Protocoale de nivel aplicaţie:
HTTP

WWW: concepte

- inventator: Tim Berners-Lee, CERN, 1993
- Pagină Web:
  - formată din “obiecte”
  - adresată de un URL
- Pagina web conţine:
  - pagina HTML de bază
  - alte obiecte referite de către pagina de bază
- URL formată din: host name şi path name:

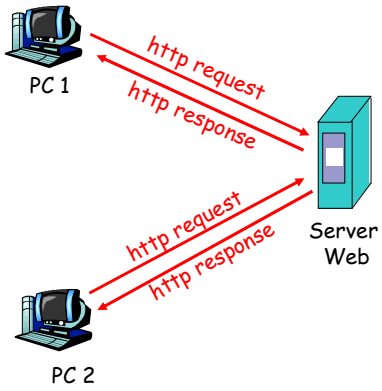
`www.someSchool.edu/someDept/pic.gif`

- clientul (*user agent*) se numeşte *browser*:
  - Internet Explorer
  - Firefox
  - Chrome, etc
- Serverul se numeşte *Web server*:
  - Apache (public domain)
  - MS Internet Information Server

Protocolul HTTP

http: hypertext transfer protocol

- protocol de nivel aplicaţie pt. web
- model client/server
  - *client:* browser care cere, primeşte şi afişează "obiectele" web
  - *server:* Web server trimite "obiectele" ca răspuns la cereri
- http1.0: RFC 1945 (înainte de 1997)
- http1.1: RFC 2068 (după 1998)



Protocolul HTTP

http: bazat pt transport TCP:

- clientul crează un socket TCP către server, pe portul destinaţie 80
- serverul acceptă conexiunea TCP
- mesaje http messages între client şi server
- socketul TCP se închide

http este "stateless" (nu menţine informaţie de stare)

- serverul nu memorează informaţii despre cererile anterioare ale unui client

Protocoalele "stateful" (care memorează informaţia de stare) sînt complexe

- memorie pentru memorarea "istoriei"
- cod suplimentar pentru implementarea "sincronizării" între client şi server dacă conexiunea se întrerupe între client şi server - posibile inconsistenţe

Conexiuni HTTP

HTTP Nonpersistent

- cel mult un obiect transmis pe o conexiune TCP.
- utilizat de HTTP/1.0

HTTP Persistent

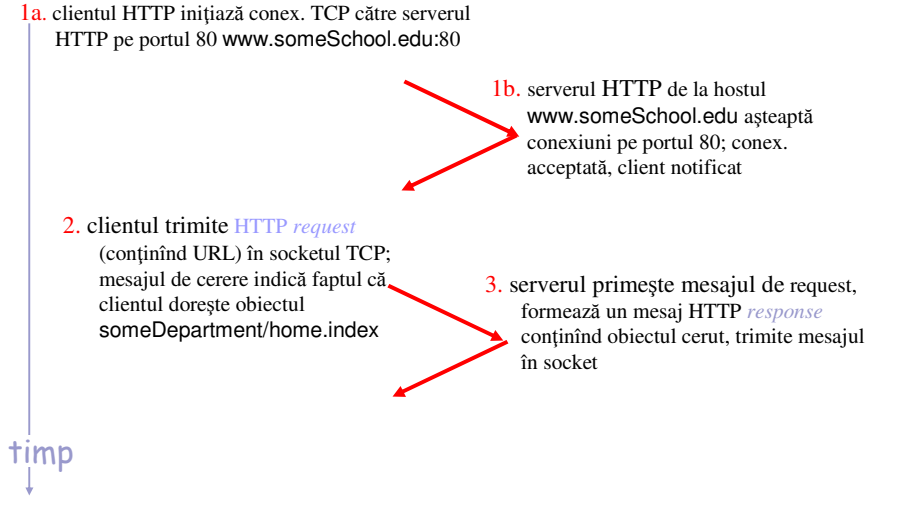
Mai multe obiecte pot fi trimise peste o singură conexiune TCP între client şi server.

- este modul implicit în HTTP/1.1

HTTP Nonpersistent

utilizatorul introduce URL:
`www.someSchool.edu/someDepartment/home.index`

(conţine text şi referinţe către 10 imagini JPEG)



HTTP Nonpersistent (cont.)

4. serverul închide conexiunea TCP.

5. clientul HTTP primeşte mesajul de răspuns care conţine fişierul HTML, afişează HTML; prin parcurgerea mesajului se găsesc 10 referinţe către obiecte JPEG

timp

6. paşii 1-5 se repetă pt. fiecare obiect JPEG

Dezavantaj: multe conexiuni TCP, la fiecare se consumă timp cu stabilirea conexiunii (3-way handshake)
TCP nu este optimizat pentru conexiuni scurte şi numeroase

HTTP Persistent

HTTP Persistent

- conexiunea NU este închisă de către server după ce trimite răspunsul HTTP
- toate mesajele între client şi server se trimit pe aceeaşi conexiune

Persistent fără *pipelining*:

- clientul trimite un request nou numai cînd răspunsul precedent a sosit deja
- un RTT pt fiecare obiect

Persistent cu *pipelining*:

- implicit în HTTP/1.1
- clientul trimite un request pt. fiecare obiect întîlnit
- RTT nu se adună ci se suprapun

Format mesaje HTTP: request

- 2 tipuri de mesaje HTTP: *request*, *response*
- formatul mesajului request (cerere):
  - ASCII (ca şi la alte protocoale studiate)

metoda (GET, POST, HEAD,...)

header

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Host: www.someschool.edu
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

Carriage return, line feed indică sfîrşitul mesajului

(carriage return, line feed suplimentar)

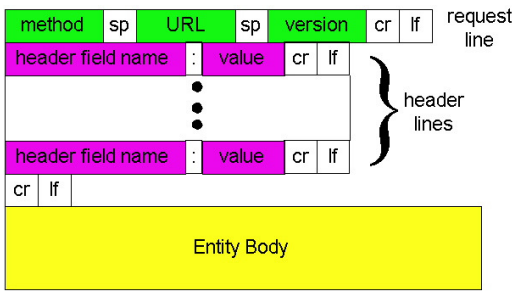
Format mesaje HTTP: request

HTTP/1.0

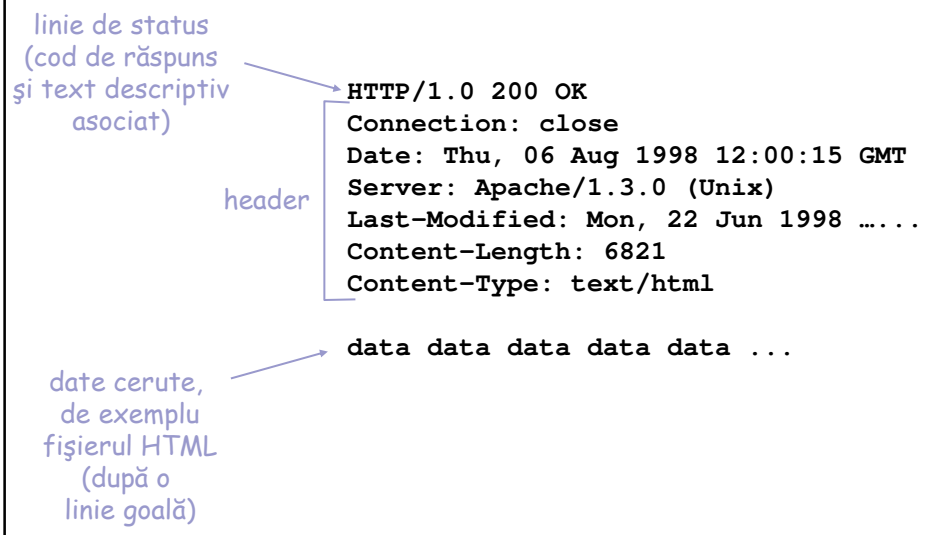
- GET
- POST
- HEAD
  - ca şi GET dar nu întoarce decît headerul, nu şi documentul(entity body gol); util pt verificare

HTTP/1.1

- GET, POST, HEAD
- PUT
  - încarcă (upload) un fişier specificat în "entity body" în calea specificată în URL
- DELETE
  - şterge fişierul specificat în URL



Format mesaje HTTP: response



Coduri de răspuns HTTP

Pe prima linie de răspuns

Exemple:

200 OK

- request a avut succes, răspunsul va fi după header

301 Moved Permanently

- obiectul cerut a fost mutat, noua locaţie specificată mai jos în mesaj (Location:)

400 Bad Request

- mesajul de request nu este înţeles de server

404 Not Found

- documentul/obiectul nu a fost găsit

505 HTTP Version Not Supported

CGI: Cum transmitem către server date utilizator

Pe pagina de web pot exista formulare sau în general cîmpuri în care utilizatorul să introducă date

Aceste date for fi transmise unui program (script *CGI*) lansat de către server
În *environment*-ul programului, serverul va seta unele variabile relevante:

- QUERY_STRING, REMOTE_HOST, CONTENT_TYPE, CONTENT_LENGTH…

Metoda GET (bazată pe URL):

- pentru volum redus de date
- input-ul este trimis în cîmpul URL din linia de request, după semnul ?
- datele se separă cu & :
  `www.somesite.com/animalsearch?monkeys&banana`
- serverul pasează acest input în variabila QUERY_STRING

Metoda POST:

- pentru volume mai mari de date
- input-ul este trimis către server în "entity body"
- serverul pasează acest input în *stdin*-ul programului
- lungimea input-ului în CONTENT_LENGTH

Testare HTTP din telnet

1. faceţi telnet către Web server:

```
telnet matrix.elcom.pub.ro 80
```

deschide conexiunea TCP port 80 (implicit pt HTTP). Tot ce se introduce în continuare va fi trimis pe acea conexiune

2. introduceţi un mesaj GET:

```
GET / HTTP/1.0<ENTER>
<ENTER>
```

se introduce ENTER de 2 ori căci linia goală de după header e obligatorie

3. ce mesaj întoarce serverul ?

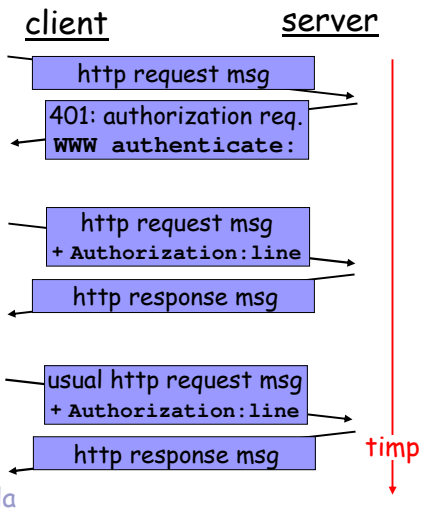
Identificarea userului

- Serverul este "stateless"
- Cum se identifică un user
  - Autentificare
  - Cookies
- Caching
  - GET Conditional

Autentificarea userului pe server

Scop: acces limitat la anumite documente

- stateless: clientul trebuie să se autentifice la fiecare mesaj request
- authorization: linie care conţine user şi password
  - **`authorization:`** linie de header în request



Browser-ul memorează numele şi parola a.î. userul să nu le introducă decît prima dată

Cookies: memorarea "stării"

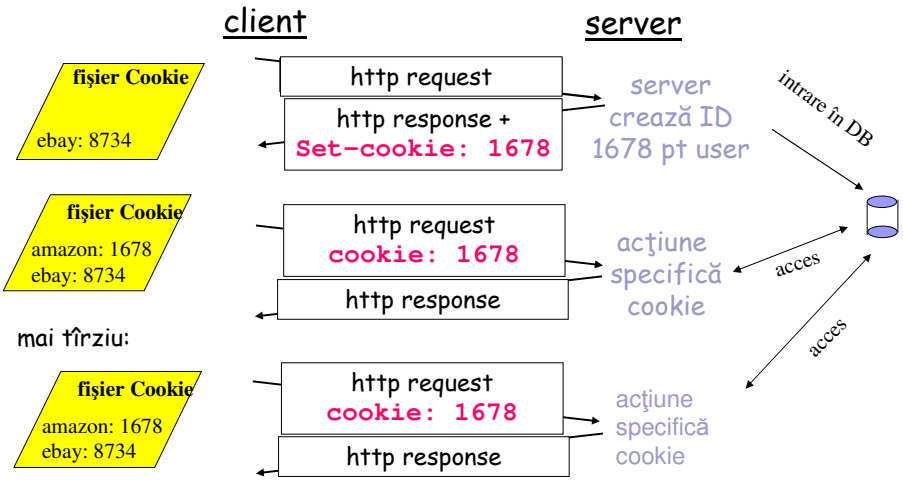
HTTP este în continuare stateless

cookies au 4 componente:

1) *cookie header line* în HTTP response
2) *cookie header line* în HTTP request
3) fişierul cookie este păstrat pe host-ul userului şi este administrat de browser
4) pe server se menţine o bază de date de cookies pt. a memora informaţii despre utilizatori

Primul request care ajunge pe server de la un user necunoscut determină serverul să creeze un ID nou şi să-l salveze în baza de date

Cookies: memorarea "stării"



Cookies (cont.)

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

Ce permit cookies:

- conturi utilizator
- coşuri de cumpărături (*shopping cart)*
- portale web
- reclame specifice pt utilizator
  - colectează informaţii despre obiceiurile userului
- de aceea browserul poate fi setat să le refuze

GET Conditional : client-side caching

Scop: obiectul nu va fi retrimis de către server dacă clientul are versiunea actuală în cache

1) Primul request

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

2) Primul răspuns al serverului

```
HTTP/1.1 200 OK
Date: Mon, 7 Jul 2003 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 2 Jul 2003 09:23:24
Content-Type: image/gif

(data data data …)
```

3) Clientul salvează în cache ultima dată/oră de modificare a obiectului (2 iulie 2003, 9:23:24)

4) request ulterior din cache

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 2 Jul 2003 09:23:24
```

5) răspuns ulterior al serverului

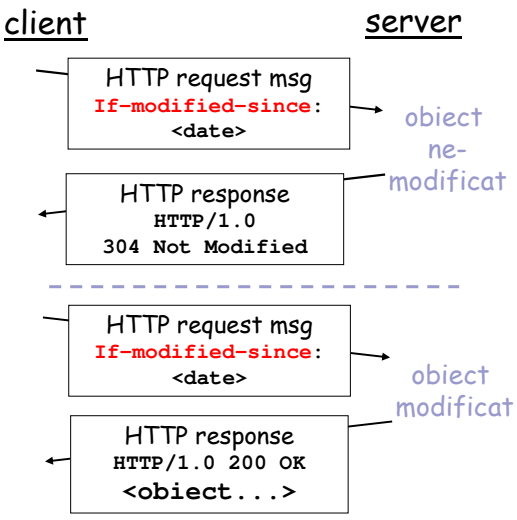
```
HTTP/1.1 304 Not Modified
Date: Mon, 14 Jul 2003 15:39:29
Server: Apache/1.3.0 (Unix)

(entity body gol …)
```

6) clientul afişează informaţia din cache

5-6') obiect modificat: serverul răspunde normal, clientul afişează normal

GET Conditional : client-side caching

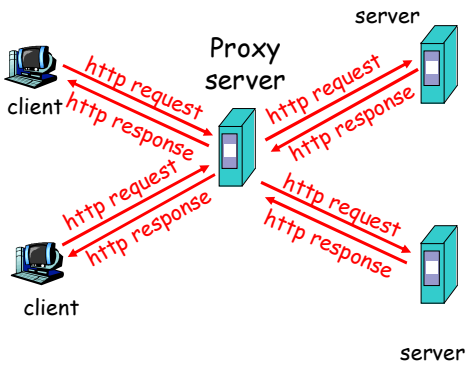


Exemplu de schimb de mesaje

Web Caches (proxy server)

Scop: proxy-ul se interpune între client (browser) şi server, serveşte din cache paginile disponibile, fără a mai accesa serverul. Cache-ul se realizează pe proxy, în plus faţă de cache-ul normal de pe client.

- userul configurează manual adresa serverului proxy (de ex: www.proxy.pub.ro)
- clientul trimite automat toate *http requests* către proxy
  - dacă obiectul există în cache, proxy-ul trimite obiectul în mesajul http response
  - dacă nu, proxy-ul cere obiectul, îl memorează în cache şi îl întoarce şi userului
  - alt user care cere acelaşi obiect îl va lua din cache
- avantaj: mulţi useri într-o locaţie (instituţie) care accesează cam aceleaşi servere - se reduce traficul extern de la acea locaţie
- alt avantaj: configurînd un proxy la o adr. IP, se poate accesa un site care este blocat pt. clienţii de la alte adrese IP



HTTPS

- Versiune sigură a HTTP
- HTTPS = HTTP + TLS → **vezi slide-uri SSL, TLS**
- Procedură:
  - Clientul se conectează
  - Clientul şi serverul fac schimb de chei publice
  - Clientul şi serverul negociază un algoritm rapid de criptare simetrică (e.g. 3DES)
  - Serverul se autentifică folosind un certificat
  - Clientul îl acceptă sau rejectează
  - (Clientul prezintă un certificat - opţional)
  - etc

Bibliografie

- RFC1945: HTTP 1.0
- RFC2068: HTTP 1.1