Upper Layers
Upper Layers

- Session
  - SIP
- Presentation
  - XDR
  - ASN.1
  - NDR
  - XML
- Application
  - DNS
  - SMTP
  - HTTP
Session

- RFC 2543: Session Initiation Protocol
- “Text-based protocol for initiating interactive communication sessions between users.”
- e.g. voice, video, chat, interactive games, VR
SIP Issues

- User location
- Call initiation
- User media capabilities
- User availability
- PSTN connectivity
The presentation problem

**Q:** does perfect memory-to-memory copy solve “the communication problem”?

**A:** not always!

```c
struct {
    char code;
    int x;
} test;
test.x = 256;
test.code='a'
```

<table>
<thead>
<tr>
<th>test.code</th>
<th>test.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00000001</td>
</tr>
<tr>
<td></td>
<td>00000011</td>
</tr>
</tbody>
</table>

- **host 1 format**
- **host 2 format**

**problem:** different data format, storage conventions
Presentation

• Problem:
  • Big Endian v. Little Endian
  • ASCII v. EBCDIC v. Unicode
  • 16 bit v. 32 bit v. 64 bit v. ???
  • Floating Point Representation

• Solutions:
  • Canonical Representation
  • Native Format
  • Others?
Canonical Representation

1. Translate local-host format to host-independent format
2. Transmit data in host-independent format
3. Translate host-independent format to remote-host format
External Data Representation

- Used with SunRPC
- Supports C type system (minus pointers)
- Sample from API:
  - `xdr_long(XDR *xdrs, long *lp);`
  - `xdrmem_create(xdrs, addr, sz, op);`
    - `op` determines stream direction
    - `XDR_ENCODE, XDR_DECODE`
ASN.1: Abstract Syntax Notation 1

- ISO standard X.680
  - used extensively in Internet
  - like eating vegetables, knowing this “good for you”!
- defined data types, object constructors
  - like SMI
- BER: Basic Encoding Rules
  - specify how ASN.1-defined data objects to be transmitted
  - each transmitted object has Type, Length, Value (TLV) encoding
TLV Encoding

- **Idea**: transmitted data is self-identifying
  - **T**: data type, one of ASN.1-defined types
  - **L**: length of data in bytes
  - **V**: value of data, encoded according to ASN.1 standard

<table>
<thead>
<tr>
<th>Tag Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boolean</td>
</tr>
<tr>
<td>2</td>
<td>Integer</td>
</tr>
<tr>
<td>3</td>
<td>Bitstring</td>
</tr>
<tr>
<td>4</td>
<td>Octet string</td>
</tr>
<tr>
<td>5</td>
<td>Null</td>
</tr>
<tr>
<td>6</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>9</td>
<td>Real</td>
</tr>
</tbody>
</table>
TLV encoding:

**Value**, 259

**Length**, 2 bytes

**Type**=2, integer

**Value**, 5 octets (chars)

**Length**, 5 bytes

**Type**=4, octet string
Network Data Representation

- Used in DCE
- Receiver-makes-right
- C type system
- First 2 bytes of message specify
  - Integer: 0 = big endian; 1 = little endian
  - Char: 0 = ASCII; 1 = EBCDIC
  - Float: 0 = IEEE 754; 1 = VAX; 2 = Cray; 3 = IBM
- Next two bytes are for future use
Extensible Markup Language

• Being widely used for data exchange
• http://www.xml.org/
• HTML-like
• Entirely text-based
• Self-defining (Really?)
  • Either there is a Document Type Definition
  • Or it must be without ambiguity
DNS: Domain Name System

People: many identifiers:
- SSN, name, Passport #
- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., gaia.cs.umass.edu - used by humans

Domain Name System:

a) distributed database implemented in hierarchy of many name servers

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

Q: map between IP addresses and name?

- 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 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
The DNS Name Space

A portion of the Internet domain name space.
Name Servers

Part of the DNS name space showing the division into zones.
Name Servers (2)

How a resolver looks up a remote name in eight steps.
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:
  a) may not know authoritative name server
  b) may know intermediate name server: who to contact to find authoritative name server
**DNS: iterated queries**

**Recursive query:**
- a) puts burden of name resolution on contacted name server
- b) heavy load?

**Iterated query:**
- a) contacted server replies with name of server to contact
- b) “I don’t know this name, but ask this server”
DNS: caching and updating records

a) once (any) name server learns mapping, it *caches* mapping
   - cache entries timeout (disappear) after some time
b) update/notify mechanisms under design by IETF
   - RFC 2136
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 an alias name for some “cannonical” (the real) name
  - value is cannonical name

- Type=MX
  - value is hostname of mailserver associated with name
## Resource Records

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA</td>
<td>Start of Authority</td>
<td>Parameters for this zone</td>
</tr>
<tr>
<td>A</td>
<td>IP address of a host</td>
<td>32-Bit integer</td>
</tr>
<tr>
<td>MX</td>
<td>Mail exchange</td>
<td>Priority, domain willing to accept e-mail</td>
</tr>
<tr>
<td>NS</td>
<td>Name Server</td>
<td>Name of a server for this domain</td>
</tr>
<tr>
<td>CNAME</td>
<td>Canonical name</td>
<td>Domain name</td>
</tr>
<tr>
<td>PTR</td>
<td>Pointer</td>
<td>Alias for an IP address</td>
</tr>
<tr>
<td>HINFO</td>
<td>Host description</td>
<td>CPU and OS in ASCII</td>
</tr>
<tr>
<td>TXT</td>
<td>Text</td>
<td>Uninterpreted ASCII text</td>
</tr>
</tbody>
</table>

The principal DNS resource records types.
### Resource Records (2)

<table>
<thead>
<tr>
<th>Host</th>
<th>Type</th>
<th>Class</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs.vu.nl</td>
<td>SOA</td>
<td>IN</td>
<td>star boss (952771,7200,7200,2419200,86400)</td>
</tr>
<tr>
<td>cs.vu.nl</td>
<td>TXT</td>
<td>IN</td>
<td>&quot;Divisie Wiskunde en Informatica.&quot;</td>
</tr>
<tr>
<td>cs.vu.nl</td>
<td>TXT</td>
<td>IN</td>
<td>&quot;Vrije Universiteit Amsterdam.&quot;</td>
</tr>
<tr>
<td>cs.vu.nl</td>
<td>MX</td>
<td>IN</td>
<td>1 zephyr.cs.vu.nl</td>
</tr>
<tr>
<td>cs.vu.nl</td>
<td>MX</td>
<td>IN</td>
<td>2 top.cs.vu.nl</td>
</tr>
<tr>
<td>flits.cs.vu.nl</td>
<td>HINFO</td>
<td>IN</td>
<td>Sun Unix</td>
</tr>
<tr>
<td>flits.cs.vu.nl</td>
<td>A</td>
<td>IN</td>
<td>130.37.16.112</td>
</tr>
<tr>
<td>flits.cs.vu.nl</td>
<td>A</td>
<td>IN</td>
<td>192.31.231.165</td>
</tr>
<tr>
<td>flits.cs.vu.nl</td>
<td>MX</td>
<td>IN</td>
<td>1 flits.cs.vu.nl</td>
</tr>
<tr>
<td>flits.cs.vu.nl</td>
<td>MX</td>
<td>IN</td>
<td>2 zephyr.cs.vu.nl</td>
</tr>
<tr>
<td>flits.cs.vu.nl</td>
<td>MX</td>
<td>IN</td>
<td>3 top.cs.vu.nl</td>
</tr>
<tr>
<td><a href="http://www.cs.vu.nl">www.cs.vu.nl</a></td>
<td>CNAME</td>
<td>IN</td>
<td>star.cs.vu.nl</td>
</tr>
<tr>
<td>ftp.cs.vu.nl</td>
<td>CNAME</td>
<td>IN</td>
<td>zephyr.cs.vu.nl</td>
</tr>
<tr>
<td>rowboat</td>
<td>A</td>
<td>IN</td>
<td>130.37.56.201</td>
</tr>
<tr>
<td>little-sister</td>
<td>A</td>
<td>IN</td>
<td>130.37.62.23</td>
</tr>
<tr>
<td>laserjet</td>
<td>A</td>
<td>IN</td>
<td>192.31.231.216</td>
</tr>
</tbody>
</table>

A portion of a possible DNS database for *cs.vu.nl*. 
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

```
+-----------------+---------------------------+
<table>
<thead>
<tr>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of questions</td>
<td>number of answer RRs</td>
</tr>
<tr>
<td>number of authority RRs</td>
<td>number of additional RRs</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>questions</td>
<td>(variable number of questions)</td>
</tr>
<tr>
<td>answers</td>
<td>(variable number of resource records)</td>
</tr>
<tr>
<td>authority</td>
<td>(variable number of resource records)</td>
</tr>
<tr>
<td>additional inform.</td>
<td>(variable number of resource records)</td>
</tr>
</tbody>
</table>
```

12 bytes
DNS protocol, messages

<table>
<thead>
<tr>
<th>identification</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of questions</td>
<td>number of answer RRs</td>
</tr>
<tr>
<td>number of authority RRs</td>
<td>number of additional RRs</td>
</tr>
</tbody>
</table>

- Name, type fields for a query
- RRs in response to query
- Records for authoritative servers
- Additional “helpful” info that may be used

12 bytes
Electronic Mail

- Architecture and Services
- The User Agent
- Message Formats
- Message Transfer
- Final Delivery
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 (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 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)
So I did:

```
[pasward@sandrock pasward] $ telnet www.ynpcsc.gov.cn 25
Trying 202.98.190.193...
Connected to www.ynpcsc.gov.cn.
Escape character is '^[].'
220 www.ynpcsc.gov.cn ESMTP Server (Microsoft Exchange
Internet Mail Service 5.5.2650.21) ready
MAIL FROM: <jqRandom@hotmail.com>
250 OK - mail from <jqRandom@hotmail.com>
RCPT TO: <pasward@big.uwaterloo.ca>
250 OK - Recipient <pasward@big.uwaterloo.ca>
DATA
354 Send data.  End with CRLF.CRLF
yet another open relay
.
250 OK
Quit
221 closing connection
Connection closed by foreign host
```
And this was e-mailed to me:

pasward@big pasward]$ cat /var/spool/mail/pasward
From jqRandom@hotmail.com Wed Mar  6 12:57:00 2002
Return-Path: <jqRandom@hotmail.com>
Received: from www.ynpcsc.gov.cn ([202.98.190.193])
    by big.uwaterloo.ca (8.11.0/8.11.0) with ESMTP id g26HuvY23119
    for <pasward@big.uwaterloo.ca>;
Wed, 6 Mar 2002 12:56:58 -0500
Date: Wed, 6 Mar 2002 12:56:58 -0500
From: jqRandom@hotmail.com
Message-Id: <200203061756.g26HuvY23119@big.uwaterloo.ca>
Received: from sandrock.uwaterloo.ca ([129.97.105.32])
    by www.ynpcsc.gov.cn with SMTP (Microsoft Exchange Internet
    Mail Service Version 5.5.2650.21) id GH8G9YG8;
    Thu, 7 Mar 2002 01:58:55 +0800
yet another open relay
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
Problems with international languages:

- Languages with accents (French, German).
- Languages in non-Latin alphabets (Hebrew, Russian).
- Languages without alphabets (Chinese, Japanese).
- Messages not containing text at all (audio or images).
Message format: multimedia extensions

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

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`

- **Image**
  - example subtypes: `jpeg, gif`

- **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`
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--
(a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent. (b) Reading e-mail when the receiver has a dial-up connection to an ISP.
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**

Example of interaction:

S: +OK POP3 server ready
C: user alice
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
# IMAP

A comparison of POP3 and IMAP.

<table>
<thead>
<tr>
<th>Feature</th>
<th>POP3</th>
<th>IMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where is protocol defined?</td>
<td>RFC 1939</td>
<td>RFC 2060</td>
</tr>
<tr>
<td>Which TCP port is used?</td>
<td>110</td>
<td>143</td>
</tr>
<tr>
<td>Where is e-mail stored?</td>
<td>User's PC</td>
<td>Server</td>
</tr>
<tr>
<td>Where is e-mail read?</td>
<td>Off-line</td>
<td>On-line</td>
</tr>
<tr>
<td>Connect time required?</td>
<td>Little</td>
<td>Much</td>
</tr>
<tr>
<td>Use of server resources?</td>
<td>Minimal</td>
<td>Extensive</td>
</tr>
<tr>
<td>Multiple mailboxes?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Who backs up mailboxes?</td>
<td>User</td>
<td>ISP</td>
</tr>
<tr>
<td>Good for mobile users?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>User control over downloading?</td>
<td>Little</td>
<td>Great</td>
</tr>
<tr>
<td>Partial message downloads?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Are disk quotas a problem?</td>
<td>No</td>
<td>Could be in time</td>
</tr>
<tr>
<td>Simple to implement?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Widespread support?</td>
<td>Yes</td>
<td>Growing</td>
</tr>
</tbody>
</table>
The World Wide Web

- Architectural Overview
- Static Web Documents
- Dynamic Web Documents
- HTTP – The Hyper-Text Transfer Protocol
- Performance Enhancements
Architectural Overview

The parts of the Web model.
The Web: some jargon

- Web page:
  - consists of “objects”
  - addressed by a URL
- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects.
- URL has two components: host name and path name:

- User agent for Web is called a browser:
  - MS Internet Explorer
  - Netscape Communicator
  - Opera
  - Lynx
  - Etc., etc., etc.
- Server for Web is called Web server:
  - Apache (public domain)
  - MS Internet Information Server
URLs – Uniform Resource Locators

<table>
<thead>
<tr>
<th>Name</th>
<th>Used for</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
<td>Hypertext (HTML)</td>
<td><a href="http://www.cs.vu.nl/~ast/">http://www.cs.vu.nl/~ast/</a></td>
</tr>
<tr>
<td>file</td>
<td>Local file</td>
<td>file:///usr/suzanne/prog.c</td>
</tr>
<tr>
<td>news</td>
<td>Newsgroup</td>
<td>news:comp.os.minix</td>
</tr>
<tr>
<td>news</td>
<td>News article</td>
<td>news:<a href="mailto:AA0134223112@cs.utah.edu">AA0134223112@cs.utah.edu</a></td>
</tr>
<tr>
<td>gopher</td>
<td>Gopher</td>
<td>gopher://gopher.tc.umn.edu/11/Libraries</td>
</tr>
<tr>
<td>mailto</td>
<td>Sending e-mail</td>
<td><a href="mailto:JohnUser@acm.org">mailto:JohnUser@acm.org</a></td>
</tr>
<tr>
<td>telnet</td>
<td>Remote login</td>
<td>telnet://www.w3.org:80</td>
</tr>
</tbody>
</table>

Some common URLs.
The Web: the http protocol

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
- 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 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, and 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.)


6. Steps 1-5 repeated for each of 10 jpeg objects
## Non-persistent and persistent connections

**Non-persistent**
- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs to fetch each object
- Each object transfer suffers from slow start

**Persistent**
- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs and less slow start.

But most 1.0 browsers use parallel TCP connections.
http message format: request

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

```
method sp URL sp version cr lf
header field name : value cr lf

Entity Body
```

request line

header lines
HTTP Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request to read a Web page</td>
</tr>
<tr>
<td>HEAD</td>
<td>Request to read a Web page’s header</td>
</tr>
<tr>
<td>PUT</td>
<td>Request to store a Web page</td>
</tr>
<tr>
<td>POST</td>
<td>Append to a named resource (e.g., a Web page)</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove the Web page</td>
</tr>
<tr>
<td>TRACE</td>
<td>Echo the incoming request</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Query certain options</td>
</tr>
</tbody>
</table>

The built-in HTTP request methods.
http message format: response

status line
(protocol
status code
status phrase)

HTTP/1.0 200 OK
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 data data ...

data, e.g., requested html file

header lines
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
HTTP Methods (2)

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Information</td>
<td>100 = server agrees to handle client’s request</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 = request succeeded; 204 = no content present</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>301 = page moved; 304 = cached page still valid</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>403 = forbidden page; 404 = page not found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>500 = internal server error; 503 = try again later</td>
</tr>
</tbody>
</table>

The status code response groups.
HTTP Message Headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Agent</td>
<td>Request</td>
<td>Information about the browser and its platform</td>
</tr>
<tr>
<td>Accept</td>
<td>Request</td>
<td>The type of pages the client can handle</td>
</tr>
<tr>
<td>Accept-Charset</td>
<td>Request</td>
<td>The character sets that are acceptable to the client</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>Request</td>
<td>The page encodings the client can handle</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>Request</td>
<td>The natural languages the client can handle</td>
</tr>
<tr>
<td>Host</td>
<td>Request</td>
<td>The server's DNS name</td>
</tr>
<tr>
<td>Authorization</td>
<td>Request</td>
<td>A list of the client's credentials</td>
</tr>
<tr>
<td>Cookie</td>
<td>Request</td>
<td>Sends a previously set cookie back to the server</td>
</tr>
<tr>
<td>Date</td>
<td>Both</td>
<td>Date and time the message was sent</td>
</tr>
<tr>
<td>Upgrade</td>
<td>Both</td>
<td>The protocol the sender wants to switch to</td>
</tr>
<tr>
<td>Server</td>
<td>Response</td>
<td>Information about the server</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>Response</td>
<td>How the content is encoded (e.g., gzip)</td>
</tr>
<tr>
<td>Content-Language</td>
<td>Response</td>
<td>The natural language used in the page</td>
</tr>
<tr>
<td>Content-Length</td>
<td>Response</td>
<td>The page's length in bytes</td>
</tr>
<tr>
<td>Content-Type</td>
<td>Response</td>
<td>The page's MIME type</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>Response</td>
<td>Time and date the page was last changed</td>
</tr>
<tr>
<td>Location</td>
<td>Response</td>
<td>A command to the client to send its request elsewhere</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>Response</td>
<td>The server will accept byte range requests</td>
</tr>
<tr>
<td>Set-Cookie</td>
<td>Response</td>
<td>The server wants the client to save a cookie</td>
</tr>
</tbody>
</table>

Some HTTP message headers.
Trying out http (client side) for yourself

1. Telnet to your favorite Web 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!
Statelessness and Cookies

<table>
<thead>
<tr>
<th>Domain</th>
<th>Path</th>
<th>Content</th>
<th>Expires</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>toms-casino.com</td>
<td>/</td>
<td>CustomerID=497793521</td>
<td>15-10-02 17:00</td>
<td>Yes</td>
</tr>
<tr>
<td>joes-store.com</td>
<td>/</td>
<td>Cart=1-00501;1-07031;2-13721</td>
<td>11-10-02 14:22</td>
<td>No</td>
</tr>
<tr>
<td>aportal.com</td>
<td>/</td>
<td>Prefs=Stk:SUNW+ORCL;Spt:Jets</td>
<td>31-12-10 23:59</td>
<td>No</td>
</tr>
<tr>
<td>sneaky.com</td>
<td>/</td>
<td>UserID=3627239101</td>
<td>31-12-12 23:59</td>
<td>No</td>
</tr>
</tbody>
</table>

Some examples of cookies.
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

Browser caches name & password so that user does not have to repeatedly enter it.
User-server interaction: cookies

a) server sends “cookie” to client in response mst
   Set-cookie: 1678453

b) client presents cookie in later requests
   cookie: 1678453

c) server matches presented cookie with server-stored info
   – authentication
   – remembering user preferences, previous choices
User-server interaction: conditional GET

a) **Goal**: don’t send object if client has up-to-date stored (cached) version

b) **client**: specify date of cached copy in http request

\[
\text{If-modified-since:} \quad \langle \text{date} \rangle
\]

c) **server**: response contains no object if cached copy up-to-date:

\[
\text{HTTP/1.0 304 Not Modified}
\]

c) **server**: response contains object if cached copy modified:

\[
\text{HTTP/1.1 200 OK}
\]

\[
\langle \text{data} \rangle
\]
Web Caches (proxy server)

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

- user sets browser: Web 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 Web 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
smtp/http: final words

- 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

Comparison with http

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