Lecture 11: Overview of WWW Technologies
(Part II)

HTTP; Recasting C-S as a global repository;
Scalability: caching, load-balancing.

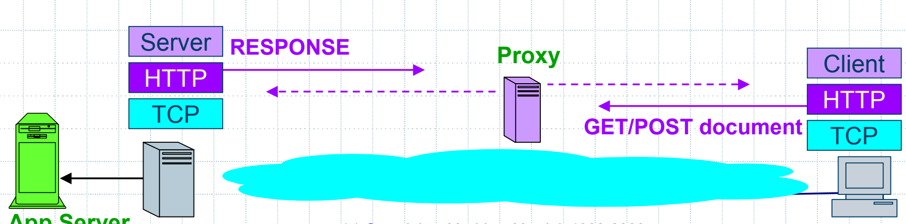
Prof. Yechiam Yemini (YY)
Computer Science Department
Columbia University

Confluence of Two Themes

I. *Recast computing* as tagged-data interpretation

II. *Recast client-server interactions* as global file access

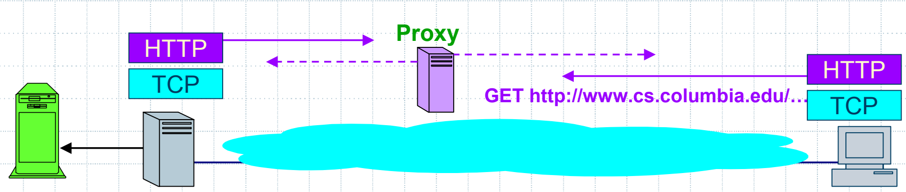
- URL= a global naming scheme (directory structure) for a file-repository
  - Repository access: GET/POST…;URL ("file") can hide processing functions
- A network of servers, proxies, load-balancers provides scalable file access



HTTP Overview

HTTP Functions (RFC 2068)

- Transport objects between client-server
- Manage client-server sessions
- Support traffic management: caching;redirection…



HTTP By Example

```
TELNET
>Open www.cs.columbia.edu 80
…….
GET /~yemini/index.html  HTTP/1.1
Host: www.cs.columbia.edu
Accept: text/html
```

Telnet to port 80

An ASCII protocol

GET request

Mail-like headers

Client capabilities

Completion code

```
HTTP/1.1 200 OK
Date: Wed, 26 Feb 2003 03:01:03 GMT
Server: Apache/1.3.12 (Unix) mod_ssl/2.6.6
OpenSSL/0.9.5a
Last-Modified: Thu, 31 Oct 2002 16:54:42 GMT
ETag: "a2204-1089-3dc16052"
Accept-Ranges: bytes
Content-Length: 4233
Content-Type: text/html
X-Pad: avoid browser bug

<html>
…
</html>
```

Response

Response headers

Metadata

Entity identifier

Payload document

Another Example

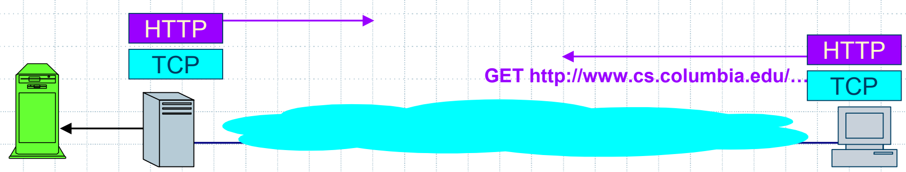
```
GET /dcc/netscript HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html,image/gif,image/jpeg
```

```
HTTP/1.1 301 Moved Permanently
Connection: close
Date: Mon, 06 Mar 2000 04:08:21 GMT Server:
Apache/1.3.6
Last-Modified: Sun, 09 May 1999 10:23:12 GMT
Content-Length: 17832
Content-Type: text/html
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
2.0//EN">
<HTML><HEAD> <TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>…..</BODY></HTML>
```

Completion code

Fundamentals

- Protocol Data Unit structure
  - ASCII protocol
  - MIME-based headers
  - Headers separated by <CRLF>
  - Payload separated by <CRLF><CRLF>
- Primitive Methods
  - GET <URI>: Retrieve document at URI (Uniform Resource Identifier)
  - HEAD: Retrieve just the response header
  - POST: Pass entity to be server-processed using URI (e.g., form, email)
  - PUT: Create/modify document at URI
  - DELETE: document at URI
  - TRACE: loop-back diagnostic test



Headers

- Request headers

User-Agent	specifies the browser version
Accept	specifies browser capabilities
Referer	tracks source system
From	contains email address of user
Authorization	username and password
If-Modified-Since	retrieve document only if newer
Cache-request-directive	client controls caching proxies
Cookie	reports a cookie value

- Response headers

Server	identifies server
Content-length	document size in bytes
Content-type	file type (e.g., html, gif, pdf)
Last-modified	GMT when document last changed
Expires	TTL for caching
Etag	entity tag (used for state synchronization)
Content-MD5	assure entity integrity through message-digest
Location	redirect the client to URI
Cache-response-directive	server controls caching
Set cookie	deposits a cookie at client

Status Codes

- 1xx: Informational –request received, being processed
- 2xx: Success –request received and processed
- 3xx: Redirection – action required to complete request
- 4xx: Client error – bad syntax or cannot be fulfilled
- 5xx: Server error – server failed to handle valid request

Status-Code =
"100" ; Continue | "101" ; Switching Protocols |

"200" ; OK | "201" ; Created | "202" ; Accepted | "203" ; Non-Authoritative Information | "204" ; No Content |
"205" ; Reset Content | "206" ; Partial Content |

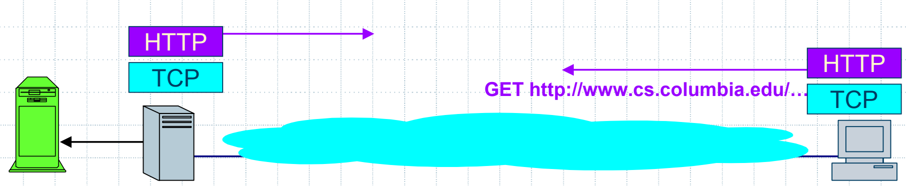
"300" ; Multiple Choices | "301" ; Moved Permanently | "302" ; Moved Temporarily | "303" ; See Other |
"304" ; Not Modified | "305" ; Use Proxy |

"400" ; Bad Request | "401" ; Unauthorized | "402" ; Payment Required | "403" ; Forbidden | "404" ; Not Found |
"405" ; Method Not Allowed | "406" ; Not Acceptable | "407" ; Proxy Authentication Required | "408" ; Request Time-out |
"409" ; Conflict | "410" ; Gone | "411" ; Length Required | "412" ; Precondition Failed | "413" ; Request Entity Too Large |
"414" ; Request-URI Too Large | "415" ; Unsupported Media Type |

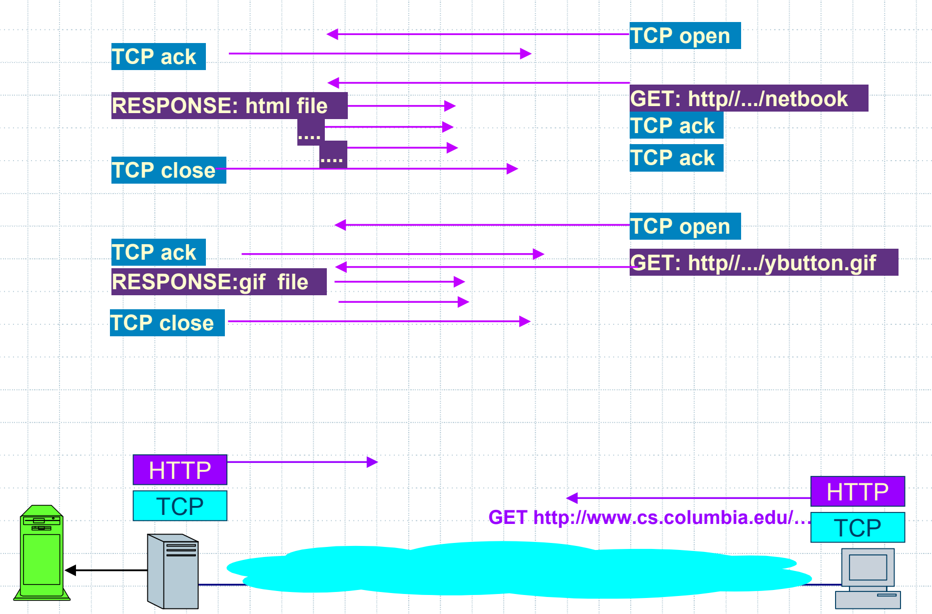
"500" ; Internal Server Error | "501" ; Not Implemented | "502" ; Bad Gateway | "503" ; Service Unavailable |
"504" ; Gateway Time-out | "505" ; HTTP Version not supported

Transport Models

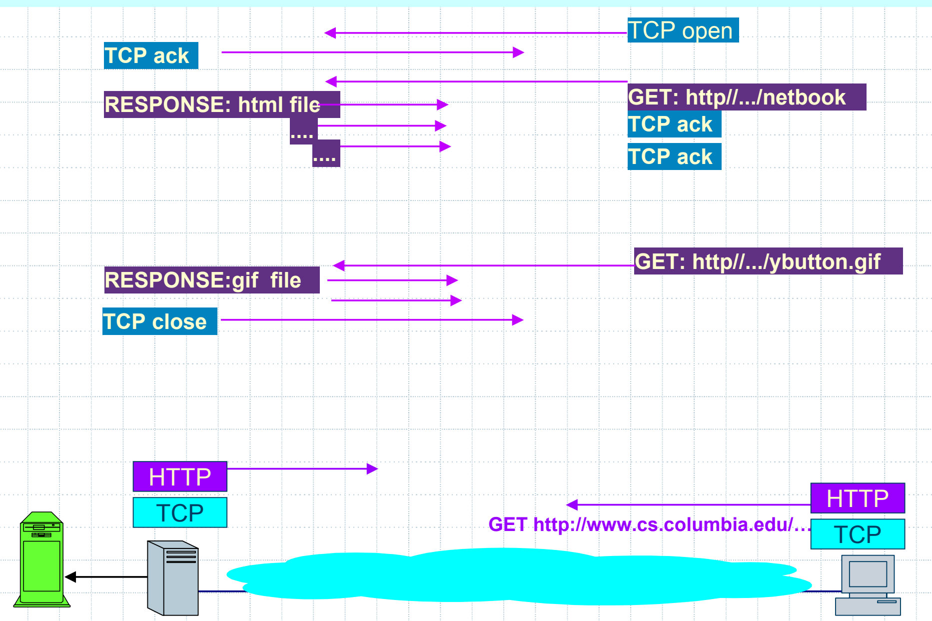
- First, a note about TCP
  - Provides reliable stream transport
  - Assume symmetric duplex transport
  - Window flow control mechanisms control congestion
  - TCP interactions with application protocols can lead to performance problems
- HTTP 1.0: Non-persistent connections
  - Sequential vs. parallel retrievals
- HTTP 1.1: Persistent connections



Sample HTTP 1.0 Non-Persistent Session

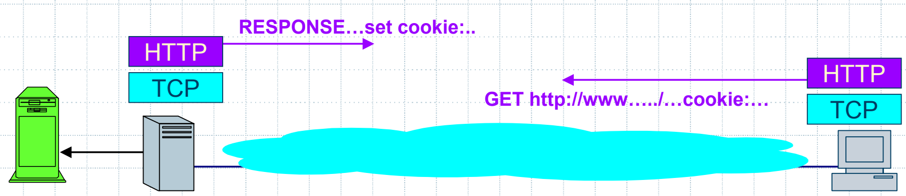


Persistent Connections



State Management With Cookies (RFC2109)

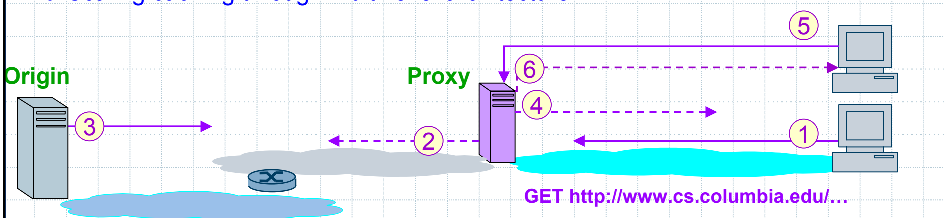
- Cookies provide persistent client state
  - Identify the client and/or state of a transaction
  - E.g., subscriber login; shopping cart
- Server: sets a cookie at client to identify "session" state
  - Cookie is bound to a URL; dispatched with all requests to URL
- Client: reports cookie with requests to URL



Managing scalability: caching

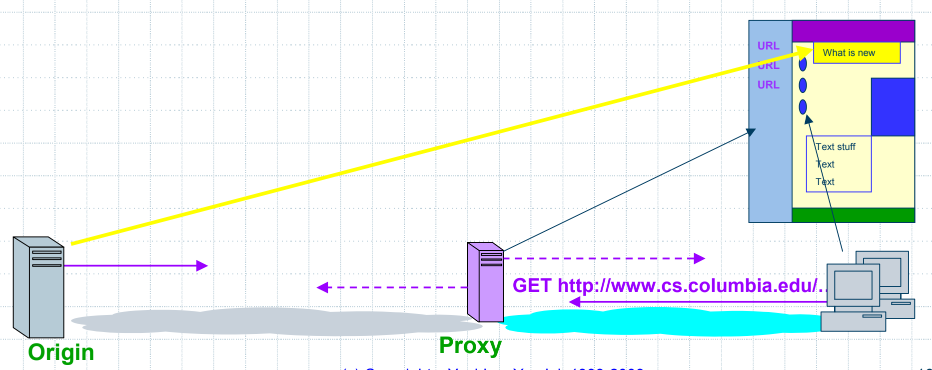
Caching

- Basics:
  - Web generates most Internet traffic; most of it is GET;
  - Access has locality property; most page components remain static for long periods
- Goals:
  - Improving bandwidth utilization over MAN/WAN links
  - Accelerating response time by reducing delay-distance
- Issues:
  - HTTP support of caching
  - Managing cache coherence
  - Scaling caching through multi-level architecture



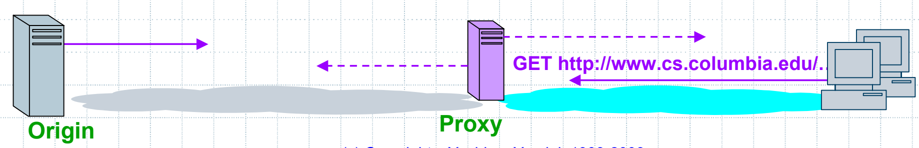
How Does It Work

- DOC is assembled from dynamic and static components
- Static components are cached at proxies and client
- Dynamic components are delivered on demand



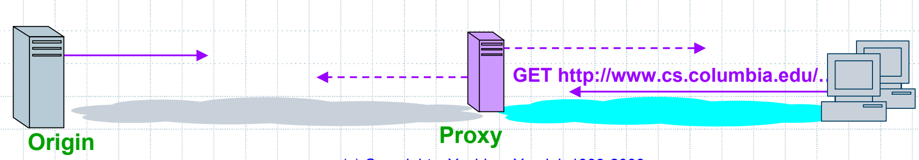
Architecture & Operations of Caching

- Updating caches
  - Validating and refreshing
  - Handling varying information dynamics; control entity granularity
- HTTP support
  - HEAD request is used for validation
  - Conditional requests: If-Modified-Since; if-not-modified-since…
  - Tagging entities for validation (Etag)
  - Client control of caching:
    - Cache-control requests: min-fresh, max-stale, no-transform
  - Server control of caching
    - Cache-control responses: must-revalidate, public, private, no-cache
    - Expires, max-age…
  - Redirection: application-level URL-based routing



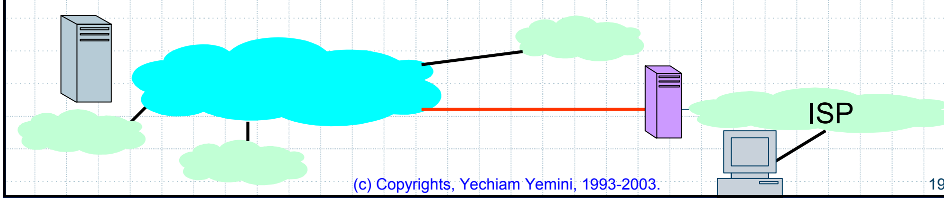
Additional issues

- Managing large caches (how to index pages)
- Managing a hierarchy/mesh of proxies
  - How do proxies coordinate the state of their caches
  - Cache management protocol (ICP)
  - Replicating the metadata (indexing of pages)
- Interaction of caching with dynamic pages
  - Distributed assembly of objects ➔ proxy-side computing
- Interaction of caching with services
  - Access control and accounting is of great importance to providers
  - Proxies hide information and relax controls



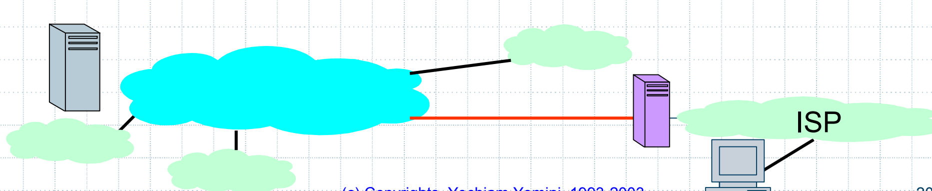
Rationale: Bandwidth Saving

- How much bandwidth is saved?
  - Assume 70% of objects are static and no validation (optimistic)
  - 70% saving of WAN/MAN link; e.g., reduce 1Gbps flow to 0.3Gbps
- Is "Saving" always good?



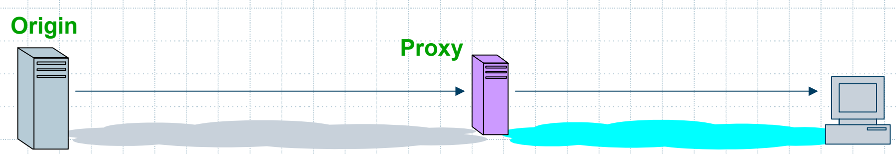
Rationale: Bandwidth Saving

- Bandwidth vs. caching
  - ISP cost of MAN link: 1Mbps ~ $1 (2003)
  - The cost of CPU cycles is much higher than the cost of MAN bandwidth
    - Why? ➔ CPUs require space, electrical, cooling…
      ➔ managing CPUs is complex and costly
  - 1 Mbps ~ 25 URL/sec (assume URL ~5KB) ➔ 0.1-0.5 CPU
    - Total Cost of Ownership (TCO) of a 24x7 CPU can range in the $100-$1000
    - Cost of caching: $10-$500 per 1Mbps/month >> cost of MAN links
  - For WAN links cost of bandwidth may be potentially larger than caching



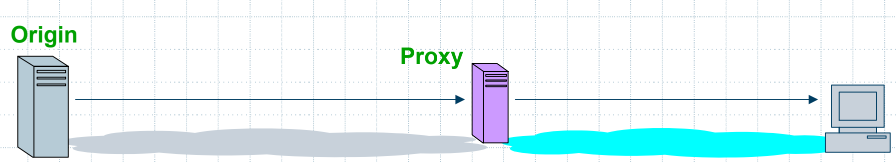
Rationale: Accelerating Response Time

- Assume:
  - o s= server response time; c=cache response time; r=s/c; h=hit rate.
- Speed-up: s/(c*h+s*(1-h)) =r/(h+r*(1-h))
  - o E.g., h=70%, r=6 ➔ speed-up =6/(0.7+6*0.3)=2.4
  - o E.g., h=50% r=2 ➔speed-up= 2/(0.5+1)=1.3
  - o Validation time may transform speed-up to slow-down (why?)
  - o Link delays have limited contribution to response time
- Conclusions
  - o Caching can improve bandwidth usage and response time
  - o But the gains must be carefully evaluated



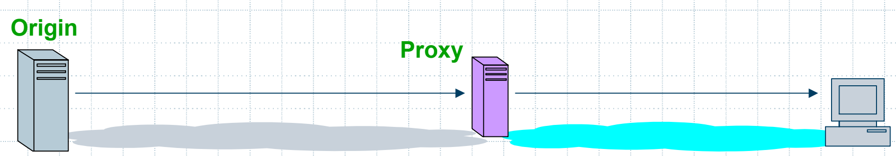
More Caching Examples

- Rationale: BW-saving ~h*BW ; speed-up ~ r/(h+r(1-h))
- Large scale content distribution (movies, music)
  - o h ~ 1; BW-saving ~ (request rate)x(object-size)
  - o Speed up ~ r
- Server-side: database caching
  - o Cycle bandwidth savings ~ (rate of a query)x(# cycles needed)
  - o Speed up ~ r = computing-time/retrieval-time



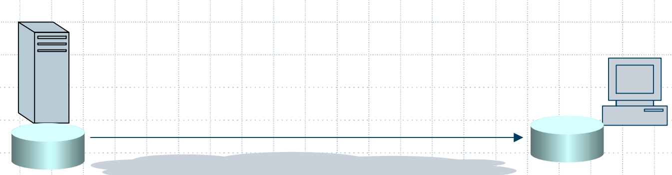
What Should Be Cached?

- Intuitively:
  - o if the page does not change
  - o If demand remains high
- "Working set" the set of frequently accessed pages
- Caching: rate-demand/rate-change>>1



The StoreWidth-Abundance Limit

- In the limit: r >>1
  - o The client can access all cacheable info from its storage
  - o Time to access server is much larger than time to access cache
- In the limit: the working set is nearly all data
  - o Exception: real-time data (changes and needed in real-time)
- The rise of client-storage networks
  - o The function of the network is to multicast content to storage



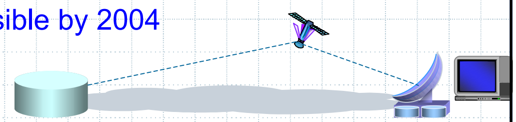
What If All The Bits You Need Were…

- In your pocket

Type	Storage	Daily Downloads	
Lifetime msgs: voice/email	20GB	10 voice mail, 200 email	0.005GB
Personal files/Photos	5GB		0.005GB
Entertainment: 200CD, 5DVD, 10hrTV	35GB	20 CDs, 1DVD, 2hrTV	7GB
Other: SW, downloads	20Gb		1GB
Total	80GB		8GB

  - ~Consumer priced 30GB feasible by 2004

- At your home



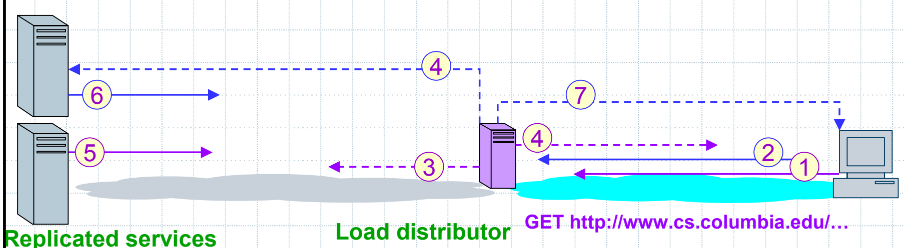
Type	Storage (TB)	Daily Replenishment (GB)	
Lifetime Personal Files	0.03		0.01
Video/photos	1		0.0001
Music Store (10,000 CDs; MP3)	0.4	1000 CDs	50
Video Rental (1000 DVD movies)	4	20 new releases	80
TV (100 channels; 800hrs MPEG)	1	400hrs	250
Internet sites push (1000 sites x2GB)	2	20MB per site	20
Total	9TB		400GB

  - ~Consumer priced 1TB Digital Video Recorded (DVR) by 2005

Managing scalability: load-balancing

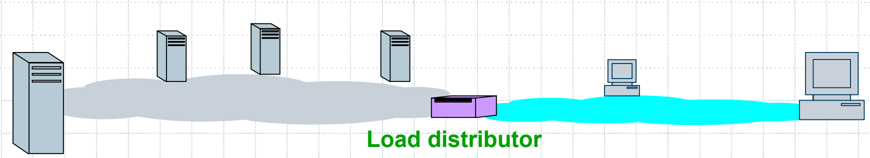
Load Balancing

- How to admit scalable growth in service demand
  - How to distribute load among replicated servers
  - How to direct traffic to optimize load distribution
  - How to manage replication of services
- Duality
  - Caching: sharing server access among replicated clients
  - Load balancing: sharing client access among replicated servers



How To Direct Client Requests?

- What layer is in charge of directing traffic?
  - L7 routing through DNS round-robin
  - L7 routing through HTTP redirection
  - L4/7 routing based on URL
  - L2/4, or L3/4 routing based on a cluster load-balancer
- How should the “best” server be selected?
  - Randomly
  - Lowest server load: monitoring updating load statistics
  - Best client response: by location, round-trip delay…



Concluding Notes

- Web service architecture is continuing to evolve rapidly
- Focusing on server-side organization of services
- HTML and HTTP are insufficient
  - Tag-computing has been generalized from HTML to XML
  - HTTP client-server interactions have been generalized to XML messaging (SOAP)
- Replication & caching are resurfacing on server-side
  - Cluster replication
  - Storage network architecture
- Storage networks could result in sweeping new paradigms