Application Layer

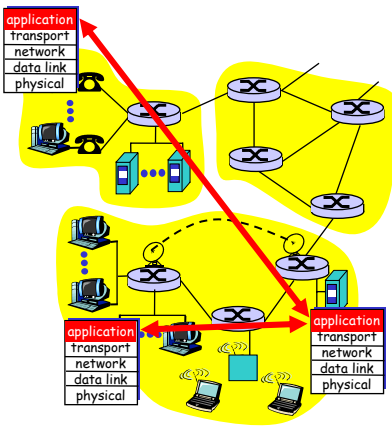
Goals:

- Conceptual aspects of network application protocols
  - Client server paradigm
  - Service models
- Learn about protocols by examining popular application-level protocols
  - HTTP
  - DNS

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
- User services provided by lower layer protocols



Client-server paradigm

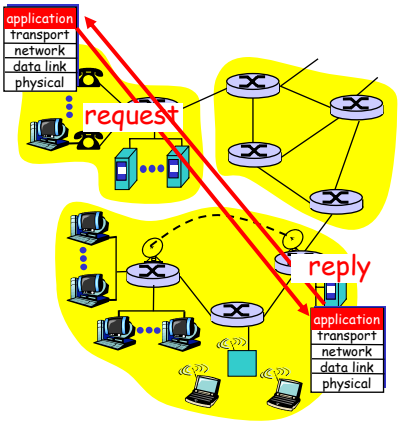
Typical network app has two pieces: *client* and *server*

Client:

- Initiates contact with server ("speaks first")
- Typically requests service from server,
- E.g., request WWW page, send email

Server:

- Provides requested service to client
- E.g., sends requested WWW page, receives/stores received email



Services provided by Internet transport protocols

TCP service:

- *Connection-oriented:* setup required between client, server
- *Reliable transport* between sending and receiving process
- *Flow control:* sender won't overwhelm receiver
- *Congestion control:* throttle sender when network overloaded
- *Does not providing:* timing, minimum bandwidth guarantees

UDP service:

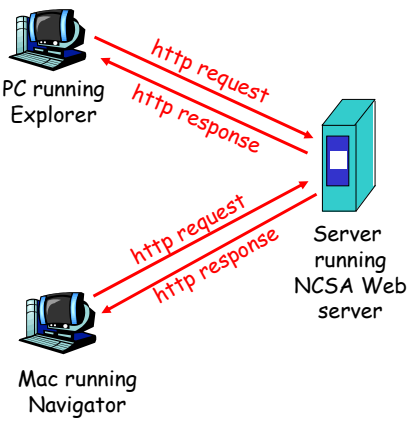
- Unreliable data transfer between sending and receiving process
- Does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: Why bother? Why is there a UDP?

WWW: the HTTP protocol

HTTP: hypertext transfer protocol

- WWW's application layer protocol
- Client/server model
  - *Client:* browser that requests, receives, "displays" WWW objects
  - *Server:* WWW server sends objects in response to requests
- HTTP/1.0: RFC 1945
- HTTP/1.1: RFC 2616



The HTTP protocol: more

HTTP: TCP transport service:

- 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 WWW server (http server)
- TCP connection closed

HTTP is "stateless"

- Server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

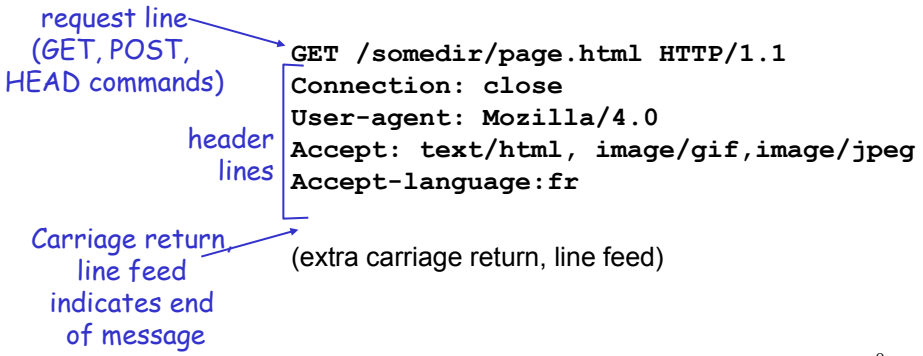
- Past history (state) must be maintained
- If server/client crashes, their views of "state" may be inconsistent, must be reconciled

The HTTP protocol: even more

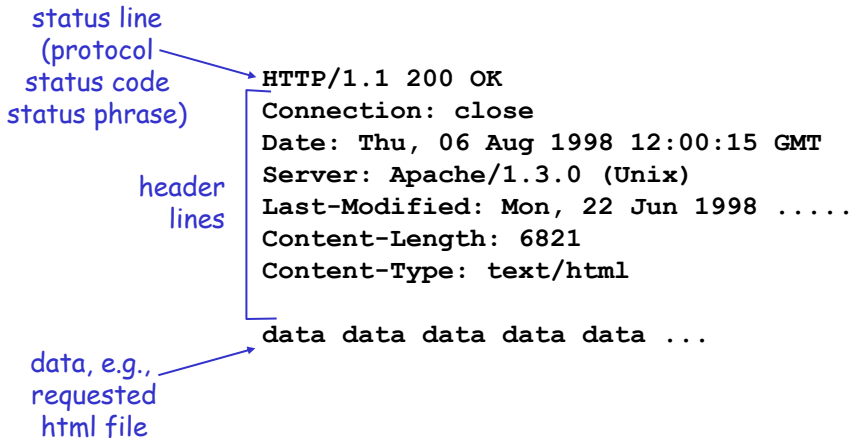
- Non-persistent connection:
  One object in each TCP connection
  - Some browsers create multiple TCP connections *simultaneously* – one per object
- Persistent connection:
  Multiple objects transferred within one TCP connection
- Pipelined persistent connections:
  Multiple requests issued without waiting for response

http message format: request

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



http message format: reply



http reply 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

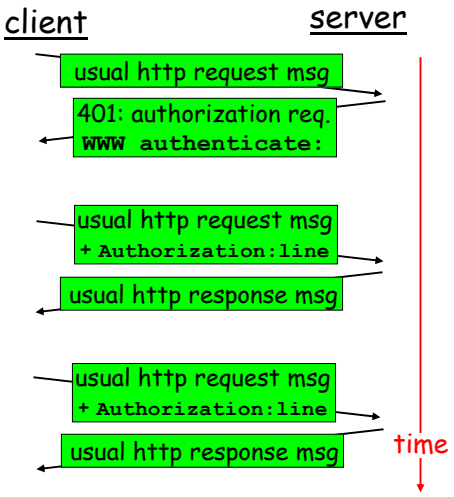
- requested document not found on this server

505 HTTP Version Not Supported

User-server interaction: authentication

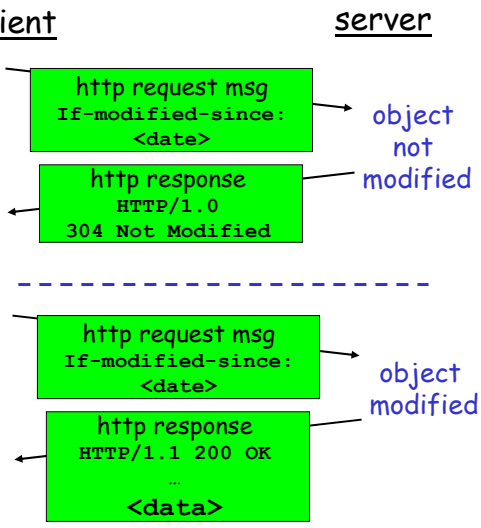
Authentication goal: control access to server documents

- Stateless: client must present authorization in each request
- Authorization: typically name, password
  - `authorization:` header line in request
  - If no authorization, server refuses access, sends `WWW authenticate:` header line in response



User-server interaction: conditional GET

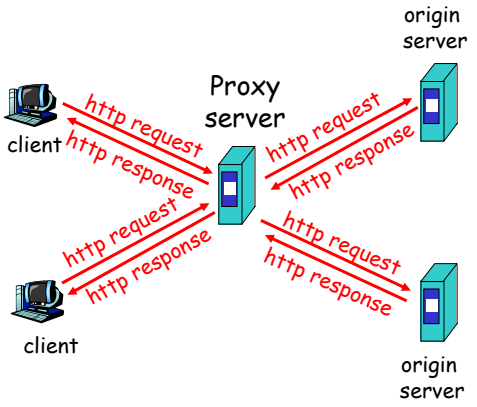
- Goal: don't send object if client has up-to-date stored (cached) version
- Client: specify date of cached copy in http request
  `If-modified-since: <date>`
- Server: response contains no object if cached copy up-to-date:
  `HTTP/1.0 304 Not Modified`



Web Caches (proxy server)

Goal: satisfy client request without involving origin server

- User sets browser: WWW accesses via web cache
- Client sends all http requests to web cache
  - If object at web cache, web cache immediately returns object in http response
  - Else requests object from origin server, then returns http response to client



DNS: Domain Name System

People: many identifiers:

- SSN, name, Passport #

Internet hosts, routers:

- IP address (32 bit) – used for addressing datagrams
- "name", e.g., gaia.cs.umass.edu – used by humans

Q: Map between IP addresses and name?

- Secure Domain Name System (DNS) Dynamic Update: RFC 3007

DNS: Domain Name System

Domain Name System:

- *Distributed database:* implemented in hierarchy of many *name servers*
- *Application-layer protocol:* host, routers, name servers communicate to *resolve* names (address/name translation)
  - Core Internet function implemented as application-layer protocol
  - Complexity at network's "edge"

DNS name servers

Why not centralize DNS?

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance

Does not *scale!*

DNS name servers (2)

No server has all name-to-IP address mappings

Local name servers:

- Each ISP, company has *local (default) name server*
- Host DNS query first goes to local name server

Authoritative name server:

- For a host: stores that host's IP address, name
- Can perform name/address translation for that host's name

DNS records

DNS: distributed db storing resource records (RR)

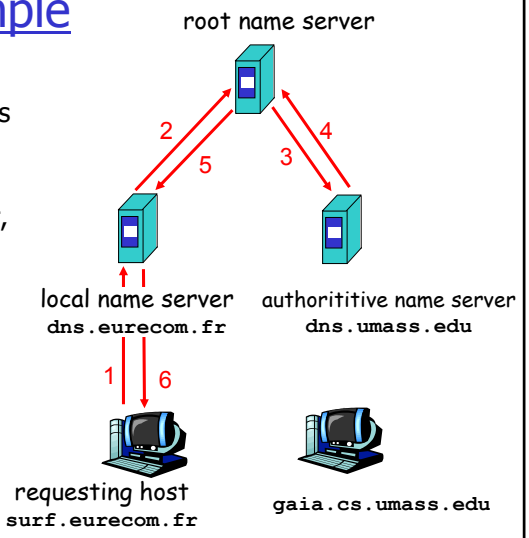
RR format: `(name, value, type, ttl)`

- 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
  - for alias
- Type=MX
  - for mail

Simple DNS example

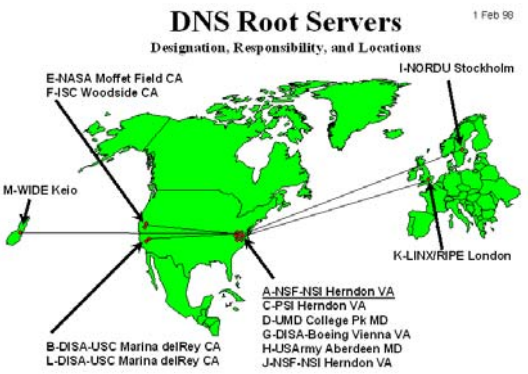
Host **surf.eurecom.fr** wants IP address of **gaia.cs.umass.edu**

1. Contacts its local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** contacts root name server, if necessary
3. Root name server contacts authoritative name server, **dns.umass.edu**, if necessary



DNS: Root name servers

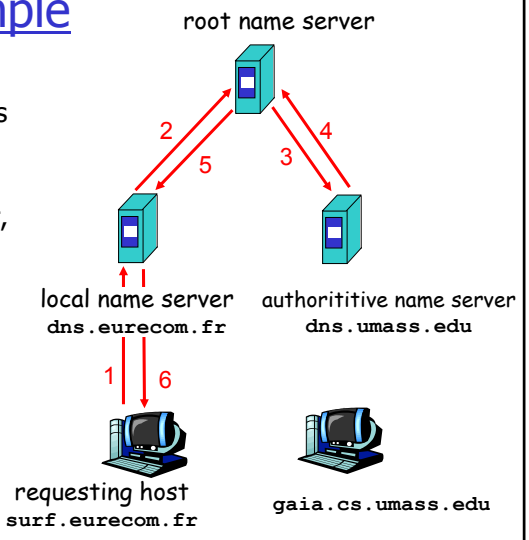
- Contacted by local name server that can not resolve name
- Root name server:
  - Contacts authoritative name server if name mapping not known
  - Gets mapping
  - Returns mapping to local name server
- ~ dozen root name servers worldwide



Simple DNS example

host **surf.eurecom.fr** wants IP address of **gaia.cs.umass.edu**

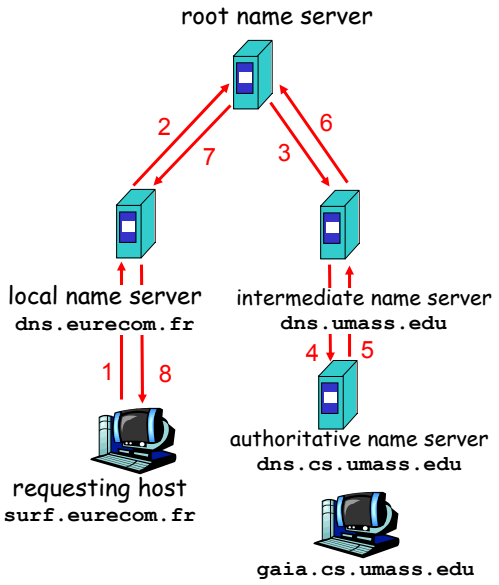
1. Contacts its local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** contacts root name server, if necessary
3. Root name server contacts authoritative name server, **dns.umass.edu**, if necessary



DNS example

Root name server:

- May not know authoritative name server
- May know *intermediate name server:* who to contact to find authoritative name server



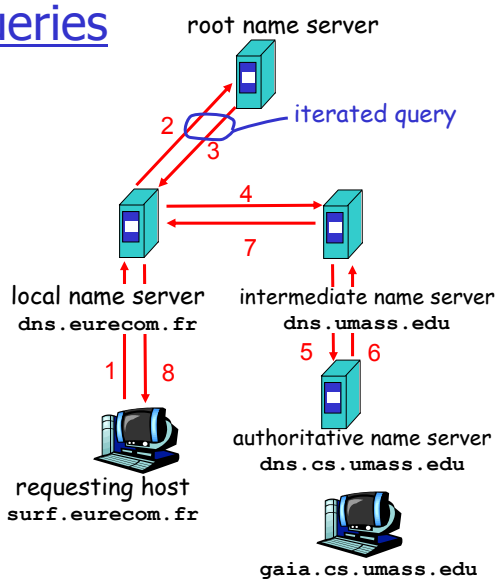
DNS: iterated queries

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
- Update/notify mechanisms under design by IETF
  - RFC 3007 (Feb. 2004)
  - http://www.ietf.org/html.charters/dnsind-charter.html