

today: the World Wide Web: HTTP and HTML

course: Web Technology

Maarten Lamers
www.maartenlamers.com/WT2006/
Leiden University

previous lecture

the World Wide Web (www) is not a network of computers,
but a network of hypertext documents

hypertext is text containing hyperlinks
a hyperlink is a pointer from one document to another

hypertext is written in the *Hypertext Markup Language* (HTML)
www uses hypermedia; not restricted to text documents

previous lecture

the application layer protocol used to request and send
hypertext documents is the *Hypertext Transfer Protocol* (HTTP)

your web-browser is an HTTP client
a web-server is an HTTP server
when they communicate, they "speak HTTP"

where did the WWW come from?

it was invented in 1989 by Tim Berners-Lee at CERN

he created HTTP, URLs, HTML (sort of), and the first web-server
and web-browser as we know it (1990)
in 1991 it was available on the Internet

he is now director of W3C the organization that 'manages' the
World Wide Web, and a researcher at MIT
in 2004 he won the first Millennium Technology Prize and was
made *Sir Tim* by the British Queen

he never made any (commercial) profits from his invention
he considered calling it "The Information Mine" (TIM)
he seems like a nice guy

Uniform Resource Locator (URL)

within the www, all resources are identified by a
Uniform Resource Locator (URL)

it is the address of a resource on the www
usually a webpage, an image, a sound file, ...

it consists of a 'scheme' or 'protocol' to use (usually HTTP), a
hostname, and pathname, e.g. <http://www.liacs.nl/index.html>

sometimes a port number is included, e.g.
<http://www.liacs.nl:80/index.html>

URLs can be relative to the source that they are mentioned in:
a mention of </edu/index.html> in the above document is short
for <http://www.liacs.nl:80/edu/index.html>

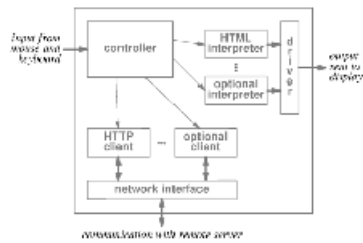
URLs: schemes

for example:

web-resource	http://mediatechnology.liacs.nl
remote login (telnet)	telnet:krypton.wi.leidenuniv.nl
file transfer (ftp)	ftp://ftp.cs.uu.nl/pub/
Usenet newsgroups	news:comp.lang.javascript
e-mail	mailto:maarten@proton.nl
local file	file:c:\maarten\mypage.html

your web-browser often adds the <http://> part
when you type www.maartenlamers.nl in a modern browser,
<http://> is added by default

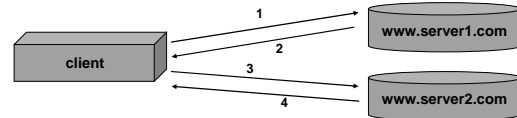
web browser internals



optional interpreter can be: JavaScript interpreter, Flash plugin, XML parser, ...
optional client can be: ftp client, news client, mail client, ...

navigating the web

1. user types a URL into her browser (<http://www.server1.com>), the browser sends request to that machine
2. the server responds with an HTML homepage
3. user clicks hyperlink <http://www.server2.com> on the page, a request is sent
4. the server responds with its HTML homepage



remember!

we are talking about communication at the *Application Layer*
all this relies on having access to a network, through the lower layers in the protocol stack

HTTP runs on top of TCP, which runs on top of IP

HTTP, or the web, is just one way of using the Internet

HTML: Hypertext Markup Language

a language for writing hypertext documents
HTML markup specifies the *structure* of the document not its presentation!

philosophy: the author of an HTML document only determines its structure, the browser determines its presentation

documents can be rendered (made visible) on various devices
e.g. graphical browsers, textual browsers (e.g. Lynx), printers, mobile devices, ...

HTML is strictly defined by www consortium (w3c):
<http://www.w3.org/MarkUp/html-spec/html-pubtext.html>
however, implementations are often of poor quality and may differ from the w3c definition

HTML markup

markup is composed of *tags*, *attributes*, and *entity references*

tags are written `<x>` and `</x>`, where x is the tag name
tags specify a specific markup, e.g. `<P>` for paragraphs, `<H1>` for headings, `<A>` for anchors (hyperlinks)

attributes specify additional parameters to tags, e.g. href for hyperlink target URLs

entity references encode special characters, and are written between `'&'` and `' '` characters, e.g. `<` for character `'<`

for example: `<`

HTML markup example

```
<HTML>
<HEAD>
  <TITLE>Hello World!</TITLE>
</HEAD>
<BODY>
  <H1>Hello World!</H1>
  <P>Here is a paragraph. Would you like to visit my
  <A HREF="http://www.maartenlamers.com">webpage</A>?
  Or would you rather look at my picture?
  <IMG SRC="images/maarten.gif" BORDER="10" ALT="Maarten">
  </P>
</BODY>
</HTML>
```

note that the document does not contain the image, just a reference to it
[see the result here](#)

HTML forms

when surfing the web, you request web-pages from servers
but how do you get information *to* the server?
for example, when you buy a book online

normally data is sent from server to client (browser) for display
sometimes you want to submit information from client to server

one way is through an HTML *form*
provides interactivity between web-user and web application
originally used to provide data to *CGI scripts*

HTML forms example

```
<form action="http://www.liacs.nl/~joostd/form.cgi">
  Your Name: <input type="text" name="surname">
  Male   <input type="radio" name="sex" value="male">
  Female <input type="radio" name="sex" value="female">
  Dutch Citizen: <input type="checkbox" name="dutch">
  Age
    <select name="age">
      <option value="young">0-18
      <option value="middle">18-65
      <option value="old">65+
    </select>
  <input type="submit">
</form>
```

<http://www.liacs.nl/~joostd/WebTech/Day2/form.html> (*)

the evolving web

initially, only very simple client/server architecture

- client sends requests for a document
- server retrieves document from local storage and sends contents to the client
- documents contain static HTML

need for more dynamic content

- server side: SSI (server-side includes), CGI, PHP, ...

need for interaction

- client side: HTML Forms, JavaScript, Java Applets, ActiveX

need for data exchange

- XML, ...

HTTP: Hypertext Transfer Protocol

application layer protocol; basis of World Wide Web
transfers hypertext documents (written in HTML) between
clients and servers
also other type of documents...

HTTP client = browser, *HTTP server* = web-server

HTTP runs on top of TCP/IP

HTTP servers by default run on TCP port 80

one could say that the World Wide Web is *all the resources and users on the Internet that are using the Hypertext Transfer Protocol (HTTP)*

HTTP scenario

typical HTTP scenario:

- client browser connects to server, using TCP
- client sends *HTTP request* to server, using HTTP
- server responds by sending *HTTP response*, using HTTP
- server disconnects the TCP connection

HTTP request methods

GET: retrieve a document

HEAD: retrieve information about the document, not the document itself!

POST: provide information for the server (generally used for fill-in web-forms)

PUT: provide a new or replacement document to be stored on the server

DELETE: remove a document from the server

TRACE asks that proxies declare themselves in the headers, so the client can learn the path that the document took

OPTIONS: what other methods can be used?

HTTP/0.9

HTTP/0.9 is the basic protocol described by Tim B-L in 1991
very basic, only GET request method, nothing else
simply returns the HTML content only

request:

```
GET /-joostd/WebTech/
```

response:

```
<h1>Webtechnologie</h1>
<h2>Inleiding Internet</h2>
<ul>
<li><a href="Day1/internet.htm">Slides</a>
<li><a href="Day1/tutorial.html">Tutorial</a>
</ul>
...
```

HTTP/1.0 request and response structure

an HTTP request contains:

- *request method* (usually GET – retrieve a document)
- a URL, identifying the document to be retrieved
- an HTTP version number: HTTP/1.0
- additional information in *header lines*
- an empty line
- optionally, a *request body* (when request method is POST)

an HTTP response contains:

- an HTTP version number: HTTP/1.0
- a *status code* (e.g. 200), indicating success or failure, and a textual annotation (e.g. "OK")
- additional information in *header lines*
- an empty line
- a *response body*: the data to be retrieved

HTTP/1.1

currently common version
works pretty much the same as HTTP/1.0

most importantly (for this course):
it offers *persistent connections* (HTTP sessions)
discussed later in this lecture

HTTP transaction

request:

```
GET /-joostd/WebTech/ HTTP/1.0
<empty line>
```

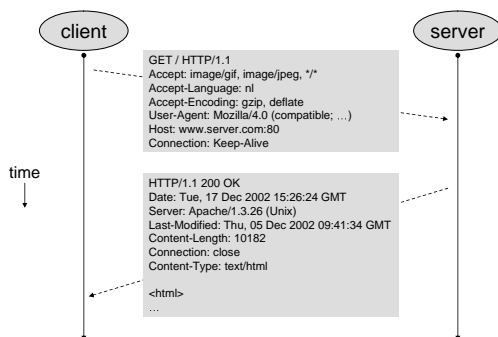
response:

```
HTTP/1.1 200 OK
Date: Wed, 14 Nov 2001 22:45:45 GMT
Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.18
mod_perl/1.23
Last-Modified: Mon, 12 Nov 2001 20:20:14 GMT
ETag: "69f43-148-3bf02efe"
Accept-Ranges: bytes
Content-Length: 328
Connection: close
Content-Type: text/html

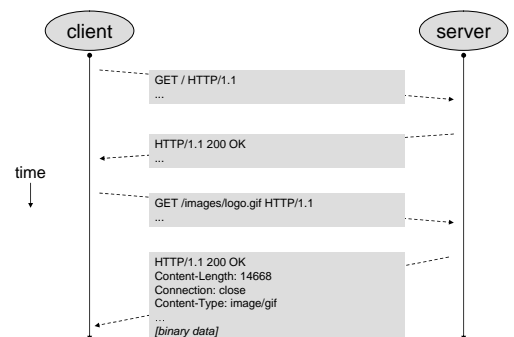
<HTML>
<H1>Webtechnologie</H1>
...
```

} response header
} response body

HTTP transaction



HTTP transaction, document with an image



HTTP status codes

HTTP status codes are organized in ranges

Code	Response Meaning
100-199	informational
200-299	client request successful
300-399	client request redirected, further action necessary
400-499	client request incomplete
500-599	server errors

most famous are: "200 OK", "404 Not found"

the myth about 404 being Tim Berners-Lee's room-number at CERN is not true. There is no room "04" in CERN building "4"

HTTP status codes example

example: response from request <http://msdn.microsoft.com/xml>

```
HTTP/1.1 302 Object Moved
Location: http://cpmsftwbn05/xml/
Server: Microsoft-IIS/5.0
Content-Type: text/html
Content-Length: 146

<head><title>Document Moved</title></head>
<body>
<h1>Object Moved</h1>
This document may be found
<a HREF="http://cpmsftwbn05/xml/">here</a>
</body>
```

MIME: Multipurpose Internet Mail Extensions

originally developed for sending non-text mail attachments

remember, HTTP was originally meant for *hypertext*
MIME is used from HTTP/1.0 to enable sending *hypermedia*
instead of simply *hypertext*
for example, an image file or sound file

based on *mime types*
sent as HTTP response header, e.g. Content-type: text/html

used by server to indicate type of media (associates filename extensions of resources with mime types)
used by client to interpret the response body (or let a *plugin* or external program handle it)

MIME types example

```
text/plain
text/html
text/xml
text/vnd.wap.wml
image/gif
image/jpeg
video/mpeg
audio/wav
application/msword
...
```

see <http://www.iana.org/assignments/media-types/>

HTTP headers

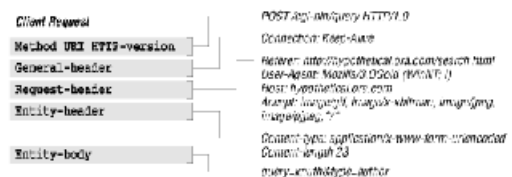
they contain (meta-)information about

- the server
- the client
- the body (which is called *entity* in HTTP)

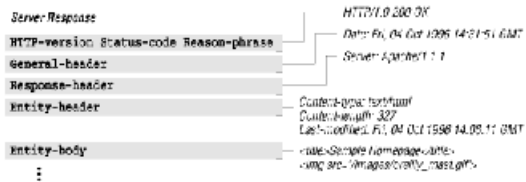
and are used for

- handling multimedia content
- internationalization
- redirection
- logging
- session maintenance
- ...

HTTP: client request structure



HTTP: server response structure



HTTP: general headers

CacheControl: specifies behavior for caching
Connection: indicates whether network connection should close after this connection, or not
Date: specifies the current date
MIME-Version: specifies the version of MIME used in the HTTP transaction
Transfer-Encoding: indicates the type of transformation that was applied to the message body for safe transfer
Upgrade: specifies the preferred communication protocols
Via: used by gateways and proxies to indicate the protocols and hosts that processed the transaction between client and server
...

HTTP: request headers

Accept: specifies which media formats the client accepts
Accept-Language: specifies the language in which the client prefers the data
Cookie: used to convey data stored for the specific server
From: indicates the e-mail address of the user
Host: specifies the host and port number that the client connected to (required for all clients in HTTP/1.1)
If-Modified-Since: requests the document only if newer than the specified date
Referer: specifies the URL of the document that contained the link to this one (i.e. the previous document)
User-Agent: identifies the client program

note how "referer" is misspelled in the protocol!

HTTP: response headers

RetryAfter: specifies either the number of seconds or a date after which the server becomes available again
Server: specifies the name and version number of the server
Set-Cookie: defines a *name=value* pair to be associated with this server
WWW-Authenticate: specifies the authorization type and the realm of the authorization

HTTP: entity headers

Content-Language: specifies the language used in the document (entity) being returned
Content-Length: specifies the length of the entity
Content-Type: specifies the media type of the entity (MIME)
Expires: gives a date and time that the contents may change
Last-Modified: gives the date and time that the entity was last changed
Location: specifies the location of a created or moved document

HTTP: request headers example

example: retrieving a cached document

```
GET / HTTP/1.1
If-Modified-Since: Thu, 03 May 2001 16:01:18 GMT
```

response:

```
HTTP/1.1 304 Not Modified
Date: Wed, 18 Dec 2002 22:17:02 GMT
Server: Apache/2.0.43 (Win32)
Connection: close
ETag: "1f164-5d6-8ba74f80;1f180-9c0-f1868a00"
Content-Location: index.html.en
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Expires: Wed, 18 Dec 2002 22:17:02 GMT
...
```

HTML forms: passing parameters

Q: how can we pass data from a form to a web application?

A: all data entered in the form is collected by the browser and transferred from the client to the server using either:

1. a GET request with the encoded data in the URL *query string* (after a '?' character):

```
GET /cgi-bin/ice.cgi?scoops=2&flavor=cherry HTTP/1.0
```

2. a POST request with the encoded data in the entity body:

```
POST /cgi-bin/ice.cgi HTTP/1.0
Content-type: application/x-www-form-urlencoded
Content-length: 22
```

```
scoops=2&flavor=cherry
```

HTML forms: POST method

HTML forms commonly use the POST method

```
<h1>Ice Cream Stand</h1>
<form
  method="post"
  action="/cgi-bin/ice.cgi"
  enctype="application/x-www-form-urlencoded">
<p>What would you like? <br>
<input type="text" name="scoops" maxlength="1"> <br>
<input type="text" name="flavor"> <br>
<input type="submit" value="Send your order">
</form>
```

HTML forms: URL encoding of data

to prepare data for transfer to a web-app (via a POST or GET request), the web client encodes the data for you.

1. collects all form element *names* and *values*
2. glues each name/value pair together with '='
3. glues resulting pairs together with '&'
4. replaces each space character by '+'
5. replaces all non-alphanumeric (weird) characters by '%xx', where xx is their hexadecimal ASCII code

example: Han la Poutré wants 2 scoops of cherry ice cream

```
name=la+Poutr%É9&scoops=2&flavor=cherry
```

HTTP cookies

an HTTP cookie is a packet of textual information sent by a web server for storage on a client machine, and then sent back by the client each time it accesses that same server

a HTTP server uses the `Set-cookie:` response header field

```
Set-cookie: cart=2242; items=198+234; expire: Wed,...
Set-cookie: sex=male; car=1991-Peugeot-205-1.1
```

when the client requests another page from the same server, the cookie value is returned via the HTTP request's `Cookie:` header field

```
Cookie: cart=2242; items=198+234
Cookie: sex=male; car=1991-Peugeot-205-1.1
```

HTTP cookies and privacy

cookies are good for storing useful information but, they can also be abused to compromise a user's privacy data about the user can be stored for future use, without the user (usually) knowing this for example:

1. when viewing a web-page, images from a third-party server, say www.spam.com, are loaded automatically without the user being aware
2. each resulting HTTP request to www.spam.com includes a cookie, containing information that uniquely identifies the client to the server www.spam.com
3. when visiting other web-pages later, images from the same server www.spam.com may be used to identify you, log your web-surfing habits, ban you, or give you custom adds

HTTP sessions

HTTP is a *stateless* protocol: an HTTP response depends on its corresponding request *only*. Not on previous requests from that same client

after a server sends an HTTP response, the TCP connection is terminated immediately (well, up to HTTP/1.0 at least) on a next request, the client must reestablish a new connection the server has no way of knowing which client made which previous requests

sessions are required to create for instance a shopping kart on an e-commerce website

Q: how can a server relate subsequent requests from a specific client in order to create the notion of a *session*?

HTTP sessions

simulating a *persistent connection* (a.k.a. *session*) requires a mechanism to relate separate client requests by maintaining a *session ID* throughout HTTP communication

three commonly used mechanisms exist:

- *cookies*: include the session ID as a text value in HTTP request and response header fields. The server maintains a list of cookie values to simulate state between multiple HTTP transactions
- *URL rewriting*: each URL linking to the next page is rewritten to include the session ID in the query string (after the '?' character)
www.liacs.nl/hello.index?id=19680128
- *hidden fields*: include a form in each page with a hidden field containing the value of the session ID (user is unaware of the form)

Request for Comments (RFC)

as you must understand by now, standards are very important to organize the Internet, www, e-mail, streaming media, ... standards and other notes regarding the Internet are described in many formal public RFC documents
they are made by the broad Internet community and published by the open *Internet Engineering Task Force* (IETF)

RFC 791	IP
RFC 793	TCP
RFC 1738	Uniform Resource Locators (URLs)
RFC 1812	IPv4 routers
RFC 2045 - 2049	MIME
RFC 2525	TCP problems
RFC 2616	HTTP
RFC 2821	SMTP
... etcetera	