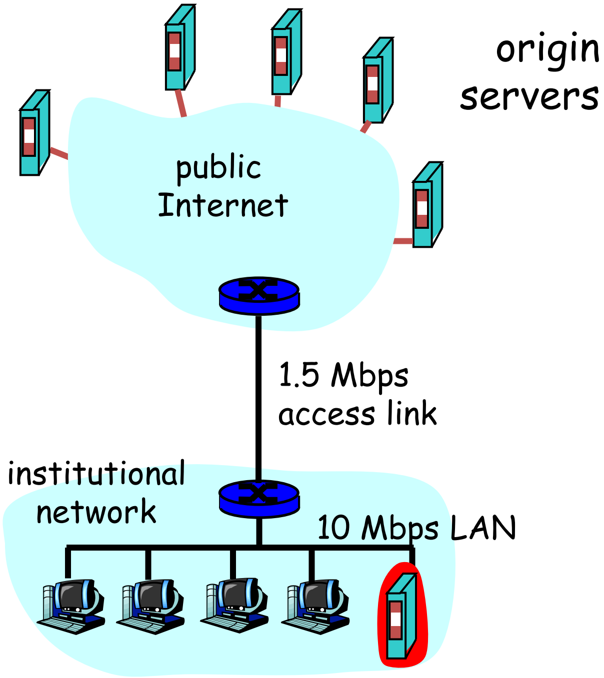
Caching example (cont)

Install cache

- suppose hit rate is .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*(2.01) secs + milliseconds < 1.4 secs

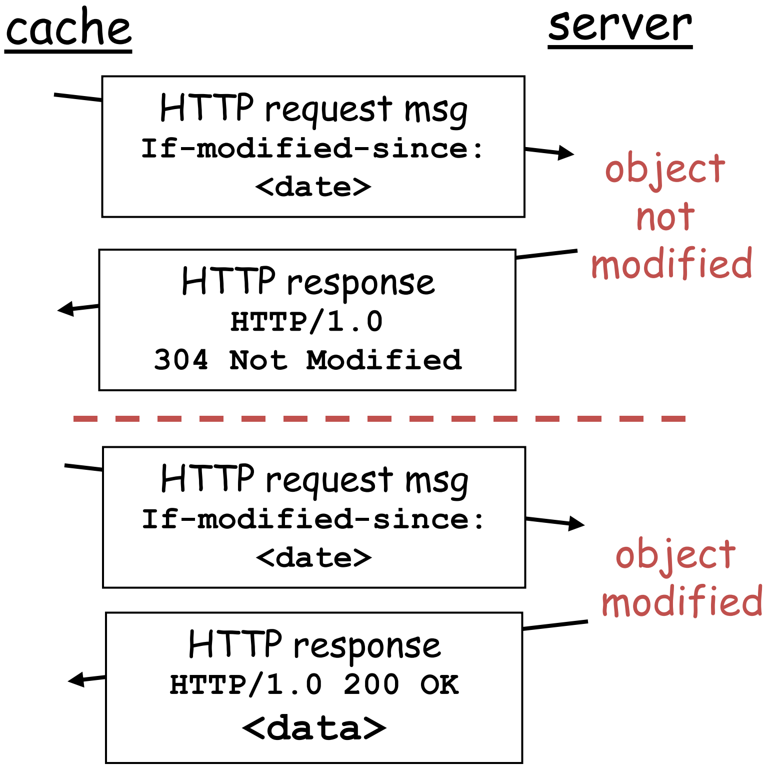


Conditional GET

- 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`



Trying out http (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.seas.gwu.edu 80
```

Opens TCP connection to port 80 (default http server port) at www.seas.gwu.edu. Anything typed in sent to port 80 at www.seas.gwu.edu

2. Type in a GET http request:

```
GET /~cheng/index.html HTTP/1.0
```

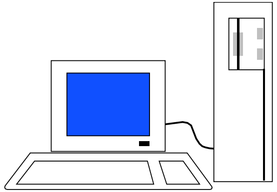
By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by http server!

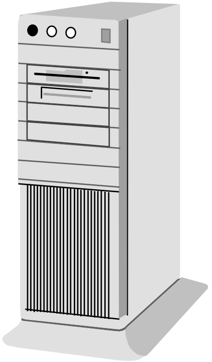
Web and HTTP Summary

Transaction-oriented (request/reply), use TCP, port 80

Client



Server



GET /index.html HTTP/1.0

HTTP/1.0
200 Document follows
Content-type: text/html
Content-length: 2090
-- blank line --
HTML text of the Web page

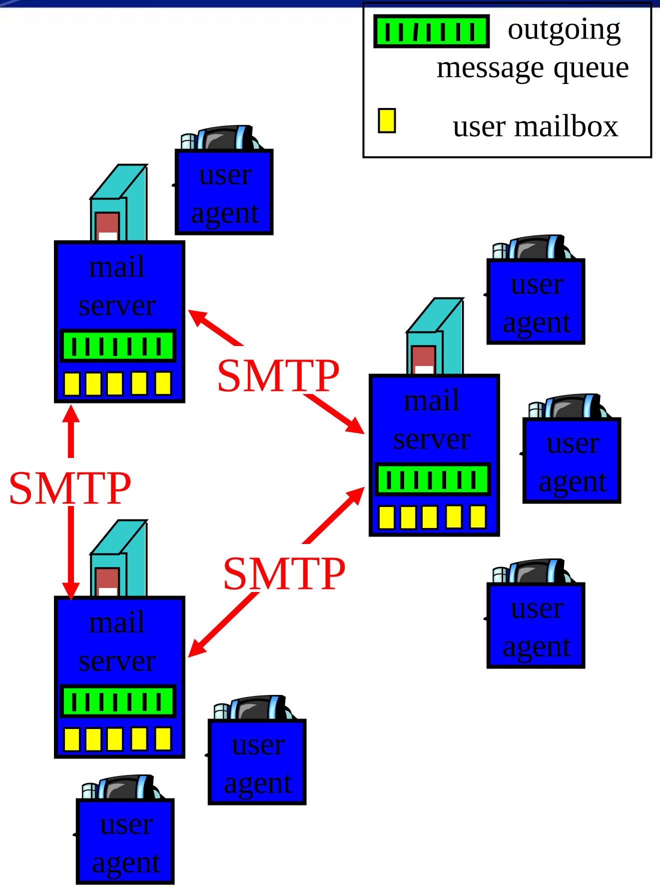
Electronic Mail

Three major components:

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

User Agent

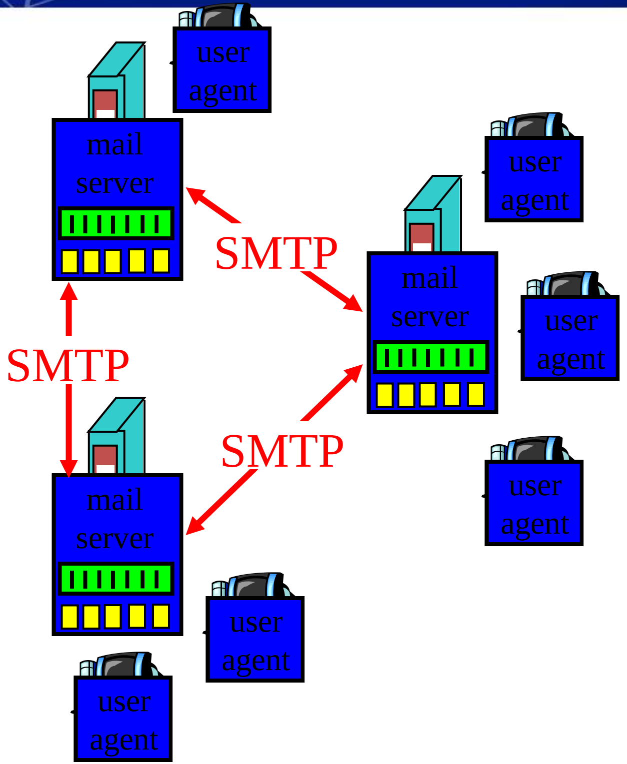
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, pine, Netscape Messenger
- outgoing, incoming messages stored on server



Electronic Mail: mail servers

Mail Servers

- mailbox contains incoming messages (yet to be read) 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 821]

- uses tcp to reliably transfer email msg 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

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 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

Comparison with http

- smtp uses persistent connections
- smtp requires that message (header & body) be in 7-bit ascii
- certain character strings are not permitted in message (e.g., `CRLF.CRLF`). Thus message has to be encoded (usually into either base-64 or quoted printable)
- smtp server uses `CRLF.CRLF` to determine end of message

- http: pull
- email: push
- both have ASCII command/response interaction, status codes
- http: each object is encapsulated in its own response message
- smtp: multiple object message sent in a multipart message

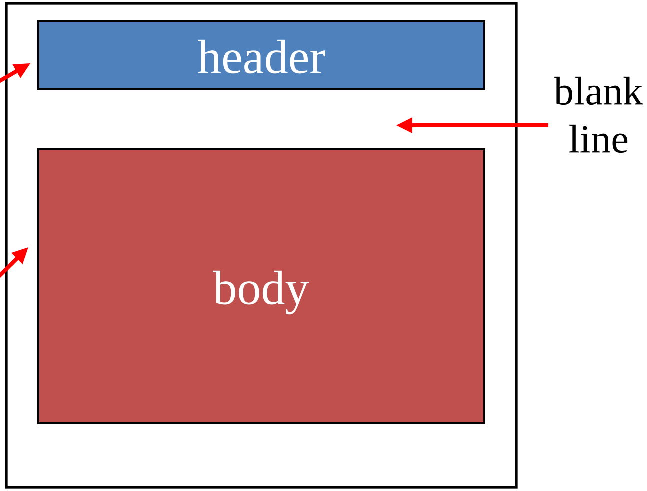
Mail Message Format

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

MIME version

method used
to encode data

multimedia data
type, subtype,
parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

MIME Types

`Content-Type: type/subtype; parameters`

Text

- example subtypes: **`plain`**, **`html`**. Eg: text/plain, or text/richtext

Image

- example subtypes: **`jpeg`**, **`gif (image/jpeg)`**

Audio

- example subtypes: **`basic`** (8-bit mu-law encoded), **`32kadpcm`** **(32 kbps coding)**

Video

- example subtypes: **`mpeg`**, **`quicktime`**

Application

- other data that must be processed by reader before "viewable"
- example subtypes: **`msword`**, **`octet-stream (application/postscript, application/msword)`**

Multipart Type

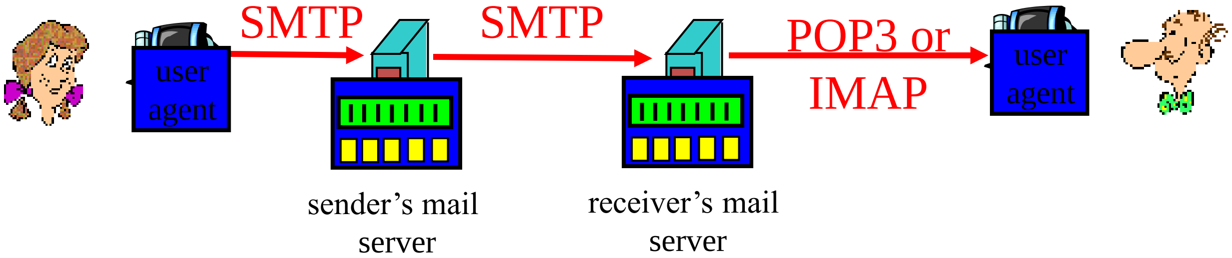
```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```

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: Gmail, , Yahoo! Mail, etc.

Email Summary