Network Applications: HTTP/1.0

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

9/26/2017
Admin

- Assignment 2 status
Recap: TCP Sockets

- TCP server socket demux by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
Recap: FTP

- A stateful protocol
  - state established by commands such as
    - USER/PASS, CWD, TYPE

- Multiple TCP connections
  - A control connection
  - Data connections
    - Two approaches: PORT vs PASV
    - GridFTP: concurrent data connections; block data transfer mode
Outline

- Recap
- HTTP
From Opaque Files to Web Pages

- **Web page:**
  - authored in HTML
  - addressed by a URL
    - URL has two components:
      - host name, port number, and
      - path name
  
  `http://www.cs.yale.edu:80/index.html`

- **Most Web pages consist of:**
  - base HTML page, and
  - several referenced objects

The Web pages are requested through HTTP: hypertext transfer protocol
HTTP is Still Evolving

HTTP/0.9  HTTP/1.0  RFC 1945  RFC 2068  HTTP/2  RFC 7540
HTTP 1.0 Message Flow

- Server waits for requests from clients
- Client initiates TCP connection (creates socket) to server, port 80
- Client sends request for a document
- Web server sends back the document
- TCP connection closed

- Client parses the document to find embedded objects (images)
  - repeat above for each image
HTTP 1.0 Message Flow (more detail)

Suppose user enters URL
www.cs.yale.edu/index.html


1b. server “accepts” connection, ack. client

2. http client sends http request message (containing URL) into TCP connection socket

3. http server receives request message, forms response message containing requested object (index.html), sends message into socket (the sending speed increases slowly, which is called slow-start)

0. http server at host www.cs.yale.edu waiting for TCP connection at port 80.
HTTP 1.0 Message Flow (cont.)


5. http client receives response message containing html file, parses html file, finds embedded image

6. Steps 1-5 repeated for each of the embedded images
Discussion

- How about we use FTP as HTTP?
HTTP1.0 Message Flow

- **HTTP1.0 servers are stateless servers**: each request is self-contained

  - FTP client
    - USER xxx
    - PASS xxx
    - CWD home
    - PORT clientip:cport
    - RETR index.html
    - Server initiates TCP data connection
      - server:20
      - clientip:cport
  
  - FTP server
  
  - HTTP client
    - GET /home/index.html
      - USER: xxx
      - PASS: xxx
      - Server sends file on same connection

  - HTTP server
HTTP Request Message: General Format

- ASCII (human-readable format)
Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

   telnet www.cs.yale.edu 80
   Opens TCP connection to port 80 (default http server port) at www.cs.yale.edu. Anything typed in sent to port 80 at www.cs.yale.edu

2. Type in a GET http request:

   GET /index.html HTTP/1.0
   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 the http server.
Trying out HTTP (client side) for yourself

- Try telnet GET on www.yale.edu
HTTP Request Message Example: GET

GET /somedir/page.html HTTP/1.0
Host: www.somechool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: en

(request line (GET, POST, HEAD, PUT, DELETE, TRACE ... commands))
(header lines)

Virtual host multiplexing
Connection management
Content negotiation

(extra carriage return, line feed)

Carriage return, line feed indicates end of message
HTTP Response Message

status line (protocol status code status phrase)

HTTP/1.0 200 OK
Date: Wed, 23 Jan 2008 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 ...

data, e.g., requested html file

header lines
HTTP Response Status Codes

In the first line of the 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 Use Chrome to visit Course Page
Design Exercise

- Workflow of an HTTP server processing a GET request that maps to a file:

  GET /somedir/page.html HTTP/1.0
  Host: www.somechool.edu
Basic HTTP Server Workflow

1. Create ServerSocket(6789)
2. connSocket = accept()
3. Read request from connSocket
4. Map URL to file
5. Read from file/write to connSocket
6. close connSocket

TCP socket space:
- state: listening
- address: {*.6789, *.*}
- completed connection queue: sendbuf: recvbuf:

TCP socket space:
- state: established
- address: {128.36.232.5:6789, 198.69.10.1500}
- sendbuf: recvbuf:

TCP socket space:
- state: listening
- address: {.25, *.*}
- completed connection queue: sendbuf: recvbuf:
Example Code

- See BasicWebServer.java

- Try using telnet and real browser, and fetch
  - file1.html
  - index.html

  what difference in behavior?
Static -> Dynamic Content

Create ServerSocket(6789)

connSocket = accept()

read request from connSocket

Map URL to file

Read from file/write to connSocket

close connSocket

It does not have to be a static file
Dynamic Content Pages

- There are multiple approaches to make dynamic web pages:
  - Embed code into pages (server side include)
    - http server includes an interpreter for the type of pages
  - Invoke external programs (http server is agnostic to the external program execution)

http://www.cs.yale.edu/index.shtml
http://www.cs.yale.edu/cgi-bin/ureserve.pl
http://www.google.com/search?q=Yale&sourceid=chrome
Example SSI

- See programming/examples-java-socket/BasicWebServer/ssi/index.shtml, header.shtml, ...
Example SSI

- See programming/examples-java-socket/BasicWebServer/ssi/index.shtml, header.shtml, ...

- To enable ssi, need configuration to tell the web server (see conf/apache-htaccess)
  - https://httpd.apache.org/docs/2.2/howto/htaccess.html (Server Side Includes example)
CGI: Invoking External Programs

- Two issues
  - Input: Pass HTTP request parameters to the external program
  - Output: Redirect external program output to socket
Example: Typical CGI Implementation

- Starts the executable as a child process
  - Passes HTTP request as environment variables
    - [http://httpd.apache.org/docs/2.2/env.html](http://httpd.apache.org/docs/2.2/env.html)
  - Redirects input/output of the child process to the socket
Example: CGI

Example:

- GET /search?q=Yale&sourceid=chrome HTTP/1.0

- setup environment variables, in particular
  $QUERY_STRING=q=Yale&sourceid=chrome

- start search and redirect its input/output

https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html
Example

http://zoo.cs.yale.edu/classes/cs433/cs433-2016-spring/programming/examples-java-socket/BasicWebServer/cgi/price.cgi?appl

#!/usr/bin/perl -w

$company = $ENV{'QUERY_STRING'};
print "Content-Type: text/html\n"
print "\n"

print "<html>"
print "<h1>Hello! The price is ";

if ($company =~ /appl/) {
   my $var_rand = rand();
   print 450 + 10 * $var_rand;
} else {
   print "150";
}

print "</h1>"
print "</html>";}
Client Using Dynamic Pages

- See ajax.html and wireshark for client code example

  http://zoo.cs.yale.edu/classes/cs433/cs433-2016-spring/programming/examples-java-socket/BasicWebServer/cgi/ajax.html
Discussions

- What features are missing in HTTP that we have covered so far?
HTTP: POST

- If an HTML page contains forms or parameter too large, they are sent using POST and encoded in message body.

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

Entity Body
HTTP: POST Example

POST /path/script.cgi HTTP/1.0
User-Agent: MyAgent
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

item1=A&item2=B

Example using nc:
programming/examples-java-socket/BasicWebServer/nc/
Stateful User-server Interaction: Cookies

Goal: no explicit application level session

- Server sends “cookie” to client in response msg
  Set-cookie: 1678453

- Client presents cookie in later requests
  Cookie: 1678453

- Server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices
Authentication of Client Request

Authentication goal: control access to server documents

- **stateless**: client must present authorization in each request
- **authorization**: typically name, password
  - **Authorization**: header line in request
  - if no authorization presented, server refuses access, sends
    - **WWW-authenticate**: header line in response

**Browser caches name & password so that user does not have to repeatedly enter it.**
Example: Amazon S3

- Amazon S3 API
HTTP as the Thin Waist

HTTP

WebMail  WebApp...

TCP  UDP

Ethernet  Wireless Cable/DSL
Protocol Flow of Basic HTTP/1.0

- >= 2 RTTs per object:
  - TCP handshake --- 1 RTT
  - Client request and server responds --- at least 1 RTT (if object can be contained in one packet)
Discussion: How to Speedup HTTP/1.0