Traditional Web Apps
Ch. 4 +

Application-Layer Protocols

Must Define:

The syntax and semantics of exchanged messages.

Whether the client or server starts first.

How to handle errors.

How to know when you're done.

Standard or Private

Text-Based Protocols

Many common protocols based on sending text messages.

Built atop the TCP stream protocol.

Lines terminated with `\r\n`.

The line convention essentially breaks the stream up into messages.

HTTP

Simple File Transfer Protocol.

Built on a TCP data stream.

Very simple, given TCP.

Files transferred: Any type.
Most often HTML.

Traditional text-based protocol.
Each line ends with CRLF.

Request/Response

The client sends a request to the server.

The server responds.

The response will contain the requested data,
or an error indication.

Request

A request for the Sandbox home page.

```
GET /index.html HTTP/1.0\r\n
User-Agent: FredView/0.03\r\n
Host: sandbox.mc.edu\r\n
Accept: */*\r\n
Connection: Keep-Alive\r\n
\r\n
```


Response

```
HTTP/1.1 200 OK\r\n
Date: Fri, 07 Jan 2005 22:45:45 GMT\r\n
Server: Apache/2.0.51 (Fedora)\r\n
Last-Modified: Thu, 11 Mar 2004 21:15:14 GMT\r\n
Content-Length: 2000\r\n
Connection: close\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
```

Contents of document

Request and Response Format

Header, blank line, body.

The body may be empty.

Requests start with an operation, file name, and protocol.

Responses start with a protocol, response code, and message.

Following the first line are zero or more headers.
Name: Value format.

Headers describe such things as the client and server, or the length and type of the body.

HTTP Operations

GET	Request a document. The body of the response will contain the document.
HEAD	Request the header for the document. Like a GET, but but the response body will be empty. The main use is to acquire the Last-Modified header to see if a local copy of the document should be refreshed.
POST	Send data to the server. The request body will contain the data. This is usually used to send form data.
PUT	Send data to the server, and store it in the indicated file. It is not often used.

More Operations

There are more operations.

Servers may omit some operations (except GET and HEAD).

Servers may add additional operations.

Servers may *not* redefine operations listed in the standard.

Response Codes

Three digits — first is type

1xx Information; continuing.

2xx Success

3xx Redirection.

4xx Client error

5xx Server error

Example Response Codes

200 Ok

206 Partial Content

301 Moved Permanently

400 Bad Request

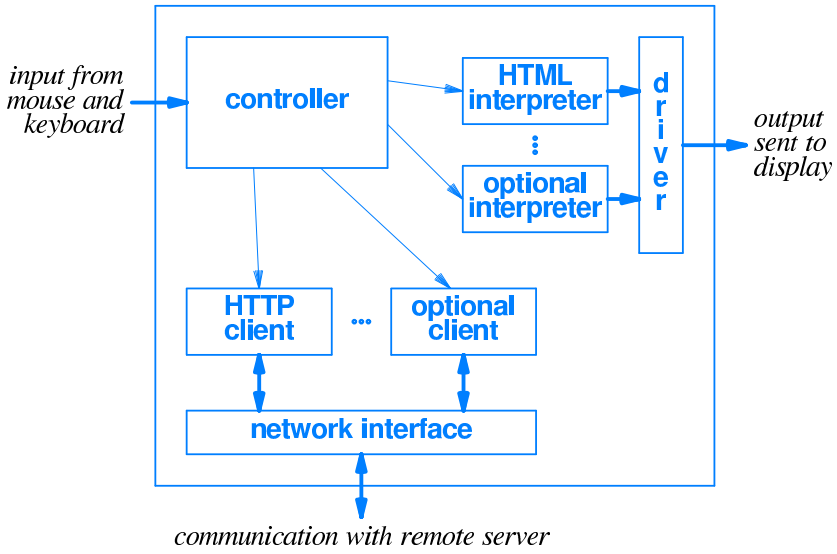
403 Forbidden

404 Not found

500 Internal Server Error

501 Not Implemented.

Browser Architecture



Browser Caching

Browsers keep items in a cache.

Cached pages need not be requested.

Expire after a while.

Servers may request the document not be cached.

Browsers may use the `HEAD` directive to check that a cached copy is up to date.

Versions

HTTP 1.1: Persistent connections.

1.0: each request/response needs its own TCP connection.

1.1: several request/response cycles on one connection.
More efficient.

File Transfer Protocol (FTP)

The FTP protocol is actually older than TCP/IP.
A version existed for the earlier Arpanet.

Protocol does not specify the client interface.

Many command-line FTP tools are based on the BSD tool.

Most commands are never used.

Various file formats are supported.

FTP Protocol

Open a control connection.

Authorize with user and password.

Open a data connection to transfer files.

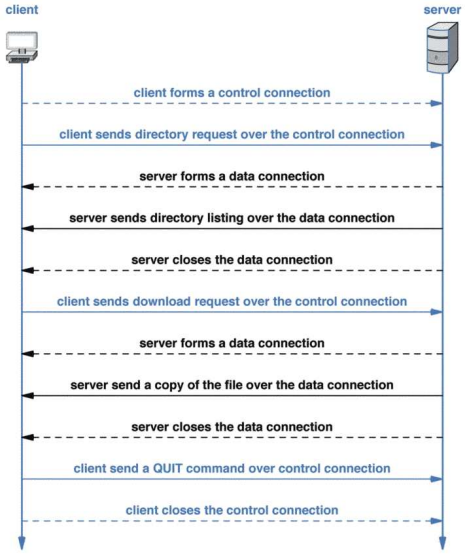
Data is transferred on a second connection.

Default behavior is for the server initiate the connection.

The client/server role is reversed.

Many firewalls will block this connection.

Standard FTP



Passive FTP

In *passive mode* the client makes the second connection.

Part of the FTP protocol for a long time.

Not often used until firewalls became common.

FTP Exchange

The client sends 4-letter commands.

The server responses start with a three-digit status code.
The meaning of each digit is specified in RFC 959.

```
S: 220 Welcome to the FTP Server\r\n
C: USER smith\r\n
S: 331 Please specify password.\r\n
C: PASS This Is The Password\r\n
S: 230 Login successful\r\n
C: SYST\r\n
S: 215 UNIX Type: L8\r\n
```

FTP Exchange Cont.

```
C: TYPE I\r\n
S: 200 Switching to binary mode\r\n
C: PASV\r\n
S: 227 Entering Passive Mode (10,27,0,14,75,41)\r\n
```

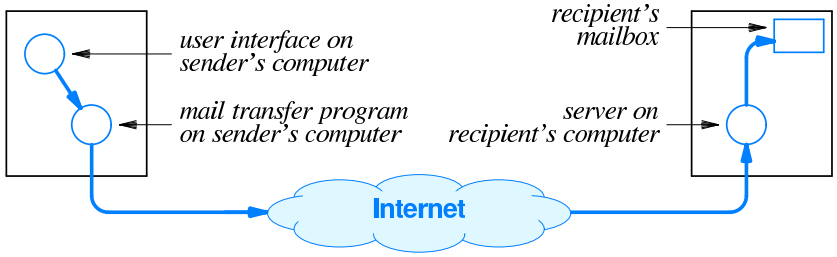
Client connects to the server at port $19241 = 256 \times 75 + 41$

```
C: RETR fredfile\r\n
S: 150 Opening BINARY connection for fredfile\r\n
```

Client reads contents of file on control connection

```
S: 226 File send OK\r\n
C: QUIT\r\n
S: 221 Goodbye\r\n
```


Mail Transfer



Mail is delivered to a mailbox on the receiver's server.

SMTP (RFC 821)

Collecting Your Mail

Originally, users logged into the "servers" to read mail.

Though they probably wouldn't have called them servers.

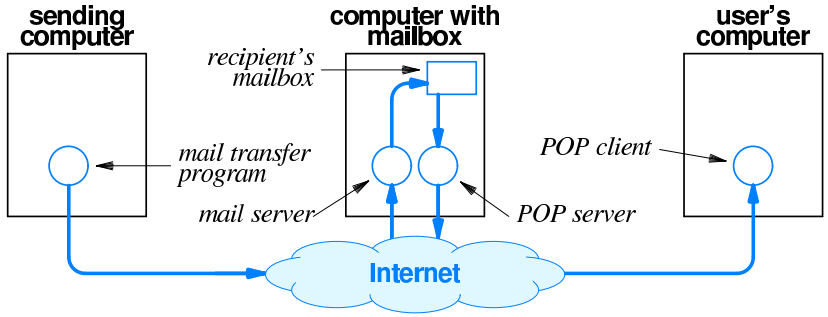
PCs Are Invented

Not suitable to run SMTP servers.

Want a way to read mail from the desktop.

POP and IMAP serve this purpose.

POP and IMAP



Message Format

Series of header lines.

Blank line.

Any text.

```
From: mike@bogus.edu
To: x_w_alice@zml.dingo.com
Date: Tue, 30 Mar 2002 21:45:05 CDT
Subject: Please mount a tape.

Please put a backup tape into
the tape drive.
```


Headers

Other headers:
`Cc:`
`Reply-To:`

Software which does not understand a header simply passes it on.

This allows software to define special headers.

SMTP Interaction

Text-based protocol.

Any server can contact any other.
Essentially a peer-to-peer protocol.

Designed before client-server was current.

Designed before SPAM: Hosts generally trust each other.
Not so much now.

```
R: 220 mail.fred.com SMTP ready\r\n
S: HELO sender.alice.com\r\n
R: 250 OK\r\n
```


SMTP, Cont.

```
S: MAIL FROM:<bill@alice.com>\r\n
R: 250 OK\r\n
S: RCPT TO:<jones@fred.com>\r\n
R: 250 OK\r\n
S: RCPT TO:<william@fred.com>\r\n
R: 550 No such user here\r\n
S: RCPT TO:<sally@fred.com>\r\n
R: 250 OK\r\n
S: DATA\r\n
R: 354 Start mail input; end with <CRLF>.<CRLF>\r\n
S: Blah blah blah...\r\n
S: ...etc. etc. etc.
S: \r\n
R: 250 OK\r\n
```

POP Interaction

```
S: +OK POP3 server ready\r\n
C: USER fred\r\n
S: +OK send password\r\n
C: PASS the password\r\n
S: +OK maildrop locked and ready\r\n
C: LIST\r\n
S: +OK 2 messages (320 octets)\r\n
S: 1 120\r\n
S: 2 200\r\n
S: .\r\n
C: RETR 1\r\n
S: +OK 120 octets\r\n
S: <the POP3 server sends the entire message here>
S: .\r\n
C: DELE 1\r\n
S: +OK message 1 deleted\r\n
```


Computers Aren't Just Text Anymore

These text-based protocols assume messages are ASCII text.

SMTP was defined by RFC 821 in 1982. According to it:
All communication is in ASCII
Messages may not have lines over 1000 characters

Nowadays, we like to send binary attachments.
Programs, Images, Word-processor documents

A binary file has non-ASCII codes, and need not contain a newline every thousand bytes.

Solution: Code the binary data as text.

Anybody else smell a hack here?

Multipurpose Internet Mail Extensions (MIME)

Allows for binary data to be encoded.
Any encoding is allowed
MIME specifies which.

Different portions of the email may be coded differently.

An additional header announces that the message is MIME-encoded.

If the receiver understands MIME, it will see the header and render the message accordingly.

If there is no MIME header, message is plain text.

MIME

```
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="------------010305030303020103000607"
. . .
--------------010305030303020103000607
Content-Type: text/plain; charset=us-ascii;
    format=flowed
Content-Transfer-Encoding: 7bit
. . .
--------------010305030303020103000607
Content-Type: image/jpeg; name="Flowers2.jpg"
Content-Transfer-Encoding: base64
Content-Disposition: inline; filename="Flowers2.jpg"
```

Base 64

A common MIME encoding for arbitrary binary data.

Each three bytes is regrouped into four groups of six bits.

A standard table assigns an ASCII character to each group.

Bytes:	11010101	00000110	11010001	
Redivide:	110101	010000	011011	010001
Assign:	1	Q	b	R

Domain Name System

Each host on the Internet has an IP number.

Those numbers can be hard to remember.
Names are better.

Need a system to map from names to numbers.

A Distributed System is Needed

The Internet is huge.

A centralized name server would have to:

Be Huge

Support tremendous traffic to a single point

Be modified whenever anyone, anywhere, changes a host name

Not practical.

Hierarchical Structure

Names are a series of levels, downward from the right.

`www.mc.edu`

`gatekeeper.dec.com`

`mordred.cs.purdue.edu`

`hsa150.pool020.at101.earthlink.net`

Domains

The top-level domains are defined by ICANN.

Second-level domains are registered by the controller of the top-level domain.

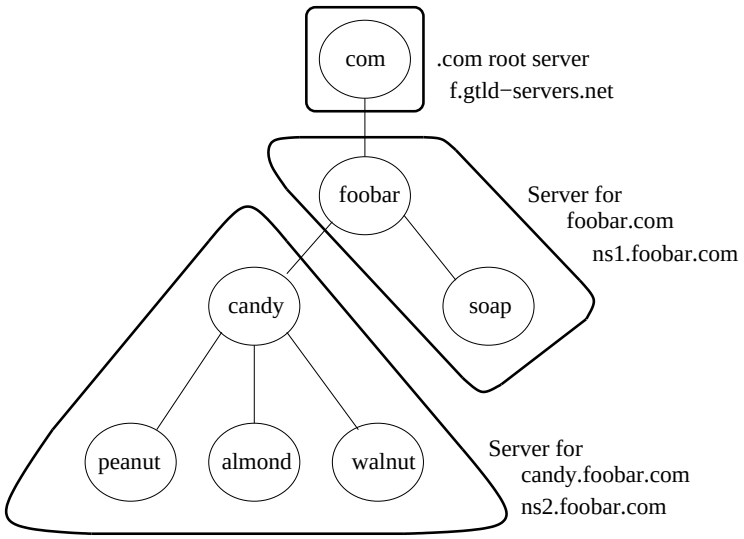
Subdivisions of the second-level domain are up to the owner of the second-level.

A set of servers is associated with each domain at each level.

A single server may handle more than one domain.

Domain holders may run the server themselves or contract out.

Server Assignment



Top-Level Domains

.com .edu .gov .mil

.net .org .arpa .int

Country Codes
.us .ca .uk . . .

New Domains
.aero, .biz, .coop, .info, .museum, .name .pro

Matching a Name

Start with the longest postfix which you know.
May be the empty string.

Send the name to the name server for that domain.
If you know nothing, use a top-level server.

It will respond with as much of the host name as it knows.

If it does not know the entire name, it will tell you what server knows more.
A referral.

Example I

`sandbox.mc.edu` looks up `walnut.candy.foobar.com`

Sandbox sends a request to its name server, `ns.mc.edu`.

`ns.mc.edu` knows nothing of `walnut.candy.foobar.com`.
So it contacts a root server, say `b.root-servers.net`.

The root server responds with a referral to a top-level server that knows about `com`, say `f.gtld-servers.net`.
It actually sends the IP address, of course.

`ns.mc.edu` asks `f.gtld-servers.net` about `walnut.candy.foobar.com`.

Example I, Cont.

`f.gtld-servers.net` responds with a referral to `ns.foobar.com`.

`ns.mc.edu` asks `ns.foobar.com` about `walnut.candy.foobar.com`.

`ns.foobar.com` responds with a referral to `ns.candy.foobar.com`.

`ns.mc.edu` queries `ns.candy.foobar.com`.

It responds with the IP address of `walnut.candy.foobar.com`.

`ns.mc.edu` responds to `sandbox.mc.edu`.

Caching

`sandbox.mc.edu` looks up `almond.candy.foobar.com`

Sandbox sends a request to its name server, `ns.mc.edu`.

`ns.mc.edu` had the good sense to remember the nameserver for `.candy.foobar.com`. It sends its request there.

`ns.candy.foobar.com` replies with the address for `almond.candy.foobar.com`.

`ns.mc.edu` responds to `sandbox.mc.edu`.

Clients may also cache.
A second request for `sandbox.mc.edu` to look up `walnut.candy.foobar.com` should produce no traffic.

Locality

`soap.foobar.com` looks up `walnut.candy.foobar.com`

Soap sends a request to its name server, `ns.foobar.com`.

`ns.foobar.com` knows that `ns.candy.foobar.com` is the name server for `ns.candy.foobar.com`, and queries it.

`ns.candy.foobar.com` returns the IP address of `walnut.candy.foobar.com`.

Locality of Reference.

Keeping the Load Down

Caching and local requests don't query any root servers.

The system would collapse without these.

Authority

Responses contain a time-to-live which indicates how long the information should be kept in cache.

Responses from the server assigned to the domain are *authoritative*; others, such as cached responses, are not.

Recursive and Non-Recursive Operation

A name server may operate recursively or non-recursively.
A client may request recursive operation.
If both agree, the request will be handled recursively.

In recursive mode, the server does not respond with referrals, but follows them itself.

The earlier examples assume the local server does a recursive retrieval and the others do not.
This allows the local server to cache more entries.

Record Types

DNS servers contain several types of records.

A query asks for records of a particular type.

A	Records the IP address for a given name.
PTR	Map an IP address to a host name.
MX	Where to send mail addressed to a host.
CNAME	Name is an alias for another.

Several names may map to the same IP address.

To And Fro

The various DNS record types may be set separately.

Any number of names may map to the same address.
Web hosting providers often use this fact.

A name may map to a number,
but the number not map back.

Load Sharing

A DNS server may respond to a name with various addresses
in rotation.

A name may map to a list of addresses.
The client chooses.

Name Resolvers

Most systems have a library function to resolve names.

Unix: `gethostbyname`.

Plays the client and returns the result.

Resolvers are configured with a list of local name servers.
Unix: `/etc/resolv.conf`

Resolvers typically have a list of default domains to try
so local references work.
We can use "ssh sandbox" on campus.

International Domains

Names must contain large-numbered Unicode characters.

Domain names are stored as ASCII text.

Names are coded as `xn--`α`-`β.

α Plain ASCII characters.
β Specifies the non-ascii characters.

`www.zürich.com`
`www.xn--zrich-kva.com`

Client software must translate.
Otherwise, the user sees the `xn--`α`-`β.

Representation

Encoding is quite complex.

Seems designed to keep the encoded string small
Especially when there are many characters to encode.

The second is a modified base-36 number giving the code and position of each super-ASCII character.

z	r	i	c	h	
0: 80	1: 80	2: 80	3: 80	4: 80	5: 80
6: 81	7: 81	8: 81	9: 81	10: 81	10: 81

...

Not Quite Base 36

Base 36 modified to:

Separate numbers in a list w/o delimiter

Chain efficiently when many non-ASCII characters are present.

Seems designed to keep the coding small.

Host name parts max 63 characters.
Unicode up to 6 hex digits.
Straightforward coding may not allow enough characters.

Standard Hacking

MIME and the international domain representation are both ways to use plain-ASCII infrastructure to support a more flexible representation.

Translation is left to the application.

It's hard to get folks to change existing stuff that works.
Even if it would pay in the long run.

The downside of getting a technology established is that you won't get to fix it later.

Sources

Comer, *Computer Networks and Internets*
(Our beloved textbook.)

RFC 821 (SMTP)

RFC 2616 (HTTP)

RFC 3492 (Punycode)

http://en.wikipedia.org/wiki/Punycode

http://www.motobit.com/util/punycode-decoder-encoder.asp