Application Layer

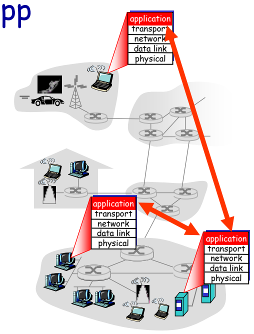
- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube)
- voice over IP
- real-time video conferencing
- cloud computing
- …
- …
-

Creating a network app

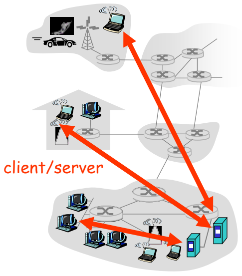
Write programs that

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

No need to write software for network-core devices



Client-server architecture



server:

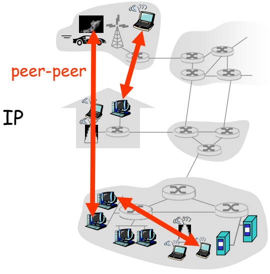
- always-on
- permanent IP address
- server farms for scaling

clients:

- communicate with server
- not always connected
- dynamic IP addresses
- do not communicate directly with each other

Pure P2P architecture

- *no* always-on server
- end systems directly communicate
- peers are not always connected and change IP addresses



Hybrid of client-server and P2P

Skype

- voice-over-IP P2P application
- server: finding address of remote party
- client-client speech: direct

Processes communicating

process: program running within a host.

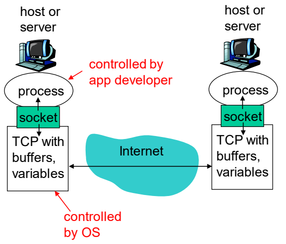
- processes communicate by exchanging messages

client process: process that initiates communication

server process: process that waits to be contacted

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door



Addressing processes

- a process must have an *identifier*
- host device has unique IP address
- *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

App-layer protocol defines

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

public-domain protocols:

- defined in RFCs
  e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

What transport service does an app need?

Data loss

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

Timing

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

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

…

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport*
- *flow control*
- *congestion control*
- *does not provide:* timing, minimum throughput guarantees

UDP service:

- unreliable data transfer between sending and receiving process

Web and HTTP

- web page consists of objects
- object can be HTML file, JPEG image, audio file,…
- web page consists of base HTML-file with referenced objects
- each object is addressable by a URL
- example URL:

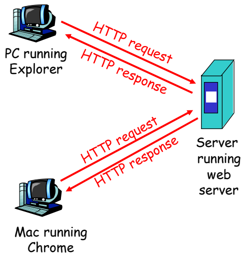
`www.someschool.edu/someDept/pic.gif`

host name   path name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser
  - *server:* Web server



HTTP overview

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

protocols that maintain "state" are complex!

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

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection.

persistent HTTP

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

HTTP request message

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

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

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->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`

User-server state: cookies

many Web sites use cookies

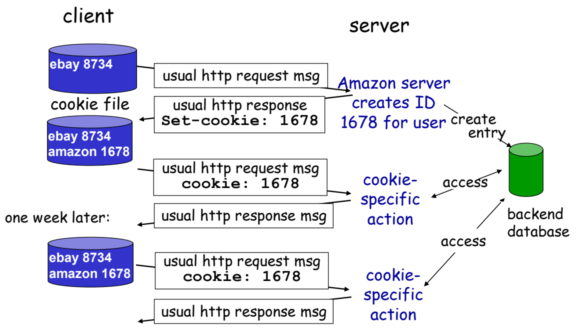
four components:

1) cookie header line of HTTP *response* message
2) cookie header line in 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.)



client

server

ebay 8734

usual http request msg

Amazon server creates ID 1678 for user

cookie file

usual http response
`Set-cookie: 1678`

create entry

ebay 8734
amazon 1678

usual http request msg
`cookie: 1678`

cookie-specific action

access

backend database

one week later:

usual http response msg

access

ebay 8734
amazon 1678

usual http request msg
`cookie: 1678`

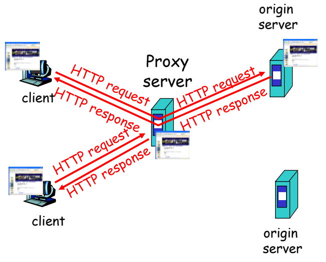
cookie-specific action

usual http response msg

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

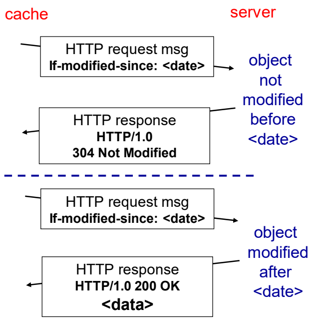
- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link

Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request
- server: response contains no object if cached copy is up-to-date:



cache

server

HTTP request msg
If-modified-since: <date>

object not modified before <date>

HTTP response
HTTP/1.0
304 Not Modified

HTTP request msg
If-modified-since: <date>

object modified after <date>

HTTP response
HTTP/1.0 200 OK
<data>

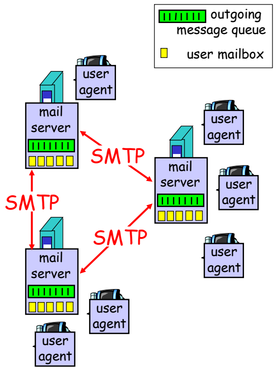
Email

Three major components:

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

User Agent

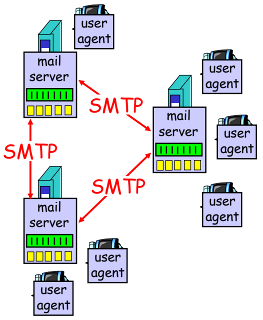
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, elm, Mozilla Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



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



SMTP

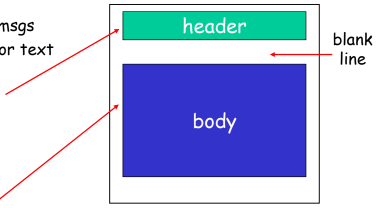
- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction
  - commands: ASCII text
  - response: status code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

Mail message format

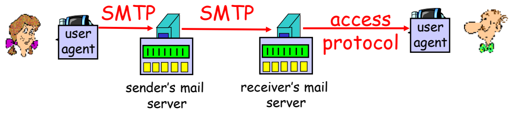
SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:
- body
  - the "message"



Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

DNS

DNS services

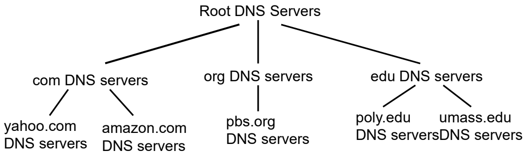
- hostname to IP address translation
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

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

doesn't *scale!*

Distributed, Hierarchical Database



Root DNS Servers

com DNS servers

org DNS servers

edu DNS servers

yahoo.com DNS servers

amazon.com DNS servers

pbs.org DNS servers

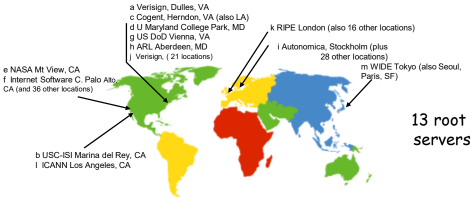
poly.edu DNS servers

umass.edu DNS servers

client wants IP for www.amazon.com:

- 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



13 root name servers worldwide

TLD and 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

Authoritative DNS servers:

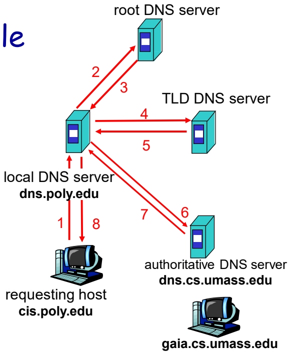
- organization's DNS servers, hostname to IP mappings for organization's servers
- can be maintained by organization or service provider

DNS name resolution example

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

iterated query:

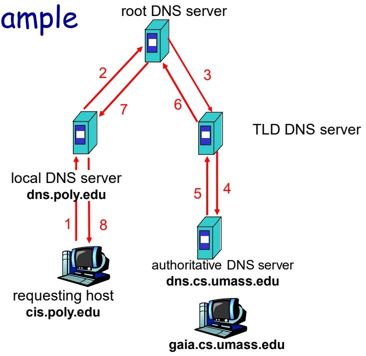
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

- puts burden of name resolution on contacted name server

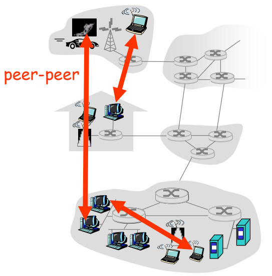


DNS: caching and updating records

- 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

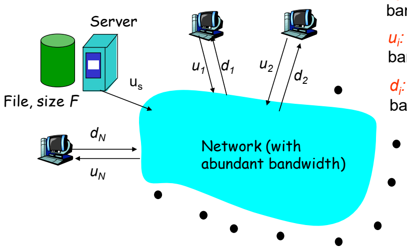
Pure P2P architecture

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



File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to N peers?

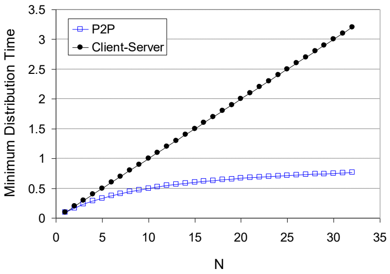


u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

Server-client vs. P2P: example

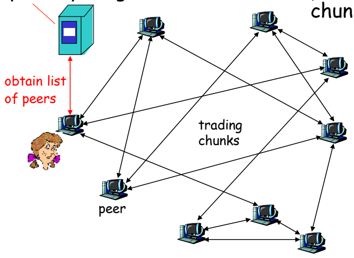


File distribution: BitTorrent

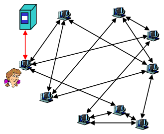
P2P file distribution

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



BitTorrent



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4

P2P example: Skype

- pairs of users communicate
- proprietary application-layer protocol
- Index maps usernames to IP addresses