Topic #16
Today’s Lecture Objectives

- HW#3 questions?
- DNS
- HTTP
DNS Functions

• IP and host name address translation
  – IP address to host
  – Host to IP address

• Host aliasing
  – Multiple host names per IP address

• Mail server aliasing
  – Specific function for Port 25
  – Mail type “MX”

• Load distribution
  – Host to IP address returns multiple IP addresses
**DNS Misc**

- Truly distributed
  - Avoid problems of centralization (which are?)

- UDP-based
  - In *most* situations
  - Uses Port 53

- Types of servers
  - Local: within and serves an admin domain
  - Root: highest level of DNS “tree”
    - May point to “Top Level Domain” (TLD) servers (based on suffix, e.g., .com, .edu, .uk, etc.)
  - Authoritative: *the* server that maintains a particular DNS record
DNS: Root Name Servers

- Contacted when local name server cannot resolve name
- Root server may then contact authoritative name server if name mapping not known

13 root name servers worldwide
TLD and Authoritative Servers

• Root servers
  – May contact top-level domains for help

• Top-Level Domain (TLD) servers
  – Responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
    • Ex: Network solutions maintains servers for .com TLD
    • Ex: Educause for .edu TLD

• Authoritative DNS servers
  – Organization’s DNS servers, providing authoritative hostname to IP mappings for organization’s servers (e.g., Web and mail).
  – Can be maintained by organization or service provider
Local Name Server

- Does not strictly belong to hierarchy

- Each ISP (residential ISP, company, university) has one or more
  - Also called “default name server”

- When a host makes a DNS query, query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy
Distributed, Hierarchical Database

Client wants IP for www.amazon.com; 1st approx:

• Client queries a 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
Two Ways to do Queries

- Iterative
- Recursive

- Can also do a combination of the two in a kind of “hybrid” approach
Iterative Queries

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

- Two ways to do queries...
Recursive Queries

**recursive query:**
- puts burden of name resolution on contacted server
- heavy load?

**iterative query:**
- contacted server replies with name of server to contact
- “I don’t know, but ask this server”
DNS: Caching and Updating Records

• Once name servers learn mapping, it *caches* mapping
  – cache entries timeout (disappear) after some time
  – TLD servers typically cached in local name servers
    • Thus root name servers not often visited

• Update/notify mechanisms under design by IETF
  – RFC 2136
DNS records

DNS: stores resource records (RR)

RR format: (name, value, type, ttl)

• Type=A
  - name is hostname
  - value is IP address

• Type=NS
  - name is domain (e.g. foo.com)
  - value is IP address of authoritative name server for this domain

• Type=CNAME
  - name is alias for some “cannonical” (real) name
  - www.ibm.com is really servereast.backup2.ibm.com

• Type=MX
  - value is name of mailserver associated with name
DNS Protocol Messages

**DNS protocol**: *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

<table>
<thead>
<tr>
<th>Identification</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of questions</td>
<td>number of answer RRs</td>
</tr>
<tr>
<td>number of authority RRs</td>
<td>number of additional RRs</td>
</tr>
<tr>
<td>questions (variable number of questions)</td>
<td>answers (variable number of resource records)</td>
</tr>
<tr>
<td>authority (variable number of resource records)</td>
<td>additional information (variable number of resource records)</td>
</tr>
</tbody>
</table>
### DNS Protocol Messages

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>identification (question)</td>
<td>number of questions</td>
</tr>
<tr>
<td>flags</td>
<td>number of answer RRs</td>
</tr>
<tr>
<td>number of authority RRs</td>
<td>number of additional RRs</td>
</tr>
</tbody>
</table>

- **Name, type fields** for a query
- **RRs in response to query**
- **records for authoritative servers**
- **additional “helpful” info that may be used**
Inserting Records into DNS

• Example: just created startup “Network Utopia”
• Register name networkuptopia.com at a registrar (e.g., Network Solutions)
  – Need to provide registrar with names and IP addresses of your authoritative Name Server (primary and secondary)
  – Registrar inserts two RRs into the com TLD server:

  (networkutopia.com, dns1.networkutopia.com, NS)
  (dns1.networkutopia.com, 212.212.212.1, A)

• Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkutopia.com
• How do people get the IP address of your Web site?
The Web and HTTP
The Web and HTTP

First some jargon

• Web page consists of objects
  • Object can be HTML file, JPEG image, Java applet, audio file,…
  • Web page consists of base HTML-file which includes several referenced objects
  • Each object is addressable by a URL
  • Example URL:

\[
\text{http://www.cs.ucsb.edu/courses/fall.html}
\]

  \begin{align*}
  \text{host name} & \quad \text{path name}
  \end{align*}
HTTP Overview

- Web’s application layer protocol
- client/server model
  - **client**: browser that requests, receives, “displays” Web objects
  - **server**: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068
HTTP Overview (con’t)

• Uses TCP at the transport layer:
  – Client starts TCP connection (creates socket) to server, port 80
  – Server accepts TCP connection from client
  – HTTP messages between client and server
  – TCP connection closed

• HTTP is “stateless”
  – server maintains no information about past client requests

• Connection “characteristics”
  1. (a) Non-Persistent vs. (b) Persistent
  2. (a) Parallelism for non-persistent connections
  3. (b) Pipelining for persistent connections
HTTP Connections

1a. Non-Persistent HTTP
- At most one object is sent over a TCP connection
- HTTP/1.0 uses nonpersistent HTTP

1b. Persistent HTTP
- Multiple objects can be sent over single TCP connection between client and server
- HTTP/1.1 uses persistent connections in default mode
1a. Non-Persistent HTTP

Suppose user enters URL
www.cs.ucsb.edu/courses/fall.html

1a. HTTP client initiates TCP connection to HTTP server (process) at www.cs.ucsb.edu on port 80

1b. HTTP server at host www.cs.ucsb.edu waiting for TCP connection at port 80. “accepts” connection, notifying client

2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object courses/fall.html

3. HTTP server receives request message, forms response message containing requested object, and sends message into its socket
1a. Non-Persistent HTTP (con’t)

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
1a. HTTP 1.0 (Non-Persistent)
1a. Non-Persistent Connections

• Advantages
  – Means server doesn’t have to keep TCP state around—connections are quickly closed

• Disadvantages
  – Each transfer must go through 3-way handshake
  – Each transfer suffers from TCP’s initial slow rate
  – If multiple simultaneous connections allowed, increases number of connections (and required state)
2a. Adding Parallelism to Non-Persistence

- Allows for multiple outstanding connections to request objects as soon as they are needed
  - Problem: still have 2 RTTs to fetch each object
1b. Persistent Connections

Persistent connection

• Default for http/1.1
• Idea: on the same TCP connection: server, parses request, responds, parses new request,..
• Then client sends requests for all referenced objects as soon as it receives base HTML
• Adv: fewer RTTs
• Disadv: holds resources at the server open

• With Pipelining
  – Client issues request as soon as it encounters a reference
  – Reduces RTT Delay
  – Prevents idle time at server

• Without Pipelining
  – Client issues new request only after previous response received
    • Each object has one RTT
  – Disadvantage: server waits between object requests with open connection
1a. Non-Persistence vs. 1b. Persistence

- Non-persistence
  - Connection request
  - Connection establishment
  - File request
  - Data

- Persistence without pipelining

- Persistence with pipelining
HTTP Request Message

- Types of HTTP messages: request, response
- HTTP request message:
  - ASCII (human-readable format)
    - GET /courses/fall.html HTTP/1.1
    - Host: www.cs.ucsb.edu
    - User-agent: Mozilla/4.0
    - Connection: close
    - Accept-language: fr
  (extra carriage return, line feed)
Request Message General Format

```
method  sp  URL  sp  version  cr  lf
header field name  :  value  cr  lf

…

header field name  :  value  cr  lf
```

Entity Body
HTTP Response Message

status line
(protocol
status code
status phrase)

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ...
Content-Length: 6821
Content-Type: text/html

data data data data data data data ...

data, e.g., requested HTML file

header lines
HTTP Response Status Codes

In first line in server->client response message.

A few sample codes:

200 OK
   – request succeeded, requested object later in this message

301 Moved Permanently
   – requested object moved, new location specified later in this message (Location:)

400 Bad Request
   – request message not understood by server

404 Not Found
   – requested document not found on this server

505 HTTP Version Not Supported
Trying out HTTP (client side) for Yourself

1. Telnet to your favorite Web server:
   
   `telnet www.cs.ucsb.edu 80`

   Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu.
   Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:
   
   `GET / HTTP/1.1`
   `Host: jackson.cs.ucsb.edu`

   By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server
Conditional GET

- **Goal:** don’t send object if cache has up-to-date cached version
- **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
- **server:** response contains object if cached copy is modified:
  - HTTP/1.0 200 OK
  - `<data>`
Many major Web sites use cookies.

**Four components:**

1) cookie header line in the HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user’s host and managed by user’s browser
4) back-end database at Web site

- Susan accesses Internet always from same PC
- Visits the same e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID
Cookies: Keeping “State”

The diagram illustrates the process of cookie management in a client-server interaction. The server creates an ID, 1678, for a user and sets a cookie in the response. The client sends this cookie in subsequent requests, and the server accesses the cookie-specific action. One week later, the process repeats with the same cookie values.
Cookies (continued)

What cookies can bring:
• authorization
• shopping carts
• recommendations
• user session state (Web e-mail)

Cookies and privacy:
• cookies permit sites to learn a lot about you
• you may supply name and e-mail to sites
• search engines use redirection & cookies to learn yet more
• advertising companies obtain info across sites
Extra Slides Specifically About Kurose & Ross
DNS: Domain Name System

People: many identifiers:
   - SSN, name, passport#

Internet hosts, routers:
   - IP address (32 bit) - used for addressing datagrams
   - “name”, e.g.,
     ww.yahoo.com - used by humans

Q: map between IP addresses and name?

Domain Name System:
- distributed database implemented in hierarchy of many name servers
- application-layer protocol host, routers, name servers to communicate to resolve names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”
DNS

DNS services
• Hostname to IP address translation
• Host aliasing
  – Canonical and alias names
• Mail server aliasing
• Load distribution
  – Replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?
• single point of failure
• traffic volume
• distant centralized database
• maintenance
doesn’t scale!
Distributed, Hierarchical Database

Client wants IP for www.amazon.com; 1st approx:

- Client queries a 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:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

13 root name servers worldwide

- a Verisign, Dulles, VA
- c Cogent, Herndon, VA (also Los Angeles)
- d U Maryland College Park, MD
- e NASA Mt View, CA
- f Internet Software C. Palo Alto, CA (and 17 other locations)
- g US DoD Vienna, VA
- h ARL Aberdeen, MD
- i Verisign, ( 11 locations)
- j Verisign, ( 11 locations)
- k RIPE London (also Amsterdam, Frankfurt)
- l ICANN Los Angeles, CA
- m WIDE Tokyo
TLD and Authoritative Servers

• **Top-level domain (TLD) servers**: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  – Network solutions maintains servers for com TLD
  – Educause for edu TLD

• **Authoritative DNS servers**: organization’s DNS servers, providing authoritative hostname to IP mappings for organization’s servers (e.g., Web and mail).
  – Can be maintained by organization or service provider
Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - Also called “default name server”
- When a host makes a DNS query, query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy.
Example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
Recursive queries

**recursive query:**
- puts burden of name resolution on contacted name server
- heavy load?

**iterated query:**
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”
DNS: caching and updating records

• once (any) name server learns mapping, it *caches* mapping
  – cache entries timeout (disappear) after some time
  – TLD servers typically cached in local name servers
    • Thus root name servers not often visited
• update/notify mechanisms under design by IETF
  – RFC 2136
**DNS records**

**DNS:** distributed db storing resource records (RR)

| RR format: | (name, value, type, ttl) |

- **Type=A**
  - name is hostname
  - value is IP address

- **Type=NS**
  - name is domain (e.g. foo.com)
  - value is IP address of authoritative name server for this domain

- **Type=CNAME**
  - name is alias name for some “cannnonical” (the real) name
    - www.ibm.com is really servereast.backup2.ibm.com
  - value is canonical name

- **Type=MX**
  - value is name of mailserver associated with name
DNS protocol, messages

DNS protocol: 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identification</td>
<td>16 bit identification</td>
</tr>
<tr>
<td>flags</td>
<td>Query or reply, recursion desired, recursion available, reply is authoritative</td>
</tr>
<tr>
<td>number of questions</td>
<td>(variable number of questions)</td>
</tr>
<tr>
<td>number of answer RRs</td>
<td>(variable number of resource records)</td>
</tr>
<tr>
<td>number of authority RRs</td>
<td>(variable number of resource records)</td>
</tr>
<tr>
<td>number of additional RRs</td>
<td>(variable number of resource records)</td>
</tr>
</tbody>
</table>
### DNS protocol, messages

<table>
<thead>
<tr>
<th>Name, type fields for a query</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR in response to query</td>
</tr>
<tr>
<td>Records for authoritative servers</td>
</tr>
<tr>
<td>Additional &quot;helpful&quot; info that may be used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: DNS Message Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identification</strong></td>
</tr>
<tr>
<td><strong>Number of Questions</strong></td>
</tr>
<tr>
<td><strong>Number of Authority RRs</strong></td>
</tr>
<tr>
<td><strong>Questions</strong> (variable number of questions)</td>
</tr>
<tr>
<td><strong>Answers</strong> (variable number of resource records)</td>
</tr>
<tr>
<td><strong>Authority</strong> (variable number of resource records)</td>
</tr>
<tr>
<td><strong>Additional Information</strong> (variable number of resource records)</td>
</tr>
</tbody>
</table>
Inserting records into DNS

- Example: just created startup “Network Utopia”
- Register name networkuptopia.com at a registrar (e.g., Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the com TLD server:

  (networkutopia.com, dns1.networkutopia.com, NS)
  (dns1.networkutopia.com, 212.212.212.1, A)

- Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkutopia.com
- How do people get the IP address of your Web site?
Application in Depth: Web and HTTP

First some jargon

• Web page consists of objects
• Object can be HTML file, JPEG image, Java applet, audio file,…
• Web page consists of base HTML-file which includes several referenced objects
• Each object is addressable by a URL
• Example URL:

  www.cs.ucsb.edu/courses/fall.html

  host name                  path name
HTTP overview

HTTP: hypertext transfer protocol
- Web’s application layer protocol
- client/server model
  - **client**: browser that requests, receives, “displays” Web objects
  - **server**: Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068

PC running Netscape

Server running Apache Web server

Mac running Safari
HTTP overview (continued)

Uses TCP at the transport layer:

• client initiates TCP connection (creates socket) to server, port 80
• server accepts TCP connection from client
• HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
• TCP connection closed

HTTP is “stateless”

• server maintains no information about past client requests

Protocols that maintain “state” are complex!

• past history (state) must be maintained
• if server/client crashes, their views of “state” may be inconsistent, must be reconciled
HTTP connections

Nonpersistent HTTP
• At most one object is sent over a TCP connection
• HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP
• Multiple objects can be sent over single TCP connection between client and server.
• HTTP/1.1 uses persistent connections in default mode
Nonpersistent HTTP

Suppose user enters URL

www.cs.ucsb.edu/courses/fall.html

1a. HTTP client initiates TCP connection to HTTP server (process) at www.cs.ucsb.edu on port 80

2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object courses/fall.html

1b. HTTP server at host www.cs.ucsb.edu waiting for TCP connection at port 80. “accepts” connection, notifying client

3. HTTP server receives request message, forms response message containing requested object, and sends message into its socket
Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.


6. Steps 1-5 repeated for each of 10 jpeg objects.
HTTP 1.0

Client

conn. req.

conn. est.

file request

data

Server

base object (connection 1)

object 1 (connection 2)

object 2 (connection 3)
Response time modeling

Definition of round trip time (RTT): time to send a small packet to travel from client to server and back.

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 total = 2RTT+transmit time
Non-persistent connections

**Non-persistent**
- http/1.0: server parses request, responds, closes TCP connection
- 2 RTTs to fetch each object
  - TCP connection
  - object request/transfer
- each transfer suffers from TCP’s initial slow sending rate
- many browsers open multiple parallel connections

**parallelism**
- Allows for multiple outstanding connections to request objects as soon as they are needed
- problem: still have 2 RTTs to fetch each object
Persistent connections

**Persistent**
- default for http/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- client sends requests for all referenced objects as soon as it receives base HTML
- fewer RTTs

**With Pipelining**
- Client issues request as soon as it encounters a reference
- Reduces RTT Delay
- Prevents idle time at server

**Without Pipelining**
- Client issues new request only after previous response received
  - Each object has one RTT
- Disadvantage: server waits between object requests with open connection
Non-persistence vs. persistence

- Non-persistence
- Persistence without pipelining
- Persistence with pipelining
HTTP request message

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

```
GET /courses/fall.html HTTP/1.1
Host: www.cs.ucsb.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
```

(extra carriage return, line feed)
HTTP request message: general format

<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
<th>cr</th>
<th>lf</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name</td>
<td>:</td>
<td>value</td>
<td>cr</td>
<td>lf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>header field name</td>
<td>:</td>
<td>value</td>
<td>cr</td>
<td>lf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cr</td>
<td>lf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

request line

header lines

Entity Body
HTTP response message

status line
(protocol
status code
status phrase)

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ...
Content-Length: 6821
Content-Type: text/html

data

...
HTTP response status codes

In first line in server->client response message.

A few sample codes:

200 OK
  – request succeeded, requested object later in this message

301 Moved Permanently
  – requested object moved, new location specified later in this message (Location:)

400 Bad Request
  – request message not understood by server

404 Not Found
  – requested document not found on this server

505 HTTP Version Not Supported
Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

   telnet cis.poly.edu 80
   Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu.
   Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

   GET /~ross/ HTTP/1.1
   Host: cis.poly.edu
   By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server
User-server state: cookies

Many major Web sites use cookies

Four components:
1) cookie header line in the HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user’s host and managed by user’s browser
4) back-end database at Web site

Example:
- Susan accesses Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID
Cookies: keeping “state” (cont.)

Client

Cookie file
- eBay: 8734

Server

Usual HTTP request msg

Usual HTTP response + 
Set-cookie: 1678

Server creates ID 1678 for user

Cookie-specific action

One week later:

Cookie file
- Amazon: 1678
- eBay: 8734

Usual HTTP request msg

Cookie: 1678

Usual HTTP response msg

Cookie-specific action
Cookies (continued)

What cookies can bring:
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites
Web caches (proxy server)

**Goal:** satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client
More about Web caching

- Cache acts as both client and server
- Typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?
- Reduce response time for client request.
- Reduce traffic on an institution’s access link.
- Internet dense with caches enables “poor” content providers to effectively deliver content (but so does P2P file sharing)

Install cache
- suppose hit rate is .4

Consequence
- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%
Conditional GET

- **Goal**: don’t send object if cache has up-to-date cached version
- **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
- **server**: response contains object if cached copy is not up-to-date:
  - HTTP/1.0 200 OK
  - `<data>`