CS 6204
Cloud Environments

Godmar Back
Announcements

• Please sign up for Piazza CS 6204
• Choose first topic from list
Plan for today

• Review HTTP/HTML
• What you should know/remember about how web pages/applications work
• Informally, the following slides just serve as reference
The World Wide Web ca. 1993

Client

Server

http://www.vt.edu

HTTP

HTML

CS 6204 Spring 2014
Web Pages

• 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 1.0: RFC 1945
- HTTP 1.1: RFC 2068
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

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
Nonpersistent HTTP

http://courses.cs.vt.edu/~cs3214/index.html

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

2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object /~cs3214/index.html

1b. HTTP server at host www.cs.vt.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

(time contains text, references to n images)
Nonpersistent HTTP (cont.)

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds $n$ referenced image objects

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

4. HTTP server closes TCP connection.
Response time modeling

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
Persistent Connections

Idea: reuse same connection to retrieve multiple objects

Saves repeated connection establishment & teardown

\[ \text{total} = \text{RTT} + N \times (\text{RTT} + \text{transmit time}) \]
Pipelining

Idea: send next request before response for previous request arrives

Server should receive next request even before last response is sent

\[ \text{total} = 3 \times \text{RTT} + N \times \text{transmit time} \]
Persistent HTTP

Nonpersistent HTTP issues:
• requires 2 RTTs per object
• OS must work and allocate host resources for each TCP connection
• but 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 are sent over connection

Persistent without pipelining:
• client issues new request only when previous response has been received
• one RTT for each referenced object

Persistent with pipelining:
• default in HTTP/1.1
• client sends requests as soon as it encounters a referenced object
• as little as one RTT for all the referenced objects
Pipelining In Practice

• Current browsers disable pipelining by default, preferring multiple persistent connections instead
  - Reason: realizing latency advantages of pipelining is difficult, requires that server sends responses back to back with no bubble
    - Head of line blocking can occur
    - Buggy servers
• It’s easier for most server to commence processing of requests in parallel if they arrive on different connections
  - And responses can be sent as soon as they are ready
• See [Chan 2013] and [Nottingham 2011] for details
HTTP Request Message

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

```
GET /~cs3214/index.html HTTP/1.1
Host: www.cs.vt.edu
User-agent: Lynx/2.8.4rel.1
Connection: close
Accept-language: en, fr
```

Needed because multiple sites may be hosted on same host

 seulement si non-persistent

Carriage return, line feed

(extra carriage return, line feed)
HTTP Request Msg: general format

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

Entity Body
Grammar Formulation

HTTP-message = Request | Response
Request = rfc822-generic-message
Response = rfc822-generic-message
rfc822-generic-message = start-line
   *(message-header CRLF)
CRLF
   [ message-body ]
start-line = Request-Line | Status-Line
message-header = field-name ":" [ field-value ]
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
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
Uploading Form Input

**POST method:**
- `<form action="/form" method="post" ...
  <input name="username" type="text"> ....
- Input is uploaded to server in entity body

**GET method:**
- `<form action="/form" method="get" ...
  <input name="username" type="text"> ....
- Uses GET method
- Input is uploaded in URL field
- Needs RFC 2396 encoding, e.g. " " → %20

```
POST /form HTTP/1.1
Host: www.cs.vt.edu
Content-Length: 12
<CRLF>
username=Joe&...
```

```
GET /form?username=Joe&... HTTP/1.1
Host: www.cs.vt.edu
<CRLF>
```
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 ...

header lines

Only if non-persistent

data, e.g., requested HTML file
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.vt.edu 80`
   - Opens TCP connection to port 80 (default HTTP server port) at www.vt.edu. Anything typed is sent to port 80 at www.vt.edu

2. Type in a GET HTTP request:
   - `GET / HTTP/1.1`
   - `Host: www.vt.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!
Historical Note

- First version by Tim Berners-Lee
  - Aka HTTP/0.9

% telnet www.vt.edu 80
Trying 198.82.160.129...
Connected to www.vt.edu.
Escape character is '\].'
GET /

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Virginia Tech -- Home Page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
....
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

1. usual http request msg
2. usual http response + 
   Set-cookie: 1678

Server

3. server creates ID 1678 for user
4. cookie-specific action
5. cookie-specific action

One week later:

Client

1. usual http request msg
2. cookie: 1678
3. usual http response msg

Server

1. entry in backend database
2. access
3. access

Cookie file

amazon: 1678
ebay: 8734

Cookie file

amazon: 1678
ebay: 8734

Cookie file

amazon: 1678
ebay: 8734
Mashups

• Web applications that combine and mix data from different sources
Proxy-Based Mash-Up

Client Browser

Base Page

HTML+JavaScript

Proxy
http://www.lib.edu/proxy

XMLHttpRequest

Source A
http://www.lib.edu/sourceA

Source B
http://opac.lib.edu/sourceB

Source C
http://any.host.domain/sourceC

Base Server
http://www.lib.edu

CS 6204 Spring 2014
In modern web applications, HTTP is increasingly initiated from code
- Either directly, e.g., XMLHttpRequest
- Or indirectly, via SCRIPT/JSON

Example:
- `http://books.google.com/books?jscmd=viewapi&bibkeys=0060731338&callback=callThisFunction`

```javascript
callThisFunction({"0060731338":{"bib_key":"0060731338", "info_url": "http://books.google.com/books?id=HCInGwAACAAJ\x26source=gbs_ViewAPI", "preview_url": "http://books.google.com/books?id=HCInGwAACAAJ\x26source=gbs_ViewAPI", "preview":"noviev", "embeddable":false}});
```