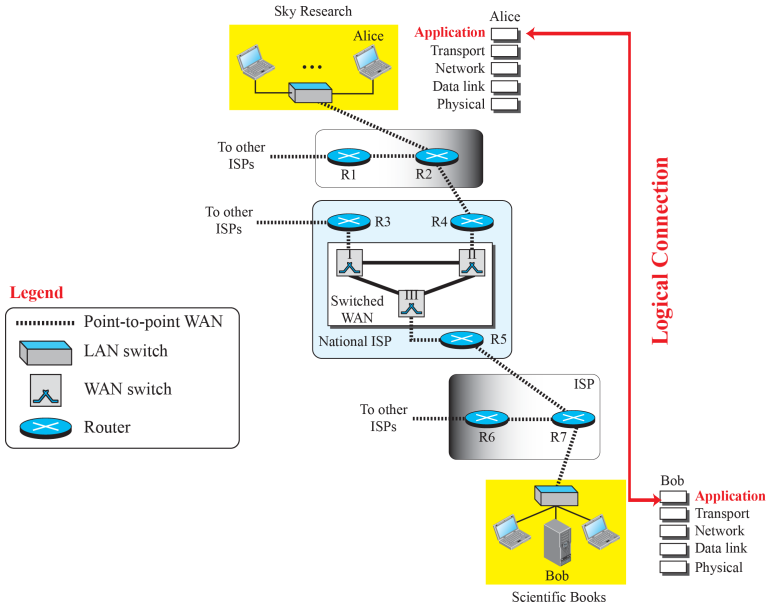
Application Layer Protocols

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

1

Connecting a Network Application



2

3

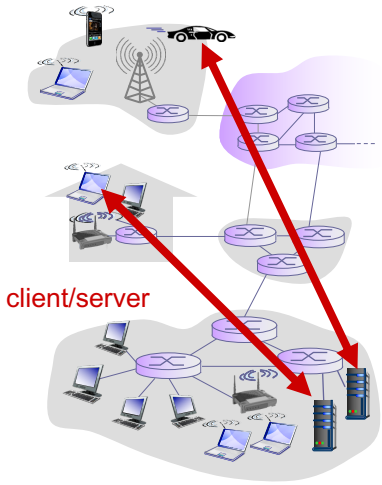
Connecting a Network App

- 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
  - Applications on end systems allows for rapid app development, propagation
- Possible structure of applications
  - Client-Server
  - Peer-to-Peer (P2P)

3

4

Client-Server Architecture



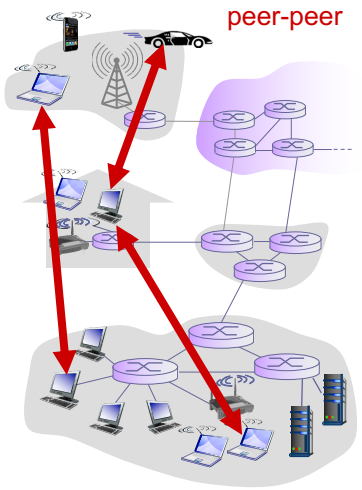
- Server
  - Always-on host
  - Data centers for scaling
- Clients
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP addresses

4

5

P2P Architecture

- *No* always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
- Peers are intermittently connected and change IP addresses
  - Complex management



5

6

Processes

- Process: program running within a host
- Processes in different hosts communicate by exchanging *messages*

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

- Applications with P2P architectures have client processes & server processes

6

7

What Transport Services Does an App Need?

Data Loss

- Some apps (e.g., file transfer, web transactions) require 100% reliable data transfer

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

Timing

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

7

8

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

8

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

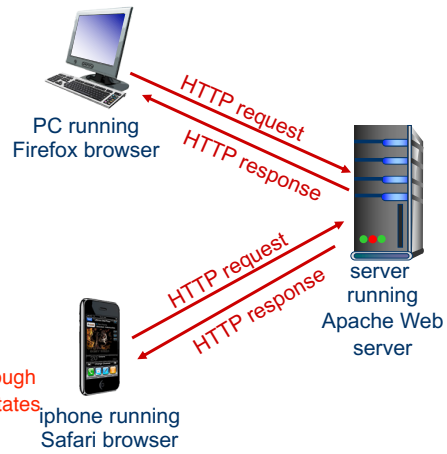
`www.someschool.edu/someDept/pic.gif`

host name | path name

9

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" Technically no though bc cookies save states
  - Server maintains no information about past client requests - why?



10

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
    - Q: What is the advantage?

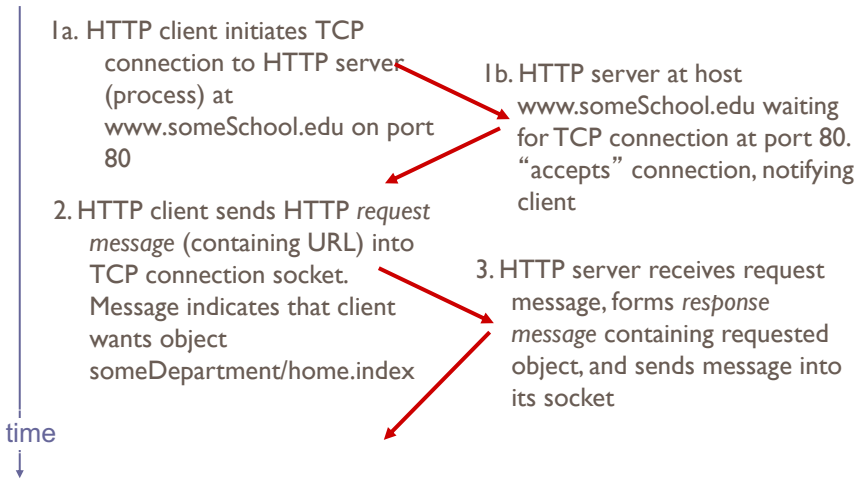
      Less overhead
- Default mode of HTTP
  - Persistent connections with pipelining. (But in HTTP 2.0 uses multiplexing)

11

Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



12

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

time

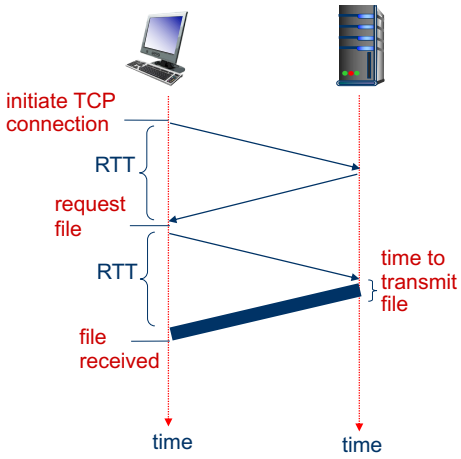
6. Steps 1-5 repeated for each of 10 jpeg objects

Non-persistent HTTP: Response Time

RTT - round trip 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? 2 RTTs because all objects are requested at once

15

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

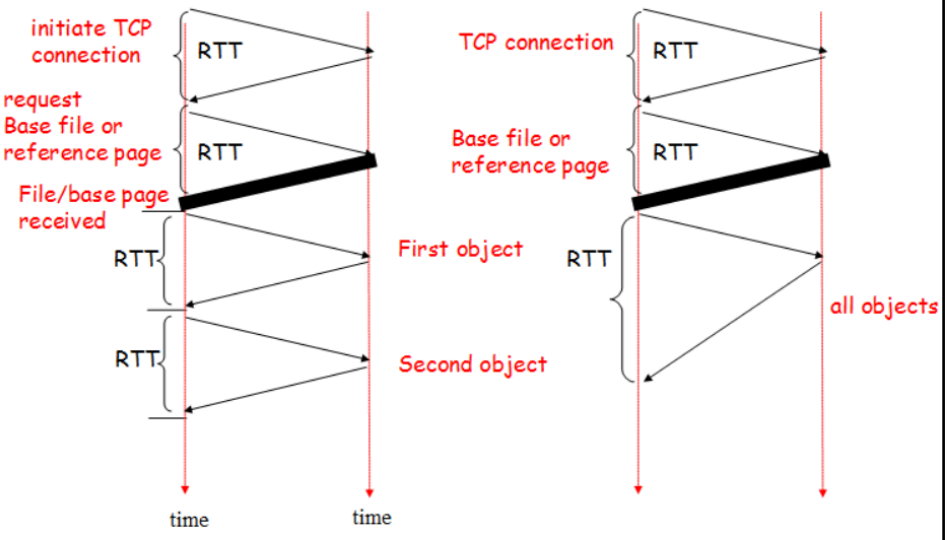
- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server sent over open connection
- Client sends requests as soon as it encounters a referenced object

  As little as 1 RTT for all referenced objects

15

16

Persistent: Pipelined/Non-pipelined Connections



Persistent without pipelining

Persistent with pipelining

16

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)

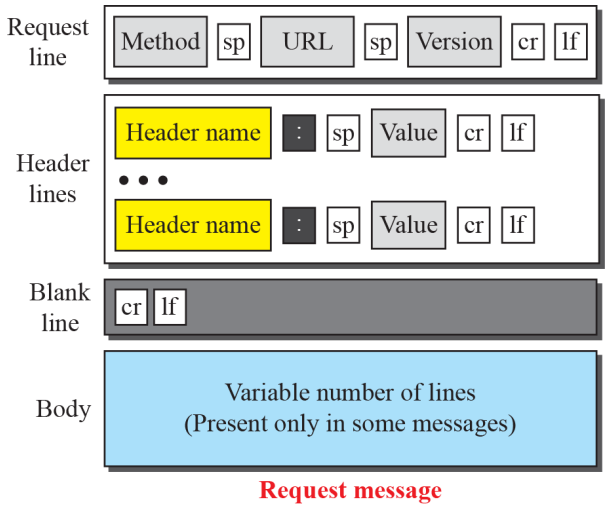
header lines

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
```

17

HTTP Request Msg Format



Request message

18

19

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
    - 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?

19

20

HTTP Response Message

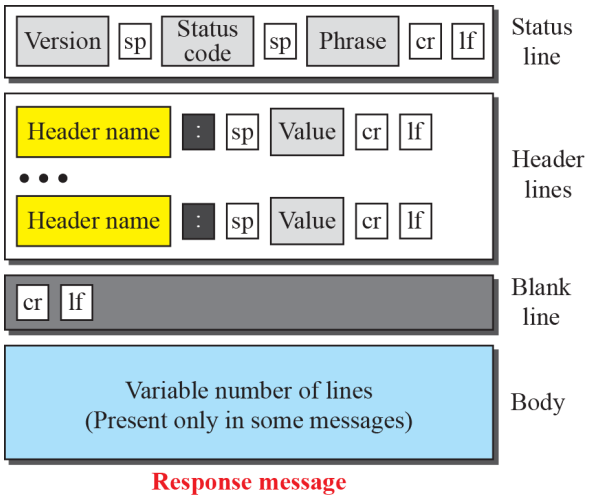
header lines

```
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 ...
```

data, e.g., requested HTML file

20

HTTP Response Msg Format



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

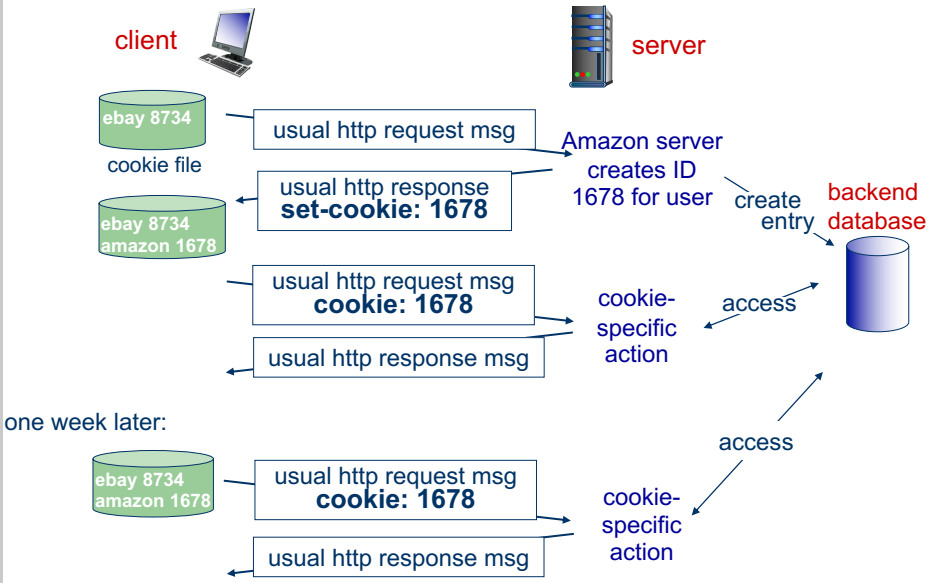
Cookies

- What can cookies be used for?
  - Authorisation
  - Shopping carts
  - Recommendations
  - User session state (Web Mail)

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

- 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

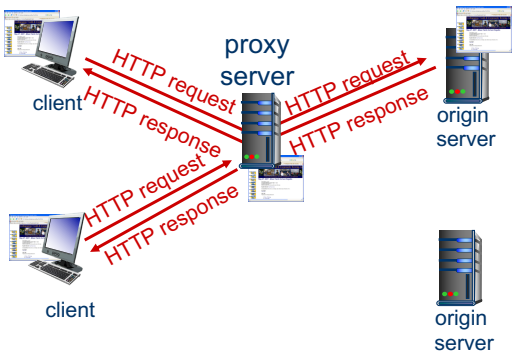
Cookies: Keeping State



Web Caches (Proxy Server)

- Goal: satisfy client request without involving origin server
  - User sets browser to access 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



25

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

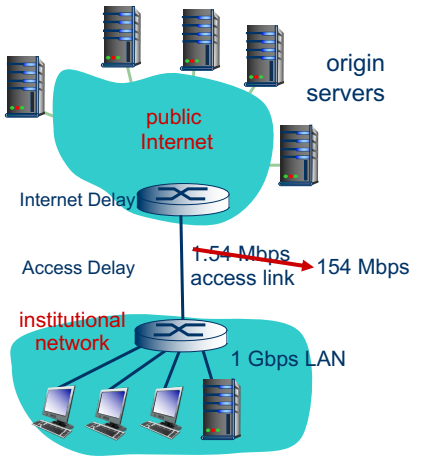
26

27

Caching Example: Fatter Access Link

usage of >80% will incur huge delays

- 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%
    - $1.5 * 10^6/1.54 * 10^6$
  - Total Delay = Internet Delay + Access Delay + LAN Delay
  - = 2 sec + ~~minutes + usecs~~ → msecs



Cost: increased access link speeed is not cheap!

Access link upgrades are periodically required, college obviously has a fatter link

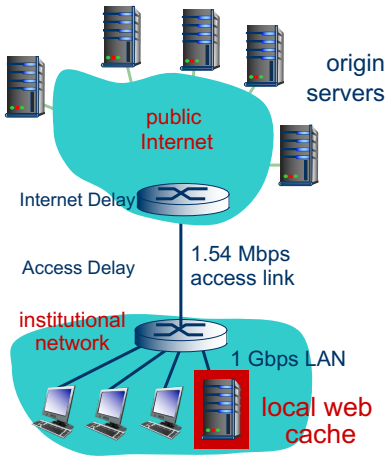
27

28

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!)

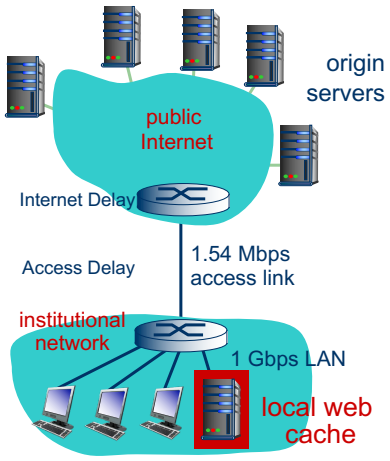
28

29

Caching Example: Install Local Cache II

Calculating access link utilization, delay with cache

- Suppose cache hit rate is 0.4
  - 40% satisfied at cache
  - 60% 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 (0.01) (~msec)
  - = ~ 1.2 secs
  - Less than with 154 Mbps link (and cheaper too!)



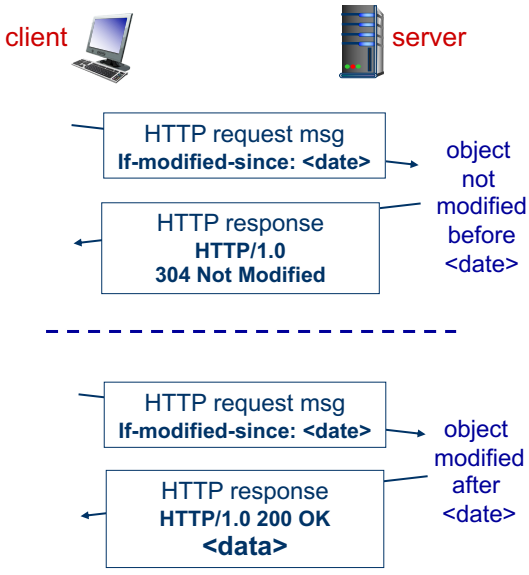
29

30

Conditional GET

- Goal: Do not send object if cache has up-to-date cached version
  - No object transmission delay
    - Lower link utilisation
- 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

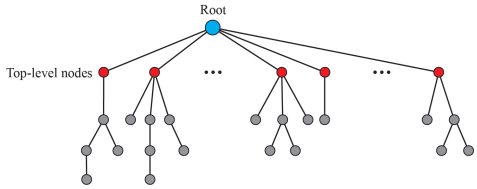


30

31

Domain Name System (DNS)

- Distributed database implemented in hierarchy of many name servers

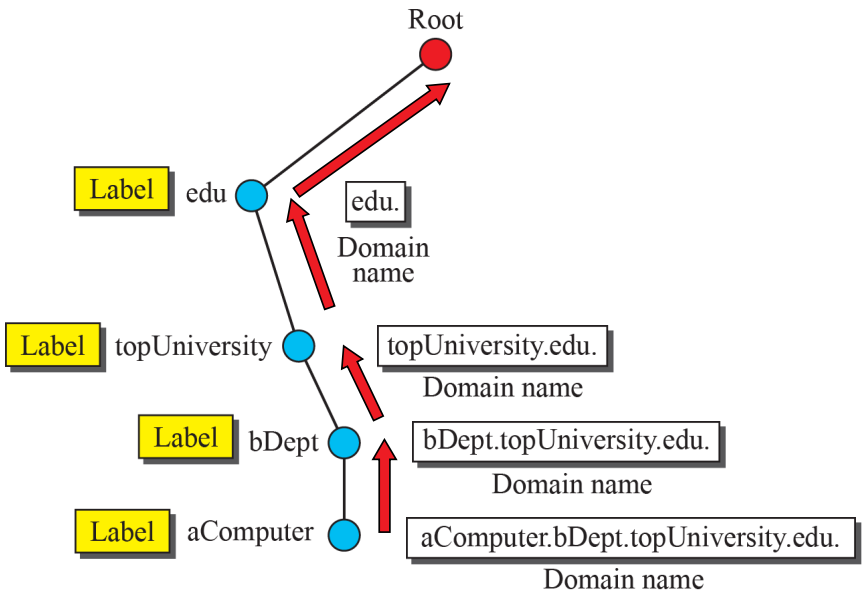


- Application-layer protocol
  - Hosts, name servers communicate to resolve names (address/name translation)
- DNS services
  - Host aliasing
    - Canonical names
  - Mail server aliasing
  - Load distribution e.g. replicated Web servers
    - Many IP addresses correspond to one name

31

32

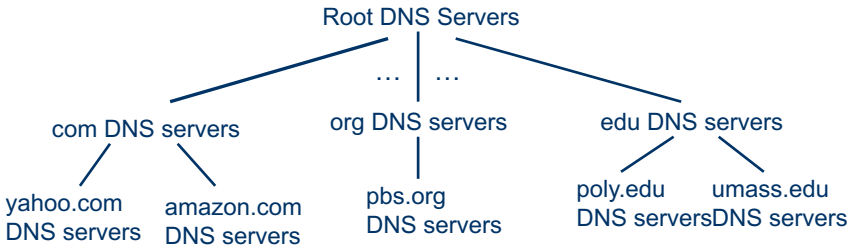
Domain Name and Labels



32

33

DNS: A Distributed Hierarchical Database



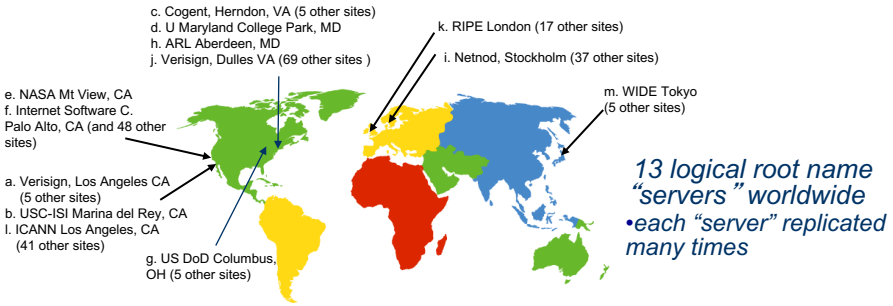
- Client wants IP for www.amazon.com
- Client queries .com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com

33

34

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



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

34

35

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
    - Educause for .edu TLD
- Authoritative DNS Servers
  - Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts

35

36

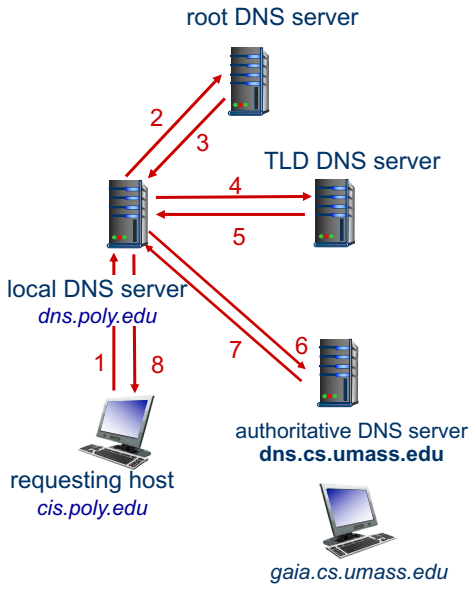
Local DNS

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
- When host makes DNS query, query is sent to its local DNS server
  - Acts as proxy, forwards query into hierarchy

36

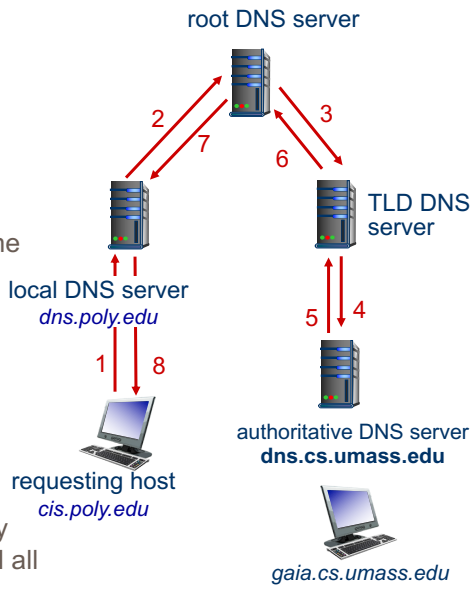
DNS Name Resolution Example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
- Iterated Query
  - Contacted server replies with name of server to contact



DNS Name Resolution & Caching

- Recursive query
- 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
- Cached entries may be out-of-date
  - If host changes IP address, may not be known Internet-wide until all TTLs expire



39

DNS Records

- Distributed database storing resource records (RR)

RR format: `(Name, TTL, Type, Data)`

- A: IPv4 address record

- NS: Name Server record
- CNAME: Canonical Name (i.e. alias) record
- MX: Mail Exchange record

39

40

A & AAAA Records

- A Record - IPv4 address record

```
www.ripe.net. IN A 193.0.6.139
```

- AAAA Record – IPv6 address record

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

40

41

NS Record

- Name is domain (e.g. foo.com)

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

41

42

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

42

MX Record

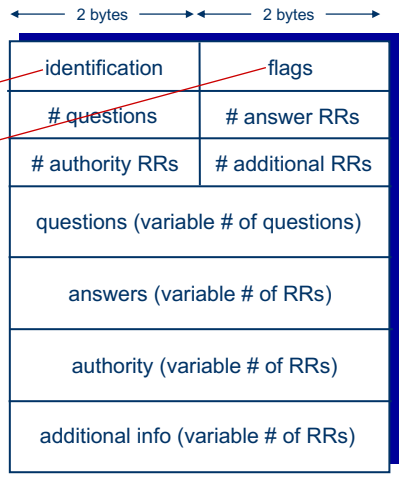
- Data is name of mail server associated with Name

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.	

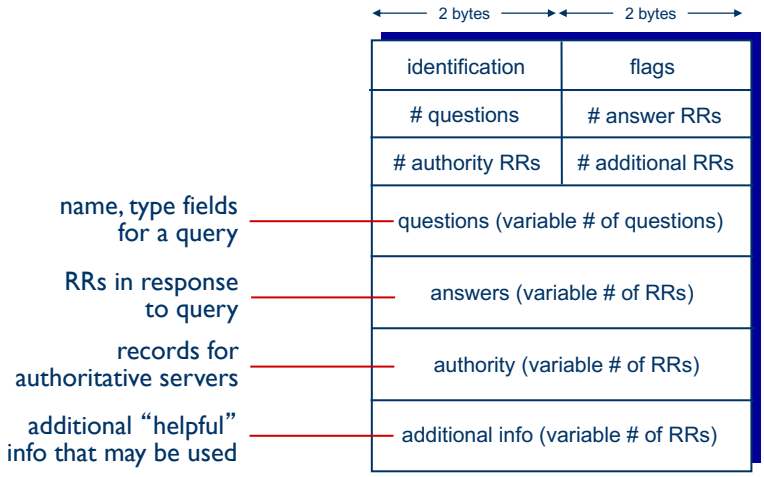
DNS protocol, messages

- *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)
- Create
  - Authoritative server type A record for www.networkuptopia.com

Attacking DNS

DDoS Attacks

- Bombard root servers with traffic
  - Not successful to date

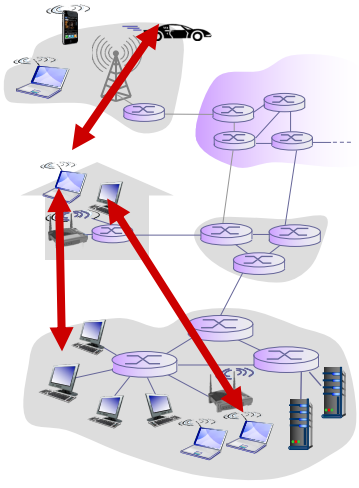
Redirect Attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning

47

Pure P2P Architecture

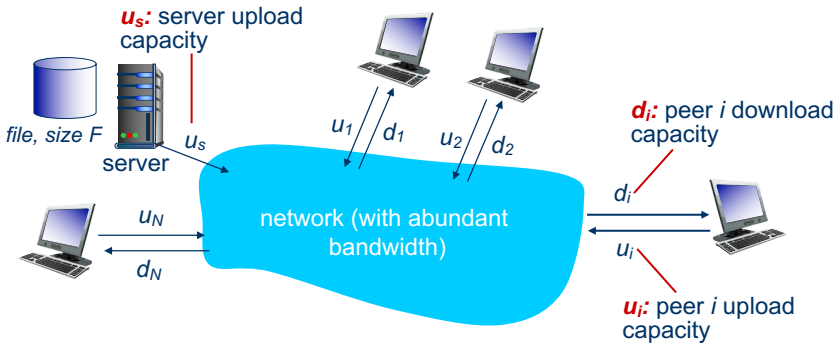
- No always-on server
- Arbitrary end systems directly communicate
- Examples
  - File distribution (BitTorrent)
  - Streaming (KanKan)
  - VoIP (Skype)



48

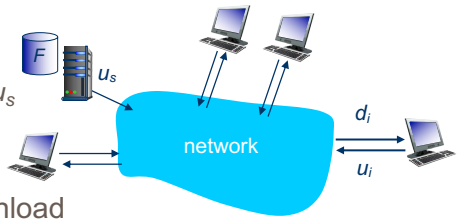
File Distribution – Client-Server vs P2P

- Question: How much time to distribute file (size *F*) from one server to *N* peers?



File Distribution Time: Client-Server

- Server: must sequentially send (upload) *N* file copies
  - –
  - Time to send *N* copies: NF/u_s



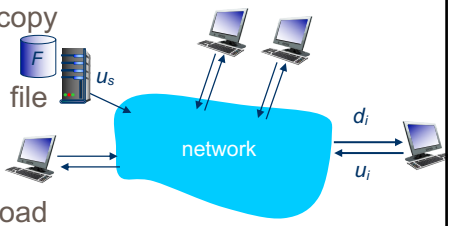
- Client: each client must download file copy
  - d_{min} = min client download rate
  -

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: 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

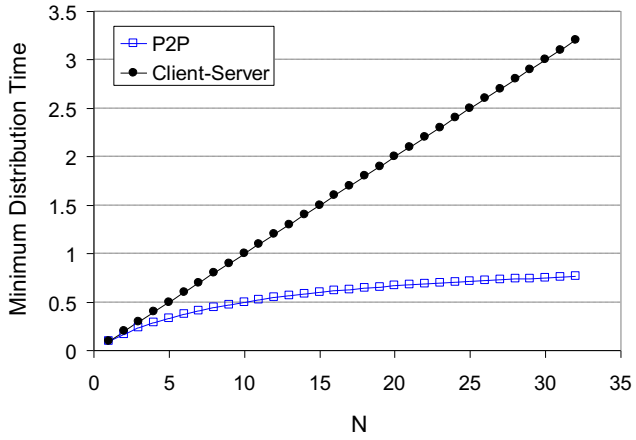
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in N …

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

Client-Server vs P2P Example

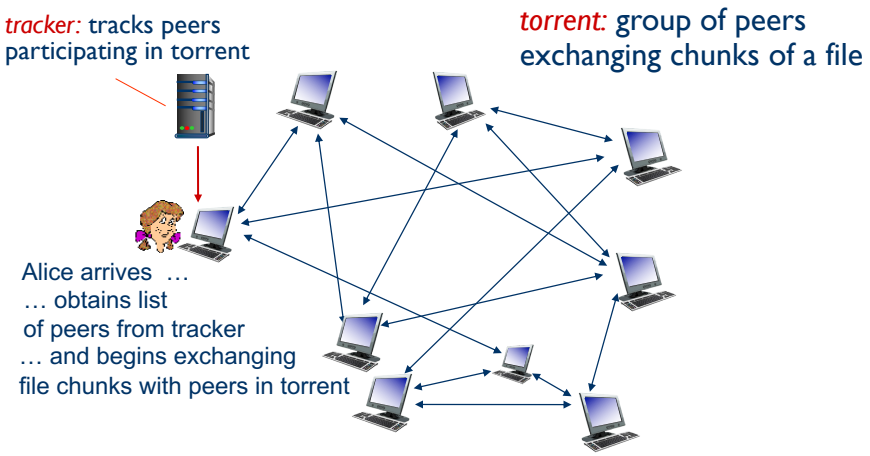
client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



53

P2P File Distribution: BitTorrent

- File divided into 256KB chunks

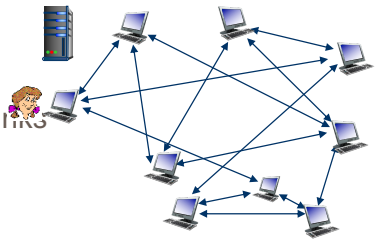


53

54

P2P File Distribution: BitTorrent II

- Peer joining torrent
  - Has no chunks, but will accumulate them over time from other peers



- While downloading, peer uploads chunks to other peers
- Peer may change peers with whom it exchanges chunks
- Once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

54

55

BitTorrent: Requesting, Sending File Chunks

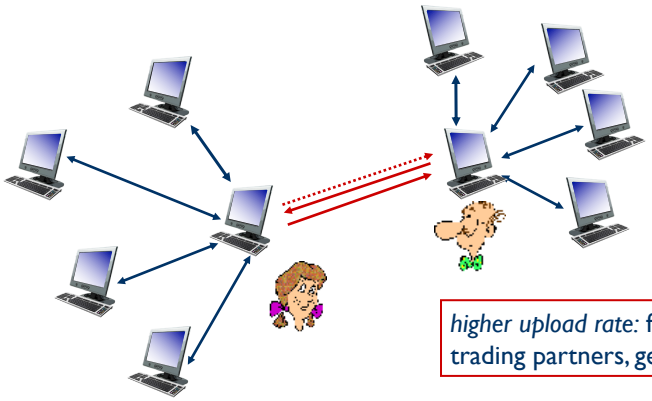
- Requesting chunks
  - Periodically, Alice asks each peer for list of chunks that they have
  - Alice requests missing chunks from peers
- Sending chunks: tit-for-tat
  - Alice sends chunks to those four peers currently sending her chunks at highest rate
    - Other peers are choked by Alice
- Every 30 secs: randomly select another peer, starts sending chunks
  - "Optimistically unchoke" this peer

55

56

BitTorrent: Tit-for-Tat

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



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

56

57

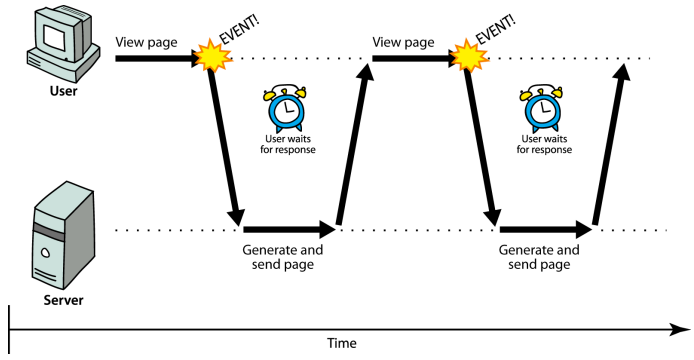
Web Applications

- 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
- RIA performance
  - Comes from Ajax which uses client-side scripting to make web applications more responsive

57

58

Synchronous Web Communication



- Synchronous: user must wait while new pages load
- While a synchronous request is being processed on the server, the user cannot interact with the client web browser

58

Traditional Web Applications

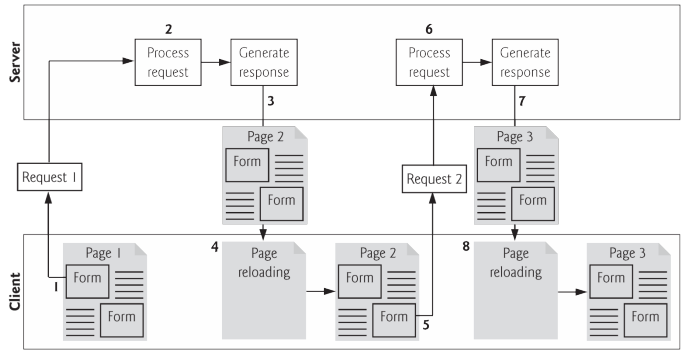
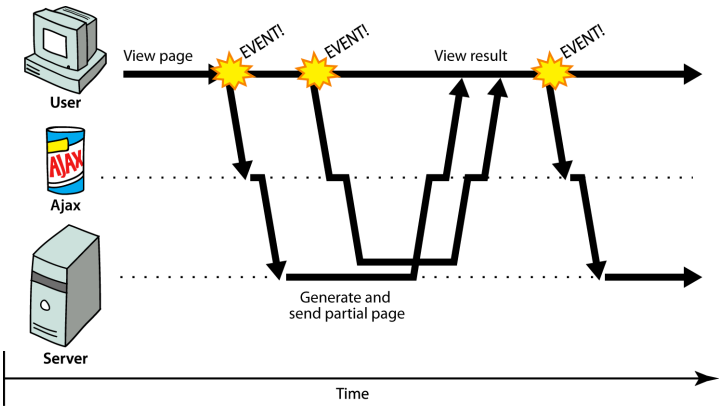


Fig. 16.1 | Classic web application reloading the page for every user interaction.

Web Applications and Ajax

- Web Application: a dynamic web site that mimics the feel of a desktop app
  - Examples: Gmail, Google Maps, Google Docs, Flickr
- Ajax: *Asynchronous XML and JavaScript*
  - Not a programming language; a particular way of using JavaScript
  - 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

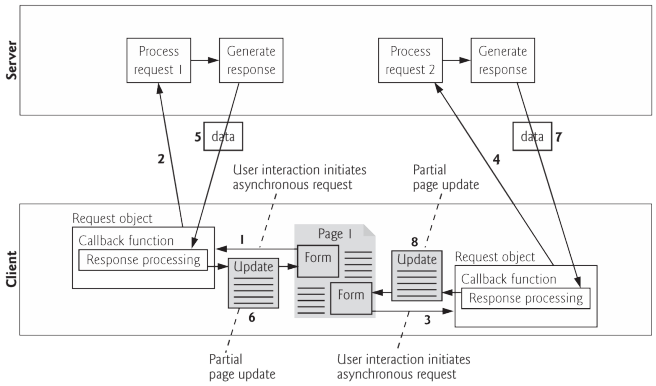
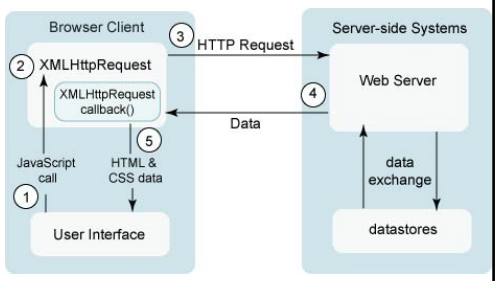


Fig. 16.2 | Ajax-enabled web application interacting with the server asynchronously.

Typical Ajax Request II

1. User clicks, invoking an event handler
2. XHR object requests page from server
3. Server retrieves appropriate data, sends it back
4. XHR object fires an event when data arrives
   - Often called a *callback*
5. Callback event handler processes the data and displays it



63

XMLHttpRequest (XHR) Object

- XMLHttpRequest object
  - 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

64

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.

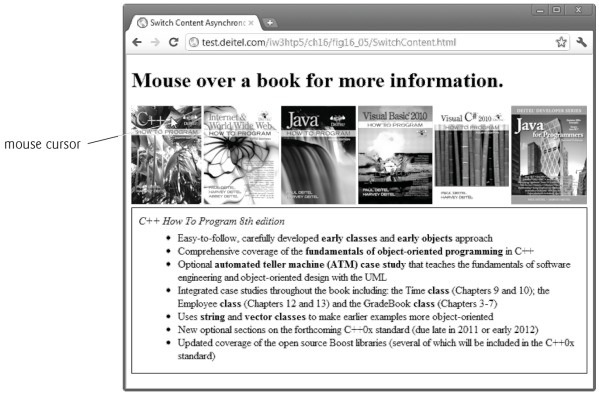


Fig. 16.5 | Asynchronously display content without reloading the page. (Part 6 of 7.)

65

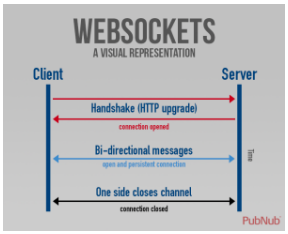
The WebSocket Protocol

- AJAX always has to poll the server for data rather than receive it via push from the server
- 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

66

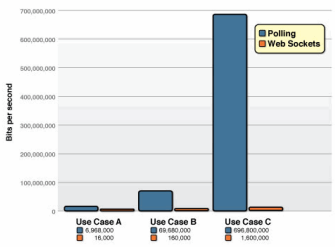
How WebSockets Work

- WebSocket specification defines an API establishing a two-way "socket" connections between a web browser and server
  - Persistent connection between the client and server
- 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

WebSocket Efficiency



- With WebSockets you can transfer as much data as you like in both directions simultaneously
- 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)
- Using this frame-based messaging system helps to reduce the amount of non-payload data that is transferred