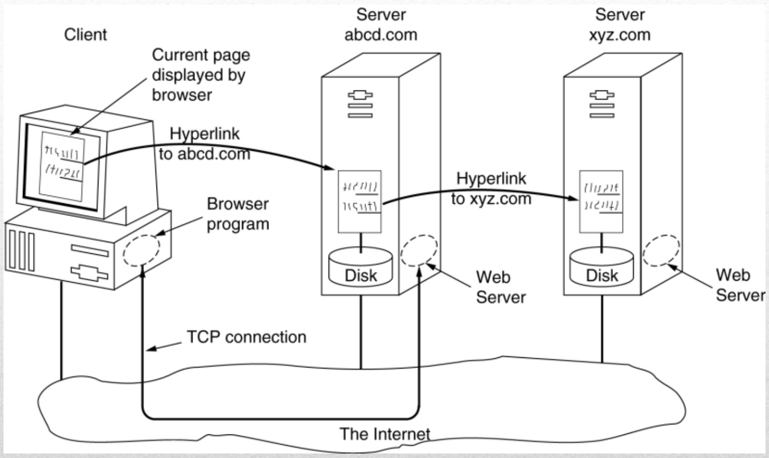
WWW

The World Wide Web

- The World Wide Web
- Architectural Overview
- Static Web Documents
- Dynamic Web Documents
- HTTP –The Hyper Text Transfer Protocol
- Performance Enhancements

The World Wide Web



The parts of the Web model.

Architectural Overview

What do you see in this link?

http://ccis.ksu.edu.sa/ar/computer-science/vision

This URL contains three main parts:

Protocol Name: **HTTP**

Machine Name: **www.ksu.edu.sa**

File Name: **/ar/computer-science/vision.htm**

Client Side: Website Request

How to make a browser view more file types? 1) Plug-ins

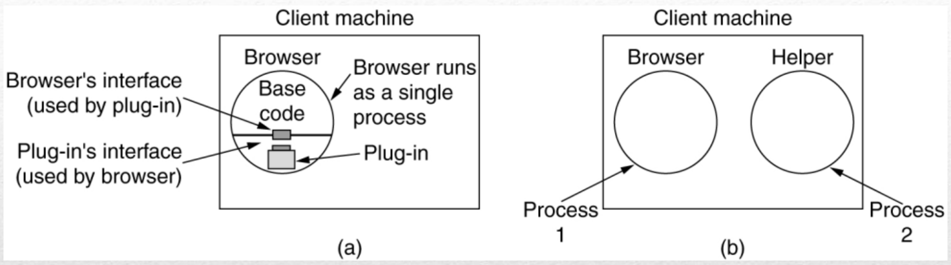
- A plug-in is a code module that the browser fetches from a special directory on the disk and installs as an extension to itself.
- A plug-in runs as an integral part of the browser (i.e. in the same process).
- Plug-ins has access to, and may modify the appearance of the current page (eg. run a video sequence within the browser window).
- A plug-in is removed from the browser’s memory upon leaving the page from where it is referenced.
- The interaction between the plug-in and the browser is through a browser-specific procedures interface.

Helper Applications

- A standalone application run as a separate process.
- The only interaction between the browser and the application is at invocation time (command line arguments, eg. a file path) and upon termination of the application.

Examples:

- Adobe Acrobat Reader (could be a plug-in too ??)
- Microsoft Word


Client machine
Browser
Base code
Browser runs as a single process
Browser's interface (used by plug-in)
Plug-in's interface (used by browser)
Plug-in
(a)
Client machine
Browser
Helper
Process 1
Process 2
(b)

(a)A browser plug-in. (b) A helper application

How to make a browser view more file types?

1.Accept a TCP connection from a client
2.Get the name of the file requested
3.Get the file (from disk)
4.Return the file to the client
5.Release the TCP connection

Problem:

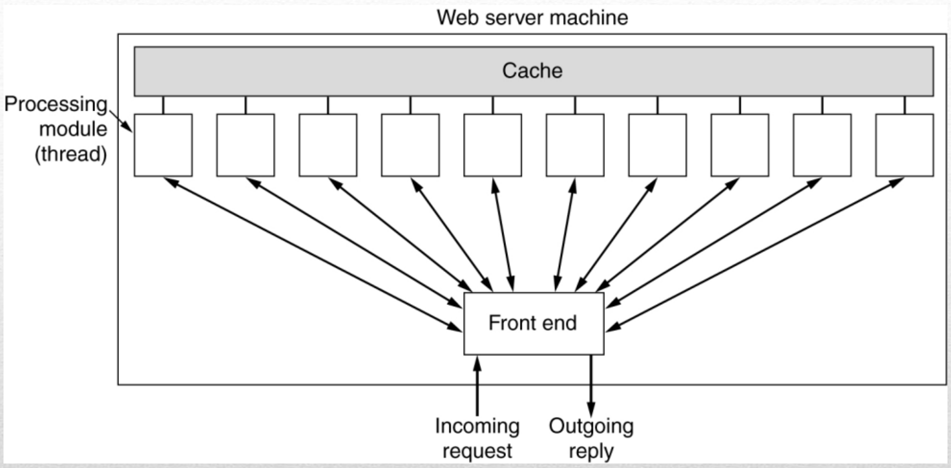
Every request requires making a disk access

1.Cache
2.Multithreading

The Server Side

- A problem with this design is that **every request requires making a disk access to get the file**. The result is that the Web server cannot serve more requests per second than it can make disk accesses. A high-end SCSI disk has an average access time of around 5 msec, which limits the server to at most 200 requests/sec, less if large files have to be read often. For a major Web site, this figure is too low.

The Server Side



A multithreaded Web server with a front end and processing modules.

The Server Side

- One obvious improvement (used by all Web servers) is to maintain a cache in memory of the *n* most recently used files. Before going to disk to get a file, the server checks the cache. If the file is there, it can be served directly from memory, thus eliminating the disk access. Although effective caching requires a large amount of main memory and some extra processing time to check the cache and manage its contents, the savings in time are nearly always worth the overhead and expense.
- The next step for building a faster server is to make the server multithreaded. In one design, the server consists of a front-end module that accepts all incoming requests and *k* processing modules. The *k* + 1 threads all belong to the same process so the processing modules all have access to the cache within the process' address space. When a request comes in, the front end accepts it and builds a short record describing it.

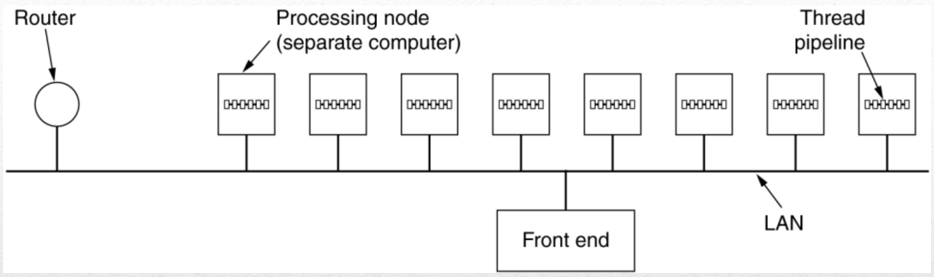
The Server Side

- It then hands the record to one of the processing modules. In another possible design, the front end is eliminated and each processing module tries to acquire its own requests, but a locking protocol is then required to prevent conflicts.
- The processing module first checks the cache to see if the file needed is there. If so, it updates the record to include a pointer to the file in the record. If it is not there, the processing module starts a disk operation to read it into the cache (possibly discarding some other cached files to make room for it). When the file comes in from the disk, it is put in the cache and also sent back to the client.
- The advantage of this scheme is that while one or more processing modules are blocked waiting for a disk operation to complete (and thus consuming no CPU time), other modules can be actively working on other requests. Of course, to get any real improvement over the single-threaded model, it is necessary to have multiple disks, so more than one disk can be busy at the same time. With *k* processing modules and *k* disks, the throughput can be as much as *k* times higher than with a single-threaded server and one disk.

The Server Side

1.Accept TCP connection
2.Resolve the name of the Web page requested
3.Authenticate the client
4.Perform access control on the client
5.Perform access control on the Web page
6.Check the cache
7.Fetch the requested page from disk
8.Determine the MIME type to include in the response
9.Take care of miscellaneous odds and ends
10.Return the reply to the client
11.Make an entry in the server log

Web server Advanced Functionality



What to do if too many requests come to the CPU?

Problem: no single cache.

Solutions:
1)Let Front End keep all requests
2) Use a shared memory multiprocessor

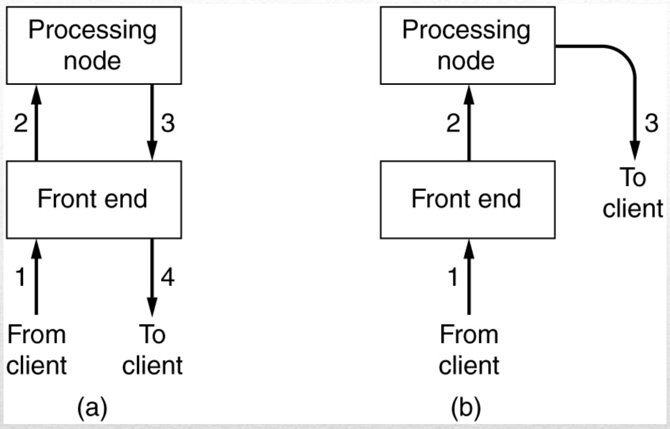
The Server Farm

The Server Farm

- If too many requests come in each second, the CPU will not be able to handle the processing load, no matter how many disks are used in parallel.
- The solution is to add more nodes (computers), possibly with replicated disks to avoid having the disks become the next bottleneck.
- This leads to the **server farm** model. A front end still accepts incoming requests but sprays them over multiple CPUs rather than multiple threads to reduce the load on each computer. The individual machines may themselves be multithreaded and pipelined as before.

The Server Side

- One problem with server farms is that there is no longer a shared cache because each processing node has its own memory—unless an expensive shared-memory multiprocessor is used.
- One way to counter this performance loss is to have a front end keep track of where it sends each request and send subsequent requests for the same page to the same node. Doing this makes each node a specialist in certain pages so that cache space is not wasted by having every file in every cache.
- Another problem with server farms is that the client's TCP connection terminates at the front end, so the reply must go through the front end. Where the incoming request (1) and outgoing reply (4) both pass through the front end.
- A trick, called **TCP handoff**, is used to get around this problem. With this trick, the TCP end point is passed to the processing node so it can reply directly to the client. This handoff is done in a way that is transparent to the client.



(a) Normal request-reply message sequence.
(b) Sequence when TCP handoff is used.

The Server Side

Name	Used for	Example
http	Hypertext (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
news	Newsgroup	news:comp.os.minix
news	News article	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Sending e-mail	mailto:JohnUser@acm.org
telnet	Remote login	telnet://www.w3.org:80

Some common URLs

URLs –Uniform Resource Locaters

HTTP is a stateless protocol

- HTTP is simply defines how clients and servers communicate with each other over the Web
- In many Web applications, maintaining state is important

–Ex: When a customer logs into a site such as Amazon, he may go through multiple pages

- We don't want to make him reenter information for each page

One way of maintaining state is via Cookies

What are Cookies

- A cookie is a small piece of information as a file (up to 4K) stored on the **client machine** in a user-specific cookies-directory
- Cookies are good for keeping track of return visitors
- Cookies are generated at the server side and is delivered to the browser before the Web page

If a user clicks on the link
 http://ccis.ksu.edu.sa/ar/computer-science/vision, the steps to display the page on the user's screen are as follows:

1.The browser determines the URL (by seeing what was selected).
2.The browser asks DNS for the IP address of *www.ksu.edu.sa*.
3.DNS replies with 212.138.39.200.
4.The browser makes a TCP connection to port 80 on 212.138.39.200.
5.It then sends a GET /ar/computer-science/vision.htm command.
6.The *www.ksu.edu.sa* server sends the file vision.htm.
7.The TCP connection is released.
8.The browser displays all the text in vision.htm.
9.The browser fetches and displays all images in vision.htm.

Steps to fetch and display a page

Client side:

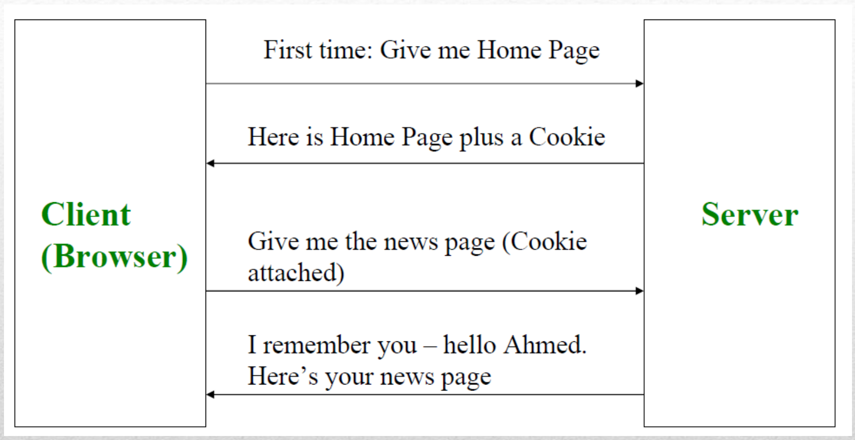
- When the user specifies a URL, the browser searches it's cookie directory for a cookie with the domain name specified in the URL.
- If a cookie for the actual domain exists, it is uploaded to the server with the page request.

Cookies –how do they work?

Server side:

- The first time a Web page is requested no cookie follows the request so the server creates a cookie and returns it before the requested page.
- For later visits to the same page, the request will contain the cookie generated at the previous visit.
- The server updates the cookie and returns it with the page
- This way the server ”remembers” the client from one visit to the next.

Cookies –how do they work?


Client (Browser)
First time: Give me Home Page
Here is Home Page plus a Cookie
Give me the news page (Cookie attached)
I remember you – hello Ahmed. Here’s your news page
Server

Cookie conversation

- Name :name of the cookie –typically used to extract / examine the cookie
- Value: contents of the cookie –seems like a simple value but can be an array if generated correctly
- Domain: domain of the server that is to receive the cookie –actual domain of server must match domain stored in the cookie
- –Idea is that other servers cannot look at all of your cookies tosee what you have
- –If not explicitly set in the cookie, it is the full domain of the server that created the cookie
- Expires: When cookie will expire
- Path: Path in server from which cookie can be sent
- –If not specified it is the full path from where cookie was set
- Secure: Does cookie require secure server using https
- –Default is no

Cookie format

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

Some examples of cookies

Cookie format

Cookie Storage

Temporary cookies:

A temporary or session cookie is stored only for your current browsing session, and is deleted from your computer when you close Internet Explorer.

Persistent cookies:

A persistent cookie is one stored as a file on your computer, and it remains there when you close Internet Explorer. The cookie can be read by the Web site that created it when you visit that site again.

Cookie Originator

- **A first-party cookie** either originates on or is sent to the Web site you are currently viewing. These cookies are commonly used to store information, such as your preferences when visiting that site.
- **A third-party cookie** either originates on or is sent to a Web site different from the one you are currently viewing. Third-party Web sites usually provide some content on the Web site you are viewing. For example, many sites use advertising from third-party Web sites and those third-party Web sites may use cookies. A common use for this type of cookie is to track your Web page use for advertising or other marketing purposes. Third-party cookies can either be persistent or temporary.

Cookies’Security

a cookie security loophole

- The loophole is available via email clients that can display graphics retrieved from the Web. You read the message, the graphic is downloaded, and you can then be assigned a unique serial number via a cookie.
- The serial number can be matched to your email address (because the assigner of the cookie sent you the email), and then you're effectively 'branded' --as you browse Web, any site with access to the data cross-referencing the cookie with your email ID knows who you are.

A Web site's privacy policy

- It tells you what kind of information the Web site collects, to whom it gives that information, and how it uses the information.
- Many Web sites provide privacy statements as written documents that you can view on the Internet. Web sites also might provide a Platform for Privacy Preferences (P3P) privacy policy.
- If a Web site has a P3P privacy policy, Internet Explorer can display it. Internet Explorer also might be able to compare your privacy settings to a representation of the P3P privacy policy, and determine whether or not to allow the Web site to save cookies on your computer.