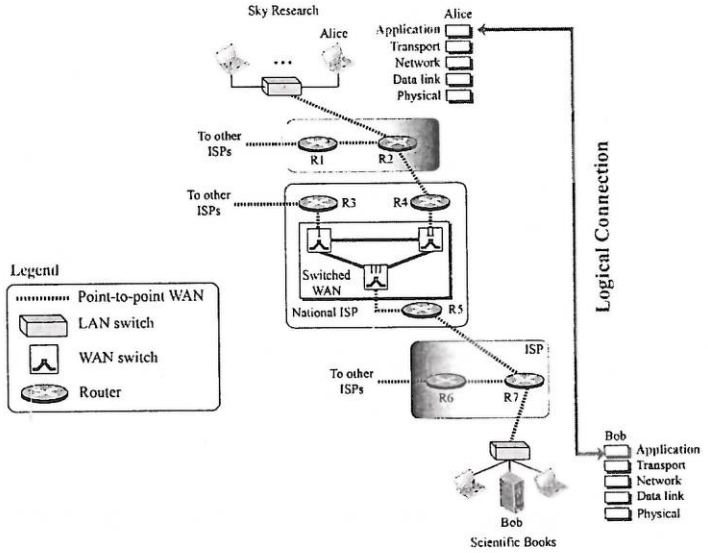
Application Layer Protocols

- Network Application Architectures
- WWW & HTTP
- Simple Mail Transfer System (SMTP)
- Domain Name System (DNS)
- P2P Applications
- Web Applications

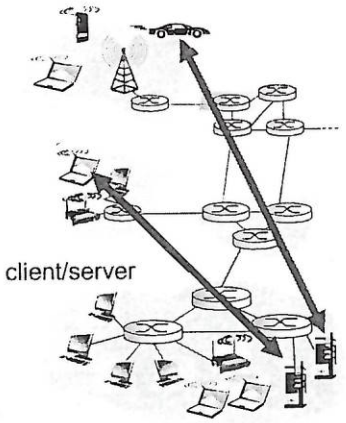
Connecting a Network Application



Connecting a Network App

- Write programs that
  - Run on (different) *end systems*
  - Communicate over network
  - e.g., web server software communicates with browser software
- No need to write software for network-core devices
  - Network-core devices do not run user applications
  - Applications on end systems allows for rapid app development, propagation
- Possible structure of applications
  - Client-Server
  - Peer-to-Peer (P2P)

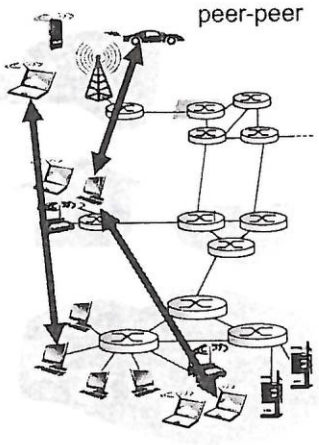
Client-Server Architecture



- Server
  - Always-on host
  - Permanent IP address
  - Data centers for scaling
- Clients
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly w/ each other

P2P Architecture

- *No* always-on server
- Arbitrary end system directly communicate
- Peers request service from other peers, provide service in return to other peers
  - *Self scalability* – new peers bring new service capacity, as well as new service demands
- Peers are intermittently connected and change IP addresses
  - Complex management



Processes

- Process: program running within a host
- Within same host, 2 processes communicate using inter-process communication (defined by OS)

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

- Processes in different hosts communicate by exchanging *messages*
- Applications with P2P architectures have client processes & server processes

What Transport Services Does and App Need?

Data Loss

- Some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- Other apps (eg. audio) can tolerate some loss

Timing

- Some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Throughput

- Some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- Other apps ("elastic apps") make use of whatever throughput they get

Security

- Encryption, authentication, data integrity, …

Transport Service Requirements: Common Apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec yes and no

WWW and HTTP

- A web page consists of objects
  - An object can be HTML file, JPEG image, Java applet, audio file, …
- A web page consists of base HTML file which includes several referenced objects

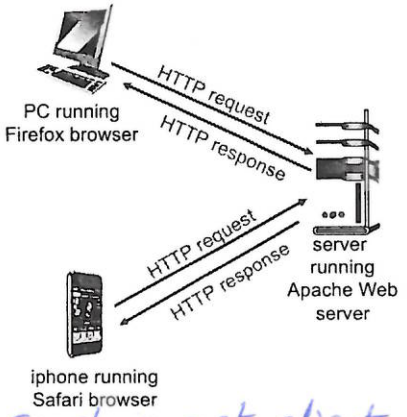
-Each object is addressable by a Uniform resource locator (URL)

`www.someschool.edu/someDept/pic.gif`

host name | path name

Hyper Text Transfer Protocol (HTTP)

- HTTP uses TCP
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP msgs exchanged between Web browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed
- HTTP is "stateless"



-Server maintains no info about past client requests
↳Because TCP does it

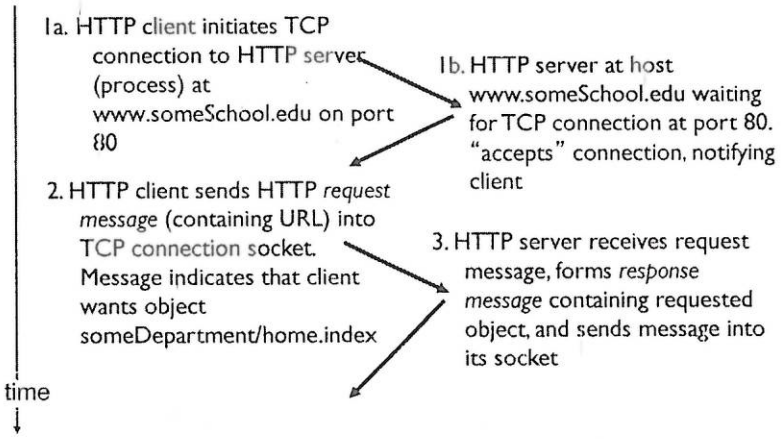
HTTP Connections

- Non-persistent HTTP
  - At most one object sent over TCP connection
  - Connection then closed
  - Downloading multiple objects requires multiple connections
- Persistent HTTP
  - Multiple objects can be sent over single TCP connection between client and server
    - What is the advantage?
- Default mode of HTTP

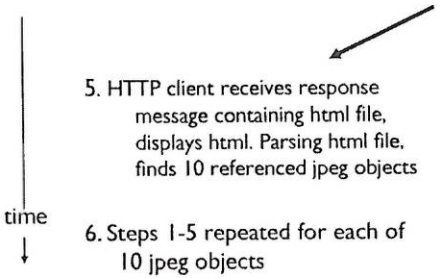
-Persistent connections w/ pipelining

Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`
(contains text, references to 10 jpeg images)



Non-persistent HTTP II



4. HTTP server closes TCP connection

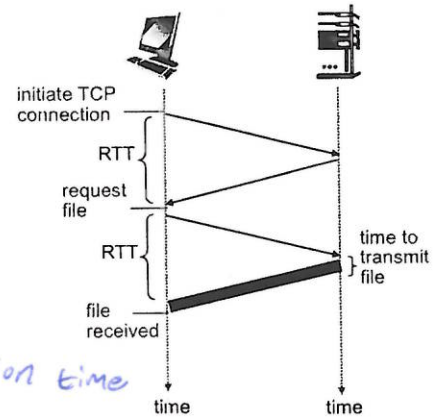
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

Non-persistent HTTP: Response Time

- One RTT to initiate TCP connection
- One RTT for HTTP request, and first few bytes of HTTP response to return
- File transmission time
- Non-persistent HTTP response time =

2 RTT + File transmission time



Q: What is the typical RTT for persistent HTTP for a file with ten objects? Bit bigger than non-persistent

Persistent HTTP

Non-persistent HTTP issues

- Requires 2 RTTs per object
- OS overhead for each TCP connection
- Browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server sent over open connection
- Client sends requests as soon as it encounters a referenced object

HTTP Request Message

- Two types of HTTP messages
  - *request, response*
- HTTP request message
  - ASCII (human-readable format)

carriage-return
line-feed character

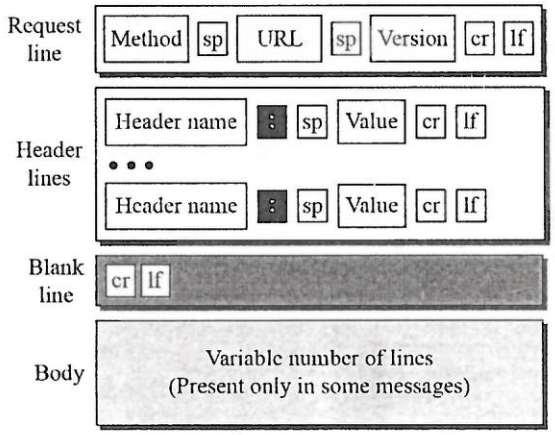
request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP Request Msg Format



Request message

HTTP/1.1 Method Types

- GET, POST, HEAD
- PUT
  - Uploads file in entity body to path specified in URL field
    - Used in conjunction with Web publishing tools
- DELETE
  - Deletes file specified in the URL field
- Uploading Form Input
  - POST method
    - Web page often includes form input
    - Input is uploaded to server in entity body
  - URL method
    - Uses GET method
    - Input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Q: What are the advantages and disadvantages of the GET & POST methods?

HTTP Response Message

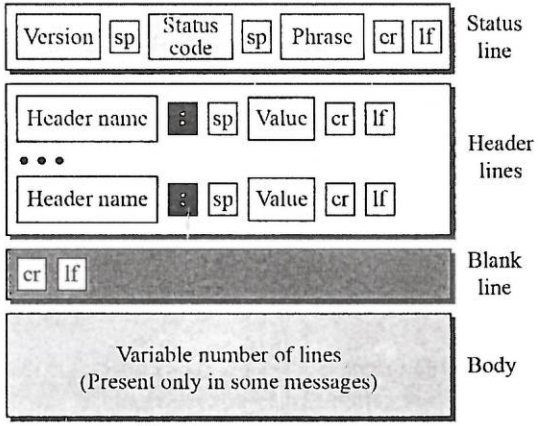
header lines

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

- used for caching

data, e.g., requested HTML file

HTTP Response Msg Format



Response message

Assignment: Use library to parse header. Use heap or something for cache. Web socket more efficient than socket. "Ping pong" between browser and real server, me in the middle

HTTP Response Codes

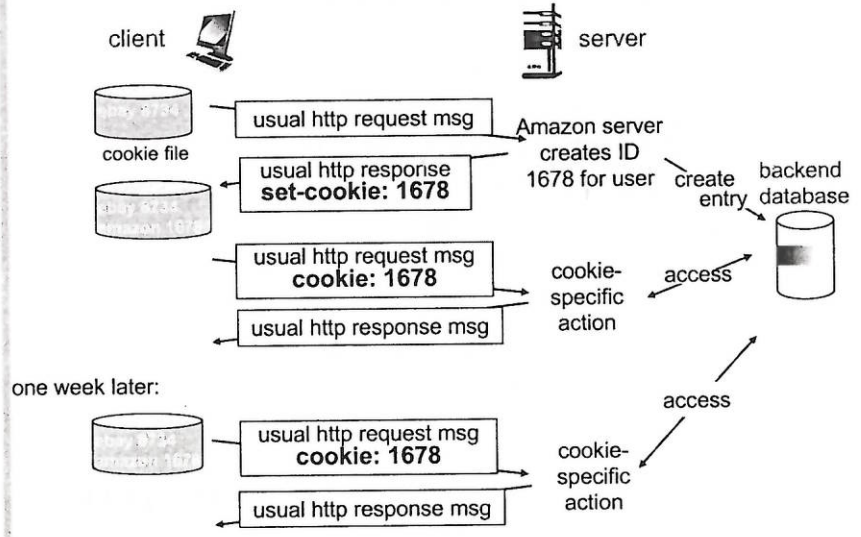
- Status code appears in 1st line in server-to-client response msg
- Some sample codes
  - 200 OK
    - Request succeeded, requested object later in this msg
  - 301 Moved Permanently
    - Requested object moved, new location specified later in this msg (Location)
  - 400 Bad Request
    - Request msg not understood by server
  - 404 Not Found
    - Requested document not found on this server
  - 505 HTTP Version Not Supported

Cookies

- What can cookies be used for?
  - Authorization
  - Shopping carts
  - Recommendations
  - User session state (Web Mail)
- How to keep "state"
  - HTTP has been designed as a stateless protocol
  - Maintain state at sender/receiver over multiple transactions
    - Cookies: HTTP messages carry state

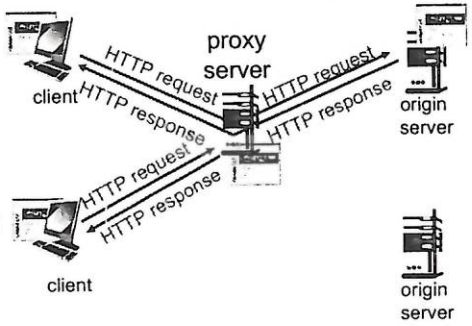
cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

Cookies: Keeping State



Web Caches (Proxy Server)

- Goal: satisfy client request without involving origin server
- User sets browser to accesses the web via proxy.
- Browser sends all HTTP requests to proxy server
  - Object in cache
    - Proxy returns object
  - Else proxy requests object from origin server
    - Returns object to client



More About Web Caching

- Cache acts as both client and server
  - Server for original requesting client
  - Client to origin server
- Typically cache is installed by ISP
  - University, company, residential ISP

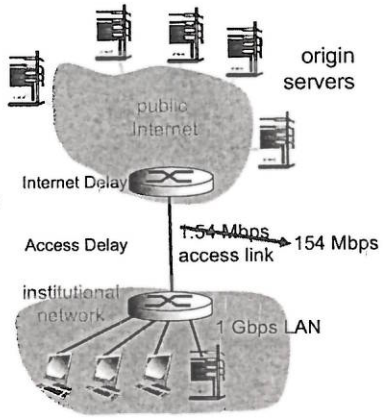
Q: Why Web caching?

- Reduce response time for client request

• Reduce traffic on an institution's access link

Caching Example: Fatter Access Link

- Assumptions
  - Avg object size: 100K bits
  - Avg request rate from browsers to origin servers:15 requests/sec
  - Avg data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec (Internet Delay)
  - Access link rate: ~~1.54 Mbps~~ → 154 Mbps
- Consequences
  - LAN utilization: $15 * 10^5/10^9 = 0.15\%$
  - Access link utilization = ~~97%~~ → 0.97%
    - $15 * 10^5/1.54 * 10^6$
  - Total Delay = Internet Delay + Access Delay + LAN Delay
  - = 2 sec + ~~minutes~~ + usecs → msecs



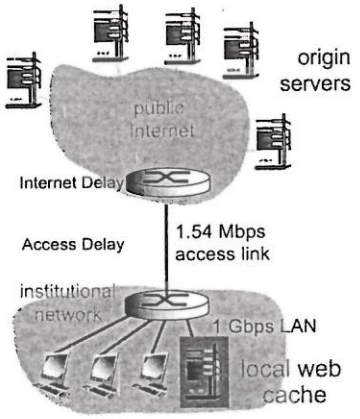
Cost: increased access link speed (not cheap)

Caching Example: Install Local Cache

- Assumptions
  - Avg object size: 100K bits
  - Avg request rate from browsers to origin servers:15 requests/sec
  - Avg data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - Access link rate: 1.54 Mbps
- Consequences
  - LAN utilization: 0.15%
  - Access link utilization = ?
  - Total delay = ?

How to compute link utilization, delay?

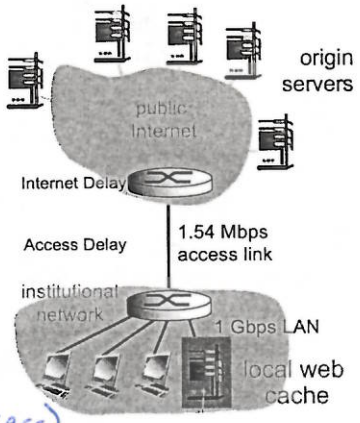
Cost: web cache (cheap!)



Caching Example: Install Local Cache II

Calculating access link utilization, delay with cache

- Suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- Access link utilization
  - 60% of requests use access link
- Data rate to browsers over access link = 0.6*1.50 Mbps = 0.9 Mbps
  - Utilization = 0.9/1.54 = 0.58
- Total Delay
  - = 0.6 * (delay from origin servers) + 0.4 * (delay when satisfied at cache)
  - 0.6 (2.01) + 0.4 (msecs)
  - = ~ 1.2 secs
  - Less than with 154 Mbps link (and cheaper too!)

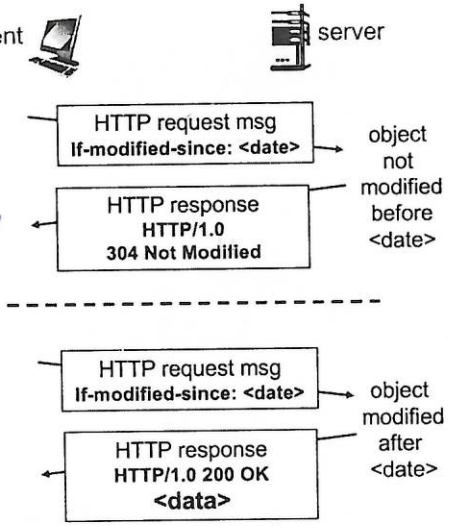


Conditional GET

- Goal: Do not send object if cache has up-to-date cached version
  - No object transmission delay
  - Longer link utilization
- Cache: specify date of cached copy in HTTP request

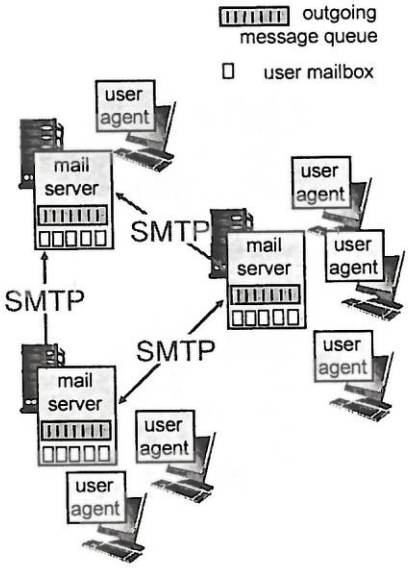
  If-modified-since: <date>
- Server: response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified



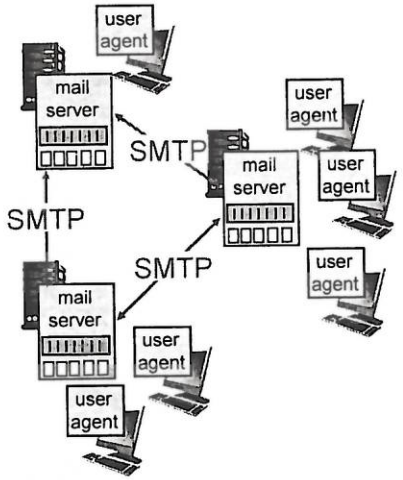
Electronic Mail

- Three major components
  - User Agents
  - Mail Servers
  - Simple Mail Transfer Protocol (SMTP)
- User Agent
  - a.k.a. "mail reader"
  - Composing, editing, reading mail messages
  - e.g., Outlook, Thunderbird, iPhone mail client
  - Outgoing, incoming messages stored on server



Mail Servers

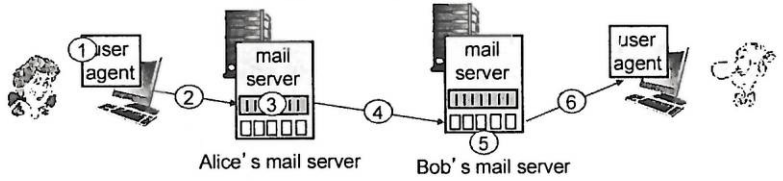
- Mailbox contains incoming messages for user
- Message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - Client: sending mail server
  - Server: receiving mail server



SMTP

- Uses TCP to reliably transfer email message from client to server on port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - Handshaking (greeting)
  - Transfer of messages
  - Closure
- Command/response interaction (like HTTP, FTP)
  - Commands: ASCII text
  - Response: status code and phrase

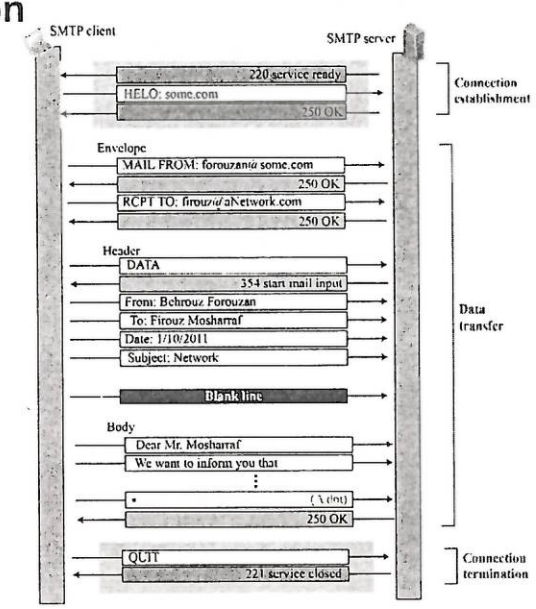
Alice Sends Msg to Bob



- 1) Alice uses UA to compose message "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server
  - Msg placed in queue
- 3) Client side of SMTP opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message

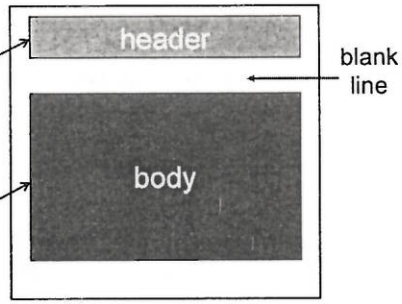
SMTP Interaction

Exercise: Try SMTP interaction for yourself
– telnet *servername* 25



Mail Message Format

- RFC 822: standard for text message format
  - header lines, e.g.,
  - To:
  - From:
  - Subject:



- Body: the "message"
  - ASCII characters only

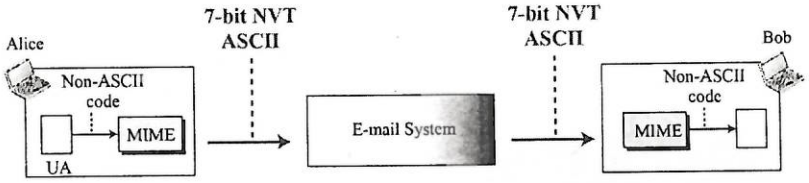
SMTP Final Words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF to determine end of message

Comparison with HTTP

- HTTP: pull
  - SMTP: Push
- Both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
  - SMTP: Multiple objects in a single message

Multipurpose Internet Mail Extension (MIME)



- Can only send messages in 7-bit ASCII format
  - Cannot ~~send~~ be used for other languages e.g. French, Chinese etc...
- MIME is a supplementary protocol that allows non-ASCII data to be sent via SMTP
  - Converts non-ASCII to ASCII and vice versa

MIME Header

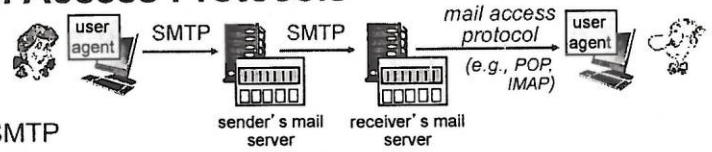
MIME headers

E-mail header
MIME-Version: 1.1 Content-Type: type/subtype Content-Transfer-Encoding: encoding type Content-ID: message ID Content-Description: textual explanation of nontextual contents
E-mail body

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix C)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

Type	*Description*
7-bit	NVT ASCII characters with each line less than 1000 characters
8-bit	Non-ASCII characters with each line less than 1000 characters
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters encoded as an equal sign plus an ASCII code

Mail Access Protocols



- SMTP
  - Delivery/storage to receiver's server
- Mail Access Protocol
  - Retrieval from server
- POP: Post Office Protocol [RFC 1939]
  - Authorization, Download
- IMAP: Internet Mail Access Protocol [RFC 1730]
  - More features, including manipulation of stored msgs on server
- HTTP - Gmail, hotmail etc

POP3 Protocol

Authorization Phase

- Client commands
  - user: declare username
  - pass: password
- Server responses
  - +OK
  - -ERR

Transaction phase, client:

- list: list message numbers
- retr: retrieve message by number
- dele: delete
- quit

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 and IMAP

POP3

- Previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients

• POP3 is stateless across sessions

IMAP

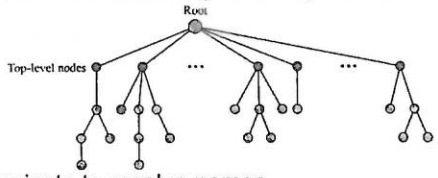
- Keeps all messages in one place: at server
- Allows user to organize messages in folders

• keep user state across sessions

  - Names of folders and mappings between message IDs and folder name

Domain Name System (DNS)

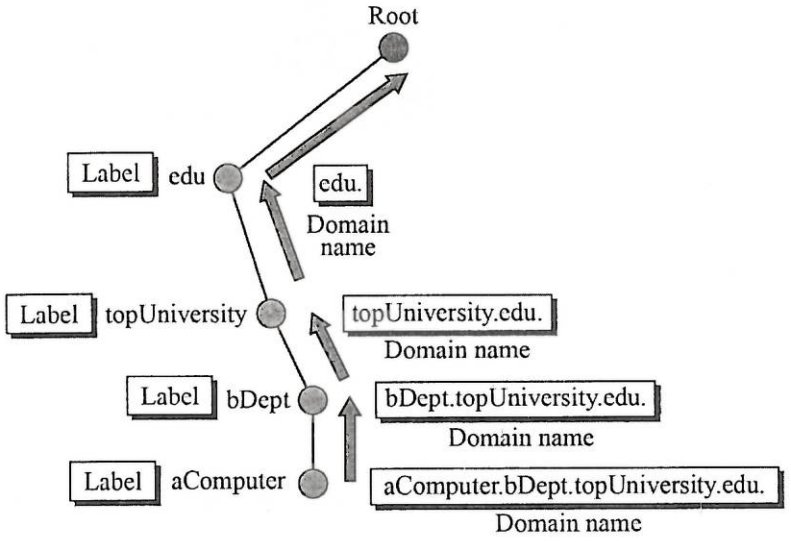
- Distributed database implemented in hierarchy of many name servers



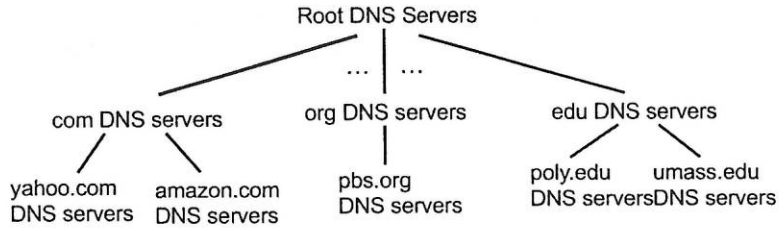
- Application-layer protocol
  - Hosts, name servers communicate to resolve names (address/name translation)
- DNS services
  - Hostname to IP address translation
  - Host aliasing
  - Canonical, alias names
  - Mail server aliasing
  - Load distribution
  - Replicated Web servers

• Many IP addresses correspond to one name

Domain Name and Labels



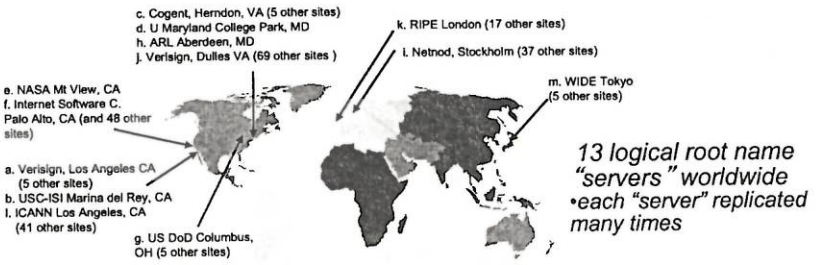
DNS: A Distributed Hierarchical Database



- Client wants IP for www.amazon.com
- Client queries *root* server to find .com DNS server
- Client queries .com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: Root Name Servers

- Contacted by local name server that can not resolve name
- Root name server
  - Returns list of IP addresses for responsible TLD servers
  - www.digwebinterface.com



13 logical root name "servers" worldwide
•each "server" replicated many times

TLD and Authoritative Servers

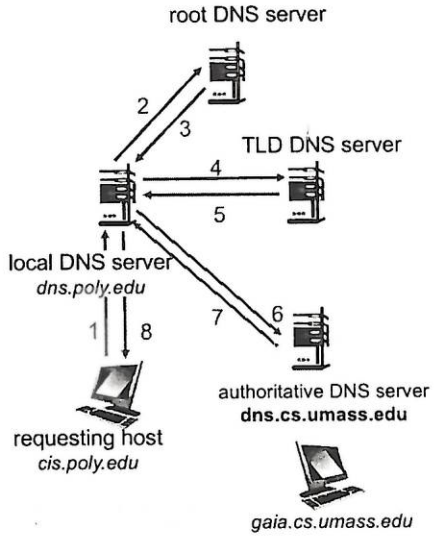
- Top-level Domain (TLD) Servers
  - Responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g. uk, fr, ca, jp
    - Network Solutions maintains servers for .com TLD
    - Educause for .edu TLD
- Authoritative DNS Servers
  - Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
    - Can be maintained by organization or service provider

Local DNS

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
  - Also called "default name server"
- When host makes DNS query, query is sent to its local DNS server
  - Has local cache of recent name-to-address translation pairs (but may be out of date!)
  - Acts as proxy, forwards query into hierarchy

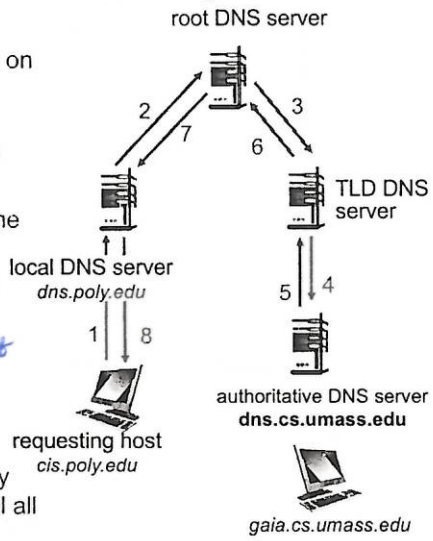
DNS Name Resolution Example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
- Iterated Query
  - Contacted server replies with name of server to contact
    - "I don't know this name, but ask this server"



DNS Name Resolution & Caching

- Recursive query
  - Puts burden of name resolution on contacted name server
- Once a name server learns of a mapping, it caches it
  - Cache entries timeout after some time (TTL)
  - TLD servers typically cached in local name servers
- Root name servers not often visited
- Cached entries may be out-of-date
  - If host changes IP address, may not be known Internet-wide until all TTLs expire



DNS Records

- Distributed database storing resource records (RR)

RR format: `(Name, Class, Type, TTL, Value)`

- A: IPv4 address record
- AAAA: IPv6 address record (128 bits)
- NS: Name Server record
- CNAME: Canonical Name (i.e. alias) record
- MX: Mail Exchange record

A & AAAA Records

- A Record - IPv4 address record

www.ripe.net. IN A 193.0.6.139

name type value

- AAAA Record – IPv6 address record

www.ripe.net. IN AAAA 2001:67c:2e8:22::c100:68b

NS Record

- Name is domain (e.g. foo.com)
- Value is hostname of authoritative server for this domain

NAME	TTL	TYPE	DATA
ns.example.com.	1800	A	192.168.1.2
example.com.	1800	NS	ns.example.com.

Local / domain – never use this

CNAME Record

- Name is alias for some "canonical" (the real) name

NAME	TTL	TYPE	DATA
www.example.com.	1800	A	192.168.1.2
ftp.example.com.	1800	CNAME	www.example.com

MX Record

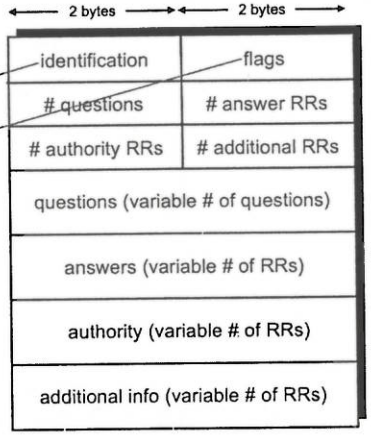
- Value is name of mail server associated with Name

NAME	TTL	TYPE	DATA	MX LEVEL
mail1.example.com.	1800	A	192.168.1.2	
mail2.example.com.	1800	A	192.168.1.4	
example.com.	1800	MX	mail1.example.com.	10
example.com.	1800	MX	mail2.example.com.	20
example.com.	1800	MX	mail100.backupexample.com.	

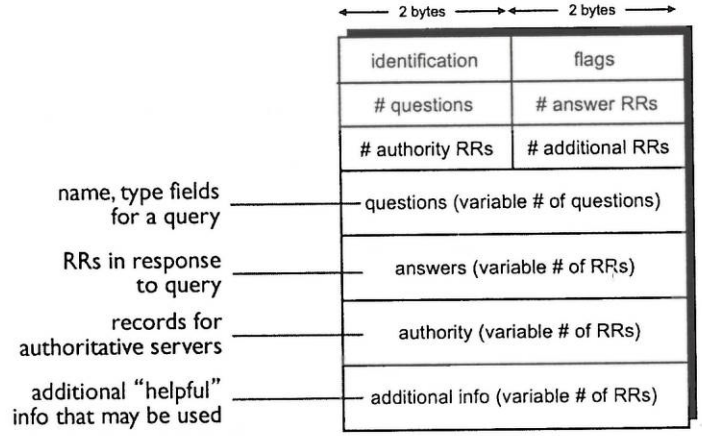
DNS protocol, messages

- *query* and *reply* messages, both with same message format

msg header
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS protocol, messages



Inserting Records into DNS

- Example: new startup "Network Utopia"
- Register name networkuptopia.com at DNS registrar (e.g., Network Solutions)
  - Provide names, IP addresses of authoritative name server (primary and secondary)
  - Registrar inserts two RRs into .com TLD server
    - (networkutopia.com, NS, dns1.networkutopia.com)
    - (dns1.networkutopia.com, A, 212.212.212.1)
- Create
  - Authoritative server type A record for www.networkuptopia.com

- Type MX record for networkutopia.com

Attacking DNS

DDoS Attacks

- Bombard root servers with traffic
  - Not successful to date
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass

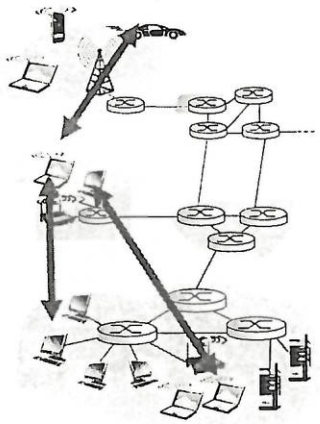
Redirect Attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus replies to DNS servers which caches.

- Response come back in single UDP packet to be quick and avoid complications

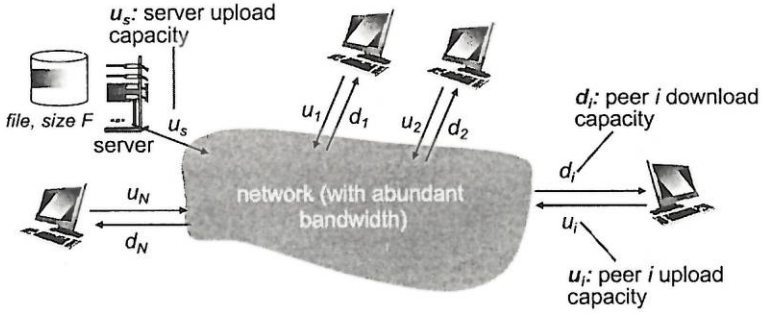
Pure P2P Architecture

- No always-on server
- Arbitrary end systems directly communicate
  - Peers are intermittently connected & change IP addresses.
- Examples
  - File distribution (BitTorrent)
  - Streaming (KanKan)
  - VoIP (Skype)



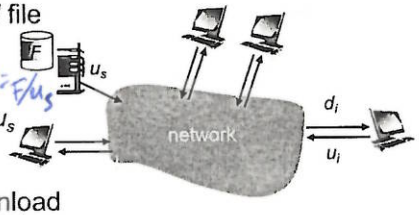
File Distribution – Client-Server vs P2P

- Question: how much time to distribute file (size F) from one server to N peers?
  - Peer upload/download capacity is limited resource



File Distribution Time: Client-Server

- Server transmission: must sequentially send (upload) N file copies
  - Time to send one copy: F/u_s
  - Time to send N copies: NF/u_s



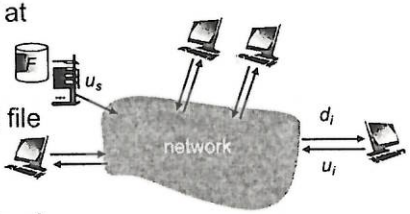
- Client: each client must download file copy
  - d_{min} = min client download rate
  - Min client download time: F/d_{min}

time to distribute F to N clients using client-server approach $D_{cs} \geq max\{NF/u_s, F/d_{min}\}$

increases linearly in N

File Distribution Time: P2P

- Server transmission: must upload at least one copy
  - Time to send one copy: F/u_s
- Client: each client must download file copy
  - Min client download time: F/d_{min}
- Clients: as aggregate must download NF bits
  - Max upload rate (limiting max download rate) is $u_s + \Sigma u_i$



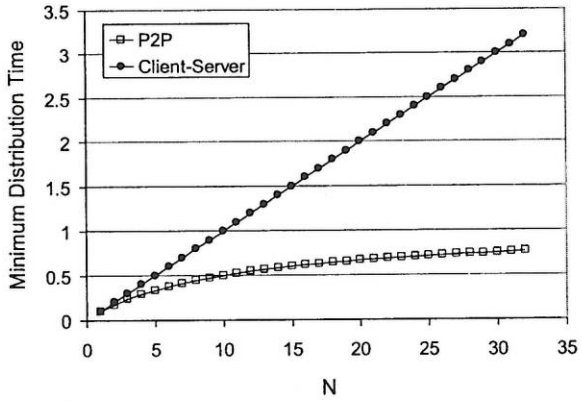
time to distribute F to N clients using P2P approach $D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$

increases linearly in N …

… but so does this, as each peer brings service capacity

Client-Server vs P2P Example

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

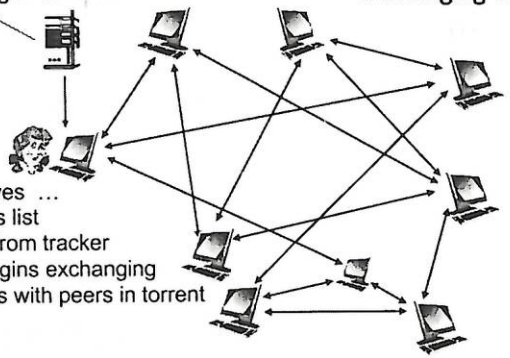


P2P File Distribution: BitTorrent

- File divided into 256KB chunks
- Peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

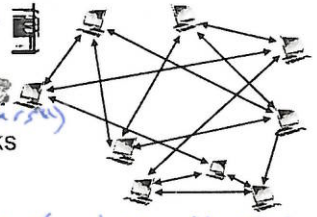
torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

P2P File Distribution: BitTorrent II

- Peer joining torrent
  - Has no chunks, but will accumulate them over time from other peers
  - Registers w/ tracker to get list of peers, connects to subset of peers ("neighbors")
- While downloading, peer uploads chunks to other peers
- Peers may change peers w/ whom it exchanges chunks
- Churn: peers may come and go
- Once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

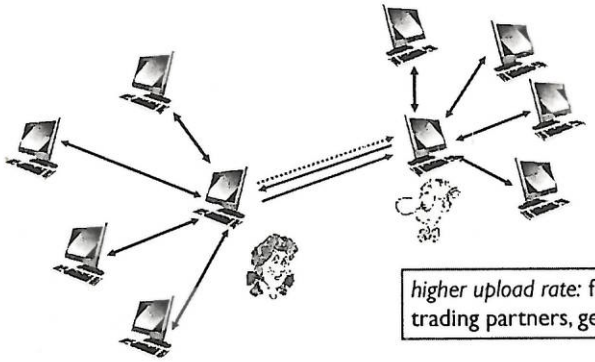


BitTorrent: Requesting, Sending File Chunks

- Requesting chunks
  - At any given time, different peers have different subsets of file chunks
  - Periodically, Alice asks each peer for list of chunks that they have
  - Alice requests missing chunks from peers
    - Rarest first
- Sending chunks: tit-for-tat
  - Alice sends chunks to those four peers currently sending her chunks at highest rate
    - Other peers are choked by Alice
    - Re-evaluate top 4 every 10 secs
- Every 30 secs: randomly select another peer, starts sending chunks
  - "Optimistically unchoke" this peer
    - Newly chosen peer may join top 4

BitTorrent: Tit-for-Tat

- Alice "optimistically unchokes" Bob
  - Alice becomes one of Bob's top-four providers
- Bob reciprocates
  - Bob becomes one of Alice's top-four providers

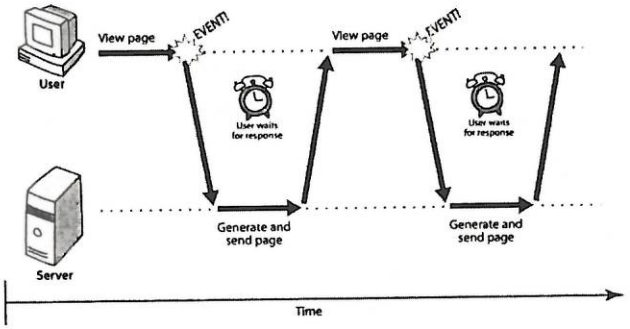


higher upload rate: find better trading partners, get file faster!

Web Applications

- Usability of web applications has lagged behind that of desktop applications
- Rich Internet Applications (RIAs)
  - Web applications that approximate the look, feel and usability of desktop applications
- Two key attributes
  - Performance and rich GUI.
- RIA performance
  - Comes from Ajax which uses client-side scripting to make web applications more responsive

Synchronous Web Communication



- Synchronous: user must wait while new pages load
  - Typical communication pattern used in web pages (click, wait, refresh)
- While a synchronous request is being processed on the server, the user cannot interact with the client web browser

Traditional Web Applications

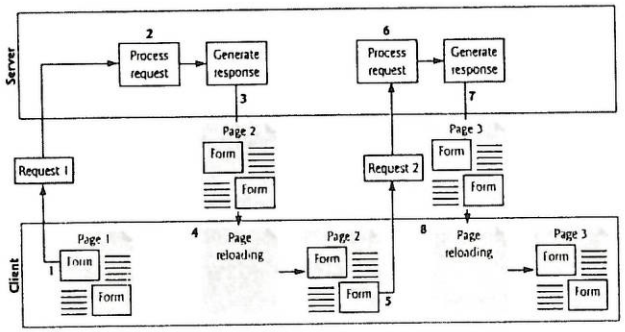
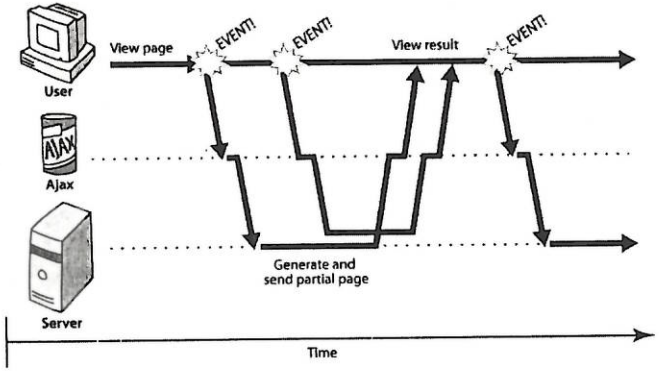


Fig. 16.1 | Classic web application reloading the page for every user interaction.

Web Applications and Ajax

- Web Application: a dynamic web site that mimics the feel of a desktop app
  - Presents a continuous user experience rather than disjoint pages
  - Examples: Gmail, Google Maps, Google Docs, Flickr
- Ajax: *Asynchronous XML and JavaScript*
  - Not a programming language; a particular way of using JavaScript
    - Download data from server in the background
  - Allows dynamically updating a page without making the user wait
    - Avoids the "click-wait-refresh" pattern

Asynchronous Web Communication



- Asynchronous: user can keep interacting with page while data loads
  - Communication pattern made possible by Ajax

Typical Ajax Request I

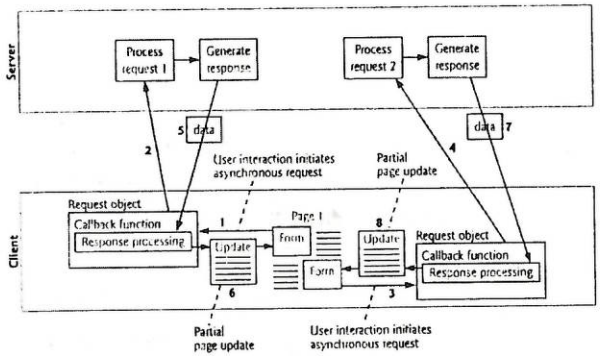
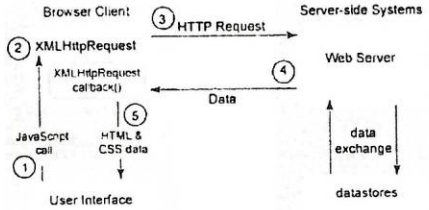


Fig. 16.2 | Ajax-enabled web application interacting with the server asynchronously.

Typical Ajax Request II

1. User clicks, invoking an event handler
2. Handler's code creates an XMLHttpRequest (XHR) object
3. XHR object requests page from server
4. Server retrieves appropriate data, sends it back
5. XHR object fires an event when data arrives
   - Often called a callback
   - You can attach a handler function to this event
6. Callback event handler processes the data and displays it



XMLHttpRequest (XHR) Object

- XMLHttpRequest object
  - Resides on the client
  - Layer between the client and the server that manages asynchronous requests in Ajax applications
- To initiate an asynchronous request
  - Create an instance of the XMLHttpRequest object
  - Use its *open* method to set up the request, and its *send* method to initiate the request
- For security purposes the XHR object does not allow a web application to request resources from domains other than the one that served the application
  - Same Origin Policy (SOP)

Example Using the XHR Object

a) User hovers over C++ *How to Program* book-cover image, causing an asynchronous request to the server to obtain the book's description. When the response is received, the application performs a *partial page update* to display the description.

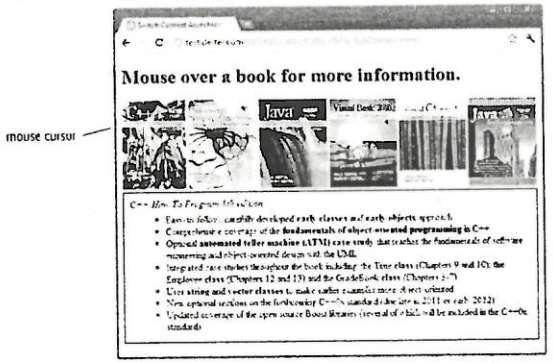


Fig. 16.5 | Asynchronously display content without reloading the page. (Part 6 of 7.)

Example Using the XHR Object II

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 16.5: SwitchContent.html -->
4  <!-- Asynchronously display content without reloading the page. -->
5  <html>
6  <head>
7     <meta charset = "utf-8">
8     <style type = "text/css">
9        .box { border: 1px solid black; padding: 10px }
10    </style>
11    <title>Switch Content Asynchronously</title>
12    <script>
13       var asyncRequest; // variable to hold XMLHttpRequest object
14
15       // set up event handlers
16       function registerListeners()
17       {
18          var img;
19          img = document.getElementById( "cpphtp" );
20          img.addEventListener( "mouseover",
21             function() { getContent( "cpphtp8.html" ); }, false );
22          img.addEventListener( "mouseout", clearContent, false );
```


Example Using the XHR Object III

```
45       // set up and send the asynchronous request.
46       function getContent( url )
47       {
48          // attempt to create XMLHttpRequest object and make the request
49          try
50          {
51             asyncRequest = new XMLHttpRequest(); // create request object
52
53             // register event handler
54             asyncRequest.addEventListener(
55                "readystatechange", stateChange, false);
56             asyncRequest.open( "GET", url, true ); // prepare the request
57             asyncRequest.send( null ); // send the request
58          } // end try
59          catch ( exception )
60          {
61             alert( "Request failed." );
62          } // end catch
63       } // end function getContent
64
```

When the third argument to XMLHttpRequest method open is true, the request is asynchronous

The WebSocket Protocol

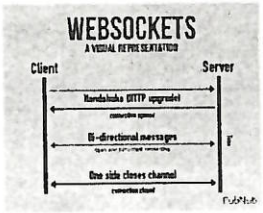
Half duplex: one req, one response

- AJAX always has to poll the server for data rather than receive it via push from the server
  - If you're moving a high volumes of data - overhead of creating an HTTP connection every time is going to be a bottleneck
- WebSockets allow your client-side JavaScript to open and persist a connection to a server
  - With WebSockets, data is exchanged as messages, which can happen very quickly due to the persistent connection
- Another powerful aspect of WebSockets is a capability called full duplex

– Contrast w/. AJAX where server has no method for pushing messages to client

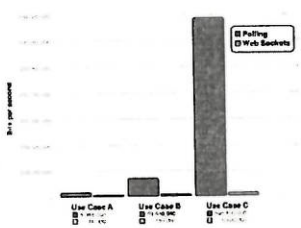
How WebSockets Work

- WebSocket specification defines an API establishing a two-way "socket" connections between a web browser and server
  - Persistent connection between the client and server
    - Both parties can start sending data at any time
- The client establishes a WebSocket connection through a process known as the WebSocket handshake
  - Process starts with the client sending a regular HTTP request to the server



  - *Upgrade* header is included in this request
    - Informs server that client wishes to establish WebSocket connection.

WebSocket Efficiency



- With WebSockets you can transfer as much data as you like in both directions simultaneously
  - Without incurring overhead associated w/ traditional HTTP requests
- Data is transferred through a WebSocket as *messages*
  - Each of which consists of one or more frames containing the data you are sending (the payload)
    - 2-byte framing overhead
- Using this frame-based messaging system helps to reduce the amount of non-payload data that is transferred
  - Leading to significant reductions in latency

WebSocket API

```
var ws = new WebSocket('wss://example.com/socket'); (1)

ws.onerror = function (error) { ... } (2)
ws.onclose = function () { ... } (3)

ws.onopen = function () { (4)
  ws.send("Connection established. Hello server!"); (5)
}

ws.onmessage = function(msg) { (6)
  if(msg.data instanceof Blob) { (7)
    processBlob(msg.data);
  } else {
    processText(msg.data);
  }
}
```

1. Open a new secure WebSocket connection (wss)
2. Optional callback, invoked if a connection error has occurred
3. Optional callback, invoked when the connection is terminated
4. Optional callback, invoked when a WebSocket connection is established
5. Client-initiated message to the server
6. A callback function invoked for each new message from the server
7. Invoke binary or text processing logic for the received message

- Use websocket library to connect proxy to real server.