
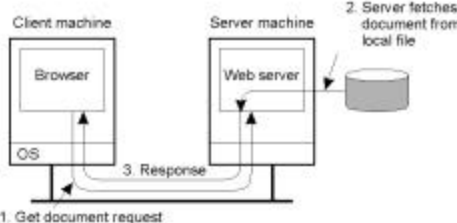


# CS4513 Distributed Computer Systems


The Web  
(Ch 11.1)



## The World Wide Web




- Huge client-server system
- Document-based
  - Referenced by "Uniform Resource Locator" (URL)



## Outline

- Introduction (done)
- Document Model (next)
- Architecture
- Communication
- Processes
- Naming
- Caching
- Security



## Document Model

- All information in documents
  - Typically in Hypertext Markup Language (HTML)
  - Different types: ASCII, scripts


```

<HTML>                                <!-- Start of HTML document -->
<BODY>                                  <!-- Start of the main body -->
<H1>Hello World</H1>                  <!-- Basic text to be displayed -->
</BODY>                                 <!-- End of main body -->
</HTML>                                 <!-- End of HTML section -->

<HTML>                                <!-- Start of HTML document -->
<BODY>                                  <!-- Start of the main body -->
<SCRIPT type = "text/javascript">      <!-- identify scripting language -->
  document.writeln ("<H1>Hello World</H1>"); // Write a line of text
</SCRIPT>                               <!-- End of scripting section -->
</BODY>                                 <!-- End of main body -->
</HTML>                                 <!-- End of HTML section -->

```


- Scripts give you "mobile code" (more later)
- Can also have Extensible Markup Language (XML)
  - Provides structure to document



## XML DTD

(1) <ELEMENT article (title, author+, journal)>	(#PCDATA is
(2) <ELEMENT title (#PCDATA)>	primitive type,
(3) <ELEMENT author (name, affiliation?)>	series of chars)
(4) <ELEMENT name (#PCDATA)>	
(5) <ELEMENT affiliation (#PCDATA)>	
(6) <ELEMENT journal (jname, volume, number?, month? pages, year)>	
(7) <ELEMENT jname (#PCDATA)>	
(8) <ELEMENT volume (#PCDATA)>	
(9) <ELEMENT number (#PCDATA)>	
(10) <ELEMENT month (#PCDATA)>	
(11) <ELEMENT pages (#PCDATA)>	
(12) <ELEMENT year (#PCDATA)>	

- Definition above refers to a journal article. Specifies type.
  - In a Document Type Definition (DTD)
  - Provides structure to XML documents




## XML Document

```

(1) <?xml = version "1.0">
(2) <!DOCTYPE article SYSTEM "article.dtd">
(3) <article>
(4)   <title>Prudent Engineering Practice for Cryptographic Protocols</title>
(5)   <author><name>M. Abadi</name></author>
(6)   <author><name>R. Needham</name></author>
(7)   <journal>
(8)     <jname>IEEE Transactions on Software Engineering</jname>
(9)     <volume>22</volume>
(10)    <number>12</number>
(11)    <month>January</month>
(12)    <pages>6 - 15</pages>
(13)    <year>1996</year>
(14)  </journal>
(15) </article>

```

- An XML document using the XML definitions from previous slide
- Formatting rules usually applied by embedding in HTML



## Document Types

- Beyond text → can include other types
  - Multipurpose Internet Mail Extensions (MIME)

Type	Subtype	Description
Text	Plain	Unformatted text
	HTML	Text including HTML markup commands
	XML	Text including XML markup commands
Image	GIF	Still image in GIF format
	JPEG	Still image in JPEG format
Audio	Basic	Audio, 8-bit PCM sampled at 8000 Hz
	Tone	A specific audible tone
Video	MPEG	Movie in MPEG format
	Pointer	Representation of a pointer device for presentations
Application	Octet-stream	An uninterrupted byte sequence
	Postscript	A printable document in Postscript
	PDF	A printable document in PDF
Multipart	Mixed	Independent parts in the specified order
	Parallel	Parts must be viewed simultaneously

- Includes types and sub-types
- Application specifies application-specific data type



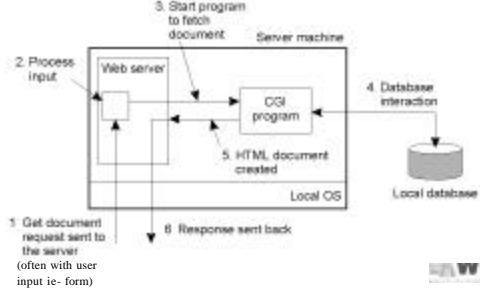
## Outline

- Introduction (done)
- Document Model (done)
- Architecture (next)
- Communication
- Processes
- Naming
- Caching
- Security



## Architectural Overview

- Text documents typically "processed" on client
  - But can be done at server, too
    - Common Gateway Interface (CGI)



## Server-Side Scripts

- Like Client, Server can execute JavaScript

```

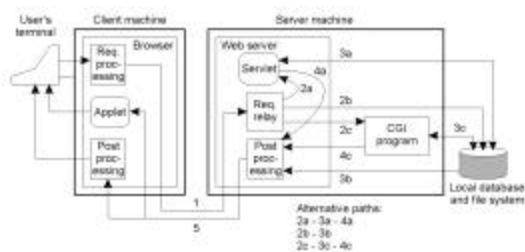
(1) <HTML>
(2) <BODY>
(3) <P>The current content of <pre>/data/file.txt</PRE>is:</P>
(4) <P>
(5) <SERVER type = "text/javascript">
(6)   clientFile = new File("/data/file.txt");
(7)   if (clientFile.open("r")){
(8)     while (!clientFile.eof())
(9)       document.writeln(clientFile.readLine());
(10)  clientFile.close();
(11)  }
(12) </SERVER>
(13) </P>
(14) <P>Thank you for visiting this site.</P>
(15) </BODY>
(16) </HTML>
    
```

(The tag <SERVER...> is system specific)

- Server can also pass pre-compiled code → *applet*
- <OBJECT codetype="application/java" classid="java.welcome.class">
- Servlets* are applets that run on the server side



## Overall Architectural Overview



## Outline

- Introduction (done)
- Document Model (done)
- Architecture (done)
- Communication (next)
- Processes
- Naming
- Caching
- Security



## HTTP Connections

- Communication based on Hypertext Transfer Protocol (HTTP)
  - client request, server reply protocol
  - uses TCP (why?\_)

- TCP connection setup expensive
- a) Using nonpersistent connections (HTTP 1.0)
- b) Using persistent connections (HTTP 1.1)
- Can also have requests in parallel

## HTTP Methods

Operation	Description
Head	Request to return the header of a document
Get	Request to return a document to the client
Put	Request to store a document
Post	Provide data that is to be added to a document (collection)
Delete	Request to delete a document

- *Head* used to verify object, get time modified
- *Get* can also retrieve only if matches tags
- *Put* and *Delete* used only if authorized (security later)

## HTTP Messages: Client → Server

- Request line required
- (Slide of additional headers later)

## HTTP Messages: Server → Client

- Status code indicates response
  - 200 means honor request ("OK")
  - 400 ("Bad Request")
  - 403 ("Forbidden")
  - 404 ("Not Found")

## HTTP Additional Headers

- Augment Client request or Server Response
- Accept encoding of gzip
- Upgrade to Secure HTTP
- Redirect for load balance

Header	Source	Contents
Accept	Client	The type of documents the client can handle
Accept-Charset	Client	Character sets are acceptable for the client
Accept-Encoding	Client	Document encodings the client can handle
Accept-Language	Client	The natural language the client can handle
Authorization	Client	A list of the client's credentials
WWW-Auth	Server	Security challenge to the client
Date	Both	Date and time the message was sent
ETag	Server	Tags associated with the returned document
Expires	Server	The time how long the response remains valid
From	Client	The client's e-mail address
Host	Client	The TCP address of the document's server
If-Match	Client	The tags the document should have
If-None-Match	Client	The tags the document should not have
If-Modified-Since	Client	Only return a document if newly modified
If-Unmodified-Since	Client	Return a document only if it has not been modified since the specified time
Last-Modified	Server	Time the returned document was last modified
Location	Server	Reference to which the client should redirect
Referer	Client	Client's most recently requested document
Upgrade	Both	App protocol the sender wants to switch to
Warning	Both	Information about the status of the data

## Outline

- Introduction (done)
- Document Model (done)
- Architecture (done)
- Communication (done)
- Processes (next)
- Naming
- Caching
- Security

### Client Process: Extensible Browser

- Need client browser to be extensible
  - Plug-in
  - Associated with document type (MIME type)

Browser's interface for plug-in

Plug-in is loaded on demand

Local file system

Plug-in's interface for browser

WPI

### Client-Side Process: Web Proxy

- Initially, handle connection when browser does not "speak" language

Browser

Web proxy

FTP server

HTTP request

FTP request

FTP response

HTTP response

- Initially, handle connection when browser does not "speak" language
- Now, most browsers can handle, but proxies still popular for common cache for many browsers
  - NZ, AOL

WPI

### Servers

Module

Module

Module

Handler

Apache core

Request

Response

Logical flow of control

- Core invokes modules with data
  - Actual module path depends upon data type
- Phases:
  - authentication, response, syntax checking, user-profile, transmission
- Extend server to support different types (PHP)

WPI

### Server Clusters (1)

- Single server can become heavily loaded

Web server

Web server

Web server

Web server

LAN

Front end

Request

Response

Front end handles all incoming requests and outgoing responses

- Front-end replicates request to back-end (horizontal distribution)

WPI

### Server Clusters (2)

Client

Request

Front end

Request (handed off)

Web server

Response

Web server

Web server

Logically a single TCP connection

(a)

- The principle of TCP handoff
  - But can't take advantage of document knowledge or caching
  - But higher-layer has to do more work, making front-end a bottleneck

WPI

### Server Clusters (3)

Client

Other messages

Setup request

Switch

Distributor

Web server

Dispatcher

Web server

Web server

1. Pass setup request to a distributor

2. Dispatcher selects server

3. Hand off TCP connection

4. Inform switch

5. Forward other messages

6. Server responses


(b)

- Distributor talks to dispatcher initially, then hands off connection
- Front-end switch can stay at TCP layer, told where to send data

WPI

### Outline

- Introduction (done)
- Document Model (done)
- Architecture (done)
- Communication (done)
- Processes (done)
- Naming (naming)
- Caching
- Security



### Uniform Resource Locators

Scheme	Host name	Pathname
http	:// www.cs.vu.nl	/home/steen/mbor

(a)


Scheme	Host name	Port	Pathname
http	:// www.cs.vu.nl	: 80	/home/steen/mbor

(b)

Scheme	Host name	Port	Pathname
http	:// 130.37.24.11	: 80	/home/steen/mbor


(c)

- Location-specific document location.
- a) Using only a DNS name (lookup IP, default port)
- b) Combining a DNS name with a port number (lookup IP).
- c) Combining an IP address with a port number.
- Note: tricks with DNS for load balancing



### URL Examples

Scheme Name	Used for	Example
http	HTTP	http://www.cs.vu.nl:80/globe
ftp	FTP	ftp://ftp.cs.vu.nl/pup/minx/README
file	Local file	file:/edu/book/work/chp/11/11
data	Inline data	data:text/plain;charset=iso-8859-7,%e1%e2%e3
telnet	Remote login	telnet://flits.cs.vu.nl
tel	Telephone	tel:+31201234567
modem	Modem	modem:+31201234567;type=v32




### Uniform Resource Names (URN)

- Location independent document specification


"urn"	Name space	Name of resource
urn	: ietf	: rfc:2648

- Easy to define name spaces, but hard to resolve
  - No general mechanisms
- URL + URN = URI
  - Uniform Resource Identifier




### Outline

- Introduction (done)
- Document Model (done)
- Architecture (done)
- Communication (done)
- Processes (done)
- Naming (done)
- Caching (next)
- Security



### Web Caching

- Browser keeps recent requests
  - Proxy can be valuable if *shared* interests
- Check cache first, server next
- Cache is full. How to decide replacement?
  - LRU (what is different than pages or disk blocks?)
  - GreedyDual (value divided by size)
- How consistent should the cache be to the server content? What are the tradeoffs?

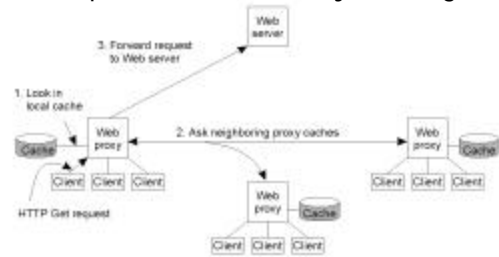


## Cache Coherency

- Strong consistency
  - validate each access
  - server indicates if invalid
  - but requires request to server for *each* client request
- Weak consistency
  - validate only when client clicks "refresh"
  - Or, using a heuristic Time To Live (TTL)
    - Squid  $T_{\text{expire}} = \alpha(T_{\text{cached}} - T_{\text{last\_modified}}) + T_{\text{cached}}$
    - $\alpha = 0.2$  (derived from practice)
- Why not have server *push* invalidation?
- In practice, cache hits low (50% max, only if really large)
  - Make "cooperative" caches



## Cooperative Web Proxy Caching



- Proxy first checks neighbors before asking server
  - Shown effective for 10,000+ user
- But complicated, and often not a clear win over single proxy



## Misc Caching

- Static vs. Dynamic Documents
  - Caching only effective for static documents (non CGI)
    - But Web increasingly dynamic (personalized)
    - Cookies used since server (mostly) stateless
  - Make proxies support active caching
    - Generate the HTML
    - Need copies of server-side scripts/code
    - Accessing databases harder
- Caching large documents
  - Can only send changes from original
  - Often, connection request is the large cost

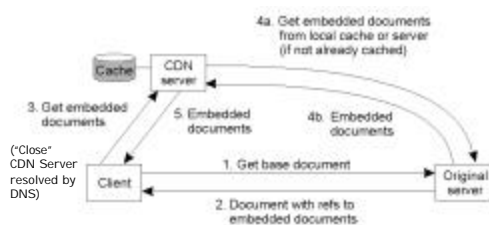


## Server Replication

- Clusters (covered)
- Deploy entire copy of Web site at another site (mirror)
  - Often done with FTP servers
  - Non-transparent
- Content Delivery Network (CDN)
  - Have network of cooperative caches run by the provider



## Akamai CDN



- Embedded documents have names that are resolved by Akamai DNS to a local CDN server
  - Use Internet "map" to determine local server
- Local server gets copy from original server
- Akamai has many CDN servers "close" to clients



## Outline

- Introduction (done)
- Document Model (done)
- Architecture (done)
- Communication (done)
- Processes (done)
- Naming (done)
- Caching (done)
- Security (next)
  - Secure Socket Layer (SSL)



## Security: Secure Communication Channel

- Need secure channel for transactions
  - Netscape's Secure Socket Layer (SSL)
  - More recent Transport Security Layer (TLS)

- Application independent
- Sits above transport layer
- Invoked by scheme "https"



## Establishing an SSL connection

1. Client sends SSL version number, cipher settings, randomly generated data and other information server needs.
2. Server sends server SSL version number, cipher settings, randomly generated data, servers own certificate.
  - (Optional) Server may request client's certificate. Client authenticates server certificate by using public key of certificate authority (CA)
3. Client creates *premaster key* for session and encrypts it with servers public key (obtained from server's certificate) and sends to server.
  - (Optional) Client sends encrypted data based on own private key if client needs authentication.
4. Server generates *master secret*, sends to server
5. Both client and server use master secret to generate *session keys*, which are symmetric keys for encryption/decryption of exchanged information during SSL session.
6. Client and server inform each other session key has been created.
7. SSL handshake is complete.

