World-Wide Web Protocols

CS 571
Fall 2006

© 2006 Kenneth L. Calvert
All rights reserved
World-Wide Web – The Information Universe

• World-Wide Web structure: hypertext
  - Nonlinear presentation of information

• Key component protocols:
  - A language for encoding hypertext objects
    - Including embedded links to other objects
      Hypertext Markup Language (HTML)
  - A mechanism for resolving links to HT objects
    Universal Resource Identifiers (URIs)
    Uniform Resource Identifiers (URIs)
    Uniform Resource Locators (URLs)
    Uniform Resource Names (URNs)
  - A protocol for transferring HT objects
    Hypertext Transfer Protocol (HTTP)
Uniform Resource Identifiers

• RFC 3986 (Jan '05):
  
  "Uniform Resource Identifier: Generic Syntax"
  
  other RFCs dealing with URIs/URLs/URNs: 1738, 2616, 2168, 3406...
  
  – URLs are a form of URI

• Syntax
  
  <scheme>: <scheme-specific part>
  
  – example URL schemes:
    
    http://<hostport>[/<path>/?<search>]
    
    <hostport> ::= <FQDN>[:<port>]
    
    ftp://<login>/<path>
    
    <login> ::= [ <user>[:<password>]@]<hostport>

URNs vs. URLs

• Uniform Resource Locators
  - Tell where something is
  - Must change if object moves
  - Advantage: can be resolved today using DNS

• Uniform Resource Names
  - Permanent, location-independent names
  - Require more complex resolution mechanism
  - Several attempts have been made to develop a URN system
  - Difficult to get agreement even on requirements
  - So far, nothing has succeeded
    • Observation: A universal naming system would be a boon to most network applications, but the standardization process has been often derailed by politics
HTTP

- HTTP 1.0 – RFC 1945, May 1996
  - After-the-fact documentation of what was already deployed in the Mosaic web browser

- HTTP 1.1 – RFC 2068, 2616, June 1999
  - Corrected many problems in original version
    - Added: persistent connections, authentication, pipelining, ...
  - Also documented many deployed features not in 1.0
HTTP Operation

- Client-Server protocol
- Messages = lines of ASCII text
  - Lines terminated by <crlf>
  - A la SMTP, FTP, POP, ...
- Client sends **Request** to server
  
  `<Request><URI><Version><crlf>
  [<Header><crlf>]`
  
  `<crlf>
  <body>`
- Request: GET, POST, HEAD, ...
- Version: HTTP/1.0
HTTP Operation, continued

• Server sends response:

  <status-line><crlf>
  [ <Header><crlf>]*
  <crlf>
  <body>

  - Status-line: <version> <response code> <phrase>
    • E.g. HTTP/1.1  304 Not Modified
  
  - Headers provide additional information or describe body (requested object) content
    
    Date   MIME-Version   Host (mandatory in 1.1)   User-Agent
    If-modified-since   Content-encoding   Content-length
    Expires   Last-modified
Basic HTTP Operation

Client

Open TCP connection (server port: 80)

GET ~calvert/classes/571/ HTTP/1.1
Allow: <list of content-types>
Host: protocols.netlab.uky.edu
If-modified-since: 2006-11-06 23:55:33
User-agent: Mozilla 2.5

HTTP/1.1 200 Success
Last-Modified: 2006-11-07 11:06:17
Expires: 2006-11-23 12:00:00
Content-Encoding: Base64
Content-Length: 8145
Content-type: text/html
[blank line]

[8145 bytes of content]
Problems with HTTP

• Originally, Server closed connection to delimit transferred object
  - One object transferred per connection (!)
  - Server ends up with TCP PCB in CLOSE-WAIT state for each object transferred

• URLs are a poor substitute for URNs
  - The Web is full of "dangling pointers"
  - URLs are becoming even more ephemeral
  - The good news: Google has made this problem irrelevant!
HTTP 1.0 Connection-per-Object (Typical) Operation

Connection:
13 packets total
3 data packets
HTTP 1.0 Connection-per-Object (Optimal) Operation

Client

- `connect( )`
- `send( )`
- `receive( )`
- `receive( )`
- `receive( )`
- `close( )`

Server

- `accept( )`
- `receive( )`
- `send( )`
- `close( )`

`7 packets total`
`3 data packets`

- `ack`
- `ack`
- `ack`
- `ack`
- `ack`
- `ack`
HTTP 1.1 Persistent Connections

- Default behavior in 1.1 and later:
  - Connection remains open unless "connection: close" header is sent by client or server
  - All object bodies are delimited using Content-length headers
    - Only possible when size is known in advance
    - Not possible for, e.g. live audio streaming
  - If content length is unknown, server sends connection: close before transmitting body, closes connection after

- Enables pipelining of requests and responses
Example Transaction (via Telnet)

GET ~calvert/classes/571/ HTTP/1.0 crlf
host: protocols.netlab.uky.edu crlf

HTTP/1.1 400 Bad Request crlf
Date: Fri, 17 Nov 2006 14:18:22 GMT crlf
Server: Apache/2.0.54 (Ubuntu) PHP/4.4.0-3ubuntu2 crlf
    mod_ssl/2.0.54 OpenSSL/0.9.7g crlf
Vary: accept-language,accept-charset crlf
    Accept-Ranges: bytes crlf
    Connection: close crlf
Content-Type: text/html; charset=iso-8859-1 crlf
    Content-Language: en crlf

[Body containing HTML explanation]