### Internet apps: their protocols and transport protocols

<table>
<thead>
<tr>
<th>Application</th>
<th>Application layer protocol</th>
<th>Underlying transport protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>smtp [RFC 821]</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>telnet [RFC 854]</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>http [RFC 2068]</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>ftp [RFC 959]</td>
<td>TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>proprietary</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td></td>
<td>(e.g. RealNetworks)</td>
<td></td>
</tr>
<tr>
<td>remote file server</td>
<td>NSF</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>proprietary</td>
<td>typically UDP</td>
</tr>
<tr>
<td></td>
<td>(e.g., Vocaltec)</td>
<td></td>
</tr>
</tbody>
</table>

### 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
- http1.0: RFC 1945
- http1.1: RFC 2068
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

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 example

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. Port 80 is default for http server.
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
3. http server receives request message, forms response message containing requested object (someDepartment/home.index), sends message into socket
**http example (cont.)**

- **4.** http server closes TCP connection.

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

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

- **- non-persistent 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

---

**http message format: request**

- **two types of http messages:** *request, response*

- **http request message:**
  - *ASCII* (human-readable format)

  ```
  GET /somedir/page.html HTTP/1.1
  Connection: close
  User-agent: Mozilla/4.0
  Accept: text/html, image/gif, image/jpeg
  Accept-language: fr
  ```

  (extra carriage return, line feed)
**HTTP request message: general format**

- **Method**: The HTTP method (e.g., GET, POST).
- **URL**: The Uniform Resource Locator.
- **Version**: The HTTP version (e.g., HTTP/1.1).
- **Header lines**: Additional header fields.
- **Entity Body**: The body of the request.

**HTTP message format: reply**

- **Status line**: HTTP response status code and message (e.g., HTTP/1.1 200 OK).
- **Header lines**: Additional header fields.
- **Entity body**: The body of the response, e.g., HTML content.
http reply status codes

In first line in server->client response message.

A few sample codes:

200 OK
  o request succeeded, requested object later in this message

301 Moved Permanently
  o requested object moved, new location specified later in
    this message (Location:)

400 Bad Request
  o request message not understood by server

404 Not Found
  o requested document not found on this server

505 HTTP Version Not Supported

Trying out http (client side) for yourself

1. Telnet to your favorite WWW server:

   telnet www.eurecom.fr 80

   Opens TCP connection to port 80
   (default http server port) at www.eurecom.fr.
   Anything typed in sent
   to port 80 at www.eurecom.fr

2. Type in a GET http request:

   GET /-ross/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!
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 presented, server refuses access, sends
    WWW authenticate: header line in response

User-server interaction: cookies

- server sends "cookie" to client in response
  Set-cookie: #
- client present cookie in later requests
  cookie: #
- server matches presented-cookie with server-stored cookies
  - authentication
  - remembering user preferences, previous choices

client

usual http request msg

usual http request msg
+ Authorization:line

usual http response msg

usual http request msg
+ Authorization:line

usual http response msg

401 authorization req.
WWW authenticate:

usual http response +
Set-cookie: #

usual http response msg

cookie-specific action

usual http request msg

cookie: #

usual http response msg

cookie-specific action

usual http request msg

cookie: #

usual http response msg

usual http response msg
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
  - object 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
Why WWW Caching?

Assume: cache is “close” to client (e.g., in same network)
- smaller response time: cache “closer” to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck

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?

Domain Name System:
- 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 name servers

Why not centralize DNS?

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

doesn't scale!

- 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: Root name servers

- contacted by local name server that cannot 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