Plan

- Application layer
  - Principles of application layer protocols
  - **The Web and http**
  - File transfer: FTP
  - Electronic mail in the Internet
  - DNS — The Internet's directory service
  - Socket programming with TCP and UDP
  - Content distribution

Application layer/The Web and HTTP

The **Hyper-Text Transfer Protocol** (**http**) is the Web application-layer protocol.

HTTP is implemented in two programs: the client and the server, running on different and connected hosts, and exchanging HTTP messages.

A **Web page** (or **document**) consists of **objects**. An object is simply a file (e.g. an HTML file, a JPEG image, a Java applet, an audio file etc.) which is addressable by a single **Universal Resource Locator** (**URL**).

For instance, if a web page contains a **base html file** and five JPEG images, then the Web page contains six objects.

Application layer/The Web and HTTP (cont)

An URL is made of

- a protocol name,
- a Web server name,
- a file path on the server.

For instance `http://www.ietf.org/rfc/rfc2396.txt` specifies

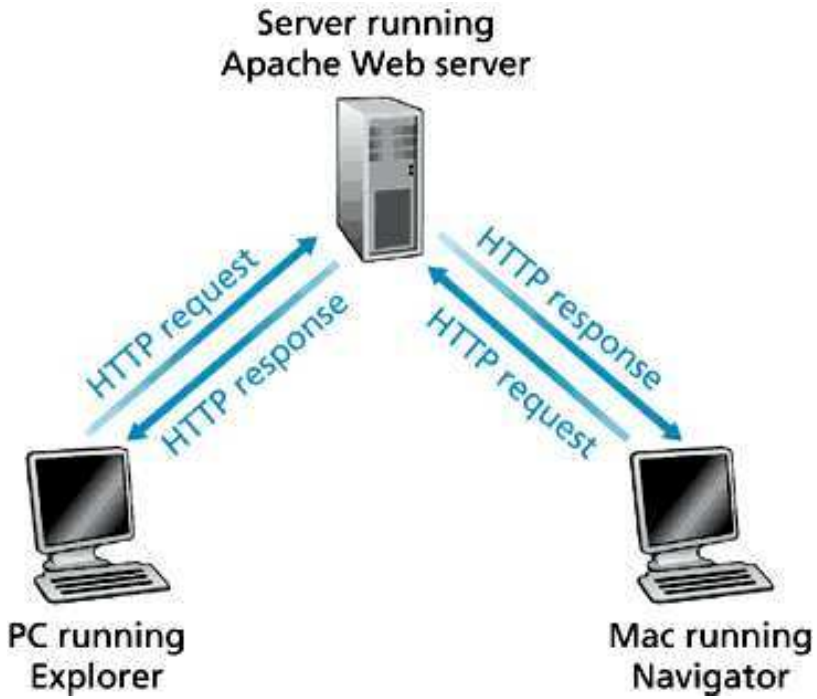
- the use of the HTTP,
- a unique computer named `www.ietf.org`,
- and the location of a text file or page to be accessed on that computer whose pathname is `/rfc/rfc2396.txt`.

Application layer/The Web and HTTP (cont)

A Web server houses web objects, each addressable by a URL. Popular Web servers include Apache.] When a user clicks on an **hyperlink**, its browser sends HTTP request messages for the objects in the referred page to the server, which in turn responds with HTTP response messages that contain the objects.

Because it relies on TCP, HTTP does neither worry about data loss nor ordering.

Application layer/The Web and HTTP (cont)



Application layer/The Web and HTTP (cont)

It is important to understand that the server sends requested objects to the client without storing any state information about the client.

For instance, if the same client asks for the same file two times in a short lapse, the server never responds that it has just send it *because it does not remember*.

That is why HTTP is said to be a **stateless protocol**.

There are two kinds of TCP connections: **persistent** and **non-persistent**.

The Web and HTTP/Non-persistent connections

Suppose a client uses a non-persistent connection to query a page made of a base HTML file and ten JPEG images, all objects being stored on the same server. The URL for the base HTML file is `http://www.school.org/dep/index.html`

- The HTTP client initiates a TCP connection to the server `www.school.org` on port number 80, which is the default port number for HTTP.
- The HTTP client send an HTTP request message to the server via the socket associated with the TCP connection that was established in step 128. This message includes the path `/dep/index.html`.

The Web and HTTP/Non-persistent connections

- The HTTP server receives the request via the socket associated with the connection, retrieves the object `/dep/index.html` from its storage (RAM or disk), encapsulates the object in an HTTP response message and sends it to the client via the same socket.
- The HTTP server tells TCP to close the connection (but the TCP server waits for the client acknowledgement).
- The HTTP client receives the response message. The TCP connection terminates. The message indicates that the embedded object is an HTML file. The client extracts this file, parses the file and find references to the ten JPEG objects.
- The first four steps are repeated for the ten JPEG images.

The Web and HTTP/Non-persistent connections (cont)

The steps described use **non-persistent connections** because each TCP connection is closed after the server sends the object — it does not persist for other objects.

Thus, in our example, 11 TCP connections are generated.

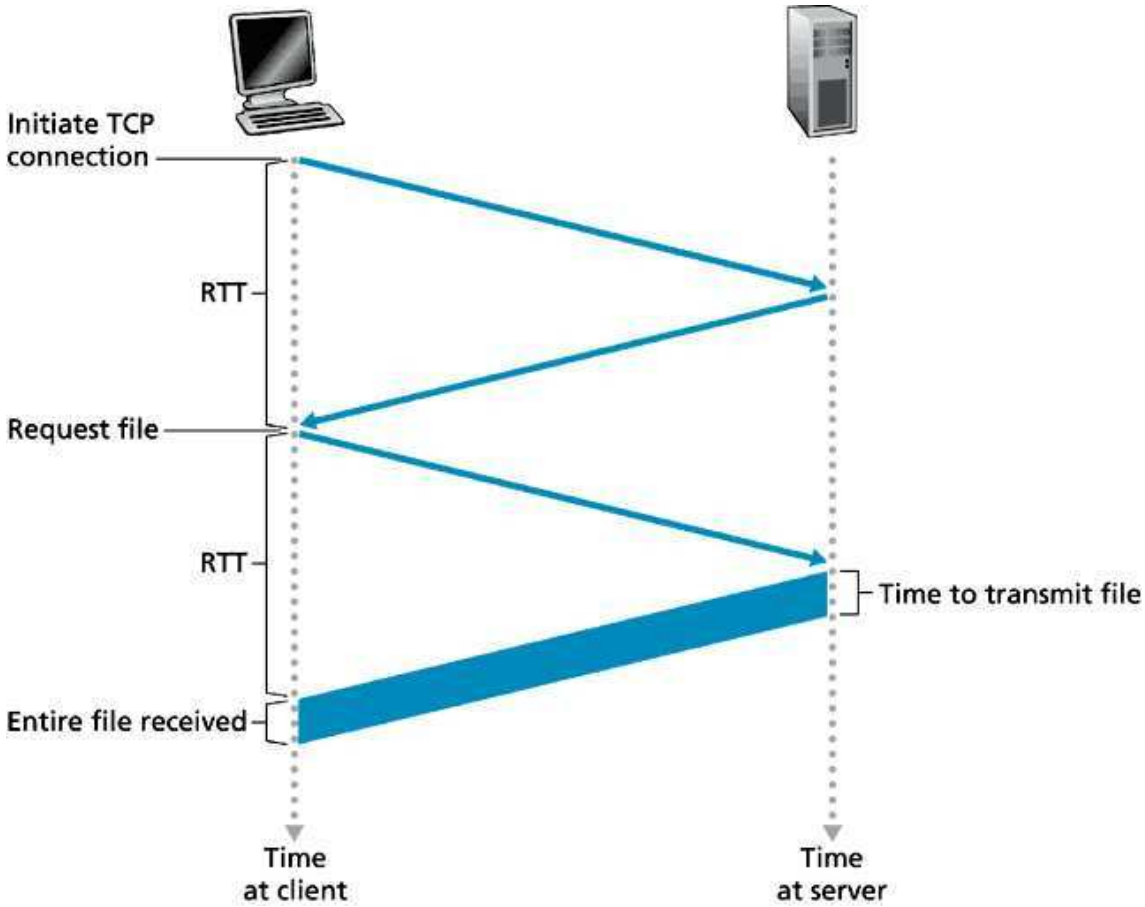
However, it is possible that the browser actually opens several TCP connections in parallel, but these connections still are non-persistent.

The Web and HTTP/Non-persistent connections (cont)

Let us estimate the amount of time that elapses between the moment the client requests a base HTML file and the time it is received entirely.

The **round-trip time** (**RTT**) is the time needed for a small packet to travel from the client to the server and back to the client. It is roughly two RTTs plus the transmission time.

The Web and HTTP/Non-persistent connections (cont)



The Web and HTTP/Persistent connections

Non-persistent connections allow only one object to be transmitted over a given TCP connection, suffering two RTT delays.

With **persistent connections** the server let open the connection after sending a response. For instance, the eleven previous objects could have been sent during a single TCP persistent connection.

A persistent connection is usually closed after a configurable time interval. There are two kinds of persistent connections:

- **without pipelining**: the response must be received before another request is made;
- **with pipelining**: multiple requests can be gathered into the same TCP segment, so the server can fulfill them in parallel (leading to less Internet traffic).

The Web and HTTP/HTTP requests

As expected, there are two kinds of HTTP messages: requests and responses.

For example:

```
GET /dep/index.html HTTP/1.1
Host: www.school.org
Connection: close
User-agent: Mozilla/4.0
Accept-language:fr
```

A request is written in ordinary ASCII text (encoding characters with 7 bits).

The Web and HTTP/HTTP requests (cont)

This message is made of five lines, but can have less or more lines in general.

The first line `GET /dep/index.html HTTP/1.1` is called the **request line** and the following are the **header lines**.

The request line has three fields:

- the method field (`GET`),
- the URL field (`/dep/index.html`),
- the version field (`HTTP/1.1`).

The method field can take several values, `GET`, `POST`, `HEAD` etc. The `GET` method is used to request an object.

The Web and HTTP/HTTP requests (cont)

The header line `Host: www.school.org` specifies the host on which the object resides.

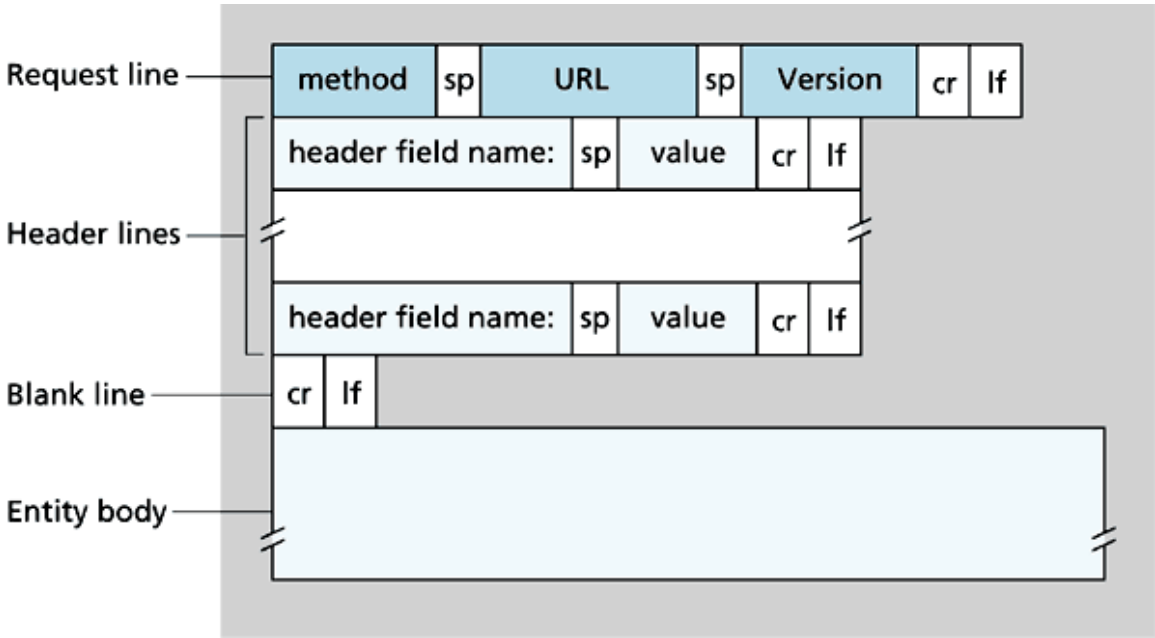
By including the line `Connection: close` the browser tells the server it does not want a persistent connection (thus it should be closed after the object has been sent).

Line `User-agent: Mozilla/4.0` specifies the browser type and version, a Mozilla user agent.

Finally, `Accept-language:fr` indicates that the user prefers to receive a French version of the object, if available, otherwise a default version.

The Web and HTTP/HTTP requests (cont))

Let us take a look to the general shape of an HTTP request:



There is another field in the message called **body**. It is empty when sending a `GET` request method but is filled when using `POST`. This method is used to request a page selected using some specific information (in the body field), as with a form.

The Web and HTTP/HTTP requests (cont)

In fact, not all forms use a `POST` method: sometimes the user's information is put in the URL itself and a `GET` method is used instead. For instance, if a form consists in two fields whose given values are `monkeys` and `bananas`, the following URL is possible:

```
www.some-site.com/search?monkeys&bananas
```

The `HEAD` method is similar to a `GET` one, except the server does not return the requested object — even if it sends back a message. This method is often used by application developers for debugging purposes. Protocol HTTP version 1.1 allows a `PUT` method to upload objects to a web server and a `DELETE` method to delete an object in a web server.

The Web and HTTP/HTTP responses

This could be a response message to the request we presented:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

... data ... data ... data ...
```

The Web and HTTP/HTTP responses (cont)

The first line `HTTP/1.1 200 OK` is the **status line**. It tells here that the object was found and is returned. The number 200 is the status number (equivalent to the status name).

The following six lines are the **header lines**.

The first one, `Connection: close` acknowledges the non-persistent underlying TCP connection.

The `Date` line gives the time when the server retrieved the object (not when the object was created).

The `Server` line states what kind of web server is running on what kind of operating system (here, an Apache server on Unix). It is analogous to the `User-Agent` line in the request.

The Web and HTTP/HTTP responses (cont)

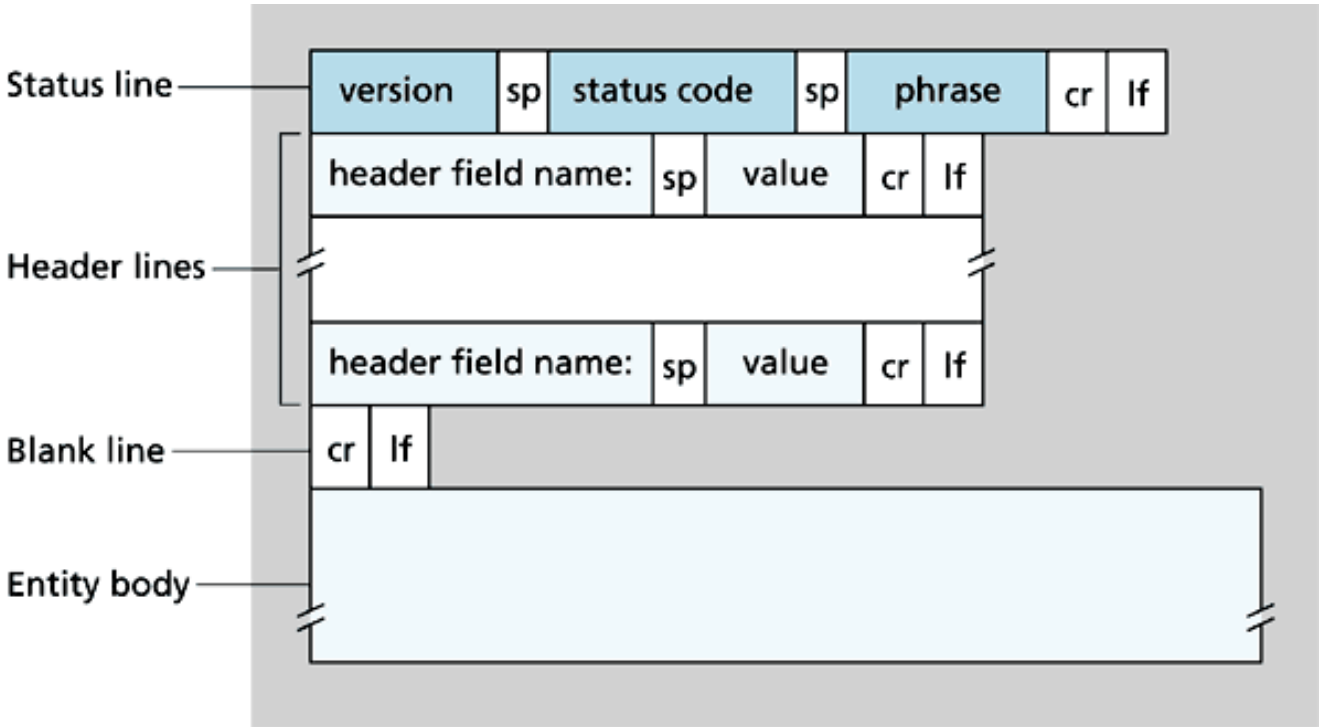
The `Last-Modified` line indicates the date when the object was created or last modified. The `Last-Modified` line is very important for object caching, both on the client side (browser cache) and on the network side (proxy servers).

The `Content-Length` line gives the number of bytes the object is made.

The `Content-Type` line records the kind of the object, in this case a HTML file (the file extension, like `.html` or `.htm` does not tell officially the content type).

The Web and HTTP/HTTP message format/Responses (cont)

The general structure of an HTTP response message is



The Web and HTTP/HTTP responses (cont)

The status code and its associated phrase can be of several kinds:

- `200 OK` means that the request succeeded;
- `301 Moved Permanently` means that the requested object has been permanently moved; the new URL is given in a `Location` header line in the response message and the client browser will automatically retrieve the new URL;
- `400 Bad Request` means that the request was not understood by the server;
- `404 Not Found` means that the object was not found on the server;
- `505 HTTP Version Not Supported` is self explanatory.

The Web and HTTP/HTTP responses (cont)

It is recommended that you create some small HTTP request messages, send them to some web server and examine the response messages.

If you get access to Unix machine (or Windows running Cygwin[1]), just type in a terminal

```
$ telnet konkuk.ac.kr 80
Trying 194.254.134.22...
Connected to konkuk.ac.kr
Escape character is '^]'.
HEAD /~rinderkn/index.html HTTP/1.0
```

(press the return key twice) and you will see the response from the server.

[1] http://www.cygwin.com

The Web and HTTP/HTTP responses (cont)

```
HTTP/1.1 200 OK
Date: Mon, 02 May 2005 05:25:56 GMT
Server: Apache/1.3.27 (Unix) mod_jk PHP/4.3.10
X-Powered-By: PHP/4.3.10
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

Try to request a non-existent object, like `foo.html` and check the response again.

The Web and HTTP/Authorisation and cookies

We mentioned before that an HTTP server is stateless. This keeps the design of the server simple.

But sometimes it is desirable to identify users, for instance in order to restrict the access or to save personal informations or to provide specific services to subscribers only.

Authorisation
Many web sites require users to provide a name and a password in order to grant the access to the services and objects. Let us follow the steps of such a process.

The Web and HTTP/Authorisation and cookies (cont)

First the user sends an ordinary request. The server responds with a message with an empty body an with a `401 Authorization Required` status code. Also it includes a `WWW-Authenticate` header line telling the client how to authenticate (here: username and password).

The browser receives that response message and prompts the user for a name and a password. The client then resends the request with the name and the password.

After getting the first object, the client has to resend the username and password, but the user does not need to re-enter these data: the browser keeps them in a cache memory.

Cookies

Many commercial sites make use of **cookies**. These contain the following components:

- a cookie header line in the HTTP response;
- a cookie header line in the HTTP request;
- a cookie file kept on the user's system and managed by the browser;
- a database at the server site.

The Web and HTTP/Authorisation and cookies (cont)

Let us describe an example of session using cookies.

- A user connects for the first time to an e-commerce site using cookies.
- The web server creates a unique identification number for him and create an entry in its customer database indexed by this number (or **cookie**).
- The server responds to the client browser including a `Set-cookie` header line containing the cookie.
- The browser reads the `Set-cookie` line and appends at the end of a special file a line containing the web site name together with the cookie.

The Web and HTTP/Authorisation and cookies (cont)

- Now, each time the user requests a page on the server, the browser sends the cookie along as a `Cookie` header line. This way the server knows exactly the visited pages and the order and time of visit.
  This allows the site to provide a "shopping card" service: during a single visit, the server keeps track of all the intended purchases, so that the user pays for all of them once, at the end of the session.
- The customer pays by giving his name, address and bank card number. The site now associates the cookie (and all the details of the visit) to the personal data about the customer.

The Web and HTTP/Authorisation and cookies (cont)

- After shopping, if the customer returns to the site, his browser will put again the cookie in the requests. Thus the e-commerce site can propose new products based on the previous visit and he can even by with one click since the site recorded his personal banking information during last session.

Now we understand how can cookies enable a stateful session between the client and the server on top of a stateless HTTP session.

The Web and HTTP/Authorisation and cookies (cont)

Privacy

As you also understand, cookies are a controversial feature because they can provide a lot of private information to commercial web sites.

These sites can even record the behaviour of a specific user *across several web sites*.

Many pages of commercial site request advertisement banners (GIFs or JPEG images) from an advertising company. These requests may contain a cookie, and if the user visits several sites which share the same advertisement company, these company can track the user behaviour across all the sites he visited.

You can visit `http://www.cookiecentral.com/` for news about the cookie controversy.

The Web and HTTP/Conditional GET

By caching the previously retrieved objects, the traffic on the internet is decreased: in case the client requests several times the same object, the copy in the cache memory is delivered instead of relaying the request again to the server.

This caching can take place either at the client side (i.e. in the browser) or at a special node in the network.

But caching can create a new problem: the cached objects can be *stale*, i.e. the original object (at the server side) may have changed since the last request.

The mechanism in HTTP which ensures that all retrieved objects are up-to-date despite caching is called **conditional** GET.

The Web and HTTP/Conditional GET (cont)

An HTTP request message is a conditional GET if and only if

- the request message uses the GET method,
- the request message includes an If-Modified-since header line.

Let us consider an example.

First, a browser requests an *uncached object* from some web server:

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
```

The Web and HTTP/Conditional GET (cont)

Second, the web server sends a response message with the object:

```
HTTP/1.0 200 OK
Date: Wed, 12 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24
Content-Type: image/gif

... data ... data ... data ...
```

The client displays the object and stores it in its local cache *along with the last modification date*.

The Web and HTTP/Conditional GET (cont)

One week later, the client requests the same object, but this may have changed in the server. In order to get an up-to-date version the user-agent issues a conditional GET:

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
If-modified-since: Mon, 22 Jun 1998 09:23:24
```

Note that the value of the `If-modified-since` header line is exactly the same as the one in the previous (and first, here) request. This conditional GET instructs the server to send back the requested object if and only if the object has been modified since the specified date.

The Web and HTTP/Conditional GET (cont)

Suppose the object has not been modified in the meanwhile.

Then, fourth, the server sends back he following response message:

```
HTTP/1.0 304 Not Modified
Date: Wed, 19 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
```

The body is empty: the object has not been sent (again). Note also the status line `Not Modified`.

As a final note on HTTP, let us mention that this protocol can be used without any browser or human user and also can carry objects that are not part of web pages, such as XML files.