CMPE 150: Introduction to Computer Networks

Katia Obraczka
Computer Engineering
UCSC Baskin Engineering

Lecture 6
Announcements/Reminders

• Homework 1 posted.
  – Due January 29th.
  – Submission via eCommons.
Important: Assignment Submission Guidelines

• Please visit class Web site for homework, labs, and guidelines for submission.
Important: Academic Integrity

• Must follow academic integrity rules.
• In particular:
  – MUST cite sources!!!!
Last Class…

- Application layer.
  - Client – server.
  - P2P.

- Application layer operation:
  - Process.
  - Application program (user interface).
  - Application-layer protocol.
• Application layer addresses.
  – Port numbers.
  – Well-known ports.
  – Sockets.
• Application layer protocol.
  – Relationship with transport layer.
  – Loss tolerance/intolerance.
  – Delay tolerance/intolerance.
• Web and HTTP.
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.someschool.edu/someDept/pic.gif

  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 overview (continued)

Uses TCP:
- 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

Aside

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.

Persistent HTTP
• Multiple objects can be sent over single TCP connection between client and server.
Nonpersistent HTTP

Suppose user enters URL www.someSchool.edu/someDepartment/home.index

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

1b. HTTP server at host www.someSchool.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 someDepartment/home.index

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

CMPE 150 Winter'13
Nonpersistent HTTP (cont.)

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

Definition of RTT: time for 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
Persistent HTTP

Nonpersistent 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 one RTT for all the referenced objects
HTTP request message

- two types of HTTP messages: request, response
- HTTP request message:
  - ASCII (human-readable format)
    - request line (GET, POST, HEAD commands)
      - GET /somedir/page.html HTTP/1.1
      - Host: www.someschool.edu
      - User-agent: Mozilla/4.0
      - Connection: close
      - Accept-language: fr
      - Carriage return, line feed indicates end of message

(extra carriage return, line feed)
Method types

HTTP/1.0
- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1
- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
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 ...
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 of HTTP response message
2) cookie header line in HTTP request message
3) cookie file kept on user’s host, managed by user’s browser
4) back-end database at Web site

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

Client:
- ebay 8734
- cookie file
- ebay 8734, amazon 1678

Server:
- usual http request msg
- usual http response
- Set-cookie: 1678

Amazon server creates ID 1678 for user
- create entry

one week later:
- usual http request msg
- cookie: 1678

usual http response msg
- cookie-specific action

ebay 8734, amazon 1678
- usual http request msg
- cookie: 1678

usual http response msg
- cookie-specific action

backend database
- access
Cookies (continued)

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

How to keep “state”:
- protocol endpoints: 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
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)
Caching example

Assumptions

• average object size = 1,000,000 bits
• avg. request rate from institution’s browsers to origin servers = 15/sec
• delay from institutional router to any origin server and back to router = 2 sec

Consequences

• utilization on LAN = 15%
• utilization on access link = 100%
• total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + milliseconds
possible solution
• increase bandwidth of access link to, say, 100 Mbps

consequence
• utilization on LAN = 15%
• utilization on access link = 15%
• Total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + msecs
• often a costly upgrade
Caching example (cont)

**possible solution: install cache**
- suppose hit rate is 0.4

**consequence**
- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = \(0.6 \times (2.01)\) secs + \(0.4\) milliseconds < 1.4 secs

CMPE 150 Winter’13
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

- object not modified

- object modified

CMPE 150 Winter’13
Domain Name System (DNS)
DNS: Domain Name System

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

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., www.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, 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
- b USC-ISI Marina del Rey, CA
- c Cogent, Herndon, VA (also LA)
- d U Maryland College Park, MD
- e NASA Mt View, CA
- f Internet Software C. Palo Alto, CA (and 36 other locations)
- g US DoD Vienna, VA
- h ARL Aberdeen, MD
- i Verisign, (21 locations)
- j Verisign, (21 locations)
- k RIPE London (also 16 other locations)
- l ICANN Los Angeles, CA
- m WIDE Tokyo (also Seoul, Paris, SF)
- n Autonomica, Stockholm (plus 28 other locations)
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, 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 host makes DNS query, query is sent to its local DNS server
  – 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 example

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