

HTTP Protocol, Proxy, and COSC 6377 Term Project Tutorial

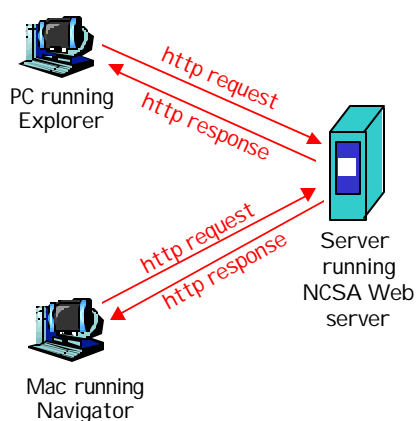
T. Mark Huang

<http://www.cs.uh.edu/~jsteach/cosc6377/>

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, May 1996
- http1.1: RFC 2068, Jan. 1997



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

URL (Uniform Resource Locator)

has three parts: **scheme**, **host name (w/port)**, and **path name**:

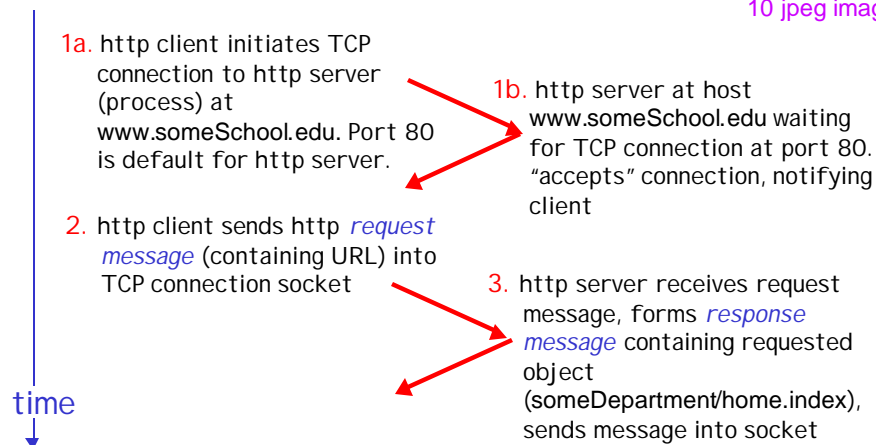
http://www.someSchool.edu:port/someDept/pic.gif

http example

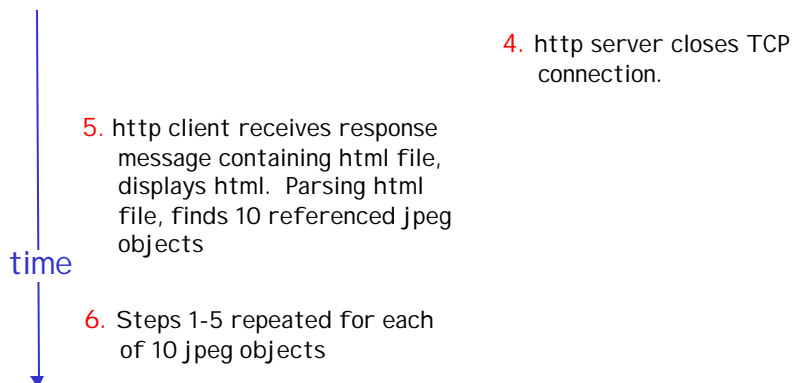
Suppose user enters URL

http://www.someSchool.edu/someDepartment/index.html

(contains text,
references to
10 jpeg images)



http example (cont.)



Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs (round-trip time) to fetch each object
- Each object transfer suffers from slow start

But most 1.0 browsers use parallel TCP connections.

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.

http message format: request

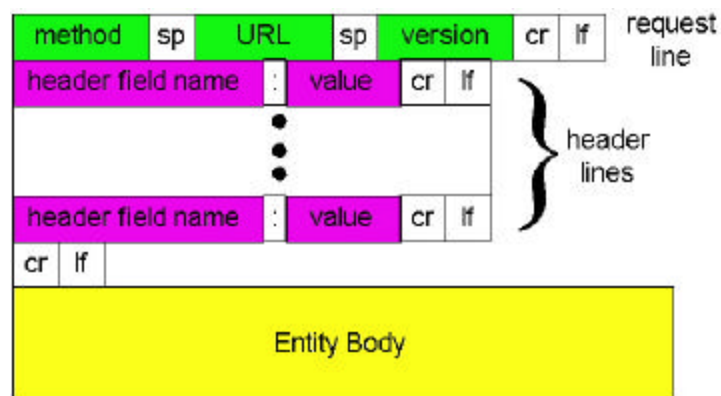
- two types of http messages: *request, response*
- **http request message:**
 - ASCII (human-readable format)

request line
(GET, POST, HEAD commands) → `GET http://www.uh.edu/index.html HTTP/1.0`

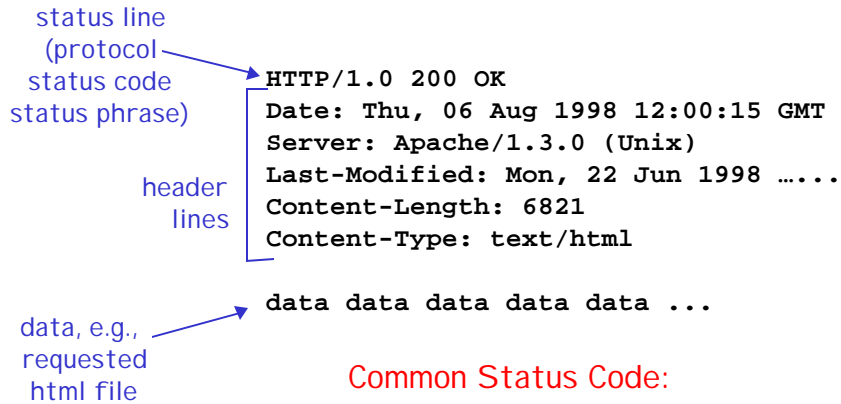
header lines → `User-agent: Mozilla/4.0`
`Accept: text/html, image/gif, image/jpeg`
`Accept-language: fr`

Carriage return, line feed → (extra carriage return, line feed)
 indicates end of message

http request message: general format



http message format: response

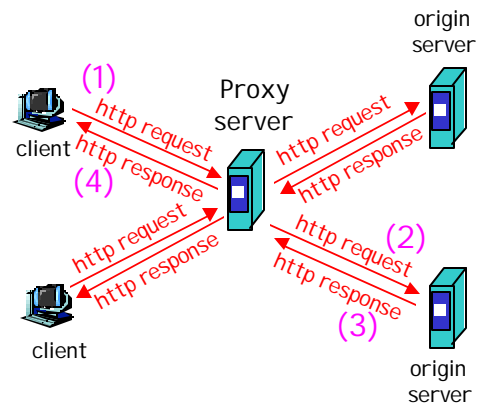


Common Status Code:

- 200 OK
- 301 Moved Permanently
- 400 Bad Request
- 403 Forbidden
- 404 Not Found

HTTP Proxy

- A HTTP proxy server deals with HTTP servers (web servers) on behalf of HTTP clients (browsers).
- Proxy clients talk to proxy servers, which relay approved client requests on to real servers, and relay answers back to clients.



COSC 6377 Term Project

Midget Service Corp (MSC) HTTP Proxy Server

- Processing HTTP/1.0
- Configurable - configuration file (proxyrc)


```
Port 500                                #proxy_port
refuse pegasus.cs.uh.edu                #proxy_client
block www.microsoft.com                 #proxy_location
redirect www.netscape.com/(.*)\
    "www3.netscape.com/\1"              #proxy_location
rewrite User-Agent "Netscape (.*)" "Mozilla \1" #proxy_hfield
filter text/(.*) sed s/ok/good/g        #proxy_filter
```
- Processing Request - GET / POST / QUIT
- Processing Reply - media filter
- Multiple connections - fork() or select()

Configuration Library

proxy.h

- int proxy_init(*proxy_t* p, const char* rcfile*)
- int proxy_port(*proxy_t* p*)
- int proxy_client(*proxy_t* p, const char* host*)
- int proxy_location(*proxy_t* p, const char* loc, char* buf, size_t bufsiz*)
- int proxy_hfield(*proxy_t* p, const char* key, const char* value, char* buf, size_t bufsiz*)
- int proxy_filter(*proxy_t* p, const char* media, char* buf, size_t bufsiz*)

fork() v.s. select()

- to handle multiple connections
- `fork()`: start a new process to handle each request. Easy to implement, but not scale well and require interlocking for shared resource.
- `select()`: one process handles all requests. Scale well and do not require interlocking for shared resource, but more complex to implement (may need to build state diagram).

Sample programs

Located in ~jsteach/www/cosc6377/proxy/skel/

- `sample1.c` - clear channel echo TCP client
- `sample2.c` & `sample2-sig.c` - clear channel echo TCP server

For above *client* and *server* programs, *client* sends whatever input from console to *server*, and *server* sends whatever input from console to *client*. You can use *client* to mimic browser and *server* to mimic web server, and your proxy runs in-between them.

- `pipe.cc` - pipe(2) example
- `http.cc` - proxy filter library sample

Assignment

- **Due midnight, Tu., Nov. 21st, 2000**
- **Start immediately!!**
- 20% of your final grade
- Group project: 1 or 2 people
- Submission: detail will be posted on the web
- Turn in all required files (*.c, Makefile, & Readme)
- **Read FAQ** in the project web page
- Questions: send to leila_n@cs.uh.edu or post in the newsgroup
- MOSS: Measure Of Software Similarity (UC Berkeley)