CSE2029: Data Communication & Computer Networks

Lecture-6: Web and HTTP Cont’d…

Faculty: Dr. Sandeep Kumar

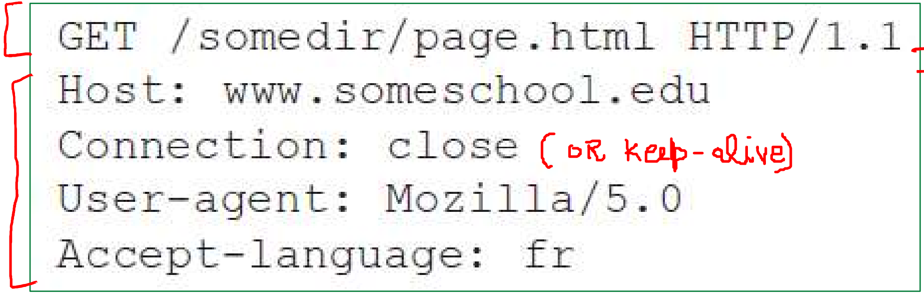
Outline

- *HTTP Message Format: HTTP Request Message*
- *HTTP Message Format: HTTP RESPONSE Message*
- *User-Server Interaction: Cookies*
- *Web Caching*

HTTP Message Format: HTTP Request Message

- There are two types of HTTP messages, request messages and response messages, both of which are discussed as follows:

HTTP Request Message:



```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

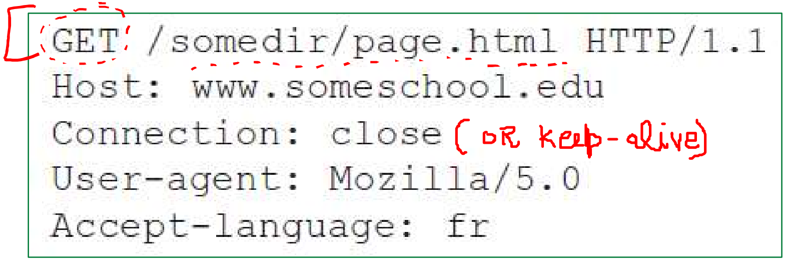
(OR keep-alive)

request line

header lines

- The first line of an HTTP request message is called the **request line**; the subsequent lines are called the **header lines**.

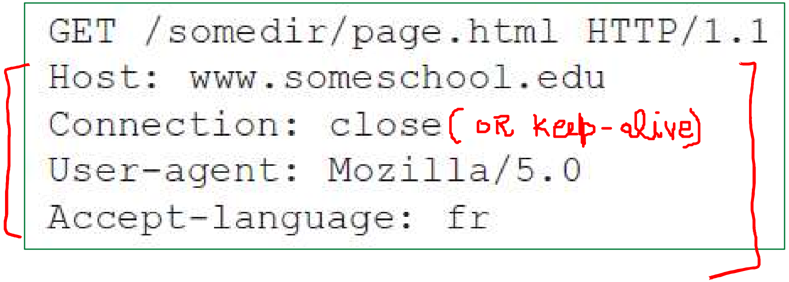
HTTP Message Format: HTTP Request Message



```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close (OR keep-alive)
User-agent: Mozilla/5.0
Accept-language: fr
```

- The first line of an HTTP request message is called the **request line**; the subsequent lines are called the **header lines**.
- The **request line** has three fields: ***method field***, ***URL field***, *and* ***HTTP version field***.
  - The **method field** can take on several different values, including **GET, POST, HEAD, PUT, and DELETE**. The **GET method** is used when the browser requests an object from the server.
  - The requested object is identified in the **URL field**. In this example, the browser is requesting the object "/somedir/page.html".
  - The **HTTP version field** is self-explanatory; in this example, the browser implements version HTTP/1.1.

HTTP Message Format: HTTP Request Message



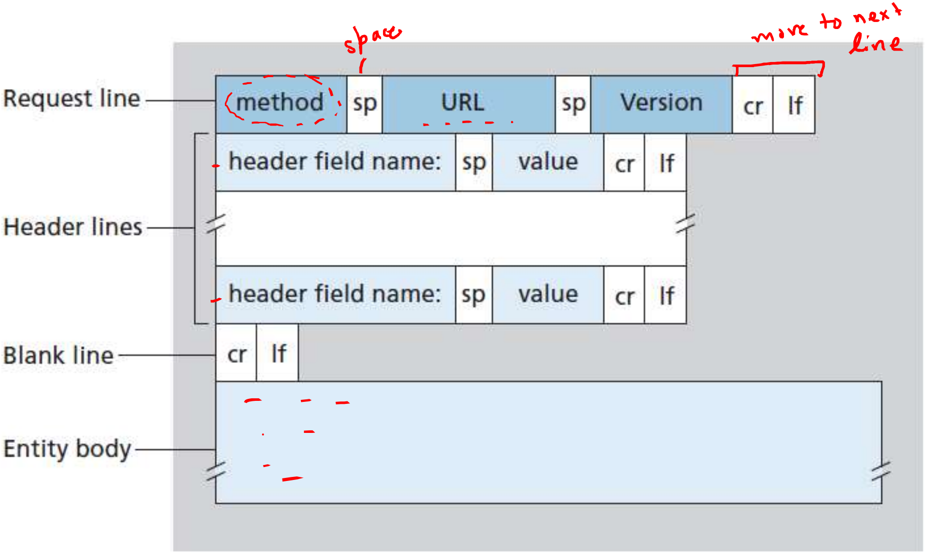
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close (OR keep-alive)
User-agent: Mozilla/5.0
Accept-language: fr
```

- Now let’s look at the **header lines** in the example.
- The header line ***Host: www.someschool.edu*** specifies the host on which the object resides.
- By including the ***Connection: close in*** header line, browser is telling the server that it doesn’t want to bother with persistent connections; it wants the server to close the connection after sending the requested object. **Connection: keep-alive** indicates that the client wants to keep the connection open or alive after sending the response message.
- The ***User-agent:*** header line specifies the user agent (the browser type) that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.
- The ***Accept-language***: header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

HTTP Message Format: HTTP Request Message

- Having looked at an example, let's now look at **the general format of a request message**, as shown in Figure:

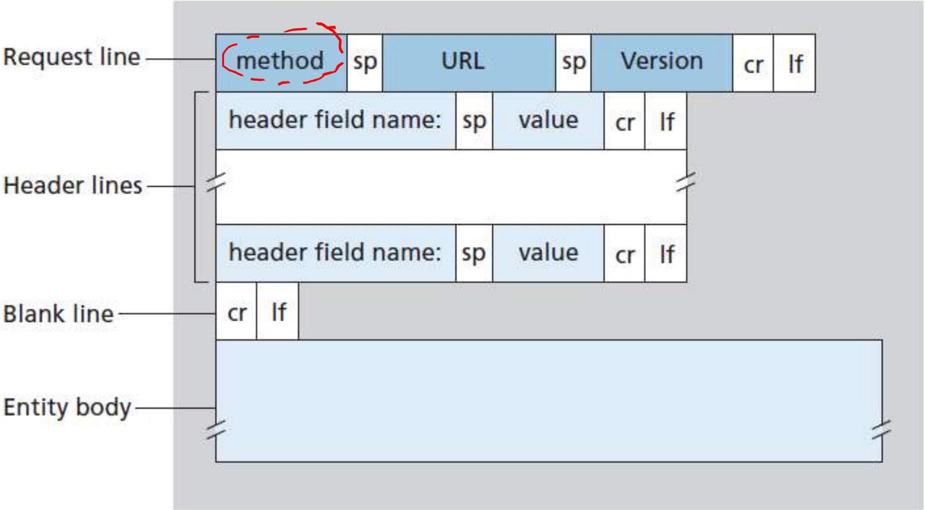
- We see that the general format closely follows our earlier example.
- We may have noticed, however, that after the header lines (and the additional carriage return and line feed) there is an "entity body."
- The entity body is empty with the **GET method**, but is used (will not be empty) with the **POST method**.
- An HTTP client often uses the **POST method** when the user fills out a form—for example, when a user provides search words to a search engine.



♦ General format of an HTTP request message

HTTP Message Format: HTTP Request Message

- An HTTP client often uses the **POST method** when the user fills out a form—for example, when a user provides search words to a search engine. (e.g. Google)
- With a **POST message**, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields.
- If the value of the method field is POST, then the entity body contains what the user entered into the form fields.



Request line — method | sp | URL | sp | Version | cr | lf

Header lines — header field name: | sp | value | cr | lf

header field name: | sp | value | cr | lf

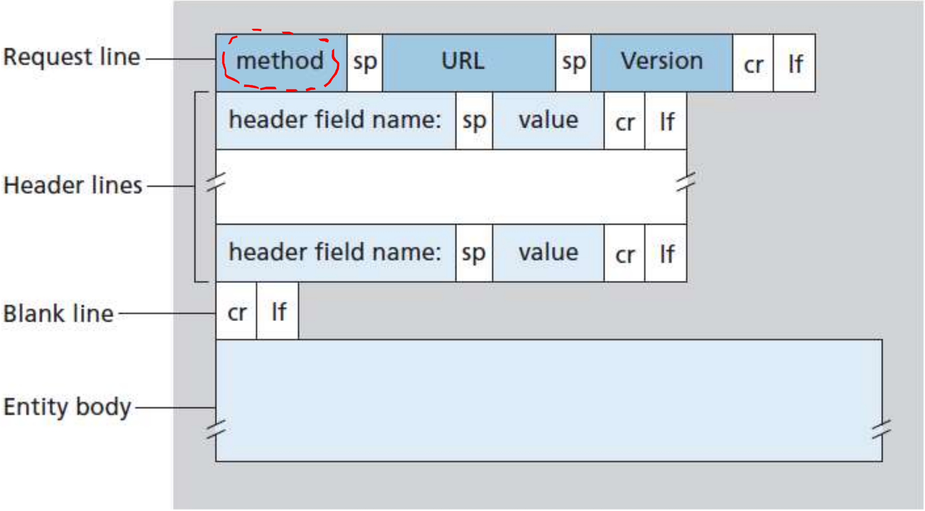
Blank line — cr | lf

Entity body

♦ General format of an HTTP request message

HTTP Message Format: HTTP Request Message

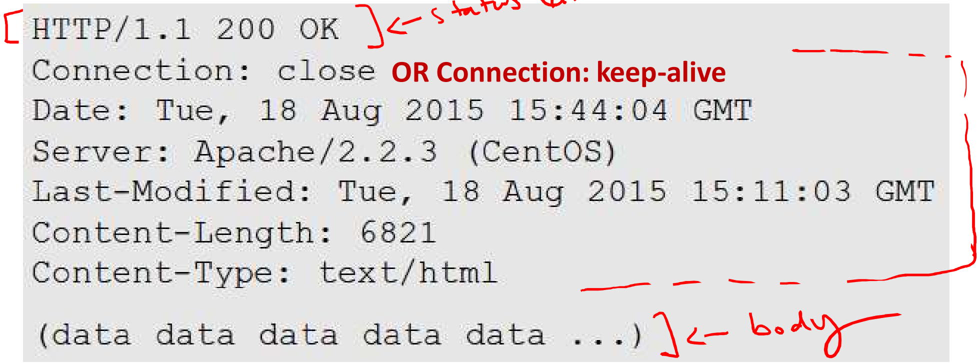
- The **HEAD method** is similar to the GET method. The HEAD method is used to ask only information about a document, not the document itself. For example, if a URL might produce a large download, a HEAD request could read (tell) its Content-Length to check the file size without actually downloading the file.
- The **PUT method** is often used in conjunction with Web publishing tools. It allows a user to upload an object to a specific path (directory) on a specific Web server.
- The **DELETE method** allows a user, or an application, to delete an object on a Web server.



♦ General format of an HTTP request message

HTTP Message Format: HTTP RESPONSE Message

- A typical response message is presented hereunder:



```
HTTP/1.1 200 OK
Connection: close OR Connection: keep-alive
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

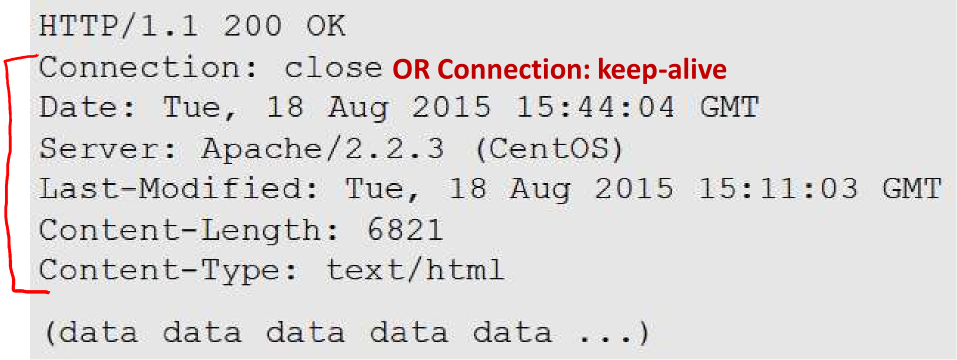
<- status line

Header lines

<- body

- It has three sections: a **status line**, **six header lines**, and then the entity **body**.
- The **body** contains the requested object itself (represented by data data data data data ...).
- The **status line** has three fields: the **protocol version field**, a **status code**, and a corresponding **status message**. In this example, the status line indicates that the protocol version is HTTP/1.1, the status code is 200 and the status message is OK.

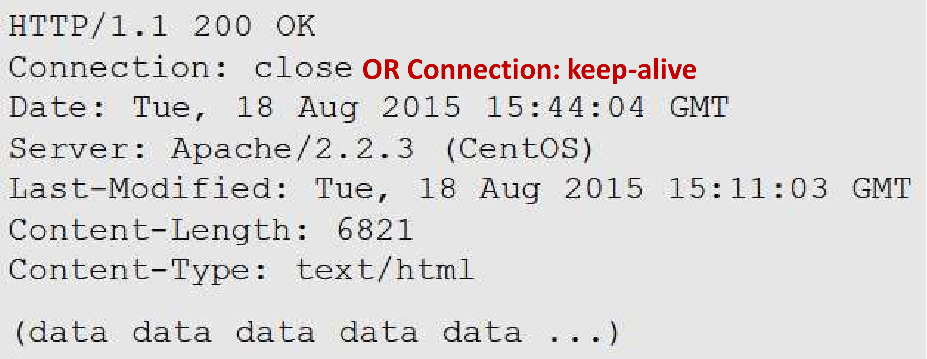
HTTP Message Format: HTTP RESPONSE Message



```
HTTP/1.1 200 OK
Connection: close OR Connection: keep-alive
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

- Looking at the **Header Lines**:
- The server uses the “**Connection: close”** to tell the client that it is going to close the TCP connection after sending the message. However, if “**Connection: keep-alive**” is used the connection is not closed, but is instead kept open. When the client sends another request, it uses the same connection.
- The “**Date:”** header line indicates the time and date when the HTTP response was created and sent by the server.

HTTP Message Format: HTTP RESPONSE Message



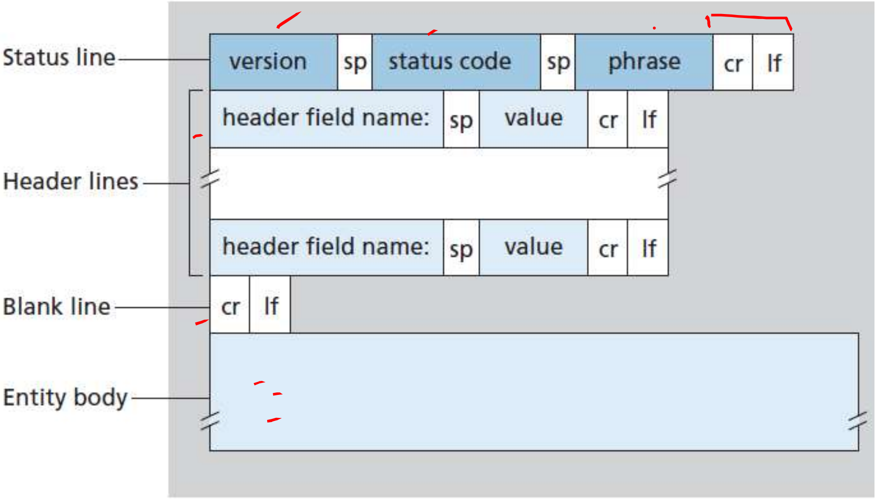
```
HTTP/1.1 200 OK
Connection: close OR Connection: keep-alive
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

- The “**Server:**” header line indicates that the message was generated by an Apache Web server.
- The “**Last-Modified:**” header line indicates the time and date when the object was created or last modified.
- The “**Content-Length:**” header line indicates the number of bytes in the object being sent.
- The “**Content-Type:**” header line indicates that the object in the entity body is HTML text.

HTTP Message Format: HTTP RESPONSE Message

- Having looked at an example, let’s now look at **general format of a RESPONSE message**, as shown in Figure:

- In the status line, the **status code** and associated **phrase** indicate the result of the request. Some common status codes and associated phrases are: **200 OK**: Request succeeded and the information is returned in the response. **400 Bad Request**: This is a generic error code indicating that the request could not be understood by the server. **404 Not Found**: The requested document does not exist on this server. **505 HTTP Version Not Supported**: The requested HTTP protocol version is not supported by the server.
- The header lines and entity body are same as discussed previously.

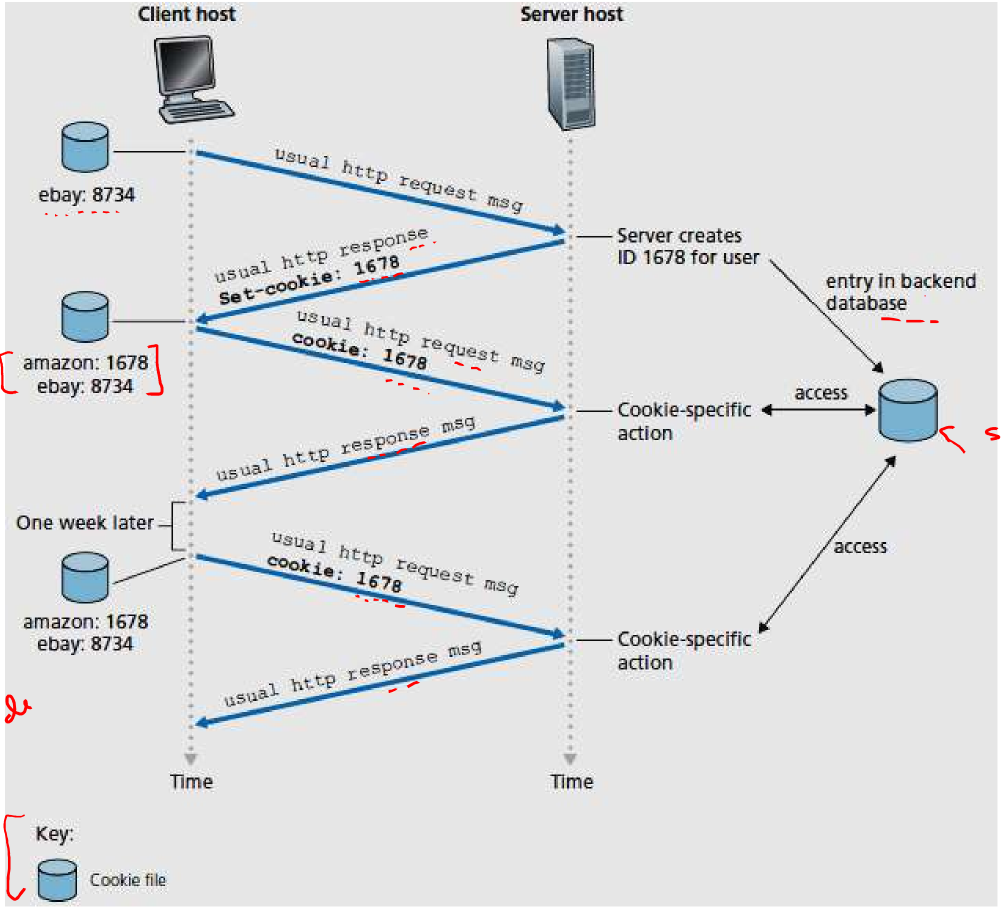


♦ General format of an HTTP response message

User-Server Interaction: Cookies

- We have discussed that an HTTP server is stateless. This simplifies server design and has permitted engineers to develop high-performance Web servers that can handle thousands of simultaneous TCP connections.
- However, it is often desirable for a Web site to identify users, either because the server wishes to restrict user access or because it wants to serve content as a function of the user identity. For these purposes, HTTP uses cookies. Cookies allow sites to keep track of users. Most major commercial Web sites use cookies today.
- The figure on next slide shows the cookie technology and its components:

User-Server Interaction: Cookies



server side.

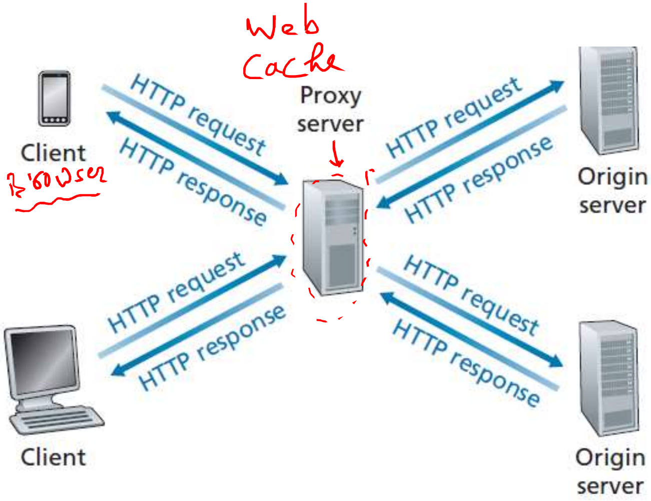
Browser side

User-Server Interaction: Cookies

- As shown in Figure (on previous slide), cookie technology has four components:
- (1) a cookie header line in the HTTP response message; (2) a cookie header line in the HTTP request message; (3) a cookie file kept on the user’s end system and managed by the user’s browser; (4) a back-end database at the Web site.
- Suppose Susan (user) always accesses the Web using Internet Explorer from her home PC, contacts Amazon.com for the first time. Let us suppose that in the past she has already visited the eBay site. When the request comes into the Amazon Web server, the server creates a unique identification number and creates an entry in its back-end database. The Amazon Web server then responds to Susan’s browser and includes **Set-cookie: header** in HTTP response which contains that unique identification number.
- When Susan’s browser receives the HTTP response, it sees the Set-cookie: header. Then the browser appends a line (identification number given by Amazon.com) to the special cookie file that it manages.
- When Susan browse Amazon.com again each time, her browser extracts her identification number for this site, and puts a cookie header line that includes that identification number in the HTTP request.

Web Caching

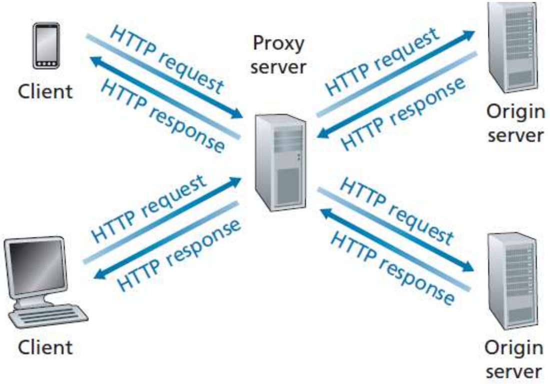
- A **Web cache**—also called a proxy server—is a network entity that satisfies HTTP requests on the behalf of an origin Web server.
- The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.
- As shown in Figure, a user's browser can be configured so that all of the user's HTTP requests are first directed to the Web cache. Once a browser is configured, each browser request for an object is first directed to the Web cache.
- As an example, suppose a browser is requesting the object http://www.someschool.edu/campus.gif. Here is what happens:



♦ Clients requesting objects through a Web cache

Web Caching

- (1) The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
- (2) The Web cache checks to see if it has a copy of the object stored locally. If it is having it, the Web cache returns the object within an HTTP response message to the client browser.
- (3) If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that is, to www.someschool.edu. The Web cache then sends an HTTP request for the object into the cache-to-server TCP connection. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.
- (4) When the Web cache receives the object, it stores a copy in its local storage and sends a copy (within an HTTP response message) to the client browser (over the existing TCP connection between the client browser and the Web cache).



- Clients requesting objects through a Web cache

Web Caching

- Typically a Web cache is purchased and installed by an ISP. For example, a university might install a cache on its campus network and configure all of the campus browsers to point to the cache. Or a major residential ISP (such as Comcast) might install one or more caches in its network and preconfigure its shipped browsers to point to the installed caches.
- Web caching has seen deployment in the Internet for two reasons:
  - ✓ First, a Web cache can substantially reduce the response time for a client request, particularly if the bottleneck bandwidth between the client and the origin server is much less than the bottleneck bandwidth between the client and the cache.
  - ✓ Second, Web caches can substantially reduce traffic on an institution's access link to the Internet. By reducing traffic, the institution (for example, a company or a university) does not have to upgrade bandwidth as quickly, thereby reducing costs

Thank you.