

HTTP Part 5

HTTP Under The Hood

Write a little “echo” server that listens for HTTP requests on port 8181, and then just echoes it back, so we can see the details for the browser request...

Echo Server Code

```
#!/usr/bin/perl
use Socket;
use FileHandle;

use strict 'vars';

my $server_port;
$server_port = 8181;

my $EOL = "\015\012";

## Setup our SERVER socket
CreateServerSocket(SERVER, $server_port);

listen(SERVER, SOMAXCONN);

## Wait for incoming client connections
my $client_addr;
while ($client_addr = accept(CLIENT, SERVER)) {           ## accept() blocks

    autoflush CLIENT, 1;

    ## Read the client request ... until blank line
    my($line, $request);
    $request = "";
    while (1) {
        $line = <CLIENT>;
        $line =~ s/\015|\012//g;
        if ($line eq "") { last; }

        $request = $request . $line . "\n"; ## accumulate request lines
    }

    ## Print what we got from the client locally
    print "Got from the client:\n$request\n";

    ## Spit back a minimal HTTP response, echoing the client response back to them...
    ## We don't put in content-length, but the client seems to deal ok
    ## To generate proper HTML, we would need to worry about characters
    ## like < and & that we dump out
    print CLIENT "HTTP/1.0 200 OK$EOL";
    print CLIENT "Content-type: text/html$EOL";
    print CLIENT "Server: home grown$EOL";
    print CLIENT "Connection: close$EOL";
    print CLIENT "$EOL";
}
```

```

print CLIENT "<html>Here's what we got from the client:\n<pre>\n";
print CLIENT $request;
print CLIENT "</pre></html>\n";

close(CLIENT);
}

```

Mozilla Client Output

Run echo server ... click on link in course page

```

<li>A <a href="http://elaine1.stanford.edu:8181/dir/x.html?pi=3#fragment">echo link</a>
(only works when the echo server is running)

```

Echo server output:

```

GET /dir/x.html?pi=3 HTTP/1.1
Host: elaine1.stanford.edu:8181
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4a) Gecko/20030
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x
mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.stanford.edu/class/cs193i/

```

Internet Explorer Output

```

GET /dir/x.html HTTP/1.1
Host: elaine1.stanford.edu:8181
Accept: */*
Accept-Language: en
Connection: Keep-Alive
If-Modified-Since: Mon, 28 Apr 2003 19:17:11 GMT
User-Agent: Mozilla/4.0 (compatible; MSIE 5.1b1; Mac_PowerPC)
UA-OS: MacOS
UA-CPU: PPC
Extension: Security/Remote-Passphrase

```

User-Agent

The browser version and OS of the client

"Mozilla" was the user-agent of the first Netscape browser. As a result, most browsers claim to be "mozilla" since there are (ancient) sites that only work with that user-agent. The real user-agent is usually mentioned in parenthesis after the fake user-agent. This is an example of standards gone amok.

The user-agent field can be used to special-case the HTML generated to work around browser bugs – generally a bad idea. Better to write out correct HTML, and let incorrect browsers fail.

Referer

The page that contained the link that lead to this request

Allows you to see who is pointing to you

Web servers can save the referer for each request in the logs, so a site can get a sense of where they are getting hits from.

Note: “referrer” → referer is misspelled in the spec, but we're stuck with it forever – network effect inertia

Accept

MIME types the client can handle – image/png is there, in this case, since my browser has a plug-in that allows it to handle png

These are just suggestions – the server can send whatever MIME type, and the client will do the best it can.

Some browsers just send */* and sort it out on the client side when the data arrives.

Accept-Encoding

The client indicates that they can deal with the gzip encoding – HTML compresses really well, so sending the HTML gzip compressed can be a 50% bandwidth savings, and gives better performance as well.

The server will indicate the encoding used in the response header.

If-Modified-Since request

Use this field in the request, specifying the last mod date of the cached copy

The Server can respond with “304 not modified”, if the content has not changed. The client can just use the cached copy they already have.

Saves bandwidth, but still requires a little 2*latency interaction

Proprietary Headers

See the Internet Explorer headers above.

UA-OS is a Microsoft-only header, presumably understood when a Microsoft client talks to a Microsoft web server. Such headers is not regarded as good citizenship in the profoundly standards-oriented Internet engineering community.

HTTP 1.1 Virtual Hosts

Suppose you want to host 20 web sites on one machine

Hosting a web site is not very demanding, especially for plain web pages. You typically run out of networking bandwidth before you run out of CPU, memory, etc. Therefore, using one machine for many sites is common.

Want http://foo.com/, http://bar.com/, all on the one machine – "virtual hosts"

DNS: multiple names, foo.com, bar.com, www.foo.com, ... can all map to one IP address.

Problem: when a server gets a request "GET / HTTP/1.0" – which virtual host is it for?

Solution: with HTTP 1.1, the client includes a **host: www.foo.com** header in the request

Alternative: the other way to do this, is "multihoming" – one machine has multiple IP addresses. Then the server can tell which virtual server is intended by which IP addr the request is addressed to. However, this method is deprecated, since it can use up lots of IP addresses.

Virtual Host Example

```
elaine0:~> telnet cslibrary.stanford.edu 80
Trying 171.64.64.168...
Connected to cslibrary.Stanford.EDU (171.64.64.168).
Escape character is '^]'.
GET / HTTP/1.1

HTTP/1.1 400 Bad Request
Date: Wed, 24 Apr 2002 18:50:04 GMT
Server: Apache/1.3.23 (Darwin)
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

176
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
client sent HTTP/1.1 request without hostname (see RFC2616 section 14.23): /<P>
<HR>
<ADDRESS>Apache/1.3.23 Server at cslibrary.stanford.edu Port 80</ADDRESS>
</BODY></HTML>
```

0

Connection closed by foreign host.

```
elaine0:~> telnet cslibrary.stanford.edu 80
Trying 171.64.64.168...
Connected to cslibrary.Stanford.EDU (171.64.64.168).
Escape character is '^]'.
GET / HTTP/1.1
host:cslibrary.stanford.edu

HTTP/1.1 200 OK
Date: Wed, 24 Apr 2002 18:50:15 GMT
Server: Apache/1.3.23 (Darwin)
Last-Modified: Mon, 22 Apr 2002 18:56:54 GMT
ETag: "10cd0-1b34-3cc45cf6"
Accept-Ranges: bytes
Content-Length: 6964
Content-Type: text/html

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
...
```

HTTP 1.1 Persistent Connections

Problem: typical web page with one body of HTML and 20 little GIF buttons requires 21 separate connections.

TCP and HTTP are not efficient for many little connections – must bring up the TCP connection many times (cost each time), and TCP is most efficient for longer running connections.

Persistent Connection – Keep-Alive

Add one of the following to the request

Connection: keep-alive – client requests to keep connection open

Connection: close – alternately, client requests that the server close connection after response (like HTTP 1.0)

With HTTP 1.1, keep-alive is the default

Server indicates if it is closing the connection with

Connection: close – in the response header (note that the same syntax is used in both the request and response headers)

Persistent Connection / Pipelining

Client can send more requests (GET ...) on the same connection – avoid the connection setup cost

“Pipelining” – client can send multiple requests, even before receiving responses. This can be a great speedup – avoids the latency*2 cost of a typical request/response system by not waiting for the response.

We say that multiple requests are “in flight” at one time.

Chunked Transfer Encoding

Problem: content-length header is costly for server – have to wait before sending data

Solution: omit the content-length header, send data back in chunks, each chunk is prefixed by its length in hex.

Server puts **Transfer-encoding: chunked** in the header

End is marked with a chunk length 0

With HTTP1.1, a client must be capable of getting the response back chunked. For this reason, it's easier to use HTTP/1.0 for simple web client software.

HTTP 1.1 Pipelined Example

Here I have hacked up the web client to request /0.html, /1.html, .. in very quick succession.

Only 0.html actually exists, the others generate 404.s

Demonstrates multiple, pipelined requests on a keep-alive connection

Notice that the 404 responses come back chunked (when to chunk and when not is up to the server)

Pipelined Client Code

```
## ask for /0.html /1.html ... without waiting for response
## -- demonstrates keep-alive / pipelining
my($i, $req, $count);
```

```

$count = 5;
for ($i=0; $i<$count; $i++) {
    $req = "GET /$i.html HTTP/1.1$EOL";
    $req .= "host:$host$EOL";

    ## Ask for keep-alive, except the last time
    ## (in reality, we should check the server response
    ## to see if it is actually keeping the connection open)
    my($mode);
    if ($i==$count-1) { $mode = "close"; } else { $mode="keep-alive"; }
    $req .= "connection:$mode$EOL";

    $req .= "$EOL";

    print SOCK $req;

    print $req;
}

## Read back and print all the responses

my($line);

while ($line = <SOCK>) {
    $line =~ s/\015|\012//g; ## strip off line endings
    print $line, "\n"; ## print with our local line ending
}

close(SOCK);

exit(0);

```

Pipelined Example

Demonstrates persistent connection, pipelined requests
 Also demonstrates chunked encoding

```

> webclient2.pl cse.stanford.edu
GET /0.html HTTP/1.1
host:cse.stanford.edu
connection:keep-alive

GET /1.html HTTP/1.1
host:cse.stanford.edu
connection:keep-alive

GET /2.html HTTP/1.1
host:cse.stanford.edu
connection:keep-alive

GET /3.html HTTP/1.1
host:cse.stanford.edu
connection:keep-alive

GET /4.html HTTP/1.1

```

host:cse.stanford.edu
connection:close

HTTP/1.1 200 OK
Date: Mon, 28 Apr 2003 19:25:25 GMT
Server: Apache/1.3.26 (Darwin)
Last-Modified: Fri, 26 Apr 2002 18:40:53 GMT
ETag: "115bb-dc-3cc99f35"
Accept-Ranges: bytes
Content-Length: 220
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
  <title>Test</title>
</head>
<body bgcolor="#FFFFFF">

<h1>
Test</h1>
<p>Just a little test doc.
<p>Just a little test doc.

</body>
</html>
```

HTTP/1.1 404 Not Found
Date: Mon, 28 Apr 2003 19:25:25 GMT
Server: Apache/1.3.26 (Darwin)
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

```
111
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /1.html was not found on this server.<P>
<HR>
<ADDRESS>Apache/1.3.26 Server at cse.stanford.edu Port 80</ADDRESS>
</BODY></HTML>
```

0

HTTP/1.1 404 Not Found
Date: Mon, 28 Apr 2003 19:25:25 GMT
Server: Apache/1.3.26 (Darwin)
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

111

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /2.html was not found on this server.<P>
<HR>
<ADDRESS>Apache/1.3.26 Server at cse.stanford.edu Port 80</ADDRESS>
</BODY></HTML>
```

0

<and so on>